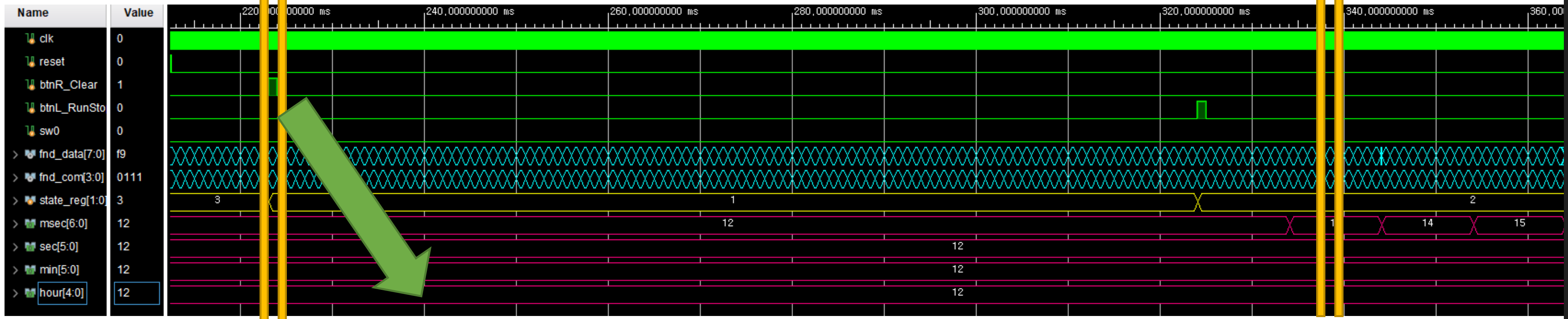
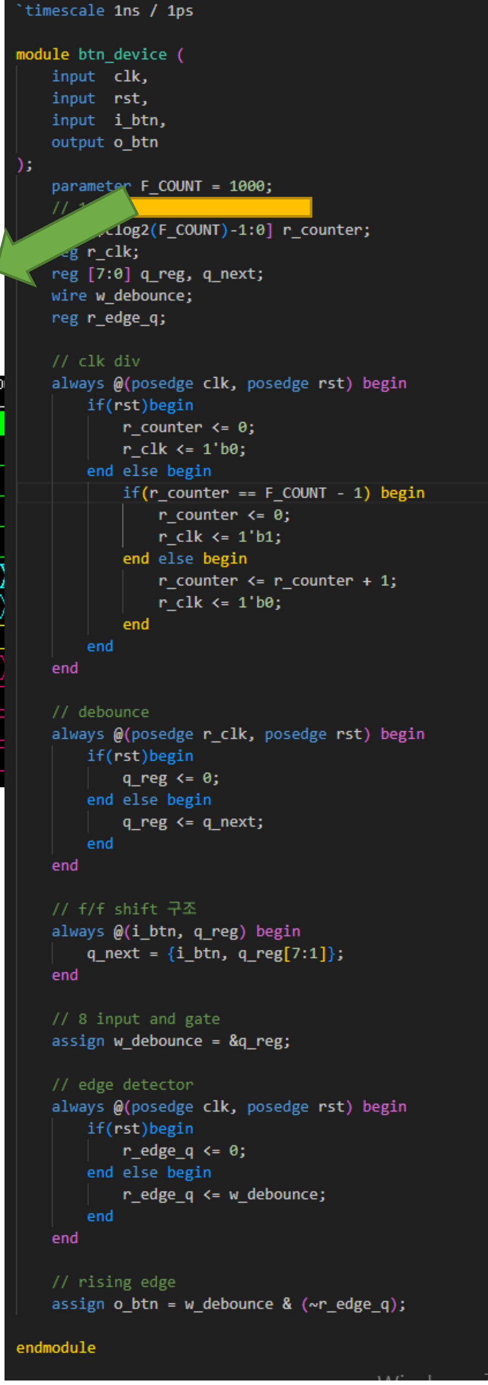
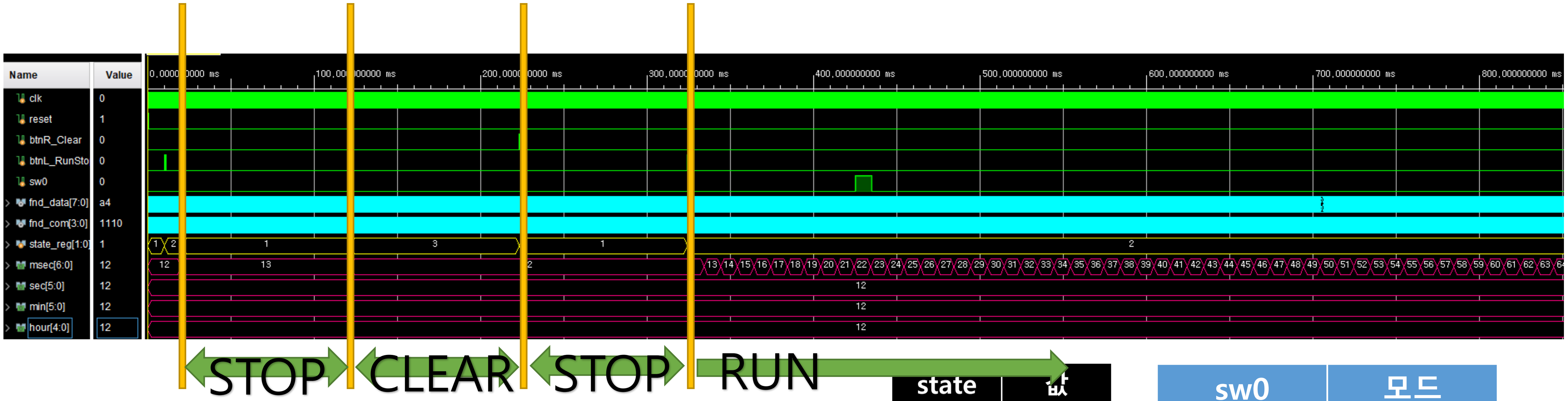


