Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS3310, Advanced Digital Systems Design

Project Report

Traffic Light Controller

Prepared by: Zainab Jaradat

ID Number : 1201766

Instructor : Dr. Abdellatif Abu-Issa

Section : 3

Date : January 27, 2023

Contents

Table of Figures

Brief Introduction and Background

In this project, we will design a solution to real problem which is the traffic light controller and design the highway and farm road intersection as shown in Figure1 , by using Aldec Active-HDL student Edition to write Verilog codes, simulate the system and check the output.

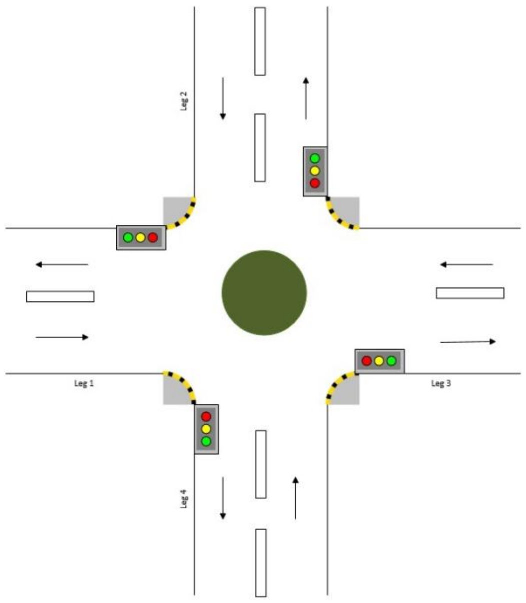
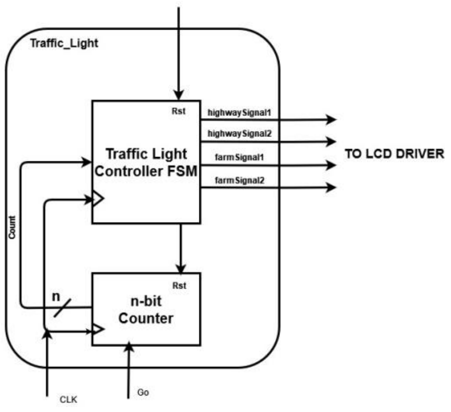


Figure 1 : Highway and Farm Roads

From the block diagram shown below we notice that our traffic light controller has three inputs (CLK , RST , GO), and four 2-bit outputs (highway_signal_1, highwaysignal2, farmsignal1, and farmsignal2).



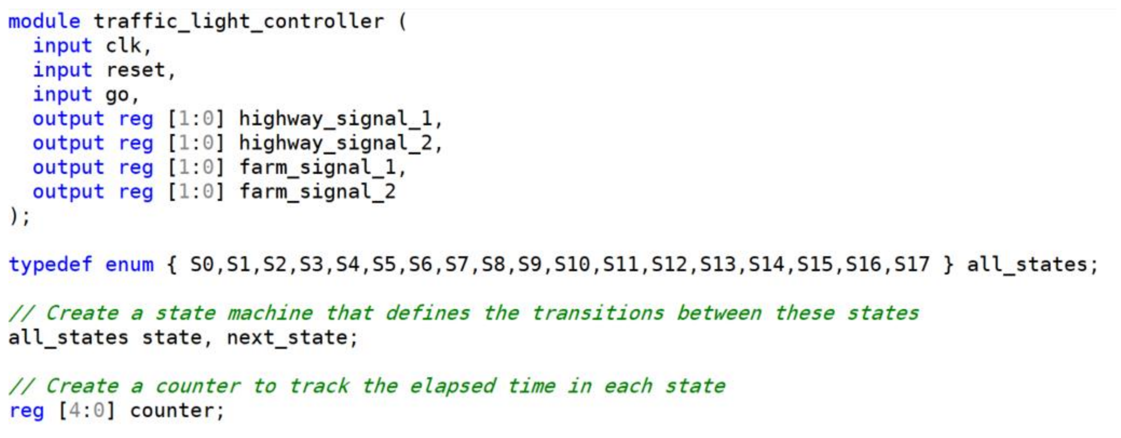
To implement this block we have to design finite state machine with 18 states given (S0-S17) . FSM is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

Design philosophy

First, I have created a module and named it "traffic_light_controller" as shown in Figure 1. The module has three input signals : "clk", "reset", and "go", and four 2-bit output signals : "highway_signal_1", "highway_signal_2", "farm_signal_1", and "farm_signal_2".

