DATE. 2025.08.25

VerilogHDL

STOPWATCH / WATCH + UART

김은성

CONTENTS

목차

DATE. 2025.8.25

김은성

STOPWATCH / WATH + UART

Introduction

Introduction

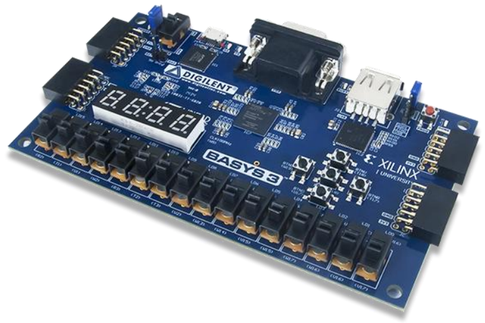
프로젝트 목표

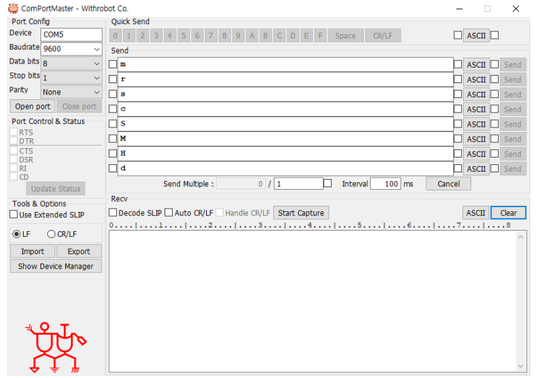
- STOPWATCH & WATCH 시스템 설계 및 구현
- PC와 FPGA 보드 간의 UART 통신 시스템 설계 및 구현

프로젝트 요약

- STOPWATCH & WATCH 시스템 구현하여
  7 segment display에 시간 정보를 시각화
- 하이브리드 인터페이스 구현
  - 보드 위의 버튼 및 스위치로 직접 제어
  - PC에서 UART 터미널을 통해 원격 제어
- STOPWATCH & WATCH MODE 변환 – PC 제어 우선권 부여

Introduction





Vivado 2020.2

AMD-Xilinx의 FPGA 통합 설계 환경(IDE)

Basys 3

FPGA (Field-Programmable Gate Array):
AMD Artix-7 XC7A35T 칩을 사용

COMPOSTMASTER

PC의 시리얼 포트(COM) 통신을 위한
터미널 프로그램

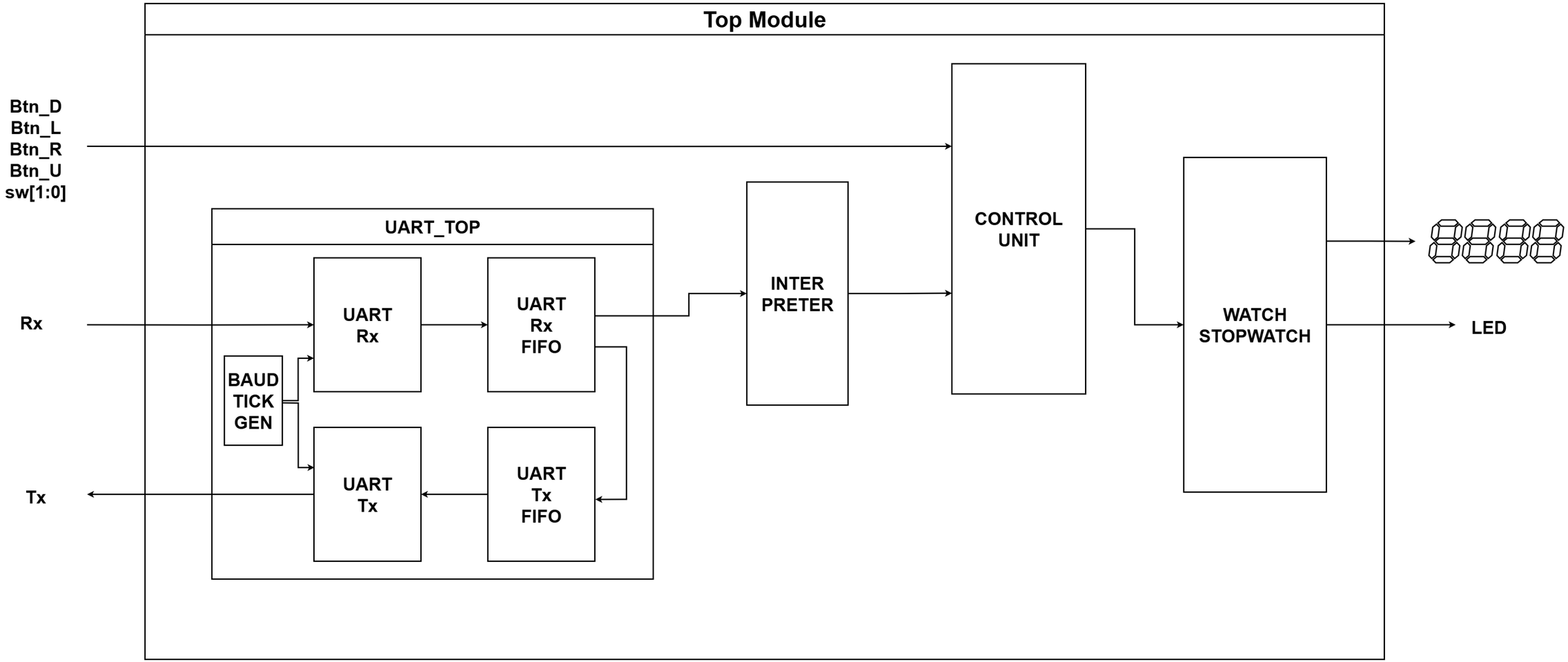
DATE. 2025.8.25

김은성

STOPWATCH / WATH + UART

Block Diagram

Block Diagram



DATE. 2025.8.25

김은성

STOPWATCH / WATH + UART

Code Review

UART BAUD TICK GENERATE

FPGA의 빠른 시스템 클럭(100MHz)과 UART 통신의 느린 속도(9600bps) 사이의 간극을 메워주기 위한 모듈

```
module baud_tick_gen (
    input  clk,
    input  rst,
    output b_tick
);
    // baudrate
    parameter BAUDRATE = 9600 * 16;
    // rx over sampling
    localparam BAUD_COUNT = 100_000_000 / BAUDRATE;
    reg [$clog2(BAUD_COUNT)-1 : 0] counter_reg, counter_next;
    reg tick_reg, tick_next;

    // next CL
    always @(*) begin
        counter_next = counter_reg;
        tick_next = tick_reg;
        if (counter_reg == BAUD_COUNT - 1) begin
            counter_next = 0;
            tick_next = 1'b1;
        end else begin
            counter_next = counter_reg + 1;
            tick_next = 1'b0;
        end
    end
```

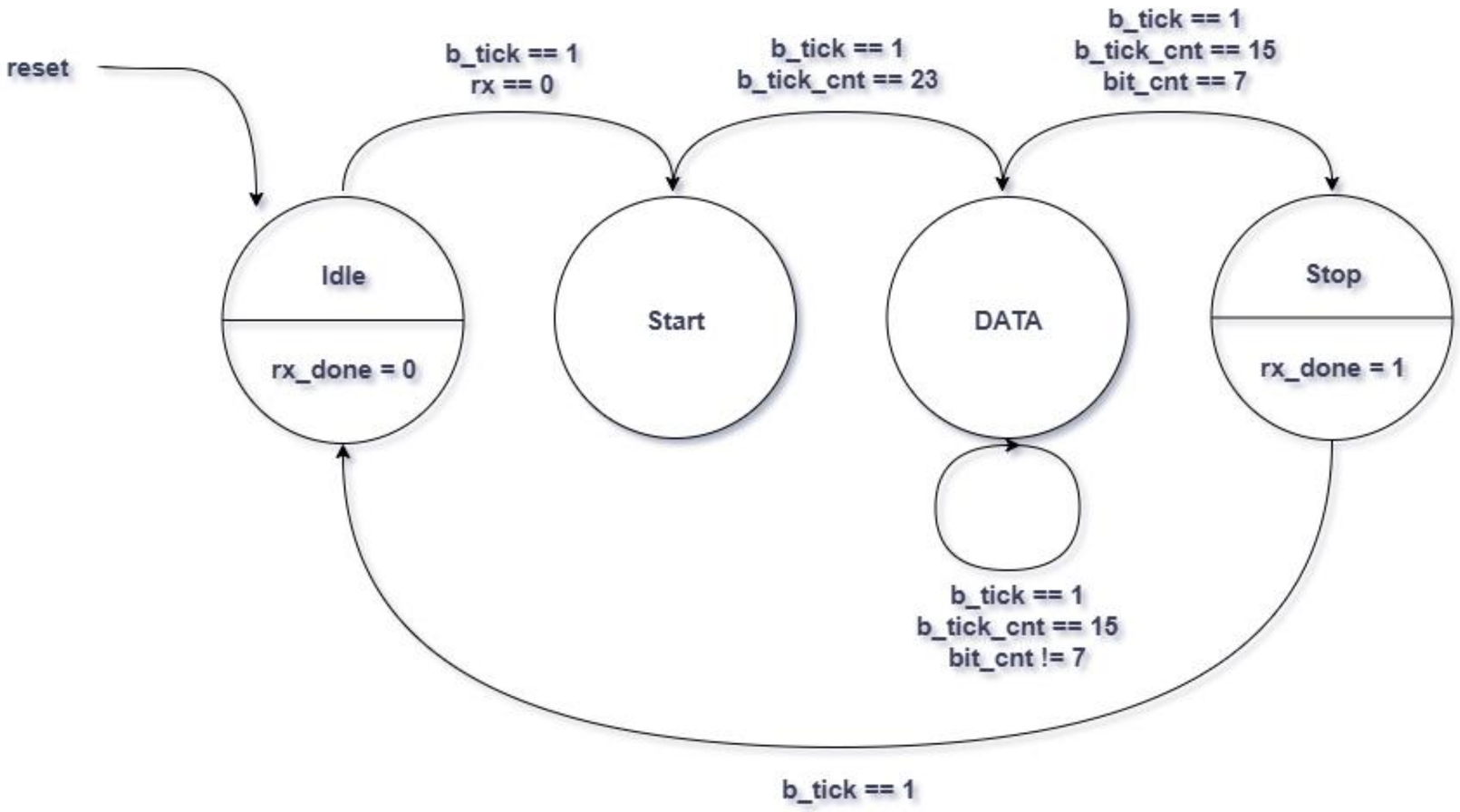
16x Oversampling

UART의 안정성을 높이기 위해
16배 오버샘플링

- 가장 안정적인 중앙 지점 값 읽어오기 위해서

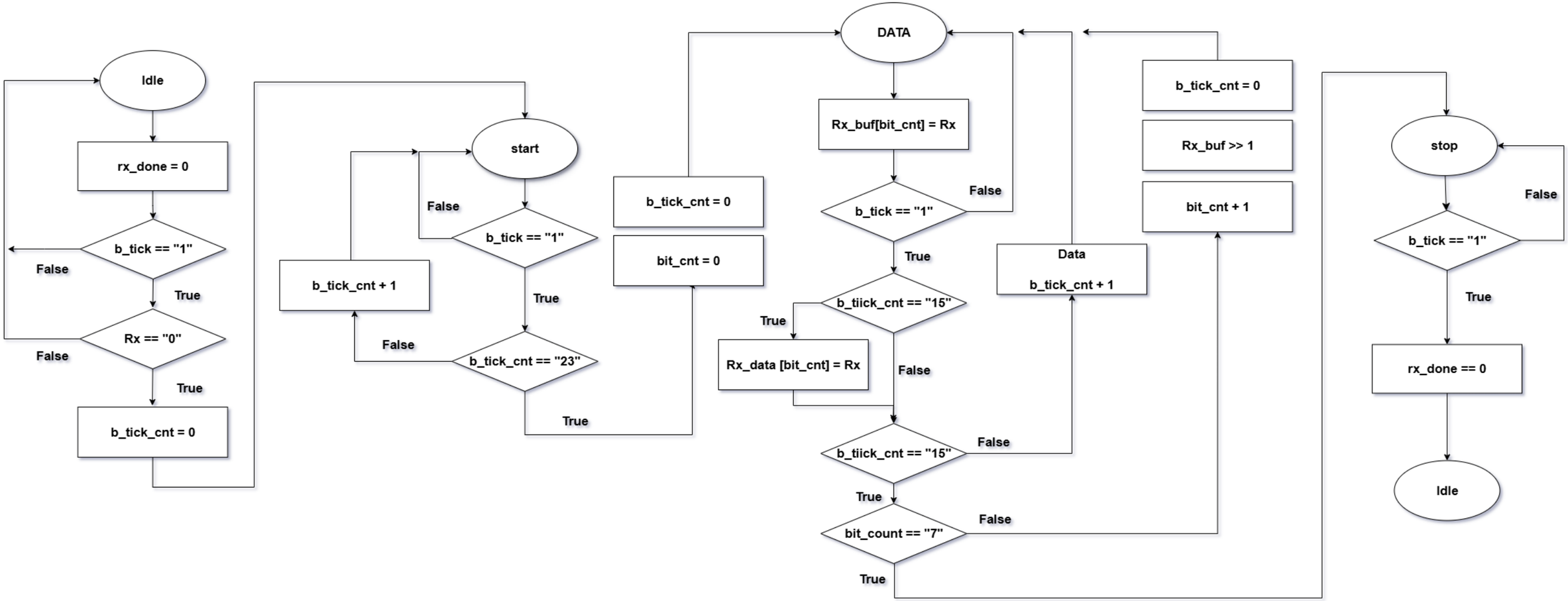
UART Rx

FSM


reset
b_tick == 1
rx == 0
b_tick == 1
b_tick_cnt == 23
b_tick == 1
b_tick_cnt == 15
bit_cnt == 7
Idle
rx_done = 0
Start
DATA
Stop
rx_done = 1
b_tick == 1
b_tick_cnt == 15
bit_cnt != 7
b_tick == 1

