200-MHz Superscalar RISC Microprocessor

Nader Vasseghi, Kenneth Yeager, Egino Sarto, and Mahdi Seddighnezhad

***Abstract*— Design and implementation details of the MIPS R10000, 200-MHz, 64-b superscalar dynamic issue RISC microprocessor is presented. It fetches and decodes four instructions per cycle and dynamically issues them to five fully pipelined, low latency execution units. Its hierarchical nonblocking memory system helps hide memory latency with two levels of set-associative, write-back caches. The processor has over 6.8 M transistors and is built in 3.3-V, 0.30-μm, four-layer metal CMOS technology with under 30 W of power consumption. The processor delivers peak performance of Spec95int of 9 and Spec95fp of 19 operating at 200 MHz. Clock and power distribution as well as circuit design techniques of several blocks is addressed.**

I. Introduction

THIS processor[1] is a dynamic issue superscalar microprocessor that implements the 64-b MIPS-4 instruction set architecture [1]. It fetches and decodes four instructions in order and dynamically issues them to five pipelined execution units after dependency resolution.

The chip includes the processor, floating-point units, two 32 Kbyte primary caches for instructions and data, secondary cache controller, and a 64-b system interface. The 16.6 $\times$ 17.8 mm chip is implemented in a 3.3-V, 0.3-μm, four-layer metal CMOS process and contains approximately 6.8 million transistors.

The design aims for high performance even in large real-world applications which have poor memory locality. With speculative execution, it calculates memory addresses and initiates cache refills early. Its hierarchical nonblocking memory system helps hide memory latency with two levels of set-associative caches. It delivers peak performance of Spec95int of 9 and Spec95fp of 19 operating at 200 MHz [7], [8].

To cope with complexity inherent in dynamic issue superscalar designs, it uses a modular design approach in which much of the control logic is within regular custom design structures.

II. Microarchitecture Overview

This is a dynamic issue superscalar microprocessor that implements the 64-b MIPS-4 instruction set architecture [1]. It fetches and decodes four instructions in order and dynamically issues them to five pipelined execution units after dependency resolution. Instructions graduate in program order upon completion. It provides sequential memory consistency and precise exception handling. Fig. 1 contains a block diagram and pipeline diagram for this processor. Six pipelines process instructions.

Manuscript received April 1996; revised July 22, 1996.

The authors are with Silicon Graphics Inc., Mountain View, CA 94043-1389 USA.

Publisher Item Identifier S 0018-9200(96)07775-X.

[1]This processor is the MIPS R10000 microprocessor.

A. Instruction Fetch Pipeline

The first pipeline fetches and decodes four instructions per cycle from the instruction cache. To facilitate rapid decoding, instructions are predecoded as they are refilled into this cache. Predecoding rearranges fields of each 32-b instruction and appends a 4-b unit code which identifies which functional unit will execute it. Each decoded instruction is recorded in a 32-entry active list which keeps track of the original program order.

During pipeline stage 1, instructions are fetched sequentially until a jump or conditional branch instruction selects a new program address. Target addresses are calculated during stage 2, therefore causing a single-cycle gap in the pipeline. The efficient execution of a branch instruction requires complex circuits, since the condition tested by the branch might not be known for many cycles. Rather than waiting, the path taken by the branch is predicted using a 512 entry by 2-b prediction table [3]. The current program state is saved in a four-entry branch stack. Subsequent instructions are fetched and executed speculatively. If the prediction is later verified to be correct, these speculative actions speed the program's execution. Otherwise, they are discarded, the program state is restored using the branch stack, and fetching is resumed along the other path. Instructions are fetched and executed speculatively along the predicted branch path, for up to four deep.

During stage 2, four instructions are decoded and loaded into one of three instructions queues, depending on which functional unit is required. Instructions are renamed as they are decoded. Register renaming is an elegant method for keeping track of register dependencies in an out-of-order processor.

Integer and floating-point registers are mapped separately, using similar circuit structures. Each logical register (the 5-b number in the instruction field) is mapped to a physical register. There are more physical registers (64) than logical registers (32), because the register files contain both committed and temporary values. A 32-entry mapping table assigns each logical register to the physical register which contains its current value. A 32-entry free list contains a list of physical registers which are not currently used. As each instruction is decoded, its operand registers are mapped to 6-b physical register numbers, and its destination register is assigned to the next register from the free list. This new assignment is written into the mapping table and the previous assignment is copied into the active list. Later, when this instruction graduates, this old physical register is returned to the free list for reuse. Thus, values are uniquely assigned to physical registers. After mapping, dependencies can be determined simply by comparing physical register numbers, without regard to instruction order.

