Frank I Gomez
4/22/21
Lab 3, Section D.
ID: 1650550
figomez
CSE 100L

Lab 3 Write Up

1. Description

The purpose of this lab was to expand on our full-adder from Lab 2, this time adding two 8-bit two's complement (positive or negative) numbers together, in addition to subtracting them. We were then to represent this value on our seven segment displays, using 2 of the 4 displays this time. All of this was constructed with multiplexers, specifically 2-1, 4-1, and 8-1. Finally, we had to display them on the physical boards this time. Buses were also used for the first time in this lab.

2. Design

- Top Level

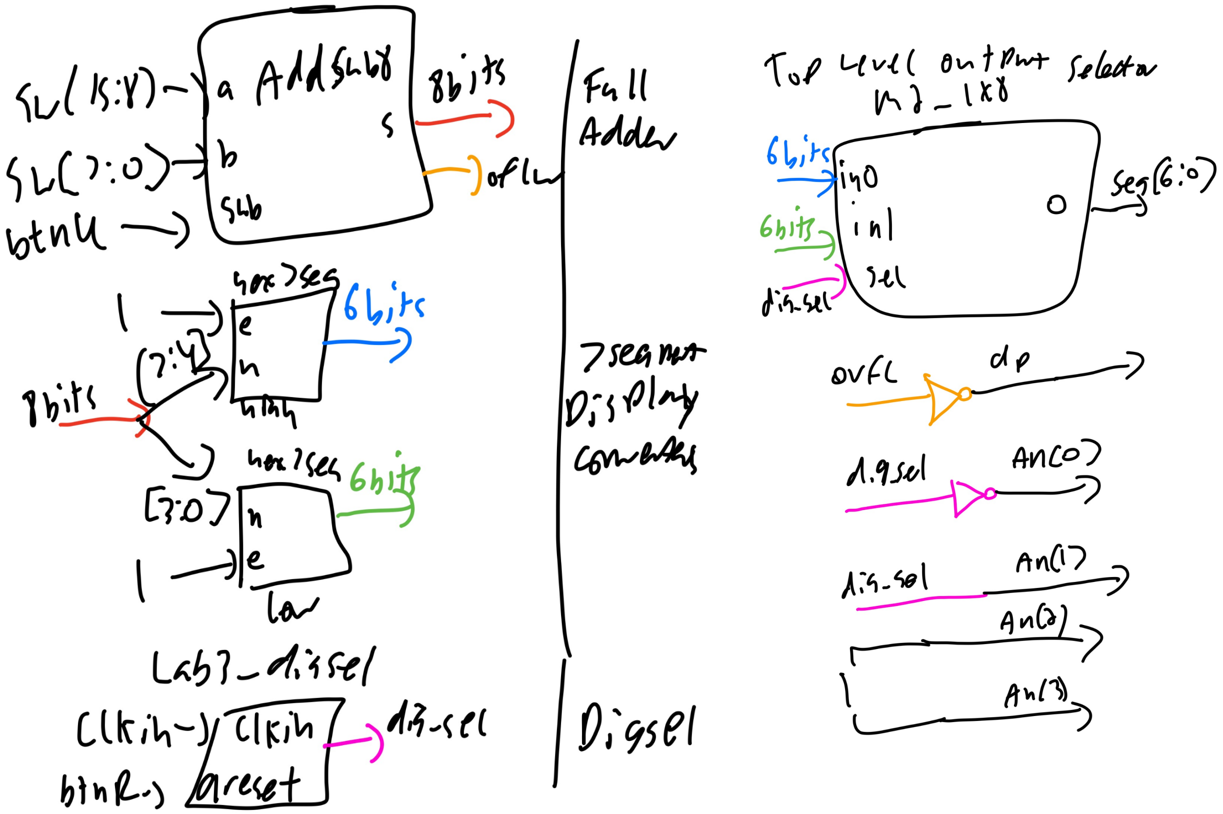


Diagram 1

There's a lot going on in the top level this time around. Starting from the top left, there's the full adder, with an extra input btnU, which allows subtraction. Following this, the 8 bits from the full adder are then split into 2 4 bit busses, which are then converted into their respective seven segment display values. The main point of the digsel module is to select between these two display values in the final, top level selector, where dig_sel will alternate between the high and low components of the 7 segment display, to display them on the board. Overflow is also inverted here, to fit with the board's inverted display. AN2 and AN3 are high, the same as before.

- m4_1e

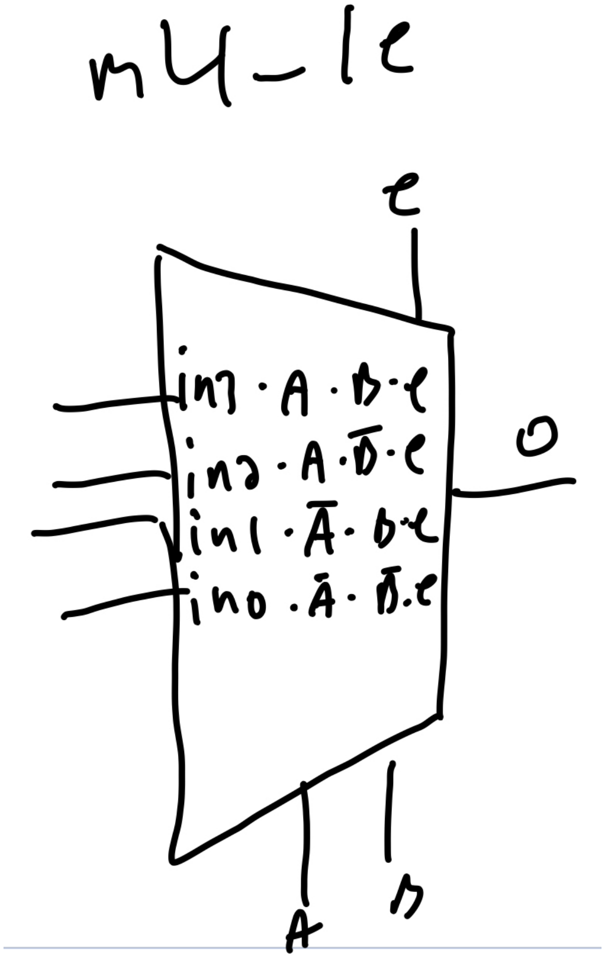


Diagram 2

The 4-1 mux selects between 4 possible outputs, based on the values of the two selectors, A and B, and the input bits in the bus in[3:0]. It’s just a sum of products at the end of it, so only one case needs to pass for the entire thing to pass. The enabler bit, e, forces the result to always be 0, when it is low.

- m8_1e

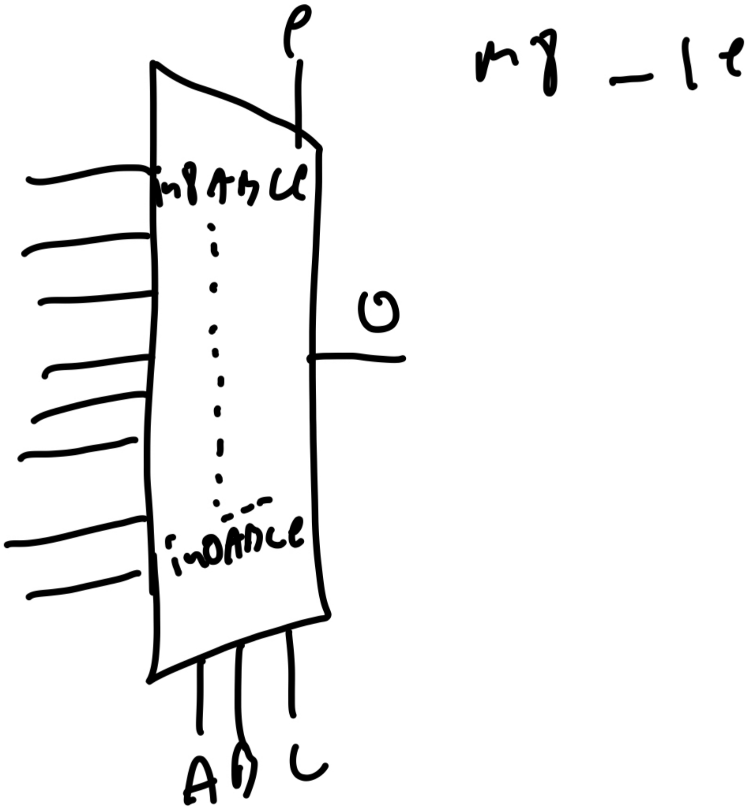


Diagram 3

The 8-1 mux follows the same process as the 4-1 mux, just with an 8 bit input, alongside 3 selector bits.

