Design with PLDs and FPGAs Laboratory(11EC312L)

**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**

S.Najia tasleem (Reg. No.12004338)

P.Manasvi (Reg. No. 12004339)

D.Teja sri (Reg. No. 12004341)

R.Monica (Reg. No. 12004343)

**Lab Instructor**

**Dr.Fazal NoorBasha**



Department of ECE,

KL University, Vaddeswaram, Guntur, AP – 522502

Academic Year : 2013-2014

Department of Electronics and Communication Engineering

KL University, Vaddeswaram, Guntur, AP – 522502



**C E R T I F I C A T E**

This is to certify that the mini project entitled **"DESIGN AND IMPLEMENTATION OF TRAFFIC LIGHTS CONTROLLER USING FPGA"** is for the project based laboratory for the subject “Design with PLDs & FPGAs” done by S.NAJIA TASLEM , P.MANASVI, D.TEJA SRI, R.MONICA bearing registration number’s R12004338,4339,4341,4343 is a student of III/IV B.Tech I-Semester during the academic year 2013-2014in partial fulfilment for the award of III/IV B.Tech I-Semester of Department of Electronics and Communication Engineering, K L University, Vaddeswaram, Guntur, Andhra Pradesh, INDIA.

Signature of the HOD-ECE Signature of the Lab Faculty

**DECLARATION**

We declare that the project entitled, “**DESIGN AND IMPLEMENTATION OF TRAFFIC LIGHTS CONTROLLER USING FPGA**” is our own work conducted under the esteemed guidance of Dr. Fazal Noorbasha (*Section Instructor*), Dr. Harikishore K, Mr.B.Kali Vara Prasad and Ms. M Preeti, at the Department of Electronics and Communication Engineering of KL University, Vaddeswaram, Guntur, A.P., INDIA.

S.Najia tasleem (Reg. No.12004338)

P.Manasvi (Reg. No. 12004339)

D.Teja sri (Reg. No. 12004341)

R.Monica (Reg. No. 12004343)

**ACKNOWLEDGEMENT**

Working on this Project based Lab is certainly a memorable and enjoyable event in our life. We have learned a lot of interesting new things that have broadened our view of the technology field. In here, we would like to offer our appreciation and thanks to several grateful and helpful individuals. Without them, this work could not have been completed and the experience would not be so enjoyable.

We are very grateful to our esteemed Professors **Dr. Fazal Noorbasha** (Section Instructor), Dr. K Harikishore, Mr.B.Kali Vara Prasad and Ms. M Preeti, Department of ECE, KL University, Vaddeswaram, Guntur, A.P., INDIA, for their valuable guidance and creative suggestions that helped us to complete this Lab project.

Furthermore, we are also very thankful **Dr. Fazal Noorbasha,** Microelectronics Group Head,to have an opportunity to learn from him on the aspect of using the advancing FPGA technology to improve the performance for different memory and processor applications. Hopefully, this experience will inspire us to come up with new and interesting research ideas in the future.

Furthermore, we want to offer our thanks to **Dr. ASCS Sastry**, Professor & Head, Department ECE, KL University, Vaddeswaram, Guntur, A.P., INDIA, for providing the required facilities to carry out the project work successfully.

We would like to thank all those helped us directly or indirectly during this project work.

S.Najia tasleem (Reg. No.12004338)

P.Manasvi (Reg. No. 12004339)

D.Teja sri (Reg. No. 12004341)

R.Monica (Reg. No. 12004343)

**ABSTRACT**

The traffic in road crossings /junctions is controlled by switching ON/OFF Red, Green & Amber lights in a particular sequence. The Traffic Light Controller is designed to generate a sequence of digital data called switching sequences that can be used to control the traffic lights of a typical four roads junction in a fixed sequence. It is also proposed to implement the day mode and night mode operations. It plays more and more important role in modern management and control of urban traffic to reduce the accident and traffic jam in road. It is a sequential machine to be analyzed and programmed through a multistep process. The device that involves an analysis of existing sequential machines in traffic lights controllers, timing and synchronization and introduction of operation and flashing light synthesis sequence. The methods that are used in this project are design the circuit, write a coding, simulation, synthesis and implement in hardware. In this project, XILINX Software was chosen to design a schematic using schematic edit, writes a coding using Verilog HDL (Hardware Description Language) text editor and implements the circuit on Programmable Logic Device [PLD].

LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Fig. No. | Title | Page No |
| 1.1 | FPGA Design Flow | 3 |
| 1.2 | FPGA Synthesis | 4 |
| 1.3 | FPGA Translate | 5 |
| 1.4 | FPGA Map | 6 |
| 1.5 | FPGA Place and route | 6 |
| 2.1 | TLC Flow Chart | 8 |
| 2.2 | Traffic Signals at Junction | 9 |
| 2.3 | TLC State Diagram | 11 |
| 3.1 | RTL Schematic | 13 |
| 3.2 | Technology Schematic | 14 |
| 3.3 | Wave Form | 15 |
| 4.1 | Structure of Road | 16 |
| 4.2 | Spartan-3E trainer kit | 17 |
| 4.3 | FPGA Implementation of TLC | 18 |

CONTENTS

|  |  |
| --- | --- |
| **Certificate** | i |
| **Declaration** | ii |
| **Acknowledgements** | iii |
| **Abstract** | iv |
| **List of Figures** | V |
| **List of Tables** | Vi |
| **Contents** | Vii |
|  |  |
| **Chapter – 1** |  |
| **Introduction** | 1-2 |
| **1.1 Traffic lights** | 1 |
| **1.2 Electronic design with FPGA’S** | 2 |
| **1.3 FPGA design flow** | 3 |
|  |  |
|  |  |
| **Chapter – 2** |  |
| **System Block Diagram and Working Principle** | 3-4 |
| **2.1 System block diagram** | 3 |
| **2.2 Working Principle** | 4 |
|  |  |
| **Chapter – 3** | 5-6 |
| **System Modeling** | 5 |
|  |  |
| **Chapter – 4** |  |
| **Implementation results** | 7-8 |
| **4.1 System Analysis** | 7 |
| **4.2 Results Report** | 8 |
|  |  |
| **Chapter 5** |  |
| **Conclusions and Applications** | 9-10 |
| **5.1 Conclusion** | 9 |
| **5.2 Applications** | 10 |
|  |  |
| **Appendix** | 11-13 |
| **Appendix – A: Verilog Code and Test Bench** | 11 |
| **Appendix – B: References** | 13 |

**Chapter – 1**

**INTRODUCTION**

**1.1 Need for Traffic Light Controller**

Traffic congestion is a severe problem in many modern cities around the world. Traffic congestion has been causing many critical problems and challenges in the major and most populated cities. To travel to different places within the city is becoming more difficult for the travelers in traffic. Due to these congestion problems, people lose time, miss opportunities, and get frustrated. Traffic congestion directly impacts the companies. Due to traffic congestions there is a loss in productivity from workers, trade opportunities are lost, delivery gets delayed, and thereby the costs goes on increasing. To solve these congestion problems, we have to build new facilities & infrastructure but at the same time make it smart. The only disadvantage of making new roads on facilities is that it makes the surroundings more congested. So for that reason we need to change the system rather than making new infrastructure twice. Therefore many countries are working to manage their existing transportation systems to improve mobility, safety and traffic flows in order to reduce the demand of vehicle use. Therefore, many researches about traffic light system have been done in order to overcome some complicated traffic phenomenon but existent research had been limited about present traffic system in well-travelled traffic scenarios. The time of allocation is fixed from east to west or opposite way and from north to south way in crossroads. Field Programmable Gate Arrays (FPGAs) are extensively used in rapid prototyping and verification of a conceptual design and also used in electronic systems when the mask-production of a custom IC becomes prohibitively expensive due to the small quantity. Many system designs that used to be built in custom silicon VLSI are now implemented in Field Programmable Gate Arrays. This is because of the high cost of building a mask production of a custom VLSI especially for small quantity.

