UltraSPARC-III : An indepth study

Sanjeev Dwivedi and Anandha Gopalan

Abstract

This paper does an indepth study of the UltraSPARC-III processor from Sun. The UltraSPARC-III is the third processor from Sun to implement the 64-bit SPARC V9 compliant architecture. This processor can issue up to four instructions per cycle and can reach clock rates of greater than 1-Ghz. The performance is improved over the earlier versions of UltraSPARC by scaling both bandwidth (ILP) and latency. The design permits high scalability to build systems consisting of thousands of UltraSPARC-III processors. UltraSPARC-III maintains binary compatibility with all existing SPARC applications and the Solaris operating system. This paper also talks about the Visual Instruction Set, which is an extension to the instruction set of SPARC V9.

Keywords - Microprocessors, Computer architecture, pipelining, integrated circuits, cache.

1 Introduction

The UltraSPARC-III processor from Sun is the third processor to implement the 64-bit SPARC V9 compliant architecture, [1, 2, 3] it is the newest in the UltraSPARC generation of processors. The design goals were based on three important factors : scalability, compatibility and performance. Figure 1 shows the comparison between the three generations of SPARC processors.

	1st Generation	2nd Generation	3rd Generation
Architecture	SPARC V9	SPARC V9	SPARC V9
Device Count	5.2M	5.4M	23M
Die Size	17.7x17.8mm^2	12.5x12.5mm^2	15x15.5mm^2
Data Cache	16KB	16KB	64KB
Instruction Cache	16KB	16KB	32KB
External Cache Tag	-	-	88KB
Frequency	167MHz	330MHz	1000MHz
Supply Voltage	3.3V	2.5V	1.6V
Power dissipation	<30W	<30W	<80W
Metal layers	4	5	7
Lpoly	0.45µm	0.28µm	0.15µm
Metal1 pitch	1.6µm	1.12µm	0.56µm

Figure 1: Comparisons of designs of three generations of SPARC processors

The design permits high scalability of to build systems consisting of thousands of UltraSPARC-III processors. The on-chip memory system and the bus interface were designed to be able to handle systems built using one to thousands of UltraSPARC-III processors. There are more than 10,000 third-party applications available for SPARC processors, hence binary level compatibility is an essential goal of any new SPARC processor. The goal was to try and improve application performance without having to recompile the application. This improvement in performance must be application independent - it must apply to all applications and not to only those which are a good match for the architecture. Also, this design ensures compatibility with all existing SPARC applications and the Solaris operating system.

High performance was achieved using a multiprong approach. Recent research focussed on designing ways to extract more instruction-level parallelism from programs. The disadvantage was that the speedup varies greatly across different programs. The reason for this was that ILP techniques varied greatly between programs, as many programs use algorithms whose data dependency forces the instructions to be executed serially. This led to the belief that scaling bandwidth (ILP) alone cannot improve performance for all programs. The focus was then to scale both bandwidth and execution latency, (*i.e*) to increase the bandwidth and reduce the execution latency. Compared to the previous generation, the clock rate is increased by 150%, and the instruction parallelism is improved by 15% [2]. Test and debug capabilities are also included in UltraSPARC-III which has an IEEE compliant 1149.1 test access port (TAP) and controller [7].

The rest of the paper is organized as follows. Section II gives a detailed description about the architecture of the processor. Section III talks about the Visual Instruction Set, an extension to the SPARC V9 instruction set, which provides instructions that greatly enhance the multimedia and image processing capabilities of the SPARC processors, and Section IV talks about the test and debug capabilities built into UltraSPARC-III. Section V concludes the paper.

2 Architecture

The UltraSPARC-III has a deep pipeline, which has 14 stages in it. Table 1 shows the details of the pipeline and the functions of the various stages. The number of stages are the highest in any UltraSPARC processor so far. The instruction fetch stage spans the stages A through J and the instruction execution stage uses stages R through D. The data cache unit executes in parallel with stages E, C, M, W and X as shown in figure 2.

