ROSS TECHNOLOGY, INC.

WHITE PAPER

hyperSPARC: The Next-Generation SPARC

Introduction

Several years ago, ROSS Technology set itself a goal: to develop the highest-performance microprocessor in the industry. This microprocessor had to have the following: a high degree of manufacturability, upgradeability from prior generation SPARC CPU modules, and 100% binary compatibility with existing SPARC software.

The result of this ambitious endeavor is hyperSPARC™. The hyperSPARC program achieved every aspect of its stated objectives:

- 3 to 5 times the performance of prior-generation SPARC implementations
- Completely SPARC compatible (Version 8 Architecture, Reference Memory Management Unit (MMU), Level 2 MBus)
- Manufactured using proven CMOS technology and offered in Multi-Die Packaging (MDP) form
- Implemented as a SPARC-standard MBus module and interchangeable with existing modules

But hyperSPARC is more than a new microprocessor: hyperSPARC is a milestone. It enables the manufacturers of SPARC systems and boards to continue their market-leading price/performance pace, while providing software writers with a vehicle for exploiting multilevel parallel processing. hyperSPARC's architecture allows it to be used in a wide range of machines, from mainframes to minicomputers, servers to desktops, laptops to notebooks, making its impact in the SPARC world significant.

General Description of Product

hyperSPARC is designed as a tightly coupled chip set and implemented as a SPARC MBus module using Multi-Die Packaging (MDP). Each hyperSPARC CPU supports either 256, 512, or 1024 Kbytes of second-level cache, and each module contains one or two CPUs. The chip set is comprised of the RT620 Central Processing Unit (CPU), the RT625 or RT626 Cache Controller, Memory Management, and Tag Unit (CMTU), and four RT627 Cache Data Units (CDUs) for 256 Kbytes of second-level cache, four RT628 CDUs for 512 Kbytes of second-level cache, or eight RT628 CDUs for 1 Mbyte second-level cache. The chip set can be configured for uniprocessing (Level 1 MBus) or multiprocessing (Level 2 MBus). Figure 1 is a block diagram of the hyperSPARC chip set.

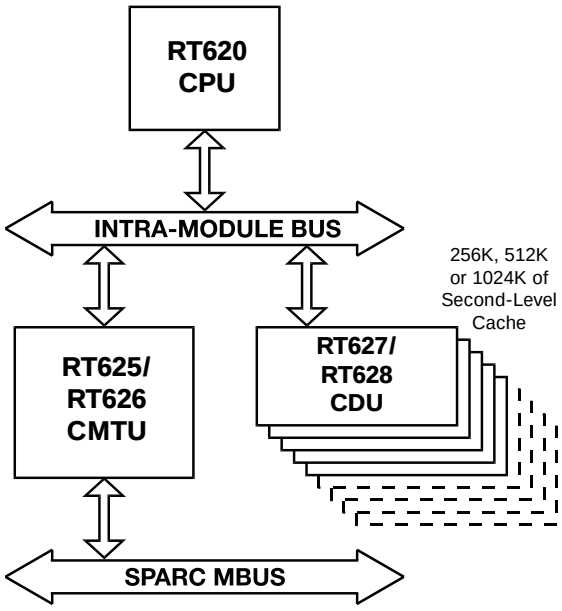

Figure 1. hyperSPARC Chipset

The RT620 is the primary processing unit in hyperSPARC. This chip is comprised of an integer unit, a floating-point unit, and an 8-Kbyte, two-way, set-associative instruction cache. The integer unit contains the ALU and a separate Load/Store data path, constituting two of the chip's four execution units. The RT620 also includes the floating-point unit and a branch/call unit (for processing control transfer instructions). Two instructions are fetched every clock cycle. In general, as long as these two instructions require different execution units and have no data dependencies, they can be launched simultaneously. (It is also possible to fetch and dispatch two floating-point adds or two floating-point multiplies at a time.) The RT620 contains two register files: 136 integer registers configured as 8 register windows, and 32 separate floating-point registers in the floating-point unit (see Figure 1).

hyperSPARC's second-level cache is built around the RT625 or RT626 CMTU, a combined cache controller and memory management unit that supports shared-memory and symmetric multiprocessing. The RT625 cache controller portion supports 256 Kbytes of cache, made up of four RT627 CDUs. The RT626 CMTU supports 512 Kbytes or 1 Megabyte of cache (four or eight RT628 CDUs, respectively). The cache is direct-mapped with 4K tags (RT625) or 16K tags (RT626). The cache is physically tagged and virtually indexed so that the CMTU's cache coherency logic can quickly determine snoop hits and misses without stalling the RT620's access to the cache. Both copy-back and write-through caching modes are supported.

