A Single-Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP

B. Ackland, A. Anesko, D. Brinthaupt, S. J. Daubert, A. Kalavade, J. Knobloch, E. Micca, M. Moturi, C. J. Nicol, J. H. O'Neill, J. Othmer, E. Säckinger, *Member, IEEE*, K. J. Singh, *Member, IEEE*, J. Sweet, C. J. Terman, and J. Williams

***Abstract*—An MIMD multiprocessor digital signal- processing (DSP) chip containing four 64-b processing elements (PE's) interconnected by a 128-b pipelined split transaction bus (STBus) is presented. Each PE contains a 32-b RISC core with DSP enhancements and a 64-b single-instruction, multiple-data vector coprocessor with four 16-b MAC/s and a vector reduction unit. PE's are connected to the STBus through reconfigurable dual-ported snooping L1 cache memories that support shared memory multiprocessing using a modified-MESI data coherency protocol. High-bandwidth data transfers between system memory and on-chip caches are managed in a pipelined memory controller that supports multiple outstanding transactions. An embedded RTOS dynamically schedules multiple tasks onto the PE's. Process synchronization is achieved using cached semaphores. The 200-mm^2, 0.25-μm CMOS chip operates at 100 MHz and dissipates 4 W from a 3.3-V supply.**

***Index Terms*—Digital signal processor, multiprocessing systems, multiprocessor interconnection, split-transaction bus.**

I. Introduction

NEXT-GENERATION digital signal processing (DSP) applications like modem banks, cellular base stations, broad-band access modems, and multimedia processors require levels of DSP performance that far exceed the capability of today's programmable processors. Although ASIC solutions provide the most cost-effective implementations, systems implemented in software reduce prototype development times and enable faster market penetration.

Recently, new DSP architectures have been proposed and built to address the needs of these applications [1], [2]. High performance is achieved through some combination of very long instruction word, single-instruction, multiple-data (SIMD), and multiple-instruction, multiple-data (MIMD) parallelism. Invariably, the result represents a tradeoff among performance, code density, scalability and programmability.

This work presents a scalable DSP architecture, called Daytona, that offers a wide range of implementation choices from a purely homogeneous multiprocessor to application specific solutions based on a programmable core enhanced by hardware accelerators. Performance is achieved by a combination of MIMD and SIMD parallelism. Scalability is provided by the simplicity and flexibility of the bus-based architecture. Code density and programmability are achieved by using a combination of RISC plus DSP coprocessor technology together with a powerful software development environment.

Manuscript received July 27, 1999; revised October 19, 1999.

B. Ackland, A. Kalavade, C. J. Nicol, J. H. O'Neill, J. Othmer, E. Säckinger, K. J. Singh, and J. Williams are with DSP & VLSI Systems Research, Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA.

A. Anesko, D. Brinthaupt, S. J. Daubert, J. Knobloch, E. Micca, M. Moturi, and J. Sweet, are with the Microelectronics Group, Lucent Technologies, Holmdel, NJ 07733 USA.

C. J. Terman is with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Publisher Item Identifier S 0018-9200(00)00536-9.

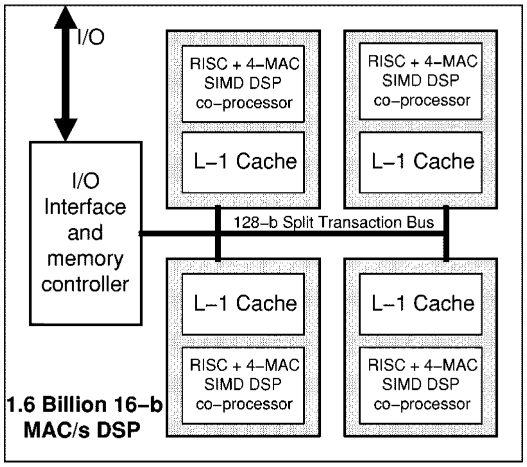

I/O
RISC + 4–MAC SIMD DSP co–processor
L–1 Cache
I/O Interface and memory controller
128–b Split Transaction Bus
1.6 Billion 16–b MAC/s DSP

Fig. 1. The Daytona multiprocessor DSP architecture.

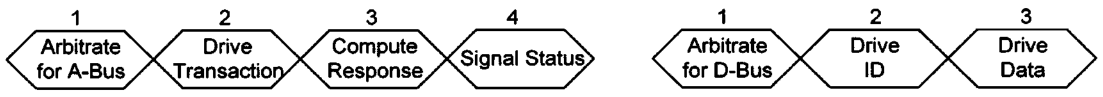
This paper is organized as follows. First, a description of the Daytona multiprocessing platform is described including descriptions of the bus protocol, bus arbitration, I/O, and memory interface. Section III describes the DSP processing element in detail. Section IV describes the software development environment, and Section V describes the four-processing-element (PE) test-chip implementation.

II. Architecture Overview

The chip described in this paper is the first implementation of the Daytona MIMD-DSP architecture. Shown in Fig. 1, it contains four PE's that communicate with each other and with off-chip components over an on-chip 128-b split transaction bus (STBus) operating at 100 MHz. A cache hierarchy is used to provide low-latency memory accesses to the processors and to ensure that the bus does not become a performance bottleneck.

A. The Operation of the Split Transaction Bus (STBus).

In a bus-based multiprocessor, the bus is the principal shared resource. After many simulations with typical applications, a split transaction bus was chosen. The bus is designed to minimize the average latency in a multiprocessor system where several simultaneous requests are made for large amounts of



1 Request Address Bus: High or Low Priority

2 On Grant, drive transaction:

Ba_Id	- Transaction ID (8-b)
Ba_Type	- RD,WR, Idle, CR, CI, CRI, CWI
Ba_Addr	- 32-bit address
Ba_Size	- 1 Byte - 128 Bytes
Ba_Priority	- 4 levels: Instr, Data, Touch, DMA

3 Decode transaction and determine response

4 Respond with (RETRY, ACK, MEMINH, SHARED)

1 Request Data Bus

2 On Grant, drive Transaction ID

3 Drive Bd_Data (16 bytes)

Fig. 2. The STBus protocol.

data. The STBus, with multiple outstanding data transactions, achieves this with a cost of increased complexity in the bus controllers.

The STBus has separately arbitrated address and data buses. A transaction ID is associated with every transaction on the 32-b address bus and then matched to the ID's of the appropriate 128-b data transactions. This enables multiple outstanding transactions to be serviced by the system. Transactions have variable sizes and can be prioritized under program control (e.g., with instruction cache refills having high priority). The bus protocols are deeply pipelined to maximize the peak sustainable bandwidth on the data bus to 1.6 GB/s. The protocol is shown in Fig. 2. It is possible to initiate a new bus transaction every cycle.

On the address bus, the first cycle is used for bus arbitration. Round-robin bus arbitration is performed with programmable levels of priority. When a PE receives a grant, it drives a transaction record containing transaction ID, type, address, size, and priority. The next cycle is reserved to allow PE's on the bus to snoop their data caches and prepare responses. Responses can be ACK to accept, MEMINH to indicate that the transaction will be processed on-chip without need for memory controller intervention, SHARED to indicate that the address is shared between one or more caches, or RETRY'd. Any bus element that is unable to process a transaction can RETRY the transaction. This can occur when a PE's bus interface is busy or its resources are all in use. A RETRY'd transaction may be attempted again at a later time.

The bus supports shared memory multiprocessing with data coherency in the level-1 cache memories using a modified version of the MESI write-invalidate snoopy coherency protocol [3]. Each data cache line has one of five states: invalid, exclusive clean, exclusive modified, shared clean, and shared modified. Accesses to variables in shared memory result in coherent transactions. Various types of coherent transactions are supported: coherent read (CR), coherent invalidate (CI), coherent read with invalidate (CRI) and coherent write with invalidate (CWI). This protocol enables one cache to supply modified data to another cache without the intervention of the main memory controller—a feature not supported by other write-invalidate protocols [4]. Normally, the memory controller (and central direct memory access (DMA) controller) services all requests for data. Coherent address transactions trigger snooping of all the data caches. If a PE has modified data in its cache, it responds to the transaction with MEMINH to prevent the memory controller from servicing the request with off-chip data.

B. The Memory and I/O Controller

The memory and I/O subsystem contains a memory controller, DMA controller, transaction manager, and host I/O interface. The external memory is designed to be a unified multimedia memory subsystem. This is opposed to the approach of using memories dedicated to particular media storage. Care is taken to ensure that the memory subsystem has enough bandwidth to handle all the data streams and is responsive to the real-time constraints of the streams.

