TEAM PROJECT

# UART

# STOPWATCH

# SR04_DHT11

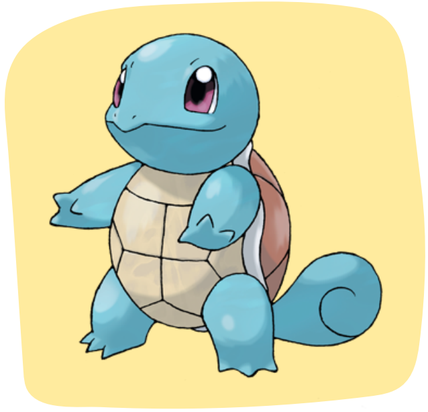
7조 : 심재훈, 진우석, 이승후, 유지훈

# 목차

# 01 팀구성원소개

## 진우석

stopwatch + UART



## 심재훈

sr04 + UART

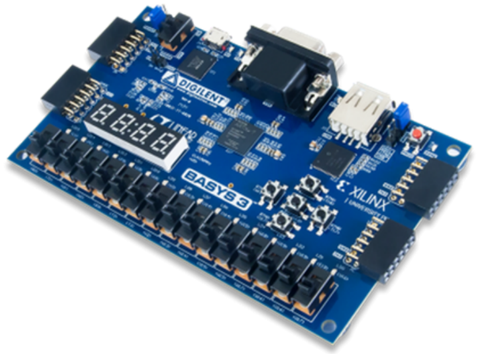
## 이승후

dht11 + UART

## 유지훈

전체 Top 모듈 통합
동작영상 제작

# 01 부품 스펙

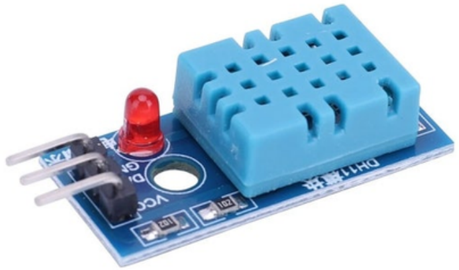



## Basys3

FPGA 교육용 보드
Artix-7 , 100 MHz clk,
USB-UART


## 초음파 센서

초음파를 이용해 물체까지의
거리를 측정하는 센서



## 온습도 센서

주위 환경의 온도(Temperature)와
습도(Humidity)를 측정하는 센서

# 02 개요

## 전체기능

스탑워치, 초음파 센서, 온습도 센서를 하나의 시스템으로 통합하여, 측정 및 제어 기능을 UART 통신을 통해 PC와 주고받으며 확인할 수 있는  시스템을 구현.

## 목표

- 스탑워치, 초음파, 온·습도를 하나의 통합 모듈
- PC와 시리얼 연결을 통해 측정값 출력 및 제어
- Verilog/Vivado 기반 FPGA 설계와 UART 통신을 통한 임베디드 시스템 통합 능력 강화.

## 요약

- FPGA 보드에서 스탑워치, 초음파 거리 측정, 온·습도 센서 측정 기능을 실행.
- UART 통신으로 PC에 결과를 출력하고, 사용자는 명령어 입력으로 기능 제어 가능.
- 버튼 입력과 UART 명령을 모두 지원하여 보드 독립 실행 + PC 제어 실행이 가능

# 03 stopwatch + UART

## 프로젝트의 목적 및 목표

UART를 통해 PC 〈-〉 FPGA간 통신 가능하게 설계

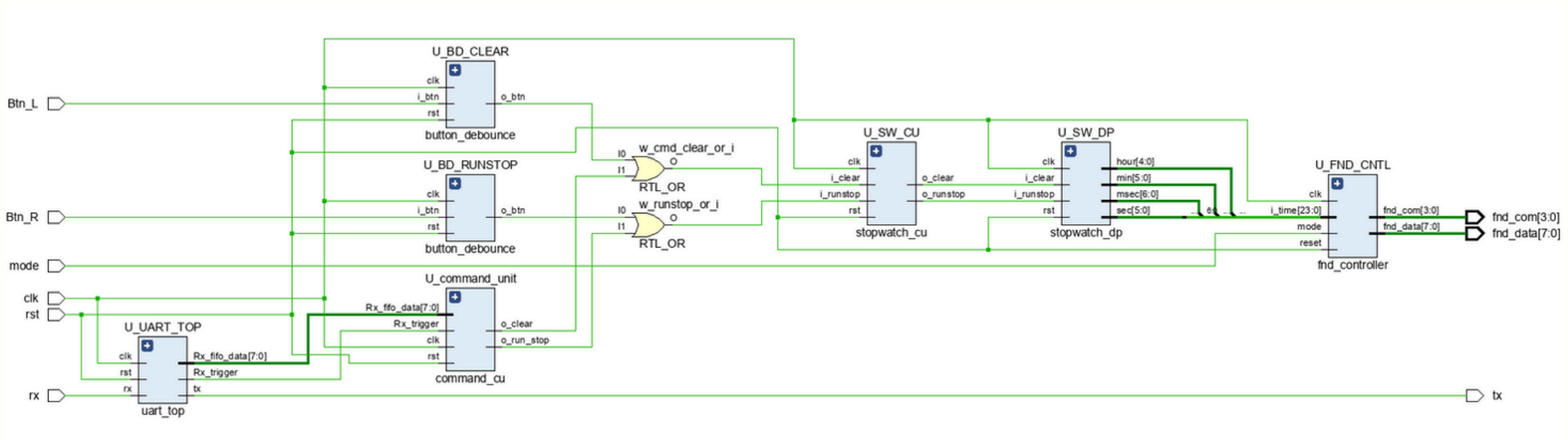
PC에서 전송된 명령에 따라 스톱워치 상태가 실시간 반응

## 기능

스톱워치는 Run, Stop, Reset 기능을 수행하며 버튼 입력으로 제어

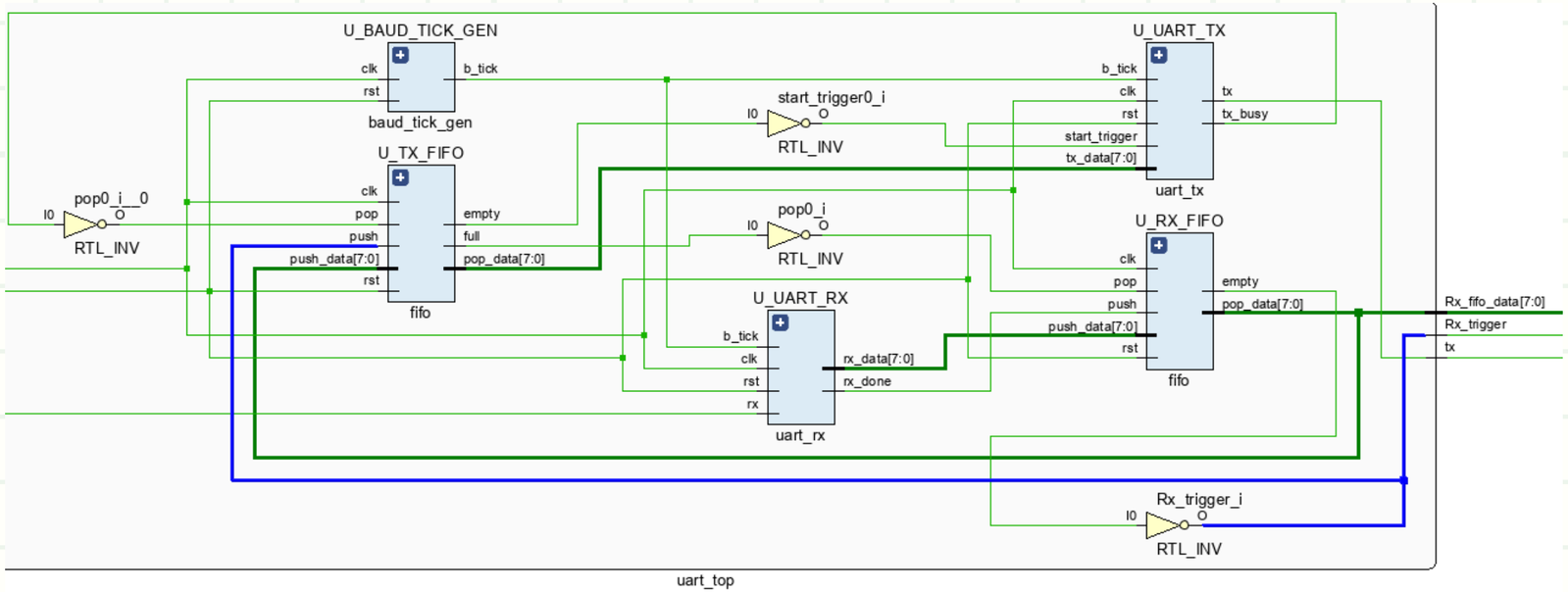
UART를 통해 수신된 명령을 해석하여 제어 신호로 변환 (Rx_fifo_data, Rx_trigger)

# 03 stopwatch + UART




- PC → UART → command_cu로 명령 수신
- 명령어 → run/stop/clear 제어 신호로 변환
- 버튼 입력 + UART 명령 OR 연산으로 스톱워치 제어

# 03 uart_top 기능




- FIFO 버퍼로 데이터 임시 저장 및 흐름 제어
- 데이터 손실 없이 안정적인 UART 통신 보장

★ FIFO – 데이터 안정성과 CDC 방지

# 03 코드리뷰(uart_top)

```
assign Rx_fifo_data = w_rx_fifo_popdata;
assign Rx_trigger   = ~w_rx_empty;


uart_tx U_UART_TX (
    .clk           (clk),
    .rst           (rst),
    .start_trigger(~w_tx_fifo_empty), // FIFO 비어있지 않으면 Rx_trigger 신호 발생
    .tx_data       (w_tx_fifo_popdata),
    .b_tick        (w_b_tick),
    .tx            (tx),
    .tx_busy       (w_tx_busy)
);
```

- FIFO가 비어있지 않으면 (w_rx_empty=0) → Rx_trigger=1
- FIFO가 비어있으면 (w_rx_empty=1) → Rx_trigger=0

# 03 코드리뷰(uart_top)

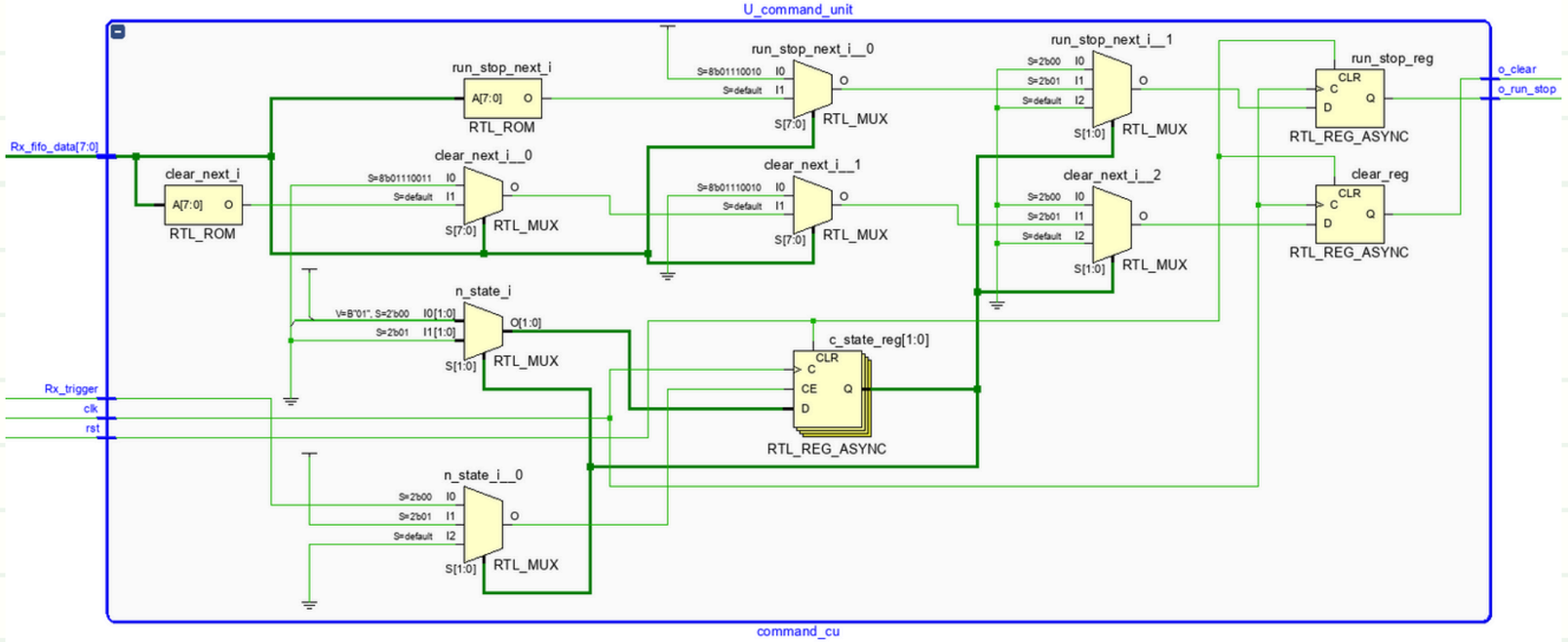
```
fifo U_TX_FIFO (
    .clk       (clk),
    .rst       (rst),
    .push_data(w_rx_fifo_popdata),
    .push      (~w_rx_empty),
    .pop       (~w_tx_busy),          //from uart tx
    .pop_data  (w_tx_fifo_popdata),   // to uart tx
    .full      (w_tx_fifo_full),
    .empty     (w_tx_fifo_empty)      // to uart tx
);

fifo U_RX_FIFO (
    .clk       (clk),
    .rst       (rst),
    .push_data(w_rx_data),            // from uart rx
    .push      (rx_done),             // from uart rx
    .pop       (~w_tx_fifo_full),     // to tx fifo
    .pop_data  (w_rx_fifo_popdata),   // to tx fifo
    .full      (),
    .empty     (w_rx_empty)           // to tx fifo
);
```

