TRAFFIC LIGHT CONTROLLER
USING
FINITE STATE MACHINE

EECT / CE 6325 – VLSI DESIGN

Instructor

Prof. Carl Sechen

Submitted by

Ganesh Manjunath

Net ID: dal542146

&

Aryan Verma

Net ID: dal820508

Date: 09/24/2025

INTRODUCTION

Traffic management is a crucial aspect of modern day infrastructure, ensuring the safe and efficient movement of vehicles and pedestrians. Old traffic light systems rely on fixed timers, which often fail to adapt to complex traffic conditions. With the growth of smart cities, there is a strong need for intelligent traffic control mechanisms that are reliable, scalable, and hardware efficient.

Finite State Machines (FSMs) provide an effective way to model sequential behavior in digital systems, making them well-suited for traffic signal controllers. By defining distinct states—such as green, yellow, and red phases for both North-South (NS) and East-West (EW) directions—an FSM can accurately represent real-world traffic flow transitions. Implementing such a design in Verilog allows for simulation, verification, and eventual hardware realization using FPGA or ASIC technology.

This project discusses designing a traffic light controller using Verilog HDL, based on the Finite state machine approach. The design not only elaborates the systematic state transitions of a basic controller but also integrates features like paarameterization, pulse-width modulation (PWM) for light intensity control, and scalability through different modules.. The work emphasizes hardware mapping & synthesizability, providing insight into practical VLSI design flow and digital system implementation.

