Traffic Signal Controller

Devender Sharma

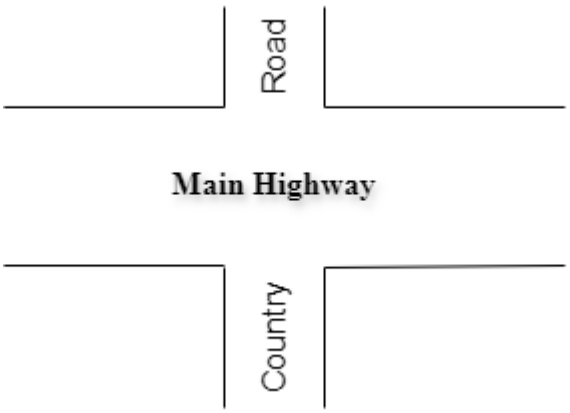
VERILOG PROJECT

Overview

This project presents the design and simulation of a digital traffic signal controller using Verilog HDL. The controller manages traffic signals for both a highway and a country road based on vehicle presence. The design was verified using a testbench, which simulated various traffic conditions. The results confirmed that the controller successfully managed traffic signals, transitioning between states in response to vehicle presence and timing delays. The project demonstrates the application of finite state machines (FSM) in traffic management systems.

Specifications

Consider a controller for traffic at the intersection of a main highway and a country road.



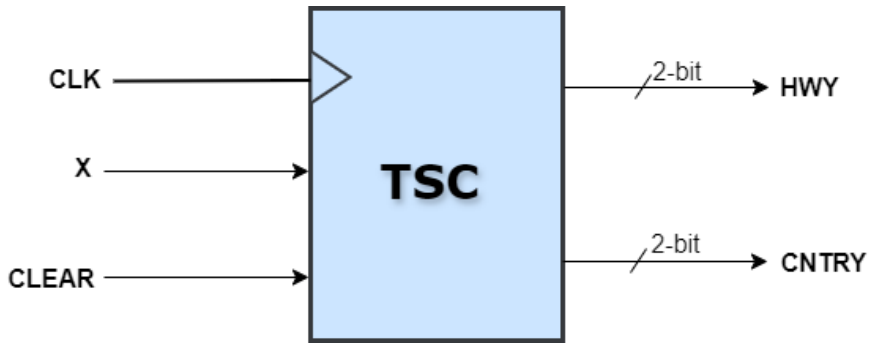
The following specifications must be considered.

- The traffic signal for the main highway gets the highest priority because cars are continuously present on the main highway. Thus, the main highway signal remains green by default.
- Occasionally, cars from the country road arrive at the traffic signal. The traffic signal for the country road must turn green only long enough to let the cars on the country road go.

- As soon as there are no cars on the country road, the country road traffic signal turns yellow and then red and the traffic signal on the main highway turns green again.
- There is a sensor to detect cars waiting on the country road. The sensor sends a signal X as input to the controller, X = 1 if there are cars on the country road; otherwise, X = 0.
- There are delays on transitions from S1 to S2, from S2 to S3, and from S4 to S0. The delays must be controllable.

System Design

Block Diagram:
The system consists of:

