Course Code: CS223

Course Name: Digital Design

Section: 1

Project

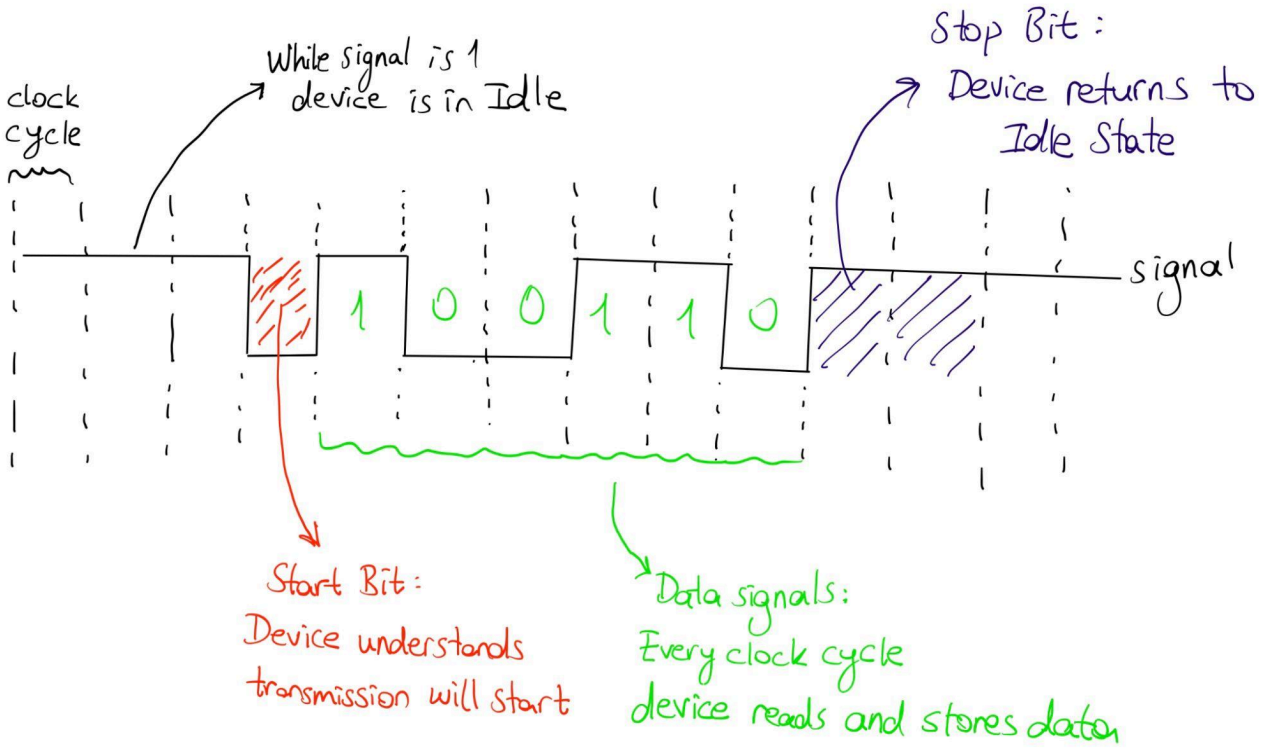
Mustafa Mert Gülhan

22201895

Date: 05.05.2024

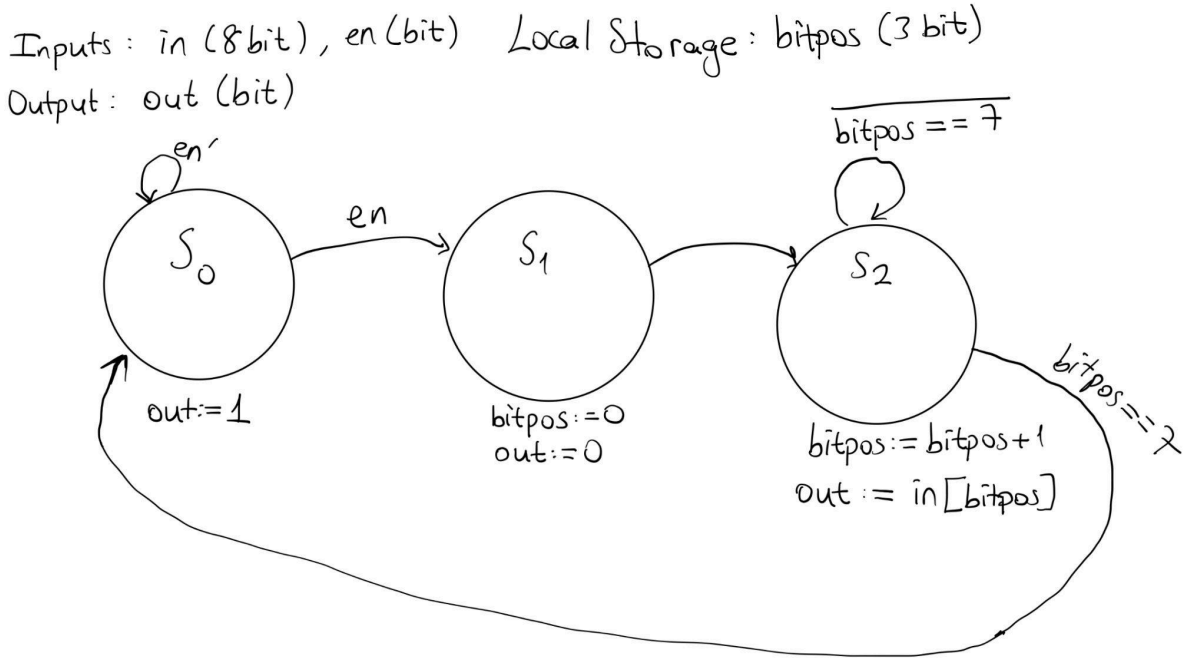
Detailed Description of UART Design:

Mainly the idea behind the UART device is to send multiple bits of data to another UART device with a single signal in a way that it can read the data from that signal and store it. The signal works in this way

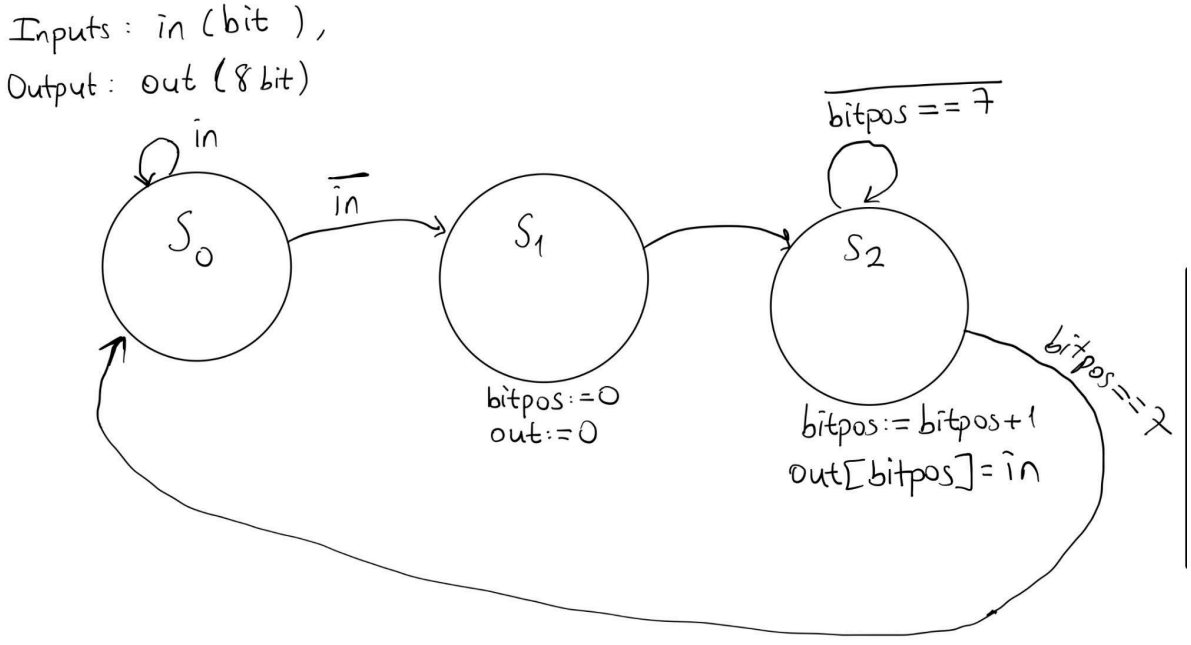


The UART_tx is responsible for generating this signal where UART_rx is responsible for reading that signal and storing the data. To work properly both tx and rx devices must agree on the amount of bits will be transmitted and the clock cycle.

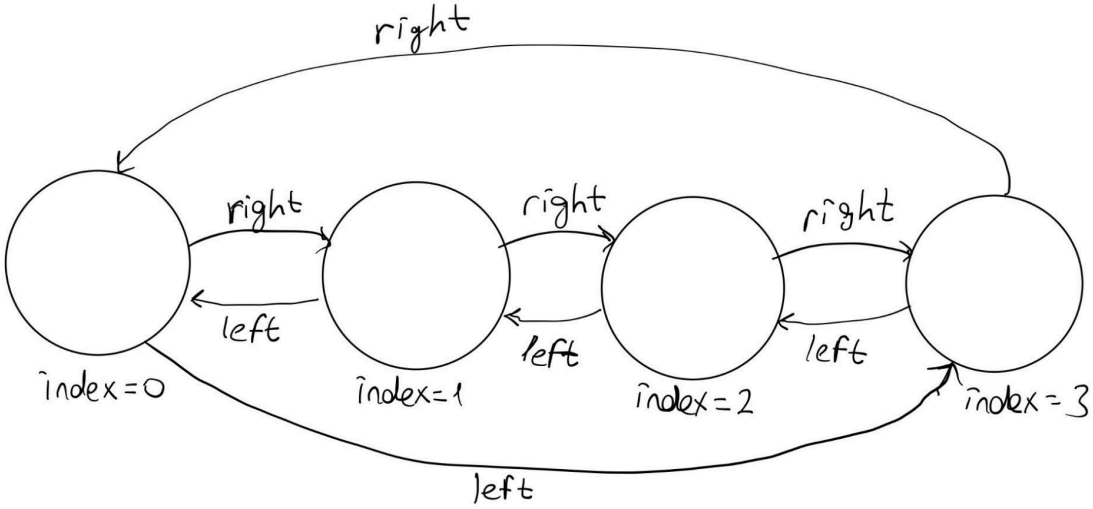
State Diagram for UART_tx:



State Diagram for UART_rx:

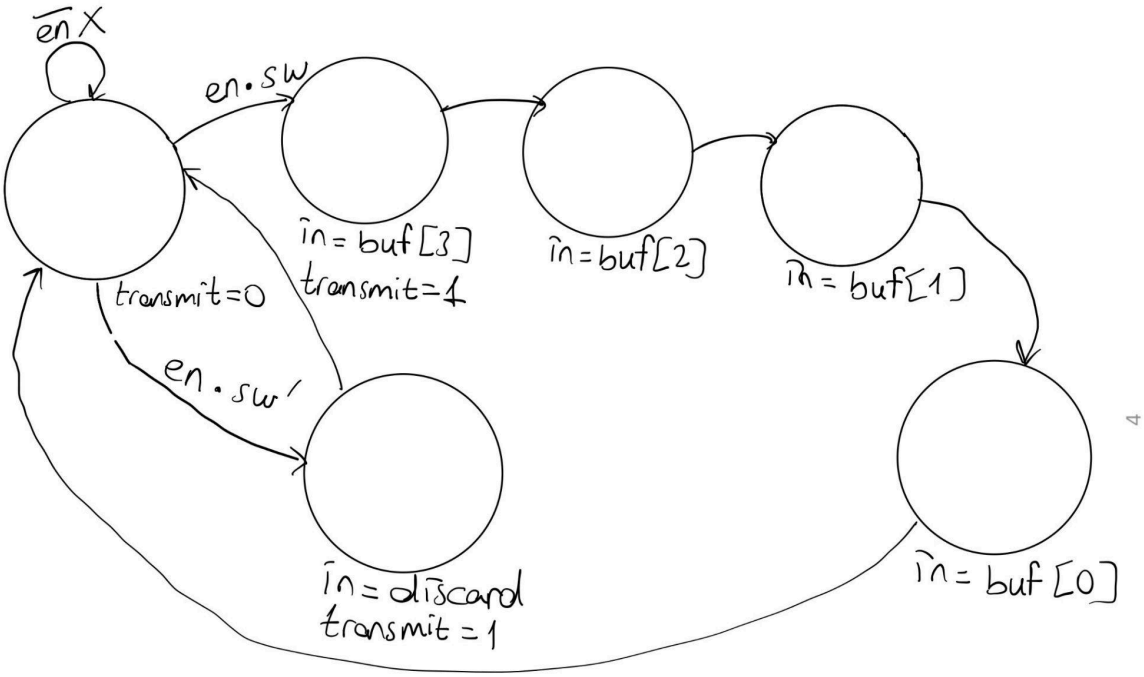


State Diagram for Display:

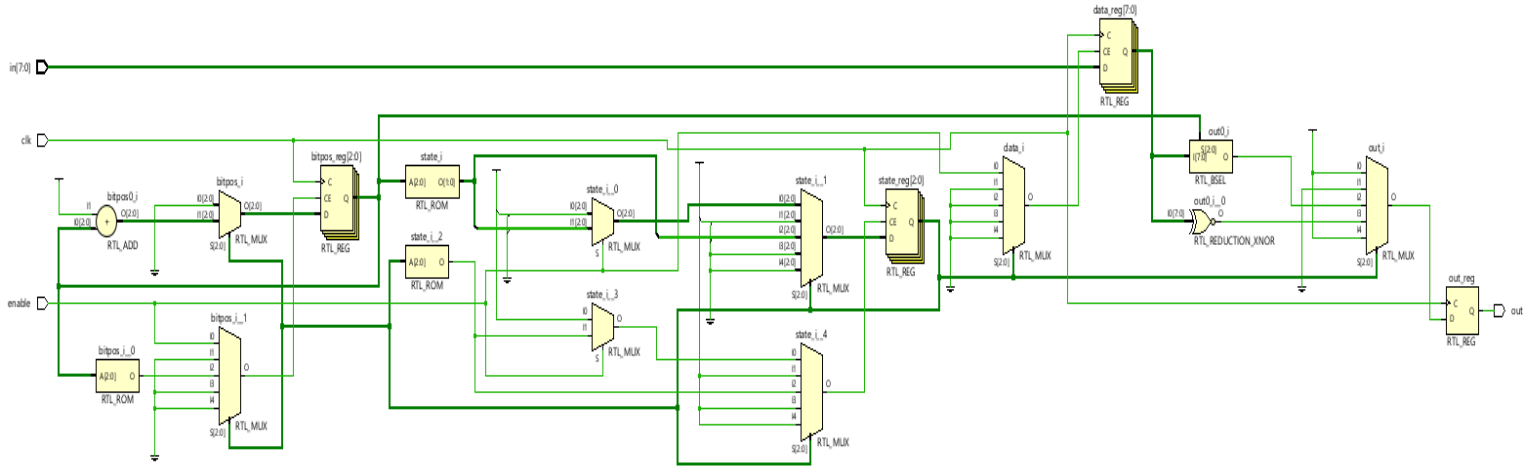


Out = buf [index]

