The AlphaServer 8000 Series: High-end Server Platform Development

by David M. Fenwick, Denis J. Foley, William B. Gist,
Stephen R. VanDoren, and Daniel Wissell

ABSTRACT

The AlphaServer 8400 and the AlphaServer 8200 are Digital's newest high-end server products. Both servers are based on the 300-MHz Alpha 21164 microprocessor and on the AlphaServer 8000-series platform architecture. The AlphaServer 8000 platform development team set aggressive system data bandwidth and memory read latency targets in order to achieve high-performance goals. The low-latency criterion was factored into design decisions made at each of the seven layers of platform development. The combination of industry-leading microprocessor technology and a system platform focused on low latency has resulted in a 12-processor server implementation--the AlphaServer 8400--capable of supercomputer levels of performance.

INTRODUCTION

The new AlphaServer 8000 platform is the foundation for a series of open, Alpha microprocessor-based, high-end server products, beginning with the AlphaServer 8400 and AlphaServer 8200 systems and continuing through at least three generations of products. When combined with the power of the industry-leading 300-megahertz (MHz) Alpha 21164 microprocessor,[1] this innovative server platform offers the outstanding performance and price/performance required in technical and commercial markets. In uniprocessor performance benchmark tests, the AlphaServer 8400/8200 SPECfp92 rating of 512 instructions per second is 1.4 times the rating of its nearest competitor, the SGI Power Challenge XL product. In multiprocessor benchmark tests of systems with 1 to 12 processors, the AlphaServer 8400 system posts SPECrate levels greater than 3.5 times those of the HP9000-800 T500 system. In the category of cost for performance, NAS Parallel Class B SP benchmarks show that the AlphaServer 8400 system provides 1.7 times the performance per million dollars of the SGI Power Challenge XL system.[2] Perhaps most impressive is the AlphaServer 8400 performance on the Linpack NxN benchmark.[3] With a Linpack NxN result of 5 billion floating-point operations (GFLOPS), a 12-processor AlphaServer 8400 achieves the performance levels of supercomputers such as the NEC SX-3/22 system and the massively parallel Thinking Machines CM-200 system.

There are two keys to the remarkable performance of the AlphaServer 8400 and AlphaServer 8200 systems: the Alpha 21164 microprocessor chip and the AlphaServer 8000 platform architecture. This paper is concerned with the second of these, the AlphaServer 8000 platform architecture. Specifically, it

discusses the principal design issues encountered and resolved in the pursuit of the aggressive performance and product goals for the AlphaServer 8000 series. The paper concludes with an evaluation of the success of this platform development based on the performance results of the first AlphaServer 8000-series products, the AlphaServer 8400 and AlphaServer 8200 systems.

ALPHASERVER 8400 AND ALPHASERVER 8200 PRODUCT GOALS

The AlphaServer 8000 platform technical requirements were derived from a set of product goals. This set comprised minimum performance goals and a number of specific configuration and expandability requirements developed from Digital's server marketing profiles. The motivations that shaped the list of goals below were many. Support for legacy I/O subsystems and DEC 7000/10000 AXP compatibility, for example, was motivated by the need to provide Digital's customer installed base with a cost-effective upgrade path from 7000-series hardware to AlphaServer 8000-series hardware. The goals for low-cost I/O subsystem, peripheral component interconnect (PCI), and EISA support and for rackmount cabinet support were included to take advantage of emerging industry standards and open systems and their markets. The processor, I/O, and memory capacity goals were driven simply by the competitive state of the server market.

- Provide industry-leading enterprise and open-office server performance.
- Provide twice the overall performance of the DEC 7000/10000 AXP server products.
- Support up to 12 Alpha 21164 processors.
- Support at least 14 gigabytes (GB) of main memory.
- Support multiple I/O port controllers--up to 144 I/O slots.
- Provide a low-cost, integrated I/O subsystem.
- Support new, industry-standard PCI and EISA I/O subsystems.
- Continue to support legacy I/O subsystems, such as XMI and Futurebus+.
- Make centerplane hardware compatible with an industry-standard rackmount cabinet.
- Make centerplane hardware mechanically compatible with the DEC 7000/10000 AXP-series cabinet.

PERFORMANCE GOALS AND MEMORY READ LATENCY ISSUES

Although "providing industry-leading performance" and "doubling the performance" of an existing industry-leading server present excellent goals for the development of a new server, it is difficult to design to such nebulous goals. To quantify the actual technical requirements for the new AlphaServer 8000 platform, the design team utilized a performance study of the DEC 7000/10000 AXP systems and conducted a detailed analysis of symmetric multiprocessing (SMP) system operation. As a result of the analyses, aggressive system data bandwidth and memory read latency goals were established, as well as a design philosophy that emphasized low memory read latency in all aspects of the platform development. This section addresses the read latency issues and goals considered by the design team. The 8000 platform development is the focus of the section following.

Read latency is the time it takes a microprocessor to read a piece of data into a register in response to a load instruction. If the data to be read is found in a processor's cache, the read latency will typically be very small. If, however, the data to be read resides in a computer system's main memory, the read latency is typically much larger. In either case, a processor may have to wait the duration of the read latency to make further progress. The smaller the read latency, the less time a processor waits for data and thus the better the processor performs.

Cache memories are typically used to minimize read latency. When caches do not work well, either because data sets are larger than the cache size or as the result of non-locality of reference, a computer system's processor-memory interconnect contributes significantly to the average read latency seen by a processor. The system characteristics that help determine a processor's average read latency are the system's minimum memory read latency and data bandwidth.

A system's minimum memory read latency is the time required for a processor to fetch data from a system's main memory, unencumbered by system traffic from other processors and I/O ports. As processors and I/O ports are added to a system, their competition for memory and interconnect resources tends to degrade the system's memory read latency from the minimum memory read latency baseline. A system's data bandwidth, i.e., the amount of data that a system can transfer between main memory and its processors and I/O ports in a given period of time, will determine the extent to which processors and I/O ports will degrade each other's read latency. As data bandwidth increases, so too does a system's ability to support concurrent data references from various processors and I/O ports. This increased bandwidth and concurrent data referencing serve to reduce competition for resources and, as a result, to reduce memory read latency. Thus we can conclude that the more available data bandwidth in a system, the closer the memory read latency will be to the minimum. This conclusion is supported by the results of a queuing

model used to support the AlphaServer 8000 platform development. This queuing model, originally implemented to evaluate bus arbitration schemes, outputs the average read latencies experienced by each processor in a system as the number of processors and the number of memory resources are varied. It is important to note that in this model memory resources, or banks, determine the amount of system bandwidth: the more memory banks, the more bandwidth. It is also important to note that the minimum read latency in this model is 168 nanoseconds (ns). The results of the model are shown in Table 1. These results clearly show that latency degrades as the number of system processors is increased and that latency improves as the system's bandwidth--number of memory banks--is increased.

Table 1 Average Read Latency as a Function of the Number of Processors and Bandwidth (Number of Memory Banks)

Number of Processors	{ Average Read Latency (Nanoseconds) }			
	2 Memory Banks	4 Memory Banks	6 Memory Banks	8 Memory Banks
1	185	179	177	176
2	224	200	193	190
4	358	253	230	220
8	928	439	338	299

Many technical market benchmarks, such as the Linpack benchmarks and the McCalpin Streams benchmark, stress a computer system's data bandwidth capability. The regularity of data reference patterns in these benchmarks allows a high degree of data prefetching. Consequently, data can be streamed into a processor from main memory so that a piece of data has an unnaturally high probability of being resident in the processor's cache when it is needed for some calculation. Ironically, this amounts to using smart software to minimize read latency. By reading a piece of data into a processor's cache before it is actually needed, the software presents the processor with a small cache read latency instead of a long memory latency when the data is needed. Thus the streaming techniques applied in these benchmarks allow processors in high-bandwidth systems to stall for a full memory read latency period only when starting up a stream of data. Therefore memory latency can be amortized over lengthy high-bandwidth data streams, minimizing its significance. It is important to note, however, that although bandwidth is the system attribute that dominates performance in these workloads, it dominates performance through its effect on read latency.

Commercial workloads like the Transaction Processing Performance Council's benchmark suite, on the other hand, typically have more

complex data patterns that frequently defy attempts to prefetch data. When some of these codes parse data structures, in fact, the address of each data access may depend on the results of the last data access. In any case where a processor must wait for memory read data to make progress, a system's memory read latency will determine the period of time that the processor will be stalled. Such stall periods directly affect the performance of computer systems on commercial workloads. These assertions supported by a study on the performance of commercial workloads on Digital's Alpha 20164-based DEC 7000/1000 AXP server.[4] It is important to note here that the latency ills flagged in this study cannot be cured with raw system data bandwidth or software-enhanced latency reduction. Low memory latency alone can address the needs of these workloads.

Comparable industry systems from IBM[5] and Hewlett-Packard (HP)[6] do not stress low memory latency system development in their respective RISC System/6000 SMP or Hawks (PA-8000-based) SMP systems. In fact, neither directly acknowledges memory latency as a significant system attribute. This mind set is reflected in the results: Based on IBM's documentation, we estimate the RISC System/6000 SMP's minimum main memory read latency to be in the neighborhood of 600 to 800 ns.

IBM and HP do emphasize system bandwidth in their designs. HP provides a 960-MB-per-second (MB/s) "runway" processor-memory bus in its Hawks system. The actual data bandwidth of this bus is slightly less than the quoted 960 MB/s, since the bus is shared between address and data traffic. IBM, on the other hand, goes to the extent of applying a data crossbar switch in conjunction with a serial address bus to reach an 800-MB/s rate in its RISC System/6000 SMP system. The actual attainable data bandwidth in IBM's system is determined by the bandwidth of its address bus.

In the past, Digital's systems have shown much the same balance of bandwidth and latency as have their competitors. The DEC 7000/10000 AXP system has a minimum main memory read latency of 560 ns and a system data bandwidth of 640 MB/s. The AlphaServer 8000 platform, however, was developed with a marked emphasis on low memory read latency. This emphasis can be seen through nearly all phases of system development, including the system topology, clocking strategy, and protocol. This latency- oriented mindset is reflected in the results: The AlphaServer 8000 platform boasts minimum memory read latencies of 200 ns. The AlphaServer 8400 and 8200 systems feature a minimum memory read latency of 260 ns. To back up these latencies, the AlphaServer 8000 platform supports a tremendous 2,100 MB/s of data bandwidth. The AlphaServer 8400 and 8200 systems, although not capable of providing the full 2,100 MB/s, still provide 1,600 MB/s of bandwidth. This gives the systems less than half the memory latency of comparable industry systems while providing nearly twice the bandwidth. Furthermore, these attributes improve upon the DEC 7000/10000 AXP attributes by factors of 2 to 3. Although difficult to determine exactly how these attributes would translate into overall system

performance, they were accepted as sufficient to meet the AlphaServer 8000 platform performance goals. A comparison of the maximum DEC 7000/10000 AXP SPECrates of approximately 25,000 integer and 40,000 floating point with the maximum AlphaServer 8400 SPECrates of 91,580 integer and 14,0571 floating point indicates that these attributes were sound choices.

