Lab 10: A Simple Digital Combination Lock

ECEN 248 - 505

TA: Younggyun Cho

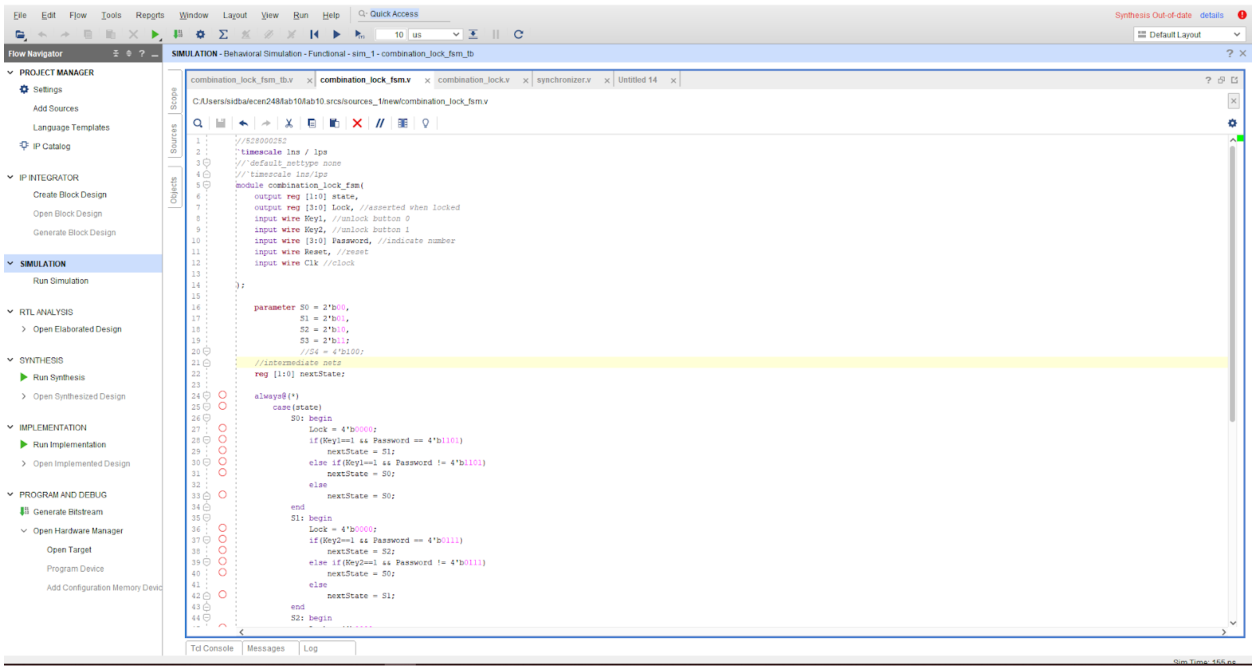
Date: November 10, 2020

Objectives

The purpose of this lab is to design the actions of a rotary combination lock by writing code for a digital circuit and uploading it to a zybo board. The student must ensure that the circuit can detect when the right combination is entered,and it will give us the “unlocked” message. Otherwise they must ensure when the wrong combination is entered and give the “locked” message. This lab demonstrates the use of the Moore finite state machine.

