명지 공학인 윤리 서약서

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

시험 윤리 서약

1. 나는 대리시험을 청탁하거나 청탁받지 않겠습니다.
2. 나는 허용되지 않은 교과서, 노트 및 타학생의 답안지 등을 보고 답안지를 작성하지 않겠습니다.
3. 나는 타인에게 답안지를 보여주지 않겠습니다.
4. 나는 감독관의 지시와 명령에 따라 시험 과정에 참여하겠습니다.

나는 시험에 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

2022년 11월 14일

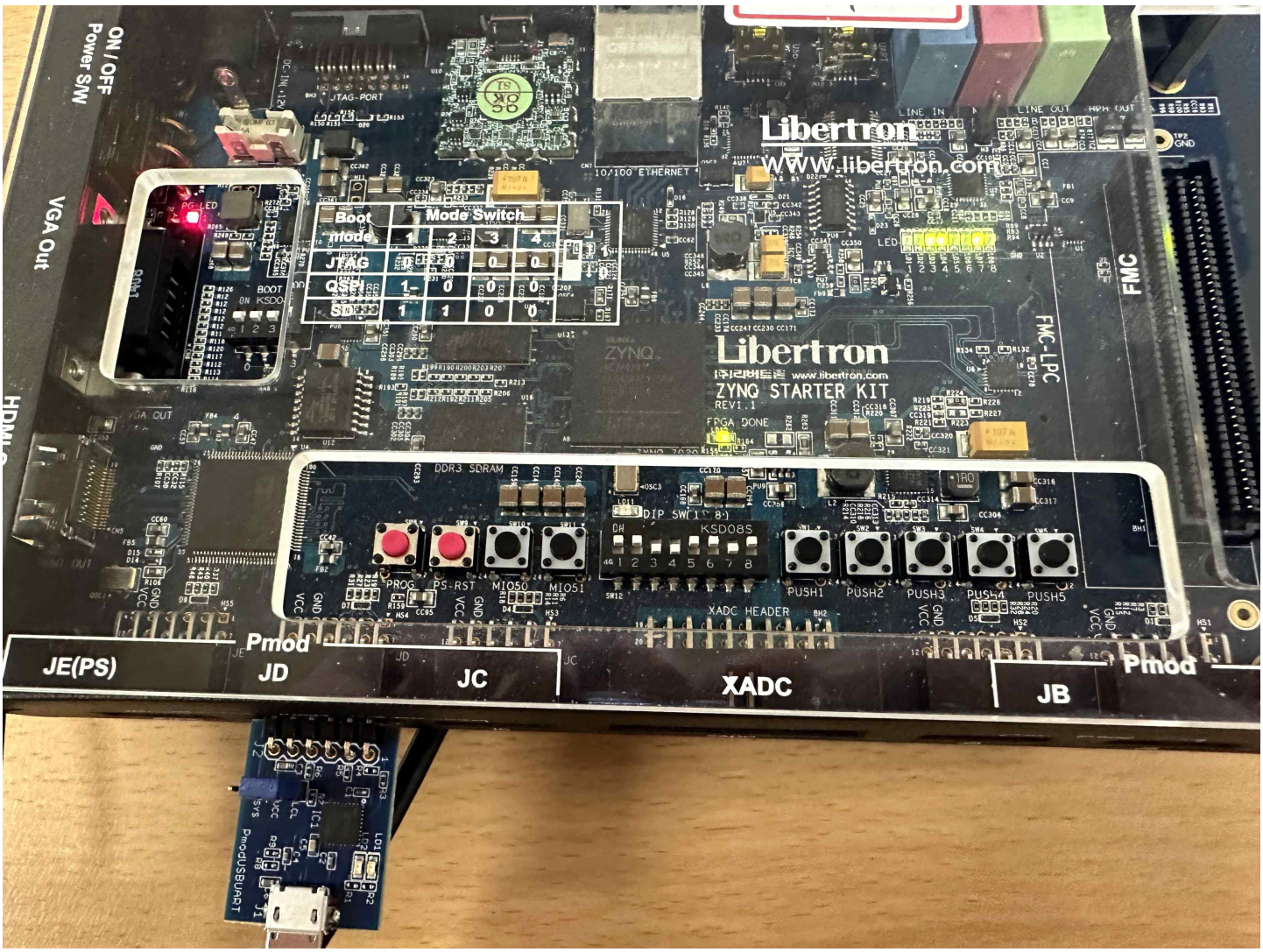
서약자

(학번) 60171878

(성명) 허무혁 **(인)**

UART 구현

60171878 허무혁



목차

1. Verilog 소스코드

top.v 코드

```
module uart_top(
        input clk,
        input rst,
        // tx
        input uart_tx_en,
        input [7:0] uart_tx_data,
        output uart_txd,
        //rx
        input uart_rxd,
        output [7:0] uart_rx_data
);

uart_tx tx_inst(clk, rst, uart_tx_en, uart_tx_data, , uart_txd);
uart_rx rx_inst(clk, rst, uart_rxd, , uart_rx_data);

endmodule
```

