Alan Rieger and Dylan Thornburg
ELEN 21L Lab 3
4/25/23

Introduction

This week's lab's purpose was to simulate building a light system for a highway on-ramp. We intend to build a circuit where only one light is ever on at a time and changes per lane depending on if cars in each lane are present. The carpool lane had priority, there was a round robin decider if there were cars in the left and right lane, and if no cars were present, the right light was on by default. For the pre-lab, we made truth tables for the lights and realized it would be more efficient to utilize the POS equations instead of SOP.

Procedures

At the start of this lab, we created a new Quartus file. Next, we created the circuit based on our POS equation that we found in the Pre-Lab. Next, we tested the circuit using a timing diagram and found that it matched our truth table in the pre-lab which means it worked. Next, we finished writing some verilog code given to us to create a symbol that, when implemented into a circuit, allowed us to use the 7-segment display to tell us how many cars are waiting at the entrance ramp. To implement this 7-segment circuit we had to write some K-maps and find the POS equations and create the final circuit. Lastly, we compiled, fixed some errors with wiring, and uploaded the circuit onto the FPGA, tested and demoed to the TA where we found it was correct.

Questions

Q: Describe the simulation strategy that you used to test your circuit design. Did it identify any errors in design or implementation? If so, what were they and how did you correct them?

A: We used the timing diagram simulation to make sure our circuit worked. Anytime we had an error, we used the drop down arrow to see exactly what went wrong and then fix it.

Q: Do you think your simulation strategy would detect all design errors for this circuit before downloading the circuit to the FPGA? Why or why not?

A: I believe it would catch every logic error but not a physical FPGA error if hypothetically the FPGA was faulty. There's no reason I can think of besides a faulty FPGA on why our circuit wouldn't work if it worked in the simulation.

Q: When you tested your circuit operation on the FPGA, did you find any design or implementation errors that were not identified in the simulation? If so, describe the errors and discuss why the simulation did not reveal them.

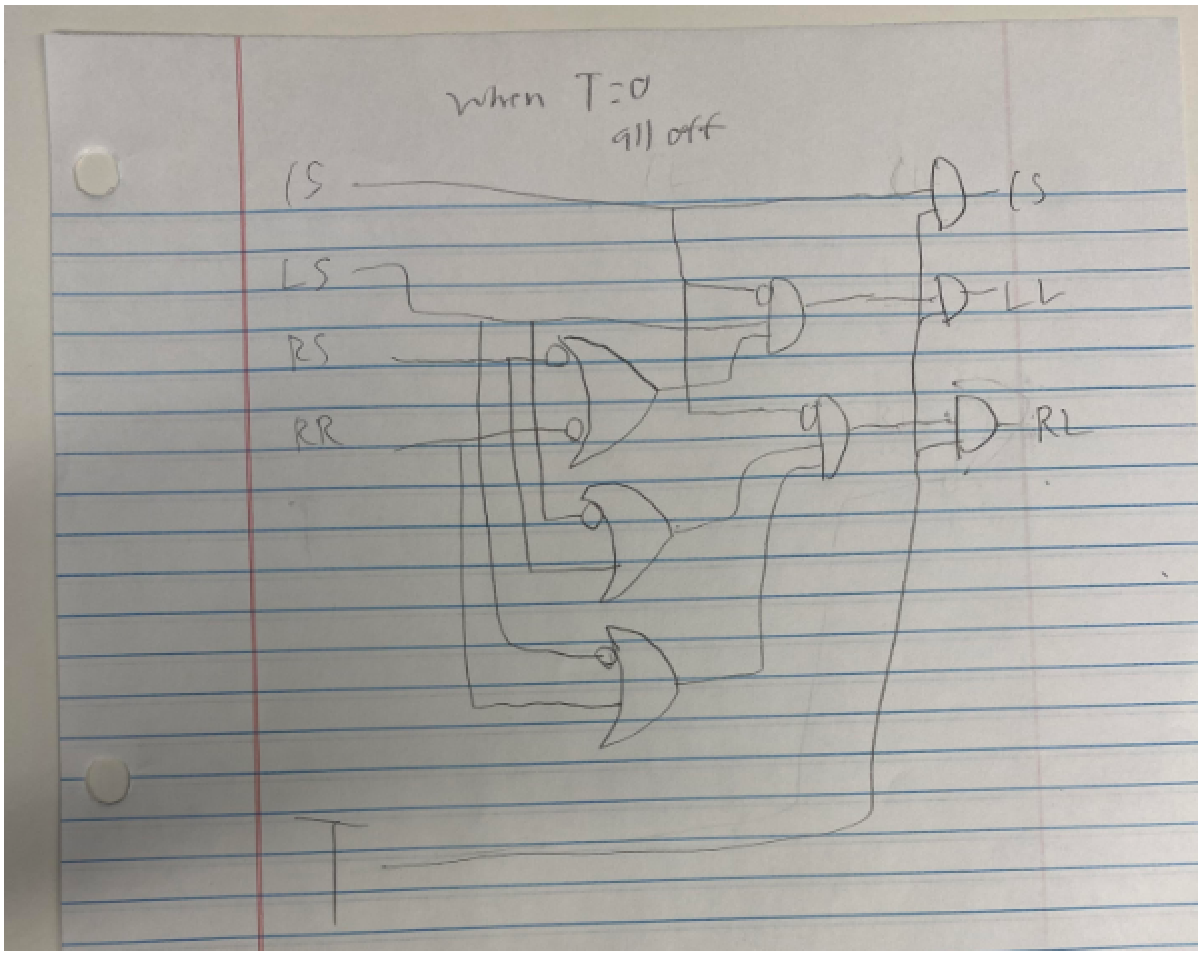
A: No, we did not find any errors. It worked perfectly on the FPGA the first time after we got a perfect simulation.

