論文94-31A-10-18

내장형 32비트 RISC 콘트롤러의 VLSI 구현

(A VLSI Implementation of 32-bit RISC Embedded Controller)

李 文 基*, 崔 炳 允**, 李 承 浩*

(Moon Key Lee, Byeong Yoon Choi and Seung Ho Lee)

要 約

본 논문에서는 내장형 제어시스템에 활용될 수 있는 RISC 프로세서의 설계 및 구현에 관하여 기술하였다. 이 프로세서는 레지스터 파일, 파이프라인화 되어있는 실행유닛, FPU 인터페이스, 메모리 인터페이스, 그리고 명령어 프리펫치 유닛을 포함하고 있다. 대부분의 명령어는 20MHz의 2 페이즈 클럭에서 단일 사이클안에 실행하며, 인터럽트에 대한 최악 지연시간은 7 사이클로 이를 벡터형식의 인터럽트 처리방식에 의해 구현함으로써 실시간 처리 시스템에 응용될 수 있도록 하였다. 다중 사이클 명령어를 효과적으로 처리하기 위하여 사이클 카운터를 내장하는 데이타 정적 제어방식을 이용하였다. 이 칩은 1.0um CMOS 이중 금속 공정에 의해 제작되어 9.1mm × 9.2mm의 크기에 139,000개의 트랜지스터를 집적하고 있다. 소비전력은 5V 전원에서 20MHz에 동작할때 0.8Watt이다.

Abstract

This paper describes the design and implementation of a RISC processor for embedded control systems. This RISC processor integrates a register file, a pipelined execution unit, a FPU interface, a memory interface, and an instruction prefetcher. Its characteristics include both single cycle executions of most instructions in a 2 phase 20MHz frequency and the worst case interrupt latency of 7 cycles with the vectored interrupt handling that makes it possible to be applicable to the real time processing system. For efficient handling of multi-cycle instructions, data stationary hardwired control scheme equipped with cycle counter was used. This chip integrates about 139K transistors and occupies 9.1mm X 9.1mm in a 1.0um DLM CMOS technology. The power dissipation is 0.8 Watts from a 5V supply at 20 MHz operation.

*正會員, 延世大學校 電子工學科
(VLSI & CAD Laboratory, Dept. of Elec. Eng., Yonsei Univ.)
**正會員, 東義大學校 컴퓨터工學科
(Dept. of computer Eng., Dongeui Nat'l Univ.)
接受日字 : 1994年 1月 19日

I. INTRODUCTION

For a single chip implementation of a high performance microprocessor, the RISC approach has been favored because of the effective usage of the limited resources.[8.12.17] The RISC processors have been mainly

adopted in high performance workstation or file servers for their powerful computation capabilities. On the other hand, the embedded control systems such as laser printers, disk controllers and personal information equipment that do not require sophisticated memory subsystem and virtual address translation are very sensitive to the real time processing capability. Especially, these applications require the characteristics such as low power consumption and excellent cost-performance [16,18,11,13,14] It is, therefore, necessary to utilize the RISC architecture with streamlined instruction sets to take advantage of simpler and faster pipelined implementation to overcome the disadvantages of the conventional controllers having CISC architecture with poor computing power and low data bandwidth. The RISC architecture and 32-bit word length make it possible to enhance the performance of the system by reducing the number of clock cycles needed to execute an instruction and to obtain both low interrupt latency and efficient multi-tasking support.

This paper describes the VLSI implementation and measured results of the 32-bit RISC processor for embedded controller. The paper is organized as follows. Section II describes the design consideration of the architecture. The hardware implementation of the RISC processor will be presented in section III, and the verification methodology will be described in section IV. The fabrication, measurement, and conclusion will be shown at section V and VI, respectively..

Ⅱ. ARCHITECTURE

The architectural design of the RISC processor was focused on supporting both minimum worst case interrupt latency and fast context switching. For this processor to be applicable to the embedded control systems, we have implemented the vectored interrupt handling hardware and large register file with discrete register banking concept. [1,2,3] Fig.1 illustrates the block diagram of the RISC processor. Most of instructions except for load/store and jump/branch instructions are issued at a peak rate of one instruction per cycle with four stage pipeline. [4,5] All instructions are classified into three categories as follows : call type, branch type, and memory access type. The integer pipeline is four-stages deep : Instruction Fetch(IF), Decode(DEC), Execute(EXE), and Write-Back (WB). The virtual stage of IF-1 is the cycle immediately before the IF and is used to calculate the address of the next instruction and to drive it into address bus. In IF cycle, the next instruction to be executed is fetched by the address that has been calculated during the phase 1 of IF-1 stage and then output to the address bus in the phase 2 of IF-1 stage.

