論文 91-28B-7-3

RISC용 ALU와 시프터의 설계

(Design of an ALU and a Shifter for RISC)

崔 炳 允,* 崔 相 勳,* 李 文 基*

(Byeong Yoon Choi, Sang Hoon Choi, and Moon Key Lee)

要 約

본 논문에서는 RISC 마이크로 프로세서의 CPU 데이타 패스중 성능향상에 큰 영향을 미치는 ALU와 시프터를 설계하였다. RISC 데이타 패스는 4단의 파이프라인과 20MHz의 동작 주파수를 가진다.

ALU는 RISC에 요구되는 면적과 속도 사양에 따라 고속의 연산 특징과 규칙적인 배선 구조를 갖는 BLC 가산기를 사용하는 방식으로 구현하였다. ALU에서 지원하는 연산은 덧셈과 뺄셈의 산술 인스트럭션과 AND, OR, XOR의 논리 인스트럭션과 어레이 인덱스 등의 계산에 자주 사용된는 정수 곱셈으로 이를 효율적으로 지원하기 위해, 이를 지원하는 기능을 ALU에 추가하였다. 시프터 회로는 64-TO-32 퍼널 시프터를 구현하였으며, 시프터에서 지원하는 기능은 기존의 논리 및 살술 시프트 기능이외에 STORE 명령시 요구되는 big endian 형태의 바이트 어드레스에 따라 데이타를 정렬하는 기능도 지원하도록 하였다. 설계된 ALU와 시프터 블럭은 각각 7K개와 15K개의 트랜지스터로 구성된다. 설계된 회로에 대한 회로와 논리 시뮬레이션에 의하면 BLC 가산기를 사용하는 ALU의 경우 15.9NS, 시프터의 경우는 9.9NS의 지연시간을 갖는 것으로 나타나, 본 연구의 타겟 RISC에서 필요로 하는 사양에 만족된다.

Abstract

This paper describes the design of an ALU and a shifter for RISC. The RISC datapath is designed to have a 4-stage pipeline and a 20 MHz operating frequency. The ALU makes use of the 32-bit BLC adder which has the characteristics of high speed and regular structure and executes the arithmetic instructions-addition and subtraction- and the logical instructions-AND, OR, and XOR. Additionally, multiplication is possible by iterative executions of step instructions to perform shift and add operations. The shifter is implemented by using the modified of funnel shifter. The shifter is able to perform the arithmetic and logical shift instructions without masking. Moreover, it carries out data align operation which conforms to big endian byte address. The logical operation of the desinged ALU and the shifter were simulated using YSLOG and VLSIsim. SPICE simulation results using 1.2um double metal process parameters show that the ALU and shifter have a delay time of 15.9NS and 9.9NS, respectively. Therefore, the ALU and the shifter operates correctly above 20[Mhz] clock frequency and are composed of about 7K and 15K transistors, respectively.

*正會員, 延世大學校 電子工學科
(Dept. of Elec. Eng., Yonsei Univ.)

接受日字 : 1991年 3月 8日
(※ 이 연구는 1990년도 상공부 연구비 지원에 의한 결과임.)

I. 서 론

RISC (reduced instruction set computer) 아키텍쳐의 기본 개념은 기존의 CISC (complex instruction set computer) 아키텍쳐에 대한 성능분석을 통해,

컴퓨터 성능 향상을 위해 70년대 중반 IBM 801로 제안된 이래 학교와 기업체 연구실에서 향상을 거듭하여 지금은 다수의 상업화된 RISC칩이 사용되고 있다.[1]

RISC 마이크로 프로세서는 자주 사용되는 명령어는 빠르게 수행토록 하고, 자주 수행되지 않는 명령어는 과감히 삭제함으로써 칩 내부의 논리기능을 단순화 하고, 명령어의 단일 사이클 수행에 바탕을 두는 파이프라인 연산개념을 채택한 컴퓨터 구조이다.

RISC 아키텍쳐는 CISC 아키텍쳐에 비하여 다음과 같은 특징을 갖는다. 먼저, RISC는 CISC와 비교하면 상당히 적은 수의 인스트럭션을 가진다. 즉, 자주 사용되는 기본적인 인스트럭션만을 하드웨어에서 제공하며, 나머지는 이들의 조합으로 소프트웨어로 처리하게 된다. 또한, 인스트럭션의 형태도 매우 규칙적이고 단순하며, 제공되는 어드레싱 모드도 매우 적은 수이다. 따라서 인스트럭션의 디코딩 시간을 단축시킬 수 있다.

두번째는 load-store architecture라는 점이다. 이는 register-register architecture라고도 하는데, 수행되는 모든 연산이 load와 store 인스트럭션을 제외하면, 레지스터상에서 이루어 진다는 뜻이다. 이러한 특성으로 인하여 많은 RISC 마이크로 프로세서들은 많은 수의 레지스터를 가지고 있다. 따라서 인스트럭션의 수행에서 메모리를 직접 액세스 하는것은 load와 store뿐이다. 따라서 인스트럭션을 빠른 시간안에 수행할 수 있다.

세번째는 인스트럭션 파이프라인 처리기법의 효율적인 사용을 통해 대부분의 인스트럭션을 단일 사이클에 수행하게 된다. 단, 단일 사이클에 처리가 불가능한 load와 store 같은 인스트럭션은 파이프라인의 효율을 떨어뜨린다. 이 경우에는 파이프라인의 진행을 정지시키거나, 또는 인스트럭션버퍼의 사용으로 처리한다. 또한 곱셈과 같이 여러 사이클이 소요되는 인스트럭션의 단계 인스트럭션 (step instruction) 으로 이를 반복 실행하여 수행하게 된다.[7][8]

네번째 특징으로는 RISC가 효율적으로 동작하기 위해서는, 최적화 컴파일러가 요구된다. 그 이유는 아키텍쳐의 기능성 (functionality)이 기존의 CISC아키텍쳐상에서는 하드웨어에 의존했던 것에 반해, RISC의 경우 소프트웨어로 옮겨졌다는 점을 의미한다.

일반적으로 RISC칩의 파이프라인 클럭 주파수에 가장 큰 영향을 미치는 time critical한 데이타패스가 ALU와 시프터 회로이다. 특히, ALU는 캐리전파에 의한 지연시간으로 인해 데이터패스 중에서 큰 지연시간을 갖는 부분으로서, 시스템의 클럭 주파수를 결정하는데 가장 큰 영향을 미친다. 일반적으로 디지탈 시스템의 데이타패스 구성요소 결정은 응용 시스템의 사양에 따라 면적, 속도 및 배선 길이등을 고려하여 결정하는데, RISC의 경우 다단 파이프라인과 고속의 클럭 주파수를 통해 높은 성능을 만족시켜야 하므로, 작은 면적보다는 상대적으로 적절한 배선 조건을 만족하는 고속의 하드웨어가 바람직하다. 따라서 본 논문에서 ALU의 핵심요소인 가산기 회로로서 면적은 최소이지만 속도가 떨어지는 ripple carry 가산기는 배제했다. 단 시프터의 경우는 캐리 전파와 같은 문제가 없어서, ALU에 비해서는 빠른 시간에 연산이 이루어지므로, 속도보다는 규칙성과 면적측면을 고려하였다.

ALU의 가산기로는 manchester carry chain 가산기, binary lookahcad carry(BLC) 가산기, Carry Select 가산기 등이 일반적으로 널리 사용된다.[13] Carry Select 가산기는 기존의 rippe carry 가산기에 비해 두배 이상의 하드웨어를 제공하여 빠른 속도로 캐리를 계산하는 가산기이다. 이러한 가산기는 빠른 시간안에 캐리를 계산할 수 있다는 장점이 있는 반면, 많은 하드웨어를 필요로 하기 때문에 넓은 칩 면적을 차지한다는 약점이 있다. Manchester Carry Chain 가산기는 몇개의 비트를 하나의 블럭으로 설정하여 이들의 Carry Propagate 신호가 모두 '1' 일 경우는 이 블럭으로 입력되는 캐리를 그대로 블럭의 출력 캐리로 내보내는 방법이다. 하나의 블럭은 보통4개의 비트로 구성하는데, 이는 그보다 많은 비트를 한 블럭으로 구성할 경우는 입력 캐리를 전달 (propagate) 시키는 패스에서의 신호 감쇄가 있기 때문이다. 따라서 32비트 가산기를 구성하는 경우는 4비트로 이루어진 블럭이 8개 필요하다. 이러한 가산기의 경우는 속도와 면적의 측면에서 비교적 좋은 특성을 가지고 있으나, 전체적인 회로의 동작이 예비충전 (precharging) 기법을 요구하는 동적 회로로 구현되어야만 가능하기 때문에 회로의 설계가 어렵다. BLC 가산기는 캐리를 발생시키고, 전달하는 몇개의 기본 셀과 규칙적인 배선을 제공하는 더미 셀 (dummy cell)을 정의하여 이들의 상호 조합으로 덧셈을 행한다. 이러한 BLC 가산기는 일단 몇개의 기본적인 셀만 정의하면, 이들을 이용하여 간단히 가산기를 설계할 수 있으며, 회로의 확장은 이 셀들을 덧붙여서 쉽게 구현된다. 또한, 기본 셀의 조합으로 구성된 가산기는 정적회로로 구성되기 때문에 동작이 인정된다. 본 논문에서는 규칙적인 구조와 확장성, 속도 측면을 고려하여, ALU에 사용되는 가산기로 BLC 가산기를 사용하는 방식을 채택하였다.

