Familiarization of hardware design using Verilog HDL with it's simulation and synthesis in FPGA.

A MINI PROJECT

Submitted in Partial Fulfillment for Requirements of the Degree of

BACHELOR OF TECHNOLOGY

(Computer Science And Technology)

By

Meduri Venkata Shivaditya *(510516053)*

Shruti Singh *(510516001)*

Nikhil Raj *(510516050)*

Anunay Kumar *(510516038)*

Under the guidance of

Dr.Surajeet Ghosh

3rd Semester (December 2017)

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY

FORWARDING

We hereby forward the progress report entitled "**Familiarization of hardware design using verilog HDL with it's simulation and synthesis in FPGA**" submitted by Venkata Shivaditya Meduri (Enrollment number: 510516053),Nikhil Raj(Enrollment number: 510516050),Anunay Kumar (Enrollment number: 510516038) and Shruti Singh (Enrollment number: 510516001) under our guidance and supervision in partial fulfillment of the requirements for the degree of 'Bachelor of Technology' in the department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur.

(Dr.Surajeet Ghosh)

Assistant Professor

Department of Computer Science and Engineering

Indian Institute of Engineering Science And Technology, Shibpur

ABSTRACT

There has been exponential advancement in the world of technology and hardware. New hardware designs keep emerging in the market everyday. The new hardware before being launched is vigorously simulated and tested under various conditions.

The report deals with the concept of simulation and synthesis in the field of hardware design. Simulation is conceived in the “Model Sim Student Edition” and it’s implemented in FPGA board –“Spartan 3”.

Since FPGA boards need a hardware description language to communicate with, Verilog has been familiarized thoroughly. The process of synthesis in FPGAs is also described. Traffic control system is simulated and synthesis of stepper motor is also included in the report.

ACKNOWLEDGEMENT

We dived deep into the ocean of hardware design after taking this project where we were completely an alien to this new world. But lots of people helped us remain a float. The zeal of our teammate Shruti Singh served as a beacon of light in this untrodden path. Her hard work and discipline throughout this project was unmatched and deserves commendation.

Our mentor Dr.Surajeet Ghosh believed in us and assigned us this project in first place. We received constant support from Arrendhu Bhaiya who helped us in working of FPGA. Our friend Ujjwal and Gupta deserves special mention. Without their help this technical report would not have been possible.

We extend our heartfelt thanks to our DL lab instructor Suranto sir. He shared lots of insights about FPGA and it’s architecture.

Friends, family and the almighty may not have contributed to this project but without their support and blessing this project would not have been conceived.

CONTENTS

1.Familiarization of
Verilog Hardware Description Language

1.1HDL(*Hardware Description Language)*

A hardware description language or HDL is a specialized computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital logic circuits. A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. A hardware description language looks much like a programming language such as C; it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time. With the advent of HDLs designers no longer have to manually place gates to build digital circuits. They could describe in the form of data flow or functionality. HDL are used for simulation of system boards, interconnect buses, **FPGA**s (Field Programmable Gate Array) and **PAL**s (Programmable Array Logic).

1.2Importance of HDLs

HDLs have many advantages compared to traditional schematic-based design.

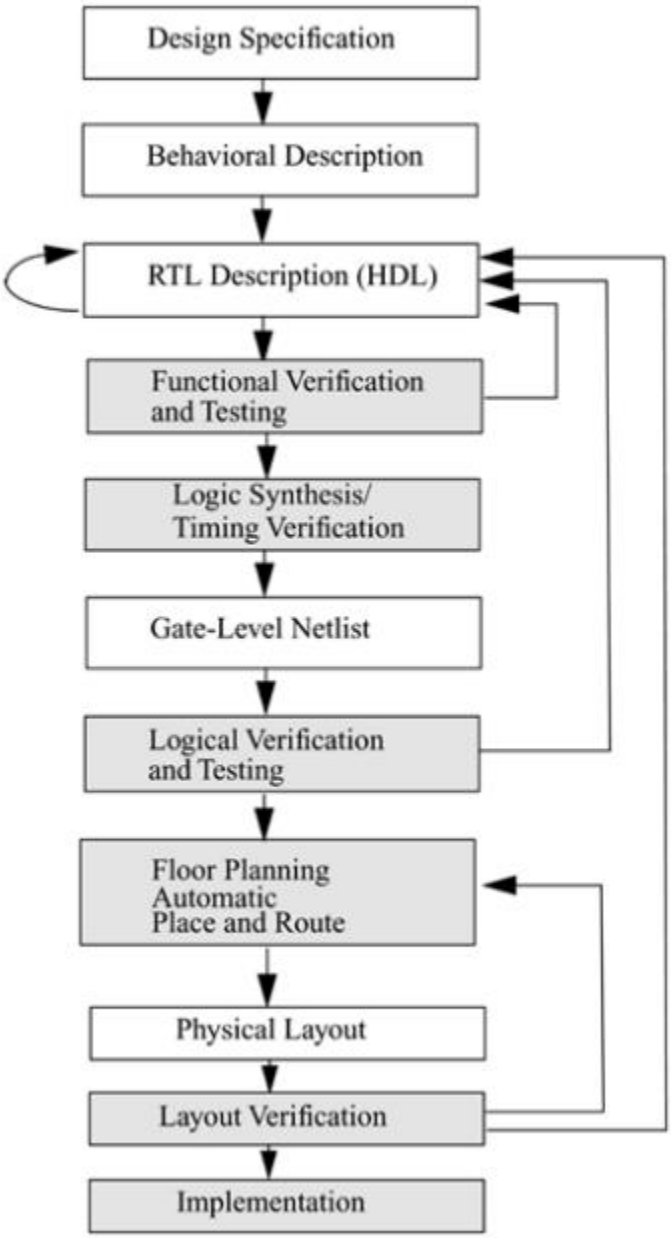
- Designs can be described at a very abstract level by use of HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new emerges designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.

- By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug a later time in the gate-level netlist or physical layout is minimized.

- Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics are almost incomprehensible for very complex designs.

With rapidly increasing sophisticated EDA tools, HDLs are now dominant method for large digital designs. No digital circuit designer can afford to ignore HDL-based design.

1.3Typical Design Flow



Drawing 1: Flowchart depicting the sequential process of the hardware design

1.4 VerilogHDL

Verilog HDL is one of the most used HDLs. It can be used to describe designs at four levels of abstraction:

1. **Behavioral or Algorithmic level**

   This is the highest level of abstraction provided by Verilog HDL. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details. Designing at this level is very similar to C programming.

