Department of E& TC Engineering
Vishwakarma Institute of Technology, Pune

COURSE PROJECT REPORT
Digital Design (ET 3206)

TITLE

Traffic Signal Controller

BATCH

A3

GROUP MEMBERS

Branch	Div	Roll No	GR No	Name
E&Tc	A	63	11811112	Aditya Kulkarni
E&Tc	A	64	11810384	Atharva Kulkarni
E&Tc	A	69	11810076	Ankit Lad
E&Tc	A	72	11810950	Laksh Maheshwari

Batch guide

Prof. Dr. Abhay Chopde

INDEX

S.NO	CONTENT	Page No
1	Title of project	3
2	Objective	3
3	Introduction	3
4	Methodology	4
5	Code	7
6	Simulation results	11
7	Conclusion	13
8	References	13

Title of the Project:

TRAFFIC SIGNAL CONTROLLER

Objective:

Design a digital controller to control traffic at an intersection of a busy main street (North-South) and an occasionally used side street (East-West).

Introduction:

Since its inception early in the 20th century, traffic signals have witnessed a meteoric rise in traffic congestion densities and as a result have had increased importance in maintaining order amongst daily road commuters. While the current traffic controller system involves receiving a manually defined instruction set from a control room based on the density of traffic at a particular junction, the ever rising vehicular traffic coupled with an increase in the number of traffic signals calls for a more automated and intelligent process to deal with the problem of intersection traffic signal control.

ITSCP or Intersection Traffic Signal Control Problem has become even more important as traffic congestion has been more intractable. The ITSCP seeks an efficient schedule for traffic signal settings at intersections with the goal of maximizing traffic flow while considering various factors such as real-time strategies, signal timing constraints, rapid developments in traffic systems, and practical implementation. Given the time and signal specifications along with information about current state of traffic lights, it is possible to comment on the next state or future state of traffic for a particular intersection.

The said project is designed in verilog using the Xilinx Vivado tool to simulate this very situation for a particular predefined scenario having the following conditions:

- North South must be Green for a minimum of 32 seconds and will remain Green until traffic is present on East-West.

- East West will remain Green for a maximum of 16 seconds
- Yellow lights on both streets must be for 4 seconds

Methodology:

Designing any controller system involves having a solid architectural flow of the steps or protocols to be followed throughout the interval of the task. To begin with, we define the time and signal specifications for the said controller.

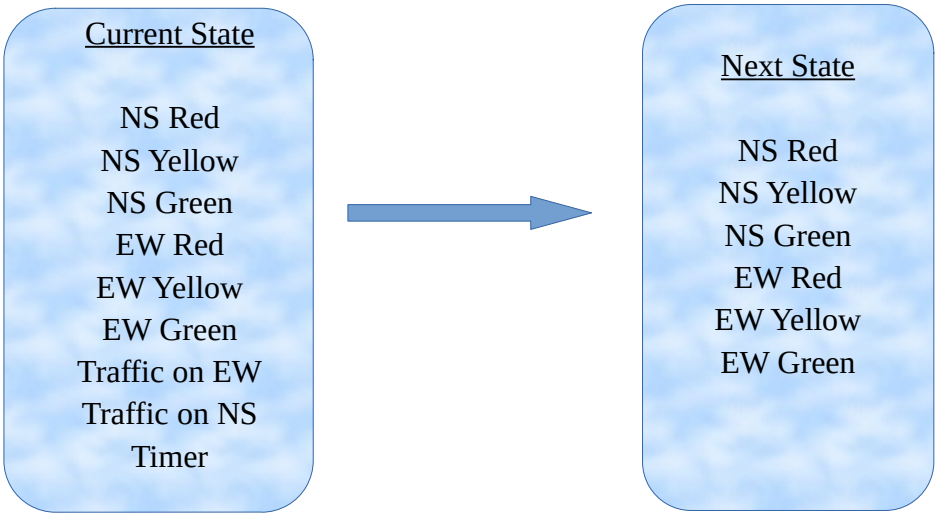
Time Specifications :

