BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY
Department of Electrical & Electronic Engineering

EEE 304
Digital Electronics Laboratory

Project Report

Traffic Light Controller

1606147-RATUL KUNDU

1606148-PRASUN DATTA

1606151-RAISA BENTAY HOSSAIN

DRIVE LINK-

https://drive.google.com/drive/folders/1U99BuEqo6o3NxUA4uBo-laksNB2ZM9ti?fbclid=IwAR1gAAOf8-m68trCsFrKK5gFkMVntHWJPywhP_vwgLv_8bWP6aq9VKrl5uM

Table of Contents

PROJECT NAME

TRAFFIC LIGHT CONTROLLER DESIGN USING VERILOG FSM With Implementation in Proteus

1.ABSTRACT

Traffic light plays an important role in controlling the traffics in urban area. It helps to reduce traffic jam and prevent accidents. Traffic light controller is designed for a four roads junction with eight lenses. It will generate a sequence of digital data to switch the color of the traffic light among red, green and yellow. We have used the idea of Finite State Machine (FSM) to define different states.

For this project we have used Quartus and Proteus in software part to implement the coding and synthesis the circuit. For the analysis part we have wave form analysis and circuit analysis using Proteus.

2.PROJECT EXPLANATION

There are 8 lanes and at most one way is opened for safely passing the vehicle. When one of the four junctions is signaled green vehicles can pass from that side to all together three sides. The illustration for the junction is drawn here.

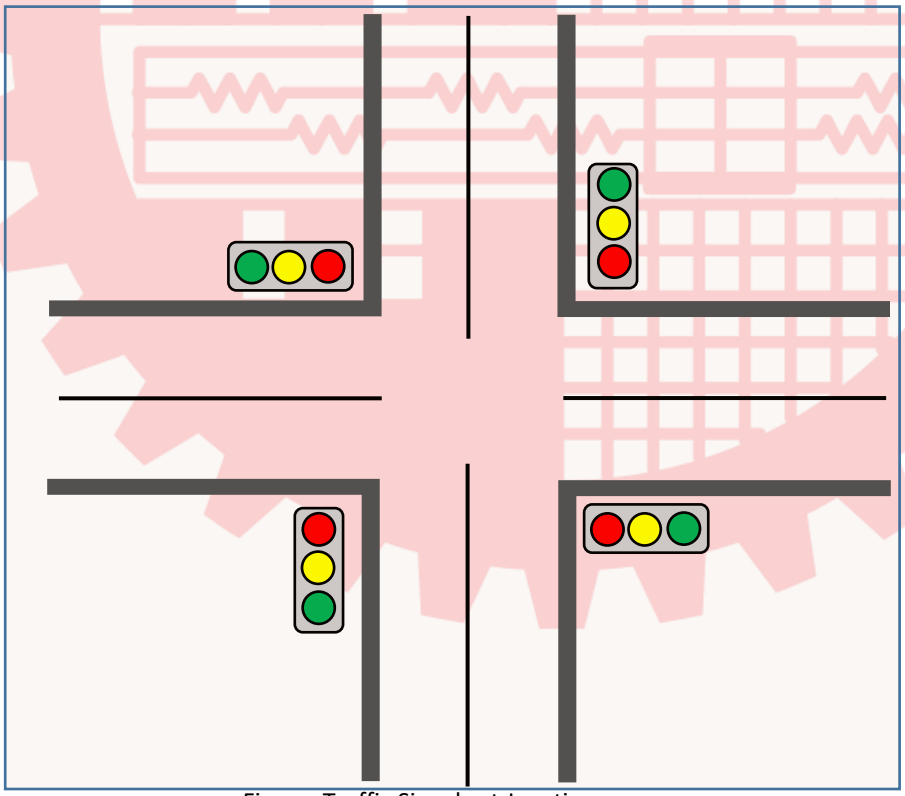


Figure: Traffic Signals at Junctions

The green light will be turned on for 8 clock cycles, the yellow light will be turned no for 4 clock cycles and the remaining 32 clock cycles will be for red. Before turning green from red the light will become yellow for 4 clock cycles again. Thus, the color switching sequence for a particular side is, green-yellow-red-yellow- green. The sequence with clock cycle is shown in the table below,

Color Switching Sequence	Green -> Yellow -> Red -> Yellow -> Green -> go on.
Clock Cycles	8 clk cyc -> 4 clk cyc -> 32 clk cyc -> 4 clk cyc -> 8 clk cyc -> go on.

The system will start when reset, and enter the north state and start giving output on the negative edge of reset signal. From the north state, it will go to the next state and keep going. We have defined 8 states in this design. The states are specified in the table below.

3.States

State No	State Name	Output
1	North	n_lights = Green s_lights = Red e_lights = Red w_lights = Red
2	North_y	n_lights = Yellow s_lights = Yellow e_lights = Red w_lights = Red
3	South	n_lights = Red s_lights = Green e_lights = Red w_lights = Red
4	South_y	n_lights = Red s_lights = Yellow e_lights = Yellow w_lights = Red
5	West	n_lights = Red s_lights = Red e_lights = Red w_lights = Green
6	West_y	n_lights = Yellow s_lights = Red

		e_lights = Red w_lights = Yellow
7	East	n_lights = Red s_lights = Red e_lights = Green w_lights = Red
8	East_y	n_lights = Red s_lights = Red e_lights = Yollow w_lights = Yollow

4.Outputs

We have four outputs n-lights, s-lights, e-lights and w-lights denoting the traffic lights on north, south, east and west. These outputs are three bits wires. Each traffic lights have three colors green, red and yellow. Binary notation for each colors are noted below.

Color	Binary notation
Green	3'b001
Red	3'b100
Yellow	3'b010

5.Inputs

Here we have two inputs, clock cycle and reset. On the reset signal, design will enter into north state and start giving output after reset will go low. Design will turn on Green light for eight clock cycles and Yellow light for four clock cycles. Design will start with north, then goes into south, then east and finally into west and by this it will keep going.