- m2_1x8

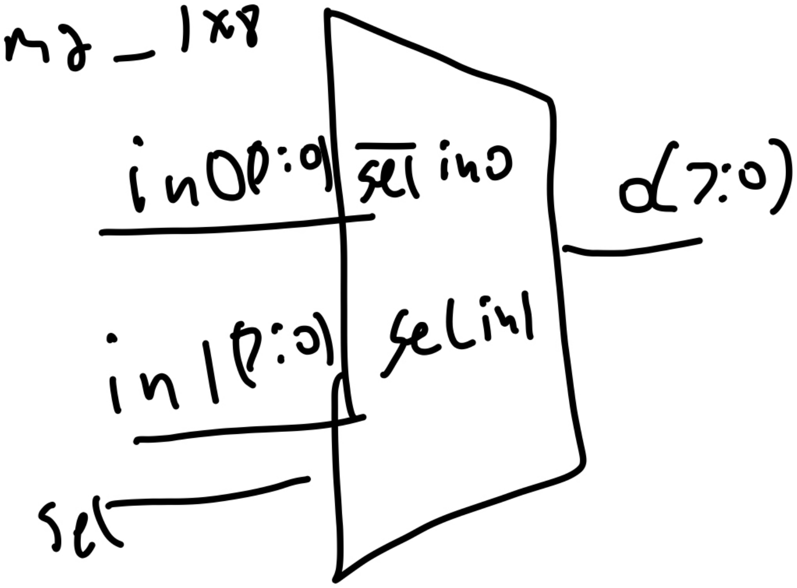


Diagram 4

The 2-1 mux, on the other hand, follows a different process. It has 3 inputs, 2 8-bit buses, and a selector bit. This time around, the output is also an 8-bit bus, and will be equal to either in0 or in1, entirely dependent on the value of sel. If sel is low, the output will be in0, and likewise for high and in1.

- Full_Adder

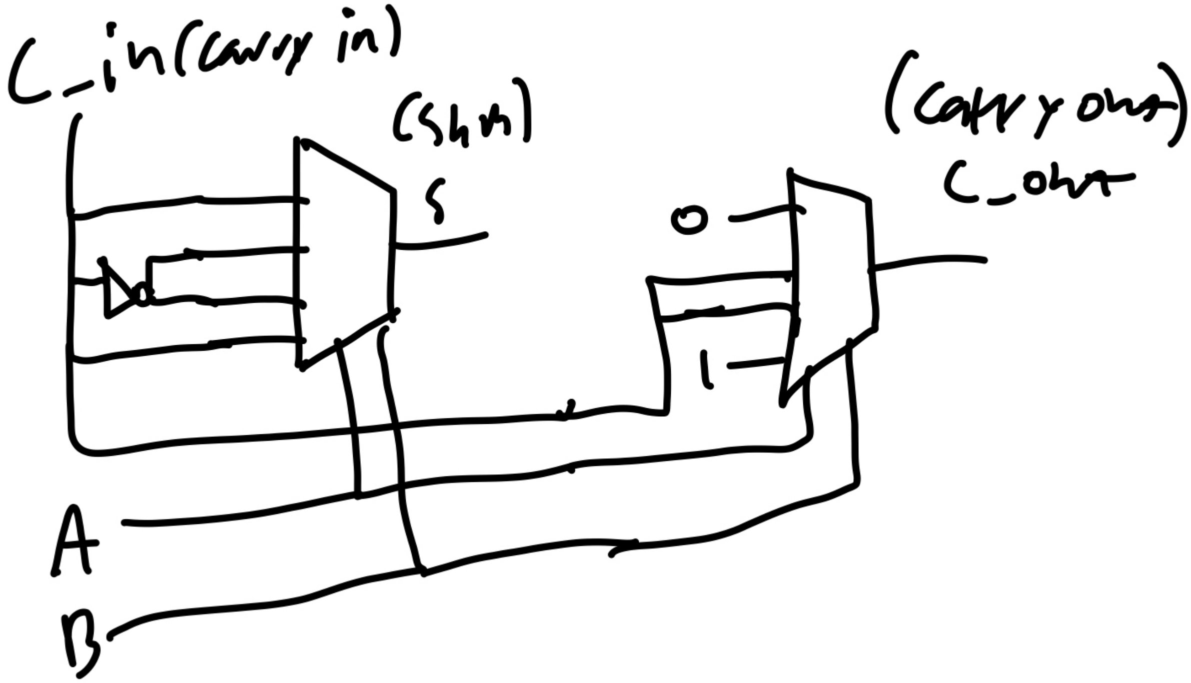


Diagram 5

In all regards, the full adder for lab 3 was the same as the full adder for lab 2, with relatively the same underlying logic, only using 2 4-1 muxes instead of XOR gates. A 4-1 mux with the inverse of the input at the 1st and 2nd positions is essentially the same as a 3 input XOR gate. Carry out is similar, receiving a 0 and 1 at the 0th and 3rd positions, and the input at the 1st and 2nd.

- AddSub8

AddSub8 is just a series of 8 full adders, adding the two full, 8-bit inputs. If subtraction is enabled, the second variable, B, will be converted to two's complement, by inverting its values and adding a 1 at the first carry in. Overflow is also handled here, a simple logic equation based around the values and carry_outs of the 2 numbers at the 6th and 7th bit positions.

- hex7seg

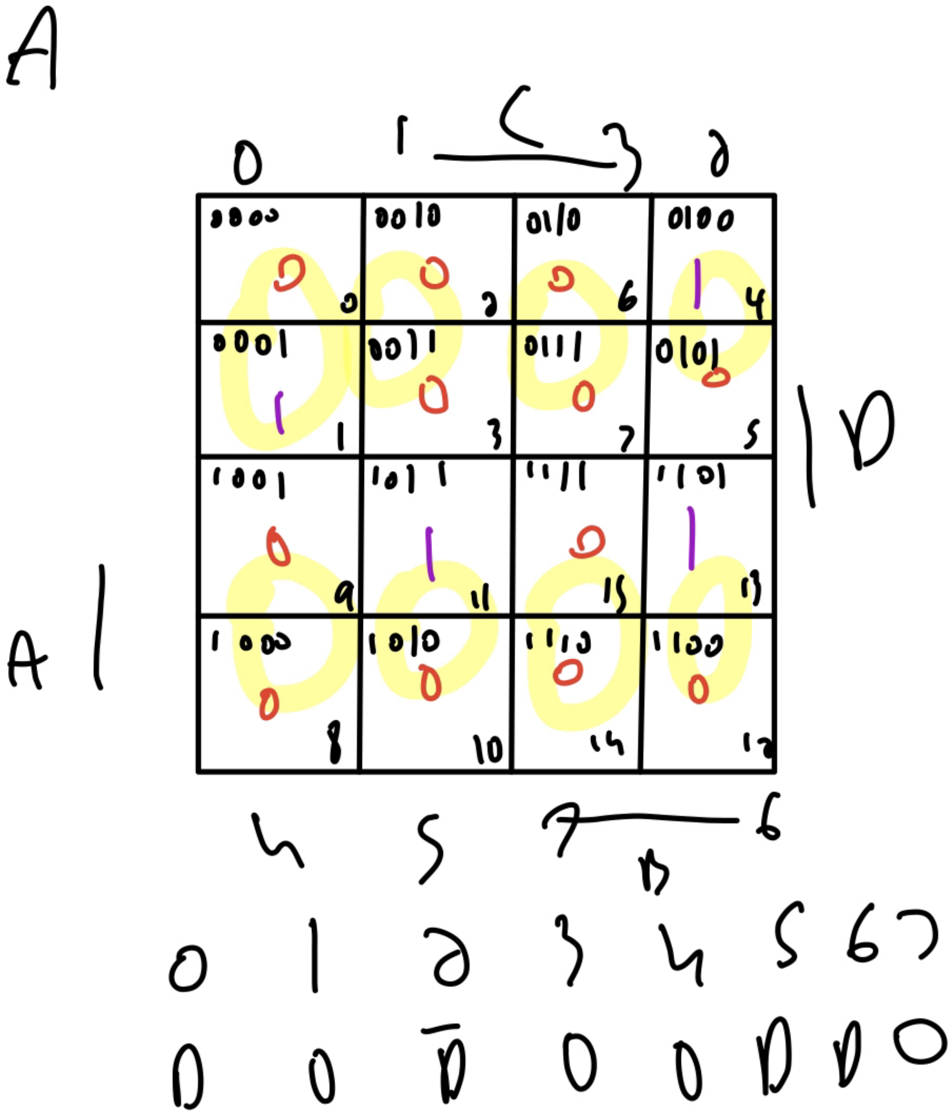


Diagram 6

The last coded section, hex7seg, takes in a 4-bit input, and obtains the respective 7-segment hex display. The logic for this section was obtained by creating 7 K-maps, each representing one of the 7 segments, which is then used as the 8-bit input for the respective 8-1 mux seen early. All 16 hex values, 0-F, can be obtained through this.

