순차논리회로 및 기타 논리회로 설계

Lecture's Goal

- 순차논리 회로를 설계하기 위해 FSM도를 작성하고 Verilog HDL로 설계하는 과정을 알아보고, 설계된 순차논리 회로를 시뮬레이션으로 설계를 검증하고 동작을 확인한다.
- 기타 논리회로를 설계할 때 사용 할 수 있는 설계 방법에 대해서 알아보고, 설계된 회로를 시뮬레이션으로 확인함으로써 설계를 검증한다.

Agenda

1. 순차논리회로 설계

   1) 간단한 상태도의 구현

   2) 레지스터의 구현

   3) Up-down 카운터

   4) 순차 검출기

2. 기타 논리회로 설계

   1) 클럭을 사용하는 회로와 사용하지 않는 회로

   2) 스텝 클럭(펄스) 발생회로

   3) 양방향 버스

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

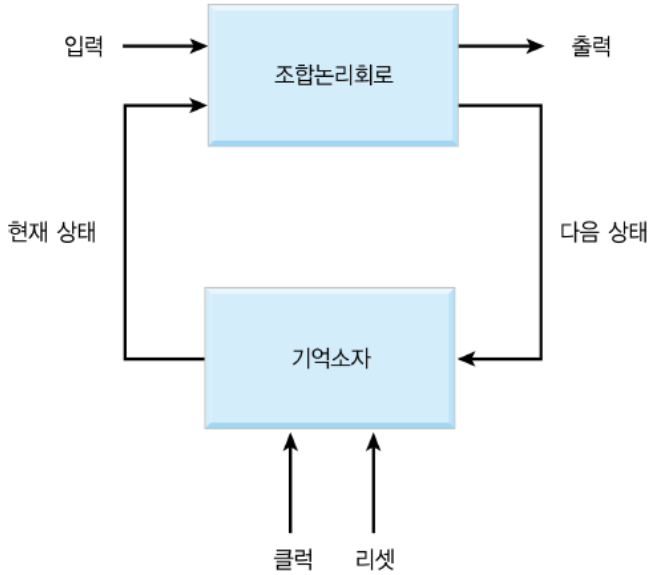
순차논리회로와 상태도

상태도(FSM : Finite State Machine)

- 조합논리회로만으로 디지털 논리회로를 설계하는 것이 쉽지 않음
- 상태도는 순차논리회로의 동작을 표현 한 후에 Verilog HDL로 설계
- FSM(Finite State Machine)은 순차논리회로를 설계하는 하나의 방법

순차논리회로의 구성요소

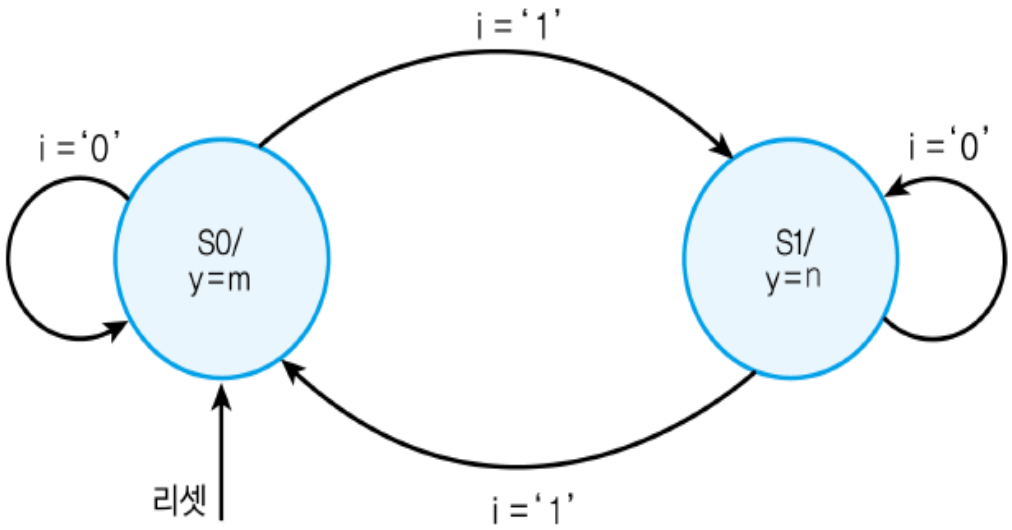
- 기억소자 : 플립 플롭을 포함하며 순차 논리 회로의 상태를 기억
- 조합논리회로 : 외부 입력과 상태 정보 등 두 입력 신호와, 다음 상태 정보와 외부 출력 신호 등 두 출력 신호가 있음
- 클럭 : FSM의 상태가 변하는 동기 클럭이 필요



순차논리회로와 상태도

무어 머신(Moore Machine), 무어 모델(Moore Model)

- 상태 머신(State Machine)이 현재 상태에만 영향을 받음

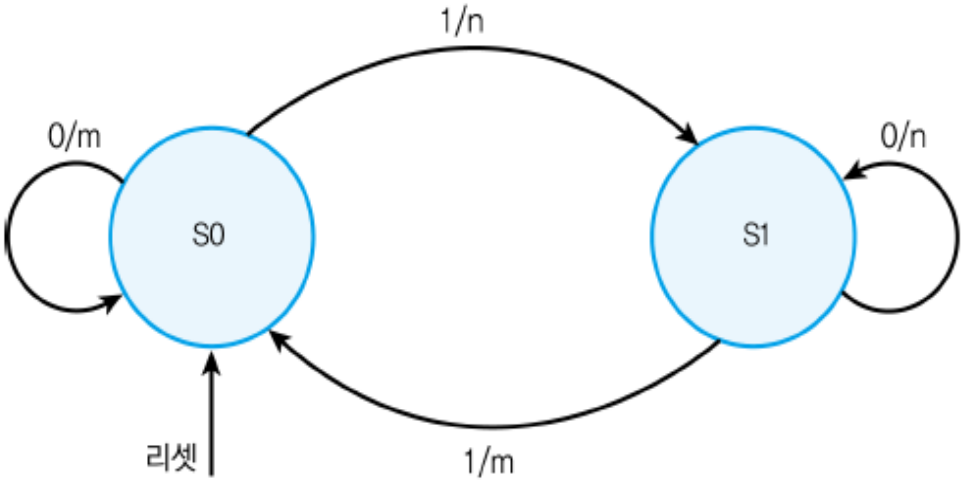


- 초기 상태에서 클럭의 에지에서 i가 '0'이면 S0상태 유지, '1'이면 S1으로 천이
- 출력 y는 S0에서 항상 입력 m을 출력하고, S1에서 항상 입력 n을 출력

