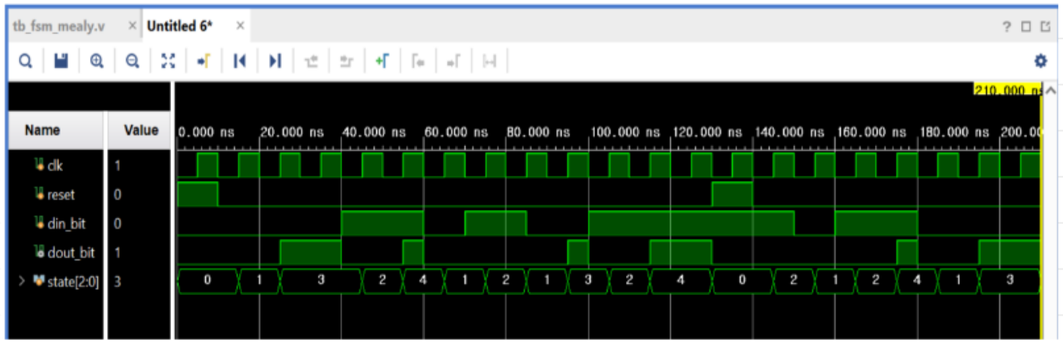
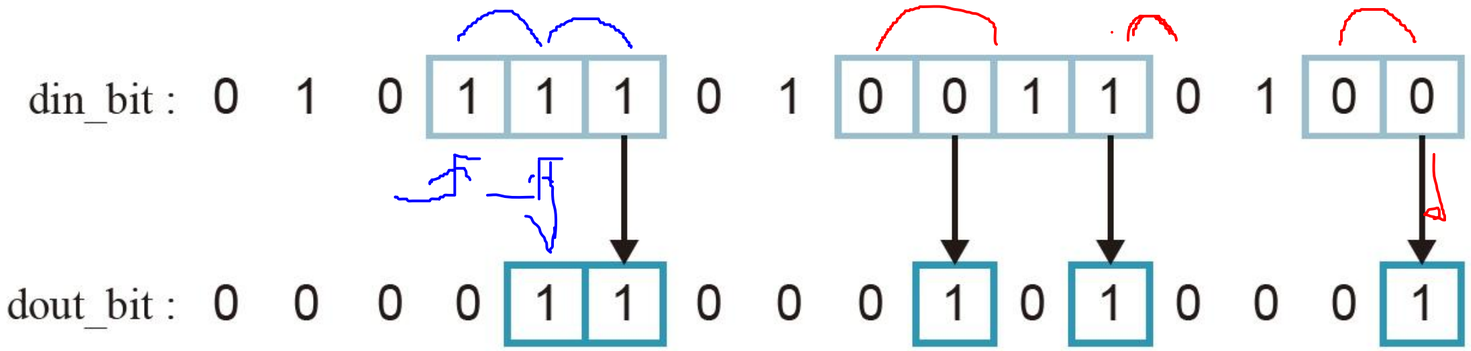
```
25      // 테스트 시나리오
26      initial begin
27          // 초기화
28          reset = 1; din_bit = 0;
29          #10 reset = 0; // 리셋 해제
30
31          // 테스트 케이스: 연속된 입력 신호 검증
32          #10 din_bit = 0;  // 첫 번째 0 입력 -> rd0_once 상태로 전환
33          #10 din_bit = 0;  // 두 번째 0 입력 -> rd0_twice 상태로 전환 (out = 1)
34          #10 din_bit = 1;  // 첫 번째 1 입력 -> rd1_once 상태로 전환 (out = 0)
35          #10 din_bit = 1;  // 두 번째 1 입력 -> rd1_twice 상태로 전환 (out = 1)
36          #10 din_bit = 0;  // 첫 번째 0 입력 -> rd0_once 상태로 전환 (out = 0)
37          #10 din_bit = 1;  // 첫 번째 1 입력 -> rd1_once 상태로 전환 (out = 0)
38          #10 din_bit = 1;  // 두 번째 1 입력 -> rd1_twice 상태로 전환 (out = 1)
39          #5  din_bit = 0;
40          #15  din_bit = 1;
41          #20;
42
43          // 리셋 테스트
44          #10 reset = 1; // 리셋 활성화 -> START 상태로 복귀
45          #10 reset = 0;
46
47          // 추가 시나리오: 혼합된 입력 신호 검증
48          #10 din_bit = 0;
49          #10 din_bit = 1;
50          #10 din_bit = 1;
51          #10 din_bit = 0;
52          #10 din_bit = 0;
53
54          // 시뮬레이션 종료
55          #20 $finish;
56      end
```