- lab3_digsel

Finally, lab3_digsel. This file was given to us, and its function is providing a single bit output that rapidly switches between high and low, allowing the 7-segment display to display 2-1 mux to alternate which 4 bits are being displayed on the display.

3. Testing and Simulation

Similarly to lab 2, verilog’s simulation feature was used to test every important value of this lab, confirming that each module was functioning properly, or displaying when they weren’t. All the muxes functioned properly on their first try, and the 7 segment display only had a single incorrect value that was easily fixed. The source of the most problems, unfortunately, was the full adder, particularly, the overflow. Coming up with all possible cases of overflow was difficult at first, only getting easier when I realized I could obtain all of them with 3-bit addition, which could be done mentally. Finally, once this was complete, it was programmed onto the board, where the overflow bit was, again, wrong, as I had forgotten that high means off, and vice versa, for the boards. This was quickly fixed.

4. Results

- How fast the signal **dig_sel** is oscillating.

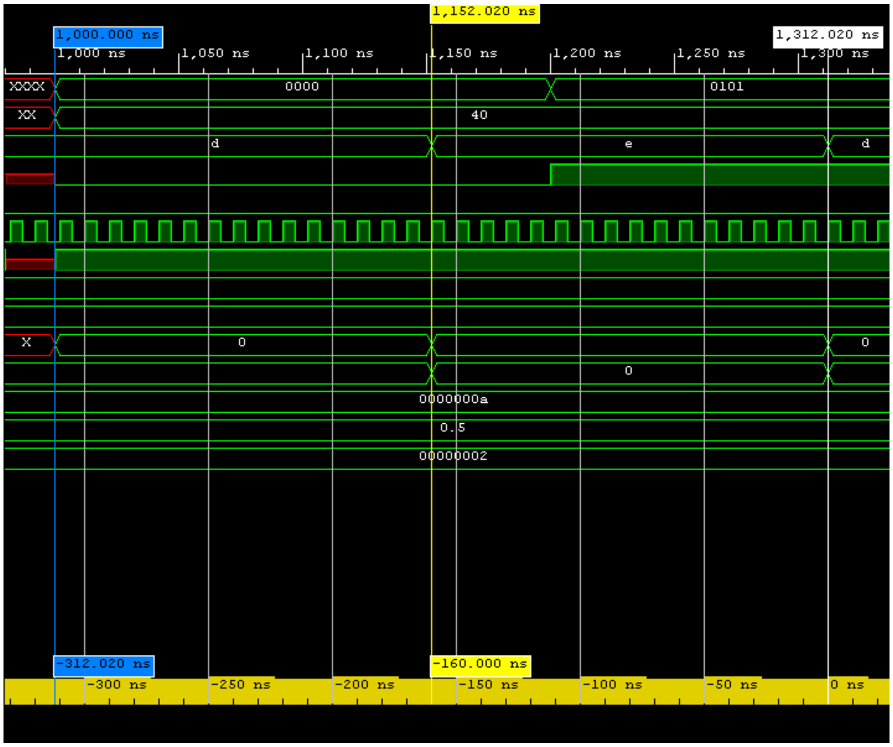


Diagram 7

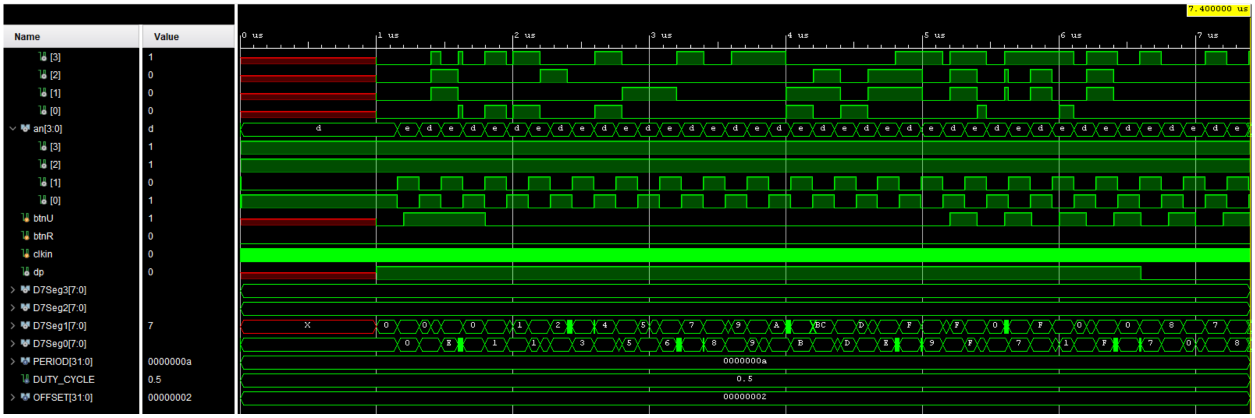
Placing markers at the start and end of a full oscillation period, it can be estimated that the period of dig_sel is 312.2 nanoseconds, flipping every 156.1 nanoseconds. This means it oscillates at a frequency of 3.2 Megahertz.

5. Conclusion

There was one more error in the program, not stated earlier, as it was less of a code problem, and more of a conceptual one. I had forgotten that, in verilog, the value 1 is different from the value 1’b1, as “1” by itself is an integer. The actual result of this is that the first simulation run that involved these values resulted in every expected output being incorrect, and about an hour and a half of troubleshooting, until randomly swapping from 1 to 1’b1 while trying to figure out why it was not working. If there was anything I would change on a subsequent run of this lab, would be to remind students that the variables are in fact, different. Otherwise, I learned how to use busses, multiplexers, and how to run the simulator for specific increments of time.

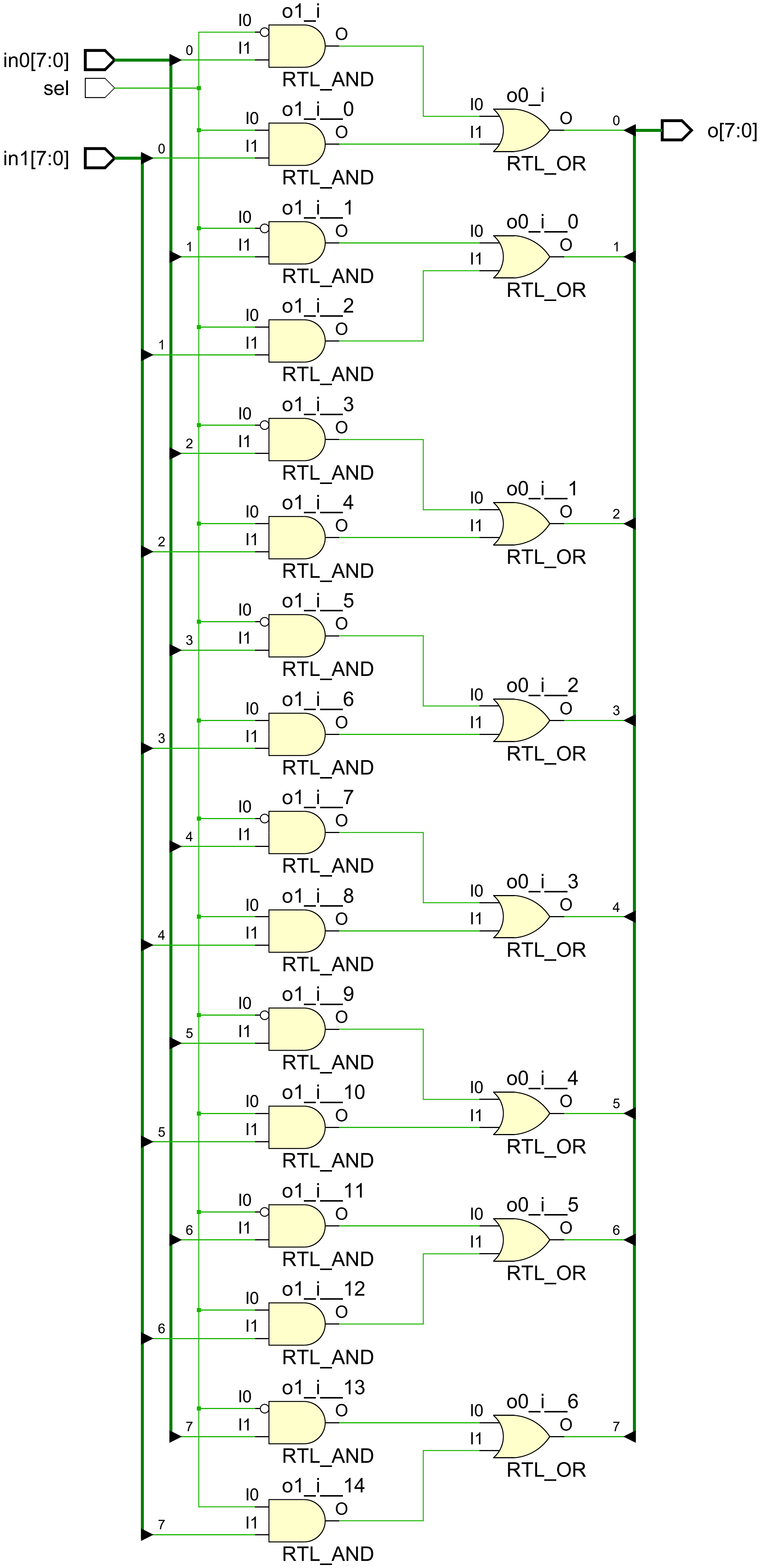
6. Appendix

(Note: Waveform Viewer cannot be printed to PDF, and as such, it is added here as a screenshot.)



