論文2001-38CI-6-8

# 휴대 단말기용 32 비트 RISC 코어 구현

# (Implementation of a 32-Bit RISC Core for Portable Terminals)

丁 甲 天 *, 朴 性 模 **

(Gab Cheon Jung and Seong Mo Park)

요 약

본 논문은 셀룰러 폰, PDA, 노트북 등과 같은 휴대 단말 시스템에서 내장형으로 사용될 수 있는 32비트 RISC 코어 구현에 대해서 기술하였다. RISC 코어는 ARM®V4 명령어 셋을 따르며 전형적인 5단 파이프라인으로 동작한다. 또한 보다 향상된 코드 밀도를 위해 Thumb 코드를 지원하고, 파이프라인 레지스터의 동적 전력 관리 기법을 사용한다. RTL 수준에서 VHDL로 모델링된 코어는 ADS의 ARMulator와 비교 검증되었으며 평균 CPI는 1.44이다. 검증이 완료된 코어는 0.6㎛ CMOS 1-poly 3-metal 셀라이브러리를 사용하여 합성 및 레이아웃되었으며 크기는 약 41,000 게이트이고, 예상 동작주파수는 45 MHz이다.

**Abstract**

This paper describes implementation of an embedded 32-Bit RISC core for portable communication/information equipment, such as cellular phones, PDA(Personal Digital Assistants), notebook, etc. The RISC core implements the ARM®V4 instruction set, operates with typical 5-stage pipeline. It supports Thumb code to improve the code density, and uses the dynamic power management method of pipeline registers. It was modeled and simulated in RTL level using VHDL, and verified with ARMulator of ADS(Arm Developer Suite) and had average CPI of 1.44. The core is synthesized automatically using the cell library based on 0.6㎛ CMOS 1-poly 3-metal CMOS technology. It consists of about 41,000 gates and the clock frequency is expected to be above 45 MHz.

## I. 서 론

최근 휴대 단말 시스템들의 사용이 대폭적으로 증가함에 따라 전력소비를 감소시키는 것은 중요한 설계목적이 되었다. 휴대 단말 시스템은 제한된 전력 공급원인 배터리로서 구동되기 때문에 속도와 성능을 유지하면서 전력을 최소화할 수 있어야 한다. 또한 멀티미디어의 발달로 영상, 음성 등의 데이터 압축/신장, 음성 인식/합성, 그래픽스 처리 등 이전의 휴대 단말 시스템과는 비교할 수 없을 정도의 높은 성능이 필요하다.

휴대 단말 시스템에서 사용되는 프로세서는 RISC 코

* 正會員, 全南大學校 電子工學科
(Dept. of Electronics Eng., Chonnam National Univ.)
** 正會員, 全南大學校 컴퓨터工學科
(Dept. of Computer Eng., Chonnam National Univ.)
※ 본 논문은 전남대학교 학술연구비 지원에 의하여 연구되었음
※ This paper was partially supported by IDEC(IC Design Education Center)
接受日字:2001年5月2日, 수정완료일:2001年7月20日

어를 CPU로 사용하고 캐시, 메모리 관리 유닛, 타이머, 시리얼 I/O, 버스 인터페이스 등 많은 주변기능 블록들을 코프로세서화하여 단일 칩에 집적하는 SOC (System-On-Chip)형태를 가진다. 이러한 SOC 내에서 내장형으로 사용되어지는 RISC 코어는 전력소비가 적어야 하며 다양한 응용 소프트웨어를 수행할 수 있는 마이크로프로세서의 능력이 요구되어진다.

시스템에서 소모되는 전력을 살펴보면 소비 전력은 칩에 전원이 들어올 때마다 존재하는 주파수와 독립적인 정적(static) 전력과 노드값이 변하지 않는다면 전력이 소모되지 않는 주파수 의존적인 동적(dynamic) 전력으로 구분되어진다. 대부분의 전력 최소화 기술은 동적회로의 스위칭 커패시턴스를 최소화하여 동적 전력을 감소하는데 중점을 두고 있다. 동적 전력을 감소하기 위한 방법으로는 시스템 레벨 전력 관리와 동적 전력 관리로 구분된다.

시스템 레벨 관리는 프로세서 내에 전력 관리 유닛을 두어 일정시간에 구동되지 않은 장치에 전원 공급 및 클럭을 중지시키는 방법으로 칩과 모듈 레벨에서 이전부터 널리 적용되어왔으며, 설계된 프로세서의 휴면 상태에 따라 휴지(Idle) 모드, 수면(Sleep) 모드 외 여러 모드의 파워다운(Power down) 모드로 구성되어진다. 이와 같은 파워다운 모드의 상황은 운영체제에 의해 발견되고, 운영체제는 프로세서 동작을 멈추도록 한다[1][2].

동적 전력 관리는 프로세서의 동작하지 않은 주기들을 발견하고, 그 주기동안 동작하지 않는 회로의 전원 또는 클럭신호를 turn off 시키는 관리 방법이다. 이것은 몇몇의 입력 조건이 만족되었을 때 레지스터 load-enable 신호를 사용하거나, 클럭을 gating함으로써 입력/상태 레지스터들을 disable하는 방법으로, 순차회로들에서 스위칭을 감소시키는데 매우 효율적인 관리방법이다. 매 클럭 싸이클마다 하드웨어의 shutdown이 결정되므로 입력/상태 레지스터의 입력조건들을 식별할 수 있는 부가적인 회로가 추가된다. 이러한 동적 전력 관리 기법은 gated 클럭 FSM 접근[3], n개의 중첩되지 않는 다중 클럭 사용[4], FSM의 decomposition[5], pre-computation 기반 전력 감소[6] 등의 여러 가지 관리기법이 제안 및 사용되어왔다.

본 논문에서는 높은 코드 밀도를 위한 Thumb 코드 지원과 로직 레벨에서의 전력 소비를 감소시킬 수 있는 기법들에 바탕을 두어 32 비트 RISC 코어를 구현하였다. 구현된 코어는 전형적인 5단 파이프라인으로 동작하며, 성능 향상을 위한 하바드 구조를 가진다. 2장에서 프로세서의 특징에 대하여 기술하고, 3장에서는 설계된 RISC 코어의 데이터패스 부의 상세한 구조에 대하여 기술하며, 4장에서는 코어의 제어유니트에 대해 기술하였다. 그리고 5장에서는 코어의 검증 환경 및 구현 결과를 기술하며, 6장에서 결론을 맺었다.

