A REPORT ON VEHICLE SPEED MONITORING AND OVERSPEED ALERT SYSTEM USING ESP32

A Microcontrollers and Embedded Systems Project Report

Submitted in partial fulfillment of the requirements

for the award of the Degree of

Bachelor of Technology
In
Electronics and Communication Engineering
By

N. Manikanta Raghava (23485A0424)
N. Satish (22481A04H1)
M. Sushma (22481A04E1)
M. Govardhana Rao (22481A04F1)

Under the Supervision of

Dr. B. RAJA SEKHAR
(Professor & HoD)

Department of Electronics and Communication Engineering

SESHADRI RAO GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

SESHADRI RAO KNOWLEDGE VILLAGE

GUDLAVALLERU - 521356

ANDHRA PRADESH

2024-25

Department of Electronics and Communication Engineering

SESHADRI RAO GUDLAVALLERU ENGINEERING COLLEGE

SESHADRI RAO KNOWLEDGE VILLAGE

GUDLAVALLERU - 521356

CERTIFICATE

This is to certify that the Microcontrollers and Embedded Systems project report entitled " **VEHICLE SPEED MONITORING AND OVERSPEED ALERT SYSTEM USING ESP32** " is a Bonafide record of work carried out by **N.Manikanta Raghava (23485A0424),N.Satish (22481A04H1), M. Sushma (22481A04E1), M.Govardhana Rao (22481A04F1)** under my guidance and supervision in partial fulfillment of the requirements, for the award of the degree of Bachelor of Technology in **Electronics and Communication Engineering** of **Seshadri Rao Gudlavalleru Engineering.**

Dr. B. RAJA SEKHAR
Project Supervisor

Dr. B. RAJA SEKHAR
Head of the Department

Acknowledgements

We are very glad to express our deep sense of gratitude to **Dr. B. Raja Sekhar,** Professor & HoD, Electronics and Communication Engineering for guidance and cooperation for completing this project.

We convey our heartfelt thanks to him for his inspiring assistance till the end of our project. We convey our sincere and indebted thanks to our beloved Head of the Department, Electronics and Communication Engineering, **Dr. B. Raja Sekhar** for his encouragement and help for completing our project successfully.

We also extend our gratitude to our Principal, **Dr. B. Karuna Kumar** for the support and for providing facilities required for the completion of our project.

We impart our heartfelt gratitude to all the Lab Technicians for helping us in all aspects related to our project.

We thank our friends and all others who rendered their help directly and indirectly to complete our project.

N. Manikanta Raghava (23485A0424)
N. Satish (22481A04H1)
M. Sushma (22481A04E1)
M. Govardhana Rao (22481A04F1)

ABSTRACT

This project presents a practical solution for real-time vehicle speed monitoring and overspeed alerting, using the ESP32 microcontroller and infrared (IR) sensors. The system aims to detect vehicle movement and calculate its speed accurately based on sensor data, providing both local and remote alerts if the speed exceeds a defined limit. It is ideal for use in school zones, residential areas, and smart city environments where controlling vehicle speed is essential.

The system setup consists of two IR sensors positioned at a fixed distance. As a vehicle passes each sensor, the ESP32 records the time at which they are triggered. Using the known distance and the time difference, the microcontroller calculates the vehicle's speed using the formula:

Speed = Distance / Time.

If the computed speed goes beyond the set threshold, the ESP32 activates a buzzer to issue a local audible warning. Simultaneously, a real-time notification is sent via the Blynk IoT platform, allowing remote monitoring. This dual-alert mechanism ensures immediate awareness and documentation of speed violations.

The Blynk dashboard provides a live view of vehicle speed data, enabling users to observe traffic behavior in real time. It supports remote access via mobile devices, allowing users or authorities to monitor events from virtually anywhere.

The system's design is cost-effective and scalable, relying on affordable components like the ESP32 and IR sensors. It can be expanded to include more features such as GSM alerts or camera modules for more comprehensive monitoring.

By combining embedded systems with IoT technology, this project offers a reliable tool for improving road safety and traffic management. It lays the groundwork for future intelligent transportation systems that emphasize automation, monitoring, and preventive safety measures.

Table of Content

CHAPTER 1

INTRODUCTION

1.1 Introduction: Vehicle speed monitoring is an essential part of intelligent traffic systems and smart transportation infrastructure. The rapid increase in vehicular movement on roads requires efficient and reliable systems to monitor speed and alert for overspeed conditions to prevent accidents. This project aims to build a speed monitoring and overspeed alerting system using the ESP32 microcontroller. The project involves using IR sensors placed at a known distance apart. When a vehicle passes these sensors, the ESP32 calculates the time difference between interruptions and computes the speed using distance/time formula. If the speed exceeds a predefined threshold, the system alerts through a buzzer and also sends a notification to a Blynk dashboard.

1.2 Aim of the Project:

To develop an IoT-based system for monitoring vehicle speed and alerting on mobile and using buzzer when the speed exceeds a certain limit using ESP32 and Blynk.