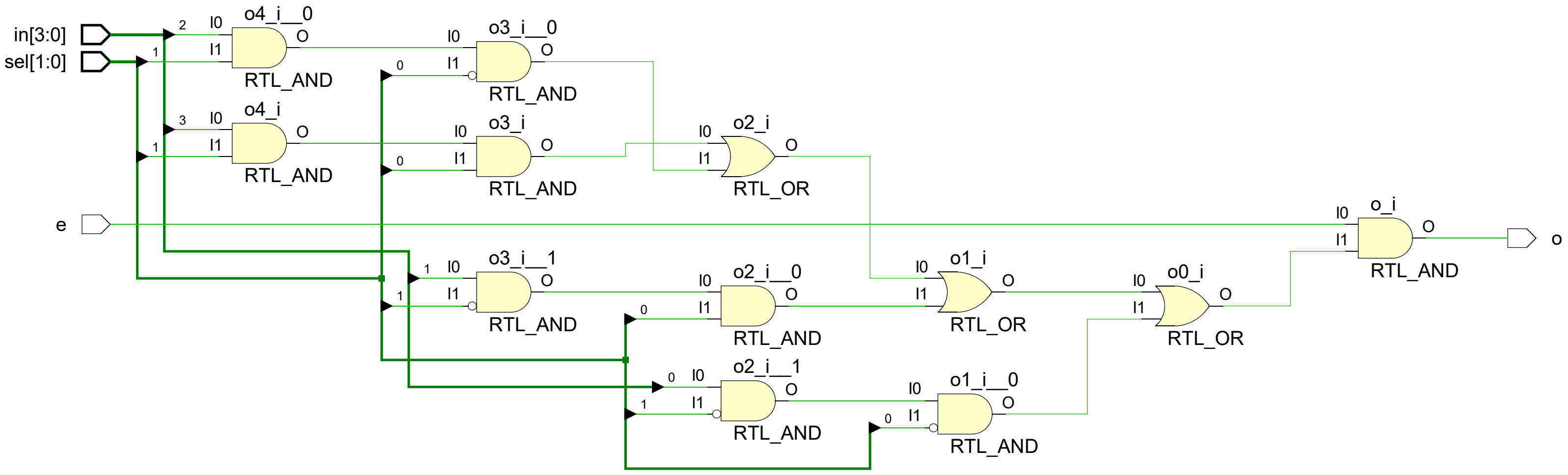
Name	Value
[3]	1
[2]	0
[1]	0
[0]	0
an[3:0]	d
[3]	1
[2]	1
[1]	0
[0]	1
btnU	1
btnR	0
clkin	0
dp	0
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	7
D7Seg0[7:0]	
PERIOD[31:0]	0000000a
DUTY_CYCLE	0.5
OFFSET[31:0]	00000002

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/18/2021 04:17:22 PM
// Design Name: 
// Module Name: m2_1x8
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );
    assign o[0] = (~sel & in0[0]) | (sel & in1[0]);
    assign o[1] = (~sel & in0[1]) | (sel & in1[1]);
    assign o[2] = (~sel & in0[2]) | (sel & in1[2]);
    assign o[3] = (~sel & in0[3]) | (sel & in1[3]);
    assign o[4] = (~sel & in0[4]) | (sel & in1[4]);
    assign o[5] = (~sel & in0[5]) | (sel & in1[5]);
    assign o[6] = (~sel & in0[6]) | (sel & in1[6]);
    assign o[7] = (~sel & in0[7]) | (sel & in1[7]);
endmodule
```


in0[7:0]
sel
in1[7:0]
o[7:0]
o1_i
o1_i__0
o1_i__1
o1_i__2
o1_i__3
o1_i__4
o1_i__5
o1_i__6
o1_i__7
o1_i__8
o1_i__9
o1_i__10
o1_i__11
o1_i__12
o1_i__13
o1_i__14
RTL_AND
o0_i
o0_i__0
o0_i__1
o0_i__2
o0_i__3
o0_i__4
o0_i__5
o0_i__6
RTL_OR
I0
I1
O
0
1
2
3
4
5
6
7

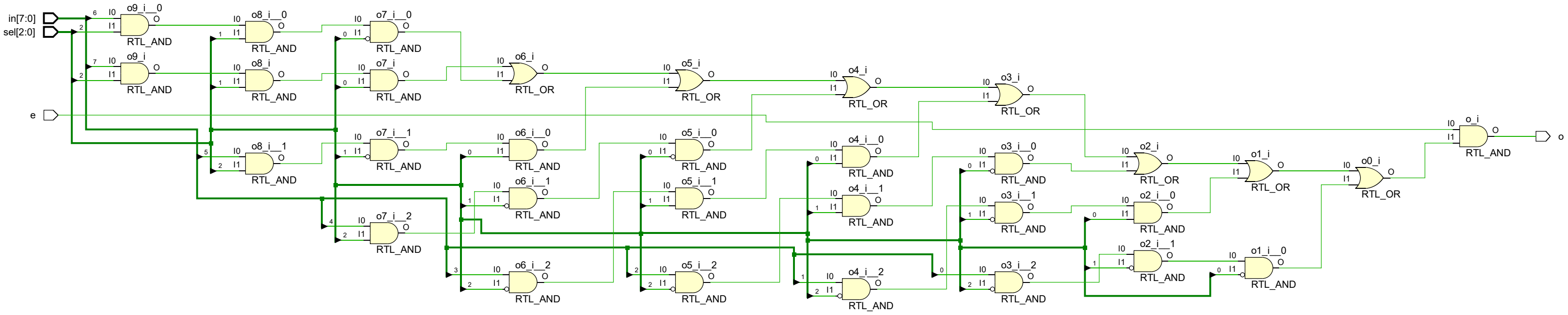
```
module m2_1x8_testbench();

   reg input1;
   reg [7:0] input_bus0;
   reg [7:0] input_bus1;
   wire [7:0] output_bus1;

   m2_1x8   UUT   ( .in0(input_bus0), .in1(input_bus1), .sel(input1), .o(output_bus1)

// below is the "stimuli," the values for the inputs
// be sure to select a range of inputs that will fully exercise your design

