Abstract

This report presents the design and implementation of a traffic light controller system using Verilog, a hardware description language (HDL). The primary objective is to develop an efficient and reliable traffic management system for controlling traffic signals at an intersection, optimizing traffic flow while ensuring safety. The system operates on a finite state machine (FSM) model, where different traffic light states (green, yellow, and red) are activated based on the timing and sensor inputs. The design is aimed at enhancing traffic signal coordination while considering factors such as pedestrian safety and vehicular congestion. Verilog is used to describe the hardware behavior, simulating and synthesizing the controller onto an FPGA platform. This approach demonstrates the potential of digital logic design in solving real-world traffic management challenges.

Introduction

In modern cities, traffic congestion and accidents at intersections are significant concerns that necessitate the implementation of efficient traffic light control systems. A well-designed traffic light controller is essential for maintaining traffic flow, reducing accidents, and enhancing road safety. The objective of this project is to design a traffic light controller using Verilog, a hardware description language, to simulate and implement a digital logic-based system capable of managing traffic signals for both vehicles and pedestrians at a busy intersection.

The controller is designed using a finite state machine (FSM) approach, where the system transitions through predefined states corresponding to the green, yellow, and red lights for each direction. Timing sequences for each light are based on real-world traffic patterns, and the controller ensures that vehicles and pedestrians are efficiently managed. Additionally, the system can adapt to various traffic conditions by incorporating sensor inputs that detect the presence of vehicles, enabling dynamic adjustment of the light duration.

Verilog, a powerful tool for designing digital circuits, allows for precise modeling, simulation, and verification of the traffic light controller before deployment. This report will describe the design, implementation, and testing of the system, outlining how Verilog code is structured to simulate the traffic light controller and its operation. The proposed design demonstrates the application of finite state machines and digital logic design principles in solving practical, real-world problems.

System and design

The traffic light controller system is designed to control the operation of traffic lights at an intersection. It utilizes a finite state machine (FSM) to manage the transition between different light states (Red, Green, Yellow) for two directions (e.g., North-South and East-West). The system is implemented using Verilog, a hardware description language, to describe the functionality of the controller.

Features:

FSM-Based Design: The traffic light system uses a finite state machine to ensure the lights change in a timed and orderly manner.

Multiple States: The system supports multiple states like Red, Green, Yellow for both directions of traffic.

Timer Control: Timers are used to control the duration for which each light stays in a given state.

Pedestrian Button: Optionally, a button for pedestrians can be integrated to switch to a "Walk" signal when pressed.

Inputs:

Clock Signal (clk): The clock signal that drives the FSM state transitions.

Reset (rst): A reset signal to initialize the system to a known state.

NS_Red, NS_Green, NS_Yellow: Output signals for North-South direction indicating the status of the traffic light.

EW_Red, EW_Green, EW_Yellow: Output signals for East-West direction indicating the status of the traffic light.

State Machine Design:

The controller is based on a finite state machine that transitions between states based on clock cycles. Below are the primary states of the FSM:

NS_Green_EW_Red: North-South traffic moves (green), East-West traffic is stopped (red).

NS_Yellow_EW_Red: North-South traffic is preparing to stop (yellow), East-West traffic remains stopped (red).

NS_Red_EW_Green: East-West traffic moves (green), North-South traffic is stopped (red).

NS_Red_EW_Yellow: East-West traffic is preparing to stop (yellow), North-South traffic remains stopped (red).

The transitions between these states are controlled by a timer that dictates how long each direction's light remains in a given state (green, yellow, or red).

