AUTOMATED OVERSPEEDING DETECTION

Presented To:
Doc. Gil B. Barte
ECE 421 Faculty In Charge
The National Engineering University
Alangilan, Batangas City

In Partial Fulfillment
of the Requirements for
ECE 415 - Microprocessor and Microcontroller Systems and Design
Bachelor of Science in Instrumentation and Control Engineering

Presented By: Group 5
Alcaide, Emmanuel L.
Maglinao, John Christoper L.
Sabile, Carl Maverick M.

MAY 23, 2025

INTRODUCTION:

Road safety is a growing concern worldwide. Speeding causes many accidents, so keeping roads safe is something we all need to care about. Our project, Automated Overspeeding Detection, offers a practical solution to help drivers stay within safe speed limits and lower accident risks. The system uses GPS and microcontrollers to constantly monitor the vehicle's speed, detecting overspeeding immediately.

When the vehicle exceeds a preset speed limit, the system immediately alerts the driver with visual and sound warnings. At the same time, it uses IoT technology to notify traffic authorities about the violation, allowing for faster response and better enforcement of speed regulations.

This community-based project focuses on improving road safety by combining easy-to-use technology with real-time monitoring and alerts. By helping drivers stay within safe speed limits and enabling authorities to respond quickly, our system aims to reduce accidents and make roads safer for everyone.

GENERAL OBJECTIVES:

- To develop an automated system that monitors vehicle speed in real-time and alerts both the driver and authorities when the vehicle exceeds the set speed limit.

SPECIFIC OBJECTIVES:

- To measure the vehicle's real-time speed using a GPS module.
- To activate a buzzer and red LED when the speed limit is exceeded.
- To implement IoT-based alerts that notify traffic authorities of overspeeding incidents for better road safety management.

WORKING PRINCIPLE:

The Automated Overspeeding Detection system is designed to keep drivers safe and help reduce accidents in the community. It uses a GPS module (GY-NEO6MV2) to receive signals from satellites and calculate the vehicle's speed by measuring how much its position changes over time. This speed data is sent to the ESP32 microcontroller, which continuously compares it to a preset speed limit stored in its memory. When the vehicle is within the safe speed limit, a green LED lights up to show normal speed.

If the vehicle exceeds the speed limit, the system activates a red LED and sounds a buzzer to warn the driver to slow down. The current speed is also displayed on an LCD screen to keep the driver informed. The microcontroller constantly updates the speed data to provide real-time monitoring. This setup helps drivers stay within safe speeds and avoid accidents caused by overspeeding.

Additionally, the system uses IoT technology to automatically alert the traffic authority when the vehicle overspeeds. This helps improve road safety by allowing authorities to respond quickly to speeding violations.

PROJECT COMPONENTS:

1. **ESP32 Microcontroller**: Control sensors and send data.
2. **GPS Module (GY-NEO6MV2):** Tracks the vehicle's location and calculates its real-time speed.
3. **LCD Display(I2C)**: To display the vehicle's current speed.
4. **Buzzer(Active buzzer)**: Provides a warning sound when the speed exceeds the limit.
5. **Led Light**: For visual alert
    - Red - Overspeed detection
    - Green - Normal speed detection
6. **Power Supply**: Powers the ESP32 and other components.

