32 비트 RISC/DSP CPU 를 위한 고속 3 포트 레지스터 파일의 설계

고 재 명, 유 동 렬, 이 용 환, 안 상 준, 이 용 석
연세대학교 전자공학과
서울시 서대문구 신촌동 134 번지
Tel 02)361-2872, Fax 02)312-4584
jemings@dubiki.yonsei.ac.kr

High Speed Triple-port Register File for 32-bit RISC/DSP Processors

Jaemyoung Koh, Dongryul Ryu, Yonghwan Lee, Sangjun An, Yongsurk Lee
Dept. of Elec. Eng., Yonsei University
134, Shinchon-dong, Seodaemun-gu, Seoul
Tel 02)361-2872, Fax 02)312-4584
jemings@dubiki.yonsei.ac.kr

Abstract

This paper describes a 72-word by 32-bit 2-read/1-write multi-port register file, which is suitable for 32-bit RISC/DSP microprocessors. To minimize area and achieve high speed, advanced single-ended sense amplifiers are used. Each part of circuit is optimized at transistor level. The verification of functionality and timing is performed using HSPICE simulations. After modeling and validating the circuit at transistor level, it was laid out in a 0.6um 1-poly 3-metal layer CMOS technology. The simulation results show maximum operating frequency is 179MHz in worst case conditions. It contains 27,326 transistors and the size is 3.02mm by 2.20mm.

1. 서 론

내장형으로 사용되는 CPU는 대부분이 RISC (Reduced Instruction Set Computer) 구조를 채택하고 있다. RISC는 대부분의 명령어를 단일 사이클 내에 수행할 수 있는 특징이 있으며 이러한 동작은 수행되는 명령어의 오퍼랜드를 외부 메모리에서 참조하는 경우에는 불가능하게 된다. 그러므로, 모든 명령어의 오퍼랜드를 CPU 내의 레지스터에서 처리함으로써, 단일 사이클 내에 수행이 가능하도록 한다. 레지스터는 다중 포트 구조를 가지는 SRAM 형태이며 CPU 내부에 위치하여 한 사이클에 하나 이상의 읽기/쓰기 억세스를 허용하는 임시 기억 장치이다. RISC는 CPU 내에 많은 레지스터를 파일 형태로 구성하며, 이 레지스터 파일의 사용 빈도를 극대화함으로써 프로세서의 성능 향상을 얻는다.

본 논문에서는 멀티미디어 기능을 강화한 32 비트 RISC/DSP 마이크로프로세서를 위한 72x32 2-read/1-write 포트 레지스터 파일을 설계하였다. 기존의 회로보다 작은 면적을 차지하면서도 고속을 구현하는데 중점을 두었다. 본 논문의 제 2 장에서는 레지스터 파일의 구조를 설명하였으며, 제 3 장에서는 레지스터 파일의 각 부분에 대한 설계를 기술하였다. 제 4 장에서는 회로의 검증을 위한 시뮬레이션 과정과 레이아웃을 다루었고, 제 5 장에서 최종 결과를 분석하였다.

2. 레지스터 파일의 구조

32 비트 RISC/DSP 프로세서의 레지스터 파일은 64 개의 지역 레지스터(local register)와 32 개의 전역 레지스터(global register)로 구성되어 있다. 지역 레지스터는 응용 프로그램을 위해 사용되는 레지스터이며 전역 레지스터는 일반 전역 레지스터와 특수 레지스터로 구성되어 있다. 본 논문에서 설계한 영역은 지역 레지스터와 일반 전역 레지스터를 대상으로 한다. 포트는 2 개의 읽기 포트(read port)와 1 개의 쓰기 포트(write port)를 가지도록 하여 단일 사이클에 2 개의 읽기와 1 개의 쓰기가 동시에 가능하도록 한다. 레지스터 파일 억세스의 임계 경로(critical path)는 read 억세스 시간이고 이것은 주소 신호가 디코더를 거치면서 해당 레지스터 셀의 워드를 구동하고 레지스터 셀에 저장된 내용이 비트 라인을 따라서 외부로 출력되기까지 걸리는 시간이다. 일반적인 레지스터 셀만으로는 외부 출력까지 구동

※ 본 연구의 일부는 과학 재단의 특정 기초 연구 과제 연구비 지원으로 이루어 졌음. (97-0100-0701-2)

할 수 없고, 레벨이 천이(transition)하는데 시간이 오래 걸리기 때문에 센스 증폭기를 사용하여 보다 빠른 억세스가 가능하도록 한다. 레지스터 파일의 구조는 크게 read 디코더, write 디코더, 셀 어레이(cell array), 센스 증폭기(sense amplifier)로 나누어진다. 그림 1 은 레지스터 파일의 블록 다이어그램을 나타낸 것이다.

