UART PROJECT

윤종민, 권혁진

개요

- UART와 STOPWATCH를 연결하여 STOPWATCH 제어를 외부 입력을 통해서도 가능하게 하기 위해 제작

- 외부 입력이 rx, tx를 통과하여 control unit을 통해 기존 stopwatch에 신호를 전달 버튼과 스위치를 통한 제어와 동일한 제어 가능

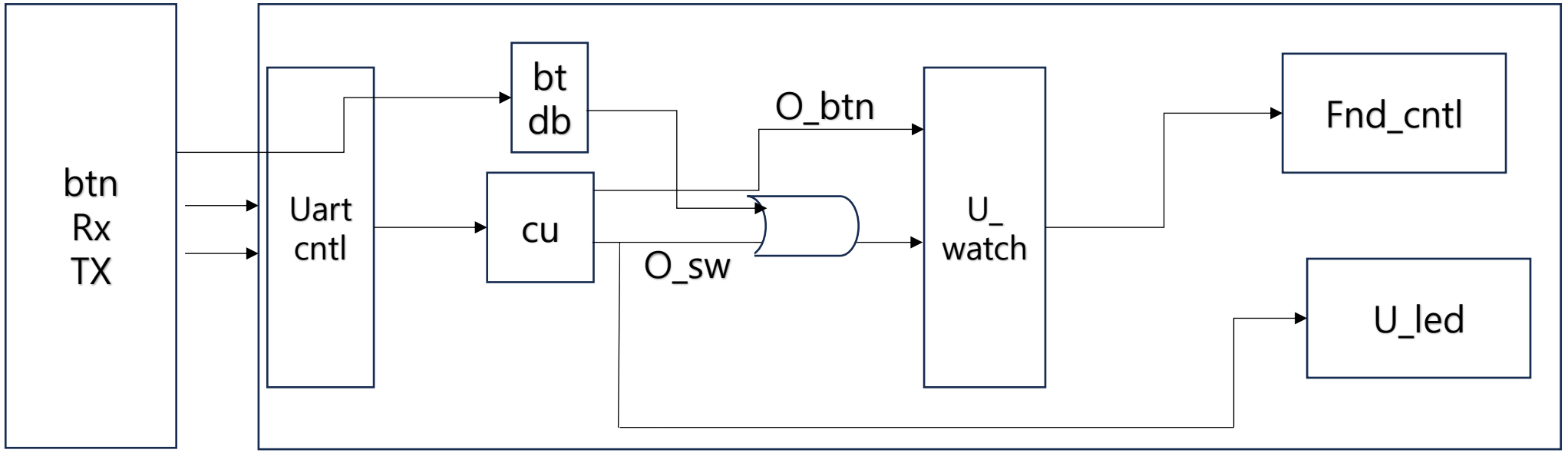
목차

- Block diagram, 부품 스펙
- Asm & Fsm
- 코드, 기능
- Simulation, Signal
- 동작 영상
- 마무리

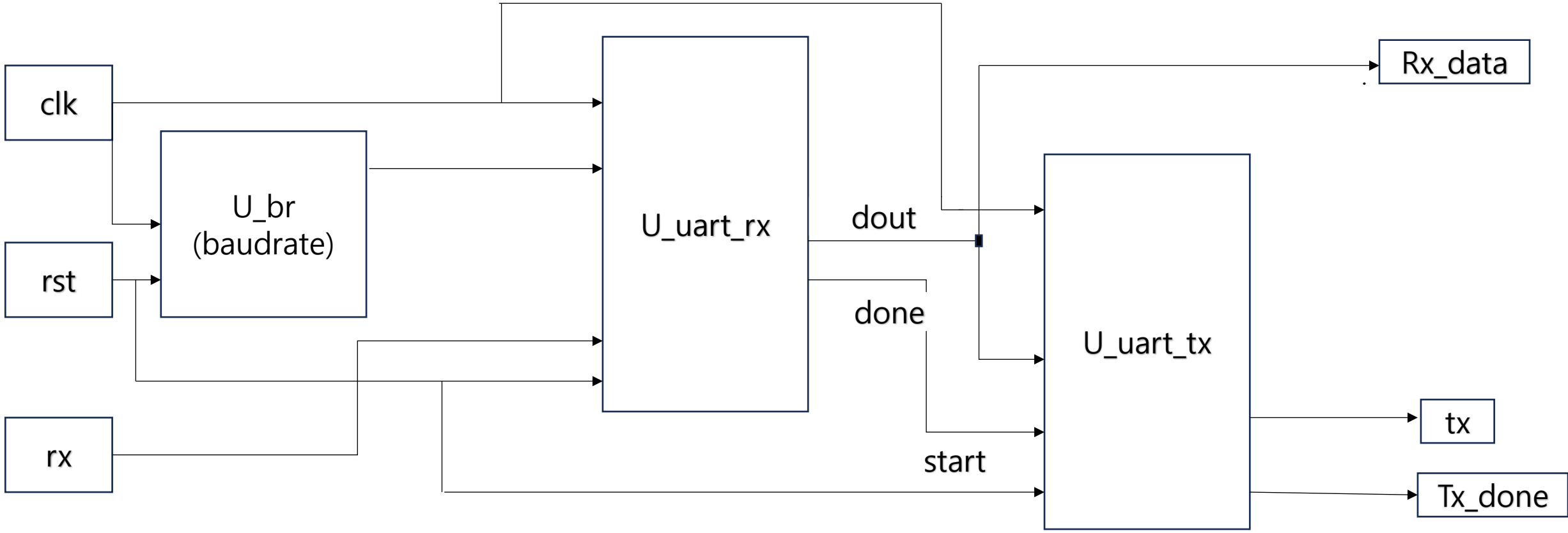
부품 스펙

- 16 user switches
- 16 user leds
- 5 user pushbuttons
- 4 digit 7-segment display

