대한상공회의소
서울기술교육센터

AI 시스템 반도체 설계 2기

FINAL PROJECT

2조 신상학, 엄찬하, 이현수, 정은지

UART + WATCH/STOPWATCH + SR04 sensor + HDT11 sensor

CONTENTS

CONTENTS

REVIEW

CHAPTER 01

지난 프로젝트에서는 FPGA BOARD에 WATCH 및 STOPWATCH를 구현하였습니다.
BUTTON 및 ComPortMaster를 통해 명령어로 기능을 제어하며
UART 통신에 대하여 학습하였습니다.

RIVIEW

AS IS	TO BE
FPGA BOARD 7 SEGMENT에 WATCH/STOPWATCH 기능 구현	초음파 센서로 측정한 거리를 7 Seg Display에 구현
	온·습도로 측정한 결과를 7 Seg Display 에 구현
UART 통신을 통해 명령어로 WATCH/STOPWATCH 제어	UART 통신에 FIFO 버퍼 추가
	센서 측정 결과를 UART 통신을 통해 송신

Conclusion : UART + WATCH/STOPWATCH + SR04 sensor + DHT11 sensor

OVERVIEW

CHAPTER 02

WATCH/STOPWATCH를 구현하고 BUTTON 및 UART 명령어로 제어한 데 이어
초음파 센서와 온·습도 센서를 사용하여 센서 결과를 FPGA Board 7 segment display에 연출하고,
UART 송신을 통해 ComPortMaster에 출력하는 것이 목표입니다.

OVERVIEW

개발 환경

개발 언어: Verilog HDL
시뮬레이션: Xilinx Vivado
에디터: VS code
UART 송수신 확인: ComPortMaster

Basys 3 FPGA Board

USED I/O Pins
WATCH/STOPWATCH:
Switch, Button, Led, 7-Seg Display
SRO4 sensor, HDT11 sensor: Switch
작동 전압: 3.3 V(logical level), 5 V

HC – SR04 sensor

초음파를 통해 물건을 감지
감지 범위: 2 cm ~ 400cm
작동 전압: 3.5 V ~ 5 V

DHT11 sensor

주변 환경의 습도와 온도를 측정
습도 측정 범위: 20% ~ 95%
온도 측정 범위: 0 ~ 50도
작동 전압: 3.3 V ~ 5 V

Final Project

UART 통신을 통해
Watch/Stopwatch 제어
및 SR04, HDT11 센서
결과 출력

OVERVIEW

프로젝트 구성 및 세부 목표

- **Watch/Stopwatch**
  - WATCH/STOPWATCH 출력 구현
  - BUTTON(U/D/L/R)으로 STATE 변경
  - PC → FPGA 명령어 수신: WATCH/STOPWATCH 제어
- **SR04 Sensor**
  - 초음파 센서를 통해 측정한 거리를 출력 구현
  - FPGA → PC UART 송신
- **DHT11 Sensor**
  - 습·온도 센서를 통해 측정한 결과를 출력 구현
  - FPGA → PC UART 송신
- **MERGE**
  - UART + WATCH/STOPWATCH + SR04 + HDT11
  - 스위치로 MODE, UART TX DATA SELECT & FND 통합

* FPGA Board 7 Segment Display에 출력 구현

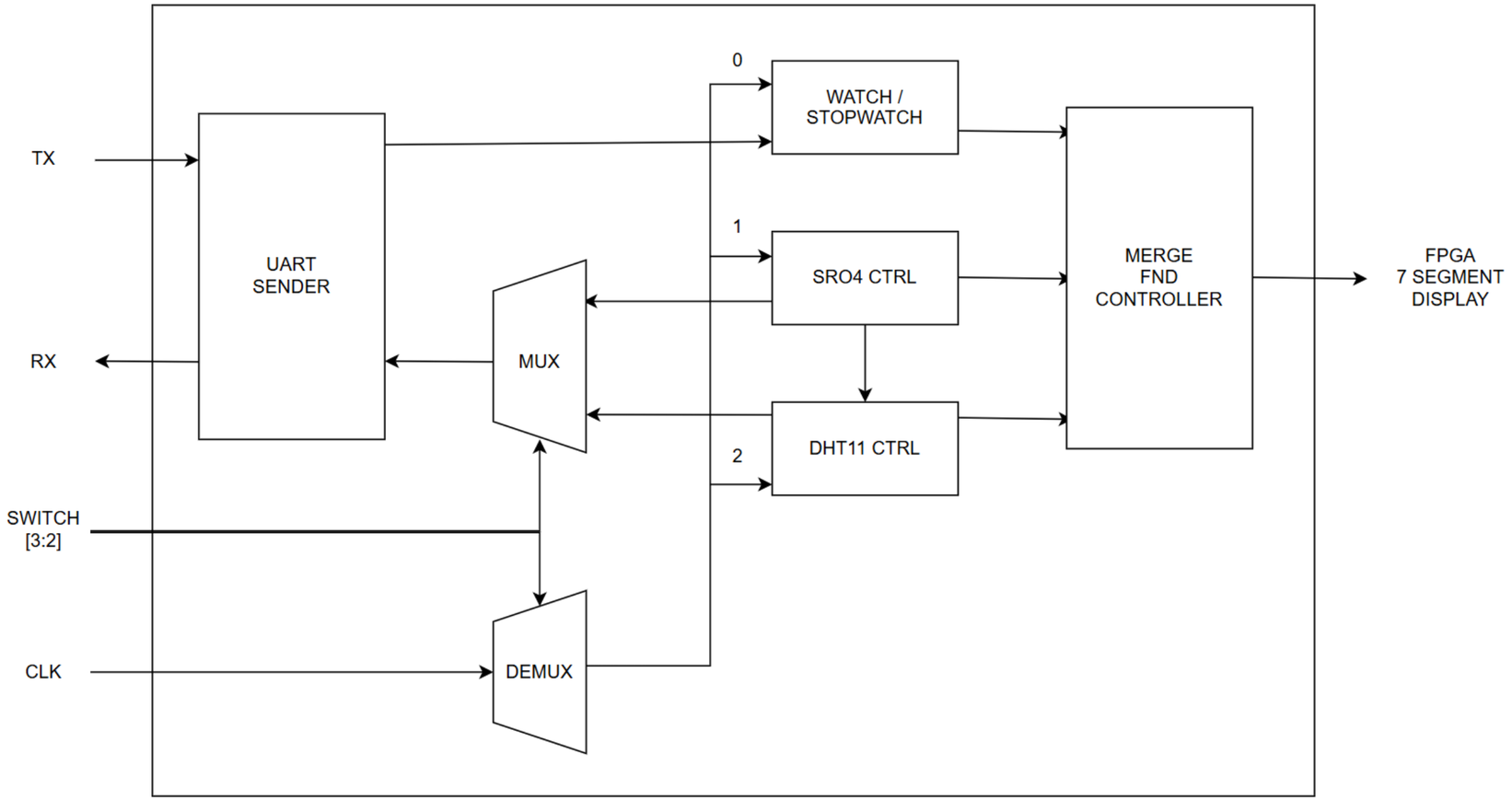
* ComPortMaster에서 UART 통신 송·수신 결과를 확인

MERGE

CHAPTER 03

UART에 전송할 DATA를 고르는 MUX, SWTICH에 따른 작동을 결정하는 DEMUX,
모든 FND CTRL를 통합한 TOP FND CONTROLLER를 생성했습니다.

FULL BLOCK DIAGRAM

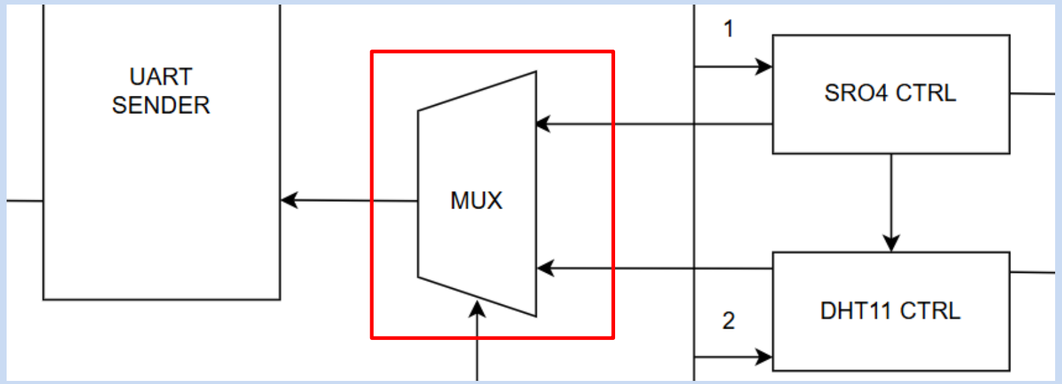


UART CONTROL

1

BLOCK DIAGRAM

전송할 신호 SELECT



2

UART CONROL PART CODE

전송할 신호 SELECT

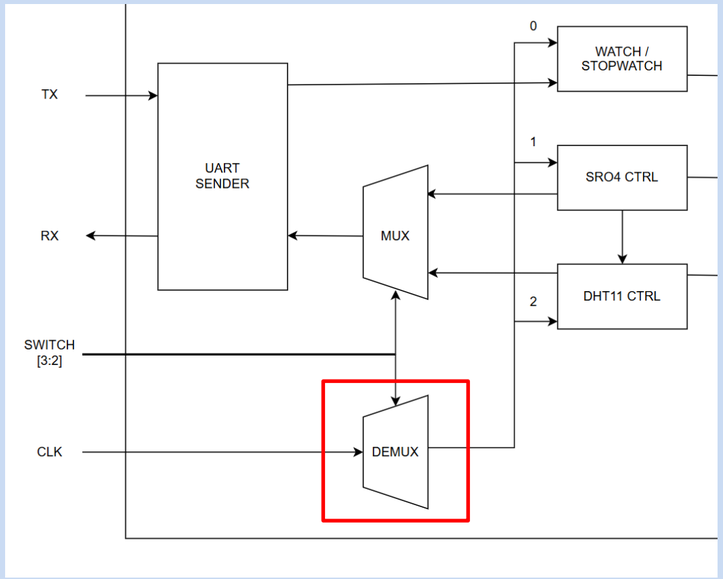
```
assign selected_data  = (mode_sel == 2'b01) ?
                        w_dist : (mode_sel == 2'b10) ?
                        ({w_rh_data, w_t_data}) : 16'd0;

sender_uart U_sender_uart (
    .clk(clk),
    .rst(rst),
    .rx(rx),
    .i_send_data(selected_data),
    .btn_sender_up(w_done_dht11),
    .btn_sender_down(w_done_sr04),
    .tx(tx_sensor),
    .tx_done()
);
```

SWITCH CONTROL

1

BLOCK DIAGRAM

전송할 신호 SELECT



2

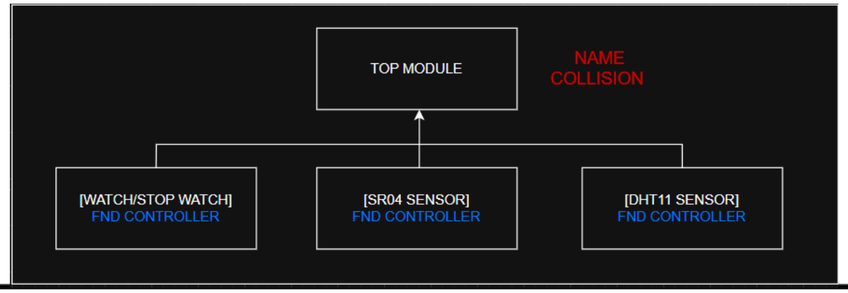
UART CONROL PART CODE

전송할 신호 SELECT

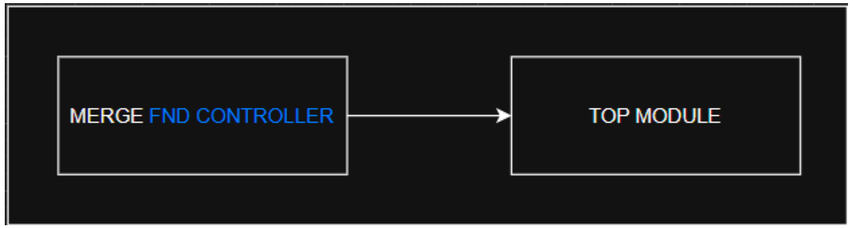
```
module demux (  // demux 추가
    input clk,
    input [3:0] sw,
    output [2:0] o_clk
);

    reg clk_stw, clk_sr04, clk_hdt;

    assign o_clk = {clk_hdt, clk_sr04, clk_stw};

    always @(*) begin
        clk_stw  = (sw[3:2] == 2'b00) ? clk : 0;
        clk_sr04 = (sw[3:2] == 2'b01) ? clk : 0;
        clk_hdt  = (sw[3:2] == 2'b10) ? clk : 0;
    end

endmodule
```

FND MERGE

문제: FND Controller 이름 중복



해결: Watch/Stopwatch, SR04, DHT11 FND 통합



FND MERGE CODE

```
module fnd_controller_top (
    input         clk,
    input         reset,
    input  [3:0] sw,

    input  [6:0] msec_s,
    input  [5:0] sec_s,
    input  [5:0] min_s,
    input  [4:0] hour_s,
    input  [6:0] msec_r,
    input  [5:0] sec_r,
    input  [5:0] min_r,
    input  [4:0] hour_r,

    input  [9:0] dist,
    input  [7:0] t_data,
    input  [7:0] rh_data,

    output [7:0] fnd_data,
    output [3:0] fnd_com
);
```

```
////////////////////////// 센서용 ////
digit_spliter_sr04 #(
    .BIT_WIDTH(10)
) U_DigitSpliter_SR04 (
    .count_data(dist),
    .digit_1  (w_dist_1),
    .digit_10 (w_dist_10),
    .digit_100(w_dist_100),
    .digit_1000(w_dist_1000)
);

digit_spliter #(
    .BIT_WIDTH(8)
) U_DigitSpliter_DHT11_T (
    .count_data(t_data),
    .digit_1  (w_t_1),
    .digit_10 (w_t_10)
);

digit_spliter #(
    .BIT_WIDTH(8)
) U_DigitSpliter_DHT11_RH (
    .count_data(rh_data),
    .digit_1  (w_rh_1),
    .digit_10 (w_rh_10)
);
////////////////////////////////////
```

SR04 + SENDER_UART

CHAPTER 04

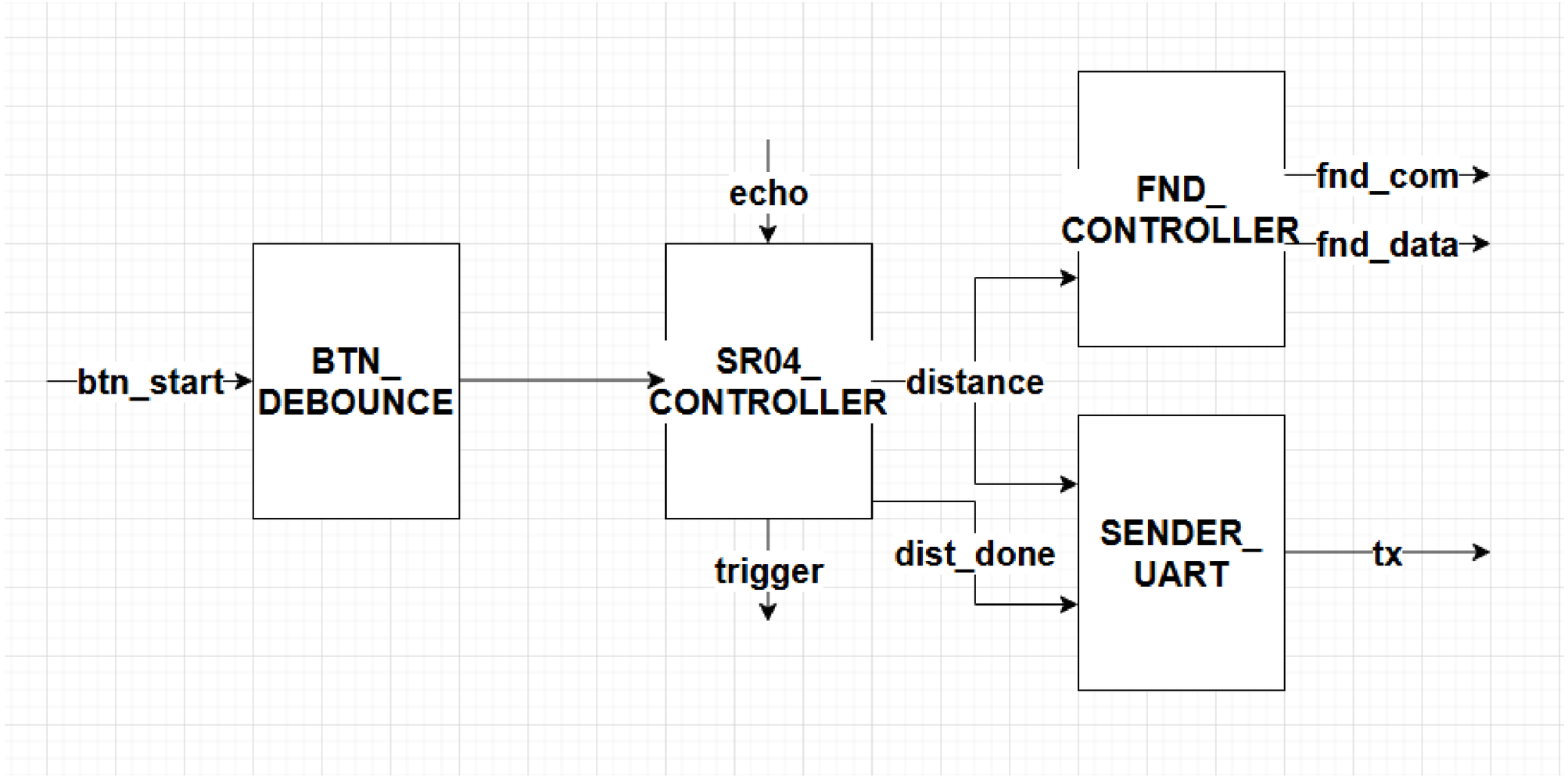
신상학

초음파를 통해 거리를 측정하는 SR04 센서와 FIFO 및 UART를 이용하여
측정 데이터를 받는 동작을 구현

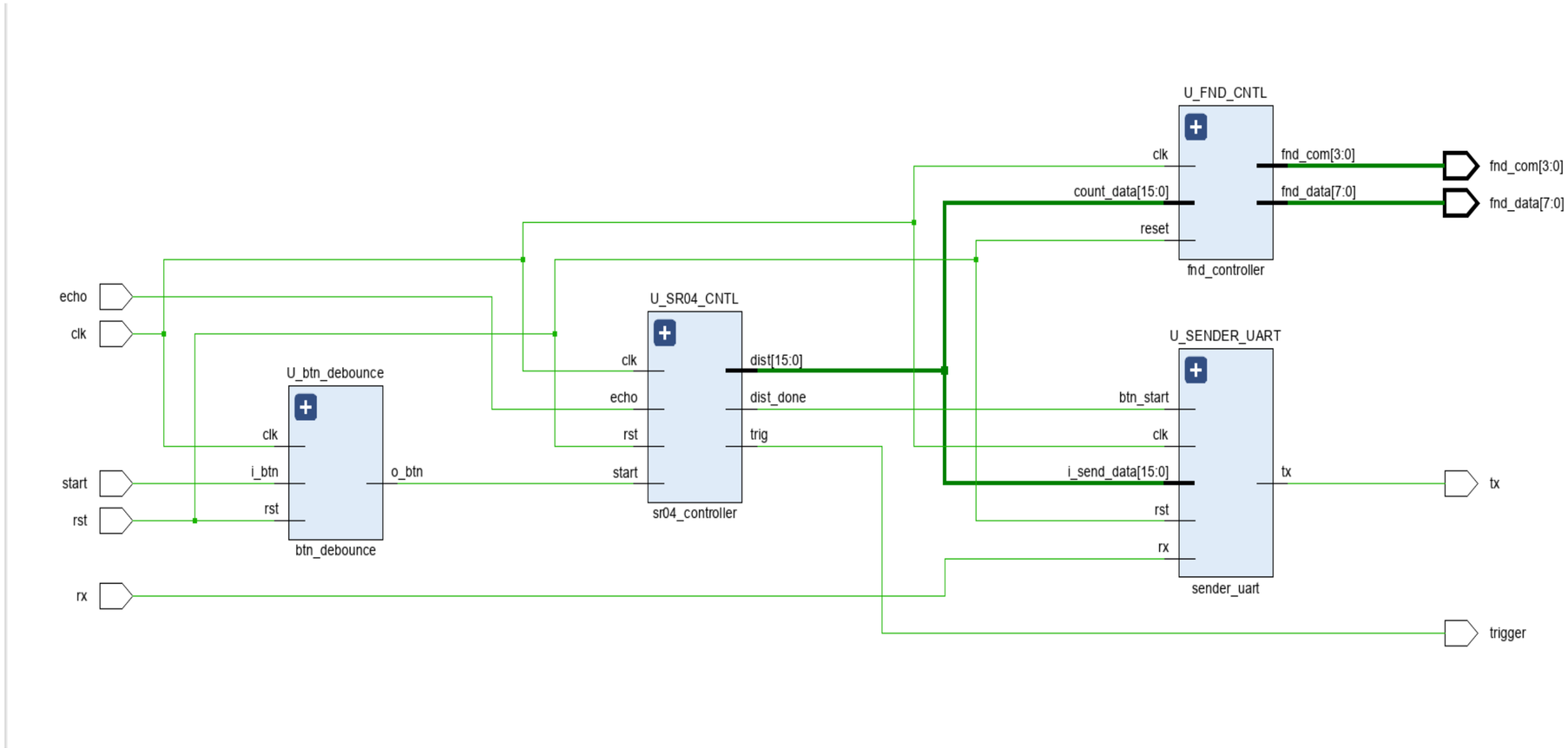
목차

- Block Diagram & Schematic
  - SR04 센서 + Sender UART 모듈
  - FSM 및 ASM
- AS-IS TO-BE
  - 기능적 개선 사항
- Simulation Verification
  - 시뮬레이션 결과 및 분석
  - 시연 영상
- Trouble Shooting
  - 어려웠던 점 + 해결 방법