## Ⅱ. 프로세서의 특징

### 1. 명령어 셋

전력 효율적인 프로세서 구조 설계시 가장 먼저 고려해야 할 사항은 명령어 셋으로, 명령어 셋은 높은 코드 밀도(code density)를 가지고 있어야 한다. 높은 코드 밀도는 응용 프로그램에 대해 실행될 명령어 수를 감소시켜 명령어들을 적게 페치하게 된다. 명령어 페치는 프로세서가 소모하는 전력의 약 1/3을 차지하게되므로 증가된 코드 밀도는 보다 효율적인 메모리 사용을 통한 높은 프로세서 성능과 낮은 명령어 캐시 미스 율을 가져온다[7].

범용 RISC 프로세서의 경우 명령어들이 고정길이 명령어 포맷으로 인해 사용되지 않은 필드를 지니며, 거의 인코딩 되지 않기 때문에 일반 CISC 보다 약 30% 정도 코드밀도가 낮다. 본 논문에서는 현재 유용한 RISC 프로세서들의 명령어 셋 중 16비트 CISC 만큼 높은 코드 밀도를 가지는 ARM®V4 명령어 셋을 따른다[8].

명령어 셋은 8가지 기본적인 명령어 타입으로 구성된다. 2가지 명령어 타입(Data Processing, Multiply)은 연산에, 3가지 명령어 타입(Data Transfer, Data Swap, PSR Transfer)은 레지스터와 메모리사이의 데이터 전달을 제어하는데, 나머지 2가지(Branch, Software interrupt)는 명령어 실행 흐름 전이와 특권모드(privilege mode)로 들어갈 때 사용되어진다.

어드레싱 모드는 크게 PC 상대 모드와 베이스 레지스터 상대 모드가 있으며, 모든 명령어들이 CPSR (Current Program Status Register)내의 플래그들(N, Z, C, V)과 명령어 내 15가지 타입(equal, not equal, negative. etc)의 조건코드 값에 따라 조건적인 실행을 수행함으로써 일반 명령어와 분기 명령어를 하나의 명령어로 수행할 수 있다. 모든 단일 load와 store 명령어들은 옵션으로써 수정된 값을 자신의 베이스 레지스터

들로 다시 쓸 수 있어 스택과 큐의 소프트웨어적인구현이 편리하다. 특히 16개의 연속적인 load/store 명령어를 하나의 명령어로 대치할 수 있는 블록 데이터 명령어는 프로세서 상태 및 컨텍스트를 저장, 복구하거나 메모리로부터 데이터를 블록으로 전달할 수 있게한다.

2. Thumb 코드

본 논문에서는 ARM 명령어 셋 이외에도 ARM 코드의 압축된 형태의 16비트 Thumb 코드를 지원함으로써 전체 시스템 동작에서 가장 전력소모가 많은 메모리 접근 시 소모되는 전력을 감소시킬 수 있도록 하였다[8]. 이 16비트 명령어 코드는 명령어 메모리에서 페치되어 코어 내부에서 명령어 실행을 위해 32비트 ARM 명령어로 디코딩 되어진다.

Thumb 코드는 ARM 명령어 셋에 비해 30%정도 코드 밀도가 높은 반면 레지스터 수의 감소나 조건 코드를 사용하지 않는 등의 감소된 기능성에 의해 수행될 명령어의 수가 약 18% 정도 증가하게 되고, 이는 결국 성능을 감소시키게 되므로 성능과 코드 밀도 사이의 trade-off를 고려해야 한다. 전형적인 내장형의 제어 코드상에서 고성능이 요구되는 루틴들은 ARM 코드로 컴파일하고, 응용의 나머지는 Thumb 코드로 컴파일하여 수행시키면 보다 효율적일 수 있다[9]. 그림 1은 Thumb 코드의 ARM 명령어로 디코딩되는 예를 보여준다.

3. 동작 모드

프로세서 동작은 정상적인 프로그램 상태인 사용자 모드, 일반적인 인터럽트 처리를 위한 IRQ(Interrupt Request) 모드, 외부적인 데이터 I/O에 대한 우선순위가 높은 인터럽트 처리를 위한 FIQ(Fast Interrupt Request) 모드, 운영체제에 대해 보호되는 모드인 수퍼바이저(Supervisor) 모드, 데이터나 명령어 페치 중 단시 들어가게 되는 중단(Abort) 모드, 정의되지 않은 명령어를 만났을 때 들어가는 비정의(Undefined) 모드 등 6개의 동작모드를 가진다. 모드 전환은 PSR 명령어에 의한 CPSR(Current Program Status Register) 레지스터의 비트 전이, 외부적인 인터럽트들, 예외적인 처리에 의해 수행된다.

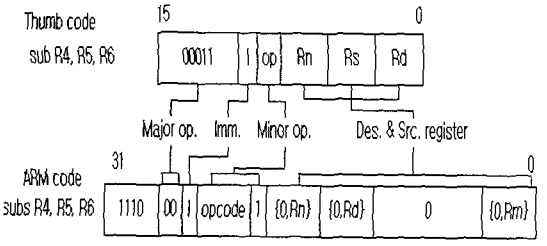


그림 1. Thumb 코드의 ARM 명령어 디코딩
Fig. 1. Decoding of Thumb code to ARM instruction.

4. 파이프라인

설계된 RISC 코어는 그림 2에 나타난바와 같이 전형적인 5단 파이프라인(Fetch, Decode, Execute, Memory, Writeback)으로 동작하며, 성능향상을 위한 하바드 구조를 가진다.

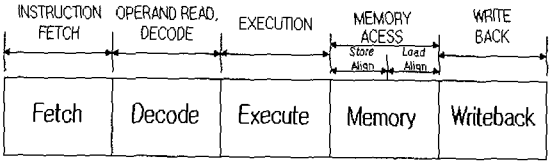


그림 2. RISC 코어의 파이프라인
Fig. 2. Pipeline of the RISC core.

## Ⅲ. 코어 데이터패스

RISC 코어의 데이터 패스는 휴대형 단말 시스템에 적합하도록 전력 소비와 성능향상을 동시에 고려하여 설계되었으며, 그림 3은 설계된 RISC코어의 전체 블록도를 나타낸다.

1. Thumb Decompressor

데이터패스에서 Thumb 코드의 처리를 위해 D-단의 명령어 디코딩 전에 16 비트 Thumb 코드를 대응하는 32 비트 코드로 변환하는 기능을 수행하는 Thumb Decompressor를 두었으며, 그림 4는 디코드 단에서의 Thumb Decompressor를 나타낸다.

Thumb 모드 비트(Tbit)에 따라 명령어 디코더는 Decompressor 출력에 따른 명령어 또는 명령어 레지스터의 32비트 명령어를 디코딩 하게 된다. Thumb 분기 명령어 중 offset 범위가 큰 서브루틴을 호출할 때 사용되어지는 Long Branch With Link 명령어는 2개의 분기명령어(offset high, offset low)를 연속 사용하여 23 비트 offset를 생성하는데, 이 명령어에 대응하는 32 비트 명령어는 존재하지 않는다. 이와 같은 분기 명령어에 대해서는 Decompressor는 제어 신호(bl_h, bl_l)만 발생하고, 명령어 디코더에서 제어 신호들을 받아 디코딩하도록 하였다.