기존의 마이크로프로세서의 경우 일반적으로 시프

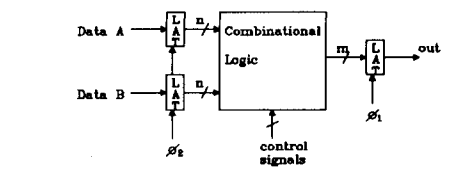
터는 소수의 거리, 즉 1비트 좌우 논리 및 산술 시프터 기능을 갖고 있다.[11] 그러나 C언어등과 같이 Field Extraction 기능이 강조되는 언어를 효율적으로 지원하기 위해, RISC칩에서는 다수 비트 거리를 단일 사이클에 쉬프트 하는 Flow-Through 형태의 배럴 시프터가 널리 사용된다.[16] 이러한 배럴 시프터 설계시 모든형태의 시프트 기능을 지원하는 Crossbar Switch의 경우 다수의 하드웨어가 요구된다. 따라서 본 연구의 타겟 RISC에서 요구되는 시프터 사양은 소수의 시프트 기능과 데이타 정렬 기능이므로, 이러한 사양을 만족시키는 시프터로, 최소의 제어회로로 규칙성이 보장되는 64-TO-32퍼널 시프터 회로를 변형하여 구현하였다. 일반적으로 MIPS,[7] MISP-X[8]와 같은 대학에서 개발된 연구용 RISC에서는 워드 단위의 데이타만을 갖고 바이트 단위의 기능은 필드 삽입과 선택 기능을 하는 전용하드웨어를 통해 지원하든지 또는 배럴시프터에서 구별된 명령으로 구현한다. 이렇게 하는 경우 하드웨어 또는 소프트웨어 지원이 추가적으로 요구되므로 상업화된 칩에서는 거의 선택치 않는다. 본 연구에서는 STORE 명령 수행시 외부 캐쉬 메모리로 나가는 데이타는 어드레스에 비해 1사이클뒤에 핀으로 나가는 점을 이용해, 메모리에 대한 어드레스는 ALU에서 계산하고 계산된 어드레스가 메모리에서 사용됨과 동시에 하위 2비트는 다음사이클에 시프터에서 이루어지는 바이트 또는 하프 워드, 워드 등의 데이타의 정렬 기능에 사용하도록 한다. 이렇게 함으로서 제안한 시프터 회로는 기존의 시프터로서의 기능에서 데이타 정렬 기능을 추가로 갖도록 함으로서 하드웨어의 효율적인 사용을 가능케 하였다.

본 논문에서는 20Mhz의 클럭 주파수를 목표로 하는 타겟 RISC에 사용될 수 있는 ALU와 시프터를 설계하였다. RISC의 데이타패스는 4스테이지 파이프라인을 가지며, 20MHz의 동작 주파수를 갖는 2-phase non-overlapping clock을 사용하였다.

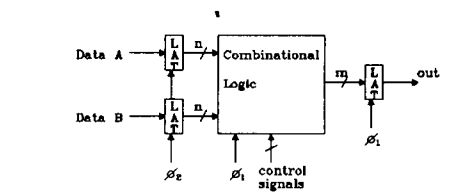
Ⅱ. RISC 데이타 패스 설계 측면에서 동적 CMOS회로와 정적 CMOS회로

디지탈 회로 설계 기법은 크게 정적회로 설계 기법과 동적 회로 설계 기법으로 나눈다. 일반적으로 정적회로 기법은 full custom CMOS 회로로 일반적으로 N-과 P-블럭이 동일수의 트랜지스터를 가지며, 조합논리 회로 부분에 클럭을 사용치 않고 제어하는 기법을 말한다. 이 방식은 다수의 면적이 필요하지만 부동(floating node)가 존재하지 않으므로 안정된 동작이 보장된다는 장점이 있다. 반면에 동적 회로 기법은 P-블럭을 클럭에 의해 제어되는 단일 PMOS 로드 트랜지스터로 대체함으로서 면적감소와 함께, 회로 동작이 상대적으로 높은 이동도를 갖는 N-block으로 이루어지는 특징으로 인해 고속의 동작이 가능하다. 그러나 부동 노드의 존재로 전하 배분 및 커플링의 문제가 생길 수 있으므로 회로 설계시 세심한 주의가 필요하다.

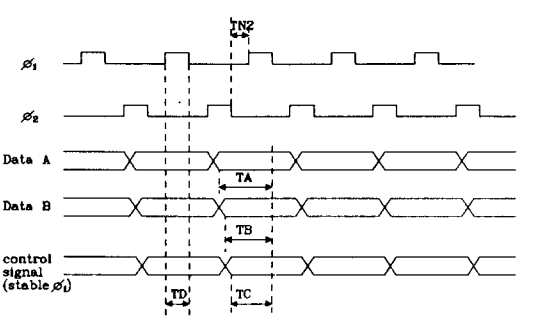
그러나 이러한 비교는 클럭 배분기법과 clock skew 등을 고려치 않은 분석이다. 본 논문에서는 RISC 설계시 플립 플롭을 바탕으로 한 설계가 아닌 래치



(a) static CMOS logic



(b) dynamic CMOS logic



T_{DL}: dynamic logic computation time
$T_{DL} = T_D$
T_{SL}: static logic computation time
$T_{SL} = \max[T_A, T_B, T_C]$
T_m: computation time difference
$T_m = T_{SL} - T_{DL} \geq T_{N2}$

(c) waveform

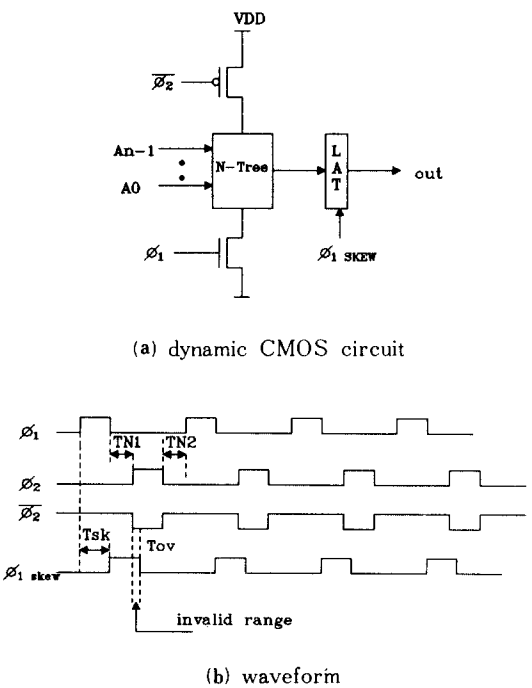
그림 1. 비중첩 2위상 클럭 시스템 측면에서 동적 동적회로와 정적회로 특성

Fig. 1. Characteristics of static and dynamic CMOS circuits under nonoverlapping 2 phase clock system.

를 바탕으로 한 설계를 따르고 있는데, ALU와 배럴 시프터 회로 설계 이전에 이러한 기법의 타당성을 검토하기 위해, 클락 기법으로 채택한 non-overlapping 2 phase clocking[5] 측면에서 동적 CMOS 회로와 정적 CMOS회로를 그림1과 같이 모델링하고 각각의 조합논리 회로에 할당되는 계산 시간측면에 대해 2가지 회로 설계 기법을 비교 분석하였다. 그림 1에 보는 바와 같이 조합논리회로에 사용되는 제어신호와 데이타가 stable ϕ_2 신호라면, 정적회로의 경우 데이타가 안정되는 즉시 연산이 시작될 수 있지만, 동적 회로의 경우 ϕ_1의 값이 1로 되는 시점부터 계산이 시작되므로 상대적으로 조합논리회로에 할당된 시간이 적다. 그러므로 최악조건을 고려하더라도 ϕ_1과 ϕ_2의 비중첩 시간(nonoverlapping time) 만큼의 시간 할당이 정적회로에 여유가 있다. 그러므로 조합논리 회로의 연산 시간이 정적회로와 동적 회로가 동일하다고 가정하면, 정적 회로가 보다 높은 클럭 주파수를 사용할 수 있음을 의미한다. 이러한 비중첩 시간은 반도체 공정 기술이 발달함에 따라 게이트 지연 시간보다 라인에 의한 지연 시간이 문제가 됨에 따라 clock skew 문제를 해결하기 위해서는 불가피하다. 그리고 그림2는 동적 회로가 clock skew 문제에 따라 오동작할 수 있는 상황을 해석하였다. ϕ_1의 skew된 신호를 ϕ_{1skew}라 하면, 이러한 신호가 ϕ_2 신호의 반전된 값과 중첩되게 되면, ϕ_{1skew}에 의해 래치되는 "out"값은 앞서 계산된 결과가 아닌 예비충전(precharging) 값을 가지게 되어 오동작을 할 수 있다. 이러한 문제는 칩 내부에서 clock skew를 최소화하는 방향으로 배선을 효율적으로 조정하면 그 효과를 감소시킬 수 있으나 완전히 해결 되지는 않는다. 따라서 본 연구의 ALU와 배럴 시프터에서는 이러한 회로 특성 비교 결과를 바탕으로 다소의 면적 증가가 있더라도 안정성을 고려하여 정적 회로 설계 기법을 채택하였다.

Ⅲ. RISC 칩의 구성

본 논문에서는 설계된 ALU와 시프터가 사용될 타겟 RISC칩의 블럭도는 그림3과 같다. RISC 칩은 크게 제어부, ALU와 배럴시프터부, 레지스터파일부, 정렬기2, 특수용도 레지스터부, 명령어 어드레스 발생부, 그리고 프로그램 카운터부로 구성된다. RISC는 명령어의 효율적인 수행을 위해 그림4와 같이 4단 파이프라인 구조를 갖고 있다. F 스테이지는 명령어를 읽어오는 파이프라인 단계이고, D 스테이지는 읽



(a) dynamic CMOS circuit

(b) waveform

그림 2. 클럭 스큐와 동적 CMOS 회로
Fig. 2. Clock skew and dynamic CMOS circuit.