Block Diagram

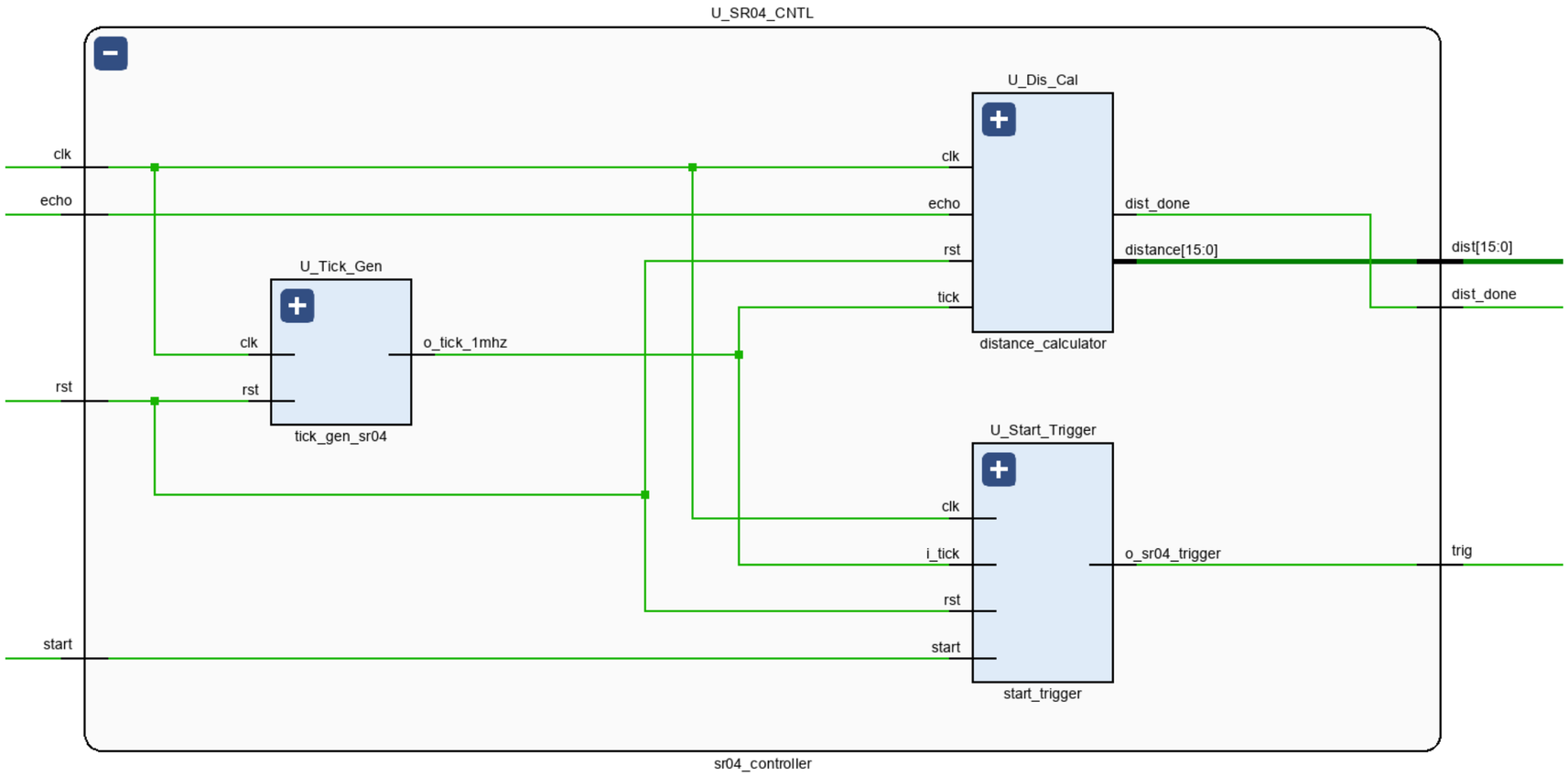


Schematic of TOP



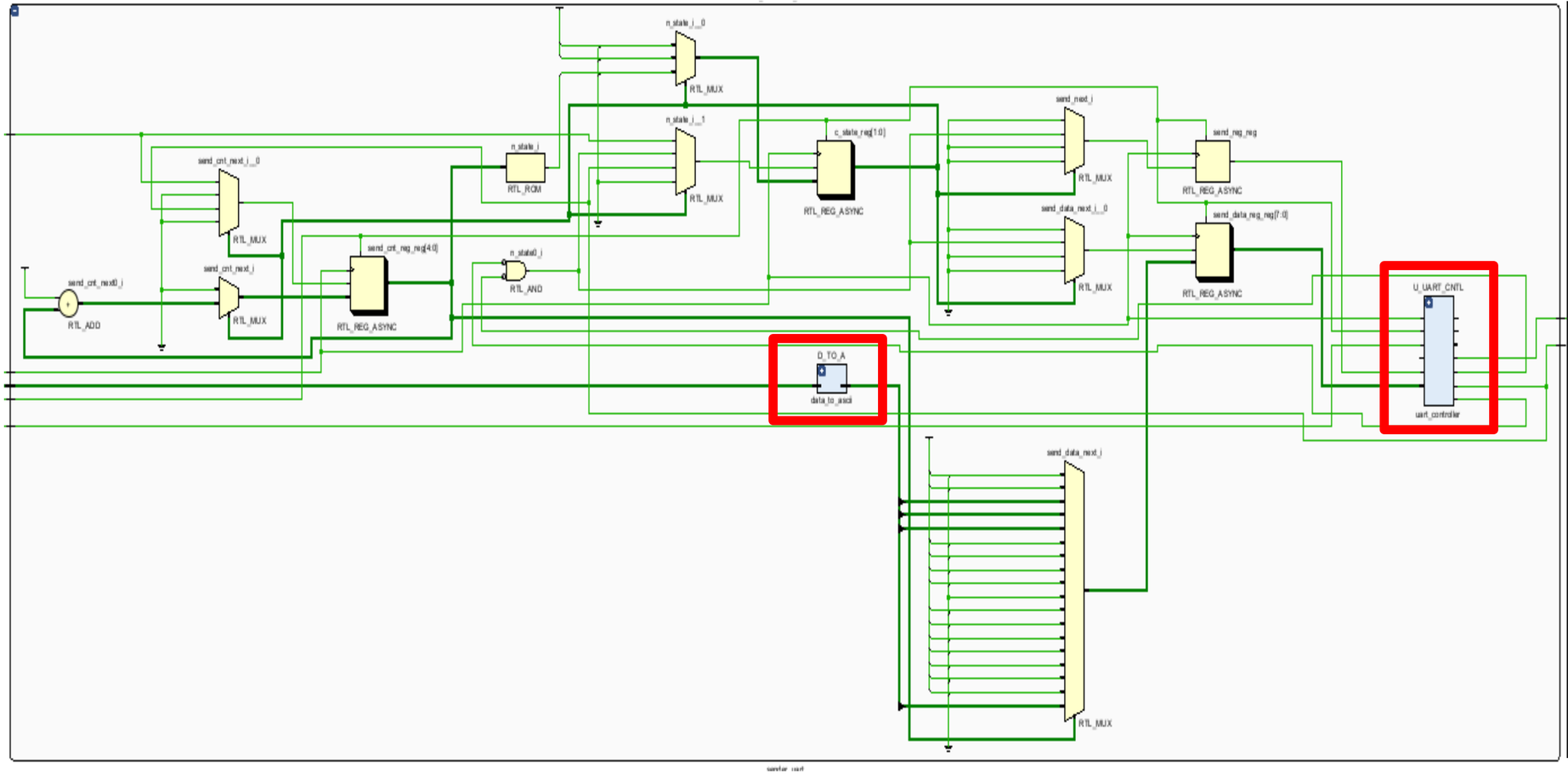

```
btn_debounce U_btn_debounce (
    .clk  (clk),
    .rst  (rst),
    .i_btn(start),
    .o_btn(w_start)
);

sender_uart U_SENDER_UART (
    .clk(clk),
    .rst(rst),
    .rx(rx),
    .i_send_data(w_dist),
    .btn_start(w_done),
    .tx(tx),
    .tx_done()
);

sr04_controller U_SR04_CNTL (
    .clk(clk),
    .rst(rst),
    .start(w_start),
    .echo(echo),
    .trig(trigger),
    .dist(w_dist),
    .dist_done(w_done)
);

fnd_controller U_FND_CNTL (
    .clk(clk),
    .reset(rst),
    .count_data(w_dist),
    .fnd_data(fnd_data),
    .fnd_com(fnd_com)
);
```

Schematic of SR04_Controller



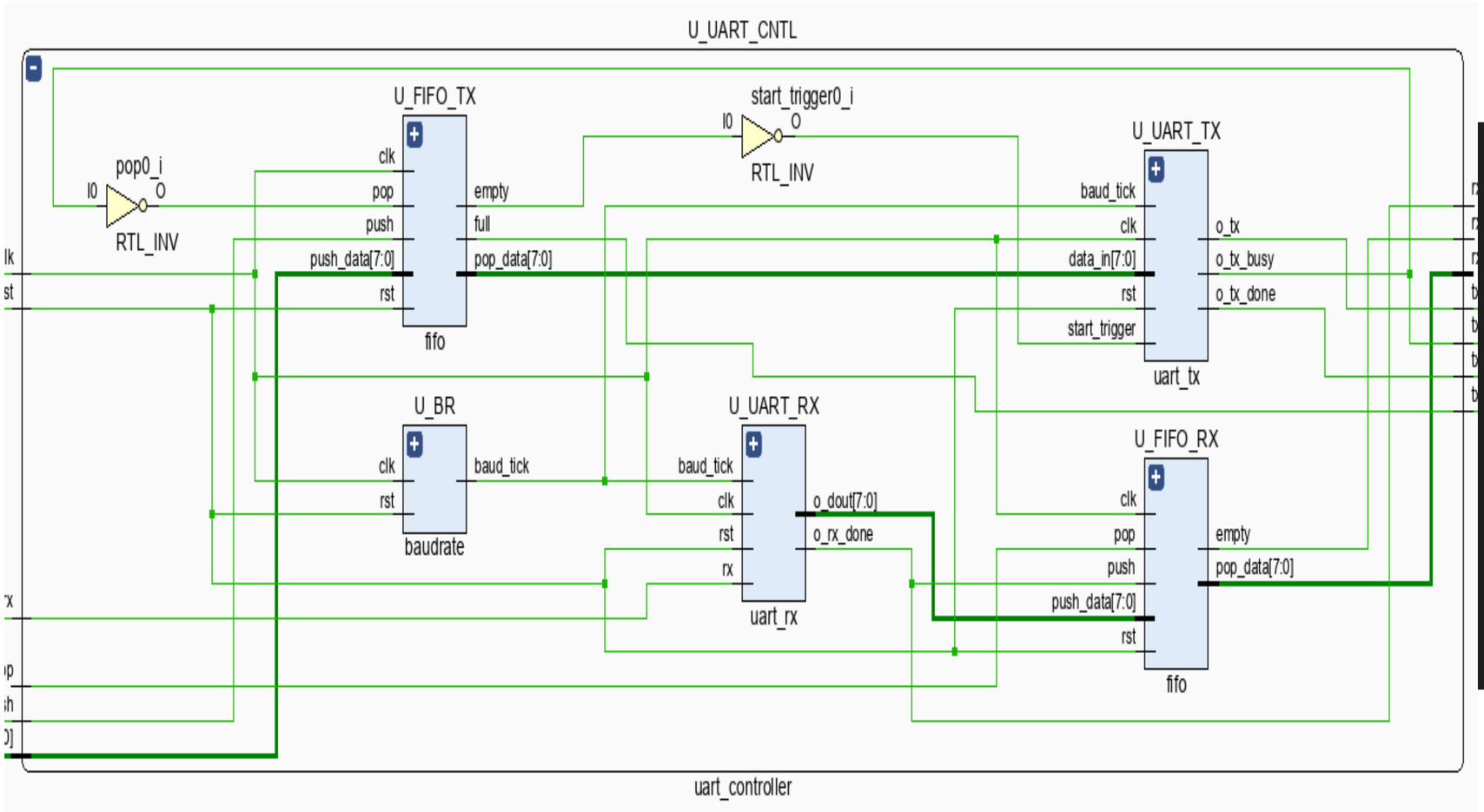

```
distance_calculator U_Dis_Cal(
.clk(clk),
.rst(rst),
.echo(echo),
.tick(w_tick),
.distance(dist),
.dist_done(dist_done)
);

start_trigger U_Start_Trigger(
    .clk(clk),
    .rst(rst),
    .i_tick(w_tick),
    .start(start),
    .o_sr04_trigger(trig)
);

tick_gen_sr04 #(
    .F_COUNT(100)
) U_Tick_Gen(
    .clk(clk),
    .rst(rst),
    .o_tick_1mhz(w_tick)
);
```

Schematic of Sender_UART




```
uart_controller U_UART_CNTL (
    .clk(clk),
    .rst(rst),
    .rx(rx),
    .rx_pop(),
    .tx_push_data(send_data_reg),
    .tx_push(send_reg),

    .rx_pop_data(),
    .rx_empty(),
    .rx_done(),
    .tx_full(w_tx_full),
    .tx_done(tx_done),
    .tx_busy(w_tx_busy),
    .tx(tx)
);

data_to_ascii D_TO_A (
    .i_data(i_send_data),
    .o_data(w_send_data)
);
```

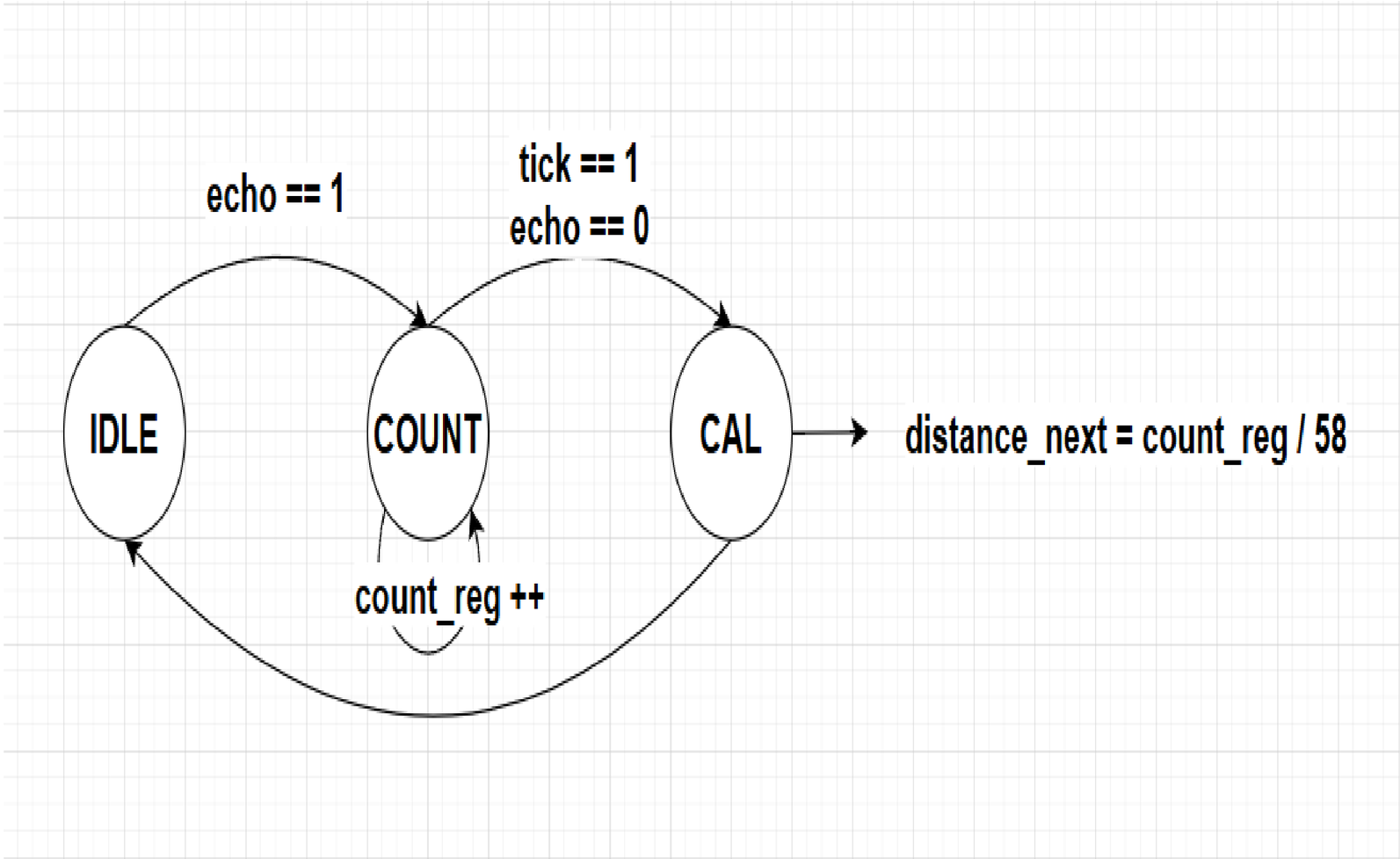
Schematic of UART_Controller




```
baudrate U_BR (
    .clk(clk),
    .rst(rst),
    .baud_tick(w_bd_tick)
);

uart_rx U_UART_RX (
    .clk(clk),
    .rst(rst),
    .baud_tick(w_bd_tick),
    .rx(rx),
    .o_rx_done(w_rx_done),
    .o_dout(w_rx_data)
);

fifo U_FIFO_RX (
    .clk(clk),
    .rst(rst),
    .push(w_rx_done),
    .pop(rx_pop),
    .push_data(w_rx_data),
    .full(),
    .empty(rx_empty),
    .pop_data(rx_pop_data)
);
```