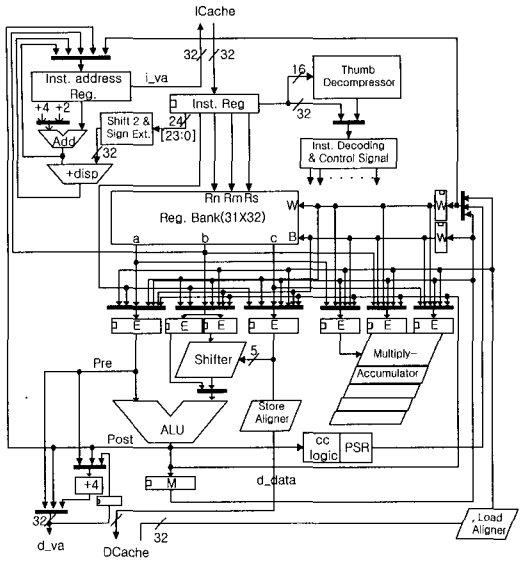


그림 3. 설계된 RISC 코어의 전체 블록도
Fig. 3. Block diagram of the RISC core.

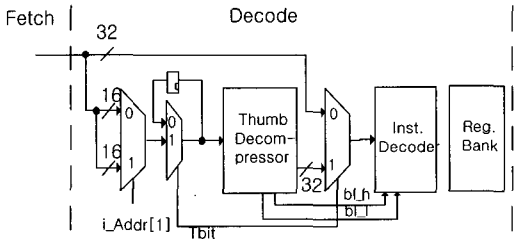


그림 4. Decode 단에서의 Thumb decompressor
Fig. 4. Thumb decompressor in Decode stage.

2. 분기 덧셈기

디코드 단에 분기 어드레스 계산을 위해 32 비트 덧셈기(CLA)를 추가하였으며, 이는 코어 실행 시간동안 계속 활성화되어 전력 소모를 가져오나 분기 taken시

| | F | D | E | B | W |
|---|---|---|---|---|---|
| 100: ADDS R3,R2,R4 | PC<- 100 | | | | |
| 104: BE y (target pc = 400) | PC<- 104 | Read R2, R4 | | | |
| 108: | PC<- 108 | target PC <- 400 | W<- R2+R4 cc <-alu.cc | | |
| 400: y | PC<- 400 | do nothing | do nothing | W'<-W | |
| 404: | PC<- 404 | Decode y | do nothing | do nothing | R3 <- W' |

그림 5. 분기 명령어의 파이프라인 실행도
Fig. 5. Pipeline execution diagram of branch instruction.

따로 분기 예측 하드웨어를 필요로 하지 않고 1싸이클 penalty 성능 향상을 가져온다.

그림 5는 분기 명령어의 실행과정을 나타내는데 PC가 104인 지점에서 분기명령어 BE(Branch Equal) 명령어가 이전 명령어인 ADD 명령어에 의해 발생된 조건 코드들에 의존하는 경우를 나타낸다. ADD 명령어의 실행(Execute) 단으로부터의 조건 코드들에 따라 다음 페치 싸이클에 분기 target PC를 지시하게되어 1싸이클 penalty를 갖는다.

3. 레지스터 뱅크

전체 프로세서 전력소모의 10% 정도를 차지하는 레지스터 파일은 휴대용에 적합하기 위해서 레지스터들의 수가 전력 효율적으로 최적화 되어야한다. SPARC 같이 중첩된 레지스터 윈도우를 사용하는 경우는 데이터 캐시 접근 빈도를 감소하기 위해 거대한 레지스터 파일을 사용하는데 이는 레지스터 접근시 복잡한 디코딩으로 인해 코어의 전력 소모가 증가하게된다[10].

설계된 레지스터 뱅크는 31개의 범용 레지스터들로 구성된 레지스터 뱅크 구조로서 6가지 동작모드에 따라 16개의 범용 레지스터(R0-R15)가 할당되며, 6개의 상태 레지스터들과 함께 예외상황에서의 초기화 관련 처리를 빨리 수행할 수 있게 설계되었다. 즉 FIQ경우는 7개의 레지스터를 따로 두어 레지스터 저장 복구 과정을 거의 생략 할 수 있다. 설계된 레지스터 뱅크는 3개의 읽기포트와 2개의 쓰기 포트를 지니며, 전력 감소를 위해 사용되지 않은 모드의 레지스터들의 클럭을 gated 클럭을 사용하여 disable함으로써 현재 수행중인 해당 모드의 레지스터들만 활성화되게 하였다.

4. ALU

ALU는 데이터 처리명령어의 산술 및 논리 연산, load/store 명령어의 데이터 메모리 접근을 위한 주소 계산, 곱셈 명령어의 최종 결과를 위한 부분 합과 부분 carry들의 덧셈 연산 등을 수행한다. ARM®V4 명령어 셋의 사용빈도를 분석한 결과를 살펴보면[11] 약 20%는 산술 연산을 약 20%는 논리연산 수행을 요구하며 load/store 연산을 포함하면 약 60% 정도 ALU 연산을 요구한다. 또한 전력 소모도 코어 전력의 약 20% 이상 차지하게 되므로 ALU는 속도 및 전력 소모를 고려하여 설계되어야한다.

수행할 ALU 연산은 표 1과 같으며, ALU 블록도는 그림 6과 같이 논리 연산 수행을 위한 LU(Logic Unit)

와 1의 보수 연산을 수행하는 보수기(Compl.), 32비트 덧셈기로 구성되어있다. 특히 덧셈기의 경우 전력 소모와 성능면에서 좋은 것을 고려하여, 본 논문에서는 32 비트 CSA(Carry Select Adder)를 4-4-7-9-8 단으로 구성하였다[12].

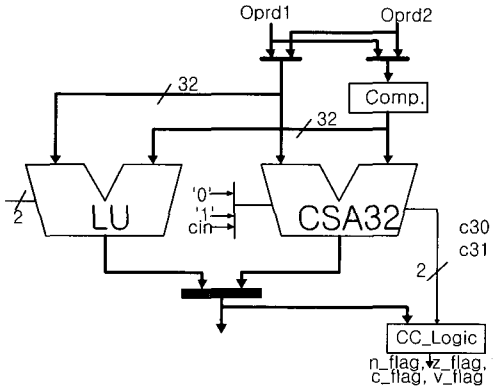


그림 6. ALU 블록도
Fig. 6. Block diagram of ALU.

