A VLSI Chip Set for a Multiprocessor Workstation:

PART I:
A RISC Microprocessor with Coprocessor Interface and Support for Symbolic Processing

Daebum Lee, et al.

PART II:
A Memory Management Unit and Cache Controller

Deog-Kyoon Jeong, et al.

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

A VLSI Chip Set for a Multiprocessor Workstation - Part I: A RISC Microprocessor with Coprocessor Interface and Support for Symbolic Processing

David D. Lee
Shing I. Kong
Mark D. Hill
George S. Taylor
David A. Hodges
Randy H. Katz
David A. Patterson

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

This two-part paper describes two key components used in building a 40-70 MIPS multiprocessor workstation. In the first part, VLSI implementation of the central processing unit (CPU) chip, based on reduced instruction set computer (RISC) architecture and with support for LISP is described. The $1.3cm^2$ CPU chip uses a direct-mapped 512-byte on-chip instruction cache, and 138 40-bit registers organized in 8 overlapping windows to achieve 10 MIPS per processor peak performance with a 10 MHz, four-phase clock.

The second part of the paper [1] describes the memory management unit and cache controller (MMU/CC) chip. System level design issues such as multiprocessor cache coherency and synchronization among chip sets are also considered in the second part. Both chips are implemented in a 1.6 μm double-layer-metal CMOS technology, and are being used in a multiprocessor workstation (SPUR) successfully executing its own operating system called Sprite as well as many applications including LISP programs.

I. Introduction

SPUR (Symbolic Processing Using RISCs) is a multiprocessor workstation developed at the University of California at Berkeley as a testbed for research on parallel processing, particularly in LISP [2]. A SPUR workstation, shown in Figure 1, can have 6 to 12 identical processors, each of which consists of a 128K-byte cache, a CPU, a floating point coprocessor, and a cache control and memory management unit (MMU/CC) that assures the cache coherency among multiple processors. The picture of a fully populated SPUR processor board is shown in Figure 2. This paper describes the VLSI implementation of the CPU chip, a 32-bit RISC microprocessor.

The SPUR CPU supports a multilevel cache scheme that includes a prefetching on-chip instruction cache, a coprocessor interface, and support for fast execution of LISP through a tagged 40 bit architecture. The coprocessor interface supports concurrent CPU and FPU operations. It uses 27 pins to implement a low-overhead interface between the CPU and the FPU. The chip, implemented in 1.6 μm, double metal CMOS technology, contains 115K transistors. The chip statistics are summarized in Table 1, and a chip photomicrograph is shown in Figure 3. An on-chip clock generator, based on a charge pump phase-locked loop with tapped delay line, provides accurate phase relationship with the board clock and also with clock phases of the other chips [3]. Nominal operating frequency with a 4-phase non-overlapping clock (18 nsec nominal per phase and 7 nsec non-overlap time between phases) is 10 MHz (12.5 MHz Max). A SPUR uniprocessor running LISP programs (Gabriel benchmarks) at 10 MHz can provide 2X performance improvement on the average over the Symbolics 3600 or VAX 8650, according to simulation [4]. A SPUR workstation with 6 to 12 processors is predicted to yield a sustained throughput of 40 to 70 MIPS, respectively.

The organization of the paper is as follows: Section II gives an overview of the CPU architecture and execution pipeline. Section III focuses on the hardware required to implement various features of the SPUR CPU architecture. Section IV describes the design, verification, and testing

This work is sponsored by DARPA under contract order 482427-25840 California MICRO, Texas Instruments, National Semiconductor, Cypress Semiconductor, Tektronix, and HP.

methodologies of the full custom SPUR CPU chip. Finally, the summary and conclusion are given in section V.

II. An Overview of the SPUR CPU Architecture

The SPUR CPU is a third-generation RISC microprocessor developed at the University of California at Berkeley. It is specifically designed to be used in the SPUR multiprocessor workstation. The architecture of the SPUR CPU is akin to those of previous RISC projects at U.C. Berkeley [5],[6]. Some new features, however, have been added: a coprocessor interface to support floating-point computation, an efficient interface to the cache-control and memory-management unit, and run-time hardware tag checking for fast execution of LISP programs. The instruction set of the SPUR CPU is carefully chosen such that an efficient implementation of the single-cycle execution of all instructions is possible.

Like previous RISC processors, the SPUR CPU is a load-store machine. Memory is accessed only through load and store instructions. All other instructions are register-to-register or immediate-to-register oriented. There are four generic instruction types: register-to-register, store, compare-and-branch, and call-jump. Load and return instructions are special cases of register-to-register in which $(R_{s1} + R_{s2})$ or $(R_{s1} + Immediate)$ is used as an effective address. The R_d field specifies the register to be loaded for the load instruction type and is not used for the return instruction type. All instructions (40 integer and 20 floating point) are 32-bits wide and use fixed formats. The seven instruction formats are shown in Figure 4. The opcode and the register specifiers are in the same positions in all formats. The three-register format (RRR) is used for loads, register-to-register operations, special register operations and co-processor operations. The two-register and one-immediate (RRI) is used for loads and register-to-register operations. Compare-and-branch instructions have three slightly different formats depending on the field specifying the condition.

The CPU registers are organized in eight overlapped windows (128 registers) and 10 global registers accessible from any window (total 32 registers visible from one window). The

overlapped window scheme considerably reduces the register save and restore overheads between procedure calls. The registers are 40-bit registers with 32 bits for data and an 8-bit tag used for runtime type checking and garbage collection. The 8-bit tag consists of a 6-bit object's type tag and a 2-bit generation numbers. LISP is supported with three types of hardware tag checking with traps to a software trap handler: data type checking for general computations, pointer type checking for list operations, and generation number checking for garbage collection based on the generation scavenging algorithm [7].

The on-chip instruction cache provides the effect of an extra memory port, allowing simultaneous data memory reference and instruction fetch by the execution unit (EU). This leads to a four-stage pipeline (Figure 5) that eliminates the need for pipeline stalling whenever a load instruction is executed. Consequently, the CPU can issue and complete one instruction per cycle (peak performance rate of 10 MIPS per processor) as long as there are no instruction or external data cache misses. Branch conflict in the pipeline is resolved by a single cycle delayed branch with one instruction in the delayed slot. Data conflicts are resolved by hardware internal forwarding logic.

In order to facilitate the high-precision floating point computations and other possible coprocessing capabilities, the SPUR CPU incorporates a parallel interface to coprocessors. The floating-point coprocessor interface implemented in the current version of the CPU chip supports concurrent CPU and FPU operations. It uses 27 pins to implement a low-overhead interface between the CPU and the FPU. The FPU tracks CPU instructions issued by the instruction cache in the CPU via 22 pins carrying opcode and register specifiers. The CPU sends 2 control signals to the FPU, and the 3-bit FPU status is sent to the CPU. The CPU treats all FPU instructions as illegal instructions when the FPU is disabled. When the FPU is enabled, all FPU instructions except FPU load and store are treated by the CPU as NO_OP. For FPU load and store, the CPU computes the effective memory address and the FPU reads and writes the data directly from the external cache.

In the SPUR instruction set, a number of special load (7) and store (3) instructions are dedicated to cache control and virtual memory management. Although these instructions look almost identical to the CPU, appropriate cache operations are provided to the external MMU/CC through the MMU/CC interface. The interface consists of a 4-bit cache-opcode, two bits indicating the mode of operations (user *vs* kernel and physical *vs* virtual), and 9 other status bits of both the CPU and the MMU/CC.

The unusual conditions that the CPU may face at runtime can be divided into four groups. Unusual conditions detected inside the CPU are called CPU exceptions: integer overflow, tag checking, window overflow and underflow, and so on. Unusual conditions caused by the FPU are called floating-point exceptions. All other unusual conditions occurring outside the CPU are called faults and interrupts. Faults occur in response to the execution of an instruction, while interrupts are asynchronous events that come from outside the processor (e.g. an i/o interrupt). The CPU responds to exceptions, faults, and interrupts by taking a vectored trap. The trap vector consists of a trap base address concatenated with the trap type field. There is a priority ordering for cases when more than one unusual condition occurs at the same time. All traps are taken during an instruction's third pipeline stage, and hence only one instruction can cause a trap in any cycle. Traps can be disabled or enabled selectively by controlling the 8 bits in both kernel and user processor status words (KPSW and UPSW).

III. Hardware Implementation of the SPUR CPU

The major functional blocks are shown in Figure 6 and outlined in the chip photomicrograph (Figure 3). The major blocks are the execution unit (EU) and the instruction unit (IU). The EU is further divided into the upper data path, the lower data path, and the control. The 30-bit upper data path contains pipelined program counters and special registers. It is used for instruction address calculations and special register references. The 40-bit lower data path is for general computation on the tagged registers.

A. The instruction unit

The SPUR IU consists of a 512-byte (128 instructions) direct-mapped (16 blocks with 8 subblocks or 8 instructions per block) instruction cache. A novel feature of the SPUR instruction cache is a valid bit associated with each instruction word in the cache so that any subset of instructions within a block may be valid. The SPUR IU uses this flexibility to reduce demand miss time by loading only the fetched instruction rather than the entire block and to permit instruction prefetching to load the rest of a block in parallel with subsequent instruction fetches [8],[9]. If subsequent prefetches are successful, the miss penalty is just two cycles for the entire block containing the missed instruction.

The IU can operate in three different modes: (1) disabled, (2) enabled-without-prefetching, and (3) enabled-with-prefetching, controlled by two bits in the Kernel Processor Status Word (KPSW). In disabled mode, the IU fetches every instruction requested by the EU from the external cache. Disabled mode is useful for initial chip testing and for allowing chips with stuck-at-type errors in the cache or tag array to function correctly, albeit more slowly. In enabled-without-prefetching mode, the IU will cache instructions upon demand misses but will not initiate any prefetches.

The normal mode is the enabled-with-prefetching. After the missed instruction is cached, prefetches are made to subsequent words within the block until another demand miss occurs or prefetch is blocked by the EU's external data access. These prefetches are "free", as they never interfere with external cache accesses, such as instruction fetch and external data reference, by the EU, because prefetch has the lowest priority. If prefetch causes an external cache miss, the cache controller simply ignores the request.

The instruction unit is controlled by two finite-state machines: one controls the fetching and the other controls the prefetching of instructions. Two finite state machines and other random control logic are partitioned into the total of 6 PLAs considering the timing constraints. Both IU and register file use the same 6T SRAM memory cell [10]. The data portion of the cache is an array of 128 33-bit words. The tags are stored in a separate array (16 24-bit words) whose access

time is significantly less than that of the data array. This allows the tag comparison to be done while the instruction is being read out from the data array. Bitwise comparison using an XOR gate is used for tag comparison and is followed by dynamic logic to determine a hit. The effective access time of the instruction cache including hit logic is under 12 nsec without using a sense amplifier.

B. The register file

The SPUR CPU has a total of 138 general-purpose registers organized in 8 overlapped windows and 10 global registers. Thirty-two registers are visible to the compiler at any one time: 10 globals, 10 locals, 6 overlapped with caller window, and 6 overlapped with callee window. Each register is 40 bits wide having a 6-bit tag, 2 bits for generation number and 32 bits for data. The same 6T SRAM cell used in the IU is used in the register file. The layout of SRAM cell is constrained by the pitches of the data path bit slice and the register decoders (two decoders per register). The result is a large but fast SRAM cell that does not require a sense amplifier.

The SPUR CPU architecture is register oriented and requires two reads and one write per cycle. The register access is time multiplexed for the separate reads and the write and is pipelined to minimize the critical path. Bit lines are decoded and precharged in the same phase, and the register array is accessed in the following phase by driving the wordline. The access time of the register file read is the critical path of the chip. It is measured to be under 14 nsec. For registers in the overlapped window, a special decoder shown in Figure 7a is used to map two different register addresses (one from the caller's window and the other from the callee's window) to one register [5].

In the pipelined execution of the instruction stream, data interdependencies among instructions in the pipeline may arise. In the SPUR CPU, these interdependencies are detected and resolved by the hardware internal forwarding. That is, the results from preceding instructions are forwarded to the following instructions by the hardware before being written back to the register file, as indicated by the arrows in Figure 5. In the case of a 4-stage pipeline like the SPUR CPU,

the data interdependencies may exist among 3 consecutive instructions since the write-back stage of the pipeline is delayed by two cycles after the execution stage. The result available from each instruction's execution stage, therefore, needs to be stored in temporary registers for two cycles and then forwarded to the following instructions. When both operands are registers, each register address is compared to the destination register address of the two preceding instructions. This may result in double internal forwarding, in that both operands are results of two preceding instructions and hence supplied from the temporary registers.

The hardware internal forwarding logic is in the critical path of the register file access, and it must be implemented without slowing down the cycle time. Like decoding and accessing the register array, it is also pipelined. Address comparisons are done in parallel with the decoding of the register file, and internal forwardings are made if necessary while the register file is accessed. Four address comparisons are necessary to detect all possible data dependencies. The address comparator must be fast to keep the cycle time short, and it must be compact to fit in the area between register decoders and temporary registers, as seen in Figure 3 (block IF). Bitwise comparison is done using a dynamic XOR, shown in Figure 7b, and then the outputs are fed into the domino circuit for an address match. Since this XOR does not require complementary inputs, routing and area required are significantly reduced. A special multiplexor, shown in Figure 7b, is used to minimize the signal delay through the internal forwarding logic that lies between the register file and the functional unit. If internal forwarding is necessary, the bus from the register file is disconnected by the transmission gate, and the bus to the functional unit is driven by the temporary register. The access time of register file reading (14 nsec) includes the delay through the internal forwarding logic.

C. The data path

The data path is divided into two parts: the upper data path for program counter logic and special registers, and the lower data path for general computations on tagged registers. Functional units in the lower data path include a byte-extractor, a byte-inserter, a simple shifter that shifts up

to three bits, and an ALU. The ALU provides XOR, OR, AND, ADD, and SUBTRACT operations and comparison for two 32-bit operands. The upper data path consists of a number of program counters to hold instruction addresses in the pipeline, an address incrementer and adder, and special registers such as window pointers and processor status words. All registers and counters are made of pseudo-static latches, such that each register is refreshed once per cycle. This is necessary because an indefinite pipeline stall is possible due to a long external cache miss.

In the SPUR CPU, compare-and-branch instructions are executed in only one cycle. A separate adder in the upper data path calculates the target addresses for all the compare-and-branch instructions while the ALU is in use for the comparison. Two different adder designs are employed. The 32-bit ALU uses four 8-bit carry lookahead adders implemented in domino logic, and evaluates the carry within 11 nsec. The 30-bit address adder is more compact because it uses a Manchester carry chain which has a carry propagation delay of 13.5 nsec.

The upper 8-bit slices of the lower data path are for tag-related operations. Operations on the tag and the data are logically independent, that is, no information moves between the two parts by carry propagation or any other implicit mechanism. For operations, the 6-bit tag type is checked in parallel with the data operation. If there is a tag mismatch and the tag trap enable bit is set in the user processor status word (UPSW), the CPU traps to the software. Generation tag checking (2 MSB) is done when a special store instruction (ST_40 R_{S1}, R_{S2}, Immediate) is executed. Generation tag exception may occur if the object (R_{S2}) with a higher (younger) generation number is stored into the object (R_{S1}) with a lower generation number [7]. The read_tag and write_tag instructions move a tag to and from the data portion of a register using the byte-extractor and the byte-inserter respectively, so that any arithmetic or logical operations may be performed on it.

To reduce the chip area and improve the circuit speed, the dynamic circuit technique called domino logic [11] is heavily used in the design. Potential charge sharing problems are prevented either by the use of abundant clock phases or by careful layout of the critical nodes. The SPUR CPU has 7 major busses to provide communications both externally and internally. Some of these

busses have high capacitive loadings, and hence precharging is used to improve the speed of data flow through the highly capacitive busses. The high capacitance bus is precharged to high before being used and discharged conditionally through a strong NMOS pull down network when used. This not only reduces the signal delay through the bus but also minimizes the chip area required for a strong, large driver. Some logic function may be included in the pull-down network as well, further saving the chip area. Critical paths of the data path, register file, and instruction cache are summarized in Table 2.