2. Tx FIFO가 여유 있으면 RX_FIFO에서 꺼내 전달

3. Tx FIFO → UART 전송

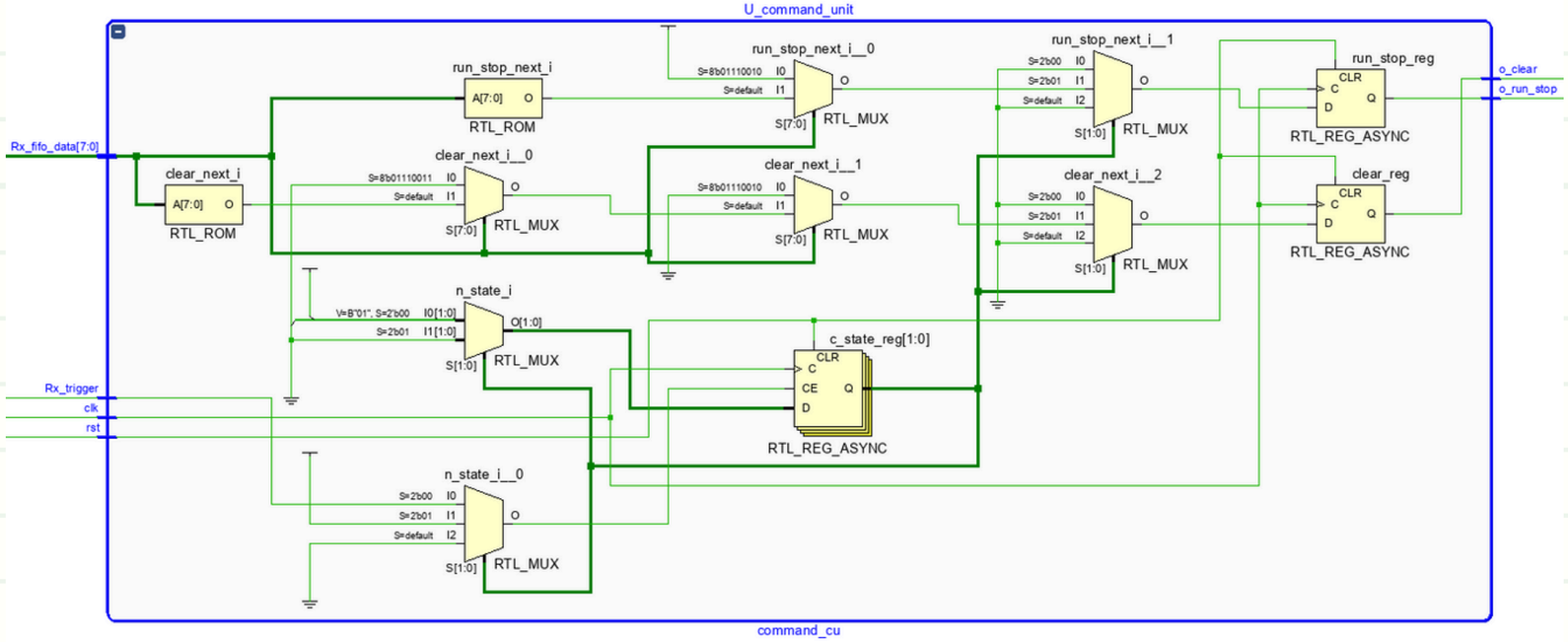
1. rx_done이 올라가면 w_rx_data가 Rx_FIFO에 저장됨

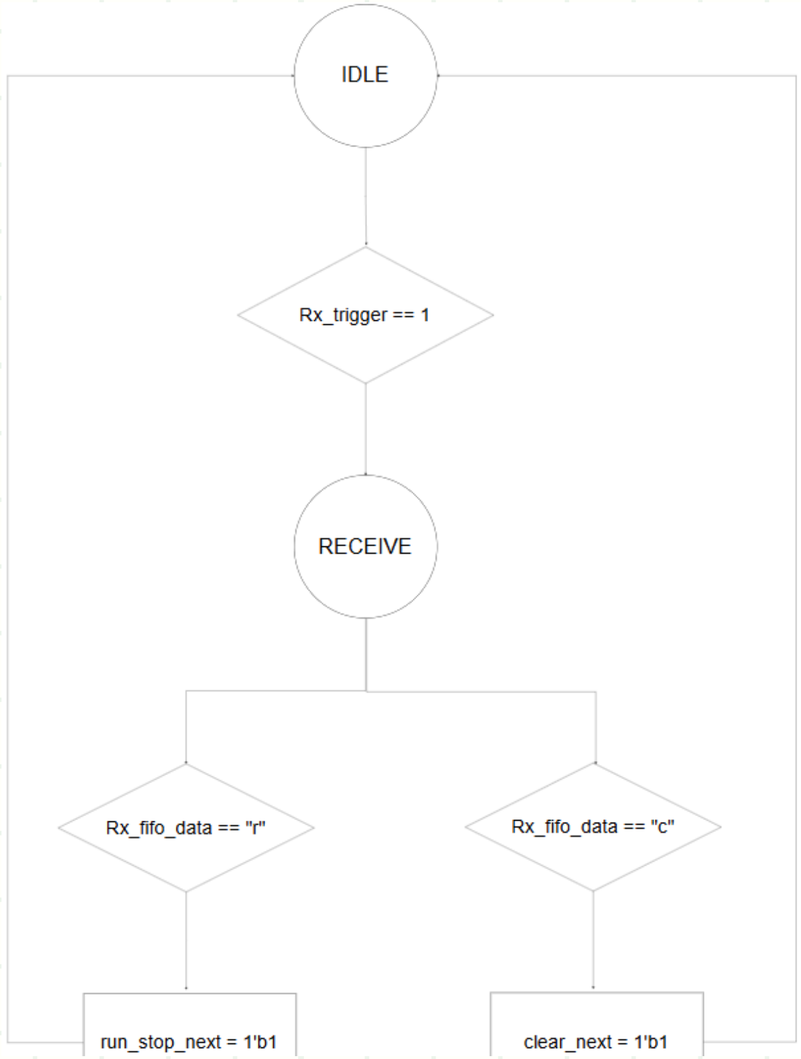
# 03 command_cu 기능




- uart_top에서 Rx_pop데이터를 받아 제어 신호를 만듦
- IDLE과 RECEIVE 두 가지 상태로 구성되어, 명령어 수신 시만 동작하고 그 외에는 대기

# 03 ASM(command_cu )



# 03 코드 리뷰(command_cu)

```
    // next-state logic
    always @(*) begin
        n_state = c_state;
        run_stop_next = 1'b0;
        clear_next = 1'b0;

        case (c_state)
            IDLE: begin
                if (Rx_trigger == 1) begin
                    n_state   = RECEIVE;
                    data_next = Rx_fifo_data;
                end
            end

            RECEIVE: begin
                if (data_reg == 8'h72) begin  // 'r'
                    run_stop_next = 1'b1;
                end else if (data_reg == 8'h63) begin  // 'c'
                    clear_next = 1'b1;
                end
                n_state = IDLE;
            end
        endcase
    end

endmodule
```

IDLE

- 평소엔 대기(IDLE)
- 데이터 도착 신호(Rx_trigger)가 오면 → RECEIVE 상태로 전환, 데이터 저장

RECEIVE

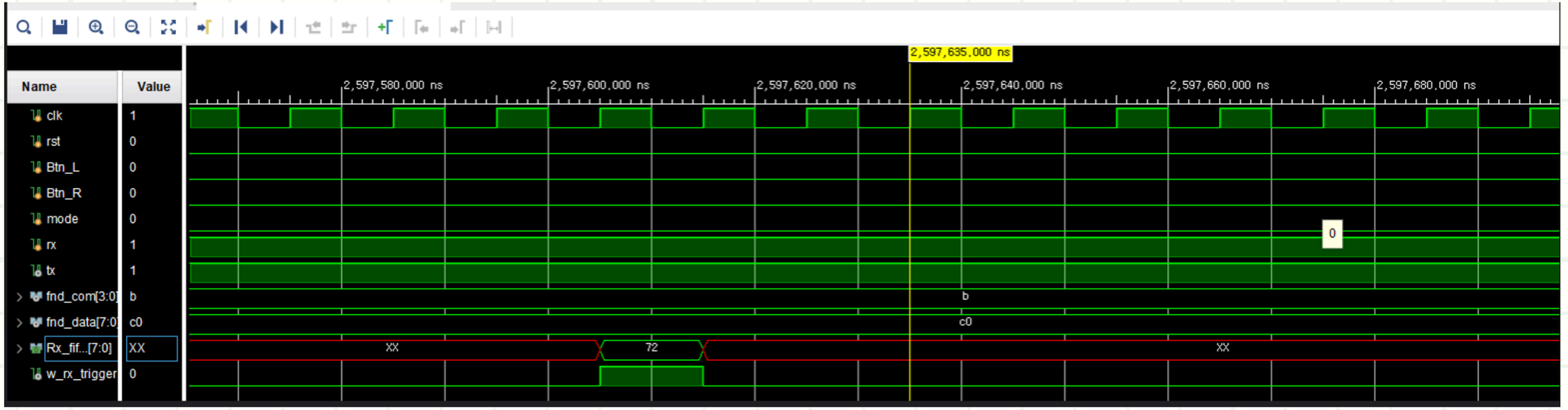
- 'r'(0x72)이면 → run_stop 신호 발생
- 'c'(0x63)이면 → clear 신호 발생
- 처리 끝나면 다시 IDLE로 돌아감

# 03코드 리뷰(stopwatch)

```
wire       w_btn_l;
wire       w_btn_r;
wire       w_rx_trigger;
wire    w_cmd_clear;
wire [7:0] w_fifo_data;  // UART에서 받은 데이터 (8비트)
wire       w_cmd_runstop;  // command_cu 출력 (1비트)
wire       w_runstop_or;  // 버튼 + UART OR 결과 (1비트)

assign w_runstop_or = w_btn_r | w_cmd_runstop;
assign w_cmd_clear_or = w_btn_l | w_cmd_clear;
```

버튼 & UART 명령 통합
→ 버튼 누르거나 UART 명령 받거나, 둘 중 하나만 발생해도 스톱워치 제어 가능.