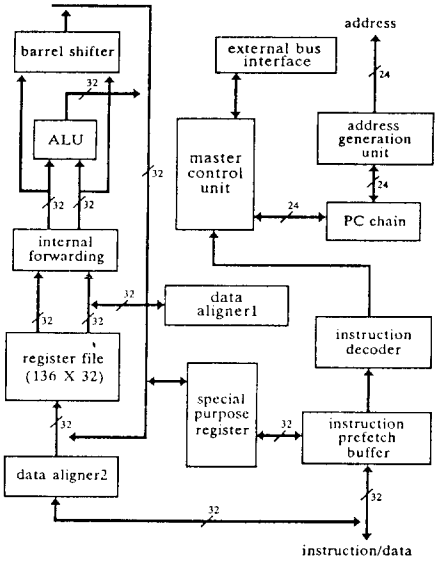


Fig.1. Block diagram of RISC processor.

The decode cycle makes the instruction decoder in the control unit decode all the control signals. During the execution cycle, the processor performs logic & arithmetic, and shift operations in the ALU and SAU

(Shift/Align Unit), respectively. Both load and store instructions also use this cycle to generate the absolute address.

Ⅲ. HARDWARE IMPLEMENTATION

1. Execution unit

Because the adder is an important data path for attaining high speed throughput, a 32-bit CSA(Carry Save Adder) equipped with 4 ripple carry blocks was adopted. This ALU does support 32-bit integer multiplication that can be achieved by shift & add operations as well as 14 arithmetic and logical operations. The SAU performs normal shifting, sign extension, and data alignment. Operating in parallel with the ALU ensures that it will not be in the critical path. The shift network was implemented using a modified 64-to-32 funnel shifter. Two least significant bits of the effective address determine the types of the byte swapping. Therefore, remarkable increase in the efficiency of hardware can be obtained not by an additional aligner, but by only executing the shift & align instruction. The register file contains 136 entry general purpose registers each of which is 32-bit, and it is divided into 8 overlapping windows to exploit the characteristics of high speed function call and return procedures.

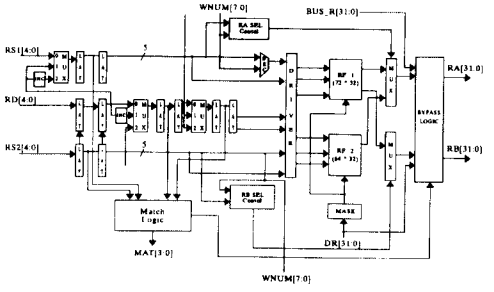


Fig.2. Block diagram of register file.

To reduce the read and write access time of the register file, we partitioned the register cell array into RFA1 and RFA2 that contain 72 and 64 registers respectively, and it makes an improvement in the access speed of about 3ns compared to that of one 136 cell array. Fig.2 shows the block diagram of the register file.

The eight-transistor register cell used in the register file has a D latch structure with two read ports and a write port. In the pipelined execution of the instruction stream, the data path incorporates two bypass paths for three dependencies that can occur between the current operation and the results of the previous two instructions. Bypass unit lets the current instruction use the results of the previous instructions that have not yet been written back into the destination.

The special purpose registers(SPR) consist of four registers to be used for trap handling, system control, integer multiplication, and window invalidation. The PSR(Process Status Register) contains condition codes, system status control codes, trap control codes, and register window invalidation code. The Y register is used for 32-bit integer multiplication, and WIM(Window Invalid Mask) limits the number of available windows in the register file. Because some registers of the SPR may be in the incorrect state, recovering the state of SPR into the previous one is needed before the trap handling takes place. To provide the SPR with this rollback capability, we implemented the SPR composed of a normal state register, a backup copy register and a multiplexer. The rollback action is performed by transferring the data from the back-up copy register to the normal state register in the occurrence of a trap.[17] The instruction fetch unit is organized into a program counter chain, an offset adder, an incrementer, and an instruction prefetch queue that is responsible for generating the next instruction address. There are 4 program counters corresponding with the four stages of the instruction pipeline. The incrementer is a 30-bit adder that always adds 4 to the current PC value

