마이크로프로세서 설계 무작정 따라하기 (1)

KAIST 전자전산학과 박사과정 배영돈(donny@ics.kaist.ac.kr)

본 강좌에서는 초보자를 위한 마이크로프로세서 설계방법을 앞으로 약 4 회 동안 연재하고자한다. 본 강좌는 관련전공의 대학생이 이해할 수 있는 수준으로 최대한 쉬운 설명과 쉬운 예제를 통하여 프로세서의 설계를 처음부터 실제 칩을 만들 수 있는 단계까지 다루고자 한다

.

본 강좌를 완벽하게 이해하기위해선 다음에 대한 기본적인 지식을 필요로 한다.

1) Verilog HDL

RTL(Register Transfer Level) 기술 방법. AND/OR/NOT 게이트와 플립플롭을 기술하고 합성하여 게이트레벨(gate-level)의 Verilog netlist 로 만들 수 있는 수준.

참고자료: Cadence Verilog-XL Reference Manual

2) 컴퓨터 구조(Computer Architecture)

기본적인 용어와 컴퓨터의 동작 개요

참고자료: Hennessy & Patterson, Computer Architecture A Quantative Approach

3) 기초적인 C 언어 사용능력

4) 작업환경

Synopsys 의 DesignCompiler, Cadence 의 Verilog-XL 과 Silicon Ensemble 을 실행할 수 있는 환경

참고자료: 해당 툴의 매뉴얼

1. 서론: ASIC 설계과정

그림 1 은 전형적인 ASIC 의 설계과정(design flow) 이다.

먼저 구체적인 설계에 앞서 칩의 기본적인 구조를 정하고, Verilog 또는 VHDL 을 사용하여 RTL(register transfer level) 기술을 한다. 그리고 Verilog-XL 과 같은 시뮬레이터를 통하여 동작을 검증한다. 그 다음, Synopsys 의 Design Compiler 와 같은 합성(synthesis) 툴을 이용하여 RTL 코드를 게이트 레벨 회로(gate-level netlist)로 합성한다. 합성된 회로는 Cadence SliconEnsemble 과 같은 툴을 이용하여 배치 및 배선(place & route)과정을 통하여 실제 칩 제작을 위한 레이아웃을 생성하게 된다.

