

## 마이크로프로세서 설계 무작정 따라하기 part-III (3)

부제: 합성, 배치 및 배선

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

지난 강좌에서는 Synopsys 사의 Design Compiler 를 이용하여 회로를 합성하는 방법을 설명하였으며 조합논리회로를 최적화 하는 방법에 대하여 설명하였다. 이번 강좌에서는 클럭이 있는 회로 즉, 순차 회로를 최적화 하는 방법과 복잡한 회로를 효율적으로 합성하는 방법에 대하여 설명한다.

## 1. 조합회로와 순차회로의 차이점

조합논리회로의 최적화는 기본적으로 set\_max\_delay 와 set\_max\_area 의 constraint 를 이용하는 것으로 사용방법이 매우 간단하다. 이에 비하여 순차회로의 constraint 는 다소 복잡하다.

먼저 간단한 예제를 살펴보도록 하자

add16.v: 16-bit adder with registered outputs

---

```
module add16 (a, b, clk, out);
input  [15:0] a, b;
input      clk;
output [15:0] out;
reg      [15:0] out;
always@(posedge clk)
    out = a + b;
endmodule
```

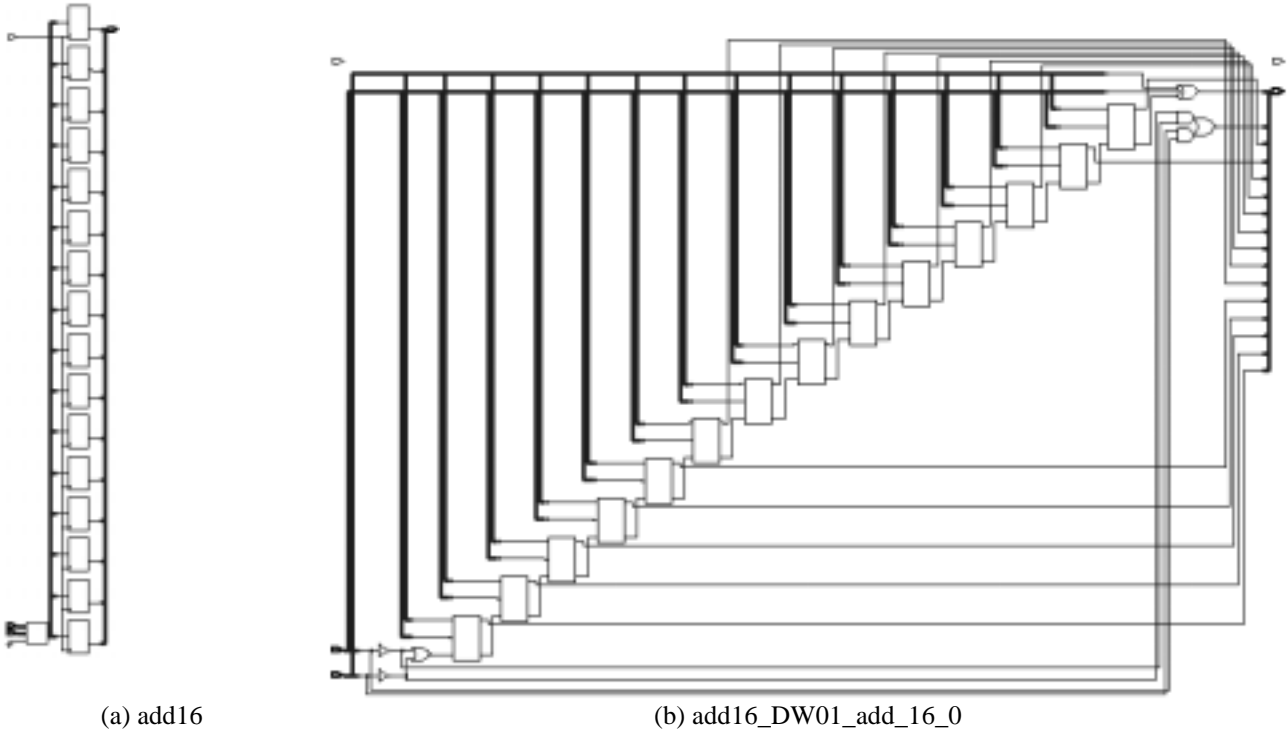


그림 1. add16 의 합성결과

그림 1 은 add16 를 constraint 없이 합성한 결과이다. 지난 회에서 설명한 것과 같이 Synopsys 에서는 덧셈기 합성을 위해서 DesignWare(그림 1b)를 사용하였고, 그 출력에 flip-flop 들이 있음을 볼 수 있다. 이 회로의 동작속도를 알아보기 위하여 report\_timing 명령을 실행해보자

```
dc_shell> report_timing
```

Point	Incr	Path
out_reg[15]/cp (dfnrq1)	0.00	0.00 r
out_reg[15]/q (dfnrq1)	0.49	0.49 f
out[15] (out)	0.00	0.49 f
data arrival time		0.49

위와 같이 0.49ns 의 결과를 얻었다. 그렇다면 이 회로는 2GHz 에서 동작한다는 의미일까? add16 의 critical path(임계 경로)를 살펴보면 출력단의 flip-flop(out\_reg)의 cp 입력에서 q 출력까지임을 알 수 있다. 그렇다면, 16-bit 덧셈기의 지연시간은 왜 포함되지 않은 것일까? 클럭신호가 상승하기 전까지는 입력의 변화가 출력에 영향을 미치지 못하고 클럭신호가 상승하면 그 순간의 입력이 출력으로 전달되기 때문이다. 이것이 바로 순차회로의 기본적인 특성이며 조합논리회로를 합성할 때와 다른 constraint 를 쓰게 되는 이유이다.