D. The control

Four-phase clocking and a uniform four-stage pipeline for all SPUR integer instructions make the control section of the CPU relatively simple. The SPUR CPU uses internal instructions to handle pipeline interrupts, rather than requiring complex sequences for those exceptions. These internal instructions are miss, trap_call, and read_pc to handle instruction cache miss and all kinds of traps. These instructions are executed in the same way other instructions are executed. The use of these instructions further simplies the control design.

The control can be divided into three parts: master control, trap logic, and the interface to the cache control/memory management unit (MMU/CC). The latter two are separated out from the master control to simplify the control design. Trap logic detects all unusual conditions during the pipelined execution of an instruction. All traps are taken during an instruction's third pipeline stage, and hence only one instruction can cause a trap in any cycle. The trap logic consists of pipelined modules, each of which operates at the corresponding stage of the instruction in the pipeline. The MMU/CC interface logic generates cache opcodes according to the current instruction and the status of the CPU. It also buffers signals to and from the MMU/CC.

The block diagram of the master control is shown in Figure 8. A centralized master control unit controls the processor sequencing and decodes the opcode into high level control signals. Local random logic blocks then decode the high level signals into low level signals using clocks. They also provide buffering of the low level signal according to the loading requirement. All

signals controlling the data path are individually optimized so as to have equal delays relative to the clock edges. The separation of master control and local decoding/buffering significantly reduces the amount of routing between two sides, particularly in CMOS design where complementary signals are required in controlling the data path.

Most of the control logic in the SPUR CPU is implemented in static PLAs. The largest PLA is the one that decodes the opcode, which has 69 product terms with 40 outputs. The propagation delay through this PLA is about 15 nsec, well below the required timing of two phases or 50 nsec. All PLA outputs are evaluated once per cycle and need to be held in registers until the next cycle. The routing between the PLA and the registers may consume substantial chip area since the PLA output pitch is so small compared to the pitch of the registers. Thus, the registers (pseudo-static latches) are integrated into the output section of the PLA by widening the PLA output pitch (16 lambda to 20 lambda). This results in an unusually large PLA, but the chip area required is much less than if the PLA and the registers were separated, and the timing requirement is still satisfied.

IV. Design, Verification, and Testing Methodology

Methodologies employed in the SPUR CPU design have been influenced by the following two themes of the SPUR project: (1) an overall system-wide rather than local optimization, and (2) designing a chip for a working system rather than an experimental prototype. Consequently, methodologies became very important since the chip being designed must meet all the functional requirements set for the system design as well as performance goals.

A. Design methodology

The design strategy incorporated both top-down and bottom-up approaches. The top-down flow was as follows: architecture definition, instruction set design, microarchitecture design, and a detailed functional/behavioral description of the hardware. The bottom-up flow was circuit design of basic components, layout of basic cells, assembly of major blocks using those cells, and global placement and interconnections. Both approaches were taken in parallel from the

beginning, in order to achieve the highest performance at given technology and system design goals. For instance, many microarchitecture decisions were made after the feasibility of a certain hardware resource was carefully considered. Division of design tasks followed the same hierarchical boundaries of design abstractions: architecture and instruction set design, microarchitecture design, and VLSI implementation. One- or two-person groups were formed to take the responsibilities of each design level. Close interaction among different groups was necessary to make clean interfaces among themselves and design specifications.

Most of the CAD tools used in designing the SPUR CPU chip were developed at Berkeley, except those for the behavioral level design. The detailed design started with describing the behavior of the chip and its interactions with other components within the system. The functional behavior was written in ISP′, a hardware description language, and simulated using the N.2 simulator [12]. The implementation of the hardware can be divided into two parts. Most parts of the control design were done using a set of CAD tools that automatically synthesizes the behavioral description of the combinational logic into the PLA [13],[14]. Other parts of the control logic (sequential) and data paths were designed manually but aided by another set of tools. These two paths are diagrammed in Figure 9. For the automated synthesis path, only those parts of the hardware description containing combinational logic can be synthesized. For the manual part, logic and circuit design were done first for each block and followed by layout. Layout was done using an interactive layout editor, Magic [15], with background design rule checking and hierarchical extraction. The extracted layout, which is a switch-level description of the chip, was simulated using *bdsim*, a switch-level logic simulator [13].

Timing analysis was done before the layout, to make early tradeoffs among many alternatives and after the layout, to perform an exact timing analysis with all parasitics correctly annotated. To estimate the critical paths of the chip more accurately, and thus to be able to determine the cycle time, a test chip containing a register file with internal forwarding was fabricated and tested [16]. The measured critical path (register file read) was below 18 nsec, and this encouraged us to set the cycle time goal at 100 nsec.

B. Verification methodology

The verification methodology was constructed following a bottom-up approach. As each individual module was designed, switch-level simulation was performed on the extracted layout to verify the design. A small set of hand-written test vectors was used for the simulation. Once individual modules were verified, they were connected and then simulated together until the integration reached the major blocks, the execution unit and the instruction unit. Test vectors up to this point were small and easy to generate by hand, since the test sequences required to verify operations on these units separately were relatively simple. After all major blocks were integrated, the verification effort was directed at both functional and switch levels.

Functional simulations are performed not only on each major component, to verify its internal functions but also on the external system level, to verify interactions among major chip sets. The diagnostics for the functional simulation were coded in SPUR instructions, and an instruction level simulator called *Barb* was used to debug the diagnostics. The diagnostics were intended to be stored in the start-up ROM on the processor board. The N.2 system provided simulated memories that could be used to model ROM or other types of memory. Therefore the diagnostics were assembled and loaded into the simulated memory. When the N.2 simulation was started, it was forced to go through a series of start-up sequences, making the CPU begin fetching instructions from the ROM containing the diagnostics. The diagnostics were then executed to completion or until failure. The same ROM image was used to program the EPROMs on the processor board, to be used for on-board testing of the chip.

Running extensive simulations on the hardware description verified many design ideas and functionalities, but it was still necessary to extract and simulate the layout of the entire chip. The extracted description is almost guaranteed to accurately model the real chip. However, developing the tests and examining the results for a complete switch-level simulation would be very difficult. To minimize the required work, the functional simulation should drive the switch-level simulation with automatically verifying that the two match at every clock cycle. Fortunately, the N.2 simulator provides a "tracing" capability that logs all changes to a specified set of signals into

a file. By tracing all inputs and outputs of an N.2 module, it is possible to obtain a set of switch-level test vectors automatically. These vectors along with expected results on output nodes are fed into the the switch-level simulation. The switch level simulator, *bdsim*, sets the input nodes according to the timing and vectors specified and verifies the output nodes with the expected results. Any unusual condition is recorded so as to be used in debugging.

A problem may arise because functional simulation and switch-level simulation may show different results under unusual states, such as unknown and initial states. For example, the functional simulator initializes all nodes to zero, while all nodes are set to unknowns initially in the switch-level simulation. When the chip is tested neither of these initial conditions is correct. To alleviate the problem, all internal states are initialized explicitly in the functional simulations. In the switch-level simulation, on the other hand, the detailed verifications are made after the initialization is done and all internal states are synchronized with those of functional simulation.

In order to have a working system rather than a prototype chip, all aspects of the design had to be verified, especially the interfaces to external chips. Table 3 summarizes the diagnostics vectors simulated in both functional and switch-level simulations.

C. Testing methodology

Several features were incorporated into the SPUR CPU chip to increase its testability. Passive scan registers are attached to all major busses to increase the observability. All signals put on these busses can be scanned out for examination. All major blocks are connected and communicate through these busses, so that the diagnostics capability is greatly improved. Many signals, like state bits of finite state machines in IU and the LSBs of the instruction address bus (busPC), are also routed out to pins to determine the exact status of the processor at any time. The CPU sends out an instruction every cycle to the FPU (via busI), and it also provides the observability of the instruction being executed, including internal instructions. The IU and the EU can be physically separated by setting certain *diagnostic pins*. Furthermore, some of the lower order bits of instruction address bus were routed out to pins. Using these features, instructions can

be delivered directly to the EU in case the instruction unit is not functional, by monitoring the instruction address (busPC<10:2>) available on pins.

The initial testing was done on a special board made for the SPUR CPU chip. The Tektronix DAS 9100 system is connected to the board and controlled from a SUN workstation. The test set-up is shown in Figure 10. The same vectors used in the switch-level simulations are converted into test vectors. For short-cycle testing, test vectors were downloaded to DAS and testing was performed. A special set-up was necessary for long-cycle testing, since the DAS can only hold up to 256 cycles of test vectors. Long vectors are divided into several parts to fit in the DAS capacity. The division was made at the instruction accessing memory (external cache), such that the CPU was deliberately made to stall on cache miss by controlling the MMU/CC interface pin (cache busy), while the next portion of the vector is being down loaded. All signals acquired during the testing are transferred back to the SUN workstation for a cycle-by-cycle verification with the expected result. Most of the CPU functionalities are tested using the initial test set-up. After the debugging is done, the CPU chip is put on a SPUR processor board to test interactions with other components on the board, especially with the MMU/CC.

D. Results

The first-pass silicon had a few bugs, including circuit design, layout, and timing errors, but it worked enough to be used for initial debugging of the processor board. The layout errors discovered were misplaced well and substrate contacts onto signals rather than power supply lines. These effectively shorted the signal to either the ground or the power line, resulting in a stuck-at type fault. Some of these errors were corrected by isolating the misplaced contacts from the power supply using the laser restructuring technique provided by the Information Science Institute (ISI). Either the first level or the second level metal can be disconnected by using a laser shot through the passivation layer. The second (topmost) level metal lines with width of 3 μm were cut successfully without affecting other structures nearby. Other problems found were timing errors and glitches on signals controlling the dynamic circuits. The glitch was caused by the

excessive ringing on clock lines. The long running clock lines (10 mm) can have parasitic inductance and capacitance large enough to cause a substantial ringing, which may trigger any hazardous glitch.

Several electrical-rule checks were performed to avoid repeating the same errors for the second pass. However, there was still another layout error discovered after the fabrication. A portion of metal wire was missing, leading to a disconnected signal. A focused ion beam (FIB) IC development system, provided by the Seiko instrument company was used to fix the problem. Two holes were drilled on separated wires through the passivation layer to reach metal lines, using an ion beam, and connected using FIB-CVD (chemical vapor deposition) metal film deposition between the two points. The revised and repaired chip is fully functional and is used in a working SPUR processor board successfully executing its own operating system (Sprite) as well as many applications including LISP programs. The nominal operating frequency of the chip on the processor board is 10 MHz, while the maximum operating frequency is 12.5 MHz (80 nsec cycle time),

V. Summary

The SPUR CPU is a single-chip RISC microprocessor designed for a multiprocessor workstation. It supports a a multilevel cache scheme including a prefetching on-chip instruction cache, a coprocessor interface, and support for the fast execution of LISP through a tagged 40-bit architecture. In order to build a working computer system based on the SPUR CPU chip, reliable and efficient methodologies were necessary throughout the design. The chip, fabricated in a 1.6 μm double metal CMOS process, works well in the multiprocessor system prototype, and it met both of the functional and performance goals set at the initial stage of the design. It runs at 10 MHz consistently for all programs and dissipates less than 0.8 W of power.

ACKNOWLEDGEMENTS

The authors wish to thank all members of the SPUR project who made it an exciting project to work on, and acknowledge the technical contributions of R. Duncombe, W. Koh, K. Lutz, and J. Mak. The chip has been fabricated at HP via MOSIS.

REFERENCES

[1] D.K.Jeong et al., "A VLSI Chip Set for a Multiprocessor Workstation - Part II: A Memory Management Unit and Cache Controller", this issue, companion paper.

[2] M.D. Hill et al., "Design Decisions in SPUR", *IEEE Computer*, vol.19, no. 10, pp. 8-24, Nov. 1986.

[3] D.K. Jeong et al., "Design of PLL-Based Clock Generation Circuits", *IEEE J. Solid-State Circuits*, vol. SC-22, no. 2, pp. 255-261, April 1987.

[4] G.S.Taylor et al.,"Evaluation of the SPUR Lisp Architecture", *Proc. 13th International Symposium on Computer Architecture*, Tokyo, Japan, June 1986.

[5] M.G.H. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", PhD Thesis, UC Berkeley, Oct. 1983.

[6] J.M. Pendleton et al, "A 32-bit Microprocessor for Smalltalk", *IEEE J. Solid-State Circuits*, vol. SC-21, no. 5, pp. 741-749, October, 1987

[7] D. Ungar, "Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm", *ACM Software Engineering Notes/SIGPLAN Notices Software Engineering Symposium on Practical Software Development Environments*, Pittsburg, April, 1984.

[8] J.R. Goodman, "Using Cache Memory to Reduce Processor Memory Traffic", *Proc. Tenth International Symposium on Computer Architecture*, Stockholm, Sweden, June 1983.

[9] M.D. Hill and A.J. Smith, "Experimental Evaluation of On-Chip Microprocessor Cache Memories", *Proc. Eleventh International Symposium on Computer Architecture*, Ann Arbor, MI, June 1984.

[10] R.W. Sherburne Jr, M.G.H. Katevenis, D.A. Patterson, and C.H. Sequin, "A 32b NMOS Microprocessor with a Large Register File", *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 682-689, October, 1984.

[11] R.H. Krambeck, C.M. Lee and H.S. Law, "High-Speed Compact Circuits with CMOS", *IEEE J. Solid-State Circuits*, vol. SC-17, pp. 614-619, June, 1982.

[12] *N.2 Simulator User's Manual*, ENDOT, Inc., Cleveland, OH, 1985.

[13] R.B. Segal, "BDSYN: Logic Description Translator, BDSIM: Switch-level Simulator", *Electronics Research Laboratory Memorandom*, No. M87/33, University of California, Berkeley, May 1987.

[14] OCT Tools Distribution 2.1, Electronics Research Laboratory, University of California, Berkeley, March 1988.

[15] J.K. Ousterhout et al., "The Magic VLSI Layout System", *IEEE Design & Test of Computers*, vol. 2, pp. 19-30, Feb. 1985.

[16] D. Lee "Data Path Design Considerations for a High Performance VLSI Multiprocessor", Technical Report No. UCB/CS Division 87/318, University of California, Berkeley, Nov. 1986.

