APPENDİX

```
`timescale 1ns / 1ps

module receiver#(parameter WIDTH = 8, DEPTH = 4,STOP_BIT =
2'b00,BAUD_RATE = 120, TOTAL_BITS = 10)
   (
   input logic clk,
   input logic reset,
   input logic RXserial,
   output logic switch,
   input logic idleHigh,
   output logic [DEPTH-1:0][WIDTH+1:0] RXBUF,
   output logic RXdone
   );

   //logic [DEPTH-1:0] [WIDTH+1:0]RXBUF; // the buffer that we load the data
   logic [24:0] counter; // counts the baud rate (21 is enough but allocate extra
memory)
   logic [3:0] bitcounter; // Counts the number of bits.
   logic [WIDTH+1:0] data; // This stores the received bits
   logic [1:0] switchstateCounter;

   typedef enum logic [1:0] {idlestate ,receivestate} statetype;
   statetype state;

   always_ff @(posedge clk)begin
      if(reset == 1'b1) begin
         counter <= 0;
         bitcounter <= 0;
         RXdone <= 1'b0;
         RXBUF <= 36'b0;
         state <= idlestate;
      end else begin
         case (state)
            idlestate: begin
               counter <= 0;
```

```
                bitcounter <= 0;
                RXdone <= 1'b0;
                if(idleHigh == 1'b0) begin// check whether or not the system turns off 0
bit.
                    state <= receivestate;
                end
            end

            receivestate: begin
                counter <= counter + 1'b1;
                if(counter == BAUD_RATE) begin
                    counter <= 0;
                    bitcounter <= bitcounter + 1;
                    data <= {RXserial, data[WIDTH+1:1]};
                    if(idleHigh == 1'b1) begin
                        RXdone <= 1'b1;
                        RXBUF <= {RXBUF[DEPTH-2:0],data};
                        state <= idlestate;
                    end
                end
            end
        endcase
    end
end
endmodule

`timescale 1ns / 1ps

module transmitter#(parameter WIDTH = 8, DEPTH = 4,STOP_BIT =
2'b00,BAUD_RATE = 120, TOTAL_BITS = 10)
    (
    input logic clk,
    input logic reset,
    output logic BTND,
    output logic BTNC,
    output logic switch,
    input logic [WIDTH-1:0]data,
    output logic TXdone,
    output logic idleHigh,
    output logic TXserial, // Now this TX serial gets 8 bits
    output logic [DEPTH-1:0][WIDTH+1:0]TXBUF
    );

     // the buffer that we load the data
```

```
   logic [24:0] counter; // counts the baud rate (21 is enough but allocate extra
memory)
   logic [3:0] bitcounter; // Counts the number of bits.
   logic [3:0] elementCounter;
   logic [WIDTH+1:0] discardedData;
   logic [1:0] switchstateCounter;

   typedef enum logic [1:0] {idlestate, switchstate,  transferstate} statetype;
   statetype state;

   always_ff @(posedge clk)begin
      if(reset == 1'b1) begin // Then we are in reset state which is idle state
         state <= idlestate;
         TXdone <= 1'b0;
         counter <= 0;
         bitcounter <= 0;
         TXBUF <= 36'b0;
         elementCounter <= 0;
         idleHigh <= 1'b1;
         TXserial <= 1'b0;
      end else begin
         // list the state here because I could not able to do that in always comb
statements.
         case (state)
         // I will have 3 states
            idlestate: begin
               TXdone <= 1'b0;
               TXserial <= 1'b0;
               idleHigh <= 1'b1;
               if(switch == 1'b1 && BTNC == 1'b1) begin
                  state <= switchstate;
               end
               if(BTND == 1'b1) begin
                  TXBUF <= {TXBUF[DEPTH-2:0],STOP_BIT,data};
               end
               if(BTNC == 1'b1) begin
               counter <= 0;
               bitcounter <= 0;
               idleHigh <= 1'b1;
               discardedData <= TXBUF[3];
