**FPGA BASED TRAFFIC CONTROLLING SYSTEM**

**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.

Traffic light controller is a set of rules and instructions that drivers, pilots, train engineer, and ship captains rely on to avoid collisions and other hazards. Traffic control systems include signs, lights and other devices that communicate specific directions, warnings, or requirements. Traffic light controller (TLC) has been implemented using verilog HDL. It has many advantages over other with reference to the speed, number of input/output ports and performance which are all very important in design. This paper concerns with an design implementation of an advanced traffic light controller system that was built as a term project of a VLSI design subject using VHDL. The system has been successfully tested and implemented in hardware using Xilinx Spartan 3 FPGA. The system has many advantages over the other exciting Traffic Light Controller. The VHDL code is being used in order to implement the design and the simulation is being tested using the Isim Simulator.

Keywords: Xilinx, Traffic controlling System, Verilog, VHDL , FPGA, Timing , collisions.

**INTRODUCTION**

* 1. **TRAFFIC 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.

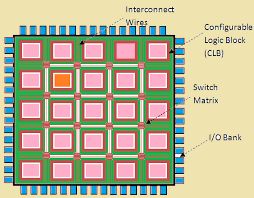
The time of allocation is ﬁxed from east to west or opposite way and from north to south way in crossroads. Field Programmable Gate Arrays (FPGAs) are extensively used in rapid proto typing and veriﬁcation of a conceptual design and also used in electronic systems when the mask-production of a custom 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.

FPGA based traffic controller for five intersecting main roads with minimum resource usage. It is interesting to note that the FPGA can have up to 64 logic states (i.e. 64 individual output ports), and his design requires only 25 logic states leaving enough room for other operations.

* 1. Introduction to FPGA:

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), similar to that used for an application-specific integrated circuit (ASIC).

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. It is used in broad range of applications.

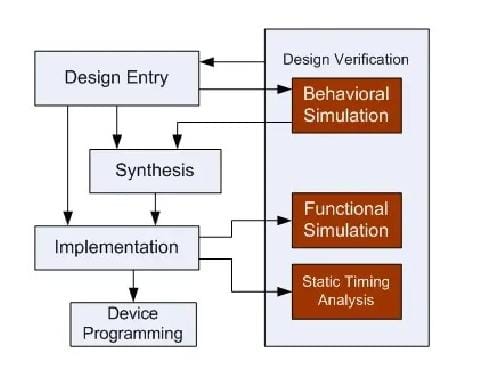


FPGA is the abbreviation of Field Programmable Gate Array. This denotes an integrated circuit which is programmed in the ﬁeld, i.e. by the system manufacturer. FPGAs can be characterized by the following items:

* 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
* Relaxed veriﬁcation
* Physical design is “hands-oﬀ

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).

**FPGA design flow**

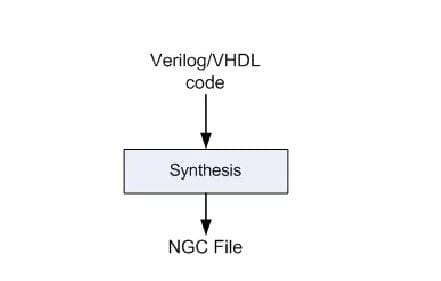


**Design Entry:**

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 density. HDLs represent a level of abstraction that can isolate the designers from thedetails of the hardware implementation. Schematic based entry gives designers much more visibility into the hardware. It is the better choice forthose 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 aseries 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)).