under the normal operations whereas the offset adder adds an offset to the instruction address for the control transfer in the DEC stage. All the possible next addresses have to be calculated in the PC chain whether they will be used or not, because it may exhaust additional 18ns delay to evaluate the branch condition if there is a branch instruction. Both an incrementer and an offset adder were implemented using the 30-bit CSA. The distance between a conditional branch instruction and the special one that updates the condition codes of the PSR affect the normal pipelined execution. By evaluating the condition code fields of the PSR and 4-bit condition codes of the branch instruction, the conditional branch instruction of which the distance is larger than 1 cycle can determine whether it should jump to the target instruction or not. Because the instructions that update the condition code field of the PSR modify it in the second half cycle of the EXE stage, control branch instruction should use the ALU-generated condition code. To control both cases, we designed dual-branch evaluation circuit as shown in Fig.3.

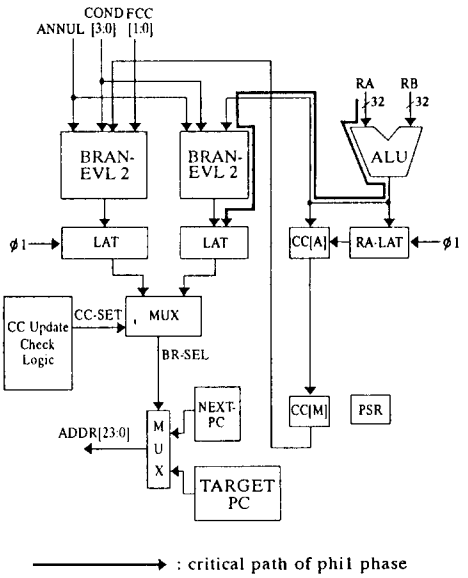


Fig.3. Branch address selection with dual-branch evaluation circuit.

It is divided into a BRAN-EVL1 and a BRAN-EVL2. The BRAN-EVL1 contains the evaluation logic for condition codes of the PSR. The BRAN-EVL2 includes only the branch evaluation logic of CPU using condition code outputs of the ALU. The CC-SET signal represents whether instruction immediately before the conditional branch is the one that updates the condition code or not. This signal selects one of the control signal outputs of the BRAN-EVL1 and the BRAN-EVL2. By adopting this circuit, one-cycle branch execution can be possible. As shown in the Fig.3, the routing path from the ALU to the BRAN-EVL2 is the critical path of 33ns for phase 1.

2. Control unit

(1) Organization

The control unit is organized into an exception module, a pipeline control module, an immediate data module, and a data dependency check module. The pipeline control module manages the intrapipeline control and generates the various signals to control the data path efficiently. The immediate data module handles the data constant needed in the data path using sign extension and formatting logic. The data dependency check module detects the dependencies that can be generated between the current instruction and the previous two instructions, and then transfers the results to the bypass control logic and the hardware interlock control logic that are embedded in the pipeline control module. The exception module manages the external interrupts or traps generated during the execution of instruction, outputs each trap vector, and enables the pipeline flush logic embedded in the pipeline control module. Although a data stationary pipelined control method is more expensive in terms of hardware, this arrangement is certainly more flexible and has faster decoding time than that of time stationary control approach because the

decoding operation can be distributed among the various pipeline stages through the delay chains.[19] Therefore, for an intrapipeline control scheme, the data stationary hard-wired pipeline control scheme was adopted to exploit the merits of both high speed and distributed decoding. To control the initiation

