論文 97-22-10-5

# PDA를 위한 32비트 RISC 코어의 설계

正會員 곽 승 호*, 최 병 윤**, 이 문 기*

# A Design of 32-Bit RISC Core for PDA

Sung Ho Kwak*, Byeong Yun Choi**, Moon Key Lee* *Regular Members*

※이 논문은 1994년도 한국학술진흥재단의 대학부설 연구소 연구과제 연구비에 의하여 연구되었음.

요 약

본 논문에서는 PDA나 PCS와 같은 내장형 응용을 위한 RISC 코어를 설계하였다. 이 RISC 프로세서는 내장형 응용의 중요한 특성인 빠른 인터럽트 핸들링, 빠른 컨텍스트 스위칭과 저전력 소모를 지원한다. 또한 조건부로 수행 가능한 명령어 군과 블럭 전송 명령 그리고 곱셈 명령을 이용하여 프로세서의 성능을 향상시켰다. 3단 파이프라인을 이용하였으며 2-phase 클럭을 사용한 단일 사이클 명령어 수행이 가능하다. 이 프로세서는 5.0×5.0mm²의 면적에 약 88,000개의 트랜지스터가 집적되었으며 0.6μm 삼중 금속 단일 폴리 공정을 이용하여 레이아웃 되었다. 최대 동작 주파수는 40MHz이며 예상 전력 소비는 179mW이다.

ABSTRACT

This paper describes RISC core that has been designed for embedded and portable applications such as PDA or PCS. This RISC processor offers low power consumption and fast context switching. Processor performance is improved by using conditional instruction execution, block data transfer instruction, and multiplication instruction. This architecture is based on RISC principles. The processor adopts 3-stage instruction execution pipeline and has achieved single cycle execution using a 2-phase 40MHz clock. This results in a high instruction throughput and real-time interrupt response. This chip is implemented with 0.6μm triple metal CMOS technology and consists of about 88K transistors. The estimated power dissipation is 179mW.

## Ⅰ. 서 론

인텔에서 1971년 처음으로 프로세서가 개발된 이후로 프로세서의 성능과 그 응용 분야는 발전을 거듭해 오고 있다. 4비트 계산기용 프로세서로 처음 개발된 인텔의 4004에서부터 8비트, 16비트, 32비트, 64비트의 데이터 처리 능력과 수백 MHz의 계산 속도를 가

*연세대학교 전자공학과
**동의대학교 컴퓨터공학과
論文番號:97163-0515
接受日字:1997年 5月 15日

진 프로세서가 계속 개발고 있으며 그 응용 분야는 하드디스크나 LAN 제어기와 같은 내장형 응용에서 슈퍼 컴퓨터까지 범위를 넓혀가고 있다. 다목적용 프로세서로 개발된 인텔의 펜티엄 칩이나 DEC사의 알파칩은 400~600MHz의 처리 속도에 이르고 있으며 그 성능 경쟁은 끊임 없이 계속되고 있다. 하지만 웍스테이션이나 PC와 같은 일반적인 목적의 용도로 사용되는 프로세서와는 달리 특정 응용을 위한 프로세서의 필요성 또한 증대되고 있다. Z80에서부터 가속화된 내장형 응용을 위한 프로세서의 개발은 그 응용 분야를 계속 넓혀 가고 있으며 성능 또한 향상되고 있다. 내장형 프로세서의 응용 범위의 확대로 인하여 1980년대 초에는 모토롤라의 6805 아키텍처나 인텔의 8051과 같은 8비트 프로세서에서 16비트 프로세서로 확장되기 시작하였다. 이 때 개발된 16비트 프로세서인 모토롤라의 68000과 인텔의 8086은 PC와 같은 다목적용 프로세서로서 뿐만 아니라 내장형 응용을 위한 프로세서로도 사용되었다. 그러나 다목적용 프로세서와는 다른 내장형 프로세서의 특징으로 인하여 전문화된 내장형 프로세서가 필요하다. 다목적용 프로세서는 시스템의 성능 증가로 끊임없는 프로세서의 성능 경쟁을 계속하고 있지만 그와는 달리 내장형 프로세서는 응용 범위의 한계로 인하여 8비트와 16비트에 머무르고 있었다.

그러나 1980년대 중반 이후로 I/O 콘트롤, 데이터 관리, 통신과 같은 강력한 계산 능력을 필요로 하는 응용 분야가 늘어나면서 32비트 내장형 프로세서에 대한 요구가 증가되기 시작하였다. 32비트 내장형 프로세서는 컬러 복사기, 프린터, 네트웍 브리지, 개인정보단말기(PDA), CDMA, 주문형 비디오(VOD), 비디오 게임, 인터액티브 텔레비전 등에 응용될 수 있으며 이러한 내장형 응용들은 다목적용 프로세서의 특징인 빠른 수행 속도보다는 내장형 응용에 적합한 효율적인 실시간 처리를 요구한다.[1] 표 1은 지금까지 개발된 32비트 내장형 프로세서의 특징을 요약한 것이다.

표 1. 32비트 내장형 프로세서의 특징 요약
Table 1. Summary of 32-bit embedded processors

| processor | V810 | SH7032 | ATT92010 | ARM610 | E1 |
|---|---|---|---|---|---|
| Developer | NEC | Hitachi | AT&T | ARM | Hyperstone |
| Clock(MHz) | 25 | 20 | 30 | 25 | 40 |
| MIPS(MIPS) | 17 | 16 | 20.3 | 15 | 32 |
| Power Consumption(mW) | 500 | 500 | 900 | 500 | 500 |
| Total Transistors | 240,000 | 593,000 | 419,000 | 359,000 | 85,000 |
| Technology($\mu$m) | 0.8 | 0.8 | 0.9 | 0.8 | 1.0 |
| Chip Size($mm^2$) | 7.7×7.7 | 10.8×10.1 | 11.1×8.49 | 8.4×8.4 | 6.7×7.4 |

PDA와 같은 응용 분야에 대해 최적의 프로세서를 설계하기 위해서는 응용 분야의 동작 특성을 고려하여 아키텍처의 사양을 결정하여야 한다. 따라서 응용 분야에 적합한 명령어 군과 내부 기능 블럭, 그리고 파이프라인 구조가 선택되어져야 한다. 또 내장형 응용 시스템은 다양한 주변기기(A/D 변환기, 내부 타이머, LCD 제어기, 그래픽 컨트롤러 등)의 기능적인 통합과, 버스/메모리 관리와 필요에 따라서는 FPU와 같은 특수한 기능을 필요로 할 수 있으며 이러한 기능들은 응용 분야의 특수성에 따라 결정된다. PDA와 같은 내장형 응용은 전력 소비가 적어야 하고 다양한 주변 장치를 내장할 수 있도록 프로세서의 크기가 작아야 한다. 또 저전력 소비를 위해 프로세서의 정지 모드를 필요로 하므로 완전한 CMOS 정적회로로 설계되어야 한다. 이외에도 높은 연산 처리 능력과 다양한 응용 소프트웨어의 설계가 가능하여야 하며 소프트웨어를 저장하기 위한 메모리의 크기가 한정되어 있으므로 응용 프로그램은 되도록 작은 크기의 코드를 가져야 한다.[2]

본 논문에서는 PDA 응용을 위한 32비트 RISC 프로세서 코어를 설계하였다. 설계된 프로세서는 특정 응용에 적합한 명령어 세트, 다양한 인터럽트에 대한 빠른 처리, 저전력 소모 등의 특성을 갖고 있으며 다른 내장형 응용이나 DSP와 같은 응용에도 사용될 수 있다. 이 프로세서는 3단 명령어 파이프라인, 조건부 명령어 실행, 블럭 데이터 전송, 하드웨어를 이용한 빠른 곱셈, 특히 MAC(Multiplication and Accumulation)명령의 지원과 같은 특징을 가지고 있다.

