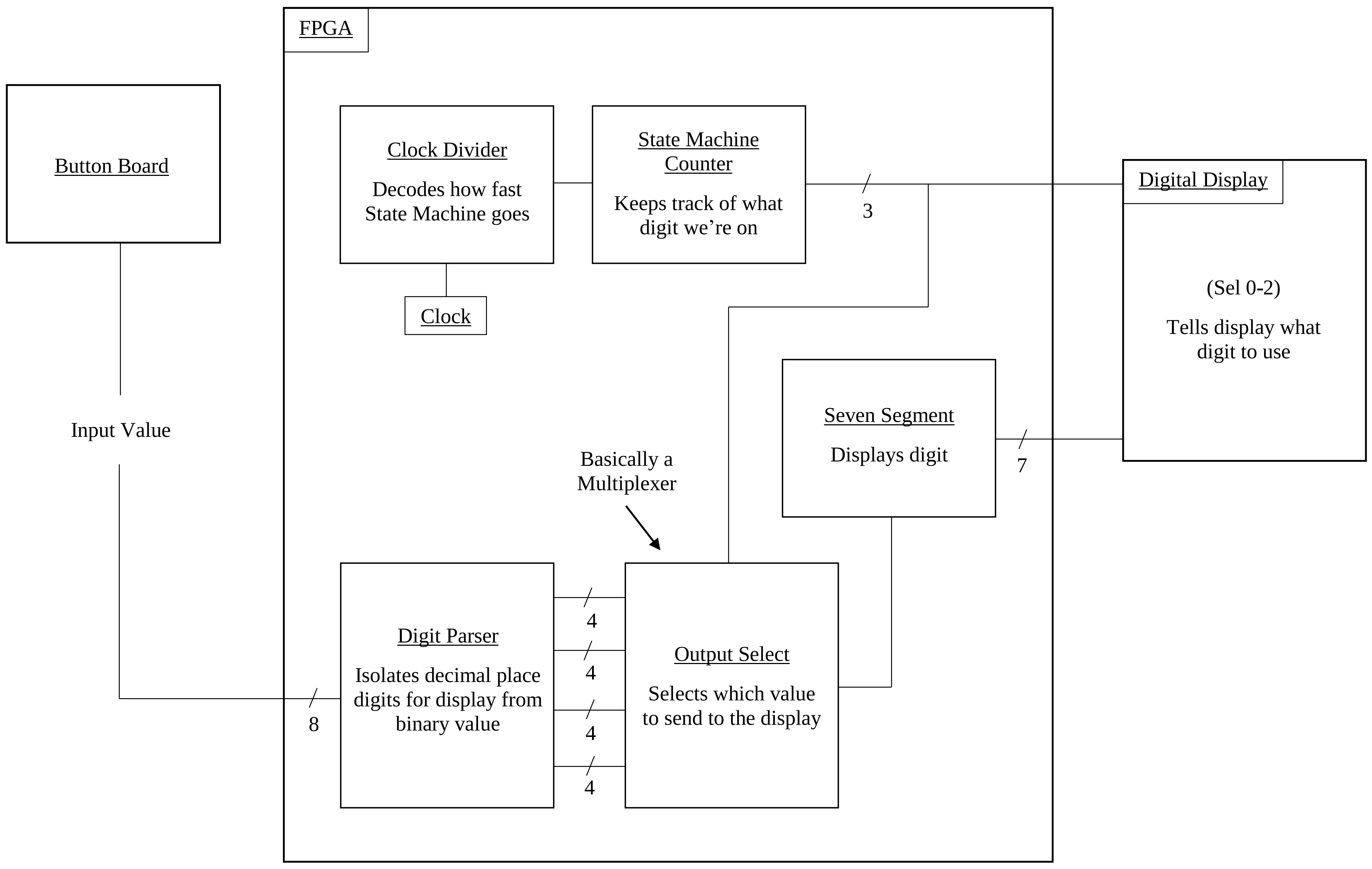