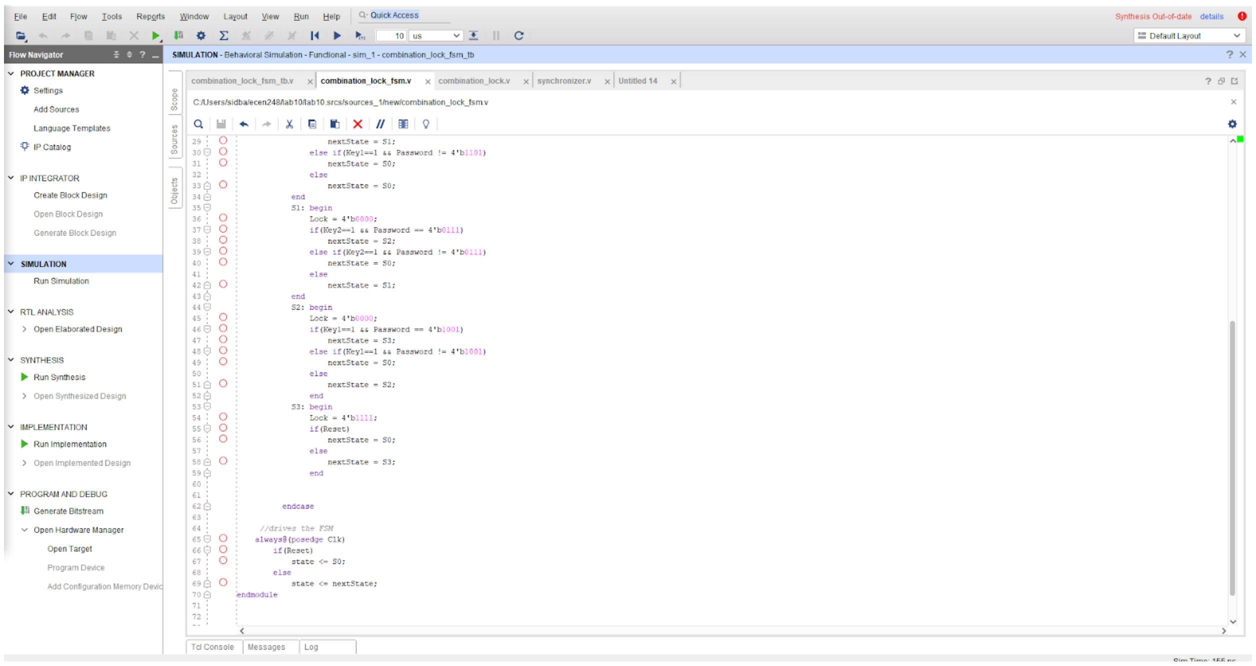
Design

//Combination lock FSM (3 States)