Q: What logic would you add to create a new output, ERR1 which would be 1 if two or more lights (CL, LL, and RL) were turned on at the same time. How is that logic similar to the logic that created the T1 output?

A: I suppose we would add a couple different "and" gates with the lights as inputs and then if it's ever true, the ERR1 would turn on. However, this shouldn't happen if you just implement the circuit correctly.

Q: Describe how you would modify your circuit to include a fifth input TM, a timer that is turned on at intervals controlled by the traffic density. When TM is "1", the circuit operates exactly as specified in the problem statement. When TM is "0", all output lights are red. Show a schematic for your modified circuit which includes the timer input.

A: We would add and gates stemming from the standard circuit outputs and T to make it so that for the circuit to behave normally, T must equal 1, otherwise all and gates will be zero and hence all lights "off" (or red).



Q:· In the specification for the circuit, the carpool lane always has priority. If there were a long line of cars in the carpool lane, all other traffic would stop completely until the last car in the carpool lane had entered the highway. Describe a possible strategy to prevent total blocking of the cars in the non-carpool lanes but still allow cars in the carpool lanes to wait less time than other cars.

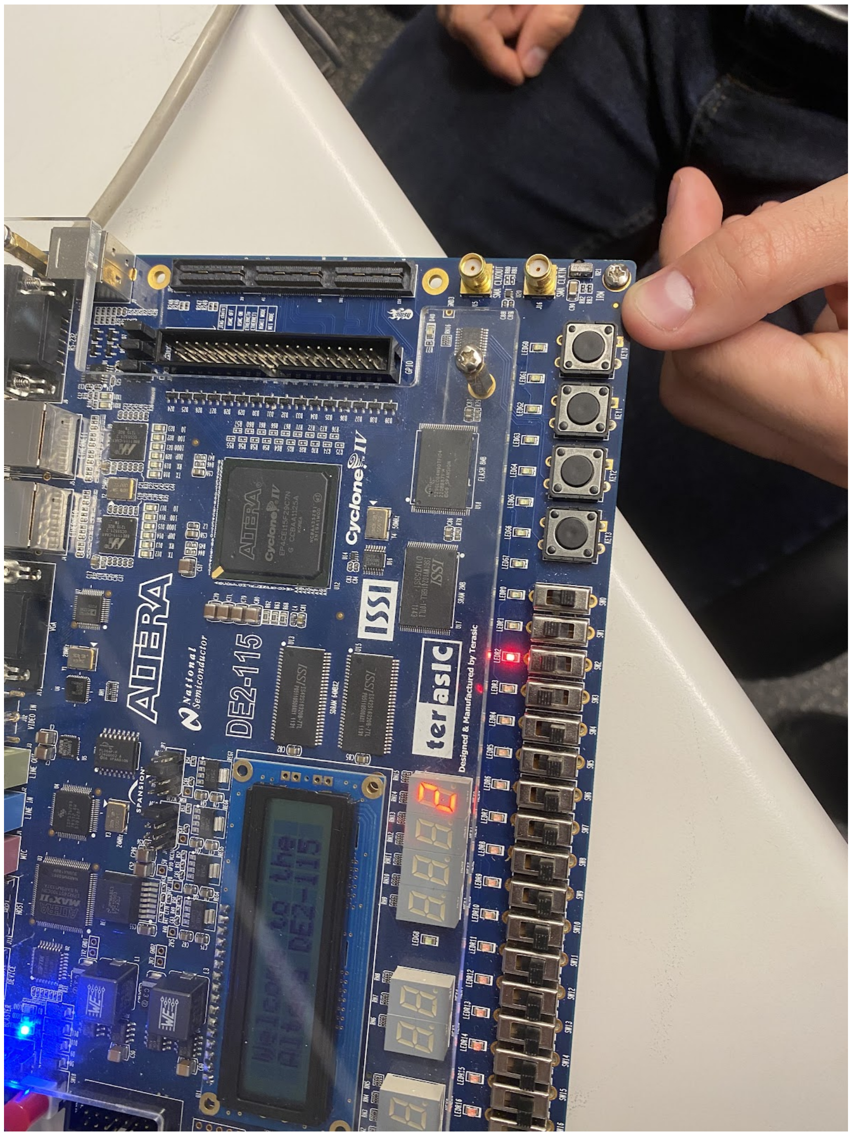
A: You could have a second priority that after allowing 2 cars to come through the carpool lane, the lights shift to letting one car from both the right and left lanes through. This would allow the carpool lane to still go faster since it would go two cars at a time while the other two lanes would only go one car at a time.

