ULTRASPARC-II*i*:

Expanding the Boundaries of a System on a Chip

Kevin B. Normoyle

Michael A. Csoppenszky

Allan Tzeng

Timothy P. Johnson

Christopher D. Furman

Jamshid Mostoufi

Sun Microsystems, Inc.

This processor uses a significant amount of integration and other techniques to enable the construction of cost-efficient SPARC computer systems that retain excellent absolute performance.

The central mission of the UltraSPARC-II*i* is optimized price/performance and ease of use for the system designer. Stated differently, the CPU must deliver a lot of performance for the least impact on overall system cost and also enable simplified system design.

An important strategy in enabling low-cost SPARC-based systems is to leverage the PC industry economy of scale by using the industry-standard PCI I/O bus. The integrated 66-MHz PCI interface gives the UltraSPARC-II*i* access to cost-efficient PC graphics accelerators as well as other I/O cards. Since the CPU also contains an interface to Sun's proprietary high-speed UPA64 bus, our own high-end graphics accelerators such as Creator3D can be used.

A CPU in the desktop arena must strike the "sweetspot" in a multidimensional battle of conflicting design objectives: high-performance, controlled power consumption, low cost, quick time to market, and a capability for simplified system design. The net result of a system containing an UltraSPARC-II*i* will realize a substantial cost savings. More functionality is on the die, and as a result, there are fewer system components; reduced board-level routing, layers, and complexity; and system-level power consumption.

Integration attacks overall system cost, but integrating too much or the wrong type of functionality can become counterproductive. Absolute performance is also obviously important, but we attempt to achieve the knee of the cost/performance curve. We wanted to avoid reaching the point where exorbitant area and complexity is required to gain that last 5% of performance. The resulting CPU from this delicate balancing act of simultaneously optimizing many different objectives is an efficient machine capable of taking SPARC into new markets while solidly preserving those that already exist.

Microarchitecture highlights

The 64-bit, four-way superscalar CPU core provides in-order instruction issue and limited out-of-order completion. In particular, long-latency floating-point operations such as divide and square root do not prevent independent instructions from completing. Also, depending on the memory model used, loads and stores may be reordered. The CPU core is a SPARC V9 machine with VIS extensions. VIS offers excellent flexibility and acceleration of 2D and 3D graphics, image processing, real-time compression and decompression, and video effects applications.

We leveraged the core from the UltraSPARC-II[1,2] and modified the second-level (L2) cache controller and interface, load-store unit, and other changes needed to accommodate the requirements of many new integrated system functions. The core has a 44-bit virtual address space and a 41-bit physical address space with 64-bit address pointers. There are six pipelines: two integer, two floating-point/graphics, one load-store, and one branch.

We used sophisticated instruction prefetching and a nonblocking data cache. A block load-store capability that operates on 64-byte blocks allows excellent bandwidth to and from main memory with arbitrary source and destination alignment. Where prefetch completion is not constrained by the memory model, the CPU supports nonblocking software prefetching into the L2 cache. The compiler can use this feature to further hide the latency of an L2 miss.

Processor pipeline. The main processor pipeline has nine stages, as described in the "Pipeline stages" box. An instruction is considered terminated upon going through the write stage, where the architectural integer and floating-point register files are updated. In other words, the processor's architectur-

0272-1732/98/$10.00 © 1998 IEEE

al state is updated upon completion of the write stage. There are three additional stages, N1, N2, and N3, added to the integer pipeline to make it symmetrical with the floating-point/graphics pipeline.

All floating-point and graphics instructions, except divide and square root, are pipelined and have latencies between one to three cycles. This symmetry simplifies pipeline synchronization and exception handling. Long-latency floating-point operations (divide, square root, and inverse square root) behave slightly differently than other instructions; the pipeline is essentially extended when the instruction reaches stage N1. These long-latency instructions may complete out of order with respect to independent instructions. In case a dependent instruction immediately follows a long-latency instruction, the integer pipeline will stall. This scenario, however, can be minimized by having the compiler schedule independent instructions between the long-latency operation and the instruction that consumes that operation's results. By bypassing the register file, the results of instructions that complete sooner than N3 (or alternatively, X3 in the case of floating-point/graphics) are immediately available to subsequent instructions. Figure 1 (on p. 17) illustrates the microarchitecture block diagram of the UltraSPARC-II*i*.

L1 caches. The UltraSPARC-II*i* uses separate first-level data and instruction caches. The 16-Kbyte instruction cache is two-way set-associative with 32-byte blocks. It is physically indexed and physically tagged. The cache way is predicted in advance, so that only the index bits of an address are needed to address the cache. The prediction of the cache way allows access to the instruction cache to begin in parallel with the virtual-to-physical address translation. The cache returns up to four instructions from a line that is eight instructions wide.

The 16-Kbyte, write-through data cache is nonallocating and direct-mapped with two 16-byte subblocks per line. It is virtually indexed and physically tagged. Virtual addresses are used to index into the cache tag and data arrays while accessing the data translation look-aside buffer, or dTLB. The resulting tag is compared against the translated physical address to determine a cache hit. The tag array is dual-ported so that tag updates due to line fills do not collide with tag reads for incoming loads. Snoops to the cache use the second tag port so that incoming loads are not disturbed. Software handles address aliasing.

External (L2) cache control unit. Since its main role is to handle instruction and data cache misses efficiently, the ECU is the central unit that controls the external L2 cache. A load buffer whose depth mimics the latency of an L2 access, and careful load-use scheduling by the compiler, make a stall on an L1 miss unlikely. The ECU handles all direct memory accesses and maintains data coherency between the L2 cache and main memory. It also acts as a traffic controller between the load-store unit (LSU), prefetch and dispatch unit (PDU), PCI bus module (PBM), and memory control unit (MCU). The ECU can support one access every other cycle to the L2 cache, over an 8-byte (plus parity) bus.

In addition to handling the line fill for instruction and data cache misses, the ECU supports direct memory accesses, which hit in the L2 cache while maintaining coherency between the L2 and main memory. The ECU overlaps processing during load and store misses. Stores that hit in the L2 cache can proceed while a load miss is being processed. The ECU can handle up to three outstanding L2 misses simultaneously without stalling, with the memory control unit returning the corresponding data for the misses in order. The ECU also processes reads and writes without a costly turnaround penalty on the bidirectional L2 cache data bus. Lastly, the ECU supports multiple outstanding data transfer requests to the memory control unit and the PCI bus.