Stage	Function
A	Generate instruction fetch addresses, generate predecoded instruction bits on cache fill
P	Fetch first cycle of instructions from cache; access first cycle of branch prediction
F	Fetch second cycle of instructions from cache; access second cycle of branch prediction; translate virtual-to-physical address
B	Calculate branch target addresses; decode first cycle of instructions
I	Decode second cycle of instructions; enqueue instructions into the queue
J	Steer instructions to execution units
R	Read integer register file operands; check operand dependencies
E	Execute integers for arithmetic, logical, and shift instructions; read, and check dependency of, first cycle of data cache access floating-point register file
C	Access second cycle of data cache, and forward load data for word and double-word loads; execute first cycle of floating-point instructions
M	Load data alignment for half-word and byte loads; execute second cycle of floating-point instructions
W	Write speculative integer register file; execute third cycle of floating-point instructions
X	Extend integer pipeline for precise floating-point traps; execute fourth cycle of floating-point instructions
T	Report traps
D	Write architectural register file

Table 1 : UltraSPARC-III Pipeline stages and their functionalities

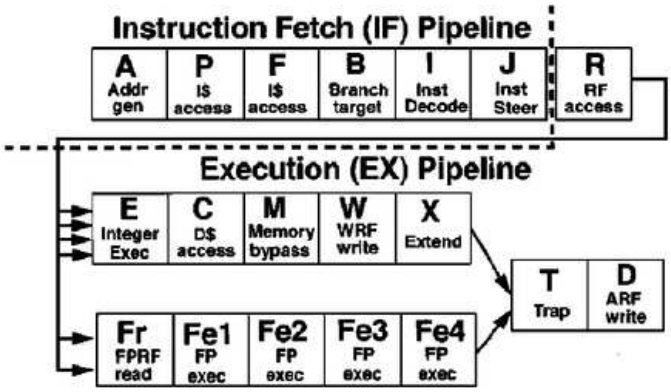

Figure 2: The 14-stage pipeline of UltraSPARC-III

One problem that can occur with a deep pipeline is that the cost of a branch misprediction is very high as we have to flush the whole pipeline and start fetching from stage A again. This means that we incur a penalty of eight cycles (A through E stage). This is indeed a heavy penalty to pay, the design has taken this into consideration and there is an implementation of a miss queue, which has all the instructions that were fetched during the predict taken phase, but was on the other path. These instructions will be immediately available to start the I stage. This will be dealt in more detail when we talk about the instruction issue unit.

There are six major functional units : instruction issue unit (IIU), floating point unit (FPU), integer execution unit (IEU), data cache unit (DCU), external interface unit (EIU), and system interface unit (SIU). These are explained in much greater detail below:

2.1 Instruction Issue Unit

This unit is responsible to fill the pipeline with instructions. It predicts the control flow of the program and fetches the predicted path from memory. The design decision was to keep UltraSPARC-III a static speculative machine instead of a dynamic speculative one. The reason behind this was that dynamic speculative machines need a very high bandwidths to fetch enough instructions to fill the pipeline and find instruction-level parallelism. In a static speculation machine, we can make the instruction fetch unit much simpler as we put fewer requirements on it. The fetched instructions are staged in a queue before being forwarded to the two execution units. The IIU includes a 32-Kbyte, four-way associative instruction cache, the instruction address translation buffer, and a 16-K entry branch predictor. Figure 3 shows the different blocks of the IIU.

Address fetch happens in stage A. Also, during this stage there is a 32-byte buffer that supports sequential prefetching into the instruction cache. This is very useful, as when we have a cache miss, we request 32 bytes, but instead of requesting 32 bytes as needed by the cache, the processor requests that 64 bytes be brought in. The first 32 bytes are filled into the instruction cache and the next 32 bytes are stored in the buffer. The cache can use the buffer to get the next instruction if the next sequential cache line is also a miss.

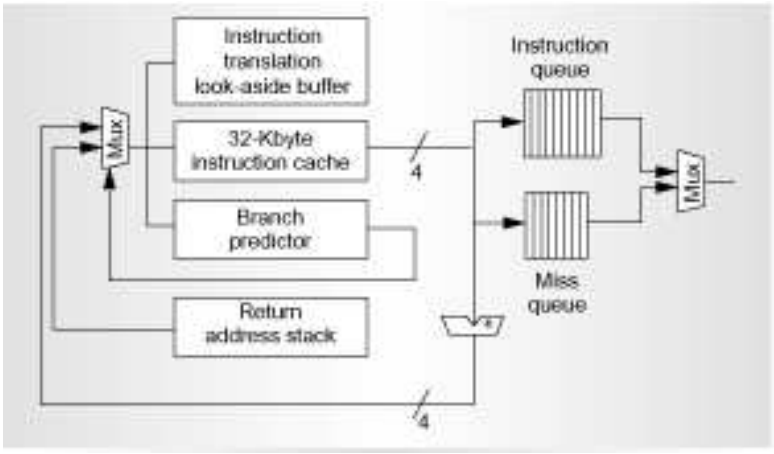

Figure 3: Instruction issue unit (IIU) block diagram

The branch predictor used here is a 16-K 2 bit up/down saturating counter, Gshare predictor. This is a very very large predictor, which has a lot of entries, which means more overhead to access the entries. While this might give us better prediction, we might incur more latency while trying to get the values out of the predictor, but the tradeoff pays off as a better predictor with a little extra latency is much better compared to a predictor that does not do a good job of prediction, as the branch misprediction penalties are quite large. The scheme offsets the history bits such that the three lower-order index bits that index into the predictor use information from the PC only.

