UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	**CPE-300L**	Semester:	**Spring 2020**
Points		Document author:	**Brysen Kokubun**
		Author's email:	**Brysenkokubun96@gmail.com**
		Document topic:	**Postlab 3**
Instructor's comments:			

1. Introduction / Theory of Operation

- Lab experiment #3 familiarize with ModelSim and its software for testbenches. We will also be reviewing and designing combinational circuits. Some of the circuits being designed are Ripple adder, ALU, implementing 7-seg on DE2 board, and Comparator.

1. What is the mega function in Quartus? List applications.
   - Mega function is a black box containing a logic design. The mega function can be used as an instantiation of other functions in a design file and inputs are parameters to the function.
2. What is continuous assignment and procedural assignment in Verilog?
   - Continuous assignments are structural logic connections and can be used to symbolize combinational logic. Procedural assignments simply assign values to variables, registers, etc.

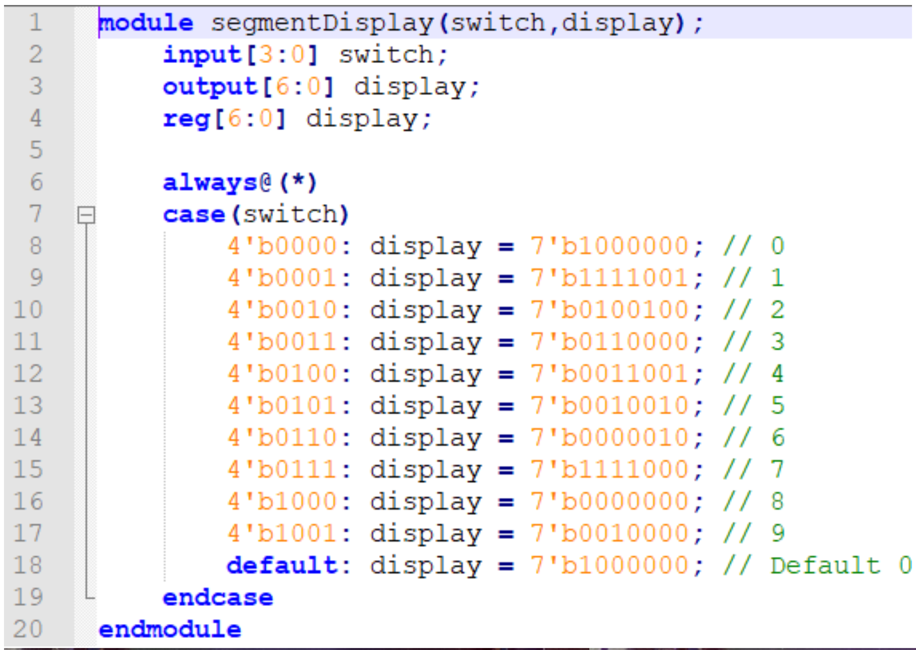
2. Prelab

- Attached

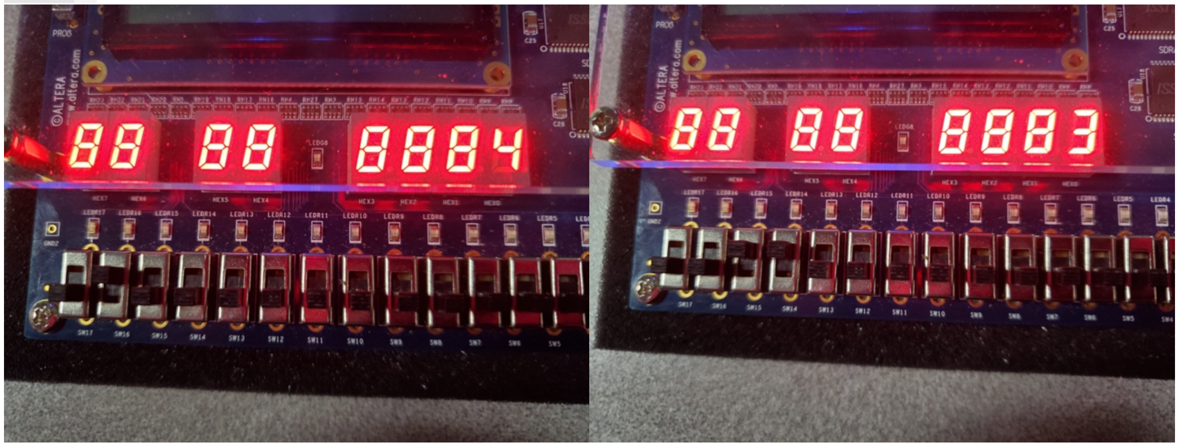
3. Experiment Results

Experiment #1: 7-Segment in Verilog

- Wrote Verilog code to use 7-segment on the DE2 board using toggle switches.