Outside the UltraSPARC-II*i* is a unified SRAM L2 cache. This write-back, allocating, direct-mapped cache is parity protected. It is physically indexed and physically tagged. The L2 can range in size from 256 Kbytes to 2 Mbytes, and uses a 64-byte line size. The L2 always includes the contents of the L1 instruction and data caches. To avoid pollution, block loads and stores that transfer a 64-byte line from and to memory to the floating-point register file, do not allocate into the L2 cache. Two SRAM configurations are supported: 2-2-2 (pipelined) mode and 2-2 (register-latched) mode. In 2-2-2 mode, the L2 SRAMs have a cycle time equal to twice the processor cycle

UltraSPARC-II*i* summary

The UltraSPARC-II*i* allows a system the benefits of UltraSPARC performance and bandwidth while enabling the cost-effective use of PC-class system components. It fully supports the Sun Solaris operating system, and all UltraSPARC software is compatible. UltraSPARC-II*i* is a 64-bit superscalar SPARC V9[1] machine, targeted for use in uniprocessor systems, that is an example of a high-performance system on a chip. We combined functions spanning at least three separate, complex system ASICs along with a modified UltraSPARC-II core into one coherent system on a single die running at 300 MHz.

The net effect is a general-purpose CPU that contains an integrated DRAM controller; PCI 2.1-compatible, 66-MHz/32-bit I/O interface; high-end UltraSPARC Port Architecture (UPA)[2] bus interface; second-level cache controller; three MMUs; three TLBs; flexible interrupts; and other important system functions and V9 extensions. The most notable of the extensions to V9 is VIS, an extended instruction set for imaging and networking.[3] The result is a very cost-effective platform for desktop, server, and high-performance networking, telecommunications, and imaging embedded systems.

References

1. D. Weaver and T. Germond, eds., *The SPARC Architecture Manual, Version 9,* Prentice Hall, Englewood Cliffs, N.J., 1994.
2. K. Normoyle and Z. Ebrahim, "UltraSPARC Port Architecture," Hot Interconnects Symp., 1995; copies available from coauthor Normoyle.
3. L. Kohn et al., "The Visual Instruction Set (VIS) in UltraSPARC," *Proc. 40th Ann. Compcon,* IEEE Computer Society Press, Los Alamitos, Calif., 1995, pp. 462-469.

Pipeline stages

These nine stages make up the main processor pipeline in the UltraSPARC II*i*. See Figure A.

Fetch. Up to four instructions are read from the two-way set-associative instruction cache (I-cache). Branch prediction information, the predicted branch target address, and the predicted instruction cache way of the target are also obtained in this stage.

Decode. After fetch, instructions are predecoded and then inserted into the instruction buffer.

Group. The main task here is to assemble a group of up to four instructions and dispatch all of them in one cycle. Of this group of four, two may be integer and two may be floating-point/graphics. The integer register file is accessed.

Execute. The two integer ALUs process data from the integer register file. Results are computed and, via bypassing, made available for other dependent instructions in the very next cycle. This stage also calculates the virtual address for a memory operation, in parallel with ALU computation. On the floating-point/graphics side, the floating-point register file is accessed during this stage.

Cache. The virtual address of memory operations calculated in the execute stage is sent to the tag RAM to determine if the access (load or store) is a hit or miss in the data cache. In parallel, the virtual address is also sent to the data memory management unit (DMMU) to be translated into a physical address. ALU operations from the execute stage generate condition codes in this stage. These codes are sent to the prefetch and dispatch unit (PDU), which checks whether a conditional branch in the group was correctly predicted. In case of a misprediction, earlier instructions in the pipeline are flushed, and the correct instructions are fetched. Floating-point/graphics instructions start their execution during the X1 stage.

N1. A data cache hit or miss or a TLB hit or miss is determined during this stage. If a load misses the data cache, it enters the load buffer. The access will arbitrate for the external (L2) cache if there are no older unissued loads. If a TLB miss is detected, a trap will be taken, and the address translation is obtained via a software trap handler. The physical address of a store is sent to the store buffer in this stage. To avoid pipeline stalls when store data is not immediately available, the store address and data parts are decoupled and sent to the store buffer separately. For floating-point/graphics, execution continues for most operations during the X2 stage.

N2. The integer pipe waits for the floating-point pipe. Most floating-point/graphics instructions finish execution during this stage. After N2, data can be bypassed to other stages or forwarded to the data portion of the store buffer. All loads that have entered the load buffer in N1 continue their progress through the buffer; they will reappear in the pipeline only when the load data comes back. Normal dependency checking is performed on all loads, including those in the load buffer. For floating-point/graphics, execution continues in the X3 stage.

N3. Traps are resolved during this stage.

Write. All results are written to the architectural register files (integer and floating-point) during this stage. All actions performed here are irreversible. Upon completion of this stage, instructions are considered terminated.

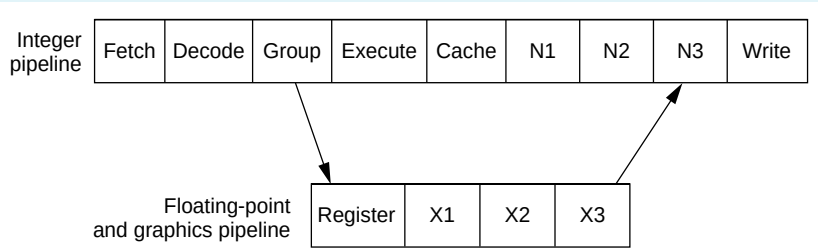


Figure A. Pipeline summary.

time. The name "2-2-2" means that it takes two processor clocks to send the address, two to access the SRAM array, and two to return the data. This mode has a six-cycle pin-to-pin latency and provides the least expensive SRAM solution at a given frequency.

In 2-2 mode, the SRAMs also have a cycle time equal to twice the processor cycle time. This mode takes two processor clocks to send the address, and two more to access and return the data from the SRAMs. In 2-2 mode, there is only a four-cycle pin-to-pin latency, which provides lower overall latency to access the L2 cache. In addition, no dead cycles are necessary when alternating between reads and writes.

Memory control unit. One of the new major functions integrated onto the UltraSPARC-II*i* is the MCU, which handles all transactions to the UPA64S bus interface and DRAM. Access to both of these spaces occurs in a multiplexed fashion on a single 8-byte (plus ECC) data bus. However, there are separate address buses to the UPA64S and DRAM. This shared use of the data bus saves pins and actually results in little performance degradation. We took special care to minimize the number of dead cycles on the shared bus at various speeds, even for transfers between main memory and UPA64S. For instance, the CPU can sustain data transfers at 92% of the theoretical peak on this shared bus during a block load-store code loop, which transfers 64 bytes from main memory to the UPA64S. The basic operations of the MCU are reads, writes, and read-modify-writes. The read-modify-write operation is necessary for a DMA write request with byte-write capability (where only certain bytes within a word are written to) or a DMA write that does not align on a 16-byte boundary.