uart_tx.v 코드

```
module uart_tx (
    input clk,
    input rst,
    input uart_tx_en,          //전송 버튼,debouncer입력으로
    input [7:0] uart_tx_data, //송신할 8비트 data
    output reg tx_busy,        //idle 상태가 아닐때 '1'
    output reg uart_txd        //송신 데이터
);
wire uart_start_pulse;
reg [3:0] state;
reg [3:0] next;
reg [10:0] o;
parameter IDLE=4'b0000, START=4'b0001, BIT0=4'b0010,
BIT1=4'b0011, BIT2=4'b0100, BIT3=4'b0101, BIT4=4'b0110,
BIT5=4'b0111, BIT6=4'b1000, BIT7=4'b1001, STOP=4'b1010;

always@(*)begin
    case(state)
        IDLE : begin tx_busy = 1'b0; uart_txd = 1'b1; next = IDLE; end
        START : begin tx_busy = 1'b1; uart_txd = 1'b0; next = BIT0; end
        BIT0 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[0]; next = BIT1; end
        BIT1 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[1]; next = BIT2; end
        BIT2 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[2]; next = BIT3; end
        BIT3 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[3]; next = BIT4; end
        BIT4 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[4]; next = BIT5; end
```

```
        BIT5 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[5]; next = BIT6; end
        BIT6 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[6]; next = BIT7; end
        BIT7 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[7]; next = STOP; end
        STOP : begin tx_busy = 1'b1; uart_txd = 1'b1; next = IDLE; end
        default : begin tx_busy = 1'b1; uart_txd = 1'b1; next = IDLE; end
    endcase
end

always@(posedge clk or posedge rst) begin
    if (rst) begin state <= IDLE; end
    else if(uart_start_pulse) state <= START;
    else if(o == 868) begin state <= next; end
end

always@(posedge clk or posedge rst) begin
    if (rst) o <= 0;
    else if(tx_busy) begin
        if(o == 868) o <= 0;
        else o <= o + 1;
    end
end

 // assign uart_start_pulse = uart_tx_en; //for simulation
 debounce debounce_inst (clk, rst, uart_tx_en, , uart_start_pulse); //for kit
endmodule
```

uart_rx.v 코드

```
module uart_rx (
    input clk,
    input rst,
    input uart_rxd,          //수신 데이터
    output reg rx_busy,      //idle 상태가 아닐때 '1'
    output reg [7:0] uart_rx_data //수신한 8비트 data
);
reg [3:0] state;
reg [3:0] next;
reg [10:0] o;
reg [2:0] count;

parameter IDLE=4'b0000, START=4'b0001, BIT0=4'b0010,
BIT1=4'b0011, BIT2=4'b0100, BIT3=4'b0101, BIT4=4'b0110,
BIT5=4'b0111, BIT6=4'b1000, BIT7=4'b1001, STOP=4'b1010;

always@(*) begin
    case(state)
        IDLE : begin rx_busy = 1'b0; next = IDLE; end
```

```
        START : begin rx_busy = 1'b1; next = BIT0; end
        BIT0 : begin rx_busy = 1'b1; next = BIT1; end
        BIT1 : begin rx_busy = 1'b1; next = BIT2; end
        BIT2 : begin rx_busy = 1'b1; next = BIT3; end
        BIT3 : begin rx_busy = 1'b1; next = BIT4; end
        BIT4 : begin rx_busy = 1'b1; next = BIT5; end
        BIT5 : begin rx_busy = 1'b1; next = BIT6; end
        BIT6 : begin rx_busy = 1'b1; next = BIT7; end
        BIT7 : begin rx_busy = 1'b1; next = STOP; end
        STOP : begin rx_busy = 1'b1; next = IDLE; end
        default : begin rx_busy = 1'b1; next = IDLE; end
    endcase
end

always@(posedge clk or posedge rst) begin
    if(rst) begin uart_rx_data <= 0; state <= IDLE; end
    else if(rx_busy == 1'b0 && uart_rxd == 1'b0) state <= START;
    else if(o == 868) state <= next;
    else if(o == 434) begin
        case(state)
            BIT0 : begin uart_rx_data[0] <= uart_rxd; end
            BIT1 : begin uart_rx_data[1] <= uart_rxd; end
            BIT2 : begin uart_rx_data[2] <= uart_rxd; end
            BIT3 : begin uart_rx_data[3] <= uart_rxd; end
            BIT4 : begin uart_rx_data[4] <= uart_rxd; end
            BIT5 : begin uart_rx_data[5] <= uart_rxd; end
            BIT6 : begin uart_rx_data[6] <= uart_rxd; end
            BIT7 : begin uart_rx_data[7] <= uart_rxd; end
        endcase
    end
end

always@(posedge clk or posedge rst) begin
    if (rst) o <= 0;
    else if (rx_busy) begin
        if(o == 868) o <= 0;
        else o <= o + 1;
    end
end

endmodule
```

