ELC 2137 Lab 08: 4-digit Display

My Nguyen

October 22, 2020

Summary

This lab's purpose is to use parameter to create flexible, reusable module and use import modules to create modular design. First part is to create a mux2 module with uses parameter to generate mux2 module with different number of bit for input and output. Then, create mux4, and sseg4 module accordingly to the given schematic. Afterward, create a sseg4_manual as a top-level module to do on-board testing. Finally, import basys3.sv and disp.sv file to do top-level simulation.

Table and Figure

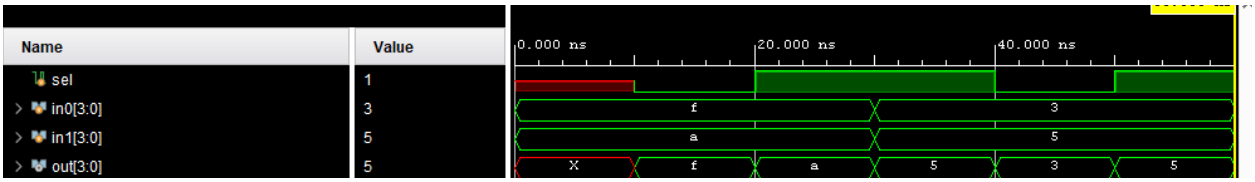

Figure 1: Mux2 ERT

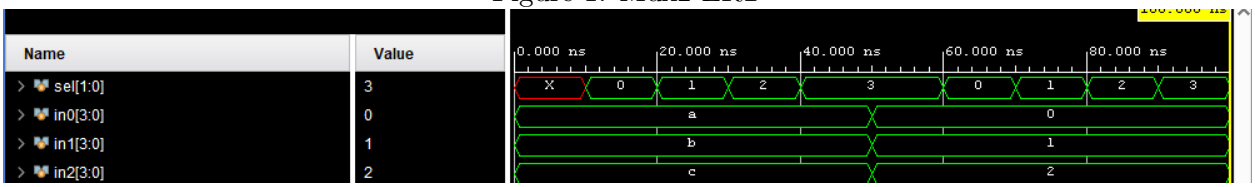

Figure 2: Mux4 ERT

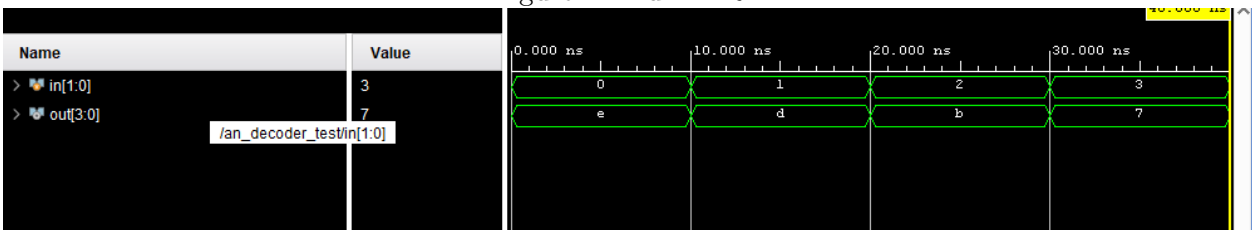

Figure 3: Anode Decoder ERT

Code

Listing 1: Mux2 Implementation

```
module mux2
    #(parameter BITS=4)
    (
```

```
    input sel,
    input [BITS-1:0] in0, in1,
    output [BITS-1:0] out
    );

    assign out = sel ? in1 : in0;
endmodule
```

Listing 2: Mux4 Implementation

```
module mux4(
    input [1:0] sel,
    input [3:0] in0, in1, in2, in3,
    output reg [3:0] out
    );

    always @*
        begin
            case (sel)
                2'b00: out = in0;
                2'b01: out = in1;
                2'b10: out = in2;
                2'b11: out = in3;
            endcase
        end
endmodule
```