6.VERILOG CODE

```
module
traffic_control(n_lights,s_lights,e_lights,w_lights,clk,rst_a);

   output reg [2:0] n_lights,s_lights,e_lights,w_lights;
   input      clk;
   input      rst_a;

   reg [2:0] state;
```

```
parameter [2:0] north=3'b000;
parameter [2:0] north_y=3'b001;
parameter [2:0] south=3'b010;
parameter [2:0] south_y=3'b011;
parameter [2:0] east=3'b100;
parameter [2:0] east_y=3'b101;
parameter [2:0] west=3'b110;
parameter [2:0] west_y=3'b111;

reg [2:0] count;


always @(posedge clk, posedge rst_a)
  begin
     if (rst_a)
         begin
             state=north;
             count =3'b000;
         end
     else
         begin
             case (state)
             north :
                 begin
                     if (count==3'b111)
                         begin
                         count=3'b000;
                         state=north_y;
                         end
                     else
                         begin
                         count=count+3'b001;
                         state=north;
                         end
                 end

             north_y :
                 begin
                     if (count==3'b011)
                         begin
                         count=3'b000;
                         state=south;
                         end
                     else
```

```
                begin
                count=count+3'b001;
                state=north_y;
            end
        end

    south :
        begin
            if (count==3'b111)
                begin
                count=3'b0;
                state=south_y;
                end
            else
                begin
                count=count+3'b001;
                state=south;
            end
        end

south_y :
    begin
        if (count==3'b011)
            begin
            count=3'b0;
            state=east;
            end
        else
            begin
            count=count+3'b001;
            state=south_y;
            end
        end

east :
    begin
        if (count==3'b111)
            begin
            count=3'b0;
            state=east_y;
            end
        else
            begin
            count=count+3'b001;
```

```
                state=east;
                end
            end

    east_y :
        begin
            if (count==3'b011)
                begin
                count=3'b0;
                state=west;
                end
            else
                begin
                count=count+3'b001;
                state=east_y;
                end
            end

    west :
        begin
            if (count==3'b111)
                begin
                state=west_y;
                count=3'b0;
                end
            else
                begin
                count=count+3'b001;
                state=west;
                end
            end

    west_y :
        begin
            if (count==3'b011)
                begin
                state=north;
                count=3'b0;
                end
            else
                begin
                count=count+3'b001;
                state=west_y;
                end
```

```
                    end
                endcase // case (state)
            end // always @ (state)
        end


always @(state)
     begin
         case (state)
            north :
                begin
                    n_lights = 3'b001;
                    s_lights = 3'b100;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: north

            north_y :
                begin
                    n_lights = 3'b010;
                    s_lights = 3'b010;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: north_y

            south :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b001;
                    e_lights = 3'b100;
                    w_lights = 3'b100;
                end // case: south

            south_y :
                begin
                    n_lights = 3'b100;
                    s_lights = 3'b010;
                    e_lights = 3'b010;
                    w_lights = 3'b100;
                end // case: south_y

            west :
                begin
                    n_lights = 3'b100;
```

```
                s_lights = 3'b100;
                e_lights = 3'b100;
                w_lights = 3'b001;
            end // case: west

        west_y :
            begin
                n_lights = 3'b010;
                s_lights = 3'b100;
                e_lights = 3'b100;
                w_lights = 3'b010;
            end // case: west_y

        east :
            begin
                n_lights = 3'b100;
                s_lights = 3'b100;
                e_lights = 3'b001;
                w_lights = 3'b100;
            end // case: east

        east_y :
            begin
                n_lights = 3'b100;
                s_lights = 3'b100;
                e_lights = 3'b010;
                w_lights = 3'b010;
            end // case: east_y
        endcase // case (state)
    end // always @ (state)
endmodule
```

6.1.Verilog Output (Timing diagram)

Waveform generated from the Verilog code is attached here,

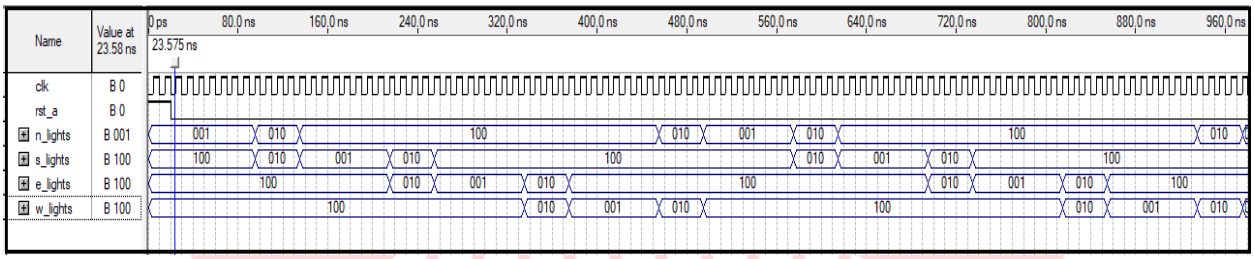


Figure: Vector Waveform

6.2.Verilog Output (MODELSIM)

