An On-chip Multiprocessor Microprocessor with Shared MMU and Cache

Yong-Hwan Lee, Woo-Kyeong Jeong, Sang-Jun An, and Yong-Surk Lee

Abstract

A multiprocessor microprocessor named SMPC(scaleable multiprocessor chip) that contains two IU (integer unit) is presented in this paper. It can execute multiple instructions from several tasks exploiting task-level parallelism that is free from instruction dependencies, and provide high performance and throughput on both single program and multiprogramming environments. The IU is a 32-bit scalar processor especially designed to boost up the performance of string manipulations which are frequently used in RDBMS(relational data base management system) applications. A memory management unit and a data cache shared by two IUs improve the performance and reduce the chip area required. The SMPC is implemented in VLSI circuit by custom design and automated design tools.

I. Introduction

The performance of microprocessors can be improved by decreasing cycle time or increasing the number of instructions that are executed in parallel. Pipelining divides the execution of an instruction into many stages in order to reduce the machine cycle time between each stages. However, the pipelining has physical limits such as clock skew which prevents further dividing the pipeline stage and complicates the pipeline control and trap handling.

Superscalar technique and VLIW architecture allow two or more instructions to be executed concurrently. A superscalar processor uses hardware to detect independent instructions and to dispatch them. In a VLIW processor design, it is the responsibility of a compiler to schedule codes so that dependencies between instructions are never violated. But, in either case, the instruction-level parallelism exploited by processors limits the number of instructions that can be issued together to at most four instructions on the average in typical programs. Four-way superscalar microprocessors generally produce the performance of about 2.5 IPC. Even microprocessors that can execute up to eight instructions take a little advantage over four-way superscalar processors and added functional units will be idle most of time. Thus the increment of the ability of executing multiple instructions from one thread could not be an optimum solution to improve the performance.

Manuscript received May 2, 1997; accepted July 7, 1997.

Y. H. Lee, W. K. Jeong, S. J. An, and Y. S. Lee are with Dept. of Electronic Eng., Yonsei University, Korea.

With the help of advanced integrated circuit technologies that can integrate the huge number of transistors, multiprocessor microprocessor is becoming increasingly attractive as a design option. The multiprocessor microprocessor executes multiple instructions from several tasks exploiting task-level parallelism that is free from instruction dependencies, and provides high performance and throughput on both single program and multiprogramming environment. In this paper, we present a multiprocessor microprocessor, called SMPC, in which two IUs (integer units) are connected in an efficient way.

II. SMPC Architecture

The SMPC microprocessor consists of two IUs, two instruction caches, one shared data cache, four small level-1 MMU, one large shared level-2 MMU, and a bus unit as shown in Fig. 1. The overall architecture of SMPC was designed with a goal that it should consume minimal chip area by simplified architecture while offering efficient communications and synchronizations between IUs.

1. Integer Unit

The IU is based on the SPARC 32-bit RISC architecture, which defines a processor capable of executing at a rate approaching one instruction per clock cycle [2]. In addition to the basic SPARC v8 instruction set, IU can execute the following two sets of extended instructions : SMIS (String Manipulation Instruction Set) and MCIS (Multiprocessor Control Instruction Set).

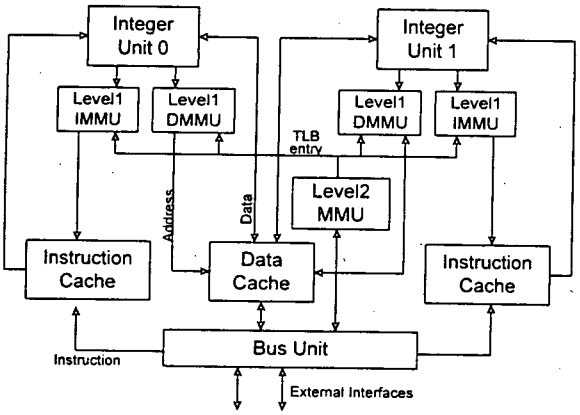


Fig. 1. Block diagram of SMPC.

The SMIS is devised for a need to boost up the performance of string manipulations which are frequently used in RDBMS applications. The SMIS includes *load/store update* instructions which load or store data together with effective address, *load/store string* instructions which load or store unaligned string in aligned form, and *compare string* instructions which compare one string register with another. These instructions can be processed as a single cycle instruction and if, instead of the SMIS, other instructions are to be used, two or more instructions would be involved. The SMIS makes it possible to enhance the performance of applications and to make programs compact with an optimizing compiler that is aware of the SMIS. The implementation of the SMIS requires a little additional hardware and does not degrade the overall speed since it is not on the critical path of IU.