Number of Transistors	115,214
Number of PLA's	13
Die Size	11.5mm x 11.5mm
Package	208-pin PGA (40 pins for power supply)
Process	Double-Metal 1.6μm N-Well CMOS
Operating Frequency	10MHz
Power Dissipation	0.8W at 10MHz with 5V Supply

Table 1. Chip Statistics

phase	operation	critical path (nsec)
phi1	Register file - read	14.0
phi2	Instruction Cache - fetch	12.0
phi3	ALU - 32b carry propagation	11.0
phi4	Address adder - 30b carry propagation	13.5

Table 2. Critical path timing

diagnostics	test vector length (cycles)
CPU functions	13,113 (24%)
MMU/CC interface	16,356 (29%)
FPU interface	1,543 (3%)
Lisp tags and traps	8,675 (16%)
Boot-up diagnostics	15,829 (28%)

Table 3. Diagnostics

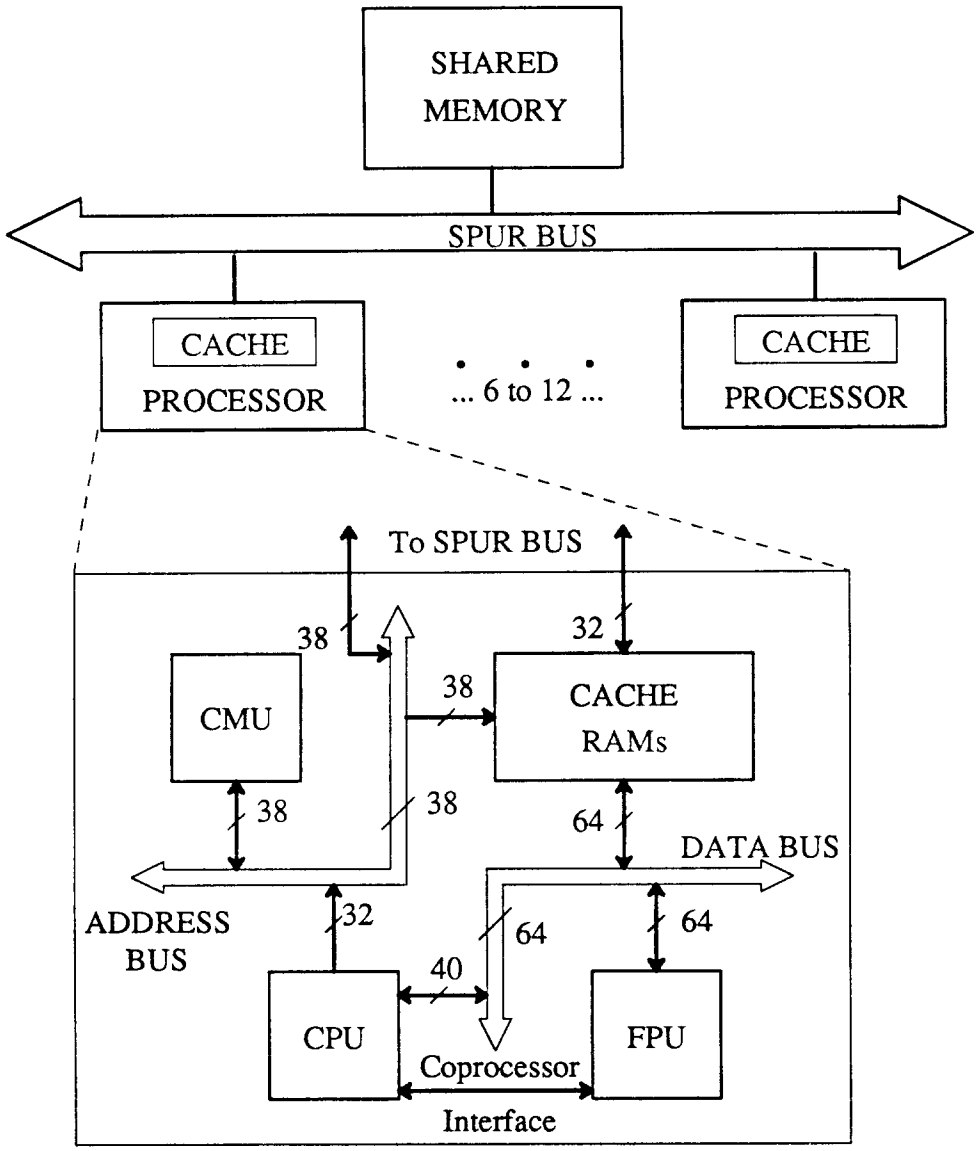

Figure 1. SPUR multiprocessor workstation

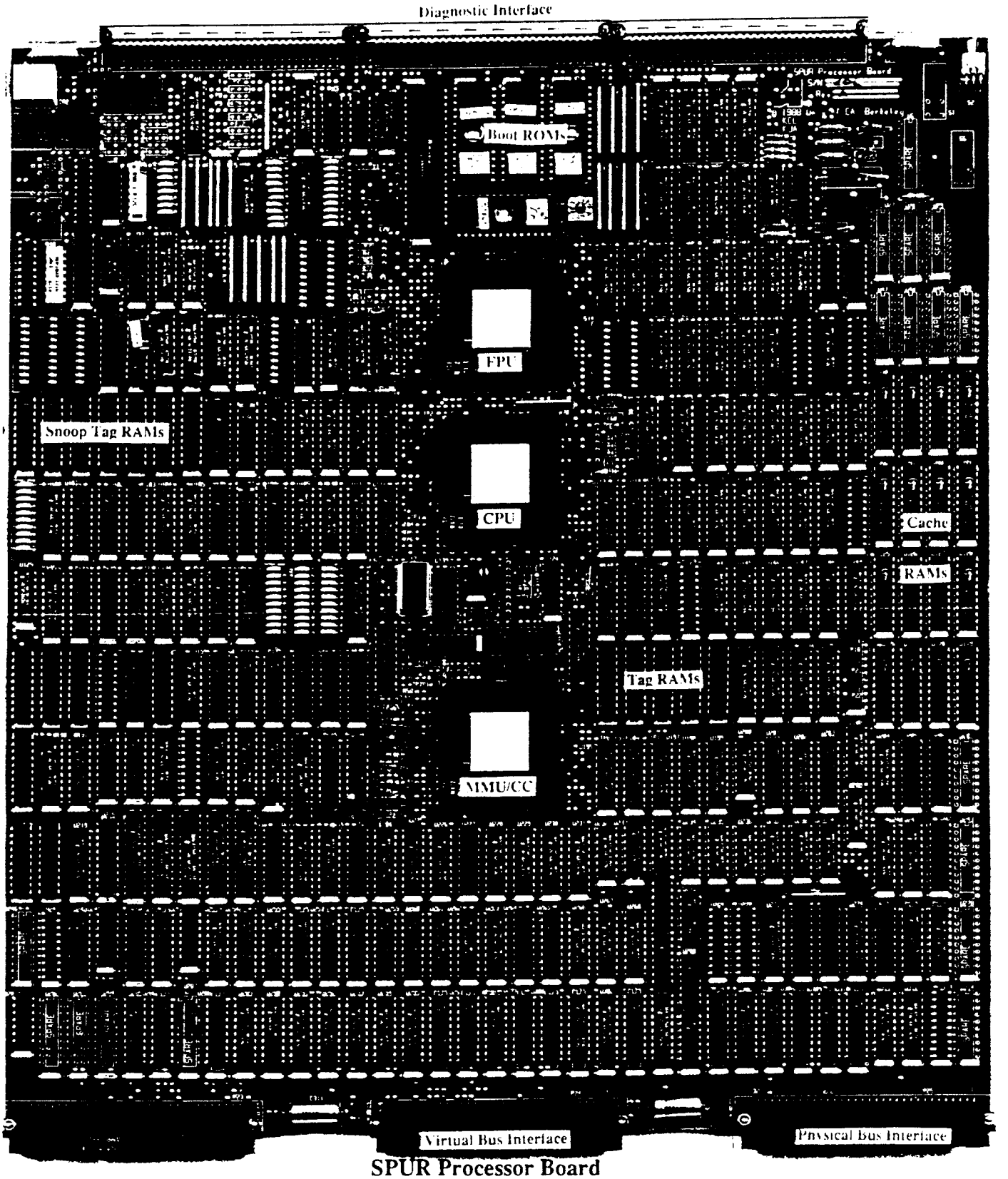

Figure 2. A SPUR processor board

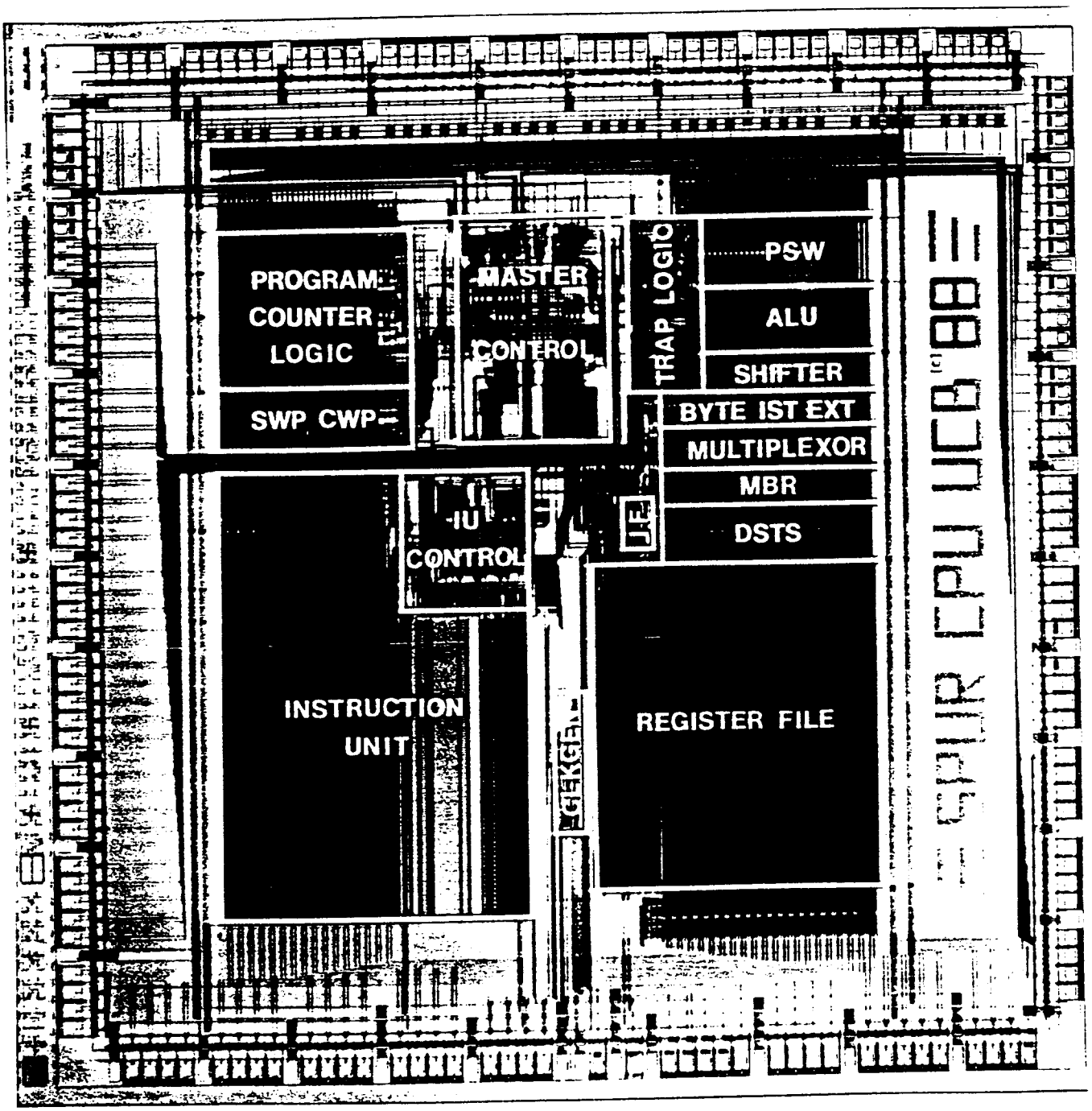

Figure 3. Chip microphotograph

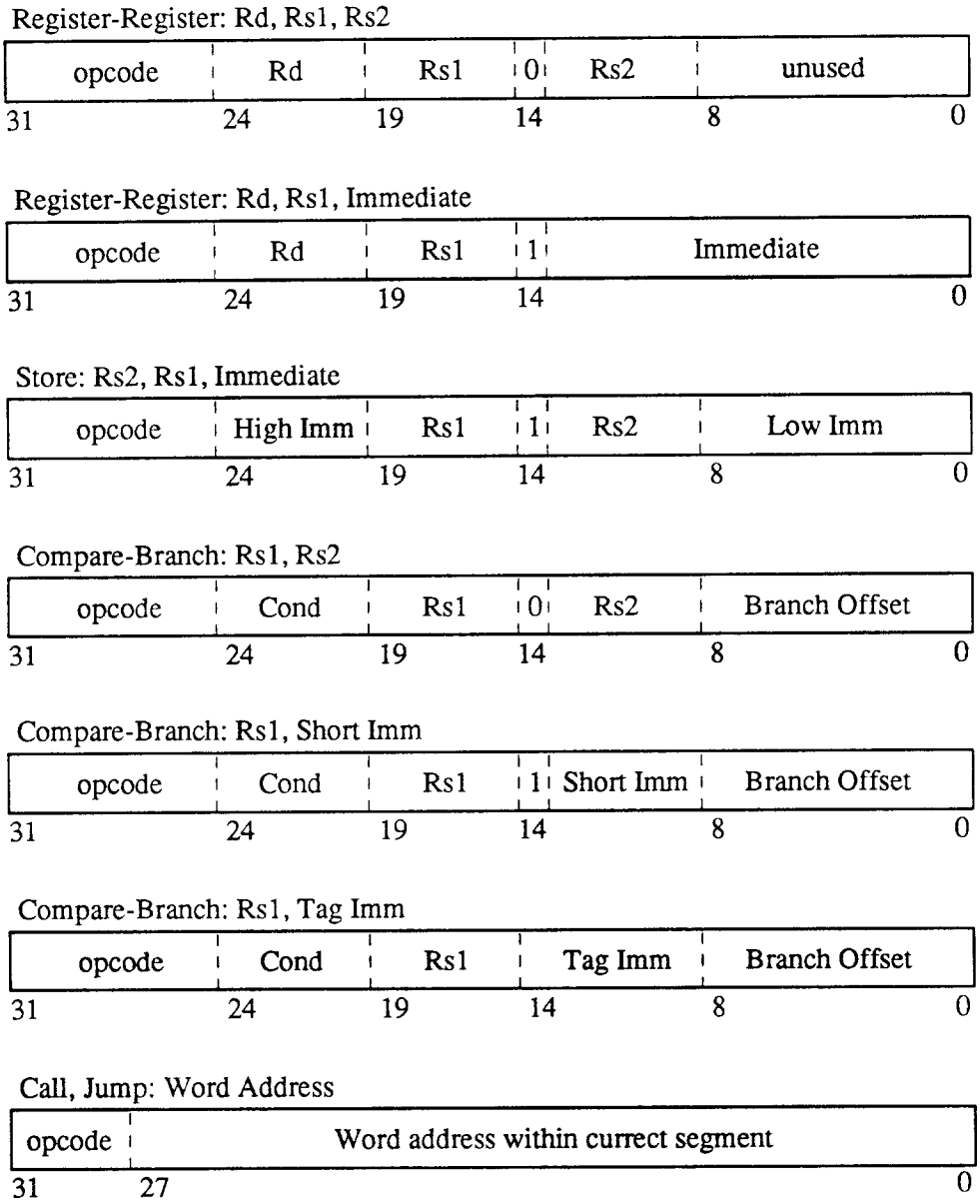

Figure 4. SPUR instruction formats

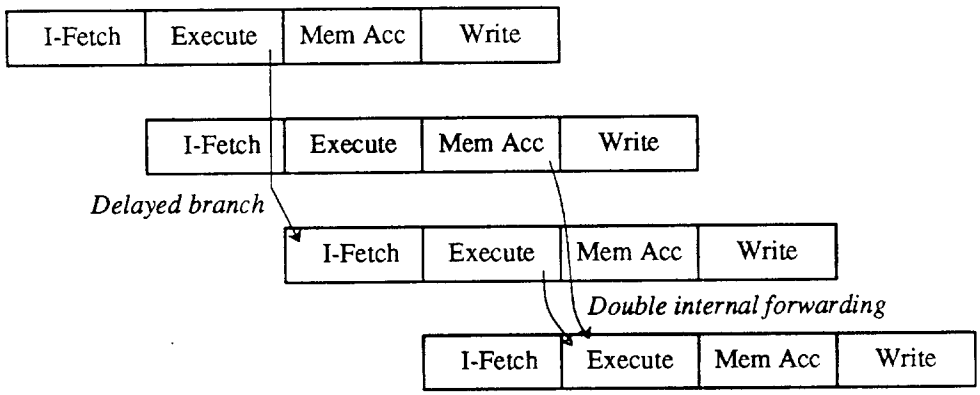

Figure 5. SPUR CPU pipeline

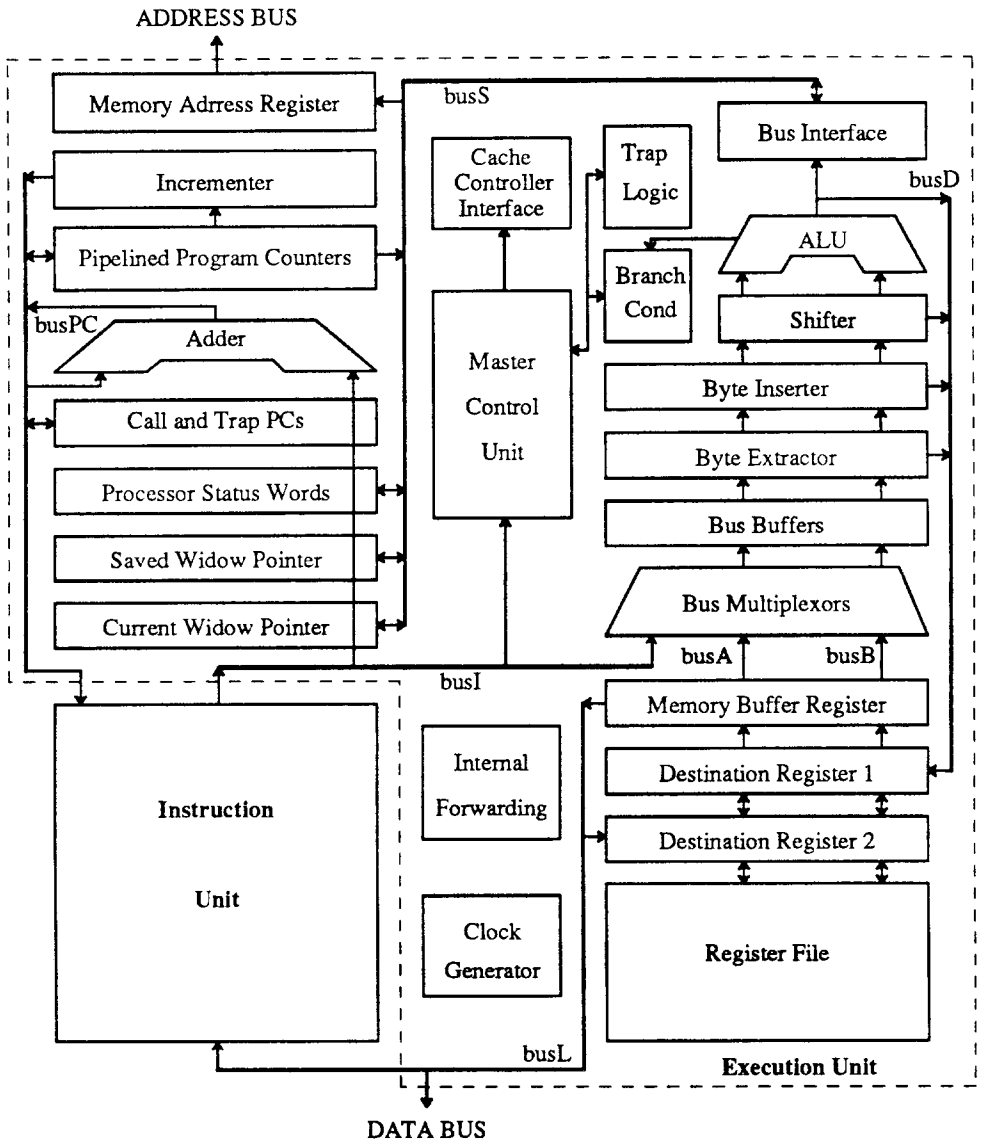

Figure 6. SPUR CPU block diagram

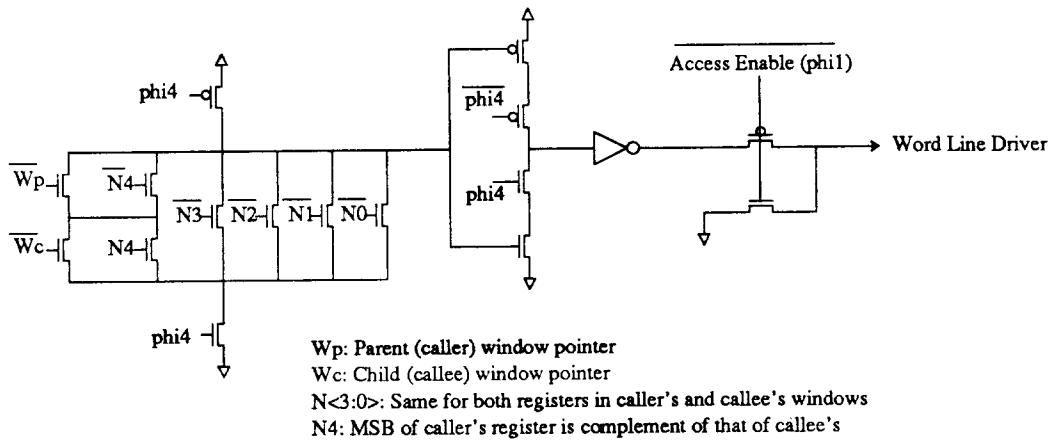

Figure 7a. Overlapped window register decoder

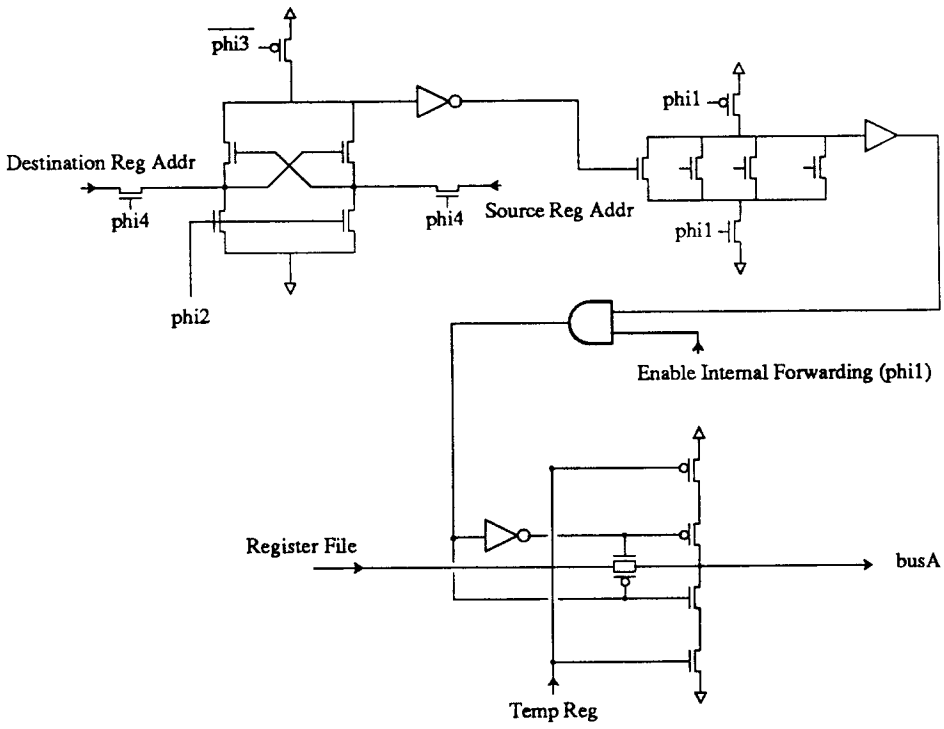

Figure 7b. Internal forwarding logic

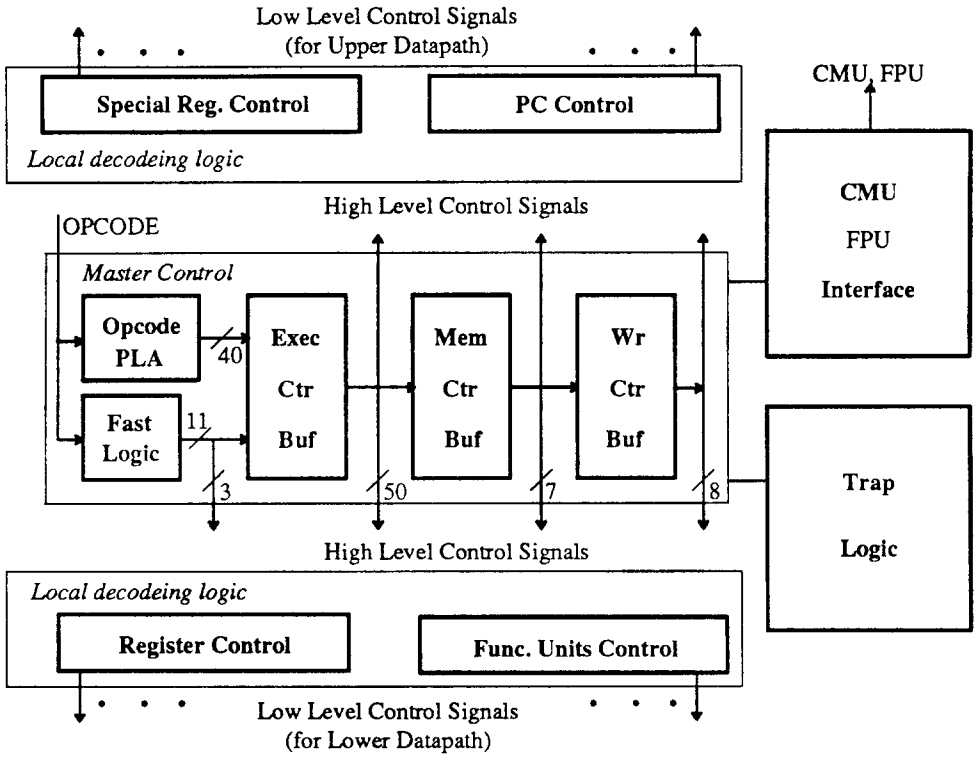

Figure 8. Block diagram of master control