ALPHASERVER 8000 PLATFORM DEVELOPMENT

Referring to the AlphaServer 8000 platform as a "foundation" for a series of server products does not give a clear picture of what constitutes a server platform. The AlphaServer 8000 platform has both physical and architectural components. The physical component consists of the basic physical structure from which 8000-series server products are built. This includes power systems, thermal management systems, system enclosures, and a centerplane card cage that implements the interconnect between processor, memory and I/O port modules. The processor, memory, and I/O modules are printed circuit board (PCB) assemblies that can be implemented with varying combinations of processor, dynamic random-access memory (DRAM), and application-specific integrated circuit (ASIC) components. The assemblies are inserted into the platform centerplane card cage in varying configurations and in varying enclosures to create the various 8000-series products. The AlphaServer 8200 system, for example, comprises up to six Alpha 21164-based TLEP processor modules, TMEM memory modules, or ITIOP and TIOP I/O port modules in an industry-standard rack-mount system. The AlphaServer 8400 system comprises up to nine TLEP processor modules, TMEM memory modules, or ITIOP and TIOP I/O port modules in a DEC 7000 AXP-style data center cabinet.

The architectural component of the AlphaServer 8000 platform consists primarily of a collection of technological, topological, and protocol standards. This collection includes the processor-memory interconnect strategy, the bus interface technology, the clock technology and methodology, and the signaling protocols. For example, the TLEP, TMEM, and TIOP modules all implement bus interfaces in the same integrated circuit (IC) packages with the same silicon technology and drive their common interconnect bus with the same standard bus driver cell. Furthermore, all these modules apply nearly identical clocking circuits and communicate by means of a common bus protocol. The ephemeral architectural standards that constitute the "platform" specify exact physical requirements for designing the AlphaServer processor-memory-I/O port interconnect and the various modules that will populate it. It is important to note that the key to AlphaServer 8000 performance is found in these standards. As we explore the design decisions and trade-offs that shaped the AlphaServer 8000 platform, it is this collection of architectural standards that we actually probe.

Throughout this analysis of the AlphaServer 8000 architecture,

two themes frequently recur: low memory latency and practical engineering. As discussed in the context of the AlphaServer 8000 goals, low memory read latency was identified as the key to system performance. As such, low latency was factored into nearly every system design decision. Design decisions in general can be thought of as being resolved in one of two ways: by emphasizing Digital's superior silicon technology or by effecting architectural finesse. Use of superior technology is self explanatory; it involves pushing leading-edge technology to simply overwhelm and eliminate a design issue. Architectural finesse, on the other hand, typically involves a shift in operating mode or configuration that allows a problem to be avoided altogether. Practical engineering is the art of finding a balance between leading-edge technology and architectural finesse that produces the best product.

LAYERED PLATFORM DEVELOPMENT

Platform development typically involves a simple three-layer process: (1) determine a basic system topology, (2) establish the electrical means by which various computer components will transmit signals across the system topology, and (3) apply a signaling protocol to the electrical transmissions to give them meaning and to allow the computer components to communicate. System topology determines how processor, memory, and I/O components of a computer system are interconnected. Computer interconnects may involve simple buses, multiplexed buses, switches, and multitiered buses. The electrical means for transmitting signals across a computer interconnect may involve bus driver technology, switch technology, and clock technology. Signaling protocols apply names to system interconnect signals and define cycles in which the signals have valid values. This naming and definition allows each computer component to understand the transmissions of other components.

As the AlphaServer 8000 platform development progressed, this simple three-layer platform development model was found to be insufficient. Efforts to achieve the low-latency performance goal and the simple product goals uncovered unexpected design issues. The resolution of these design issues led to the creation of a more robust seven-layer platform development model. When certain multi-driver bus signals threatened the cycle time of the AlphaServer 8000 system bus, for example, the system's latency goals were threatened as well. The practical solution to this multi-driver signal problem was the creation of specific signaling conventions for problematic classes of signals. This innovation led to the birth of the Signaling Layer of the development model. Similarly, when the integration of PCI I/O into the system was found to conflict with primary protocol elements that were key to low latency processor-memory communication, the concept of a "superset protocol" was created. This led to the creation of the Superset Protocol Layer of the development model. The seven-layer platform development model is

contrasted with the simple three-layer development model in Figure 1.

The analysis of the AlphaServer 8000 platform design presented here traces the key system design decisions through each of the seven layers of the development process. Each layer will be described in greater detail as this analysis proceeds.

[Figure 1 (Comparison of Conventional Three-Layer Model with Seven-Layer Platform Development Model) is not available in ASCII format.]

Topological Layer

Server-class computers typically comprise processor, memory, and I/O port components. These components are usually found in the form of PCB modules. A computer system's topology defines how these computer components are interconnected. Computer topologies are many and varied. The IBM RISC System/6000 SMP, for example, links its modules by means of an address bus and a data switch. Its memory modules are grouped into a single memory subsystem with one connection to the address bus and one connection to the data switch. The HP Hawks SMP system, by comparison, links its modules by means of a single bus onto which address and data are multiplexed. The Hawks system also groups its memory into a single memory subsystem with one connection to the multiplexed bus.[7] Digital's DEC 7000/10000 AXP also uses a single multiplexed address and data bus. Unlike the IBM and HP systems, the DEC 7000/10000 AXP system allows its memory to be distributed, with multiple connections to its multiplexed bus.

None of the IBM, HP, or prior Digital systems meet the latency goals of the AlphaServer 8000 platform. Exactly how much system topology contributes to these systems' latencies is unclear. A multiplexed address and data bus certainly creates a system bottleneck and can contribute to latency. Likewise, unified memory subsystems can often have associated overhead that can translate into latency. In addition to performance issues, topologies such as the IBM switch-based system have significant cost issues. If, for example, a customer were to purchase a sparsely configured--two processors perhaps--IBM system, such a customer would be required to pay for the switch support for up to eight processors. This creates a high system entry cost and a potentially lower incremental cost as functionality is added to the system. In a simple bused system, a customer pays only for what is needed to support the specific functionality required. This creates a more manageable entry cost and a smooth, if slightly steeper, incremental cost. From Digital's marketing perspective, this makes a bused system preferable, provided it can satisfy bandwidth and latency requirements.

Uniprocessor computer topologies, an example of which is shown in Figure 2, typically exhibit the lowest memory read latencies of

any computer class. As such, this simple uniprocessor topology was chosen as the basis from which to develop the AlphaServer 8000 platform topology. In the uniprocessor model, processor chips communicate with DRAM arrays through separate address and data paths. These paths include address and data interfaces and buses. The AlphaServer 8000 topology was created by adding a second set of interfaces between the address and data buses and the DRAM array, and connecting additional microprocessors, memory arrays, and I/O ports to the buses by means of similar interfaces. The resultant topology is shown in Figure 3. This topology features separate address and data buses. These buses together are referred to as the AlphaServer 8000 system bus.

[Figure 2 (Simple Uniprocessor System Topology) and Figure 3 (AlphaServer 8000 Multiprocessor System Topology) are not available in ASCII format.]

The topology presented in Figure 3 is an abstract. To flesh out this abstract and measure it against specific system goals, signal counts, cycle times, and bus connection (slot) counts must be added. It is in this effort that practical engineering must be applied. To achieve the system's bandwidth goal, for example, the data bus could be implemented as a wide bus with a high clock frequency, or it could be replaced with a switch-based data interconnect, like that of the IBM RISC System/6000 SMP. The high-frequency bus presents a significant technological challenge in terms of drivers and clocking. This challenge grows as the number of bus slots grows. The growth of the technological challenge is a significant issue given the system's configuration goals. The switch interconnect, on the other hand, avoids the technological challenges by providing more data paths at lower clock frequencies. The lower clock frequencies, however, can translate directly into additional latency. Given the emphasis placed on memory latency and the advantages associated with simple bused systems, the practical design choice was to adopt a wide, high-frequency data interconnect. The resultant AlphaServer 8000 system bus features 9 slots, an address bus that supports a 40-bit address space, and a 256-bit (plus error-correcting code [ECC]) data bus. To meet configuration goals, processor modules necessarily support at least two microprocessors per module, memory modules support up to 2 GB of DRAM storage, and I/O port modules support up to 48 PCI slots. To meet performance goals, both buses must operate at a frequency of 100 MHz (10-ns cycle).

The AlphaServer 8000 platform topology has a number of advantages. The most significant advantage is that memory read latency from any processor to any memory array is comparable to the latency of a uniprocessor system. The delay associated with two interfaces--one address interface and one data interface--is all that is added into the path. In addition, the platform's simple bus topology features a low entry cost, a simple growth path (just insert another module) and flexible configuration (just about any module can be placed in any slot).

Operational Layer

The Operational Layer is so named for lack of a better descriptor. The layer is actually a place to define a high-level system clocking strategy. This strategy has two key components: definition of target operating frequencies and definition of a design methodology to support operation across all the defined operating frequencies. The design methodology component of this strategy may seem better suited for a higher order development layer, such as the Protocol Layer. However, because the methodology is logically associated with the system's operating frequency range and the operating frequency range provides a foundation for the Electrical Transport Layer, it seemed appropriate to include both components of the strategy in the Operational Layer.

In personal computer (PC)-class microprocessor systems, clock rates are typically slow (33 MHz to 66 MHz). Complementary components capable of operating at these speeds are readily available, e.g., transceivers, static random-access memory (SRAM), ASIC, DRAM, and programmable array logic (PAL). Therefore entire PC systems are typically run synchronously, i.e., the system logic (typically a motherboard) and the microprocessor run at identical clock speeds. Alpha processors, on the other hand, run at clock rates exceeding 250 MHz. The current state of complementary components makes running system logic at Alpha processor rates impractical if not impossible. Many of these components cannot perform internal functions at a 250-MHz rate, let alone transfers between components.