# 03 트러블 슈팅




Buffer 적용 전 문제점

- rx_trigger 발생 시 rx_fifo_data 값이 X(unknown) 상태
- ComportMaster에서 두 번 눌러야 정상 동작 (첫 입력은 무효)
- 원인: 데이터 갱신 타이밍이 맞지 않아, trigger와 data 불일치 발생

Buffer 적용 후 개선점

- rx_fifo_data를 버퍼(data_reg)에 저장 후 사용
- Trigger 순간의 데이터를 안정적으로 유지
- ComportMaster 한 번 입력만으로 정상 동작
- 타이밍 문제 해결, 데이터 안정성 확보

# 04 SR04

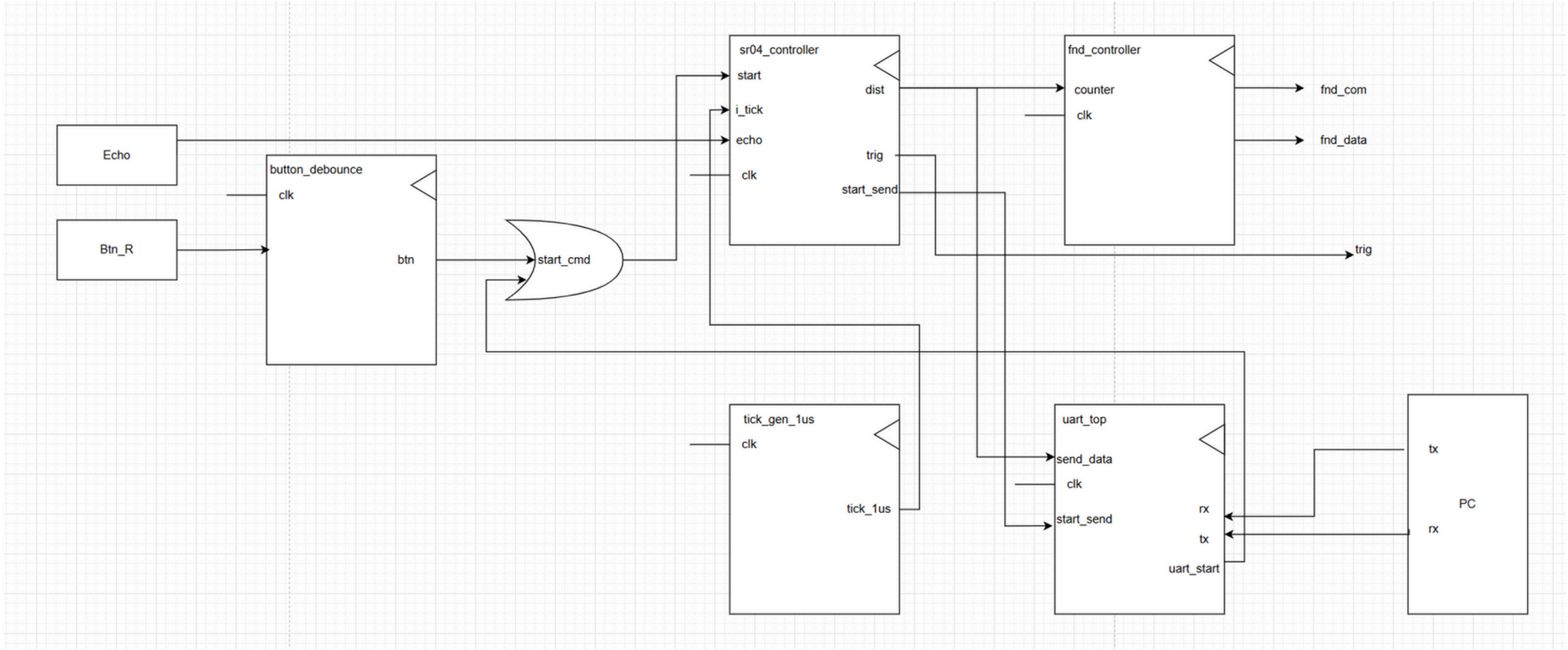
## 프로젝트의 목적 및 목표

- SR04 초음파 센서로 거리를 측정
- 측정된 값을 실시간으로 변환하여 UART 통신을 통해 PC 등 외부 장치로 전송하는 시스템을 구현
- FPGA 상에서 센서 제어, 거리 연산, 데이터 처리, UART 통신을 모두 통합하는 게 핵심 목적.

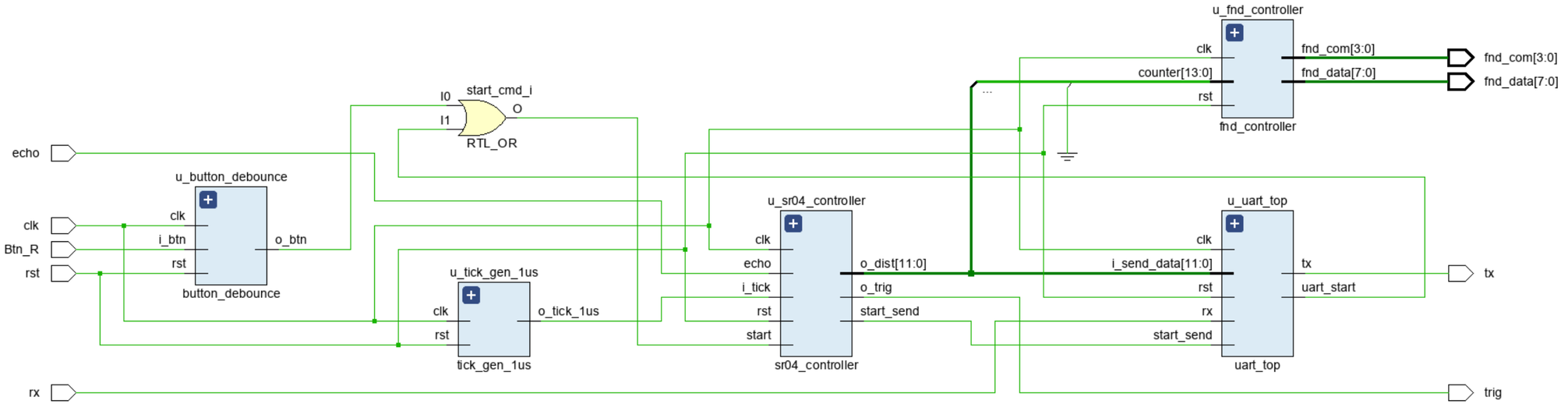
## 프로젝트의 중요성

센서 제어 + UART 통신
단순히 센서값만 읽는 게 아닌, FPGA가 직접 제어, 계산, 통신까지 처리

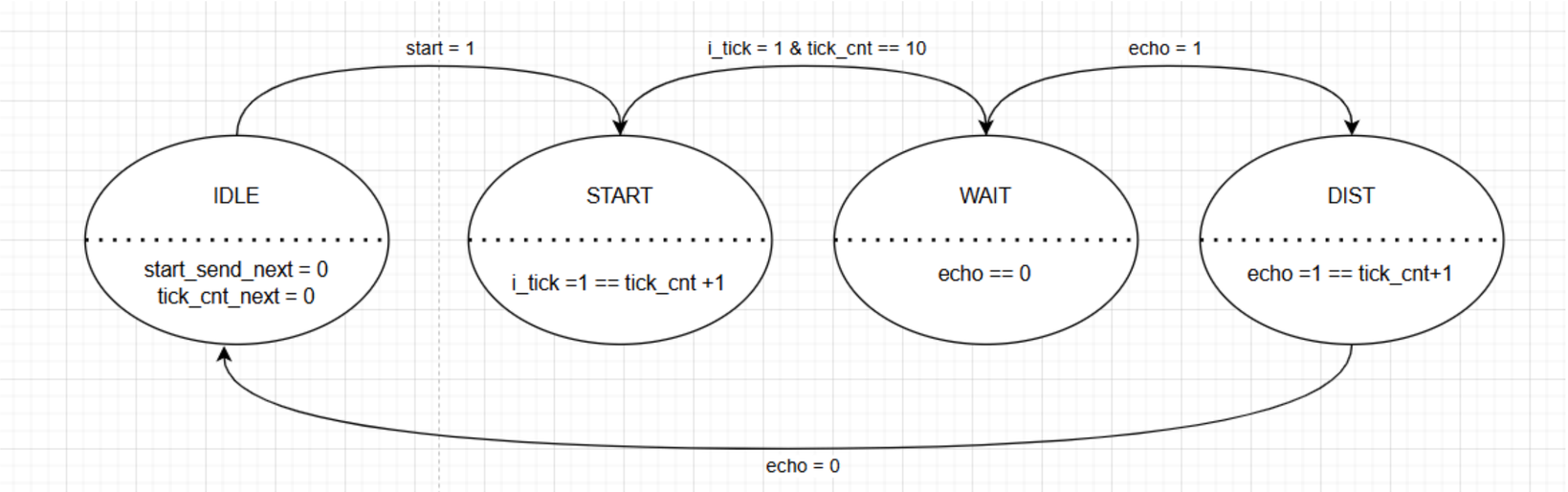
# 04 BLOCK Diagram



# 04 Schematic

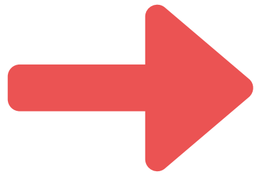


# 04 FSM(SR04_CONTROLLER)



# 04 코드리뷰(SR04_CONTROLLER)

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state     <= IDLE;
        tick_cnt_reg <= 0;
        trig_reg <= 0;
        dist_reg <= 0;
        start_send_reg <= 0;
        dist_div_reg <= 0;
        dist_mul_reg <= 0;
    end else begin
        state <= next;
        trig_reg <= trig_next;
        dist_reg <= dist_next;
        dist_div_reg <= dist_reg / 58;
        tick_cnt_reg <= tick_cnt_next;
        start_send_reg <= start_send_next;
    end
end
```



register update

echo 시간 cm로 변환

# 04 코드리뷰(SR04_CONTROLLER)

```
IDLE: begin
    start_send_next = 0;
    tick_cnt_next   = 0;
    if (start) begin
        next = START;
    end
end
START: begin
    trig_next = 1'b1;
    if (i_tick) begin
        tick_cnt_next = tick_cnt_reg + 1;
        if (tick_cnt_reg == 10) begin
            next = WAIT;
            tick_cnt_next = 0;
        end
    end
end
WAIT: begin
    trig_next = 1'b0;
    if (i_tick) begin
        if (echo) begin
            next = DIST;
```

모든값 초기화
start 시 start state 이동

trig = 1로 유지
tick_cnt = 10이 되면 WAIT로 이동

echo = 1까지 대기
echo 감지 시 dist 상태로 전이

# 04 코드리뷰(SR04_CONTROLLER)

```
DIST: begin
    if (i_tick) begin
        if (echo) begin
            tick_cnt_next = tick_cnt_reg + 1;
        end
        if (!echo) begin
            dist_next = tick_cnt_reg;
            start_send_next = 1'b1;
            next = IDLE;
        end
    end
end
```

echo가 high 인 시간만큼 cnt 증가
LOW가 되면 echo 시간 확정 → 거리 계산
→ UART 전송 요청 후 IDLE 이동

# 04 코드리뷰(SENDER)

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state    <= IDLE;
        send_cnt_reg <= 0;
        send_data_reg <= 0;
        tx_done_reg <= 0;
        push_reg <= 0;
    end else begin
        state <= next_state;
        send_cnt_reg <= send_cnt_next;
        send_data_reg <= send_data_next;
        tx_done_reg <= tx_done_next;
        push_reg <= push_next;
    end
end

always @(*) begin
    next_state          = state;
    send_cnt_next   = send_cnt_reg;
    send_data_next = send_data_reg;
    tx_done_next    = tx_done_reg;
    push_next       = push_reg;
```

CLK마다 FSM상태와 register update

# 04 코드리뷰(SENDER)

```
        case (state)
            IDLE: begin
                tx_done_next = 0;
                send_cnt_next = 0;
                push_next = 0;
                if (start_send) begin
                    next_state = SEND;
                end
            end
            SEND: begin
                if (~full) begin
                    push_next = 1;
                    if (send_cnt_reg < 4) begin
                        case (send_cnt_reg)
                            2'b00: send_data_next = w_send_data[31:24];
                            2'b01: send_data_next = w_send_data[23:16];
                            2'b10: send_data_next = w_send_data[15:8];
                            2'b11: send_data_next = w_send_data[7:0];
                        endcase
                        if (send_cnt_reg < 3) begin
                            send_cnt_next = send_cnt_reg + 1;
                        end else begin
                            next_state = IDLE;
                            tx_done_next = 1'b1;
                        end
                    end
                end else next_state = state;
            end
        endcase
    end
endmodule
```

대기상태 start_send 들어오면 SEND로 전환

full = 0이면 uart fifo 에 push
w_send_data에서 순서대로 4자리 꺼내 전송
다 전송시 IDLE로 돌아가고 rx_done =1

# 04 코드리뷰(data_ascii)

```
module data_ascii (
    input  [13:0] i_data,
    output [31:0] o_data
);

    assign o_data[7:0]   = i_data % 10 + 8'h30;
    assign o_data[15:8]  = (i_data / 10) % 10 + 8'h30;
    assign o_data[23:16] = (i_data / 100) % 10 + 8'h30;
    assign o_data[31:24] = (i_data / 1000) % 10 + 8'h30;
```