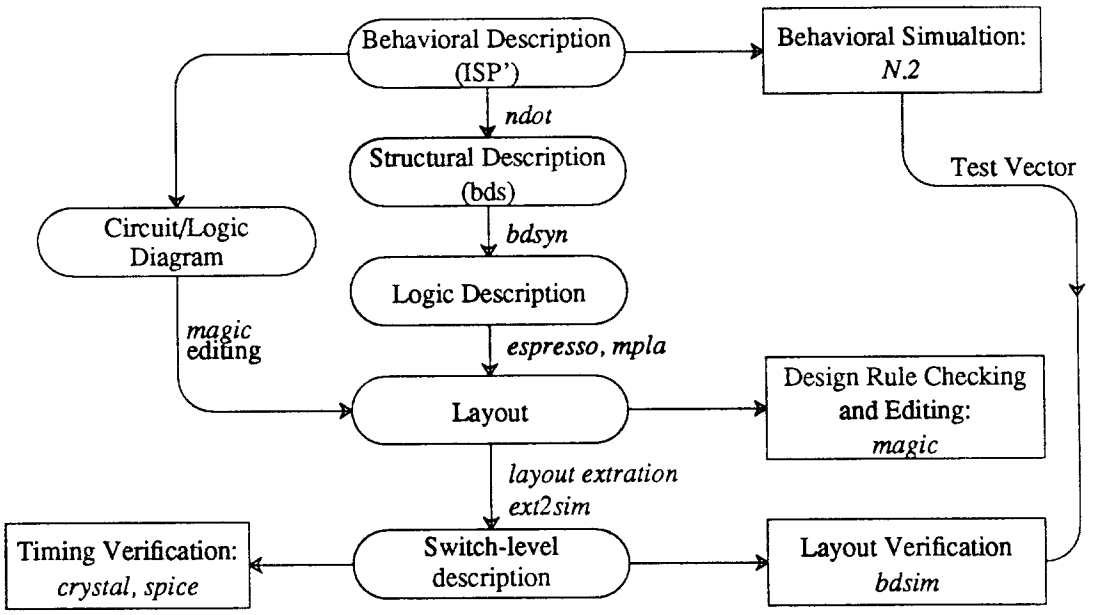

Figure 9. Design methodology

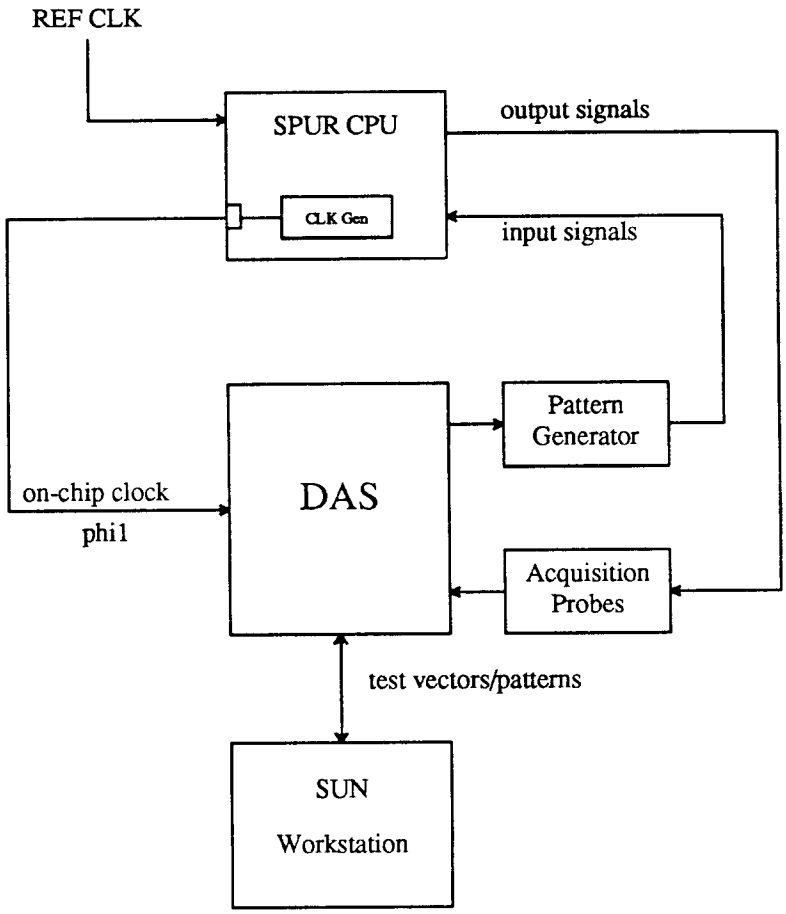

Figure 10. Chip test set-up

A VLSI Chip Set for a Multiprocessor Workstation - Part II: A Memory Management Unit and Cache Controller

Deog-Kyoon Jeong
David A. Wood
Garth A. Gibson
Susan J. Eggers
David A. Hodges
Randy H. Katz
David A. Patterson

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

This two-part paper describes two key components used in building a 40-70 MIPS multiprocessor workstation. In the first part [1], VLSI implementation of the central processing unit (CPU) chip, based on reduced instruction set computer (RISC) architecture and with support for LISP is described.

The second part discusses required hardware support for a shared memory multiprocessor, and describes a memory management and cache controller (MMU/CC). The MMU/CC implements an in-cache address translation mechanism that does not require the use of a translation look-aside buffer. Also, a snooping bus protocol is implemented to assure coherency among multiple caches distributed across the system.

Both chips are implemented in a 1.6 μm double-layer-metal CMOS technology, and are being used in a multiprocessor workstation (SPUR) successfully executing its own operating system called Sprite as well as many applications including LISP programs.

