TRAFFIC LIGHT CONTROLLER

Introduction:

Modelled in Verilog, traffic light controller synthesised is functional to operate at junction of highway and a country road. A camera is installed at country road to detect presence of vehicles, and will provide sensor input to traffic light controller which operates accordingly.

Explanation:

This model has two parts, first it detects vehicles from the camera installed using haar_cascade and openCv in python. This can be used to give sensor input to the module designed in Verilog to work accordingly.

Coming to the traffic light controller, it gives highest priority to Highway, takes input as clock, reset_button, and sensor. Highway light and country light are the outputs.

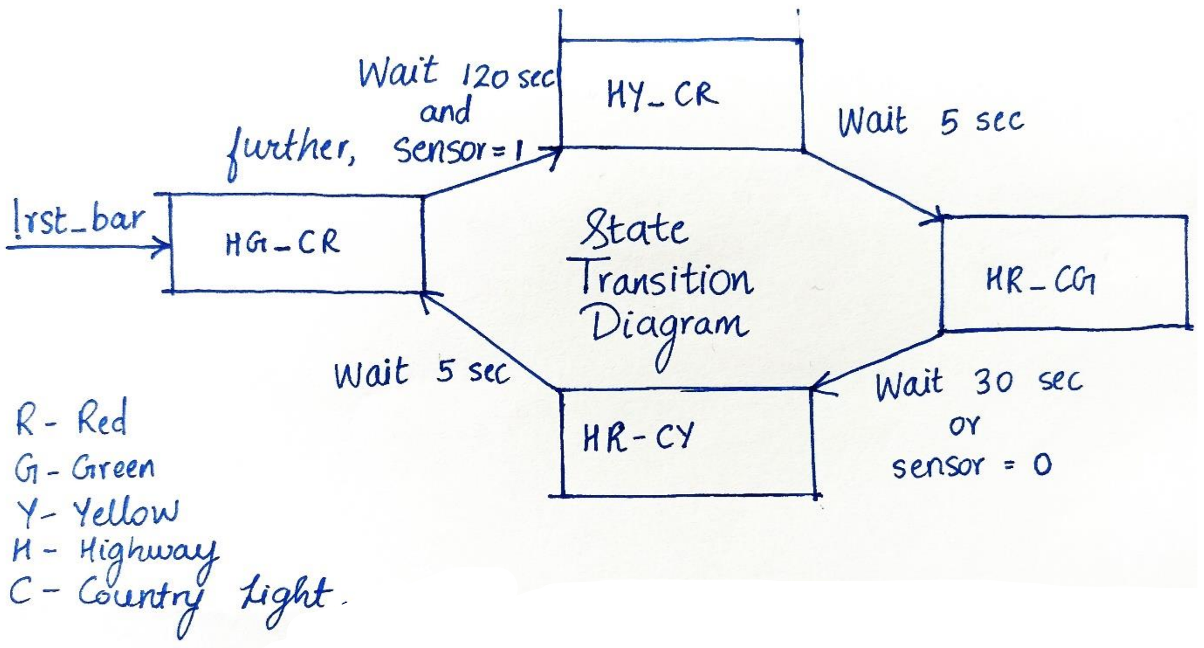
- reset_button – we are using an active low reset button, as soon as the reset button is active highway light is turned green and country light is turned red.
- Sensor – sensor is high when it detects the vehicles at country road, else it remains low
- Clock – we are using a 50 Hz (20 ms) clock

In our model there are four possible states –

- HG_CR – highway light is green, and country light is red
- HY_CR – highway light is yellow, and country light is red
- HR_CG – highway light is red, and country light is green
- HR_CY – highway light is red, and country light is yellow

state HG_CR is default state or state with highest priority.

