

SMART TRAFFIC MANAGEMENT SYSTEM

MINI PROJECT REPORT

*Submitted in partial fulfillment of the
Requirements for the award of Bachelor of Technology Degree
In Electronics and Communication Engineering
Of APJ Abdul Kalam Technological University*

By

ABHIJITH RV

(RegNo: MBT22EC002/B22EC1102)

ABIJITH SS

(RegNo: MBT22EC006/B22EC1105)

GEORGE K JOHN

(RegNo: MBT22EC048/B22EC1125)

JEFFIN I PATRICK

(RegNo: MBT22EC059/B22EC1132)



MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous)

MAR IVANIOS VIDYANAGAR, NALANCHIRA, THIRUVANANTHAPURAM, 695015

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

2025

MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous)

MAR IVANIOS VIDYANAGAR, NALANCHIRA,

THIRUVANANTHAPURAM, 695015

DEPARTMENT OF ELECTRONICS AND

COMMUNICATIONENGINEERING



CERTIFICATE

This is to certify that this Mini project report entitled “**SMART TRAFFIC MANAGEMENT SYSTEM**” is a Bonafide record of work done by **Abhijith RV, Abijith SS , George K John and Jeffin I Patrick** of the sixth semester Electronics and Communication branch towards the partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics and Communication Engineering of APJ Abdul Kalam Technological University

GUIDE

COORDINATOR

HEAD OF THE DEPARTMENT

Mr Niyas K Haneefa
Assistant Professor
Dept of ECE
MBCET

Dr.Vineetha Mathai
Assistant Professor
Dept of ECE
MBCET

Dr.Luxy Mathews
Associate Professor
Dept of ECE
MBCET

ACKNOWLEDGEMENT

With great enthusiasm and pleasure, we are bringing out this Mini project report here. We use this opportunity to express our heartiest gratitude to the support and guidance offered to us from various sources during the course of completion of our project. We are also grateful to **Dr.Luxy Mathews, Associate Professor**, Head of the Department, Electronics and Communication Engineering, for her valuable suggestions. It is our pleasant duty to acknowledge our Mini project Co-coordinator **Dr.Vineetha Mathai Assistant Professor** in the Department of Electronics and Communication Engineering, for helping us all throughout the project. Let us express our heartfelt gratitude to our guide, **Mr.Niyas K Haneefa, Assistant Professor** in the Department of Electronics and Communication Engineering who has guided and given supervision of the project work.. We would also like to acknowledge the **Technical experts at Keltron**, whose valuable insights and advice during our visit greatly contributed to the technical depth and practical orientation of our project. Above all, we owe our gratitude to the **Almighty** for showering abundant blessing upon us. We also express our wholehearted gratefulness to all our classmates who have expressed their views and suggestions about our projects and have helped us during the course of the project. We extend our sincere thanks and gratitude once again to all those who helped us make this undertaking a success.

ABSTRACT

Traffic congestion at intersections leads to delays in commuting daily and hinders emergency vehicles. Traditional traffic signals are based on solid-state timers, making them inefficient when adapting actual conditions. This project presents a smart traffic system system that dynamically adapts signals based on traffic flow and emergency vehicle detection. YOLO model analyzes live cameras to count vehicles, and the microphone recognizes the ambulance siren. The Raspberry Pi 4 processes this data, while the basys 3 FPGA adjusts traffic lights, reduces congestion and optimizes signal times to prioritize rescue vehicles. This approach improves traffic efficiency, minimizes delays, improves emergency times, and contributes to a more intelligent urban transport system.

CONTENTS

1. INTRODUCTION	1
2. TECHNOLOGY IN GENERAL	3
2.1 Block Diagram	5
2.2 Block diagram explanation	6
3. TECHNOLOGY IN SPECIFIC	10
3.1 Webcam	10
3.2 Microphone	11
3.3 raspberry pi 4	13
3.4 FPGA BASYS 3	15
3.5 TRAFFIC LIGHTS	18
4.HARDWARE IMPLEMENTATION	22
4.1 CIRCUIT DIAGRAM	22
5.SOFTWARE IMPLEMENTATION	23
5.1Algorithm	23
5.2 Flow Chart	27
5.3 Program	28
6.RESULT	38
7.CONCLUSION	39
REFERENCES	40

LIST OF FIGURES

1.	Fig 2.1 block diagram of circuit	5
2.	Fig 2.2.1 Webcam	6
3.	Fig 2.2.2 Microphone	6
4.	Fig 2.2.3 Raspberry pi 4	7
5.	Fig 2.2.4 FPGA Basys3	8
6.	Fig 2.2.5 Traffic Light	9
7.	Fig 3.1 Webcam	10
8.	Fig 3.2 Microphone	12
9.	Fig 3.3 Raspberry pi 4	14
10.	Fig 3.4 FPGA Basys 3	17
11.	Fig 3.5 Traffic Light	20
12.	Fig 4.1 CIRCUIT DIAGRAM	23

CHAPTER 1

INTRODUCTION

Traffic jams at gateway points create substantial delays that produce daily operational slowdowns and reduce ambulance emergency response times. The daily travel routine creates delays and emergency vehicles such as ambulances encounter challenges in reaching their intended destinations with speed. The current traffic signal system operates by strict timetable management while ignoring actual traffic conditions. The failure to adjust signal timing to current traffic flow creates longer delays and raises traffic jams. The movement of emergency vehicles becomes obstructed by heavy traffic since emergency vehicles become trapped resulting in delayed emergency response actions.

We have developed an automated traffic management system that actively manages signals according to traffic congestion together with emergency vehicles identification. The system modifies traffic signal operations through combination of traffic volumes and response to detected emergency vehicles. The system implements a camera paired with a YOLO model which has received customized training to identify and track vehicles that pass through an intersection. A microphone combined with audio processing technology listens for ambulance sirens through its built-in system. Raspberry Pi 4 uses the processed data to analyze traffic patterns along with detecting emergency vehicles. The Basys 3 FPGA engages to manage traffic signals along with providing increased lengths of green lights to roads. The system provides emergency vehicles with prior access as well as controlling lights based on traffic volume density. The process uses real-time data collection together with vehicle detection capabilities and ambulance identification protocols and traffic signal regulation techniques. The system applies modifications to traffic signals according to its analysis results.

Several strong benefits result from this system which leads to better traffic management and lowered congestion while enabling expedited emergency responses. This approach allows automated control of traffic signals to minimize The system reduces unnecessary waiting time to provide continuous traffic flow between vehicles. Emergency medical personnel can save lives through enhanced operation of ambulances by shortening their journey times at intersections.

SMART TRAFFIC MANAGEMENT SYSTEM

In conclusion, our Smart Traffic Management System promotes urban traffic control by increasing the intelligence and effectiveness of intersections. It greatly enhances road efficiency and emergency response capabilities by combining real-time data processing with automatic signal adjustments, making city traffic more intelligent and sensitive to actual situations.

CHAPTER 2

TECHNOLOGY IN GENERAL

The Smart Traffic Management System prioritizes emergency vehicle passage, reduces traffic congestion, and increases road efficiency by integrating cutting-edge technologies. The following state-of-the-art technologies are used by the system:

Real-Time Traffic Signal Adjustment

Conventional traffic lights are inefficient since they are run on scheduled times. Our solution optimizes traffic flow at intersections by dynamically modifying traffic lights based on real-time data. As a consequence, commuters experience less traffic and shorter wait times.

Vehicle Detection via YOLO (You Only Look Once) Model

A specially trained YOLO (You Only Look Once) model is used to use cameras to identify and count cars at intersections in order to determine traffic density. Modern object detection framework YOLO is capable of processing images rapidly and effectively, allowing for real-time signal control decision-making.

Emergency Vehicle Detection

This device detects ambulance sirens by using a microphone and audio processing technology. The system makes sure emergency vehicles are given precedence at junctions by identifying their distinct aural signature, which minimizes significant delays and speeds up emergency response times.

Data Processing with Raspberry Pi 4

The Raspberry Pi 4 is the system's brain, processing information from the microphone and car detecting cameras. It determines the best time to change signals in order to maintain efficient traffic flow and ambulance prioritization by analyzing traffic patterns and emergency vehicle notifications.