2. **Dataflow or Register transfer level (RTL)**

   At this level, the module is designed by specifying the data flow. The designer is aware of how data flows between hardware registers and how the data is processed in the design.

3. **Gate level**

   The module is implemented in terms of logic gates and interconnections between these gates. Design at this level is similar to describing a design in terms of a gate-level logic diagram.

4. **Switch level (the switches are MOS transistors inside gates)**

   This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, storage nodes, and the interconnections between them. Design at this level requires knowledge of switch-level implementation details.

1.5 Design Methodologies

Digital design methods are of two types:

5. ***Top-down design method*** :

   In this design method we first define the top-level block and then we build necessary sub-blocks, which are required to build the top-level block. Then the sub-blocks are divided further into smaller-blocks, and so on. The bottom level blocks are called as leaf cells. By saying bottom level it means that the leaf cell cannot be divided further.

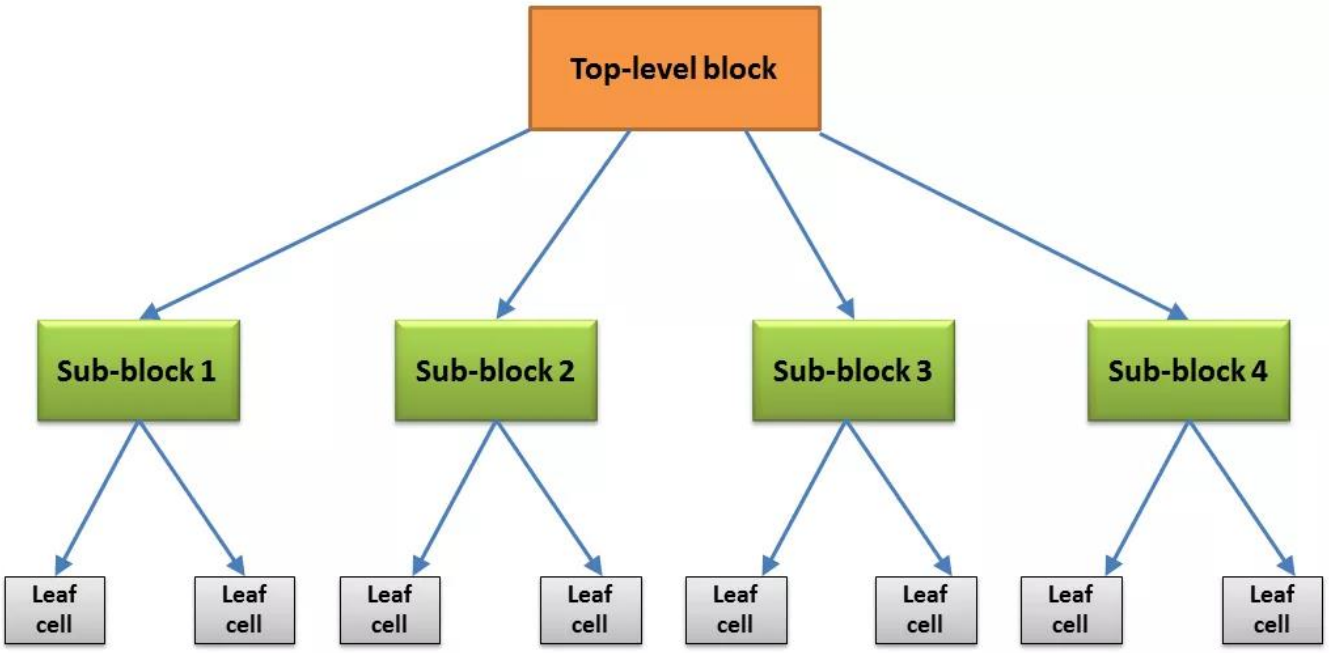


Figure 2: Hierarchy of Blocks

6. ***Bottom-up design method:***

In a bottom-up design methodology, we first identify the building blocks that are available to us, We build bigger cells, using these building blocks. These cells are then used for higher level blocks until we build the the top-level block in the design.

In general a combination of both types is used. These types of design methods helps the design architects, logics designers, and circuit designers. Design architects gives specifications to the logic designers, who follow one of the design methods or both. They identify the leaf cells. Circuit designers design those leaf cells, and they try to optimize leaf cells in terms of power, area, and speed. Hence all the design goes parallel and helps finishing the job faster.

1.6Components of a Simulation

Once a design block is completed, it must be tested. The functionality of the design block can be tested by applying stimulus and checking results. We call such a block the stimulus block. It is good practice to keep the stimulus and design blocks separate. The stimulus block can be written in Verilog. A separate language is not required to describe stimulus. The stimulus block is also commonly called a test bench. Different test benches can be used to thoroughly test the design block. Two styles of stimulus application are possible.

In the first style, the stimulus block instantiates the design block and directly drives the signals in the design block. In the given figure, the stimulus block becomes the top-level block. It manipulates signals clock and reset, and it checks and displays output signal q.

The second style of applying stimulus is to instantiate both the stimulus and design blocks in a top-level dummy module. The stimulus block interacts with the design block only through the interface. This style of applying stimulus is shown in Figure 2-7. The stimulus module drives the signals d_clk and d_reset, which are connected to the signals clk and reset in the design block. It also checks and displays signal c_q, which is connected to the signal q in the design block. The function of top-level block is simply to instantiate the design and stimulus blocks.

2.Familiarization of FPGA(Spartan-3)

2.1 INTRODUCTION

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL).

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

2.2 SPECIFICATION

FPGA board contains the following parts

1.Base board
2.Daughter Boards
3.Programming tool
4.Power supply

2.2.1 Base board

a. 32/16 TOGGLE/DIP switch for I/P selection.
b. 32/16 OPLEDs connected to the output of the FPGA.
c. 16*2 Alpha-numeric LCD display with the backlight.
d. Four-digit 7-segment display.
e. 4*4 Key matrix.
f. One PUSH BUTTON switch.
g. On board 10MHz oscillator.
h. 10MHz clock & one of four different frequency clocks (5MHz,1MHz,500KHz & 100KHz).
i. User I/O’s available for pattern generator and Logic Analyzer.
j. On-Board differently supply voltage generation to match the multi Volt with O/P LED indication.
k. Standard VGA interface through 15-pin D type connector.
l. PS/2 standard external keyboards/mouse interface.
m. RS232 serial interface through 9-pin D type connector.

n. All the voltages 1.2V,1.5V,2.5V,3.3V & VCC are indicated through LEDs.
o. 26-PIN FRC cable for connecting to ALS standard interface board like Stepper motor, ADC, DAC, Traffic light controller, elevator, printer interface etc.
p. Four sets of 20*2 female berg connector to plug the DAUGHTER BOADRD.