STATE TRANSITION DIAGRAM

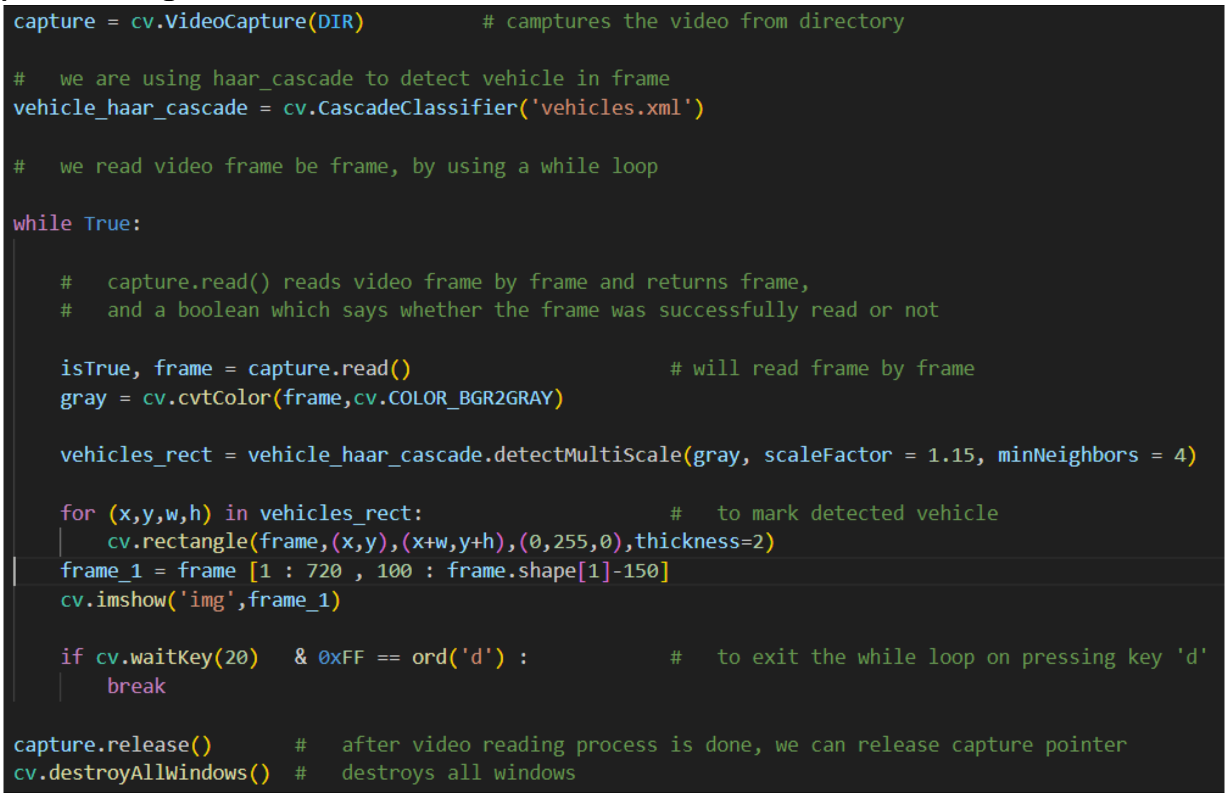


We will start with present state as HG_CR. It will remain at HG_CR until sensor becomes high illustrating vehicles at country road, which turns the state to HY_CR, this state is persisted for five seconds letting the vehicles at highway to stop for awaiting red light at highway. Then as soon as five seconds are over, we move to HR_CG. Here comes the interesting part, this state can be persisted for maximum of thirty seconds, we move to next state as soon as thirty seconds are over or sensor goes down, whatever happens first giving priority to highway. If the sensor goes down before thirty seconds or thirty seconds are over, we move to HR_CY. Yellow light of country side is onn, again we will remain at this state for five seconds, letting the vehicles settle.

Now imagine a situation, when we are at HR_CG, thirty seconds are over, but all the vehicles haven't passed that implies sensor will be high. We will move to HR_CY, then to HG_CR with sensor high. In this situation, giving priority to highway, highway light will be green for minimum of two minutes, irrespective of sensor.

Code Structure

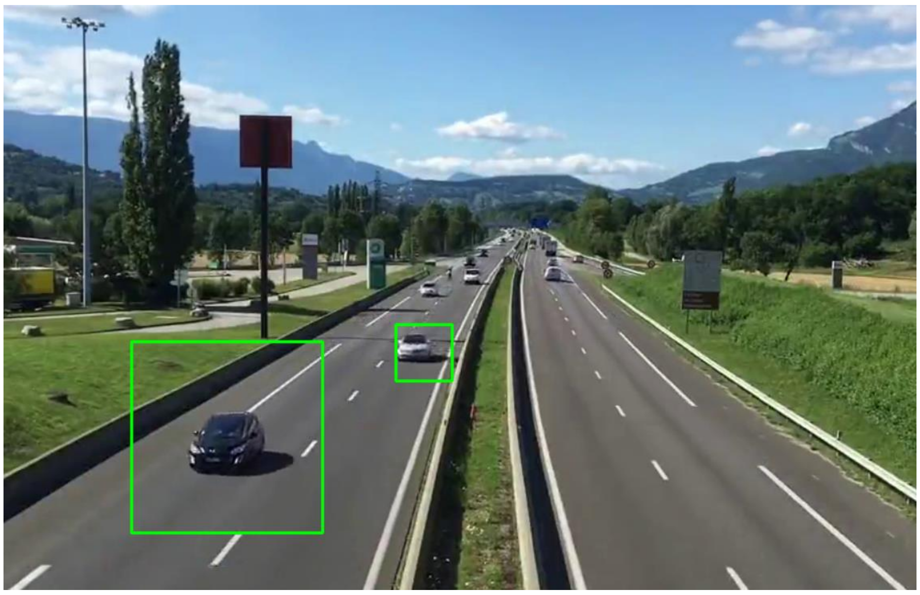
1) Detecting the vehicles



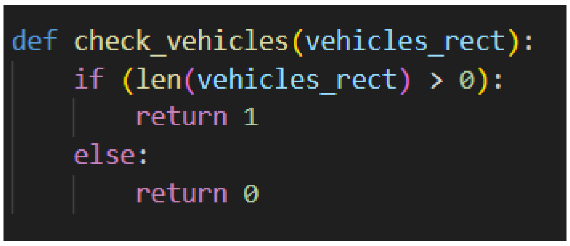
```
capture = cv.VideoCapture(DIR)          # camptures the video from directory

#   we are using haar_cascade to detect vehicle in frame
vehicle_haar_cascade = cv.CascadeClassifier('vehicles.xml')

#   we read video frame be frame, by using a while loop

while True:

    #   capture.read() reads video frame by frame and returns frame,
    #   and a boolean which says whether the frame was successfully read or not

    isTrue, frame = capture.read()                  # will read frame by frame
    gray = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)

    vehicles_rect = vehicle_haar_cascade.detectMultiScale(gray, scaleFactor = 1.15, minNeighbors = 4)

    for (x,y,w,h) in vehicles_rect:                 #   to mark detected vehicle
        cv.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),thickness=2)
    frame_1 = frame [1 : 720 , 100 : frame.shape[1]-150]
    cv.imshow('img',frame_1)

    if cv.waitKey(20)  & 0xFF == ord('d') :         #   to exit the while loop on pressing key 'd'
        break

capture.release()       #   after video reading process is done, we can release capture pointer
cv.destroyAllWindows()  #   destroys all windows
```

By using OpenCV and haar_cascade we can detect vehicles from a video frame by frame. To detect vehicles using haar_casacade we need to get the frame convert it into grayscale, and then use file 'vehicle.xml' to detect the vehicles. We get the coordinates of vehicles detected in a list 'vehicles_rect'. This is a list of lists, where number of elements in list gives number of vehicles detected in present frame, and each elemental list give rectangular coordinates of vehicles detected.

Using rectangular coordinates we can draw rectangles at those coordinates, using built in OpenCV functions.



Now a bool function can be created at number of elements in vehicles_rect, which can be fed as sensor input to traffic_light_controller module.