As mentioned earlier, with a very high pipeline depth, branch mispredictions can be very costly. They can have as much as a 8 cycle penalty. The problem is taken care of by buffering the instructions. There are two instruction buffering queues in the UltraSPARC-III; the instruction queue and the miss queue. The fetch unit predicts the path of execution and keeps filling up the instruction queue until it's full. The four-entry miss queue contains the sequential instructions which would be executed in case the branch is not taken. In the event of a branch misprediction, there are instructions already present in the miss queue, and can be sent to the execution units for processing.

2.2 Integer Execution Unit

This unit is responsible for executing all integer data type instructions namely arithmetical calculations, logical calculations, shifts, loads, stores, and branches. Four independent data paths enable upto four integer instructions to be executed per cycle. The instruction mix that is allowed for these four instructions is: 2 arithmetic operations, 1 memory operation (load/store) and 1 control transfer operation (branch).

2.2.1 Combined Working and Architectural Register File

UltraSPARC-III has a working and architectural register file (WARF). This can be regarded as two separate register files. The Working Register File (WRF) consists of 32 64-bit registers, each with three write ports, seven read ports and a 1984-bit write port to transfer data from the Architectural Register File (ARF). The 160 entry ARF has three write ports and a 1984-bit read port to transfer data to the WRF. The ARF has only 156 physical registers, since the GLOBAL0 register GO, of the WRF is always 0. Sixty-four of the 156 ARF registers are eight register windows, each exclusively for the eight *local* registers. Another 64 registers, are eight sets, each shared between the 16 IN/OUT registers.

The processor access the working register file to get the integer operands needed for execution. The file is also written to as soon as results from the execution are known. If an exception occurs, the written results must be undone. This can be accomplished using the architectural register file. Results are not written to the architectural register file until the end of the pipeline, by which time all exceptions have been resolved. We can copy the architectural register file back into the working register file, once we have handled the exception. This gives us a fast, clean way to repair the pipeline state when exceptions do occur.

The WARF lets us remove the result bypass buses from most of the integer execution pipeline stages. Without bypass buses, we could shorten the integer data path and narrow the bypass multiplexing, both of which contribute to a short cycle time.

2.3 Data Cache Unit (on-chip memory system)

This unit comprises of the on-chip caches, namely the level-one (L1) cache. There are three first-level on-chip data caches: data - 64-Kbyte, four-way associative, 32-byte line; prefetch - 2-Kbyte, four-way associative, 64-byte line; and write - 2-Kbyte, four-way associative, 64-byte line.

The major philosophy in designing the on-chip memory system was: achieve uniform performance scaling by scaling both bandwidth and latency. The reason for scaling both bandwidth and latency was that some programs, due to inherent sequential dependence of data, do not have very good ILP and hence, just scaling bandwidth would not help.

Memory scaling issues are three prong: reducing average latency, reducing main memory latency, and scaling on-chip bandwidth.

2.3.1 Memory latency

This is achieved by using a first-level sum-addressed memory data cache [6]. Mixing the memory address adder with the word line decoder for the data cache largely eliminates the address adder's latency. This gives us a linear memory latency improvement.

2.3.2 Main memory latency

For programs which are dominated by main memory latency, two techniques are used: a prefetch catch and an on-chip memory controller. The prefetch cache is a 2-Kbyte SRAM organized of 32 entries of 64 bytes and using four-way associativity with an LRU replacement policy. Data is prefetched into the prefetch cache utilizing 100% of the available main memory bandwidth. When a process needs data, it can just fetch it from the prefetch cache instead of going off-chip to main memory, which is expensive in terms of CPU cycles.

2.3.3 On-chip bandwidth

This is solved using two techniques: wave-pipelined SRAM and a write cache for store traffic.

2.4 Floating Point Unit

This unit contains the data paths and the control logic to execute all instructions relating to floating points, and partitioned fixed-point data type instructions (graphics). At any given time, three data paths can concurrently execute floating point or graphic instructions, one each per cycle from the following classes: Divide/multiply, Add/subtract/compare, An independent division data path.

