UltraSPARC-II(™): The Advancement of UltraComputing

Gary Goldman and Partha Tirumalai

SPARC Technology Business -- Sun Microsystems, Inc.

2550 Garcia Ave., Mountain View, CA 94043

Abstract

UltraSPARC-II extends the family of Sun's 64-bit SPARC V9 microprocessors, building on the UltraSPARC-I pipeline and adding critical enhancements to boost data bandwidth, hide memory latency, and improve floating-point and multimedia performance. New external cache and interface options allow more flexibility in system implementations. This paper describes the motivation and implementation of UltraSPARC-II's enhancements.

1.0 Introduction

UltraSPARC-II is Sun Microsystems' second generation high-performance, highly integrated superscalar processor implementing the SPARC V9 64-bit architecture [1]. Building on the UltraSPARC-I pipeline, UltraSPARC-II scales up its computation, multimedia, and networking performance. The design was retargeted for 0.35µ CMOS, 5-layer metal technology, and adds functional enhancements to boost data bandwidth, reduce cache-miss penalties, and improve floating-point and multimedia (visual instruction set (VIS)) performance. Clock rates are scaled to 250-300 MHz, to achieve an estimated 350-420 SPECint92 and 550-660 SPECfp92 (2Mbyte cache).

TABLE 1. UltraSPARC-1 and -II Comparison

	UltraSPARC-I	UltraSPARC-II
Architecture	SPARC V9	SPARC V9
Device count	5.2 million	5.4 million
Die size (mm)	17.7 x 17.8	12.5 x 12.5
Operating Frequency	167 MHz	250 MHz
Supply Voltage	3.3 Volts	2.5 Volts
Power dissipation	30 watts	26 watts
Technology	0.5µ CMOS, 4-layer metal	0.35µ CMOS, 5-layer metal

Functional enhancements in UltraSPARC-II include:

- Allowing up to 3 outstanding 64-byte block reads to memory
- Up to 2 outstanding block writebacks
- Extensions for V9 Prefetch instruction
- Multiple external cache configuration and system clocking options.
- Larger maximum External Cache size -- 16Mbytes vs. 4Mbytes.

1.1 UltraSPARC-I Background[1]

UltraSPARC-I was Sun's first 64-bit SPARC V9 processor. Like its 32-bit predecessor, SPARC V8, V9 is an instruction set architecture created by SPARC International. SPARC V9 is strictly upwards binary compatible with V8, while providing improved system performance through features such as 64-bit integers and virtual addresses, additional floating-point registers, and relaxed memory ordering models.

Performance goals required UltraSPARC-I to be a 4-way superscalar design. The four instructions dispatched in a given clock can include most combinations of:

- two integer
- two floating-point/graphics
- 1 load/store
- 1 branch

Only floating-point/graphics instructions and certain branches are candidates for the 4th instruction in a group. Instructions are dispatched in-order, but may complete out-of-order.

UltraSPARC-I and -II include the following features:

- Nine concurrent execution units
- 64-bit/128-bit datapaths
- In-cache dynamic branch prediction
- Precise exceptions and multiple, nested traps
- 16kB 2-way instruction cache with pre-decoded bits
- 16kB non-blocking, direct-mapped data cache
- Separate 64-entry, fully associative instruction and data TLBs.
- Hardware-assist for software-managed TLB miss handling
- 9-entry Load Buffer / 8-entry Store Buffer
- Integrated external cache controller supporting 512Kbytes - 4Mbytes (16Mbytes on -II) of synchronous SRAM.
- 128-bit pipelined and split transaction system bus
- Snooping and directory-based cache coherency support

1. This section is taken from UltraSPARC(TM): The Next Generation Superscalar 64-bit SPARC [2]

A block diagram is shown in Fig. 1.