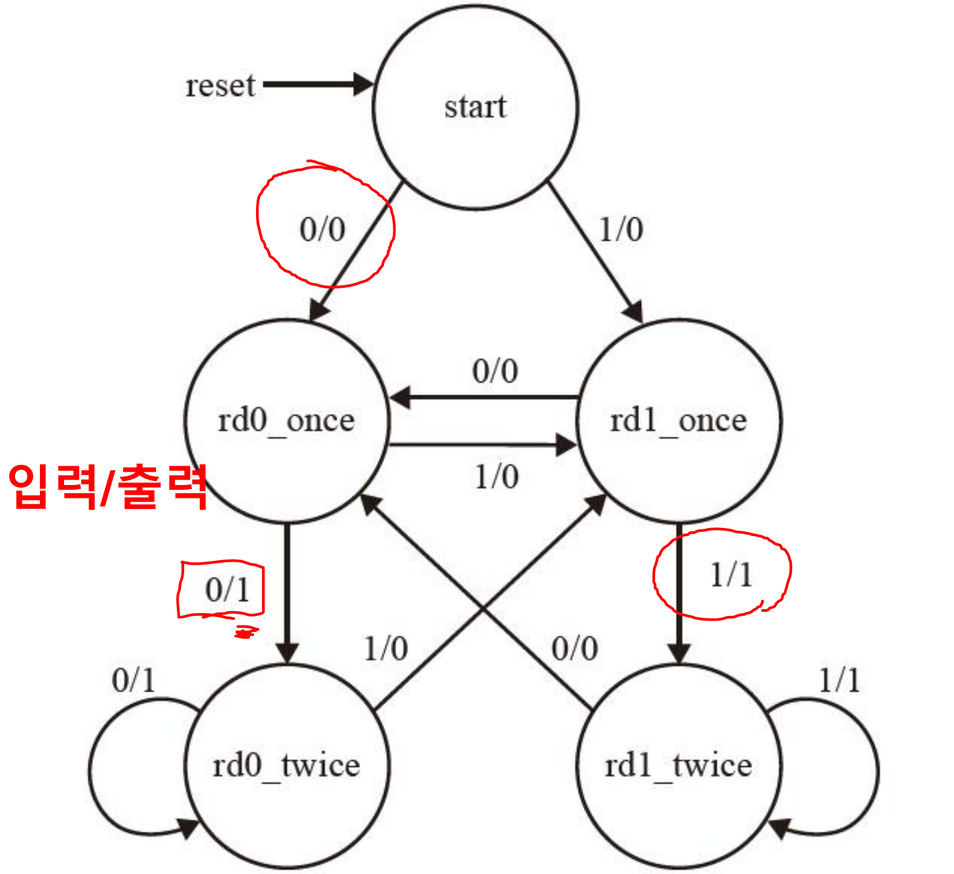


❖ 연속된 0 또는 1 입력을 검출기



[그림 11-32] 연속된 0 또는 1 입력을 검출기의 동작

❖ 연속된 0 또는 1 입력을 검출기



[그림 11-33] 연속된 0 또는 1을 검출하는 Mealy 유한상태머신

11.4.2 Mealy FSM 회로

코드 11-33 [그림 11-33]의 Mealy FSM 모델링

```
module seq_det_mealy (input clk, input rst, input din_bit,
                      output dout_bit);
  reg [2:0] state_reg, next_state;

// 상태 선언
  parameter  start     = 3'b000;
  parameter  rd0_once  = 3'b001;
  parameter  rd1_once  = 3'b010;
  parameter  rd0_twice = 3'b011;
  parameter  rd1_twice = 3'b100;

// 다음 상태 결정을 위한 always 조합회로 블록
always @(state_reg or din_bit) begin
  case(state_reg)
    start    : if     (din_bit == 0)   next_state = rd0_once;
               else if(din_bit == 1)   next_state = rd1_once;
               else                    next_state = start;
   rd0_once  : if(din_bit == 0)        next_state = rd0_twice;
               else  if(din_bit == 1)  next_state = rd1_once;
               else                    next_state = start;
   rd0_twice : if(din_bit == 0)        next_state = rd0_twice;
               else  if(din_bit == 1)  next_state = rd1_once;
               else                    next_state = start;
   rd1_once  : if(din_bit == 0)        next_state = rd0_once;
               else  if(din_bit == 1)  next_state = rd1_twice;
               else                    next_state = start;
   rd1_twice : if(din_bit == 0)        next_state = rd0_once;
               else  if(din_bit == 1)  next_state = rd1_twice;
               else                    next_state = start;
    default :                          next_state = start;
  endcase
```