	North-South	East-West	Both Streets
Max ‘ON’ time (in seconds)	32	16	4
Signal Color	Green	Green	Yellow

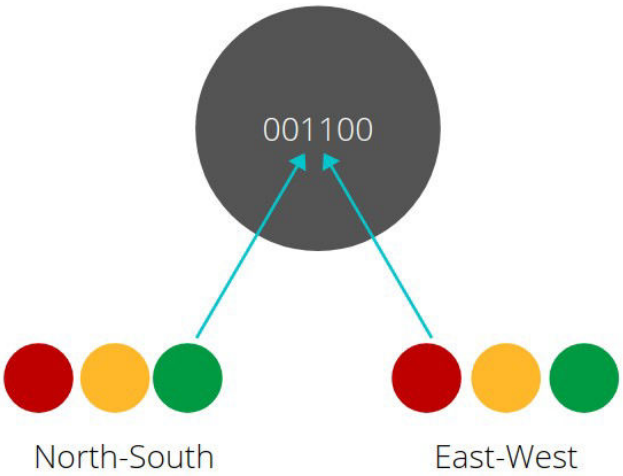
Signal Constraints :

	East-West	Both Streets
Possible signal outcomes	No green until vehicle is present	Signals don’t have to be both Red
		Signals can’t be both Green

Block Schematic :



The current states of signal values across both the North-South and East-West signals are represented as categorical encodings of active highs (‘1’) and active lows (‘0’). For example, suppose the NS and EW signals display ‘green’ and ‘red’ respectively, then the encoded vector can be represented as ‘001100’



Upon obtaining information about current active states of the signals, a characteristic table can then be formulated for that particular encoding so as to get the next state encoding for the signals.

Characteristic Table :

Inputs													
Present State								Future State					
NS			EW					NS			EW		
R	Y	G	R	Y	G	EW	Clk	R	Y	G	R	Y	G
0	0	1	1	0	0	0	X	0	0	1	1	0	0
0	0	1	1	0	0	1	0	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	1	0	1	0	0
0	1	0	1	0	0	X	0	0	1	0	1	0	0
0	1	0	1	0	0	X	1	1	0	0	0	0	1
1	0	0	0	0	1	X	0	1	0	0	0	0	1

1	0	0	0	0	1	X	1	1	0	0	0	1	0
1	0	0	0	1	0	X	0	1	0	0	0	1	0
1	0	0	0	1	0	X	1	0	0	1	1	0	0

Evaluating such a characteristic table, helps us with the following:
a) Solving karnaugh maps for next state elements
b) Formulating the state diagram