```verilog
module segmentDisplay(switch,display);
    input[3:0] switch;
    output[6:0] display;
    reg[6:0] display;

    always@(*)
    case(switch)
        4'b0000: display = 7'b1000000; // 0
        4'b0001: display = 7'b1111001; // 1
        4'b0010: display = 7'b0100100; // 2
        4'b0011: display = 7'b0110000; // 3
        4'b0100: display = 7'b0011001; // 4
        4'b0101: display = 7'b0010010; // 5
        4'b0110: display = 7'b0000010; // 6
        4'b0111: display = 7'b1111000; // 7
        4'b1000: display = 7'b0000000; // 8
        4'b1001: display = 7'b0010000; // 9
        default: display = 7'b1000000; // Default 0
    endcase
endmodule
```

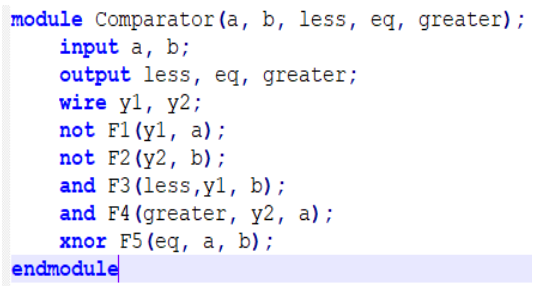


Here are 2 examples of 7-segment on DE2 board 4 = 0100, second 3 = 0011.

Experiment #2: Magnitude Comparator from prelab 3

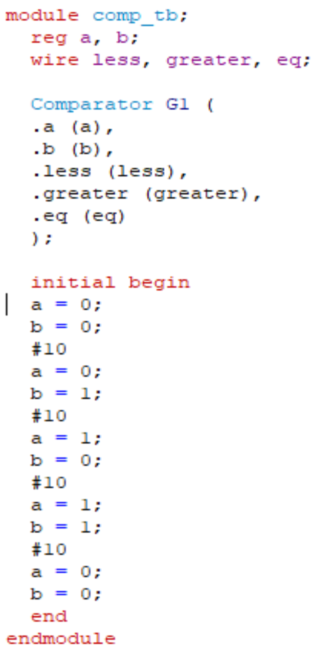
- Comparing two 1-bit values for magnitude. 3 cases a<b, a = b, a >b

Verilog Code



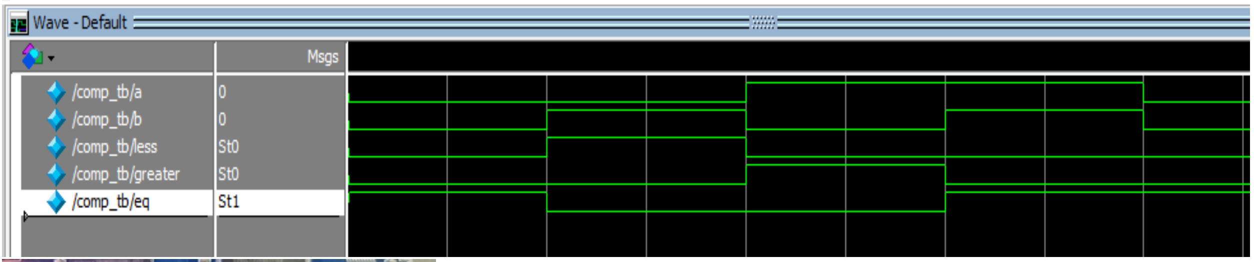
```verilog
module Comparator(a, b, less, eq, greater);
    input a, b;
    output less, eq, greater;
    wire y1, y2;
    not F1(y1, a);
    not F2(y2, b);
    and F3(less,y1, b);
    and F4(greater, y2, a);
    xnor F5(eq, a, b);
endmodule
```