I. INTRODUCTION

Computer architecture evolves by responding to advances in the underlying implementation technology. Current CMOS VLSI technology makes it possible to integrate a very powerful processor on to a single chip. Computer architects respond with RISC (Reduced Instruction Set Computer) concepts as a result of careful study of trade-offs between VLSI technology and compiler technology [2]. RISC-based microprocessors have demonstrated an ability to provide more computing power at a given level of integration than conventional microprocessors [3-4]. Next logical step is multiprocessors composed of RISC processing elements. We have designed and built such a multiprocessor called SPUR [5]. SPUR (Symbolic Processing Using RISC's) is a bus-oriented shared memory multiprocessor developed at U.C. Berkeley to explore RISC applicability to symbolic programming languages such as LISP, and parallel processing with shared memory. The features of SPUR in a symbolic processing area are discussed in the companion paper and this paper focuses on the hardware for multiprocessing.

Shared memory multiprocessors use private write-back caches to reduce the memory bandwidth through a common bus as well as reducing the average memory access time. Since each processor has its own cache blocks, multiple copies of the same address may reside in different caches. Thus, a mechanism for assuring cache consistency among multiple caches is required so that a consistent view of memory is maintained. It is a challenging problem to successfully design, verify and implement cache coherency protocols because of their very complex operations involving many possible state transitions. This paper describes the MMU/CC that implements such a protocol. The innovative functions of the MMU/CC are a virtual memory management mechanism via in-cache address translation [6] and a write-invalidate cache

This work was sponsored by DARPA under contract order 482427-25840, California MICRO, Texas Instruments, National Semiconductor, Cypress Semiconductor, Tektronix, and Hewlett Packard.

coherency protocol [7]. The MMU/CC controls access to the cache and generates all control signals for the board and backplane. An interval timer, an interrupt controller and performance counters are also integrated onto the MMU/CC.

The MMU/CC has been fabricated in a double metal 1.6μm N-well CMOS process at Hewlett Packard through MOSIS. The chip integrates 68,400 transistors on a die measuring 11.5 × 11.5 mm^2 and consumes 0.7 W. Figure 2 shows the chip microphotograph. The chip statistics are summarized in Table 1.

In the next section, we present a functional description of the MMU/CC. In Section III, implementation details and circuit techniques are described. Section IV contains a specific considerations in the design and implementation of the MMU/CC based on the design methodology described in the companion paper. Finally, status and conclusions are given in Section V.

II. FUNCTIONAL DESCRIPTION

The MMU/CC combines two control units - a processor cache controller (PCC) and a "snooping" bus controller (SBC). The PCC handles memory references for the CPU, while the SBC interacts with the backplane bus. The PCC and SBC run independently unless an event that requires the other's attention occurs.

A. The Processor Cache Controller

The PCC's memory management scheme is based on an in-cache address translation mechanism [6]. Virtual, rather than physical, addresses are used for the cache index and tag. The advantage of the virtually addressed and virtually tagged caches is that address translation is needed only on cache misses. Because the cache is quite large, misses occur infrequently, and a high speed translation mechanism is not needed. The virtual to physical address translation map is located in the virtual address space and the data cache serves as a translation look-aside buffer (TLB), reducing the complexity of translation and eliminating the need for a separate unit for this

function.

On each CPU reference, the PCC first transforms the 32-bit virtual processor address into a 38-bit global virtual address. It maps the most significant two bits of the virtual address into an 8 bit segment number by selecting one of 4 global segment number registers in the datapath. The mapping is done in parallel with the cache tag and data RAM access, since the high order bits are part of the tag and this simple map is faster than the cache tag store access. If the address tag matches the cache tag, the data is returned and the MMU/CC takes no further action. On a cache miss, the PCC generates the virtual address of the page table entry (PTE) using a page table base register and a special shifter. It uses the PTE virtual address to access the cache in the next cycle. If this access finds the PTE, the physical address of the data is extracted and a main memory reference is made to fetch the desired cache block. On a PTE miss, a third cache reference is needed to access a root page table entry (RPTE) with the assistance of a root page table base register and another shifter. The desired cache block is fetched in a recursive manner. This recursive process ends if the third cache reference misses in the cache. Instead of going to a deeper level, it uses a root page table entry map register that contains the base address of the root page table entry in the physical address space. Because of the size of the cache and locality in accesses, PTE misses and especially RPTE misses are extremely infrequent [6]. The translation mechanism is shown in Figure 3 and Figure 4.

B. The Snooping Bus Controller

The controller provides special hardware support for maintaining coherency among private caches. When a memory block is shared by more than two caches, a local write into one of the shared cache blocks should be reflected in the other cache's corresponding block, so that the system maintains a consistent, single-level view of memory. The Snooping Bus Controller (SBC) implements a distributed cache coherency mechanism, called Berkeley Ownership [7]. The SBC not only initiates its own bus transactions on behalf of the processor, but also "snoops" on other

bus activities to detect when one of the local cache blocks is involved.

The protocol works as follows. A cache block is in one of four coherency states: *Invalid*, *UnOwned*, *OwnShared* and *OwnPrivate*. Also, four kinds of bus transactions are possible: *ReadShared*, *ReadForOwnership*, *Write*, and *WriteForInvalidation*. Owning a cache block means that the owning SBC has the responsibility to provide an up-to-date copy on a read request from the bus and to write the cache block to memory if it needs to replace the block in the cache. There is at most one SBC that owns a memory block. Main memory is the implicit owner of the block if it is not cached by any processor. On a write to a valid, non-*OwnPrivate* cache block, the PCC stalls the processor while the SBC initiates a *WriteForInvalidation* to invalidate corresponding cache blocks in other caches. The state of the local cache block becomes *OwnPrivate*. Until another SBC takes ownership, subsequent writes can be made locally without informing others because it is a unique copy. On receiving a *ReadShared* bus request, the state of the OwnPrivate cache block is changed to *OwnShared*. In the *OwnShared* state, the next local write should accompany *WriteForInvalidation* because there are other copies of the same block. A special processor request, *ReadPrivate*, is included for improving performance under software direction. Ownership can be obtained immediately when reading a non-shared block, instead of waiting until a processor write operation. In this way, an unnecessary bus transaction, *WriteForInvalidation* can be avoided on "private" data. Figure 5 shows a state transition diagram of the protocol.

We have extended a standard microcomputer bus, Texas Instrument's *NuBus*, to incorporate the Berkeley Ownership protocol [8]. We wanted to use existing commercial memory and I/O boards. Since the devices did not participate in the Berkeley Ownership protocol, we used a separate set of backplane lines for inter-cache transactions.

C. Asynchronous Interface

While the SBC derives its clock from the SpurBus (10 MHz fixed frequency), the PCC shares the processor clock which is asynchronous with the bus. The frequency of the processor

clock can, therefore, be set according to its implementation technology, regardless of the bus clock and other processors. We chose to do this to provide more flexibility during testing and integration with memory and I/O devices. An asynchronous interface handles a request/acknowledgement handshake for communication between the PCC and SBC.

Other system functions are integrated onto the MMU/CC such as performance counters, interval timers, interrupt controller. Performance counters monitor various system activities to measure performance metrics. The purpose of including these counters is to aid in the performance analysis of the working multiprocessor without perturbing the system. They count 32 kinds of coherency-related and cache access events in user and/or kernel mode such as *ReadForOwnership* bus operations, cycles spent by the PCC waiting for a bus transfer to finish, instruction fetches, and so on.

III. Implementation

Figure 6 shows a detailed block diagram of the MMU/CC internals. The chip consists of the PCC, the SBC, an asynchronous interface and other system functions. A complete specification for the MMU/CC is in [9].

A sequencer implementing the PCC consists of a programmable logic array (PLA), a stack and a decoder that sends low level control signals both on-chip and off-chip. It is configured as a push-down automaton rather than as a finite state machine to efficiently implement a recursive algorithm used in address translation. State information is stored in the 4 entry stack and can be pushed, popped, replaced, or flushed under the control of the sequencer. Such a machine structure is also convenient to handle SBC request servicing. When the SBC requests the PCC's attention, the PCC is able to save its current state while executing the SBC's request.

The PCC's sequencer PLA has 41 inputs, 36 outputs and 207 product terms. A small sense

amplifier that fits into the pitch of the array has been designed for fast signal detection in large PLAs. It also limits the voltage swing in the highly capacitive nodes in the AND plane and the OR plane. Figure 7 contains a circuit diagram and its transfer characteristics. The range of the voltage swing is set to approximately 1 V, which is determined by a reference voltage generator. The simple reference voltage generator is composed of a diode-connected transistor and a set of the pull-up and pull-down transistor with the same size and orientation as the ones in the array. It assures insensitivity to process and power line variations. Assuming only one array transistor is selected in the data line, the voltage swing is limited to ΔV_{GS} of M2. An array transistor at logic threshold sinks the same amount of current as the pull-down in the reference generator if ΔV_{GS} of the cascode transistor, M1, is half as much as that of the diode-connected transistor, M2, in the reference generator. Thus, the logic threshold voltage of the highly capacitive node is placed in the middle of the voltage swing, regardless of the variations, by making the width of M1 2.5 times larger than the width of M2. The worst-case delay (fully loaded AND/OR plane) of a 50 input, 50 output, 200 product PLA was simulated to be 31ns with a 100μA pull-up current. An equivalent PLA without the sense amplifiers would have had a 55ns delay. However, real PLAs with a sparse AND/OR plane would have significantly less delay because of reduced capacitance in the select and data lines. Actual delay of the sequencer PLA is estimated to be 18 ns.

A comparison with other PLAs is shown in Figure 8. PLAs with polysilicon select lines have the longest delay because the RC delay increases quadratically with the size of the PLA. Without low ohmic silicided polysilicon lines, first level metal could be used as the select line and second level metal as the data line. Since the pitch of the metal lines and their contact sizes are larger than that of the polysilicon lines, the area of a PLA that uses metal for both lines is approximately 2 times larger than the PLAs with polysilicon or silicided polysilicon lines. Our sense amplifiers are more efficient as the size of the PLA increases. Overhead delays due to inverter stages amortize, and the total delay time reduces asymptotically to $[\frac{V_{SWING}}{V_{DD}}]$ times the

delay of the PLA without sense amplifiers.

The address translation datapath includes special purpose registers and shifters. It also shares its buses with other system utility functions: performance counters, interval timers, and interrupt registers. All registers and buses are implemented with fully static or pseudo-static logic. Although fully static circuits take more area, they are relatively immune to noise and tend to generate less noise.

The SBC consists of 7 PLAs, 19 OR gates, and several logic blocks with random gates and latches. Each PLA is a controller partitioned to specific functional operations, running in parallel with other PLAs. A PLA named *master* generates almost all backplane control signals when it acts as a bus master, while a *slave* PLA does the similar operation responding to the bus as a bus slave. A smaller controller, *nubus*, receives interrupts and notifies the *master* about SpurBus availability. A *virmach* PLA manages data transmissions and receptions on the inter-cache backplane lines. It may be triggered either by the *master* to override memory's copy of the locally requested block or by the *slave* to transmit the block requested by the inter-cache backplane. A *physrec* PLA handles the transfers from the memory to the cache and it may be requested by the *master* to release the cache RAMs in favor of the *virmach*. A *reset* PLA continuously monitors bus activities to check for reset conditions and potentially override all other PLAs. The total numbers of inputs, outputs and product terms of the SBC PLAs are 100, 123, and 236, respectively, with the largest PLA having 25 inputs, 39 outputs, and 76 product terms. Of the total SBC area, 65% is consumed by routing. There are 209 nets among major blocks in total. Net length distribution is shown in Figure 9, not including clock lines and scan path related routing.

Since the MMU/CC must generate board-level signals as well as internal datapath control signals, stringent timing relationships are required. Clock skew must be minimized for high speed synchronous communication among the PCC, CPU and FPU. Multi-phase clocks are needed to provide many different timings for enables, chip selects, and address/data drivers. Two

charge pump Phase-Locked Loop's (PLL's) with tapped delay lines [10] provide the flexibility needed to generate multiple clock phases, in addition to maintaining accurate phase relationships with clock phases on other chips and the backplane. Figure 10 shows the 16 internal clock phases. Since the internal frequency follows the external reference clock, all phases stretch or contract proportionally. It is more forgiving to critical paths since any phase including non-overlap time can be stretched by slowing down the external clock. With a 10 MHz external clock, the minimum timing quantization of any internal clock phase is 5 ns. Conventional clocking would have required a 200 MHz external clock to obtain 5 ns timing resolution. Special techniques would have been required to distribute the external clock in a printed circuit board and design an on-chip clock generator operating at such a high frequency.

As with all asynchronous interfaces, metastable states can cause system failures. To reduce the probability of these failures, we employ a two-prong design strategy: first, maximize the bandwidth of internal core amplifiers in each synchronizer, and second, allow the synchronizer half a cycle to settle. Because most interface transactions occur on infrequent cache misses, the extra latency for synchronization does not significantly degrade system performance. The core of the synchronizer is an RS flip-flop composed of two NAND gates that has been carefully laid out to reduce parasitic capacitance. Simulation shows that the characteristic time constant of the synchronizer is 0.24ns. Initial condition of the input voltage difference in the synchronizer core that cause metastability to persist longer than 20 ns is 3.2×10^{-36} V [11]. Assuming the initial input voltage difference caused by asynchronous inputs is uniformly distributed (conservative assumption), the probability of metastability persisting longer than 20 ns is 6.4×10^{-37}. When synchronizations happen at a 10 MHz rate, the system's Mean Time Between Failure (MTBF) due to synchronization error is more than 10^{21} years.

Two channels are needed for bidirectional communication between the PCC and SBC. One channel is responsible for delivering to the SBC the PCC's requests to fetch or write data to/from

the backplane on cache misses. Acknowledgements must also be returned from the SBC to inform the status of the current transactions - whether they have been finished successfully or resulted in errors. The other channel running in the opposite direction is mostly involved in snooping operations. The requests from the SBC are initiated when the SBC detects backplane transactions that require the PCC to relinquish the cache RAMs so that the SBC can invalidate, update or transmit a cache block. A logic diagram of one of the two asynchronous interface channels is shown Figure 11. Instead of using a conventional 4 cycle handshake mechanism, a variant of 2 cycle handshake mechanism is used. Interface logic allows a request line to be asserted for only one cycle to log the request to the receiver, without requiring the sender to hold the request line all the way until an acknowledgement arrives. Similarly, a one-cycle acknowledgement informs the sender of the completion of the transaction. This mechanism in the interface reduces the complexity of implementing a handshake protocol in the PCC and SBC, as well as retaining the speed advantage of a 2 cycle handshake. Speed independent operation is achieved as long as the cycle time of each side does not exceed the pulse width of the edge detector output which is approximately 10 ns. Since all data (ReqCode and AckCode) arrives at the destination at the same time or before a request/acknowledgement is asserted, there is no need for synchronization for data. About a half cycle is allowed for synchronizing request/acknowledgement signals.

IV. Design Methodologies and CAD Tools

For the design of a VLSI chip with as much functionality as the MMU/CC, good computer-aided design tools and adequate design methodologies are essential. Our design methodologies include design verification at the behavioral level, layout generation, layout verification, and a test suite. General description of the SPUR VLSI design methodology that is common to all SPUR chips is included in Section IV of the companion paper [1]. In this section, only the MMU/CC-specific extensions to the general picture is described.

The entire SPUR system has been described in ISP', and system level design verification has been done using a set of diagnostics in behavioral level. Although the MMU/CC can be simulated as part of the whole system, it is very difficult and time-consuming to write diagnostics to test all the cases for the MMU/CC. MMU/CC events are not single cycle and much state information is concealed in protocol - so the number of possible state configurations is astronomical. A random tester was developed to verify the memory system including the MMU/CC in a multiprocessor configuration [12]. A *stub* module that simulates the CPU's memory reference behavior with an accurate interface between the CPU and the MMU/CC replaces the CPU to speed up verification time. The stub CPU generates memory references by randomly selecting from a set of predefined scripts. The scripts are composed of two parts, an action part that generates references to cause state changes, and a check part that checks if the correct state change is made. The actions and checks are executed at different times with other actions or checks intervening, causing complex cases to be generated. For example, one of the scripts includes an action part that writes a word to a memory address, and a check part that reads a word from the same address. If they do not match, an error will be signaled. Although random testing takes significant amount of computer time and memory space, a significant amount of human effort devising test vectors can be saved. The total number of simulation cycles was between 50-100 million cycles, and the simulator ran 1000 cycles per hour in a SUN-3 workstation. The random tester uncovered more than half of the functional bugs found during the simulation. The random tester alone is not powerful enough - it does not stop simulation exactly where the fault occurs. Rather, it stops later when the fault is detected. The monitor module is a passive "watcher" that stops simulation whenever the SpurBus or cache coherency protocol is broken. There is also a daemon module that generates *NuBus* I/O transactions that Spur Boards must cope with, but do not generate. These three together form the "random tester system."

