




Lab03

One-Counter design



휴먼지능정보공학과
201810756
김대환

Lab 3 _ Instruction

- ▶ Design an “one-counter” with the state machine design technique.
 - ▶ Inputs:
 - ▶ A: 8 bit data
 - ▶ Load_A (LA)
 - ▶ S: Start counting indicator
 - ▶ Outputs:
 - ▶ B: 4 bit data
 - ▶ Done
 - ▶ Function:
 - ▶ If ‘Load’ is ‘true’ then receive 8 bit data input A
 - ▶ If S is ‘1’ then count ‘1’s in A’s bitstream, and output the number to B, and set Done to ‘1’. (example: if A is 11010010 then B=4.)
- ▶ Refer Pseudo-code, timing diagram, & Shift register code given.



Lab 3 _ 접근

▶ 무엇을 설계하는가?

▶ Input :

- A : 전송하고자 하는 data
- Load : data를 옮기는 signal
- S : Right Shift 시작하는 signal

▶ Output:

- B : data A의 bitstream LSB '1' count 결과값 저장하는 data
- done : 원하는 결과값(B)를 얻었음을 알려주는 signal

▶ 기대하는 결과 :

- ▶ Load=1 -> data A 로드, start=1 -> clk의 rising edge에 A bitstream right shift
- ▶ 8 bits data A의 bitstream LSB(0)을 확인하여 '1'인 경우 4 bits data B++
- ▶ 최종적으로 B 에는 초기 data A의 '1'의 개수가 4 bits 형태로 저장



Lab 3 _ 설계

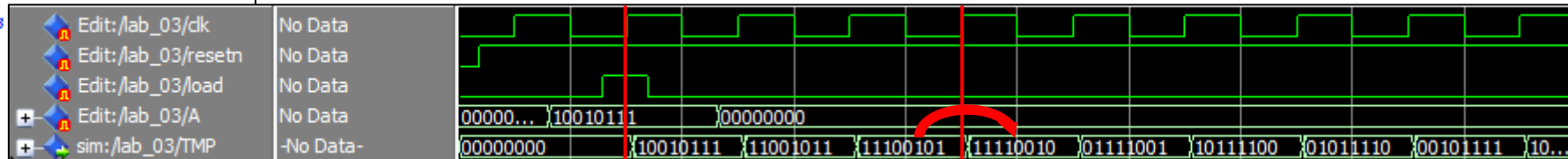
▶ [Right shift register]

- ▶ 우선 단순히 load 기능만 구현하여 data A를 메모리에서 옮겨 담고 bitstream 단위로 Right shift 해주는 것을 구현해보았다.

VHDL Code

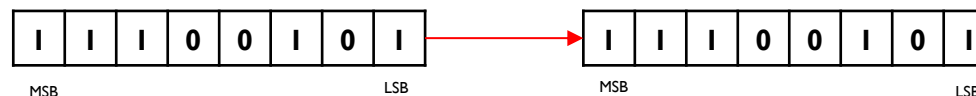
```
entity lab_03 is
  Port ( clk, resetn : in STD_LOGIC;
        load : in STD_LOGIC;
        A : in STD_LOGIC_VECTOR (7 downto 0);
        TMP : BUFFER STD_LOGIC_VECTOR(7 downto 0));
end lab_03;
architecture Behavioral of lab_03 is
begin
  process (clk,resetn,load)
  begin
    if(resetn='0') THEN
      TMP<="00000000";
    Elsif(clk'event AND clk='1') THEN
      if(load='1') THEN
        TMP<=A;
      else
        TMP<=TMP(0) & TMP(7 downto 1);
      end if;
    end if;
  end process;
end Behavioral;
```

