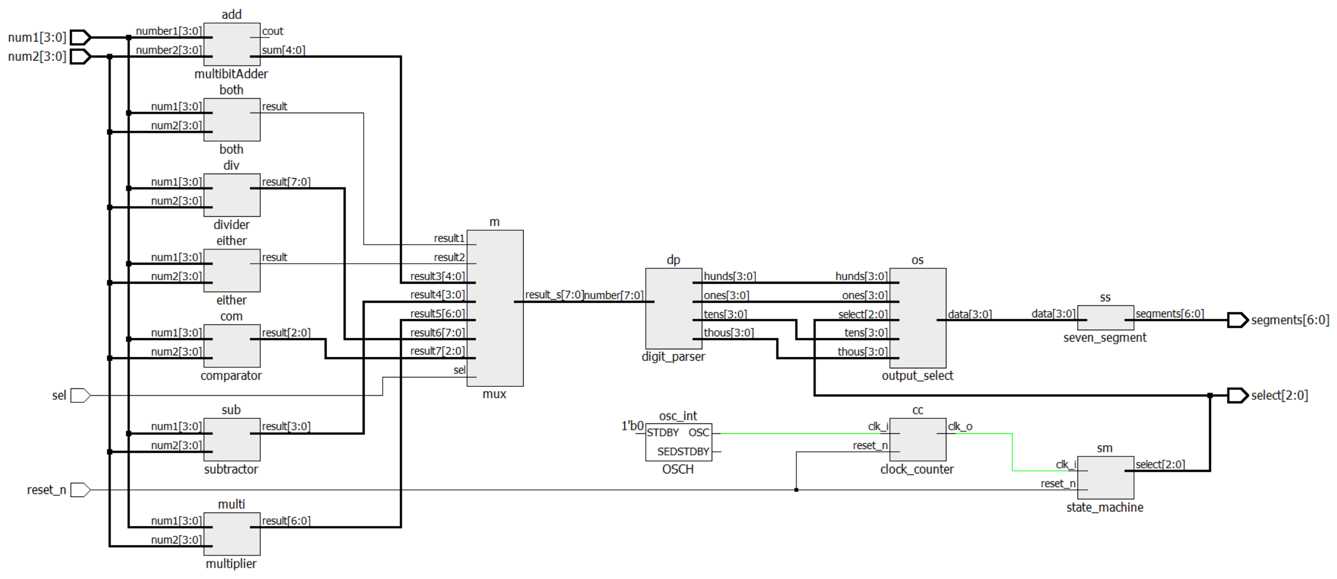
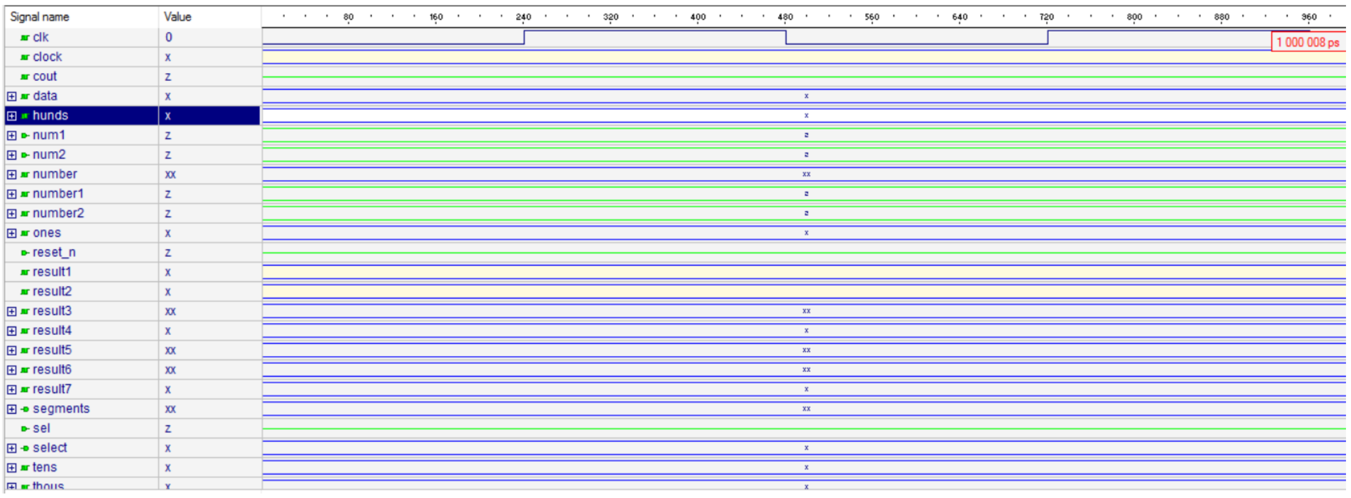
Rhea Mae Edwards
