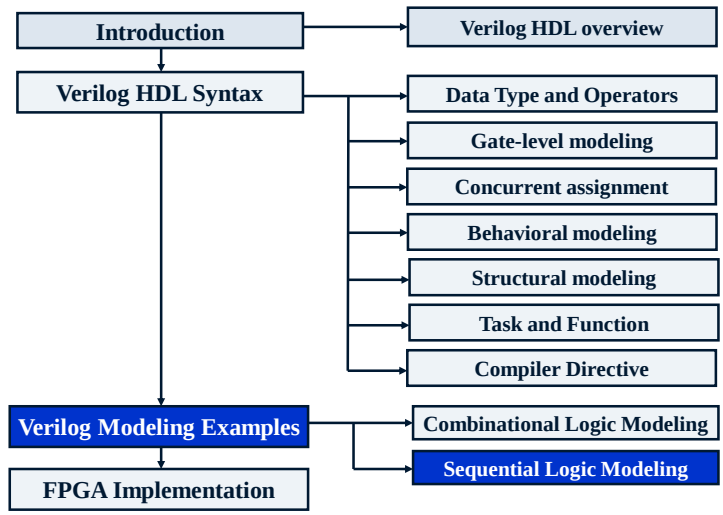
순차회로 모델링(1)

Kyung-Wook Shin
kwshin@kumoh.ac.kr

School of Electronic Eng.,
Kumoh National Institute of Technology

Learning Map

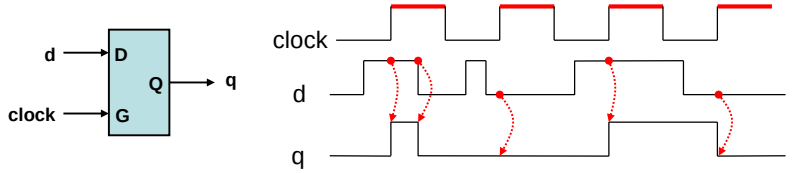


11.0 순차회로 모델링

- 순차회로
  - 현재의 입력, 과거의 입력, 회로에 기억된 상태값에 의해 출력이 결정
  - 과거의 입력, 현재의 상태값을 저장하는 기억소자(래치 또는 플립플롭)와 조합논리회로로 구성
  - 데이터 레지스터, 시프트 레지스터, 계수기(counter), 직렬/병렬 변환기, 유한상태머신(Finite State Machine; FSM), 주파수 분주기, 펄스 발생기 등
    - 클록신호에 의해 동작되는 래치 또는 플립플롭을 포함
- 래치와 플립플롭
  - 래치 : 클록신호의 레벨(즉, 0 또는 1)에 따라 동작하는 저장소자
  - 플립플롭 : 클록신호의 상승 또는 하강에지에 동기되어 동작하는 저장소자
  - always 구문 내부에 if 조건문을 이용하여 모델링
- 순차회로의 모델링
  - always 블록을 이용한 행위수준 모델링, 게이트 프리미티브 및 하위모듈 인스턴스, 연속 할당문 등 다양한 Verilog 구문들이 사용됨
  - 할당문의 형태 (nonblocking 또는 blocking) 에 따라 회로의 동작과 구조가 달라짐

11.1.1 래치(Latch)

- Positive level-sensitive D latch



Positive level-sensitive D latch

```
module dlatch(clock, d, q);
   input  clock, d;
   output q;
   reg    q;

  always @(clock or d) begin
      if(clock)
         q = d;
  end
endmodule
```

코드 11.1

11.1.1 래치(Latch)

Testbench for D latch

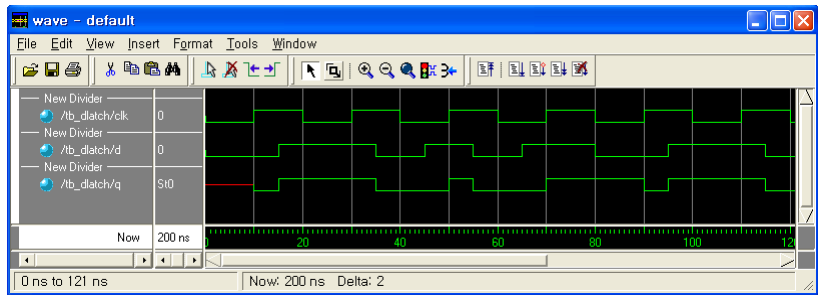
```
module tb_dlatch ;
   reg clk, d;

   dlatch U0(clk, d, q);

   initial begin
     clk = 1'b0;
     forever #10 clk = ~clk;
   end
   initial begin
      d = 1'b0;
      forever begin
        #15 d = 1'b1;  #20 d = 1'b0;
        #10 d = 1'b1;  #10 d = 1'b0;
        #10 d = 1'b1;  #15 d = 1'b0;
      end
   end
endmodule
```

코드 11.2

11.1.1 래치(Latch)



[그림 11.2] [코드 11.1]의 시뮬레이션 결과

설계과제 11.1

❑ **Negative level-sensitive** 방식으로 동작하는 **8**비트 **D** 래치 회로를 설계하고, 테스트벤치를 작성하여 기능을 검증한다.

