UART + Sensor + WATCH/STOPWATCH

하만세미콘아카데미 2기

4조 김은성 ooo ooo ooo

2025.08.28

목차

Intro

- 목적

FPGA 보드를 이용한 Watch/Stopwatch, DHT11, HC-SR04 시스템 설계 및 구현

PC와 FPGA 보드 간의 UART 통신 시스템 설계 및 구현

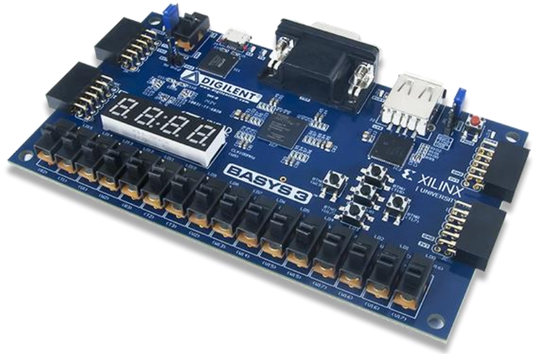
- 주요 기능

1. Watch / Stopwatch
2. HC-SR04 센서를 이용한 거리 측정
3. DHT11 센서를 이용한 온도/습도 측정
4. 7 segment display에 시간 정보를 시각화
5. UART 및 모든 기능 통합

Intro

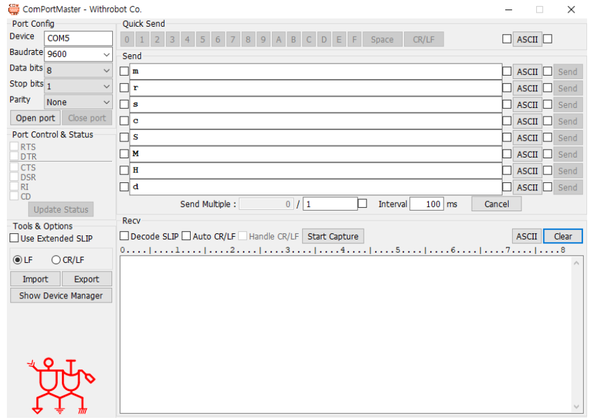
Vivado 2020.2

AMD-Xilinx의
FPGA 통합 설계 환경(IDE)



Basys 3

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



COMPORTMASTER

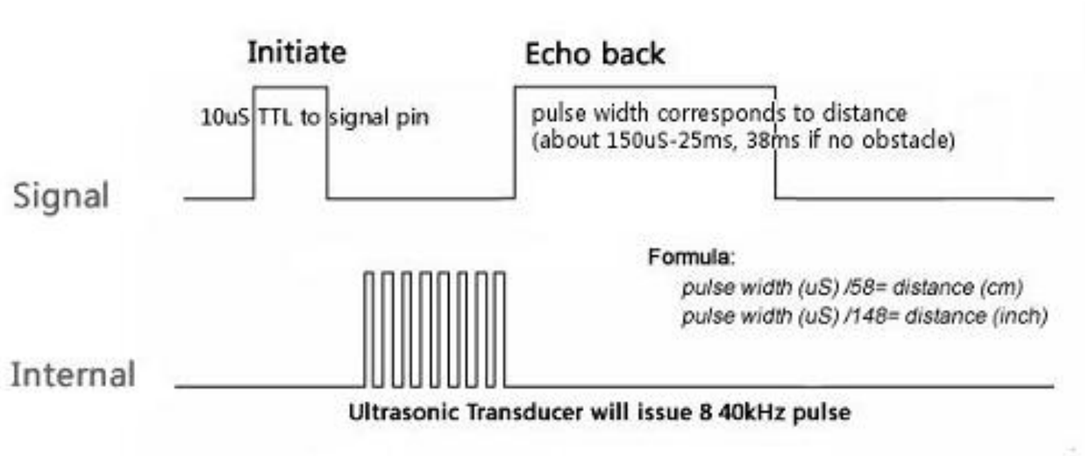
PC의 시리얼 포트(COM) 통신을 위한
터미널 프로그램

Sensor

1. HC-SR04

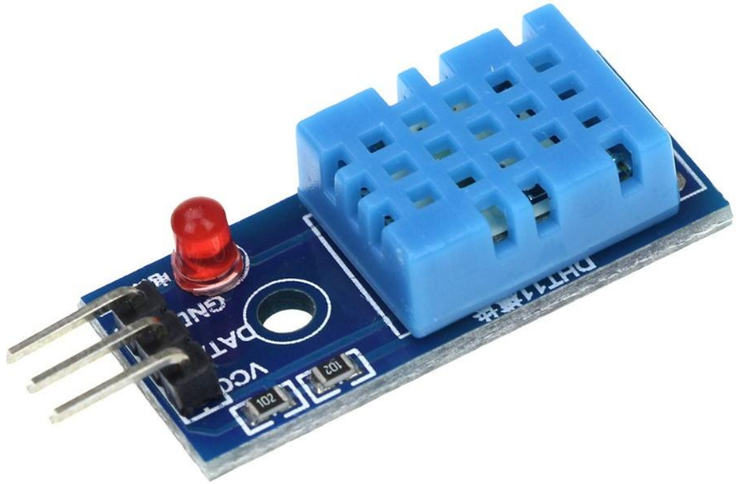
- Trig에서 초음파를 발사하고, 물체에 맞고 되돌아온 초음파를 Echo로 수신.
  걸린 시간으로 거리 계산.

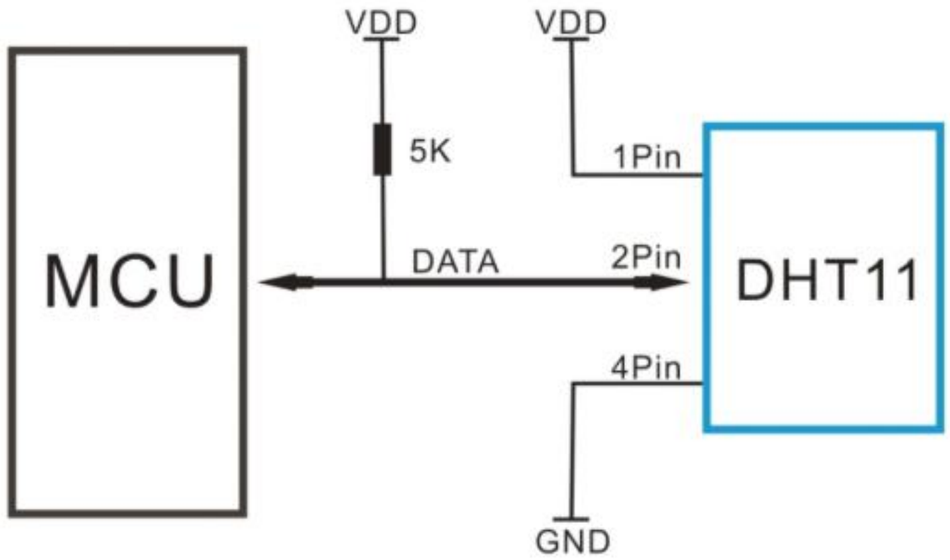
계산식 : 걸린 시간(us) / 58 = 거리(cm)



Sensor

2. DHT11



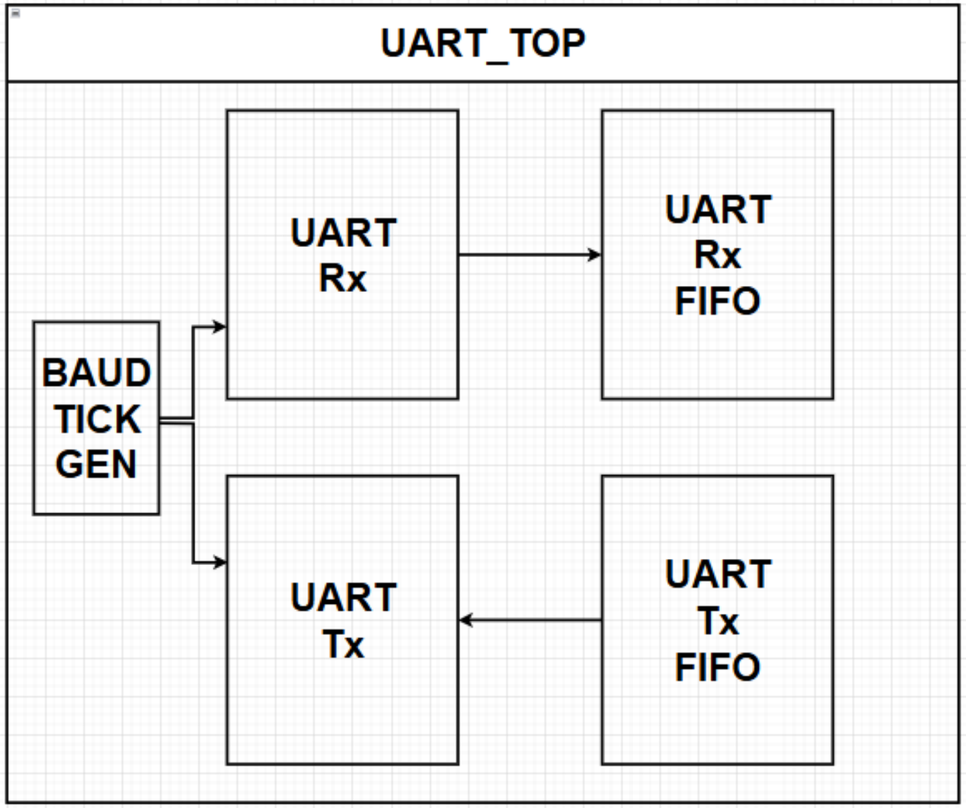


- 공기 중의 온도와 상대 습도를 측정하는 디지털 센서

- 온도 측정 범위 : 0 ~ 50°C (정확도: ±2°C)
- 습도 측정 범위 : 20 ~ 90% (정확도: ±5%)
- 샘플링 주기 : 1Hz

- 감지한 아날로그 값을 내부 칩에서 40비트의 디지털 데이터 패킷으로 변환.
- 단일 데이터 핀을 통해 측정된 값(습도 정수/소수, 온도 정수/소수)과 오류 검증을 위한 Checksum을 출력

UART


UART_TOP
UART
Rx
UART
Rx
FIFO
BAUD
TICK
GEN
UART
Tx
UART
Tx
FIFO

Sender

```
3   module sender_uart (
4       input         clk,
5       input         rst,
6       input         start_send,
7       input  [13:0] i_send_data,
8       input         full,
9       output        push,
10      output        tx_done,
11      output [ 7:0] send_data
12  );
13
14      wire [31:0] w_send_data;
15
16      reg [1:0] state, next;
17      reg [2:0] send_cnt_reg, send_cnt_next;
18
19      reg push_reg, push_next;
20      reg tx_done_reg, tx_done_next;
21      reg [7:0] send_data_reg, send_data_next;
22
23      assign push = push_reg;
24      assign tx_done = tx_done_reg;
25      assign send_data = send_data_reg;
26
27      localparam IDLE = 2'b00, SEND = 2'b01;
```

- **입력 (Inputs)**

start_send : 데이터 전송 시작 펄스 신호

i_send_data[13:0]: 전송할 원본 숫자 데이터

full : 후행 모듈인 TX_FIFO가 찼는지 알려주는 상태 신호

- **출력 (Outputs)**

push : 변환된 ASCII 문자 데이터(send_data)를 TX_FIFO에 저장하라는 제어 신호

tx_done : 4개의 문자 전송이 모두 완료되었음을 알리는 신호

send_data[7:0] : TX_FIFO로 전달될 8비트 ASCII 문자 데이터

Sender

```
51      always @(*) begin
52          next           = state;
53          send_cnt_next  = send_cnt_reg;
54          send_data_next = send_data_reg;
55          tx_done_next   = tx_done_reg;
56          push_next      = push_reg;
57          case (state)
58              IDLE: begin
59                  tx_done_next = 0;
60                  send_cnt_next = 0;
61                  push_next = 0;
62                  if (start_send) begin
63                      next = SEND;
64                  end
65              end
66              SEND: begin
67                  if (~full) begin
68                      push_next = 1;
69                      if (send_cnt_reg < 4) begin
70                          case (send_cnt_reg)
71                              2'b00: send_data_next = w_send_data[31:24];
72                              2'b01: send_data_next = w_send_data[23:16];
73                              2'b10: send_data_next = w_send_data[15:8];
74                              2'b11: send_data_next = w_send_data[7:0];
75                          endcase
76                          if (send_cnt_reg < 3) begin
77                              send_cnt_next = send_cnt_reg + 1;
78                          end else begin
79                              next = IDLE;
80                              tx_done_next = 1'b1;
81                          end
82                      end
83                  end else next = state;
84              end
85          endcase
86      end
87  endmodule
```

•IDLE :
start_send 입력이 1이 되는 것을 감지하면 SEND 상태로 전환.

• SEND :
send_cnt_reg 카운터를 0부터 3까지 1씩 증가시키며 4번 반복.

각 카운트 값에 맞춰 case 문이 32비트 w_send_data에서 보낼 8비트 ASCII 문자를 하나 선택하여 send_data_next에 할당.

TX_FIFO가 꽉 차지 않았다면, push_next를 1로 만들어 선택된 문자를 FIFO에 저장.

send_cnt_reg가 마지막 값인 3에 도달하면, tx_done_next를 1로 설정하여 전송 완료를 알리고 다음 상태(next)를 IDLE로 복귀

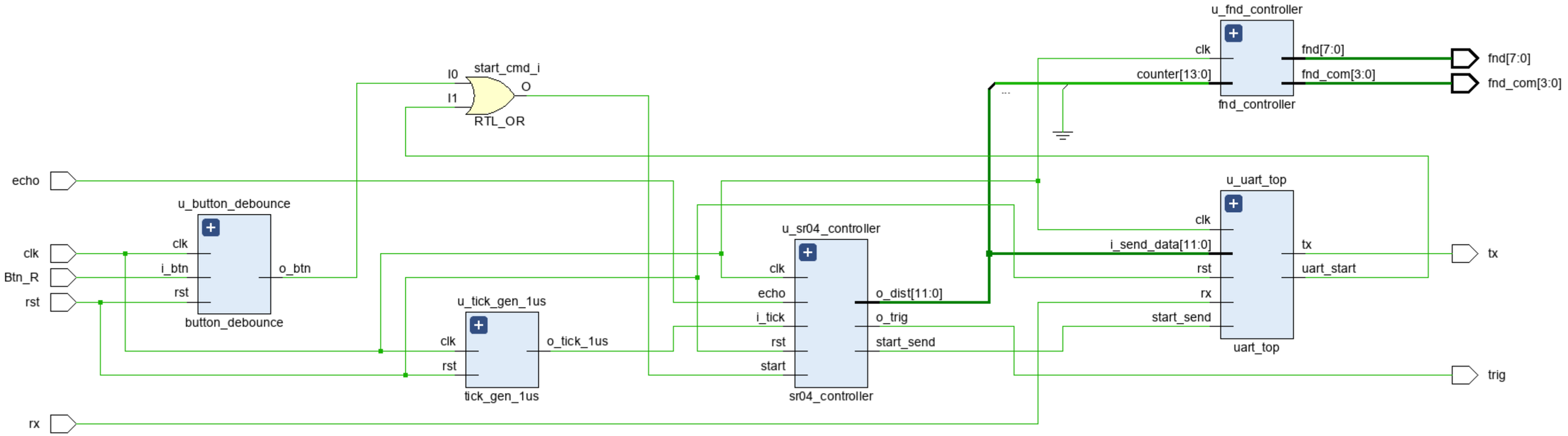
Sender

```
90  module datatoascii (
91      input  [13:0] i_data,
92      output [31:0] o_data
93  );
94
95      assign o_data[7:0]   = i_data % 10 + 8'h30;
96      assign o_data[15:8]  = (i_data / 10) % 10 + 8'h30;
97      assign o_data[23:16] = (i_data / 100) % 10 + 8'h30;
98      assign o_data[31:24] = (i_data / 1000) % 10 + 8'h30;
99
100 endmodule
```

