Design & Implementation of Traffic light control using Verilog on FPGA

17BEC004 (Kashyap Adodariya)

I. INTRODUCTION

FPGA is an integrated circuit which have array of configurable logic blocks connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. FPGA is advantageous compared to microcontroller in terms of number of IO (input & output) ports and performance. FPGA, an inexpensive solution compared to ASIC design; is effective with respect to cost in the case of production of large number of units but for fabrication in small number of units it is always costly and time consuming. So that I am design and implementation of traffic light control system using Verilog on Altera cyclone-2 FPGA kit. Verilog is used as HDL for circuit description to code the Traffic light control system.

II. COMPARISON WITH OTHER AVAILABLE METHODS/ALGORITHMS

A. Fix time based traffic light control

In this system any time only one light (red, green, yellow) is on for fix time interval. It doesn't care about traffic volume, any emergency vehicle priority. In that other two type, one is two-way control and second is one-way control. Two-way control, at any time two signal is RED or GREEN. One-way control, at any time only one signal is GREEN other is RED. In fix time based control require very less hardware [2].

B. Sydney Coordinated Adaptive Traffic System (SCATS)

The Sydney Coordinated Adaptive Traffic System abbreviated SCATS, is an intelligent transportation system that manages the dynamic (on-line, real-time) timing of signal phases at traffic signals, meaning that it tries to find the best phasing (cycle times, phase splits and offsets) for a traffic situation (for individual intersections as well as for the whole network). SCATS is based on the automatic plan selection from a library in response to the data derived from loop detectors or other road traffic sensors [3].

C. Grid based traffic light system

Grid based traffic light system, is divided in grid layout and each intersection consist one system. All grid point has a different algorithm to operate traffic light and all point communicate each other. Based on communication decide time cycle which is static or dynamic also. i.e. grid point working on SCATS so that according to traffic volume decide that point time cycle and that information pass on to nearby point and that point recalculate time cycle. That process is continuing and accumulate. This algorithm required more hardware and it is more complex.

III. SYSTEM BLOCK DIAGRAM

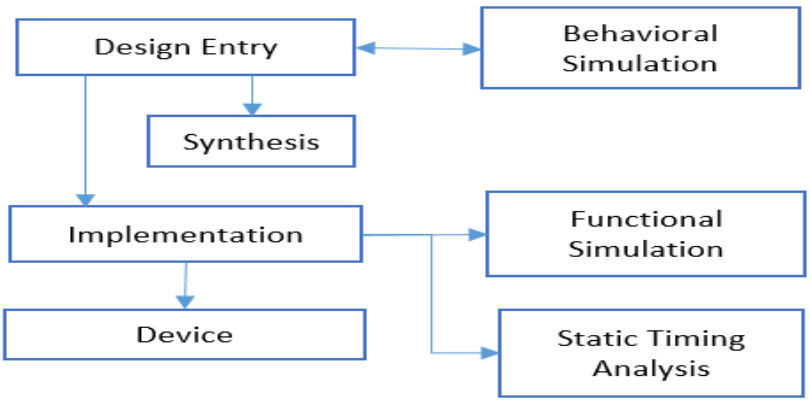

Figure 1: System Block diagram of Design traffic light control system on FPGA[4]

There are various techniques for design entry. Schematic based, Hardware Description Language and combination of both etc. Selection of a method depends on the design and designer. If the designer wants to deal more with Hardware, then Schematic entry is the good choice. When the design is complex or the designer thinks the design in an algorithmic way then HDL is the better choice. Language based entry is faster but lag in performance and density. HDLs represent a level of abstraction that can separate the designers from the details of the hardware implementation. Schematic based entry gives designers much more visibility into the hardware. It is the good choice for those who are hardware adapted to. Another method but rarely used is state-machines. It is the better choice for the designers who think the design as a series of states. But the tools for state machine entry are limited. In this documentation we are going to deal with the HDL based design entry. [2]