               state <= transferstate;
               end
            end
```

```
        switchstate: begin
                idleHigh <= 1'b0;
                TXserial <= TXBUF[switchstateCounter][0];
                counter <= counter + 1'b1;
                if(counter == BAUD_RATE+1) begin // Then we can send one bit
                  counter <= 0;
                  bitcounter <= bitcounter +1;
                  TXBUF[switchstateCounter][8:0] <=
TXBUF[switchstateCounter][9:1];
                  if(bitcounter == TOTAL_BITS-1) begin
                    switchstateCounter <= switchstateCounter + 1;
                    TXBUF <= {TXBUF[DEPTH-2:0],10'b0};
                    if(switchstateCounter == DEPTH-1) begin
                      state <= idlestate;
                      TXdone <= 1'b1;
                      idleHigh <= 1'b1;
                      TXserial <= 1'b0;
                     end else begin
                      state <= switchstate;
                      bitcounter <= 0;
                    end
                  end
                end
              end


      // Third state where we send the datas
        transferstate: begin
          idleHigh <= 1'b0;
          TXserial <= discardedData[0];
          counter <= counter + 1'b1;
          if(counter == BAUD_RATE+1) begin // Then we can send one bit
            counter <= 0;
            bitcounter <= bitcounter +1;
            discardedData[8:0] <= discardedData[9:1];
            if(bitcounter == TOTAL_BITS-1) begin
              TXdone <= 1'b1;
              TXserial <= 1'b0;
              idleHigh <= 1'b1;
              state <= idlestate;
            end
          end
        end
      endcase
```

```
        end
    end

endmodule

`timescale 1ns / 1ps

module buttonSync(
    input logic bi,
    input logic reset,
    input logic clk,
    output logic bo
    );

    typedef enum logic [1:0] {S0,S1,S2} statetype;
    statetype state;
    reg [16:0] counter;

    always_ff @(posedge clk)begin
    if(reset == 1'b1)begin
        bo <= 1'b0;
        state <= S0;
    end else  begin
        case (state)
            S0: begin
                bo <= 1'b0;
                if(bi == 1'b1)begin
                    state <= S1;
                end else begin
                    state <= S0;
                end
            end

            S1: begin
                bo <= 1'b1;
                if(bi == 1'b0)begin
                        state <= S0;
                    end else begin
                        state <= S2;
                    end
            end

            S2: begin
                bo <= 1'b0;
            if(bi == 1'b1) begin
               state <= S2;
            end else begin
               state <= S0;
            end
         end
         default: begin
            bo <= 1'b0;
            if(bi == 1'b1)begin
               state <= S1;
            end
         end
      endcase
   end
   end
endmodule


`timescale 1ns / 1ps

module displaySecond #(parameter WIDTH = 8, DEPTH = 4)
   (
   input logic clk,
   input logic reset,
   output logic BTNR,
   output logic BTNL,
   output logic BTNU,
   output logic [DEPTH-1:0][WIDTH+1:0] TXBUF,
   output logic [DEPTH-1:0][WIDTH+1:0] RXBUF,
   output logic lastLed,
   output logic [1:0] stateLed
   );

   typedef enum logic [1:0] {S0,S1,S2,S3} statetype;
   statetype state, nextstate;

   reg [4:0] counter;
   reg [32:0] clkdiv;

always_ff @(posedge clk) begin
   if(reset) begin
      state <= S0;
      clkdiv <= 0;
      counter <= 0;
```

```
end else begin
      case (state)
   S0: begin
      lastLed <= 1'b0;
      stateLed <= 2'b01;
      if(BTNU == 1'b1) begin
         state <= S1;
      end
      if(BTNR || BTNL) begin
