# DELHI TECHNOLOGICAL UNIVERSITY


## PROJECT FOR INNOVATIVE WORK
## DIGITAL CIRCUITS AND SYSTEMS (EE-204),
## 4th SEMESTER (JANUARY 2021)

**TOPIC: DESIGN AND MODELLING OF TRAFFIC SIGNAL CONTROLLERS BASED ON FINITE STATE MACHINE USING VERILOG HARDWARE DESCRIPTION LANGUAGE**

**SUBMITTED BY: DIGANT RASTOGI**

**ROLL NO.: 2K19/EE/092**

**BRANCH: ELECTRICAL ENGINEERING**

**SUBMITTED TO: DR. RAJESH KUMAR**

**DEPARTMENT OF ELECTRICAL ENGINEERING**

# ACKNOWLEDGEMENT

I express deepest gratitude to our respected Professor Dr. Rajesh Kumar, Faculty, Department of Electrical Engineering, Delhi Technological University (DTU) for his benevolent guidance in accomplishment of this project successfully.

Dr. Rajesh Kumar inspired me to work hard and gave me enough time and support whenever needed. I feel extremely fortunate to have worked under his kind supervision. His active encouragement has enabled me to complete the project in due course of time. I gratefully acknowledge the precious advice and suggestions received from the other faculty members and the knowledge gained during the discussions in the class.

I express my special thanks to the university library, for help in accessing online journals and research papers and my parents and friends for their cooperation.

**- DIGANT RASTOGI**
**(2K19/EE/092)**

# Traffic Light Controller Design and Simulation based on Verilog

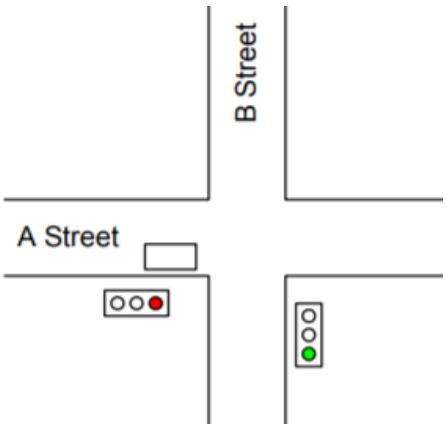
## ABSTRACT

All modern roadways have to deal with high traffic densities which lead to congestion and road blocks for hours. Conventional traffic control systems have a major drawback: Due to lack of adjustments in timing of traffic signals, the traffic has to wait for a long duration on the lane with few vehicles, while on the same lane the traffic cannot pass through in a short time due to rush on lane. So, there is a need to develop a secure, fast and reliable traffic control system capable of controlling vehicular traffic in rush hours without a need for a traffic sergeant.

Such a problem can be easily solved with the use of automatic traffic signal controllers.

Through this project I will design an FPGA based intelligent traffic light controller for a 4 road junction, as shown in the figure.

There are numerous advantages of working with FPGAs instead of ASIC or microcontrollers for such an application, some being reduced cost, real time programmability and higher processing speeds. Therefore, I will base our traffic controller on FPGA.



Finite state machines are computational models used to simulate sequential logic circuits. Since a traffic light will be based on sequential logic, it will be modeled as a finite state machine.

Verilog is a hardware description language (HDL) extensively used in research and industry to design, model and for verification of electronic systems. There are again, numerous advantages of using verilog over other HDLs, some being ease of understanding, adaptability and comparatively lesser code.

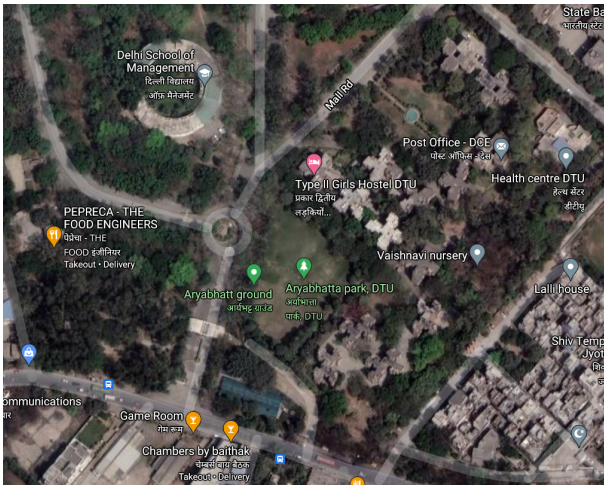
# CHAPTERS

# 1. INTRODUCTION & PROBLEM STATEMENT

Traffic control is a challenging problem in many cities. This is due to the large number of vehicles and the high dynamics of the traffic system. Poor traffic systems are the big reason for accidents, time losses. In this it will reduce the waiting time of the vehicles at traffic signals. A Traffic Light Control (TLC) system can be based on a microcontroller and microprocessor. There are numerous advantages of working with FPGAs instead of ASIC or microcontrollers for such an application, some being reduced cost, real time programmability and higher processing speeds.

Other advantages of this system is that the system senses emergency vehicles on the individual road moreover it gives priority to the traffic of that particular road where the emergency vehicles are sensed. In this, I am using an FPGA with traffic sensors to control traffic accordingly which means we can change the program if it requires and thus reduces the waiting time. The hardware design has been developed using Verilog Hardware Description Language (HDL) programming. The output of the system has been tested using Xilinx Vivado.

I drew inspiration from the traffic congestion in front of the Main Gate of our university, Delhi Technological University. The said place constitutes a 3-way road junction. I have first developed a novel sensor based TLC and compared the same with traditional timer based TLCs. Further to expand our study I have designed systems and built models for a Highway and State road junction and also a 4-way junction.


## 2. METHODOLOGY

The methodology followed includes first developing a traffic signalling model i.e., designing the various stages where which particular signal will be red which will be yellow or green and the timing is studied for the most efficient system. Then the above model is translated into a State table and state diagram representing each state and the flow of the system.

Next this finite state machine is modelled using Verilog HDL in Xilinx Vivado where it is simulated and the outcomes are studied to further make the design more efficient. Through this project FSM, System Design and Verilog HDL skills were acquired and implemented.

## 3. FINITE STATE MACHINES

A synchronous sequential circuit is also called a Finite State Machine (FSM) if it has a finite number of states.

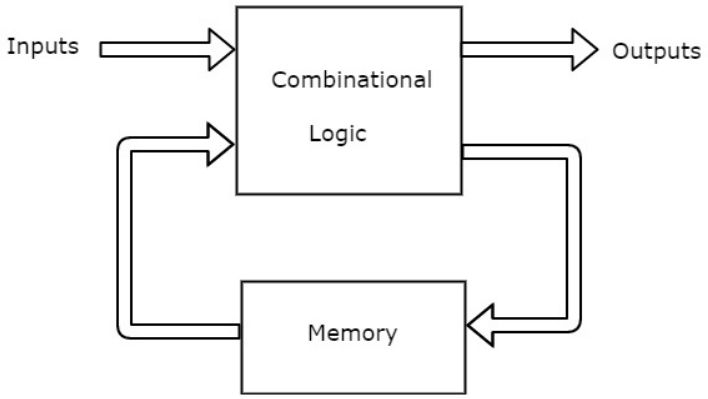
There are two types of FSMs.

- Mealy State Machine
- Moore State Machine

It is a machine that can be in exactly one of a finite number of *states* at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a *transition*. An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition

## 3. (a) MEALY STATE MACHINE

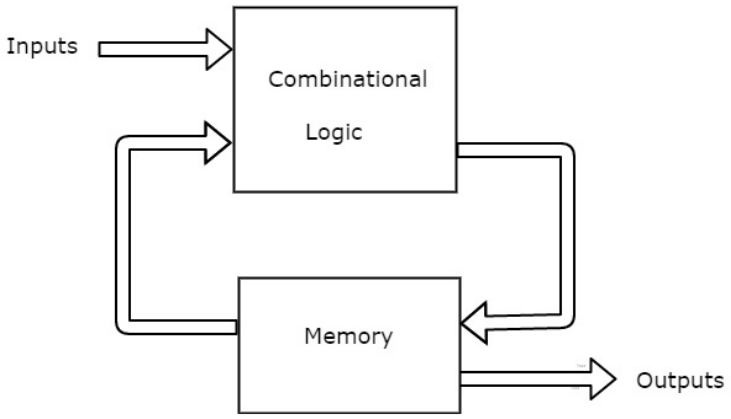
A Finite State Machine is said to be a Mealy state machine, if outputs depend on both present inputs & present states.




As shown in the figure, there are two parts present in the Mealy state machine. Those are combinational logic and memory. Memory is useful to provide some or part of previous outputs present states as inputs of combinational logic.

## 3.(b) MOORE STATE MACHINE

A Finite State Machine is said to be a Moore State Machine, if the outputs depend only on the present states. The block diagram of the Moore state machine is shown in the following figure.




A finite state machine is a model in which the states of the system are represented using a finite collection of ìmodes. The dynamics of a finite state machine are given by transitions between these modes, possibly in response to external signals.

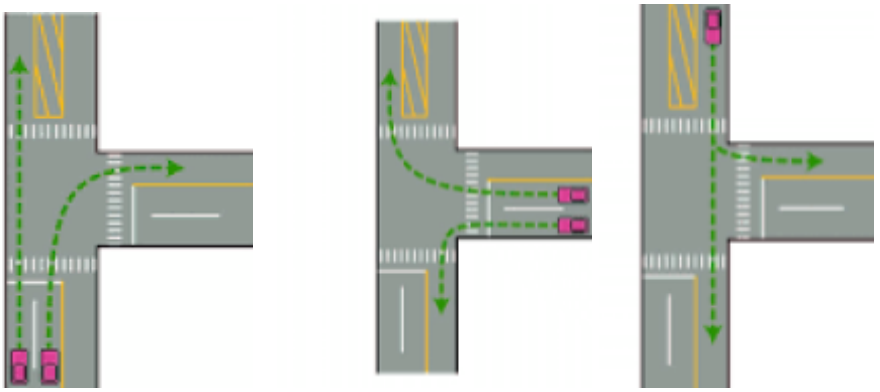
I have represented the state of the system in terms of the set of traffic lights that are turned on. In addition, once a light is turned on it should stay that way for a certain minimum time, and then only change when a car comes up to the intersection in the opposite direction. This gives us two states for each direction of the lights: waiting for a car to arrive and waiting for the timer to expire. Thus, we have four states for the system.