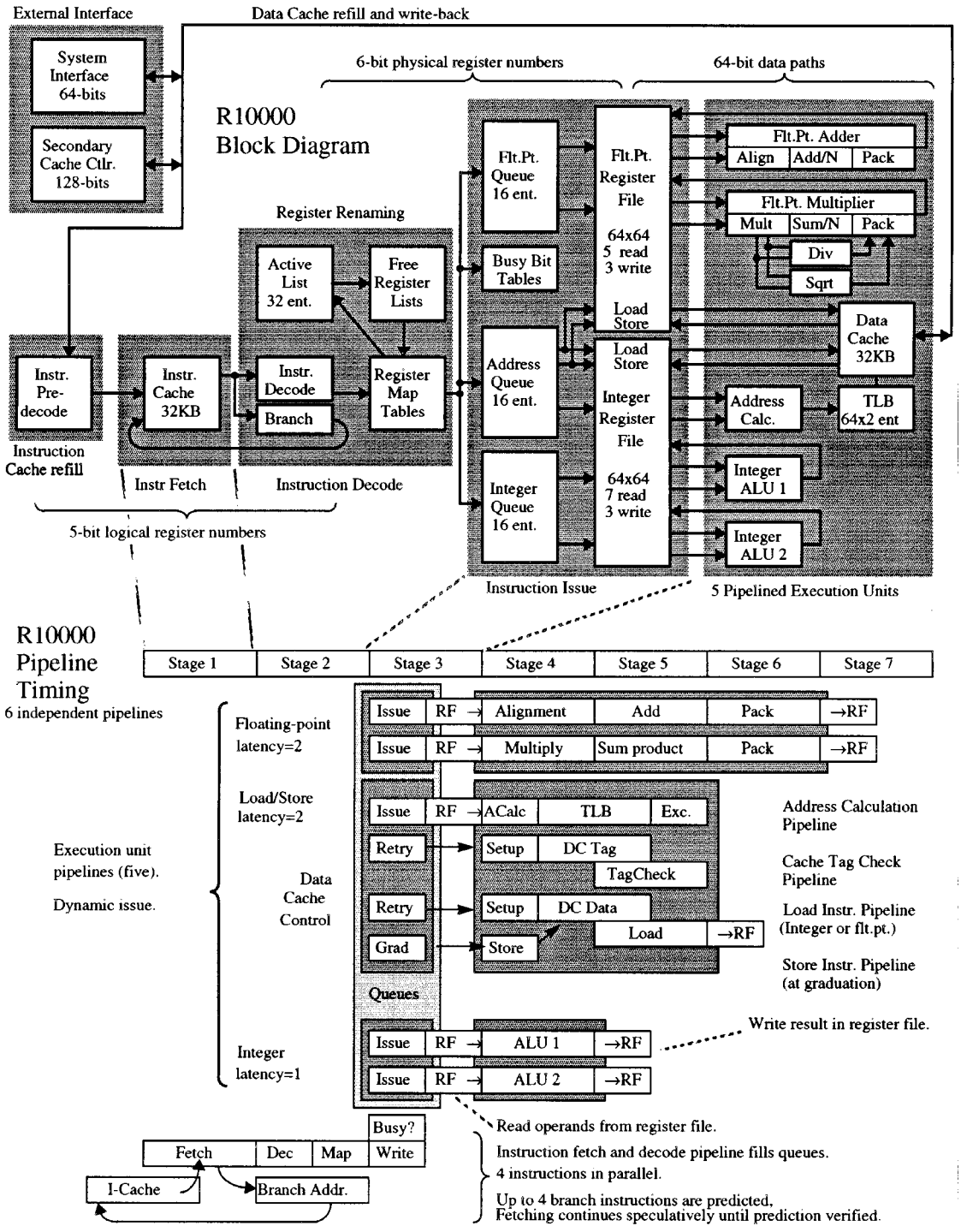

Fig. 1. Functional block diagram and pipeline timing.

Whenever a branch is predicted, the branch stack makes a complete "shadow" copy of both mapping tables, the active list write pointer, and both free list read pointers. When the misprediction is detected, this shadow is copied back into the mapping tables in a single cycle.

B. Execution Pipelines

Instructions are executed in parallel in five pipelined execution units. These units include two integer arithmetic logic units (ALU's), a floating-point multiplier, a floating-point adder, and a load/store unit. Each execution unit is pipelined with a single cycle repeat rate. Integer ALU's have a single-cycle latency and floating point units have two-cycle latency with result bypassing. Two independent iterative units perform floating point division and square-root operations and share the register file ports with the multiplier.

The three instruction queues dynamically issue operations to the execution units after all their operands become available. The integer queue issues instructions to the two ALU's. The floating-point queue issues instructions to the floating-point

TABLE I
TECHNOLOGY CHARACTERISTICS

Parameters	Characteristics
Process	Twin well on P EPI
Channel length (drawn)	0.3μm
Gate oxide	75 Å
Vtn/Vtp	0.5 V/-0.5 V
Power supply	3.3 V nominal
RAM Cell	6T, 29μm^2
Feature size (contact)	0.5 μm
Metal 1	0.6 μm thick, 1.0 μm pitch
Metal 2	0.7 μm thick, 1.2 μm pitch
Metal 3	1.0 μm thick, 1.4 μm pitch
Metal 4	1.6 μm thick, 8.0 μm pitch

multiplier and adder. An instruction is deleted from either of these two queues as soon as it is issued. Instructions have no particular order within these queues.

The address queue issues instructions to the address calculation unit and the data cache. The data cache contains two independent banks which can operate concurrently to perform load, store, and refill operations. Load and store operations can be performed out of order and can overlap with up to four cache refill operations. For each refill, the critical word is transferred first and can be bypassed directly to an execution unit.

C. External Interface

Memory latency has a major impact on processor performance. To run large programs efficiently, this processor has a nonblocking memory hierarchy with two levels of set associative caches. The processor includes on chip 32 Kbyte two-way set associative primary data and instruction caches. An external secondary cache is scalable from 512 K to 16 Mbytes of memory using synchronous SRAM's. With out-of-order speculative execution, it finds cache misses early and begins refills while other instructions are being executed in parallel, and therefore hiding memory latencies [6].

III. GLOBAL IMPLEMENTATION

A. Process Technology

The processor is implemented on a 3.3-V, 0.3-μm twin-well CMOS process. The major process characteristics are shown in Table I. Although multiple technology partners manufacture this processor, the layout is drawn with a single set of design rules that serves as a common denominator among the partners. Upon tape-out, the final database goes through appropriate sizing operations to meet the specific partner's design rules. Six-transistor cells are used for the RAM arrays for stability and noise immunity. Approximate RAM cell size is 29 μm^2. It is a four-layer metal technology with the top layer, which is about twice as thick as the lower layer metals, used primarily for global chip power and clock distribution.

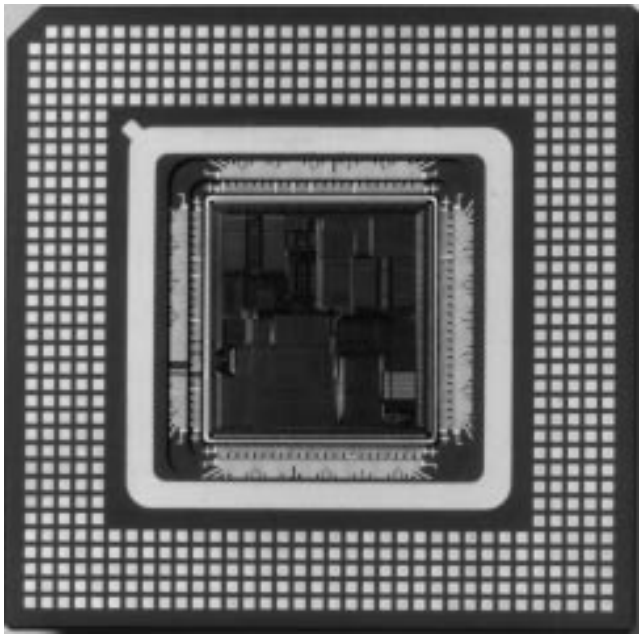

Fig. 2. Chip and package micrograph.

TABLE II
CHIP AND PACKAGE FEATURES

Parameter	Characteristics
Die Size	16.6mm X 17.8mm
Transistor Count	6.8 Million
Package	599 pin CLGA
Bond Pads	364 signal, 105 Vss, 54 Vcc, 69 Vccq
IO types	LV-CMOS or HSTL
On-Chip decoupling cap.	50 nF
Power consumption	30W @ 200MHz

B. Chip and Package

The chip contains about 6.8 million transistors including 4.4 million in the cache arrays. The die size is 16.6 $\times$ 17.8 mm and it is packaged in a 599 pin ceramic land grid array (CLGA) as shown in Fig. 2. Table II summarizes major chip and package features.

The processor and the CLGA package are designed for conventional die-attach and wirebond assembly. The package is an 11-layer cavity-down ceramic package with a Cu–W (Copper–Tungsten) slug for efficient thermal performance. The package was designed with controlled characteristic impedance signal traces and designed to minimize power and ground inductance with a simultaneous switching output (SSO) to power-ground pair ratio of 4:1. The package was co-designed with the input/output (I/O) buffers to guarantee acceptable SSO-noise and crosstalk for 200 MHz I/O operation. The package mounts on the next-level substrate using a high-performance socket designed specifically for the package.

