CS 223: Digital Design
Section: 1
Term Project

Halil Arda Özongun
22202709
05/05/2024

Detailed explanation of your design and implementation along with UART functionality:

Overview:

UART (Universal Asynchronous Receiver/Transmitter) is simply a data transmission tool. The word asynchronous in its name means that it is not necessary for the two devices to run on a common clock. It is sufficient for two devices exchanging data to agree on a pre-agreed baud rate.

The simple operating logic of the devices is as follows: The devices send high data to each other at any time, when a message is to be sent, it is sent low instead of high at the first moment. This way the receiver knows that it will receive a message. The following bits represent the data. The data is converted from 1's and 0's into real messages and communication is established.

To simplify the design I have grouped the project into 3 submodules under a main module called UART, these are TXBUF, RXBUF, SevenSegmentController. These modules also utilise some sub-modules within themselves. Here is a detailed description of each module:

1. **UART (Main Controller)**

   Serves as the top-level module integrating the transmitter (TXBUF), receiver (RXBUF), and display controller (SevenSegmentController). It contains 4 txbuf, 4 rxbuf. It sends these to the required modules and receives the data from the modules. It helps data transmission by sending the received data to SevenSegmentController. It controls inputs and outputs, and also provides communication between constrains and modules.

2. **TXBUF (Transmitter Buffer)**

   Txbuf module is the module where data transmission is done. It works with two buttons, one of the buttons allows the data in the switches to be loaded into the buffers while the other one allows the data in the buffers to be sent bit by bit. Buffers work with fifo (first in first out) logic. Accordingly, data input by switches is stored in txbuf0, and each time new data is stored, txbuf[i] is transferred to txbuf[i+1]. It uses a slow clock derived from the clk to meet baud rate requirements.

   There are two options when sending data, these are determined depending on whether switch15 is switched on or off. If sw15 is switched on this means that we send each data in the 4 txbufs in sequence. If sw15 is off, only the data in txbuf3 is sent.

   It works with more than one state during the data transmission process. It sends data including the start bit, data bits, parity bit and stop bit. The module is normally in the IDLE state. In this state, it outputs 1 continuously. When a button is pressed to send data, it first

sends 0 (start bit). After that, it sends other bits including the parity bit. After the data transmission is finished, it returns to IDLE and waits for the data transmission command.

3. **RXBUF (Receiver Buffer)**

   This module receives data from the other device and stores it in arrays. It receives one bit from the rx_line at a time. The received data is initially logic high. When data is to be received, a bit logic low is first received to indicate this. From then on, data is received bit by bit, and a parity bit is received after the data. The reason for the parity bit is to try to see if the received data is correct. If the check with the parity bit fails, an error is signalled via an LED output. If not, the data is received and stored in the rxbufs.

   The operating logic of the rxbufs is also similar to the logic of the TXBUF in the manner of fifo. The received data is first stored in rxbuf0. Before saving the data to rxbuf0, the data in rxbuf[i] is saved to rxbuf[3] and the data in rxbuf[3] is discarded.

4. **SevenSegmentController**

   This module takes 8 arrays, 4 txbuf and 4 rxbuf, and aims to display them according to the signals received from 3 buttons. depending on the value to be displayed, the first digit is t/r, the second digit is the array number of t and r, and the last two digits are intended to display the value inside in this array. It get helps from the HexTo7Segment module to determine the last two digits.

5. **Debounce**

   This module provides a stable button input by filtering out the mechanical noise generated during button presses. It implements a simple state machine to provide a stable high signal output after a button has been continuously pressed for a specified time defined by DEBOUNCE_LIMIT.

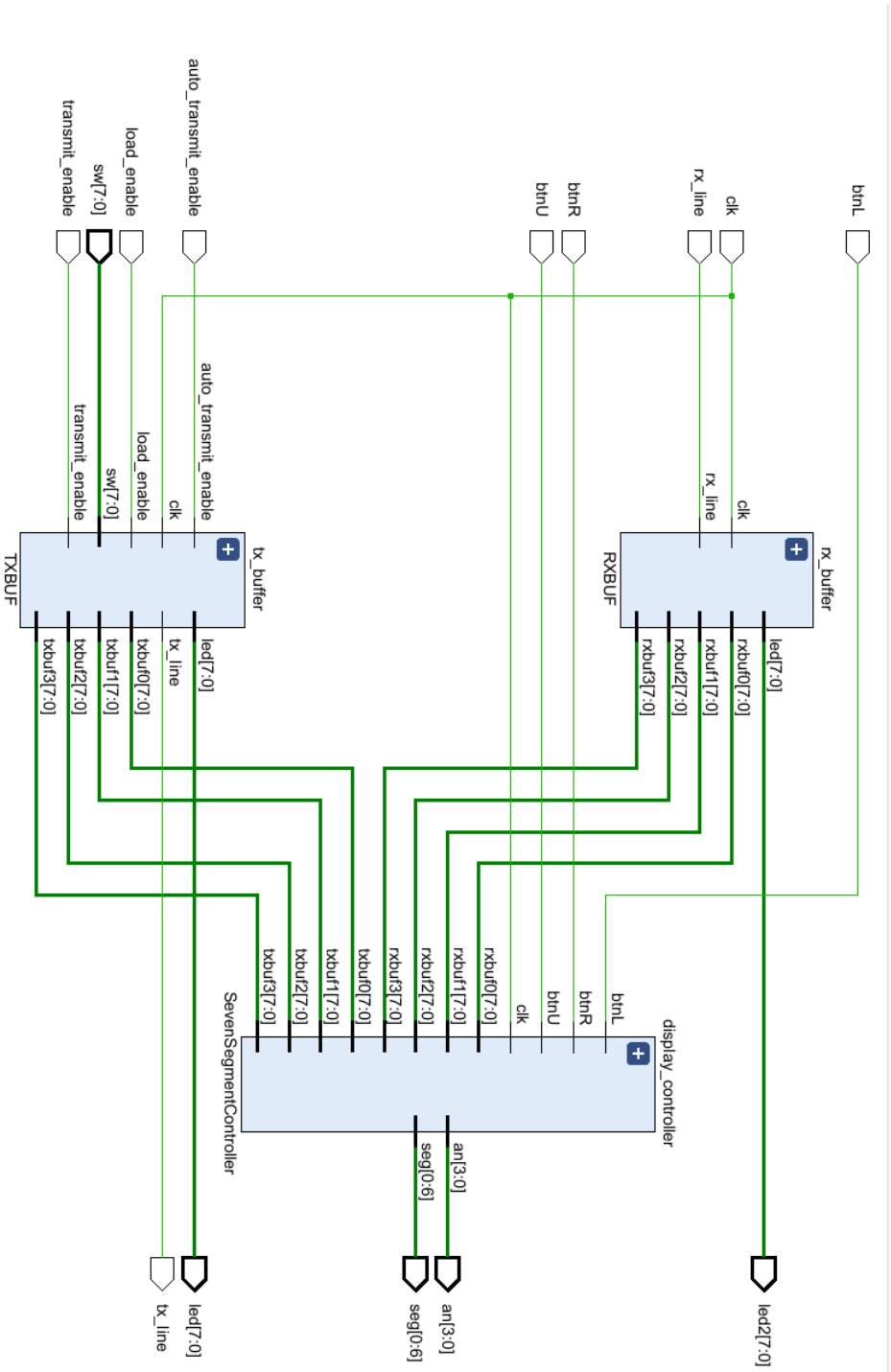
6. **HexTo7Segment**

   Takes the 8-bit data in the buffer to be displayed at that moment, and converts the hexadecimal values into 7-segment display codes. By doing so, the two rightmost digits to be displayed on the 7-segment display are converted to the appropriate values.