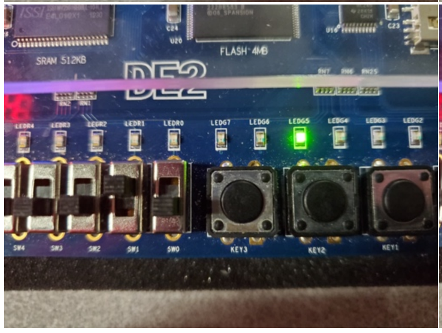
Test Bench Code



```verilog
module comp_tb;
  reg a, b;
  wire less, greater, eq;

  Comparator G1 (
  .a (a),
  .b (b),
  .less (less),
  .greater (greater),
  .eq (eq)
  );

  initial begin
  a = 0;
  b = 0;
  #10
  a = 0;
  b = 1;
  #10
  a = 1;
  b = 0;
  #10
  a = 1;
  b = 1;
  #10
  a = 0;
  b = 0;
  end
endmodule
```

Waveform



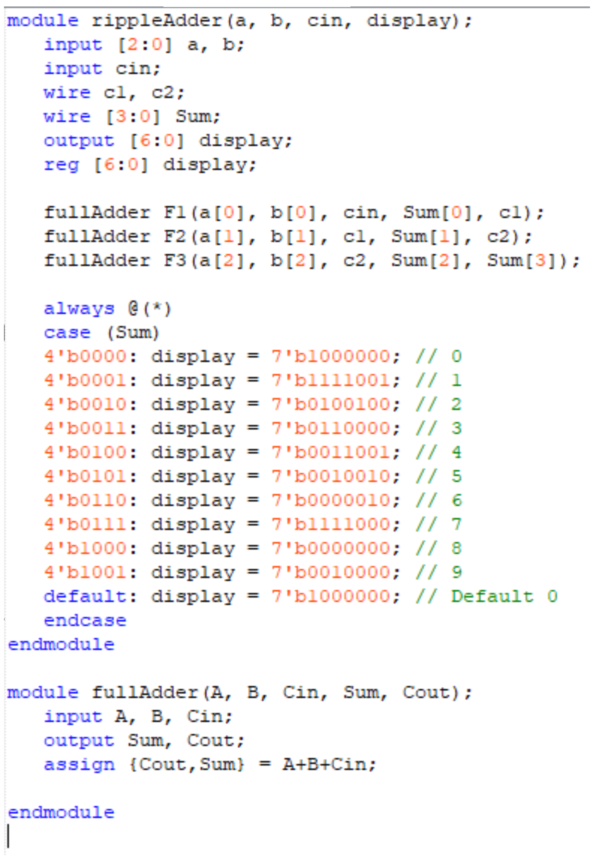


00 equal to (Middle LED ON)
01 less than (Left LED ON)
10 greater than (Right LED ON)
11 equal to (Middle LED ON)

Experiment #3: Ripple Adder

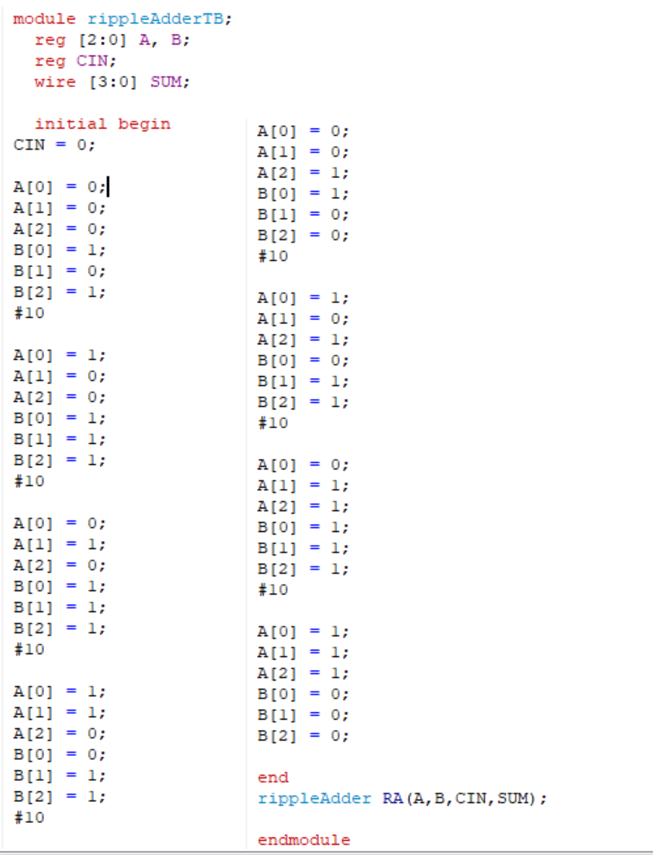
- Designed a 3-bit ripple adder and implemented on the DE2 board with 7-segment