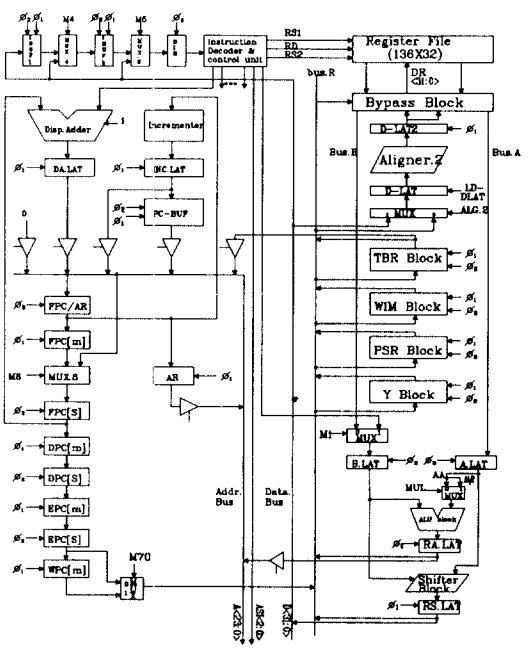


그림 3. 타겟 RISC의 블럭도
Fig. 3. Block diagram of target RISC.

어온 명령어를 디코딩하고 레지스터 화일로부터 2개의 오퍼랜드를 읽는 파이프라인 단계이며, E 단계는 ALU와 배럴 시프터가 필요한 기능을 수행하는 단계이며, W 단계는 정렬기 2를 통해 load 명령시 메모리에서 읽어온 데이타를 정렬하여 레지스터 화일에 쓰는 파이프라인 단계이다. 그리고 채택한 명령어 세트는 그림5와 같이 3가지 명령어 양식을 가지며, 46개의 명령어로 구성된다. 또한 본 연구의 RISC칩은 SPUR[6]와 SPARC[11]칩 등과 같이 lisp과 같은 인공 지능언어를 효율적으로 지원하기 위해 태그(tagged) 연산기능을 지원한다. 또한 RISC 칩은 설계중인 FPU(floating point unit)과 MMU(memory management unit)에 대한 효율적인 인터페이스도 지원하고 있다.

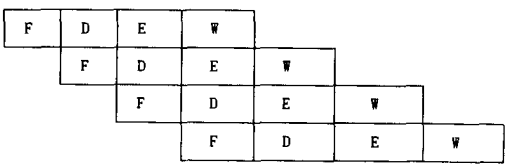


그림 4. 타겟 RISC의 파이프라인 구성
Fig. 4. Pipeline of target RISC.

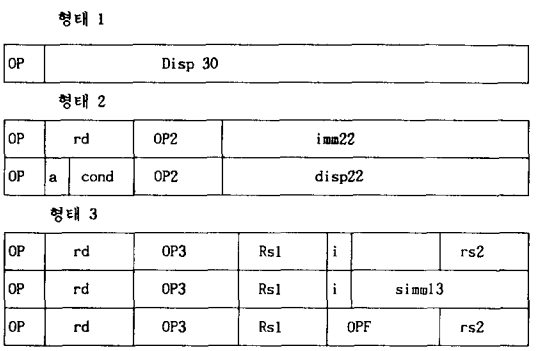


그림 5. 타겟 RISC의 명령어 형태
Fig. 5. Instruction format of target RISC.

Ⅳ. ALU 회로의 설계

그림6은 ALU와 시프터와의 연결관계 및 주변회로와의 인터페이스 신호들을 보여주고 있다. ALU 결과의 하위2비트, 즉, EA(effective address)의 LSB 2비트는 STORE 명령시에 데이터 정렬을 위한 제

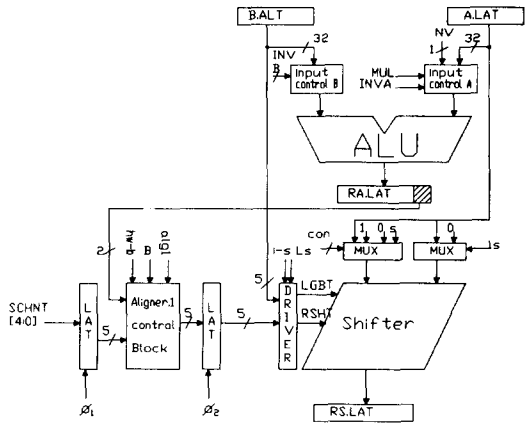


그림 6. ALU, 시프터와 주변 블럭과의 관계
Fig. 6. ALU and shifter block and its peripheral circuit.

어신호로 사용되며, 이에 대한 자세한 설명은 시프터 설계 부분에서 다루게 된다. 그림6에서 ALU의 입력에 존재하는 입력 제어 블럭은 입력 오퍼랜드 가 반전될 필요가 있는 경우와 곱셈 연산시 승수(mulitplier)의 LSB가 0일 경우, 부분곱을 출력으로 바이패스 시키는 기능도 지원한다.

ALU회로 설계시 먼저 지원해야 할 기능을 결정해야 하는데 표1은 설계된 RISC가 지원하는 기능을 나타낸다. 여기서 메모리에 대한 어드레스 계산시 ALU에서는 일반적인 덧셈을 행한다. 표1에 보는 바와 같이 덧셈과 뺄셈시 ALU의 LSB 캐리 입력으로 0, 1, condition code(CC)의 캐리 비트, 그리고 캐리비트의 반전된 값등이 사용되는 4가지 경우가 존재한다. 그림7은 ALU블럭과 주변회로를 보여주고 있다. ALU회로는 기본적으로 논리와 연산 기능을 행하는 부분과 태그 연산을 지원하는 부분, 그리고 CC를 계산하는 부분으로 크게 나눈다. 그림에서 SUBcc신호와 LOGcc신호는 각각 논리와 뺄셈연산시 CC값이 덧셈과 구별되게 동작하므로 이를 지원하기 위해 제공되는 제어신호이다. 또한 TAGcc신호, 그리고 TAGccTV신호는 태그 연산 수행시 CC의 값을 교정하기 위한 신호이다. 또한 PSR-CIN과 CABAR 신호는 각각 캐리 입력 값으로 0, 1 또는 PSR의 C_m값 또는 PSR의 C_m 값이 반전되어 사용하는지를 제어하게 된다.

ALU에 사용되는 가산기 구조로 맨체스터 캐리 체인 (manchester carry chain), 캐리 룩 어헤드(carry lookahead), BLC(binary lookahead carry)adder, 캐리 선택(carry select) 가산기 등이 고려되었다.[14][18]

표 1. ALU가 지원하는 기능
Table 1. ALU operation.

기 능	동 작	제어신호									
		INVA	INVB	ASS	MUL	ST1	ST0	PSR_CIN	CA_BAR	Y[0]	TAG_{cc}
ADD	A+B	1	1	0	0/1	1	1	0	0	X/1	0
ADDX	A+B+C_m	1	1	0	0	1	1	1	0	X	0
SUB	A−B	1	1	1	0	1	1	0	1	X	0
SUBX	A−B−C_m	1	1	1	0	1	1	1	1	X	0
AND	A & B	1	1	0	0	0	0	0	0	X	0
AND	A & B*	1	1	1	0	0	0	0	0	X	0
OR	A : B	1	1	0	0	0	1	0	0	X	0
ORN	A : B*	1	1	1	0	0	1	0	0	X	0
XOR	AB* : A*B	1	1	0	0	1	0	0	0	X	0
XNOR	A*B* : AB	1	1	1	0	1	0	0	0	X	0
TADD	A+B	1	1	0	0	1	1	0	0	X	1
TSUB	A−B	1	1	1	0	1	1	0	1	X	1
PATH_B	B	0	1	0	0	1	1	0	0	X	0
PATH_A	A	1	0	0	1	1	1	0	0	0	0

(주 : *는 반전된 값을 의미)

ALU에 사용될 가산기를 선택하기 위해 고려한 가산기의 계산효율 η는 다음의 식으로 표현할 수 있다.[18]

$$\eta = \frac{W}{I} = \frac{n}{\rho \cdot \Phi} = \frac{n}{\rho \cdot \log_2 G} \qquad (1)$$

여기서 I는 investment를 W는 계산 능력을 나타낸다. 따라서 I는 하드웨어 복잡성 Φ와 가산기의 연산 속도를 결정하는 전체 가산 시간 ρ의 곱으로 표현되며, W는 가산기의 비트수 n에 의해 측정된다. 식(1)에서 Φ는 가산기를 구성하는 게이트수 G의 로그함수, 즉 $\Phi = \log_2 G$로 표현가능하다. 2개의 입력 게이트인 AND와 OR 및 인버터만이 Sklansky의 분석에서 가정하고 있기 때문에, 로그함수에서 베이스 2를 사용한다. 식(1)과 VLSI 설계에서 강조하는 규칙성과 짧은 배선 길이등의 효과를 추가로 고려하여, ALU에 사용되는 가산기로 BLC 가산기를 채택하였다.[14]