Then I used an enumerated type "all_states" to define the different states that the traffic light can be in. The current state is stored in the "state" variable and the next state is stored in "next_state".

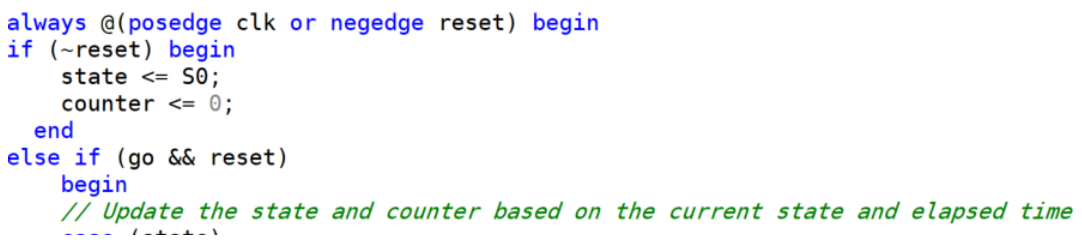
A counter is also used to track the time spent in each state. The reason why the counter is a 5-bit register in the code is because it needs to be able to count up to the maximum delay specified in the table, which is 30 seconds ($2^5 = 32$).



```
module traffic_light_controller (
  input clk,
  input reset,
  input go,
  output reg [1:0] highway_signal_1,
  output reg [1:0] highway_signal_2,
  output reg [1:0] farm_signal_1,
  output reg [1:0] farm_signal_2
);

typedef enum { S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16,S17 } all_states;

// Create a state machine that defines the transitions between these states
all_states state, next_state;

// Create a counter to track the elapsed time in each state
reg [4:0] counter;
```

Figure 1 : My Module

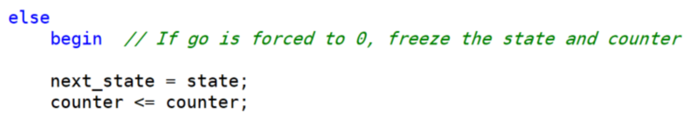
My finite state machine implemented using an always block, which is triggered on the rising edge of the "clk" input or the falling edge of the "reset" input. Figure 2 shows that when the "reset" input is 0, the state is set to S0 and the counter is reset to zero. When the "go" input is active, the state machine transitions through different states, depending on the current state and the value of the counter.



```
always @(posedge clk or negedge reset) begin
if (~reset) begin
    state <= S0;
    counter <= 0;
  end
else if (go && reset)
    begin
    // Update the state and counter based on the current state and elapsed time
```

Figure 2: Rst and Go

Figure 3 shows that If go is forced to zero the counter will stop counting.

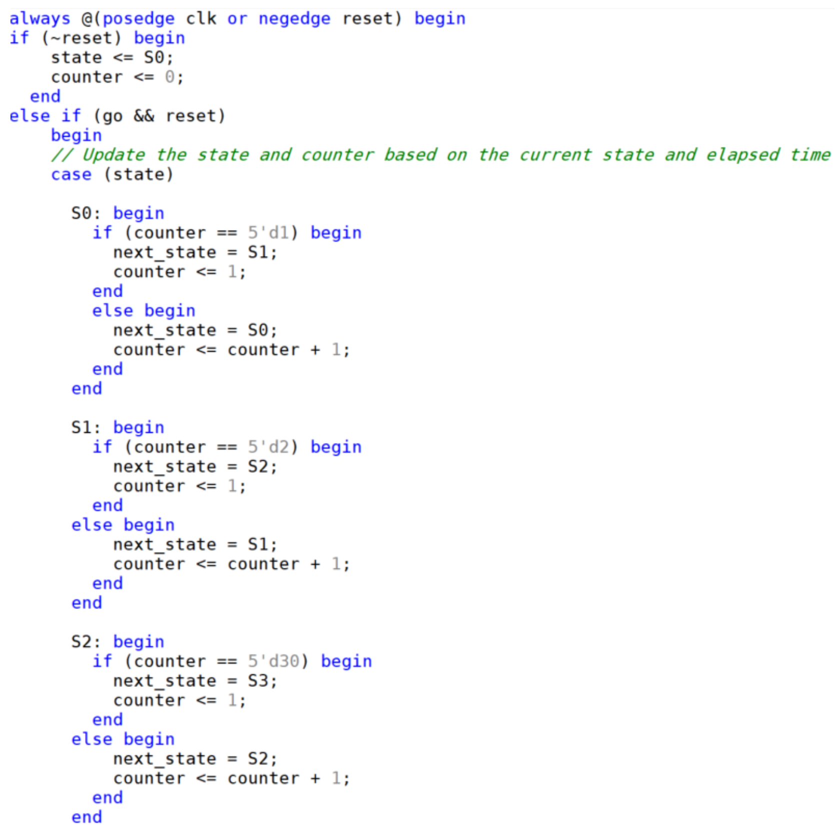


```
else
    begin  // If go is forced to 0, freeze the state and counter

    next_state = state;
    counter <= counter;
```