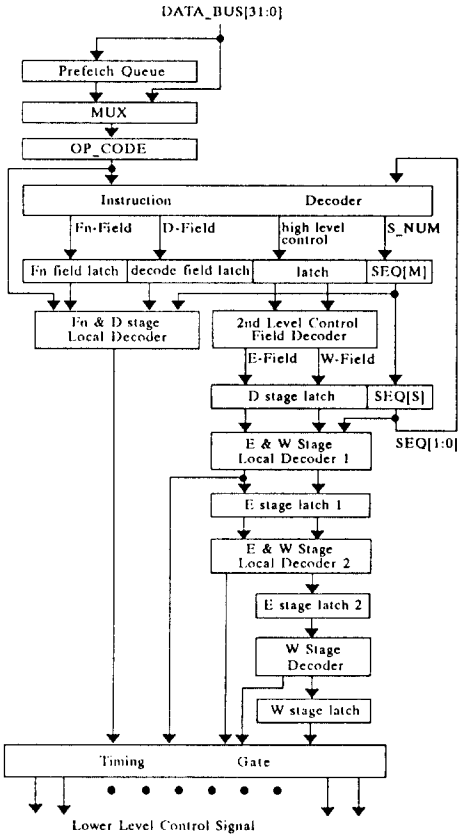


Fig.4. Data stationary pipeline control module with cycle counter and status bit.

of both single-cycle instructions and multi-cycle instructions, we added a cycle counter to the conventional data stationary pipelined control module. Fig.4 represents the pipeline control module equipped with a 2-bit cycle counter and 5 state bits. The cycle counter is used to trace the progression of the current multi-cycle instruction. A 2-bit counter is good enough because the maximum number of cycles of any instructions can't exceed 4. As shown in Fig.4, the new cycle value is calculated depending on the values of op code and the current cycle counter SEQ〈s〉 [1:0] , and is clocked into the master latch SEQ〈m〉 and the slave latch in phase 1 and phase 2, respectively.

The value of the cycle counter SEQ〈M〉 [1:0] can be calculated as follows.

$$SEQ<m>[n+1] = \begin{cases} 0 & , \text{if } ((IF\_IR[OP\_code] == 1 \text{ cycle inst}) \,||\, Reset) \\ R-1 & , \text{if } ((IF\_IR[OP\_code] == R \text{ cycle inst}) \,\&\& \\ & \quad (SEQ<s>[n]) \,\&\& \sim Reset) \\ SEQ<s>[n] & , \text{if } ((IF\_IR[OP\_code] == R \text{ cycle inst}) \,\&\& \\ & \quad (SEQ<s>[n] != 0) \,\&\& \sim Reset) \end{cases}$$

In the above equation, SEQ〈s〉 [n] and SEQ 〈m〉 [n+1] represent the values of the slave output latch in the current cycle and that of master output latch in the next cycle, respectively. The IF_IR [OP_code] represents the op code in the IF_IR register. The n and R correspond to the time step and the number of clock cycles needed for multi-cycle instructions, respectively. SEQ〈m〉 [1:0] of being 00 means that the current cycle is the last pseudo cycle of multi-cycle instruction, and following phase 2 is the one that IF-IR must receive the new instruction from off-chip memory or instruction prefetch queue. The values of IF-IR and 2-bit cycle counter make it possible for multi-cycle instructions to behave like several pseudo-instructions. This method is different from the IOP (internal op code) technique that has been adopted as a multi-cycle instruction handling scheme in the past RISC processor.[9,10,15] The IFQ(instruction prefetch queue) is used for capturing one or two instructions to make the pipeline busy. 5 state bits contain the information that indicate the final decode cycle of multi-cycle instruction, detection of interlock condition, squashed branch, and the current state of pipeline flush logic

enabled by a trap or an interrupt.

(2) Exception and error mode control

Program interrupt and external interrupt such as reset, illegal instruction, window overflow/underflow, floating point exception and tag overflow have an influence on the exception handling of the CPU. To support the precise exception handling based on a sequential architectural model, an instruction restart exception handling with pipelined exception acknowledge method was adopted. This type of exception handling gives the highest priority over the instruction that comes in first when multiple instructions incur the trap simultaneously. Fig.5 describes the exception handling circuit. In this figure, the prefetch trap decoder is an exception detector that is related to the instruction fetch operation. The exception check logic compares the various traps and interrupts occurred in each pipeline stages, determines only one exception to be processed first among them. Also, it generates the control signals that are accompanied by pipeline flush, saving of PC and next PC to the trap windows, and prohibition of additional trap detection. The first address of exception service routine should be reported by vector encoder according to the result of exception check logic.