UART Rx

ASM


Idle
rx_done = 0
b_tick == "1"
False
True
Rx == "0"
False
True
b_tick_cnt = 0
start
False
b_tick == "1"
True
b_tick_cnt + 1
False
b_tick_cnt == "23"
True
b_tick_cnt = 0
bit_cnt = 0
DATA
Rx_buf[bit_cnt] = Rx
b_tick == "1"
False
True
b_tiick_cnt == "15"
True
Rx_data [bit_cnt] = Rx
False
b_tiick_cnt == "15"
False
True
bit_count == "7"
False
True
Data
b_tick_cnt + 1
b_tick_cnt = 0
Rx_buf >> 1
bit_cnt + 1
stop
False
b_tick == "1"
True
rx_done == 0
Idle

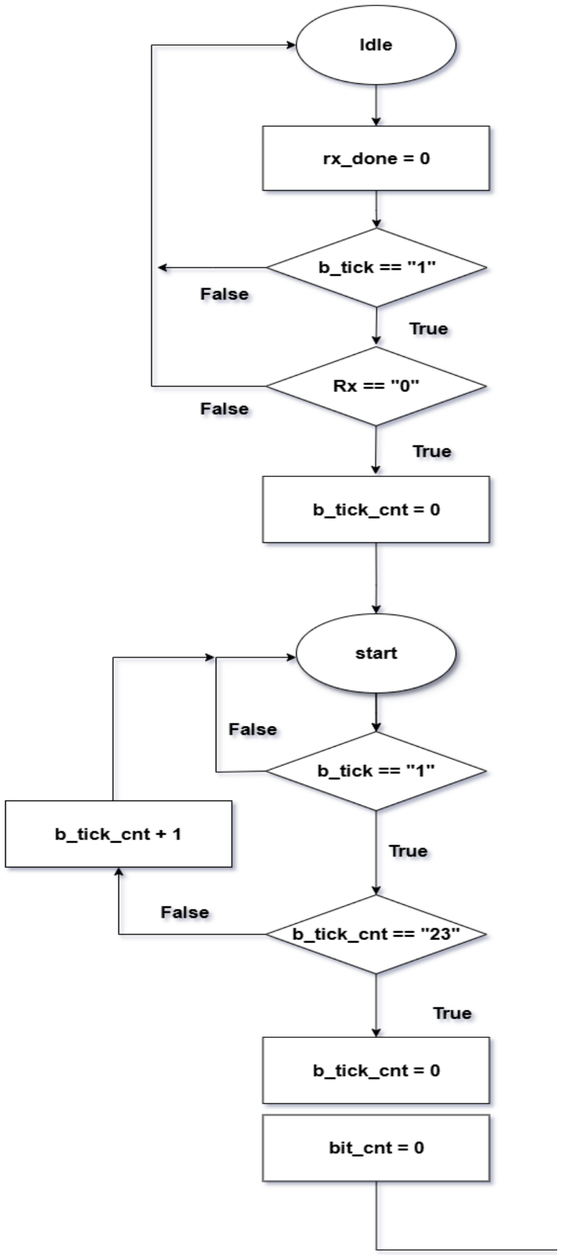
UART Rx

code & ASM

```
case (state)
    IDLE: begin
        rx_done_next = 1'b0;
        if (b_tick) begin
            if (!rx) begin
                b_tick_cnt_next = 0;
                next = START;
            end
        end
    end
    START: begin
        if (b_tick) begin
            if (b_tick_cnt_reg == 23) begin
                b_tick_cnt_next = 0;
                bit_cnt_next    = 0;
                next = DATA;
            end else begin
                b_tick_cnt_next = b_tick_cnt_reg + 1;
            end
        end
    end
```

Rx = 0 -> UART receive 시작

오버샘플링으로
각 데이터 비트 중앙에서
안정적인 값 읽어 오기



UART Rx

code & ASM

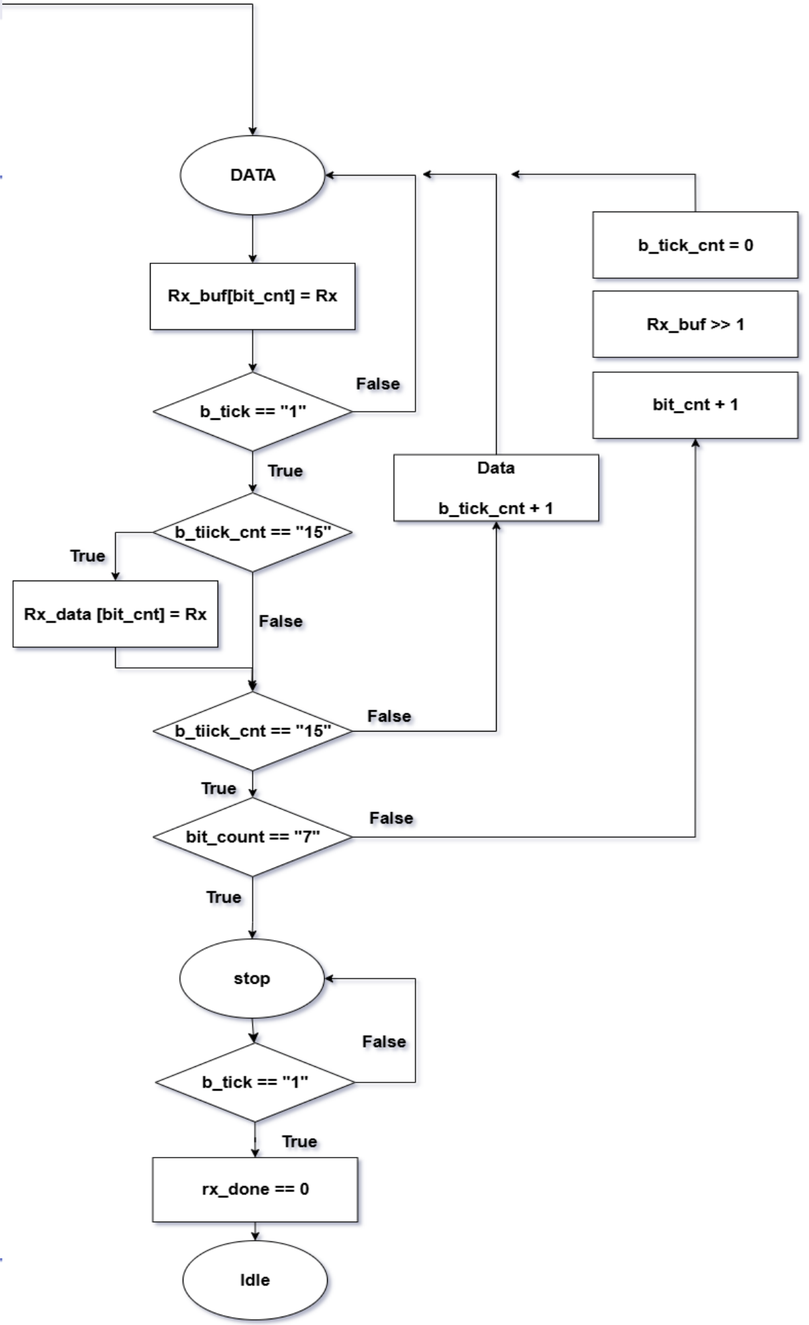
```
DATA: begin
    if (b_tick) begin
        if (b_tick_cnt_reg == 0) begin
            rx_buf_next[7] = rx;
        end
        if (b_tick_cnt_reg == 15) begin
            if (bit_cnt_reg == 7) begin
                next = STOP;
            end else begin
                b_tick_cnt_next = 0;
                bit_cnt_next = bit_cnt_reg + 1;
                rx_buf_next = rx_buf_reg >> 1;
            end
        end else begin
            b_tick_cnt_next = b_tick_cnt_reg + 1;
        end
    end
end
STOP: begin
    if (b_tick) begin
        rx_done_next = 1'b1;
        next = IDLE;
    end
end
endcase
```

오버샘플링으로
각 데이터 비트 중앙에서
안정적인 값 읽어 오기

Rx의 값을 읽어
데이터 버퍼에 저장

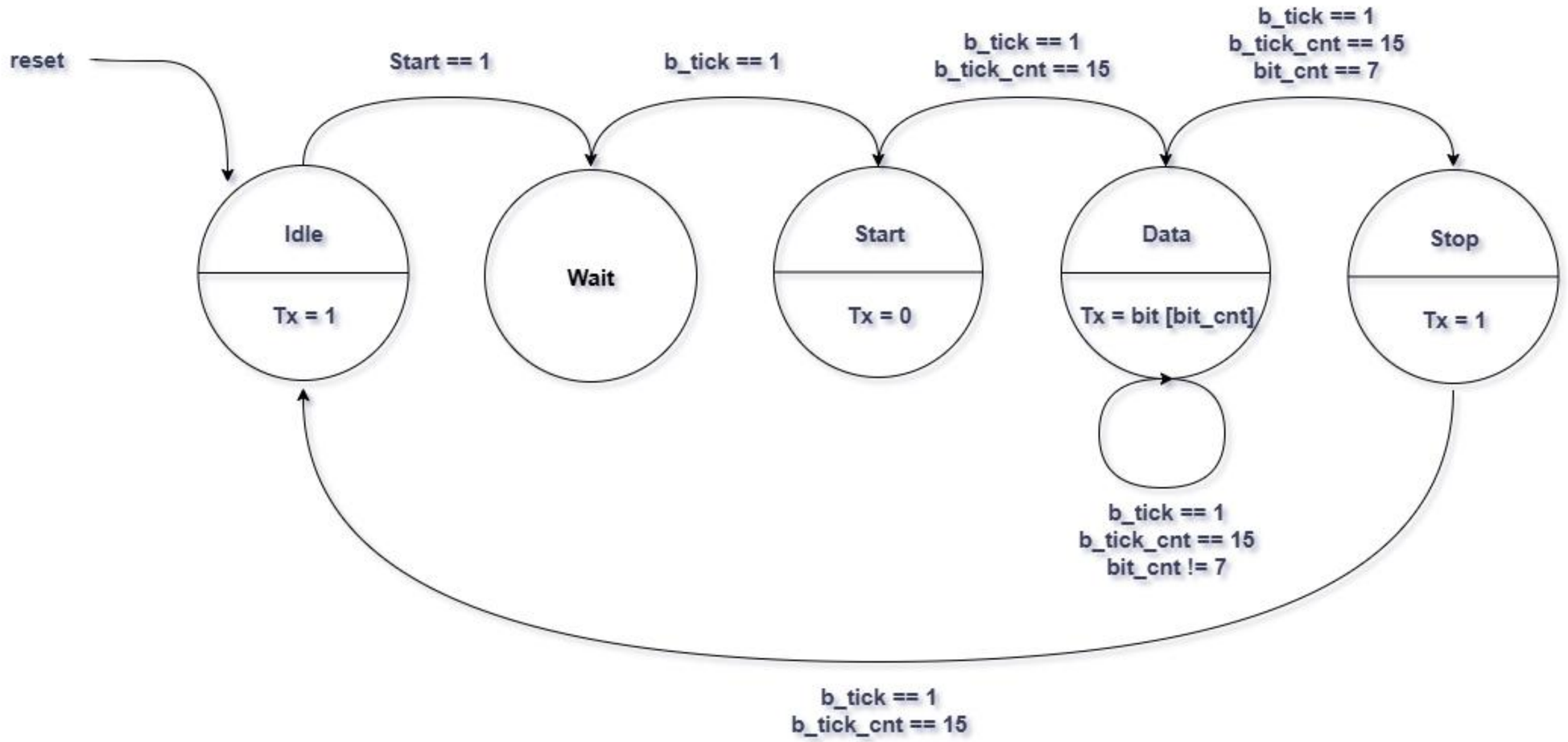
버퍼 시프트하면서
이 과정을 8번 반복하여
8비트 데이터를 완성
(직렬 => 병렬)