Digital's DEC 7000/10000 AXP systems solved the problem of Alpha microprocessor and system clock disparity by running both the Alpha microprocessor and the DEC 7000/10000 AXP system hardware at their respective maximum clock rates and synchronizing address and data transfers between the microprocessor and the system. Each time a transfer was synchronized, however, a synchronization latency penalty was added to the latency of the transfer. In the DEC 7000/10000 AXP system, two synchronization penalties--one for an address transfer to the system and one for a data transfer to the processor--are added to each memory read latency. With multiple data transfers, the data transfer from the system to the processor can be particularly large. When combined, the two penalties added nearly 125 ns to the DEC 7000/10000 AXP read latency, or approximately 25 percent of the total 560-ns latency. The same 125 ns, however, could add another 60 percent to the AlphaServer 8000 platform's lower target latency of 200 ns.

Given its latency goals, the AlphaServer 8000 platform implements a clocking methodology that minimizes synchronization penalties and thus minimizes read latency. This methodology involves clocking the entire AlphaServer system--up to the I/O channels--synchronous to the microprocessor in such a way that the Alpha microprocessor operates at a clock frequency that is a

direct multiple of the system clock frequency. With a 100-MHz (10-ns cycle) clock rate, for example, the AlphaServer 8000 could support a 200-MHz (5-ns cycle) Alpha processor using a 2[X] clock multiplier. Since the processor must still synchronize with a system clock edge when transferring address and data to the system, synchronization penalties are not eliminated altogether. They can, however, be limited to less than 10 ns, or 5 percent of the AlphaServer 8000 platform's total read latency.

Synchronous clocking by means of clock multiples is not unique and innovative in and of itself. The uniqueness of the AlphaServer 8000 clocking strategy lies in its flexibility. Since the AlphaServer 8000 platform must support at least three generations of Alpha processors to satisfy its product goals and the specific operating frequencies of those processors is difficult to predict, the AlphaServer 8000 platform must be capable of operating across a range of clock frequencies. Specifically the AlphaServer 8000 platform is capable of operating at clock frequencies between 62.5 MHz (16-ns cycle) and 100 MHz (10-ns cycle).

Operating across a range of frequencies may seem a trivial requirement to meet; if logic were designed to operate at a 10-ns cycle time, it should certainly continue to function electrically at a 16-ns cycle time. The real issues that this frequency range creates, however, are much more subtle. DRAMs, for example, require a periodic refresh. The refresh period for typical DRAM may be 50 milliseconds (ms). If a system were designed to a 10-ns clock rate, the system would be designed to initiate a DRAM refresh every 5,000,000 cycles. If the system were to be slowed to a 16-ns clock rate, the system would initiate a DRAM refresh every 80 ms based on the same 5,000,000 cycles. This could cause DRAMs to lose state and corrupt system operation. Similarly, DRAMs have a fixed read access time. The AlphaServer 8400/8200 TMEM module, for example, uses 60-ns DRAMs. If the DRAM's controller is designed as a 7-cycle controller and clocked at a 10-ns clock rate, it would access the 60-ns DRAM in 70 ns. If the system were slowed to a 16-ns clock rate, the system would, using the same controller, consume 112 ns in accessing the same 60-ns DRAM. This application of a single simple controller over a frequency range directly increases the DRAM's read latency and decreases the DRAM's bandwidth. This non-optimal DRAM performance in turn directly increases the system read latency and decreases the system bandwidth.

The AlphaServer 8000 platform design addresses these issues by implementing controllers that can be reconfigured based on the system's specific operating frequency. The TMEM module, for example, implements a reconfigurable controller for sequencing the reads and writes of its DRAMs. This controller has three settings: one for cycle times between 10 ns and 11.2 ns, one for cycle times between 11.3 ns and 12.9 ns, and one for cycle times between 13 ns and 16 ns. Each setting accesses the DRAMs in differing numbers of system clock cycles, but all three modes

access the DRAMs in approximately the same number of nanoseconds. By allowing flexible reconfiguration, this controller allows the TMEM to keep the DRAM's read latency and bandwidth as close to ideal as possible. Other examples of reconfigurable controllers are the TMEM's refresh timer and the TLEP's cache controller.

It should be noted here that the AlphaServer 8000 operating frequency range and processor-based frequency selection account for the disparities between the AlphaServer 8000 platform's bandwidth capability and the AlphaServer 8400 and 8200 products' bandwidth capabilities. The Alpha 21164 processor is the basis for the 8400 and 8200 products. This 300-MHz (3.33-ns cycle) microprocessor, combined with a 4[X] clock frequency multiplier, sets the system clock frequency at 75 MHz (13.3-ns cycle). This 13.3-ns cycle time, when applied to the 256-bit data bus, produces the 1,600 MB/s of data bandwidth. The cycle time increases the read latency of the 8400 and the 8200 to some extent as well, but the reconfigurable DRAM controllers help to mitigate this effect.

Electrical Transport Layer

When the bused system topology was selected in the Topological Layer of the AlphaServer 8000 platform development, a practical engineering decision was made to emphasize leading-edge technology as the means to accomplish our performance goals, as opposed to elegant architectural chicanery. It was observed in the topological discussion that, with the selected system topology, bus cycle time was critical to meeting the platform's performance goals. The Electrical Transport Layer of the platform development involved selecting or developing the centerplane, connector, clocking, and silicon interface technology that would allow the AlphaServer 8000 system bus to operate at a 100-MHz clock frequency. The most innovative of the technological developments that resulted from this effort were the platform's clocking system and its custom bus driver/receiver cell.

To put the AlphaServer 8000 100-MHz system bus goal in perspective, consider the operating frequencies of a number of today's highly competitive microprocessors.[8] The NexGen Nx586 operates at 93 MHz. The Intel Pentium, Cyrix M1, and AMD K5 all operate at 100 MHz. The Intel P6 operates at 133 MHz. In all these microprocessors, the 100+/-- MHz operation takes place on a silicon die less than 1 inch square. To meet its goals, the AlphaServer 8000 system bus must transfer data from an interface on a module in any slot on the system bus to an interface on another module in any other slot on the system bus across a 13-inch-long wire etch, with nine etch stubs and nine connectors, in the same 10 ns in which these microprocessors transfer data across 1-inch dies. By any measure this is a daunting task.

A breakdown of the elements that determine minimum cycle time aptly demonstrates the significance of clock system design, bus

driver design, and bus receiver design in the AlphaServer 8000 system bus development. Minimum bus cycle time is the minimum time required between clock edges during which data is driven from a bus driver cell on one clock edge and is received into a bus receiver cell on the next clock edge. An equation for determining the minimum cycle time is shown below. Tcmin is the minimum cycle time. Tprop is the time, measured from a rising clock edge, that is required for a bus driver to drive a new bus signal level to all system bus receivers. Tsetup is the time a bus receiver needs to process a new bus signal level before the signal can be clocked into the receiver cell. Tskew is the variation between the clock used to clock the bus driver and the clock used to clock the bus receiver. Tprop, Tsetup, and Tskew must all be minimized to achieve the lowest possible cycle time. The value of Tskew is determined by the system clock design. The values of Tprop and Tsetup are determined by the bus driver/receiver cell design.

$$Tcmin = Tprop + Tsetup + Tskew$$

AlphaServer 8000 System Bus Interface. To provide some context for the clock and bus driver/receiver discussions, it is necessary to briefly describe the standard AlphaServer 8000 system bus interface. Each AlphaServer 8000 module implements a standard system bus interface. This interface consists of five ASICs: one interfaces to the AlphaServer 8000 address bus and four interface to the AlphaServer 8000 data bus.[9] Each ASIC is implemented in Digital's 0.75-micrometer, 3.3-volt (V) complementary metal-oxide semiconductor (CMOS) technology and features up to 100,000 gates. Each ASIC is packaged in a 447-pin interstitial pin grid array (IPGA) and features up to 273 user I/Os.

Essential to the AlphaServer 8000 development were the speed of the CMOS interface ASIC technology and the development team's ability to influence the ASIC design process. "Influencing the design process" translated to the ability to develop a standard cell design library and process that is for and in concert with the development of the AlphaServer 8000 platform. The standard cell library, together with the CMOS silicon technology, provided the AlphaServer 8000 platform's required speed; complex logic functions (5 to 8 levels of complex logic gates) can be performed within a 10-ns cycle. "Influencing the design process" also translated to the ability to design a fully custom bus driver/receiver cell. Thus the development team could create a custom driver/receiver cell tailored to the specific needs of the AlphaServer 8000 system bus.

Clock Technology. The primary goal of the AlphaServer 8000 platform clock distribution system was to maintain a skew (Tskew) as small as possible between any two clocks in the system, while delivering clocks to all clocked system components. The goal of minimum skew is consistent with attaining the lowest possible bus cycle time, the highest possible system data bandwidth, and the

lowest possible memory read latency. It is important to note that in the AlphaServer 8000 platform, skew between clocks is not simply measured at the clock pins of the various clocked components. Skew is measured and, more important, managed at the actual "point of use" of the clock, for example, at the clock pins of ASIC flip-flops. This is an important point when dealing with ASICs. Since different copies of even the same ASIC design can have different clock insertion delays, additional skew can be injected between clocks after the clocks pass their ASIC pins.

The AlphaServer 8000 clock distribution system is implemented according to a two-tier scheme. The first tier, the system clock distribution, distributes a clean radio frequency (RF) sine wave clock to each system bus module. The second tier, the module clock distribution, converts the system RF sine wave clock to a digital clock and distributes the digital clock to each module's components. The module clock distribution tier also manages the skew between the system RF sine wave clock and all copies of each module's digital clock by means of an innovative "remote delay compensation" mechanism. The system clock distribution delivers clocks to the nine system bus module slots with a maximum of 40 picoseconds (ps) of skew. The module clock distribution delivers clocks to the various module components, most notably system bus interface ASICs, with a maximum of 980 ps of skew. The skew between any ASIC flip-flop on any AlphaServer 8000 module and any ASIC flip-flop on any other AlphaServer 8000 module is guaranteed to be less than 1100 ps.

The AlphaServer 8000 system clock distribution begins on the system clock module with a single-ended RF oscillator, a constant impedance bandpass filter, and a nine-way power splitter. The power splitter, by way of the bandpass filter, produces nine spectrally clean, amplitude-reduced copies of the oscillator sine wave. These nine outputs are tightly matched in phase and amplitude. They are distributed to the nine system bus module connectors by means of matched-length, shrouded, controlled-impedance etch. This design provides the modules with low skew (30 to 40 ps), high-quality (greater than 20-decibel signal-to-noise ratio) clocks.