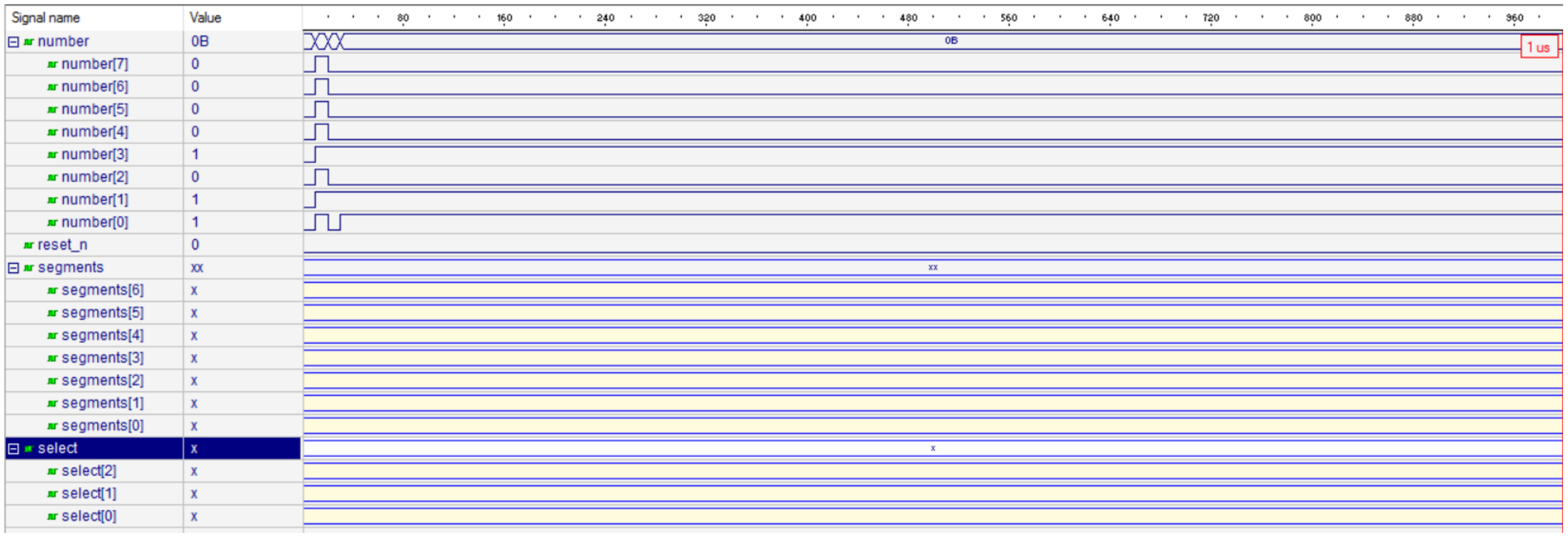
Rhea Mae Edwards