Outputs observed using MODELSIM

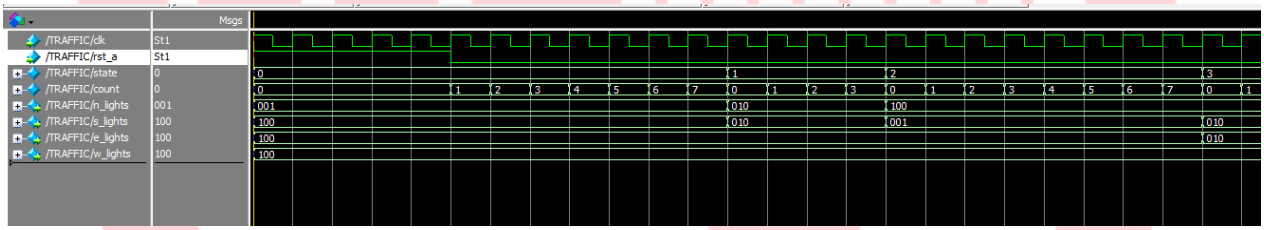


Figure: Waveform in MODELSIM

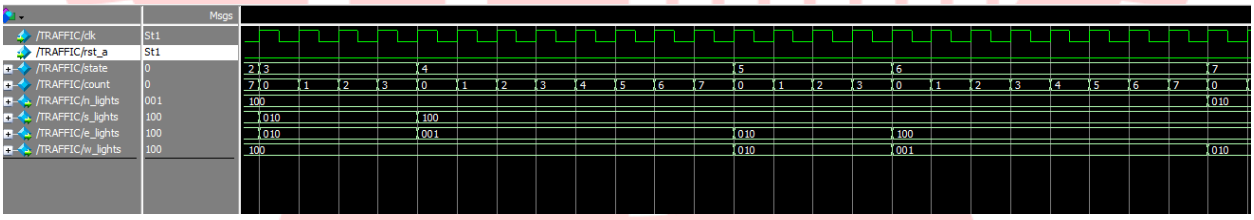


Figure: Waveform in MODELSI

6.3.CIRCUIT USING RTL VEIWER

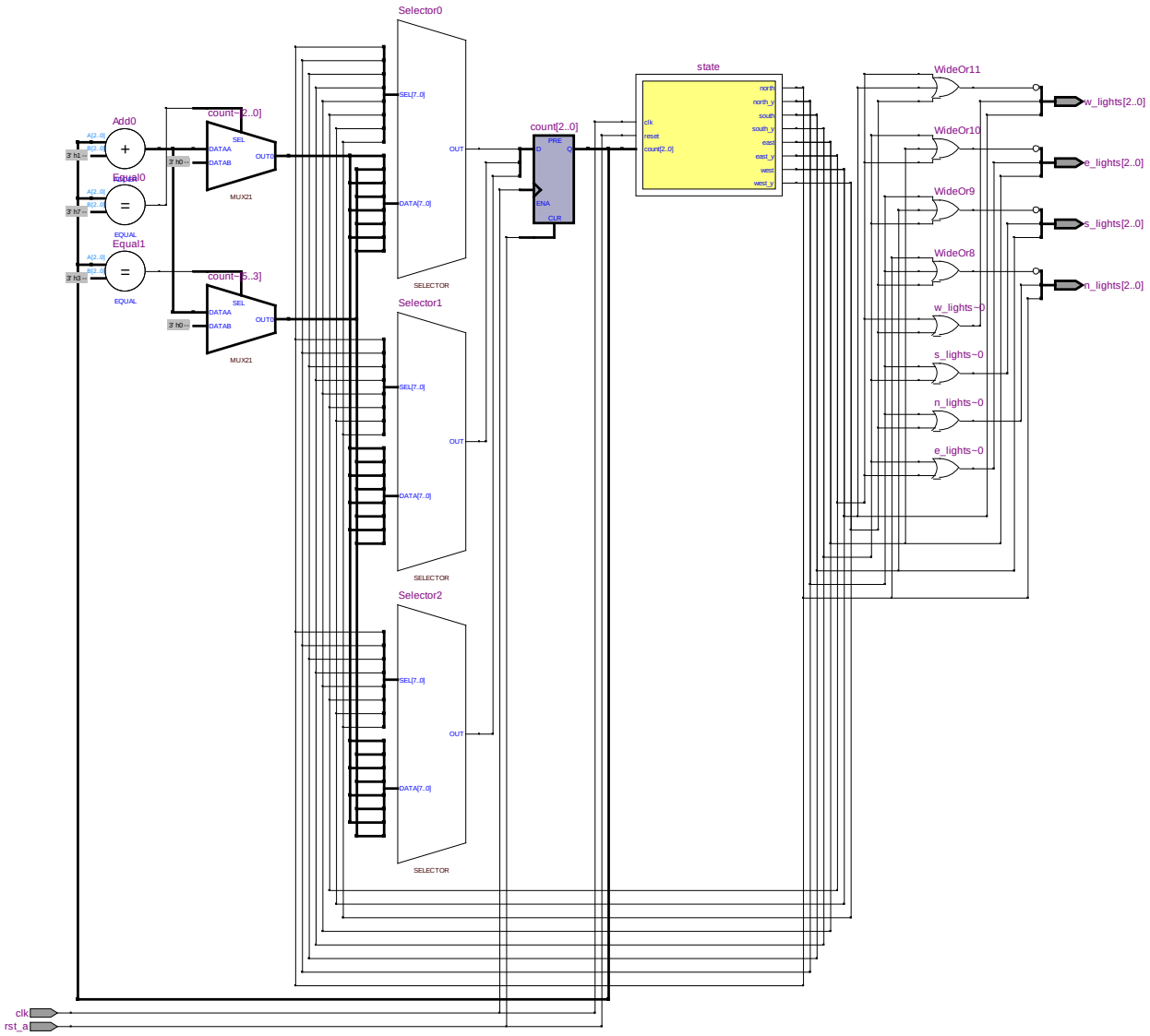
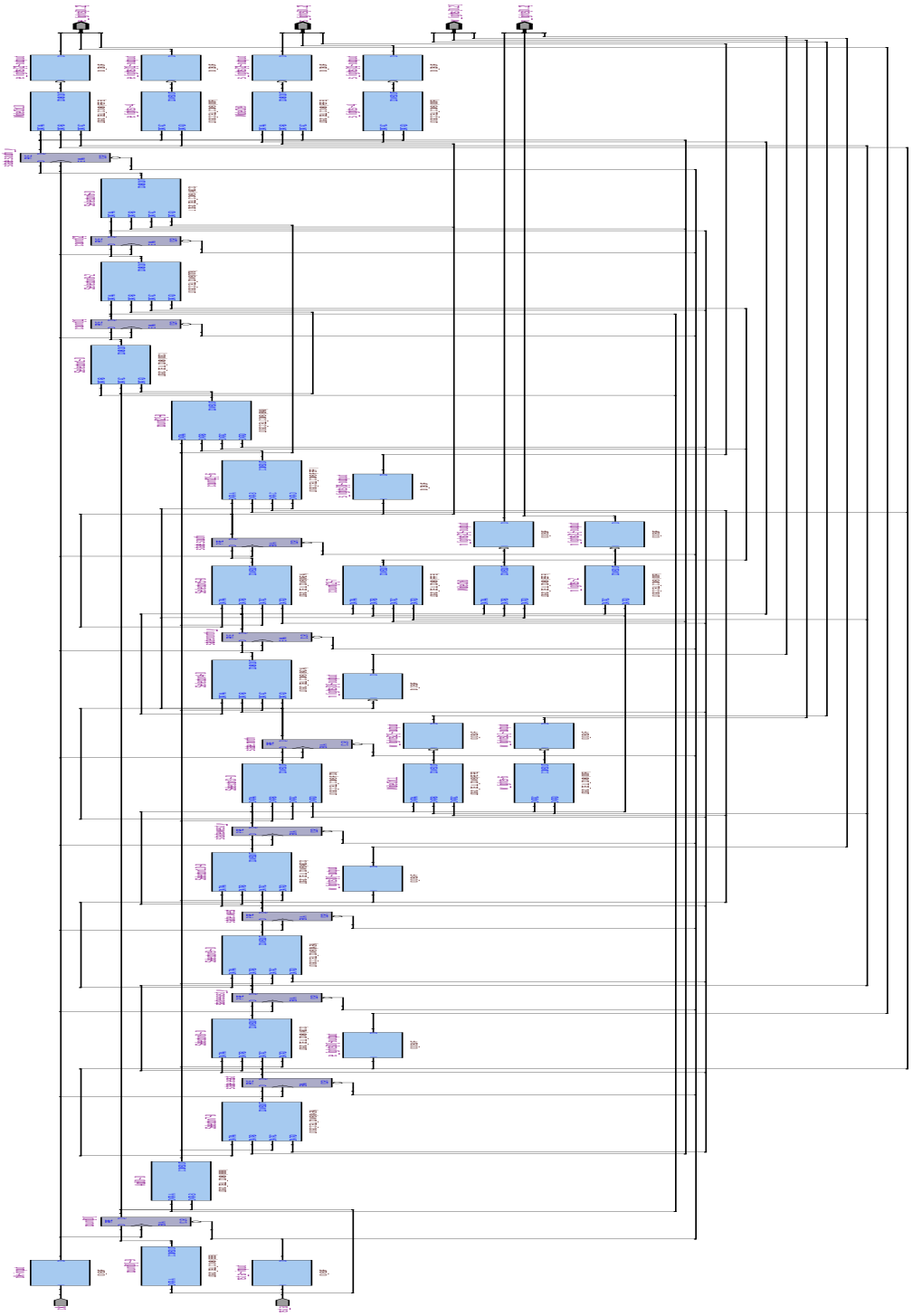


Figure: RTL Schematics

6.4.TECHNOLOGY SCHEMETIC

The figure below shows the technology schematics of traffic light controller



6.5.STATE DIAGRAM USING STATE MACHINE VEIWER

The state diagram of the design can be generated automatically using state machine generator,