The MMU is a SPARC Reference MMU with a 64-entry, fully set-associative Translation Lookaside Buffer (TLB) that supports 4096 contexts. The RT625 contains a read buffer (32 bytes deep) and a write buffer (64 bytes deep) for buffering the 32-byte cache lines in and

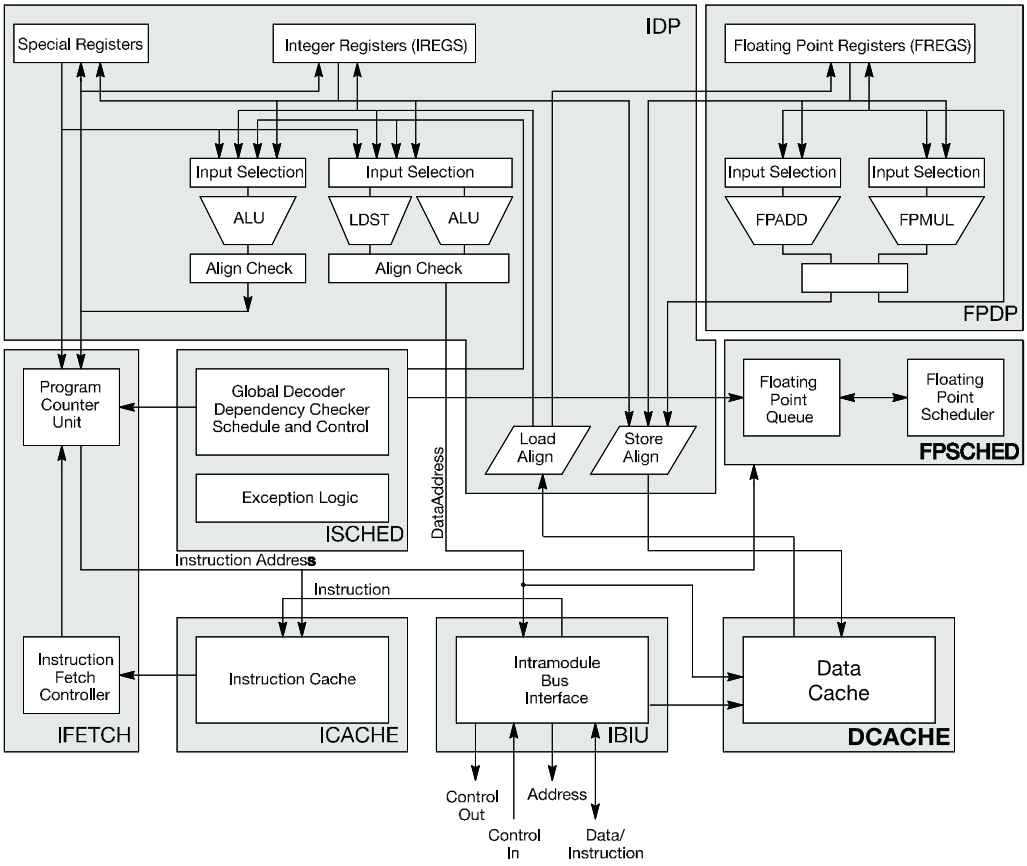

Figure 2. RT620 CPU - Major Functional Blocks

out of the second-level cache. It also contains synchronization logic for interfacing the virtual Intra-Module Bus (IMB) to the SPARC MBus for asynchronous operation (see Figure 3).

The RT627 is a 16K x 32 SRAM that is custom-designed for hyperSPARC's cache requirements (256-Kbyte configuration). It is organized as four arrays of 16-Kbyte static memory with byte-write logic, registered inputs, and data-in and data-out latches (see Figure 4). The RT628, used in 512-Kbyte and 1-Mbyte cache versions, is organized as four arrays of 32 Kbytes each. The RT627 and RT628 provide a zero-wait-state cache to the CPU with no pipeline penalty (i.e., stalls) for loads and stores that hit the cache. The RT627 is designed specifically for hyperSPARC, so it doesn't require glue logic for interfacing to the RT620 (CPU) and the RT625 (CMTU). The RT628 requires no glue logic to interface to the RT620 and RT626.