7. **baudClock**

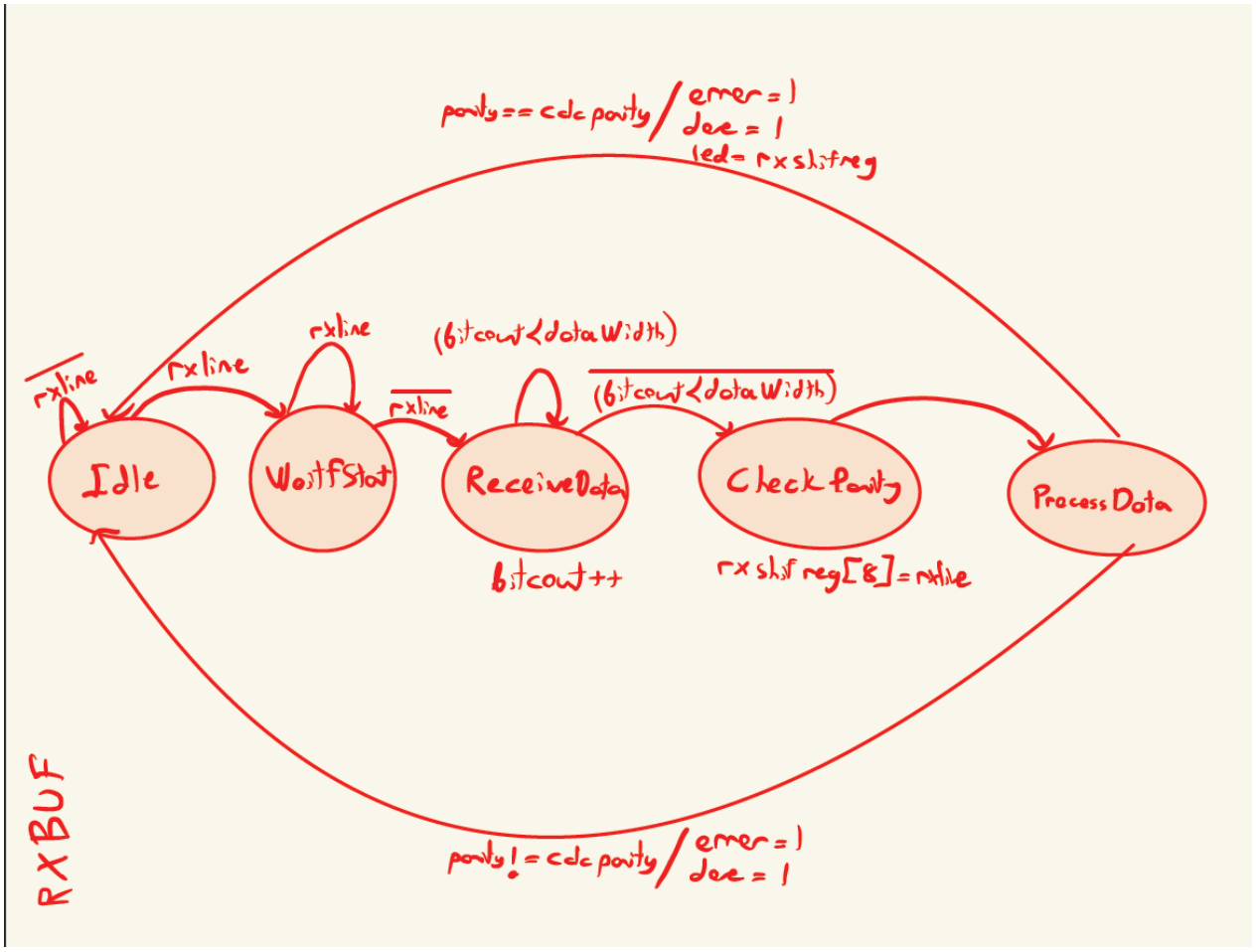
   This module generates a slower clock signal derived from the main system’s clock to control the timing of UART transmissions depending on the desired baud rate.

RTL schematics for UART TX, RX, memory arrays, 7-segment display, continuous transfer:

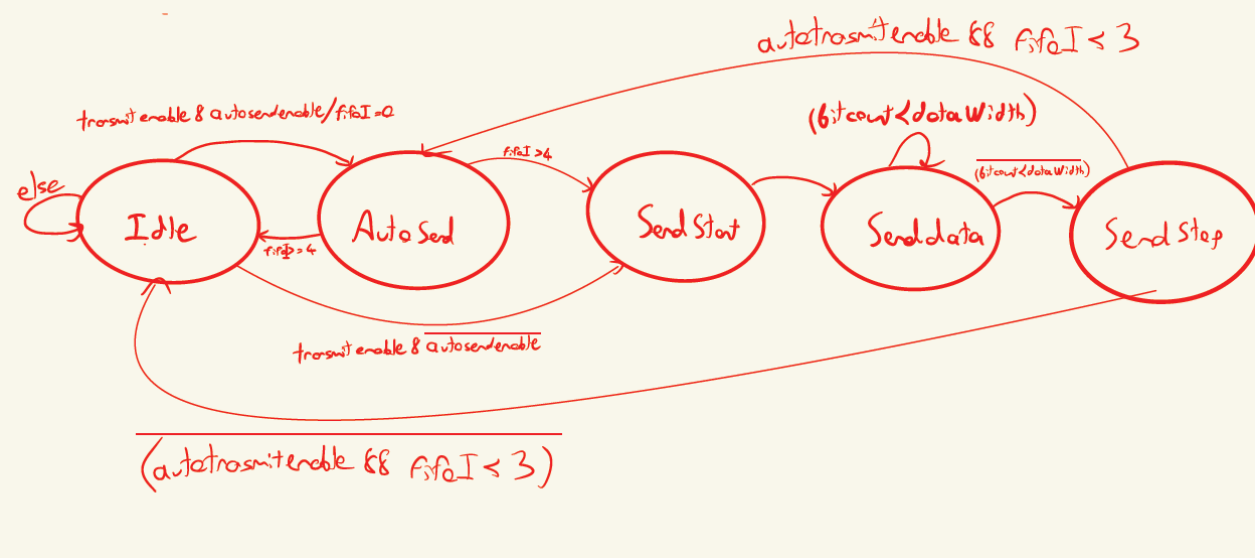


State diagram for UART TX, RX, 7-segment display, continuous transfer:

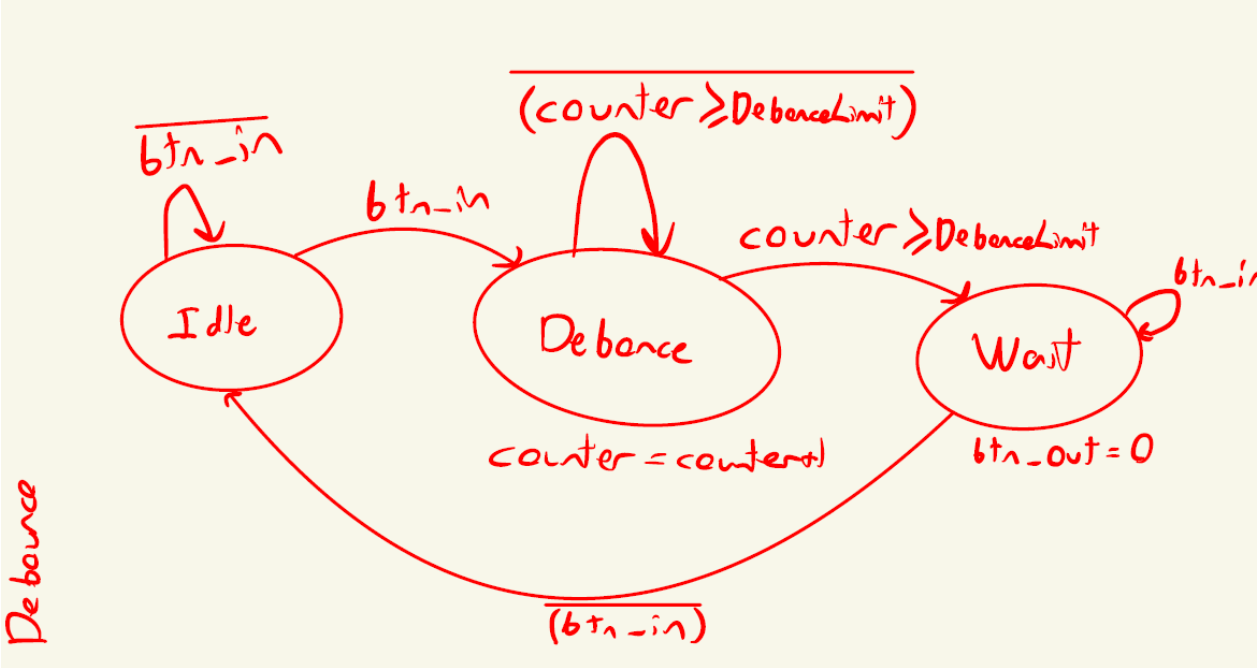
RXBUF:



TXBUF:

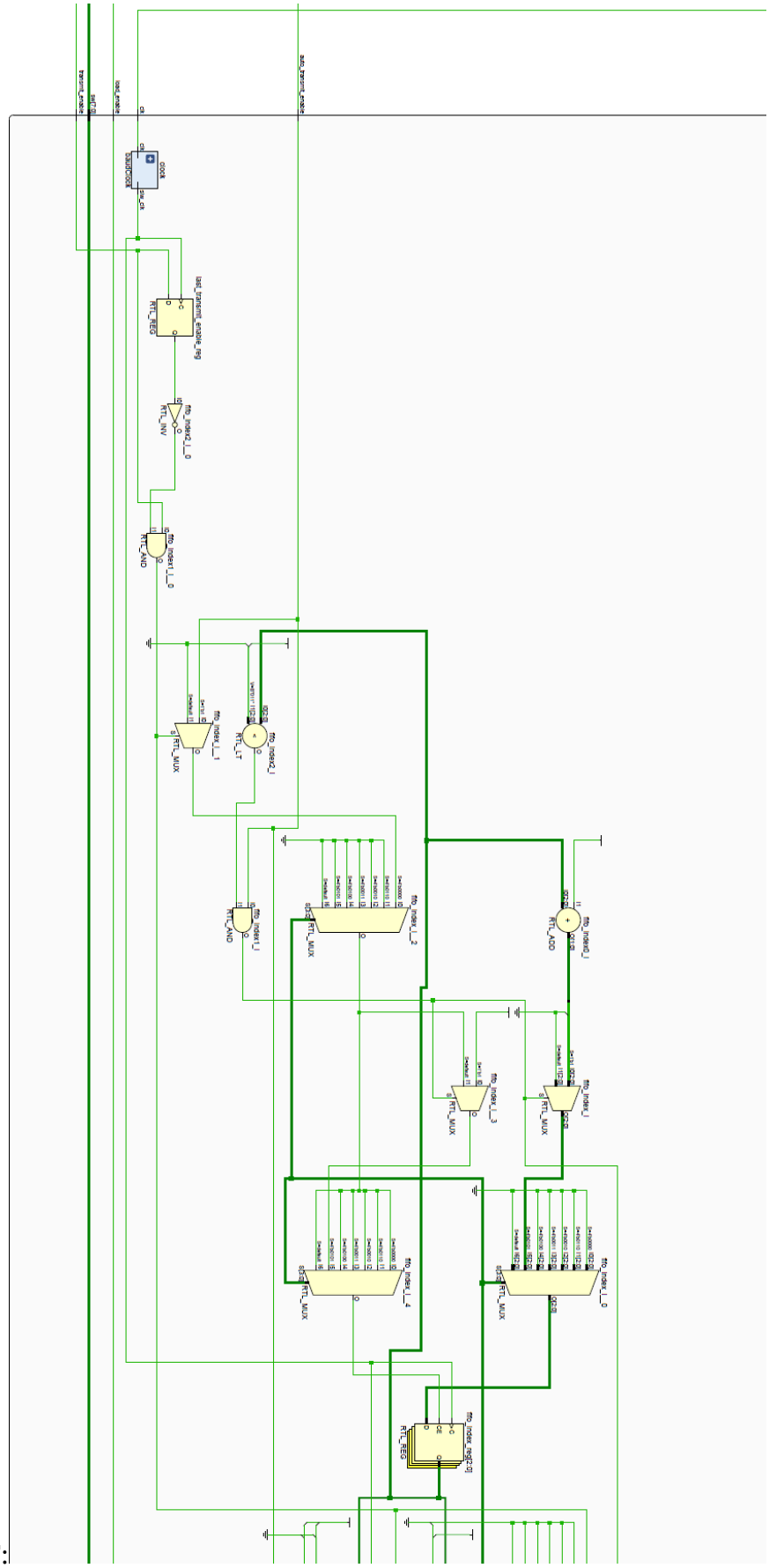


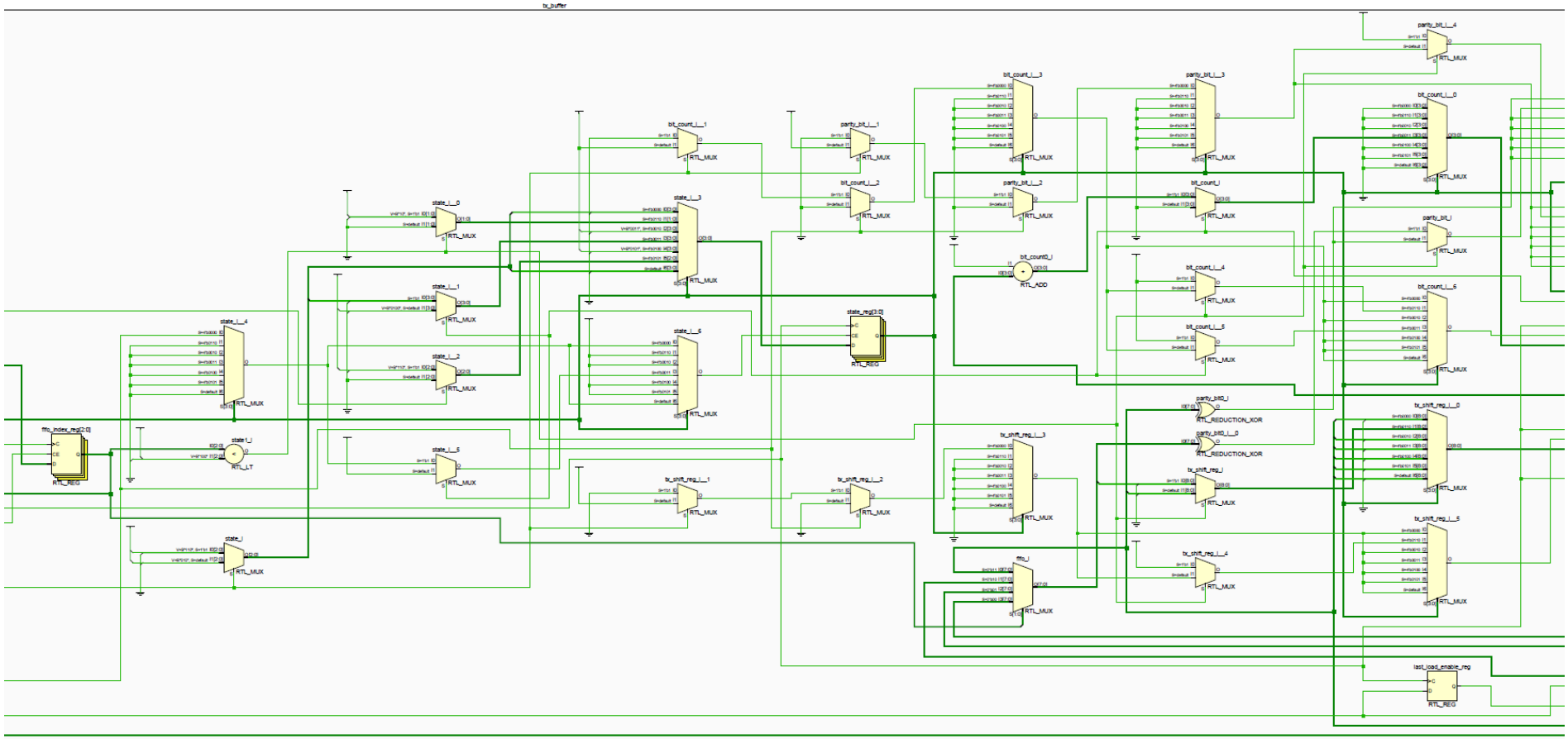
Debouncer:

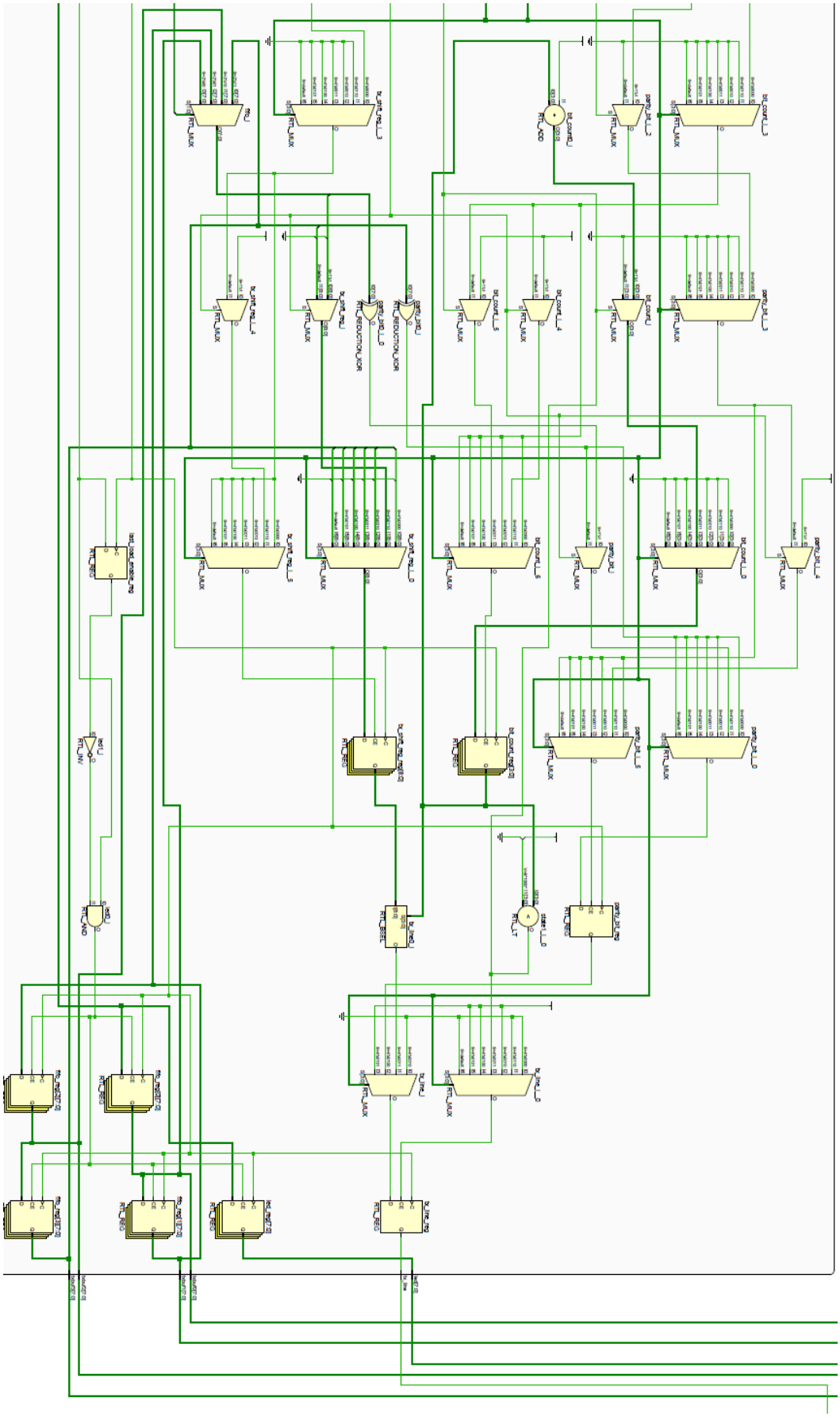


Block diagram of each module you implement:

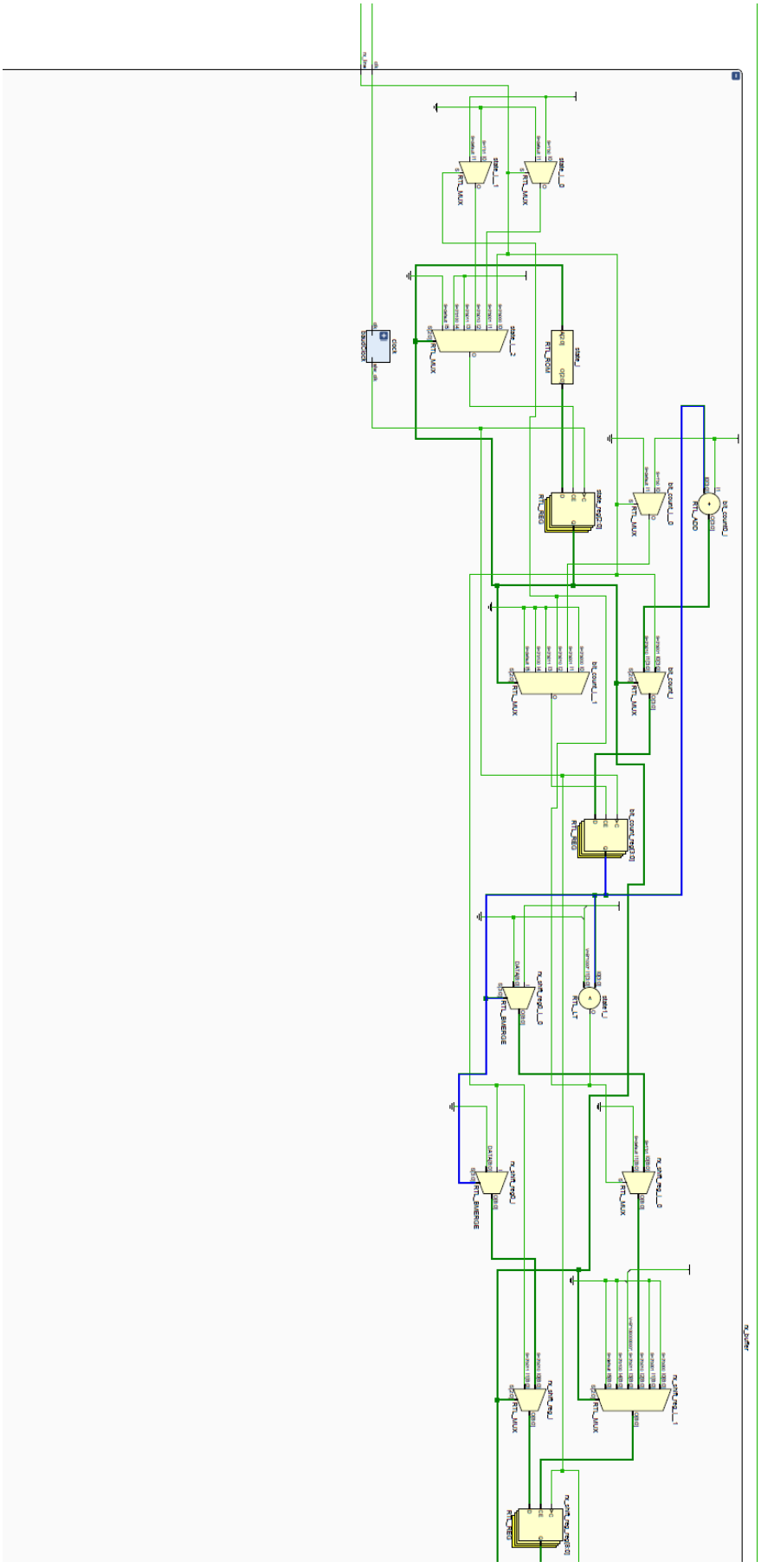
TXBUF:

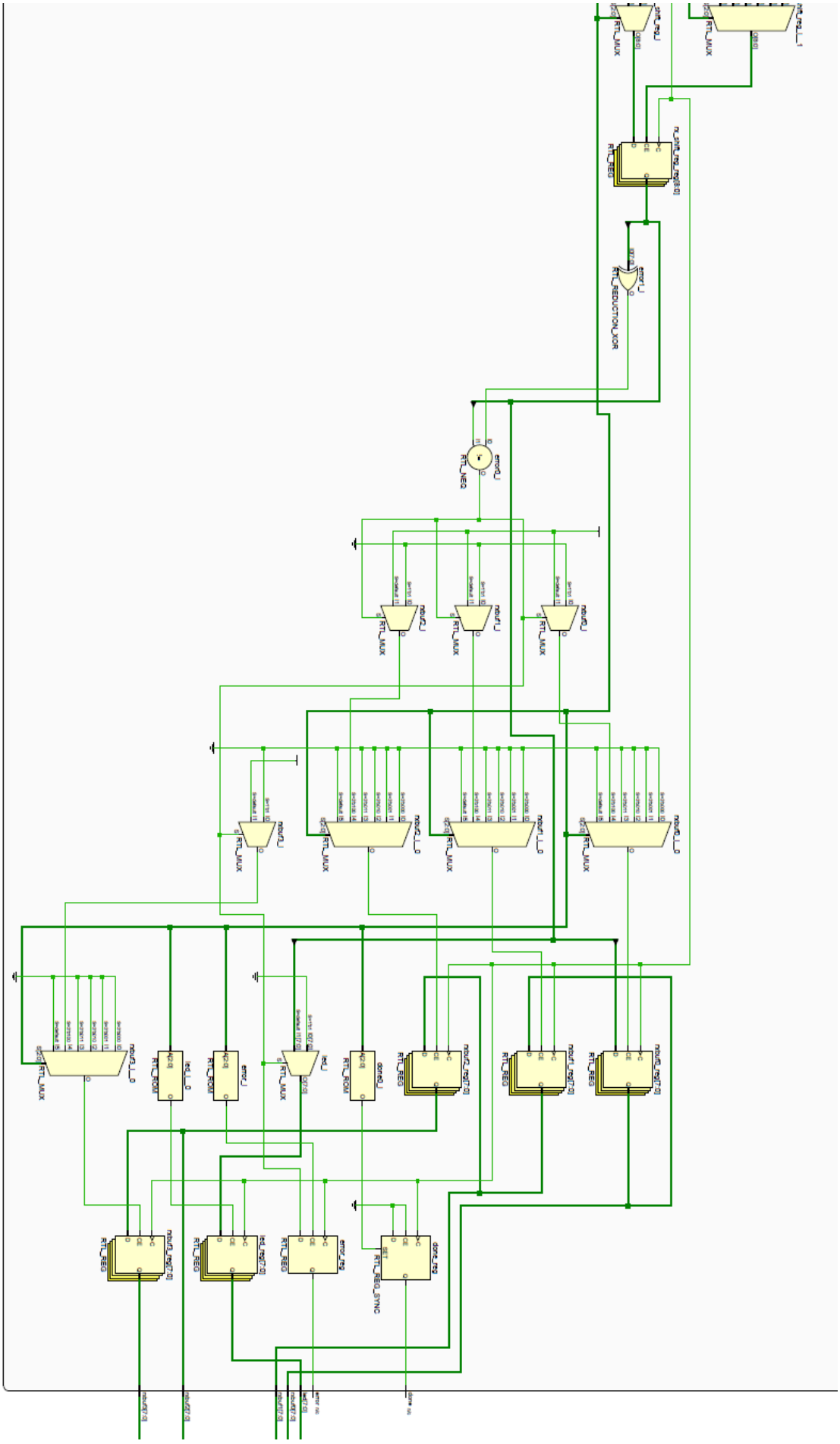



tx_buffer
RTL_MUX
RTL_REG
RTL_LT
RTL_ADD
RTL_REDUCTION_XOR

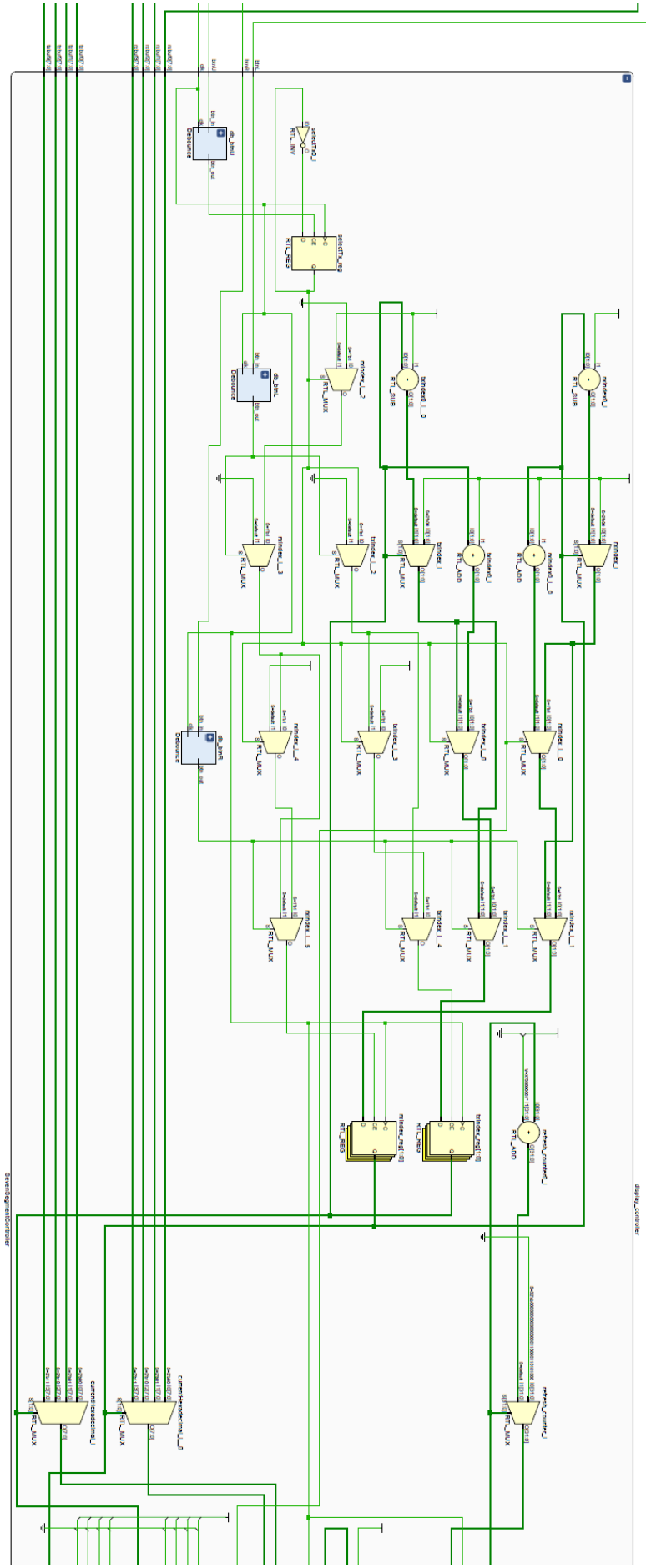


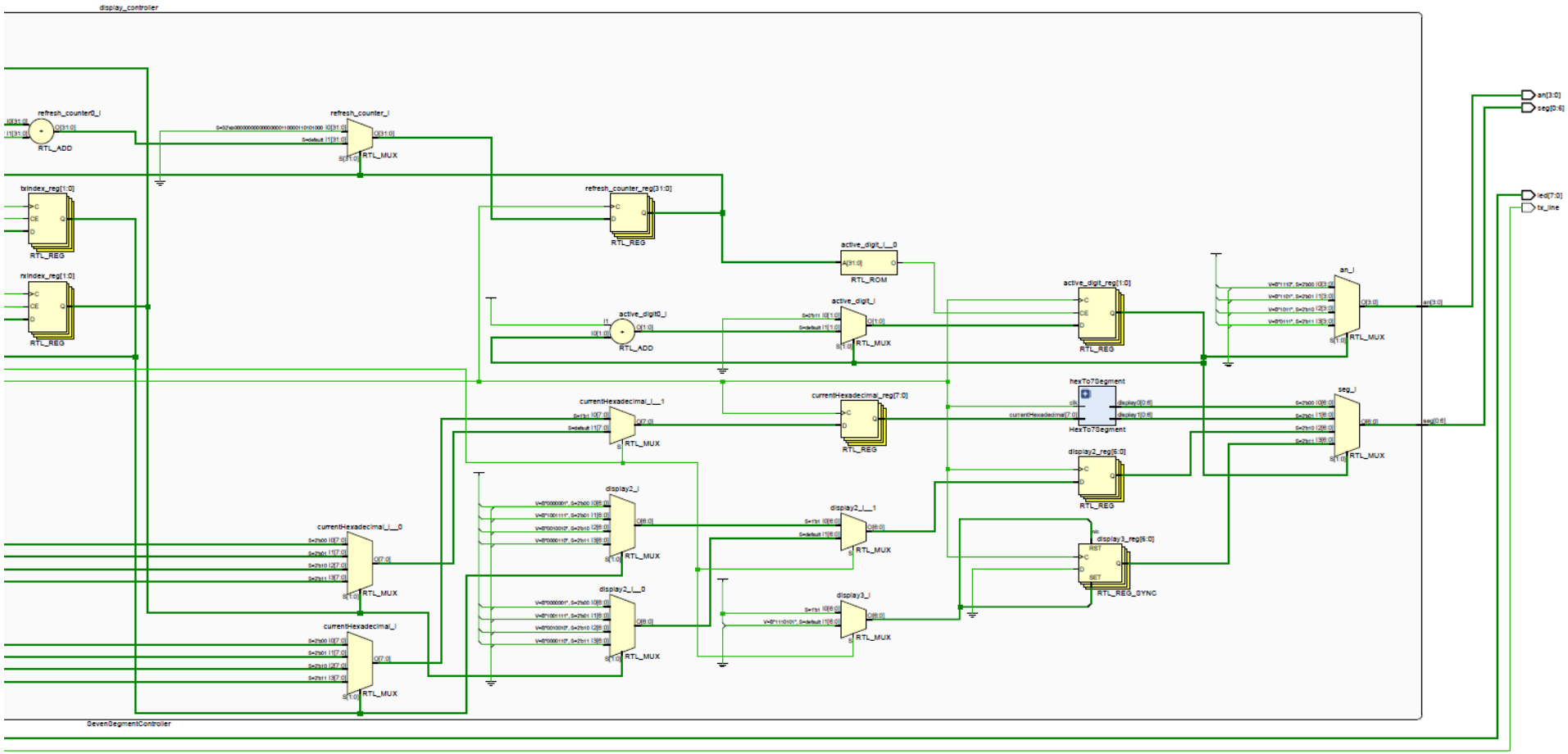
RXBUF:





SevenSegmentController:




display_controller
refresh_counter0_i
RTL_ADD
refresh_counter_i
RTL_MUX
txindex_reg[1:0]
RTL_REG
rxindex_reg[1:0]
refresh_counter_reg[31:0]
active_digit_i__0
RTL_ROM
active_digit0_i
active_digit_i
active_digit_reg[1:0]
an_i
currentHexadecimal_i__1
currentHexadecimal_reg[7:0]
hexTo7Segment
HexTo7Segment
seg_i
display2_reg[6:0]
currentHexadecimal_i__0
display2_i
display2_i__1
display3_reg[6:0]
RTL_REG_SYNC
currentHexadecimal_i
display2_i__0
display3_i
SevenSegmentController
an[3:0]
seg[0:6]
led[7:0]
tx_line

Appendix:

Constraints:

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
# create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
```

```
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {auto_transmit_enable}]

        set_property IOSTANDARD LVCMOS33 [get_ports {auto_transmit_enable}]

# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
```

```
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]

set_property PACKAGE_PIN V13 [get_ports {led2[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[0]}]
set_property PACKAGE_PIN V3 [get_ports {led2[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[1]}]
set_property PACKAGE_PIN W3 [get_ports {led2[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[2]}]
set_property PACKAGE_PIN U3 [get_ports {led2[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[3]}]
set_property PACKAGE_PIN P3 [get_ports {led2[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[4]}]
set_property PACKAGE_PIN N3 [get_ports {led2[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[5]}]
set_property PACKAGE_PIN P1 [get_ports {led2[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led2[6]}]
set_property PACKAGE_PIN L1 [get_ports {led2[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led2[7]}]


#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
```

```
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN V7 [get_ports dp]
        set_property IOSTANDARD LVCMOS33 [get_ports dp]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]

#Buttons
set_property PACKAGE_PIN U18 [get_ports transmit_enable]
        set_property IOSTANDARD LVCMOS33 [get_ports transmit_enable]
set_property PACKAGE_PIN T18 [get_ports btnU]
        set_property IOSTANDARD LVCMOS33 [get_ports btnU]
set_property PACKAGE_PIN W19 [get_ports btnL]
        set_property IOSTANDARD LVCMOS33 [get_ports btnL]
```

```
set_property PACKAGE_PIN T17 [get_ports btnR]
        set_property IOSTANDARD LVCMOS33 [get_ports btnR]
set_property PACKAGE_PIN U17 [get_ports load_enable]
        set_property IOSTANDARD LVCMOS33 [get_ports load_enable]


#Pmod Header JA
#Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {tx_line}]
        set_property IOSTANDARD LVCMOS33 [get_ports {tx_line}]
#Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {rx_line}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rx_line}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
#Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
#Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
#Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
```

```
#Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]
#Pmod Header JB
#Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports {JB[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[0]}]
#Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports {JB[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[1]}]