IV. ALGORITHMIC FLOW CHART

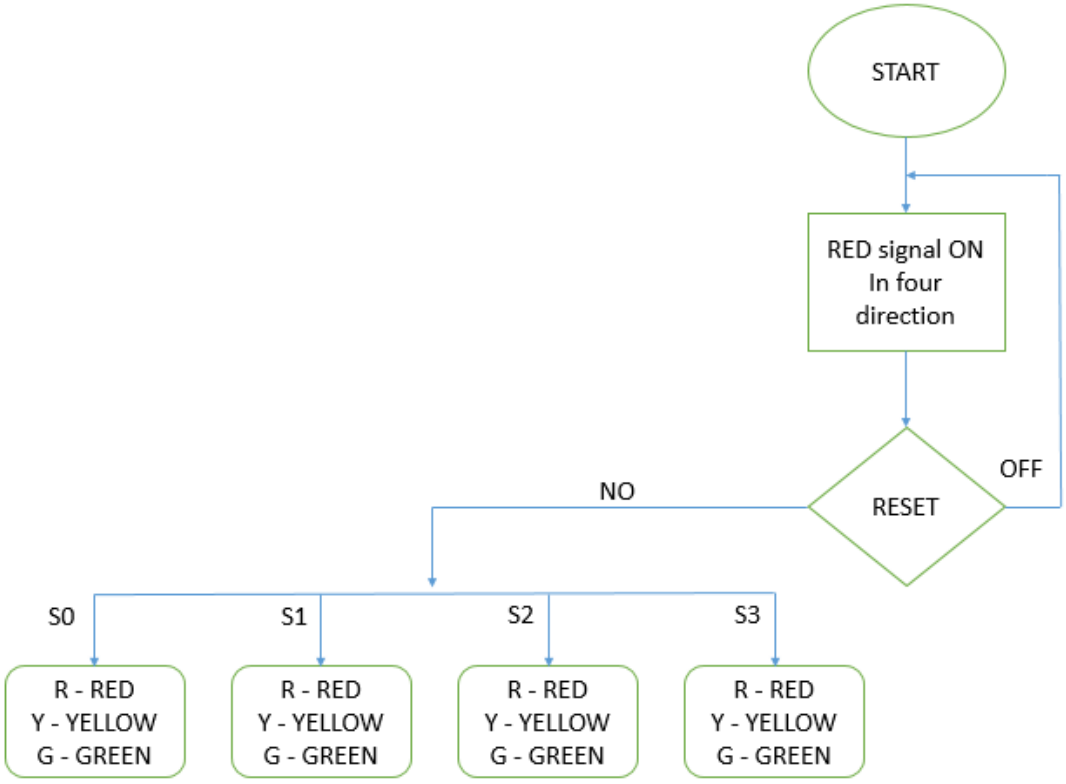

Figure 2: Fix time based traffic light control system flow chart[4]

design code for Traffic Light Controller using Finite State Machine(FSM). In this clk and rst_a are two input signal and n_lights, s_lights, e_lights and w_lights are 3-bit output signal. In output signal, "001" represents Green light, "010" represents Yellow light and "100" represents Red light. 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.

	Source State	Destination State
1	east	east
2	east	east_y
3	east_y	east_y
4	east_y	west
5	north	north
6	north	north_y
7	north_y	north_y
8	north_y	south
9	south	south
10	south	south_y
11	south_y	east
12	south_y	south_y
13	west	west
14	west	west_y
15	west_y	north
16	west_y	west_y

