Course: CS 223 Digital Design

Section: 6

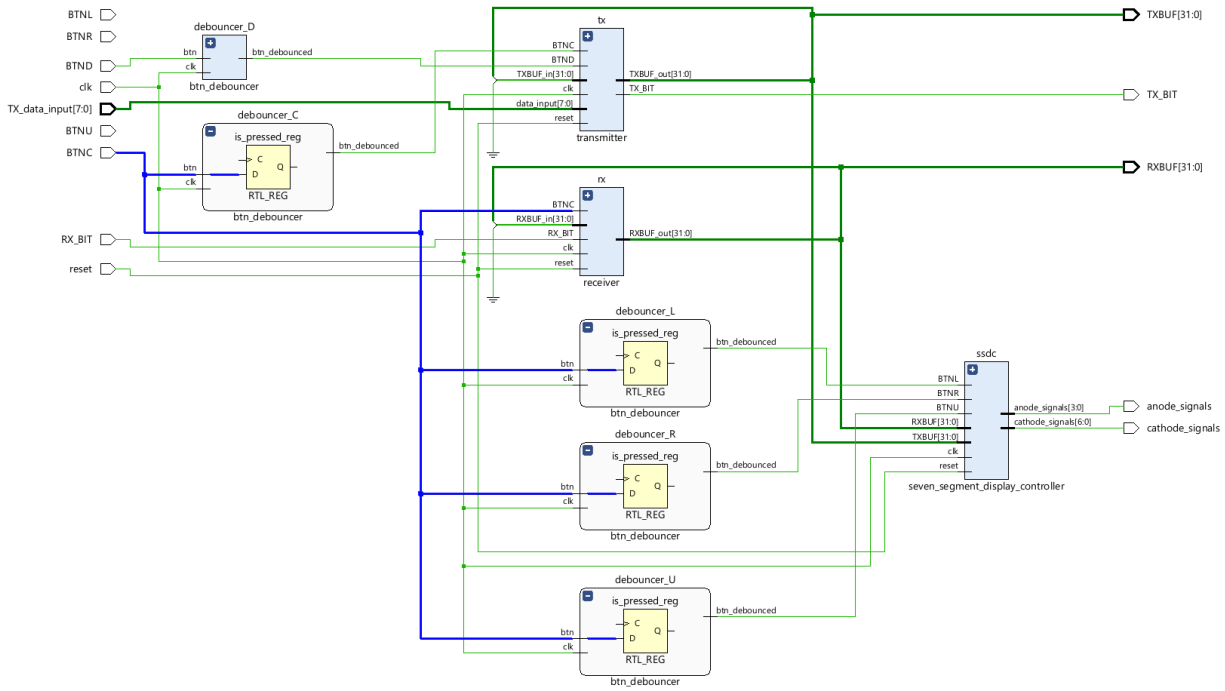
PROJECT

Name: Halil AVCI

Student ID: 22003476

Date: 26/11/2023

RTL DESIGN



The UART, consists of transmitter, receiver, seven segment display controller, and button rebouncer. Transmitter transfers the input value bit by bit and manages the down and center buttons. When overloaded, It shifts and discards the oldest data. Receiver receives the bits and packs them into meaningful 1-byte of data. It has similar data storing system with the transmitter. Seven segment display controller takes inputs from RXBUF and TXBUF, and shows the intended value on the seven segment displayer in hexadecimal. Button rebouncer controls the button behaviors. It tracks the button bits. When button becomes 1 in 5 clock cycle consecutively, it turns on the button and assigns it to untill the user stops pushing the button.