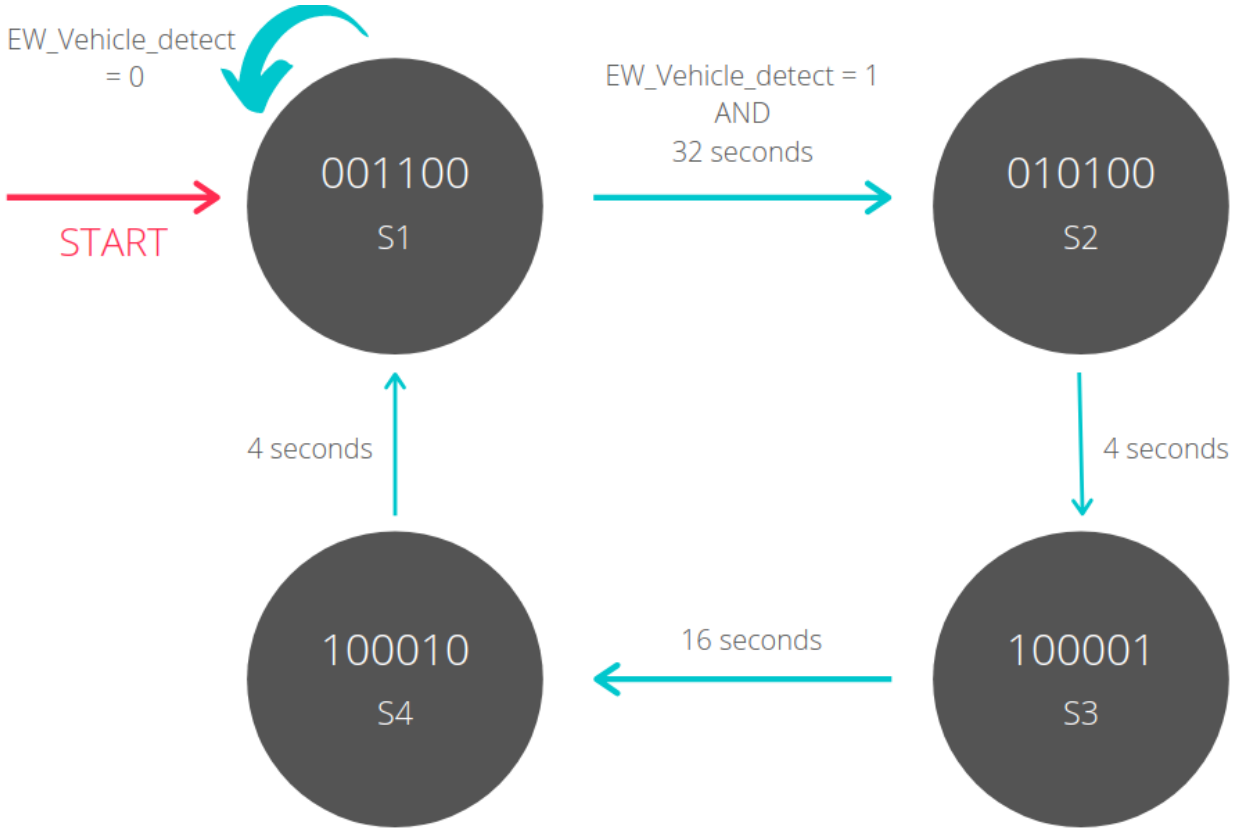
For example: Solving for next-state NS Red using 16x16 k-map looks like

	$\bar{E}\bar{F}\bar{G}\bar{H}$	$\bar{E}\bar{F}\bar{G}H$	$\bar{E}\bar{F}G\bar{H}$	$\bar{E}\bar{F}GH$	$\bar{E}F\bar{G}\bar{H}$	$\bar{E}F\bar{G}H$	$\bar{E}FG\bar{H}$	$\bar{E}FGH$	$E\bar{F}\bar{G}\bar{H}$	$E\bar{F}\bar{G}H$	$E\bar{F}G\bar{H}$	$E\bar{F}GH$	$EF\bar{G}\bar{H}$	$EF\bar{G}H$	$EFG\bar{H}$	$EFGH$
$\bar{A}\bar{B}\bar{C}\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}\bar{B}\bar{C}D$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}\bar{B}C\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}\bar{B}CD$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}B\bar{C}\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}B\bar{C}D$	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}BC\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{A}BCD$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$A\bar{B}\bar{C}\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$A\bar{B}\bar{C}D$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$A\bar{B}C\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$A\bar{B}CD$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$AB\bar{C}\bar{D}$	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	1
$AB\bar{C}D$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$AB\bar{C}D$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$AB\bar{C}\bar{D}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

where,
A = NS Red B = NS Yellow C = NS Green
D = EW Red E = EW Yellow F = EW Green
G = Traffic detected on EW
H = Timer

Upon solving the k-map, NS Red given by:
A'BC'DE'F'H + AB'C'DE'F + AB'C'D'EF'H'
= 0101001 + 100101 + 100010

State Diagram :