수신 완료 신호 생성



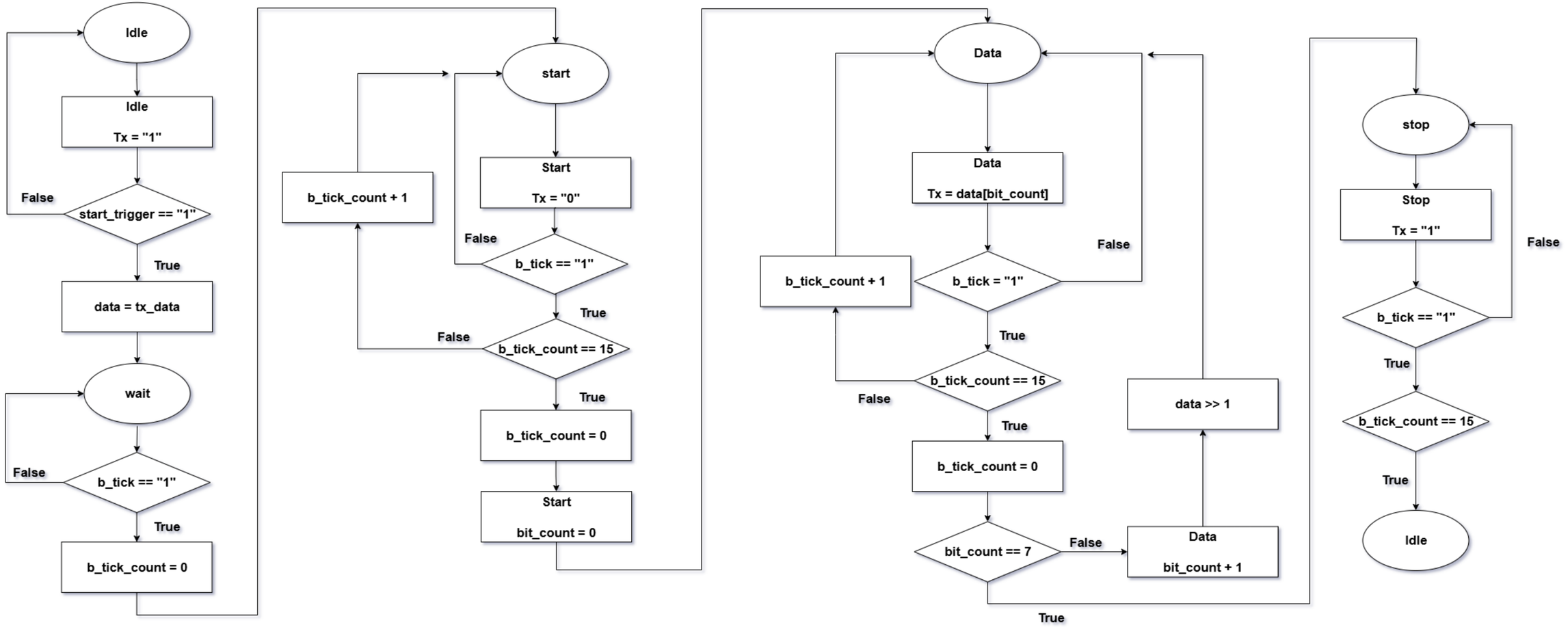
UART Tx

FSM


reset
Start == 1
b_tick == 1
b_tick == 1
b_tick_cnt == 15
b_tick == 1
b_tick_cnt == 15
bit_cnt == 7
Idle
Tx = 1
Wait
Start
Tx = 0
Data
Tx = bit [bit_cnt]
Stop
Tx = 1
b_tick == 1
b_tick_cnt == 15
bit_cnt != 7
b_tick == 1
b_tick_cnt == 15

UART Tx

ASM


Idle
Idle
Tx = "1"
False
start_trigger == "1"
True
data = tx_data
wait
False
b_tick == "1"
True
b_tick_count = 0
b_tick_count + 1
start
Start
Tx = "0"
False
b_tick == "1"
True
False
b_tick_count == 15
True
b_tick_count = 0
Start
bit_count = 0
Data
Data
Tx = data[bit_count]
False
b_tick_count + 1
b_tick = "1"
True
False
b_tick_count == 15
True
b_tick_count = 0
bit_count == 7
False
Data
bit_count + 1
data >> 1
True
stop
Stop
Tx = "1"
False
b_tick == "1"
True
b_tick_count == 15
True
Idle

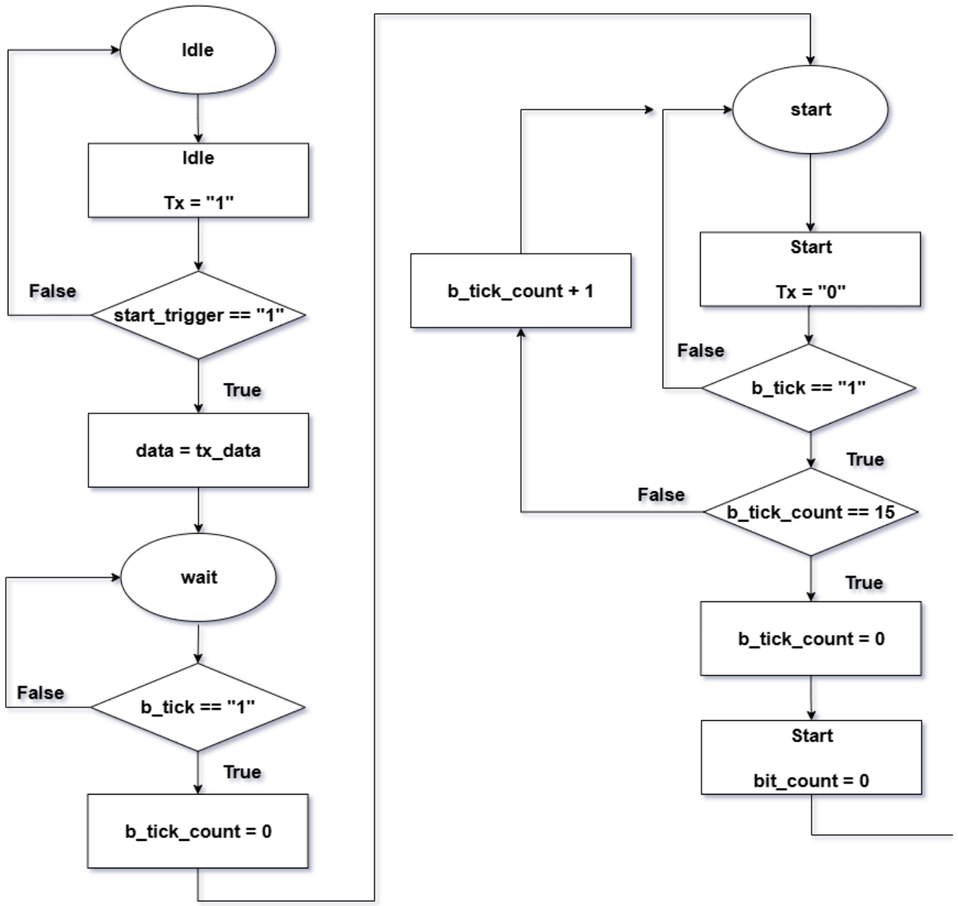
UART Tx

code & ASM

```
case (state)
    IDLE: begin
        tx_next = 1'b1;
        tx_busy_next = 1'b0;
        if (start_trigger) begin
            tx_busy_next = 1'b1;
            data_next = tx_data;
            next = WAIT;
        end
    end
    WAIT: begin
        if (b_tick) begin
            b_tick_cnt_next = 0;
            next = START;
        end
    end
    START: begin
        tx_next = 1'b0;
        if (b_tick) begin
            if (b_tick_cnt_reg == 15) begin
                b_tick_cnt_next = 0;
                bit_cnt_next = 0;

                next = DATA;
            end else begin
                b_tick_cnt_next = b_tick_cnt_reg + 1;
            end
        end
    end
```

Start trigger 들어오면
Tx_busy를 통해
현재 상황(송신 중)을 알리기

수신 측에서 안정적으로
신호 감지할 수 있도록
정확한 시간 제어



UART Tx

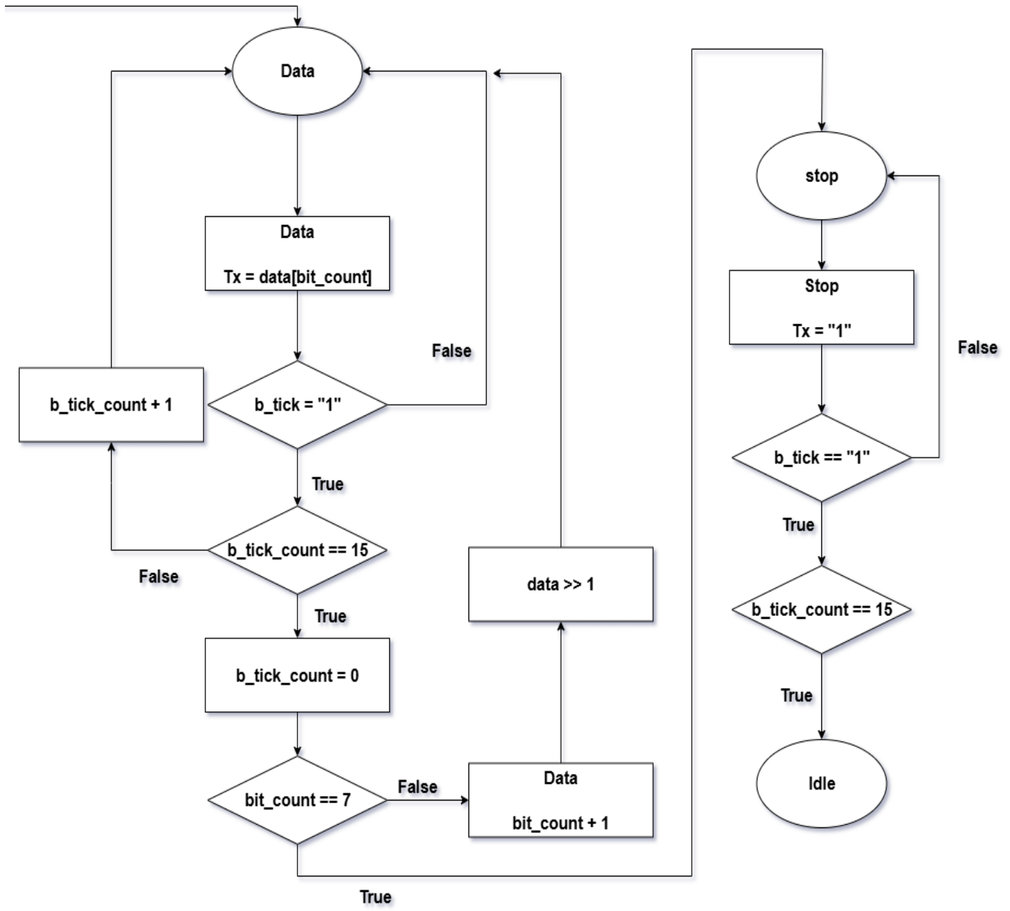
code & ASM

```
    DATA: begin
        // output tx = tx_data[0]
        tx_next = data_reg[0];
        if (b_tick) begin
            if (b_tick_cnt_reg == 15) begin
                b_tick_cnt_next = 0;
                if (bit_cnt_reg == 7) begin
                    next = STOP;
                end else begin
                    bit_cnt_next = bit_cnt_reg + 1;
                    data_next = data_reg >> 1;
                end
            end else begin
                b_tick_cnt_next = b_tick_cnt_reg + 1;
            end

        end
    end
    STOP: begin
        tx_next = 1'b1;
        if (b_tick) begin
            if (b_tick_cnt_reg == 15) begin
                tx_busy_next = 1'b0;
                next = IDLE;
            end else begin
                b_tick_cnt_next = b_tick_cnt_reg + 1;
            end
        end
    end
endcase
```

수신 측에서 안정적으로
신호 감지할 수 있도록
정확한 시간 제어

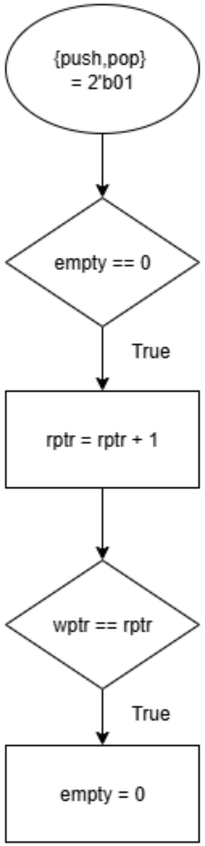
8비트 데이터를
데이터 시프트하면서
1비트씩 전송
(병렬 => 직렬)

