마이크로프로세서 설계 무작정 따라하기 part-III (1)

부제: 합성, 배치 및 배선

KAIST 전자전산학과 박사과정 배영돈(http://www.donny.co.kr)

지난 9 회의 강좌를 통하여 마이크로 프로세서의 설계방법과 개발환경을 구축하는 방법에 대해서 설명하였다. 따라서, SimpleCore(본 강좌에서 예제로 사용하고 있는 16-bit RISC 프로세서)를 설계하고 이를 이용하여 응용프로그램까지 개발할 수 있게 된 것이다.

그림 1 은 일반적인 반도체의 설계과정(design flow) 이다. 지난 강좌(Part I)에서 우리는 SimpleCore 의 구조설계와 Verilog 를 이용한 RTL(register transfer level) 기술 그리고 Verilog-XL 을 이용한 검증(simulation)을 하였다. 따라서, SimpleCore 를 실제 칩으로 제작하기까지 합성(synthesis) 및 게이트 레벨 검증, 배치 및 배선의 과정을 남겨두고 있다.

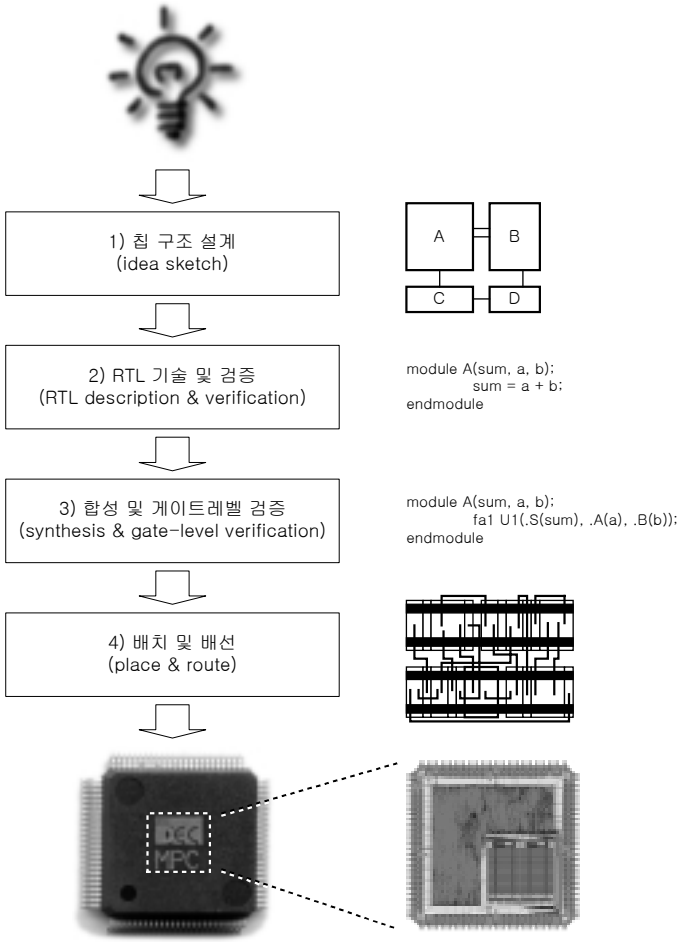


그림 1. 반도체 설계과정

1. 합성 (Synthesis)

합성은 Verilog 로 기술되어 있는 회로를 로직 게이트로 이루어진 netlist 로 만드는 과정을 말한다. 본 강좌에서는 Synopsys 사의 Design Compiler 를 사용하여 합성하는 방법을 설명한다.

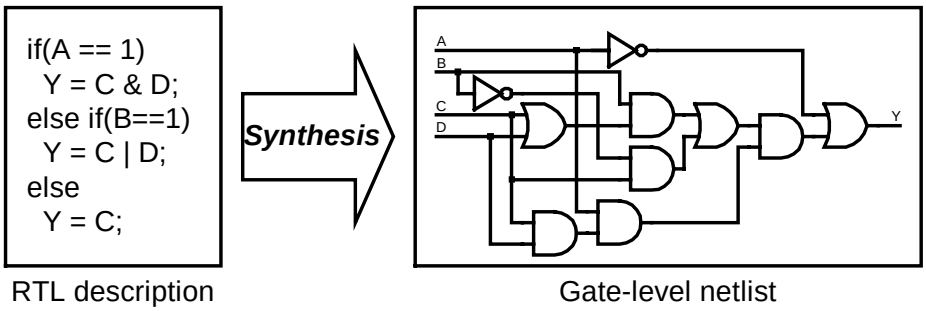


그림 2. 합성

2. 배치 및 배선 (P&R: Placement & Routing)

P&R 과정은 후단계 설계(back-end)라고도 부르며 설계된 회로를 반도체 공정에 사용되는 레이아웃으로 만드는 과정을 말한다 본 강좌에서는 Avanti 사의 Apollo 를 사용하여 P&R 과정을 설명한다.