1.3 Objective

The system is designed to detect vehicle movement using IR sensors by monitoring the interruption of an infrared beam. When a vehicle passes through the sensor's range, the time between consecutive activations is recorded. This time interval is then used to calculate the vehicle's speed based on a predefined distance or wheel rotation parameter. If the calculated speed exceeds a set threshold, a buzzer is triggered to alert nearby personnel or users of over speeding. Simultaneously, the system sends real-time speed data and overspeed alerts to the Blynk app via Wi-Fi using the ESP32 microcontroller, enabling remote monitoring and notification for safety and traffic regulation purposes.

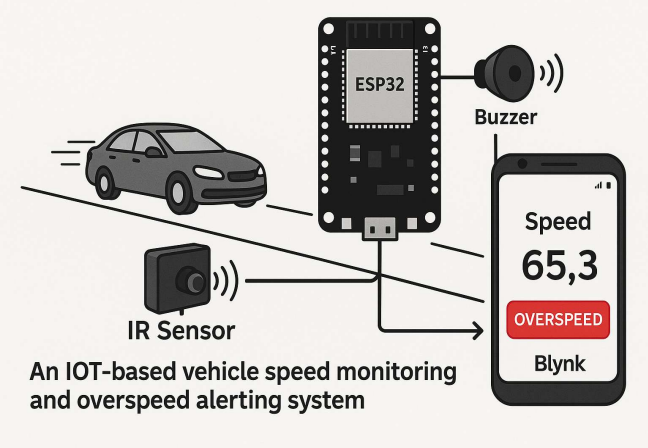


Fig.1.1 Project Objective

1.4 Software Required:

- **Arduino IDE Overview**

Arduino IDE (Integrated Development Environment) is an open-source software platform used for writing, compiling, and uploading code to Arduino boards. It provides a simple and intuitive interface that supports C/C++ programming. With built-in libraries and board management, the Arduino IDE makes it easy to prototype and develop embedded systems and IoT projects. It features a serial monitor for real-time debugging and supports a wide range of official and third-party boards through the Boards Manager.

Fig.1.2 Arduino IDE

- **Blynk Platform Overview**

Blynk is a powerful and user-friendly IoT platform that enables you to build mobile and web applications for controlling and monitoring hardware like **Arduino**, **ESP32**, **Raspberry Pi**, and more. With Blynk, you can create custom dashboards using drag-and-drop widgets, send real-time data from sensors, and control devices remotely via the Blynk mobile app or web dashboard.

Blynk supports features such as:

- **Virtual Pins** for flexible data communication
- **Device-to-cloud connectivity** via Wi-Fi, Ethernet, or cellular
- **Event notifications** (e.g., push alerts, emails)
- **Automations** and **scheduling**
- **Secure and scalable cloud infrastructure**

It's widely used for developing smart home, automation, and remote monitoring systems.