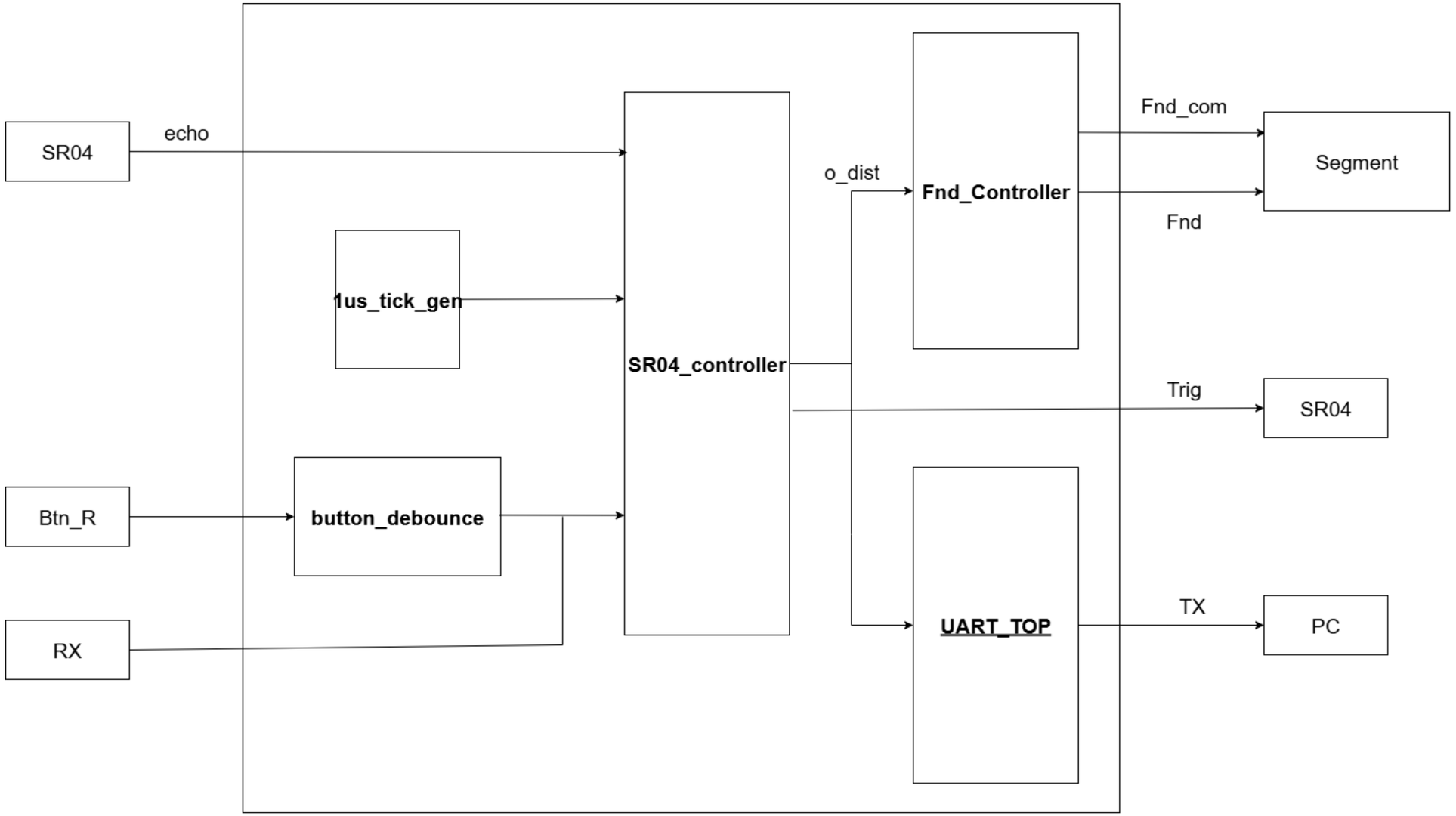
SR04

SR04 + FND Controller

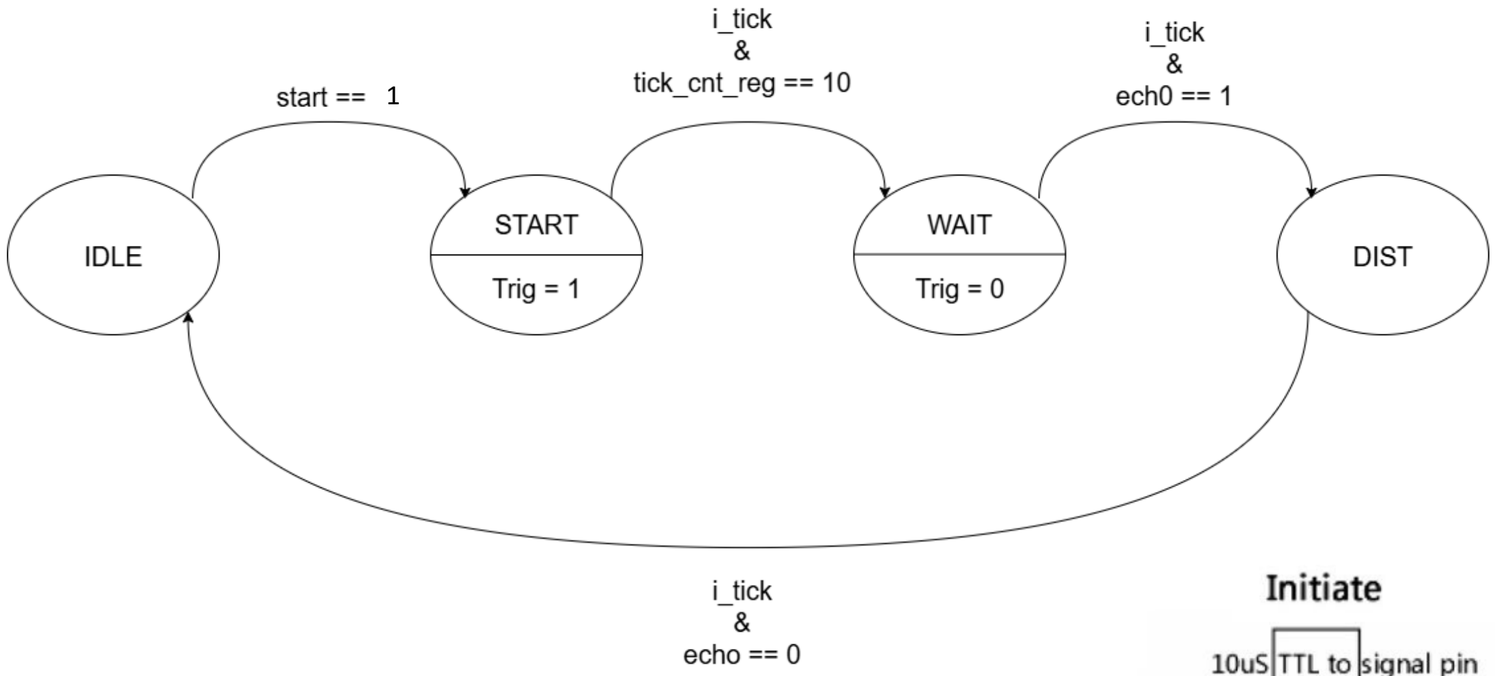
SR04 Schematic

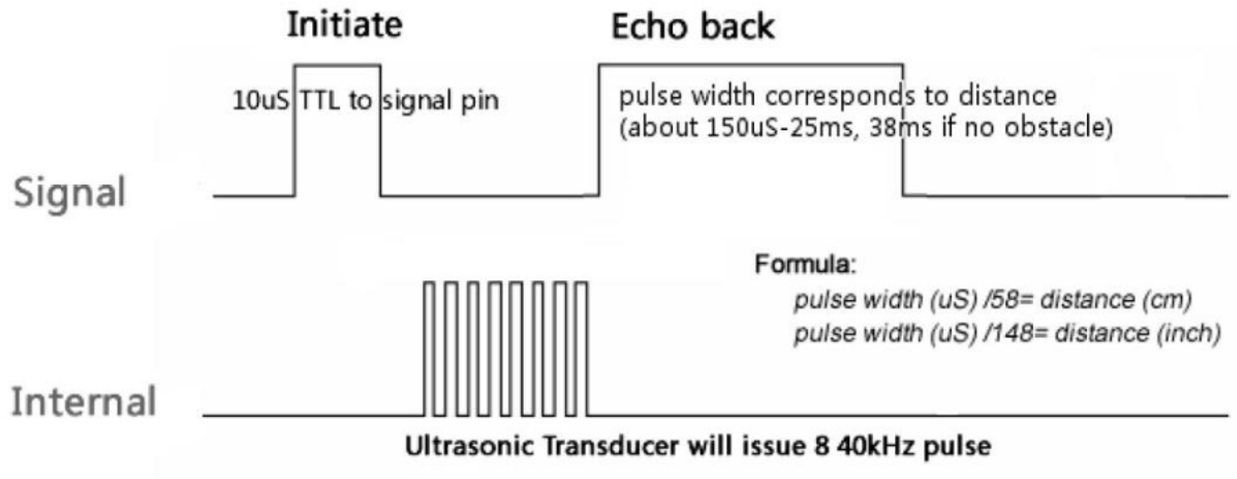


SR04 Diagram



SR04 FSM


start == 1
i_tick
&
tick_cnt_reg == 10
i_tick
&
ech0 == 1
IDLE
START
Trig = 1
WAIT
Trig = 0
DIST
i_tick
&
echo == 0


Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

SR04 ASM & Code

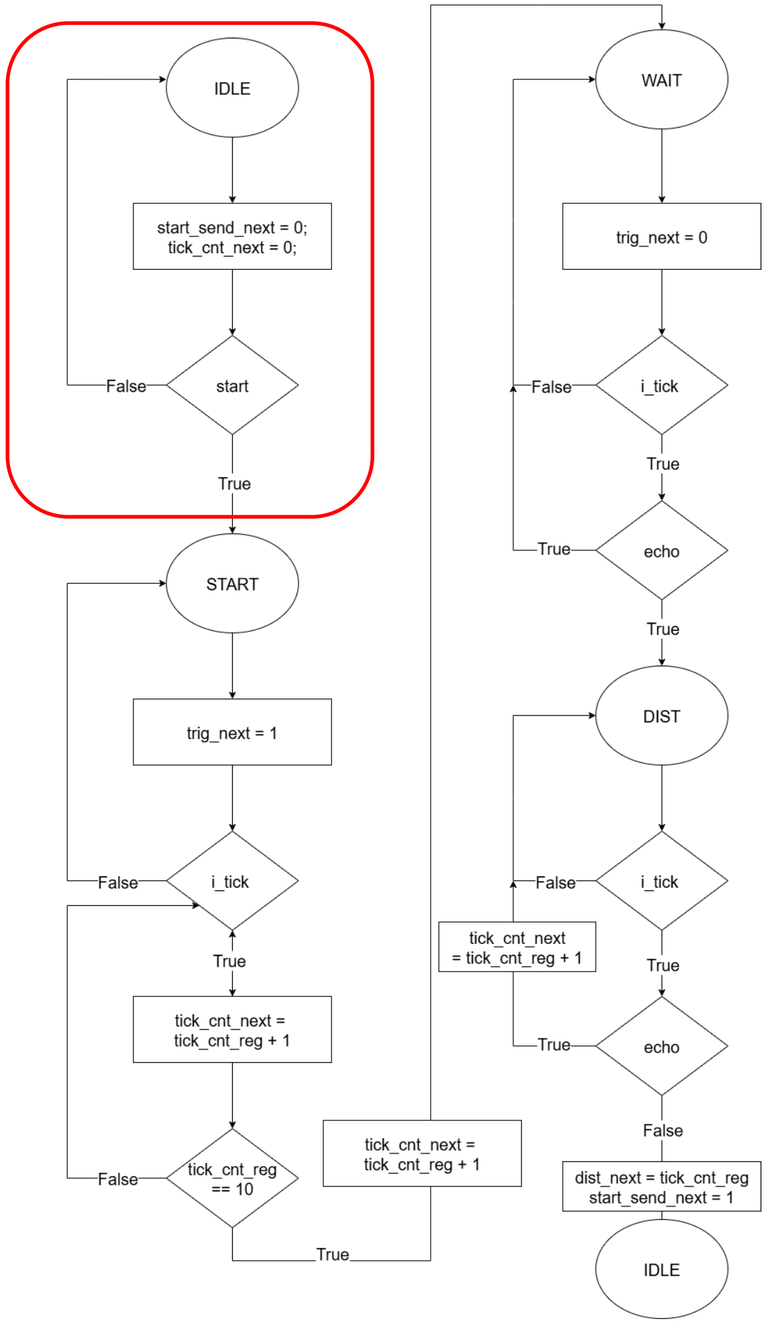
```
always @(*) begin
    case (state)
        IDLE: begin
            start_send_next = 0;
            tick_cnt_next   = 0;
            if (start) begin
                next = START;
            end
        end
        START: begin
            trig_next = 1'b1;
            if (i_tick) begin
                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;
                end
            end
        end
        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
    endcase
end
```

Start_send_next = 0

-> sender 정지상태

Tick_cnt_next = 0

-> tick_cnt 초기화

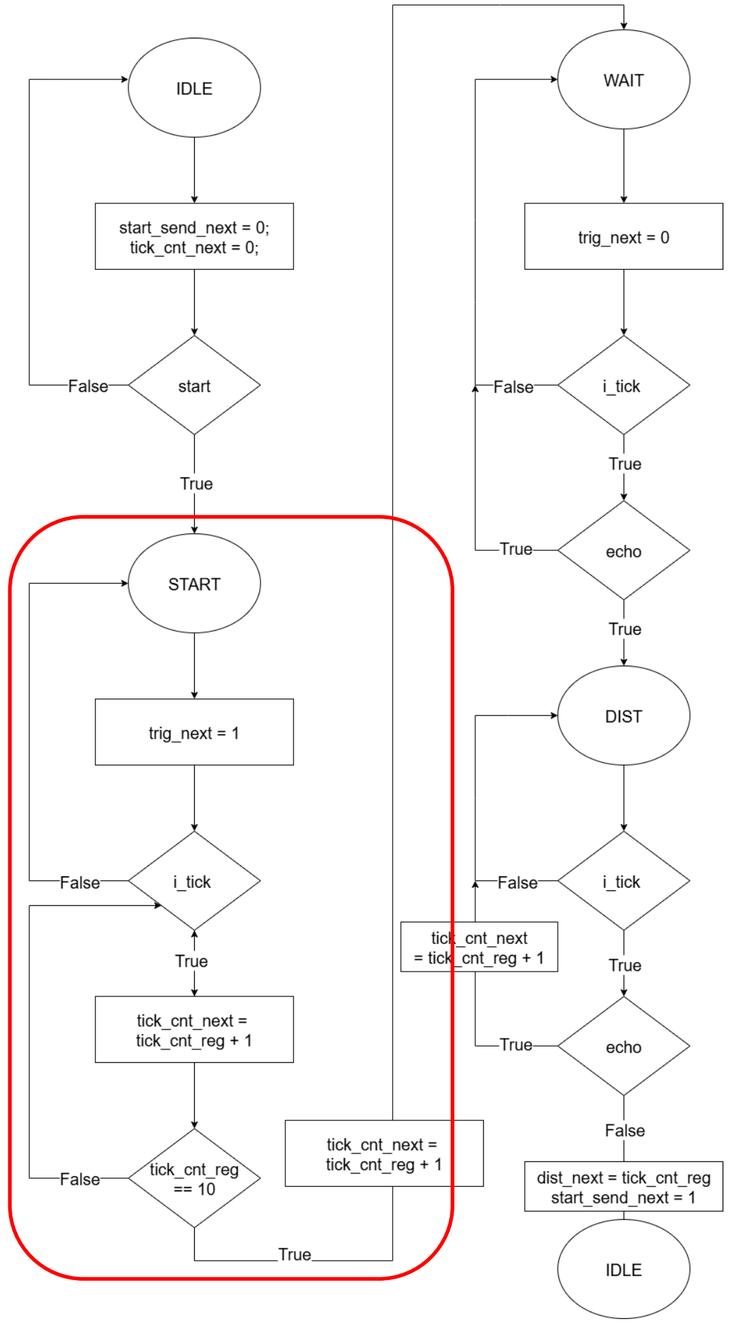


SR04 ASM & Code

```
always @(*) begin
    case (state)
        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;
                end
            end
        end
        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
    endcase
end
```

Trig_next = 1

-> SR_04로 Trig전송



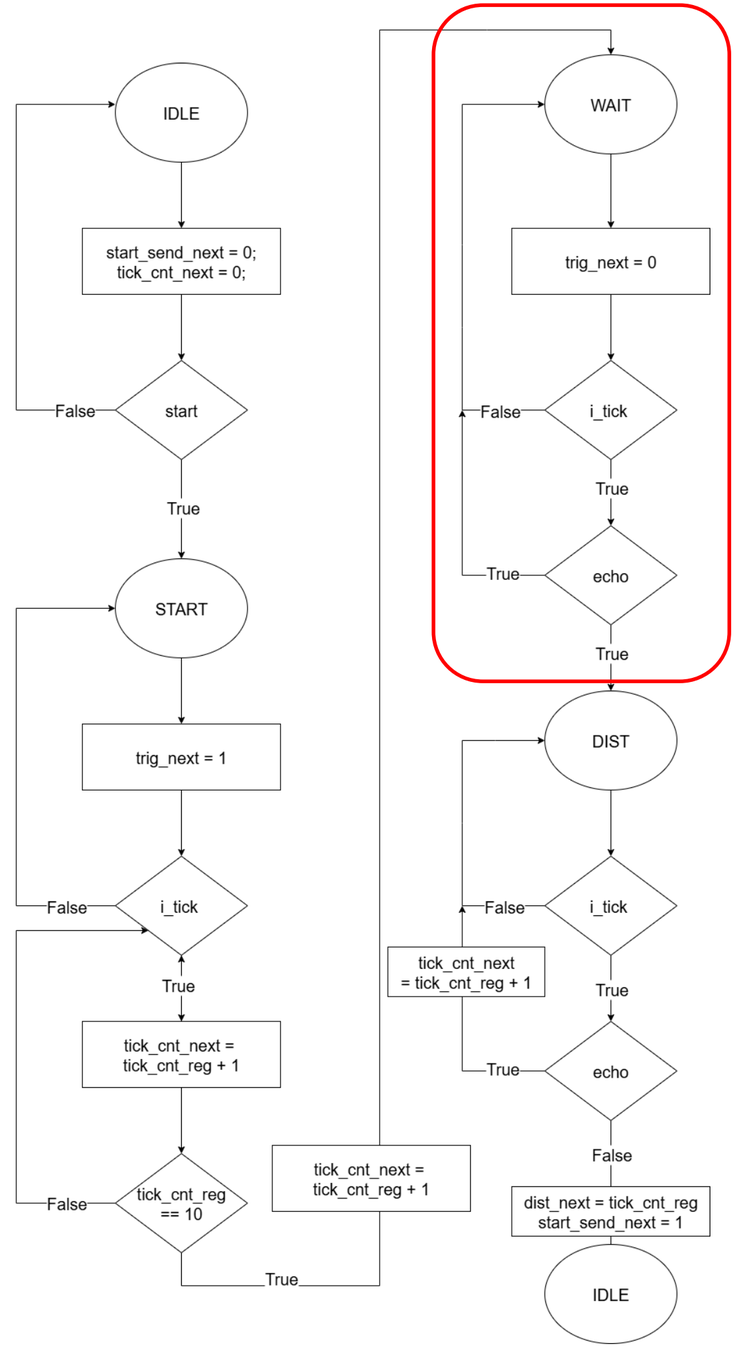
SR04 ASM & Code

```
always @(*) begin
    case (state)
        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;
                end
            end
        end
        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
    endcase
end
```