11.4.2 Mealy FSM 회로

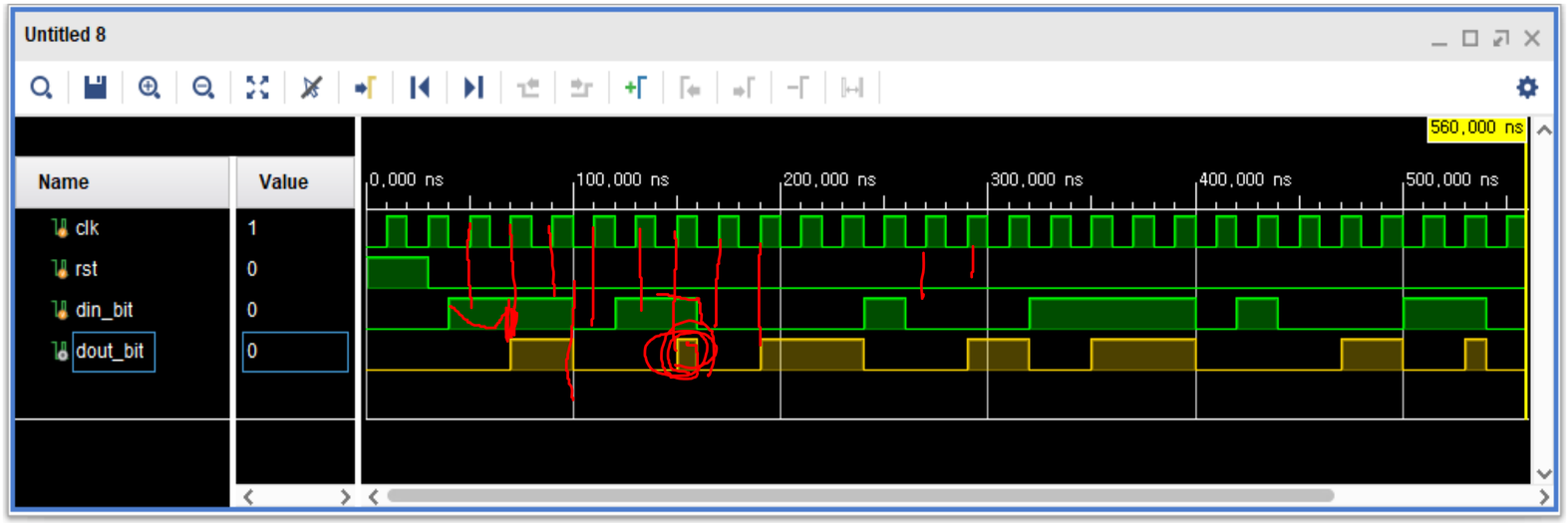
코드 11-33 [그림 11-33]의 Mealy FSM 모델링

```
// 상태 레지스터를 위한 always 순차회로블록
always @(posedge clk or posedge rst) begin
    if(rst == 1) state_reg <= start;
    else         state_reg <= next_state;
end

// 출력값 결정
assign dout_bit =(((state_reg == rd0_twice) && (din_bit == 0) ||
                   (state_reg == rd1_twice) && (din_bit == 1))) ? 1 : 0;

endmodule
```

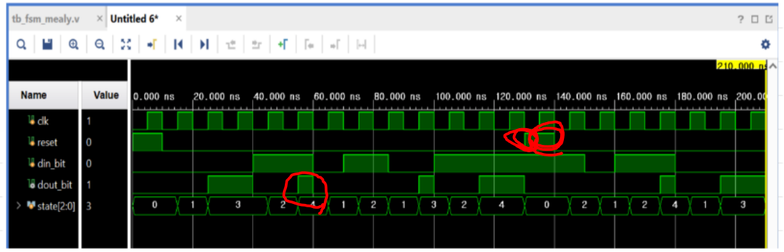
11.4.2 Mealy FSM 회로



[그림 11-34] [코드 11-33]의 시뮬레이션 결과

```
25    // 테스트 시나리오
26    initial begin
27        // 초기화
28        reset = 1; din_bit = 0;
29        #10 reset = 0; // 리셋 해제
30
31        // 테스트 케이스: 연속된 입력 신호 검증
32        #10 din_bit = 0; // 첫 번째 0 입력 -> rd0_once 상태로 전환
33        #10 din_bit = 0; // 두 번째 0 입력 -> rd0_twice 상태로 전환 (out = 1)
34        #10 din_bit = 1;  // 첫 번째 1 입력 -> rd1_once 상태로 전환 (out = 0)
35        #10 din_bit = 1;  // 두 번째 1 입력 -> rd1_twice 상태로 전환 (out = 1)
36        #10 din_bit = 0;  // 첫 번째 0 입력 -> rd0_once 상태로 전환 (out = 0)
37        #10 din_bit = 1;  // 첫 번째 1 입력 -> rd1_once 상태로 전환 (out = 0)
38        #10 din_bit = 1;  // 두 번째 1 입력 -> rd1_twice 상태로 전환 (out = 1)
39        #5  din_bit = 0;
40        #15  din_bit = 1;
41        #20;
42
43        // 리셋 테스트
44        #10 reset = 1; // 리셋 활성화 -> START 상태로 복귀
45        #10 reset = 0;
46
47        // 추가 시나리오: 혼합된 입력 신호 검증
48        #10 din_bit = 0;
49        #10 din_bit = 1;
50        #10 din_bit = 1;
51        #10 din_bit = 0;
52        #10 din_bit = 0;
53
54        // 시뮬레이션 종료
55        #20 $finish;
56    end
```