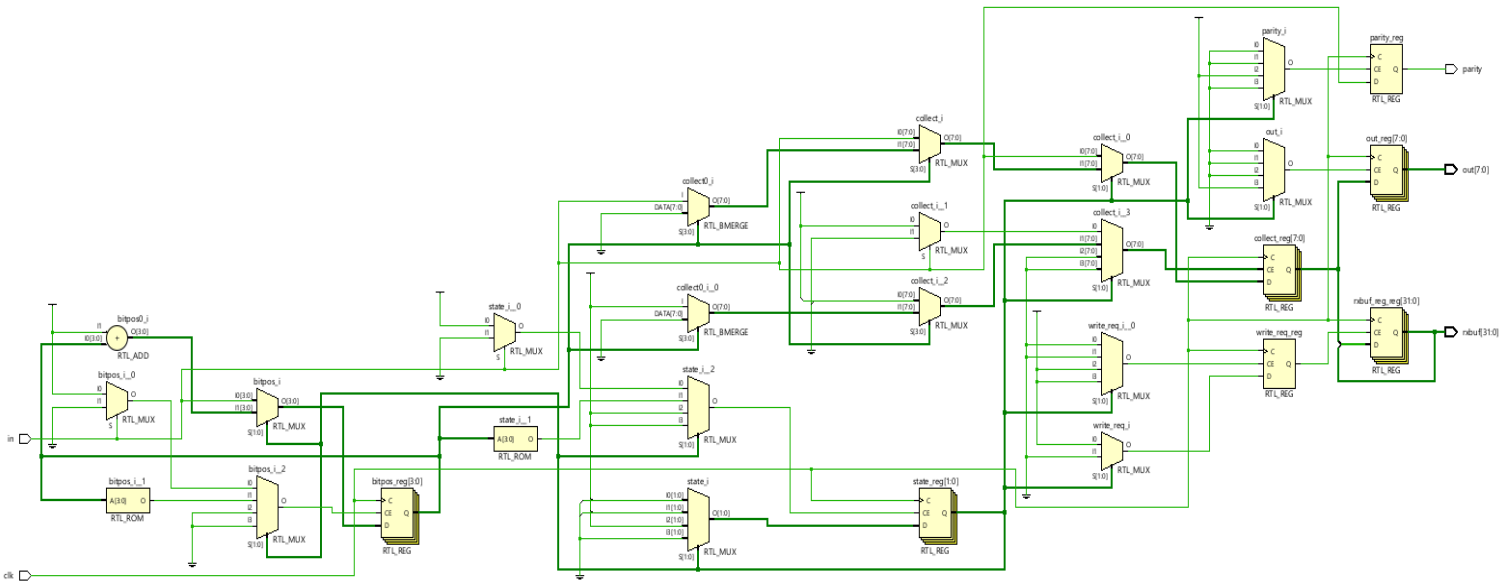
State Diagram for Continuous Transfer



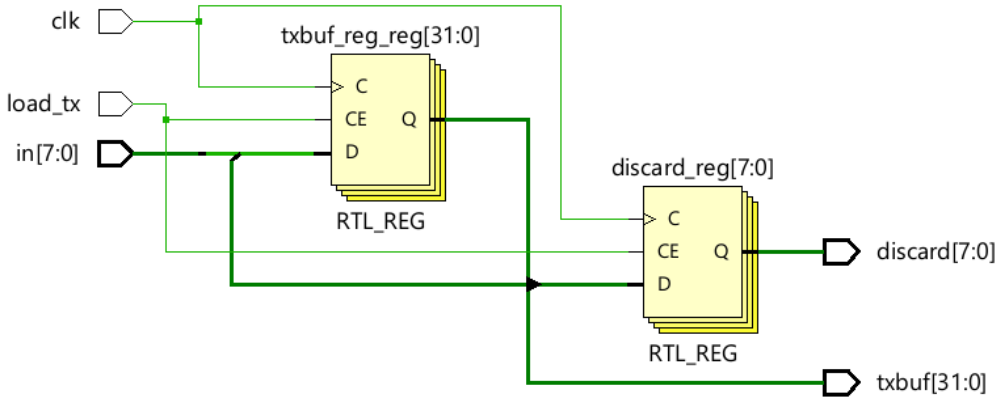
Schematic for UART_tx:



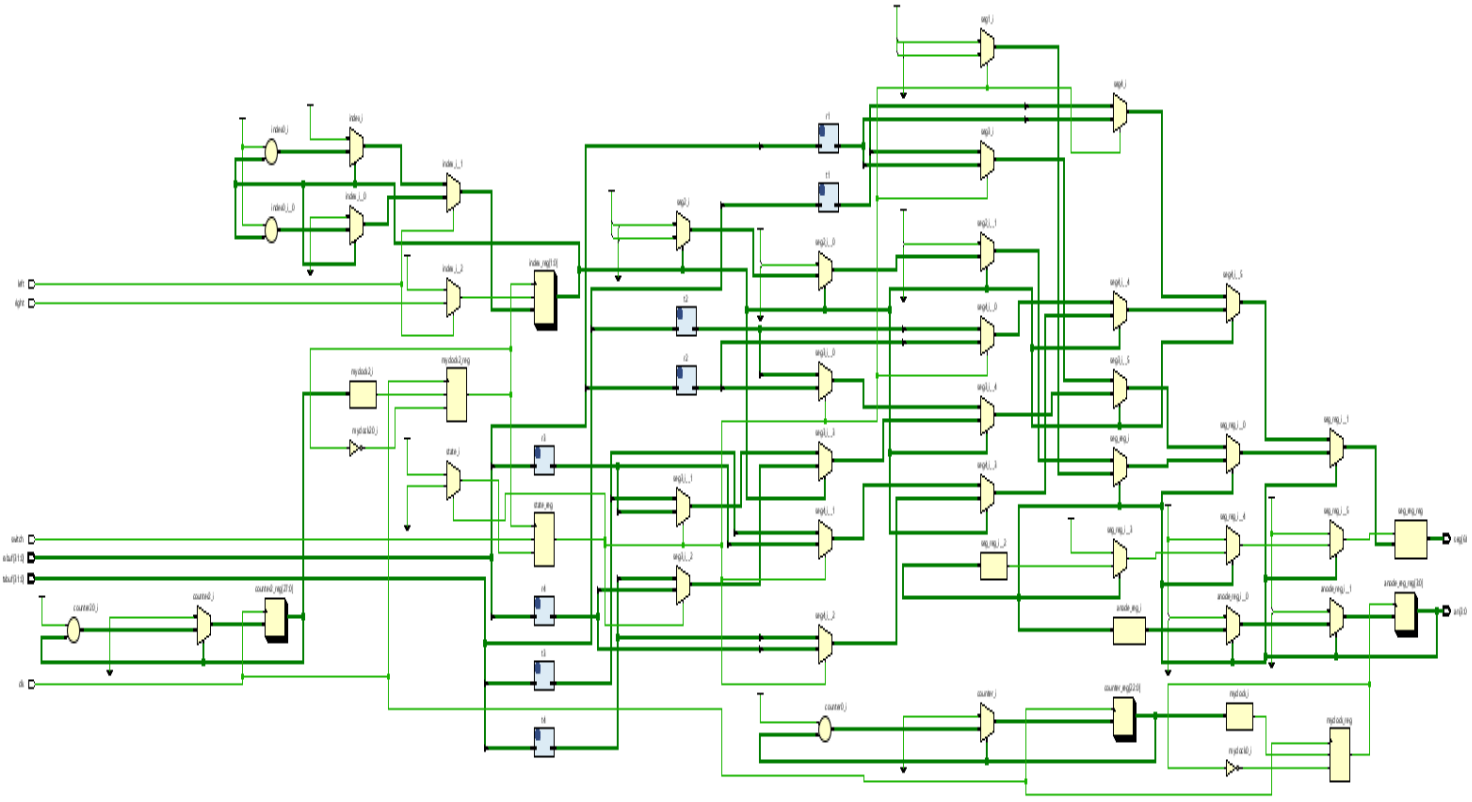
Schematic for UART_rx:



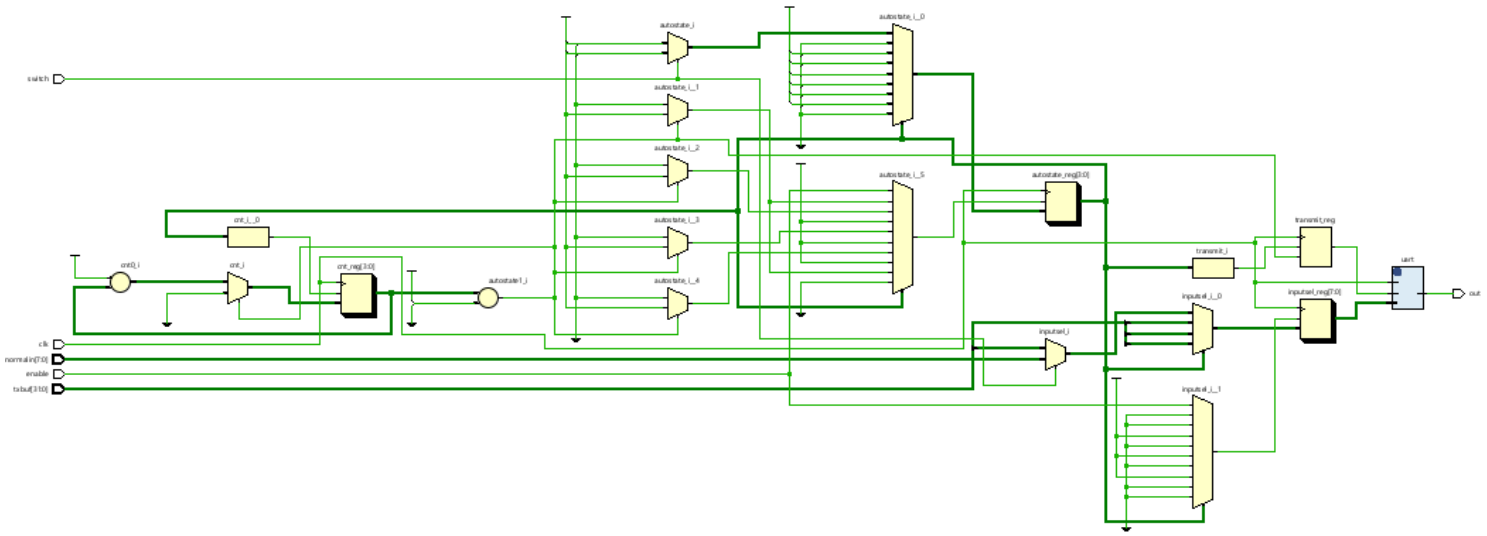
Schematic for Memory Arrays



Schematic for 7-Segment Display:



Schematic for Continuous Transfer:



Testbenches:

```
module autotransfer_tb();
    logic [31:0]txbuf;
    logic clk = 0;
    logic en = 0;
    logic out;
    logic [7:0] currout;
    logic par;
    logic [31:0] rxbuf;
    logic switchin = 0;
    logic [7:0] normalin;

    autotransfer dut(txbuf,clk,en,switchin,normalin,out);
    uart_rx test(out,clk,currout,par,rxbuf);

    always begin
        clk = ~clk; #5;
    end
```

```
    initial begin
        txbuf = 32'b10010001_01101100_11110010_00011101;
normalin = 8'b00010001; switchin = 1; #20;
        en = 1; #20; en = 0; #630;
        switchin = 0; #20; en = 1; #20; en = 0;
    end

endmodule

module uart_rx_tb();
   logic [7:0] tx_in;
    logic clk = 1;
    logic en;
    logic tx_out;
    logic [7:0] out;
    logic parity;
    logic [31:0] rxbuf;

    uart_tx signal(tx_in,clk,en,tx_out);
    uart_rx dut(tx_out,clk,out,parity,rxbuf);
    always begin
        clk = ~clk; #5;
    end

    initial begin

          en = 1; tx_in = 8'b10010011; #20; en = 0; #100;
          en = 1; tx_in = 8'b11000010; #20; en = 0; #100;
          en = 1; tx_in = 8'b10010101; #20; en = 0; #100;
          en = 1; tx_in = 8'b11110011; #20; en = 0; #100;
          en = 1; tx_in = 8'b10110111; #20; en = 0; #100;

    end
endmodule

module buttonlatch_tb();
```

```
    logic clk= 0;
    logic btn,stable;

    always begin
        clk = ~clk; #10;
    end
    buttonlatch dut(clk,btn,stable);
    initial begin
        btn = 0; #20;
        btn = 1; #50;
        btn = 0; #50;
        btn = 1; #50;
        btn = 0;
    end
endmodule

module sevenseg_tb();
    logic [7:0] in;
    logic [13:0] out;
    sevenseg dut(in,out);
    initial begin
        in = 8'b1001_0111; #20;
        in = 8'b1111_1000; #20;
        in = 8'b1011_0011; #20;
    end
endmodule

module uart_tx_tb();
  logic [7:0]in;
  logic clk = 1;
  logic out;

  uart_tx dut(0,0,in,clk,1'b1,out);
  always begin
    clk = ~clk; #10;
  end
  initial begin
```

```
    in = 8'b10110110; #205;
    in = 8'b01110001;
  end
endmodule
```