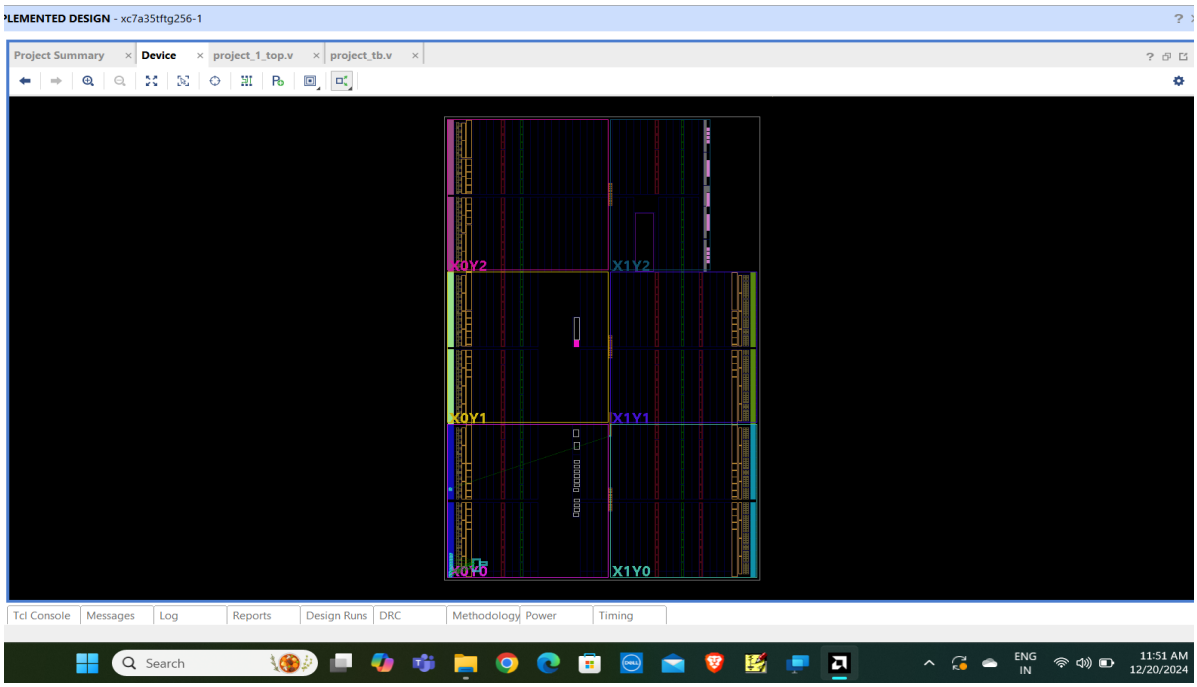
Timing Control:

Green Light Duration: Controlled by a timer. The system keeps the green light on for a predefined period (e.g., 30 seconds for North-South and 30 seconds for East-West).

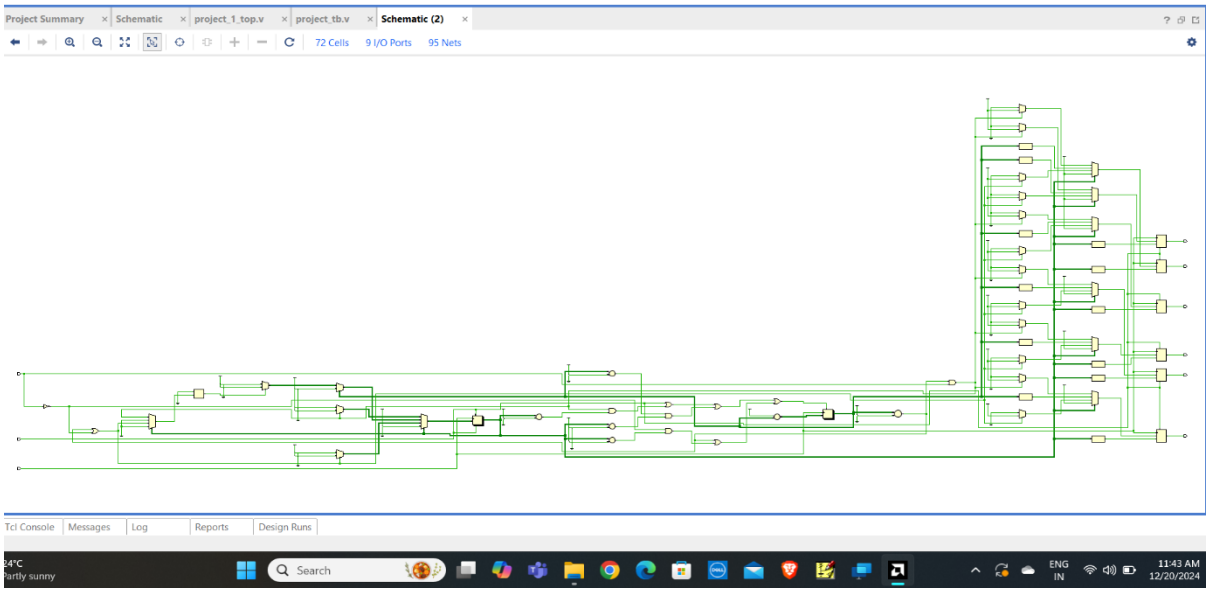
Yellow Light Duration: The yellow light is active for a short duration (e.g., 5 seconds).

Red Light Duration: Red lights are typically held for the remaining time until the next light transition, depending on the system's design.