    initial
    begin

        //-------------  Current Time:  0ns
        input1 = 1'b0;
        input_bus0 = 8'b00000000;
        input_bus1 = 8'b11111111;
        #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        input1=1'b1;
        #100; // -------------  Current Time:  200ns
        input1=1'b0;
        input_bus0 = 8'b10000000;
        input_bus1 = 8'b00000001;
        #100; // -------------  Current Time:  300ns
        input1=1'b1;
        #100; // -------------  Current Time:  400ns
        input1=1'b0;
        input_bus0 = 8'b01010101;
        input_bus1 = 8'b10101010;
        #100; // -------------  Current Time:  500ns
        input1=1'b1;
    end
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/18/2021 04:17:22 PM
// Design Name: 
// Module Name: m4_1e
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module m4_1e(
    input [3:0] in,
    input [1:0] sel,
    input e,
    output o
    );
    assign o = e & ((in[3] & sel[1] & sel[0])
    | (in[2] & sel[1] & ~sel[0])
    | (in[1] & ~sel[1] & sel[0])
    | (in[0] & ~sel[1] & ~sel[0]));
endmodule
```


in[3:0]
sel[1:0]
e
o
2
1
3
1
0
0
1
1
0
0
1
0
o4_i__0
RTL_AND
o4_i
RTL_AND
o3_i__0
RTL_AND
o3_i
RTL_AND
o3_i__1
RTL_AND
o2_i
RTL_OR
o2_i__0
RTL_AND
o2_i__1
RTL_AND
o1_i
RTL_OR
o1_i__0
RTL_AND
o0_i
RTL_OR
o_i
RTL_AND
I0
I1
O

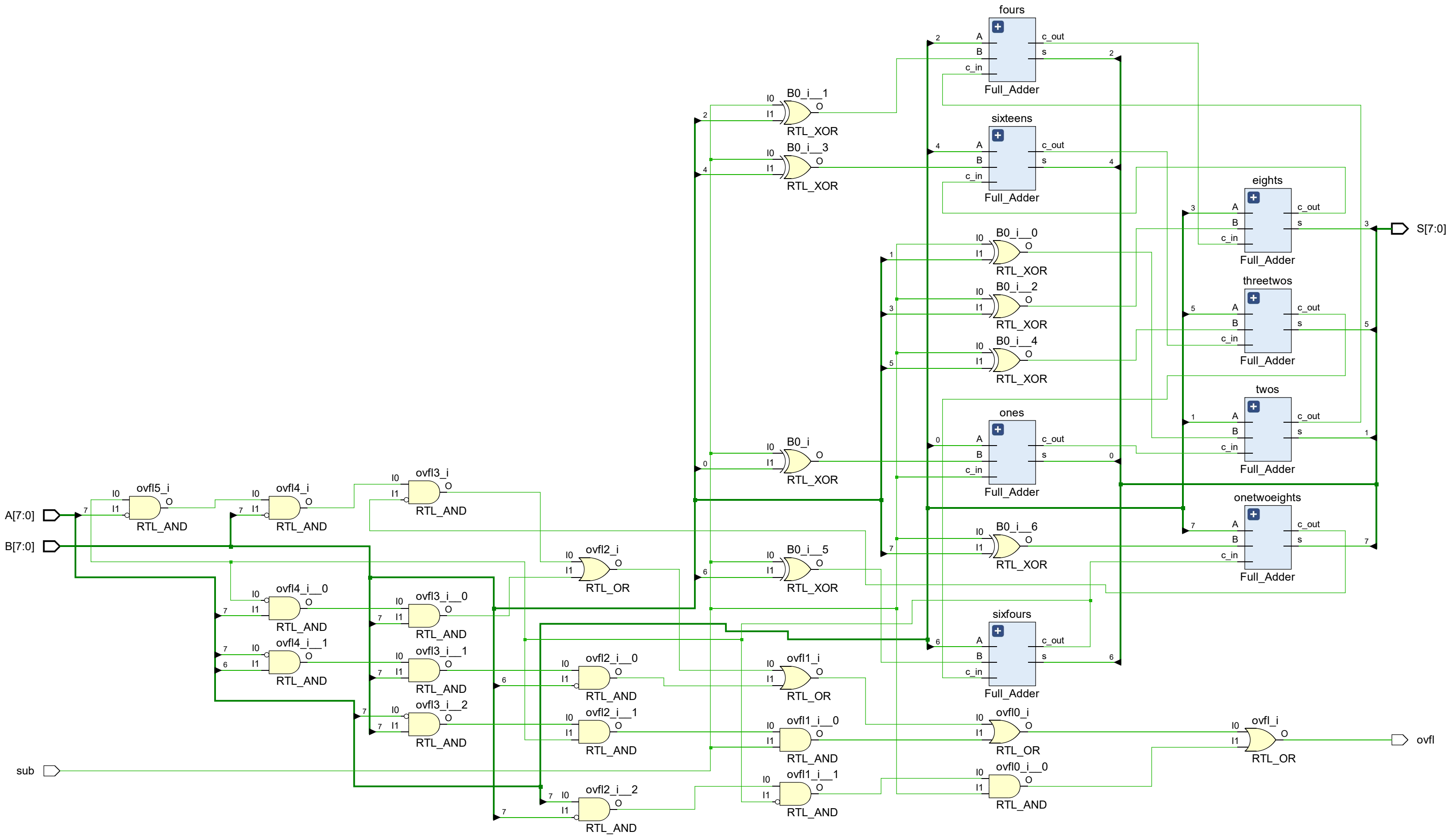
```
module m4_1e_testbench();
   reg input3;
   reg [3:0] input_bus;
   reg [1:0] input_bus2;
   wire output1;
   //wire [3:0] output_bus1;

   m4_1e   UUT   ( .in(input_bus), .sel(input_bus2), .e(input3),
                      .o(output1));

// below is the "stimuli," the values for the inputs
// be sure to select a range of inputs that will fully exercise your design

    initial
    begin

        //-------------  Current Time:  0ns
        //input1=1'b0;
        //input2=1'b0;
        input3=1'b1;
        input_bus = 4'b0000;
        input_bus2 = 2'b00;
        #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        input_bus = 4'b1100;
        input_bus2 = 2'b00;
        #100; // -------------  Current Time:  200ns
        input_bus2 = 2'b01;
        #100; // -------------  Current Time:  300ns
        input_bus2 = 2'b10;
        #100; // -------------  Current Time:  400ns
        input_bus2 = 2'b11;
        #100; // -------------  Current Time:  500ns
        input_bus = 4'b1111;
        input_bus2 = 2'b00;
        #100; // -------------  Current Time:  600ns
        input_bus2 = 2'b01;
        #100; // -------------  Current Time:  700ns
        input_bus2 = 2'b10;
        #100; // -------------  Current Time:  800ns
        input_bus2 = 2'b11;
        #100; // -------------  Current Time:  900ns
        input3=1'b0;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/18/2021 04:17:22 PM
// Design Name: 
// Module Name: m8_1e
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e,
    output o
    );
    assign o = e & ((in[7] & sel[2] & sel[1] & sel[0])
    | (in[6] & sel[2] & sel[1] & ~sel[0])
    | (in[5] & sel[2] & ~sel[1] & sel[0])
    | (in[4] & sel[2] & ~sel[1] & ~sel[0])
    | (in[3] & ~sel[2] & sel[1] & sel[0])
    | (in[2] & ~sel[2] & sel[1] & ~sel[0])
    | (in[1] & ~sel[2] & ~sel[1] & sel[0])
    | (in[0] & ~sel[2] & ~sel[1] & ~sel[0]));
endmodule
```


in[7:0]
sel[2:0]
e
o9_i__0
RTL_AND
o9_i
RTL_AND
o8_i__0
RTL_AND
o8_i
RTL_AND
o8_i__1
RTL_AND
o7_i__0
RTL_AND
o7_i
RTL_AND
o7_i__1
RTL_AND
o7_i__2
RTL_AND
o6_i
RTL_OR
o6_i__0
RTL_AND
o6_i__1
RTL_AND
o6_i__2
RTL_AND
o5_i
RTL_OR
o5_i__0
RTL_AND
o5_i__1
RTL_AND
o5_i__2
RTL_AND
o4_i
RTL_OR
o4_i__0
RTL_AND
o4_i__1
RTL_AND
o4_i__2
RTL_AND
o3_i
RTL_OR
o3_i__0
RTL_AND
o3_i__1
RTL_AND
o3_i__2
RTL_AND
o2_i
RTL_OR
o2_i__0
RTL_AND
o2_i__1
RTL_AND
o1_i
RTL_OR
o1_i__0
RTL_AND
o0_i
RTL_OR
o_i
RTL_AND
o

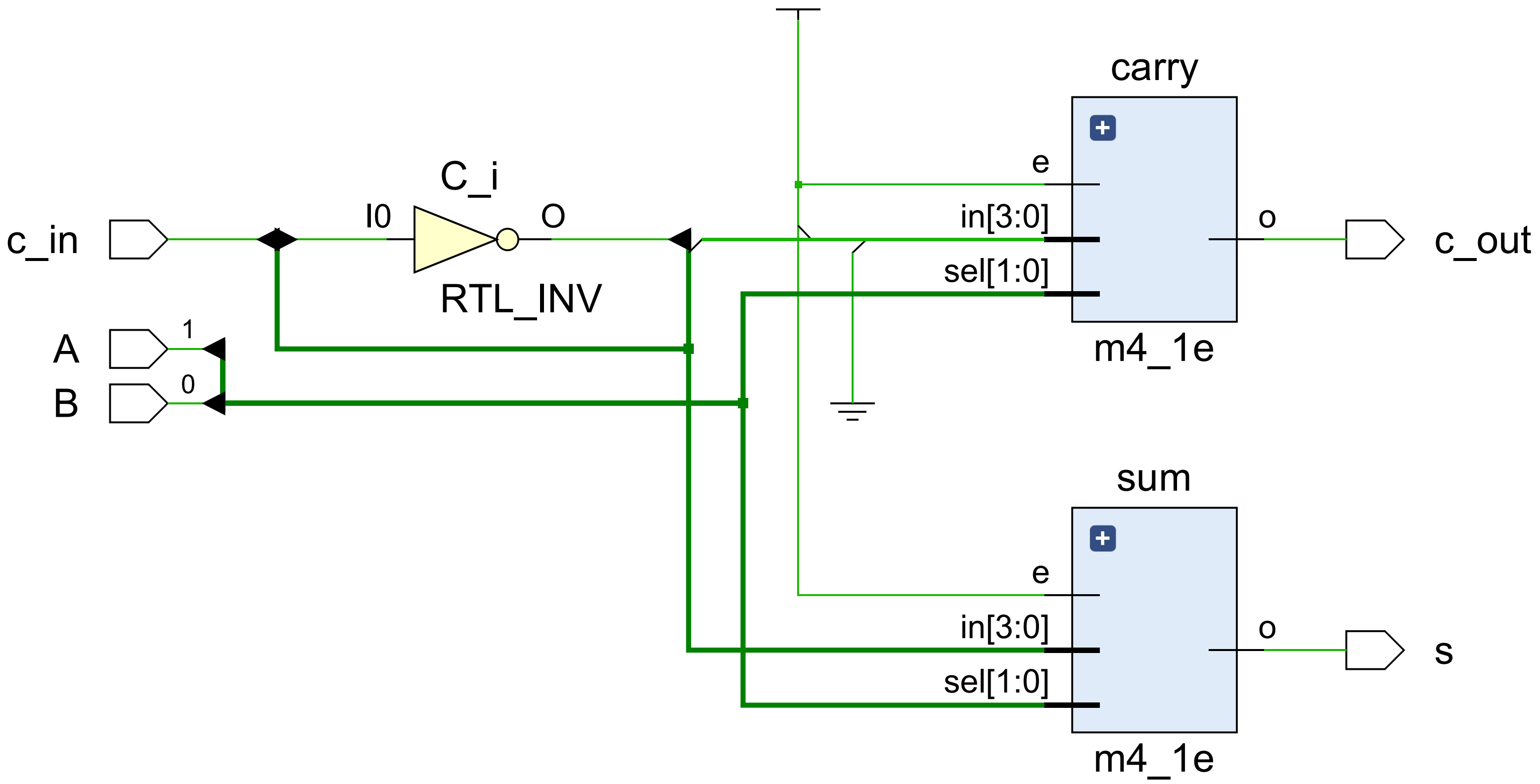
```
module m8_1e_testbench();