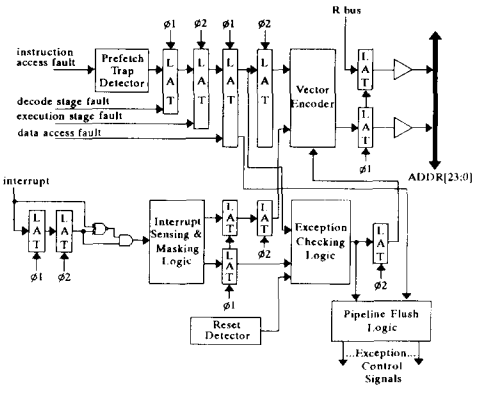


Fig.5. Block diagram of exception handling circuitry.

3. layout design

Layout of the chip was performed by partitioning the chip into several standard cell areas and data path blocks. We implemented the ALU, SAU, some part of IFU, and the register file by a data path compiler. The physical layout of the standard cell areas was tailed to have a regular structure and satisfy the boundary connection constraints. To reduce the RC delay induced by two clock lines and some internal nets that was very sensitive to skew, the highest priority was given in routing those signals. To verify the final layout, switch level simulation was carried out using the netlist extracted from the layout, and net comparison between the netlist and schematic was performed. To minimize the dI/dt noise due to the simultaneous current switching of some output buffers of the chip, we focused on the following constraints : First, assigning a pair of VDD and GND pads per four outputs or bi-directional pins instead of an on-chip decoupling capacitor. Second, separate pad rings were used for internal core and output buffers. Third, all of 38 VSS pads were connected internally to alleviate the fluctuation of VSS voltage. In this paper, a pair of VDD & VSS pads were assigned for each bi-directional pin and M outputs according to the following equation.

Nvss(Nvdd) = (Nout + N bi) / M

where Nvdd(Nvss) is the number of VDD or VSS pads,

Nout is the number of output pins and

Nbi is the number of bidirectional pins.

By this equation, the number of VSS & VDD were calculated by adopting M value of 3, and 4, respectively.

Ⅳ. DESIGN VERIFICATION

To verify the operations of RISC processor, instruction level, functional level, logic level, and circuit level simulation were performed. As a scheme of instruction level verification

shown in Fig.6, we have executed the test program written in C language, such as Hanoi tower, bubble sort, and matrix multiplication on the processor. The test programs were compiled by C compiler with static binding option and then we analyzed the address trace for the compiled code using GNU debugger program. [23] Based on the analysis of address trace for the test program, we constructed the test system composed of behavioral-level models of the CPU, 32K x 32-bit ROM for booting procedure code, test program, and 32K x 32-bit RAM for data and stack area. After converting the assembled code into the machine code appropriate for ROM and RAM model, we performed the behaviorallevel simulation by downloading the test program and data code into the memory. The result of the behavioral-level simulation was compared with that of SUN workstation.

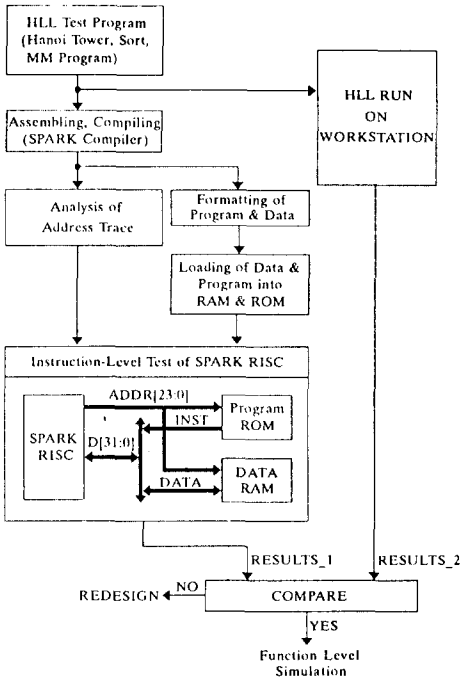


Fig. 6. Instruction level verification of the RISC processor.

By running several benchmark programs on the test system, we verified many design ideas and fixed a number of design bugs related to window control logic and hardware interlock control.

We also performed the gatelevel simulation and fault simulation of the RISC processor using the test instruction sequences generated with the functional fault model. [21,22,24,25] The fault coverage obtained by these test sequences was about 89 percent.