**1.2 Introduction to FPGA**

In most digital designs, the circuitry can be classified by the following categories:

**Standard products**

These products provide a functionality which is not associated with a specific application area but common to a broad range of devices. Typical parts in this category are processors and memories.

**Application Specific Standard Products or ASSPs**

These products provide functionality which is not associated with a specific implementation, but common to an application area. Typical parts in this category are MPEG decoders.

**Custom Logic**

This logic is associated with a specific application and is the essence of what distincts one product from another. Often this is glue logic, connecting standard products or ASSPs with each other.

There are several options on how to implement custom logic, FPGAs being one amongst them.

**FPGAs [Field Programmable Gate Array].**

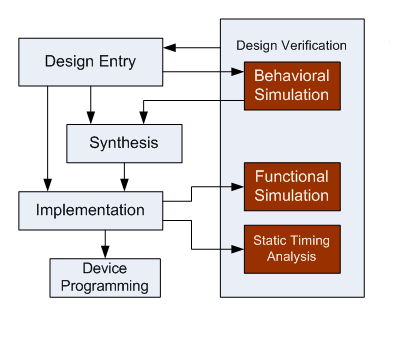
FPGA is the abbreviation of Field Programmable Gate Array. This denotes an integrated circuit which is programmed in the field, i.e. by the system manufacturer. FPGAs can be characterized by the following items:

* High production cost
* Low design density
  + Programmable fabric adds significant overhead
* No NRE and Re-Spin cost
* Low development effort
  + Low dead-time
  + simplified timing
  + No test vectors
  + Relaxed verification
  + Physical design is “hands-off”

FPGA is an Integrated Circuit consisting of an array of uncommitted elements; interconnection between these elements is user-programmable. Using Random Access Memory, high density logic is provided. 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. The Design flow of FPGA shown in Fig. 1 is used to implement the traffic light controller using FPGA. The circuit description can be done using HDLs, followed by the functional simulation and synthesis. The design flow is followed till the timing simulation and then the generated file is downloaded into the target device (FPGA). Verilog is used as HDL for circuit description to code the TLC module. Verilog HDL is used because of the difficulty in writing a VHDL code which has to integrate the source code, ChipScope Pro-Integrated Controller (ICON) and Virtual Input Output (VIO).

**1.3 FPGA Design Flow**

A simplified version of design flow is given in the flowing diagram.



**Fig 1.1 FPGA Design Flow**

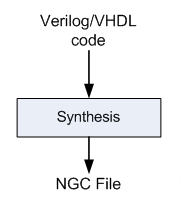
**Design Entry**

There are different 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 better 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 isolate the designers from the details of the hardware implementation.  Schematic based entry gives designers much more visibility into the hardware. It is the better choice for those who are hardware oriented. 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.

**Synthesis**

The process which translates VHDL or Verilog code into a device netlist format .i.e a complete circuit with logical elements( gates, flip flops, etc…) for the design.If the design contains more than one sub designs, ex. to implement  a processor, we need a CPU as one design element and RAM as another and so on, then the synthesis process generates netlist for each design element  
Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC( Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).



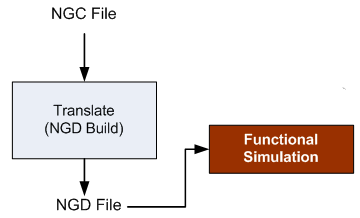
**Fig 1.2 FPGA Synthesis**

**Implementation**

This process consists a sequence of three steps  
1. Translate  
2. Map  
3. Place and Route

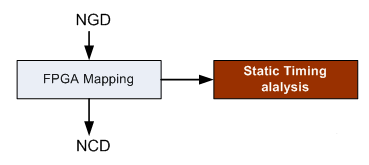
**Translate** process combines all the input netlists and constraints to a logic design file. This information is saved as a NGD (Native Generic Database) file. This can be done using NGD Build program. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File).