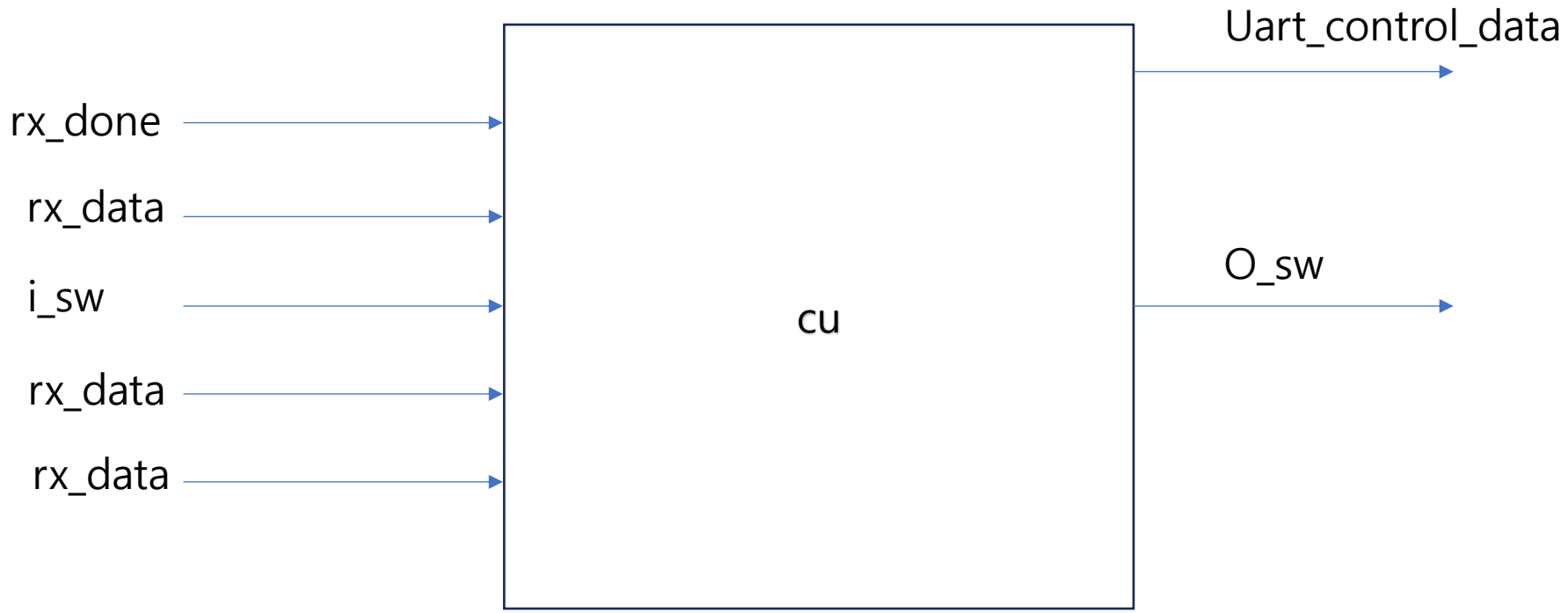
전체 block diagram

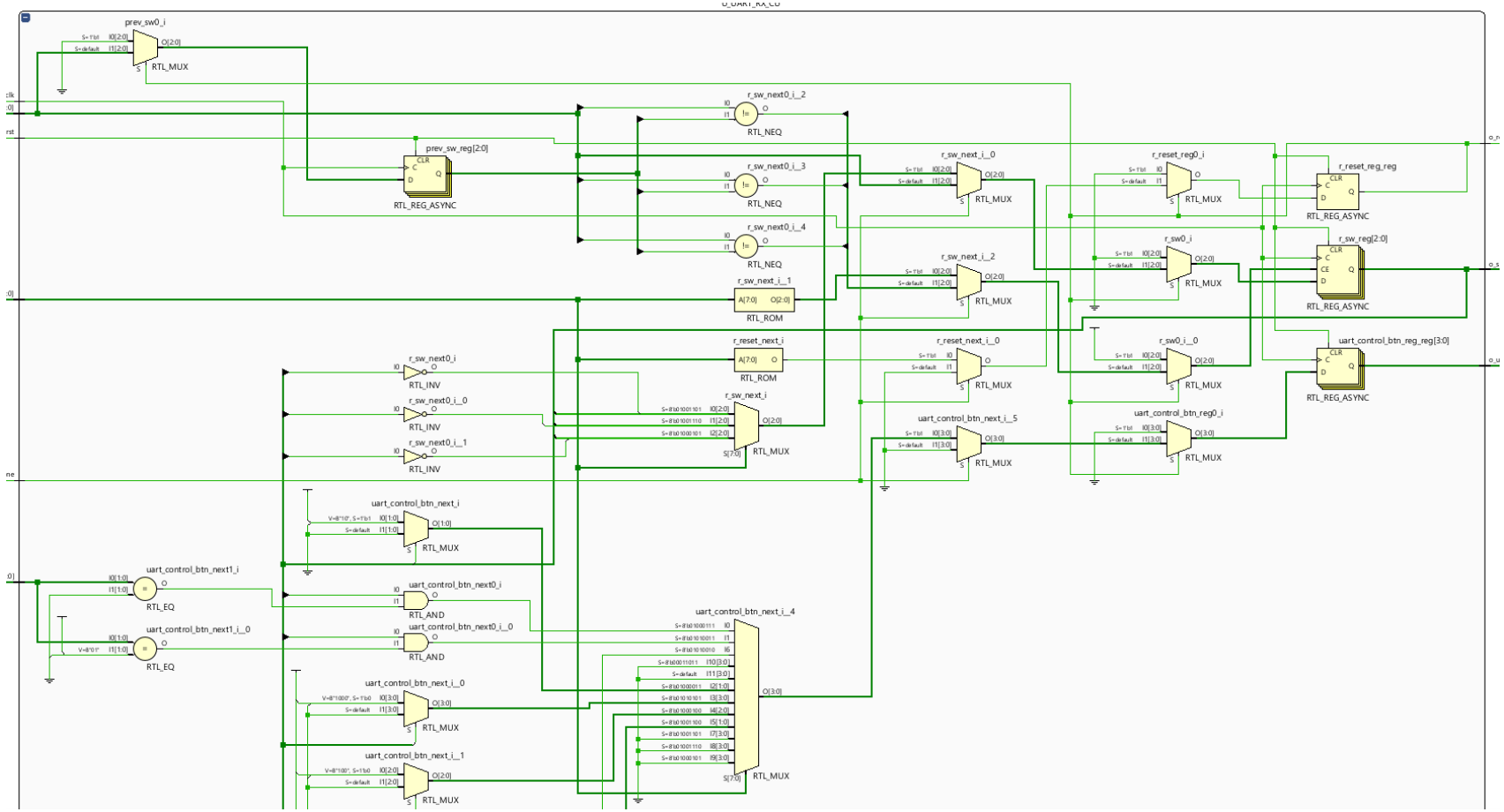


부분 diagram – uart_controller

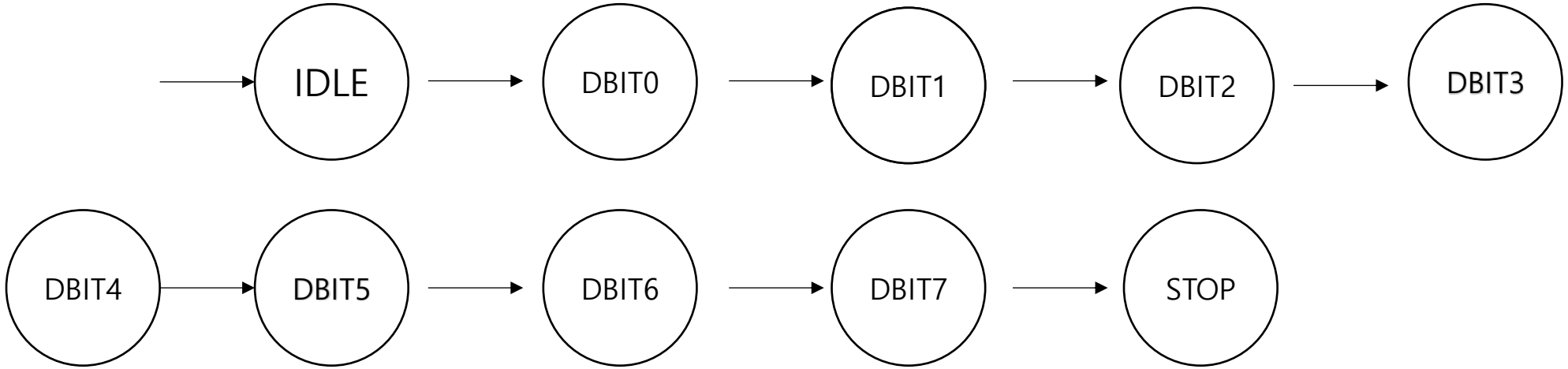


부분 diagram - cu

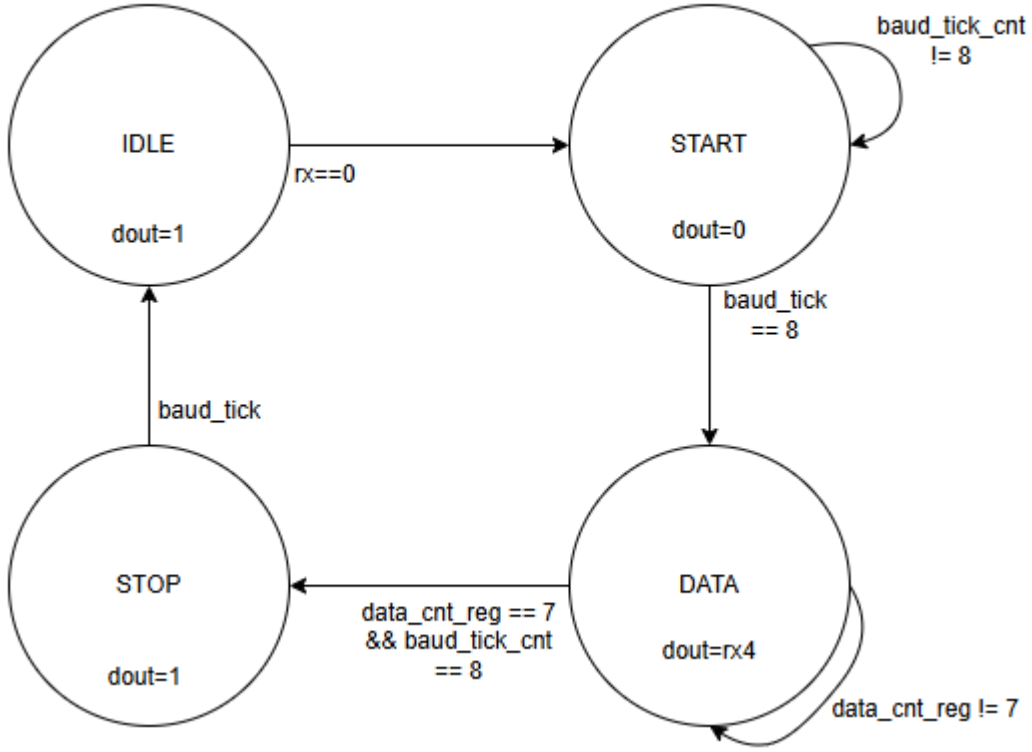



prev_sw0_i
RTL_MUX
r_sw_next0_i_2
RTL_NEQ
prev_sw_reg[2:0]
RTL_REG_ASYNC
r_sw_next0_i_3
RTL_NEQ
r_sw_next0_i_4
RTL_NEQ
r_sw_next_i__0
RTL_MUX
r_reset_reg0_i
RTL_MUX
r_reset_reg_reg
RTL_REG_ASYNC
r_sw0_i
RTL_MUX
r_sw_reg[2:0]
RTL_REG_ASYNC
r_sw_next_i__2
RTL_MUX
r_sw_next_i__1
RTL_ROM
r_reset_next_i
RTL_ROM
r_reset_next_i__0
RTL_MUX
r_sw0_i__0
RTL_MUX
uart_control_btn_reg_reg[3:0]
RTL_REG_ASYNC
r_sw_next0_i
RTL_INV
r_sw_next0_i__0
RTL_INV
r_sw_next0_i__1
RTL_INV
r_sw_next_i
RTL_MUX
uart_control_btn_next_i__5
RTL_MUX
uart_control_btn_reg0_i
RTL_MUX
uart_control_btn_next_i
RTL_MUX
uart_control_btn_next1_i
RTL_EQ
uart_control_btn_next0_i
RTL_AND
uart_control_btn_next1_i__0
RTL_EQ
uart_control_btn_next0_i__0
RTL_AND
uart_control_btn_next_i__4
RTL_MUX
uart_control_btn_next_i__0
RTL_MUX
uart_control_btn_next_i__1
RTL_MUX

