순차회로 모델링(3)

Kyung-Wook Shin
kwshin@kumoh.ac.kr

School of Electronic Eng.,
Kumoh National Institute of Technology

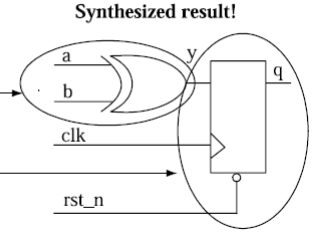
11.2 Blocking과 Nonblocking 할당문

❑ 코딩 가이드라인

❖ 가이드라인-1 : 순차회로 또는 래치를 모델링하는 always 블록에서는 nonblocking 할당문을 사용한다.

❖ 가이드라인-2 : 조합논리회로를 모델링하는 always 블록에서는 blocking 할당문을 사용한다.

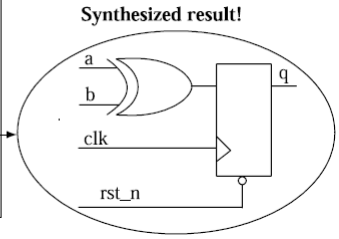
```
module nbex1 (q, a, b, clk, rst_n);
  output q;
  input  clk, rst_n;
  input  a, b;
  reg    q, y;

  always @(a or b)
    y = a ^ b;

  always @(posedge clk or negedge rst_n)
    if (!rst_n) q <= 1'b0;
    else        q <= y;
endmodule
```



11.2 Blocking과 Nonblocking 할당문

❖ 가이드라인-3 : 동일한 always 블록에서 순차회로와 조합논리회로를 함께 표현하는 경우에는 nonblocking 할당문을 사용한다.

```
module nbex2 (q, a, b, clk, rst_n);
  output q;
  input  clk, rst_n;
  input  a, b;
  reg    q;

  always @(posedge clk or negedge rst_n)
    if (!rst_n) q <= 1'b0;
    else        q <= a ^ b;
endmodule
```



11.2 Blocking과 Nonblocking 할당문

❖ 가이드라인-4 : 동일한 always 블록 내에서 blocking 할당문과 nonblocking 할당문을 혼합해서 사용하지 않는다.

```
module ba_nba1(q, a, b, clk, rst_n);
   output q;
   input  a, b, rst_n, clk;
   reg    q, tmp;
   always @(posedge clk or negedge rst_n)
      if(!rst_n) q <= 1'b0;
      else begin
         tmp = a & b;
         q  <= tmp;
      end
endmodule
```

Not Recommended 코드 11.18(a)

```
module ba_nba2(q, a, b, clk, rst_n);
   output q;
   input  a, b, rst_n, clk;
   reg    q, tmp;
   always @(posedge clk or negedge rst_n)
      if(!rst_n) q = 1'b0;     //blocking
      else begin
         tmp = a & b;
         q <= tmp;             //nonblocking
      end
endmodule
```

Not Recommended 코드 11.18(b)

11.2 Blocking과 Nonblocking 할당문


```
module ba_nba3(q, a, b, clk, rst_n);
   output q;
   input  a, b, rst_n, clk;
   reg    q;

   always @(posedge clk or negedge rst_n)
      if(!rst_n) q <= 1'b0;
      else       q <= a & b;
endmodule
```

Recommended 코드 11.18(c)

```
module ba_nba4(q, a, b, clk, rst_n);
   output q;
   input  a, b, rst_n, clk;
   reg    q;
   wire   tmp;

   assign tmp = a & b;

   always @(posedge clk or negedge rst_n)
      if(!rst_n) q <= 1'b0;
      else       q <= tmp;
endmodule
```

Recommended 코드 11.18(d)

11.2 Blocking과 Nonblocking 할당문

❖ 가이드라인-5 : 다수의 always 블록에서 동일한 reg 변수에 값을 할당하지 않는다.

```
module badcode1(q, d1, d2, clk, rst_n);
   output q;
   input  d1, d2, clk, rst_n;
   reg    q;

   always @(posedge clk or negedge rst_n)
      if(!rst_n) q <= 1'b0;
      else       q <= d1;

   always @(posedge clk or negedge rst_n)
      if(!rst_n) q <= 1'b0;
      else       q <= d2;
endmodule
```

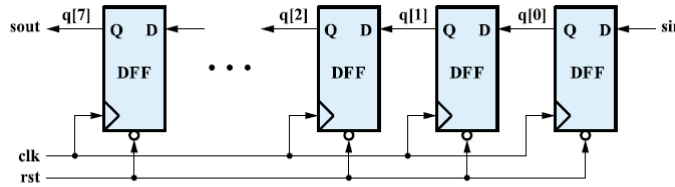
코드 11.19

Multiple source driving이 발생하는 코딩

11.3 시프트 레지스터

- 시프트 레지스터
  - 클록신호가 인가될 때마다 데이터가 왼쪽 또는 오른쪽으로 이동되는 회로
  - 여러 개의 플립플롭이 직렬로 연결된 구조
  - 형태
    - 직렬입력 – 직렬출력 (Serial-In, Serial-Out)
    - 직렬입력 – 병렬출력 (Serial-In, Parallel-Out)
    - 병렬입력 – 직렬출력 (Parallel-In, Serial-Out)
    - 병렬입력 – 병렬출력 (Parallel-In, Parallel-Out)
    - 왼쪽 시프트, 오른쪽 시프트, 양방향 시프트
  - nonblocking 할당문, 시프트 연산자, 결합 연산자, 반복문 등 다양한 구문으로 모델링

