ELC 2137 Lab 06: MUX and 7-segment Decoder

Yiting Wang

October 15, 2020

Summary

In this lab, we will be setting up a circuit to display an 8-bit number on two 7-segment displays (i.e.two hexadecimal digits). And we will design a couple of combinational logic modules in Verilog and then implement your design on an FPGA. After completing this lab, we should be able to write a multiplexer in Verilog using the conditional operator, Write combinational Verilog components using an always block, Define and use multi-bit signals (vectors), Instantiate and connect components in a top-level module, Use provided constraint files to specify package pins, and Implement design on FPGA (Digilent Basys3 board).

Q&A

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

   After I ran the Simulation, I found the results is not match with my expected results table, so I found some mistakes about my code. Most of those mistakes are about assign the outputs, some of them are about copy mistake.

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?

   How many wires are connected to the 7-segment display? 12. sseg (7 wires), dp (1 wire), and an (4 wires) for a total of 12.

   If the segments were not all connected together, how many wires would there have to be? 32.

   Why do we prefer the current method vs. separating all of the segments? because Less wires, less power, and yes in even bigger projects it helps us organize it more. In another words, when we do some bigger project in the future, we will have more inputs and outputs, if we use the current method, it will help us to organize them easier.

Results

Firgure 1 is the simulation waveform and ERT of the 4-bit Multiplexer.

Time (ns):	0	10	20	30
in1	1111	1111	0100	0100
in0	0000	0000	1101	1101
sel	0	1	0	1
out	0000	1111	1101	0100

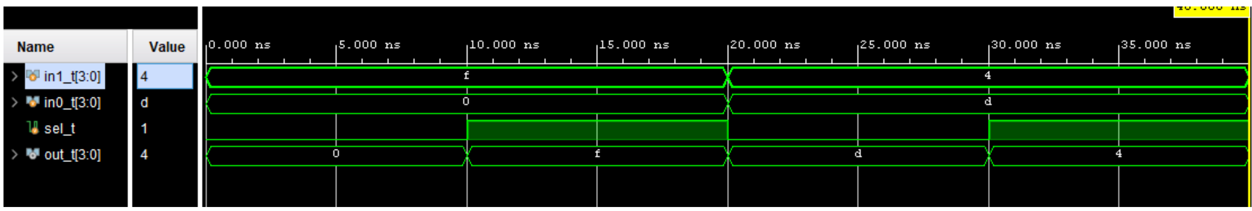


Figure 1: the simulation waveform and ERT of the 4-bit Multiplexer

Firgure 2 is the simulation waveform and ERT of the Seven-segment Decoder.

Time (ns):	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
num (hex)	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
sseg (hex)	40	79	24	30	19	12	02	78	00	18	08	03	46	21	06	0e

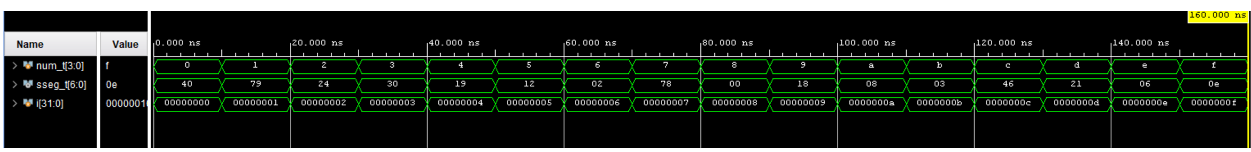


Figure 2: the simulation waveform and ERT of the Seven-segment Decoder

Firgure 3 is the simulation waveform and ERT of the Top-level Module.

This is the picture of my board which showing a value on the first digit

Time (ns):	0	10	20	30	150	160	170	180	290	300	310	320
sw	0000	0040	8040	0079	0078	8078	0000	8000	0006	8006	000e	800e
an	11	11	11	11	11	11	11	11	11	11	11	11
sseg	40	40	19	18	00	78	40	40	02	40	06	40
dp	1	1	1	1	1	1	1	1	1	1	1	1
an1	1	1	0	1	1	0	1	0	1	0	1	0
an0	0	0	1	0	0	1	0	1	0	1	0	1

