# Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor

by John H. Edmondson, Paul I. Rubinfeld, Peter J. Bannon, Bradley J. Benschneider, Debra Bernstein, Ruben W. Castelino, Elizabeth M. Cooper, Daniel E. Dever, Dale R. Donchin Timothy C. Fischer, Anil K. Jain, Shekhar Mehta, Jeanne E. Meyer, Ronald P. Preston, Vidya Rajagopalan, Chandrasekhara Somanathan, Scott A. Taylor, and  Gilbert M. Wolrich

## ABSTRACT

A new CMOS microprocessor, the Alpha 21164, reaches 1,200 mips/600 MFLOPS (peak performance). This new implementation of the Alpha architecture achieves SPECint92/SPECfp92 performance of 345/505 (estimated). At these performance levels, the Alpha 21164 has delivered the highest performance of any commercially available microprocessor in the world as of January 1995. It contains a quad-issue, superscalar instruction unit; two 64-bit integer execution pipelines; two 64-bit floating-point execution pipelines; and a high-performance memory subsystem with multiprocessor-coherent write-back caches.

## OVERVIEW OF THE ALPHA 21164

The Alpha 21164 microprocessor is now a product of Digital Semiconductor. The chip is the second completely new microprocessor to implement the Alpha instruction set architecture. It was designed in Digital's 0.5-micrometer (um) complementary metal-oxide semiconductor (CMOS) process. First silicon was powered on in February 1994; the part has been commercially available since January 1995. At SPECint92/SPECfp92 ratings of 345/505 (estimated), the Alpha 21164 achieved new heights of performance.

The performance of this new implementation results from aggressive circuit design using the latest 0.5-um CMOS technology and significant architectural improvements over the first Alpha implementation.[1] The chip is designed to operate at 300 MHz, an operating frequency 10 percent faster than the previous implementation (the DECchip 21064 chip) would have if it were scaled into the new 0.5-um CMOS technology.[2] Relative to the previous implementation, the key improvements in machine organization are a doubling of the superscalar dimension to four-way superscalar instruction issue; reduction of many operational latencies, including the latency in the primary data cache; a memory subsystem that does not block other operations after a cache miss; and a large, on-chip, second-level, write-back cache.

The 21164 microprocessor implements the Alpha instruction set architecture. It runs existing Alpha programs without modification. It supports a 43-bit virtual address and a 40-bit physical address. The page size is 8 kilobytes (KB).

In the following sections, we describe the five functional units of the Alpha 21164 microprocessor and relate some of the design decisions that improved the performance of the microprocessor. First, we give an overview of the chip's internal organization and pipeline layout.

### Internal Organization

Figure 1 shows a block diagram of the chip's five functional units: the instruction unit, the integer function unit, the floating-point unit, the memory unit, and the cache control and bus interface unit (called the C-box). The three on-chip caches are also shown. The instruction cache and data cache are primary, direct-mapped caches. They are backed by the second-level cache, which is a set-associative cache that holds instructions and data.

[Figure 1 (Five Functional Units on the Alpha 21164 Microprocessor) is not available in ASCII format.]

### Alpha 21164 Pipeline

The Alpha 21164 pipeline length is 7 stages for integer execution, 9 stages for floating-point execution, and as many as 12 stages for on-chip memory instruction execution. Additional stages are required for off-chip memory instruction execution. Figure 2 depicts the pipeline for integer, floating-point, and memory operations.

[Figure 2 (Alpha 21164 Pipeline Stages) is not available in ASCII format.]

## INSTRUCTION UNIT

The instruction unit contains an 8-KB, direct-mapped instruction cache, an instruction prefetcher and associated refill buffer, branch prediction logic, and an instruction translation buffer (ITB).

The instruction unit fetches and decodes instructions from the instruction cache and dispatches them to the appropriate function units after resolving all register and function-unit conflicts. It controls program flow and all aspects of exception, trap, and interrupt handling. In addition, it manages pipeline control for the integer and floating-point units, controlling all data bypasses and register file writes.

The instruction cache has 32-byte blocks. The cache tags hold virtual address information. Its tags also support PALcode through a bit which indicates that the tag contains a physical address. (PAL stands for privileged architecture library and refers to physically addressed code executed in a privileged hardware mode that implements an architecturally defined interface between the operating system and the hardware.)

## Instruction Pipeline

The first four pipeline stages of the Alpha 21164 microprocessor are the instruction unit pipeline stages, stage 0 through stage 3. The logic in the stage before stage 0 is normally considered to operate in stage 1 of the pipeline. In that stage, the new instruction cache address is calculated either by incrementing the previous address or by selecting a new address in response to a predicted or actual flow change operation.

During stage 0, the 8-KB instruction cache is accessed. It returns a naturally aligned block of four instructions (16 bytes) with 20 bits of previously decoded instruction information (5 bits per instruction). The precalculated decode information is used in stage 1 for branch and jump processing and in stage 2 for instruction slotting.

In stage 1, the four-instruction block is copied into one entry of the two-entry instruction buffer (IB). Also in stage 1, the instruction cache and ITB each check for hits, and the branch-and-jump prediction logic determines new fetch addresses.

The main function of stage 2 is steering each instruction to an appropriate function unit. This process, called instruction slotting, resolves all static execution conflicts. The instruction slotter accepts the next four-instruction block from the IB into a staging register at the beginning of stage 2 and routes the individual instructions to the appropriate functional pipelines as it advances them to stage 3. If the block contains certain mixes of instruction types, it is able to slot all four instructions in a single cycle. Otherwise, it advances as many instructions as possible in the first cycle.  The remaining instructions in the block are slotted during subsequent cycles. Instructions are slotted strictly in program order. A new four-instruction block enters stage 2 when every instruction in the prior block has been slotted and advanced to stage 3.

The issue stage operates in stage 3. It performs all dynamic conflict checks on the set of instructions advanced from stage 2. The issue stage contains a complex register scoreboard to check for read-after-write and write-after-write register conflicts. This stage also detects function-unit-busy conflicts, which can occur because the integer multiplier and floating-point divider are not fully pipelined. The register scoreboard logic detects all integer and floating-point operand bypass cases and sends the necessary bypass control signals.

The issue stage issues instructions to the appropriate function units unless it encounters a dynamic conflict. If a conflict occurs, the instruction and logically subsequent instructions are stalled (not issued). A stall in stage 3 also stalls the advance of the next set of slotted instructions from stage 2. This stall ends when all instructions in stage 3 have been issued.