Fig.1.3 Blynk Iot

CHAPTER 2
BLOCK DIAGRAM AND EXPLANATION

2.1 Block Diagram:

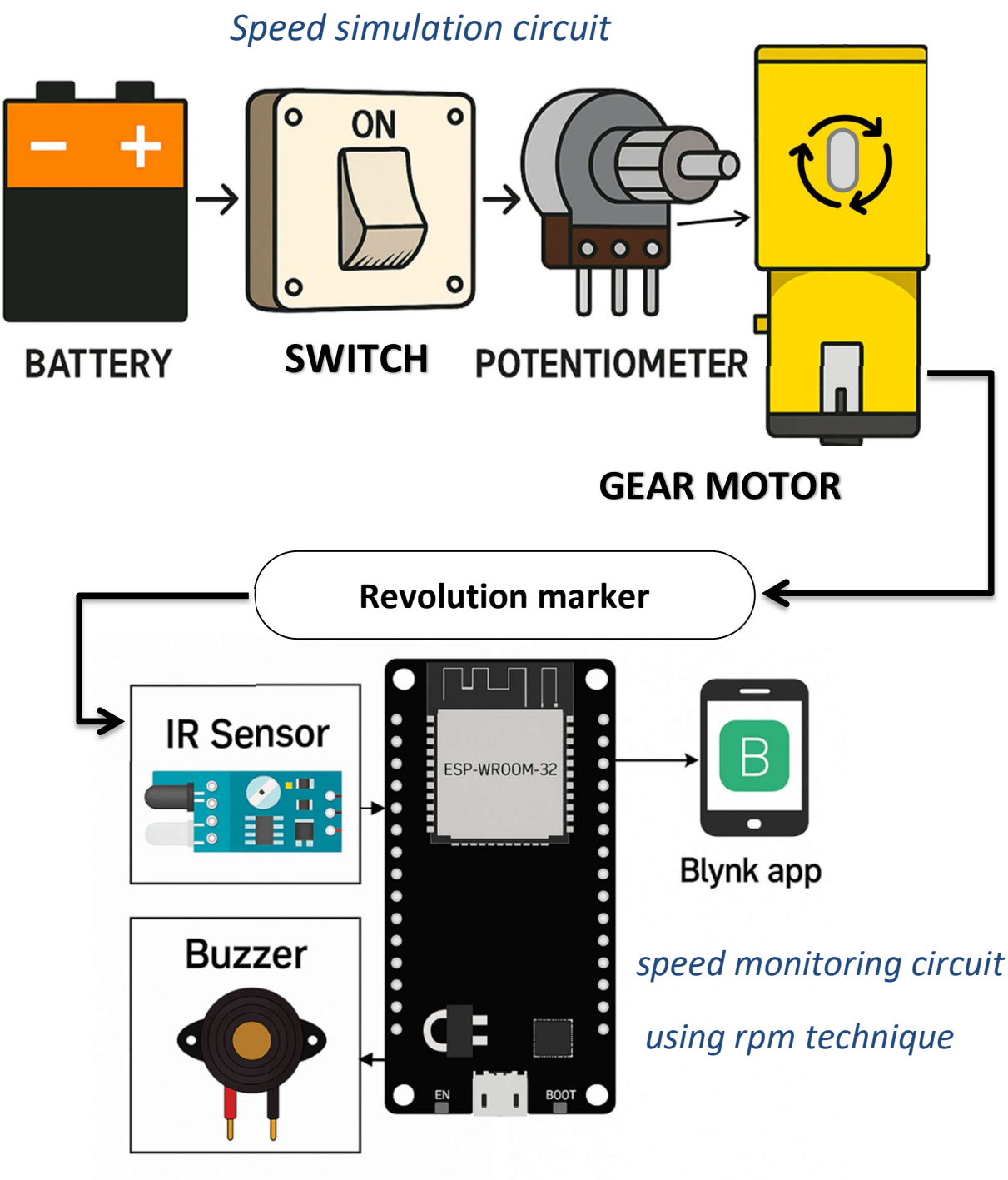


Fig.2 Block Diagram for the Project

2.2 Block Diagram Explanation:

The block diagram of the real-time motor speed monitoring and overspeed alert system begins with the **power supply**, typically a battery, which provides the necessary voltage to activate the entire setup. The circuit is controlled using a **switch (Ponitch)**, which acts as the system's power controller, turning the device ON or OFF. Once powered, a **potentiometer** is used to adjust the voltage supplied to the **DC gear motor**, simulating different motor speeds based on the resistance set by the user.

Attached to the shaft of the DC motor is a **revolution marker**, such as a reflective sticker or slot, which helps in detecting the motor's rotation. Positioned next to the motor is an **infrared (IR) sensor** that senses the passing of the marker during each full revolution. Each time the marker passes in front of the IR sensor, a signal (or pulse) is generated and sent to the **ESP-WROOM-32 microcontroller**.

The ESP-WROOM-32 receives these pulses and calculates the motor's speed in **revolutions per minute (RPM)** by measuring the time interval between the pulses. After computing the RPM, the microcontroller uses its built-in **Wi-Fi capability** to send this data wirelessly to the **Blynk mobile application**. The Blynk app provides a user-friendly interface where the real-time motor speed can be monitored remotely from a smartphone or other connected device.

Additionally, if the RPM exceeds a set threshold, the app sends a push **notification labeled "Overspeed"** to alert the user instantly—even if the app is running in the background. This push notification includes details like the time of occurrence and the measured RPM, helping the user take corrective action quickly. The notification ensures users are continuously informed about critical motor behavior, improving preventive maintenance and operational reliability. Users can also configure the alert threshold remotely via the Blynk interface, offering greater flexibility and control.

If the motor speed exceeds a predefined RPM threshold, indicating an overspeed or potential fault, the ESP-WROOM-32 triggers a **buzzer**, which gives an audible warning. This buzzer serves as a local, real-time alert mechanism, especially useful when monitoring is not being done via the mobile app.

Overall, the system integrates basic components like sensors, a microcontroller, and IoT connectivity to provide an efficient and responsive motor monitoring solution with real-time alerts, remote access, and enhanced safety. Its compact design and wireless capability make it suitable for both educational projects and industrial applications.

CHAPTER 3

COMPONENTS OVERVIEW

Components required

- ESP32-WROOM-32 Microcontroller
- IR Sensor
- DC Gear Motor
- Potentiometer
- Revolution Marker
- Buzzer
- Power Supply (Battery)
- Switch (Ponitch)
- Blynk Application (Mobile Device)

3.1 ESP32-WROOM-32 Microcontroller

The **ESP32** is a low-cost, low-power **system-on-chip (SoC)** developed by **Espressif Systems**. It features a **dual-core or single-core Tensilica Xtensa processor**, and includes built-in **Wi-Fi** and **Bluetooth (Classic and BLE)**, making it ideal for a wide range of wireless applications.

This microcontroller is known for its **high performance**, **versatility**, and **energy efficiency**, which makes it popular in **IoT (Internet of Things)** projects, **home automation**, **wearable devices**, **industrial automation**, and more.

Key features of the ESP32 include:

- Built-in Wi-Fi and Bluetooth
- Multiple GPIO pins for digital I/O
- Support for analog input (ADC) and analog output (DAC)
- PWM, SPI, I2C, UART, and CAN communication interfaces
- Timers and watchdogs for real-time applications
- Low-power sleep modes for battery-powered projects

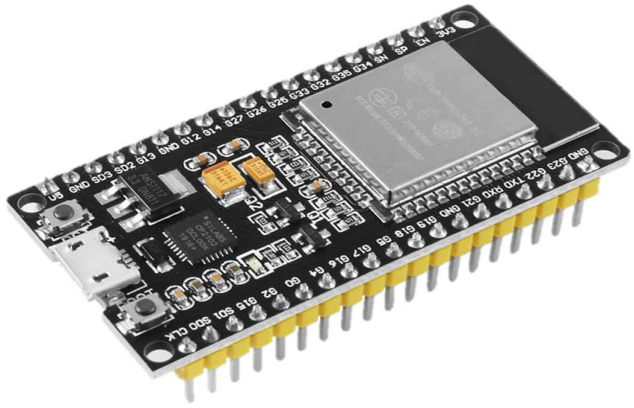


Fig. 3.1 ESP32-WROOM-32 Microcontroller

3.2 IR Sensor

An IR sensor works by detecting infrared radiation (heat). Some have an emitter that sends out an IR beam and a receiver that detects its reflection when it bounces off an object. Others are passive and only sense the IR radiation naturally emitted by objects.

Fig. 3.2 IR Sensor

3.3 DC Gear Motor

A DC gear motor uses a gearbox to reduce the output speed and increase the torque from a standard DC motor, providing controlled rotational motion. The speed can be varied using a potentiometer, which adjusts the voltage supplied to the motor. A revolution marker attached to the motor shaft allows an IR sensor to detect each full rotation. These motors are well-suited for applications requiring high torque at low speeds, offering a balance of power and control.

Fig. 3.3 DC Gear Motor

3.4 Potentiometer

A potentiometer is a variable resistor that controls motor speed by adjusting the voltage supplied to it by Changing the resistance.

Fig. 3.4 Potentiometer

3.5 Revolution Marker

The **revolution marker** is a small strip (e.g., reflective tape or black line) fixed to the motor shaft. Each time it passes the IR sensor; it generates a signal that allows the system to count revolutions for RPM Measurement.

Fig. 3.5 Revolution Marker

3.6 Buzzer

The ESP32 sends electricity to the buzzer. This electricity causes something inside the buzzer (like a small plate) to vibrate rapidly, creating sound waves that we hear as a buzz.

Fig. 3.6 Buzzer

3.7 Power Supply and Switch (Ponitch)

A **battery** provides power to the entire system, and a **Ponitch (power switch)** is used to manually control the ON/OFF state of the circuit.This ensures controlled startup and energy efficiency during operation.

Fig. 3.7 Power Supply & Ponitch (power switch)

3.8 Blynk Application (Mobile Device)

The **Blynk App** is a mobile IoT platform that displays RPM values in real-time.The ESP32 transmits data to the Blynk dashboard over Wi-Fi, allowing remote monitoring and alert notifications on your smartphone.