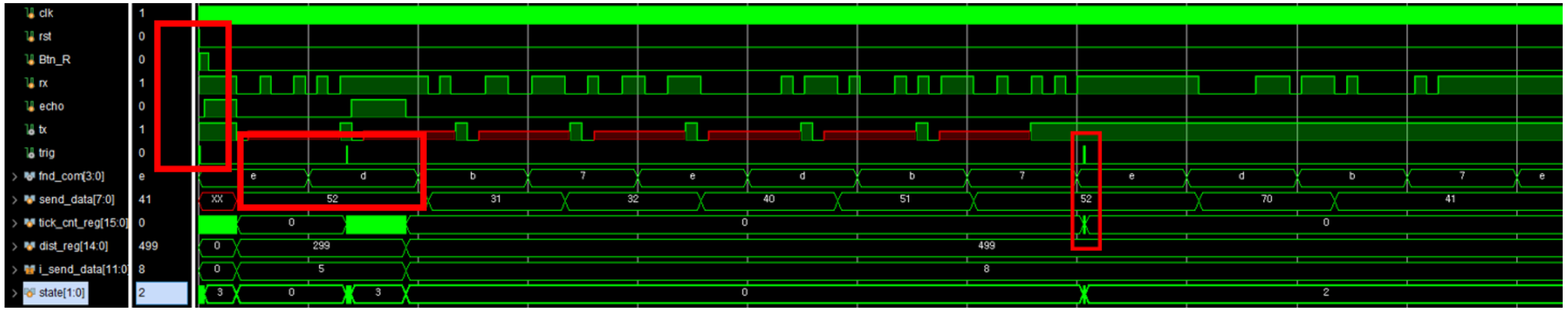
숫자를 Ascii 코드로 변환
LSB부터 1의 자리 순서로 저장

# 04 코드리뷰(command_cu)

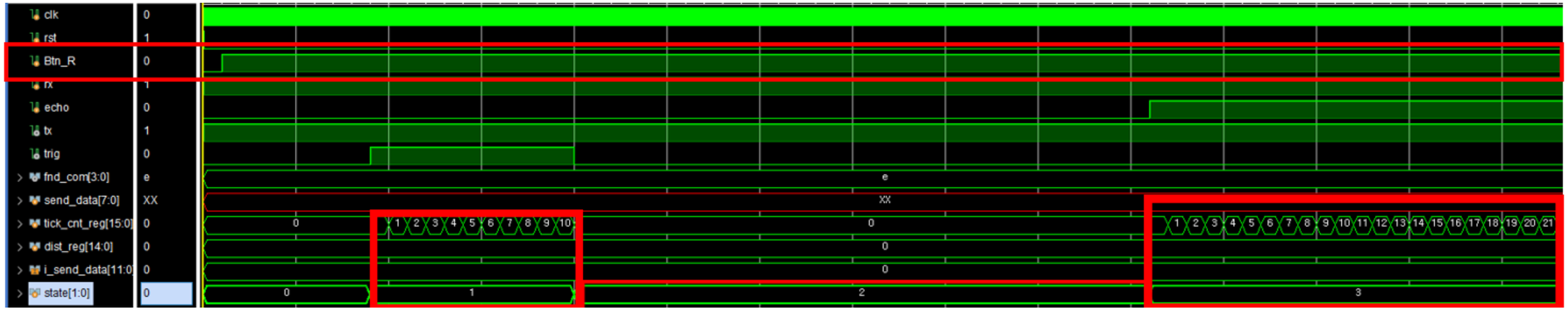
```
module command_cu (
    input clk,
    input rst,
    input rx_trigger,
    input [7:0] rx_fifo_data,
    output start
);
    reg start_reg;

    assign start = start_reg;

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            start_reg <= 1'b0;
        end else begin
            start_reg <= 1'b0;
            if (rx_trigger) begin
                if (rx_fifo_data == 8'h52) begin
                    start_reg <= 1'b1;
                end
            end
        end
    end
```

UART로 들어온 데이터를 해석해서 특정 명령('R')이 들어왔을 때 시작 신호를 발생시키는 명령 제어 유닛

# 04 Testbench



# 04 Testbench

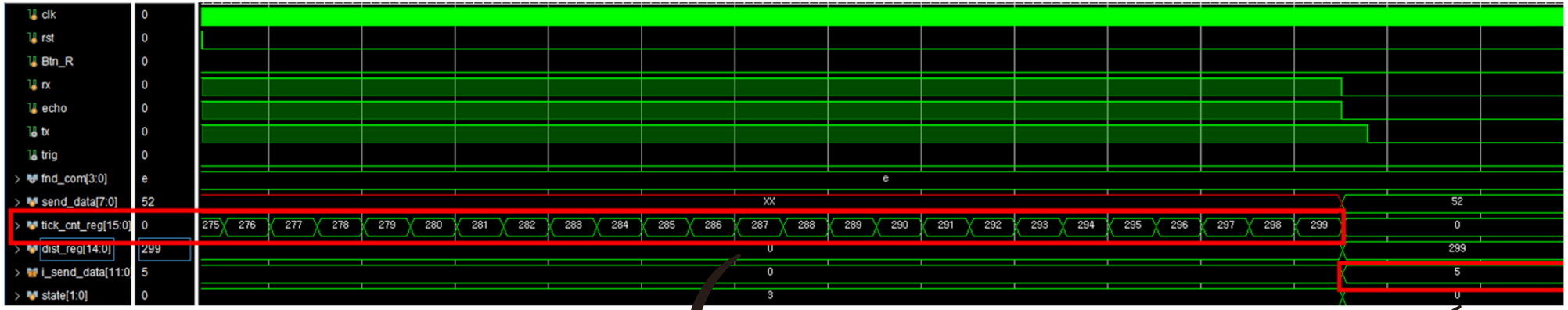



start state
tick_cnt == 10

wait state

dist state

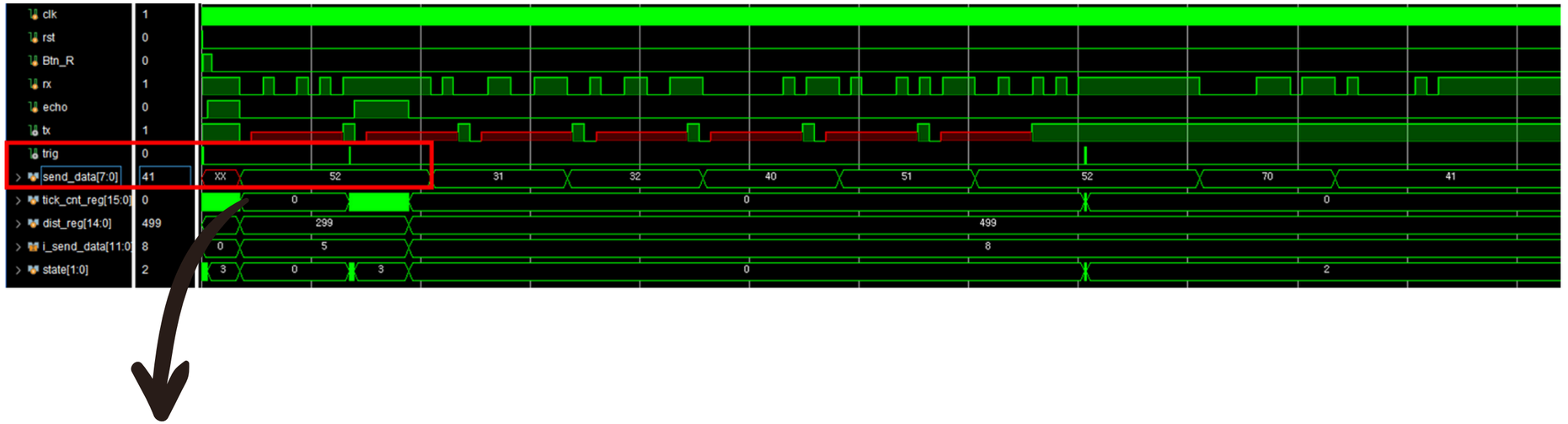
# 04 Testbench




300까지 count

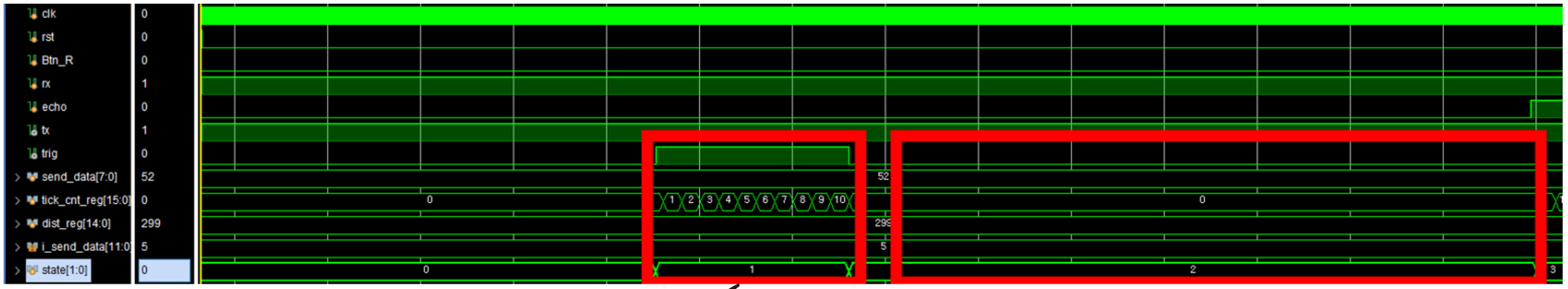
300ms 줬을때
300 / 58 = 5

# 04 Testbench




uart_send ‘R’

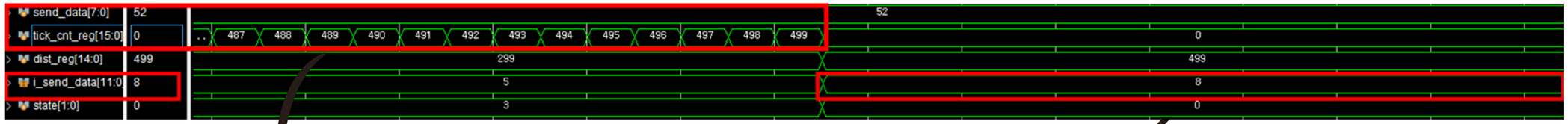
# 04 Testbench




Start state

Wait

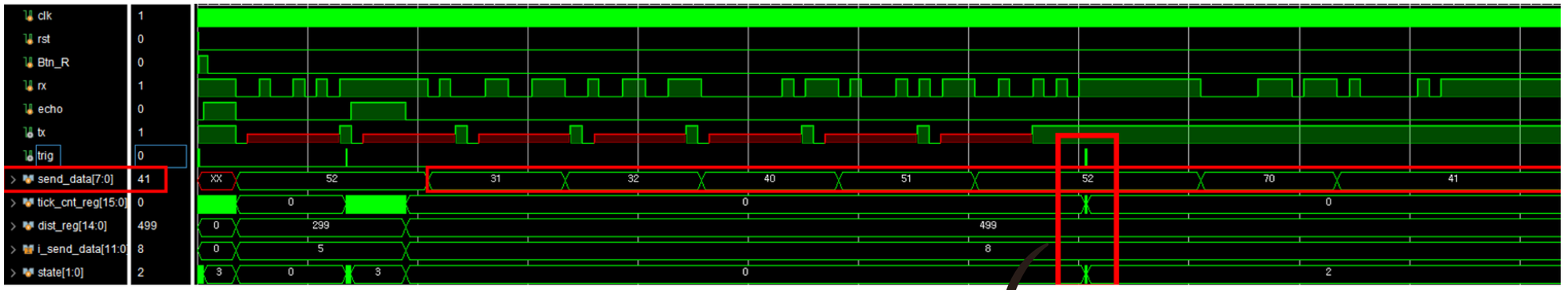
# 04 Testbench




500ms 까지 count

500/58 = 8.62

# 04 Testbench




uart 통신 ‘R’에만 반응

# 05 DHT11

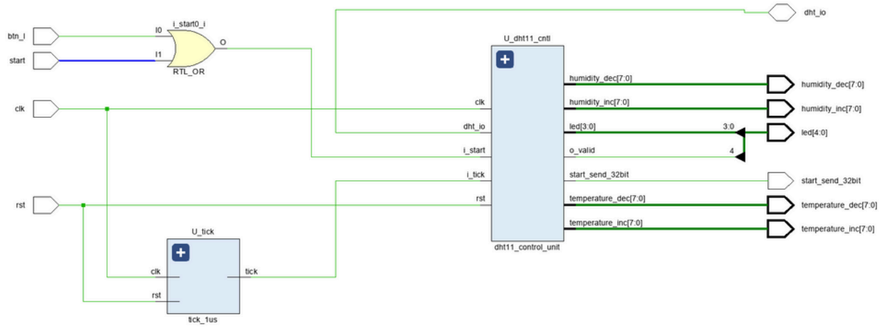
## 프로젝트의 목적 및 목표

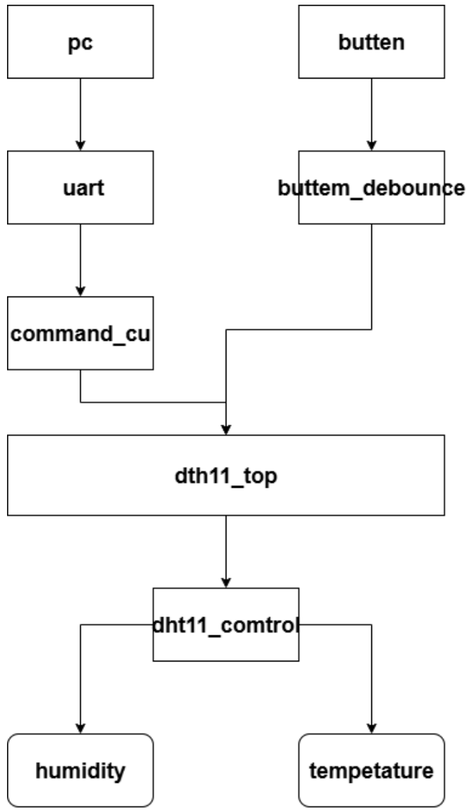
dht11센서로 온도와 습도를 측정하고,
측정된 값을 실시간으로 변환하여 UART 통신을 통해 PC 등 외부 장치로 전송하는 시스템을 구현