리셋,



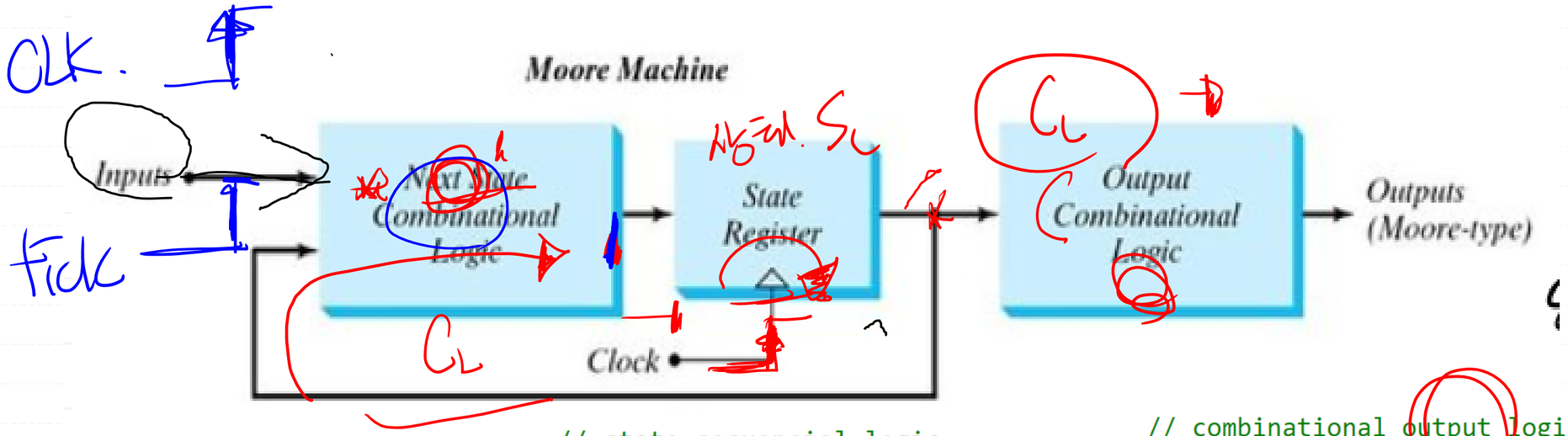
```
reg [$clog2(1_000_000)-1:0] r_counter;
reg r_tick_100hz;

assign o_tick_100hz = r_tick_100hz;

always @(posedge clk, posedge reset) be
    if (reset) begin
        r_counter <= 0;
```