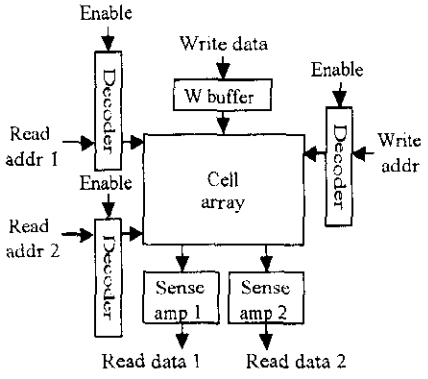

그림 1. 레지스터 파일의 블록 다이어그램

본 논문의 프로세서는 Fetch-Decode-Execute-Memory-Write Back 의 5 단 파이프라인을 사용하기 때문에 D(decode)단에서 레지스터의 값을 읽어 들이고 WB(Write Back)단에서 레지스터에 값을 저장한다. 따라서 파이프라인이 진행하면서 한 사이클에 같은 장소의 레지스터를 동시에 읽기/쓰기 억세스를 할 경우가 발생할 수 있는데, 이때의 억세스 순서는 반드시 쓰기가 먼저 수행된 다음에 읽기를 수행하여야 한다. 다시 말해서 읽는 데이터는 이전의 값이 아닌 반드시 새로운 값이어야 한다는 것인데 이것이 지켜지지 않을 경우 RAW(read after write) 데이터 해저드(data hazard)가 발생한다.[1]

이를 해결하기 위하여 read 페이즈와 write 페이즈를 구별해서 동작을 시킬 경우 레지스터 파일의 속도 저하를 피할 수 없기 때문에 레지스터 회로상 유닛에서 데이터 포워딩(data forwarding)을 지원하여 레지스터 파일에서는 이를 의식하지 않게 함으로써 보다 고속의 동작을 수행할 수 있게 한다.[5]

3. 레지스터 파일의 설계

3.1 레지스터 셀 (Register cell)

본 논문의 레지스터 파일은 설계 면적을 축소하기 위하여 single-ended 포트를 사용하였다. Single-ended 포트는 설계 면적을 20%정도 줄일 수 있기 때문에 포트의 수가 비교적 많고 레지스터 파일의 개수가 많은 경우에는 fully-differential 포트보다 single-ended 포트가 유리하다. 하지만 센스 증폭기의 설계에서 작은 레벨의 천이하는 것만으로 값을 결정해야 하기 때문에 fully-differential 포트보다 억세스 속도가 떨어지게 된다.[2] 따라서 동작속도를 높이기 위해 본 논문에서는 읽기 포트의 게이트를 구동하는 방식을 사용한다.[3] 이 방식은 레치의 인버터는 nMOS 의 게이트만을 구동하기 때문에 레치의 크기를 줄일 수 있고 입력 포트의 nMOS 크기에 영향을 받지 않게 된다. 또한 안정적인 쓰기 동작을 위하여 피드백 인버터의 게이트 길이(gate length)를 크게 하여 weak 인버터로 동작하게 한다. 그림 2 는 레지스터 셀의 회로도를 나타낸 그림이다.

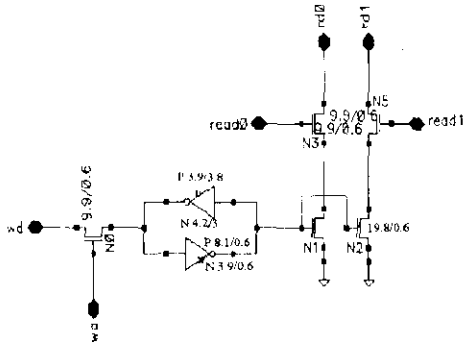

그림 2. 레지스터 셀의 회로도

3.2 디코더 (Decoder)

