대한상공회의소
서울기술교육센터

UART WATCH CONTROL SYSYEM

Verilog Project

AI 시스템반도체 설계 2기

엄찬하

20205.05.26

목 차

Title

프로젝트 개요

주요 기능

- PC에서 UART를 통해 명령을 수신하여 FPGA 내부에서 스톱워치를 제어하는 시스템 구현
- CMD_PROCESSOR를 통해 명령 해석
- 시계를 FPGA에서 제어

목적 및 목표

- UART 통신을 통한 명령 수신 및 전송
- PROCESSOR를 통한 명령 해석 및 실행
- 디지털 스톱워치 및 시계 기능
- 디지털 디스플레이 (FND) 출력

설계 도구

설계 언어

- SystemVerilog

개발 툴

- Vivado 2020.2

합성

- Vivado Synthesis

<Basys-3>

Xilinx Artix-7

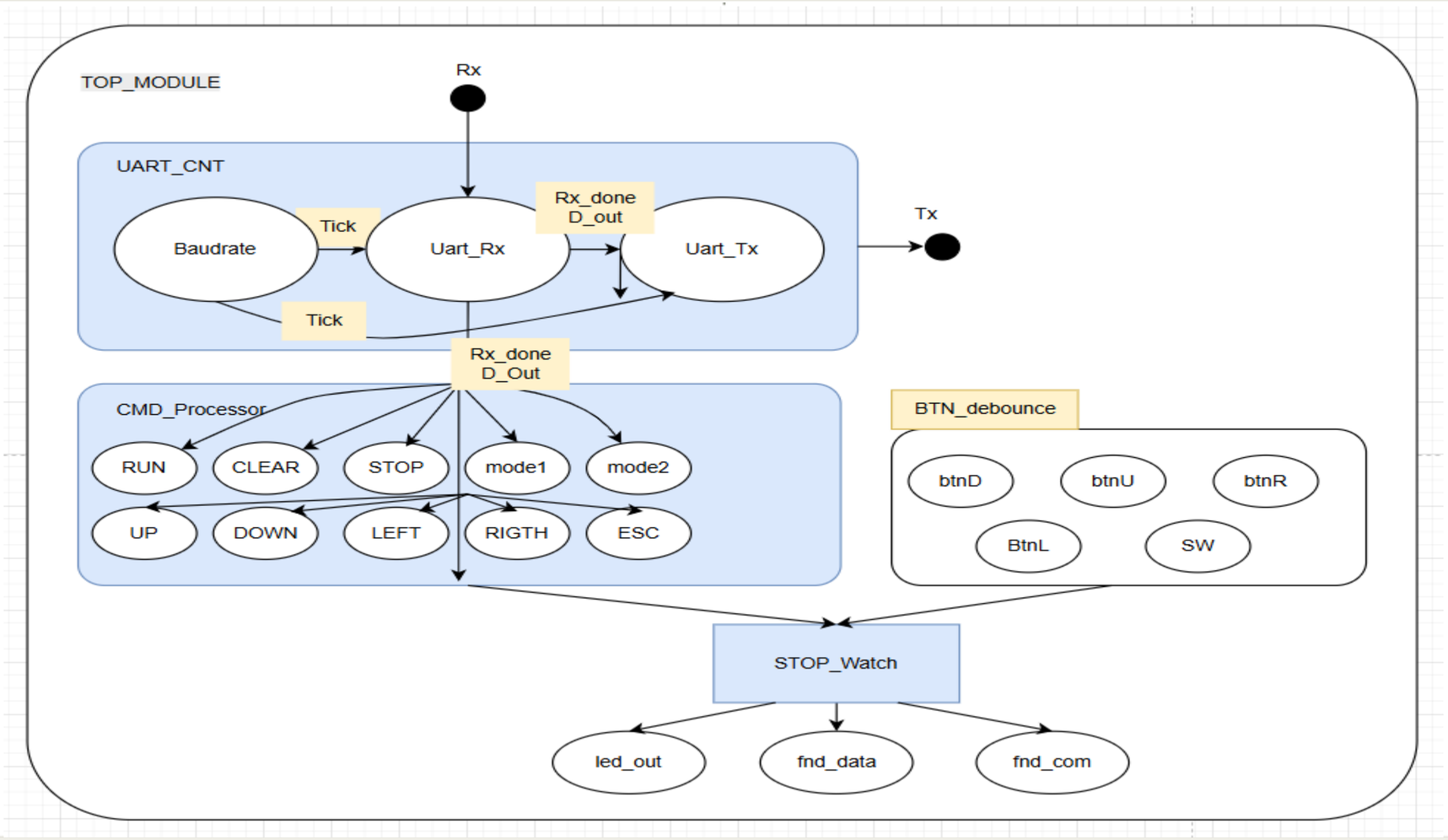
100MHz CMOS clk

3.3V

<ComPortMaster>

UART 송수신 데이터 실시간 확인

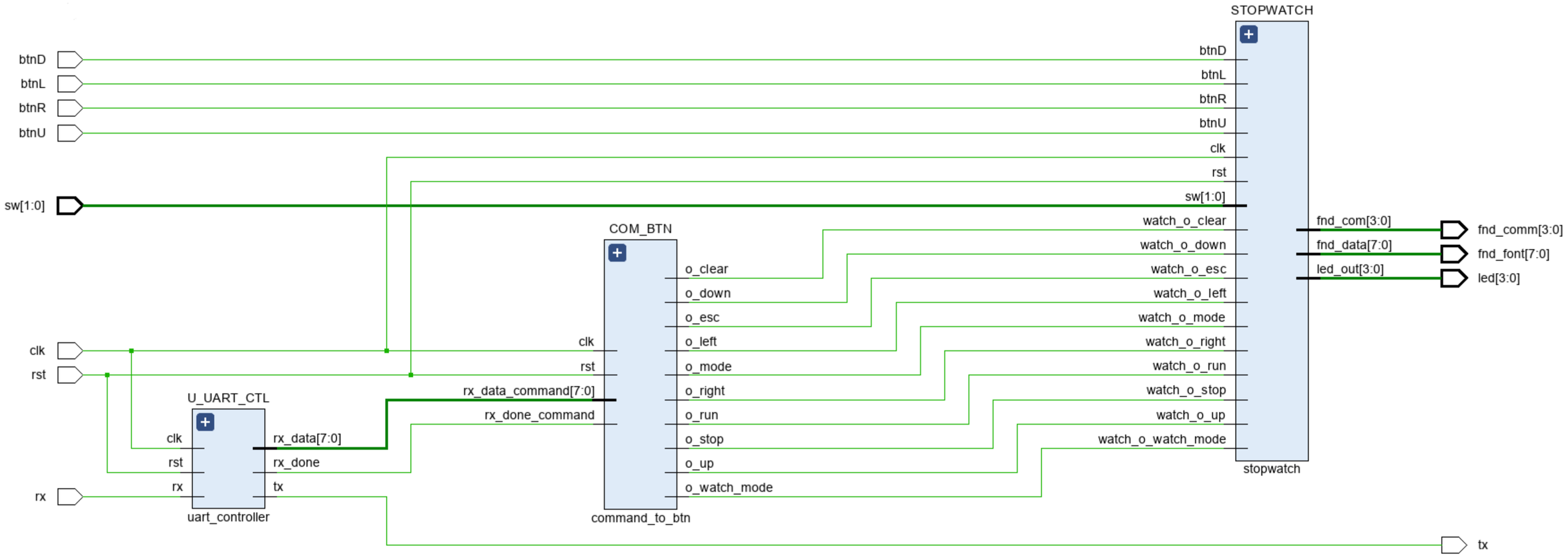
BLOCK DIAGRAM



WATCH CONTROL SYSTEM

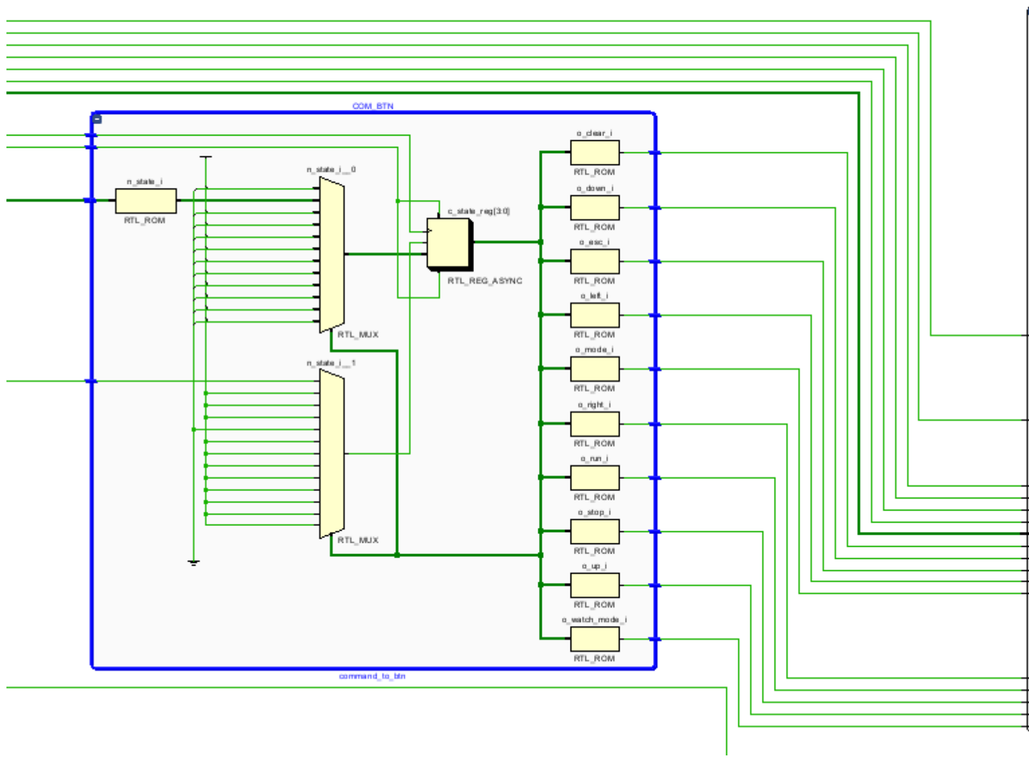
Schematic

- UART_CTN
- CMD_BTN
- STOPWATCH

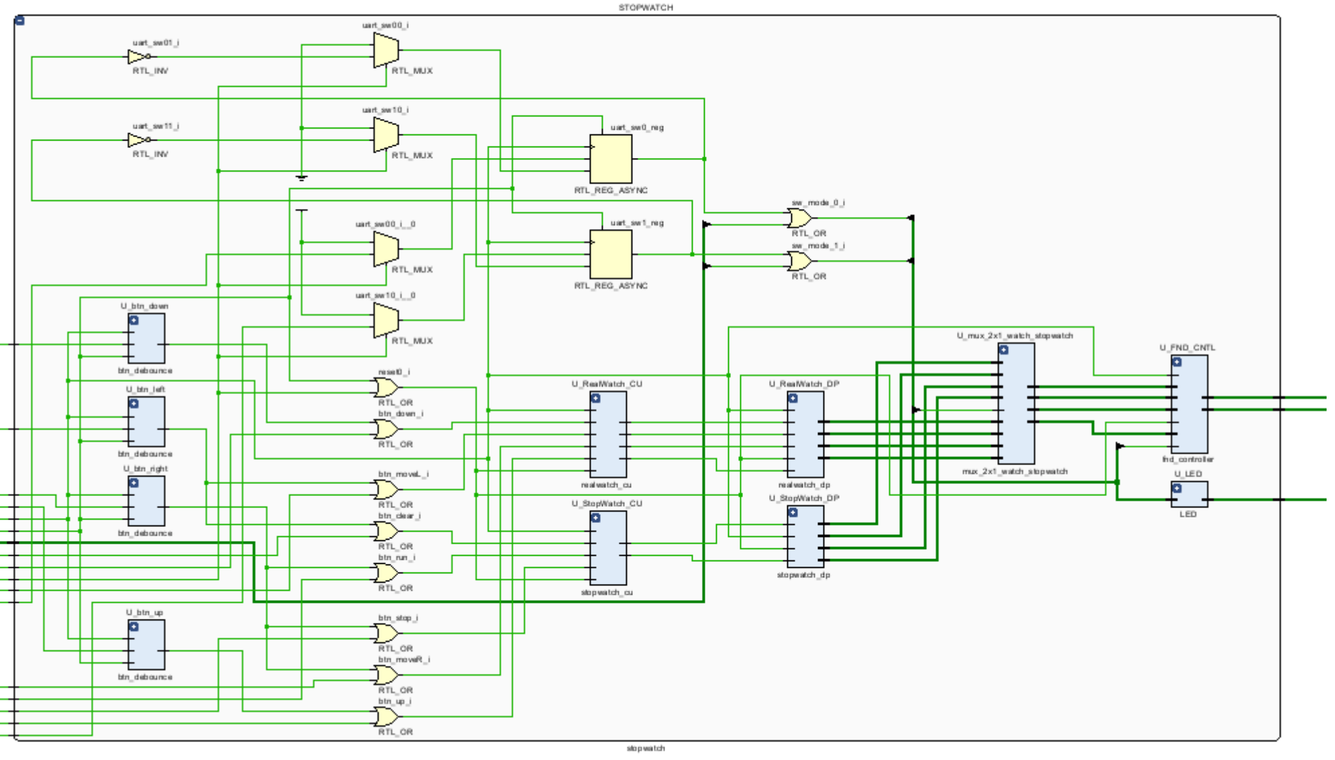


Schematic (세부)