Figure 3 : Go = 0

The always block contains a case statement, which updates the state and counter based on the current state and time, and sets the output signals based on the current state and the value of the counter. For example, when the current state is S0 and the counter value is 1, the next state will be S1 and the counter will be reset to 0. Similarly, when the current state is S1 and the counter value is 2, the next state will be S2, and so on.

In the Figure below , for each case, the if statement in the each block checks the value of the counter signal. If counter is equal to delay then update next_state signal and the reset counter to 1. If counter is not equal to delay, the next_state signal is set to the current stat and the counter is incremented by 1.



```
always @(posedge clk or negedge reset) begin
if (~reset) begin
    state <= S0;
    counter <= 0;
  end
else if (go && reset)
    begin
    // Update the state and counter based on the current state and elapsed time
    case (state)

      S0: begin
        if (counter == 5'd1) begin
          next_state = S1;
          counter <= 1;
        end
        else begin
          next_state = S0;
          counter <= counter + 1;
        end
      end

      S1: begin
        if (counter == 5'd2) begin
          next_state = S2;
          counter <= 1;
        end
      else begin
          next_state = S1;
          counter <= counter + 1;
        end
      end

      S2: begin
        if (counter == 5'd30) begin
          next_state = S3;
          counter <= 1;
        end
      else begin
          next_state = S2;
          counter <= counter + 1;
        end
      end
```

Figure 4 : Counter Block

I used the "always_comb" and "case" statements to determine the outputs based on the current state. The "always_comb" statement ensures that the outputs are updated continuously, and the "case" statement checks the current state from S0 to S17 and sets the outputs accordingly.

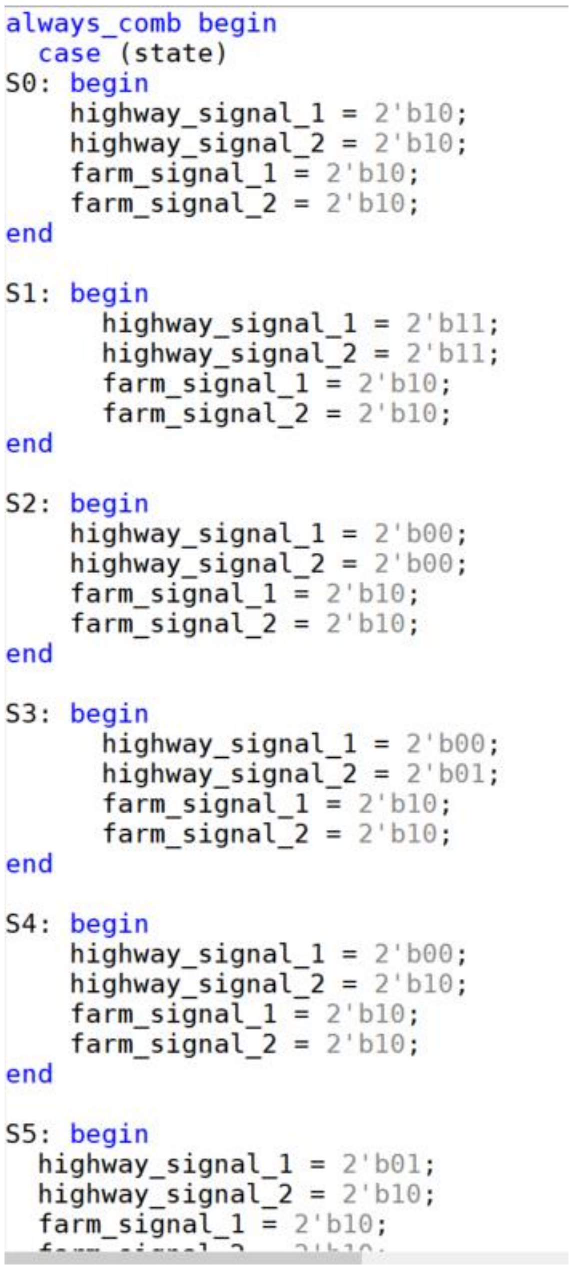
The following encoding is used for both output signals:

00 : Green

01 : Yellow (when changing from green)

10 : Red

11 : Red-Yellow (when changing from red)

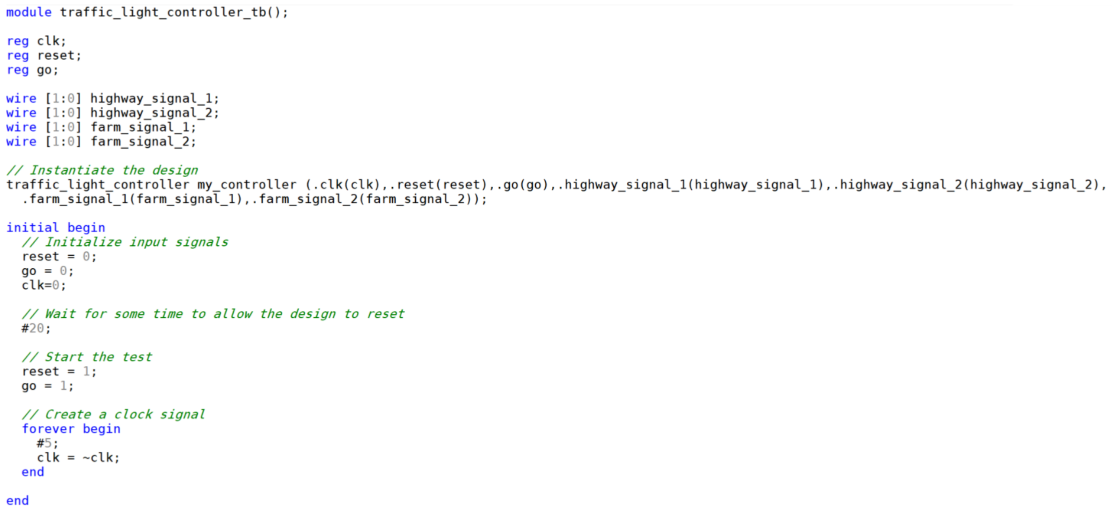


```
always_comb begin
  case (state)
S0: begin
    highway_signal_1 = 2'b10;
    highway_signal_2 = 2'b10;
    farm_signal_1 = 2'b10;
    farm_signal_2 = 2'b10;
end

S1: begin
      highway_signal_1 = 2'b11;
      highway_signal_2 = 2'b11;
      farm_signal_1 = 2'b10;
      farm_signal_2 = 2'b10;
end

S2: begin
    highway_signal_1 = 2'b00;
    highway_signal_2 = 2'b00;
    farm_signal_1 = 2'b10;
    farm_signal_2 = 2'b10;
end

S3: begin
      highway_signal_1 = 2'b00;
      highway_signal_2 = 2'b01;
      farm_signal_1 = 2'b10;
      farm_signal_2 = 2'b10;
end

S4: begin
    highway_signal_1 = 2'b00;
    highway_signal_2 = 2'b10;
    farm_signal_1 = 2'b10;
    farm_signal_2 = 2'b10;
end

S5: begin
  highway_signal_1 = 2'b01;
  highway_signal_2 = 2'b10;
  farm_signal_1 = 2'b10;
```