Read/write 주소에 해당하는 워드 라인(word line)을 선택하여 구동하는 부분으로서 전체 경로에서 가장 시간이 많이 걸리는 부분이기 때문에 속도를 빠르게 하기 위해서 프리디코더(predecoder)와 메인 디코더(main decoder) 2 단계로 나누어 디코드한다.[2]

레지스터 주소는 A7~A0 로 나타내지만 A6 는 쓰이지 않기 때문에 실제 사용되는 비트 수는 7 비트이고 실제로 디코드되는 주소는 일반 전역 레지스터 영역인 2~15 와 지역 레지스터 영역인 128~191 이다. 7 비트 중 상위 5 비트 A7-A2 는 row 디코더로 들어가고 하위 2 비트 A1, A0 는 column 디코더로 들어간다. row 디코더에 들어가는 A7-A2 는 다시 나뉘어져 그 중 상위 3 비트 A7, A5, A4 는 프리디코더로 들어가고 하위 2 비트 A3, A2 는 메인 디코더의 입력으로 들어간다. 그림 3 은 디코더의 회로도를 보인 것이다

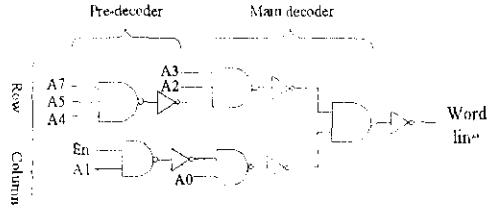

그림 3. 디코더의 회로도

3.3 센스 증폭기 (Sense amplifier)

레지스터 파일에 읽기 주소가 들어와서 셀의 데이

터가 비트 라인(bit-line)의 값을 변화시키는 것은 매우 천천히 동작한다. 따라서 고속의 동작을 위해서는 비트 라인 값의 변화를 감지하여 빠르게 구동시키는 회로가 필요하다. 센스 증폭기는 이와 같이 셀 어레이(cell array)에서 비트 라인으로 전달된 read 데이터의 작은 변화를 확실하게 감지하고 증폭하여 빠른 천이(transition)를 하도록 해서 full-swing으로 외부에 연결시키는 역할을 한다. 그림 4는 센스 증폭기의 회로도이다.

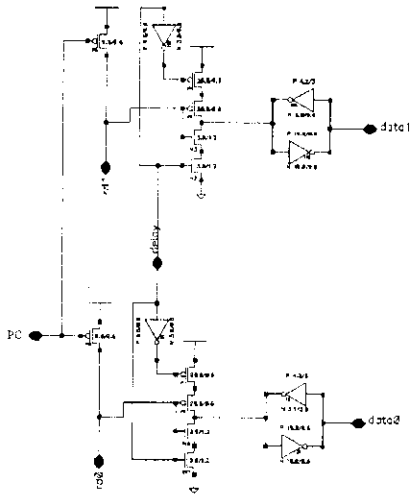

그림 4. 센스 증폭기의 회로도

이것은 pseudo-nMOS 회로를 이용한 센스 증폭기로서 이 방식을 사용하여 구현하면 동작 속도는 향상되지만 전력 소모가 커진다는 단점이 있다. 따라서 평상시에는 전류를 소모하지 않다가 원하는 때에만 전류 경로를 형성하는 동적(dynamic) 회로를 사용함으로써 소모전력을 감소시킨다.[4] 그림 4에서 pMOS의 게이트로 들어가는 신호 PC는 프리차지(precharge) 신호를 나타내고 PC가 0(low)일 때 VDD에 연결된 pMOS 트랜지스터가 turn on 되어 비트 라인(bit line)이 프리차지 된다. PC가 1(high)가 되면 pMOS는 turn off 되고 레지스터 셀에 저장되어 있는 값에 따라 0으로 천이를 일으키거나 1의 상태를 유지하여 값을 전달한다.

