A SUMMER INTERNSHIP REPORT

On

Electronic Circuit Analysis using PSPICE& Xilinx Vivado

(TRAFFIC LIGHT CONTROLLER USING VERILOG HDL)

Submitted for the partial fulfilment
of

B. Tech.

In

ELECTRONICS AND COMMUNICATION ENGINEERING

SUBMITTED BY:

Sanyam Agarwal
2100270310122

5th Semester - Third Year
Section- ECE-3

SUBMITTED TO:

Ms. Uma Sharma
(ASST. PROF.)
ECE DEPT.

Ajay Kumar Garg Engineering College, Ghaziabad
27th Km Milestone, Delhi-Meerut Expressway, P.O. Adhyatmik Nagar, Ghaziabad-201009

Dr. A. P. J. Abdul Kalam Technical University, Lucknow

NOVEMBER 2023

Acknowledgement

I want to express my sincere gratitude and thanks to **Prof. (Dr.) Neelesh Kumar Gupta (HoD, ECE Department), Ajay Kumar Garg Engineering College, Ghaziabad** for granting me permission for my industrial training in the field of **“Electronic Circuit Analysis using PSPICE & Xilinx Vivado”.**

I express my sincere thanks to **Dr. Pankaj Goel**, **Asst. Prof. Uma Sharma and Asst. Prof. Prashant Mani Tripathi** for his cooperative attitude and consistence guidance, due to which I was able to complete my training successfully.

Finally, I pay my thankful regard and gratitude to the faculty members and lab technicians of **Ajay Kumar Garg Engineering College, Ghaziabad** for their valuable help, support and guidance.

Sanyam Agarwal
2100270310122
5th Sem-Third Year
Section- ECE-3

TABLE OF CONTENTS

Chapter 1. Introduction to Verilog HDL & Xilinx Vivado Design Suite 4-12

1.1 Introduction

1.2 Keywords and Identifiers

1.3 Data Types and Modelling Styles

1.4 Verilog programming for various combinational and Sequential circuits

Chapter 2. Introduction to PSPICE 13-15

2.1 Introduction

2.2 Key features of PSPICE Simulator

2.3 Xilinx Vivado Design flow

Chapter 3. Mini Project Description 16-28

3.1 Introduction to TRAFFIC LIGHT CONTROLLER (Verilog/VHDL/PSPICE)

3.2 Key Features

3.3 Description of Project

3.3.1 Block Diagram/ Flow Chart

3.3.2 Algorithm Description

3.3.3 Simulation and Synthesis Results

Chapter 4. Conclusion and Future Scope 29-30

References 31

CHAPTER-1

INTRODUCTION TO VERILOG HDL

1.1 INTRODUCTION :

Verilog HDL (Hardware Description Language) is a hardware description language used to model electronic systems. It is primarily used in the design, modeling, and verification of digital circuits. Verilog HDL allows designers to describe the behavior and structure of electronic systems, such as integrated circuits (ICs) and computer hardware, using a textual representation.

Verilog is commonly used in the design and simulation of digital systems at various levels of abstraction, ranging from high-level system design down to gate-level modeling. It allows designers to specify the functionality, timing, and structure of digital circuits, making it an essential tool in the field of digital design and hardware engineering.

Designs written in Verilog describe how digital circuits should behave under different conditions, enabling simulation, verification, and synthesis processes that ultimately result in the creation of hardware components.

Verilog HDL is widely utilized in the design and verification of electronic systems and is supported by numerous electronic design automation (EDA) tools used by hardware designers to create and verify complex digital systems.

KEY FEATURES:

Verilog HDL (Hardware Description Language) is a powerful language used for describing and modeling digital systems. Some of its key features include:

1. **Conciseness and Simplicity:** Verilog allows concise descriptions of complex hardware behaviors and structures, making it relatively easy to learn and use.

2. **Behavioral Modeling:** It allows designers to specify the functionality or behavior of digital circuits, abstracting away from the underlying implementation details. This facilitates higher-level design and verification.

3**. Structural Modeling:** Verilog enables the description of the structure of digital systems, specifying the interconnection of various hardware components like gates, flip-flops, multiplexers, etc.

4**. Hierarchy:** Designs in Verilog can be organized hierarchically, allowing the creation of modules that can be instantiated multiple times within a design, promoting reusability and scalability.

5. **Simulation and Verification:** Verilog supports simulation tools that enable designers to test and verify the functionality of their designs before physical implementation. This helps in catching errors and ensuring correct behavior early in the design cycle.

6**. Synchronous and Asynchronous Modeling:** Verilog supports both synchronous (clock-based) and asynchronous modeling, allowing designers to capture various types of digital system behaviors accurately.

7**. Synthesizability:** Verilog designs can be synthesized into actual hardware using synthesis tools. This process transforms the high-level Verilog code into gate-level descriptions that can be implemented in hardware.

8. **Testbench Development:** Verilog allows the creation of testbenches, which are used to simulate and test the behavior of designs. Testbenches help verify the correctness of the design by generating stimulus and checking the responses.

9. **Support for System Verilog:** System Verilog, an extension of Verilog, adds several additional features such as object-oriented programming constructs, assertions, and more advanced verification capabilities.

10**. Industry Standard:** Verilog HDL is an industry-standard language widely used in the design and verification of digital systems. It's supported by various EDA (Electronic Design Automation) tools used by hardware engineers and designers.

These features collectively make Verilog HDL a powerful language for describing, simulating, synthesizing, and verifying digital hardware designs.

1.1.2 INTRODUCTION TO XILINX VIVADO DESIGN SUITE:

Vivado Design Suite is an integrated development environment (IDE) created by Xilinx, a leading provider of programmable logic devices (PLDs) like Field-Programmable Gate Arrays (FPGAs) and System on Chips (SoCs). Vivado offers a comprehensive set of tools for designing, verifying, synthesizing, implementing, and debugging digital systems targeted for Xilinx devices

KEY COMPONENTS:

Key components and features of the Vivado Design Suite include:

1. **Design Entry:** Vivado supports different methods for entering designs, including schematic entry, Hardware Description Languages (such as Verilog, VHDL), and higher-level abstractions like C/C++/SystemC using High-Level Synthesis (HLS).

2. **IP (Intellectual Property) Integrator:** It provides a platform for integrating pre-designed and pre-verified IP blocks into larger designs, facilitating faster development cycles by reusing established functionalities.

3. **High-Level Synthesis (HLS):** Vivado HLS enables the conversion of C, C++, and SystemC code into RTL (Register Transfer Level) code, accelerating the design process and allowing designers to work at higher levels of abstraction.

4. **Synthesis and Implementation:** The suite includes synthesis and implementation tools that convert high-level descriptions of the design into configurations that can be loaded onto Xilinx FPGAs or SoCs. This involves mapping the design onto the target device, optimizing for performance, and generating programming files.

5. **Verification and Simulation:** Vivado provides simulation capabilities for verifying the functionality of the designed system before synthesis and implementation. This helps catch errors and bugs early in the design process.