Traffic Signal Control via Basys 3 FPGA

The traffic lights are managed by the Basys 3 FPGA (Field-Programmable Gate Array) using inputs from the Raspberry Pi. Roads with a high vehicle density are given longer green lights, allowing emergency vehicles to pass through without delay, thanks to the FPGA's dynamic adjustment of traffic signal timing.

When combined, these technologies produce a creative, responsive traffic control system that changes to the circumstances of the moment. Urban traffic efficiency is greatly increased, safety is improved, and the impact of congestion on daily commutes and emergency response times is reduced.

2.1 BLOCK DIAGRAM OF CIRCUIT

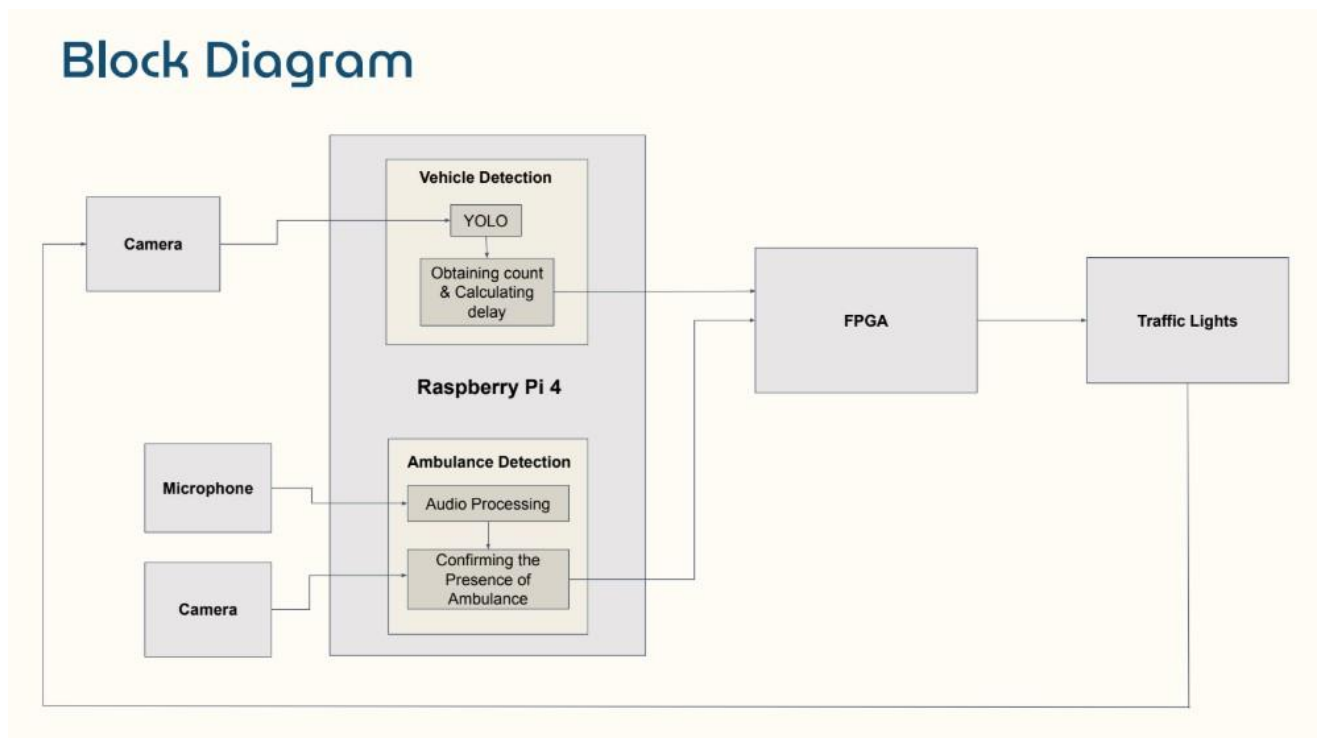


Fig 2.1 Block diagram

2.2 BLOCK DIAGRAM EXPLANATION

2.2.1 WEB CAM



Fig 2.2.1 WEBCAM

- **Vehicle Detection:** Traffic at intersections is being captured on camera in real time. This video feed is analyzed by the Raspberry Pi 4's YOLO (You Only Look Once) model. In order to identify and count the number of vehicles that are passing the intersection, the YOLO model examines the webcam's images.
- **Traffic Density Assessment:** The webcam helps measure traffic density by continuously observing the intersection, giving vital information to dynamically modify traffic signal timing according to the volume of traffic on each road.

In overall, the **webcam** is the main sensor used to record traffic footage, which is subsequently processed to improve real-time signal control and traffic flow.

2.2.2 MICROPHONE



Fig2.2.2 MICROPHONE

- **Emergency Vehicle Siren Detection:** The microphone listens for the distinctive sound signature of sirens on emergency vehicles. (e.g., ambulances, fire trucks).
- **Priority Signal Adjustment:** When the microphone picks up the siren, it alerts the **Raspberry Pi**, which analyzes the data and prioritizes emergency vehicles' passage by

SMART TRAFFIC MANAGEMENT SYSTEM

modifying traffic lights appropriately. This guarantees the timely passage of emergency vehicles through intersections.

In summary, the system can modify traffic lights to prioritize and expedite the passage of emergency cars by using the **microphone** to detect emergency vehicle sirens.

2.2.3 RASPBERRY PI4

2.2.3 RASPBERRY PI4



1. **Processing Data:** The ambulance detection system and the vehicle detection system (YOLO camera) both provide real-time data to the Raspberry Pi (microphone with audio processing).
2. **Analyzing Traffic Patterns:** In order to enable dynamic signal modifications, it analyzes traffic conditions and locates high-density locations using this data.
3. **Coordinating with FPGA:** Once the data has been analyzed, the Raspberry Pi talks to the Basys 3 FPGA to modify the traffic signals accordingly, either granting emergency vehicles precedence or prolonging the duration of green lights.

In short, the Raspberry Pi serves as the system's brain, managing communication between sensors and traffic signal controllers as well as data processing and decision-making.

2.2.4 FPGA BASYS3



Fig 2.2.4 FPGA BASYS3

In your Smart Traffic Management System, the **FPGA (Field-Programmable Gate Array)** plays the crucial role of **real-time traffic signal control**. Specifically, it:

1. **Receives Instructions:** The Raspberry Pi processes the traffic and emergency vehicle data and sends it data and instructions for adjusting the traffic signals.
2. **Controls Traffic Lights:** The traffic signal timings (such as the duration of the green, yellow, and red lights) are directly controlled by the FPGA in real-time, guaranteeing faster reactions to shifting traffic situations and emergency vehicle prioritizing.
3. **Real-Time Execution:** The FPGA is perfect for conducting signal adjustments without delays, guaranteeing smooth traffic flow and quick emergency reaction because it is built for high-speed, parallel processing.

In brief, the FPGA is in charge of putting the Raspberry Pi's recommendations into action by managing the actual traffic signals at junctions.

2.2.5 TRAFFIC LIGHTS



Fig 2.2.5 TRAFFIC LIGHTS

In this project, the **traffic lights** are controlled by the FPGA to **adjust signal timings** based on realtime traffic data. They are responsible for:

1. **Improve Traffic Movement: Help ease congestion at intersections by adjusting green light durations based on how heavy the traffic is.**
2. **Prioritize Emergency Vehicles:** Give priority to ambulances and other emergency vehicles by prolonging the green lights on their route.

To clarify, traffic lights are the system's way of implementing data-driven, dynamic signal adjustments for more efficient traffic flow and quicker emergency vehicle response. **2.2.6**

YOLO

Yolo is an object detection algorithm. It divides the images into grids and predicts bounding boxes. It is fast and accurate

CHAPTER 3

TECHNOLOGY IN SPECIFIC

3.1 WEBCAM



Fig 3.1 Webcam

In this project, the webcam (integrated with a custom-trained YOLO model for vehicle detection) plays a key role in monitoring traffic conditions and detecting vehicles at intersections. Here is an overview of its specifications and role:

Webcam Specifications :

- Resolution : A high-definition (HD) webcam (e.g., 1080p or higher) for clear vehicle detection.
- Frame Rate : A frame rate of at least 30 FPS (frames per second) to capture real-time movement of vehicles without lag.
- Field of View : A wide-angle lens to cover a broad intersection area, ensuring the camera can detect vehicles in multiple lanes.
- Connectivity : USB or Ethernet for easy integration with the Raspberry Pi to transmit video data.
- Low-light Capability : If operating in different lighting conditions, the webcam might need good performance in low-light or night-time settings.