   reg input1;
   reg [7:0] input_bus;
   reg [2:0] input_bus2;
   wire output1;

   m8_1e   UUT   ( .in(input_bus), .sel(input_bus2), .e(input1), .o(output1));

// below is the "stimuli," the values for the inputs
// be sure to select a range of inputs that will fully exercise your design

    initial
    begin

        //-------------  Current Time:  0ns
        input1=1'b1;
        input_bus = 8'b10101010;
        input_bus2 = 3'b000;
    #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        input_bus2 = 3'b001;
        #100; // -------------  Current Time:  200ns
        input_bus2 = 3'b010;
        #100; // -------------  Current Time:  300ns
        input_bus2 = 3'b011;
        #100; // -------------  Current Time:  400ns
        input_bus2 = 3'b100;
        #100; // -------------  Current Time:  500ns
        input_bus2 = 3'b101;
        #100; // -------------  Current Time:  600ns
        input_bus2 = 3'b110;
        #100; // -------------  Current Time:  700ns
        input_bus2 = 3'b111;
        #100; // -------------  Current Time:  800ns
        input_bus2 = 3'b001;
        input1=1'b0;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/18/2021 06:09:49 PM
// Design Name:
// Module Name: AddSub8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module AddSub8(
    input [7:0] A,
    input [7:0] B,
    input sub,
    output [7:0] S,
    output ovfl
    );
    wire c0, c1, c2, c3, c4, c5, c6, c7;
    Full_Adder ones (.A(A[0]), .B(sub ^ B[0]), .c_in(sub), .s(S[0]), .c_out(c0));
    Full_Adder twos (.A(A[1]), .B(sub ^ B[1]), .c_in(c0), .s(S[1]), .c_out(c1));
    Full_Adder fours (.A(A[2]), .B(sub ^ B[2]), .c_in(c1), .s(S[2]), .c_out(c2));
    Full_Adder eights (.A(A[3]), .B(sub ^ B[3]), .c_in(c2), .s(S[3]), .c_out(c3));
    Full_Adder sixteens (.A(A[4]), .B(sub ^ B[4]), .c_in(c3), .s(S[4]), .c_out(c4));
    Full_Adder threetwos (.A(A[5]), .B(sub ^ B[5]), .c_in(c4), .s(S[5]), .c_out(c5));
    Full_Adder sixfours (.A(A[6]), .B(sub ^ B[6]), .c_in(c5), .s(S[6]), .c_out(c6));
    Full_Adder onetwoeights (.A(A[7]), .B(sub ^ B[7]), .c_in(c6), .s(S[7]),
.c_out(c7));
    assign ovfl = (c6 & ~A[7] & ~B[7] & ~c7) | (~c6 & A[7] & B[7]) | (~A[7] & A[6] &
B[7] & ~B[6]) | (~A[7] & B[7] & c6 & sub) | (A[7] & ~B[7] & ~c6 & sub);
    //The third case is to handle >=64 - (<-65).
    //Fourth case is for 64 - (-64).
    //Fifth case is for -128 - 8.
endmodule
```

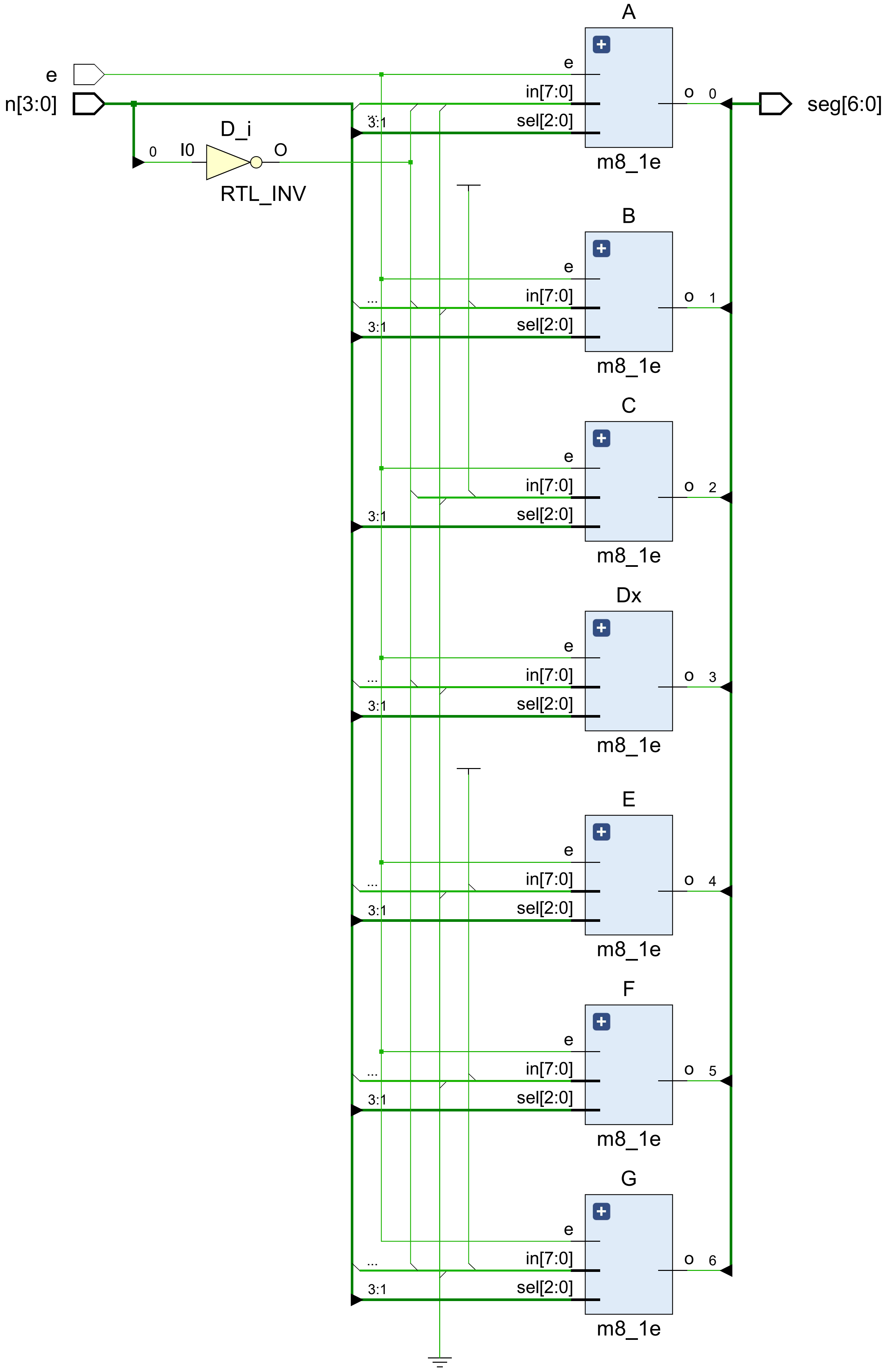

A[7:0]
B[7:0]
sub
ovfl5_i
RTL_AND
ovfl4_i
RTL_AND
ovfl3_i
RTL_AND
ovfl4_i__0
RTL_AND
ovfl4_i__1
RTL_AND
ovfl3_i__0
RTL_AND
ovfl3_i__1
RTL_AND
ovfl3_i__2
RTL_AND
ovfl2_i
RTL_OR
ovfl2_i__0
RTL_AND
ovfl2_i__1
RTL_AND
ovfl2_i__2
RTL_AND
B0_i__1
RTL_XOR
B0_i__3
RTL_XOR
B0_i
RTL_XOR
B0_i__5
RTL_XOR
ovfl1_i
RTL_OR
ovfl1_i__0
RTL_AND
ovfl1_i__1
RTL_AND
fours
Full_Adder
sixteens
Full_Adder
B0_i__0
RTL_XOR
B0_i__2
RTL_XOR
B0_i__4
RTL_XOR
ones
Full_Adder
B0_i__6
RTL_XOR
sixfours
Full_Adder
ovfl0_i
RTL_OR
ovfl0_i__0
RTL_AND
eights
Full_Adder
threetwos
Full_Adder
twos
Full_Adder
onetwoeights
Full_Adder
ovfl_i
RTL_OR
S[7:0]
ovfl
I0
I1
O
A
B
c_in
c_out
s