그렇다면 순수한 덧셈기 부분의 지연시간을 확인해보자.

```
dc_shell> current_design add16_DW01_add_16_0
```

```
dc_shell> report_timing
data arrival time 5.76
```

따라서, 실제로 이 회로가 동작할 수 있는 속도는  $1/(\text{덧셈기의 지연시간} + \text{flip-flop의 clock-to-q 지연시간}) = 1/(5.76 + 0.49) = 160 \text{ MHz}$  이다.

이 회로의 동작속도를 높이기 위해 `set_max_delay` 를 사용하여보자.

```
dc_shell> current_design add16
dc_shell> set_max_delay 0.1 -from {a,b} -to out
```

그러나, 이 회로의 `critical_path` 는 `a,b → out` 이 아니라 `clk → out` 이므로 `set_max_delay` 는 합성에 영향을 미치지 못한다.

## 2. 클록의 정의 (create\_clock)

이제 본격적으로 순차회로를 합성하는 방법을 알아보도록 하자. `add16` 을 `400MHz` 에서 동작하도록 최적화 해보자. 이를 위해서 먼저 `create_clock` 을 이용하여 클록을 정의한다.

```
dc_shell> create_clock -period 2.5 -waveform {0 1.25} clk
dc_shell> compile
dc_shell> report_timing
```

Point	Incr	Path
-----		
clock (input port clock) (rise edge)	0.00	0.00
input external delay	0.00	0.00 r
b[10] (in)	0.00	0.00 r
add_8/B[10] (add16_DW01_add_16_1)	0.00	0.00 r
add_8/SUM[12] (add16_DW01_add_16_1)	0.00	2.33 r
out_reg[12]/d (dfnrq1)	0.00	2.33 r
data arrival time		2.33
library setup time	-0.16	2.34
data required time		2.34
data arrival time		-2.33
-----		
slack (MET)		0.01

위와 같이 create\_clock 이 후 report\_timing 의 결과가 달라졌음을 알 수 있다. 즉, critical path 가 clk→out 에서 a, b→out 이 된 것이다. 그림 2 와 같이 덧셈기 내부의 회로도 변화하여 결과적으로 add16 은 400MHz 에 동작할 수 있는 회로가 되었다.