Trig_next = 0

-> SR04로 보내는 Trig 중지

echo돌아오면 DIST로

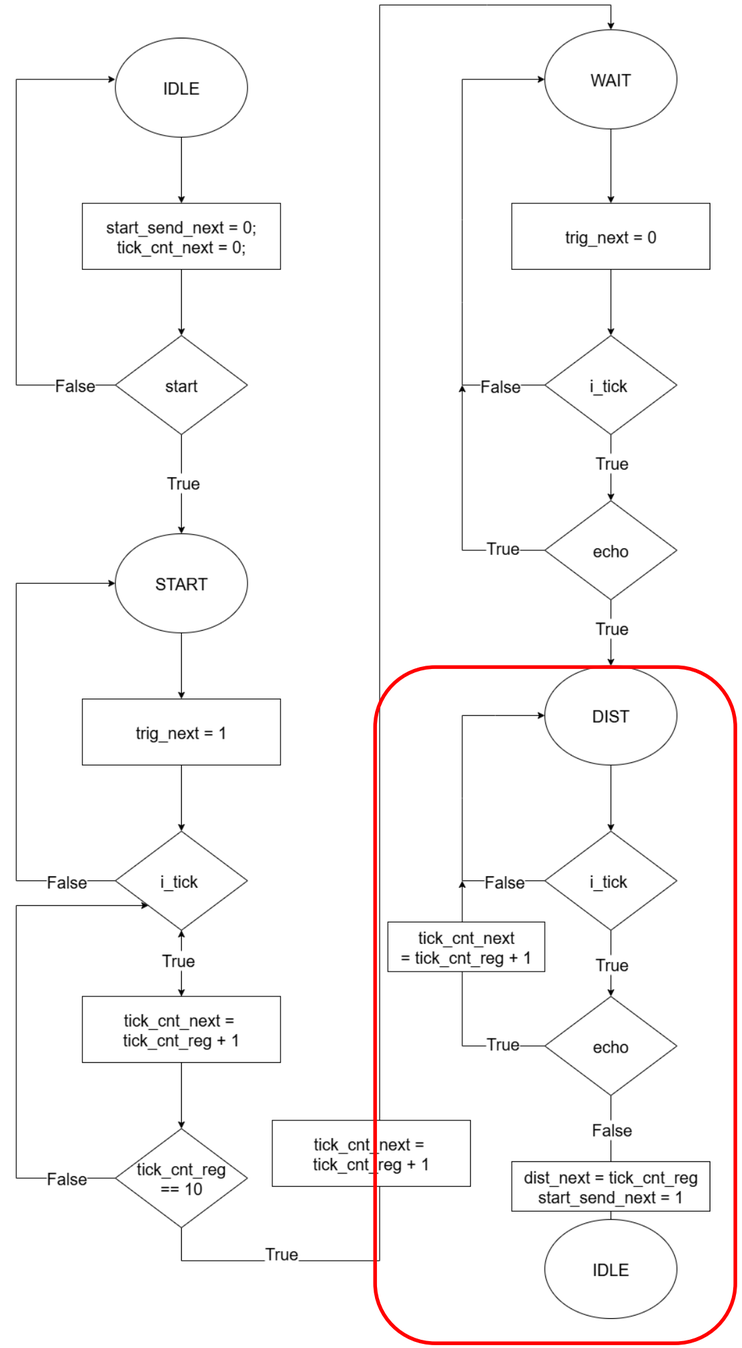


SR04 ASM & Code

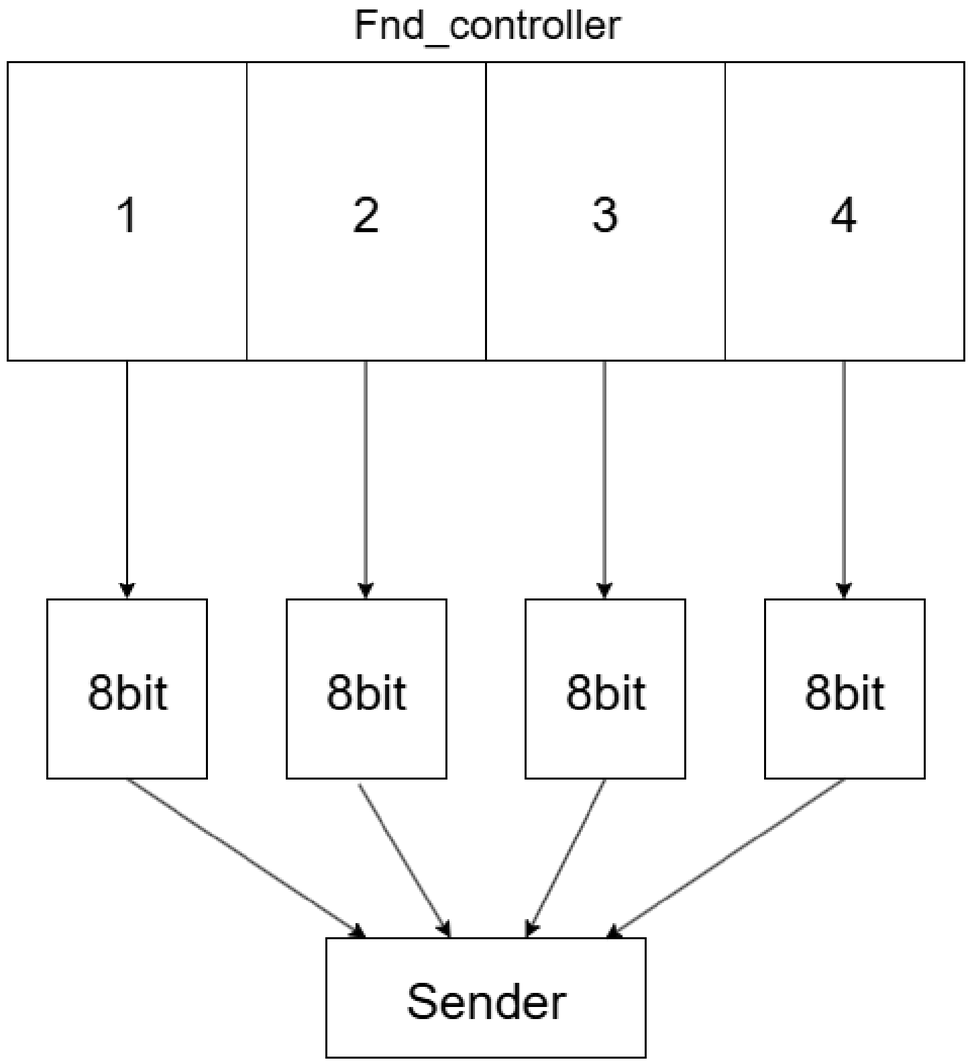
```
always @(*) begin
    case (state)
        IDLE: begin
            start_send_next = 0;
            tick_cnt_next   = 0;
            if (start) begin
                next = START;
            end
        end
        START: begin
            trig_next = 1'b1;
            if (i_tick) begin
                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;
                end
            end
        end
        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
    endcase
end
```

Start_send_next = 1

-> Sender활성화



SR04 Sender



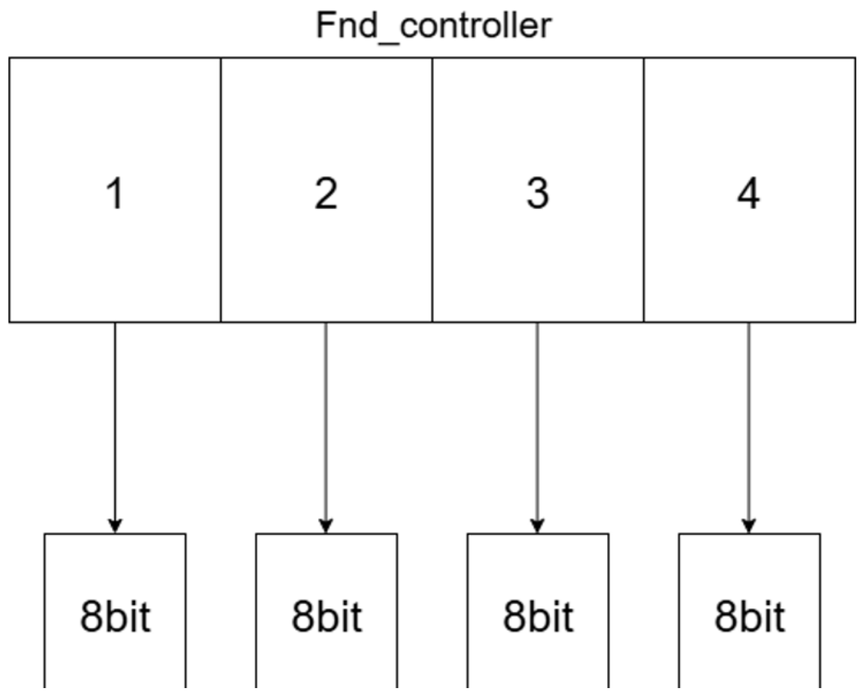
Segment 에 출력된 값을 Comport master출력

각 자릿수의 해당하는 수를 ASCII로

총 32비트 전송

SR04 Sender

```
module datatoascii (
    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;

endmodule
```



O_dist에서 보낸 데이터를 4개의 ASCII코드로 분리

SR04 Sender

```
always @(*) begin
    next           = state;
    send_cnt_next  = send_cnt_reg;
    send_data_next = send_data_reg;
    tx_done_next   = tx_done_reg;
    push_next      = push_reg;
    case (state)
        IDLE: begin
            tx_done_next = 0;
            send_cnt_next = 0;
            push_next = 0;
            if (start_send) begin
                next = 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 = IDLE;
                        tx_done_next = 1'b1;
                    end
                end
            end else next = state;
        end
    endcase
end
```

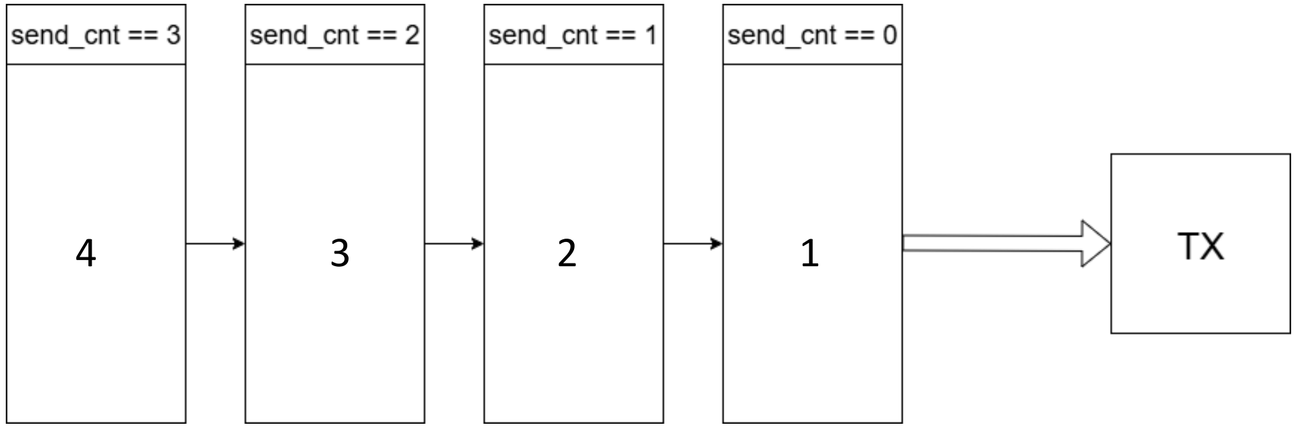
tx_done_next -> 전송완료 신호

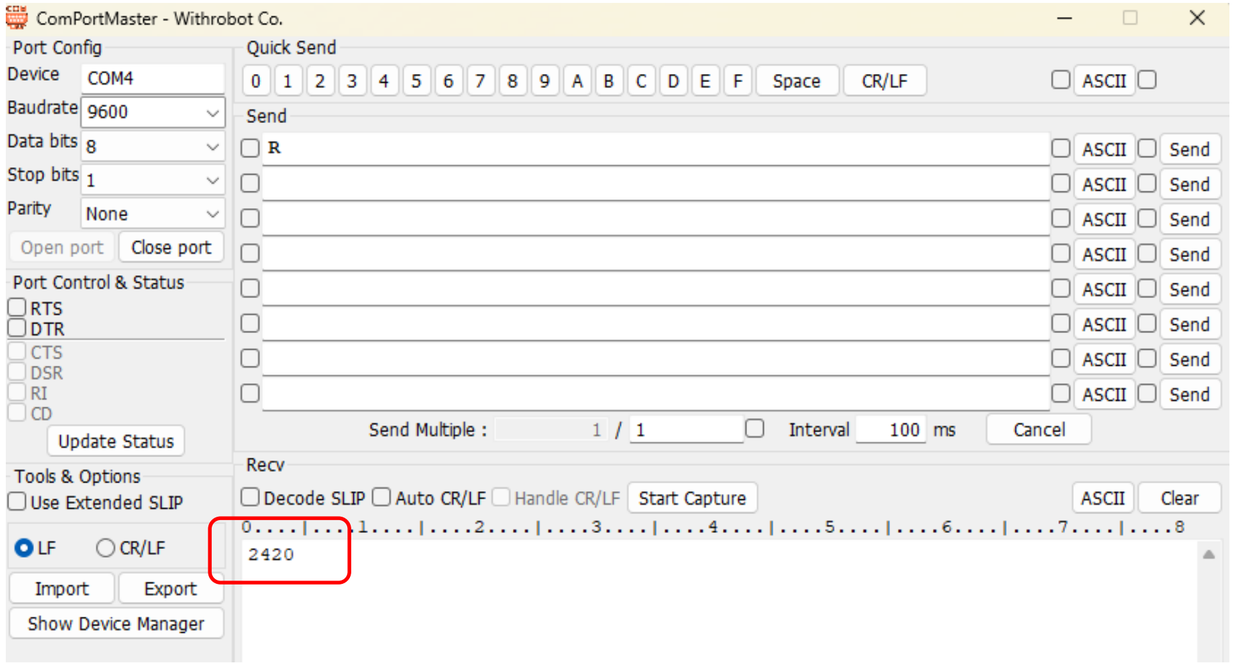
Push_next -> 푸쉬 신호

Start_send -> 전송 시작 트리거
(ASM의 start_send에서 받음)

SR04 Sender

```
always @(*) begin
    next           = state;
    send_cnt_next  = send_cnt_reg;
    send_data_next = send_data_reg;
    tx_done_next   = tx_done_reg;
    push_next      = push_reg;
    case (state)
        IDLE: begin
            tx_done_next = 0;
            send_cnt_next = 0;
            push_next = 0;
            if (start_send) begin
                next = 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 = IDLE;
                        tx_done_next = 1'b1;
                    end
                end
            end else next = state;
        end
    endcase
end
```