The STBus, with its multiple outstanding transactions of variable size, enables the internal bus bandwidth to be matched to the burst modes of off-chip synchronous and multibank DRAM's. For memories with large access latencies, the use of pipelining and fine-grained transfers allow multiple accesses to be overlapped resulting in higher external memory utilization. By tightly coupling the external memory and internal bus controllers, we can achieve high utilization for different external memory configurations.

The transaction manager implements simultaneous point-to-point transfers between external memory, I/O interfaces, and the on-chip STBus. Internal bus bandwidth is not used for external transfers. The DMA controller can perform two-dimensional data transfers for video processing, including operations like subsampling and shuffling. Large transfers are partitioned into smaller packets enabling bus bandwidth to be better utilized.

C. Semaphores

A semaphore block is included that supports up to 64 semaphores. These can be used via system calls for access to shared

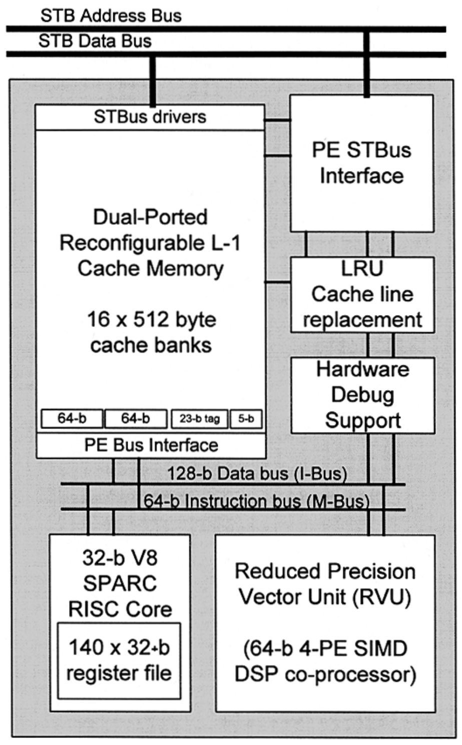

STB Address Bus
STB Data Bus
STBus drivers
PE STBus Interface
Dual-Ported Reconfigurable L-1 Cache Memory
16 x 512 byte cache banks
LRU Cache line replacement
Hardware Debug Support
64-b
64-b
23-b tag
5-b
PE Bus Interface
128-b Data bus (I-Bus)
64-b Instruction bus (M-Bus)
32-b V8 SPARC RISC Core
140 x 32-b register file
Reduced Precision Vector Unit (RVU)
(64-b 4-PE SIMD DSP co-processor)

Fig. 3. The PE architecture.

resources. The STBus coherency protocol is used so that semaphores can be cached in the local PE memories to reduce bus traffic when PE's are waiting for a semaphore to be released.

III. Digital Signal-Processing Element

Each of the processing elements (shown in Fig. 3) features a SPARC RISC processor and an SIMDcoprocessor that operate simultaneously to form a two-slot LIW machine. The instruction slots may be filled with instructions for either processor to minimize code size. Each PE also contains a dual-ported reconfigurable level-1 cache memory, a bus interface and DMA controller, a least recently used (LRU) cache line replacement unit, and a hardware debug support (HDS) processor. This section will describe each of these in detail.

A. The 32-b SPARC RISC Core

The RISC core is a 32-b SPARC V8 core with the five-stage pipelined architecture shown in Fig. 4. An off-the-shelf SPARC C compiler can therefore be used for application development. The RISC core is optimized for loop control, address calculation, and the execution of control code. It features additional DSP-oriented instructions as well as some functions specific for use in the Daytona PE. These include support for two-cycle signed/unsigned 16 × 32 multiply/accumulate, divide step, single-cycle decrement and branch, conditional call, leading one/zero count, coprocessor operations, and cache management. Assembly macros are defined for all new instructions. The *touch* instruction is used to explicitly prefetch data, and the *force* instruction is used to force the write-back of data in the data cache. Special I/O's are provided to assist the HDS processor in formulating a trace of execution history—informing the HDS when the processor is taking branches, etc.

The register file contains 144 registers—eight register windows for nested single-cycle subroutine calls and two sets of eight global registers that speed up interrupt processing. The capacitance of the bit lines is halved by folding the register file into a 72 × 64-b structure. Pairs of bit lines are multiplexed into a single-ended charge redistribution sense amp (from [5]) using an nMOS-only MUX.

The datapath uses a regular three-bus architecture with two operand buses feeding execute units and a result bus. A prefix tree adder [6] is used for add/subtract instructions and can be connected to the MAC to resolve the carry-save MAC outputs for multiply/MAC instructions. A zero look-ahead circuit [7] is used to speed up zero-flag evaluation. The hardware MAC is an array multiplier structure using Booth-add circuits from [8]. Power is reduced by preventing data transitions on the operand buses from entering the MAC when it is not being used. This is achieved without increasing the critical path through the MAC. The critical path for the circuit passes through the X1, X2 inputs to the partial product output (indicated "pp" in Fig. 5). The AND gates on the A inputs (multiplicand) can be inserted without affecting the critical path. The NEG output of the Booth recoder is also gated out of the critical path. When ENABLE is low, the internal node X remains at logic zero independent of the multiplier and multiplicand inputs. Simulations predict a 40% saving in the total power consumption of the SPARC processor using this technique (with a MAC instruction on average every ten instructions). Power can be further saved by tristating the output of the XOR gate (holding its current state rather than forcing it to zero when the MAC is not being used). By implementing the XOR with a MUX, the output of the XOR gate can be tristated by zeroing both the NEG and NEGB control signals. The AND gates on the multiplicand inputs are not needed, but a keeper is required to prevent short circuit currents caused by leakage during long periods of inactivity [9].

B. The 64-b Vector Coprocessor

The vector coprocessor [called the reduced precision vector unit (RVU)] is provided in each PE for speedup of core DSP functions [10]. Fig. 6 shows a block diagram of the RVU coprocessor.

An SIMD-type coprocessor that operates on 64-b split-word data was chosen because it takes advantage of the data parallelism and reduced precision requirements typically found in multimedia and DSP algorithms. Furthermore, an SIMD processor is very area efficient because instruction memory and control logic is shared among the multiple arithmetic elements. The use of split-word operations allows for a flexible, instruction-by-instruction tradeoff between parallelism and computational precision. The RVU supports 8 × 8-b, 4 × 16-b and 2 × 32-b modes. While 8-b precision is sufficient for video and image processing algorithms, 16 or 32 b are required for modem and mobile wireless base-station applications.

The RVU is a tightly coupled coprocessor in the sense that it shares the address generator and instruction fetch mechanism, as well as data and instruction memory with the SPARC. A tightly

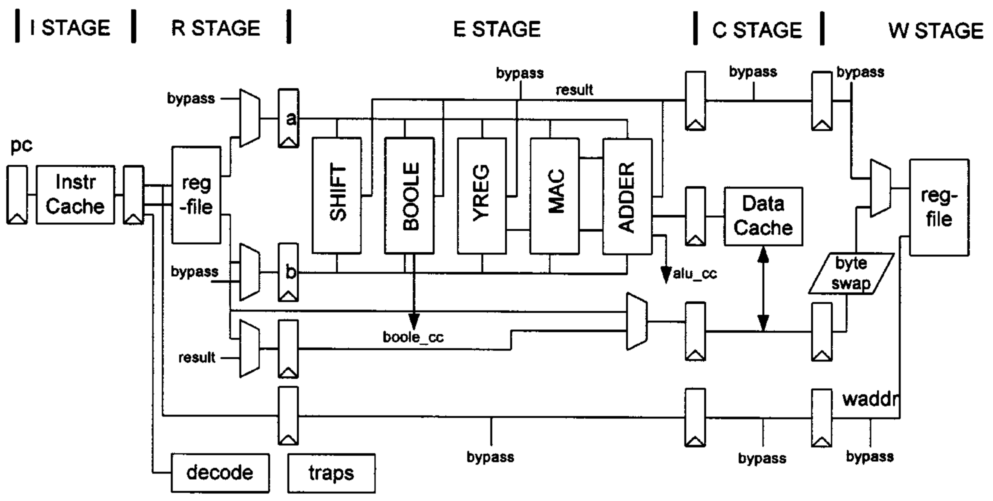


Fig. 4. The Sparc RISC core.

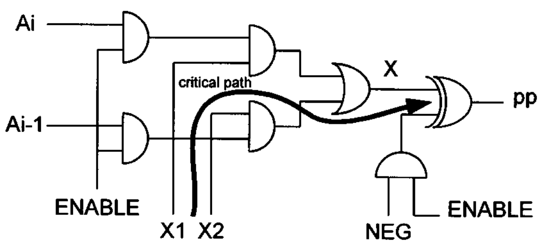


Fig. 5. Low-power partial product circuit.

coupled coprocessor was chosen because of its area efficiency and ease of use. No additional controller and memories are required and the programmer does not need to load instructions and data into coprocessor memories before starting a subroutine.

