ECEN 248: Introduction to Digital Design
Department of Electrical and Computer Engineering
Texas A&M University
Lab Report

Laboratory Exercise #11

The Traffic Light Controller Lab

ECEN-248-509

Joaquin Salas

731000141

12-4-2022

TA: Sri Hari Pada Chandanam Kodi

Objectives:

In this week's lab, we continue our introduction to FSMs with a Traffic Light Controller design. In the pre-lab, we had the opportunity to create a state diagram of a Mealy Machine. Its purpose is to implement a traffic light controller based on the given guidelines. We will also use the state diagram to write a behavioral Verilog description of the traffic light controller. The testing of the traffic light control will take place in the lab session.

Lab Procedures:

Experiment Part 1:

In the first part of this experiment, we synthesized the traffic light controller using the Finite State machine. The FSM code we used is the one we created in the prelab. Once we moved the code into Vivado and debugged it, we synchronized and tested it with the testbench titled 'tlc_controller.xdc.' After debugging more we could finally upload the code into the FPGA board and test if our code works or not.

Code: *Prelab, Traffic Light Controller FSM.*

```
`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
    /*another always block for these as well*/
    output reg [1:0] highwaySignal, farmSignal;
    input wire [30:0] Count, //use n(31) computed earlier
    input wire Clk, Rst //clock and reset
    );

    //defining states
    parameter s0 = 3'b000;
    parameter s1 = 3'b001;
```

```
    parameter s2 = 3'b010;
    parameter s3 = 3'b011;
    parameter s4 = 3'b100;
    parameter s5 = 3'b101;

    /*intermediate nets*/
    reg [2:0] nextState;
    reg [1:0] state;

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

    /*describing next state logic*/
    always@(state or Count)
        case(state)
            //Srst: nextState = s0;
            s0: begin
                if(Count == one_sec) //if count reached
                    nextState = s1; //transition
                else //otherwise remain in current state
                    nextState = s0;
            end

            s1: begin
                if(Count == thirty_sec) //if this count is reached
                    nextState = s2; //transition
                else //remain in current state
            end

            s2: begin
                if(Count == three_sec) //if count reached
                    nextState = s3; //transition
                else //otherwise
                    nextState = s2; //remain in current state
            end

            s3: begin
                if(Count == one_sec) //if count reached
                    nextState = s4; //transition
                else //otherwise
                    nextState = s3; //remain in current state
            end

            s4: begin
                if(Count == fifteen_sec) //if count reached
                    nextState = s5; //transition
```

```
            else //otherwise
                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 = s0;
    endcase

/*describe output logic*/
always@(state or Count)
    case(state)
        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
                highwaySignal = red;
                farmSignal = red;
                RstCount = 1;
            end
        endcase

    /*behavior for input clock*/

    always@(posedge Clk)
        case(state)
            if(Rst)
                state <= s0;
            else
                state <= nextState;
        endcase
endmodule
```

Code: *Traffic Light Controller.*

```
`timescale 1ns / 1ps

module tlc_controller_ver1( //for the inputs and outputs
    output wire [1:0] highwaySignal, farmSignal,
    output wire [3:0] JB,
    input wire Clk,
    input wire Rst,
    input wire farmSensor
);
    wire RstSync; //set wires for reseting syncs and counts
    wire RstCount;
    reg [30:0] Count; //register
    wire sensorSync;

    assign JB[3] = RstCount;

    synchronizer syncRst(RstSync, Rst, Clk);
    synchronizer syncSensor(sensorSync, farmSensor, Clk);

    tlc_fsm FSM( //controller logic
        .state(JB[2:0]),
        .RstCount(RstCount),
        .highwaySignal(highwaySignal),
        .farmSignal(farmSignal),
        .Count(Count),
        .Clk(Clk),
        .farmSensor(sensorSync)
    );

        always@(posedge Clk)
        if(RstCount)
            Count <= 0;
        else
            Count <= Count + 1;
endmodule
```

Experiment Part 2:

The second part of the experiment included us implementing a sensor that detects when a car is on the farm road. In order to achieve this, there needs to be another module that specifies the different wait times for the traffic light when a car does come to it. Once this code was completed and debugged, we synthesized it again and implemented it onto the Zybo Z7-10 board.