The microarchitecture of hyperSPARC boasts classic RISC and superscalar features for improving instruction processing throughput. hyperSPARC also employs architectural features that differentiate it from other next-generation microprocessor designs. The following sections highlight some of hyperSPARC's most important attributes.

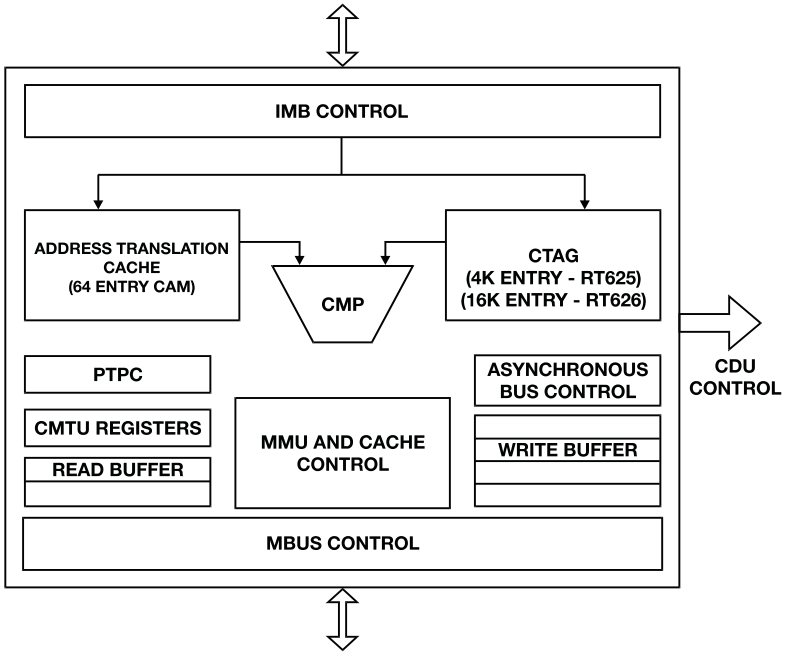

Figure 3. RT625 CMTU

Multilevel Parallelism

hyperSPARC's high performance is mainly due to its parallel program execution, which is based on the idea that software tasks can be dissected into pieces at several levels, and can run concurrently. Hardware can be designed to take advantage of the parallelism offered by software. hyperSPARC was designed with this in mind, taking advantage of the nature of software.

At the top level of hyperSPARC's parallel processing model is the industry's most efficient use of shared-memory multiprocessing support. This VLSI hardware support for connecting multiple CPUs provides a cost-effective solution for creating a tightly coupled multiprocessing system. Combining this configuration with a symmetric, multi-threading operating system provides users with a powerful computing node that has many times the performance of a single-CPU system.

Applying the parallel processing model to the microprocessor leads to today's most sophisticated approach to microprocessor architecture: superscalar. Just as multiple CPUs can work in parallel to tackle several tasks at the same time, the multiple processing elements within each of these CPUs can simultaneously execute several instructions. This concept of duplicating pipeline and other processing logic within the IC, which allows multiple instructions to be fetched and launched, is well suited to the simpler RISC design structures.