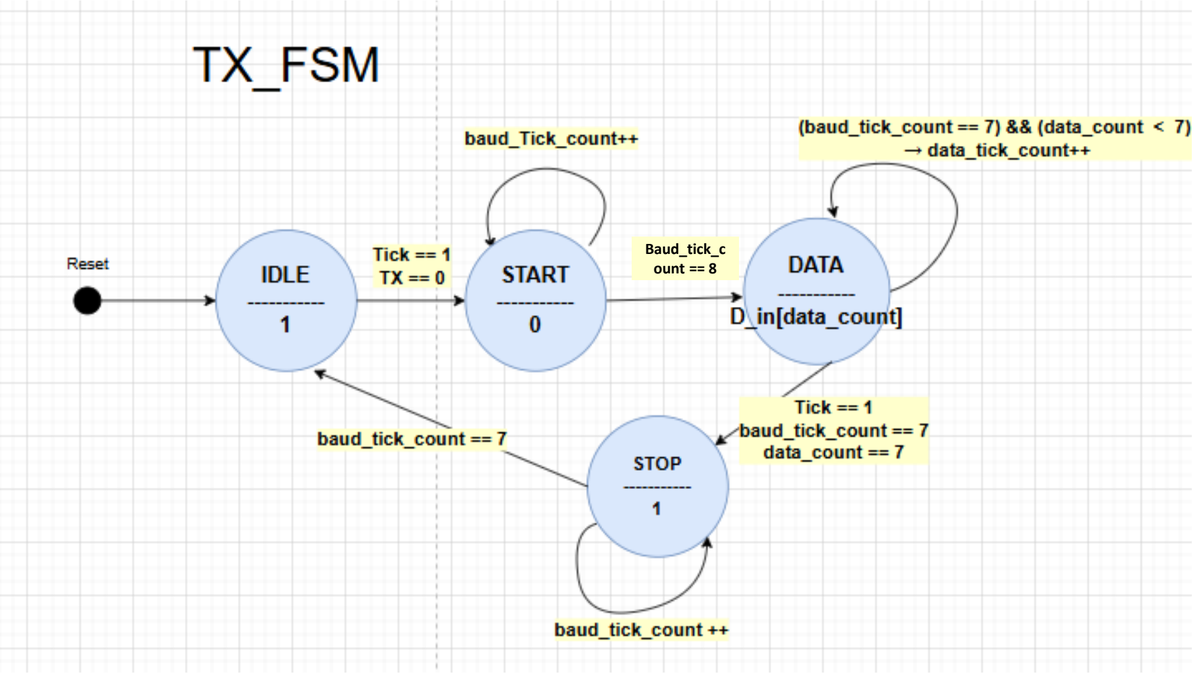
CMD_BTN



STOPWATCH



UART_TX_FSM



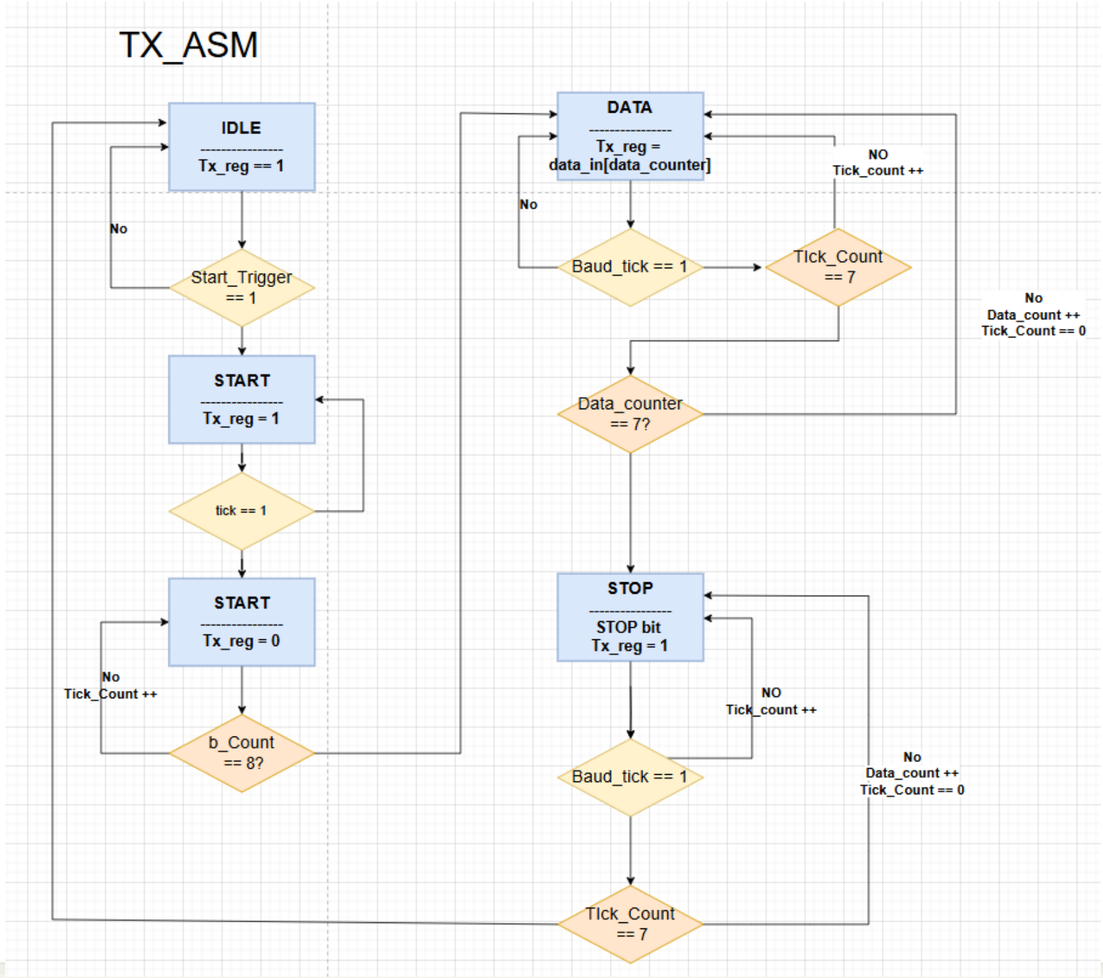
<CMD_BTN> Tx **대기상태**
Start_Trigger **감지되면 이동**