Conclusion

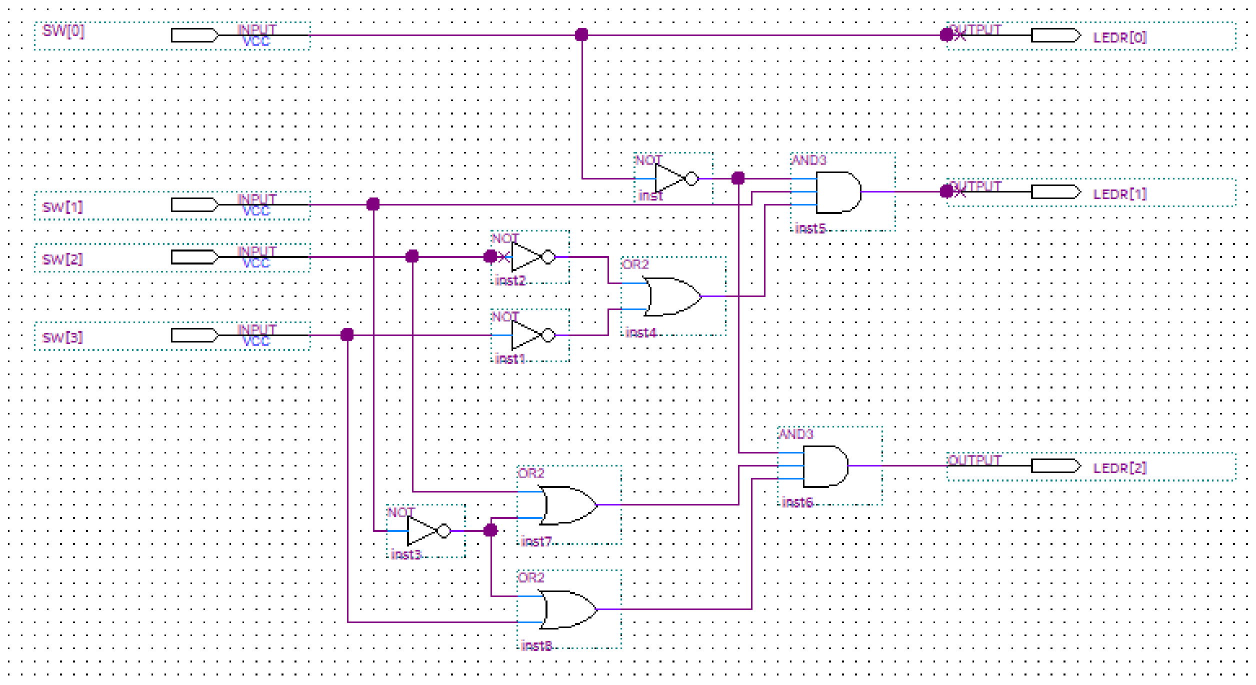
In the end, the pre-lab circuit that we created helped a lot because we were able to go straight into creating it on Quartus. We were able to test that circuit and see right away that it would work with the design we made. We learned that when implementing a 7-segment display in Quartus you can name wires to have them all go into one input without having to physically connect it. We also learned that verilog code can be transformed into a symbol to be used in circuit design. Overall, this lab was very interesting to see how multiple circuits can be created in Quartus and uploaded onto an FPGA to run simultaneously.