## Ⅱ. 명령어 군의 설계

응용 프로그램을 구성하는 명령어의 수를 줄이는 것은 제한된 메모리만을 이용할 수 있는 응용 분야에 적합하다. 특히 PDA와 같은 포터블 응용에서는 메모

리 확장이 거의 불가능하므로 되도록 응용 프로그램의 크기가 작을 수록 유리하다. RISC 형태의 프로세서의 단점은 단일 명령이 수행해야 하는 동작을 단순화시킴으로써 단일 명령 수행 시간을 줄이는 대신에 응용 프로그램의 코드 크기가 증가하는 것이다. 따라서 응용 프로그램의 크기를 줄이기 위해 단순한 동작을 복합시켜 실행하는 CISC적인 명령어의 이용이 도움이 될 수 있다.[3][4][5][6]

그림 1은 본 논문에서 설계한 프로세서의 명령어 포맷이다. 명령어 군은 RISC 형태로 32비트의 필드로 고정되었으며 모두 11개의 기본 형태로 구성된다.

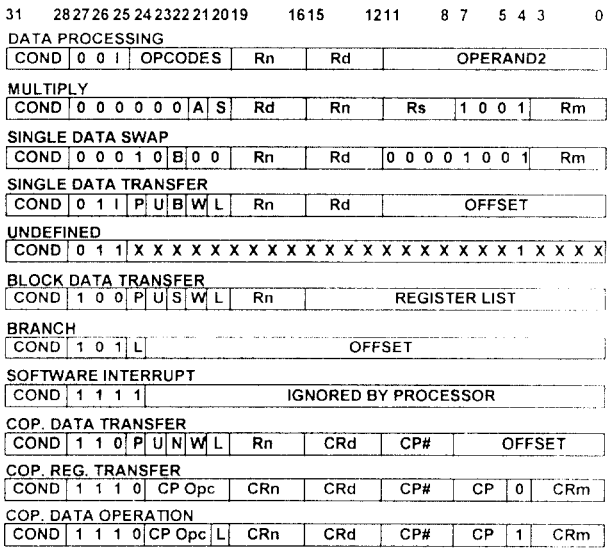



그림 1. 명령어 군
Fig. 1 Instruction set

응용 프로그램의 크기를 줄일 수 있는 명령어 군을 갖도록 데이터 처리 명령에서는 ALU연산과 쉬프터 동작을 한 명령어에 수행할 수 있고, 메모리 읽기/쓰기 명령에서 한 명령어로 16개까지의 메모리 읽기/쓰기가 가능하도록 하였다. 이러한 명령은 인터럽트 처리시 빠른 컨텍스트 스위칭을 가능하게 한다. 또 곱셈 명령을 사용하여 단일 곱셈 명령어로 32비트 정수 곱셈의 수행이 가능하도록 하였다. 특히 MAC와 같은 연산을 하드웨어적으로 빠르게 처리하는 기능은 DSP응용에도 적합하다. 그림 2는 설계된 명령어 군에 의한 프로그램의 예이다.

| Common Program code | Traditional RISC Code | Code of Designed Porsessor |
|---|---|---|
| Multiply by 3 | SHL R1, R0, 1<br>ADD R1 R1, R0 | RSB R1, R1, R1 LSL #2 |
| Logical OR<br>if(r0==p \|\| r1==q)<br>then goto label | CMP R0, #p<br>BEQ label<br>CMP R1, #q<br>BEQ label | CMP R0, #p<br>CMPNE R1, #q<br>BEQ label |
| If statement<br>if(ptr!=0) ptr = ptr ->next | CMP R1 0<br>BEQ skip<br>NOP<br>LOAD R1, R1, #next<br>skip: | TEQ R1, #0<br>LDRNE R1, [R1], #next |

그림 2. 명령어 사용 예
Fig. 2 The examples of instructions

명령어 군의 가장 큰 특징은 모든 명령어가 조건부로 실행된다는 것이다. 이를 위하여 명령어 코드 필드의 상위 비트 중 4비트([31:28])는 조건 코드로 구성된다. 그러나 이것은 모든 명령어에 조건 코드가 4비트를 차지함으로써 명령어 필드 자원의 낭비가 될 수 있다. 예로 레지스터 주소 할당에 4비트만을 이용함으로써 프로그램이 참조할 수 있는 레지스터가 16개로 한정된다는 단점이 있다.

보통 일반적인 응용 프로그램의 경우 분기 명령이 전체 프로그램 크기의 20%를 차지하며 이중 대부분이 조건부 분기 명령들이다.[3][4][5] 명령어의 조건부 실행은 이러한 조건부 분기 명령과 같은 효과를 가지며 보통 명령어와 분기 명령어를 하나의 명령어로 수행하게 되므로 그림 2에서와 같이 코드 크기를 줄이는 효과를 얻는다.

또한 분기 명령어는 프로세서가 코드를 실행할 때 코드의 순차적인 수행에 영향을 준다. 즉 파이프라인 구조를 갖는 프로세서에서는 분기 명령어의 타겟 명령어에 대한 명령어 페치가 분기 명령어 페치의 바로 다음 사이클에 이루어지지 못하므로 분기 명령의 실행이 지연되는 결과를 낳는다. 특히 분기 명령어 다음으로 되돌아 와야 하는 콜(call)과 같은 명령어는 프로그램의 비순차적인 수행에 의해 프로세서의 수행 시간을 낭비하게 된다. 이것을 해결하기 위해 소프트웨어적인 방법과 하드웨어적인 방법이 많이 연구되고 있다. 하지만 조건부 실행 명령은 명령어 자체의 수행 여부에만 영향을 주므로 코드 수행 순서에 변화가 없게 된다. 따라서 프로세서의 수행 사이클의 낭비를 줄일 수 있다. 또 이러한 순차적인 명령어 수행

은 DRAM의 페이지 모드에서와 같이 메모리 억세스에 걸리는 시간을 줄일 수 있다. 즉 메모리 억세스 주소의 순차적인 특성 여부를 미리 결정함으로써 DRAM의 페이지 모드를 최대로 이용할 수 있다.

## Ⅲ. 파이프라인의 구조

설계된 컨트롤러는 F(Fetch)-D(Decode)-E(Execute)의 3단 파이프라인 구조를 갖는다. 3단 형태의 파이프라인에서는 레지스터 읽기/쓰기가 모두 E-stage에서 이루어지므로 파이프라인 각 단 간의 데이터 의존성(data dependency)이 발생하지 않으며 따라서 파이프라인을 제어하기 위한 하드웨어가 단순해진다.[7][8]

그러나 메모리 읽기/쓰기, 곱셈 명령과 같은 다중 사이클 명령은 E-stage에서 한 사이클 내에 처리될 수 없으므로 어드레스 계산이나 레지스터로부터의 연산항 페치(operand fetch) 또는 메모리 억세스를 위한 별도의 내부 사이클이 첨가되도록 하였다. 단 파이프라인의 효율적인 이용을 위하여 다중 사이클 명령 수행 동안 한 개의 명령을 프리페치(prefetch)하여 명령어 프리페치 버퍼에 넣어 두는 방식을 이용하였다.[7][8]

그림 3(a)는 설계된 3단 파이프라인 구조에서 명령어가 실행되는 과정을 나타낸다. 일반 명령의 경우에는 F-D-E의 3단의 파이프라인 단계를 거쳐서 연산이 완료되지만, 다수 개의 클럭 사이클이 필요한 명령(다중 사이클 명령)의 경우, 다수의 내부 사이클(internal cycle)을 필요로 하므로 뒤따르는 명령은 미리 가져온 명령어를 저장하는 명령어 버퍼(instruction prefetch buffer)에 머물게 된다.[9] 그림 3(b)는 다중 사이클 명령이 실행될 때의 프리페치 버퍼, 명령어 레지스터, 그리고 PC(Program Counter)의 내용과 파이프라인의 동작이다.

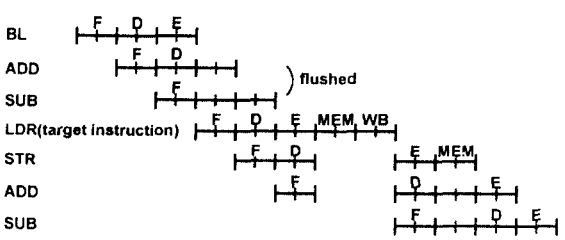



그림 3(a). 파이프라인 동작의 예
Fig. 3(a) Pipelining example

| | | | | | | |
|---|---|---|---|---|---|---|
| LDR(I1) | F1 | D1 | E1 | | | |
| I2 | | F2 | D11 | E11 | | |
| I3 | | | F3 | D12 | E12 | |
| ‘ | | | | + | D2 | E2 |
| • | | | | | • | D3 | E3 |
| I4 | | | | | | I4 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| P_BUF | | | I3 | I3 | | |
| IR | I1 | I2 | I2 | I2 | I3 | I4 |
| DR | | I1 | I1 | I1 | I2 | I3 |
| ER | | | I1 | I1 | I1 | I2 |

| | | | | | | |
|---|---|---|---|---|---|---|
| PC | PC+4 | PC+8 | PC+12 | PC+12 | PC+12 | PC+16 |
| AR | PC+4 | PC+8 | alu | PC+12 | PC+12 | PC+16 |

그림 3(b). 다중 사이클 명령어 수행시 명령어 파이프라인
Fig. 3(b) Instruction pipeline in multi-cyle instruction

그림 3. 3단 파이프라인
Fig. 3 3-stage pipeline

그리고 표 2에 파이프라인 단에서의 동작을 요약하였다. 각 파이프라인 단은 비겹침 이중 클럭(nonoverlapped two-phase clock)에 의해 두 부분으로 나뉘어지며 각 클럭 페이즈 타이밍에 대해 동작이 세분화된다.

표 2. 각 파이프라인 단에서의 동작
Table 2. The operation at each pipeline stage

| | | |
|---|---|---|
| F-I | $\varphi_1$ | Calculate an instruction address |
| | $\varphi_2$ | Address bus ← A calculated instruction address |
| F | $\varphi_1$ | Off-chip memory access |
| | $\varphi_2$ | Instruction register ← An instruction from off-chip memory |
| D | $\varphi_1$ | Decode-stage instruction register ← Instruction register |
| | $\varphi_2$ | Decode an instruction |
| E | $\varphi_1$ | Operand fetch and Shifter operation<br>· A bus ← Rs1<br>· B bus ← Rs2 or the output of Field extractor |
| | $\varphi_2$ | ALU operation and Result write operation<br>· Result bus ← ALU<br>· Destination register ← Result bus |

## Ⅳ. 데이터 패스의 설계

설계된 프로세서는 내장형 응용에 적합하도록 기능적인 면과 전력 소비 등을 고려하여 단순하면서도 최대의 기능을 갖도록 데이터 패스 블럭을 설계하였다. 각 데이터 패스는 설계된 명령어 군이 최대 3개의

연산항을 필요로 하기 때문에 3개의 내부 버스가 있으며 쉬프터와 ALU 동작은 연속적으로 발생하게 된다. 그림 4는 설계된 프로세서의 전체 블럭도이다.

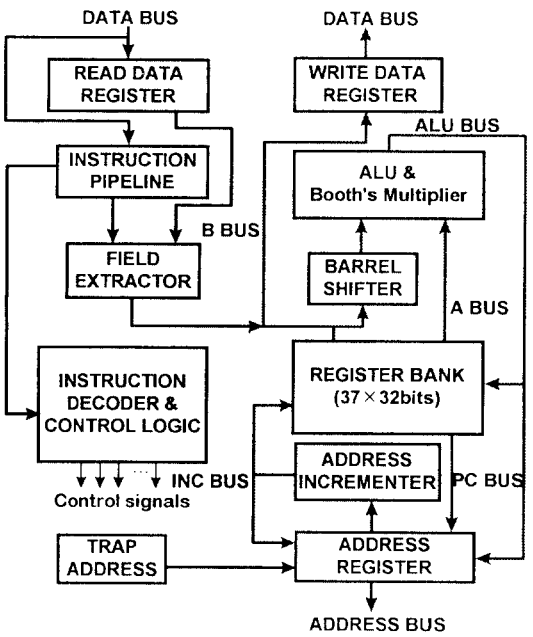



그림 4. CPU 전체 블럭도
Fig. 4 CPU Block Diagram

### 1. ALU의 설계

ALU의 설계에서는 명령어 군의 분석을 바탕으로 필요한 기능을 결정하게 된다. 설계된 ALU에서는 데이터 처리 명령에 대한 논리 산술 연산, 데이터 메모리 억세스에 필요한 주소 계산, 곱셈 명령 수행 동안 부스 인코딩(booth encoding) 결과에 의해 부분합과 쉬프트된 피승수의 덧셈 동작, 분기 명령의 주소 계산, 예외 처리나 분기 명령의 실행시 귀환될 주소의 계산을 32비트 연산항에 대해 수행한다. 이와 같이 ALU 연산 기능이 기존 RISC에 비해 복잡한 이유는 보다 간결한 코드를 얻고 하드웨어의 기능적 반복을 막기 위해 다양한 연산 기능이 하나의 ALU에서 수행될 수 있도록 명령어를 정의했기 때문이다.

ALU에 사용한 가산기 구조로는 규칙성, 면적, 그리고 속도 등을 고려하여 4비트 단위로 group carry를 발생시키는 carry-look-ahead 가산기를 채택하였다.[10] 그림 5는 ALU의 내부 구조를 나타낸다.

ALU는 입력 발생기(input generator), GP 발생기(carry generator/propagander generator), 캐리 체인(carry chain), 합계 발생기(sum generator)로 구성되어 있다. 입력 발생기 블럭은 다양한 ALU 동작을 위한 입력 패턴을 발생시키는 부분이다. 제어신호 f7~f0는 다양한 입력 패턴 발생과 출력을 제어하여 필요한 연산이 ALU에서 이루어지도록 한다. GP 발생기에서 발생시킨 carry generator와 propagander는 4비트 그룹으로 carry-chain인을 거치게 된다. 이렇게 함으로써 carry-chain에 의한 지연과 게이트의 fan-in/fan-out을 줄일 수 있다.

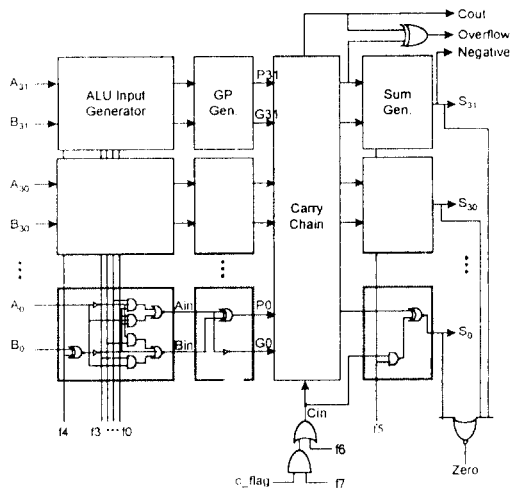



그림 5. ALU 블럭
Fig. 5 ALU block

### 2. 쉬프터의 설계

본 논문에서 설계한 쉬프터는 ALU 동작과 직렬로 수행된다. 즉 ALU의 두 연산항 중 하나는 레지스터로부터 나온 출력이고 다른 하나는 쉬프터의 출력이다. 따라서 쉬프터와 ALU의 동작이 파이프라인에서 한 클락 사이클 이내에 수행되어야 하므로 고속의 쉬프터가 요구된다.

본 논문에서는 cross-bar 스위치 형태의 배럴 쉬프터를 이용하였다.[10] 쉬프터는 산술 명령어에서 좌우 쉬프트/회전 동작, 메모리 읽기/쓰기 명령에서 주소 계산을 위한 쉬프트, 메모리에서 읽은 데이터의 정렬, 그리고 곱셈 명령에서 피승수의 쉬프트 동작에 이용

된다. 따라서 좌우 쉬프트 연산이 모두 필요하므로 32비트의 데이터를 쉬프트하기 위해 32×32 개의 스위치와 2×32 개의 제어신호가 필요하다.

Cross-bar 스위치에서는 논리 및 산술 쉬프트에서 요구되는 '0' 확장, 부호 확장이 지원되지 않으므로 스위치 네트워크 이외에도 쉬프트 시 상위 또는 하위 비트의 '0' 확장, 부호 확장을 제어하기 위한 매스크 발생기를 필요로 한다. 설계된 매스크 발생기는 5단 구조로 되어 있으며 규칙적인 확장이 가능한 구조를 갖고 있다. 여기서 발생된 신호는 스위치 네트워크의 출력과 부호, zero중에서 하나를 선택하는 제어신호로 이용된다. 그림 6은 설계된 매스크 발생기이고 그림 7은 쉬프터의 전체 블럭을 보여주고 있다.

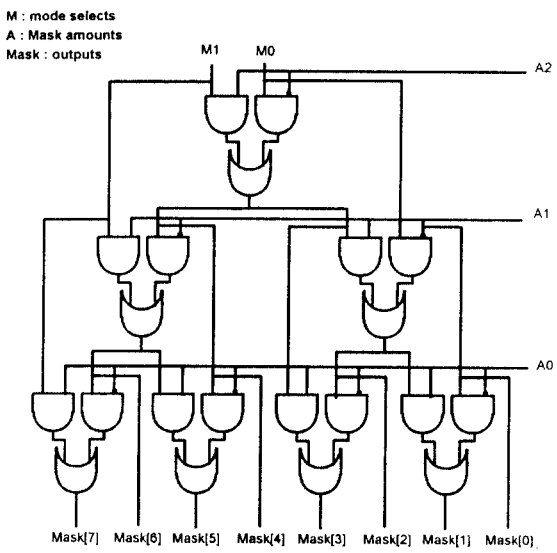



그림 6. 8비트 매스크 발생기
Fig. 6 8-bit mask generator

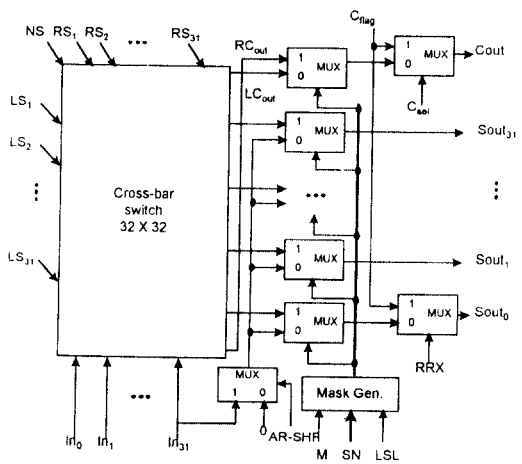



그림 7. 쉬프터 블럭
Fig. 7 Shifter block

### 3. Multiplication을 위한 하드웨어 설계

설계된 프로세서는 하드웨어적인 방법을 이용하여 두 개의 32비트 레지스터를 곱하여 32비트의 곱셈 결과를 얻는다. 하드웨어의 크기를 고려하여 병렬 승산기를 사용하지 않고 ALU와 쉬프터, 그리고 booth encoder를 제어하기 위한 별도의 하드웨어 만을 두는 방식을 사용하였다.

ALU와 쉬프터를 이용한 곱셈 과정은 그림 8과 같다. 곱셈 명령의 첫 사이클에서 승수를 승수 레지스터에 저장하고 승수를 3비트씩 booth encoding을 위한 블럭으로 이동시킨다. 이때 조기 종료 감지를 위한 신호 회로는 승수가 '0' 또는 '1'임을 확인하여 필요하지 않은 곱셈 동작의 진행을 방지한다. Booth encoder에서 발생된 신호는 쉬프터와 ALU를 제어하여

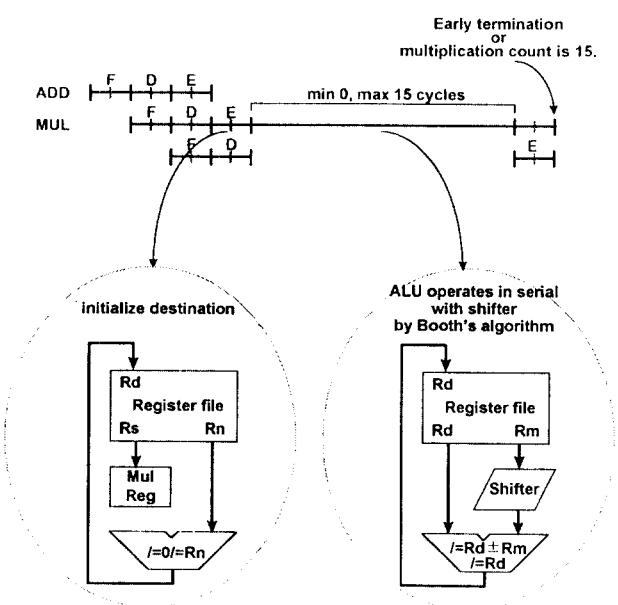



그림 8. booth 알고리즘을 이용한 곱셈 과정
Fig. 8 Multiplication using 3-bit Booth's algorithm

피승수와 부분합을 가산 또는 감산하게 된다. 이 때 다른 곱셈기와는 달리 부분합을 쉬프트하지 않고 피승수를 쉬프트하게 되므로 곱셈의 조기 종료가 가능하다. 곱셈에 수행되는 시간은 2~17 사이클이다.

그림 9는 곱셈 알고리듬을 구현하기 위한 승수 레지스터의 구조이다. 승수 레지스터는 매 사이클마다 승수를 2비트씩 우측으로 쉬프트하게 되고 승수 레지스터의 출력은 조기 종료 감지 회로와 booth encoder의 입력으로 이용된다.

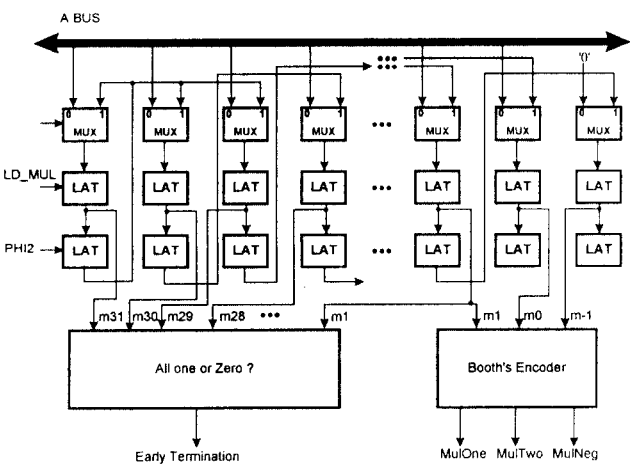



그림 9. 승수 레지스터
Fig. 9 Multiplier register

## 4. Register File의 설계

레지스터 파일의 구조는 프로세서의 성능에 많은 영향을 주는 부분이다. 기존의 SPARC RISC 프로세서와 같은 중첩된 레지스터 윈도우 개념을 사용한 레지스터 파일의 경우, 칩 면적을 너무 많이 차지하며 전력 소모가 많다. 그리고 빠른 컨텍스트 스위칭(context switching)을 필요로 하는 응용분야에서는 레지스터 파일의 내용을 메모리에 저장하는 동작은 큰 부담이 된다.[11][12] 따라서 본 논문에서는 중첩 레지스터 윈도우 구조보다는 6가지의 프로세서 동작 모드에 따라 레지스터를 할당할 수 있는 레지스터 뱅크 구조를 이용하여 인터럽트 처리가 빠르게 수행될 수 있도록 하였다.

각 모드에서 개별적으로 사용할 수 있는 레지스터 수는 수행되고 있는 동작의 특성을 고려하였으며 전체적으로는 각 모드당 16개의 32비트 레지스터를 사용할 수 있다. 각 모드에서만 사용 가능한 레지스터를 이외에는 사용자 모드에서 사용하는 레지스터를 공유하여 이용한다. 그림 10은 6가지 동작 모드에 대한 레지스터 프로그래밍 모델을 나타낸다. 범용 레지스터 이외에도 프로세서는 6개의 32비트 상태 레지스터를 가지고 있으며 동작 모드 스위칭이 발생할 때마다 프로세서의 상태를 저장하게 된다.

General Registers and PC

| User | FIQ | Supervisor | Abort | IRQ | Undefined |
|---|---|---|---|---|---|
| R0 | R0 | R0 | R0 | R0 | R0 |
| R1 | R1 | R1 | R1 | R1 | R1 |
| R2 | R2 | R2 | R2 | R2 | R2 |
| R3 | R3 | R3 | R3 | R3 | R3 |
| R4 | R4 | R4 | R4 | R4 | R4 |
| R5 | R5 | R5 | R5 | R5 | R5 |
| R6 | R6 | R6 | R6 | R6 | R6 |
| R7 | R7 | R7 | R7 | R7 | R7 |
| R8 | R8_fiq | R8 | R8 | R8 | R8 |
| R9 | R9_fiq | R9 | R9 | R9 | R9 |
| R10 | R10_fiq | R10 | R10 | R10 | R10 |
| R11 | R11_fiq | R11 | R11 | R11 | R11 |
| R12 | R12_fiq | R12 | R12 | R12 | R12 |
| R13 | R13_fiq | R13_svc | R13_abt | R13_irq | R13_und |
| R14 | R14_fiq | R14_svc | R14_abt | R14_irq | R14_und |
| R15(PC) | R15(PC) | R15(PC) | R15(PC) | R15(PC) | R15(PC) |

Program Status Registers

| | | | | | |
|---|---|---|---|---|---|
| CPSR | CPSR | CPSR | CPSR | CPSR | CPSR |
| | SPSR_fiq | SPSR_svc | SPSR_abt | SPSR_irq | SPSR_und |

그림 10. 6가지 모드에 대한 레지스터 뱅크 구조
Fig. 10 The register bank structure for 6 operation modes

범용 레지스터는 모두 31개의 32비트 레지스터로 구성되며 프로세서의 현재 모드에 따라 최대 16까지 이용할 수 있다. 레지스터 디코더 블럭은 명령어에 있는 레지스터 주소를 해석하고 현재의 동작 모드를 나타내는 상태 레지스터의 모드 비트를 이용하여 필요한 레지스터를 읽거나 쓰게 된다.

## 5. field extractor, 주소 레지스터, 데이터 읽기/쓰기 레지스터의 설계

이외의 주요한 블럭은 외부에서 들어오는 연산항을 제어하기 위한 field extractor, 주소 발생을 제어하는 주소 레지스터, 데이터를 메모리에 읽고 쓸 때를 위한 데이터 읽기/쓰기 레지스터가 있다.

Field extractor는 명령어 필드에 포함된 데이터를 부호 확장이나 '0' 확장을 하여 32비트로 만든다. 데이터 처리 동작은 명령어의 종류에 따라 다르다. 예로 분기 명령어에 대해서는 부호를 가진 24비트 오프셋의 부호 확장과 워드 단위로 정렬된 메모리 억세스를 위한 하위 2비트의 '0'처리가 이루어진다.

주소 레지스터는 메모리 억세스를 위한 주소를 발생시키는 블럭이다. 본 논문에서 설계한 프로세서는 명령어와 데이터 버스가 분리되지 않은 폰 노이만 구조이므로 데이터 주소와 명령어 주소가 같은 레지스터에 있게 된다.

데이터 읽기/쓰기 레지스터는 메모리 억세스시 데이터를 임시로 저장하는 역할을 한다. 바이트 데이터를 메모리에 저장할 때는 저장될 데이터 중 하위 8비트가 외부로 출력되기 전에 상위 24비트로 복사되며 외부 메모리 제어회로가 주소의 하위 2비트로 메모리에 쓰여질 바이트를 결정하게 된다.

## V. 제어부의 설계

### 1. 명령어 파이프라인과 상태 제어회로의 설계

제어부는 hardwired 제어 기법을 이용하여 설계하였으며 명령어 파이프라인과 명령어 디코더, 그리고 상태 제어 회로와 타이밍 회로로 구성된다.

명령어 파이프라인은 프리페치 버퍼와 각 파이프라인별 명령어 레지스터로 구성된다. 이 레지스터에는 현재 실행중인 명령어와 이미 메모리로부터 페치되어 실행을 기다리는 명령어를 저장하게 된다. 명령어 파이프라인의 구조와 타이밍은 그림 11과 같다. 명령어 프리페치 버퍼는 수행중인 명령어가 다중 사이클일 때 페치된 명령어를 다중 사이클 명령어의 실행이 종료될 때까지 유지한다.

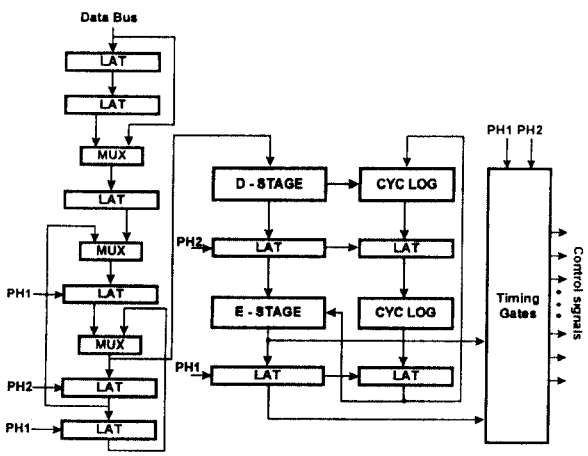



그림 11. 명령어 파이프라인과 제어 타이밍
Fig. 11 Instruction pipeline and control timing

Hardwired 제어 기법을 파이프라인 프로세서에 적용하는 방식으로 시간 정적 제어 방식과 데이터 정적 제어 방식이 제안되고 있다. 시간 정적 제어 기법은 데이터 패스에 대한 제어 신호가 파이프라인된 구조로 된 각각의 데이터 패스 제어 시점에서 발생되는 방법이다. 이와는 달리 본 논문에서 채택한 데이터 정적 제어 방식은 제어 신호 발생이 실제로 사용되는 파이프라인 단계에 관계없이 디코딩 단계에서 모두 발생된다.[13] 이후 레지스터로 구성된 지연 체인을 통해 실제로 사용되는 파이프라인 단계까지 이동되어 데이터 패스를 제어하게 된다. 이 방법에서는 명령어 디코더가 현재 페치된 명령어에 대한 디코딩 기능만을 수행하면 되므로 제어 회로가 단순화 된다. 또 다중 사이클이 요구되는 명령을 처리하기 위해 추가 사이클을 제어하는 상태 제어 회로를 추가하였다. 이러한 제어 기법과 타이밍은 그림 11의 오른쪽과 같다.

제어부의 사이클 상태 비트는 모두 5비트로 구성된다. 다섯번째 비트는 파이프라인의 플러쉬 모드(flush mode) 상태를, 네번째 비트는 블럭 전송 명령어의 실행중에 발생한 abort 상태를, 그리고 세번째에서 첫번째 비트는 정상적인 명령어(normal instruction)의 실행을 제어하게 된다. 이 상태 비트는 파이프라인의 진행과 함께 상태 천이를 하게되며 파이프라인의 흐름을 제어하게 된다. 그림 12는 상태 사이클에 대한 단일 사이클 명령어와 다중 사이클 명령어, 그리고 파이프라인 플러쉬 모드시 상태 천이를 나타낸 것이다.

상태 제어 회로에서 발생된 상태 비트는 명령어 디코더에서 발생된 신호와 함께 파이프라인 각 단의 동작 블럭을 제어하는 신호를 만드는 기준이 된다. 명령어 디코더는 레벨 1 디코더에서 명령어의 종류를 결정하고 레벨 2에서 각각의 데이터 패스 블럭에서 필요로 하는 제어신호를 발생 시키는 두 레벨로 구현하였다. 이 이외에 제어가 복잡한 블럭 전송 명령과 곱셈 명령은 별도의 하드웨어를 이용하여 상태 사이클을 제어함으로써 제어회로가 복잡해지는 것을 방

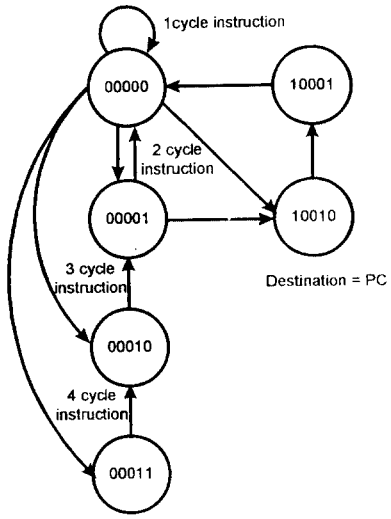



그림 12. 사이클 상태 천이도
Fig. 12 The diagram of the transfer of cycle state

지하였다.