이러한 BLC 가산기는 몇개의 기본 셀의 조합으로 구성된다. 그런데 일반적으로 알려진 BLC 가산기는 덧셈만 가능하도록 설계되어 있다. 그러나 ALU의 구성 요소가 되는 가산기는 덧셈뿐만 아니라 뺄셈과 논리 인스트럭션도 함께 수행할 수 있어야 한다. 따라서 기존의 BLC 가산기의 기본 셀을 그대로 사용할 수 없으므로, 이를 수정하는 것이 불가피하다. 이를 수정하는 방안으로 기존의 캐리 발생과 전달 (GP) 셀을, 교육용 32비트 마이크로프로세서인 OM2[17]에서 제안한 기능셀 (function cell)을 사용하는 방식을 생각할 수 있다. 그러나 이러한 기능셀은 다양한 기능 제공으로 하드웨어가 복잡하다는 문제점이 있다. 또한 본 연구의 ALU에서 필요로 하는 기능은 소수로 제한되어 있어 이 방식을 배제하였다. 두번째 방안은 기존의 GP셀에 약간의 기능을 추가하는 방안이다. 즉 기존의 BLC 가산기의 GP 셀에 존재하는 XOR 게이트와 AND 게이트 값이 논리 기능에 그대로 사용될 수 있고, 단지 OR 연산 기능만 추가하는 방안이다. 본 연구에서는 두번째 방안을 채택하여, 그림8에서 보는 것처럼 논리 연산 기능을

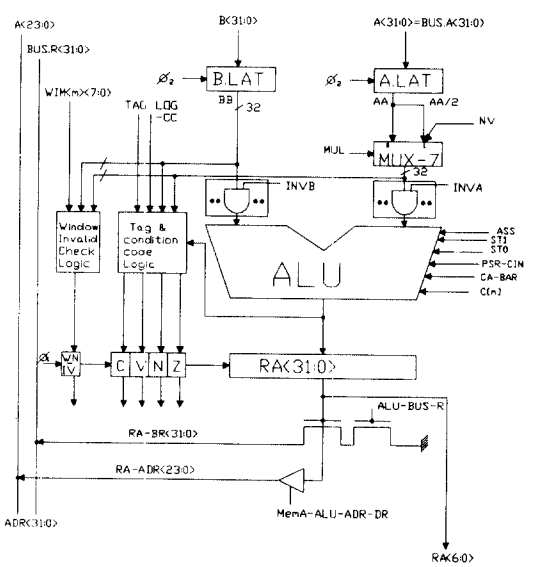


그림 7. ALU블럭 구성
Fig. 7. Block diagram of ALU.

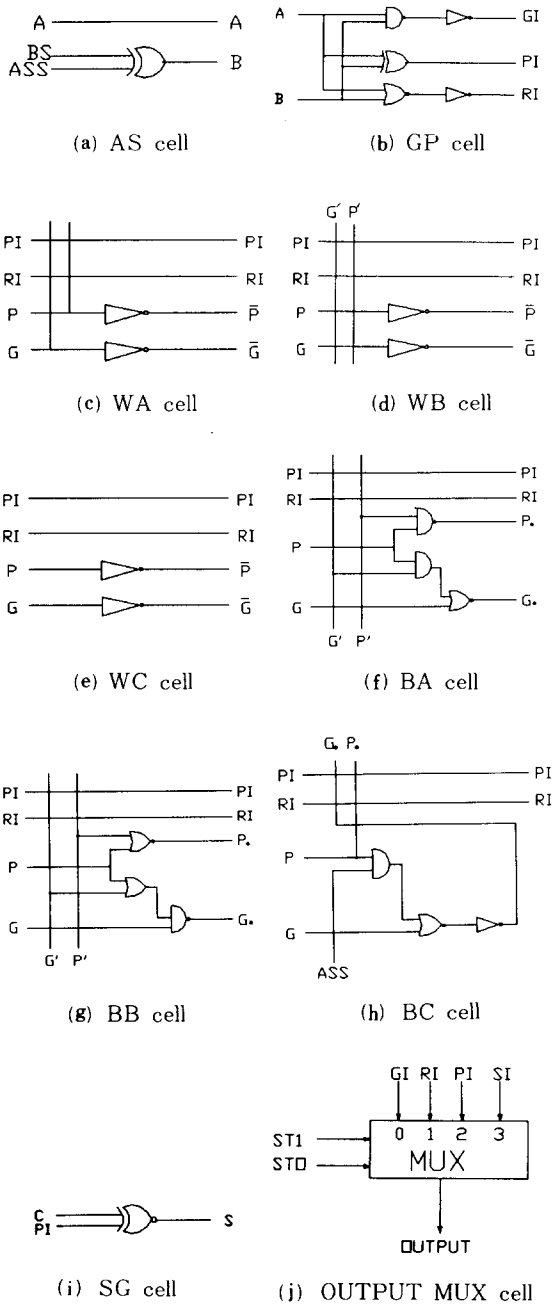


그림 8. ALU에서 사용되는 기본 셀
Fig. 8. Basic cells used in ALU.

지원할 수 있도록 기본 셀을 다시 정의하였다. 단 입력 오퍼랜드를 반전시키는 기능을 하는 AS 셀을 추가해 뺄셈과 래치 B값의 출력을 반전시키는 논리 기능을 지원한다. 그림9는 그림8에서 정의한 기본

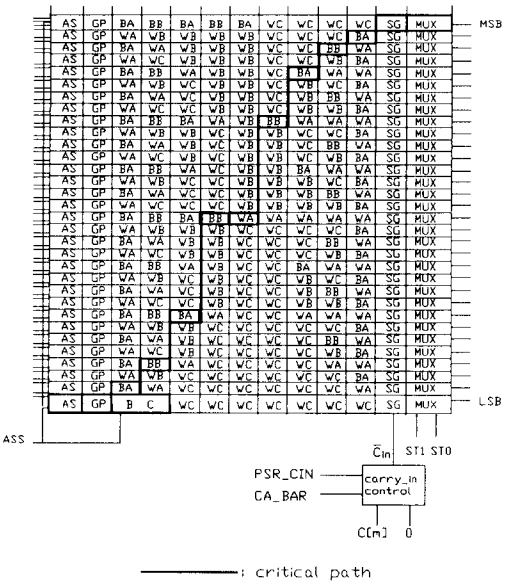


그림 9. 32비트 ALU와 기본셀 배치
Fig. 9. 32bit ALU and the basic cell placement.

셀을 배치하여 구성한 32비트 BLC 가산기를 이용한 ALU를 나타낸다. 여기서 WA와 WB셀은 규칙성을 위해 제공된 셀이 된다. 그리고 출력에 부착된 멀티프렉서 회로는 ST1과 ST0 신호를 사용하여, 최종 출력으로 BLC 가산기의 출력을 선택할 지 아니면, GP 셀에서 계산되어 출력으로 바로 전달된 논리 연산값을 선택할 지를 결정하는 회로이다. 이러한 회로에서 뺄셈 연산은 기존 하드웨어의 효율적인 이용과 2진 보수 데이타 방식의 특성에 따라, GP 셀의 입력 오퍼랜드 중 감하는 수를 반전시키고, ASS 신호를 사용하여 LSB 캐리 입력에 영향을 받는 BC 셀을 조정한다. 설계된 ALU에서는 Shift & Add 형태의 동작을 수행하는 스텝 명령어인 MULScc를 정의하고, 이러한 명령어의 반복수행을 통하여 정수 곱셈을 지원하고 있으며, 나눗셈은 지원하지 않는다. 이러한 곱셈 지원은 그림7의 ALU와 특수용도 레지스터 블락내의 Y 레지스터가 결합되어 수행하게 된다. 그림10과 같은 구조를 갖는 Y 레지스터가 승수(multiplier) 역할을 함과 동시에 A. LAT와 결합하여 부분곱(partial product)을 저장하는 역할을 하게된다. 곱셈의 수행은, 먼저 승수(multiplier)가 Y 레지스터에 담기고, 그 다음에 피승수(multiplicand)각 래치B에, 부분곱(partial product)가 래치 A에 담기게 된다. 그리고 초기에 PSR내의 CC(condition code)의 연산결과의 부호를 나타내는 N비트와 오브플로우를 나

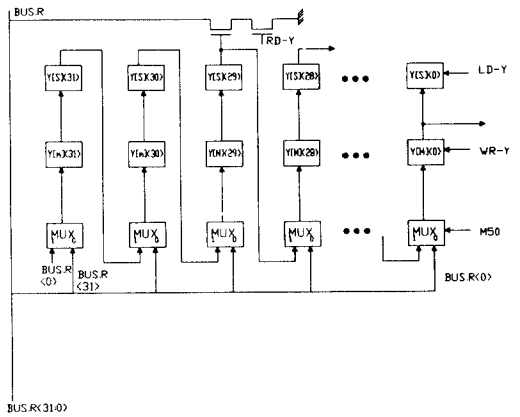


그림10. Y 레지스터의 블럭
Fig.10. Y register block.