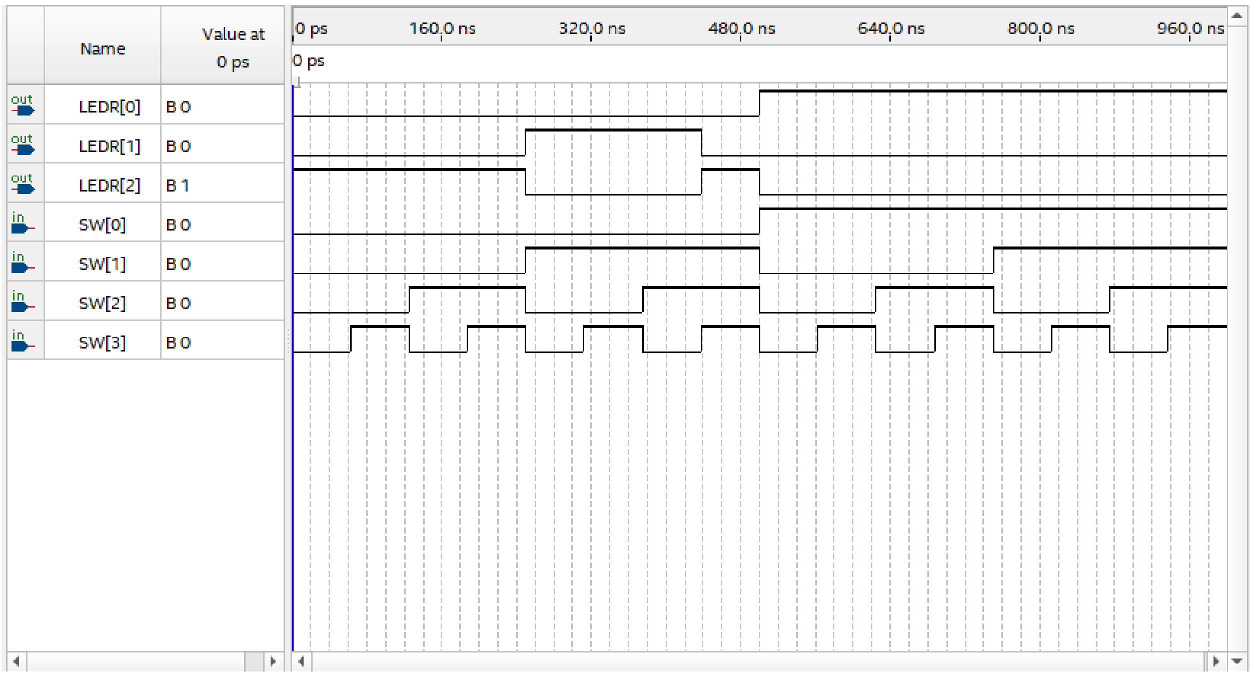
Photos

FPGA Working: (Figure 1)