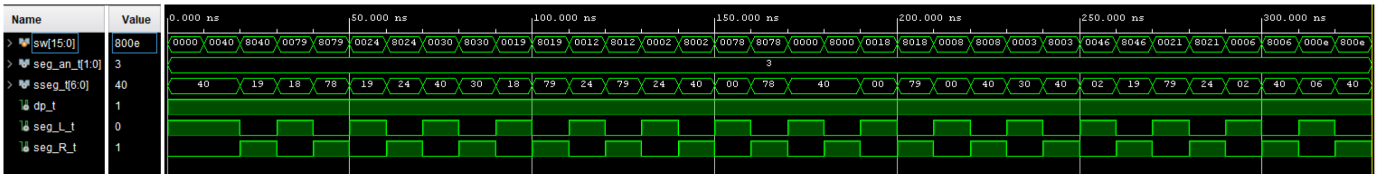
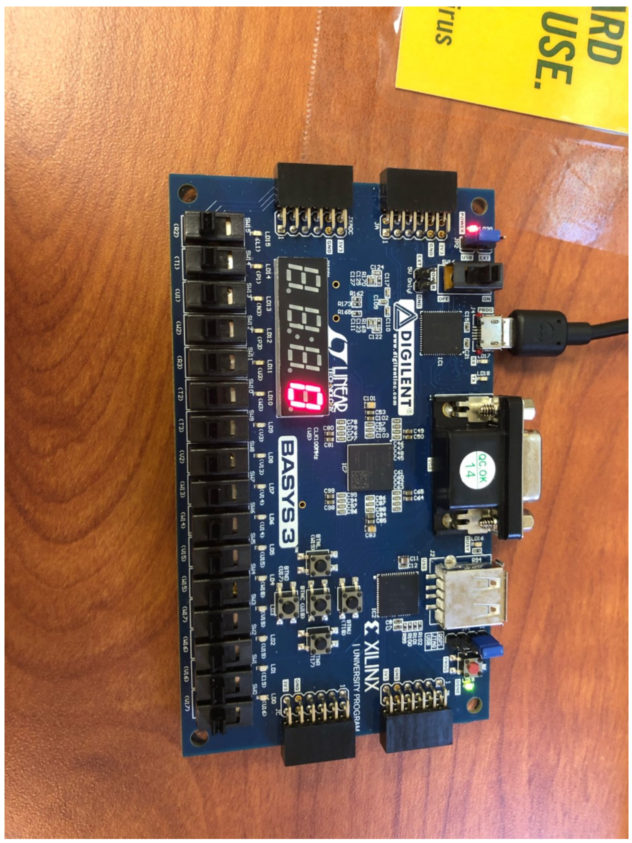
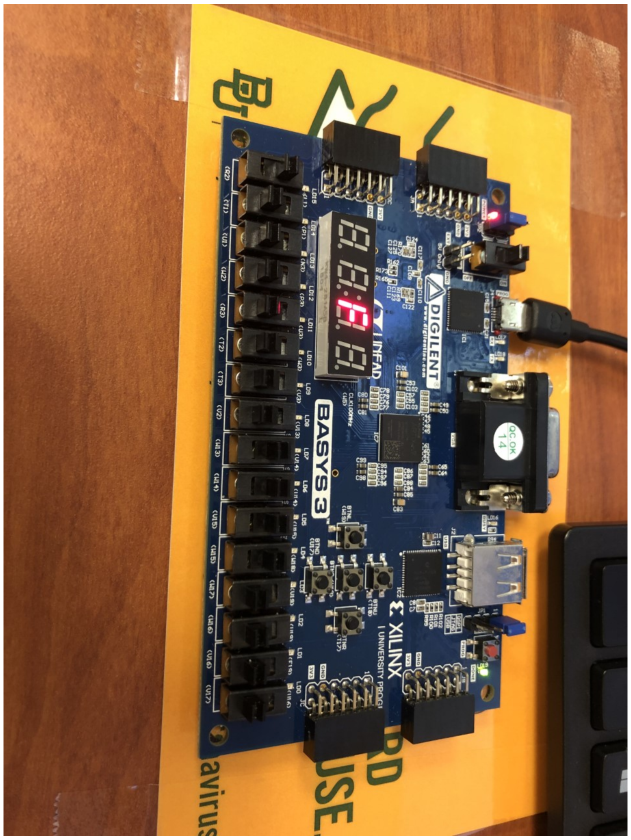


Figure 3: the simulation waveform and ERT of the Top-level Module



This is the picture of my board which showing a value on the second digit



Code

my Verilog source file for the 4-bit Multiplexer.

File Inclusion

Listing 1: 4-bit Multiplexer Verilog code

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   //////////////////////////////////////////////////////////////////////////////////



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

```
    assign out = sel ? in1 : in0; //sel is the  select  line that chooses
       between in0 (when sel=0) and in1(when sel=1).

endmodule //mux2_4b
```

my Verilog source file for the 4-bit Multiplexer test.

File Inclusion

Listing 2: 4-bit Multiplexer Test Benches Verilog code

```
`timescale 1ns / 1ps
//
   ///////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   ///////////////////////////////////////////////////////////////////////////////



module mux2_4b_test();

    reg [3:0] in1_t, in0_t;
    reg sel_t;
    wire [3:0] out_t;

    mux2_4b dut(
        .in1(in1_t), .in0(in0_t), .sel(sel_t),
        .out(out_t)
    );

    initial begin
        in1_t = 4'b1111; in0_t = 4'b0000; sel_t = 0; #10;
        sel_t = 1; #10;
        in1_t = 4'b0100; in0_t = 4'b1101; sel_t = 0; #10;
        sel_t = 1; #10;
        $finish;
    end

endmodule//mux2_4b_test
```

my Verilog source file for the Seven-segment Decoder.