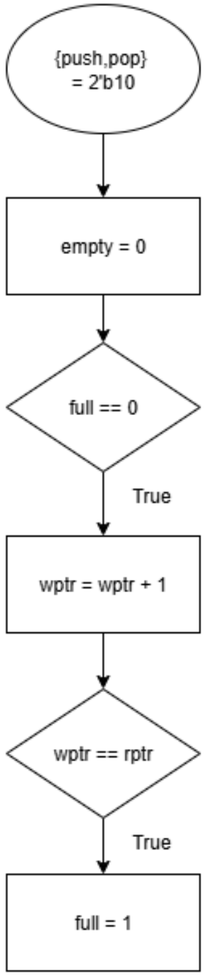
Tx_busy를 통해
현재 상황(송신 완료)을 알리기

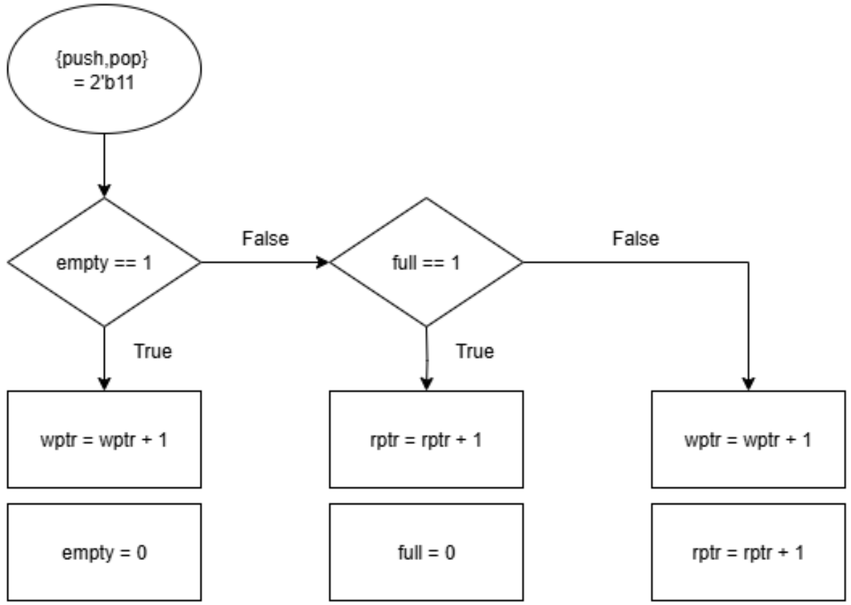


UART FIFO

ASM







UART FIFO

code & ASM

```
case ({
    push, pop
})
    2'b01: begin
        // pop
        full_next = 1'b0;
        if (!empty_reg) begin
            rptr_next = rptr_reg + 1;
            if (wptr_reg == rptr_next) begin
                empty_next = 1'b1;
            end
        end
    end
    2'b10: begin
        // push
        empty_next = 1'b0;
        if (!full_reg) begin
            wptr_next = wptr_reg + 1;
            if (wptr_next == rptr_reg) begin
                full_next = 1'b1;
            end
        end
    end
```

POP

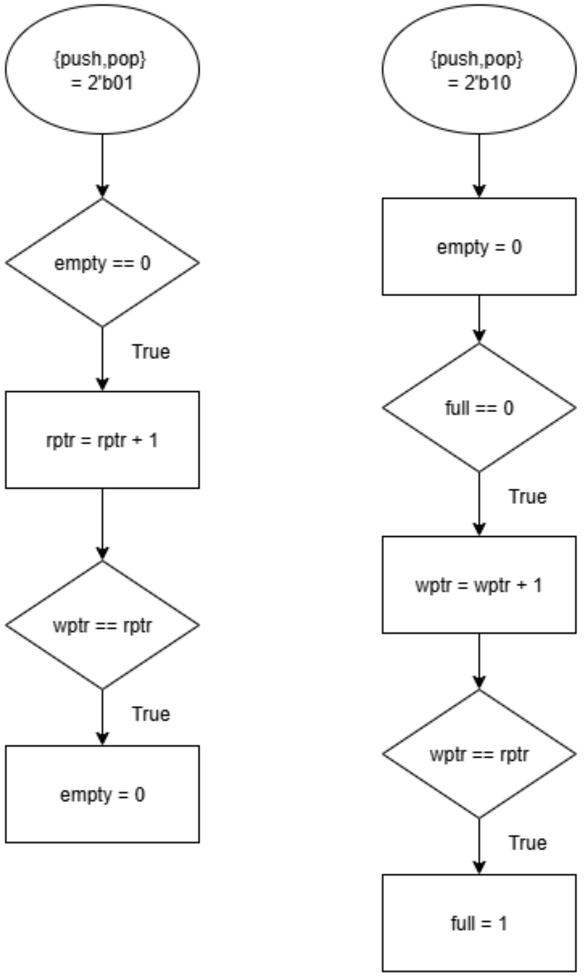
FIFO empty : 0
-> pop & 읽기 포인터 (rptr) 이동

-> empty : 1
(읽기 포인터와 쓰기 포인터 위치 일치할 경우)

PUSH

FIFO full : 0
-> push & 쓰기 포인터 (wptr) 이동

-> full : 1
(쓰기 포인터와 읽기 포인터 위치 일치할 경우)



UART FIFO

code & ASM

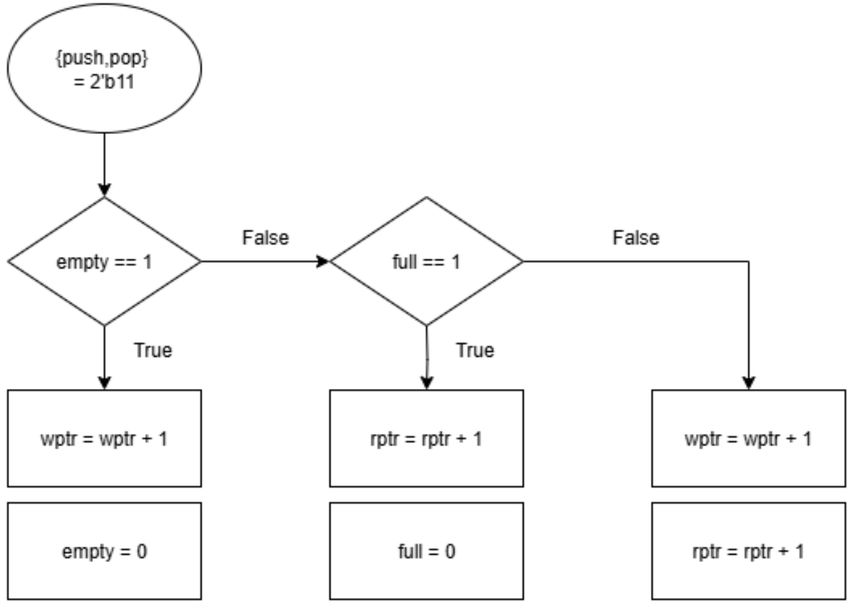
```
    2'b11: begin
        if (empty_reg == 1'b1) begin
            wptr_next  = wptr_reg + 1;
            empty_next = 1'b0;
        end else if (full_reg == 1'b1) begin
            rptr_next = rptr_reg + 1;
            full_next = 1'b0;
        end else begin
            // not be full, empty
            wptr_next = wptr_reg + 1;
            rptr_next = rptr_reg + 1;
        end
    end
endcase
```

PUSH, POP 동시 처리

empty : 1
->push & 쓰기 포인터 (wptr) 1 증가
-> empty : 0

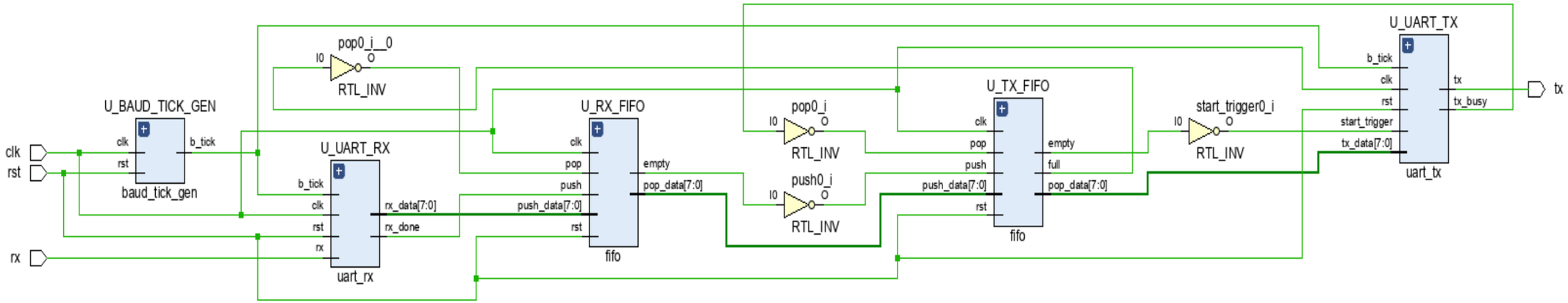
full : 1
-> pop & 읽기 포인터 (rptr) 1 증가
-> full : 0

empty / full : 0
-> 데이터 push, pop &
쓰기 포인터 (wptr), 읽기 포인터 (rptr)
둘 다 이동



UART Schematic

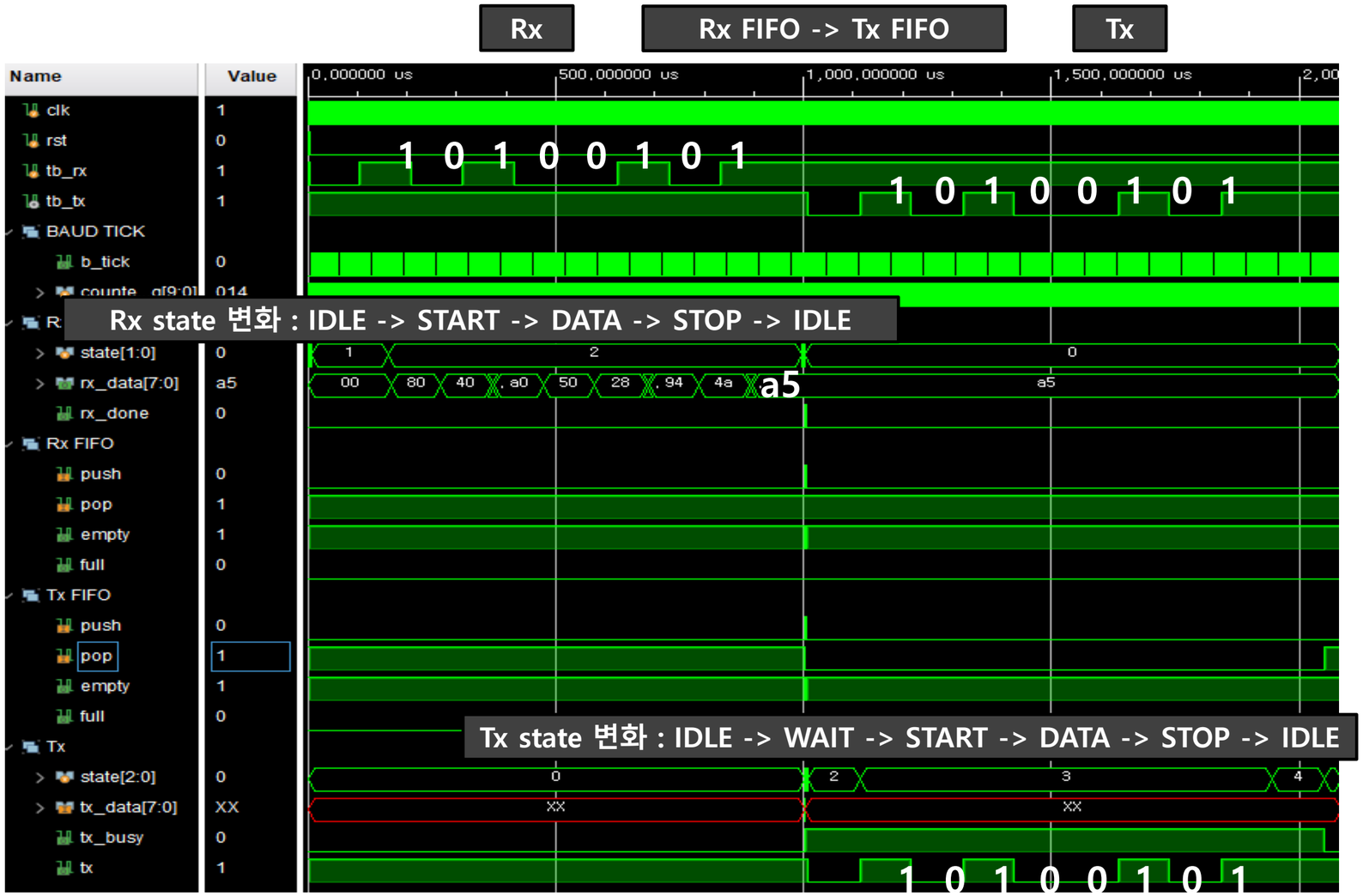
uart



검증을 위해 loop back 구조로 설계

UART Simulation

uart



UART loop back 구조 검증

PC → FPGA (수신)

10100101 (직렬 데이터)가
UART Rx 를 통해
8'ha5 (병렬 데이터)로 변환

FPGA 내부

8'ha5 데이터가
Rx FIFO를 거쳐
Tx FIFO로 전달

FPGA → PC (송신)