A three-entry read request queue supports three outstand-

ing read requests from the ECU. A victim buffer address and data buffer are also present to detach the CPU core from being directly tied to, and overly constrained by, DRAM write activity. A separate DMA request buffer hosts DMA requests that have different data size and alignment requirements. This buffer also provides the flexibility of reordering events, which works closely with the ECU to ensure that coherency is maintained between the L2 cache and main memory. Furthermore, sequential store compression is realized by detecting same addresses and merging the data and bytemask fields.

UPA64S is a slave-only interface protocol used most commonly for the UltraSPARC-II*i* to communicate with high-performance Sun-proprietary graphics acceleration boards. However, this bus can be used for any high-bandwidth control or data transfers between the CPU and a dedicated subsystem. Transfers to and from the UPA64S are fully synchronous since it receives a positive emitter-coupled logic (PECL) clock that is aligned with the PECL CPU clock. PECL enables low-noise, low-skew, high-speed signaling. It also allows tight control of the skew between the processor and UPA64S devices, and skew between the processor and SRAMs. The UPA64S interface runs at one third the CPU clock frequency, which results in 8-byte transfers of up to 100 MHz.

The MCU transfers data to and from ECC-protected extended data out (EDO) DRAM through off-chip transceivers. The transfer rate is programmable, but typically occurs at one fourth the CPU clock frequency. Other options are one third and one fifth of the CPU clock frequency. The off-chip transceivers are very attractive since they allow the actual data path to DRAM to be twice as wide as that of the DRAM data portion coming from the CPU. This results in an effective 16-byte (plus 16 bits of ECC) data path to DRAM, with a column address strobe (CAS) cycle of 26.5 ns at 300 MHz. The MCU supports a superset of 60-ns EDO DRAM specifications from all major vendors. The off-chip transceivers are commodity parts from Texas Instruments. Use of faster EDO DRAM is allowed, which may result in increased overall CPU performance. This is possible because the MCU's various timing parameters to interact with DRAM are programmable in software.

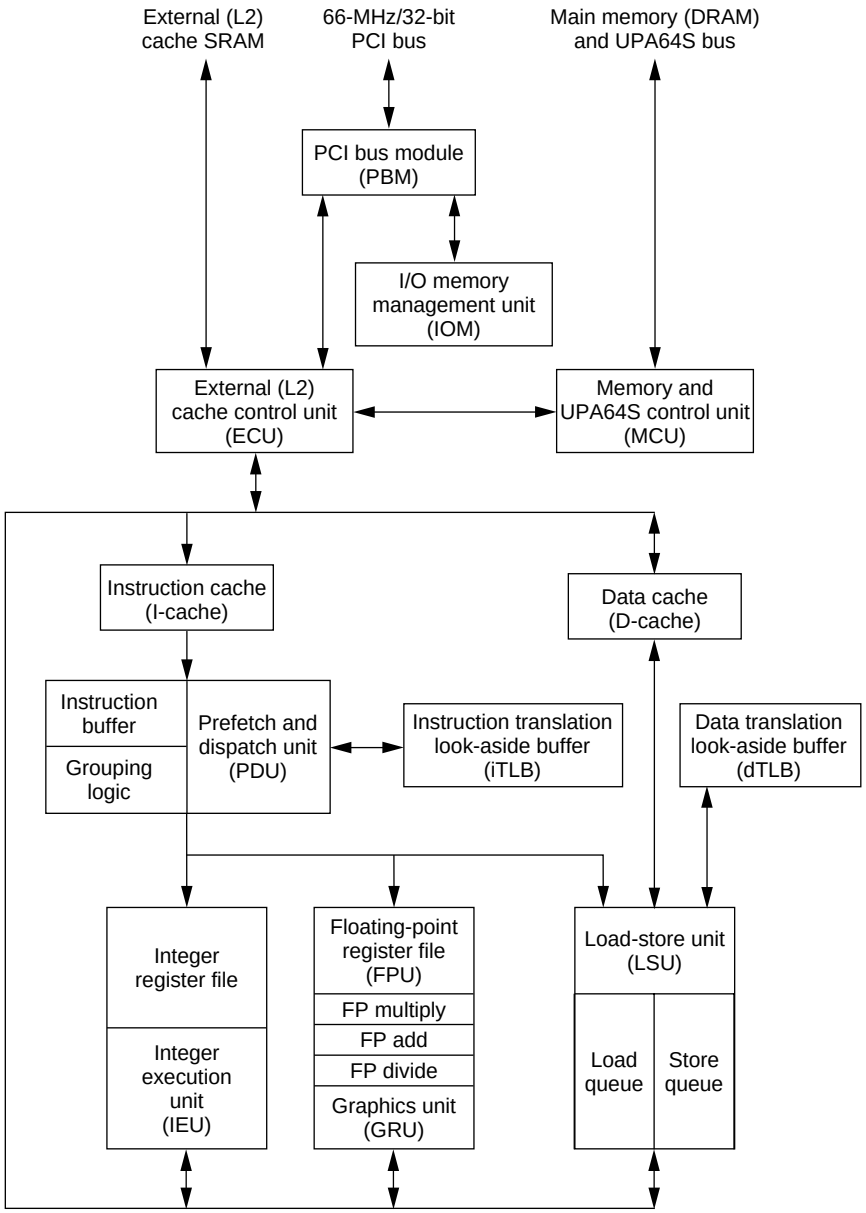


Figure 1. UltraSPARC-II*i* microarchitecture block diagram.

Total latency from the time an address is available on the pins of the CPU to the time data from DRAM is latched at the pins for a 64-byte read is 110 ns for the first 8 bytes. It takes 220 ns total to complete an entire 64-byte read (based on simulation at 300 MHz).

We used several optimizations to reduce memory latency. No row address strobe (RAS) deassertion occurs if back-to-back reads are made to the same page. This saves RAS precharge time and delay to the first word at the DRAM. For back-to-back reads to different SIMMs, the second RAS is asserted at the same time as the first RAS deassertion. This is possible since we can independently control a RAS to the different SIMMs. SIMM pairs are selected via the PA[29:17]

signals. (PA is the physical memory address.) Also, an internal memory request queue can service memory reads ahead of writes, which boosts performance.

As with the L2 cache, the DRAM resides off chip. The UltraSPARCII*i* supports JEDEC-standard, 168-pin DIMMs. Four DIMM pairs using a density of 16-Mbit DRAMs result in a memory capacity of 256 Mbytes. Using 64-Mbit DRAMs allows for a 1-Gbyte main memory size.

Load-store unit. The LSU generates the virtual address of all loads and stores for accessing the data cache, for decoupling load misses from the pipeline via the load buffer, and for decoupling store misses by using the store buffer. The LSU supports one load or store per cycle. In conjunction with the store buffer, it can compress multiple stores to the same 8-byte address range into a single L2 cache access.