이와 같이 순차회로의 동작속도는 클럭주파수에 의해 결정되며, report\_timing 명령으로 정의된 클럭주파수에서 동작 할 수 있는지 여부를 알 수 있다. 위의 결과를 살펴보면 slack 이라는 용어가 나오는데, slack 의 의미는 ‘느슨한’으로 양의 값인 경우에는 timing 에 여유가 있음을 의미하며, 음의 값인 경우엔 주어진 조건에서 동작하지 못하는 것을 의미한다. 위의 경과에서는 0.01 로 slack 이 만족(MET)되었다 (반대의 경우에는 VIOLATED 로 표시된다).

또한, 400MHz 에 동작하기 위한 회로의 지연시간이 2.5ns 가 아니라 2.34ns(data required time)임을 볼 수 있는데, 이것은 setup time 을 만족하도록 (라이브러리에 정해진 setup time 을 기준으로) 계산된 값이다.

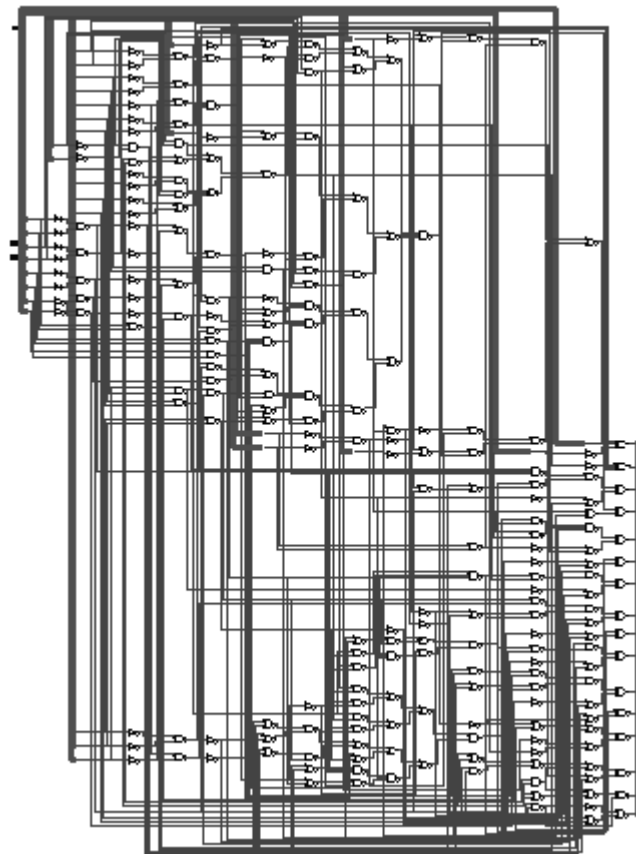


그림 2. 클럭 정의 후 합성결과 (add16\_DW01\_add\_16\_0)

### 3. 입/출력 지연시간 (set\_input\_delay, set\_output\_delay)

앞서 add16 을 합성할 때 기본적인 조건이 있었다. 즉, a 와 b 입력이 clock 과 함께 들어온다는 것이다. 즉, 그림 3a 와 같이 입력 지연시간이 0 인 경우이다. 하지만, add16 은 그림 3b 와 같은 형태로 사용될 수도 있으며 이러한 경우 적절히 입력지연시간을 합성 틀에 알려주어 add16 의 critical path delay 가 (clock period - T<sub>delay</sub>)를 만족하도록 하여야한다.

