Task:

Build a dynamic traffic light controller for a 4-way intersection that adapts to the flow in traffic.

Selected Methodology:

Following are the attributes of the selected Model:

- Traffic signals turns: North -> East -> South -> West -> North
- Every traffic signal sequence of working: RED -> RED-YELLOW -> GREEN -> YELLOW -> RED
- Adaptive traffic signal using sensors: sensor_north, sensor_south, reset.

Reset	**Sensor north**	**Sensor south**	**North traffic**	**East traffic**	**South traffic**	**West traffic**	**timing**
0	0	0	Always RED	working	Always RED	working	5 secs
0	0	1	Always RED	working	working	working	3 secs
0	1	0	working	working	Always RED	working	3secs
0	1	1	working	working	working	working	2secs
1	x	x	Always RED	working	Always RED	Always RED	-------

Figure no. 1 State table of designed System

- Adaptive timings are:

Red-Yellow → Green	$1\ secs$
Green → Yellow	$(timing - 1)\ secs$
Yellow → Red	$1\ secs$
Red → Red-Yellow	on call

State Machine:

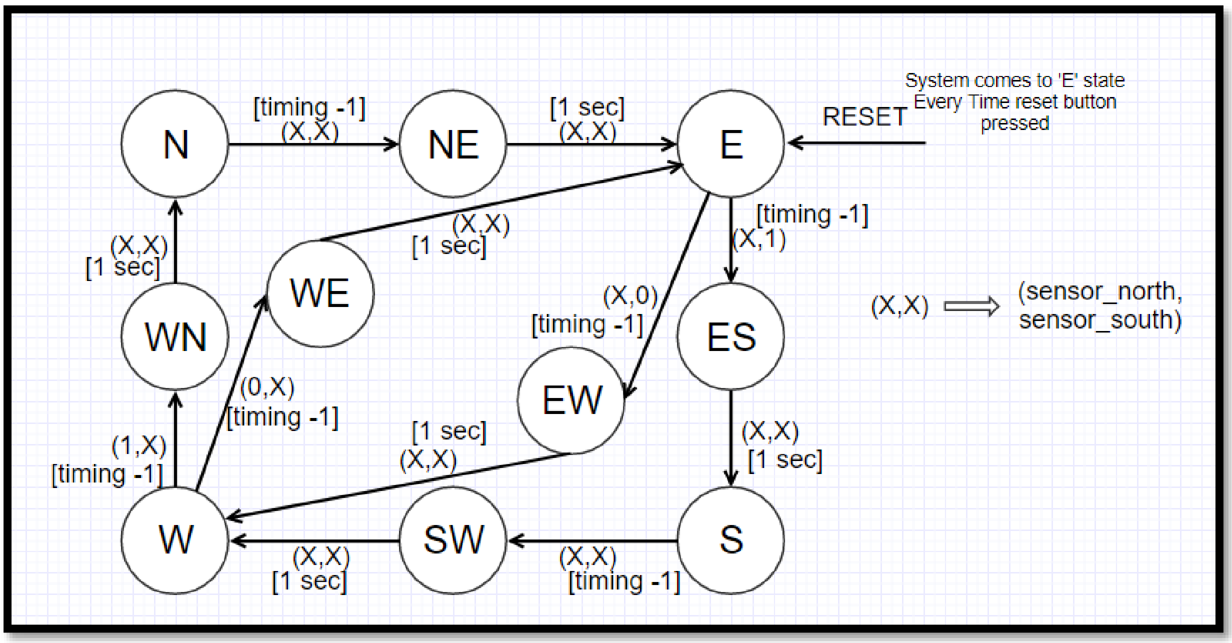


Figure no. 2 Moore FSM of designed System

Outputs:

*FSM is **Moore FSM** because output only depending upon the present state.*

States	North Traffic			South Traffic			East Traffic			West Traffic		
	R	Y	G	R	Y	G	R	Y	G	R	Y	G
E	1	0	0	1	0	0	0	0	1	1	0	0
ES	1	0	0	1	1	0	0	1	0	1	0	0
EW	1	0	0	1	0	0	0	1	0	1	1	0
S	1	0	0	0	0	1	1	0	0	1	0	0
SW	1	0	0	0	1	0	1	0	0	1	1	0
W	1	0	0	1	0	0	1	0	0	0	0	1
WE	1	0	0	1	0	0	1	1	0	0	1	0
WN	1	1	0	1	0	0	1	0	0	0	1	0
N	0	0	1	1	0	0	1	0	0	1	0	0
NE	0	1	0	1	0	0	1	1	0	1	0	0

Verilog Code:

Traffic.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
//////////////
// Company: SEECS, NUST
// Engineer: Saad Iqbal, Noor Muhammad Malik, Tayyab Hassan
//
// Create Date:    13:46:47 11/11/2017
// Design Name:   Adaptive Traffic Control
// Module Name:    Traffic
// Project Name:
// Target Devices: Virtex-5 ML-507 Board
// Tool versions:
// Description:
//
// Dependencies: one_sec_clk.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
//////////////
module Traffic(clk,
                           reset,
                           sensor_north,
                           sensor_south,
                           North_LEDs,
                           East_LEDs,
                           South_LEDs,
                           West_LEDs
                           );
     input                      clk, reset, sensor_north,
sensor_south;
     output reg [2:0] North_LEDs, East_LEDs, South_LEDs, West_LEDs;

     parameter            T1 = 4'd2, T2= 4'd2, T3= 4'd4, T4= 4'd8;
     parameter            E = 4'b0000, ES = 4'b0001, EW = 4'b0010,
     S = 4'b0011, SW = 4'b0100;
     parameter            W = 4'b0101, WN = 4'b0110, WE = 4'b0111,
     N = 4'b1000, NE = 4'b1001;
     parameter            C1 = 3'b100, C2 = 3'b110,  C3 = 3'b010,
     C4 = 3'b001;

     reg            [3:0] p_state = 4'b0000, n_state = 4'b0001;
     wire                       clk_p5s;
     reg                        clk_adapt = 0;
     reg                        transition = 0;
     reg            [3:0] require_delay;
     reg            [3:0] count = 4'b0001;

     assign                     reset_n = ~reset;

//measuring current required delay
     always @(sensor_north, sensor_south, transition)
     begin
          if (transition)
```

```
                require_delay = T1;
           else if (sensor_north & sensor_south)
                require_delay = T2;
           else if (!sensor_north & !sensor_south)
                require_delay = T4;
           else
                require_delay = T3;
      end

//obtaining 0.5 sec clock
      pone_sec_clk clock1 (.clk_in     (clk),
                                        .clk_out (clk_p5s)
                                        );

//obtaining adaptive delay clock
      always @(posedge clk_p5s, negedge reset_n)
      begin
           if (reset_n == 0)
           begin
                count <= 4'd1;
                clk_adapt <= 0;
           end
           else
           begin
                if (count == require_delay)
                begin
                     count <= 4'd1;
                     clk_adapt <= 1;
                end
                else
                begin
                     count <= count + 4'd1;
                     clk_adapt <= 0 ;
                end
           end
      end

//obtaining current state and transition bit value

      always @(sensor_north, sensor_south, p_state)
      begin
           case (p_state)
                E:
                begin
                     if (sensor_south)
                     begin
                          n_state = ES;
                          transition = 0;
                     end
                     else
                     begin
                          n_state = EW;
                          transition = 0;
                     end
                end
                ES:
```