Code :

```
`timescale 1ns / 1ps

module Traffic
(
  input [4:0] nsCounter,
  input [3:0] ewCounter,
  input [1:0] yellowCounter,
  input NS_VEHICLE_DETECT,
  input EW_VEHICLE_DETECT,
  output reg NS_RED,
  output reg NS_YELLOW,
  output reg NS_GREEN,
  output reg EW_RED,
  output reg EW_YELLOW,
  output reg EW_GREEN
);

// State S1: 001100

  initial begin
    NS_RED <= 0;
    NS_YELLOW <= 0;
    NS_GREEN <= 1;
```

```
    EW_RED <= 1;
    EW_YELLOW <= 0;
    EW_GREEN <= 0;
  end

// State S2: 010100

  always @ (nsCounter) begin
    if (nsCounter == 31 & EW_VEHICLE_DETECT & NS_GREEN) begin
       NS_RED <= 0;
       NS_YELLOW <= 1;
       NS_GREEN <= 0;
       EW_RED <= 1;
       EW_YELLOW <= 0;
       EW_GREEN <= 0;
    end
  end

// State S4: 100010

  always @ (ewCounter) begin
    if (ewCounter == 15 & EW_GREEN) begin
       NS_RED <= 1;
       NS_YELLOW <= 0;
       NS_GREEN <= 0;
       EW_RED <= 0;
       EW_YELLOW <= 1;
       EW_GREEN <= 0;
    end
  end

// State S1 and S3: 100001

  always @ (yellowCounter) begin
    if (yellowCounter == 3 & NS_YELLOW) begin
       NS_RED <= 1;
       NS_YELLOW <= 0;
       NS_GREEN <= 0;
       EW_RED <= 0;
       EW_YELLOW <= 0;
       EW_GREEN <= 1;
    end

    if (yellowCounter == 3 & EW_YELLOW) begin
       NS_RED <= 0;
       NS_YELLOW <= 0;
       NS_GREEN <= 1;
       EW_RED <= 1;
       EW_YELLOW <= 0;
       EW_GREEN <= 0;
    end
  end
endmodule


// Defining NS counter: counts from 31 to 0

module nsCounter
(
    input clk,
    output [4:0] count
);

  wire clk;
```

```
  reg[4:0] count;

  initial
      count = 0;

  always@(negedge clk)
      count[0] <= ~count[0];
  for(i=0; i<4; i=i+1)
      begin
        always@(negedge count[i])
            begin
              count[i+1] <= ~count[i+1];
            end
      end
endmodule


// Defining EW counter: counts from 15 to 0

module ewCounter
(
    input clk,
    output [3:0] count
);

  wire clk;
  reg[3:0] count;

  initial
      count = 0;
  always@(negedge clk)
      count[0] <= ~count[0];
  for(i=0; i<3; i=i+1)
      begin
        always@(negedge count[i])
            begin
              count[i+1] <= ~count[i+1];
            end
      end
endmodule


// Defining common street counter: counts from 3 to 0

module yellowCounter
(
  input clk,
      output [1:0] count
);

  wire clk;
  reg[1:0] count;

  initial
      count = 0;
  always@(negedge clk)
      count[0] <= ~count[0];
  for(i=0; i<2; i=i+1)
      begin
        always@(negedge count[i])
            begin
              count[i+1] <= ~count[i+1];
            end
      end
```

```
endmodule
```

Testcases :

Depending upon the situation of traffic at the NS and EW signals, 8 testcases can be defined, 5 of which have been considered as follows:

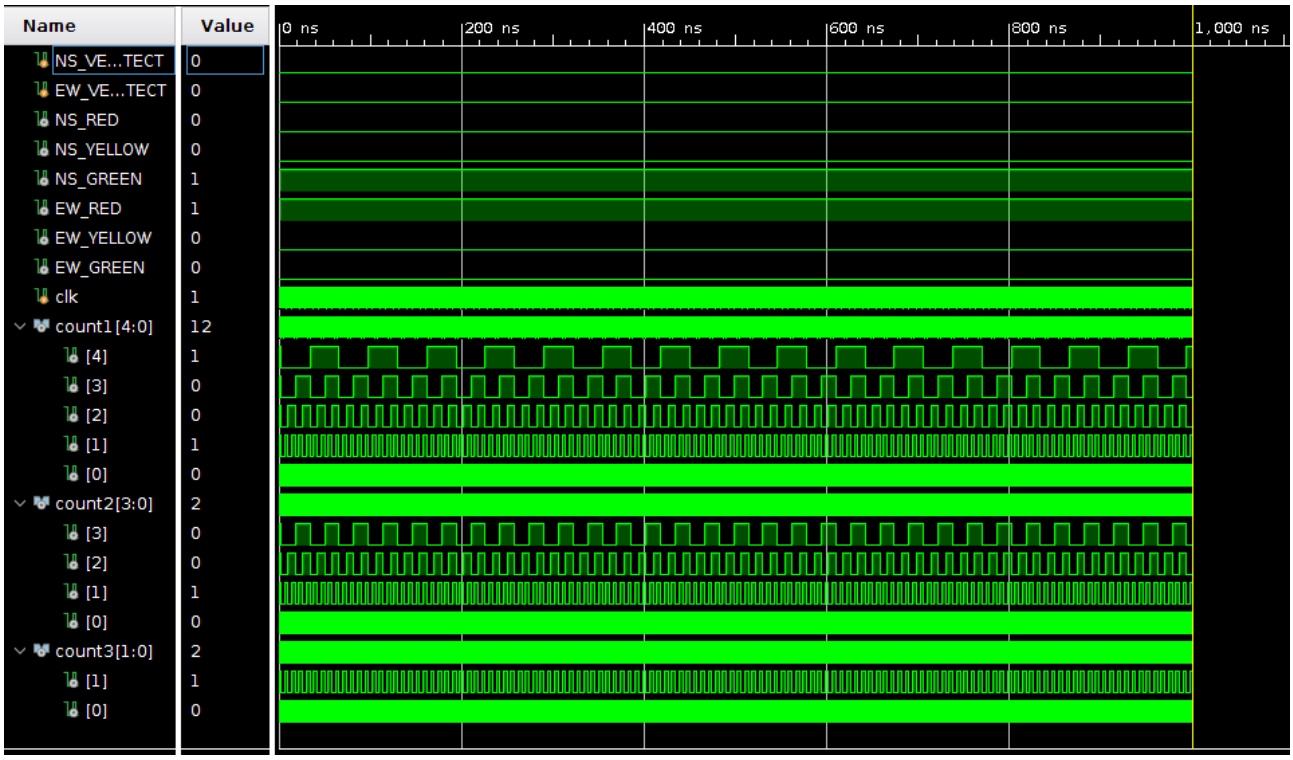
a) No traffic on either NS or EW
b) Steady traffic on NS, none on EW
c) Steady traffic on EW, none on NS
d) Intermittent traffic on NS, none on EW
e) Constant Ongoing traffic on both NS and EW

```
// Testbench for case a: no traffic on either NS or EW

`timescale 1ns / 1ps

module Traffic_Test;

  reg NS_VEHICLE_DETECT;
  reg EW_VEHICLE_DETECT;
  wire NS_RED;
  wire NS_YELLOW;
  wire NS_GREEN;
  wire EW_RED;
  wire EW_YELLOW;
  wire EW_GREEN;
  reg clk;
  wire[4:0] count1;
  wire[3:0] count2;
  wire[1:0] count3;
  nsCounter clock1(clk, count1);
  ewCounter clock2(clk, count2);
  yellowCounter clock3(clk, count3);

   Traffic CORE (count1, count2, count3, NS_VEHICLE_DETECT, EW_VEHICLE_DETECT,
NS_RED, NS_YELLOW, NS_GREEN, EW_RED, EW_YELLOW, EW_GREEN);

  initial begin

    clk = 0;
    NS_VEHICLE_DETECT = 0;
    EW_VEHICLE_DETECT = 0;

    $display("  NS | EW ");
    $display("R Y G R Y G ");
       $monitor("%h  %h  %h  %h  %h  %h", NS_RED, NS_YELLOW, NS_GREEN, EW_RED,
EW_YELLOW, EW_GREEN);

    #1000 $finish;
  end

  always begin
    #1 clk = ~clk;
  end
endmodule
```

Similarly, testbenches can be defined for the other 4 cases.