Button Board
Input Value
8
FPGA
Clock Divider
Decodes how fast State Machine goes
Clock
State Machine Counter
Keeps track of what digit we're on
3
Seven Segment
Displays digit
7
Basically a Multiplexer
Digit Parser
Isolates decimal place digits for display from binary value
4
4
4
4
Output Select
Selects which value to send to the display
Digital Display
(Sel 0-2)
Tells display what digit to use

Rhea Mae Edwards



Signal name	Value
number	0B
number[7]	0
number[6]	0
number[5]	0
number[4]	0
number[3]	1
number[2]	0
number[1]	1
number[0]	1
reset_n	0
segments	xx
segments[6]	x
segments[5]	x
segments[4]	x
segments[3]	x
segments[2]	x
segments[1]	x
segments[0]	x
select	x
select[2]	x
select[1]	x
select[0]	x

```
module digit_parser(
    input [7:0] number,
    output reg [3:0] ones,
    output reg [3:0] tens,
    output reg [3:0] hunds,
    output reg [3:0] thous );

    always @ (*)
        begin
            ones = number % 10;
            tens = number * 13/128 % 10;
            hunds = number * 13/128 * 13/138 % 10;
            thous = number * 13/128 * 13/128 * 13/128 % 10;
        end

endmodule

module output_select(
    input [2:0] select,
    input [3:0] ones,
    input [3:0] tens,
    input [3:0] hunds,
    input [3:0] thous,
    output reg [3:0] data );


    always @ (*)
        begin
            case (select)
                0: data = ones;
                1: data = tens;
                3: data = hunds;
                4: data = thous;
            endcase
        end

endmodule

module state_machine( //example of a Moore type state machine
    input clk_i,
    input reset_n,

    output reg [2:0] select );

        //state and next state registers
        reg [1:0] state;
        reg [1:0] state_n;

        //each possible value of the state register is given a unique name for easier use later
        parameter S0 = 2'b00;
        parameter S1 = 2'b01;
        parameter S2 = 2'b10;
        parameter S3 = 2'b11;

        //asynchronous reset will set the state to the start, S0, otherwise, the state is changed on the positive edge of the clock signal
        always @ (posedge clk_i, negedge reset_n)
            begin
                if(!reset_n)
                    state = S0;
                else
                    state = state_n;

            end

        //this section defines what the next state should be for each possible state. in this implementation, it simply rotates through each state automatically
        always @ (*)
            begin
                case(state)
                    S0: state_n = S1;
                    S1: state_n = S2;
                    S2: state_n = S3;
                    S3: state_n = S0;

                    default: state_n = S0;
                endcase
            end

        //this is the output definition section of the state machine. outputs are based on which state the state machine is currently on
        //hex values are used for the LEDs since there are 8 of them and hex values are an effective way of represiting this size
        //numbers are represented by the number of bits (not digits!), an apostrophe, the base system, then the value
        //common bases are binary, b, octal, o, decimal, d, hexadecimal, h
        always @ (*)
            begin
                if(state == S0)
                    select = 0;
                else if(state == S1)
                    select = 1;
                else if(state == S2)
                    select = 3;
                else
                    select = 4;
            end


endmodule

module clock_counter(
  input clk_i,      //often, "tags" are added to variables to denote what they do for the user
  input reset_n,    //here, 'i' is used for input and 'o' for the output, while 'n' specifies an active low signal ("not")

  output reg clk_o );

    reg [18:0] count = 0;                     //register stores the counter value so that it can be modified on a clock edge. register size needs to store as large of a number as the counter reaches
                                            //for this implementation, count must reach 415999, so 2^n >= 415999, n = 19

    always @ (posedge clk_i)
      begin
        count <= count + 1;                 //at every positive edge, the counter is increased by 1
        if(!reset_n)
          begin
            clk_o <= 0;
            count <= 0;                     //if reset (active low) is pushed, the counter is reset
          end
        else
          if(count >= 5000)                 //count value of greater than or equal to this value causes the output clock to be inverted. the resulting frequency will be input_frequency/(1+count_value)
            begin                           //for this implementation, a frequency of 5 Hz was desired, so 2.08e6/5 - 1 = 415999
              clk_o <= ~clk_o;
              count <= 0;                   //resets the counter after the output clock has been inverted
            end
      end


endmodule

module seven_segment(
  input [3:0] data,
  output reg [6:0] segments );

  always @(*)
    case( data ) // 7'bABCDEFG
      0 : segments = 7'b0000001;
      1 : segments = 7'b1001111;
      2 : segments = 7'b0010010;
      3 : segments = 7'b0000110;
      4 : segments = 7'b1001100;
      5 : segments = 7'b0100100;
      6 : segments = 7'b0100000;
      7 : segments = 7'b0001111;
      8 : segments = 7'b0000000;
      9 : segments = 7'b0001100;
      default : segments = 7'b1111111;
    endcase
```

```

endmodule

module fpga(
    input [7:0] number,
    input reset_n,
    output [2:0] select,
    output [6:0] segments );

    wire clock;
    wire clk;
    wire [3:0] ones;
    wire [3:0] tens;
    wire [3:0] hunds;
    wire [3:0] thous;
    wire [3:0] data;


    digit_parser dp(
        .number(number),
        .ones(ones),
        .tens(tens),
        .hunds(hunds),
        .thous(thous));

    output_select os(
        .select(select),
        .ones(ones),
        .tens(tens),
        .hunds(hunds),
        .thous(thous),
        .data(data));

    //This module is instantiated from another file, 'State_Machine.v'
    //It contains a Moore state machine that will take a clock and reset, and output LED combinations
    state_machine sm(
        .clk_i(clock),
        .reset_n(reset_n),
        .select(select));

    //This module is instantiated from another file, 'Clock_Counter.v'
    //It will take an input clock, slow it down based on parameters set inside of the module, and output the new
clock. Reset functionality is also built-in
    clock_counter cc(
        .clk_i(clk),
        .reset_n(reset_n),
        .clk_o(clock));

    seven_segment ss(
        .data(data),
        .segments(segments));

    OSCH #("2.08") osc_int (                    //"2.03" specifies the operating frequency, 2.03 MHz. Other clock
frequencies can be found in the MachX02's documentation
        .STDBY(1'b0),                           //Specifies active state
        .OSC(clk),                              //Outputs clock signal to 'clk' net
        .SEDSTDBY());

endmodule


```

Rhea Mae Edwards