Student ID # 932-389-303

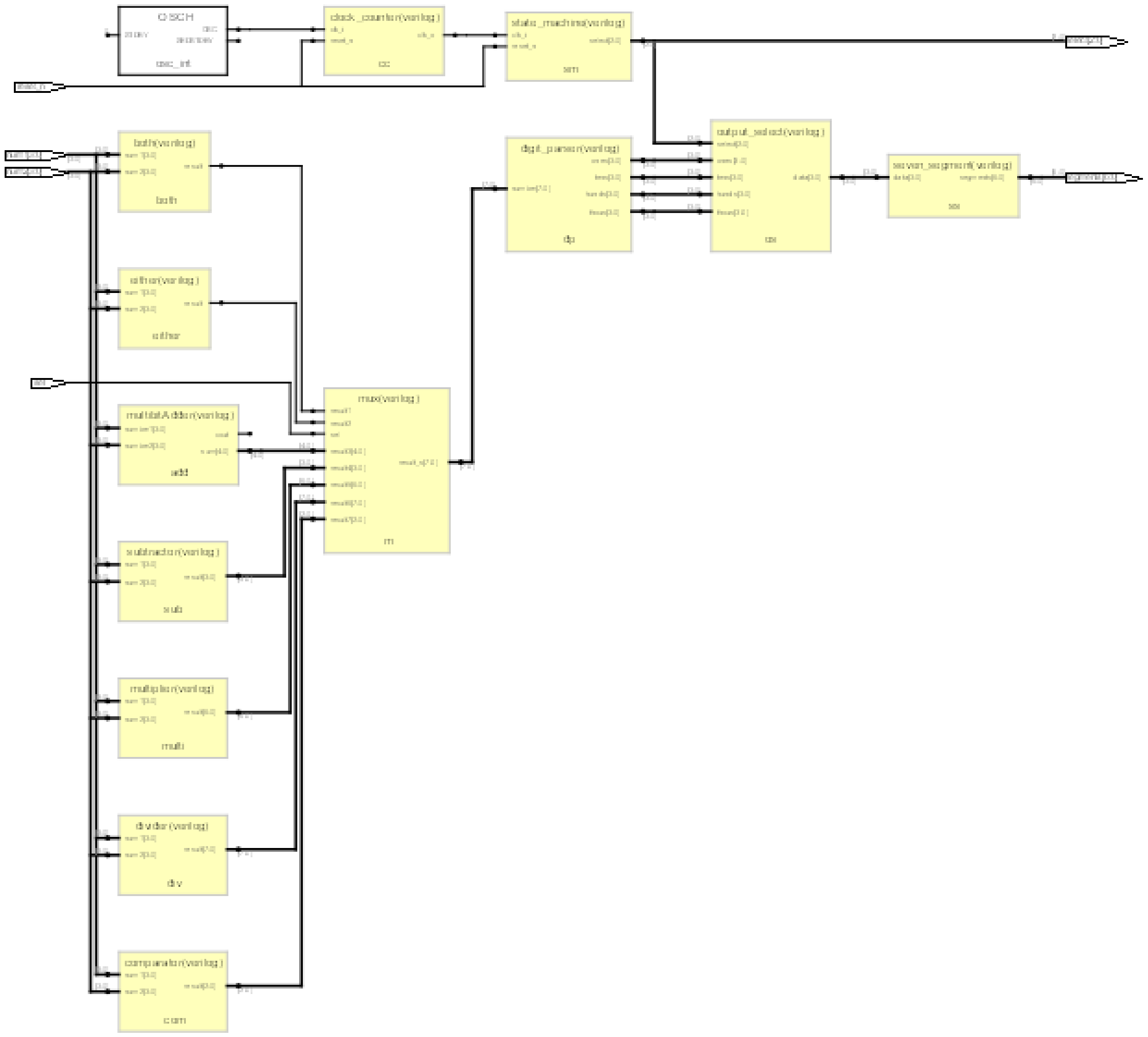
Section 6 Study Questions