***Our system design is based on the Moore state machine, since our system will describe flow based on states and not inputs.***

# PRACTICAL CASE SCENARIOS

## 4. CASE - 1: 3 WAY JUNCTION

A 3 way junction is a T intersection. A particular example of such an intersection is the DTU main gate intersection with the Main Bawana Road. The possible traffic movements at a T intersection are shown in the figure below:



In the above figure, we can assume that the road on the right is the entrance to DTU.

This T intersection gives rise to 4 travel directions, i.e. signals. These are:

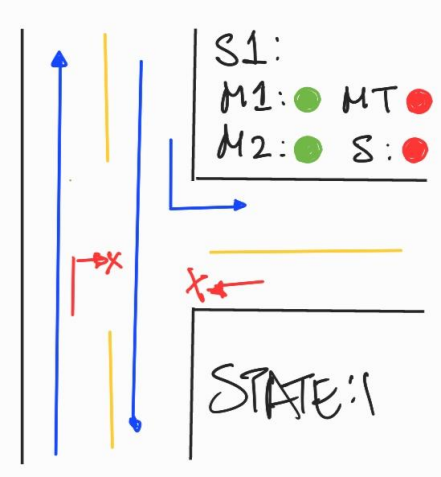
1. Main Road 1: This is the travel direction south to north
2. Main road Turn: This is the turn from main road to side road as shown in figure 1.
3. Main road 2: This is travel direction north to south
4. Side road: this is the turn from side road to north direction.

Our controller is designed to be able to do proper signalling at every moment for all 4 travel possibilities.

### 4.1 SYSTEM DESIGN & GRAPHICAL REPRESENTATION

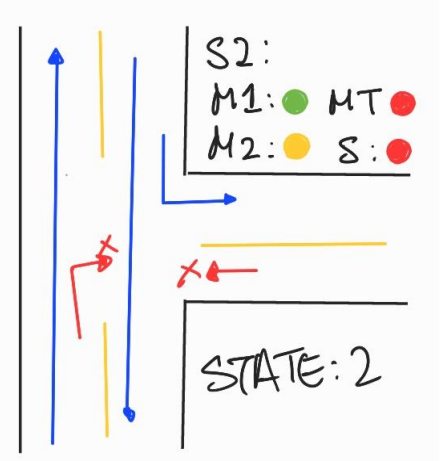
These 4 possibilities give rise to 6 states for traffic movement and signalling. These 6 states and the flow is described below along with pictorial representation.

1. **<u>STATE 1 (S1)</u>**: In this state the traffic is allowed to move from south to north. That is Main road 1 is green. Also traffic going north to south is allowed making Main road 2 also green.



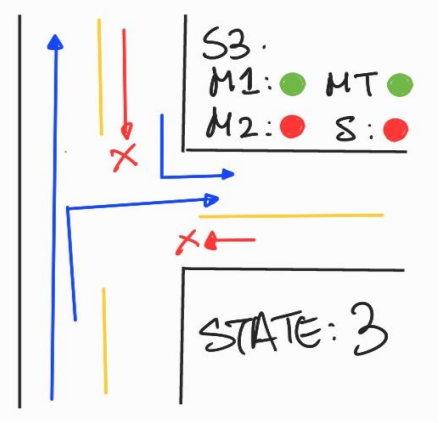

| | |
|---|---|
| Main road 1 | **Green** |
| Main road 2 | **Green** |
| Main road turn | **Red** |
| Side road turn | **Red** |

2. **<u>STATE 2 (S2)</u>**: In this state we transition to state 3. Therefore Main road 2 signal is turned yellow.



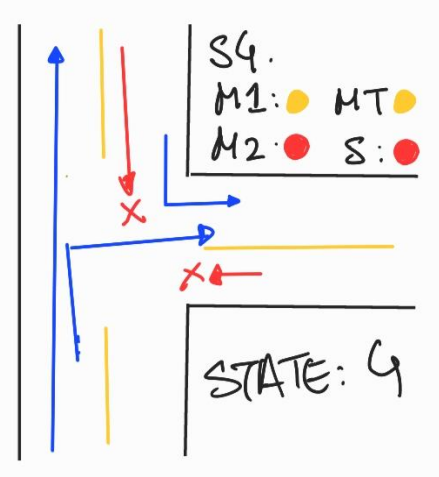

| | |
|---|---|
| Main road 1 | **Green** |
| Main road 2 | **Yellow** |
| Main road turn | **Red** |
| Side road turn | **Red** |

3. **<u>STATE 3 (S3):</u>** In this state Main road turn traffic is allowed. Hence traffic must be stopped from main road 2.



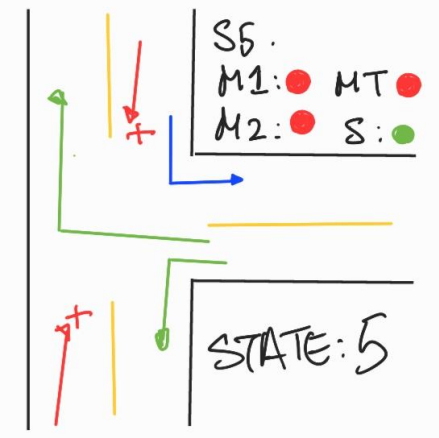

| | |
|---|---|
| Main road 1 | **Green** |
| Main road 2 | **Red** |
| Main road turn | **Green** |
| Side road turn | **Red** |

4. **STATE 4 (S4):** This is a transition state to state 5. Here we turn both main road 1 and main road turn signals yellow to finally allow traffic from side turn to flow in the next state.



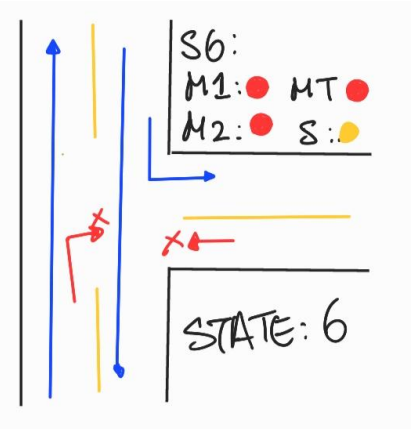

| | |
|---|---|
| Main road 1 | **Yellow** |
| Main road 2 | **Red** |
| Main road turn | **Yellow** |
| Side road turn | **Red** |

5. **STATE 5 (S5):** This state allows traffic from the side road to merge into the main road. The signals are:




| | |
|---|---|
| Main road 1 | **Red** |
| Main road 2 | **Red** |
| Main road turn | **Red** |
| Side road turn | **Green** |

6. **STATE 6 (S6):** This is the final state after which the system will again flow back to state 1. The side road signal is changed to yellow to move on to state 1 for movement of main road traffic.




| | |
|---|---|
| Main road 1 | **Red** |
| Main road 2 | **Red** |
| Main road turn | **Red** |
| Side road turn | **Yellow** |

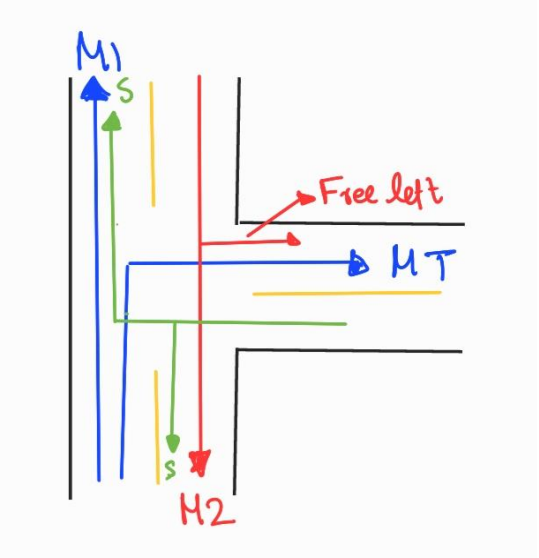
After state 6 the system then moves to state 1 and the cycle is completed. The system flows through various states with time delays for each state. The time delays are as follow:

1. Time for State 1, Main roads green= 7 seconds (TMG)
2. Time for State 2, Main road 2 yellow= 2 seconds (TMY)(TY)
3. Time for State 3, Main road turn green = 5 seconds (TTG)
4. Time for State 4, Main road turn yellow= 2 seconds (TTY)(TY)
5. Time for State 5, Side road green = 5 seconds (TSG)
6. Time for State 6, Side road yellow= 2 seconds (TSY)(TY)

The time delays are decided to give more time for main road traffic to flow as compared to Side road and yellow signal is 2 seconds for every road for proper intimation of signal turning red.

Each traffic direction to be signalled can be named as follows:

1. Main road 1: M1
2. Main road 2: M2
3. Main road Turn: MT
4. Side road: S




The states represent the general flow signals that are from green to yellow to red to green.