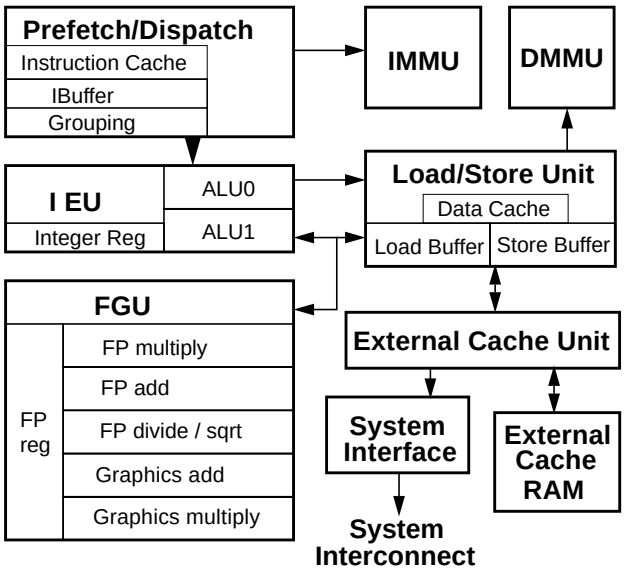

FIGURE 1. UltraSPARC Block diagram

UltraSPARC-I and -II use a 9-stage pipeline pictured in Fig. 2. Three additional stages are added to the integer pipe in order to make it symmetrical with the floating-point pipe. This simplifies pipeline synchronization and exception handling.

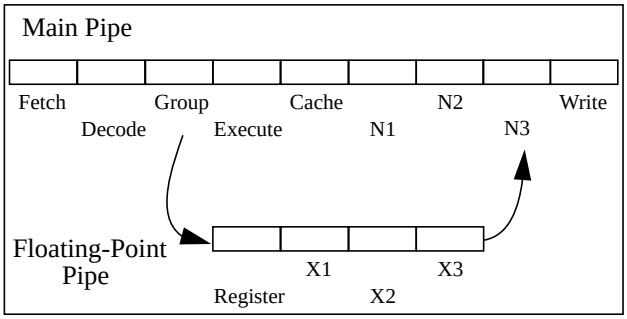

FIGURE 2. UltraSPARC-I Pipeline

The ***fetch*** stage is used to read a set of 4 instructions and branch prediction information from the instruction cache (I$).In the ***decode*** stage, these instructions are further decoded and placed in the 12-entry instruction buffer. In the ***group*** stage, the Integer Register File (RF) is accessed

at the same time as the 4 oldest entries in the instruction buffer (or the instructions read directly from the I$) are considered for dispatch.

At this point, the Integer and Floating-point/Graphics pipes differ. The next stage in the Integer pipe is the ***execution*** stage, where all integer instructions are executed and virtual addresses are calculated. During the ***cache*** stage, the data cache (D$) and the data TLB are accessed using the virtual address. Branches are resolved in this stage.

In the ***n1*** stage, D$ misses are detected and load data is available for cache hits. Accesses that miss the D$ are enqueued on the load buffer. In the ***n2*** stage, the Integer pipe waits for the Floating-point pipe. In the ***n3*** stage, the two pipes converge and all traps are resolved. In the ***write*** stage, results are written to both Register Files and instructions are committed.

Like integer instructions, floating-point instructions are dispatched in the ***group*** stage. In the ***register*** stage, the floating-point RF is accessed. The ***x1***, ***x2***, and ***x3*** stages represent the Floating-point execute stages.

2.0 Microarchitecture enhancements for UltraSPARC-II

While the UltraSPARC-II design is based strongly on UltraSPARC-I, performance and system-design goals necessitated critical enhancements. Floating point performance is boosted by implementing software-controlled prefetch and increasing the available memory-interface bandwidth of the processor. The additions account for an estimated 10% improvement in SPECfp92, with a negligible impact on die area.

System design realities required additional flexibility in the external cache and system interface. Two types of SRAM may be used for the external cache. This provides system designers a wide array of price/performance options, and the ability to create high-end systems targeted to specific application environments.

3.0 Data prefetch

For many floating-point intensive applications, the main performance bottleneck is providing enough data from the caches and/or memory to keep the FP execution units busy. For applications with small to medium data-sets (up to a few Mbytes), the data can be provided from the on-chip or external data caches at a sustained rate of 8-Bytes/cycle [3]. For larger data-sets, external cache misses become limiting. In UltraSPARC-I, each access to a new 64-byte block would lock up the load buffer for the duration of the memory access. This would in turn stall the pipeline on the “use” instruction for that data.

UltraSPARC-II employs the SPARC V9 Prefetch instruction to alleviate this bottleneck.The instruction is handled much like a load, except that data is prefetched into the external L2 cache (E$) in 64-byte blocks, and not written to on-chip D$ or a register. This allows the load buffer to retire the prefetch once it has accessed the E$, regardless of hit or miss. As a result, prefetch “misses” are non-blocking; subsequent loads and stores which hit the caches may complete while the prefetch is outstanding to memory. Concurrently, the External Cache Unit sends the miss request on to the memory system.