11.1.1 래치(Latch)

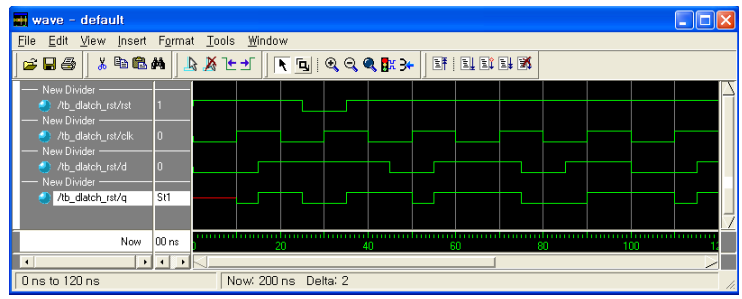
❑ Active-low 리셋을 갖는 positive level-sensitive D latch

```
module dlatch_rst(rst, clock, d, q);
   input  rst, clock, d;
   output q;
   reg    q;

  always @(clock or rst or d) begin
     if(!rst)
        q = 1'b0;
     else if(clock)
        q = d;
  end
endmodule
```

코드 11.3

11.1.1 래치(Latch)



[그림 11.3] [코드 11.3]의 시뮬레이션 결과

설계과제 11.2

❑ Active-low 셋과 리셋을 갖는 negative level-sensitive 8비트 D 래치 회로를 설계하고, 테스트벤치를 작성하여 기능을 검증한다.

11.1.1 래치(Latch)

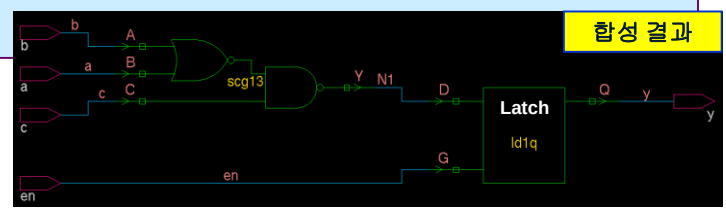
❑ Latch가 포함된 회로에 blocking 할당문이 사용된 경우

코드 11.4

```
module latch_blk(en, a, b, c, y);
   input  en, a, b, c;
   output y;
   reg    m, y;

  always @(en or a or b or c) begin
      if(en) begin
          m = ~(a | b);
          y = ~(m & c);
      end
  end
endmodule
```

합성 결과



11.1.1 래치(Latch)

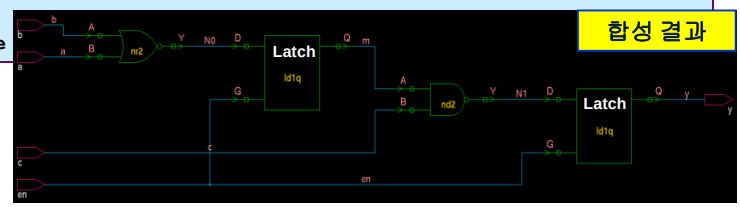
❑ Latch가 포함된 회로에 nonblocking 할당문이 사용된 경우

코드 11.5

```
module latch_nonblk(en, a, b, c, y);
   input  en, a, b, c;
   output y;
   reg    m, y;

  always @(en or a or b or c) begin
      if(en) begin
          m <= ~(a | b);
          y <= ~(m & c);
      end
  end
endmodule
```

합성 결과



11.1.1 래치(Latch)

설계과제 11.3

- 코드 **11.6**은 코드 **11.4**에서 **blocking** 할당문의 순서를 바꾼 경우이다. 코드 **11.4**와 동일한 회로인지, 만약 다른 회로라면 어떤 회로가 되는지 시뮬레이션과 합성을 통해 확인하고, 그 이유를 생각해 본다.

코드 11.6

```
module latch_blk2(en, a, b, c, y);
   input  en, a, b, c;
   output y;
   reg    m, y;

  always @(en or a or b or c) begin
      if(en) begin
          y = ~(m & c);
          m = ~(a | b);
      end
  end
endmodule
```


11.1.1 래치(Latch)

설계과제 11.4

- 코드 **11.7**은 코드 **11.5**에서 **nonblocking** 할당문의 순서를 바꾼 경우이다. 코드 **11.5**와 동일한 회로인지, 만약 다른 회로라면 어떤 회로가 되는지 시뮬레이션과 합성을 통해 확인하고, 그 이유를 생각해 본다.

코드 11.7

```
module latch_nonblk2(en, a, b, c, y);
   input  en, a, b, c;
   output y;
   reg    m, y;

  always @(en or a or b or c) begin
      if(en) begin
          y <= ~(m & c);
          m <= ~(a | b);
      end
  end
endmodule
```