The RF sine wave clock was an ideal selection for system clock distribution. By eliminating all high-order harmonics, the edge rates and propagation times of the clock wave are fixed and predictable across the distribution network. This predictability eliminates variation in the clock as perceived by the clock receiver on each module, thus minimizing skew. It also greatly reduces constraints on the design of connectors, etch, termination, etc.

The AlphaServer 8000 module clock distribution is a boilerplate design that is replicated on each AlphaServer 8000 module. On each module, the system sine wave clock is terminated by a single-ended-to-dual-differential output transformer. This transformer produces two phase- and amplitude-matched

differential clocks that are fed into one or two AlphaServer 8000 clock repeater chips (DC285 chips). These chips convert the sine wave clocks into CMOS-compatible digital clocks; distribute multiple copies of the digital clocks to various module components, including the system bus interface ASICs; and perform remote delay clock regulation on each clock copy.

The remote delay clock regulation is performed by a custom, digital delay-locked loop (DLL) circuit. This DLL circuit was devised specifically to deskew clocks all the way to their point of use in the system bus interface ASICs. The principles of DLL-based remote delay clock regulation are simple. The sum of the delays associated with (1) the clock repeater chips, (2) the module clock distribution etch, and (3) the ASIC clock distribution network constitutes the insertion delay of the ASIC point-of-use clock with respect to the system sine wave clock. With no clock regulation, this delay appears as skew between the system clock and the point-of-use ASIC clock. Between ASICs on different modules, a fixed portion of the clock insertion delay will correlate and need not be factored into the overall system skew. Since the insertion delay can easily approach 7 ns, however, the variation in the insertion delays to different ASICs, which must be factored into the overall system skew, can also be significant. To reduce the skew between the system sine wave clock and the point-of-use ASIC clock, the clock repeater uses a digital delay line to add delay to the clock repeater output clock. Enough delay is added so that the insertion delay plus the delay-line delay is equal to an integer multiple of the system clock. This delay moves the point-of-use clock ahead to a point where it again lines up with the system clock. As the system operates, the system and point-of-use clocks may drift apart. In response, the clock repeater adjusts its delay line to pull the clocks back together. This process of delaying clocks and dynamically adjusting the delay is called remote delay clock regulation. When the clock separation, or drift, is measured by a clock "replica loop" and the clock delay is inserted by means of a digital delay line, the process is called DLL-based remote delay clock regulation.[10] Using the clock repeater chips in this way, AlphaServer 8000 modules are able to achieve point-of-use to point-of-use skew of approximately 930 to 980 ps. Combined with the system module-to-module skew of 30 to 40 ps, this provides the quoted system-wide clock skew of no more than 1,100 ps.

It is worth noting that although the AlphaServer clock repeater was primarily developed for use with system bus interface ASICs, it is a generally versatile part. It may, for instance, be used with non-ASIC parts such as transceivers and synchronous SRAMs. In these cases, the clock pin of the non-ASIC part is treated as the point of use of the clock. The clock repeater may also be used for precise positioning of clock edges. On the TLEP module, for example, the Alpha 21164 microprocessor's system clock is synchronized to a clock repeater output by means of a digital phase-locked loop (PLL) on the microprocessor. The Alpha 21164's

PLL operates in such a way that the 21164's clock is always in phase with or always trailing the system (reference) clock. It can trail by as much as 2 ns. Such a large clock disparity in this fixed orientation can create setup time problems for transfers from the Alpha 21164 to the system and hold-time problems for transfers from the system to the Alpha 21164. The TLEP design addressed this problem by lengthening the replica loop associated with the Alpha 21164 clock and thereby shifting the microprocessor clock 1 ns earlier than the balance of the clock repeater output clocks. Since the Alpha 21164 clock was either in phase or 2 ns later than its associated clock repeater clock, which is 1 ns earlier than the rest of the clock repeater clocks, the 21164 clock now appears to be either 1 ns earlier or 1 ns later than the rest of the clock repeater system clocks. This centering of the module clocks with respect to the 21164 clock halves the required setup or hold margin.[11, 12, 13, 14]

Bus Driver Technology. Like the AlphaServer 8000 clock system, the AlphaServer 8000 system bus driver/receiver cell was specifically designed to minimize bus cycle time. As with the clock logic, the goal of minimizing cycle time was a result of the effort to minimize system read latency and maximize system data bandwidth. In the effort to minimize the bus cycle time, the design of the AlphaServer 8000 bus driver/receiver cell was focused on minimizing the propagation delay (Tprop) of the system bus driver circuit and minimizing the setup time (Tsetup) of the system bus receiver.

The AlphaServer 8000 system bus driver/receiver cell is a fully custom CMOS I/O cell, which incorporates a bus driver, a bus receiver, and an output flip-flop and an input flip-flop in a single cell. Consisting of nearly 200 metal oxide semiconductor field-effect transistors (MOSFETs), the bus driver cell is powered by standard 3.3-V CMOS power, but drives the bus at a much lower 1.5-V level (i.e., voltage swings between 0 and 1.5 V). This low voltage output serves to reduce the bus driver's power consumption and permits compatibility with future CMOS technologies that are powered by voltages less than 3.3 V. Many of the bus driver cell's critical characteristics are "programmable," such as the 1.5 V output, the receiver switching point, the driver's drive current limit, and the driver's rise and fall times. These values are programmed and, most important, are held constant by means of reference voltages and resistances external to the bus driver/receiver cell's ASIC package. They allow the cell to produce uniform, predictable, high-performance waveforms and to transmit and receive data in a clock cycle of 10 ns.

The bus driver/receiver's high performance begins with its output flip-flop and driver logic. The output flip-flop is designed for minimum delay and is integrally linked to the output driver. This configuration produces clock-to-output times of 0.5 ns to 1 ns. The output driver itself, with its programmable output voltage and edge rates, allows the shape of the output waveform to be

carefully controlled. The cell's programmable values are set such that the AlphaServer system bus waveform balances the edge rate effects of increased crosstalk with increased propagation delay. Furthermore, the bus waveform is shaped in such a way that it allows incident wave transmission of signals. As such, a signal can be received on its initial propagation across the bus centerplane, as opposed to waiting for signal reflections to settle. All the driver characteristics serve to reduce bus settling time. When combined with the low clock-to-output time of the output flip-flop, this reduced settling time produces a very low driver circuit propagation delay (Tprop).

The bus driver/receiver cell's receiver and input flip-flop further contribute to its high performance. Designed with a programmable reference voltage, the receiver has a very precise switching point. Whereas typical receivers may have a 200-millivolt (mV) to 300-mV switching window, the bus driver/receiver cell's receiver has a switching window as small as 40 mV. This diminished switching uncertainty directly reduces the receiver's maximum setup time. The input flip-flop's master latch is a sense-amplifier-based latch as opposed to a simple inverter-based latch. The sense amplifier, with its ability to resolve small voltage differentials much faster than standard inverters, allows the master latch to determine its next state much more rapidly than a standard latch. This characteristic serves to reduce both the receiver's setup and hold time requirements.

In general, the setup and hold time requirements of a state element are interrelated. The setup time, for example, can be reduced at the expense of hold time. Since setup time contributes to cycle time and hold time may not, reducing setup time is desirable. The AlphaServer 8000 bus driver/receiver cell requires at most 300 ps of combined setup and hold time. However, since the edge rates of the cell driver are so well controlled, the minimum propagation time for a bus signal is always guaranteed to exceed 300 ps. As a result, the bus receiver circuit is designed with all 300 ps charged as hold time. This renders a minimized receiver setup time (Tsetup) of 0 ps.

The AlphaServer 8000 bus driver/receiver cells have a number of additional features that further reduce the propagation delay (Tprop) of the driver circuit. The cell, for example, features in-cell bus termination, which provides the system bus with full, distributed termination. Simulations have shown that such distributed termination can provide an advantage of 500 ps over common end termination. The bus driver/receiver cell's termination resistance, like other cell parameters, is programmable and made identical throughout all system ASICs by means of a reference resistor external to each ASIC.

The bus driver/receiver cell also features a special preconditioning function that improves the driver's propagation delay by as much as 1,500 ps. This feature causes all bus drivers

to begin driving toward the opposite state each time they receive a new value from the bus. If the bus is changing state from one cycle to the next, the feature causes all drivers to begin driving the bus to a new state in the next cycle. In doing so, all bus driver cell drivers contribute current and accelerate the bus transition. If the bus is not changing from one cycle to the next, the drivers simply push the state of the bus toward the opposite state, but only to a benign voltage well short of the switching threshold.

All of the bus driver cell's programmable features, such as switching point, output voltage, edge rates, and termination resistance, make the bus driver cell a very stable and high-performance interface cell. The existence of these features, however, is an element of the bus driver cell's complementary process-voltage-temperature (PVT) compensation function. PVT compensation is meant to make a device's operating characteristics independent of variations in the semiconductor process, power supply voltage, and operating temperature. By applying PVT compensation in every AlphaServer system bus interface ASIC, bus driver cells in different ASICs, for example, can drive nearly identical system bus waveforms even if those ASICs come from manufacturing lots with varying speed characteristics. AlphaServer 8000 PVT compensation is based on reference voltages and resistances provided by very precise, low-cost, module-level components. The PVT compensation circuit measures these references and configures internal voltages and resistances so that all bus driver cells can operate uniformly and predictably. By creating predictability and thus reducing uncertainty and skew, bus cycle time is minimized.

Signaling Layer

Powerful though it may be, the AlphaServer 8000 bus driver/receiver cell is not without limitations. During its development, it was found that the bus driver cell could be developed to drive the AlphaServer 8000 system bus in 10 ns under a limited number of conditions. When the driver cell asserted a deasserted (near 0 V) bus line or deasserted a bus line that had been asserted (near 1.5 V) for only one cycle, for example, 10-ns timing could readily be met. When the driver attempted to deassert a bus line that had been asserted for more than one cycle by multiple drivers, however, 10-ns timing could not be met. These limitations have significant implications for protocol development. Protocols typically have a number of signals that can be driven by multiple drivers. These may include cache status signals and bus flow control signals. Protocols also typically include a number of signals that can be asserted for many cycles. These may include bank busy signals or arbitration request signals. Clearly the implications are that the limitations of the bus driver/receiver cell would cause the system either to fall short of its cycle time and performance goals or to be incapable of supporting a workable bus protocol.

