Design and Implementation of Traffic Light Controller

RTL CODE:

```
module traffic_light_controller (
 input clk,              // Clock signal (3 Hz recommended)
 input reset,            // Reset signal to initialize the FSM
 output reg NS_R,        // North-South Red Light
 output reg NS_Y,        // North-South Yellow Light
 output reg NS_G,        // North-South Green Light
 output reg EW_R,        // East-West Red Light
 output reg EW_Y,        // East-West Yellow Light
 output reg EW_G         // East-West Green Light
);
// State encoding
parameter S0 = 3'd0, // NS Green
        S1 = 3'd1, // NS Yellow
        S2 = 3'd2, // All Red
        S3 = 3'd3, // EW Green
        S4 = 3'd4, // EW Yellow
        S5 = 3'd5; // All Red again
reg [2:0] state, next_state;  // State and next state
reg [3:0] counter;            // Counter for delays (4-bit for up to 15)
parameter delay_S0 = 15, // 5 seconds (5 × 3 Hz cycles)
        delay_S1 = 3,  // 1 second
        delay_S2 = 3,  // 1 second
        delay_S3 = 15, // 5 seconds
        delay_S4 = 3,  // 1 second
        delay_S5 = 3;  // 1 second
reg [3:0] delay;  // Delay value for the current state
```

```
// State transition logic
always @(posedge clk or posedge reset) begin
    if (reset) begin
        state <= S0;  // Reset to initial state (NS Green)
        counter <= 0; // Reset the counter
    end else begin
        if (counter == delay) begin
            state <= next_state;   // Transition to next state
            counter <= 0;          // Reset the counter
        end else begin
            counter <= counter + 1;  // Increment the counter
        end
    end
end

// Next state logic based on current state
always @(*) begin
    case (state)
        S0: next_state = S1; // From NS Green to NS Yellow
        S1: next_state = S2; // From NS Yellow to All Red
        S2: next_state = S3; // From All Red to EW Green
        S3: next_state = S4; // From EW Green to EW Yellow
        S4: next_state = S5; // From EW Yellow to All Red
        S5: next_state = S0; // From All Red back to NS Green
        default: next_state = S0; // Default to NS Green
    endcase
end
// Output logic based on current state
always @(*) begin
    // Default all lights OFF
    NS_R = 0;
    NS_Y = 0;
    NS_G = 0;

    EW_R = 0;
    EW_Y = 0;
```

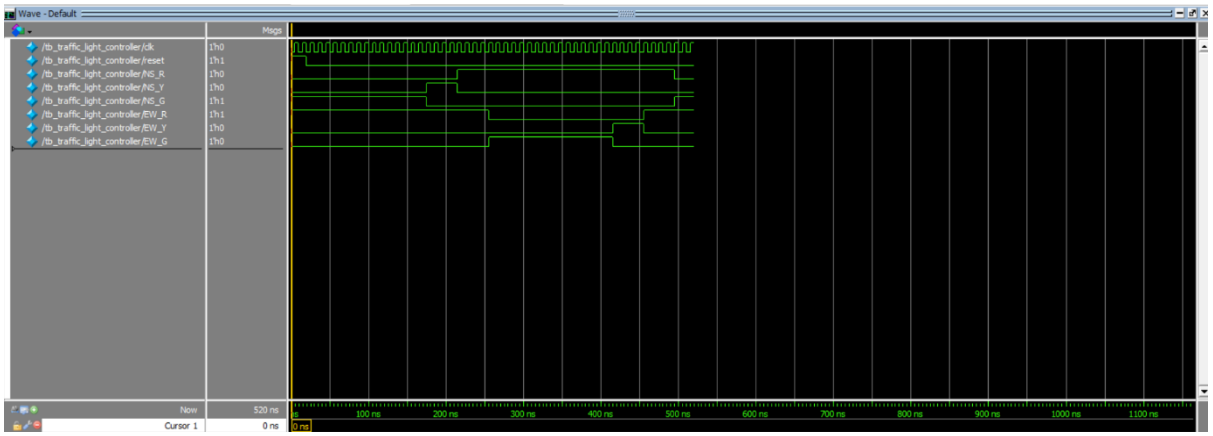
```
    EW_G = 0;
    case (state)
        S0: begin
            NS_G = 1; // NS Green
            EW_R = 1; // EW Red
            delay = delay_S0; // Set delay for NS Green
        end
        S1: begin
            NS_Y = 1; // NS Yellow
            EW_R = 1; // EW Red
            delay = delay_S1; // Set delay for NS Yellow
        end
        S2: begin
            NS_R = 1; // NS Red
            EW_R = 1; // EW Red
            delay = delay_S2; // Set delay for All Red
        end
        S3: begin
            NS_R = 1; // NS Red
            EW_G = 1; // EW Green
            delay = delay_S3; // Set delay for EW Green
        end
        S4: begin
            NS_R = 1; // NS Red
            EW_Y = 1; // EW Yellow
            delay = delay_S4; // Set delay for EW Yellow
        end
        S5: begin
            NS_R = 1; // NS Red
            EW_R = 1; // EW Red
            delay = delay_S5; // Set delay for All Red
        end
    endcase
end
endmodule
```