debounce.v 코드

```
module debounce(clk, rst, btn_in, btn_out, btn_out_pulse);
parameter SIZE = 16; //if pressed for 1/clk*2^(SIZE-1)sec, it can be debounced. 5.46ms for 6MHz
parameter BTN_WIDTH = 5;
```

```
input clk, rst;
input [BTN_WIDTH-1:0] btn_in;
output [BTN_WIDTH-1:0] btn_out, btn_out_pulse;
reg [BTN_WIDTH-1:0] btn_in_d [1:4]; //btn delayed
wire set; //sync reset to zero
reg [SIZE-1:0] o = {SIZE{1'b0}}; //counter is initialized to 0
always @(posedge clk or posedge rst)
begin
if (rst) begin
btn_in_d[1] <= 0;
btn_in_d[2] <= 0;
btn_in_d[3] <= 0;
o <= 0;
end
else begin
btn_in_d[1] <= btn_in;
btn_in_d[2] <= btn_in_d[1];
if (set == 1) o <= 0; //reset counter when input is changing
else if (o[SIZE-1] == 0) o<=o+1; //stable input time is not yet met
else btn_in_d[3] <= btn_in_d[2]; //stable input time is met, catch the btn and retain.
end
end
assign btn_out = btn_in_d[3]; //debounced button stable out
assign set = (btn_in_d[1] != btn_in_d[2])? 1 : 0; //determine when to reset counter
always @(posedge clk or posedge rst) begin
if (rst) btn_in_d[4] <= 0;
else btn_in_d[4] <= btn_in_d[3];
end
assign btn_out_pulse = btn_in_d[3] & (~btn_in_d[4]); //debounced button pulse out
endmodule
```

테스트벤치 코드

```
module tb_tx;
reg clk, rst ,uart_tx_en, uart_rxd;
reg [7:0] uart_tx_data;
wire[7:0] uart_rx_data;
wire tx_busy, rx_busy;
wire uart_txd;

initial begin
clk = 0; rst = 1; uart_tx_en = 0; uart_rxd = 0;
uart_tx_data = 00010001;
#15; rst = 0;
#10; uart_tx_en = 1;
#15; uart_tx_en = 0;
end
```

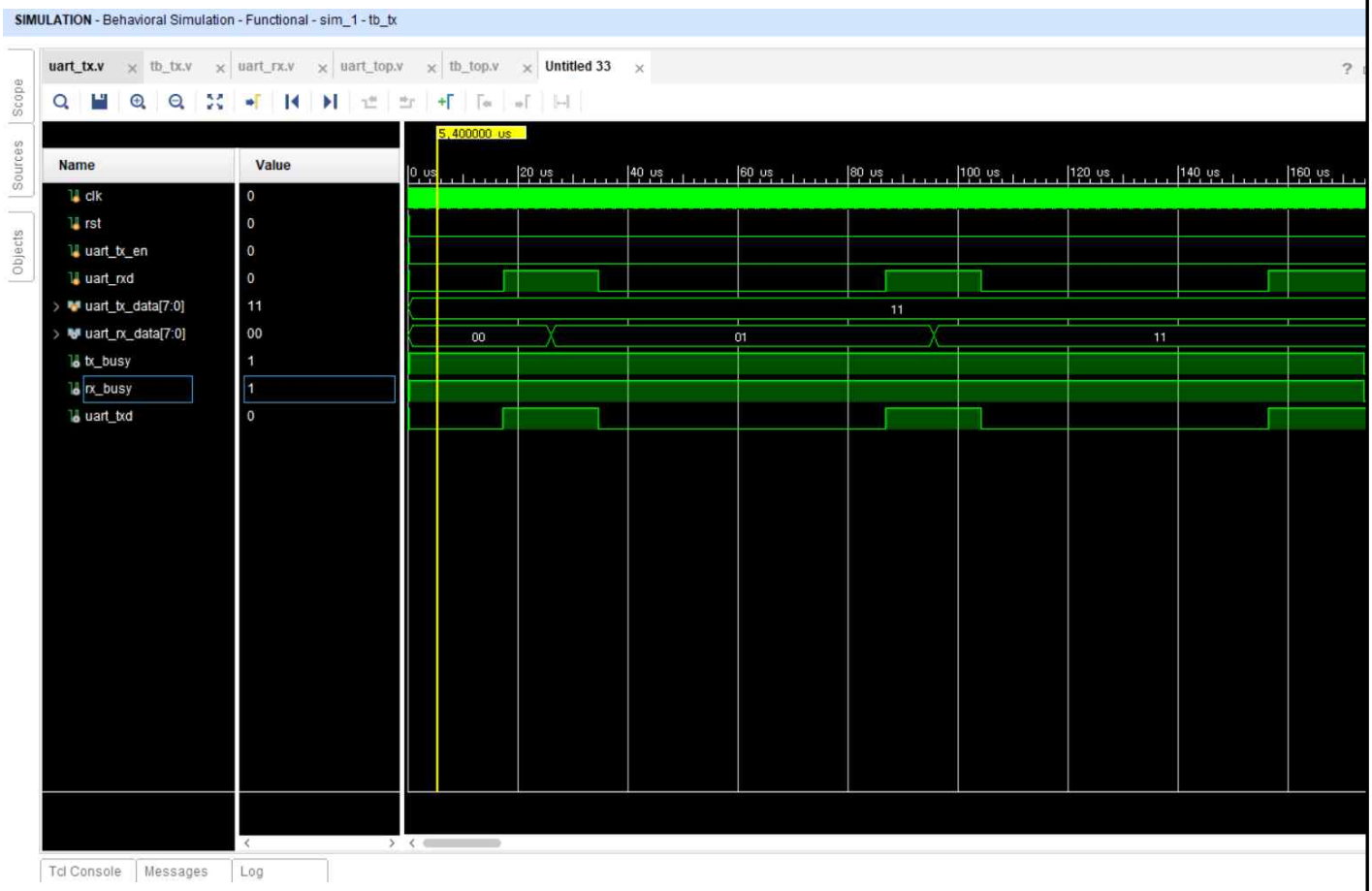
```
always begin
#10; clk = ~clk; uart_rxd = uart_txd;
end

uart_tx dut1(clk, rst, uart_tx_en, uart_tx_data, tx_busy, uart_txd);
uart_rx dut2(clk, rst, uart_rxd, rx_busy , uart_rx_data);
endmodule
```