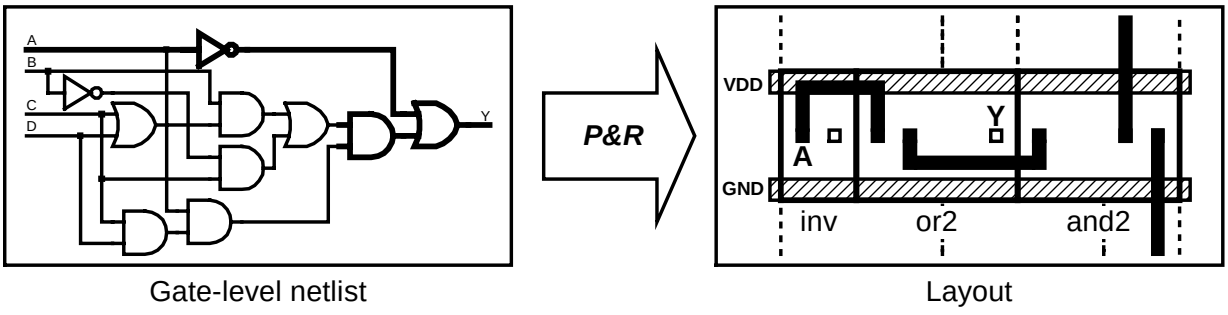


그림 3. 배치 및 배선

3. 셀 라이브러리

RTL 설계와 합성 단계 이후의 과정의 중요한 차이점은 각각 공정에 비의존적, 의존적이라는 것이다. Verilog 를 이용한 설계의 가장 큰 장점이 바로 공정에 비의존적이라는 것이다. 하지만, 이러한 특징은 RTL 수준까지만 적용된다. 합성을 위해서는 칩을 제작하고자 하는 공정의 정보가 필요하며 합성된 회로는 해당 공정에서만 사용할 수 있게 된다. 공정의 정보는 셀 라이브러리라고 불리는 형태를 갖고 있으며, 합성과 P&R 을 하기 위해서는 반드시 필요하다. 셀 라이브러리가 포함하고 있는 정보는 그림 2 에서와 같이 합성에 사용될 셀(로직 게이트)에 대한 특성(속도, 크기, 등), 게이트 레벨 검증에 필요한 동작 특성(Verilog 모델), 그림 3 과 같이 P&R 에 사용될 각각의 셀의 레이아웃이다. 이러한 정보는 해당 공정을 개발한 회사의 기밀 자료이므로 실제 칩을 제작할 설계자들에게만 제공되며, 외부에 유출하지 않겠다는 서약서 (NDA: Non Disclosure Agreement)를 작성해야 사용이 가능하다. MPW 를 이용해 칩을 제작하는 경우에는 IDEC 을 통해서 이러한 라이브러리를 사용할 수 있다. 본 강좌에서는 Hynix 0.35μm MPW 공정의 라이브러리를 사용하여 설명할 것이다. 따라서, 라이브러리가 필요한 독자는 IDEC 에 요청하여 NDA 작

성 후 사용해야 한다.