Modelsim simulation result



• Clk rising edge에 load=1
=> Data load

• Clk rising edge에 따라
Data의 bitstream 단위로 right shift 진행

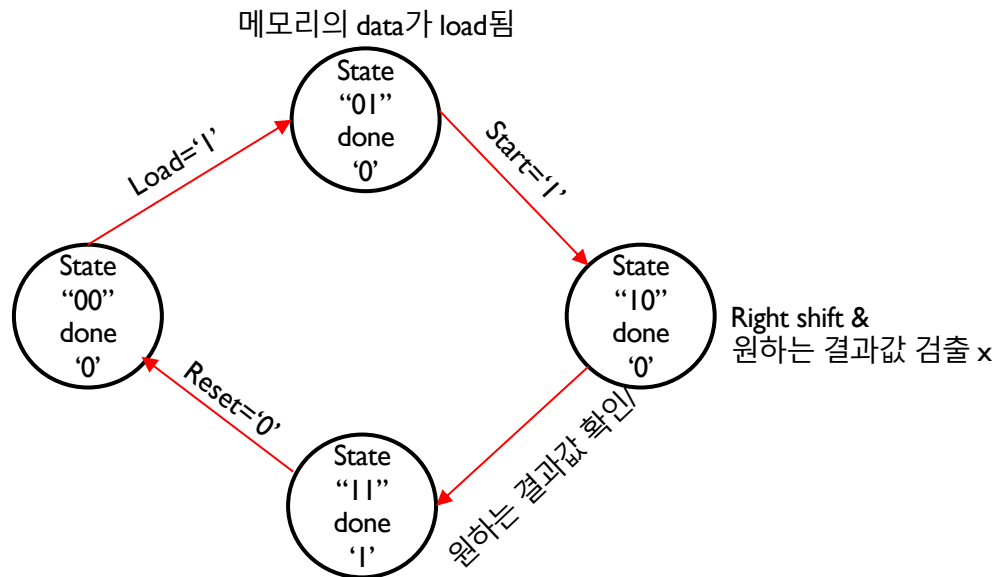


Lab 3 _ 설계

▶ [Start, Done 구현]

- ▶ 'start' signal은 data 의 right shift를 시작하는 signal의 역할을 한다.
- ▶ 'done' signal은 원하는 결과값을 얻은 상태를 보여주는 signal의 역할을 한다.
- ▶ 이를 구현하기 위해 현재의 상태를 표현하는 state도 함께 구현하였다.

□ State : 2bits (00,01,10,11) by Moore machine(?)



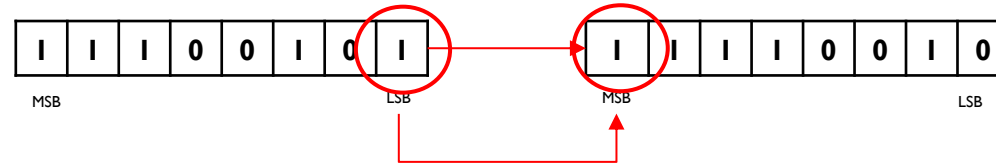
Lab 3 _ 설계

▶ [Start, Done 구현]

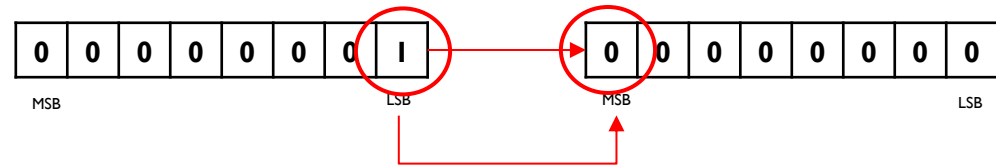
- ▶ 'done' signal / 어떻게 원하는 결과값이 검출되었음을 인지할 수 있을까?

□ 데이터를 right shift 할 때 기존의 코드에서는 다음과 같이 `TMP<=TMP(0) & TMP(7 downto 1);`

Bitstream의 LSB를 다음 데이터의 MSB로 옮겨줌으로서 right shift 기능을 수행하였다.



기능적 목표가 '1'을 검출하는 것이므로 확인된 LSB에 대해서는 '0'으로 바꾸어 MSB로 옮겨준다면 모든 '1'이 검출된 후에는 bitstream내에는 0의 값만 즉, data는 '0'이 되었을 것이다.



모든 '1' 검출 후 data = "00000000" -> value = '0'

LSB '1' 확인 후 '0'으로 바꾸어서 다음 data MSB로 전달

따라서, data가 '0'의 값을 가지는 그 순간 원하는 결과값을 얻었다고 판단하면 된다.

Lab 3 _ 참고

▶ 강의노트7_Complex Sequential Logic design p.44

Synthesizable VHDL Code

```
proc1: process (reset_n, clk) is begin
  if (reset_n = '0') then
    req <= '0';
    state <= "00"; -- state should be a 2-bit unsigned signal
  elsif rising_edge(clk) then -- on posedge clk
    state <= state + 1; -- auto-increment current state
    case (state) is
      when "00" => if (ready = '0') then
        state <= "00"; -- repeat until ready
      end if;
      when "01" => data <= data_in; -- assume data_in is data to be sent
      when "10" => req <= not req; -- send request signal
      when others => if (ack /= req) then -- case "11"
        state <= "11"; -- stay in this state until (ack = req)
      end if;
    end case;
  end if; -- of elsif rising_edge(clk)
end process proc1;
```

- 이 참고 코드에서 주의 깊게 봐야할 부분은 State 값을 매 rising edge에 증가시키면서 조건에 따라 state 값을 재조정해주는 것이다.

State 값이 증가하는 순간 값이 바뀌는게 아닌 다음 Rising edge에서 증가하므로 매 순간 증가시키더라도 조건문을 확인할 때에는 기존의 값으로 조건문에 진입하게 된다.

따라서 이 코드에 기반하여 원하는 결과값을 구현해내는 코드를 작성해보자.

Lab 3 _ 구현

▶ [entity 구성]

```
entity lab_03 is
  Port ( clk, resetn : in STD_LOGIC;
        load : in STD_LOGIC;
        start : in STD_LOGIC;
        A : in STD_LOGIC_VECTOR (7 downto 0);
        TMP : BUFFER STD_LOGIC_VECTOR(7 downto 0);
        state : BUFFER STD_LOGIC_VECTOR(1 downto 0);
        B : BUFFER STD_LOGIC_VECTOR(3 downto 0);
        done : OUT STD_LOGIC);
end lab_03;
```

- ☐ clk,resetn,load,start : std_logic input
- ☐ A,TMP,state,B : std_logic_vector buffer
- ☐ done : std_logic output



Lab 3 _ 구현

▶ [Behavioral]

- ▶ resetn : negative reset / '0' 일때 초기화

```
if(resetn='0') THEN
  done<='0';
  state<="00";
  B<="0000";
  TMP<="00000000";
```

- ▶ clk의 rising edge에서 state는 매순간 증가(auto-increment)

```
ELSIF rising_edge(clk) THEN
  state<=state+1;
```

- ▶ state="00" 일 때, 만약 load='0'이면 state="00" 유지

혹은 load='1'이면 data를 메모리에서 Load(state="00"->"01")

```
when "00" => if(load='0') then
  state<="00";
  elsif(load='1') then
    TMP<=A;
  end if;
```



Lab 3 _ 구현

- ▶ state="01" 일 때, start='0' 이면 state 유지, 'start'='1'이면 state="10"으로

```
when "01" => if (start='0') then  
    state<="01";  
end if;
```

- ▶ state="10" 일 때,

- 만약 TMP가 0이 아니라면(아직 bitstream 내 '1' 남아있는 경우) TMP의 LSB를 확인하여 '1'인 경우에는 B ++ 그리고 TMP right shift(LSB '0'으로 전환 후 MSB이동) 그리고 아직 원하는 결과값이 안나타났으므로 state="10" 고정

- TMP가 0이라면(원하는 결과값 추출 완료) done='1' and state 변화

```
when "10" => if (not (TMP=0)) THEN  
    if (TMP(0)='1') THEN  
        B<=B+1;  
    end if;  
    TMP<='0' & TMP(7 downto 1);  
    state<="10";  
else  
    done<='1';  
end if;
```