<START> **Tx를 low(0)로 설정, bit 전송**
Baud_tick_count == 8까지 증가

<DATA> **Tx_reg에 1비트씩 전송**
Baud_tick_count == 7이고 data 모두 전송 시 STOP으로 이동

<STOP> **Tx_reg = 1로 stop 상태 전송**
Baud_tick == 7일 때 IDLE로 복귀

UART_TX_FSM



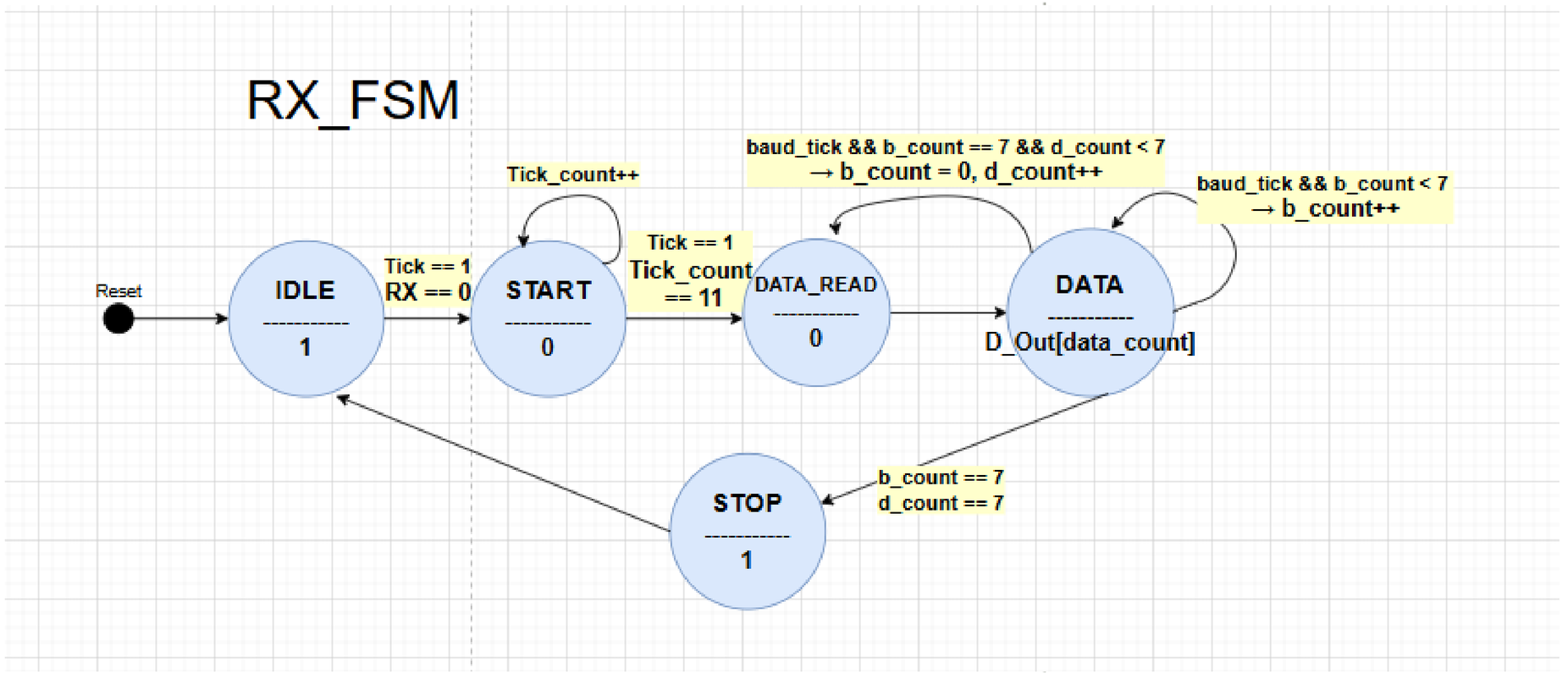
<CMD_BTN> Tx 대기상태
Start_Trigger 감지되면 이동

<START> Tx를 low(0)로 설정, bit 전송
Baud_tick_count == 8까지 증가

<DATA> Tx_reg에 1비트씩 전송
Baud_tick_count == 7이고 data 모두 전송 시 STOP으로 이동

<STOP> Tx_reg = 1로 stop 상태 전송
Baud_tick == 7일 때 IDLE로 복귀

UART_RX



<IDLE>
Rx = 0 일 때 START로 분기
모든 카운터 초기화

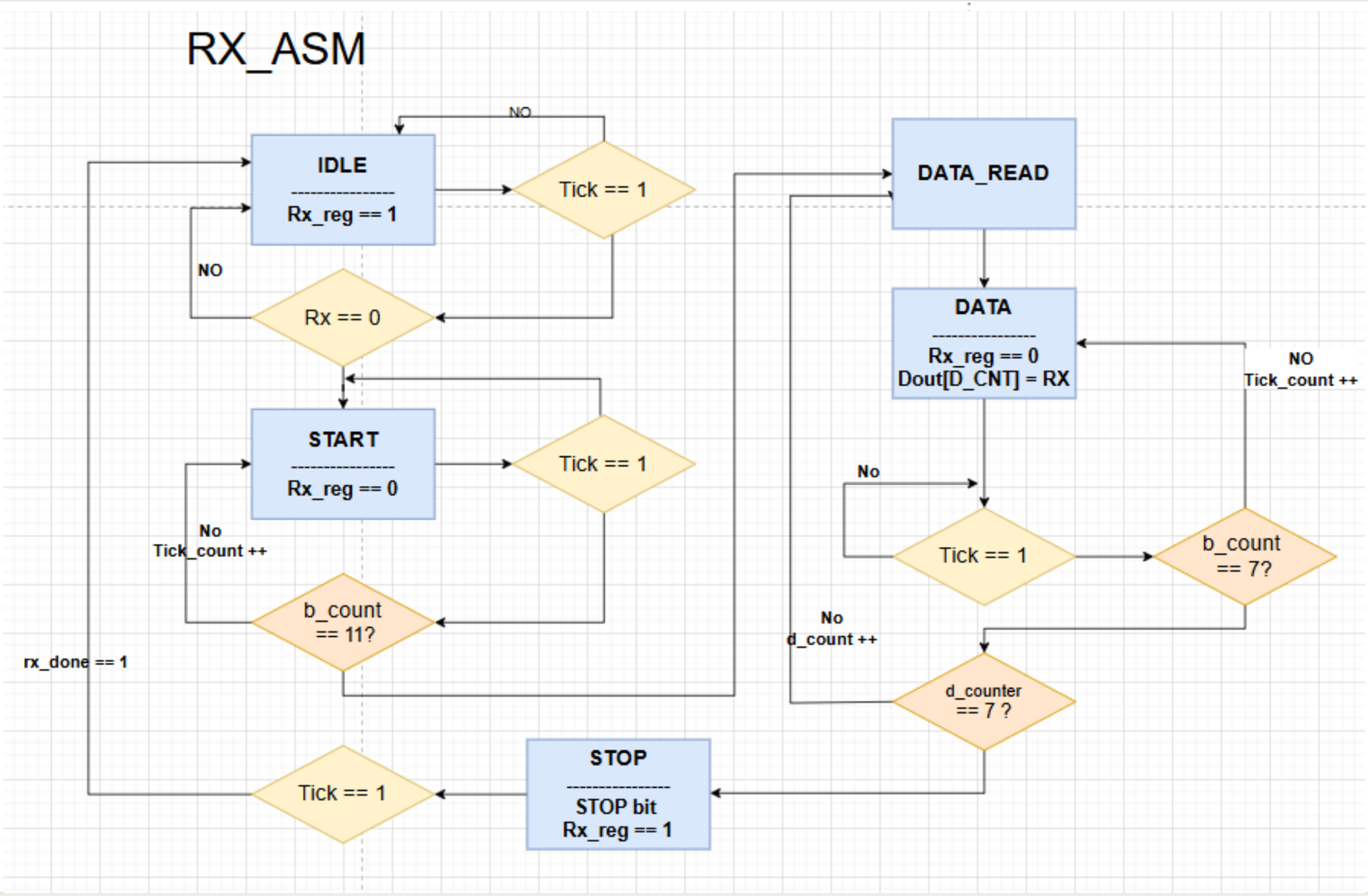
<START>
Rx == 0
B_tick = 11이 되면 DATA로 분기

<DATA_READ>
D_out = {rx, dout_reg[7:1]}
시프트 방식으로 1비트씩 누적 저장

<DATA>
B_count == 7 후 데이터 샘플링
8비트 모두 수신되면 STOP

<STOP>
Rx_reg == 1, tick이 들어오면
Rx_done == 1로 만들고 IDEL 복귀

UART_RX



<IDLE>
Rx = 0 일 때 START로 분기
모든 카운터 초기화

<START>
Rx == 0
B_tick = 11이 되면 DATA로 분기

<DATA_READ>
D_out = {rx, dout_reg[7:1]}
시프트 방식으로 1비트씩 누적 저장

<DATA>
B_count == 7 후 데이터 샘플링
8비트 모두 수신되면 STOP

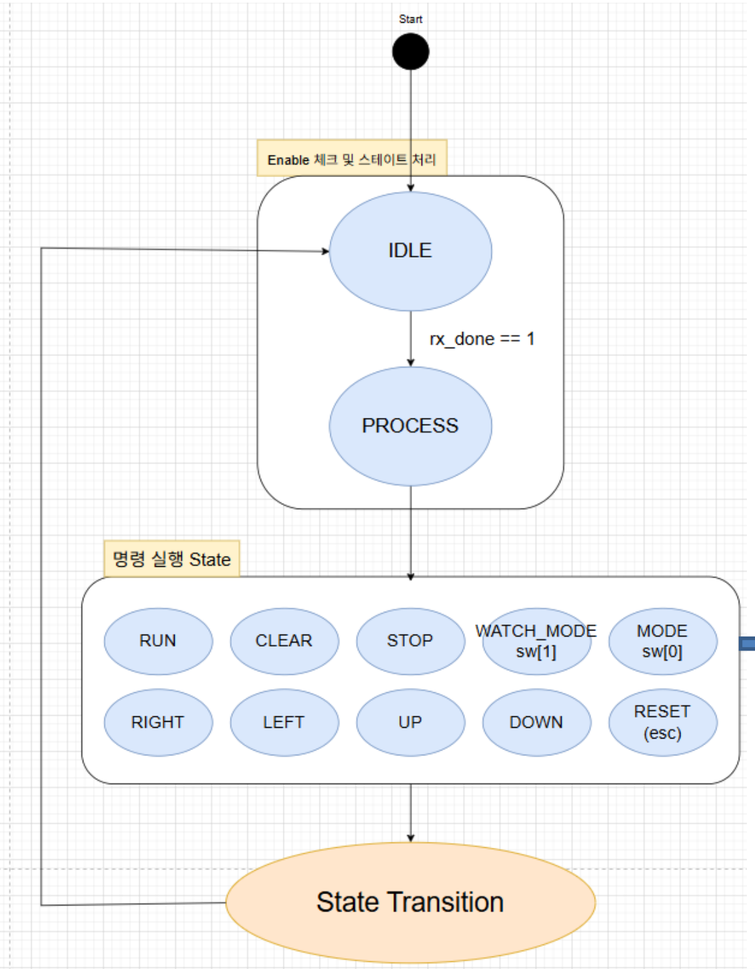
<STOP>
Rx_reg == 1, tick이 들어오면
Rx_done == 1로 만들고 IDEL 복귀

COMMAND_BTN

Rx_data를 읽어 Case 따라 명령 해석
G(run), S(stop), C(clear), M(sw[0]), N(sw[1]), R(move_R),
L(move_L), U(move_U), D(move_D), ESC(rst) 명령 처리

IDLE	Rx_done 수신 대기
PROCESS	데이터 읽기 및 case로 분할 된 명령 해석

RUN	스톱워치 시작	RIGHT	Move_R
CLEAR	스톱워치 clear	LEFT	Move_L
STOP	스톱워치 정지	UP	Move_U
WATCH_M	SW[1]	DOWN	Move_D
MODE	SW[0]	ESC	RESET




```
always @(*) begin
    n_state = c_state;
    case (c_state)
        IDLE: begin
            if (rx_done_command) begin
                n_state <= PROCESS;
            end
        end
        PROCESS: begin
            case (rx_data_command)