표 1. ALU 연산 기능
Table 1. Functions of ALU operation.

| 명령어 | | 연산 | A | B | Funt. |
|---|---|---|---|---|---|
| Data proc. | AND | And | A | B | AND |
| | EOR | Exclusive-OR | A | B | XOR |
| | SUB | Subtract | A | B′ | add |
| | RSB | Reverse sub. | A′ | B | add |
| | ADD | add | A | B | add |
| | ADC | add with carry | A | B | add |
| | SBC | subtract with carry | A | B′ | add |
| | RSC | reverse sub w/carry | A′ | B | add |
| | TST | test bits | A | B | AND |
| | TEQ | test equal | A | B | XOR |
| | CMP | compare | A | B′ | add |
| | CMN | compare negative | A | B | add |
| | ORR | OR | A | B | OR |
| | MOV | move | '0' | B | B |
| | BIC | bit clear | A | B | AND |
| | MVN | move NOT | '0' | B′ | B′ |
| PSR | | bypass | '0' | B | B |
| BL | | subtract | A | B′ | add |
| Multiply | | add | A | B | add |
| LDR/STR(+) | | add | A | B | add |
| LDR/STR(-) | | subtract | A | B′ | add |
| LDM/STM(+) | | add | A | B | add |
| LDM/STM(-) | | subtract | A | B′ | add |

그래픽 처리와 같은 연산은 쉬프트 연산을 많이 요구하는데, ALU 및 쉬프트 연산을 한 싸이클에 처리하기 위해 ALU의 오퍼랜드 입력 중 하나에 32비트 배럴 쉬프터를 연결하였다. 배럴 쉬프터는 MUX 기반으로한 양방향 배럴 쉬프터로 설계되었으며, LSL(Logical Shift Left), LSR(Logical Shift Right), ASR(Arithmetic Shift Right). ROR(Rotate Right), RRX(Rotate Right Extended)의 5가지의 기능을 수행한다.

설계된 쉬프터는 명령어 셋에 주어지는 쉬프트 양을 직접 제어 신호로 사용함으로써 어레이 배럴 쉬프터에서 사용되는 5:32 복호기를 사용하지 않는다.

5. MAC

본 논문에서 설계된 MAC은 2의 보수의 recoding에 기반을 둔 수정된 부스 알고리즘을 사용하였는데 부분곱 생성시 완전 부호확장 대신 3개의 부호확장 비트만을 사용하였다[13]. 특히 속도와 면적을 고려하여 싸이클당 32×8 비트 연산을 수행하도록 하여 32×8 비트 하드웨어 면적을 지니도록 하였고, 불필요한 연산에 따른 전력을 소모하지 않도록 조기 종결 조건회로를 두어 곱셈의 최소 실행이 가능하도록 하였다. 지원되는 곱셈 연산 기능들은 표2에 나타내었으며 Rm은 피승수, Rs는 승수, Rn은 MUL 또는 MLA 때 누산될 수를, Rd는 결과를 저장하는 레지스터를 나타낸다.

표 2. 설계된 MAC의 연산 기능
Table 2. Operation functions of the MAC.

| 명령어 | 동 작 | 기 능 | 결과 |
|---|---|---|---|
| MUL | multiply | Rd := Rm * Rs | 하위 32비트 |
| MLA | multiply and accumulate | Rd := Rm*Rs+Rn | 하위 32비트 |
| UMULL | unsigned multiply long | RdHi,RdLo := Rm*Rs | 64비트 |
| UMLAL | unsigned multiply and accumulate | RdHi,RdLo := Rm*Rs+RdHi,RdLo | 64비트 |
| SMULL | signed multiply long | RdHi,RdLo := Rm*Rs | 64비트 |
| SMLAL | signed multiply and accumulate | RdHi,RdLo :=Rm*Rs+RdHi,RdLo | 64비트 |

MAC 구성은 그림 7의 블록도와 같이 부분곱 발생기, Wallace 트리부, 최종 부분합들의 결과를 위한 ALU 등의 세 부분으로 구성되어진다.

곱셈 싸이클의 수는 승수 레지스터에 저장되어진 값에 따라 달라지는데 승수 레지스터는 쉬프트 레지스터로서 싸이클당 8비트 쉬프트 하게되어 싸이클당 32×8 비트 연산을 수행하게 되며, 8~32 비트 값들이 모두 '0'이거나 '1'이면 부스 인코딩값이 모두 '0'이 되므로 조기 종결하도록 하였다.

부분곱 발생기(PPG)는 싸이클당 4개의 1의 보수 연산된 부분합들을 출력하며, 하드웨어 면적을 적게하기

위해 완전 부호확장을 하지 않고 부분곱의 부호 비트 외에 2 비트만을 추가하여 설계하였다.

부분곱들의 합을 위한 CSA 트리 구조는 그림 7에 나타낸바와 같이 3 단의 CSA들로 구성되어진다. 4:2 compressor 들로 구성된 첫 번째 CSA는 PPG에서의 결과 4개를 입력으로 받아 2개를 출력하며, 두 번째 CSA는 1의 보수결과를 2의 보수로 전이하기 위해 부분곱들의 부호 비트들을 트리 내에서 더하는 역할을 수행하고, 세 번째 CSA는 이전 부분 곱들의 합과 현재 부분 곱들을 누산한다. MAC 연산이 누산 연산의 경우에는 첫 번째 싸이클 때 누산 오퍼랜드 값(Addend)을

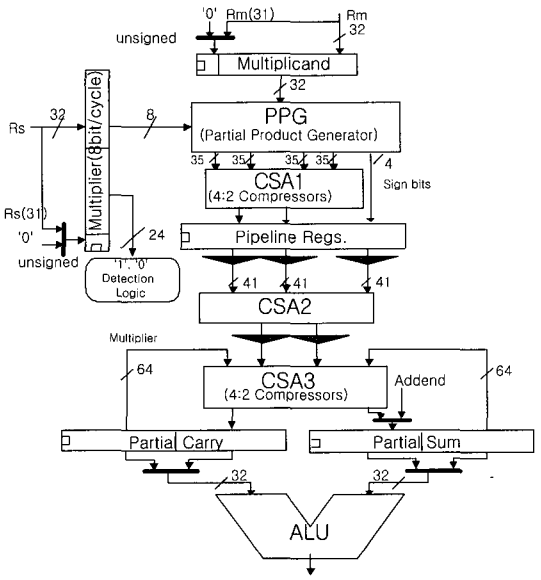


그림 7. 설계된 MAC의 블록도
Fig. 7. Block diagram of the MAC.