The load buffer allows the load and execution pipelines to be decoupled; thus, loads that cannot return data immediately will not stall the pipeline. Rather, loads will be buffered until they can return data. To be more specific, the pipelines are not fully decoupled because the UltraSPARC-II*i* supports precise traps.

All store operations, including atomics and barriers, are entered into the store buffer. This buffer normally has lower priority than the load buffer when arbitrating for the data cache or the L2 cache since, for performance reasons, returning load data is usually more important than store completion. To guarantee that stores complete in a finite amount of time, as required by the SPARC-V9, the CPU eventually raises the priority of the store buffer over the load buffer when a lock-out condition is detected. As mentioned previously, consecutive stores with a write-gathering attribute set may be combined into aligned 8-byte entries in the store buffer to improve store bandwidth. To maintain strong ordering for I/O accesses, stores without the write-gathering attribute set are not combined with other stores. The attribute also influences whether loads and stores are strongly ordered relative to one another.

Prefetch and dispatch unit. The PDU fetches instructions before they are needed in the pipeline so that the execution units do not starve. Instructions can be prefetched from all levels of the memory hierarchy, including the L1 instruction cache, external L2 cache, and main memory. Up to 12 prefetched instructions may be stored in the instruction buffer.

To prefetch across conditional branches, a dynamic branch prediction scheme is implemented in hardware, based on a 2-bit history of the branch. A "next field" associated with every four instructions in the instruction cache points to the next instruction cache line to be fetched. This makes it possible to follow taken branches and provides the same instruction bandwidth as sequential code. A 2-Kbyte, dual-ported next-field RAM contains the branch prediction states, branch target addresses, cache way prediction bits, and LRU bits. We used a 2-bit scheme to predict the direction of control transfer instructions and to speculatively prefetch, decode, and dispatch/issue instructions. From the 2 bits, the four possible states are strongly not taken, likely not taken, likely taken, and strongly taken. This 2-bit branch prediction information is initialized for an instruction when it is first fetched from the L2 cache using the SPARC V9 software branch prediction bit. SPARC V8 branches are initialized as likely not taken. Updates to the prediction information are then based on branch history.

Integer execution unit. The IEU contains two complete ALUs; a multicycle, nonpipelined integer multiplier and a multicycle, nonpipelined integer divider; and eight register windows. Two integer instructions may execute in parallel per cycle, and most integer operations complete within a single clock cycle.

Floating-point and graphics unit. The separation of the execution units in the FGU allows the CPU to issue and execute two floating-point instructions each cycle, or one floating-point and one graphics instruction each cycle. Floating-point instruction source and destination data are stored in a 32-entry array of double-precision floating-point registers. A subset of the registers may contain either two single- or one double-precision value. Most instructions are pipelined (throughput of one each cycle), and have a latency of three cycles, independent of whether the operation is single or double precision. The divide and square-root instructions are not pipelined and take 12 and 22 cycles to complete, for single and double precision. They do not stall the pipeline however. Other instructions following the divide or square root may issue, execute, and be retired to the register file before a divide or square-root operation completes. Precise exceptions are maintained because the integer and floating-point/graphics pipelines are synchronized and the traps for long-latency operations are predicted.

The graphics unit supports the VIS extensions to SPARC-V9. They provide support for 2D/3D graphics, image processing, real-time compression and decompression, audio processing, and video effects and similar applications. They also provide 16-bit and 32-bit partitioned adds, Booleans, and compares, and support 8-bit and 16-bit partitioned multiplies. In addition, they implement single-cycle pixel distance, data alignment, packing, and merge operations.

PCI bus module. The PBM interfaces the CPU directly with a 32-bit PCI bus that is compatible with the PCI specification, Revision 2.1, as shown in Figure 2. The unit also queues the pending interrupts received from the off-chip interrupt concentrator. As with the MCU, PCI functionality on the same die as the CPU core is a major benefit in many regards, very low latency being one.

The PBM supports PCI bus frequencies up to 66 MHz. It is optimized for 16-/32-/64-byte transfers and supports up to four bus masters. The PCI subsystem on the UltraSPARC-II*i* operates in a completely asynchronous clock domain relative to the CPU core. This allows maximum flexibility and scalability, in that there is no required timing relationship between the CPU clock and the PCI clock for the chip to operate correctly and reliably. Internal advanced asynchronous interface logic achieves very high reliability and allows for low-latency communication between the two clock domains, while remaining fully predictable and testable for at-speed production testing. For data transfers from the PCI to the CPU clock domain, latency through the asynchronous interface can range from three to four CPU clock cycles. For data transfers from the CPU to the PCI domain, latency is two P2CLK cycles, where P2CLK is twice the frequency of the PCI bus clock.

Note, however, that the effective use of queueing on the UltraSPARC-II*i* hides much of this latency for a given data stream moving across the asynchronous interface. Solving the asynchronous interface issue on the die itself relieves the system designer of having to solve this precarious communications issue between PCI and the CPU at the board level.

The entire PCI address space is noncacheable for CPU references such as programmed I/O but fully coherent for direct memory access. This means that an external PCI device that reads or writes into the UltraSPARC-II*i* 's memory space can experience a performance benefit in interacting with any level of the chip's memory hierarchy. Because the processor's ECU handles coherency issues, the device is not burdened with them.

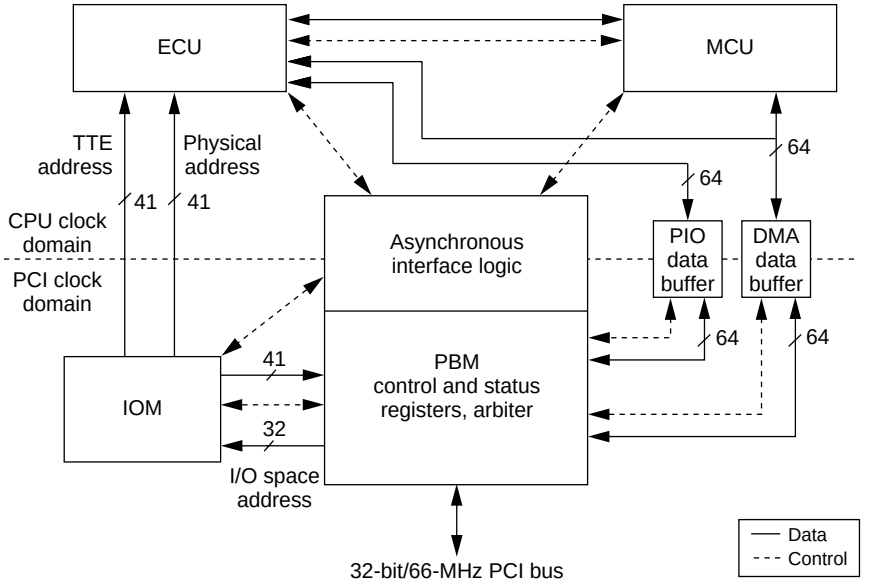


Figure 2. PCI subsystem interface with the CPU core.