Role of the Webcam in the Project:

1. Vehicle Detection:

The intersection is continuously captured on camera by the webcam. It functions in combination with the deep learning-based object identification model known as YOLO (You Only Look Once). In order to recognize and count the automobiles in the footage, the model has been specially trained.

2. Traffic Density Analysis:

SMART TRAFFIC MANAGEMENT SYSTEM

The Raspberry Pi receives a real-time video feed from the webcam and processes it to examine traffic patterns. The system can assess whether an intersection is congested by counting the number of cars in each lane, which allows the traffic light time to be dynamically adjusted.

The technology determines how long each light should remain red or green based on the number of identified vehicles. To alleviate traffic, the system can, for instance, prolong the green light on a road with a high vehicle density.

4. Emergency Vehicle Detection (Supplementary Role):

The webcam can help identify emergency vehicles while the microphone picks up ambulance sirens, guaranteeing that the traffic signal system gives priority to their movement through the crossing.

In conclusion, the project webcam records traffic at the crossroads in real time, allowing for vehicle detection and traffic density analysis, which power the traffic light dynamic control. It is an essential component for the system's capacity to prioritize emergency vehicles and manage traffic flow.

3.2 MICROPHONE



Fig 3..2 MICROPHONE

In this project, the microphone plays an important role in detecting emergency vehicles (such as ambulances) based on the sound of their sirens. Here's a breakdown of its **specifications** and **role** in the system:

Microphone Specifications:

- **Sensitivity:** The microphone's great sensitivity allows it to detect even distant, weak noises, such as sirens from rescue vehicles.
- **Frequency Response:** To ensure it detects the sound accurately, the microphone can detect a wide variety of frequencies, particularly in the 500–1500 Hz region that is typical of ambulance sirens.

Noise Cancellation: Since traffic sounds and general environmental noise could interfere, a microphone with noise-canceling features are used

- **Connectivity:** The Raspberry Pi is capable of processing sound data when it is connected to the microphone through a headphone connection.

Role of the Microphone in the Project:

1. Emergency Vehicle Detection:

The microphone listens for unique sound patterns, including the characteristic frequency and modulation of ambulance sirens. The microphone assists in real-time emergency vehicle detection by recording these sounds.

2. Signal Adjustment for Prioritization:

When an ambulance is detected, the **Raspberry Pi** receives the signal from the microphone, processes it, and then notifies the FPGA to modify the traffic lights. The method permits the ambulance to cross crossings without any delays, guaranteeing that the emergency vehicle is given preference.

3. Improving Response Time:

The traffic light system will respond quickly enough to clear the intersection if the siren is detected precisely and swiftly, enabling the ambulance to pass through safely and on time.

4. Integration with Traffic System:

The microphone guarantees that emergency vehicles' priority is maintained in the system when used in conjunction with webcam-based vehicle detection. Even when traffic is high, it enables the traffic signals to automatically prioritize the ambulance.

In the final analysis, the microphone in this project is in charge of hearing ambulance sirens and setting off the system that prioritizes emergency vehicles' traffic. It ensures more efficient traffic flow and faster emergency reaction times by collaborating with other sensors, such as the webcam.

3.3 RASPBERRY PI4

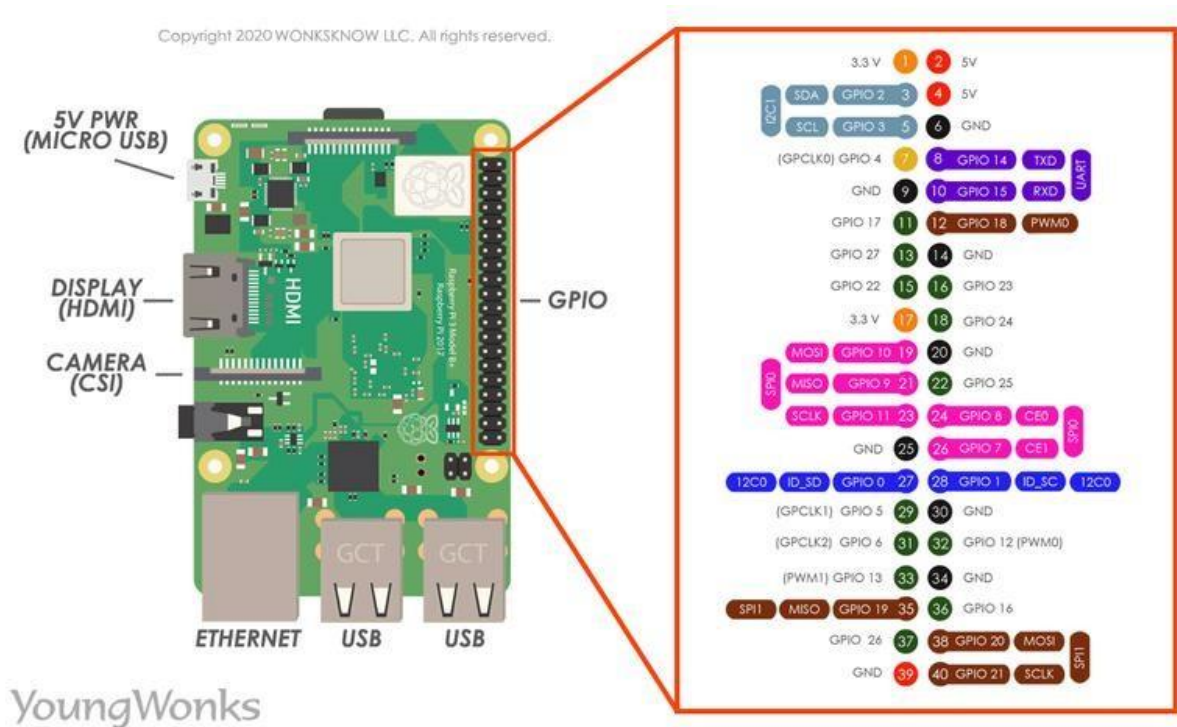


Fig 3.3 raspberry pi 4

The **Raspberry Pi 4** plays a central role in our Smart Traffic Management System, acting as the **data processor and coordinator**. Below is an overview of its **specifications** and **role** in this project:

Raspberry Pi 4 Specifications:

- **Processor:** A powerful 64-bit quad-core processor (Cortex-A72) running at 1.5 GHz, providing sufficient processing power for real-time data handling and analysis.
- **RAM:** Multiple versions of LPDDR4-3200 SDRAM are available, usually in sizes of 2GB, 4GB, or 8GB. Higher RAM configurations can provide more complex data processing (such as audio or video analysis in real time).
- **Connectivity:**

SMART TRAFFIC MANAGEMENT SYSTEM

- **USB Ports:** 2 USB 3.0 ports and 2 USB 2.0 ports for connecting peripherals like cameras, microphones, or other sensors.
 - **Ethernet:** Gigabit Ethernet for fast networking, which is useful for communicating with other devices or servers in a larger system.
 - **Wi-Fi:** Built-in 802.11ac Wi-Fi for wireless connectivity, useful for remote monitoring or control.
 - **Bluetooth:** Bluetooth 5.0 support for wireless device connections.
 - **GPIO Pins:** 40 GPIO pins for interfacing with external hardware, like sensors, buttons, or directly controlling the FPGA or traffic lights.
 - **HDMI:** Dual micro-HDMI ports for display output if required.
- **Storage:** MicroSD card slot for storage of the operating system, data, and program files.

Role of Raspberry Pi 4 in the Project:

The Raspberry Pi 4 serves as the **central processing unit** in the Smart Traffic Management System.

Here are its specific roles:

1. Data Collection and Processing:

- The Raspberry Pi receives real-time video data from the **webcam** and sound data from the **microphone**.
- Using its processing power, the Raspberry Pi **runs object detection models** (like the custom YOLO model) to identify and count vehicles from the camera feed, as well as analyze the audio input for ambulance sirens using YAMNET.
- It uses the processed data to analyze **traffic density** and **detect emergency vehicles**.