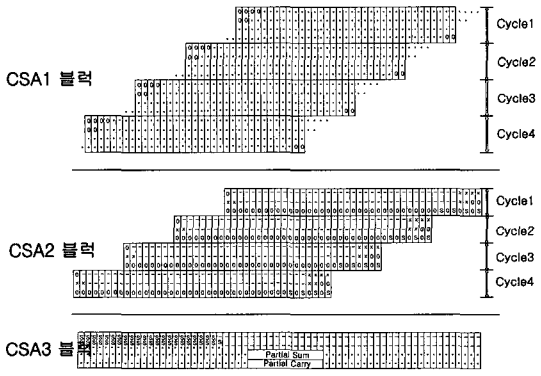


그림 8. CSA 트리부의 각 싸이클별 연산
Fig. 8. Operation of the CSA tree per cycle.

Partial Sum에 저장함으로써 따로 누산할 때 부가적인 싸이클이 요구되지 않도록 하였다. 그림 8은 CSA 트리부의 싸이클 별 연산을 나타낸다.

최종적인 출력 값은 앞에서 설명한 ALU를 이용하는데, 요구되는 연산 출력이 64비트일 경우 처음에 하위 32비트를 내보내고, 다음 싸이클 때 상위 32비트를 내보도록 하였다. 표 3은 승수 값에 따른 싸이클 수를 나타내며 최소 3에서 최대 7싸이클이 소요됨을 알 수 있다.

표 3. 승수 값에 따른 곱셈 싸이클
Table 3. Multiplication cycles according to the value of multiplier.

| 명령어 \ 승수값 | $2^{8}$ 이하 | $2^{8}$ ~$2^{16}$ | $2^{16}$~$2^{24}$ | $2^{24}$~$2^{32}$ |
|---|---|---|---|---|
| MUL/MLA | 3 | 4 | 5 | 6 |
| UMULL/UMLAL | 4 | 5 | 6 | 7 |
| SMULL/SMLAL | 4 | 5 | 6 | 7 |

6. 블록데이터 전달 레지스터 인코더

최대 16개의 load/store 명령어를 대치할 수 있는 명령어인 블록 데이터 전달 명령어(LDM, STM)의 구현을 위해 코어 내부 디코드 단에 블록데이터 전달 레지스터 인코더를 추가하였으며, 인코더는 매 싸이클마다 전달될 레지스터 번지를 발생시킨다. 설계된 인코더 블록도는 그림 9에 나타내었으며, 동작은 다음과 같다.

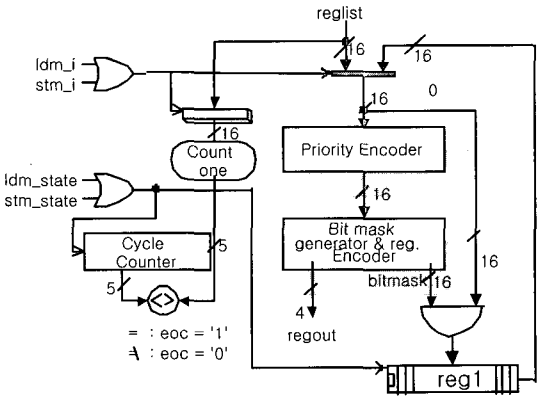


그림 9. 블록데이터 전달 레지스터 인코더 블록도
Fig. 9. Block diagram of the block data transfer register encoder.

블록 데이터 전달 명령어가 명령어 디코더에서 디코딩되면 명령어 레지스터로부터 전달될 해당 레지스터

를 지시하는 16비트 값이 '1'의 수를 출력으로 하는 countone 회로에 전달되고, 또한 Priority Encoder 회로를 통해 전달하고자 하는 레지스터 중 가장 낮은 수의 레지스터 해당 비트가 셋팅된다. Priority encoder 회로 출력중 '1'인 부분의 해당 레지스터 번지 4비트가 출력되고, 현재 출력된 레지스터 비트는 비트 마스크를 통해 '0'이 되며 이는 reg1에 저장된다. 전달될 레지스터의 수가 countone회로의 출력만큼 카운트하게 되면 (eoc='1') 인코더 동작을 마치게 되며, eoc가 '0'이면 reg1의 출력이 다시 Priority Encoder 입력이 되어 전달될 레지스터 수만큼 인코딩작업을 반복하게 된다. 이와 같은 블록 데이터 전달 레지스터 인코더는 데이터 어드레스 발생 로직과 결합하여 블록전달 명령어 실행 시 싸이클당 1개의 데이터 전달을 수행 할 수 있다. 단 r15(프로그램 카운터 값)를 포함한 ldm 명령어는 r15 값 load시 프로그램 카운터의 값이 유용할 때까지 페치와 명령어 디코딩 작업이 정지해야 하기 때문에 추가적인 3 싸이클이 더 소요된다.

## Ⅳ. 코어 제어 유닛

### 1. 파이프라인 제어

파이프라인 구조의 프로세서에서 명령어 디코딩은 데이터 정적(data stationary)과 시간 정적(time stationary)의 파이프라인 제어 방식이 사용되어진다. 본 논문에서는 모든 제어 신호를 Decode 단 때 발생시켜 지연 회로를 통해 필요한 시점까지 지연시키는 데이터 정적제어 방식을 사용하였다. 파이프라인 전체의 진행에 대한 제어는 파이프라인 전역 제어기를 두어

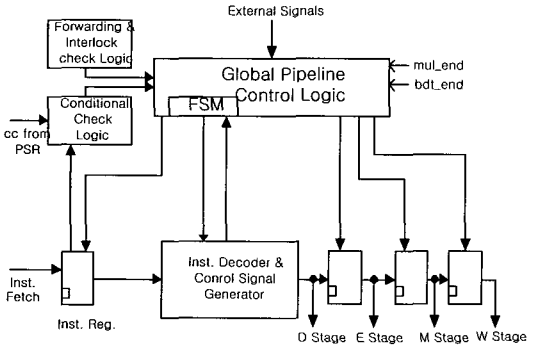


그림 10. 파이프라인 제어 블록도
Fig. 10. Block diagram of pipeline control.

메모리, 버스 hold 신호, 캐시 미스, 외부적인 트랩 등의 외부적인 신호와 포워딩 및 인터락 체크 로직, 조건 평가회로 등의 상황을 고려하여 파이프라인 진행에 대한 제어를 담당하도록 하였다. 또한 다중 싸이클 명령어 처리는 제어기내 Mealy Model로 구성된 FSM (Finite State Machine)을 두어 처리하도록 하였는데, 단일 싸이클 명령어는 명령어 레지스터 값에 따른 명령어 디코더에 따라 제어 신호를 발생하며, 다중 싸이클의 경우는 각 상태에 따라 1 싸이클 1 명령어 수행 개념으로 제어신호를 발생하도록 하였다. 그림 10은 파이프라인 제어 블록도를 나타낸다.