CODE:

```
`timescale 1ns / 1ps
module UART #(parameter DATA_WIDTH = 8, STOP_BITS = 2, BAUD_RATE = 115200) (
    input clk,
    input logic reset,
    input logic BTND,
    input logic BTNC,
    input logic BTNL,
    input logic BTNR,
    input logic BTNU,
    input logic [DATA_WIDTH - 1:0] TX_data_input,
    input logic RX_BIT,
    output logic TX_BIT,
    output logic anode_signals,
    output logic cathode_signals,
    output logic [4 * DATA_WIDTH - 1:0] TXBUF,
    output logic [4 * DATA_WIDTH - 1:0] RXBUF
    );

    wire internal_TXBUF = TXBUF;
    wire internal_RXBUF = RXBUF;

    btn_debouncer debouncer_D(clk, BTND, debounced_BTND);
    btn_debouncer debouncer_C(clk, BTNC, debounced_BTNC);
    btn_debouncer debouncer_L(clk, BTNC, debounced_BTNL);
    btn_debouncer debouncer_R(clk, BTNC, debounced_BTNR);
    btn_debouncer debouncer_U(clk, BTNC, debounced_BTNU);

    seven_segment_display_controller #(DATA_WIDTH) ssdc(
```

```
        clk, reset,debounced_BTNL, debounced_BTNR, debounced_BTNU,
            RXBUF, TXBUF, anode_signals, cathode_signals);

    transmitter #(DATA_WIDTH, STOP_BITS, BAUD_RATE) tx(
        clk, reset, debounced_BTND, debounced_BTNC,
            TX_data_input, internal_TXBUF, TXBUF, TX_BIT);

    receiver #(DATA_WIDTH, STOP_BITS, BAUD_RATE) rx(
        clk, reset, BTNC, RX_BIT, internal_RXBUF, RXBUF);

endmodule

`timescale 1ns / 1ps

module btn_debouncer(
    input clk,
    input logic btn,
    output logic btn_debounced
    );

    logic [4:0] counter;
    logic is_pressed;
    logic stable_value;

    always@(posedge clk) begin
        is_pressed <= btn;
        counter <= {counter[3:0], is_pressed};
        if (&counter)
            stable_value <= 1'b1;
```

```
        else
            stable_value <= 1'b0;
    end

    assign btn_debounced = stable_value;

endmodule

`timescale 1ns / 1ps

module seven_segment_display_controller #(parameter DATA_WIDTH = 8) (
    input clk,
    input logic reset,
    input logic BTNL,
    input logic BTNR,
    input logic BTNU,
    input logic [4 * DATA_WIDTH - 1:0] RXBUF,
    input logic [4 * DATA_WIDTH - 1:0] TXBUF,
    output logic [3:0] anode_signals,
    output logic [6:0] cathode_signals
    );

    logic [19:0] counter;
    logic [1:0] LED_activation;
    logic [1:0] display_options;
    logic [15:0] data_to_display;
    logic [3:0] digit_to_display;
    logic change_display;
```

```
always@(posedge clk) begin
    if (reset) begin
        counter <= 0;
    end
    else begin
        counter <= counter + 1;
    end
end

assign LED_activation = counter[19:18];

always@(posedge clk) begin
    if (BTNL || BTNR) begin
        if (display_options[0])
            display_options[0] <= 1'b0;
        else
            display_options[0] <= 1'b1;

        change_display <= 1;
    end

    if (BTNU) begin
        if (display_options[1])
            display_options[1] <= 1'b0;
        else
            display_options[1] <= 1'b1;

        change_display <= 1;
```

```
end

case (LED_activation)
    2'b00: begin
        anode_signals <= 4'b0111;
        digit_to_display[3:0] <= data_to_display[15:12];
    end
    2'b01: begin
        anode_signals <= 4'b1011;
        digit_to_display[3:0] <= data_to_display[11:8];
    end
    2'b10: begin
        anode_signals <= 4'b1101;
        digit_to_display[3:0] <= data_to_display[7:4];
    end
    2'b11: begin
        anode_signals <= 4'b1110;
        digit_to_display[3:0] <= data_to_display[3:0];
    end
    default: begin
        anode_signals <= 4'b0111;
        digit_to_display[3:0] <= 4'b0000;
    end
endcase

if (change_display) begin
    change_display <= 0;
    case (display_options)
    2'b00: begin
```

```
        data_to_display[15:0] <= TXBUF[4 * DATA_WIDTH - 1: 2 * DATA_WIDTH];
    end
    2'b01: begin
        data_to_display[15:0] <= TXBUF[2 * DATA_WIDTH - 1:0];
    end
    2'b10: begin
        data_to_display[15:0] <= RXBUF[4 * DATA_WIDTH - 1: 2 * DATA_WIDTH];
    end