#Sch name = JB3
set_property PACKAGE_PIN B15 [get_ports {JB[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[2]}] ##Sch name = JB4
set_property PACKAGE_PIN B16 [get_ports {JB[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[3]}]
#Sch name = JB7
set_property PACKAGE_PIN A15 [get_ports {JB[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[4]}]
#Sch name = JB8
set_property PACKAGE_PIN A17 [get_ports {JB[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[5]}]
#Sch name = JB9
set_property PACKAGE_PIN C15 [get_ports {JB[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[6]}]
#Sch name = JB10
set_property PACKAGE_PIN C16 [get_ports {JB[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JB[7]}]
```

```
#Pmod Header JC
#Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {JC[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[0]}]
#Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports {JC[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[1]}]
#Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {JC[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[2]}]
#Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports {JC[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[3]}]
#Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports {JC[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[4]}]
#Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports {JC[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[5]}]
#Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports {JC[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[6]}]
#Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports {JC[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {JC[7]}]

#Pmod Header JXADC
#Sch name = XA1_P
```

```
set_property PACKAGE_PIN J3 [get_ports {JXADC[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[0]}]
#Sch name = XA2_P
set_property PACKAGE_PIN L3 [get_ports {JXADC[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[1]}]
#Sch name = XA3_P
set_property PACKAGE_PIN M2 [get_ports {JXADC[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[2]}]
#Sch name = XA4_P
set_property PACKAGE_PIN N2 [get_ports {JXADC[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[3]}]
#Sch name = XA1_N
set_property PACKAGE_PIN K3 [get_ports {JXADC[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[4]}]
#Sch name = XA2_N
set_property PACKAGE_PIN M3 [get_ports {JXADC[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[5]}]
#Sch name = XA3_N
set_property PACKAGE_PIN M1 [get_ports {JXADC[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[6]}]
#Sch name = XA4_N
set_property PACKAGE_PIN N1 [get_ports {JXADC[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[7]}]
#USB-RS232 Interface
set_property PACKAGE_PIN B18 [get_ports RsRx]
        set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
set_property PACKAGE_PIN A18 [get_ports RsTx]
        set_property IOSTANDARD LVCMOS33 [get_ports RsTx]
```

UART:

```
`timescale 1ns / 1ps

module UART
 #(parameter DATA_WIDTH = 8,BAUD_RATE  = 115200)
(
    input logic clk,
    input logic rx_line,
    input logic transmit_enable,

    input logic auto_transmit_enable, // sw[15]

    input logic load_enable,
    input logic btnL,
    input logic btnR,
    input logic btnU,
    input logic [DATA_WIDTH-1:0] sw,

    output logic tx_line,
    output logic [7:0] led,
    output logic [7:0] led2,
    output logic [0:6] seg,
    output logic [3:0] an
    );

    logic [DATA_WIDTH-1:0] txbuf0, txbuf1, txbuf2, txbuf3;
    logic [DATA_WIDTH-1:0] rxbuf0, rxbuf1, rxbuf2, rxbuf3;
```

```
logic tx_busy, rx_ready;



