FACULTY OF ENGINEERING AND TECHNOLOGY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ADVANCED DIGITAL DESIGN ENCS3310

COURSE PROJECT

Prepared by:

Name: Shereen Ibdah

Student ID: 1200373

Instructor: **Dr. Abdellatif Abu-Issa**

Section: 1

Date: 27\1\2023

Table of Contents

Brief Introduction and Background 4

Design Philosophy 5

Test Bench........ 10

Analyzer 11

Results........ 13

AnalyzerResult 14

Conclusion and Future works 16

Table of Figures:

Figure 1:counter.......... 5
Figure 2:Traffic light.......... 5
Figure 3:State Code.......... 7
Figure 4: outputs of states.......... 9
Figure 5test bench.......... 10
Figure 6:Analyzer.......... 11
Figure 7simulationofS2.......... 13
Figure 8simulationofallstates.......... 13
Figure 9AnalyzerOutput.......... 14
Figure 10NewVlauesOfState.......... 14
Figure 11TheErrorMessage.......... 15
Figure 12ResetCheck.......... 15

Tables :

Table 1:states delay.......... 4
Table 2:encoding for the colors.......... 8

Brief Introduction and Background

The goal of this project is to design a solution for a traffic light system, hence this solution aims to regulate the traffic between the highway and farm road intersection through the synchronization of the work between the traffic lights on both sides. We used Aldec Active HDL software for this project.

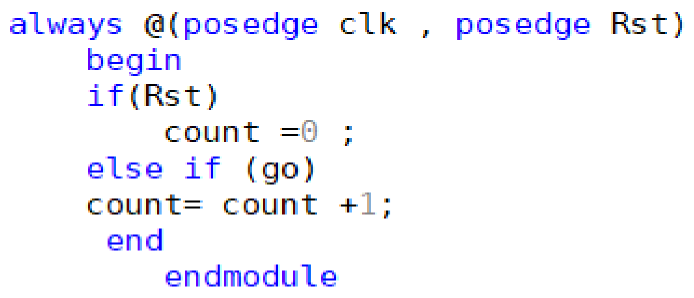
In order to implement this design, we designed a counter and a traffic light controller (FSM) that are synchronized through a common "clk". The design includes a state machine to control the traffic lights, a counter to keep track of the delay between states, and transitions between states based on the delay set for each state and the counter value. Additionally, we have included a Reset feature which resets the system and sets the counter value to zero, and set the current state to S0, considered the system's starting point. After implementation, we designed a test bench and analyzer to ensure that the results are correct.

State	Delay [Sec]
S0	1
S1	2
S2	30
S3	2
S4	10
S5	2
S6	1
S7	2
S8	15
S9	2
S10	5
S11	2
S12	10
S13	2
S14	1
S15	2
S16	15
S17	3

Table 1:states delay

Design Philosophy

counter: We implemented a counter to create a delay for each state. The counter will only count if the value of "Go" equals one, otherwise, it will "freeze". The counter is asynchronous and has a "Reset" feature that works independently of the "Clk" value. Once the "Reset" value becomes one (on the rising edge), it resets the value of the counter.



```
always @(posedge clk , posedge Rst)
    begin
    if(Rst)
        count =0 ;
    else if (go)
    count= count +1;
     end
        endmodule
```

Figure 1:counter

traffic_light: The system is composed of a counter and a Traffic Light Controller (FSM) that are connected and operate simultaneously. The figure represents this design configuration.

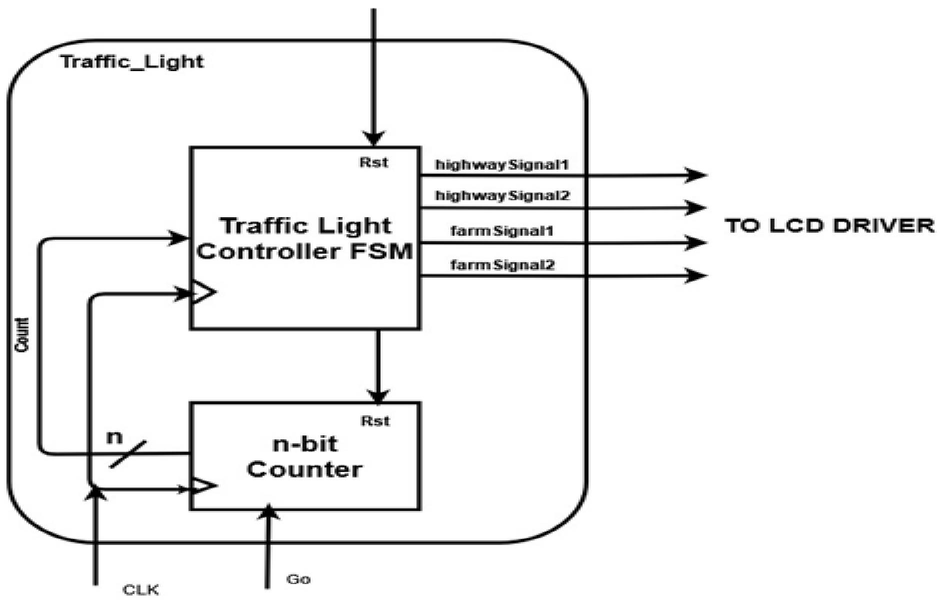
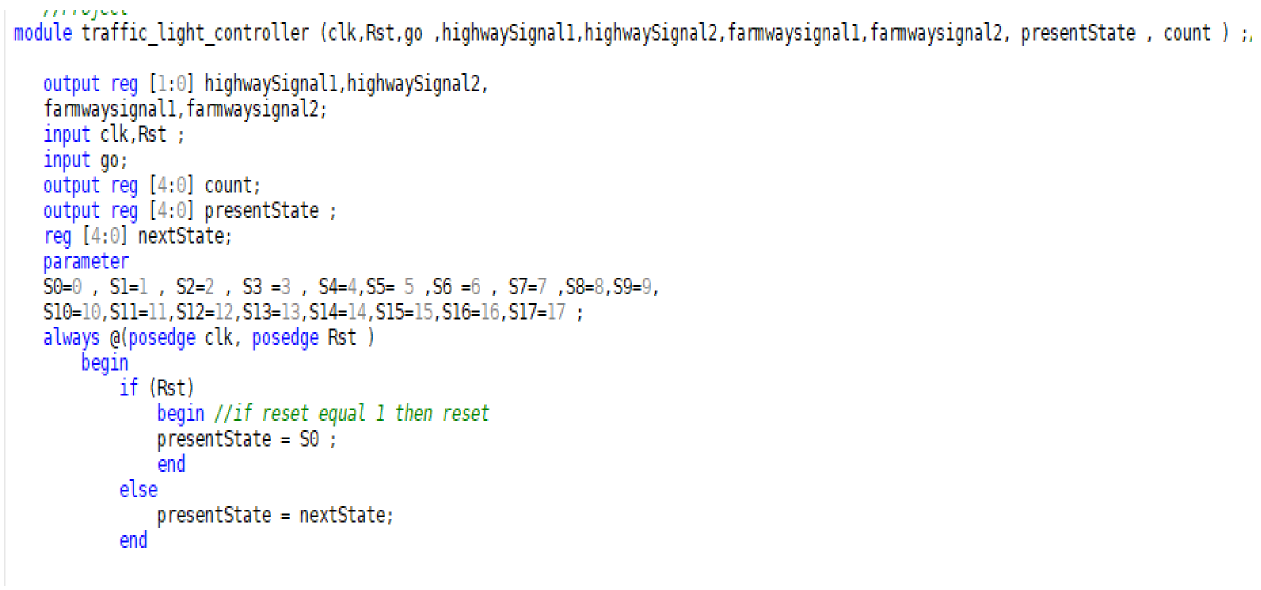