Note that it might seem possible to get the same prefetch effect by scheduling loads sufficiently ahead of the “use”, i.e, to account for main memory latency in instruction scheduling. However, the load buffer design requires loads to complete in FIFO order, although loads may complete out-of-order with respect to other types of instructions. While any load miss is outstanding to the memory subsystem, subsequent load hits are blocked, and the load buffer “locks up”. Prefetch avoids this lock up since the load buffer need not wait for the memory access before retiring a prefetch.

The SPARC V9 Prefetch instruction defines several variants, to fetch data in preparation for reading or for writing. UltraSPARC-II maps these into two types of prefetch:

1. Prefetch for coherent read access. If the desired block is not valid in the E$, a read-to-share request (P_RDS_REQ) is sent to the UPA bus.[1]

1. UltraSPARC, and the UltraSPARC Port Architecture (UPA) interconnect use a MOESI protocol for maintaining system-wide cache coherence [4].

2. Prefetch for coherent write access. If the desired block is not in Modified or Exclusive state in E$, a read-to-own request (P_RDO_REQ) is sent to the UPA bus.

4.0 Multiple-outstanding read requests

UltraSPARC-I was limited to a maximum of one coherent read and one dirty-victim writeback request outstanding to the memory subsystem. UltraSPARC-II extends this to three reads and two dirty-victim writebacks. When combined with prefetch, this significantly boosts E$ miss handling bandwidth, since memory accesses can effectively be pipelined.

For a typical UltraSPARC-based desktop system, memory latency is about 30 clock cycles from request to data (pin to pin, @ 250MHz). Accounting for overhead in the processor (i.e., delay in sending a fourth read request when the first request completes), peak demand bandwidth would be 1.33 GBytes/sec:

$$BW = \frac{\text{E\$ line size}}{\left(\frac{\text{Tmemory + Toverhead}}{\text{N outstanding}}\right)} \times \text{Clock rate}$$

$$= \frac{64}{\left(\frac{30 + 6}{3}\right)} \times 250\ \text{MHz} = 1.33\ \text{GBytes / sec}$$

This would completely saturate the UPA system data bus, which also has a peak bandwidth of 1.33 GBytes/sec @ 83.3 MHz.

UltraSPARC-II also supports up to two outstanding dirty-victim writebacks. This is sufficient to balance the three outstanding reads, for codes exhibiting from 0% to 100% dirty victimization on reads. In the extreme of 100% victimization, a typical memory system design would be saturated with two each of reads and writebacks outstanding. In the "typical" case of 50% writebacks, providing buffering for the second writeback eliminates occasional stalls due to non-uniform writeback patterns.

Consider execution of a DAXPY[1] loop in a typical system with and without the UltraSPARC-II extensions. This loop performs two loads, one store, and two floating point operations for each eight bytes of results:

$$Y[i] = A*X[i] + Y[i]$$

With two floating-point issue slots but only one memory-op issue slot per instruction group, it's clear that memory accesses dominate execution time. For this example, we presume that data is not initially E$-resident. Assuming unit-stride vectors, this generates two E$ misses and (on average) one E$ writeback per 64-bytes of results.

Without UltraSPARC-II's extensions, loop time is determined by the delay of serial E$ misses, plus processing time by the CPU in between the E$ misses. For a typical deskside (server) computer[2], this would be ~144 cycles. The breakdown is:

T_{loop} = $(T_{mem\_access} + T_{E\$\_Fill} + T_{load\_store})$
$\approx 102 + 18 + 24 = 144$ cycles

where

T_{mem_access} = Nominal read-miss latency for two back-to-back E$ misses, where memory latency (for a server-type system) is about 45 CPU cycles.

$T_{E\$\_Fill}$ = E$-busy time while miss data is written into E$, before loads & stores can consume the data. (One of the E$ fills can be hidden during the next miss's memory access.)

T_{load_store} = load & store processing time in between E$ misses. This is an idealized approximation for 16 loads and 8 stores.