TXBUF #(
    .DATA_WIDTH(DATA_WIDTH),
    .BAUD_RATE(BAUD_RATE)
) tx_buffer (
    .clk(clk),
    .transmit_enable(transmit_enable),
    .auto_transmit_enable(auto_transmit_enable),
    .load_enable(load_enable),
    .sw(sw),
    .txbuf0(txbuf0),
    .txbuf1(txbuf1),
    .txbuf2(txbuf2),
    .txbuf3(txbuf3),
    .led(led),
    .tx_line(tx_line)
);

RXBUF #(
    .DATA_WIDTH(DATA_WIDTH),
    .BAUD_RATE(BAUD_RATE)
) rx_buffer (
    .clk(clk),
    .rx_line(rx_line),
    .rxbuf0(rxbuf0),
    .rxbuf1(rxbuf1),
```

```
        .rxbuf2(rxbuf2),
        .rxbuf3(rxbuf3),
        .led(led2),
        .error(rx_ready)
    );

    SevenSegmentController #(
        .DATA_WIDTH(DATA_WIDTH)
    ) display_controller (
        .clk(clk),
        .txbuf0(txbuf0),
        .txbuf1(txbuf1),
        .txbuf2(txbuf2),
        .txbuf3(txbuf3),
        .rxbuf0(rxbuf0),
        .rxbuf1(rxbuf1),
        .rxbuf2(rxbuf2),
        .rxbuf3(rxbuf3),
        .btnL(btnL),
        .btnR(btnR),
        .btnU(btnU),
        .seg(seg),
        .an(an)
    );

endmodule
```

TXBUF:

```
`timescale 1ns / 1ps
```

```
module TXBUF
#(parameter DATA_WIDTH = 8, BAUD_RATE = 115200)
(
    input logic clk,
    input logic transmit_enable, // BTNC
    input logic load_enable, // BTND
    input logic auto_transmit_enable, // sw[15]
    input logic [DATA_WIDTH-1:0] sw,

    output logic [DATA_WIDTH-1:0] txbuf0,
    output logic [DATA_WIDTH-1:0] txbuf1,
    output logic [DATA_WIDTH-1:0] txbuf2,
    output logic [DATA_WIDTH-1:0] txbuf3,

    output logic [DATA_WIDTH-1:0] led,
    output logic tx_line
);

    logic slw_clk;
    baudClock clock(
        .clk(clk),
        .slw_clk(slw_clk)
    );

    typedef enum logic[3:0] {
        IDLE,
        LOAD,
```

```
    SEND_START_BIT,
    SEND_DATA,
    SEND_PARITY,
    SEND_STOP_BIT,
    AUTO_SEND_NEXT
} state_t;

state_t state = IDLE;
logic [DATA_WIDTH:0] tx_shift_reg;
logic [3:0] bit_count = 0;
logic parity_bit = 0;
logic last_load_enable;
logic last_transmit_enable;
logic [DATA_WIDTH-1:0] fifo[3:0];

logic [2:0] fifo_index = 3'b100;
logic last_auto_transmit_enable;



always_ff @(posedge slw_clk) begin
    if (load_enable && !last_load_enable) begin
        fifo[3] <= fifo[2];
        fifo[2] <= fifo[1];
        fifo[1] <= fifo[0];
        fifo[0] <= sw;
        led <= sw;
    end
    last_load_enable <= load_enable;
```

```
end

assign txbuf0 = fifo[0];
assign txbuf1 = fifo[1];
assign txbuf2 = fifo[2];
assign txbuf3 = fifo[3];

always_ff @(posedge slw_clk) begin
    case (state)
        IDLE: begin
            if (transmit_enable && !last_transmit_enable) begin
                if (auto_transmit_enable) begin
                    fifo_index = 0;
                    state = AUTO_SEND_NEXT;
                end else begin
                    tx_shift_reg = {1'b0, fifo[3]};
                    parity_bit = ^fifo[3];
                    bit_count = 0;
                    state = SEND_START_BIT;
                end
            end

        end
        AUTO_SEND_NEXT: begin
            if (fifo_index < 4) begin
                tx_shift_reg <= {1'b0, fifo[fifo_index]};
                parity_bit <= ^fifo[fifo_index];
                bit_count <= 0;
```

```
            state <= SEND_START_BIT;
        end else begin
            state <= IDLE;
        end
    end
    SEND_START_BIT: begin
        tx_line <= 0;
        state <= SEND_DATA;
    end
    SEND_DATA: begin
        if (bit_count < DATA_WIDTH) begin
            tx_line <= tx_shift_reg[bit_count];
            bit_count <= bit_count + 1;
        end else begin
            state <= SEND_PARITY;
        end
    end
    SEND_PARITY: begin
        tx_line <= parity_bit;
        state <= SEND_STOP_BIT;
    end
    SEND_STOP_BIT: begin
        tx_line <= 1;
        if (auto_transmit_enable && (fifo_index < 3)) begin
            fifo_index <= fifo_index + 1;
            state <= AUTO_SEND_NEXT;
        end else begin
            state <= IDLE;
```

```
            end
        end
    endcase
    last_transmit_enable <= transmit_enable;
    last_auto_transmit_enable <= auto_transmit_enable;
end

endmodule
```

RXBUF:

```
`timescale 1ns / 1ps

module RXBUF
#(parameter DATA_WIDTH = 8, BAUD_RATE  = 115200)
(
    input logic clk,
    input logic rx_line, // to receive data

    output logic error,
    output logic [DATA_WIDTH-1:0] led, // left 8 digit
    output logic [DATA_WIDTH-1:0] rxbuf3,
    output logic [DATA_WIDTH-1:0] rxbuf2,
    output logic [DATA_WIDTH-1:0] rxbuf1,
    output logic [DATA_WIDTH-1:0] rxbuf0,
    output logic done
    );

    logic slw_clk;
    baudClock clock(
```

```
.clk(clk),
.slw_clk(slw_clk)
);

typedef enum logic[2:0] {
   IDLE,
   WAIT_FOR_START,
   RECEIVE_DATA,
   CHECK_PARITY,
   PROCESS_DATA
} state_t;

state_t state = IDLE;
logic [DATA_WIDTH:0] rx_shift_reg = 0;
logic [3:0] bit_count =  4'b0000;
logic calculated_parity = 0;

always_ff @(posedge slw_clk) begin
   case (state)
      IDLE: begin
         if (rx_line == 1)
            state <= WAIT_FOR_START;
      end
      WAIT_FOR_START: begin
         if (rx_line == 0) begin // Start bit
            state <= RECEIVE_DATA;
            bit_count <= 0;
         end
```

```
end
RECEIVE_DATA: begin
    if (bit_count < DATA_WIDTH) begin
        rx_shift_reg[bit_count] <= rx_line;
        bit_count <= bit_count + 1;
    end else begin
        state <= CHECK_PARITY;
    end
end
CHECK_PARITY: begin
    rx_shift_reg[DATA_WIDTH] <= rx_line;
    state <= PROCESS_DATA;
end
PROCESS_DATA: begin
    calculated_parity = ^rx_shift_reg[DATA_WIDTH-1:0];
    if (calculated_parity != rx_shift_reg[DATA_WIDTH]) begin
        error <= 1;
        done <= 1;
        led <= 8'b00000000; // error maybe? dont know how to show
    end else begin
        error <= 0;
        done <= 1;
        rxbuf3 <= rxbuf2;
        rxbuf2 <= rxbuf1;
        rxbuf1 <= rxbuf0;
        rxbuf0 <= rx_shift_reg[DATA_WIDTH-1:0];
        led <= rx_shift_reg[DATA_WIDTH-1:0];
    end
```

```
            state <= IDLE;
        end
    endcase
end

endmodule
```

SevenSegmentController:

```
`timescale 1ns / 1ps
module SevenSegmentController
#(parameter DATA_WIDTH = 8)
(
    input logic clk,
    input logic [7:0] txbuf0,
    input logic [7:0] txbuf1,
    input logic [7:0] txbuf2,
    input logic [7:0] txbuf3,
    input logic [7:0] rxbuf0,
    input logic [7:0] rxbuf1,
    input logic [7:0] rxbuf2,
    input logic [7:0] rxbuf3,
    input logic btnL, // left
    input logic btnR, // right
    input logic btnU, // switch between TXBUF and RXBUF

    output logic [0:6] seg,
    output logic [3:0] an
    );
```

```
logic selectTx;  // 1 for TXBUF, 0 for RXBUF
logic [1:0] txIndex, rxIndex;  // Current indices for TX and RX buffers

initial begin
    selectTx = 0;
    txIndex = 2'b00;
    rxIndex = 2'b00;
end

logic btnLD, btnRD, btnUD;


Debounce db_btnL (
    .clk(clk),
    .btn_in(btnL),
    .btn_out(btnLD)
);

Debounce db_btnR (
    .clk(clk),
    .btn_in(btnR),
    .btn_out(btnRD)
);

Debounce db_btnU (
    .clk(clk),
    .btn_in(btnU),
```

```
    .btn_out(btnUD)
);

always_ff @(posedge clk) begin
    if (btnLD) begin
        if (selectTx) begin
            if (txIndex == 0) txIndex <= 3;
            else txIndex <= txIndex - 1;
        end else begin
            if (rxIndex == 0) rxIndex <= 3;
            else rxIndex <= rxIndex - 1;
        end
    end
    if (btnRD) begin
        if (selectTx) begin
            txIndex <= (txIndex + 1) % 4;
        end else begin
            rxIndex <= (rxIndex + 1) % 4;
        end
    end
    if (btnUD) begin
        selectTx <= ~selectTx;
    end
end



/*logic btnLdebouncer = 0;
logic btnRdebouncer = 0;
```

```
logic btnUdebouncer = 0;
always_ff @(posedge clk) begin
    if (btnL & ~btnLdebouncer) begin
        if (selectTx) begin
            if (txIndex == 0) txIndex <= 3;
            else txIndex <= txIndex - 1;
        end else begin
            if (rxIndex == 0) rxIndex <= 3;
            else rxIndex <= rxIndex - 1;
        end
        btnLdebouncer <= 1;
    end
    else if (~ btnL)begin
        btnLdebouncer <= 0;
    end
    if (btnR & ~btnRdebouncer) begin
        if (selectTx) begin
            txIndex <= (txIndex + 1) % 4;
        end else begin
            rxIndex <= (rxIndex + 1) % 4;
        end
        btnRdebouncer <=1;
    end else if(~btnR)begin
        btnRdebouncer <= 0;
    end
    if (btnU & ~btnUdebouncer) begin
        selectTx <= ~selectTx;
        btnUdebouncer <= 1;
```

```
    end
    else if(~btnU)begin
        btnUdebouncer <= 0;
    end
end*/

logic [6:0] display3,display2,display1, display0;// 3 is leftmost
logic [7:0] currentHexadecimal;

always @(posedge clk) begin
    if(selectTx)begin
        display3 = 7'b1110000; // display 3 is t
        case(txIndex)
            2'b00: begin
                display2 = 7'b0000001;  // '0'
                currentHexadecimal = txbuf0;
            end
            2'b01: begin
               display2 = 7'b1001111;  // '1'
               currentHexadecimal = txbuf1;
            end
            2'b10: begin
                display2 = 7'b0010010;  // '2'
                currentHexadecimal = txbuf2;
            end
            2'b11: begin
                display2 = 7'b0000110;  // '3'
                currentHexadecimal = txbuf3;
```

```
            end
        endcase

    end else begin
        display3 = 7'b1111010; // display 3 is r
        case(rxIndex)
            2'b00: begin
                display2 = 7'b0000001;  // '0'
                currentHexadecimal = rxbuf0;
            end
            2'b01: begin
                display2 = 7'b1001111;  // '1'
                currentHexadecimal = rxbuf1;
            end
            2'b10: begin
                display2 = 7'b0010010;  // '2'
                currentHexadecimal = rxbuf2;
            end
            2'b11: begin
                display2 = 7'b0000110;  // '3'
                currentHexadecimal = rxbuf3;
            end
        endcase

    end
end

HexTo7Segment hexTo7Segment (
```

```
        .clk(clk),
        .currentHexadecimal(currentHexadecimal),
        .display1(display1),
        .display0(display0)
    );

    reg [1:0] active_digit = 0;
    integer refresh_counter = 0;
    always @(posedge clk) begin
        refresh_counter <= refresh_counter + 1;
        if (refresh_counter == 25000) begin
            refresh_counter <= 0;
            active_digit <= active_digit + 1;
            if (active_digit == 3) active_digit <= 0;
        end
    end

    always @(*) begin
        case (active_digit)
            2'b00: begin seg <= display0; an <= 4'b1110; end
            2'b01: begin seg <= display1; an <= 4'b1101; end
            2'b10: begin seg <= display2; an <= 4'b1011; end
            2'b11: begin seg <= display3; an <= 4'b0111; end
        endcase
    end

endmodule
```

Debounce:

```
`timescale 1ns / 1ps
module Debounce(
    input logic clk,
    input logic btn_in,
    output logic btn_out
);
    parameter DEBOUNCE_LIMIT = 21234;
    integer counter = 0;

    typedef enum logic [1:0] {IDLE, DEBOUNCE, WAIT} stateType;
        stateType [1:0] state = IDLE;

    always_ff @(posedge clk) begin
       case (state)
        IDLE: begin
            btn_out <= 0;
            if(btn_in) begin
                state <= DEBOUNCE;
            end
            else if (!btn_in) begin
                state <= IDLE;
            end
        end

        DEBOUNCE: begin
        counter <= counter+1;
        if(counter >= DEBOUNCE_LIMIT)begin
```

```
            counter <= 0;
            btn_out <= 1;
            state <= WAIT;
        end
        end

        WAIT: begin
            btn_out <= 0;
            if(btn_in) begin
                btn_out <= 0;
            end
            else begin
                state <= IDLE;
            end
        end
        endcase

    end
endmodule
```

HexTo7Segment:

```
`timescale 1ns / 1ps

module HexTo7Segment
(
    input logic clk,
    input logic [7:0] currentHexadecimal,
```

```
    output logic [0:6] display1,
    output logic [0:6] display0
);

    function [0:6] decodeHexTo7Segment(input [3:0] hexDigit);
        case (hexDigit)
        4'h0: decodeHexTo7Segment = 7'b0000001; // 0
        4'h1: decodeHexTo7Segment = 7'b1001111; // 1
        4'h2: decodeHexTo7Segment = 7'b0010010; // 2
        4'h3: decodeHexTo7Segment = 7'b0000110; // 3
        4'h4: decodeHexTo7Segment = 7'b1001100; // 4
        4'h5: decodeHexTo7Segment = 7'b0100100; // 5
        4'h6: decodeHexTo7Segment = 7'b0100000; // 6
        4'h7: decodeHexTo7Segment = 7'b0001111; // 7
        4'h8: decodeHexTo7Segment = 7'b0000000; // 8
        4'h9: decodeHexTo7Segment = 7'b0000100; // 9
        4'hA: decodeHexTo7Segment = 7'b0001000; // A
        4'hB: decodeHexTo7Segment = 7'b1100000; // B
        4'hC: decodeHexTo7Segment = 7'b0110001; // C
        4'hD: decodeHexTo7Segment = 7'b1000010; // D
        4'hE: decodeHexTo7Segment = 7'b0110000; // E
        4'hF: decodeHexTo7Segment = 7'b0111000; // F
        default: decodeHexTo7Segment = 7'b1111111; // blank
        endcase
    endfunction

    always_ff @(posedge clk) begin
        display1 <= decodeHexTo7Segment(currentHexadecimal[7:4]);
```

```
        display0 <= decodeHexTo7Segment(currentHexadecimal[3:0]);
    end

endmodule
```

baudClock:

```
`timescale 1ns / 1ps

module baudClock
#( parameter BAUD_RATE = 115200, parameter BAUD_FREQ = 100000000/BAUD_RATE)
(
    input logic clk,
    output logic slw_clk
    );

    logic [30:0] counter;

    always_ff @(posedge clk) begin
        if (counter == BAUD_FREQ) begin
            counter <= 0;
            slw_clk <= ~slw_clk;
        end else begin
            counter <= counter + 1;
        end
    end

endmodule
```