With the bus driver/receiver cell pushing technology to its limits, the solutions to this problem were extremely limited. The system cycle time could be slowed down to accommodate all signal transitions within a single cycle, regardless of the charge state of the signal line; or a signaling protocol could be developed that would avoid charging a signal to the point where it could not transition in 10 ns; or the physical topology of the system could be reconsidered with the goal of finding a new topology that met the system goals at a slower clock rate. The first option of slowing the clock was clearly unacceptable; it could not satisfy the system's latency and bandwidth goals given the system's topology. The third option could potentially satisfy the system's latency and bandwidth goals, but came at the expense of the favorable qualities of the simple bus outlined in the Topological Layer and at the risk that the new topology would suffer similar, unforeseen pitfalls. The option of developing a signaling protocol, on the other hand, could satisfy the system's performance goals with little or no risk. A signaling protocol was clearly the practical solution to the bus driver/receiver cell limitations.

The Signaling Layer of the platform development model introduces the AlphaServer 8000 signaling protocol. This protocol was developed by creating a list of signal classes, based on driver counts and assertion and deassertion characteristics, and by associating a specific signaling protocol with each class. The signal classes and their protocols are listed in Table 2. As the AlphaServer 8000 primary protocol was developed, each bus signal was assigned a signal class. As AlphaServer 8400/8200 hardware was developed, each bus signal was designed to operate according to the signaling protocol associated with its signaling class. The system bus address and data signals, for example, fall into the second class of signals. As a result, the AlphaServer 8400/8200 modules are designed to leave tristate cycles between each address and data transfer on the system bus.

The AlphaServer system bus cache status signals (TLSB_Shared and TLSB_Dirty) and the system bus flow control signals (TLSB_Hold and TLSB_Arb_Suppress) demonstrate a noteworthy paradigm that results from the AlphaServer 8000 signaling protocol. All these signals are defined such that at times they must be asserted for multiple cycles. All these signals also fall into the fourth signal class, which expressly prohibits driving the signals for multiple cycles. When these two contradictory requirements exist, the result is a class of signals pulsed to indicate multiple cycles of constant assertion. Logic inside each AlphaServer 8000-based module must be designed to convert these pulsed signals to constantly asserted signals within its system bus interface. Note that when signals such as these are discussed in the protocol sections of this paper, the term "asserted" is used to imply constant assertion, with the understanding that the signals may in fact be pulsed.

Table 2 AlphaServer 8000 Signal Classes

Signal Class	Driver Count and Signal Assertion/Deassertion Characteristics	Signaling Protocol
1	Single driver with multiple receivers	Never driven more than two consecutive cycles
2	Multiple drivers with multiple receivers	Tristate cycle on the bus when driver changes
	One driver at a time	Never driven more than two consecutive cycles
3	Multiple drivers with multiple receivers	Value received on signal deassertion is unpredictable and must be ignored
	Many drivers at once possible	Tristate cycle on the bus when driver changes
	Assertion time may differ from driver to driver	Never driven in two consecutive cycles
	Deassertion time is fixed	
4	Multiple drivers with multiple receivers	Value received on signal deassertion is unpredictable and must be ignored
	Many drivers at once possible	Tristate cycle on the bus when driver changes
	Timing is fixed	Never driven in two consecutive cycles

Consistency Check Layer

The Consistency Check Layer defines a method for maintaining system integrity. Specifically, it defines methods for detecting errors and inconsistencies in the system and, more important, methods for logging errors in the presence of historically disabling errors. Although it does not contribute directly to the AlphaServer 8000 platform's performance goals or stated product goals, the Consistency Check Layer contributes an extremely useful feature to the AlphaServer 8000 products. It is included in the paper for the sake of completeness in the analysis of the seven-layer platform development model.

The AlphaServer 8000-based systems employ a number of error-checking mechanisms. These include transmit checks, sequence checks, assertion checks, and time-outs. If any error is detected by an AlphaServer 8000 module by means of these mechanisms, the module responds by asserting a special "Fault"

signal on the AlphaServer 8000 system bus. This Fault signal has the effect of partially resetting all system bus interfaces and processors, and trapping the processors to "machine check" error-handling routines. The partial reset clears all system state, with the exception of error registers. This resynchronizes all system bus interfaces and eliminates all potentially unserviceable transactions left pending in the system. Thus the system can begin execution of the machine-check routines in a reset system. Although the routines are not guaranteed to be able to complete an error log in the presence of an error, it is believed that this mechanism will increase the probability of a successful error log.

The AlphaServer 8000 platform's Fault error-handling feature is particularly useful in recovering error state from a computer in a "hung" state. A computer enters a hung state when an error occurs that stops all progress in the computer system. If a processor is waiting for a response to a read, for example, and the read response is not forthcoming due to an error, the system hangs while waiting for the response. The desktop model for error handling would require a system reset to recover from such an error. The process of the system reset, however, would purge error state. The purge, in turn, makes error diagnosis extremely difficult. This desktop model is not unique to desktop systems. It is also employed in server-class machines such as Digital's DEC 7000/10000 AXP systems. Although this model may be acceptable on the desktop, it is most undesirable in an enterprise server system. The AlphaServer 8000-based systems use a time-out counter to detect a hung system and the Fault error-handling technique to recover an error log in the event of a hung system. The result is a robust error-handling system that is appropriate in an enterprise server.

Primary Protocol Layer

The Primary Protocol Layer of the platform development assigns names and characteristics to the various system bus signals and uses these names and characteristics to define higher-order system bus transactions and functions. System bus transactions may include reads of data from memory or writes of data to memory. These transactions are the primary business of a computer system and its protocol. If a system efficiently executes read and write transactions, it will perform better than a system that does not. System bus functions may include mapping memory addresses to specific memory banks or arbitrating for access to system buses. These functions enable system bus transactions to operate in environments with multiple processors arbitrating for access to the system bus and multiple banks of memory.

AlphaServer 8000 system bus transactions relate directly into the platform's performance metrics. The system's memory read latency, for example, is equal to the time it takes for a processor to issue and complete a system bus read transaction. The number of

system bus transactions and their associated data that the system bus can process in a given period of time define the system bus bandwidth.

The components of a typical memory read transaction are shown in a timeline in Figure 4. This timeline of components is based on a system that is an abstract of the DEC 7000/10000 AXP systems. To minimize a system's memory read latency, each component of the read transaction timeline must be minimized. Components 1, 3, 7, and 8 of the timeline are simply data and address transfers across buses and through interfaces. The delays associated with these components are largely determined by system cycle time; they cannot be affected by the protocol to any great extent. Component 5 is the DRAM access time. It is minimized by the reconfigurable controllers described in the Operational Layer. The remaining components, (2) address bus arbitration, (4) memory bank decode, and (6) data bus arbitration, fall into the domain of the primary protocol. These elements must be designed to contribute minimal delay to the overall latency.

[Figure 4 (Components of Memory Read Latency) is not available in ASCII format.]

The effects of protocol on a system's data bandwidth are a little more difficult to quantify than the effects of protocol on memory read latency. In general, the theoretical maximum system bandwidth is equal to either the sum of the bandwidths of the system's memory banks or the maximum system bus bandwidth, whichever is smaller. If the system bandwidth is limited by memory module bandwidth, it is essential to keep as many memory modules active as possible. If, for example, eight banks of memory are required to sustain 100 percent of the maximum system bandwidth, but the system can support only four outstanding commands, only four banks can be kept busy and only 50 percent of the maximum bandwidth can be rendered. In another example, if 10 percent of the time this system freezes all but one bank of memory to perform special atomic functions on special data blocks, the system's bandwidth will suffer nearly a 10 percent penalty (73/80 possible memory accesses versus 80/80 possible memory accesses). If the system bandwidth is limited by the bandwidth of the system bus, the maximum system bandwidth can be achieved only when the protocol allows system modules to drive data onto the system data bus in every available cycle on the data bus. When a processor reads a block of data from a second processor's cache, for example, the second processor may have to stall the data bus to allow it to drive the read data onto the system's data bus as prescribed by the system protocol. A stall of the data bus translates into unused data bus cycles and degradation of real system bandwidth. Thus to maximize real system bandwidth, system bus and memory bank utilization must be maximized, and stalls in system bus activity and stalls in memory bank activity must be minimized.

The following sections begin with an overview of the basic

AlphaServer 8000 platform protocol and how this basic protocol influences system performance. This section is followed by a discussion of how the various protocol components identified as elements of memory read latency (i.e., memory bank mapping, address bus arbitration, and data bus arbitration) affect the latency. These sections conclude with a discussion of subblock write transactions and their effects on system bandwidth.

AlphaServer 8000 Protocol Overview. The platform development Topological Layer defined the AlphaServer 8000 system bus as having separate address and data buses. The AlphaServer 8000 system bus protocol defines how system bus transactions are performed using these two buses. According to the protocol, processor and I/O port modules initiate read and write transactions by issuing read and write commands to the system address bus. These address bus commands are followed sometime later by an associated data transfer on the data bus. All data transfers are initiated in the order in which their associated address bus commands are issued. Cache coherency information for each system bus transaction is broadcast on the system bus as each transaction's data bus transfer is initiated. Each data transfer moves 64 bytes of data (only 32 bytes of which are valid for programmed I/O transfers). Figure 5 shows an example of AlphaServer 8000 system bus traffic. In cycle 1 a read transaction, r0, is initiated on the system address bus. In cycle X, the data transfer for read r0 is initiated on the system data bus by means of the system bus Send_Data signal, the assertion of which is indicated with a value of i0. As this data transfer is initiated, the status, s0, is also driven on the system bus. In cycle X+2, all system bus modules have an opportunity to stall or to control the flow to the system data bus. In this example, the bus is not stalled, as indicated by a value of n. Finally, given that the bus is not stalled, the 64 bytes of read data associated with read r0 are transferred across the system bus during cycles X+5 and X+6. In addition to read r0, Figure 5 also illustrates the execution of a write, w1, and another read, r2. Note that data transfer initiation, data bus flow control, and data transfer are pipelined on the system data bus in the same order as their associated commands were issued to the address bus. Note further that this diagram represents 100 percent utilization of the system data bus (one data transfer every three cycles). With a 10-ns cycle time, this utilization would translate to 2.1 GB per second of bandwidth.

[Figure 5 (Example of AlphaServer 8000 System Bus Traffic) is not available in ASCII format.]