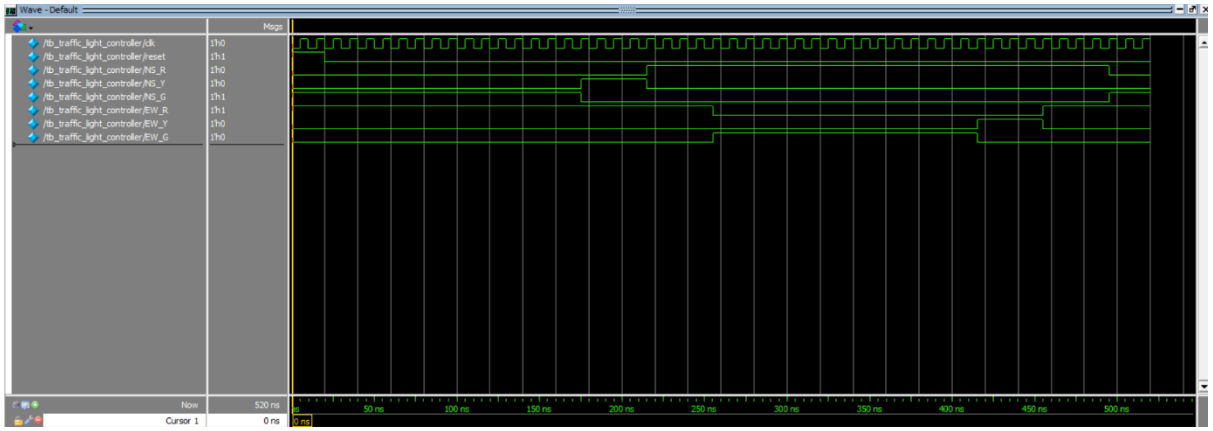
TEST BENCH:

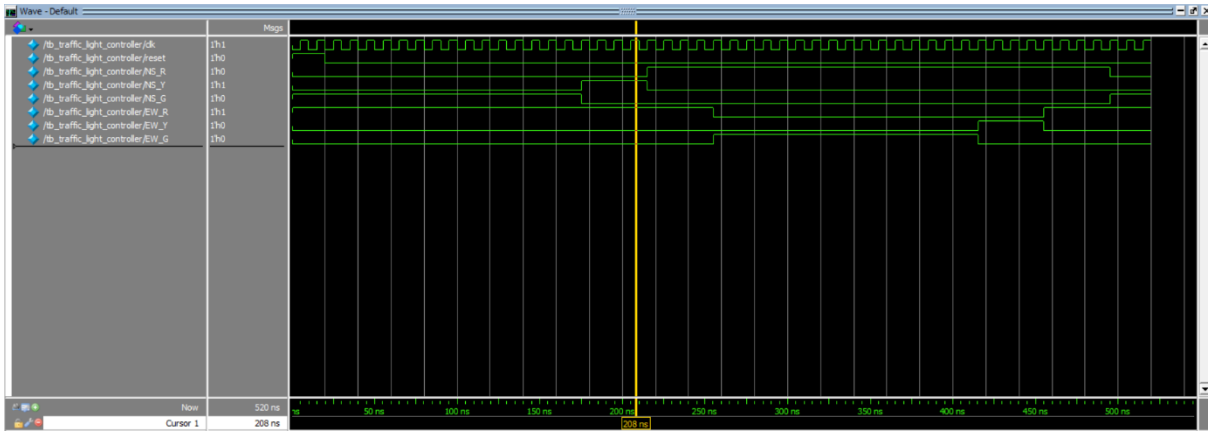
```
module tb_traffic_light_controller;
// Inputs to the DUT (Device Under Test)
reg clk;
reg reset;
// Outputs from the DUT
wire NS_R, NS_Y, NS_G, EW_R, EW_Y, EW_G;
// Instantiate the Traffic Light Controller module
traffic_light_controller uut (
    .clk(clk),
    .reset(reset),
    .NS_R(NS_R),
    .NS_Y(NS_Y),
    .NS_G(NS_G),
    .EW_R(EW_R),
    .EW_Y(EW_Y),
    .EW_G(EW_G)
);
// Clock generation (3 Hz clock)
always begin
    #5 clk = ~clk;  // 10ns period, so 5ns high, 5ns low (3Hz clock)
end
// Initial block to apply reset and test the controller
initial begin
    // Initialize signals
    clk = 0;
    reset = 1;  // Start with reset active
    // Apply reset for a few clock cycles
    #20 reset = 0;  // Deassert reset after 20ns

    // Let the simulation run for a while to see the state transitions
    #500;  // Run for 500ns (enough time for a few cycles)
    // Finish the simulation
    $finish;
end
// Monitor outputs (optional)
```

```
initial begin
    $monitor("At time %t: NS_R = %b, NS_Y = %b, NS_G = %b, EW_R = %b, EW_Y = %b, EW_G = %b",
            $time, NS_R, NS_Y, NS_G, EW_R, EW_Y, EW_G);
end
endmodule
```

OUTPUT:


Wave - Default
Msgs
/tb_traffic_light_controller/clk
/tb_traffic_light_controller/reset
/tb_traffic_light_controller/NS_R
/tb_traffic_light_controller/NS_Y
/tb_traffic_light_controller/NS_G
/tb_traffic_light_controller/EW_R
/tb_traffic_light_controller/EW_Y
/tb_traffic_light_controller/EW_G
Now
520 ns
Cursor 1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
1100 ns


Wave - Default
Msgs
/tb_traffic_light_controller/clk
/tb_traffic_light_controller/reset
/tb_traffic_light_controller/NS_R
/tb_traffic_light_controller/NS_Y
/tb_traffic_light_controller/NS_G
/tb_traffic_light_controller/EW_R
/tb_traffic_light_controller/EW_Y
/tb_traffic_light_controller/EW_G
Now
520 ns
Cursor 1
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns


Wave - Default
Msgs
/tb_traffic_light_controller/clk
/tb_traffic_light_controller/reset
/tb_traffic_light_controller/NS_R
/tb_traffic_light_controller/NS_Y
/tb_traffic_light_controller/NS_G
/tb_traffic_light_controller/EW_R
/tb_traffic_light_controller/EW_Y
/tb_traffic_light_controller/EW_G
Now
520 ns
Cursor 1
208 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns