Alejandro Torres
915207383

Lab 2: An Adder-Based Design

- I initialized my inputs and outputs in the top level module. We are outputting to 2 hex displays, so I instantiated 2 4-bit adders. Each output of the 4-bit adder was then inputted into the respective seg7 instantiated objects.

```
   ////////////// Adder ///////
   output                      co1,
   output                      co2,
   output           [3:0]      sum_1,
   output           [3:0]      sum_2

);

//=======================================================
//  Structural coding
//=======================================================

// Instantiate 2 copies of add4bit
add4bit add1(SW[3:0], 4'b0000, 0, co1, sum_1);
add4bit add2(SW[3:0], 4'b0001, 0, co2, sum_2);

// Instantiate 2 copies of the 7 segment display
seg7 hex0(sum_1,HEX1);
seg7 hex1(sum_2,HEX0);
```

Figure 1 : Top.v

- I instantiated 4 full adder objects to follow the design of the 4-bit ripple-carry adder provided in the lab manual. This would produce the sum that would be inputted into the seg7 module.

```
module add4bit(
    input [3:0]in1,
    input [3:0]in2,
    input ci,
    output co,
    output [3:0]sum
);

// Output of the first 3 adders
wire [2:0] fa_co;

fa inst1(in1[0], in2[0], ci, fa_co[0], sum[0]);
fa inst2(in1[1], in2[1], fa_co[0], fa_co[1], sum[1]);
fa inst3(in1[2], in2[2], fa_co[1], fa_co[2], sum[2]);
fa inst4(in1[3], in2[3], fa_co[2], co, sum[3]);

endmodule
```

Figure 2 : Add4Bit.v

- This module was pretty straightforward. The logic for a one bit adder was provided and was translated into verilog.

```
module fa(
    input a,
    input b,
    input ci,
    output wire co,
    output wire sum
);

assign sum = a ^ b ^ ci;
assign co = (a&b) | (a&ci) | (b&ci);

endmodule
```

Figure 3 : fa.v

- There is an always block that is always executing when the output of the 4-bit adder is changed. There are 16 possible cases, so depending on the case that is satisfied the output of the hex display should be changed. Figure 4 does not show all cases.

Figure 4 : seg7.v

```
module seg7(
    input [3:0] in,
    output reg [7:0] out
);

always @(*) begin
    case(in)
        // 0
        4'b0000: begin
            out = 8'b11000000;
        end

        // 1
        4'b0001: begin
            out = 8'b11111001;
        end

        // 2
        4'b0010: begin
            out = 8'b10100100;
        end
```