                8'h47, 8'h67: n_state <= RUN_STATE;  //'G', 'g'
                8'h43, 8'h63: n_state <= CLEAR_STATE;  // 'C', 'c'
                8'h53, 8'h73: n_state <= STOP_STATE;  // 'S', 'sn_
                8'h4E, 8'h6E: n_state <= WATCH_MODE_STATE;  // 'N'
                8'h4D, 8'h6D: n_state <= MODE_STATE;  // 'M', 'm'
                8'h52, 8'h72: n_state <= R_STATE;  // 'R', 'r'
                8'h55, 8'h75: n_state <= U_STATE;  // 'U', 'u'
                8'h44, 8'h64: n_state <= D_STATE;  // 'D', 'd'
                8'h4C, 8'h6C: n_state <= L_STATE;  // 'L', 'l'

                8'h1B:   n_state <= RESET_STATE;  // 'ESC'
                default: n_state <= IDLE;
            endcase
        end
        RUN_STATE: begin
            n_state <= IDLE;
        end
        CLEAR_STATE: begin
            n_state <= IDLE;
        end
        STOP_STATE: begin
            n_state <= IDLE;
        end
        WATCH_MODE_STATE: begin
            n_state <= IDLE;
        end
        MODE_STATE: begin
            n_state <= IDLE;
        end
        RESET_STATE: begin
            n_state <= IDLE;
        end
        R_STATE: begin
            n_state <= IDLE;
        end
        U_STATE: begin
            n_state <= IDLE;
        end
        D_STATE: begin
            n_state <= IDLE;
        end
        L_STATE: begin
            n_state <= IDLE;
        end
    endcase
end
```

Troubleshooting

개선 전

```
assign btn_L      = w_btnL | watch_o_clear | watch_o_left;
assign btn_R      = w_btnR | watch_o_run | watch_o_stop | watch_o_right ;
assign btn_U      = w_btnU | watch_o_up;
assign btn_D      = w_btnD | watch_o_down;
```

