BERZIET UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENCS3310

Traffic light controller

Project report

Prepared by:

Omar Daraghmeh

1200162

Supervised by:

Dr. Abdellatif Abu-Issa

Table of Contents

Table of Figures:

Brief introduction and background:

In this this project we want to design a Finite State Machine (FSM) based traffic light controller for the highway and farm road intersection as shown in the figure 1 below, using the Verilog hardware description language. The FSM will have 3 inputs: "Rst" for resetting the design to state 0 and counter to 0, "Go" for controlling the state transitions, and "clk" for synchronizing the system. The FSM will have 4 output signals, highway Signal 1 and highway signal 2 and farm Signal 1 and 2, which will control the traffic lights on the highway and farm road, the design will be as in the figure 2. The design will be implemented in Verilog and will be verified using a simple test bench that will implement tests for all states of the FSM.

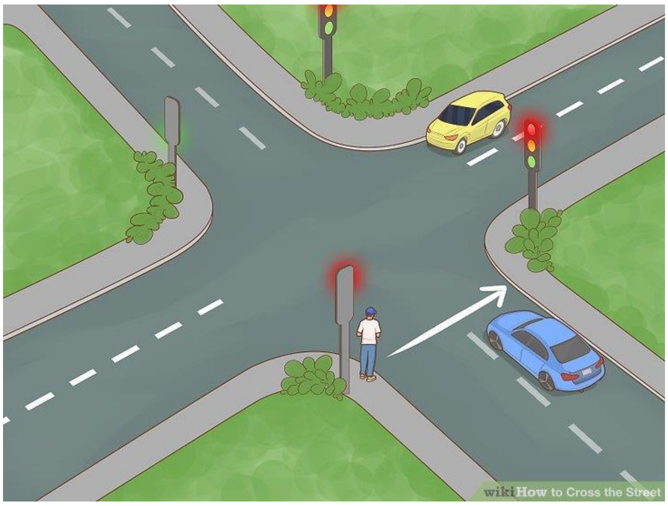

Figure 1:road intersection

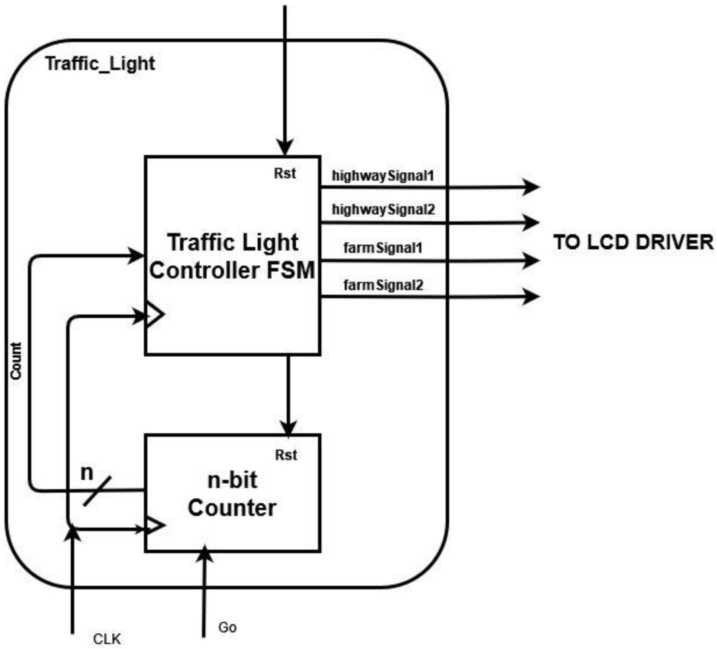

Figure 2:design

Design philosophy:

First we have three inputs: clk, reset, go. And the output as a 4 outputs with two bits inside each as an 2d array (4x2). The parameters the refer for the output as in RED=00, etc. another parameters to refer the state as in S0 to S17. And we have a register for the counter and for the state with initial 0 for the count and S0 for the state.

```
module traffic_light(clk  ,reset,go, out);

    input clk,reset,go;
    output reg [1:0]out [3:0];
    parameter RED=2'b00,RED_YELLOW=2'b01,YELLOW=2'b10,GREEN=2'b11;
    parameter S0=5'd0,S1=5'd1,S2=5'd2,S3=5'd3,S4=5'd4
    ,S5=5'd5,S6=5'd6,S7=5'd7,S8=5'd8,S9=5'd9,S10=5'd10,S11=5'd11
    ,S12=5'd12,S13=5'd13,S14=5'd14,S15=5'd15,S16=5'd16,S17=5'd17 ;

    reg [4:0]count=0;
    reg [4:0]state=S0;
```

Figure 3:I\O implementation

When the clk changes or there is a change in the reset input. If the reset is =1 meaning it will start from the beginning by making the state =S0 again and the count =0 . if reset =0 we check for the go button if =0 nothing will happen (freeze) but if =1 it will wait for each state after finishes its time to move for the next one, for example if the state is S2 it will wait till count reaches 30 then it will make the state = S3 and put the counter 0 to start counting again. It will do this for all cases from S0 to S17 and the next state for S17 is S0.

```
always@(posedge clk,posedge reset)
    begin
         if (reset==1)
            begin
                count<=0;
                state<=S0;
            end
        else
            begin
                    if(go)begin
                    count<=count+1;end

                case(state)

                S0:if(count==1)
                    begin
                        state<=S1;
                        count<=0;
                    end
                S1:if(count==2)
                    begin
                        state<=S2;
                        count<=0;
                    end
                S2:if(count==30)
                    begin
                        state<=S3;
                        count<=0;
                    end
                S3:if(count==2)
                    begin
                        state<=S4;
                        count<=0;
                    end
```

Figure 4:states

Then we start to put the values in the output depending the state we are in. we initialize output to have {red,red,red,red} as first value and start changing when the states changes for example if the state changes to S2 then the output will be {green,green,red,red} .it will implement for the 17 states.

```
initial begin
        out[3]=RED;
        out[2]=RED;
        out[1]=RED;
        out[0]=RED;
end

always @(state)begin
    case(state)
        S0:begin
            out[3]=RED;
            out[2]=RED;
            out[1]=RED;
            out[0]=RED;
        end
        S1:begin
            out[3]=RED_YELLOW;
            out[2]=RED_YELLOW;
            out[1]=RED;
            out[0]=RED;
        end
        S2:begin
            out[3]=GREEN;
            out[2]=GREEN;
            out[1]=RED;
            out[0]=RED;
        end
        S3:begin
            out[3]=GREEN;
            out[2]=YELLOW;
            out[1]=RED;
            out[0]=RED;
        end
        S4:begin
```

Figure 5:The output values

Results:

Test bench:

We can see the results in the simulation for the test bench. The results were as expected to be (all cases happened in their times).

If we initialize the reset button on 0 and the go button(enable) on 1 and make the clk change every time unit as in the figure below the system should be running.

```
module traffic_lighttb;
    reg clk,reset,enable;
    wire [1:0]out[3:0];
    traffic_light test(clk,reset,enable,out);
    initial begin
        clk=0;
        reset=0;
        enable=1;
        #1000 $finish;
    end
    always #1 clk=~clk;


endmodule
```