8'ha5 (병렬 데이터)가
UART Tx 를 통해
10100101 (직렬 데이터)로
변환되어 출력

UART Simulation

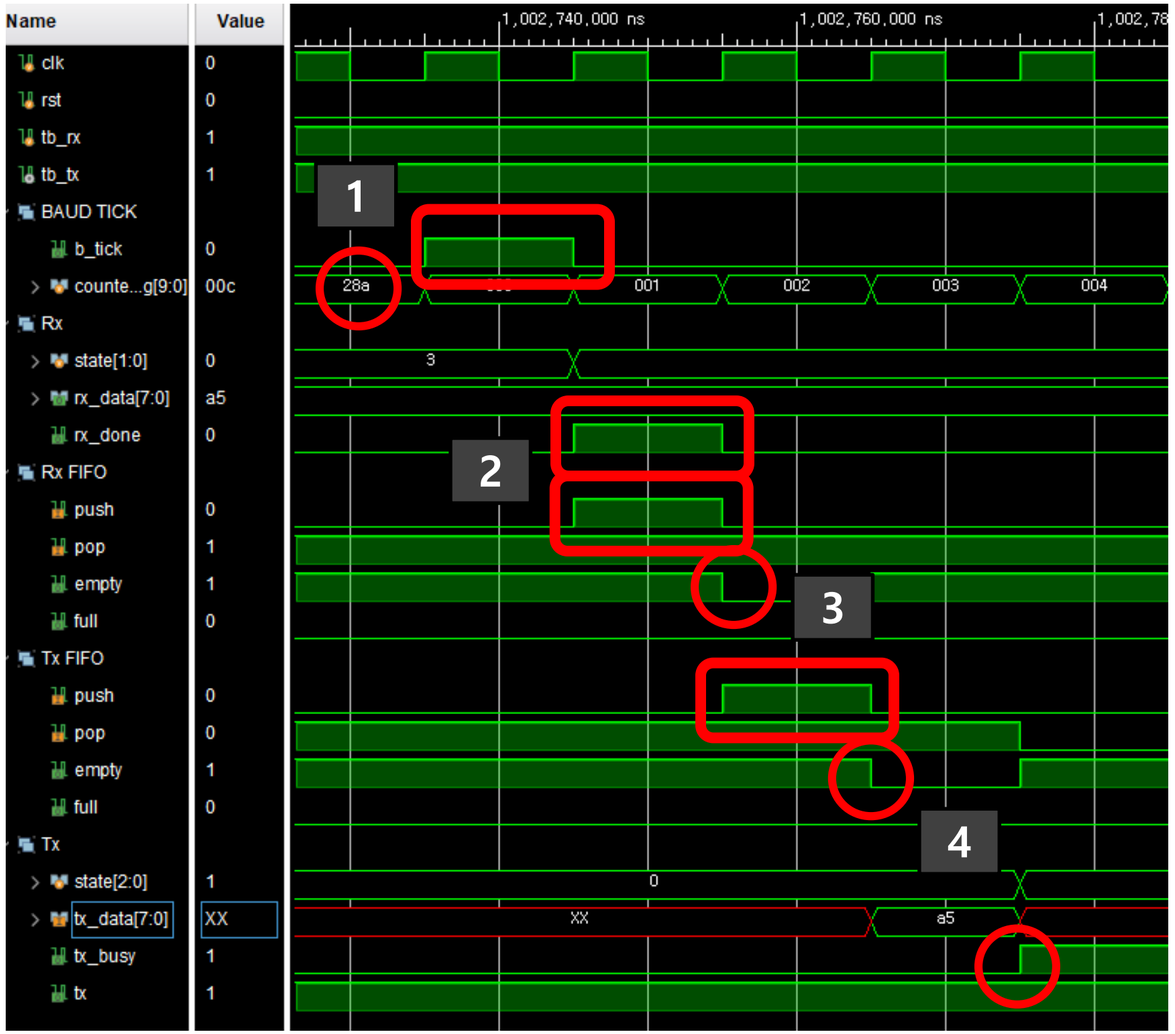
uart rx fifo / tx fifo

1

BAUD TICK
counter_reg = 650(28a) 되면
b_tick 생성

2

Rx : rx_done 1
(수신 완료)
⇒ Rx FIFO : push 1
(수신 데이터 Rx FIFO에 저장)



3

Rx FIFO : empty 0
Rx FIFO : pop 1
⇒ Tx FIFO : push 1
(Rx FIFO 데이터
Tx FIFO로 전달)

4

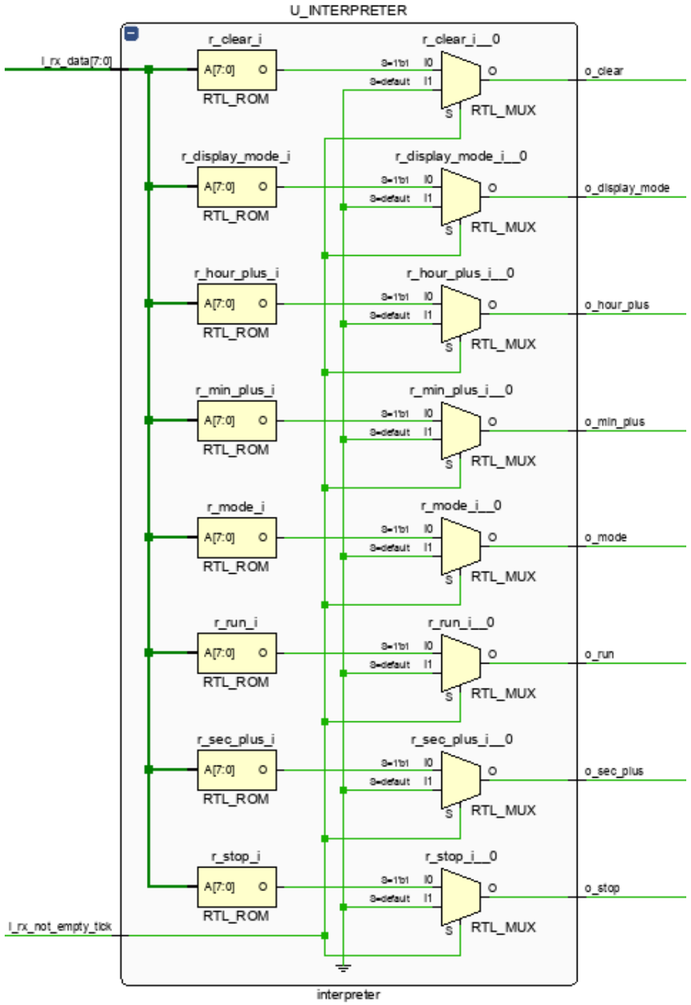
Tx FIFO : empty 0
⇒ Tx : tx_busy 1
(Tx FIFO로부터 데이터 송신 중)

INTERPRETER

code & schematic

PC 에서 받은 입력을 pulse으로 만들어주는 모듈

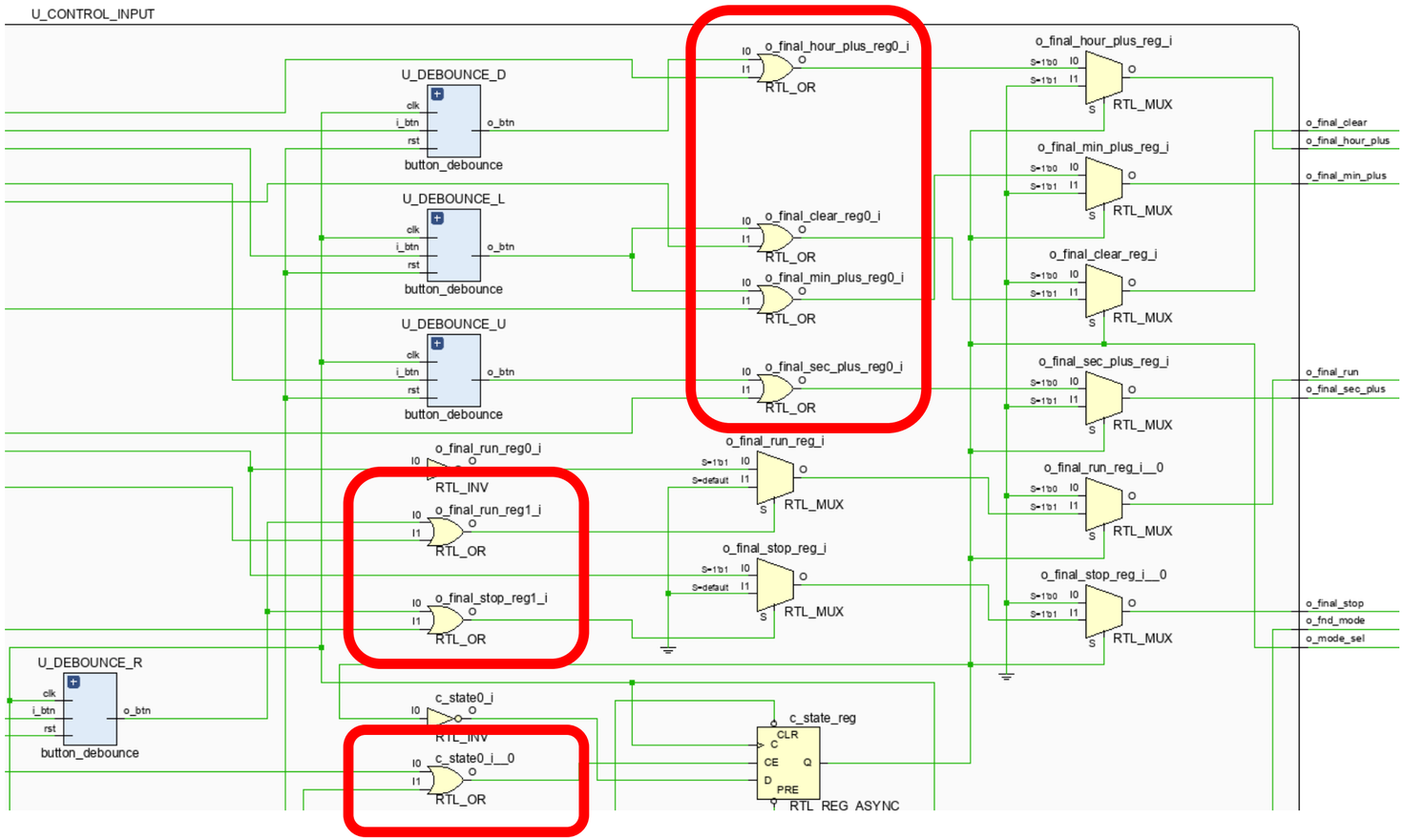
```
always @(*) begin
    r_run = 1'b0;
    r_stop = 1'b0;
    r_clear = 1'b0;
    r_mode = 1'b0;
    r_display_mode = 1'b0;
    r_sec_plus = 1'b0;
    r_min_plus = 1'b0;
    r_hour_plus = 1'b0;

    if (i_rx_not_empty_tick) begin
        case (i_rx_data)
            "r": r_run   = 1'b1;
            "s": r_stop  = 1'b1;
            "c": r_clear = 1'b1;
            "m": r_mode  = 1'b1;
            "d": r_display_mode = 1'b1;
            "S": r_sec_plus = 1'b1;
            "M": r_min_plus = 1'b1;
            "H": r_hour_plus = 1'b1;
        endcase
    end
end
```



Control Unit

code & schematic

Button/ Switch를 통해 받은 입력과 PC 에서 받은 입력을 제어하여 최종 동작 신호를 출력하는 모듈




```
case (c_state)
    WATCH_MODE: begin
        o_final_sec_plus_reg  = w_btn_U_d | i_sec_plus;
        o_final_min_plus_reg  = w_btn_L_d | i_min_plus;
        o_final_hour_plus_reg = w_btn_D_d | i_hour_plus;
    end
    STOPWATCH_MODE: begin
        if (w_btn_R_d || i_run) o_final_run_reg = !i_is_running;
        if (w_btn_R_d || i_stop) o_final_stop_reg = i_is_running;
        o_final_clear_reg = w_btn_L_d | i_clear;
    end
endcase
```

Button/Switch 로 직접 제어

OR

PC에서 UART를 통해 원격 제어

Control Unit

code

```
case (c_state)
    WATCH_MODE: begin
        o_final_sec_plus_reg  = w_btn_U_d | i_sec_plus;
        o_final_min_plus_reg  = w_btn_L_d | i_min_plus;
        o_final_hour_plus_reg = w_btn_D_d | i_hour_plus;
    end
    STOPWATCH_MODE: begin
        if (w_btn_R_d || i_run) o_final_run_reg = !i_is_running;
        if (w_btn_R_d || i_stop) o_final_stop_reg = i_is_running;
        o_final_clear_reg = w_btn_L_d | i_clear;
    end
endcase
```

Stopwatch run / stop

- 보드 내에서 하나의 버튼(Btn_R)으로 두 개의 동작
  - PC에서는 각각 입력 존재 (r / s)