clk 감시

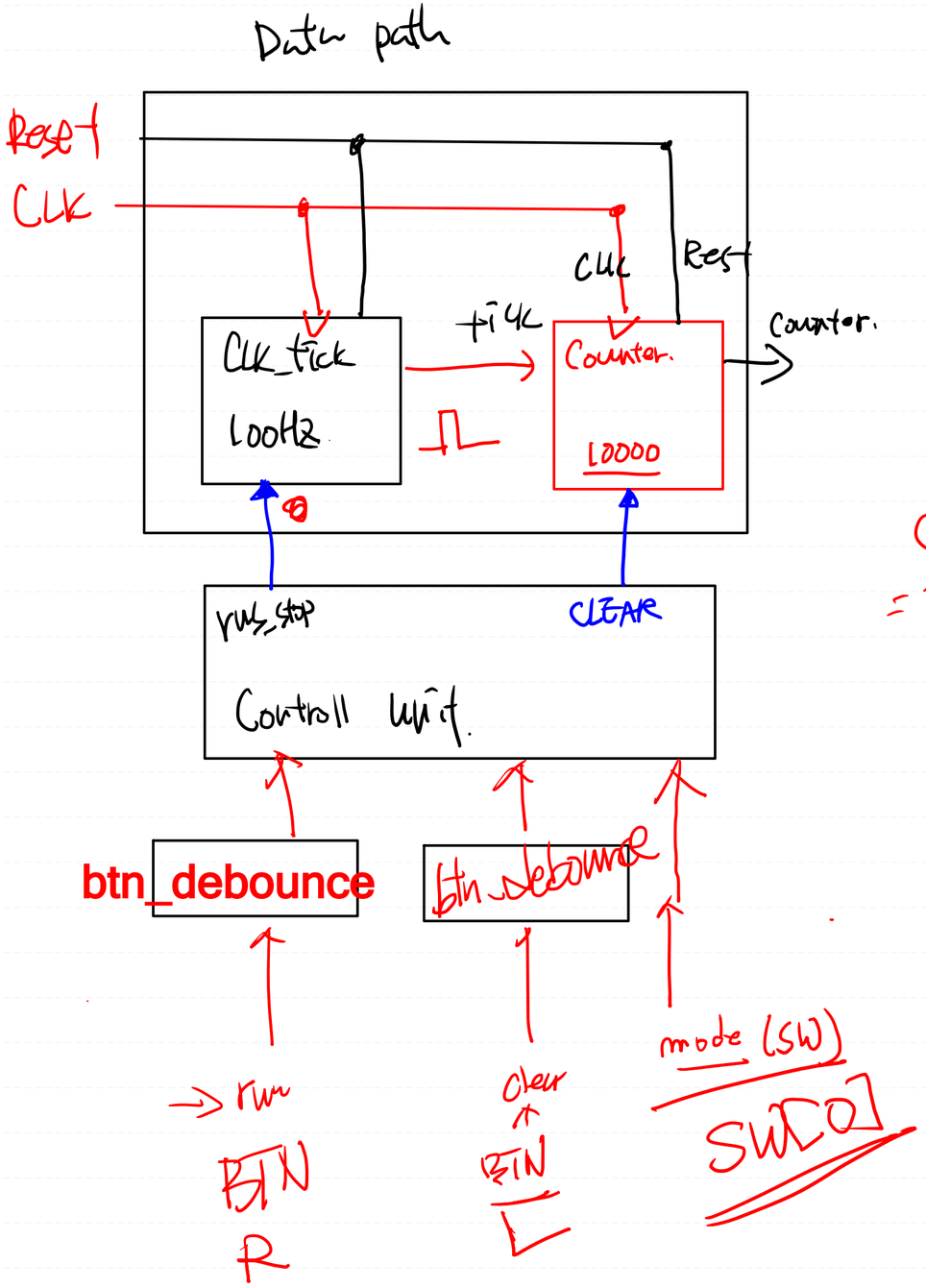
clk의 입력과 상관없이 감시

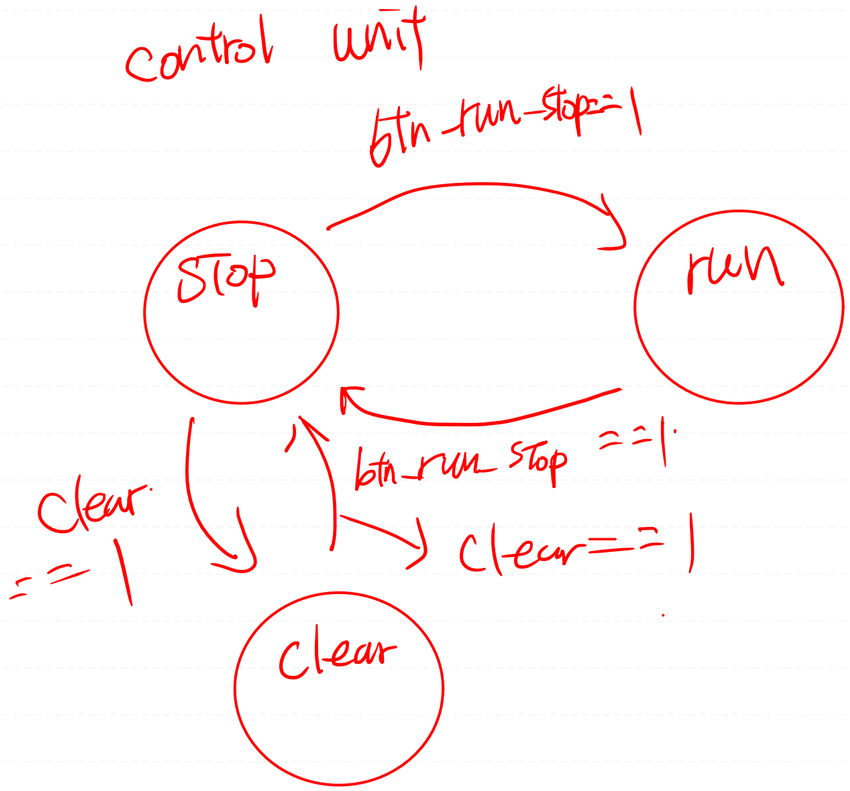


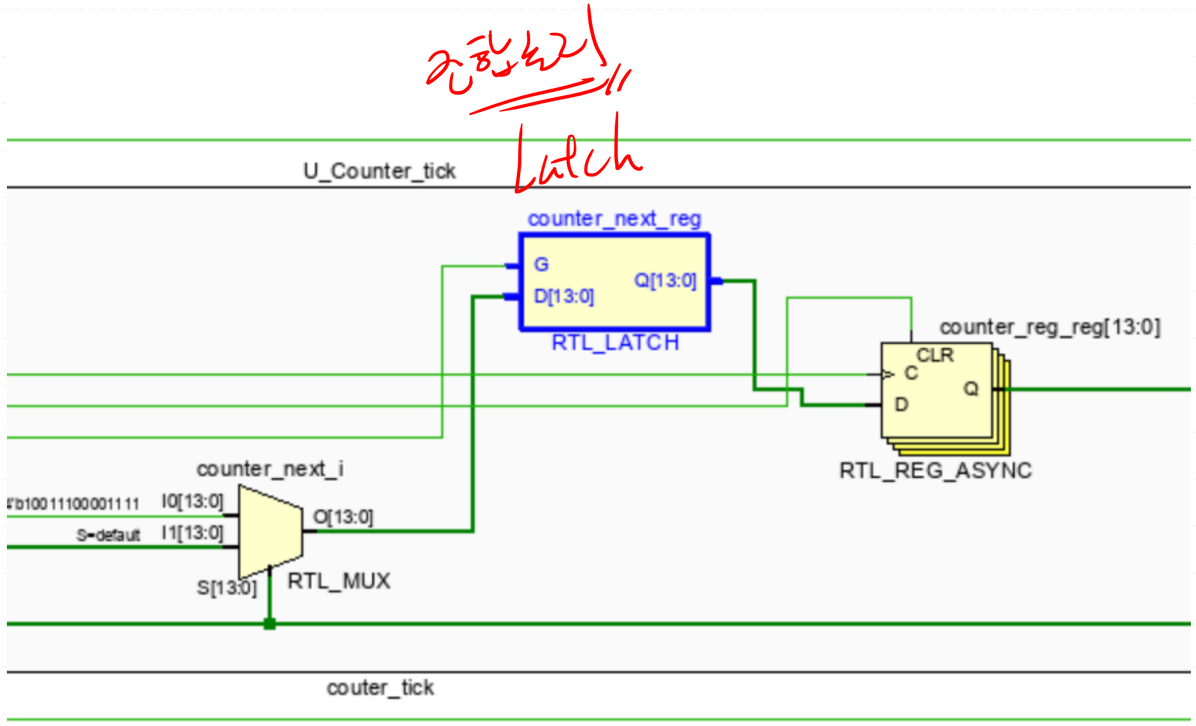

```
// next combinational logic
always @(*) begin
    next = state;
    case (state)
        STOP: begin
            if (i_run_stop == 1'b1) begin
                next = RUN;
            end else if (i_clear == 1'b1) begin
                next = CLEAR;
            end else begin
                next = state;
            end
        end
        RUN: begin
```