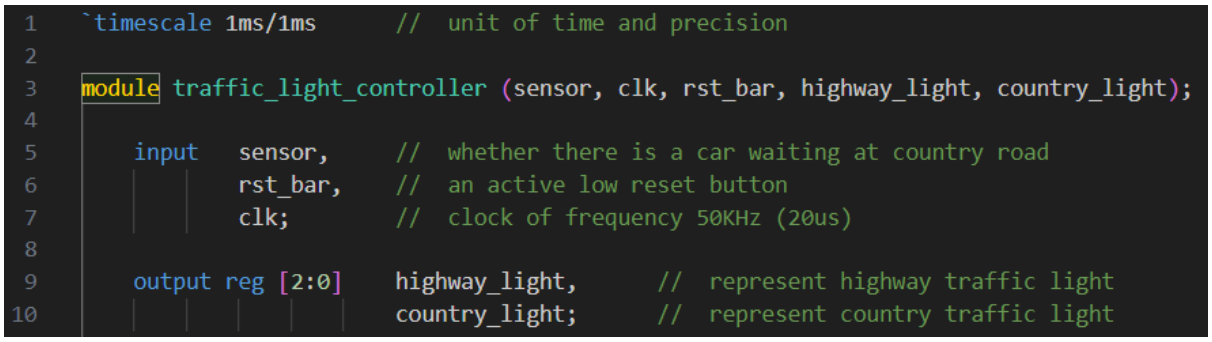


```python
def check_vehicles(vehicles_rect):
    if (len(vehicles_rect) > 0):
        return 1
    else:
        return 0
```

2) traffic_light_controller module

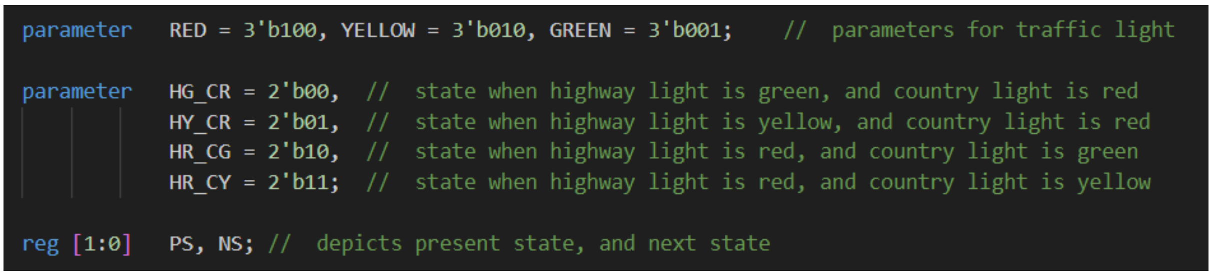
- Input and Output ports

Timescale sets units of time as 1ms with precision of 1ms.
Module is made with name 'traffic_light_controller', and input & output ports are declared first



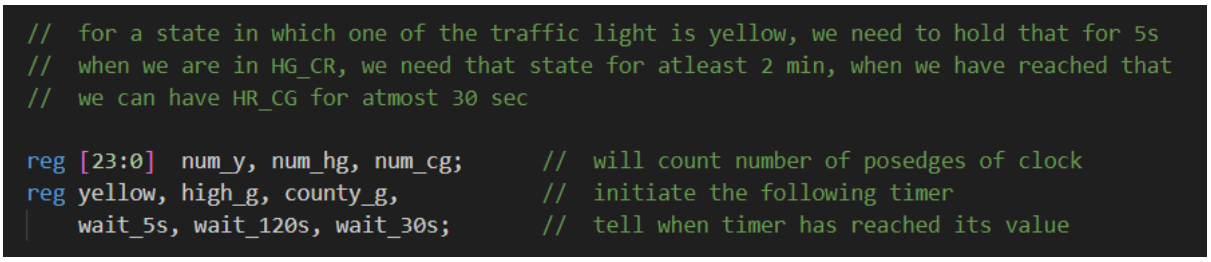
```verilog
`timescale 1ms/1ms      //  unit of time and precision

module traffic_light_controller (sensor, clk, rst_bar, highway_light, country_light);

    input   sensor,     //  whether there is a car waiting at country road
            rst_bar,    //  an active low reset button
            clk;        //  clock of frequency 50KHz (20us)

    output reg [2:0]    highway_light,      //  represent highway traffic light
                        country_light;      //  represent country traffic light
```

- Parameters
  Parameters for traffic light and state, as depicted earlier along with register for present state, and next state.

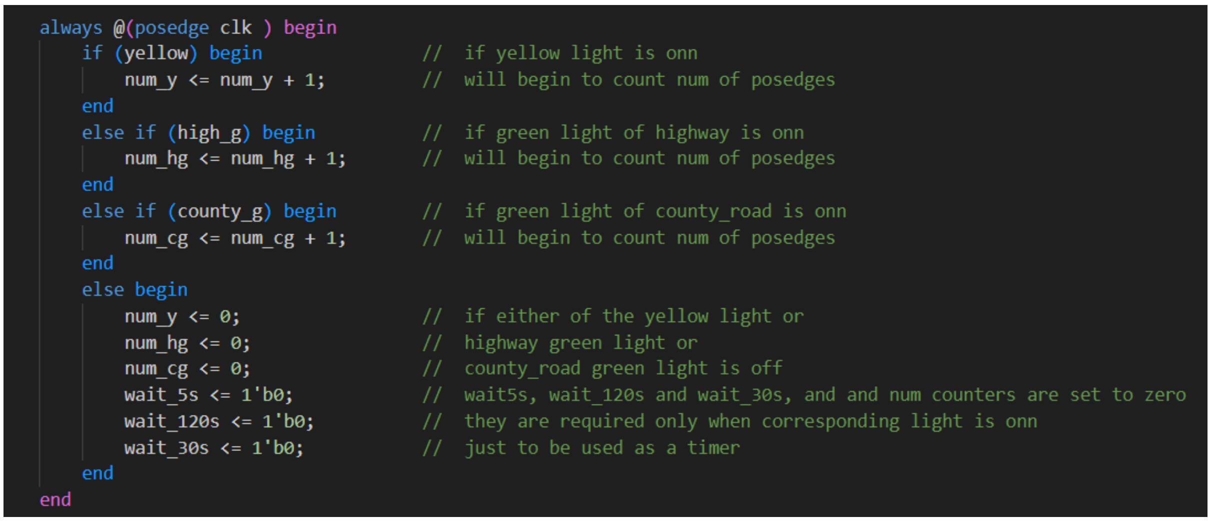

```
parameter   RED = 3'b100, YELLOW = 3'b010, GREEN = 3'b001;    //  parameters for traffic light

parameter   HG_CR = 2'b00,  //  state when highway light is green, and country light is red
            HY_CR = 2'b01,  //  state when highway light is yellow, and country light is red
            HR_CG = 2'b10,  //  state when highway light is red, and country light is green
            HR_CY = 2'b11;  //  state when highway light is red, and country light is yellow

reg [1:0]   PS, NS; //  depicts present state, and next state
```