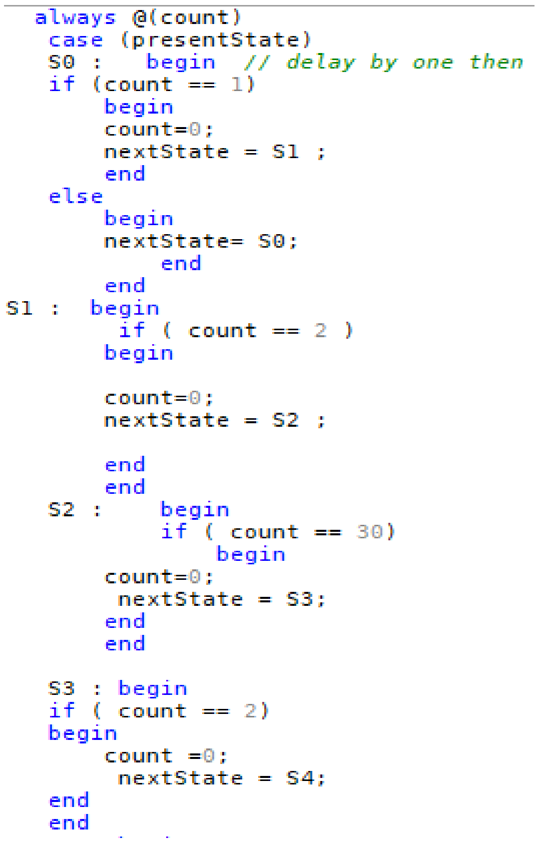
Figure 2:Traffic light

Here is the code:

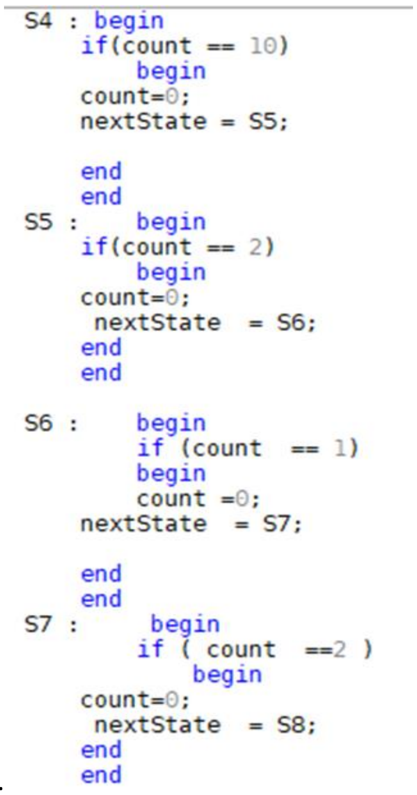
- The traffic light controller outputs four 2-bit signals, highway Signal 1 and highway signal 2, and farm Signal 1 and 2, for the highway road and the farm road, respectively.
- We can start by designing the FSM with 18 states S0 -> S17



```
 //Project
module traffic_light_controller (clk,Rst,go ,highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2, presentState , count ) ;

   output reg [1:0] highwaySignal1,highwaySignal2,
   farmwaysignal1,farmwaysignal2;
   input clk,Rst ;
   input go;
   output reg [4:0] count;
   output reg [4:0] presentState ;
   reg [4:0] nextState;
   parameter
   S0=0 , S1=1 , S2=2 , S3 =3 , S4=4,S5= 5 ,S6 =6 , S7=7 ,S8=8,S9=9,
   S10=10,S11=11,S12=12,S13=13,S14=14,S15=15,S16=16,S17=17 ;
   always @(posedge clk, posedge Rst )
       begin
           if (Rst)
               begin //if reset equal 1 then reset
               presentState = S0 ;
               end
           else
               presentState = nextState;
           end
```

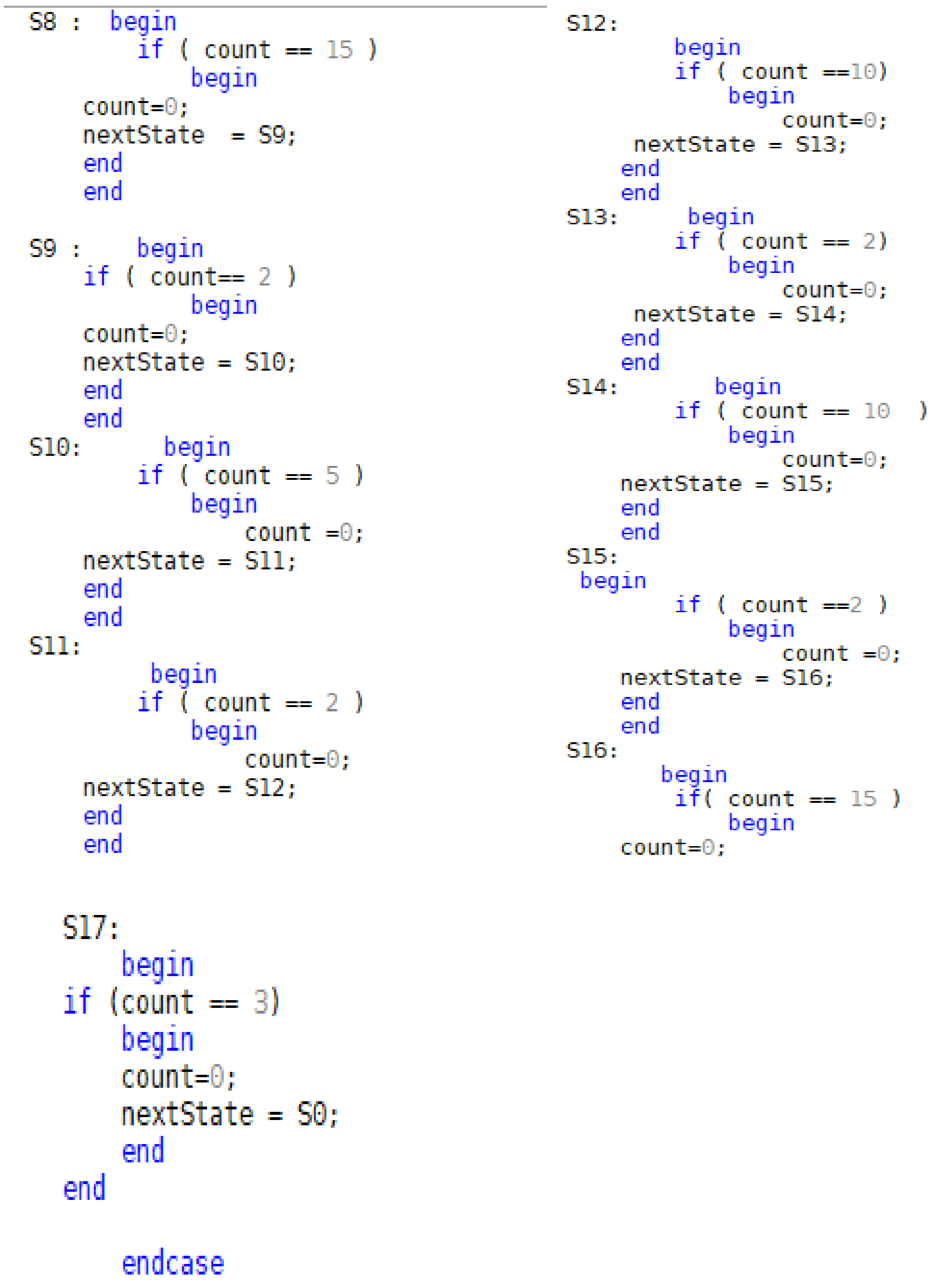


```
  always @(count)
   case (presentState)
   S0 :   begin  // delay by one then
   if (count == 1)
       begin
       count=0;
       nextState = S1 ;
       end
   else
       begin
       nextState= S0;
           end
       end
S1 :   begin
        if ( count == 2 )
       begin

       count=0;
       nextState = S2 ;

       end
       end
   S2 :    begin
           if ( count == 30)
               begin
       count=0;
        nextState = S3;
       end
       end

   S3 : begin
   if ( count == 2)
   begin
       count =0;
        nextState = S4;
   end
   end
```



```
S4 : begin
    if(count == 10)
        begin
    count=0;
    nextState = S5;

    end
    end
S5 :    begin
    if(count == 2)
        begin
    count=0;
     nextState  = S6;
    end
    end

S6 :    begin
        if (count   == 1)
        begin
        count =0;
    nextState  = S7;

    end
    end
S7 :     begin
        if ( count  ==2 )
            begin
    count=0;
     nextState  = S8;
    end
    end
```