Tools used to create or modify the UCF are PACE, Constraint Editor etc.



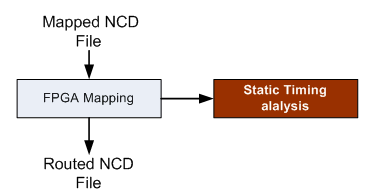
**Fig 1.3 FPGA Translate**

**Map** process divides the whole circuit with logical elements into sub blocks such that they can be fit into the FPGA logic blocks. That means map process fits the logic defined by the NGD file into the targeted FPGA elements (Combinational Logic Blocks (CLB), Input Output Blocks (IOB)) and generates an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA. MAP program is used for this purpose.



**Fig 1.4 FPGA map**

**Place and Route** PAR program is used for this process. The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. Ex. if a sub block is placed in a logic block which is very near to IO pin, then it may save the time but it may affect some other constraint. So trade off between all the constraints is taken account by the place and route process. The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consists the routing information.



**Fig 1.5 FPGA Place and route**

**Device Programming**

Now the design must be loaded on the FPGA. But the design must be converted to a format so that the FPGA can accept it. BITGEN program deals with the conversion. The routed NCD file is then given to the BITGEN program to generate a bit stream (a .BIT file) which can be used to configure the target FPGA device. This can be done using a cable. Selection of cable depends on the design.

**Design Verification**

Verification can be done at different stages of the process steps.

**Behavioural Simulation** (RTL Simulation) This is first of all simulation steps; those are encountered throughout the hierarchy of the design flow. This simulation is performed before synthesis process to verify RTL (behavioural) code and to confirm that the design is functioning as intended. Behavioural simulation can be performed on either VHDL or Verilog designs. In this process, signals and variables are observed, procedures and functions are traced and breakpoints are set. This is a very fast simulation and so allows the designer to change the HDL code if the required functionality is not met with in a short time period. Since the design is not yet synthesized to gate level, timing and resource usage properties are still unknown.

**Functional simulation** (Post Translate Simulation) Functional simulation gives information about the logic operation of the circuit. Designer can verify the functionality of the design using this process after the Translate process. If the functionality is not as expected, then the designer has to made changes in the code and again follow the design flow steps.

**Static Timing Analysis** This can be done after MAP or PAR processes Post MAP timing report lists signal path delays of the design derived from the design logic. Post Place and Route timing report incorporates timing delay information to provide a comprehensive timing summary of the design.

**Chapter – 2**

**Design of Traffic Light Controller**

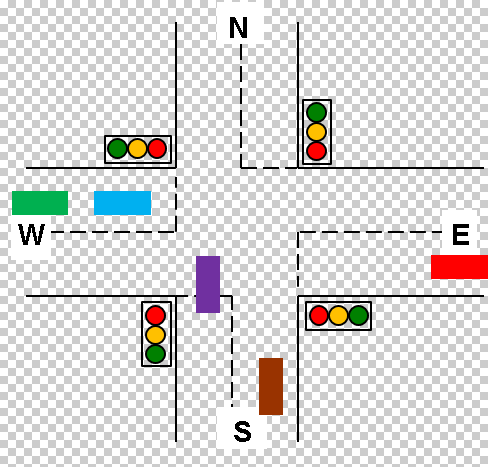
Traffic Light Controller can be designed by starting with some arbitrary assumptions. At first the North traffic will be allowed to move and then traffic in the East, South and West direction will be allowed to move in sequence. The advantage of writing Traffic Light Controller program is that in a program, modifications as per requirements can be done easily i.e., suppose the traffic on main road should be allowed for more time and for side roads the traffic should be allowed for less time; then the clock is divided in such a way that for main road the clock period will be more and for side roads the clock period will be less, this is because the main road traffic is heavy when compared to the side road traffic. In general TLC System will be having three lights (red, green and yellow) in each direction where red light stands for traffic to be stopped, green light stands for traffic to be allowed and yellow light stands for traffic is going to be stopped in few seconds.