V. FABRICATION AND MEASUREMENT

The chip was fabricated in a 1.0um twin-tub CMOS process technology with double layer metal. A microphotograph of the fabricated

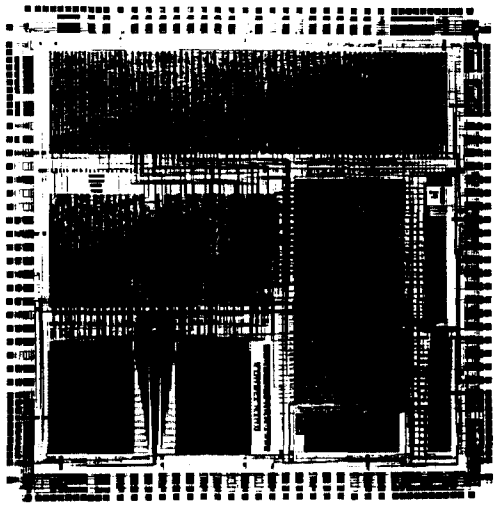


Fig.7. A microphotograph of the chip.

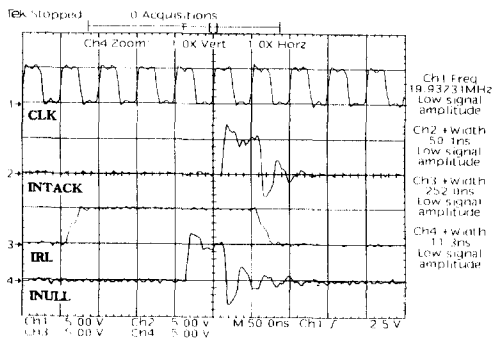


Fig.8. Measured waveform of interrupt handling operation.

chip is shown in Fig.7. Table1 shows the characteristics of the chip. Fig.8 shows the measured operating waveforms at room temperature for the external interrupt acknowledge response that was measured with 50pF load capacitance and 20MHz of operating frequency.

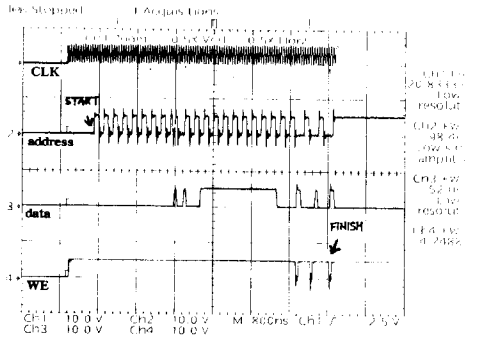


Fig.9. Measured waveform for 32-bit integer multiplication.

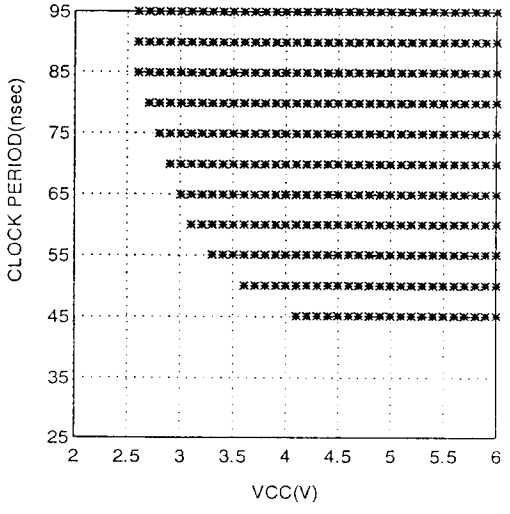


Fig.10. Shmoo plot of the clock period versus power supply.

This figure shows that INTACK(INTerrupt ACKnowledge) signal is generated exactly in five cycles after IRL(Interrupt Request Level) signal is issued. Fig.9 represented the digital oscilloscope trace for a 32-bit x 32-bit integer multiplication. It takes about 2.0us to multiply two 32-bit integer values including the cycles that are required to load multiplicand and multiplier in Y register and input latch B of ALU, respectively. Those additional 32-bit Y register and 2 multiplexers could enhance the integer multiplication performance compared with that of other RISC chip not having a multiplication instruction.