**IMPLEMENTATION:**

`timescale 1ns / 1ps

module Traffic\_Light\_Controller(

input clk,rst,

output reg [2:0]light\_M1,

output reg [2:0]light\_S,

output reg [2:0]light\_MT,

output reg [2:0]light\_M2

);

parameter S1=0, S2=1, S3 =2, S4=3, S5=4,S6=5;

reg [3:0]count;

reg[2:0] ps;

parameter sec7=7,sec5=5,sec2=2,sec3=3;

always@(posedge clk or posedge rst)

begin

if(rst==1)

begin

ps<=S1;

count<=0;

end

else

case(ps)

S1: if(count<sec7)

begin

ps<=S1;

count<=count+1;

end

else

begin

ps<=S2;

count<=0;

end

S2: if(count<sec2)

begin

ps<=S2;

count<=count+1;

end

else

begin

ps<=S3;

count<=0;

end

S3: if(count<sec5)

begin

ps<=S3;

count<=count+1;

end

else

begin

ps<=S4;

count<=0;

end

S4:if(count<sec2)

begin

ps<=S4;

count<=count+1;

end

else

begin

ps<=S5;

count<=0;

end

S5:if(count<sec3)

begin

ps<=S5;

count<=count+1;

end

else

begin

ps<=S6;

count<=0;

end

S6:if(count<sec2)

begin

ps<=S6;

count<=count+1;

end

else

begin

ps<=S1;

count<=0;

end

default: ps<=S1;

endcase

end

always@(ps)

begin

case(ps)

S1:

begin

light\_M1<=3'b001;

light\_M2<=3'b001;

light\_MT<=3'b100;

light\_S<=3'b100;

end

S2:

begin

light\_M1<=3'b001;

light\_M2<=3'b010;

light\_MT<=3'b100;

light\_S<=3'b100;

end

S3:

begin

light\_M1<=3'b001;

light\_M2<=3'b100;

light\_MT<=3'b001;

light\_S<=3'b100;

end

S4:

begin

light\_M1<=3'b010;

light\_M2<=3'b100;

light\_MT<=3'b010;

light\_S<=3'b100;

end

S5:

begin

light\_M1<=3'b100;

light\_M2<=3'b100;

light\_MT<=3'b100;

light\_S<=3'b001;

end

S6:

begin

light\_M1<=3'b100;

light\_M2<=3'b100;

light\_MT<=3'b100;

light\_S<=3'b010;

end

default:

begin

light\_M1<=3'b000;

light\_M2<=3'b000;

light\_MT<=3'b000;

light\_S<=3'b000;

end

endcase

end

endmodule.

This process consists a sequence of three steps

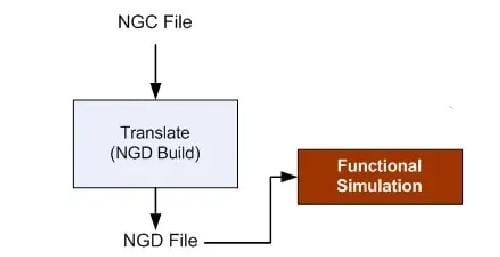
1. Translate

2. Map

3. Place and Route

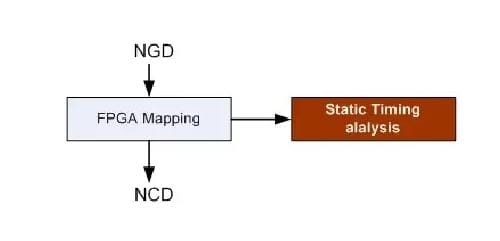
**Translate**

process combines all the input netlists and constraints to a logicdesign 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 timerequirements 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 et