1. Include a detailed block diagram of Section 6, the HDL Schematic, and a copy of the Verilog source.




```
Signal name   Value      80    160    240    320    400    480    560    640    720    800    880    960
clk           0                                                                      1 000 008 ps
clock         x
cout          z
data          x                                  x
hunds         x                                  x
num1          z                                  z
num2          z                                  z
number        xx                                 xx
number1       z                                  z
number2       z                                  z
ones          x                                  x
reset_n       z
result1       x
result2       x
result3       xx                                 xx
result4       x                                  x
result5       xx                                 xx
result6       xx                                 xx
result7       x                                  x
segments      xx                                 xx
sel           z
select        x                                  x
tens          x                                  x
thous         x                                  x
```

Rhea Mae Edwards
Student ID # 932-389-303



Rhea Mae Edwards
Student ID # 932-389-303

```
module both( //AND, Meagan
    input [3:0] num1,
    input [3:0] num2,
    output reg result);

    //assign result = num1&num2;

    always @ (*)
        begin
            result = num1&num2;
        end

endmodule

module either( //OR, Meagan
    input [3:0] num1,
    input [3:0] num2,
    output reg result);

    //assign result = num1 | num2;

    always @ (*)
        begin
            result = num1 | num2;
        end

endmodule

module adder( //Meagan
    input num1,
    input num2,
    input cin,
    output reg result,
    output reg cout);

    //wire p, g;

    reg p, g;

    //assign p = num1^num2;
    //assign g = num1&num2;

    //assign result = p^cin;
    //assign cout = g | (p&cin);

    always @ (*)
        begin
            p = num1^num2;
            g = num1&num2;

            result = p^cin;
```

```
                cout = g | (p&cin);
            end

endmodule

module multibitAdder(
    input [3:0] number1, number2,
    output [4:0] sum,
    output cout);

    wire adder1_o;
    wire adder2_o;
    wire adder3_o;

    adder Adder1(
            .num1(number1[0]),
            .num2(number2[0]),
            .cin(1'b0),
            .result(sum[0]),
            .cout(adder1_o));

    adder Adder2(
            .num1(number1[1]),
            .num2(number2[1]),
            .cin(adder1_o),
            .result(sum[1]),
            .cout(adder2_o));

    adder Adder3(
            .num1(number1[2]),
            .num2(number2[2]),
            .cin(adder2_o),
            .result(sum[2]),
            .cout(adder3_o));

    adder Adder4(
            .num1(number1[3]),
            .num2(number2[3]),
            .cin(adder3_o),
            .result(sum[3]),
            .cout(sum[4]));

endmodule

module subtractor( //Jeff
    input [3:0] num1,
    input [3:0] num2,
    output reg [3:0] result);

    always @ (*)
            begin
```

```
                if(num1 < num2)
                        result = num2 - num1;
                else if(num1 > num2)
                        result = num1 - num2;
        end

endmodule

module multiplier( //Rhea Mae
    input [3:0] num1,
    input [3:0] num2,
    output reg [6:0] result); //2N Result

    always @ (*)
        begin
                result = num1 * num2;
        end

endmodule

module divider( //Rhea Mae
    input [3:0] num1,
    input [3:0] num2,
    output reg [7:0] result); //2^N-1 Result


    always @ (*)
        begin
                if(num2 == 0)
                        result = 0;
                else
                        result = num1/num2;
        end

endmodule

module comparator( //Rhea Mae
    input [3:0] num1,
    input [3:0] num2,
    output reg [2:0] result);

    always @ (*)
        begin
                if(num1 == num2)
                        result = result + 1;
                if(num1 < num2)
                        result = result + 2;
                if(num1 > num2)
                        result = result + 4;
        end
```

```
endmodule