3.4 지연 회로(Delayed clock)

지연 회로는 센스 증폭기의 pseudo-nMOS를 동적으로 동작시켜 전력 소모를 줄임과 동시에 프리차지된 비트 라인이 원하지 않는 천이를 함으로서 발생할 수 있는 글리치(glitch)를 제거하는 역할도 한다. 지연 회로의 출력은 tri-state 인버터에 연결되어 0일 때에는 비트 라인(bit line)의 값이 외부 출력으로 연결되지 않아 high-z 상태로 만들고 지연 회로의 출력이 1이 됐을 때 비로소 비트 라인의 값이 tri-state 인버터를 거쳐 외부로 전달된다. 이렇게 함으로써 디코더의 출력이 스위치 트랜지스터를 구동하기까지의 불안정한 동작이 출력으로 전달되는 것을 막아준다. 지연 시간은 SPICE 파라미터를 이용하여 시뮬레이션을 한 다음 10%의 마진을 두고 결정하였다. 그림 5는 지연 회로의 회로도이고 그림 6은 지연 파형을 나타낸 것이다. 클럭이 1로 올라갈 때 최악 조건에서 1.49ns 이후에 1로 천이하고 클럭이 0으로 떨어질 때에는 거의 동시에 0으로 천이함을 보여준다.

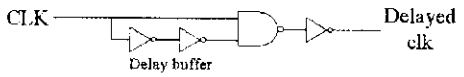

그림 5. Delayed clock의 회로도

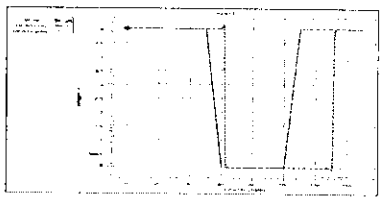

그림 6. Delayed clock의 시뮬레이션 파형

그림 7은 레지스터 파일의 SPICE 시뮬레이션을 하기 위한 모델을 나타낸 것이다. 임계 경로(critical path)인 read 경로를 모델링한 것이며, 기생 커패시턴스 성분을 시뮬레이션에 반영하기 위하여 연결되는 수만큼의 트랜지스터를 모두 계산하여 모델링 하였다.

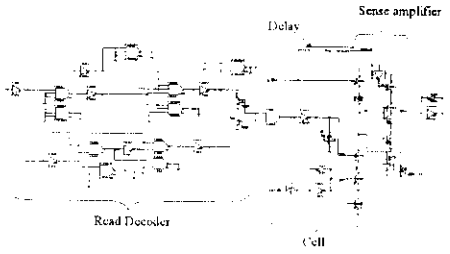

그림 7. SPICE 시뮬레이션 모델

4. 검증 및 구현

앞에서 보인 회로도는 모두 HSPICE를 이용한 시뮬레이션으로 동작과 타이밍을 검증하였다. SPICE 시뮬레이션은 다음과 같은 과정을 거쳤다. Schematic 툴을 사용하여 회로도의 넷리스트(netlist)를 추출한 다음, 여기에 테스트 입력을 집어넣어서 결과 파형을 확인하였다. 테스트 입력의 종류는 PLL(Phase Locked Loop)에서 공급된 시스템 클럭과 레지스터 제어 유닛에서 넘겨받는 신호인 X, Y 레지스터의 주소와 데이터를 쓸 레지스터의 주소, 그리고 레지스터에 저장하려는 데이터이다. 그림 8은 SPICE 시뮬레이션 파형을 나타낸 것이다. 디코더를 통과하는데 걸리는 시간은 최악 조건

(worst condition)에서 1.10ns 가 걸리고, 최상 조건(best condition)에서 0.87ns 가 걸렸다. 최악 조건은 83℃ 4.5V 전원전압 공급의 경우로 설정하였고, 최상 조건은 20℃ 5.5V 전원전압 공급, 그리고 정상 조건(normal condition)은 20℃ 5V 전원전압 공급으로 하였다. HSPICE 파라미터를 이용하여 기생 커패시턴스 성분을 고려하였을 때, 디코더를 통과하는데 걸리는 시간은 최악 조건에서 1.23ns 가 걸리고, 정상 조건에서 1.04ns 가 걸렸다. 그리고 비트 라인을 거쳐서 센스 증폭기를 통과하는 시간은 최악 조건에서 2.6ns, 정상 조건에서 2.1ns 가 걸렸다. 그리고 출력 버퍼에 데이터가 도달하는 시간은 최악 조건에서 2.8ns 가 걸리고, 정상 조건에서 2.3ns 가 걸렸다. 이것은 최악 조건에서 179MHz 에서 동작함을 보여주며 목적 주파수(target frequency)인 100MHz 를 만족한다. 표 1 은 레지스터 파일의 각 부분에서 걸리는 시간을 나타낸 것이다.