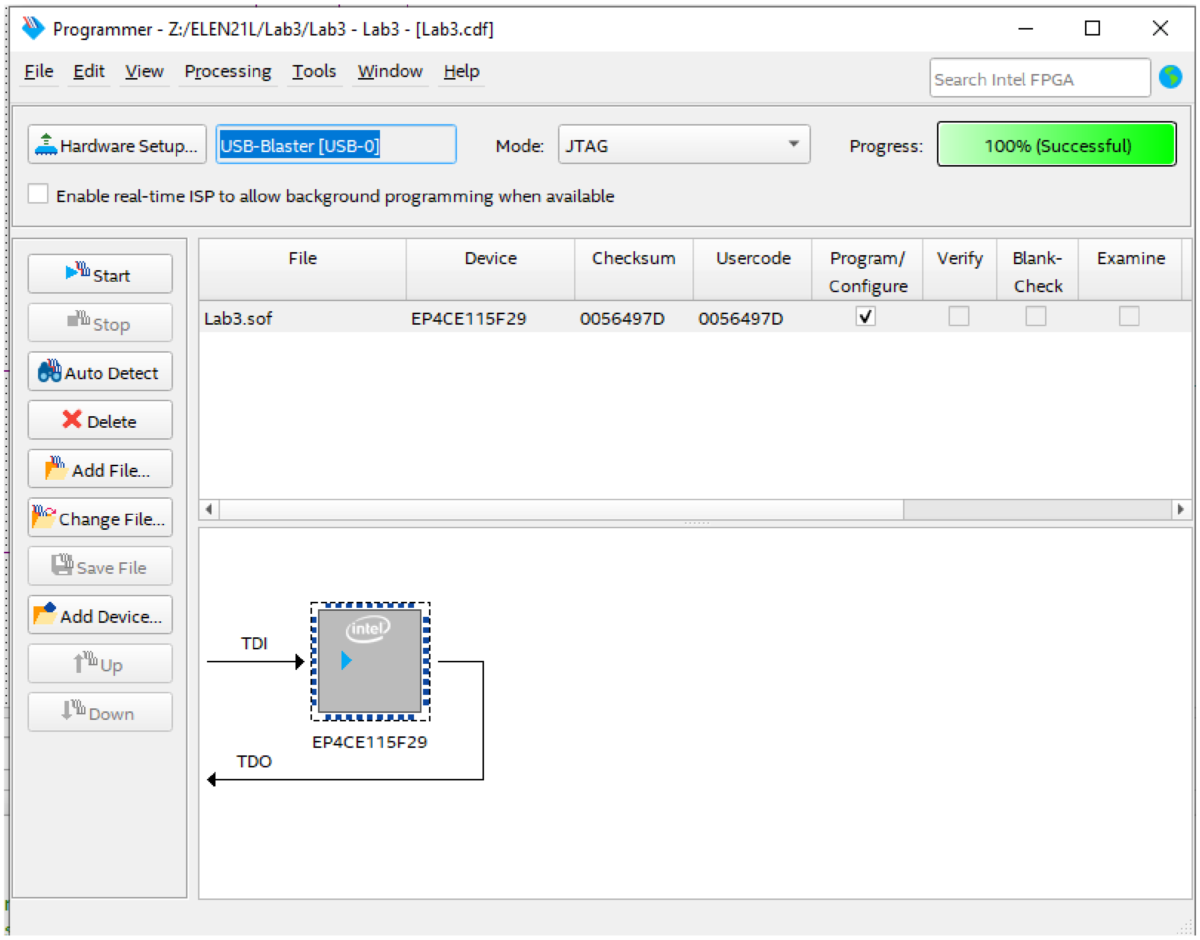
Car Entrance Ramp Circuit and Time Diagram (Figure 2 and 3):





