Lab 11: The Traffic Light Controller

ECEN 248 - 505

TA: Younggyun Cho

Date: November 17, 2020

Objectives

The goal of this lab was to learn how to set up the state diagram for a mealy machine, using verilog for different states and actions of a Traffic Light Controller. The student has to implement the guidelines for the design. The student creates files for tlc_fsm.v, tlc_controller_ver1.v, and use synchronizer.v, tlc_controller.xdc from given files, to load to the zybo. The same steps are repeated but with sensors.

Design

TLC FSM code in Experiment 1

```
//tlc fsm without sensor
//528000252
`timescale 1ns/1ps
`default_nettype none

`define one_sec 50000000
`define three_sec 150000000
`define thirty_sec 1500000000
`define fifteen_sec 750000000

module tlc_fsm(
        output reg [2:0] state,     //output for debugging
        output reg RstCount,        //use an always block
        output reg [1:0] highwaySignal, farmSignal,
        input wire [30:0] Count,//use n computed earlier
        input wire Clk, Rst         //clock and reset
);

        //defining states
        parameter       Srst = 3'b110,
        S0 = 3'b000,
        S1 = 3'b001,
        S2 = 3'b010,
        S3 = 3'b011,
        S4 = 3'b100,
        S5 = 3'b101;

        //defining colors
        parameter       green = 2'b00,
        yellow = 2'b01,
        red     = 2'b10;

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

```
//next state logic
always@(*)
case(state)

Srst: nextState = S0;
S0: begin
if(Count >= `one_sec)    //if count reached
            nextState = S1;  //transition
else //otherwise
            nextState = S0;  //remain in current state
end
S1: begin
if(Count >= `thirty_sec) //if count reached
            nextState = S2;  //transition
else //otherwise
            nextState = S1;  //remain in current state
end

S2: begin
if(Count >= `three_sec)  //if count is reached
            nextState = S3;  //transition
else
            nextState = S2;  //remain in current state
end
S3: begin
if(Count >= `one_sec)    //if count is reached
            nextState = S4;  //transition
else
            nextState = S3;
end
S4: begin
if(Count >= `fifteen_sec)            //if count is reached
            nextState = S5;  //transition
else
            nextState = S4;  //remain in current state
end
S5: begin
if(Count >= `three_sec)  //if count reached
            nextState = S0;  //transition
else        //otherwise
            nextState = S5;  //remain in current state
end

default://avoid latches
```

```
nextState = Srst;
endcase

/*describe output logic*/
always@(state or Count)
case(state)
Srst : begin
highwaySignal = red;
farmSignal = red;
RstCount = 1;
end
S0: begin
highwaySignal = red;
farmSignal = red;
if(Count >= `one_sec)    //if count reached
          RstCount = 1;    //reset counter
else      //otherwise
          RstCount = 0;    //let counter run
end
S1: begin
highwaySignal = green;
farmSignal = red;
if(Count >= `thirty_sec) //if count reached
          RstCount = 1;    //reset counter
else      //otherwise
          RstCount = 0;    //let counter run
end
S2: begin
highwaySignal = yellow;
farmSignal = red;
if(Count >= `three_sec)  //if count reached
          RstCount = 1;    //reset counter
else      //otherwise
          RstCount = 0;    //let counter run
end
S3: begin
highwaySignal = red;
farmSignal = red;
if(Count >= `one_sec)    //if count reached
          RstCount = 1;    //reset counter
else //otherwise
          RstCount = 0;    //let counter run
end
S4: begin
```

```
        highwaySignal = red;
        farmSignal = green;
        if(Count >= `fifteen_sec)           //if count reached
                RstCount = 1;   //reset counter
        else    //otherwise
                RstCount = 0;   //let counter run
        end
        S5: begin
        highwaySignal = red;
        farmSignal = yellow;
        if(Count >= `three_sec) //if count reached
                RstCount = 1;   //reset counter
        else    //otherwise
                RstCount = 0;   //let counter run
        end
        default: begin   //avoid latches! highwaySignal = red;
        farmSignal = red;
        RstCount = 1;
        end
        endcase

        //the clock for the system
        always@(posedge Clk)
        if(Rst)
        state <= Srst;
        else
        state <= nextState;
endmodule
```

TLC FSM code in Experiment 2

