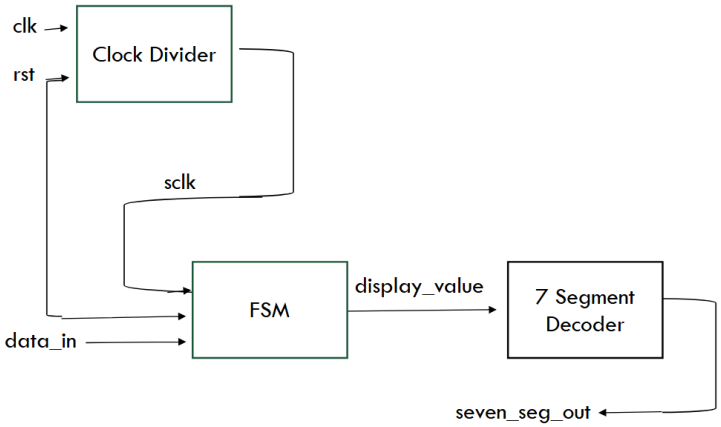
Sara Nessim
Alexis Nhan
EGEC 446

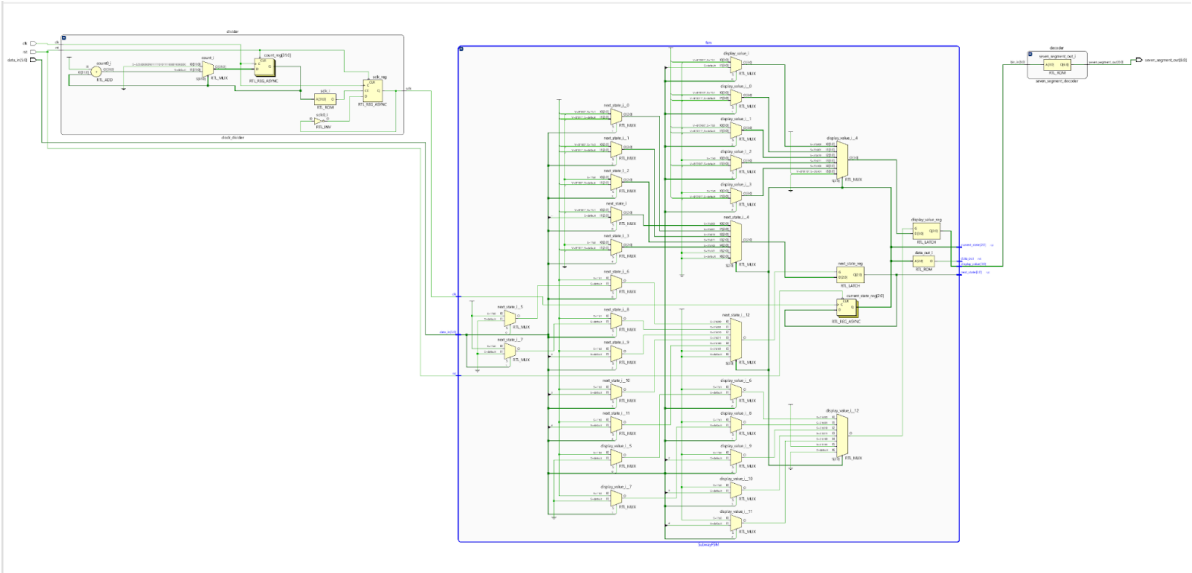
Subway Ticketing FSM Report

Our project is a subway ticketing system that uses a finite state machine, FSM. How the project works is by having the code go through various states until it gets to the last state that will restart the state cycle. The states go as follows: is the machine open, how much are you paying, paying 10, paying 20, paying 30, and dispense ticket. We have the states being triggered based on what switch is flipped on. The FSM would only start if the first switch is flipped on, which tells the user that the machine that it is open. From there the states will transition based on what the following switch inputs are used. With only combinations that will get the user to the end of the FSM which is the subway ticket getting dispensed.