**Fig 2.1 TLC Flow Chart**

**2.1 Explanation of Traffic Light Controller**

In this structure, there are four traffic signals, represented by R1, R2, R3 and R4 to be controlled. All the four signals have same priority as they all are main roads.

****

**Fig 2.2 Traffic Signals at Junction**

First of all the signal controller is in the reset mode where in the signal of road (R1) is green whereas all the other roads R2, R3 and R4 are red. This state we have assigned as S0.

 Later the controller sends the control to state S1 where the R1 is yellow whereas all the other signals are still red only. In this state the controller checks whether the sensor at road R2 which is X2 is low or not. If the sensor gives a low signalling that there is no traffic on that road, then that signal on road R2 is skipped transferring control to the state S4 where signal on road R3 is turned whereas rest of the signals are showing red. On the hand if the traffic is present on the road R2 then the control is sent to state S2 which switches on the signal on road R2 to green and rest of the signals are red only when the control is with state S2 after showing the green signal the signal light changes from green to yellow for signal on the road R2 while all the other signals continue to be in red light mode only which is the operation of state S3.

Again when the controller is in state S3 it checks for the response of sensor X3 on road R3.  If the output of sensor is low the control of the system will be transferred to state S6 skipping the working of the signal on road R3 otherwise the control is given to corresponding next state S4.

When in S4 the traffic signal of road R3 turns green on the other hand the signals of roads R1, R2 and R4 remain red itself. The control is then transferred to state S5.

When the control is with state S5 it checks for the output of the sensor X4 on the road R4. Depending on the output of X4 the further state change takes place accordingly. If low then the control is transferred to state S0 skipping the operation of the signal on road R4 otherwise the control is with the S6. When the controller is in state S5 there is change of signal on road R3 from green to yellow.

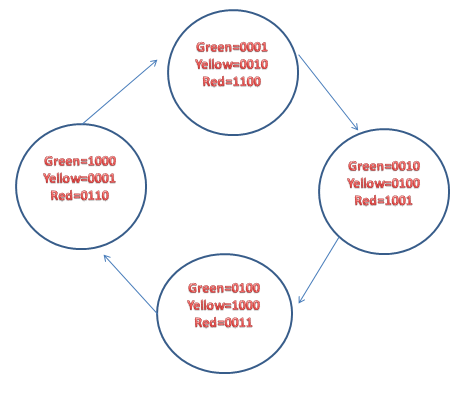
When the control is with state S6 the signal of road R4 turns green whereas  all the signal turn or remain in red signal only. Thee control is then shifted to state S0.

In state S7 the signal of road R4 turns from green to yellow. Simultaneously the sensor on the first road R1 which is X1 is checked for its output. If the signal is low then the control is shifted directly to state S2 otherwise the control is shifted to default state S0.

These states are not mandatory. The number of states, the order of the lights and the delay can be specified by the user. This is one of the most advantages in this project.

**2.2 TLC State Diagram**

The TLC state diagram shown in Fig 2.3 illustrates that whenever cnt=00 and dir=00,then green light in north direction will be ON for few seconds and red signal light in all other directions namely west, south and east will be ON. When cnt=01 and dir=00 then yellow light (y1) will be ON for few seconds and when cnt=01 yellow light (y2) and pedestrian north will be ON and then dir is incremented by one and cnt is assigned to zero. So when cnt=00 and dir=01, the green light in east direction will be ON for few seconds and all red lights in other directions be ON. Whenever cnt=01 and dir=01 then yellow light (y1) will be ON for few seconds and when cnt=01 yellow light (y2) and pedestrian east will be ON and then dir is incremented by one and cnt is assigned to zero.