- Timers

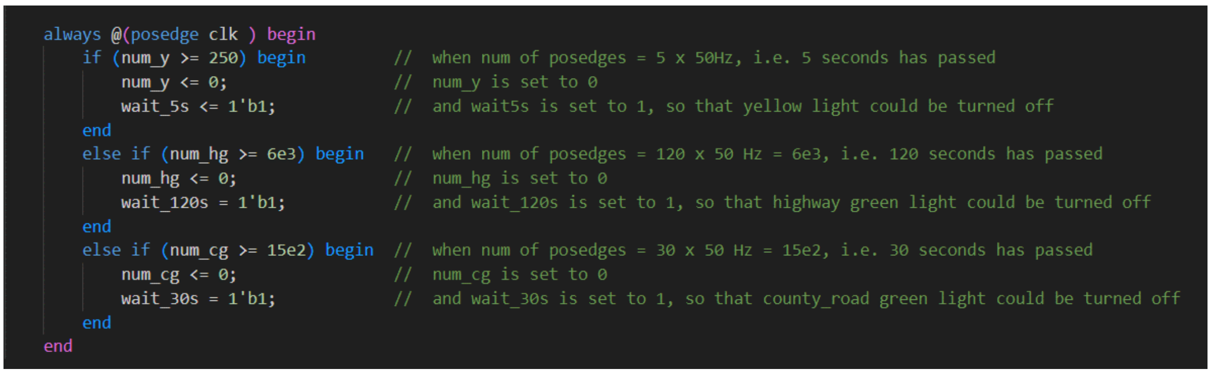
Registers for counting, basically for timers of 5, 30 and 120 seconds


```
//  for a state in which one of the traffic light is yellow, we need to hold that for 5s
//  when we are in HG_CR, we need that state for atleast 2 min, when we have reached that
//  we can have HR_CG for atmost 30 sec

reg [23:0]  num_y, num_hg, num_cg;      //  will count number of posedges of clock
reg yellow, high_g, county_g,           //  initiate the following timer
    wait_5s, wait_120s, wait_30s;       //  tell when timer has reached its value
```

- Count the number of posedges of clock, based on which at which state the module is in

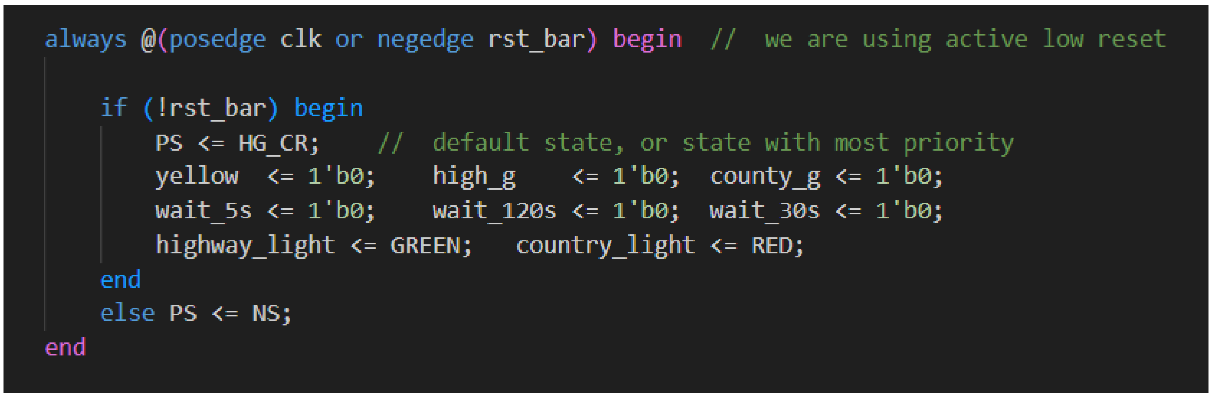

```
always @(posedge clk ) begin
    if (yellow) begin               //  if yellow light is onn
        num_y <= num_y + 1;         //  will begin to count num of posedges
    end
    else if (high_g) begin          //  if green light of highway is onn
        num_hg <= num_hg + 1;       //  will begin to count num of posedges
    end
    else if (county_g) begin        //  if green light of county_road is onn
        num_cg <= num_cg + 1;       //  will begin to count num of posedges
    end
    else begin
        num_y <= 0;                 //  if either of the yellow light or
        num_hg <= 0;                //  highway green light or
        num_cg <= 0;                //  county_road green light is off
        wait_5s <= 1'b0;            //  wait5s, wait_120s and wait_30s, and and num counters are set to zero
        wait_120s <= 1'b0;          //  they are required only when corresponding light is onn
        wait_30s <= 1'b0;           //  just to be used as a timer
    end
end
```

Conditions to set the timer based on number of posedges counted


```
always @(posedge clk ) begin
    if (num_y >= 250) begin         //  when num of posedges = 5 x 50Hz, i.e. 5 seconds has passed
        num_y <= 0;                 //  num_y is set to 0
        wait_5s <= 1'b1;            //  and wait5s is set to 1, so that yellow light could be turned off
    end
    else if (num_hg >= 6e3) begin   //  when num of posedges = 120 x 50 Hz = 6e3, i.e. 120 seconds has passed
        num_hg <= 0;                //  num_hg is set to 0
        wait_120s = 1'b1;           //  and wait_120s is set to 1, so that highway green light could be turned off
    end
    else if (num_cg >= 15e2) begin  //  when num of posedges = 30 x 50 Hz = 15e2, i.e. 30 seconds has passed
        num_cg <= 0;                //  num_cg is set to 0
        wait_30s = 1'b1;            //  and wait_30s is set to 1, so that county_road green light could be turned off
    end
end
```