System Diagram:

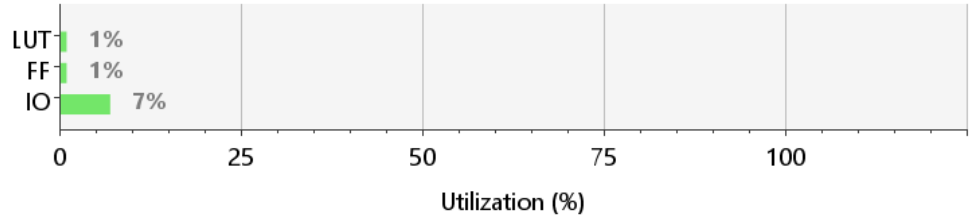


RTL Schematic:



Area Report:

Resource	Utilization	Available	Utilization %
LUT	51	32600	0.16
FF	39	65200	0.06
IO	14	210	6.67



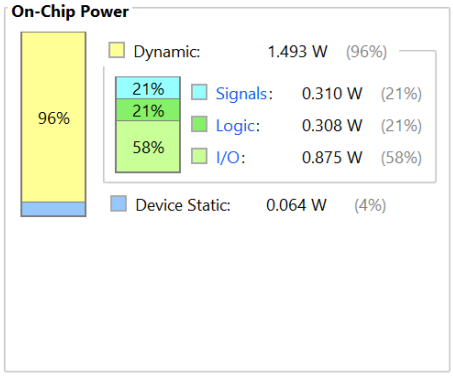
Power Report:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.557 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**32.4°C**
Thermal Margin:	67.6°C (14.1 W)
Effective ϑJA:	4.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Static Timing Analysis Report:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.869 ns	Worst Hold Slack (WHS):	0.117 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	33	Total Number of Endpoints:	33	Total Number of Endpoints:	34

All user specified timing constraints are met.

Discussion:

We designed a Mealy finite state machine that was dependent on current state and inputs. We had a 6-bit input called data_in. This would be considered by the FSM if we had tickets available. Data_in[0] when switched on would mean that we had tickets available. We used a clock divider so that we would be able to have the timing of the display to be visible as well as the changing states. We connected the output called slowclock to the input of our FSM. We had a reset to simulate if for example a user walked away from the screen and didn't want the ticket anymore, it would reset the FSM to 0. We output values to the 7 segment display using the BCD code to be able to successfully output from the FSM module to the bin_in to the seven segment display. Data_out was an output and display_value was what connected the module. Once we were successful and the sequence '$30' was complete and not overflowing, The 7-segment display output a 'P' for paid and 'd' for display ticket.

Challenges:

Some challenges we faced were creating a testbench to test the finite state machine and creating the output display for the seven segment display for each state. The finite state machine testbench was solved by slowly going through our state diagram figuring out what inputs would trigger an output. As for the seven segment display, the issue was that only the initial state was displaying an output programmed by the display_value. To fix this we found that the clock was cycling through the FSM at too fast of a rate causing each state to not be displayed. So we added a clock divider to slow down the rate the clock would cycle through each state. This ultimately fixed the display timing issue we were facing and outputted the display at each state.

References:

The references used were the lecture slides provided from the class.