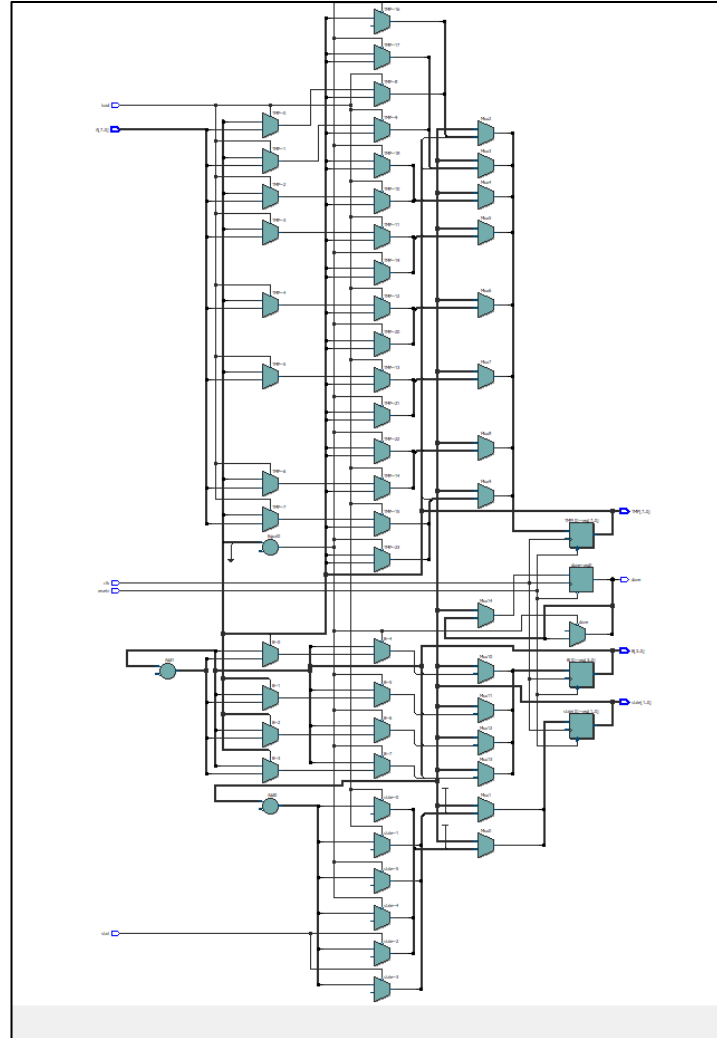
- ▶ state="11" 일 때,

```
when "11" => state<="11";
```

Lab 3 _ 코드

```
entity lab_03 is
  Port ( clk, resetn : in STD_LOGIC;
        load : in STD_LOGIC;
        start : in STD_LOGIC;
        A : in STD_LOGIC_VECTOR (7 downto 0);
        TMP : BUFFER STD_LOGIC_VECTOR(7 downto 0);
        state : BUFFER STD_LOGIC_VECTOR(1 downto 0);
        B : BUFFER STD_LOGIC_VECTOR(3 downto 0);
        done : OUT STD_LOGIC);
end lab_03;
architecture Behavioral of lab_03 is
begin
  process (clk,resetn,load)
  begin
    if(resetn='0') THEN
      done<='0';
      state<="00";
      B<="0000";
      TMP<="00000000";
    ELSIF rising_edge(clk) THEN
      state<=state+1;
      case(state) is
        when "00" => if(load='0')then
                      state<="00";
                    elsif(load='1')then
                      TMP<=A;
                    end if;
        when "01" => if(start='0')then
                      state<="01";
                    end if;
        when "10" => if(not(TMP=0)) THEN
                      if(TMP(0)='1') THEN
                        B<=B+1;
                      end if;
                      TMP<='0'&TMP(7 downto 1);
                      state<="10";
                    else
                      done<='1';
                    end if;
        when "11"=>state<="11";
      end case;
    end if;
  end process;
end Behavioral;
```