순차논리회로와 상태도

밀리 머신(Mealy Machine), 밀리 모델(Mealy Model)

- 상태 머신(State Machine)이 현재 상태와 입력에 영향을 받음



- 초기 상태에서 클럭의 에지에서 i가 '0'이면 S0상태 유지, '1'이면 S1으로 천이
- 출력 y는 S0에서 S1으로 천이될 때 입력 n을 출력, S1에서 S0로 천이될 때 입력 m을 출력

Verilog를 이용한 간단한 상태도 설계

무어 모델의 Verilog 설계

```
1   module SimpleStateMachine(clk, rst, i, m, n, y);
2     input      clk, rst, i, m, n;
3     output     y;
4
5     reg        y;
6     reg        state;
7     reg        next_state;
8
9     parameter S0=1'b0, S1=1'b1;
10
11    always @(posedge clk)
12      if (~rst)    state = S0;                    // 리셋
13      else         state = next_state;
14
15    always @(i or state)
16      case(state)                                 // 상태 천이
17          S0:
18              if(i) next_state = S1;
19              else  next_state = S0;
20          S1:
21              if(i) next_state = S0;
22              else  next_state = S1;
23      endcase
```

Verilog를 이용한 간단한 상태도 설계

```
25   always @(state or m or n)
26     case (state)
27         S0: y = m;
28         S1: y = n;
29     endcase
30 endmodule
```

상태 정의

- 상태를 정의하려면 FSM에서 천이할 수 있는 상태를 정의
- 2개의 상태를 parameter를 사용하여 정의

```
parameter S0=1'b0, S1=1'b1;
```

- 두 상태이므로 한 비트가 할당되어 '0'일 때에는 S0, '1'일 때에는 S1이 각각 할당

Verilog를 이용한 간단한 상태도 설계

상태 천이 표현

```
always @(clock edge)                          // 클럭 에지가 발생했을 때
if (reset = '1')                              // 리셋 버튼이 눌릴 때
  <state_name> = <initial state>;
else
  <state_name> = <next_state>;
always @(input or <state_name>)
case <state_name>                             // 상태 및 입력에 따른 동작 설정
  <state0> :
      if (input =  …) begin
          <next_state> = <state_x>;
              …
              …
          end
      else if (input =  …) begin   <next_state> = <state_y>;
              …
              …
              else …
                   …
                   …
```

Verilog를 이용한 간단한 상태도 설계

```
    <state1> :
            …
        <next_state> = <state_z>;
            …
endcase;
```

- 무어 모델에 대한 일반적인 형식
- 리셋에 의해 초기 상태로 되고, 클럭의 에지에 의해 동기가 되어 다음 상태로 천이
- always 블록에서 다음 상태를 결정
- case~endcase 에 의한 다음 상태 결정

S0 상태에서 입력 i가 '1'이면 다음 상태는 S1, 입력 i가 '0'이면 다음 상태가 S0를 유지

```
case(state)
  S0:                                   // 상태가 S0일 때
      if(i)    state = S1;              // i가 1이면 다음 상태로 천이
      else     state = S0;
  S1:                                   // 상태가 S1일 때
      if(i)    state = S0;              // i가 1이면 다음 상태로 천이
      else     state = S1;
endcase
```

Verilog를 이용한 간단한 상태도 설계

출력 값 할당

- 입력과 상관없이 출력이 현재 상태에 의해 결정 ➔ 무어 모델
- 밀리 모델에 의한 출력 값 할당 : 출력 부분을 상태 천이 case 문에 포함

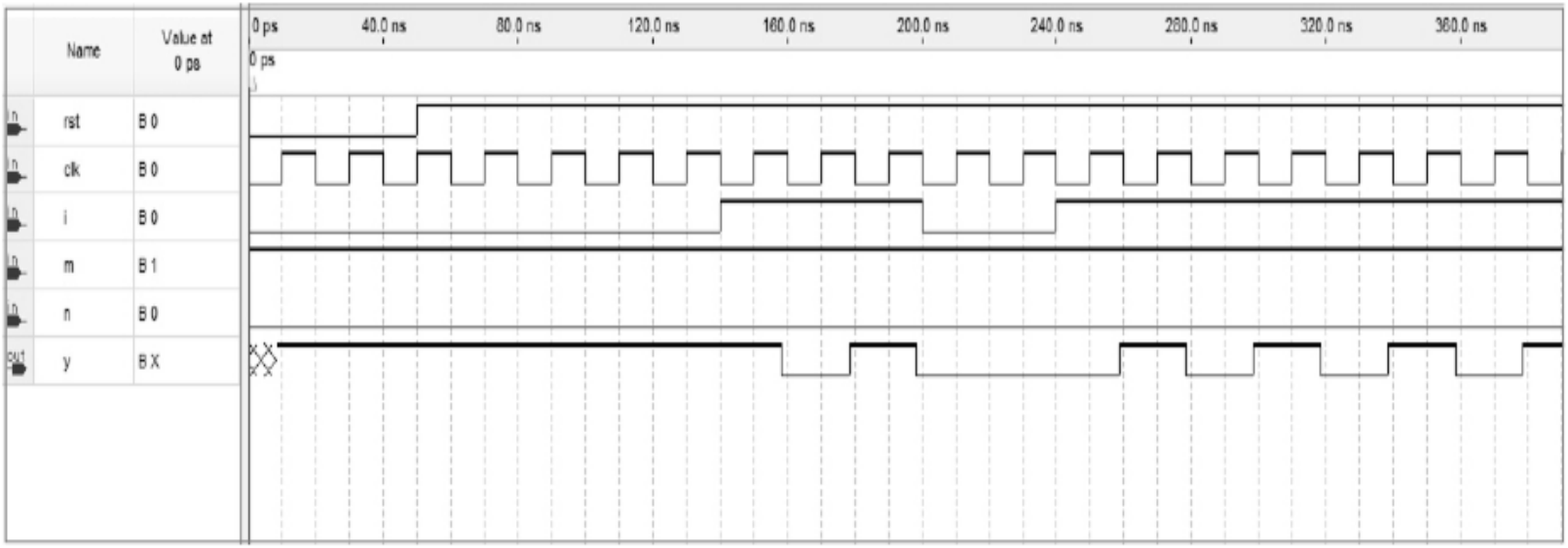
```
1   always @(i or state or m or n)
2     case(state)
3       S0:
4           if(i) begin
5               next_state = S1; y = n;
6           end
7           else  begin
8               next_state = S0; y = m;
9           end
10      S1:
11          if(i) begin
12              next_state = S0; y = m;
13          end
14          else begin
15              next_state = S1; y = n;
16          end
17    endcase
```