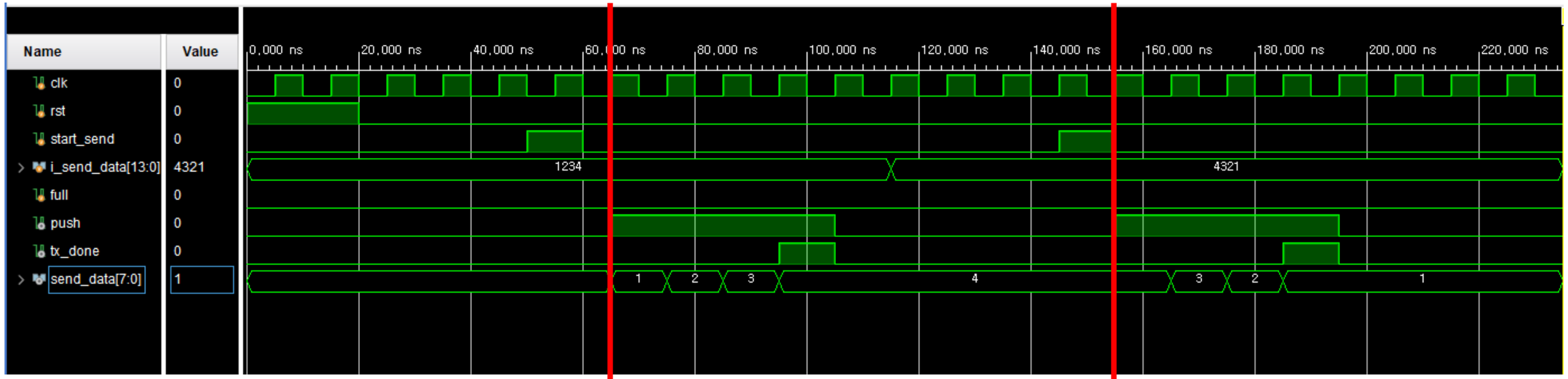




Sender Testbench

Demical

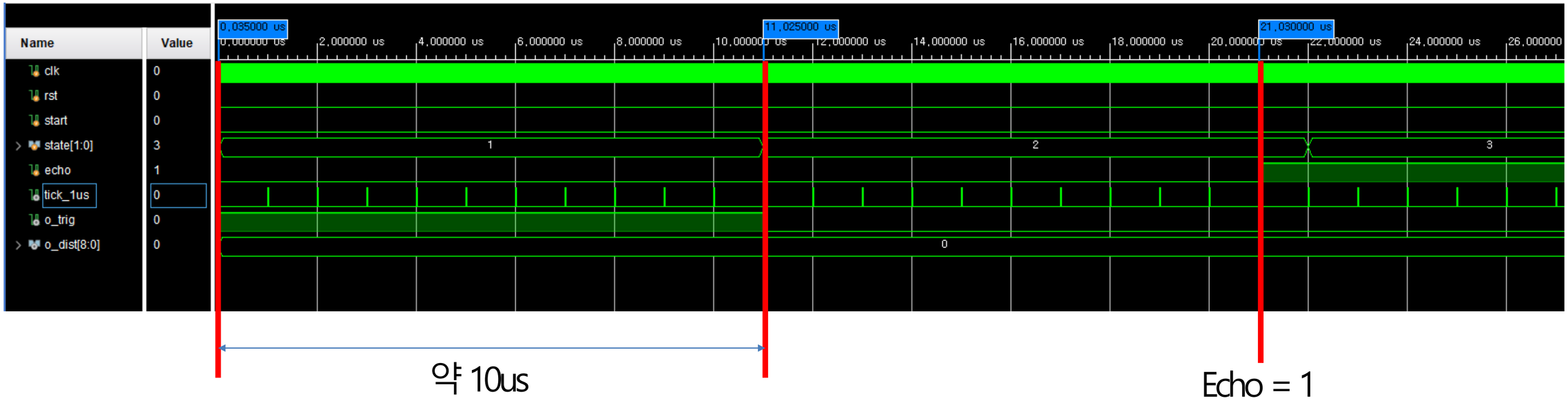
ASCII



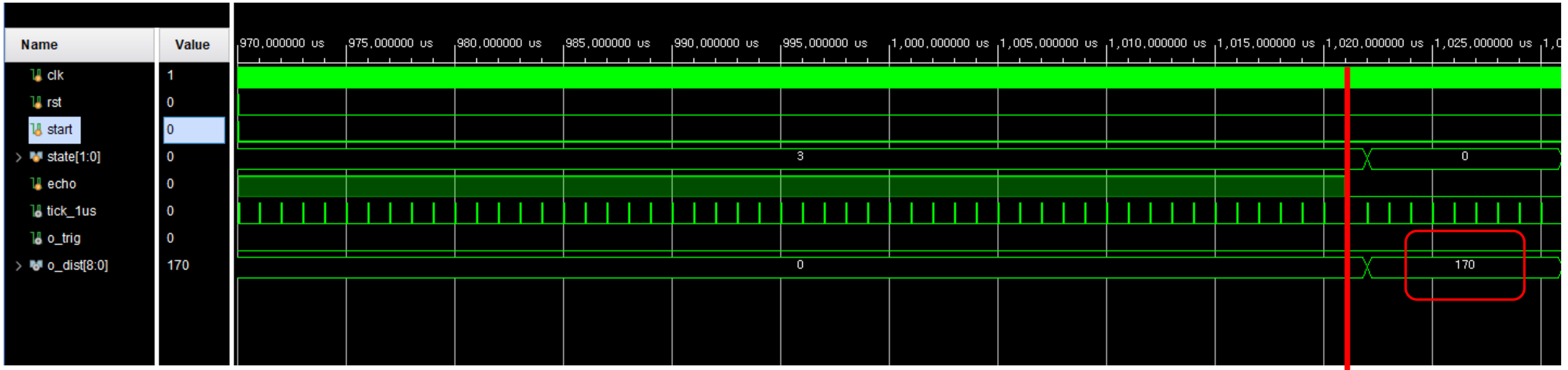
Data_push

Data_push

Top Testbench



Top Testbench

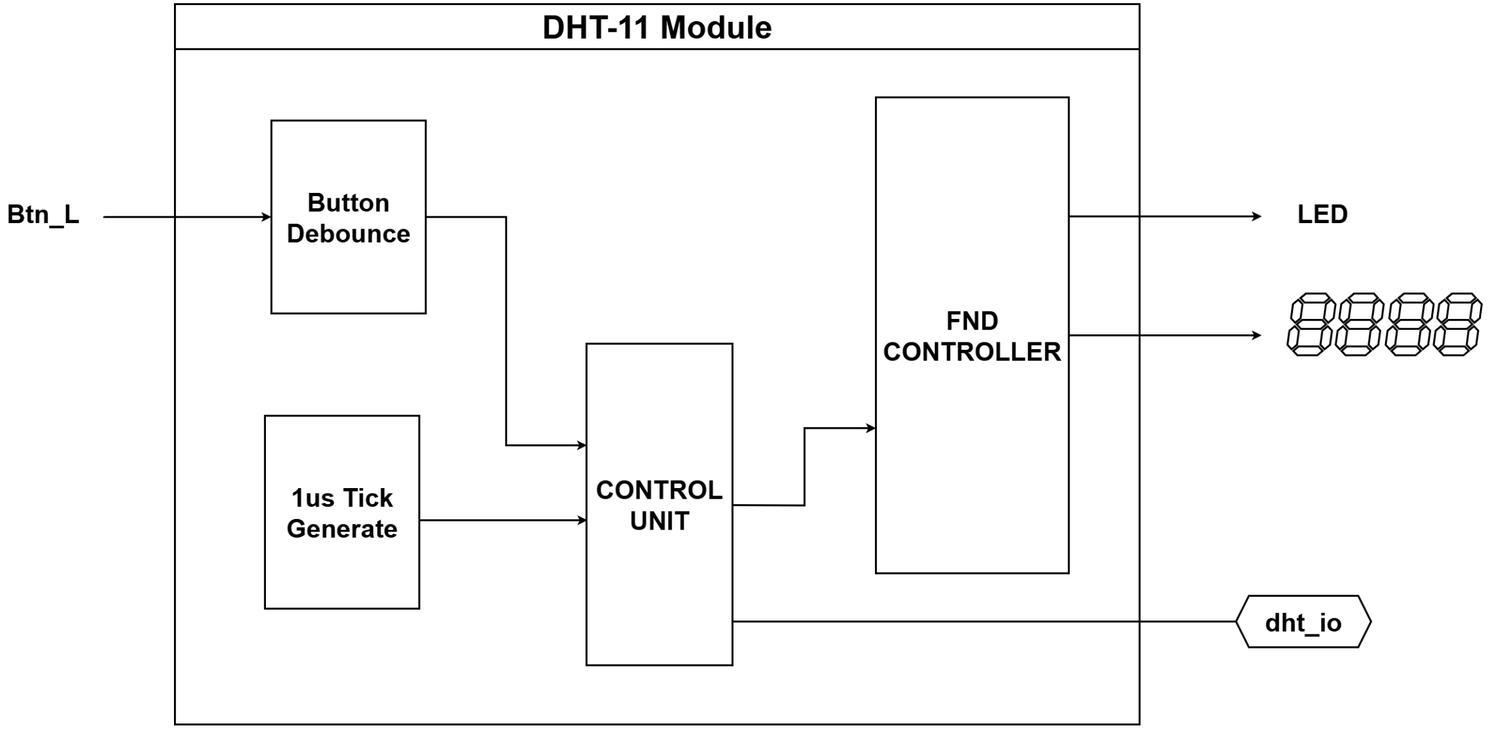


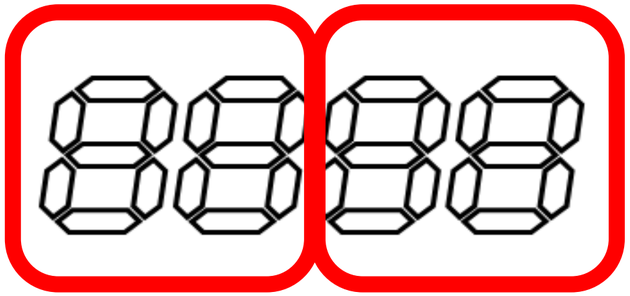
Echo = 0

DHT-11

DHT-11 + FND Controller

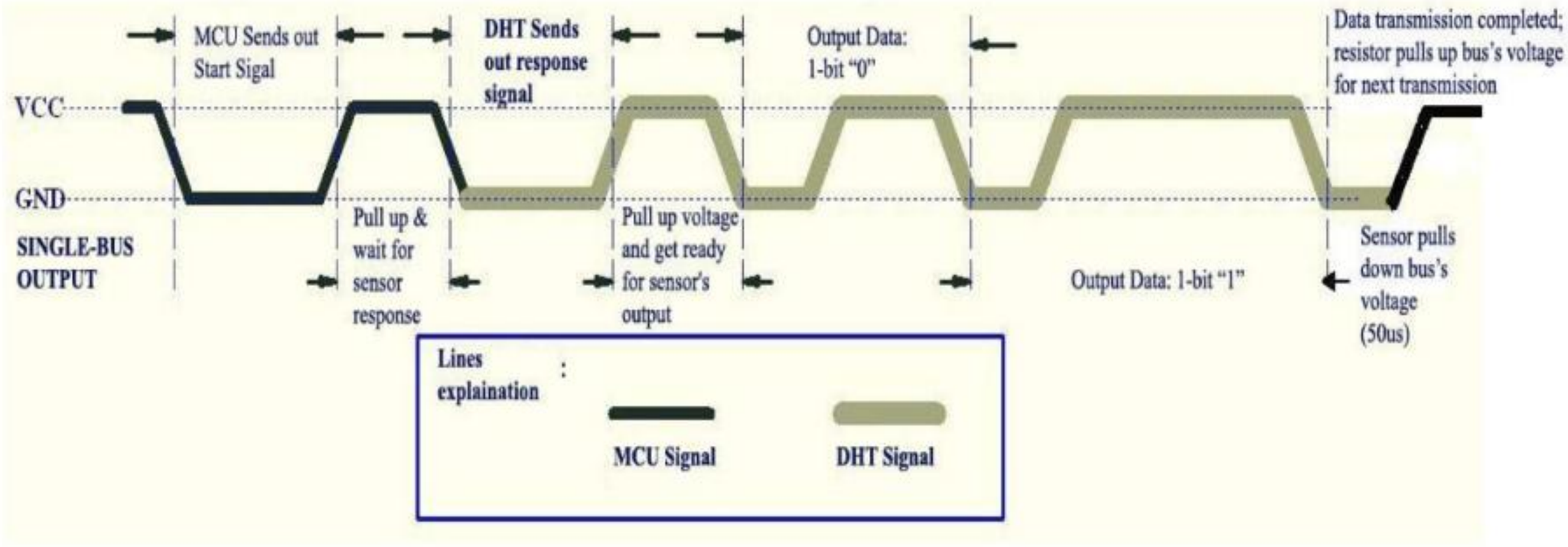
DHT-11 BLOCK DIAGRAM



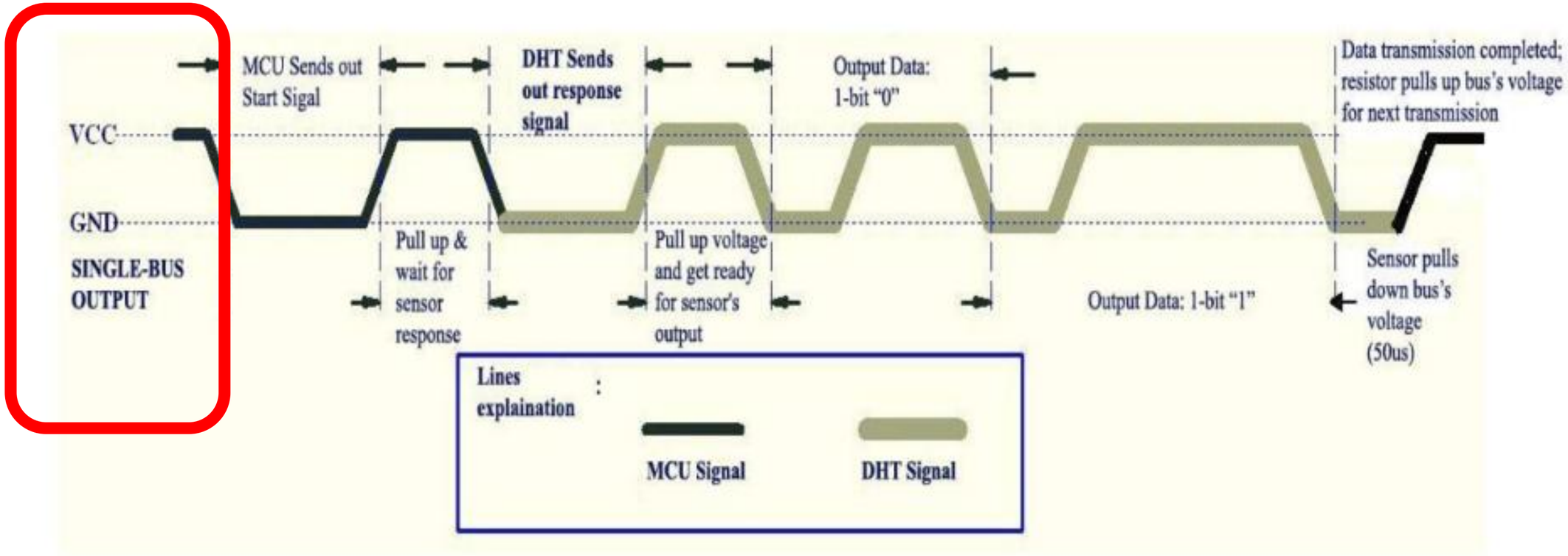


습도 온도

DHT-11 Communication Process

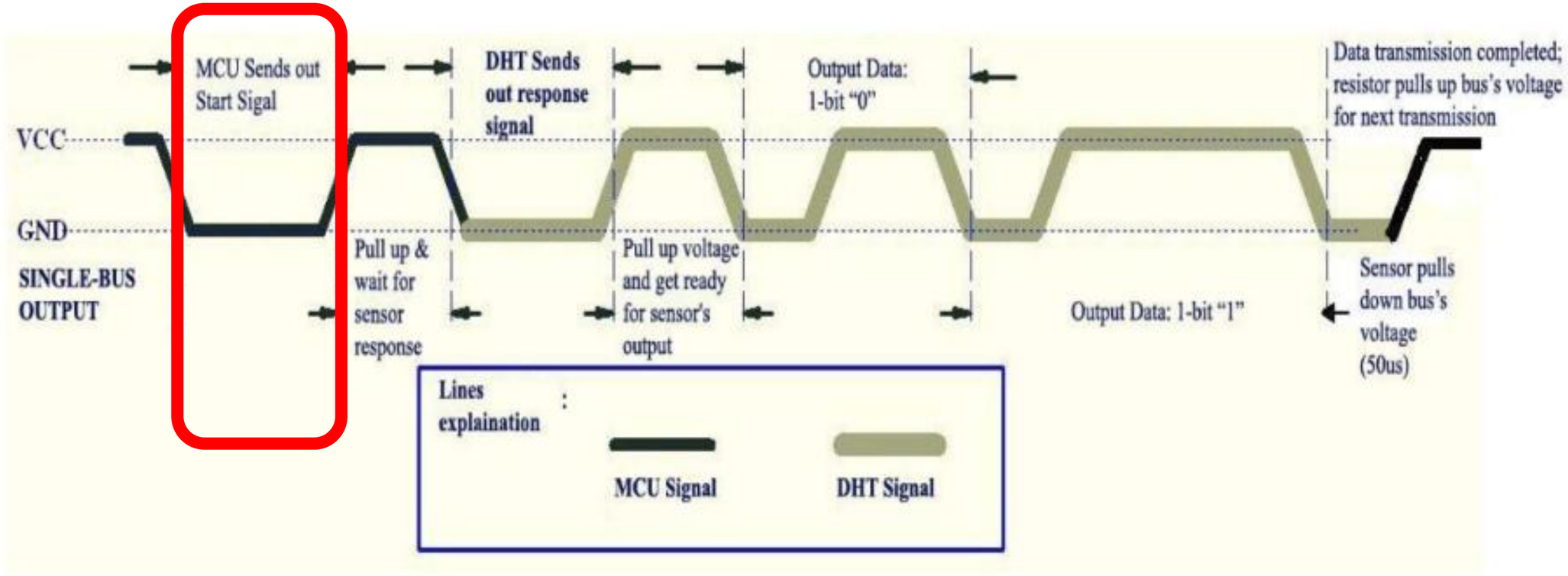


DHT-11 IDLE



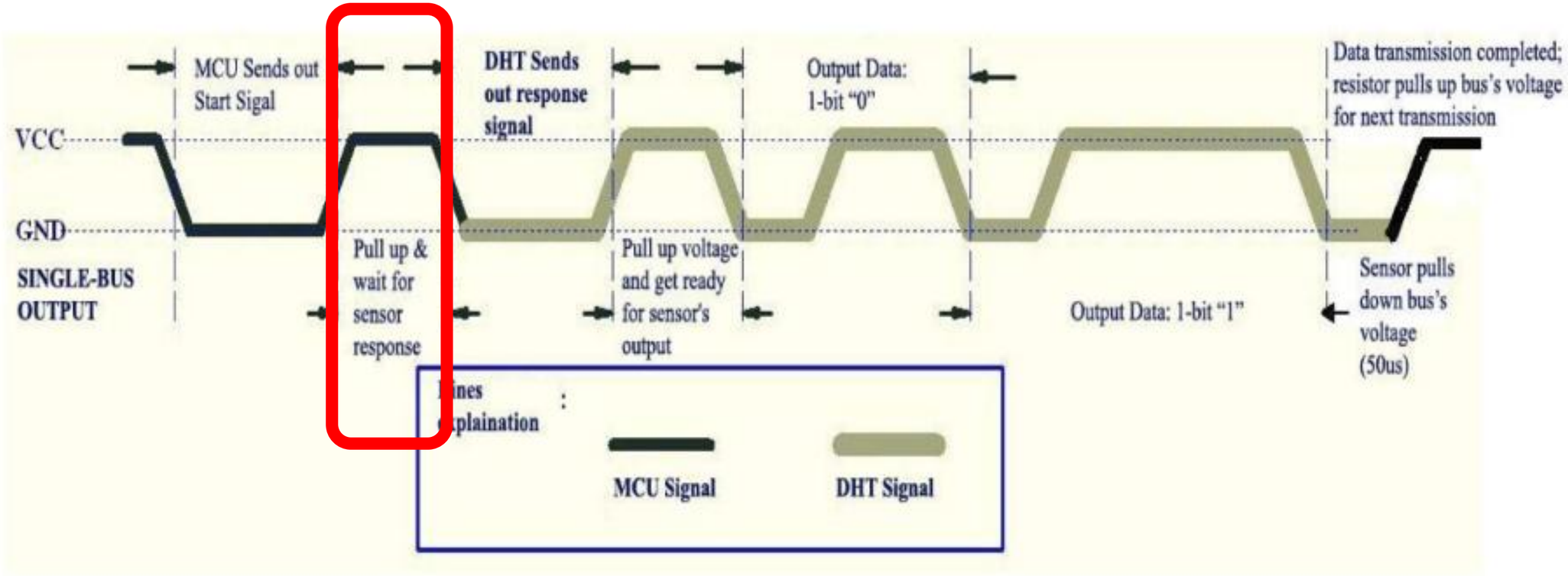
- 보드와 센서 간의 통신이 없는 대기 상태
- 보드는 dht_io를 HIGH 상태로 유지하며, 사용자의 시작 신호(버튼 입력)를 기다림

DHT-11 START



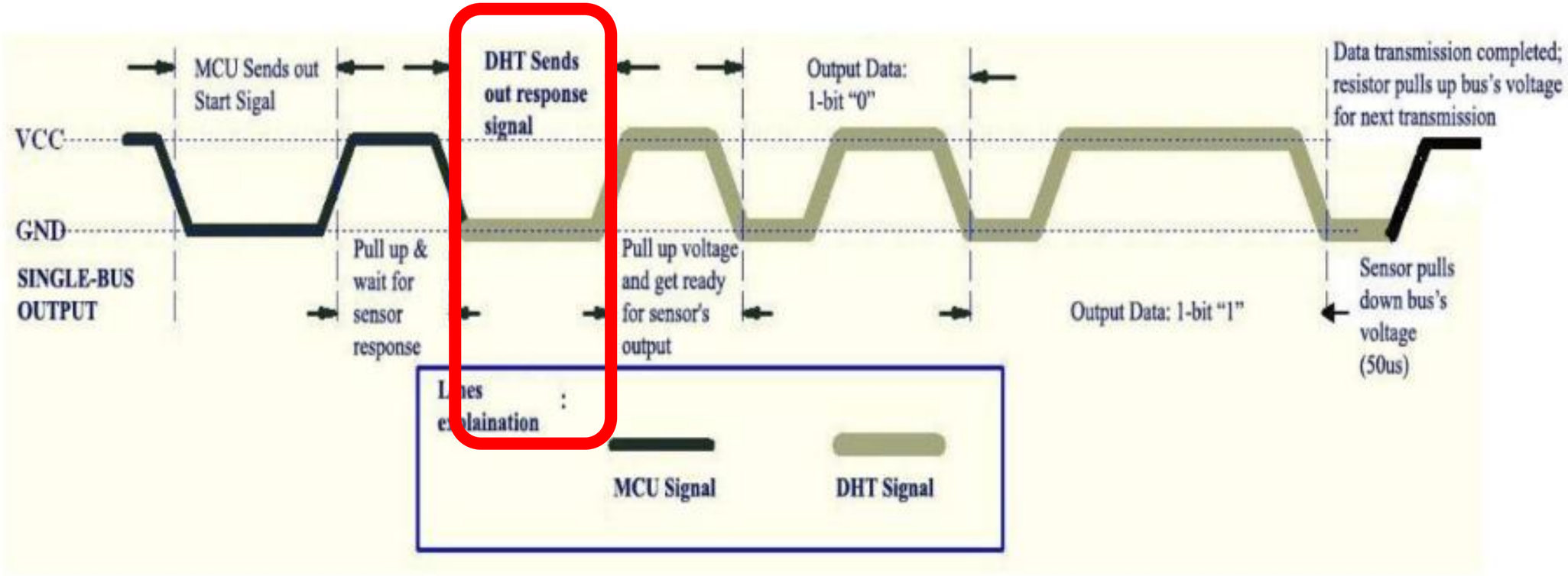
- 보드가 DHT-11 센서에 통신 시작을 알리는 상태
- 보드가 dht_io를 약 18ms 동안 LOW로 만들어 센서 깨우기 (설계 시 20ms)