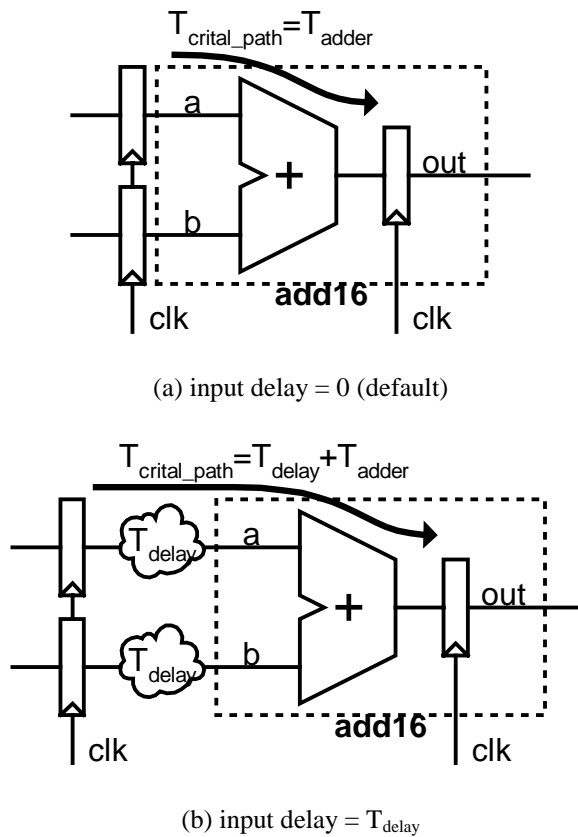


그림 3. 입력 지연시간 (input delay)

이와 같이 합성 틀에 입력이 클록을 기준으로 얼마나 늦게 들어올지 알려주는 constraint 가 바로 set\_input\_delay 이다. 예를 들어 a, b 의 입력에 지연시간이 각각 1ns 일 경우 다음과 같이 입력하면 된다.

```
dc_shell> set_input_delay 1.0 {a, b} -clock clk
```

이와 유사하게 출력에 대해서도 비슷한 경우가 발생한다. 입력에 레지스터가 있는 덧셈기(add16')를 가정해보자. 그림 4b 와 같이 출력이 다른 조합논리회로를 거쳐서 레지스터의 입력이 되는 경우 합성 틀에 출력지연시간( $T_{delay}$ )를 알려주어 덧셈기의 지연시간을 충분히 줄여야 한다. 출력지연시간을 설정하는 방법은 다음과 같다. ( $T_{delay} = 1.0ns$ )

```
dc_shell> set_output_delay 1.0 find(port,out*)
```