6. **Timing Analysis:** It includes tools for analyzing and ensuring that the designed circuits meet the required timing constraints, critical for proper functioning in real-world scenarios.

7. **Debugging and Validation:** Vivado offers features for debugging FPGA designs, such as interactive debug capabilities, which help in identifying and resolving issues during the development phase.

8. **Programming and Configuration:** Once the design is finalized and implemented, Vivado generates programming files (such as bitstreams) that configure the target Xilinx FPGA or SoC, enabling it to function according to the design specifications.

9. **Support for Various Xilinx Devices:** Vivado supports a wide range of Xilinx devices, including FPGAs, SoCs, and MPSoCs, allowing designers to create designs for different applications and performance requirements.

CONCLUSION: Verilog HDL and Xilinx Vivado Design Suite are integral components in the digital design ecosystem. Verilog provides a robust language for describing digital circuits, while Vivado offers a comprehensive suite of tools for designing and implementing FPGA- based systems efficiently. Together , they enable engineers to create complex digital designs with a balance of abstraction and control.

1.2 KEYWORDS AND IDENTIFIERS:

1.2.1.1 Keywords in Verilog HDL:

In Verilog HDL (Hardware Description Language), several keywords are used to define various constructs, behaviors, and structures within a digital design. Here are some essential keywords in Verilog.

Verilog HDL, a hardware description language, contains several essential keywords used to define various aspects of digital designs. These keywords encompass diverse functionalities within the language. "Module" is fundamental, defining a module that encapsulates hierarchical designs, while "Input" and "Output" keywords specify the ports of a module for interfacing with external components. "Wire" signifies the net connecting modules or elements, while "Reg" denotes a register capable of storing values, representing sequential logic elements. Keywords like "Always" and "Initial" commence code blocks, describing continuous or initial behaviors. "Assign" facilitates continuous value assignments to wires, and conditional statements such as "If/Else" or "Case" aid in decision-making within the design. Operators like '=' and '<=' assist in assigning values, with the latter specifically for sequential logic. "Parameter" helps declare constants throughout the design, while "Function" and "Task" create reusable procedures for specific operations. Additionally, "Delay" permits timing delays in simulation, replicating real-world circuit behavior. Familiarity with these keywords is crucial for accurately defining and modeling complex digital systems in Verilog HDL.

These keywords, among others, form the basis for creating designs in Verilog HDL, allowing hardware designers to describe and model complex digital systems efficiently. Understanding these keywords is essential for constructing accurate and functional hardware descriptions in Verilog.

1.2.1.2 Identifiers in Verilog HDL:

In Verilog HDL (Hardware Description Language), identifiers are names assigned to various elements within a hardware design to uniquely identify and reference them. These elements can include modules, variables, parameters, ports, registers, wires, tasks, and functions.Module Names: Names given to modules in Verilog, typically starting with a letter or an underscore.module my_module;Signal Names: Identifiers for wires and registers used to represent data flow. reg [7:0] data_in;wire [7:0] result;Instance Names: Names given to instances of modules when instantiated within another module.my_module instance1 (.input_A(a), .input_B(b), .output_Z(z));Variable Names: Identifiers for variables used within procedural blocks like always or initial. int count = 0;Port Names: Identifiers used for module ports in the module definition. module my_module (input a, output b); These identifiers play a crucial role in describing and organizing the hardware components within the Verilog design, ensuring readability, maintainability, and proper functionality of the hardware description.

1.2.2.1 Keywords in Xilinx Vivado Design Suite:

In the Xilinx Vivado Design Suite, keywords represent essential commands or reserved terms used within its Tcl (Tool Command Language) scripting environment and graphical user interface (GUI). These keywords serve as fundamental instructions to execute specific actions or configurations throughout various stages of FPGA (Field-Programmable Gate Array) design and implementation. Some commonly used keywords within Vivado include commands such as "synth_design" for initiating synthesis, "implement_design" to trigger the implementation phase, and "write_bitstream" to generate the bitstream file necessary for FPGA configuration. Additionally, keywords like "add_files" enable the addition of design files to projects, "set_property" facilitates the configuration of design elements and constraints, and "report_timing" generates timing analysis reports crucial for understanding design performance. Understanding and using these keywords proficiently empower designers to effectively navigate the Vivado Design Suite, facilitating the creation and optimization of FPGA designs while leveraging its powerful capabilities for efficient development and implementation of complex digital systems

<u>1.2.2.2 Identifiers in Xilinx Vivado Design S.uite:</u>

In Xilinx Vivado Design Suite, identifiers are names assigned to various design elements, including modules, signals, ports, parameters, variables, and constraints, to uniquely identify and reference them within the FPGA (Field-Programmable Gate Array) design. These identifiers serve as labels or handles for different components and are crucial for organizing, specifying, and interconnecting elements within the design hierarchy.

Identifiers within Vivado can be used in HDL (Hardware Description Language) files, such as Verilog or VHDL, as well as in constraints files (e.g., XDC constraints) and scripts (Tcl scripts) that configure and manage the FPGA design.

Proper naming conventions and rules must be followed when using identifiers in Vivado. Typically, identifiers should begin with a letter or an underscore (_) character, followed by letters, digits, or underscores. They are case-sensitive, distinguishing between uppercase and lowercase letters.

Throughout the design process in Vivado, identifiers play a critical role in specifying design components, defining connectivity, setting constraints, and enabling efficient design implementation. Choosing clear, descriptive, and consistent identifiers helps improve the readability, maintainability, and understanding of the FPGA design among design team members and across different stages of the design flow within the Vivado environment. They play a crucial role in specifying, accessing, and interconnecting different design components, facilitating effective communication and understanding among designers and throughout the design process.

1.3 DATA TYPES AND MODELLING STYLES:

1.3.1 Data Types in Verilog HDL:

Verilog HDL (Hardware Description Language) includes various data types used to define and manipulate different kinds of data within digital hardware designs. These data types serve specific purposes and enable the representation of various elements within the design, such as signals, registers, buses, and more. Some commonly used data types in Verilog include:

1. **Wire:** The wire data type represents a continuous assignment that connects different modules or elements within a design. It is commonly used for connections between modules and represents combinational logic.
   Example : wire[7.0] data_wire;

2. **Reg :** The reg data type represents a register, which holds a value that can be updated sequentially in procedural blocks like `always` or `initial` blocks. It stores state information and is used for sequential logic
   Example : reg[3:0] counter;

3. **Integer :** Integer data type is used for representing integer values. It's commonly used for counting or indexing purposes within procedural blocks.
   Example: integer i;

4. **Parameter :** Parameters are constants used to define values that remain constant during simulation and synthesis. They enable easy modification of values without changing the code directly.
   Example: parameter WIDTH = 8;

5. **Logic :** Introduced in Verilog-2001, the logic data type is similar to wire but includes additional logic values (`'0`, `'1`, `'x`, `'z`, `'`), providing better support for representing digital logic.
   Example:10ns;

6. **Arrays :** Verilog supports arrays of various data types, allowing the creation of vectors or arrays of signals or registers.
   Example: reg[7:0]:memory[0.255];

These data types play a fundamental role in describing the behavior and structure of digital hardware within Verilog designs, enabling designers to represent and manipulate different types of data and signals effectively. These data types in Verilog HDL allow the representation of various types of data and their interactions within a hardware design, providing flexibility in modeling hardware systems.