The major part of the MMU/CC is composed of controllers with widely varying sizes.

Instead of merging all the controllers together and implement them with a standard cell approach, we decided to implement them with separate PLAs and connect them with global routing. By doing so, a behavioral description and the corresponding layout match closely and as a result a minor change in the controller description does not result in major layout revision. Only automatic regeneration of the PLA layout is involved without any change in the routing. Also, since the terminal names are preserved, it is easier to verify the layout.

For testability, "passive" scan paths were included that snapshot internal states and shift the result out under external control. Although they do not provide the ability to introduce arbitrary states, they are useful for debugging errors.

V. STATUS AND CONCLUSIONS

The first version of the MMU/CC was sent out for fabrication in November 1987, and first silicon was received in February 1988. Omitting wafer probing, all chips were packaged and tested on a printed circuit board specially designed to connect to the Tektronix Digital Acquisition System (DAS). After downloading test vectors from a SUN workstation, the DAS exercised a chip and acquired result vectors. Result vectors were compared against the expected results using the SPUR specific tool, *ccdas*. Simple, short test vectors were used at this stage of testing. Although some chips passed all functional testing, we have experienced occasional errors. The errors were traced using scan paths and we discovered that some stack entries occasionally changed from 0 to 1 due to floating wells. In our methodology, instead of drawing wells explicitly, we relied on the *magic* layout editor for generating wells automatically. A few PMOS transistors were more than 12 λ's away from well contacts, so their wells were not properly biased. An ad hoc electrical rule checker was developed by changing the technology file of *magic*, and used for the next version of the chip. A second version arrived in September, 1988 and is fully functional. The SPUR CPU, MMU/CC and processor board now run the Sprite

operating system at the intended clock frequency, 10 MHz. A three processor system is reliably running parallel processes. The working prototype of SPUR MMU/CC demonstrates the manageability of complexity in implementing both address translation mechanism and cache coherency in a full custom VLSI chip.

ACKNOWLEDGEMENT

The authors would like to thank all the SPUR members for technical discussions and support, especially, Ken Lutz for providing an excellent test environment, Walter Beach for datapath design and layout, Dr. G. Luicki and S. Lu at MOSIS and R. Duncombe at HP for useful discussions and fabrication support.

REFERENCES

[1] Lee, D., et al., "A VLSI Chip Set for a Multiprocessor Workstation - Part I. A RISC Microprocessor with Coprocessor Interface and Support for Symbolic Processing," this issue, companion paper.

[2] Katevenis, M., et al., "Reduced Instruction Set Computer Architectures for VLSI," Ph.D. Thesis, U.C. Berkeley, Oct. 1983.

[3] Moussouris, J., et al., "A CMOS RISC processor with integrated system functions," *24th Annual IEEE Computer Conference (COMPCON '86)*, March 1986.

[4] Garner, R., et al., "The Scalable Architecture (SPARC)," *26th Annual IEEE Computer Conference (COMPCON '88)*, March 1988.

[5] Hill, M. D. et al., "Design Decisions in SPUR," *IEEE Computer*, vol. 19, no. 10, pp. 8-24, Nov. 1986.

[6] Wood, D. A. et al., "An In-cache Address Translation Mechanism," Proceedings 13th Annual Symposium on Computer Architecture, Tokyo, Japan, pp. 358-365, June 1986.

[7] Katz, R. H. et al., "Implementing a Cache Coherency Protocol," Proceedings 12th Annual Symposium on Computer Architecture, Boston, MA, pp. 276-283, June 1985.

[8] Gibson, G., "SpurBus Specification," *Proc. of CS292i: Implementation of VLSI Systems*, R.H. Katz (Editor), University of California, Berkeley, Sept. 1985.

[9] Wood, D., et al., "SPUR Memory System Architecture," Computer Science Division Report No. UCB/CSD 87/394, University of California, Berkeley, Dec. 1987.

[10] Jeong, D.-K. et al., "Design of PLL-Based Clock Generation Circuits," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 2, pp. 255-261, April 1987.

[11] Pechoucek, M., "Anomalous Response Times of Input Synchronizers," *IEEE Trans. Comput.*, vol. C-25, no. 2, Feb. 1976.

[12] Wood, D. A., et al., "Verifying a Multiprocessor Cache Controller Using Random Case Generation," Computer Science Division Report No. UCB/CSD 89/490, University of California, Berkeley, Jan. 1989.

TABLE 1 - Chip Summary.

Number of Transistors	68,395
Number of circuit nodes	30,285
Total gate capacitance	1280 pF
Total wire/junction capacitance	2630 pF
Number of PLAs	19
Total number of PLA inputs	262
Total number of PLA outputs	277
Total number of PLA product terms	707
Die Size	11.5mm x 11.5mm
Package	208-pin PGA
Power Dissipation	0.7W @ 5V, 10MHz
Process	Double-metal 1.6um CMOS

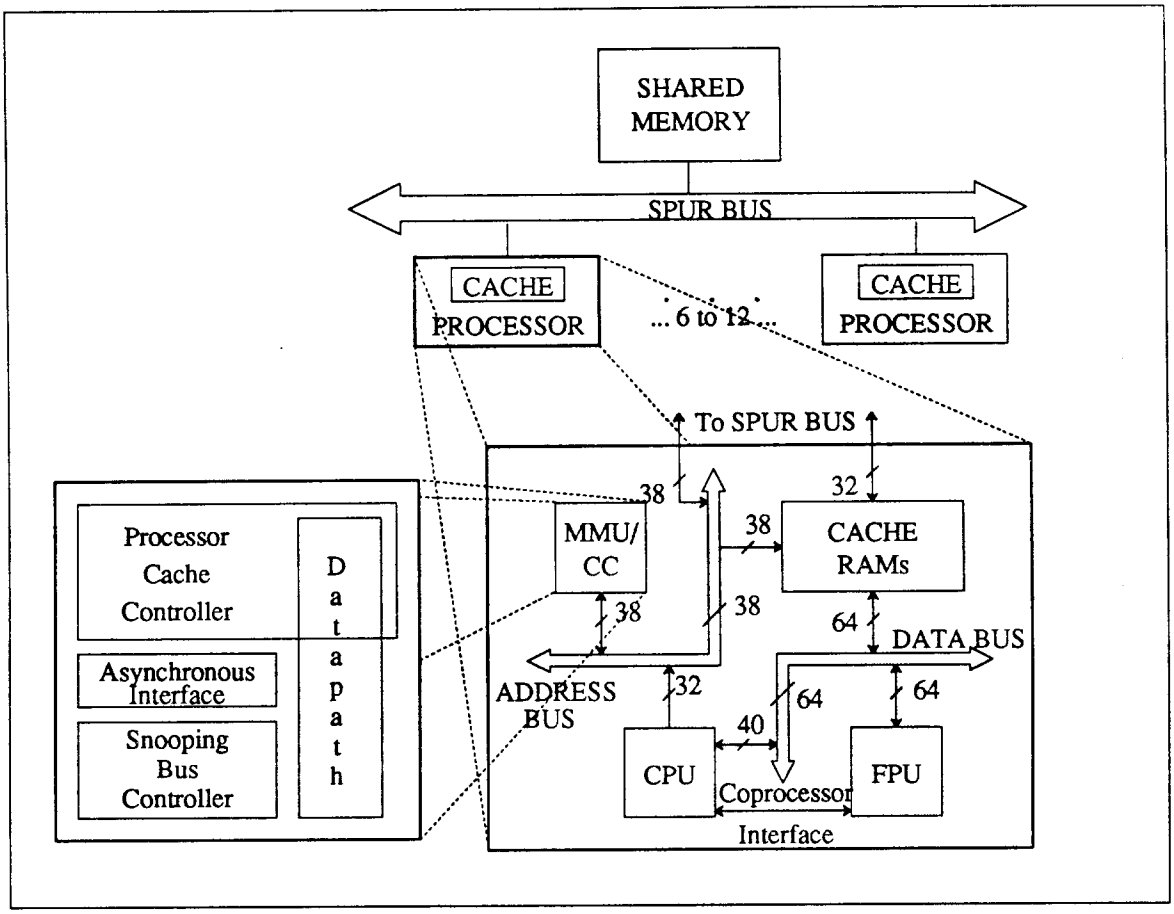

Figure 1 SPUR block diagrams: system, processor board and MMU/CC block diagram.

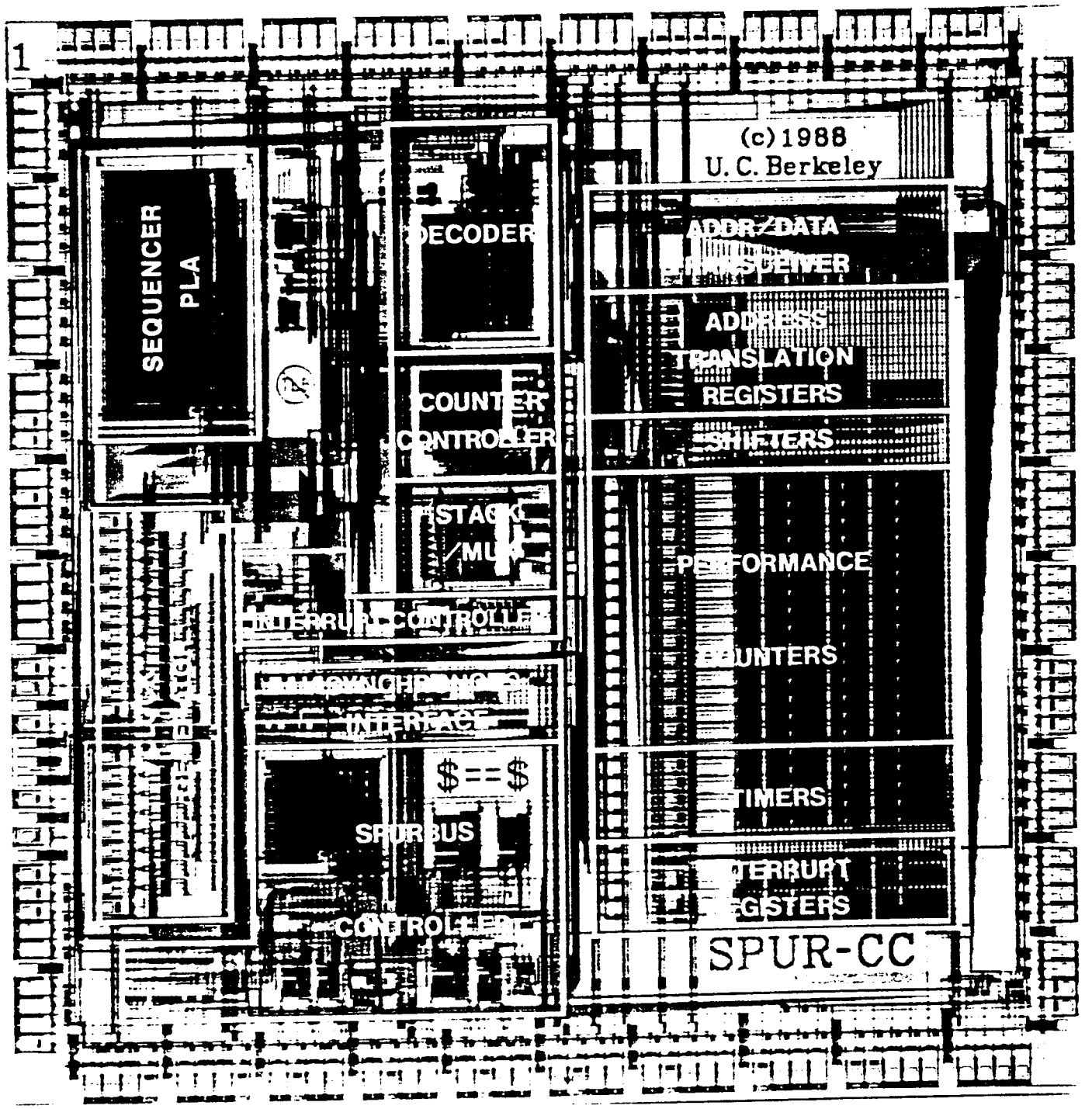

Figure 2 Chip microphotograph.

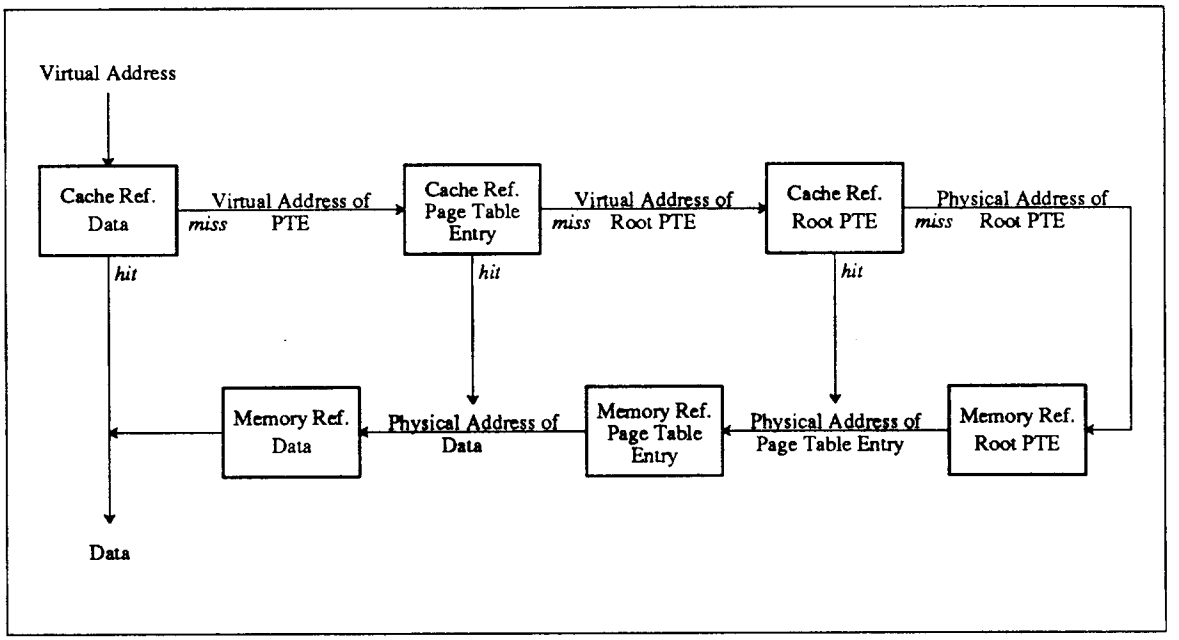

Figure 3 In-cache address translation mechanism - cache/memory access flow.