         state <= S2;
      end
   end
   S1: begin
      lastLed <= 1'b1;
      stateLed <= 2'b01;
      if(BTNU == 1'b1) begin
         state <= S0;
      end
      if (BTNR == 1'b1 || BTNL == 1'b1) begin
         state <= S3;
      end
   end

   S2: begin
      lastLed <= 1'b0;
      stateLed <= 2'b10;
      if(BTNU == 1'b1) begin
         state <= S1;
      end
      if (BTNR== 1'b1 || BTNL == 1'b1) begin
         state <= S0;
      end
   end

   S3: begin
      lastLed <= 1'b1;
      stateLed <= 2'b10;
      if(BTNU == 1'b1) begin
         state <= S0;
      end
      if (BTNR == 1'b1 || BTNL == 1'b1) begin
         state <= S1;
      end
   end
```

```
        default: state <= S0;
    endcase

    end
end

endmodule


module FourDigitDisplayController (
  input logic clk,
  output logic [3:0][9:0]TXBUF,
  output logic [3:0][9:0]RXBUF,
  output logic [6:0] segDisp,
  output logic [1:0] stateLed,
  output logic lastLed,
  output logic [3:0] an
);
    logic [32:0] clkdiv;
    logic [15:0] data;
always_comb begin
   if(stateLed == 2'b01 && lastLed == 1'b0) begin // First two element of TXBUF
    data <= {TXBUF[1][7:0],TXBUF[0][7:0]};
  end
  if(stateLed == 2'b10 && lastLed == 1'b0) begin // Last two of TXBUF
    data <= {TXBUF[3][7:0],TXBUF[2][7:0]};
  end
  if(stateLed == 2'b01 && lastLed == 1'b1) begin // First two of RXBUF
    data <= {RXBUF[1][7:0],RXBUF[0][7:0]};
  end
  if(stateLed == 2'b10 && lastLed == 1'b1) begin // Last Two of RXBUF
    data <= {RXBUF[3][7:0],RXBUF[2][7:0]};
  end
end
  always_ff @(posedge clk) begin
    if(clkdiv == 32'b11111111111111111) begin
        clkdiv <= 32'b0;
        case (an)
            4'b1110: begin // 2. segment
            case (data[7:4]) // 2. segment
                 4'b0000: segDisp = 7'b1000000;
                 4'b0001: segDisp = 7'b1111001;
                 4'b0010: segDisp = 7'b0100100;
                 4'b0011: segDisp = 7'b0110000;
```

```
          4'b0100: segDisp = 7'b0011001;
          4'b0101: segDisp = 7'b0010010;
          4'b0110: segDisp = 7'b0000010;
          4'b0111: segDisp = 7'b1111000;
          4'b1000: segDisp = 7'b0000000;
          4'b1001: segDisp = 7'b0011000;
          4'b1010: segDisp = 7'b0001000;
          4'b1011: segDisp = 7'b0000011;
          4'b1100: segDisp = 7'b1000110;
          4'b1101: segDisp = 7'b0100001;
          4'b1110: segDisp = 7'b0000110;
          4'b1111: segDisp = 7'b0001110;
          default: segDisp = 7'b1111111;
    endcase
    an <= 4'b1101;
end

4'b1101: begin // 3. segment
        case (data[11:8])
          4'b0000: segDisp = 7'b1000000;
          4'b0001: segDisp = 7'b1111001;
          4'b0010: segDisp = 7'b0100100;
          4'b0011: segDisp = 7'b0110000;
          4'b0100: segDisp = 7'b0011001;
          4'b0101: segDisp = 7'b0010010;
          4'b0110: segDisp = 7'b0000010;
          4'b0111: segDisp = 7'b1111000;
          4'b1000: segDisp = 7'b0000000;
          4'b1001: segDisp = 7'b0011000;
          4'b1010: segDisp = 7'b0001000;
          4'b1011: segDisp = 7'b0000011;
          4'b1100: segDisp = 7'b1000110;
          4'b1101: segDisp = 7'b0100001;
          4'b1110: segDisp = 7'b0000110;
          4'b1111: segDisp = 7'b0001110;
          default: segDisp = 7'b1111111;
        endcase
     an <= 4'b1011;
end

4'b1011: begin // 4. segment
         case (data[15:12])
          4'b0000: segDisp = 7'b1000000;
          4'b0001: segDisp = 7'b1111001;
```

```
            4'b0010: segDisp = 7'b0100100;
            4'b0011: segDisp = 7'b0110000;
            4'b0100: segDisp = 7'b0011001;
            4'b0101: segDisp = 7'b0010010;
            4'b0110: segDisp = 7'b0000010;
            4'b0111: segDisp = 7'b1111000;