2.2.2 Daughter boards

There are three types of daughter boards. Any of these boards can be plugged on to the baseboard. The required program can be downloaded from the PC in to the FPGA on the daughter board.

The three types of boards are:

1.Containing SPARTAN-3 FPGA from XILINX
2.Containing XC9572 from XILINX
3.Containing Cyclone EP1C6 FPGA from ALTERA

2.3 HARDWARE DESCRIPTION

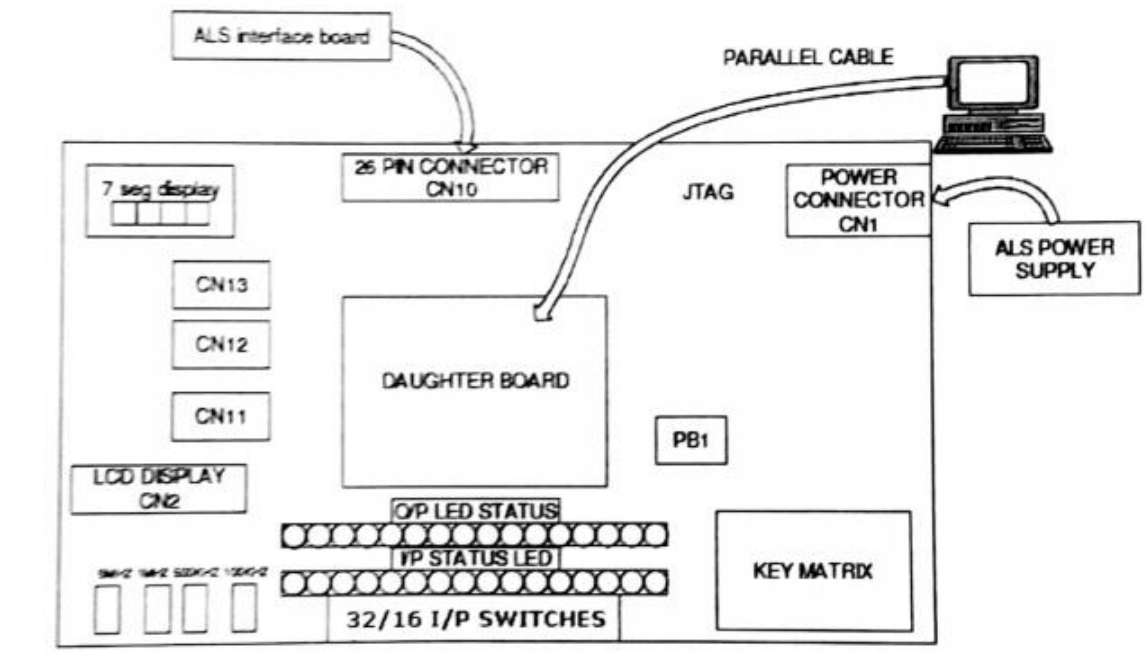


Figure 3: Connection of baseboard

The baseboard has a RS-232 serial port. The RS-232 transmits and receives signals that appears on the female DB9 connector, labeled as CN9. Use a standard straight-through serial cable to connect baseboard to the computer serial port. The device FPGA supplies serial output data as LVTTL or LVCMOS levels to the MAX232 device, which in turn converts the logic value to the appropriate RS232 voltage level. Likewise the MAX232 device converts the RS232 serial input data to LVTTL level for FPGA device. The clock and data signals are only driven when data transfer occur, & otherwise they are held in the idle state at logic high.

2.3.1 PS2 Mouse/keyboard port

The baseboard also includes PS/2 mouse/keyboard port and the standard 6-PIN mini – DIN connector, labeled CN4 on the board. Only pins 1 and 5 of the connector are connected to CN17.The PS/2 includes clock & data for both mouse and keyboard drive.

2.3.2 VGA-Port (Video Graphics Adaptor)

The baseboard includes a VGA display PORT DB15 connector ,indicated as CN3. Connect this port directly to most PC monitor or to a flat-panel LCD-Displays using a standard monitor cable.

2.3.3 26-Pin controller (CN10)

This connector is provided on baseboard to connect it to ALS interface board. The corresponding daughter board I/O lines are fed to this connector.

2.4 DAUGHTER BOARD-1(FPGA – XILINX-XCS350)

2.4.1 INTRODUCTION

The Spartan-3 family of Field-Programmable Gate Arrays is specially designed to meet the needs of high volume, Cost-sensitive consumer electronic applications. The eight member family offers densities ranging from 50,000 to five million – system gates. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications ; including broadband access, home networking, display/projection and digital television equipment.

2.4.2 Features:

a. Spartan-3 FPGA device XCS350-PQ208 of Xilinx.
b. It has 50k system gates.
c. It consists of 1728 logic cells.
d. And it employs 12k Distributed RAM.
e. It also has 72k Block RAM.
f. It is an FPGA IC in a PQFP208 pin in package with 124 I/O lines

Daughter Board features:

q. Push Button switch with PROG to initiate FPGA during master serial mode.
r. Optional PROM.
s. Four sets of 20*2 berg connector for plugging on to the base board
t. Mode selection jumpers(JP1)
u. Power supply from +3.3V, 2.5V and 1.2V are provided from the baseboard.

NOTE: Usage of PROM is optional, if the prom is used, user has to go for Master serial mode, which is , as given below otherwise user has to configure through Boundary-scan mode.

FPGA contains SRAM memory for configuration , thus the configuration is lost when the power is switched OFF. The FPGA is configured after the power ON by various means like JTAG, serial modes.

1. For master serial mode configuration, a serial PROM has to be used . Initially program is loaded to the serial PROM. Subsequently after every power on, the OPTIONAL PROM configures the FPGA. PROG switch can also be used to re-program the FPGA.

2. For Boundary-scan mode configuration, a 10-PIN FRC connector (CN1) is provided. In this mode, the program is directly loaded into the FPGA Through CN1 used as JTAG connector.