- **CONSTRAINTS:**

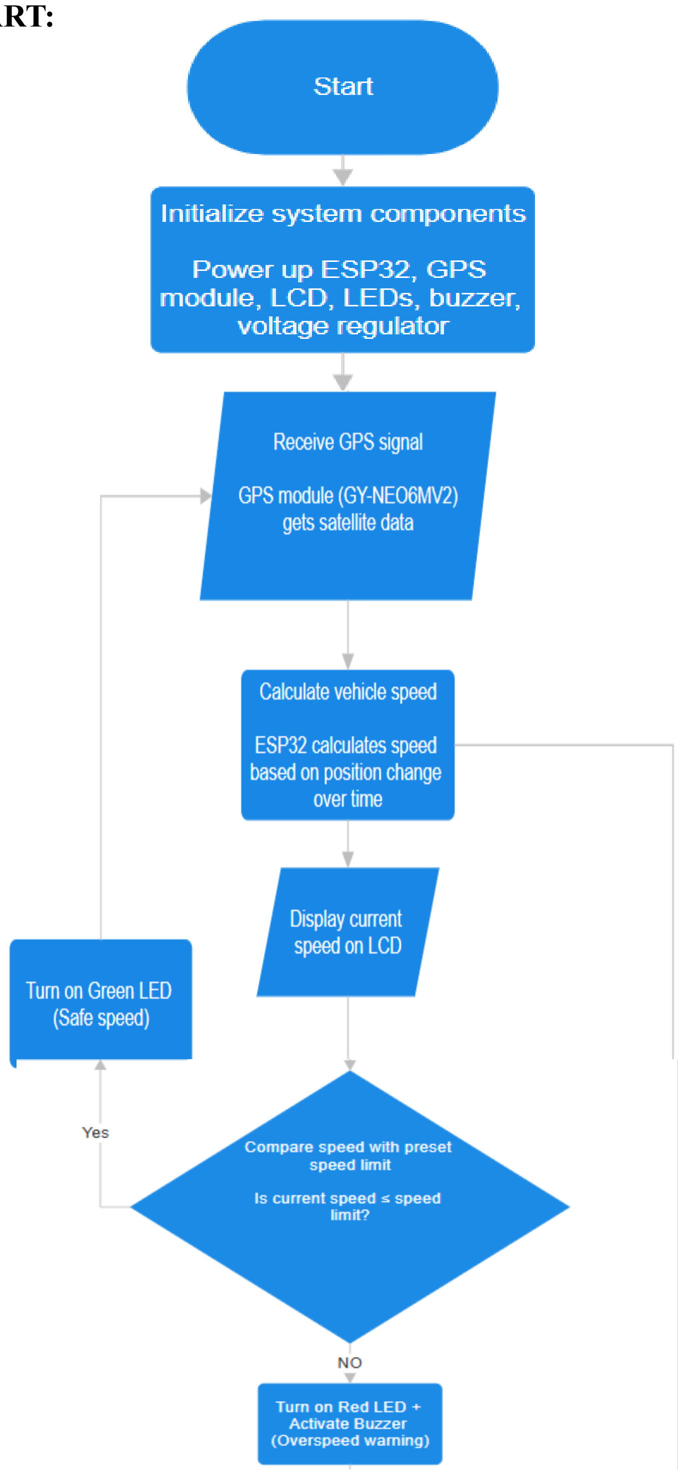
1. **Dependence on stable GPS signal:** GPS accuracy may be compromised in tunnels, near tall buildings, or in remote areas, affecting speed calculations.
2. **Variable speed readings:** Changes in GPS signal quality and weather conditions can affect speed measurement accuracy.
3. **Fixed speed limit setting:** The system uses a preset speed threshold and does not automatically adjust for varying road speed zones.
4. **Limited alert information:** Alerts sent to authorities contain only basic data like speed and location, lacking driver identification or vehicle number.

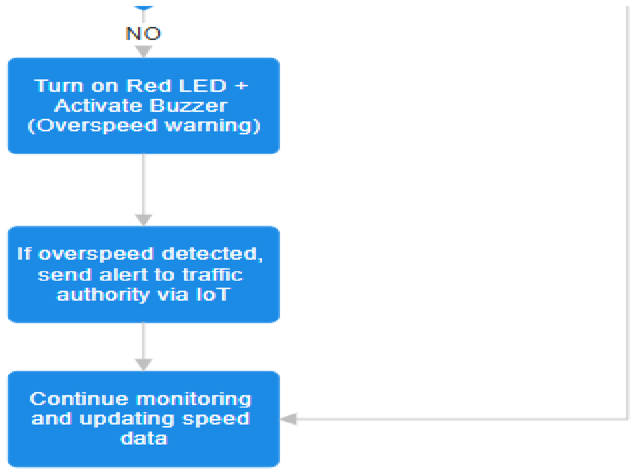
- **CONSIDERATIONS:**

In designing the Automated Overspeeding Detection system, it's important to consider the quality and strength of the GPS signal to ensure accurate speed readings. The power supply must be stable and sufficient to support all components without causing interruptions. Choosing durable hardware that can withstand vibrations, heat, and weather changes inside a vehicle is essential for long-term reliability. The system should be easy to install and maintain so users can manage it without technical difficulties. The speed limit setting must be clear and adjustable to fit different road regulations and environments. Ensuring the IoT connection via the Blynk app is secure helps protect data and prevent unauthorized access. The alerts sent to authorities should be timely and contain enough information for a quick response without violating user privacy.