With the extensions, loop speed is limited by E$ busy time, not memory latency. There are two E$ fills, plus one writeback -- and the writeback uses E$ cycles which can't be hidden during E$ misses. (In the prior example, writebacks were performed while the processor was stalled on a load-miss). Two E$ pipeline slots are now used for prefetch cache-lookup as well, assuming the loop is unrolled eight times to cover 64 bytes/vector/iteration:

T_{loop} = $(T_{E\$\_fill} + T_{E\$\_writeback} + T_{load\_store} + T_{prefetch})$
$\approx 30 + 10 + 24 + 2 = 66$ cycles

1. DAXPY: Double-precision AX +Y vector operation. The inner loop of Linpack.

2. E.g., a multiprocessor system employing UltraSPARC-II @ 250 MHz, using the UPA bus interface running at 83.3 MHz.

The ideal case predicts a 118% speedup (1 - 144/66) for this loop; In simulations, we see actual loop times of 165 and 84 cycles for the two cases, a 96% speedup. The bandwidth demanded of the memory system is 571 Mbytes/sec for the 84-cycle loop.

4.1 Compiler support for prefetch

A key problem that the compiler has to deal with is the basic question of when to use and when not to use prefetch. Indiscriminate use of prefetch will slow down some loops and accelerate others yielding a less than optimal result. Prefetch should be used for loops that miss in the L2 cache and should *not* be used for loops that hit in L2 (or L1). At compile time a guess has to be made as to whether a given loop is a candidate for prefetching or not. Static program analysis, information obtained through profile feedback, command line options and directives embedded in the source are some methods used to assist the compiler in making this decision.

Another problem is that prefetch instructions consume the memory issue slot in a group. Since UltraSPARC-II has two integer units and two floating point functional units (an adder and a multiplier) but only one memory issue slot, many loops are limited by the number of memory operations that have to be executed. Therefore, to utilize this valuable resource most efficiently, a minimum number of prefetches should be issued. This can be done by unrolling the loop a sufficient number of times such that one prefetch per unrolled loop fetches the data needed by multiple loads in the same unrolled body. For example, if a double precision vector is being accessed with unit stride in the given loop, the loop is unrolled 8 times because one prefetch can bring in 64 bytes or 8 double precision values.

The compiler has to deal with several problems with this form of unrolling. High degrees of unroll result in a higher number of remainder iterations when the loop is exited, which usually execute with less efficiency. When multiple vectors with different element sizes or strides are being accessed, a common degree of unroll has to be determined for all of them. In some cases, the stride may be unknown at compile time or it may be variable. These cases may force the compiler into making conservative assumptions and result in more prefetches than are needed. For these reasons, the impact of executing more than one prefetch per block is minimized. A prefetch which hits the E$, or which matches an outstanding prefetch, is immediately retired by the hardware. The main cost is the loss of one E$ pipe cycle for each extraneous prefetch.

The compiler must also schedule prefetches sufficiently ahead of the first referencing load instruction to account for memory latency, to assure that all loads hit the L2-cache. This is a function not only of memory latency and loop execution time but also of the address alignment of the streams being prefetched. Since the address alignment is usually unknown, additional spacing between prefetches and referencing loads is often required. For large prefetch-ahead distance (e.g, fast, single-vector loop in a high-latency memory system) UltraSPARC-II can enqueue two 64-byte reads internally, in addition to the three outstanding to the memory system, before stalling the load-buffer.

The E$ is burst-filled for prefetch misses. There is no requirement to minimize latency of the first sub-block to the CPU, so data is buffered in the UltraSPARC Data Buffer until it can be written into the E$ in the minimum possible number of cycles. This reduces usage of E$ cycles for installing prefetched data in the E$, as compared to load or store misses. It does not increase the prefetch-ahead distance requirement, nor impact loop start-up.

5.0 External cache modes

Two types of synchronous static RAM are supported for use in the external cache:

- Pipelined SRAM clocked at the CPU frequency provides the lowest latency and highest bandwidth.
- Flow-through access SRAM, clocked at one-half the CPU rate, allows different cost options (with only a modest performance loss) as well as larger cache sizes.

5.1 1-1-1 mode

For best performance, the E$ interface is directly scaled up from UltraSPARC-I, up to 250MHz. This utilizes 250MHz pipelined synchronous SRAMs, with the same architecture, package and pinout as those used for UltraSPARC-I [5]. This is referred to as “1-1-1” mode, since one CPU

cycle each is required for address, array-access, and data return. Read latency (pin-to-pin) is three cycles, with read or write bandwidth of 16 bytes per CPU cycle. One “dead” cycle is required on the data bus when switching between reading and writing, for electrical reasons.