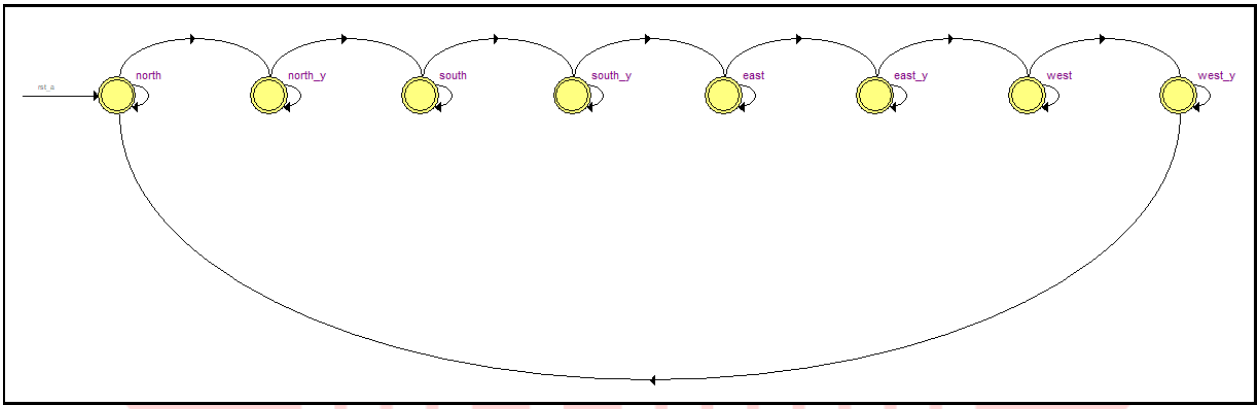
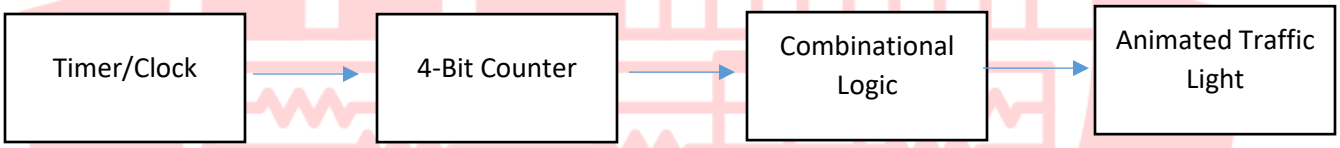


Fig: State Diagram

7.Circuit Implementation in Proteus:

7.1. General Flow of Circuit:



- For timing purposes & for operating the flip flops synchronously, a clock pulse has been used.

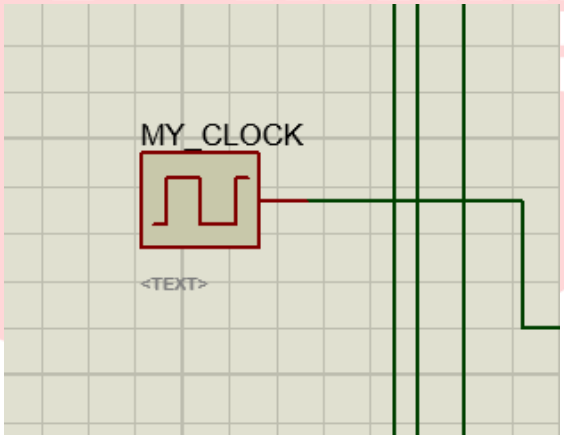


Fig: Clock Pulse

- 4-bit synchronous up counter has been designed using T flip-flop. Here the mathematical analysis has been done by the T flip flop whereas in the proteus, J-K flip flop has been used as the T flip flop. For that purpose, J & K of each of the flip flop has been shorted.

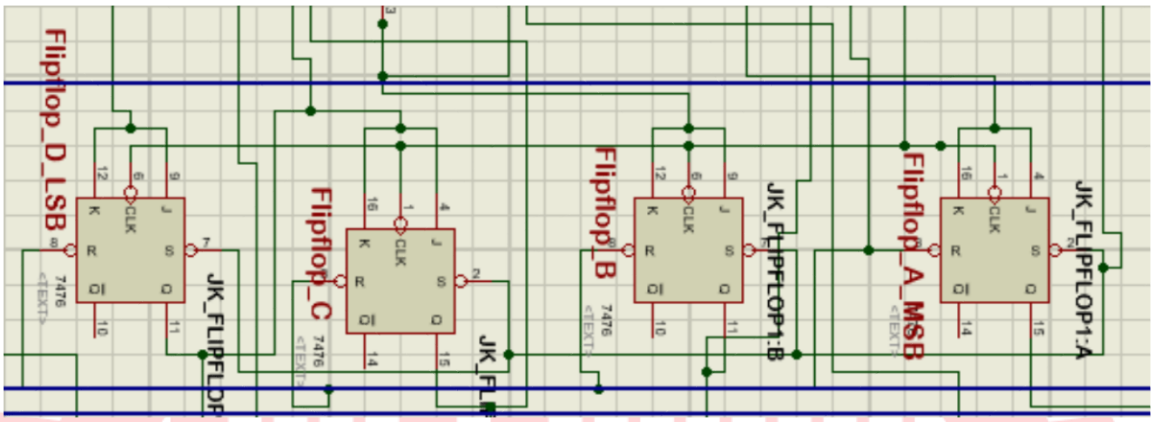


Fig: 4-bit synchronous up counter

- K-map minimization technique has been performed for each of the output of the flip flop.
- Afterwards, a combinational circuit has been designed according to the time delay requirements.
- Eventually respected equation for each light has been calculated using K-map. These combination circuit's output has been demonstrated by the four animated traffic light model.

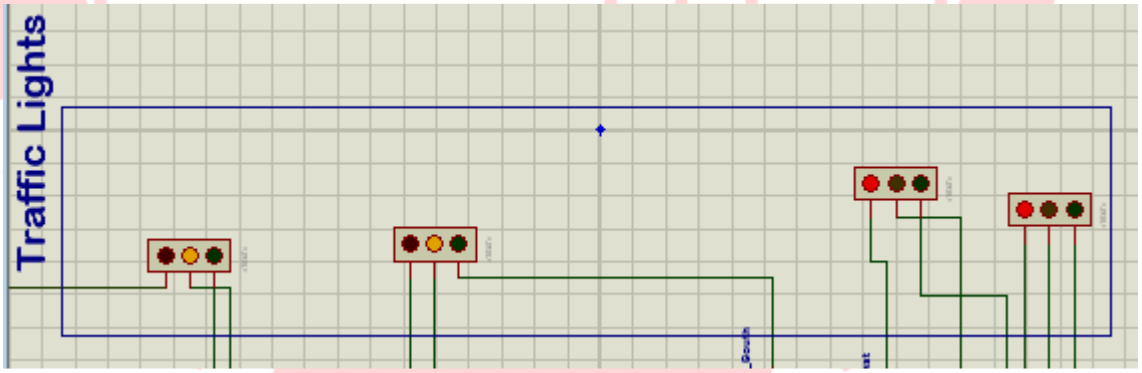


Fig: Animated Traffic Light Model

7.2. Mathematical Analysis of Proteus Circuit:

- In this design 4 T -flip flops have been used. As such circuit excitation table of T flip-flop will be needed.