	Name	Value at 0 ps
out	LEDR[0]	B 0
out	LEDR[1]	B 0
out	LEDR[2]	B 1
in	SW[0]	B 0
in	SW[1]	B 0
in	SW[2]	B 0
in	SW[3]	B 0

Successful Upload to FPGA and Full Circuit including 7-segment Display (Figure 4 and 5):



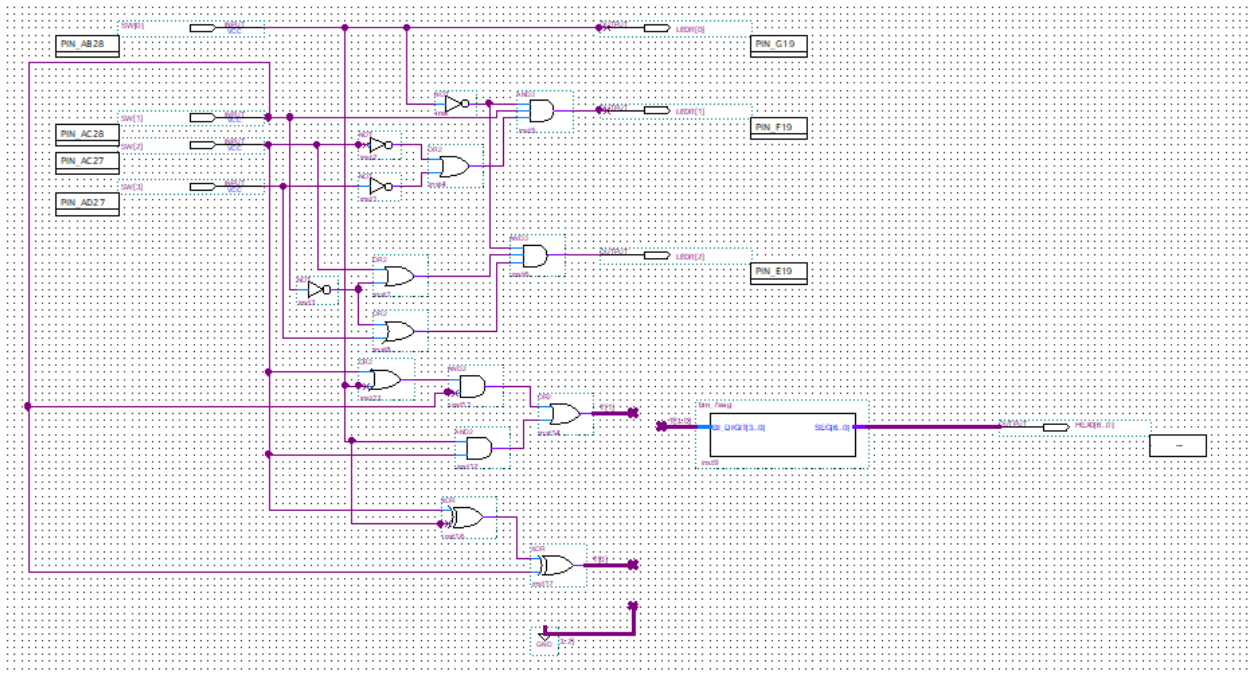
Programmer - Z:/ELEN21L/Lab3/Lab3 - Lab3 - [Lab3.cdf]