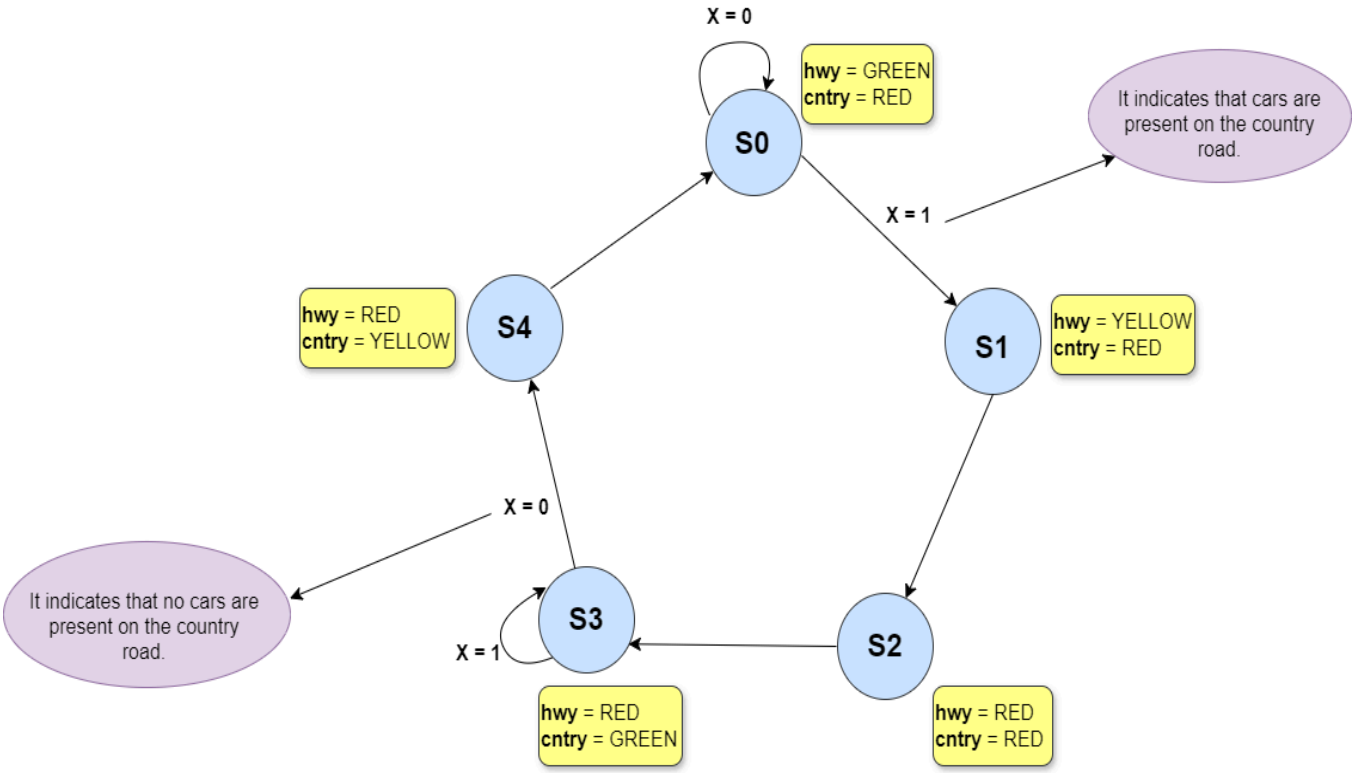


Traffic Signal Controller Module: Manages the state transitions and output signals.

- **Inputs:**
  - `X`: A signal indicating whether there is a car on the country road (`1` means there is a car, `0` means there isn't).
  - `clk`: This is the clock signal that synchronizes the state changes.
  - `clear`: This input is used to reset the controller to its starting state (S0).
- **Outputs:**
  - `hwy`: A 2-bit output that controls the traffic signal for the highway.The signal can be GREEN,YELLOW or RED.
  - `cntry`:A 2-bit output that controls the traffic signal for the country road.The signal can be GREEN,YELLOW or RED.

State Diagram:

The state diagram includes the following states:



- **S0:** Highway = G Country = R
- **S1:** Highway = Y Country = R
- **S2:** Highway = R Country = R
- **S3:** Highway = R Country = G
- **S4:** Highway = R Country = Y

State transitions are based on vehicle presence (X) and predefined delays.

Design and Implementation

Design Code:

```
`define TRUE 1'b1
`define FALSE 1'b0

// Delays
`define Y2RDELAY 3       // Yellow to Red Delay
```

```
`define R2GDELAY 2        // Red to Green Delay

module traffic_sig_controller(
    output reg [1:0] hwy,
    output reg [1:0] cntry,
    input X,
    input clk,
    input clear
);
```

Code Explanation :

- `define TRUE 1'b1` and `define FALSE 1'b0`:
  - Defines constants `TRUE` and `FALSE` with values `1` and `0`, respectively. These are used throughout the code to make the logic easier to understand.
- `define Y2RDELAY 3` and `define R2GDELAY 2`:
  - These define the number of clock cycles to wait when changing the traffic signal from Yellow to Red (`Y2RDELAY`) and from Red to Green (`R2GDELAY`).
- `module traffic_sig_controller`:
  - This starts the definition of the traffic signal controller module.
  - Outputs:
    - `hwy`: 2-bit output representing the highway traffic light (GREEN, YELLOW, RED).
    - `cntry`: 2-bit output representing the country road traffic light (GREEN, YELLOW, RED).
  - Inputs:
    - `X`: Indicates if there is a car on the country road. `TRUE` means there is a car; `FALSE` means there isn't.
    - `clk`: The clock signal that drives the timing of the controller.
    - `clear`: A reset signal that, when active, resets the controller to its initial state.

```
1  `timescale 1ns / 1ps
2
3  `define TRUE 1'b1
4  `define FALSE 1'b0
5
6  // Delays
7  `define Y2RDELAY 3       // Yellow to Red Delay
8  `define R2GDELAY 2       // Red to Green Delay
9
10 module traffic_sig_controller(
11     output reg [1:0] hwy,    // 2-bit output for 3 states of signal: GREEN, RED, YELLOW
12     output reg [1:0] cntry,  // 2-bit output for 3 states of signal: GREEN, RED, YELLOW
13     input X,                 // if TRUE, indicates that there is a car on the country road, other
14     input clk,
15     input clear
16 );
17
```

```
parameter RED = 2'd0,
          YELLOW = 2'd1,
          GREEN = 2'd2;
```

Code Explanation：

- `parameter RED = 2'd0`:
  - Defines the RED state of the traffic signal as a 2-bit binary value 00.
- `parameter YELLOW = 2'd1`:
  - Defines the YELLOW state of the traffic signal as a 2-bit binary value 01.
- `parameter GREEN = 2'd2`:
  - Defines the GREEN state of the traffic signal as a 2-bit binary value 10.