타내는 V비트를 0으로 만드는 명령어 과정이 수행된다. 그리고 나서 단계 명령어인 정수 곱셈이 필요한 사이클 만큼 반복적으로 이루어 진다. 즉 Y 레지스터의 LSB를 체크하여 '1'일 경우는 래치A와 래치B의 레지스터의 값을 더하고, '0'일 경우는 래치A 값에 '0'을 더하게 된다. 이 경우 Y 레지스터는 곱셈 스텝 연산시 1비트씩 오른쪽으로 이동되며, 시프트로비게 되는 맨 상위 비트는 ALU의 연산 결과의 LSB값으로 채워진다. 단 곱셈 연산시 앞서 수행된 곱셈 스텝 명령어의 레지스터 파일에 대한 저장 번지 Rd를, 바로 다음에 오는 곱셈 스텝 명령어의 래치A로 읽혀지는 레지스터 화일에 대한 번지 Rs1과 동일하게 한다. 따라서 앞서 계산된 부분값이 래치A로 바이패스 되도록 하여 매번의 곱셈 스텝 연산의 시작단계에서는 래치A는 부분곱을 저장하게 되며, Y 레지스터는 부분곱과 승수(multiplier)의 일부를 갖게 된다. 그리고 원래 올바른 동작을 하기 위해서는 래치A에 저장되는 부분곱도 1비트 우측으로 쉬프트된 값을 저장해야 한다. ϕ_2페이즈 단계에 이를 지원하는 경우 기존의 바이패스 회로에 추가의 시프트 기능을 하는 회로가 필요하고, 그렇게되면 바이패스 회로에 의한 지연시간이 증가하여, 레지스터 파일의 READ시간이 증가하는 문제를 야기시킬 수 있기 때문에, 요구되는 시프트 기능을 정수 스텝 명령어의 ALU사이클의 ϕ_1에 지원하도록 하였다. 즉 곱셈연산의 부분합이 담겨있는 래치A에 있는 값을 오른쪽으로 1비트 시프트시켜 ALU 입력으로 사용하는데, 이때 가장 상위 비트는 정수 곱셈시 과도기적으로 발생하는 Overflow를 교정하기 위해 CC(condition code)의 N과 V비트의 XOR값으로 채워진다. 이러한 방식으로 곱셈의 스텝 인스트럭션을 반복 수행하여 정수 곱셈을 행하게 되며, 최종적으로는 래치A에는 64비트 결과 값중 상위 32비트가, Y 레지스터에 하위 32비트가 담기어 곱셈을 마치게 된다.

이러한 ALU를 제어하기 위한 제어신호 발생 기법은 다음과 같다. ALU의 모든 제어신호는 실제로 사용되는 E-스테이지 $\Phi 1$보다 한 스테이지 앞선 D-스테이지 $\Phi 2$에서 발생시켜, stable Φ_1신호 조건을 만족케 하였다. 단 곱셈 제어 신호의 경우에는 D- 스테이지에서 결정된 곱셈 지시신호 MUL과 Y 레지스터의 LSB비트인 Y[0]가 결합되어 ALU에 들어갈 입력값으로 래치 B값을 사용할 지 또는 0을 사용할 지를 결정하게 된다. RISC 제어회로방식으로 data-stationary hardwried control 방식을 채택하고 있는데, 그림11은 전체제어 회로중 ALU와 시프터에 대한 제어부를 나타낸다. 그림에서 Prefetch Buffer는 타겟 RISC에서 지원하는 Multi-cycle 명령 수행시 prefetch 기능을 수행한다. 제어회로의 동작을 살펴보면, F stage ϕ_2에서 명령어 레지스터 IR에 명령어가 저장되고 D stage ϕ_1에서 분산 디코딩(distributed decoding)기법에 의해 명령어 디코더(instruction decoder)에서, 기본적인 기능별로 1차적인 디코딩 작업

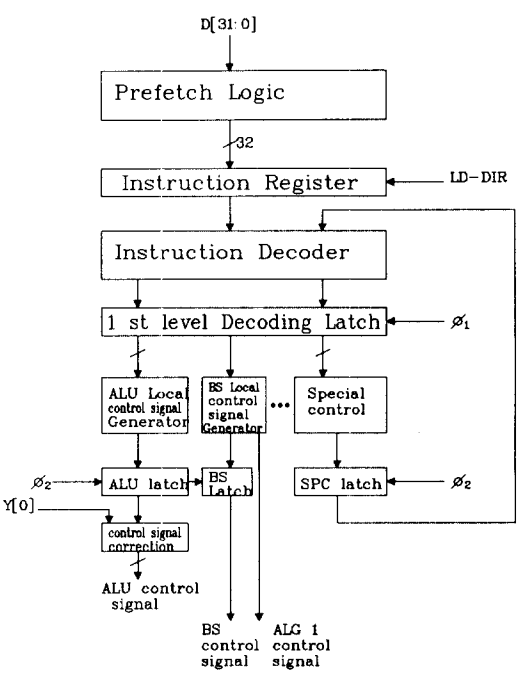


그림11. ALU와 시프터에 대한 제어 신호 발생회로
Fig.11. RISC controller for ALU and shifter.

을 수행하고, D stage ϕ_2에 세부적인 ALU 제어 신호를 발생시킨다. 이렇게 하면 명령어 디코더가 간소화되어 제어 신호 발생이 데이타패스 연산 시간에 영향을 미치지 않게 된다.

태그 연산 명령어는 LISP, smalltalk 및 prolog와 같은 태그를 사용하는 언어에 유용하다.[10][11] 이 경우 LSB비트가 태그 비트로 사용된며, 이 값은 0 으로 설정되어 있다. 나머지 30비트는 태그 데이터 이다. 태그 연산시 2개의 오퍼랜드 중에 하위 2비트중 어느 하나라도 0이 아니거나 정상적인 오버플로가 발생하면 오버플로 비트에 1을 담는다. 그림12는 CC 발생 회로에서 가장 다수의 시간이 요구될 수 있는 부분이 Z(zero)를 감지하는 회로인데 본 연구에서는 빠른 ZERO 감지를 위해, 32비트를 8비트 단위의 4개의 블럭으로 나누고 각각의 블럭에서 tri_state inverter의 순차적인 연결을 통해 빠른 시간내에 ZERO 감지작업이 이루어지도록 하였다. ZERO 감지기는 최악의 경우 ALU의 연산결과의 MSB 비트가 계산되고 나서 1스테이지의 인버터 지연후에 얻어지게 된다. 따라서 ALU의 최악 지연 경로에서 큰 시간지연을 야기시키지 않는다.

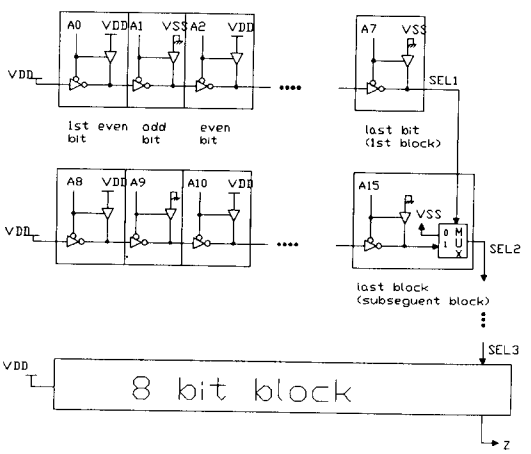
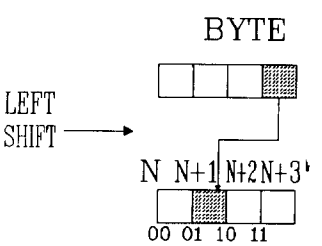


그림12. ZERO 감지회로
Fig.12. ZERO detector.

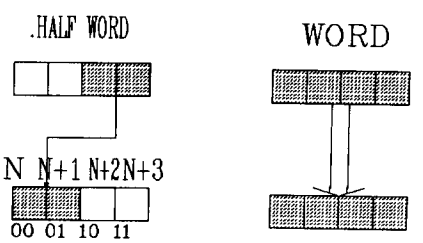
V. 데이타의 정렬 기능을 갖는 시프터의 설계

마이크로 프로세서의 CPU를 구성하는 데이타패스중 시프터는 논리연산이나 산술 연산의 경우에 널리 사용되는 시프트 기능, 부호 확장(sign-extension) 기능을 지원한다. 그리고 Big Endian 또는 Little Endian 형태의 바이트 어드레스를 지원하는 마이크로 프로세서에서 store 명령시 수행하는 데이타 정렬(align) 기능은, 일반적으로 바이트, 해프워드(half-word) 및 워드 형태로 되어있는 입력을 받아 들여서, 데이타버스의 알맞은 자리에 담기도록 정렬 (align) 하는 기능이다. 이러한 2가지 기능을 지원하기 위해 현재 상용되고 있는 RISC와 대부분의 마이크로 프로세서들은 시프트와 정렬을 수행하는데 있어서 각기 다른 하드웨어를 사용하고 있다. 즉, 시프터에서는 시프트만을 전담하고, 정렬기(aligner)에서는 정렬만을 맡아서 실행하는 기법을 채택한다.[10][11]

그러나 이러한 방식의 설계는 여분의 하드웨어가 필요해 바람직한 설계 방식이 설계가 되지 못한다. 그 이유는 여러가지 산술 및 논리 시프트 기능과 store 명령 수행시 필요로 하는 데이타 정렬 기능이 궁극적으로 유사한 시프트 작업으로 수행될 수 있기 때문이다. 그리고 본 연구의 외부 캐쉬 메모리(off-chip cache memory)를 갖는 타겟 RISC에서 store 명령 수행시, 데이타가 어드레스에 비해 1사이클 뒤에 칩 바깥으로 나가는 타이밍 특성을 이용하여, ALU회로의 지원을 받아 데이타 정렬 기능을 기존의 시프터에서 지원하는 방안을 시프터 회로 설계 사양



(a) byte align



(b) half-word align (c) word align

그림13. Big endian형태의 바이트 어드레스를 사용한 데이타 정렬 기능
Fig.13. Data align operations based on big endian byte address.

표 2. 시프터에서 수행되어야 할 연산
Table 2. Shifter operations.