To perform conflict checking and to handle exceptions (including traps and interrupts), the instruction unit tracks the instructions issued during stage 4 through stage 8. The instruction unit sends register file write strobes and addresses to the integer and floating-point register files for instructions that reach the retire point (stage 6) without an exception. In the event of an exception, write strobes are withheld (gated) to prevent incomplete instructions from updating the register file. These instructions do not complete either because they caused an exception or because they are in the "shadow" of an exception. The shadow of an exception includes all instructions that are in the pipeline when an exception is recognized but are logically subsequent to the instruction taking the exception.

The issue stage stalls for a single cycle to permit the integer multiplier or floating-point divider to return a result into its associated pipeline. This is necessary because the register files do not have extra write ports dedicated to receiving these results. The issue stage also stalls for one cycle in similar cases to permit data fills for load instructions that missed in the data cache to write to the register file and data cache. The issue stage stalls indefinitely when necessary to execute the trap barrier and memory barrier instructions.

## No-op Instructions

New instructions are shifted into the slotting and issue stages when a given stage becomes completely empty. Compared to an ideal design in which instructions are shifted to fill a given stage partially, this design has a slightly increased average cycles-per-instruction ratio. We considered the alternative in which instructions are shifted in as slots become available. This alternative would have created critical paths that would increase the CPU cycle time by approximately 10 percent. An evaluation of our trace-driven performance model showed that the alternative did not reduce the cycles-per-instruction ratio enough to compensate for the reduction in cycle time. As a result, we chose the simpler and faster design.

Compilers and assembly language programmers can insert no-op instructions to minimize and, in most cases, to eliminate any negative performance effect. To facilitate this process, the Alpha 21164 microprocessor handles three different kinds of no-op instruction.

The first two kinds of no-op instruction are the integer no-op

(NOP) and the floating-point no-op (FNOP). NOP (BIS R31,R31,R31) can issue in either integer execution pipeline. FNOP (CPYS F31,F31,F31) can issue in either floating-point execution pipeline. The compiler uses these to improve performance when two instructions would be slotted together even though they cannot issue in the same cycle. If one instruction in a pair is dependent on the other, issuing them together guarantees the second will stall in the issue stage and prevent later instructions from entering that stage. The compiler inserts a NOP or FNOP to delay the issue of the second instruction. With this improvement, the second instruction can be issued with later instructions.

The third kind of no-op instruction, the universal no-op (UNOP), is detected in stage 2. UNOP [LDQ_U R31,0(Rnn)] is discarded in stage 2 so that it does not require an issue slot in either pipeline. UNOP allows compilers to align instructions without the unnecessary use of pipeline issue slots. For example, the compiler can align the target of a branch without necessarily slowing execution of the fall-through path to that branch.

## Instruction Prefetcher and Refill Buffer

The instruction prefetcher operates in parallel with the instruction cache. When an instruction is not in either the instruction cache or refill buffer, the prefetcher generates a stream of 32-byte instruction block fetch requests to fill the 4-entry refill buffer with instruction data. Each instruction block contains 8 instructions. Fetched instruction data is stored in the refill buffer when it is returned. Four-instruction subblocks of instruction data are moved from the refill buffer to the IB when needed. At that time, the instruction cache is also updated. If this data movement empties an entry in the refill buffer, an additional fetch request is initiated. Fetched instruction data is buffered in the refill buffer rather than the instruction cache to avoid evicting valid cache blocks unnecessarily.

The refill buffer is a type of stream buffer. Each entry stores a virtual address and has a comparator so the refill buffer can be probed for instruction data on a cache miss. Instruction fetching begins only if an access misses in both the instruction cache and the refill buffer. Fetching stops when any instruction flow change occurs (i.e., branch, jump, exception, etc.). It also stops if at any time the instructions needed in stage 1 are found in the instruction cache.

The combination of the on-chip, 96-KB second-level cache and the instruction prefetcher significantly reduces the benefit of enlarging the instruction cache beyond its current size of 8 KB. The prefetcher generates requests at a high rate. Because it is on-chip, the second-level cache has the bandwidth to handle requests quickly and with relatively little effect on data-stream requests. In general, the performance benefit from making the instruction cache larger is very small. This is one of the benefits of the two-level on-chip cache hierarchy.

## Instruction Stream Address Translation and the Instruction Translation Buffer

The instruction unit contains a 48-entry, fully associative instruction translation buffer (ITB) that holds instruction stream address translations and protection information. Each entry in the ITB can map 1, 8, 64, or 512 contiguous 8-KB pages.

During stage 1, the ITB entries are checked for a match with the program counter (PC). If the page is found, its protection bits are checked against the current operating mode. If the page is not found, an ITB miss trap occurs. If the page is found in the ITB and the access is an instruction cache miss, the ITB supplies the physical page address to the prefetcher.

## Branch and Jump Prediction

The branch prediction logic examines the block of instructions coming from the instruction cache or refill buffer during stage 1. It checks the block for control instructions (taken conditional branches, jumps, subroutine return instructions, and other flow-change instructions) and calculates the new fetch address. Since the new fetch address is available at the end of stage 1, the read of the instruction cache for the target instruction occurs in the next cycle. This means the control instruction is in stage 2 at the same time as the target instruction is in stage 0, resulting in a one-cycle branch delay that creates an empty cycle in the pipeline. The IB quashes this empty cycle if any stall occurs ahead of it in the pipeline.

The branch prediction logic predicts conditional branch instructions using a branch history table with 2K entries addressed by low-order bits of the PC. Each is a two-bit counter that increments when branches are taken and decrements when branches are not taken. The counter saturates at the top and bottom counts. A branch is predicted to be taken if the current counter value is one of the two highest counts; otherwise, it is predicted to be not-taken. This method is more effective than the method used in the first Alpha microprocessor (which had only one bit of history per entry), partly because it reduces the misprediction rate for typical loop branches by half.

A 12-entry return address stack is used to predict the target address on subroutine returns (i.e., RET, JSR_COROUTINE) and returns from PALcode. Each entry stores 11 bits of address, which is sufficient to address the 8-KB instruction cache. The upper 32 bits of the target address are predicted by using the value in the instruction cache tag that is addressed by the return address stack. The same basic mechanism is used to predict the full target address of jump and jump-type subroutine call instructions since the Alpha architecture provides a hint field in these instructions that indicates the target cache address.