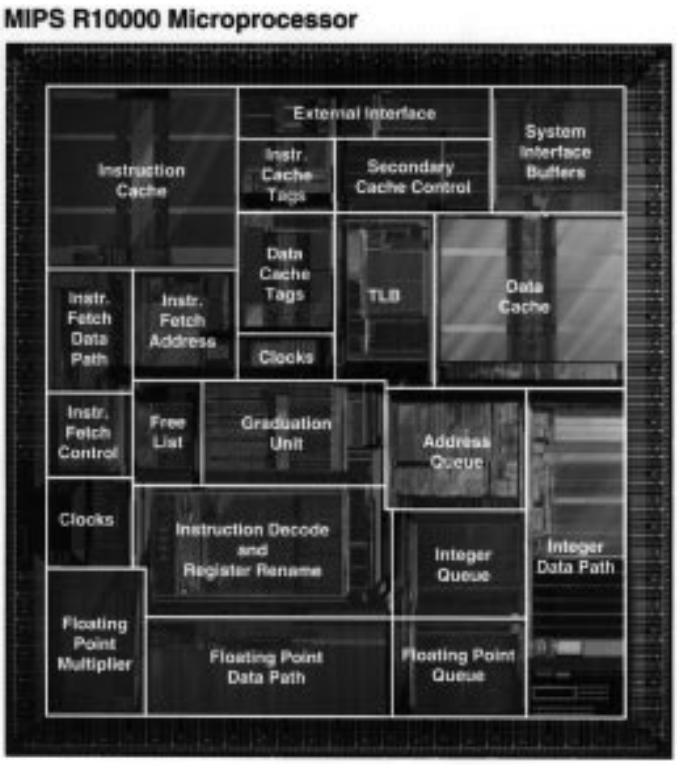

Fig. 3. Chip micrograph.

Four types of I/O buffers drive the system interface, secondary cache data, address, and clock signals. Each type can be separately configured to conform to either low voltage CMOS (LV-CMOS) or high speed transistor logic (HSTL) standards. The buffer design for each group has special characteristics. The system interface buffer contains additional open-drain pull down transistors to drive HSTL Class-2 multidrop buses [1]. The secondary cache data buffer is designed to reduce overlap current spikes when switching, thus reducing simultaneous switching noise. The cache address buffer uses large totem-pole transistors to rapidly drive multiple distributed loads. The cache clock buffer drives low impedance differential signals with minimum output delay. These clocks are precisely aligned by a low jitter delay element which is configured to adjust for propagation delays in the printed circuit board clock net.

There is about 50 nF of explicit on-chip de-coupling capacitor between power and ground to control power supply switching noise. The chip dissipates 30 W running at 200 MHz and 3.3 V supply voltage. This enables practical usage of this processor in both high-end as well as mid-range system configurations. Fig. 3 shows the chip micrograph and outlines the processor's major functional blocks.

C. Clock Distribution

An on-chip phase-locked loop (PLL) takes the system clock input and synthesizes a low skew internal processor clock, a system interface clock, and secondary cache clock [2]. Independent clock divisors allow the choice of five secondary cache and seven system interface clock frequencies. This allows for flexible and scalable system designs while taking full advantage of the processor speed.

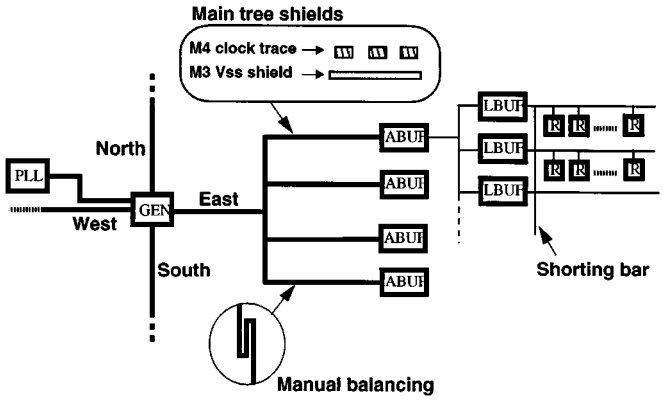

Fig. 4. Clock distribution.

Both 200 and 400 MHz clocks are distributed to four quadrants of the chip and balanced to form 16 equal phase points at each area buffer.

A parallel balanced clock tree is used to provide six copies of a programmable delay clock for the synchronous secondary SRAM's. This allows skew compensation without introducing additional jitter between the processor and the secondary cache clocks.

The PLL generates 200 MHz and 400 MHz clocks, which are sent to the center, and are then distributed to the four quadrants of the chip in a balanced tree. For added isolation and controllability, the main clock trunks are distributed on thicker metal 4 layer over metal 3 ground shields as shown on Fig. 4. Short branches are manually jogged to make them equidistant. All 16 branches are further tuned and electrically balanced to within 10 ps using results from a three-dimensional (3-D) run length code (RLC) extraction and simulation model. Waveform comparisons of RC against RLC extraction models showed that inductance effects need to be taken into account for accurate delay and skew predictions on the main clock net. For most other nets, where wire resistances dominated, inductance effects did not have a significant contribution at the operating frequencies and were ignored.

Area clock buffers at the end of each branch use the 400 MHz clock signal to resynchronize the other clocks and perform divide-by-two to achieve 50% duty cycle. Pad area buffers also generate divide-by-two to eight frequency clocks for the I/O interface and secondary cache.

Local clock buffers generate the complementary two-phase clocks which are used for all register, latch, and clocked elements. Clock buffers use a common layout and orientation throughout the chip for device matching and include decoupling capacitors for switching noise suppression. After size balancing, the outputs from local buffers within each area buffer region are shorted together to provide a means to reduce the effects of extraction inaccuracies and process variations.

Tools were developed to automatically size and balance the local clock buffers based on layout parasitic extraction and perform clock skew margin checks throughout the design.

Measured worst-case clock skews of 200 ps have been achieved at the output of the local clock buffer stages with this scheme using electron beam probing.

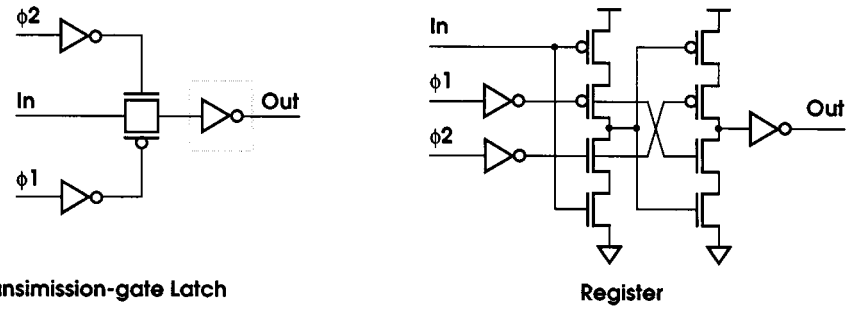

Fig. 5. Register and latch structures.

D. Register and Latch Methodology

Dynamic latches are used for speed. Where data to output delays are important, simple pass gate latches are used. Data feedthrough problems are avoided by ensuring a minimum of three inverter delays between clocking points. When back-to-back latches are required, dynamic stacked latches are used due to immunity to clock skew and data feedthrough problems [5]. Fig. 5 shows register and latch implementations. Careful standard layout styles were used to minimize coupling to dynamic nodes and maintain good noise immunity.

Local to register/latch inverter buffers are used to improve and control the final clock edge rates and reduce clock loading. For speed critical areas, logic functions and/or multiplexers at the input or output of the latch were merged into the latch structure to reduce levels of logic.