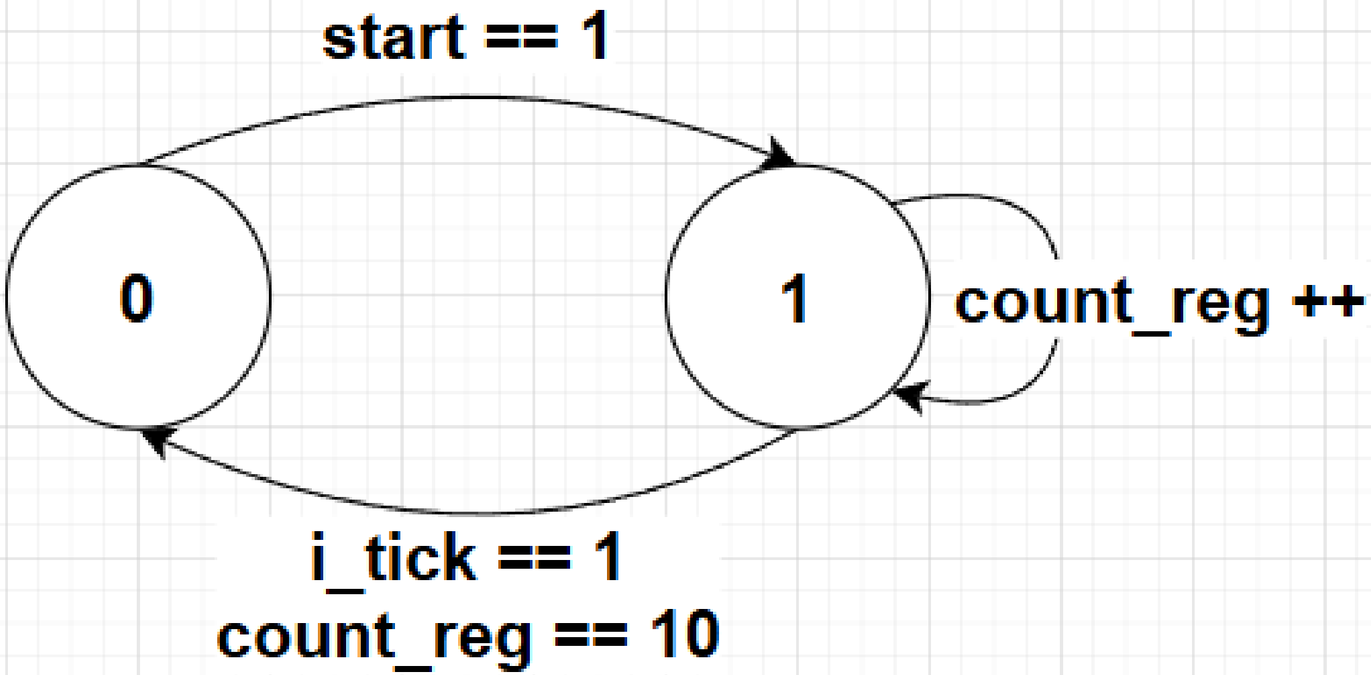
```
fifo U_FIFO_TX (
    .clk(clk),
    .rst(rst),
    .push(tx_push),
    .pop(~w_tx_busy),
    .push_data(tx_push_data),
    .full(tx_full),
    .empty(w_tx_start),
    .pop_data(w_tx_pop_data)
);

uart_tx U_UART_TX (
    .clk(clk),
    .rst(rst),
    .baud_tick(w_bd_tick),
    .start_trigger(~w_tx_start),
    .data_in(w_tx_pop_data),
    .o_tx_done(tx_done),
    .o_tx_busy(w_tx_busy),
    .o_tx(tx)
);
```

SR04 Controller (distance_calculator)



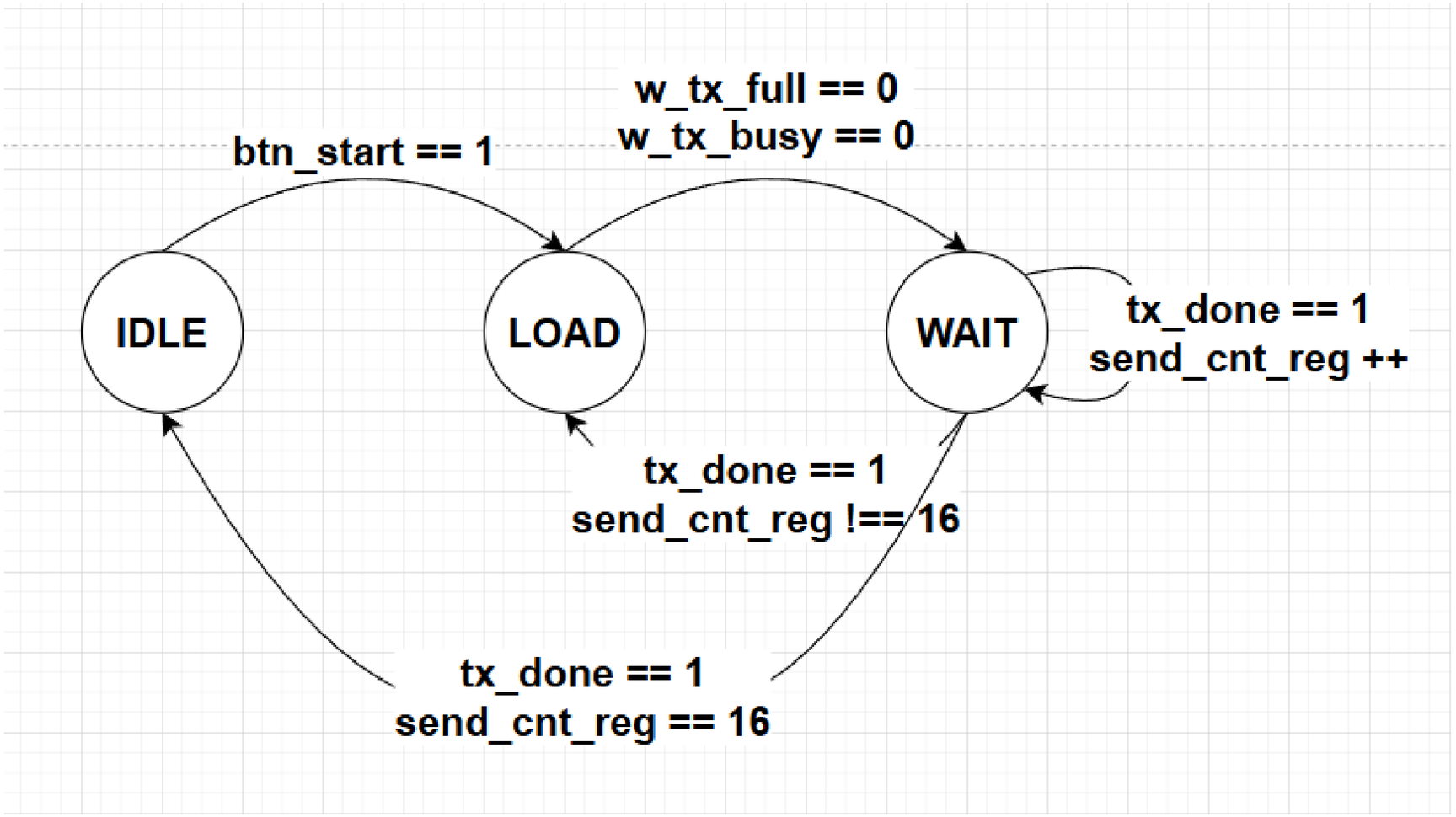

```
always @(*) begin
    state_next    = state_reg;
    count_next    = count_reg;
    distance_next = distance_reg;
    done_next     = done_reg; // 0
    case (state_reg)
        IDLE: begin
            done_next  = 1'b0;
            count_next = 0;
            if (echo) begin
                state_next = COUNT;
                distance_next = 0;
            end
        end
        COUNT: begin
            distance_next = 0;
            if (tick) begin
                if (echo == 0) begin
                    state_next = CAL;
                end
                count_next = count_reg + 1;
            end
        end
        CAL: begin
            distance_next = count_reg / 58;
            done_next = 1'b1;
            state_next = IDLE;
        end
    endcase
end
```

SR04 Controller (start_trigger)




```
always @(*) begin
    start_next      = start_reg;
    sr04_trigg_next = sr04_trigg_reg;
    count_next      = count_reg;
    case (start_reg)
        0: begin
            count_next = 0;
            sr04_trigg_next = 1'b0;
            if (start) begin
                start_next = 1;
            end
        end
        1: begin
            if (i_tick) begin
                sr04_trigg_next = 1'b1;
                if (count_reg == 10) begin
                    start_next = 0;
                end
                count_next = count_reg + 1;
            end
        end
    endcase
end
```

Sender_UART (Sender_UART)




```
always @(*) begin
    n_state = c_state;
    send_data_next = send_data_reg;
    send_next = 0;
    send_cnt_next = send_cnt_reg;
    case (c_state)
        IDLE: begin
            if (btn_start) begin
            send_cnt_next = 0;
                n_state = LOAD;
            end
        end

        LOAD: begin  // send
            send_next = 0;
            if (~w_tx_full && ~w_tx_busy) begin
                    case (send_cnt_reg)
                        0:  send_data_next = "D";
                        1:  send_data_next = "i";
                        2:  send_data_next = "s";
                        3:  send_data_next = "t";
                        4:  send_data_next = "a";
                        5:  send_data_next = "n";
                        6:  send_data_next = "c";
                        7:  send_data_next = "e";
                        8:  send_data_next = ":";
                        9:  send_data_next = w_send_data[31:24];
                        10: send_data_next = w_send_data[23:16];
                        11: send_data_next = w_send_data[15:8];
                        12: send_data_next = w_send_data[7:0];
                        13: send_data_next = "c";
                        14: send_data_next = "m";
                        15: send_data_next = "\n";
                        16: send_data_next = "\r";
                        default: begin
                            send_data_next = 0;
                            n_state = IDLE;
                        end
                    endcase
                    send_next=1;
                    n_state    = WAIT;
                end
                end
        WAIT: begin
            if(tx_done) begin
                send_cnt_next = send_cnt_reg + 1;
                if (send_cnt_reg == 16) n_state = IDLE;
                else n_state = LOAD;
            end
        end
    endcase
end
```

AS-IS TO-BE

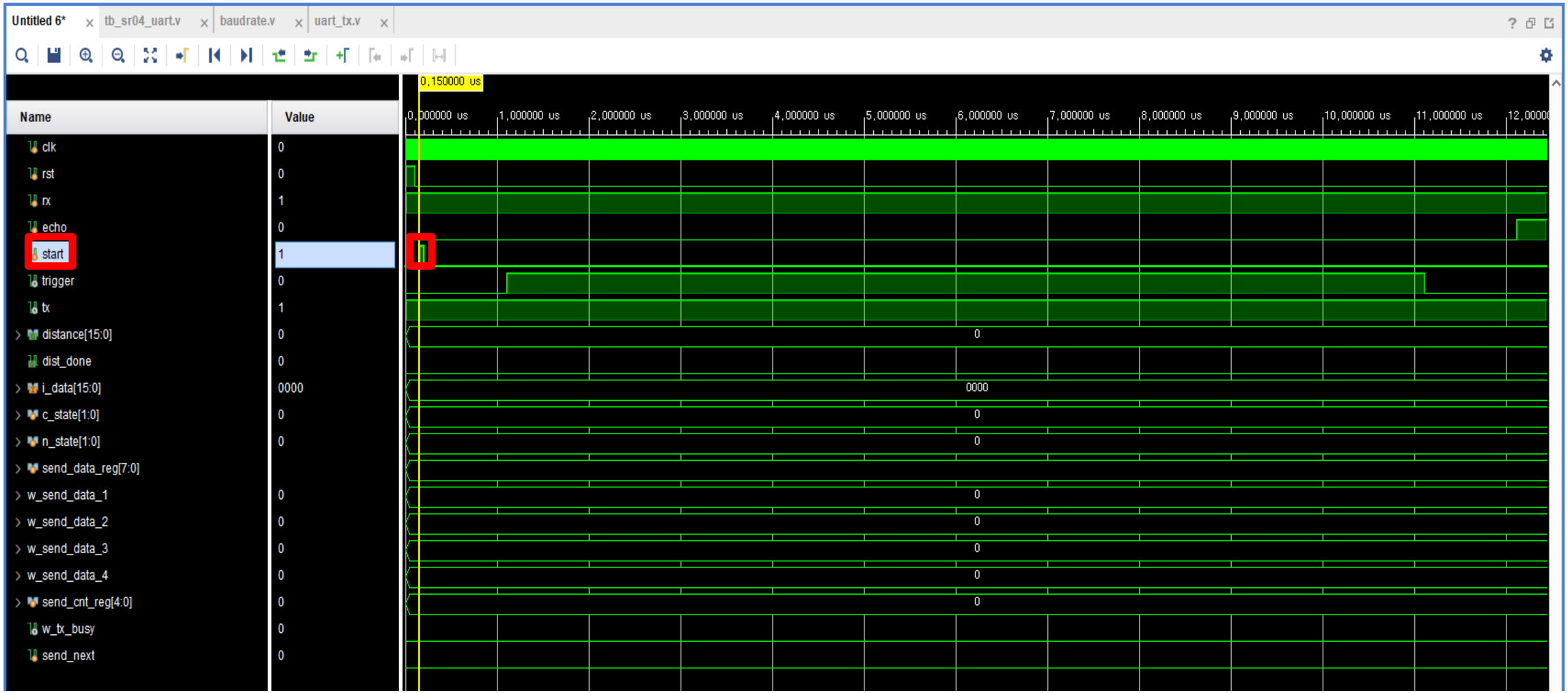
AS-IS

```
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];
```

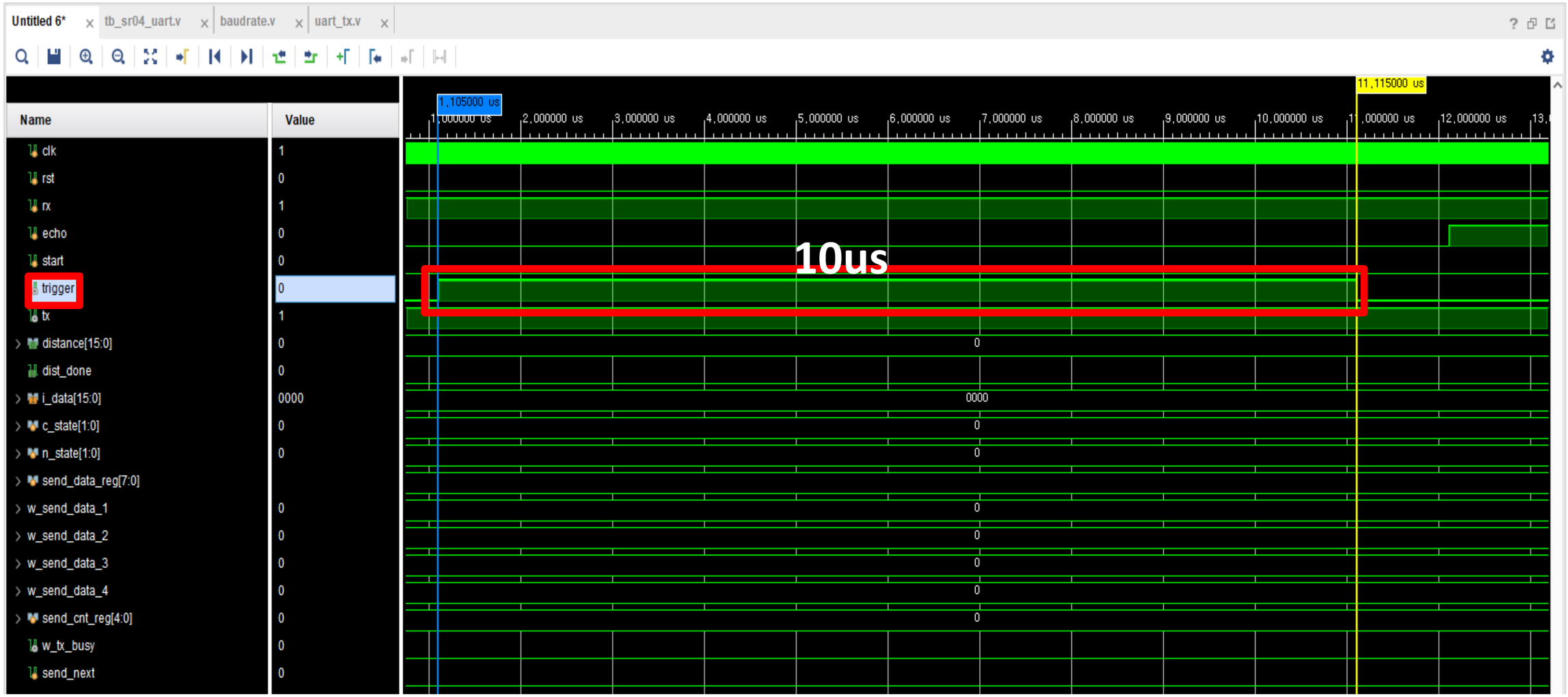
TO-BE

```
case (send_cnt_reg)
    0:  send_data_next = "D";
    1:  send_data_next = "i";
    2:  send_data_next = "s";
    3:  send_data_next = "t";
    4:  send_data_next = "a";
    5:  send_data_next = "n";
    6:  send_data_next = "c";
    7:  send_data_next = "e";
    8:  send_data_next = ":";
    9:  send_data_next = w_send_data[31:24];
    10: send_data_next = w_send_data[23:16];
    11: send_data_next = w_send_data[15:8];
    12: send_data_next = w_send_data[7:0];
    13: send_data_next = "c";
    14: send_data_next = "m";
    15: send_data_next = "\n";
    16: send_data_next = "\r";
```