Appendix

```
module sevenseg(input logic [7:0] in, output logic [13:0]
out );
    always @(*) begin
        case(in[3:0])
            4'b0000: out[6:0] = 7'b0000001;
            4'b0001: out[6:0] = 7'b1001111;
            4'b0010: out[6:0] = 7'b0010010;
            4'b0011: out[6:0] = 7'b0000110;
            4'b0100: out[6:0] = 7'b1001100;
            4'b0101: out[6:0] = 7'b0100100;
            4'b0110: out[6:0] = 7'b0100000;
            4'b0111: out[6:0] = 7'b0001111;
            4'b1000: out[6:0] = 7'b0000000;
            4'b1001: out[6:0] = 7'b0000100;
            4'b1010: out[6:0] = 7'b0001000; //a
            4'b1011: out[6:0] = 7'b1100000; //b
            4'b1100: out[6:0] = 7'b1110010; //c
            4'b1101: out[6:0] = 7'b1000010; //d
            4'b1110: out[6:0] = 7'b0110000; //e
            4'b1111: out[6:0] = 7'b0111000; //f
            default: out[6:0] = 7'b1111110;
        endcase
        case(in[7:4])
            4'b0000: out[13:7] = 7'b0000001;
            4'b0001: out[13:7] = 7'b1001111;
            4'b0010: out[13:7] = 7'b0010010;
            4'b0011: out[13:7] = 7'b0000110;
```

```
            4'b0100: out[13:7] = 7'b1001100;
            4'b0101: out[13:7] = 7'b0100100;
            4'b0110: out[13:7] = 7'b0100000;
            4'b0111: out[13:7] = 7'b0001111;
            4'b1000: out[13:7] = 7'b0000000;
            4'b1001: out[13:7] = 7'b0000100;
            4'b1010: out[13:7] = 7'b0001000; //a
            4'b1011: out[13:7] = 7'b1100000; //b
            4'b1100: out[13:7] = 7'b1110010; //c
            4'b1101: out[13:7] = 7'b1000010; //d
            4'b1110: out[13:7] = 7'b0110000; //e
            4'b1111: out[13:7] = 7'b0111000; //f
            default: out[13:7] = 7'b1111110;

        endcase
    end
endmodule

module display(
input logic [31:0]txbuf,
input logic [31:0]rxbuf,
input logic switch,
input logic right,
input logic left,
input logic clk,
output logic[3:0] an,
output logic [6:0] seg);

    logic[13:0] tx0;
    logic[13:0] tx1;
    logic[13:0] tx2;
    logic[13:0] tx3;

    logic[13:0] rx0;
    logic[13:0] rx1;
    logic[13:0] rx2;
    logic[13:0] rx3;
```

```
    sevenseg t1(txbuf[7:0],tx0);
    sevenseg t2(txbuf[15:8],tx1);
    sevenseg t3(txbuf[23:16],tx2);
    sevenseg t4(txbuf[31:24],tx3);

    sevenseg r1(rxbuf[7:0],rx0);
    sevenseg r2(rxbuf[15:8],rx1);
    sevenseg r3(rxbuf[23:16],rx2);
    sevenseg r4(rxbuf[31:24],rx3);

    parameter state_tx = 0;
    parameter state_rx = 1;
    reg state = state_tx;

    reg [6:0]seg_reg = 7'b1110000;
    reg [3:0]anode_reg = 4'b0111;


    logic [6:0] seg1 = 7'b1110000; //displays t or x on
current state
    logic [6:0] seg2 = 7'b0000001; //display the index
    logic [6:0] seg3 = 0;
    logic [6:0] seg4 = 0;

    logic[22:0] counter = 0;
    logic[27:0] counter2 = 0;
    logic myclock = 0;
    logic myclock2 = 0;

    logic [1:0] index = 0;

    always @(posedge myclock2) begin
        if(switch) begin
            if(state == state_tx)
                state = state_rx;
            else
```

```
            state = state_tx;
    end
end

always @(posedge myclock2) begin
    if(right) begin
        if(index == 2'b11)
            index <= 0;
        else
            index <= index + 1'b1;
    end
    if(left) begin
        if(index == 2'b00)
            index <= 2'b11;
        else
            index <= index - 1'b1;
    end
end
always @(state) begin
    if(state == state_tx)
        seg1 <= 7'b1110000;
    else
        seg1 <= 7'b1111010;
end
always @(index) begin
    if(index == 2'b00) begin
        seg2 <= 7'b0000001;
        if(state == state_tx) begin
            seg3 <= tx0[13:7];
            seg4 <= tx0[6:0];
        end
        else begin
            seg3 <= rx0[13:7];
            seg4 <= rx0[6:0];
        end
    end
    else if(index == 2'b01) begin
```

```
            seg2 <= 7'b1001111;
            if(state == state_tx) begin
                seg3 <= tx1[13:7];
                seg4 <= tx1[6:0];
            end
            else begin
                seg3 <= rx1[13:7];
                seg4 <= rx1[6:0];
            end
        end
        else if(index == 2'b10) begin
            seg2 <= 7'b0010010;
            if(state == state_tx) begin
                seg3 <= tx2[13:7];
                seg4 <= tx2[6:0];
            end
            else begin
                seg3 <= rx2[13:7];
                seg4 <= rx2[6:0];
            end
        end
        else begin
            seg2 <= 7'b0000110;
            if(state == state_tx) begin
                seg3 <= tx3[13:7];
                seg4 <= tx3[6:0];
            end
            else begin
                seg3 <= rx3[13:7];
                seg4 <= rx3[6:0];
            end
        end
    end
    always @(posedge clk) begin
        if(counter  == 100000)
        begin
            counter <= 0;
```

```