SIMULATION - Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb

C:/Users/sidba/ecen248/lab10/lab10.srcs/sources_1/new/combination_lock_fsm.v

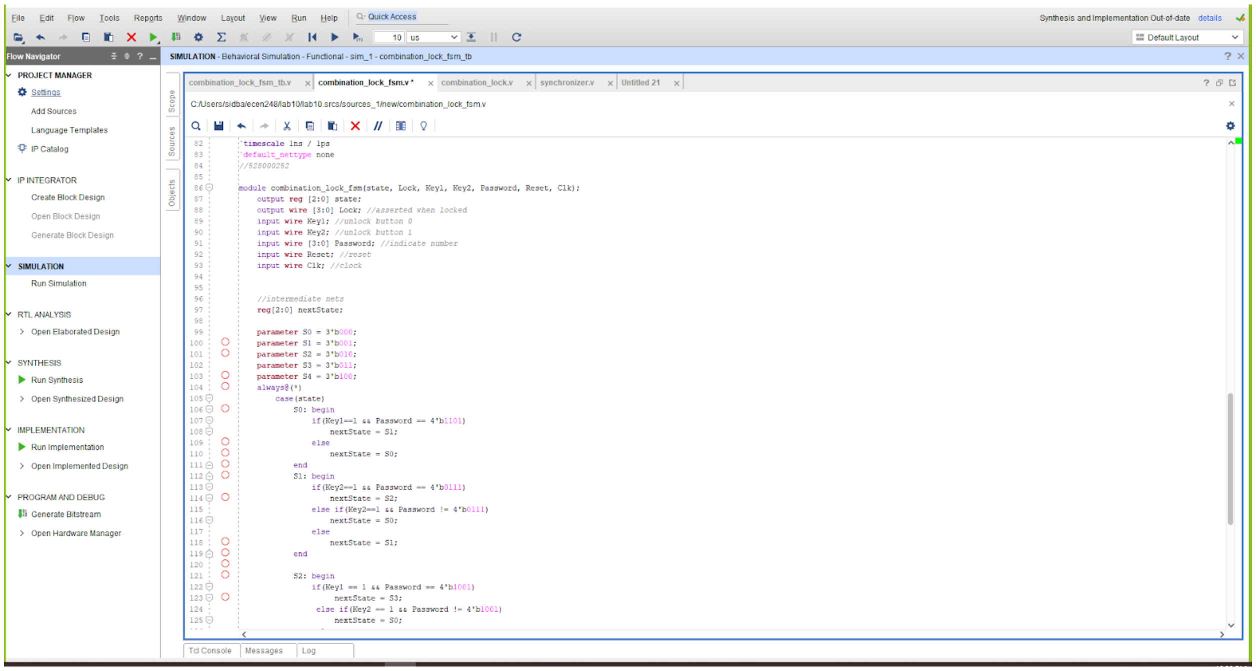
```verilog
//528000252
`timescale 1ns / 1ps
//`default_nettype none
//`timescale 1ns/1ps
module combination_lock_fsm(
    output reg [1:0] state,
    output reg [3:0] Lock, //asserted when locked
    input wire Key1, //unlock button 0
    input wire Key2, //unlock button 1
    input wire [3:0] Password, //indicate number
    input wire Reset, //reset
    input wire Clk //clock

);

    parameter S0 = 2'b00,
              S1 = 2'b01,
              S2 = 2'b10,
              S3 = 2'b11;
              //S4 = 4'b100;
    //intermediate nets
    reg [1:0] nextState;

    always@(*)
        case(state)
            S0: begin
                Lock = 4'b0000;
                if(Key1==1 && Password == 4'b1101)
                    nextState = S1;
                else if(Key1==1 && Password != 4'b1101)
                    nextState = S0;
                else
                    nextState = S0;
            end
            S1: begin
                Lock = 4'b0000;
                if(Key2==1 && Password == 4'b0111)
                    nextState = S2;
                else if(Key2==1 && Password != 4'b0111)
                    nextState = S0;
                else
                    nextState = S1;
            end
            S2: begin
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb

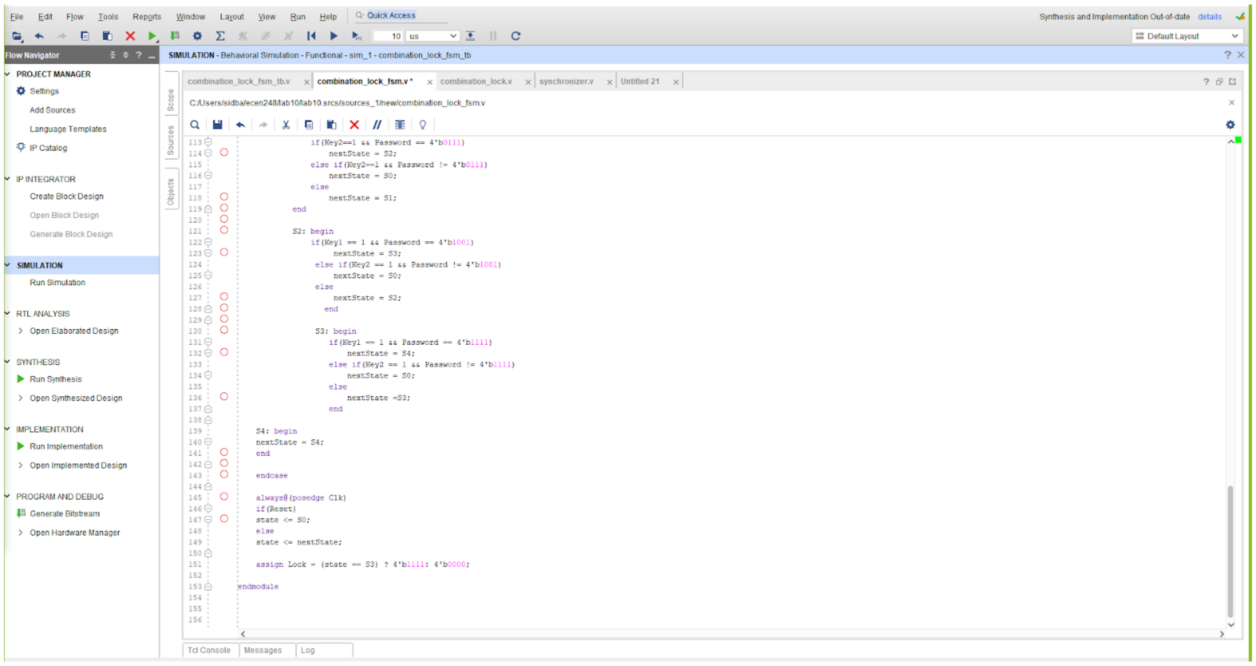
C:/Users/sidba/ecen248/lab10/lab10.srcs/sources_1/new/combination_lock_fsm.v