### 2. 포워딩 및 인터락 체크로직

포워딩은 각 단계에서의 연산 결과를 레지스터 파일에 저장하기 전에 다음 명령어가 그 값을 필요로 할 때 연산된 결과 값을 미리 다음 단으로 넘겨줄 때 사용되어진다. E-D, M-D, W-D 경로에 대해 E-단계, M-단계, W-단계의 목적지 레지스터 주소와 D 단계 피연산자(Rm, Rn, Rs) 레지스터 주소를 비교하여 포워딩하도록 하였으며, 동시에 가능한 경우는 가장 최근의 결과 값을 포워딩하도록 설계하였다.

인터락은 명령어간의 의존성이나 하드웨어적인 의존관계가 있을 때 파이프라인을 정지시켜 의존관계를 보존시키는 방법으로 설계된 구조에서는 load 인터락과 mrs 인터락이 발생할 수 있다. Load 인터락은 load 명령어 다음 명령어가 load되는 데이터를 피연산자로 이용하고자 할 때 발생한다. mrs 인터락은 M단에 있는 상태레지스터 값을 레지스터에 저장하는 mrs 명령어 다음 명령어가 mrs에서 전달될 데이터를 피연산자로 이용할 때 발생한다. 이와 같은 경우는 파이프라인을 한 단계 stall 시켜 제어하도록 하였다.

### 3. 트랩 처리부

설계된 코어 구조에서의 트랩은 명령어 디코딩 시 발생하는 소프트웨어 인터럽트, 정의되지 않은 명령어 트랩과 명령어 실행 중 메모리 접근(명령어, 데이터)시 발생하는 prefetch, data abort, 그리고 명령어 흐름과 관계없이 외부적으로 발생하는 Reset, IRQ, FIQ 등의 3 종류가 존재한다. 이와 같은 트랩 발생 요인중 데이터 메모리 접근시 발생하는 data abort는 M-단에서 발생하여 W-단에서 트랩을 확인 할 수 있기 때문에 M-단 이전에 발생된 트랩들은 M-단까지 지연시켜 발생하도록 하였다. 위의 트랩들이 동시에 발생하는 경우는 트

랩 우선 순위기를 두어 우선 순위에 따라 트랩을 처리하도록 하였다[14].

4. 동적 전력 관리

설계된 프로세서의 동적관리는 레지스터 뱅크의 각 모드에 따른 gated 클럭 사용 외에도 각 명령어 실행에 따른 프로세서 데이터 패스부의 파이프라인 레지스터들을 load-enable 신호를 사용하거나, 클럭을 gating하여 비활성화 시킴으로써 전력소모를 감소시킬 수 있도록 하였다. 데이터 정적 제어 방식의 경우 명령어 각 단에서의 제어 신호들이 디코드 단 때 모두 발생되기 때문에 명령어를 실행함에 있어 각 단에서 사용되지 않는 블록들을 미리 알 수 있다. 따라서 명령어 페치와 디코딩 단을 제외한 E-단, M-단, W-단의 파이프라인에 동적 전력 관리 기법을 적용할 수 있어서, E-단에서는 load-enable 파이프라인 레지스터를 사용하였고 M-단, W-단에서는 gated 파이프라인 레지스터를 사용하였다.

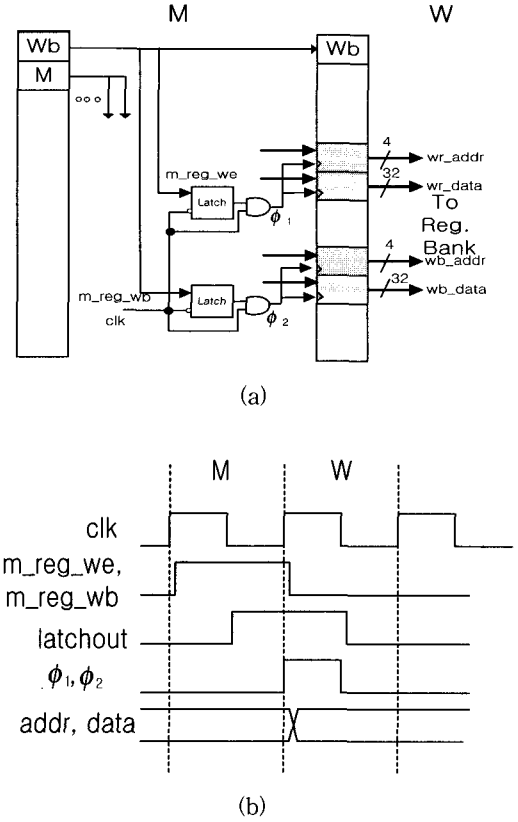


그림 11. 선택적인 Gated 파이프라인 레지스터 (a) 레지스터 쓰기 주소와 데이터에 대한 gated 클럭 (b) 타이밍도

Fig. 11. Selective gated pipeline register. (a) Gated clock for register write address, data. (b) Timing diagram.

M-단, W-단에서는 데이터가 넘어가기 전의 이전 단(E-단, M-단)에서 각 단에 사용될 제어 신호들을 이용하여 gated 클럭을 사용함으로써 선택적으로 클럭을 비활성화 시킬 수 있다. 그림 11은 W-단에서의 레지스터 뱅크 주소와 데이터에 대한 gated 파이프라인 레지스터를 나타낸다.

그림에서 보는 바와 같이 M-단에서 W-단의 사용될 제어 신호(m_reg_we, m_reg_wb)와 하강 에지 래치 및 AND 게이트를 사용하여 레지스터 쓰기 명령어가 수행될 때만(reg_we='1', reg_wb='1') 해당 파이프라인 레지스터를 활성화하도록 하였다. M-단 파이프라인 레지스터들의 클럭도 같은 방식으로 제어하여 약간의 부가적인 회로를 추가함으로써 명령어 수행 시 필요한 파이프라인 레지스터들의 클럭만을 활성화하도록 하였다.

E-단의 파이프라인 레지스터의 경우는 조건 평가회로의 결과 또는 인터락의 결과가 한 클럭내에서 후반에 나타나므로 클럭 중간에 래치를 사용하는 gated 클럭 방식은 부적절한 신호전이를 발생하기 때문에 load-enable 파이프라인 레지스터를 사용하여 구현하였다.