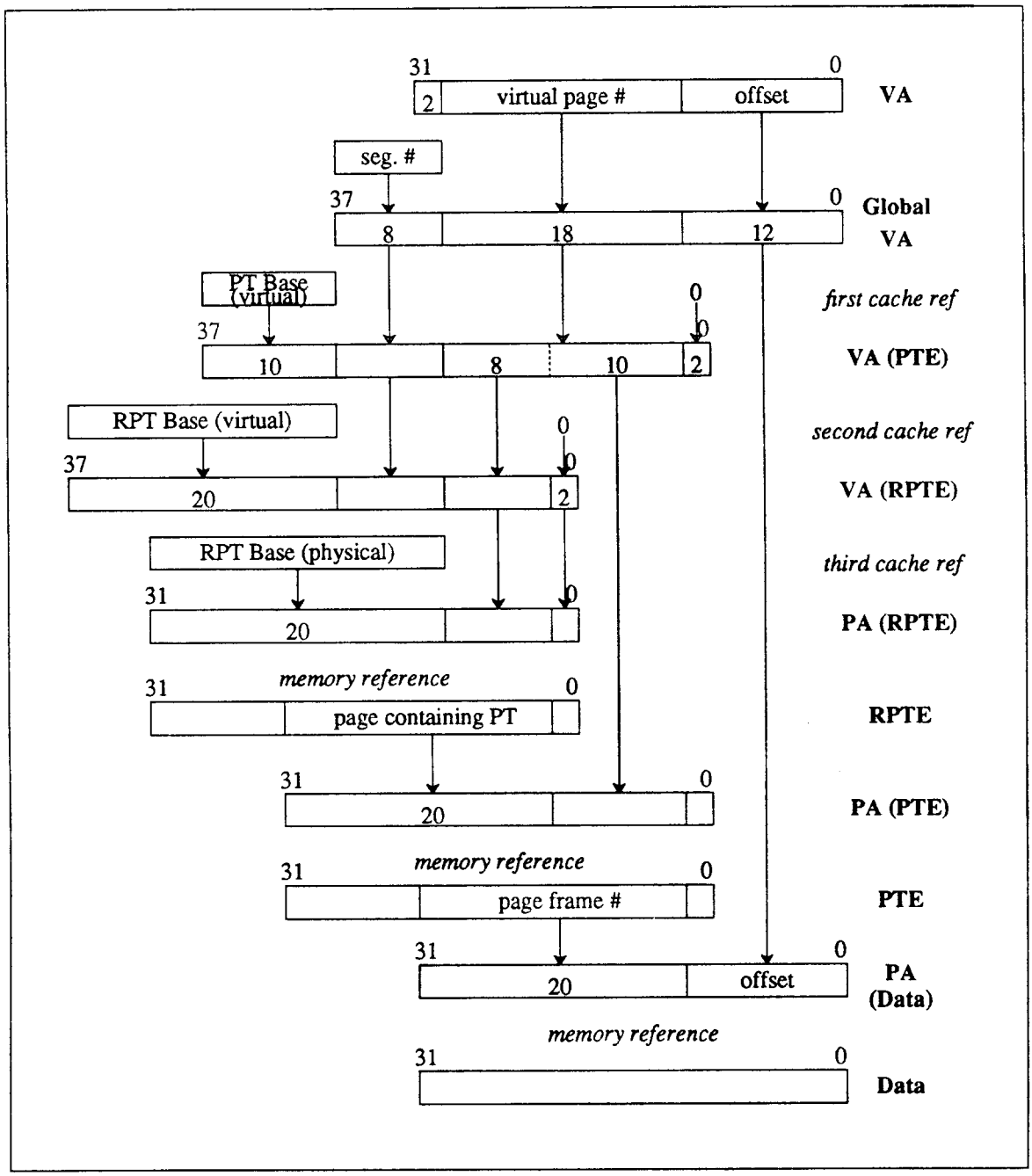

Figure 4 In-cache address translation mechanism - address mapping.

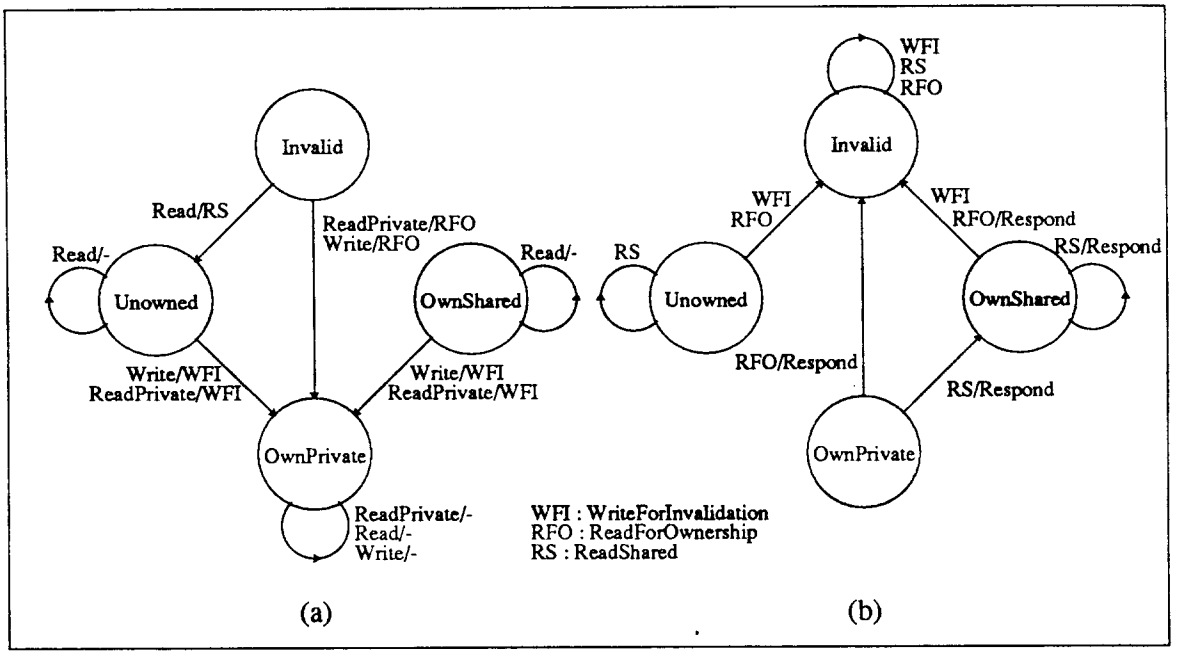

Figure 5 Cache block state transition diagram: (a) state transition due to CPU operations, and (b) state transition due to snoop operations. A label in an arc represents (Request Received)/(Bus Action). Processor FLUSH operations are omitted for simplicity.

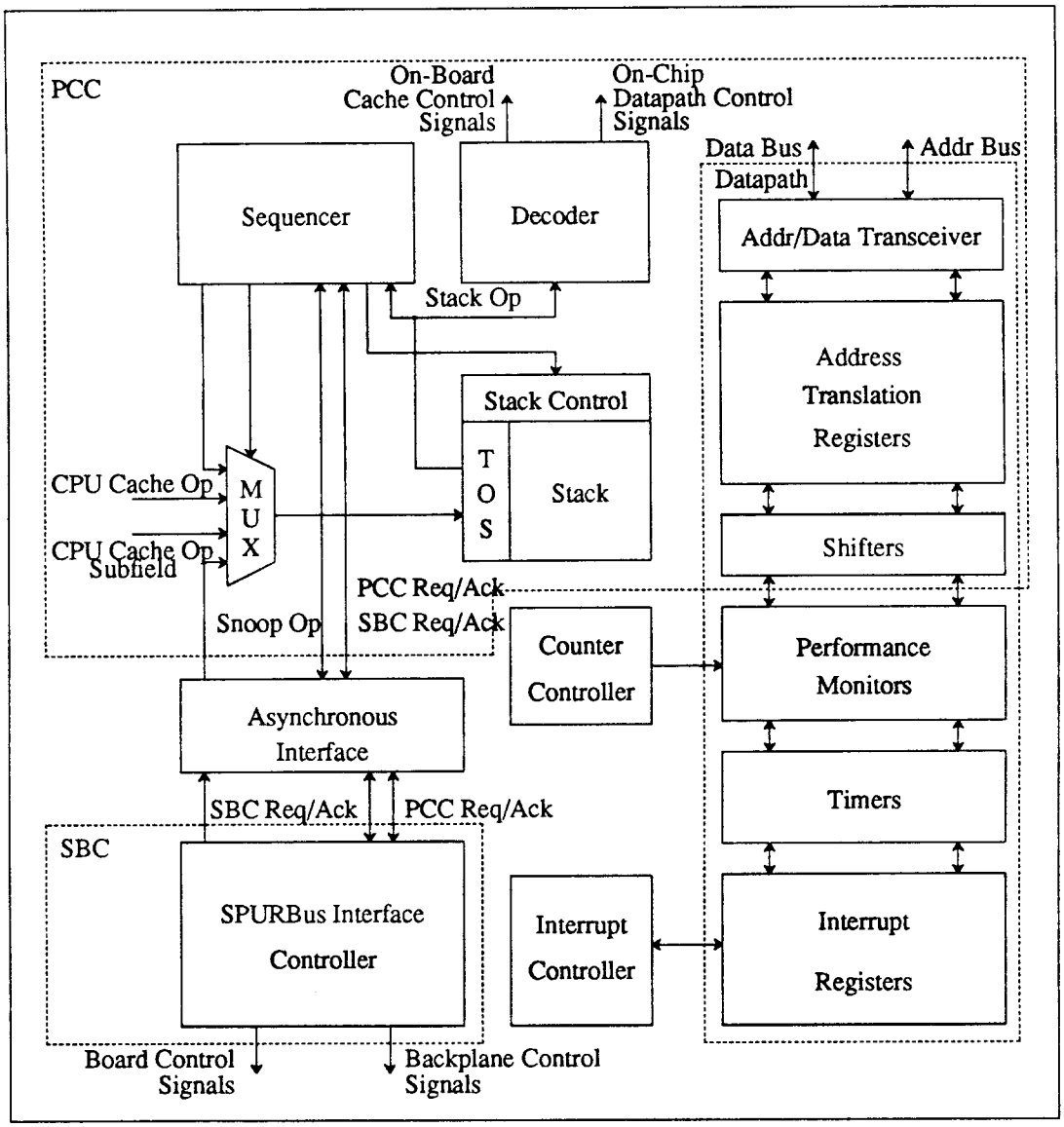

Figure 6 SPUR MMU/CC block diagram.

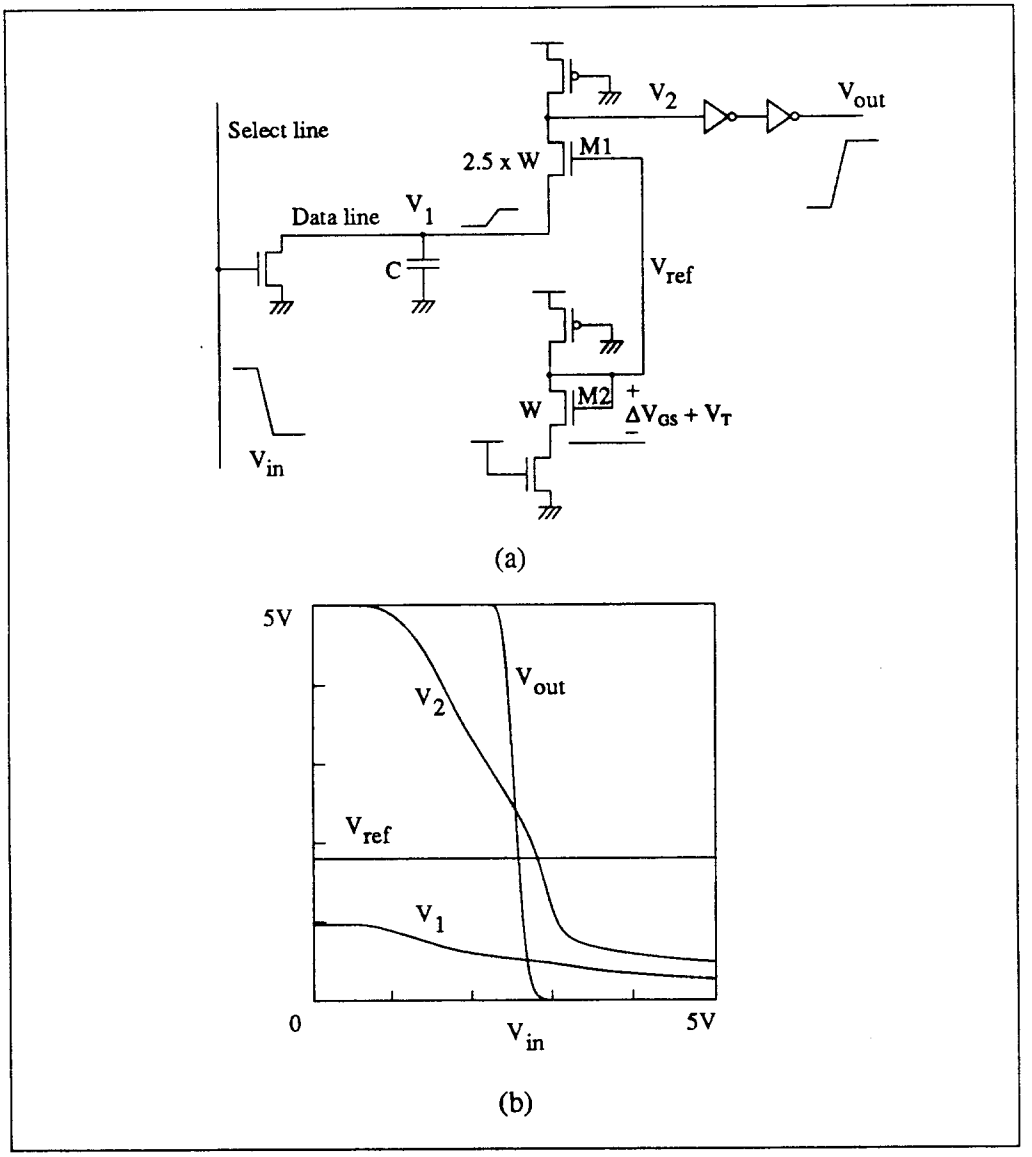

Figure 7 A PLA sense amplifier: (a) schematic, and (b) transfer curve.

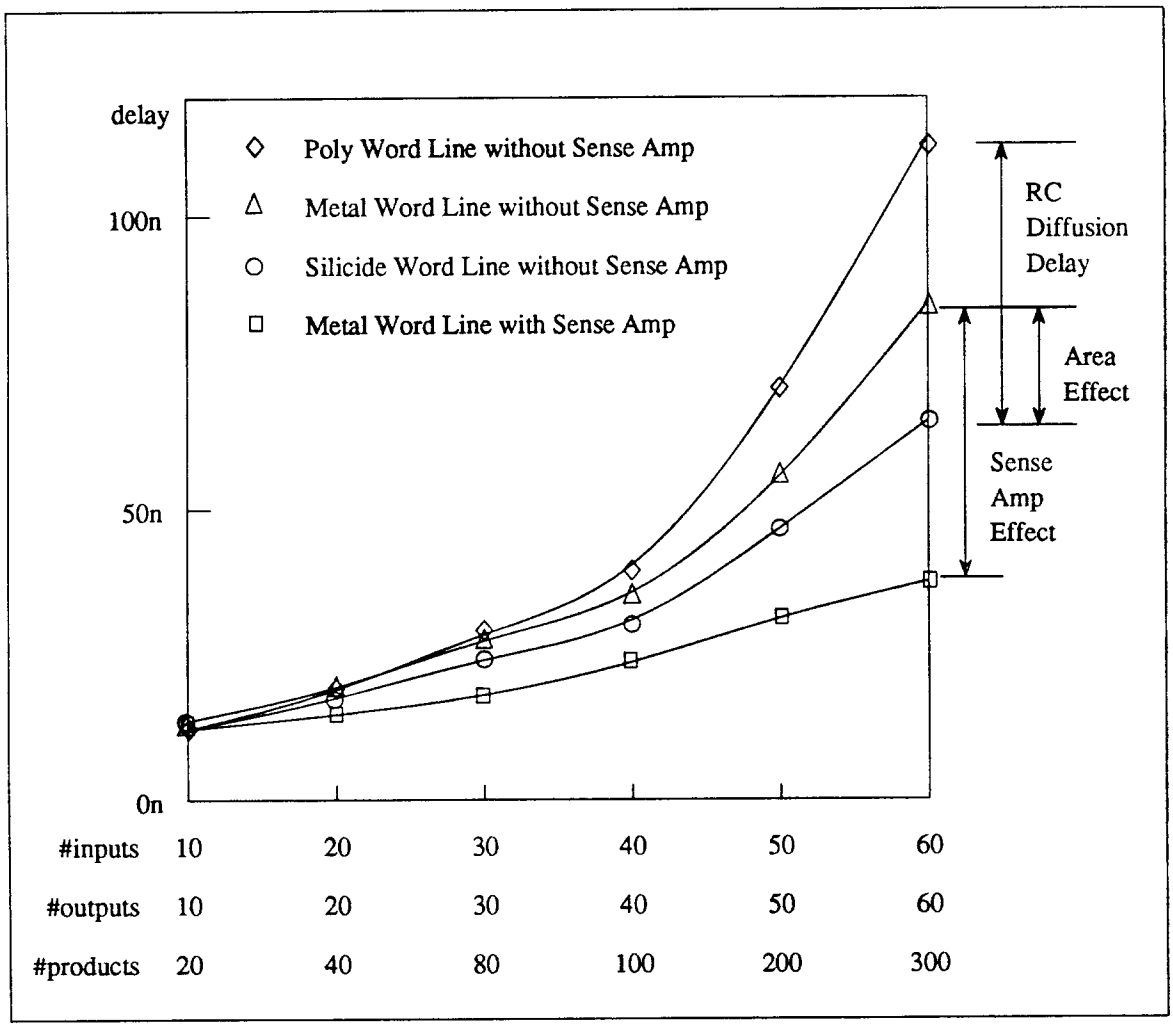

Figure 8 Comparison of worst-case delay among different PLAs.