❖ In each state the value of the counter will be zero, we return the value of the counter to zero so that the count starts from zero in the next state



```
S8 :  begin
        if ( count == 15 )
            begin
    count=0;
    nextState  = S9;
    end
    end

S9 :    begin
    if ( count== 2 )
            begin
    count=0;
    nextState = S10;
    end
    end
S10:      begin
        if ( count == 5 )
            begin
                count =0;
    nextState = S11;
    end
    end
S11:
          begin
        if ( count == 2 )
            begin
                count=0;
    nextState = S12;
    end
    end
```

```
S12:
        begin
        if ( count ==10)
            begin
                count=0;
     nextState = S13;
    end
    end
S13:     begin
        if ( count == 2)
            begin
                count=0;
     nextState = S14;
    end
    end
S14:        begin
        if ( count == 10   )
            begin
                count=0;
    nextState = S15;
    end
    end
S15:
 begin
        if ( count ==2 )
            begin
                count =0;
    nextState = S16;
    end
    end
S16:
       begin
        if( count == 15 )
            begin
    count=0;
```

```
S17:
    begin
if (count == 3)
    begin
    count=0;
    nextState = S0;
    end
end

    endcase
```

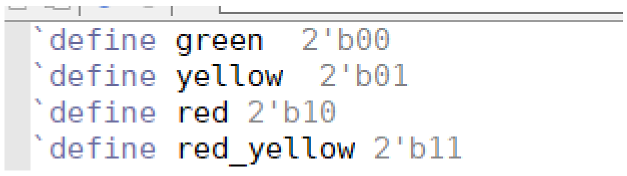
Figure 3:State Code

- in the system, state “i” can only transition to state “i+1” only if the value of go = “1” after a specified time ,and state S0 will remain in its current state if the reset value is 1
- The counter determines the specific time (delay) of the state
- When we reach the state S17 the next State will be S0
- encoding is used for both signals:

Green	00
Red	10
Yellow	01
Red-*Yellow*	11

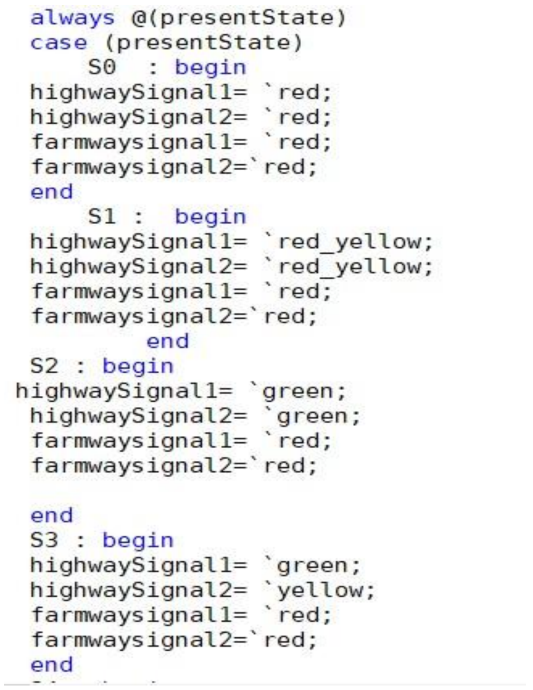
Table 2:encoding for the colors

- The values defined as constant

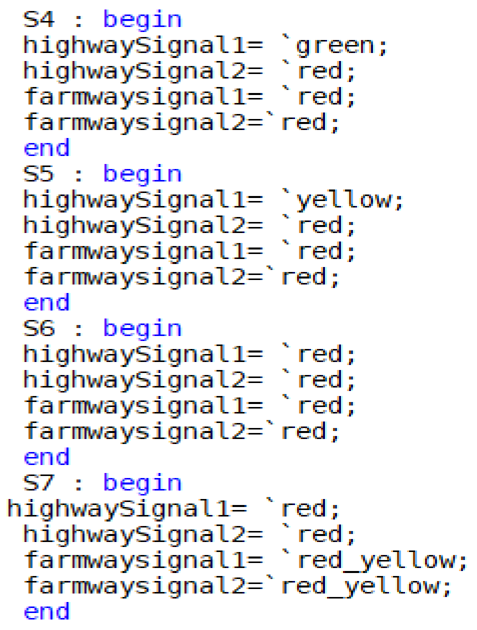


```
`define green  2'b00
`define yellow  2'b01
`define red 2'b10
`define red_yellow 2'b11
```