2.4.3 10-Pin JTAG Connector details

The CN1 JTAG connector on the Daughter board is used to configure the FPGA or to program the optional Xilinx OPTIONAL PROM. The JTAG cable is connected to the board through CN1 at one end and another end to the 10-Pin FRC connector of JTAG parallel adapter from the PC parallel port.

If user wants to work in boundary - scan mode then pin1 and pin2 of JP1 jumper has to be shorted .

2.4.3.1 PROG SWITCH

This Daughter Board provides a push button switch for initiating the configuration of the FPGA if only when the OPTIONAL PROM is used. This is used to configure FPGA from the already programmed ISP OPTIONAL PROM. Upon activation of the PROG signal, the ISP OPTIONAL PROM initiates the configuration of FPGA.

2.4.3.2 DONE LED

This LED is used on the FPGA Daughter board to indicate that the configuration of FPGA has occurred. During the program, it is ‘OFF’. If it is programmed successfully then it is switches ‘ON’ and remains ‘ON’ afterwards.

3.Implementation Of Stepper Motor Driver in FPGA

3.1Introduction:

A stepper motor or step motor or stepping motor is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor feedback(an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed.

A stepper motor with 6 wires in 4-phase unipolar, but with two common wires. They are of same colors.

There are two stages of sorting out to identify the type of wire in 6-wire unipolar stepper motor.

1.Isolate the common power wire by using an ohmmeter to check the resistances between pairs of wires. The common power wire will be the one with only half as much resistance between it and all the others.

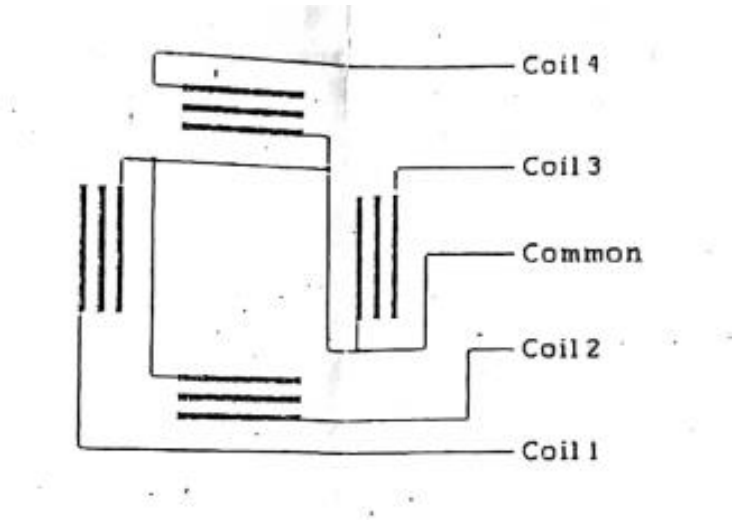


Figure 4:Output terminals of the Stepper Motor

This is because the common power wire only has one coil between it and each other wire, whereas each of the other wires have two coils between them. Hence half the resistance.

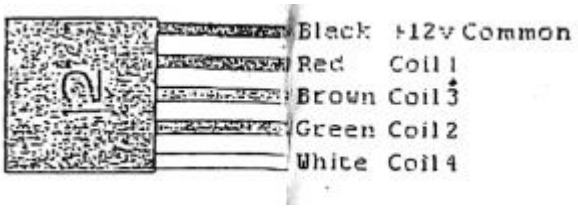


Figure 5:Wires of the Stepper Motor

3.2 Four wire connection:

The ULN2003 is a 7-bit 50 V 500mA TTL-input NPN Dalington driver. This is more than adequate to control a four-phase unipolar stepper such as the KM4P4-001.

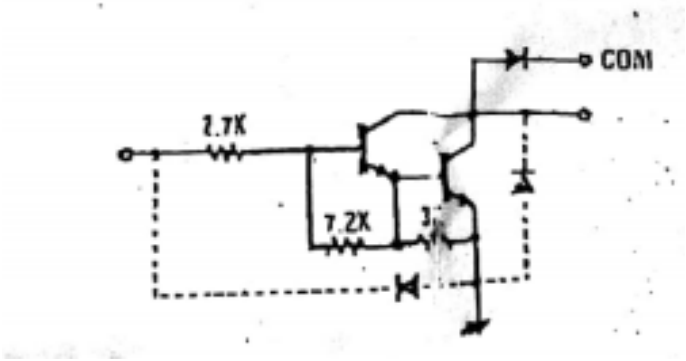


Figure 6: Internal circuit diagram of driver of ULN2003

3.3 Overall connection diagram:

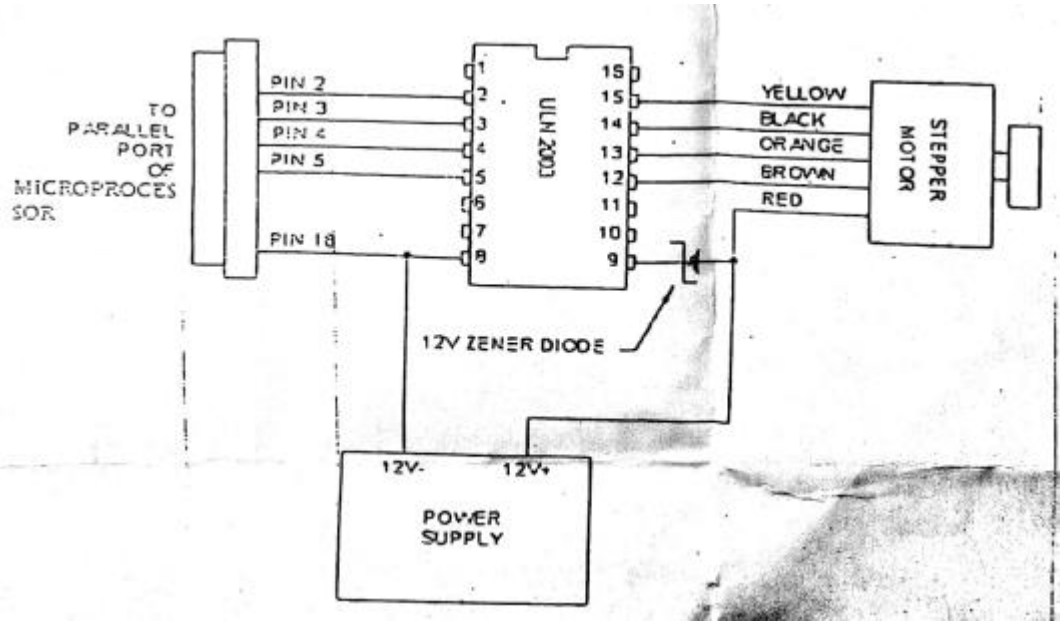


Figure 7:Connection of Stepper Motor to FPGA board

PS: We have not covered the electrical aspects of stepper motor as our objective is to realize it's working through FPGA board.