Figure 6: test bench

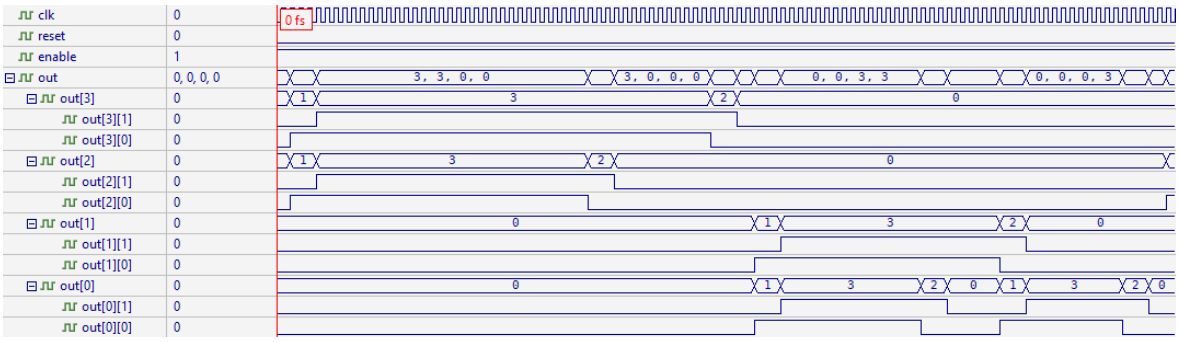

Figure 7: simulation

If we want to check for the reset if it works. We make the reset changes to 1 after a random time units the outputs should return to S0 state(00,00,00,00)

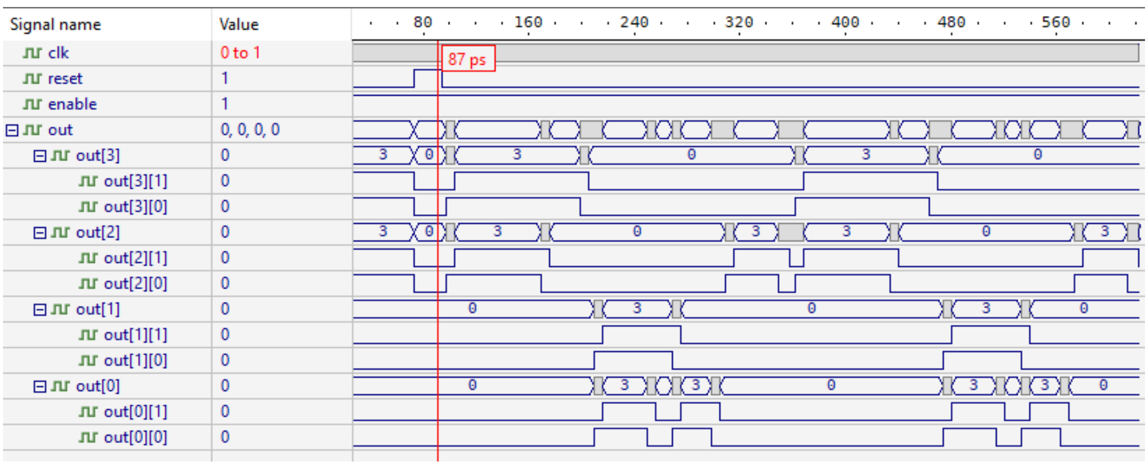

Figure 8: simulation (reset)

If we want to check for the go button (enable) we put it to zero in a random time and until it changes back it will freeze in the state it's in.

We try to make go =0 in start and then make it again zero to see if it will works properly.

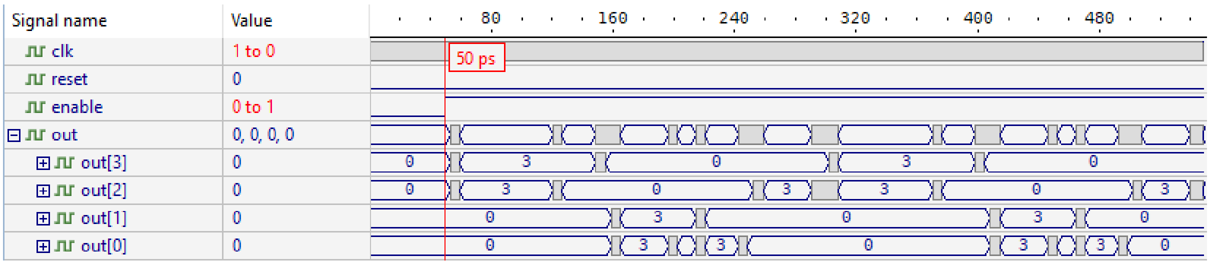

Figure 9: simulation (go)

As expected it freezes for 50 times units.