2. 타이밍도 캡쳐

타이밍도

//다음 타이밍도는 uart_tx_en 신호가 들어 왔을 때 uart_rx_data의 8bit데이터가 정해둔 클럭분주(baud rate = 115,200bps)를 타고 1bit uart_txd 출력신호를 분주에 맞추어 내보내는 모습을 보여준다.
uart_rxd 신호가 1에서 0으로 떨어졌을 때 수신을 시작하여 uart_rxd 입력신호를 정해둔 클럭분주에 맞추어 8bit uart_rx_data를 통하여 읽어들이는 모습을 보여준다.
이때 테스트 벤치 검증을 위하여 uart_txd의 출력과 uart_rxd의 입력을 연결하였다.



3. Discussion

설계방법

UART(Universal Asynchronous Receiver / Transmitter)는 ASynchronus로 송신기와 수신기가 공통 클럭 신호를 공유하지 않는다. 따라서 UART는 양 끝단에서 동일한 비트 타이밍을 보장하기 위해 사전에 정의된 속도로 통신해야 한다. 가장 일반적으로 사용되는 UART 보 레이트(baud rates)는 4800, 9600, 19.2K, 57.6K, 115.2K이다. 동일한 baud rate를 적용해야 한다.
UART의 통신 구조는 다음과 같다.

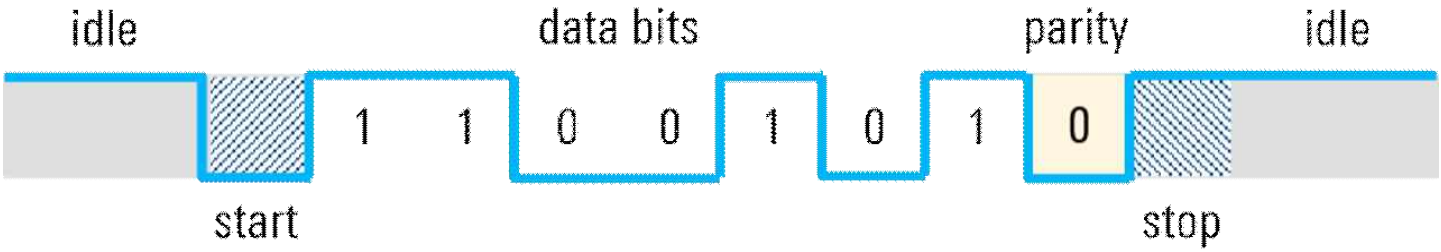
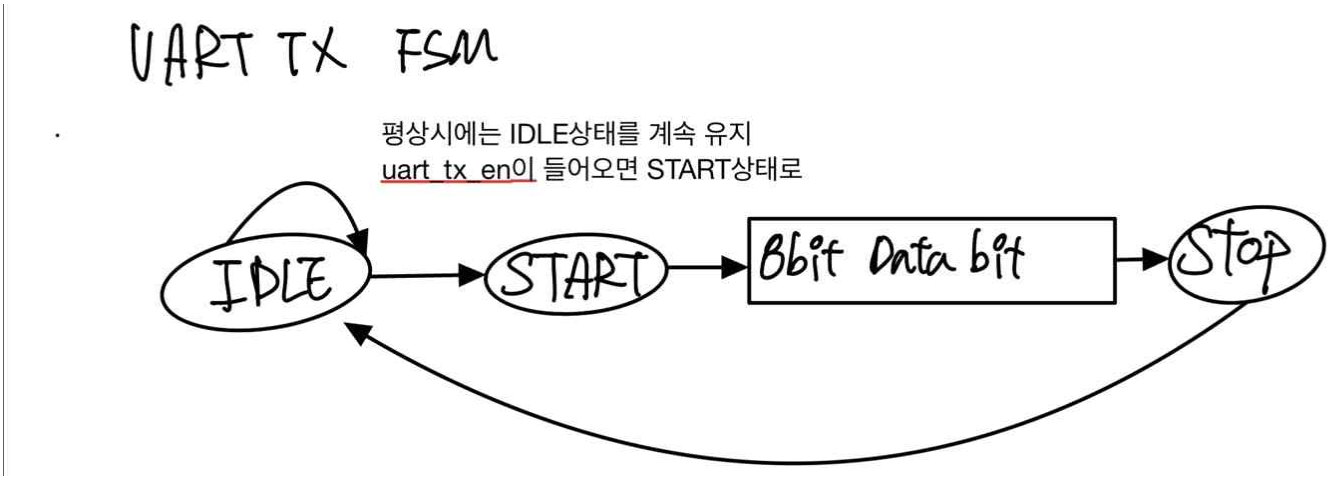