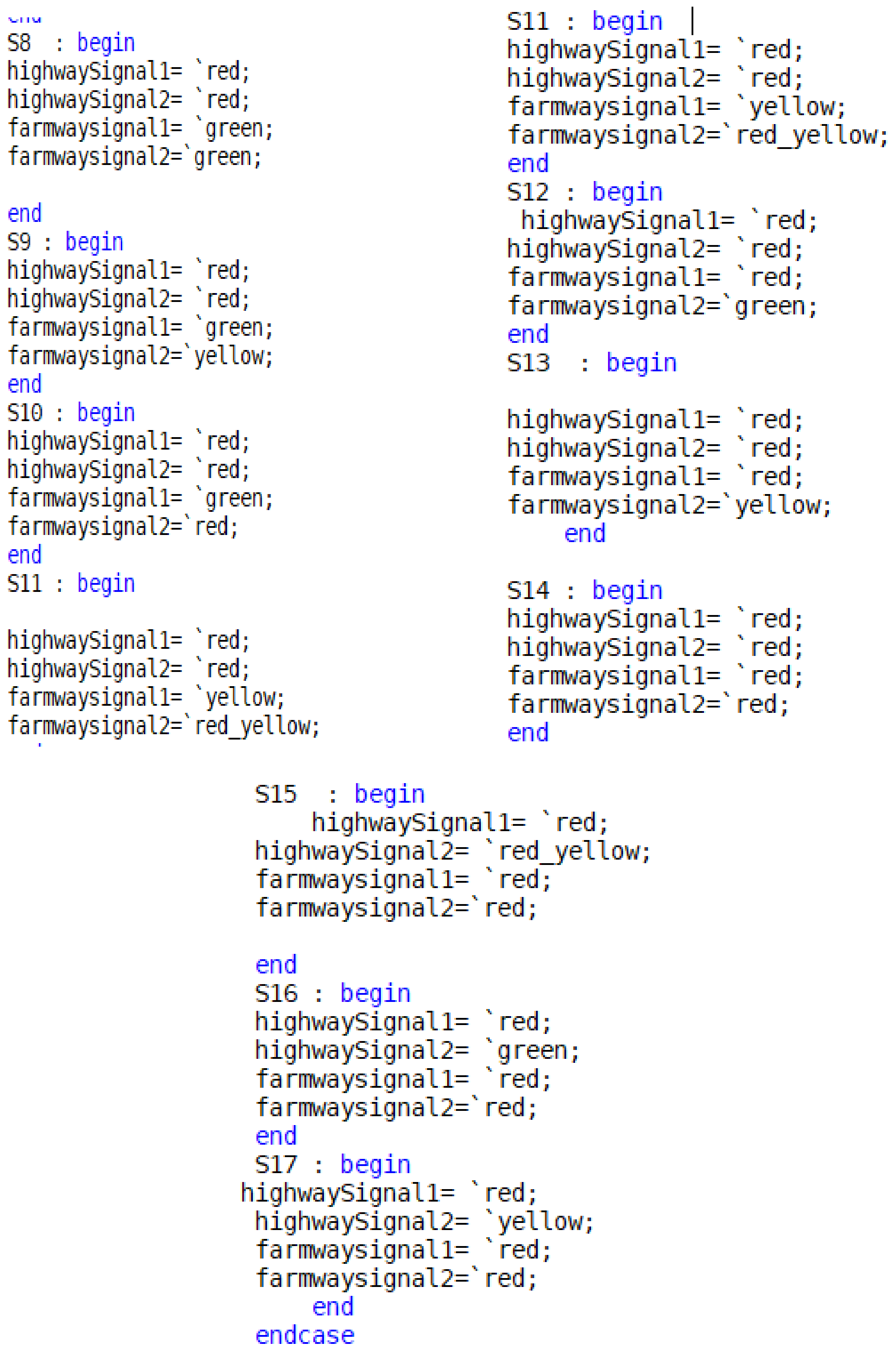
- The output for each state due to the given table in the Project will be:



```
always @(presentState)
case (presentState)
    S0  : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
    S1 :  begin
highwaySignal1= `red_yellow;
highwaySignal2= `red_yellow;
farmwaysignal1= `red;
farmwaysignal2=`red;
        end
S2 : begin
highwaySignal1= `green;
highwaySignal2= `green;
farmwaysignal1= `red;
farmwaysignal2=`red;

end
S3 : begin
highwaySignal1= `green;
highwaySignal2= `yellow;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
```



```
S4 : begin
highwaySignal1= `green;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
S5 : begin
highwaySignal1= `yellow;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
S6 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
S7 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red_yellow;
farmwaysignal2=`red_yellow;
end
```



```
end
S8  : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `green;
farmwaysignal2=`green;

end
S9 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `green;
farmwaysignal2=`yellow;
end
S10 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `green;
farmwaysignal2=`red;
end
S11 : begin

highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `yellow;
farmwaysignal2=`red_yellow;
```

```
S11 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `yellow;
farmwaysignal2=`red_yellow;
end
S12 : begin
 highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`green;
end
S13  : begin

highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`yellow;
    end

S14 : begin
highwaySignal1= `red;
highwaySignal2= `red;
farmwaysignal1= `red;
farmwaysignal2=`red;
end
```

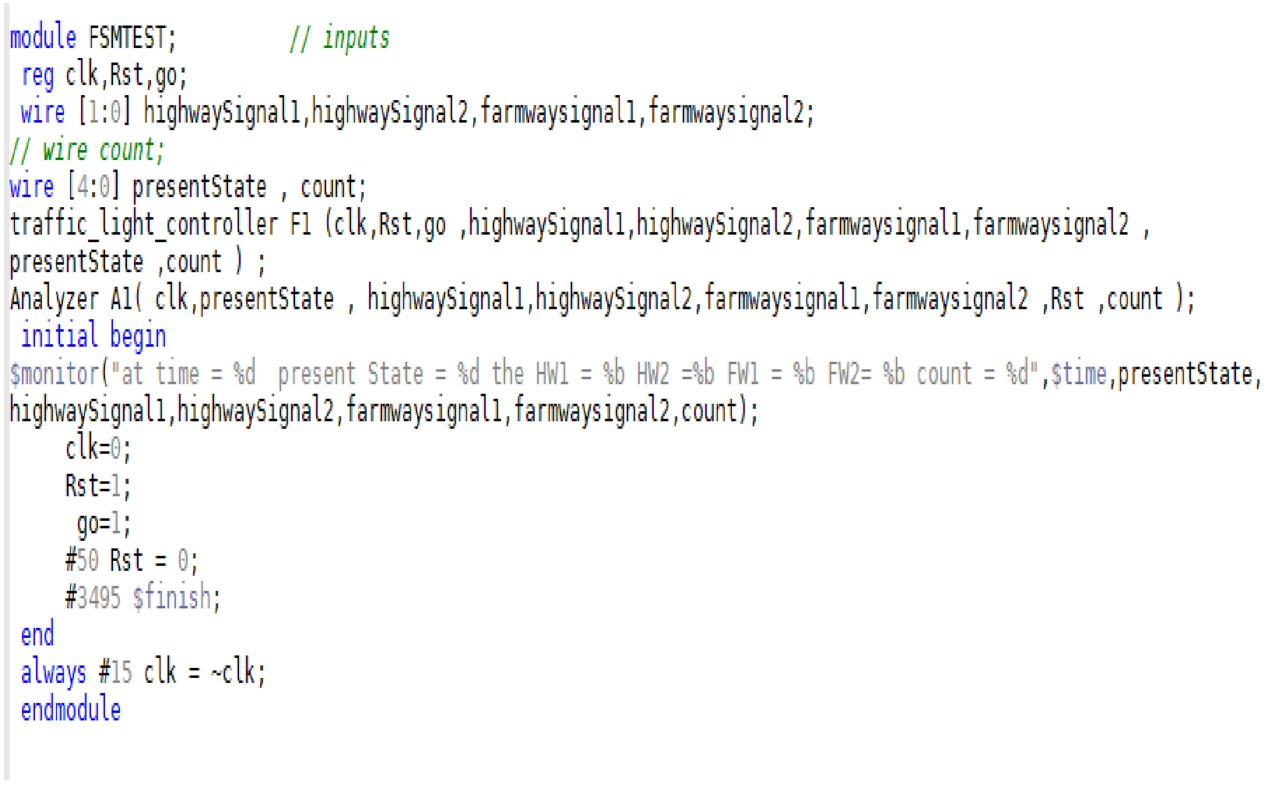
```
 S15  : begin
     highwaySignal1= `red;
 highwaySignal2= `red_yellow;
 farmwaysignal1= `red;
 farmwaysignal2=`red;

 end
 S16 : begin
 highwaySignal1= `red;
 highwaySignal2= `green;
 farmwaysignal1= `red;
 farmwaysignal2=`red;
 end
 S17 : begin
highwaySignal1= `red;
 highwaySignal2= `yellow;
 farmwaysignal1= `red;
 farmwaysignal2=`red;
     end
 endcase
```