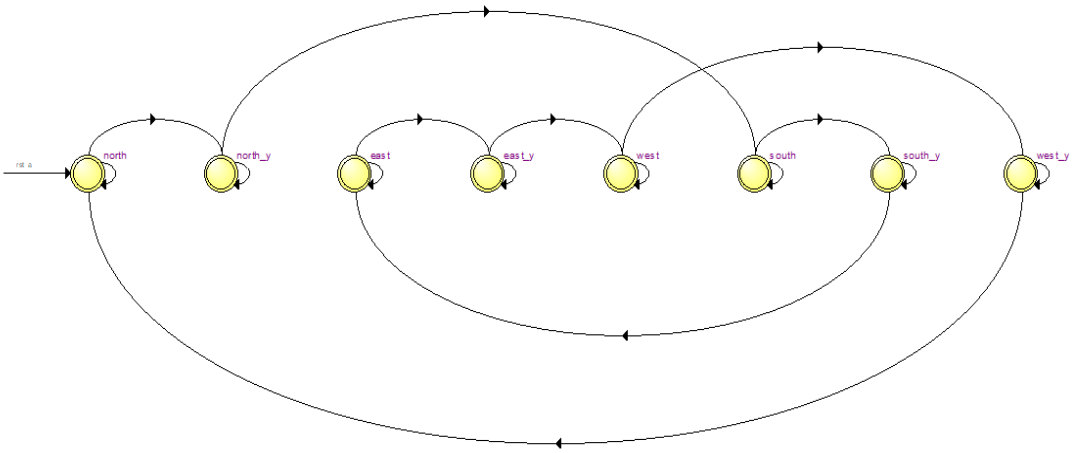

Figure 3: State dig. of traffic lights

V. RTL RESULTS

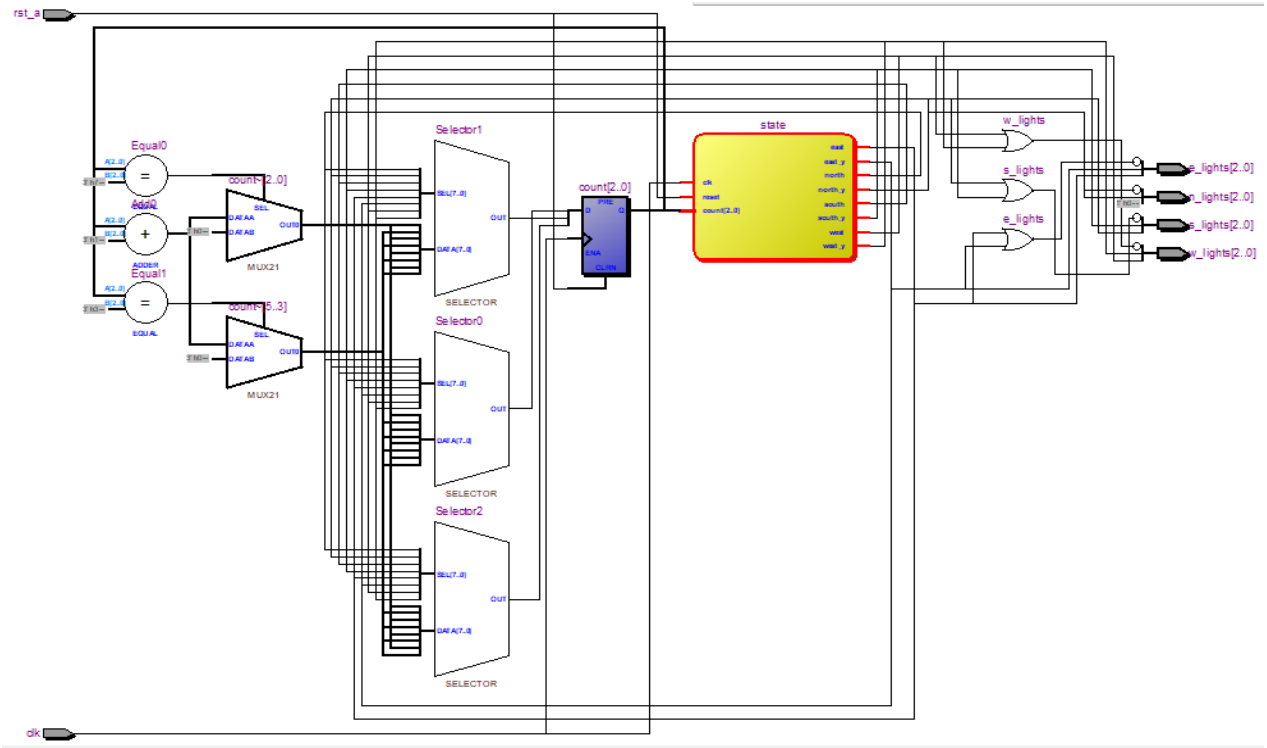

Figure 4: RLT View

VI. SIMULATION RESULTS

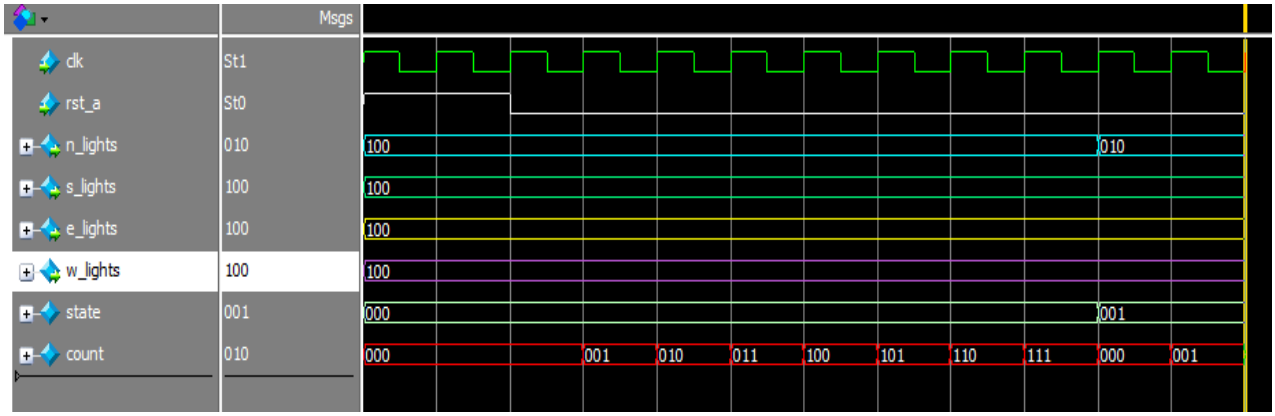

Figure 5:Simulation of initial stage where rst_a become high to low transition

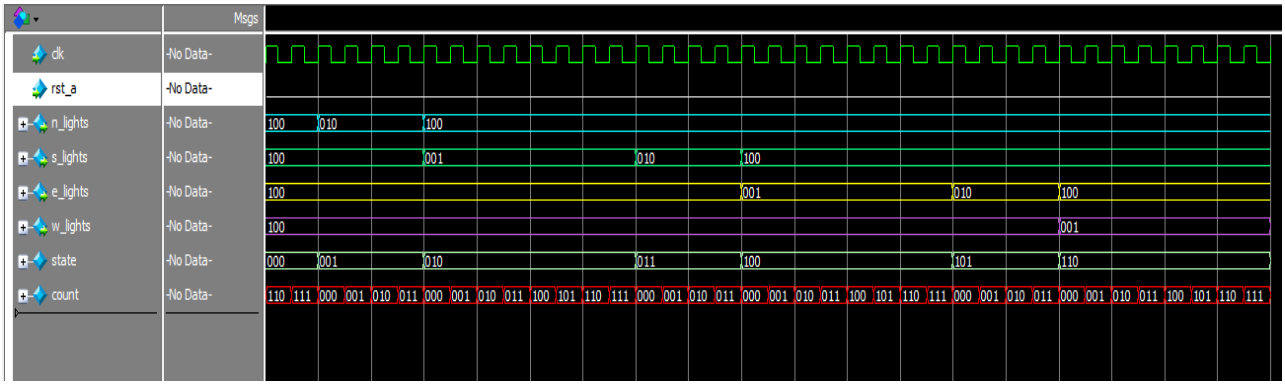

Figure 6: Simulation of different state where rst_a become low

VII. LEARNING OUTCOME AND CONCLUSION

In this project, I learnt about how to represent finite state machine in Verilog HDL and I also learnt to write testbench. In this project, present four-way fix time based traffic light control system. As per the result show every eight cycle system change the state and according to the any one direction light state which is represent in 3-bit was changed and except the direction other was remain previous state. This process was keep going until and unless rest become high. So this fix time based mechanism was working on time cycle. Now a day this algorithm is not worked. Instated of using this algorithm we go for adaptive time based system which was discussed above.

VIII. REFERENCES

[1] "FPGA-Based Advanced Real Traffic Light Controller System Design". El-Medany, W.M. ; Univ. of Bahrain, Sakhir ; Hussain, M.R.DOI:10.1109/IDAACS.2007.4488383