    2'b11: begin
        data_to_display[15:0] <= TXBUF[2 * DATA_WIDTH - 1:0];
    end
    default: begin
        data_to_display[15:0] <= TXBUF[4 * DATA_WIDTH - 1: 2 * DATA_WIDTH];
    end
    endcase
end

    case(digit_to_display)
    4'b0000: cathode_signals = 7'b0000001; // "0"
    4'b0001: cathode_signals = 7'b1001111; // "1"
    4'b0010: cathode_signals = 7'b0010010; // "2"
    4'b0011: cathode_signals = 7'b0000110; // "3"
    4'b0100: cathode_signals = 7'b1001100; // "4"
    4'b0101: cathode_signals = 7'b0100100; // "5"
    4'b0110: cathode_signals = 7'b0100000; // "6"
    4'b0111: cathode_signals = 7'b0001111; // "7"
    4'b1000: cathode_signals = 7'b0000000; // "8"
    4'b1001: cathode_signals = 7'b0000100; // "9"
    4'b1010: cathode_signals = 7'b0001000; // "A"
```

```
        4'b1011: cathode_signals = 7'b0000000; // "B"
        4'b1100: cathode_signals = 7'b0110001; // "C"
        4'b1101: cathode_signals = 7'b0000011; // "D"
        4'b1110: cathode_signals = 7'b0110000; // "E"
        4'b1111: cathode_signals = 7'b0111000; // "F"
        default: cathode_signals = 7'b0000001; // "0"
        endcase
    end
endmodule

`timescale 1ns / 1ps
module transmitter #(parameter DATA_WIDTH = 8, STOP_BITS = 2, BAUD_RATE = 115200)
(
    input clk,
    input logic reset,
    input logic BTND,
    input logic BTNC,
    input logic [DATA_WIDTH-1:0] data_input,
    input logic [4*DATA_WIDTH-1:0] TXBUF_in,
    output logic [4*DATA_WIDTH-1:0] TXBUF_out,
    output logic TX_BIT
    );

    logic state; //0:IDLE 1:TRANSFER
    logic [3:0] data_counter;
    logic [15:0] baud_counter;
    logic [DATA_WIDTH + STOP_BITS:0] TX_LINE;
    logic DIV = 100000000 / BAUD_RATE;
```

```
always_ff@(posedge clk) begin
    if (reset) begin
            state <= 0;
            TX_LINE <= 1'b1;
            TX_BIT <= 1'b1;
    end

    if (BTND) begin
        TXBUF_out[4*DATA_WIDTH-1:8] <= TXBUF_in[4*DATA_WIDTH-9:0];
        TXBUF_out[4*DATA_WIDTH-25:0] <= data_input;
        TX_LINE <= {1'b0, data_input, 2'b1};
    end

    if (BTNC) begin
        state <= 1;
    end

    if (state) begin
        TX_BIT <= TX_LINE[DATA_WIDTH + STOP_BITS];
        baud_counter <= baud_counter + 1;

        if (baud_counter == DIV - 1) begin
            data_counter <= data_counter + 1;
            TX_LINE[10:1] <= TX_LINE[9:0];
            baud_counter <= 0;

            if (data_counter == 10) begin
                state <= 0;
            end
```

```
            end
        end
    end
endmodule

`timescale 1ns / 1ps
module receiver #(parameter DATA_WIDTH = 8, STOP_BITS = 2, BAUD_RATE = 115200) (
    input clk,
    input logic reset,
    input logic BTNC,
    input logic RX_BIT,
    input logic [4*DATA_WIDTH-1:0] RXBUF_in,
    output logic [4*DATA_WIDTH-1:0] RXBUF_out
    );

    logic state; //0:IDLE 1:RECEIVE
    logic [3:0] data_counter;
    logic [15:0] baud_counter;
    logic [9:0] data_received;
    logic DIV = 100000000 / BAUD_RATE;

    always_ff@(posedge clk) begin
        if (reset) begin
            state <= 0;
            data_counter <= 0;
            baud_counter <= 0;
        end

        if (!state) begin
```

```
        if (!RX_BIT) begin
            baud_counter <= baud_counter + 1;
        end
        else begin
            baud_counter <= 0;
        end

        if (baud_counter == (DIV / 2)) begin //[15:1]
            state <= 1;
            data_counter <= 0;
            baud_counter <= 0;
        end
    end
    else begin
        baud_counter <= baud_counter + 1;
        if (baud_counter == DIV) begin
            data_received <= {data_received[8:0], RX_BIT};
            data_counter <= data_counter + 1;
            baud_counter <= 0;

            if (data_counter == 10) begin
                RXBUF_out[4*DATA_WIDTH-1:8] <= RXBUF_in[4*DATA_WIDTH-9:0];
                RXBUF_out[4*DATA_WIDTH-25:0] <= data_received[9:2];
                state <= 0;
            end
        end
    end
end
```

```
endmodule
```