버튼에 나머지 동작을 or 연산하여 다른 버튼으로도 중복 동작이 됨

```
assign sw_mode[1] = (watch_o_watch_mode) | sw[1];
assign sw_mode[0] = (watch_o_mode) | sw[0];
```

Sw의 경우 UART 우선순위로 지정하며 동작이 잘 되지 않음

개선 후

```
reg uart_sw0, uart_sw1;
reg sw1_check = 0;

always @(posedge clk or posedge rst) begin
    if (rst || watch_o_esc) begin
        if (uart_sw1 == 1 & !sw1_check)   begin
            uart_sw0 <= 0;
            sw1_check <= 1;
        end else begin
            uart_sw0 <= 0;
            uart_sw1 <= 0;
            sw1_check <= 0;
        end
    end else begin
        if (watch_o_mode) uart_sw0 <= ~uart_sw0;
        if (watch_o_watch_mode) uart_sw1 <= ~uart_sw1;
    end
end
```

```
assign btn_clear = w_btnL | watch_o_clear;
assign btn_run   = w_btnR | watch_o_run;
assign btn_stop  = w_btnR | watch_o_stop;

assign btn_moveL = w_btnL | watch_o_left;
assign btn_moveR = w_btnR | watch_o_right;
assign btn_up    = w_btnU | watch_o_up;
assign btn_down  = w_btnD | watch_o_down;
```

Uart로 받은 sw에 따른 상태를 저장하게 하여 N,M으로 스위치 제어 및 반전을 통해 재동작 가능하게 설정

나머지 버튼의 동작을 세부적으로 나누어 처리하여 중복 방지

Troubleshooting

개선 전

```
always @(*) begin
    next = state;
    case (state)
        STOP: begin
            if (i_btn_runstop) begin
                next = RUN;
            end else if (i_btn_clear) begin
                next = CLEAR;
            end
        end
        RUN: begin
            if (i_btn_runstop) begin
                next = STOP;
            end
        end
        CLEAR: begin
            if (i_btn_clear) begin
                next = STOP;
            end
        end
        default: next = state;
    endcase
end
```

Stopwatch의 cu에서 run과 stop이 같은 라인에 배치

개선 후

```
always @(*) begin
    next = state;
    case (state)
        STOP: begin
            if (i_btn_run) begin
                next = RUN;
            end else if (i_btn_clear) begin
                next = CLEAR;
            end
        end
        RUN: begin
            if (i_btn_stop) begin
                next = STOP;
            end
        end
        CLEAR: begin
            if (i_btn_clear == 0) begin
                next = STOP;
            end
        end
        default: next = state;
    endcase
end

always @(posedge clk or posedge rst) begin
    if (rst) o_runstop <= 0;
    else if (next == RUN) o_runstop <= 1;
    else if (next == STOP) o_runstop <= 0;
end
```

Stopwatch의 cu에서 run과 stop을 분리하여 안정적인 처리

SIMULATION

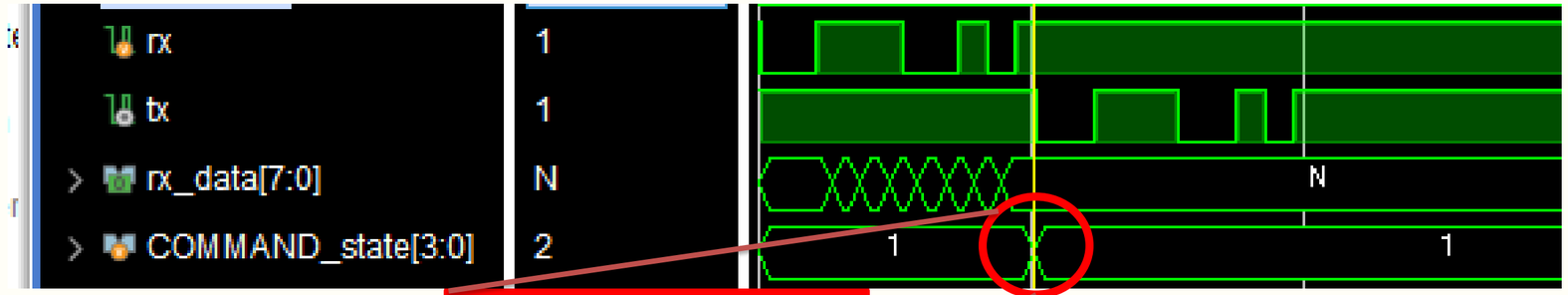
```
// UART send Task
task uart_send_byte;
    input [7:0] data;
    integer i;
    begin
        // Start bit
        rx = 0;
        #(10416 * 10);

        // Send 8 bits (LSB first)
        for (i = 0; i < 8; i = i + 1) begin
            rx = data[i];
            #(10416 * 10);
        end

        // Stop bit
        rx = 1;
        #(10416 * 10);
    end
endtask
```

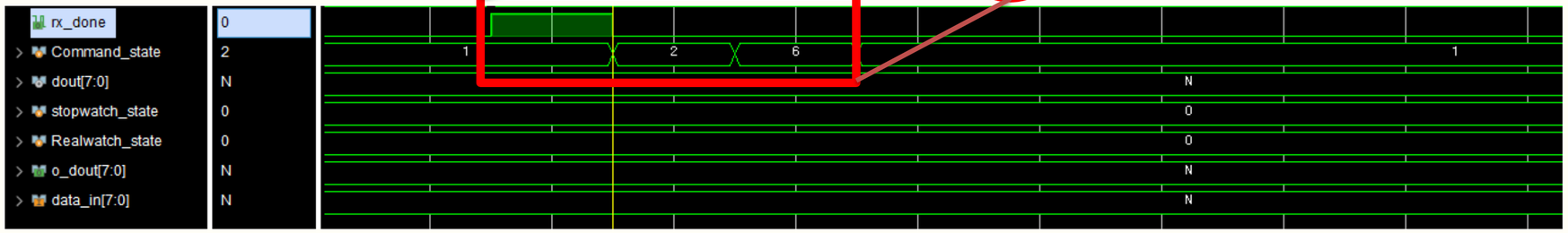
```
// Send all uppercase command characters
uart_send_byte("N");  // Watch -> stopwatch mode
#(200000 * 10);

uart_send_byte("G");  // Run
#(200000 * 10);

uart_send_byte("S");  // stop
#(200000 * 10);

uart_send_byte("C");  // clear
#(200000 * 10);

uart_send_byte(8'h1B);  // esc -> rst
#(200000 * 10);

uart_send_byte("N");  // stopwatch -> Realwatch mode
#(200000 * 10);

uart_send_byte("U");  // Up -> sec 올리기
#(200000 * 10);

uart_send_byte("M");  // Mode -> hour로 전환
#(200000 * 10);

uart_send_byte("L");  // Left
#(200000 * 10);

uart_send_byte("U");  // Up -> min 올리기
#(200000 * 10);

uart_send_byte("D");  // Down -> min 낮추기기
#(200000 * 10);

uart_send_byte("L");  // Left -> hour 선택
#(200000 * 10);

uart_send_byte("U");  // Up -> hour 올리기
#(200000 * 10);

uart_send_byte("R");  // Right -> min 선택
#(200000 * 10);

uart_send_byte("U");  // Up -> min 올리기
#(200000 * 10);

uart_send_byte(8'h1B);  // esc -> rst
#(200000 * 10);
```

SIMULATION

Sending N, G, S, C

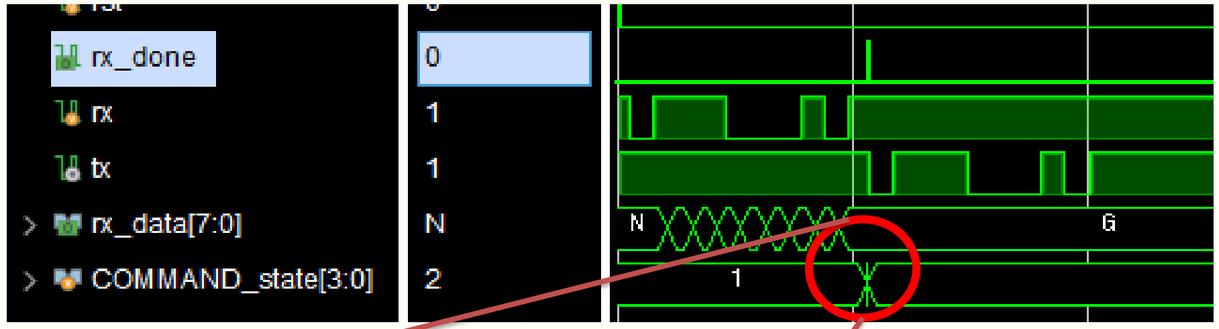
```
parameter IDLE = 4'd1, PROCESS = 4'd2, RUN_STATE = 4'd3, CLEAR_STATE = 4'd4, STOP_STATE = 4'd5,
          WATCH_MODE_STATE = 4'd6, MODE_STATE = 4'd7, RESET_STATE = 4'd8,
          R_STATE = 4'd9, U_STATE = 4'd10, D_STATE = 4'd11, L_STATE = 4'd12, WAIT = 4'd13;
    parameter STOP = 2'b00, RUN = 2'b01, CLEAR = 2'b10;
```