## 4.2 STATE DIAGRAM AND STATE TABLE

From the above descriptions of states and the time delays, i.e, using the above sequence of events and pseudo-code we can make a state graph/diagram as follow:

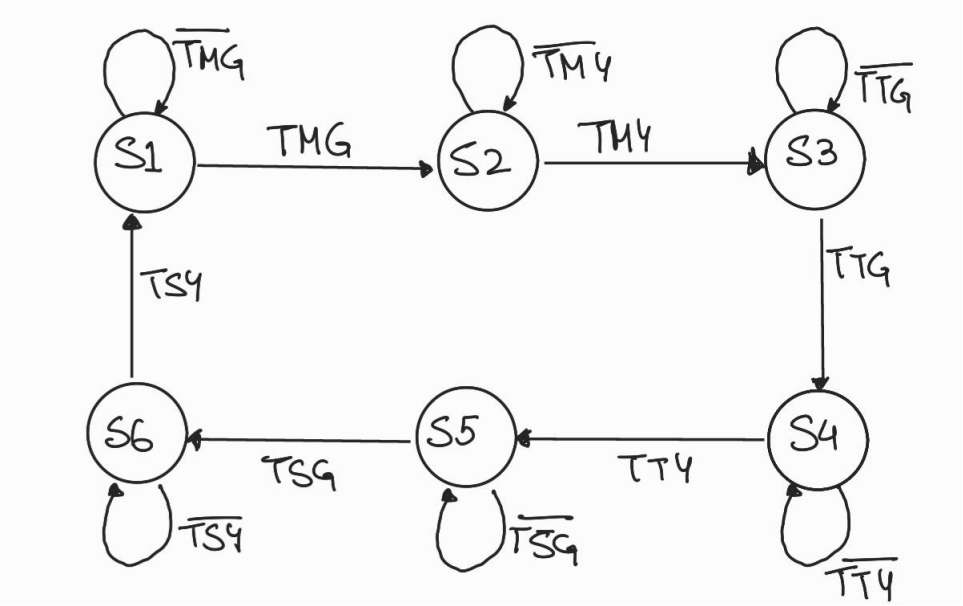



FIGURE: MOORE STATE GRAPH

The above state diagram can be explained as follows- if TMG is false, i.e. $\overline{TMG}$=1, the system remains in state S1, but if TMG, i.e. $TMG$=1 is true then the system moves to state S2. This happens through every state. TMG remains false till that many clock cycles have elapsed. A system clock or counter is used to calculate the various clock cycles for the time delays.

## 4.3 CIRCUIT IMPLEMENTATION

As the total number states is 6 we need 3 variables to switch between states ($2^3$=8>6). The number timer inputs are 4 (TMG, TY, TTG, TSG) and 3 variables for the state, hence total inputs are 7. The number of lights are 4 in total, one for each road, then total outputs are 12 (4x3) along with 3 outputs for state variables and one for clock reset.

Hence we need a combinational circuit with 7 inputs and 16 outputs, with 3 flip-flops one for each state variable for memory purposes.

From our knowledge thus far, the combinational logic can be implemented using logic gates, multiplexers, PROMs or Programmable Array Logic.

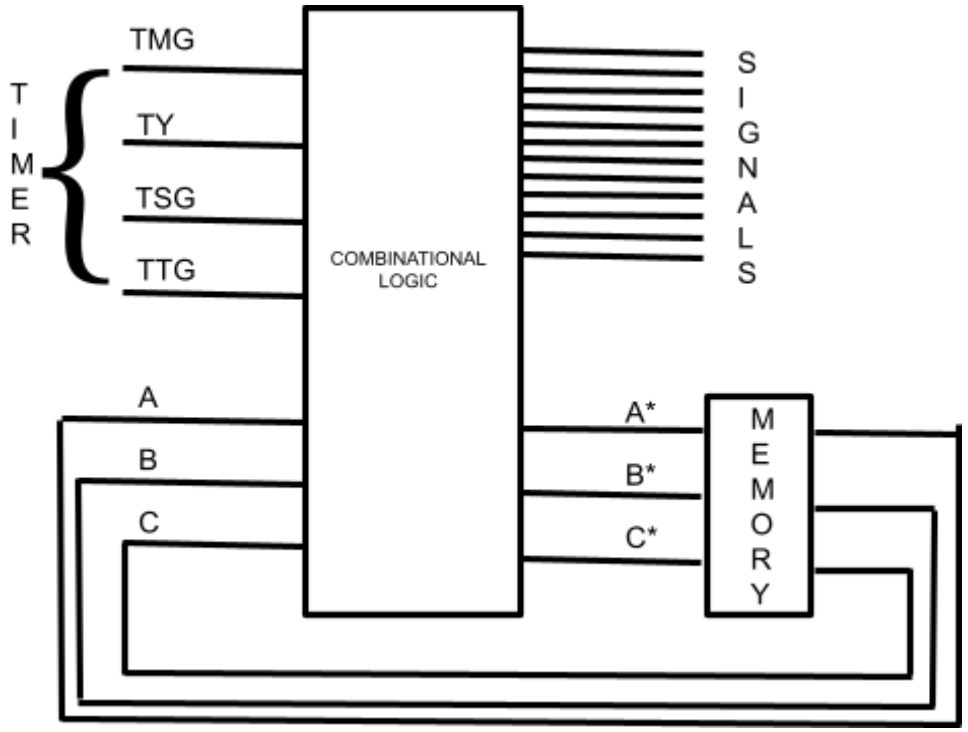



FIGURE: DRAWING OF LOGIC CIRCUIT

The combinational circuit can be a PROM of the size $2^7$x16 bits and memory can be implemented using 3 D-Flip Flops. The timer inputs will come from another circuit called Timer:

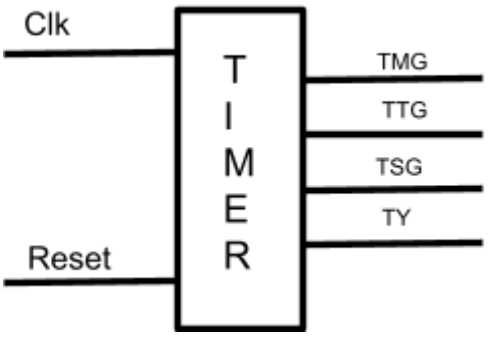



FIGURE: TIMER CIRCUIT

We can code each of the states using 3 variables as follows:

S1=000, S2=001, S3=010, S4=100, S5=101, S6=110

And each Red, Yellow Green can be coded for ease as red=100, Yellow=010 and Green=001. The reset timer tells us when to reset the clock back to zero after one particular time interval has passed.

Hence the state table can be made using the above information.

| **Present State** | **Reset** | **Input** | **Next State** | **M1** | **M2** | **T** | **S** |
|---|---|---|---|---|---|---|---|
| 000 | 0 | $\overline{TMG}$ | 000 | 001 | 001 | 100 | 100 |
| | 1 | $TMG$ | 001 | | | | |
| 001 | 0 | $\overline{TY}$ | 001 | 001 | 010 | 100 | 100 |
| | 1 | $TY$ | 010 | | | | |
| 010 | 0 | $\overline{TTG}$ | 010 | 001 | 100 | 001 | 100 |
| | 1 | $TTG$ | 100 | | | | |
| 100 | 0 | $\overline{TY}$ | 100 | 010 | 100 | 010 | 100 |
| | 1 | $TY$ | 101 | | | | |
| 101 | 0 | $\overline{TSG}$ | 101 | 100 | 100 | 100 | 001 |
| | 1 | $TSG$ | 110 | | | | |
| 110 | 0 | $\overline{TY}$ | 110 | 100 | 100 | 100 | 010 |
| | 1 | $TY$ | 000 | | | | |

**TABLE: STATE TABLE FOR THE SYSTEM**

The reset variable changes the timer output from $\overline{TMG}$ to $TMG$, and similarly for other time values.

## 4.4 CODE AND SIMULATION RESULTS

In the code I have used 2 inputs, clock and reset that will generate the 4 timer signals. I have used 4 output reg variables that ***hold*** values for lights for road M1, M2, MT, S. count is the output of the timer that keeps track of time delays in each state.

The following is the :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module Traffic_Light_Controller(
    input clk,rst,
    output reg [2:0]light_M1,
    output reg [2:0]light_S,
    output reg [2:0]light_MT,
    output reg [2:0]light_M2
    );

    parameter  S1=0, S2=1, S3 =2, S4=3, S5=4,S6=5;
    reg [3:0]count;
    reg[2:0] ps;
    parameter  sec7=7,sec5=5,sec2=2,sec3=3;
    always@(posedge clk or posedge rst)
        begin
        if(rst==1)
        begin
        ps<=S1;
        count<=0;
        end
        else
            case(ps)
                S1: if(count<sec7)
                        begin
                        ps<=S1;
                        count<=count+1;
                        end
                    else
                        begin
                        ps<=S2;
                        count<=0;
                        end
                S2: if(count<sec2)
                        begin
                        ps<=S2;
                        count<=count+1;
                        end
                    else
```

```
                begin
                ps<=S3;
                count<=0;
                end
        S3: if(count<sec5)
                begin
                ps<=S3;
                count<=count+1;
                end
            else
                begin
                ps<=S4;
                count<=0;
                end
        S4:if(count<sec2)
                begin
                ps<=S4;
                count<=count+1;
                end
            else
                begin
                ps<=S5;
                count<=0;
                end
        S5:if(count<sec3)
                begin
                ps<=S5;
                count<=count+1;
                end
            else
                begin
                ps<=S6;
                count<=0;
                end

        S6:if(count<sec2)
                begin
                ps<=S6;
                count<=count+1;
                end
            else
                begin
                ps<=S1;
                count<=0;
                end
        default: ps<=S1;
        endcase
    end

    always@(ps)
    begin
      case(ps)
```

```
                S1:
                begin
                   light_M1<=3'b001;
                   light_M2<=3'b001;
                   light_MT<=3'b100;
                   light_S<=3'b100;
                end
                S2:
                begin
                   light_M1<=3'b001;
                   light_M2<=3'b010;
                   light_MT<=3'b100;
                   light_S<=3'b100;
                end
                S3:
                begin
                   light_M1<=3'b001;
                   light_M2<=3'b100;
                   light_MT<=3'b001;
                   light_S<=3'b100;
                end
                S4:
                begin
                   light_M1<=3'b010;
                   light_M2<=3'b100;
                   light_MT<=3'b010;
                   light_S<=3'b100;
                end
                S5:
                begin
                   light_M1<=3'b100;
                   light_M2<=3'b100;
                   light_MT<=3'b100;
                   light_S<=3'b001;
                end
                S6:
                begin
                   light_M1<=3'b100;
                   light_M2<=3'b100;
                   light_MT<=3'b100;
                   light_S<=3'b010;
                end
                default:
                begin
                   light_M1<=3'b000;
                   light_M2<=3'b000;
                   light_MT<=3'b000;
                   light_S<=3'b000;
                end
                endcase
            end
endmodule
```

The testbench for the above verilog simulation is as follows:

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
module Traffic_Light_Controller_TB;
    reg clk,rst;
    wire [2:0]light_M1;
    wire [2:0]light_S;
    wire [2:0]light_MT;
    wire [2:0]light_M2;
    Traffic_Light_Controller dut(.clk(clk) , .rst(rst) , .light_M1(light_M1) ,
.light_S(light_S)  ,.light_M2(light_M2),.light_MT(light_MT)   );

initial begin
    clk=1'b0;
    forever #(1000000000/2) clk=~clk;
end
//    initial
//    $stop;//to add ps
initial begin
    rst=0;
    #1000000000;
    rst=1;
    #1000000000;
    rst=0;
    #(1000000000*2000);
    $finish;
end

endmodule
```

I have initialized the 4 light signal outputs as wires here as we do not need to hold the values after passing them. Here *dut* acts as the bridge link between the testbench and the verilog code.