Appendix:

```
`timescale 1ns / 1ps

module clock_divider(

input clk,
input rst,
output reg sclk
);
reg [31:0] count;
   always@(posedge clk or negedge rst)
      begin
   if(rst == 1'b0) begin
```

```
    count <= 32'd0;
    sclk <= 1'b0;
       end else begin
    if(count == 32'd50000000) begin //this is for 10s, 50000000 for 1 sec
    count <= 32'd0;
    sclk <= ~sclk;
       end
       else begin
    count <= count + 1;
       end
    end
    end
    endmodule

module SubwayFSM(
    input [5:0] data_in,
    input clk,
    input rst,
    output reg data_out,
    output reg [3:0] display_value,
    reg [2:0] current_state,
    reg [2:0] next_state
    );

    parameter S0 = 3'b000;
    parameter S1 = 3'b001;
    parameter S2 = 3'b010;
    parameter S3 = 3'b011;
    parameter S4 = 3'b100;
    parameter S5 = 3'b101;


   always @(posedge clk or negedge rst)
       begin
         if (rst == 1'b0)
   //            display_value = 4'b0011;
               current_state = S0;
         else
             current_state = next_state;
      end
```

```
always @(current_state && data_in) //money is data in
   begin
      case (current_state)
         S0: begin
            if (data_in[0] == 1'b1)
               begin
               display_value = 4'b0001; //open (0)
               next_state = S1; //S1 is how much money
               end
            else if (data_in[0] == 1'b0)
               begin
               display_value = 4'b0000; //N
               next_state = S0;
               end

            end
         S1: begin
            if (data_in[1] == 1'b1)
               begin
               display_value = 4'b0010; //1 (10)
               next_state = S2;
               end
            else if (data_in[1] == 1'b0)
               begin
               display_value = 4'b0011; //2 (20)
               next_state = S3;
               end
            end
         S2: begin
            if (data_in[2] == 1'b0)
               begin
               display_value = 4'b0100; //30
               next_state = S4;
               end
            else if (data_in[2] == 1'b1)
               begin
               display_value = 4'b0011;
               next_state = S3;
               end
```

```
            end
        S3: begin
            if (data_in[3] == 1'b0)
                begin
                display_value = 4'b0000;
                next_state = S0;
                end
            else if (data_in[3] == 1'b1)
                begin
                display_value = 4'b0100;
                next_state = S4;
                end
            end
        S4: begin
            if (data_in[4] == 1'b0)
                begin
                display_value = 4'b0000;
                next_state = S0;
                end
            else if (data_in[4] == 1'b1)
                begin
                display_value = 4'b0101; //P(pay)
                next_state = S5;
                end
            end
        S5: begin
            display_value = 4'b0110; //Dispense (D)
            next_state = S0;

                end
        default: begin
                next_state = S0;
                end

        endcase
    end
always @(next_state)
    begin
        case (current_state)
            S0: data_out <= 1'b0;
```

```
                S1: data_out <= 1'b0;
                S2: data_out <= 1'b0;
                S3: data_out <= 1'b0;
                S4: data_out <= 1'b0;
                S5: data_out <= 1'b1;
                default: data_out <= 1'b0;
            endcase
        end

endmodule


module seven_segment_decoder(
    input [3:0] bin_in,
    output [6:0] seven_segment_out
    );
    reg [6:0] seven_segment_out;


        always @(bin_in)
          begin
          case (bin_in) //case statement
          0 : seven_segment_out = 7'b0001001; //N
          1 : seven_segment_out = 7'b0000001; //0 (open)
          2 : seven_segment_out = 7'b1111001; //10
          3 : seven_segment_out = 7'b0010010; //20
          4 : seven_segment_out = 7'b0000110; //30
          5 : seven_segment_out = 7'b0011000; //P (pay)
          6 : seven_segment_out = 7'b1000010; //D(dispense)
          7 : seven_segment_out = 7'b0001111;
          8 : seven_segment_out = 7'b0000000;
          9 : seven_segment_out = 7'b0000100;
          //switch off 7 segment character
          default : seven_segment_out = 7'b1111111; //nothing being displayed
        endcase
    end
endmodule


module combined(
```

```
    input [5:0] data_in,
    input clk,
    input rst,
    output [6:0] seven_segment_out
);
    wire Slowclock;
    wire [3:0] data_out;
    wire [2:0] current_state;
    wire [3:0] display_value;

    SubwayFSM fsm(
        .data_in(data_in),
        .clk(Slowclock),
        .rst(rst),
        .display_value(display_value)
        //.display_value(current_state)
        //.data_out(display_value)

    );
    clock_divider divider(.clk(clk), .rst(rst), .sclk(Slowclock));

    seven_segment_decoder decoder(
        .bin_in(display_value),
        .seven_segment_out(seven_segment_out)
    );
endmodule
```