module mux( //Jeff
    input result1,
    input result2,
    input [4:0] result3,
    input [3:0] result4,
    input [6:0] result5,
    input [7:0] result6,
    input [2:0] result7,
    input sel,
    output reg [7:0] result_s);

    always @ (*)
        begin
            case(sel)
                0 : result_s = result1;
                1 : result_s = result2;
                2 : result_s = result3;
                3 : result_s = result4;
                4 : result_s = result5;
                5 : result_s = result6;
                6 : result_s = result7;
                default : result_s = 0;
            endcase
        end

endmodule

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)/128) % 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 = 3'b000;
                else if(state == S1)
                    select = 3'b001;
                else if(state == S2)
                    select = 3'b011;
                else
                    select = 3'b100;
            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 [3:0] num1,
    input [3:0] num2,
    input sel,
    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;
wire result1;
wire result2;
wire [4:0] result3; //wire [5:0] result3
wire [3:0] result4;
wire [6:0] result5;
wire [7:0] result6;
wire [2:0] result7;
wire [3:0] number1;
wire [3:0] number2;
wire cout;
wire [7:0] number;

both both(
		.num1(num1),
		.num2(num2),
		.result(result1));

either either(
		.num1(num1),
		.num2(num2),
		.result(result2));

multibitAdder add(
		.number1(num1),
		.number2(num2),
		.sum(result3),
		.cout(cout));

subtractor sub(
		.num1(num1),
		.num2(num2),
		.result(result4));

multiplier multi(
		.num1(num1),
		.num2(num2),
		.result(result5));

divider div(
		.num1(num1),
		.num2(num2),
		.result(result6));

comparator com(
		.num1(num1),
		.num2(num2),
		.result(result7));

mux m(
		.result1(result1),
		.result2(result2),
		.result3(result3),
		.result4(result4),
		.result5(result5),
		.result6(result6),
		.result7(result7),
		.sel(sel),
		.result_s(number));

	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
Student ID # 932-389-303

2. What was the toughest aspect of ECE 272? What should be changed or added to the ECE 272 manual to make this course better?

   Honestly every section of my ECE 272 experience was tough, because I was needed to attend two lab sections almost every week in order to get all of my labs completed. I definitely learned a lot and understood most of what was taught to me, but it did take up a lot of my time. Due to so much extra time and stress invested with taking this course as a Computer Science major, I'm kind of unsure what I should suggest that should be changed or added to make this course better… I mean maybe definitely warn computer science majors that they will most likely not have the needed equipment for the labs, and that they also have to pick up a lab kit before going to their first lab. Also they would have to learn some basic soldering skills especially to physically build their FPGA, along with common vocabulary that would mean nonsense to them initially when they run into a problem… Oh and also how they may not have the previous experience on how to even debug their piece of hardware… Which may seem impossible to do due to a lack of knowledge with using/dealing with hardware…

3. What would you like to explore further about Lattice Diamond or Digital Logic Design?

   Being a Computer Science major taking Electrical Computer Engineering courses, there is not much (or I don't know much) that I would like to explore further about Lattice Diamond or Digital Logic Design. But I really did have a beneficially, along with very time-consuming, experience with taking these courses.

4. What section of ECE 272 did you dislike the most? Why?

   … It was a stressful, but honestly a good experience.

5. What was your favorite section of ECE 272? Why?

   The final project, because I understood enough Verilog used in the labs by then, and I was able to figure out how to program and put together the needed components on how to create an ALU which the group I was in decided to create with the Verilog. Working in a group and knowing what I was doing by applying what I actually did remember of what I learned definitely made the final project the least stressful section for me.