```
module AddSub8_testbench();

   reg input1;
   reg [7:0] input_bus1;
   reg [7:0] input_bus2;
   wire overflow;
   wire [7:0] output_bus;

   AddSub8   UUT   (.A(input_bus1), .B(input_bus2), .sub(input1), .S(output_bus),
.ovfl(overflow));

// below is the "stimuli," the values for the inputs
// be sure to select a range of inputs that will fully exercise your design

    initial
    begin

        //-------------  Current Time:  0ns
        input_bus1 = 8'b00000001;
        input_bus2 = 8'b00000001;
        input1 = 1'b0;
        //Simple test, 1+1.
        #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        input_bus1 = 8'b01111111;
        input_bus2 = 8'b00000001;
        input1 = 1'b0;
        //Overflow test.
        #100; // -------------  Current Time:  200ns
        input_bus1 = 8'b00000100;
        input_bus2 = 8'b00000001;
        input1 = 1'b1;
        //Simple subtraction test, 4 - 1.
        #100; // -------------  Current Time:  300ns
        input_bus1 = 8'b11111111;
        input_bus2 = 8'b00000100;
        input1 = 1'b0;
        //A harder test, -1 + 4.
        #100; // -------------  Current Time:  400ns
        input_bus1 = 8'b01111111;
        input_bus2 = 8'b10000001;
        input1 = 1'b0;
        //127 + -127. Zero and positive + negative.
        #100; // -------------  Current Time:  500ns
        input_bus1 = 8'b11111000;
        input_bus2 = 8'b11110000;
        input1 = 1'b1;
        //A complicated test. -8 - (-16).
        #100; // -------------  Current Time:  600ns
        input_bus1 = 8'b10000001;
        input_bus2 = 8'b11111110;
        input1 = 1'b0;
        //Negative overflow, -127 + -2.
        #100; // -------------  Current Time:  700ns
        input_bus1 = 8'b11111111;
        input_bus2 = 8'b11111111;
        input1 = 1'b0;
        //Negative simple test, -1 + -1.
        #100; // -------------  Current Time:  800ns
        input_bus1 = 8'b10000000;
        input_bus2 = 8'b10000000;
        input1 = 1'b0;
        //Lowest negative overflow, -128 + -128.
        #100; // -------------  Current Time:  900ns
        input_bus1 = 8'b01111111;
        input_bus2 = 8'b01111111;
        input1 = 1'b0;
        //Highest positive overflow, 127 + 127.
        #100; // -------------  Current Time:  1000ns
        input_bus1 = 8'b10000001;
        input_bus2 = 8'b10000001;
        input1 = 1'b0;
        //Second lowest negative overflow, -127 + -127.
        #100; // -------------  Current Time:  1100ns
        input_bus1 = 8'b01111111;
        input_bus2 = 8'b10000000;
        input1 = 1'b0;
        //Highest possible positive overflow with subtraction, 127 - (-128).
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/18/2021 09:27:17 PM
// Design Name: 
// Module Name: Full_Adder
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module Full_Adder(
    input A,
    input B,
    input c_in,
    output s,
    output c_out
    );
    wire C = ~c_in;
    m4_1e sum (.in({c_in, C, C, c_in}), .sel({A,B}), .e(1), .o(s));
    m4_1e carry (.in({1'b1, c_in, c_in, 1'b0}), .sel({A,B}), .e(1), .o(c_out));
endmodule
```

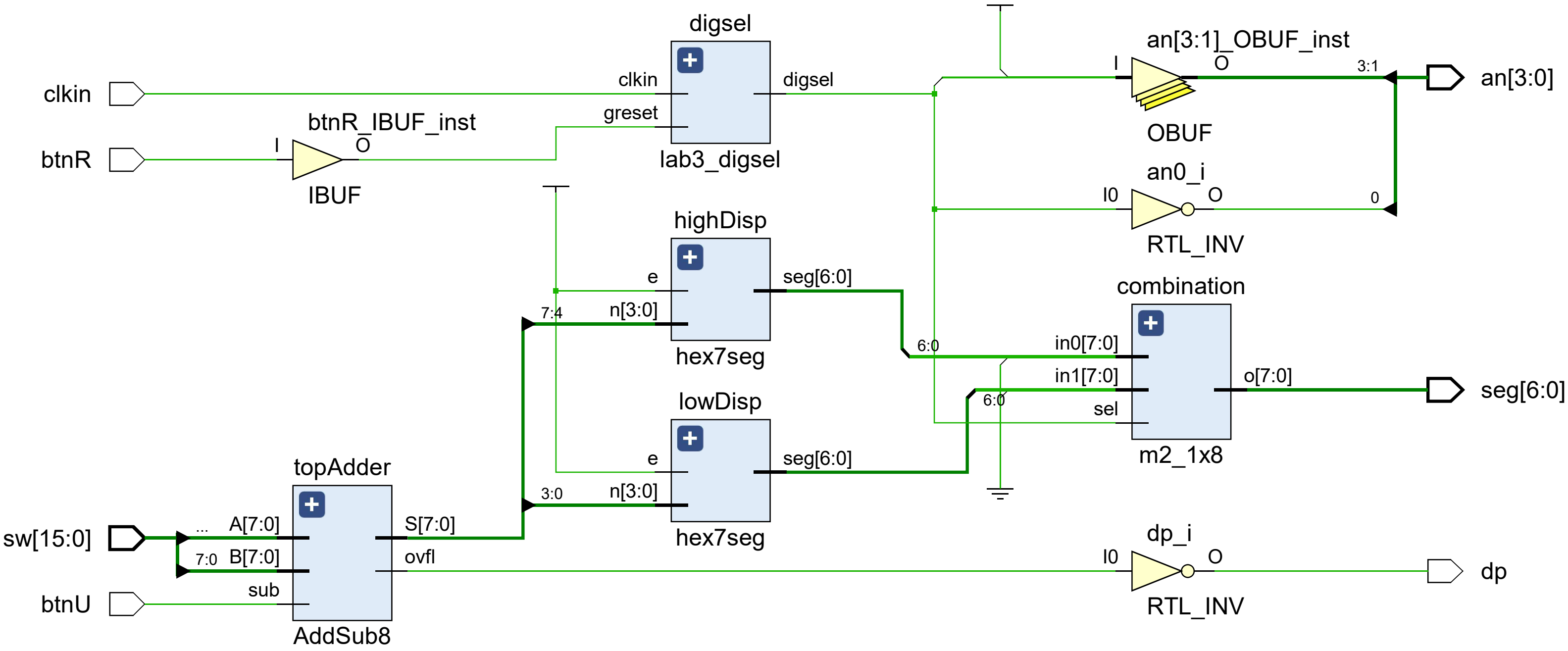

c_in
A
B
1
0
C_i
I0
O
RTL_INV
carry
e
in[3:0]
sel[1:0]
o
m4_1e
c_out
sum
e
in[3:0]
sel[1:0]
o
m4_1e
s

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/19/2021 04:03:04 PM
// Design Name: 
// Module Name: hex7seg
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input [3:0] n,
    input e,
    output [6:0] seg
    );
    wire D = ~n[0];
    wire d = n[0];
    //Might need to invert these. Who knows, maybe they'll work.
    //A
    m8_1e A (.in({1'b0, d, d, 1'b0, 1'b0, D, 1'b0, d}), .sel(n[3:1]), .e(e), 
.o(seg[0]));
    // F
    m8_1e F (.in({1'b0, d, 1'b0, 1'b0, d, 1'b0, 1'b1, d}), .sel(n[3:1]), .e(e), 
.o(seg[5]));
    //B
    m8_1e B (.in({1'b1, D, d, 1'b0, D, d, 1'b0, 1'b0}), .sel(n[3:1]), .e(e), 
.o(seg[1]));
    //G
    m8_1e G (.in({1'b0, D, 1'b0, 1'b0, d, 1'b0, 1'b0, 1'b1}), .sel(n[3:1]), .e(e), 
.o(seg[6]));
    //E
    m8_1e E (.in({1'b0, 1'b0, 1'b0, d, d, 1'b1, d, d}), .sel(n[3:1]), .e(e), 
.o(seg[4]));
    //C
    m8_1e C (.in({1'b1, D, 1'b0, 1'b0, 1'b0, 1'b0, D, 1'b0}), .sel(n[3:1]), .e(e),
.o(seg[2]));
    //D
    m8_1e Dx (.in({d, 1'b0, D, d, d, D, 1'b0, d}), .sel(n[3:1]), .e(e), .o(seg[3]));
endmodule
```