1) Datapath and Instruction Set: The RVU contains a five-port, 16 × 64-b register file (see Fig. 6). Two 128-b ports permit the loading or storing of two 64-b registers from/to the data memory, while the other three 64-b ports are used for the simultaneous reading of two operands and the write-back of one result. The wide load/store path to memory is necessary to keep the datapath fed and all execution units busy.

The datapath is structured as a pipeline with four logical sections. The first section contains an alignment unit, which can rearrange (shift, swap, replicate, etc.) elements in the split-word data read from the register file. The vector operation unit performs component-wise operations such as four 16-b multiplications or eight 8-b multiplications in a single cycle. Many other componentwise operations such as add, subtract, shift, absolute difference, signum, min-max, select, and Boolean operations are supported as well. Most of these instructions can set vector condition codes controlling a “select” instruction later on, thus avoiding inefficient conditional branching. The next section, the vector reduction unit, provides two ways to accumulate results from the previous section: 1) results are accumulated *within* each vector component or 2) results are accumulated *across* vector components. Fig. 7 illustrates how both types of accumulation can be used to compute four MAC’s in parallel. Component-wise accumulation is useful when multiple dot-products must be computed in parallel, while across-component accumulation is preferable when a single dot-product or related vector operation must be computed. Eight 64-b accumulators are provided besides the register file. The full precision of the computations is kept and overflow is avoided with guard bits throughout the pipeline until the data formatting section is reached. This last section performs scaling, rounding, saturation, data packing/unpacking, etc., on both scalar and vector results without the need for additional formatting instructions. The physical implementation of the RVU compute pipeline consists of four stages, but due to pipeline balancing these stages do not match exactly the logical sections described above.

The RVU instruction set is embedded into the SPARC 32-b instruction space. Unused SPARC op-codes are used to control the RVU. The instructions are encoded in an orthogonal way: each datapath section is controlled by a field in the instruction word, making the RVU flexible and easy to program. Due to the limit in available instruction bits, mode registers are used to fully control the datapath function. Sixteen mode registers are implemented and can be selected quickly through an indexing mechanism.

The peak performance of the RVU is 2400 Megaoperations per second (MOPS), which is achieved when computing the 8-b Manhattan distance (e.g., used in video motion search). Twenty-four arithmetical operations, not counting scaling, rounding, etc., are performed every cycle. For algorithms based on 16-b MAC operations, the peak performance is 800 MOPS. Table I gives examples of the actual performance achieved.

2) Integration with SPARC Processor: The SPARC processor and RVU vector coprocessor together form a two-slot LIW machine. Two instructions (64-b) are fetched in one cycle and decoded simultaneously by the SPARC and RVU. If one instruction is a SPARC instruction and the other an RVU instruction, they are issued simultaneously in a single cycle. If the two instructions are of the same type (two SPARC or two RVU instructions), they are issued sequentially on the respective processor. It is the programmer's goal to alternate SPARC and RVU instruction to get the highest possible performance. A typical instruction pairing is 1) a load instruction to the RVU register file—which counts as a SPARC instruction and 2) an SIMD MAC instruction executed on the RVU. Similarly,

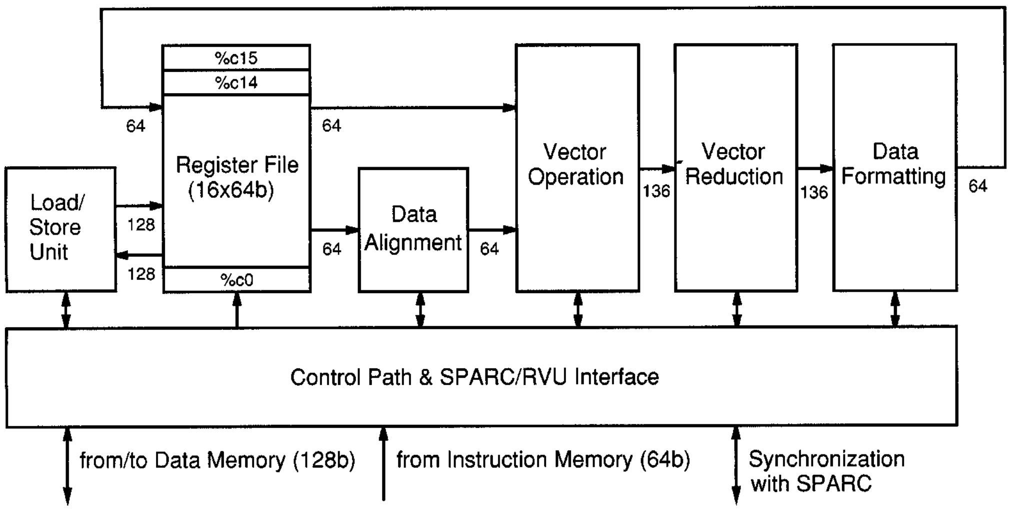


Fig. 6. The architecture of the RVU vector coprocessor.

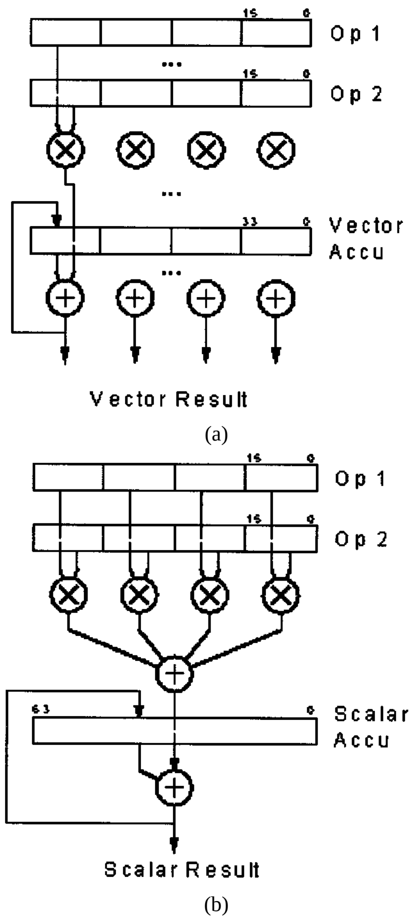


Fig. 7. Two types of parallel MAC: (1) Componentwise accumulation versus (2) across-component accumulation.

SPARC pointer operations and flow-control operations can be paired with SIMD-type operations.

The SPARC and RVU pipelines are shown schematically in Fig. 8; they always operate in lockstep. An RVU instruction is fetched in the I stage and delayed by three cycles (stages D1–D3) before the RVU register file is read (RD stage). Subsequently, the instruction enters the four execution stages (E1–E4) performing the datapath functions described earlier; then the result is written back to the RVU register file (WR stage). The delay stages together with the load bypass make it possible to issue a load instruction and a vector operation instruction using this loaded data in the same cycle. This feature simplifies the code writing process significantly. Full pipeline bypassing is provided (not shown in Fig. 8) to avoid pipeline hazards and simplify application development.

TABLE I
DSP PERFORMANCE OF THE DAYTONA PE ON APPLICATION KERNELS

Application	Perf. (MOPS)	RVU gain	Code size (Kbytes)
10th Order FIR	510	9.5	0.3
GSM LPC Filter	350	7.3	0.3
512-pt complex FFT	230	7	1.6
Sub-sampl motion srch	1670	27	0.47
Full-pixel motion srch	1280	20	0.41
Half-pixel motion srch	780	17	0.43
MPEG-2 DCT / iDCT	490	7	0.3
MPEG-2 quantization	300	13	0.24

C. The DSP Performance of the PE

The DSP performance of the PE (assuming no cache misses) for different core routines is shown in Table I. The routines were coded for the SPARC with and without the coprocessor. When operating without the RVU, the SPARC is executing a single instruction every cycle at 100 MHz (with two cycles needed for a multiplication). The ratio of the execution times is reported as the RVU gain—shown to be as high as 27 for motion estimation. To date, we have successfully mapped three complete asymmetrical digital subscriber line (ADSL) discrete multitone (DMT)-lite modem applications onto a single PE—and correctly executed 12 ADSL DMT-lite modems on the four-PE test chip when operating at 100 MHz. This demonstrates the high DSP performance available for central-office DSL applications.

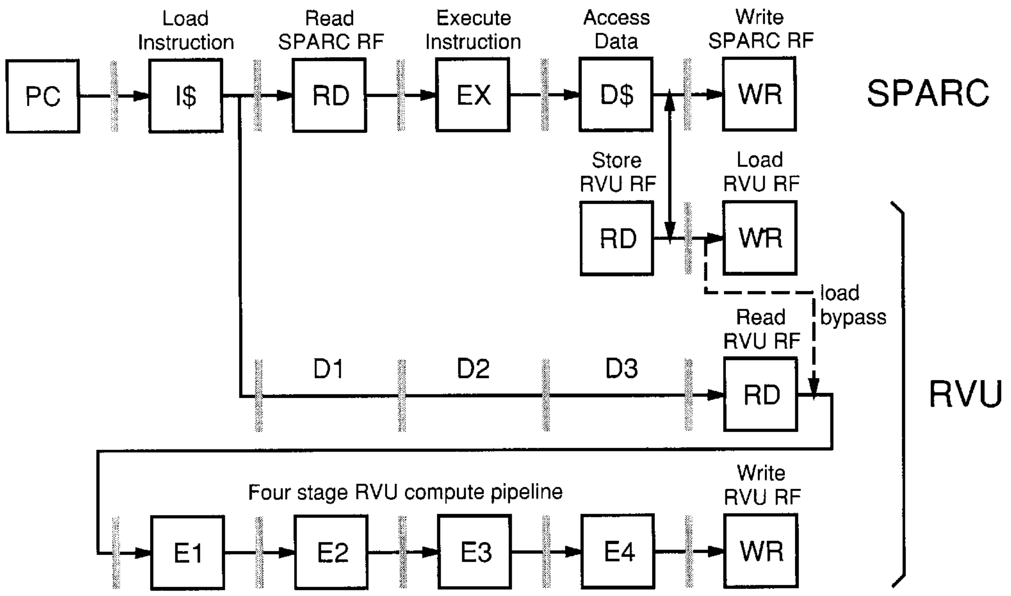


Fig. 8. SPARC/RVU pipeline architecture (only one bypass is shown).

D. The PE STBus Controller

The PE communication controller performs the following operations.

1) Generate the appropriate transfers on the STBus to accomplish the processor's request for data transfers
2) Snoop the STBus to identify memory-mapped operations for the PE and maintain data coherency
3) Perform the DMA transfers that are requested by the processor.

The architecture of the local memory greatly affects the area and performance of the PE. As an example, consider a local memory that provides a single read/write port. Such a memory has a smaller footprint than a dual ported memory. However, there is a large performance penalty since accesses from the processor side and from the STBus will need to be serialized to resolve access contention for the single port. In order to provide maximal concurrency of operations, we should be able to perform the following accesses:

- cache reads for the Instruction Bus (cached or direct addressing);
- load/store accesses from the Data Bus of the processor (cached and direct addressing);
- cache state and tag queries and stores to determine the operations required to handle nonblocking operations (touch, stores);
- snoop the state of data cache lines to determine appropriate responses to coherent transactions;
- update the state of data cache lines based on the coherent transactions that are accepted on the STBus;
- read/write the state, tag, and data RAM's to source/sink data to/from the bus.

The operations suggested above may be simultaneous or separated in time. The correctness of the memory operations needs to be enforced by the appropriate control protocol based on the capabilities of the memory.

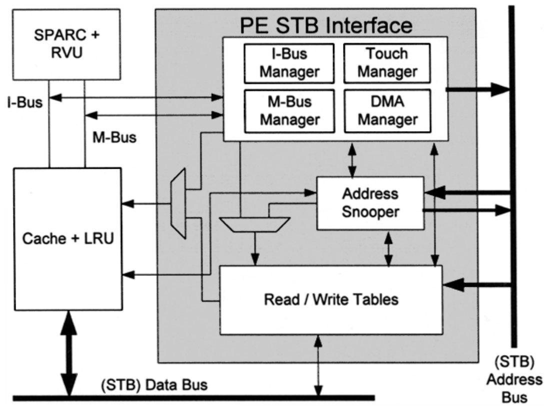


Fig. 9. The PE STBus Interface unit.

The PE bus controller manages the local cache memory to maximize the performance of the system by performing many of the above operations transparently while ensuring correct operation of the STBus. The architecture of the PE STBus Interface is shown in Fig. 9. It contains a transaction controller that manages the PE buses as well as separate units for the management of nonblocking *touch* and DMA operations. The transaction controller can store up to four outstanding transactions. The bus interface unit can also manage up to eight outstanding data operations. If these resources are full, coherent transaction on the bus will be RETRY'd. An address snooper executes in unison with the transaction controller. The snooper must also check for outstanding read/write transactions that might be affected by coherent transactions on the STBus. The read/write tables therefore support associative search mechanisms to support snooping. One of the most complex functions is the implementation of the cache coherency protocol. To demonstrate this, consider the actions taken when the processor writes to a shared variable in data cache—the state diagram is shown in Fig. 10. This is further complicated by the fact that the STBus protocol

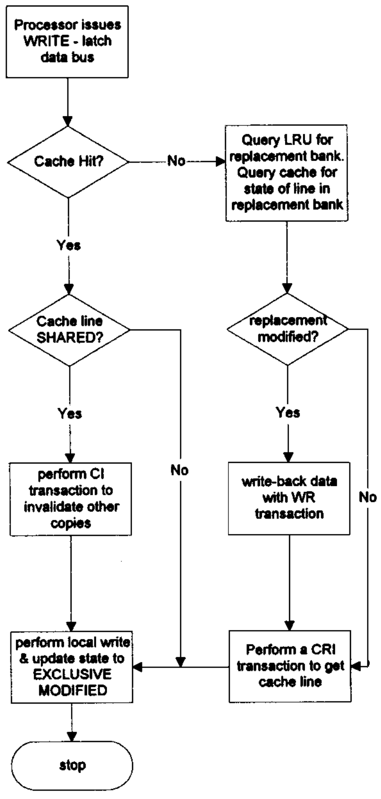

Processor issues WRITE - latch data bus
Cache Hit?
No
Query LRU for replacement bank. Query cache for state of line in replacement bank
Yes
Cache line SHARED?
replacement modified?
Yes
Yes
No
No
perform CI transaction to invalidate other copies
write-back data with WR transaction
perform local write & update state to EXCLUSIVE MODIFIED
Perform a CRI transaction to get cache line
stop

Fig. 10. The state diagram for a processor WRITE to data cache.

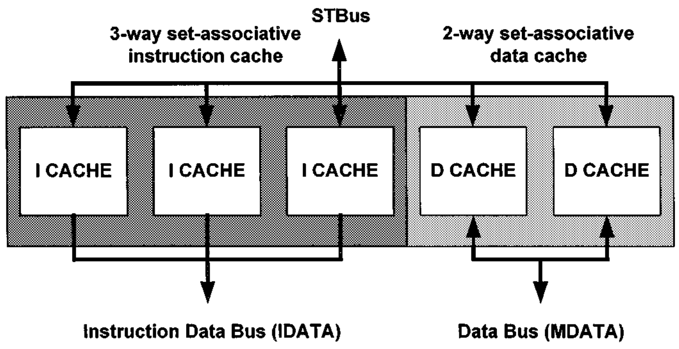

STBus
3-way set-associative instruction cache
2-way set-associative data cache
I CACHE
I CACHE
I CACHE
D CACHE
D CACHE
Instruction Data Bus (IDATA)
Data Bus (MDATA)

Fig. 11. An example of a reconfigurable set-associative cache.

is pipelined so that many coherent transactions may need to be processed simultaneously.

One important consideration in the design of the memory architecture is that there is no RETRY of transactions on the STB data bus. This means that the local memory must be able to sink data off the bus. The write to the cache from the STBus is therefore the highest priority access to the cache. Also, the nonblocking data accesses (DMA and *touch* of cache blocks) are made lower priority than the blocking accesses (load of instruction and data caches). The PE STBus Interface prioritizes all cache accesses and handles all cache access collisions—accesses to the same cache line. When the processor writes to data cache, the processor is not stalled on a cache miss—the data are stored in a write buffer (as shown in the state diagram in Fig. 10) and the store is completed later. Once the write buffer is occupied, subsequent memory accesses stall the processor until the outstanding write is completed. Hence, it is important to handle this transaction with high priority.

E. The Reconfigurable Level-1 Cache Memory

In a bus-based multiprocessor system, the bus bandwidth is a precious resource. If it is not balanced correctly, the processors may be starved of data, and the desired MIMD speedup is not achieved. Whereas it is common practice to include a large amount of single-cycle SRAM on a DSP chip for instructions, we chose to have a limited amount of dynamically reconfigurable level-1 cache memory with each processor. This has the advantages of allowing more processors to be integrated onto a single chip and optimizing the use of the local memory depending on the application.

