멀티미디어 휴대 단말기용 32 비트 RISC 코어 구현

정갑천*, 기용철**, 박성모**
*전남대학교 전자공학과 **전남대학교 컴퓨터공학과
gcjung@ciscom.chonnam.ac.kr

Implementation of a 32-Bit RISC Core for Multimedia Portable Terminals

Gab-Cheon Jung*, Yong-Chol Kee**, Seong-Mo Park**
*Dept. of Electronics Eng., Chonnam National Univ.,
**Dept. of Computer Eng., Chonnam National Univ.

Abstract

In this paper, we describe implementation of 32-Bit RISC Core for portable communication/information equipment, such as cellular telephones and personal digital assistants, notebook, etc. The RISC core implements the ARM®V4 instruction set on the basis of low power techniques in architecture level and logic level. It operates with 5-stage pipeline, and has harvard architecture to increase execution speed. The processor is modeled and simulated in RTL level using VHDL. Behavioral Cache and MMU are added to the VHDL model for instruction level verification of the processor. The core is implemented using Mentor P&R tools with IDEC C-631 Cell library of 0.6㎛ CMOS 1-poly 3-metal CMOS technology.

Ⅰ. 서 론

멀티미디어 휴대 단말 시스템에서 소형, 고성능, 저소비 전력 등은 중요한 과제이며, 멀티미디어 응용들을 위해서는 데이터 입출력, 사용자 인터페이스 외에도 영상, 음성 등의 데이터 압축/신장, 음성 인식/합성, 문자인식 등 이전의 단말 시스템과는 비교할 수 없을 정도의 높은 성능이 필요하다. 특히 무선통신망에서 영상 통신 서비스를 구현하기 위해서는 고집적화와 저전력화 기술 개발이 필수적이다.

특별한 응용을 위한 프로세서를 개발하는데 RISC 코어를 CPU로 사용하고 다른 기능 블록들을 코프로세서화하여 단일 칩에 집적하는 경향이 늘고있는데, 휴대형 단말기의 경우 제한된 전력 공급원인 배터리로서 구동되기 때문에 휴대형 단말기에 사용되는 프로세서는 속도와 성능을 유지하면서 전력을 최소화하도록 재조정되어야 한다.

현재 상용화되고 있는 범용 마이크로프로세서의 경우 속도와 성능향상을 위해 한 싸이클에 보다 많은 명령어를 실행하기 위한 수퍼 스칼라, 수퍼 파이프라인, VLIW(Very Long Instruction Word) 기술들이 사용되고 있다. 그러나 이러한 기술은 일반적으로 제어 로직을 복잡하게 하고, 메모리 인터페이스가 증가되는 등 전력소모를 증가시키게 된다[1].

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.

Ⅱ. 저전력 Methodology

RISC 코어의 저전력화를 위해 구조레벨에서는 Thumb코드 지원을, 로직 레벨에서는 precomputation 방법에 기초를 둔 전력관리를 사용하였다.

1. 코드 밀도(Code Density)

전력 효율적인 프로세서 구조적인 측면에서 가장 고려해야 할 사항은 명령어 셋으로, 명령어 셋은 높은 코드 밀도(code density)를 가지고 있어야 한다. 즉 명령어 페치는 프로세서 전력의 약 1/3을 차지하므로 증가된 코드 밀도는 보다 효율적인 프로세서 메모리 사용을 통한 높은 프로세서 성능과 낮은 명령어 캐시 미스 율을 가져온다[2].

ARM 프로세서는 다른 RISC 프로세서보다 높은 코드밀도를 가지는 것을 볼 수 있는데, 범용 RISC 프로세서 경우 명령어들이 고정길이 명령어 포맷으로 인해 사용되지 않은 필드를 지니며, 거의 인코딩 되지 않기

본 논문은 전남대학교 학술연구비 지원에 의하여 연구되었음

때문에 범용 CISC 보다 약 30%정도 코드밀도가 낮다. 그러나 ARM 프로세서 명령어들은 모든 명령어의 조건적인 실행, ALU 연산과 쉬프터 연산 작용을 할수 있는 ADD R2, R4, R5 LSR R4 명령어, 레지스터 write-back 옵션 예에서 나타나듯이, RISC 구조이면서 CISC 구조와 같이 각 명령어 속에 많은 내용을 인코딩하기 때문에 CISC 프로세서만큼 높은 코드 밀도를 지닌다.

ARM Thumb 코드의 경우 컴파일된 프로그램의 전체 크기를 감소하기 위해 정의된 16비트 코드로서 원래의 ARM 32 비트 명령어를 압축한 코드이다[3]. Thumb 코드는 ARM 명령어와 일대일 대응되며 ARM 명령어에 비해 30% 정도 코드 밀도가 높다. 명령어 메모리에서 페치된 16비트 명령어는 코어 내부에서 명령어 실행을 위해 32비트 명령어로 디코딩된다. 그림 1은 Thumb 코드의 ARM 명령어로 디코딩되는 예를 보여준다.