## V. 검증 및 구현

1. 시뮬레이션

본 논문에서는 32비트 RISC 코어를 VHDL로 합성가능한 RTL 수준에서 모델링하였으며, 검증을 위해 명령어 및 데이터 캐쉬의 행위수준 모듈을 추가한 환경에서, 먼저 프로세서의 각 명령어 별 시뮬레이션을 수행하였다. 각 명령어 별 검증 후 프로그램 수준의 검증을 위해서 C언어로 기술된 응용 프로그램들을 트랩 처리기, ROM/RAM 매핑 및 각 모드별 스택 포인터 초기화 프로그램들을 포함한 환경에서 ADS(ARM Developer Suite)의 ARM 컴파일러 및 Thumb 컴파일러를 사용하여 컴파일을 수행하였으며, 컴파일된 명령어들의 이진 코드를 추출한 후 행위레벨의 명령어 캐쉬에 로드하여 VHDL 시뮬레이션을 수행하였다

. 그림 12는 설계된 코어의 검증 환경을 보여주는데 검증을 위해, VHDL 시뮬레이션 결과들 중 레지스터 뱅크내 각 모드별 레지스터 값과 상태 레지스터(CPSR, SPSR) 값, 그리고 메모리 접근 어드레스와 데이터, 프

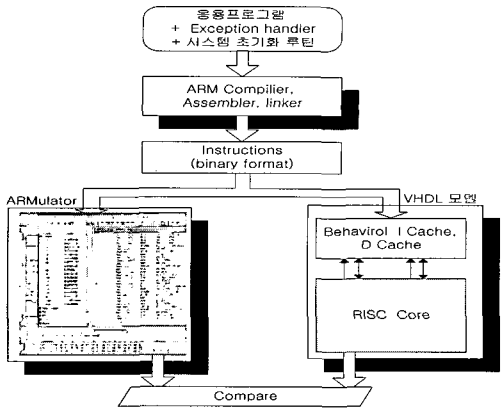


그림 12. 응용프로그램의 검증 환경
Fig. 12. Verification environment for application program.

표 4. C 프로그램의 ARM 이진코드 수행 결과
Table 4. Execution result of the ARM binary code for C programs.

| | qsort 10.c | hanoi tower9.c | matrix 10.c | Bubble sort10.c | merge sort16.c |
|---|---|---|---|---|---|
| 명령어수 | 880 | 8,176 | 1,190 | 701 | 1,693 |
| 수행 싸이클 | 1,493 | 12,532 | 1,616 | 902 | 2,212 |
| 데이터 load 수 | 270 | 2,561 | 419 | 186 | 318 |
| load inter lock 수 | 72 | 1,020 | 110 | 108 | 110 |
| CPI | 1.696 | 1.533 | 1.358 | 1.287 | 1.306 |

표 5. C 프로그램의 Thumb 이진코드 수행 결과
Table 5. Execution result of the Thumb binary code for C programs.

| | qsort 10.c | hanoi tower9.c | matrix 10.c | Bubble sort10.c | merge sort16.c |
|---|---|---|---|---|---|
| 명령어수 | 1,161 | 11,246 | 1,629 | 677 | 2,240 |
| 수행 싸이클 | 1,897 | 16,371 | 2,274 | 917 | 3,081 |
| 데이터 load 수 | 289 | 2,561 | 625 | 188 | 517 |
| load inter lock 수 | 106 | 1,533 | 310 | 137 | 327 |
| CPI | 1.634 | 1.456 | 1.396 | 1.354 | 1.375 |

로그램 카운터 값들을 같은 프로그램에 대해 ADS에서 제공하는 코어 시뮬레이터인 ARMulator에서의 해당 내용과 명령어 단위로 비교하여 일치하는 것을 확인하였다. 표 4는 테스트 C 프로그램에 대한 ARM 이진 코드의 실행결과를, 표 5는 테스트 C 프로그램에 대한 Thumb 이진 코드의 실행결과를 나타낸다.

표 4과 표 5에 나타낸 결과는 캐시 미스를 제외한 시뮬레이션 결과이며, 코어의 평균 CPI는 1.44 정도(ARM 컴파일러 사용-1.43, Thumb 컴파일러 사용-1.44)인데 이는 3단 파이프라인의 ARM7TDMI 코어(CPI=1.6)보다는 감소된 결과이나, MIPS, DLX, SPARC 등의 전형적인 4 단 또는 5 단 파이프 라인으로 동작하는 RISC 코어들의 CPI(약 1.2-1.4) 보다 약간 증가된 결과이다[9][15]. 이는 블록 데이터 전달 명령어 및 곱셈 명령어에 기인하며, 특히 블록 데이터 전달 명령어의 경우 메모리 접근시 성능이 같은 기능의 단일 싸이클 명령어인 메모리 전달 명령어(load/store) 여러 개로 구현했을 때 보다 코드 밀도가 높고, 연속적인 메모리 접근에 따른 성능이 약 4 배까지 증가할 수 있는 명령어임을 고려한다면 충분히 예상되는 결과이다[14]. 표4와 표5의 결과들을 비교해 보면 thumb 코드로 컴파일 했을 경우 수행될 명령어 수가 약 1.34 배 증가하여 명령어 수행 싸이클 수는 증가하나 thumb 코드는 16비트 명령어이기 때문에 명령어 저장 면적이 적어 내장형의 응용 프로그램에 적합하다는 것을 알 수 있다.

2. 합성 및 구현

응용 프로그램에서 검증을 완료된 RISC 코어는 IDEC(IC Design Education Center)에서 제공하는 0.6 ㎛ CMOS 1-poly 3-metal IDEC C-631 셀라이브러리를 사용하여 Synopsys사의 Design Compiler에서 논리 합성되어 회로의 네트리스트가 추출된 후 Cadence 사의 Simwave를 이용하여 게이트 레벨 시뮬레이션 되었다. 합성된 회로의 게이트 수는 2 입력 nand 게이트 기준으로 약 41,000이며, 합성된 RISC 코어는 Mentor 사의 IC Station에서 P&R 되어 SDF(Standard Delay

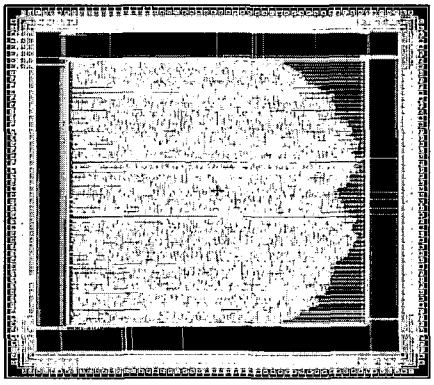

그림 13. RISC 코어의 레이아웃
Fig. 13. Layout of the RISC core.