S0일 때 입력 n 출력

S1일 때 입력 n 출력

시뮬레이션 및 실행

시뮬레이션 및 실행



- 상태 S0에서는 출력 y는 입력 m 값을 출력, S1에서는 n 값을 출력
- 클럭 : 푸시버튼 스위치를, 출력 y : LED, 입력 i, m, n : 슬라이드 스위치를 할당 실행

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

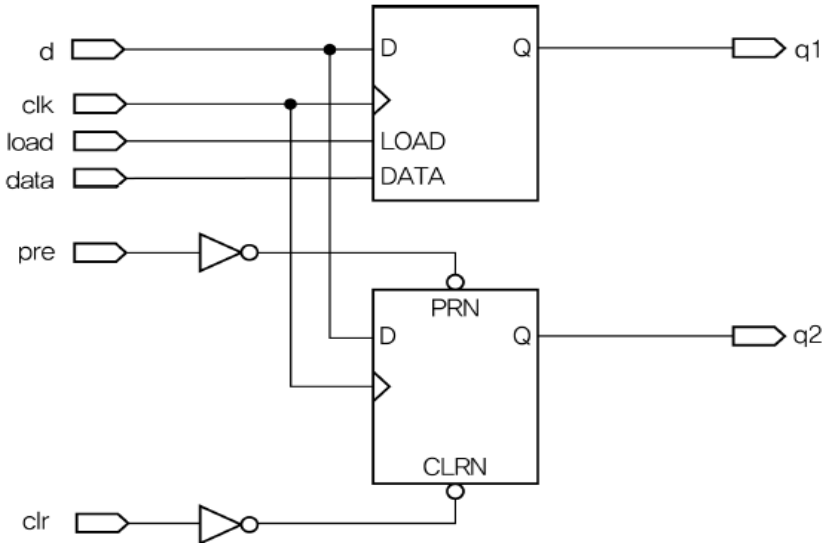
2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

레지스터

레지스터

- 모든 순차논리회로는 하나 이상의 레지스터를 포함
- 레지스터는 클럭에 의해 값이 변하므로 순차논리회로의 하나



- 비동기 로드(load) 레지스터 : load 신호가 클럭에 영향을 받지 않음(첫 번째)
- 비동기 클리어, 프리셋 레지스터 : 클리어, 프리셋 신호가 클럭에 영향을 받지 않음(두 번째)
- Verilog HDL의 인퍼런스(Inference)에 의해 동작을 표현

Verilog를 이용한 레지스터의 표현

레지스터의 Verilog 표현

```
1  module RegisterInference(clk, clr, pre, load, data, d, q1, q2);
2    input    clk, clr, pre, load, data, d;
3    output   q1, q2;
4    reg      q1, q2;
5
6    always @(posedge clk or posedge load)                    // 첫 번째 레지스터
7      if (load)
8          q1 <= data;
9      else
10         q1 <= d;
11
12   always @(posedge clk or negedge clr or negedge pre)      // 두 번째 레지스터
13     if (!clr)
14         q2 <= 1'b0;
15     else if (!pre)
16         q2 <= 1'b1;
17     else
18         q2 <= d;
19 endmodule
```

- 하나의 레지스터는 하나의 always 블록에서 표현
- 첫 번째 always 블록 : 첫 번째 비동기 load 레지스터를 표현
- 두 번째 always 블록 : 두 번째 비동기 클리어, 프리셋 레지스터를 표현

디바이스 핀 할당 및 실행

신호 이름과 할당된 입출력 장치

신호 이름	입/출력	장치 종류	핀 번호(장치 번호)	
			DIGCOM-X1.3	DIGCOM-A1.2
clk	입력	클럭(1MHz)	E16(clock)	P84(clock)
load	입력	슬라이드 스위치	K9(SW2)	P117(SW2)
data	입력	슬라이드 스위치	T9(SW1)	P118(SW1)
d	입력	슬라이드 스위치	R9(SW0)	P119(SW0)
q1	출력	LED	F11(D8)	P105(D8)
pre	입력	푸시버튼 스위치	P8(FGO)	P82(FGO)
clr	입력	푸시버튼 스위치	L8(RESET)	P133(RESET)
q2	출력	LED	A15(D9)	P104(D9)

- 비동기 load 테스트 : load에 '1'을 입력, data 스위치에 '0'과 '1'을 입력하고 q1에 출력이 되는지 확인
- pre(프리셋) 테스트 : 입력 d에 '0'을 입력하고 pre 입력 스위치를 누르면 비동기 프리셋이므로 q2에 '1'이 출력됨
- clr(클리어) 테스트 : 입력 d에 '1'을 입력하고 clr 입력 스위치를 누르면 비동기 클리어이므로 q2에 '0'이 출력됨

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

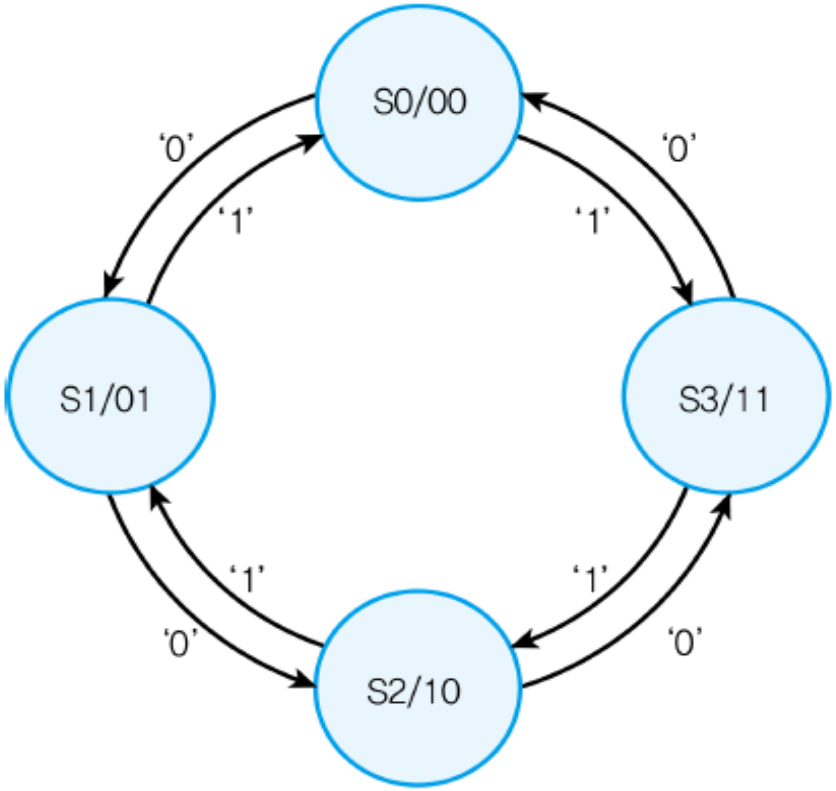
2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