기 능	동작 설명	제어 신호							
		HW_B	BB	ALG 1	LS	MC_SH	LDSTO	LOD_SH	AR_SH
SLL	논리 왼쪽 시프트	0	0	0	1	1	0	0	0
SRA	산술 오른쪽 시프트	0	0	0	0	0	0	0	1
SRL	논리 오른쪽 시프트	0	0	0	0	0	0	1	0
ST	워드 정렬후 저장	0	0	1	0	1	0	0	0
STB	바이트 정렬후 저장	1	1	1	0	1	0	0	0
STH	하프 워드 정렬후 저장	1	0	1	0	1	0	0	0
PATH_32	FFFFFFFF[H]을출력	0	0	1	0	0	1	0	0

으로 채택하였다. 이렇게 하는 경우 시프트 동작만을 처리하는 전용 시프트보다는 더 많은 제어 신호가 필요하다. 그러나 이러한 방식은 정렬기를 따로 구성했을 때의 칩 면적이나 하드웨어에 미치는 오버헤드(overhead)에 비하면 무시할 수 있을 정도이다.

이러한 데이타 정렬 기능을 포함해서 타겟 RISC에서 지원해야할 기능은 표2와 같다. 표2에서 PATH_32기능은 타겟 RISC에서 다중 프로세싱을 지원하기 위해 test & set 명령어 해당되는 명령어를 갖고 있는데, 이를 지원하기 위해 메모리의 특정 번지를 전부 1로 만들기 위해 사용된다. 그림13은 표2에서 Big Endian 형태의 바이트 어드레스를 지원하는 데이타 정렬 기능을 나타낸다. 이러한 Big Endian 바이트 어드레스는 워드의 최상위 바이트가 가장 낮은 이트 어드레스를 갖는다. 이러한 데이타 정렬 기능을 ALU에서 계산되는 데이타 메모리에 대한 어드레스(EA : effective addres)의 LSB 2비트값에 따라 메모리로 전송될 데이타가 정렬된다. 표2에 기술된 기능을 지원하기 위한 데이타 정렬 기능을 갖는 시프터 회로를 설계하기 위해, 3가지 방식의 시프트 방식을 고려하였다.

첫번째 방안으로 다양한 시프트 기능을 지원하는 크로스바 스위치(crossbar switch)를 이용하는 방식이다.[19] 이러한 형태로 구성된 시프터는 이동시켜야 할 거리에 관계없이 빠른 시간안에 시프트를 할 수 있다는 잇점이 있다. 그러나 이 방식의 경우 필요한 제어 신호의 갯수는 입력의 수에 따라 다음식과 같이 결정된다.

$$C=2\times(N-1)+1 \qquad (2)$$

단, C는 제어신호의 갯수
N은 입력신호의 갯수

또한 시프터가 차지하는 면적도 입력 신호의 수가 증가함에 따라, 기하급수적으로 증가하게 된다.

두번째 방안은 일방향의 시프트 기능과 로테이트(rotate) 기능을 가지는 32-TO-32 배럴 시프터(barrel shifter)를 사용하는 방법이다.[16] 이 방식은 일방향의 시프팅 기능을 지원하는 약점을 보완하기 위해, 로테이트 연결관계와 일방향의 시프트 기능을 결합하여 반대방향의 시프트도 지원할 수 있게 한다. 이러한 배럴 시프터를 사용할 경우에는 필요한 제어 신호의 갯수는 식3과 같다.

$$C=\mathrm{Log}_2 N+1 \qquad (3)$$

단, C는 제어신호의 갯수
N은 입력신호의 갯수

그러나 이러한 배럴 시프터에서는 sign extension, 또는 논리 시프트등을 지원하기 위해서는, 시프트로 인해 비게되는 위치에 바람직한 데이타를 제공하기 위한 마스크 발생회로(mask generator)가 필요하다. 이러한 마스킹 기능을 효율적으로 지원하는 기법으로 출력 포트에 sign extension과 zero extension을 지원하는 패스 트랜지스터를 제공하는 방법도 가능하지만, 최대31비트의 패스 트랜지스터를 거쳐야 하므로, 신호 지연등의 문제를 야기시키므로 바람직하지 못하다. 또한 본 연구에서 시프터 사양에서 요구하는 데이타 정렬 기능을 효율적으로 만족시키지 못한다는 문제가 있다.

세번째 방법은 배럴 시프터 회로에서, 특히 field extraction 기능을 강화한 방식으로, 기존의 32-TO-32 shift network가 아닌 64-TO-32 shift network을 사용하는 퍼널 시프터(funnel shifter)를 이용하는 방법이다.[17] 이러한 방식은 입력 port가 앞서의 2가지 방식에 비해 2배로 늘어났기 때문에, 산술 및 논리 시프터 기능에서 요구되는 sign extension과 zero extension 기능을 효율적으로 지원할 수 있다. 또한 field extraction 기능으로 2개의 32비트 워드에서 각 워드의 일부분을 선택해서 출력시킬 수 있는 기능도

쉽게 지원 가능하다. 이 방식의 경우 필요한 제어 신호는 식4와 같다.

$$C = \mathrm{Log}_2 N + 1 \qquad (4)$$

단, C는 제어신호의 갯수
N은 입력신호의 갯수

이 방법은 입력의 수가 64개이기 때문에 배럴 시프터를 응용한 방법보다 면적은 많이 차지하나, 데이타 전달 패스에 tri-state buffer를 사용하는 정적 회로 설계 기법을 사용하였기 때문에, 패스 트랜지스터를 사용하는 경우 야기되는 전하 커플링 또는 신호 크기 감소 등의 문제가 없으므로 회로의 동작이 안정된다. 또한 마스크 발생 유니트와 로테이트 경로가 필요없다. 표3은 3가지 방식에 대한 비교를 나타낸다.

표 3. 3가지 시프터 회로 비교
Table 3. Characterisitcs comparison of 3 types of shifter.

Shifter Network / 항목	Crossbar Shifter Network	Barrel Shifter Network	Funnel Shifter Network
제어신호수	2(N−1)+1	$\log_2$N+1	$\log_2$N+1
입력 비트수	N	N	2N
출력 비트수	N	N	N
shift stage수	1	$\log_2$N+1	$\log_2$N+1
mask회로 여부	No	Yes	No
정렬 기능지원	No	No	Yes

본 논문에서는 표3의 평가를 통해, 데이타 정렬 기능을 효율적으로 지원하며, 안전한 회로 동작이 보장되는 funnel shift network을, 일반적인 시프트기능과 데이타 정렬기능을 갖는 시프터회로의 shift network으로 선택하였으며, 이의 자세한 연결관계는 그림14와 같다. 그림14를 보면 시프트 네트워크는 6개의 스테이지로 구성되며, 6개의 제어신호 LGBT, RSH0, RHS1, RSH2, RSH3, RSH4로 각 스테이지의 시프트를 결정하게 된다.

그러나 기존의 퍼널 시프터(funnel shifter)는 데이타 정렬 기능을 지원치 않기 때문에, 본 연구에서 제안한 시프터 회로는 이를 지원하기 위해, 2개의 입력 포트에 추가의 멀티프렉서 회로를 두고, 수행하여야 할 연산에 따라, 적절한 입력값을 시프트 네트워크로 제공하는 방식을 채택하였는데, 그림15는 각각의 연산에 대한 입력 포트의 입력값 할당을 나타낸다. 기본적으로 논리 시프트 경우 포트A에 00000000(H)를

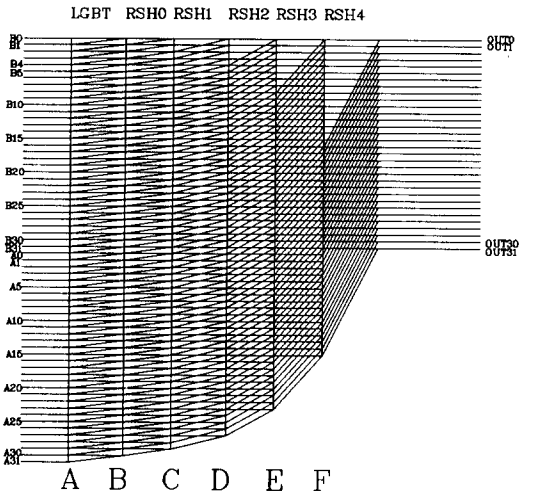


그림14. 64−TO−32 퍼널 시프트 네트워크
Fig. 14. 64−TO−32 funnel shifter network.

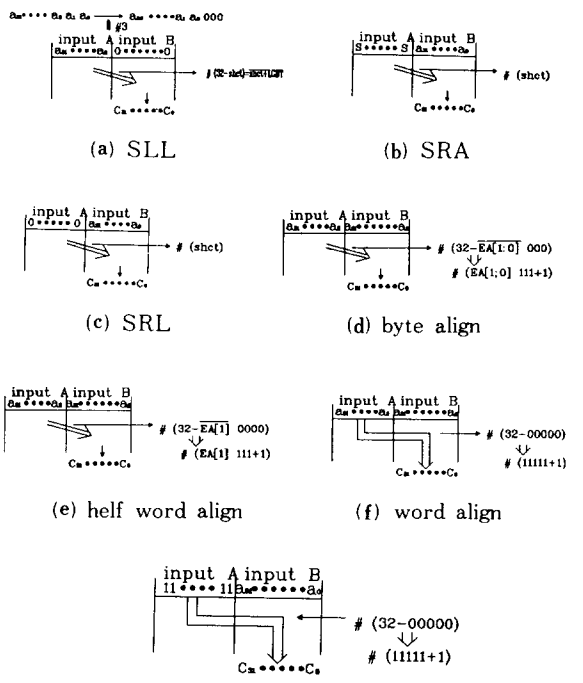


그림15. 연산에 따른 퍼널 시프터의 입력 포트에 입력 할당 기법
Fig. 15. Input data assignment for funnel shifter according to operation.