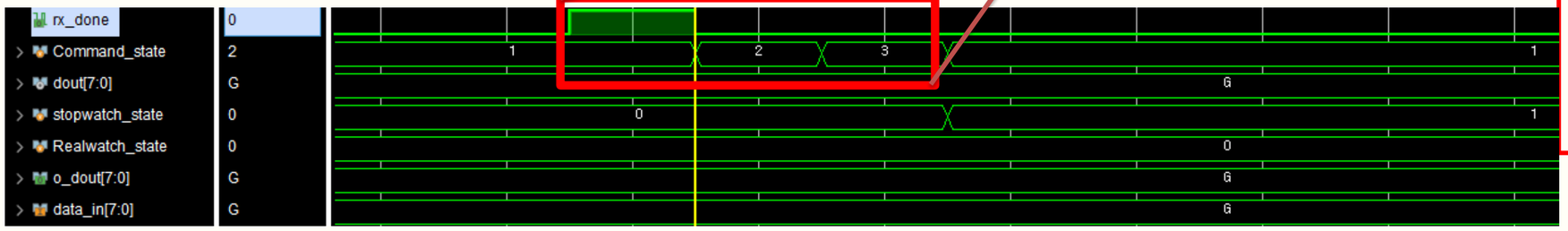




Rx_done이 들어왔을 때, (N)

IDLE -> PROCESS -> Watch_MODE_STATE
1 -> 2 -> 6





Rx_done이 들어왔을 때, (G)

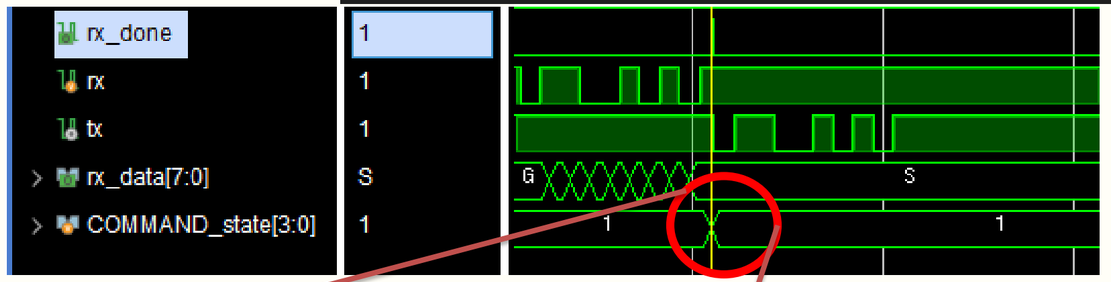
IDLE -> PROCESS -> RUN_STATE
1 -> 2 -> 3

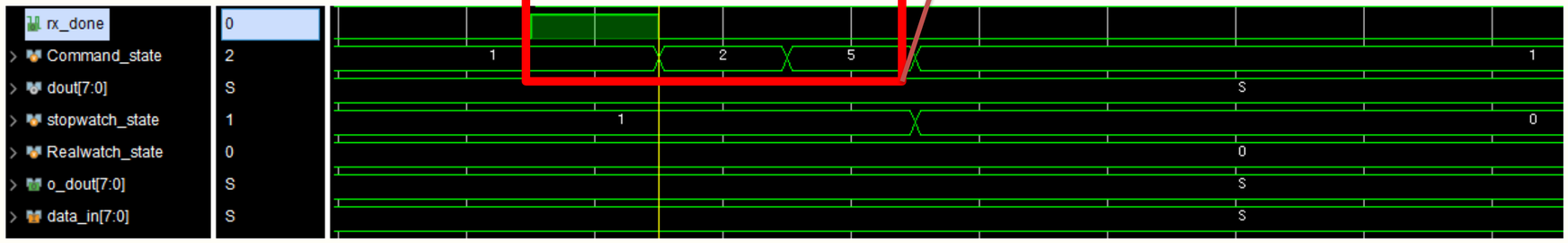
Stop watch의 state가 0->1로
(stop->run)

SIMULATION

Sending N, G, S, C

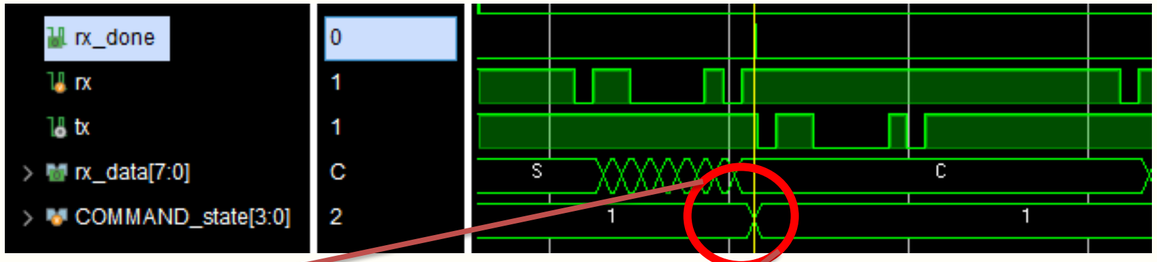
```
parameter IDLE = 4'd1, PROCESS = 4'd2, RUN_STATE = 4'd3, CLEAR_STATE = 4'd4, STOP_STATE = 4'd5,
          WATCH_MODE_STATE = 4'd6, MODE_STATE = 4'd7, RESET_STATE = 4'd8,
          R_STATE = 4'd9, U_STATE = 4'd10, D_STATE = 4'd11, L_STATE = 4'd12, WAIT = 4'd13;
parameter STOP = 2'b00, RUN = 2'b01, CLEAR = 2'b10;
```

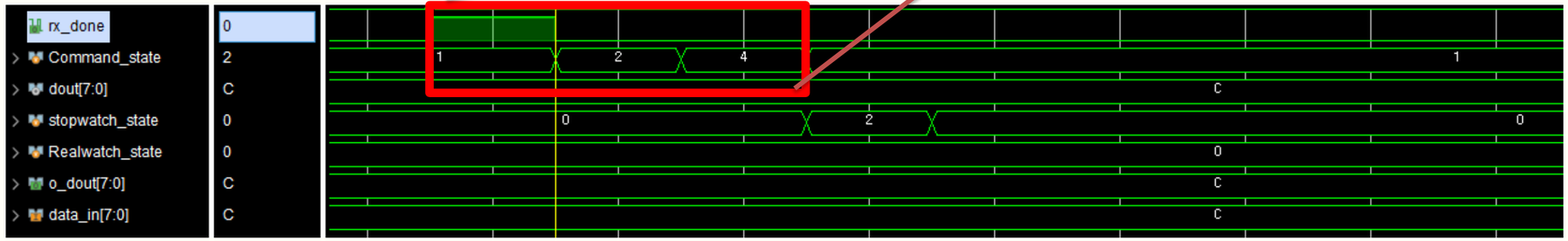




Rx_done이 들어왔을 때, (S)

IDLE -> PROCESS -> STOP_STATE
1 -> 2 -> 5
Stop watch의 state가 1 -> 0로
(run -> stop)