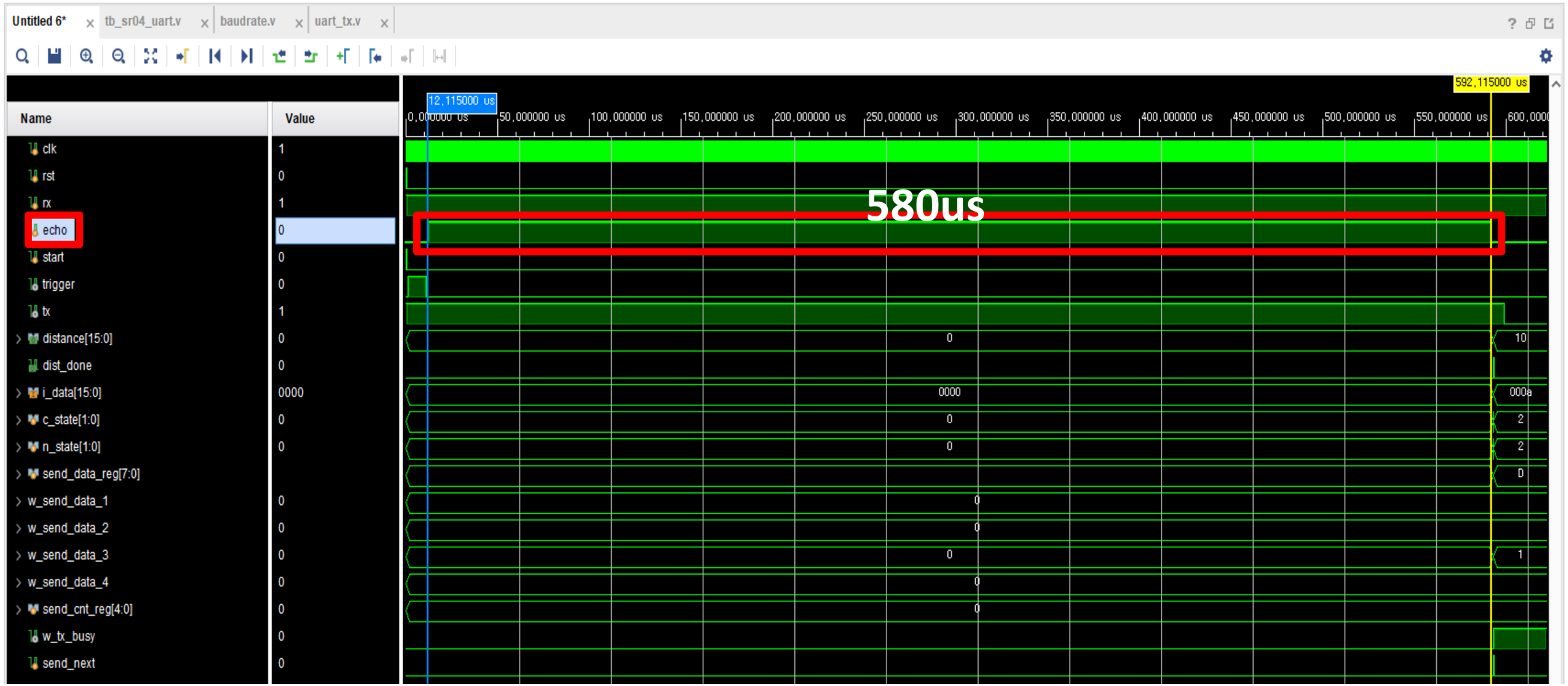
Simulation Verification - start



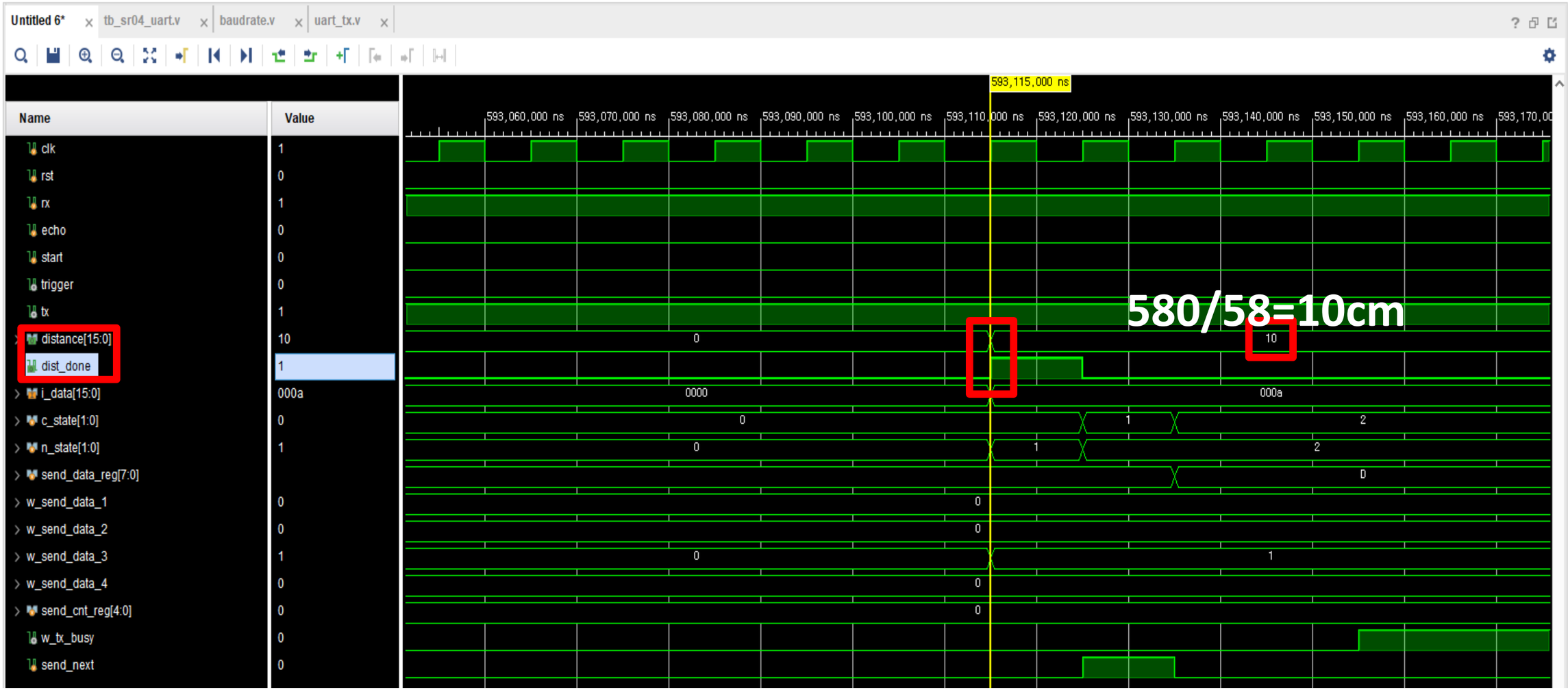
Simulation Verification - trigger



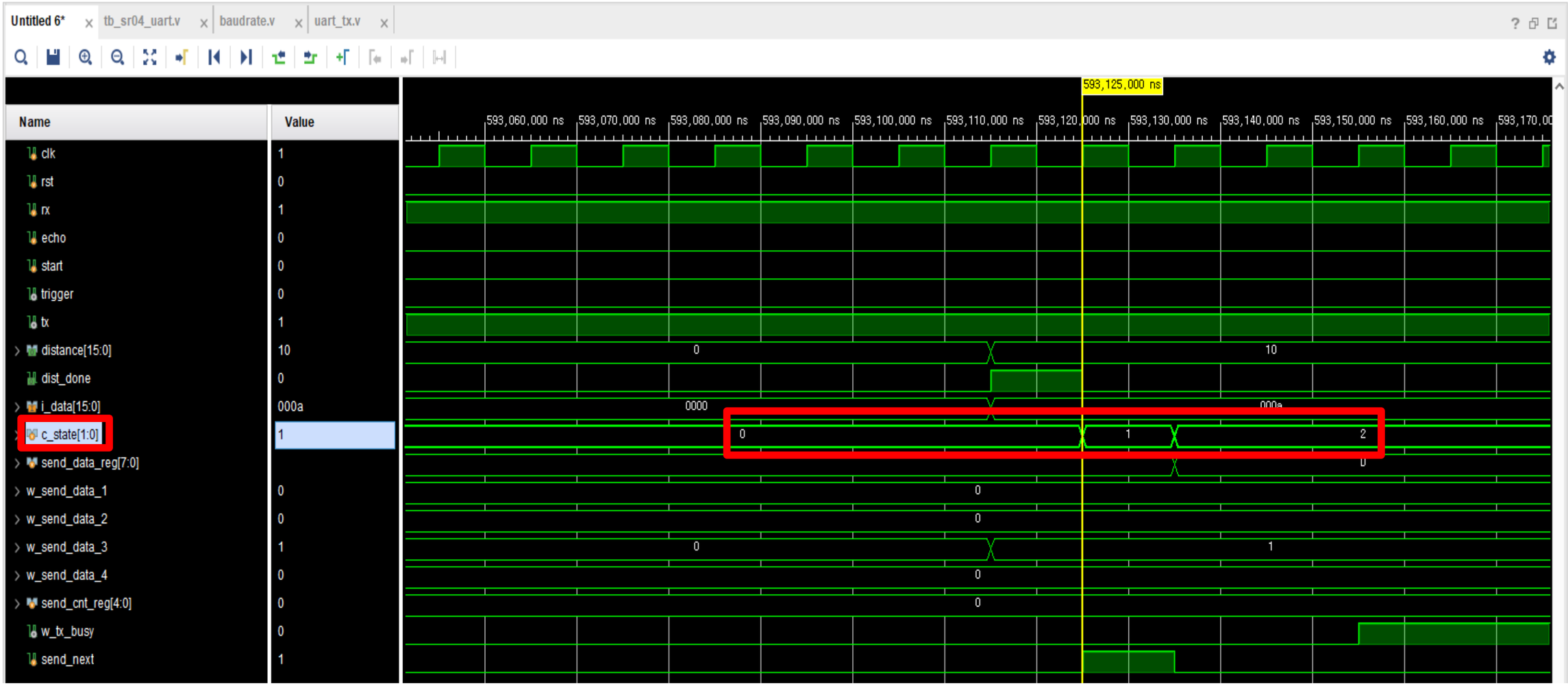
Simulation Verification - echo



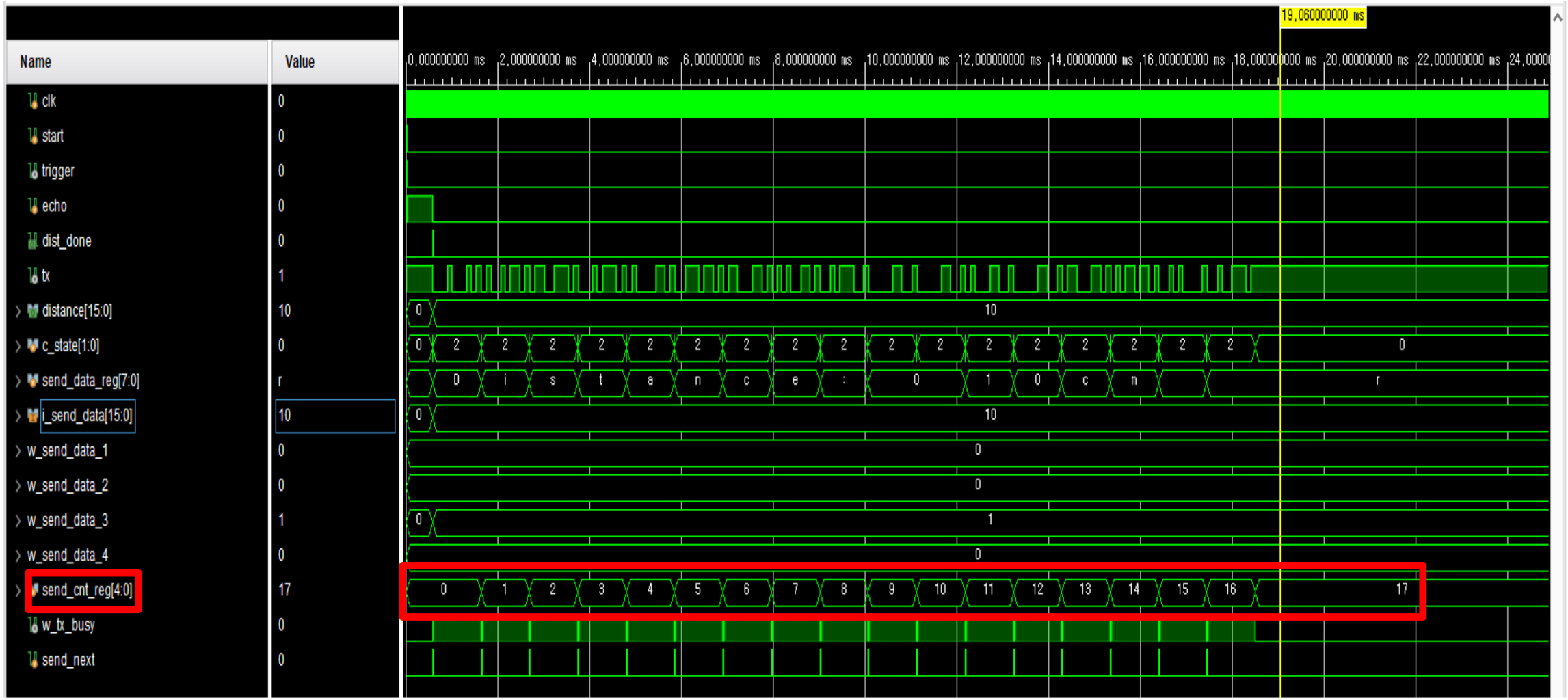
Simulation Verification - distance & dist_done