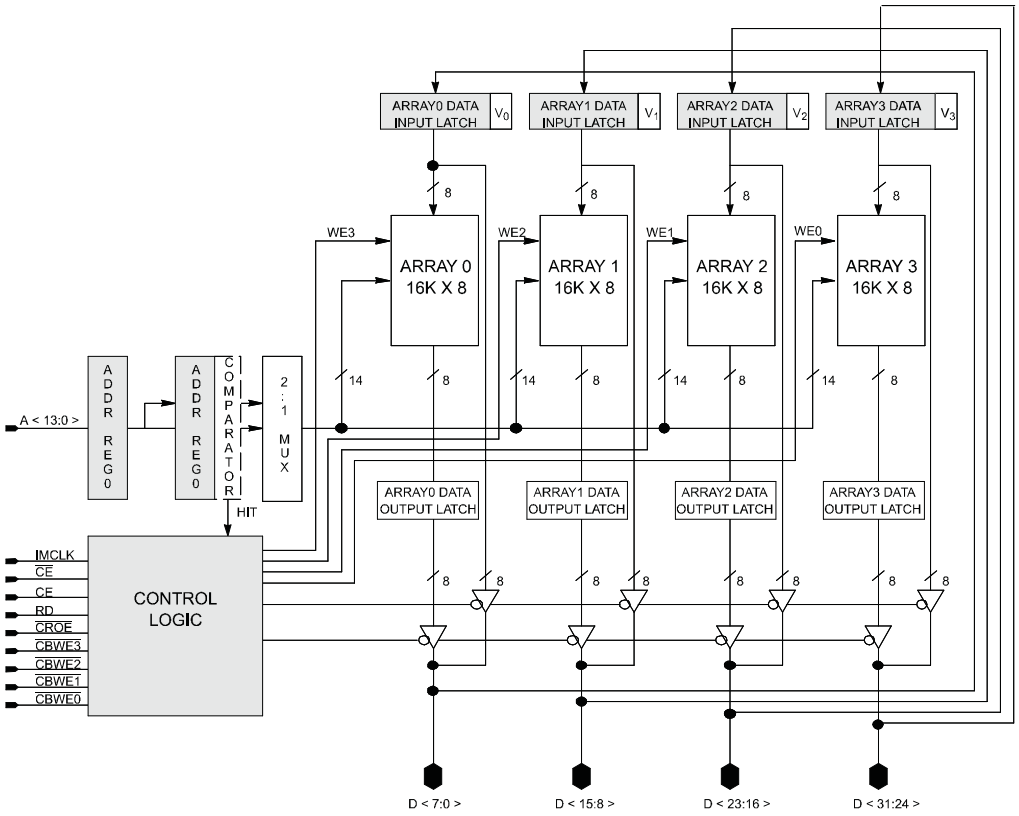

Figure 4. RT627/RT628 Logic Block Diagram

The combination of parallel program execution at both the process and instruction level is at the heart of the hyperSPARC architecture. The highly pipelined processing paths for both the integer and floating-point units add additional processing power to the hyperSPARC architecture.

High Frequency of Operation

Fundamentally, hyperSPARC is built for speed. In order to facilitate high-clock frequencies, particular attention is paid to the five-stage integer and floating-point pipelines, keeping them simple and well-balanced. Each stage of the pipeline is carefully partitioned so that the number of gates per stage are similar, making it easier to do process shrinks for scaling to higher clock rates. In addition, only a single rising clock edge is used to propagate instructions through the pipeline stages, because using both clock edges creates more complex timing issues, especially for critical paths.

hyperSPARC's performance scales independently of the external bus (MBus). The hyperSPARC chip set is partitioned to allow synchronous or asynchronous operation through synchronization logic contained in the RT625 and RT626. This decoupling of the CPU bus from the external bus allows scaling of hyperSPARC's clock frequency independent of the memory and I/O subsystems. This provides for longer product life cycles because upgrades to higher-performance hyperSPARC modules do not require hardware changes to the underlying system design.

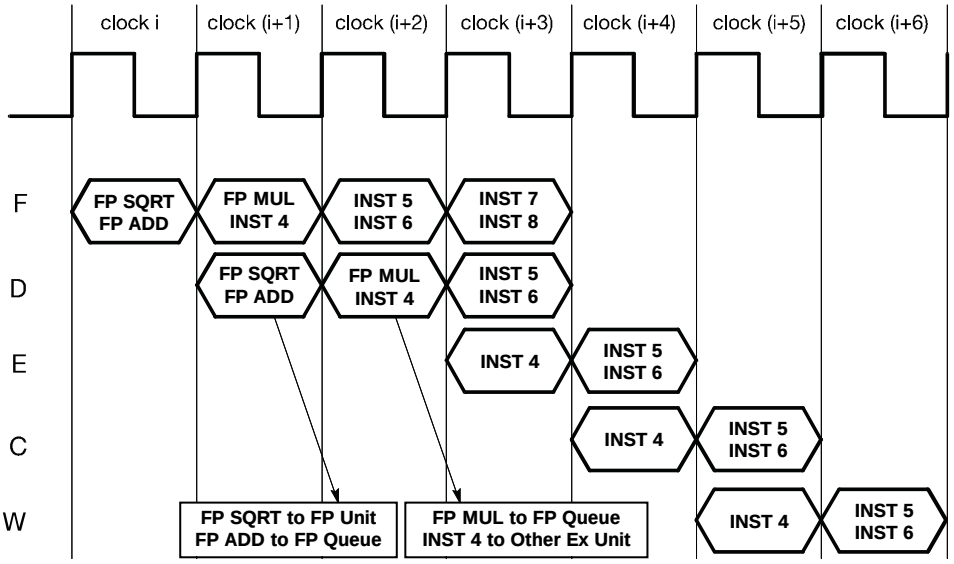

Figure 5. Floating-Point Instruction Launching and Its Effect on the Integer Pipeline