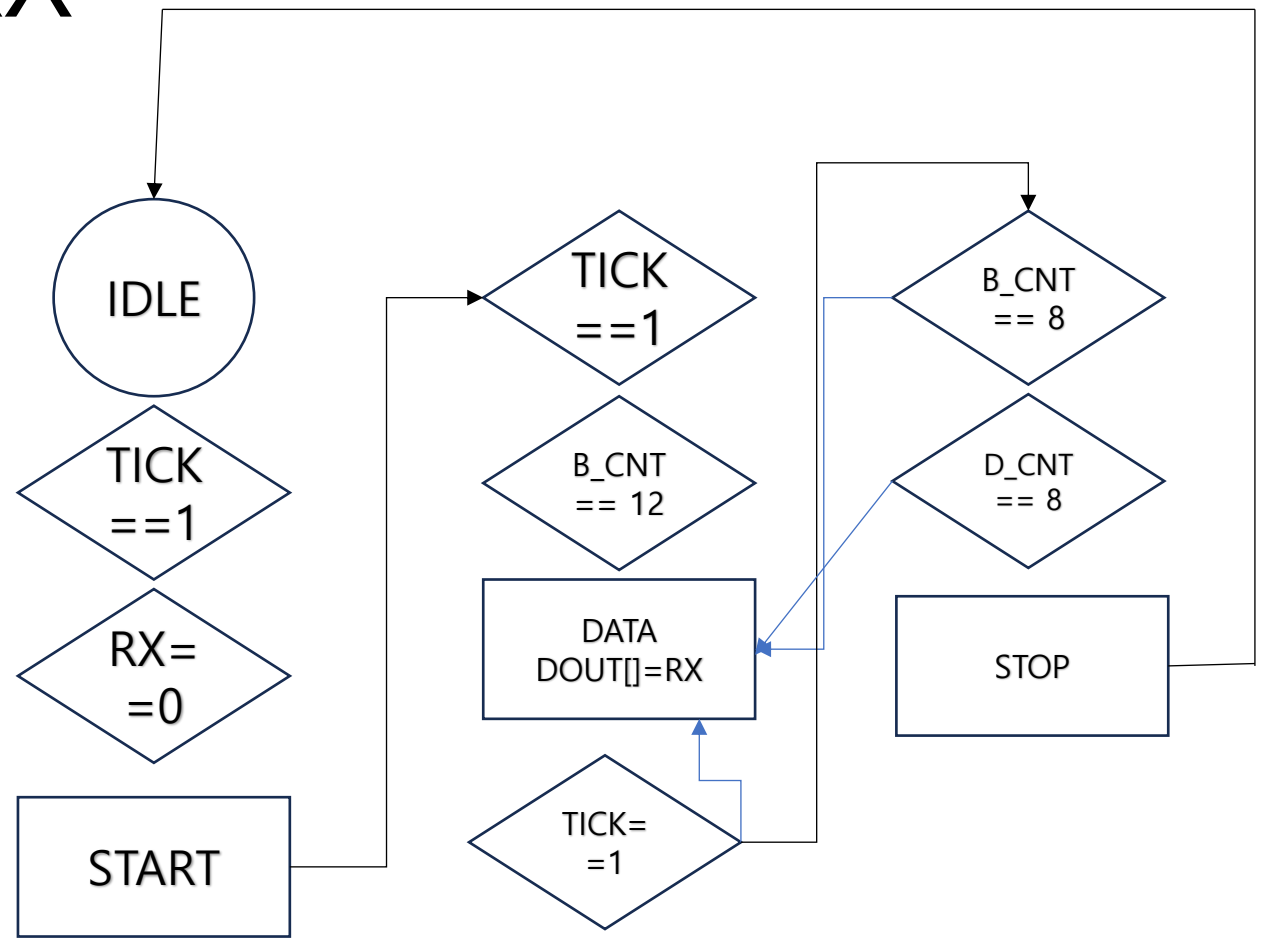
FSM-TX



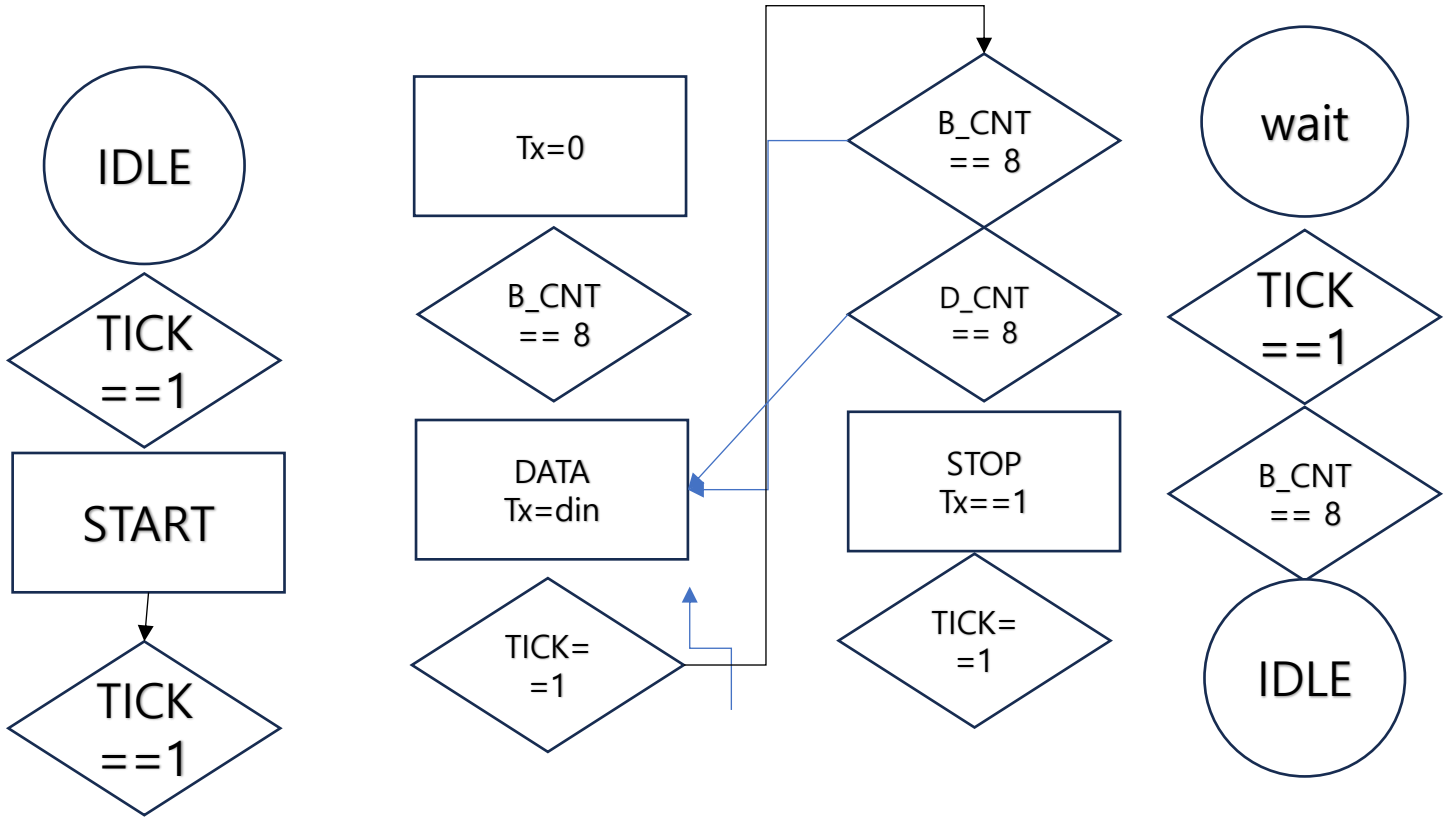
FSM-RX


IDLE
dout=1
rx==0
START
dout=0
baud_tick_cnt
!= 8
baud_tick
== 8
DATA
dout=rx4
data_cnt_reg != 7
data_cnt_reg == 7
&& baud_tick_cnt
== 8
STOP
dout=1
baud_tick

ASM - RX

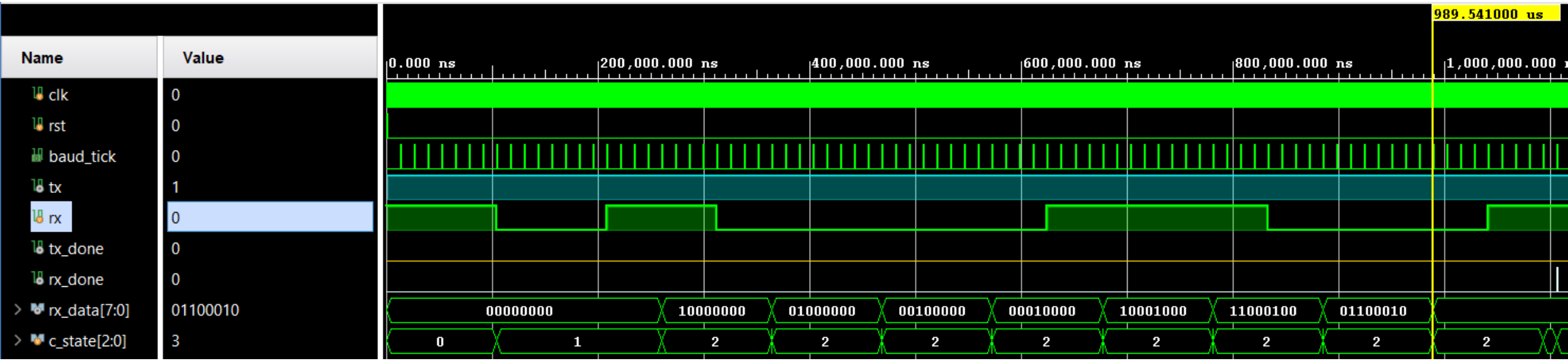

IDLE
TICK ==1
RX= =0
START
TICK ==1
B_CNT == 12
DATA DOUT[]=RX
TICK= =1
B_CNT == 8
D_CNT == 8
STOP

ASM - TX

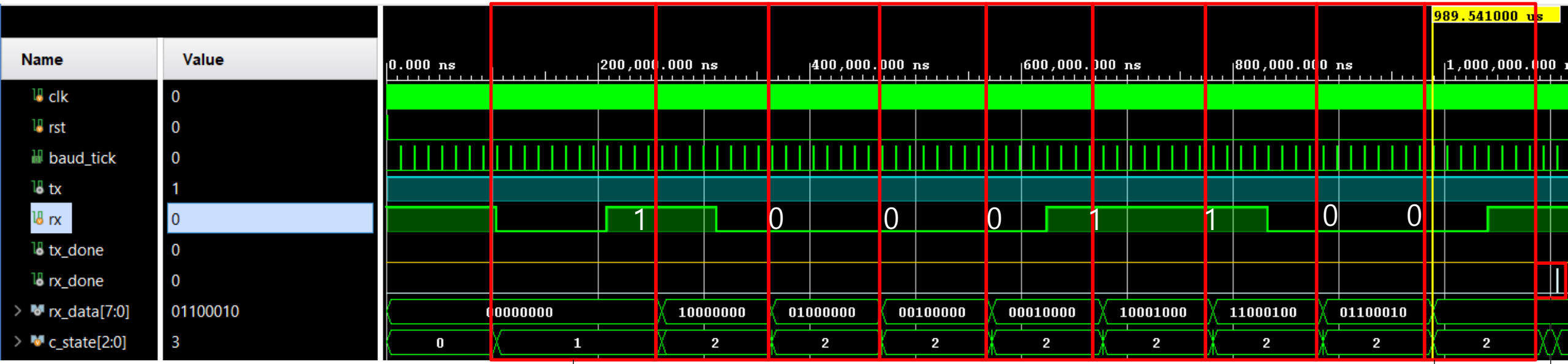

IDLE
TICK
==1
START
TICK
==1
Tx=0
B_CNT
== 8
DATA
Tx=din
TICK=
=1
B_CNT
== 8
D_CNT
== 8
STOP
Tx==1
TICK=
=1
wait
TICK
==1
B_CNT
== 8
IDLE

simulation

Uart rx


Name
Value
clk
0
rst
0
baud_tick
0
tx
1
rx
0
tx_done
0
rx_done
0
rx_data[7:0]
01100010
c_state[2:0]
3
989.541000 us
0.000 ns
200,000.000 ns
400,000.000 ns
600,000.000 ns
800,000.000 ns
1,000,000.000
00000000
10000000
01000000
00100000
00010000
10001000
11000100
01100010
0
1
2
2
2
2
2
2
2
2