Present State/Q(n)	Next State/Q(n+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

7.3. 4-Bit Counter Truth Table using T Flip-flop

Q(A)	Q(B)	Q(C)	Q(D)	Q(A+1)	Q(B+1)	Q(C+1)	Q(D+1)	T(A)	T(B)	T(C)	T(D)
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	1	0	1	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	1	1	0	1	0	0	0	1
1	1	0	1	1	1	1	0	0	0	1	1

1	1	1	0	1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0	1	1	1	1

7.4. K-Map Minimization of Equations of 4-bit flip flops

T(A) Flip-flop

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	0	0	0

T(A): Q(B)Q(C)Q(D)

T(B) Flip-flop

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

T(B): Q(C)Q(D)

T(C) Flip-flop

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

T(C): Q(D)

T(D) Flip-flop

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

T(D): 1

7.5. 4-bit Up Counter State Diagram:

Since there are 4 flip-flops, eventually there will be 16 states.

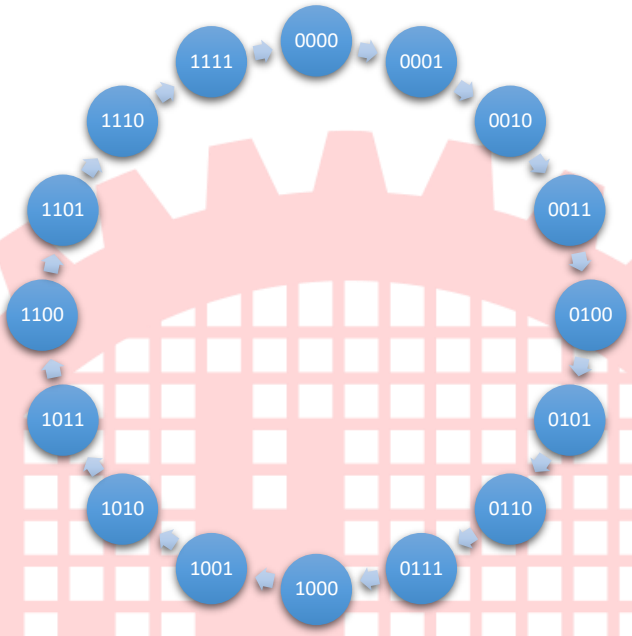


Fig: State Diagram of 4-bit up counter

7.6.Truth Table for Traffic Light Signals:

Q(A)	Q(B)	Q(C)	Q(D)	North			South			East			West		
Counter States				Red	Yellow	Green	Red	Yellow	Green	Red	Yellow	Green	Red	Yellow	Green
0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0
0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0
0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0
0	1	0	1	1	0	0	0	0	1	1	0	0	1	0	0
0	1	1	0	1	0	0	0	0	1	1	0	0	1	0	0
0	1	1	1	1	0	0	0	0	1	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	1	0	1	0	0
1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0
1	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0
1	0	1	1	1	0	0	1	0	0	0	0	1	1	0	0
1	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1
1	1	1	0	1	0	0	1	0	0	1	0	0	0	0	1
1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	1

7.7. Logical Explanation of Truth Table of Light Signals:

- As it can be seen from the above table, initially North & West light will be yellow which means that either one of them will be red and the other will be green in the next counter state.
- From the table, it's clear that North light will be green for the next three counter state. Eventually vehicles will be allowed to go from north to the other three direction for three clock cycles. In fact, this is the pattern which will be followed by the South, East & West light also. In short out of 16 counter states, each light will be green for three consecutive counter states. At a time, there will be only one green light signal & as such the other three signals will be red.
- Another important observation is that none of the green lights will be instantaneously transformed into the red light. This phenomenon imitates the real-world traffic situation. The vehicles which are on the green signal lane should be allowed a minimum time to halt. And the intermediate yellow light signal which lasts for one clock cycle highlights this phenomenon.
- Finally, the remaining states of a fixed direction lane will remain red for the rest of the counter states precisely for the remaining eleven states.

7.8. Respective Equation for Each Light Using K-Map:

K-Map Minimized Signal for North Lights
Red Light

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

Red_North :Q(A) +Q(B)Q(D)+Q(B)Q(C)

Yellow Light

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Yellow_North : (~Q(A).~Q(C).~Q(D)

Green Light

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	0	0	0
10	1	0	0	0

Green_North : (~Q(A))(~Q(B))Q(D)+ (~Q(A))(~Q(B))Q(C)

K-Map Minimized Signal for South Lights

Red Light

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	1	0	1	0
01	1	0	1	1
11	1	0	1	1
10	1	0	1	1

Red_South ; (~Q(A))(~Q(B))+ Q(A)Q(B)+(~Q(B)Q(D)+(~Q(B))Q(C)

Yellow Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	0	1	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Yellow_South : (~Q(A))Q(B)(~Q(C))(~Q(D))+ Q(A)(~Q(B))(~Q(C))(~Q(D))

Green Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

Green_South : (~Q(A))Q(B)Q(D) + (~Q(A))Q(B)Q(C)

K-Map Minimized Signal for East Lights

Red Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	1	1	0	0
01	1	1	1	0
11	1	1	1	0
10	1	1	1	0

Red_East : (~Q(A))+Q(B)Q(D)+Q(B)Q(C)

Yellow Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Yellow_East : Q(A)(~Q(C))(~Q(D))

Green Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	0	0	0	0
01	0	0	0	1
11	0	0	0	1
10	0	0	0	1

Green_East : Q(A)(~Q(B)) Q(D) + Q(A)(~Q(B))Q(C)

K-Map Minimized Signal for West Lights

Red Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	0	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	1

Red_West : (~Q(A))Q(B)+Q(A)(~Q(B))+ (~Q(A))Q(D)+ (~Q(A))Q(C)

Yellow Light

Q(A)Q(B) / Q(C)Q(D)	00	1	11	10
00	1	0	1	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

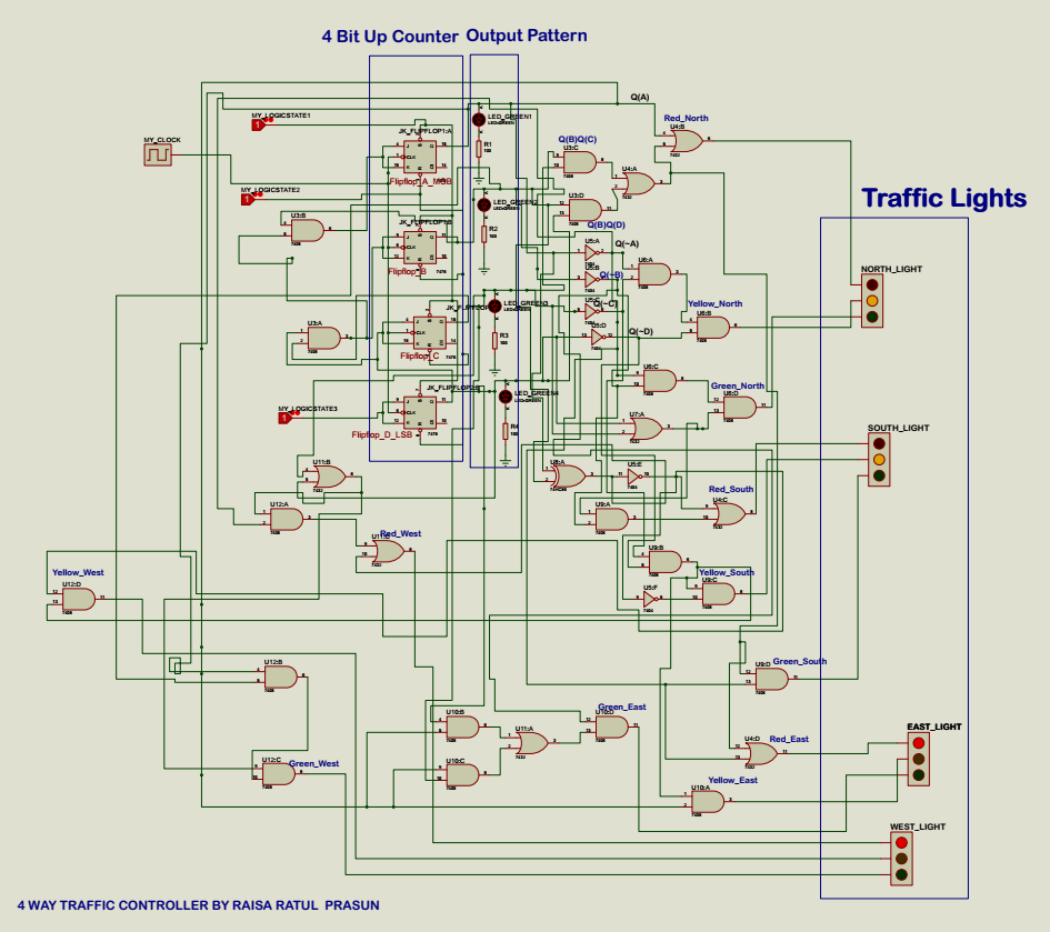
Yellow_West: (~Q(A))(~Q(B))(~Q(C))(~Q(D)) + Q(A)Q(B)(~Q(C))(~Q(D))