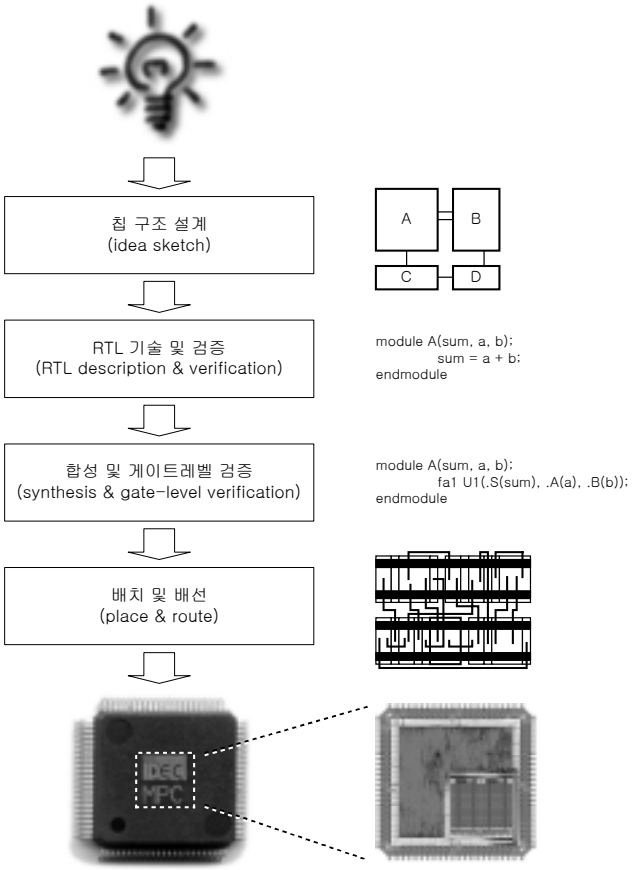

그림 1. ASIC 설계 과정

2. 마이크로프로세서의 개요

마이크로프로세서는 컴퓨터는 물론이며 핸드폰, 프린터, TV 거의 모든 전자제품에 사용되고 있는 핵심부품으로 일반적으로 컴퓨터에 사용되는 중앙처리장치(CPU)로 알려져 있다.
마이크로프로세서는 크게 데스크탑 프로세서와 임베디드(embedded) 프로세서로 구분할 수 있으며, 데스크탑용 프로세서의 경우 Intel 사의 펜티엄과 같이 고성능을 추구하는 대신 크기가 크고 전력을 많이 소비한다. 임베디드 프로세서의 경우 대표적인 응용분야가 휴대용 전화기로 전력소비가 적고, 크기가 작은 특징을 갖고있으며, ARM 사의 제품이 널리 알려져 있다. 본 강좌에서는 16 비트 임베디드 프로세서의 설계를 예제로 ASIC 개발과정을 익히고 마이크로프로세서 설계방법을 설명하고자 한다.

3. 명령어 구조(Instruction Set)

마이크로프로세서의 명령어구조는 프로세서의 성능과 특징을 결정짓는다. 크게 RISC(Reduced Instruction Set Computer)와 CISC(Complex Instruction Set Computer)로 구분할 수 있으며 명령어의 크기(비트 수)가 프로세서의 대략적인 성능을 나타낸다. 최근 임베디드(embedded) 분야에서 널리 사용되는 프로세서는 16 비트 또는 32 비트 RISC 이며, 데스크 탑 분야에서는 64 비트 RISC 가 사용된다. RISC 는 CISC 에 비하여 간단하고 적은 수의 명령어를 사용하며, 파이프라인(pipeline)구조를 사용하여 높은 성능을 갖고 있는 것이 특징이다.

본 강좌에서는 매우 간단한 구조의 16 비트 RISC(이하 SimpleCore)를 설계하고자 한다. 복잡한 구조의 경우에도 기본적인 설계방법은 동일하다.

SimpleCore 의 명령어는 표 1 과 같다.

표 1. SimpleCore 의 명령어

	15	14	13	12	11	10 7	6 4	3 0	Pseudo code
ALU imm.	0	0	Opcode			Rd	Immediate		Rd = Rd op Immediate
ALU reg.	0	1	Opcode			Rd	Rs2	Rs1	Rd = Rs1 op Rs2
Shift/Rotate	1	0	0	Shift		Rd	-	Rs1	Rd = Rd Shift by Amount
Load	1	0	1	0	-	Rd	-	Rb	Rd = Mem[Rb]
Store	1	0	1	1	-	Rd	-	Rb	Mem[Rb] = Rd
Branch	1	1	Cond			Offset			if(cond) PC = PC + Offset
Multiply	1	1	1	1	-	Rd	Rs2	Rs1	Rd = Rs1 * Rs2

Opcode			Mnemonic	Description	Pseudo Code
0	0	0	MOV	Rd = Immediate	Rd = Rs1
0	0	1	ADD	Rd = Rd + Immediate	Rd = Rs1 + Rs2
0	1	0	SUB	Rd = Rd – Immediate	Rd = Rs1 – Rs2
0	1	1	AND	Rd = Rd & Immediate	Rd = Rs1 & Rs2
1	0	0	OR	Rd = Rd | Immediate	Rd = Rs1 | Rs2
1	0	1	CMP	Compare Rd with Immediate	Compare Rs1 with Rs2
1	1	0	MSR	Status Register = Immediate	Status Register = Rs1
1	1	1	MRS	N/A	Rs1 = Status Register

Shift		Mnemonic	Description
0	0	LSL	Shift left
0	1	LSR	Shift right
1	0	ASR	Arithmetic shift right
1	1	ROR	Rotate

Cond		Mnemonic	Description
0	0	EQ	Equal
0	1	NE	Not equal
1	0	AL	Always (unconditional)
1	1	N/A	Unused

SimpleCore 의 명령어는 ALU, Shift/Rotate, Load, Store, Branch 그리고 Multiply 로 간단한 명령어 구조이지만, 상용 임베디드 프로세서의 대부분의 명령어를 포함하고있다.