- Reset button

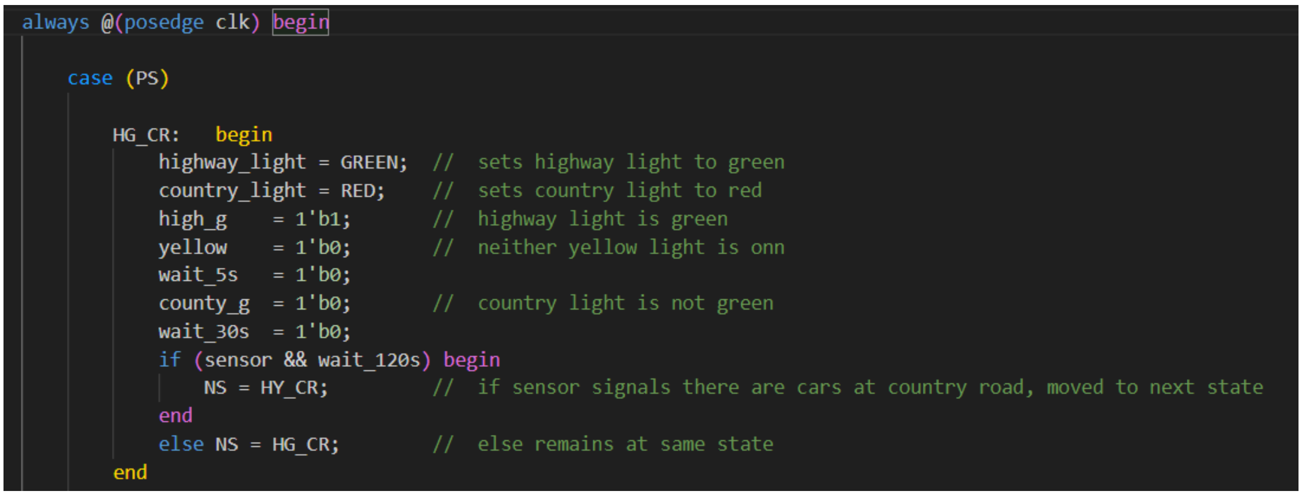
Reset function, reset is active low, so whenever negative edge of rst_bar comes and rst_bar is low it moves to HG_CR, reseting all timer registers to 0, and turning highway light to green, country light to red.



```
always @(posedge clk or negedge rst_bar) begin  //  we are using active low reset

    if (!rst_bar) begin
        PS <= HG_CR;    //  default state, or state with most priority
        yellow  <= 1'b0;    high_g    <= 1'b0;  county_g <= 1'b0;
        wait_5s <= 1'b0;    wait_120s <= 1'b0;  wait_30s <= 1'b0;
        highway_light <= GREEN;   country_light <= RED;
    end
    else PS <= NS;
end
```

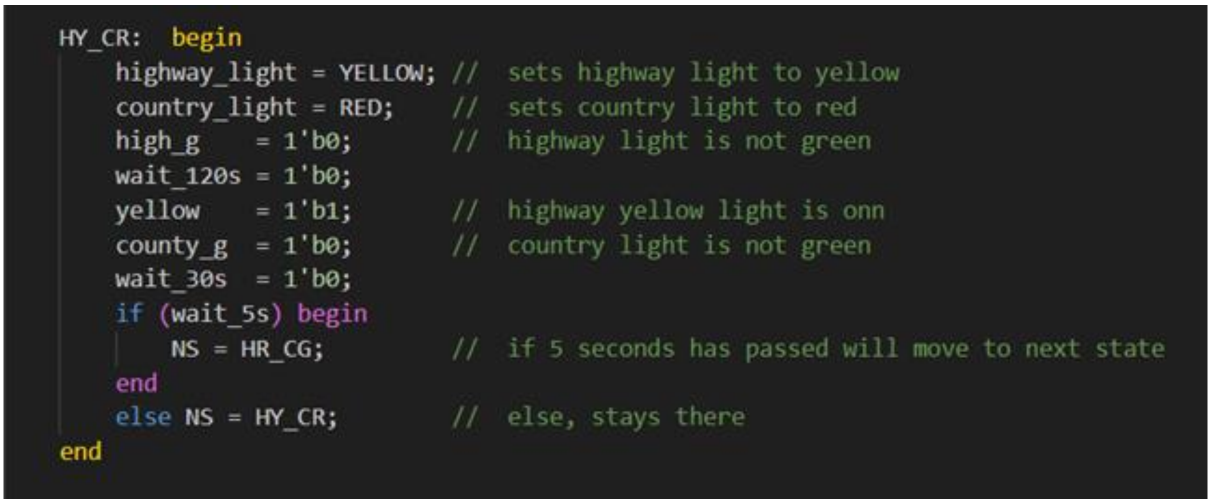
- switch case

Moves to next state HY_CR when both wait_120s and sensor are high



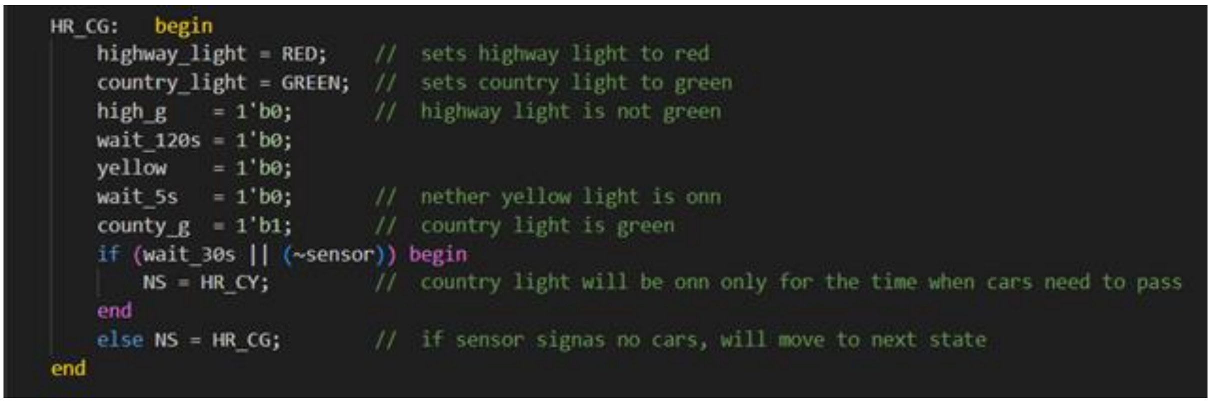
```
always @(posedge clk) begin

    case (PS)

        HG_CR:   begin
            highway_light = GREEN;  //  sets highway light to green
            country_light = RED;    //  sets country light to red
            high_g    = 1'b1;       //  highway light is green
            yellow    = 1'b0;       //  neither yellow light is onn
            wait_5s   = 1'b0;
            county_g  = 1'b0;       //  country light is not green
            wait_30s  = 1'b0;
            if (sensor && wait_120s) begin
                NS = HY_CR;         //  if sensor signals there are cars at country road, moved to next state
            end
            else NS = HG_CR;        //  else remains at same state
        end
```