1.3.2 Modeling Styles in Verilog HDL:

In Verilog HDL (Hardware Description Language), several modeling styles are used to represent hardware designs at various levels of abstraction. These styles define how components, their behavior, and their interactions are described. Here are the common modeling styles in Verilog:

1. **Structural Modeling** : Describes hardware using lower-level constructs like gates and modules. Focuses on connections and interconnections between different structural elements.

2. **Behavioral Modeling** : Describes hardware behavior using procedural constructs like always blocks. Defines operations based on events or conditions.

3. **RTL (Register-Transfer Level) Modeling** : Focuses on the transfer of data between registers. Describes data transfer based on clock edges and includes register behaviors.

4. **Dataflow Modeling** : Describes operations using continuous assignments based on data flow .Updates outputs continuously based on changes in inputs.

5. **FSM (Finite State Machine) Modeling** : Models systems with states and transitions using conditional blocks. Represents behavior based on states and inputs.

6. **Verification-Oriented Modeling** : Involves creating test benches to verify design correctness. Uses behavioral constructs to create stimulus and check hardware functionality.

7. **Component-Based Modeling** : Hierarchical design approach using reusable modules or components. Allows building complex systems by composing smaller, well-defined components.

8. **Testbench Modeling** : Describes the environmernt used for testing and verifying the functionality of a design. Includes stimulus generation, checking expected results, and reporting .

9. **Gate Level Modeling** : Describes a design using primitive logic gates . Specifies the exact gates and connection used in the design . Rarely used for large -scale designs due to its low abstraction level.

Each modeling style has its own characteristics and use cases, providing flexibility in representing hardware designs at different levels of abstraction in Verilog HDL. Each modeling style has its unique advantages and is applicable at different stages of the design process. Designers choose a modeling style based on the design complexity, level of abstraction, and specific requirements of the project. By employing these modeling styles effectively, Verilog HDL allows for the efficient and comprehensive representation of hardware. Each modeling style has its own characteristics and use cases, providing flexibility in representing hardware designs at different levels of abstraction in Verilog.

1.4 Verilog Programming For Various Combinational And Sequential Circuits :

Combinational Circuits :

1. 2-to-1 Multiplexer:

Boolean expression: $Y = \bar{S}.I0 + S.I2$

```
module mux_2x1_gl(
input I0,I1,S,
output Y);
wire sb,a,b;

not(sb,S);
and(a,sb,I0);
and(b,S,I1);
or(Y,a,b);

endmodule
```

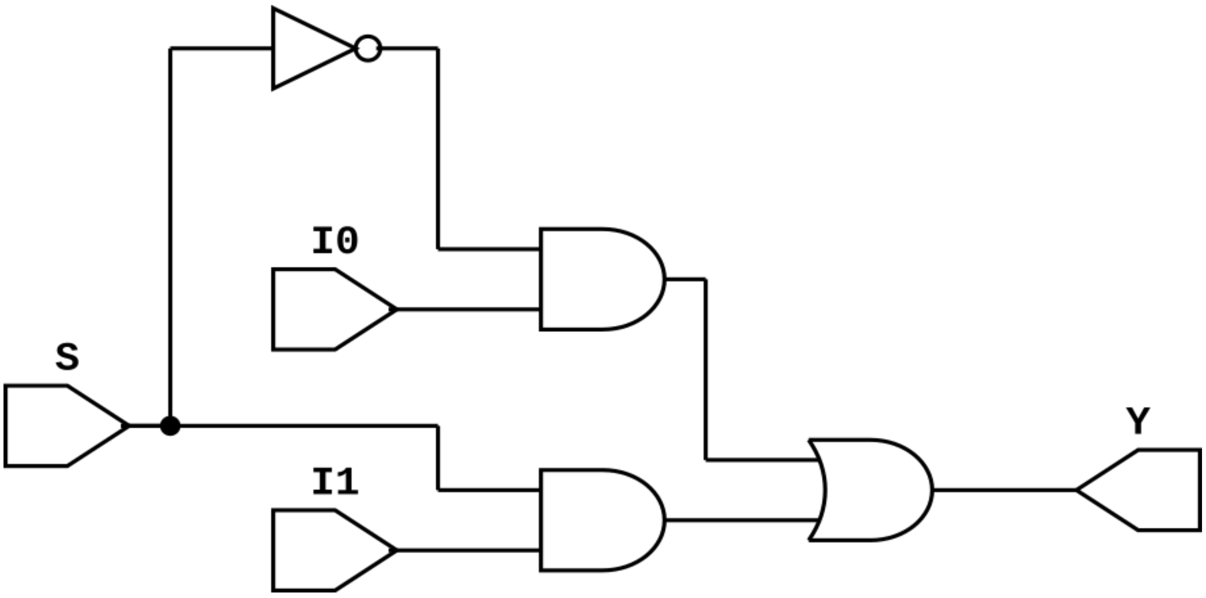


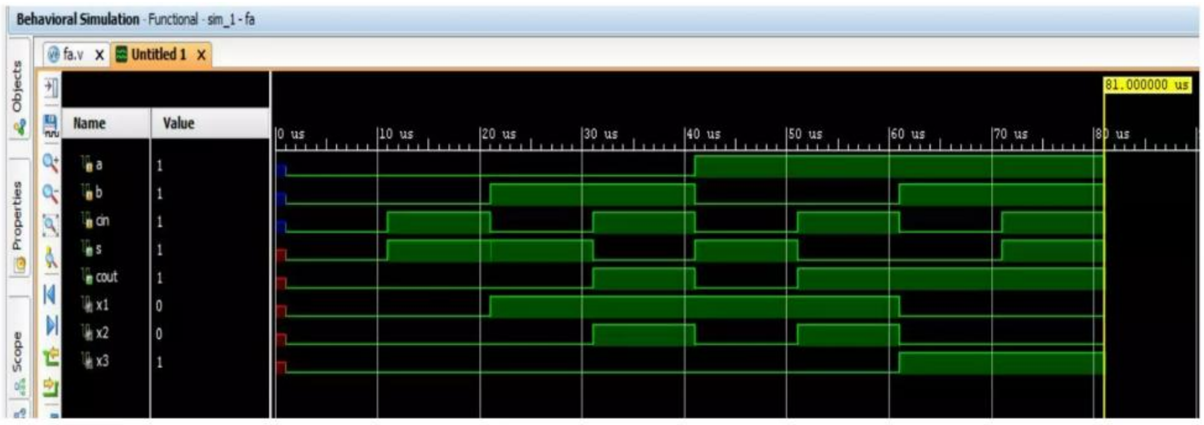
Fig. 1.2 Flow Chart of above program

Table 1. Truth Table of 2x1 MUX

S	Y
0	I0
1	I1

2. Full Adder :

```
module fa(
input a,
input b,
input cin,
output s,
output cout);
wire x1,x2,x3;
xor(x1,a,b);
and(x3,a,b);
xor(s,x1,cin);
and(x2,x1,cin);
 or(cout,x2,x3);
 endmodule
```