```verilog
                    nextState = S1;
                else if(Key1==1 && Password != 4'b1101)
                    nextState = S0;
                else
                    nextState = S0;
            end
            S1: begin
                Lock = 4'b0000;
                if(Key2==1 && Password == 4'b0111)
                    nextState = S2;
                else if(Key2==1 && Password != 4'b0111)
                    nextState = S0;
                else
                    nextState = S1;
            end
            S2: begin
                Lock = 4'b0000;
                if(Key1==1 && Password == 4'b1001)
                    nextState = S3;
                else if(Key1==1 && Password != 4'b1001)
                    nextState = S0;
                else
                    nextState = S2;
                end
            S3: begin
                Lock = 4'b1111;
                if(Reset)
                    nextState = S0;
                else
                    nextState = S3;
                end


           endcase

     //drives the FSM
    always@(posedge Clk)
        if(Reset)
            state <= S0;
        else
            state <= nextState;
endmodule
```

//Combination lock FSM (4 States)


C:/Users/sidba/ecen248/lab10/lab10.srcs/sources_1/new/combination_lock_fsm.v

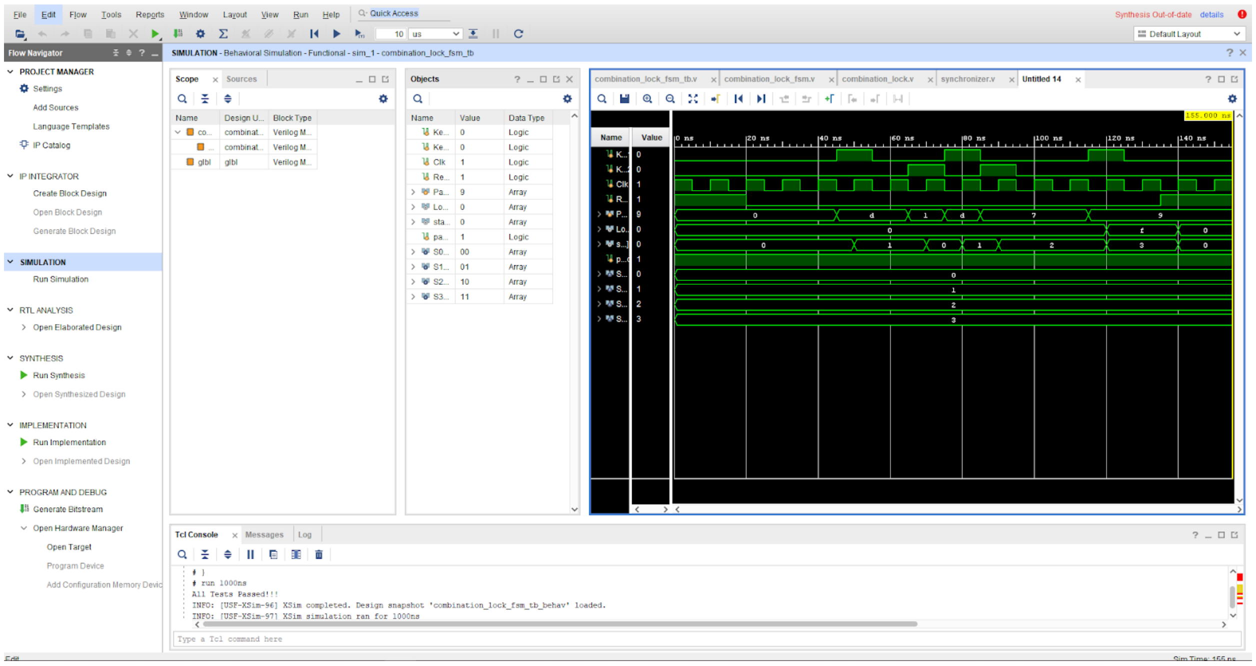
```verilog
`timescale 1ns / 1ps
`default_nettype none
//528000252

module combination_lock_fsm(state, Lock, Key1, Key2, Password, Reset, Clk);
   output reg [2:0] state;
   output wire [3:0] Lock; //asserted when locked
   input wire Key1; //unlock button 0
   input wire Key2; //unlock button 1
   input wire [3:0] Password; //indicate number
   input wire Reset; //reset
   input wire Clk; //clock


