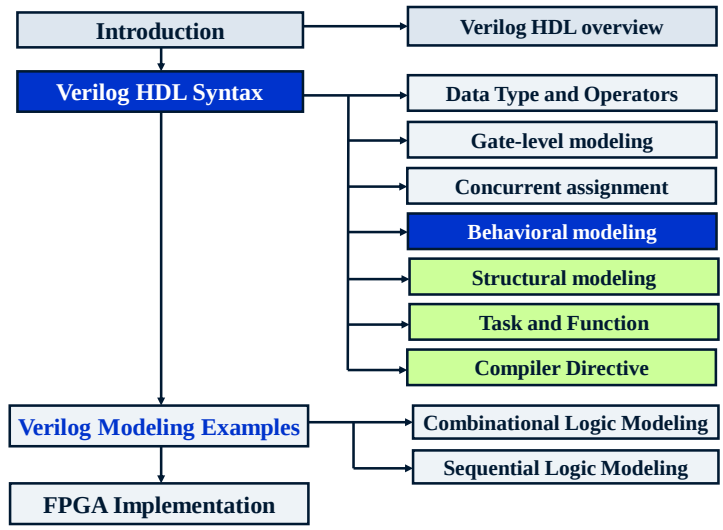
행위수준 모델링

Kyung-Wook Shin
kwshin@kumoh.ac.kr

School of Electronic Eng.,
Kumoh National Institute of Technology

Learning Map



5.1.1 always 구문

- 행위수준 모델링
  - 조합논리회로와 순차논리회로의 설계, 설계된 회로의 시뮬레이션을 위한 테스트벤치의 작성에 사용
  - always 구문, initial 구문, task, function 내부에 사용
- always 구문

```
always [@(sensitivity_list)] begin
   blocking_or_nonblocking statements;
end
```

  - @(sensitivity_list)는 always 문의 실행을 제어
    - sensitivity_list (감지신호목록)에 나열된 신호들 중 하나 이상에 변화(event)가 발생했을 때 always 내부에 있는 begin – end 블록의 실행이 트리거됨
    - begin – end 블록은 절차형 문장들로 구성
    - blocking 할당문 또는 nonblocking 할당문에 따라 실행 방식이 달라짐
    - 시뮬레이션이 진행되는 동안 무한히 반복 실행됨

5.1.1 always 구문

예 5.1.1 always 구문을 이용한 조합논리회로 모델링

2입력 OR 게이트

```
module sample(a, b, out);
  input  a, b;
  output out;
  reg    out;

  always @(a or b) begin // 감지신호 목록의 or를 콤마(,)로 대체 가능함
                         //  always @(a, b) 또는 always @(*)
     if(a==1 || b==1) out = 1;   // blocking 할당문
     else             out = 0;   // blocking 할당문
  end
endmodule
```

코드 5.1

5.1.1 always 구문

예 5.1.2 always 구문을 이용한 순차회로 모델링

D 플립플롭

```
module dff(clk, din, qout);
  input  clk, din;
  output qout;
  reg    qout;

  always @(posedge clk)
       qout <= din;     // Non-blocking 할당문
endmodule
```

코드 5.2

5.1.1 always 구문

- always 구문의 sensitivity_list (감지신호목록)
  - 조합논리회로 모델링
    - always 구문으로 모델링되는 회로의 입력 신호가 모두 나열되어야 함
    - 일부 신호가 감지신호목록에서 빠지면, 합성 이전의 RTL 시뮬레이션 결과와 합성 후의 시뮬레이션 결과가 다를 수 있음
    - 함축적 감지신호 표현 (@*) 을 사용 가능
  - 순차회로 모델링
    - 동기식 셋/리셋을 갖는 경우 : 클록신호만 포함
    - 비동기식 셋/리셋을 갖는 경우 : 클록신호, 셋, 리셋신호를 포함

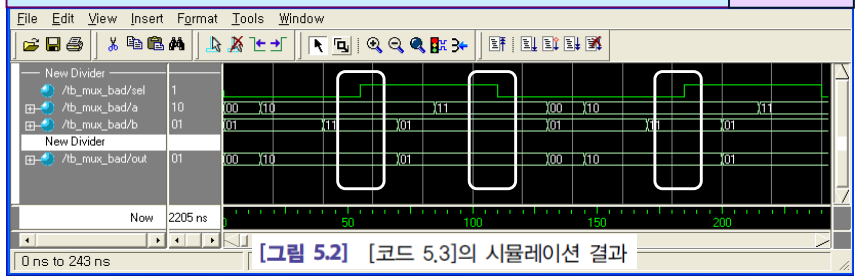
```
always @(*)      // equivalent to @(a or b or c or d or f)
   y =(a & b) |(c & d) | f;
```