The Alpha 21164 microprocessor recovers from incorrect branch and PC predictions by taking a mispredict trap when the incorrectly predicted branch or jump-type instruction executes in the execution unit. For a typical branch misprediction, the execution time is five cycles longer.

### Replay Traps

In a replay trap, the instruction unit prevents completion of a given instruction by trapping the instruction and then restarting execution immediately with that instruction. The trap mechanism prevents completion of subsequent instructions. This mechanism replays the instruction from the beginning of the Alpha 21164 pipeline. It is used when a stall after stage 3 would otherwise be required.

There are three main reasons stalls are not implemented for stages later than stage 3. The ability to stall adds complexity to clocking circuits, particularly in execution unit data paths. In addition, it adds control complexity. An example of this is a stalled two-input function unit in which one input operand is invalid. To end the stall, certain latches must be enabled while others are not, because the valid data must be held in one pipeline latch while the invalid data is replaced in another. Finally, adding stall logic would create additional critical paths. The elimination of stalls beyond stage 3 and the use of the replay trap mechanism avoid these complexities.

The replay trap mechanism is used for a number of unusual memory instruction conflicts and memory unit resource overruns. For example, the load-miss-and-use replay trap is used when a load misses in the data cache and a dependent instruction issues exactly two cycles after the load. The issue decision for such a dependent instruction is made prior to the actual determination of cache hit, so a hit is predicted. If this prediction is wrong, the dependent instruction is restarted from the front of the pipeline and will arrive at the issue stage one cycle before data arrives from the second-level cache. Because the instruction arrives before the data, there is no performance loss due to the trap mechanism.

## INTEGER FUNCTION UNIT

The integer function unit executes integer operate instructions, calculates virtual addresses for all load and store instructions, and executes all control instructions except floating-point conditional branches. It includes the register file and several integer functional subunits, most of which are contained in two parallel four-stage pipelines. Both pipelines contain an adder and a Boolean logic unit. The first pipeline contains the shifter, and the second pipeline contains the control instruction execution unit. The first pipeline also attaches to the partially pipelined integer multiplier, which operates in the background. Except for the issue cycle and a cycle to return the result, the first pipeline and integer multiplier operate in parallel.

## Integer Register File and Bypasses

The integer register file is read during stage 3 and written in stage 6. Bypass paths are implemented to allow all subunits other than the multiplier to receive and use the result of a previous instruction from stage 4, 5, or 6 of either pipeline. Due to implementation constraints, the multiplier can only receive bypassed data from stage 6 of the pipeline. This increases multiply latency by as many as two cycles when multiply input operands are produced by preceding integer operate instructions.

The integer register file contains 40 registers: the 32 integer registers specified by the architecture (R0 through R31) with R31 always reading as 0; and 8 shadow registers available to PALcode as scratch space. The register file is accessed by 4 read ports (2 for each pipeline) and 2 write ports (1 for each pipeline).

## Instruction Latencies

Most instructions executed in the integer function unit have a latency of 1 cycle. These instructions execute in stage 4. The conditional move instruction has a latency of 2 cycles. It executes in stage 4 and stage 5.

Multiply latency depends on the data size and the operation being performed. Thirty-two-bit multiplies have an 8-cycle latency, and the multiplier can start a second multiply after 4 cycles, provided that the second multiply has no data dependency on the first. Sixty-four-bit signed multiplies have a 12-cycle latency; the 64-bit multiply unsigned high instruction has a 14-cycle latency; and for both of these 64-bit multiplies, the multiplier can start a nondependent multiply after 8 cycles.

Because of a special bypass, compare and Boolean logic instructions can have a latency of 0 cycles when a conditional move or a branch test input operand is the result of an immediately preceding compare or Boolean logic instruction. The integer unit uses the bypass to allow dual issue of the producer and consumer in this case.

To realize the full benefit from the increased issue width relative to the first Alpha microprocessor, the DECchip 21064, it is critical to reduce operational latencies. As the issue width increases, the cost in instruction execution opportunities for a given latency increases. In the integer unit, the following latencies are reduced relative to the 21064: the shifter latency (from 2 cycles to 1), the byte and word operation latencies (from 2 cycles to 1), and the multiplier latency (from 19 to 23 cycles in the 21064 to 8 to 16 cycles in the Alpha 21164). Also the special bypass for conditional instructions reduces that latency from 1 cycle in the 21064 to 0 cycles in the Alpha 21164. For the most part, these latency reductions are achieved by circuit design improvements.

### Integer Load and Store Instructions

Integer load instructions issue in either pipeline and as many as two can issue per cycle. Integer store instructions issue in the first pipeline only. For integer load instructions that hit in the data cache, the data is multiplexed into the output of stage 5 of the pipeline in which the load issued; the data is then written to the register file through the write port associated with that pipeline. For integer load instructions that miss in the data cache, the data is returned later by the memory subsystem. The data is then multiplexed into the output of stage 5 as before, and the instruction unit inserts a properly timed NOP cycle by stalling the issue stage for one cycle to make the pipeline's register write port available.

## FLOATING-POINT UNIT

The floating-point unit consists of the floating-point register file and two pipelined functional subunits: an add pipeline that executes all floating-point instructions except for multiply, and a multiply pipeline that executes floating-point multiplies. All IEEE and VAX rounding modes are done in hardware, including IEEE round to plus and minus infinity.

### Pipeline Structure and Operation Latencies

Each floating-point subunit on the Alpha 21164 CPU chip contains three functional stages implemented in four pipeline stages, stage 5 through stage 8. The floating-point register file is read in stage 4 and written at the end of stage 8. Figure 3 depicts the physical layout of the floating-point unit.  Figure 4 shows the pipelining of instructions executed in the floating-point unit.

[Figure 3 (Physical Layout of the Floating-point Unit)
is not available in ASCII format.]

As in the integer unit, latency is reduced in the floating-point unit relative to the previous Alpha implementation. The latency of all floating-point operate instructions, except floating-point divide, is 4 cycles. In the DECchip 21064, most floating-point operations take 6 cycles. The floating-point divide latency varies depending on the input data values. For a single-precision divide, the latency is reduced from 34 cycles in the 21064 to an average of 19 in the 21164; and for a double-precision divide, it is reduced from 63 cycles to an average of 31. As discussed previously, reducing latency is important as issue width increases. As in the integer unit, the reduced latency is achieved mostly by circuit design improvements.

## Register File and Bypasses

The floating-point register file has nine ports: two read ports and one write port per functional unit for source and destination operand accesses, one read port for floating-point stores, and two write ports to support two floating-point loads per cycle. Bypass paths forward data from each of the four write buses in the floating-point register file to each of the five read buses.