In the simulation result I have coded each signal's output as red, yellow and green. Using a 3 bit representation for just 3 variables has the benefit that each wave can be encoded as unique, that is, 100, 010, 001 have ones in different positions. Hence the simulation clearly depicts when the red signal is turned, when yellow and when green.

Each time the reset variable changes value the clock is resting as the needed amount of clock cycles have elapsed and hence the system moves to the new cycle. A 0.5 second pre-state delay has been added in the beginning which can represent switching on time of the system.

***The simulation results are as below:***

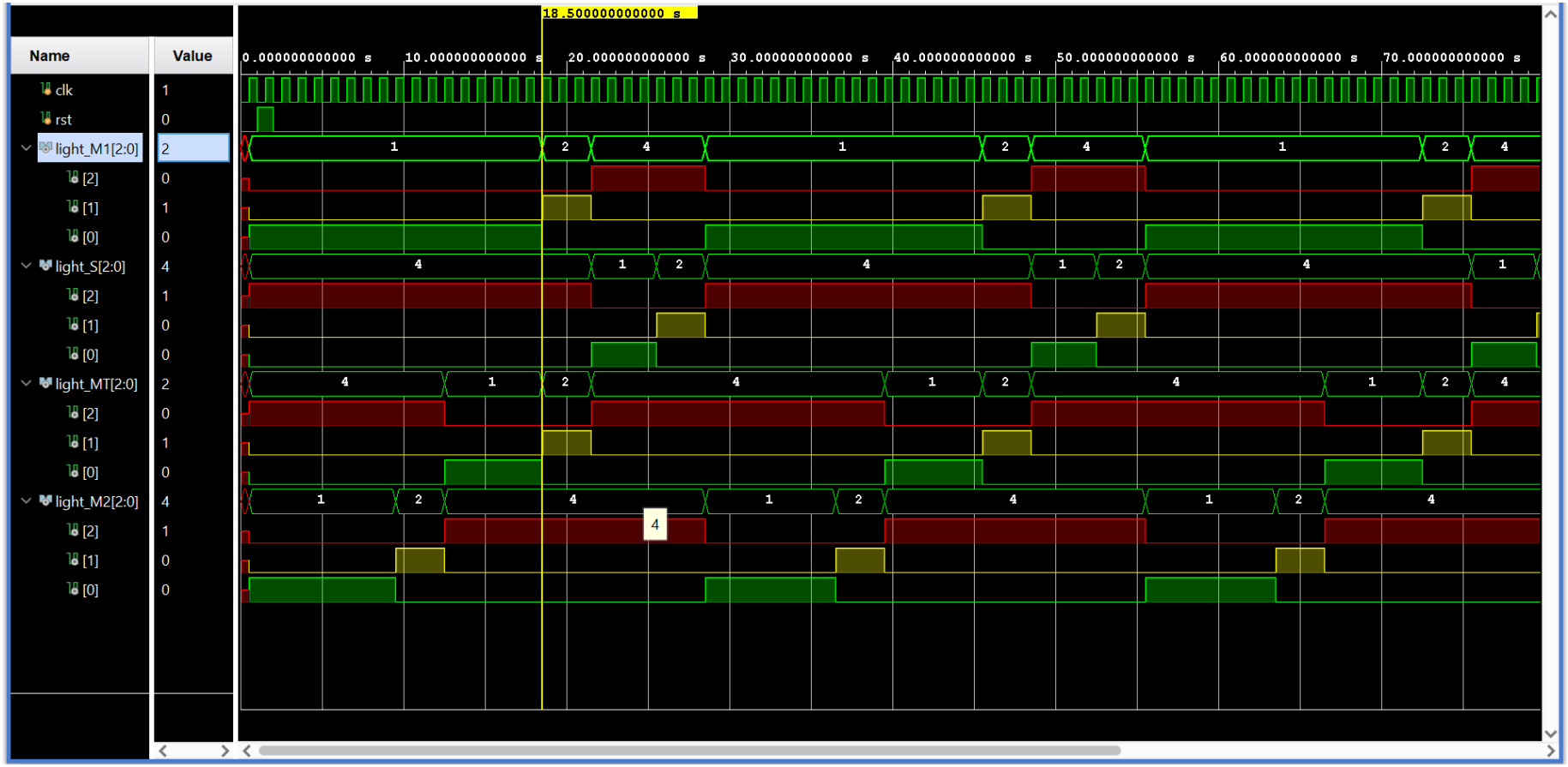



FIGURE: Simulation results at 18 seconds

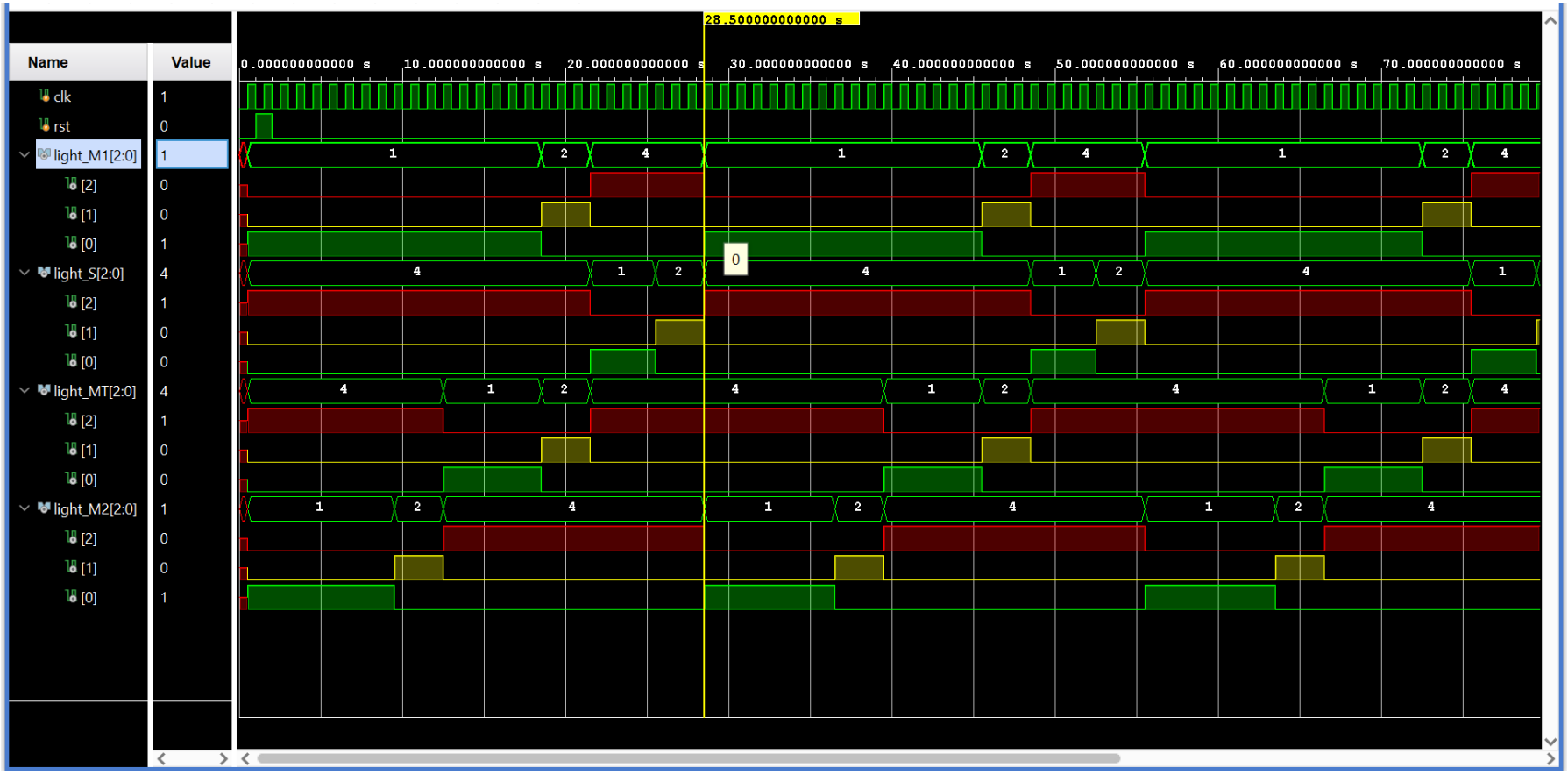



FIGURE: SIMULATION RESULTS AT 28 SECONDS

## 4.4 ANALYSIS AND IMPLEMENTATION

The system was successfully coded and simulated in verilog using Xilinx Vivado environment. As it is visible from the output graphs, the system works as planned and designed.