```
//tlc with sensor
//528000252
`timescale 1ns/1ps
`default_nettype none

`define one_sec 50000000
`define three_sec 150000000
`define thirty_sec 1500000000
`define fifteen_sec 750000000

module tlc_fsm(
        output reg [2:0] state,     //output for debugging
        output reg RstCount,        //use an always block
```

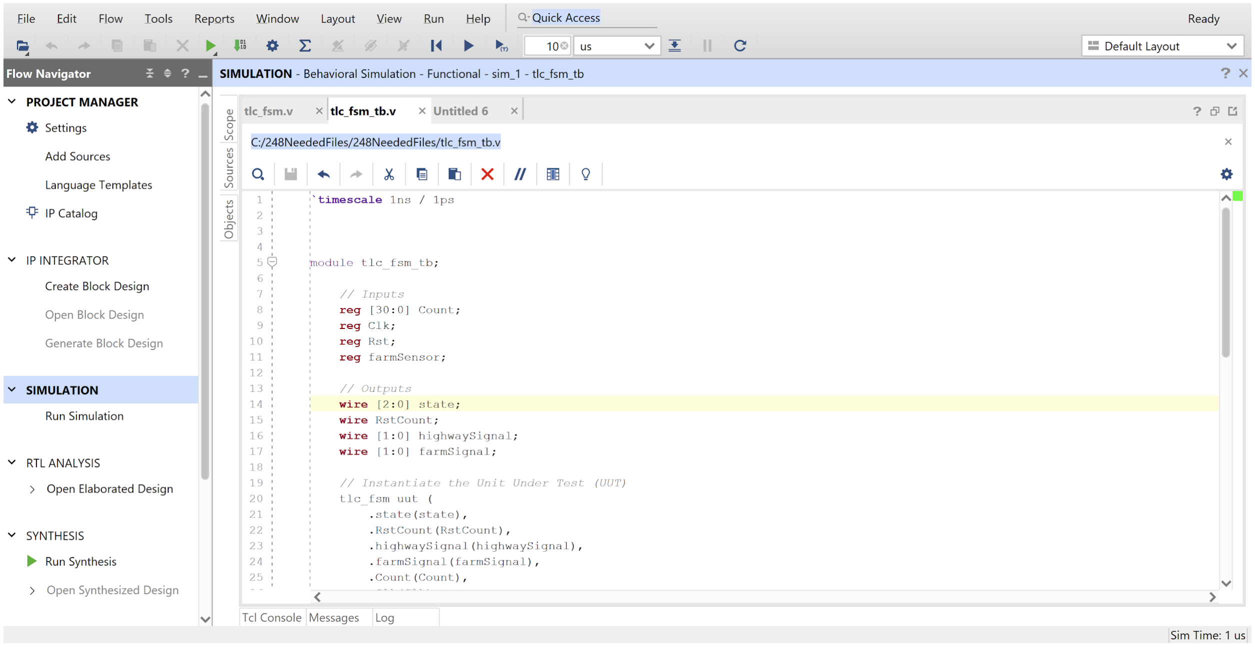
```
    output reg [1:0] highwaySignal, farmSignal,
    input wire [30:0] Count,//use n computed earlier
    input wire Clk, Rst,    //clock and reset
    input wire farmSensor
);

    //defining states
    parameter       Srst = 3'b110,
    S0 = 3'b000,
    S1 = 3'b001,
    S2 = 3'b010,
    S3 = 3'b011,
    S4 = 3'b100,
    S5 = 3'b101;

    //defining colors
    parameter       green = 2'b00,
    yellow = 2'b01,
    red     = 2'b10;

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

    //next state logic
    always@(*)
    case(state)

    Srst: nextState = S0;
    S0: begin
    if(Count >= `one_sec)   //if count reached
        nextState = S1;  //transition
    else //otherwise
        nextState = S0;  //remain in current state
    end
    S1: begin
    if(Count >= `thirty_sec && farmSensor==1)       //if count reached
        nextState = S4;  //transition
    else //otherwise
        nextState = S1;  //remain in current state
    end

    S2: begin
    if(Count >= `three_sec) //if count is reached
        nextState = S3;  //transition
```