3.4 Implementation of the stepper motor driver through FPGA:

The four output terminals of the stepper motor red, blue green and yellow are connected to the specified output pins (to be mentioned in the UCF file) of the FPGA board .To drive or control the motor we have to give output to the pins in specific way.

The given figure is the state diagram of the terminals of the stepper motor where R,B,G and W stands for red, blue, green and white terminals of the stepper motor.

The following state diagram shows how to rotate the motor in clockwise direction. The motor rotates one step if we go from one state to other. If these states are repeated continuously the motor rotates without stopping.

If we follow the state diagram in opposite direction the motor rotates anticlockwise:

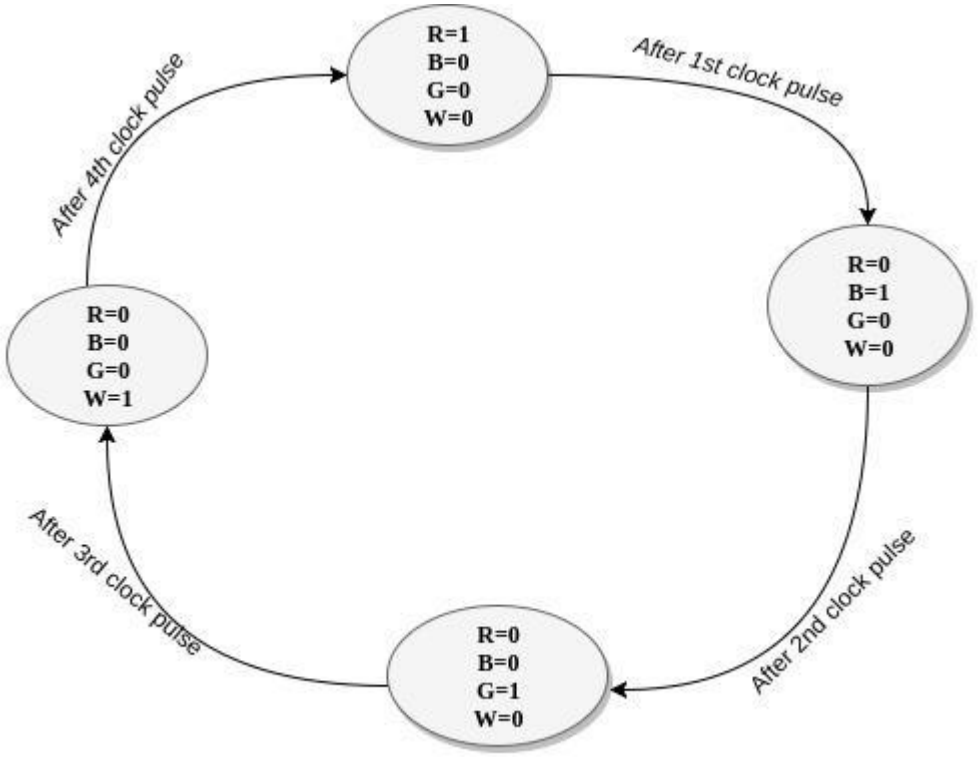


Figure 8:State Table of output terminals of the stepper motor

the Verilog code we define four outputs which corresponds to the input of the stepper motor. The next state can be triggered with the negative edge of the clock pulse defined in the code itself. Now, using these output pins we can control the stepper motor according to our application.

Mention output and input pins in the UCF file (included in the appendices). The speed of the stepper can be controlled by varying the frequency of the clock.

4.Simulation traffic light control system

4.1 Introduction

Traffic signals are integral part of modern life. There proper orientation can spell the difference between smooth flowing traffic and four lane grid lock. Proper operations entails precise timing, scanning through the states correctly and responding to outside inputs. The traffic light controller is designed to meet the above specifications .It consists of operations specifications which describes the different functions a controller must perform . It contains a detailed protocol for running the lights . Each of these requirements impose new constraints on the controller system . The controller to be designed controls the lights of a 4 way lane. This can be achieved by following a protocol . The model discussed below is a behavioral model.