The AlphaServer system address bus uses two mechanisms to control the flow of system bus transactions. First, processor and I/O port modules are not allowed to issue commands to memory modules that are busy performing some DRAM access for a previously issued system bus transaction. The state of each memory bank is communicated to each processor by means of system bus Bank_Available signals. If a processor or I/O port seeks access

to a given memory bank and that memory bank's Bank_Available signal indicates that the bank is free, the processor or I/O port may request access to the address bus and, if granted access by the system arbitration logic, issue its transaction to the address bus. If a processor or I/O port seeks access to a given memory bank and that memory bank's Bank_Available signal indicates that the bank is not free, the processor or I/O port will not request access to the system address bus. Thus, unless all memory banks are busy or unless the total of the busy memory banks includes all banks that are needed to service the system's processors and I/O ports, the address bus will continue to transmit commands. The second mechanism for controlling the flow through the address bus is the system bus Arb_Suppress signal. If any system bus module runs out of any command/address-related resource, such as command queue entries, it can assert this signal and prevent the system arbitration logic from granting any more transactions access to the bus. The Arb_Suppress signal is useful, for example, in a system configuration with 16 memory banks but only eight entries worth of command queuing in a processor.

The AlphaServer 8000 system data bus has its own flow-control mechanism, the system bus Hold signal, which is independent of the address bus flow-control mechanisms. The Hold signal, shown as Data Bus Flow Control in Figure 5, is asserted in response to the initiation of a data bus transfer. Normally, data bus transfers are initiated on the data bus when an AlphaServer 8000 memory module asserts the Send_Data signal. Send_Data is asserted by a memory module based on the state of the module's DRAMs: When servicing a read transaction, the memory will assert Send_Data when its DRAM read is complete; when servicing a write transaction, the memory will assert Send_Data as soon as its turn on the data bus comes up. Five cycles after the assertion of Send_Data, some module drives data onto the data bus. If a module is required to drive data in response to an assertion of Send_Data and is unable to do so, it will assert the Hold signal two cycles after the assertion of Send_Data. This may occur if a processor module must source read data from its cache and cannot fetch the data from the cache as quickly as the memory module can fetch data from its DRAMs. If, on the other hand, a module is required to receive data in response to an assertion of Send_Data and is unable to do so, it too will assert the Hold signal two cycles after the assertion of Send_Data. This may occur if no receiving module's data buffers are available to receive data. Each module that asserts Hold two cycles after Send_Data will continue to assert Hold every other cycle--as prescribed by the AlphaServer 8000 signaling protocol--until it is ready for the data transfer. Three cycles after all modules are ready and deassert the Hold line, data is finally transferred. Figure 6 shows a read, r0, that experiences one pulse of the system bus Hold signal.

[Figure 6 (Read with One Cycle of Hold--Five Reads Sourced by a Processor) is not available in ASCII format.]

It is important to note that the address bus and the data bus have independent means and criteria for initiating transactions and controlling the flow of transactions. The address bus initiates address bus commands based on processor and I/O port module requests and controls the flow based on the state of address-related resources. The data bus initiates data transfers in the same order as the address bus transmitted commands by means of the Send_Data signal. Send_Data is usually asserted by a memory module based on the state of the module's DRAMs. The data bus flow is controlled based on the state of various data-related resources. The differing means and criteria for initiation and flow control allow the two buses to operate almost independently of one another. This independence translates into performance because it allows the address bus to continue to initiate commands even as the data bus may be stalled because of a conflict. Continuous command initiation translates into more continuous system parallelism and thus more system bandwidth. Figures 6 and 7 illustrate this point. Both figures illustrate systems that are issuing a series of processor reads to blocks that must be sourced from another processor's cache. In both cases, processors require two more cycles than main memory banks to source read data. As such, two cycles of Hold assertion must periodically occur on the data bus. Figure 6 illustrates the operation of the AlphaServer 8000 system bus, showing that although the data bus had to be held in cycle 6, the address bus was able to continue issuing commands. As a result, each processor sourcing data begins its read of cache data as soon as possible and is guaranteed to be ready to drive data without Hold cycles when its turn comes up on the data bus. With the illustrated series of five reads, the two Hold cycles result in a 12 percent degradation in system bandwidth. If the series of reads is lengthened toward infinity, the percent of degradation approaches 0. Figure 7 illustrates the operation of a rigidly slotted bus, like that of the DEC 7000/10000 AXP system, normalized to the AlphaServer 8000 topology. As shown, each time the data bus is stalled, so too is the address bus. This prevents the fourth and fifth reads from getting the headstart necessary to prevent subsequent stalls of the data bus. The result is a 20 percent degradation in performance for the five reads illustrated. If the series of reads is lengthened toward infinity, the percent of degradation settles to 18 percent. Clearly the AlphaServer 8000 approach produces superior data bandwidth characteristics.

[Figure 7 (Five Reads Sourced by a Processor in a Rigidly Slotted System) is not available in ASCII format.]

It is also important to note that the AlphaServer 8000 address bus and data bus have different maximum bandwidths. Commands can be issued to the address bus every other cycle. With a 10-ns cycle time, this translates into 50 million commands per second. The data bus, on the other hand, can transfer one block of data every three cycles. With a 10-ns cycle time, this translates into

33.3 million data blocks per second. This excess of address bus bandwidth is useful in the development of low-latency arbitration schemes.

Memory Bank Mapping. Digital's previous server systems, like the VAX 6000 series and the DEC 7000/10000 AXP-series, have employed a common approach to address-to-memory-bank mapping. In this approach, all memory modules implement address range registers. As commands and addresses are transmitted across the system bus, the memory banks compare the addresses against their address range registers to determine if they must respond to the command. An address range comparison can involve a significant number of address bits and, as a result, can become logically complex enough to consume two 10-ns cycles of time. These two cycles can be added directly to memory read latency.

The low-latency focus of the AlphaServer 8000 platform prompted a change in bank mapping schemes. In AlphaServer 8000 systems, the address range registers have been moved onto the processor and I/O port modules. The range registers output a 4-bit bank number that is shipped across the system bus with each command and address. Each memory bank compares each bank number transmitted across the system bus to 4 bits in a programmable bank number register to determine if it should respond to the system bus command.

This bank mapping logic configuration helps to reduce AlphaServer 8000 memory read latency. Because the bank mapping is done on the nodes that issue commands to the address bus, the lengthy address comparison can be done in parallel with address bus arbitration, eliminating its two-cycle delay from the memory read latency. The address comparison traditionally done in the memory bank logic is now replaced with a simple 4-bit comparison, which can easily be done in a single cycle. The overall effect is that the AlphaServer 8000 bank mapping protocol consumes at least one cycle less than Digital's traditional bank mapping protocol. This equates to one less cycle--10 ns minimum--of memory read latency.

Address Bus Arbitration. AlphaServer 8000 systems employ a distributed, rotating-priority arbitration scheme to grant access to their address buses. Processor and I/O port modules request access to the address bus based on requests from microprocessors and I/O devices, and on the state of the system's memory banks, as described in the section AlphaServer 8000 Protocol Overview. Each module evaluates the requests from all other modules and, based on a rotating list of module priorities, determines whether or not it is granted access to the bus. Each time a module is granted access to the bus, its priority is rotated to the lowest priority spot on the priority list.

The AlphaServer 8000 arbitration scheme operates in a pipelined fashion. This means that modules request access to the bus in one cycle, arbitrate for access to the bus in the next cycle, and finally drive a command and address onto the bus one cycle later.

In terms of processor-generated read requests, this means that, at best, a system bus read command can be driven onto the system address bus two cycles after its corresponding cache read miss is generated on the processor module. This adds two cycles of delay to the memory read latency.

To reduce memory read latency in components associated with address bus arbitration, the AlphaServer 8000 platform employs a technique called "early arbitration." Early arbitration allows a module to request access to the address bus before it has determined if it really needs access to the data bus. If the module is granted access to the address bus but determines that it does not need or cannot use the access, it will drive a No-Operation or NoOp command in the command slot that it is granted. This feature is particularly useful on processor modules. It allows a processor to request access to the bus for a read command in parallel with determining if the read command will hit or miss in the processor's cache. If the read results in a cache hit and the processor is granted access to the address bus, then the processor issues a NoOp command. If the read results in a cache hit and the processor is not granted access to the address bus, the processor discontinues requesting access to the bus. When applied in this manner, this feature can remove two cycles of delay from the memory read latency. This feature is also key to the AlphaServer 8000 memory bank decode feature that allows address-to-memory bank decode to proceed in parallel with system bus arbitration. This is to say, it allows a processor or I/O port module to request access to the address bus before it can determine which memory bank it is trying to access and before it can determine if that memory bank is available. If a module is granted access the bus and the bank it is trying to access is not available, then the module issues a NoOp command. If a module is not granted access to the bus and the bank it is trying to access is not available, then the module discontinues requesting access to the bus until the bank becomes available. When applied this way, this feature eliminates at least one cycle from the memory read latency, as described in the section Memory Bank Mapping.

The excess address bus bandwidth noted in the protocol overview allows some amount of early arbitration to take place without affecting system performance. When system traffic increases, however, excessive early arbitration can steal useful address bus slots from nonspeculative transactions and as a result degrade bus bandwidth. In fact, in certain pathological cases, excessive early arbitration by modules with high arbitration priority can permanently lock out requests from lower priority modules. To eliminate the negative effect of early arbitration, the AlphaServer 8000 employs a technique called "look-back-two" arbitration. This technique relies on the fact that modules must resolve all cache miss or bank availability uncertainties for early arbitrations within the two cycles required for an early request and its arbitration. This fact implies that any module that has been requesting access to the address bus for more than two consecutive cycles is requesting in a nonspeculative manner.

As such, the AlphaServer 8000 arbiter keeps a history of address bus requests and creates two prioritized groups of requests based on this history. It creates a high-priority group of requests from those requests that have been asserted for more than two cycles and a low-priority group of requests from those requests that have been asserted for two cycles or less. It applies the single set of rotating priorities, described above, to both sets of requests. If there are any requests in the high-priority group, the arbiter selects one of these based on the rotating priority set. If there are no high-priority requests, the arbiter selects a request from the lower priority group based on the rotating priority set. This functionality limits early arbitration to only those times when there are nonspeculative requests in the system. It allows the AlphaServer 8000 platform to take advantage of latency gains associated with early arbitration and processor and I/O port based bank decode, without degrading bandwidth in the process.