그림 13은 명령어 디코더의 구조를 보여주고 있다. 본 논문에서 설계한 프로세서는 모든 명령어를 조건부로 실행하기 때문에 명령어의 수행 여부를 결정하기 위한 조건 평가회로가 필요하다. 조건 평가회로는 분기 명령의 주소 구동 시점이 파이프라인의 실행단(E-stage)이므로 그림 13에서와 같이 조건 평가회로가 E-stage에 있게 된다. 단 조건 평가회로의 결과가 수행 방지(false)로 나타났을 때 명령어 디코딩이 이미 이루어 졌기 때문에 현재 E-stage에서 수행중인 명령어에 대한 플러쉬(flush)처리가 필요하다.

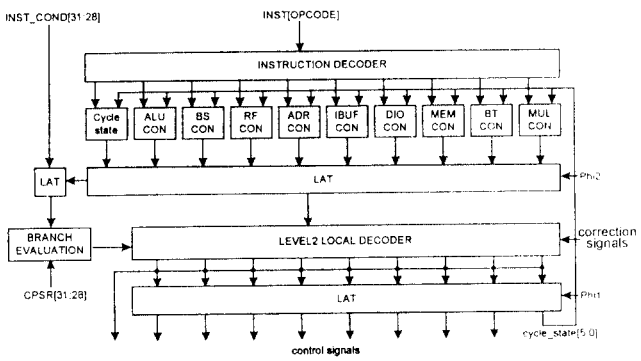


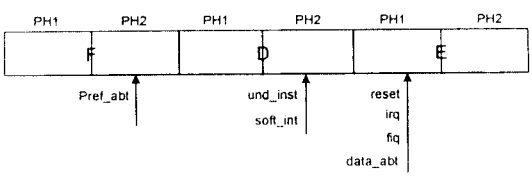

그림 13. 명령어 디코더
Fig. 13 Instruction decoder

## 2. 예외 처리 회로

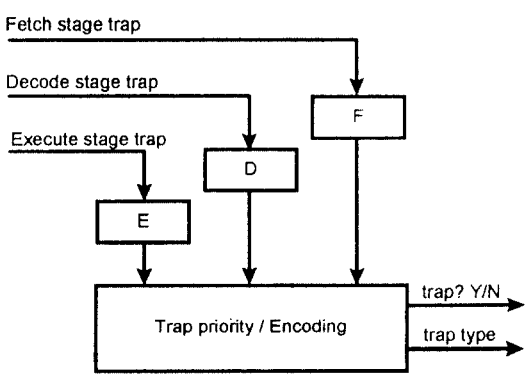
프로세서의 예외상황은 크게 트랩(program interrupt)과 외부 인터럽트(external interrupt)로 나뉘어 진다. 여기서 트랩은 명령어 수행 또는 페치 과정에서 발생하는 예외상황이며 인터럽트는 외부 장비에 의해 발생되는 예외상황이다.[14][15] 본 논문에서 설계한 프로세서는 예외상황을 처리하기 위한 PC(Program Counter) 체인을 별도로 두지 않고 하나의 PC만 존재하는 구조를 취하고 있다.

각 파이프라인 단에서 예외상황이 발생되면 E-stage까지 지연시키지 않고 발생 즉시 예외상황의 처리 여부를 결정하도록 하였다. 즉 발생된 예외상황에 대해 우선순위와 예외 처리를 위한 벡터, 그리고 예외상황 종류를 결정하여 예외 처리 루틴으로 갈 수 있도록 PC와 상태 레지스터를 수정하게 된다.

각 예외상황의 종류와 인터럽트 감지 시점은 그림 14(a)와 같으며 감지된 예외 처리 과정은 그림 14(b)와 같다.




(a) 예외상황 감지 시점
(a) The detection timing of exception




(b) 예외 처리 과정
(b) The process of exception detected

그림 14. 예외 처리
Fig. 14 Exception handling

### 3. 메모리 인터페이스

프로세서보다 느린 외부 메모리 억세스와 관련된 신호는 실제 메모리 억세스가 발생하는 시점보다 한 클럭 사이클 또는 반 클럭 사이클 만큼 미리 발생시켜 메모리 억세스 동작을 제어할 수 있도록 하였다. 이렇게 함으로써 DRAM의 페이지 모드를 효율적으로 이용할 수 있다. 즉 연속적인 메모리 억세스의 주소가 동일하거나 4만큼 변할 때 이러한 정보를 외부 메모리에 한 사이클 미리 전달함으로써 DRAM이 row address는 고정시키고 column address만 변화시키는 페이지 모드를 이용하여 데이터 입출력 동작을 할 수 있게 하였다. 이에 대한 사이클 타이밍은 그림 15와 같다. 여기서 nMREQ는 메모리 억세스 여부를, SEQ는 주소의 순차성 여부를 나타내며 이 두 신호의 조합에 의해 다음 사이클이 어떤 형태인지 미리 예측하게 된다.

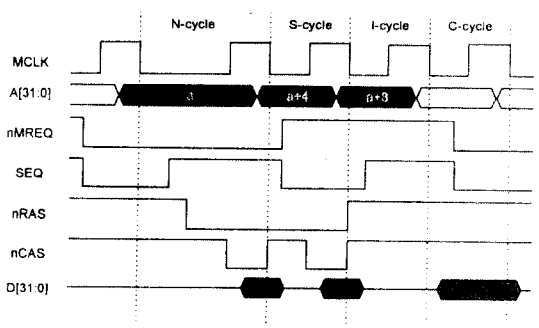



그림 15. 메모리 억세스 사이클 타이밍
Fig. 15 Memory access cycle timing

## Ⅵ. Verification과 Simulation

본 논문에서는 설계 시간을 단축시키고 오류를 줄일 수 있도록 HDL의 기술에 의한 탑-다운(top-down) 설계 방식을 적용하였다. HDL에 의한 설계 방식은 상위 수준(behavioral level)에서 특정 기능을 기술하고 ASIC 합성기에 의해 이를 자동 합성하는 방식이다. 그러나 상위 수준으로 기술할수록 회로 수준에서는 기능상으로는 동일하나 설계자의 의도와 다른 형태로 합성될 수 있고, ASIC 합성기의 성능에 제한적인 HDL 기술만이 합성 가능하다는 단점이 있으므로 본 논문에서는 모든 기능 블럭을 구조적 수준(structural level)으로 기술하고 이를 합성하는 방식을 채택하였다. 또 latch나 multiplexer 또는 tri-state buffer와 같이 각 블럭에서 공통적으로 자주 사용되는 기능 블럭은 HDL에 의한 라이브러리를 구축하여 이를 최대한으로 이용하였다.

HDL을 이용해 설계된 기능 블럭의 검증은 각 블럭의 특성에 맞게 인위적으로 작성된 테스트 패턴을 적용하였다. HDL에 의해 개별적으로 설계되고 검증된 기능 블럭은 전체 블럭으로 합쳐진 후 명령어 수준에서의 검증을 거치게 된다. 이때 명령어 수준에서의 검증을 위해 설계된 프로세서를 이용한 의사 시스템을 구성하여 시스템 수준에서의 검증이 가능하도록 하였다. 그림 16은 프로세서 검증을 위해 설계된 의사 시스템의 구조이다. 의사 시스템은 설계된 프로세서와 메모리, 메모리 컨트롤러와 시스템 수준에서 필요한 제어 신호를 발생시키는 환경 제어 블럭으로 구성되며 이 시스템 환경 제어 블럭에서는 프로세서의 초기화 신호 및 인터럽트를 발생시키고 프로세서가 필요로 하는 보조 프로세서(coprocessor)로서의 역할을 수행하게 된다.