Device:



SCHEMATIC DIAGRAM



VERILOG CODE

`timescale 1ns / 1ps

```
module project_1_top(

//Global signals
   input wire i_clk_100M     ,      //Clock -100Mhz
   input wire i_rst_p        ,      //Reset -Active high

   //Input
   input wire i_EW_vd        ,      //east west direction vehicle detection signal

   //Output
   output wire o_NS_red      ,      //NS RED
   output wire o_NS_yellow   ,      //NS yellow
   output wire o_NS_green    ,      //NS  green
   output wire o_EW_red      ,      //NS RED
   output wire o_EW_yellow   ,      //NS yellow
   output wire o_EW_green           //NS  green

);

//State Declaration
localparam [2:0] NS_GREEN =3'b001,
                 YELLOW   =3'b010,
                 EW_GREEN =3'b100;

//Internal Regs/wires
reg[2:0]state  ,next_state            ;
reg     r_ns_red   , r_next_ns_red    ;
```

```
reg     r_ns_yellow, r_next_ns_yellow   ;
reg     r_ns_green , r_next_ns_green    ;
reg     r_ew_red   , r_next_ew_red      ;
reg     r_ew_yellow, r_next_ew_yellow   ;
reg     r_ew_green , r_next_ew_green    ;
reg[9:0]counter                         ;
reg     ns_to_ew                        ;

//Update next state (clock)
always@(posedge i_clk_100M) begin

   if(i_rst_p)begin

      r_ns_red    <= 1'b0;
      r_ns_yellow <= 1'b0;
      r_ns_green  <= 1'b1;
      r_ew_red    <= 1'b1;
      r_ew_yellow <= 1'b0;
      r_ew_green  <= 1'b0;
      state       <=NS_GREEN;

    end else begin

      r_ns_red    <= r_next_ns_red      ;
      r_ns_yellow <= r_next_ns_yellow   ;
      r_ns_green  <= r_next_ns_green    ;
      r_ew_red    <= r_next_ew_red      ;
```

```
    r_ew_yellow <= r_next_ew_yellow   ;
    r_ew_green  <= r_next_ew_green    ;
    state       <= next_state         ;

  end

  //counter logic
  //RESET ||
  if(i_rst_p
    ||(counter >= 25 && i_EW_vd && state ==NS_GREEN)      // State change NS-EW
    ||(counter ==4 && state == YELLOW)                    //change from Yellow state
    ||(counter >=25|| !i_EW_vd && state == EW_GREEN) )    //State change EW-NS
    counter <=1'b0;
  else
    counter <=counter+1;

end

//State machine Logic  (combo)
always@(*) begin

  //Store to memory
  r_next_ns_red    =r_ns_red    ;
  r_next_ns_yellow =r_ns_yellow ;
  r_next_ns_green  =r_ns_green  ;
```

```
r_next_ew_red    =r_ew_red    ;
r_next_ew_yellow =r_ew_yellow ;
r_next_ew_green  =r_ew_green  ;
next_state       =state       ;

case(state)

   NS_GREEN:begin

      if(counter >= 25 && i_EW_vd) begin

         next_state=YELLOW ;
         ns_to_ew     =1'b0;


      end else begin
         r_next_ns_red    =1'b0;
         r_next_ns_yellow =1'b0;
         r_next_ns_green  =1'b1;
         r_next_ew_red    =1'b1;
         r_next_ew_yellow =1'b0;
         r_next_ew_green  =1'b0;
         next_state       =EW_GREEN;

      end

   end
```

```
YELLOW:begin

   if(counter==4) begin

      if(ns_to_ew)
         next_state =EW_GREEN ;
      else
         next_state =NS_GREEN ;

   end else begin

       r_next_ns_red    =1'b0;
       r_next_ns_yellow =1'b1;
       r_next_ns_green  =1'b0;
       r_next_ew_red    =1'b0;
       r_next_ew_yellow =1'b1;
       r_next_ew_green  =1'b0;
       next_state       =YELLOW;

   end

end

EW_GREEN:begin

   if(counter >= 25 || !i_EW_vd) begin
```

```
        next_state=YELLOW ;
        ns_to_ew  =1'b0   ;

      end else begin

        r_next_ns_red    =1'b1;
        r_next_ns_yellow =1'b0;
        r_next_ns_green  =1'b0;
        r_next_ew_red    =1'b0;
        r_next_ew_yellow =1'b0;
        r_next_ew_green  =1'b1;
        next_state       =NS_GREEN;

      end

end

default:begin
    r_next_ns_red    =1'b0;
    r_next_ns_yellow =1'b0;
    r_next_ns_green  =1'b1;
    r_next_ew_red    =1'b1;
    r_next_ew_yellow =1'b0;
    r_next_ew_green  =1'b0;
    next_state       =NS_GREEN;
```

```
      end

   endcase

end



//Output assignment
assign o_NS_red    =r_ns_red   ;
assign o_NS_yellow =r_ns_yellow;
assign o_NS_green  =r_ns_green ;
assign o_EW_red    =r_ew_red   ;
assign o_EW_yellow =r_ew_yellow;
assign o_EW_green  =r_ew_green ;


endmodule
```

TEST BENCH

To validate the functionality of the traffic light controller, a testbench is created to simulate the inputs and verify the output light states. This involves applying a clock signal, a reset to ensure proper transitions occur.