Verilog Code

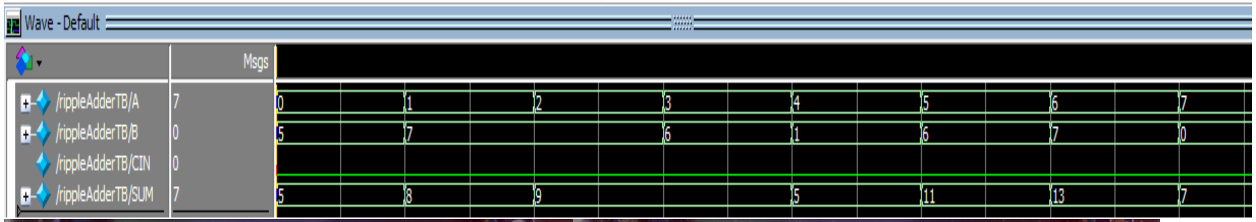


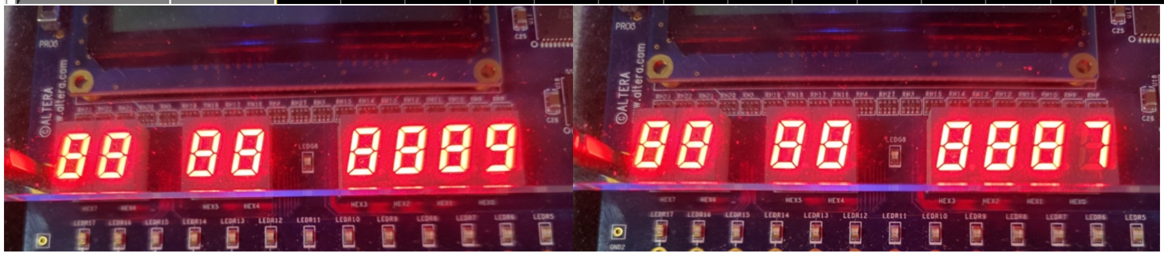
```
module rippleAdder(a, b, cin, display);
   input [2:0] a, b;
   input cin;
   wire c1, c2;
   wire [3:0] Sum;
   output [6:0] display;
   reg [6:0] display;

   fullAdder F1(a[0], b[0], cin, Sum[0], c1);
   fullAdder F2(a[1], b[1], c1, Sum[1], c2);
   fullAdder F3(a[2], b[2], c2, Sum[2], Sum[3]);

   always @(*)
   case (Sum)
   4'b0000: display = 7'b1000000; // 0
   4'b0001: display = 7'b1111001; // 1
   4'b0010: display = 7'b0100100; // 2
   4'b0011: display = 7'b0110000; // 3
   4'b0100: display = 7'b0011001; // 4
   4'b0101: display = 7'b0010010; // 5
   4'b0110: display = 7'b0000010; // 6
   4'b0111: display = 7'b1111000; // 7
   4'b1000: display = 7'b0000000; // 8
   4'b1001: display = 7'b0010000; // 9
   default: display = 7'b1000000; // Default 0
   endcase
endmodule

module fullAdder(A, B, Cin, Sum, Cout);
   input A, B, Cin;
   output Sum, Cout;
   assign {Cout,Sum} = A+B+Cin;

endmodule
```