Listing 3: Anode Decoder Implementation

```
module an_decoder (
    input [1:0] in,
    output reg [3:0] out
    );

    always @*
        begin
            case (in)
                2'b00: out = 4'b1110;
                2'b01: out = 4'b1101;
                2'b10: out = 4'b1011;
                2'b11: out = 4'b0111;
            endcase
        end
endmodule
```

Listing 4: Top Level Implementation

```
module sseg4(
    input [15:0] data,
    input [1:0] digit_sel,
    input hex_dec, sign,
    output [6:0] seg,
    output [3:0] an,
    output dp
```

```
    );

    assign dp = 1'b1;

    an_decoder an_decoder_0(
        .in(digit_sel),
        .out(an)
    );

    wire sel_mux2_7;
    assign sel_mux2_7 = sign & ~an[3];

    wire [15:0] bcd_mux2;
    bcd11 bcd11_0(
        .in(data[10:0]),
        .ones(bcd_mux2[3:0]),
        .tens(bcd_mux2[7:4]),
        .hundreds(bcd_mux2[11:8]),
        .thousands(bcd_mux2[15:12])
    );

    wire [15:0] mux2_mux4;
    mux2 #(.BITS(16)) mux2_0(
        .sel(hex_dec),
        .in0(bcd_mux2),
        .in1(data[15:0]),
        .out(mux2_mux4)
    );

    wire [4:0] mux4_decoder;
    mux4 mux4_0(
        .sel(digit_sel),
        .in0(mux2_mux4[3:0]),
        .in1(mux2_mux4[7:4]),
        .in2(mux2_mux4[11:8]),
        .in3(mux2_mux4[15:12]),
        .out(mux4_decoder)
    );

    wire[6:0] decoder_mux2;
    sseg_decoder decoder0(
        .num(mux4_decoder),
        .sseg(decoder_mux2)
    );

    mux2 #(.BITS(7)) mux2_1(
        .sel(sel_mux2_7),
        .in0(decoder_mux2),
        .in1(7'b0111111),
        .out(seg)
    );


endmodule
```