그림 4

https://cdn.rohde-schwarz.com/pws/solution/research___education_1/educational_resources_/oscilloscope_and_probe_fundamentals/05_Understanding-UART_02_w1280_hX.png

UART는 비동기식이므로 TX에서 데이터 비트가 들어오고 있다는 신호를 보내야 한다. 데이터가 입력되면 START BIT는 1(5V, IDLE 상태)에서 0(0V)로 전환된다, START BIT가 들어오고 다음으로 DATA BIT가 전송된다. PARITY BIT는 오류검증을 위한 BIT이며 이번 과제에서는 생략하였다. DATA BITS(8bit)가 모두 전송되면 STOP BIT를 통하여 정지됨을 알린다. 여기에서는 1BIT를 사용하였다. 이후 다시 IDLE 상태로 들어간다.

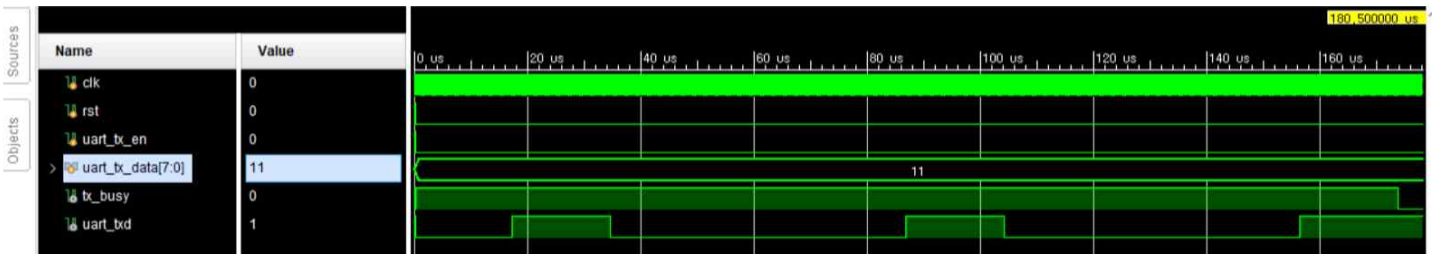
RX단에서 데이터를 수신할 때는 입력신호가 1(5V, IDLE)이었다가 0으로 떨어지면 데이터 수신이 시작됨을 알린다. 이후 8bit data를 수신하고 STOP BIT를 만든 뒤 다시 IDLE 상태로 돌아간다.
다음은 TX단의 FSM 설계 모델이다.




```
always@(*)begin
    case(state)
        IDLE : begin tx_busy = 1'b0; uart_txd = 1'b1; next = IDLE; end
        START : begin tx_busy = 1'b1; uart_txd = 1'b0; next = BIT0; end
        BIT0 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[0]; next = BIT1; end
        BIT1 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[1]; next = BIT2; end
        BIT2 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[2]; next = BIT3; end
        BIT3 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[3]; next = BIT4; end
        BIT4 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[4]; next = BIT5; end
        BIT5 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[5]; next = BIT6; end
        BIT6 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[6]; next = BIT7; end
        BIT7 : begin tx_busy = 1'b1; uart_txd = uart_tx_data[7]; next = STOP; end
```

```
        STOP : begin tx_busy = 1'b1; uart_txd = 1'b1; next = IDLE; end
        default : begin tx_busy = 1'b1; uart_txd = 1'b1; next = IDLE; end
    endcase
end
always@(posedge clk or posedge rst) begin
    if (rst) begin state <= IDLE; end
    else if(uart_start_pulse) state <= START;
    else if(o == 868) begin state <= next; end
end
```

상태는 평시에는 IDLE 상태를 유지하다 전송버튼이 눌리면 상태를 START로 옮긴다. 그 후 정해진 count마다 상태를 옮기며 출력과 다음상태를 결정한다. DATA BIT들의 상태가 되면 1bit uart_txd의 출력을 uart_tx_data[해당bit]의 입력으로 결정한다. 다음은 입력으로 00010001을 넣었을 때 uart_txd가 결정되는 모습이다.