Test Bench Code



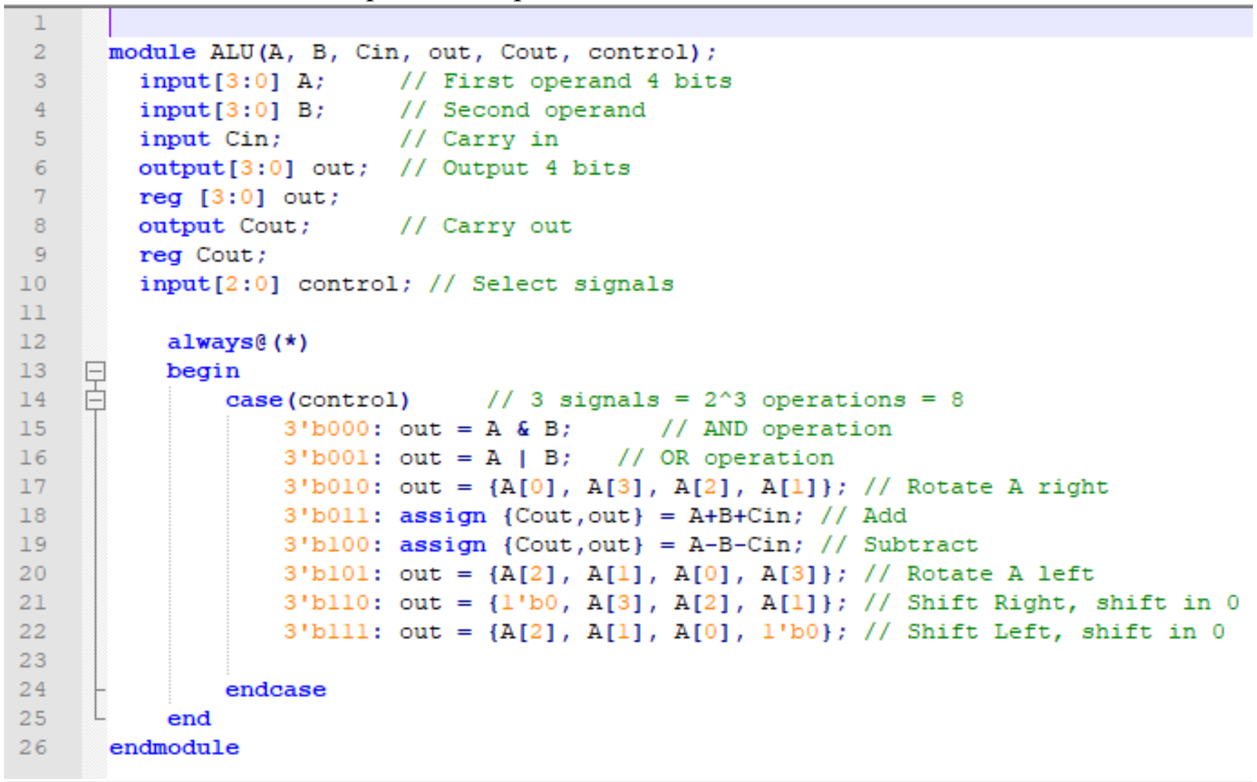
```
module rippleAdderTB;
  reg [2:0] A, B;
  reg CIN;
  wire [3:0] SUM;

  initial begin
CIN = 0;

A[0] = 0;
A[1] = 0;
A[2] = 0;
B[0] = 1;
B[1] = 0;
B[2] = 1;
#10

A[0] = 1;
A[1] = 0;
A[2] = 0;
B[0] = 1;
B[1] = 1;
B[2] = 1;
#10

A[0] = 0;
A[1] = 1;
A[2] = 0;
B[0] = 1;
B[1] = 1;
B[2] = 1;
#10

A[0] = 1;
A[1] = 1;
A[2] = 0;
B[0] = 0;
B[1] = 1;
B[2] = 1;
#10

A[0] = 0;
A[1] = 0;
A[2] = 1;
B[0] = 1;
B[1] = 0;
B[2] = 0;
#10

A[0] = 1;
A[1] = 0;
A[2] = 1;
B[0] = 0;
B[1] = 1;
B[2] = 1;
#10

A[0] = 0;
A[1] = 1;
A[2] = 1;
B[0] = 1;
B[1] = 1;
B[2] = 1;
#10

A[0] = 1;
A[1] = 1;
A[2] = 1;
B[0] = 0;
B[1] = 0;
B[2] = 0;

end
rippleAdder RA(A,B,CIN,SUM);

endmodule
```


Wave - Default
Msgs
/rippleAdderTB/A 7 | 0 1 2 3 4 5 6 7
/rippleAdderTB/B 0 | 5 7 6 1 6 7 0
/rippleAdderTB/CIN 0
/rippleAdderTB/SUM 7 | 5 8 9 5 11 13 7