Data Bus Arbitration. The AlphaServer 8000 data bus transfers blocks of data in the same order that the commands corresponding to those blocks are issued on the address bus. This eliminates data bus arbitration per se. In-order data return is accomplished by a simple system of counters and sequence numbers. Each time a command is issued to the address bus, it is assigned a sequence number. Sequence numbers are assigned in ascending order. Each time a block of data is driven on the data bus, a data bus counter is incremented. Each module waiting to initiate a data transfer in response to some address bus command compares the sequence number associated with its command with the data bus counter. When a module's sequence number matches its data bus counter, it is that module's turn to initiate a data bus transfer.

It is arguable that in-order data return is not the optimum data scheduling algorithm. If the scenario shown in Figure 6 were reshaped such that only read r0 sourced data from another processor and the penalty for sourcing data from a processor were more severe--a longer data bus Hold requirement--the result would be more significant bandwidth degradation. This new scenario is illustrated in Figure 8. With more efficient data scheduling, it is conceivable that data bus utilization could be improved by using data slots abandoned under the sizable Hold window in Figure 8. The latter scenario is illustrated in Figure 9. Clearly the system in Figure 9 has improved upon the bandwidth of the system in Figure 8.

[Figure 8 (Bandwidth Degradation as a Result of In-Order Data Transfers) and Figure 9 (Improved Bandwidth with Out-of-Order Data Transfers) are not available in ASCII format.]

What Figure 9 cannot show are all the implications of out-of-order data transfers. With as many as 16 outstanding transactions (8 in the AlphaServer 8400/8200) active in the system at any one time, the task of producing a logic structure

capable of retiring the transactions in order is enormous. Furthermore, the retiring of transactions out of order complicates the business of maintaining coherent, ordered memory updates. Finally, it was felt that the parallelism made possible by the independent address and data bus would help to mitigate many of the negative effects associated with the in-order data transfers. For these reasons, a practical decision was taken to transfer data on the system data bus in the order that the associated commands were issued to the system address bus.

Subblock Writes. To support a range of I/O subsystems, AlphaServer 8000 I/O port modules must support writes of data as small as longwords (32 bits), words (16 bits), and bytes. Given the AlphaServer system bus block size of 64 bytes, these writes are referred to as subblock writes. The execution of a subblock write consists of reading a block of data from a system memory bank, overwriting just the portion of the block addressed by the subblock write, and writing the entire block back to memory. The difficulty with performing this operation arises when a "third-party" module--defined here as a module other than the one performing the subblock write--modifies the block between the read portion of the subblock write and the write portion of the subblock write. To correctly complete the subblock write, the I/O port module must merge the subblock write data into the block as it was after the third-party module modified it. This problem can be resolved in one of two ways: (1) by means of a small cache on the I/O port module that updates the I/O port's copy of the block based on the third-party write, or (2) by means of an atomic read-modify-write that disallows the third-party write altogether.

In an ideal world, I/O port modules would implement a small one-block cache for the purpose of subblock writes. This cache would allow the I/O module performing the subblock write to update its copy of the block targeted by the subblock write with modified data from third-party modules. Unfortunately, not all processors broadcast modified data to the system. Many processors, for example, use a read-invalidate protocol. In a read-invalidate protocol, when a processor wishes to modify a block, it issues a command that invalidates all other copies of that block in the system and then modifies the block of data in its cache. If such an invalidate command invalidated the block in an I/O port module's subblock write cache, the I/O port module would be forced to re-read the block. There is no guarantee, however, that another invalidate will not occur between the re-read of the block and the write of merged data back to memory. As such, the I/O port module may never be able to complete the subblock write. I/O port caching is therefore not a workable solution.

Atomic read-modify-write sequences disallow third-party writes to a given block between the read portion of a subblock write and the write portion of a subblock write. As such, the atomic read-modify-write sequence does guarantee the timely completion

of a subblock write. Implementations of atomic read-modify-write sequences are designed to disallow accesses to some size portion of the memory region that contains the subblock address, between the read and write portions of the subblock write. The size of the memory region can vary from a single block of data to a single bank of memory to the entirety of memory. If the size of the memory region is small, such as a single data block, design complexity is significant; but the impact of locking out access to a single block of memory is insignificant to bandwidth. Conversely, if the size of the memory region is large, such as the entirety of memory, design complexity is insignificant; but the impact of locking out accesses to the entirety of memory for any period of time can be significant to system bandwidth.

The AlphaServer 8000 platform supports atomic read-modify-write sequences by locking out accesses within memory-bank-sized memory regions. This middle ground memory-region size provides the AlphaServer 8000 with a practical balance between design complexity and system bandwidth. The AlphaServer 8000 platform implements memory bank granularity atomic read-modify-write accesses by means of special Read_Bank_Lock and Write_Bank_Unlock address bus commands, and by leveraging the existing memory bank flow control mechanisms. Specifically, Read_Bank_Lock commands function like normal read commands, except that their targeted memory banks are left busy after the read transaction is complete. Memory banks locked by Read_Bank_Lock commands remain busy until a Write_Bank_Unlock command is issued from the same module that issued the Read_Bank_Lock command. While a memory bank is busy, no module other than the module that locked the bank by means of a Read_Bank_Lock command will even request access to the bank, as required by standard arbitration protocol. This approach provides for atomic read-modify-write sequences and coherent subblock writes. This protocol works regardless of the number of I/O modules in the system and regardless of arbitration priorities.

Superset Protocol Layer

The AlphaServer 8000 primary protocol provides all the basic constructs required to perform basic system functions, such as memory reads and writes, local register reads and writes, and mailbox-based I/O register reads and writes. The protocol performs these basic functions with a high level of efficiency and performance. Some additional functionality, such as PCI direct-programmed I/O register accesses, can be functionally satisfied by the primary protocol but cannot be satisfied in a way that does not severely degrade the performance of the entire AlphaServer 8000 system. As such, the AlphaServer 8000 platform allows for Superset Protocols, i.e., protocols that are built upon the basic constructs (reads and writes) of the AlphaServer 8000 primary protocol.

PCI direct-programmed I/O register reads can take more than a

microsecond to complete. If these reads were completed by means of the AlphaServer 8000 nonpended, strictly ordered primary protocol, the AlphaServer system data bus would be stalled for a full microsecond each time a PCI programmed I/O read was executed. Such stalls would have a disastrous effect on system bus bandwidth and system performance.

The PCI programmed I/O problem is solved on the AlphaServer 8000 platform by implementing a PCI-specific pended read protocol using the simple read and write commands already included in the basic AlphaServer 8000 primary protocol. This special superset protocol works as follows:

- When a microprocessor issues a PCI programmed I/O read, the read is issued to the AlphaServer 8000 system bus as a register read. This read is pended with a unique identification number that is associated with the issuing processor by driving the identification number on the system bank number lines when the register read command is issued to the system address bus. The bank number lines are otherwise unused during register accesses. The issuing processor also sets a flag, indicating that it has issued a PCI programmed I/O read command.

- The I/O port module interfacing to the addressed PCI local bus responds to the register read by forwarding the read to the PCI, storing the processor identification number specified by the address bus bank number lines and driving "dummy data" onto the data bus in the register read's associated data slot. The value of the dummy data is irrelevant; it is ignored by all system bus modules and is typically whatever was left in the I/O ports register read buffer as a result of the last read it serviced.

- When the PCI local bus returns read data to the I/O port module, the I/O module issues a register write to a special PCI read-data-return register address on the system bus. This write is pended with the issuing processor's identification number, which was stored by the I/O port module. This identification number is again pended by driving it onto the system bank number lines as the register write command is issued to the system address bus. The PCI read data is returned in the data cycle associated with this register write.

- When a processor module identifies a register write that addresses the PCI read-data-return register address, it checks the state of its PCI read flag and compares the value driven in the system bank number lines with it unique identification number. If the PCI read flag is set and the value on the bank number lines matches the processor's identification number, then the processor completes the PCI programmed I/O read with the data

supplied by the register write.

The AlphaServer 8000 PCI programmed I/O read superset protocol allows AlphaServer 8000 systems to complete PCI programmed I/O reads without stalling system buses. Furthermore, it allows AlphaServer systems to support PCI I/O in such a way that system bus modules not participating in the superset transaction need not be alerted to the presence of special bus transactions and therefore need not contain logic that recognizes and responds to these special cases. This approach demonstrates a practical way to simplify overall system design without affecting system performance.

ALPHASERVER 8400 AND ALPHASERVER 8200 SYSTEMS

The AlphaServer 8400 and 8200 systems are the first products based on the AlphaServer 8000 platform. The AlphaServer 8200 system is an "open office"-class server (i.e., the AlphaServer 8200 can be located in any office area, for example, where photocopier machines are typically placed). It features up to six system bus modules in an industry-standard 47.5-centimeter (19-inch) rackmount cabinet. The 8200 system can support up to six 300-MHz Alpha 21164 microprocessors, 6 GB of main memory, and 108 PCI I/O slots. The AlphaServer 8400 system is an "enterprise"-class server (i.e., a machine on which a business can be run). It features up to nine system bus modules in a DEC 7000-style cabinet. It can support up to twelve 300-MHz Alpha 21164 microprocessors, 14 GB of main memory, and 144 PCI I/O slots.

The clock frequencies of both the AlphaServer 8400 system and the AlphaServer 8200 system are determined by the clock frequency of the 300-MHz (3.33-ns cycle time) Alpha 21164 microprocessor chip. Both systems use a 4[X] clock multiplier to arrive at a system clock frequency of 75 MHz (13.3-ns cycle time). At this speed, the systems feature 265-ns minimum read latencies and 1,600 MB/s of data bandwidth.

Both systems are based on the same set of AlphaServer 8000 architecturally compliant system bus modules. In addition, both systems support a new PCI I/O subsystem designed specifically for these classes of systems. The constituent modules and I/O subsystems that compose the AlphaServer 8400 and the AlphaServer 8200 systems are as follows.

TLEP Processor Module--Each TLEP processor module supports two 300-MHz Alpha 21164 microprocessors. Each Alpha 21164 processor is paired with a 4-MB external cache. This cache is constructed with 10-ns asynchronous SRAMs. The cache latency to first data is 20 ns, and with one 3.33-ns processor cycle of wave pipelining, its maximum bandwidth is 915 MB/s. The TLEP module operates with a 75-MHz (13.33-ns cycle time) clock frequency.