```
                begin
                      n_state = S;
                      transition = 1;
                end
                EW:
                begin
                      n_state = W;
                      transition = 1;
                end
                S:
                begin
                      n_state = SW;
                      transition = 0;
                end
                SW:
                begin
                      n_state = W;
                      transition = 1;
                end
                W:
                begin
                      if (sensor_north)
                      begin
                            n_state = WN;
                            transition = 0;
                      end
                      else
                      begin
                            n_state = WE;
                            transition = 0;
                      end
                end
                WN:
                begin
                      n_state = N;
                      transition = 1;
                end
                WE:
                begin
                      n_state = E;
                      transition = 1;
                end
                N:
                begin
                      n_state = NE;
                      transition = 0;
                end
                NE:
                begin
                      n_state = E;
                      transition = 1;
                end
                default:
                begin
                      n_state = E;
                      transition = 0;
```

```
                    end
               endcase
          end

//assigning value to present state with async reset
          always @(posedge clk_adapt, negedge reset_n)
          begin
               if (reset_n==0)
                    p_state <= E;
               else
                    p_state <= n_state;
          end

//assign output depending upon p_state
          always @(p_state)
          begin
               case (p_state)
                    E:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C4;
                         South_LEDs = C1;
                         West_LEDs  = C1;
                    end
                    ES:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C3;
                         South_LEDs = C2;
                         West_LEDs  = C1;
                    end
                    EW:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C3;
                         South_LEDs = C1;
                         West_LEDs  = C2;
                    end
                    S:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C1;
                         South_LEDs = C4;
                         West_LEDs  = C1;
                    end
                    SW:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C1;
                         South_LEDs = C3;
                         West_LEDs  = C2;
                    end
                    W:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C1;
                         South_LEDs = C1;
                         West_LEDs  = C4;
                    end
                    WN:
                    begin
                         North_LEDs = C2;
                         East_LEDs  = C1;
                         South_LEDs = C1;
                         West_LEDs  = C3;
                    end
                    WE:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C2;
                         South_LEDs = C1;
                         West_LEDs  = C3;
                    end
                    N:
                    begin
                         North_LEDs = C4;
                         East_LEDs  = C1;
                         South_LEDs = C1;
                         West_LEDs  = C1;
                    end
                    NE:
                    begin
                         North_LEDs = C3;
                         East_LEDs  = C2;
                         South_LEDs = C1;
                         West_LEDs  = C1;
                    end
                    default:
                    begin
                         North_LEDs = C1;
                         East_LEDs  = C4;
                         South_LEDs = C1;
                         West_LEDs  = C1;
                    end
               endcase
          end
     endmodule
```

pone_sec_clk.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: SEECS, NUST
// Engineer: Saad
//
// Create Date:    13:47:32 11/11/2017
// Design Name:
// Module Name:    pone_sec_clk
// Project Name:
```

```
// Target Devices: Virtex-5 ML-507
// Tool versions:
// Description: Generate 1Hz clk from 100 MHz source
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
//////////////
module pone_sec_clk(clk_in,
                                  clk_out
                                  );

      input           clk_in;
      output reg      clk_out = 0;

      reg [26:0] clk_reg = 27'd0;
//obtain clk_out of 0.5 second period
//clk_in is 100MHz
      always @(posedge clk_in)
      begin
           if(clk_reg == 27'd50000000)
           begin
                clk_out <= 1;
                clk_reg <= 27'd1;
           end
           else
           begin
                clk_reg <= clk_reg+ 27'd1;
                clk_out <= 0;
           end
      end
endmodule
```

Traffic.ucf

```
# PlanAhead Generated physical constraints

NET "clk" LOC = AH15;
NET "reset" LOC = AJ6;
NET "sensor_north" LOC = U8;
NET "sensor_south" LOC = V8;

NET "East_LEDs[0]" LOC = H18;
NET "East_LEDs[1]" LOC = L18;
NET "East_LEDs[2]" LOC = G15;
NET "North_LEDs[0]" LOC = AF13;
NET "North_LEDs[1]" LOC = AG23;
NET "North_LEDs[2]" LOC = AG12;
NET "South_LEDs[0]" LOC = AD26;
NET "South_LEDs[1]" LOC = G16;
NET "South_LEDs[2]" LOC = AD25;
NET "West_LEDs[0]" LOC = AD24;
NET "West_LEDs[1]" LOC = AE24;
NET "West_LEDs[2]" LOC = AF23;
```