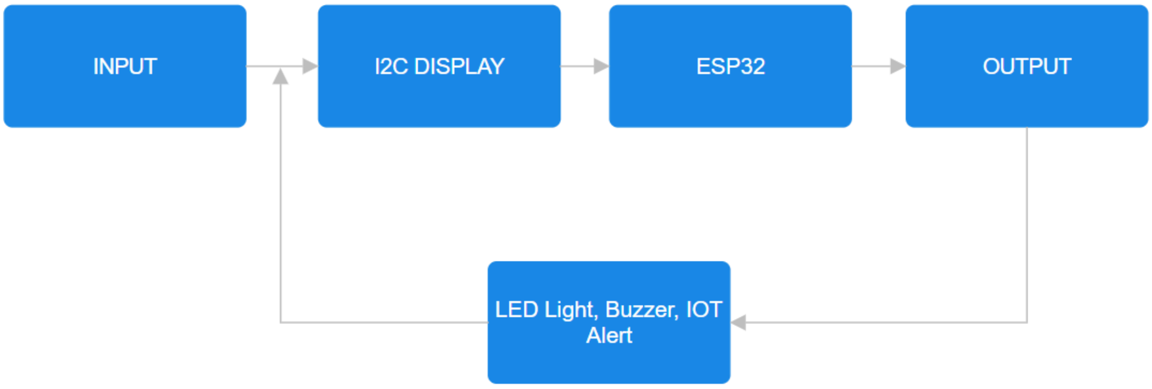
Visual and audio warnings inside the vehicle need to be noticeable but not distracting to the driver. The system's components should be cost-effective to make the solution affordable for widespread use.

- **FLOWCHART:**





- **BLOCK DIAGRAM:**



The block diagram represents the flow of a vehicle speed monitoring and alert system. This diagram illustrates a speed monitoring system using an ESP32 microcontroller. It begins with input sensors, such as GPS or IR sensors, which detect the vehicle's speed and send the data into the system. The detected speed is shown on an I2C display, providing real-time feedback to the user. This display uses a simple two-wire connection, making it efficient for embedded systems. The ESP32 processes the speed data and checks whether it exceeds a predefined limit. If the speed is too high, it triggers the system's output components. The output includes an LED light, a buzzer, and IoT alerts. These components activate to warn the user of overspeeding and can also send alerts to external devices or platforms, enhancing safety and awareness.