4. SimpleCore 의 구조(Architecture)

SimpleCore 의 프로세서의 구조는 프로세서의 복잡도와 성능에 큰 영향을 미친다. 따라서 매우 신중하게 결정되어야 한다. 본 강좌에서는 쉽게 설계할 수 있고, 높은 성능을 낼 수 있도록, Harvard Architecture 와 3 단계의 파이프라인(Three-stage Pipeline)구조를 선택하였다. Harvard Architecture 는 명령어 메모리와 데이터 메모리를 동시에 접근할 수 있는 구조이며, 3 단계의 파이프라인 구조는 그림 2 와 같이 명령어 페치(fetch), 명령어 해석(decode), 실행(execute)의 작업을 동시에 수행하는 구조이다.

가장 대표적인 RISC 의 구조인 5 단계 파이프라인을 사용하지 않은 이유는 본 강좌의 목적에 따라 설계와 검증을 쉽게 하기 위한 것이다. 5 단계 파이프라인을 사용할 경우 포워딩(forwarding)이 필요해지므로,

프로세서의 구조가 복잡해지고 이에 따른 검증과정도 복잡해지게 된다. 반면, 3 단계 파이프라인은 5 단계 파이프라인에 비해 한 단계에서 처리할 일이 많으므로 상대적으로 낮은 성능(MIPS: Million Instruction Per Second)을 갖게 된다.

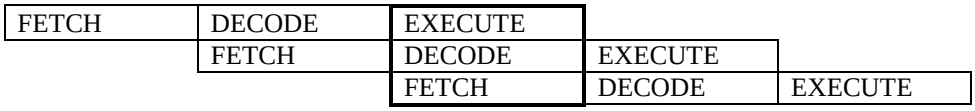

그림 2. 3 단계 파이프라인 구조

SimpleCore 의 레지스터구조는 그림 3 과 같다.
14 개의 범용 레지스터와 한 개의 Program Counter 그리고 Status Register 를 갖고있다.

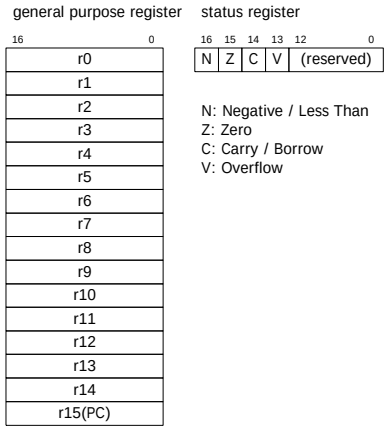

그림 3. 레지스터 구조

SimpleCore 의 데이터패쓰(Datapath) 구조는 그림 4 와 같다.