DHT-11 WAIT



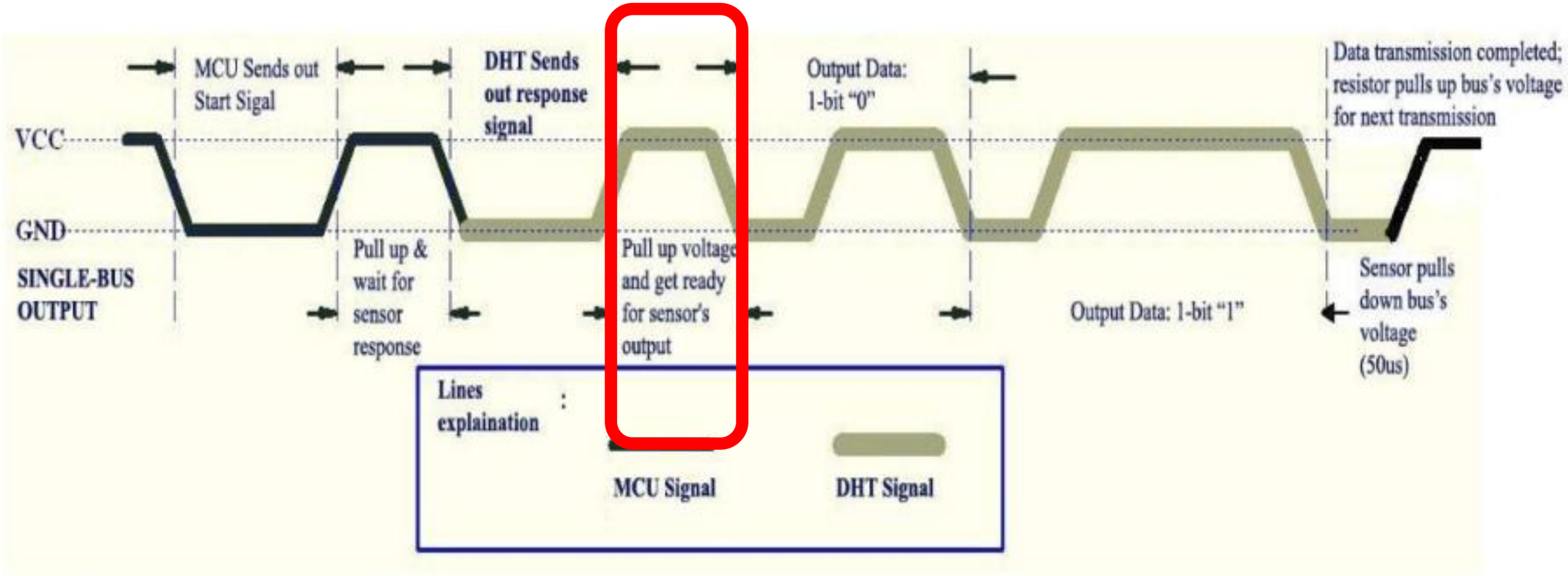
- 보드가 dht_io를 HIGH로 전환한 후, 센서의 응답을 기다리는 상태
- 이때 보드는 dht_io 의 제어권을 센서가 가져갈 수 있도록 dht_io를 z로 만들기

DHT-11 SYNC LOW



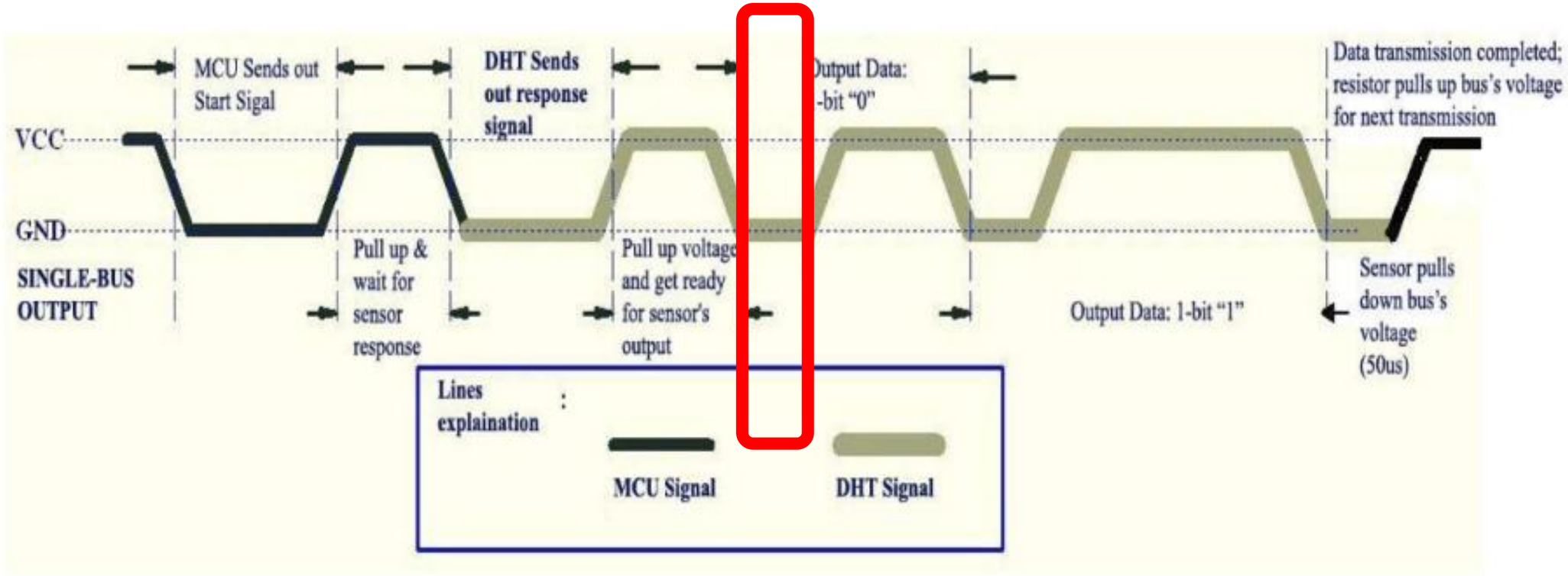
- 센서가 응답의 시작으로 dht_io 을 약 80μs 동안 LOW로 만드는 구간을 감지하는 상태

DHT-11 SYNC HIGH



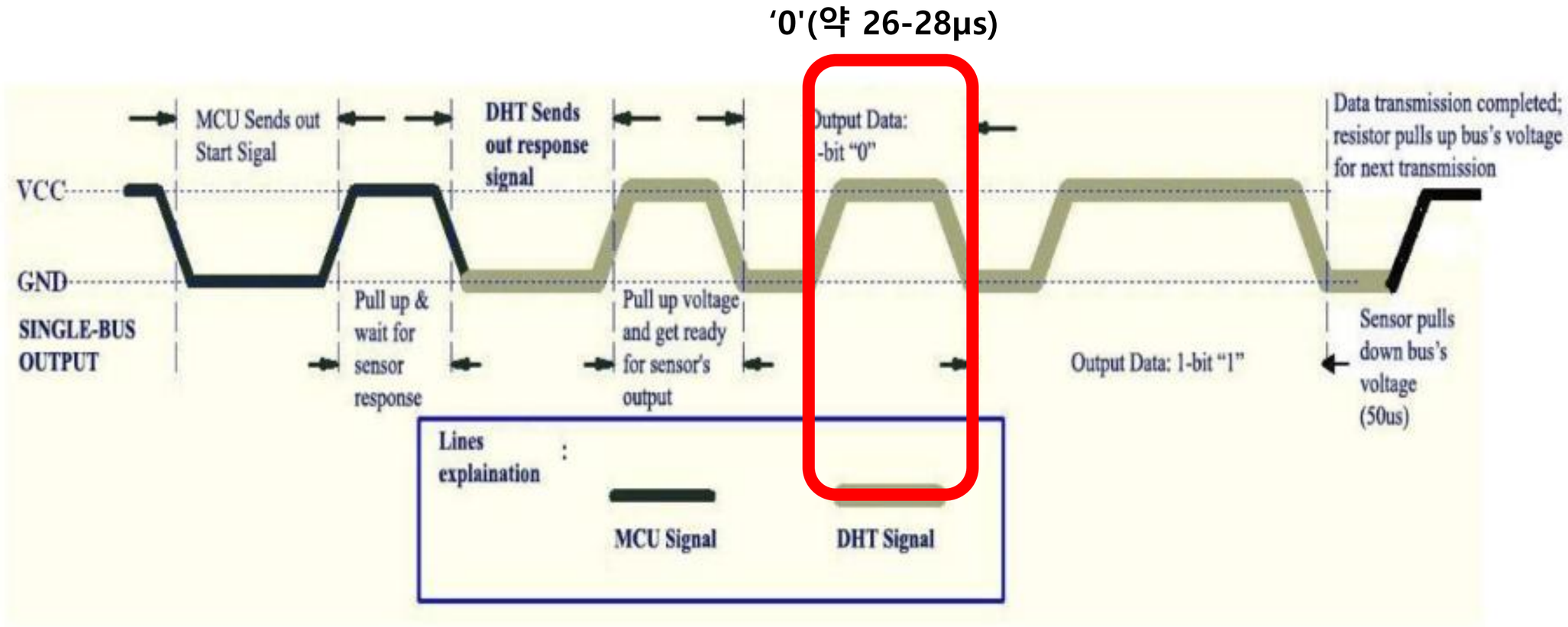
- 센서가 데이터 전송 시작을 알리기 위해 dht_io 를 약 80μs 동안 HIGH로 만드는 구간을 감지하는 상태

DHT-11 DATA SYNC



- 본격적인 데이터 수신에 앞서, 각 데이터 비트의 시작을 알리는 50μs의 LOW 신호를 확인하는 동기화 단계

DHT-11 DATA DETECT

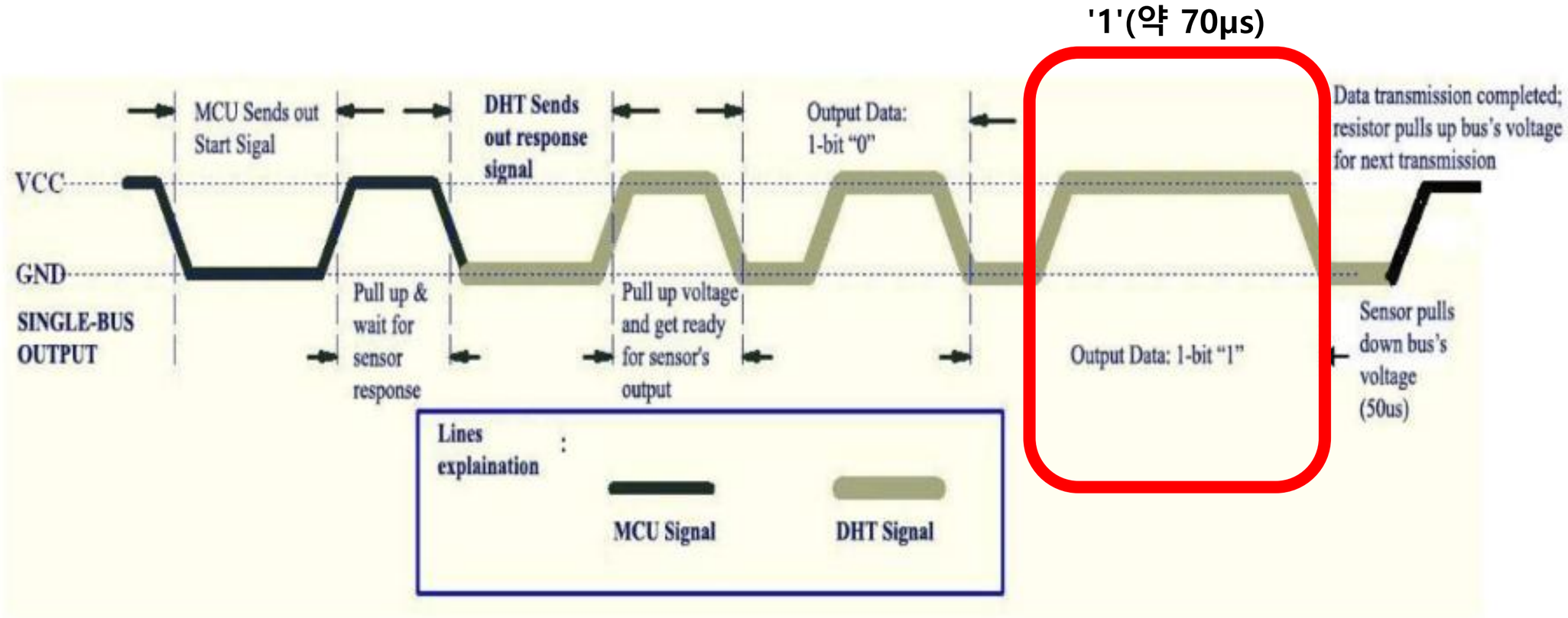


- dht_io가 HIGH로 유지되는 시간의 길이를 측정하여

  '0'(약 26-28μs) 또는 '1'(약 70μs)을 판별하고, 40비트의 데이터를 순차적으로 수신

- 설계 시 50μs 기준으로 데이터 판별

DHT-11 DATA DETECT

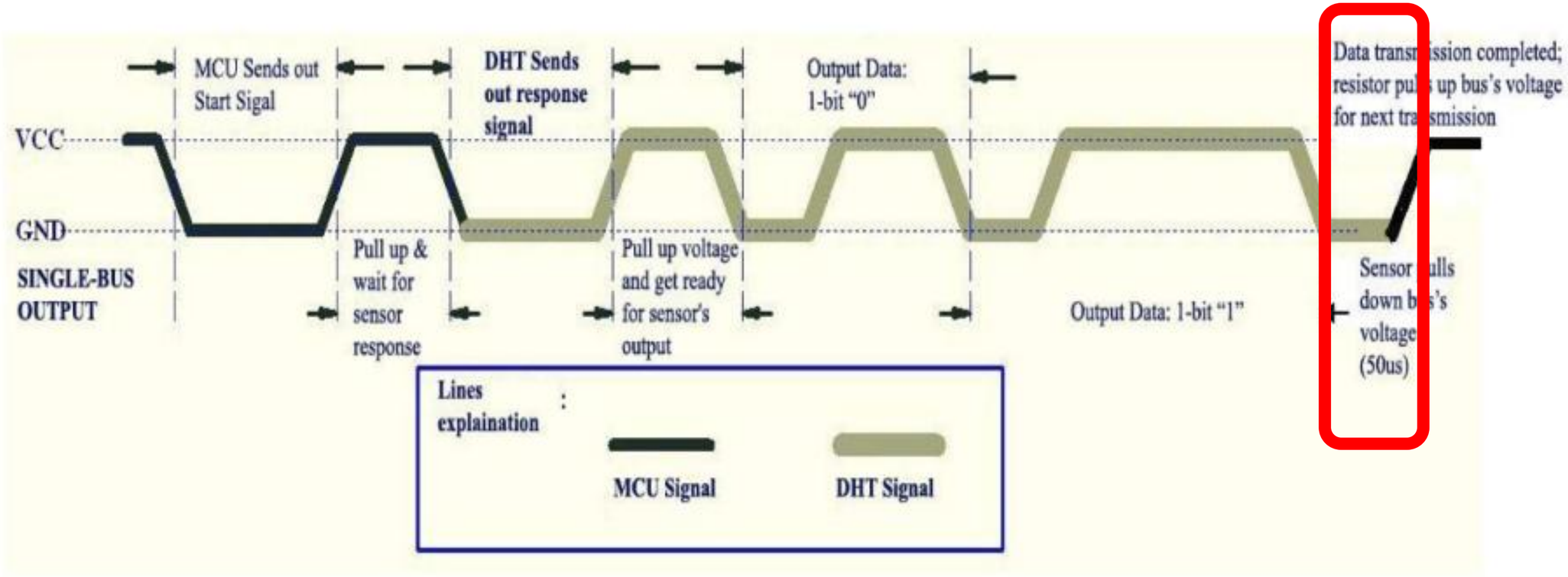


- dht_io가 HIGH로 유지되는 시간의 길이를 측정하여

  '0'(약 26-28μs) 또는 '1'(약 70μs)을 판별하고, 40비트의 데이터를 순차적으로 수신

- 설계 시 50μs 기준으로 데이터 판별

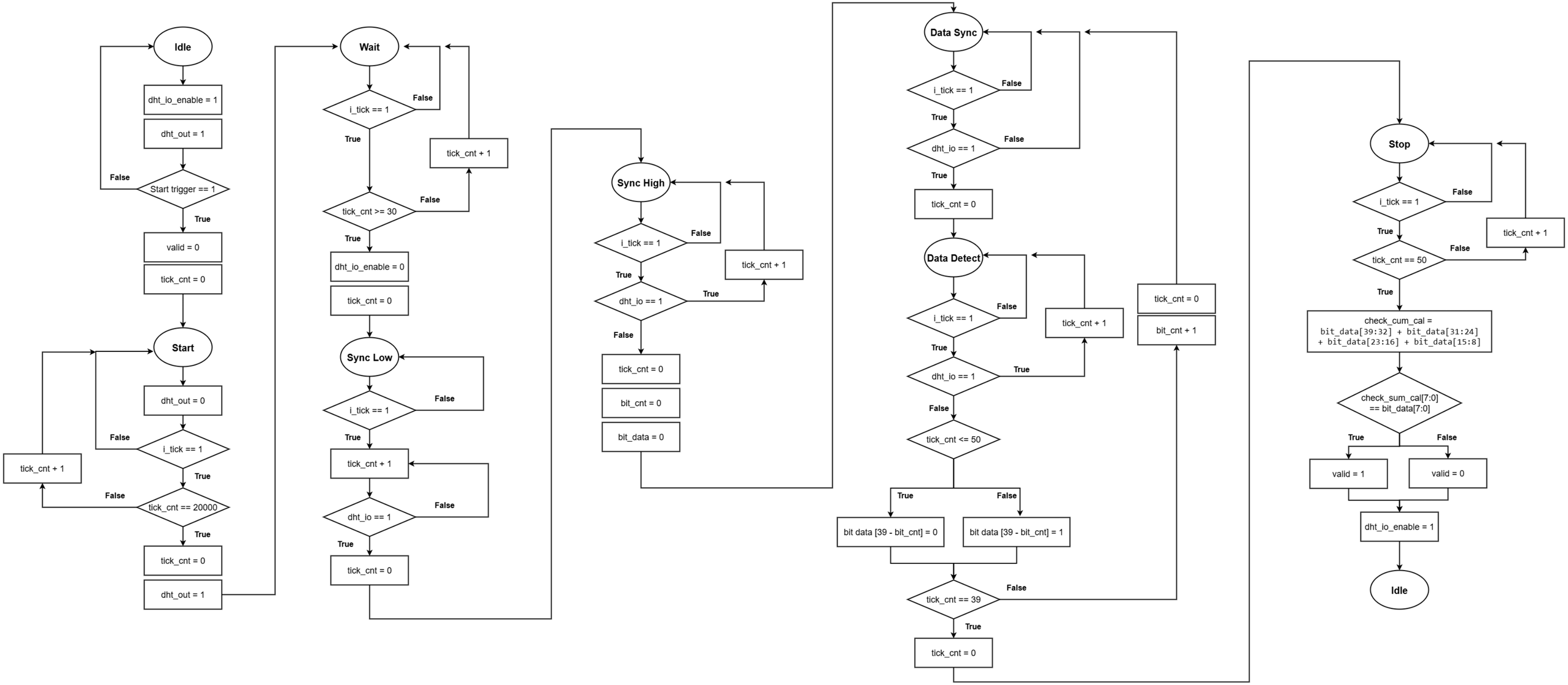
DHT-11 STOP



- 40비트(습도 16비트, 온도 16비트, Checksum 8비트) 데이터 수신을 모두 완료한 상태
- 수신된 데이터의 Checksum을 계산하여 데이터 유효성을 검증하고,