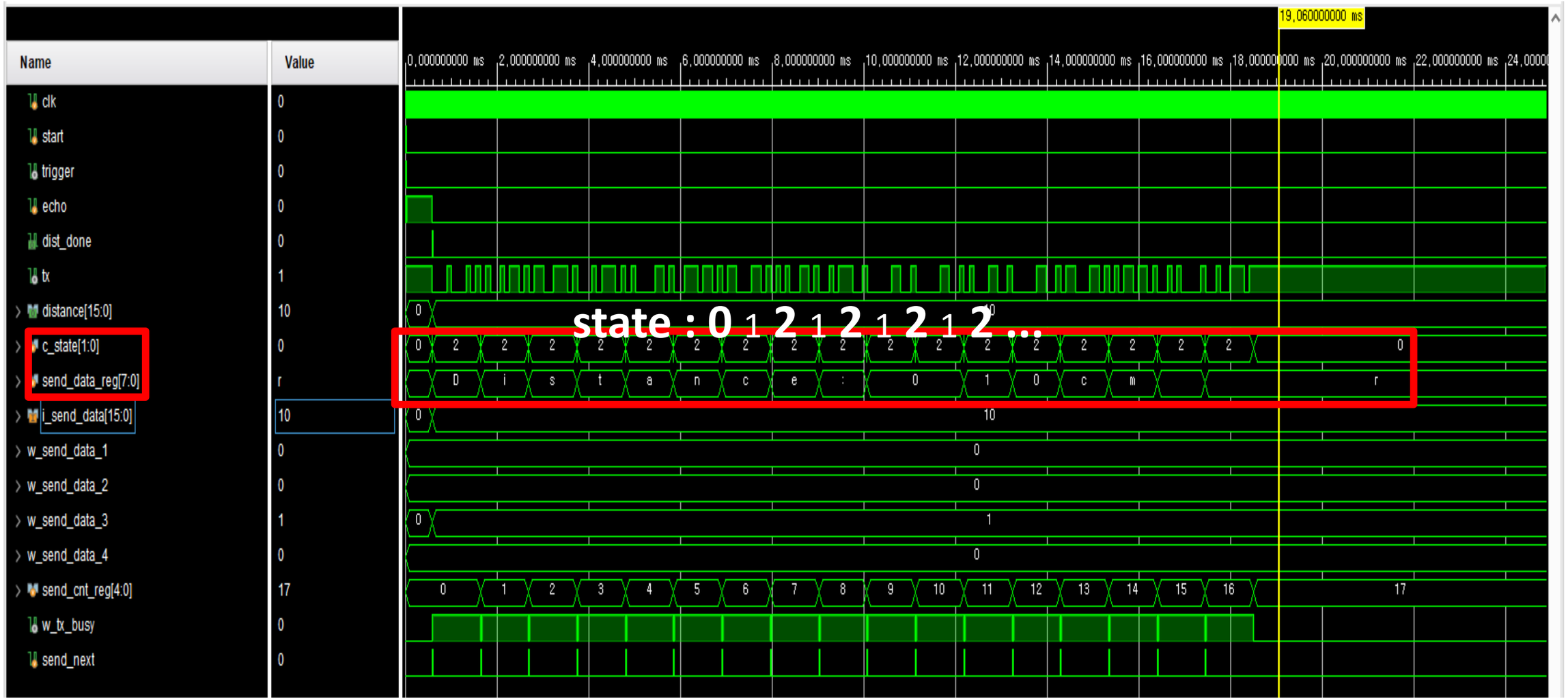
Simulation Verification - state



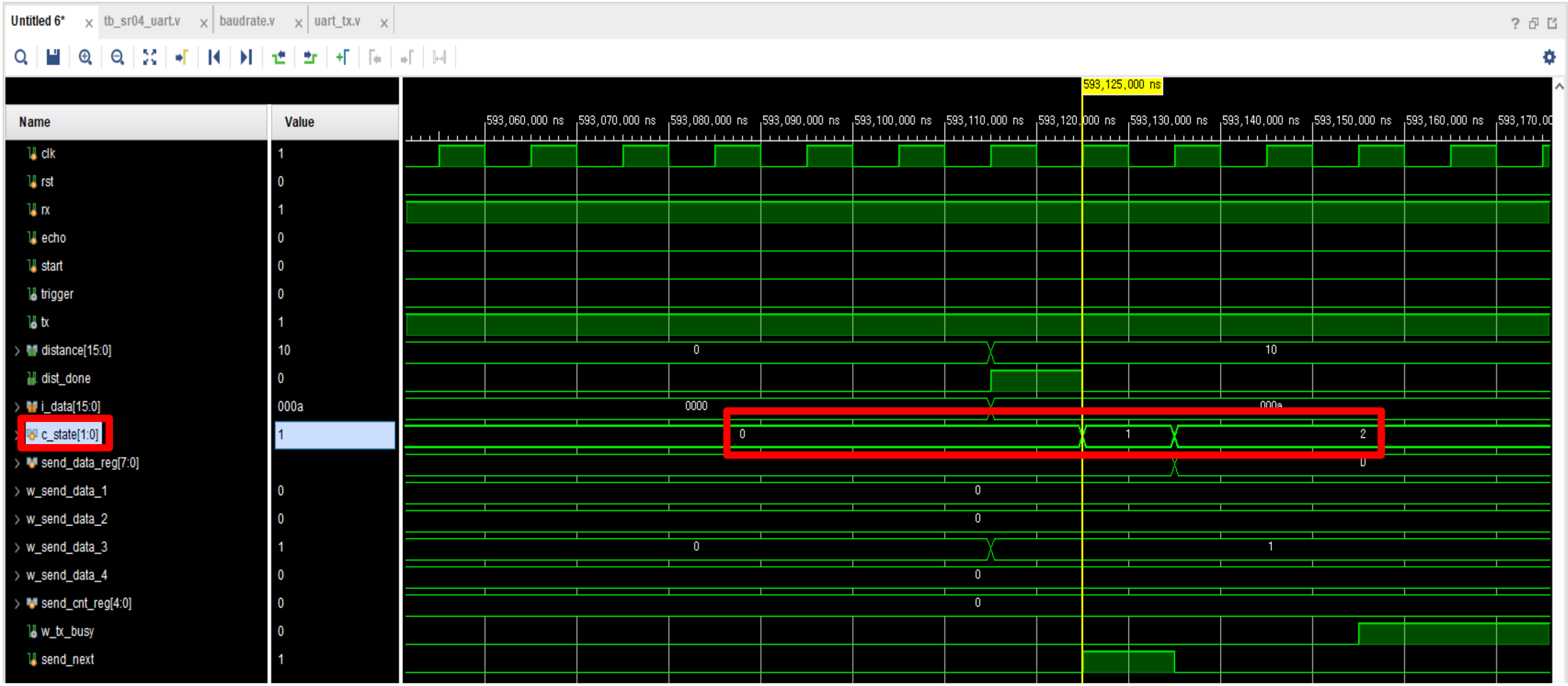
Simulation Verification - send_cnt_reg



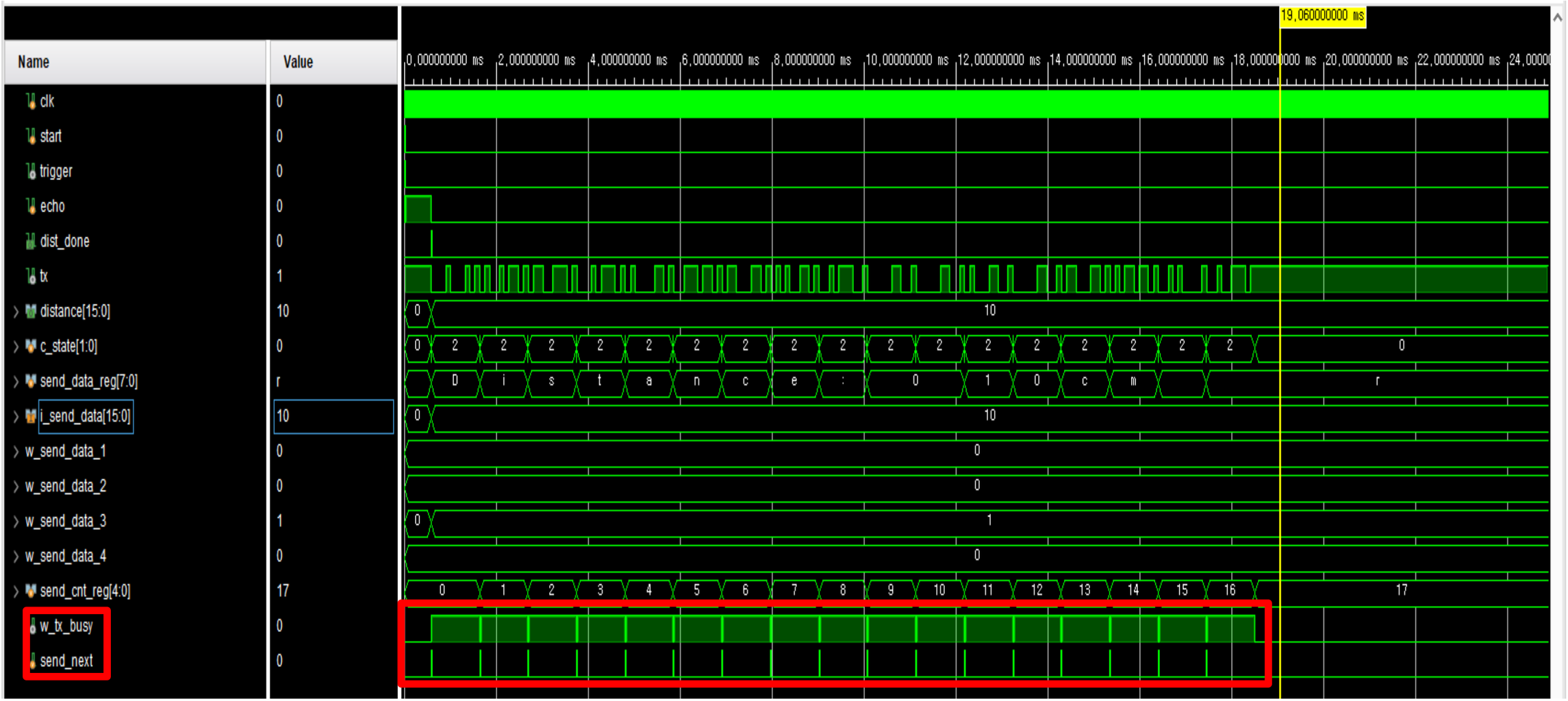
Simulation Verification - state & send_data_reg



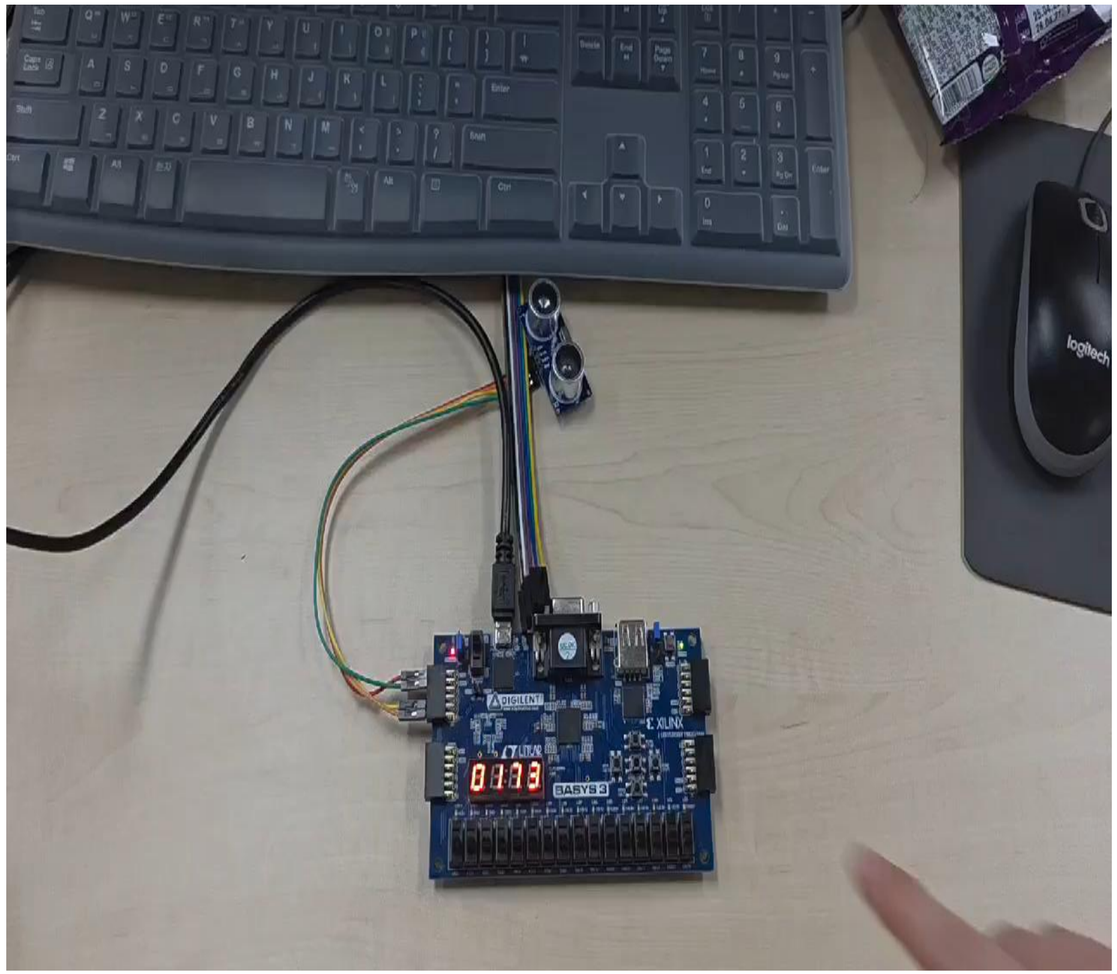
Simulation Verification - state

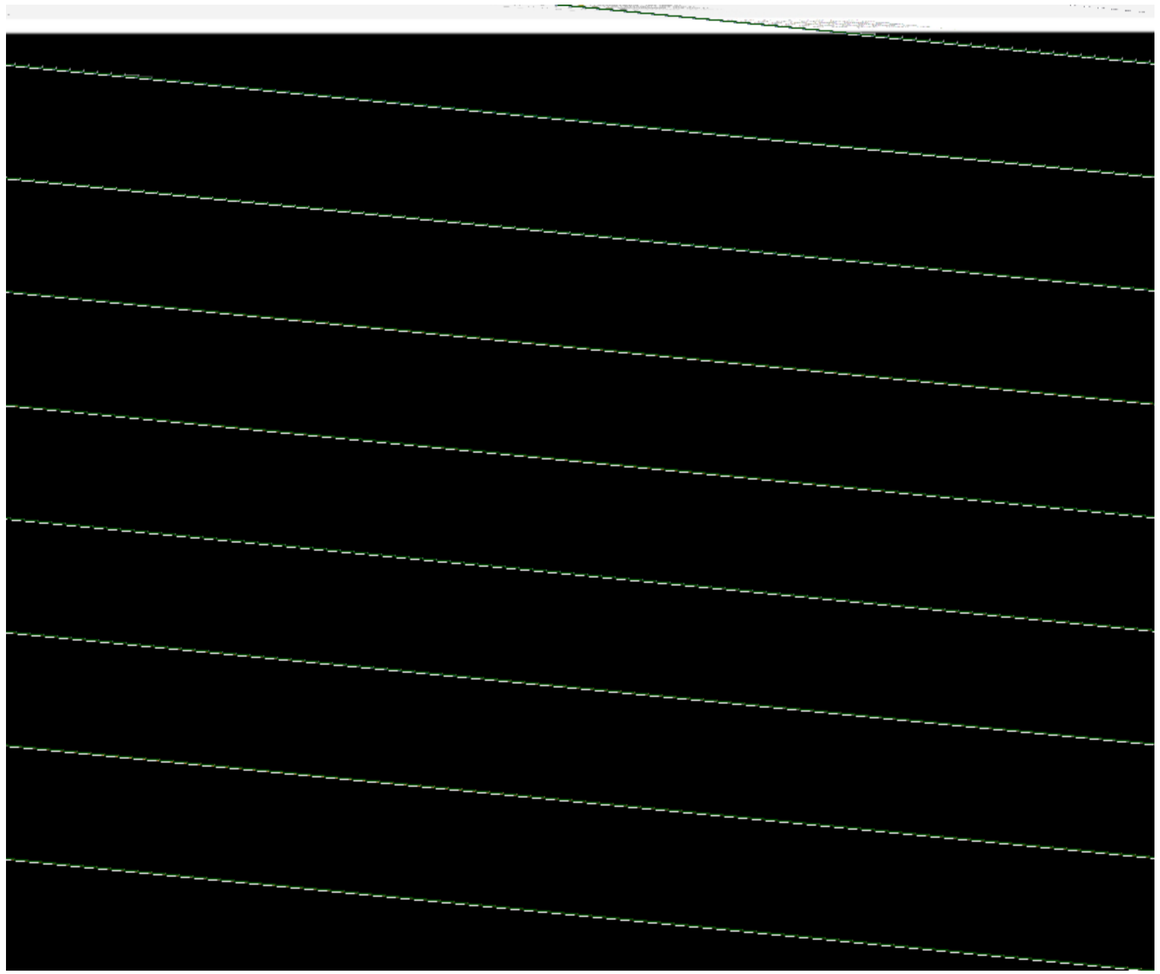


Simulation Verification - w_tx_busy & send_next



Simulation Verification - 시연 영상





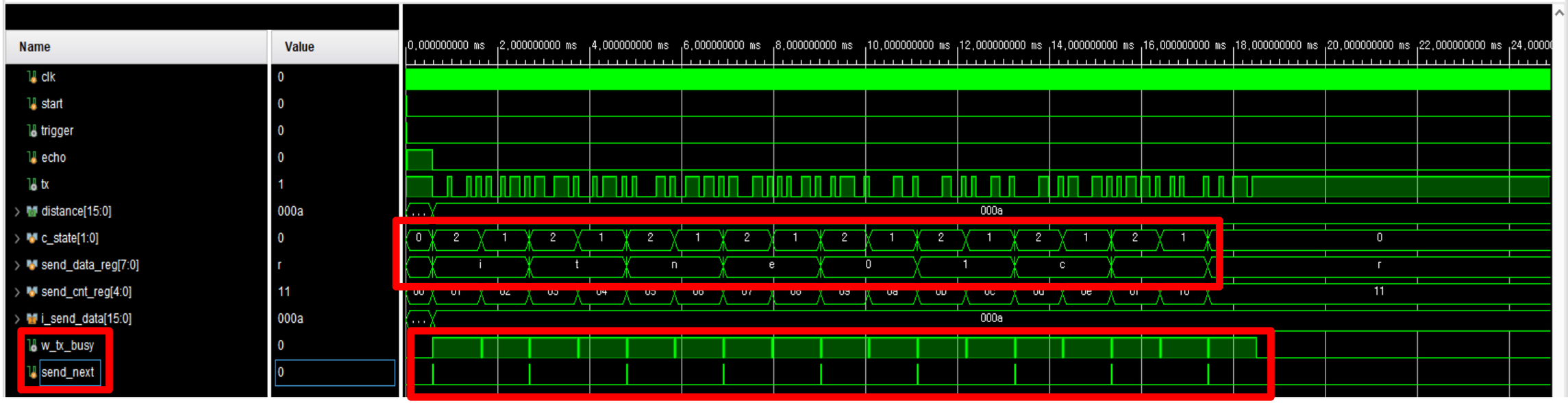
Trouble Shooting - Before

~ w_tx_full

```
WAIT: begin
    if(~w_tx_busy) begin
        send_cnt_next = send_cnt_reg + 1;
        if (send_cnt_reg == 16) n_state = IDLE;
        else n_state = LOAD;
    end
end
```

2개의 state

IDLE / START



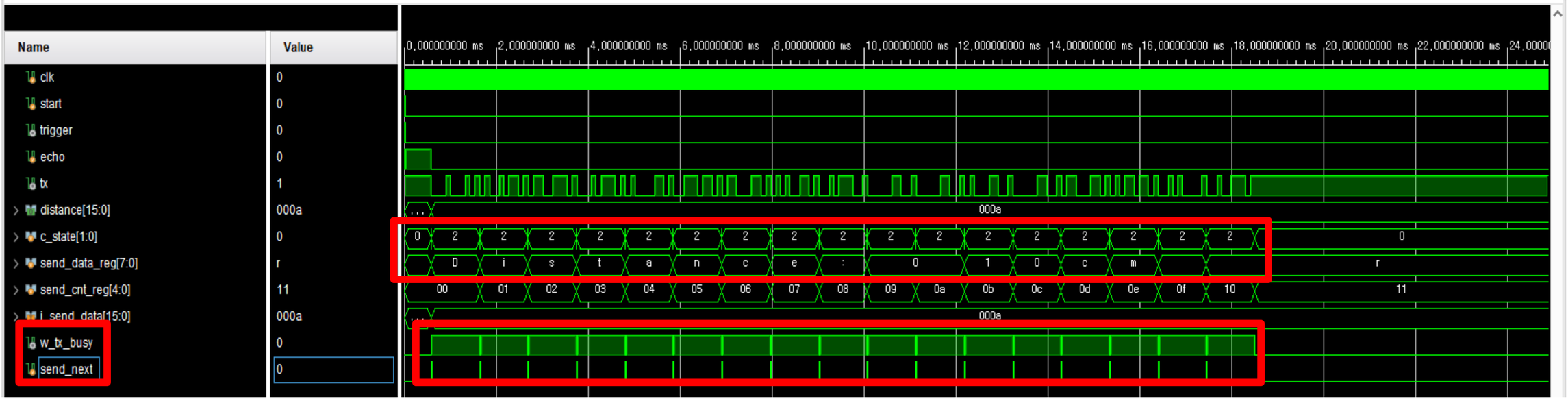
Trouble Shooting - After

tx_done

```
WAIT: begin
    if(tx_done) begin
        send_cnt_next = send_cnt_reg + 1;
        if (send_cnt_reg == 16) n_state = IDLE;
        else n_state = LOAD;
    end
end
```

3개의 state

IDLE / LOAD / WAIT



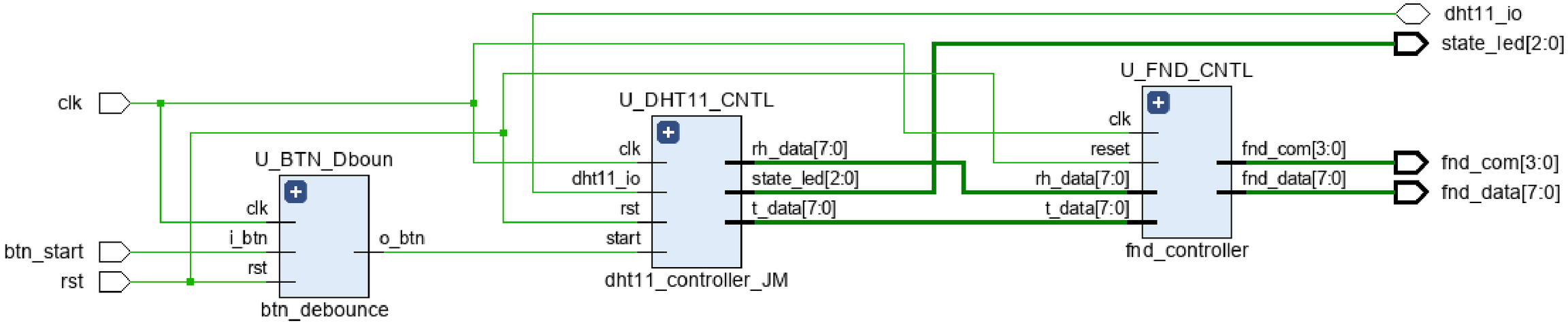
DHT11 Sensor

CHAPTER 05

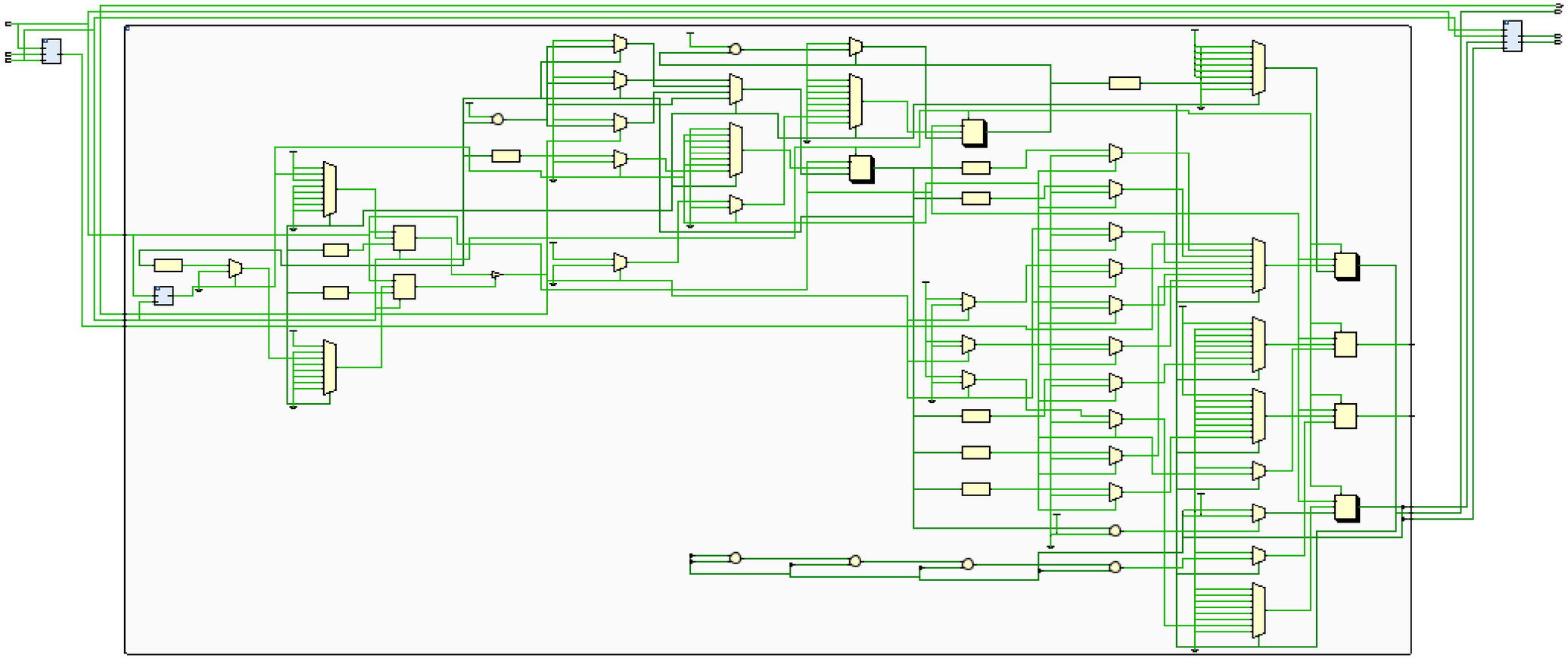
이현수

온,습도를 측정하는 DHT11 센서와 FIFO 및 Uart unit을 이용하여 측정 데이터를 받는 동작을 구현했습니다.