Green Light

Q(A)Q(B) / Q(C)Q(D)	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

Green_West : Q(A)Q(B)Q(D) + Q(A)Q(B)Q(C)

7.9. Proteus Complete Circuit Diagram:

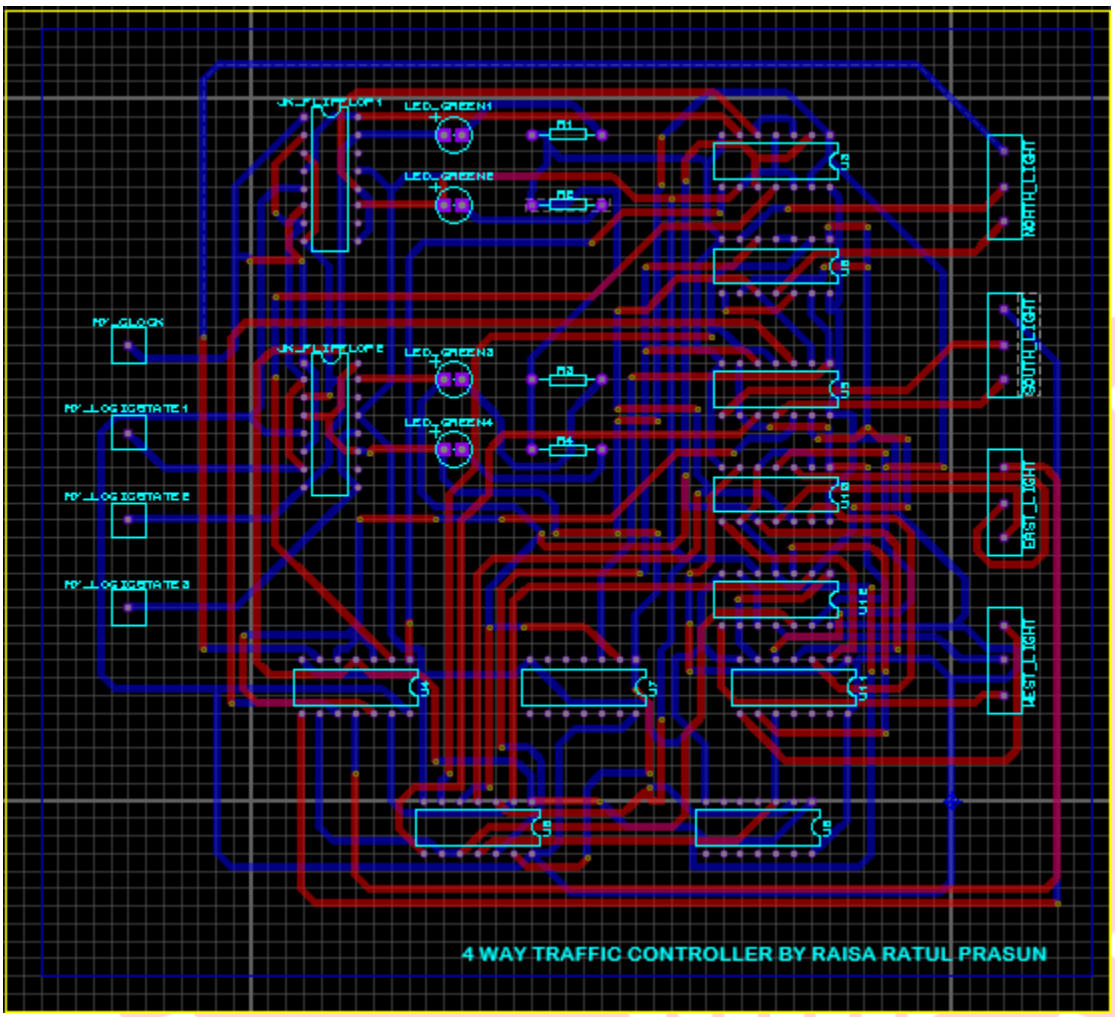


For better view of the circuit diagram click the following link

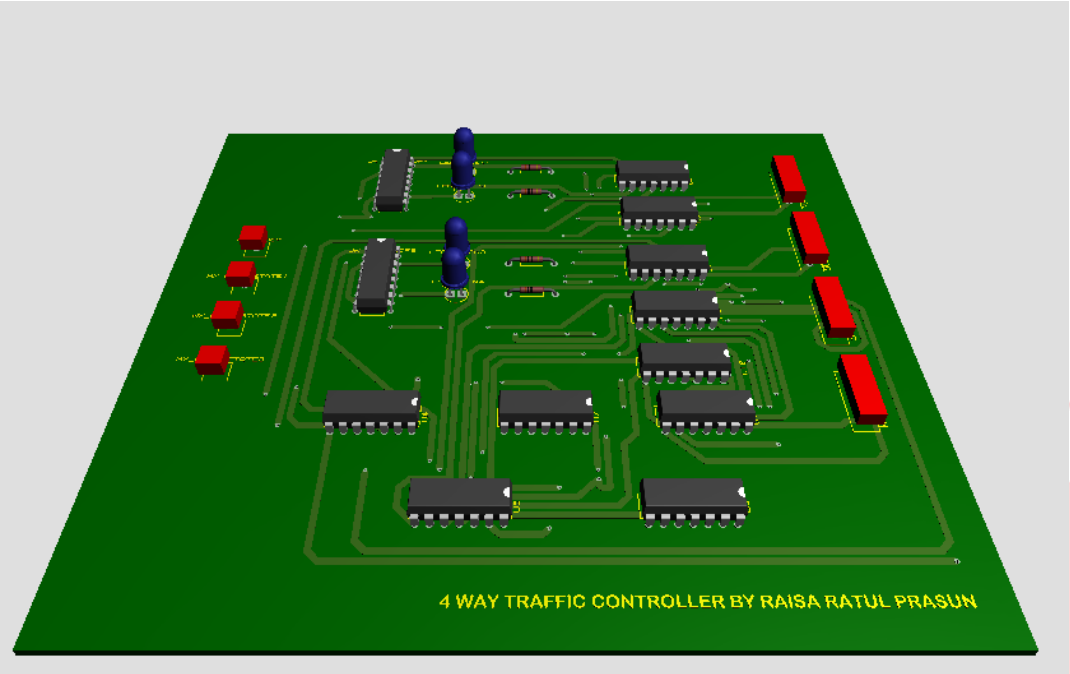
8. PCB Design:

- For the purpose of PCB design, two-layer PCB has been designed such as Top Copper & Bottom Copper layer. Hence, in the PCB circuit diagram, the blue wires indicate the bottom copper & the red wires indicate the top copper layer.
- In the proteus schematic design some of the design have been used whose PCB packages are not available such as Green led, Animated Traffic Light, Logic State etc. Consequently, the PCB packages of these parts have been designed manually using using “Square Through Hole Pad Mode”.

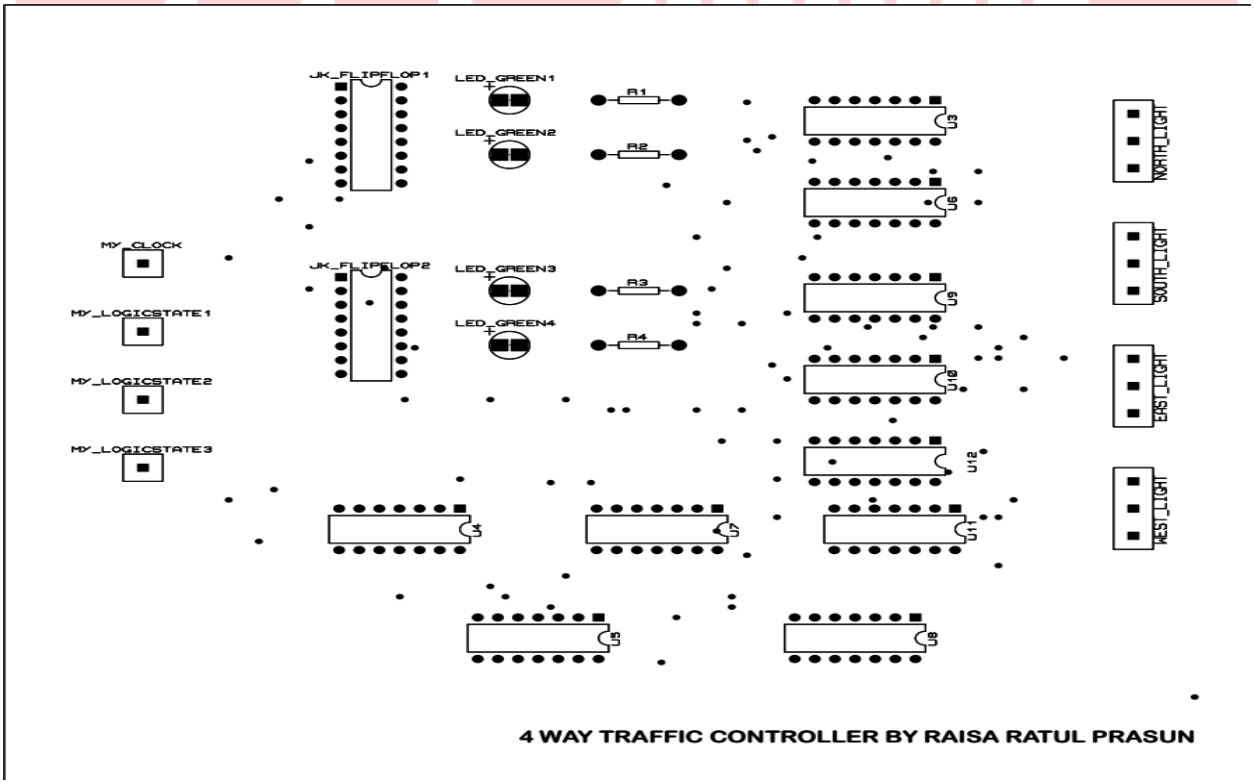
8.1.PCB Circuit Diagram:



8.2.3D View of the Designed Circuit:



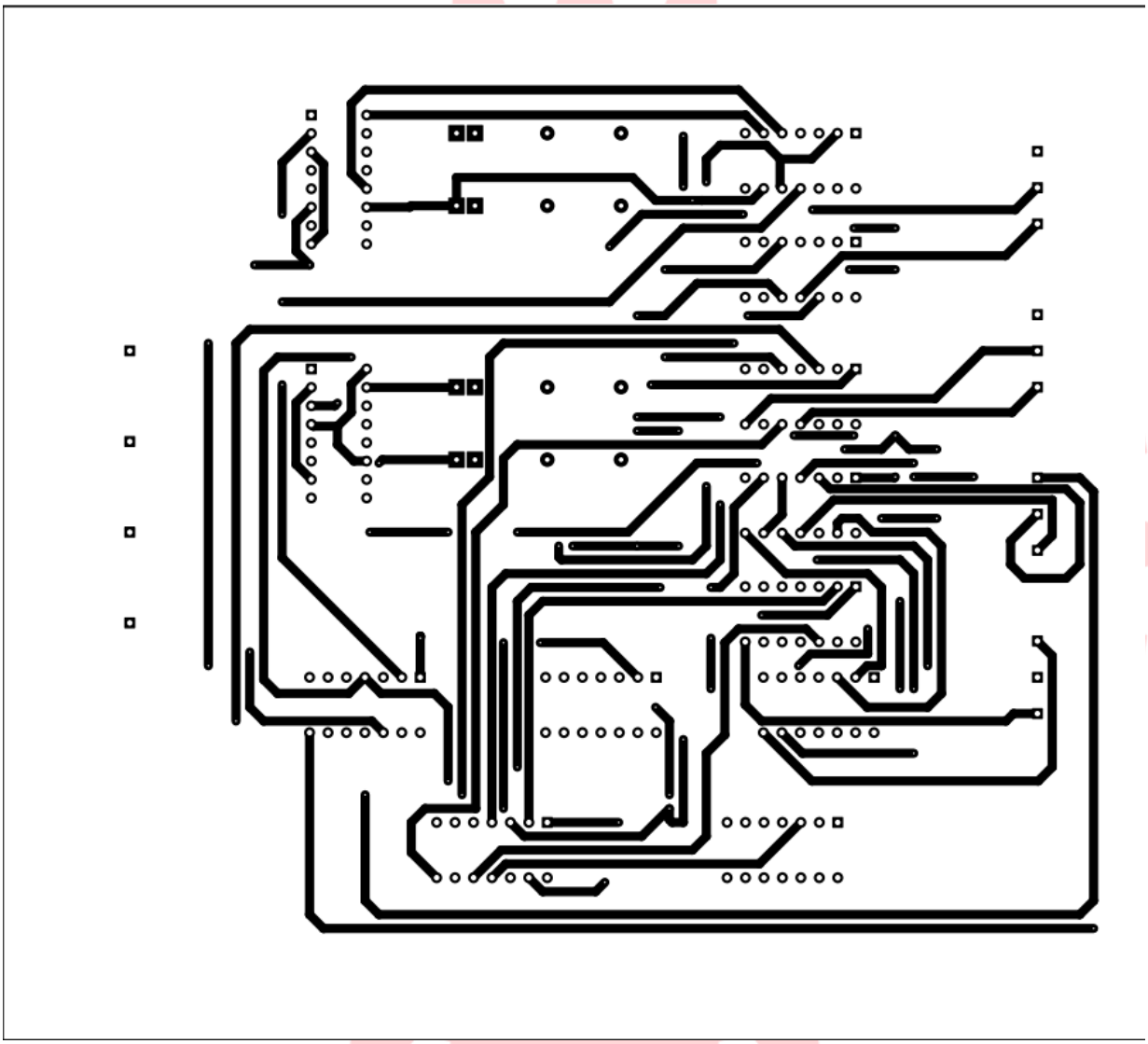
8.3.PCB Top Silk Diagram:



For a better view please click the following link

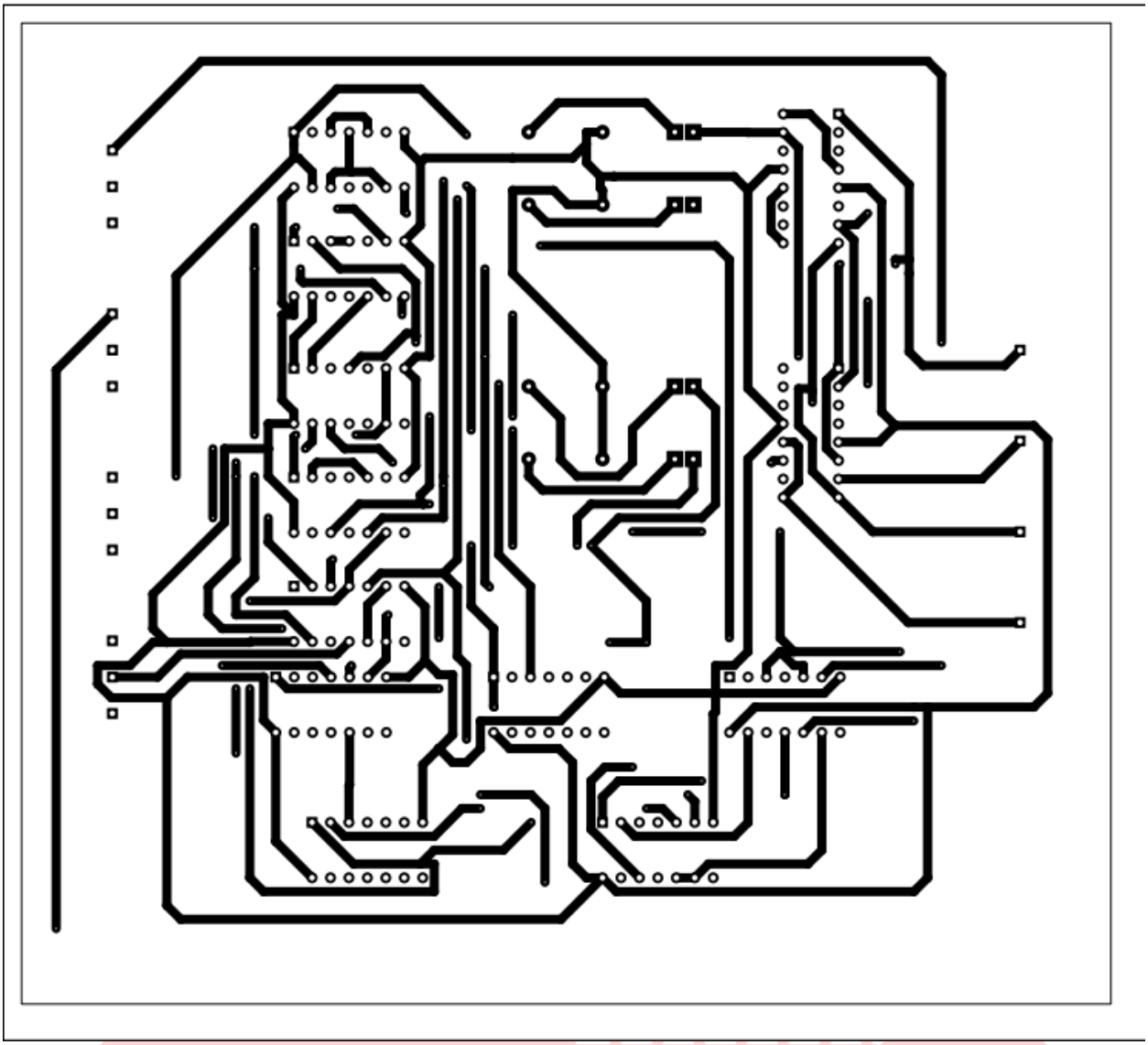
PCB_TopSilk

8.4.PCB Top Copper Diagram:



For a better view please click the following link ***PCB_TopCopper***

8.4.PCB Bottom Copper Design:



For a better view, please click the following link:

PCB_BottomCopper

9.SCOPE OF IMPROVEMENT

The model we have designed here is a very simple form of traffic light controller. There remains a lot of scopes for improvement. Different kind of combination can be achieved. Here are some ideas that can be induced to improve the management of traffic in an efficient way.

1. DAY NIGHT MODE: We can take into consideration that number of cars at night is fewer than that of at daytime. Thus, this gives us an opportunity to create two different modes of the design, where the traffic light controller will work differently from day at night.

2. PRESENCE OF SENSOR : We can use sensors to determine the pressure of traffic in a particular side of the junction and change the duration of switching sequence according to that for the better movement of traffic.

3. WEEKDAY AND WEEKEND SPECIFICATIONS: Just as the case of day/night, at week days there are a lot more cars than at weekends. We can have separate design entities for different days.

4. A SEPARATE SIGNAL FOR PEDASTRIAN PASS: In addition, with the vehicles we can also consider another signal for the passersby to cross the road safely.

10.CONCLUSION

Traffic light controller can play a very important role in managing the ever-growing traffic of urban areas. An efficient digital traffic controller will reduce the use of manpower and increase the speed of operation. It can be a great help in reducing traffic jam and number of accidents.

The digital traffic controller designed here is a primary level system which will work in only one way with constant time difference. With the help of advanced logic, we can create a system with variable time difference, it will further increase the efficiency.

This project helps us to reflect the theoretical knowledge we obtained about Finite State Machine and sequential circuits. Also concepts of up counter, flip-flops, combinational logic circuit, logic circuit minimization technique have been extensively applied in this project. We hope to work further to improve this model more.

11.Contribution:

- Modeling the controller. Designing how the controller will work.
- Assigning appropriate states to implement the model.
- Finding outputs against each state. Duration of each output, color changing sequence of the outputs.
- Verilog coding using QUARTUS.