Uart rx

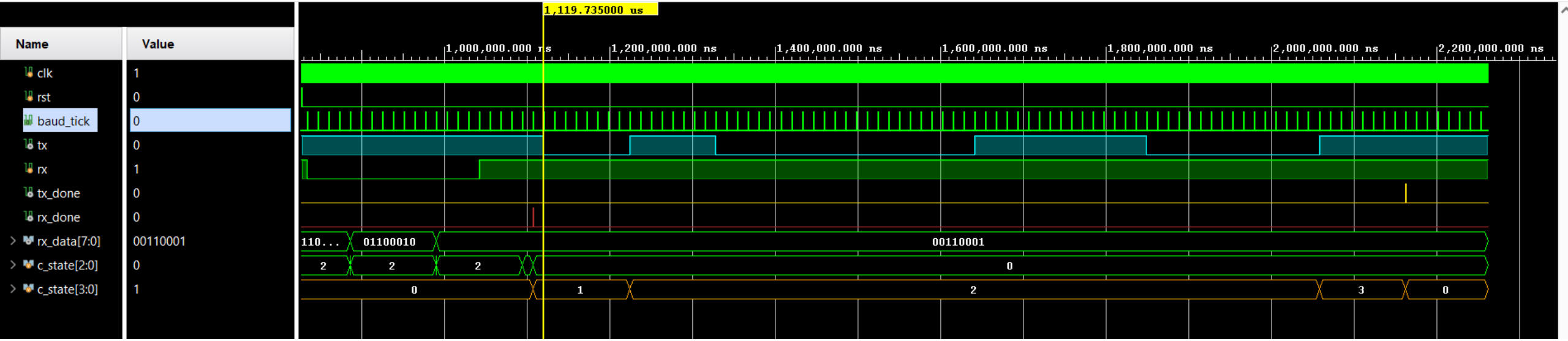


12baud_tick 후 read 시작

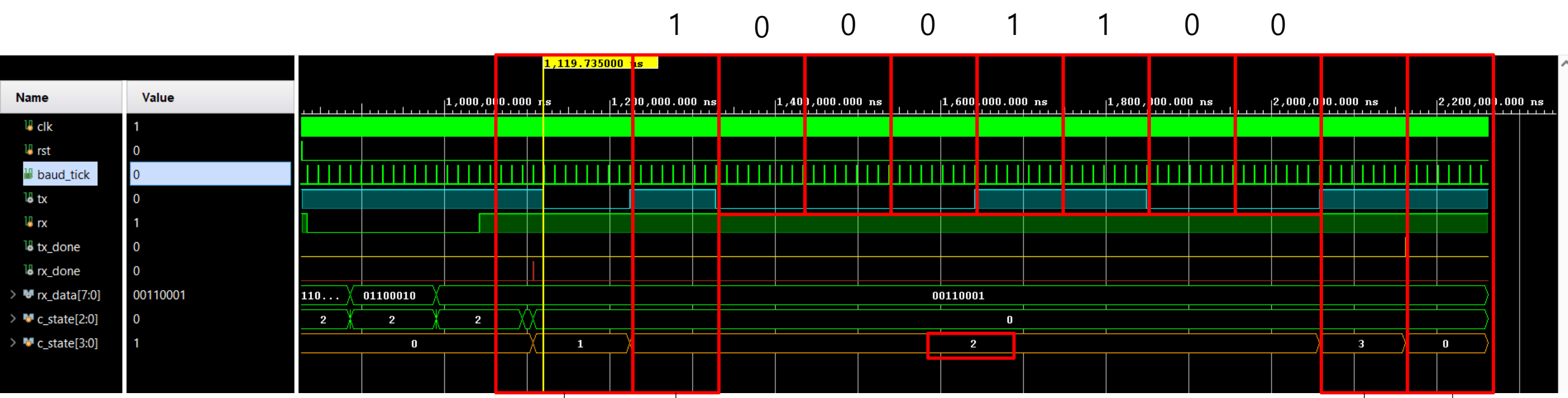
Read_data

Rx_done

Uart tx


Name
Value
clk
1
rst
0
baud_tick
0
tx
0
rx
1
tx_done
0
rx_done
0
rx_data[7:0]
00110001
c_state[2:0]
0
c_state[3:0]
1
1,119.735000 us
1,000,000.000 ns
1,200,000.000 ns
1,400,000.000 ns
1,600,000.000 ns
1,800,000.000 ns
2,000,000.000 ns
2,200,000.000 ns
110...
01100010
00110001
2
2
2
0
0
1
2
3
0