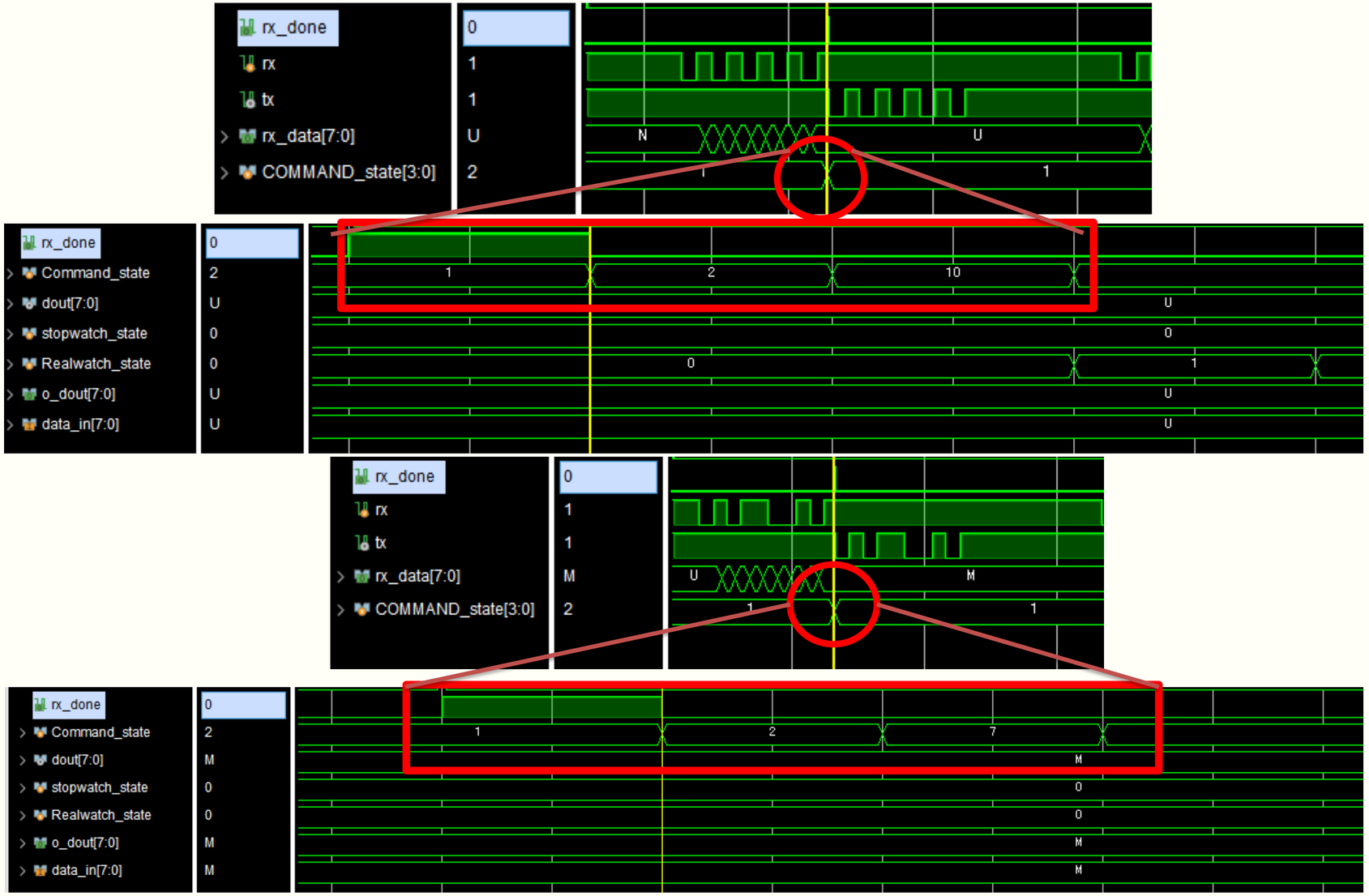
Rx_done이 들어왔을 때, (C)

IDLE -> PROCESS -> CLEAR_STATE
1 -> 2 -> 4
Stop watch의 state가 0- > 2로
(stop -> clear)

SIMULATION

Sending U, M, L, D

```
parameter IDLE = 4'd1, PROCESS = 4'd2, RUN_STATE = 4'd3, CLEAR_STATE = 4'd4, STOP_STATE = 4'd5,
          WATCH_MODE_STATE = 4'd6, MODE_STATE = 4'd7, RESET_STATE = 4'd8,
          R_STATE = 4'd9, U_STATE = 4'd10, D_STATE = 4'd11, L_STATE = 4'd12, WAIT = 4'd13;

parameter IDLE = 3'b000, UP = 3'b001, DOWN = 3'b010, MOVE_LEFT = 3'b011, MOVE_RIGHT = 3'b100;
reg [2:0] n_state, c_state;
```



Rx_done이 들어왔을 때, (U)

IDLE -> PROCESS -> U_STATE
1 -> 2 -> 10
REAL watch의 state가 0->1로
(IDLE->Up)

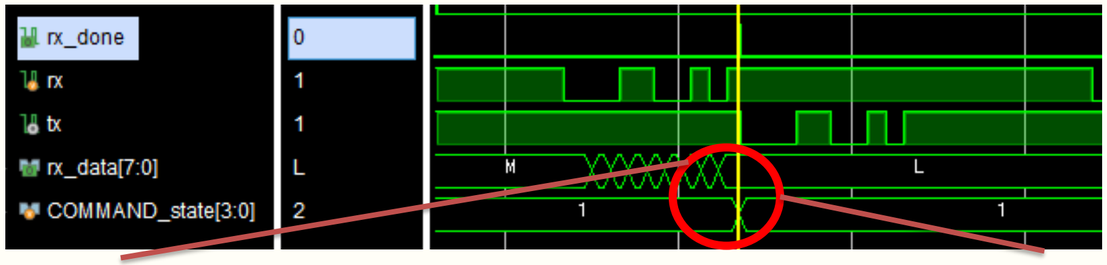
Rx_done이 들어왔을 때, (M)

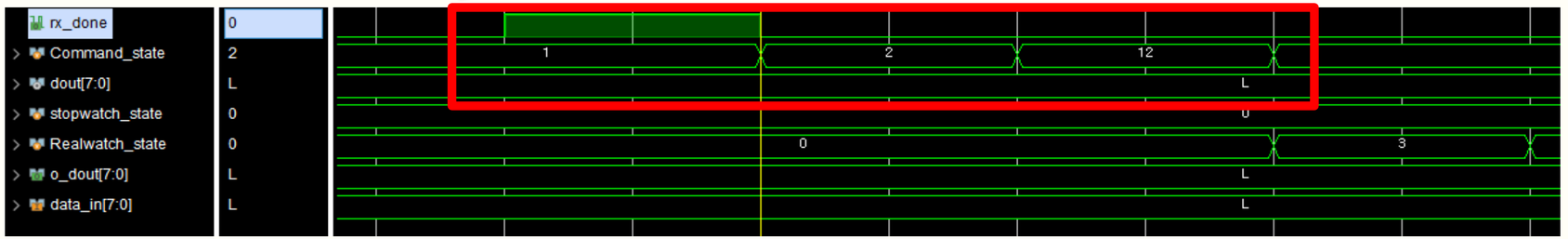
IDLE -> PROCESS -> MODE_STATE
1 -> 2 -> 7
MIN-Hour 모드로 전환

SIMULATION

Sending U, M, L, D

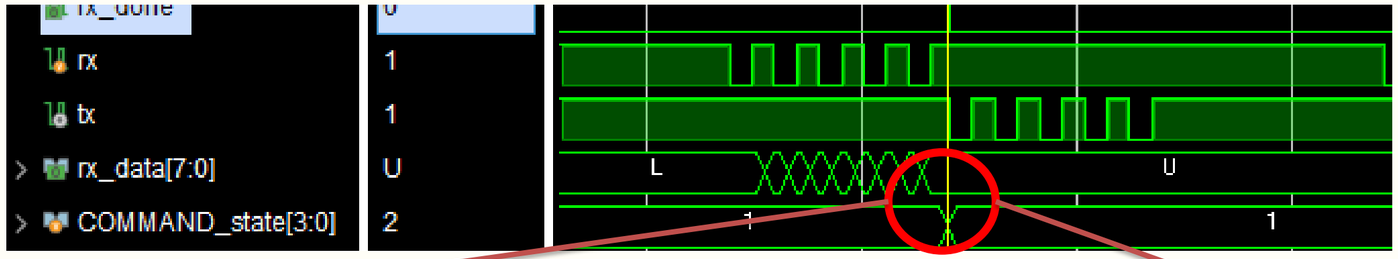
```
parameter IDLE = 4'd1, PROCESS = 4'd2, RUN_STATE = 4'd3, CLEAR_STATE = 4'd4, STOP_STATE = 4'd5,
          WATCH_MODE_STATE = 4'd6, MODE_STATE = 4'd7, RESET_STATE = 4'd8,
          R_STATE = 4'd9, U_STATE = 4'd10, D_STATE = 4'd11, L_STATE = 4'd12, WAIT = 4'd13;

parameter IDLE = 3'b000, UP = 3'b001, DOWN = 3'b010, MOVE_LEFT = 3'b011, MOVE_RIGHT = 3'b100;
reg [2:0] n_state, c_state;
```