Lab 3 _ RTL view



Lab 3 _ Modelsim Simulation

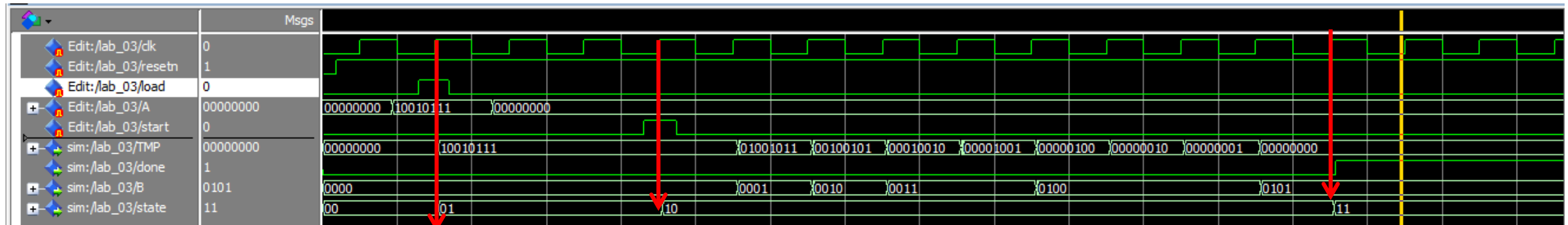
▶ Simulation 세팅 값

- ▶ clk : Clock pattern, End time = 2000ns, Initial Value = 0, Clock Period = 100ns
- ▶ resetn : Constant pattern, End time = 2000ns, Initial Value = 1
- ▶ load, start : Constant pattern, End time = 2000ns, Initial Value = 0
- ▶ A : Constant pattern, End time = 2000ns, Initial Value = 00000000



Lab 3 _ Modelsim Simulation

- ▶ A : “10010111”
- ▶ state 변화 확인



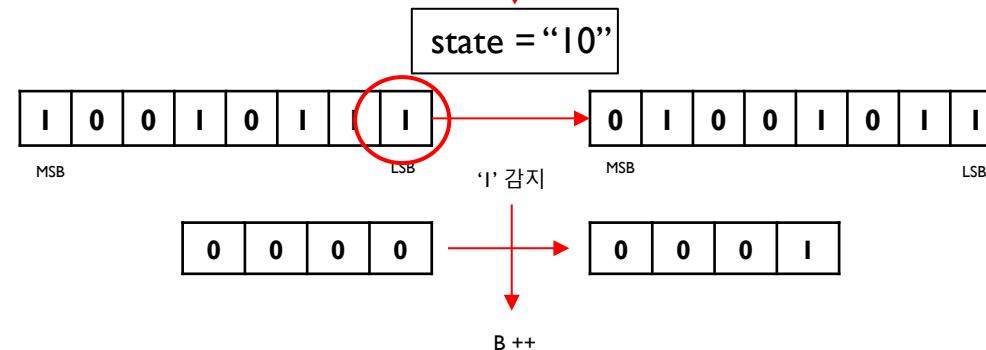
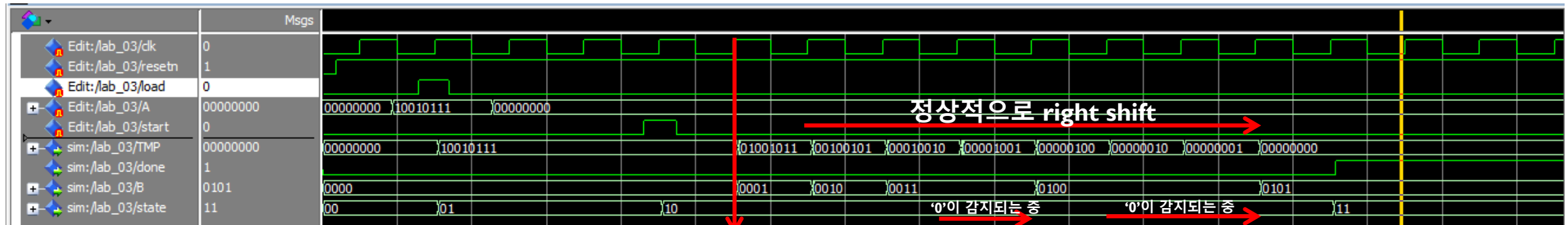
load='1' 감지한 후
state "00" -> "01"

start='1' 감지한 후
state "01" -> "10"

원하는 결과 얻은 후
state "10" -> "11"