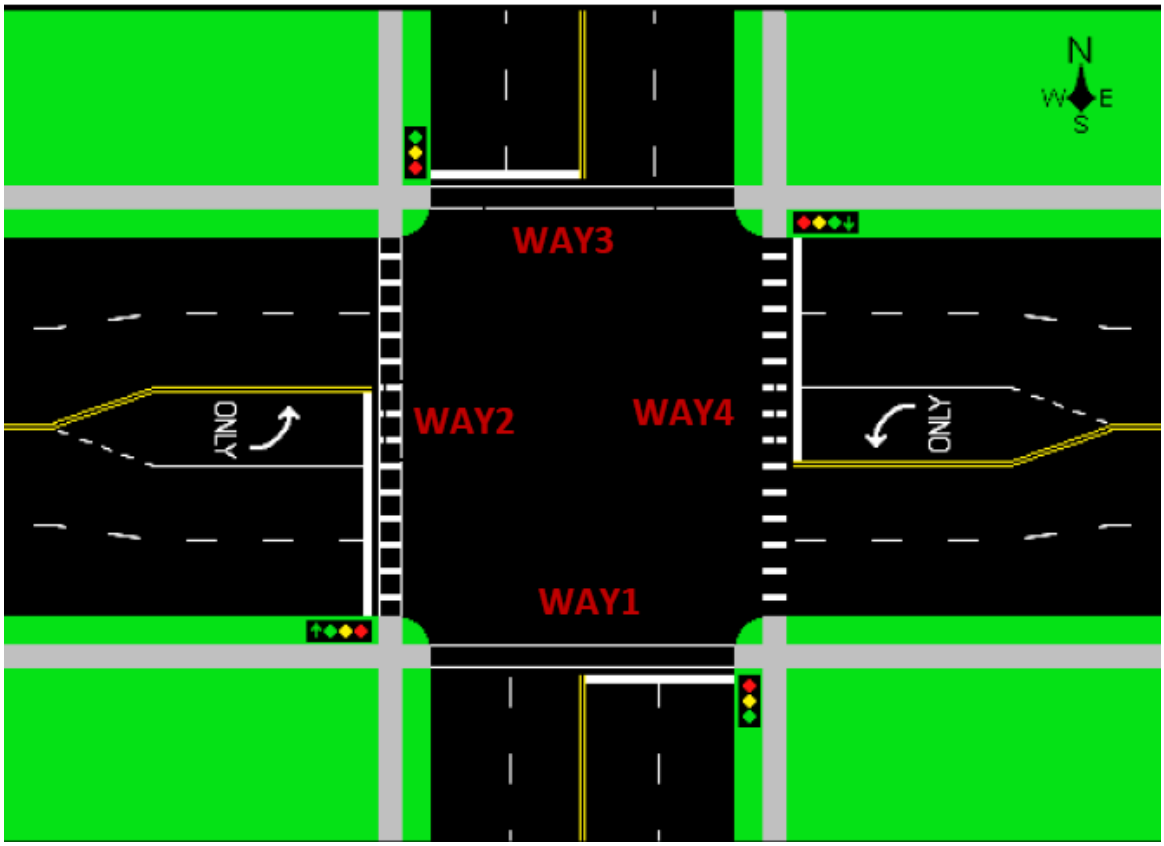


Figure 9:Working of Traffic Light Contoller

4.2 Protocol:

The real time traffic light controller is a complex piece of equipment which consists of power cabinet, main controller or processor, relays, control panel with switches or keys, communication ports etc.

In this project, a simple traffic light system for a 4 way intersection is implemented using an FPGA although it is not the ideal implementation for real life scenarios, it gives an idea of the process behind the traffic light control system.

The aim of the project is to implement a simple traffic light controller using FPGA the traffic is controlled in a pre-defined timing system. The working of the project is very simple and is explained below.

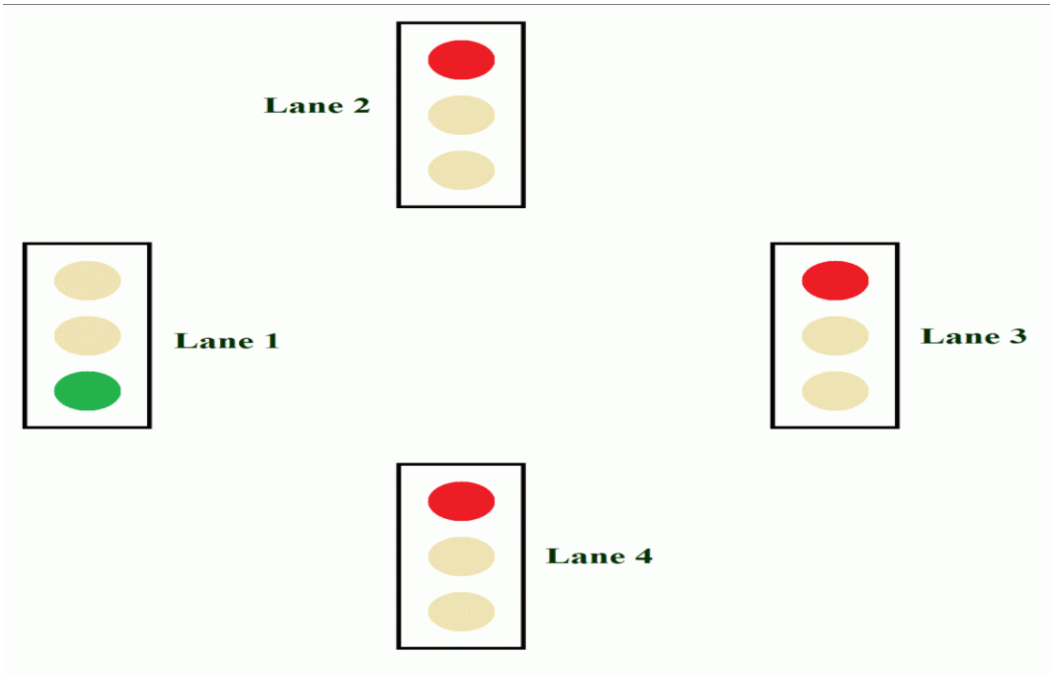


Figure 10

In that, first the Lane 1 gets its Green light turned. Hence, in all the other Lanes, their corresponding Red lights are turned on. After a time delay of predefined time say 5 seconds, the Green light in the Lane 3 must be turned on and the Green light in the Lane 1 must be turned off.

As a warning indicator, the Yellow light in Lane 1 is tuned on indicating that the red light is about to light up. Similarly, the yellow light in the Lane 3 is also turned as an indication that the green light about to be turned on.

The yellow lights in Lanes 1 and 3 are turned for a small duration say 2 seconds after with the red light in the Lane 1 is turned on and green light in Lane 3 is also turned on.

The green light in Lane 3 is also turned on for a predefined time and the process moves forward to Lane 4 and finally Lane 2. The system then loops back to Lane 1 where the process mentioned above will be repeated all over again.

Traffic light controller card consist of 12 Nos. point led arranged by 4Lanes. Each lane has Go(Green), Listen(Yellow) and Stop(Red) LED is being placed. Each LED has provided for current limiting resistor to limit the current flows to the LEDs.