A
B
C
Dx
E
F
G
m8_1e
e
in[7:0]
sel[2:0]
o
3:1
...
0
1
2
3
4
5
6
e
n[3:0]
seg[6:0]
D_i
I0
O
RTL_INV

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/20/2021 01:36:19 PM
// Design Name: 
// Module Name: lab3_top
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module lab3_top(
    input [15:0] sw,
    input btnU,
    input btnR,
    input clkin,
    output [6:0] seg,
    output dp,
    output [3:0] an
    );
    wire [7:0] summation;
    wire [6:0] C_display;
    wire [6:0] B_display;
    //wire B_enable = 1'b1;
    //wire C_enable = 1'b1;
    wire dummy;
    wire overflow_dummy;
    wire dig_sel;
    //wire [7:0] segbus;
    
    lab3_digsel digsel (.clkin(clkin), .greset(btnR), .digsel(dig_sel));
    
    AddSub8 topAdder (.A(sw[15:8]), .B(sw[7:0]), .sub(btnU), .S(summation), 
.ovfl(overflow_dummy));
    hex7seg highDisp (.n(summation[7:4]), .e(1'b1), .seg(B_display));
    hex7seg lowDisp (.n(summation[3:0]), .e(1'b1), .seg(C_display));
    m2_1x8 combination (.in0({1'b0, B_display}), .in1({1'b0, C_display}),
.sel(dig_sel), .o({dummy, seg}));

    //assign seg = ~segbus[6:0];
    assign an[0] = ~dig_sel;
    assign an[1] = dig_sel;
    assign an[2] = 1;
    assign an[3] = 1;
    assign dp = ~overflow_dummy;
endmodule
```


clkin
btnR
btnR_IBUF_inst
I
O
IBUF
digsel
clkin
greset
digsel
lab3_digsel
an[3:1]_OBUF_inst
I
O
OBUF
3:1
an[3:0]
an0_i
I0
O
RTL_INV
0
highDisp
e
n[3:0]
seg[6:0]
hex7seg
7:4
combination
6:0
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
6:0
seg[6:0]
lowDisp
e
n[3:0]
seg[6:0]
hex7seg
3:0
topAdder
sw[15:0]
...
A[7:0]
7:0
B[7:0]
sub
S[7:0]
ovfl
AddSub8
btnU
dp_i
I0
O
RTL_INV
dp

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/20/2021 03:07:01 PM
// Design Name:
// Module Name: lab3_simulation
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lab3_simulation();
    reg [15:0] sw;
    wire [6:0] seg;
    wire [3:0] an;
    reg btnU, btnR, clkin;
    wire dp;
    lab3_top UUT (.sw(sw), .btnU(btnU), .btnR(btnR), .clkin(clkin), .seg(seg),
.dp(dp), .an(an));

    wire [7:0] D7Seg3, D7Seg2, D7Seg1, D7Seg0; // Change the Radix of these signals
to ASCII
    show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
     .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
        forever
        begin
```

```
         #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
     end
 end

initial
begin
  // add your stimuli here
  // to set signal foo to value 0 use
  // foo = 1'b0;
  // to set signal foo to value 1 use
  // foo = 1'b1;
  //always advance time my multiples of 100ns
  // to advance time by 100ns use the following line
  btnR = 1'b0;
  #1000;
  //Skip first 1000ns.
  sw[15:8] = 8'b00000000;
  sw[7:0] = 8'b00000000;
  btnU = 1'b0;
  //btnR = 1'b0;
  //0 + 0 = 0.
  #200;
  sw[15:8] = 8'b00000001;
  sw[7:0] = 8'b00000001;
  btnU = 1'b1;
  //1 - 1 = 0.
  #200;
  sw[15:8] = 8'b11111111;
  sw[7:0] = 8'b00000001;
  //-1 - 1 = -2.
  #200;
  sw[15:8] = 8'b00000010;
  sw[7:0] = 8'b00000001;
  //2 - 1 = 1.
  #200;
  sw[15:8] = 8'b10000000;
  sw[7:0] = 8'b00000001;
  btnU = 1'b0;
  //-128 + 1 = -127.
  #200;
  //Trying 1-F on all bits now.
  sw[15:8] = 8'b00000001;
  sw[7:0] = 8'b00010000;
  //11
  #200;
  sw[15:8] = 8'b00000010;
```

```
sw[7:0] = 8'b00100000;
//22
#200;
sw[15:8] = 8'b00000011;
sw[7:0] = 8'b00110000;
//33
#200;
sw[15:8] = 8'b00000100;
sw[7:0] = 8'b01000000;
//44
#200;
sw[15:8] = 8'b00000101;
sw[7:0] = 8'b01010000;
//55
#200;
sw[15:8] = 8'b00000110;
sw[7:0] = 8'b01100000;
//66
#200;
sw[15:8] = 8'b00000111;
sw[7:0] = 8'b01110000;
//77
#200;
sw[15:8] = 8'b00001000;
sw[7:0] = 8'b10000000;
//88
#200;
sw[15:8] = 8'b00001001;
sw[7:0] = 8'b10010000;
//99
#200;
sw[15:8] = 8'b00001010;
sw[7:0] = 8'b10100000;
//AA
#200;
sw[15:8] = 8'b00001011;
sw[7:0] = 8'b10110000;
//BB
#200;
sw[15:8] = 8'b00001100;
sw[7:0] = 8'b11000000;
//CC
#200;
sw[15:8] = 8'b00001101;
sw[7:0] = 8'b11010000;
//DD
```

```
     #200;
     sw[15:8] = 8'b00001110;
     sw[7:0] = 8'b11100000;
     //EE
     #200;
     sw[15:8] = 8'b00001111;
     sw[7:0] = 8'b11110000;
     //FF
     #200;
     //Also include 8 tests where the two numbers have the same and opposite signs and
the input sub alternates.
     sw[15:8] = 8'b00000100;
     sw[7:0] = 8'b00000101;
     //4 + 5 = 9.
     #200;
     sw[15:8] = 8'b00000100;
     sw[7:0] = 8'b00000101;
     btnU = 1'b1;
     //4 - 5 = -1.
     #200;
     sw[15:8] = 8'b11111100;
     sw[7:0] = 8'b00000101;
     btnU = 1'b0;
     // -4 + 5 = 1.
     #200;
     sw[15:8] = 8'b11111100;
     sw[7:0] = 8'b00000101;
     btnU = 1'b1;
     // -4 - 5 = -9.
     #200;
     sw[15:8] = 8'b11111100;
     sw[7:0] = 8'b11111011;
     btnU = 1'b0;
     // -4 + -5 = -9.
     #200;
     sw[15:8] = 8'b11111100;
     sw[7:0] = 8'b11111011;
     btnU = 1'b1;
     // -4 - -5 = 1.
     #200;
     sw[15:8] = 8'b00000100;
     sw[7:0] = 8'b11111011;
     btnU = 1'b0;
     //4 + -5 = -1.
     #200;
     sw[15:8] = 8'b00000100;
     sw[7:0] = 8'b11111011;
     btnU = 1'b1;
     //4 - -5 = 9.
     #200;
     //Make sure to have some values that cause overflows.
     sw[15:8] = 8'b01111111;
     sw[7:0] = 8'b00001000;
     btnU = 1'b0;
     //127 + 8 = overflow.
     #200;
     sw[15:8] = 8'b01000000;
     sw[7:0] = 8'b11000000;
     btnU = 1'b1;
     //64 - (-64) = overflow.
     #200;
     sw[15:8] = 8'b10000000;
     sw[7:0] = 8'b11111000;
     btnU = 1'b0;
     //-128 + -8 = overflow.
     #200;
     sw[15:8] = 8'b10000000;
     sw[7:0] = 8'b00001000;
     btnU = 1'b1;
     //-128 - 8 = overflow.

    end

endmodule
```