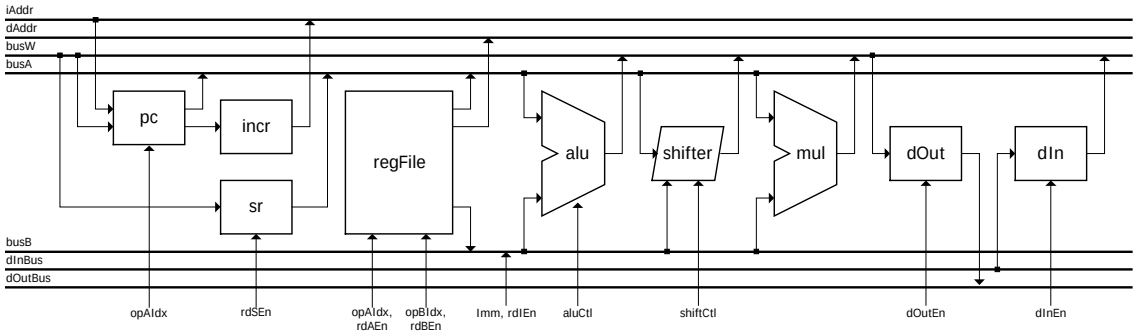

그림 4. SimpleCore 의 데이터패쓰

5. RTL(Register Transfer Level) 기술

1) 데이터패쓰의 설계

마이크로프로세서는 크게 데이터패쓰(datapath)와 컨트롤(control logic)로 구분이 되며, 이 두 가지 부분을

명확히 구분해서 설계하는 것이 중요하다. 그 이유는 프로세서 동작속도는 데이터패쓰를 얼마나 최적화해서 설계했는가에 따라 결정되며, 데이터패쓰를 최적화하기 위해서는 컨트롤에 해당하는 회로를 최대한 제거해야 한다. 데이터패쓰는 말 그대로 데이터가 흘러가는 길을 의미하며, 데이터가 어디로 흘러갈 것인가를 조절하는 부분이 바로 컨트롤이다. 그림 5 는 펜티엄프로세서의 칩 사진으로 정수연산 유닛(integer execution unit)을 살펴보면 데이터패쓰와 컨트롤의 구분이 명확한 것을 알 수 있다. 그리고, 데이터패쓰 부분은 매우 세밀하고 규칙적으로 설계되어있다.

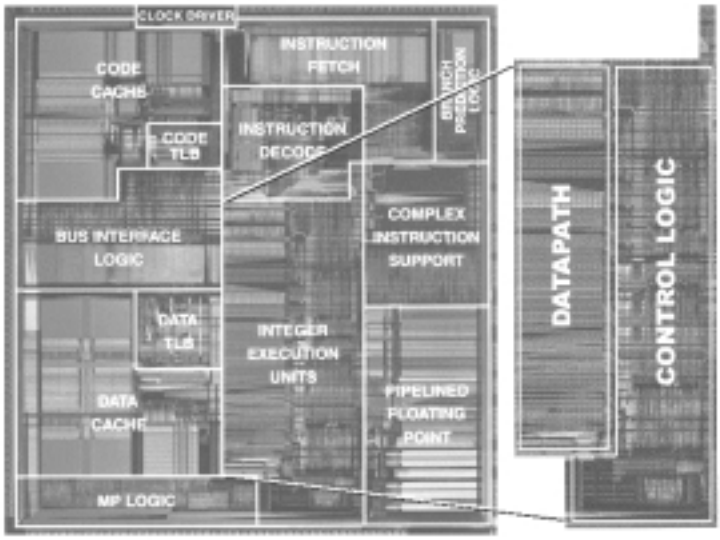

그림 5. 펜티엄프로세서의 데이터패쓰와 컨트롤

데이터패쓰의 설계는 각 기능 블록들이 작은 면적을 사용하고 빠른 속도로 동작하도록 설계하는 것이 중요하다. 상용 마이크로프로세서의 경우, 데이터패쓰의 기능 블록을 데이터패쓰합성기(datapath compiler), 모듈 생성기(module generator), 풀 커스텀 (full-custom layout) 방식을 사용하여 공정에 최적화된 설계가 가능하다. 하지만, 이러한 설계방법은 공정에 매우 의존적인 방식이고 특수한 CAD 툴을 필요로 하기 때문에 사용에 어려움이 있다. 본 강좌에서는 순수 Verilog HDL 만을 이용하여 완전히 합성 가능한(fully synthesizable) 프로세서를 설계하고자 한다.

다음 회에 계속됩니다. 다음 회에서는 데이터패쓰의 Verilog RTL 기술 방법과 컨트롤 회로의 설계 및 시뮬레이션에 대한 내용을 설명합니다.