Verification:

We make a model that have the correct values as in the figure below.

```
module model(state,out);
    input [4:0]state;
    output reg [1:0]out[3:0];
    reg [7:0]memory[17:0];
    initial begin
        memory[0]=8'b00000000;
        memory[1]=8'b01010000;
        memory[2]=8'b11110000;
        memory[3]=8'b11100000;
        memory[4]=8'b11000000;
        memory[5]=8'b10000000;
        memory[6]=8'b00000000;
        memory[7]=8'b00000101;
        memory[8]=8'b00001111;
        memory[9]=8'b00001110;
        memory[10]=8'b00001100;
        memory[11]=8'b00001001;
        memory[12]=8'b00000011;
        memory[13]=8'b00000010;
        memory[14]=8'b00000000;
        memory[15]=8'b00010000;
        memory[16]=8'b00110000;
        memory[17]=8'b00100000;
    end
    always@ ( state)
        begin
            out[3]=memory[state][7:6];
            out[2]=memory[state][5:4];
            out[1]=memory[state][3:2];
            out[0]=memory[state][1:0];
        end

endmodule
```

Figure 10:correct model

Then we make an analyzer to compare between the results from the model and from our system by passing a state to the model and the state is generated by adding one so it will go through the memory in the model by order. The analyzer compare between the two outputs if they are not equal will display a message that there is an error

```
module analyzer;
    reg clk,reset,enable;
    reg [4:0]state=5'd17;
    wire [1:0]out[3:0];
    wire [1:0]outtest[3:0];
    model model(state,outtest);
    traffic_light tl(clk,reset,enable,out);

    always @( out[3] or  out[2] or  out[1] or  out[0]) begin
        if (state==17)
            state=0;
        else
            state=state+1;
    end
    always @(negedge clk)begin
        if(out[3:0] != outtest[3:0])
            begin
                $display("error at state=  0x%0h at time %d",state,$time);
            end
        end

    initial begin
        clk=0;
        reset=0;
        enable=1;
        #500 $finish;
    end
    always #1 clk=~clk;
endmodule
```

Figure 11:analyzer

If we make simulation we will find that the output from the model and the output from the system are identical and no message was sent as in the figure below.

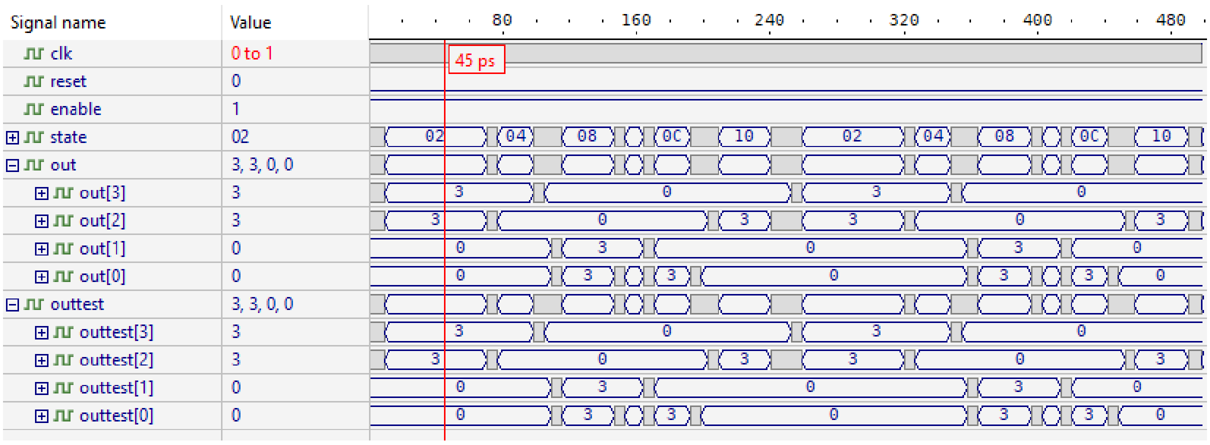

Figure 12:successful simulation

We will make an error on purpose by changing the next state for S12 instead of S13 to S10 and we will see that a message will be sent that there is error in this state as in the figure below.