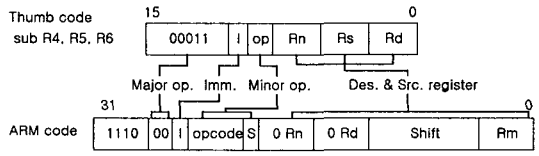

그림 1. Thumb 코드의 ARM 명령어로의 디코딩

2. 전력관리(Power Management)

일반적으로 전력은 정적(static) 전력과 동적(dynamic) 전력에 의해 소모된다. 정적 전력은 칩 전원이 들어올 때마다 존재하는 주파수와 독립적인 전력 요소이며, 동적 전력은 만약 노드값이 변하지 않는다면 전력이 소모되지 않는 주파수 의존적인 전력 요소로서 현존하는 대부분의 전력 최소화 기술들은 회로의 스위칭 커패시턴스를 최소화함으로써 동적 전력을 감소하는 중점을 두고 있다. 동적 전력을 감소하기 위한 방법으로는 시스템 레벨 전력 관리(System Level Power Management)와 동적 전력 관리(Dynamic Power Management)로 구분된다.

시스템 레벨 관리는 시스템 내에서 일정시간 동안 구동되지 않은 장치에 전원 공급 및 클럭을 중지시키는 방법으로 칩과 모듈 레벨에서 널리 사용되어왔다. 이에 반해 동적 전력 관리는 몇몇의 입력 조건이 만족되었을 때 레지스터 load-enable 신호를 사용하거나, 클럭을 gating함으로써 입력/상태 레지스터들을 disable하는 방법으로 순차회로들에서 스위칭을 감소시키는데 매우 효율적인 관리 방법이다. 이는 매 클럭싸이클마다 하드웨어의 shutdown이 결정된다는 점으로 시스템 레벨 전력 관리와 구별된다. 그러나 단점은 입력/상태 레지스터의 입력조건들을 식별할 수 있는 부가적인 회로가 추가된다는 점이다. 이러한 동적 전력 관리 기법은 gated 클럭 FSM 접근[4], n개의 중첩되지 않는 다중 클럭 사용[5], FSM의 decomposition [6], precomputation 기반 전력 감소[7] 등의 여러 가지 관리기법이 제안 및 사용되어왔다.

설계된 RISC 코어의 전력관리는 동적 전력관리 중 precomputation 방법에 기초를 두었으며 특히 전력 소모가 많은 파이프라인 레지스터의 스위칭 커패시턴스 감소에 중점을 두었다.

Ⅲ. RISC 코어 구조

본 논문에서 설계된 RISC 코어는 위의 저전력 Methodology들과 휴대형 단말기에 주로 사용되는 Strong ARM 코어 구조에 기초를 두었으며, 5단 파이프라인으로 동작하고 하바드 구조를 가진다.

1. 명령어 셋

명령어 셋은 현재 유용한 RISC 프로세서들의 명령어 셋 중 범용 16비트 CISC 만큼 높은 코드 밀도(Code Density)를 가지는 ARM®V4 명령어 셋을 따른다[8]. 특히 ARM 코드의 압축된 형태의 16비트 Thumb 코드를 지원함으로써 전체 시스템에서 가장 전력소모가 많은 메모리 접근 시 소모되는 전력을 감소시킬 수 있도록 하였다.

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 flow와 특권 모드로 들어갈 때 사용되어진다.

명령어들의 특징은 모두 조건적인 실행을 수행함으로써 일반 명령어와 분기 명령어를 하나의 명령어로 수행할 수 있다. 모든 단일 load와 store 명령어들은 옵션으로써 수정된 값을 자신의 base 레지스터들로 다시 쓸 수 있어 스택과 큐가 소프트웨어적으로 구현이 편리하며, 블록 데이터 명령어는 16개의 load/strore 명령어를 대치할 수 있는 명령어로서 context를 복구하거나 메모리로부터 데이터를 블록으로 전송할 때 유용한 명령어이다.

2. 동작 모드

프로세서 동작은 정상적인 프로그램 상태인 User 모드, 시스템 모드들인 IRQ(Interrupt Request)모드, FIQ(Fast Interrupt Request), Supervisor 모드, Abort 모드, Undefined 모드 등 6개의 동작모드를 가진다. 모

드 전환은 레지스터 비트들에 의한 제어, 소프트웨어 제어 혹은 외부적인 인터럽트들, 예외적인 처리에 의해 수행된다.

3. 데이터 패스

설계된 32비트 RISC 코어는 5단 파이프라인(Fetch, Decode, Execute, Buffer, Writeback)으로 구성되어지며 전체 블록도는 그림 2와 같다.