E. Power Distribution

Ideally, both power and clock distribution want to be on the thicker, low-resistance, top-layer metal. This requirement is hard to be satisfied with a balanced clock tree style due to its irregularity. Fig. 6 shows global power distribution on metal 4 layer while maintaining balanced clock tree distribution on metal 4 over metal 3 ground shields. The clock tree was first routed on third-layer metal using a global auto router. The traces were then mirrored to the forth-layer metal and the parallel metal 3 shadows underneath were grounded. Efficient pad utilization is achieved by dividing the chip into four power regions corresponding to the four edges, from which the power is distributed to the center of the chip on metal 4. Power is propagated down through metal 3 and 2 regular grid structures in each region. The power regions are stitched together using metal 2 layer across metal 3 clock shields. Equal phase area clock buffer points have been distributed evenly based on the specific clock usage and loading requirements of each region. The area to local clock buffer distribution is performed on metal 3 layer with adjacent shields for added controllability.

Metal 4 power routing and hook-up to metal 3 power grid was script driven, which allowed flexibility and fast turn around in cases of changes.

F. Repeater Insertion

Interconnect delays have become a dominant factor, limiting the performance of submicron VLSI designs. Large RC delays in long wires cause timing violations and signal integrity

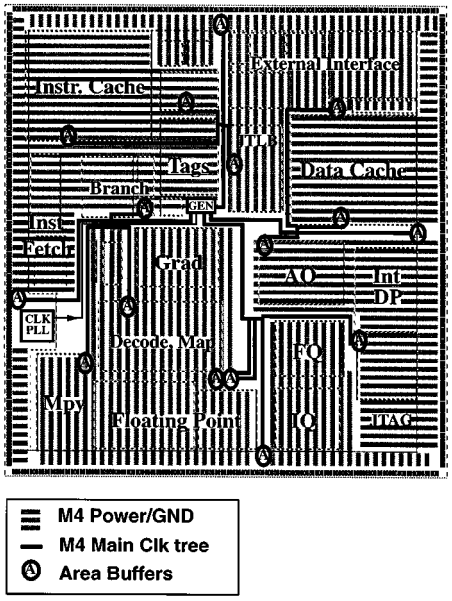

Fig. 6. Power distribution.

problems. Careful attention was given to the floorplan and the modularity of the design to keep the length of critical global wires short. In spite of that, over 6000 global nets needed to be examined. Due to the large number of global nets with multiple irregular branch segments, tools were written to identify and analyze problem nets, place and allocate optimum repeater sites, and modify netlists. Flexible, nonintrusive repeater placements were necessary to avoid blockage of channel routing areas. The repeater cells are abutted at the edges of top level blocks, introducing only metal 1 blockages in the routing channel, and share the adjacent block's power grid structure. Where increased drive was desired, multiple repeaters cells were used in parallel.

Fig. 7 shows the repeater insertion flow. The netlist with no repeaters was fed into the global auto router. From this initial route, global wire RC information is obtained and given to the repeater insertion tool, along with the gate capacitance information from the original netlist. At this stage, user preference repeater locations are also provided to the tool. The resulting RC network is analyzed and optimum repeater locations are identified and mapped onto the allocated repeater sites on the chip. The tool calculates signal delays and the theoretical optimum repeater-location coordinates are determined. These optimum coordinates are mapped to available repeater sites.

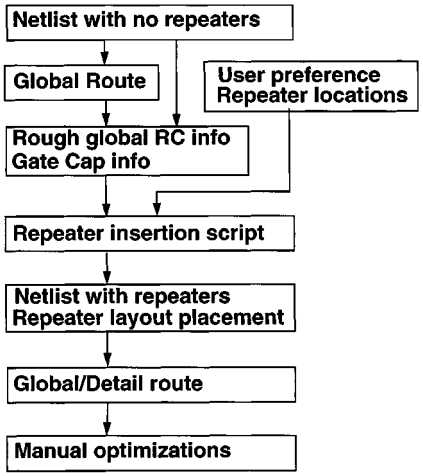

Fig. 7. Repeater insertion flow.

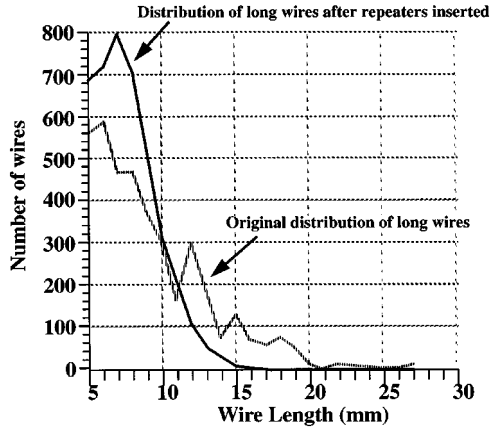

Fig. 8. Global wire length before and after repeater insertion.

The tool assigns a repeater location that minimizes any delay increase for cases where a repeater was not available at the optimum coordinate. In the case of signals with multiple branches, repeaters could be required in each branch leg. This would lead to an increase in necessary repeater sites. The tool attempts to "slide" repeaters back to the point where individual branch repeaters collapse to a single site. This is often possible when the length of the line is minimally over the maximum allowable wire length. The tool then generates a new netlist with repeaters and modifies the schematic database. With this new netlist and allocated repeater site information, the global auto router is invoked for the final global and detailed route. Further manual optimizations are performed at this stage if necessary.

About 700 noninverting repeaters were inserted into the long signal nets. Fig. 8 shows the distribution of global wire length before and after the repeaters were inserted.

IV. Functional Block Implementation

A. Integer Load Path

Load latency is the minimum number of cycles after issuing a load instruction before issuing a dependent instruction. This latency significantly affects the performance of integer programs, because addresses are often loaded from memory. Load latency was optimized to be only two cycles using a combination of logic and circuit techniques. Both the data path and control signals are time critical.

Load instruction execution is depicted in Fig. 9. Each load instruction executes the following steps.

1) The address queue dynamically issues the load instruction after all its operands become ready.
2) The integer register file reads the instruction's index register(s).
3) A 64-b adder calculates the instruction's "virtual memory address" as the sum of an index register and a 16-b immediate value, or as the sum of two index registers. Bit 5 of this address selects between the two banks of the data cache. Bits 13 : 6 are used as an index for addressing locations in the cache.
4) The translation look-aside buffer (TLB) translates a virtual page (addressed using bits 63 : 12 of the virtual address) into a 28-b physical page number. The page number and the low 12 address bits are concatenated into a 40-b "physical memory address."
5) The tag array of the selected cache bank reads two address tags.
6) The data array of the selected cache bank reads two 64-b data doublewords.
7) Each address tags is compared to the physical page number to generate a cache hit signal for each of the two cache ways.
8) The cache hit signal selects data from the addressed way.

Operands can be bypassed around the register file. Thus, the latency is computed beginning with step 3).

The 64-b virtual address is generated in half a cycle using a carry select scheme with dynamic hybrid 2-b carry look ahead (CLA), and 2-b Manchester carry chain. The TLB, the cache tag arrays, and cache data arrays operate in parallel. The TLB matches the high address bits with virtual page addresses stored in a 64-entry content-addressable-memory (CAM). If an entry matches, a corresponding physical page address is read from a 128-entry RAM. (Two physical page addresses are stored for each virtual page address.) Low address bits are decoded to select a row within the cache tag and data arrays.