Name	Value
a	1
b	1
cin	1
s	1
cout	1
x1	0
x2	0
x3	1

Fig. 1.4 Simulation of above program

Table 3. Truth Table of Full Adder

a	***b***	***cin***	***s***	***cout***
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3. 4-to-1 Multiplexer :

```
module m41(out, a, b, c, d, s0, s1);
output out;
input a, b, c, d, s0, s1;
wire sobar, s1bar, T1, T2, T3, T4;
not (s0bar, s0), (s1bar, s1);
and (T1, a, s0bar, s1bar), (T2, b, s0bar, s1),(T3, c, s0, s1bar), (T4, d, s0, s1);
or(out, T1, T2, T3, T4);
endmodule
```

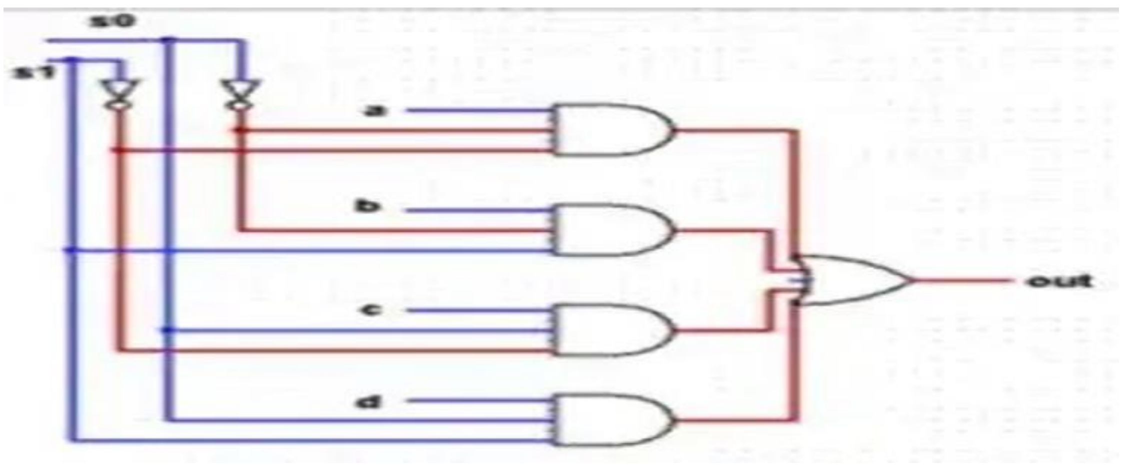


Fig. 1.3 Logic Diagram of 4x1 MUX

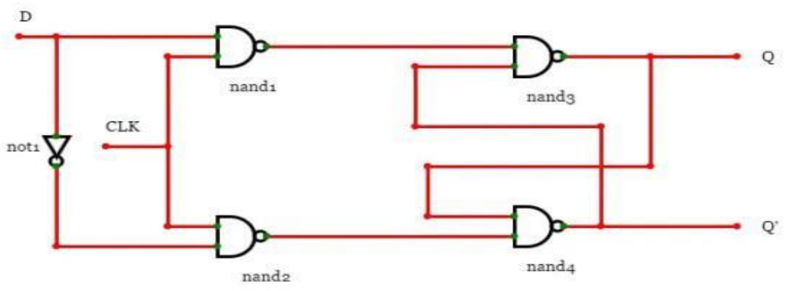
Table 2. Truth Table of 4x1 MUX

INPUT	S1	S0	out
a	0	0	a
b	0	1	b
c	1	0	c
d	1	1	d

Sequential Circuits:

1. D Flip-Flop:

```
module nand_gate(c,a,b);
input a,b;
output c;
assign c = ~(a&b);
endmodule
module not_gate(f,e);
input e;
output f;
assign f= ~e;
endmodule
module d_ff_struct(q,qbar,d,clk);
input d,clk;
output q, qbar;
not_gate not1(dbar,d);
nand_gate nand1(x,clk,d);
nand_gate nand2(y,clk,dbar);
nand_gate nand3(q,qbar,y);
nand_gate nand4(qbar,q,x);
endmodule
```



A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

D	Q
0	0
1	1

CHAPTER-2

INTRODUCTION TO PSPICE

2.1 Introduction:

PSPICE stands for Personal Simulation Program with Integrated Circuit Emphasis. It is a widely used simulation tool for electronic circuits and systems. PSPICE is developed by Cadence Design Systems and is used by engineers, students, and researchers to simulate and analyze electronic circuits before they are physically built.

2.2 KEY FEATURES:

PSPICE (or PSpice) is a powerful simulation tool for electronic circuit design and analysis. Here are some of its key features:

1. **Mixed-Signal Simulation** : PSPICE supports mixed-signal simulation, enabling the modeling and analysis of both analog and digital circuits within the same environment. It allows the simulation of circuits containing a combination of analog, digital, and mixed-signal components.

2. **Extensive Component Libraries** : PSPICE includes extensive libraries of pre-built models for various electronic components such as resistors, capacitors, inductors, diodes, transistors, operational amplifiers (op-amps), analog and digital ICs, and more. These models accurately represent the behavior of real-world components.

3. **Behavioral Modeling** : Users can create custom behavioral models for components that are not available in the standard libraries or to create specialized models for specific simulations. This feature allows the representation of component behavior through equations or code.

4. **Transient and Steady-State Analysis** : PSPICE enables transient analysis to simulate the time-domain behavior of a circuit, showing how the circuit responds to changes over time. Additionally, steady-state analysis allows for the investigation of the circuit's behavior when it reaches a stable condition.

5. **AC and DC Sweep Analysis** : It supports AC analysis for determining the frequency response of a circuit, and DC sweep analysis for varying DC voltage or current sources to study circuit characteristics under different operating conditions.

6**. Monte Carlo Analysis** : PSPICE can perform Monte Carlo analysis, a statistical method used to run multiple simulations with varying component values to assess the circuit's performance across a range of possible scenarios, considering component tolerances and variations.

7. **Parameterized Models** : Users can create parameterized models allowing the variation of component values during simulation, facilitating sensitivity analysis and optimization studies.

8. **Interactive Schematic Capture** : It provides a user-friendly graphical interface for creating and editing circuit schematics, allowing for easy circuit design and modification.

9. **Graphical Output and Waveform Analysis** : PSPICE generates graphical outputs such as waveform plots, frequency response plots, and various analysis results, enabling users to visualize and interpret circuit behavior easily.