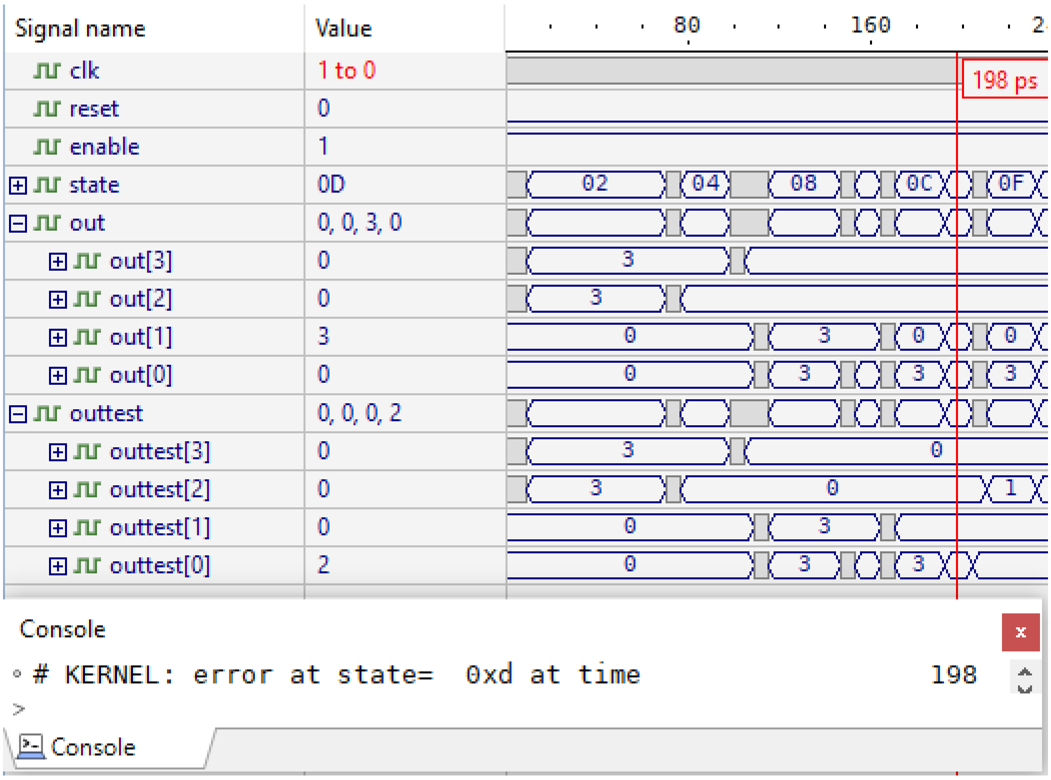

Figure 13: error simulation

Conclusion and Future works:

In this project, a Finite State Machine (FSM) based traffic light controller for a highway and farm road intersection will be designed using the Verilog hardware description language. The FSM will have 3 inputs: "Rst" for resetting the design to state 0 and counter to 0, "Go" for controlling the state transitions, and "clk" for synchronizing the system. The FSM will have 4 output signals, highway Signal 1 and highway signal 2 and farm Signal 1 and 2, which will control the traffic lights on the highway and farm road. The design will be verified using a simple test bench that will implement tests for all states of the FSM.

We did make a successful system code and we test it with multiple ways like the verification way and the simple test bench way. The system was successful in all cases.