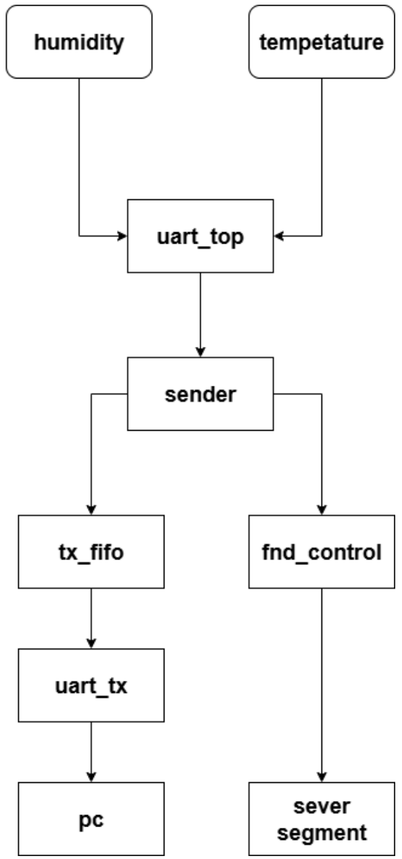
## 프로젝트의 중요성

센서를 하나만 사용하는 것이 아닌 2가지의 센서를 연결하여 각각의 기능을 살려 모듈을 제어하는 능력 향상에 도움

# 05 BLOCK Diagram



# 05 CODE REVIEW

## 중점사항

다음 state로 넘어가기 위한 조건

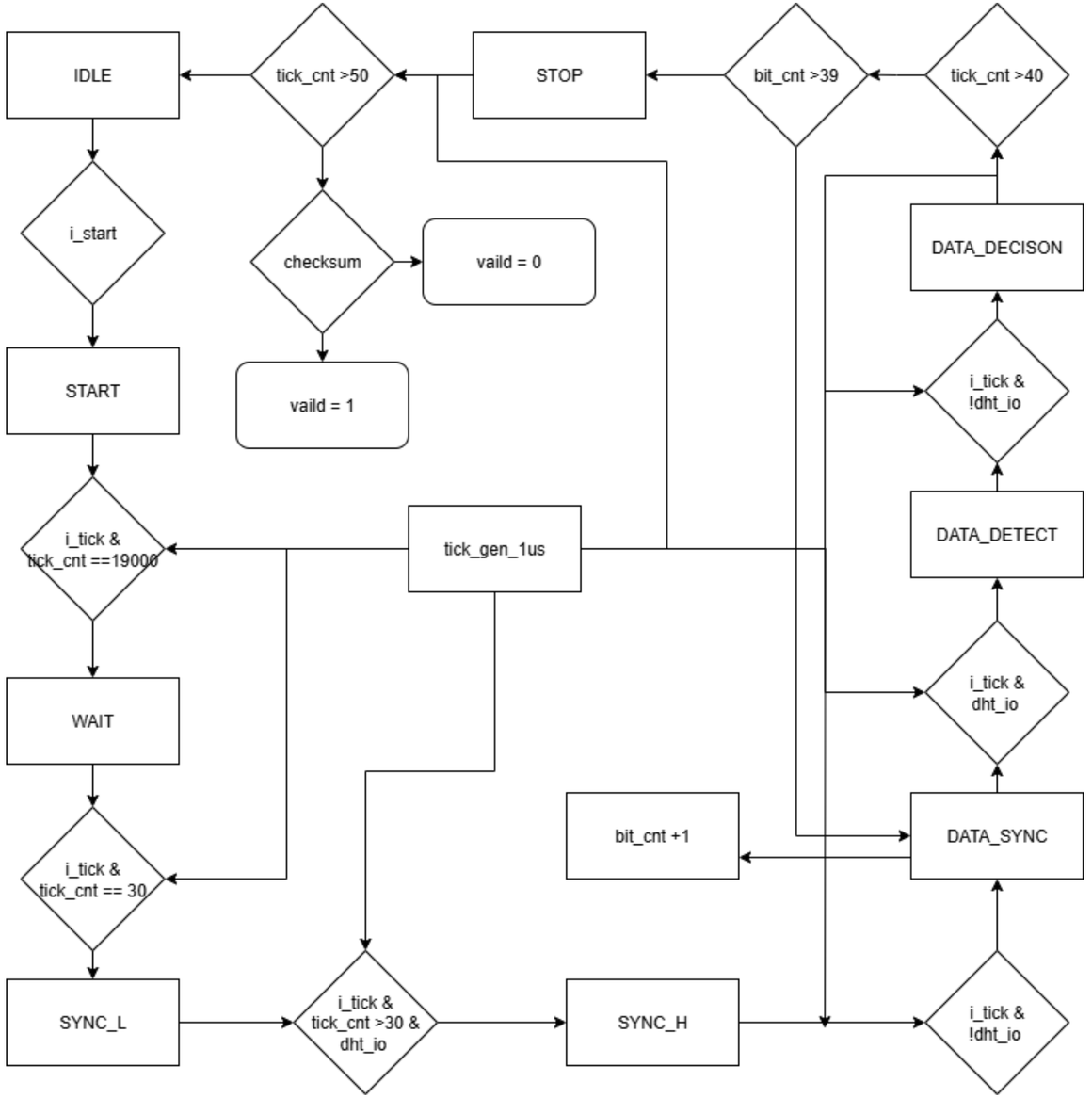
```
always @(*) begin
    state_next         = state_reg;
    dht11_data_next    = dht11_data_reg;
    b_tick_cnt_next    = b_tick_cnt_reg;
    bit_cnt_next       = bit_cnt_reg;
    dht_io_enable_next = dht_io_enable_reg;
    dht_out_next       = dht_out_reg;
    r_vaild_next = r_vaild;
    r_send_next = r_send;
    case (state_reg)
        IDLE: begin
            r_send_next = 1'b0;
            bit_cnt_next = 0;
            dht_out_next = 1'b1;
            if (i_start) begin
                r_vaild_next = 1'b0;
                dht_out_next = 1'b0;
                state_next   = START;
            end
        end
        START: begin
            if (i_tick) begin
                if (b_tick_cnt_reg == 19000) begin
                    dht_out_next = 1'b1;
                    b_tick_cnt_next = 0;
                    state_next = WAIT;
                end else begin
                    b_tick_cnt_next = b_tick_cnt_reg + 1;
                end
            end
        end
        WAIT: begin
            if (i_tick) begin
                if (b_tick_cnt_reg == 30) begin
                    b_tick_cnt_next = 0;
                    dht_io_enable_next = 0;  //fpga changed TX => RX
                    state_next = SYNC_L;
                end else begin
                    b_tick_cnt_next = b_tick_cnt_reg + 1;
                end
            end
        end
```

```
SYNC_L: begin
    if (i_tick) begin
        if (b_tick_cnt_reg > 30) begin
            if (dht_io) begin
                state_next = SYNC_H;
            end
        end else begin
            b_tick_cnt_next = b_tick_cnt_reg + 1;
        end
    end
end
SYNC_H: begin
    if (i_tick) begin
        if (!dht_io) begin
            state_next = DATA_SYNC;
            bit_cnt_next = 0;
        end
    end
end
DATA_SYNC: begin
    if (i_tick) begin
        if (dht_io) begin
            state_next = DATA_DETECT;
            bit_cnt_next = bit_cnt_reg + 1;
        end
    end
end
DATA_DETECT: begin
    if (i_tick) begin
        if (!dht_io) begin
            state_next = DATA_DECISION;
        end else begin
            b_tick_cnt_next = b_tick_cnt_reg + 1;
        end
    end
end
```

# 05 CODE REVIEW

```
    DATA_DECISION: begin
        if (b_tick_cnt_reg > 40) begin
            dht11_data_next[40-bit_cnt_reg] = 1'b1;
        end else begin
            dht11_data_next[40-bit_cnt_reg] = 1'b0;
        end
        b_tick_cnt_next = 0;
        if (bit_cnt_reg > 39) begin
            state_next = STOP;
        end else begin
            state_next = DATA_SYNC;

        end
    end
    STOP: begin
        if (i_tick) begin
            if (b_tick_cnt_reg > 50) begin
                dht_io_enable_next = 1;
                r_send_next = 1'b1;
                if (((dht11_data_reg[39:32] + dht11_data_reg[31:24] + dht11_data_reg[23:16] + dht11_data_reg[15:8]) & 8'hFF) == dht11_data_reg[7:0]) begin
                    r_vaild_next = 1'b1;
                end  else begin
                    r_vaild_next = 1'b0;
                end
                b_tick_cnt_next = b_tick_cnt_reg + 1;
            end
        end
    end
endcase
```

# 05 ASM


IDLE
tick_cnt >50
STOP
bit_cnt >39
tick_cnt >40
i_start
checksum
vaild = 0
DATA_DECISON
START
vaild = 1
i_tick &
!dht_io
i_tick &
tick_cnt ==19000
tick_gen_1us
DATA_DETECT
WAIT
i_tick &
dht_io
bit_cnt +1
DATA_SYNC
i_tick &
tick_cnt == 30
SYNC_L
i_tick &
tick_cnt >30 &
dht_io
SYNC_H
i_tick &
!dht_io

# 05 TEST BENCH

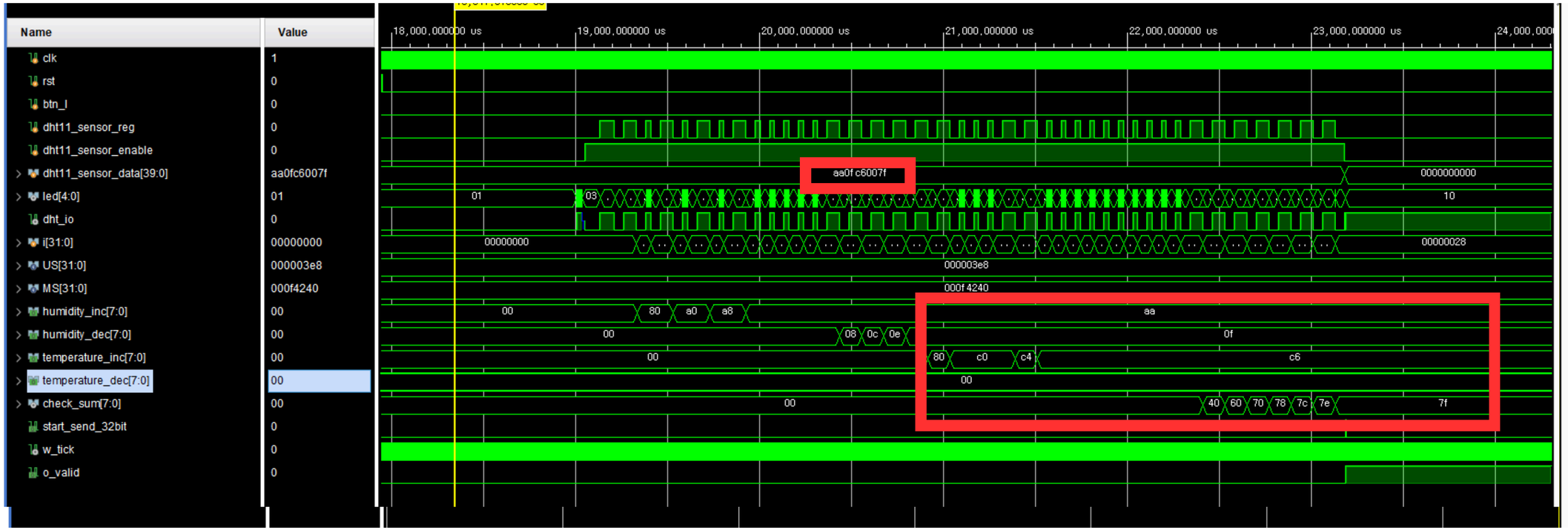
## 시나리오

DHT11이 측정 될수 있는 값중에 VAILD가 1이 나올 수있을 는 값을 넣고 테스트 진행해 VAILD값이 나오는 지 확인

```
`timescale 1ns / 1ps

module tb ();

    parameter US = 1000,MS = 1000000;

    reg clk, rst, btn_l;
    reg dht11_sensor_reg, dht11_sensor_enable;
    reg [39:0] dht11_sensor_data;
    wire [15:0] humidity,temperature;
    wire [4:0] led;
    wire dht_io;

    integer i;

    assign dht_io = (dht11_sensor_enable) ? dht11_sensor_reg : 1'bz;

    dht11_top UUT(
    .clk(clk),
    .rst(rst),
    .btn_l(btn_l),
    .dht_io(dht_io),
    .led(led)
);
    always #5 clk =~clk;

    initial begin
        #0;
        clk =0;
        rst = 1;
        dht11_sensor_enable = 0;
        btn_l = 0;
        dht11_sensor_reg = 0;
        i=0;
        dht11_sensor_data = 40'b10101010_00001111_11000110_00000000_01111111;
        #10;
        rst = 0;
        #10;
        btn_l =1;
        #20000;
        #10; btn_l =0;
        #(19*MS);
        #(30*US);
        dht11_sensor_enable =1;
        #(80*US);
        dht11_sensor_reg = 1;
        #(80*US);

        for (i =0 ;i<40 ;i=i+1 ) begin
            dht11_sensor_reg = 0 ;
            #(50*US);
            dht11_sensor_reg = 1;
            if (dht11_sensor_data[39-i]) begin
                #(70*US);
            end else begin
                #(28*US);
            end
        end
        dht11_sensor_reg = 0;
        #(50*US);
        dht11_sensor_enable =0;
        dht11_sensor_data = 0;
        □#1000;
        $stop;
    end
endmodule
```