Moves to next state HR_CG after waiting for 5 seconds



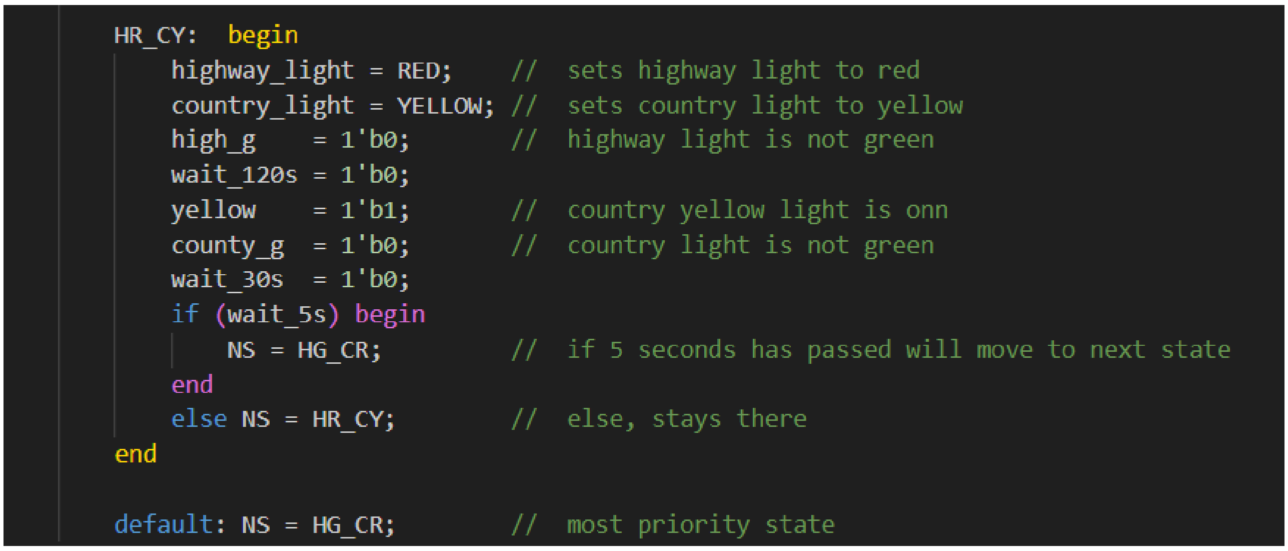
```
HY_CR:  begin
    highway_light = YELLOW; //  sets highway light to yellow
    country_light = RED;    //  sets country light to red
    high_g    = 1'b0;       //  highway light is not green
    wait_120s = 1'b0;
    yellow    = 1'b1;       //  highway yellow light is onn
    county_g  = 1'b0;       //  country light is not green
    wait_30s  = 1'b0;
    if (wait_5s) begin
        NS = HR_CG;         //  if 5 seconds has passed will move to next state
    end
    else NS = HY_CR;        //  else, stays there
end
```

Moves to next state when either sensor is low, or wait_30s is high



```
HR_CG:   begin
    highway_light = RED;    //  sets highway light to red
    country_light = GREEN;  //  sets country light to green
    high_g     = 1'b0;      //  highway light is not green
    wait_120s = 1'b0;
    yellow     = 1'b0;
    wait_5s    = 1'b0;      //  nether yellow light is onn
    county_g   = 1'b1;      //  country light is green
    if (wait_30s || (~sensor)) begin
        NS = HR_CY;         //  country light will be onn only for the time when cars need to pass
    end
    else NS = HR_CG;        //  if sensor signas no cars, will move to next state
end
```

Moves to next state after waiting for 5 seconds, and default is HG_CR

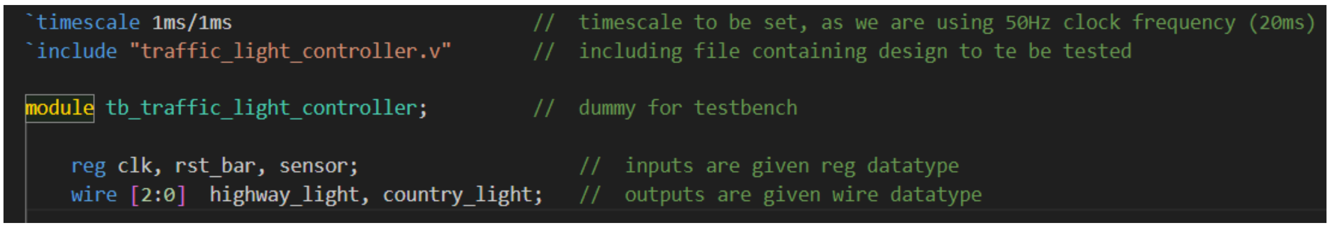


```
HR_CY:  begin
    highway_light = RED;    //  sets highway light to red
    country_light = YELLOW; //  sets country light to yellow
    high_g     = 1'b0;      //  highway light is not green
    wait_120s = 1'b0;
    yellow     = 1'b1;      //  country yellow light is onn
    county_g   = 1'b0;      //  country light is not green
    wait_30s   = 1'b0;
    if (wait_5s) begin
        NS = HG_CR;         //  if 5 seconds has passed will move to next state
    end
    else NS = HR_CY;        //  else, stays there
end

default: NS = HG_CR;        //  most priority state
```

3) TestBench

- Input and Output ports

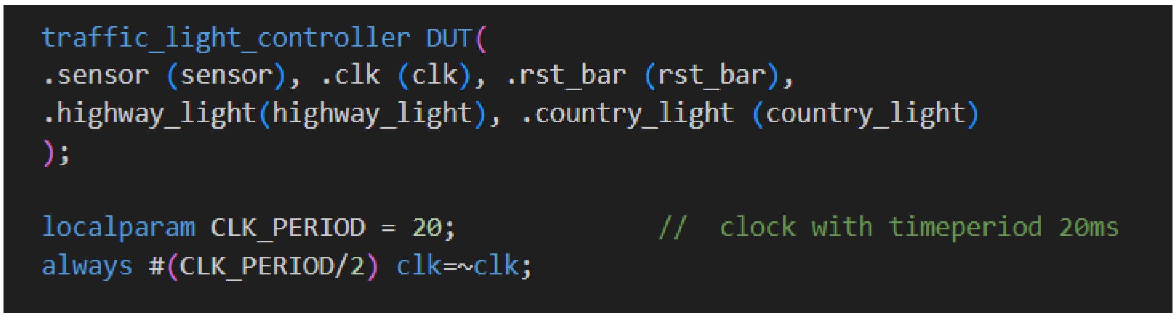
Setting the timescale, including the file containing design under test
Input ports are to be declared reg type and output as wire



```
`timescale 1ms/1ms                    //  timescale to be set, as we are using 50Hz clock frequency (20ms)
`include "traffic_light_controller.v"    //  including file containing design to te be tested

module tb_traffic_light_controller;       //  dummy for testbench

    reg clk, rst_bar, sensor;                //  inputs are given reg datatype
    wire [2:0]  highway_light, country_light;   //  outputs are given wire datatype
```