Up-down 카운터의 동작

modulo-4 up-down 카운터

- 입력 x가 '0'이면 0→1→2→3→0 up 카운터 동작
- 입력 x가 '1'이면 3→2→1→0→3 down 카운터 동작
- 카운트 동작의 상태도



Verilog를 이용한 Up-down 카운터 설계

modulo-4 up-down 카운터의 Verilog 설계

- 입력 x가 '0'이면 0→1→2→3→0 up 카운터 동작
- 입력 x가 '1'이면 3→2→1→0→3 down 카운터 동작
- 카운트 동작의 Verilog 표현

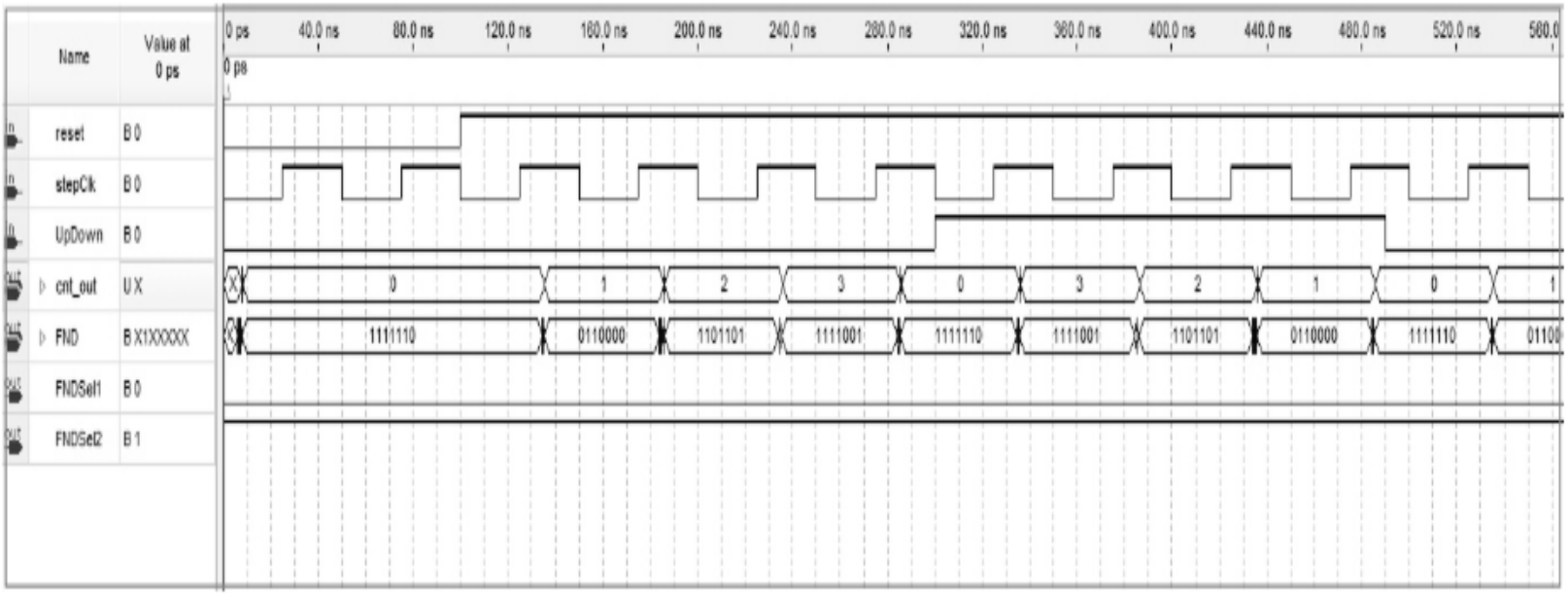
```
1   module UpDownCounter(reset, stepClk, UpDown, cnt_out, FND, FNDSel2, FNDSel1);
2     input          reset, stepClk, UpDown;
3     output [1:0]   cnt_out;
4     output [6:0]   FND;
5     output         FNDSel2, FNDSel1;
6
7     reg [6:0]      FND;
8     wire           FNDSel2=1'b1, FNDSel1=1'b0;
9
10    parameter      S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
11    reg [1:0]      state=S0;
12
13    always @(posedge stepClk or negedge reset) begin
14      if (~reset)                                    // 리셋
15          state <= S0;
```

Verilog를 이용한 Up-down 카운터 설계

```
16      else
17          case(state)                                  // 상태 천이
18              S0:
19                  if(~UpDown)  state <= S1;
20                  else         state <= S3;
21              S1:
22                  if(~UpDown)  state <= S2;
23                  else         state <= S0;
24              S2:
25                  if(~UpDown)  state <= S3;
26                  else         state <= S1;
27              S3:
28                  if(~UpDown)  state <= S0;
29                  else         state <= S2;
30          endcase
31    end
32
33    always @(state)                                    // 7-세그먼트에 결과 출력
34    begin
35      case(state)
36          S0 : FND = 7'b1111110;
37          S1 : FND = 7'b0110000;
38          S2 : FND = 7'b1101101;
39          S3 : FND = 7'b1111001;
40      endcase
41    end
42    assign cnt_out = state;
43  endmodule
```