2. Traffic Signal Control Coordination:

- After analyzing the data, the Raspberry Pi sends instructions to the **FPGA** to adjust the traffic lights based on the current traffic conditions or emergency vehicle needs.
- It can adjust the green light duration for high-traffic lanes or trigger an immediate signal change to prioritize emergency vehicles like ambulances.

3. Real-Time Decision Making:

- The Raspberry Pi makes decisions in **real time**, processing data and triggering actions with minimal delay, which is crucial for effective traffic management and ensuring timely emergency responses.

4. Communication Hub:

- The Raspberry Pi facilitates communication between the different components of the system, such as the **webcam**, **microphone**, **FPGA**, and possibly a **central server** or **user interface** for monitoring or control.

5. System Integration:

- It integrates all aspects of the system, managing input from the sensors, processing it, and then controlling output (i.e., the traffic light signals) based on the processed data.

In summary, the Smart Traffic Management System's **brain** is the **Raspberry Pi 4**. In order to guarantee that the traffic system adjusts to changing circumstances, it processes data from the webcam and microphone, decides in real time how to modify the traffic lights, and interacts with the FPGA. It is the perfect option for such a dynamic and responsive system because of its power, adaptability, and connectivity.

3.4 FPGA BASYS 3

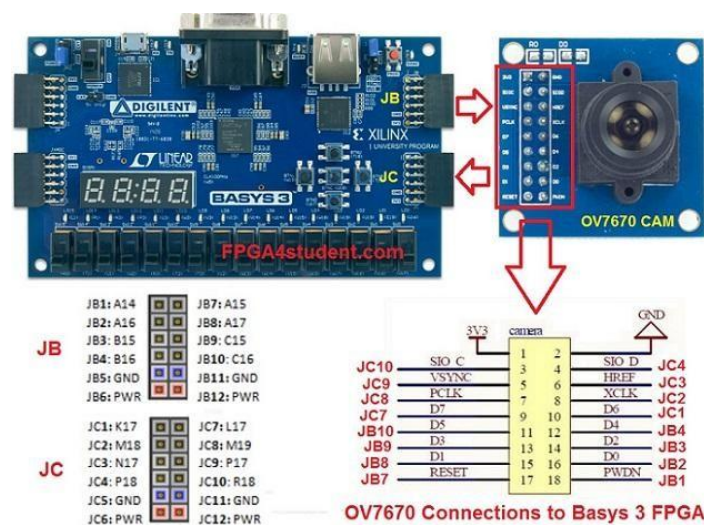


Fig 3.4 FPGA BASYS 3

In our Smart Traffic Management System, the **FPGA (Field-Programmable Gate Array)** plays a critical role in **real-time control of traffic signals** based on the processed data from the Raspberry Pi

FPGA Specifications (Basys 3):

- **Model:** **Basys 3 FPGA** (from Digilent) is often used in educational and prototype applications due to its versatility and ease of use.
- **Chip:** It uses a **Xilinx Artix-7 FPGA** with a high-speed, low-power design.
- **Logic Cells:** It contains around **33,280 logic cells**, allowing for complex processing and control tasks.
- **Clock Speed:** The FPGA typically operates at **100 MHz** (or higher), which is fast enough for controlling traffic signals in real time.
- **I/O Pins:** The Basys 3 board has numerous **general-purpose I/O (GPIO)** pins that can be configured to interface with various external devices like traffic lights, sensors, and communication modules.
- **Memory:** It includes internal block RAM that can store intermediate data, like the current traffic light status or emergency vehicle detection flags.
- **Power Consumption:** The FPGA is energy-efficient compared to other processors, making it ideal for embedded systems like traffic management.

Role of FPGA in the Project:

The **FPGA** serves as the **real-time traffic signal controller**, carrying out immediate adjustments based on data from the Raspberry Pi.

Real-Time Signal Control:

- After receiving instructions from the Raspberry Pi (which processes data from the webcam and microphone), the FPGA **adjusts the traffic signals** in real time.

- It directly controls the traffic light circuits (red, yellow, green), ensuring they switch based on traffic conditions or emergency vehicle needs.
- It makes decisions like **extending green lights** for lanes with high vehicle density or giving **priority green lights** for emergency vehicles (e.g., ambulances).

2. **Speed and Efficiency:**

- The FPGA is designed for parallel processing, allowing it to handle multiple tasks simultaneously without delay. This is crucial for fast traffic signal adjustments in busy intersections, where quick decisions are needed to optimize traffic flow and prioritize emergency vehicles.
- Its ability to perform **low-latency operations** ensures that the traffic lights change without noticeable delay, crucial for reducing congestion and facilitating emergency response.

3. **Direct Control Over Traffic Signals:**

- The FPGA directly interfaces with the **traffic light controllers**, enabling it to implement decisions made by the Raspberry Pi, such as adjusting the green light duration or switching signals for emergency vehicles.
- It controls the traffic lights directly using the data it receives, without relying on an operating system, which allows it to respond and act extremely quickly..

4. **Flexible Configuration:**

- The FPGA can be reprogrammed to adjust or expand the system as needed. For example, you can modify the logic to support more complex decision-making algorithms or additional features, such as adapting to weather conditions.

5. **Handling Multiple Inputs:**

- The FPGA can process **multiple inputs** at once, such as vehicle detection signals from the webcam, siren detection from the microphone, and even other sensors that may be added in the future (e.g., weather sensors, traffic cameras).
- It ensures that the system reacts quickly to changing conditions by managing these inputs in parallel.

6. **Ensuring Reliability:**

SMART TRAFFIC MANAGEMENT SYSTEM

- Since the FPGA runs on hardware, it is generally more **reliable** and **robust** than software-based solutions. This makes it suitable for mission-critical systems like traffic management, where reliability and uptime are essential.

Summary:

The **FPGA (Basys 3)** in our Smart Traffic Management System is in charge of controlling traffic signals in real time. Based on information from sensors (webcam and microphone), it gets commands from the Raspberry Pi and makes the required adjustments to the traffic light system, like lengthening green lights for clogged routes or giving priority to emergency vehicles. Because of its low latency, parallel processing, and dependability, it is the perfect option for managing the intricate and everchanging metropolitan traffic systems.

3.5 TRAFFIC LIGHTS



Fig 3.5 TRAFFIC LIGHTS

In our Smart Traffic Management System, the **traffic lights** are the **output devices** that visually indicate traffic control based on the instructions from the **FPGA**. The traffic lights help **regulate the flow of vehicles** by switching between **green, yellow, and red** signals in real time.

Traffic Light Specifications (Using 3V LEDs):

- **LED Type:** The traffic lights use **3V LEDs** for energy efficiency and bright visibility. These LEDs are typically used for the **red, yellow, and green** signals.
- **Color LEDs:**

- **Red LED:** Typically used to signal stop or halt.
- **Yellow LED:** Used for caution or to indicate the transition between green and red.
- **Green LED:** Used to indicate go or allow movement.
- **Voltage:** The LEDs are powered by a **3V supply** from FPGA, which is standard for low power, efficient lighting. This voltage ensures the LEDs are bright enough for visibility while consuming minimal energy.
- **Current:** Each LED may draw a current of around 20mA (milliamps) per color, depending on the specific LED used. Multiple LEDs will be used for each traffic signal, requiring adequate current control.
- **Mounting:** The LEDs are typically mounted in a **traffic light housing** with a diffuser lens to ensure visibility from a distance and at different angles.

Role of Traffic Lights in This Project:

In this project, the **traffic lights** are controlled by the **FPGA** based on the real-time data processed by the **Raspberry Pi**. Here's how they contribute to the overall system:

1. Visual Traffic Control:

- The primary role of the traffic lights is to **physically manage the flow of traffic** at intersections by providing clear, visual signals (red, yellow, green) to drivers.
- The traffic lights regulate when vehicles should stop or go, reducing the chances of accidents and ensuring smooth traffic flow.

2. Real-Time Signal Adjustments:

- The traffic lights are controlled dynamically based on real-time traffic conditions, which are analyzed by the Raspberry Pi using data from the **webcam** (vehicle detection) and the **microphone** (ambulance siren detection).
- The **FPGA** adjusts the light durations (e.g., extending green for high-density traffic or prioritizing green for emergency vehicles like ambulances) to adapt to these conditions, optimizing traffic flow and improving emergency response times.