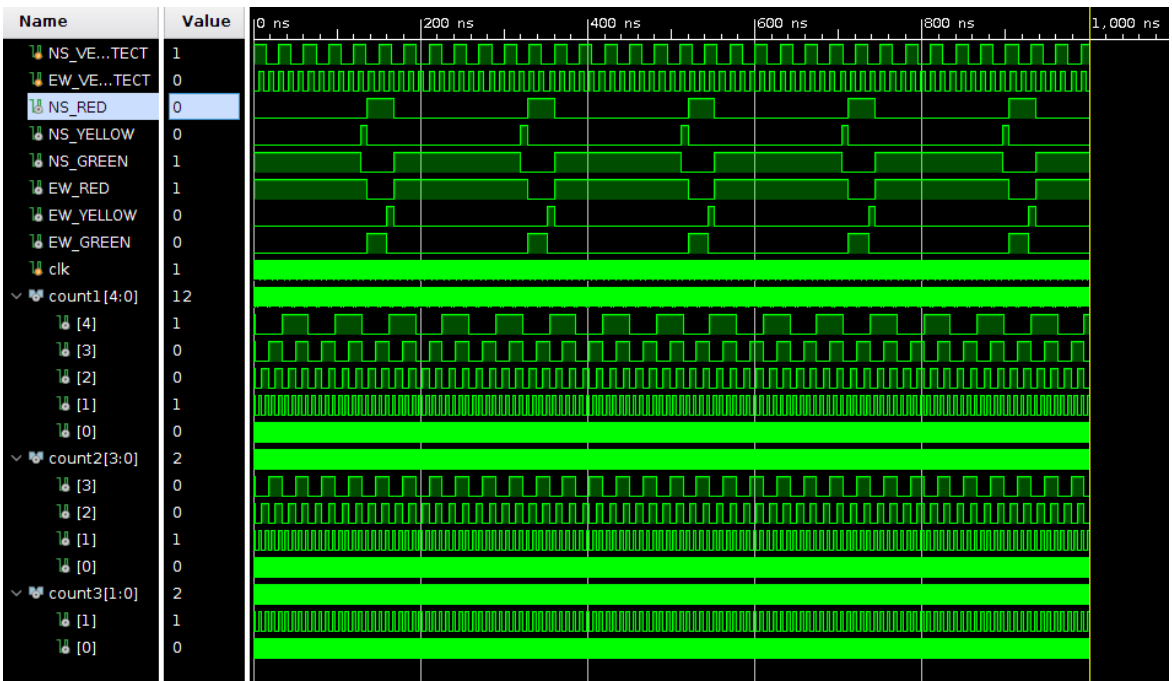
Simulation Results :

Case A: No traffic on either NS or EW



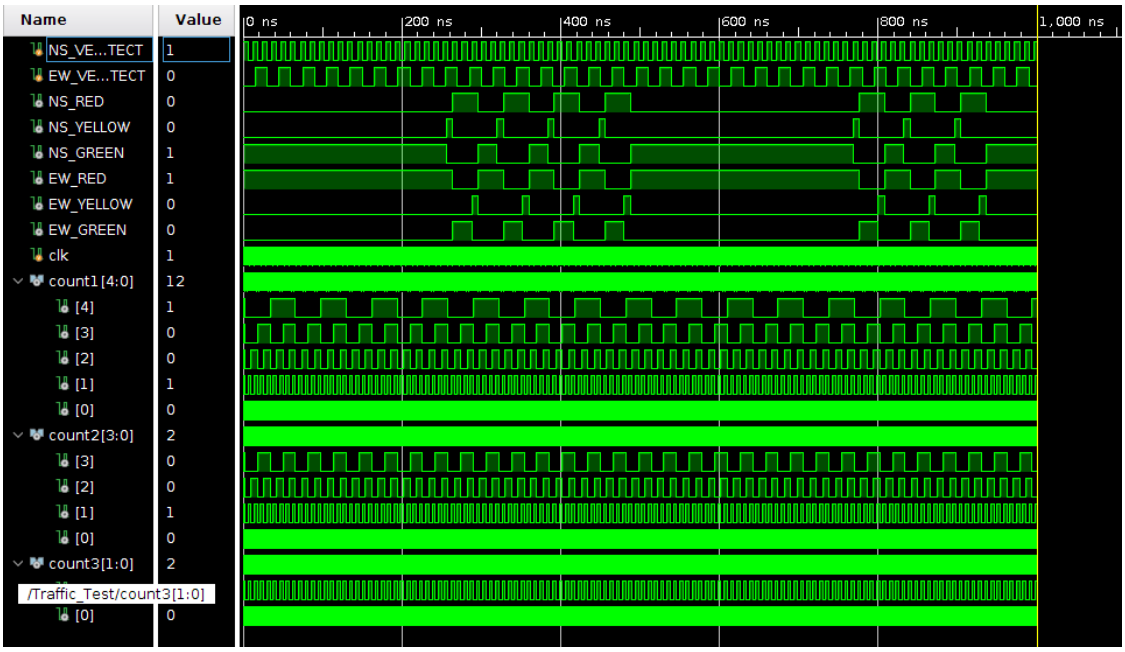
Case B: Steady traffic on NS, none on EW