10. **Education and Industry Use** : Widely used in educational institutions for teaching electronics and circuit design principles. In industry, it is employed for design verification, optimization, and troubleshooting of electronic circuits and systems.

PSPICE's comprehensive set of features makes it a versatile tool for engineers, designers, and students to simulate, analyze, and validate electronic circuits before actual implementation, helping to reduce design time and costs.

2.3 Xilinx Vivado Design Flow:

Xilinx Vivado is an advanced FPGA design and verification tool suite used for developing and deploying FPGA-based designs. The Vivado Design Suite offers a comprehensive design flow that covers various stages from design creation to implementation and verification. Here is an overview of the Xilinx Vivado Design Flow:

1. **Design Creation and Entry** : Project Creation: Start by creating a new project in Vivado, specifying project settings, target FPGA device, and design type. Design Entry : Use Vivado's integrated development environment (IDE) to enter the design. Design entry can be done using HDL (Hardware Description Language) such as Verilog, VHDL, or by using high-level synthesis (HLS) for C/C++-based designs.

2. **Design Analysis and Simulation**:

Functional Simulation : Verify the functionality of the design through simulation using Vivado Simulator or third-party simulators. Ensure that the design behaves as expected before moving to synthesis.

3**. Synthesis and Optimization** :

Synthesis : Translate the RTL (Register Transfer Level) code into a gate-level netlist that represents the design's functionality in terms of logic gates and flip-flops. Optimization : Perform optimizations to improve the design's performance, area, and timing constraints. Vivado offers various optimization techniques for better implementation results.

4. **Constraint Specification**:

Constraints Definition : Define constraints such as timing constraints, pin assignments, clock definitions, and I/O standards. These constraints ensure proper operation of the design within specified requirements.

5**. Implementation** :

Place and Route : Place the design elements (logic, memory, I/Os) onto the FPGA device and route the interconnections between them. Vivado's place and route algorithms optimize for timing, area, and power. Timing Analysis : Perform static timing analysis (STA) to verify that the design meets the timing constraintsand meets the required performance criteria. Identify any timing violations and address them through design changes or constraints adjustments.

6. **Design Verification** :

Post-Implementation Simulation : After the design is implemented, run post-implementation simulations to verify its functionality and performance against the RTL simulation results. Verification with Hardware Emulation or Prototyping : For more rigorous verification, utilize hardware emulation or prototype boards to test the design in a real hardware environment.

7**. Design Validation and Debugging** :

Validation: Validate the implemented design against the intended functionality and performance specifications. Debugging Tools : Vivado provides various debugging tools and utilities to debug issues such as timing violations, functional errors, or resource utilization problems.

8. **Generate Bitstream and Configuration**:

Bitstream Generation : Create the bitstream file that configures the FPGA with the synthesized and implemented design.Configuration Programming : Load the generated bitstream onto the target FPGA device for configuration, either via JTAG, PROM, or other programming methods.

9. **Hardware Validation and Deployment** :

Hardware Validation : Deploy the programmed FPGA onto the target hardware platform for real-world validation and testing. Deployment to Production : Once validated, the design can be deployed for production or deployment in end-user applications.

10**. Documentation and Project Management** :

Project Documentation : Document the design, constraints, synthesis, implementation, and verification steps for future reference and team collaboration. Project Management**:** Manage and organize design files, constraints, and project settings within the Vivado project environment.

CHAPTER-3

MINI PROJECT DESCRIPTION

“Verilog Implementation of Digital Circuit Designs on FPGA Using Vivado”

Topic: Traffic Light Controller Design using Verilog

3.1 INTRODUCTION TO TRAFFIC LIGHT CONTROLLER:

A traffic light controller is a device or system that manages the operation and timing of traffic lights at intersections or roadways. Its primary purpose is to regulate the flow of vehicular and pedestrian traffic by controlling when each signal (red, yellow/amber, and green lights) is displayed.

The controller uses a combination of hardware and software to coordinate the sequence and duration of the traffic signal phases. The main components of a traffic light controller typically include:

1. **Sensors:** These detect the presence of vehicles or pedestrians waiting at the intersection. Inductive loops embedded in the road, cameras, infrared sensors, or pressure sensors are commonly used for this purpose.

2. **Controller Unit:** This is the central processing unit that manages the timing and sequencing of the traffic lights based on inputs from the sensors. It decides when to change the signals and how long each phase should last.

3. **Signal Heads:** These are the actual lights mounted on poles or overhead structures at the intersection. They include red, yellow/amber, and green lights for different directions of traffic flow and pedestrian crossings.

4. **Communication System:** In some advanced systems, controllers may be connected to a central traffic management system or network to receive real-time data, adjust signal timing dynamically, or synchronize signals across multiple intersections to optimize traffic flow.

The operation of a traffic light controller typically follows a predefined sequence of phases:

Green Phase: One direction of traffic is given the right-of-way, allowing vehicles to proceed while other directions are stopped.
Yellow/Amber Phase: This acts as a warning to clear the intersection before the signal changes to red.
Red Phase: All traffic is required to stop, allowing other directions or pedestrians to safely cross.

Modern traffic light controllers may use advanced algorithms to adaptively adjust signal timings based on traffic conditions, prioritizing bus lanes, emergency vehicles, or high-traffic routes during peak hours to improve overall traffic efficiency and safety.

Overall, traffic light controllers play a crucial role in managing traffic flow, enhancing safety, and optimizing the efficiency of transportation systems in urban areas.

3.2 KEY FEATURES OF TRAFFIC LIGHT CONTROLLER:

Modern traffic light controllers are equipped with several key features aimed at efficiently managing traffic flow and ensuring safety on roadways. These controllers offer adaptive control mechanisms that dynamically adjust signal timings based on real-time data from sensors, cameras, and traffic flow information. They facilitate intersection coordination, synchronizing signals to create green waves along corridors, minimizing stops and delays for vehicles. Pedestrian safety features, such as specific walk signals and crossing times, prioritize safe crossings. These controllers also support transit priority, giving precedence to public transportation vehicles, and provide preemption capabilities for emergency vehicles to swiftly navigate intersections. Additionally, they incorporate fault detection systems, enabling timely maintenance, and offer manual override and remote management functionalities for adjustments. Integrated with Intelligent Transportation Systems (ITS), traffic light controllers exchange data with traffic management centers and other infrastructure elements, optimizing traffic flow and contributing to energy efficiency through the use of LED lights and optimized timings. Data collection capabilities assist traffic engineers in analyzing trends for further enhancements in traffic management strategies.

Overall, these key features collectively enable traffic light controllers to create safer, smoother, and more efficient traffic operations.

Multi-Modal Control: Capable of controlling traffic for vehicles, pedestrians, cyclists, and other modes of transport at intersections or junctions.

Adaptive Control: Utilizes real-time data from sensors, cameras, and traffic flow information to adjust signal timings dynamically based on current traffic conditions, optimizing traffic flow and minimizing congestion.