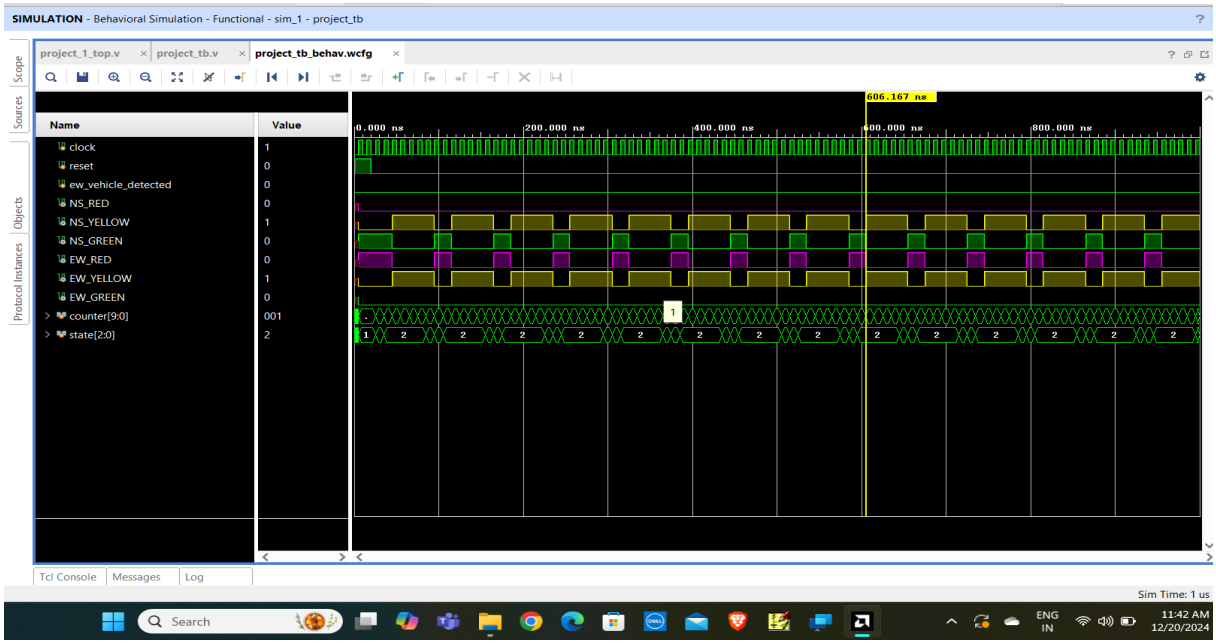
TEST BENCH CODE

```
`timescale 1ns / 1ps


module project_tb;


//Internal REgisters
```

```
reg clock;
reg reset;

reg ew_vehicle_detected;

wire NS_RED   ;
wire NS_YELLOW;
wire NS_GREEN ;
wire EW_RED   ;
wire EW_YELLOW;
wire EW_GREEN ;

//gen Clk
always #5 clock =~clock;

//Logic block
initial begin

   clock=0;
   reset=1;
   ew_vehicle_detected=0;
   #20 reset=0;
   #1000 ew_vehicle_detected=1;
   #500 ew_vehicle_detected=0;
   #500 ew_vehicle_detected=1;
   #1000 ew_vehicle_detected=0;
```

```
    #5000 $stop;

end



//Instantiations
project_1_top DUT(

    .i_clk_100M    (clock),
    .i_rst_p       (reset),
    .i_EW_vd       (ew_vehicle_detected),
    .o_NS_red      (NS_RED),
    .o_NS_yellow   (NS_YELLOW),
    .o_NS_green    (NS_GREEN),
    .o_EW_red      (EW_RED),
    .o_EW_yellow   (EW_YELLOW),
    .o_EW_green    (EW_GREEN)
);
endmodule
```

OUTPUT WAVEWFORMS



CONCLUSION

This Verilog-based traffic light controller design implements an FSM to handle the state transitions of traffic lights for two directions, with timing controls for each light. It is scalable for more complex intersections and can be further enhanced by adding features like pedestrian signals or additional sensors. The FSM approach ensures a clear and manageable design for simulation and hardware implementation