Listing 5: On-Board Implementation

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

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

endmodule
```

Screenshot

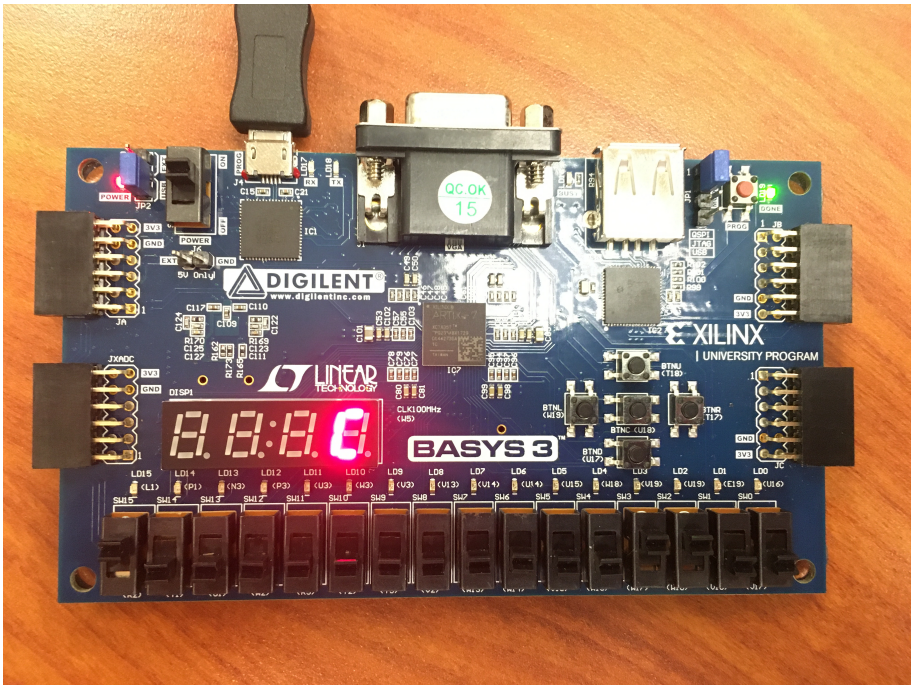

Figure 4: 1st Operation

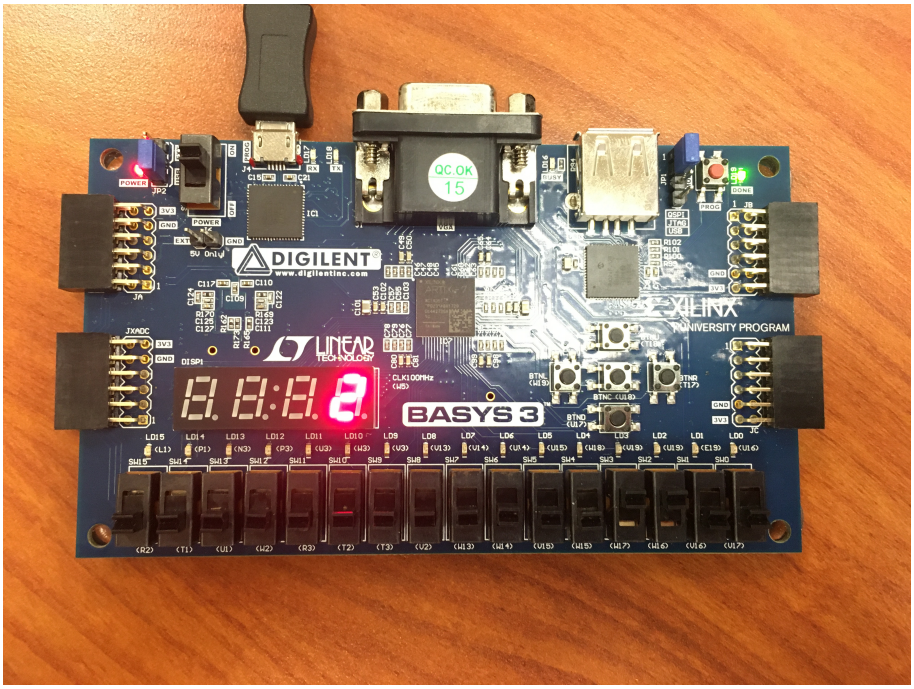

Figure 5: 2nd Operation

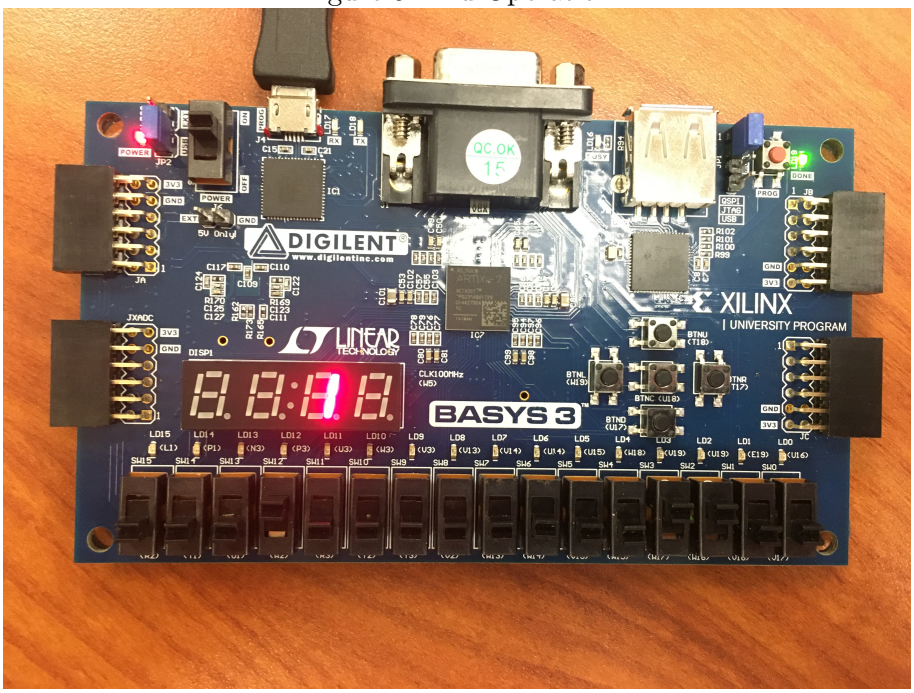