Figure 4: outputs of states

Test Bench

A testbench is an HDL module that is used to test another module, called the device under test (DUT). The testbench contains statements to apply inputs to the DUT and, ideally, **to check that the correct outputs are produced**.

In my design, the (DUT) is the "traffic_light_controller" module, in hence to test the module



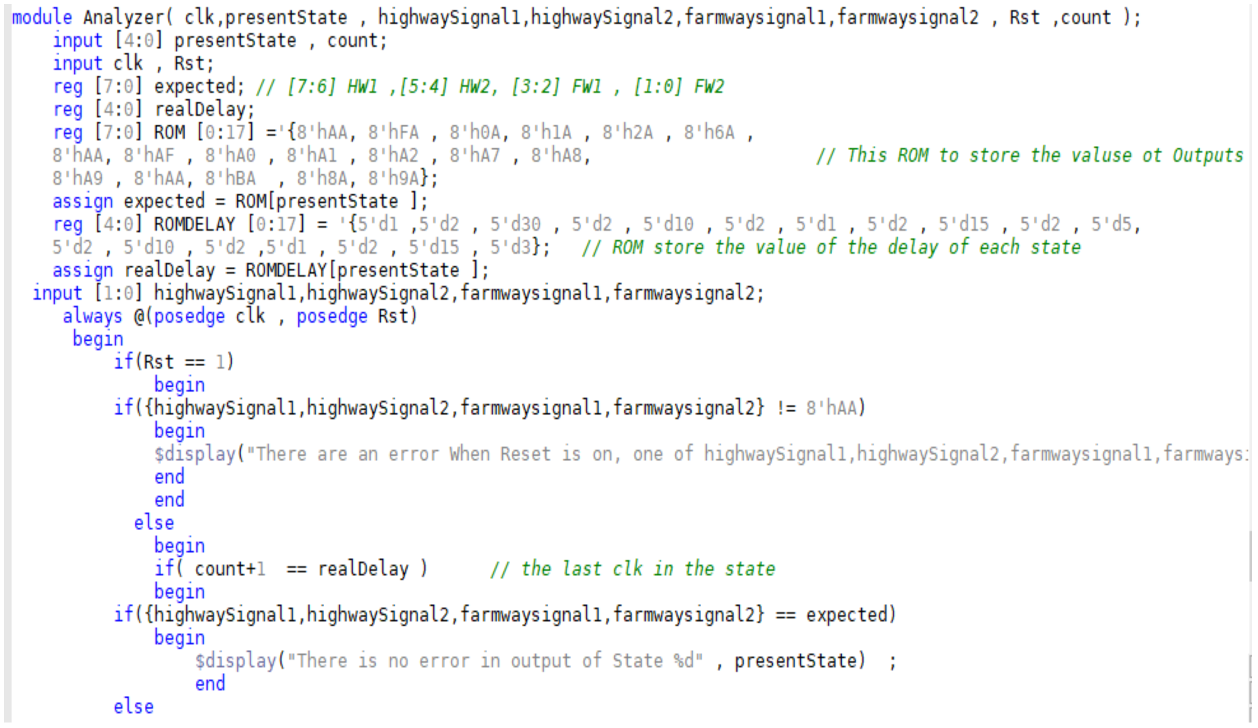
```
module FSMTEST;          // inputs
 reg clk,Rst,go;
 wire [1:0] highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2;
// wire count;
wire [4:0] presentState , count;
traffic_light_controller F1 (clk,Rst,go ,highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2 ,
presentState ,count ) ;
Analyzer A1( clk,presentState , highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2 ,Rst ,count );
 initial begin
$monitor("at time = %d  present State = %d the HW1 = %b HW2 =%b FW1 = %b FW2= %b count = %d",$time,presentState,
highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2,count);
     clk=0;
     Rst=1;
      go=1;
     #50 Rst = 0;
     #3495 $finish;
 end
 always #15 clk = ~clk;
 endmodule
```