## Floating-point Load and Store Instructions

In Alpha microprocessors, floating-point numbers are stored in one format in memory and in another format in the floating-point registers. Floating-point load and store instructions convert from one format to the other as they move the data. In the Alpha 21164 pipeline, floating-point input operands are read from the floating-point register file one cycle later than integer input operands are read from the integer register file. This skew provides an extra cycle for floating-point load data format conversion.

Floating-point load and store instructions first issue to the integer unit for address calculation. The issue restrictions are exactly the same as for integer load or store instructions. For floating-point load instructions, the data is written to the register file using one of the two write ports reserved for that purpose. When a conflict for these write ports occurs between a write due to a new load that hit in the data cache and a write due to a previous load that missed, the conflict is resolved by forcing the new load to miss in the data cache.

[Figure 4 (Floating-point Unit Pipeline) is not available in ASCII format.]

## Add Pipeline

The key components of the add pipeline design are the fast fraction adder, operand data-path alignment, normalization shift detection, sticky-bit calculation, and round-adder design. The fast-adder design operates in a single phase (one phase equals one-half of a CPU cycle). It is used in the function stage 1 and stage 3 fraction adders. To reduce formatting and rounding complexity, the least significant bits in fractions are aligned to one of two different bit positions: one for single-precision data (IEEE S and VAX F) and 4-byte integers, and one for double-precision data (IEEE T, and VAX G and D) and 8-byte integers.

For effective subtracts with exponent differences of {--}1, 0, or 1, a new normalization shift detect algorithm uses three leading bit chains to examine stage 1 input operands to determine the required normalization shift. The normalization shift amount is chosen by comparing the least significant bit of one exponent to the least significant bit of the other.

The sticky bit for adds and subtracts is determined by comparing the exponent difference with an encoded value for the number of trailing zeros in the fraction being aligned.

The stage 3 round adder operates in one cycle and consists of a fraction adder and an output selector. The fraction adder takes one phase and adds two operands plus rounding bits based on the round mode. The selector assembles the fraction result based on global carry-and-propagate information from the adder. It also examines the adder result alignment and performs a final normalization shift of as much as one bit left or right. The exponent result is also selected in stage 3 before the complete result is sent to the register file write bus and bypass logic.

## Multiply Pipeline

Multiplication is done using radix-eight Booth encoding, which requires 18 partial products to be summed.[3] The first stage of the multiply pipeline is used to create three times the multiplicand and to determine the Booth encodings. The multiplier array is composed of 14 rows of carry-save adders that perform the addition of multiplicands. The carry and sum outputs of the array are reduced by combining carry-save adders and then are passed through a half adder to facilitate rounding.

The sticky bit for multiplication is determined by summing the number of trailing zeros in both operands.  The carry output from the less significant product bits is used by the round selector of the multiply pipeline to determine the correct final product.

## Divider

Floating-point divide instructions issue into the add pipeline. The operands are immediately passed to the divider. Instruction issue to the add pipeline continues while a divide is in progress until the result is ready. At that point, the issue stage in the instruction unit stalls one cycle to allow the quotient to be sent to the round adder and then be written into the register file.

The divider uses a normalizing nonrestoring algorithm that determines 1 to 4 bits of quotient per cycle, averaging 2.4 quotient bits per cycle.[4] Implementation of this algorithm requires that an exact partial remainder be produced every cycle. The implementation uses a fast adder that produces its result in half of a cycle.

## MEMORY UNIT

The memory unit contains a fully associative, 64-entry, data translation buffer (DTB); an 8-KB, direct-mapped, primary data cache; a structure called the miss address file (MAF); and a write buffer. It processes load, store, and memory barrier instructions.

The write-through data cache has 32-byte blocks and 2 read ports. Its tags hold physical address data.

The memory unit receives as many as 2 virtual addresses from the integer unit each cycle. Because it has 2 read ports, the DTB can translate both virtual addresses to physical addresses and detect memory management faults. (Like the ITB, each entry in the DTB can map 1, 8, 64, or 512 contiguous 8-KB pages.)

Load instructions access the data cache and return data to the register file if there is a hit. The latency for loads that hit in the data cache is two cycles. Again, latency is reduced relative to the DECchip 21064 microprocessor where the latency is three cycles for loads that hit. The reduced latency was achieved by circuit design improvements. Reducing this latency is particularly important as issue width increases because of the frequent use of loads in programs.

For loads that miss, the physical addresses are sent to the MAF, where they wait to be sent to the C-box. Store instructions write the data cache if there is a hit; they are always placed in the write buffer, where they wait to be sent to the C-box.

### Memory Unit Pipeline Structure

Virtual address calculation begins in the integer unit early in stage 4. The data cache access begins later in stage 4 and completes early in stage 5. Address translation is done in parallel with data cache access. Data cache hit is determined late in stage 5. If the access hits, the data is written to the register file (for a load access) or the cache (for a store access) in stage 6. In the case of a data cache miss, the memory access advances to pipeline stages in the C-box.

### Miss Address File

The MAF consists of two sections that store data. The first section holds load misses (called DREADs) in six entries, and the other section holds instruction fetch addresses (called IREFs) in four entries. For DREADs, the MAF stores the physical address, destination register, and instruction type (integer/floating-point, 4-byte/8-byte/IEEE-S-Type/VAX-G-Type, etc.). For IREFs, the MAF stores only the physical address.

Buffered accesses in the MAF and write buffer are sent to the C-box at a peak rate of one every other cycle. DREADs have highest priority, writes have the next highest priority, and IREFs have lowest priority.

When the C-box returns data for a DREAD, the memory unit provides the destination register and instruction type information from the MAF. This information is then used to convert the data to its in-register format, to determine which registers to write, and to update the register scoreboard in the instruction unit. The DREAD entry is removed from the MAF when the second half of the data fill arrives.

The C-box returns IREF data directly to the instruction unit's cache and refill buffer. The IREF entry is removed from the MAF as soon as the command has been accepted by the C-box.

Merging Capability.  One key performance feature of the MAF is that it merges multiple load misses that access the same 32-byte block of memory into a single C-box DREAD request. One load instruction requests at most 8 bytes of a 32-byte memory block. As many as 4 load misses can be merged into 1 DREAD request. This improves latency and reduces unnecessary bandwidth consumption in the second-level cache.