File Edit View Processing Tools Window Help

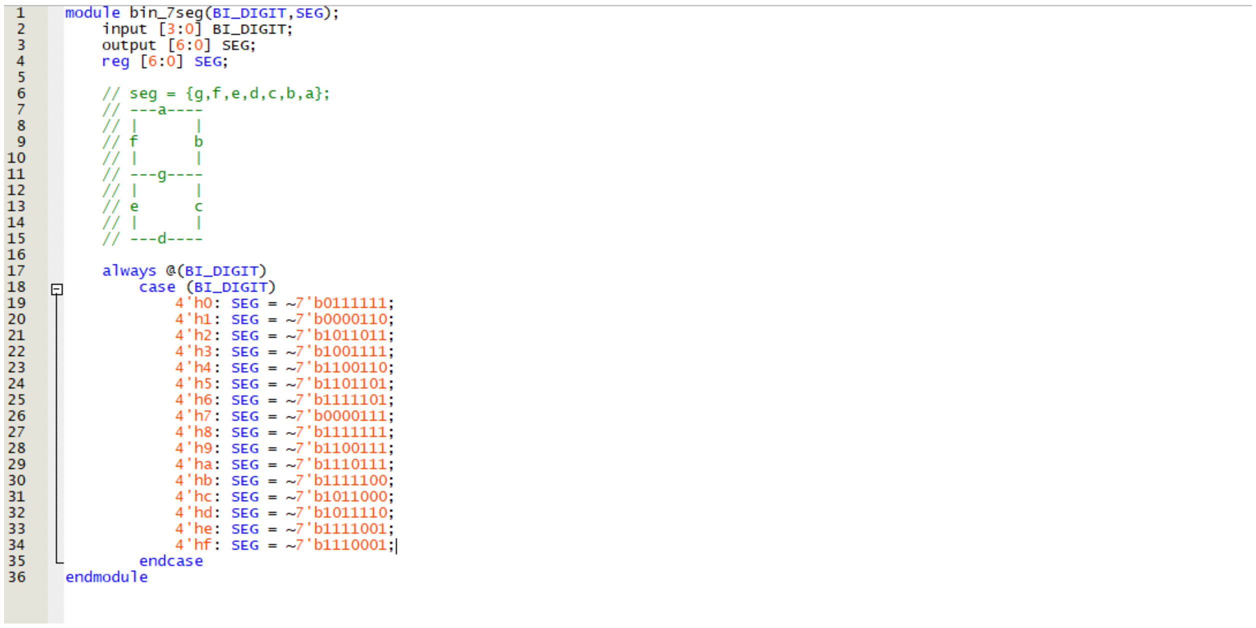
Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
Lab3.sof	EP4CE115F29	0056497D	0056497D	✓			



Verilog Code (Figure 6):



```verilog
module bin_7seg(BI_DIGIT,SEG);
    input [3:0] BI_DIGIT;
    output [6:0] SEG;
    reg [6:0] SEG;

    // seg = {g,f,e,d,c,b,a};
    // ---a----
    // |      |
    // f      b
    // |      |
    // ---g----
    // |      |
    // e      c
    // |      |
    // ---d----

    always @(BI_DIGIT)
        case (BI_DIGIT)
            4'h0: SEG = ~7'b0111111;
            4'h1: SEG = ~7'b0000110;
            4'h2: SEG = ~7'b1011011;
            4'h3: SEG = ~7'b1001111;
            4'h4: SEG = ~7'b1100110;
            4'h5: SEG = ~7'b1101101;
            4'h6: SEG = ~7'b1111101;
            4'h7: SEG = ~7'b0000111;
            4'h8: SEG = ~7'b1111111;
            4'h9: SEG = ~7'b1100111;
            4'ha: SEG = ~7'b1110111;
            4'hb: SEG = ~7'b1111100;
            4'hc: SEG = ~7'b1011000;
            4'hd: SEG = ~7'b1011110;
            4'he: SEG = ~7'b1111001;
            4'hf: SEG = ~7'b1110001;
        endcase
endmodule
```