Code: *Traffic Light Controller with Farm road Car.*

```
`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
    /*another always block for these as well*/
    output reg [1:0] highwaySignal, farmSignal,
    input wire [30:0] Count, //use n(31) computed earlier
    input wire Clk, Rst, //clock and reset
    input wire farmSensor
    );

    //defining states
    parameter s0 = 3'b000;
    parameter s1 = 3'b001;
    parameter s2 = 3'b010;
    parameter s3 = 3'b011;
    parameter s4 = 3'b100;
    parameter s5 = 3'b101;

    /*intermediate nets*/
    reg [1:0] nextState;
    reg [1:0] state;

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

    /*describing next state logic*/
    always@(state or Count)
```

```
        case(state)
            //Srst: nextState = s0;
            s0: begin //with the follow
                if((Count == `one_sec)) //if count reached
                begin
                    highwaySignal = red;
                    farmSignal = red;
                    RstCount = 1;
                    nextState = s1; //transition
                end
                else //otherwise remain in current state
                begin
                    highwaySignal = red;
                    farmSignal = red;
                    RstCount = 0;
                    nextState = s0;
                end
            end

            s1: begin
                if((Count == `thirty_sec) && (farmSensor == 1)) //if this count is
reached
                begin
                    highwaySignal = green;
                    farmSignal = red;
                    RstCount = 1;
                    nextState = s2; //transition
                end
                else //remain in current state
                begin
                    highwaySignal = green;
                    farmSignal = red;
                    RstCount = 0;
                    nextState = s1;
                end
            end

            s2: begin
                if((Count == `three_sec)) //if count reached
                begin
                    highwaySignal = yellow;
                    farmSignal = red;
                    RstCount = 1;
                    nextState = s3; //transition
                end
                else //otherwise
                begin
                    highwaySignal = yellow;
```

```
                    farmSignal = red;
                    RstCount = 0;
                    nextState = s2; //remain in current state
                end
            end

            s3: begin
                if((Count == `one_sec)) //if count reached
                begin
                    highwaySignal = red;
                    farmSignal = red;
                    RstCount = 1;
                    nextState = s4; //transition
                end
                else //otherwise
                begin
                    highwaySignal = red;
                    farmSignal = red;
                    RstCount = 0;
                    nextState = s3; //remain in current state
                end
            end

            s4: begin
                if((Count == `fifteen_sec) || (farmSensor == 0 && Count ==
`three_sec)) //if count reached
                begin
                    highwaySignal = red;
                    farmSignal = green;
                    RstCount = 1;
                    nextState = s5; //transition
                end
                else //otherwise
                begin
                    highwaySignal = red;
                    farmSignal = green;
                    RstCount = 0;
                    nextState = s4; //remain in current state
                end
            end

            s5: begin
                if((Count == `three_sec)) //if count reached
                begin
                    highwaySignal = red;
                    farmSignal = yellow;
                    RstCount = 1;
                    nextState = s0; //transition
```

```
                end
                else //otherwise
                begin
                    highwaySignal = red;
                    farmSignal = yellow;
                    RstCount = 0;
                    nextState = s5; //remain in current state
                end
            end

            //default: //avoid latches
                //nextState = s0;
        endcase

    always@(posedge Clk)
        if(Rst)
            state <= s0;
        else
            state <= nextState;

endmodule
```

Results:

This experiment resulted in multiple waveforms and codes which are labeled throughout the lab report appropriately.

Conclusion:

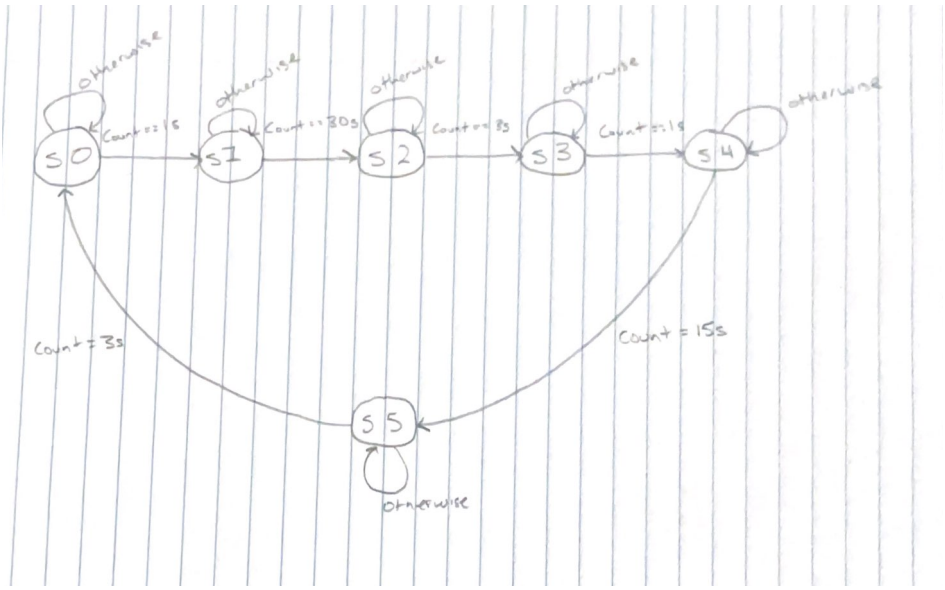
This lab gave us a better understanding of a Mealy Finite State machine by using behavioral Verilog and giving us a real-world example such as a traffic light controller with real-world situations.

Post-lab Deliverables:

1. Include the source code with comments for all modules 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!

All the code with comments is provided above.

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



Important Student Feedback

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

I enjoyed learning the difference between the three types of ways that code is written in Verilog, structural, dataflow, and behavioral, the pre-lab did take me a fair amount of time but I believe it was much worth the work. I did not like how the Zybo-10 board did not work with our computer at the end of the lab.

2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?

All parts of the lab manual were very clear, except for the last part on experiment 3 where there was a crucial step that was left out to get the Zybo board to work correctly.

3. What suggestions do you have to improve the overall lab assignment?

I have no suggestions for improving the overall lab assignment.