ELC 2137 Lab #6: MUX and 7-segment Decoder

Sebastian Lopez and Megan Gordon

February 27, 2020

Summary

In this lab we wrote a multiplexer in Verilog using the conditional operator, as well as combinational Verilog components using an always block. We defined and used multi-bit signals (vectors), then instantiated and connected components in a toplevel module. We then used the provided constraint files to specify package pins. Finally, we implemented the design on out Digilent Basys3 Board's.

Q&A

1. **How many wires are connected to the 7-segment display?**

   There are 7 wires connected to the 7-segment display.

2. **If the segments were not all connected together, how many wires would there have to be?**

   If the segments were not all connected together, there would need to be 28 wires.

3. **Why do we prefer the current method vs. seperating all of the segments?**

   We prefer the current method over seperating all the segments, because less wires implies that there is a lesser chance in errors.

Results

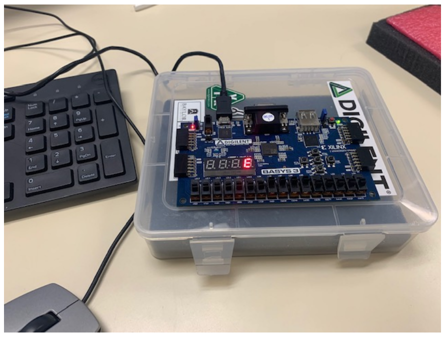

Figure 1: Here the board is showing a value on the first digit. (Value E)

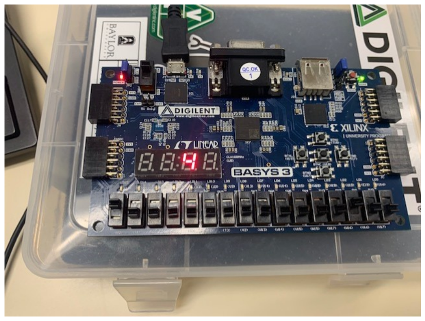

Figure 2: Here the board is showing a value on the second digit. (Value 4)

Errors

- A very prominent error was that we could not seem to find the correct path for our 'out' variable in the sseg1.sv file.
- We then figured out that our statements in the sseg1test.sv were incorrect. Instead of repeating the same statements we did in the source code file, we decided it would be right to make an exclusive statement for all values of sw, and an.
- Both of these errors show that running simulations both constantly and consistently, strongly secure that your code is functioning properly throughout the process of writing it. If one were to write down all their code in one shot and process it at the very end there could be several errors (maybe even minor ones) that can really affect your outcome. It will be much easier to find the problem within your code if you run your simulations properly, also considering that you will be looking a smaller amount of code rather than the entirety of it.

Code

Listing 1: mux2 4b Source Code

```
`timescale 1ns / 1ps

module mux2_4b(
input [3:0] in0, in1,
input sel,
output [3:0] out
);

assign out = sel? in0:in1;

endmodule //mux2_4b
```

Listing 2: mux2 4b Test Code

```
`timescale 1ns / 1ps
// ELC 2137 - Lab6 - 02/20/2020
// Sebastian Lopez and Megan Gordon

module mux2_4b_test( );
```

```
reg [3:0] in0, in1;
reg sel;
wire [3:0] out;

mux2_4b Test1(
.in0(in0), .in1(in1), .sel(sel),
.out(out)
);

initial
begin

sel = 1;
in0 = 0;
in1 = 1;
#10;
sel = 1;
in0 = 1;
in1 = 0;
#10;
sel = 0;
in0 = 0;
in1 = 1;
#10;
sel = 0;
in0 = 1;
in1 = 0;
#10;
$finish;

end

endmodule//mux2_4b_test
```

Listing 3: Sseg Decoder Source Code

```
`timescale 1ns / 1ps
// ELC 2137 - Lab6 - 02/20/2020
// Sebastian Lopez and Megan Gordon

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

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'b0010000;
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 //sseg_decoder
```

Listing 4: Sseg Decoder Test Bench Code

```
`timescale 1ns / 1ps
// ELC 2137 - Lab6 - 02/20/2020
// Sebastian Lopez and Megan GordonS

module sseg_decoder_test();

reg [3:0] num;
wire [6:0] sseg;

integer i;

sseg_decoder d1(
.num(num),
.sseg(sseg)
);

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

Listing 5: Sseg1 Source Code

```
`timescale 1ns / 1ps
// ELC 2137 - Lab6 - 02/20/2020
// Sebastian Lopez and Megan Gordon

module sseg1(
input [15:0] sw, //switches
output [3:0] an, // 7-segment digits
output [6:0] seg, // 7-seg segments
output dp // decimal point
);

wire [3:0] out;
```

```
not not1(an[1],sw[15]);
assign an[0] = sw[15];
assign an[3:2] = 3;
assign dp = 1;

mux2_4b mux1(
.in0(sw[3:0]), .in1(sw[7:4]), .sel(sw[15]),
.out(out)
);

sseg_decoder sseg2(
.num(out),
.sseg(seg)
);

endmodule //seg1
```

Listing 6: Sseg1 Test Bench Code

```
`timescale 1ns / 1ps
// ELC 2137 - Lab6 - 02/20/2020
// Sebastian Lopez and Megan Gordon

module sseg1_test(
input [15:0] sw, //switches
output [3:0] an, // 7-segment digits
output [6:0] seg, // 7-seg segments
output dp // decimal point
);

wire [3:0] out;

not not1(an[1],sw[15]);
assign an[0] = sw[15];
assign an[3:2] = 3;
assign dp = 1;

mux2_4b mux1(
.in0(sw[3:0]), .in1(sw[7:4]), .sel(sw[15]),
.out(out)
);

sseg_decoder sseg2(
.num(out),
.sseg(seg)
);

endmodule //seg1_test
```