The MCIS is proposed to support the programming environment in multiprocessor configurations. Each thread executed in IU should be synchronized if programs are to run as desired. This is accomplished by global variables and barrier registers. A lock array stores the addresses of the global variables and accesses to these variables are monitored. If there are multiple accesses from IUs, the sequence of accesses is arbitrated. The barrier registers store the barrier state of each IU to synchronize threads. The MCIS is composed of lock instructions, *lock set* and *lock clear*, and barrier instructions, *barrier set* and *barrier clear*. These instructions allow a multiprocessor to be used in efficient and simple way. Besides the MCIS, the SMPC offers interrupt distribution, reset generation and global clock to support multiprocessor.

The IU block diagram is shown in Fig. 2. By the implementation of a simple scalar architecture the IU has low design complexity to reduce silicon die size and to shorten design time. The IU has a simple five-stage pipeline for integer instructions that consists of F (fetch), D (decode), E (execute), M (memory) and W (write) as shown in Fig. 3. The instruction register for each stage stores the instruction flown from the previous stage with the progress of the clock signal. Then, the instruction

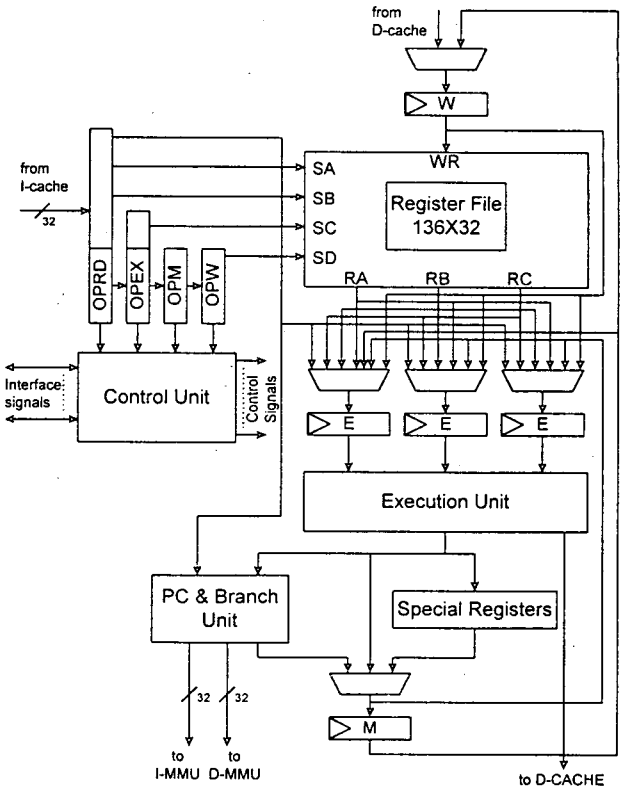


Fig. 2. Integer unit block diagram.

register is decoded to generate control signals for functional units. Particularly for instructions that require multi-cycle, IOPs (internal operation codes) are used. They are not regular codes defined by the instruction sets but are decoded in the same way as the instructions are processed. The IOPs are generated from a IOP table in the decoder and are inserted into the E-stage instruction register in case of multi-cycle instruction. The control unit includes decoder and IOP generator. OPRD, OPEX, OPM and OPW are the instruction registers corresponding to each pipeline stage. Such a time stationary decoding scheme with IOP makes the design of control unit simple and easy to implement.

The IU has three hardware interlocks, those are, register interlock, load interlock and store interlock. The register interlock can be removed by forwarding data. Possible forwarding paths are E-D, M-D(E), W-D(E) and load aligner-D(E). The paths through which the data are forwarded to D-stage provide source operands. The forwarding paths to E-stage supply the data for store instruction that follows. These forwarding paths are used when the source register field of an instruction and the destination register field of the next instruction are matched. The load interlock due to true dependencies delays the instruction that follows a load instruction by one clock cycle. The store interlock is caused by the fact that the store address may conflict with the address of the next store or load instruction on the bus since the store address is asserted during two cycles to