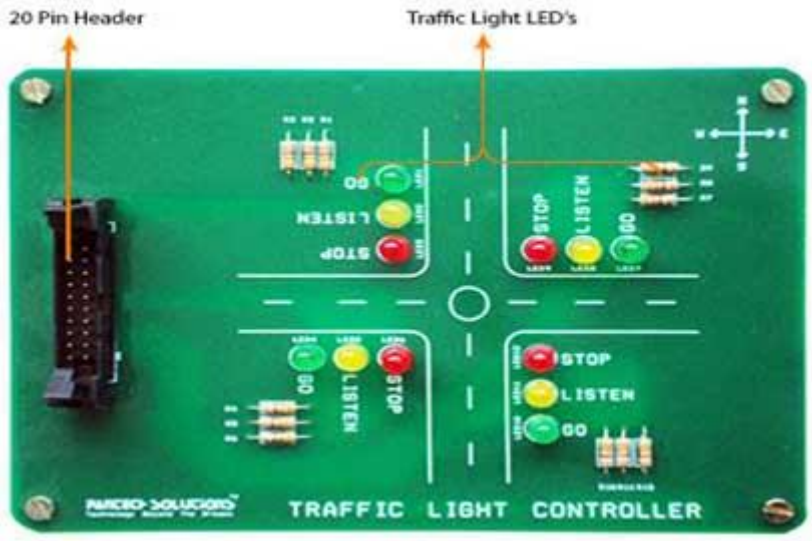


Figure 11:Structural model of the Traffic Light

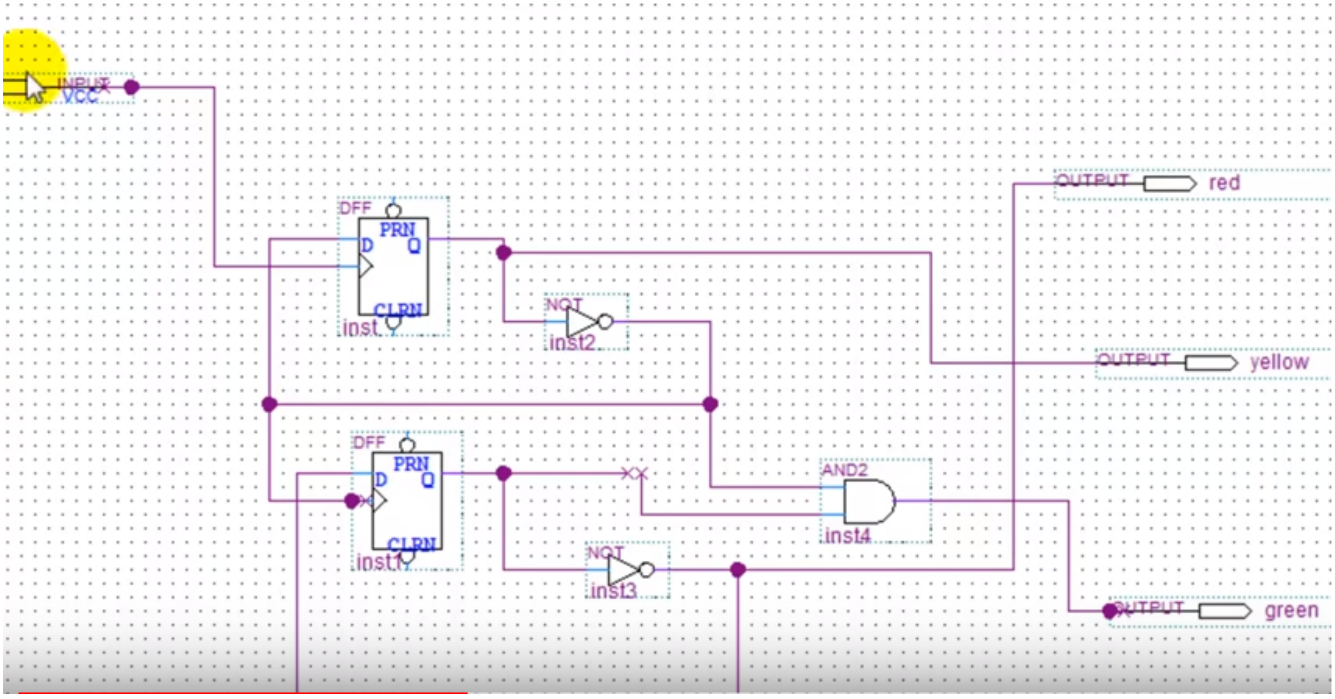


Figure 12

The input can either be provided through a push button or through a clock signal. On each clock pulse it changes one state and gives us the required red, yellow and green output. The above circuit represents one traffic light. This can be implemented in an FPGA.