These constants are used to set the state of the hwy and `cntry` outputs.

```
// State Definition
 parameter S0 = 3'd0,
           S1 = 3'd1,
           S2 = 3'd2,
           S3 = 3'd3,
           S4 = 3'd4;
```

Code Explanation：

- **`parameter S0 = 3'd0:`**
  - Defines the state S0 as 3'd0 (binary 000), where the highway light is green, and the country road light is red.
- **`parameter S1 = 3'd1:`**
  - Defines the state S1 as 3'd1 (binary 001), where the highway light turns yellow, and the country road light remains red.
- **`parameter S2 = 3'd2:`**
  - Defines the state S2 as 3'd2 (binary 010), where both highway and country road lights are red.
- **`parameter S3 = 3'd3:`**
  - Defines the state S3 as 3'd3 (binary 011), where the country road light turns green, and the highway light remains red.
- **`parameter S4 = 3'd4:`**
  - Defines the state S4 as 3'd4 (binary 100), where the country road light turns yellow, and the highway light remains red.

These states represent different scenarios of traffic light configurations.

```
// Internal State Variables
  reg [2:0] prest_state;
  reg [2:0] next_state;
```

Code Explanation:

- **reg [2:0] prest_state:**
  - Declares a 3-bit register prest_state that holds the current state of the traffic signal controller.
  - The value of prest_state will determine the current output of the traffic signals.
- **reg [2:0] next_state:**
  - Declares a 3-bit register next_state that holds the next state the traffic signal controller will move to.
  - The value of next_state is determined by the current state (prest_state) and the input (X).

These registers are used to store and update the state of the controller as it runs.

```
18   parameter RED = 2'd0,
19             YELLOW = 2'd1,
20             GREEN = 2'd2;
21
22
23   // State Definition
24
25   parameter S0 = 3'd0,
26             S1 = 3'd1,
27             S2 = 3'd2,
28             S3 = 3'd3,
29             S4 = 3'd4;
30
31  // Internal State Variables
32
33   reg [2:0] prest_state;
34   reg [2:0] next_state;
35
```

State Changes on the Positive Edge of the Clock :

```
always @(posedge clk) begin
    if (clear)
        prest_state <= S0;
```

```
    else
        prest_state <= next_state;
end
```

Code Explanation:

- **always @(posedge clk):**
  - This is a block that triggers on the rising edge (positive edge) of the clock signal (clk).
  - It means that the code inside this block runs every time there is a positive transition of the clock signal.
- **if (clear):**
  - This checks if the clear signal is active (logic high).
  - If clear is active, the controller resets to the initial state S0.
- **prest_state <= S0;:**
  - If clear is active, the current state (prest_state) is set to S0, which is the initial state where the highway light is green, and the country road light is red.
- **else prest_state <= next_state;:**
  - If clear is not active, the current state (prest_state) is updated to the next_state.
  - This means the controller transitions to the next state based on the logic defined in other parts of the code.

Compute Values of Highway and Country Road Signals

```
always @(prest_state) begin
    case (prest_state)
        S0: begin
            hwy = GREEN;
            cntry = RED;
        end

        S1: begin
            hwy = YELLOW;
            cntry = RED;
        end

        S2: begin
            hwy = RED;
            cntry = RED;
        end

        S3: begin
            hwy = RED;
            cntry = GREEN;
```

```
        end

        S4: begin
            hwy = RED;
            cntry = YELLOW;
        end

        default: begin
            hwy = GREEN;
            cntry = RED;
        end
    endcase
end
```

Code Explanation:

- **always @(prest_state):**
  - This block triggers whenever there is a change in the value of prest_state.
  - The block is used to determine the output values (hwy and cntry) based on the current state.
- **case (prest_state):**
  - This is a case statement that checks the value of prest_state and sets the hwy and cntry signals accordingly.
- **S0: begin hwy = GREEN; cntry = RED; end:**
  - When the controller is in state S0, the highway light (hwy) is set to **Green**, and the country road light (cntry) is set to **Red**.
- **S1: begin hwy = YELLOW; cntry = RED; end:**
  - When the controller is in state S1, the highway light (hwy) is set to **Yellow**, and the country road light (cntry) remains **Red**.
- **S2: begin hwy = RED; cntry = RED; end:**
  - When the controller is in state S2, both the highway light (hwy) and the country road light (cntry) are set to **Red**.
- **S3: begin hwy = RED; cntry = GREEN; end:**
  - When the controller is in state S3, the highway light (hwy) is **Red**, and the country road light (cntry) is set to **Green**.
- **S4: begin hwy = RED; cntry = YELLOW; end:**
  - When the controller is in state S4, the highway light (hwy) remains **Red**, and the country road light (cntry) turns **Yellow**.