**Fig 2.3 TLC State Diagram**

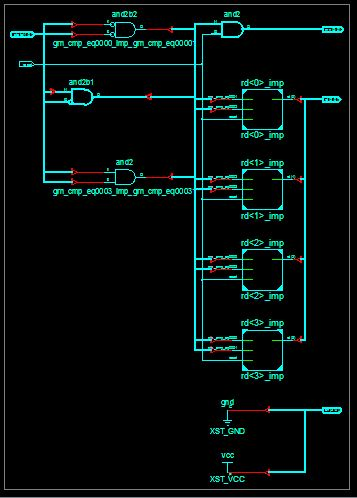
So whenever cnt=00 and dir=10, the green light in south direction will be ON for few seconds and all red lights in other directions will be ON. Whenever cnt=01 and dir=10 then yellow light (y1) will be ON for few seconds and when cnt=01 yellow light (y2) and pedestrian south will be ON and then dir is incremented by one and cnt is assigned to zero. So whenever cnt=00 and dir=11, the green light in west direction will be ON for few seconds and all red lights in other directions will be ON. Whenever cnt=01 and dir=11 then yellow light (y1) will be ON for few seconds and when cnt=01 yellow light (y2) and pedestrian west will be ON and then dir is assigned to 00 and cnt is assigned to zero. This sequence repeats and the traffic flow will be controlled by assigning time periods in all the four directions.

**Chapter – 3**

**Simulation Results**

**3.1 RTL Schematic**

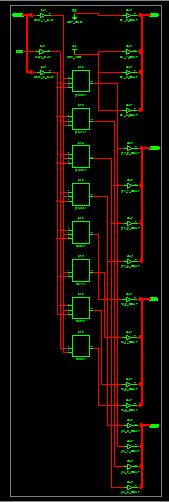
The below figure shows the RTL Schematic of the Traffic Light Controller.



**Fig 3.1 RTL Schematic**

**3.2 Technology Schematic**

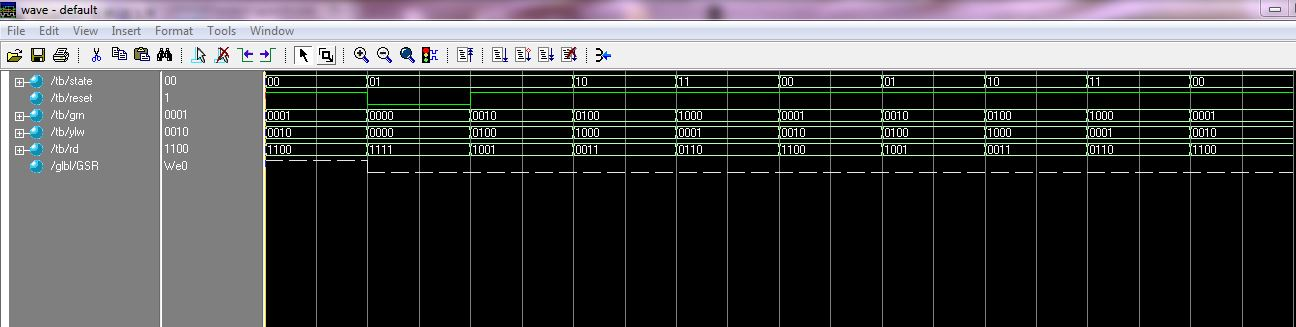
The below figure shows the Technology Schematic of the Traffic Light Controller.



**Fig 3.2 Technology Schematic**

**3.3 Wave Form**

The below figure shows the Wave form of the Traffic Light Controller when the test bench is applied to the source code.



**Fig 3.3 Wave Form**

**Chapter – 4**

**Hardware Implementation**

Fig. 4.1 depicts a general four road structure which consists of north, east, west and south directions each with a set of three lights namely green, yellow and red. Green light in a direction will be ON when left, straight and right side is set to be free for traffic in that direction.



**Fig 4.1 Structure of Road**

The figure shows the design of traffic light model. To distinguish each lane and the traffic signal lights, they are labeled separately with North, East, South and West. Signal lights at each lane have their set of traffic light signal “Red, Yellow, and Green”. Operation of this signal light is similar to common traffic light signal. Along with these specifications, each lane has a light to represent a sensor of the corresponding road. Linear sensor or electromagnetic sensor is suitable for design of a real traffic light system. The first sensor detects the presence of vehicles and the second sensor determines the volume of the traffic corresponding to that lane. Through the two sensors, we will know the expected time for green signal ON and when the signal light at each lane should be changed to green.