The address compare and data selection use domino dynamic logic for speed. Each of the two cache ways contains a dynamic 28-b comparator. This comparator combines the outputs of 28 static exclusive-or gates using a 28-input dynamic-or gate. This circuit creates a pulse if any tag bit does *not* match the corresponding address bit. The resulting two "cache miss" signals drive 64 dynamic two-wide and-or gates. Because a "miss" indicates that the corresponding data is not wanted, each "miss" signal enables data from the opposite cache way. This circuit is logically equivalent to a multiplexer when the addressed data is in the cache. Otherwise, it uselessly logically OR's data from the two cache ways, but in this case its output is ignored anyway.

The Mips instruction set architecture defines byte, halfword, word, and doubleword data types [1]. The shorter operands

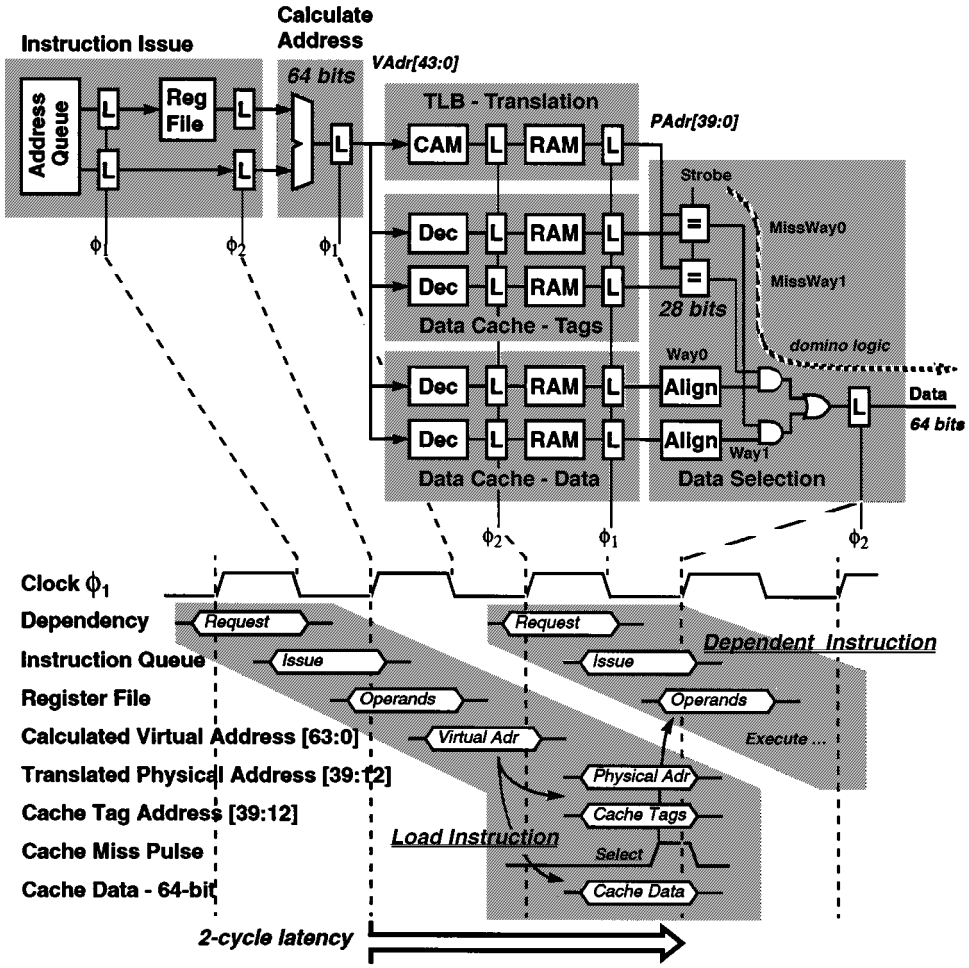

Fig. 9. Load instruction execution.

require "byte alignment" based on the low three address bits. To avoid adding delay, the load alignment circuitry is duplicated for each cache way so that it can operate in parallel with the tag comparator logic.

The cache miss pulses are gated together to generate a "LoadDone" signal, which indicates if the load instruction was completed successfully. This is a timing critical signal, because it affects the "operand busy" logic in the register rename logic and in all three instruction queues. Since it is a single signal with many loads, it is driven by a very large buffer and is routed with wide traces. Because it is generated using domino gates, one edge of switching is important for critical path timing. This edge is optimized by ratioing the transistors in drivers and following repeaters.

To achieve two-cycle load latency, dependent instructions must be issued tentatively, one cycle before the "LoadDone" signal is generated. Thus, the request logic assumes that the cache will "hit." If it does not, issuing of any dependent instruction will be aborted.

B. Instruction Queues and Dependency Matrix

Each instruction queue contains 16 entries. Decoded instructions are held in the queues until they are selected and issued to the corresponding execution units. For speed criticality, the instruction issue and priority selection uses domino logic and is implemented in two stages as shown in Fig. 10. The first-level multiplexers select one instruction from each group of four entries. The second level selects one of these four groups, or bypasses a newly decoded instruction, for conditional issue. The control selects for these multiplexers are generated using two levels of 4-b priority encoders. Each level contains duplicate circuits to arbitrate between high and low priority requests. The low priority request is used only if there are no higher priority requests. This organization reduces delay by evaluating first-level multiplexers while the second-level priority encoders generate the select signals required by the second-level multiplexers.

Each 4-b priority encoder, shown in Fig. 11, uses a combined domino circuit structure with six outputs. Evaluation starts in phase 1 high, beginning with request 0 and works up. For each pair, if the request is active, the corresponding output is evaluated and subsequent requests are ignored. Intermediate nodes within the ladder get indirectly precharged to $V_{cc} - V_{tn}$ since one of each transistor pair is on. This avoids charge sharing problems, without adding extra transistors to precharge the intermediate nodes. The "AnyL" and "NoL" outputs go to the second stage priority encoders, and L0 to L3 outputs provide the control for the first stage multiplexer path, shown in Fig. 10.