```
else
        nextState = S2;  //remain in current state
end
S3: begin
if((Count >= `three_sec && farmSensor==0)|| Count >= 15)      //if count is reached
        nextState = S1;  //transition
else
        nextState = S3;
end
S4: begin
if(Count >= `fifteen_sec)          //if count is reached
        nextState = S5;  //transition
else
        nextState = S4;  //remain in current state
end
S5: begin
if(Count >= `three_sec)  //if count reached
        nextState = S0;  //transition
else    //otherwise
        nextState = S5;  //remain in current state
end

default://avoid latches
nextState = Srst;
endcase

/*describe output logic*/
always@(state or Count)
case(state)
Srst : begin
highwaySignal = red;
farmSignal = red;
RstCount = 1;
end
S0: begin
highwaySignal = red;
farmSignal = red;
if(Count >= `one_sec)    //if count reached
        RstCount = 1;    //reset counter
else    //otherwise
        RstCount = 0;    //let counter run
end
S1: begin
highwaySignal = green;
```

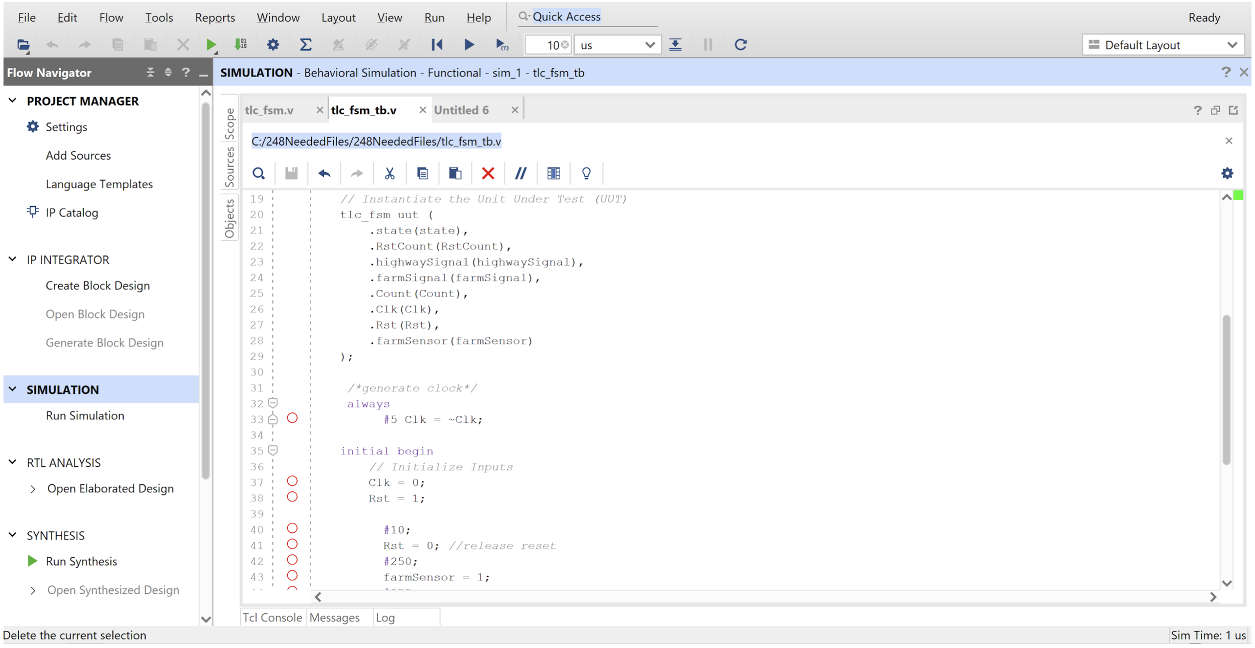
```
farmSignal = red;
if(Count >= `thirty_sec) //if count reached
        RstCount = 1;   //reset counter
else    //otherwise
        RstCount = 0;   //let counter run
end
S2: begin
highwaySignal = yellow;
farmSignal = red;
if(Count >= `three_sec) //if count reached
        RstCount = 1;   //reset counter
else    //otherwise
        RstCount = 0;   //let counter run
end
S3: begin
highwaySignal = red;
farmSignal = red;
if(Count >= `one_sec)   //if count reached
        RstCount = 1;   //reset counter
else //otherwise
        RstCount = 0;   //let counter run
end
S4: begin
highwaySignal = red;
farmSignal = green;
if(Count >= `fifteen_sec)       //if count reached
        RstCount = 1;   //reset counter
else    //otherwise
        RstCount = 0;   //let counter run
end
S5: begin
highwaySignal = red;
farmSignal = yellow;
if(Count >= `three_sec) //if count reached
        RstCount = 1;   //reset counter
else    //otherwise
        RstCount = 0;   //let counter run
end
default: begin   //avoid latches! highwaySignal = red;
farmSignal = red;
RstCount = 1;
end
endcase
```

```
    //the clock for the system
    always@(posedge Clk)
    if(Rst)
    state <= Srst;
    else
    state <= nextState;
endmodule
```

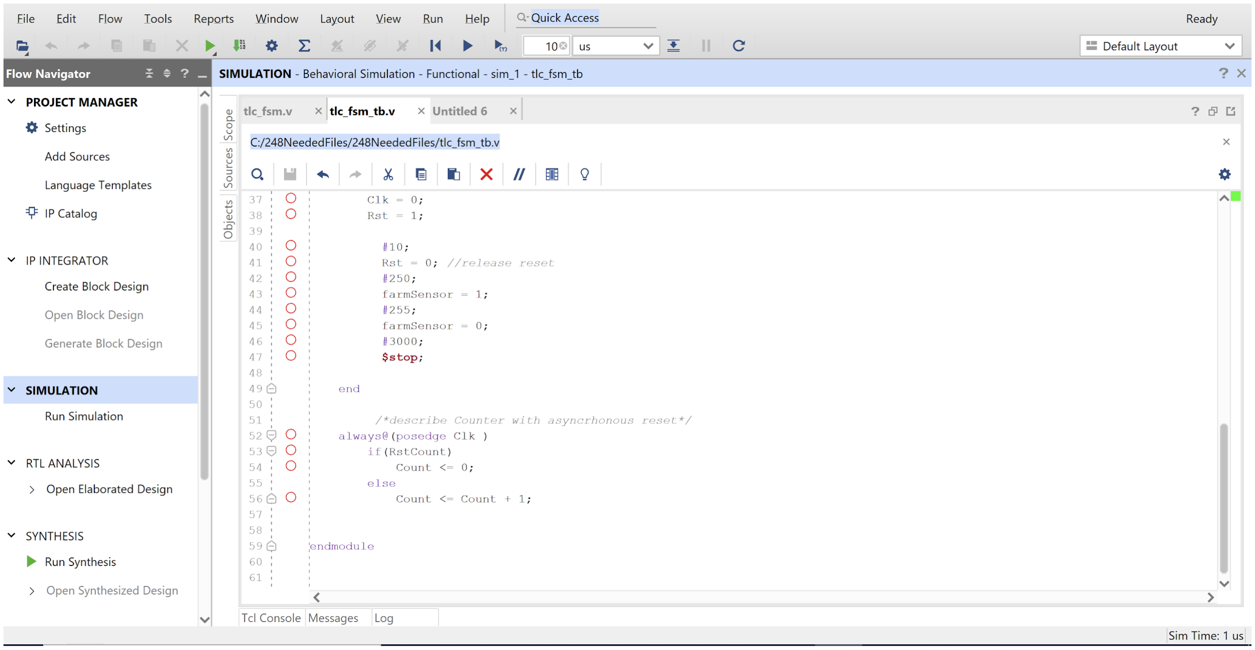
TLC FSM test bench code in Experiment 12



```
`timescale 1ns / 1ps



module tlc_fsm_tb;

    // Inputs
    reg [30:0] Count;
    reg Clk;
    reg Rst;
    reg farmSensor;

    // Outputs
    wire [2:0] state;
    wire RstCount;
    wire [1:0] highwaySignal;
    wire [1:0] farmSignal;

    // Instantiate the Unit Under Test (UUT)
    tlc_fsm uut (
        .state(state),
        .RstCount(RstCount),
        .highwaySignal(highwaySignal),
        .farmSignal(farmSignal),
        .Count(Count),
```



```
    // Instantiate the Unit Under Test (UUT)
    tlc_fsm uut (
        .state(state),
        .RstCount(RstCount),
        .highwaySignal(highwaySignal),
        .farmSignal(farmSignal),
        .Count(Count),
        .Clk(Clk),
        .Rst(Rst),
        .farmSensor(farmSensor)
    );