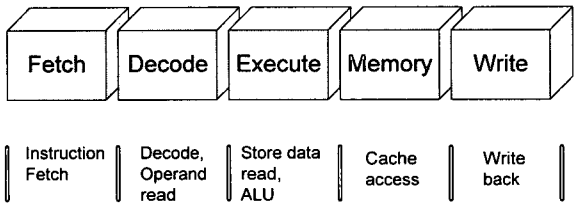


Fig. 3. Five stage pipeline.

check cache tag for a hit. In such a case, the store instruction becomes a two cycle instruction. The IU supports precise exception. Except for data access trap that is detected at M-stage and serviced at W-stage, all the traps are serviced at M-stage. Thus, the traps detected at D or E-stage are delayed until the trapping instruction arrives at M-stage.

ALU is responsible for integer arithmetic/logical computations. A 32-bit conditional sum adder is used for the ALU. To reduce die area, neither integer multiplier nor integer divider is included, but *step multiply* with early-out and *step divide* instructions are implemented. The ALU has an additional logic for SMIS and MCIS but its overhead is negligible. A separate 30-bit adder is provided to calculate branch addresses.

2. *Memory Management Unit*

The MMU provides one 32-bit virtual address to 36-bit physical address translation per cycle in most cases. To accomplish this, a TLB (translation lookaside buffer) is used to cache the virtual-to-physical address mappings of the most recently translated addresses. For the case of SMPC, two small level-1 TLBs are used for two MMUs, i.e., a two-entry instruction TLB for instruction address translations and an eight-entry data TLB for data addresses. Because there are two IUs in the SMPC, total four level-1 TLBs are used.

If accesses to the level-1 TLBs are missed, a fully-associative, 128-entry, level-2 shared TLB will be consulted for the missed page translations. Single die implementation allows IUs to share a large level-2 TLB without considering of pin counts. If an access to the level-2 TLB is a hit, the TLB miss penalty is one clock cycle. In addition, if there are multiple accesses to the level-2 TLB, the arbitration of requests adds upto three cycles to the miss penalty. However, the increased penalty does not degrade the performance too much because level-1 TLB miss occurs rarely and the probability of multiple misses is quite low.

The MMU uses a tree-structured page table in order to reduce page table sizes in the main memory. A level-2 TLB miss is handled by a hardware mechanism, that is, table walk that searches for a page table entry through a series of four page tables. To reduce the latency for a table walk, a page table pointer cache is used to decrease the number of steps.

There are cases where an access to the level-2 TLB is missed during table walk due to other requests. In these cases, the penalty of level-2 TLB miss is further increased. However, it is noticeable that the penalty hardly differs from that of conventional non-shared TLBs. Also in the conventional architecture, a table walk process cannot be initiated until other table walk process releases a main memory bus.

One advantage of the shared level-2 TLB is that it can reduce the total size of the TLBs. In addition, the table walk hardware with a page table pointer cache requires large area. The shared TLB of the SMPC make it possible to decrease the number of the table walk hardwares from four to one, further reducing the chip size. It is also advantageous that it can improve the performance by processors that prefetch the TLB entries of shared pages for each other while preventing storing multiple copies of the same TLB entry into TLBs.

3. *Cache Memories*

Each IU executes its own instruction stream fetched from its private instruction cache. There are two instruction caches for two IUs and each cache is a 8-kbyte, 2-way set-associative cache. Line size is 32-byte and replacement policy is LRU. It is virtually-indexed and physically-tagged cache and does not have the problems caused from address aliasing, which is difficult to resolve in case of virtually-tagged caches. The instruction cache need not to snoop bus transactions because the memory address space containing instructions is often marked as read-only pages. Therefore, *flush* instruction should be used if self-modifying code are to operate properly.

The advantage of shared-memory multiprocessors is the simplest and the most general programming model that allows easier development of parallel software and supports efficient sharing of instruction and data. But it suffers from potential problems in achieving high performance because of the inherent contention in accessing shared resources, which is responsible for longer latencies of shared memory. Private caches using write-back scheme as a write policy help to reduce average latencies, to increase effective memory, and to reduce bus utilization, achieving the same effect as multiprocessor systems with a wider bus. However, this solution imposes serious cache coherency problem that multiple copies of the same cache data block may exist in several caches. In this case, one processor may be unable to load the latest data immediately from its private cache because another processor modify its private cache only. If processors are freely allowed to update their own copies, an inconsistent view of the memory leads to program malfunction. Therefore, cache coherency protocol must be maintained to provide the latest data to any processors.