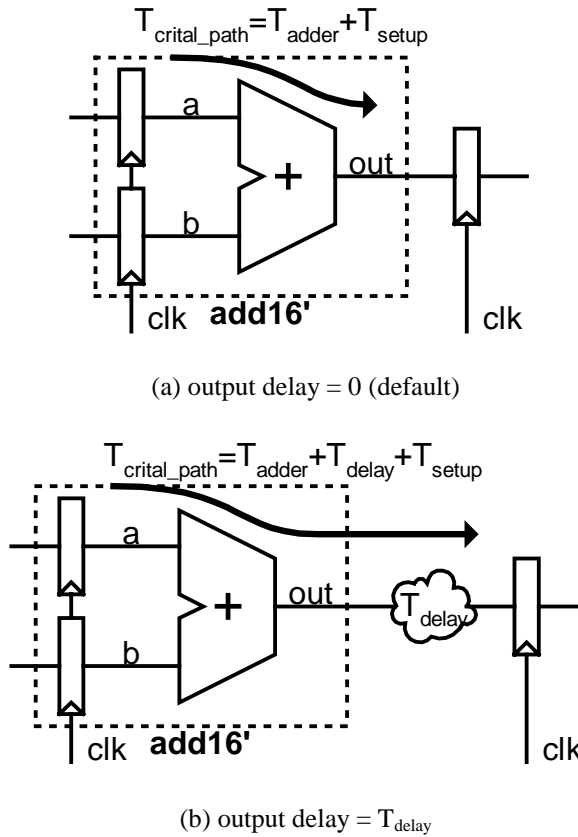


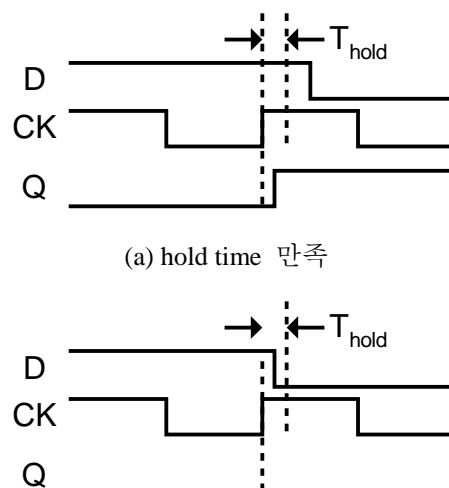
그림 4. 출력 지연시간 (output delay)

따라서, 임의의 순차회로의 critical path delay 는 clock period - input delay - output delay - setup time 을 만족해야 한다. 또한 앞에서 설명한 slack 은 다음과 같이 정의 될 수 있다.

$$\text{slack} = \text{clock period} - \text{input delay} - \text{output delay} - \text{setup time} - \text{critical path delay}$$

#### 4. 홀드타임 위반

홀드타임(hold time)은 flip-flop 이 올바르게 동작하기 위해서 클록이 변화한 다음 입력(D)를 유지해야 하는 시간을 말한다.



(b) hold time 위반

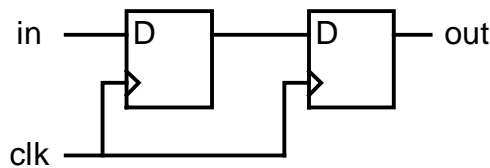
그림 5. hold time

그림 6a 와 같이 한 개 이상의 flip-flop 이 직접 연결된 회로를 생각해보자. 이와 같은 회로는 쉬프트레지스터를 만들거나 파이프라인 레지스터가 직접 연결된 경우에 나타날 수 있다. 이상적인 경우에는  $T_{\text{clock-to-Q}} > T_{\text{hold}}$  조건을 만족하여 hold time violation 이 발생하지 않는다. 하지만, 실제 회로에서는 매우 위험한 경우에 해당한다. 일반적으로 clock net 은 보통 net 보다 load 가 매우 크고 칩의 넓은 면적을 지나기 때문에 전달 속도가 느리다(clock skew). 따라서 이러한 지연시간( $T_{\text{delay}}$ )을 고려해야 한다. 즉,  $T_{\text{clock-to-Q}} > T_{\text{hold}} + T_{\text{delay}}$  이어야 올바른 동작을 할 수 있다.

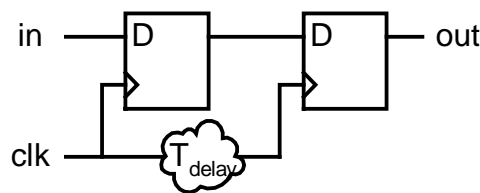
이를 해결하기 위한 방법은 그림 6c 와 같이 지연시간을 추가하는 방법과 하강엣지에서 동작하는 flip-flop 을 추가하는 방법이 있다. Synopsys 에서는 전자의 방법으로 set\_fix\_hold 기능을 제공한다.

set\_fix\_hold 의 사용 예는 다음과 같다.