Figure 6: 3rd Operation

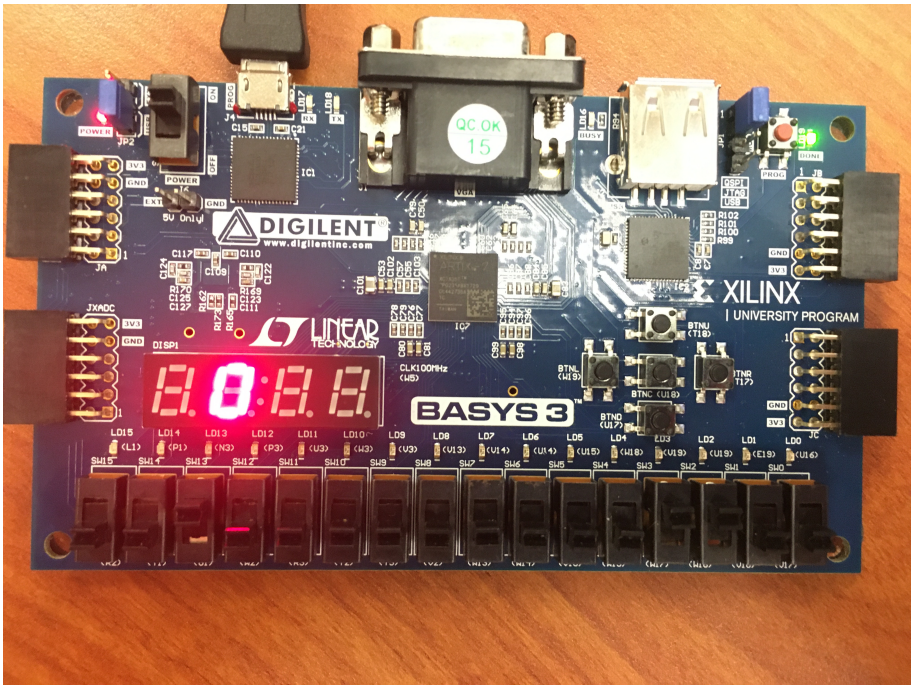

Figure 7: 4th Operation

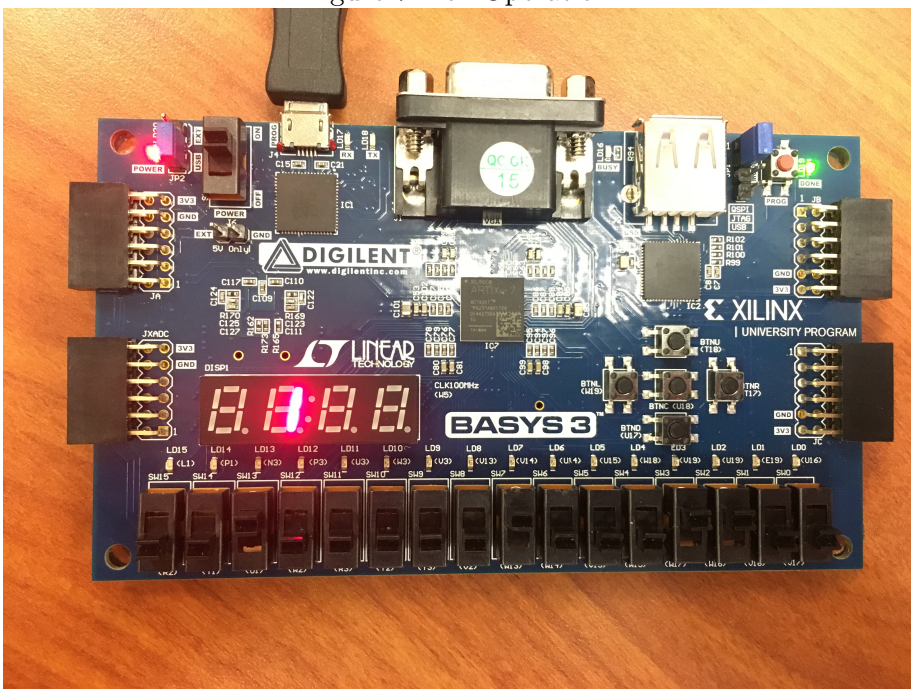