Figure 5: Output block

Verification

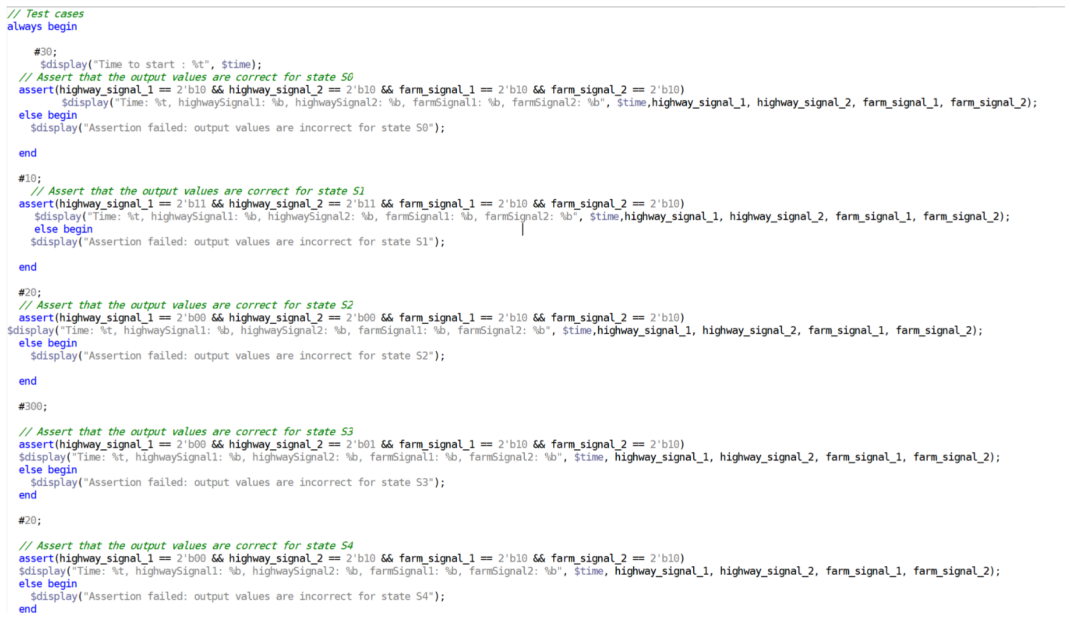
I had implemented TestBench and named it traffic_light_controller_tb , to test my traffic_light_controller design. It instantiates the traffic light controller module and assigns input and output signals to the corresponding ports. The Figure below shows that "reset" and "go" signals are set to 0 initially and then set to 1 after a delay of 20 time units to allow the design to reset. The "forever" loop creates a clock signal with a period of 10 time units.



```
module traffic_light_controller_tb();

reg clk;
reg reset;
reg go;

wire [1:0] highway_signal_1;
wire [1:0] highway_signal_2;
wire [1:0] farm_signal_1;
wire [1:0] farm_signal_2;

// Instantiate the design
traffic_light_controller my_controller (.clk(clk),.reset(reset),.go(go),.highway_signal_1(highway_signal_1),.highway_signal_2(highway_signal_2),
  .farm_signal_1(farm_signal_1),.farm_signal_2(farm_signal_2));

initial begin
  // Initialize input signals
  reset = 0;
  go = 0;
  clk=0;

  // Wait for some time to allow the design to reset
  #20;

  // Start the test
  reset = 1;
  go = 1;

  // Create a clock signal
  forever begin
    #5;
    clk = ~clk;
  end

end
```

Figure 6 : TestBench

Then I used the **assert** statement to check if the output values are correct for <u>each</u> state. If the assertion is true, it displays the current time, output values using $display else it will display Assertion failed: output values are incorrect for state.