   //intermediate nets
   reg[2:0] nextState;

   parameter S0 = 3'b000;
   parameter S1 = 3'b001;
   parameter S2 = 3'b010;
   parameter S3 = 3'b011;
   parameter S4 = 3'b100;
   always@(*)
      case(state)
         S0: begin
            if(Key1==1 && Password == 4'b1101)
               nextState = S1;
            else
               nextState = S0;
         end
         S1: begin
            if(Key2==1 && Password == 4'b0111)
               nextState = S2;
            else if(Key2==1 && Password != 4'b0111)
               nextState = S0;
            else
               nextState = S1;
         end

         S2: begin
            if(Key1 == 1 && Password == 4'b1001)
                nextState = S3;
             else if(Key2 == 1 && Password != 4'b1001)
                nextState = S0;
```


```verilog
            if(Key2==1 && Password == 4'b0111)
               nextState = S2;
            else if(Key2==1 && Password != 4'b0111)
               nextState = S0;
            else
               nextState = S1;
         end

         S2: begin
            if(Key1 == 1 && Password == 4'b1001)
                nextState = S3;
             else if(Key2 == 1 && Password != 4'b1001)
                nextState = S0;
             else
                nextState = S2;
               end

             S3: begin
                if(Key1 == 1 && Password == 4'b1111)
                   nextState = S4;
                else if(Key2 == 1 && Password != 4'b1111)
                   nextState = S0;
                else
                   nextState =S3;
                end

   S4: begin
   nextState = S4;
   end

   endcase

   always@(posedge Clk)
   if(Reset)
   state <= S0;
   else
   state <= nextState;

   assign Lock = (state == S3) ? 4'b1111: 4'b0000;

endmodule
```

//Combination lock

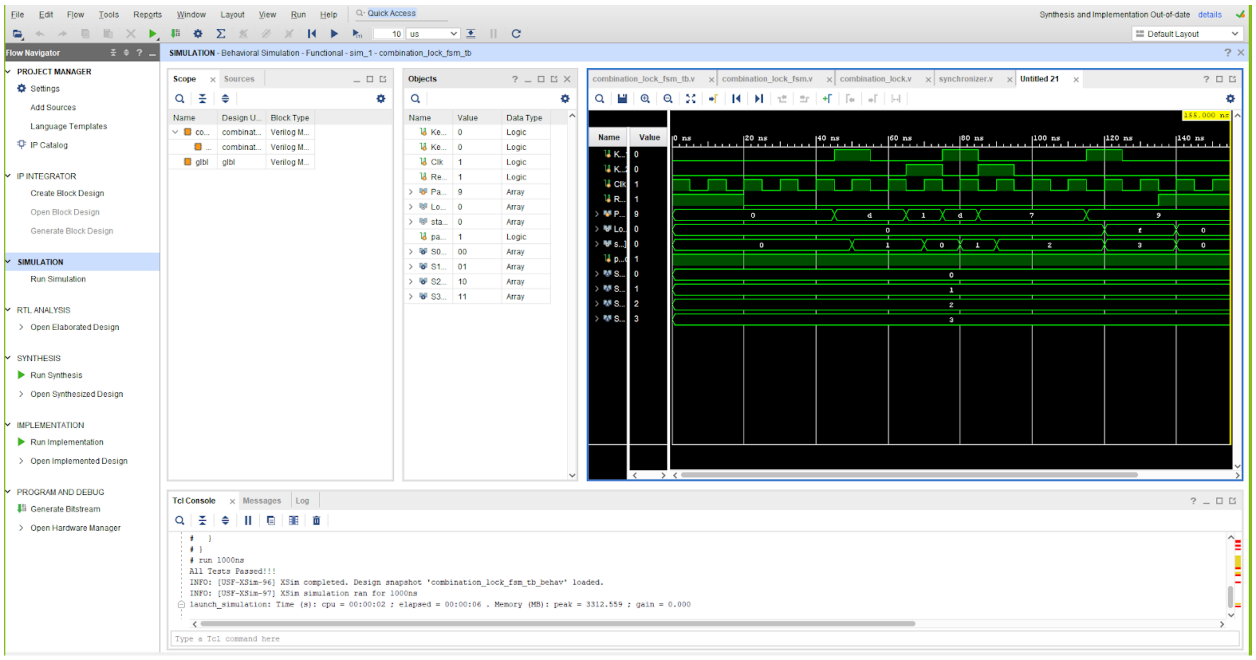
```verilog
module combination_lock(LEDs, JB, Clk, Reset, Key1, Key2, Password);
    output wire [3:0] LEDs;//connect to lock
    /*Let's output state for debugging!*/
    output wire [2:0] JB;
    input wire  Clk;
    input wire  Key1,Key2;
    input wire  Reset;
    input wire  [3:0]Password;