            4'b1000: segDisp = 7'b0000000;
            4'b1001: segDisp = 7'b0011000;
            4'b1010: segDisp = 7'b0001000;
            4'b1011: segDisp = 7'b0000011;
            4'b1100: segDisp = 7'b1000110;
            4'b1101: segDisp = 7'b0100001;
            4'b1110: segDisp = 7'b0000110;
            4'b1111: segDisp = 7'b0001110;
            default: segDisp = 7'b1111111;
           endcase
        an <= 4'b0111;
    end

    4'b0111: begin // 1. segment
            case (data[3:0])
            4'b0000: segDisp = 7'b1000000;
            4'b0001: segDisp = 7'b1111001;
            4'b0010: segDisp = 7'b0100100;
            4'b0011: segDisp = 7'b0110000;
            4'b0100: segDisp = 7'b0011001;
            4'b0101: segDisp = 7'b0010010;
            4'b0110: segDisp = 7'b0000010;
            4'b0111: segDisp = 7'b1111000;
            4'b1000: segDisp = 7'b0000000;
            4'b1001: segDisp = 7'b0011000;
            4'b1010: segDisp = 7'b0001000;
            4'b1011: segDisp = 7'b0000011;
            4'b1100: segDisp = 7'b1000110;
            4'b1101: segDisp = 7'b0100001;
            4'b1110: segDisp = 7'b0000110;
            4'b1111: segDisp = 7'b0001110;
            default: segDisp = 7'b1111111;
           endcase
          an <= 4'b1110;
        end
        default: an <= 4'b1110;
endcase
end else begin
```

```
        clkdiv <= clkdiv + 1;
      end
    end
    endmodule


`timescale 1ns / 1ps

module uart#(parameter WIDTH = 8, DEPTH = 4)
    (
    // Transmitter
    input logic clk,
    input logic reset,
    input logic BTND,
    input logic switch,
    input logic BTNC,
    input logic BTNL,
    input logic BTNR,
    input logic BTNU,
    input logic [WIDTH-1:0] data,
    output logic [7:0] LEDA,
    output logic lastLed,
    output logic [1:0] stateLed,
    output logic [6:0] seg_outputs,
    output logic [3:0] an
    );

    // These 3 reg's are for transmitter.
    logic idleHigh;
    logic TXserial;
    logic RXdone;
    logic TXdone;
    logic [DEPTH-1:0][WIDTH+1:0] TXBUF;
    // Receiver
    logic [DEPTH-1:0][WIDTH+1:0] RXBUF;

    // Buttons
    reg BTNCout;
    reg BTNDout;
    reg BTNLout;
    reg BTNUout;
    reg BTNRout;

    initial begin
```

```
      TXBUF <= 36'b0;
      RXBUF <= 36'b0;
  end

   receiver uut_receiver (
  .clk(clk),
  .switch(switch),
  .reset(reset),
  .RXserial(TXserial),
  .idleHigh(idleHigh),
  .RXBUF(RXBUF),
  .RXdone(RXdone)
);

  buttonSync uut_buttonSync(
      .clk(clk),
      .reset(reset),
      .bi(BTNC),
      .bo(BTNCout)
  );

  buttonSync uut_buttonSync2(
      .clk(clk),
      .reset(reset),
      .bi(BTND),
      .bo(BTNDout)
  );

  buttonSync uut_buttonSync3 (
      .clk(clk),
      .reset(reset),
      .bi(BTNL),
      .bo(BTNLout)
  );
  buttonSync uut_buttonSync4 (
      .clk(clk),
      .reset(reset),
      .bi(BTNR),
      .bo(BTNRout)
  );

  buttonSync uut_buttonSync5 (
      .clk(clk),
      .reset(reset),
        .bi(BTNU),
        .bo(BTNUout)
    );

    transmitter uut_transmitter (
    .clk(clk),
    .reset(reset),
    .switch(switch),
    .BTND(BTNDout),
    .BTNC(BTNCout),
    .data(data),
    .TXdone(TXdone),
    .idleHigh(idleHigh),
    .TXBUF(TXBUF),
    .TXserial(TXserial)
  );

  displaySecond uut_stateController(
    .clk(clk),
    .reset(reset),
    .BTNL(BTNLout),
    .BTNR(BTNRout),
    .BTNU(BTNUout),
    .TXBUF(TXBUF),
    .RXBUF(RXBUF),
    .lastLed(lastLed),
    .stateLed(stateLed)
  );

  FourDigitDisplayController sevenSeg(
    .clk(clk),
    .TXBUF(TXBUF),
    .RXBUF(RXBUF),
    .stateLed(stateLed),
    .lastLed(lastLed),
    .an(an),
    .segDisp(seg_outputs)
  );

  assign LEDA = TXBUF[0][WIDTH-1:0];

endmodule
```