Instruction Scheduling

Instruction scheduling and dispatching is a critical portion of any superscalar design. Optimal instruction scheduling includes minimizing both pipeline stalls and minimizing conditions that prevent simultaneous instruction launching. These problems are costly for any RISC machine, but even more so when multiple instructions are being held.

All superscalar processors are not the same. The ability to fetch and launch multiple instructions is only a valuable asset if it is frequently used. Compilers can help reduce the occurrences of instructions that can't be launched together by being aware of hardware implementation. However, the software can only optimize for the hardware; it is still the job of the microprocessor to reduce the occurrence of sequential instruction launches.

hyperSPARC is partitioned into four execution units in order to facilitate parallel processing of major instruction types. These execution units are the load/store unit, branch/call unit, integer unit, and floating-point unit. The floating-point unit is actually comprised of a queue and two parallel pipelines: an adder and multiplier.

hyperSPARC fetches two instructions every clock cycle and evaluates them for simultaneous launch. hyperSPARC's primary scheduler is invoked for this evaluation and determines the hardware resources required for processing the instructions, as well as any data dependencies. This critical juncture in the instruction processing path exposes the strengths and weaknesses of the microarchitecture.

Poorly architected designs require more frequent splits of instruction groupings. Internal constraints, such as bus design/bandwidth and number of read/write register ports, some-

times make it impossible to schedule multiple-instruction launches. These design constraints manifest themselves in many ways, most often by restricting simultaneous launches based on the types of instructions, and/or order of the instructions, within groupings. The result is frequent sequential (instead of simultaneous) instruction launches.

hyperSPARC's ability to launch multiple instructions simultaneously is not restricted by the order and type of instructions within groupings. Sequential launch is required when there are resource conflicts or data dependencies. But, unlike other superscalar designs, any instruction can occupy any position in the grouping and still be considered for simultaneous launch.

hyperSPARC also provides special support for launching floating-point instructions. The hyperSPARC floating-point unit employs two queues: a pre-queue and post-queue.

The post-queue maintains information on instructions currently in execution in either the floating-point adder or multiplier units. This information includes the instruction type, address, and stage of the pipeline for any given clock, and it is required for exception handling to recover the instructions aborted when the floating-point pipeline is flushed due to a trap. This principle is extended to a floating-point pre-queue, which holds the same information for up to four instructions aborted when the floating-point instructions are pending execution (a floating-point buffer of sorts). The significance of this pre-queue is that it allows floating-point instructions to be sent off to the pre-queue from the normal integer (i.e., load/store) instruction stream. The hyperSPARC scheduler is capable of fetching and dispatching any two floating-point instructions at a time, sending both to the pre-queue in the same clock cycle. (If the floating-point unit is not busy, one of the instructions actually bypasses the pre-queue and begins final decode and execution immediately; the other is placed in the pre-queue.) The integer pipeline proceeds uninterrupted to fetch, decode, and execute more instructions in the next clock cycle. For example, in Figure 5, the FP MUL, which requires the multiplier occupied by the FP SQRT, is offloaded to the FP pre-queue so that the integer unit can continue instruction fetching and launching. This is made possible by hyperSPARC's dual-level instruction decoding, which offloads final instruction decode to the floating-point unit itself.

Integer multiplication/division is new to SPARC (Version 8). Some next-generation CPUs offload this integer processing to the floating-point unit. This results in more contention for this execution unit, further limiting the cases in which simultaneous instruction launch is possible, and imposing an added burden on the compiler and assembly-level programmers who wish to write software for optimum performance. In hyperSPARC, integer multiplies and divides are executed in the integer ALU, removing this workload from the floating-point unit. As more and more applications take advantage of these functions, hyperSPARC will not compromise floating-point performance.