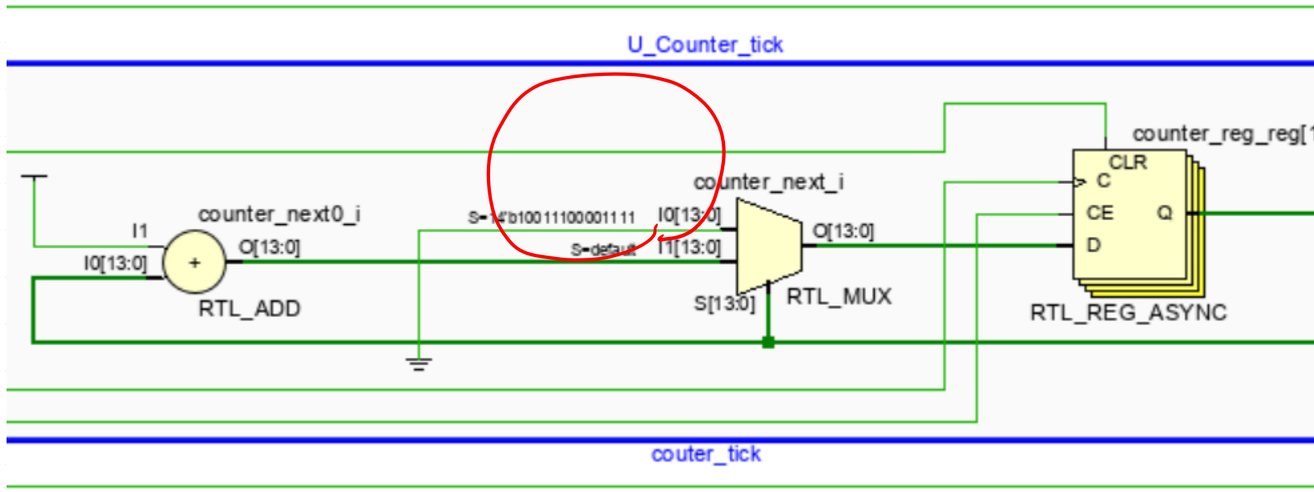
```
// state sequencial logic
always @(posedge clk, posedge reset)
    if (reset) begin
        state <= STOP;
    end else begin
        state <= next;
    end
end
```

```
// combinational output logic
always @(*) begin
    case (state)
        STOP: begin
            o_run_stop = 1'b0;
            o_clear = 1'b0;
        end
        RUN: begin
            o_run_stop = 1'b1;
            o_clear = 1'b0;
        end
        CLEAR: begin
```


Data path
Reset
CLK
CLK_tick
100Hz
tick
Counter
10000
CLK
Reset
Counter
run_stop
CLEAR
Controll unit
btn_debounce
btn_debounce
run
BTN
R
clear
BTN
L
mode (SW)
SW[0]