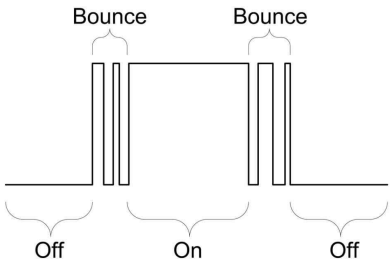
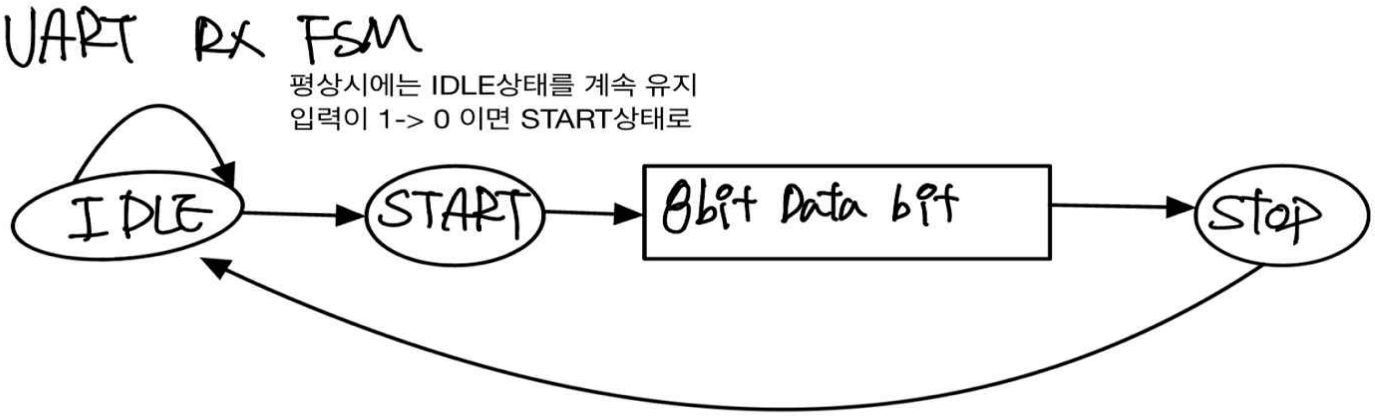


그림 7
https://blog.naver.com/roboholic84/220880882127

이 때 TX의 자료전송을 버튼을 눌러하므로 떨림을 방지하기 위해 debounce 코드로 방지한다.

다음과 같이 ASCII코드를 H로 두고 전송하였을 때 TERA TERM이 송신값을 잘 수신하고 있다.

다음은 RX단의 모델이다.