The Xilinx Vivado software provides us with the functionality to obtain RTL based schematics from designed systems. The following is the said schematic for our system:

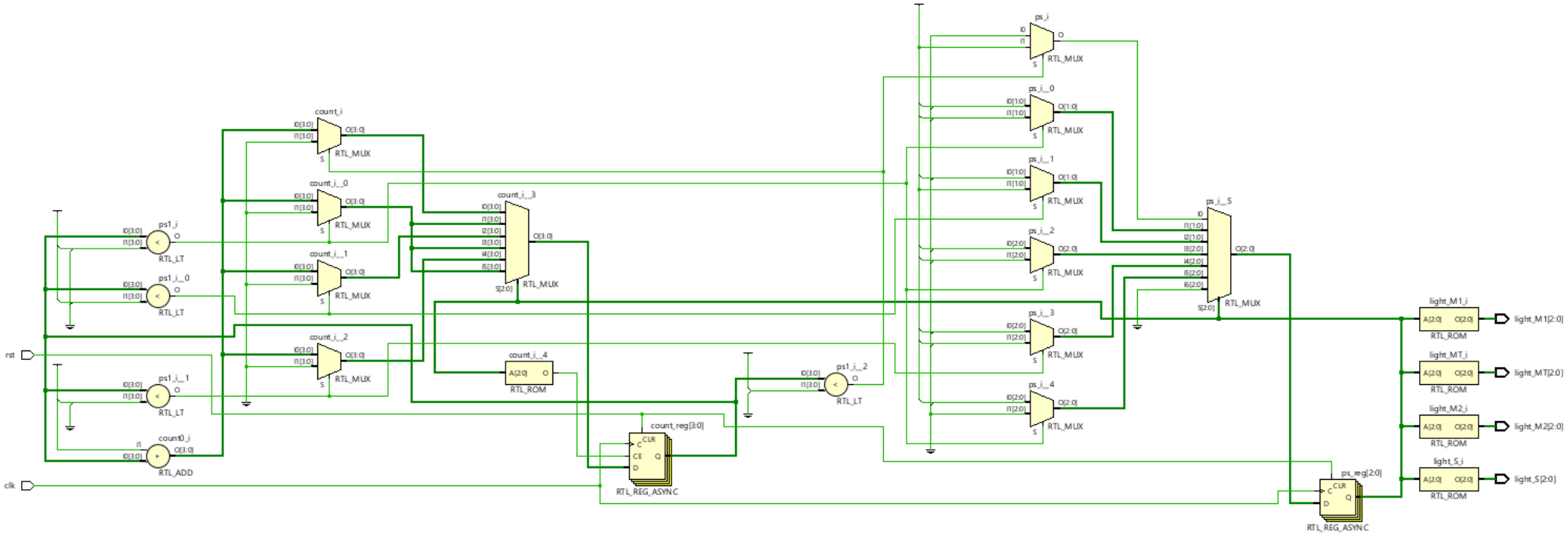



FIGURE: RTL SCHEMATIC OBTAINED

It can be observed that the combinational circuit has been designed using Multiplexers in the above schema. This can be understood as follows:

If we construct a mux base circuit for switching A*, we need a 8x1 mux with 3 select lines as A B C, whose combination represents present state, A=A* remains zero for the first 3 states , till TTG is zero. When TTG is 1, A* becomes 1. After changing to 1, A* changes to zero only when TY is low, so it remains 1 for the next 3 states. Hence we can implement this with a mux as follows:

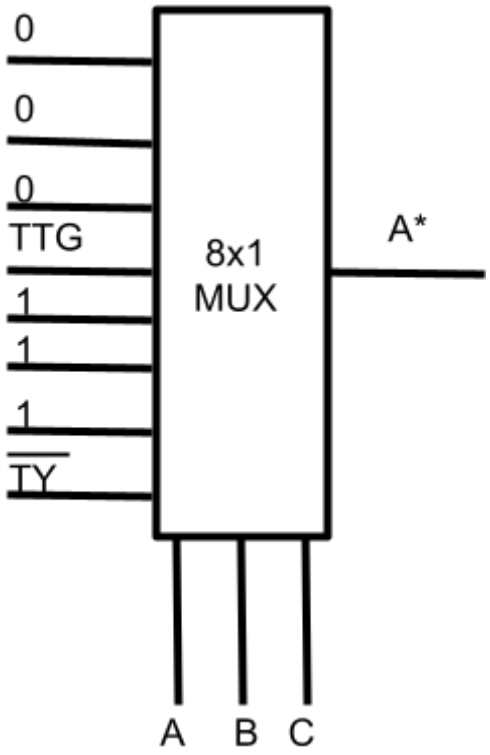



FIGURE: MUX IMPLEMENTATION TO OBTAIN A*

Similarly all inputs and outputs can be constructed with mux. To store previous state variables, we use D flip-flops to store A*, B*, C* in memory to provide previous state information for deciding the next state.

In the next case we study and design a system using sensors to sense the presence of cars instead of using time delays. This is done for higher efficiency.

## 5. CASE - 2: 3-WAY JUNCTION WITH SENSORS

In this case a sensor is placed on the side road. Hence lights for side roads will only turn green upon sensing a vehicle. Therefore lights for the main road will not turn red without any traffic flow from the side road. This increases traffic flow on the main road reducing time wasted and hence increases the efficiency of the model.

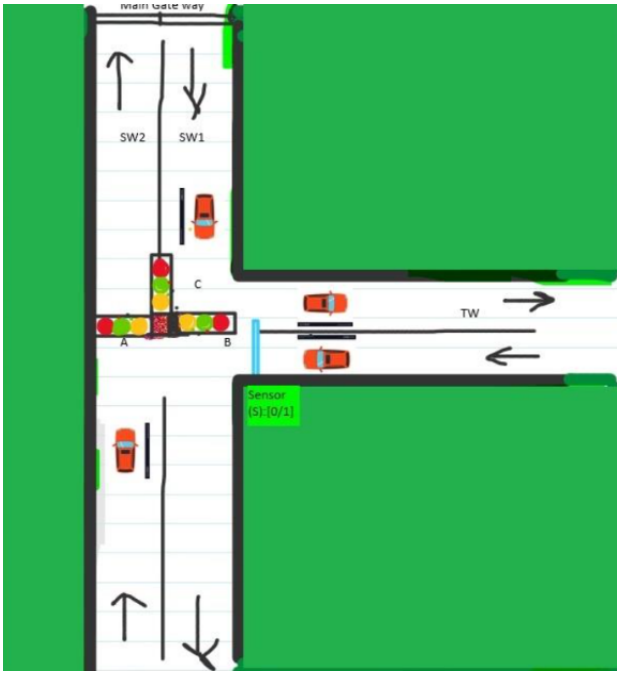



FIGURE: GRAPHICAL REPRESENTATION OF SENSOR PLACEMENT

A simple visualization of flow of events for sensor based:

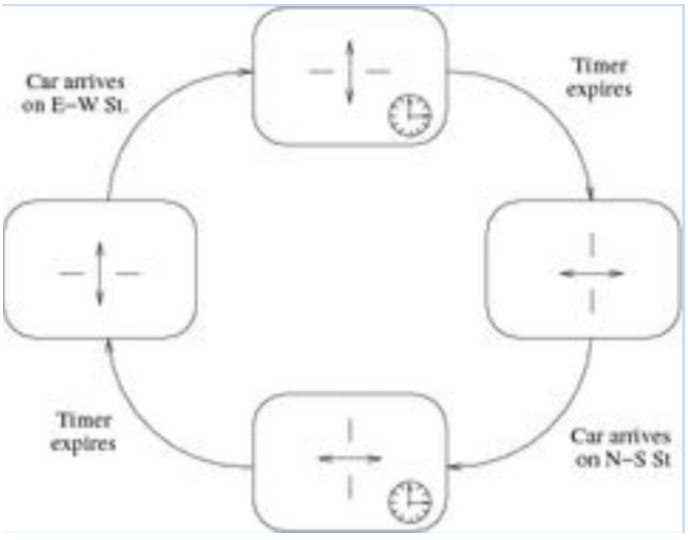



FIGURE: FLOW OF EVENTS

In this case, if the sensor senses a vehicle, its output changes to true, i.e. from 0 to 1. When this happens after a certain time delay the system moves into the state where it turns all main road lights to yellow so that ongoing traffic can stop. After this state, all main road lights are turned red and for a set time side road light is turned green. After the set time expires, the system again moves into main road traffic and main road turning traffic.

## 5.(a) CODE AND SIMULATION RESULTS

Here we have a special input as a sensor which will be simulated through test bench code.

```
`timescale 1ms / 1ns
//////////////////////////////////////////////////////
module sensorThreeRd(ML1,ML1T, ML2, SR,SENSOR,clk,second);
parameter S0=3'b000,//G,R,G,R
          S1=3'b001,//G,R,Y,R
          S2=3'b010,//G,G,R,R
          S3=3'b011,//Y,Y,R,R
          S4=3'b100,//R,R,R,G
          S5=3'b101,//R,R,R,Y
          S6=3'b110;//G,Y,R,R
input SENSOR, //side road  sensor
    clk;
output reg[2:0] ML1,
                ML1T,
                ML2,
                SR;
reg[2:0] state,next_state; //to change the lights
output reg[7:0] second=-1;
initial state<=2'b000;
initial ML1<=3'b001;
initial ML1T<=3'b100;
initial ML2<=3'b001;
initial SR<=3'b100;
always@(posedge clk)begin
    state<=next_state; //to change state to next state
end
always @(*) begin //switch between states
    case(state)
        S0:
            begin
               ML1<=3'b001;
               ML1T<=3'b100;
               ML2<=3'b001;
               SR<=3'b100;
               if((SENSOR==1))begin
                       next_state<=S3;end
```

```
     else begin
           next_state<=S1;end
     end
 S1:
    begin
        ML1<=3'b001;
        ML1T<=3'b100;
        ML2<=3'b010;
        SR<=3'b100;
        if(second==3) begin
            next_state<=S2;end
    end
  S2:
    begin
        ML1<=3'b001;
        ML1T<=3'b001;
        ML2<=3'b100;
        SR<=3'b100;
        #4000;
        if(SENSOR==1) begin
            next_state<=S3;end
        else begin
            #500;
            next_state<=S6;end
      end
S3:
    begin
        ML1<=3'b010;
        ML1T<=3'b010;
        ML2<=3'b100;
        SR<=3'b100;
        if(second==2)begin
             next_state<=S4;
             end
    end
S4:
    begin
        ML1<=3'b100;
        ML1T<=3'b100;
        ML2<=3'b100;
        SR<=3'b001;
        if(second==6)begin
            next_state<=S5;end
    end
S5:
    begin
        ML1<=3'b100;
        ML1T<=3'b100;
        ML2<=3'b100;
        SR<=3'b010;
        if(second==2) begin
```

```
                    next_state<=S0;end
            end
          S6:
            begin
                ML1<=3'b001;
                ML1T<=3'b010;
                ML2<=3'b100;
                SR<=3'b100;
                if (second==2)begin
                    next_state<=S0;end
            end
         default state<=S0;
         endcase
 end

 always@(posedge clk)begin
    if(state!=next_state)begin
        state<=next_state;
        second<=0;end
     else begin
        second<=second+1;end
 end
 endmodule
```

The testbench for the above verilog simulation is as follows:

```
`timescale 1ms / 1ns
////////////////////////////////////////////////////////////////////////////////

module TB_sensorThreeRd();
reg SENSOR,clk;
wire[2:0] main_road1;
wire[2:0] main_road1T;
wire[2:0] main_road2;
wire[2:0] side_road;
wire[7:0] count;

sensorThreeRd DUT(main_road1,main_road1T,main_road2,side_road,SENSOR,clk,count);

initial
    begin                              //Change clock every 500ms
       clk=1; forever #500 clk=~clk; //so a cycle is 1s
    end
initial
    begin                                          //Change SENSOR every
        SENSOR=0; forever #40000 SENSOR=~SENSOR;    //80 seconds
    end
endmodule
```

For simulation purposes we change values for sensors periodically using clocks.