I/O memory management unit. The IOM translates addresses from a 32-bit PCI I/O address to a 41-bit physical address, as well as handling memory protection for I/O. The IOM translates addresses when the UltraSPARC-II*i* is a DMA target (that is, when a PCI device requests either a write to or a read from the UltraSPARC-II*i* 's memory). The unit contains a custom 16-entry, fully associative TLB to accelerate address translations. Blocks that are located in the PCI clock domain are clocked at twice the PCI bus clock frequency; the IOM is among these blocks. When the IOM hits into its TLB, it can translate a PCI address into a physical address in one P2CLK cycle. (P2 denotes two times the PCI bus clock frequency.) Upon a TLB miss, the IOM supports a single-level hardware tablewalk into a large translation storage buffer in memory. This hardware-assisted miss processing significantly speeds up address translation when TLB misses happen, as compared to a fully software-only approach.

The IOM supports 8-Kbyte, 64-Kbyte, and mixed page sizes. Mixed page sizes mean that address-mapping information within the internal 16-entry TLB can reference any combination of 8-Kbyte and 64-Kbyte pages at the same instant in time. (For example, the address mapping for entry 0 can reference an 8-Kbyte page, and entry 1 can be referencing a 64-Kbyte page at the same point in time.)

Implementation highlights

The UltraSPARC-II*i* is fabricated on the Texas Instruments Epic 4-GS2 process. This 0.25-micron (polysilicon) CMOS process has five metal layers. The processor core operates at 2.6 V, while the I/O is a mixed 2.6 V and 3.3 V. The 3.3 V is a requirement of the PCI bus specification. At the top level of the design are 36 layout blocks (where a layout block may actually be a complete cluster in itself; the FGU for example, counts as one block), 35 channels, and over 7,300 nets.

To minimize clock skew at the system level, we placed two phase-locked loops on the die, one to support the CPU core clock domain and the other, the PCI. To minimize latency for internal communication between the CPU and PCI clock domains, the *internal* PCI clock runs at twice the PCI bus clock speed. (The PCI PLL multiplies the bus clock by two.)

Table 1. UltraSPARC-II*i* package ball usage. Total with power and grounds: 587.

Interface	No. of balls
PCI plus arbiter	53
Interrupt	9
UPA64S bus	35
DRAM, DRAM buffer control	34
DRAM data	72
Clock and reset	18
JTAG/test	10
L2cache data/parity, tag data/parity	90
L2 cache byte write enables	8
L2 cache address/control	37

The UltraSPARC-II*i* is packaged in an advanced 37.5-mm × 37.5-mm, flip-chip, plastic ball grid array (PBGA) with a total of 587 balls. The package consists of a surface laminar circuit and a metal cover to protect the flip-chip die and to dissipate heat while the processor is operating. Table 1 lists the details of the UltraSPARC-II*i* ball usage.

System highlights

The UltraSPARC-II*i* resides on a module with the L2 cache and logic to generate the CPU, PCI, and UPA64S clocks. In turn, the module plugs into the system motherboard. The use of a module allows easy upgrades to faster processors

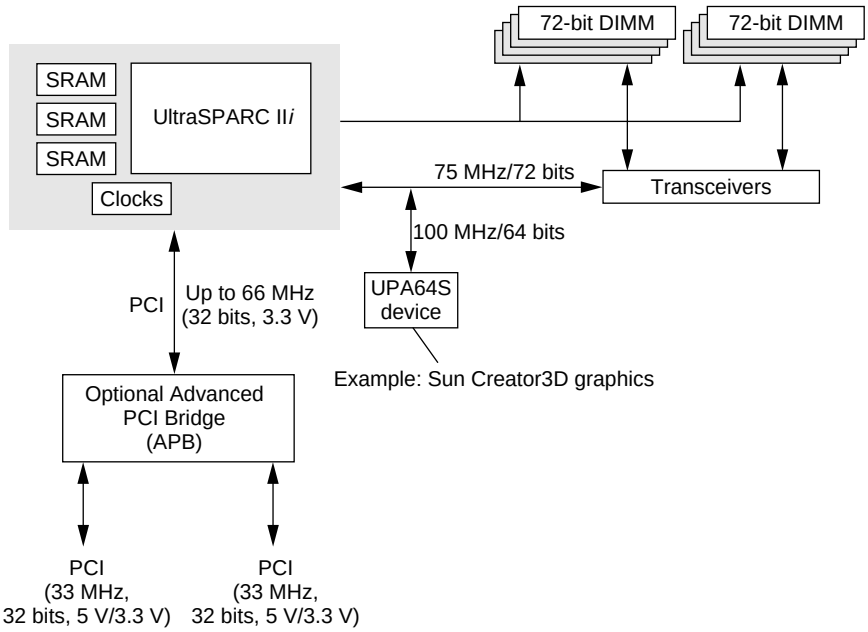


Figure 3. Simple example of an UltraSPARC-II*i*-based system.

that become available. Figure 3 shows a simple example of an UltraSPARC-II*i*-based system. This system includes the APB (Advanced PCI Bridge) chip, which we designed concurrently with the UltraSPARC-II*i* project. This optional ASIC can expand the number of PCI devices that the UltraSPARC-II*i* can communicate with. The 32-bit/66-MHz, 3.3-V primary bus connects directly to the same physical PCI bus as the UltraSPARC-II*i*. The APB also has two fully independent secondary PCI buses, which are 32 bits/33 MHz and 5 V/3.3 V. Each secondary bus can support up to four PCI masters. Designers can attach up to four APBs to the primary bus along with the UltraSPARC-II*i* so the CPU can communicate with up to 32 PCI devices.

Project execution

This section describes how we turned the UltraSPARC-II*i* concept into a reality.

Design methodology. In addition to pursuing performance, cost, and ease of use, one of our goals for the UltraSPARC-II*i* project was an aggressive time to market. This requires that the design cycle be kept as short as possible. In line with this objective, the term design reuse had true meaning for the UltraSPARC-II*i* project. Instead of reinventing the entire CPU core functionality, we included much of the UltraSPARC-II core to form the heart of the UltraSPARC-II*i*. The existing UltraSPARC-II core was already an efficient design that met with the *i*-series objective of an optimized performance/area-complexity cost function. This reuse saved significant time, effort, and risk. As mentioned previously, we modified some of the core and associated pin interfaces, but we expended most of our time and effort concentrating on the integration of key system functionality onto the die. This is the important value added to the processor.

Early identification of custom megacells (transistor-level blocks) was important since these blocks required significant resources from the circuit design group to implement required functionality and timing objectives and to verify functionality, timing, and physical design. We implemented functionality that was extremely critical for performance, area sensitive, or otherwise not synthesizable as a megacell. (This did not include some data path blocks that were hand-designed at the gate level.) Work on the solution to the multiple asynchronous clock domains also started very early in the project to ensure a very reliable and efficient solution.

Much of the new logic on the UltraSPARC-II*i* followed an ASIC-like register-transfer level/synthesis/placement-and-routing flow. This flow had obvious attractions from a time-to-market viewpoint. Getting the resulting logic to operate at the 300-MHz timing objective required the RTL designers to write more structural code than is normally found on traditional ASICs. This was a manageable compromise that neither significantly eroded the design time benefits of a traditional synthesis-based methodology or impacted the full-chip timing objective.

Timing verification methodology. The UltraSPARC-II*i* project involved a number of challenges in timing verification. One was the target 300-MHz frequency for the CPU core in a 0.25-micron process. With this deep-submicron technology, interconnect delays are, of course, significant, requiring careful interconnect design through well-thought-out floor planning. Another challenge was the integration of the PCI subsystem into the UltraSPARC-II*i*, which resides in a completely asynchronous clock domain with respect to that of the CPU core.

We defined groups of functional blocks as clusters in their respective clock domain and performed static timing at both the cluster level and the full chip level. Cluster designers, after validating their local timing, passed the cluster information to a dedicated full-chip timing team. Their function was to perform full-chip timing and feed back the results to the cluster level. The full chip was timed in two runs, one for the CPU core and another for the PCI domain. The team analyzed the timing of the interface between the two clock domains separately, in a fairly manual process.

As clock speeds increase and process geometries shrink, interconnect effects definitely become more dominant. Timing optimization of the UltraSPARC-II*i* involved a significant amount of routing tweaks. We widened many chip-level signals to reduce RC effects and made a significant number of RTL changes to accommodate interconnect delays. Inter-

connect effects were heightened because of the increased die size necessary to accommodate the integrated functions of the UltraSPARC-II*i*. Critical timing paths between subsystems inherited from the UltraSPARC-II became longer. The extra die size increased signal tracks in the worst case by 1 mm, which corresponds to an approximate interconnect delay increase of 130 picoseconds. With a cycle time of only 3,333 ps, this extra delay is significant. In many cases, these subsystems could only accept minor design enhancements, requiring significant ingenuity to ensure these paths met the cycle time goal.

Timing verification involved accurate analysis of the entire chip, dissemination of massive amounts of data associated with a 5.75-million-transistor design, and efficient allocation of people to solve critical timing paths. Timing methodology focused on the dual clock domains, a 50-MHz test domain, and full-chip minimum-path timing to eliminate hold-time violations in the design. We based the timing methodology on two tools: a static timing analyzer to run at the full-chip level to identify the critical paths, and an in-house tool based around SPICE to perform the detailed analysis necessary to propose and implement timing solutions. We wrote many tools to facilitate timing analysis, including sorting scripts, data formatting, and modeling tools. We analyzed the full-chip timing on a weekly basis. Figure 4 shows an overview of the full-chip timing methodology.

The design hierarchy is represented by RC netlist files. These files contain the resistance and capacitance information for all signals, and instances of other components (which are either other netlist files or library cells). The library cells are characterized using SPICE-like tools, and they contain timing arcs, drive resistance, and pin capacitance information. The static timing tool takes in these design netlists together with clocking information and false-path details (a fairly small list). An important philosophy used on the UltraSPARC-II*i* is to not allow false paths, unless those paths cannot be made to meet timing with reasonable time, effort, and cost.

Clearly, an important issue of static timing in general is that of false paths. To be able to statically analyze timing on something as complex as a CPU, our methodology of basically forcing all paths (real and otherwise) to meet timing constraints greatly simplified this problem, without significant ill effects. This greatly reduced any risk of potential mistakes, in erroneously labeling a path false when in fact it is an actual concern. The clock skew values provided to the static timing tool were worst case; other tools and methodologies ensured that we met these clock skew values. For the timing verification runs, the clock network had no interconnect RC delays, since the static timing tool was aware of the worst-case skew budget.

Static-timing analysis produces a critical-path report that shows all violating paths. These timing paths were sorted, based on device and node name criteria, into similar groups of data called buckets. This made it possible for cluster engineers to be assigned buckets of data to investigate and resolve. An assigned engineer would propose and implement a solution to the bucket. We also used detailed SPICE simulations to analyze the worst critical paths and to validate bucket analysis and solutions. Since the runtime for

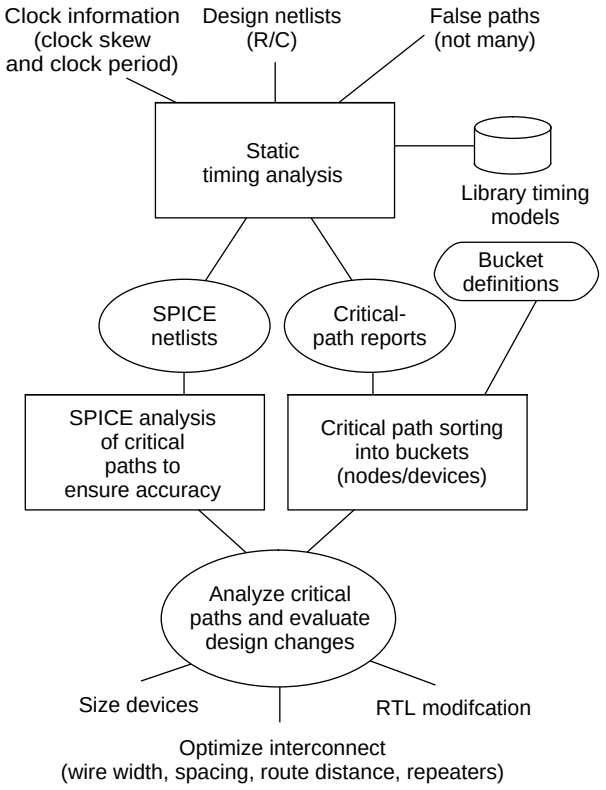


Figure 4. UltraSPARC-II*i* timing verification overview.

SPICE analysis was considerable, we analyzed only the top paths in each bucket, about 1,000 paths using SPICE. This analysis took place on a weekly basis.

Critical timing paths involving complex logic blocks contain many hidden paths that are only slightly less critical than the reported path. A solution to the critical path may not fix the hidden path, which then becomes the next critical path. Reducing the cycle time of the bucket in this manner is a highly inefficient iterative process. To reduce the time taken to fix a bucket, we implemented an "onion-peeling" methodology. This involved examination of all slack paths in a critical path; by peeling back all critical timing paths, we could propose a solution to solve the entire path.