Verilog Code

```
module traffic_light_controller #(
    parameter integer GREEN_TIME  = 3000,
    parameter integer YELLOW_TIME = 500,
    parameter integer CWIDTH      = 256,
    parameter integer UNIQUE_ID   = 0
)(
    input  wire clk,
    input  wire rst,
    input  wire sensor,
    output reg  NS_Red,
    output reg  NS_Yellow,
    output reg  NS_Green,
    output reg  EW_Red,
    output reg  EW_Yellow,
    output reg  EW_Green
);
    localparam [3:0] S_NS_G = 4'b0001,
                     S_NS_Y = 4'b0010,
                     S_EW_G = 4'b0100,
                     S_EW_Y = 4'b1000;
    reg [3:0] state, next_state;
    reg [CWIDTH-1:0] counter, next_counter;
    reg [7:0] unique_reg;
    always @(posedge clk) begin
        if (rst) unique_reg <= 8'b00000000;
        else unique_reg <= unique_reg + UNIQUE_ID;
```

```
    end
    always @(posedge clk) begin
        if (rst) begin
            state   <= S_NS_G;
            counter <= {CWIDTH{1'b0}};
        end else begin
            state   <= next_state;
            counter <= next_counter;
        end
    end
    always @* begin
        next_state   = state;
        next_counter = counter + {{(CWIDTH-1){1'b0}},1'b1};
        case (state)
            S_NS_G: if (counter >= GREEN_TIME-1)  begin next_state=S_NS_Y;
next_counter={CWIDTH{1'b0}}; end
            S_NS_Y: if (counter >= YELLOW_TIME-1) begin next_state=S_EW_G;
next_counter={CWIDTH{1'b0}}; end
            S_EW_G: if (counter >= GREEN_TIME-1)  begin next_state=S_EW_Y;
next_counter={CWIDTH{1'b0}}; end
            S_EW_Y: if (counter >= YELLOW_TIME-1) begin next_state=S_NS_G;
next_counter={CWIDTH{1'b0}}; end
            default: begin next_state=S_NS_G; next_counter={CWIDTH{1'b0}}; end
        endcase
    end
    always @* begin
        NS_Red=0; NS_Yellow=0; NS_Green=0;
        EW_Red=0; EW_Yellow=0; EW_Green=0;
```

```
        case (state)
            S_NS_G: begin NS_Green=1;  EW_Red=1;  end
            S_NS_Y: begin NS_Yellow=1; EW_Red=1;  end
            S_EW_G: begin EW_Green=1;  NS_Red=1;  end
            S_EW_Y: begin EW_Yellow=1; NS_Red=1;  end
            default: begin NS_Green=1; EW_Red=1;  end
        endcase
    end
endmodule
module pwm #(
    parameter integer BITS = 256
)(
    input  wire clk,
    input  wire rst,
    input  wire [BITS-1:0] duty,
    output reg  y
);
    reg [BITS-1:0] cnt;
    always @(posedge clk) begin
        if (rst) cnt <= {BITS{1'b0}};
        else     cnt <= cnt + {{(BITS-1){1'b0}},1'b1};
    end
    always @* y = (cnt < duty);
endmodule
module ctl_with_pwm #(
    parameter integer GREEN_TIME  = 3000,
    parameter integer YELLOW_TIME = 500,
```

```
    parameter integer CWIDTH      = 256,
    parameter integer PWM_BITS    = 256,
    parameter integer UNIQUE_ID   = 0
)(
    input  wire clk,
    input  wire rst,
    input  wire sensor,
    output wire NS_Red, NS_Yellow, NS_Green,
    output wire EW_Red, EW_Yellow, EW_Green
);
    wire nsr, nsy, nsg, ewr, ewy, ewg;
    traffic_light_controller #(
        .GREEN_TIME(GREEN_TIME), .YELLOW_TIME(YELLOW_TIME),
.CWIDTH(CWIDTH), .UNIQUE_ID(UNIQUE_ID)
    ) u_ctl (
        .clk(clk), .rst(rst), .sensor(sensor),
        .NS_Red(nsr), .NS_Yellow(nsy), .NS_Green(nsg),
        .EW_Red(ewr), .EW_Yellow(ewy), .EW_Green(ewg)
    );
    localparam [PWM_BITS-1:0] FULL = {PWM_BITS{1'b1}};
    pwm #(.BITS(PWM_BITS)) p0(.clk(clk), .rst(rst), .duty(nsr ? FULL : {PWM_BITS{1'b0}}),
.y(NS_Red));
    pwm #(.BITS(PWM_BITS)) p1(.clk(clk), .rst(rst), .duty(nsy ? FULL : {PWM_BITS{1'b0}}),
.y(NS_Yellow));
    pwm #(.BITS(PWM_BITS)) p2(.clk(clk), .rst(rst), .duty(nsg ? FULL : {PWM_BITS{1'b0}}),
.y(NS_Green));
    pwm #(.BITS(PWM_BITS)) p3(.clk(clk), .rst(rst), .duty(ewr ? FULL : {PWM_BITS{1'b0}}),
.y(EW_Red));
```

```
    pwm #(.BITS(PWM_BITS)) p4(.clk(clk), .rst(rst), .duty(ewy ? FULL : {PWM_BITS{1'b0}}),
.y(EW_Yellow));
    pwm #(.BITS(PWM_BITS)) p5(.clk(clk), .rst(rst), .duty(ewg ? FULL : {PWM_BITS{1'b0}}),
.y(EW_Green));
endmodule
module big_system #(
    parameter integer N             = 200,
    parameter integer GREEN_TIME    = 3000,
    parameter integer YELLOW_TIME   = 500,
    parameter integer CWIDTH        = 256,
    parameter integer PWM_BITS      = 256
)(
    input  wire               clk,
    input  wire               rst,
    input  wire [N-1:0]       sensor,
    output wire [N-1:0]       NS_Red, NS_Yellow, NS_Green,
    output wire [N-1:0]       EW_Red, EW_Yellow, EW_Green
);
    genvar i;
    generate
        for (i=0; i<N; i=i+1) begin : G
            ctl_with_pwm #(
                .GREEN_TIME(GREEN_TIME), .YELLOW_TIME(YELLOW_TIME),
                .CWIDTH(CWIDTH), .PWM_BITS(PWM_BITS),
                .UNIQUE_ID(i)
            ) u (
                .clk(clk), .rst(rst), .sensor(sensor[i]),
                .NS_Red(NS_Red[i]), .NS_Yellow(NS_Yellow[i]), .NS_Green(NS_Green[i]),
```

```
            .EW_Red(EW_Red[i]), .EW_Yellow(EW_Yellow[i]), .EW_Green(EW_Green[i])
        );
      end
  endgenerate
endmodule
```