- **default: begin hwy = GREEN; cntry = RED; end:**
    - This is the default case. If for some reason prest_state has an undefined value, the controller will default to the S0 state where the highway light is **Green**, and the country road light is **Red**.

```
// State Changes only at positive edge of
always @(posedge clk)begin
if(clear)
     prest_state <= S0;
else
    prest_state <= next_state;
end

// Compute values of main signal and country signal.

    always @(prest_state)
        begin
          case(prest_state)

            S0 : begin
                 hwy = GREEN;
                 cntry = RED;
                 end


           S1: begin
                hwy = YELLOW;
                cntry = RED;
               end

           S2: begin
                hwy = RED;
                cntry = RED;
               end

           S3: begin
                hwy = RED;
                cntry = GREEN;
               end


            S4: begin
                 hwy = RED;
                 cntry = YELLOW;
                end

            default : begin
                        hwy = GREEN;
                        cntry = RED;
                        end

          endcase
end
```

This part of the code handles the logic for determining the next state of the traffic signal controller based on the current state (prest_state) and the input signal X.

```
// Logic Part

      always @(prest_state or X)
        begin
          case(prest_state)

              S0 : next_state = X ? S1 : S0;


              S1 : begin
                   repeat (`Y2RDELAY) @(posedge clk);
                   next_state = S2;
                   end


              S2 : begin
                   repeat (`R2GDELAY) @(posedge clk);
                   next_state = S3;
                   end

              S3 :    next_state = X ? S3 : S4;

              S4 : begin
                   repeat (`Y2RDELAY) @(posedge clk);
                   next_state = S0;
                   end

              default : next_state = S0;

            endcase
        end

      endmodule
```

Code Explanation:

- **`always @(prest_state or X)`:**
  - This block triggers whenever there is a change in the current state (`prest_state`) or the input signal (X).

- Inside this block, the next state (`next_state`) is determined based on the current state and the value of `X`.

- **`case (prest_state):`**
  - This `case` statement checks the value of `prest_state` and determines the appropriate `next_state` based on it.

State Transition Logic

State `S0`

Explanation:

- If the controller is in state `S0`, it checks the value of `X`.
- If `X` is `TRUE` (meaning there is a car on the country road), the next state will be `S1` (highway light turns yellow).
- If `X` is `FALSE`, the next state remains `S0` (highway light stays green, country road light stays red).

State `S1`

Explanation:

- In state `S1`, the highway light is yellow, and the country road light is red.
- The code waits for `Y2RDELAY` clock cycles before transitioning to the next state (`S2`).
- The `repeat (Y2RDELAY) @(posedge clk);` line makes the controller wait for `Y2RDELAY` clock cycles (defined as 3) before proceeding.

State `S2`

Explanation:

- `In state S2, both the highway and country road lights are red.`
- `The controller waits for R2GDELAY clock cycles (defined as 2) before transitioning to the next state (S3).`

State `S3`

Explanation:

- `In state S3, the highway light is red, and the country road light is green.`
- `If X is TRUE (meaning there is still a car on the country road), the controller stays in state S3.`
- `If X is FALSE (no car on the country road), the controller transitions to state S4 (country road light turns yellow).`

State_S4

Explanation:

- In state S4, the highway light remains red, and the country road light is yellow.
- The controller waits for Y2RDELAY clock cycles before transitioning back to state S0 (highway light turns green, country road light turns red).

Default_Case

Explanation:

The default case ensures that if the controller somehow ends up in an undefined state, it will default back to S0, the initial state.

```
94  // Logic Part
95
96      always @(prest_state or X)
97        begin
98          case(prest_state)
99
100             S0 : next_state = X ? S1 : S0;
101
102
103             S1 : begin
104                  repeat (`Y2RDELAY) @(posedge clk);
105                  next_state = S2;
106                  end
107
108
109             S2 : begin
110                  repeat (`R2GDELAY) @(posedge clk);
111                  next_state = S3;
112                  end
113
114               S3 :  next_state = X ? S3 : S4;
115
116               S4 : begin
117                    repeat (`Y2RDELAY) @(posedge clk);
118                    next_state = S0;
119                    end
120
121               default : next_state = S0;
122
123           endcase
124       end
125
126     endmodule
```

Testbench Code

Project Summary | traffic_sig_controller.v * | stimulus.v

D:/Vivado1/Traffic Signal Controller/Traffic Signal Controller.srcs/sim_1/new/stimulus.v

```
1  `timescale 1ns / 1ps
2
3
4  module stimulus;
5
6  wire [1:0] HWY_SIG, CNTRY_SIG;
7  reg CAR_ON_CNTRY_RD;
8  reg CLK, CLEAR;
9
10 // Instantiate signal controller
11 traffic_sig_controller TSC(HWY_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD, CLK, CLEAR);
12
13 // Set Up Monitor
14 initial
15 $monitor($time, "HWY Sig = %b Country Sig = %b  Car_on_cntry = %b",HWY_SIG,CNTRY_SIG,CAR_ON_CNTRY_RD);
16
17 // Set Up Clock
18 initial
19 begin
```

Project Summary | traffic_sig_controller.v * | stimulus.v

D:/Vivado1/Traffic Signal Controller/Traffic Signal Controller.srcs/sim_1/new/stimulus.v

```
20 CLK = `FALSE;
21 forever #5 CLK = ~ CLK;
22 end
23
24 // Control Clear Signal
25
26 initial
27 begin
28      CLEAR = `TRUE;
29      repeat (5) @(negedge CLK);
30      CLEAR = `FALSE;
31 end
32
33 // Apply Stimulus
34
35 initial
36 begin
37      CAR_ON_CNTRY_RD = `FALSE;
38
39      #200 CAR_ON_CNTRY_RD = `TRUE;
40      #100 CAR_ON_CNTRY_RD = `FALSE;
41
42      #200 CAR_ON_CNTRY_RD = `TRUE;
43      #100 CAR_ON_CNTRY_RD = `FALSE;
44
45      #200 CAR_ON_CNTRY_RD = `TRUE;
46      #100 CAR_ON_CNTRY_RD = `FALSE;
47
48      #100 $stop;
49      end
50
51 endmodule
52
53
```

A testbench is used to simulate and verify the behavior of a hardware design (Traffic Signal Controller).