버튼이 1ms 유지해야
state에 변화가 생기는 모습

1ms debounce 방지



```
`timescale 1ns / 1ps

module btn_device (
    input  clk,
    input  rst,
    input  i_btn,
    output o_btn
);
    parameter F_COUNT = 1000;
    // 1[illegible]
    [illegible]clog2(F_COUNT)-1:0] r_counter;
    [illegible]eg r_clk;
    reg [7:0] q_reg, q_next;
    wire w_debounce;
    reg r_edge_q;

    // clk div
    always @(posedge clk, posedge rst) begin
        if(rst)begin
            r_counter <= 0;
            r_clk <= 1'b0;
        end else begin
            if(r_counter == F_COUNT - 1) begin
                r_counter <= 0;
                r_clk <= 1'b1;
            end else begin
                r_counter <= r_counter + 1;
                r_clk <= 1'b0;
            end
        end
    end

    // debounce
    always @(posedge r_clk, posedge rst) begin
        if(rst)begin
            q_reg <= 0;
        end else begin
            q_reg <= q_next;
        end
    end

    // f/f shift 구조
    always @(i_btn, q_reg) begin
        q_next = {i_btn, q_reg[7:1]};
    end

    // 8 input and gate
    assign w_debounce = &q_reg;

    // edge detector
    always @(posedge clk, posedge rst) begin
        if(rst)begin
            r_edge_q <= 0;
        end else begin
            r_edge_q <= w_debounce;
        end
    end

    // rising edge
    assign o_btn = w_debounce & (~r_edge_q);

endmodule
```



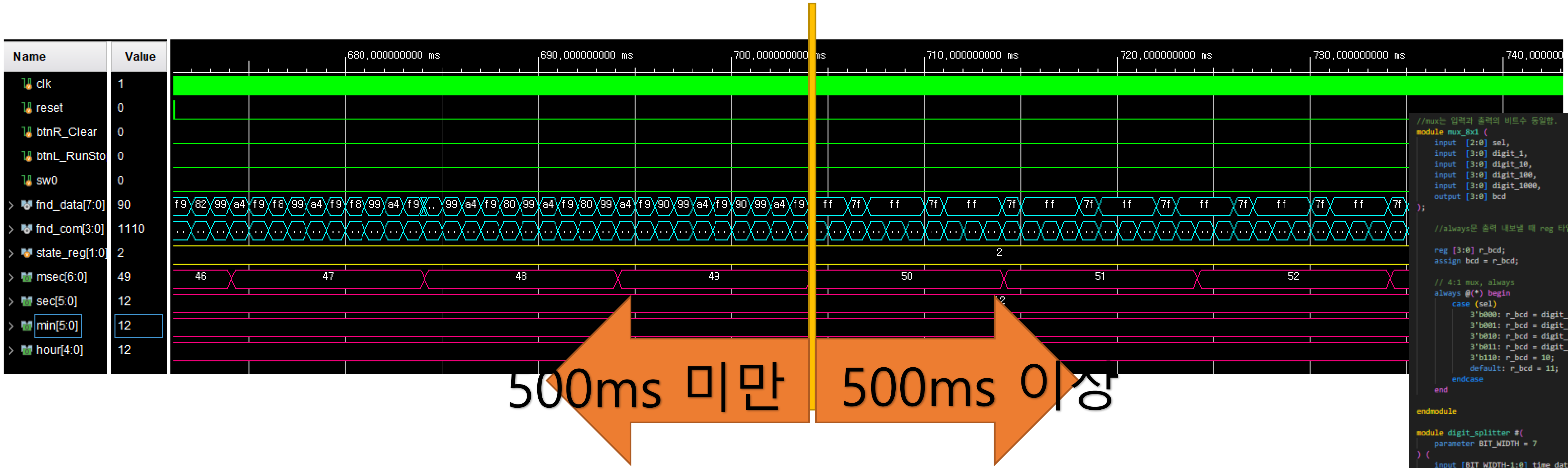
Name	Value
clk	0
reset	1
btnR_Clear	0
btnL_RunSto	0
sw0	0
fnd_data[7:0]	a4
fnd_com[3:0]	1110
state_reg[1:0]	1
msec[6:0]	12
sec[5:0]	12
min[5:0]	12
hour[4:0]	12