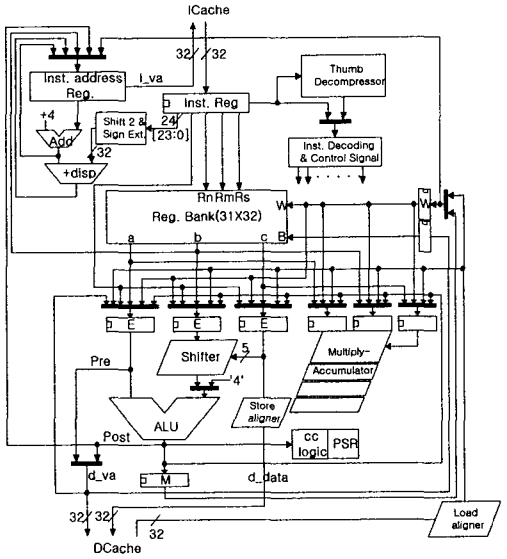

그림 2. 설계된 32비트 RISC 코어의 전체 블록도

데이터패스에서 높은 코드 밀도를 위해 Decode단의 명령어 디코딩 전에 Thumb decompressor를 두었으며, 이는 16 비트 thumb 코드를 대응하는 32 비트 ARM 코드로 변환하는 기능을 수행한다. Thumb 코드 처리 시에서는 그림 3과 같이 Decode 단에서 2 싸이클이 걸리도록 하였다.

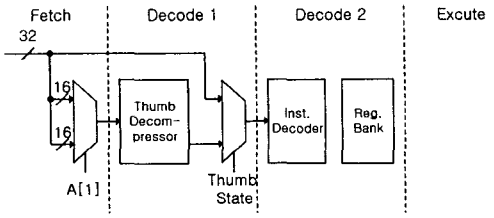

그림 3. Decode 단에서의 Thumb decompressor

Decode단에 32 비트 빠른 분기 덧셈기를 추가하였는데, 이는 코어 실행 시간동안 계속 활성화되어 전력 소모를 가져오나 분기 taken시 따로 분기 예측 하드웨어를 필요로 하지 않고 1싸이클 penalty 성능 향상을 가져온다. 그림 4는 분기 명령어의 실행과정을 나타내는데 PC가 104인 지점에서 분기명령어 BE 명령어가 이전 명령어인 ADD 명령어에 의해 발생된 조건코드들에 의존하는 경우를 나타낸다. ADD 명령어의 Execute 단으로부터의 조건 코드들에 따라 실행 유닛에서는 다음 fetch 싸이클 동안 branch target PC를 지시하게된다.

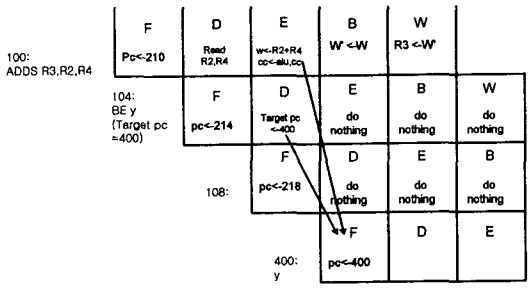

그림 4. 분기 명령어의 파이프라인 블럭도

그래픽 처리에 적합한 명령어 지원을 위한 배럴 쉬프터는 MUX 기반으로 한 양방향 배럴 쉬프트로 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 쉬프트양을 직접 제어 신호로 사용함으로써 어레이 배럴 쉬프터에서 사용되는 5:32 복호기를 사용하지 않는다.

영상의 압축/신장 등의 처리 등의 효율적인 DSP 처리를 위한 MAC(Multiplication and Accumulation)은 별도의 Modified Booth 곱셈기로 설계되었는데, 최소 실행시간을 위한 early termination detection 회로를 두어 곱셈의 조기 종료가 가능하여 전력을 감소하도록 하였다.

레지스터 뱅크는 31개의 범용 레지스터들로 구성되어있으며 각 모드마다 16개의 32 비트 레지스터가 할당되며, 6개의 상태 레지스터들과 함께 인터럽트 처리를 빨리 수행할 수 있다. FIQ경우는 7개의 레지스터를 따로 두어 레지스터 저장 복구 과정을 거의 생략 할 수 있다. 전력 감소를 위해 사용되지 않은 모드의 레지스터의 클럭을 disable함으로써 현재 수행중인 해당 모드 레지스터들만 활성화되게 하였다.

4. 제어 유니트

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 정적 제어 방식을 사용하였으며, 또한 파이프라인 전역 제어기를 두어 다중 싸이클 명령어나 파이프라인 전체의 진행에 대한 제어를 하였다.