# 05 SIMULATION

# 05 SIMULATION



# 06 ALL_top

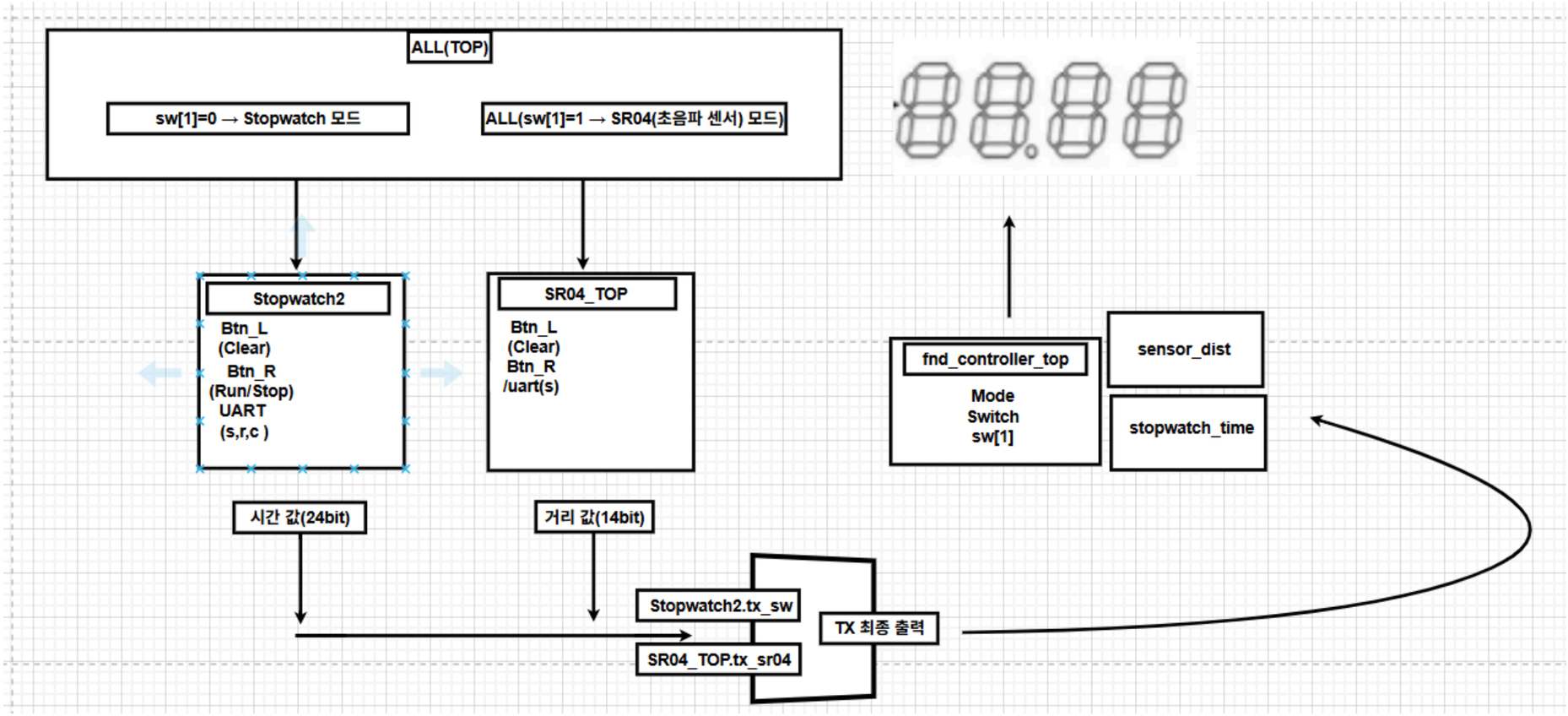
## 프로젝트의 목적 및 목표

Stopwatch 기능과 초음파 센서 온습도 센서 기능을 하나의 FPGA 보드에서 동시에 사용할 수 있도록 설계
UART 통신을 통해 외부 PC와 데이터를 주고받으며, 사용자 제어 및 결과 확인 가능하게 구현
FND를 활용하여 현재 동작 모드(시간/거리)에 따른 값을 직관적으로 표시

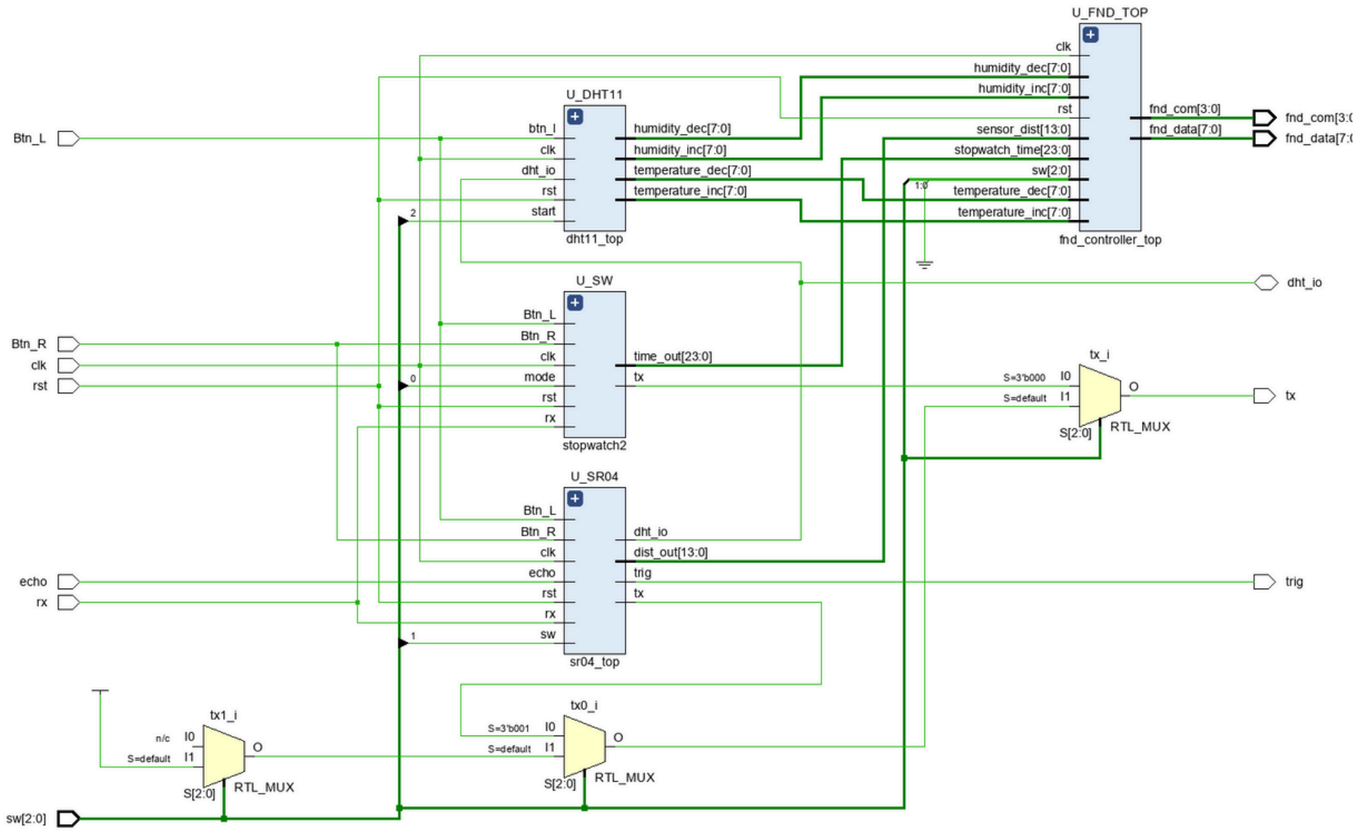
## 목표 기능

모드 전환 (SW[1])
SW[1] = 0 → Stopwatch 모드 동작
SW[1] = 1 → 초음파 센서(SR04) 거리 측정 모드 동작
SW[2] =1 → 온습도 센서(DHT11) 온습도 측정 모드 동작
FND Controller
모드에 따라 입력값 선택
Stopwatch → stopwatch_time
SR04 → sensor_dist
UART TX MUX
모드에 따라 전송 값 선택
Stopwatch → 시간 값
SR04 → 거리 값