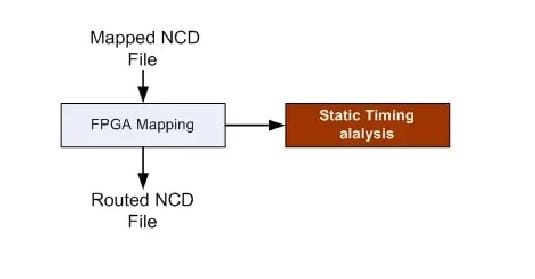
**MAP**

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**.**



**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.



**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

**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.

**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.

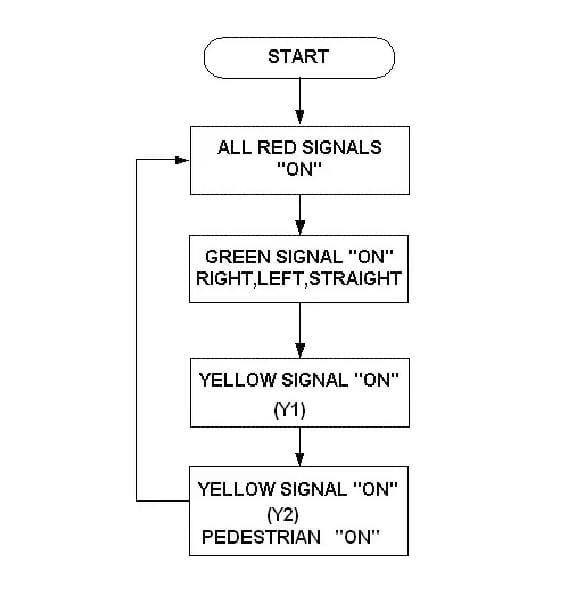
**Static timing Analysis:**

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**

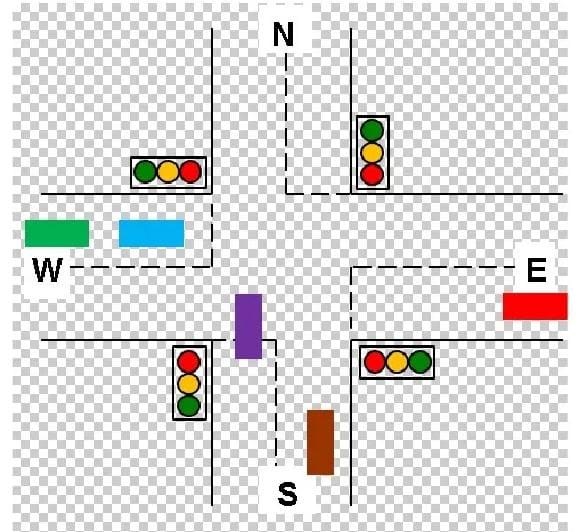
**DESIGN OF TRAFFIC CONTROLLER**

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.



**EXPLANATION OF TRAFFIC CONTROLLING SYSTEM**

In this structure, there are four traﬃc signals, represented by R1, R2, R3and R4 to be controlled. All the four signals have same priority as they all are main roads.



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 roadR3 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 X4on 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 S0skipping 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 roadR3 from green to yellow.

When the control is with state S6 the signal of road R4 turns green where as all the signal turn or remain in red signal only. The 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.

3.**SOURCE CODE :TEST BENCH**

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

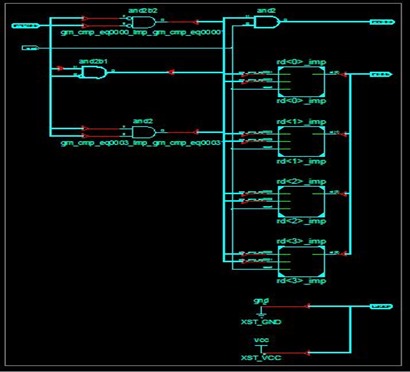
#100;

end

endmodule

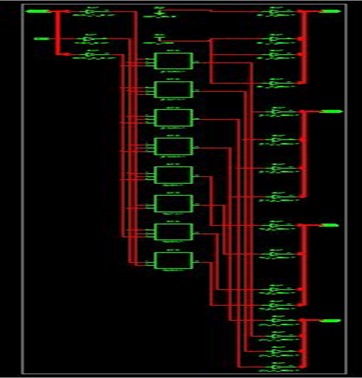
**CHAPTER-3 SIMULATION RESULTS**

**3.1 The below figure shows the RTL Schematic of the Traffic Light Controller**



**3.2 Technology Schematic**

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



**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.**