11.3.1 직렬입력-직렬출력 시프트 레지스터



[그림 11.17] 직렬입력–직렬출력 시프트 레지스터

```
module shift_reg_nblk1(clk, rst, sin, sout);
  input     clk, rst, sin;
  output    sout;
  reg [7:0] q;

  assign sout = q[7];

  always @(posedge clk) begin
    if(!rst)
       q <= 8'b0;
    else begin
       q[0] <= sin;
       q[1] <= q[0];
       q[2] <= q[1];
       q[3] <= q[2];
       q[4] <= q[3];
       q[5] <= q[4];
       q[6] <= q[5];
       q[7] <= q[6];
    end
  end
endmodule
```

코드 11.21

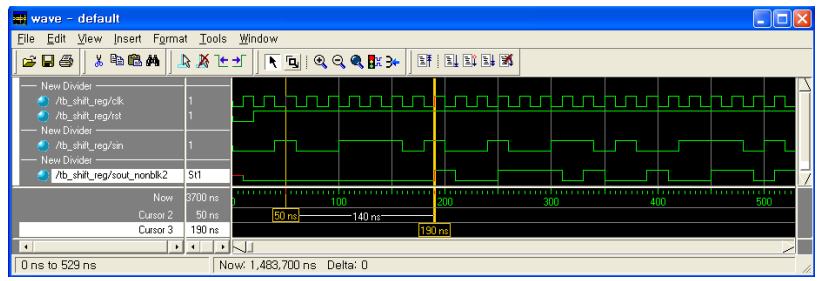
11.3.1 직렬입력-직렬출력 시프트 레지스터

```
module shift_reg_nblk2(clk, rst, sin, sout);
   input      clk, rst, sin;
   output     sout;
   reg  [7:0] q;

   assign sout = q[7];

   always @(posedge clk) begin
      if(!rst)
         q <= 0;
      else begin
         q[0]   <= sin;
         q[7:1] <= q[6:0];
      end
   end
endmodule
```

```
if (!rst)
   q <= 0;
else begin // ALT code
   q[7:1] <= q[6:0];
   q[0]   <= sin;
end
```

벡터 할당문을 사용한 경우 코드 11.22

11.3.1 직렬입력-직렬출력 시프트 레지스터



[그림 11.18] [코드 11.22]의 시뮬레이션 결과

11.3.1 직렬입력-직렬출력 시프트 레지스터

설계과제 11.9

❑ 그림 11.17의 active-low 동기식 리셋을 갖는 시프트 레지스터를 다음의 방법으로 모델링하고, 시뮬레이션한다.

① 결합 연산자를 사용하는 방법

② 시프트 연산자를 사용하는 방법

③ for 반복문을 사용하는 방법

```
if (!rst)
   q <= 8'b0;
else //결합 연산자
   q <= {q[6:0],sin};
```

```
if (!rst)
   q <= 8'b0;
else begin //시프트 연산자-①
   q    <= q << 1;
   q[0] <= sin;
end
```

```
if (!rst)
   q <= 8'b0;
else begin //시프트 연산자-②
   q[0] <= sin;
   q    <= q << 1;
end
```


11.4 계수기 (Counter)

❑ 계수기(counter)

- ❖ 클록펄스가 인가될 때마다 값을 증가 또는 감소시키는 회로
- ❖ 주파수 분주기, 타이밍 제어신호 생성 등에 사용
- ❖ 동기식 계수기
  - ➢ 모든 플립플롭이 하나의 공통 클록신호에 의해 구동되며, 모든 플립플롭의 상태변경이 동시에 일어남
  - ➢ 장점 : 설계와 검증이 용이하며, 계수 속도가 빠름
  - ➢ 단점 : 비동기식 카운터에 비하여 회로가 복잡함
- ❖ 비동기식 계수기
  - ➢ 첫단의 플립플롭에 클록신호가 인가되면, 플립플롭의 출력이 다음 단의 플립플롭을 트리거시키는 방식으로 동작
  - ➢ 리플 계수기(ripple counter)라고도 함
  - ➢ 장점 : 동기식 계수기에 비해 회로가 단순해짐
  - ➢ 단점 : 각 플립플롭의 전파 지연시간이 누적되어 최종단의 출력에 나타나므로 계수속도가 느림

11.4 계수기 (Counter)