These SRAMs use a late-write architecture, in which write data trails the address by one cycle. Since the data bus is bidirectional, this alleviates one stall cycle (due to data bus contention) in a write following a read. This is apparent in Fig. 3; note that data Dn+2 trails address n+2 by a cycle.

5.2 2-2 mode

For flexibility of system design, UltraSPARC-II also supports flow-through access (more specifically, register-latch type) synchronous SRAMs. For a 250MHz CPU, these SRAMs have ~7ns access time from the address register, operating at 125MHz. The read access is essentially flow-through from the address register, though output data is controlled by a latch to improve output hold time. This is referred to as “2-2” mode, since SRAM reads take two CPU cycles to transmit the address to SRAM, and two cycles to access and return data to the processor. Write data follows the address by one (SRAM) cycle, which eliminates bus contention when switching from reads to writes.

This E$ option provides three essential advantages:

1. Technology and cost: these SRAMs are approximately one generation “older” in SRAM design than the ultra-fast 250MHz parts. They use an industry-standard design which differs from the pipelined SRAMs only in the output register. This assures a stable supply of parts from multiple vendors.
2. Headroom for scaling speed: As the UltraSPARC-II clock rate is further scaled up over time, the SRAMs must keep up. In addition, external switching rates must scale. In both aspects, commercially available SRAMs are not expected to match processor clock rate improvements. Register-latch SRAMs provide a solution for CPU clock rates up to 300MHz and beyond.
3. Density: Larger E$ sizes are possible with these relatively slower SRAMs. In general, 2-2 mode caches can be four times the size of 1-1-1 caches. Cache-size sensitive applications, as well as systems with large memory latency, can realize better performance from 2-2 mode despite its latency and bandwidth penalties.

As seen in Fig. 3, E$ read latency is increased by one cycle relative to 1-1-1 SRAM mode. Peak bandwidth is reduced from 16 bytes/cycle to eight. However, no dead cycles are required to switch between reading and writing the SRAMs; this increases the usable bandwidth. In the 1-1-1 mode, two stall cycles are needed between read and write cycles to accommodate the databus turnaround time. Overall performance degradation, relative to 1-1-1 SRAM mode, is 4.5% SPECint92 and 8.1% SPECfp92 (estimated).

6.0 System interface and clock domains

UltraSPARC-II is module-level pin compatible with UltraSPARC-I, and can replace or cohabit with -I modules in multiprocessor systems. The UPA interface is synchronous, requiring the CPU clock frequency to be an integer multiple of the system clock. For UltraSPARC-II, system:CPU clock ratios of 2:1, 3:1 and 4:1 are supported, to cover a wide spectrum of processor and system speeds. Examples are shown in table 2.

Unlike UltraSPARC-I, the -II doubles the clock frequency internally, allowing a lower speed clock to be delivered to the pin; e.g., 125MHz clock for a 250MHz CPU.

TABLE 2. System and Processor Clock Ratios

CPU freq (MHz)	system freq (MHz)	clock ratio	implementation
167	83	2:1	emulating UltraSPARC-I
250	83	3:1	standard configuration
300	75	4:1	highest performance processor; module can be added to existing 83MHz system
300	100	3:1	highest-performance system

7.0 Summary

UltraSPARC-II’s mission is to extend the UltraSPARC family’s industry-leading capabilities and to increase flexibility for a wide range of system implementations. This is accomplished by building on the solid foundation of UltraSPARC-I, and exploiting technology improvements for a 50% gain in clock speed. Incremental capabilities were added in the cache and memory interfaces which,