5.1.1 always 구문

감지신호목록에 sel이 빠진 경우

```
module mux21_bad(a, b, sel, out);
   input  [1:0] a, b;
   input        sel;
   output [1:0] out;
   reg    [1:0] out;

   always @(a or b)  // sel is omitted
      if(sel ==0)
         out = a;
      else
         out = b;
endmodule
```

코드 5.3



[그림 5.2] [코드 5.3]의 시뮬레이션 결과

5.1.1 always 구문

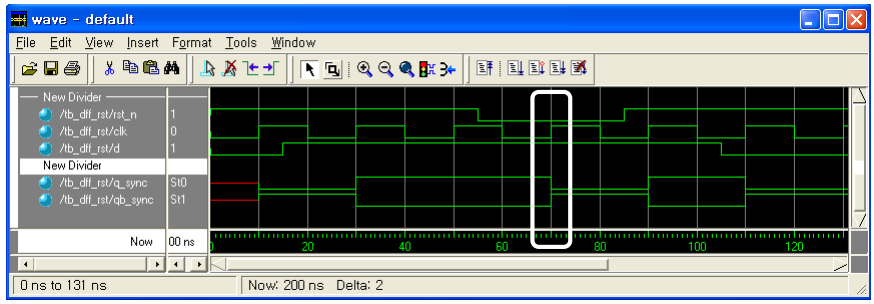
참고: 동기식 Active-Low reset을 갖는 D 플립플롭

```
module dff_sync_rst(clk, d, rst_n, q, qb);
  input  clk, d, rst_n;
  output q, qb;
  reg    q;

  assign qb = ~q;

  always @(posedge clk) // include only clk
  begin
    if(!rst_n)          // active-low reset
       q <= 1'b0;
    else
       q <= d;
  end
endmodule
```

코드 11.11

5.1.1 always 구문

참고: 동기식 Active-Low reset을 갖는 D 플립플롭



코드 11.11의 시뮬레이션 결과

5.1.1 always 구문

- always 구문이 테스트벤치에 사용되는 경우
  - 시뮬레이션 시간의 진행에 관련된 제어가 포함되어야 함
  - 그렇지 않으면 zero-delay 무한 루프(infinite loop)가 발생되어 교착 상태 (deadlock)에 빠지게 되어 시뮬레이션이 진행되지 않음

```
always
  clk = ~clk;  // zero-delay infinite loop
```

```
always
  #20 clk = ~clk;  // 주기가 40ns인 신호 clk를 생성
```

```
initial
   clk = 1'b0;
always
  #20 clk = ~clk;  // 주기가 40ns인 신호 clk를 생성
```

5.1.2 initial 구문

❑ initial 구문

```
initial begin
    blocking_or_nonblocking statements;
end
```

- ❖ 시뮬레이션이 실행되는 동안 한번만 실행
- ❖ 절차형 문장들로 구성되며, 문장이 나열된 순서대로 실행
- ❖ 논리합성이 지원되지 않으므로 시뮬레이션을 위한 테스트벤치에 사용

예 5.1.3

```
initial begin
  areg = 0;                          // initialize areg
  for(index = 0; index < size; index = index + 1)
     memory[index] = 0;              //initialize memory word
end
```


5.1.2 initial 구문

예 5.1.4 시뮬레이션 입력벡터 생성

```
initial begin
      din = 6'b000000;   // initialize at time zero
  #10 din = 6'b011001;   // first pattern
  #10 din = 6'b011011;   // second pattern
  #10 din = 6'b011000;   // third pattern
  #10 din = 6'b001000;   // last pattern
end
```

5.1.2 initial 구문

예 5.1.5 주기 신호의 생성

코드 5.4

```
module behave;
  reg  a, b;

  initial begin  // 초기값 지정
    a = 1'b1;
    b = 1'b0;
  end

  always
    #50 a = ~a;

  always
    #100 b = ~b;
endmodule
```

[그림 5.4] [코드 5.4]의 시뮬레이션 결과