3. Handling High Traffic Density:

- By extending the green light duration on routes with a high vehicle density, the **FPGA** can ease traffic by enabling more cars to cross the intersection. This is especially helpful during periods of high traffic.
- The **LED traffic lights** provide visual cues to drivers, ensuring that they follow the adjusted signal timings.

4. **Emergency Vehicle Priority:**

- In the event of an emergency vehicle (e.g., an ambulance) approaching, the **FPGA** can give priority to the vehicle by changing the traffic lights. For instance, it may **extend the green light** for the lane the ambulance is traveling on, ensuring that it can pass through the intersection without delay.
- The **LED traffic lights** will immediately reflect these changes, signaling other vehicles to stop and clear the way for the emergency vehicle.

5. **Energy Efficiency:**

- Since **3V LEDs** are utilized, the system has low power consumption and excellent visibility. LEDs are dependable and reasonably priced for long-term usage in the traffic system since they use less energy and last longer than conventional incandescent bulbs.

6. **Durability:**

- Compared to conventional incandescent or halogen traffic light bulbs, LEDs are more resilient and less likely to break. This increases the traffic lights' dependability and guarantees that they will continue to function in a variety of environmental circumstances.

7. **Adaptability:**

- The traffic light system can be modified or reprogrammed in response to the city's

evolving needs. For instance, the system can dynamically modify signal timings to minimize traffic disruptions during construction, special events, or accidents.

Summary:

In this project, the **FPGA** uses data from the **Raspberry Pi** to control the **traffic lights** (which use **3V LEDs**) and modify the signal timings in real time. Their main responsibilities include controlling traffic flow at junctions, easing congestion, and giving priority to emergency vehicles by displaying conspicuous visual signals (red, yellow, and green). Energy economy, durability, and visibility in a range of weather situations are all guaranteed by the use of **LEDs**, which enhances the traffic management system's overall efficacy and predictability.

CHAPTER 4

HARDWARE IMPLEMENTATION

4.1 CIRCUIT DIAGRAM

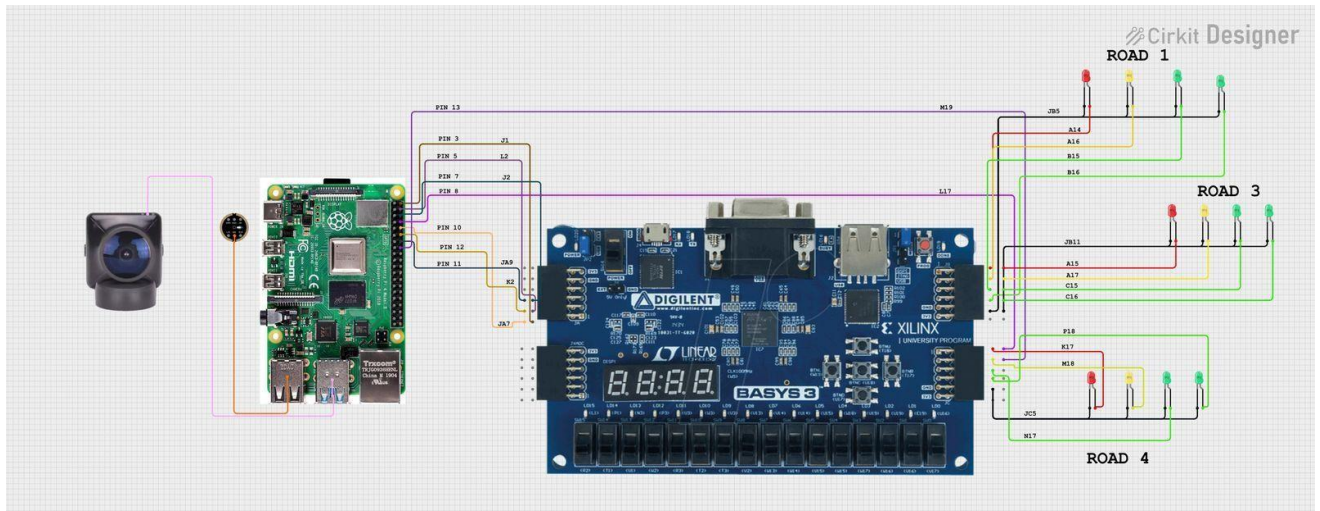


Fig 4.1 CIRCUIT DIAGRAM

4.2 WORKING

For Object Detection: The web cam captures the real time footage of the traffic and using YOLO algorithm which is installed on raspberry pi will process the webcam footage and count the number of vehicles. The raspberry pi will process the data and send an 8 bitdata to fpgabasys 3. The duration of this data determines the duration of each signals. The fpga will control the lights accordingly.

For emergency vehicle detection: The microphone captures the emergency vehicle's siren..After this the data is send to raspberry pi. We use YAMNET which is audio detection algorithm by google. YAMNET detects the siren sound .Then the webcam will turn on the confirms the presence of emergency vehicle using YOLO. The raspberry pi will process the data and generate the sequence to turn the signal for emergency vehicle to pass and sends to fpga. The fpga will control the lights accordingly.

4.3 List Of Components

Raspberry Pi4
 FPGA BASYS3
 WEBCAM
 MICROPHONE
 TRAFFIC LIGHTS

CHAPTER 5

SOFTWARE IMPLEMENTATION

5.1 Algorithm

python(Raspberry pi)

1. System Initialization

1. Set Raspberry Pi GPIO mode.
2. Configure GPIO pins for traffic lights on all lanes.
3. Load:
 - YAMNetTFLite model (for siren detection)
 - YOLO model (for object detection)
 - YAMNet class labels (from CSV) - Region masks (for vehicle counting)
4. Initialize event ``siren_detected_event``.

2. Thread 1: Siren Detection (Audio) Loop

continuously:

1. Record ~2 seconds of audio via PyAudio.
2. Preprocess audio → waveform (float32).
3. Pass audio to YAMNetTFLite model.
4. Get top class label from model output.
5. If label includes ``"siren"`, `"police"`, or `"ambulance"`:
 - Set `siren_detected_event`.`
6. Else:
 - Clear ``siren_detected_event``.
7. Sleep 1 second to reduce CPU load.

3. Thread 2: Ambulance Detection (Vision) Loop

continuously:

1. If ``siren_detected_event`` is `**set**`:
 - Open camera (if available).
 - Capture a single frame.

SMART TRAFFIC MANAGEMENT SYSTEM

- Run YOLO detection on the frame.
- Check if `"ambulance"` or `"police"` is detected.
- If found:
- Print "Ambulance Detected".
- Trigger `emergency_lights_on()` to change lights. - Wait 3 seconds.
- Release camera.

4. Vehicle Counting (Region-based)

1. Start video stream.
2. Load two region masks (lane-based detection).
3. On every frame:
 - Apply masks to clone of frame.
 - Detect vehicles using YOLO (stream=True).
 - Filter detections by "car".
 - Use `SORT` tracker to assign consistent IDs.
 - Count new unique IDs for each region.
 - Display frame, bounding boxes, and counts.
 - Break if 'q' is pressed.

5. Main Thread: Traffic Light Control Loop

infinitely:

1. Call `count_vehicles()` → get vehicle counts on Lane 1 and Lane 2.
2. Convert count to dynamic delay (e.g. `delay = count * 2`).
3. Cycle through light phases:
 - GREEN for Lane 1 & 3, RED for Lane 4 - YELLOW for Lane 1 & 3
 - RED for Lane 1 & 3, GREEN for Lane 4
 - YELLOW for Lane 4
 - GREEN for Lane 1 & 2, RED for others
4. Use `time.sleep()` based on vehicle count delay.
5. Repeat.

6. Emergency Lights Override (if ambulance detected) When called:

- Set:
- GREEN for Lane 1 & 2
- RED for Lane 3 & 4
- Activate all red/yellow/green GPIOs accordingly.

Verilog(FPGA)

1. Start System Initialization - Wait

for system clock to start (`CLK`).

- Begin counting using a `reset_counter`.

2. Power-On Reset

- If `reset_counter` is less than `RESET_DELAY`:
- Keep `rst_n` = 0 (active low reset).
- Once `reset_counter` \geq `RESET_DELAY`:
- Set `rst_n` = 1 to release reset.

3. Initialize and Toggle Left-Turn Signals (`G31`, `G41`) -

On each rising edge of `CLK`:

- If `rst_n` = 0:
- Set `G31` = 0, `G41` = 0.
- Reset the `counter`.
- Else:
- Increment `counter`.
- If `counter` == ONE_SECOND:
- Reset `counter`.
- Toggle `G31` and `G41` (i.e., blink every second).
- Optionally: only toggle `G41` if `G4` == 1.

4. Evaluate Traffic Light Signals (Combinational Logic)

These outputs change ****immediately**** based on inputs `A` through `H`:

SMART TRAFFIC MANAGEMENT SYSTEM

Lane 1:

- $\text{`R1} = A \& B \& \sim C \& \sim D \& E \& F \& (G \text{ XOR } H)\text{'}$

- $\text{`Y1} = A \& \sim B \& \sim C \& \sim D \& E \& \sim F \& G \& H\text{'}$ - $\text{`G1} = \sim A \& B \& \sim C \& F \& G \& H \& \sim (D \text{ XOR } E)\text{'}$ Lane 2:

- $\text{`G2} = \sim A \& B \& \sim C \& D \& E \& F \& G \& H\text{'}$ Lane 3:

- $\text{`R3} = (A \& B \& \sim C \& \sim D \& E \& F \& (G \text{ XOR } H)) \mid (\sim A \& B \& \sim C \& D \& E \& F \& G \& H)\text{'}$

- $\text{`Y3} = A \& \sim B \& \sim C \& \sim D \& E \& \sim F \& G \& H\text{'}$

- $\text{`G3} = \sim A \& B \& \sim C \& \sim D \& \sim E \& F \& G \& H\text{'}$ - $\text{`G31} = \text{toggles every 1s'}$ Lane 4:

- $\text{`R4} = (\sim A \& B \& \sim C \& F \& G \& H \& \sim (D \text{ XOR } E)) \mid (A \& \sim B \& \sim C \& \sim D \& E \& \sim F \& G \& H)\text{'}$

- $\text{`Y4} = A \& B \& \sim C \& \sim D \& E \& F \& G \& \sim H\text{'}$

- $\text{`G4} = A \& B \& \sim C \& \sim D \& E \& F \& \sim G \& H\text{'}$

- $\text{`G41} = \text{toggles every 1s'}$ *(optional: only if $\text{`G4} = 1\text{'}$)*

5. Loop Continuously

- Repeat all operations on every clock cycle.

- Lights and arrow signals respond in real-time to input changes.

Summary Diagram Components

- Start

- Reset Counter < RESET_DELAY? ** → Yes → $\text{`rst_n} = 0\text{'}$ → Loop

- Reset Over? ** → Yes → $\text{`rst_n} = 1\text{'}$

- Toggle G31/G41 every 1s

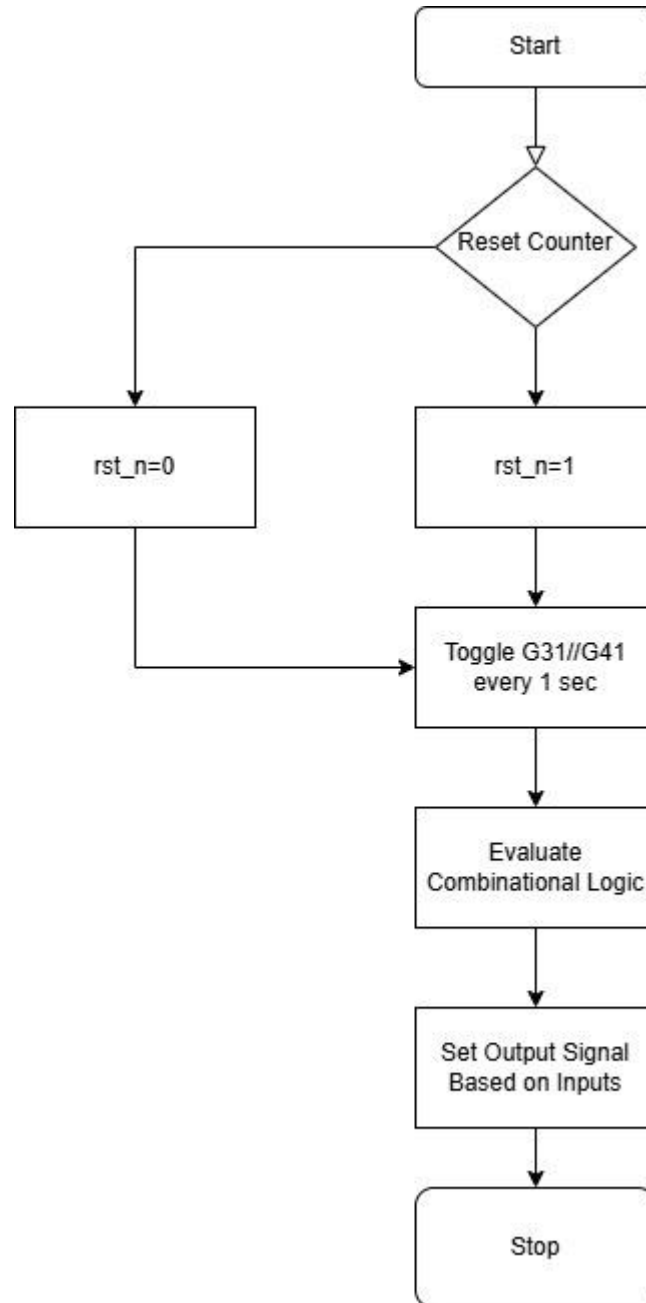
- Evaluate Combinational Logic

- Set Output Signals Based on Inputs

- Repeat

5.2 Flow Chart

Verilog



5.3 PROGRAM

```
Python
import time import numpy
as np import pyaudio
import RPi.GPIO as GPIO
import requests
import csv from sort
import Sort import
cv2

import threading
import tf.lite_runtime.interpreter as tflite from
ultralytics import YOLO

# Load TensorFlow Lite model for siren detection
tflite_model_path = "yamnet.tflite" # Update with the correct path interpreter
= tflite.Interpreter(model_path=tflite_model_path)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details() output_details
= interpreter.get_output_details()

GPIO.setmode(GPIO.BCM)
GPIO.setup(2, GPIO.OUT)
GPIO.setup(3, GPIO.OUT)
GPIO.setup(4, GPIO.OUT)
GPIO.setup(14, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)

def load_yamnet_labels():
url =
"https://raw.githubusercontent.com/tensorflow/models/master/research/audioset/yamnet/yamnet_classes_map.csv"
response = requests.get(url)
labels = [] if
response.status_code == 200:
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
decoded_content = response.content.decode("utf-8").splitlines()
csv_reader = csv.reader(decoded_content)      next(csv_reader) #
Skip header
    labels = [row[2] for row in csv_reader]
return labels
```

```
class_labels = load_yamnet_labels()
```

```
def detect_siren():    audio =
pyaudio.PyAudio()
    stream = audio.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True,
frames_per_buffer=1024)

    while True:        frames = []        for _ in
range(0, int(16000 / 1024 * 2)):
        data = stream.read(1024)
frames.append(data)

    waveform = np.frombuffer(b".join(frames), dtype=np.int16).astype(np.float32) / 32768.0
waveform = np.expand_dims(waveform, axis=0).astype(np.float32)

interpreter.set_tensor(input_details[0]['index'], waveform)  interpreter.invoke()
    scores = interpreter.get_tensor(output_details[0]['index'])[0]

top_class = np.argmax(scores)
detected_class = class_labels[top_class] if top_class<len(class_labels) else "Unknown"

    if "siren" in detected_class.lower() or "ambulance" in detected_class.lower() or "police" in
detected_class.lower():
print("Siren detected")
siren_detected_event.set()    else:
siren_detected_event.clear()
```

```
time.sleep(1) # Reduce CPU usage
```

```
def detect_ambulance(model):
global ambulance_detected    while
True:        if
siren_detected_event.is_set():
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
print("Siren detected")        cap =
cv2.VideoCapture(0)           if not
cap.isOpened():
print("Camera not accessible.")
return False

    ret, frame = cap.read()
if not ret:
print("Failed to read from camera")        break

    results = model(frame)
detected_ambulance = any(
    "ambulance" in model.names[int(box.cls)].lower() or "police" in
model.names[int(box.cls)].lower()
    for result in results for box in result.bboxes
)

    if detected_ambulance:
print("Ambulance Detected")
emergency_lights_on()
time.sleep(3)

cap.release()
cv2.destroyAllWindows()

def emergency_lights_on():
print("GREEN at lane 1")
print("GREEN at lane 2")
print("RED at lane 3")  print("RED
at lane 4")
GPIO.output(2, GPIO.LOW)
GPIO.output(3, GPIO.HIGH)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.HIGH)
GPIO.output(15, GPIO.HIGH)
GPIO.output(18, GPIO.HIGH)
GPIO.output(17, GPIO.HIGH)
GPIO.output(27, GPIO.HIGH)

def count_vehicles():
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
# Assume model is trained only on "car"
classNames = ["Ambulance", "car"]

# Initialize one webcam
cap = cv2.VideoCapture(0)
cap.set(3, 1280)
cap.set(4, 720)

if not cap.isOpened():
    print("Camera failed to open.")
    exit()

# Load YOLO model
model = YOLO("../yolo-weights/main1.pt")

# Initialize SORT trackers    tracker1 = Sort(max_age=20,
min_hits=3, iou_threshold=0.3)    tracker2 =
Sort(max_age=20, min_hits=3, iou_threshold=0.3)

# Vehicle count storage
totalCount1 = []    totalCount2
= []

# Load masks    mask1 =
cv2.imread("../Videos/mask1.png")    mask2 =
cv2.imread("../Videos/mask2.png")

# Helper functions    def
process_detections(results, detections):
for r in results:        for box in r.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])            conf
= float(box.conf[0])    cls = int(box.cls[0])            if
cls<len(classNames) and classNames[cls] == "car":
                detections = np.vstack((detections, [x1, y1, x2, y2, conf]))
return detections

def count_vehicles(resultsTracker, totalCount):
for result in resultsTracker:    _, _, _, _
obj_id = result        if obj_id not in totalCount:
totalCount.append(obj_id)
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
def draw_boxes(frame, resultsTracker):
for result in resultsTracker:
    x1, y1, x2, y2, obj_id = map(int, result[:5])
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.putText(frame, f"ID: {obj_id}", (x1, y1 - 10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

# Main loop
while True:
    success, frame = cap.read()
    if not success:
        print("Failed to read frame from camera.")
        break

    # Clone frame for two regions
    img1 = frame.copy()    img2 =
    frame.copy()

    # Resize masks to match frame size    mask1_resized =
    cv2.resize(mask1, (img1.shape[1], img1.shape[0]))    mask2_resized
    = cv2.resize(mask2, (img2.shape[1], img2.shape[0]))

    # Apply masks    imgRegion1 =
    cv2.bitwise_and(img1, mask1_resized)    imgRegion2
    = cv2.bitwise_and(img2, mask2_resized)

    # YOLO detections    results1 =
    model(imgRegion1, stream=True)    results2
    = model(imgRegion2, stream=True)

    detections1 = process_detections(results1, np.empty((0, 5)))
    detections2 = process_detections(results2, np.empty((0, 5)))

    # Tracking    resultsTracker1 =
    tracker1.update(detections1)    resultsTracker2 =
    tracker2.update(detections2)

    # Counting
    count_vehicles(resultsTracker1, totalCount1)
    count_vehicles(resultsTracker2, totalCount2)
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
# Draw detections and counts
draw_boxes(img1, resultsTracker1)
draw_boxes(img2, resultsTracker2)

cv2.putText(img1, f"Count: {len(totalCount1)}", (30, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)    cv2.putText(img2,
f"Count: {len(totalCount2)}", (30, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)    # Display video feeds
cv2.imshow("Region 1", img1)    cv2.imshow("Region 2", img2)

# Break on 'q' key    if
cv2.waitKey(1) & 0xFF == ord('q'):
break

def control_traffic_lights():
while True:
    c1, c2 = count_vehicles()
    d1, d2 = c1 * 2, c2 * 2
    print(f"Lane 1: {c1} vehicles, Lane 2: {c2} vehicles")

print("GREEN at lane 1") print("NO
LIGHT at lane 2") print("GREEN at
lane 3") print("RED at lane 4")
GPIO.output(2, GPIO.LOW)
GPIO.output(3, GPIO.HIGH)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.LOW)
GPIO.output(15, GPIO.LOW)
GPIO.output(18, GPIO.HIGH)
GPIO.output(17, GPIO.HIGH)
GPIO.output(27, GPIO.HIGH)    time.sleep(5)

print("YELLOW at lane 1") print("NO
LIGHT at lane 2") print("YELLOW at
lane 3") print("RED at lane 4")
GPIO.output(2, GPIO.HIGH)
GPIO.output(3, GPIO.LOW)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.LOW)
GPIO.output(15, GPIO.HIGH)
GPIO.output(18, GPIO.LOW)
```


SMART TRAFFIC MANAGEMENT SYSTEM

```
GPIO.output(17, GPIO.HIGH)
GPIO.output(27, GPIO.HIGH)    time.sleep(2)
```

```
print("RED at lane 1")
print("NO LIGHT at lane 2") print("RED
at lane 3") print("GREEN at lane 4")
GPIO.output(2, GPIO.HIGH)
GPIO.output(3, GPIO.HIGH)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.LOW)
GPIO.output(15, GPIO.HIGH)
GPIO.output(18, GPIO.HIGH)
GPIO.output(17, GPIO.LOW) GPIO.output(27,
GPIO.HIGH)
time.sleep(d1)
```

```
print("RED at lane 1") print("NO
LIGHT at lane 2") print("RED at
lane 3") print("YELLOW at lane 4")
GPIO.output(2, GPIO.HIGH)
GPIO.output(3, GPIO.HIGH)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.LOW)
GPIO.output(15, GPIO.HIGH)
GPIO.output(18, GPIO.HIGH)
GPIO.output(17, GPIO.HIGH)
GPIO.output(27, GPIO.LOW) time.sleep(2)
```

```
print("GREEN at lane 1")
print("GREEN at lane 2") print("RED
at lane 3") print("RED at lane 4")
GPIO.output(2, GPIO.LOW)
GPIO.output(3, GPIO.HIGH)
GPIO.output(4, GPIO.LOW)
GPIO.output(14, GPIO.HIGH)
GPIO.output(15, GPIO.HIGH)
GPIO.output(18, GPIO.HIGH)
GPIO.output(17, GPIO.HIGH)    GPIO.output(27,
GPIO.HIGH)
time.sleep(d2)
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
# Initialize the siren detection event
siren_detected_event = threading.Event() if
_name_ == "_main_": model = YOLO("../yolo-
weights/yolo11n.pt")
```

```
siren_thread = threading.Thread(target=detect_siren, daemon=True) siren_thread.start()
```

```
ambulance_thread = threading.Thread(target=detect_ambulance, args=(model,), daemon=True)
ambulance_thread.start()
```

```
control_traffic_lights()
```

Verilog

```
module FPGA_BLINK_NEW(
    input A, B, C, D, E, F, G, H, CLK,    // Input signals    output reg R1, Y1,
    G1, G2, R3, Y3, G3, R4, Y4, G4, // Output signals    output reg G31, G41
    // Left green arrows for Lane 3 and Lane 4
);
    // Parameters for timing
    parameter RESET_DELAY = 24'd1000000; // Reset active duration (e.g., 10 ms for 100 MHz
    clock)
    parameter ONE_SECOND = 24'd100000000; // One-second counter (100 MHz clock)

    // Internal signals    reg [23:0] reset_counter;           // Counter for
    generating reset pulse    reg rst_n;                       // Internal reset
    signal
    reg [23:0] counter;           // Counter for 1-second delay

    // Generate automated reset pulse at power-up    always
    @(posedge CLK) begin        if (reset_counter < RESET_DELAY)
    begin            reset_counter <= reset_counter + 1; // Increment reset
    counter            rst_n <= 0; // Keep reset active low during the delay
    end else begin            rst_n <= 1; // De-assert reset after the delay
    end
    end

    // Initialize G31 and G41 at reset    always
    @(posedge CLK or negedge rst_n) begin        if
    (~rst_n) begin // Reset logic
        G31 <= 0;
        G41 <= 0;
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
        counter <= 0;
end else begin
    counter <= counter + 1; // Increment counter on each clock cycle

    // Toggle G31 every 1 second
if (counter == ONE_SECOND) begin
    counter <= 24'd0; // Reset the counter after 1 second
    G31 <= ~G31;      // Toggle G31 (Lane 3 left turn signal)
    G41 <= ~G41;
    // Toggle G41 only when G4 is green

                // Toggle G41 (Lane 4 left turn signal)
end
end
end

// Main traffic light signal logic (Combinational logic)
always @(*) begin
    // Define R1 = ABC'D'EF(G XOR H)
    R1 = A & B & ~C & ~D & E & F & (G ^ H);

    // Define Y1 = AB'C'D'EF'GH
    Y1 = A & ~B & ~C & ~D & E & ~F & G & H;

    // Define G1 = A'BC'FGH(D XOR E)'      G1
    G1 = ~A & B & ~C & F & G & H & ~(D ^ E);

    // Define G2 = A'BC'D'EF'GH
    G2 = ~A & B & ~C & D & E & F & G & H;

    // Define R3 = ABC'D'EF(G XOR H) + A'BC'D'EF'GH
    R3 = (A & B & ~C & ~D & E & F & (G ^ H)) | (~A & B & ~C & D & E & F & G & H);

    // Define Y3 = AB'C'D'EF'GH
    Y3 = A & ~B & ~C & ~D & E & ~F & G & H;

    // Define G3 = A'BC'D'E'FGH
    G3 = ~A & B & ~C & ~D & ~E & F & G & H;

    // Define R4 = A'BC'FGH(D XOR E)' + AB'C'D'EF'GH
    R4 = (~A & B & ~C & F & G & H & ~(D ^ E)) | (A & ~B & ~C & ~D & E & ~F & G & H);
```

SMART TRAFFIC MANAGEMENT SYSTEM

```
// Define Y4 = ABC'D'EFGH'  
Y4 = A & B & ~C & ~D & E & F & G & ~H;  
// Define G4 = ABC'D'EFG'H  
G4 = A & B & ~C & ~D & E & F & ~G & H;  
end  
  
endmodule
```

CHAPTER 6

RESULT

A smart traffic management system prototype was developed. Traffic control based on real-time vehicle density was successfully implemented. A Raspberry Pi 4 was used to process video input and analyze traffic patterns using computer vision algorithms. It was also utilized for detecting emergency vehicles. The Basys 3 FPGA was responsible for controlling the traffic lights. It dynamically adjusted green light durations, giving priority to roads with higher vehicle density. The emergency vehicle detection system—based on audio signal processing for siren recognition—worked effectively as a standalone module on a laptop. However, integrating this functionality into the Raspberry Pi 4 proved to be highly challenging. This difficulty was primarily due to the computational load and timing constraints associated with real-time audio processing on the Raspberry Pi. The complete system was demonstrated using a miniature model of the actual traffic junction.

CHAPTER 7

CONCLUSION

The Smart Traffic Management System successfully demonstrates how real-time traffic analysis and FPGA-based control can significantly improve traffic flow at intersections. By using a Raspberry Pi 4 to process video input and detect vehicle density with a custom YOLO model, and controlling traffic lights through a Basys 3 FPGA, the system dynamically adjusts green light durations based on actual traffic conditions. An audio-based emergency vehicle detection system was also developed and tested independently on a laptop. However, integration into the Raspberry Pi was limited by processing constraints. Despite this, the project validates the potential of combining computer vision, edge computing, and hardware-level control for intelligent traffic management. The system was showcased using a miniature junction model, proving its real-world applicability and scalability. With further refinement and integration, this approach can play a key role in smart city infrastructure by enhancing road efficiency and emergency response times.

REFERENCES

1. *Intelligent Traffic Signal Management Using Raspberry Pi and OpenCV*

1Dr. Ch. Rambabu, 2J. Divya Madhuri, 3M. Vamsi, 4D. Tharun Nayak, 5L. Pavan Prasad
1Associate Professor of Dept. of ECE, 2Student of Dept. of ECE, 3Student of Dept. of ECE,
4Student of Dept. of ECE, 5Student of Dept. of ECE. Department of Electronics and
Communication Engineering, 1Seshadri Rao Gudlavalleru Engineering College,
Gudlavalleru, Andhra Pradesh, India.

2. *TRAFFIC DENSITY DETECTION USING RASPBERRY Pi*

Yadav Prajwal Shankar¹, Dhanawale Sagar Dnyanoba², More Ajinkya Namdev³, Kumbhar
Akshay Gurudev⁴ Department of Computer Engineering, SCSCOE, Savitribai Phule Pune
University, Pune, India.^{1,2,3,4} M. B. Wagh⁵ Head of Department Dept. Computer
Engineering, SCSCOE, Savitribai Phule Pune University, Pune, India

3. *Smart Traffic Controller Implementation using FPGA*

Sowmya KB, Sagar T, N R Pavan Santosh

APPENDIX

Abhijith Ss

Smart Traffic Management System 2_removed.pdf



Mar Baselios College of Engineering and Technology

Document Details

Submission ID trn:oid:::3618:92157807

Submission Date

Apr 21, 2025, 1:01 PM GMT+5:30

Download Date

Apr 21, 2025, 1:02 PM GMT+5:30File Name

Smart Traffic Management System 2_removed.pdf

File Size

1.2 MB

45 Pages

7,367 Words





44,469 Characters

Page 1 of 51 - Cover Page




19% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

MatchGroups

-  **106 Not Cited or Quoted 16%**
Matches with neither in-text citation nor quotation marks
-  **6 Missing Quotations 2%**
Matches that are still very similar to source material
-  **4 Missing Citation 1%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 15%  Internet sources
- 15%  Publications
- 0%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

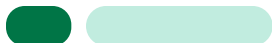


<1%



<1%

Das Gupta, Chanda.

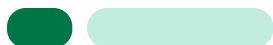


<1%

12	Internet	mjrobot.org	<1%
13	Internet	forum.beagleboard.org	<1%
14	Internet	idoc.pub	<1%
15	Internet	dspace.univ-eloued.dz	<1%
16	Internet	medium.com	<1%
17	Internet	silo.pub	<1%
18	Internet	mro.massey.ac.nz	<1%
19	Internet	www.ijcsmc.com	<1%
20	Internet	www.rajashekar.org	<1%
21	Publication	"Advances in Electrical Control and Signal Systems", Springer Science and Business Media	<1%
22	Publication	Yoshiyasu Takefuji. "PyPI: An internet-enabled learning tool to boost learner motivation"	<1%
23	Publication	K. Mohaideen AbdulKadhar, G. Anand. "Data Science with Raspberry Pi", Springer	<1%
24	Publication	D. Rajesh, M. Ganesan, S. Kumarakrishnan, P. Nidhish Kumar. "Leveraging Transfer Learning for Sentiment Analysis"	<1%

"Benchmarkin

g of



<1%

Energy and Latency of CNN Models on Ras...



<1%

pastebin.com

26

Publication

Blair,NicholasPaul."DevelopmentofaReferenceDesignforaCyber-PhysicalSys ...

<1%

27

Internet

fastercapital.com

<1%

28

Internet

git.interlegis.leg.br

<1%

29

Internet

huggingface.co

<1%

30

Internet

ses.library.usyd.edu.au

<1%

31

Internet

www.scribd.com

<1%

32

Internet

www.theseus.fi

<1%

33

Internet

documents.mx

<1%

34

Internet

ethesis.nitrkl.ac.in

<1%

35

Internet

aeh-werker.de

<1%

36

Internet

github.com

<1%

37

Internet

piembsystech.com

<1%

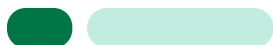
38

Publication

M.DhilsathFathima,R.Hariharan,GeethaC,EbenazerRoselinS."Chapter10Sma ...

<1%

Internet



<1%

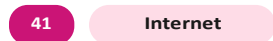
Publication

Simões, Francisco Duarte Lourenço. "Debugging Statecharts Extended With Class ..."



wiki.itcollege.æ

<1%



www.jetir.org

<1%