- Run stop 각각의 펄스를 만들기 위해 Toggle 이용

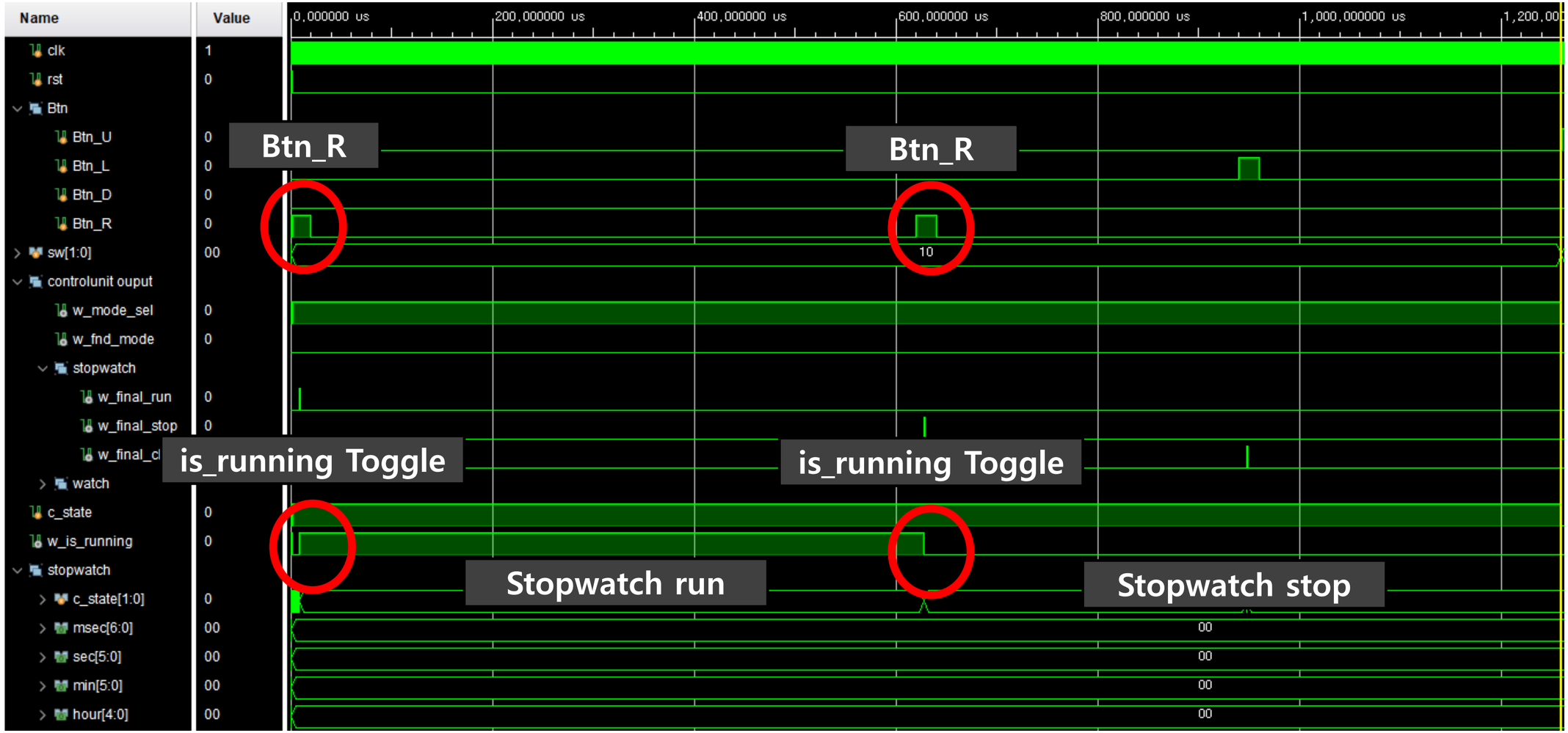
  - i_is_running : stopwatch cu 에서 받아온 신호
    (run state일 경우 1 , stop state일 경우 0)

STOPWATCH CU
Run state일 경우
-> o_final_run_reg = 0
-> o_final_stop_reg = 1
=> Stop pulse 생성

STOPWATCH CU
Stop state일 경우
-> o_final_run_reg = 1
-> o_final_stop_reg = 0
=> Run pulse 생성

Control Unit

stopwatch run / stop simulation



Control Unit

우선순위 설정 – PC 제어 우선권

```
reg pc_mode_lock;
always @(posedge clk, posedge rst) begin
    if (rst) begin
        pc_mode_lock <= 1'b0;
    end else if (i_mode) begin
        pc_mode_lock <= 1'b1;
    end
end
```

UART 에서 mode 변환 신호 보낼 경우
=> pc mode lock : 1

```
reg  r_sw1_prev;
wire w_sw1_changed = (i_sw[1] != r_sw1_prev);

always @(posedge clk, posedge rst) begin
    if (rst) begin
        c_state <= i_sw[1];
        r_sw1_prev <= i_sw[1];
    end else begin
        c_state <= n_state;
        r_sw1_prev <= i_sw[1];
    end
end
```

Mode(stopwatch-watch) 스위치 변화 감지

```
always @(*) begin
    n_state = c_state;
    if (i_mode || (w_sw1_changed && !pc_mode_lock)) begin
        n_state = ~c_state;
    end
end
```

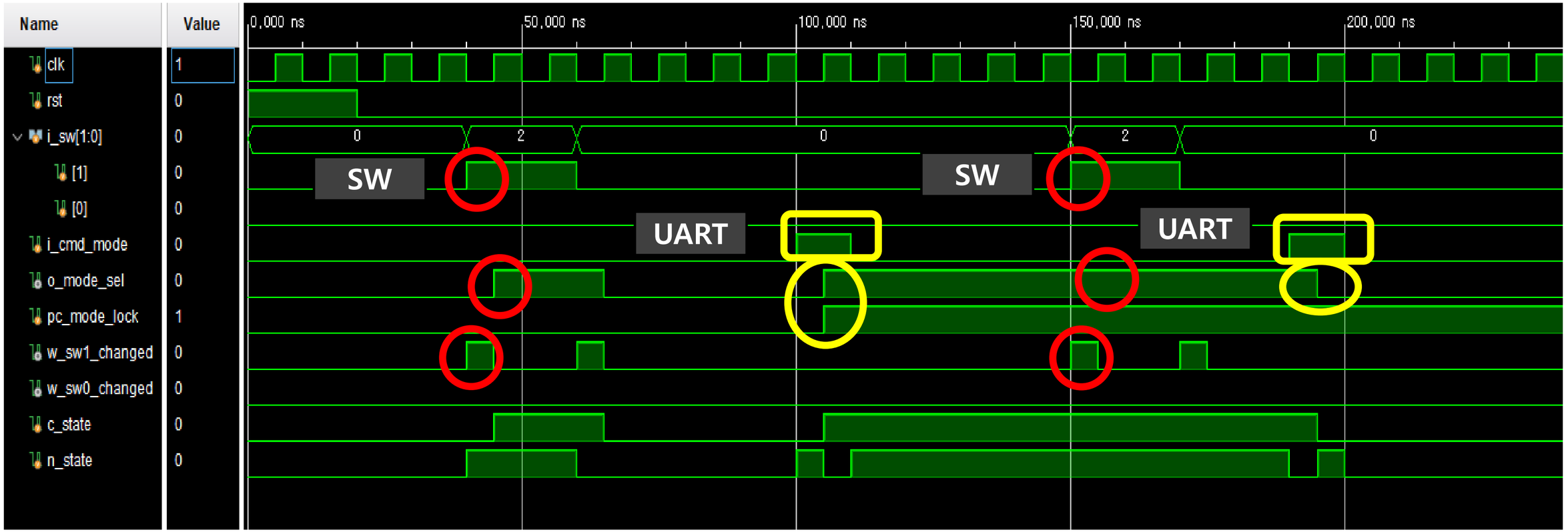
UART 에서 mode 변환 신호 송신
OR
mode 변환 스위치 변화 & pc mode lock : 0

State 변화
(stopwatch -> watch/
watch -> stopwatch)

Control Unit

우선순위 설정 – PC 제어 우선권 simulation

PC로 mode 제어 이후 스위치로 모드 제어 불가능



스위치로 mode 전환 : O

UART로 mode 전환 : O
-> mode lock : 1

스위치로 mode 전환 : X

UART로 mode 전환 : O

Control Unit

우선순위 설정 – PC 제어 우선권 simulation

PC로 mode 제어 이후 스위치를 이용한 모드 제어 이외의 기능은 정상 작동

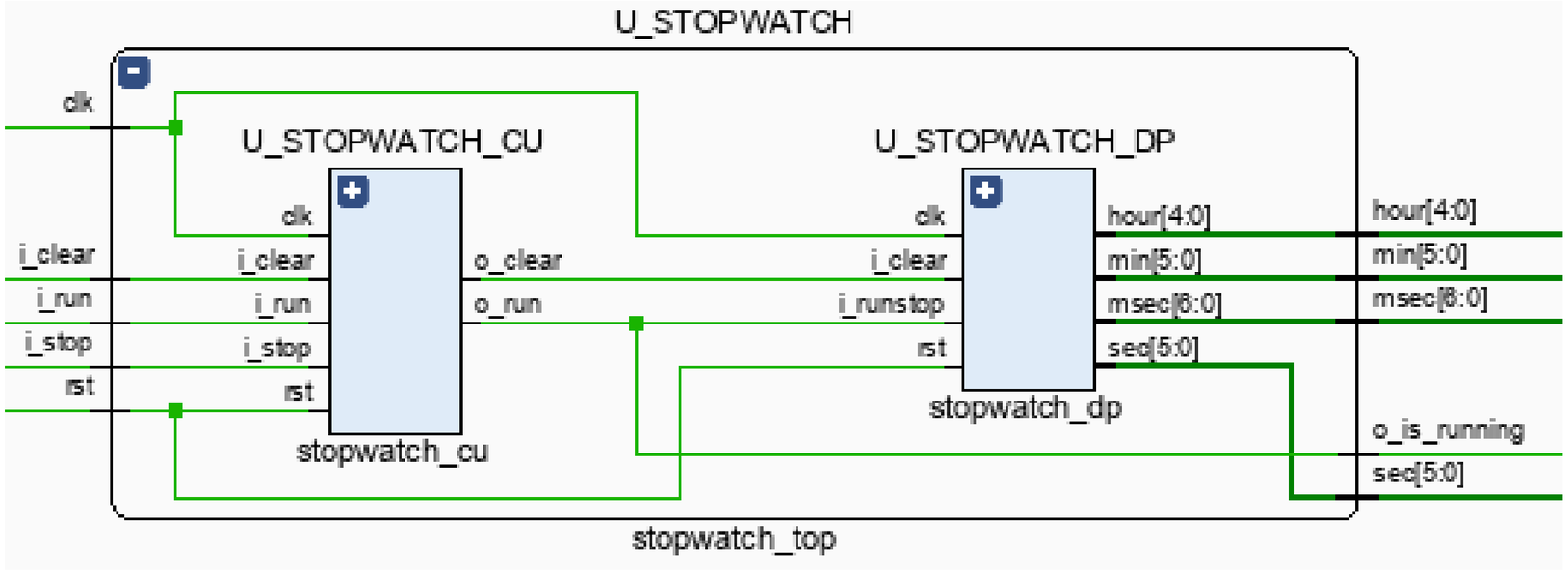
UART로 mode 전환 : O
-> mode lock : 1

UART로 mode 전환 이후에도
버튼을 사용한 제어 정상 작동

Stopwatch

code & schematic

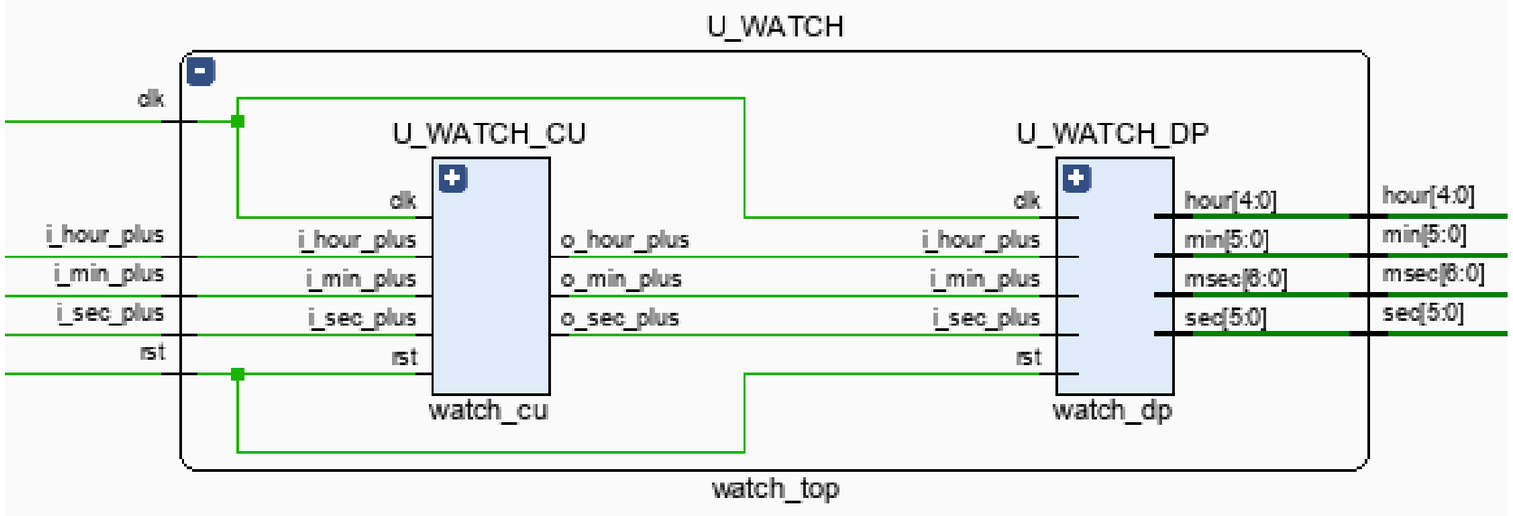
```
case (c_state)
    STOP: begin
        run_next = 1'b0;
        clear_next = 1'b0;
        if (i_run) begin
            n_state = RUN;
        end else if (i_clear) begin
            n_state = CLEAR;
        end
    end
    RUN: begin
        run_next = 1'b1;
        if (i_stop) begin
            n_state = STOP;
        end
    end
    CLEAR: begin
        clear_next = 1'b1;
        n_state = STOP;
    end
endcase
```