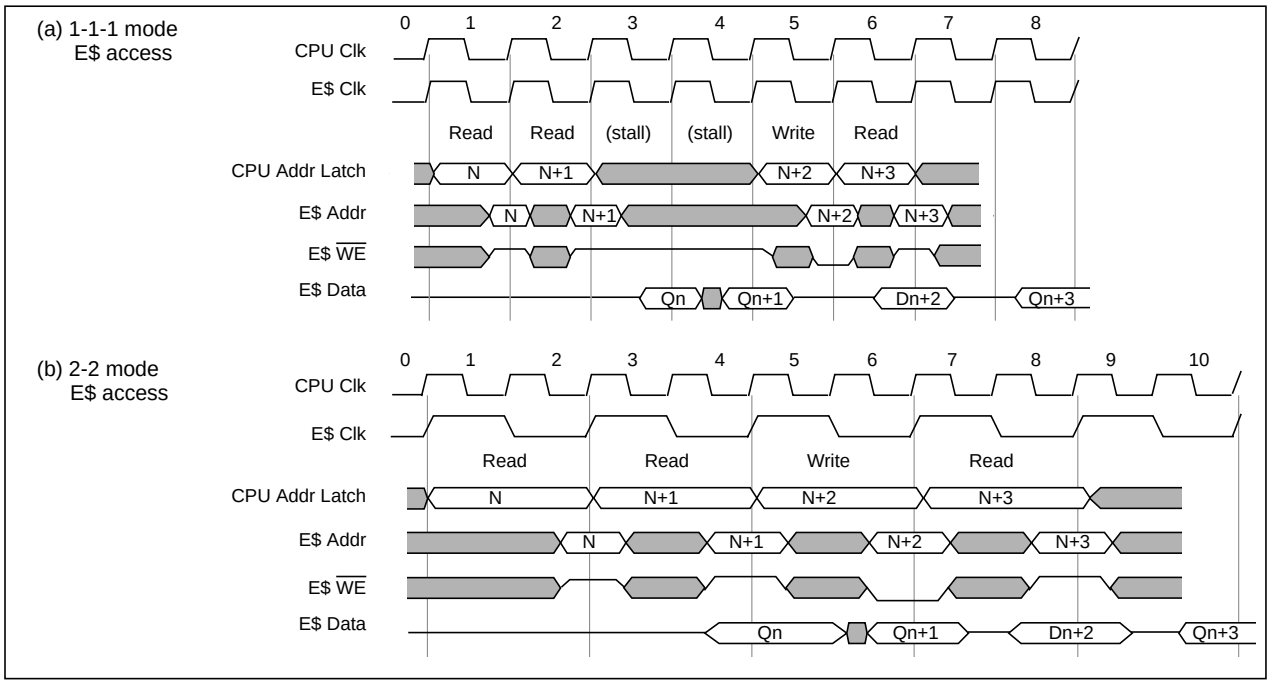

FIGURE 3. Waveforms of E$ SRAM access: (a) 1-1-1 Mode and (b) 2-2 Mode. Each waveform depicts read/read/write/read accesses to illustrate latency, bandwidth, and read-write turnaround penalties

while inexpensive in design time and die area, address the need for greater floating-point and VIS performance.

8.0 Acknowledgments

The design and implementation of UltraSPARC-II relied on the efforts of many individuals in logic design, circuit design, CAD, verification, emulation, and system and product engineering. The authors would like to thank them for making this chip a reality. We would particularly like to thank the logic design team of Eric Anderson, Jim Bauman, P.K. Chidambaran, Philip Ferolito, Tim Goldsbury, Suresh Gopalakrishnan, Sailendra Koppala, Ramachandra Kunda, Richard Landes, Hung-yi Lee, Al Martin, Nasima Parveen, Bruce Petrick, Duy Pham, Samir Sanghani, Gregory Schulte, Paul Serris, Manish Singh, John Sullivan, Michelle Wong, David Yee, and Robert Yu.

We would also like to acknowledge Les Kohn and Marc Tremblay for their contributions to the microarchitecture design.

Finally, we would like to acknowledge the leadership of Andy Charnas, Ron Melanson, Mike Splain, Hem Hingarh, and Anant Agrawal, for their guidance and support in implementing UltraSPARC-II.

9.0 References

[1] D. Weaver and T. Germond editors, “*The SPARC Architecture Manual, Version 9*”, Prentice Hall, 1994.

[2] D. Greenley, et al, *“UltraSPARC: The Next Generation Superscalar 64-bit SPARC”*, 1995 Compcon Proceedings, IEEE

[3] P. Tirumalai, et al, "*UltraSPARC: Compiling for maximum floating point performance*", submitted to Compcon '96, Santa Clara, CA.

[4] “*UltraSPARC-I User’s Manual*”, Sun Microsystems, Inc., 1995

[5] JEDEC Std. 21-C, Section 3.7.8.3 "*32K to 512k BY 32 & 36 SRAM in 7X17 BGA*"