     /*generate clock*/
     always
          #5 Clk = ~Clk;

    initial begin
        // Initialize Inputs
        Clk = 0;
        Rst = 1;

          #10;
          Rst = 0; //release reset
          #250;
          farmSensor = 1;
```



C:/248NeededFiles/248NeededFiles/tlc_fsm_tb.v

```
        Clk = 0;
        Rst = 1;

          #10;
          Rst = 0; //release reset
          #250;
          farmSensor = 1;
          #255;
          farmSensor = 0;
          #3000;
          $stop;

    end

         /*describe Counter with asyncrhonous reset*/
    always@(posedge Clk )
        if(RstCount)
            Count <= 0;
        else
            Count <= Count + 1;


endmodule
```

Result

State	Highway Output	Farm Road Output	Delay (s)	Delay (cc)
S0	red	red	1	50,000,000
S1	green	red	30	1,500,000,000
S2	yellow	red	3	150,000,000
S3	red	red	1	50,000,000
S4	red	green	15	750,000,000
S5	red	yellow	3	150,000,000

Conclusion

For the lab I utilized Verilogn in order to implement the design to execute a Traffic Light Controller that switched states and these outputs were based on two different traffic lights in perpendicular roads. For the second experiment, I created a design using sensor signals to indicate if theres any cars on the road. This taught me more about mealy machines and how these simple designs and concepts are implenemtn in real life situations.

Post - Lab Questions

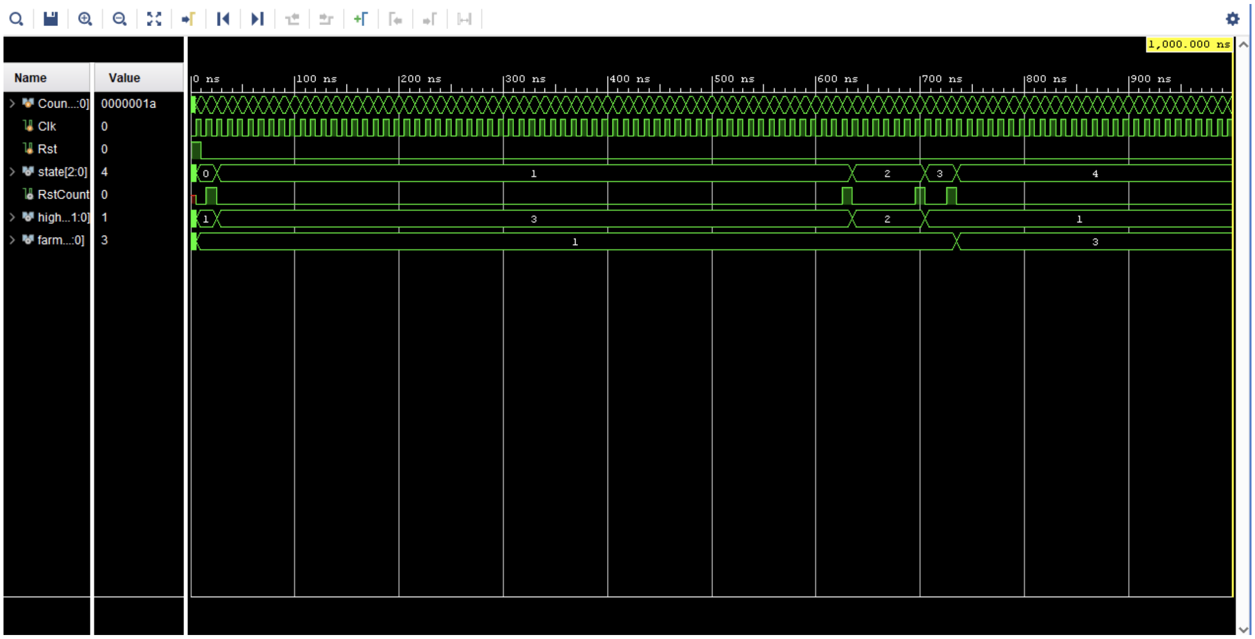
1. Include the source code with comments for all modules that you wrote or modified in the lab. You do not need to include the code that was provided! Remember that code without comments will not be accepted!

In the design section.

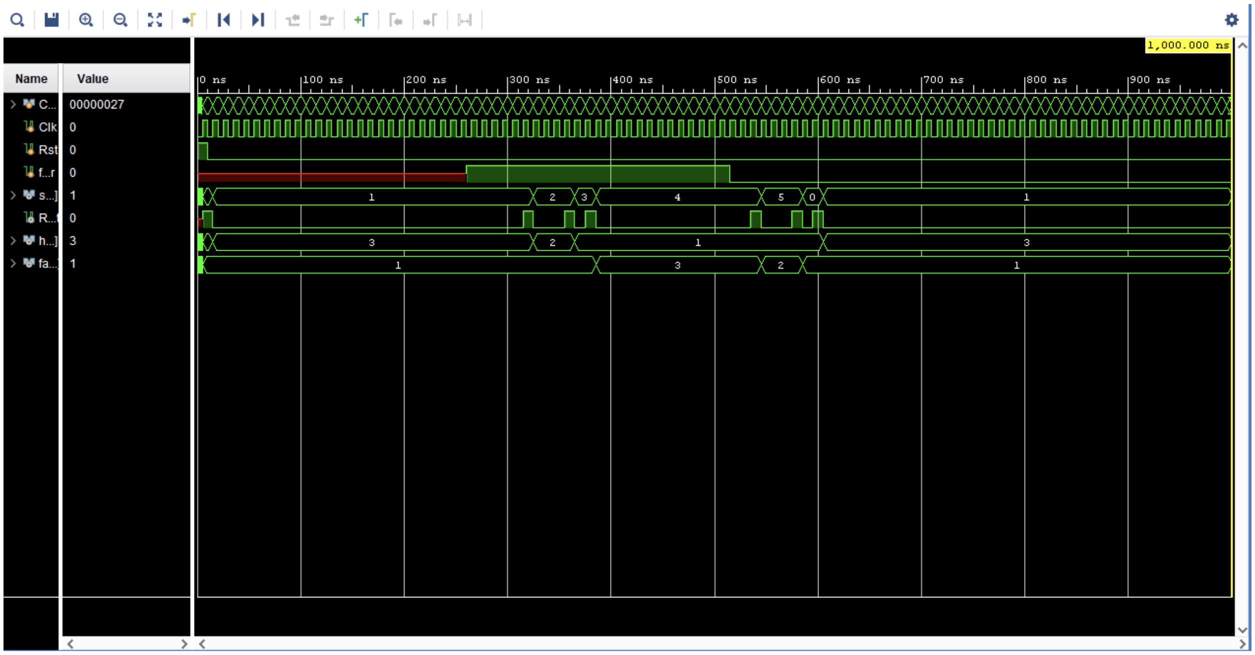
2. Include screenshots of all waveforms captured during the simulation. You can put marks on the waveforms to further explain the behaviors.

The waveform in Experiment 1.

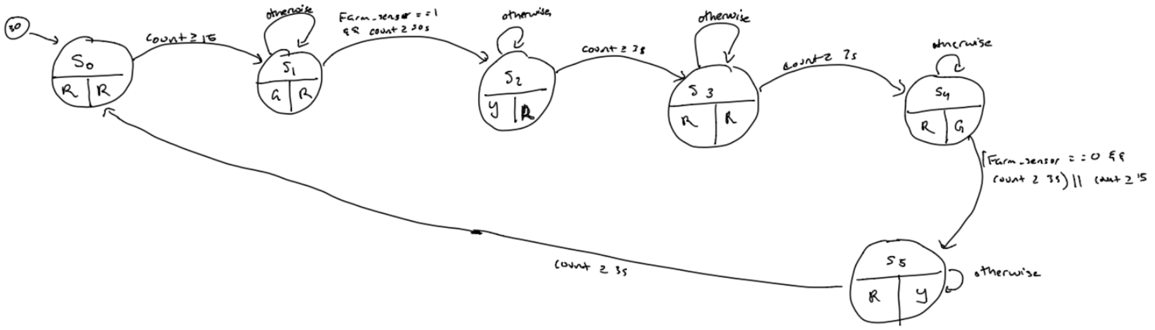
Without sensor



With sensor



3. Include the state diagram for the modified traffic light controller FSM.



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 I got confused with adding sensors to my tlc_fsm code.

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 instructions to help with the coding.