Interconnect delays are a major component in most timing paths, delays between top-level subsystems being especially critical. To reduce these delays, we had to route many signals in nonminimal width and spacing, which reduced both the resistive and capacitive components of the signal at the expense of extra area. In addition, we had to break many long signals traveling across the chip into two or more shorter segments, inserting repeaters to reduce the overall interconnect delay. (RC delay is proportional to the length of a wire squared.) As a general rule, all signals traveling more than 10 mm were candidates for repeater insertion. We added spare repeaters to the periphery of many blocks and one dedicated repeater block in a strategic position in the top-level floor plan.

To aid the investigation of possible interconnect strategies,

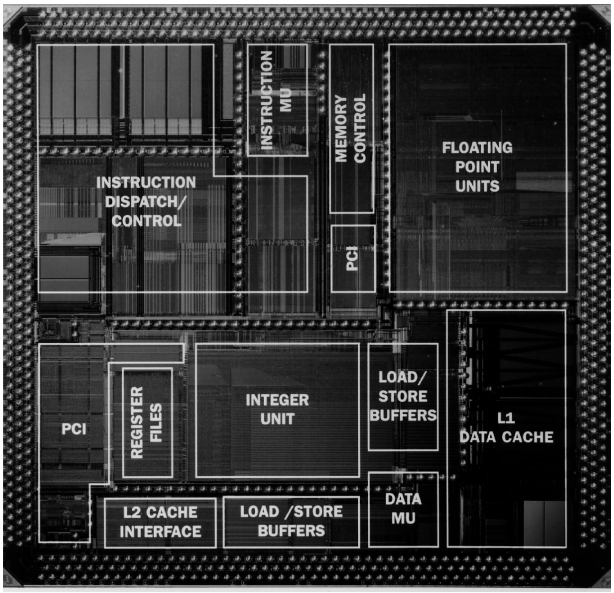


Figure 5. UltraSPARC-II*i* die.

the team designed a tool to model wire segments and allow various interconnect width, spacing, layer, and driver options to be simulated. This was later developed into a Java-based interconnect analysis tool for future projects. If device and interconnect sizing did not reduce the delay significantly, or were not practical, we inserted a repeater. The use of repeaters was usually the last option since they were a limited resource. In total, we used 1,200 repeaters to improve full-chip timing, leaving 100 unused repeaters.

Our timing methodology enabled the identification and elimination of over 3,000 critical timing paths. While we met the UltraSPARC-II*i* timing challenge, we addressed a number of issues with less-than-optimal solutions. The repeater methodology was entirely manual. Future projects using less than 0.25-micron process geometries will encounter more severe interconnect effects. The number of signals requiring repeaters will increase, and many signals will require multiple repeaters to meet their cycle time goals. The use of synchronization logic between the two clock domains also involved fairly manual identification and analysis of all the control signals between the PCI/CPU interface.

Functional verification methodology. Absence of an effective functional verification methodology, especially on a project as complex as the UltraSPARC-II*i*, can quite easily be the gating item in the success or failure of a project. We attacked functional verification at multiple levels. Block designers typically used stand-alone tests to validate the functionality of a particular block. At the cluster level (collection of blocks), they also had another set of stand-alone tests to stress functionality at that level. Finally, and most importantly, the validation group used a very extensive suite of tests, written in SPARC assembly, to exercise the entire chip. These top-level tests also included various system-level components, such as the Advanced PCI Bridge ASIC, various L2 cache/DRAM configurations and speeds, and many variants of different ratios between the CPU and PCI clocks. The vast majority of the full-chip test suite could compare the outcome of the UltraSPARC-II*i* design with a software model of the processor written in C by an independent group. To a lesser extent, we also used self-checking tests.

The simulation environment had to be robust enough to model the real world and be capable of handling the two independent clock domains. It had to be able to stress all specified design functionality and had to run fast enough that random code generators could wring out all corner-case functional bugs. We executed all functional tests at both the register-transfer and gate levels (the gate netlists were extracted from the layout). Since the UltraSPARC-II*i* is fully backward compatible with prior UltraSPARC processors, we also executed the tests run on these earlier chips.

As mentioned earlier, the environment supported both directed and random testing. The random tests were particularly effective in uncovering hard-to-find corner-case bugs, and the directed tests were very effective at exercising particular design areas. Both schemes complement one another quite nicely. We used several different random code generators developed by different people within Sun.

Another effective technique to expose functional bugs was the use of monitors. A monitor can essentially be viewed as a shell around a bus, block, or cluster that checks various signal handshakes and relationships. Take a simple internal tristate bus for an example. Among other items, the monitor for this bus checks for multiple drivers (bus conflict) or no drivers (bus is floating). A monitor can halt the simulation and flag an error if it detects something incorrect. They are very useful; when the CPU executes a test and fails, they allow speedy isolation of the source of the problem. Monitors also help in the early exposure of hidden functional problems that may not yet have propagated to the CPU pins. Then the designer can resolve problems sooner. In addition to our own PCI bus monitors, we used two independent, third-party PCI bus monitors to ensure protocol compliance.

Lastly, an extensive toggle-checking mechanism highlighted any functionality not being exercised sufficiently. This capability was invaluable in ensuring that the test suite had very high coverage of the intended functionality and that nothing was left untested.

As a final layer of pre-tapeout verification, we emulated the entire UltraSPARC-II*i* and Advanced PCI Bridge ASIC via hardware. Among other tests, we booted Solaris and executed Openwindows successfully prior to tapeout. Since the core of the UltraSPARC-II*i* was leveraged from a prior design, we were able to take advantage of the emulation technology investment already made in the original design project. Hardware, tools, and methodologies all leveraged forward to the benefit of our emulation effort.

The emulation setup was still not easy, as it was a combined hardware and software effort, but it was definitely beneficial. Roughly two bugs were identified on the UltraSPARC-II*i* with emulation before tapeout. Using strategically placed monitors, we were able to duplicate the problems found on the emulator in our simulation environment to fully debug the root cause of the problem. Two bugs may not sound like a lot, but these in fact were critical defects that would have impeded our bringing up and debugging the system.

Results

Now that we have shown what the UltraSPARC-II*i* is, and a little about how we built it, we briefly highlight some of the results.

Area and power. Figure 5 is an illustrated die photo of the UltraSPARC-II*i*. The UltraSPARC-II*i* occupies a 11.8-mm × 12.5-mm die area and uses 5.75 million transistors. The power consumption (in watts) is in the low 20s. Recall that since this is a heavily integrated processor, the power consumption includes the I/O, memory, graphics controllers, and interrupt subsystem in addition to that of the CPU core itself.

Performance. One of the benefits of being an integrated processor is that the 300-MHz UltraSPARC-II*i* can offer high bandwidth as well as low latency connections to both memory (including the L2 cache) and I/O. The 8-byte (plus ECC) interface to the DRAM transceivers can sustain a 400-Mbyte/s transfer rate. Over the same 8-byte bus, the transfer rate to UPA64S (to communicate with a Sun Creator3D graphics accelerator, for example) can sustain 800 Mbytes/s. The interface to the L2 cache is also 8-bytes wide and supports a sustained 1.2-Gbyte/s data rate. The 4-byte interface to the 66-MHz PCI bus can sustain 190-Mbyte/s direct memory accesses. Table 2 lists more details of throughput on a 300-MHz UltraSPARC-II*i*. Table 3 lists the overall CPU performance for SPECint95 and SPECfp95 benchmarks, with two different L2 cache sizes.

Functionality and timing. The UltraSPARC-II*i* booted Solaris in record time after receiving first silicon. Within four weeks, 100-plus systems containing the first-silicon UltraSPARC-II*i* were running all applications at the target clock frequency of 300 MHz.

Table 2. 300-MHz UltraSPARC-II*i* interface throughput summary.

Operation	Size (bytes/s)	Throughput (bytes/s)
Memory		
L2 cache read		1.2G
L2 cache write		1.2G
DRAM read (random)		300M
DRAM write		350M
DRAM read (to same page)		400M
DRAM, memcopy		303M
DRAM, memcopy to UPA64S		550M
PCI (66 MHz, 32-bits, single device)		
PCI to DRAM (DMA)	64-byte writes	151M
DRAM to PCI (DMA)	64-byte reads	132M
PCI to L2 cache (DMA)	64-byte writes	186M
L2 cache to PCI (DMA)	64-byte reads	163M
CPU to PCI (PIO)	64-byte writes	200M
Stream (compiled/VIS block store)		
Copy		199M/303M
Scale		199M/296M
Add		244M/277M
Triad		210M/277M
UPA64S (100 MHz, 64 bits)		
CPU to UPA (PIO)	16-byte writes	800M
CPU to UPA (PIO)	64-byte writes	600M

Table 3. Estimated 300-MHz UltraSPARC-II*i* SPEC95 results (ref. platform, SCS.0 compiler).

Benchmark	0.5-Mbyte L2	2.0-Mbyte L2
SPECint95	12.1	12.7
SPECfp95	12.9	15.2

WHILE THE UltraSPARC-II*i* optimized price/performance and eased the task of system design, it is also an example of design reuse. Since the UltraSPARC-II core was already an efficient design, and thus met with the requirements of an *i*-series CPU, we saved significant time, effort, and risk in not having to completely reinvent a new processing core from the ground up. With only relatively minor design changes required in the UltraSPARC-II core to support the significant amount of integrated system functions added on chip, the design team could concentrate on the true value-added aspect of the UltraSPARC-II*i*: system integration. This brought a significant amount of functionality found on the motherboard onto the chip itself.

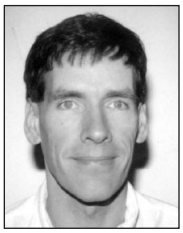
The integrated UltraSPARC-II*i* enables the construction of a large variety of cost-efficient computer systems, while at the same time retaining significant absolute performance and excellent bandwidth and latency to memory, the L2 cache, and I/O. We maintained full compatibility with the Solaris operating system and all prior UltraSPARC code.

Acknowledgments

Tremendous thanks and appreciation go to the entire UltraSPARC-II*i* project team. Without the hard work of many people, UltraSPARC-II*i* would not exist today. Due to space limitations, we can't mention everyone here, but we heartily thank the project team, and all the people at Sun who helped make this project a success.

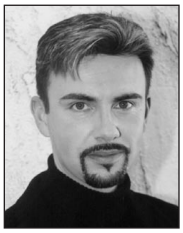
References

1. D. Greenhill et al., "A 330MHz 4-Way Superscalar Microprocessor," *Proc. Int'l Solid-State Circuits Conf.,* IEEE, New York, 1997.
2. D. Greenley et al., "UltraSPARC: The Next Generation Superscalar 64-Bit SPARC," *Proc. 40th Ann. Compcon,* CS Press, 1995, pp. 442-451.



Kevin B. Normoyle is a senior staff engineer in the Sun Microelectronics Division of Sun Microsystems and the chief architect of the UltraSPARC-II*i*. He has been involved with CPU and chip interface design beginning with the UltraSPARC I.

Normoyle holds a BS in electrical engineering from Cornell University.



Michael A. Csoppenszky is a staff engineer with the Sun Microelectronics Division of Sun Microsystems. He was a cluster technical leader for the PCI subsystem, owner of the full-chip asynchronous interface between the CPU and PCI clock domains and associated megacells, and block owner for the I/O memory management unit and PCI data-path units on the UltraSPARC-II*i*. His interests include computer architecture, asynchronous design, and the VLSI design of deep-submicron integrated circuits.

Csoppenszky holds an MS degree in electrical engineering from the University of Washington, Seattle. He is a member of Eta Kappa Nu.

Allan Tzeng is a hardware design manager in the Sun Microelectronics Division of Sun Microsystems, where he manages a cluster for a next-generation UltraSPARC processor. Earlier, he was the cluster technical leader of the UltraSPARC-II*i* external cache controller unit. His interests lie in computer architecture and implementation.

Tzeng holds an MS degree in computer engineering from Syracuse University in New York.

Timothy P. Johnson is a senior design engineer with the Sun Microelectronics Division of Sun Microsystems. He was responsible for the full-chip timing verification on the UltraSPARC-II*i*. Previous microprocessor contributions include the Inmos T9000 Transputer, SGS-Thompson ST40, and Sun Microsystems UltraSPARC-II*i*. His technical interests include floating-point arithmetic design, high-speed microprocessor design, and overcoming the physical limitations inherent in deep-submicron designs.

Johnson received a MEng in electronic systems engineering from York University, England. He is a member of the IEEE.

Christopher D. Furman is a staff engineer with the Sun Microelectronics Division of Sun Microsystems. He was responsible for the full-chip assembly and top-level routing of the UltraSPARC-II*i*. His interests include the architecture and vertical design of high-end processors and peripherals.

Furman attended the University of Washington, Seattle.

Jamshid Mostoufi is a senior verification engineer with the Sun Microelectronics Division of Sun Microsystems. He was involved in the functional verification of the UltraSPARC-II*i*. His technical interests are in the pre- and post-silicon verification of microprocessors.

Mostoufi received a BS in computer science from San Francisco State University, California.

Direct questions concerning this article to Michael A. Csoppenszky at M/S USUN02-301, 901 San Antonio Road, Palo Alto, CA 94303-4900; michael.csoppenszky@eng.sun.com.