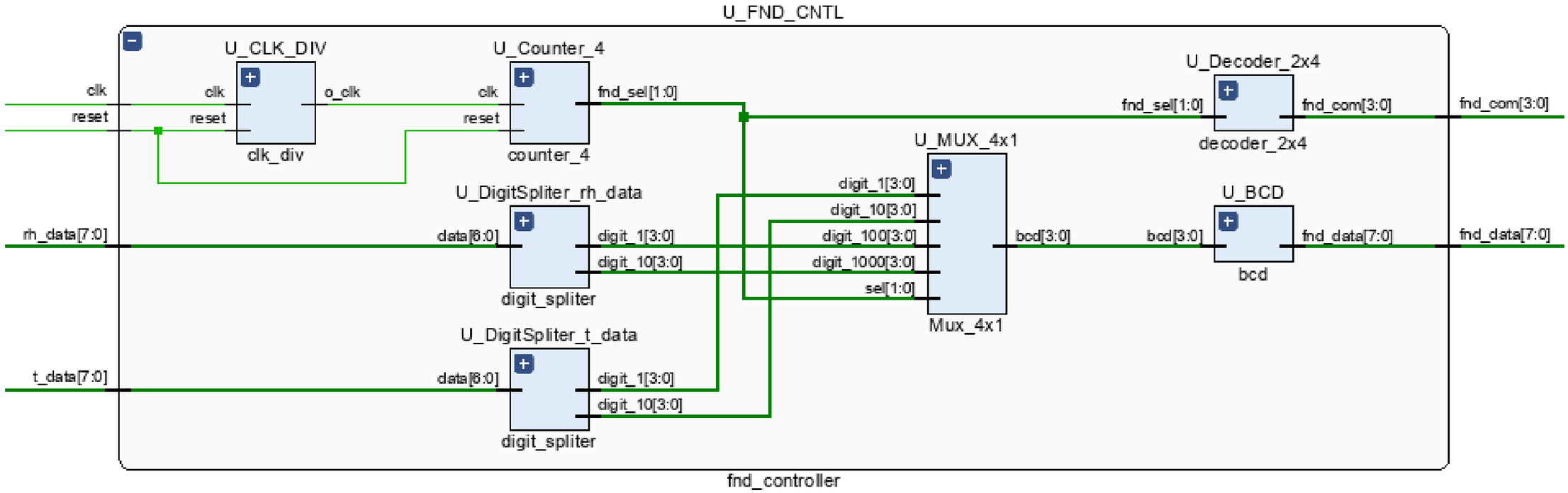
Schematic (RTL)



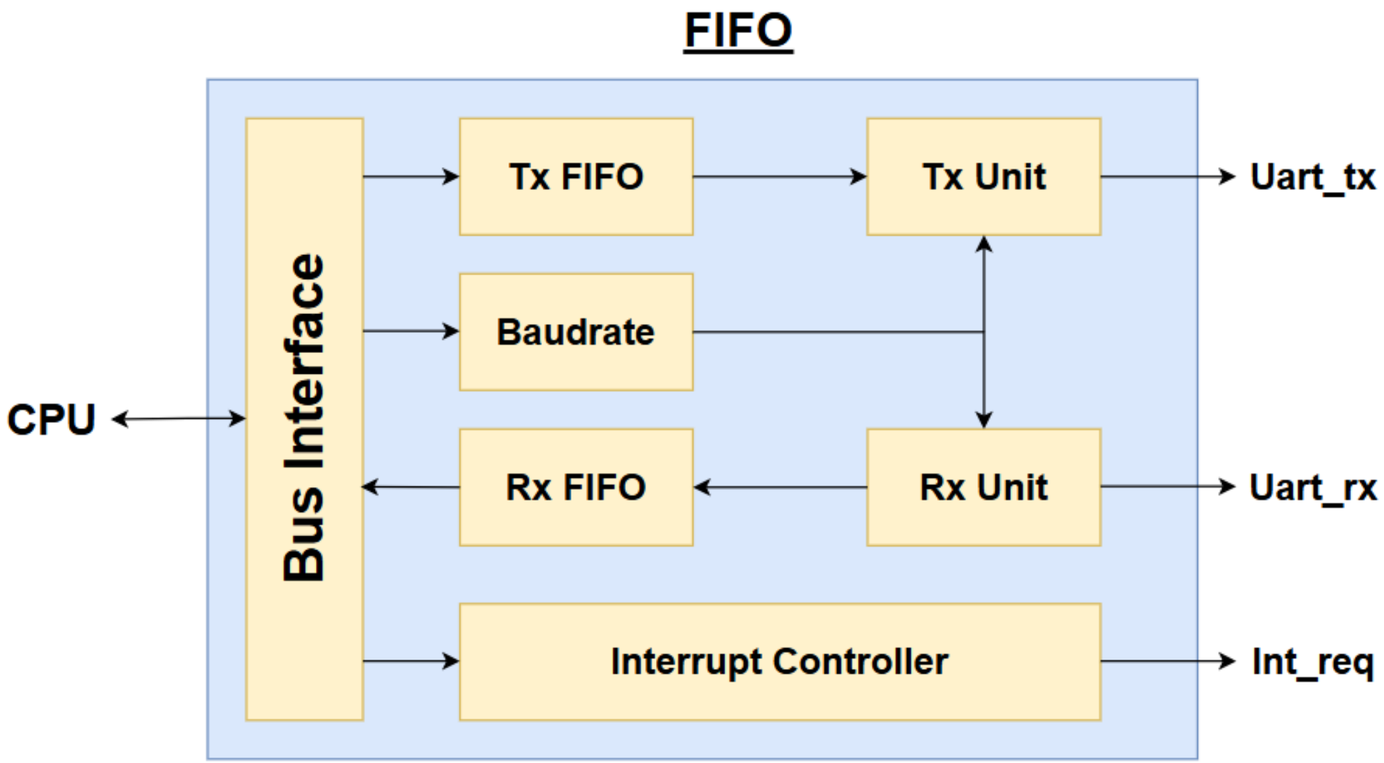
Schematic of DHT_Cntrl



Schematic of Fnd_Cntrl



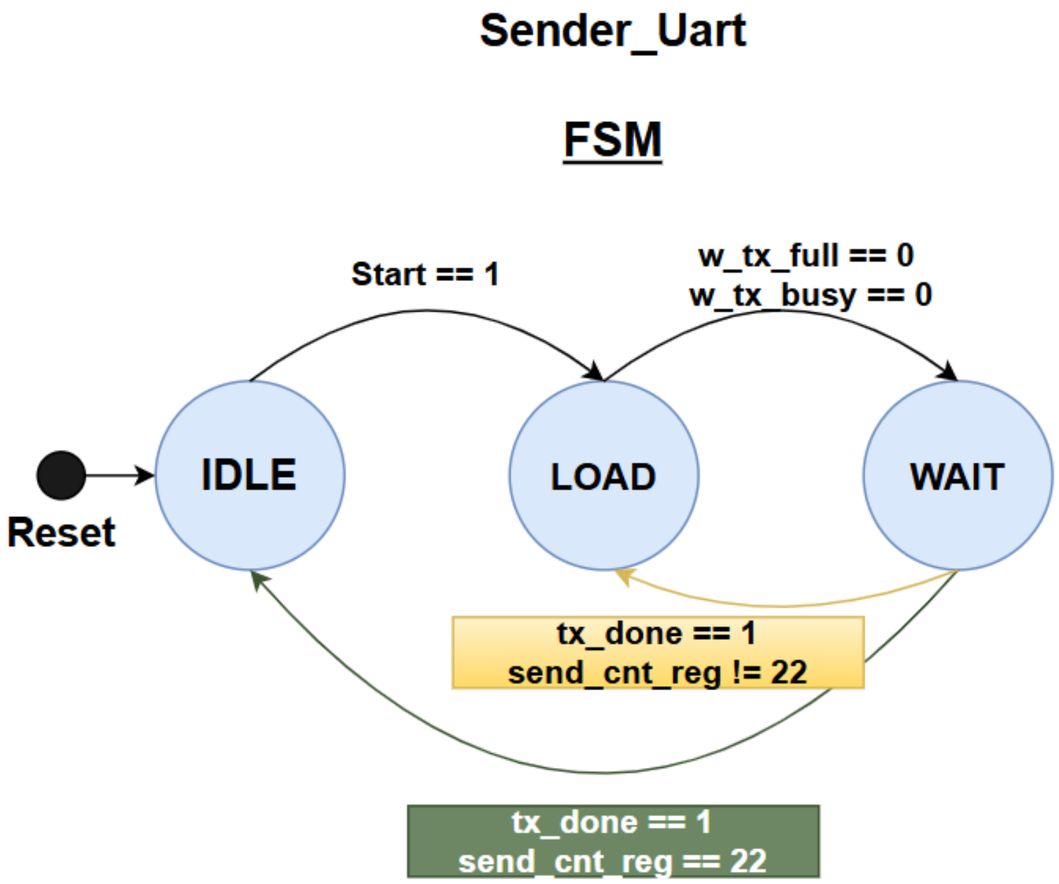
FIFO(First in First Out)




```
FIFO U_FIFO_TX (

    .clk       (clk),
    .rst       (rst),
    .push      (tx_push),              //
    .pop       (~w_tx_busy),
    .push_data(tx_push_data),
    .full      (tx_full),              //
    .empty     (w_tx_start),
    .pop_data (w_tx_pop_data)

);

FIFO U_FIFO_RX (

    .clk(clk),
    .rst(rst),
    .push(w_rx_done),
    .pop(rx_pop),  // rx fifo
    .push_data(w_rx_data),
    .full(),
    .empty(rx_empty),  // rx fifo empty
    .pop_data(rx_pop_data)  // pop data

);
```

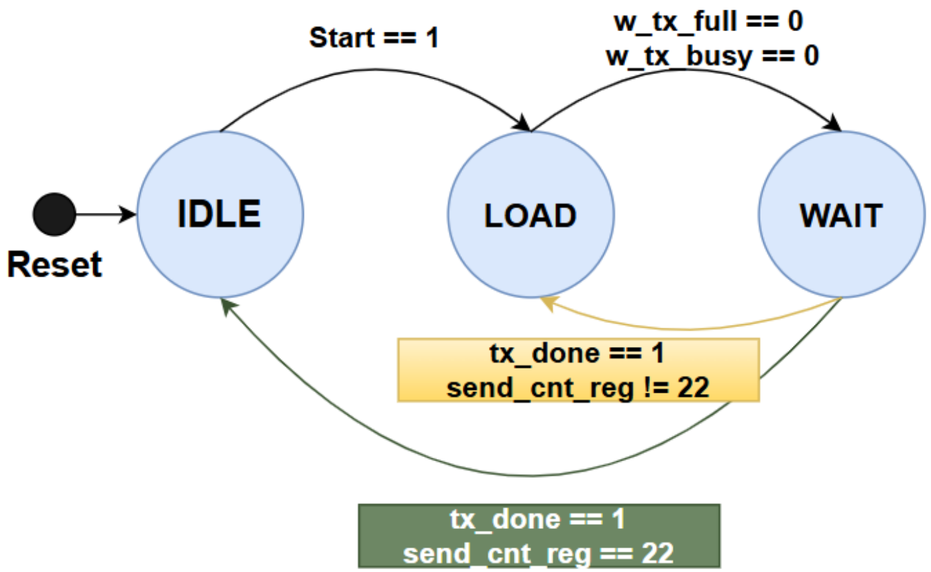
```
baudrate U_BR (
    .clk(clk),
    .rst(rst),
    .baud_tick(w_bd_tick)
);

Uart_Rx U_UART_RX (
    .clk(clk),
    .rst(rst),
    .b_tick(w_bd_tick),
    .rx(rx),
    .o_rx_done(w_rx_done),
    .o_dout(w_rx_data)
);

Uart_Tx U_UART_TX (
    .clk(clk),
    .rst(rst),
    .baud_tick(w_bd_tick),
    .start(~w_tx_start),
    .din(w_tx_pop_data),
    .o_tx_done(tx_done),
    .o_tx_busy(w_tx_busy),
    .o_tx(tx)
);

endmodule
```

FIFO(First in First Out)




```
        LOAD: begin
            if (~w_tx_full && ~w_tx_busy) begin
                case (send_cnt_reg)
                    0:  send_data_next = "T";
                    1:  send_data_next = "e";
                    2:  send_data_next = "m";
                    3:  send_data_next = "p";
                    4:  send_data_next = ":";
                    5:  send_data_next = " ";
                    6:  send_data_next = temp_ascii[0];
                    7:  send_data_next = temp_ascii[1];
                    8:  send_data_next = "'";
                    9:  send_data_next = "C";
                    10: send_data_next = ",";
                    11: send_data_next = " ";
                    12: send_data_next = "H";
                    13: send_data_next = "u";
                    14: send_data_next = "m";
                    15: send_data_next = "i";
                    16: send_data_next = "d";
                    17: send_data_next = ":";
                    18: send_data_next = " ";
                    19: send_data_next = humi_ascii[0];
                    20: send_data_next = humi_ascii[1];
                    21: send_data_next = "%";
                    22: send_data_next = "\n";
                    default: begin
                        send_data_next = 0;
                        next_state = IDLE;
                    end
                endcase
                send_next  = 1;
                next_state = WAIT;
            end
        end
```

Sender_Uart




```
datatoascii2 U_temp (
    .i_data(i_send_data[7:0]),
    .o1(temp_ascii[0]),
    .o2(temp_ascii[1])
);

datatoascii2 U_humi (
    .i_data(i_send_data[15:8]),
    .o1(humi_ascii[0]),
    .o2(humi_ascii[1])
);

Uart_controller U_UART_CNTL (
    .clk(clk),
    .rst(rst),
    .rx(rx),
    .rx_pop(),
    .rx_pop_data(),
    .rx_empty(),
    .rx_done(),
    .tx_push_data(send_data_reg),
    .tx_push(send_reg),
    .tx_full(w_tx_full),
    .tx_done(tx_done),
    .tx_busy(w_tx_busy),
    .tx(tx)
);
```

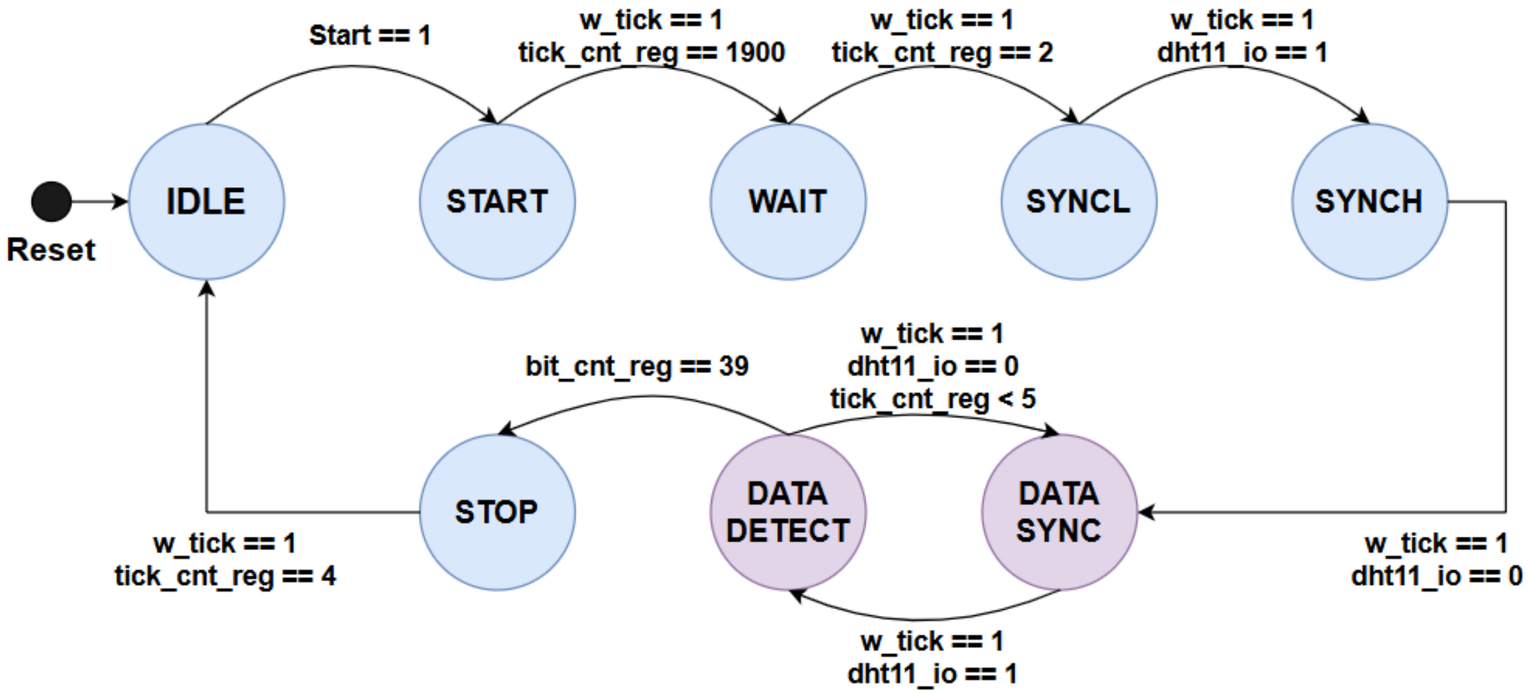
```
module datatoascii2 (
    input  [7:0] i_data,
    output [7:0] o1,
    output [7:0] o2
);
    assign o1 = (i_data / 10) + 8'd48;
    assign o2 = (i_data % 10) + 8'd48;
endmodule
```

DHT11_Controller

DHT11_Controller

FSM

```
parameter IDLE=0, START=1, WAIT=2, SYNCL=3, SYNCH=4, DATA_SYNC=5,
          DATA_DETECT=6, STOP=7;
```



DHT11_Controller

```
            DATA_DETECT: begin
                dht11_next = 1'b1;
                if (w_tick) begin
                    if (!dht11_io) begin
                        if (tick_cnt_reg < 5) begin  //data 입력 0
                            data_next = {data_reg[38:0], 1'b0};
                        end else begin  //data 입력 1
                            data_next = {data_reg[38:0], 1'b1};
                        end

                        if (bit_cnt_reg== 39) begin  //state 이동
                            bit_cnt_next = 0;
                            n_state = STOP;
                            tick_cnt_next = 0;
                        end else begin
                            bit_cnt_next = bit_cnt_reg + 1;
                            n_state = DATA_SYNC;
                            tick_cnt_next = 0;
                        end
                    end else begin
                        tick_cnt_next = tick_cnt_reg + 1;
                    end
                end

            end
```

```
assign rh_data = data_reg[39:32];
assign t_data = data_reg[23:16];
```

```
            STOP: begin
                dht11_next = 1'b0;
                if (w_tick) begin
                    if (tick_cnt_reg == 4) begin

                        done_next = 1'b1;
                        valid_next = ((data_reg[39:32] + data_reg[31:24] +
                            data_reg[23:16] + data_reg[15:8]) == data_reg[7:0]);
                        n_state = IDLE;
                    end else begin
                        tick_cnt_next = tick_cnt_reg + 1;
                    end
                end

        endcase
    end

endmodule
```

