ELC 2137 Lab 6: MUX and 7-segment Decoder

Aaron Mendoza

October 7, 2020

Summary

The purpose of this lab was to set up a circuit to display an 8-bit number on two 7-segment displays which are two hexadecimal digits. In order to do this, I designed four combinational logic modules in Verilog and implemented my design on a Basys3 board.

The first combinational logic module built was a multiplexer, or MUX. My multiplexer has two inputs and one output that depends on another input called select. In this case, my two inputs are 4 bit binary numbers while select is a 1 bit number; if select is on, my MUX outputs the second input (in1) and if select is off, my MUX outputs the first input (in0). So, the output is also a four bit number.

The second combinational logic module built was a seven-segment decoder. The seven-segment decoder takes in a four bit hexadecimal number and outputs a seven bit binary number that corresponds to the display. The most significant bit of the output relates to "G" on the seven-segment display, and in this case, the output is active low, which means that 1 actually means off while 0 means on. So, segments "A-F" of the seven-segment display would be on to make the shape of a 0.

The third combinational logic module built was the actual seven-segment itself. It combines the MUX and decoder using a four bit wire; the wire takes the output of the MUX and inputs it into the decoder to give the 7 bit output. The top module is a wrapper around this module. The wrapper has a 16 bit switch input with a 1 bit output for the decimal point, a 4 bit output for the anode of the display, and a 7 bit output for the 7 segments of the display. These correspond directly to a constraints file for the Basys3 board, allowing me to generate a bitsream and program the board to behave with the code I wrote in Verilog.

Q&A

1. List of errors found during simulation. What does this tell you about why we run simulations?

   The only error I had when I ran the simulation of my top-level module was my an outputs were not connected properly which resulted in them having a value of Z in the simulation. To fix this, I checked each instance of those outputs and realized that I mispelled the labels that I had for them within the sseg1.sv code. This shows the importance of running simulations because it reveals minor errors and where there are disconnections within my build.

2. How many wires are connected to the 7-segment display? If the segments were not all connected together, how many wires would there have to be? Why do we prefer the current method vs. separating all of the segments?

There is 1 wire connected to the 7-segment display. If the segments were not all connected together, there would have to be 7 wires. We prefer the current method because it is much easier to follow and less prone to having a mistake like missing a wire or repeating the same one.

Results

Table 1: ERT for 4-bit MUX

sel	in0	in1	out
0	0001	0000	0001
1	0001	0000	0000
0	1010	0101	1010
1	0001	0000	0101

Table 2: ERT for Seven-segment Decoder

	Time(ns)	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
Input	num(hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	sseg(hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 3: ERT for Top-level Module

A	B	sel	Output
1010	0101	0	0001000
1010	0101	1	0010010
1001	0111	0	0011000
1001	0111	1	1111000

Code

Listing 1: MUX Verilog code

```
module mux2_4b(
    input [3:0] in0,
    input [3:0] in1,
    input sel,
    output [3:0] out
    );
    assign out = sel ? in1 : in0;
endmodule
```

—

Listing 2: MUX Test Verilog code

```
module mux2_4b_test();
    reg [3:0] in0;
```

```
    reg [3:0] in1;
    reg sel;
    wire [3:0] out;

mux2_4b dut(
    .in0(in0),
    .in1(in1),
    .sel(sel),
    .out(out)
    );

initial begin
in0=4'b0001;in1=4'b0000;sel=0;#10;
in0=4'b0001;in1=4'b0000;sel=1;#10;
in0=4'b1010;in1=4'b0101;sel=0;#10;
in0=4'b1010;in1=4'b0101;sel=1;#10;
$finish;
end

endmodule
```

—

Listing 3: Seven-segment Decoder Verilog code

```
module sseg_decoder(
    input [3:0] num,
    output reg [6:0] sseg
    );

// 4-bit to 7-segment decode logic
// (note: output is active low)
always @*
    case(num)
    4'h0: sseg = 7'b1000000;
    4'h1: sseg = 7'b1111001;
    4'h2: sseg = 7'b0100100;
    4'h3: sseg = 7'b0110000;
    4'h4: sseg = 7'b0011001;
    4'h5: sseg = 7'b0010010;
    4'h6: sseg = 7'b0000010;
    4'h7: sseg = 7'b1111000;
    4'h8: sseg = 7'b0000000;
    4'h9: sseg = 7'b0011000;
    4'hA: sseg = 7'b0001000;
    4'hB: sseg = 7'b0000011;
    4'hC: sseg = 7'b1000110;
    4'hD: sseg = 7'b0100001;
    4'hE: sseg = 7'b0000110;
    4'hF: sseg = 7'b0001110;
    endcase

endmodule
```

Listing 4: Seven-segment Decoder Test Verilog code

```
module sseg_decoder_test();
    reg [3:0] num;
    wire [6:0] sseg;

integer i;

sseg_decoder dut(
    .num(num),
    .sseg(sseg)
    );

initial begin
    for (i=0; i<=8'hF; i=i+1) begin
        num = i;
        #10;
    end
    $finish;
end
endmodule
```

Listing 5: Seven-segment Wrapper Verilog code

```
module sseg1_wrapper(
    input [15:0] sw,
    input clk,
    output [3:0] an,
    output [6:0] seg,
    output dp
    );

    sseg1 my_sseg1(
    .A(sw[7:4]),
    .B(sw[3:0]),
    .sel(sw[15]),
    .seg_un(an[3:2]),
    .dp(dp),
    .sseg(seg),
    .seg_L(an[1]),
    .seg_R(an[0])
    );

endmodule
```

Listing 6: Seven-segment 1 Verilog code

```
module sseg1(
    input [15:0] sw,
    output [3:0] an,
    output [6:0] seg,
    output dp
    );
endmodule
```

—

Listing 7: Seven-segment 1 Test Verilog code

```
module sseg1_test();

    reg [3:0] A;
    reg [3:0] B;
    reg sel;
    wire [1:0] seg_un;
    wire dp;
    wire[6:0] sseg;
    wire seg_L;
    wire seg_R;

    sseg1 dut(
        .A(A),
        .B(B),
        .sel(sel),
        .seg_un(seg_un),
        .dp(dp),
        .sseg(sseg),
        .seg_L(seg_L),
        .seg_R(seg_R)
        );

    initial begin
    A[3:0]=4'b0000; B[3:0]=4'b0000; sel=1'b0; #10;
    A[3:0]=4'b1010; B[3:0]=4'b0101; #10;
    sel=1'b0; #10;
    sel=1'b1; #10;
    A[3:0]=4'b0000; B[3:0]=4'b0000; sel=1'b0; #10;
    A[3:0]=4'b1001; B[3:0]=4'b0111; #10;
    sel=1'b0; #10;
    sel=1'b1; #10;
    $finish;
    end
endmodule
```

—