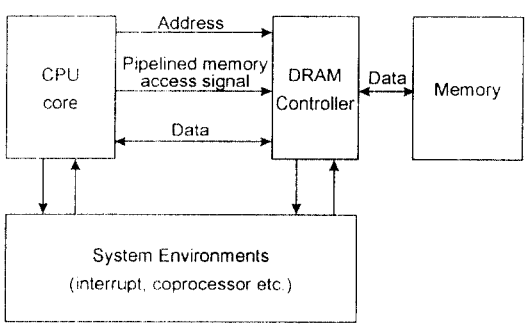



그림 16. 프로세서 검증을 위한 의사 시스템
Fig. 16 The pseudo system for processor verification

이렇게 시스템 수준에서의 프로세서 검증은 프로세서의 기능 검증 뿐만 아니라 프로세서의 성능을 평가할 수 있다는 장점이 있다. 본 논문에서는 시스템 수준의 시뮬레이션을 위해 몇 가지 알고리즘을 C 언어를 이용한 응용 프로그램으로 작성하여 테스트 벡터를 발생시켰다.

그림 17은 다양한 응용 프로그램에 의해 측정된 프

로세서의 CPI(Cycles Per Instruction)이다. CPI의 계산은 다음과 같다.

$$CPI = \frac{CPU\ time \times clock\ rate}{instruction\ count} = \frac{Cycles}{instruction\ count}$$

프로세서의 측정된 평균 CPI는 1.74정도이며 1.3~1.5정도인 다른 RISC 프로세서와 비교할 때 약간 큰 편이다. 이것은 세분화되지 않은 3단의 파이프라인과 다중 사이클 명령, 특히 블럭 전송 명령과 곱셈 명령에 의한 영향이다. 프로그램 실행시의 명령어를 분석해 보면 블럭 전송명령이 보통 10% 내외를 차지하고 있으며 이 명령에 의해 더욱 빠른 메모리 억세스가 가능하다.

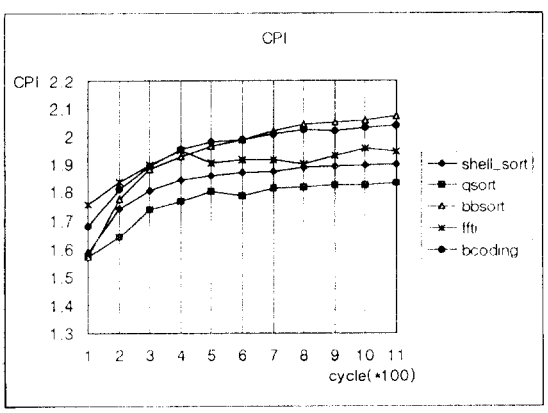
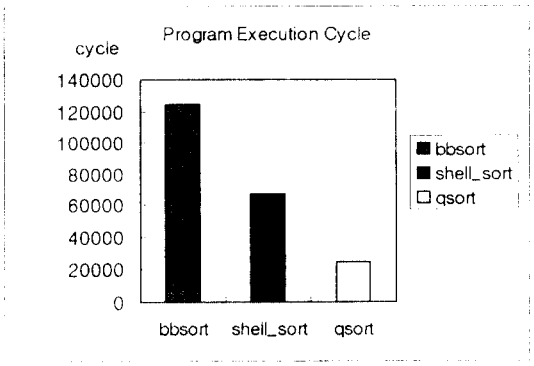


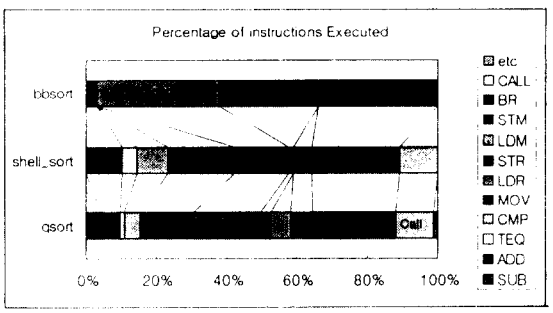

그림 17. CPI의 측정
Fig. 17 The measure of CPI

그림 18(a)는 3 개의 정렬 알고리즘에 대한 HDL 수준에서의 성능 평가이다. 그리고 그림 18(b)와 (c)에서 각 알고리즘에 대한 명령어 구성과 데이터 참조 특성 및 수행 시간을 비교하였다. 알고리즘을 구성하는 대부분의 명령어는 메모리 읽기/쓰기와 분기 명령이다. 각 알고리즘에서 블럭 전송 명령은 보통 10% 정도이며 이 명령어에 의해 메모리 읽기/쓰기 시 1.7배 정도의 성능 향상을 얻을 수 있다.

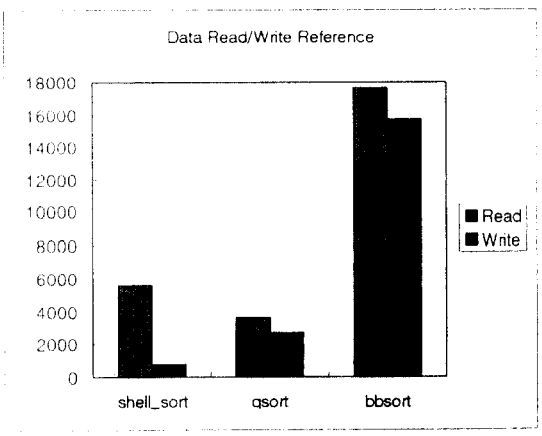
그림 19는 행렬 연산에 대해 수행된 명령어의 구성비이다. 행렬 연산과정 동안 발생된 곱셈 명령어는 쉬프터와 ALU를 이용한 곱셈기에 의해 수행되며 소프트웨어적인 방법보다 3배 이상의 성능 향상을 얻을 수 있다.




(a) 프로그램 수행 시간
(a) Program execution time




(b) 수행된 명령어의 구성비
(b) Percentage of Instructions executed




(c) 데이터 참조 회수
(c) The count of data reference

그림 18. 정렬 알고리즘의 비교
Fig. 18 The comparison of sorting algorithm

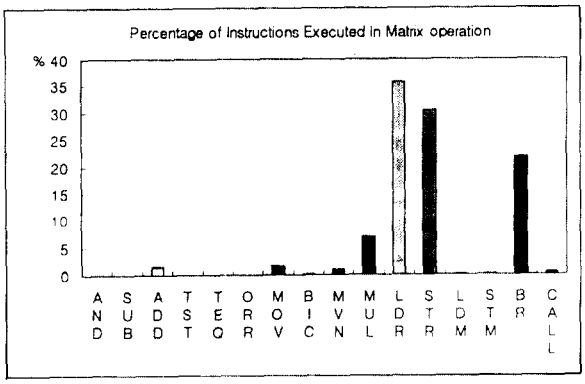



그림 19. 행렬 연산시 수행된 명령어 구성비
Fig. 19 Percentage of Instructions executed in matrix operations

HDL 수준에서 검증된 프로세서는 ASIC 합성기를 이용하여 게이트 수준으로 합성되고 동작을 검증하게 된다. 게이트 수준의 시뮬레이션을 위한 테스트 벡터는 HDL 수준에서의 검증 과정동안 자동 발생된다. ASIC synthesizer를 이용해 얻은 네트리스트를 가지고 타이밍 시뮬레이션을 실행하고 최대 지연 경로를 찾아내었다. 합성된 로직회로는 0.6$\mu$m 삼중 금속 단일 실리콘 CMOS 공정을 이용한 표준 셀 방식으로 레이아웃 되었다.

합성된 마이크로프로세서는 약 88,000만개의 트랜지스터로 구성되었으며 최대 동작 주파수는 40MHz이다. 최대 지연 경로는 ALU와 쉬프터로 구성되며 각각 9ns와 5ns의 delay를 보였다. 그림 20은 최대 지연 경로에 대한 게이트 수준의 시뮬레이션 결과이다. 칩 면적은 5.0×5.0mm$^2$이며 프로세서 코어 면적은 3.85×3.78mm$^2$이다. 표 3과 그림 21은 합성된 칩의 전기적 특성과 레이아웃이다.