The measured schmoo plot of the functional operating clock period versus supply voltage is shown in Fig.10. This chip operates as high as 25MHz between power supply of 4.2V and 6V. Also, we made sure that the chip runs correctly at maximum clock frequency of 25MHz with critical path delay of 28.8ns. The CPI(Clock Per Instruc-tion) can be calculated as 1.3 by taking into account the occurrence of many single cycle instructions and a few multi-cycle instructions.

Table 1. Characteristics of the fabricated chip.

Number of Instructions	46
Number of transistors	139K
Chip area	9.1mm X 9.1mm
Process technology	1.0um CMOS DLM
Package	160pin QFP
MAX frequency	25MHz
Power dissipation	0.8Watts @20Mhz
Clock Per Instruction	1.3
Performance	20 MIPS

Ⅵ. CONCLUSION

A RISC processor designed for embedded control application implemented using the 1.0um double metal CMOS process has been described. This chip is composed of an instruction fetch unit, an execution unit, a control unit, an 136 entries register file, a memory interface unit, and a floating-point interface unit. The data stationary hardwired control scheme with cycle counter is

implemented to handle multi-cycle instruction and data path efficiently. The structured verification methodology based on HLL test program was applied during the design of the chip. This chip has a structure appropriate for embedded control systems because of its characteristics such as multi-window register file with register banking scheme to support fast context switching time, worst case interrupt latency time of 7 cycles with vectored interrupt handling, and a memory-mapped input/output addressing which can be an advantage in the core architecture. The fabricated chip occupies 9.1mm X 9.1mm and consists of about 139K transistors. The measured results show that this processor achieves a performance of 20MIPS. Because of a few multi-cycle instructions, this processor delivers the performance of an average CPI of about 1.3.

REFERENCE

[1] Moon Key LEE, A Study on the design of Multiprocessing RISC, VLSI & CAD Lab., Yonsei Univ., TR.90-1-1, Feb. 1990.

[2] Moon Key LEE, A Study on the Design of Control Unit for RISC, VLSI&CAD Lab., Yonsei Univ,TR90-1-1, Feb.,1990.

[3] B.Y.Choi, E.K.Kim, and M.K.Lee, "Design of a register file and its peripheral circuit for RISC", J. KIEE, vol.28, no.7, pp.10-21, July. 1991.

[4] M.K.Lee, B.Y.Choi, S.I.Son, "VLSI Design of High Performance RISC," Proc. ICEIC '91, Yanbian, China, Aug. 1991.

[5] M.K.Lee, B.Y.Choi, S.H.Lee, and S.I. Son, "VLSI Design of High Performance RISC Microprocessor," International Conference on VLSI and CAD '91, Seoul, KOREA, pp.88-91, Oct. 1991.

[6] M.K.Lee, B.Y.Choi, K.Y.Lee, and S. H.Lee,"Data stationary controller for 32bit application-specific RISC", Preceedings of ISCAS 93, Chicago, Vol. 3 of 4, pp. 1933-1936, May3-6. 1993.

[7] M.K.Lee, B.Y.Choi, S.H.Lee, and K. Y.Lee,"ASIC Design of a High Performance RISC", Proceedings of EURO ASIC92, Paris, pp.262-265, June1-5, 1992.

[8] John L Hennessy, and David A Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann Publishers, 1990.

[9] Minolis G.H.Katevenis, *Reduced Instruction Set Computer Architecture for VLSI,* MIT Press, 1984.

[10] Mark Horowitz, and Paul Chow, "MIPS-X : A 20 MIPS Peak, 32-bit Micro-processor with On - chip cache," IEEE J. Solid State Circuits, vol. SC-22 , pp.790-799, Oct. 1987.

[11] Jiro Miyake, Toshinori, "A Highly Integrated 40 MIPS(Peak) 64-bit RISC Microprocessor," IEEE J. solid state circuits, vol. SC-25, no.5, pp.1199-1206, Oct. 1990.

[12] Shing Kong, David Wood, Garth Gibson, Randy Katz, and David Patterson, "Design mothodology for a VLSI Multiprocessor Workstation," VLSI System Design, pp.44-54, Feb. 1987.

[13] Norman P. Jouppi and Jerffery Y.F. Tang, " A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance," IEEE J. Solid-State Circuits, vol. SC-24, No.5, pp.1348-1359, Oct. 1989.