However, the data cache in SMPC does not need cache coherency protocol for two IUs because it is shared between IUs. There is no private copy of data for each IU of the SMPC. This feature of cache makes the memory operation faster and the hardware implementation associated with cache coherency simpler.

In addition, this shared data cache offers an efficient method to communicate between two IUs. One IU can communicate by simply writing data into the shared cache memory where another IU can read it immediately. Thus, the data cache allows multi-threaded programs to be executed much faster than ever.

The data cache is a 8-kbyte, 4-way set-associative, virtually-indexed and physically-tagged cache. The write policy is write-back. While it is not necessary to maintain cache coherency protocol between two IUs, cache coherency within SMPCs must be supported to operate properly. The overhead of sharing the data cache is that cache tag should be three-ported to satisfy two memory requests from each IU and one snooping request from bus simultaneously with no time delay. And the associativity of cache memory should be higher in order not to degrade the cache hit rate when independent programs are running on the SMPC. But the same prefetching effect as the shared TLB improves the performance of cache system when parallelized application is running.

The cache coherency protocol to support multiple SMPCs is maintained by the use of the five states of a cache line and the bus snooping mechanism. Basically, the five states are invalid (I), exclusive (E), shared (S), owned (O) and modified (M). But this number can be reduced to four state or three state to accommodate various system configurations. With these states, write-invalidate snoopy cache responds to processor loads, processor stores or bus transactions.

It is necessary to communicate and synchronize between IUs. The bus unit of SMPC provides global variables and barriers that are accessed by the MCIS. Global variables solve inter-processor communication problem and barriers perform synchronization between processes. The bus unit is responsible for connecting efficiently one SMPC with another, as well as interfacing these SMPCs with external cache memory or with memory sub-system through external bus. The system using multiple SMPCs results in a bus-based shared-memory multiprocessor system.

III. Simulation and Physical Design

The architecture of SMPC is validated by the performance simulations. And it is described in HDL (Hardware Description Language) to verify its functionality. Then, the SMPC layout is designed by both hand and automated synthesis tools on the basis of the verified HDL. Finally, timing simulation is performed to correct errors caused from delays and to estimate the operation frequency.

1. Performance Evaluation

The advantage of multiprocessor lies on the fact that it is possible to achieve high performance by assembling large numbers of slow processors. We developed IU with simple architecture regarding the area required. The preceding simulation we performed shows that the performance of IU is from 1.2 to 1.3 CPI [3]. And the previous work shows that the shared level-1 cache architecture usually outperforms the shared memory architecture substantially [4]. Therefore, simulations are focused on the performance evaluation of the shared MMU.

We conducted trace-driven simulations to find TLB miss rates and cache miss rates. Our workload for trace-driven simulations comes from the SPEC92 benchmark suite [5]. From that suite, nine programs were selected and were run on the SMPC IU simulator, which was previously developed and verified. Then, instruction addresses, data addresses and timing information are captured into a trace file. In order to reduce the size of the traces from long running programs, the trace was sampled down to a manageable size. The traces were fed as inputs to the MMU and cache memory model written in C language.

First, to determine the number of level-1 TLB entries, we simulated them with varying the number. For the instruction level-1 TLB, two entries are appropriate with 0.86% miss rate on the average. In case of the level-1 data TLB, more than two entries are required to keep the miss rate as low as instruction TLB since data address space is usually wider than instruction address space. Fig 4. shows the number of level-1 data TLB entries versus the miss rates in logarithmic scale. In this figure, for clarity, only five simulation results out of total nine simulations are drawn. From these simulation results, eight entries were selected as proper number showing 1.81% miss rate on the average. Next, TLB miss rates and cache miss rates were measured as shown in Table I. As expected, the small-sized level-1 TLBs exhibit relatively high miss rates. However, the overhead to handle a level-1 TLB miss is just one cycle and it only increases the total execution time by 1.53% on the average. Therefore, we expect that the prefetching effect can compensate for the increased overhead and that the reduced size of MMUs can allow more chip area to be used in other useful architectural features.