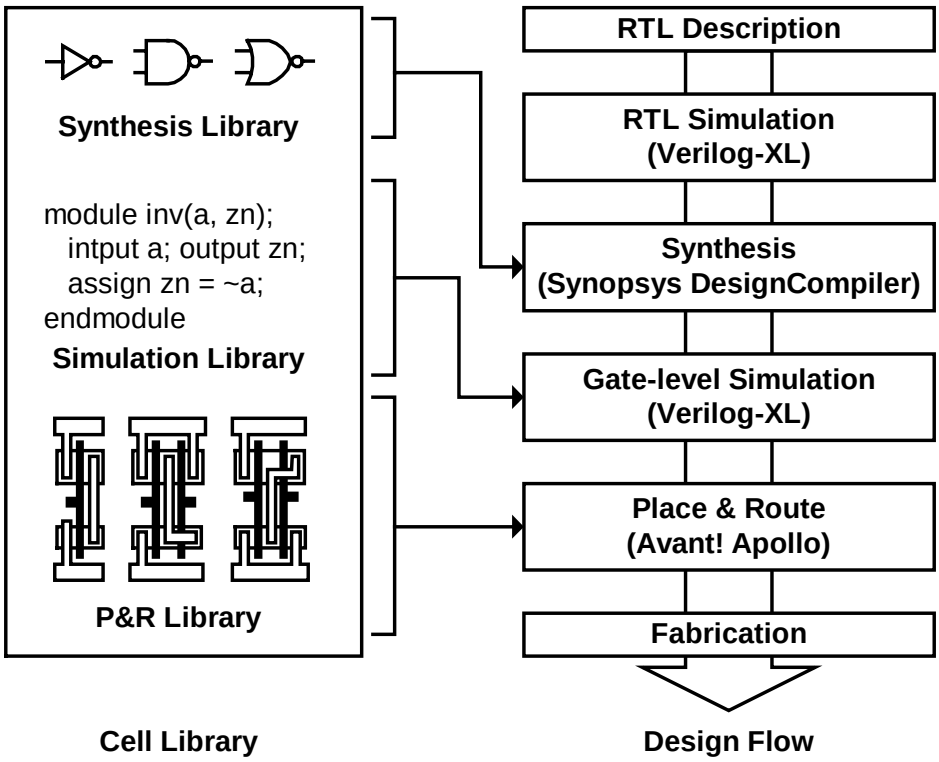


그림 4. 셀 라이브러리

4. 합성 가능한(synthesizable) 설계

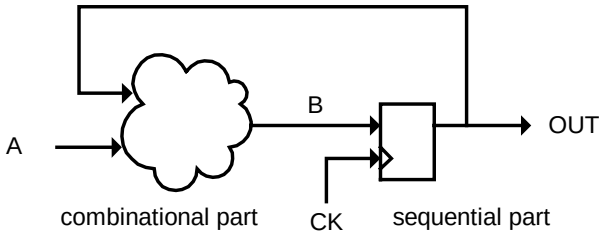
합성방법을 설명하기 전에 먼저 합성 가능한 설계와 합성 불가능한 설계에 대해서 알아보도록 하자. Verilog 는 하드웨어기술언어(HDL: Hardware Description Language)이지만 Verilog 로 기술한 모든 설계가 실제 회로로 구현이 가능한 것은 아니다. 그 이유는 다음과 같이 정리할 수 있다.

a) Verilog 언어에서 제공하는 기능 중 회로(또는 시스템)의 모델링을 위한 기능은 구현이 불가능하다.
예) 지연시간 (a = #10 b), 트랜지스터 수준의 기술, 메모리, 등
b) 합성 툴이 지원하지 않는 경우
예) 나눗셈 및 %(modulo) 연산은 피연산자가 모두 상수인 경우만 합성 가능.
c) 하드웨어로 구현은 가능하지만, 제한된 면적을 초과하거나 목표성능(동작 주파수)에 미치지 못하는 경우

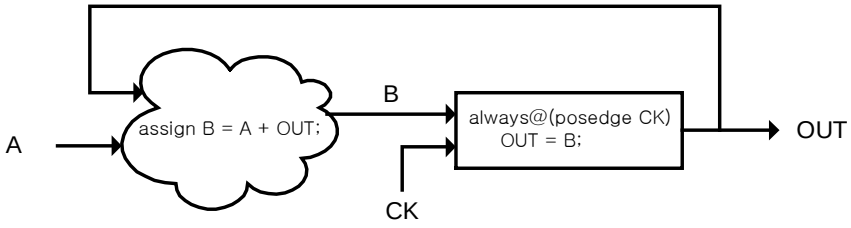
Verilog 는 실제회로로 구현하는 것 이외에도 모델링에도 사용되므로 매우 많은 기능을 지원한다. 이 기능들은 Verilog-XL 시뮬레이터에서는 올바르게 동작하지만, 합성이 불가능하거나 합성 결과가 RTL 과 다른 동작을 보이게 된다. Verilog 에 처음 입문한 사람들이 공통적으로 범하는 실수는 C 언어로 프로그램하듯이 설계를 하는 것이며, 이런 경우 대부분 합성이 불가능하다.

성공적으로 합성을 하기위해서는 마치 로직 게이트를 이용하여 회로를 그리듯이(schematic capture) Verilog 기술을 해야 한다. 즉, 회로를 미리 구상하고 Verilog 로 기술하는 것이다. 그렇다면, 합성 툴이 어떻게 합성할지를 미리 예상하여 Verilog 기술을 해야 하는데 이것은 어찌 보면 참 번거로운 작업이다. 차라리 schematic capture 방식(GUI 를 이용하여 회로를 제작하는)이 더 편한 방법일 수 있을 것이다. 하지만, 앞서 설명한 것처럼 Verilog 로 설계된 회로는 공정에 비의존적이라는 장점을 갖고 있으며, 합성 툴에서 다양한 최적화를 수행할 수 있다.