Figure 8: 5th Operation

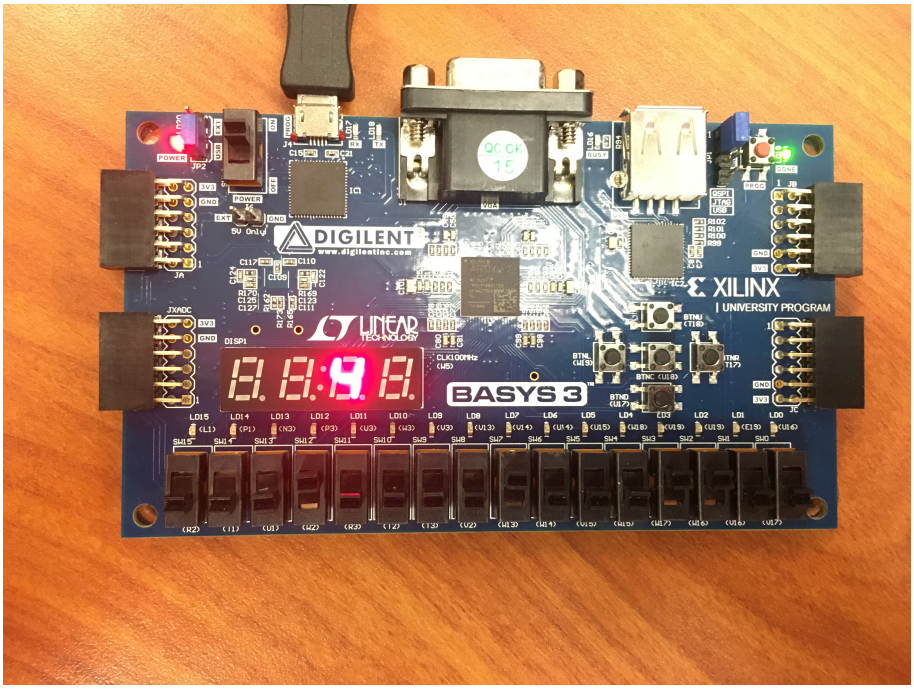

Figure 9: 6th Operation

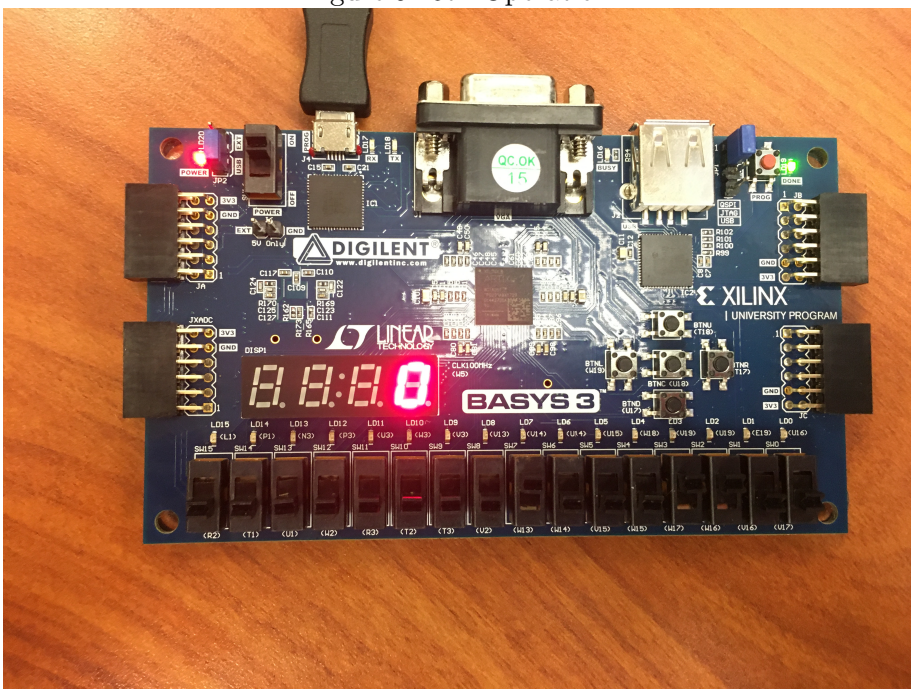

Figure 10: 7th Operation