시뮬레이션 및 실행

시뮬레이션 결과



- 클럭을 푸시버튼 스위치에 할당하고 출력을 2개의 LED 또는 FND에 할당하여 카운트 동작을 확인

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

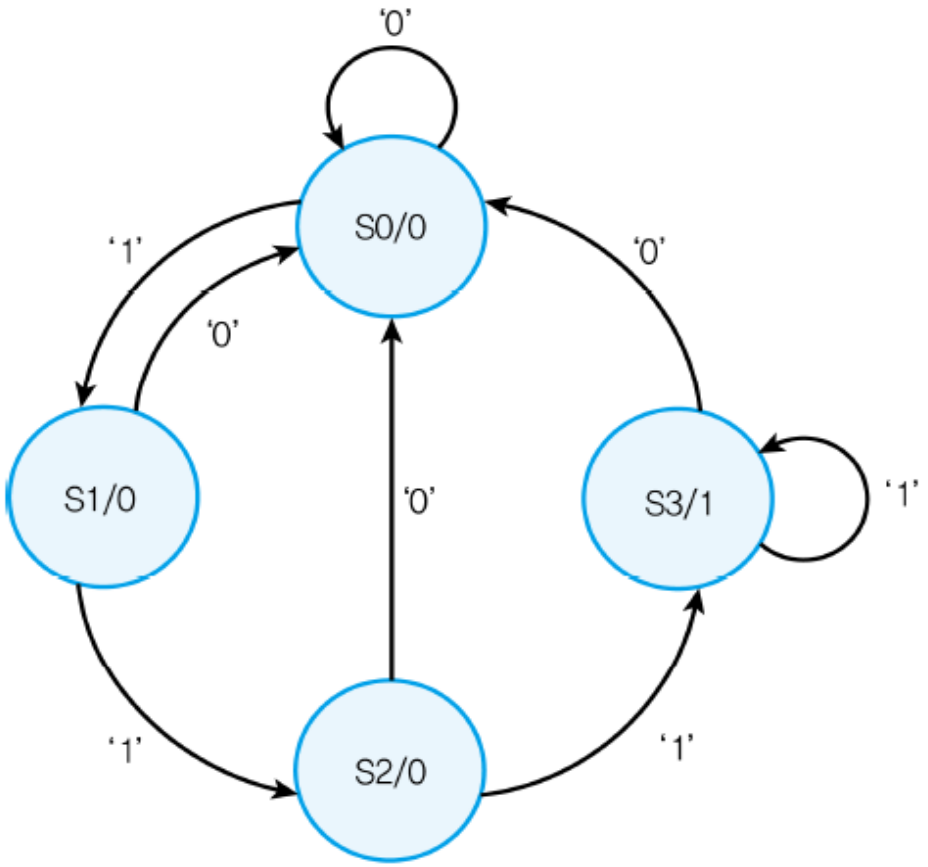
2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

순차 검출기의 동작

순차 검출기

- '1'이 연속해서 세 번 입력되면 '1'을 출력
- 순차 검출기의 상태도



Verilog를 이용한 순차 검출기 설계

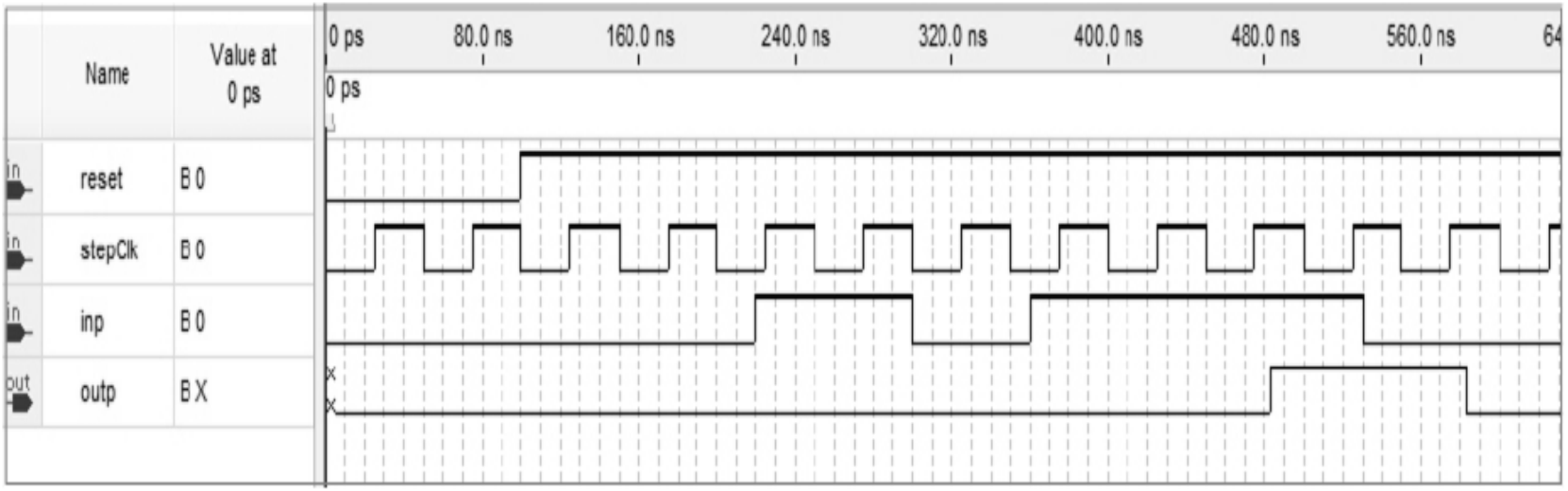
순차 검출기의 Verilog 설계

```
1  module SequenceDetector(stepClk, reset, inp, outp);
2    input     stepClk, reset, inp;
3    output    outp;
4    wire      outp;
5    reg [1:0] state;
6    parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
7
8    always @(posedge stepClk or negedge reset)
9      if (~reset)                          // 리셋
10         state <= S0;
11     else
12         case (state)                     // 상태 천이
13             S0: if (inp) state <= S1;
14                 else state <= S0;
15             S1: if (inp) state <= S2;
16                 else state <= S0;
17             S2: if (inp) state <= S3;
18                 else state <= S0;
19             S3: if (inp) state <= S3;
20                 else state <= S0;
21         endcase
22     assign outp = (state == S3);
23 endmodule
```

- S0, S1, S2, S3 : 각각 '1'이 입력 안된 상태, 한 번, 두 번, 세 번 상태를 나타냄
- 기본 형식이 Up-down 카운터와 동일, 그러나 입력 조건에 따라 다른 천이

시뮬레이션 및 실행

시뮬레이션 결과



- 클럭의 상승에지에서 입력 inp가 '1'을 세 번 유지하면 출력 outp에 '1' 출력
- 클럭 입력에 푸시버튼 스위치를 할당하여 실행하여 결과를 확인

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

클럭을 사용하는/사용하지 않는 회로

동기식 회로의 사용

- 비동기 조합 논리 : 게이트의 지연시간 차에 의해 글리치가 발생
- 동기 조합논리 : 입력 클럭에 의해 출력이 발생하므로 필요 없는 글리치 제거