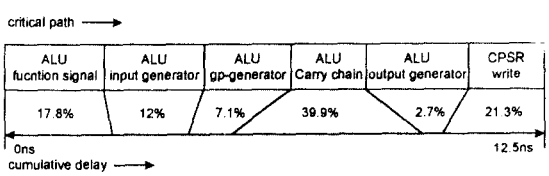



그림 20. 최대 지연 경로에 대한 게이트 수준 시뮬레이션
Fig. 20 Gate simulation of critical delay path

표 3. 전기적 특성
Table 3. Electrical characteristics

| | |
|---|---|
| Chip size | 5.0×5.0 mm$^2$ |
| Core size | 3.85×3.78 mm$^2$ |
| Core density | 0.98 gates/sq-mil |
| Estimated power consumption | 179 mW |
| No. of transistor | 88,038 |
| Operating frequency | 40 MHz |
| Number of I/O pins | Total : 100<br>Input : 13<br>Output : 40<br>Bidirectional : 32<br>VDD : 8<br>VSS : 7 |
| Package type | 100 PQFP |
| Process | 0.6 $\mu$m triple metal single-poly CMOS technology |

at 3.3V, 25℃

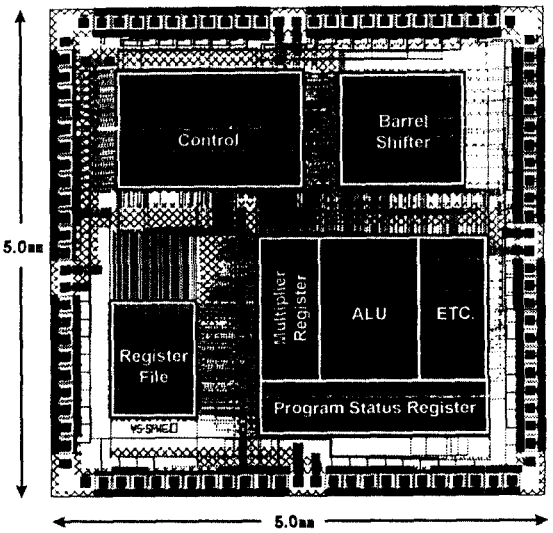



그림 21. 레이아웃
Fig. 21 Layout

## Ⅶ. 결 론

본 논문에서는 PDA와 같은 내장형 응용을 위한 콘트롤러를 설계하였다. 이 프로세서는 내장형 응용의 중요한 특성인 빠른 인터럽트 핸들링, 빠른 컨텍스트

스위칭과 저전력 소모를 지원한다. 또한 조건부로 수행 가능한 명령어 군과 블럭 전송 명령, 그리고 곱셈 명령을 이용하여 프로세서의 성능을 향상시켰다. 조건부 명령어 수행 기능은 분기 명령어의 수를 줄이고 프로그램의 순차적 실행을 가능하게 한다. 블럭 전송 명령을 이용하면 연속적인 메모리 억세스의 빠른 수행이 가능하며 인터럽트에 의한 컨텍스트 스위칭의 성능을 향상시킨다. 또 곱셈 명령은 하드웨어적인 방법으로 2~17사이클 내에 32비트 곱셈을 수행할 수 있으며 MAC과 같은 동작을 많이 사용하는 DSP와 같은 응용에 적합하다. 그리고 3단 파이프라인을 이용함으로써 데이터 의존성을 해결하기 위한 하드웨어가 필요하지 않고 제어부가 단순해지는 결과를 얻을 수 있었다. 또한 뱅크 구조로된 레지스터 파일을 이용하여 빠른 인터럽트 처리가 가능하도록 하였다.

이 프로세서는 5.0×5.0mm$^2$의 면적에 약 88,000개의 트랜지스터가 집적되었으며 0.6$\mu$m 삼중 금속 단일 폴리 공정이 이용되었다. 최대 동작 주파수는 40MHz이며 예상 전력 소비는 179mW이다.

## 참 고 문 헌


1. Kenneth Hinz and Daniel Tabak, Microcontrollers :Architecture, Implementation, and programming, New York : McGraw-Hill, 1992.
2. Ron Cates, "Processor Architecture Considerations for Embedded Controller Applications," in IEEE MICRO, 1988, pp. 28-38.
3. John L. Hennessy and David A. Patternson, Computer Architecture : A Quantitative Approach. San Mateo : Morgan Kaufmann, 1990.
4. David A. Patternson and John L. Hennessy, Computer Organization & Design : The Hardware/Software Interface. San Mateo : Morgan Kaufmann, 1994.
5. Kai Hwang, Advanced Computer Architecture : Parallelism, Scalability, Programmability, New York : McGraw-Hill, 1993.
6. Ing-Jer Huang and Alvin M. Despain, "Generating Instruction Sets and Microarchitectures from Applications," in ACM, pp. 391-396, 1994.
7. M. K. Lee, B. Y. Choi, S. H. Lee, and S. I. Son, "VLSI Design of High Performance RISC Microprocessor," in International Conference on VLSI and CAD '91, Seoul KOREA, pp. 88-91, Oct 1991.
8. M. K. Lee, B. Y. Choi, and S. I. Son, "VLSI Design of High Performance RISC," in Proceedings of ICEIC '91, Yanbian China, Aug 1991.
9. Mark Horowitz, Pawl Chow, "MIPS-X : A 20 MIPS Peak, 32 Bit Microprocessor with On chip Cache," in IEEE J. Solid State Circuits, vol. 22, pp 790-799, Oct 1989.
10. Carver Mead and Lynn Conway, Introduction to VLSI Systems, Massachusetts : Addison-Weseley, 1980.
11. Miquel Huquet and Tomas Lang, "A Reduced Register File for RISC Architectures," in Computer Architecture News, pp. 22-31, Sept 1985.
12. David G. Bradlee, Susan J. Eggers and Robert R. Henny "The Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy," in ACM, 1991.
13. Moon Key Lee, Byeong Yoon Choi, Kwang Youb Lee, and Seung Ho Lee, "Data-Stationary Controller for 32-bit Application-Specific RISC," in Proceedings of ISCAS'93, Vol. 3, pp. 1933-1936, Chicago, May 1993.
14. Hubert Kirrmann, "Events and Interrupts in Tightly Coupled Microprocessors," in IEEE micro, pp 53-66, Feb 1985.
15. James E. Smith and Andrew R. Pleszkum, "Implementing of Precise Interrupt in Pipeline Processor," in 13th Symposium on Computer Architecture, pp. 36-44, 1985.

곽 승 호(Sung Ho Kwak)정회원

1994년 2월 : 연세대학교 전자공학과 졸업(공학사)
1996년 8월 : 연세대학교 전자공학과 본대학원 졸업(공학석사)
1996년 9월~현재 : 연세대학교 전자공학과 본대학원 박사과정 재학중
※주관심분야 : 마이크로프로세서 설계, VLSI 설계

최 병 윤(Byeong Yoon Choi) 정회원

1985년 2월 : 연세대학교 전자공학과 졸업(공학사)
1987년 2월 : 연세대학교 전자공학과 본대학원 졸업(공학석사)
1992년 8월 : 연세대학교 전자공학과 본대학원 졸업(공학박사)
1993년 3월~현재 : 동의대학교 컴퓨터 공학과(조교수)
※주관심분야 : 수퍼스케일러 마이크로프로세서 설계, 신호처리 및 통신용 집적회로 설계

이 문 기(Moon Key Lee)정회원

1941년 8월 23일생
1967년 : 연세대학교 전자공학과 강사
1969년 : 광운공과대학교 전자공학과 전임 강사
1970년 : 경희대학교 전자공학과 조교수
1973년 : 경희대학교 전자공학과 부교수겸 학과장
1976년 : University of OKLAHOMA 연구소 연구원
1980년 : 한국전자기술 연구소(현 ETRI) 설계자동화(CAD) 국책연구과제 수행 책임 연구원
1983년 : 금성반도체 주식회사 비상임 기술고문
1984년 : 기아 산업 주식회사 종합연구소 비상임 기술고문
1989년 : 연세대학교 부설 아식설계 공동연구소 부소장
1990년 : 연세대학교 전자공학과 학과장
1995년 : 대한전자공학회 회장
현재 : 연세대학교 교수
※주관심분야 : 마이크로프로세서 설계, 디지탈 영상처리 칩 설계, 설계자동화, 패킷 스위칭 등