Uart tx

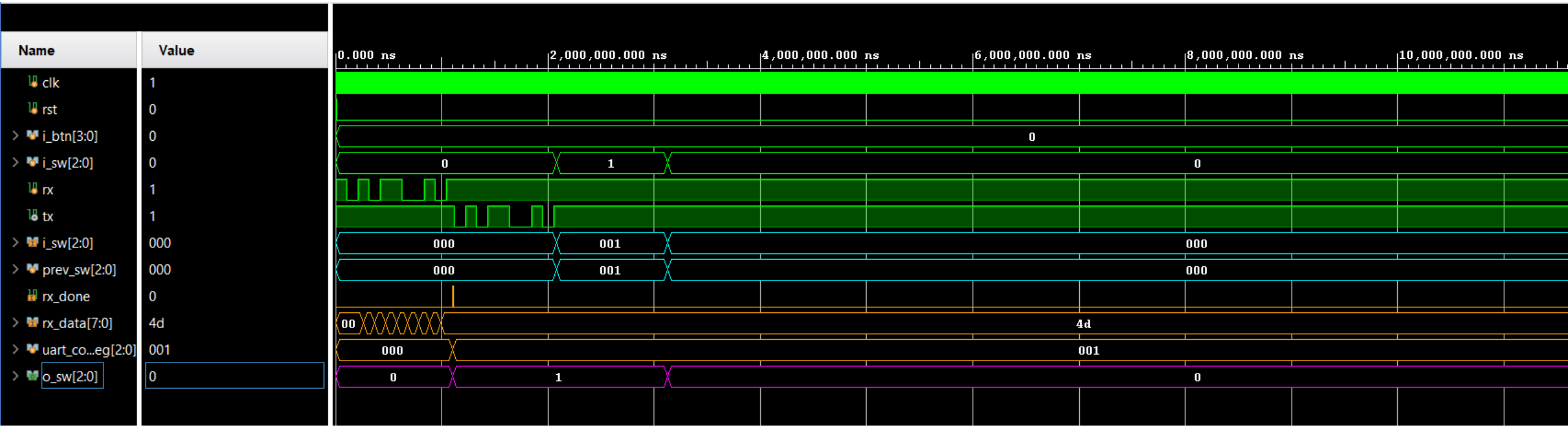


전송시작

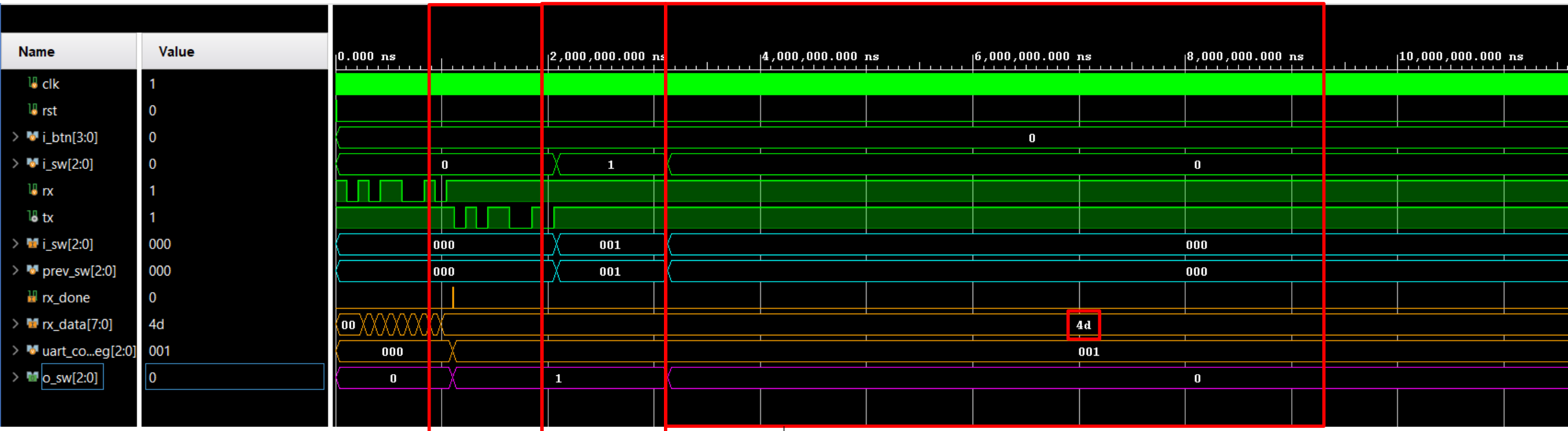
Stop bit 전송 끝

rx_done신호를 받고
한 baud_tick 후에 start

Watch,stopwatch로 들어가는 cu



Watch,stopwatch로 들어가는 cu

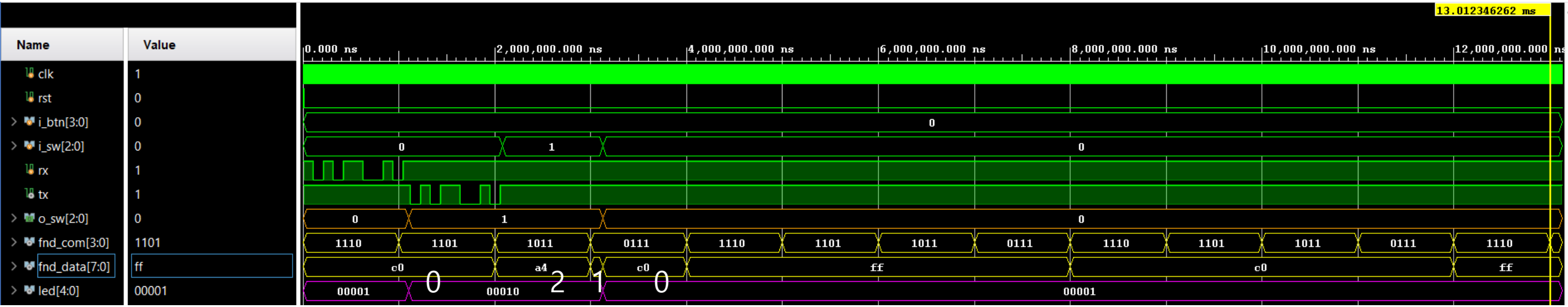


i_sw[0] = 0, 마지막 변화 -> 0이니 o_sw값 변화

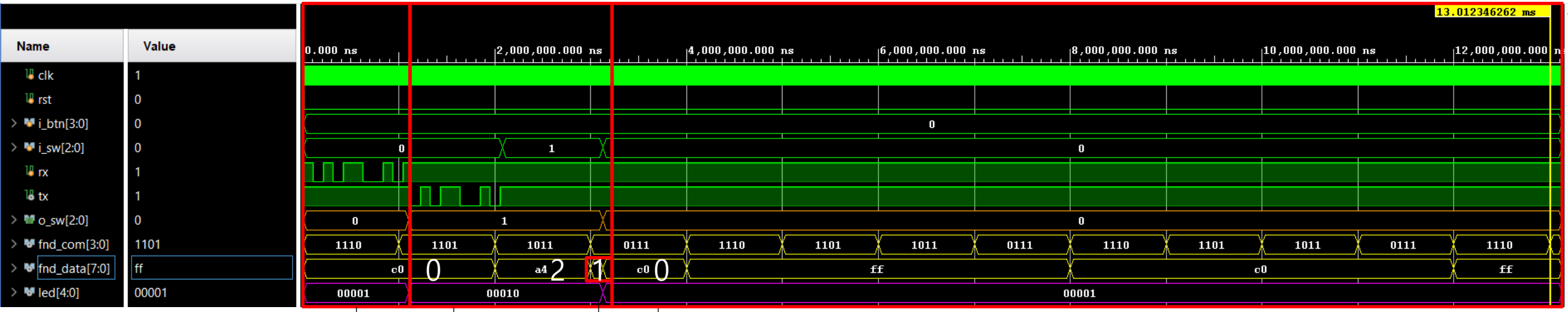
i_sw[0] = 1, 마지막 변화 -> 1이니 o_sw값 유지

Uart 입력, 'M'(0x4d)

FND, LED


Name
Value
clk
1
rst
0
i_btn[3:0]
0
i_sw[2:0]
0
rx
1
tx
1
o_sw[2:0]
0
fnd_com[3:0]
1101
fnd_data[7:0]
ff
led[4:0]
00001
13.012346262 ms
0.000 ns
2,000,000.000 ns
4,000,000.000 ns
6,000,000.000 ns
8,000,000.000 ns
10,000,000.000 ns
12,000,000.000 ns
1110
1101
1011
0111
c0
a4
ff
00001
00010
0
2
1
0

FND, LED



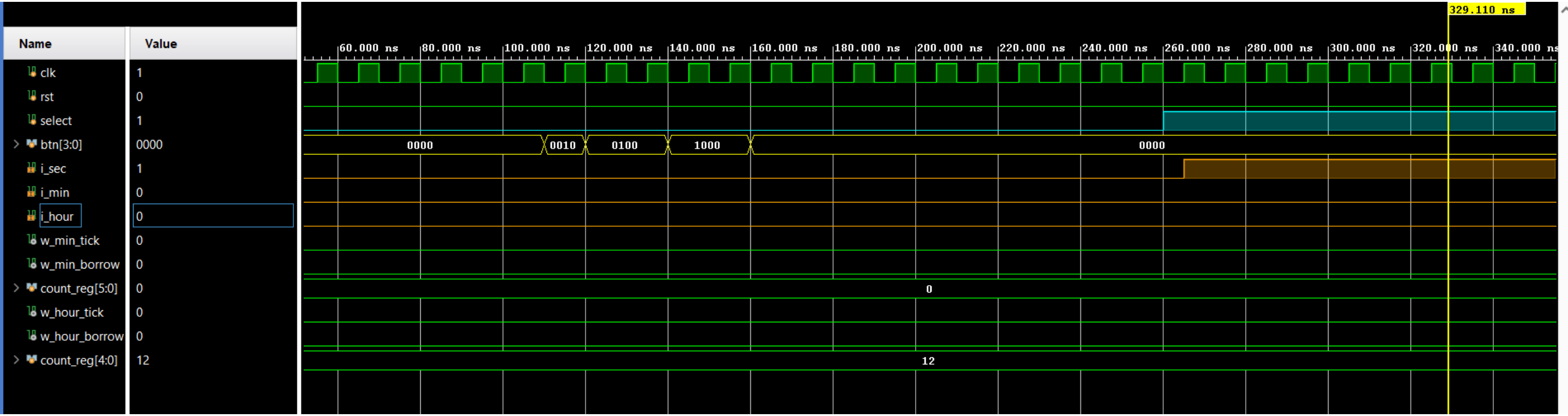
Msec,sec 출력(led00001)

Min,hour출력에서 sw변동으로 fnd값이 바로 변화

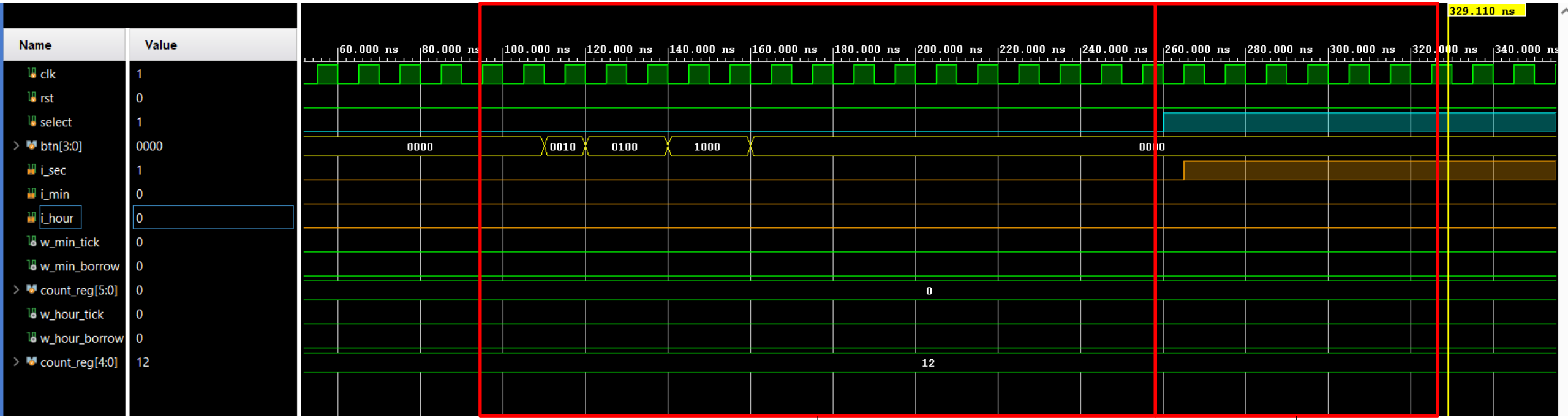
Min,hour 출력(led00010)