To implement merging, the MAF merge logic detects any load miss address to a block that has already been queued in the DREAD section of the MAF. The logic then adds the new destination register to the existing request. Merging is limited to 1 load miss per naturally aligned 8-byte portion of the 32-byte block. Also, merging is permitted only for load misses with identical instruction types. The memory unit allocates a new DREAD entry in the MAF only for load misses that do not merge. The merge logic supports the peak load instruction issue rate. It can merge as many as 2 load misses per cycle into the DREAD section and can merge loads that issue together.

The MAF merge capability is an integral part of the two-level cache hierarchy design. It can reduce the rate of memory read operations from two loads per cycle in the integer pipelines to one read every other cycle in the second-level cache pipeline. By doing so, the MAF makes the full bandwidth of the second-level cache available to the program.

The MAF can hold as many as 6 DREADs that can represent as many as 21 loads. (The theoretical maximum is 24 loads; this limit is a by-product of the overflow prevention algorithm.) Requests are sent to the C-box in the order in which they were allocated in the MAF. Accesses in the second-level cache can hit underneath (behind) second-level cache misses, allowing data fills to be returned in a different order than they were sent to the C-box.

Two-level Data Cache.  Many workloads benefit more from a reduced latency in the data cache than from a large data cache. We considered a single-level design for a large data cache. For circuit reasons, physically large caches are slower than small caches. To achieve a reduced latency, we chose a fast primary cache backed by a large second-level cache. As a result, the effective latency of reads is better in the Alpha 21164 CPU chip than it would have been in a single-level design.

The two-level data cache has other benefits. The two-level design makes it reasonable to implement set associativity in the second-level cache. Set associativity enables power reduction by making data set access conditional on a hit in that set. The two-level design also allows the second-level cache to hold instructions, which makes a larger instruction cache unnecessary.

In addition, the two-level design was simpler. Because performance studies showed that the Alpha 21164 CPU chip should have write-back caching on-chip, the data cache in the single-level design would have been write-back. Also, because of its larger size, it would have been virtually addressed, which would have required a solution to the synonym problem. Finally, it would have been difficult to make the single large cache set-associative without adding latency. The two-level design eliminated all these issues.

## Write Buffer

The write buffer contains 6 entries; each entry holds as many as 32 bytes of data and one physical address. It accumulates store instructions written to the same 32-byte block by merging them into 1 entry. It can merge 1 store instruction per cycle, matching the peak store instruction issue rate. The write buffer places no restrictions on merging until a write is sent to the second-level cache. At that time, the write buffer stops merging to that entry.

Once an entry from the write buffer has been sent to the C-box, several steps may be required to complete the write, depending on the presence of the memory block in the second-level cache and its cache coherence state. The C-box signals the memory unit upon completion of a store operation, and then the memory unit removes the corresponding entry from the write buffer.

## Access Ordering

The memory unit guarantees that all memory accesses to the same address are processed in the order given by the instruction stream. This is a design problem in any nonblocking memory subsystem design. Load misses that conflict with a store, and stores that conflict with a load miss, set conflict bits that prevent the issue of the DREAD or write until all conflicts have been cleared. If a store matches a valid entry in the write buffer and cannot merge with that entry, it is allocated a new entry that is prevented from being sent to the C-box until the earlier write is completed.

### Memory Barrier Instructions

The memory unit implements the memory barrier (MB) instruction by retiring all previous load misses and writes before sending the MB to the bus interface unit. The instruction unit stalls new memory instructions until the MB has been completed.

The memory unit implements the write memory barrier (WMB) instruction as follows:  When the WMB is executed, the memory unit marks the last write that is pending at that time. Writes added after that time are added behind the WMB mark. They are not sent to the C-box until all writes ahead of the WMB mark are completed. Unlike the MB instruction, execution of the WMB instruction does not require any stalls in the instruction unit.

### Replay Traps in the Memory Unit

The memory unit forces a replay trap if a new load or write would cause the buffer to overflow. It also forces a replay trap when a store that hits in the data cache is followed by a load to exactly the same location in the next cycle. In this case, because the store writes the data cache in stage 6, the data from the store would not yet be available to the load.

## CACHE CONTROL AND BUS INTERFACE UNIT

The cache control and bus interface unit or C-box contains the second-level cache and the following subunits: the second-level cache arbiter unit (SAU), the bus interface unit sequencer (BSQ), the victim address file (VAF), the bus interface unit address file (BAF), the write buffer unit (WBU), and the system probe arbiter (SPA). Figure 5 shows the functional units of the C-box.

[Figure 5 (Functional Units of the C-box) is not available in ASCII format.]

The C-box provides the interface to the system for access to memory and I/O. It provides full support for multiprocessor systems using a cache coherence protocol (described later in this section). It manages the second-level cache and an optional off-chip third-level cache, both of which are multiprocessor-coherent write-back caches.

The SAU arbitrates the requests for access to the second-level cache. The BSQ requests to write data fill (due to previous second-level cache misses). The VAF requests read accesses of deallocated second-level cache blocks that have been modified (called victims). The SPA requests access for external cache coherence transactions. The memory unit requests access for DREAD, IREF, and write requests. Highest priority is given to the BSQ, followed by the VAF, and then the SPA; lowest priority is given to the memory unit.

The BSQ controls data movement to and from the Alpha 21164 microprocessor. It accesses the optional off-chip third-level cache. It communicates with the system to request data that is not cached, to write back deallocated cache blocks that have been modified, to carry out coherence transactions, and to perform I/O accesses.

The VAF reads and holds victims from the second-level cache and data for memory broadcast writes, I/O writes, and external cache coherence commands that require data from the second-level cache. It has two entries for victims, each of which holds the address and data for a victim. These victims are written back to third-level cache or memory when the BSQ is idle or sooner if necessary to maintain cache coherence. These entries also hold data for memory broadcast writes and I/O writes. A separate buffer holds data for external cache coherence commands that require data from the second-level cache.

The WBU handles second-level cache writes and cooperates with other C-box subunits to maintain cache coherence.

The SPA receives cache coherence requests from the external system environment. To fulfill these coherence requests, it accesses the second-level cache and, if the off-chip cache is present, cooperates with the BSQ to access the off-chip cache. It then sends an appropriate response to the external system.