Each PE has 8 KB of reconfigurable memory that is divided into 16 banks of 512 bytes. Every bank can be configured to be instruction cache, data cache, or local buffer for explicit storage of instructions or data. When configured as local buffers, the memory is accessed by the processor via a dedicated part of the address map. Each cache bank has its own data RAM, tag RAM, state RAM, and comparators for supporting set-associative caching. It also performs cache snooping for shared-memory coherency. Using operating system calls, an application can change the configuration of the memory banks as it needed change. For example, in a typical modem application, 90% of the executed instructions are inner-loop DSP code for the transmit and receive filters. But this code only occupies about 10% of the instruction image—with the control code occupying the bulk of the memory. It follows that when the DSP is executing the inner-loop code, only a small amount of I-cache is needed (see Table I) to achieve a high cache hit rate. When the control code is executed, however, a much larger I-cache is required. Furthermore, the inner-loop code tends to process vast amounts of data, and so the memory banks not needed for I-cache can be utilized for data storage. Profiling is used by the application developer to determine the correct amount of I- and D-cache needed to achieve the required performance. The cache banks are then configured to provide the cache configuration.

When two or more banks are configured as instruction cache, they act together to form a set-associative cache that further improves the utilization of the memory. The same is true for data cache. This is demonstrated in Fig. 11, where three banks are configured as I-and two banks are configured as D-cache. The data cache banks participate in the snooping of the address transactions on the STBus for data coherency. The LRU replacement policy is used for line replacement in caches of either flavor (described in detail in the next section). Instruction locking is supported on a per-bank basis by modifying the cache bank configuration supplied to the LRU unit. The LRU unit will never target a locked bank for replacement. This can be used to guarantee that critical portions of code remain resident in cache.

The memories are 128-b wide, and the line size is 32 bytes. A cache line is filled in two STBus data cycles that may be separated in time by data cycles of other transactions. Critical-word forwarding is supported to minimize miss latency. The processor fetches two 32-b instructions over a 64-b instruction bus and up to four 32-b data words over a 128-b data bus. Fig. 12 shows the circuits needed to make a cache

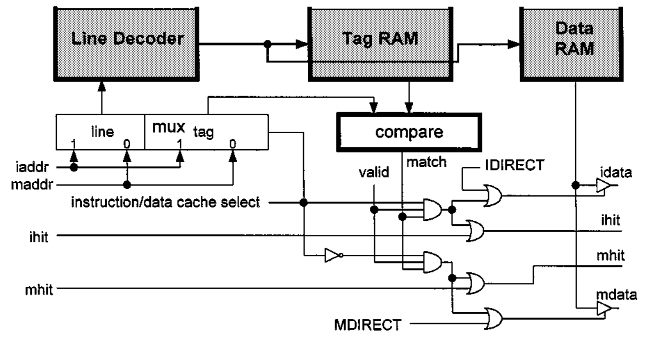


Fig. 12. The reconfigurable cache bank circuit.

bank reconfigurable and set-associative. The PE instruction address (IADDR) and data address (MADDR) are fed into a MUX that is controlled by the I/D cache configuration bit for that bank. The line address is decoded and used in the tag RAM to read out the tag from cache. The tag part of the address is sent to the comparator. In the case of a cache hit, the output of the data RAM is driven to IDATA (for instruction cache) or MDATA (for data cache). The appropriate hit signal is cascaded through all banks (they are actually precharged and sensed in the implementation). Two signals IDIRECT and MDIRECT override the cache query mechanism and enable the processor to directly address the cache memory as instruction buffer or data buffer.

To support simultaneous accesses from the PE instruction bus, the PE data bus or the system STBus, the cache banks must have access to all three buses. Simply routing these busses across the cache would significantly increase the bus capacitance. A hierarchical bus structure with a hierarchy of sense amps avoids this problem. Each bank has local bit lines and sense amps. During a read operation, the contents of the memory are sensed locally and transferred to one of the global busses. Global sense amps are located at the top (for STB) and bottom (for PE) of the cache. The memories are not truly dual-ported. Each bank has a single read port and a single write port. Simultaneous read accesses are supported from different banks but not from the same bank. Simulations showed that from an external viewpoint, dual-ported access patterns were possible for most cases—the only problem occurred when DMA reads or writes coincided with PE accesses to the same bank of the same type (both reading or writing). Configuring DMA transactions such that two banks are used in a double-buffered manner avoids this limitation. In the event of a collision within a single bank, the local STBus controller detects these and stalls the processor access accordingly without user intervention.

The tag RAM's in each cache bank contain two dynamic comparators. One for instruction or data accesses by the PE—the other for snooping. STBus snooping occurs in the same cycle as any PE access. The cache query mechanism is entirely self-timed and triggered by a clock named TAGCLK—see the timing diagram in Fig. 13.

The scheme proposed in [11] is supported to reduce power in the set-associative caches when operating at low frequency (less than 70 MHz). Normally, all cache banks will access their data RAM's at the same time as the tag RAM accesses are taking place. However, only one bank will drive data onto the output

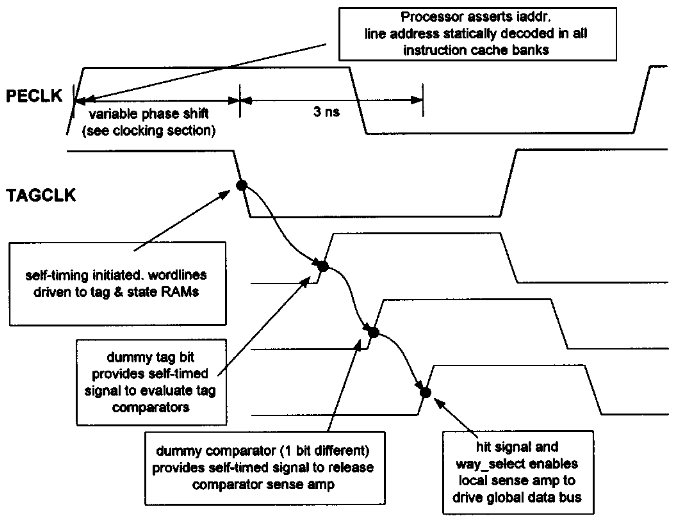

Processor asserts iaddr. line address statically decoded in all instruction cache banks
PECLK
variable phase shift (see clocking section)
3 ns
TAGCLK
self-timing initiated. wordlines driven to tag & state RAMs
dummy tag bit provides self-timed signal to evaluate tag comparators
dummy comparator (1 bit different) provides self-timed signal to release comparator sense amp
hit signal and way_select enables local sense amp to drive global data bus

Fig. 13. Self-timing for a cache access.

bus. If time permits, the way select from the tag RAM is used to initiate the data RAM access so that data are read out from the data RAM of only the bank that produced a hit. No other bank will access its data RAM. This significantly reduces power in the caches at low frequency.

One of the features of the cache memory is that the PE can override the bank configuration on any given access. For example, the PE can DMA instructions into a data buffer and then execute them. For this to occur, a cache bank configured as a data buffer, which normally decodes MADDR, must be able to decode IADDR and source data to IDATA. Similarly, instruction buffers can be accessed by the read/write data operations on MADDR—useful for cache priming and self-modifying code. This type of "direct access" could potentially impact the critical path of the cache because of the logic needed to determine the intention of the processor and select either IADDR or MADDR for decoding. However, a direct access does not require the tag comparison process at all. It can be initiated later than a regular cache access and still meet the cycle time. The IADDR/MADDR address MUX has a preconditioned operation that enables a regular cache access to proceed as quickly as possible (by always decoding the address determined by the bank configuration bit)—but it can be cancelled when an access override is detected. In such a case, the IADDR/MADDR address MUX is switched over and a new line address is decoded. When this occurs, the PECLK is used to trigger the self-timing of the RAM's instead of the earlier TAGCLK to allow the new address to be fully decoded. This simple technique reduces the timing overhead of the reconfigurable cache to the propagation delay of a single MUX.

F. The LRU Cache Line Replacement Unit

We now describe the LRU circuit that is used in conjunction with the reconfigurable set-associative cache memory. The LRU instruction and data cache banks for each line are calculated simultaneously in accordance with the cache bank programming. The module contains a multiported nibble-writeable 16 × 72-b register file and a pipelined LRU computation unit complete with forwarding to reduce memory power consumption.

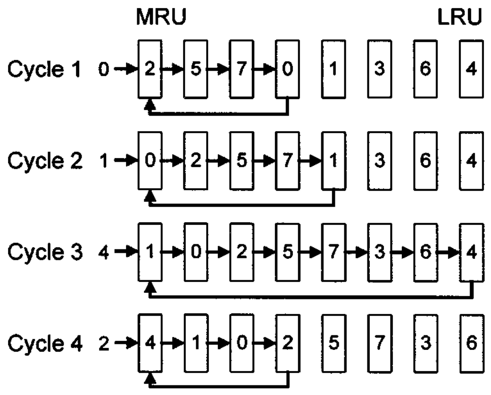

MRU
LRU
Cycle 1 0 2 5 7 0 1 3 6 4
Cycle 2 1 0 2 5 7 1 3 6 4
Cycle 3 4 1 0 2 5 7 3 6 4
Cycle 4 2 4 1 0 2 5 7 3 6

Fig. 14. A description of the LRU algorithm.