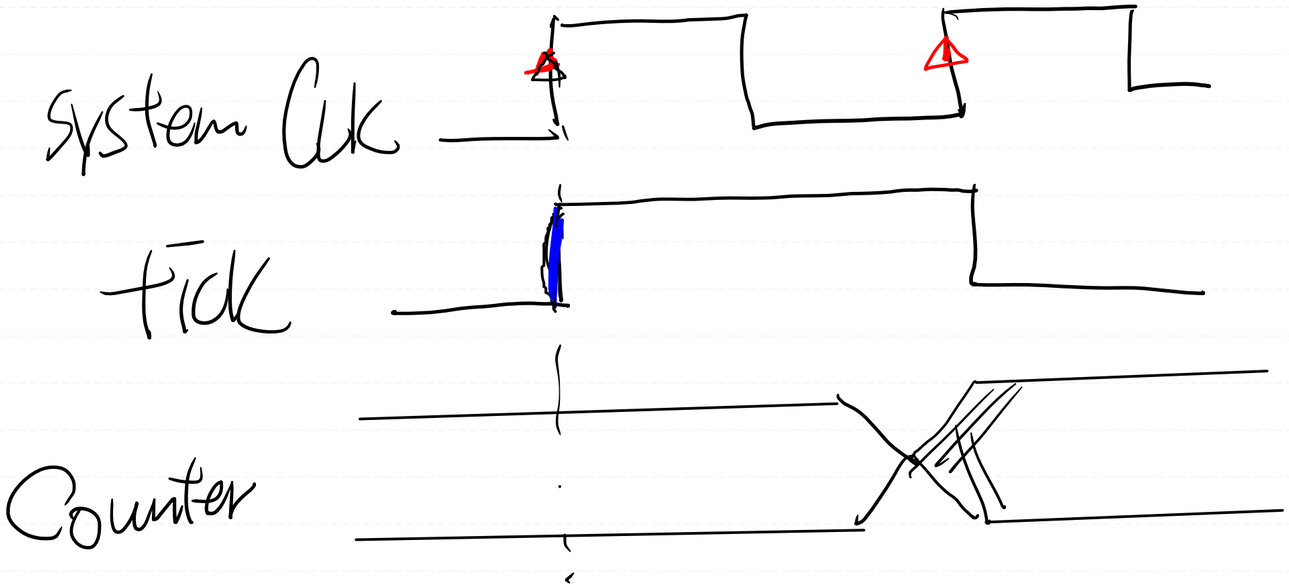

control unit
btn_run_stop==1
stop
run
clear==1
btn_run_stop==1
clear==1
clear



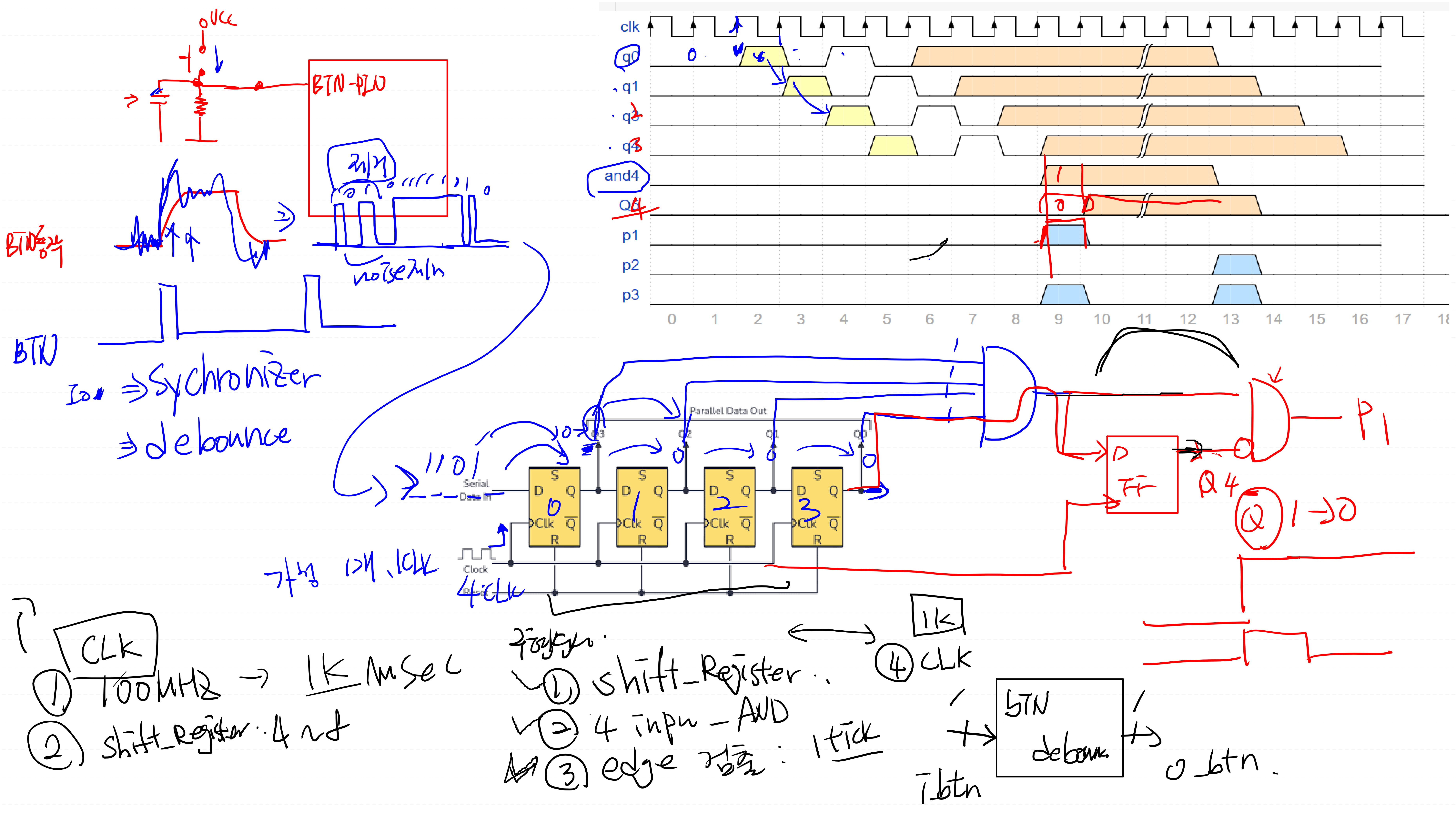



```
// next
always @(*) begin
    if (tick == 1'b1) begin  // tick count
        if (counter_reg == 10_000 - 1) begin
            counter_next = 0;
        end else begin
            counter_next = counter_reg + 1;
        end
    end
end
```