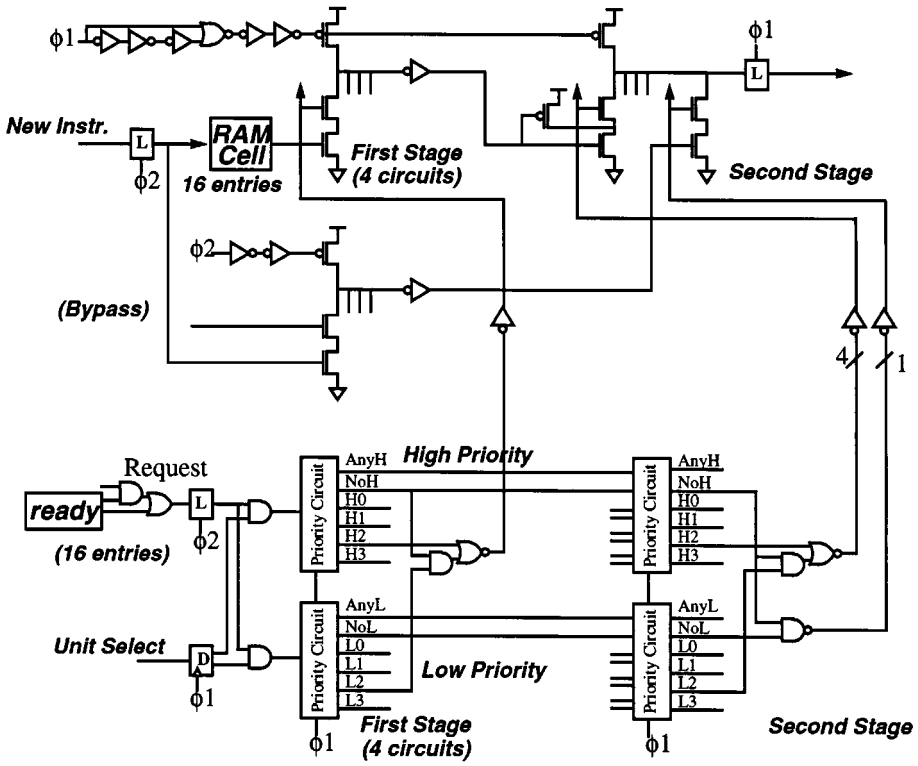

Fig. 10. Queue issue circuit.

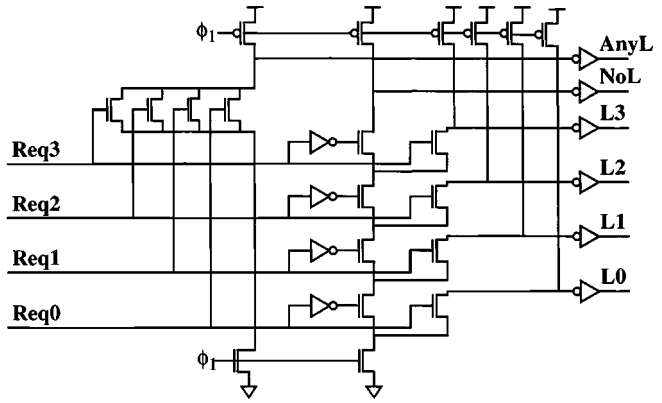

Fig. 11. Queue priority circuit.

Two 16 × 16-b matrices resolve dependencies between memory accesses. Fig. 12 shows a bit structure representation of cache dependency and store dependency matrix arrays. The cache dependency bits avoid unnecessary cache thrashing by tracking which entries access the same cache set. Store dependency bits track dependencies of a load on previous store instructions. The two memory array matrices are interleaved and pitch matched with the queue array. Row J and column K correspond to instructions in the queue. Bit [J, K] indicates that instruction J depends on instruction K. Dep [J, K] is set if entry [J] is dependant on entry [K]. Dep [J, K] is reset if either entry is not active.

When the memory address is computed for each new instruction, a 16-entry CAM array compares it to addresses already in the queue. The comparator outputs are written into the matrices' corresponding row. The bits in each row are

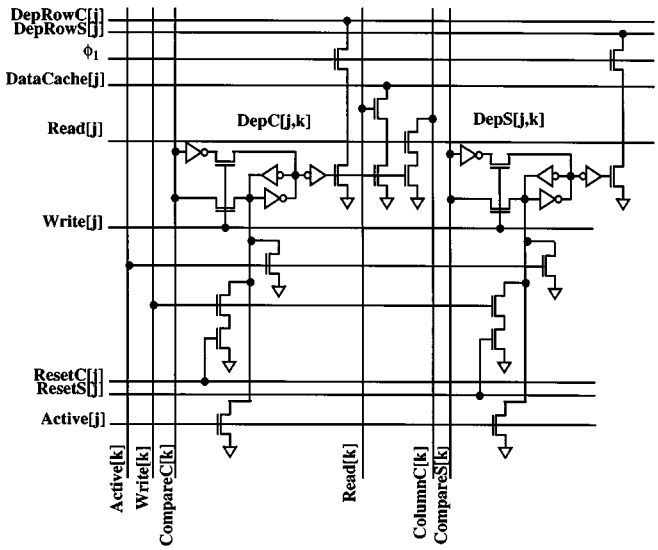

Fig. 12. Dependency matrix.

ORed together to form a 15-b distributed dynamic OR gate evaluated during phase 1. If any bit in this row is set, the corresponding instruction has a dependency and must wait. Each bit will be reset when the instruction that is causing the dependency is graduated and no longer active.

C. FP Multiplier

The floating point multiplier array uses radix 4 booth recoding leading to 27 partial products. A binary compression tree enables symmetric and ideal usage of four to two compressors [4].

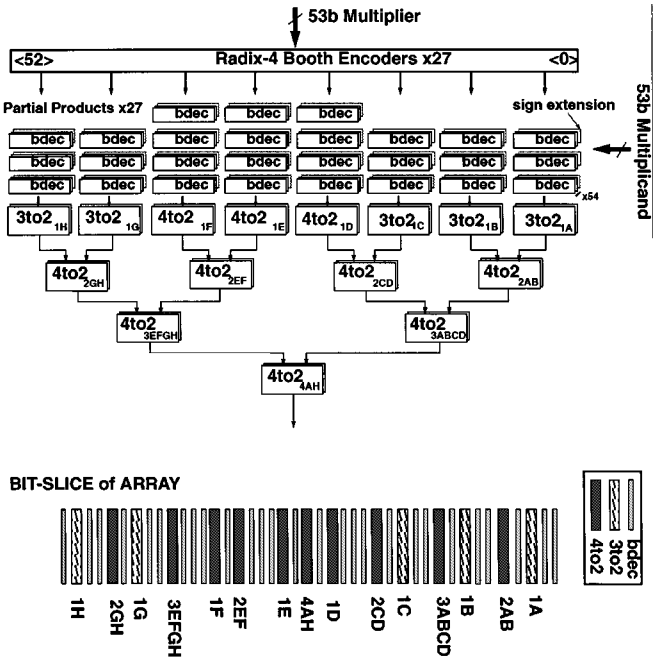

Fig. 13. FP multiplier array.

Since the design is constrained to fit in a datapath, a booth encode is embedded within the array to minimize vertical tracks. The tree is made up of static differential pass transistor four to two compressors which lead to fast, low-power, and compact layout. The circuit does not require a clock and has good noise immunity due to its balanced noise coupling.

As can be seen from the wiring track bit slice of the array, shown in Fig. 13, Booth decoding is interleaved among the compressors. This provides efficient use of metal by reutilizing metal tracks, which lead to short wires that self-isolate themselves.

Unlike most tree arrays, partial products are bit aligned, where the multiplicand traverses the array diagonally. Sum and carry, which are the more critical paths, are kept as short as possible within the tree structure. The multiplier array is folded to an effective width of about 70 b. The axis was determined by the optimum placement of subsequent stages of multiplier datapath. The 53 single-ended multiplier bits travel vertically to the embedded encoders. Differential 1X, 2X, and Sign outputs travel horizontally to the Booth decoders. The 53 multiplicand bits are buffered at the bottom and top of the array and share a common bypassed input. Repeaters are provided near the center of the array to reduce the long RC delay. Fig. 14 shows the folded array structure. Metal 1 is used for local interconnects. Metal 2 is used for global booth decode lines and for multiplicand routing. Metal 3 is used for multiplier, tree routing, and multiplicand routing. Because booth encoding generates both positive and negative numbers, sign extension and two's complement must be done. Sign generate and two's complement cells were placed in the array with the goal of maintaining a regular shape. Finally, eight columns of redundancy were added to generate the lateral carries needed for the most significant bit (MSB) portion of the array and to generate a convenient carry out for the 106 bit carry propagate adder.