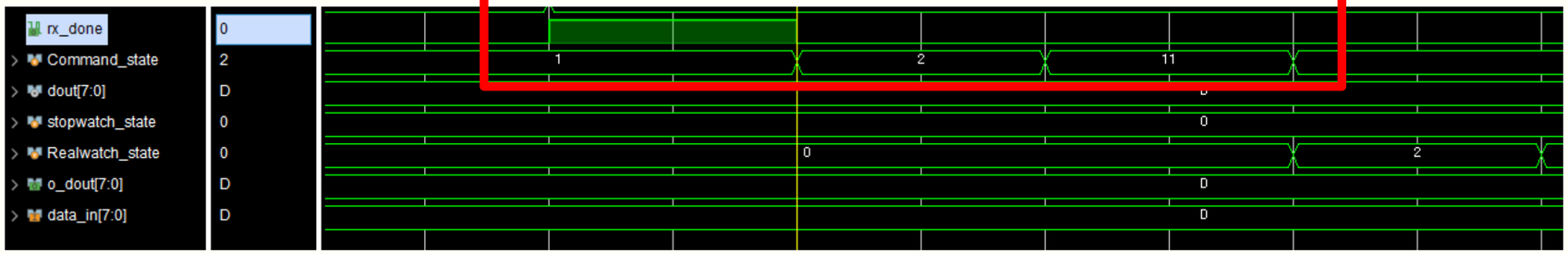




Rx_done이 들어왔을 때, (L)

IDLE -> PROCESS -> L_STATE
1 -> 2 -> 12
Real watch의 state가 0 -> 3로
(IDLE->Move_Left)





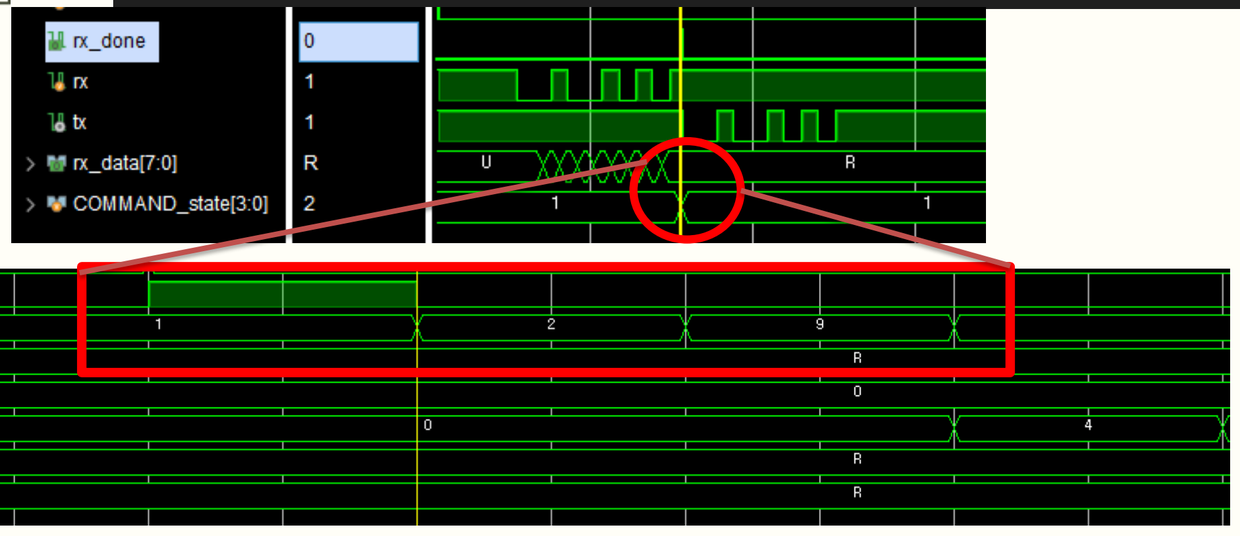
Rx_done이 들어왔을 때, (D)

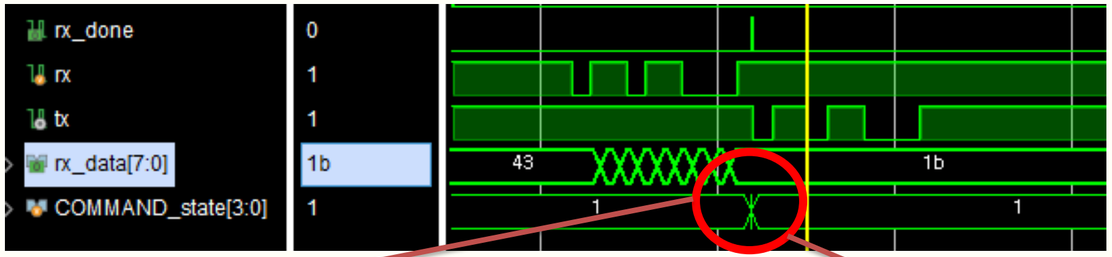
IDLE -> PROCESS -> D_STATE
1 -> 2 -> 1b
Real watch의 state가 0 -> 2로
(IDLE-> DOWN)

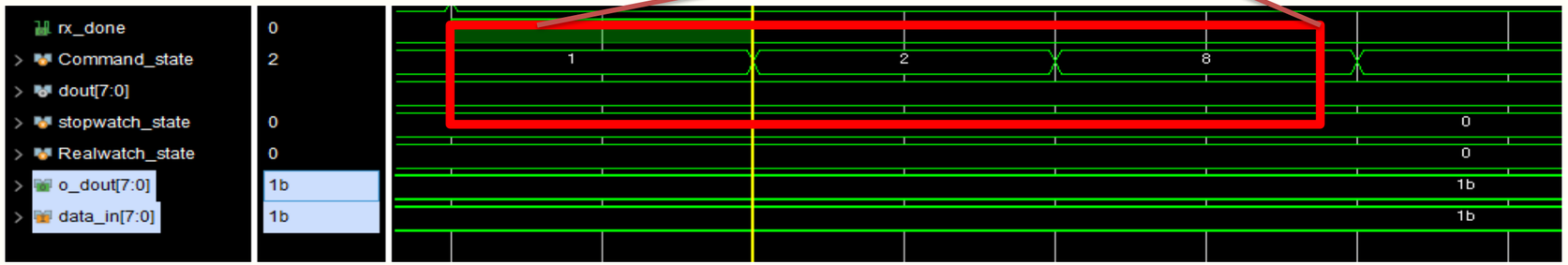
SIMULATION

Sending R, esc

```
parameter IDLE = 4'd1, PROCESS = 4'd2, RUN_STATE = 4'd3, CLEAR_STATE = 4'd4, STOP_STATE = 4'd5,
          WATCH_MODE_STATE = 4'd6, MODE_STATE = 4'd7, RESET_STATE = 4'd8,
          R_STATE = 4'd9, U_STATE = 4'd10, D_STATE = 4'd11, L_STATE = 4'd12, WAIT = 4'd13;
```







Rx_done이 들어왔을 때, (R)

IDLE -> PROCESS -> R_STATE
1 -> 2 -> 9
Real watch의 state 가 0 -> 4
(IDLE->Move_right)

Rx_done이 들어왔을 때, (ecc -> 8h'1b)

IDLE -> PROCESS -> RESET_STATE
1 -> 2 -> 8
Reset

고찰

- TX, RX 구조 설계를 통한 UART 통신 방식에서 SIPO, PISO 구조를 이해
- ASM 및 FSM 구조를 설계하며 상태 기반으로 처리하고, 전이 와 조건 분기 설계 능력을 통해 설계도를 보며 모듈을 제작하는 능력 향상
- 중첩 입력에 대한 우선순위를 정하여 문제를 해결하고 신호 처리 방법 적용을 통해 실시간 제어 가능(UART 및 물리 둘 다 동작)
- 시뮬레이션을 통한 디버깅 및 동작 여부를 미리 확인하며 Rx_done 타이밍 기반으로 상태 변화 검증
- 단순 모듈 단위를 넘어 UART 통신 + FSM 명령구조 + STOPWATCH 및 RealWatch를 통해 통합 시스템을 개발함

느낀 점

모듈을 수정하며 통합 시스템을 구성하였기에 모듈 단위로 구성 시 수정의 용이함을 더욱 크게 느낌

 시뮬레이션으로 디버깅 역량을 통해 문제가 되는 상황을 체크하는 능력이 향상됨

모듈 및 wire 등의 변수 이름을 잘 설정해야 헷갈리지 않게 연결할 수 있다.

아쉬운 점

물리 스위치는 상태가 유지되기에 스위치를 올리면 UART로 제어가 되지 않았는데, 우선순위 처리를 통해 처리하지 못함

남는 SW를 이용하여 다양한 기능을 구현하지 못한 점이 아쉽다 ex) 타이머