2.5 External Memory Unit

This unit is responsible for the off-chip memory systems, like the level-two cache (L2) built with off-chip synchronous RAMs (SRAMs), and the main memory system built with off-chip synchronous DRAMs (SDRAMs). The L2 cache controller includes a 90-Kbyte on-chip tar RAM to support L2 cache sizes up to 8Mbytes. The main memory controller can support up to four banks of SDRAM memory totalling 4Gbytes of storage.

The L2 cache controller accesses off-chip L2 cache SRAMs with a 12-cycle latency to supply a 32-byte cache line to the L1-cache. A 256-bit wide data bus between the off-chip SRAMs and the microprocessor delivers the full 32 bytes of data needed for an L1 miss in a single SRAM cycle. The latency to main memory is further reduced by placing the tags for the L2 cache on-chip. This provides an additional advantage, as future designs can use these on-chip tages to build associative L2 caches without a latency penalty.

3 Visual Instruction Set

Graphics functionality is more and more in demand in today's workstations, with applications such as video conferencing, animation, live viewing of events etc. becoming part of everyday life. One of the ways to provide such high functionality was to add additional hardware to the existing hardware, like a graphics card. This can impact system performance, due to the addition of extra hardware, also the additional hardware has to be compatible with the existing hardware. Another solution is to add extra functionality to the existing instruction set architecture to take care of the additional requirements, this removes the need for extra hardware and also results in better overall system performance.

Visual Instruction Set (VIS) is a RISC-like extension to the SPARC V9 instruction set that provides instructions that enhance the graphic and image processing capabilities of the SPARC processors [4]. There is no need to perform memory mapped I/O or to access I/O devices in order to perform multimedia functions.

The implementation of VIS directly on UltraSPARC, coupled with the highly optimized memory system already developed for general purpose computing, allows UltraSPARC to support: Video conferencing, MPEG-2 decoding with full broadcast quality (720x480 pixels, 30 frames/sec), 3D visualization and stripped down systems where the CPU does all required graphics manipulation.

Most operations in VIS act on 4 pixel components in parallel. This coupled with UltraSPARC superscalar capabilities results in very high rates of parallel processing. [5] showed that using these features, MPEG decoding at 720x480x30 resolution could be done entirely in software, which was considered very difficult at that time due to unavailability of resources.

3.1 Data Types

The data types defined in VIS are pixels and fixed data. Pixels consist of four 8-bit unsigned integers contained in a 32-bit word, these are typically used for video images. Fixed data consists of either four 16-bit fixed point components, or two 32-bit fixed point components both contained in a 64-bit word. These data types are used for intermediate results during image processing, when additional precision or dynamic range is required. Also, 16-bit fixed point components can be used for very high quality imaging, like in some medical processing or in color prepress. Some instructions also use fixed data with eight 8-bit components.

3.2 Register Set

The instruction operands are contained in either the current set of 32 64-bit registers in the integer register file or the set of 32 64-bit registers in the floating point register file. Instruction operands in the floating point register file can refer to single or double-precision registers. Pixels are typically contained in single-precision registers. In addition to the two register files, VIS instructions use the Graphics Status Register (GSR). GSR is used for conversion between formats and for memory alignment.

3.3 Instructions

The VIS instructions are classified into conversion instructions, arithmetic/logical instructions, address manipulation instructions, memory access instructions, and a motion estimation instruction.

3.4 Some Applications

Some sample applications of VIS are described in [4]. The applications that were considered were not very large, but were chosen just to illustrate the uses of VIS, and how it can help significantly improve the graphics capability of the existing processor. The applications considered were: Image composition, Changing pixel representation, Slicing 3-D images and Block copy.

3.5 Hardware Implementation

The graphics functional units aptly supplement the datapaths and control sections already needed to implement the floating-point unit (FPU), in UltraSPARC. The main functional units are explained in brief below.

3.5.1 Graphics Adder

The graphics adder is organized as 4 independent 16-bit adders. These do not propagate carries between them, and do not generate overflows.

3.5.2 Graphics Multiplier

The graphics multiplier consists of four 8x16 multipliers. The pipelining of the multipliers approximately corresponds to the one for the floating-point unit.

3.5.3 Pixel Distance

One of the important design requirements was for the *PDIST* instruction to be able to calculate back-to-back *distances* (the *PDIST* instruction is used for motion estimation, which is very important for video compression algorithms like MPEG-2 or H.261).

4 Test and Debug

Test and Debug capabilities in a processor are very important in making sure that the processor meets the quality goals. It is necessary to have a versatile and comprehensive test and debug methadology. The main test and debug features of UltraSPARC-III were implemented such as to provide economical testing during test, debug and manufacturing phases of the chip.

The major standard that is used for testing and debugging is IEEE Std. 1149.1. This was used to create new features and make enhancements with respect to previous UltraSPARC processors to improve the test and debug features on UltraSPARC-III. Some of the additions, were however, specifically requested by product, system and manufacturing groups. Some of the new features that were added are: on-line

debug, access to chip identification registers, dealing with more complex I/Os.

4.1 Test features

The public instructions and some of the private instructions were implemented. A total of 85 instructions with additional logic in the chip core and I/Os provided control for key features such as internal scah, clock, I/O test and charecterization. The support of public and private instructions had to be heavily customized, while being IEEE Std. 1149.1 compliant. The reason for this was the design of very high speed I/Os in UltraSPARC-III. Pubic instructions are mainly used for device identification and circuit board testing. Private instructions are primarily used to gain access to features used to aid in manufacturing based testing, failure analysis and debugging. All test features of the processor are carried out through the test access port.

4.2 Debug features

One of the most important features in debugging is the ability to set breakpoints. UltraSPARC-III has the feature of setting watchpoints, and they can be taken on both virtual as well as a physical address and are fully controllable via software. Another debug feature is clock control. This mechanism allows for the precise stopping of the internal clock, using one of the three methods: scan, a logic analyzer, or using watchpoints. We can now use one of the techniques to probe into the processor and watch out for any potential problems.

4.2.1 On-line Debug

This is a very powerful technique for debugging. This allows read and write control to some of the internal states of the processor while the system is still operating. Using several customised instructions, referred to as Shadow and Mask, this is accompolished. These instructions are part of the RAS (Reliability, Availability, Serviceability) features of the processor and are mainly used for system debug. The speed of operation is up to 10Mhz.

The system groups wanted some visibility into the processor while the system was operating, this included the Program Counter (PC) and some other important architectural registers. These registers were chosen as they would provide the best insight into the processor during system debug phase. Mask and Shadow chain techniques are used for implementing this. Mask chains are simple scan chains that are used to control the logging of data that is eventually accessible through the shadow chain. This is very useful as the mask chain can log the errors and they can be viewed later by the Shadow scan chain. Shadow/Mask combination can successfully detect bus protocol violations.

5 Conclusion

The UltraSPARC-III processor is a high performance superscalar processor capable of issuing four instructions per cycle. It has a deep pipeline (14 stages), which helps in improving the performance, but can be very expensive during mispredictions. This is taken care of by having a miss queue which contains the sequential instructions, and these can be used for immediate processing. The performance is also improved by scaling both the instruction bandwidth and the latency. All these improvements, lead to a significant increase in the clock speed over the earlier generation of UltraSPARC processors, with UltraSPARC-III reaching a clock speed of 1Ghz. UltraSPARC-III also has enhanced test and debug capabilities, which proved to be very useful in keeping the costs down during the design and manufacturing phases of the processor.

References

[1] T. Horel and G. Lauterbach, " UltraSPARC-III: Designing third-generation 64-bit performance," in *IEEE MICRO*, May/June 1999, pp. 73-85.

[2] R. Heald et al., "A 3rd-generation SPARC V9 64-b Microprocessor," *IEEE* Journal of Solid-State Circuits, Vol. 35, No. 11, Nov. 2000.

[3] D. Greenley et al., "UltraSPARC: The Next Generation Superscalar 64-bit SPARC," in *Compcon '95* Dig. Papers, Mar. 1995, pp. 442 - 451.

[4] M. Tremblay, et. al., "The Visual Instruction Set (VIS) in UltraSPARC", *Compcon* Spring 95, March 5-9 1995, San Francisco.

[5] C. Zhou et. al., "MPEG Video Decoding with the UltraSPARC Visual Instruction Set, " Proc. of *IEEE Comcon*, San Francisco, CA, pp 470-75, March 1995.

[6] R. Heald et al., "64-KByte Sum-Addressed Memory Cache with 1.6-ns Cycle and 2.6-ns Latency," *IEEE* Journal of Solid-State Circuits, Nov. 1998, pp. 1,682 - 1,689.

[7] F. Golshan, "Test and On-line Debug Capabilities of IEEE Std 1149.1 in UltraSPARC-III Microprocessor, " Proc. of International Test Conference, 2000