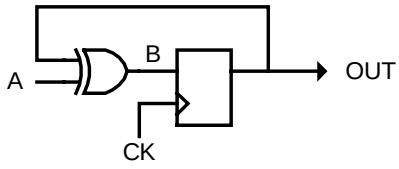
그림 5 의 예와 같이 먼저 회로의 구조를 대략적으로 설계한다. 이때, 조합논리회로(combinational logic)과 순차회로(sequential logic)으로 구분한다. 다음 Verilog 를 이용하여 조합논리회로는 assign 문을, 순차회로는 always 블록으로 기술한다. 이를 합성하면 기본구조는 유지되면서 최적화된 회로를 얻게 된다.



Initial Logic Design



Verilog Description



Synthesis Result

그림 5. 합성 가능한 설계방법

4.1. 조합논리회로의 설계

조합 논리회로를 기술하는 방법은 크게 가지가 있다. assign 문을 이용하는 방법과 always 블록을 이용하는 것이다. (그림 7)

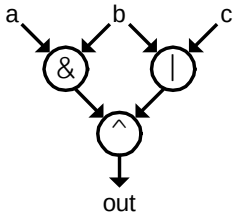


그림 6. 조합논리회로의 예

```
assign out = (a & b) ^ (b | c);
```

```
reg out;
always @(a or b) // missing input c
begin
   out = (a & b) ^ (b | c);
end
```

a) assign 문을 이용한 방법 b) always 블록을 이용한 방법

그림 7. 조합논리회로의 기술

어떤 경우나 같은 합성결과를 얻을 수 있으나 always 블록의 경우에는 sensitivity list 에 c 입력이 실수로 누락된 것을 볼 수 있다. 이런 경우, 그림 8 과 같이 원하지 않는 latch 가 합성된다. 따라서, 조합논리회로를 기술할 때에는 assign 문의 사용을 권장하며 부득이 always 블록을 사용할 경우에는 sensitivity list 에 모든 입력이 포함되도록 주의해야한다.

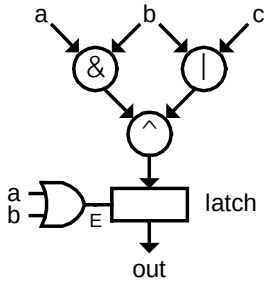


그림 8. 잘못된 합성의 예

마찬가지의 이유로 multiplexer 를 기술할 때에도 다음과 같이 assign 문의 사용을 권장한다.

```
assign out = sel ? a : c;
```

```
reg out;
always @(a or b or sel)
begin
  if(sel)out = a;
  else out = b;
end
```

a) assign 문을 이용한 방법 b) always 블록을 이용한 방법

그림 9. multiplexer 의 기술

4.2 순차회로의 설계

순차회로는 D latch 와 D flip-flop 을 이용하여 설계한다. 각각의 기술방법은 그림 10 과 같다.

```
module dlatch (d, ck, reset, q);
input d, ck, reset;
output q;
reg q;
always @(d or ck or reset)
   if(reset)
     q <= 0;
   else
     q <= d;
endmodule
```

a) D latch

```
module dff (d, ck, reset, q);
input d, ck, reset;
output q;
reg q;
always @(posedge ck or negedge reset)
   if(reset)
     q <= 0;
   else
     q <= d;
endmodule
```

b) D flip-flop

그림 10. 순차회로의 기술

순차회로의 기술 시 주의할 사항은 한 개의 always 블록에는 한 개의 단위 register 만이 존재하도록 하는 것이다. 그림 11 의 예를 보면 a)의 경우 한 개의 always 블록에서 a, b 두개의 register 에 대한 동작을 기술 하고 있다. a)와 b)는 동일한 회로를 기술하고 있는 것 같지만 assert_a 와 assert_b 가 모두 1 인 경우에는 다른 결과가 나타난다.

```
module seq_example (clk, reset,
assert_a, assert_b, a, b);
input clk, reset, assert;
output a, b;
reg a, b;
always @(posedge clk)
begin
  if(~reset)
  begin
    a = 0;
    b = 0;
  end
  else if(assert_a)
  begin
    a = 1;
  end
  else if(assert_b)
  begin
    b = 1;
  end
end
endmodule
```

a) 잘못된 경우

```
module seq_example (clk,reset,
assert_a, assert_b, a, b);
input clk, reset, assert;
output a, b;
reg a, b;
always @(posedge clk)
  if(~reset)
    a = 0;
  else if(assert_a)
    a = 1;
always @(posedge clk)
  if(~reset)
    b = 0;
  else if(assert_b)
    b = 1;
endmodule
```

b) 올바른 경우

그림 11. 순차회로의 기술 시 주의사항

다음 강좌에서는 Synopsys 사의 Design Compiler 를 이용하여 회로를 최적화 하는 방법에 대하여 설명한다.