할당하고, PATH-32의 경우는 포트A에 FFFFFFFF(H)를 할당하게 된다. 그림16은 설계한 시프트 블럭 다이아그램을 나타낸다. 그림16에서 정렬기1 제어부분은 store 명령에 관계된 정렬 제어시, ALU에서 계산된 EA[1:0]가 SCHNT래치로 로드되어 시프트 동작을 제어하는 역할을 수행한다. 데이타 정렬기능을 지시하는 ALG 1제어 신호에 따라, ALG=0이면 명령어에 들어있는 SCH[4:0] 값이 전달되고 ALG1=1이면 EA[1:0]값이 변형되어 E-스테이지 ϕ2에서 SCHNT 래치에 담긴다. BS Driver회로는 시프트 네트워크에 사용될 제어신호를 발생시키는 부분이다. 내부 동작을 살펴보면, 인스트럭션에 내장된 선택신호값인 I의 값에 따라 I=0이면 R[rs2] 즉 B래치 값의 하위 5비트가 선택되고, I=1이면 인스트럭션에 내장된 값으로 SCHNT 래치에 담긴값이 선택되었다가, 시프트 방향을 지시하는 LS 제어신호에 의해 선택된 값이 그대로 시프트 제어식호로 사용될지 아니면, 그것의 2의 보수값이 사용될 지가 정해진다. 즉 LS=0이면 오른쪽 이동 명령어가 되며, LS=1이면 왼쪽 이동에 해당한다. LGBT의 값은 LS=1 또는 ALG 1=1 인 조건시에 1이 된다. 이것은 본 논문의 시프터가 오른쪽 이동만을 지원하고 있기 때문에, 왼쪽 이동 명령어는 입력포트 A에는 래치 A값을 선택하고, 입력 포트 B에는 00000000(H)를 선택한 후, 제어신호의 보수를 취한 다음 1을 더한 양만큼 오른쪽으로 시프트 시키면, 왼쪽방향으로 원하는 양만큼의 이동시키는 것과 동일한 효과를 얻을 수 있다는 기법을 사용하여 지원하였다. 이렇게 하면 기존의 배럴 시프터와 같은 로테이트 경로가 필요없게 된다. 단 여기서 1을 더하는 연산을 5

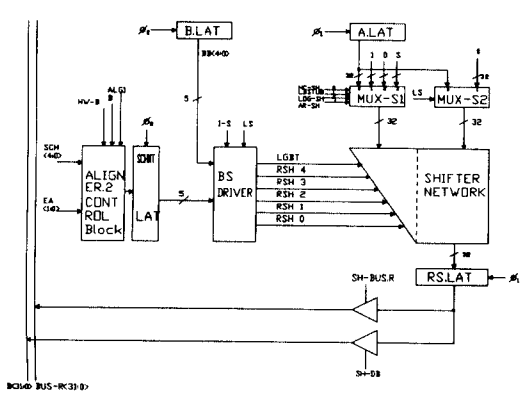


그림16. 시프터의 블락 다이어그램
Fig. 16. Block diagram of a shifter.

비트 가산기로 구현하는 경우, 최대 5비트의 캐리 전파시간이 요구되므로, 이를 방지하고 1을 더하는 것과 동일한 효과를 얻기 위해 LGBT 신호에 의해 통제되는 1비트 시프트를 위한 스테이지를 추가하였다. 설계한 시프터 회로는 6개의 제어신호로 6 스테이지 동안 0에서 32비트까지의 오른쪽 시프트를 수행할 수 있다. 여기에서 오른쪽 방향 쉬프트 양은 다음과 같은 식으로 표현 가능하다.

$$S=\sum_{k=0}^{4} S_k 2^k + LGBT \qquad (5)$$

여기서 LGBT 값은 데이타 정렬과 왼쪽방향 시프트에 대해서는 1이 되어야 하며, 오른쪽 방향 시프트에 대해서는 0이 되어야 한다. 그림14에서 LGBT 신호는 첫번째 스테이지에 대한 제어신호에 해당되며, RSH[i]는 i+1 번째 스테이지에 대한 제어신호이다. 각 스테이지 사이의 연결 구현은 패스 트랜지스터를 사용하는 방식과 Tri-state Buffer를 사용하는 방식을 고려할 수 있는데, 6스테이지가 직렬로 연결되는 관계로 회로의 안정성을 고려하여 2번째 방식을 채택하였다.

시프터에 대한 제어 신호 발생회로는 그림11에 보는 바와 같이 ALU 제어 회로와 유사하지만, ALU 1 제어 블락의 경우는 D stage ϕ_2에서 제어되므로 이러한 제어신호는 RISC 제어회로에서 2번째 디코딩 단계에서 단순한 게이트로 구성된 디코딩 회로를 거쳐 직접 발생된다.

Ⅵ. 시뮬레이션 및 결과 고찰

본 논문에서 설계한 ALU와 시프터는 YSLOG (ynsei logic simulator), VLSI$_{sim}$[22]과 범용 회로 시뮬레이터인 SPICE를 이용하여 논리 시뮬레이션과 회로 시뮬레이션을 수행하였다. 본 논문에서 사용하는 시뮬레이션 방식은 그림17과 같이 노드(node)에 의한 상호 연결 관계를 나타내는 netlist 파일을 생성한후, 이에 대한 시뮬레이션화일과 link시켜 시뮬레이션하는 방식을 사용하였다. 먼저 YSLOG와 VLSI$_{sim}$을 이용한 논리 시뮬레이션은 ALU의 경우, 가산기와 두개의 입력 레지스터, 한개의 출력 레지스터, 곱셈을 지원하기 위한 Y레지스터, 그리고 CC를 저장하는 레지스터를 포함한 회로에 대해 논리동작을 확인하였다. 시프터의 논리 시뮬레이션은 시프트 네트워크, 드라이버와 주변의 입출력 MUX를 포함하여 SLL, SRA, SRL 그리고 데이타 정렬 기능등에 대해 올바른 동작을 확인하였다.

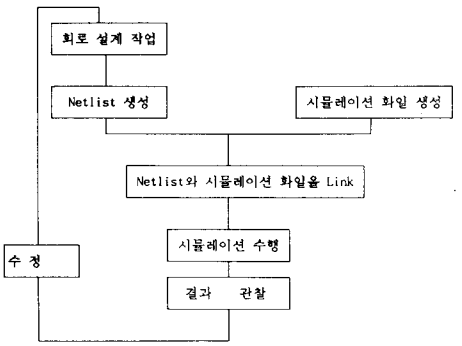


그림17. 시뮬레이션의 흐름도
Fig. 17. Flow of simulation.

SPICE를 이용한 회로 시뮬레이션은 각각의 회로가 가지고 있는 최악 지연 경로에 대하여 수행하였다. BLC 가산기를 이용한 ALU에 대한 회로 시뮬레이션은 모든 셀을 붙인 32비트의 전체 회로중 그림9의 최악 지연경로를 찾아 이를 4개의 블럭으로 나누어 시뮬레이션 하였는데, 그 결과 얻어진 최악 지연시간은 15.9[ns]이었다. 시프터는 주변 블럭을 포함한 최악 최악 지연 경로는 시프트 네트워크의 6스테이지와 입력 멀티플렉서로 구성되어, 전체 회로에서 속도 측면에서는 크게 문제가 없는 부분이다. 따라서 안정된 동작과 면적 최소화 방향으로 구현하였다. 본 논문에서 사용한 퍼널 쉬프터는 BS Driver와 입력 멀티플렉서 지연시간을 포함한 최악 지연시간은 9.9[NS]이다. 이러한 SPICE 시뮬레이션 결과로 볼때, 설계한 ALU와 데이타 정렬 기능을 갖는 시프터는 타겟 RISC에서 요구되는 20[Mhz]의 타이밍 조건을 만족함을 알 수 있다. 회로 시뮬레이션에 사용된 공정 파라미터는 NMOS의 경우 1.2um, PMOS의 경우 1.6um의 게이트 길이를 가지는 파라미터를 사용하였다.

설계된 ALU와 시프터와 타겟 RISC칩의 트랜지스터 갯수와 차지하는 면적은 표4와 같다. 이 값은 설계한 RISC의 전체 트랜지스터의 갯수가 대략 25만개인 점에 비추어 볼때, ALU와 시프터는 각각 3%와 6.5%를 차지한다.

그림18은 전체 RISC의 레이아웃을 나타낸다. 전체 칩 크기는 8.5mm×8.5mm 정도이다. 표5는 기존의 RISC인 RISC II[2], SOAR[10] SPUR[6] SPARC[11] MIPS-X[8]에서 사용한 데이타 패스와 본 연구의 타겟 RISC의 데이타패스인 ALU와 시프트의 특성 비교를 나타낸다. 표5를 살펴보면, 각각의 RISC는 각

표 4. ALU와 시프터의 트랜지스터 수
Table 4. Number of transistors in the ALU and the shifter.

블럭 명칭	트랜지스터 수(%)
ALU 블럭	7,166(3)
쉬프터 블럭	15,766(6.5)
전체 트랜지스터 수	243,672(100)

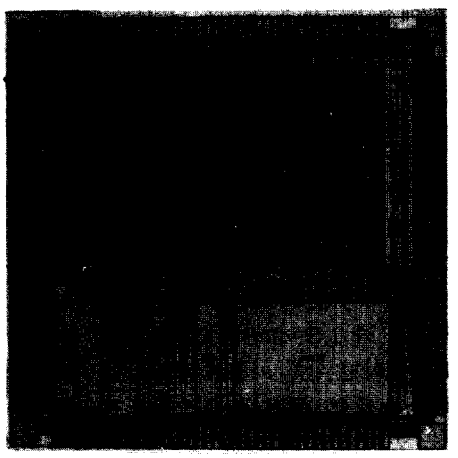


그림18. 전체 RISC의 레이아웃
Fig. 18. Layout of RISC.