## Second-level Cache and Optional Off-chip Cache

The C-box manages the on-chip second-level cache and the optional off-chip cache. Both are write-back, and both are mixed instruction and data caches. If it is present, the off-chip cache is a third-level cache. The second-level cache is 96 KB in size and is 3-way set-associative. The off-chip cache is direct-mapped and can be configured to sizes ranging from 1 megabyte (MB) to 64 MB. The off-chip cache is not set-associative because it is not feasible given pin-count constraints. The tags in both caches hold physical address data and coherence state bits for each block.

The block size for the off-chip cache is configurable to 32 bytes or 64 bytes. The second-level cache has 1 tag per 64-byte block. It can be configured to operate with 64-byte blocks or with 32-byte subblocks.

The second-level cache tags contain bits to record which 16-byte data words within the block or subblock have been modified since the block was brought on-chip. When a block or subblock is copied back to the off-chip cache, only modified 16-byte data words are transferred. This reduces the time required to write back second-level cache victims in many cases.

## Transaction Handling

A maximum of 2 second-level cache misses can be queued in the BAF for external access in the off-chip cache and memory. The BAF merges read requests to 32-byte blocks within the same 64-byte block.

For simplicity, only one operation to a given second-level cache address is allowed in the BAF at a time, except when the two requests merge. A new request with a second-level cache address that matches an existing request in the BAF is aborted. Similarly, requests that require VAF entries when the VAF is full are aborted, and new requests are aborted when the BAF is full. If a request is aborted, the memory unit retries the request repeatedly until it is accepted. Accesses to second-level blocks that are partially valid because they are being filled are aborted repeatedly until the data fill completes.

## Maintaining Cache Coherence

The Alpha 21164 CPU chip uses a cache coherence protocol implemented in hardware to provide full support for multiprocessor systems. The instruction cache is virtual and is not kept coherent by the hardware. (The Alpha architecture requires software to manage instruction cache coherence.) The data cache is a subset of the second-level cache. If the off-chip cache is present, then the second-level cache is a subset of the off-chip cache.

Three state bits record the coherence state of each block or subblock in the second-level cache and the off-chip cache: the valid bit, the shared bit, and the dirty bit. The valid bit indicates that the block contains valid data. The shared bit indicates that the block may be cached in more than one CPU's cache. The dirty bit indicates that the memory copy of the block is not correct and the cache block must eventually be written back. These state bits allow the following states to be encoded for a given cache block or subblock: invalid, exclusive-unmodified, exclusive-modified, shared-unmodified, and shared-modified.

The system bus interface is the coherence reference point in the system. Any request to modify the state of a block is arbitrated at this bus before the block is changed. For example, when the Alpha 21164 CPU chip must write to a block in the second-level cache that is in the exclusive-unmodified state, the BSQ sends a request to the system to change the state of the block to the exclusive-modified state. The C-box waits for the system to acknowledge the request, and then retries the write. If another processor reads the same block before the request is acknowledged, the block is instead changed to the shared-unmodified state. In that situation, the Alpha 21164 CPU chip subsequently sends a full-block memory write on the system bus that causes all other processors to invalidate their copy of the block and leaves the block in the exclusive-unmodified state in this processor.

## Second-level Cache Transaction Flows

DREADs, IREFs, and writes from the memory unit access the second-level cache after winning arbitration in the memory unit and the SAU. The second-level cache is fully pipelined. Figure 6 shows an example of a read that is followed by a write as both hit in the cache.

For the read access shown in Figure 6, the pipeline stages are the following. The SAU arbitrates in stage 5; the second-level cache tag store is read in stage 6; the hit is determined in stage 7; and the requested data is read from the cache data store in stage 8 and sent on the 128-bit-wide read data bus (R-bus) in stage 9. The second half of the 32-byte block is read and sent in the next pipeline cycle. The R-bus data is received by the integer unit, the floating-point unit, or the instruction unit, depending on the access type.

[Figure 6 (Second-level Cache Read/Write Flow) is not available in ASCII format.]

For data returned to the integer unit or the floating-point unit, the data cache fill begins in stage 10 and completes in stage 11. The register file write occurs in stage 11. An instruction that is dependent on the load can begin execution in the next cycle In this case, the load latency is eight cycles.

For the write access shown in Figure 6, the pipeline stages are the following. The SAU arbitrates in stage 5; the tag store is read in stage 6; the hit is determined, and data is sent on the 128-bit write data bus (W-bus) in stage 7; and the cache is written in stage 8. As before, the second half of the 32-byte write occurs in the next pipeline cycle.

A second-level cache miss that results in a victim provides an interesting case for discussion. Here, we must determine which set to fill and then remove the victim before data can be returned from the off-chip cache. Figure 7 shows an example of a DREAD that misses in the second-level cache, creating a victim, and then hits in the off-chip cache. The example shown is the fastest possible. In this case, the BSQ is idle so the BAF is bypassed and the address is sent immediately to the off-chip cache. The access time for the off-chip cache is four CPU cycles.

As shown in Figure 7, the DREAD wins arbitration in stage 5, and the miss is detected in stage 7. The set picked by the random replacement algorithm contains modified data (a victim). Since the block size in the second-level cache is 64 bytes, two 32-byte victim read sequences are needed to copy the entire victim into the on-chip victim buffer. The two victim reads arbitrate at high priority to ensure that the victim is copied before the data fills from the off-chip cache overwrite the locations.

[Figure 7 (Second-level Cache Miss Sequence with Fastest Fill Possible) is not available in ASCII format.]

The Alpha 21164 CPU chip begins sending the off-chip cache address in stage 8 (because of BAF bypass, as described above). The tag and data are clocked into the Alpha 21164 chip at the beginning of stage 12. The BSQ arbitrates speculatively for a single cycle on the second-level cache pipeline to reserve a cycle on the R-bus. That cycle is used to send the data from the off-chip cache to the execution units and data cache.

If the access hits in the off-chip cache, the BSQ arbitrates to fill the second-level cache. The fill transaction takes a single cycle in the pipeline to write the tag store in stage 6 and the data store in stage 8.

The second victim read sequence occurs after the first data fill. Because of this, the first victim read sequence always reads the data location overwritten by the first data fill.

## PALcode

The Alpha architecture defines the privileged architecture library code (PALcode) as a set of software routines that interface an operating system to a specific Alpha implementation. PALcode presents the operating system with an architecturally defined interface that is the same in all implementations even though the underlying hardware designs can be very different. PALcode currently exists to interface the Alpha 21164 microprocessor to the Windows NT, Digital UNIX (formerly DEC OSF/1), and OpenVMS operating systems.