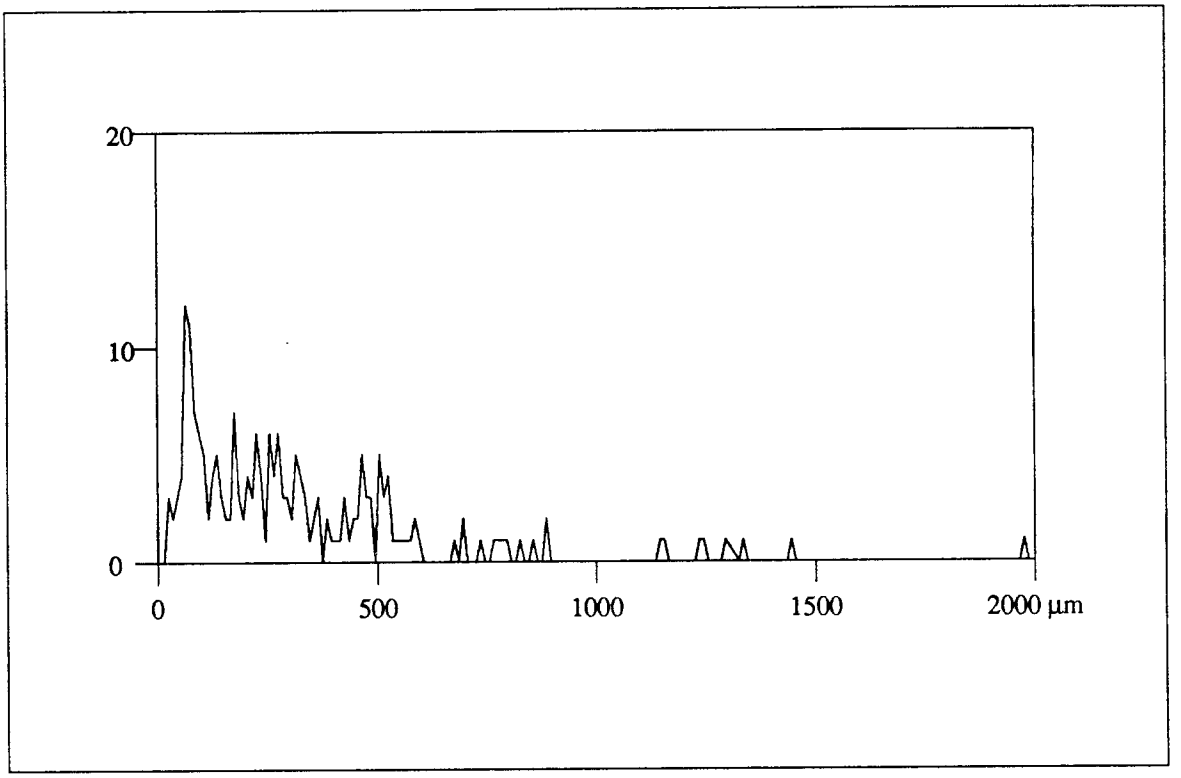

Figure 9 Net length distribution of the SBC.

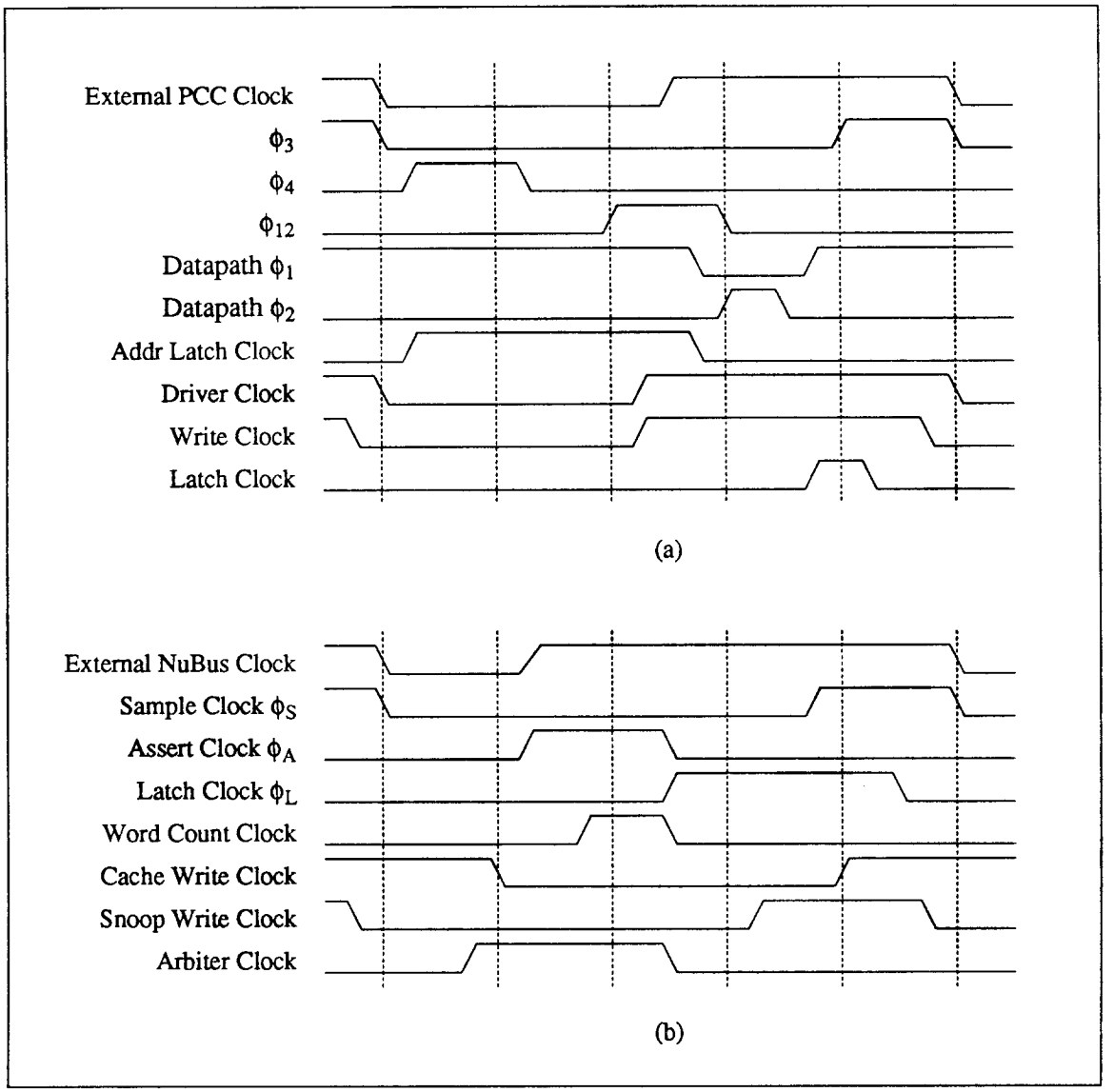

Figure 10 Internal clock phases in the MMU/CC: (a) PCC clock phases, and (b) SBC clock phases.

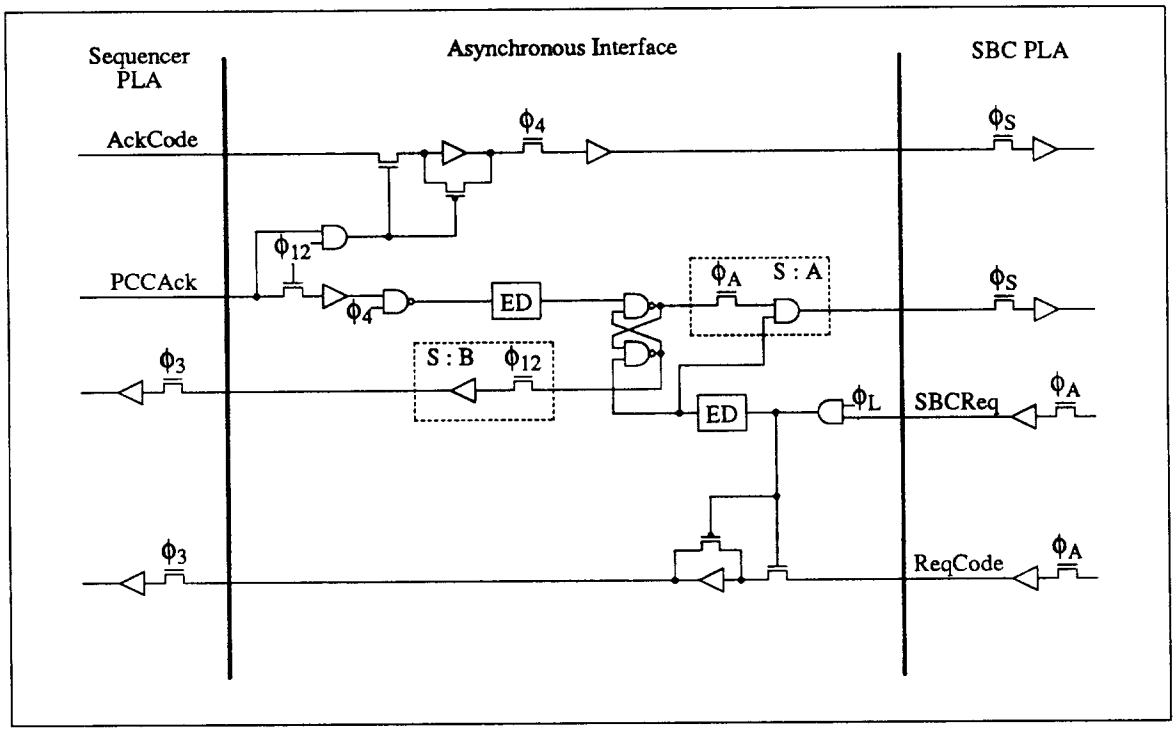

Figure 11 Interface between the PCC and SBC. ϕ_{12}, ϕ_3 and ϕ_4 are PCC clock phases, and ϕ_A, ϕ_B and ϕ_L are SBC clock phases. ED is an edge detector that generates a short, negative pulse on the rising edge of the input, and S is a synchronizer with or without reset terminal. AckCode and PCCAck are valid during ϕ_{12}. Synchronizer A is allowed a half cycle of the SBC clock for settling time while synchronizer B is allowed two fifths of the PCC cycle time.

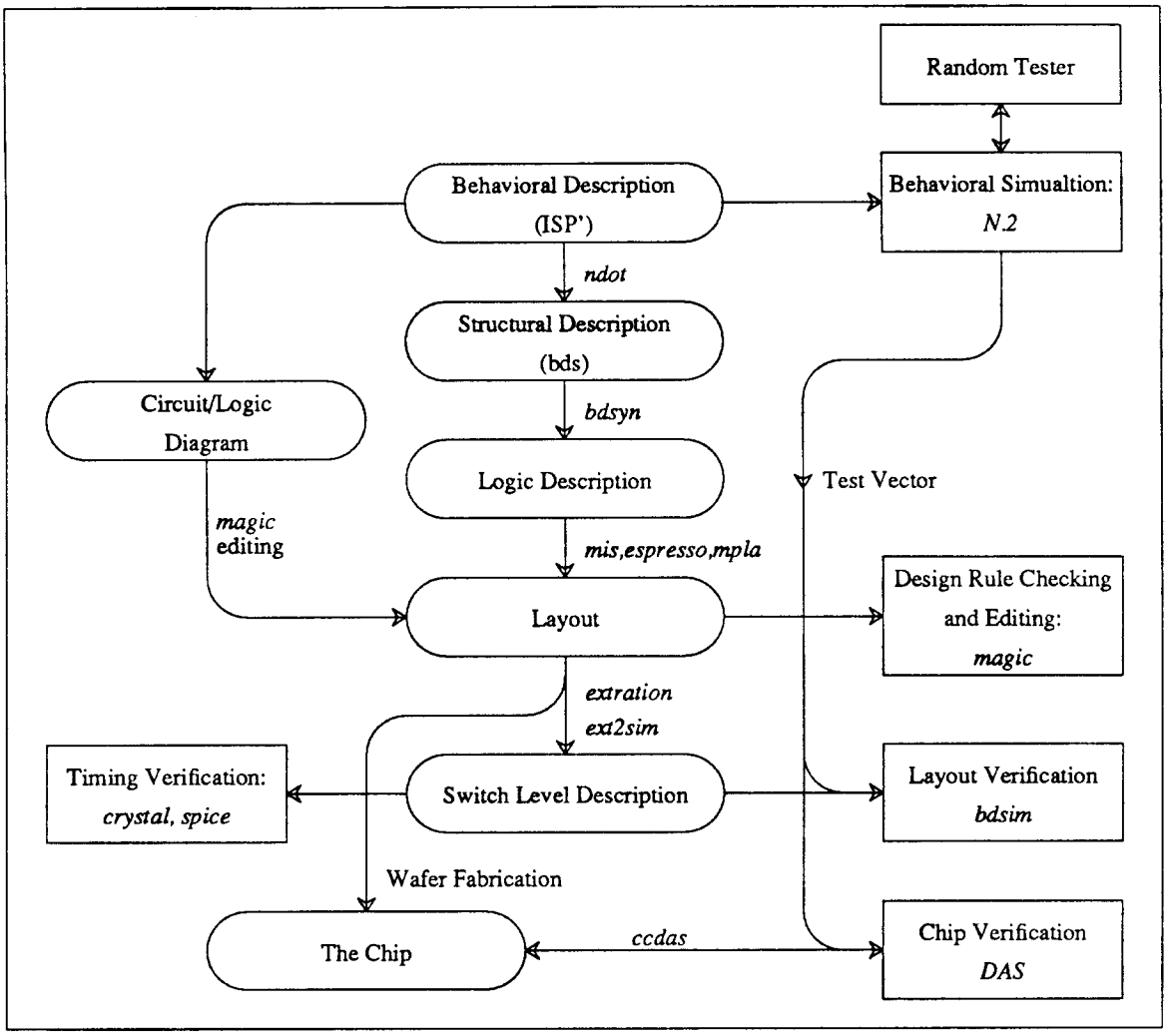

Figure 12 Design flowchart and CAD tools.