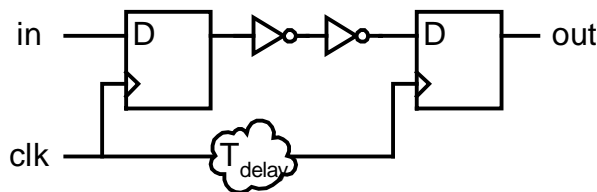
```
dc_shell> set_clock_uncertainty 0.5
dc_shell> set_fix_hold
```



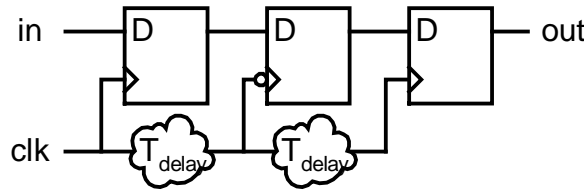
(a) 이상적인 경우(no violation)



(b) 실제의 경우( $T_{\text{clock-to-Q}} < T_{\text{hold}} + T_{\text{delay}}$  이면 violation)



(c) 해결방법 1



(d) 해결방법 2

그림 6. hold time 위반(violation)의 예

5. 클럭신호의 버퍼링

Clock net 의 load 가 커지면 Synopsys 는 자동으로 다단계의 buffer 를 추가한다. 그러나, 이것이 항상 바람직한 것은 아니다. 그림 7 과 같이 외부에서 강력한 buffer 가 연결되어있는데, 블록 내부에서 buffering 을 잘못하면 오히려 구동력을 약화시킬 수도 있다. 이것은 clock net 뿐만 아니라 보통 net 에서 동일하게 발생하는 문제이다. 외부에 강력한 buffer 가 있음을 합성 틀에 알려주는 방법은 다음과 같다. (cell 이름이 inv0d1 이고 구동하는 port 이름이 clk 인 경우)

```
dc_shell> set_driving_cell -lib_cell inv0d1 {clk}
```

만일 레이아웃을 생성하는 후단계(back-end)과정에서 CTS(clock tree synthesis)가 가능하다면 clock net 은 don't touch 시키는 것이 가장 효율적이다. 즉, 전체 clock net 에 buffer 를 전혀 사용하지 않는 것이다. CTS 는 후단계에서 레이아웃상의 위치와 load 에 따라서 최적화된 buffer tree 를 합성하는 기능을 한다. Clock net 을 don't touch 하는 방법은 다음과 같다.

```
dc_shell> set_dont_touch find(net, clk)
```

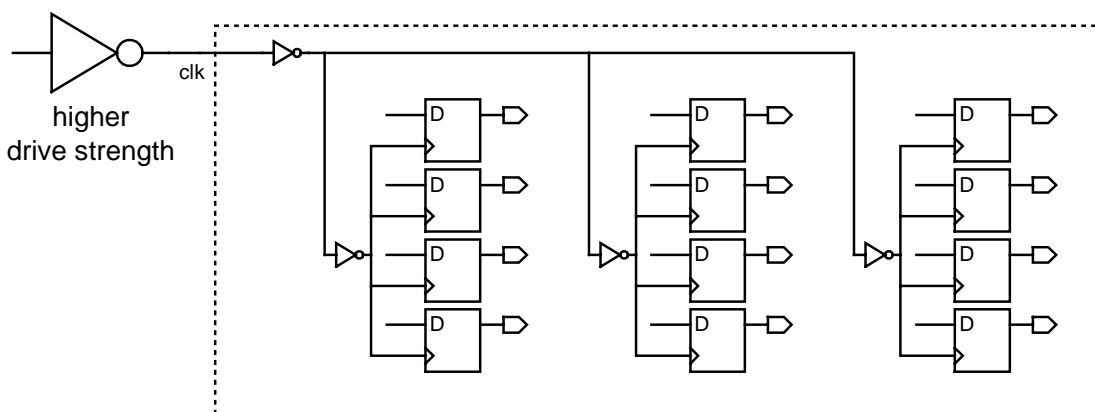


그림 7. 클럭신호의 버퍼링

지금까지 순차회로를 합성하는 방법과 중요한 명령어(command)들을 살펴보았다. 위에서 설명된 명령어들은 다양한 추가기능을 제공하므로 help [명령어] 기능을 사용하여 정확한 기능을 숙지하도록 한다. 다음 회에서는 후단계 설계방법에 대해서 설명한다.