Fig. 3.8 Blynk Application Interface

CHAPTER 4

CIRCUIT DIAGRAM AND WORK FLOW

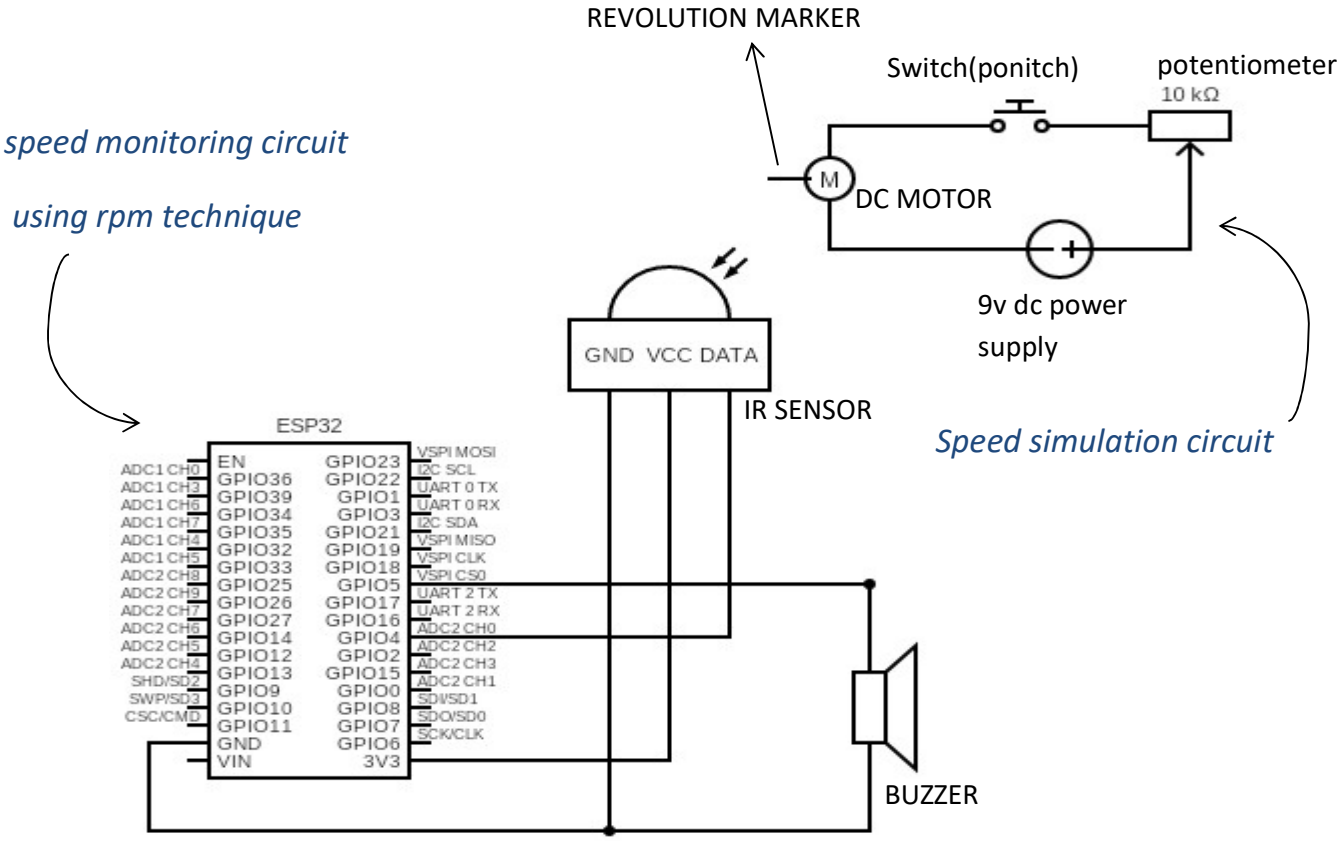


Fig.4.1 Circuit diagram

4.1Circuit Explanation

1. **IR Sensor:**
   The IR sensor detects each revolution of the DC motor using a reflective marker.Every time the marker passes, it triggers a pulse.

- **DATA pin** is connected to **GPIO4** of the ESP32.

2. **Microcontroller (ESP32):**
   The ESP32 controls the system and follows the flow shown in the diagram:

- It connects to Wi-Fi and Blynk for remote monitoring.
- On detecting a pulse from the IR sensor, it starts a timer.
- Calculates the **RPM** by measuring time between pulses.
- Then calculates **Speed** using:
  Speed = (RPM × Wheel Circumference) / 60
- It compares the speed with a preset threshold.

3. **Speed Monitoring Logic:**

- If **speed > threshold**:
  - ESP32 turns ON **Blynk LED V2 (Overspeed)** and turns OFF **V1 (Normal)**.
  - **Activates the buzzer** via **GPIO13** to give an audible overspeed alert.
- If **speed ≤ threshold**:
  - ESP32 turns ON **Blynk LED V1 (Normal)** and turns OFF **V2 (Overspeed)**.
  - **Deactivates the buzzer**, keeping it silent in normal conditions.

4. **Buzzer:**

- Connected to **GPIO13**.
- Acts as a warning system when the motor overspeeds.

5. **DC Motor:**

- Drives the shaft with the revolution marker.
- Speed is adjusted using a potentiometer.

6. **Potentiometer:**

- A 10kΩ potentiometer controls motor speed by adjusting voltage.

7. **9V Power Supply:**

- Powers the motor circuit.
- A switch is used to turn the motor ON or OFF.

Overall operation:

The ESP32 continuously monitors motor speed using the IR sensor. If overspeed is detected, it alerts the user by activating a buzzer and Blynk LED along with a notification titled as overspeed. Otherwise, it maintains normal monitoring with real-time speed display.