Module Declaration

- **module stimulus;**: Declares the testbench module named stimulus.

Signal Declarations

- **wire [1:0] HWY_SIG, CNTRY_SIG;**: 2-bit wires representing the highway and country road signals.
- **reg CAR_ON_CNTRY_RD;**: Register simulating whether a car is on the country road.
- **reg CLK, CLEAR;**: Registers for the clock signal (CLK) and reset signal (CLEAR).

Instantiate the Traffic Signal Controller

- **traffic_sig_controller TSC(HWY_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD, CLK, CLEAR);**
  - Creates an instance of the traffic_sig_controller module.
  - Connects the signals to the module's inputs and outputs.

Monitor Setup

- **$monitor($time, "HWY Sig = %b Country Sig = %b Car_on_cntry = %b", HWY_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD);**
  - Continuously monitors and displays the values of HWY_SIG, CNTRY_SIG, and CAR_ON_CNTRY_RD whenever they change.
  - Displays the simulation time ($time) along with the signal values.

Clock Signal Setup

- **initial begin CLK = FALSE; forever #5 CLK = ~ CLK; end**
  - Initializes the clock signal (CLK) to FALSE.
  - Toggles the clock every 5 time units, creating a clock with a period of 10 time units.

Control Clear Signal

- **initial begin CLEAR = TRUE; repeat (5) @(negedge CLK); CLEAR = FALSE; end**
  - Sets CLEAR to TRUE initially, resetting the traffic signal controller.
  - Waits for 5 negative edges of the clock (negedge CLK).
  - After 5 clock cycles, set CLEAR to FALSE, allowing the controller to operate normally.

Apply Stimulus

- **initial begin CAR_ON_CNTRY_RD = FALSE;:** Starts with no car on the country road.
- **#200 CAR_ON_CNTRY_RD = TRUE;:** After 200 time units, simulates a car on the country road (CAR_ON_CNTRY_RD = TRUE).
- **#100 CAR_ON_CNTRY_RD = FALSE;:** After 100 time units, simulates the car leaving (CAR_ON_CNTRY_RD = FALSE).
- **Repeats:** The car appears and disappears a few more times with similar delays.
- **#100 $stop;:** Stops the simulation after the final stimulus.

Output:

```
Time:                   0, Highway Sig = xx, Country Sig = xx, Car_on_cntry = 0
Time:                   5, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 0
Time:                 200, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 1
Time:                 205, Highway Sig = 01, Country Sig = 00, Car_on_cntry = 1
Time:                 245, Highway Sig = 00, Country Sig = 00, Car_on_cntry = 1
Time:                 275, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 1
Time:                 300, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 0
Time:                 305, Highway Sig = 00, Country Sig = 01, Car_on_cntry = 0
Time:                 345, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 0
Time:                 500, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 1
Time:                 505, Highway Sig = 01, Country Sig = 00, Car_on_cntry = 1
Time:                 545, Highway Sig = 00, Country Sig = 00, Car_on_cntry = 1
Time:                 575, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 1
Time:                 600, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 0
Time:                 605, Highway Sig = 00, Country Sig = 01, Car_on_cntry = 0
Time:                 645, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 0
Time:                 800, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 1
Time:                 805, Highway Sig = 01, Country Sig = 00, Car_on_cntry = 1
Time:                 845, Highway Sig = 00, Country Sig = 00, Car_on_cntry = 1
Time:                 875, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 1
Time:                 900, Highway Sig = 00, Country Sig = 10, Car_on_cntry = 0
Time:                 905, Highway Sig = 00, Country Sig = 01, Car_on_cntry = 0
Time:                 945, Highway Sig = 10, Country Sig = 00, Car_on_cntry = 0
$stop called at time : 1 us : File "D:/Vivado1/Traffic Signal Controller/Traffic Signal Controller.srcs/sim_1/new/stimulus.v" Line 49
xsim: Time (s): cpu = 00:00:07 ; elapsed = 00:00:06 . Memory (MB): peak = 2001.758 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'stimulus_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:27 . Memory (MB): peak = 2001.758 ; gain = 0.000
```

Simulation:

Output provides a clear view of how the traffic signal controller operates over time.