            myclock <= ~myclock;
        end
        else
        begin
            counter <= counter+1;
        end
    end

    always @(posedge clk) begin
        if(counter2  == 100000000)
        begin
            counter2 <= 0;
            myclock2 <= ~myclock2;
        end
        else
        begin
            counter2 <= counter2+1;
        end
    end

    always @(posedge myclock) begin //myclock to clk on
testbench
        if(anode_reg == 4'b1110)
            anode_reg <= 4'b1101;
        else if(anode_reg == 4'b1101)
            anode_reg <= 4'b1011;
        else if(anode_reg == 4'b1011)
            anode_reg <= 4'b0111;
        else
            anode_reg <= 4'b1110;
    end

    always @(anode_reg) begin
            if(anode_reg == 4'b1110)
                seg_reg <= seg4;
            else if(anode_reg == 4'b1101)
                seg_reg <= seg3;
```

```
            else if(anode_reg == 4'b1011)
                seg_reg <= seg2;
            else if(anode_reg == 4'b0111)
                seg_reg <= seg1;
     end

     assign an = anode_reg;
     assign seg = seg_reg;
endmodule

module uart_rx
(
    input logic in,
    input logic clk,
    output logic [7:0] out,
    output logic parity,
    output logic [31:0] rxbuf
);

    initial begin
        out = 8'b0;
    end

    parameter STATE_START = 2'b00;
    parameter STATE_DATA = 2'b01;
    parameter STATE_PARITY= 2'b10;
    parameter STATE_STOP = 2'b11;

    reg [1:0] state = STATE_START;
    reg [3:0] bitpos = 4'd0;
    reg [7:0] collect = 8'b0;
    reg [31:0] rxbuf_reg = 32'b0;

    reg write_req = 0; // New register for write request


    logic [7:0] discard = 0;
```

```
    always @(posedge clk) begin
        case(state)

        STATE_START: begin
            if(!in) begin
                state <= STATE_DATA;
                bitpos <= 4'd0;
                collect <= 0;
            end
        end
        STATE_DATA: begin
            if(bitpos == 4'd7) begin
                collect[bitpos] <= in;
                state <= STATE_PARITY;
            end
            else begin
                collect[bitpos] <= in; //bitpos[2:0] if
not work
                bitpos <= bitpos + 1'b1;
            end
        end
        STATE_PARITY: begin
            state <= STATE_STOP;
            parity <= in;
            write_req <= 1'b1;
        end
        STATE_STOP: begin
            write_req <= 1'b0;
            out <= collect;
            state <= STATE_START;
        end
        default: begin
              state <= STATE_START;
         end
        endcase
    end
```

```
    always @(posedge clk) begin
        if (write_req) begin
            rxbuf_reg[31:8] <= rxbuf_reg[23:0];
            rxbuf_reg[7:0] <= collect;
        end
    end


    assign rxbuf = rxbuf_reg; //en yeni [7:0] en eski
[31:24]

endmodule

module autotransfer(
    input logic [31:0] txbuf,
    input logic clk,
    input logic enable,
    input logic switch,
    input logic [7:0] normalin,
    output logic out
);
    parameter ASTATE_IDLE = 4'b0000;
    parameter ASTATE_TX4 = 4'b0001;
    parameter ASTATE_LOADTX3 = 4'b0010;
    parameter ASTATE_TX3 = 4'b0011;
    parameter ASTATE_LOADTX2 = 4'b0100;
    parameter ASTATE_TX2 = 4'b0101;
    parameter ASTATE_LOADTX1 = 4'b0110;
    parameter ASTATE_TX1 = 4'b0111;
    parameter ASTATE_NORMAL = 4'b1000;

    logic [7:0] inputsel;
    logic[3:0] autostate = ASTATE_IDLE;
    logic transmit = 0;

    logic [3:0]cnt = 0;
```

```
uart_tx uart(inputsel,clk,transmit,out);

always @(posedge clk) begin
    case(autostate)
    ASTATE_IDLE: begin
        if(enable) begin
            if(switch) begin
                autostate <= ASTATE_TX4;
                inputsel <= txbuf[31:24];
            end
            else begin
                autostate <= ASTATE_NORMAL;
                inputsel <= normalin;
            end
        end
    end
    ASTATE_NORMAL: begin
        transmit <= 1;
        if(cnt != 4'b1010) begin
            cnt <= cnt + 1;
        end
        else begin
            autostate <= ASTATE_IDLE;
            cnt <= 0;
            transmit <= 0;
        end
    end
    ASTATE_TX4: begin
        transmit <= 1;
        if(cnt != 4'b1010) begin
            cnt <= cnt + 1;
        end
        else begin
            autostate <= ASTATE_LOADTX3;
            cnt <= 0;
            transmit <= 0;
```

```
    end
end
ASTATE_LOADTX3: begin
    inputsel <= txbuf[23:16];
    autostate <= ASTATE_TX3;
end
ASTATE_TX3: begin
    transmit <= 1;
    if(cnt != 4'b1010) begin
        cnt <= cnt + 1;
    end
    else begin
        autostate <= ASTATE_LOADTX2;
        cnt <= 0;
        transmit <= 0;
    end
end
ASTATE_LOADTX2: begin
    inputsel <= txbuf[15:8];
    autostate <= ASTATE_TX2;
end
ASTATE_TX2: begin
    transmit <= 1;
    if(cnt != 4'b1010) begin
        cnt <= cnt + 1;