RX의 FSM모델은 TX와 같지만 차이점은 입력이 1->0으로 떨어져야 state가 start로 바뀐다는 것이다. 또다른 차이점은

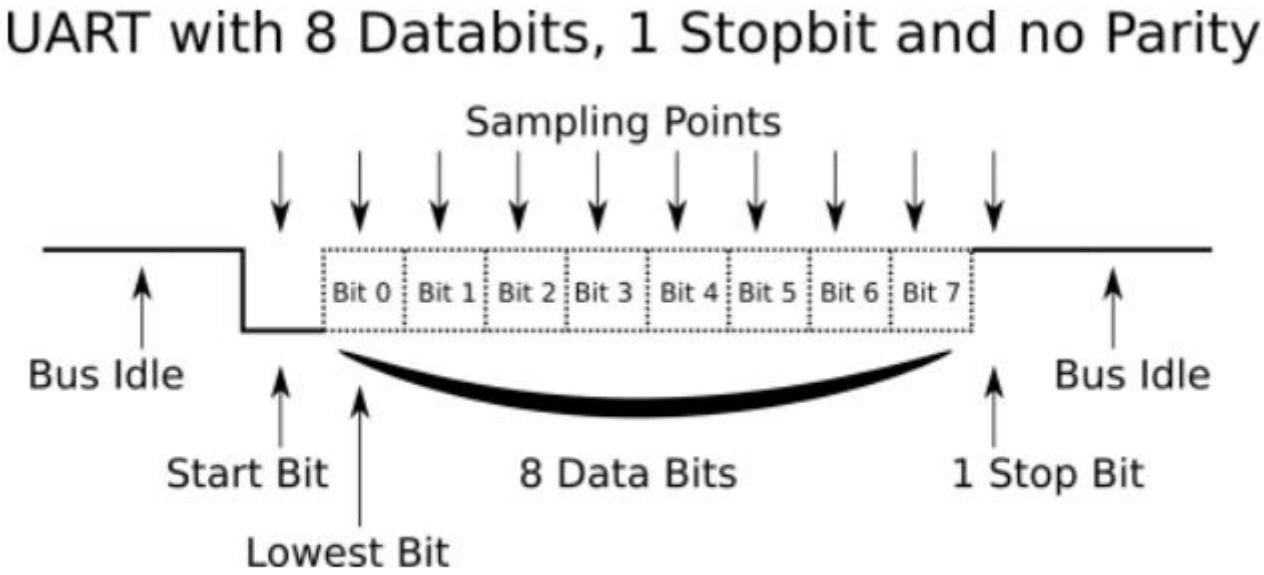


그림 10 https://shek.tistory.com/41

데이터 BIT의 값을 clock edge가 아닌 중간에서 읽어야 한다.

```
always@(*) begin
    case(state)
        IDLE : begin rx_busy = 1'b0; next = IDLE; end
        START : begin rx_busy = 1'b1; next = BIT0; end
        BIT0 : begin rx_busy = 1'b1; next = BIT1; end
        BIT1 : begin rx_busy = 1'b1; next = BIT2; end
        BIT2 : begin rx_busy = 1'b1; next = BIT3; end
        BIT3 : begin rx_busy = 1'b1; next = BIT4; end
        BIT4 : begin rx_busy = 1'b1; next = BIT5; end
```

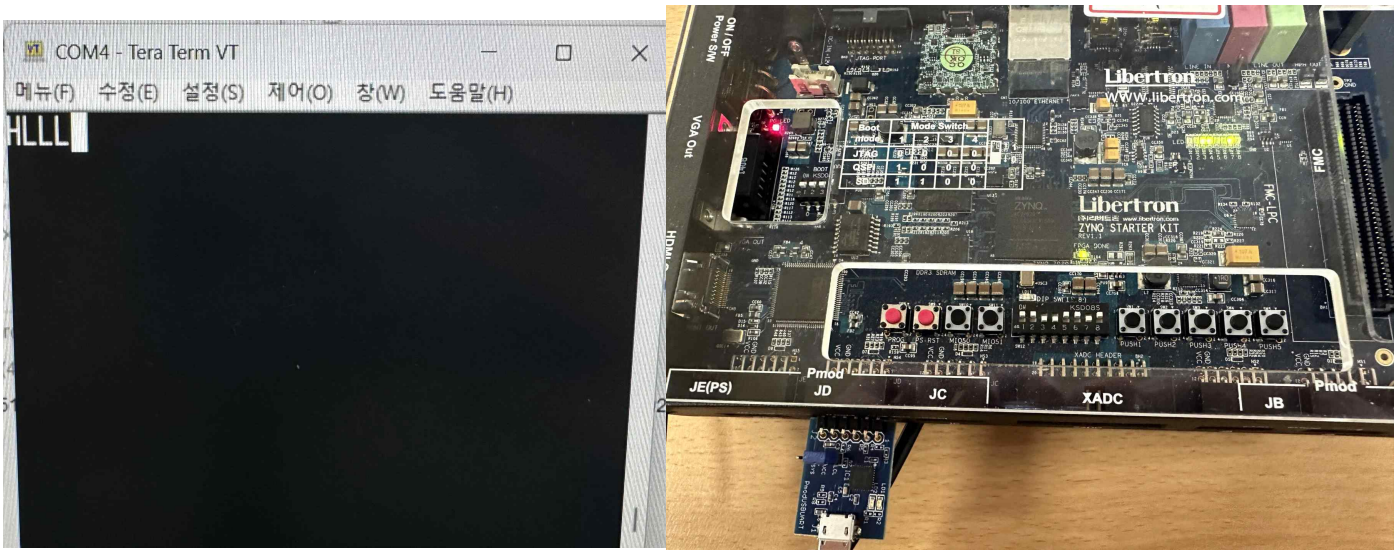
```
        BIT5 : begin rx_busy = 1'b1; next = BIT6; end
        BIT6 : begin rx_busy = 1'b1; next = BIT7; end
        BIT7 : begin rx_busy = 1'b1; next = STOP; end
        STOP : begin rx_busy = 1'b1; next = IDLE; end
        default : begin rx_busy = 1'b1; next = IDLE; end
    endcase
end

always@(posedge clk or posedge rst) begin
    if(rst) begin uart_rx_data <= 0; state <= IDLE; end
    else if(rx_busy == 1'b0 && uart_rxd == 1'b0) state <= START;
    else if(o == 868) state <= next;
    else if(o == 434) begin
        case(state)
            BIT0 : begin uart_rx_data[0] <= uart_rxd; end
            BIT1 : begin uart_rx_data[1] <= uart_rxd; end
            BIT2 : begin uart_rx_data[2] <= uart_rxd; end
            BIT3 : begin uart_rx_data[3] <= uart_rxd; end
            BIT4 : begin uart_rx_data[4] <= uart_rxd; end
            BIT5 : begin uart_rx_data[5] <= uart_rxd; end
            BIT6 : begin uart_rx_data[6] <= uart_rxd; end
            BIT7 : begin uart_rx_data[7] <= uart_rxd; end
        endcase
    end
end
```

따라서 else if(rx_busy == 1'b0 && uart_rxd == 1'b0) state <= START;에서 보면 상태가 IDLE이고 uart_rxd가 0으로 떨어졌을 때 상태를 START로 바뀌도록 설계했다.

```
 else if(o == 868) state <= next;
    else if(o == 434) begin
        case(state)
            BIT0 : begin uart_rx_data[0] <= uart_rxd; end
        ......
```

데이터를 읽을 때는 중간에서 읽도록 하였다.