- Module instantiation & clock

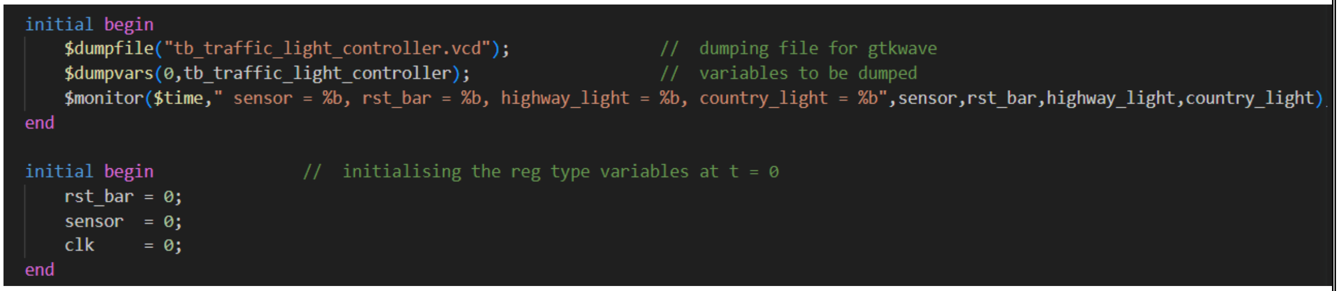
  Module is instantiated with name ‘DUT’, input and output parameters are fed. We are using clock of 50Hz, hence declared using localparam



```verilog
traffic_light_controller DUT(
.sensor (sensor), .clk (clk), .rst_bar (rst_bar),
.highway_light(highway_light), .country_light (country_light)
);

localparam CLK_PERIOD = 20;            //  clock with timeperiod 20ms
always #(CLK_PERIOD/2) clk=~clk;
```

- Initialising input values & dumping

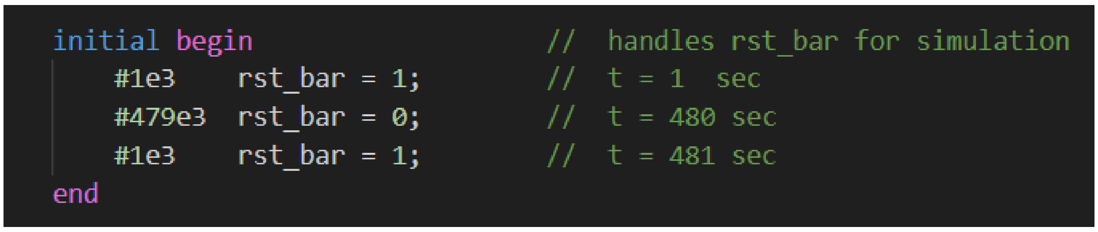
Dumping the file to '.vcd' and variables to monitor using dumpvars. Monitor to see the variables at terminal and all the input values are set to 0, at t = 0.



```verilog
initial begin
    $dumpfile("tb_traffic_light_controller.vcd");             //  dumping file for gtkwave
    $dumpvars(0,tb_traffic_light_controller);                 //  variables to be dumped
    $monitor($time," sensor = %b, rst_bar = %b, highway_light = %b, country_light = %b",sensor,rst_bar,highway_light,country_light)
end

initial begin               //  initialising the reg type variables at t = 0
    rst_bar = 0;
    sensor  = 0;
    clk     = 0;
end
```

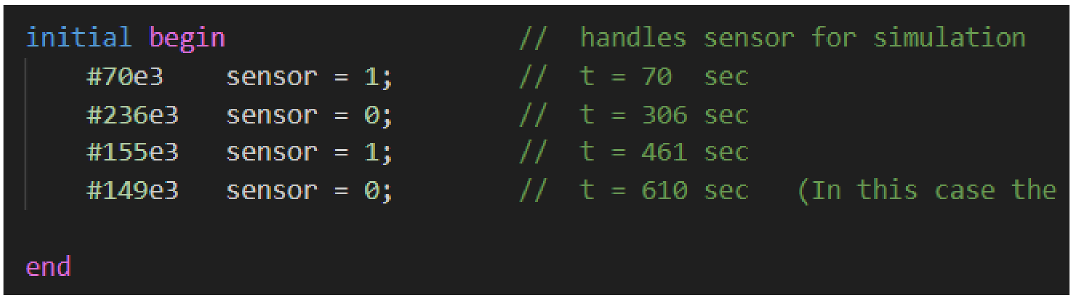
- Reset button

  Reset button becomes active at time t = 480 s, else inactive



```verilog
initial begin                       //  handles rst_bar for simulation
    #1e3    rst_bar = 1;            //  t = 1  sec
    #479e3  rst_bar = 0;            //  t = 480 sec
    #1e3    rst_bar = 1;            //  t = 481 sec
end
```

- Sensor



```verilog
initial begin                       //  handles sensor for simulation
    #70e3    sensor = 1;            //  t = 70  sec
    #236e3   sensor = 0;            //  t = 306 sec
    #155e3   sensor = 1;            //  t = 461 sec
    #149e3   sensor = 0;            //  t = 610 sec   (In this case the

end
```