    /*intermediate nets*/
    wire KeySync1,KeySync2, ResetSync;


    /*synchronize button inputs*/
    synchronizer syncA(KeySync1, Key1, Clk);
    synchronizer syncB(KeySync2, Key2, Clk);
    synchronizer syncC(ResetSync, Reset, Clk);

    /*wire up combination lock FSM*/
    combination_lock_fsm U1(
        .Lock(LEDs),
        .state(JB),
        .Clk(Clk),
        .Key1(KeySync1),
        .Key2(KeySync2),
        .Reset(ResetSync),
        .Password(Password)
    );


endmodule
```

//Synchronizer
Given to the students
<u>Result</u>
//Combination lock FSM (3 States)


SIMULATION - Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb
Flow Navigator
Scope
Objects
Untitled 14
Tcl Console

```
# }
# run 1000ns
All Tests Passed!!!
INFO: [USF-XSim-96] XSim completed. Design snapshot 'combination_lock_fsm_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

//Combination lock FSM (4 States)


SIMULATION - Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb
Flow Navigator
Scope
Objects
Untitled 21
Tcl Console

```
#   }
# }
# run 1000ns
All Tests Passed!!!
INFO: [USF-XSim-96] XSim completed. Design snapshot 'combination_lock_fsm_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 3312.559 ; gain = 0.000
```

Conclusion

The code on verilog worked as it should have and the combination lock on the FPGA displayed properly. The issue I faced during the lab was minor syntax errors which didn't catch my eye in the beginning and not saving changes, so always had to reminplent it again into the modules. I learned gained an understanding of the mechanics of a lock virtually using Verilog and using finite state machines, and how its executed on the zybo.

Post - Lab Questions

1. Include the source code with comments for all modules you simulated and/or implemented in lab. You do not have to include test bench code that was provided! Code without comments will not be accepted! If the last 4 digits of your UIN cannot be found in your verilog code, you will receive any point for that design.

(3-password combination lock code and 4-password combination lock code)

Screenshots provided in the design sections

2. Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation. Do not edit your screenshots.

(3-password combination lock simulation waveform)

Screenshots provided in the result sections

3. Take a look at the simulation waveform of your FSM and take note of the tests that the test bench performs. Is this an exhaustive test? Why or why not?

Even though it doesn't go through all combinations, it can be considered an exhaustive test. Within the test bench field it uses a system, where it tests certain combinations that would show whether other ones would fail or not.

4. A possible attack on your combination-lock is a brute-force attack in which every possible input combination is tried. Given the original design with a combination of three numbers between 0 and 15, how many possible input combinations exist? How about for the modified design with a combination of four numbers?

The total possible combination for values from 0 to 15 is $16^3 = 4096$ and for the four password combination, the total number of combinations is $16^4 = 65536$ combinations. These are a lot of possible combinations and it would take a long time.

Feedback

1. What did you like most about the lab assignment and why? What did you like least about it and why?

My favorite part of this lab was implementing the design on zybo, however my least favorite part was when the combination lock code was given, I think with instructions writing it would give more understanding.

2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving clarity?
I think the lab manual was pretty clear and to the point.
3. What suggestions do you have to improve the overall lab assignment?
I would change the demo instructions, making it more clear.