Multiprocessing

Multiprocessing is the key to dramatically higher performance from existing silicon technology. hyperSPARC provides a glueless, standard interface for tightly coupled multiprocessing system architectures.

Snoop Mechanism

hyperSPARC provides a high-performance snoop mechanism to facilitate efficient data transfers between processors. In a write-invalidate protocol, such as the one implemented in Level 2 MBus, caches residing on a shared bus must check or "snoop" each address request to shared memory space. If a cache owns the cache line at the address being requested, it can respond to the request by copying the data to memory (which will later forward the data to the requesting cache) or it can supply the data directly to the requesting processor (called direct data intervention). In the case of a direct data intervention transfer, the cache supplying the data must prevent memory from obtaining the bus and responding to the request.

The SPARC architecture allows a window of MBus clock cycles within which a cache must assert the Memory Inhibit (MIH) signal if it owns the requested cache line (i.e., there is a snoop hit). That window is A + 2 cycles to A + 7 cycles; the "A" representing the cycle in which the address of the cache line being requested is placed on the MBus. hyperSPARC, implementing this second-generation multiprocessing design, responds on snoop hits with MIH in the A + 3 cycle. This means that memory is free to respond beginning A + 4. Using the full window allowed by MBus would impose a three-cycle penalty for every memory access. Responding quickly-even though the MBus specification offers more relaxed timing-enables a very high-performance memory subsystem to be built around hyperSPARC.

Cache Architecture

hyperSPARC is designed as a chip set to take advantage of the fact that processors and caches perform best when they are tightly coupled. Our designers' understanding of the CPU's relationship with the cache is demonstrated in the RT620's design, which imposes only a one-cycle primary cache miss penalty. A pipeline stage is allotted in the RT620 for accessing the second-level cache so that no stall in CPU throughput is realized if the on-chip cache is missed and the second-level cache is hit.

The RT620's pipeline is a six-stage pipeline, as shown in Figure 6. The fourth stage of the pipeline is the Cache stage, which is a built-in recognition of the latency of accessing the second-level cache. Load and Store instructions cause the RT620 to initiate two accesses: one to the on-chip 8-Kbyte instruction cache and, at the same time, one to the second-level cache. If the address for the instruction is found within the on-chip cache, the access to the second-level cache is canceled and the instruction is available at the Decode stage of the pipeline. If there is a miss on the internal cache, and a hit on the second-level cache, the instruction will be available after a one-cycle miss penalty that is built into the pipeline. The significance of this design is that it allows the pipeline to proceed uninter-

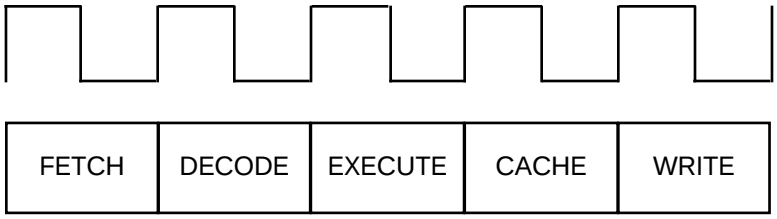

Figure 6. The hyperSPARC Pipeline

rupted as long as the instruction accesses hit either the on-chip cache or the second-level cache, which is 90% and 98% of the time, respectively, for typical workstation applications. Since the integer and floating-point pipelines mirror these five stages for reasons of architectural balance and ease of exception handling, this design enables the RT620 to achieve its high throughput rate at speeds that would not otherwise be possible.

Pipelining and Data Forwarding

The RT627 and RT628 cache data units (CDUs) borrowed from microprocessor architecture by implementing data forwarding and a unique single-stage pipeline. This pipeline design allows the CDUs to keep up with the clock rates of the processor, which includes latching the data and writing it into the RAM core within a period shorter than 10 ns. As clock rates escalate, this task becomes increasingly difficult for an SRAM without moving to smaller, lower-yielding silicon geometries.

For writes into the CDUs, however, only the address and data are latched during the write cycle. The RT620 is then free to continue normal instruction processing. The most time-consuming portion of the transaction, the write into the RAM core, is delayed until the next write access, where there will be a guaranteed cycle available for this action.

The obvious drawback of this implementation is the possibility of a read of the data being held in the latches before the RAM core is updated. The RT627 and RT628 address this using data forwarding. A comparator incorporated in the CDU compares the address of a pending write with the incoming read address. If a match occurs, data will be forwarded from the input data latches directly to output pins, bypassing the RAM core. In this way, the most recent data is always provided by the CDUs.