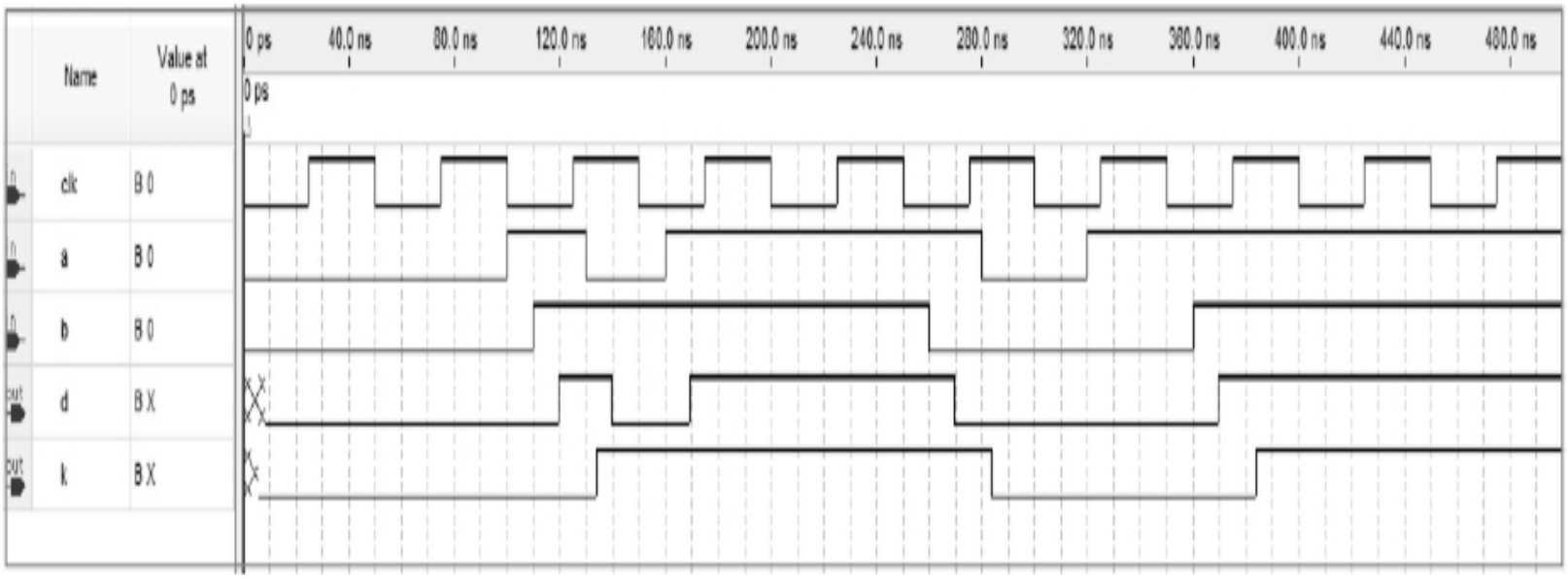
```
1   module ClockSync(clk, a, b, d, k);
2     input      clk;
3     input      a, b;
4     output     d, k;
5
6     wire       d;
7     reg        k;
8
9     assign     d = a & b;                  // 클럭에 동기하지 않는 출력
10
11    always @ (posedge clk)
12    begin
13      k <= a & b;                          // 클럭의 상승 에지에 동기한 출력
14    end
15  endmodule
```

- 출력 d와 k는 같은 논리식을 갖지만 출력 d는 비동기로 출력되지만, k는 클럭의 상승에지에 동기되어 출력

시뮬레이션

시뮬레이션 결과

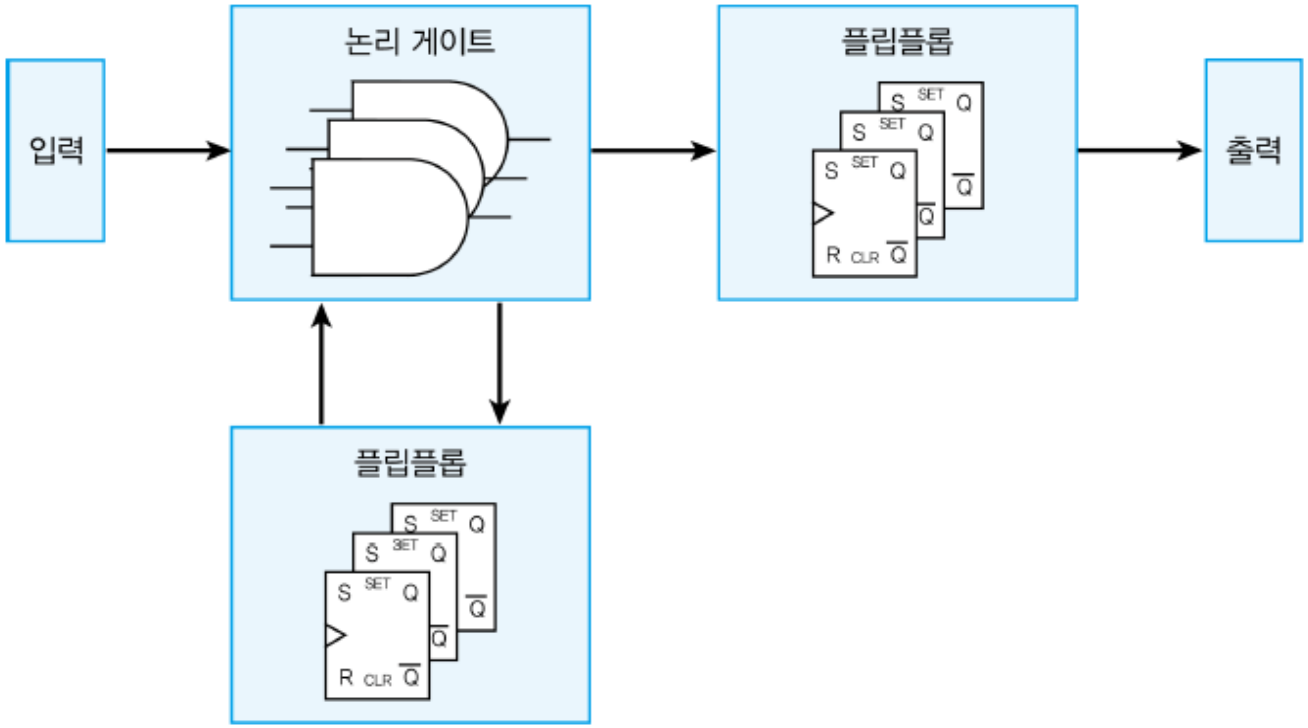
- 동기 출력과 비동기 출력 시간 차이가 짧으므로 시뮬레이션으로 확인



- 비 동기 출력 d는 135ns~165ns에 원하지 않는 신호가 일시적으로 발생
- 동기 출력 k는 131ns 지점에서 클럭에 동기되어 '1' 값을 유지