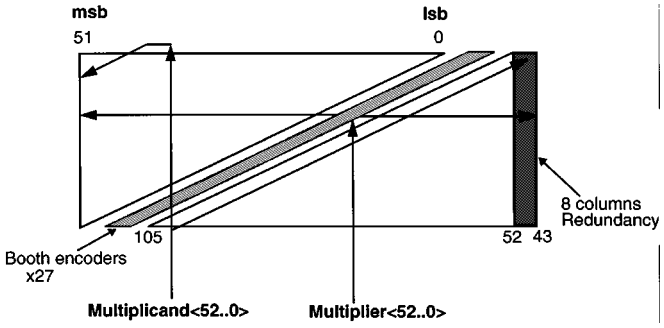

Fig. 14. FP multiply folded array.

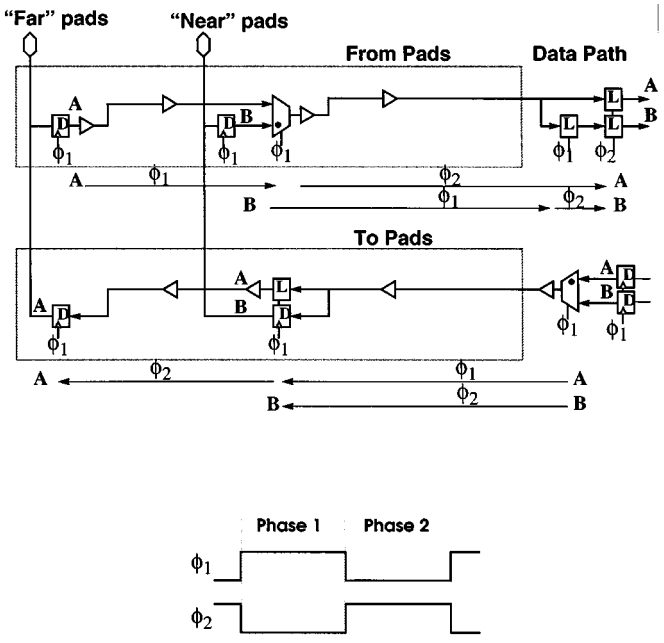

Fig. 15. Biphase bus.

D. Biphase Bus

Nearly 300 bidirectional pins must be connected from the pad ring to the core. Because these long wires require repeaters, two unidirectional buses were used instead of a single bidirectional (tristate) bus. To avoid doubling the number of wires, biphase multiplexing is used to send two bits on each wire as shown in Fig. 15.

Each biphase multiplexer combines signals from a near pad with a far pad. During phase 1, the far signal travels halfway and arrives at the multiplexer, while the near signal is transferred to the core on the biphase bus. During phase 2, the multiplexer switches and drives the far signal to the core. For outputs, the biphase bus transfers a bit for the far pad during phase 1. This bit is latched midway to hold the signal during phase 2, reaching the far PAD, during which time the multiplexer switches to transfer a bit to the near pad.

V. Design Verification and CAD Tools

A. Extraction and Timing Verification

To achieve the target timing goals on this large design, resistance and capacitance extraction, with coupling effects to adjacent nets, was required for the full chip timing verification.

To be able to manage the throughput time required and overcome capacity limitations of the extraction tool, a two-level hierarchy was adopted for the chip. The top level contained cells for the interblock routing as well as placement of 35 functional blocks. The blocks and routing could have the layout parasitics extracted independently of each other with turnaround times varying from few hours to 24 h. When feeding the data to the timing tool, this approach allowed the mixing of top blocks that were finalized with blocks that had partial layout and therefore estimated parasitics. Also, a tool was utilized that allowed back-annotation of parasitics on blocks that were previously extracted but were still incomplete due to last minute changes. In this fashion, a block could have the right parasitics on over 99% of the nets, depending on the severity of the change in progress, and estimated numbers for the areas that were being changed and final layout unavailable.

An in-house developed static timing analyzer to deal with transparent latch-based designs was utilized. It was optimized to be able to verify the whole design in one pass. A two-level hierarchical circuit netlist would be assembled and fed to the timing tool. The underlying top blocks could be in different stages of the design. Blocks could be empty, have schematics with estimated capacitance, have schematics with extracted capacitance annotation, and on the final stage, be complete with extracted resistance and capacitance annotation. This setup gave maximum flexibility allowing one or few blocks, as seen from the top level, run on users workstation or the full design run on specially configured high-performance compute servers. The small cluster of blocks approach allowed designers to verify their timing fixes with a quick turnaround time, without having to wait for the results of the jobs on the full chip, which could take from 24–30 h. Supporting tools were developed to assist designers visualize common subpaths out of several different critical paths. This enabled designers to focus circuit fixes to maximum leverage locations.

B. Physical Design Verification

Extensive layout and circuit guideline checks were performed to catch risk areas. Dynamic structures have been extensively used throughout the design to gain speed. To control noise and guarantee functionality, several signal integrity and dynamic design rule checks were performed on the design to identify risk areas.

The combined charge sharing and coupling capacitance ratio to the total capacitance on any dynamic node is kept under 8%. Noise prone wires going directly to the input of dynamic gates are detected and corrected by either isolating the wire or adding recovery PMOS pull-ups at the input. Dynamic nodes into a multiplexer connection are flagged to avoid potential charge loss, if the multiplexer enables are not guaranteed to be stable during evaluate.

Signal coupling induced voltage dip on source/drain diffusion of a transistor can momentarily turn on the device and in case of a dynamic node following this transistor, charge loss may occur.

Signal propagation time of a wire is effected depending on the switching activity of the adjacent neighboring wires due to capacitance coupling. Reverse coupling signal switching slow down effects are taken into account for setup time checks, while forward coupling speed up effects are included for hold time checks.

C. Functional Verification

Verification of a processor with this complexity needs a very detailed, systematic, and hierarchical approach. To do that, each block of the design was studied and verified independently during the first phase of the verification to make sure that it functions properly based on the target specification for that block. During the second phase of the verification, all blocks were integrated together in a full chip simulation environment. A C-based system model including the bus controller, secondary cache array, memory controller, and memory array were also added to the model to support the full system simulation. In the final phase of the verification, the processor was integrated with real system ASIC's designed by system groups to verify the functionality of the processor in the whole system. In each phase, specific directed and random diagnostics were run on the model.

An in-house hardware description language and simulator integrated with a user-friendly graphical interface were used to design and verify the processor. An instruction level simulator was also included in this package to check the content of the architectural registers and memory hierarchy at the instruction boundary. The simulator provided useful features like backup, retry, trace, and follow. Trace and follow determines all signals which drive a specific signal, or all other signals that are driven by a specific signal. Also, another in-house tool was used to check the state transition and arc coverage of each state machine in the design. Several random code generators were also used to generate full, partial, and targeted random sequence of instructions to run on the design. These tools were tuned to generate codes to stress specific parts of the design like branch unit, instruction fetch unit, load/store unit, etc.