➔ At Initial Time (0)ns

Highway Signal (HWY_SIG): xx (initial state, not yet defined)

Country Signal (CNTRY_SIG): xx (initial state, not yet defined)

Car on Country Road (CAR_ON_CNTRY_RD): 0 (no car)

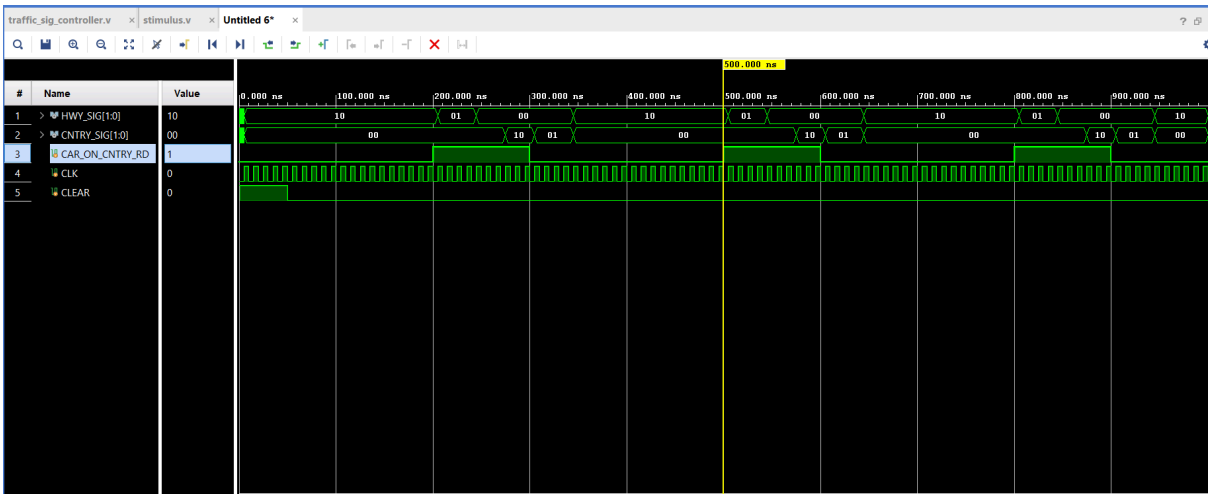


Fig:1

➔ At Time 5ns

Highway Signal (HWY_SIG): 10 (Green)
Country Signal (CNTRY_SIG): 00 (Red)
Car on Country Road (CAR_ON_CNTRY_RD): 0 (no car)

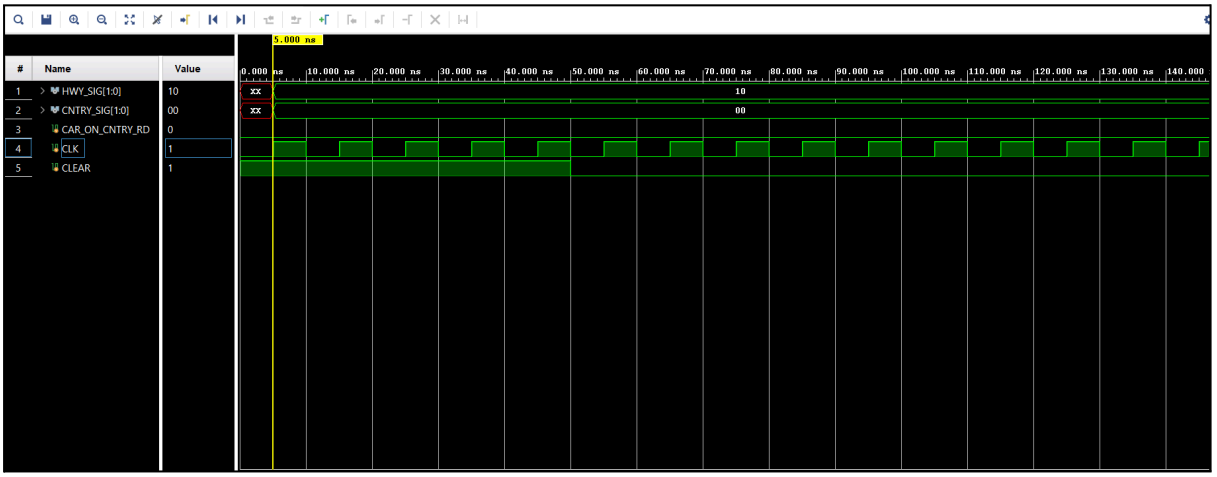


Fig:2

➔ At Time 200ns

Car on Country Road (CAR_ON_CNTRY_RD): 1 (a car appears)
Highway Signal (HWY_SIG): 10 (Green)
Country Signal (CNTRY_SIG): 00 (Red)

Fig:3

➔ At Time 205ns

Highway Signal (HWY_SIG): 01 (Yellow)
Country Signal (CNTRY_SIG): 00 (Red)

