FULLY AUTOMATED TRAFFIC LIGHT CONTROLLER SYSTEM FOR A FOUR-WAY INTERSECTION USING VERILOG

A project report submitted in partial fulfillment of the requirements for the award of

Bachelor of Technology

In

ELECTRONICS & COMMUNICATION ENGINEERING

By

K.V.L SARANYA	**M.LAVANYA**
(21BQ1A0458)	**(21BQ1A0484)**
N.LAKSHMI SRAVANI	**V.CHANDINI**
(21BQ1A0487)	**(22BQ5A0418)**

UNDER THE GUIDANCE OF

Dr. Shaik Riyazuddien

Professor

VVIT
VASIREDDY VENKATADRI
INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE and permanently affiliated to JNTUK)

Accredited by NBA and NAAC with 'A' Grade

NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508

APRIL 2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY: NAMBUR

(AUTONOMOUS)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

CERTIFICATE

This is to certify that the project titled "**FULLY AUTOMATED TRAFFICLIGHT CONTROLLER FOR A FOUR-WAY INTERSECTION USING VERILOG**" is a bonafide record of work done by **Ms.K.V.L SARANYA (21BQ1A0458), Ms.M.LAVANYA (21BQ1A0484), Ms.N.LAKSHMI SRAVANI (21BQ1A0487)** and **Ms.V.CHANDINI (22BQ5A0418)** under the guidance of **Dr. Shaik Riyazuddien, Professor** in partial fulfillment of the requirement of the degree for Bachelor of Technology in Electronics and Communication Engineering, JNTUK during the academic year 2024–25.

Dr. Shaik Riyazuddien
Project Guide

Prof. M.Y.BhanuMurthy
Head of the Department

DECLARATION

We, **K.V.L SARANYA (21BQ1A0458), M.LAVANYA (21BQ1A0484), N.LAKSHMI SRAVANI (21BQ1A0487) and V.CHANDINI (22BQ5A0418) ,** here by declare that the Project Report entitled " **FULLY AUTOMATED TRAFFIC LIGHT CONTROLLER FOR A FOUR-WAY INTERSECTION USING VERILOG**" done by us under the guidance of **Dr.SK.RIYAZUDDIEN, Professor, Department of ECE** is submitted in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING.**

DATE :

PLACE :VVIT, NAMBUR.

SIGNATURE OF THE CANDIDATES

K.V.L SARANYA

M. LAVANYA

N. LAKSHMI SRAVANI

V. CHANDINI

ACKNOWLEDGEMENT

We express our sincere thanks wherever it is due

We express our sincere thanks to the Chairman, Vasireddy Venkatadri Institute of Technology, Sri Vasireddy Vidya Sagar for providing us well equipped infrastructure and environment.

We thank Dr. Y. Mallikarjuna Reddy, Principal, Vasireddy Venkatadri Institute of Technology, Nambur , for providing us the resources for carrying out the project.

We express our sincere thanks to Dr. K. Giribabu, Dean of Academics for providing support and stimulating environment for developing the project.

Our sincere thanks to Prof. M. Y. Bhanu Murthy, Head of the Department, Department of ECE, for his co-operation and guidance which helped us to make our project successful and complete in all aspects.

We also express our sincere thanks and are grateful to our guide Dr. Shaik Riyazuddien, Professor, Department of ECE, for motivating us to make our project successful and fully complete. We are grateful for his precious guidance and suggestions.

We also place our floral gratitude to all other teaching staff and lab technicians for their constant support and advice throughout the project.

NAME OF THE CANDIDATES

K. Saranya(21BQ1A0458)

M. Lavanya (21BQ1A0584)

N. Sravani (21BQ1A0487)

V. Chandini (22BQ5A0418)

TABLE OF CONTENTS

TITLE OF CONTENTS	PAGE NO
List of Abbrevations	i
List of Figures	ii
List of Tables	iv
Abstract	v
CHAPTER 1: INTRODUCTION	
1.1 Overview of traffic light controller	1
1.2 Significance of project	2
1.3 Motivation	2
1.4 Objective	3
1.5 Scope of the study	4
1.6 Contribution of the study	5
CHAPTER 2: LITERATURE SURVEY	**8**
CHAPTER 3: EXISTING SYSTEM	
3.1 Existing method architecture	10
3.2 Limitations and Challenges	11
CHAPTER 4: DESIGN AND METHODOLOGY	
4.1 Proposed method architecture	14
CHAPTER 5: SOFTWARE DEVELOPMENT & PROJECT DEVELOPMENT	
5.1 Xilinx VIVADO	16
5.2 Verilog programming language	17
5.3 History	18
5.4 VIVADO PROCESS	19
5.5 Software Tools Used	43
5.6 State Diagram	44
5.7 State Table	45
5.8 Software Environment	46
CHAPTER 6: ANALYSIS ON SIMULATION& IMPLEMENTATION	**48**
6.1 Simulation results of proposed model	48
6.2 RTL schematic	51
6.3 Utilization Report	52
6.4 Power report	54
6.5 Technology Schematic	56

6.6 Device layout 57
6.7 Advantages 59
CHAPTER 7: CONCLUSION AND FUTURE SCOPE 60
7.1 Conclusion 60
7.2 Future scope 60
REFERENCES 61
APPENDIX 62

LIST OF ABBREVIATIONS

RTL: Register Transfer Level

FPGA: Field Programable Gate Array

LIST OF FIGURES

Fig 3.1	Traffic Signals	13
Fig 4.1	Proposed Model of Traffic Light Controller	15
Fig 5.1	VIVADO HLX Editions	17
Fig 5.2	VIVADO Home Page	20
Fig 5.3	Create Project Setup	21
Fig 5.4	Create project Dialog	21
Fig 5.5	Enter Project Name	22
Fig 5.6	Select Project Type	23
Fig 5.7	Add Sources	25
Fig 5.8	Add Constraints File	27
Fig 5.9	Select ARTIX-7 CPG26 Board	28
Fig 5.10	Create project Summary	28
Fig 5.11	VIVADO Project Window	30
Fig 5.12	Add Design Source	34
Fig 5.13	Add or Create Design Sources Window	36
Fig 5.14	Create Design Source File	37
Fig 5.15	Demo File Created Dialog	39
Fig 5.16	State Diagram	44
Fig 5.17	Software Logo	46
Fig 6.1	Simulation Result	48
Fig 6.2	RTL Schematic	51
Fig 6.3	Utilization Report	52
Fig 6.4	Power Report	54
Fig 6.5	on chip Power	55
Fig 6.6	Technology Schematic	56
Fig 6.7	Device Layout Diagram	57
Fig 6.8	Pedestrian condition	58
Fig 6.9	Emergency condition	58

LIST OF TABLES

Table 1 Truth Table of State Diagram 40

ABSTRACT

The “Fully Automated Traffic Light Controller System for a Four-Way Intersection Using Verilog” aims to design an efficient and reliable traffic management system to regulate vehicular and pedestrian movement at a four-way intersection. Traffic control is a critical aspect of modern urban infrastructure, and the proposed system addresses the challenges of congestion, safety, and efficiency. This project employs Verilog, a hardware description language, to model and simulate the functionality of the traffic light controller.

Additional features, such as pedestrian crossing controls and emergency override signals, are integrated to enhance usability and safety. The design is modular, scalable, and can be adapted to incorporate advanced functionalities, such as adaptive traffic control using real-time sensor inputs. Simulation results demonstrate the system’s ability to handle traffic flows effectively while maintaining simplicity and cost-efficiency. This project lays a foundation for future developments in smart traffic management systems and provides a framework for deploying automated traffic controllers in Urban Environments.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF TRAFFIC LIGHT CONTROLLER:

Traffic management is a crucial aspect of modern urban infrastructure, ensuring the smooth and safe movement of vehicles and pedestrians. Conventional traffic light systems operate on fixed time cycles, often leading to inefficiencies during low-traffic conditions or emergencies. To address these issues, we have developed an Automatic Traffic Light Controller for a Four-Way Intersection using Verilog, incorporating pedestrian signals and an emergency override mechanism using a Finite State Machine (FSM).

This project aims to design and implement an intelligent traffic control system that efficiently manages vehicular and pedestrian movement at an intersection. The system is implemented using Verilog hardware description language (HDL) and simulated using Vivado to ensure correct functionality before deployment. The FSM-based design ensures systematic state transitions for different traffic conditions while responding dynamically to emergency situations.

Project Features:

Four-Way Traffic Control: The system controls traffic flow at a four- way intersection, allowing vehicles from each direction to pass in a controlled manner.

Pedestrian Signal Integration: Dedicated pedestrian signals are included to ensure safe crossings, activated when a pedestrian request is detected. The system provides a sufficient time window for pedestrians to cross safely before switching back to vehicular traffic.

Four-Way Intersection Control: Manages traffic at a four-way junction.

Sequential Light Timing*:* Follows a structured cycle: Green → Yellow → Red.

Finite State Machine (FSM) Based Control: The entire system is structured using an FSM, defining states for normal traffic flow, pedestrian crossings, and emergency situations. State transitions occur based on predefined timing sequences or external inputs such as pedestrian requests and emergency signals.

Emergency Override Mechanism: An emergency vehicle detection system is implemented, allowing emergency vehicles (ambulances, fire trucks, police vehicles) to override normal traffic operations. Upon detection of an emergency request, the system immediately switches to a state that provides a green light for the emergency lane while stopping all other directions.

Verilog-Based Hardware Implementation: The system is coded in Verilog, a widely used HDL for designing digital circuits. The design is synthesized and simulated in Vivado, ensuring correctness and optimal performance before hardware implementation.

1.2 Significance of the Project:

Enhances Traffic Efficiency: The FSM driven control optimizes the traffic flow, reducing unnecessary wait times.

Improves Road Safety: Pedestrian signals ensure safe crossings,and emergency vehicles receive priority, reducing delays in critical situations.

Hardware-Ready Design: The Verilog implementation makes the system suitable for FPGA deployment, enabling real-time operation.

Improves Traffic Flow: Ensures smooth movement of vehicles by managing signal transitions efficiently.

Enhances Road Safety*:* Reduces accidents by enforcing structured signal timing and pedestrian crossing controls.

Supports Emergency Vehicles*:* Emergency mode allows ambulances and other priority vehicles to pass without delay.

Reduces Human Intervention*:* Eliminates the need for manual traffic control, making it fully automated.

1.3 Motivation:

Traffic congestion is one of the most pressing challenges in urban transportation systems, leading to long waiting times, increased fuel consumption, and higher pollution levels. Conventional traffic light control systems operate on fixed time cycles, failing to adapt to dynamic traffic conditions, pedestrian crossings, or emergency vehicle requirements. This inefficiency often results in unnecessary delays and potential risks for both motorists and pedestrians. The need for a more intelligent, adaptive, and

efficient traffic management system serves as the primary motivation for this project.

One of the major concerns in traffic control is pedestrian safety. In many intersections, pedestrians often struggle to cross roads safely due to poorly managed pedestrian signals or the lack of dedicated crossing time. A well- integrated pedestrian signal system is crucial in ensuring smooth coordination between vehicular and pedestrian movement, reducing accidents and improving overall road safety.

Another critical issue is the delay faced by emergency vehicles, such as ambulances, fire trucks, and police cars, which require immediate clearance through traffic intersections. In traditional traffic systems, emergency vehicles often get stuck in congestion, leading to life-threatening delays in medical emergencies and other critical situations. Implementing an emergency override mechanism that prioritizes emergency vehicles by temporarily altering the normal traffic light sequence can significantly enhance emergency response times and potentially save lives.

The advancement of digital hardware and hardware description languages (HDLs) like Verilog has enabled the development of high-speed, reliable, and real-time traffic control systems. Using Finite State Machine (FSM) logic, the traffic light controller can systematically manage signal transitions in an optimized manner. This approach ensures better traffic flow, minimizes waiting time, and enhances overall efficiency. The motivation behind this project is to design a smart traffic light controller that integrates pedestrian safety measures and emergency vehicle prioritization using Verilog.

1.4 OBJECTIVE:

The objective of the Fully Automated Traffic Light Controller system is to design an efficient and reliable solution for managing traffic at a four-way intersection. It aims to optimize traffic flow by dynamically adjusting light timings based on real- time traffic conditions. The proposed traffic light control system aims to ensure smooth traffic flow through automated signal management while enhancing pedestrian safety with request-based crossing signals. It prioritizes emergency vehicles by providing a clear path, reducing delays for ambulances and fire trucks. The system is designed and implemented using Verilog in Vivado, enabling efficient, real-time execution with

scalability for future enhancements. By integrating automation and adaptability, this model improves urban traffic management, ensuring safety, efficiency, and responsiveness.

The project is automatic traffic light controller for a four-way intersection using Verilog and a Finite State Machine (FSM) approach. The system ensures efficient traffic management by regulating the red, yellow, and green signals in a synchronized manner, minimizing congestion, and enhancing road safety. Additionally, pedestrian crossing signals are incorporated to facilitate safe movement, with clear "Walk" and "Don't Walk" indicators. An emergency override feature is also included, allowing priority access for emergency vehicles by temporarily halting other traffic. The project is simulated using Vivado to verify its functionality, with the potential for FPGA implementation in real-world scenarios. Ultimately, the system aims to optimize traffic flow, reduce wait times, and improve overall intersection Safety.

The objective of this project is to design and implement a fully automated traffic light controller system for a four-way intersection using Verilog and deploy it on the Basys3 FPGA kit. The system aims to efficiently manage traffic flow by ensuring that only one direction has a green light at a time while the others remain red. It incorporates predefined timing sequences for green, yellow, and red lights to regulate traffic movement effectively. Additionally, the system includes emergency and pedestrian control features, allowing emergency vehicles to pass safely and pedestrians to cross the road securely. By leveraging FPGA-based implementation, the design ensures real- time operation, low power consumption, and scalability, making it suitable for modern traffic management systems.

1.5 SCOPE OF THE STUDY:

The study's scope includes the study focuses on the design, simulation, and implementation of a fully automated traffic light controller (TLC) for a four-way intersection using Verilog. The controller regulates traffic flow efficiently and introduces additional safety features such as pedestrian crossing control and emergency vehicle priority, enhancing the overall functionality and reliability of the system. This research addresses the limitations of conventional traffic light controllers by introducing advanced features that adapt dynamically to real-time traffic situations, ensuring better

control and safety.

The scope of study for our project, "Automatic Traffic Light Controller for a Four-Way Intersection using Verilog," covers multiple areas, including traffic management, digital system design, and real-time implementation. Your project focuses on designing an efficient, FSM-based traffic control system that ensures smooth vehicle movement, pedestrian safety, and emergency vehicle prioritization. The system is implemented using Verilog and simulated in Vivado, making it suitable for FPGA-based real-time applications.

The study primarily involves finite state machine (FSM) design, sequential logic implementation, and timing control, which are crucial in digital circuit design. The system transitions through various states normal traffic flow, pedestrian crossings, and emergency overrides based on external inputs and pre-defined logic. The scope also extends to real-world applications, as the project can be further developed by integrating sensor-based traffic monitoring, AI-driven traffic optimization, and real-time FPGA implementation.

The scope includes:

Traffic Signal Control: Ensuring that only one direction has a green light at a time, while others remain red, following a systematic sequence.

Predefined Timings: Implementing fixed-duration signals (green for 30 seconds, yellow for 3 seconds) for efficient traffic regulation.

Pedestrian Crossing Management: Providing safe crossing opportunities for pedestrians when requested.

1.6 CONTRIBUTION OF THE STUDY:

The contribution of this study to your project is significant in multiple ways, particularly in improving traffic management, enhancing safety, and showcasing the application of digital design in real-world scenarios. Your project introduces an intelligent traffic control system that efficiently manages vehicle movement at a four-way intersection. By implementing a finite state machine (FSM) in Verilog, the system ensures a structured and systematic transition between different traffic states, making it more efficient than traditional timer-based systems.

One of the most important contributions of this study is the enhancement of pedestrian safety and emergency response mechanisms. The inclusion of pedestrian signals allows safe crossing by ensuring vehicles halt when required, reducing the chances of accidents. Additionally, the emergency override system ensures that emergency vehicles such as ambulances and fire trucks receive immediate priority, minimizing response time in critical situations.

From a technological perspective, the study contributes to digital circuit design by demonstrating the use of Verilog in traffic management systems. The FSM-based approach optimizes the control flow, making the system more adaptable to real-time traffic conditions. Unlike traditional fixed-time signal systems, this design responds dynamically to external triggers, improving overall efficiency.

This study also aligns with the development of smart city infrastructure, where automation plays a key role in urban planning. By integrating an FSM- driven control mechanism, your project presents a scalable solution that can be further expanded to include sensor-based traffic detection and AI-driven optimizations. Moreover, from an academic perspective, this project serves as a valuable reference for future research and practical implementation of FPGA-based traffic systems. It provides hands-on experience in hardware description languages and their real-world applications, making it a relevant and impactful study in both educational and industrial domains. Overall, the study contributes to making traffic control systems smarter, safer, and more efficient while demonstrating the practical use of Verilog and FSMs in digital design.

This study contributes to the advancement of traffic management systems by designing and implementing a fully automated traffic light controller for a four-way intersection using Verilog and FPGA technology. The key contributions of this study are:

Improved Traffic Flow: The system optimizes traffic movement by ensuring that only one direction has a green light at a time, minimizing congestion.

Enhanced Safety: The inclusion of pedestrian crossing controls ensures safer road crossings, reducing accidents.

Emergency Vehicle Priority: The system can override normal traffic sequences to allow emergency vehicles to pass without delay.

Efficient Timing Mechanism: Predefined green and yellow signal durations ensure smooth transitions between traffic phases.

Real-time Processing: The FPGA-based implementation provides high-speed processing and reliable traffic control.

Scalability and Future Enhancements: The design can be expanded to include sensor-based traffic density monitoring and IoT integration for smart city applications.

Practical Hardware Implementation: The system is deployed on the Basys3 FPGA board, demonstrating real-world feasibility and functionality.

By offering a low-latency, efficient, and reliable traffic management solution, this study contributes to the development of smarter and safer road infrastructure for urban environments.

CHAPTER 2

LITERATURE SURVEY

Traffic congestion is a growing concern in urban areas, and various traffic control methods have been proposed over the years to optimize vehicle movement at intersections. Traditional traffic light systems operate on fixed-time cycles, where each signal is assigned a predetermined duration regardless of real-time traffic conditions. While these systems are simple to implement, they often result in inefficiencies, leading to unnecessary waiting times and increased congestion, particularly during off-peak hours. Research has shown that fixed-time systems fail to adapt to varying traffic densities, making them unsuitable for modern, dynamic traffic environments.

To overcome these limitations, researchers have explored sensor-based adaptive traffic control systems. These systems utilize real-time traffic data collected from various sensors, such as inductive loops, infrared detectors, and cameras, to dynamically adjust signal durations. Studies have demonstrated that adaptive traffic control can significantly improve traffic flow by responding to changing vehicle densities. However, implementing such sensor-based systems requires a complex infrastructure, which can be costly and difficult to maintain, especially in developing regions where traffic congestion remains a significant issue.

Another widely studied approach in traffic control is the use of Finite State Machines (FSMs). FSM-based controllers ensure a well-structured and sequential transition between different traffic light states, optimizing signal changes for efficient traffic flow. FSMs provide a systematic approach to managing traffic lights by defining specific states and transitions for different road conditions. Researchers have implemented FSM-based traffic control using hardware description languages such as Verilog and VHDL, demonstrating improved efficiency over traditional fixed-time systems. However, most existing FSM-based designs focus only on basic traffic light sequences and do not integrate additional features like pedestrian signals or emergency vehicle prioritization, which are crucial for real-world traffic management.

However, many FPGA-based traffic light controllers lack emergency override mechanisms, which are essential for allowing ambulances, fire trucks, and other emergency vehicles to pass through intersections without unnecessary delays.

Emergency vehicle prioritization has been another area of research in traffic control. Some studies have proposed RFID-based and IoT-enabled solutions that detect emergency vehicles and adjust traffic lights accordingly. While effective, these solutions require additional hardware components and wireless communication infrastructure, making them complex and expensive to implement. An alternative approach is to integrate an emergency override mechanism within the FSM-based traffic controller itself, allowing seamless priority handling without requiring external devices.

Pedestrian safety is another important consideration in traffic management. Many traffic control systems do not provide dedicated pedestrian signals, increasing the risk of accidents at busy intersections. Some studies have proposed push-button pedestrian crossing systems, where pedestrians manually request a crossing signal. However, these systems require user intervention and may not always be synchronized with vehicular traffic signals. A well-designed FSM-based traffic controller can integrate pedestrian signals within the system, ensuring automatic synchronization between vehicle and pedestrian movement to improve safety.

The review of existing research highlights the need for a traffic light controller that not only manages signal transitions efficiently but also incorporates additional features such as pedestrian crossing signals and emergency vehicle prioritization. While FSM-based traffic control systems offer a structured approach, they must be enhanced to include these real-world considerations. FPGA-based implementations using Verilog provide the speed and efficiency required for modern traffic management, making them a suitable platform for developing an intelligent traffic control system.

CHAPTER 3

EXISTING SYSTEM

3.1 EXISTING METHOD:

The existing model of the automatic traffic light controller for a four-way intersection follows a fixed-time control mechanism, where traffic lights change at predetermined intervals regardless of real-time traffic conditions. This system is simple and widely implemented, but it has several drawbacks that affect traffic flow efficiency, pedestrian safety, and emergency response time.

One of the major limitations of this system is its inflexibility. Since the signal transitions are based on a predefined timer, they do not adapt to varying traffic densities. During peak hours, when traffic is high on certain roads, vehicles may experience unnecessary delays due to the fixed cycle. Conversely, during non-peak hours, roads with low or no traffic still have to wait for their turn, leading to inefficiencies. This results in unnecessary fuel consumption, increased travel time, and congestion in urban areas.

Another critical issue in the existing model is the lack of pedestrian signals. Pedestrians have to rely on vehicle signals to determine when to cross, which may not always be safe. This can increase the chances of accidents, especially at intersections with heavy traffic movement. A dedicated pedestrian crossing signal is crucial for ensuring pedestrian safety, yet the existing system does not provide this feature.

Furthermore, the system does not incorporate an emergency override mechanism, which is essential for emergency vehicles such as ambulances, fire trucks, and police vehicles. In emergency situations, these vehicles are forced to wait for the normal signal cycle to complete before they can proceed, potentially causing delays in critical response times. The absence of a priority mechanism for emergency vehicles can lead to life-threatening consequences, making the traditional model unsuitable for modern smart traffic management.

Additionally, traffic congestion is a significant challenge with fixed-time traffic signals. Since the system does not dynamically adjust to traffic flow, intersections may become clogged when vehicle density is high. This leads to longer commute times, fuel wastage, and increased air pollution, making the system inefficient for growing urban traffic demands.

To address these limitations, modern traffic management systems are moving towards adaptive and intelligent traffic control mechanisms that utilize real-time traffic data, pedestrian signals, and emergency override features. The proposed model improves upon these shortcomings by introducing pedestrian signals, emergency vehicle prioritization, and future adaptability for traffic density-based control. These enhancements aim to create a more efficient, safe, and smart traffic management system for urban intersections.

The existing traffic control system is a traditional fixed-time traffic light mechanism used at four-way intersections. In this model, the traffic lights operate on a predefined time cycle, switching between red, yellow, and green lights at fixed intervals. This system does not consider real-time traffic conditions, leading to inefficiencies such as unnecessary waiting times and traffic congestion.

One of the major drawbacks of this system is the lack of adaptability, it cannot adjust signal durations based on actual traffic density. Additionally, pedestrian safety is often overlooked, as many traditional systems do not have separate pedestrian crossing signals. Another limitation is the absence of an emergency override feature, which means emergency vehicles must wait for their turn, delaying response times in critical situations.

3.2 LIMITATIONS AND CHALLENGES:

The existing automatic traffic light controller for a four-way intersection operates on a fixed-time control mechanism without considering real-time traffic conditions. While this system is simple and easy to implement, it has several limitations and challenges that affect efficiency, safety, and adaptability.

Lack of Traffic Adaptability:

The existing model follows a predefined signal timing that does not adjust based on real-time traffic density. This results in

Unnecessary waiting times: Roads with no vehicles still follow the signal cycle, wasting time.

Traffic congestion: Roads with high vehicle flow may experience excessive delays.

Inefficient use of road capacity**:** Some lanes may remain empty while others have long queues.

Absence of Pedestrian Signals:

Pedestrians rely on vehicle signal changes to cross roads, increasing the risk of accidents. The lack of dedicated pedestrian signals results in:

Unsafe pedestrian crossings, especially in busy intersections.

Higher chances of accidents, particularly when pedestrians misjudge vehicle signals.

No Emergency Vehicle Override:

The model does not prioritize ambulances, fire trucks, or police vehicles, leading to:

Delays in emergency response times.

Increased risk to human life, as medical or rescue services cannot reach destinations quickly. Traffic blockages, as emergency vehicles get stuck in queues.

Increased Travel Time:

Due to fixed signal durations, drivers may experience:

Longer commutes, especially during peak hours.

Frequent stops and delays, reducing overall road efficiency.

No Integration with Smart Systems:

Modern cities require intelligent transportation systems (ITS), but the existing model:

Does not support sensor-based traffic detection.

Cannot be integrated with AI or IoT-based systems for real-time monitoring.Lacks flexibility to handle growing urban traffic demands.

Vulnerability to Power Failures and Hardware Malfunctions:

Unlike traditional traffic lights that rely on simple electrical circuits, an FPGA- based traffic controller depends on stable power and hardware reliability. Power failures or FPGA malfunctions could cause the system to stop functioning, leading to potential traffic hazards. Implementing backup mechanisms and fail-safe modes is necessary to ensure continuous operation during system failures.

Real-World Implementation and Deployment Challenges:

While the system can be simulated using tools like ModelSim or Xilinx Vivado, deploying it in a real-world traffic management environment requires integration with existing infrastructure. Factors such as government regulations, installation costs, and maintenance challenges need to be considered. Ensuring compatibility with current traffic systems and obtaining necessary approvals can be a time- consuming process.

Hardware Resource Limitations:

Since the system is designed using Verilog for FPGA implementation, the choice of FPGA hardware affects its performance and scalability. Low-end FPGAs may have limited logic resources, restricting the complexity of the FSM design and the number of traffic management features that can be implemented.

Additionally, hardware-based implementations require proper testing and debugging, which can be more time-consuming and expensive compared to software-based solutions.

Scalability Challenges for Larger Road Networks:

This project focuses on a four-way intersection, but expanding it to handle multiple intersections or complex road networks presents a challenge. As more intersections and traffic scenarios are added, the FSM design becomes more complex, increasing the number of states and transitions. Managing a large-scale FSM-based traffic control system requires advanced optimization techniques to maintain efficiency without excessive resource usage.

Conclusion :

The existing traffic light controller is outdated and inefficient for modern traffic conditions. The absence of adaptive signal control, pedestrian safety measures, and emergency response mechanisms makes it less effective in managing traffic efficiently. To overcome these challenges, a smart and adaptive traffic control system is needed to optimize road usage and improve overall transportation safety.

FIG 3.1: TRAFFIC SIGNALS

CHAPTER 4
DESIGN AND METHODOLOGY

4.1 PROPOSED METHOD:

The proposed model of the Automatic Traffic Light Controller for a Four-Way Intersection using Verilog is an advanced version of the traditional fixed-time traffic control system. It enhances the existing model by integrating automation, adaptability, and safety mechanisms to improve traffic management efficiency.

One of the key enhancements in the proposed system is the inclusion of pedestrian crossing signals. In traditional traffic light systems, pedestrians must rely on vehicle signal transitions to determine when to cross. However, this model introduces dedicated pedestrian signals, ensuring safer movement at intersections. When a pedestrian presses the crossing request button, the controller schedules a pedestrian green phase in the traffic light cycle. This minimizes the risk of accidents while ensuring smooth pedestrian movement without unnecessary traffic interruptions.

Another crucial feature of this model is the emergency override mechanism. Traditional traffic light systems do not prioritize emergency vehicles such as ambulances, fire trucks, or police cars, leading to delays in critical situations. The proposed model incorporates an emergency detection system that overrides normal traffic signals when an emergency vehicle is detected. In such cases, the traffic light controller immediately grants a green signal to the road with the emergency vehicle while others are switched to red, allowing quick and safe passage. Once the emergency vehicle has passed, the system resumes normal operation.

The proposed system is designed to be modular and scalable, enabling future improvements such as real-time traffic density-based control. By integrating sensors or cameras, the system could dynamically adjust signal durations based on the number of vehicles on each road. This would reduce unnecessary waiting times, prevent congestion, and improve overall traffic efficiency. Additionally, the modular architecture of the controller allows it to be adapted for larger intersections and integrated with smart city infrastructure for intelligent transportation networks.

The system is designed to be cost-effective , as it can be implemented using FPGA or ASIC- based hardware, reducing dependency on complex microcontroller-based solutions.

Simulation results indicate that the proposed model significantly optimizes traffic flow by reducing idle times and improving road utilization. Compared to traditional systems, this model achieves better safety for both pedestrians and vehicles, minimizes congestion, and ensures a smoother traffic management process. These improvements make it a smart and effective solution for urban intersections and lay the foundation for future intelligent traffic control system.

FIG 4.1: Proposed Model Of Traffic Light Controller

CHAPTER 5

SOFTWARE REQUIREMENTS & PROJECT DEVELOPMENT

For implementation of serial communication in FPGA we need to access the Xilinx VIVADO software tool and Verilog programming language.

5.1 Xilinx VIVADO:

Xilinx VIVADO Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. VIVADO represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).

Like the later versions of ISE, VIVADO includes the in-built logic simulator. VIVADO also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.The VIVADO High-Level Synthesis compiler enables C, C++ and System C programs to be directly targeted into Xilinx devices without the need to manually create RTL. VIVADO HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading.

VIVADO was introduced in April 2012, and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. VIVADO includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards- based packaging of both algorithmic and RTL IP for reuse; standards-based IP titching 26 and systems integration of all types of system building blocks; and the verification of blocks and systems. A free version Web PACK Edition of VIVADO provides designers with a limited version of the design environment.

Vivado is an FPGA design and verification tool developed by Xilinx for designing digital circuits using Verilog, VHDL, and System Verilog. It is a comprehensive software suite that enables simulation, synthesis, implementation, and debugging of FPGA-based designs. Vivado replaces older tools like ISE Design Suite and

provides a faster, more efficient workflow with advanced features such as high-level synthesis (HLS), IP integration.

Fig 5.1 VIVADO HLX Editions

VIVADO includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards-based packaging of both algorithmic and RTL IP for reuse; standards-based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.

5.2 Verilog Programming Language

Verilog is a hardware description language (HDL) that is used to describe digital systems and circuits in the form of code. It was developed by Gateway Design Automation in the mid-1980s and later acquired by Cadence Design Systems.

Verilog is widely used for design and verification of digital and mixed-signal systems, including both application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs). It supports a range of levels of abstraction, from structural to behavioral, and is used for both simulation-based design and synthesis- based design.

Verilog is a hardware description language (HDL) used for designing and simulating digital circuits. It enables engineers to describe the behavior and structure of hardware components using text-based code, making it a crucial tool in FPGA (Field-Programmable Gate Arrays) and ASIC (Application-Specific Integrated Circuits) design. Verilog supports multiple levels of abstraction, including behavioral, dataflow, and structural modeling, allowing flexibility in circuit design. One of its key features is concurrency, which mirrors the parallel nature of hardware operations. It

also supports sequential logic (such as flip-flops and registers) and combinational logic (like AND, OR, and XOR gates).

5.3 HISTORY

Before the development of Verilog, the primary hardware description language (HDL) used for digital circuit design and verification was VHDL (VHSIC Hardware Description Language). VHDL was developed in the 1980s by the U.S. Department of Defense as part of the Very High-Speed Integrated Circuit (VHSIC) program to design and test high-speed digital circuits.

VHDL is a complex language that enables designers to describe digital systems using a range of abstraction levels, from the low- level transistor and gate levels up to complex hierarchical systems. It was designed to be more descriptive and flexible than earlier HDLs, such as ABEL (Advanced Boolean Expression Language), ISP (Integrated System Synthesis Procedure), and CUPL (Compiler for Universal Programmable Logic).

Despite the development of Verilog and its increasing popularity since the 1980s, VHDL remains a widely used HDL, particularly in Europe and in the military and aerospace industries. Today, both Verilog and VHDL are widely used in digital circuit design and verification, with many companies and organizations using a combination of the two Languages.

Verilog was developed in 1984 by Phil Moorby and Prabhu Goel at Gateway Design Automation as a hardware description language (HDL) for modeling and simulating digital circuits. Initially, it was a proprietary language, but in 1990, Cadence Design Systems acquired Gateway and later opened Verilog to the public. This led to its standardization by the IEEE in 1995 as IEEE 1364-1995. Over time, Verilog evolved with enhancements, leading to the IEEE 1364-2001 and IEEE 1364-2005 versions, which introduced new features such as improved synthesis support, enhanced testbenches, and better simulation capabilities. In 2009, Verilog was merged into System Verilog (IEEE 1800-2009), making it more powerful by adding object-oriented programming and advanced verification features. Today, Verilog remains widely used for FPGA and ASIC design, serving as a fundamental language

for digital circuit development, simulation, and verification.

5.4 VIVADO PROCESS:

Create A VIVADO Project:

Vivado is an FPGA design tool developed by Xilinx for synthesis, simulation, and implementation of digital circuits. Below is a step-by-step process for designing, simulating, and implementing the Fully Automated Traffic Light Controller System using Vivado.

VIVADO “projects” are directory structures that contain all the files needed by a particular design. Some of these files are user-created source files that describe and constrain the design, but many others are system files created by VIVADO to manage the design, simulation, and implementation of projects. In a typical design, you will only be concerned with the user-created source files.

Vivado is an advanced FPGA design tool developed by Xilinx, used for designing, simulating, synthesizing, and implementing digital circuits. The process of implementing a Fully Automated Traffic Light Controller System in Vivado begins with creating a new project, selecting the appropriate FPGA board (such as Basys **3**), and adding Verilog source files. The Verilog design includes modules for handling traffic light signals, pedestrian control, and emergency conditions. Once the source files are added, a testbench is created to simulate different scenarios and validate the system’s behavior. The next step involves running behavioral simulation in Vivado, where the waveforms are analyzed to ensure the correct operation of the traffic light controller. After successful simulation, the design undergoes synthesis, which converts the Verilog code into an optimized gate-level representation. The synthesized design is then implemented onto the FPGA, where placement and routing take place to map the logic onto the hardware. Once implementation is complete, a bitstream file is generated, which is then used to program the FPGA. Finally, the programmed FPGA is tested by providing input signals through board switches and observing the output through LEDs.

But, in the future, if you need more information about your design, or if you need more precise control over certain implementation details, you can access the other files as well. When setting up a project in VIVADO, you must give the project a unique name, choose a location to store all the project files, specify the type of

project you are creating, add any pre-existing source files or constraints files (chip you are designing for. These steps are illustrated below.

START VIVADO:

In Windows, you can start VIVADO by clicking the shortcut on the desktop.

After **VIVADO** is started, the window should look similar to the picture.

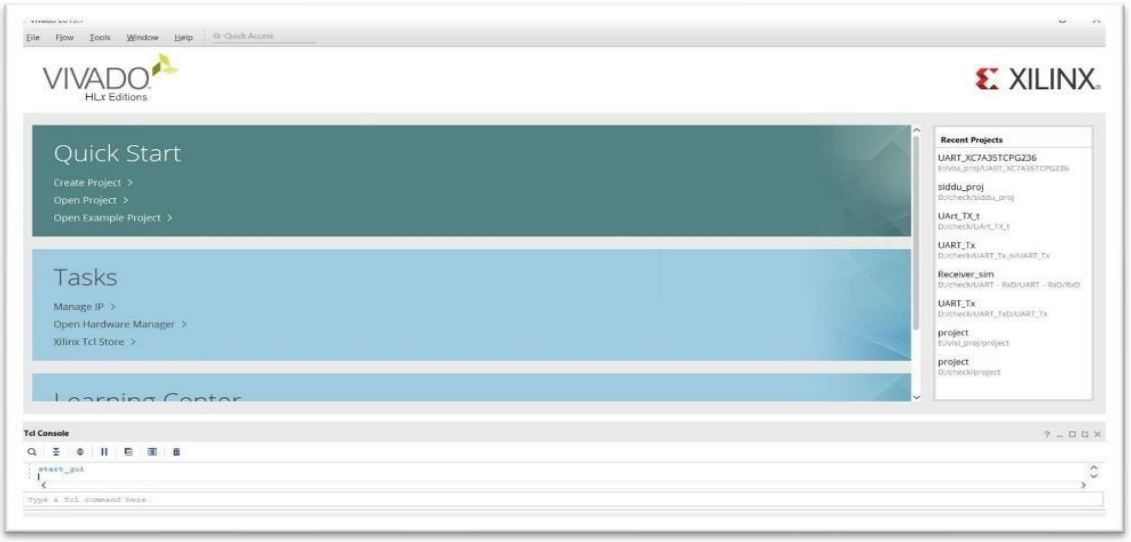

Fig 5.2: VIVADO home page

The image displays the Vivado HLx Editions startup interface, a software suite developed by Xilinx for FPGA design, simulation, and implementation. The interface consists of multiple sections, each serving a specific purpose. The Quick Start panel allows users to create new projects, open existing ones, or explore example projects for learning. The Tasks panel provides options for managing IP cores, accessing the hardware manager for FPGA programming and debugging, and utilizing the Xilinx Tcl Store for automation scripts. On the right side, the Recent Projects panel lists previously opened projects for quick access. At the bottom, the Tcl Console is available for executing Tcl commands, enabling users to automate tasks within Vivado. The presence of the start_gui command indicates that the graphical user interface has been initialized. This startup screen is the first step in FPGA design using Vivado, guiding users toward project setup, design, and hardware interaction.

1. Quick Start Panel (Left Side):

Options to Create Project, Open Project, and Open Example Project for starting a new design or resuming previous work.

2. Tasks Panel (Middle Section):

Includes options for Managing IP, accessing the Hardware Manager, and browsing the Xilinx.

Open Create Project Dialog:

In the home page you can observe a quick start, in the quick start you have to click on the create project which is look similar to the Fig 3.11.

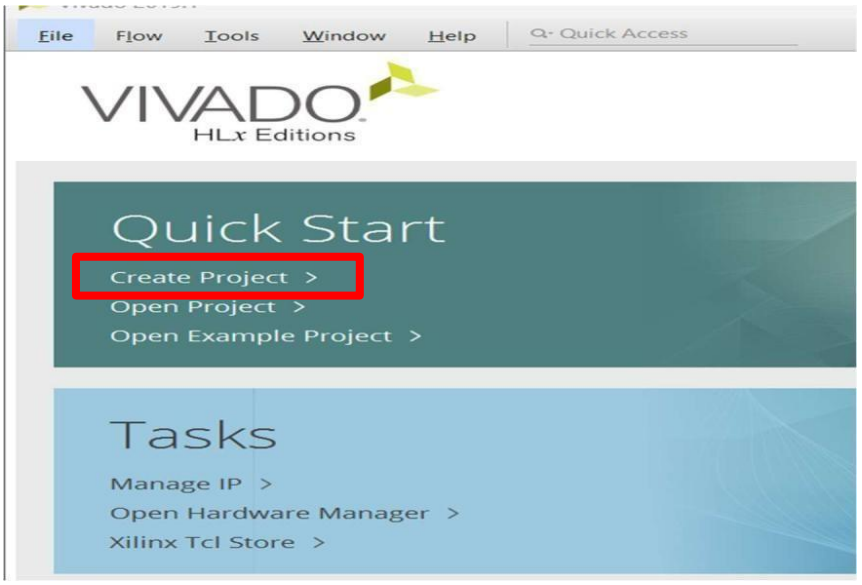

Fig 5.3: Create project set up

Click on “Create Project”, This will open the New Project dialog as shown in the figure 3.12. click on the next to continue.

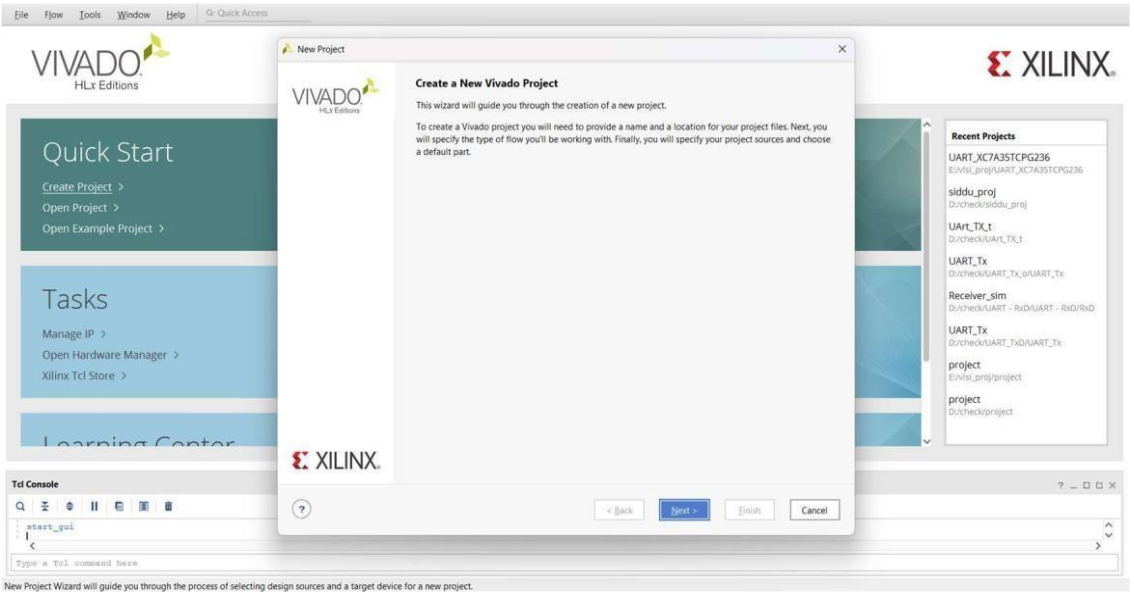

Fig 5.4: Create project dialog

Set Project Name And Location:

Enter a name for the project. In the figure, the project name is “project_1”, which isn’t a particularly useful name. It’s usually a good idea to make the project name more descriptive, so you can more readily identify your designs in the future.

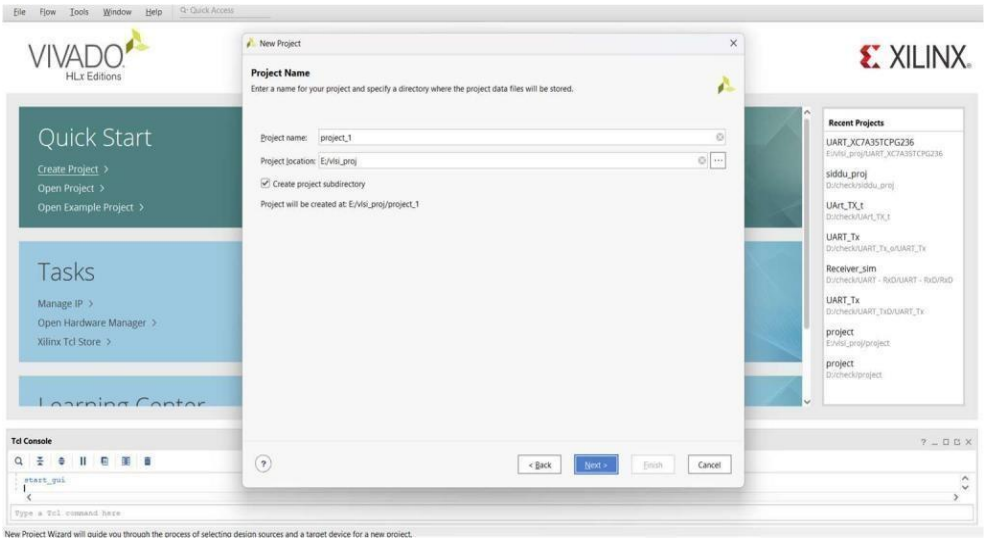

Fig 5.5: Enter project name

Key Elements in the Image:

- Vivado Main Interface (Background):
- The Quick Start panel allows creating or opening projects.
- The Recent Projects panel lists previously opened projects.
- The TCL Console at the bottom allows executing commands.

New Project Window (Foreground):

Project Name: "project_1" (user-defined name for the project).

Project Location: E:/VIV_prj/ (directory where the project files will be stored).

Create project subdirectory: Checked, meaning the final project path will be E:/VIV_prj/project_1/.

Navigation Buttons:

- "Next >" (enabled) to proceed to the next step.
- "Finish" (disabled) because further configuration is required.

The “project type” configures certain design tools and the IDE appearance based on the type of project you are intending to create. Most of the time, and for all Real Digital courses, you will choose “RTL Project” to configure the tools for the creation

of a new design. (RTL stands for Register Transfer Language, which is a term sometimes used to mean a hardware design language like Verilog)

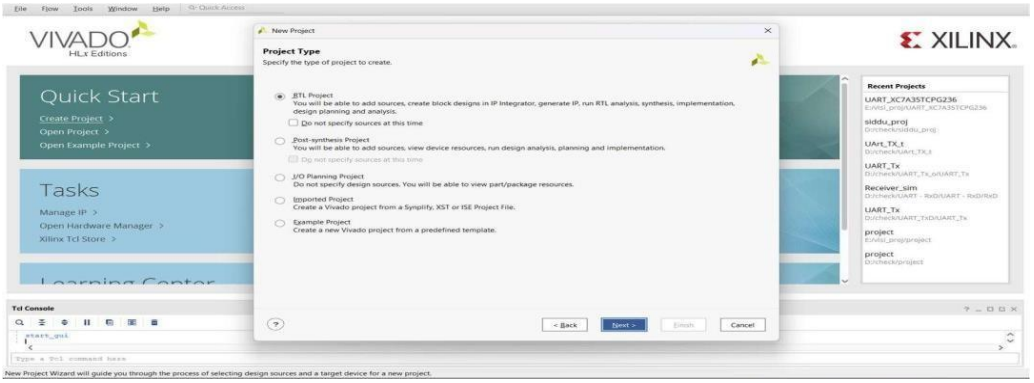

Fig5.6:select project type

RTL Project:

Description: "You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis." [1] This is the most common type of project for designing digital circuits using Hardware Description Languages (HDLs) like VHDL or Verilog, and also allows for the integration of Intellectual Property (IP) cores and the use of the IP Integrator for graphical design.

Checkbox: "Do not specify sources at this time." This option allows you to create the project structure first and add your design files later.

Post-synthesis Project:

Description: "You will be able to add sources, view device resources, run design analysis, planning and implementation." This type of project is typically used when you have already synthesized a design (possibly from another tool or a previous Vivado run) and want to perform further analysis, device assignment, and implementation.

Imported Project:

Description: "Create a Vivado project from a Synplify, XST ISE Project File." This option allows users to migrate existing projects from older Xilinx tools (ISE) or Simplify synthesis tools into the Vivado environment.

Example Project:

Description: "Create a new Vivado project from a predefined template." This option provides starting points with pre-configured settings and potentially some example designs for learning or quick prototyping.

Navigation Buttons:

< *Back:* Allows you to go back to the previous step in the New Project Wizard (if there were any).

Next >: Proceeds to the next step in the wizard based on the selected project type.

Cancel: Closes the "New Project" window without creating a new project.

Based on the highlighted radio button, the user has currently selected "RTL Project". This indicates that they intend to create a project where they will be working with RTL (Register-Transfer Level) descriptions of their hardware design, likely using VHDL or Verilog, and will utilize Vivado's full design flow, including synthesis, implementation, and bitstream generation for a Xilinx FPGA.

"Quick Start" Pane (Left Side): This is a common element in many software applications, providing quick access to the most frequent actions.

Create Project >: This button initiates the "New Project" wizard, which is currently open in the image.

Open Project >: This allows you to browse your file system and open an existing Vivado project.

Open Example Project >: This likely opens a dialog or interface to select and open pre-built example projects provided with Vivado.

Add Existing Sources:

In a typical new or early-stage design, you won't add any existing sources because you haven't created them yet. But as you complete more designs and build up a library of previously completed and known good designs, you may elect to add sources and them use them in a new design. For now, there are no existing sources to add, so just click Next.

The image shows the "Add Sources" step within the Vivado "New Project" wizard. This step appears immediately after you've selected the "RTL Project" type (as seen in the previous image). Here's a breakdown of what's visible:

1. ***Window Title and Header****:*

- "New Project": Still part of the project creation wizard.
- "Add Sources": Clearly indicates the purpose of this step.
- XILINX Logo: Identifies the software vendor.

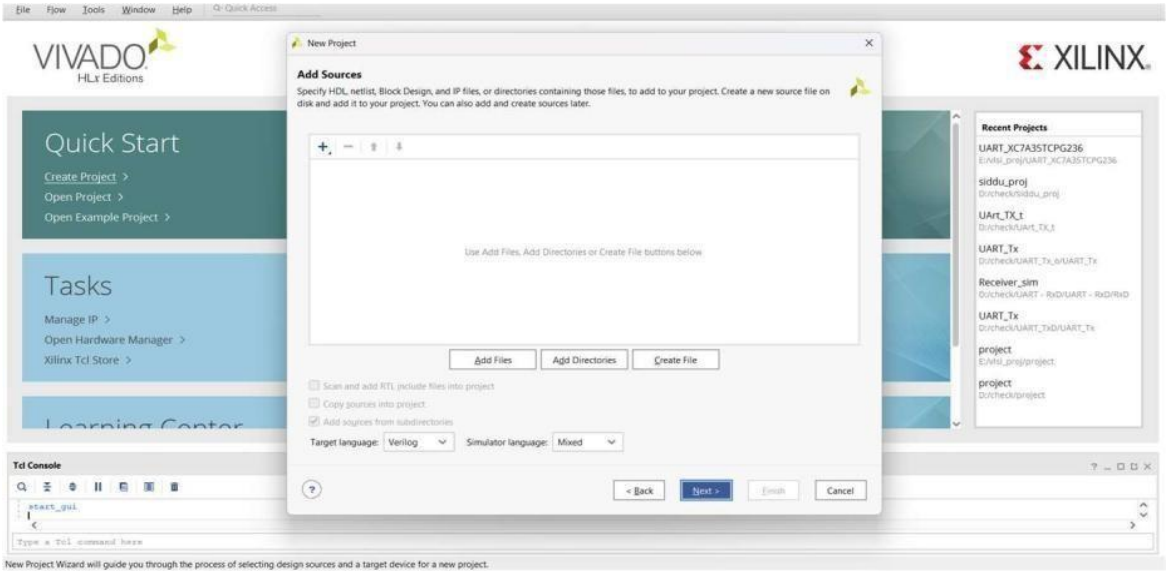

Fig 5.7: Add sources

2.Instructions: "Specify HDL, netlist, Block Design, and IP files, or directories containing these files to add to your project. Create a new source file on disk and add it to your project.

3.Source Management Area:

Empty List Box: This is where the list of source files and directories you add to the project will appear. Currently, it's empty as no sources have been added yet.

"Use Add Files, Add Directories or Create File buttons below": This is a helpful reminder of how to populate the list box.

4.Action Buttons:

+ (Add Sources): Clicking this button likely opens a file browser dialog, allowing you to select individual HDL files (.v, .vhdl), netlist files, Block Design files (.bd), or IP definition files (.xci).

- (Remove Selected Sources): This button would become active if you had selected any sources in the list box, allowing you to remove them from the project.

Up/Down Arrows: These buttons would allow you to reorder the sources in the list box, which can sometimes be relevant for compilation order (though Vivado generally handles dependencies automatically).

5.Source Addition Options:

Add Files Button: Opens a standard file selection dialog to browse and select individual source files.

Add Directories Button: Opens a directory selection dialog, allowing you to add entire directories containing your source files. Vivado will then scan these directories for relevant design files.

Create File Button: Opens a dialog where you can specify the type of file you want to create (e.g., VHDL Source File, Verilog Source File, Block Design) and its name. Vivado will then create a template file and add it to the project.

6.Project Settings (Bottom):

"Scan and add RTL include files into project" (Checkbox): If checked, Vivado will automatically search for and add any include files (e.g., .vhi, .inc) referenced by your HDL source files within the added directories.

"Copy sources into project" (Checkbox): If checked, Vivado will copy the selected source files into the project directory. If unchecked, Vivado will simply create links to the original file locations. Copying is generally recommended for project portability and to avoid issues if the original files are moved or deleted.

"Add sources from subdirectories" (Checkbox): If you add a directory, checking this option will instruct Vivado to recursively search for and add source files within any subdirectories of the selected directory.

Add Constraints:

Constraint files provide information about the physical implementation of the design. They are created by the user, and used by the synthesizer. Constraints are parameters that specify certain details about the design. As examples, some constraints identify which physical pins on the chip are to be connected to which named circuit nodes in your design; some constraints setup various physical attributes of the chip, like I/O pin drive strength (high or low current); and some constraints identify physical locations of certain circuit components. The Xilinx Design Constraints (.xdc filetype) is the file format used for describing design constraints, and you need to create an .xdc file in order to synthesize your designs for a Real Digital board. Later in this tutorial, you will create a constraints file to identify which named circuit nodes must be connected to which physical pins. But for now, you have no existing constraints file to add, so you can simply click next.

The Xilinx Design Constraints (.xdc filetype) is the file format used for describing design constraints, and you need to create an .xdc file in order to synthesize your designs for a Real Digital board. Later in this tutorial, you will create a constraints file to identify which named circuit nodes must be connected to which physical pins.

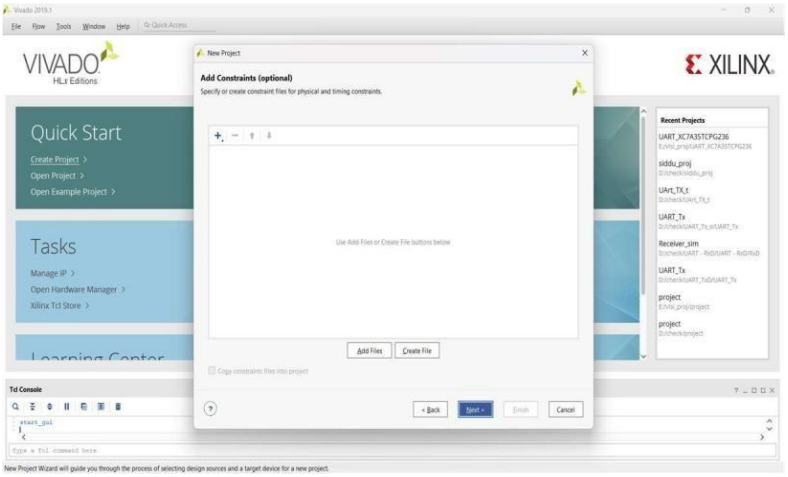

Fig 5.8: Add constraints files

Select Parts:

Xilinx produces many different parts, and the synthesizer needs to know exactly what part you are using so it can produce the correct programming file. To specify the correct part, you need to know the device family and package, and less critically, the speed and temperature grades (the speed and temperature grades only affect special- purpose simulation results, and they have no effect on the synthesizer's ability to produce accurate circuits). You must choose the appropriate part for the device installed on your board.

For example, the project uses a basys3 device with the following attributes:

Category: All

Family: Artix-7

Package: cpg236 Speed: -1

LUT's : 20800

Flip flops:41600

Select the Xilinx board for the project as xc7a35cpg236-1 which can have 50 Block RAMs, 90 DSPs. Click on next as shown in figure 5.8.

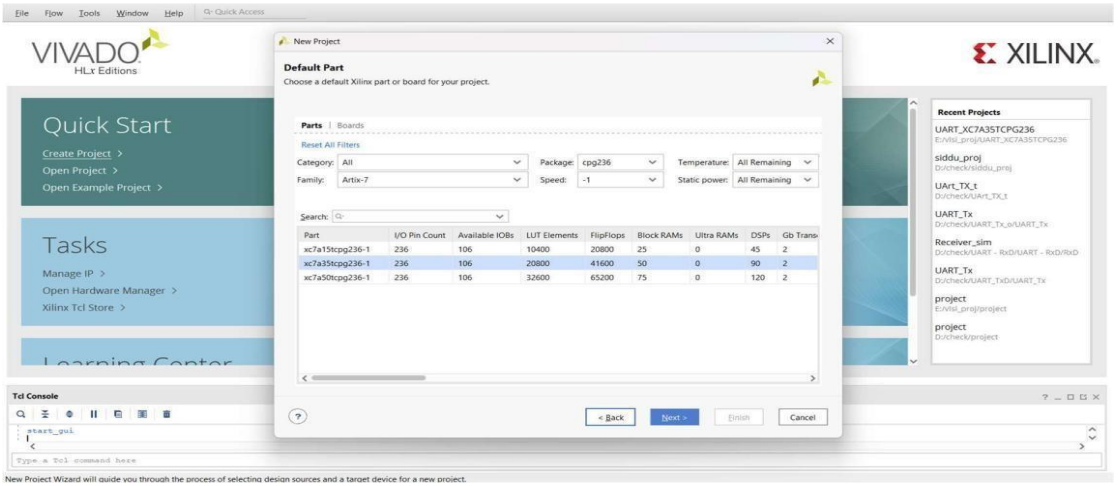

FIG 5.9: Select Artix-7 Cpg236 Board

Create Project Configuration Summary:

On the last page of the Create Project Wizard a summary of the project configuration is shown. Verify all the information in the summary is correct, and in particular make sure the correct FPGA part is selected. If anything is incorrect, click back and fix it; otherwise, click Finish to finish creating an empty project.

Fig 5.10: Create project summary

The image shows the **"**New Project Summary**"** step in the Vivado "New Project" wizard. This is the final confirmation screen before the project is actually created. It provides a summary of the choices you've made in the previous steps.

Here's a breakdown of the information presented:

1. *Window Title and Header:*

- *"New Project"*: Still part of the project creation wizard.
- *"New Project Summary":* Clearly indicates the purpose of this screen.
- *XILINX Logo*: Identifies the software vendor.

2. *Project Summary Information*:

"*A new RTL project named project_1 will be created.***"**: This confirms the type of project you selected (RTL Project) and the default name assigned to it ("project_1"). You likely had the option to change this name in an earlier step (though that step isn't shown in the current sequence of images).

"No source files or directories will be added. Use Add Sources to add **them** *later.":* This confirms that you chose not to add any source files (HDL, IP, etc.) in the "Add Sources" step. It reminds you that you can add them after the project is created.

"No constraints files will be added. Use Add Sources to add them later.": Similar to source files, this confirms that you haven't added any constraint files (XDC files for timing, pin assignments, etc.). You'll need to add these later in the design flow.

"*The default part and product family for the new project:"*: This section details the target Xilinx device for your project.

"Default Part : xc7a35tcpg236-1": Specifies the exact FPGA device that will be targeted. "xc7a35t" indicates an Artix-7 family device, "cpg236" refers to the package type, and "-1" is the speed grade.

"Product: Artix-7": Indicates the product family of the selected device.

"Family: Artix-7": Reinforces the device family.

"Package: cpg236": Specifies the physical package of the FPGA.

"Speed Grade: -1": Indicates the performance grade of the device.

3. Final Instruction:

"To create the project, click Finish.": This is the final instruction, telling you what action to take to proceed with the project creation based on the summarized settings.

4. Navigation Buttons (Bottom Right):

< *Back*: Allows you to go back to the previous steps in the wizard if you want to change any of your selections (e.g., project type, source files, target device).

Next >: This button is grayed out because this is the final step before project creation.

Finish: Clicking this button will create the Vivado project with the specified settings.

Cancel: Closes the "New Project" wizard without creating the project.

In essence, this "New Project Summary" screen gives you one last chance to review all the essential settings for your new Vivado project before it's created. It confirms the project name, the absence of initial source and constraint files, and the target FPGA device. Once you click "Finish," Vivado will create the project structure on your file system based on these settings.

VIVADO Project Window:

After you have finished with the Create Project Wizard, the main IDE window will be displayed. This is the main “working” window where you enter and simulate your Verilog code, launch the synthesizer, and program your board. The left-most pane is the flow navigator that shows all the current files in the project, and the processes you can run on those files. To the right of the flow navigator is the project manager window where you enter source code, view simulation data, and interact with your design. The console window across the bottom shows a running status log. Over the next few projects, you will interact with all of the panels.

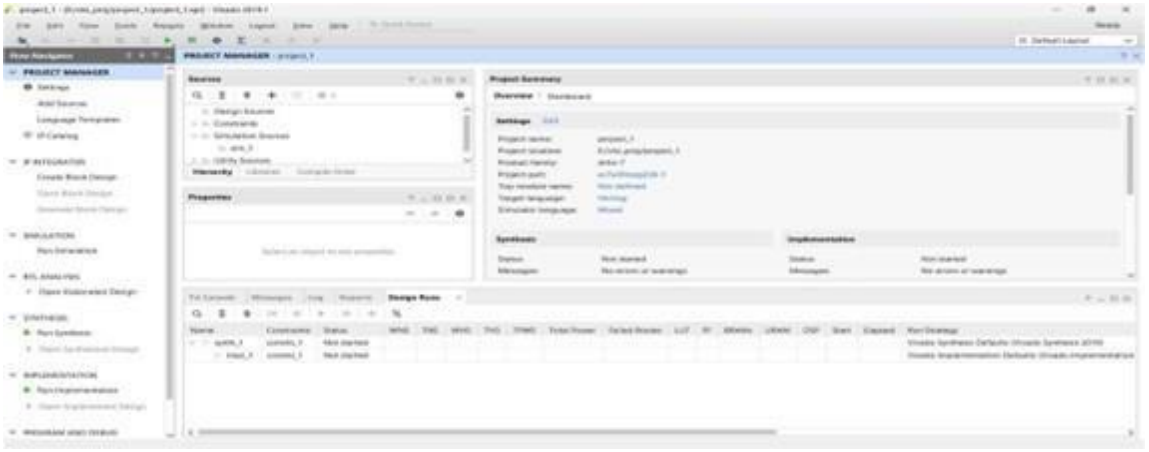

Fig 5.11: VIVADO project window

The image shows the main working environment of the Xilinx Vivado HLx Editions software after a new project named "project_1" has been created. Let's break down the different sections and their functionalities:

1. Title Bar:

- "project_1 - [c:/viv_proj/project_1/project_1.xpr] - Vivado 2019.1": This indicates the currently open project name ("project_1"), its location on the file system ("c:/viv_proj/project_1/project_1.xpr" - the .xpr file is the main Vivado project file), and the Vivado version being used (2019.1).

***2. Menu Bar**:*

- File, Edit, Flow, Tools, Reports, Window, Help: These are standard application menus providing access to various Vivado functionalities, such as project management, editing, design flow control (synthesis, implementation), accessing tools (IP Integrator, Logic Analyzer), generating reports, managing windows, and accessing documentation.

3. Toolbar:

- Contains icons providing quick access to frequently used commands from the menu bar. (The specific icons are not clearly distinguishable in the image).

4. Flow Navigator (Left Pane):

- This is a hierarchical view that guides you through the typical FPGA design flow in Vivado.

Project Manager:

Settings: Allows you to configure project settings like target device, language, and simulation options.

Add Sources: Used to add your design files (HDL, IP, Block Designs), constraint files (XDC), and simulation sources.

Language Templates: Provides templates for different HDL constructs.

IP Catalog: Opens the library of Xilinx and third-party Intellectual Property (IP) cores that can be integrated into your design.

IP Integrator:

Create Block Design: Launches the graphical IP Integrator tool for creating system-on-a-chip (SoC) designs by connecting IP blocks.

Open Block Design: Opens an existing Block Design.

Generate Block Design: Generates the underlying HDL code for a Block Design.

SIMULATION:

Run Simulation: Allows you to simulate your design using various simulators. RTL

ANALYSIS:

Open Elaborated Design: Opens a schematic view of your design after it has been elaborated (parsed and analyzed).

SYNTHESIS:

Run Synthesis: Executes the synthesis process, which translates your HDL code into a netlist of logic gates.

Open Synthesized Design: Opens a schematic view of the synthesized design.

IMPLEMENTATION:

Run Implementation: Executes the implementation process, which includes logic optimization, placement of logic elements, and routing of interconnections on the target FPGA device.

Open Implemented Design: Opens a view of the placed and routed design on the FPGA fabric.

Program And Debug: (Partially visible) Contains options for generating the bitstream file and programming it onto the target FPGA, as well as debugging tools.

5. Project Manager Pane (Center Top):

This pane provides a more detailed view of the project's structure and settings.

Sources: Displays the design sources (HDL files, Block Designs, IP instances), constraint files, and simulation sources organized in a hierarchical manner. Currently, it shows the basic categories: "Design Sources," "Constraints," and "Simulation Sources," with no files added yet.

Hierarchy: Shows the hierarchical structure of your design (once sources are added).

Libraries: Displays the available libraries for different design elements.

Compile Order: Shows the order in which the source files will be compiled.

Properties: When an item is selected in the "Sources" pane, its properties are displayed here. Currently, it says "Select an object to see properties."

6. Project Summary Pane (Center Right):

Provides an overview of the current project settings and the status of the design flow.

Overview: Shows basic project information

Project name: project_1

Project location: c:/viv_proj/project_1

Product family: Artix-7

Project part: xc7a35tcpg236-1 (the specific FPGA device)

Top module name: Not defined yet (this will be the top-level entity of your design).

Target language: Verilog (selected during project creation).

Simulator language: Mixed (selected during project creation).

Synthesis: Shows the status of the synthesis process (currently "Not started") and any messages or warnings.

Implementation: Shows the status of the implementation process (currently "Not started") and any messages or warnings.

7. Tcl Console/Messages/Log/Reports/Design Runs Pane (Bottom):

This is a multi-tabbed pane that provides important feedback and control.

Tcl Console: Allows you to enter and execute Tcl commands to interact with Vivado.

Messages: Displays informational messages, warnings, and errors generated during various Vivado operations.

Log: Shows a detailed log of the processes that have been run.

Reports: Provides access to various reports generated by Vivado (e.g., utilization reports, timing reports).

Design Runs: Manages the different synthesis and implementation runs you might perform. The table shows the status and settings of the "synth_1" (synthesis) and "impl_1" (implementation) runs, as well as the "constrs_1" (constraints) set.

8. Edit The Project - Create Source Files:

All projects require at least two types of source files — an HDL file (Verilog or VHDL) to describe the circuit, and a constraints file to provide the synthesizer with the information it needs to map your circuit into the target chip. This tutorial presents the steps required to implement a Verilog circuit on your Real Digital board: first, a Verilog source file is created to define the circuits behavior (again, for this tutorial, you can simply copy or download the completed file rather than typing it); second, a constraints files is created to define how the Verilog circuit is mapped into the Xilinx logic device (again, copied or downloaded for this tutorial); third, the Verilog source file and constraints file are synthesized into a ".bit" file that can be programmed onto your board; and fourth, the device is configured with the circuit.

After the Verilog source file is created, it can be directly simulated. Simulation (discussed in more detail later) lets you work with a computer model of a circuit, so you can check its behavior before taking the time to implement it in a physical device. The simulator lets you drive all the circuit inputs with varying patterns over time, and to check that the outputs behave as expected under all conditions.After the constraint file is created, the design can be synthesized. The synthesis process translates Verilog source code into logical operations, and it uses the constraints file to map the logical operations into a given chip. In particular (for our needs here), the constraints file defines which Verilog circuit nodes are attached to which pins on the Xilinx chip package, and therefore, which circuit nodes are attached to which physical devices on your board. The synthesis process creates a ". bit" file that can be directly programmed into the Xilinx chip.

In this first tutorial, the Verilog and constraint source files are provided for you. Instead of creating them yourself as would normally be the case, you can simply copy them into empty source files, or download them and include them in your project directly. In later designs, you will create these files yourself.

Design Source:

There are many ways to define a logic circuit, and many types of source files including VHDL, Verilog, EDIF and NGC netlists, DCP checkpoint files, TCL scripts, System C files, and many others. We will use the Verilog language in this course,and introduce it gradually over the first several projects. For now, you can get familiar with some of the basic concepts by reading the following.

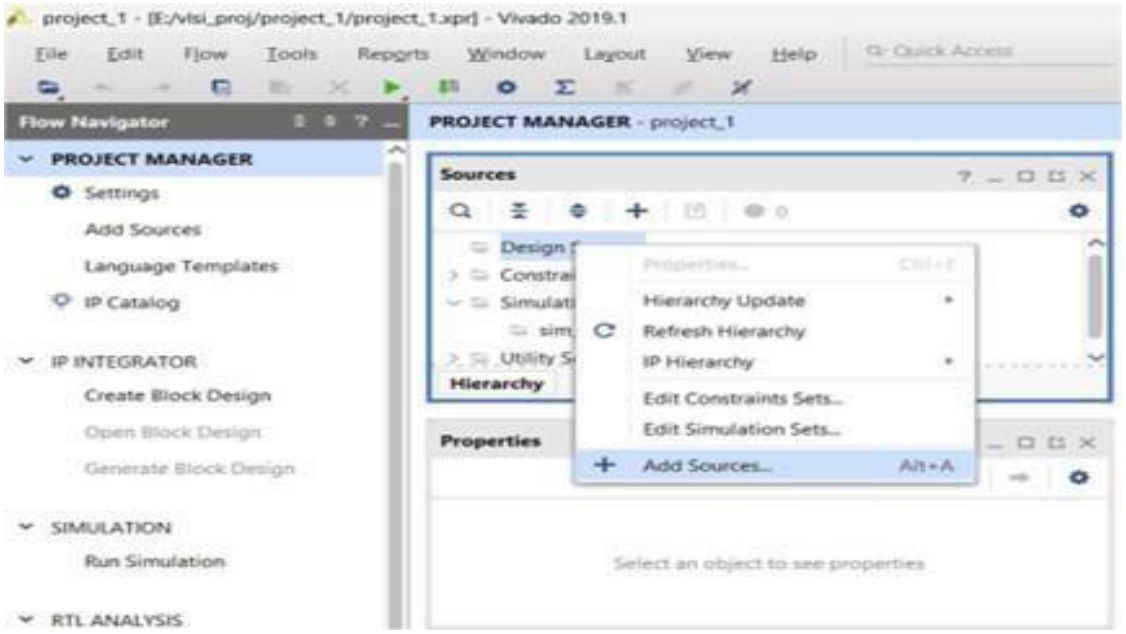

Fig 5.12: Add design source

The image shows a zoomed-in view of the "Sources" tab within the "Project Manager" pane in the Xilinx Vivado HLx Editions software. A right-click context menu has been opened.

Here's a breakdown of the elements:

1. ***"Sources" Tab:***

This tab, located within the "Project Manager," is where you manage the different types of files that make up your hardware design project. These include:

Design Sources: Your HDL files (Verilog, VHDL, SystemVerilog), Block Designs created with IP Integrator, and IP instances.

Constraints: Files (XDC) that specify timing constraints, pin assignments, and other physical implementation requirements.

Simulation Sources: Files used for simulating your design, such as testbenches.

Utility Sources: Other supporting files.

2.Hierarchy View:

The left portion of the "Sources" tab displays a hierarchical tree structure of your project's sources. Currently, it shows the top-level categories ("Design Sources," "Constraints," "Simulation Sources," "Utility Sources") with no files added yet.

3.Toolbar (Above Hierarchy):

Contains icons for common actions related to managing sources:

Search (Magnifying Glass): Allows you to search for specific files within your project sources.

Filter (Funnel): Enables you to filter the displayed sources based on certain criteria.

Add Sources (+ Icon): This is the action that has triggered the context menu. Clicking this icon (or right-clicking) allows you to add various types of sources to your project.

Add Directories (Folder Icon with +): Allows you to add entire directories containing source files.

Create File (Document Icon with +): Enables you to create new source files (HDL, constraints, etc.) directly within Vivado.

Refresh Hierarchy (Circular Arrows): Updates the source hierarchy view to reflect any changes made to the project files.

Settings (Gear Icon): Opens settings related to source management.

4.Right-Click Context Menu:

This menu appears when you right-click within the "Sources" tab (likely on one of the top-level categories like "Design Sources" or in the empty space). The options available in this menu provide quick access to common source management tasks: Properties... (Ctrl+E): Opens the properties window for the selected source or category, allowing you to view and modify its attributes.

Hierarchy Update: Contains a submenu (indicated by the right-pointing arrow) with options for updating the source hierarchy.

Refresh Hierarchy: Same as the toolbar icon, updates the source hierarchy view. IP Hierarchy: Opens a view specifically for the hierarchy of IP instances in your design (if any).

Edit Constraints Sets: Opens a dialog for managing different constraint sets in your project.

Edit Simulation Sets: Opens a dialog for managing different simulation sets.
Add Sources... (Alt+A): This is the same functionality as the "+" icon in the toolbar. Selecting this option will open the "Add Sources" wizard, allowing you to add design files, IP, Block Designs, etc., to your project.

To create a Verilog source file for your project, right-click on “Design Sources” in the Sources panel, and select Add Sources. The Add Sources dialog box will appear as shown — select “Add or create design sources” and click next.

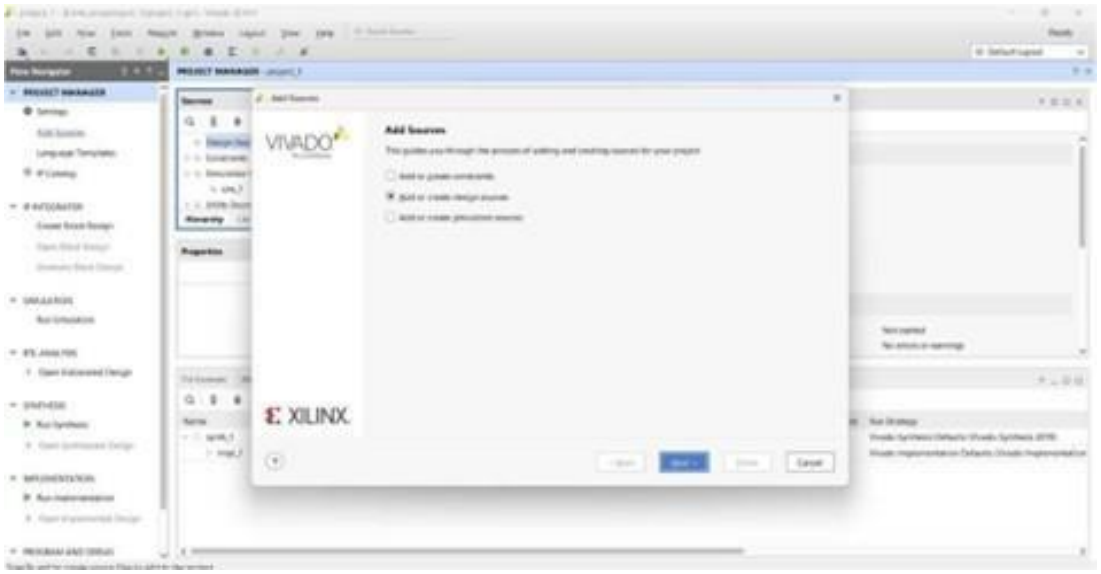

Fig 5.13: Add or create design sources window

1. Window Title and Header:

"Add Sources": Clearly indicates the purpose of this window. VIVADO HLx Editions Logo: Identifies the software.

2. Instructions:

"This guides you through the process of adding and creating sources for your project.": A brief introductory message.

3. Source Type Selection:

You are presented with three radio button options, allowing you to choose the type of sources you want to add or create:

Add or create constraints: Selecting this option will guide you through adding or creating constraint files (XDC files) for your design. These files specify timing requirements, pin assignments, and other physical constraints.

Add or create design sources (Selected): This option is currently selected. It will guide you through adding or creating your core design files, such as HDL files (Verilog, VHDL, SystemVerilog), IP cores, and Block Designs created with the IP Integrator.

Add or create simulation sources: Selecting this option will guide you through adding or creating files used for simulating your design, such as testbenches and simulation scripts.

4. Navigation Buttons (Bottom Right):

< *Back*: Allows you to go back to the previous step in the "New Project" wizard (if applicable, though in this context, it might go back to the "Sources" pane without this specific dialog).

Next >: This button is active (blue) because a source type has been selected. Clicking this will proceed to the next step based on your selection (in this case, adding or creating design sources).

Cancel: Closes the "Add Sources" dialog without adding any sources at this time.

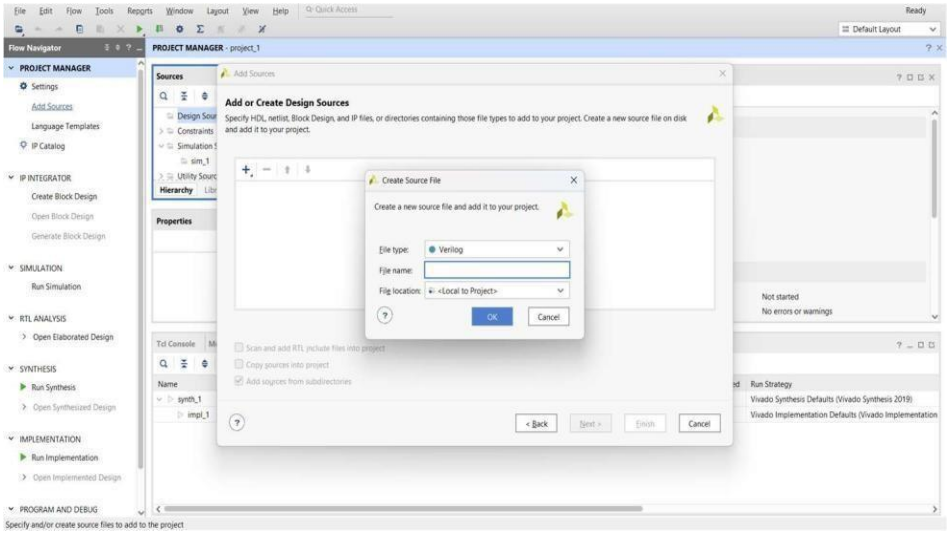

Fig 5.14: Create design source file

1.Add or Create Design Sources" Dialog (Background):

Title: "Add or Create Design Sources"

Instructions: "Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project."

Source List Area: This area is currently empty as no files have been added yet. The buttons above (+, -, etc.) would be used to add existing files or directories.

"Create File" Button: This button has been clicked, which has opened the "Create Source File" window on top.

Options (Below Source List):

"Scan and add RTL include files into project" (Checkbox) "Copy sources into project" (Checkbox)

"Add sources from subdirectories" (Checkbox)

Navigation Buttons (Bottom Right): < Back, Next >, Cancel. The Next > button is likely disabled until a source file is added or created.

1. "Create Source File" Window (Foreground):

Title: "Create Source File"

Instruction: "Create a new source file and add it to your project."

File Type Dropdown: Currently set to "Verilog". This indicates that the user intends to create a new Verilog HDL source file. Other options in the dropdown would likely include VHDL, SystemVerilog, etc.

File Name Text Box: This is where the user needs to enter the desired name for the new source file (without the extension, as Vivado will add it based on the selected file type).

File Location Dropdown: Currently set to "<Local to Project>". This means the new source file will be created within the project directory structure. You might have other options here to specify a different location.

Question Mark Icon: Likely provides help or information related to creating source files.

Action Buttons:

OK: Clicking this button will create the new Verilog file with the specified name in the chosen location and add it to the "Design Sources" in your project. The "Create Source File" window will then close.

Cancel: Clicking this button will close the "Create Source File" window without creating a new file.

The image captures a pivotal moment in the Xilinx Vivado HLx Editions software workflow, specifically within the "Project Manager" pane. The user is in the process of adding design sources to their newly created project. The "Sources" tab is active, displaying a hierarchical view of potential source categories, currently empty as no files have been added. To initiate the source addition, the user has clicked the "+" icon located in the toolbar above the source hierarchy. This action has triggered a context menu, partially visible, offering options such as "Add Files...", "Add

Directories...", "Create File...", and "Import Sources...". The user has selected "Create File...", which has subsequently opened the "Create Source File" window in the foreground. This window prompts the user to define the new source file. The "File Type" dropdown is set to "Verilog," indicating the user's intention to create a Verilog Hardware Description Language file. The "File Name" text box is currently empty, awaiting the user to provide a name for the new source file. The "File Location" is set to "<Local to Project>," signifying that the file will be created within the project's directory structure. Once the user enters a desired file name and clicks the "OK" button in the "Create Source File" window, Vivado will generate a new Verilog file with that name in the specified location and automatically add it to the "Design Sources" section of the project. The "Add or Create Design Sources" dialog, visible in the background, will remain open, allowing the user to continue adding existing files, directories, or creating further new source files as needed to build their hardware design.

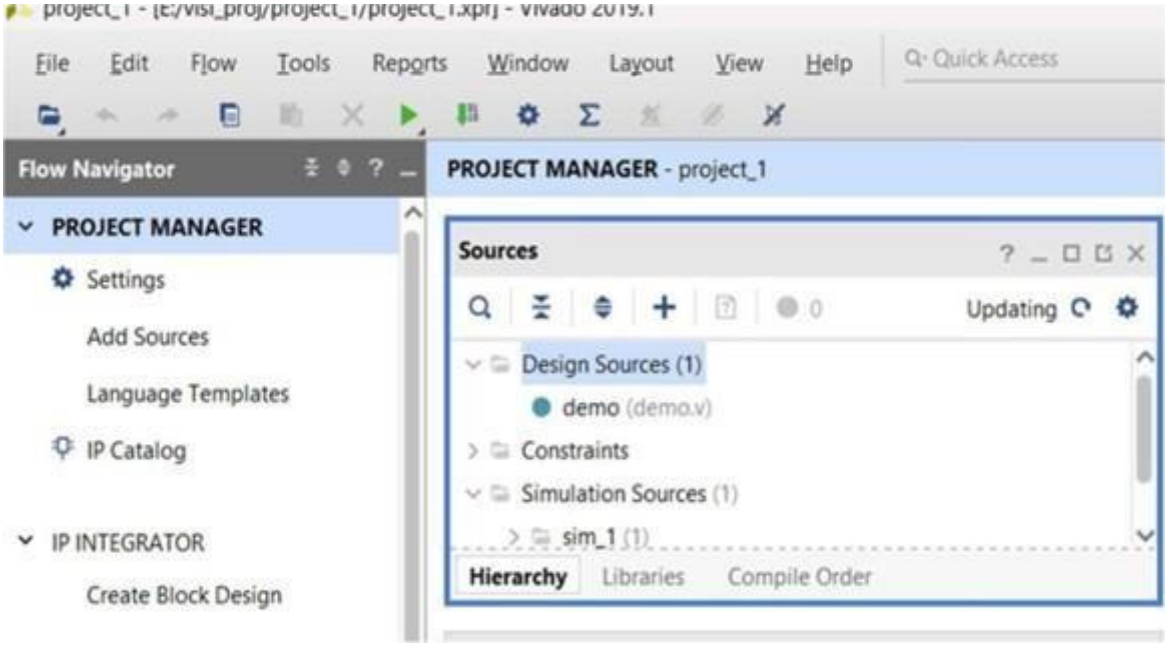

Fig 5.15: Demo file created dialog

1."Sources" Tab:

Hierarchy View:

Design Sources (1): This category is now expanded, indicated by the down arrow. The "(1)" signifies that there is one file within this category.

demo (demo.v): This is the newly created Verilog source file. The name "demo" is likely what the user entered in the "Create Source File" window, and ".v" indicates it's a Verilog file.

Constraints: This category is collapsed (indicated by the right arrow), meaning no constraint files have been added yet.

Simulation Sources (1): This category is expanded, indicating one simulation source set.

sim_1 (1): This is a simulation source set, likely created by default or by the user. The "(1)" suggests there is one file within this set (though the specific file name isn't fully visible).

Utility Sources: This category is collapsed, indicating no utility sources have been added.

Design Constraints:

Design sources, like Verilog HDL files, only describe circuit behavior. You must also provide a constraints file to map your design into the physical chip and board you are working with.

To create a constraint file, expand the Constraints heading in the Sources panel, right- click on constrs_1, and select Add Sources.

An Add Sources dialog will appear as shown. Select Add or Create Constraints and click Next to cause the “Add or Create Constraints” dialog box to appear.

Click on Create File, enter project1 for the filename and click OK. The newly created file will appear in the list as shown.

Click Finish to move to the next step. You will now see project1.xdc listed underneath Constraints/constrs_1 folder in the Sources panel. Double click project1.xdc to open the file, and replace the contents with the code below.

Alternative: Download and Add Constraints

Instead of creating an empty constraint file and using copy-and-paste to replace the contents as described above, you can also download the Project1.Xdc file and add it to your project using the Add Files button in Add Sources dialog.

The provided image from Xilinx Vivado illustrates the initial stages of a hardware design project named "project_1". Within the "Project Manager" pane, the "Sources" tab is currently displaying the project's source file hierarchy. Under the "Design Sources" category, a Verilog file named "demo.v" has been successfully added, indicated by the expanded category and the listed filename. Additionally, the "Simulation Sources" category is also expanded, revealing a simulation source set labeled "sim_1," which contains one file. Notably, the "Constraints" category is present in the hierarchy but remains collapsed, signifying that no design constraint files have been incorporated into the project at this juncture. In the upper right corner

of the "Sources" tab, an "Updating..." indicator with a spinning circle suggests that Vivado is actively processing the recent addition of the "demo.v" file and synchronizing the project view. This snapshot reflects the project's composition in terms of design and simulation sources, while highlighting the absence of any defined design constraints, which would typically be added as separate .xdc files under the "Constraints" category to guide the synthesis and implementation processes for the target FPGA device.

Code Bit Generation:

To implement the code in board we need to complete the following processes

The bitstream (bit) file generation process is essential for programming an FPGA. Here's a step-by-step breakdown of how it works, especially for the Basys3 FPGA (Xilinx FPGA using Vivado):

1. Writing the Verilog Code

Develop your Verilog HDL design, including all necessary modules. Create a testbench to verify the functionality through simulation.

2. Simulation (Optional but Recommended)

Use the Vivado Simulator to test your design before implementation. Run behavioral simulation to check logic correctness.

3. Synthesis

Convert Verilog code into gate-level netlist.

The tool maps the design to the available FPGA logic resources.

4. Implementation

Perform Placement (assigning logic to specific FPGA regions). Perform Routing (connecting logic elements with wires).

5. Bitstream Generation

Convert the implemented design into a bit file.

This file contains the configuration data required to program the FPGA

Synthesis:

After your Verilog and constraint files are complete, you can Synthesize the design project. In the synthesis process, Verilog code is translated into a "netlist" that defines all the required circuit components needed by the design (these components are the programmable parts of the targeted logic device - more on that later). You can start the Synthesize process by clicking on Run Synthesis button in the Flow Navigator panel as shown.When synthesis is running, you can select the log panel located at the bottom of Project Manager to see a log of the currently running

processes. Any errors that occur during the synthesis process will be described in the log.

Implementation:

After the design is synthesized, you must run the Implementation process. The implementation process maps the synthesized design onto the Xilinx chip targeted by the design. Click the Run Implementation button in the Flow Navigator panel as shown. When the implementation process is running, the log panel at the bottom of Project Manager will show details about any errors that occur.

Generate Bitstream

After the design is successfully implemented, you can create a .bit file by clicking on the Generate Bitstream process located in the Flow Navigator panel as shown. The process translates the implemented design into a bitstream which can be directly programmed into your board's device.

Code Dumping On Board:

Open Hardware Manager:

After the bitstream is successfully generated, you can program your board using the Hardware Manager. Click Open Hardware Manager located at the bottom of Flow Navigator panel, as highlighted in red in the Figure.

Connecting Your Board Via USB:

Connect your Blackboard to your Computer with a micro-USB cable. Make sure you connect the micro-USB cable to the port labeled "PROG UART". Turn on your board by moving the switch in the top-left corner to the on position. You'll see a red LED light up by the switch when it powers on. If your board doesn't power on, check that the blue jumper by the port labeled "EXTP" is set to "USB".

Connecting Your Board To VIVADO

Click on Open target link underneath Hardware Manager. Select Auto Connect to automatically identify your board.

Verify That Your Board Is Connected.

Steps to Verify Board Connection:

1. **Connect the Board:**

Use a micro USB cable to connect the Basys3 board to your computer. Ensure the cable supports both power and data transfer (not just charging).

2. **Power On the Board**:

   Set the power switch (SW7) to ON.

   The power LED (LD13) should light up.

3. **Check Device Manager (Windows)**

   Open Device Manager (Win + X → Device Manager).

   Look under Universal Serial Bus controllers or Ports (COM & LPT). You should see "Digilent USB Device" or similar.

   If not, try reinstalling Digilent drivers.

4. **Use Vivado Hardware Manager**

   Open Vivado.

   Click "Open Hardware Manager" → "Open Target" → "Auto Connect". Your board should appear under the Hardware section.

5. **Run a Basic Test**

   If the board is detected, try programming a simple LED blink design to confirm functionality.

Software Tools Used:

The software tools used for designing and testing the traffic light controller include:

Verilog: Programming language to design the logic for the traffic light controller.

Model Sim: A simulation tool to test and verify the Verilog code.

Vivado: Used for simulation, synthesis, and implementation of the design on FPGA hardware.

Quartus: An alternative FPGA design tool for testing and implementation.

Waveform Viewer**:** Built into simulation tools to visualize signal transitions and debug the design.

Clock Simulation: Software tools or Verilog constructs to simulate clock and timing signals.

Testbenches: Scripts written in Verilog to simulate different traffic scenarios and validate the controller logic.

Integrated Debugging Tools: Available in tools like Vivado for identifying and fixing errors.***Logic Analyzer:*** Virtual or built-in tools to observe the system's behavior during simulation.

Version Control Software: Tools like Git to manage code changes and collaboration.

Xilinx: This identifies the software vendor. Xilinx (now part of AMD) is a leading provider of FPGA and adaptive SoC technology.

Vivado: This is the name of Xilinx's integrated design environment (IDE). It's a comprehensive suite of tools designed for the entire FPGA development flow, from initial design entry to bitstream generation and hardware debugging. Vivado superseded Xilinx ISE (Integrated Synthesis Environment) as the primary toolchain for newer Xilinx devices.

HLx Editions: This denotes a specific version or set of features within the Vivado product line. The "HLx" likely refers to High-Level Synthesis and other advanced capabilities aimed at improving design productivity. Different editions of Vivado might offer varying levels of features and device support.

5.5 STATE DIAGRAM:

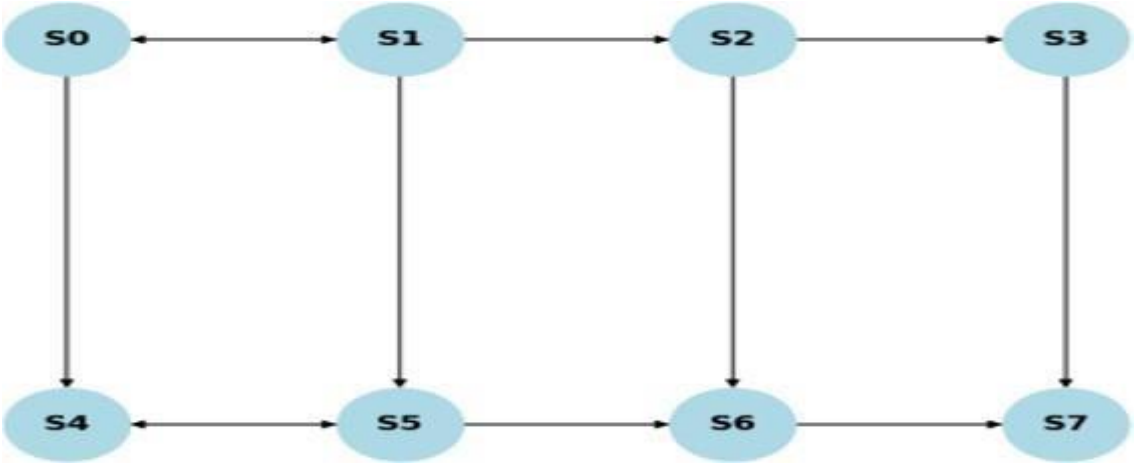

Fig 5.16: State Diagram

State Assignments & Transitions:

S0 (NS Green, EW Red): North-South (NS) traffic has a green light. EastWest (EW) traffic has a red light. Transitions to S1 after a fixed time.

S1 (NS Yellow, EW Red): NS light turns yellow, warning vehicles to stop. EastWest remains red. Transitions to S2 after a short delay.

S2 (NS Red, EW Green): NS light turns red. EastWest traffic gets a green light to move. Transitions to S3 after a fixed time

S3 (NS Red, EW Yellow): EW light turns yellow, warning vehicles to stop. North South remains red. Transitions to S4 after a short delay.

S4 (Pedestrian Walk): All vehicle signals are red. Pedestrian crossing light is on for a specific direction. Transitions to S5 after pedestrian time ends.

S5 (Pedestrian Walk for Opposite Side): Pedestrian crossing is allowed for the other direction. Vehicles remain stopped. Transitions to S6 after pedestrian time ends.
S6 (Normal Cycle Restart — Back to S0): Resets back to normal operation (S0). Alternates between NS/EW traffic as per the cycle.
S7 (Emergency Mode – Back to S0): If an emergency is detected, all normal cycles stop. The emergency direction gets a green light, others red. Once the emergency clears, the system.

5.6 STATE TABLE:

Table 1:Truth Table of State Diagram

State	clk	rst	emergency	pedestrian_NS	pedestrian_EW	NS_light[2:0]	EW_light[2:0]	NS_ped[1:0]	EW_ped[1:0]	Next State
S0 (NS Green, EW Red)	1	0	0	0	0	001 (Green)	100 (Red)	00	00	S1
S1 (NS Yellow, EW Red)	1	0	0	0	0	010 (Yellow)	100 (Red)	00	00	S2
S2 (NS Red, EW Green)	1	0	0	0	0	100 (Red)	001 (Green)	00	00	S3
S3 (NS Red, EW Yellow)	1	0	0	0	0	100 (Red)	010 (Yellow)	00	00	S4
S4 (Pedestrian Walk NS)	1	0	0	1	0	100 (Red)	100 (Red)	01 (Walk)	00	S5
S5 (Pedestrian Walk EW)	1	0	0	0	1	100 (Red)	100 (Red)	00	01 (Walk)	S6
S6 (Restart Cycle)	1	0	0	0	0	001 (Green)	100 (Red)	00	00	S0
S7 (Emergency Mode)	1	0	1	X	X	100 (Red)	100 (Red)	00	00	S0 (after clearing emergency)

In each of the states, if MSB is 1, it indicates that the light display is showing red signal, and so, the movement of vehicles in the direction corresponding to the light display is restricted. Similarly, if the middle bit is 1, it corresponds to yellow signal, and indicates that the traffic flow will stop soon. If LSB is 1, it corresponds to green signal and the flow of traffic in the corresponding direction is allowed. The odd states act as 'safe' states as few of the light displays show yellow signal indicating that the flow of traffic will be stopped soon, so that vehicles from those directions can stop, since crossing the intersection during the period of the yellow signal of the current state may lead to accidents due to the flow of traffic regulated during the succeeding even state.

5.7 SOFTWARE ENVIRONMENT:

Fig 5.17: Software Logo

- Vivado Design Suite is a software platform by Xilinx.
- Used for designing, simulating, and implementing digital circuits on FPGA.
- It supports coding in Verilog, VHDL, and provides tools to simulate designs, synthesize them into hardware, and generate bitstreams to program FPGA devices.
- Vivado includes a graphical interface, debugging tools like the Integrated Logic Analyzer (ILA), and a library of pre-built modules (IP cores) to simplify design.
- It also offers timing and power analysis to ensure optimized performance.
- Vivado is ideal for creating efficient hardware systems, ranging from simple circuits to advanced embedded designs.

Vivado is an FPGA design suite developed by Xilinx (now AMD) for designing, simulating, synthesizing, implementing, and programming FPGA devices, including the Basys3 (Artix-7) board. The design process in Vivado begins with creating a new project, selecting the appropriate FPGA board (Basys3: XC7A35T-1CPG236C), and adding Verilog or VHDL source files. Once the design is added, users can run behavioral simulation to verify functionality before synthesis. The synthesis process translates the HDL code into a gate-level netlist, mapping the logic to FPGA resources. After synthesis, implementation takes place, which involves placement and routing, ensuring the design fits within the FPGA's architecture efficiently.

CHAPTER-6

ANALYSIS ON SIMULATION& IMPLEMENTATION

6.1 SIMULATION RESULTS:

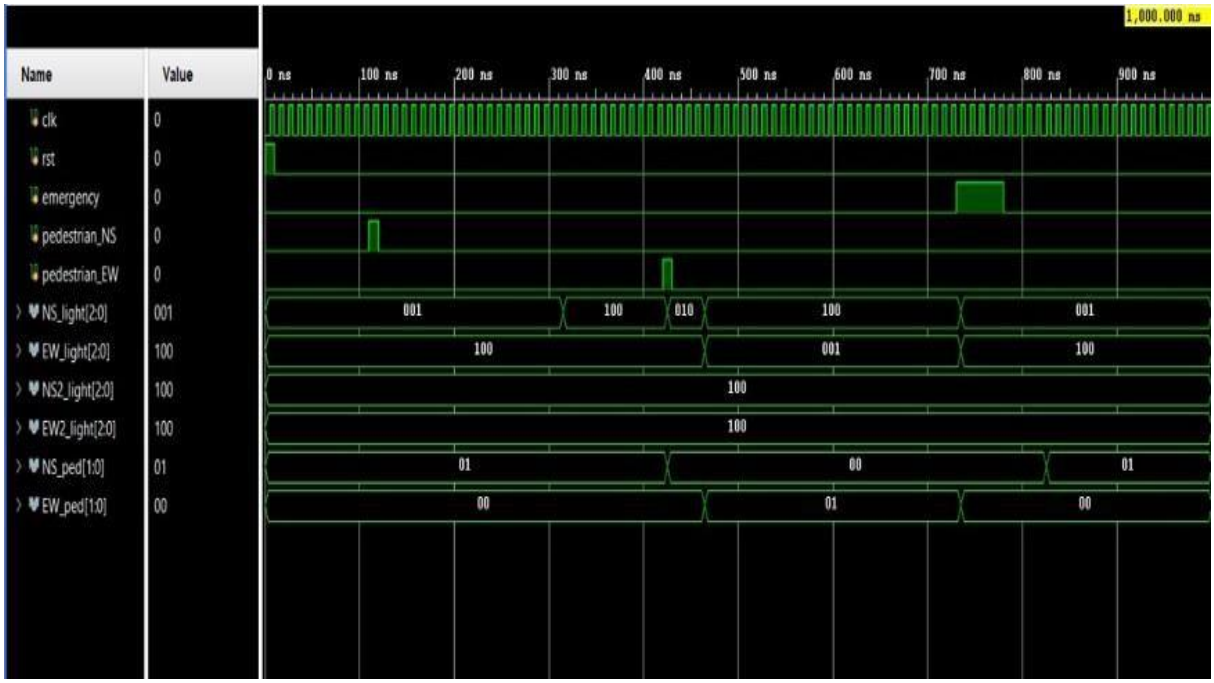

FIG 6.1: SIMULATION RESULTS

The image shows a simulation waveform viewer, likely from a hardware description language (HDL) simulator like those integrated with FPGA development tools (e.g., Xilinx Vivado Simulator, ModelSim). It displays how different signals in a digital design change over time. Let's break down the information presented:

Left Pane: Signal Names and Initial Values

This pane lists the signals being observed in the simulation and their initial values at the beginning of the simulation (time 0 ns).

- **clk:** Clock signal. Its initial value is 0. The waveform on the right will show it toggling periodically.
- **rst:** Reset signal. Its initial value is 0. Typically, a reset signal is asserted (goes high or low depending on the design) at the beginning to initialize the system.
- **emergency:** Emergency signal. Its initial value is 0. This likely represents an emergency condition that might affect the behavior of the design.
- **pedestrian_NS:** Pedestrian request for North-South crossing. Its initial value is 0. A pulse on this signal would indicate a pedestrian wants to cross in the North-South direction.

- **pedestrian_EW:** Pedestrian request for East-West crossing. Its initial value is 0. A pulse on this signal would indicate a pedestrian wants to cross in the East-West direction.
- **NS_light [2:0]:** A 3-bit signal representing the state of the North-South traffic light. Its initial value is 001. This is likely an encoded value representing a specific light state (e.g., Green, Yellow, Red).
- **EW_light [2:0]:** A 3-bit signal representing the state of the East-West traffic light. Its initial value is 100. This is also likely an encoded value for a specific light state.
- **NS2_light [2:0]:** Another 3-bit signal, possibly representing a secondary North-South traffic light or a different aspect of the North-South signaling. Its initial value is 100.
- **EW2_light [2:0]:** Another 3-bit signal, possibly representing a secondary East-West traffic light or a different aspect of the East-West signaling. Its initial value is 100.
- **NS_ped [1:0]:** A 2-bit signal representing the state of the North-South pedestrian signal. Its initial value is 01. This is likely an encoded value for the pedestrian signal state (e.g., Walk, Don't Walk).
- **EW_ped [1:0]:** A 2-bit signal representing the state of the East-West pedestrian signal. Its initial value is 00. This is also likely an encoded value for the pedestrian signal state.

Right Pane: Waveforms over Time

This pane shows how the values of the signals change over the simulated time, ranging from 0 ns to 1,000 ns (1 µs). The horizontal axis represents time, and the vertical axis shows the different signals.

- **clk:** The clock signal is a periodic square wave, indicating the passage of time and providing the timing reference for the digital circuit.
- **rst:** The reset signal remains low (0) throughout the displayed simulation time. This suggests the system is not being reset during this period.
- **emergency:** The emergency signal is low (0) for most of the time but goes high (1) briefly around the 700 ns mark. This pulse likely triggers an emergency response in the traffic light controller.
- **pedestrian_NS:** The North-South pedestrian request signal has a pulse high between approximately 50 ns and 150 ns. This indicates a pedestrian requested to cross North-South at this time.

- **pedestrian_EW:** The East-West pedestrian request signal has a pulse high between approximately 350 ns and 450 ns. This indicates a pedestrian requested to cross East-West at this time.
- **NS_light [2:0]:** The North-South traffic light starts at 001. It then changes to 100 (around 200 ns), then to 010 (around 400 ns), then back to 100 (around 550 ns), and finally back to 001 (around 800 ns). This sequence likely represents the transitions between different light states (e.g., Green -> Yellow -> Red -> Green).
- **EW_light [2:0]:** The East-West traffic light starts at 100. It remains at 100 while the North-South light is 001. It then changes to 001 (around 200 ns) when the North-South light turns 100. It goes back to 100 (around 400 ns) when the North-South light turns 010, and then back to 001 (around 550 ns) when the North-South light returns to 100. Finally, it returns to 100 (around 800 ns) when the North-South light goes back to 001. Notice the generally opposite behavior compared to the NS light, as expected for opposing traffic directions.
- **NS2_light [2:0] and EW2_light [2:0]:** These signals seem to mirror the behavior of EW_light and NS_light respectively, but with a potential delay or slightly different timing in the transitions around the pedestrian requests.
- **NS_ped [1:0]:** The North-South pedestrian signal starts at 01. When the pedestrian_NS request occurs, it changes to 00 (around 200 ns) and then back to 01 (around 800 ns). This likely represents the pedestrian signal changing to "Don't Walk" when the traffic light changes for vehicles and then back to "Walk" (or a ready state) after a cycle.
- **EW_ped [1:0]:** The East-West pedestrian signal starts at 00. When the pedestrian_EW request occurs, it changes to 01 (around 400 ns) and then back to 00 (around 800 ns). This likely represents the pedestrian signal changing to "Walk" when the East-West traffic has a red light and then back to "Don't Walk".
- The simulation waveform depicts the behavior of a traffic light controller, likely for a North-South and East-West intersection with pedestrian crossings. The controller cycles through different traffic light states, and it responds to pedestrian requests by changing the pedestrian signals and potentially adjusting the traffic light timing. The brief high pulse on the emergency signal around 700 ns might trigger a specific emergency sequence in the traffic light

behavior, although its direct effect on the displayed signals within this timeframe isn't immediately obvious from this limited view.

- To fully understand the behavior, you would need to know the specific encoding used for the NS_light, EW_light, NS_ped, and EW_ped signals (e.g., which 3-bit value corresponds to Green, Yellow, Red). However, the general operation of alternating traffic flow and responding to pedestrian requests is evident in the waveforms.

6.2 RTL SCHEMATIC:

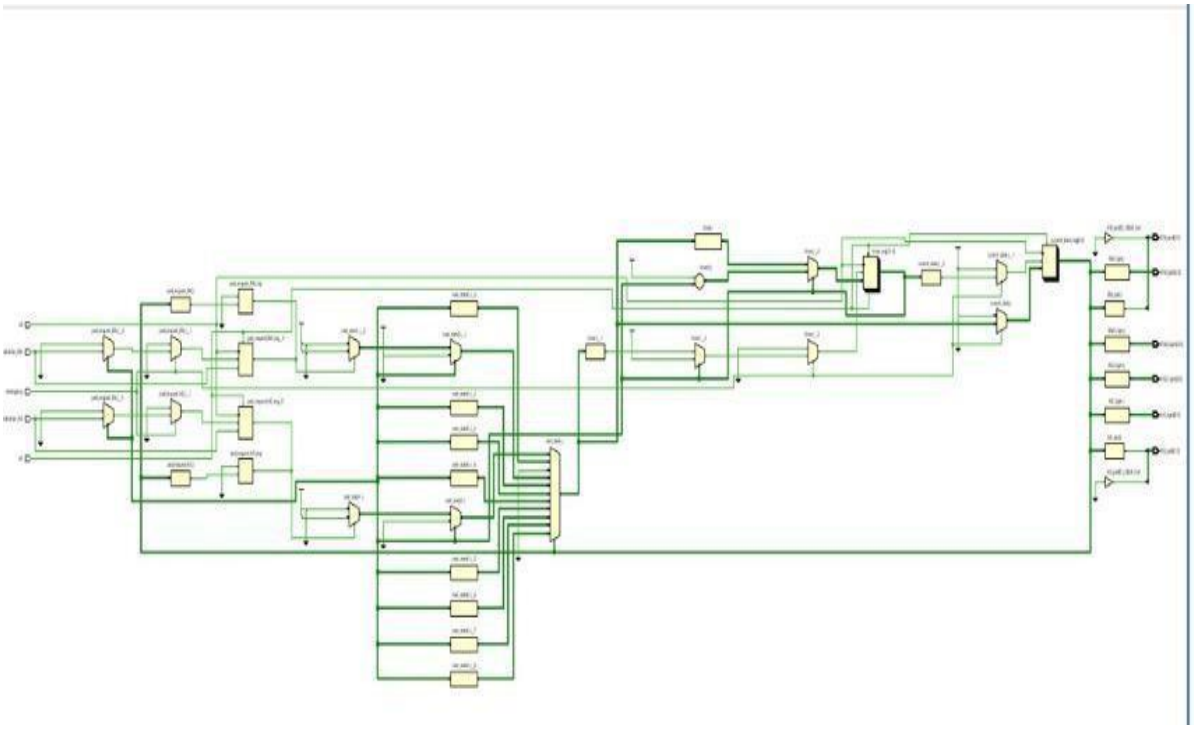

FIG 6.2: RTL SCHEMATIC DIAGRAM

Logic Gates: Symbols representing AND, OR, NOT, and possibly XOR gates are visible. These are the fundamental building blocks of digital logic.

Resistors: Rectangular boxes likely represent resistors, which are used to control current flow.

Integrated Circuits (ICs): The rectangular boxes with pins along the edges are likely integrated circuits, which contain multiple electronic components.

Connectors/Headers: The pins along the right edge suggest connectors or headers, possibly for input or output signals

At starting we have the inputs and outputs at ending.

Inputs on the Left:

- Multiple input signals (e.g., clk, rst, pedestrian requests, emergency, etc.).
- These signals feed into logic gates and registers.

Combinational Logic Blocks (Middle Section):

- Contains AND, OR, NOT, XOR gates.
- Likely implementing state transitions and traffic control logic.
- Several MUX (multiplexers) indicate state selection logic.
- Registers may be storing state information.
- Interconnections: The lines connecting the components represent wires or traces on a circuit board, showing how different parts of the circuit are electrically linked.
- Organization: The circuit seems organized into distinct sections or blocks, suggesting different functional units.

6.3 Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	88	63400	0.14
FF	40	126800	0.03
IO	21	210	10.00

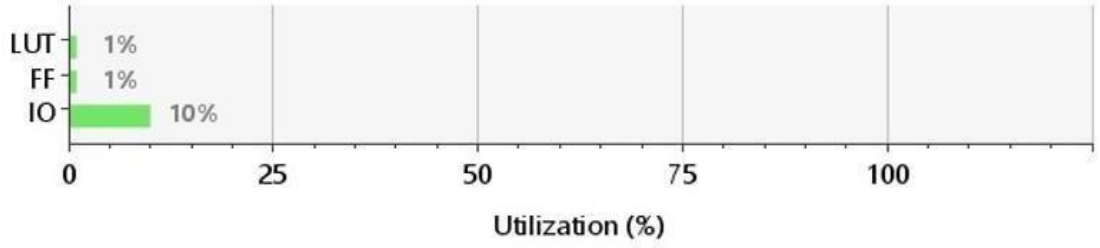

Fig 6.3: Utilization Report

The image displays a resource utilization report, showing how much of certain hardware resources are being used. It includes both a tabular format with numerical values and a graphical representation of the utilization percentages**.**

Tabular Data:

Resource: This column lists the different types of hardware resources being tracked. LUT(Look-Up Table):LUTs are fundamental building blocks in Field- Programmable Gate Arrays (FPGAs). They are used to implement logic functions. FF (Flip-Flop): Flip-flops are memory elements that store a single bit of data. They are essential for sequential logic and state machines.

IO (Input/Output): These are the pins or connections used for communication between the hardware and the outside world.

Utilization: This column shows the number of each resource that is currently being used.

Available: This column indicates the total number of each resource available on the target hardware.

LUT: 63400

FF: 126800

IO: 210

Utilization %: This column shows the percentage of each resource that is being utilized (calculated as (Utilization / Available) * 100).

LUT: 0.14%

FF: 0.03%

IO: 10.00%

Graphical Representation (Bar Chart):

The bar chart visually represents the "Utilization %" data from the table.

The horizontal axis (x-axis) represents the "Utilization (%)" from 0% to 100%. The vertical axis (y-axis) lists the resources: LUT, FF, and IO.

The length of each bar corresponds to the utilization percentage for that resource. The chart shows that:

LUT and FF have very low utilization (close to 0%). IO has a significantly higher utilization of 10%.

Analysis and Implications:

Low LUT and FF Utilization: The very low utilization of LUTs and FFs indicates that the current design or application is not making intensive use of the FPGA's logic resources. This could mean that the design is relatively simple or that it has significant room for expansion and more complex functionality.

Moderate IO Utilization: The 10% IO utilization suggests that the design is using a noticeable portion of the available input/output pins. This could be a limiting factor if the design needs to interface with many external devices or signals.

Potential for Optimization: Depending on the goals of the design, the low LUT and FF utilization might suggest opportunities for optimization. For example, if the goal is to fit a larger or more complex design onto the same hardware, the designer could consider strategies to reduce the IO usage or find ways to utilize more of the available logic resources

6.4 POWER REPORT:

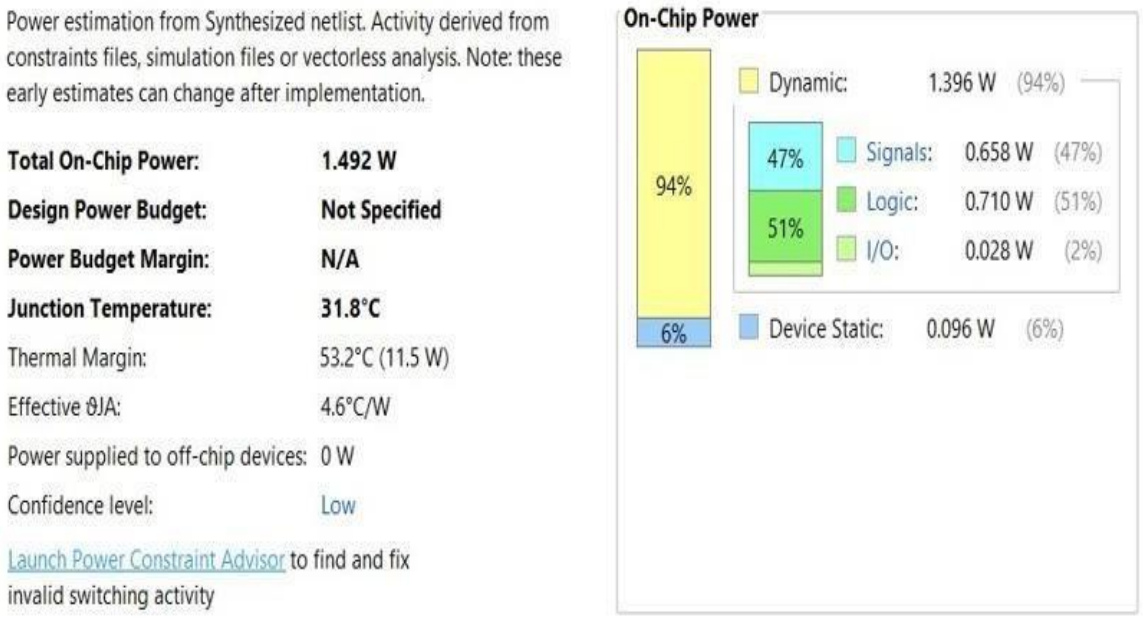

Fig 6.4: Power Report

The image shows a power estimation report for a synthesized netlist. It provides information about the total power consumption, its breakdown into dynamic and static power, and other relevant parameters like junction temperature and thermal margin.

Total On-Chip Power: 1.492 W: This is the total estimated power consumption of the chip.

Design Power Budget: Not Specified: The design power budget, which is the maximum allowable power consumption, is not defined in this report. This is important to note as it's crucial for determining if the design meets power requirements.

Power Budget Margin: N/A: Since the design power budget is not specified, the power budget margin (the difference between the budget and the estimated power) is not applicable.

Junction Temperature: 31.8 °C: This is the estimated temperature of the semiconductor junction, which is a critical parameter for device reliability.

Thermal Margin: 53.2 °C (11.5 W): The thermal margin indicates how much the junction temperature can rise before exceeding the maximum allowed temperature. The 11.5W corresponds to the power dissipation that would cause this rise.

Junction Temperature: 31.8 °C: This is the estimated temperature of the semiconductor junction, which is a critical parameter for device reliability.

Thermal Margin: 53.2 °C (11.5 W): The thermal margin indicates how much the junction temperature can rise before exceeding the maximum allowed temperature. The 11.5W corresponds to the power dissipation that would cause this rise.

Effective ΘJA: 4.6 °C/W: This is the junction-to-ambient thermal resistance, which describes how effectively heat is transferred from the junction to the surrounding environment.

Power supplied to off-chip devices: 0 W: This indicates that the report is only considering on-chip power consumption and doesn't include power supplied to external devices.

Confidence level: Low: This indicates that the power estimation has low confidence, likely due to the early stage of the d esign or the limited information available for the analysis.

On-Chip Power Breakdown (Bar Chart):

Dynamic Power: 1.396 W (94%): This is the power consumed due to switching activity in the circuit. It is further broken down into:

Signals: 0.658 W (47%): Power consumed by the switching of signals in the design.

Logic: 0.710 W (31%): Power consumed by the switching of logic gates.

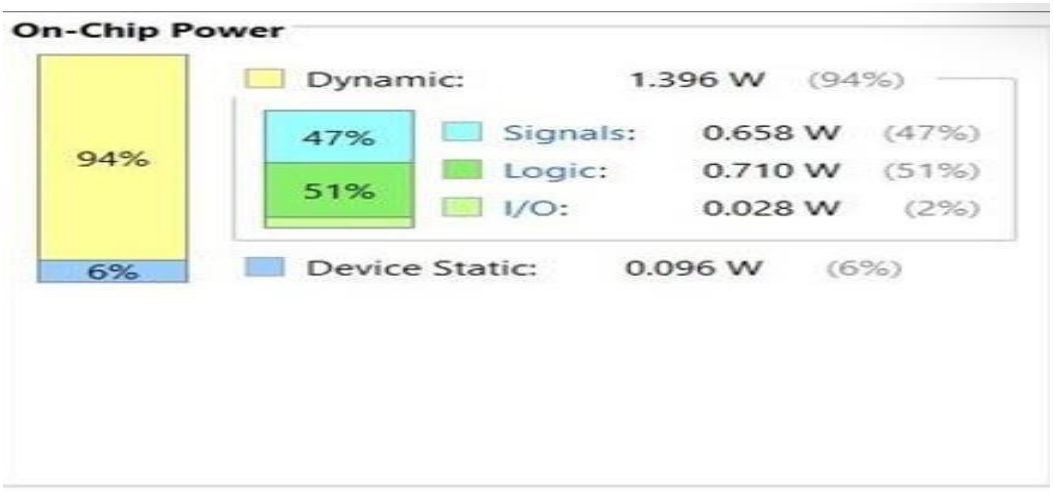

Fig 6.5: on chip power

IO: 0.028 W (2%): Power consumed by the switching of input/output buffers. Device Static: 0.096 W (6%): This is the power consumed by the device even when there is no switching activity, mainly due to leakage currents.

6.5 TECHNOLOGY SCHEMATIC:

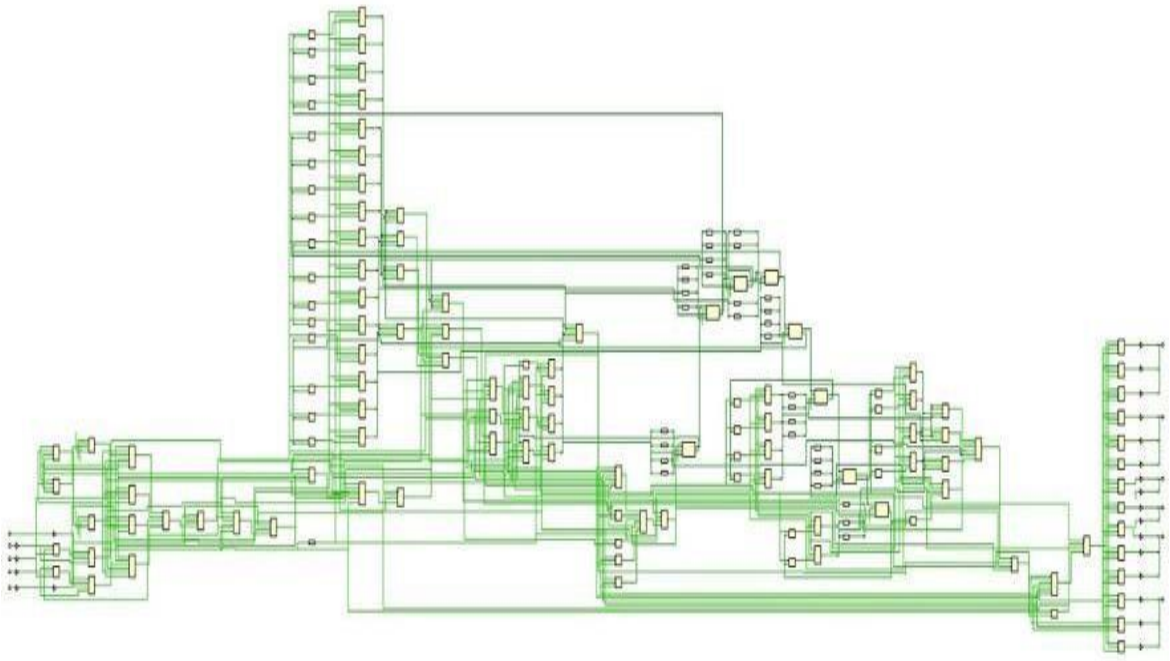

FIG 6.6: TECHOLOGY SCHEMATIC DIAGRAM

The image shows a visual representation of a digital circuit design. It displays interconnected logic gates and other circuit elements, along with their connections, forming a network that performs a specific function.

The image is a schematic, a common method for representing digital circuits at a high level of abstraction. It focuses on the functional blocks and their interconnections rather than the detailed transistor-level implementation.

Logic Gates: The numerous small boxes with lines coming in and out are likely logic gates. Common examples include AND, OR, XOR, NOT, NAND, and NOR gates. The exact type of each gate is not discernible from the image, but the shape and number of inputs/outputs could provide clues if you were familiar with the specific tool's gate representations.

Flip-Flops/Registers: Some of the larger rectangular blocks might represent flipflops or registers. These are memory elements used to store bits and are essential for sequential logic.

Other Blocks: Some blocks may represent other functional units like multiplexers, decoders, adders, or custom logic blocks.

Interconnections: The green lines connecting the blocks represent the wires or

signal paths that carry data between the different components. These connections define the flow of information within the circuit.

Hierarchical Structure: The circuit appears to have some hierarchical structure, with components grouped into distinct sections or modules. This suggests that the circuit might be designed using a modular approach, where smaller functional blocks are combined to form a larger system.

Data Flow: By tracing the signal paths (green lines), one could attempt to understand the flow of data through the circuit and infer its overall function.

6.6 DEVICE LAYOUT:

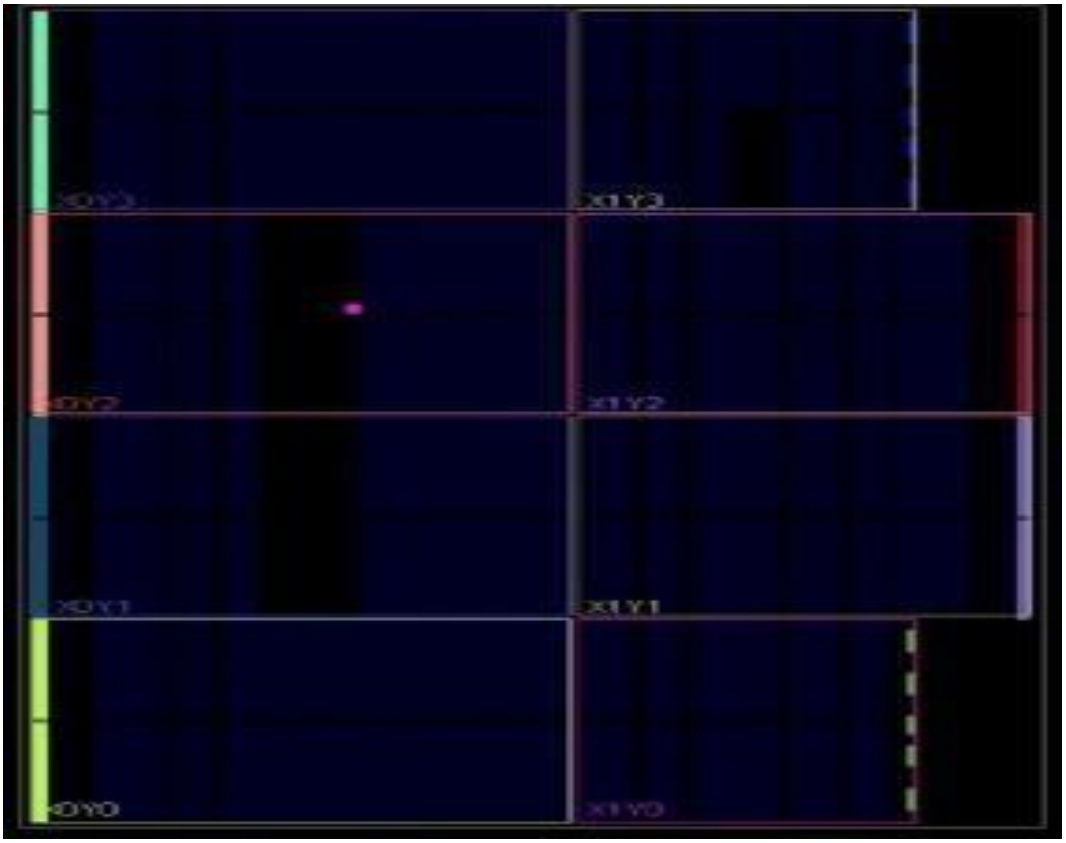

FIG 6.7: DEVICE LAYOUT DIAGRAM

Rectangular Grid: The image displays a rectangular grid-like structure, which is typical for IC layouts. This grid represents the organization of different functional blocks or modules within the chip.

Functional Blocks: The colored rectangles within the grid represent different functional blocks or modules of the IC. Each block likely performs a specific task.

Interconnects: The thin lines connecting the blocks represent the interconnects or wiring that allow communication and data transfer between the different modules.

Labels: The labels "X1Y0", "X1Y1", "X1Y2", "X1Y3", "X1Y4" are likely addresses

or coordinates used to identify the specific blocks within the grid. This suggests structured addressing scheme for accessing different parts of the chip.

Device Layout: The overall image is referred to as a "device layout," which refers to the physical arrangement of components on the chip.

Color Coding: The different colors used for the blocks and interconnects likely represent different types of signals, layers, or functionalities. For example, red might represent power lines, blue might represent data lines, and different colors for blocks might indicate different types of circuits.

Pink Dot: The single pink dot in one of the blocks is interesting. It could represent a specific test point, a defect, or a particular component of interest. Without further context, it's difficult to determine its exact meaning.

Vertical and Horizontal Lines: The vertical and horizontal lines that divide the grid into smaller rectangles likely represent routing channels or boundaries between different functional regions.

Edge Structures: The structures along the edges of the layout, especially the ones on the right side with a series of small rectangles, could represent input/output (I/O) pads or other interface circuitry.

Dark Background: The dark background likely represents the substrate material of the chip or the area where no active circuitry is present.

FPGA OUTPUTS:

FIG 6.8: Pedestrian condition

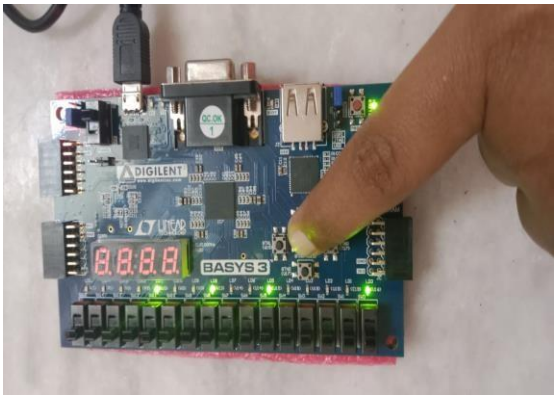

FIG 6.9: Emergency condition

6.7 ADVANTAGES :

- Prevents traffic congestion using real-time signal control.
- Improves pedestrian safety with controlled crossings.
- Allows emergency vehicle priority to reduce delays.
- Can be integrated into Smart City projects.
- All-in-One Tool
- Easy-to-Use Interface
- Pre-built Components
- Real-Time Debugging
- Optimized Designs
- Supports Many FPGAs

CHAPTER -7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion:

The "Automatic Traffic Light Controller for Four-Way Intersection Using Verilog" successfully optimizes traffic signal management by implementing a finite state machine (FSM) for efficient control. The system ensures smooth traffic flow by dynamically adjusting signal timings based on predefined conditions. Additionally, the inclusion of pedestrian and emergency override signals enhances road safety and prioritizes emergency vehicles, reducing delays and congestion. The Verilog-based implementation ensures reliable and high- speed operation, making it suitable for real-time applications.

7.2 Future Scope:

1.Integration with IoT: The system can be enhanced by incorporating IoT sensors to detect real-time traffic density and adjust signal timings dynamically.

2.AI-Based Optimization: Machine learning algorithms can be used to predict traffic patterns and optimize signal timings further.

3.Smart City Implementation: This system can be integrated into smart city infrastructure, where it can communicate with other traffic control systems for better coordination.

4.Vehicle-to-Infrastructure (V2I) Communication: Enabling communication between traffic signals and vehicles can improve traffic flow and safety.

5.Solar-Powered Traffic Signals: The system can be upgraded to run on renewable energy sources, making it more sustainable and cost-effective.

6.Emergency Vehicle Detection using RFID: Adding RFID-based emergency vehicle detection can improve response times and traffic management.

As a future scope, cameras and sensors can be integrated to the designed system so that when the traffic controller an ambulance, it can automatically divert the traffic accordingly so as to ensure that there is no obstacle and the way for the ambulance.

REFERENCES

[1] Shabarinath BB and Swetha Reddy K (2017) “Timing and Synchronization for explicit FSM based Traffic Light Controller”, IEEE 7th International Advance Computing Conference.

[2] Boon Kiat Koay and Maryam Mohd. Isa (2009) “Traffic Light System Design on FPGA”, Proceedings of 2009 IEEE Student Conference on Research and Development (SCORED 2009), 16-18 Nov. 2009, UPM Serdang, Malaysia.

[3] Venkata Kishore S (2017) “FPGA based Traffic Light Controller”, International Conference on Trends in Electronics (ICEI).

[4] D Bhavana, D Ravi Tej, Priyanshi Jain, G Mounika, R Mohini, Bhavana (2015) “Traffic Light Controller Using FPGA”, International Journal of Engineering Research and Applications (IJERA).

[5] Medany WME and Hussain (2007) “FPGA based Advanced Real Traffic Light Controller System Design”, IEEE Workshop on Intelligent Data Acquisition Computing Systems: Technology and Applications, Germany.

[6] Samir Palnitkar, “Verilog HDL: A Guide to Digital Design and Synthesis,” 2nd Edition, Pearson Education, 2003.

[7] Pong P. Chu, “FPGA Prototyping by Verilog Examples,” Wiley-IEEE Press, 2008. Offers practical and FPGA-oriented Verilog design examples, ideal for Basys 3 implementation.

[8] IEEE Standard 1364-2005: IEEE Standard for Verilog Hardware Description Language, Official language standard that defines the rules and syntax for writing Verilog programs.

[9] GitHub Repositories on Traffic Light Controllers using Verilog -Several open-source repositories showcase sample Verilog code for multi-way traffic light control.

[10] Research Paper: "Design and Implementation of Intelligent Traffic Light Controller using Verilog HDL" – International Journal of Engineering Research & Technology (IJERT), Vol. 10, Issue 5, 2021.

APPENDIX

MAIN CODE:

```
`timescale 1ns / 1ps
module traffic (
    input clk, rst,          // Clock and reset
    input emergency,         // Emergency mode
    input pedestrian_NS,     // Pedestrian request for NS &
    NS2 input pedestrian_EW,              // Pedestrian request
    for EW & EW2
    output reg [2:0] NS_light,  // North-South light (Red, Yellow, Green)
    output reg [2:0] EW_light,  // East-West light (Red, Yellow, Green)
    output reg [2:0] NS2_light, // Second North-South road
    output reg [2:0] EW2_light, // Second East-West road
    output reg [1:0] NS_ped,    // NS & NS2 pedestrian light (Red,
    Green) output reg [1:0] EW_ped                // EW & EW2 pedestrian
    light (Red, Green)
 );

    // State Encoding using parameter
    parameter NS_GREEN      = 4'b0000;
    parameter NS_YELLOW     = 4'b0001;
    parameter EW_GREEN      = 4'b0010;
    parameter EW_YELLOW     = 4'b0011;
    parameter NS2_GREEN     = 4'b0100;
    parameter NS2_YELLOW    = 4'b0101;
    parameter EW2_GREEN     = 4'b0110;
    parameter EW2_YELLOW    =
    4'b0111; parameter PEDESTRIAN_NS
    = 4'b1000; parameter
    PEDESTRIAN_EW = 4'b1001;
    parameter EMERGENCY     = 4'b1100;

    // State and Timing Registers
```

```
reg [3:0] current_state, next_state;
reg [31:0] timer;
reg ped_request_NS, ped_request_EW;
// Timing Constants
parameter GREEN_TIME = 30;
parameter YELLOW_TIME = 3;
parameter PEDESTRIAN_TIME = 10;
// State transition logic
always @(posedge clk or posedge rst)
begin if (rst) begin
current_state <=
NS_GREEN; timer <=
GREEN_TIME;
ped_request_NS <= 0;
ped_request_EW <= 0;
end else if (emergency) begin
current_state <= EMERGENCY;
end else if (timer == 0)
begin current_state <=
next_state;
case (next_state)
PEDESTRIAN_NS, PEDESTRIAN_EW: timer <= PEDESTRIAN_TIME;
NS_GREEN, EW_GREEN, NS2_GREEN, EW2_GREEN: timer <=
GREEN_TIME;
NS_YELLOW, EW_YELLOW, NS2_YELLOW, EW2_YELLOW: timer <=
YELLOW_TIME;
endcase
end else
begin
timer <= timer -1;
```

```
        end
    end

    // Next-state logic
    always @(*) begin
        case
            (current_state)
            NS_GREEN:
                next_state = (timer == 0) ? (ped_request_NS ? PEDESTRIAN_NS :
NS_YELLOW) : NS_GREEN;
            NS_YELLOW:
                next_state = (timer == 0) ? EW_GREEN : NS_YELLOW;
            EW_GREEN:
                next_state = (timer == 0) ? (ped_request_EW ? PEDESTRIAN_EW :
EW_YELLOW) : EW_GREEN;
            EW_YELLOW:
                next_state = (timer == 0) ? NS2_GREEN : EW_YELLOW;
            NS2_GREEN:
                next_state = (timer == 0) ? NS2_YELLOW : NS2_GREEN;
            NS2_YELLOW:
                next_state = (timer == 0) ? EW2_GREEN : NS2_YELLOW;
            EW2_GREEN:
                next_state = (timer == 0) ? EW2_YELLOW : EW2_GREEN;
            EW2_YELLOW:
                next_state = (timer == 0) ? NS_GREEN : EW2_YELLOW;
            PEDESTRIAN_NS:
                next_state = (timer == 0) ? NS_YELLOW : PEDESTRIAN_NS;
            PEDESTRIAN_EW:
                next_state = (timer == 0) ? EW_YELLOW : PEDESTRIAN_EW;
            EMERGENCY:
                next_state = NS_GREEN;
            default:
                next_state = NS_GREEN;
```

```
        endcase
    end

    // Output logic
      always @(*) begin
       // Default all lights to red
       NS_light = 3'b100; EW_light = 3'b100; NS2_light = 3'b100; EW2_light = 3'b100;
       NS_ped = 2'b00; EW_ped = 2'b00;
       case (current_state)
          NS_GREEN:  begin NS_light = 3'b001; NS_ped = 2'b01; end
          NS_YELLOW: NS_light = 3'b010;
          EW_GREEN:  begin EW_light = 3'b001; EW_ped = 2'b01; end
          EW_YELLOW: EW_light = 3'b010;
          NS2_GREEN: NS2_light = 3'b001;
          NS2_YELLOW: NS2_light = 3'b010;
          EW2_GREEN: EW2_light = 3'b001;
          EW2_YELLOW: EW2_light = 3'b010;
          PEDESTRIAN_NS: NS_ped = 2'b01;
          PEDESTRIAN_EW: EW_ped = 2'b01;
          EMERGENCY: begin
             NS_light = 3'b001;
             EW_light = 3'b100;
             NS2_light = 3'b100;
             EW2_light = 3'b100; end
       End
end

endmodule
```

TEST BECH CODE:

```
module traffic_tb;
    reg clk, rst, emergency;
    reg pedestrian_NS, pedestrian_EW;
    wire [2:0] NS_light, EW_light, NS2_light, EW2_light;
    wire [1:0] NS_ped, EW_ped;

    // Instantiate the DUT
    traffic uut (
        .clk(clk),
        .rst(rst),
        .emergency(emergency),
        .pedestrian_NS(pedestrian_NS),
        .pedestrian_EW(pedestrian_EW),
        .NS_light(NS_light),
        .EW_light(EW_light),
        .NS2_light(NS2_light),
        .EW2_light(EW2_light),
        .NS_ped(NS_ped),
        .EW_ped(EW_ped)
    );

    // Clock generation
    always #5 clk = ~clk; // 10ns clock period
    initial begin
         clk = 0;
         rst = 1;
        emergency = 0;
        pedestrian_NS = 0;
        pedestrian_EW = 0;
        #20 rst = 0; // Release reset
        #100 pedestrian_NS = 1; #10 pedestrian_NS = 0;
        #400 pedestrian_EW = 1; #10 pedestrian_EW = 0;
```

```
        #300 emergency = 1; #50 emergency = 0;

        #500;
        $st
    op;
    end
endmodule
```

Fully Automated Traffic Light Controller system for a four-way intersection using Verilog

Karamsetty Venkata Lakshmi Saranya1, Musyam Lavanya2,

Nandipati Lakshmi Sravani3,Vemula Chandini4

1 UG Student, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute of Technology, Nambur, Andhra Pradesh, India

2 UG Student, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute of Technology, Nambur, Andhra Pradesh, India

3 UG Student, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute of Technology, Nambur, Andhra Pradesh, India

4 UG Student, Dept of Electronics and Communication Engineering, Vasireddy Venkatadri Institute ofTechnology, Nambur, Andhra Pradesh, India

ABSTRACT

Traffic lights are placed in roads to control the flow of traffic and to prevent accidents. This paper proposes a Moore machine based fully automated and efficient traffic light controller system for four-way intersection. The system is designed on Xilinx Artix-7 xc7a100tcsg324-1 FPGA using Xilinx Vivado and Verilog Hardware Description Language. The designed system runs up to a maximum operating frequency of 10 MHz.

Keyword: *Field Programmable Gate Array, Finite State Machine, Hardware Description Language, Light Emitting Diode, Verilog.*

1.INTRODUCTION

Traffic congestion is one of the predominant problems prevailing in cities and towns. In T-intersection and four-way intersection, the probabilities of accidents are slightly higher. So, to ensure smooth flow of traffic and to avoid road accidents, traffic light systems are used.

The proposed traffic light controller system is designed for four-way intersection roads. In this system, the waiting time of vehicles at the intersection is reduced by a great extent. Microcontroller and Microprocessor based traffic light systems are already present. But the disadvantage associated with these systems is that, they work on fixed time, and doesn't have flexibility. So, this paper concentrates on developing a reconfigurable traffic light controller system, which works on Field Programmable Gate Array (FPGA) as it doesn't have a fixed hardware structure and can be reprogrammed by using Hardware Description Language (HDL). Verilog is chosen for modelling the traffic light controller system, as usage of Verilog HDL allows to define the specifications of the parameters used in the design of the system. Also, Verilog HDL is one of the commonly used HDLs as it has simple syntax and it resembles software programming languages to some extent.

FPGA boards have many input switches and output Light Emitting Diodes (LEDs) in it, which make it suitable for the design of traffic light controller systems. FPGAs are used for designing prototypes for many electronic applications. Another advantage of FPGA is that, it makes the whole system more efficient. The FPGA chosen here is Artix-7, which is a product of Xilinx, a semiconductor manufacturing company. Artix-7 is preferred as it is cost

efficient and for its**1** high performance. It is used in systems so as to have optimum power consumption.

There are different types of traffic control systems which are put forth by researchers for different real time situations. A traffic light controller was designed using Verilog HDL considering two roads [1] and for a T-junction [2]. A system for four-way intersection was implemented using two signals, red and green [3]. Another system makes use of three signals, red, yellow and green to regulate the traffic [4]. But the drawback of these systems for four-way intersection was that they don't allow the maximum possible movement of vehicles across the intersection [3][4]. Vehicles from few roads are made to wait at the intersection unnecessarily as allowing them doesn't disturb the moving vehicles. The proposed system makes sure that this drawback is removed to allow the maximum transportation of vehicles across the intersection and to prevent the unnecessary waiting time of the motorists.

Moore model of Finite State Machine (FSM) is used to design the traffic light controller system as the output of the system (traffic light signals) depends only upon the current state of the system. This feature makes the system fully automated. The system considers the four roads to have equal traffic and makes use of the Binary encoding scheme. Compared to other works, the proposed system is more efficient by making use of minimal number of states which are necessary enough to allow the maximum transportation of vehicles across the intersection. The reduction in number of states also helps in achieving minimal power consumption. The traffic controller system also makes use of the maximum possible number of safe states. Before the stoppage of traffic across each direction, yellow signal is displayed in the corresponding displays which indicate that the flow of traffic will be stopped in few seconds. The states containing yellow signals act as safe states and prevent the possibility of accidents. A Simulation based system is designed and the same is done using Xilinx Vivado. Complete information of the system designed is obtained using various facilities present in this software like timing report, utilization report, power report, etc.

2.METHODOLOGY

The paper concentrates on developing a traffic light controller system for a four-way intersection. Each road has three light displays corresponding to the flow of traffic towards the other three roads. Hence there are twelve light displays in total at the intersection. By using a common control logic, the system is designed in such a way that, certain light displays operate in the same manner. This simplifies the design with ten light displays. Each of these light displays have the provision to show red, green and yellow signals. The red signal specifies to stop, the green signal allows the flow of traffic and the yellow signal specifies that the flow of traffic will be stopped in few seconds. The proposed system helps to prevent vehicle collisions at the intersection by use of 'safe' states. The red, yellow and green signals of each of the light displays are modeled as individual output LEDs. So a total of thirty output LEDs are used. A state diagram and a state table are constructed based upon the simplified logic to model a finite state machine for the proposed traffic light controller system.

3.IMPLEMENTATION OF TRAFFIC LIGHT CONTROLLER SYSTEM

- *Light Display assignment*

The proposed traffic light controller system is designed to operate at a maximum frequency of 10.0 MHz. The time period (*T*) of the clock used in this system is given by the formula t=1/f Where *f* is the maximum operating frequency of the system. The ten light displays are labelled as D0, D1, D2, D3, D4, D5, D6, D7, D8 and D9 respectively. Light displays are assigned according to the direction of traffic flow. Two directions are grouped together in D0 and D2 states instead of employing separate light displays representing the traffic flow along those directions. Table I represents the assignment of light displays according to the directions of flow of traffic.

TABLE I. LIGHT DISPLAYS AND THE CORRESPONDING DIRECTIONS

Direction name	Corresponding movement of vehicles at the intersection
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	
D8	
D9	

- *State assignment*

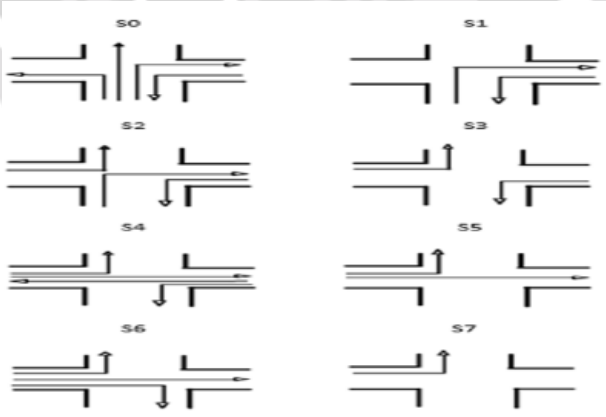

Fig 1: Assignment of States

A. State Diagram

State Assignments & Transitions

1. **S0 (NS Green, EW Red)**
   North-South (NS) traffic has a green light.East-West (EW) traffic has a red light.Transitions to **S1** after a fixed time.
2. **S1 (NS Yellow, EW Red)**
   NS light turns yellow, warning vehicles to stop.EW remains red.Transitions to **S2** after a short delay.
3. **S2 (NS Red, EW Green)**
   NS light turns red.EW traffic gets a green light to move.Transitions to **S3** after a fixed time.
4. **S3 (NS Red, EW Yellow)**
   EW light turns yellow, warning vehicles to stop.NS remains red.Transitions to **S4** after a short delay.
5. **S4 (Pedestrian Walk)**All vehicle signals are red.Pedestrian crossing light is on for a specific direction.Transitions to **S5** after pedestrian time ends.
6. **S5 (Pedestrian Walk for Opposite Side**)Pedestrian crossing is allowed for the other direction.Vehicles remain stopped.Transitions to S6 after pedestrian time ends.
7. **S6 (Normal Cycle Restart – Back to S0)**Resets back to normal operation (S0).Alternates between NS/EW traffic as per the cycle.
8. **S7 (Emergency Mode – Back to S0)**If an emergency is detected, all normal cycles stop.The emergency direction gets a green light, others red.Once the emergency clears, the system.
9.

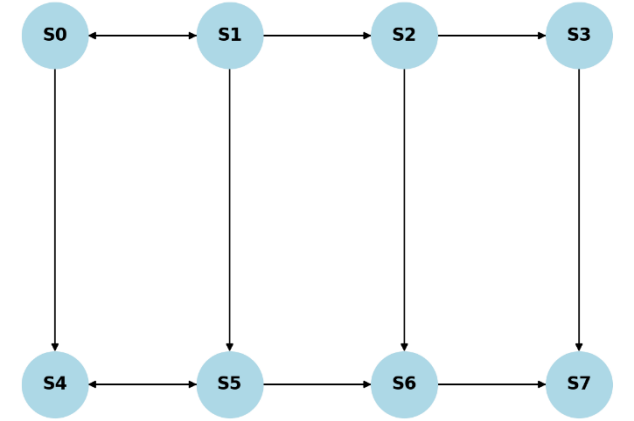

Fig -2: State Diagram

- *State Table*

Having defined the state diagram, a state table is now formed. Initially, in state S0, vehicle movement is allowed only for directions corresponding to the light displays D0, D4 and D7. So, green signal is enabled only for these directions, and for the other directions, red signal is enabled on their corresponding light displays. Similarly, according to each state, red, yellow and green signals are enabled for each of the light displays. Binary encoding scheme is used in the proposed system. Table II illustrates the state table of the system.

TABLE II. STATE TABLE OF THE PROPOSED SYSTEM

State	clk	rst	emergency	pedestrian_NS	pedestrian_EW	NS_light[2:0]	EW_light[2:0]	NS_ped[1:0]	EW_ped[1:0]	Next State
S0 (NS Green, EW Red)	1	0	0	0	0	001 (Green)	100 (Red)	00	00	S1
S1 (NS Yellow, EW Red)	1	0	0	0	0	010 (Yellow)	100 (Red)	00	00	S2
S2 (NS Red, EW Green)	1	0	0	0	0	100 (Red)	001 (Green)	00	00	S3
S3 (NS Red, EW Yellow)	1	0	0	0	0	100 (Red)	010 (Yellow)	00	00	S4
S4 (Pedestrian Walk NS)	1	0	0	1	0	100 (Red)	100 (Red)	01 (Walk)	00	S5
S5 (Pedestrian Walk EW)	1	0	0	0	1	100 (Red)	100 (Red)	00	01 (Walk)	S6
S6 (Restart Cycle)	1	0	0	0	0	001 (Green)	100 (Red)	00	00	S0
S7 (Emergency Mode)	1	0	1	X	X	100 (Red)	100 (Red)	00	00	S0 (after clearing emergency)

In each of the states, if MSB is 1, it indicates that the light display is showing red signal, and so, the movement of vehicles in the direction corresponding to the light display is restricted. Similarly, if the middle bit is 1, it corresponds to yellow signal, and indicates that the traffic flow will stop soon. If LSB is 1, it corresponds to green signal and the flow of traffic in the corresponding direction is allowed. The odd states act as 'safe' states as few of the light displays show yellow signal indicating that the flow of traffic will be stopped soon, so that vehicles from those directions can stop, since crossing the intersection during the period of the yellow signal of the current state may lead to accidents due to the flow of traffic regulated during the succeeding even state.

4.RESULTS

The proposed system is designed as a Moore FSM using Xilinx Vivado and Verilog HDL. Simulation, synthesis, implementation and generation of bit stream were done and no DRC violations were found.

A. Simulation

Fig. 3 displays the result of behavioral simulation showing the waveform of the Traffic light controller system for the test bench applied using Verilog HDL.

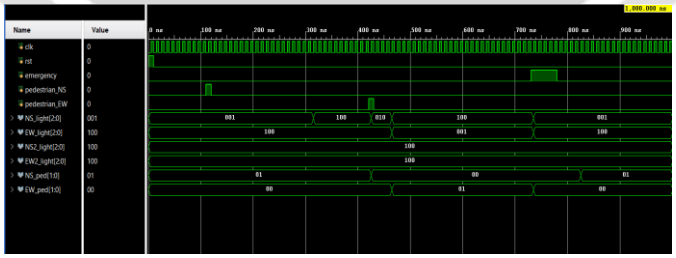

Fig 3: Simulation

B. RTL Schematic

Fig. 4 shows the RTL schematic of the designed system.

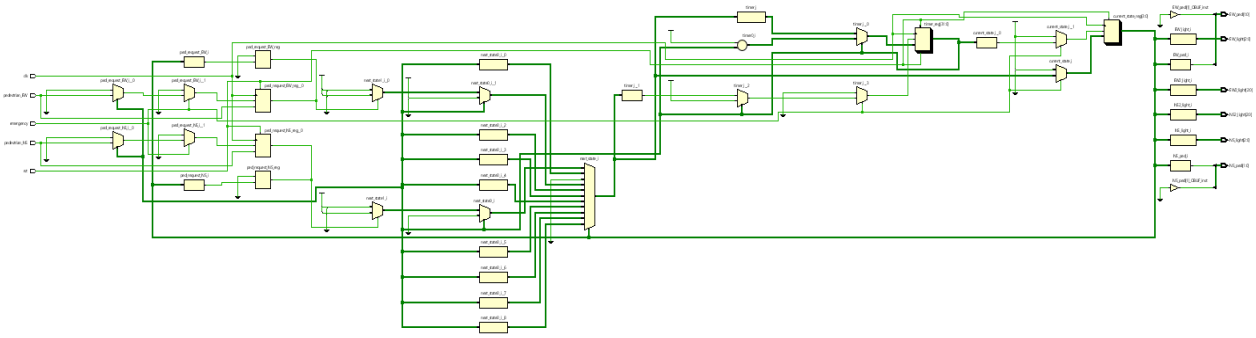

Fig. 4: RTL schematic of the system

C. Timing report

It was observed that the timing report generated during synthesis matched with the timing report generated during implementation. The obtained timing report is shown in Fig. 5.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	128	Total Number of Endpoints:	128	Total Number of Endpoints:	NA

Fig. 5.:Timing report of the system

D. Utilization report

It was observed that the utilization report generated during synthesis matched with the utilization report generated during implementation. The obtained utilization report is shown in Fig. 6.

Resource	Utilization	Available	Utilization %
LUT	88	63400	0.14
FF	40	126800	0.03
IO	21	210	10.00

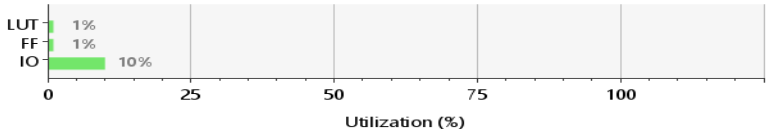

Fig. 6: Utilization report of the system

E. Power report

It was observed that the power report generated during synthesis matched with the power report generated during implementation. The obtained power report is shown in Fig. 7

Total On-Chip Power:	**1.492 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**31.8°C**
Thermal Margin:	53.2°C (11.5 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

Fig. 7. Power report of the system

F. Technology schematic

It was observed that the technology schematic generated during synthesis matched with the technology schematic generated during implementation. The obtained technology schematic is shown in Fig. 8.

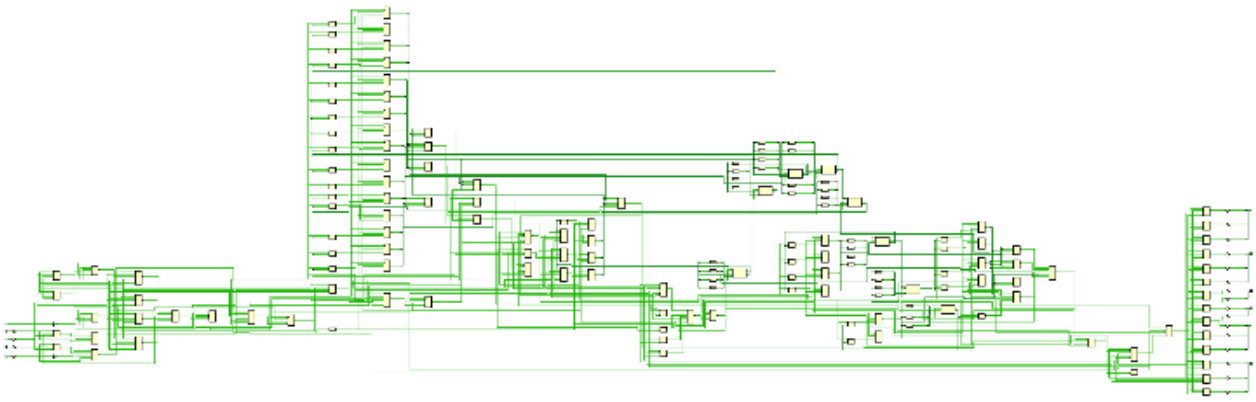

Fig. 8. Technology schematic of the system

G. Device layout after implementation

Fig. 9 shows the device layout obtained after implementation.

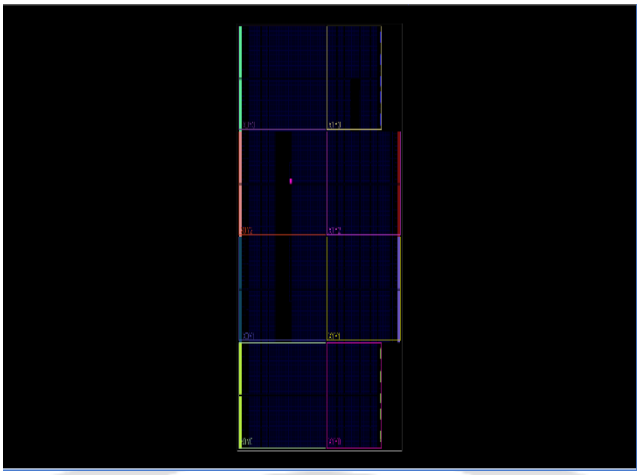

Fig. 9: Device layout of the system

H. FPGA IMPLEMENTATION

Fig. 9 :FPGA Basys 3

The image shows a fully powered and functioning Digilent Basys 3 FPGA board. The "Done" LED near the USB port is lit, indicating that the FPGA has been successfully configured with a bitstream. Several switches on the left side of the board are turned on, as shown by the illuminated green LEDs, which likely correspond to inputs such as reset, mode selection, or control signals. The seven-segment display is active and showing numbers, suggesting that your design is successfully driving output data to the display—this could be used for indicating a countdown timer or the active traffic direction in your 4-way traffic light controller. The four push buttons in the center of the board are ready to be used, typically for functions like pedestrian control or emergency override. The overall setup confirms that the board is powered, programmed, and running a design—if you're testing your traffic light controlle

5.CONCLUSION AND FUTURE WORK

The traffic light controller system designed is well suited to regulate traffic at four-way intersection. The system is designed in Artix-7 FPGA so as to utilize its advantage of efficient power consumption. Verilog HDL is used for programming purpose because if the user wishes to make any changes in the system, it is possible to apply the required changes easily through Verilog HDL code. One of the advantages of this system is its 'safe state' feature implemented in every odd state, which plays a major role in preventing vehicle collisions.

The simulated waveform matched the traffic light signals obtained from the state table. The implemented system had a minimal power utilisation of 1.492 W and had used only 0.01% of the flip flops and 15.24% of the total IO (Input Output) facilities present in the FPGA.

As a future scope, cameras and sensors can be integrated to the designed system so that when the traffic controller system sees an ambulance, it can automatically divert the traffic accordingly so as to ensure that there is no obstacle and the way for the ambulance is clear.

REFERENCES

[1] Shabarinath B B and Swetha Reddy K (2017) "Timing and Synchronisation for explicit FSM based Traffic Light Controller", IEEE 7th International Advance Computing Conference.

[2] Boon Kiat Koay and Maryam Mohd. Isa (2009) "Traffic Light System Design on FPGA", Proceedings of 2009 IEEE Student Conference on Research and Development (SCOReD 2009), 16-18 Nov. 2009, UPM Serdang,
Malaysia.

[3] Venkata Kishore S (2017) "FPGA based Traffic Light Controller", International Conference on Trends in Electronics (ICEI).

[4] D.Bhavana, D.Ravi Tej, Priyanshi Jain, G.Mounika, R.Mohini, Bhavana (2015) "Traffic Light Controller Using Fpga", International Journal of Engineering Research and Applications (IJERA).

[5] Medany W M E and Hussain (2007) "FPGA based Advanced Real Traffic Light Controller System Design", IEEE Workshop on Intelligent Data Acquisition Computing Systems: Technology and Applications, Germany.

[6] Nath S, Pal C, Sau S, Mukherjee A, Guchhait A and Kandar D (2012) "Design of an intelligent Traffic Light Controller with VHDL", International Conference on Radar, Communication and Computing, pp.92-97.