다음과 같이 컴퓨터(TX)에서 L을 전송하였을 때 키트(RX)의 LED에 LED가 해당 ASCII에 맞게 켜지는 것을 확인할 수 있다.

오류와 해결방법

1. TX의 state 머신을 처음 설계할 때 데이터가 IDLE상태에서 넘어가지 않는 오류가 있었다. always@(*)의 case 문에서 IDLE의 다음상태를 if(버튼) next =START else next = IDLE로 하고 always@(clk)에서 868 클락마다 다음 state로 넘어가게 해놓았다. 그러나 버튼이 눌리면 next가 START로 바뀌었다가 버튼을 떼면 바로 IDLE로 바뀐다. 따라서 코드를 다음과 같이 수정하여 해결하였다.

```
always@(posedge clk or posedge rst) begin
    if (rst) begin state <= IDLE; end
    else if(uart_start_pulse) state <= START;
……
```

다음과 같이 버튼이 눌릴 때 바로 다음 상태를 START로 넘어가도록 하였다.

2. 다음과 같이 수정하고 난 후 컴퓨터로 데이터를 수신할 때 다음과 같은 오류가 생겼다.

보통 이런 오류는 uart에서 정확한 타이밍에 데이터가 전송되지 않을경우에 발생한다.

```
always@(posedge clk or posedge rst) begin
    if (rst) o <= 0;
    else if(o == 868) o <= 0;
    else o <= o + 1;
    end
end
```

처음에는 다음과 같이 코드를 구성하였는데

```
always@(posedge clk or posedge rst) begin
    if (rst) o <= 0;
    else if(tx_busy) begin
```

```
        if(o == 868) o <= 0;
        else o <= o + 1;
    end
end
```

다음과 같이 코드를 고쳤다. IDLE 상태가 START 상태가 되면 tx_busy 플래그가 0이었다가 1로 바뀌는 것을 이용하여 START상태가 시작되었을 때 count를 시작하여 타이밍에 맞게 데이터를 전송하도록 수정하였다.

4. RX를 설계할 때 발생한 오류는 8bit uart_rx_data의 출력값을 결정하는 코드에서 latch로 합성되는 오류가 발생했다. 보통 latch로 합성될 때는 always 문에서 default나 else 상태를 구성하지 않을 때 발생한다. 그러나 default와 else 상태를 모두 정의했을때도 오류가 잔존했다. 따라서 uart_rx_data를 always@(clk)에 가져와 오류를 고쳤다.

5. 다음은 어떻게 RX의 데이터가 수신되고 있을 때 타이밍의 정중앙에서 데이터를 읽게하는 아이디어에 난항을 격었다. 그러다 4의 오류를 고치며 카운트가 중간쯤 올라갔을 때 값을 읽도록하여 코드를 완성했다.

```
always@(posedge clk or posedge rst) begin
    if(rst) begin uart_rx_data <= 0; state <= IDLE; end
    else if(rx_busy == 1'b0 && uart_rxd == 1'b0) state <= START;
    else if(o == 868) state <= next;
    else if(o == 434) begin
        case(state)
            BIT0 : begin uart_rx_data[0] <= uart_rxd; end
            BIT1 : begin uart_rx_data[1] <= uart_rxd; end
            BIT2 : begin uart_rx_data[2] <= uart_rxd; end
            BIT3 : begin uart_rx_data[3] <= uart_rxd; end
            BIT4 : begin uart_rx_data[4] <= uart_rxd; end
            BIT5 : begin uart_rx_data[5] <= uart_rxd; end
            BIT6 : begin uart_rx_data[6] <= uart_rxd; end
            BIT7 : begin uart_rx_data[7] <= uart_rxd; end
        endcase
    end
end
```

6. RX를 검증할 때 컴퓨터에서 입력한 값을 키트의 LED가 출력하게 되는데 같은 입력을 넣었을 때 정상 출력되는 경우가 있고 1bit씩 shift되어 출력되는 경우가 있었다. 해당 오류의 원인을 정확한 타이밍에 값을 읽지 못해서 생기는 오류라고 생각하고 다음과 같이 코드를 작성했다.

```
always@(posedge clk or posedge rst) begin
    if (rst) o <= 0;
    else if (rx_busy) begin
        if(o == 868) o <= 0;
        else o <= o + 1;
end
```

START상태가 시작되었을 때부터 count를 시작하여 보았다. 그러니 오류가 고쳐졌다.