***The simulation results are below:***


FIGURE: SIMULATION RESULT WHEN SENSOR TURNS ON

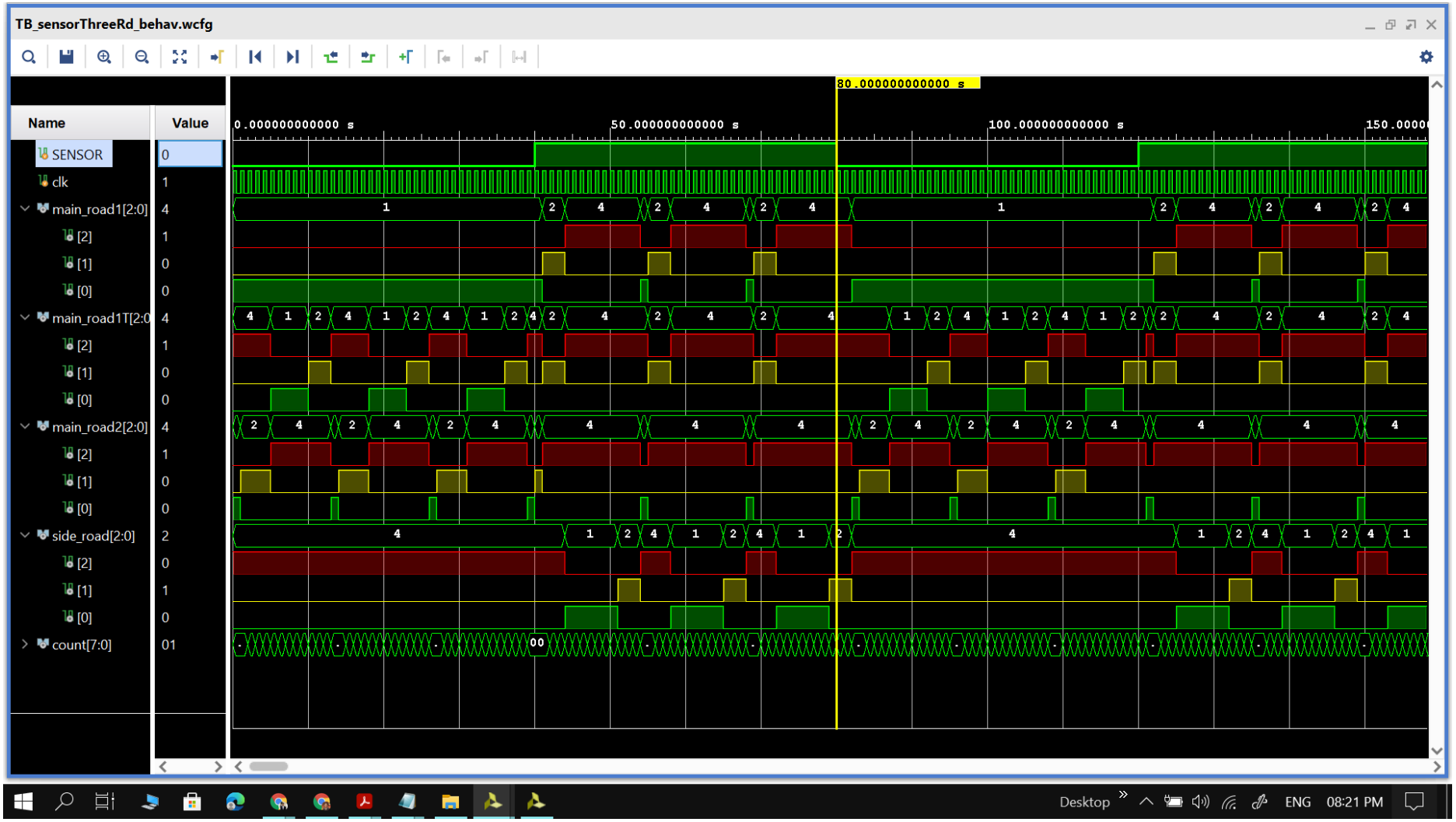



FIGURE: SIMULATION RESULT WHEN SENSOR TURNS OFF

## 5.2 ANALYSIS AND IMPLEMENTATION

The system runs as desired and designed. Whenever the sensor turns on, that is at 40 seconds in our program, after a slight delay main road lights turn yellow and then red. Once the main road lights turn red the side road lights turn green, but only for a preset time. This is done to avoid congestion on the main road for a long duration (if the sensor remains on for very long), as the main road has a higher priority.

This makes the system much more efficient than a simple time based system, like in the previous case. We propose the application of our robust sensor based digital traffic controller for smooth traffic flow and to avoid unwanted congestion.

The RTL schematic for the system is given as below:

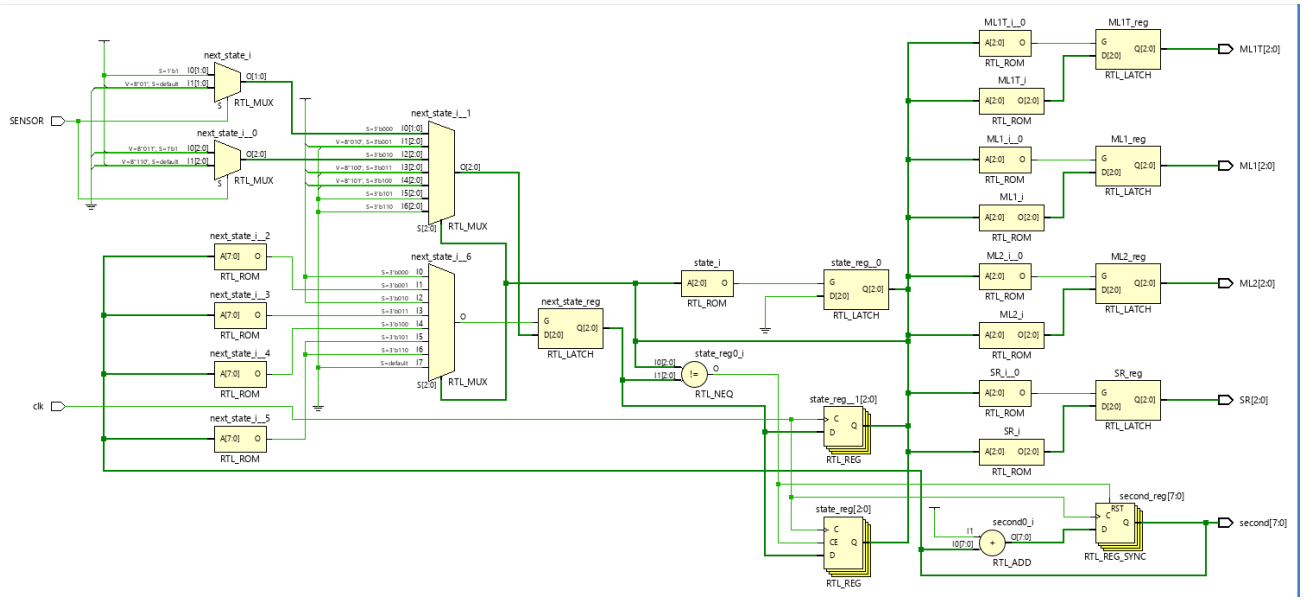



FIGURE: RTL SCHEMATIC FOR SENSOR BASED CONTROLLER

## 6. CASE - 3: HIGHWAY & COUNTRY ROAD JUNCTION

In this case we study an intersection between 2 roads, one being a Highway with heavy traffic flow and higher priority and another state road with less frequent traffic and lower priority.

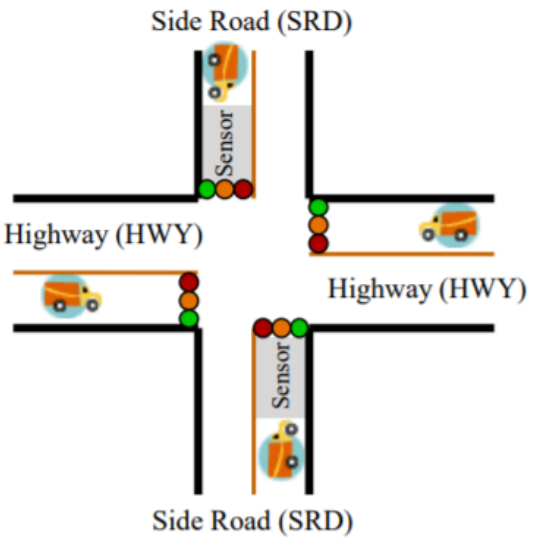


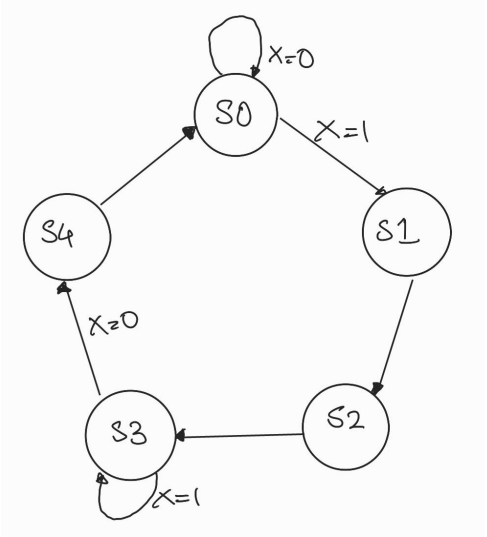

FIGURE: JUNCTION

In this system we have extended our previous design to have a 4 way junction. The sensor is placed on the low priority side road. The sensor, usually off, permits the traffic on the highway to pass and whenever the sensor value turns on the main road signal is first turned yellow and then red for a predefined amount of time.