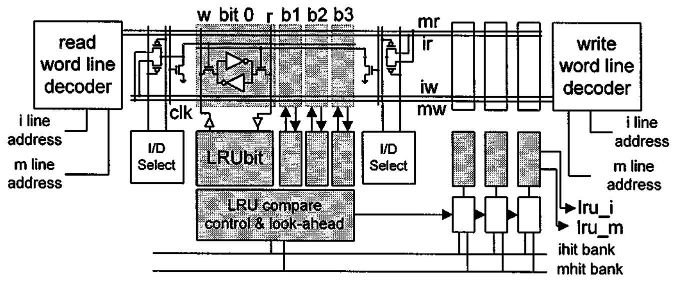


Fig. 15. The architecture of the LRU RAM.

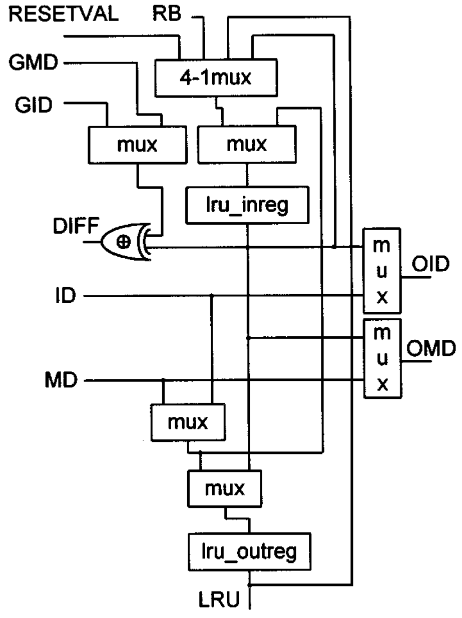


Fig. 16. The LRU logic.

Set-associative cache line replacement is usually implemented in some pseudo-LRU form to minimize the amount of state information that must be stored [3]. While this approach is trivial for two-way and practical for four-way set-associative caches, it is less feasible for memories with higher levels of associativity. Furthermore, in our architecture, the LRU algorithm is required to support a dynamically changing cache with variable set-associativity.

An example of how the LRU stack is maintained is shown in Fig. 14. In cycle 1, a "hit" on bank 0 triggers a search in the stack for a match. The other banks are moved down one position and the "0" is moved to the top of stack. In cycle 3, a hit on bank 4 causes a new LRU (bank 6) to be output. Although this computation appears trivial, it essentially involves an associative search-and-shift operation to locate and promote a bank to the top of the stack. It would be too time consuming to read the stack from the RAM, locate and shift the bank ID within the stack, and write it back to the RAM in a single cycle. For a large number of banks (16 in our case), the LRU computation requires a 16-stage daisy-chain of shift logic. We first describe the LRU RAM configuration and then the logic used to compute the LRU in a single cycle.

1) The LRU RAM: In any cycle, the LRU RAM is capable of producing the LRU states for cache lines in both the instruction and data caches according to the current cache access. It is also capable of updating the LRU state for different cache lines from a previous cache access.

This is achieved by decoding two READ line addresses and two WRITE line addresses. In each case, one address is for the instruction access and the other is for the data access, and both are extracted from the processor address buses. In each READ cycle, the RAM produces a single 72-b word containing 18 4-b bank ID's. Sixteen of these are used to compute the LRU-I and MRU-D values for the I- and D-cache lines. The LRU-I and LRU-M values are stored back into the RAM when the state is written back. This enables the LRU to be output in the same cycle as when a cache miss is detected without first needing to compute the LRU. This is important in a write-back data cache—the cache controller requires the replacement bank immediately to minimize miss-latency.

The architecture of the LRU RAM is shown in Fig. 15. Groups of 4 bits are connected to a read word line and a write word line. The word lines are connected to global I or D read/write word lines depending on the programming of the cache banks. Separate read/write row decoders are needed for I and D. These are clocked to prevent short circuit current during the precharge cycle of the bit lines. The write decoders can be forced high by I and D reset signals. This forces every column in every I or D row to be written at once. The RAM cells have separate read and write ports connected to separate read and write bit lines. The devices are sized to make the cell directional (similar to a register file). A single-ended charge-redistribution sense amplifier is used for the read circuit.

2) The LRU Logic: The LRU read logic is shown in Fig. 16. It reads a state history word from the RAM and updates the I-cache and D-cache components of the state according to the bank configuration bits and the status of the cache accesses. The LRU RAM is accessed in parallel with the cache access. The second cycle is used to compute the LRU for both I-cache and D-cache. The state is written back to the RAM (together with the LRU-I and LRU-D bank ID's) in the third cycle.

Pipeline hazard detection and forwarding are used to guarantee correct operation when the same cache line is accessed in consecutive cycles. Further optimizations are performed to minimize the accesses to the memory.

- If the LRU state for a cache line is not modified, it is not written back to the RAM.

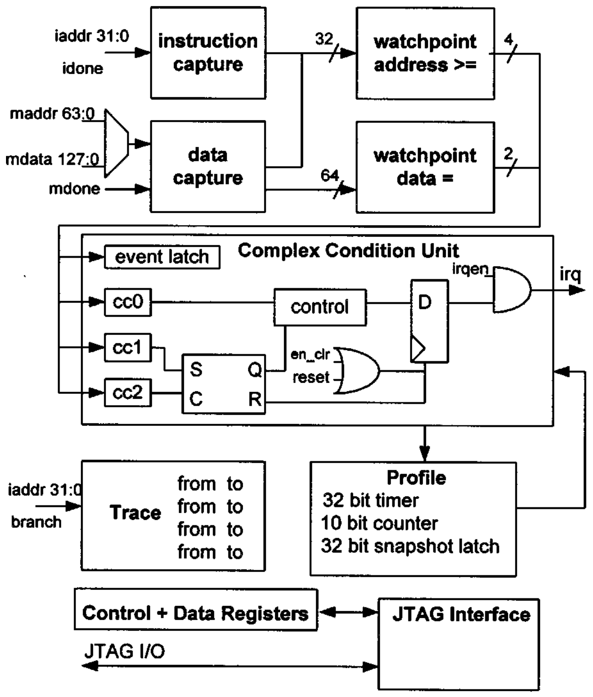


Fig. 17. The hardware debug support processor.

- If the LRU state for a cache line is already in the LRU pipeline, it is forwarded to the head of the pipe and the RAM is not read.

When the cache is reset, the LRU RAM must be initialized with the bank ID's in the appropriate sequence. In addition, the LRU states for the I- and D-caches can be separately reset.

A significant amount of look-ahead logic is used to enable the 16-stage daisy-chain LRU logic to evaluate in a single cycle. Look-ahead logic operates in groups of four.

G. The Hardware Debugging Support Processor

The PE's also contain a hardware debugging system (HDS) (shown in Fig. 17) to support the development of embedded applications. This system includes counters and comparators to support complex watch-point conditions and provide a history of recent execution. The HDS systems in the PE's are daisy-chained and externally controlled by the host via a JTAG port.

IV. Software Development Environment

An architecture of this complexity requires a comprehensive application development platform. This is described in detail in [12]. It includes systems for static and dynamic task scheduling, debugging, profiling, and compiling. An advantage of using a standard RISC core in the PE is that the bulk of the application code can be written in a high-level language (like C) and compiled using an off-the-shelf compiler. Core DSP routines that use the vector coprocessor are also written in C. A prototype vectorizing C compiler for the RVU was developed. Maximum RVU utilization can be achieved using hand-crafted assembly language.

An embedded Real-Time Operating System (RTOS) is employed to dynamically schedule tasks on PE's. The RTOS supports multiprocessor operation through a two-level scheduling paradigm. At the top level, a centralized control scheduler resides on a control processor, say, PE0. This control scheduler admits new tasks into the system and assigns them to an appropriate processor. The admission control ensures that a task is admitted only if there is sufficient processing power to support the new task and the task is assigned to a processor that can sustain its load. At the second level, each processor runs its own local scheduler and maintains its own task list. The local scheduler selects tasks to run according to the earliest-deadline-first (EDF) scheduling policy. The EDF scheduling policy guarantees that all admitted tasks will meet their deadlines.

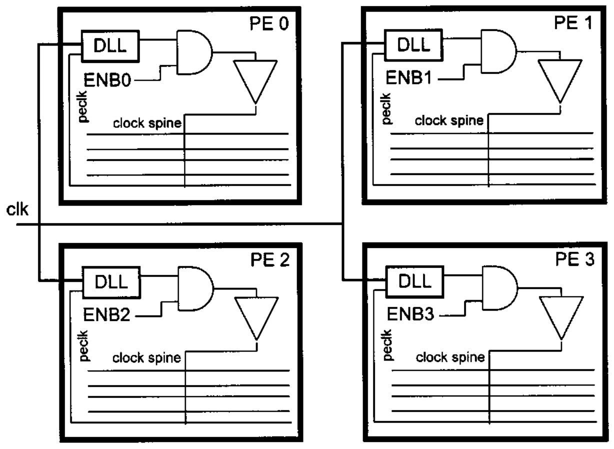


Fig. 18. The Daytona clock network.

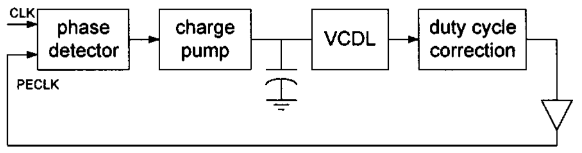


Fig. 19. The delay-locked loop.

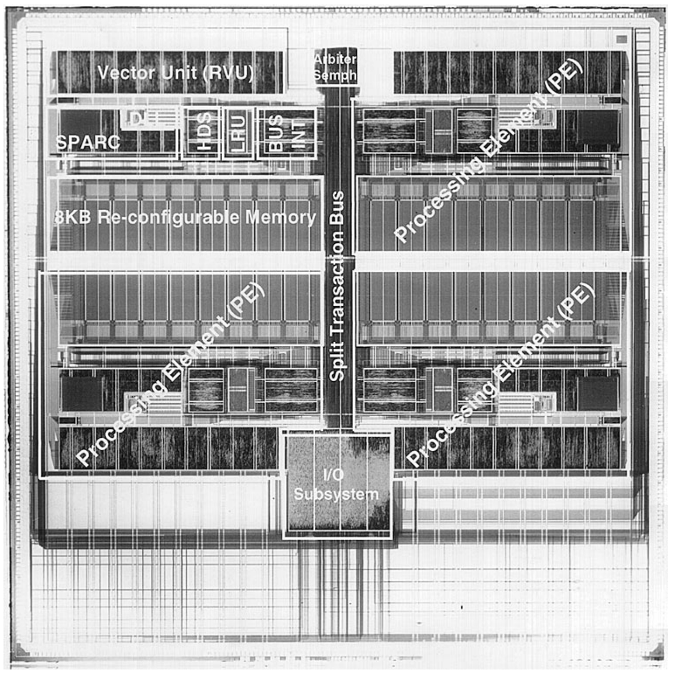


Fig. 20. A photomicrograph of the Daytona test chip.

Considerable effort went into minimizing the cycles needed for context switches. To this end, the local cache memory bank configuration can be managed by the RTOS. To manage power consumption, the RTOS has the ability to shut down any of the PE’s using an addressable register that controls the clocks supplied to each PE.

V. Chip Implementation

A. Daytona Chip Design Methodology

The chip was designed using a mix of full-custom and standard-cell design methodologies. The cache memories, LRU memory and RISC register files are designed in full custom. The remainder of the chip was synthesized from a VHDL description. A C-model of the chip was used for presilicon software application development, profiling, and architectural investigation.

B. Clock Strategy

In the four-PE Daytona chip, each PE can be independently powered up or down. It is important that the chip include a method of reducing clock uncertainty. In this design, the approach taken to reduce clock skew includes a DLL. The chip clock distribution architecture including the DLL’s in each PE is shown in Fig. 18.

Including a DLL in each PE aligns each internal PECLK to the global CLK signal even though the clock is gated and buffered within the PE. Since not every application of Daytona may require all PE's to be operating, the gating of the clock can reduce the power dissipation by shutting off unused PE's. As described in the previous section, this feature is controlled by the RTOS.

A DLL was used instead of a phase-locked loop since frequency synthesis is not needed, and a DLL has lower jitter accumulation and a faster locking time. The DLL, shown in Fig. 19, consists of a phase detector, charge pump, loop filter capacitor, voltage-controlled delay line (VCDL), and duty-cycle correction circuit. The phase detector compares the PE clock, after it has been internally buffer to the phase of the main chip clock (CLK). The phase detector operates by setting a latch on the rising edge of CLK and resetting the latch on the falling edge of PECLK. If PECLK has a 50% duty cycle, then the rising edge of CLK and PECLK will be aligned by the charge pump, which will spend an equal amount of charging up the loop capacitor and discharging it. This maintains a steady-state loop capacitor voltage that corresponds to a clock cycle delay through the VCDL. A duty cycle correction circuit is included in the DLL following the VCDL to correct any duty cycle variations that are either present in the input signal, CLK, or may occur in the VCDL. The DLL is also used to generate additional clocks needed in the cache. The TAGCLK used to trigger the self-timed cache circuits can be shifted relative to the PECLK with programmable increments of phase shift. The phase shift is preserved independent of the clock frequency.

C. Test-Chip Specifications

The chip was fabricated in 3.3-V, 0.25-μm CMOS process with four layers of metal. A single PE occupies 27 mm^2 (4.5 × 6 mm). Simulation results demonstrate performance in excess of 100 MHz at 3.3 V. The core chip area is 120 mm^2; however, the pad-limited test chip has a total area of 200 mm^2. The power estimate is 4 W with all four PE's operating. A chip photomicrograph is shown in Fig. 20.

Acknowledgment

The authors thank the contributions made to the Daytona project by the following people: J. Bauman, H. D. Lin, T. Denk, B. Denney, F. Derepas, A. Goncalves, M Guenais, A. Kulkarni, J. Leonard, U. Mueller, W. Peslak, E. Scuteri, P. A. Subrahmanyam, and M. Zalonis. They also thank M. Bickerstaff for assistance with proofreading.

References

[1] J. Turley and H. Hakkaraainen, “TI's New ‘C6X Screams at 1 600 MIPS’,” Microprocessor Rep., Feb. 1997.

[2] H. Igura *et al.*, “An 800MIPS 110 mW 1.5 V parallel DSP for mobile multimedia processing,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC98)*, Feb. 1998, pp. 292–293.

[3] J. Handy, *The Cache Memory Book*. New York: Academic, 1993.

[4] M. Tomasevic and V. Milutinovic, “Hardware approaches to cache coherence in shared memory multiprocessors: Part 1,” *IEEE Micro*, vol. 14, no. 5, pp. 52–59, Oct. 1994.

[5] A. Chandrakasan and R. W. Brodersen, “Minimizing power consumption in digital CMOS circuits,” *Proc. IEEE*, vol. 83, no. 4, Apr. 1995.

[6] R. Kolagotla, C. J. Nicol, A. Godolvsky, and M. Besz, “A 1.0-nsec 32-bit prefix tree adder in 0.25-um. static CMOS,” in *Proc. 42nd Midwest Symp. Circuits and Systems*, Aug. 1999.

[7] K. Suzuki, M. Yakashina, J. Goto, T. Inoue, Y. Koseki, T. Horiuchi, N. Hamatake, K. Kumagai, T. Enomoto, and Y. Yamada, “A 2.4 ns 17-Bit, 0.5 um CMOS arithmetic logic unit for microprocessor video signal processor LSIs,” in *Proc. IEEE Custom Integrated Circuits Conf. (CICC93)*, CA, May 1993, p. 12.4.1.

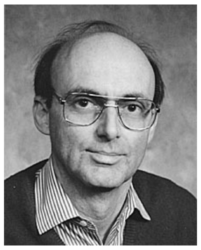
[8] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: Second Edition—Chapter 8*. Reading, MA: Addison-Wesley, 1992.

[9] C. J. Nicol, “Low power multiplier for CPU and DSP,” U.S. Patent filed, Mar. 13, 1998.

[10] E. Säckinger, U. A. Müller, and P. A. Subrahmanyam, “A parallel processor chip for image processing and neural networks,” in *ICANN’95*, Paris, France, Oct. 1995.

[11] Y. Shimazaki, K. Ishibashi, K. Norisue, S. Narita, K. Uchiyama, T. Nakazawa, I. Kudoh, R. Izawa, S. Yoshioka, S. Tamaki, S. Nagata, I. Kawasaki, and K. Kuroda, “An automatic-power-save cache memory for low power RISC processors,” in *Proc. IEEE Symp. Low Power Electronics. (SLPE95)*, San Jose, CA, Oct. 1995, pp. 58–59.

[12] A. Kalavade, J. Othmer, B. Ackland, and K. J. Singh, “Software environment for a multiprocessor DSP,” in *Proc. Design Automation Conf.*, New Orleans, LA, June 21–25, 1999, pp. 827–830.



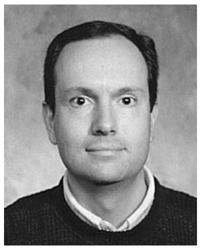
B. Ackland received the B.Sc. degree in physics from Flinders University, Australia, in 1972 and the B.E. and Ph.D. degrees in electrical engineering from the University of Adelaide, Australia, in 1975 and 1979, respectively.

In 1978, he joined Bell Laboratories as a Member of Technical Staff in the Image Processing and Display Research Department. In 1986, he was appointed Director of the VLSI Systems Research Department in Holmdel, NJ. During this time, his research has spanned a number of areas including raster graphics, symbolic layout, and verification tools for full-custom VLSI, MOS timing simulation, VLSI layout synthesis, and DSP architecture. His current interests are focused on VLSI architectures and circuits for high-performance signal-processing applications, particularly in the areas of communication and multimedia.

Dr. Ackland is a Bell Laboratories Fellow.

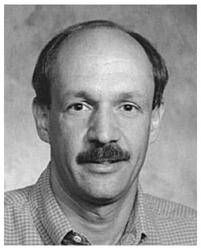
A. Anesko received the B.S.E.E. degree from Rutgers—The State University, Piscataway, NJ, in 1991 and the M.S.E.E. degree from Columbia University, New York, in 1992.

He joined the Intel DVI Operation as a Staff Engineer in 1992 and joined AT&T Bell Labs as a Member of Technical Staff in 1993. He is currently a Member of Technical Staff within the Network Communications Business Unit of Lucent Technologies Microelectronics Organization.



D. Brinthaupt received the B.S. degree in electrical engineering from Cornell University, Ithaca, NY, in 1985 and the M.S.E. degree in computer engineering from the University of Michigan, Ann Arbor, in 1986.

He is a Distinguished Member of Technical Staff at Lucent Technologies' Microelectronics Group in Holmdel, NJ. In his 14 years with Lucent (and AT&T Bell Labs), he has contributed to the design of several digital integrated circuit projects for telecommunications applications including voice, ISDN, videoconferencing, DSL, and broad-band ATM access.



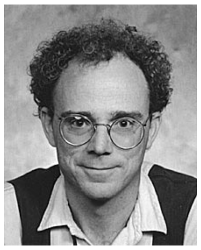
S. J. Daubert was born in Lebanon, PA. He received the B.S. degree from Pennsylvania State University, University Park, in 1980, the M.Eng. degree from Cornell University, Ithaca, NY, in 1981, and the Ph.D. degree from Columbia University, New York, in 1991, all in electrical engineering.

He joined Bell Laboratories in 1980 as a Member of Technical Staff and has designed analog and digital circuits for modem, ISDN, DSL, and video applications. He is presently involved in chip design for broad-band access at Lucent Technologies' Microelectronics Group.

A. Kalavade received the Ph.D. degree in electrical engineering from the University of California at Berkeley in 1995.

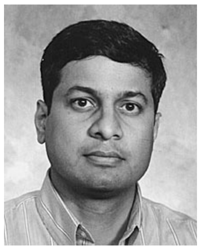
She is currently a Member of Technical Staff in the Networked Multimedia Research department at Bell Labs in Murray Hill, NJ. Her currrent research interests are in the area of wired and wireless networked multimedia systems, software environments for DSP systems, and reconfigurable computing.

J. Knobloch, photograph and biography not available at the time of publication.



E. Micca received the B.S. degree in electrical engineering technology from Fairleigh Dickinson University, Teaneck, NJ, in 1986 and the M.S. degree in computer science from Stevens Institute of Technology, Hoboken, NJ, in 1990.

He is a Member of Technical Staff at Lucent Technologies' Microelectronics Group in Holmdel, NJ. He has been involved in the design and test of digital integrated circuits for telecommunications applications during his 17 years with Lucent.

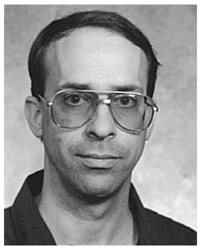


M. Moturi received the M.S. degree in electrical engineering from Syracuse University, Syracuse, NY, in 1992.

He joined Bell Laboratories in 1992 as a Member of Technical Staff. He is currently a Technical Manager in the Microelectronics Group of Lucent Technologies.

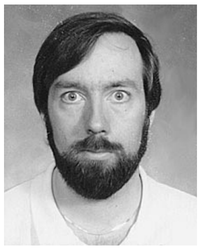
C. J. Nicol received the B.Sc. and Ph.D. degrees from the University of New South Wales, Australia, in 1991 and 1995, respectively.

His thesis was on the design of VLSI chips for real-time image processing. He spent one year with Bell Laboratories, Holmdel, NJ, in 1992–1993 in the DSP & VLSI Systems Research Department working on SRAM design. From 1995 to 1998, he was a Principal Investigator of research at Bell Laboratories working on the design of high-speed cache memories and low-power digital signal-processing architectures. He is currently based at the Bell Laboratories Australia site in Sydney and is working on baseband processors for next-generation wireless communications systems. He has received two U.S. patents and serves on the program committees of the IEEE International Low Power Symposium and the IEEE International Solid-State Circuits Conference.



J. H. O'Neill received the B.S. degree in electronic engineering and the M.S. degree in computer science from Monmouth College, West Long Branch, NJ, in 1980 and 1984, respectively.

He joined Bell Laboratories, Holmdel, NJ, in 1980, has been a Member of Technical Staff since 1985, and is currently in the DSP & VLSI Systems Research Department of Lucent Technologies. His research interests include VLSI systems and circuit design for communications.



J. Othmer received the B.S.E.E. and M.S.E.E. degrees from Monmouth College, West Long Branch, NJ, in 1982 and 1987, respectively.

He joined AT&T Bell Labs, Holmdel, NJ, in 1982. He is currently a Member of the Technical Staff in the DSP & VLSI Systems Research Department of Lucent Technologies. His research interests include video coding, VLSI, and CAD.

E. Säckinger (S'84-M'91) was born in Basel, Switzerland, on August 13, 1959. He received the Diploma and Ph.D. degrees in electrical engineering from the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, in 1983 and 1989, respectively.

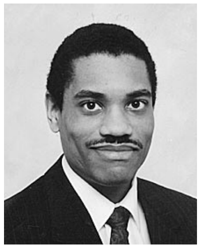
In 1983 he joined the Electronics Laboratory of ETH, where he investigated analog circuits with floating-gate devices. His doctoral work was on the regulated cascode circuit and the differential difference amplifier. In 1989, he joined AT&T Bell Laboratories in Holmdel, NJ, where he did research on artificial neural-network VLSI chips and their application to pattern recognition. In 1996, Bell Laboratories became part of Lucent Technologies, where he has been working since on VLSI chips for parallel signal processing and analog front ends for optical communication systems.

K. J. Singh (S'87-M'93) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1986 and the Ph.D. degree in electrical engineering from the University of California, Berkeley, in 1992.

His thesis dealt with performance optimization of digital circuits during logic synthesis. Since 1992, he has been a Principal Investigator in the DSP & VLSI Systems Research Department at Bell Laboratories, Holmdel, NJ, working on aspects of digital system design. He developed algorithms to extract logic functionality from layout. Since 1995, he has focused on digital design, contributing to the design of a single-chip CMOS camera, a multiprocessor DSP, and an IC for high-speed packet filtering. His current research focus is the architecture and implementation of multiprocessors and systems for high-speed data networking.

J. Sweet, photograph and biography not available at the time of publication.

C. J. Terman is a Senior Lecturer in the Electrical Engineering and Computer Science Department and a member of the Computer Architecture Group in the Laboratory for Computer Science, Massachusetts Institute of Technology (MIT), Cambridge. He implemented several early prototypes for microprocessor-based workstations that led to the development of the IEEE-1196 Bus Architecture (the NuBus), which introduced the idea of plug-and-play peripherals. In his work on CAD tools for VLSI circuits, he developed several algorithms for full-chip transistor-level timing simulation including RSIM, which was widely used in industry and academia. During ten years in industry, he cofounded several firms, including Symbolic Inc. (manufacturer of Lisp Machines), TLW Inc. (VLSI designs for communications and multimedia), and Curl Co. (software technology for the Web). After returning to MIT, he has worked on developing educational technology for use in teaching design-oriented courses.



J. Williams was born in New York, NY, in 1965. He received the B.S. and M.S. degrees in computer engineering from Rutgers—The State University, New Brunswick, NJ, in 1992.

Since 1992, he has been working in the research organization of Lucent Technologies, Bell Laboratories, Holmdel, NJ. His current interests include VLSI design of highly parallel DSP systems, parallel processor architecture, and design methodology for deep submicrometer systems on a chip.