Preemption Capabilities: Allows emergency vehicles to override regular signal patterns, providing them with priority clearance at intersections to ensure rapid response times.

Intersection Coordination: Synchronizes traffic signals at multiple intersections along a corridor to create green waves, minimizing stops and delays for vehicles traveling through consecutive intersections.

Pedestrian Safety Features: Provides specific walk signals and crossing times for pedestrians, ensuring safe crossing at intersections. Some controllers include audible signals or tactile feedback for visually impaired pedestrians.

Transit Priority: Offers priority to public transportation vehicles, such as buses or trams, by adjusting signal timings to reduce their waiting times and enhance the efficiency of public transit systems.

VERILOG CODE :

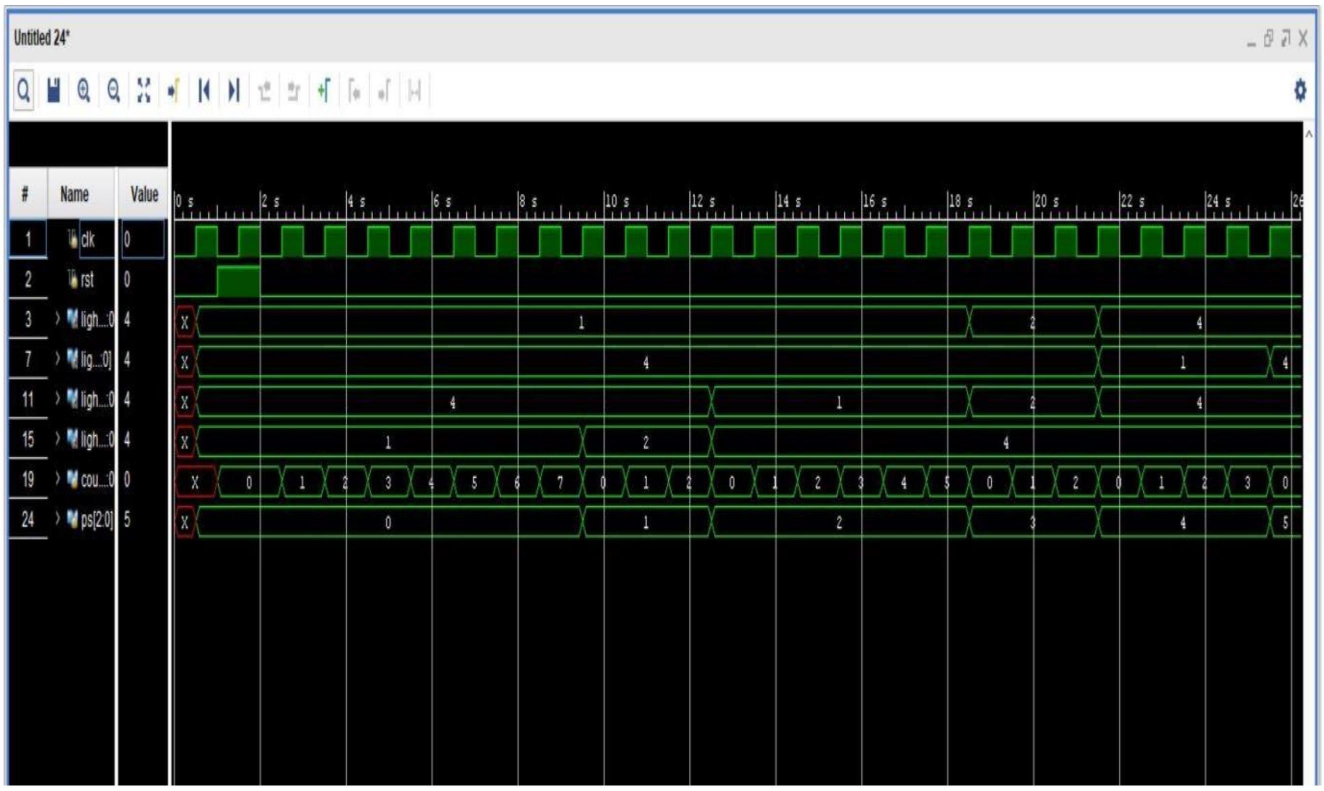
```
module Traffic_Light_Controller(
input clk,rst,
output reg [2:0]light_M1,
 output reg [2:0]light_S,
 output reg [2:0]light_MT,
output reg [2:0]light_M2
);
parameter S1=0, S2=1, S3 =2, S4=3, S5=4,S6=5;
reg [3:0]count;
 reg[2:0] ps;
parameter sec7=7,sec5=5,sec2=2,sec3=3;
always@(posedge clk or posedge 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'b100;
 end
 default:
 begin
 light_M1<=3'b000;
 light_M2<=3'b000;
 light_MT<=3'b000;
 light_S<=3'b010;
end
endcase
end
endmodule
```

SIMULATED WAVEFORM



Upon analysing the waveform we can clearly see that the FSM works perfectly. The waveform used in a traffic light controller typically involves a sequence of signals to regulate

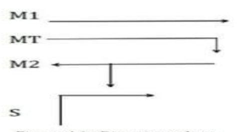


Figure 4.1 : Directions chart

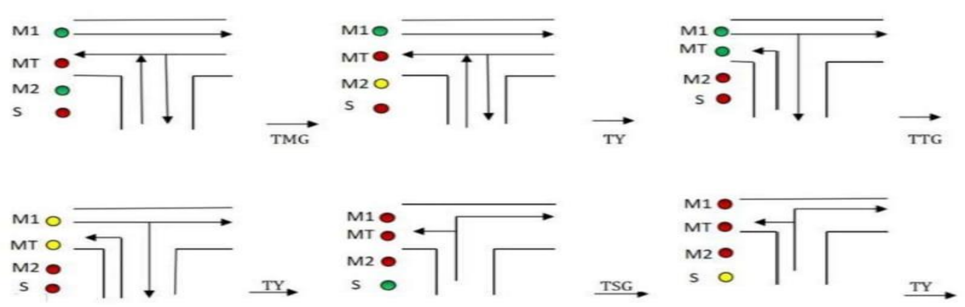


Figure 4.2 : Problem statement (Logic)

the flow of traffic. However, the specific waveform can vary based on the type of controller, region, and traffic management system used. Traffic light controllers commonly use a combination of digital signals, timers, and programming logic to control the lights.
A simple representation of a traffic light controller's waveform could be described as follows:
1. Green Light (for a set duration, let's say 30 seconds for an example)
2. Yellow Light (for a brief period, perhaps 5 seconds as a transition)
3. Red Light (for a set duration, e.g., 30 seconds)

DIRECTION FLOW CHART:

PROBLEM STATEMENT:

The aim of the project is to design a traffic light controller system for T intersection.

Let’s understand the problem statement through the image given above .

M1: Represents the road moving from left to right.