데이터 정적제어 방식의 경우 명령어의 각 단에서의 제어 신호들이 Decode 단 때 모두 발생되기 때문에 명령어를 실행함에 있어 각 단 이전에서 사용되지 않는 블록들을 알 수 있다. 이는 명령어 실행 시 데이터 패스 부의 사용되지 않는 파이프라인 레지스터들의 클럭을 disable시킴으로써 전력소모를 감소할 수 있는데, Execute, Buffer, Writeback 각 단으로 데이터가 넘어가지 전에 이전 단에서 각 단에 사용될 제어신호들을 조합하여 gated 클럭을 사용함으로써 선택적으로 clock을 disable시킬 수 있다. 그림 5는 곱셈 명령어의 예를 나타낸다.

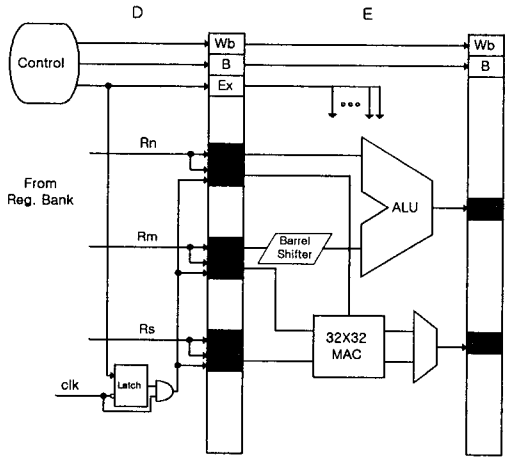

그림 5. 선택적인 Gated 파이프라인 레지스터

그림 5에서 곱셈기는 Execute 단에서 연산이 수행되므로 Decode단에서 Execute 단에서 사용될 제어 신호들을 이용하여 제어 신호를 만든후 falling edge 래치와 AND 게이트를 이용하여 곱셈 명령어가 수행될 때 만 곱셈기의 연산자의 파이프라인 레지스터가 사용되어지도록 하였다. 나머지 파이프라인 레지스터들의 클럭도 같은 방식으로 제어하여 조금의 부가적인 회로를 추가함으로써 명령어 수행시 필요한 파이프라인 레지스터들의 클럭만을 활성화하도록 하였다.

Ⅳ. 시뮬레이션 및 구현

본 논문에서는 32비트 RISC 코어를 VHDL을 사용하여 합성 가능한 RTL 수준에서 모델링하였다. 검증을 위해 명령어, 데이터 캐쉬 및 MMU의 행위수준 모델을 추가한 환경에서 먼저 프로세서 각 명령어 별로 시뮬레이션을 수행하였다. 각 명령어 별 검증 후 프로그램 수준의 검증을 위해서 C언어로 기술된 응용 프로그램들에 대한 어셈블리어를 추출하여 시뮬레이션을 수행하였다. 검증이 완료된 RISC 코어는 0.6㎛ CMOS 1-poly 3-metal 공정 기반의 IDEC C-631 셀라이브러리를 사용하여 논리 합성 및 검증되었으며, 합성된 회로는 Mentor 툴에서 Auto P&R되어 현재 circuit 레벨의 전력 분석 및 검증 중에 있다.

Ⅴ. 결 론

PDA, 노트북 등의 멀티미디 휴대 단말기에서 사용될 수 있는 32 비트 RISC 코어를 구현하였다. RISC 코어는 5단 파이프라인으로 동작하고, 구조레벨의 Thumb 코드 지원 등의 코드 밀도 고려와 로직 레벨에서 precomputation 방법 등의 동적 전력관리기법에 기초를 두어 설계되었다. 행위 수준의 명령어, 데이터 캐시, MMU 등을 추가한 환경에서 명령어 수준 및 C 언어를 이용한 프로그램 수준에서 검증되었으며, 검증된 모델은 0.6㎛ CMOS 1-poly 3-metal 공정 기반의 IDEC C-631 셀라이브러리를 사용하여 Mentor 툴에서 Auto P&R되었다. 설계된 RISC 코어는 stand-alone 코어, ASSP(Application Specific Standard Parts)의 매크로 셀로서 사용될 수 있다.

참고 문헌

[1] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Processors", Proc. of the IEEE Symposium on Low Power Electronics, pp. 12-13, Oct. 1995.

[2] J. Bunda, et al.,"16-Bit vs. 32-Bit Instructions for Pipelined Microprocessors", Proc. Int'l Symp. Computer Architecture, IEEE CS Press, pp. 237-246, 1992.

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

[4] 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.

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

[6] S. H. Chow, et al., "Low Power Realization of Finite State Machines-A Decomposition Approach ", ACM Trans. on Design Automation of Electronic Systems, Vol. 1, No. 3, pp. 315-340, July 1996.

[7] M. Alidina, et al.,"Precomputing-Based Sequential Logic Optimization for Low-Power", IEEE Trans. on VLSI, Vol.2, No. 4, pp. 426-436, Dec. 1994.

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