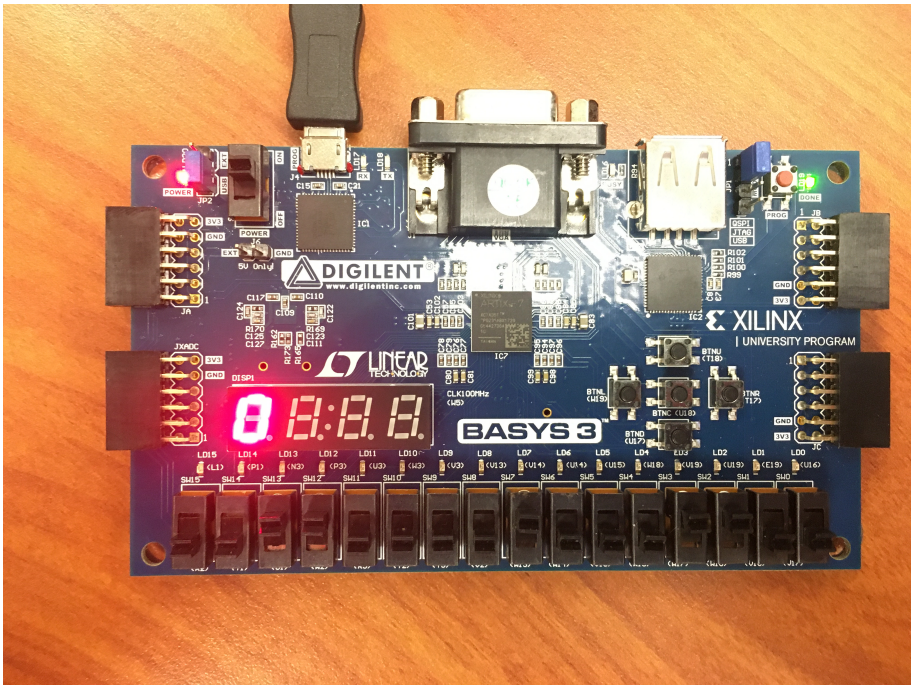

Figure 11: 8th Operation

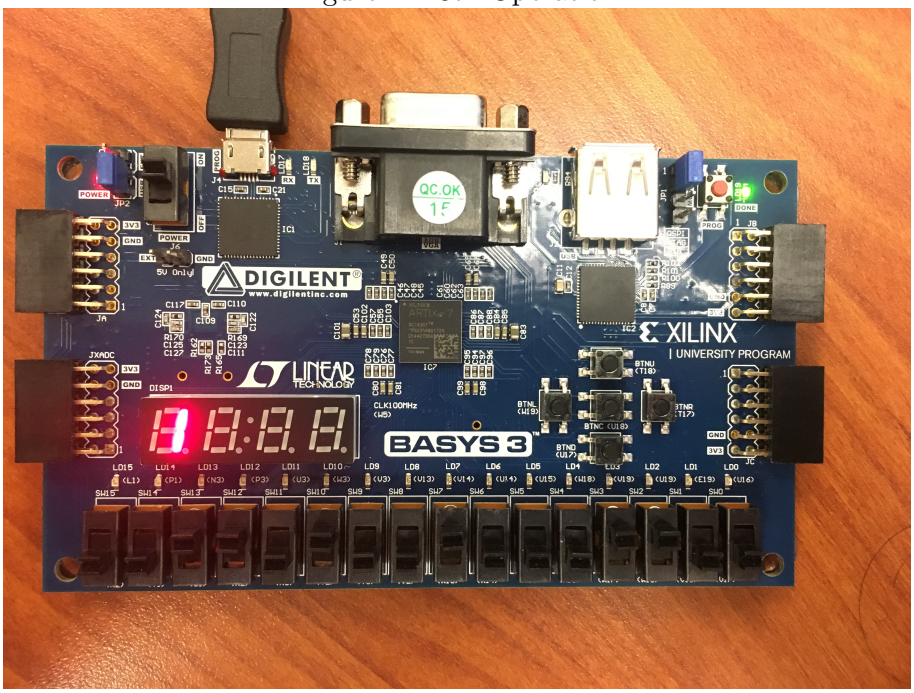

Figure 12: 9th Operation

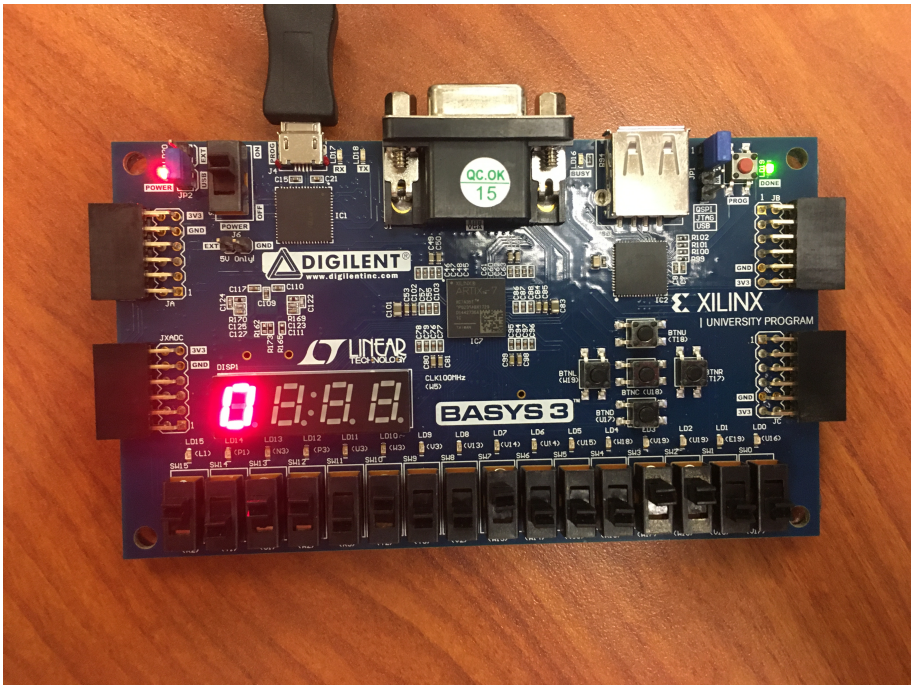