File Inclusion

Listing 3: Seven-segment Decoder Verilog code

```
`timescale 1ns / 1ps
```

```
//
   ////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   ////////////////////////////////////////////////////////////////////////////////



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'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;
        default: sseg = 7'b1111111;
    endcase

endmodule //sseg_decoder
```

my Verilog source file for the Seven-segment Decoder test.

File Inclusion

Listing 4: Seven-segment Decoder Test Benches Verilog code

```
`timescale 1ns / 1ps
//
   ////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
```

```
//
   ////////////////////////////////////////////////////////////////////////////////


module sseg_decoder_test();

    reg [3:0] num_t;
    wire [6:0] sseg_t;

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

    integer i;

    initial begin//finsh is the second, so need begin
        for (i=0; i<=8'hF; i=i+1) begin
            num_t = i;
            #10;
        end
        $finish;
    end

endmodule//sseg_decoder_test
```

my Verilog source file for the Top-level Module.

File Inclusion

Listing 5: Top-level Module Verilog code

```
`timescale 1ns / 1ps
//
   ////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   ////////////////////////////////////////////////////////////////////////////////



module sseg1(
    input [3:0] A,
    input [3:0] B,
    input sel,
    output [1:0] seg_an,
    output dp,
    output [6:0] sseg,
    output seg_L,
```

```
    output seg_R
    );

    wire [3:0] out_num;

    mux2_4b dut0(
        .in1(A), .in0(B), .sel(sel),
        .out(out_num)
    );

    sseg_decoder dut1(
        .num(out_num),
        .sseg(sseg)
    );

    assign [1:0] seg_an = 1;
    assign dp = 1,
    assign seg_L = ~sel;
    assign seg_R = sel;

endmodule
```

my Verilog source file for the Top-level Module test.

File Inclusion

Listing 6: Top-level Module Test Benches Verilog code

```
`timescale 1ns / 1ps
//
   ////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   ////////////////////////////////////////////////////////////////////////////////



module sseg1_test();

    reg [15:0] sw;
    wire [1:0] seg_an_t;
    wire [6:0] sseg_t;
    wire dp_t, seg_L_t, seg_R_t;

    sseg1 dut(
    .A(sw[7:4]), .B(sw[3:0]), .sel(sw[15]),
    .seg_an(seg_an_t), .dp(dp_t), .sseg(sseg_t), .seg_L(seg_L_t), .seg_R(
       seg_R_t)
    );
```

```
initial begin
    sw = 16'h0000; #10;
    // test case 1 for 0
    sw[7:0] = 7'b1000000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 2 for 1
    sw[7:0] = 7'b1111001;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 3 for 2
    sw[7:0] = 7'b0100100;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 4 for 3
    sw[7:0] = 7'b0110000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 5 for 4
    sw[7:0] = 7'b0011001;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 6 for 5
    sw[7:0] = 7'b0010010;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 7 for 6
    sw[7:0] = 7'b0000010;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 8 for 7
    sw[7:0] = 7'b1111000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 9 for 8
    sw[7:0] = 7'b0000000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 10 for 9
    sw[7:0] = 7'b0011000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 11 for a
    sw[7:0] = 7'b0001000;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 12 for b
    sw[7:0] = 7'b0000011;
    sw[15] = 1'b0; #10;
    sw[15] = 1'b1; #10;
    // test case 13 for c
    sw[7:0] = 7'b1000110;
    sw[15] = 1'b0; #10;
```

```
        sw[15] = 1'b1; #10;
        // test case 14 for d
        sw[7:0] = 7'b0100001;
        sw[15] = 1'b0; #10;
        sw[15] = 1'b1; #10;
        // test case 15 for e
        sw[7:0] = 7'b0000110;
        sw[15] = 1'b0; #10;
        sw[15] = 1'b1; #10;
        // test case 16 for f
        sw[7:0] = 7'b0001110;
        sw[15] = 1'b0; #10;
        sw[15] = 1'b1; #10;
        $finish;
    end

endmodule
```

File Inclusion

Listing 7: Two Bit Adder/Aubtractor Verilog code

```
`timescale 1ns / 1ps
//
   //////////////////////////////////////////////////////////////////////////////////

// Company: ELC 2137
// Engineer: Yiting Wang
// Create Date: 10/01/2020
//
   //////////////////////////////////////////////////////////////////////////////////



module sseg1_wrapper(
    input [15:0] sw,
    input clk,//do nothing with it but borad needs it to run
    output [3:0] an,
    output dp,
    output [6:0] seg
    );

    reg [15:0] sw;
    wire  [3:0] an;
    wire dp;
    wire [6:0] seg;

    sseg1 my_sseg1(
        .A(sw[7:4]),
        .B(sw[3:0]),
        .sel(sw[15]),
        .seg_an(an[3:2]),
        .dp(dp),
        .sseg(seg),
```

```
        .seg_L(an[1]),
        .seg_R(an[0])
    );

endmodule
```