Figure 5test bench

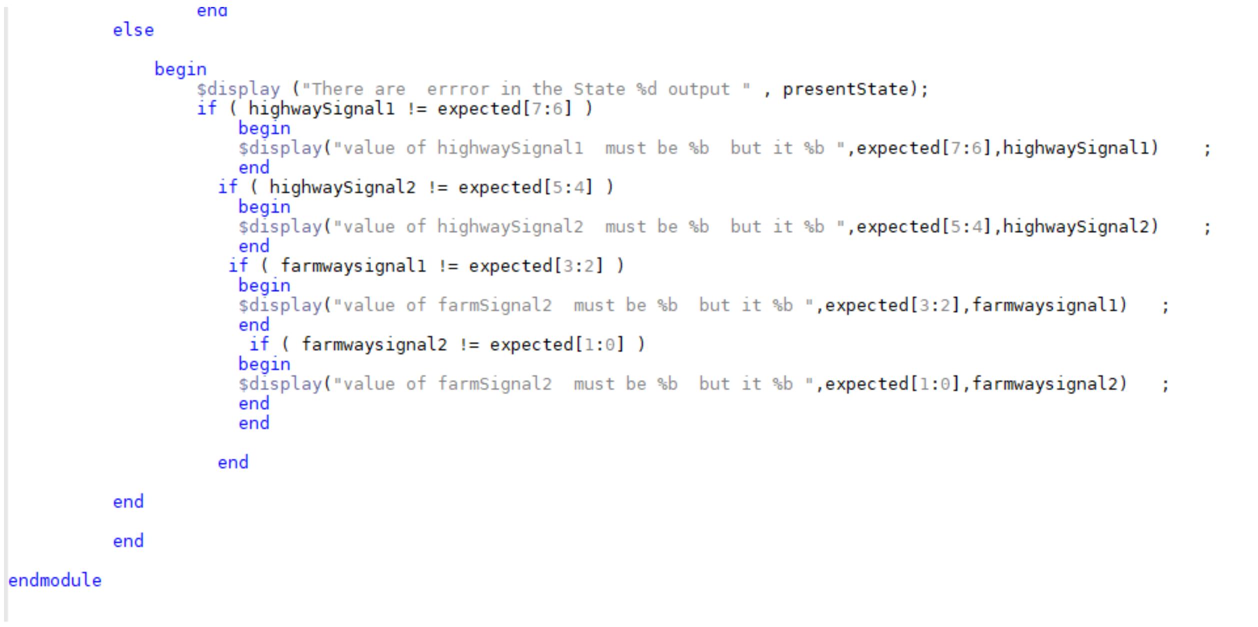
- The inputs of (DUT) becomes reg
- The outputs of (DUT) becomes wire
- We invoke an instance from the (DUT) module (traffic_light_controller)
- At the initial time of 0, the clock (clk) will be at a value of 0 and the reset (Rst) will be set to 1. As a result, the current state will be S0 and the value of the counter will also be 0. After 50 nanoseconds have passed, the reset will change to "zero," signaling for the system to begin operation. The system will then progress through the allowed states in a predetermined order.
- The System will be terminated after (3495 + 50) ns
- Whenever one of the input parameters for the traffic light controller is altered, the module will be activated and perform its function.

Analyzer

Verilog analyzers can be used to check the functionality of a design by comparing the expected behavior to the actual behavior of the circuit, then display if the result true or not.



```
module Analyzer( clk,presentState , highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2 , Rst ,count );
    input [4:0] presentState , count;
    input clk , Rst;
    reg [7:0] expected; // [7:6] HW1 ,[5:4] HW2, [3:2] FW1 , [1:0] FW2
    reg [4:0] realDelay;
    reg [7:0] ROM [0:17] ='{8'hAA, 8'hFA , 8'h0A, 8'h1A , 8'h2A , 8'h6A ,
    8'hAA, 8'hAF , 8'hA0 , 8'hA1 , 8'hA2 , 8'hA7 , 8'hA8,                  // This ROM to store the valuse ot Outputs
    8'hA9 , 8'hAA, 8'hBA  , 8'h8A, 8'h9A};
    assign expected = ROM[presentState ];
    reg [4:0] ROMDELAY [0:17] = '{5'd1 ,5'd2 , 5'd30 , 5'd2 , 5'd10 , 5'd2 , 5'd1 , 5'd2 , 5'd15 , 5'd2 , 5'd5,
    5'd2 , 5'd10 , 5'd2 ,5'd1 , 5'd2 , 5'd15 , 5'd3};   // ROM store the value of the delay of each state
    assign realDelay = ROMDELAY[presentState ];
  input [1:0] highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2;
     always @(posedge clk , posedge Rst)
      begin
          if(Rst == 1)
              begin
          if({highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2} != 8'hAA)
              begin
              $display("There are an error When Reset is on, one of highwaySignal1,highwaySignal2,farmwaysignal1,farmways
              end
              end
            else
              begin
              if( count+1  == realDelay )      // the last clk in the state
              begin
          if({highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2} == expected)
              begin
                  $display("There is no error in output of State %d" , presentState)  ;
                  end
          else
```



```
                  end
          else

              begin
                  $display ("There are  errror in the State %d output " , presentState);
                  if ( highwaySignal1 != expected[7:6] )
                      begin
                      $display("value of highwaySignal1  must be %b  but it %b ",expected[7:6],highwaySignal1)    ;
                      end
                    if ( highwaySignal2 != expected[5:4] )
                      begin
                      $display("value of highwaySignal2  must be %b  but it %b ",expected[5:4],highwaySignal2)    ;
                      end
                     if ( farmwaysignal1 != expected[3:2] )
                      begin
                      $display("value of farmSignal2  must be %b  but it %b ",expected[3:2],farmwaysignal1)   ;
                      end
                       if ( farmwaysignal2 != expected[1:0] )
                      begin
                      $display("value of farmSignal2  must be %b  but it %b ",expected[1:0],farmwaysignal2)   ;
                      end
                      end

                    end

          end

          end

endmodule
```

Figure 6:Analyzer

- For each current state, the output generated by the test bench module will be passed to the analyzer. The analyzer will then compare this output with the expected outputs stored in a read-only memory (ROM) cell with the address corresponding to the current state. In this way, the analyzer can verify that the circuit is producing the correct output for each state and detect any discrepancies between the expected and actual outputs.

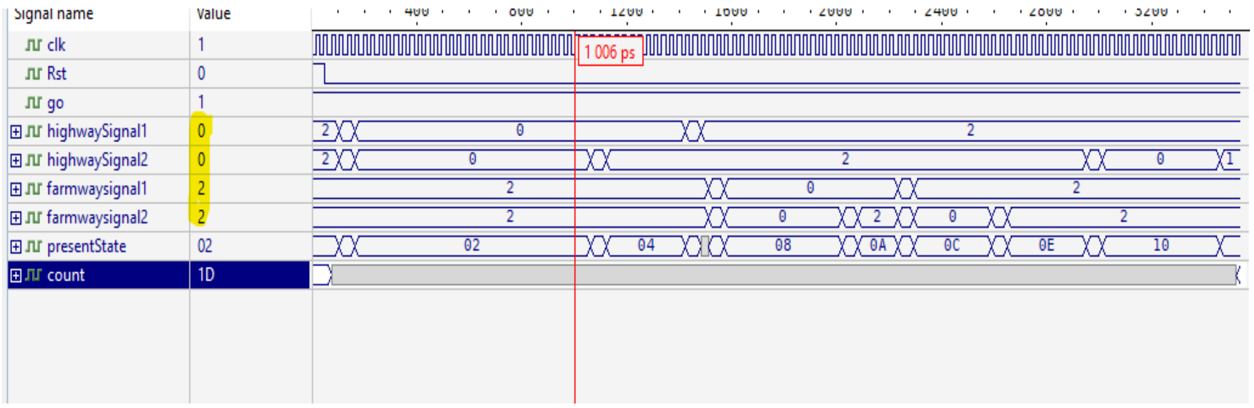
- The first ROM is used to store the correct output values for each state. This ROM has 18 storage locations and an 8-bit width. This is because the system has 18 states, and for each state there are 4 outputs, each of which is 2-bits wide. This allows the analyzer to compare the output of the circuit to the expected values stored in the ROM and determine if the circuit is functioning correctly.

- The second ROM is used to store the value of the delay for each state. This ROM will contain the time duration for each state to stay active. This information will be used by the analyzer to check the timing of the circuit and ensure that the state transitions are occurring at the correct time.