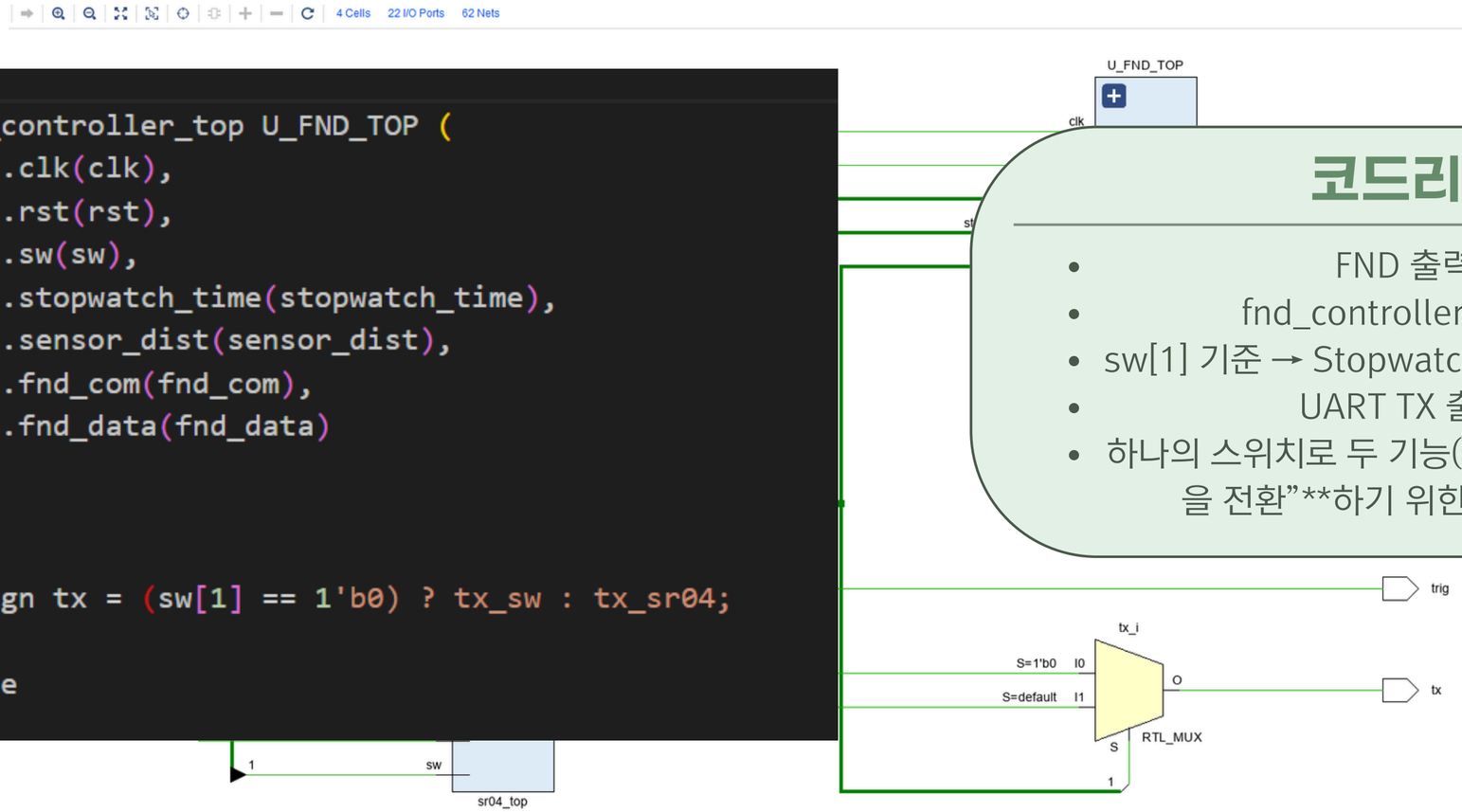
# 06 BLOCK Diagram



# 06 Schematic



# 06 Schematic

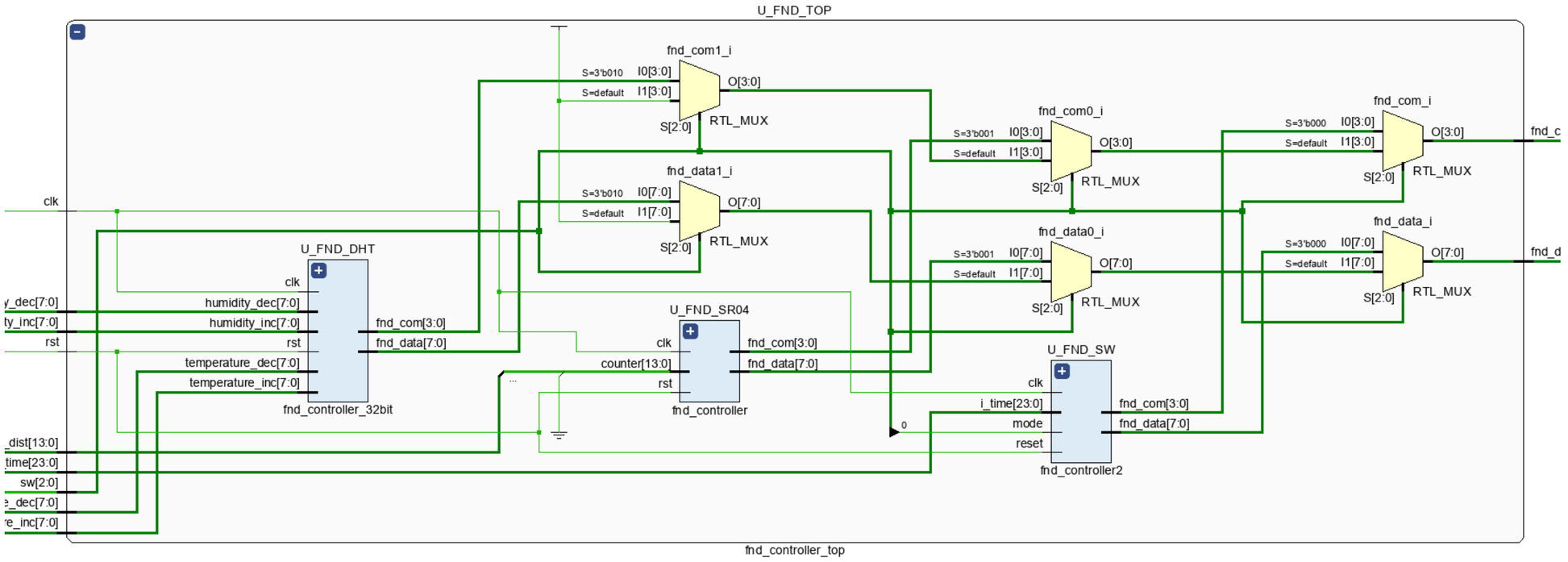



```
    fnd_controller_top U_FND_TOP (
        .clk(clk),
        .rst(rst),
        .sw(sw),
        .stopwatch_time(stopwatch_time),
        .sensor_dist(sensor_dist),
        .fnd_com(fnd_com),
        .fnd_data(fnd_data)
    );


    assign tx = (sw[1] == 1'b0) ? tx_sw : tx_sr04;

endmodule
```

## 코드리뷰

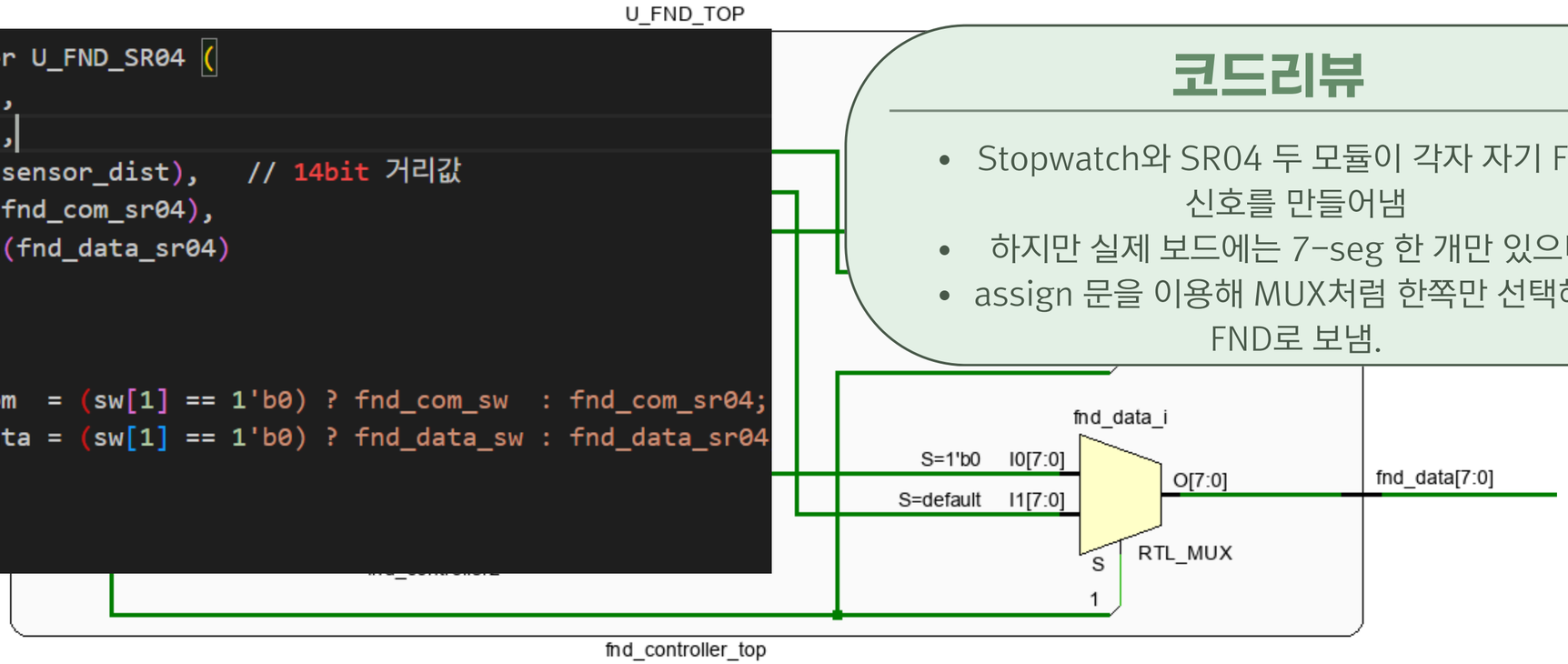
- FND 출력 선택
- fnd_controller_top 안에서
- sw[1] 기준 → Stopwatch 시간 vs 초음파 거리
- UART TX 출력 선택
- 하나의 스위치로 두 기능(Stopwatch, Sensor)을 전환”**하기 위한 최종 MUX 부분

# 06 Schematic



# 06 Schematic

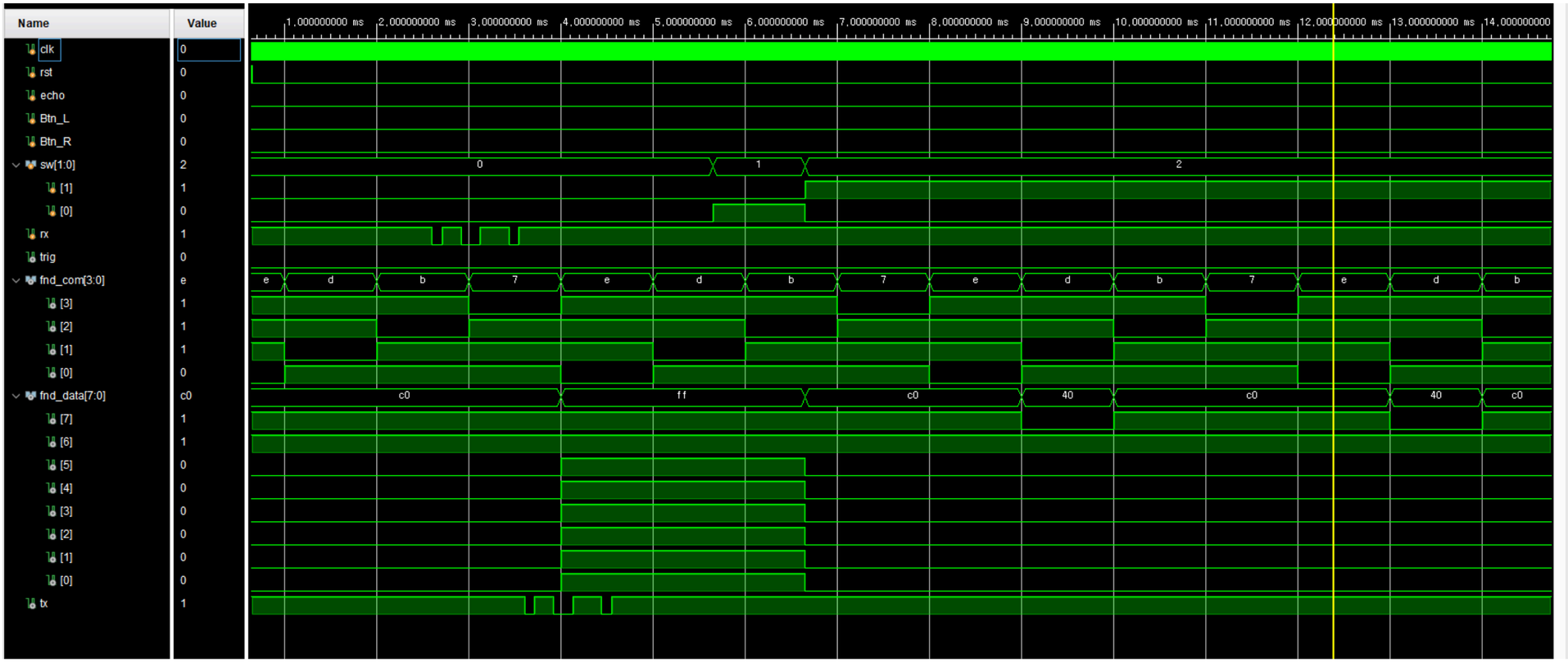
```
    fnd_controller U_FND_SR04 (
        .clk(clk),
        .rst(rst),
        .counter(sensor_dist),   // 14bit 거리값
        .fnd_com(fnd_com_sr04),
        .fnd_data(fnd_data_sr04)
    );


    assign fnd_com  = (sw[1] == 1'b0) ? fnd_com_sw  : fnd_com_sr04;
    assign fnd_data = (sw[1] == 1'b0) ? fnd_data_sw : fnd_data_sr04

module
```



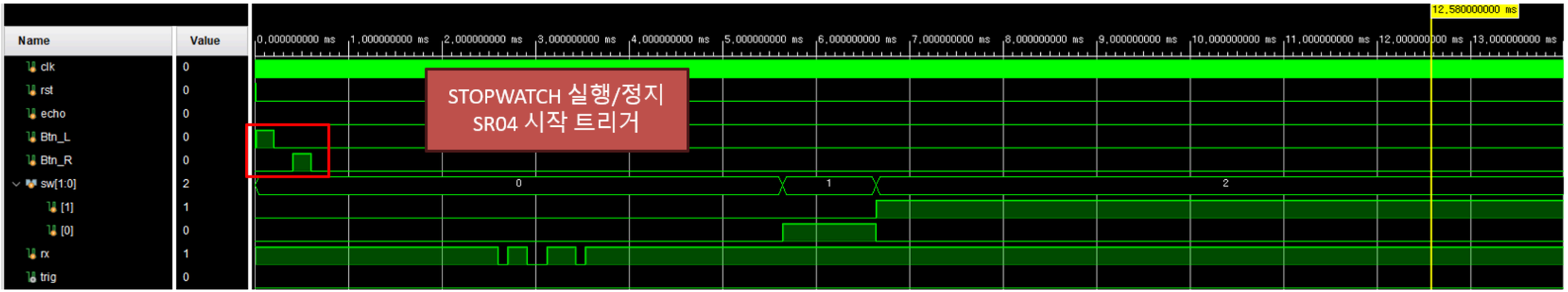

## 코드리뷰

- Stopwatch와 SR04 두 모듈이 각자 자기 FND 신호를 만들어냄
- 하지만 실제 보드에는 7-seg 한 개만 있으니,
- assign 문을 이용해 MUX처럼 한쪽만 선택해서 FND로 보냄.