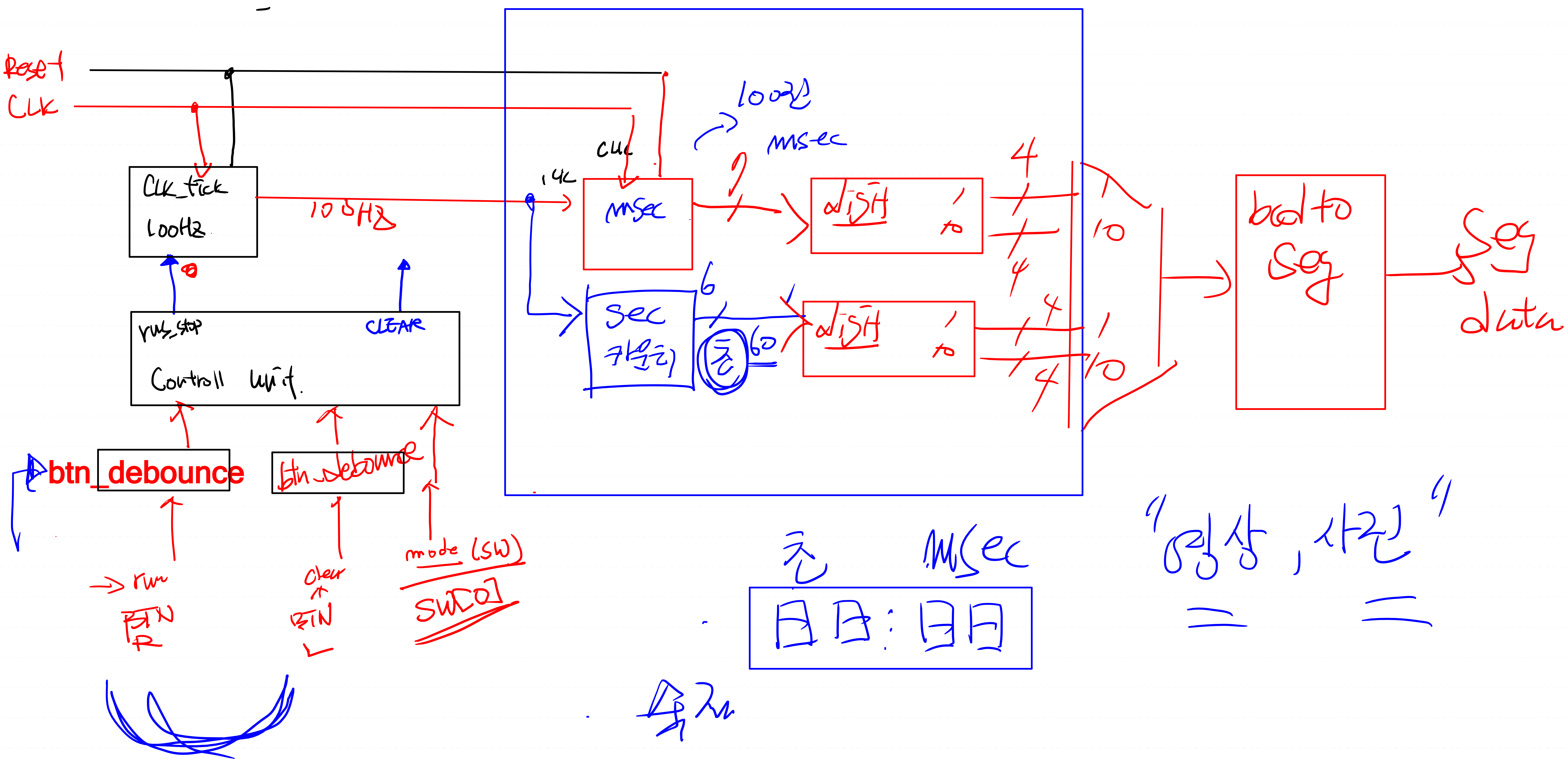
```
// next
always @(*) begin
    counter_next = counter_reg;
    if (tick == 1'b1) begin  // tick count
        if (counter_reg == 10_000 - 1) begin
            counter_next = 0;
        end else begin
            counter_next = counter_reg + 1;
        end
    end
end
```



Next = 현재값 + 1



clk
q0
q1
q2
q3
and4
Q4
p1
p2
p3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
VCC
BTN-PIN
BTN
Sychronizer
debounce
Parallel Data Out
Serial Data In
Clock
Reset
S D Q Clk R
4clk
D FF
Q4
P1
CLK
① 100MHz → 1K msec
② shift_Register 4
① shift_Register
② 4 input_AND
③ edge
1tick
④ CLK
1k
BTN debounce
i_btn
o_btn



Reset
CLK
Clk_tick
100Hz
100Hz
run_stop
CLEAR
Controll unit
btn_debounce
btn_debounce
mode (SW)
SW[0]
run
clear
BTN
clk
msec
sec
dist
bcd to seg
seg data
초
MSec
"영상, 사진"