- After the outputs have been compared, if any errors are detected, a message will be displayed on the screen indicating the presence of an error. The specific output that caused the error will also be identified.

Results

These are some screenshots for the simulation shows the results for different states.

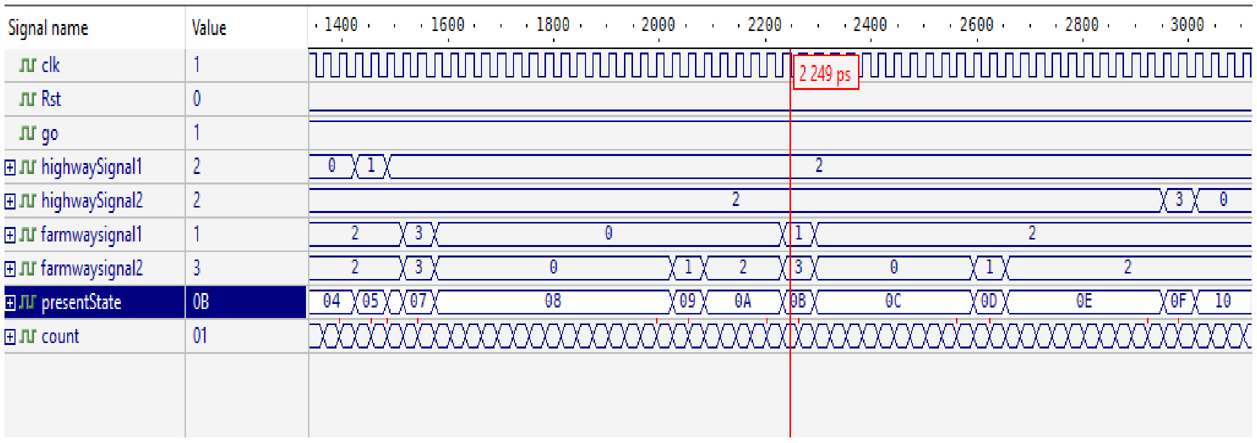


Signal name	Value
clk	1
Rst	0
go	1
highwaySignal1	0
highwaySignal2	0
farmwaysignal1	2
farmwaysignal2	2
presentState	02
count	1D

Figure 7simulationofS2

- ➢ For State S2 the output values are Green, Green ,Red, Red This corresponds to the values in the Figure 4.
- ➢ The counter value will be 1D, which is 29 in decimal. However, the delay of State S2 is 30. This is because when the counter reaches a value of 30, it will trigger the case statement for State S2 and reset the counter value back to 0. (30 clk cycles in the state)

The simulation of other states:

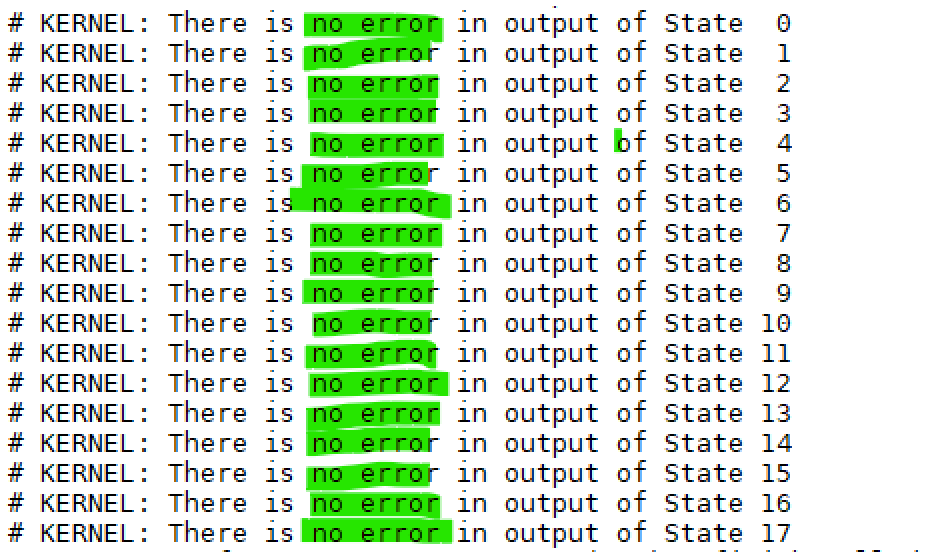


Signal name	Value
clk	1
Rst	0
go	1
highwaySignal1	2
highwaySignal2	2
farmwaysignal1	1
farmwaysignal2	3
presentState	0B
count	01

Figure 8simulationofallstates

AnalyzerResult

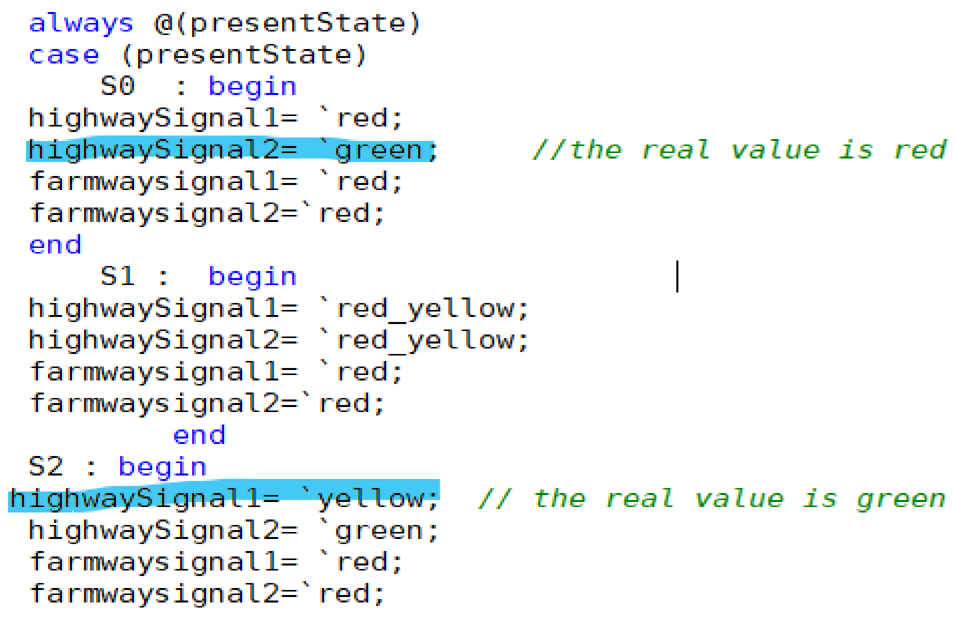
- Now the output of the Analyzer if the whole data is true.



```
# KERNEL: There is no error in output of State  0
# KERNEL: There is no error in output of State  1
# KERNEL: There is no error in output of State  2
# KERNEL: There is no error in output of State  3
# KERNEL: There is no error in output of State  4
# KERNEL: There is no error in output of State  5
# KERNEL: There is no error in output of State  6
# KERNEL: There is no error in output of State  7
# KERNEL: There is no error in output of State  8
# KERNEL: There is no error in output of State  9
# KERNEL: There is no error in output of State 10
# KERNEL: There is no error in output of State 11
# KERNEL: There is no error in output of State 12
# KERNEL: There is no error in output of State 13
# KERNEL: There is no error in output of State 14
# KERNEL: There is no error in output of State 15
# KERNEL: There is no error in output of State 16
# KERNEL: There is no error in output of State 17
```

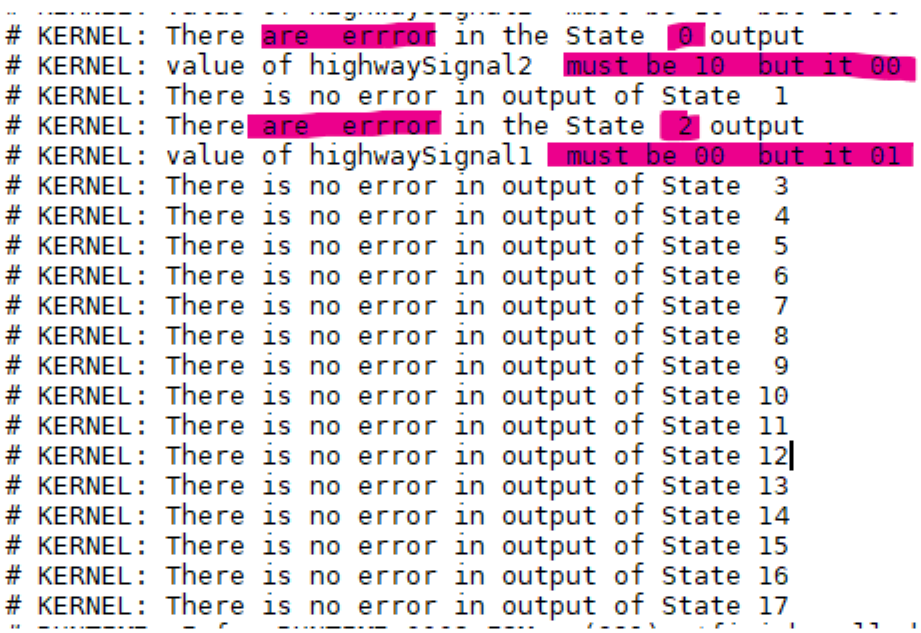
Figure 9AnalyzerOutput

- Now if we change in the output data of state S0, S2 The result will be



```
always @(presentState)
case (presentState)
    S0  : begin
highwaySignal1= `red;
highwaySignal2= `green;      //the real value is red
farmwaysignal1= `red;
farmwaysignal2=`red;
end
    S1 :  begin
highwaySignal1= `red_yellow;
highwaySignal2= `red_yellow;
farmwaysignal1= `red;
farmwaysignal2=`red;
        end
S2 : begin
highwaySignal1= `yellow;  // the real value is green
highwaySignal2= `green;
farmwaysignal1= `red;
farmwaysignal2=`red;
```

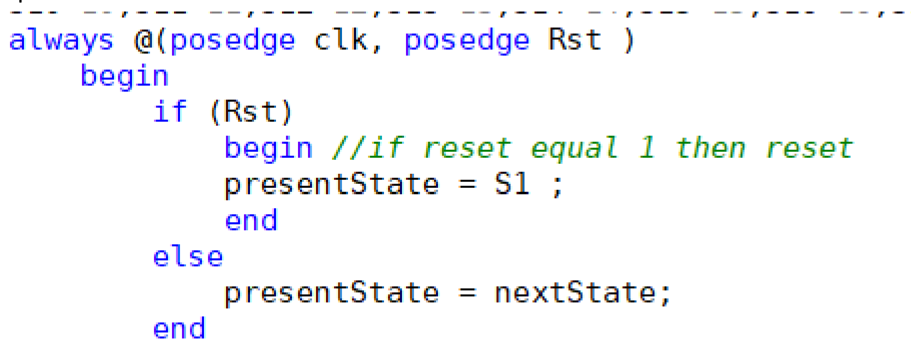
Figure 10NewVlauesOfState



```
# KERNEL: There are  errror in the State  0 output
# KERNEL: value of highwaySignal2  must be 10  but it 00
# KERNEL: There is no error in output of State  1
# KERNEL: There are  errror in the State  2 output
# KERNEL: value of highwaySignal1  must be 00  but it 01
# KERNEL: There is no error in output of State  3
# KERNEL: There is no error in output of State  4
# KERNEL: There is no error in output of State  5
# KERNEL: There is no error in output of State  6
# KERNEL: There is no error in output of State  7
# KERNEL: There is no error in output of State  8
# KERNEL: There is no error in output of State  9
# KERNEL: There is no error in output of State 10
# KERNEL: There is no error in output of State 11
# KERNEL: There is no error in output of State 12
# KERNEL: There is no error in output of State 13
# KERNEL: There is no error in output of State 14
# KERNEL: There is no error in output of State 15
# KERNEL: There is no error in output of State 16
# KERNEL: There is no error in output of State 17
```

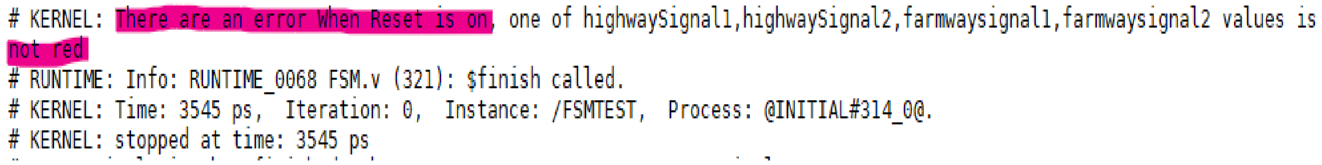
Figure 11TheErrorMessage

- Also, If we change the Present State when Reset On to be other value then S0 the error message will be printed



```
always @(posedge clk, posedge Rst )
    begin
        if (Rst)
            begin //if reset equal 1 then reset
            presentState = S1 ;
            end
        else
            presentState = nextState;
        end
```

Figure 12ResetCheck



```
# KERNEL: There are an error When Reset is on, one of highwaySignal1,highwaySignal2,farmwaysignal1,farmwaysignal2 values is
not red
# RUNTIME: Info: RUNTIME_0068 FSM.v (321): $finish called.
# KERNEL: Time: 3545 ps,  Iteration: 0,  Instance: /FSMTEST,  Process: @INITIAL#314_0@.
# KERNEL: stopped at time: 3545 ps
```

Conclusion and Future works

In summary, our understanding of state machines has improved and we have gained proficiency in working with sequential circuits. We have gained the ability to control the color of traffic lights after a set period of time and have developed the skills to design real-world devices such as traffic lights. Additionally, we have become familiar with sequential logic and the techniques for testing and debugging our designs, including the ability to print errors and implement delays.

In This project the traffic light controller was implemented using a finite state machine, which is a powerful tool for controlling complex systems. This project helped to understanding of how to design and implement a finite state machine.