Three classes of directed diagnostics, random code, and real applications were developed to run on processor system in the course of its verification. Directed diagnostics were developed under two categories: architecture verification programs (AVP) in order to check mostly the MIPS instruction set, and microarchitectural verification programs (MVP), which were a result of detailed study of the behavioral model to validate the processor at the implementation level. The system model was capable of injecting different events to the design at each instruction decode cycle. This mechanism enabled us to force the processor into a specific state at each cycle and verify its behavior under that condition. Many asynchronous events, error checking, and protections, and also cache and memory coherency were tested by using these mechanisms in our directed diagnostics.

The second class of diagnostics were random codes developed by using different random code generators described earlier. We also used some of these code generators to generate random multiprocessor (MP) code to verify the MP system that was simulated by instantiating two to four processor models. The third class were real codes like UNIX and Windows NT

operating systems and also short versions of real applications from different benchmarks.

More than 50 000 diagnostic programs (100 million cycles) were developed and ran each night or after each design release. More than 5 billion cycles of random codes ran on the final design before the tapeout as well as booting three operating systems. Having a full functional chip, booting UNIX on the first silicon proved that our methodology for verifying this complex processor was a success.

D. Test and Debug Features

The processor observes internal signals with ten 128-b linear-feedback shift registers. Effectively, these signals become virtual output pins. These registers can be used in two ways. For debugging, the registers sample signals for an externally-selected cycle, and then shifts them serially out. In a tester, a succession of cycles can be sampled, effectively creating a wide logic analyzer. For production testing, the registers generate signatures which are verified at the end of each test. Fault coverage is significantly increased without requiring special ATE features. These registers are separate structures which do not affect the processor's logic or add a noticeable load to the observed signals. They use the processor's clock to ensure synchronous behavior and avoid any special clock requirements. These internal test points partition the chip into three fully observed sections. This partitioning significantly reduces the fault grading cost, because both the test sets and faults are divided into smaller problems. The observability scan chain is also used for logic and circuit debug on the tester as well as on the system. Problems on the chip can be narrowed down to a certain area by comparing the scan output of simulated RTL model to the scan output from silicon. The scan features are especially useful in debugging the processor while it is plugged into the system.

VI. Conclusion

Despite the microarchitectural and hardware complexity inherent in out-of-order superscalar processors, this chip achieves high clock rates and functional correctness. Its design emphasizes concurrency and latency hiding techniques to efficiently run large real-world applications. A single user operating system booted 13 days after the first silicon was received quickly followed by Multiuser, MP UNIX, and Windows NT running at 200 MHz.

The performance of systems with an aggressive memory hierarchy gives peak performance of Spec95int of 9 and Spec95fp of 19. Although performance can be improved by targeted compiler optimizations, existing codes run efficiently without recompilation.

Acknowledgment

The authors would like to acknowledge the contribution of many individuals who were involved with the makings of this chip, and our semiconductor partners, Toshiba and NEC corporations, for providing the enabling process technology. Project management support included A. Nayyerhabibi, J. Brennan, Z. Hossein, A. Dixit, Y. Van Atta, A. Bashteen, and S. Studelski. Designers included P. Koike, D. Freitas, R. Conrad, L. Yang, J. Chuang, H. Nguyen, J. Maneatis, Q. Nasir, A. Bomdica, L. S. Lee, S. Gupta, M. Y. Wang, R. Chang, W. Chan, C. Li, F. Leu, U. Nawathe, E. Fang, F. Lutz, G. Cuan, E. Huang, R. Hashishita, Y. Urakawa, S. Kuusinen, K. Lam, J. Loh, A. Massoumi, S. Venkat, G. Hsieh, T. Layman, G. Shippen, T. Fu, M. Khurshid, R. Martin, V. Parmar, and B. Voegtli. Design Verification team included S. Peltier, A. Ahi, A. Bhatia, Y. C. Chen, S. Lee, M. Su, L. Chang, and H. Sucar. And thanks to R. March, D. Yang, K. Sheth, F. Jen, H. Chin, P. Schmidt, and J. W. Pan for the invaluable CAD tool development and support.

References

[1] *MIPS R10000 Microprocessor User's Manual,* June 1995.
[2] N. Vasseghi *et al.*, "200 MHz superscalar RISC processor circuit design issues," in *ISSCC '96.*
[3] K. Yeager, "MIPS R10000 superscalar microprocessor," *IEEE Micro Mag.,* Apr. 1996.
[4] P. J. Song and G. De Micheli, "Circuit and architecture trade-offs for high speed multiplication," *IEEE J. Solid-State Circuits,* vol. 26, pp. 1184–1198, Sept. 1991.
[5] S. Mirapuri, M. Woodacre, and N. Vasseghi, "The mips R4000 processor," *IEEE Micro Mag.,* Apr. 1992.
[6] J. L. Hennessey and D. A. Patterson, *Computer Architecture: A Qualitative Approach.* San Mateo, CA: Morgan Kaufmann, 1990.
[7] Spec95 http://www.specbench.org.
[8] MIPS Technologies Inc., press release Jan. 18, 1996, http://www.sgi.com.

Nader Vasseghi received the B.S.E.E. degree from the University of California at Santa Barbara in 1979 and M.S.E.E. degree from Southampton University in England in 1981.

He then joined General Instruments, Optoelectronics Division and worked on the design of OptoLogic transmitter/receiver products. From 1984 to 1989, he was a designer at Advanced Micro Devices working on the design of high-speed programmable array logic devices and network controller products. Since 1989, he has been with Mips Technologies, Silicon Graphics Inc., and has been working on the design of R4000 and R10000 high performance RISC processors. He is currently the design manager for the next generation MIPS processor development. His areas of interest are high-speed circuit design, interconnect and noise issues, process technology, and global chip integration.

Kenneth Yeager received the B.S. degrees in physics and electrical engineering from the Massachusetts Institute of Technology, Cambridge.

He currently is a designer at Mips Technologies, Silicon Graphics Inc., Mountain View, CA, where he participated in the conception and design of the R10000 processor. His areas of interest include the architecture, logic, and circuit implementation of high-performance processors.

Egino Sarto received the B.S. degree in electrical engineering from Universidade de Brasilia, Brasilia, Brazil, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1979 and 1981, respectively.

Prior to joining Mips Technologies/Silicon Graphics in 1987, he was a circuit design engineer at Intel, working on fast SRAM's and microprocessor development. At Mips Technologies, where he is a Principal Engineer, he has been engaged on the design of high performance RISC microprocessors. His interests include general circuit design, timing analysis, physical layout design, and silicon processing technologies.

Mahdi Seddighnezhad received the B.S.E.E. degree from Shiraz University, Iran, in 1986 and the M.S. degree from University of Illinois at Urbana in 1992.

He has since been with Silicon Graphics Inc., Mountain View, CA, working on the design verification and performance analysis of the R10000 processor. His areas of interest are architectural validation and performance study of high performance microprocessors.