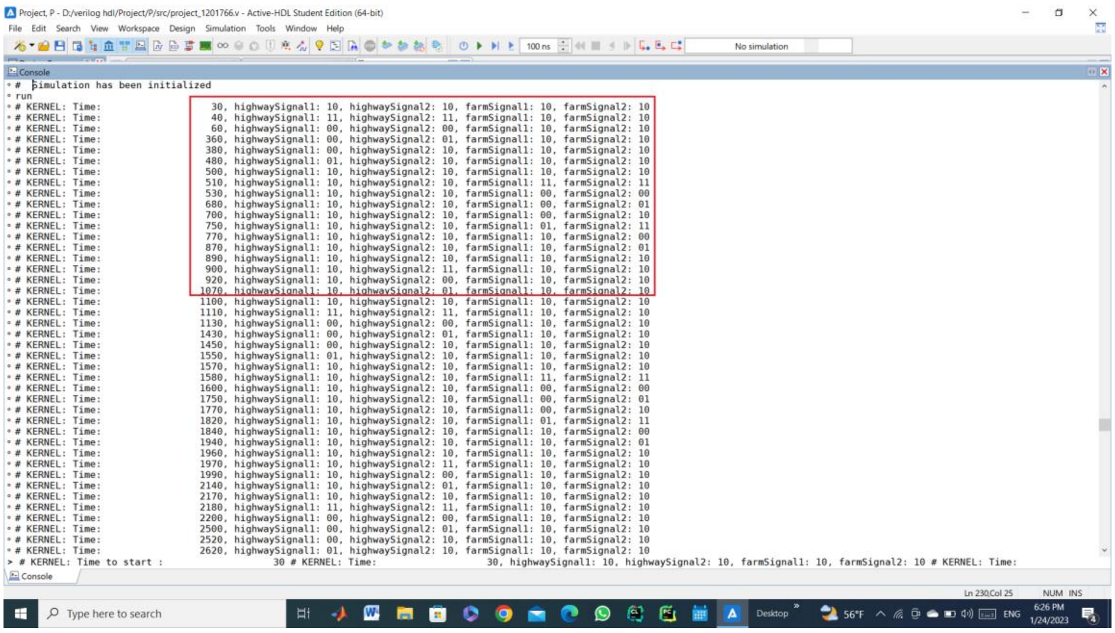


```
// Test cases
always begin

    #30;
     $display("Time to start : %t", $time);
  // Assert that the output values are correct for state S0
  assert(highway_signal_1 == 2'b10 && highway_signal_2 == 2'b10 && farm_signal_1 == 2'b10 && farm_signal_2 == 2'b10)
        $display("Time: %t, highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,highway_signal_1, highway_signal_2, farm_signal_1, farm_signal_2);
  else begin
    $display("Assertion failed: output values are incorrect for state S0");

  end

  #10;
    // Assert that the output values are correct for state S1
  assert(highway_signal_1 == 2'b11 && highway_signal_2 == 2'b11 && farm_signal_1 == 2'b10 && farm_signal_2 == 2'b10)
     $display("Time: %t, highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,highway_signal_1, highway_signal_2, farm_signal_1, farm_signal_2);
     else begin
    $display("Assertion failed: output values are incorrect for state S1");

  end

  #20;
  // Assert that the output values are correct for state S2
  assert(highway_signal_1 == 2'b00 && highway_signal_2 == 2'b00 && farm_signal_1 == 2'b10 && farm_signal_2 == 2'b10)
$display("Time: %t, highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,highway_signal_1, highway_signal_2, farm_signal_1, farm_signal_2);
  else begin
    $display("Assertion failed: output values are incorrect for state S2");

  end

  #300;

  // Assert that the output values are correct for state S3
  assert(highway_signal_1 == 2'b00 && highway_signal_2 == 2'b01 && farm_signal_1 == 2'b10 && farm_signal_2 == 2'b10)
  $display("Time: %t, highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time, highway_signal_1, highway_signal_2, farm_signal_1, farm_signal_2);
  else begin
    $display("Assertion failed: output values are incorrect for state S3");
  end

  #20;

  // Assert that the output values are correct for state S4
  assert(highway_signal_1 == 2'b00 && highway_signal_2 == 2'b10 && farm_signal_1 == 2'b10 && farm_signal_2 == 2'b10)
  $display("Time: %t, highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time, highway_signal_1, highway_signal_2, farm_signal_1, farm_signal_2);
  else begin
    $display("Assertion failed: output values are incorrect for state S4");
  end
```

Figure 7 : Assert

Results

The figure below shows the 18 states outputs and delays are true , I make it works to infinity since the traffic light don't have to stop. But when you test it, I will add finish to make it easier to understand.