플립플롭에 의한 동기 출력

- 밀리 모델의 동기식 출력을 위해 출력 값이 플립플롭에 저장되도록 설계



1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

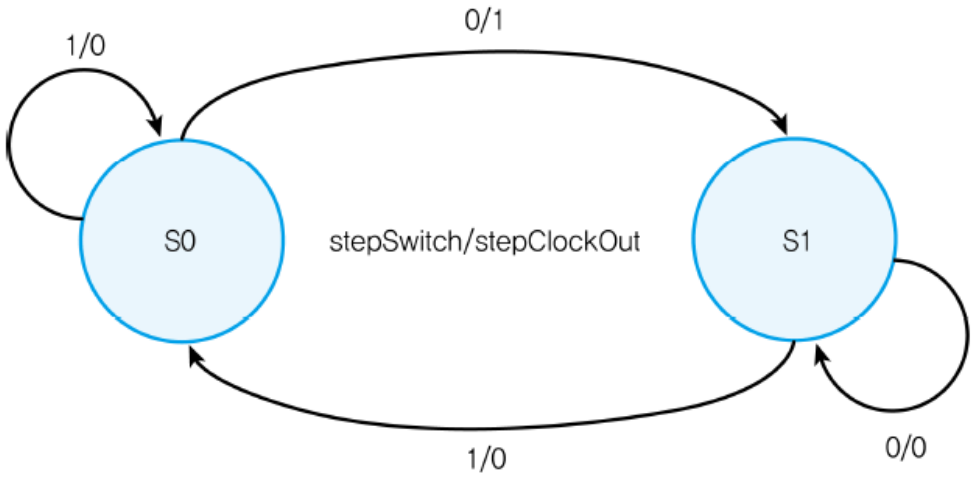
2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

스텝 클럭의 발생

스텝 클럭

- 스위치를 눌렀을 때 눌리는 시간에 상관없이 하나의 클럭을 발생
- 스위치 입력을 스텝 클럭 또는 펄스로 사용할 때 디지털 회로의 처리 속도가 스위치를 누르는 시간보다 상대적으로 매우 짧음



- 초기 상태에서 스위치를 누르면 상승 클럭에서 S1으로 천이, '1'을 출력
- S1상태에서 다음 클럭에서 '0'을 출력 ➔ 한 클럭펄스만큼의 '1'을 발생
- 스위치를 떼면 다시 초기상태 S0으로 리턴

Verilog를 이용한 스텝 클럭 발생회로 설계

스텝 클럭(펄스) 회로의 Verilog 설계

```
1  module StepClock(rst, clk, stepSwitch, stepPulse);
2    input      rst, clk, stepSwitch;
3    output     stepPulse;
4
5    reg        stepPulse;
6    reg        state;
7    reg        clk1Hz;
8    integer    m=0;
9
10   parameter  S0=1'b0, S1=1'b1;
11
12   always @ (posedge (clk)) begin                    // 스텝 클럭 발생
13     if (m >= 499999) begin
14         m <= 0;
15         clk1Hz <= ~clk1Hz;
16     end else
17         m <= m + 1;
18   end
19
20   always @(posedge clk1Hz or negedge rst)
21   begin
22     if(~rst)
23         state <= S0;                                  // 리셋
```

Verilog를 이용한 스텝 클럭 발생회로 설계

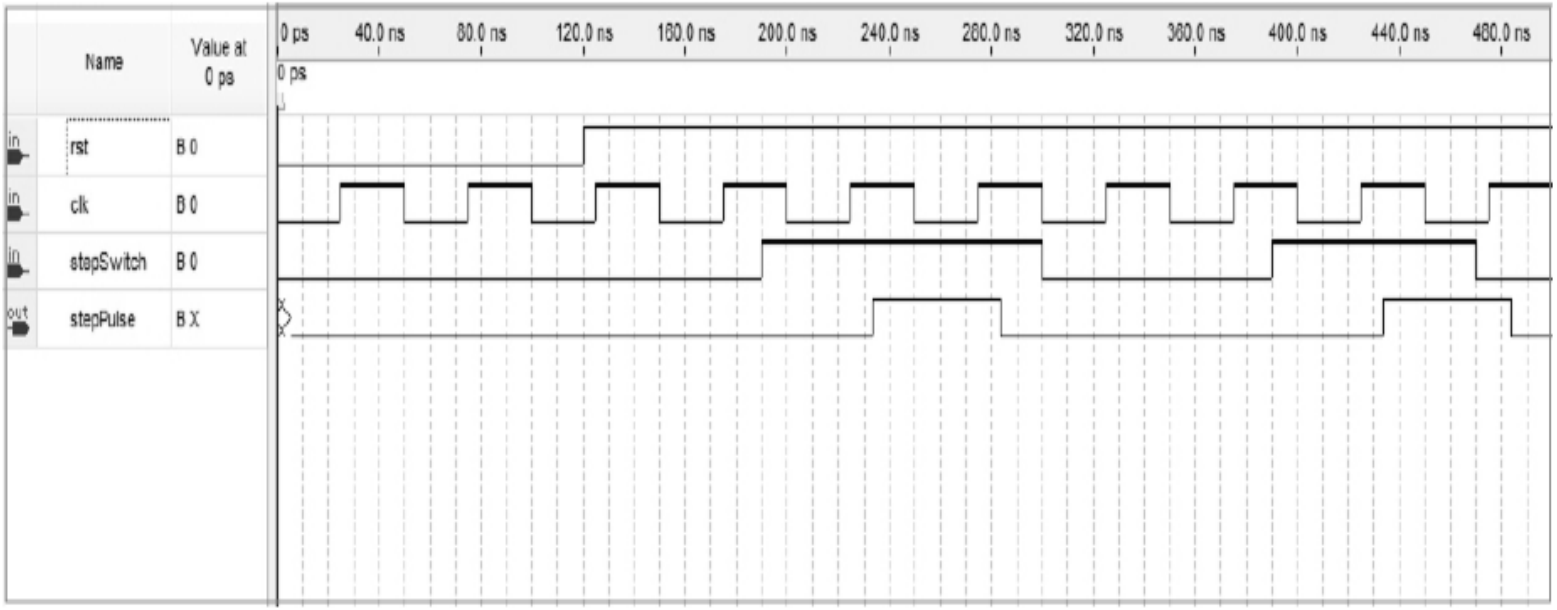
```
24      else
25          case(state)                                    // 스텝 펄스 변경
26              S0:
27                  if(stepSwitch) begin
28                      state <= S1;
29                      stepPulse <= 1'b1;
30                  end else state <= S0;
31              S1:
32                  if(stepSwitch) begin
33                      state <= S1;
34                      stepPulse <= 1'b0;
35                  end else begin
36                      state <= S0;
37                      stepPulse <= 1'b0;
38                  end
39          endcase
40    end
41  endmodule
```