[14] Jonathan Lotz, Bob Miller, Eric Delano, Joel Lamb, and Mark Forsyth, and Tom Hotochkiss, "A CMOS RISC CPU Designed for Sustained high Performance on Large applications," IEEE J. Solid State Circuits. vol. SC-24, no.5, pp.1190-1198, Oct. 1990.

[15] Mike Johnson, *Superscalar Microprocessor Design,* Prentice Hall, 1991.

[16] Rolando Carreras and Peter von Clemn, "Embedded control demands fast

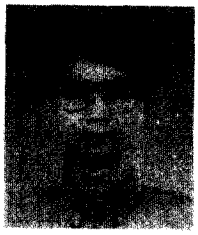
context switching," Electronics Engineering, pp.106-113, Sept. 1991.
[17] Jum Numata, and Nobuhisa Watanabe, "The CPU Core Architecture of a 4-Bit Microcomputer," VLSI Design, pp.40-48, May. 1985.
[18] LSI logic, L64901 SPARC 32-bit embedded processor system design guide, LSI Logic Corporation, 1990.
[19] Peter M Kogge, *The Architecture of Pipelined Computer*, Hemisphere Co Inc., 1981.
[20] James E. Smith and Andrew. R. Pleszkum, "Implementing of Precise Interrupt in Pipelined Processor", The 13th Symposium on Computer Architecture Conference Proceeding, Boston, MA, pp.36-44, June. 1985.
[21] Noice David Cooke, A clocking methodology Discipline for two-phase digital Integrated Circuits, Ph.D. Dissertation, Stanford Univ, 1983.
[22] A.F.Champernowne, L.B.Bushard, I. T.Rusterholz, and J.R Schomburg, " Latch - to - Latch Timing Rules", IEEE Trans. on Computer, vol. C-39, no.6, pp. 798-808, 1990.
[23] R. M.Stallmann, Using and Porting GNU CC, Free software Foundation INC., Cambridge, MA. 1989.
[24] Satish M. Thatte and Jacob A. Abrahm, "Test Generation for Micropro-cessors", IEEE Trans. on Computer, vol. C-29, no. 6, pp.429-441, June. 1980.
[25] Robert Abramovitz, Kim DeVaughn, Robert Patrie, "Testability in a RISC Environments", ICCD '86, pp.142-144, 1986.

著 者 紹 介

李 文 基(正會員)

Moon Key Lee was born in Seoul, Korea, in 1941. He earned the B.S., M.S. degree and Ph.D in electrical engineering from Yonsei University, Seoul, Korea in 1965, 1967, and 1973, respectively. Also, he received the Ph.D in electronic engineering from University of Oklahoma in 1980. From 1980 to 1982 he was a Director of semiconductor design division at Korea Institute of Electronic Technology (ETRI, at present), Kumi, Korea. In 1982 he joined the faculty of Yonsei University, Seoul, Korea, where he is currently professor of electronic engineering department. He was founding Chairman of IEEE Circuits and Systems chapter in Korea from 1989 to 1991. He has served as an international committee member of IEEE Circuits and Systems Society from 1989. He is a founder of Research Institute of ASIC Design (RIAD) which was established in 1989 and located at Yonsei University. Now, he is a vice president of Korea Institute of Telematics and Electronics. His current research interests include high performance microprocessor design, high level synthesis, silicon pressure sensor, and DSP.

著 者 紹 介



李 承 浩(正會員)

Seung Ho Lee was born in Seoul, Korea in 1966. He received the B.S., M.S. degree of electronic engineering from Yonsei University in 1988 and 1990, respectively. From 1990, he has served as a staff researcher of Research Institute of ASIC Design(RIAD) at Yonsei University. He is currently in the course of Ph.D of electronic engineering in Yonsei University. He is interested in the design of microprocessor, high performance computer architecture, and now participated in the design of memory management and cache controller.

崔 炳 尤(正會員)

Byeong Yoon Choi was born in Kyeong Ju, Korea in 1962. He received the B.S., M.S. degree, and Ph.D from Yonsei University, all in electronic engineering in 1985, 1987 and 1992, respectively. From 1990 to 1993, he has served as a staff researcher of Research Institute of ASIC Design(RIAD) at Yonsei University. He is currently a professor of the department of computer engineering of Dong Eui University, Busan. His current research interests include microprocessor design and asynchronous digital circuit design.