4.2 Hardware Setup of Circuit

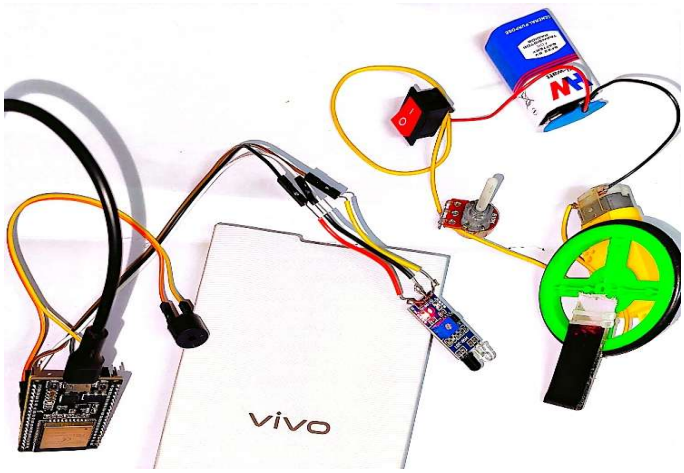
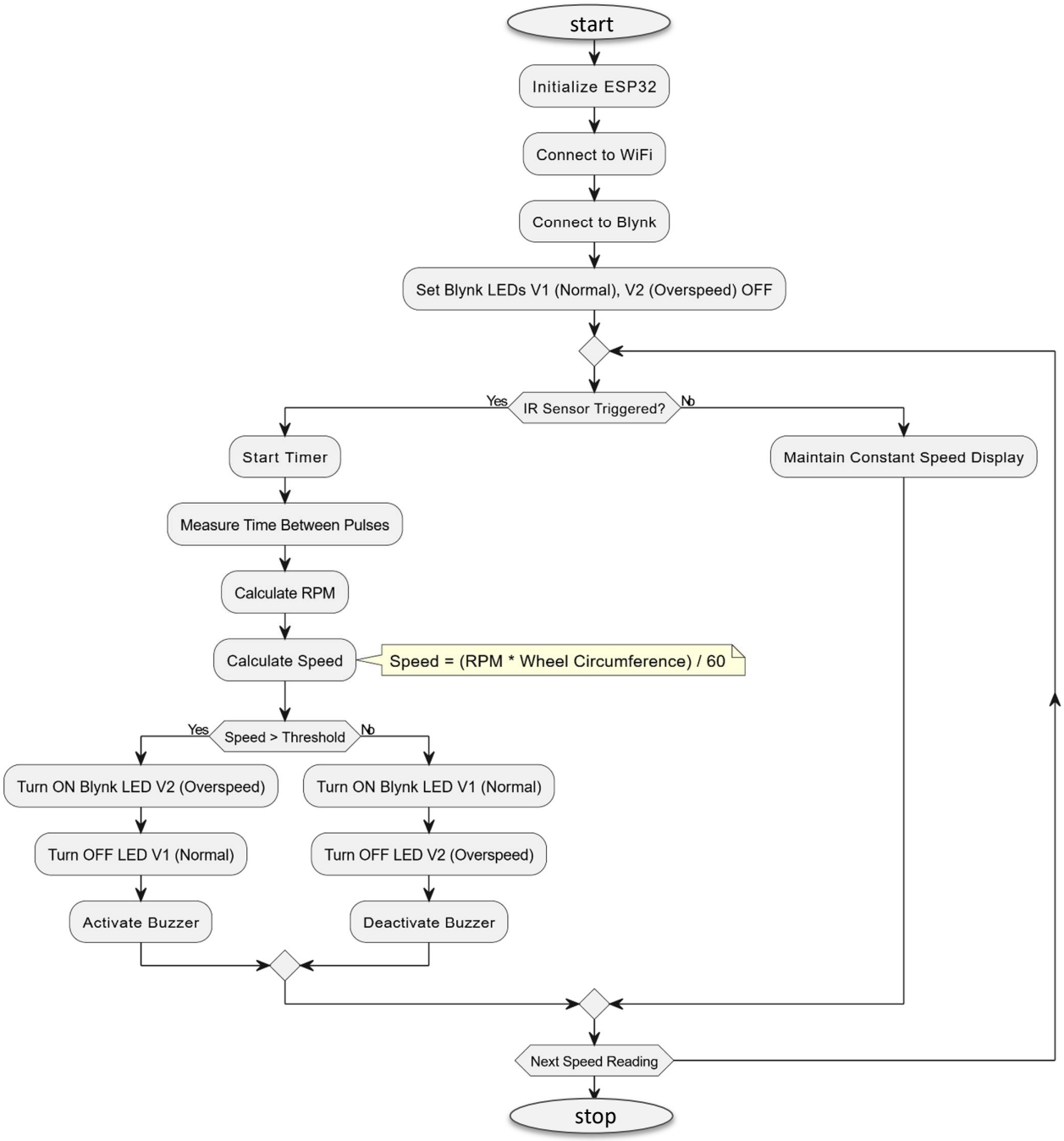