    end
    else begin
        autostate <= ASTATE_LOADTX1;
        cnt <= 0;
        transmit <= 0;
    end
end
ASTATE_LOADTX1: begin
    inputsel <= txbuf[7:0];
    autostate <= ASTATE_TX1;
end
ASTATE_TX1: begin
```

```
            transmit <= 1;
            if(cnt != 4'b1010) begin
                cnt <= cnt + 1;
            end
            else begin
                autostate <= ASTATE_IDLE;
                cnt <= 0;
                transmit <= 0;
            end
        end
        endcase
    end
endmodule

module uart_tx
(
  input logic [7:0]in,
  input logic clk,
  input logic enable,
  output logic out
);
    initial begin
        out = 1'b1;
    end
    parameter STATE_IDLE = 3'b000;
    parameter STATE_START = 3'b001;
    parameter STATE_DATA = 3'b010;
    parameter STATE_PARITY = 3'b011;
    parameter STATE_STOP = 3'b100;


    reg [7:0] data = 8'b0;
    reg [2:0] bitpos = 3'b0;
    reg [2:0] state = STATE_IDLE;
    reg [31:0] txbuf_reg = 0;
```

```
    always @(posedge clk) begin
            case(state)
            STATE_IDLE: begin
                out <= 1'b1;
                if(enable) begin
                        state <= STATE_START;
                        data <= in;
                        bitpos <= 3'b0;
                end
            end
            STATE_START: begin
                out <= 1'b0;
                state <= STATE_DATA;
            end
            STATE_DATA: begin
                if(bitpos == 3'd7) begin
                    state = STATE_PARITY;
                end
                else begin
                    bitpos <= bitpos + 3'd1;
                end
                out <= data[bitpos];
            end
            STATE_PARITY: begin
                out <= ~^data;
                state <= STATE_IDLE;
            end
            default: begin
                out <= 1'b1;
                state <= STATE_IDLE;
            end
        endcase
        end
endmodule

module buttonlatch(input logic clk,btn, output logic
stable);
```

```
    initial begin
        stable = 0;
    end
    reg isclicked = 0;
    reg temp = 0;
    always @(posedge clk) begin
        if(isclicked && btn) begin
            stable <= 0;
            isclicked <= 0;
        end
        if(!isclicked && btn && !temp) begin
            stable <= 1;
            temp <= 1;
            isclicked <= 1;
        end
        if(!isclicked && !btn) begin
            temp <= 0;
        end

    end
endmodule

module txbuf(input logic clk,input logic load_tx, input
logic[7:0] in, output logic[31:0]txbuf, output logic
[7:0] discard);
    reg [31:0]txbuf_reg = 0;
    initial begin
        discard = 0;
    end
    always @(posedge clk) begin
        if(load_tx) begin
            discard <= txbuf_reg[31:24];
            txbuf_reg[31:24] <= txbuf_reg[23:16];
            txbuf_reg[23:16] <= txbuf_reg[15:8];
            txbuf_reg[15:8] <= txbuf_reg[7:0];
            txbuf_reg[7:0] <= in;
        end
```

```
    end

    assign txbuf = txbuf_reg;

endmodule

module baudclk(input logic clk, output logic baudclk);
    parameter baudcount = 100_000_000 / 115200;
    parameter baudcount_width = $clog2(baudcount);
    reg [baudcount_width-1:0] clk_acc = 0;

    assign baudclk = (clk_acc == 10'd0);

    always @(posedge clk) begin
        if (clk_acc == baudcount[baudcount_width-1:0])
            clk_acc <= 0;
        else
            clk_acc <= clk_acc + 10'b1;
    end

endmodule

module uart(
  input logic [7:0]switchin,
  input logic transmit4,
  input logic clk,
  input logic switch,
  input logic left,
  input logic right,
  input logic tx_write,
  input logic tx_enable,
  input logic sw15,

  output logic [7:0] txbufled,
  output logic [7:0] rxbufled,
  output logic [3:0] an,
  output logic [6:0] seg
```

```
);
    //clk generator with specified baudrate
    logic tx_out; //output logic tx_out yapılacak
    logic rx_in; //input logic rx_in yapılacak


    logic baudclk;
    logic tx_out_reg;
    logic [31:0] txbuf;
    logic [31:0] rxbuf;
    logic [7:0] rx_out;
    logic parity;
    logic stable_enable;
    logic stable_write;
    baudclk clkgen(clk,baudclk);

    //module for handling 4byte txbuf read,write
    logic [7:0] discard;

    buttonlatch bt2(baudclk,tx_write,stable_write);
    txbuf
txbufwriter(baudclk,stable_write,switchin,txbuf,discard);

    //uart modules
    buttonlatch bt(baudclk,tx_enable,stable_enable);
    //uart_tx
tx(switchin,baudclk,stable_enable,tx_out_reg);
    autotransfer
at_tx(txbuf,baudclk,stable_enable,sw15,switchin,tx_out_re
g); //switchin to discard
    uart_rx rx(tx_out_reg,baudclk,rx_out,parity,rxbuf);
//tx_out_reg to rx_in
    display(txbuf,rxbuf,switch,right,left,clk,an,seg);

    assign txbufled = txbuf[7:0];
    assign rxbufled = rxbuf[31:24];
    assign tx_out = tx_out_reg;
```

```
endmodule
```