(Time)	NS R	NS Y	NS G	EW R	EW Y	EW G
0	0	0	1	1	0	0
128	0	1	0	1	0	0
136	1	0	0	0	0	1
160	1	0	0	0	1	0
168	0	0	1	1	0	0
320	0	1	0	1	0	0
328	1	0	0	0	0	1
352	1	0	0	0	1	0
360	0	0	1	1	0	0
512	0	1	0	1	0	0
520	1	0	0	0	0	1
544	1	0	0	0	1	0
552	0	0	1	1	0	0
704	0	1	0	1	0	0
712	1	0	0	0	0	1
736	1	0	0	0	1	0
744	0	0	1	1	0	0
896	0	1	0	1	0	0
904	1	0	0	0	0	1
928	1	0	0	0	1	0
936	0	0	1	1	0	0



Case C: Steady traffic on EW, none on NS

(Time)	NS R	NS Y	NS G	EW R	EW Y	EW G
0	0	0	1	1	0	0
256	0	1	0	1	0	0
264	1	0	0	0	0	1
288	1	0	0	0	1	0
296	0	0	1	1	0	0
320	0	1	0	1	0	0
328	1	0	0	0	0	1
352	1	0	0	0	1	0
360	0	0	1	1	0	0
384	0	1	0	1	0	0
392	1	0	0	0	0	1
416	1	0	0	0	1	0
424	0	0	1	1	0	0
448	0	1	0	1	0	0
456	1	0	0	0	0	1
480	1	0	0	0	1	0
488	0	0	1	1	0	0
768	0	1	0	1	0	0
776	1	0	0	0	0	1
800	1	0	0	0	1	0
808	0	0	1	1	0	0
832	0	1	0	1	0	0
840	1	0	0	0	0	1
864	1	0	0	0	1	0
872	0	0	1	1	0	0
896	0	1	0	1	0	0
904	1	0	0	0	0	1
928	1	0	0	0	1	0
936	0	0	1	1	0	0