Figure 4.2: Complete Hardware Setup of the Circuit

4.3 Circuit Work Flow



CHAPTER 5

RESULT & RESULT ANALYSIS

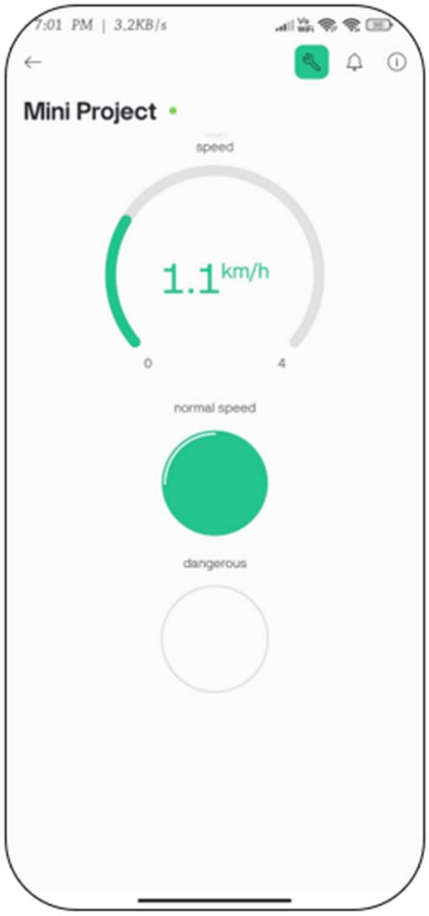


Fig 5.1 Speed Status: Normal

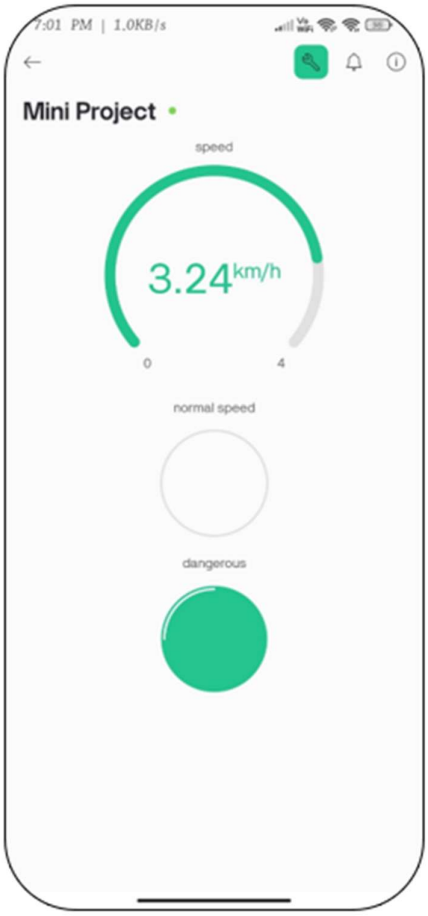


Fig 5.2 Speed Status: Overspeed

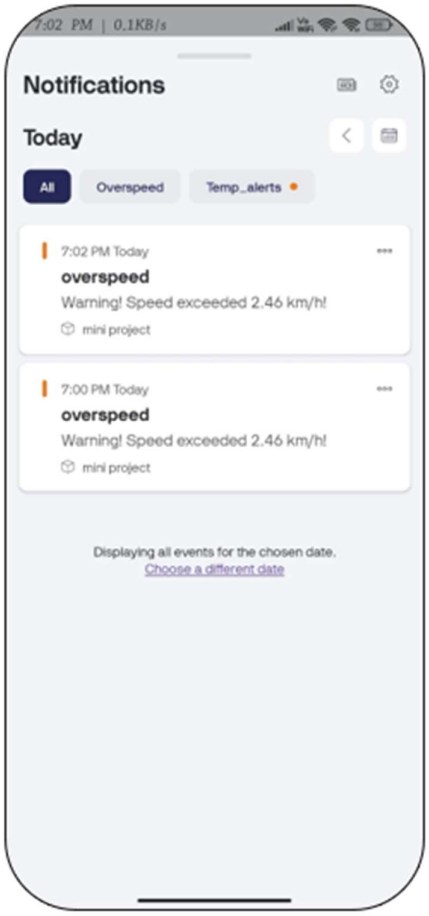


Fig 5.3 Overspeed Alerts Log

5.1 Result:

- Real-time speed monitoring works (displays km/h, classifies as Normal/Dangerous).
- **Threshold**: 2.46 km/h.
  - 3.24 km/h: Correctly triggered "Overspeed" warning.
  - 1.1 km/h: Marked as Normal.
- Logs alerts with timestamps (e.g., 7:00 PM and 7:02 PM).

5.2 Result Analysis:

- Accuracy: Correctly distinguishes Normal/Dangerous speeds.
- Threshold: 2.46 km/h validated by alerts.
- User Interface: Intuitive with colour indicators.
- Alerts: Timely warnings.
- Reliability: Consistently updates speed.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The **Vehicle Speed Monitoring and Overspeed Alert System using ESP32** demonstrates a practical and effective method for real-time speed detection and alerting using a single IR sensor. By integrating the ESP32 microcontroller with a buzzer and the Blynk IoT platform, the system successfully calculates the speed of moving vehicles and provides immediate local (buzzer) and remote (mobile app) alerts when overspeeding is detected. The use of a single IR sensor, in conjunction with a rotating wheel or reflective object, allows for RPM-based speed calculations using the formula:

Speed = (RPM × Wheel Circumference) / 60

This method ensures cost-effectiveness and simplicity without compromising on accuracy, especially in controlled environments such as test tracks, robotics, or automation systems.

The flowchart and circuit diagram provided illustrate the structured logic and practical wiring involved in implementing this system. Real-time updates and alerts through Blynk further enhance its usability for monitoring from remote locations.

Future Scope

- **Multi-Vehicle Adaptation**: The system can be expanded to monitor multiple lanes or vehicles simultaneously by incorporating multiple IR sensors with unique IDs.
- **GPS Integration**: Adding a GPS module can enhance speed accuracy over varying terrains and allow geolocation tagging of overspeed events.
- **Cloud Dashboard**: Integration with cloud platforms like Firebase or ThingSpeak could enable data logging, analytics, and historical speed records.
- **Solar Power**: For remote or highway applications, the system could be powered using solar panels for energy efficiency and independence.
- **Automatic Number Plate Recognition (ANPR)**: The system can be paired with an image recognition module for logging the license plates of overspeeding vehicles.
- **Traffic Enforcement Integration**: This setup can be linked to municipal traffic enforcement systems for automated fine generation and warning issuance.