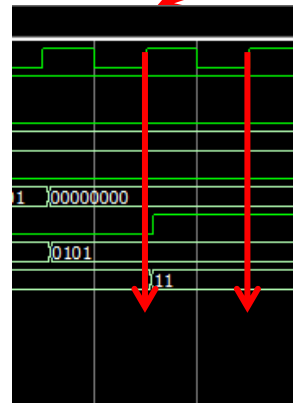
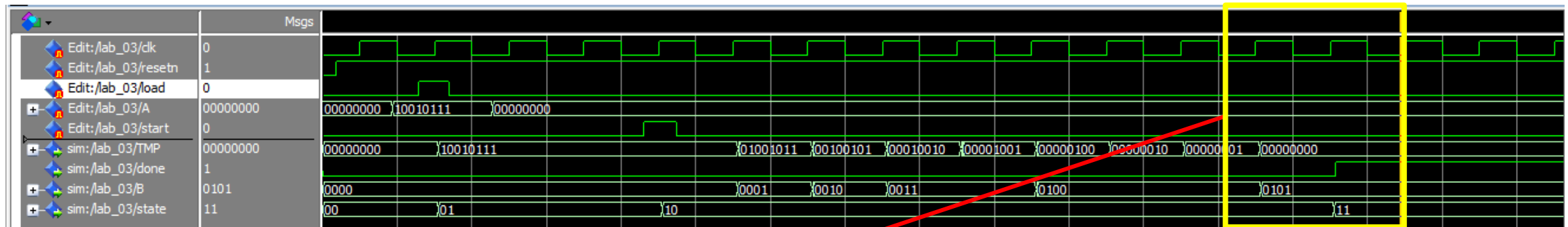
Lab 3 _ Modelsim Simulation

- ▶ A : “10010111”
- ▶ data right shift & ‘1’ counter 확인



Lab 3 _ Modelsim Simulation

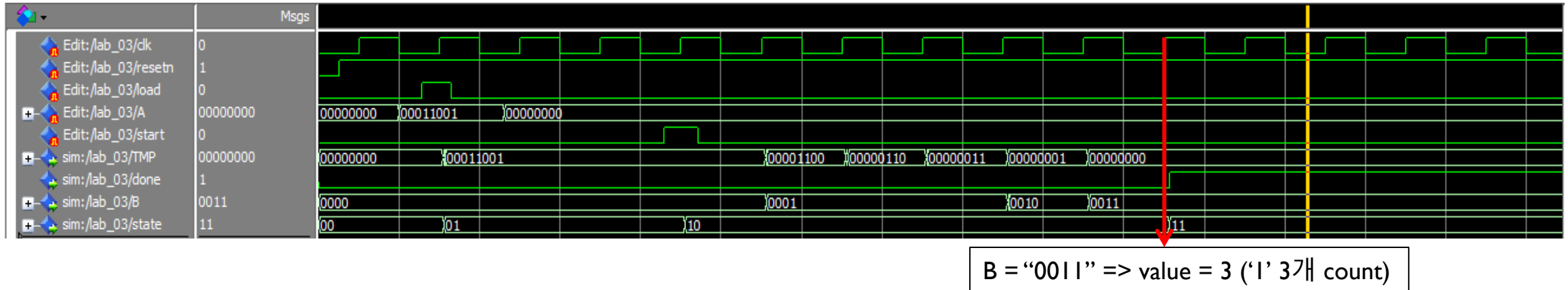
- ▶ A : “10010111”
- ▶ ‘done’ 확인



TMP = “00000000”
즉 data의 value가 0임을 감지한 rising edge 순간에
done=‘1’, state=“11”
따라서 그 다음 rising edge에 이를 인지할 수 있음

Lab 3 _ Modelsim Simulation

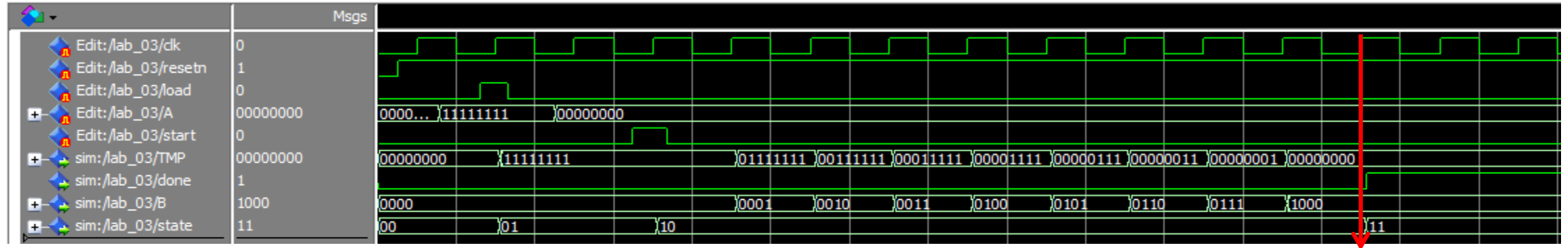
▶ A : “00011001”