The following states must be considered for this system

1. **State 0(S1):** Here Highway road is green and country road is red
2. **State 1(S2):** Here Highway road is yellow and country road is red
3. **State 2(S3):**Here Highway road is red and country road is red
4. **State 3(S4):** Here Highway road is red and country road isgreen
5. **State 4(S5):** Here Highway road is red and country road is yellow

The following is the state graph for Case - 3:




Here X represents the sensor value. The sensor sends a signal X as input to the controller. X = 1 if there are cars on the country road, otherwise, X= 0.

## 6.(a) CODE AND SIMULATION RESULTS

Here we have a special input as a sensor which will be simulated through test bench code.

```
`timescale 1s / 1ms
////////////////////////////////////////////////////////////////////////////////

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

module HC_trafficController
 (hwy, cntry, X, clock, clear);

output [1:0] hwy, cntry; //2-bit output for 3 states of signal
reg [2:0] hwy, cntry; //declared output signals are registers
input X; //if TRUE, indicates that there is car on
input clock, clear;
parameter RED = 3'b100,
 YELLOW = 3'b010,
 GREEN = 3'b001;
parameter S0 = 3'd0,
 S1 = 3'd1,
 S2 = 3'd2,
 S3 = 3'd3,
 S4 = 3'd4;
```

```
reg [2:0] state;
reg [2:0] next_state;
//state changes only at positive edge of clock
always @(posedge clock)
 if (clear)
 state <= S0; //Controller starts in S0 state
 else
 state <= next_state; //State change
always @(state)
begin
 hwy = GREEN; //Default Light Assignment for Highway light
 cntry = RED; //Default Light Assignment for Country light
 case(state)
 S0: ; // No change, use default
 S1: hwy = YELLOW;
 S2: hwy = RED;
 S3: begin
 hwy = RED;
 cntry = GREEN;
 end
 S4: begin
 hwy = RED;
 cntry = YELLOW;
 end
 endcase
end
always @(state or X)
begin
 case (state)
 S0: if(X)
 next_state = S1;
 else
 next_state = S0;
 S1: begin //delay some positive edges of clock
 next_state = S2;
 end
 S2: begin //delay some positive edges of clock
 next_state = S3;
 end
 S3: if(X)
 next_state = S3;
 else
 next_state = S4;
 S4: begin //delay some positive edges of clock
 next_state = S0;
 end
 default: next_state = S0;
 endcase
end
endmodule
```

The testbench for the above verilog simulation is as follows:

```
`timescale 1s / 1ms

module TB_HC_trafficController;
wire [2:0] MAIN_SIG, CNTRY_SIG;
reg CAR_ON_CNTRY_RD;
reg CLOCK, CLEAR;

HC_trafficController SC(MAIN_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD, CLOCK, CLEAR);

initial
 $monitor($time, " Main Sig = %b Country Sig = %b Car_on_cntry = %b",
 MAIN_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD);

initial
begin
 CLOCK = `FALSE;
 forever #5 CLOCK = ~CLOCK;
end

initial
begin
 CLEAR = `TRUE;
 repeat (5) @(negedge CLOCK);
 CLEAR = `FALSE;
end

initial
begin
 CAR_ON_CNTRY_RD = `FALSE;
 repeat(20)@(negedge CLOCK); CAR_ON_CNTRY_RD = `TRUE;
 repeat(10)@(negedge CLOCK); CAR_ON_CNTRY_RD = `FALSE;
 repeat(20)@(negedge CLOCK); CAR_ON_CNTRY_RD = `TRUE;
 repeat(10)@(negedge CLOCK); CAR_ON_CNTRY_RD = `FALSE;
 repeat(20)@(negedge CLOCK); CAR_ON_CNTRY_RD = `TRUE;
 repeat(10)@(negedge CLOCK); CAR_ON_CNTRY_RD = `FALSE;
 repeat(10)@(negedge CLOCK); $stop;
end
endmodule
```

The simulation results are as follows:

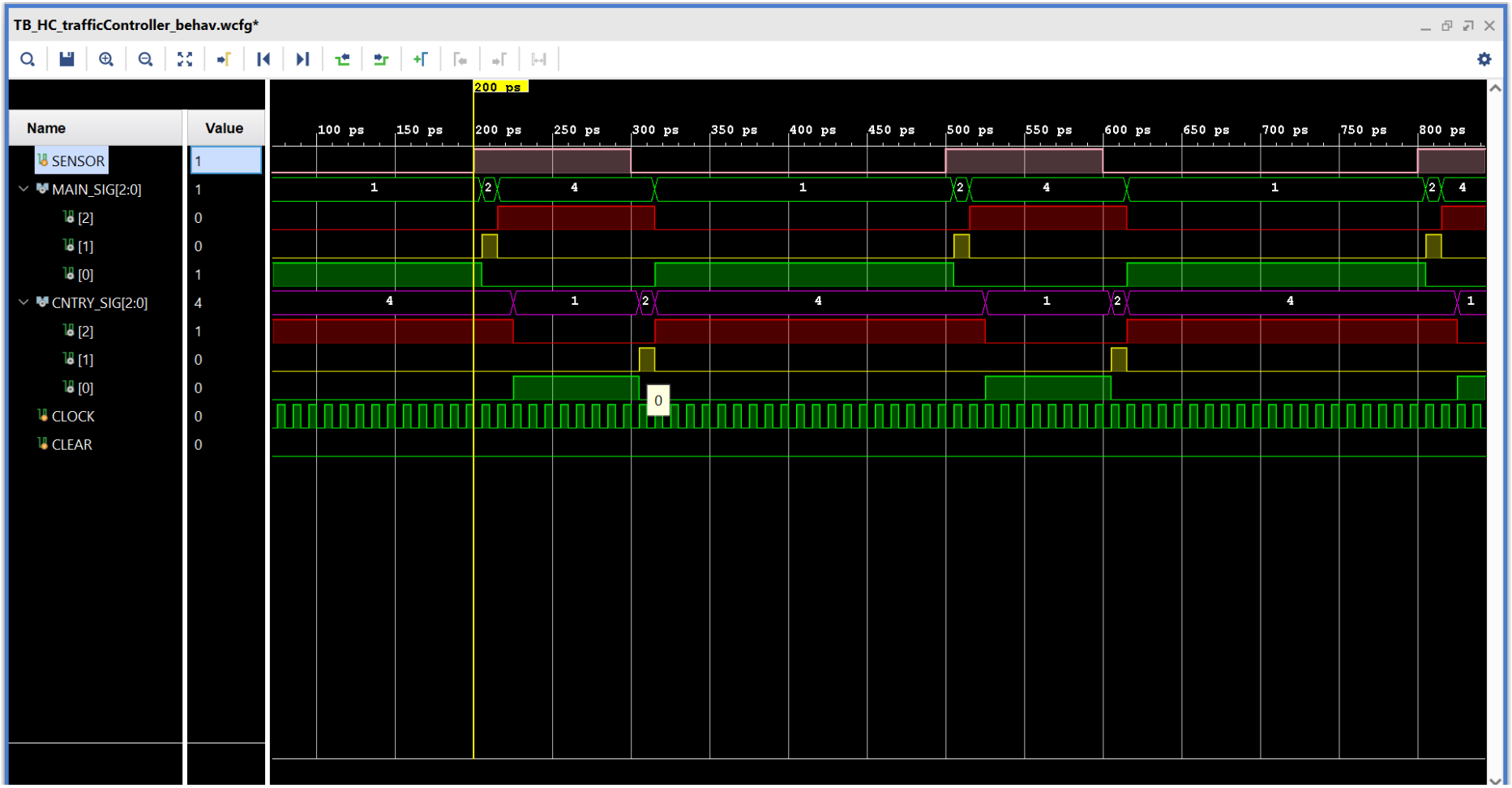



FIGURE: RESULTS WHEN SENSOR TURNS ON

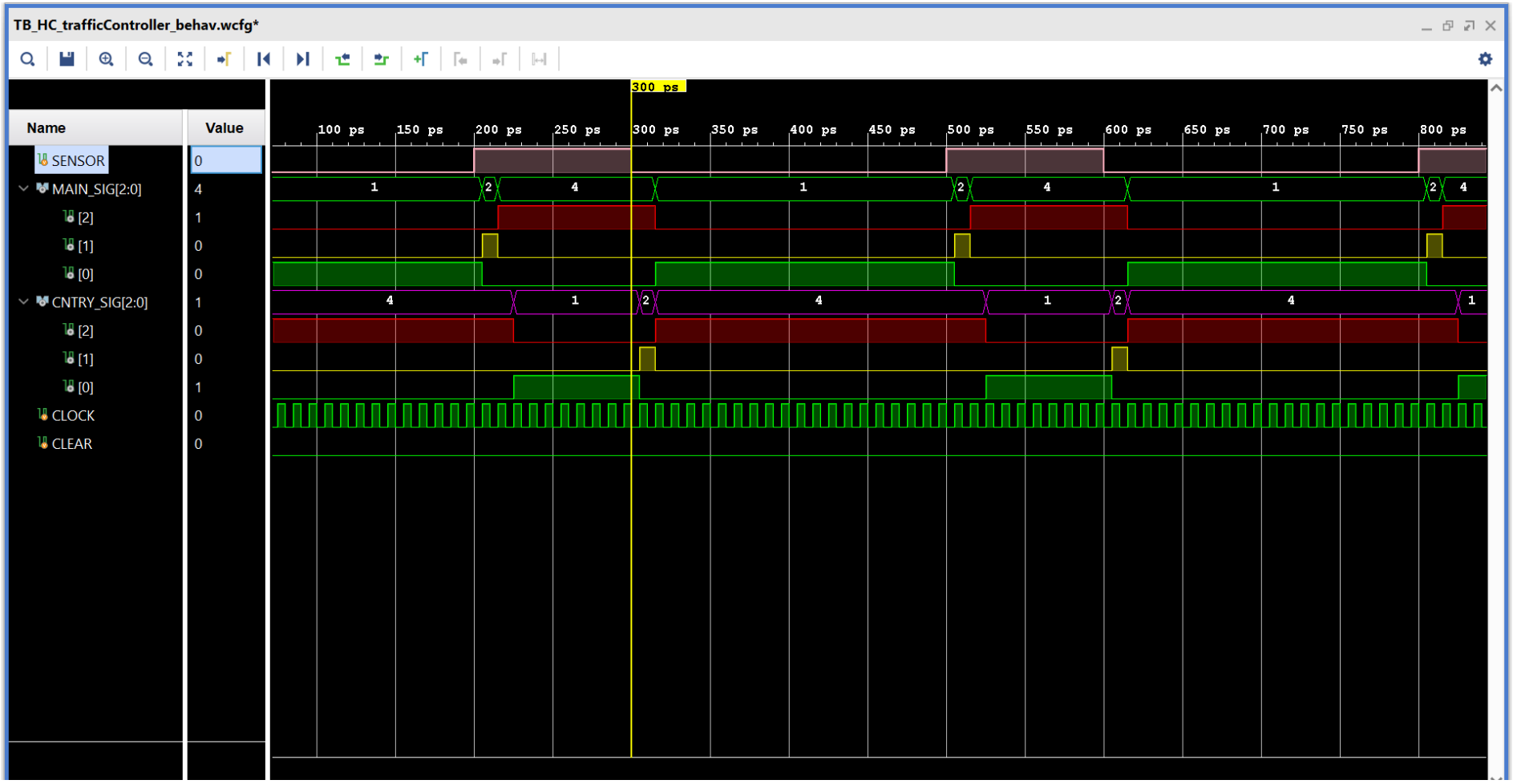



FIGURE: RESULTS WHEN SENSOR TURNS OFF

## 6.(b) ANALYSIS AND IMPLEMENTATION

The system runs as desired and designed. Whenever the sensor turns on, as can be clearly seen the main road lights turn first yellow for a short time and then turn red. Simultaneously the side road lights turn to green once the main road light has turned red.

As the Highway woad has higher priority by default it is green. Also The green light is on for a longer duration for highway road due to more traffic flow. Hence the Red light for highway road is a short signal to avoid unnecessary congestion.

Again by implementation of a sensor the system becomes much more robust and efficient.

The RTL schematic for the system is given as below:

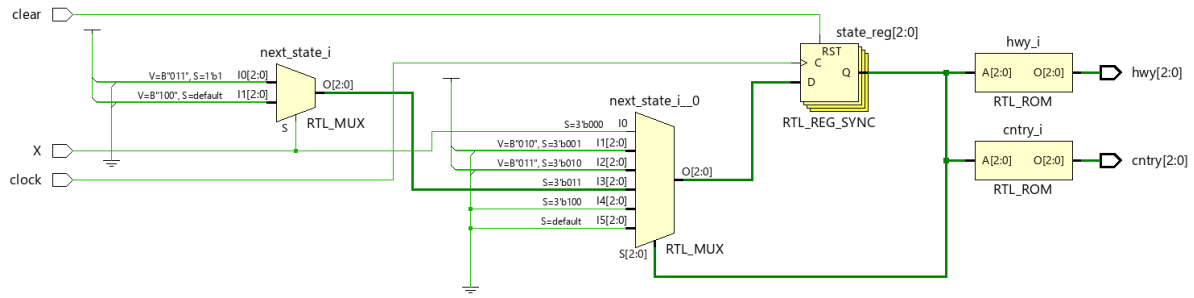



FIGURE: RTL SCHEMATIC FOR SENSOR BASED CONTROLLER

## 7. RESULTS

Traffic jamming is a serious predicament in many of the cities and towns throughout the globe. Because of this public lose time, money and mostly energy resources will be drained due to frequent use of automobiles. In this project we have designed three efficient, reliable and intelligent traffic light controllers using Verilog Hardware Description Language.

We have successfully implemented a digital traffic light controller based on finite state machines for 3 real life cases, that are:

1. 3 Way Junction
2. 3 Way Junction with Sensor
3. Highway and side road junction with Sensor

In the above cases, simulated signal output graphs and RTL schematics were obtained and studied. From the inferences drawn from these outputs of the simulation, the design of the system was made more and more efficient in several iterations till most efficient design was obtained which is ready for real life application.

We have concluded that to solve the traffic congestion in front of the Delhi Technological University, there is a need to implement the robust sensor based traffic controller.

We have also concluded from our study that there are numerous benefits of using FPGA based Traffic controllers rather than conventional ASIC based controllers. To list a few advantages

1. System has flexibility of modification in real time basis
2. Reduced number of I/O ports as compared to microcontrollers
3. Higher efficiency and Lower costs
4. More robust

We have also solved the frequent problem of important highway roadblocks due to haphazard traffic movements from side roads or state roads by using the robust system developed in the case 3, Highway and Country road junction by using sensors.

# 8. LEARNING OUTCOMES & FUTURE POSSIBILITIES

**LEARNING OUTCOMES:**

1. In depth study of Finite state machines, State diagrams and state truth tables.
2. Learnt how to make system designs more efficient and robust.
3. Gained knowledge of using PROM, MUX, etc. to implement complex system designs.
4. Learnt how to design systems from real life scenarios and translate it into an algorithm and flow of events.
5. Gained invaluable skill of coding in Verilog HDL and using Xilinx Vivado to carry out simulation and design work.
6. Learnt how to read a research paper and formulate our findings in a structured manner.
7. Realised the uses and advantages of the various cases in real life traffic congestion situations.
8. Solved real life problems using theoretical concepts taught in classroom teaching.

**FUTURE POSSIBILITIES:**

1. The future scope of this project can also be directly applied in real time by employing density based automatic traffic control.
2. In upcoming days our focus will be to make the system more optimized with regards to power consumption—making the device power efficient.
3. Future scope of this project includes the use of neuro-fuzzy or image processing techniques in order to determine the real time traffic so as to provide a greater control over congestion.
4. In particular, our focus will be to realize the given model using the FPGA board for deeper analysis to compare the vehicles entering the intersection per hour as well as comparing the length of waiting vehicles to understand which lanes should be given higher priorities.

# 9. REFERENCES


1. S. V. Kishore, V. Sreeja, V. Gupta, V. Videesha, I. B. K. Raju and K. M. Rao, "FPGA based traffic light controller," 2017 International Conference on Trends in Electronics and Informatics (ICEI), 2017, pp. 469-475, doi: 10.1109/ICOEI.2017.8300971.
2. Rodríguez-Osorio R.M., Otero M.Á.F., Ramón M.C., Navarrete L.C., Ariet L..H. (2007) The Design of a FPGA-Based Traffic Light Control System: From Theory to Implementation. In: Ince A.N., Bragg A. (eds) Recent Advances in Modeling and Simulation Tools for Communication Networks and Services. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-73908-3_8
3. Qureshi, M. & Aziz, Abdul & Raza, S.. (2012). A Verilog Model of Adaptable Traffic Control System Using Mealy State Machines. International Journal of Computer and Electrical Engineering. 400-403. 10.7763/IJCEE.2012.V4.521.
4. Rajendran, Selvakumar & Dr.S.Nirmala,. (2013). Design of Automated Day-Night Traffic Light Controller System with FPGA. 5th National Conference on Signal Processing Communications & VLSI Design (NCSCV 13). 570-575.
5. K. S. Reddy and B. B. Shabarinath, "Timing and Synchronization for Explicit FSM Based Traffic Light Controller," 2017 IEEE 7th International Advance Computing Conference (IACC), 2017, pp. 526-529, doi: 10.1109/IACC.2017.0114.
6. A. Albagul, M. Hrairi, Wahyudi and M.F. Hidayathullah, "Design and Development of Sensor Based Traffic Light System", American Journal of Applied Sciences 3 (3): 1745- 1749, 2006.
7. Dilip, B., Y. Alekhya and P. Bharathi. “FPGA Implementation of an Advanced Traffic Light Controller using Verilog HDL.” (2012).
8. NPTEL, Fundamentals of Digital Circuits and Systems, IIT Madras (2009): https://nptel.ac.in/courses/117/106/117106092/
9. NPTEL, Hardware Modelling using Verilog, IIT Kharagpur(2017): https://nptel.ac.in/courses/106/105/106105165/
10. Getting Started with Vivado, Xilinx (https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2013x/Nexys4/Verilog/docs-pdf/Vivado_tutorial.pdf)
11. Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition by Samir Palnitkar (Publishing Date: 21st February, 2003)
12. Electronics Tutorials, What are Finite State Machines? (https://www.electronics-tutorial.net/finite-state-machines/FSM-Applications/Traffic-Light-Controller/)
13. Introduction to FSM in Verilog (https://www.asic-world.com/tidbits/verilog_fsm.html#:~:text=Basically%20a%20FSM%20consists%20of,shown%20in%20the%20figure%20below.)
14. Introduction to VLSI Coding using FSM Coding and Test Bench (https://vlsicoding.blogspot.com/2013/11/verilog-code-for-traffic-light-control.html)
15. Traffic Light Controller, Stanford University (http://cva.stanford.edu/classes/cs99s/Lab6Prep.pdf)