2. Functional Verification

Top-down approach is used in design phase of the SMPC microprocessor. The architecture of the SMPC is defined and is described in HDL. Then, simulation environment, in which we would expect the SMPC architecture to be used in real systems, is also built in HDL to evaluate the performance accurately and verify the operation correctly. By doing this, after the implementation from this HDL description, the microprocessor can provide proper interface and minimal errors. Fig 5. illustrates this functional verification scheme.

The workloads to verify its functionality are two sorts. Synthetic workload is written for the purpose of verifying the operations that is not usually occurred with user codes and real workload includes user application programs that are executed

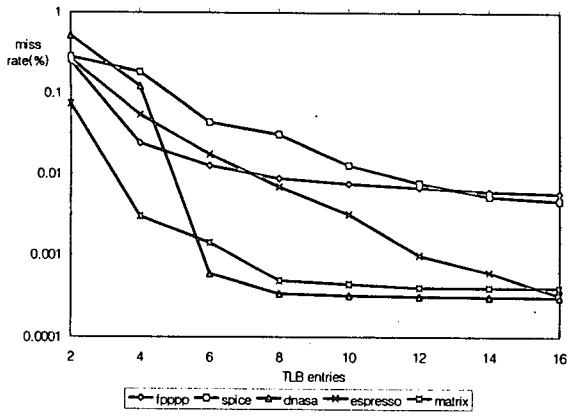


Fig. 4. Level-1 data TLB performance.

Table 1. TLB miss rates and cache miss rates.

	L1 ITLB miss	L1 DTLB miss	L2 TLB miss	Inst. Cache miss	Data Cache miss
gcc	1.8%	5.4%	0.013%	4.8%	4.5%
fpppp	0.29%	0.89%	0.001%	8.3%	2.1%
tomcatv	0.002%	2.8%	0.007%	0.005%	12.4%
spice	1.7%	3.1%	0.001%	1.5%	8.6%
dnasa	0.82%	0.03%	0.002%	0.004%	20.2%
espresso	1.2%	0.7%	0.001%	1.1%	0.45%
doduc	0.74%	2.63%	0.002%	4.3%	2.5%
xlisp	1.1%	0.69%	0.001%	0.82%	2.2%
matrix	0.001%	0.048%	0.01%	0.005%	5.1%
Average	0.86%	1.81%	0.004%	2.31%	6.45%

under the normal conditions. For the verification of the SMPC, both kinds of workloads were used. As synthetic workloads, about 140 programs written in assembly language were executed. During the execution, the results were automatically checked against the expected results on a cycle-by-cycle basis and the correctness of results was proved. Many user programs written in C and FORTRAN language were also executed as real workloads on the SMPC system described in HDL. The simulation results were compared with the results from real SPARC workstation. From those simulations, we knew that the SMPC can achieve from 1.4 to 1.8 IPC with various application programs.

3. Custom Layout

Register file and execution unit in IU consume large chip area and affect the operation speed of microprocessor. Therefore, we designed the layout of the register file and the ALU manually to get optimal design. Eight overlapped windows constitute the register file and the circuit to resolve overlapping addresses is included. As illustrated in Fig. 6., the register file has four ports,

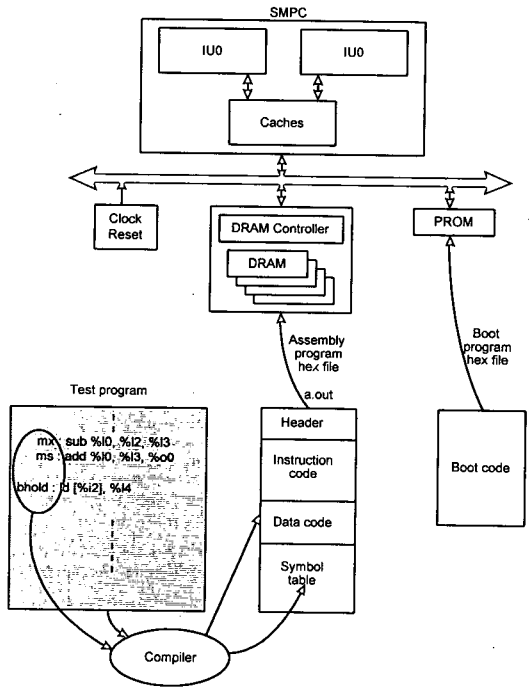


Fig. 5. Functional verification.

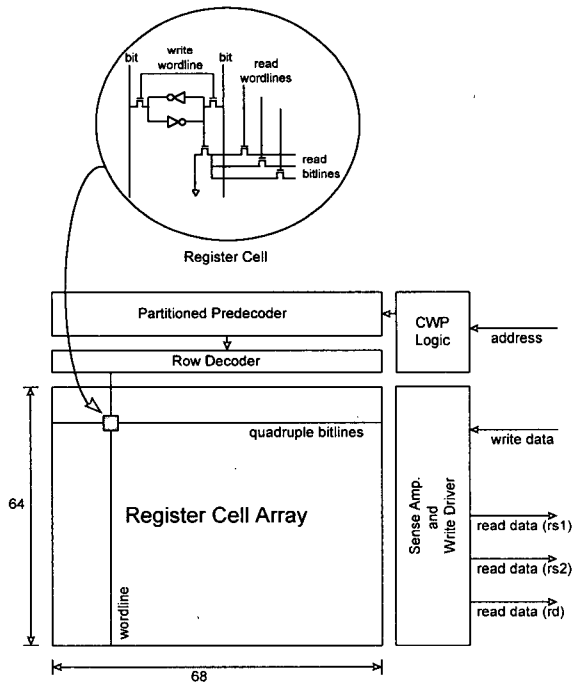


Fig. 6. Register file.

one for writing and three for reading, and total number of registers is 136 x 32-bit. Unlike conventional SPARC processors, three read ports allow store instruction to be executed in one cycle. Predecoder is partitioned to accommodate windowed register, thus reducing access time and power consumption.

The ALU in the execution unit is implemented to process

additional instructions from the SMIS and the MCIS. Particularly for the SMIS, it is required to detect null characters from operand register. The detection logic was designed to operate in parallel with 32-bit addition, keeping the total delay time to a minimum. The custom design of the ALU optimizes the placement of cells and the routing area.

4. Physical Design

The verified HDL description is converted into logic circuits by synthesis except for the register file and the ALU designed by custom layout. In the synthesis phase, SRAM is generated for the cache memories and complex control logic is synthesized into finite state machine. Other arbitrary blocks that do not affect the performance are synthesized with standard cell library.

First, the complete netlist of the SMPC are verified by performing static timing verification that detects electrical errors, ramp delays, critical paths and clock skew. Then, the timing simulation with the same stimulus that used in the verification of HDL is performed. The correctness of these results is determined by comparing with the results of previous HDL simulations. At worst case conditions, that is, 3.0V Vdd and 85°C temperature, the operation frequency is higher than 60MHz.

With automated placement and routing tools, we designed the layout of the SMPC microprocessor. It will be fabricated on 0.6um TLM CMOS fabrication process. The core size of the SMPC with two IUs is 11.0mm x 9.75mm. The layout of the SMPC is shown in Fig 7.

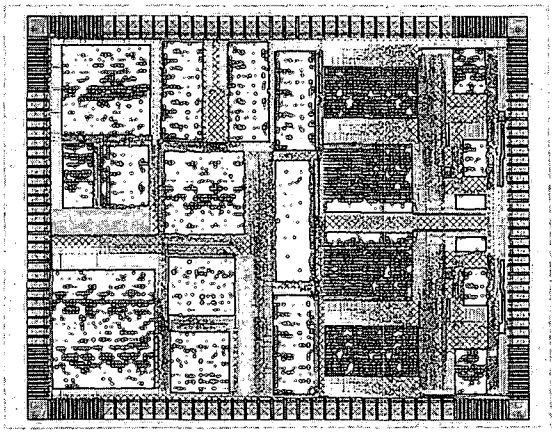


Fig. 7. SMPC layout.

Ⅳ. Conclusions

In this paper, we present the SMPC microprocessor in which two 32-bit IUs are connected efficiently to a shared MMU, two instruction caches and a shared data cache. The SMPC is a simple scalar multiprocessor which has low design complexity to reduce the silicon die size and to shorten the design time. The SMPC microprocessor can execute two tasks simultaneously, exploiting task-level parallelism. The IU was designed with the goal that the processor should be as fast as possible by the simplified implementation of the architecture, keeping the area to a minimum. In addition to the basic instruction set, the IU can execute two sets of extended instructions. The SMIS was devised to boost up the performance of string manipulations which are frequently used in RDBMS applications. The MCIS is proposed to support the programming environment in multiprocessor configuration. For fast page translations, a small level-1 TLB is employed. If an access to these level-1 TLBs is missed, a large level-2 TLB will be consulted for missed page entry. The data cache shared by two IUs allows the prefetching effect to improve the performance of system when a parallel program is running.

Top-down approach is used in design phase of the SMPC. After the performance evaluation with trace-driven simulations, the architecture of the SMPC is described in HDL. Then, simulation environment, in which we would expect the SMPC architecture to be used in real systems, is also built in HDL to verify the operation correctly. Verified description is converted into logic circuit by synthesis and custom layout by hand. After layout, timing simulation including the information of wire connections shows that the operation frequency is as high as 60MHz in worst case conditions. The die size of the SMPC with two IUs is 11.0mm x 9.75mm on 0.6um TLM CMOS fabrication process.

References

[1] J. Goodman, Cache memories and multiprocessors tutorial notes, *Proc. 3rd Intl Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 504-509, 1989.

[2] *The SPARC Architecture Manual*, SPARC International Inc., 1992.

[3] J. K. Ahn, B. I. Moon, S. K. Moon, Y. H. Lee, Y. S. Lee, S. H. Yoon, VLSI design of the control unit of a 32-bit RISC integer unit, *Proc. ITC-CSCC*, pp. 330-331, 1996.

[4] B. A. Nayfeh, L. Hammond and K. Olukotun, Evaluation of design alternatives for a multiprocessor microprocessor, *Proc. 23rd Intl Symp. on Computer Architecture*, pp. 67-77, 1996.

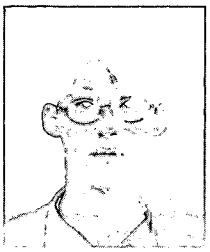
[5] K. M. Dixit, New CPU benchmark suites from SPEC, *Digest of Papers COMPCON*, pp. 305-310, 1992.

[6] A. Agarwal, B. H. Lim, D. Kranz, and J. Kubiatowicz, APRIL : a processor architecture for multiprocessing, *Proc. 17th Intl Symp. on Computer Architecture*, pp. 104-114, 1990.

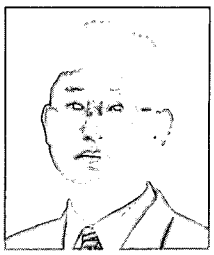
[7] S. J. Eggers, R. H. Katz, The effect of sharing on the cache and bus performance, *Proc. 3rd Intl Conf. on Architectural Support for Programming Language and Operating Systems*, pp. 257-270, 1989.

[8] K. M. Kavi, A. R. Hurson, P. Patadia, E. Abraham, P. Shanmugam, Design of cache memories for multi-threaded dataflow architecture, *Proc. 22rd Intl Symp. on Computer Architecture*, pp. 253-263, 1995.

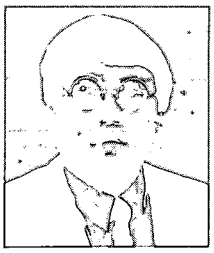
Yong-Hwan Lee was born in Korea on January 29, 1970. He received the B.S. and M.S. degrees in Electronic Engineering from Yonsei University, Seoul, Korea, in 1993 and 1995, respectively. He is currently working toward his doctoral degree. His research interests include the VLSI design of microprocessors and computer arithmetic.



Woo-Kyeong Jeong was born in Seoul on February 8, 1974. He received the B.S. degree in electrical engineering from Yonsei University in February 1996. He is now working toward the M.S.E. degree at Yonsei University. His research interests include high-performance microprocessor architectures.



Sang-Jun An received the B.S. and M.S. degrees in Electronic Engineering from Yonsei University, Seoul, Korea, in 1993 and 1995, respectively and is currently pursuing his doctoral degree. His research interests include the VLSI design of image compression algorithms.



Yong-Surk Lee received the B.S. degree in Electrical Engineering from Yonsei University, Seoul, Korea, in 1973. He received the M.S. and Ph. D. degrees in Electrical Engineering from the University of Michigan, Ann Arbor, in 1977 and 1981, respectively. He has been with various Silicon Valley semiconductor companies such as Sperry-Univac Computer, Hyundai Electronics USA, National Semiconductor, Performance Semiconductor, and Interl Corporation, as a design engineer. He is currenetly a professor in the Department of Electronic Engineering at Yonsei University, Seoul, Korea. His research interests include the design of microprocessors, SRAMs, caches, and image compression.