TMEM Memory Module--Each TMEM memory module is implemented with two equal-sized DRAM banks. TMEM modules are available in 128-MB, 256-MB, 512-MB, 1024-MB, and 2048-MB sizes. The TMEM module is designed to operate at a 100-MHz (10-ns cycle time) clock frequency.

TIOP I/O Port Module--The TIOP module interfaces the AlphaServer 8000 system bus to four I/O channels, called "hoses." Each hose can interface to one XMI, Futurebus+, or PCI/EISA I/O subsystem. Each TIOP can support up to 400 MB/s of I/O data bandwidth and is designed to operate at a 100-MHz (10-ns cycle time) clock frequency.

ITIOP Integrated I/O Port Module--The ITIOP module interfaces the AlphaServer 8000 system bus to one hose I/O channel and one semipreconfigured PCI local bus, which is integrated onto the ITIOP module. The integrated PCI bus features one single-ended small computer systems interface (SCSI) controller, three Fast Wide Differential SCSI controllers, one NI port, and optional FDDI and NVRAM controllers. Each ITIOP can support up to 200 MB/s of I/O data bandwidth and is designed to operate at a 100-MHz (10-ns cycle time) clock frequency.

PCIA PCI I/O Subsystem--The PCIA PCI I/O subsystem consists of hose-to-PCI adapter logic and a 12-slot PCI local bus. This 12-slot bus is created from 4-slot PCI buses interfaced such that they appear as a single bus. The high slot count provides the connectivity essential in an enterprise-class server. The PCIA optimizes direct memory access (DMA) reads by means of the PCI Read_ Memory_Multiple command. The Read_Miss_Multiple command allows the PCIA to stream DMA read data from memory to the PCI bus. Consequently, the PCIA can increase DMA read bandwidth, offsetting any latency penalties that result from the AlphaServer 8000 platform's multilevel I/O architecture. The PCIA's adapter logic includes a 32K entry map RAM for converting PCI addresses (32 bits) to AlphaServer 8000 system bus addresses (40 bits). This map RAM features a five-entry, fully associative translation cache.

ALPHASERVER 8400 AND ALPHASERVER 8200 PERFORMANCE

A number of performance benchmarks have been run on the AlphaServer 8400 and AlphaServer 8200 systems. The results of some of these benchmarks are summarized in Table 3.

The AlphaServer SPECint92 and SPECfp92 ratings demonstrate outstanding performance. In both ratings, the AlphaServer 8400 system performance is over 3.5 times the ratings of the HP9000-800 T500 system. The SPECfp92 rating of 512 instructions per second is 1.4 times its nearest competitor, the SGI Power Challenge XL system. Similarly, a six-processor AlphaServer 8400 system achieves the same 1,900 million floating-point operations per second (MFLOPS) as an eight-processor SGI Power Challenge XL

system. Finally, the AlphaServer 8400 system's 5-GFLOPS Linpack NxN result is beyond the performance of all other open systems servers, placing the AlphaServer at supercomputer performance levels with systems such as the NEC SX-3/22 system and the massively parallel Thinking Machines CM-200 system.

Table 3 AlphaServer 8400 and 8200 System Performance Benchmark Results

Benchmark Name	Processor Count	Units	AlphaServer 8200	AlphaServer 8400
SPECint92	1	Instructions/ second	341.4	341.4
SPECfp92	1	Instructions/ second	512.9	512.9
SPECrate_int92	1	Instructions/ second	8551	8551
	6	Instructions/ second	50788	50788
	12	Instructions/ second	not applicable	91580
SPECrate_fp92	1	Instructions/ second	11981	11981
	6	Instructions/ second	71286	71286
	12	Instructions second	not applicable	140571
Linpack 100x100	1	MFLOPS	140.3	140.3
Linpack 1000x1000	1	MFLOPS	410.5	410.5
	6	MFLOPS	1821	1902
	8	MFLOPS	not applicable	2282
	12	MFLOPS	not applicable	2675
Linpack NxN	1	MFLOPS	428.3	428.3
	6	MFLOPS	2445	2445

	12	GFLOPS	not applicable	5.0
AIM III Performance Rating	8	AIMs	not applicable	1649.8
AIM III User Loads	8	Maximum quantity	not applicable	9384
AIM III Throughput	8	Jobs/min	not applicable	16168.2
McCalpin Copy	1	MB/s	not available	186.29
	8	MB/s	not applicable	898.61
McCalpin Scale	1	MB/s	not available	174.4
	8	MB/s	not applicable	829.74
McCalpin Sum	1	MB/s	not available	198.3
	8	MB/s	not applicable	891.84
McCalpin Triad	1	MB/s	not available	195.15
	8	MB/s	not applicable	982.13

ACKNOWLEDGMENTS

Several members of the AlphaServer 8000 Development Team in addition to the authors were key contributors to the generation of this technical article. These individuals are John Bloem, Elbert Bloom, Dick Doucette, Dave Hartwell, Rick Hetherington, Dale Keck, and Rich Watson.

REFERENCES

1. W. Bowhill et al., "Circuit Implementation of a 300-MHz, 64-bit Second-generation CMOS Alpha CPU," Digital Technical Journal, vol. 7 no.1 (1995, this issue): 100-118.

2. S. Saini and D. Bailey, "NAS Parallel Benchmarks Results 3-95," Report NAS-95-011 (Moffet Field, CA: Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, saini@nas.nasa.gov, April 1995).

3. J. Dongarra, "Performance of Various Computers Using

Standard Linear Equations Software," Document Number CS-89-85, available on the Internet from Oak Ridge National Laboratory, netlib@ornl.gov, April 13, 1995.

4. Z. Cventanovic and D. Bhandarkar, "Characterization of Alpha AXP Performance Using TP and SPEC Workloads," Proceedings of the 1994 International Symposium on Computer Architecture: 60-70.

5. J. Nicholson, "The RISC System/6000 SMP System," COMPCON '95, March 1995: 102-109.

6. L. Staley, "A New MP HW Architecture for Technical and Commercial Environments," COMPCON '95, March 1995: 129-132.

7. B. Allison and C. van Ingen, "Technical Description of the DEC 7000 and DEC 1000 AXP Family," Digital Technical Journal, vol. 4 no. 4 (Special Issue 1992): 100-110.

8. L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," Microprocessor Report, February 16, 1995: 15.

9. J. Basmaji et al., "Digital's High-performance CMOS ASIC," Digital Technical Journal, vol. 7 no. 1 (1995, this issue): 66-76.

10. R. Watson, H. Collins, and R. Iknaian, "Clock Buffer Chip with Absolute Delay Regulation Over Process and Environmental Variations," 1992 Custom Integrated Circuits Conference, paper 25.2: 1-5.

11. E. Davidson, "Delay Factors for Mainframe Computers," Proceedings of the 1991 Bipolar Circuits and Technology Meeting: 116-123.

12. D. Cox et al., "VLSI Performance Compensation for Off-Chip Drivers and Clock Generation," Proceedings of IEEE 1989 Custom Integrated Circuits Conference: 14.3.1-14.3.4.

13. D. Chengson et al., "Dynamically Tracking Clock Distribution Chip with Skew Control," 1990 Custom Integrated Circuits Conference Proceedings: 15.6.1-15.6.4.

14. M. Johnson et al., "A Variable Delay Line Phase Locked Loop for CPU--Coprocessor Synchronization," ISSCC88 Proceedings: 142-143.

BIOGRAPHIES

David M. Fenwick

Dave Fenwick is the AlphaServer 8000-series system architect. As leader of the advanced development group and of the design team, he has been responsible for definition of the product and its characteristics, and for the system implementation. Dave moved from Digital's European Engineering organization in 1985 to join the U.S.-based VAXBI program and subsequently was processor architect for the VAX 6000 vector processor. A consulting engineer, he holds 3 major U.S. patents and has 13 patent applications pending. He received an Honours Degree in Electrical and Electronic Engineering from Loughborough University of Technology, United Kingdom.

Denis J. Foley

A principal hardware engineer in the AlphaServer group, Denis is the project leader for the TLEP CPU module. He joined Digital in Clonmel, Ireland, in 1983 after receiving a bachelors degree in Electrical Engineering from University College Cork, Ireland. He has contributed to the development of several communications and computing projects. Currently, he is working on the design of a CPU module for the AlphaServer 8000 platform that is based on the next generation of the Alpha microprocessor. Denis is listed on 12 patent applications that relate to his work on the AlphaServer CPU and bus designs.

William B. Gist

Bill Gist's recent responsibility was the development of the high-performance I/O system bus circuit architecture for the AlphaServer 8000-series ASICs. A principal engineer and a member of the Server Platform Development Group, he is currently developing high-performance I/O architectures for low-cost plastic packaging technologies. Joining Digital in 1977, he began work on PDP-11 systems development and later became a member of the VAX 6000-series engineering team, focusing on clock chip development and vector processor ASIC development. Bill has a B.S. degree in Electrical Engineering from Worcester Polytechnic Institute and holds three patents for the AlphaServer 8000-series I/O circuit architecture.

Stephen R. VanDoren

In 1988, Steve VanDoren came to Digital to work with the VAX 6000 vector processor design team. He later joined an advanced

development team responsible for evaluating system technology requirements for what would become the AlphaServer 8000 series of products. During the AlphaServer project, he lead the design of the address interface on the TLEP processor module. He is listed as a coinventor on 10 patents filed on the AlphaServer 8000-series architectural features. Steve is currently working on new server processor designs. He is a member of Eta Kappa Nu and Tau Beta Pi and holds a B.S. degree in Computer Systems Engineering from the University of Massachusetts.

Daniel Wissell

Consulting engineer Dan Wissell has more than 20 years of computer industry experience in analog and digital circuit design and test. While at Digital, he has worked on the VAXcluster and DEC 7000/1000 systems development teams, and more recently he contributed to the AlphaServer 8000-series design effort. He is recognized within Digital as an expert in the areas of distributed power systems, on-module energy management, and high-speed clock systems. Dan holds three patents and has filed several patent applications for his work on current and future Digital products. He has degrees in engineering from Kean College and the Milwaukee School of Engineering.

TRADEMARKS

Challenge is a trademark of Silicon Graphics, Inc.

Cyrix is a trademark of Cyrix Corporation.

Hewlett-Packard is a registered trademark of Hewlett-Packard Company.

IBM is a registered trademark of International Business Machines Corporation.

Intel and Pentium are trademarks of Intel Corporation.

SPECfp, SPECint, and SPECmark are registered trademarks of the Standard Performance Evaluation Council.