# ELC 2137 Lab 8: 4-digit Display

Aaron Mendoza

October 17, 2020

## Summary

The purpose of this lab is to build a 4-digit display that can switch between hexadecimal and BCD output using modules built in previous labs.

The modules that are used from previous labs are the eleven-bit BCD converter, the add3 module, and the seven-segment decoder. These will be used within the modules that were built in this lab.

The first module I built in this lab was a 2-input MUX module. This works exactly like the MUX module built in previous labs: it has two inputs, a select input, and an output. However, when coding for this MUX, I used a parameter. Parameters are basically like constants in c++. Prior to listing the inputs in my module, I declared a paramter called BITS that corresponds with the number of bits that are in the input and output of the MUX. For example, the first MUX of this lab required 16 bits, so I set BITS equal to 16 and in my inputs I used [BITS-1:0] when declaring the size of the inputs and output. This use of paramters allows for flexible, reusable modules.

The next module I built was a 4-input MUX module. The only difference with this MUX is that it has four inputs and the select is two bits rather than one. I used another parameter this time as well but set BITS equal to 4.

Following the mux4 module, I built a anode decoder module that decides which digit on the display will show. I used a case statement when coding this since the inputs were only two bits, so there were only four cases. The 0 case corresponds with the ones digit, and 1 case corresponds with the tens digit, and so on to the thousands digit.

The next module built was the four digit driver. This combines all of the previous modules into one single module. It had four inputs: 16 bits for data, 2 bits for digit select, one bit for hex or BCD, and one bit for the negative sign. The four outputs were the seven segment display, the decimal point, and the anode. This module was completely covered by a wrapper called sseg4 manual whose inputs and outputs corresponded with the constraints file of the Basys3 board. To check if my code was working properly, I added two simulation files that allowed me to check if my output matched the given output of the lab during the simulation of my top level module.

## Results

Figure 1: ERT for mux2 Module

| in0 | in1 | sel | out |
|---|---|---|---|
| 000000000000001 | 111111111111110 | 0 | 000000000000001 |
| 000000000000001 | 111111111111110 | 1 | 111111111111110 |
| 101010101010101 | 000000000111111 | 0 | 101010101010101 |
| 101010101010101 | 000000000111111 | 1 | 000000000111111 |


Figure 2: mux2 Simulation

Figure 3: ERT for mux4 Module

| in0 | in1 | in2 | in3 | sel | out |
|---|---|---|---|---|---|
| 0000 | 0001 | 0010 | 0011 | 00 | 0000 |
| 0000 | 0001 | 0010 | 0011 | 01 | 0001 |
| 0000 | 0001 | 0010 | 0011 | 10 | 0010 |
| 0000 | 0001 | 0010 | 0011 | 11 | 0011 |
| 0110 | 1111 | 1010 | 0101 | 00 | 0110 |
| 0110 | 1111 | 1010 | 0101 | 01 | 1111 |
| 0110 | 1111 | 1010 | 0101 | 10 | 0110 |
| 0110 | 1111 | 1010 | 0101 | 11 | 0101 |