Then finished using $finish

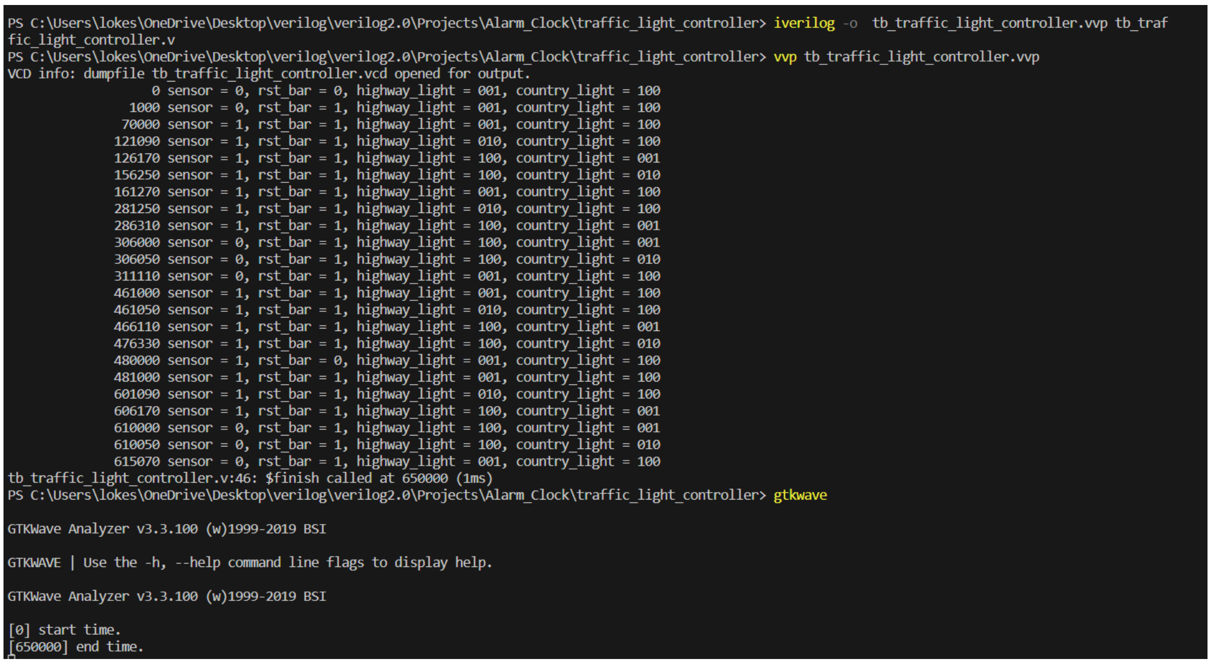
4) Synthesising and Output

Synthesised using command   – iverilog -o <destination_file.vvp> <source_file.v>

After executing this command, a .vvp file is synthesised along with a .vcd file with file name as in $dumpfile()

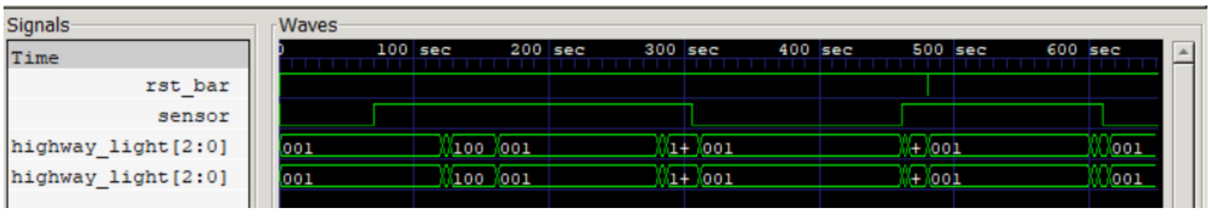
Then accessed using command – vvp <destination_file.vvp>

Then the following is displayed at terminal



```
PS C:\Users\lokes\OneDrive\Desktop\verilog\verilog2.0\Projects\Alarm_Clock\traffic_light_controller> iverilog -o  tb_traffic_light_controller.vvp tb_traf
fic_light_controller.v
PS C:\Users\lokes\OneDrive\Desktop\verilog\verilog2.0\Projects\Alarm_Clock\traffic_light_controller> vvp tb_traffic_light_controller.vvp
VCD info: dumpfile tb_traffic_light_controller.vcd opened for output.
                   0 sensor = 0, rst_bar = 0, highway_light = 001, country_light = 100
                1000 sensor = 0, rst_bar = 1, highway_light = 001, country_light = 100
               70000 sensor = 1, rst_bar = 1, highway_light = 001, country_light = 100
              121090 sensor = 1, rst_bar = 1, highway_light = 010, country_light = 100
              126170 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 001
              156250 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 010
              161270 sensor = 1, rst_bar = 1, highway_light = 001, country_light = 100
              281250 sensor = 1, rst_bar = 1, highway_light = 010, country_light = 100
              286310 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 001
              306000 sensor = 0, rst_bar = 1, highway_light = 100, country_light = 001
              306050 sensor = 0, rst_bar = 1, highway_light = 100, country_light = 010
              311110 sensor = 0, rst_bar = 1, highway_light = 001, country_light = 100
              461000 sensor = 1, rst_bar = 1, highway_light = 001, country_light = 100
              461050 sensor = 1, rst_bar = 1, highway_light = 010, country_light = 100
              466110 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 001
              476330 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 010
              480000 sensor = 1, rst_bar = 0, highway_light = 001, country_light = 100
              481000 sensor = 1, rst_bar = 1, highway_light = 001, country_light = 100
              601090 sensor = 1, rst_bar = 1, highway_light = 010, country_light = 100
              606170 sensor = 1, rst_bar = 1, highway_light = 100, country_light = 001
              610000 sensor = 0, rst_bar = 1, highway_light = 100, country_light = 001
              610050 sensor = 0, rst_bar = 1, highway_light = 100, country_light = 010
              615070 sensor = 0, rst_bar = 1, highway_light = 001, country_light = 100
tb_traffic_light_controller.v:46: $finish called at 650000 (1ms)
PS C:\Users\lokes\OneDrive\Desktop\verilog\verilog2.0\Projects\Alarm_Clock\traffic_light_controller> gtkwave

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

GTKWAVE | Use the -h, --help command line flags to display help.

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[650000] end time.
```

Output matched with the expected output. This output can also be realised using gtkwave



Conclusion

The traffic light controller for a highway and country road needs to efficiently manage the flow of traffic, taking into account factors such as traffic volume, priority rules, and safety considerations. The Verilog code for the controller would involve defining the behaviour and interaction of these components, as well as the decision-making logic for determining when to change the traffic lights.

Here, we have used it to define the various components of the system, including the traffic lights, sensors, timers, and control logic.