Figure 13: 10th Operation

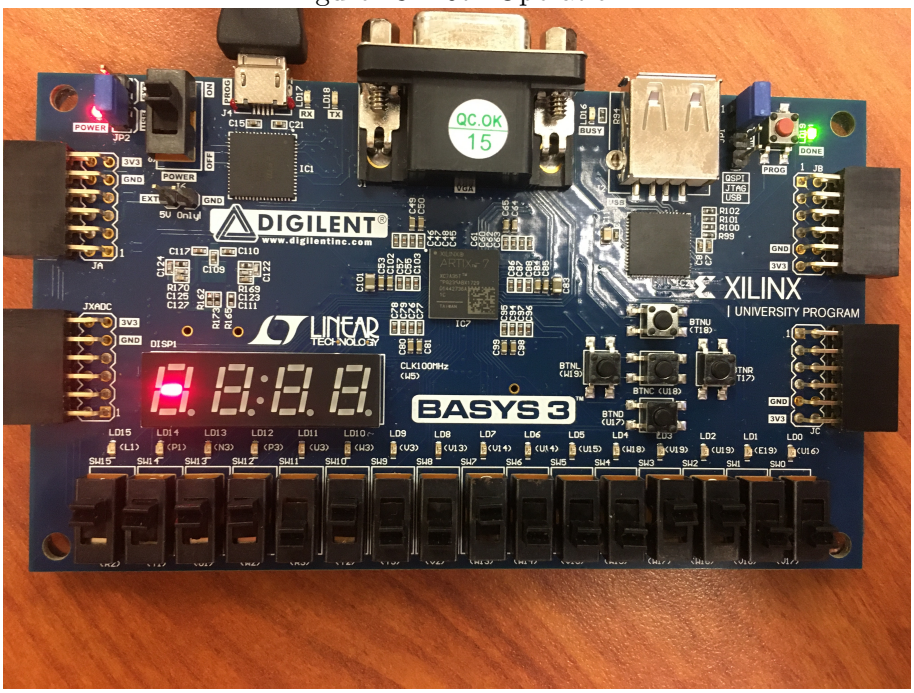

Figure 14: 11th Operation

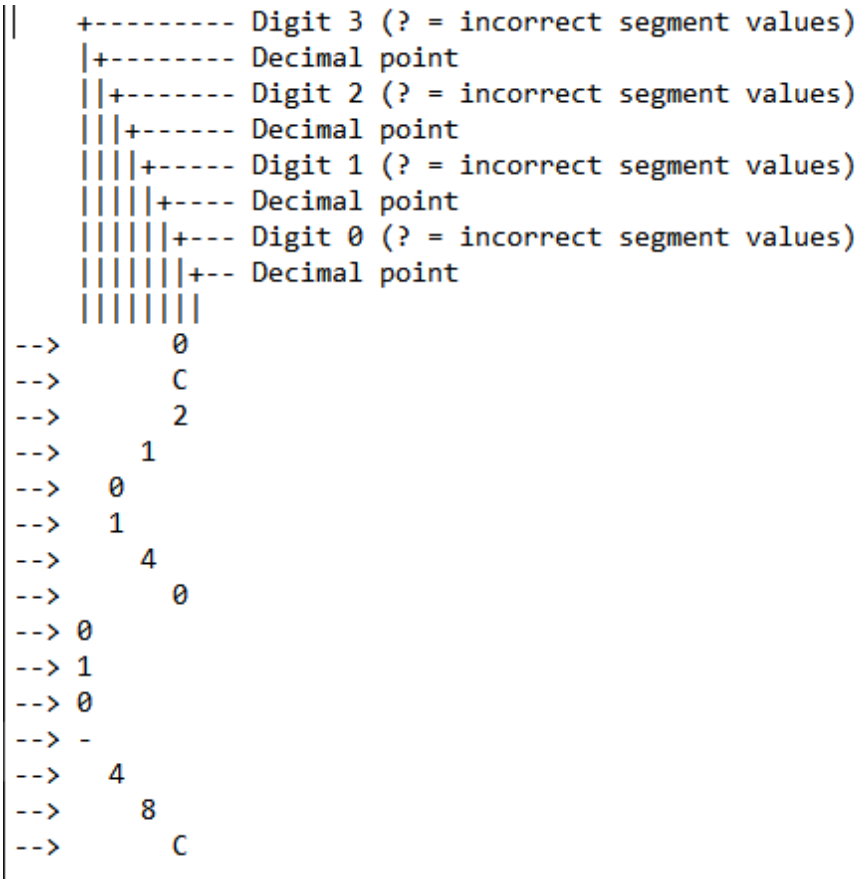

Figure 15: Simulation Results