표 1. Access time (normal condition 20℃, 5V)

	Time(ns)	Percent(%)
Decoder	1ns	43.5%
Cell(bit-line delay)	0.1ns	4%
Sense amp.	1ns	43.5%
Output buffer	0.2ns	9%

센스 증폭기에서 걸리는 시간이 비교적 큰 이유는 회로의 안정화를 위하여 지연 회로의 지연 시간을 충분히 주었기 때문이다. 지연 시간을 감소시키면 그만큼 속도는 향상되지만 아주 높은 주파수의 글리치가 발생하게 된다.

SPICE 시뮬레이션을 통하여 검증된 회로는 0.6um CMOS 공정을 이용하여 완전 주문형(full-custom)방식으로 설계되었다. 모든 회로는 트랜지스터 수준에서 설계되었으며, 레이아웃을 마친 뒤에 DRC(Design rule check)와 LVS(Layout versus Schematic) 검사를 하여 레이아웃이 제대로 설계되었음을 검증하였다.

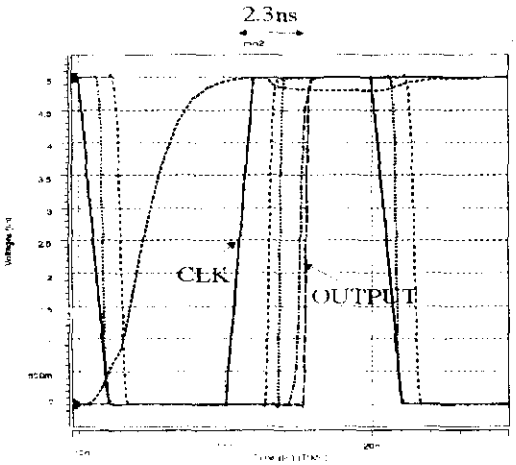

그림 8. 시뮬레이션 파형

5. 결 론

본 논문에서는 32 비트 RISC/DSP CPU 에 적합한 레지스터 파일을 설계하였다. SPICE 시뮬레이션의 결과는 회로가 정상적으로 동작함을 보여주고 있으며, 목표 동작 주파수인 100MHz 를 만족하였다. 이렇게 검증된 회로는 0.6um 1-poly 3-metal CMOS 공정을 이용한 완전 주문형 (full-custom) 방식으로 설계되었다. 이렇게 설계된 레이아웃은 다시 DRC 와 LVS 검사를 통하여 최종 검증되었다. 그림 9 는 레지스터 파일의 레이아웃이다. 최대 동작 가능 주파수는 정상 조건에서 217MHz 이며, 최악 조건에서도 179MHz 에서 동작함을 확인하였다. 코어(core)의 면적은 3.02mm x 2.20mm 이고 총 트랜지스터의 수는 27,326 개이다.

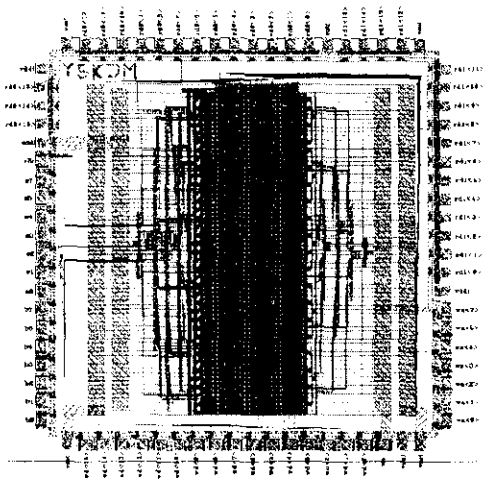

그림 9. 레지스터 파일의 레이아웃

6. 참 고 문 헌

[1] J.L.Hennessy & D.A.Patterson, "Computer Architecture, A Quantitative Approach", 2nd edition, *Morgan Kaufmann Publishers*, 1996

[2] N.Weste & K.Eshraghian, "Principles of CMOS VLSI Design", 2nd edition, *Addison-Wesley Publishing Co.*, 1993

[3] C.C.Chao & B.A.Wooley, "A 1.3ns 32x32 Three-Port BiCMOS Register File," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 6, pp 758-766, June 1996

[4] C.Asato, "A 14-port 3.8ns 116x64 Read-Renaming Register File," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp 1254-1258, November 1995

[5] 이용석, "ALU 와 Register file 의 구조", *비디오 강좌 시리즈*(http://mpu.yonsei.ac.kr), 1998