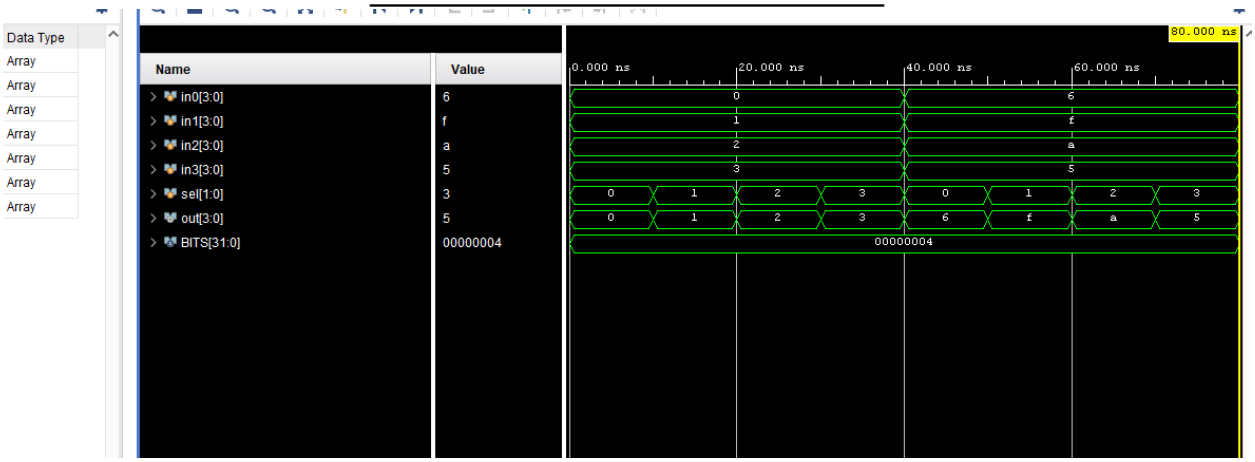


Figure 4: mux4 Simulation

Figure 5: ERT for Anode Decoder Module

| in | out |
|---|---|
| 00 | 1110 |
| 01 | 1101 |
| 10 | 1011 |
| 11 | 0111 |

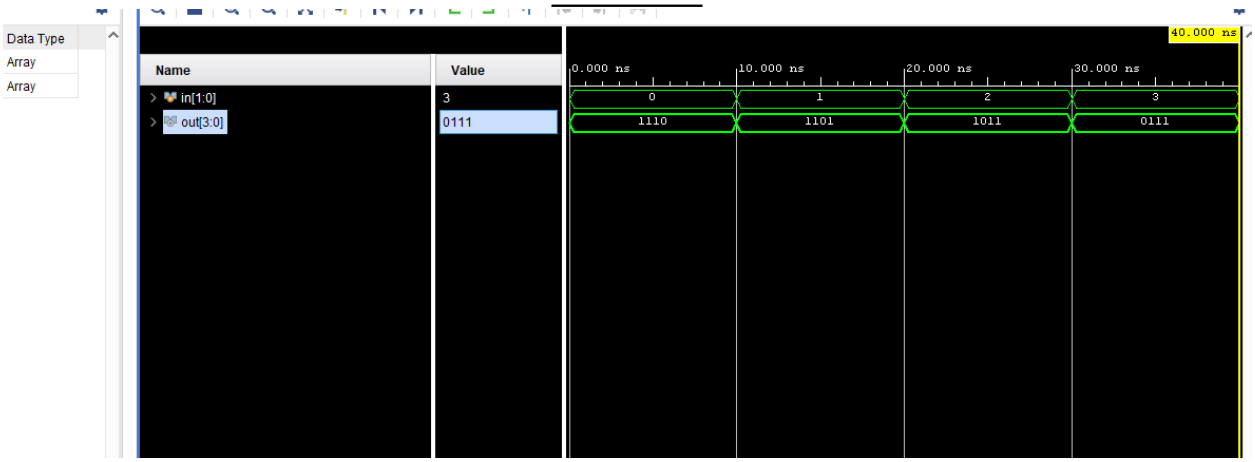


Figure 6: Anode Decoder Simulation

```
Time resolution is 1 ps
source basys3.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'F
#   }
# }
# run 1000ns
    +--------- Digit 3 (? = incorrect segment values)
    |+-------- Decimal point
    ||+------- Digit 2 (? = incorrect segment values)
    |||+------ Decimal point
    ||||+----- Digit 1 (? = incorrect segment values)
    |||||+---- Decimal point
    ||||||+--- Digit 0 (? = incorrect segment values)
    |||||||+-- Decimal point
    ||||||||
-->       0
-->       C
-->       2
-->     1
-->   0
-->   1
-->     4
-->       0
--> 0
--> 1
--> 0
--> -
-->   4
-->     8
-->       C
$finish called at time : 150 ns : File "C:/Users/aaron_mendoza1/Documents/GitHub/Lab08/Lab08_project/codedirectory/basys3.sv" Line 98
INFO: [USF-XSim-96] XSim completed. Design snapshot 'basys3_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 7: Top-Level Simulation

## Code

Listing 1: Add3 Module Code

```
module mux2
    #(parameter BITS=16)
    (input [BITS-1:0] in0,
    input [BITS-1:0] in1,
    input sel,
    output [BITS-1:0] out
    );

    assign out = sel ? in1 : in0;

endmodule
```

—

Listing 2: Add3 Module Code

```
module mux4
    #(parameter BITS = 4)
    (input [BITS-1:0] in0,
    input [BITS-1:0] in1,
    input [BITS-1:0] in2,
    input [BITS-1:0] in3,
    input [1:0] sel,
    output reg [BITS-1:0] out
    );

    always @*
        case({sel[1],sel[0]})
            2'b00: out  = in0;
            2'b01: out  = in1;
            2'b10: out  = in2;
            2'b11: out  = in3;
        endcase
endmodule
```

—

Listing 3: Add3 Module Code

```
module an_decoder(
    input [1:0] in,
    output reg [3:0] out
    );
    always @*
    case({in[1],in[0]})
    2'b00: out=4'b1110;
    2'b01: out=4'b1101;
    2'b10: out=4'b1011;
    2'b11: out=4'b0111;
```

```
    endcase
endmodule
```

—

Listing 4: Add3 Module Code

```
module sseg4(
    input [15:0] data,
    input hex_dec,
    input sign,
    input [1:0] digit_sel,
    output [6:0] seg,
    output dp,
    output [3:0] an
    );
    wire[15:0] bcd11_mux2;
    wire[15:0] mux2_mux4;
    wire[3:0] mux4_decoder;
    wire[6:0] decoder_mux2;
    wire sel_mux2;


    eleven_bit_BCD bcd11(
    .B(data[10:0]),
    .out(bcd11_mux2));

    mux2 #(.BITS(16)) mux2_1(
    .in0(bcd11_mux2),
    .in1(data),
    .sel(hex_dec),
    .out(mux2_mux4));

    mux4 #(.BITS(4)) mymux4(
    .in0(mux2_mux4[3:0]),
    .in1(mux2_mux4[7:4]),
    .in2(mux2_mux4[11:8]),
    .in3(mux2_mux4[15:12]),
    .sel(digit_sel),
    .out(mux4_decoder));

    sseg_decoder my_sseg_decoder(
    .num(mux4_decoder),
    .sseg(decoder_mux2));

    an_decoder my_an_decoder(
    .in(digit_sel),
    .out(an));

    assign sel_mux2 = sign&~an[3];

    mux2 #(.BITS(7)) mux2_2(
    .in0(decoder_mux2),
    .in1(7'b0111111),
```

```
    .sel(sel_mux2),
    .out(seg));

    assign dp = 1'b1;

endmodule
```

—

Listing 5: Add3 Module Code

```
module sseg4_manual(
    input [15:0] sw,
    input clk,
    output [6:0] seg,
    output dp,
    output [3:0] an
    );

    sseg4 mysseg4(
    .data({4'b0000,sw[11:0]}),
    .hex_dec(sw[15]),
    .sign(sw[14]),
    .digit_sel(sw[13:12]),
    .seg(seg),
    .dp(dp),
    .an(an));

endmodule
```

—