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level
signal names in the project

# 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 {data[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {data[0]}]
set_property PACKAGE_PIN V16 [get_ports {data[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {data[1]}]
set_property PACKAGE_PIN W16 [get_ports {data[2]}]

	set_property IOSTANDARD LVCMOS33 [get_ports {data[2]}]
set_property PACKAGE_PIN W17 [get_ports {data[3]}]

	set_property IOSTANDARD LVCMOS33 [get_ports {data[3]}]
set_property PACKAGE_PIN W15 [get_ports {data[4]}]

	set_property IOSTANDARD LVCMOS33 [get_ports {data[4]}]
set_property PACKAGE_PIN V15 [get_ports {data[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {data[5]}]
set_property PACKAGE_PIN W14 [get_ports {data[6]}]

	set_property IOSTANDARD LVCMOS33 [get_ports {data[6]}]
set_property PACKAGE_PIN W13 [get_ports {data[7]}]

	set_property IOSTANDARD LVCMOS33 [get_ports {data[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 {reset}]
        set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
set_property PACKAGE_PIN R2 [get_ports {switch}]
        set_property IOSTANDARD LVCMOS33 [get_ports {switch}]


# LEDs
set_property PACKAGE_PIN U16 [get_ports {LEDA[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[0]}]
set_property PACKAGE_PIN E19 [get_ports {LEDA[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[1]}]
set_property PACKAGE_PIN U19 [get_ports {LEDA[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[2]}]
set_property PACKAGE_PIN V19 [get_ports {LEDA[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[3]}]
set_property PACKAGE_PIN W18 [get_ports {LEDA[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[4]}]
set_property PACKAGE_PIN U15 [get_ports {LEDA[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[5]}]
set_property PACKAGE_PIN U14 [get_ports {LEDA[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[6]}]
set_property PACKAGE_PIN V14 [get_ports {LEDA[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {LEDA[7]}]
set_property PACKAGE_PIN W3 [get_ports {stateLed[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {stateLed[0]}]
set_property PACKAGE_PIN U3 [get_ports {stateLed[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {stateLed[1]}]
        set_property PACKAGE_PIN L1 [get_ports {lastLed}]

        set_property IOSTANDARD LVCMOS33 [get_ports {lastLed}]
```

```
#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg_outputs[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg_outputs[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg_outputs[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg_outputs[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg_outputs[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg_outputs[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg_outputs[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg_outputs[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 BTNC]

        set_property IOSTANDARD LVCMOS33 [get_ports BTNC]
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 BTND]

          set_property IOSTANDARD LVCMOS33 [get_ports BTND]
```