❑ 8비트 증가 계수기

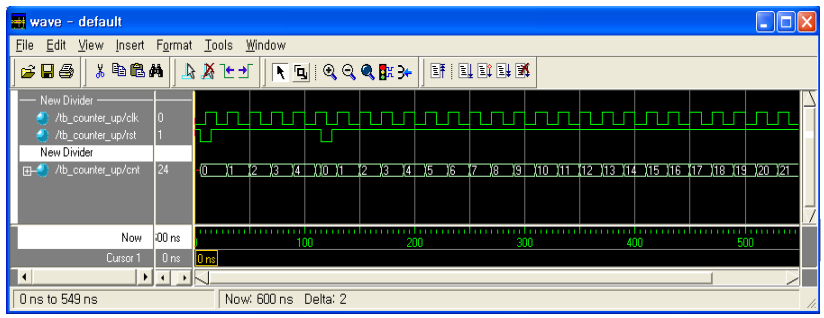
```
module counter_up(clk, rst, cnt);
  input        clk, rst;
  output [7:0] cnt;
  reg    [7:0] cnt;

  always @(posedge clk or negedge rst) begin
         if(!rst) cnt <= 0;
         else     cnt <= cnt + 1;
  end
endmodule
```

코드 11.26

Testbench: TB_chap11/tb_counter_up.v

11.4 계수기 (Counter)

❑ 8비트 증가 계수기



[그림 11.26] [코드 11.26]의 시뮬레이션 결과

설계 과제 11.13

- Active-high enable 신호를 갖는 8비트 감소 계수기를 설계하고, 시뮬레이션을 통해 동작을 확인한다. enable 신호 en=1이면 계수기가 동작하고, en=0이면 계수동작을 멈추는 기능을 가지며, Active-low 비동기식 리셋을 갖는다.

```
module counter_dn_en (clk, rst, en, cnt);
    input         clk, rst, en;
    output [7:0]  cnt;
    reg    [7:0]  cnt;

  always @(posedge clk or negedge rst) begin
       if (!rst) cnt <= 0;
       else if (en)
            cnt <= cnt - 1;
  end
endmodule
```


설계 과제 11.14

- Mode 신호에 따라 계수기 값이 증가(mode=1) 또는 감소(mode=0)되는 8비트 증가/감소 계수기를 설계하고, 시뮬레이션을 통해 동작을 확인한다. Active-low 비동기식 리셋을 갖는다.

```
module counter_ud (clk, rst, mode, cnt);
   input          clk, rst, mode;
   output [7:0]   cnt;
   reg    [7:0]   cnt;

always @(posedge clk or negedge rst) begin
     if (!rst) cnt <= 0;
     else begin
         if (mode) cnt <= cnt + 1;
         else      cnt <= cnt - 1;
     end
end
endmodule
```

11.4 계수기 (Counter)

❑ 1/10 주파수 분주기(frequency divider)

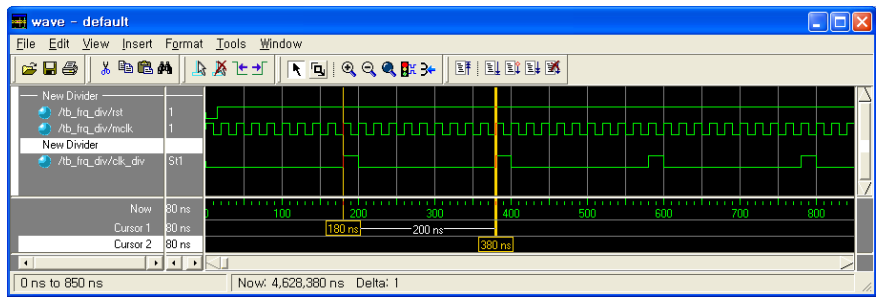
```
module frq_div(mclk, rst, clk_div);
  input     rst, mclk;
  output    clk_div;
  reg [3:0] cnt;
  reg       clk_div;

  always @(posedge mclk or negedge rst) begin
     if(!rst) begin
          cnt <= 0;
          clk_div <= 0;
     end
     else begin
          if(cnt == 9) begin
              cnt <= 0;
              clk_div <= 1'b1;
          end
          else begin
              clk_div <= 1'b0;
              cnt <= cnt + 1;
          end
     end
  end
endmodule
```

코드 11.27

Testbench: TB_chap11/tb_frq_div.v

11.4 계수기 (Counter)

❑ 1/10 주파수 분주기(frequency divider)



[그림 11.27] [코드 11.27]의 시뮬레이션 결과

설계과제 11.16

❑ Duty cycle이 50%인 대칭 분주 클록을 생성하는 1/10 주파수 분주기를 설계하고 시뮬레이션을 통해 동작을 확인한다. Active-low 비동기식 리셋을 갖는다.

설계 과제 11.16


```
module frq_div_sym (mclk, rst, clk_div);
    input        rst, mclk;
    output       clk_div;
    reg    [3:0] cnt;
    reg          clk_div;

  always @(posedge mclk or negedge rst) begin
     if (!rst) begin
         cnt <= 0;
         clk_div <= 0;
     end
     else
         if (cnt == 4) begin
              cnt <= 0;
              clk_div <= ~clk_div;
         end
         else cnt <= cnt + 1;
  end
endmodule
```