When the processor is executing PALcode, it is in PAL mode. PAL mode is entered upon execution of the CALLPAL instruction and upon the occurrence of interrupts, exceptions, and certain kinds of traps. The PALcode entry point is a hardware dispatch to a location that is determined by the entering event. In PAL mode, instructions are fetched from physical memory without address translation. Also, five PAL support instructions are enabled that give access to all hardware registers and special load/store access to virtual and physical memory. PAL mode is exited by executing a PAL instruction called HW_REI.

To meet performance goals, a number of PAL features are included in the Alpha 21164 microprocessor. For example, the integer register file contains eight shadow registers that map over R8 through R14 and R25 in PAL mode. Although this overmapping is normally enabled in PAL mode, it can be disabled through a hardware control register. This speeds PALcode entry and exit, because PALcode is free to use these registers without saving and restoring state. The shadow register mapping is designed to avoid overmapping any register used to pass data from the operating system to PALcode or vice versa.

Several of the operating systems that run on Alpha systems access memory management page tables through virtual memory.[5] The Alpha 21164 microprocessor contains hardware to speed processing of the PALcode for translation buffer miss. These PALcode routines access virtually mapped page tables. The hardware calculates the virtual address of the page table entry (PTE) based on the miss address and the address of the page table base. This eliminates the instruction sequence required for this calculation. PALcode then executes a load instruction to this virtual address to fetch the required PTE. This load is performed using a PAL instruction that signals a virtual PTE fetch. If this load misses in the DTB, a special PALcode trap routine is dispatched to fill the DTB using a multilevel, physical-address access method. After that, the original virtual PTE read is restarted and will succeed.

## TESTABILITY FEATURES

The Alpha 21164 microprocessor incorporates several testability features. Some enhance chip test, and some features provide useful module test capability.[6]

### Repairable On-chip RAMs

The Alpha 21164 microprocessor requires large random-access memory (RAM) arrays for its on-chip caches. To improve yield, the instruction and data cache arrays have spare rows and the second-level cache has spare rows and spare columns.

A working instruction cache is necessary for most chip test programs. Consequently, it is automatically tested by built-in self-test (BiSt) and automatically repaired by built-in self-repair (BiSr). During wafer probe, the test result is serially shifted off-chip for permanent repair by laser.  Upon chip reset, BiSt of the instruction cache occurs automatically, but BiSr is not necessary if the chip has been repaired.

The data cache and second-level caches are tested by programs loaded into the instruction cache during wafer probe. These programs condense the test results and write them off-chip to be captured by the tester for subsequent laser repair.

### Chip Logic Testability

To enhance core logic testability, the Alpha 21164 microprocessor contains dual-mode registers that can operate as scan registers or as linear feedback shift registers (LFSRs). The scan mode is used for initialization, for scanning out signatures, and for debugging. The LFSR mode is used for manufacturing test.

### Module Manufacturing

The Alpha 21164 microprocessor implements the IEEE 1149.1 standard for supporting testing during module manufacturing. The supported instructions are EXTEST, SAMPLE/PRELOAD, BYPASS, CLAMP, and HIGHZ.

## SUMMARY

The internal organization of the Alpha 21164, a new, high-performance Alpha microprocessor, has been presented. Mechanisms designed to enhance the CPU's performance combined with the CPU's clock speed of 300 MHz produce an extremely high-performance microprocessor. First silicon of the Alpha 21164 CPU chip was produced in February 1994, and three different operating systems were successfully booted on the first-pass silicon. The part became commercially available in January 1995. It achieved the performance level of 345 SPECint92 and 505 SPECfp92 (estimated), a performance level unmatched by commercially available microprocessors.

## ACKNOWLEDGMENTS

The authors wish to acknowledge George Chrysos, Robert Cohn, Zarka Cvetanovic, Kent Glossop, Lucy Hamnett, Steve Hobbs, John Shakshober, and Paula Smith for their work in producing to the estimates of SPEC benchmark performance quoted in this paper.

## REFERENCES

1. D. Dobberpuhl et al., "A 200-MHz 64-bit Dual-issue CMOS Microprocessor," Digital Technical Journal, vol. 4, no. 4 (Special Issue, 1992): 35-50.

2. 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.

3. E. Swartzlander, ed., Computer Arithmetic (New York: Dowden, Hutchinson, and Ross, 1980).

4. O. MacSorley, "High-speed Arithmetic in Binary Computers," Proceedings IRE, vol. 49 (1961): 67-91.

5. R. Sites, ed., Alpha Architecture Reference Manual (Burlington, Mass.: Digital Press, 1992).

6. D. Bhavsar and J. Edmondson, "Testability Strategy of the Alpha 21164 Microprocessor," International Test Conference (October 1994): 50-59.

## BIOGRAPHIES

### John H. Edmondson

John Edmondson is a consultant engineer in Digital Semiconductor. He was the architecture leader of the design team for the Alpha 21164 microprocessor. Previous to that work, he was a member of the design team for the VAX 6000 Model 600 microprocessor. Prior to joining Digital in 1987, John worked at Canaan Computer Corporation and Massachusetts General Hospital. John received a B.S.E.E. from the Massachusetts Institute of Technology in 1979.

### Paul I. Rubinfeld

Paul Rubinfeld was the engineering manager on the Alpha 21164 microprocessor project. During the last 16 years at Digital, he has worked on VAX and PDP-11 CPU development projects and a single-instruction, multiple-data, massively parallel processor system. Paul received a B.S. and an M.S. in electrical engineering from Carnegie Mellon University, where he helped build the Cm* multiprocessor. Paul is a senior engineering manager within Digital Semiconductor.

### Peter J. Bannon

Pete Bannon is a consulting engineer in Digital Semiconductor. He has participated in the design or verification of several microprocessor chips and was a member of the Alpha 21164 architecture team. He joined Digital in 1984 after receiving a B.S. (special honors) in computer system design from the University of Massachusetts. He holds three patents for VAX CPU design and has filed six patent applications for the Alpha 21164.

### Bradley J. Benschneider

Brad Benschneider is a principal hardware engineer in Digital Semiconductor. He was responsible for designing various sections of the memory management unit on the 21164, as well as defining the latching methodology for the chip. He is currently leading the implementation effort of the memory management unit for the next-generation Alpha CPU. Since joining Digital in 1987, he has contributed to several custom chip designs in the VAX 6000 family and the early Alpha implementations. He received a B.S.E.E. from the University of Cincinnati, has one patent, and has coauthored four papers.