Watch

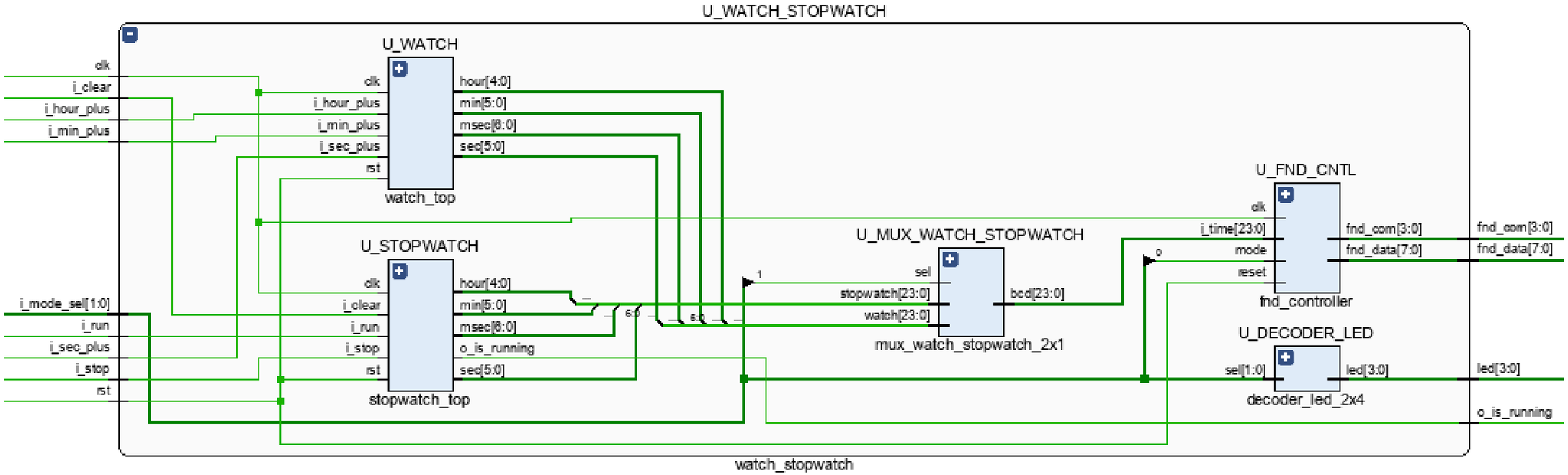
code & schematic

```
case (c_state)
    WATCH: begin
        // moore output
        secplus_next = 1'b0;
        minplus_next = 1'b0;
        hourplus_next = 1'b0;
        // next state
        if (i_sec_plus) begin
            n_state = SEC_PLUS;
        end else if (i_min_plus) begin
            n_state = MIN_PLUS;
        end else if(i_hour_plus) begin
            n_state = HOUR_PLUS;
        end
    end
    SEC_PLUS: begin
        secplus_next = 1'b1;
        if (i_sec_plus == 0) begin
            n_state = WATCH;
        end
    end
    MIN_PLUS: begin
        minplus_next = 1'b1;
        if (i_min_plus == 0) begin
            n_state = WATCH;
        end
    end
    HOUR_PLUS: begin
        hourplus_next = 1'b1;
        if (i_hour_plus == 0) begin
            n_state = WATCH;
        end
    end
```