- 1Hz 클럭을 사용하여 1초 동안 ON 상태를 유지
- stepSwitch를 누르면 1초 동안 LED가 켜짐

시뮬레이션 및 실행

시뮬레이션

- 1Hz 클럭을 위해서 시뮬레이션에서 1,000,000분주를 하기위해서는 긴 시간 필요하므로 분주 수를 줄여서 시뮬레이션 실행



- 190ns에서 약 2클럭동안 '1' 상태를 유지 했지만 한 클럭 기간동안 '1' 출력

1. 순차논리회로 설계

1) 간단한 상태도의 구현

2) 레지스터의 구현

3) Up-down 카운터

4) 순차 검출기

2. 기타 논리회로 설계

1) 클럭을 사용하는 회로와 사용하지 않는 회로

2) 스텝 클럭(펄스) 발생회로

3) 양방향 버스

Verilog를 이용한 양방향 버스 설계

양방향 버스 : 메모리에서 데이터 버스 또는 양방향 포트에서 입출력으로 데이터 전송

```
1  module BidirBus (oe, clk, inp, outp, bidir);
2    input   oe;
3    input   clk;
4    input   [7:0] inp;
5    output  [7:0] outp;
6    inout   [7:0] bidir;                  // 양방향 신호
7
8    reg     [7:0] a;
9    reg     [7:0] b;
10
11   assign bidir = oe ? a : 8'bZ;         // 제어신호
12   assign outp  = oe ? 8'bz : b;
13
14   always @ (posedge clk)
15   begin
16     b <= bidir;
17     a <= inp;                           // 입력 값 임시 저장
18   end
19 endmodule
```

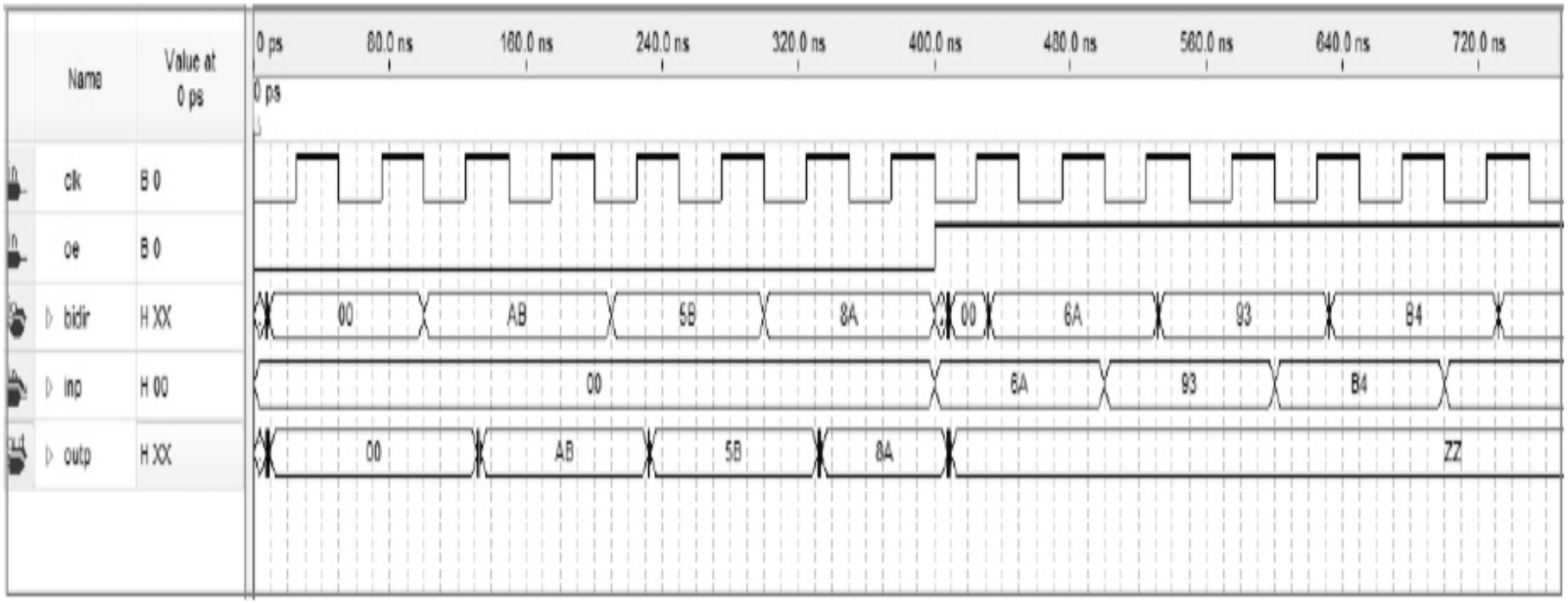
Verilog를 이용한 양방향 버스 설계

- bidir : 양방향(inout)으로 선언, 제어신호(oe)가 '1'➔출력 동작, '0'➔ 입력동작
- inp, outp를 선언, register a에 inp 입력 신호 값을 항상 저장, oe이 '1'이면 a에 저장된 값을 bidir에 할당하므로 출력 포트로 동작
- oe이 '0'이면 bidir에 'z'를 출력 high impedence 상태를 갖게하여 입력을 받음
- bidir이 입력으로 동작할 때 입력 값을 위해 register b를 bidir에 입력된 값을 저장하고 이 값을 출력포트 outp에 출력하면 bidir에 입력된 값이 outp로 출력

시뮬레이션

시뮬레이션

- 양방향 포트로 사용할 수 있는 입출력 장치가 없으므로 시뮬레이션으로 확인



- 양방향 버스(bidir) : 100ns~400ns ➔ 입력, 그 이후 ➔ 출력으로 동작

Q & A

수고하셨습니다.