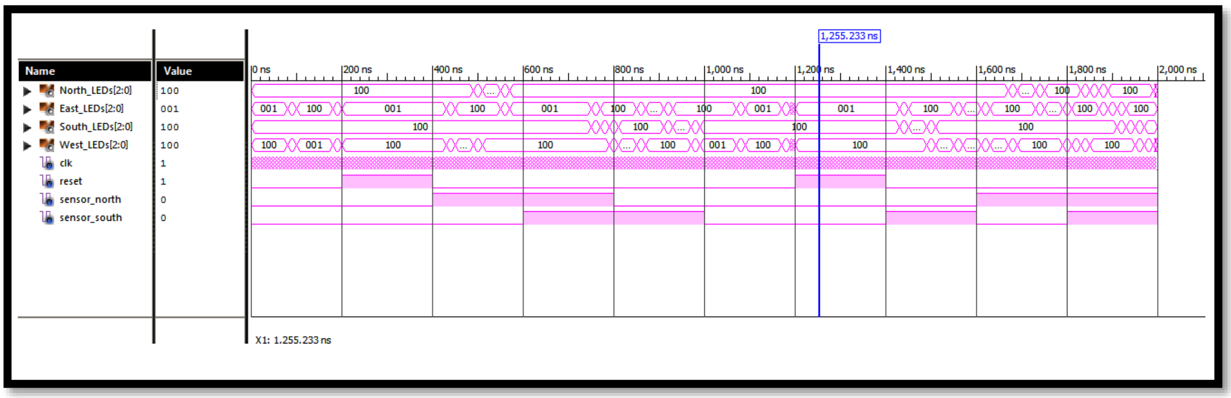
TEST BENCH for the designed system:

NOTE: before running test bench, *pone_sec_clk.v* one line should change as:

if(clk_reg == 27'd50000000) changed to if(clk_reg == 27'd5)

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////
////////////
// Company:
// Engineer:
//
// Create Date:   17:58:16 11/11/2017
// Design Name:   Traffic
// Module Name:   C:/Users/IQBAL/Documents/Xilinx/Lab_7/test.v
// Project Name:  Lab_7
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: Traffic
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
////////////

module test;

	// Inputs
	reg clk;
	reg reset;
	reg sensor_north;
	reg sensor_south;

	// Outputs
	wire [2:0] North_LEDs;
	wire [2:0] East_LEDs;
	wire [2:0] South_LEDs;
	wire [2:0] West_LEDs;

	// Instantiate the Unit Under Test (UUT)
	Traffic uut (
		.clk(clk),
		.reset(reset),
		.sensor_north(sensor_north),
		.sensor_south(sensor_south),
		.North_LEDs(North_LEDs),
		.East_LEDs(East_LEDs),
		.South_LEDs(South_LEDs),
            .West_LEDs(West_LEDs)
      );

      initial begin
            // Initialize Inputs
            clk = 0;
            reset = 0; sensor_north = 0;     sensor_south = 0;
      #200;
            reset = 1; sensor_north = 0;     sensor_south = 0;
      #200;
            reset = 0; sensor_north = 1;     sensor_south = 0;
      #200;
            reset = 0; sensor_north = 1;     sensor_south = 1;
      #200;
            reset = 0; sensor_north = 0;     sensor_south = 1;
      #200;
            reset = 0; sensor_north = 0;     sensor_south = 0;
      #200;
            reset = 1; sensor_north = 0;     sensor_south = 0;
      #200;
            reset = 0; sensor_north = 0;     sensor_south = 1;
      #200;
            reset = 0; sensor_north = 1;     sensor_south = 0;
      #200;
            reset = 0; sensor_north = 1;     sensor_south = 1;
      #200;
      end
      always #1 clk=~clk;
endmodule
```

Verification of Circuit on ModelSim:



```
Name              Value
North_LEDs[2:0]   100
East_LEDs[2:0]    001
South_LEDs[2:0]   100
West_LEDs[2:0]    100
clk               1
reset             1
sensor_north      0
sensor_south      0
```

Figure no. 3 Verification of Circuit on ModelSim

RTL Circuit diagram:

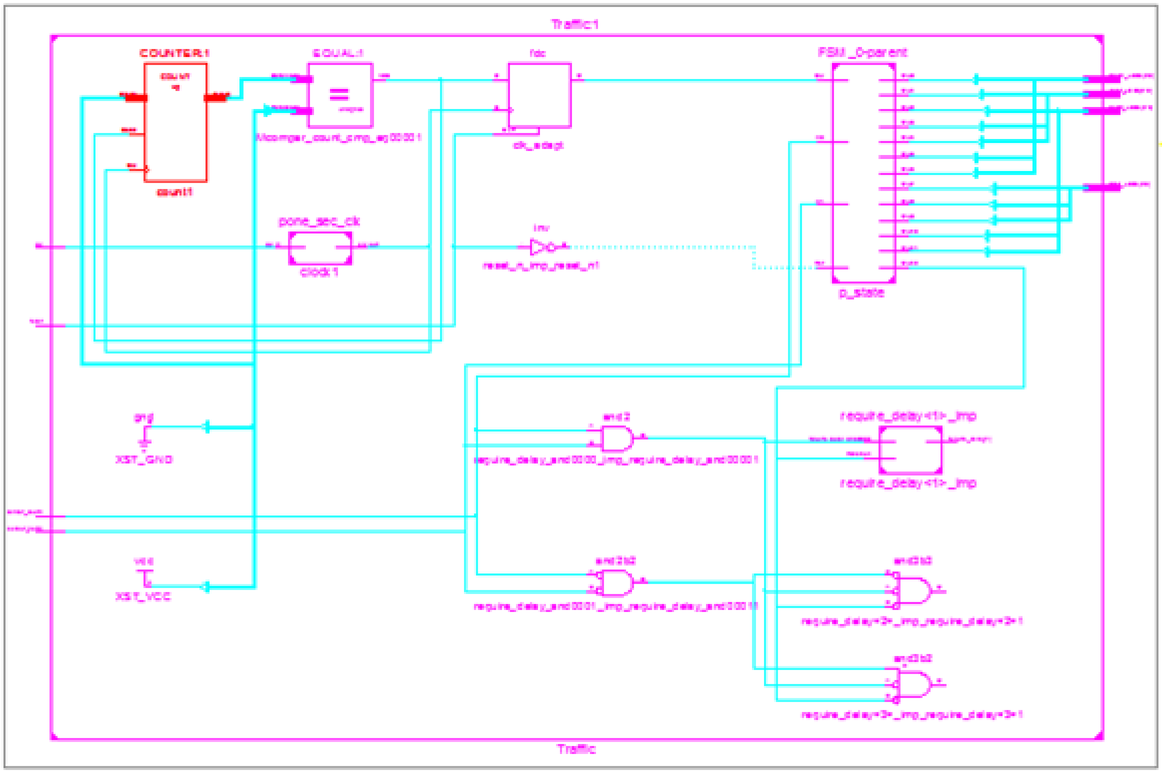


Figure no. 4 RTL Circuit Diagram

Significance of using FPGA:

Explain how FPGAs differ from your standard microcontrollers?

Designing for an FPGA requires a Hardware Description Language (HDL). HDLs are absolutely nothing at all like C. FPGA is parallel whereas microcontroller executes the program line by line. FPGAs focus way more on parallel execution. Sometimes you have to worry about how long your MCU's ISR takes to service the interrupt, and whether you'll be able to achieve your hard-real-time limits. However, an in FPGA there are lots of Finite State Machines (FSM) running all the time.

Cons: (reference to 8051)
    Price of FPGA is high, Difficult to understand and nothing like C.
Pros: (reference to 8051)
    Real-time, many FSM at a time and Close to hardware.