표 6. 코어의 주요 특성

Table 6. Main Characteristics of the Core.

| | |
|---|---|
| 공 정 | 0.6㎛ 1-poly 3-metal CMOS 공정 |
| 칩 사이즈 | 9×9 $mm^2$ |
| 코어 사이즈 | 7.2×7.2 $mm^2$ |
| 트랜지스터 수 | 166,875 |
| I/O 핀수 | 입력 : 42<br>출력 : 78<br>양방향 : 32<br>VDD : 24<br>VSS : 24<br>Total : 200 |
| 패키지 타입 | 200 QFP |
| 최대 동작주파수 | 45 MHZ |
| 전력 소모 | 174 mW |

Format)가 추출된 후 타이밍 검증되었다. 또한 소비 전력은 코어의 레이아웃결과인 GDSII 파일을 Cadence사의 Virtuso에서 읽어들인 후 spice파일을 추출하여 Synopsys사의 Epic에서 측정되었다. 그림 13은 코어의 레이아웃을 나타내며, 표 6은 Mentor사의 IC Station에서 자동레이아웃된 칩의 특성을 나타낸다.

본 논문에서는 레지스터 뱅크 및 배럴 쉬프터, ALU 등의 데이터 패스부를 합성하여 구현함으로써 칩 사이즈가 증가하였으나 이를 full-custom으로 설계하고 0.25 ㎛ 이하 공정으로 구현된다면 100 MHz 이상을 요구하는 고성능의 멀티미디어 휴대 단말시스템에 적합한 내장형 RISC 코어로 구현될 수 있을 것으로 예상된다.

## VI. 결 론

휴대 단말 시스템에서 사용될 수 있는 32 비트 RISC 코어를 설계하고 구현하였다. 설계된 RISC 코어는 전력 감소를 위해 구조레벨에서 Thumb 코드를 지원하고, 로직 레벨에서 파이프라인 레지스터들의 동적 전력 관리 기법을 사용하였고, 고성능을 위해 분기 덧셈기, 연속적인 메모리 접근을 위한 블록 데이터 전달 레지스터 인코더, DSP 기능을 위한 32×8 면적의 MAC 등을 추가하였다. 코어의 RTL 수준 VHDL 모델은 프로그램 수준에서 명령어 단위별로 ADS의 ARMulator와 비교 검증되었으며, 0.6㎛ CMOS 1-poly 3-metal IDEC 셀 라이브러리를 사용하여 합성 및 구현되었다. 합성된 코어 의 크기는 약 41,000 게이트이고, 예상 동작주파수는 45 MHz이다. 본 논문에서 설계된 RISC 코어는 stand-alone 코어, ASSP(Application Specific Standard Parts)의 매크로 셀로서 사용될 수 있으며, 0.25 ㎛ 이하 공정으로 구현된다면 100 MHz 이상을 요구하는 고성능의 멀티미디어 휴대 단말시스템에 적합한 내장형 RISC 코어로 사용될 수 있을 것이다.

## 참 고 문 헌


[1] S. Gray, P. Ippolito, G. Gerosa, C. Dietz, J. Eno, and H. Sanchez, "PowrPC 603 A Microprocessor for Portable Computers", IEEE Design & Test of Computer, pp. 14~23, Winter 1994.

[2] T. Litch and J. Slaton, "Strong ARMing Portable Communications", IEEE Micro., pp.48~55, March/April 1998.

[3] L. Benini, P. Siegel, and G. De Micheli, "Automatic Synthesis of Low-Power Gated-Clock Finite State Machines", IEEE Trans. on CAD, Vol. 15, No. 6, pp. 630~643, June 1996.

[4] C. A. Papachristou and M. Spining, "A Multiple Clocking Scheme for Low-Power RTL Design", IEEE Trans. on VLSI, Vol. 7, No. 2, pp. 266~276, June 1999.

[5] S. H. Chow, Y. C. Ho, and T. Hwang, "Low Power Realization of Finite State Machines-A Decomposition Approach", ACM Trans. on Design Automation of Electronic Systems, Vol. 1, No. 3, pp. 315~330, July 1996.

[6] M. Alidina, J. Monteiro, and S. Devadas, "Precomputing-Based Sequential Logic Optimization for Low-Power", IEEE Trans. on VLSI, Vol. 2, No. 4, pp. 426~436, Dec. 1994.

[7] J. Bunda, D Fussell, R. Jenevein, and W. C. Athas, "16-Bit vs. 32-Bit Instructions for Pipelined Microprocessors", Proc. Int'l Symp. Computer Architecture, IEEE CS Press, pp. 237~246, 1992.

[8] ARM Architecture Reference, Advanced RISC Machines. Ltd., Cambridge, U.K., 1995.

[9] S. Segars, K. Clarke, and L. Goudge, "Embedded Control Problems, Thumb, and the ARM7TDMI", IEEE Micro., pp. 22~30, Oct. 1995.

[10] T. Burd and B. Peters, A Power Analysis of a Microprocessor : A Study of an Implementation of the MIPS R3000 Architecture, ERL Technical Report, Univ. of California, Berkeley, 1994.

[11] S. Segars, "ARM7TDMI Power Consumption", IEEE Micro., pp. 12~19, July/August 1997.

[12] A. Bellaouar and M. I. Elmasry, Low-Power Digital VLSI Design, Kluwer Academic Publishers, 1995.

[13] I. S. Abu-khater, A. Bellaouar, and M. I. Elmasry, "Circuit Techniques for CMOS Low-Power High-Performance Multipliers", IEEE Journal of Solid-State Circuits, Vol. 31, No. 10, pp. 1535~1546, October 1996.

[14] S. Furber, ARM System Architecture, Addison-Wesley, 1996.

[15] J. Henessy and D. A. Patterson, Computer Architecture : A Quantitative Approach, Morgan Kaufmann, 1996.

## 저 자 소 개

丁 甲 天(正會員)

1996년 : 전남대학교 컴퓨터공학과 학사. 1998년 : 전남대학교 전자공학과 석사. 1999년~2000년 : 전남대학교 고품질전기전자부품 및 시스템 연구센터 연구원. 1998년~현재 : 전남대학교 전자공학과 박사과정. <주관심분야> 저전력 프로세서 구조, 영상압축, 영상통신용 ASIC 설계, DSP 설계, VLSI 설계 및 CAD 등

朴 性 模(正會員)

1977년 : 서울대학교 전자공학과 학사. 1979년 : 한국과학기술원 전기 및 전자공학과 석사. 1988년 : 노스캐롤라이나 주립대학 전기 및 컴퓨터공학과 공학박사. 1979년~1984년 : 한국전자통신연구소 반도체단 설계개발부 선임연구원. 1988년~1992년 : 올드도미니언 대학교 전기 및 컴퓨터공학과 조교수. 1992년~현재 : 전남대학교 컴퓨터공학과 교수. 1994년~1996년 : 컴퓨터공학과 학과장 역임. 1997년~1999년 : 전남대학교 정보통신특성화추진센터 소장 역임. <주관심분야> 마이크로프로세서, 멀티미디어 프로세서 구조, VLSI 시스템 설계, 신호처리용 ASIC 설계, 영상압축, 영상처리 등