[2] Methodology of state represent in Verilog https://www.academia.edu/21200096/DESIGN_AND_IMPLEMENTATION_OF_TRAFFIC_LIGHTS_CONTROLLER_USING_FPGA_A_Project_Based_Laboratory_Report_in_partial_fulfilment_for_the_award_of_III_IV_B.Tech_-I_Semester_Submitted_by_Lab_Instructor [Accessed March 31, 2020]

[3] Alogorithm of Sydney Coordinated Adaptive Traffic System https://en.wikipedia.org/wiki/Sydney_Coordinated_Adaptive_Traffic_System. [Accessed March 31, 2020]

[4] "Traffic Light Controller Using Fpga" D.Bhavana et al. Int. Journal of Engineering Research and Applications. ISSN : 2248-9622, Vol. 5, Issue 4, (Part -6) April 2015, pp.165-16

APPENDIX A

```
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'b100;
                    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'b100;
                    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'b100;
               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'b100;
               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'b100;
            end // case: east_y
         endcase // case (state)
      end // always @ (state)
   endmodule
```

Testbench of traffic light control system:

```
`timescale 1ns/1ps
module traffic_control_tb;
wire [2:0] n_lights,s_lights,e_lights,w_lights;
reg clk,rst_a;
```

```
traffic_control DUT (n_lights,s_lights,e_lights,w_lights,clk,rst_a);
always
begin
clk = 0;
#10;
clk = 1;
#10;
end

initial
 begin
  rst_a=1'b1;
  #15;
  rst_a=1'b0;
  #1000;
  $stop;
 end
endmodule
```