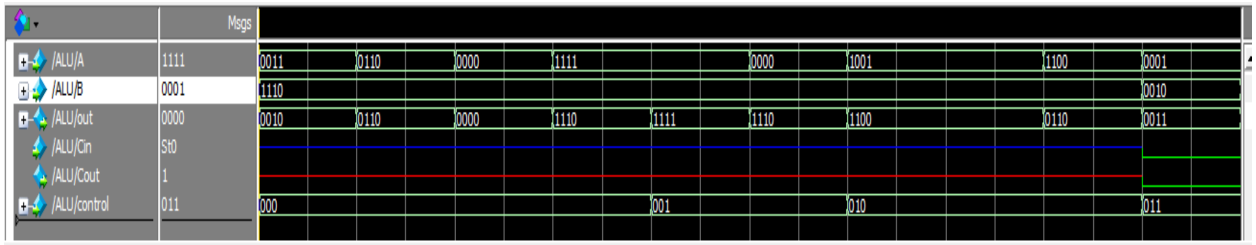


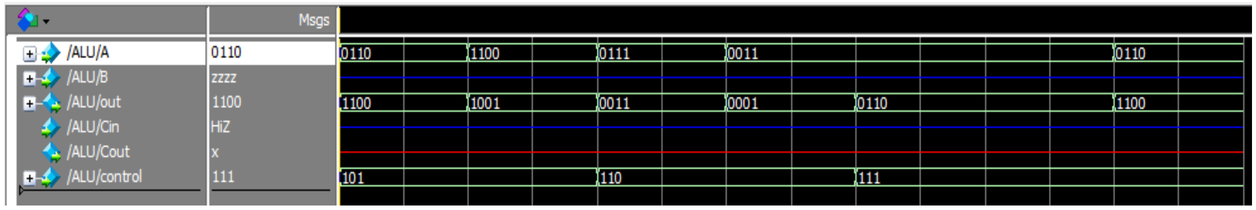
Here are 2 examples of the outputs from the RippleAdder on the DE2 board.

Experiment #4: Simple ALU
- 4-bit ALU with 8 opcodes of operations



```
module ALU(A, B, Cin, out, Cout, control);
  input[3:0] A;      // First operand 4 bits
  input[3:0] B;      // Second operand
  input Cin;         // Carry in
  output[3:0] out;   // Output 4 bits
  reg [3:0] out;
  output Cout;       // Carry out
  reg Cout;
  input[2:0] control; // Select signals

    always@(*)
    begin
        case(control)     // 3 signals = 2^3 operations = 8
            3'b000: out = A & B;       // AND operation
            3'b001: out = A | B;    // OR operation
            3'b010: out = {A[0], A[3], A[2], A[1]}; // Rotate A right
            3'b011: assign {Cout,out} = A+B+Cin; // Add
            3'b100: assign {Cout,out} = A-B-Cin; // Subtract
            3'b101: out = {A[2], A[1], A[0], A[3]}; // Rotate A left
            3'b110: out = {1'b0, A[3], A[2], A[1]}; // Shift Right, shift in 0
            3'b111: out = {A[2], A[1], A[0], 1'b0}; // Shift Left, shift in 0

        endcase
    end
endmodule
```





4. Questions

1. Why ModelSim is better than Quartus internal simulator?

   - Both ModelSim and Quartus can run simulations but ModelSim is created more for simulation and testing of projects while Quartus is used more for configurations and programing devices.

2. How does initial block differ from always block?

   - The initial block only runs once during program execution and the always block will always run in order to check for signal changes.

3. What is the difference between $finish and $stop in Verilog testbench?

   - Finish will complete the testbench code and stop will stop the testbench code in its position.

5. Encountered Problems

- For this lab experiment we had difficulty applying our Verilog code to assign to certain pins on the 7-seg display on the DE2 board. But after writing out the logic, we were able to correctly pin assign. Also, some of the labs do not have the proper version of Quartus installed which created an issue of where we could work on the experiment at school.

6. Conclusions

- To conclude this lab, we used both Quartus and ModelSim to compile, simulate, and create waveforms for our experiments. We successfully created and implemented a comparator, Ripple adder, and ALU. We were able to create testbenches to easily see outputs on waveforms. We also implemented our Verilog codes on the DE2 boards 7-seg and switches. This lab helped us get a better handling of Quartus and ModelSim and what type of functions they have to offer.