The state machine is coded using the Hardware Description Language, Verilog. Spartan-3E trainer kit is shown in Fig. 4.2.



**Fig 4.2 Spartan-3E trainer kit**



**Fig 4.3 FPGA Implementation of TLC**

Fig. 4.3 shows the FPGA Implementation of TLC. Using Xilinx ISE tool, this code is dumped into Spartan-3E FPGA trainer kit and the outputs here we considered are more than the LEDs on the FPGA.

**Chapter – 5**

**Implementation Results**

The Traffic Light Controller was designed using Verilog HDL and was implemented using FPGA. The output of the TLC is verified with NEXYS 2 FPGA. Fig 5.1 shows the following were the results taken by us after the implementation.

****

**Fig 5.1 Display output for the symbol Red**

**Chapter – 6**

**Conclusion and Future Scope**

The modern ways of multi-way traffic management improves the traffic condition up to a large extent. Advanced signaling controllers contribute to the improvement of the urban traffic; which is proportional to the complexity of the controller. These more complex controllers can be well handled using states machines. Methods to reduce the states in the state machine also help in reducing the required hardware thus leading to low power and area efficient design.

The future scope of this project is it can be directly applied in real time by employing more number of such circuits.

**Bibliography**

Parag K. Lala, “Digital System Design Programmable Logic Devices”, Page15-17, Chapter 1, B S Publications

**Appendix: Source Code and Test Bench**

**Source Code:**

module prjctr(state,reset,grn,ylw,rd,an);

input[1:0] state;

input reset;

output reg [3:0]grn,ylw,rd;

output reg [3:0]an;

always@(reset or state)

begin

if(reset==0)

begin

grn=4'b0000;

//ylw=4'b0000;

ylw=4'b1111;

rd=4'b1111;

end

else

case (state)

2'b00:

begin

grn=4'b0001;

//ylw=4'b0010;

ylw=4'b1101;

rd=4'b1100;

end

2'b01:

begin

grn=4'b0010;

//ylw=4'b0100;

ylw=4'b1011;

rd=4'b1001;

end

2'b10:

begin

grn=4'b0100;

//ylw=4'b1000;

ylw=4'b0111;

rd=4'b0011;

end

default:

begin

grn=4'b1000;

//ylw=4'b0001;

ylw=4'b1110;

rd=4'b0110;

end

endcase

an=4'b1110;

end

endmodule

**Test bench :**

module tb;

// Inputs

reg [1:0] state;

reg reset;

// Outputs

wire [4:1] grn;

wire [4:1] ylw;

wire [4:1] rd;

// Instantiate the Unit Under Test (UUT)

prjctr uut (

.state(state),

.reset(reset),

.grn(grn),

.ylw(ylw),

.rd(rd)

);

initial begin

// Initialize Inputs

state = 2'b00;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b01;

reset = 0;

// Wait 100 ns for global reset to finish

#100;

state = 2'b01;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b10;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b11;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b00;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b01;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b10;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b11;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

state = 2'b00;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

end

endmodule

UCF FILE:

net "reset" loc=p11;

net "state[0]" loc=l3;

net "state[1]" loc=k3;

net "grn[0]" loc=m5;

net "grn[1]" loc=m11;

net "grn[2]" loc=p7;

net "grn[3]" loc=p6;

net "rd[0]" loc=n5;

net "rd[1]" loc=n4;

net "rd[2]" loc=p4;

net "rd[3]" loc=g1;

net "an[0]" loc=f12;

net "an[1]" loc=m13;

net "an[2]" loc=j12;

net "an[3]" loc=k14;

net "ylw[0]" loc=l13;

net "ylw[1]" loc=l14;

net "ylw[2]" loc=m12;

net "ylw[3]" loc=p12;