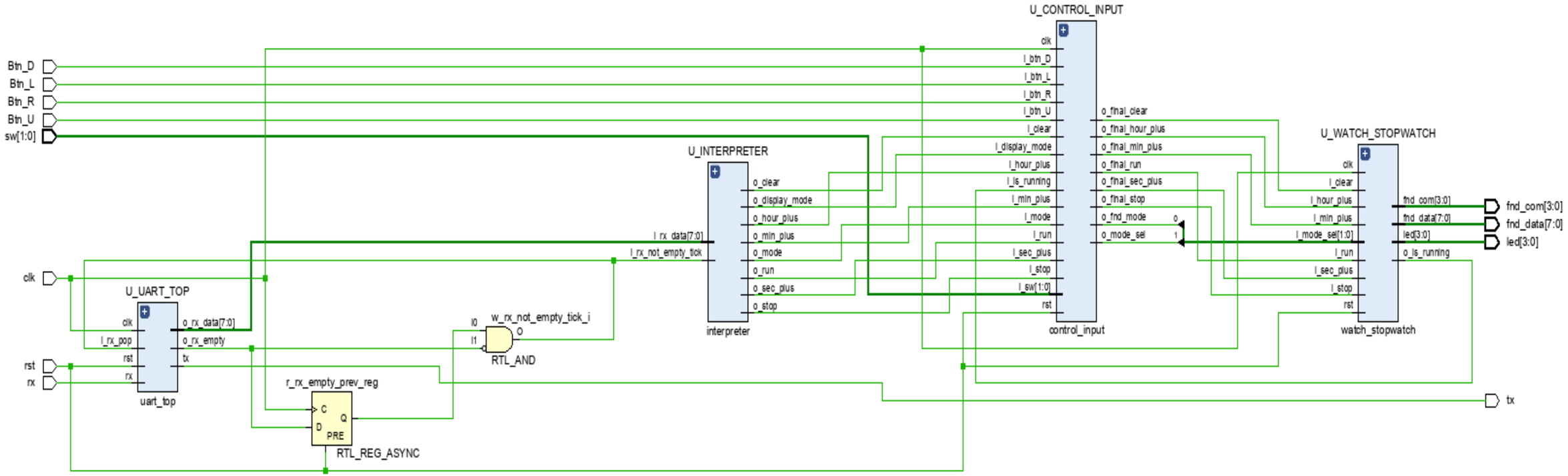
Stopwatch / Watch & FND CONTROLLER

schematic



Stopwatch / Watch + UART

schematic



DATE. 2025.8.25

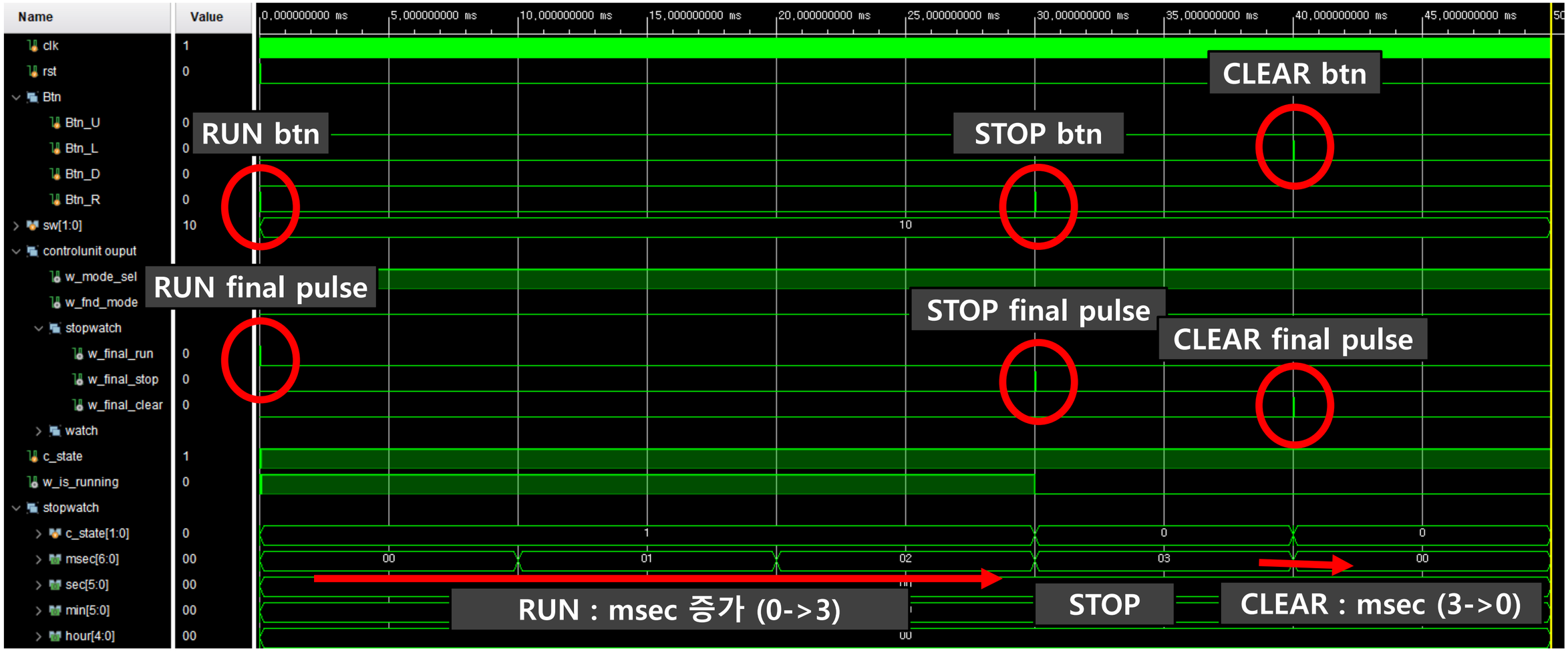
김은성

STOPWATCH / WATH + UART

Simulation

Simulation

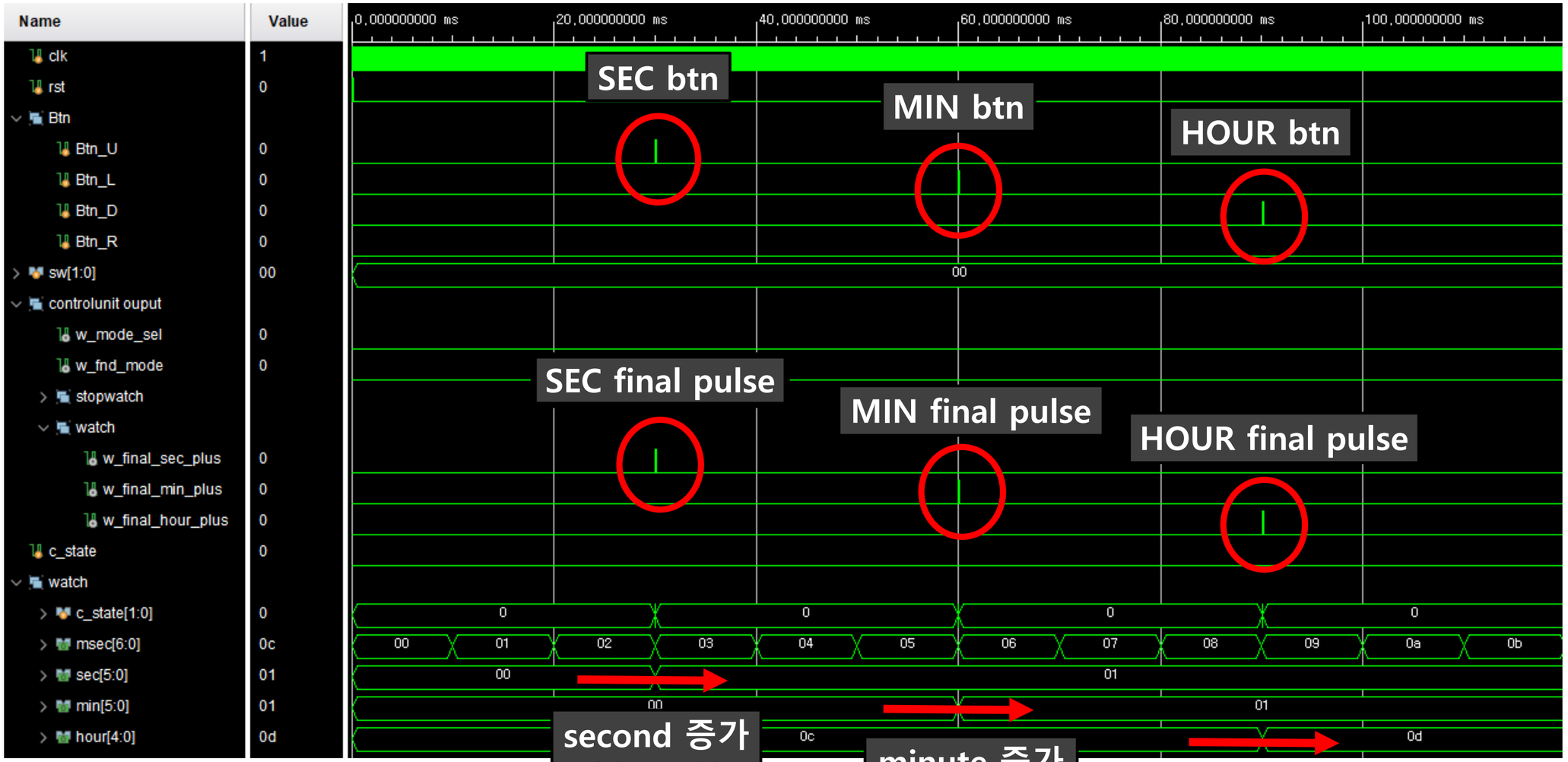
button으로 stopwatch 제어



STOPWATCH state 변화 : RUN -> STOP -> CLEAR -> STOP

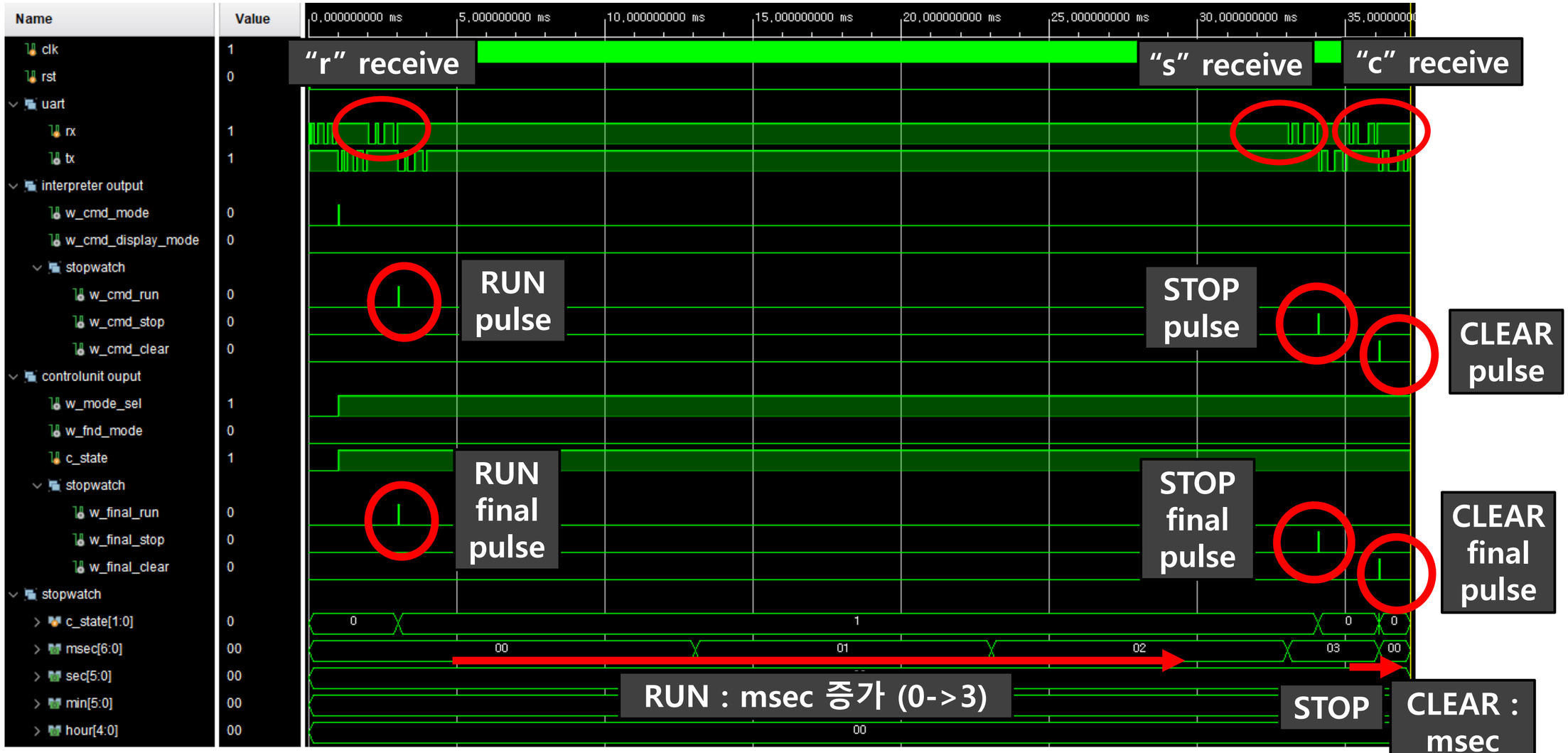
Simulation

button으로 watch 제어



Simulation

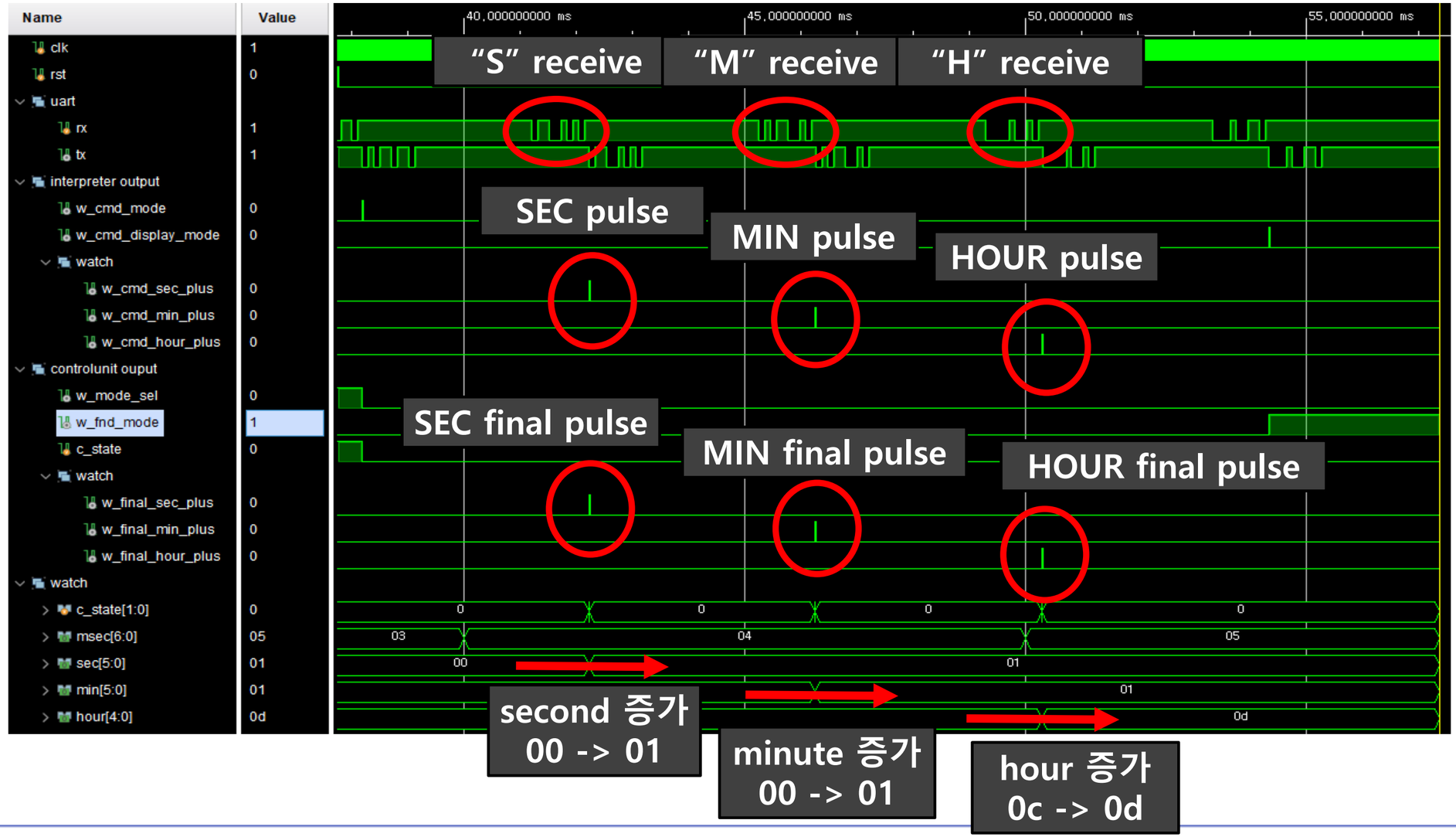
pc 입력으로 stopwatch 제어



STOPWATCH state 변화 : RUN -> STOP -> CLEAR -> STOP

Simulation

pc 입력으로 watch 제어



DATE. 2025.8.25

김은성

STOPWATCH / WATH + UART

Trouble Shooting

Trouble Shooting

스위치를 통한 모드 전환 명령어,
PC에서 UART로 보내는 모드 전환 명령어 간의
모드 전환 충돌 문제 발생
(모드 변경 오류 발생)

PC **제어에 우선권**을 주어서 해결

(PC로 모드 전환 명령어가 들어오면
물리적 스위치의 상태 변경은
다음 리셋 전까지 무시)

한 번의 입력에도 불구하고
명령이 계속해서 반복되는 오작동 발생
(메 클럭마다 명령 들어왔다고 인식)

Edge detector 사용하여 해결 (rx_empty)

한 클럭만 유지되는 Pulse 신호로 변경
(신호의 이전 클럭 상태와 현재 클럭 상태를
비교하여 상태 변경하는 순간을 감지)

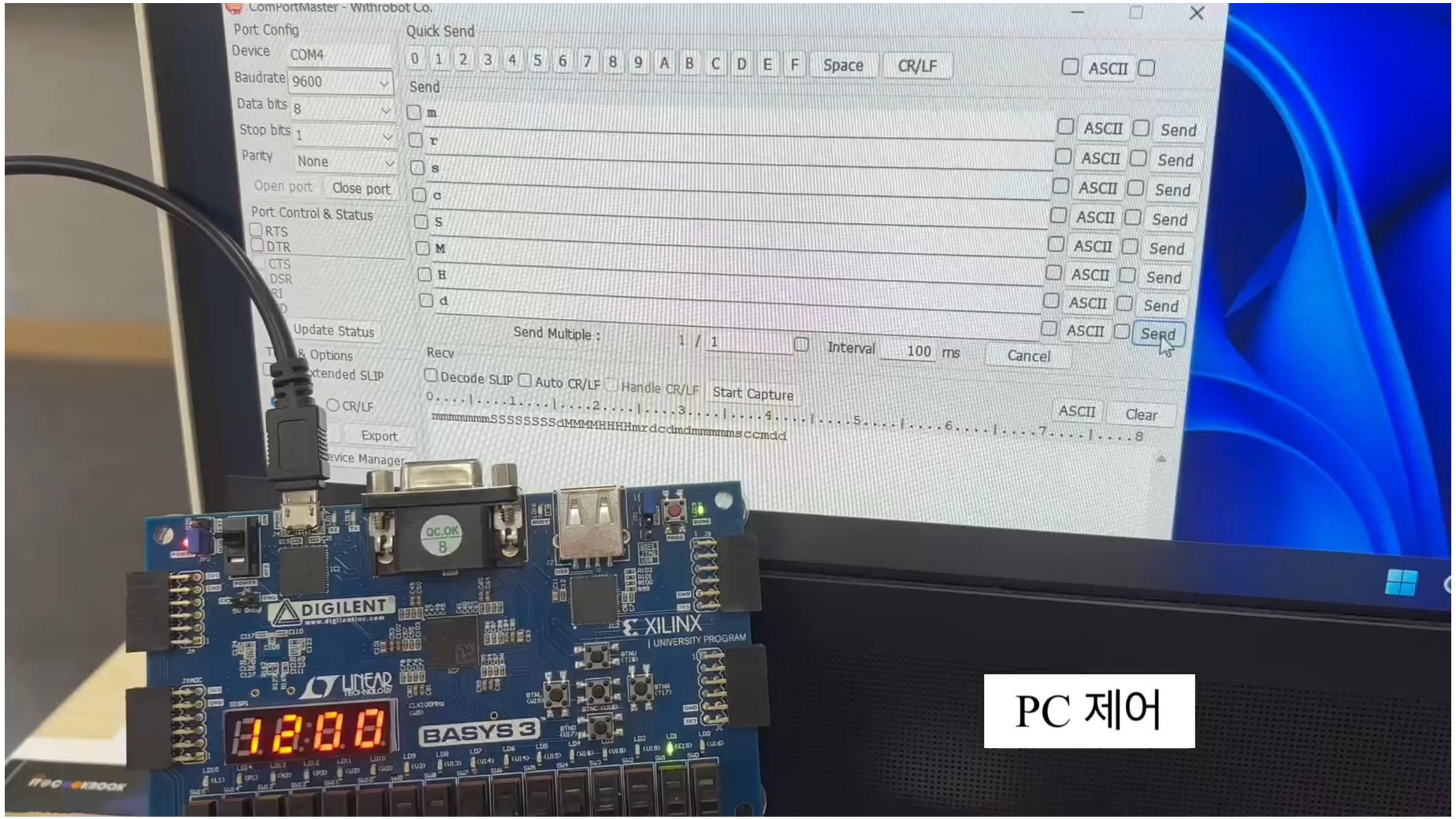
DATE. 2025.8.25

김은성

STOPWATCH / WATH + UART

동작 영상

동작 영상


ComPortMaster - Withrobot Co.
Port Config
Device COM4
Baudrate 9600
Data bits 8
Stop bits 1
Parity None
Open port
Close port
Port Control & Status
RTS
DTR
CTS
DSR
Update Status
Export
Quick Send
0 1 2 3 4 5 6 7 8 9 A B C D E F
Space
CR/LF
ASCII
Send
m
r
s
c
S
M
H
d
Send Multiple :
1 / 1
Interval 100 ms
Cancel
Recv
Decode SLIP
Auto CR/LF
Handle CR/LF
Start Capture
Clear
DIGILENT
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
PC 제어

DATE. 2025.8.25

김은성

THANK YOU

STOPWATCH / WATCH + UART

VerilogHDL