각의 응용 분야에 따라 다른 사양에 맞추어 독립적으로 설계되었기 때문에 절대적인 비교가 어렵지만, 본 연구의 타겟 RISC에 맞추어 설계된 ALU와 시프터 회로는 상대적으로 빠른 속도와 다양한 기능을 지원하고 있음을 알 수 있다. 또한 본 논문에서 설계한 ALU와 시프터 회로는 다양한 기능을 갖고 있고 융통성이 있는 구조를 갖고 있기 때문에 유사한 사양을 가진 RISC에 약간의 회로 수정만으로도 응용 가능하리라 판단된다.

Ⅶ. 결 론

본 논문에서는 20MHz의 동작 주파수와 4단의 파이프라인 스테이지를 가지는 RISC 마이크로 프로세서의 데이타 패스중 ALU와 시프터를 설계하였다.

ALU는 덧셈과 뺄셈, 그리고 AND, OR, XOR의 논리 인스트럭션을 수행할 수 있도록 설계하였다. 또한, 어레이 인덱스 등의 계산에 널리 사용되는 정수 곱셈을 효율적으로 지원하기 위해, Shift & Add 형태의 스텝 명령어를 정의하고, 이러한 명령어의 반복

표 5. 타겟 RISC의 ALU, 시프터와 기존 RISC와 특성 비교

Table 5. Characteristic comparisons of the ALU and the shifter among conventional RISC and target RISC.

항목 RISC	Cycle Time[ns]	ALU 구조 및 특성	시프터 구조 및 특성	공정 기술
RISC II	500	·Dynamic ripple carry adder	·Crossbar shifter ·Data align support	4μm NMOS
SOAR	400	·Carry Skip Adder (4Bit/black)	·Crossbar shifter ·Byte inserter + Byte extracter	4μm NMOS
SPUR	100	·Partial Carry Lookahead Adder (8bit/block)	·Crossbar shifter ·Byte inserter + Byte extracter	1.6μm CMOS
SPARC	30	·Carry Lookahead Adder	·Crossbar shifter ·dedicated data aligner	0.8μm CMOS
MIPS-X	50	·Carry Skip Adder+ Manchester Carry Chin Adder	·64-To-32 Funnel Shifter ·No data align support	1.5μm CMOS
타겟 RISC	50	·32 bit BLC Adder	·64-To-32 Funnel Shifter ·data align support	1.2μm CMOS

수행을 통해 정수 곱셈을 지원할 수 있도로 하는 하드웨어를 ALU에 추가하였다. ALU의 가산기로는 RISC에서 요구하는 고속의 특징과 배선의 규칙성을 고려하여, BLC 가산기를 사용하였다. 단, ALU 에서 필요로 하는 덧셈이외의 기능을 지원하기 위해, 기존의 BLC 가산기에 사용되는 몇개의 기본셀을 변형하고, 규칙적인 배선을 위한 더미셀(dummy cell)을 정의하여, 이들의 조합으로 다양한 ALU 연산을 지원하도록 하였다. 그리고 논리 연산과 산술 연산을 분리시킴으로서, 논리 연산이 캐리 전달로 다수의 시간이 요하는 산술 연산의 시간 지연에 영향을 주지 않도록 설계하였다. 이러한 회로는 안정된 동작을 위해, 정적 논리 회로 기법으로 구성하였다.

시프터의 경우는 기존의 배럴 시프터를 개선한 64-TO-32퍼널 시프터 회로에서 데이타 정렬 기능을 추가로 지원하게 설계하였다. 이러한 시프터는 2개의 32비트 입력 포트와 1개의 32비트 출력 포트, 2개의 입력 멀티프렉서, BS driver로 구성된다. 설계된 시프터는 입력 포트로 들어가는 입력값의 효율적인 선택을 통해 다양한 기능을 지원할 수 있으며, 마스크 발생 회로와 로테이트 경로가 필요없기 때문에 효율적인 하드웨어 사용 특성을 갖는다. 또한 기존의 시프터에서 지원하는 논리와 산술 시프트 기능이외에 store 명령시 big endian형태의 바이트 어드레스에 따라 메모리로 보내어지는 데이타에 대해 효율적인 데이타 정렬 기능을 지원해 준다. 이러한 데이타 정렬 기능 지원으로, 전용 데이타 정렬 회로를 사용하는 방식에 비해 면적 감소와 성능 향상을 기대할 수 있다. 즉 하드웨어의 효율적인 이용으로 남게되는 면적을 성능 향상에 큰 영향을 미치는 레지스터 파일이나 캐쉬 메모리 설계에 할당할 수 있다. 설계된 ALU와 시프터는 전체 RISC의 트랜지스터 갯수에서 각각 3%와 6.5%를 차지한다. 설계된 ALU와 시프터는 논리 시뮬레이션을 통해 올바른 동작이 이루어짐을 확인하였으며, ALU와 시프터의 최악 지연 시간은 SPICE 시뮬레이션 결과 15.9[ns]와 9.9[ns]이었다. 따라서 설계된 회로는 타겟 RISC에서 요구하는 20[Mhz]의 조건을 만족함을 알 수 있었다. 설계된 회로는 융통성있는 구조와 다양한 기능 지원 특징을 가지고 있기 때문에 유사한 사양을 갖는 RISC에 약간의 회로 수정으로 쉽게 적용될 수 있다.

參 考 文 獻

[1] 이문기, "Multiprocessing을 위한 RISC 마이크로프로세서 설계," '90 프로세서 설계 초청 워크샵 발표 자료집, 대한 전자공학회, pp. 35-83, June. 1990.

[2] Manolis G.H. Katevenis, Reduced Instruction Set Computer Architecture for VLSI, The MIT Press, 1984.

[3] David A. Paterson, Carlo H. Sequin, "A VLSI RISC," Computer , pp. 8-22, September 1982.

[4] William Stallings, Reduced Instruction Set Computer, IEEE Computer Society Press, pp. 57-81, 1986.

[5] Noice David Cooke, A Clocking Displine for Two-phase digital Integrated Circuits, Stanford Univ. Ph.D Dissertation 1983.

[6] M. Hill, D.A. Patterson, "Design decision in SPUR," Computer, vol. 19, no. 10, pp. 8-22, October 1986.

[7] Steven A. Przybylski, Thomas R. Gross, John L. Hennessy, Norman P. Jouppi, Christopher Roewen, Organization and

VLSI Implementation of MIPS, Stanford University Technical Report No. 84-259, April 1984.

[8] Paul Chow, The MIPS-X RISC Microprocessor, Kluwer Academic Publishers, 1989.

[9] 최상훈, 최병윤, 손승일, 이문기, "RISC용 ALU의 설계,"대한전자 공학회 추계 학술 발표 논문집, pp. 529－533, 1989.

[10] David Michael Ungar, The Design and Evaluation of a High Performance Smalltalk System, The MIT Press, 1985.

[11] M. Namjoo and F Abu-Nofal, "CMOS Custom implementation of the SPARC Architecture," Compcon, 18-20, 1988.

[12] 손승일, 최병윤, 최상훈, 김의규, 고동범, 김봉열, 이문기, "YU-RISC용 32비트 퍼널 쉬프터의 설계,"한국 통신학회 추계 학술발표 대회 논문집, pp. 375－378, 1989.

[13] Robert W. Sherburne Jr., Manolis G.H. Katevenis, David A. Patterson, Carlo H. Sequin, "Datapath Design for RISC," 1982 Conference on Advanced Research in VLSI, M.I.T. 1982.

[14] Neil Weste, Kamran Eshiraghian, Principles of CMOS VLSI Design, A Systems Perspective, Addison Wesley.

[15] Peter M. Kogge, The Architecture of Pipelined Computers, Advanced Computer Science Series.

[16] Sung Mo Kang, "Domino CMOS Barrel Switch for 32-Bit VLSI Processor," IEEE Circuits and Device Magzine, pp. 3-8, May. 1987.

[17] Carver Mead and Lynn Conney, Introduction to VLSI System, Addison-Wesley, 1980.

[18] Hai Hwang, Computer Arithmetic-Principle, Architecture, and Design, John Wiley & Sons, 1979.

[19] Amar Mukherjee, Introduction to nMOS & CMOS VLSI, Prentice Hall, 1986.

[20] VLSI & CAD 연구실, "YSLOG USER'S Manual,"연세대학교 전자공학과.

[21] VLSI Technology Inc. "Tming Verifier, Mixed Mode Simulator (VLSIsim)" ASIC Division, 1989.

著 者 紹 介

崔 炳 允 (正會員) 第25卷 第9號 參照
현재 연세대학교 대학원 전자공학과 박사과정 재학중

●

李 文 基 (正會員) 第25卷 第9號 參照
현재 연세대학교 전자공학과 교수

崔 相 勳(正會員)
1967年 3月 5日生. 1989年 2月 연세대학교 전자공학과 졸업. 1991年 2月 연세대학교 대학원 전자공학과 졸업. 석사학위 취득. 현재 현대전자 반도체연구소 근무중. 주관심분야는 마이크로 프로세서, 마이크로 콘트롤러 및 ASIC 설계임.