msec, sec, min, hour 카운터의 모든 초기값 12 설정

msec, sec, min, hour 카운터의 모든 초기값 12 설정

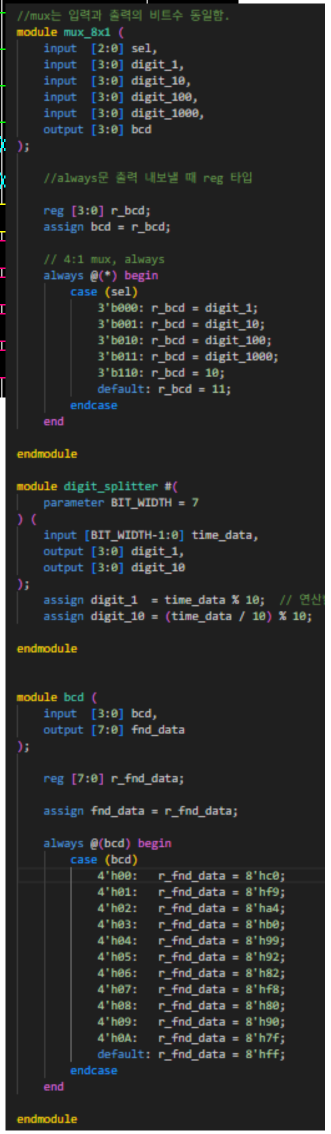
state	값
STOP	1
RUN	2
CLEAR	3

sw0	모드
0	msec, sec
1	Min, hour



```
Name            Value
clk             1
reset           0
btnR_Clear      0
btnL_RunSto     0
sw0             0
fnd_data[7:0]   90
fnd_com[3:0]    1110
state_reg[1:0]  2
msec[6:0]       49
sec[5:0]        12
min[5:0]        12
hour[4:0]       12
```

숫자 출력 . 출력



```
//mux는 입력과 출력의 비트수 동일함.
module mux_8x1 (
    input  [2:0] sel,
    input  [3:0] digit_1,
    input  [3:0] digit_10,
    input  [3:0] digit_100,
    input  [3:0] digit_1000,
    output [3:0] bcd
);

    //always문 출력 내보낼 때 reg 타입

    reg [3:0] r_bcd;
    assign bcd = r_bcd;

    // 4:1 mux, always
    always @(*) begin
        case (sel)
            3'b000: r_bcd = digit_1;
            3'b001: r_bcd = digit_10;
            3'b010: r_bcd = digit_100;
            3'b011: r_bcd = digit_1000;
            3'b110: r_bcd = 10;
            default: r_bcd = 11;
        endcase
    end

endmodule

module digit_splitter #(
    parameter BIT_WIDTH = 7
) (
    input [BIT_WIDTH-1:0] time_data,
    output [3:0] digit_1,
    output [3:0] digit_10
);
    assign digit_1  = time_data % 10;  // 연산
    assign digit_10 = (time_data / 10) % 10;

endmodule


module bcd (
    input  [3:0] bcd,
    output [7:0] fnd_data
);

    reg [7:0] r_fnd_data;

    assign fnd_data = r_fnd_data;

    always @(bcd) begin
        case (bcd)
            4'h00:   r_fnd_data = 8'hc0;
            4'h01:   r_fnd_data = 8'hf9;
            4'h02:   r_fnd_data = 8'ha4;
            4'h03:   r_fnd_data = 8'hb0;
            4'h04:   r_fnd_data = 8'h99;
            4'h05:   r_fnd_data = 8'h92;
            4'h06:   r_fnd_data = 8'h82;
            4'h07:   r_fnd_data = 8'hf8;
            4'h08:   r_fnd_data = 8'h80;
            4'h09:   r_fnd_data = 8'h90;
            4'h0A:   r_fnd_data = 8'h7f;
            default: r_fnd_data = 8'hff;
        endcase
    end

endmodule
```