REFERENCES

1.Interfacing IR Sensor with ESP32

- Cherry Electronics: A comprehensive tutorial on connecting an IR sensor to the ESP32, including wiring diagrams and code examples: https://cherryelectronics.in/interfacing-ir-sensor-with-esp32/
- DonSkyTech: Detailed guide on interfacing an IR sensor with ESP32, featuring practical examples and code snippets: https://www.donskytech.com/interface-esp32-with-infrared-sensor/

2. Interfacing Buzzer with ESP32

- ESP32IO: Step-by-step tutorial on using a piezo buzzer with ESP32, including wiring diagrams and code explanations: https://esp32io.com/tutorials/esp32-piezo-buzzer
- Circuits DIY: Guide on interfacing a piezo buzzer with ESP32 using a pushbutton, suitable for beginners: https://www.circuits-diy.com/esp32-tutorial-button-piezo-buzzer/

3. Calculating Speed Using RPM with IR Sensor and ESP32

- Circuit Schools: Tutorial on building a DIY tachometer using ESP32 and an IR sensor to measure RPM accurately: https://www.circuitschools.com/diy-tachometer-using-arduino-esp8266-esp32-to-measure-accurate-rpm-with-ir-sensor/
- GitHub Repository: ESP32-based tachometer project that measures RPM using an IR sensor and displays it on an LCD: https://github.com/OceanBhatnagar/Tachometer

4. Interfacing Blynk with ESP32

- Blynk Official Blueprint: Guide on blinking an LED with ESP32 using Blynk, demonstrating Wi-Fi provisioning and OTA updates. https://blynk.io/blueprints/blink-an-led-with-esp32
- SriTu Hobby: Tutorial on setting up the new Blynk app with an ESP32 board, ideal for beginners: https://srituhobby.com/how-to-set-up-the-new-blynk-app-with-an-esp32-board/

APPENDIX

Code:

```
#define BLYNK_TEMPLATE_ID "TMPL3ANItnEXt"
#define BLYNK_TEMPLATE_NAME "mini project"
#define BLYNK_AUTH_TOKEN "lcL6kZ2z2N6ZbCG1lFwKpYZpdSs01gtu"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "raghav";
char pass[] = "12345678";

const int irSensorPin = 4;   // IR Sensor Pin
const int buzzerPin = 5;     // Buzzer Pin
const float wheelCircumference = 0.132; // Wheel circumference in meters
const int markersPerRevolution = 1; // Number of markers per wheel rotation

unsigned long lastTime = 0;
unsigned long interval = 0;
bool overspeedAlert = false;
BlynkTimer timer;

// Function prototype for interrupt service routine (ISR) to avoid false triggers
void IRAM_ATTR sensorTriggered();

void setup() {
  Serial.begin(115200);
  pinMode(irSensorPin, INPUT_PULLUP);
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, LOW); // Buzzer OFF initially
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("\nWiFi Connected!");
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  attachInterrupt(digitalPinToInterrupt(irSensorPin), sensorTriggered, FALLING);
}

void loop() {
  Blynk.run();
  checkSpeed();
}

void IRAM_ATTR sensorTriggered() {
  unsigned long currentTime = micros();
  if (currentTime - lastTime > 10000) { // Ignore rapid false triggers
    interval = currentTime - lastTime;
    lastTime = currentTime;
  }
}

void checkSpeed() {
  static unsigned long lastValidTime = 0;
  unsigned long currentTime = micros();
  if (digitalRead(irSensorPin) == LOW) {
    if (currentTime - lastValidTime < 10000) {
      return; // Debounce - Ignore fast repeated signals
    }
    lastValidTime = currentTime;
    if (interval > 0) {
      float rpm = 60000000.0 / (interval * markersPerRevolution);
      float speed = (rpm * wheelCircumference) / 60.0;
      float speedKmH = speed * 3.6;
      Serial.print("Speed (km/h): ");
      Serial.println(speedKmH);
      Blynk.virtualWrite(V0, speedKmH);
      if (speedKmH > 2.46) {
        if (!overspeedAlert) {
```

```
        Blynk.logEvent("overspeed", "Warning! Speed exceeded 2.46 km/h!");
        overspeedAlert = true;
       }
       Blynk.virtualWrite(V1, 0);
       Blynk.virtualWrite(V2, 255);
       buzzerSiren();
      } else {
       overspeedAlert = false;
       Blynk.virtualWrite(V1, 255);
       Blynk.virtualWrite(V2, 0);
       digitalWrite(buzzerPin, LOW);
      }
    }
  }
}

void buzzerSiren() {
  for (int i = 0; i < 3; i++) {
    // First siren tone
    tone(buzzerPin, 440);  // Lower pitch
    delay(500);
    noTone(buzzerPin);
    delay(100);
    // Second siren tone
    tone(buzzerPin, 880);  // Higher pitch
    delay(500);
    noTone(buzzerPin);
    delay(100);
  }
}
```