  통신을 종료한 뒤 다시 IDLE 상태로 돌아감

DHT-11 ASM


Idle
dht_io_enable = 1
dht_out = 1
Start trigger == 1
False
True
valid = 0
tick_cnt = 0
Start
dht_out = 0
i_tick == 1
False
True
tick_cnt == 20000
False
True
tick_cnt = 0
dht_out = 1
tick_cnt + 1
Wait
i_tick == 1
False
True
tick_cnt >= 30
False
True
tick_cnt + 1
dht_io_enable = 0
tick_cnt = 0
Sync Low
i_tick == 1
False
True
tick_cnt + 1
dht_io == 1
False
True
tick_cnt = 0
Sync High
i_tick == 1
False
True
tick_cnt + 1
dht_io == 1
True
False
tick_cnt = 0
bit_cnt = 0
bit_data = 0
Data Sync
i_tick == 1
False
True
dht_io == 1
False
True
tick_cnt = 0
Data Detect
i_tick == 1
False
True
tick_cnt + 1
dht_io == 1
True
False
tick_cnt <= 50
True
False
bit data [39 - bit_cnt] = 0
bit data [39 - bit_cnt] = 1
tick_cnt == 39
False
True
tick_cnt = 0
tick_cnt = 0
bit_cnt + 1
Stop
i_tick == 1
False
True
tick_cnt == 50
False
True
tick_cnt + 1
check_cum_cal =
bit_data[39:32] + bit_data[31:24]
+ bit_data[23:16] + bit_data[15:8]
check_sum_cal[7:0]
== bit_data[7:0]
True
False
valid = 1
valid = 0
dht_io_enable = 1
Idle

DHT-11 ASM & Code

```
case (c_state)
    IDLE: begin
        dht_io_enable_next = 1'b1;
        dht_out_next = 1'b1;
        if (i_start) begin
            valid_next = 0;
            tick_cnt_next = 0;
            n_state = START;
        end
    end
    START: begin
        dht_out_next = 1'b0;
        if (i_tick) begin
            if (tick_cnt_reg == 20000) begin
                tick_cnt_next = 0;
                dht_out_next = 1'b1;
                n_state = WAIT;
            end else begin
                tick_cnt_next = tick_cnt_reg + 1;
            end
        end
    end
```

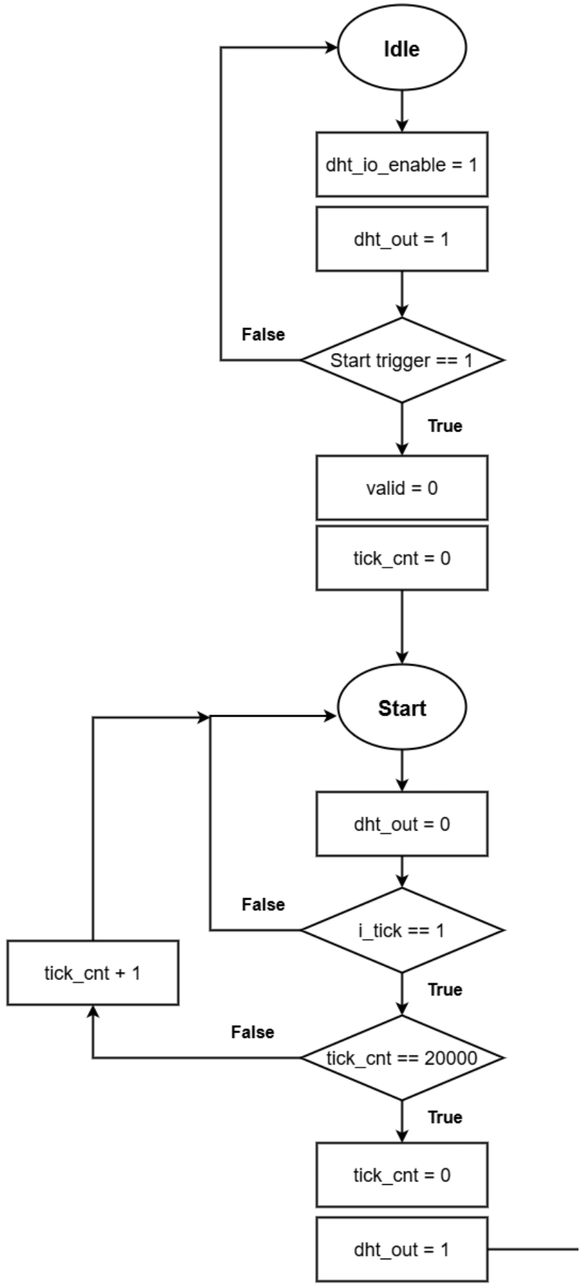
IDLE

i_start 들어오면
Valid 초기화 /
tick_count 초기화

1us tick generate 모듈 존재
-> 타이밍 제어

START

tick_count = 20000되면
dht_out : high



DHT-11 ASM & Code

```
assign dht_io = (dht_io_enable_reg) ? dht_out_reg : 1'bz;
```

```
WAIT: begin
    if (i_tick) begin
        if (tick_cnt_reg >= 30) begin
            dht_io_enable_next = 1'b0;
            tick_cnt_next = 0;
            n_state = SYNCL;
        end else begin
            tick_cnt_next = tick_cnt_reg + 1;
        end
    end
end
SYNCL: begin
    if (i_tick) begin
        tick_cnt_next = tick_cnt_reg + 1;
        if (dht_io) begin
            tick_cnt_next = 0;
            n_state = SYNCH;
        end
    end
end
SYNCH: begin
    if (i_tick) begin
        if (dht_io) begin
            tick_cnt_next = tick_cnt_reg + 1;
        end else begin
            n_state = DATASYNC;
            tick_cnt_next = 0;
            bit_cnt_next = 0;
            bit_data_next = 0;
        end
    end
end
```

WAIT

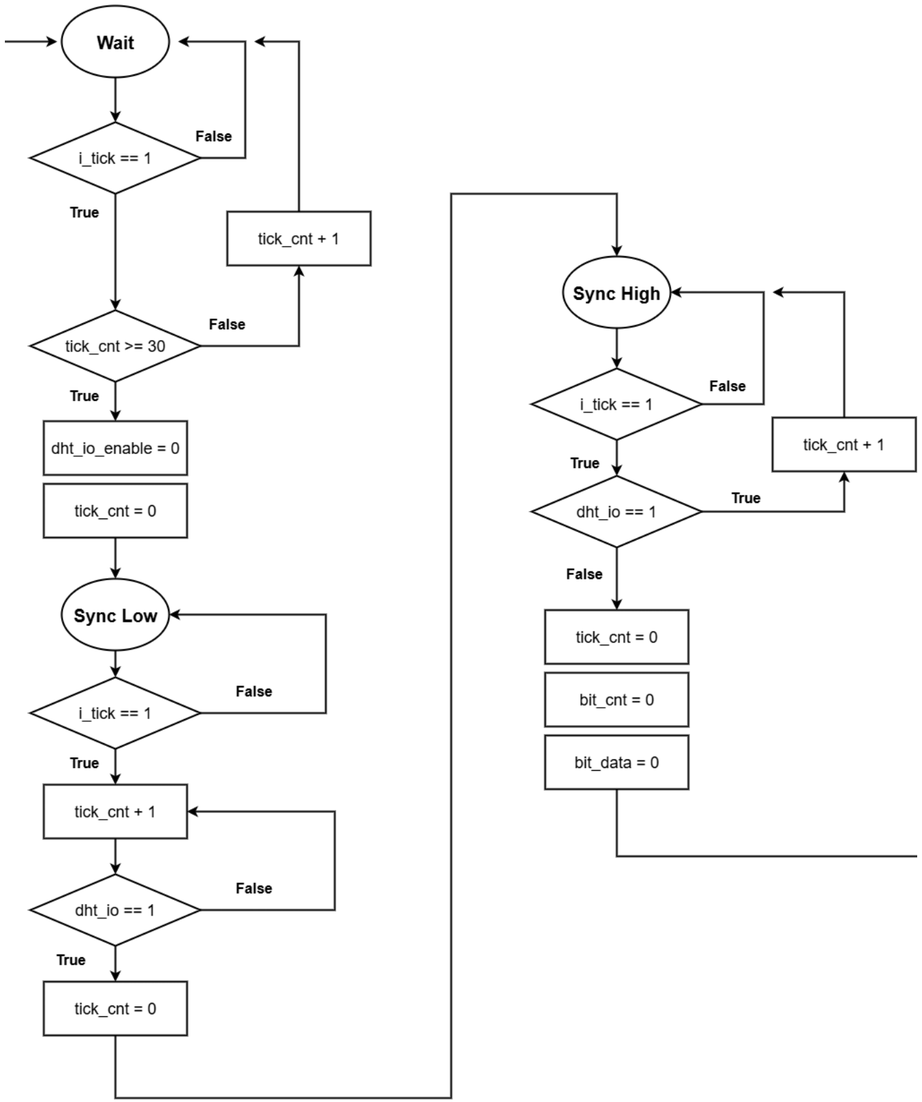
Tick count 30 이상일 경우
dht_io 제어권 전환
: 보드 -> 센서

SYNC LOW

dht_io : 1 이 되면
다음 state

SYNC HIGH

dht_io : 0 이 되면
다음 state



DHT-11 ASM & Code

```
DATASYNC: begin
    if (i_tick) begin
        if (dht_io) begin
            n_state = DATADETECT;
            tick_cnt_next = 0;
        end
    end
end
DATADETECT: begin
    if (i_tick) begin
        if (dht_io) begin
            tick_cnt_next = tick_cnt_reg + 1;
        end else begin
            if (tick_cnt_reg <= 50) begin
                bit_data_next[39-bit_cnt_reg] = 0;
            end else begin
                bit_data_next[39-bit_cnt_reg] = 1;
            end
            if (bit_cnt_reg == 39) begin
                tick_cnt_next = 0;
                n_state = STOP;
            end else begin
                tick_cnt_next = 0;
                bit_cnt_next = bit_cnt_reg + 1;
                n_state = DATASYNC;
            end
        end
    end
end
```

DATA SYNC

dht_io : 1 되면
다음 state

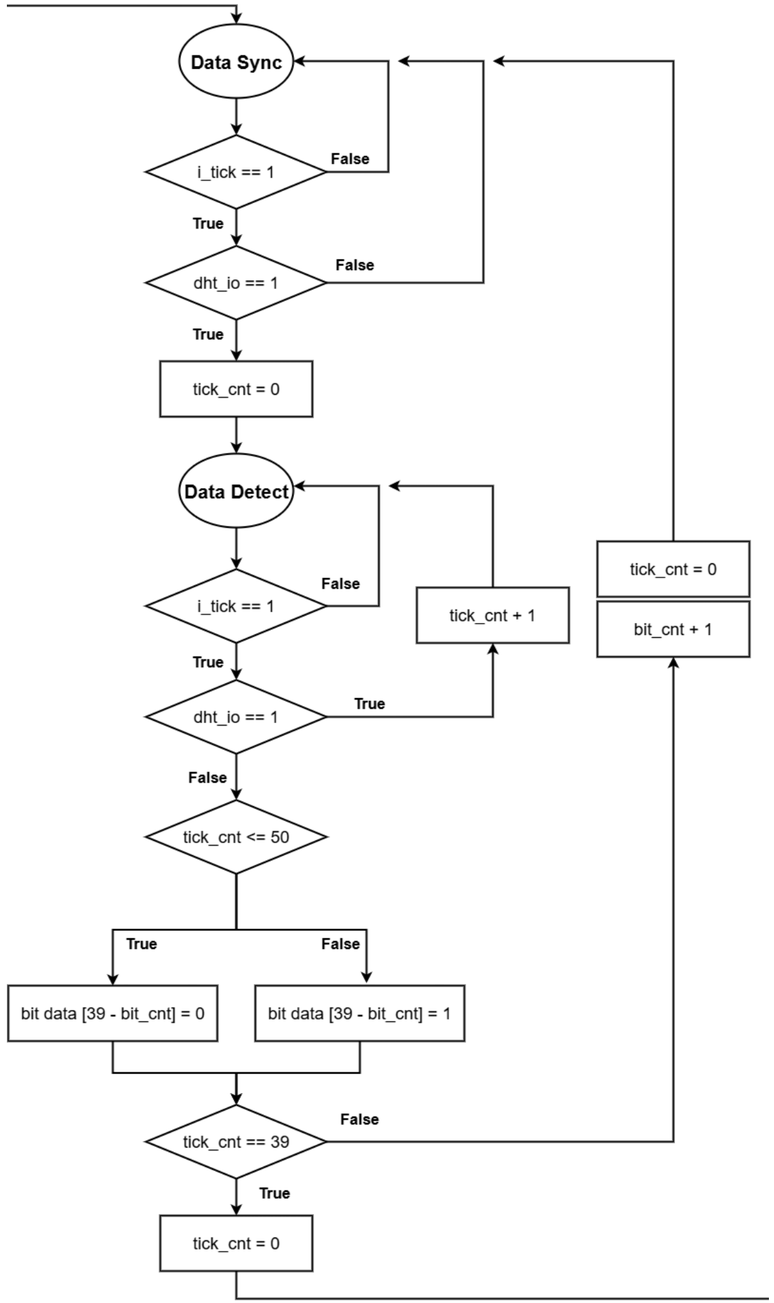
DATA DETECT

dht_io : 0 일 때

- tick count > 50 : bit_data = 1
- tick count <= 50 : bit_data = 0

- bit count == 39 : STOP state
- bit count < 39 : DATASYNC state

- 40비트 데이터 순차적으로 수신



DHT-11 ASM & Code

```
assign dht_io = (dht_io_enable_reg) ? dht_out_reg : 1'bz;

STOP: begin
    if (i_tick) begin
        if (tick_cnt_reg == 50) begin
            check_sum_cal = bit_data_reg[39:32]+bit_data_reg[31:24]+bit_data_reg[23:16]+bit_data_reg[15:8];
            if (check_sum_cal[7:0] == bit_data_reg[7:0]) begin
                valid_next = 1'b1;
            end else begin
                valid_next = 1'b0;
            end
            dht_io_enable_next = 1'b1;
            n_state = IDLE;
        end else begin
            tick_cnt_next = tick_cnt_reg + 1;
        end
    end
end
```

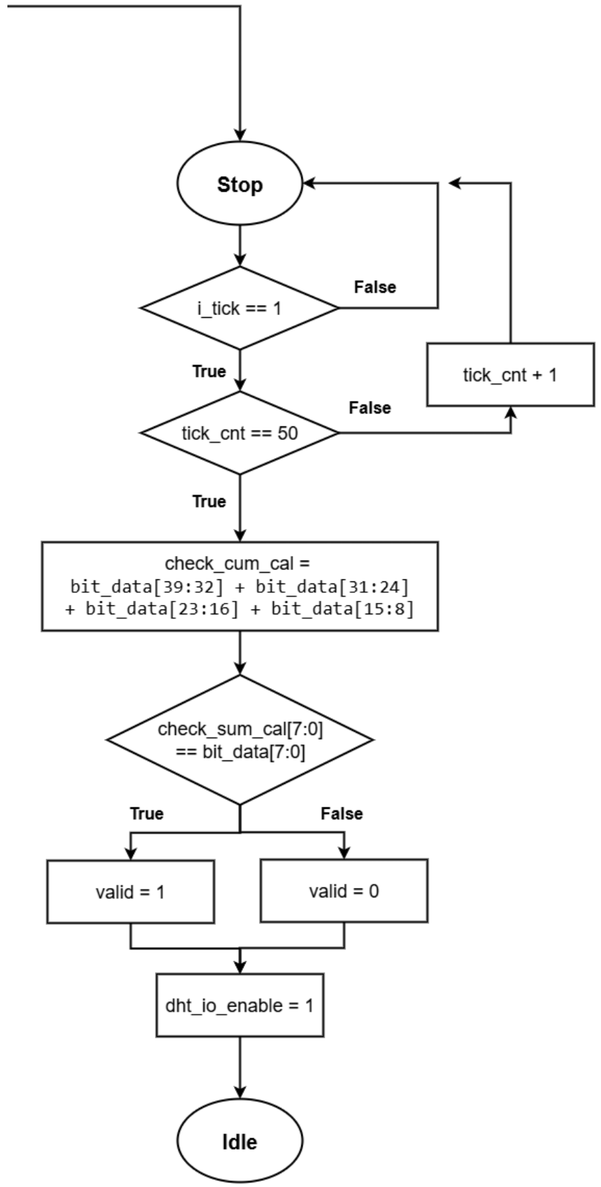
STOP

- tick count == 50

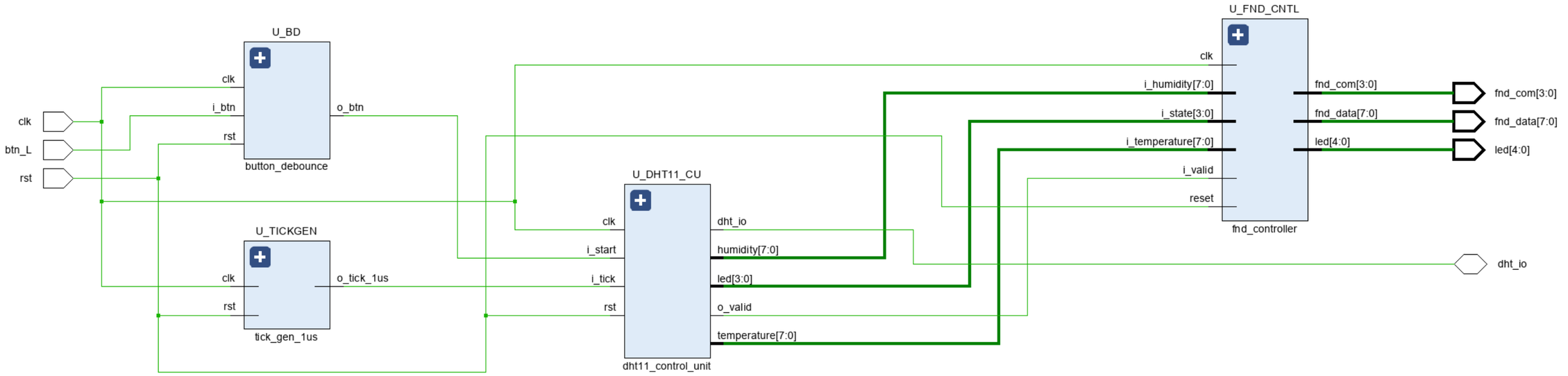
- Check_sum_cal : 습도 정수부 8bit + 습도 소수부 8bit + 온도 정수부 8bit + 온도 소수부 8bit