Msec,sec 출력(led00001)

Watch에서 en이 high일때만 값 변동



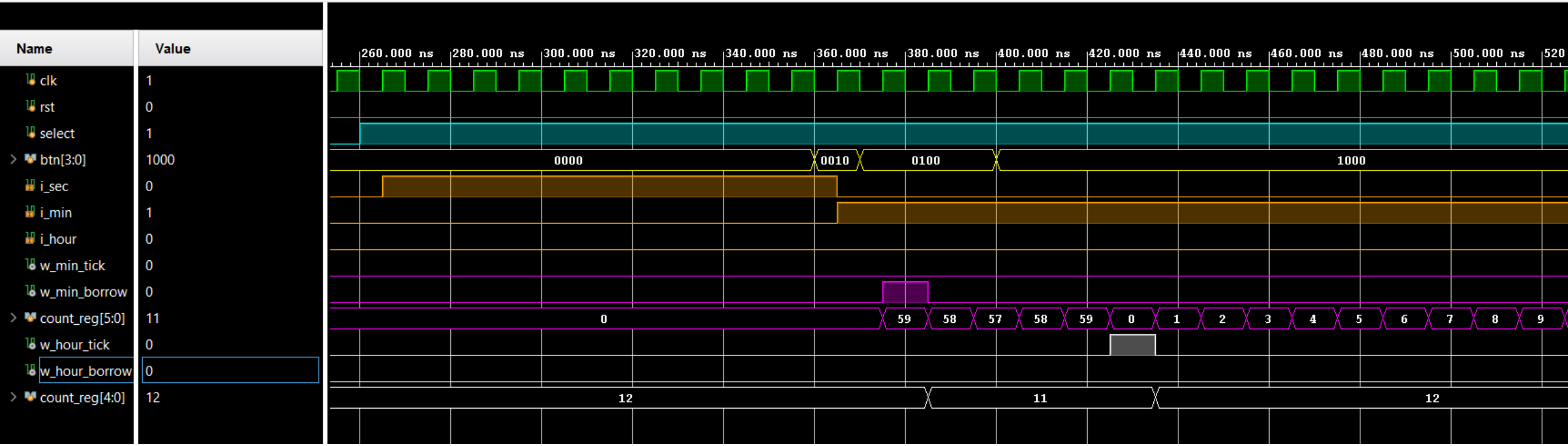
Watch에서 en이 high일때만 값 변동



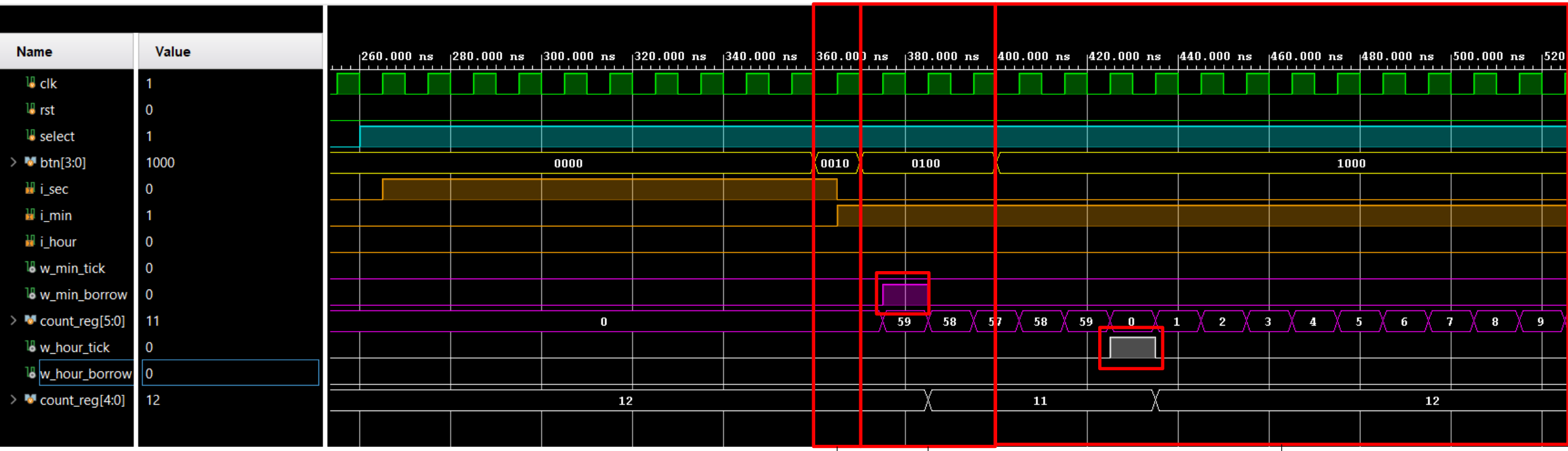
Select가 0이므로 sec, min, hour선택 안됨

Select가 1이므로 sec선택

Watch에서 값 올림, 내림



Watch에서 값 올림, 내림

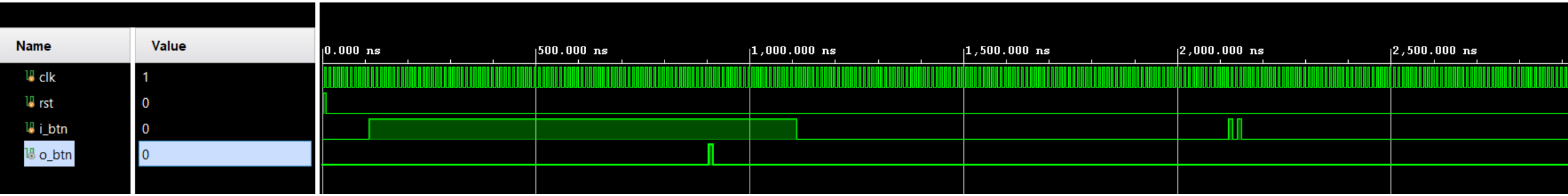


btnU 눌러서 값 증가, 올림 반영

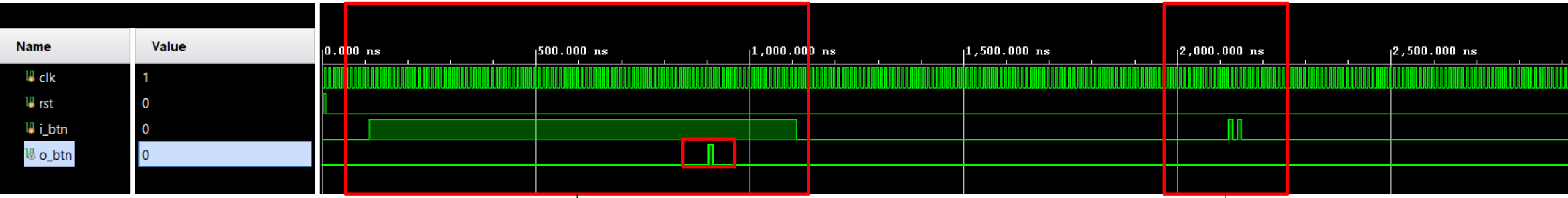
btnD눌러서 값 내림, 내림 반영

btnL 누름, 분 선택

Btn debouncing



Btn debouncing



i_btn이 일정이상 1이 들어와야 한 tick 발생

Noise는 감지 안함

code

Watch cu

```
SELECT_MIN: begin
    if (!i_select) begin
        n_state = RUN;
    end else if (i_btn == 2'b10) begin  //left
        n_state = SELECT_HOUR;
    end else if (i_btn == 2'b01) begin  //right
        n_state = SELECT_SEC;
    end else begin
        n_state = c_state;
    end
end
```