- Right shift 할 때 LSB를 ‘0’으로 바꾸고 다음 data의 MSB로 넘겨준 장점이 이 예시에서 나타난다.
=> “00011001” 에서 right shift 하다보면 실제 모든 bitstream의 bit를 검사하는 것이 아닌 필요한 부분까지만 검사하므로 더욱 효율적임을 알 수 있다. 실제로 right shift는 5번만 실행되었음을 알 수 있다.

Lab 3 _ Modelsim Simulation

▶ A: “11111111”



B = “1000” => value = 8 (‘1’ 8개 count)

- 8번의 Right shift 끝에 원하는 결과값을 얻었음을 알 수 있다.

Lab 3 _ Problems met during design & Solutions

- ▶ 간만에 VHDL 코드를 작성하고 모델심 시뮬레이션을 진행하다보니 문법적인 부분과 실습을 진행하는 과정이 조금 어색하였음.
- ▶ 주로 VHDL 에서 실수하였던 문법은 ';'를 빼거나, if – end if 의 짝을 못 맞추었거나 하는 그런 사소한 실수들이었다.
- ▶ 처음에 프로젝트 파일 이름과 코드 내 명시해주는 프로젝트 이름을 달리 작성하여 실행 시 오류를 경험했다. 과거 '컴퓨터로직설계' 에서 이 부분에 대해 설명을 들었던 기억이 나 수정하였다.
- ▶ 참고 코드에서 state 값을 매 순간 증가시키면서 조건문에 따라 state를 결정해주는 과정이 어색하여 코드를 자세하게 읽어보고 이해하는데 시간이 조금 걸렸다. 하지만 해당 코드를 이해해보니 이번 실습을 구현하는데 있어서 가장 중요한 부분을 깨달았고 바로 코드에 적용시킬 수 있었다. RTL은 실행문이 concurrent하다는 것을 인지하는 것이 중요했다. 또한 state의 결정은 맨 마지막 구문이 결정한다는 사실이 참고코드에서 중요하게 작용되었다.
- ▶ 첫 시작이 어려웠지 참고 코드의 이해와 정확한 목표설정과 함께 코드를 작성하기 시작한 후에는 큰 어려움 없이 원하는 실습 결과를 얻을 수 있었다.



Lab 3 _ Discussion

- ▶ State를 설정해주는데 있어서 이것을 Moore Machine이라고 할 수 있는지에 대해서 조금 의문이 들었다.
나름대로 한번 만들어본 Moore machine state table이다.

Present state	Input		Next State	Output (done)
	Load	start		
00	0	-	00	0
00	0	-	00	0
00	1	-	01	0
00	1	-	10	0
01	-	0	01	0
01	-	1	10	0
01	-	0	01	0
01	-	1	10	0
10	-	-	10	0
10	-	-	10	0
10	-	-	10	0
10	-	-	10	0
11	-	-	11	1
11	-	-	11	1
11	-	-	11	1
11	-	-	11	1

- 다음의 state table과 같이 현재 실습 상황에서는 state 중 일부만이 input의 영향을 받는 상황이다. state="00"은 load input에만, state="01"은 start input에만 심지어 state="10","11"은 input의 영향이 아닌 어떠한 조건에 의해 영향을 받는다. Moore machine은 output이 state에 영향을 받고, state는 input에 영향을 받는 것으로 알고 있는데 이러한 형태도 Moore machine이라고 해도 되는지 의문이 든다.

⇒ TMP 의 value에 영향을 받음(if(조건)문에 따른 boolean value)

- TMP value !=0 : state "10"
- TMP value == 0 : state "11"