- 유효성 검사 : Check_sum_cal == bit_data_reg[7:0]

dht_io 제어권 전환: 센서 -> 보드



DHT-11 Schematic

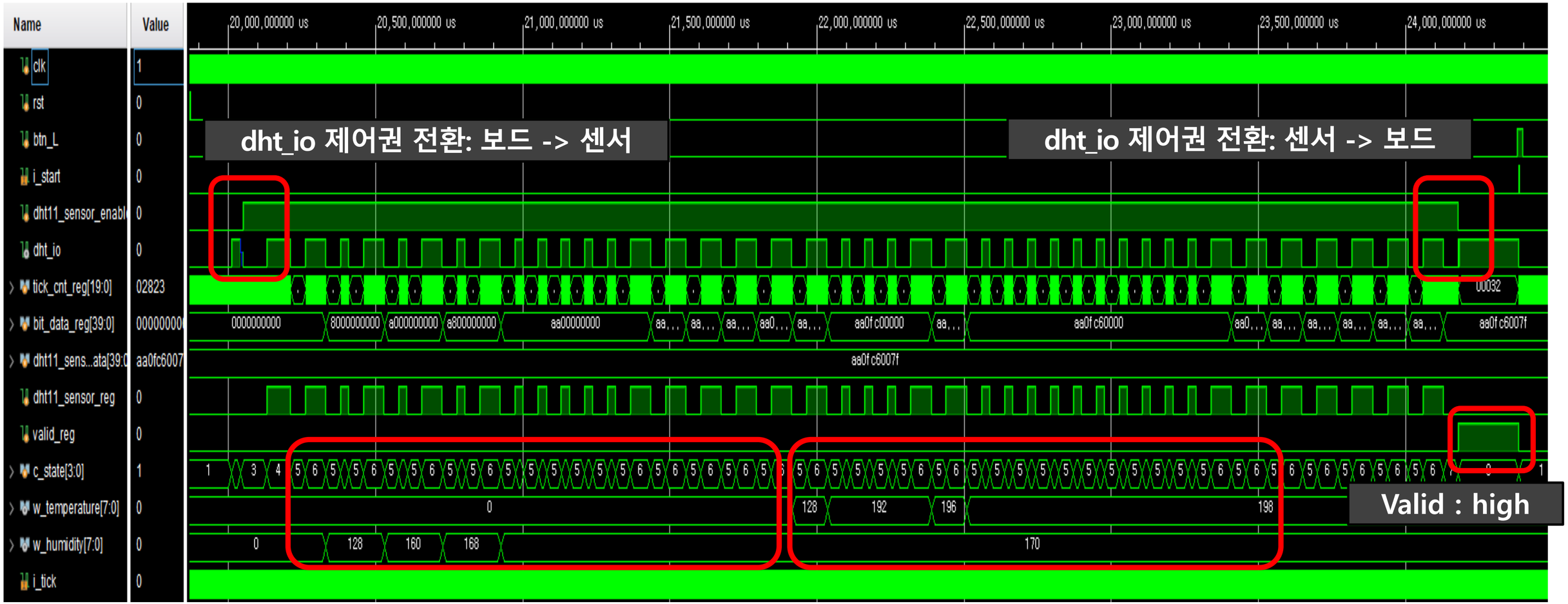


DHT-11 Simulation

Dht11 전체 simulation

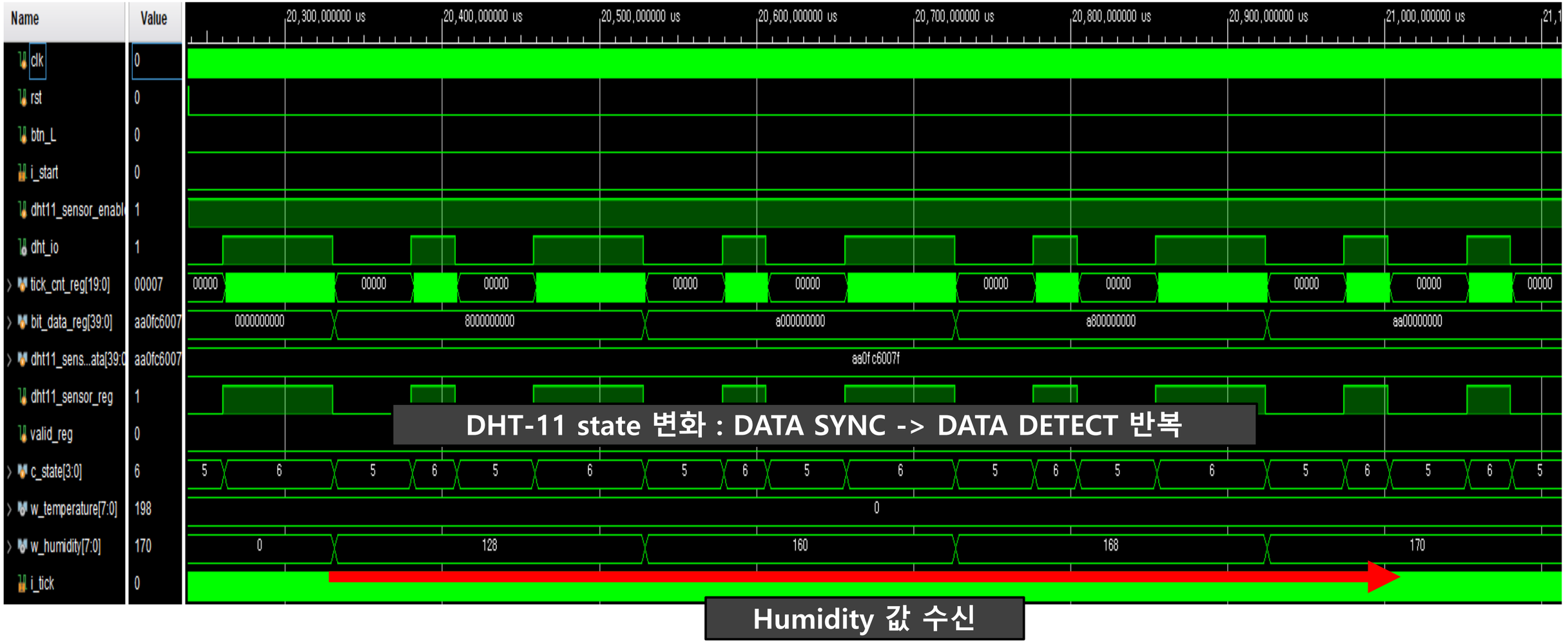
DHT-11 Simulation

dht_io 제어권 변화 / 센서 값 수신 (humidity – temperature)



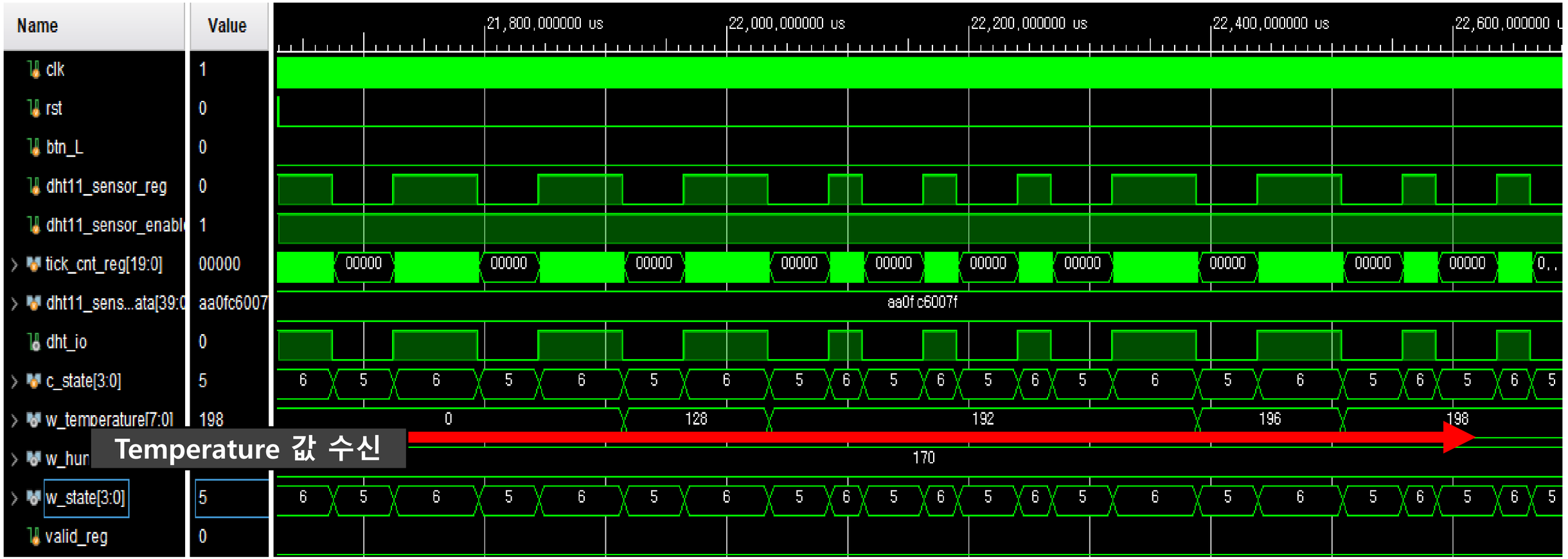
DHT-11 Simulation

humidity



DHT-11 Simulation

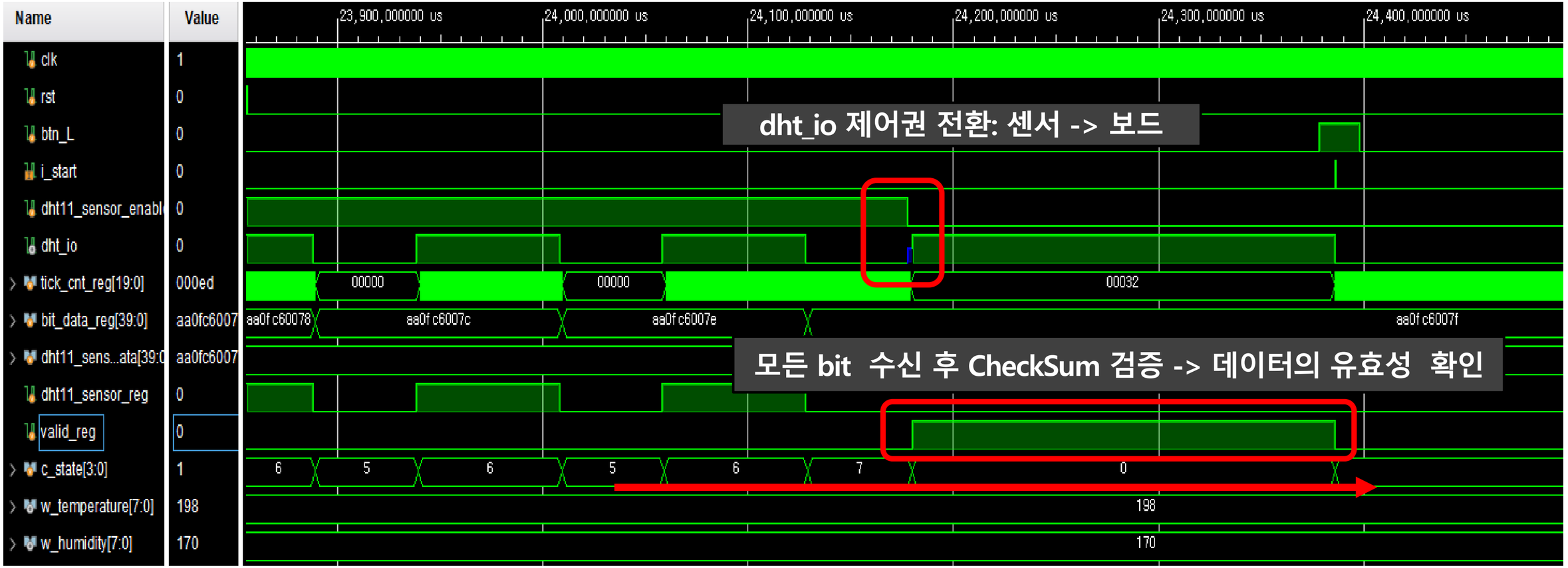
temp



DHT-11 state 변화 : DATA SYNC -> DATA DETECT 반복

DHT-11 Simulation

Valid (Check_sum)

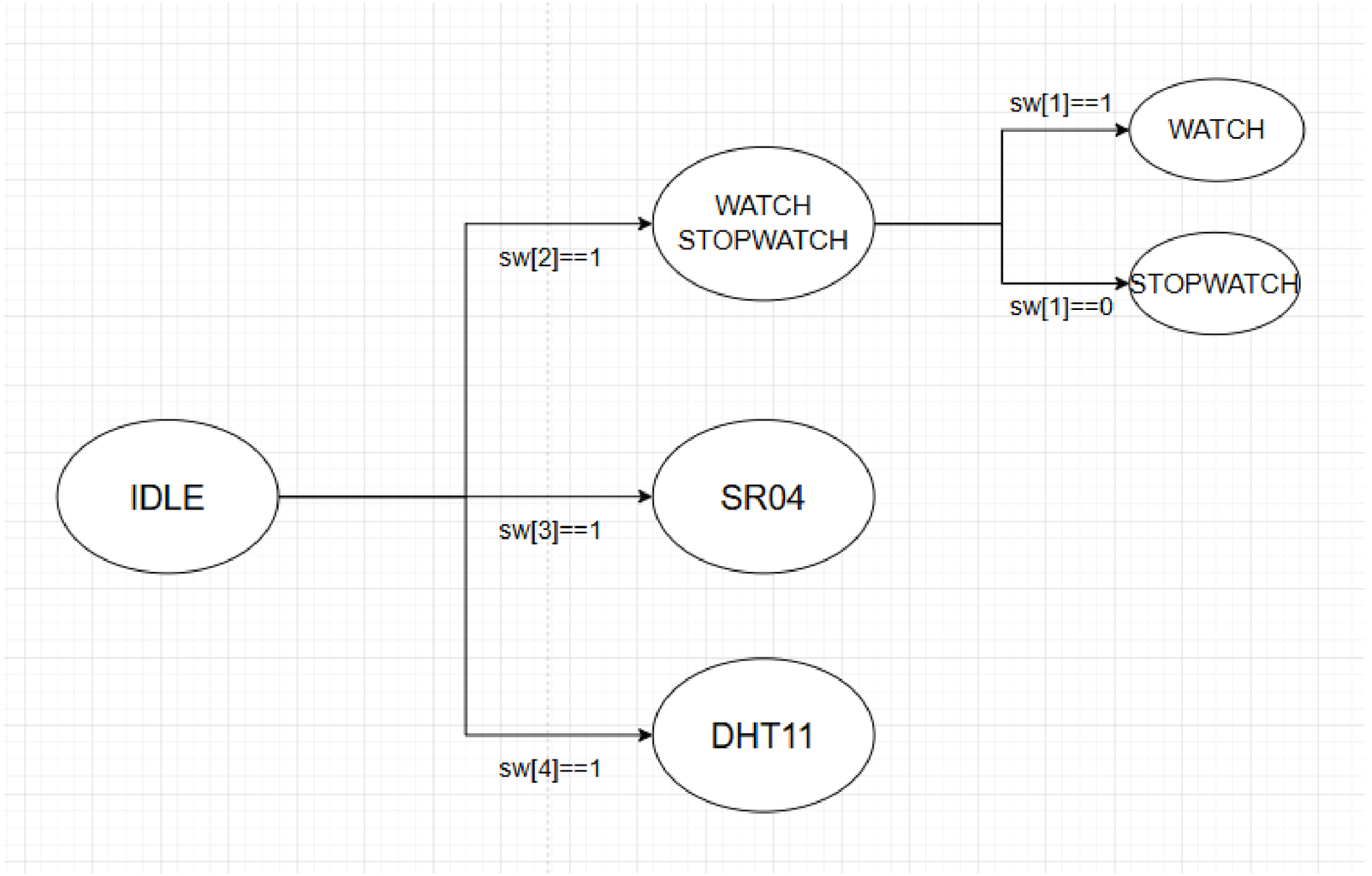


DHT-11 state 변화 : DATA SYNC -> DATA DETECT -> STOP -> IDLE

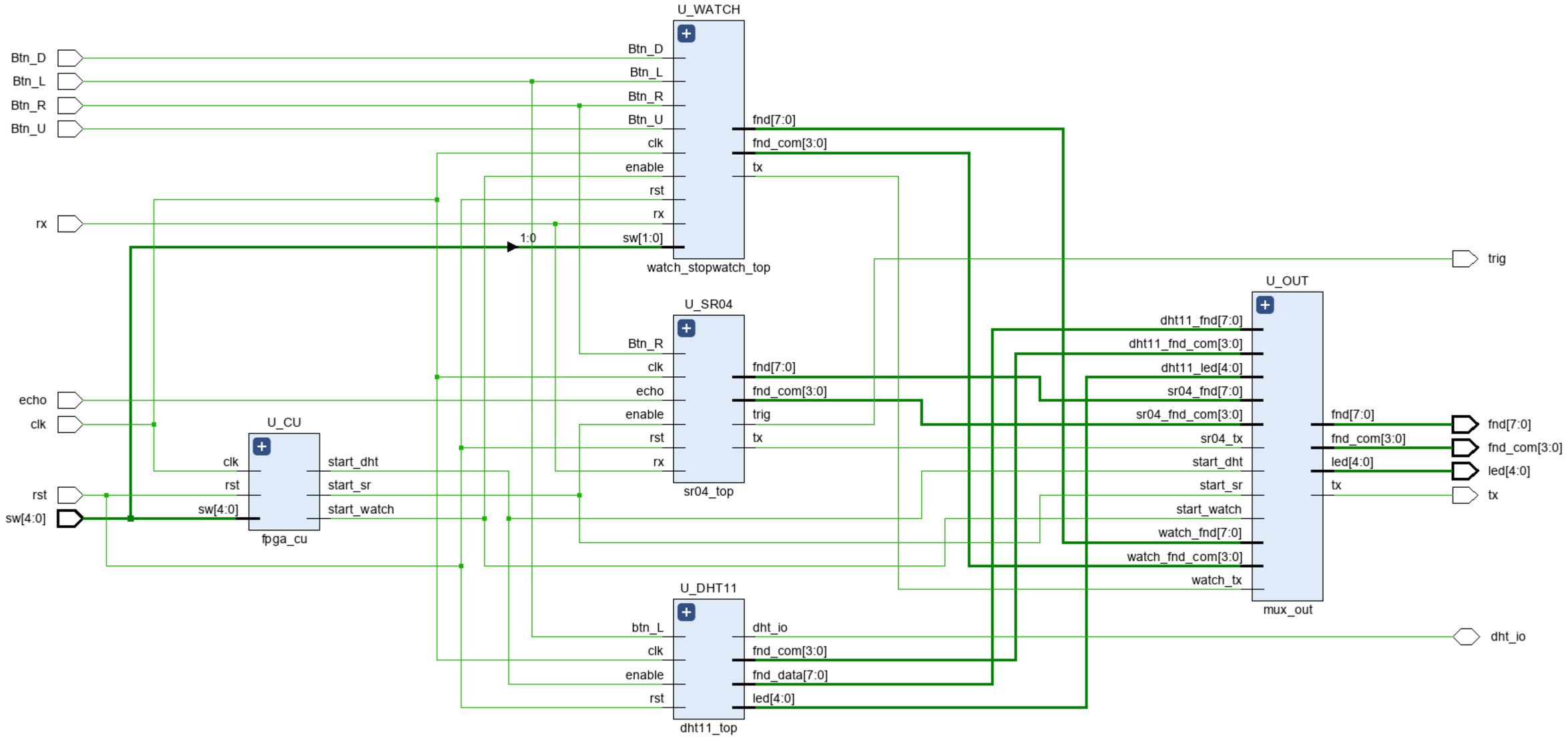
Top

Watch / Stopwatch + SR04 + DHT-11 + UART

블록다이어그램


IDLE
sw[2]==1
WATCH
STOPWATCH
sw[1]==1
WATCH
sw[1]==0
STOPWATCH
sw[3]==1
SR04
sw[4]==1
DHT11