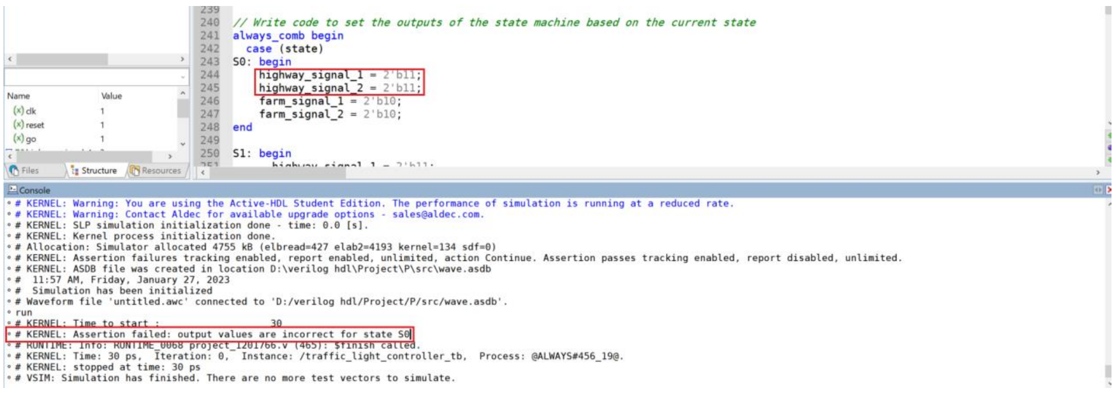


```
# Simulation has been initialized
run
# KERNEL: Time:                     30, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                     40, highwaySignal1: 11, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                     60, highwaySignal1: 00, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    360, highwaySignal1: 00, highwaySignal2: 01, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    380, highwaySignal1: 00, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    480, highwaySignal1: 01, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    500, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    510, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 11, farmSignal2: 11
# KERNEL: Time:                    530, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 00
# KERNEL: Time:                    680, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 01
# KERNEL: Time:                    700, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 10
# KERNEL: Time:                    750, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 01, farmSignal2: 11
# KERNEL: Time:                    770, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 00
# KERNEL: Time:                    870, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 01
# KERNEL: Time:                    890, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    900, highwaySignal1: 10, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                    920, highwaySignal1: 10, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1070, highwaySignal1: 10, highwaySignal2: 01, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1100, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1110, highwaySignal1: 11, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1130, highwaySignal1: 00, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1430, highwaySignal1: 00, highwaySignal2: 01, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1450, highwaySignal1: 00, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1550, highwaySignal1: 01, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1570, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1580, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 11, farmSignal2: 11
# KERNEL: Time:                   1600, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 00
# KERNEL: Time:                   1750, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 01
# KERNEL: Time:                   1770, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 00, farmSignal2: 10
# KERNEL: Time:                   1820, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 01, farmSignal2: 11
# KERNEL: Time:                   1840, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 00
# KERNEL: Time:                   1940, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 01
# KERNEL: Time:                   1960, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1970, highwaySignal1: 10, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   1990, highwaySignal1: 10, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2140, highwaySignal1: 10, highwaySignal2: 01, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2170, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2180, highwaySignal1: 11, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2200, highwaySignal1: 00, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2500, highwaySignal1: 00, highwaySignal2: 01, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2520, highwaySignal1: 00, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time:                   2620, highwaySignal1: 01, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
> # KERNEL: Time to start :              30 # KERNEL: Time:              30, highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10 # KERNEL: Time:
```

Project, P - D:/verilog hdl/Project/P/src/project_1201766.v - Active-HDL Student Edition (64-bit)
File Edit Search View Workspace Design Simulation Tools Window Help
100 ns
No simulation
Console
Ln 230,Col 25
NUM INS
Type here to search
Desktop
56°F
ENG
6:26 PM
1/24/2023

Figure 8: Output

Now, I tried to enter incorrect values for state zero and check if my system will know this or not.



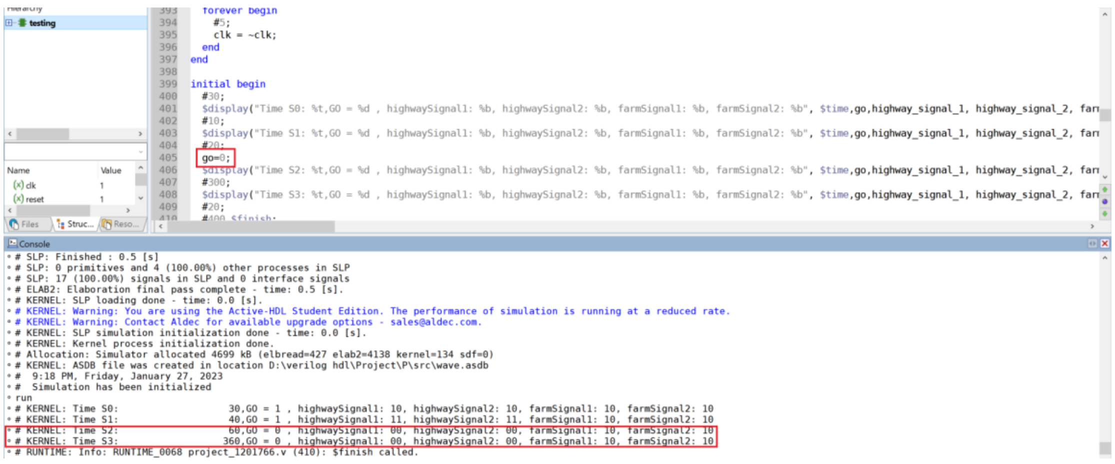
```
// Write code to set the outputs of the state machine based on the current state
always_comb begin
  case (state)
S0: begin
     highway_signal_1 = 2'b11;
     highway_signal_2 = 2'b11;
     farm_signal_1 = 2'b10;
     farm_signal_2 = 2'b10;
end

S1: begin
```

```
# KERNEL: Warning: You are using the Active-HDL Student Edition. The performance of simulation is running at a reduced rate.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4755 kB (elbread=427 elab2=4193 kernel=134 sdf=0)
# KERNEL: Assertion failures tracking enabled, report enabled, unlimited, action Continue. Assertion passes tracking enabled, report disabled, unlimited.
# KERNEL: ASDB file was created in location D:\verilog hdl\Project\P\src\wave.asdb
#  11:57 AM, Friday, January 27, 2023
#  Simulation has been initialized
# Waveform file 'untitled.awc' connected to 'D:/verilog hdl/Project/P/src/wave.asdb'.
run
# KERNEL: Time to start :                     30
# KERNEL: Assertion failed: output values are incorrect for state S0
# RUNTIME: Info: RUNTIME_0068 project_1201766.v (465): $finish called.
# KERNEL: Time: 30 ps,  Iteration: 0,  Instance: /traffic_light_controller_tb,  Process: @ALWAYS#456_19@.
# KERNEL: stopped at time: 30 ps
# VSIM: Simulation has finished. There are no more test vectors to simulate.
```