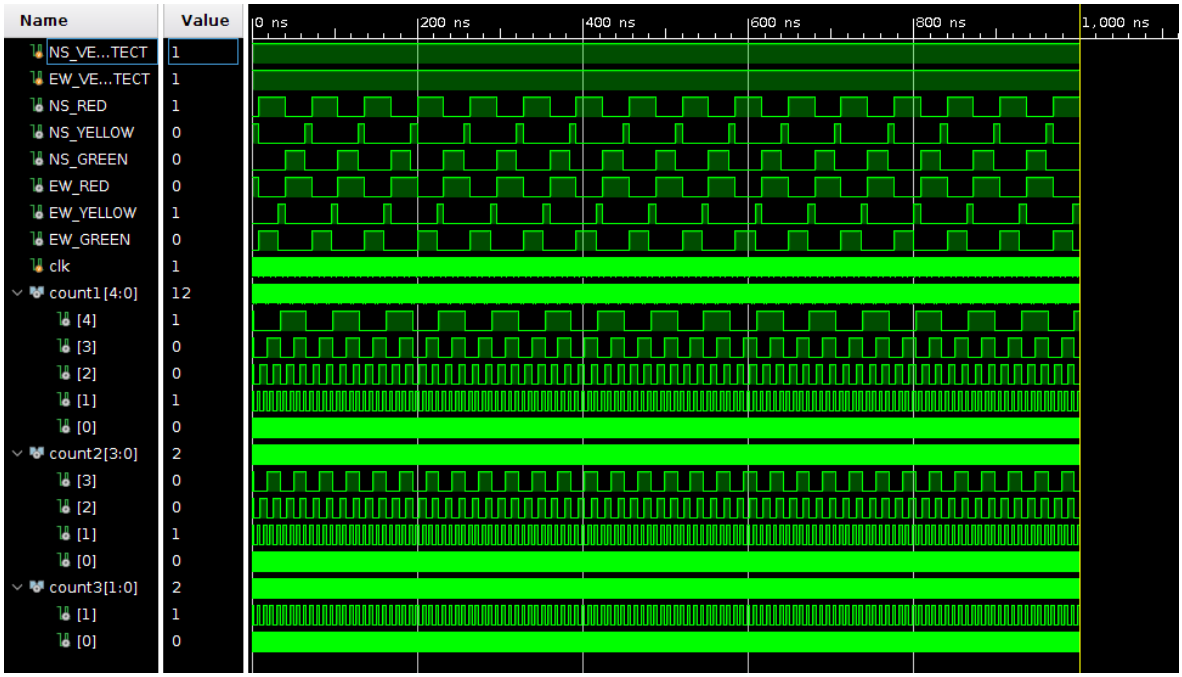


Case D: Intermittent traffic on NS, none on EW

Case E: Constant Ongoing traffic on both NS and EW

	NS			EW		
(Time)	R	Y	G	R	Y	G
0	0	1	0	1	0	0
8	1	0	0	0	0	1
32	1	0	0	0	1	0
40	0	0	1	1	0	0
64	0	1	0	1	0	0
72	1	0	0	0	0	1
96	1	0	0	0	1	0
104	0	0	1	1	0	0
128	0	1	0	1	0	0
136	1	0	0	0	0	1
160	1	0	0	0	1	0
168	0	0	1	1	0	0
192	0	1	0	1	0	0
200	1	0	0	0	0	1
224	1	0	0	0	1	0
232	0	0	1	1	0	0
256	0	1	0	1	0	0
264	1	0	0	0	0	1
288	1	0	0	0	1	0
296	0	0	1	1	0	0
320	0	1	0	1	0	0
328	1	0	0	0	0	1
352	1	0	0	0	1	0
360	0	0	1	1	0	0
384	0	1	0	1	0	0
392	1	0	0	0	0	1
416	1	0	0	0	1	0
424	0	0	1	1	0	0
448	0	1	0	1	0	0
456	1	0	0	0	0	1
480	1	0	0	0	1	0
488	0	0	1	1	0	0
512	0	1	0	1	0	0
520	1	0	0	0	0	1
544	1	0	0	0	1	0
552	0	0	1	1	0	0
576	0	1	0	1	0	0
584	1	0	0	0	0	1
608	1	0	0	0	1	0
616	0	0	1	1	0	0
640	0	1	0	1	0	0

648	1	0	0	0	0	1
672	1	0	0	0	1	0
680	0	0	1	1	0	0
704	0	1	0	1	0	0
712	1	0	0	0	0	1
736	1	0	0	0	1	0
744	0	0	1	1	0	0
768	0	1	0	1	0	0
776	1	0	0	0	0	1
800	1	0	0	0	1	0
808	0	0	1	1	0	0
832	0	1	0	1	0	0
840	1	0	0	0	0	1
864	1	0	0	0	1	0
872	0	0	1	1	0	0
896	0	1	0	1	0	0
904	1	0	0	0	0	1
928	1	0	0	0	1	0
936	0	0	1	1	0	0
960	0	1	0	1	0	0
968	1	0	0	0	0	1
992	1	0	0	0	1	0



Conclusion :

By adhering to method ascribed above, it is possible to successfully design a ‘Traffic Signal Controller’ in Xilinx given a set of time and signal specifications. This system may provide a superior traffic signal management system that the one that is currently in use.