Debra Bernstein

Debra Bernstein is a consultant engineer in Digital Semiconductor. Her work has spanned the areas of architecture, performance, simulation, logic design, firmware, PALcode, verification, and hardware debug for four generations of Digital CPUs. Deb is currently working on the hardware and software components of a PC-based multimedia solution. She received a B.S. in computer science (1982, cum laude) from the University of Massachusetts.

Ruben W. Castelino

Before receiving a B.S.E.E. from the University of Cincinnati in 1988, Ruben Castelino was a co-op student at Digital working on a chip set for the VAX 6000 Model 200. Currently a senior hardware engineer in Digital Semiconductor, he was a codesigner of the cache control and bus interface unit for the Alpha 21164 CPU. Prior to that, he worked on the instruction fetch, decode, and branch unit for the NVAX chip and performed implementation work for the NVAX virtual instruction cache. Ruben is currently a codesigner of the cache control and bus interface unit for a new Alpha microprocessor.

Elizabeth M. Cooper

Beth Cooper received B.S. degrees (summa cum laude) in electrical engineering and computer science from Washington University in St. Louis (1985) and an M.S. degree in computer science from Stanford University (1995). She is a member of Eta Kappa Nu. She joined Digital in 1985 and has worked on the implementations of several CMOS VAX and Alpha CPUs since then. Beth was the lead cache designer on the Alpha 21164 microprocessor. She is currently a principal hardware engineer in the Palo Alto Design Center.

Daniel E. Dever

Since joining Digital in 1988, Dan Dever has worked on the design and logic verification of the CMOS VAX microprocessors as well as the 21064 and 21164 Alpha microprocessors. Dan is currently involved in the design of the memory management unit for the next-generation Alpha microprocessor. He received a B.S. in electrical engineering from the University of Cincinnati in 1988.

Dale R. Donchin

Dale Donchin is an engineering manager and technical contributor in Digital Semiconductor. He designed several circuits related to the clock and cache and contributed to and led CAD tool use for the Alpha 21164 CPU. He is presently performing these duties for the development of the next-generation Alpha microprocessor. Dale joined Digital in 1978 and was previously a development manager in the RSX Operating System Group. Dale holds a B.S.E.E. (1976, honors) and an M.S.E.E. (1978) from Rutgers University College of Engineering and is a member of IEEE and ACM.

Timothy C. Fischer

Tim Fischer is a senior hardware engineer in Digital Semiconductor. He is currently working on the instruction issue logic for the next-generation Alpha microprocessor. Prior to this, Tim worked on the design of the Alpha 21164 floating-point unit, the NVAX+ bus interface unit, and the NVAX clocks and patchable control store. He has coauthored several papers. Tim joined Digital in 1989 after receiving an M.S. in computer engineering from the University of Cincinnati.

Anil K. Jain

Anil Jain, a consulting engineer in Digital Semiconductor, led the implementation of the external interface unit on the Alpha 21164 microprocessor. Prior to this, he was the project leader for the floating-point unit on the NVAX microprocessor. He also made technical contributions on the CVAX microprocessor and on device modeling of Digital's first CMOS process. Anil received a B.S.E.E. from Punjab Engineering College (1978) and an M.S.E.E. from the University of Cincinnati (1980). He holds three patents.

Shekhar Mehta

Shekhar Mehta is a senior hardware engineer in Digital Semiconductor's High Performance Computing Group. He designed the miss address file on the memory subsystem of the Alpha 21164 CPU and was responsible for the electromigration checks of the chip. He is currently leading the design of the caches on a future Alpha microprocessor. Before joining Digital in 1988, Shekhar was an engineer at Larsen & Toubro, Bombay, India. He received an M.S.E.E. from the University of Wisconsin at Madison (1988).

Jeanne E. Meyer

Since joining Digital in 1989, Jeanne Meyer has worked on the implementation, behavioral modeling, and logic verification of several microprocessor chips. In her work on the Alpha 21164 CPU, she was responsible for PALcode verification, maintenance, and support. She also contributed to the microarchitecture definition and behavioral model of the chip's memory management unit. She is currently leading the design of the memory management unit for a new Alpha microprocessor. Jeanne received a B.S.E.E. (summa cum laude, 1982) from the University of Cincinnati. She holds two patents.

Ronald P. Preston

Ronald Preston is a principal engineer in Digital Semiconductor. Since joining Digital in 1988, he has worked on the design of several microprocessors and was the implementation leader for the instruction unit on the Alpha 21164. Ron was also responsible for the architecture and implementation of the issue/bypass/scoreboard logic. Ron is the coauthor of several articles on hot carrier analysis of CMOS circuits. He received a B.S.E.E. in 1984 and an M.S.E.E. in 1988, both from Rensselaer Polytechnic Institute. Ron is a member of Eta Kappa Nu and IEEE.

Vidya Rajagopalan

Vidya Rajagopalan is currently with Quantum Effect Design Inc. Prior to joining QED, she was a member of Digital's Semiconductor Engineering Group, where she worked on the Alpha 21164 and 21064 microprocessor designs. Vidya received an M.S. in electrical engineering from the University of Maryland, College Park and a B.E. from Visvesvaraya Regional College of Engineering, Nagpur, India.

Chandrasekhara Somanathan

Chandrasekhara Somanathan received an M.B.A. from Northeastern University in 1994, an M.S. in computer science from the Rochester Institute of Technology in 1984, and a B.S. in electrical and electronics engineering from BITS, Pilani, India in 1982. While at Digital from 1985 to 1994, he designed the cache controller unit of the Alpha 21164 RISC microprocessor, and the floating-point and cache controller units of the VAX 6000/400 CISC microprocessor; he also developed Digital's MOS timing analysis CAD software. He is currently with HaL Computer Inc., developing high-performance SPARC RISC microprocessors.

Scott A. Taylor

Scott Taylor joined Digital in 1993 after receiving a B.S. degree in electrical engineering from the University of Illinois. He was involved with the functional verification of the memory, cache control, and external interface units on the Alpha 21164 microprocessor. Scott has also worked on CPU test pattern generation and debug as well as on-chip cache repair strategies. He is currently contributing to the verification of the next generation of Alpha high-performance microprocessors.

Gilbert M. Wolrich

A consultant engineer in Digital Semiconductor, Gil Wolrich was the leader and architect for the floating-point unit on the Alpha 21164 chip. He received a B.S.E.E. from Rensselaer Polytechnic Institute and an M.S.E.E. from Northeastern University.