SCHEMATIC


U_WATCH
Btn_D
Btn_L
Btn_R
Btn_U
clk
enable
rst
rx
sw[1:0]
fnd[7:0]
fnd_com[3:0]
tx
watch_stopwatch_top
1:0
U_SR04
Btn_R
clk
echo
enable
rst
rx
fnd[7:0]
fnd_com[3:0]
trig
tx
sr04_top
U_CU
clk
rst
sw[4:0]
start_dht
start_sr
start_watch
fpga_cu
U_DHT11
btn_L
clk
enable
rst
dht_io
fnd_com[3:0]
fnd_data[7:0]
led[4:0]
dht11_top
U_OUT
dht11_fnd[7:0]
dht11_fnd_com[3:0]
dht11_led[4:0]
sr04_fnd[7:0]
sr04_fnd_com[3:0]
sr04_tx
start_dht
start_sr
start_watch
watch_fnd[7:0]
watch_fnd_com[3:0]
watch_tx
fnd[7:0]
fnd_com[3:0]
led[4:0]
tx
mux_out
Btn_D
Btn_L
Btn_R
Btn_U
rx
echo
clk
rst
sw[4:0]
trig
fnd[7:0]
fnd_com[3:0]
led[4:0]
tx
dht_io

Control Unit

```
module fpga_cu (
    input [4:0] sw,
    output reg start_watch,
    output reg start_sr,
    output reg start_dht
);

    always @(*) begin
        start_watch = 0;
        start_sr = 0;
        start_dht = 0;

        if (sw[4]) begin
            start_dht = 1;
        end else if (sw[3]) begin
            start_sr  = 1;
        end else if (sw[2]) begin
            start_watch = 1;
        end
    end
endmodule
```

동작 시작 신호

```
dht11_top U_DHT11 (
    .clk(clk),
    .rst(rst),
    .enable(start_dht),
sr04_top U_SR04 (
    .clk(clk),
    .rst(rst),
    .enable(start_sr),
watch_stopwatch_top U_WATCH (
    .clk(clk),
    .rst(rst),
    .enable(start_watch),
```

우선순위 : sw[4] > sw[3] > sw[2]

enable port로 start 신호 입력

Control Unit

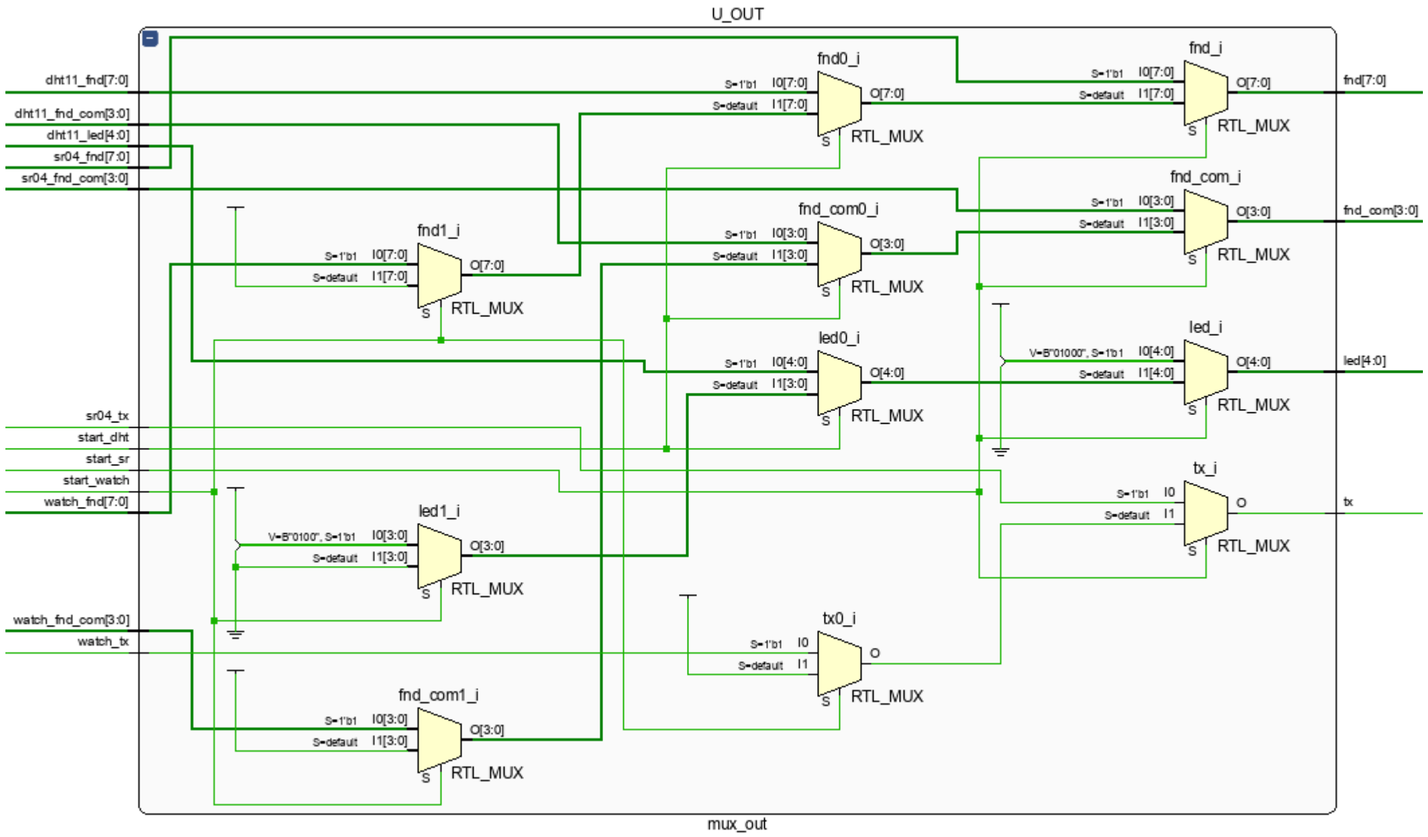
```
assign start_cmd = enable && (w_start || w_uart_start);
sr04_controller u_sr04_controller (
    .clk(clk),
    .rst(rst | ~enable),
    .i_tick(w_tick_1us),
    .start(start_cmd),
    .echo(echo),
    .o_trig(trig),
    .start_send(w_start_send),
    .o_dist(w_o_dist)
);
```

enable 와 start 신호를 AND Gate 처리

```
clock_top u_clock_top (
    .clk        (clk),
    .rst        (rst | ~enable),
    .sw0        (enable ? sw[0] : 1'b0),
    .sw1        (enable ? sw[1] : 1'b0),
    .Btn_L_watch(enable ? watch_h_up : 1'b0),
    .Btn_L_sw   (enable ? Btn_L_sw : 1'b0),
    .Btn_U      (enable ? watch_m_up : 1'b0),
    .Btn_D      (enable ? watch_s_up : 1'b0),
    .Btn_R      (enable ? Btn_R_sw : 1'b0),
    .uart_mode  (enable ? (cmd_mode | sw[1]) : 1'b0),
    .fnd_com    (fnd_com),
    .fnd        (fnd)
);
```

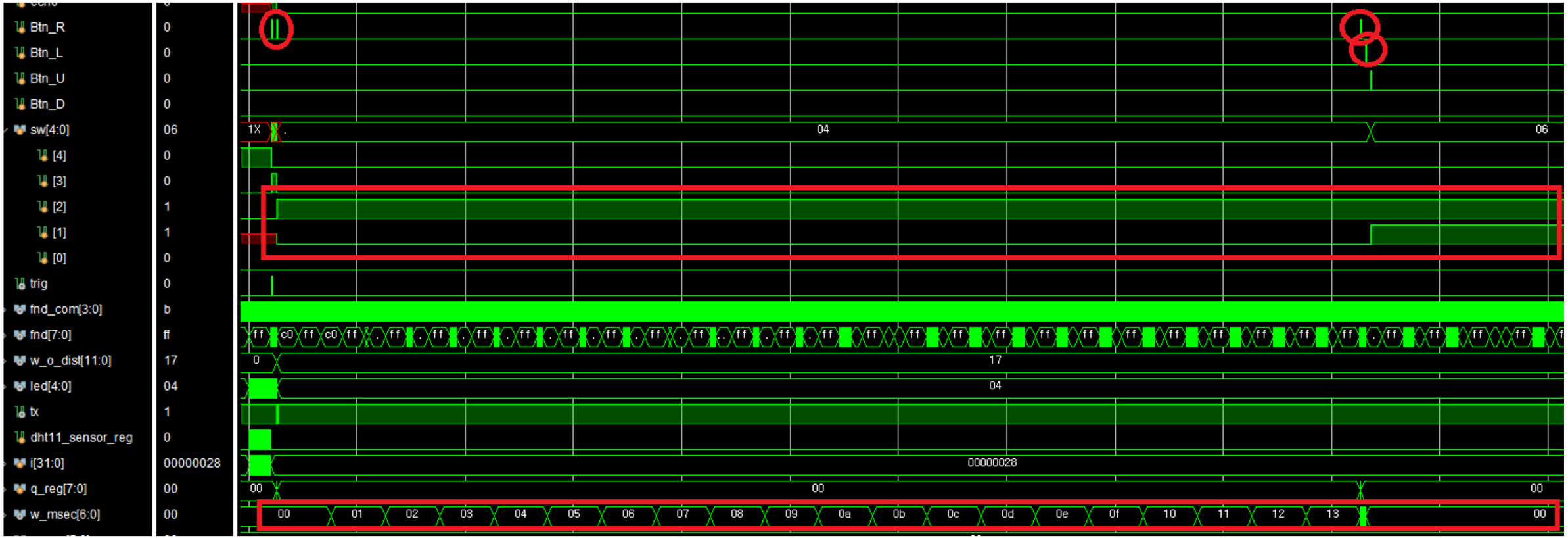
enable 신호가 1일 때만 값 출력

Mux_out




```
module mux_out (
    input        start_sr,
    input        start_dht,
    input        start_watch,
    input  [3:0] sr04_fnd_com,
    input  [7:0] sr04_fnd,
    input        sr04_tx,
    input  [3:0] dht11_fnd_com,
    input  [7:0] dht11_fnd,
    input  [4:0] dht11_led,
    input  [3:0] watch_fnd_com,
    input  [7:0] watch_fnd,
    input        watch_tx,
    output [3:0] fnd_com,
    output [7:0] fnd,
    output [4:0] led,
    output       tx
);

    assign fnd_com = start_sr    ? sr04_fnd_com  :
                     start_dht   ? dht11_fnd_com :
                     start_watch ? watch_fnd_com :
                                   4'b1111;

    assign fnd     = start_sr    ? sr04_fnd      :
                     start_dht   ? dht11_fnd     :
                     start_watch ? watch_fnd     :
                                   8'b11111111;

    assign led     = start_sr    ? 5'b01000      :
                     start_dht   ? dht11_led     :
                     start_watch ? 5'b00100      :
                                   5'b00000;

    assign tx = start_sr ? sr04_tx : start_watch ? watch_tx : 1'b1;

endmodule
```

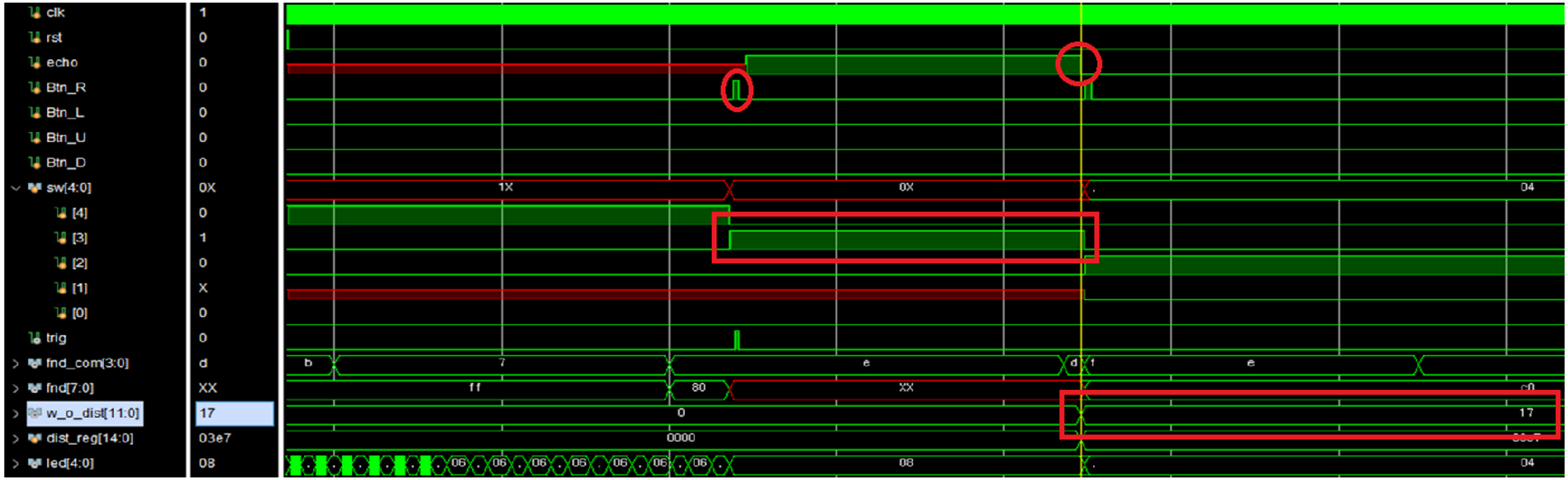
WATCH_STOPWATCH SIMULATION



SW[2]==1 상태에서 Btn_R을 눌러 실행 -> Stopwatch 작동

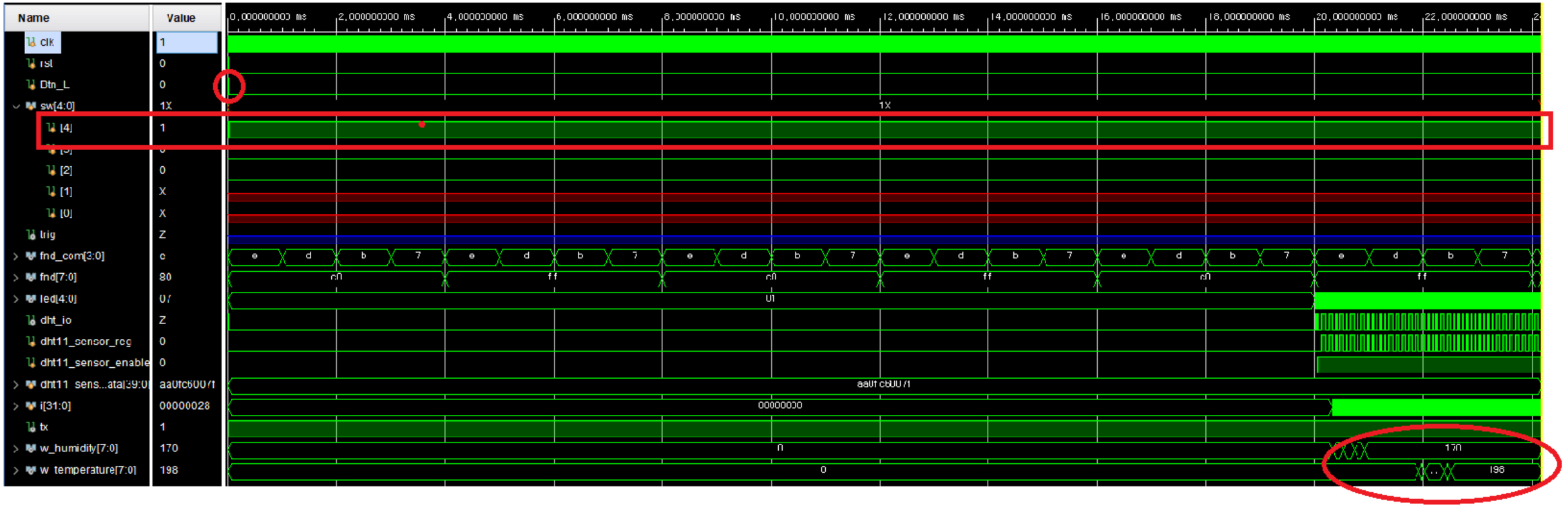
SW[2]==1 상태에서 Btn_R을 다시 눌러 STOP 후 Btn_L로 Clear

SR04 SIMULATION



SW[3]== && Right_Button : 초음파센서 작동 -> 거리값 출력

DHT11 SIMULATION



SW[4]==1 && Left_Button : 온도/습도 센서(DHT11) 작동

Trouble Shooting

문제점

- MUX로 FND 출력을 제어해 모드가 변경된 후에도 기존 값이 저장되어있는 현상

solution

- Reset과 ~enable 신호를 OR 게이트를 통해 모드가 바뀔 때마다 값을 리셋

```
sr04_controller u_sr04_controller (
    .clk(clk),
    .rst(rst | ~enable),
```

```
dht11_control_unit U_DHT11_CU (
    .clk(clk),
    .rst(rst | ~enable),
```

```
clock_top u_clock_top (
    .clk        (clk),
    .rst        (rst | ~enable),
```

Trouble Shooting

문제점

Implementation (1 critical warning)

Route Design (1 critical warning)

[Timing 38-282] The design failed to meet the timing requirements. Please see the timing summary report for details on the timing violations.

```
        state <= next;
        trig_reg <= trig_next;
        dist_reg <= dist_next / 58;
        tick_cnt_reg <= tick_cnt_next;
        start_send_reg <= start_send_next;
    end
end

assign o_dist = dist_reg;
```

Worst Negative Slack (WNS):	-1.627 ns
Total Negative Slack (TNS):	-18.340 ns
Number of Failing Endpoints:	12
Total Number of Endpoints:	885

나눗셈 연산을 한 클럭에서 처리해 타이밍 지연 발생

Solution

```
        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

assign o_dist = dist_div_reg;
```

Worst Negative Slack (WNS):	3.866 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	646

두 단계 파이프라인으로 나눠서
레지스터에 저장하여 한 클럭 늦게 처리

동작 영상