Course: CS 223 Digital Design

Section: 6

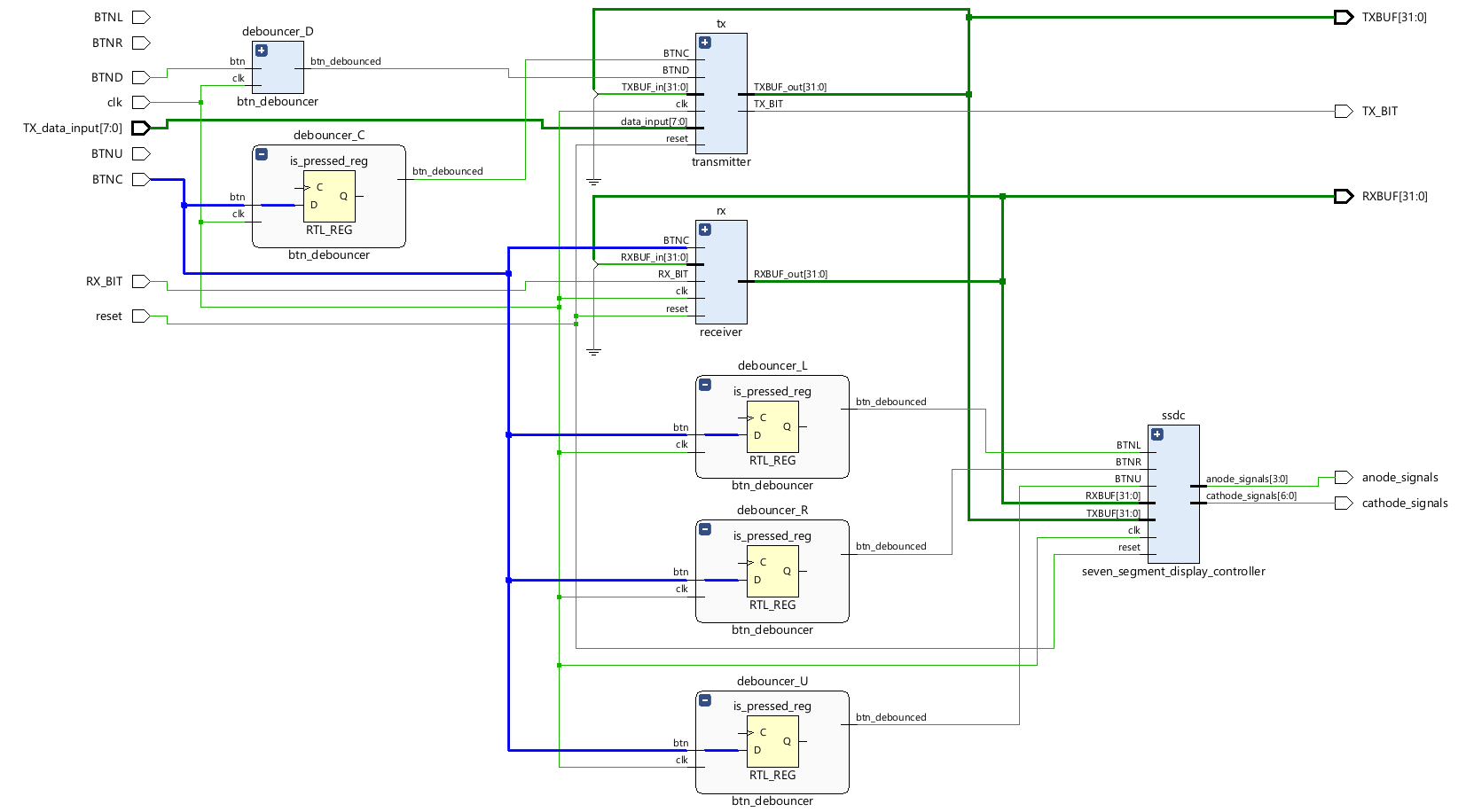
PROJECT

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