MT: Represents the main turn from the main road

M2: Represents the road moving from right to left and from top to bottom

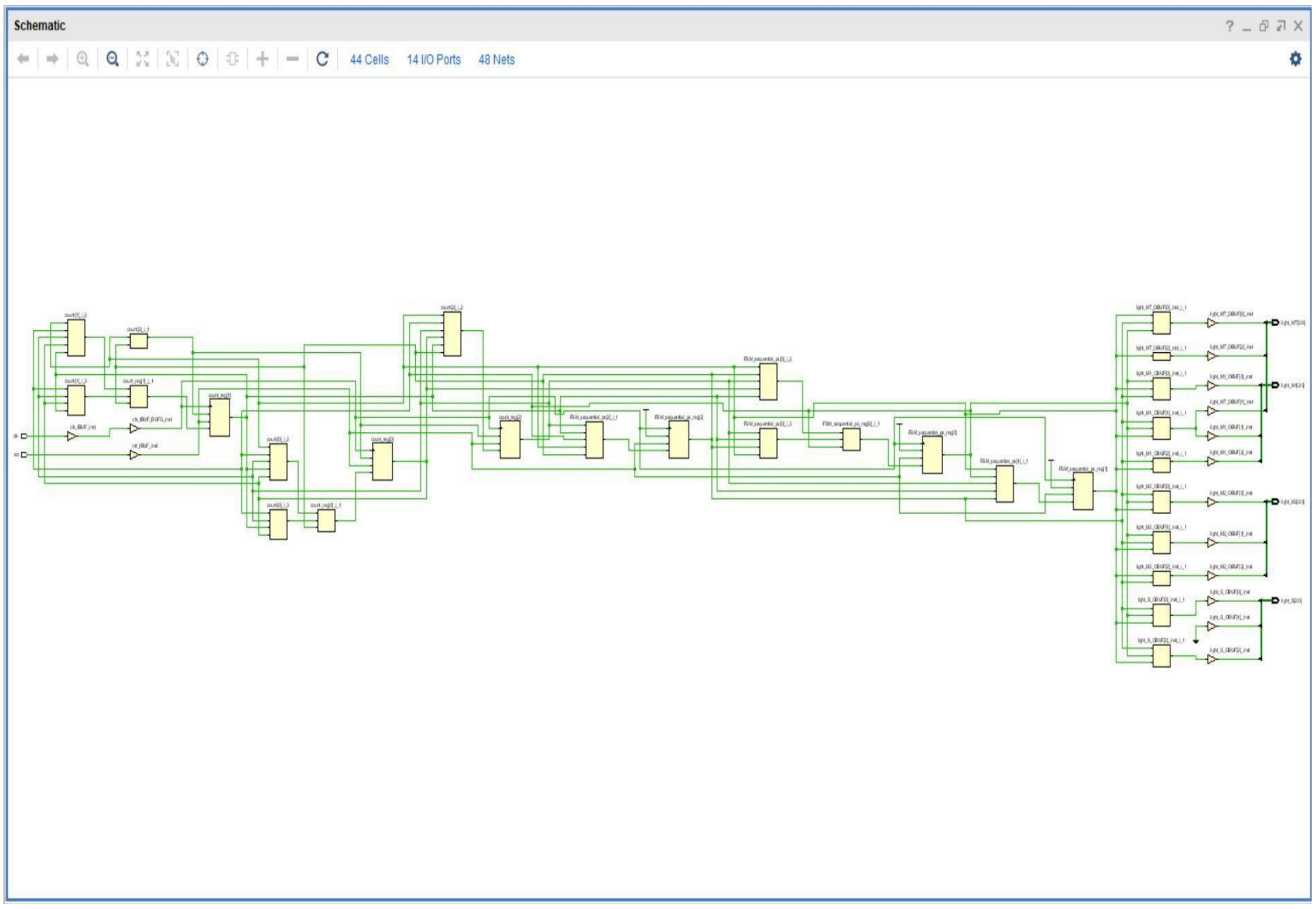
S : Represents the side road merging into the main road.

1. Green light indicates that there is no traffic and there is easy flow of vehicles in that route or direction

2. Red indicates the there is a traffic jam and that route is blocked for the vehicle to move .

3. Yellow indicates that the route has medium flow of vehicles .

SCHEMATIC AFTER IMPLEMENTATION :



DEVICE LAYOUT AFTER IMPLEMENTATION:

APPLICATIONS :

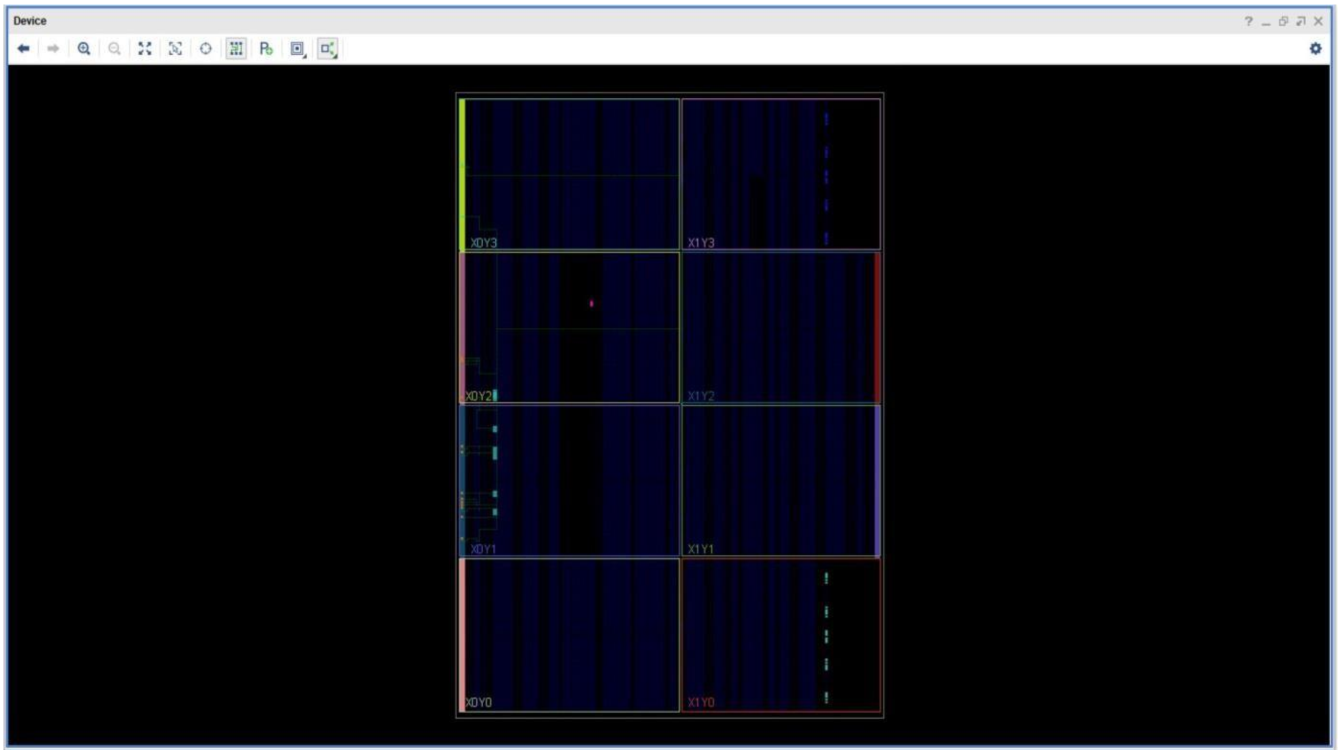
We can avoid unnecessary occurrence of traffic jams which causes public inconvenience.We cansave considerable amount of time Density based traffic light control have many advantagescompared to time based traffic control.The intelligent work which is done by traffic inspector will be perfectly done by the Arduino inthe circuit with the help of sensors and the program which is coded to the Arduino . There is noneed of traffic inspector at the junctions for supervising the traffic to run smoothly.Traffic light controllers are crucial components of traffic management systems and have various applications aimed at ensuring safety, regulating traffic flow, and optimizing transportation efficiency. Here are some key applications of traffic light controllers:

Urban Traffic Management: Traffic light controllers are extensively used in urban areas to regulate the flow of vehicles, pedestrians, and cyclists at intersections and junctions. They help in reducing congestion, managing traffic volumes, and improving overall traffic flow.

Intersection Control: Traffic light controllers play a pivotal role in controlling intersections, enabling safe and efficient movement of vehicles from different directions. They manage the timing and sequencing of traffic lights to facilitate smooth transitions between different streams of traffic.

Pedestrian Safety: These controllers include features that cater to pedestrian safety by providing specific walk signals and crossing times at intersections. They ensure that pedestrians have sufficient time to cross roads safely.

Public Transport Priority: In certain cities, traffic light controllers are configured to give priority to public transportation vehicles such as buses or trams. This feature helps in reducing travel times for public transport users and encourages the use of sustainable modes



of transportation.

Emergency Vehicle Preemption: Traffic light controllers can be equipped with systems that allow emergency vehicles, such as ambulances, fire trucks, or police cars, to override regular signal patterns, enabling them to navigate intersections swiftly and reach their destinations quickly.

Adaptive Traffic Control: Advanced traffic light controllers use real-time data from sensors, cameras, and traffic flow information to adjust signal timings dynamically. This adaptive control helps in responding to changing traffic conditions, reducing congestion, and optimizing traffic flow.

Traffic Surveillance and Data Collection: Traffic light controllers often incorporate sensors and cameras to collect data on traffic patterns, vehicle counts, and congestion levels. This data is valuable for traffic engineers and planners to analyze and optimize road networks.

n essence, traffic light controllers serve a variety of applications aimed at efficiently managing traffic, enhancing safety for all road users, and optimizing the functionality of transportation systems within urban and suburban environments.

CHAPTER-4

CONCLUSION:

In conclusion, traffic light controllers stand as integral components within transportation systems, offering a range of essential features that facilitate the smooth and safe flow of traffic. These controllers are not just static signal systems but dynamic, adaptive tools that respond to real-time data, optimizing traffic patterns and enhancing safety for motorists, pedestrians, and cyclists alike. Equipped with adaptive control mechanisms, pedestrian safety features, transit priority options, and the ability to synchronize signals for efficient corridor travel, these controllers are pivotal in reducing congestion and improving overall traffic flow. Moreover, their integration with Intelligent Transportation Systems (ITS) enables data-driven decision-making, leading to enhanced efficiency, reduced environmental impact, and improved energy conservation through advanced technologies like LED lighting and optimized timings. As urban environments evolve, traffic light controllers continue to evolve as well, playing a vital role in modern transportation management systems aimed at creating safer, smarter, and more sustainable cities for the future.

FUTURE SCOPE :

The future scope of traffic light controllers is evolving in several directions due to advancements in technology and the increasing need for efficient traffic management systems. Here are some potential developments and trends in the field:

Smart Traffic Management Systems: Traffic light controllers are moving towards becoming an integral part of larger smart traffic management systems. These systems employ AI, machine learning, and real-time data analysis to optimize traffic flow, reduce congestion, and enhance safety.

Connected and Autonomous Vehicles (CAVs): With the rise of connected and autonomous vehicles, traffic light controllers will likely integrate with these vehicles to enable communication between vehicles and infrastructure. This communication can optimize traffic patterns, reduce accidents, and improve overall efficiency.

Adaptive Traffic Control: Future traffic light controllers may use sensors, cameras, and advanced algorithms to adapt signal timings in real-time based on traffic conditions, pedestrian flows, and environmental factors. This adaptability can lead to more efficient traffic management and reduced waiting times.

Integration with IoT (Internet of Things): IoT integration could allow traffic lights to communicate with various devices, such as smartphones or wearables, to provide real-time updates to pedestrians and drivers, improving overall safety and convenience.

Energy Efficiency and Sustainability: There is a growing focus on making traffic light systems more energy-efficient by utilizing renewable energy sources, such as solar power, and optimizing power consumption through smart algorithms.

Cloud-Based Control and Analytics: Traffic light controllers may move towards cloud-based management systems, allowing for centralized control, data storage, and analytics. This could enable more comprehensive analysis of traffic patterns and trends for better decision-making.

Augmented Reality and V2X (Vehicle-to-Everything) Communication: Integration of augmented reality technology in vehicles and V2X communication can provide drivers with real-time information about traffic light timings, road conditions, and potential hazards, improving safety and traffic flow.

Environmental Considerations: Future traffic light controllers might incorporate environmental sensors to monitor air quality and adjust traffic patterns to reduce emissions in highly congested areas or during specific times of the day.

Overall, the future of traffic light controllers is likely to be shaped by advancements in connectivity, data analytics, AI, and sustainability, all aimed at creating safer, more efficient, and smarter transportation systems.

REFERENCES :

[1] Zimmermann, R., & Fichtner, W. (1997). Low-power logic styles: CMOS versus pass-transistor logic. IEEE Journal of Solid-State Circuits, 32(7), 1079-1090. https://doi.org/10.1109/4.597298

[2] Kandpal, J., Tomar, A., Agarwal, M., & Sharma, K. K. (2020). High-speed hybrid-logic full adder using high-performance 10-T XOR–XNOR cell. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 28(6), 1413-1422. https://doi.org/10.1109/tvlsi.2020.2983850

[3] Goel, S., Kumar, A., & Bayoumi, M. A. (2006). Design of robust, energy-efficient full adders for deep-submicrometer design using Hybrid-CMOS logic style. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 14(12), 1309-1321. https://doi.org/10.1109/tvlsi.2006.887807

[4] Chip-Hong Chang, Jiangmin Gu, & Mingyan Zhang. (2005). A review of 0.18-/SPL Mu/M full adder performances for tree structured arithmetic circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 13(6), 686-695. https://doi.org/10.1109/tvlsi.2005.