Special hyperSPARC Features

There are a number of subtle but clever design features implemented in hyperSPARC that improve performance for common CPU functions. One such feature is the RT620 CPU's Fast Constant/Index/Branch capability.

Fast Constant and Fast Index

Fast Constant, for example, represents a commonly occurring combination of two ALU instructions that are used to generate 32-bit constants. Specifically, the SETHI and OR instruction pair is used frequently to create the 22 high-order and 10 low-order bits, respectively. (In fact, the current SPARC compilers generate these two instructions from the pseudo-instruction SET for sufficiently large constants.) When hyperSPARC's scheduler encounters these two instructions used for generating a 32-bit constant, it launches them for execution in parallel, as if they were a single instruction. Thus, an operation that normally takes two cycles (i.e., the setting of the high- and low-order bits for the designated register) is reduced to one cycle. Fast Index works similarly, combining the SETHI and LD instructions used to generate a 32-bit base address for array indexing.

Fast Branch

Fast Branch is a feature that eliminates waiting for the outcome of a condition-code-setting ALU instruction in order to initiate a branch target fetch, so that a branch and associated ALU instruction can be launched simultaneously. This occurs when an integer branch is immediately preceded by an ALUcc instruction type in the instruction packet. In such a case, the scheduler uses a branch prediction strategy to launch both instructions simultaneously (hyperSPARC predicts branch taken). This reduces the number of cycles between branch resolution and either (1) target instruction fetch and execute, if the branch is taken, or (2) continued instruction processing, if the branch is not taken.

Block Copy and Block Fill

Block Copy/Block Fill are special features of the RT625 and RT626 CMTU. These are software-initiated operations (using the STA instructions with special ASIs) to increase the performance of data movement in and out of main memory. Taking full advantage of the RT625 and RT626's read and write buffers, these block manipulation functions allow data to move to and from main memory without having to bring it into cache. This not only saves the latency of filling the cache, but also allows the RT620 to continue processing at the same time.

Block Copy copies an entire 32-byte block of data from a cache or main memory location to another location in main memory. This is particularly useful when copying files, databases, or other large memory blocks to other memory locations. Although the transaction varies depending on whether the data to be moved is in cache and if it is cacheable or non-cacheable, the basic principle is the same. If the data is being copied from one location to another in main memory, for example, it is first read into the read buffer and then transferred to the write buffer, to be written to the specified memory location. While the RT620 must be held during the read of the 32-byte line, it is released to continue processing during the RT625 or RT626's write back to main memory. This represents the realization that to bring this block into cache is superfluous-no operations are being performed on it. Block Copy saves more than 10 clock cycles that would be encountered if the block were read into, and then out of, cache.

Block Fill copies into the specified memory location the doubleword embedded in the special Block Fill STA instruction. The Block Fill works similar to the Block Copy, except that the read transaction is not required because the source data comes from the processor. The specified doubleword pattern is written throughout the 32-byte block of memory and is very useful in initializing large blocks of memory. If the Block Fill feature were not available, this transaction would require the cache line in memory to be written, brought into cache, modified with a series of instructions, and then written back out to main memory.

Manufacturing

From the beginning, hyperSPARC was designed for cost-effective manufacturability. The goal was to make small chips (for cost considerations) behave as if they were one chip (for the performance resulting from high-integration). The chip set was partitioned to do just that. A well-architected design like hyperSPARC can remove interchip delays from critical paths so that the chip set performs as well as a single chip, but without the manufacturing and testing problems of one huge monolithic die.

hyperSPARC was designed with very manageable die sizes, the largest (RT620) being less than 1.5 million transistors (the RT625 has about 800K transistors).

About ROSS Technology . . .

ROSS was incorporated in August 1988 and is an affiliate of Fujitsu, Ltd. Functioning autonomously within the Fujitsu corporate umbrella, ROSS is fully responsible for all operational aspects of its SPARC program. Our objective is to drive SPARC, the industry's dominant RISC architecture, to increased marketshare throughout the 1990's. We will accomplish this by continuing to produce the world's most architecturally advanced SPARC microprocessor products, implemented in world-class CMOS technology, and spanning the full performance and price range of computer applications.