AS-IS TO-BE

AS-IS

16비트 데이터 추출 방식

```
wire [31:0] w_send_data;
```

온도, 습도 데이터만 출력

```
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
```

TO-BE

온,습도 데이터 각 8비트로 분할

```
wire [7:0] temp_ascii[1:0];
wire [7:0] humi_ascii[1:0];
```

글자 출력을 추가하여 가시성 확보

```
0:  send_data_next = "T";
1:  send_data_next = "e";
2:  send_data_next = "m";
3:  send_data_next = "p";
4:  send_data_next = ":";
5:  send_data_next = " ";
6:  send_data_next = temp_ascii[0];
7:  send_data_next = temp_ascii[1];
8:  send_data_next = "'";
9:  send_data_next = "C";
```

Trouble-shooting

~w_tx_busy -> tx_done

- 시뮬레이션을 확인하는 도중,
  홀수 번째와 짝수 번째 데이터들의 load되는 시간이 크게 다른 것을 확인.
- WAIT 상태에서 busy 조건을 확인하면서 타이밍이 망가짐. tx_done 으로 해결

```
WAIT: begin
    if (tx_done) begin
        send_cnt_next = send_cnt_reg + 1;
        next_state = (send_cnt_reg == 22) ? IDLE : LOAD;
    end
end
```

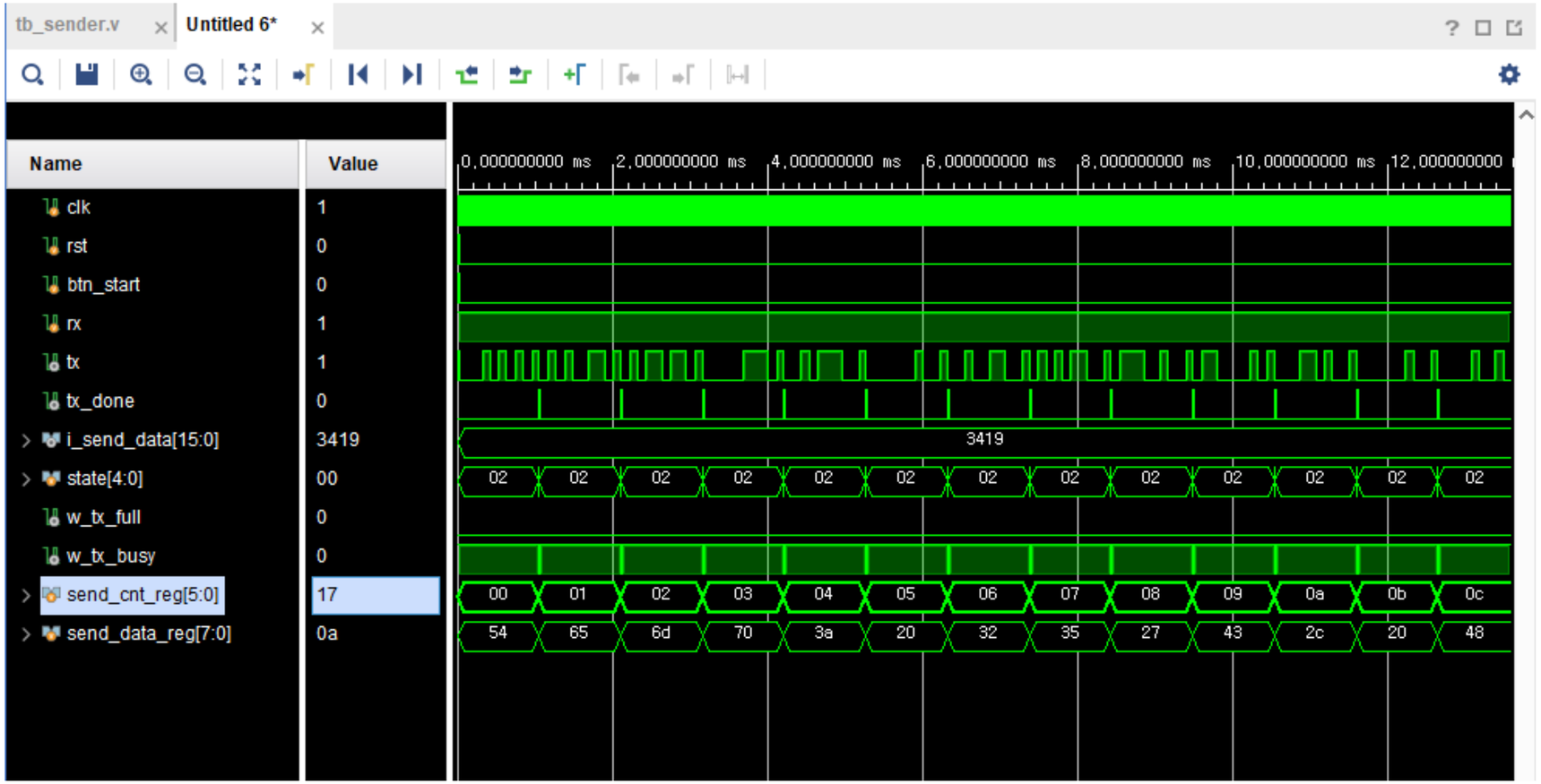
Check_Sum

- 초반 설계 당시, Check_Sum 상태에서 Stop 상태로 넘어가지 않는 문제 발생
- 원인 찾기 실패 후, Check_Sum을 STOP 상태에서 동작하도록 수정.
- 이후 시뮬레이션에서 state가 DATA_DETECT -> STOP -> IDLE로 넘어가는 것과 valid가 High가 되는 것을 확인.

```
STOP: begin
    dht11_next = 1'b0;
    if (w_tick) begin
        if (tick_cnt_reg == 4) begin

            done_next = 1'b1;
            valid_next = ((data_reg[39:32] + data_reg[31:24] +
               data_reg[23:16] + data_reg[15:8]) == data_reg[7:0]);
            n_state = IDLE;
        end else begin
            tick_cnt_next = tick_cnt_reg + 1;
```

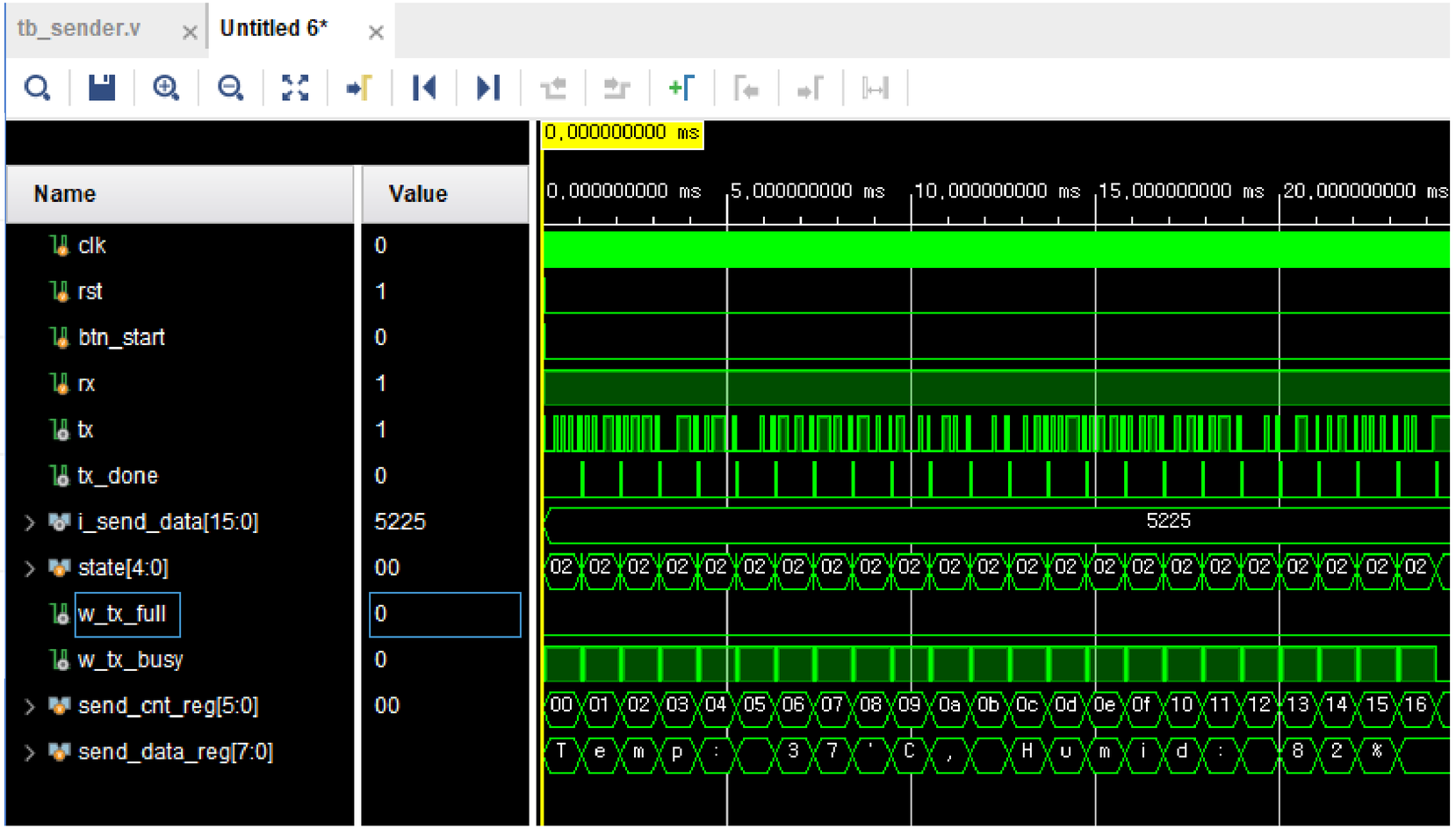
Simulation

```
`timescale 1ns / 1ps

module tb_sender ();

    reg clk = 0;
    reg rst = 1;
    reg btn_start = 0;
    reg rx = 1;
    wire tx, tx_done;

    // {humidity, temperature} = {52, 25}
    wire [15:0] i_send_data = {8'h52, 8'h25};

    // DUT 인스턴스
    sender_uart dut (
        .clk(clk),
        .rst(rst),
        .rx(rx),
        .i_send_data(i_send_data),
        .btn_start(btn_start),
        .tx(tx),
        .tx_done(tx_done)
    );

    // 클럭 생성 (100MHz)
    always #5 clk = ~clk;

    initial begin

        #0;
        clk = 0;
        rst = 1;
        btn_start = 0;

        // 초기화
        #20 rst = 0;

        #50;
        btn_start = 1;
        #10;
        btn_start = 0;

        #50000;
        $stop;
    end


endmodule
```

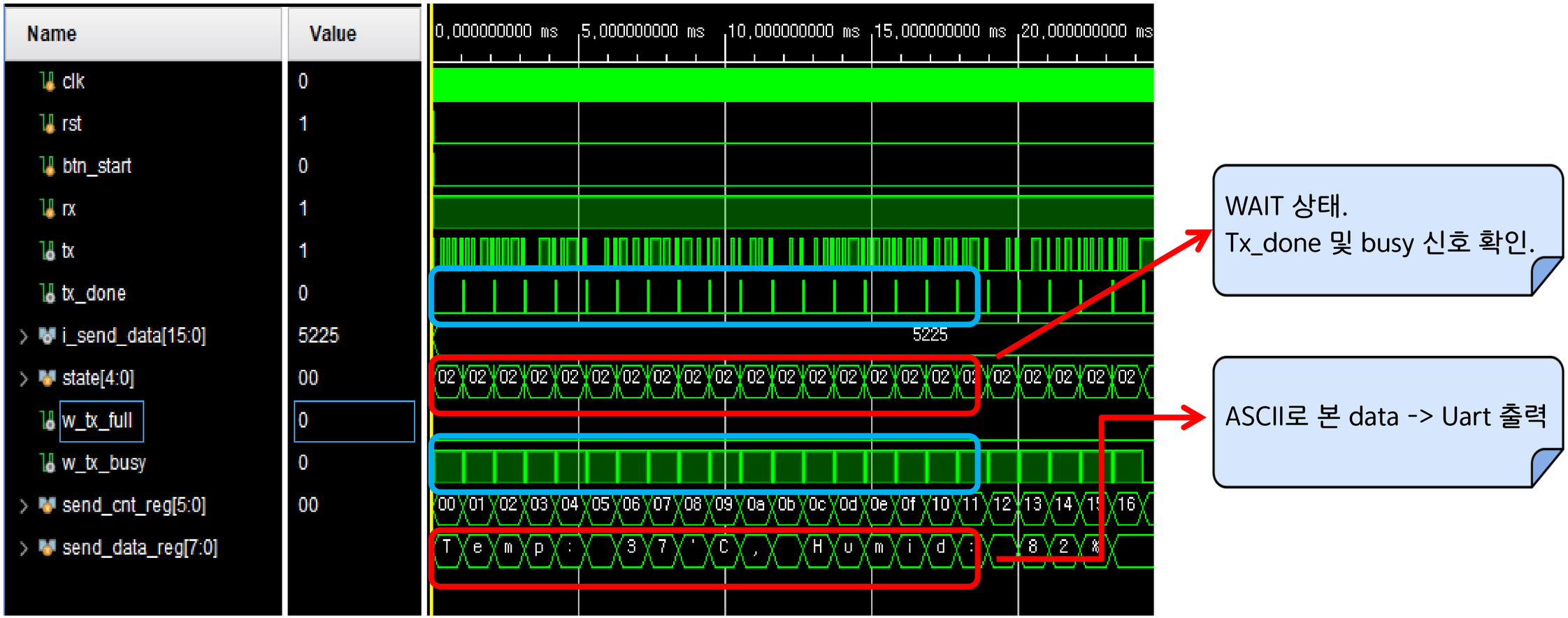


Simulation

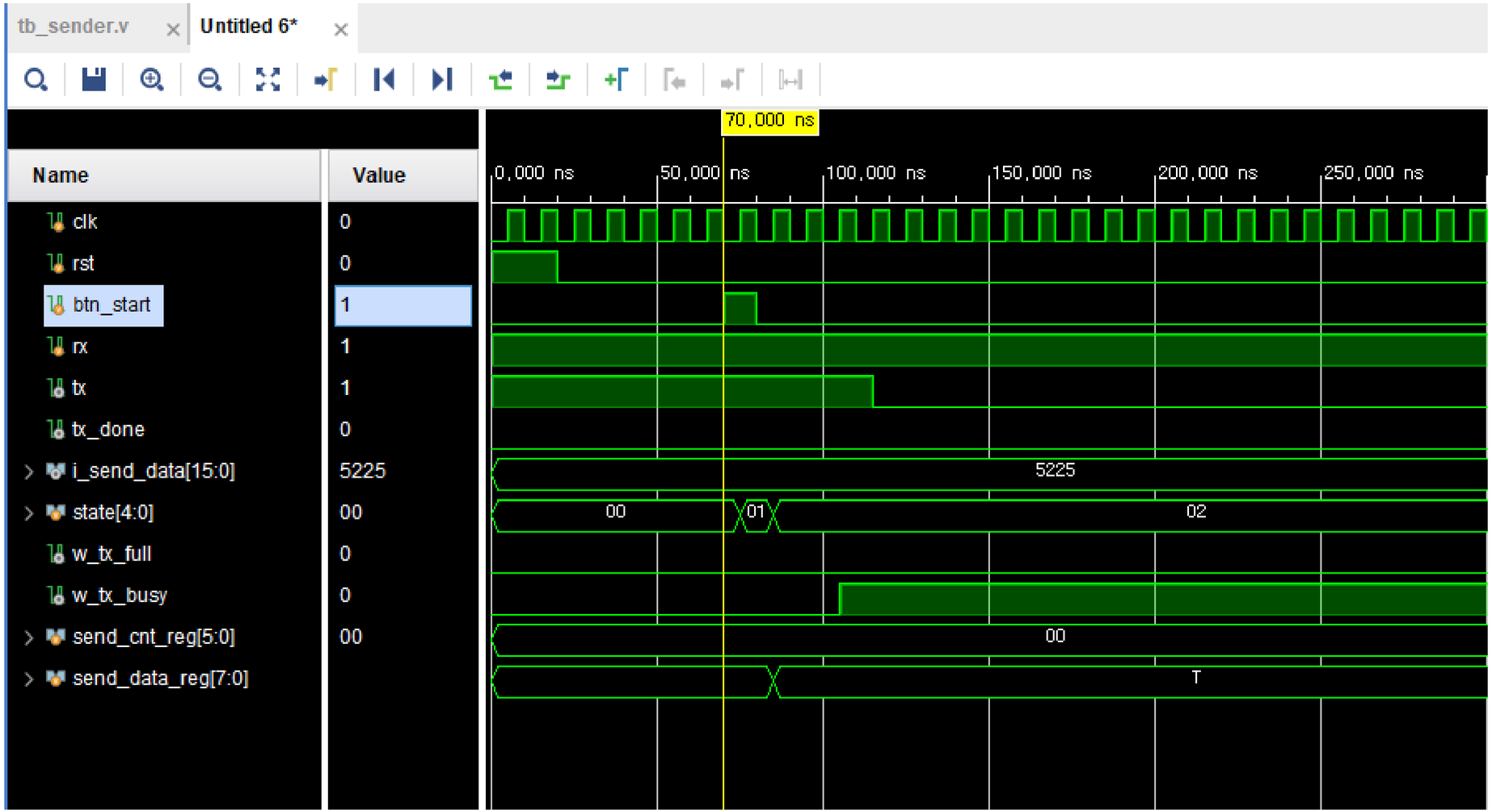
0x54	T
0x65	e
0x6D	m
0x70	p
0x3A	:



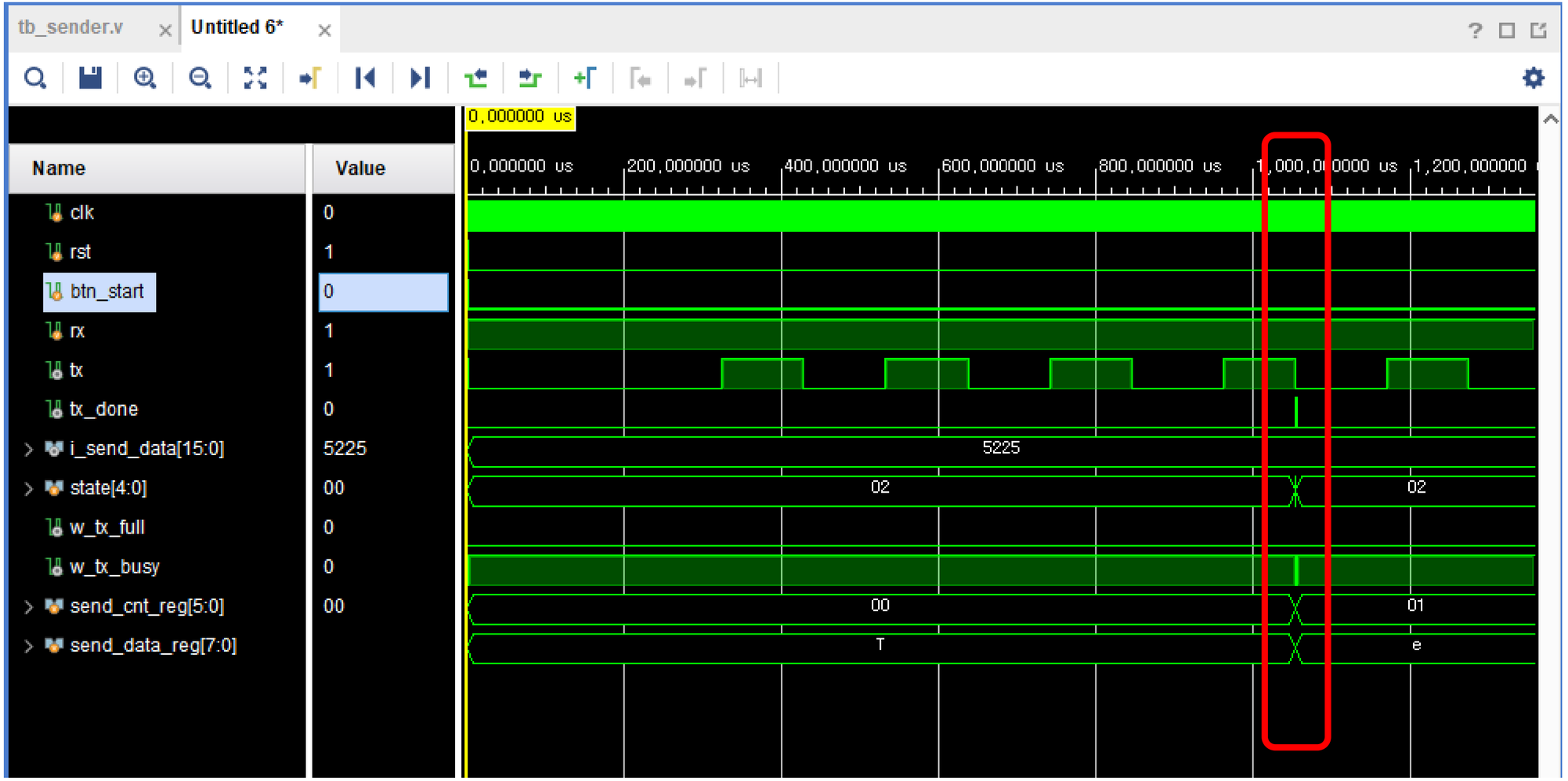
Simulation


Name
Value
clk
0
rst
1
btn_start
0
rx
1
tx
1
tx_done
0
i_send_data[15:0]
5225
state[4:0]
00
w_tx_full
0
w_tx_busy
0
send_cnt_reg[5:0]
00
send_data_reg[7:0]
0,000000000 ms
5,000000000 ms
10,000000000 ms
15,000000000 ms
20,000000000 ms
5225
WAIT 상태.
Tx_done 및 busy 신호 확인.
ASCII로 본 data -> Uart 출력

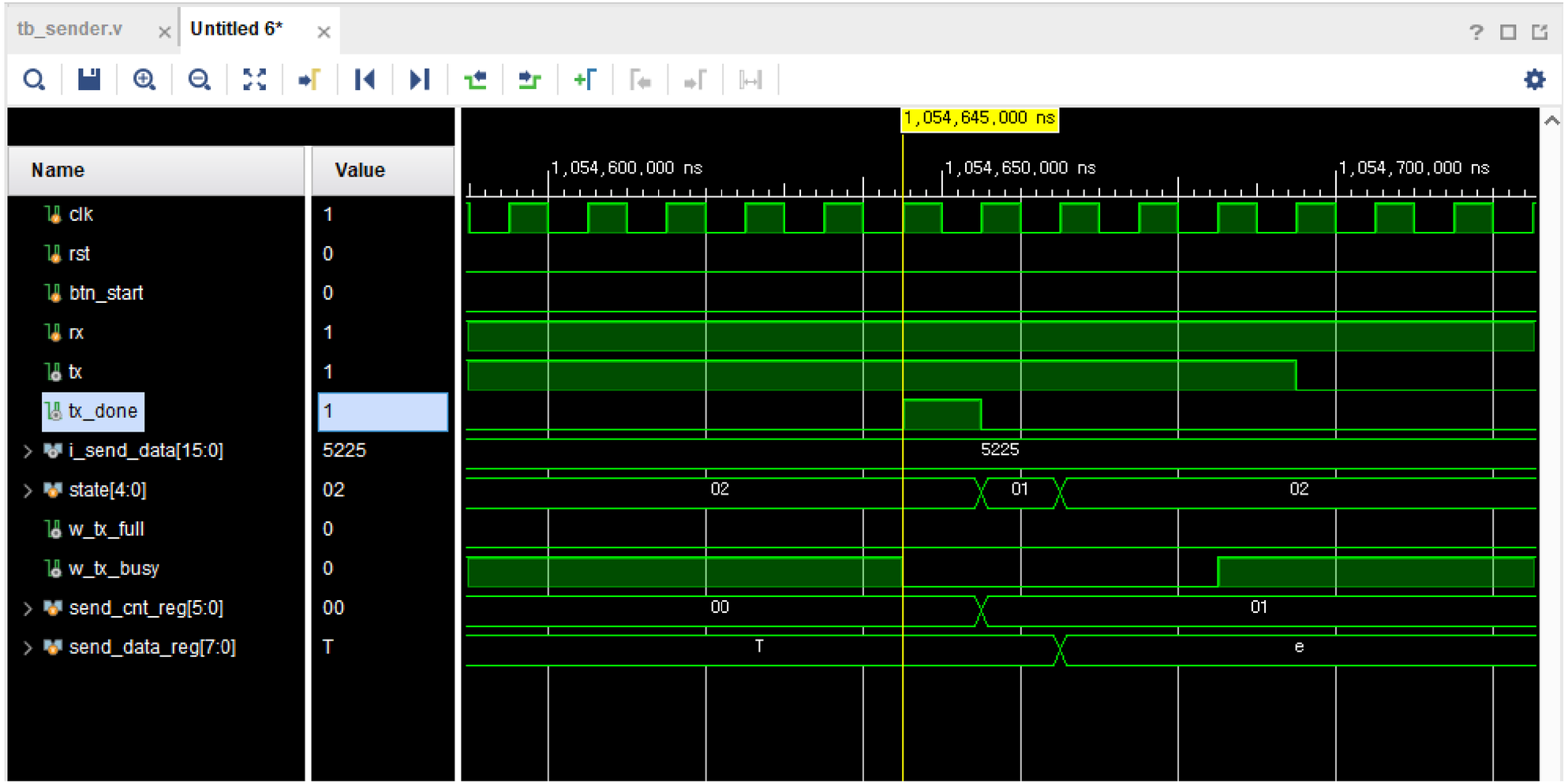
Simulation


tb_sender.v
Untitled 6*
Name
Value
clk
0
rst
0
btn_start
1
rx
1
tx
1
tx_done
0
i_send_data[15:0]
5225
state[4:0]
00
w_tx_full
0
w_tx_busy
0
send_cnt_reg[5:0]
00
send_data_reg[7:0]
70,000 ns
0,000 ns
50,000 ns
100,000 ns
150,000 ns
200,000 ns
250,000 ns
5225
00
01
02
00
T

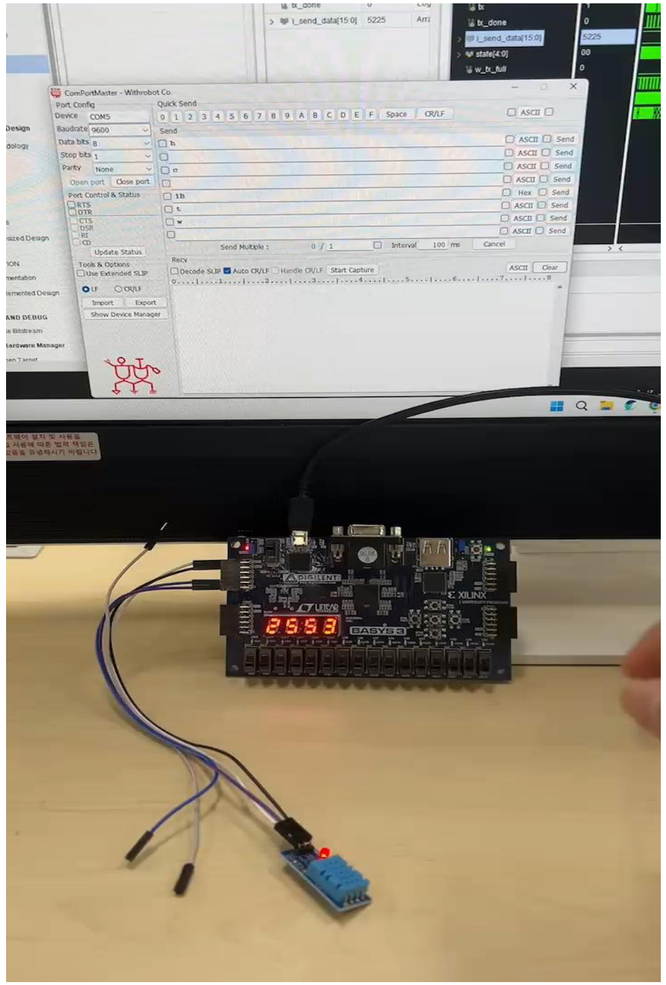
Simulation


tb_sender.v
Untitled 6*
Name
Value
clk
0
rst
1
btn_start
0
rx
1
tx
1
tx_done
0
i_send_data[15:0]
5225
state[4:0]
00
w_tx_full
0
w_tx_busy
0
send_cnt_reg[5:0]
00
send_data_reg[7:0]
0,000000 us
0,000000 us
200,000000 us
400,000000 us
600,000000 us
800,000000 us
1,200,000000
5225
02
02
00
01
T
e

Simulation


tb_sender.v
Untitled 6*
Name
Value
clk
1
rst
0
btn_start
0
rx
1
tx
1
tx_done
1
i_send_data[15:0]
5225
state[4:0]
02
w_tx_full
0
w_tx_busy
0
send_cnt_reg[5:0]
00
send_data_reg[7:0]
T
1,054,645,000 ns
1,054,600,000 ns
1,054,650,000 ns
1,054,700,000 ns
5225
02
01
02
00
01
T
e

동작 영상


ComPortMaster - Withrobot Co.
Port Config
Device COM5
Baudrate 9600
Data bits 8
Stop bits 1
Parity None
Open port
Close port
Port Control & Status
RTS
DTR
CTS
DSR
RI
CD
Update Status
Tools & Options
Use Extended SLIP
LF
CR/LF
Import
Export
Show Device Manager
Quick Send
Space
CR/LF
ASCII
Send
Send Multiple :
Interval
100
ms
Cancel
Recv
Decode SLIP
Auto CR/LF
Handle CR/LF
Start Capture
Clear
XILINX
BASYS 3

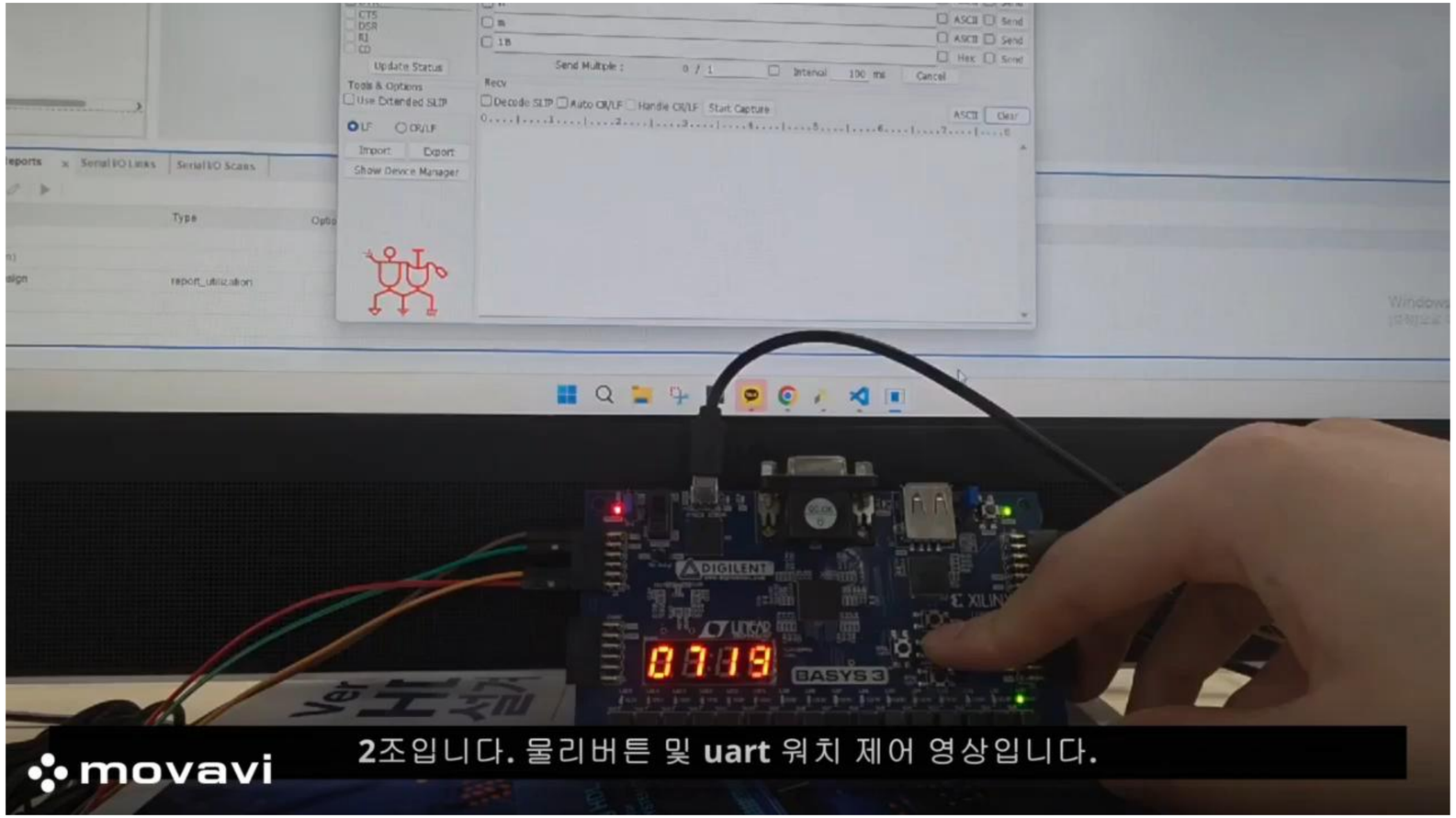
CONCLUSION

CHAPTER 06

WATCH/ STOPWATCH, SR04 SENSOR, DHT11 SENSOR를 종합한 동작 영상 이뤄낸 점, 아쉬운 점, 추가로 개발 가능성 등의 후기를 정리했습니다.

FINAL PROJECT SIMULATION VIDEO

WATCH/STOPWATCH TEST VIDEO

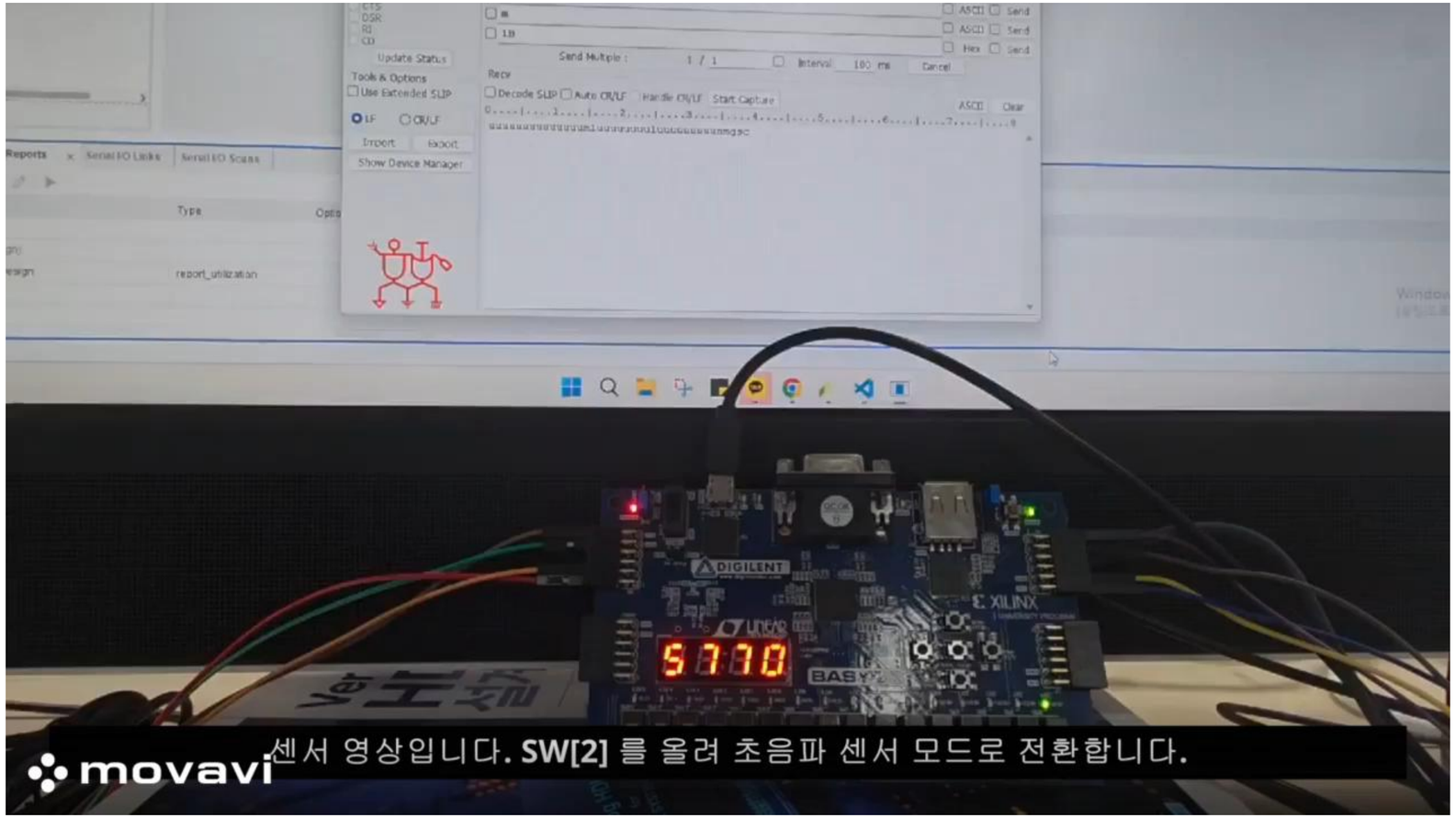


영상 설명

1. SWITCH [3:2] == 2’B00
2. SWITCH[15] 0: 버튼, 1: UART
3. 버튼 제어 검증
   WATCH U/D/L/R
   STOPWATCH RUN/STOP/CLEAR
4. UART 제어 검증: 상동 + 모드

FINAL PROJECT SIMULATION VIDEO

SR04/HDT11 SENSOR TEST VIDEO



영상 설명

1. SWITCH [3:2] == 2’B01
   초음파 센서 MODE
   DOWN 버튼 → 결과 출력
2. SWITCH [3:2] == 2’B10
   온·습도 센서 MODE
   UP 버튼 → 결과 출력

후기

S 목표 달성

- SR04 + UART 연결 성공
- DHT11 + UART 연결 성공
- 스위치로 WATCH/STOPWATCH + SR04 + DHT11 모드를 구분하며 UART 연결 성공

W 아쉬운 점

SW[15]를 통한 UART 및 Sender 멘트 추가 이외의 추가 아이디어 기능을 추가하지 못해 아쉬웠다.

O 추가 개발 가능성

StopWatch를 제어한 것처럼 CU을 만들어 각 센서 동작도 UART로 제어 (D면 거리, T면 온도 등..)

T 총평

각자 코드의 변수의 선언이 달라서 통합하는 과정에서 많은 어려움을 겪었다.
팀 프로젝트인 만큼 더 많은 소통을 통해 변수 이름을 미리 정하는 것이 좋을 것 같다.

THANK YOU!
감사합니다!

AI 시스템 반도체 설계 2기 | 2조 신상학, 엄찬하, 이현수, 정은지