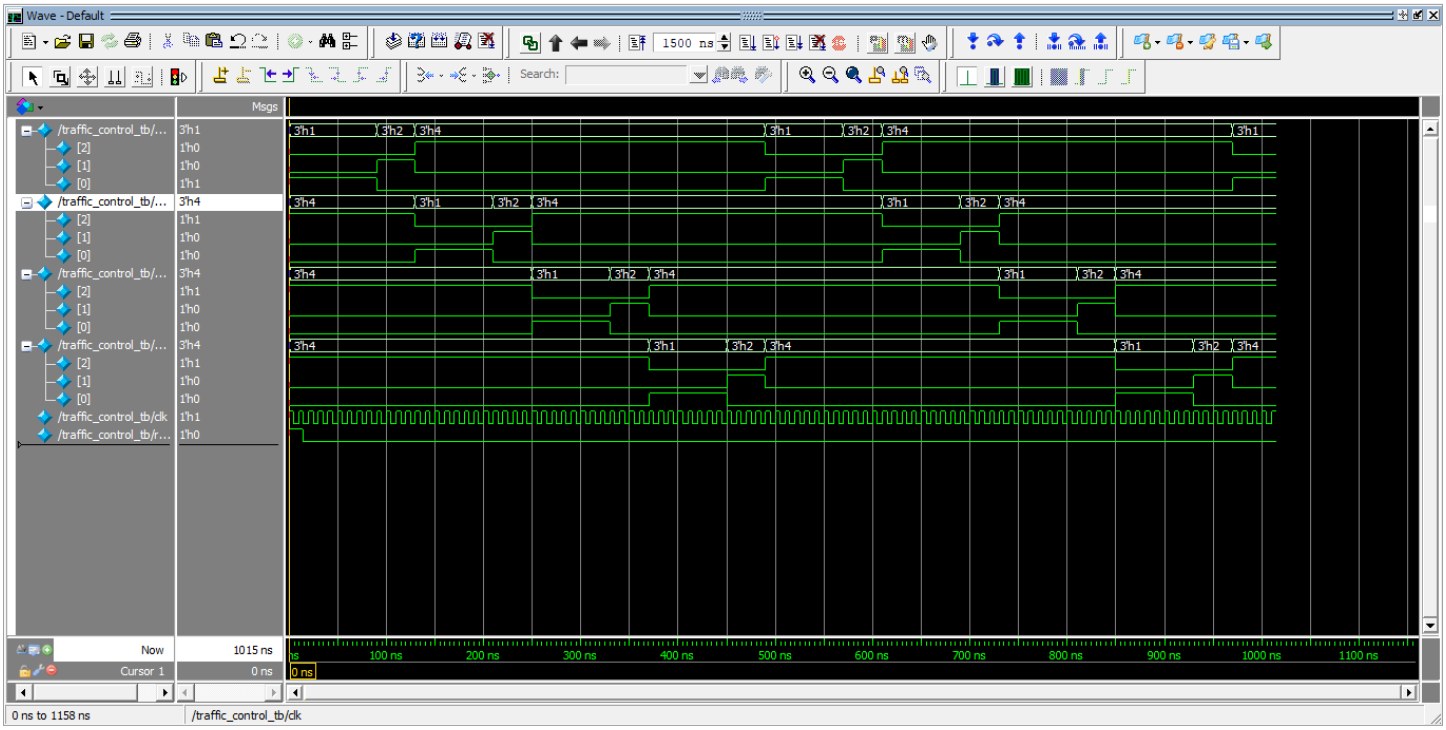
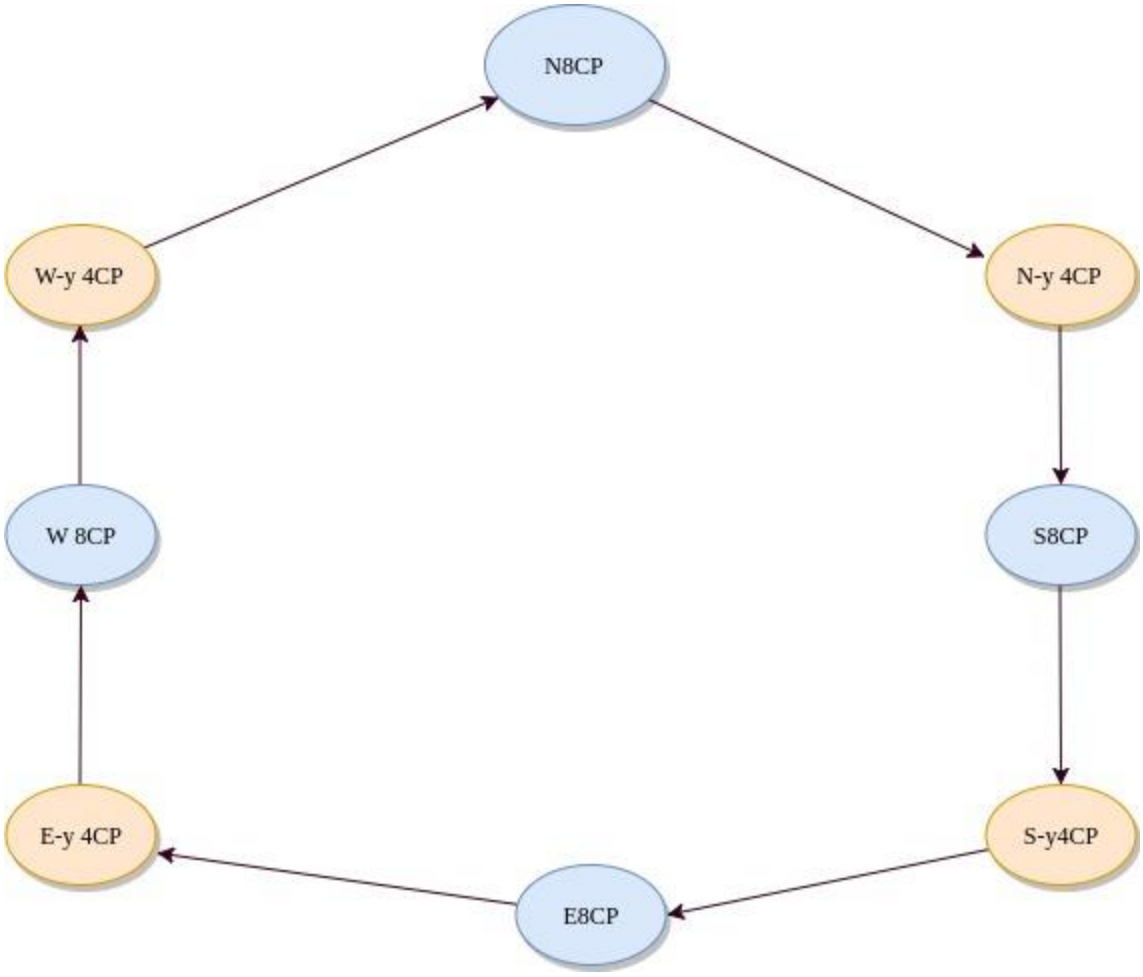


Figure 13: Screenshot of the simulation of traffic control system in ModelSim Student Edition



State Diagram of Traffic Control System

5.Appendices

5.1 Code for Stepper Motor in Verilog

```
`timescale 1ns / 1ps
module st(start,clock,dout);
	input clock;
	wire clock;
	reg [19:0]count;
	reg clock1;
	always@(posedge clock)  begin //Dividing the clock frequency
		count<=count+1;
		if(count==1000000) begin
			count<=0;
			clock1<=!clock1;
		end
	end
	output[3:0]dout;
	reg[3:0]dout;
	input start;
	wire start;
	reg [1:0]m;
	initial m=0;
	always@(posedge (clock1)) begin
		if(start)
			m<=m+1;
	end
	always@(m) begin
	case(m)
		0:dout=8;
		1:dout=4;
		2:dout=2;
		default:dout=1;
	endcase
	end
endmodule
```

5.2 UCF(User Constraint File) for Stepper Motor

```
NET "clock" LOC=V10;
NET "dout<0>" LOC=B6;
NET "dout<1>" LOC=D11;
NET "dout<2>" LOC=H15;
NET "dout<3>" LOC=F13;
```

5.3 Code for Traffic Light System in Verilog

```
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'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



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);

initial
 begin
  clk=1'b1;
  forever #5 clk=~clk;
 end

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

6.Bibliography

6.1 Websites:

a. ***https://en.wikipedia.org/wiki/Stepper_motor***

*Access date:*03/12/2017

b.*www.pantechsolution.net*

Access date:05/12/2017

c. https://www.youtube.com/watch?v=G6pylvT40Ko

Access date:05/12/2017

d. Controlling a Stepper Motor with an FPGA- Digilent Blog, BY KAITLYN FRANZ (FEB 3, 2016).

Access date: 23/10/2017

6.2 Books:

a. FPGA PROTOTYPING BY VERILOG EXAMPLES

Version: XILINX SPARTAN -3

Publication: A JOHN WILEY & SONS, INC.

Author: Pong P. Chu (Cleveland State University)

b. ALS-SDA-VHDL TRAINER LAB MANUAL

By: ALS(ADVANCED ELECTRONICS SYSTEMS)

c. Verilog HDL: A Guide to Digital Design and Synthesis

Edition:2nd Ed.

Author: S Palnitkar (Feb 21,2003).

6.3 Applications Used:

a. ModelSim Student Edition *for simulation*

b. Xilinx XCS350 Package *for synthesis*