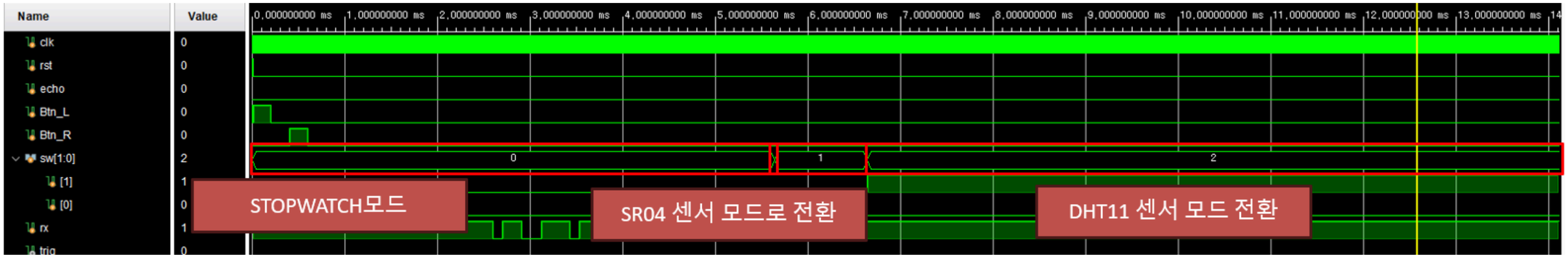
# 06 시뮬레이션


Name
Value
clk
0
rst
0
echo
0
Btn_L
0
Btn_R
0
sw[1:0]
2
[1]
1
[0]
0
rx
1
trig
0
fnd_com[3:0]
e
[3]
1
[2]
1
[1]
1
[0]
0
fnd_data[7:0]
c0
[7]
1
[6]
1
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
tx
1
1,000000000 ms
2,000000000 ms
3,000000000 ms
4,000000000 ms
5,000000000 ms
6,000000000 ms
7,000000000 ms
8,000000000 ms
9,000000000 ms
10,000000000 ms
11,000000000 ms
12,000000000 ms
13,000000000 ms
14,000000000
0
1
2
e d b 7 e d b 7 e d b 7 e d b
c0
ff
c0
40
c0
40
c0

# 06 시뮬레이션


Name
Value
clk
rst
echo
Btn_L
Btn_R
sw[1:0]
[1]
[0]
rx
trig
0
0
0
0
0
2
1
0
1
0
0,000000000 ms
1,000000000 ms
2,000000000 ms
3,000000000 ms
4,000000000 ms
5,000000000 ms
6,000000000 ms
7,000000000 ms
8,000000000 ms
9,000000000 ms
10,000000000 ms
11,000000000 ms
12,000000000 ms
13,000000000 ms
12,580000000 ms
STOPWATCH 실행/정지
SR04 시작 트리거
0
1
2

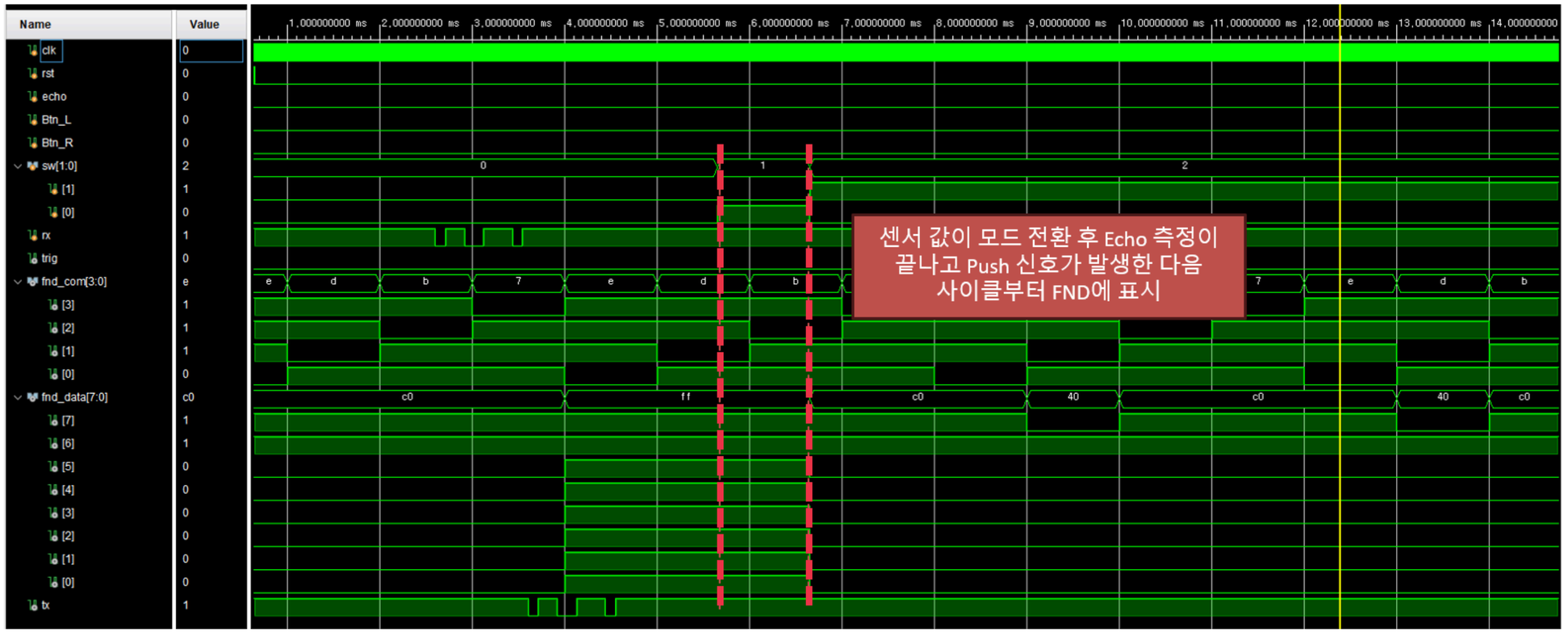
# 06 시뮬레이션


Name
Value
clk
rst
echo
Btn_L
Btn_R
sw[1:0]
[1]
[0]
rx
trig
0,000000000 ms
1,000000000 ms
2,000000000 ms
3,000000000 ms
4,000000000 ms
5,000000000 ms
6,000000000 ms
7,000000000 ms
8,000000000 ms
9,000000000 ms
10,000000000 ms
11,000000000 ms
12,000000000 ms
13,000000000 ms
0
1
2
STOPWATCH모드
SR04 센서 모드로 전환
DHT11 센서 모드 전환

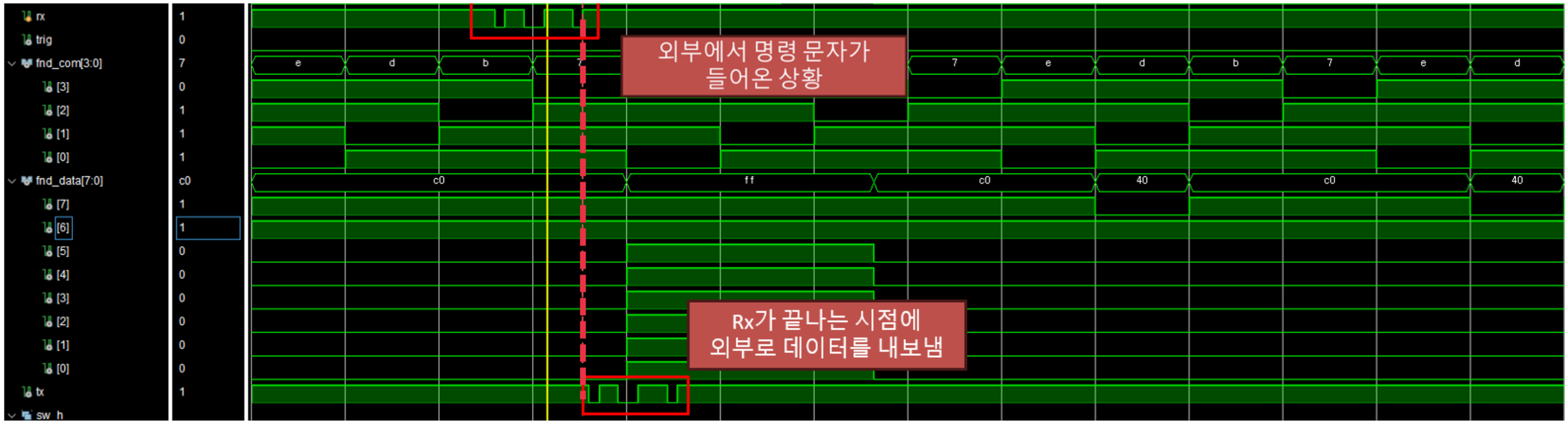
# 06 시뮬레이션


rx
trig
fnd_com[3:0]
[3]
[2]
[1]
[0]
fnd_data[7:0]
[7]
[6]
1
0
7
0
1
1
1
c0
1
1
e d b 7 e d b 7 e d b 7 e d
c0 ff c0 40 c0 40
FND 자리 선택 정상 출력

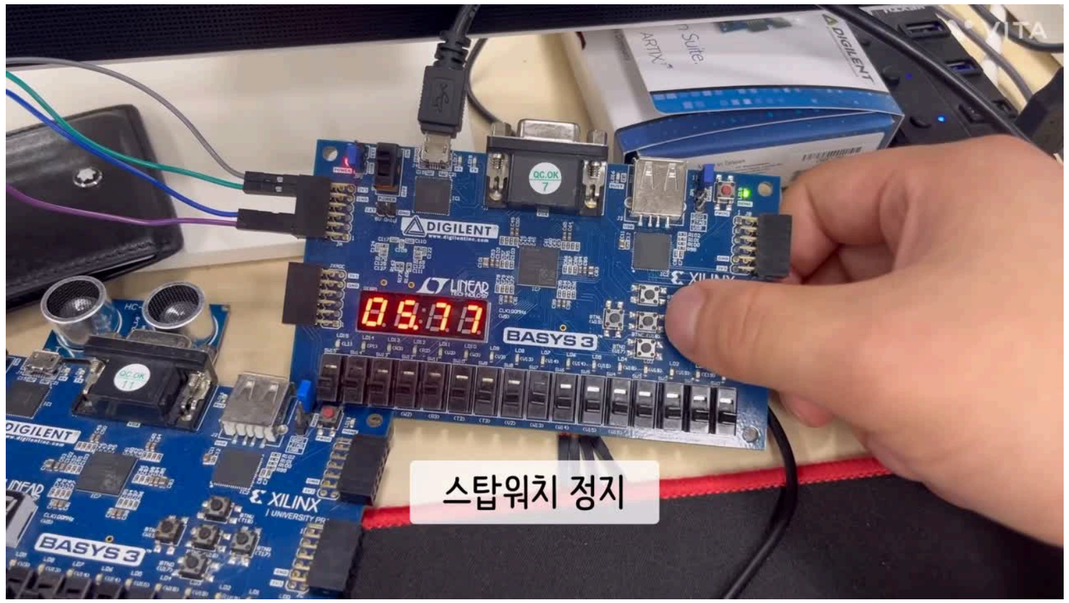
# 06 시뮬레이션


Name
Value
clk
rst
echo
Btn_L
Btn_R
sw[1:0]
[1]
[0]
rx
trig
fnd_com[3:0]
[3]
[2]
[1]
[0]
fnd_data[7:0]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
tx
센서 값이 모드 전환 후 Echo 측정이 끝나고 Push 신호가 발생한 다음 사이클부터 FND에 표시

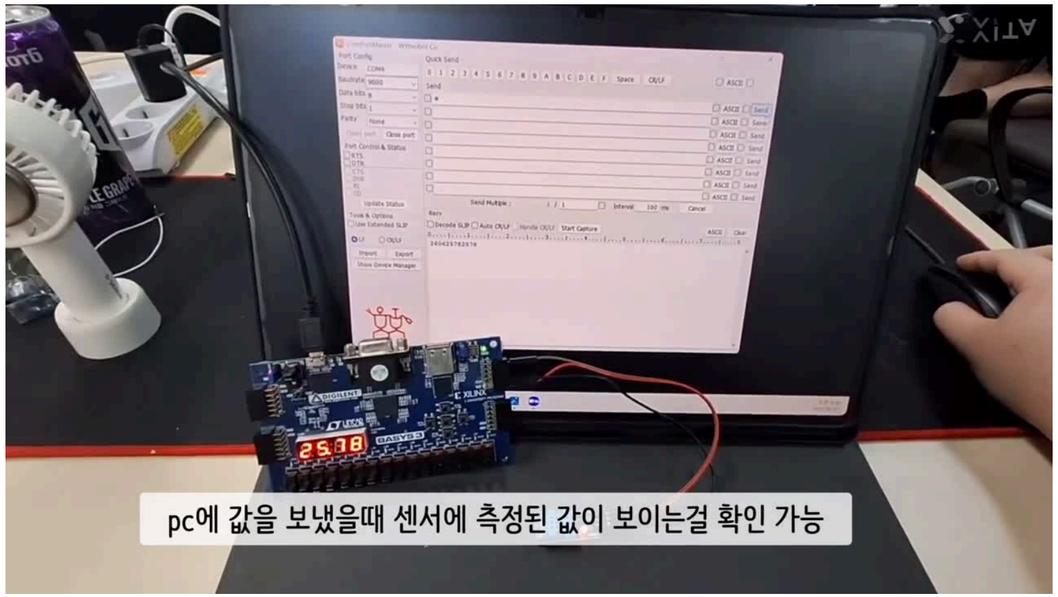
# 06 시뮬레이션


rx
trig
fnd_com[3:0]
[3]
[2]
[1]
[0]
fnd_data[7:0]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
tx
sw_h
1
0
7
0
1
1
1
c0
1
1
0
0
0
0
0
0
1
e
d
b
7
c0
ff
40
외부에서 명령 문자가
들어온 상황
Rx가 끝나는 시점에
외부로 데이터를 내보냄

# 06 동작영상




STOPWATCH/SR04(초음파센서)




DHT11온습도센서