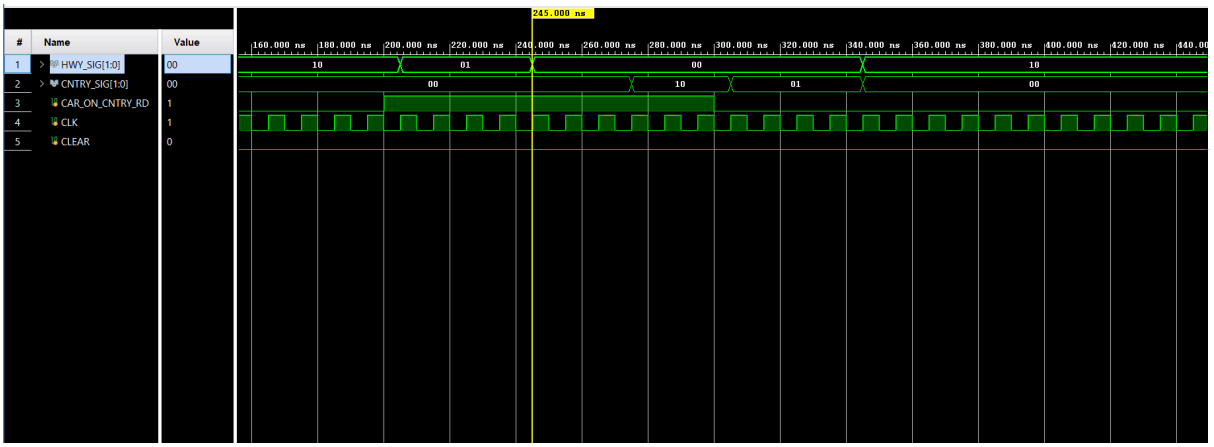


Fig:4

➔ At Time 245ns

Highway Signal (HWY_SIG): 00 (Red)
Country Signal (CNTRY_SIG): 00 (Red)

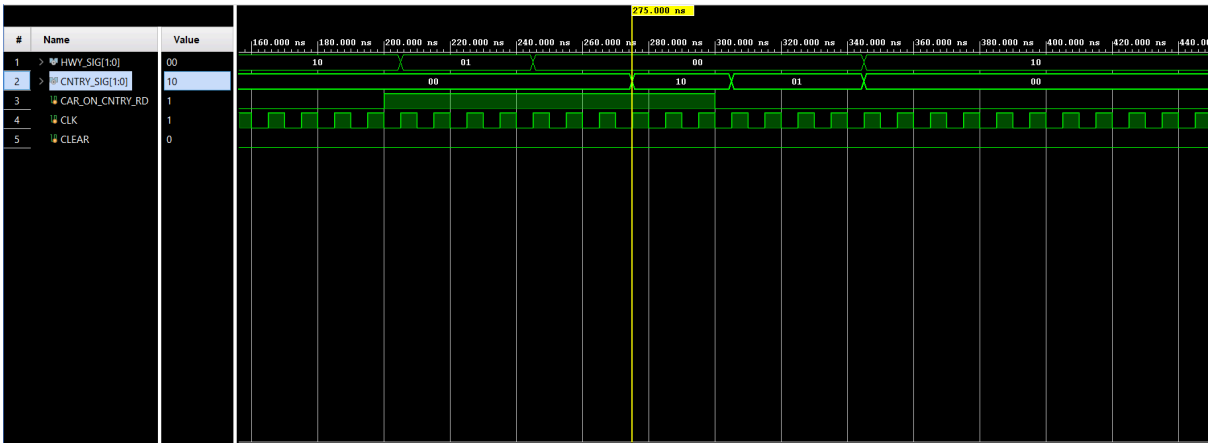


Fig:5

➔ At Time 275ns

Highway Signal (HWY_SIG): 00 (Red)
Country Signal (CNTRY_SIG): 10 (Green)

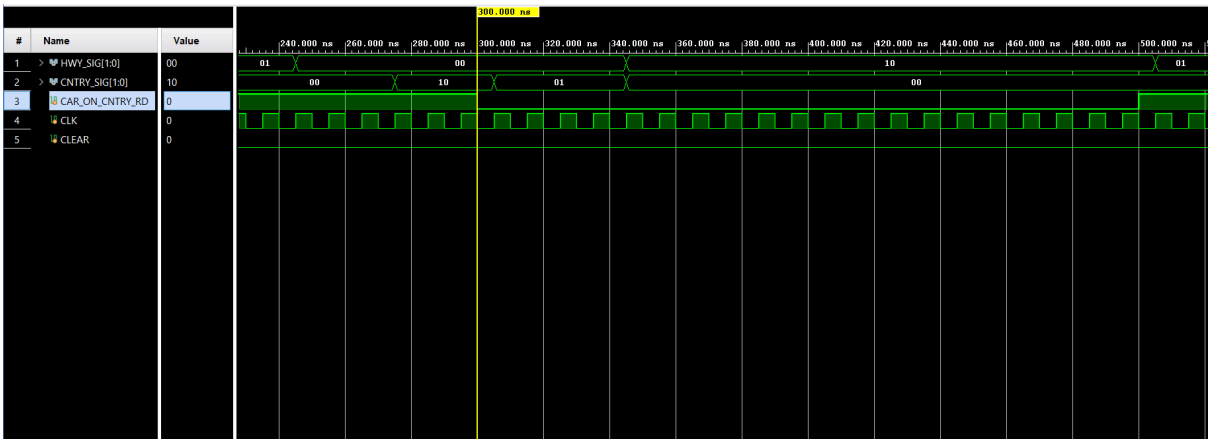


Fig:6

➔ At Time 300ns

Car on Country Road (CAR_ON_CNTRY_RD): 0 (car leaves)
Highway Signal (HWY_SIG): 00 (Red)
Country Signal (CNTRY_SIG): 10 (Green)