Testbench

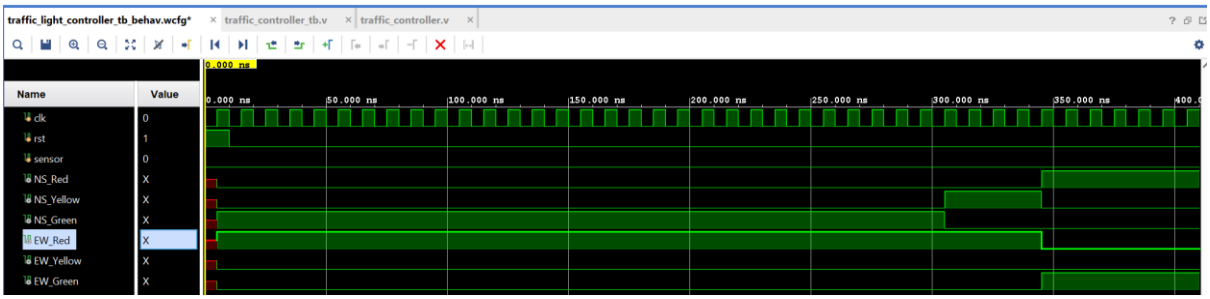
```
`timescale 1ns/1ps
module traffic_light_controller_tb;
 // ---- Tunables ----
 localparam integer CLK_PER_NS   = 10;   // 100 MHz
 localparam integer RESET_CYCLES = 10;   // hold reset this many cycles
 localparam integer RUN_CYCLES   = 4000; // main stimulus length
 // ---- DUT I/O ----
 reg  clk, rst, sensor;
 wire NS_Red, NS_Yellow, NS_Green;
 wire EW_Red, EW_Yellow, EW_Green;
 // Instantiate DUT (matches your gate netlist top)
 traffic_light_controller dut (
  .clk(clk), .rst(rst), .sensor(sensor),
  .NS_Red(NS_Red), .NS_Yellow(NS_Yellow), .NS_Green(NS_Green),
  .EW_Red(EW_Red), .EW_Yellow(EW_Yellow), .EW_Green(EW_Green)
 );
 // ---- Clock ----
 initial begin
  clk = 1'b0;
  forever #(CLK_PER_NS/2) clk = ~clk;
 end
```

```
  // ---- Optional waveform dump (VCD for non-ModelSim tools) ----
`ifdef VCD
  initial begin
    $dumpfile("traffic_light_controller_tb.vcd");
    $dumpvars(0, traffic_light_controller_tb);
  end
`endif
  // ---- Stimulus + heartbeat + basic checks ----
  integer i;
  initial begin
    $timeformat(-9,0," ns",10);
    $display("** TB start @ %0t **", $realtime);
    rst    = 1'b1;
    sensor = 1'b0;
    // reset
    for (i = 0; i < RESET_CYCLES; i = i + 1) @(posedge clk);
    rst = 1'b0;
    // main run
    for (i = 0; i < RUN_CYCLES; i = i + 1) begin
      @(posedge clk);
      // deterministic activity on 'sensor' (avoids $random quirks)
      sensor <= sensor ^ NS_Green ^ EW_Green ^ NS_Yellow ^ EW_Yellow;
      // heartbeat every 500 cycles so you see progress in Transcript
      if ((i % 500) == 0)
        $display("%0t: heartbeat (i=%0d)  NS[G,Y,R]=%0b%0b%0b  EW[G,Y,R]=%0b%0b%0b",
                 $realtime, i, NS_Green, NS_Yellow, NS_Red, EW_Green, EW_Yellow, EW_Red);
      // simple safety check: never both greens high in the same cycle
```

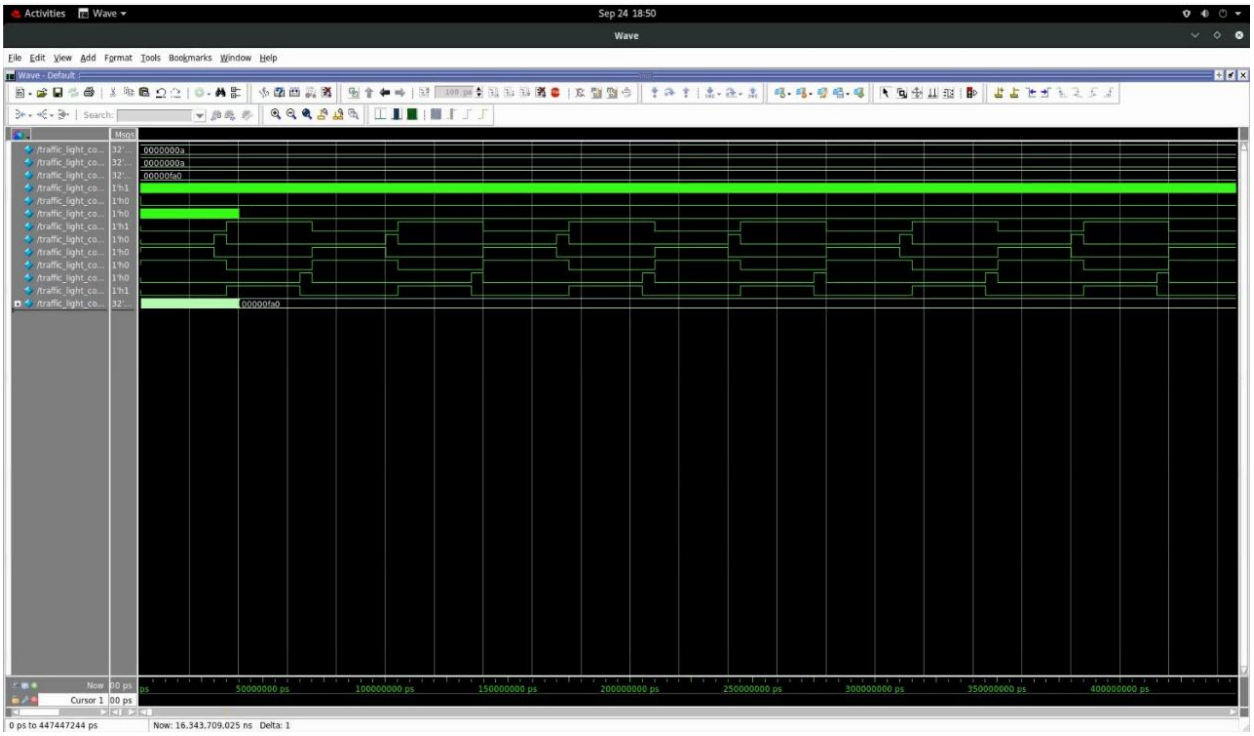
```
      if (!rst && (NS_Green & EW_Green)) begin
        $display("ERROR @ %0t: Both NS and EW GREEN!", $realtime);
        $stop;
      end
    end
    $display("** TB finish @ %0t **", $realtime);
    $finish;
  end
endmodule
```

Output

Waveform from the Behavioral code:



Waveform from the Mapped Code:



The Behavioral and mapped simulation waveforms of the traffic light controller match, confirming that the synthesized design preserves the intended functionality. The outputs transition correctly between North-South and East-West signals, validating both design and implementation.

Report from Design Vision

****

Report : cell

Design : traffic_light_controller

Version: O-2018.06-SP1

Date : Wed Sep 24 14:17:08 2025

****

Attributes:

b - black box (unknown)

h - hierarchical

n - noncombinational

r - removable

u - contains unmapped logic

Cell	Reference	Library	Area	Attributes
C283	nand2	library	1.000000	
C284	nand2	library	1.000000	
C288	nand2	library	1.000000	
C289	nand2	library	1.000000	
C293	nand2	library	1.000000	
C294	nand2	library	1.000000	
C298	nand2	library	1.000000	
C299	nand2	library	1.000000	
C1636	nand2	library	1.000000	
C1637	nand2	library	1.000000	

C1641	nand2	library	1.000000
C1642	nand2	library	1.000000
C1646	nand2	library	1.000000
C1647	nand2	library	1.000000
C1651	nand2	library	1.000000
C1652	nand2	library	1.000000
I_1	inv	library	1.000000
I_2	inv	library	1.000000
I_3	inv	library	1.000000
I_4	inv	library	1.000000
I_5	inv	library	1.000000
I_6	inv	library	1.000000
I_7	inv	library	1.000000
I_8	inv	library	1.000000
I_14	inv	library	1.000000
I_15	inv	library	1.000000
I_16	inv	library	1.000000
I_17	inv	library	1.000000
U268	oai12	library	2.000000
U269	inv	library	1.000000
U270	inv	library	1.000000
U271	inv	library	1.000000
U272	nand2	library	1.000000
U273	inv	library	1.000000
.			
.			
U1648	inv	library	1.000000

U1649 xor2 library 3.000000

U1650 xor2 library 3.000000

.

U1903 xor2 library 3.000000

U1904 nand2 library 1.000000

.

U1966 nand2 library 1.000000

U1967 inv library 1.000000

.

U2604 inv library 1.000000

counter_reg[0] dff library 7.000000 n

.

counter_reg[255] dff library 7.000000 n

r88/ULTI1 nand2 library 1.000000

r88/ULTI1_1 nand2 library 1.000000

.

r89/ULTI2_254 nand2 library 1.000000

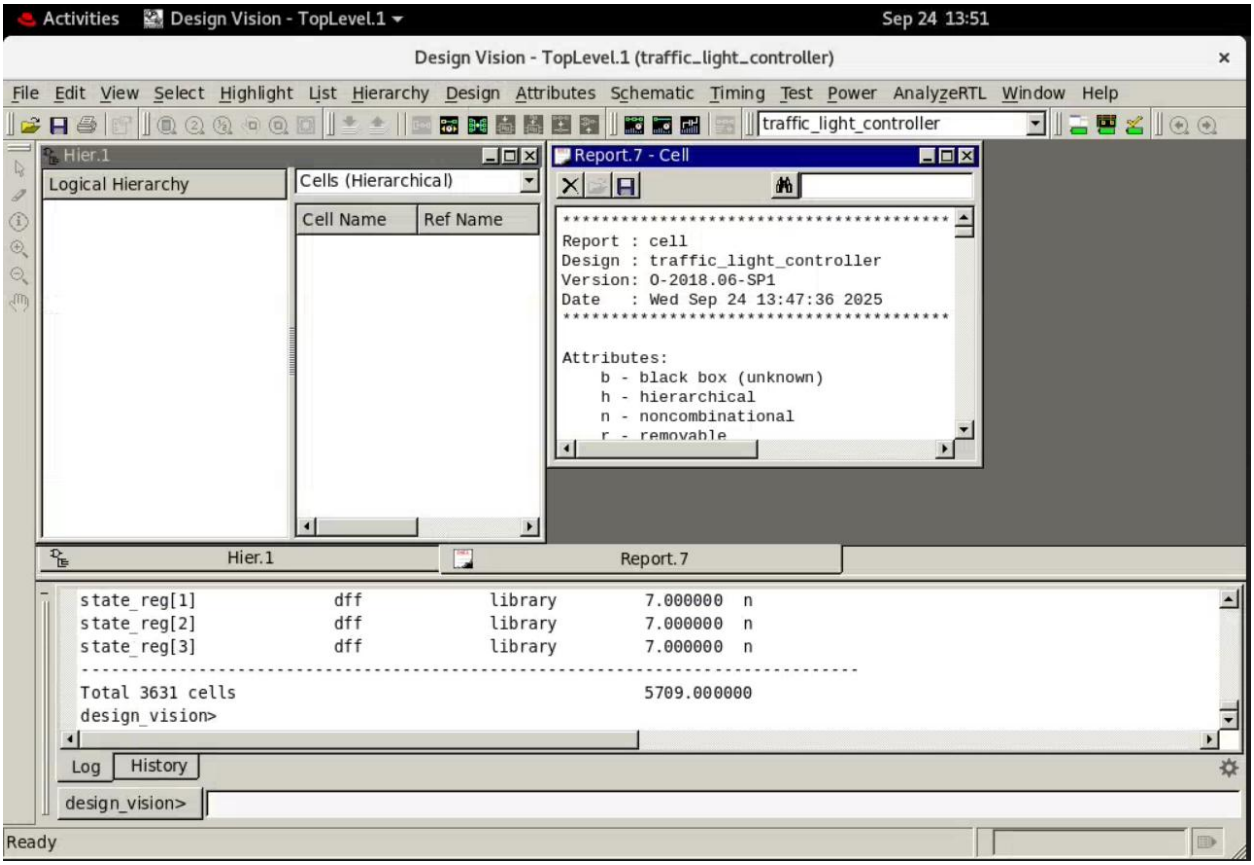
state_reg[0] dff library 7.000000 n

state_reg[1] dff library 7.000000 n

state_reg[2] dff library 7.000000 n

state_reg[3] dff library 7.000000 n

Total 3631 cells 5709.000000


Activities
Design Vision - TopLevel.1
Sep 24 13:51
Design Vision - TopLevel.1 (traffic_light_controller)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
traffic_light_controller
Hier.1
Logical Hierarchy
Cells (Hierarchical)
Cell Name
Ref Name
Report.7 - Cell
Report : cell
Design : traffic_light_controller
Version: O-2018.06-SP1
Date : Wed Sep 24 13:47:36 2025
Attributes:
b - black box (unknown)
h - hierarchical
n - noncombinational
Hier.1
Report.7
state_reg[1] dff library 7.000000 n
state_reg[2] dff library 7.000000 n
state_reg[3] dff library 7.000000 n
Total 3631 cells 5709.000000
design_vision>
Log
History
design_vision>
Ready