FSM에서 i_select(sw[2])가 1일 때만
State 이동

msec, sec, min, hour 모두 같은 구조

Watch dp

```
if (i_tick == 1'b1) begin
    if (count_reg == TICK_COUNT - 1) begin
        count_next  = 0;
        r_tick_next = 1'b1;
    end else begin
        count_next  = count_reg + 1;
        r_tick_next = 1'b0;
    end
end else if (i_select) begin
    case (i_up_down)
        2'b10: begin  //up
            if (count_reg == TICK_COUNT - 1) begin
                count_next  = 0;
                r_tick_next = 1'b1;
            end else begin
                count_next  = count_reg + 1;
                r_tick_next = 1'b0;
            end
        end
        2'b01: begin  //down
            if (count_reg == 0) begin
                count_next = TICK_COUNT - 1;
                r_borrow_next = 1'b1;
            end else begin
                count_next = count_reg - 1;
                r_borrow_next = 1'b0;
            end
        end
    endcase
end else if (i_borrow) begin
    if (count_reg == 0) begin
        count_next = TICK_COUNT - 1;
        r_borrow_next = 1'b1;
    end else begin
        count_next = count_reg - 1;
        r_borrow_next = 1'b0;
    end
end
```

Select(cu에서 오는 신호)에 의해 선택될 시 btn값으로 up,down가능

값이 0이나 최대값을 벗어나면 tick과 Borrow 발생 (올림 또는 내림 발생)

다른 모듈에서 올림이나 내림 발생시 (i_tick과 i_borrow) 값을 내리거나 올림

U_UART_RX_CU(전체 block도 상 cu)

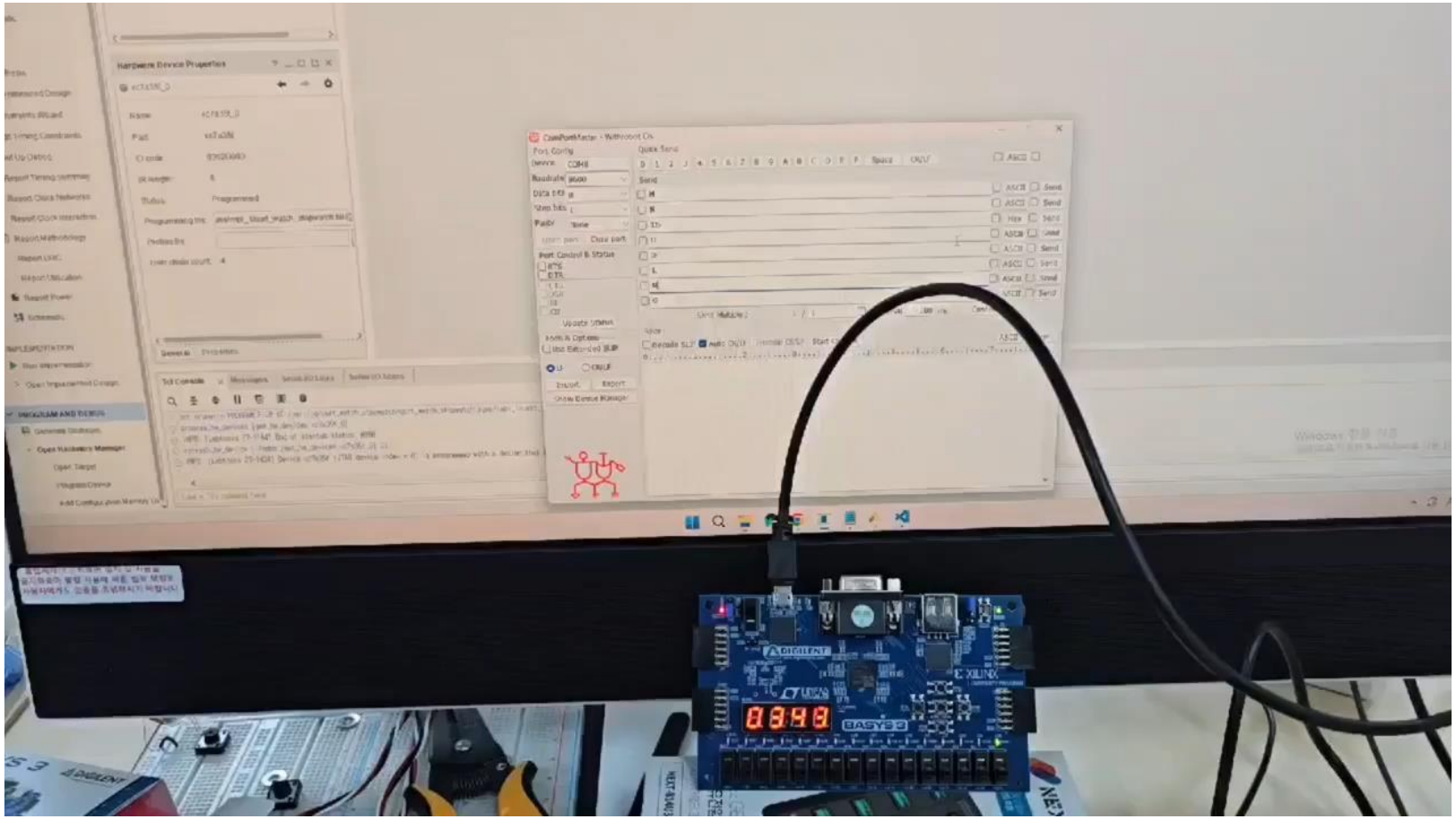
```
if (rx_done) begin
    case (rx_data)
        // stopwatch control
        8'h47: if (r_sw[1] && s_state == STOP) uart_control_btn_next = 4'b0001; // G
        8'h53: if (r_sw[1] && s_state == RUN ) uart_control_btn_next = 4'b0001; // S
        8'h43: if (r_sw[1])                    uart_control_btn_next = 4'b0010; // C

        // watch control
        8'h55: if (!r_sw[1]) uart_control_btn_next = 4'b1000; // U
        8'h44: if (!r_sw[1]) uart_control_btn_next = 4'b0100; // D
        8'h4C: if (!r_sw[1]) uart_control_btn_next = 4'b0010; // L
        8'h52: if (!r_sw[1]) uart_control_btn_next = 4'b0001; // R

        // toggle switches
        8'h4D: begin // M: toggle sw0
            uart_control_sw_next = { r_sw[2:1], ~r_sw[0] };
            r_sw_next = uart_control_sw_next;
        end
        8'h4E: begin // N: toggle sw1
            uart_control_sw_next = { r_sw[2], ~r_sw[1], r_sw[0] };
            r_sw_next = uart_control_sw_next;
        end
        8'h45: begin // E: toggle sw2
            uart_control_sw_next = { ~r_sw[2], r_sw[1:0] };
            r_sw_next = uart_control_sw_next;
        end

        // reset
        8'h1B: r_reset_next = 1'b1; // ESC
    endcase
end else begin
    // i_sw 변화 감지
    r_sw_next[0] = (i_sw[0] != prev_sw[0]) ? i_sw[0] : r_sw[0];
    r_sw_next[1] = (i_sw[1] != prev_sw[1]) ? i_sw[1] : r_sw[1];
    r_sw_next[2] = (i_sw[2] != prev_sw[2]) ? i_sw[2] : r_sw[2];
end
```

rx_done이 1일 경우
btn관련 값이 들어올 시
sw[1]과 stopwatch state를 따져서 output 결정 (btn)

esc값이 들어올 시 reset = 1 (rst)

sw관련 값이 들어올 시
관련 sw값만 반전 (sw)

i_sw값이 변화할 시 (i_sw != prev_sw)
변화한 값만 반전 (sw)


Hardware Device Properties
BASYS 3
DIGILENT
XILINX

Trouble Shooting

- sw0을 올렸을 때 외부 입력을 넣어도 sw0이 1로 유지되고 있기 때문에 fnd가 변하지 않는 case를 해결

- Debounce를 uart에서 한 번 stopwatch에서 한 번 총 2번 적용이 돼있어서 uart로 입력을 해줘도 반응이 없었던 점

- Uart와 watch를 합칠 때 uart로 값을 주어도 watch가 바뀌지 않아서 파형을 하나씩 되짚어가며 uart에서 와이어링이 빠진 것을 확인하여 해결

배운 점

- Verilog를 확장해야 할 경우를 대비하여 xdc파일에서 이름을 일관적으로 작성하는 습관을 가질 필요가 느껴졌다.

- Module, 변수 등 이름을 지을 때 좀더 명확하고 구분이 잘 가도록 지어 사용에 혼동을 줄일 필요를 느꼈다.