Fig:7

➔ At Time 305ns

Highway Signal (HWY_SIG): 00 (Red)
Country Signal (CNTRY_SIG): 01 (Yellow)

Fig:8

➔ At Time 345ns

Highway Signal (HWY_SIG): 10 (Green)
Country Signal (CNTRY_SIG): 00 (Red)

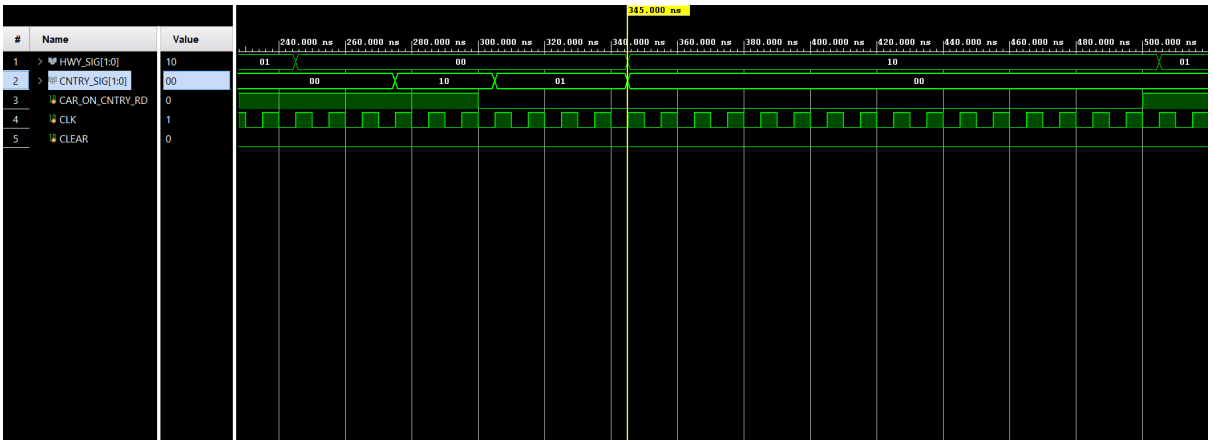
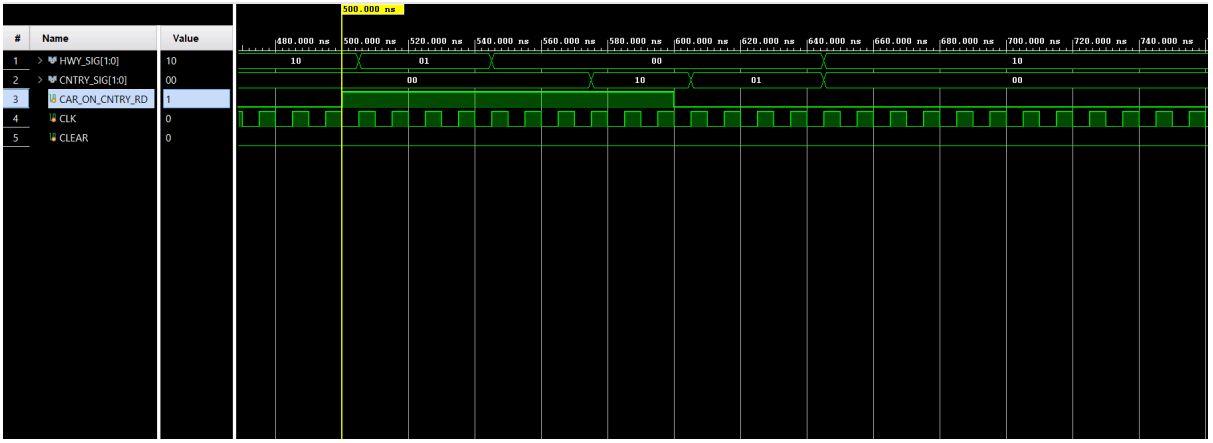


Fig:9

➔ Times 500-905 ns

The cycle of signals and car presence repeats, showing consistent behavior according to the state transitions and delays defined in the traffic signal controller.



Summary of Behavior:

Initial State: The controller starts with the highway light green and the country road light red.

With Car Present: When a car appears on the country road, the highway light transitions from green to yellow and then to red. The country road light transitions from red to green.

Without Car Present: When the car leaves, the highway light returns to green, and the country road light transitions to yellow and then to red.

Repetition: The signals and car presence cycle according to the defined delays, verifying the controller's operation in different scenarios.

Conclusion:

The output shows that the traffic signal controller behaves correctly according to the given specifications. It cycles through the different states of the traffic lights based on the presence of a car and the predefined delays, as described in the Verilog code.

THANK YOU !!

GITHUB

LINKEDIN

GMAIL