Figure 9:Testing Errors

As shown above, I changed RED to RED_YELLOW in first case for both "highway_signal_1" and "highway_signal_2", then an message occurs to show that is not correct, and the system calls $finish since traffic light controller have to stop.

To test if go working properly or not, I made another simple test bench and named it testing, the reason of this is to show how the state and the counter freezes when I set go to zero.

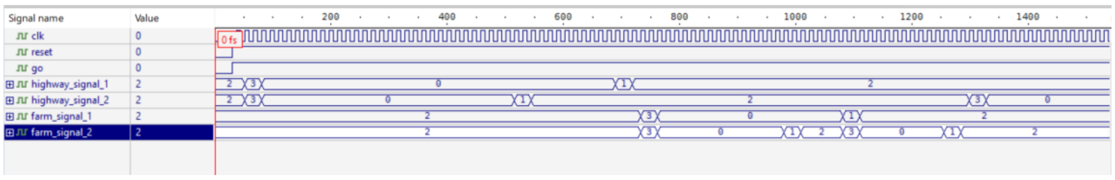


```
forever begin
  #5;
  clk = ~clk;
 end
end

initial begin
 #30;
 $display("Time S0: %t,GO = %d , highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,go,highway_signal_1, highway_signal_2, far
 #10;
 $display("Time S1: %t,GO = %d , highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,go,highway_signal_1, highway_signal_2, far
 #20;
 go=0;
 $display("Time S2: %t,GO = %d , highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,go,highway_signal_1, highway_signal_2, far
 #300;
 $display("Time S3: %t,GO = %d , highwaySignal1: %b, highwaySignal2: %b, farmSignal1: %b, farmSignal2: %b", $time,go,highway_signal_1, highway_signal_2, far
 #20;
 #400 $finish;
```

```
# SLP: Finished : 0.5 [s]
# SLP: 0 primitives and 4 (100.00%) other processes in SLP
# SLP: 17 (100.00%) signals in SLP and 0 interface signals
# ELAB2: Elaboration final pass complete - time: 0.5 [s].
# KERNEL: SLP loading done - time: 0.0 [s].
# KERNEL: Warning: You are using the Active-HDL Student Edition. The performance of simulation is running at a reduced rate.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4699 kB (elbread=427 elab2=4138 kernel=134 sdf=0)
# KERNEL: ASDB file was created in location D:\verilog hdl\Project\P\src\wave.asdb
#  9:18 PM, Friday, January 27, 2023
#  Simulation has been initialized
run
# KERNEL: Time S0:                    30,GO = 1 , highwaySignal1: 10, highwaySignal2: 10, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time S1:                    40,GO = 1 , highwaySignal1: 11, highwaySignal2: 11, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time S2:                    60,GO = 0 , highwaySignal1: 00, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# KERNEL: Time S3:                   360,GO = 0 , highwaySignal1: 00, highwaySignal2: 00, farmSignal1: 10, farmSignal2: 10
# RUNTIME: Info: RUNTIME_0068 project_1201766.v (410): $finish called.
```

Figure 10:Test Go

Simulation Waveform for the traffic light controller



Conclusion and Future works

In conclusion, we successfully design a solution to real problem which is the traffic light design for two roads. The design implements a finite state machine to control the switching of the traffic lights . The code was verified and tested using simulation tools and was found to be working as expected.

Future work for this project could include implementing additional features such as A sensor system to detect the number of vehicles on each road and adjust the traffic light timing accordingly. This would reduce wait times for vehicles and improve traffic flow. The code could also be optimized for better performance and to reduce the number of states in the finite state machine.