DOCUMENTED CODE: (FOR ACTUAL)

```
#include <LiquidCrystal_I2C.h>  // Library for I2C LCD display
#include <Wire.h>               // Library for I2C communication protocol
#include <TinyGPSPlus.h>        // Library to parse GPS data from NMEA sentences
#include <HardwareSerial.h>     // Hardware serial library for ESP32
#include <WiFi.h>               // WiFi library for ESP32
#include <BlynkSimpleEsp32.h>   // Blynk library for ESP32 devices

// --- Blynk Credentials ---
#define BLYNK_TEMPLATE_ID "TMPL6iRgbMrVV"      // Your Blynk template ID
#define BLYNK_TEMPLATE_NAME "Over Speeding Alert" // Blynk template name (for info)
#define BLYNK_AUTH_TOKEN "HsMkHtJyz7IF2dBNyk_TN_KG4hTetEBu" // Your Blynk auth token

// --- WiFi Credentials ---
char ssid[] = "Wokwi-GUEST";    // WiFi SSID to connect to
char pass[] = "";               // WiFi password (empty if open network)

// --- Pin Definitions ---
#define GPS_RX_PIN 4            // GPIO pin 4 to receive GPS data (RX)
#define GPS_TX_PIN 16           // GPIO pin 16 to transmit to GPS (TX)
#define GREEN_LED_PIN 13        // GPIO pin 13 controls Green LED (safe speed)
#define RED_LED_PIN 12          // GPIO pin 12 controls Red LED (overspeed warning)
#define BUZZER_PIN 14           // GPIO pin 14 controls buzzer (alarm sound)

// --- Objects ---
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Create LCD object at I2C address 0x27, 16x2 size
TinyGPSPlus gps;                     // Create TinyGPSPlus object to parse GPS sentences
HardwareSerial SerialGPS(1);         // Create hardware serial port 1 for GPS communication

// --- Variables ---
float Lat = 0;                  // To store GPS latitude
float Lng = 0;                  // To store GPS longitude
float actualSpeed = 0;          // To store current speed in km/h from GPS
float speedThreshold = 100;     // Speed limit threshold for overspeeding detection (km/h)

unsigned long lastGpsPrint = 0;       // Last timestamp when GPS data was printed to Serial
const unsigned long GPS_PRINT_INTERVAL = 1000; // Interval (1 second) to print GPS info

unsigned long overspeedStartTime = 0; // Time when overspeeding started (millis)
bool overspeeding = false;            // Flag to track if currently overspeeding
bool blynkNotified = false;           // Flag to avoid repeated Blynk notifications
const unsigned long OVERSPEED_DURATION = 5000; // Minimum duration to trigger Blynk alert (5
sec)

void setup() {
  Serial.begin(115200);           // Start serial monitor communication at 115200 baud
  lcd.init();                     // Initialize the LCD
  lcd.backlight();                // Turn on the LCD backlight
```

```
  // Set output pins mode for LEDs and buzzer
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  // Turn off all indicators initially
  digitalWrite(GREEN_LED_PIN, LOW);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(BUZZER_PIN, LOW);

  // Initialize GPS serial port with 9600 baud rate and custom RX/TX pins
  SerialGPS.begin(9600, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);

  // Initialize I2C for LCD with default SDA=21, SCL=22 pins
  Wire.begin(21, 22);

  delay(3000);                    // Wait 3 seconds for GPS module startup

  Serial.println("Connecting to WiFi and Blynk...");
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);  // Connect to WiFi and Blynk server
}

void loop() {
  Blynk.run();                    // Handle communication with Blynk server

  unsigned long currentMillis = millis();  // Get current system time in ms

  // Read all available GPS characters and feed them to TinyGPS parser
  while (SerialGPS.available()) {
    char c = SerialGPS.read();       // Read one character from GPS serial buffer
    if (gps.encode(c)) {             // Feed to parser; returns true if new valid data
      if (gps.location.isValid()) {  // If location is valid, update latitude and longitude
        Lat = gps.location.lat();
        Lng = gps.location.lng();

        if (gps.speed.isValid()) {   // If speed is valid, update actualSpeed
          actualSpeed = gps.speed.kmph();
        }
      }
    }
  }

  // Print GPS data to Serial Monitor every 1 second
  if (currentMillis - lastGpsPrint >= GPS_PRINT_INTERVAL) {
    if (gps.location.isValid()) {
      Serial.print("Lat: "); Serial.print(Lat, 5);        // Print latitude with 5 decimals
      Serial.print(" , Lng: "); Serial.print(Lng, 5);     // Print longitude with 5 decimals
```

```
    Serial.print(" , Speed: "); Serial.print(actualSpeed); // Print speed km/h
    Serial.println(" km/h");
  } else {
    Serial.println("Waiting for GPS fix...");         // No GPS fix yet
  }
  lastGpsPrint = currentMillis;                       // Update last print timestamp
}

// Send current speed to Blynk virtual pin V0 for dashboard gauge widget
Blynk.virtualWrite(V0, actualSpeed);

// Overspeed detection: if speed exceeds threshold
if (actualSpeed >= speedThreshold) {
  digitalWrite(GREEN_LED_PIN, LOW);      // Turn off green LED (safe indicator)
  digitalWrite(RED_LED_PIN, HIGH);       // Turn on red LED (warning indicator)
  digitalWrite(BUZZER_PIN, HIGH);        // Turn on buzzer

  if (!overspeeding) {                   // If just started overspeeding
    overspeeding = true;                 // Set overspeeding flag
    overspeedStartTime = currentMillis;  // Record overspeed start time
  } else if ((currentMillis - overspeedStartTime >= OVERSPEED_DURATION) && !blynkNotified) {
    // If overspeeding for more than 5 seconds and no notification sent yet
    Blynk.logEvent("overspeed_alert", "Speed exceeded 100 km/h for over 5 seconds.");
    blynkNotified = true;                // Set flag to avoid multiple notifications
  }

} else {
  // If speed is below threshold (normal speed)
  digitalWrite(GREEN_LED_PIN, HIGH);     // Turn on green LED (safe)
  digitalWrite(RED_LED_PIN, LOW);        // Turn off red LED
  digitalWrite(BUZZER_PIN, LOW);         // Turn off buzzer
  overspeeding = false;                  // Clear overspeed flag
  blynkNotified = false;                 // Reset notification flag
}

// Display speed on LCD - clear previous content by overwriting spaces
lcd.setCursor(0, 0);
lcd.print("Speed:         ");            // Label on first line with spaces to clear old digits
lcd.setCursor(7, 0);
lcd.print(actualSpeed, 1);               // Print speed with 1 decimal place
lcd.print("km/h");                       // Print speed unit

// Display status message on second line
lcd.setCursor(0, 1);
if (actualSpeed >= speedThreshold) {
  lcd.print("Status: Overspeeding ");
} else {
  lcd.print("Status: Normal      ");
```

```
  }

  delay(500); // Wait 0.5 seconds before next loop iteration
}
```

SCHEMATIC DIAGRAM:

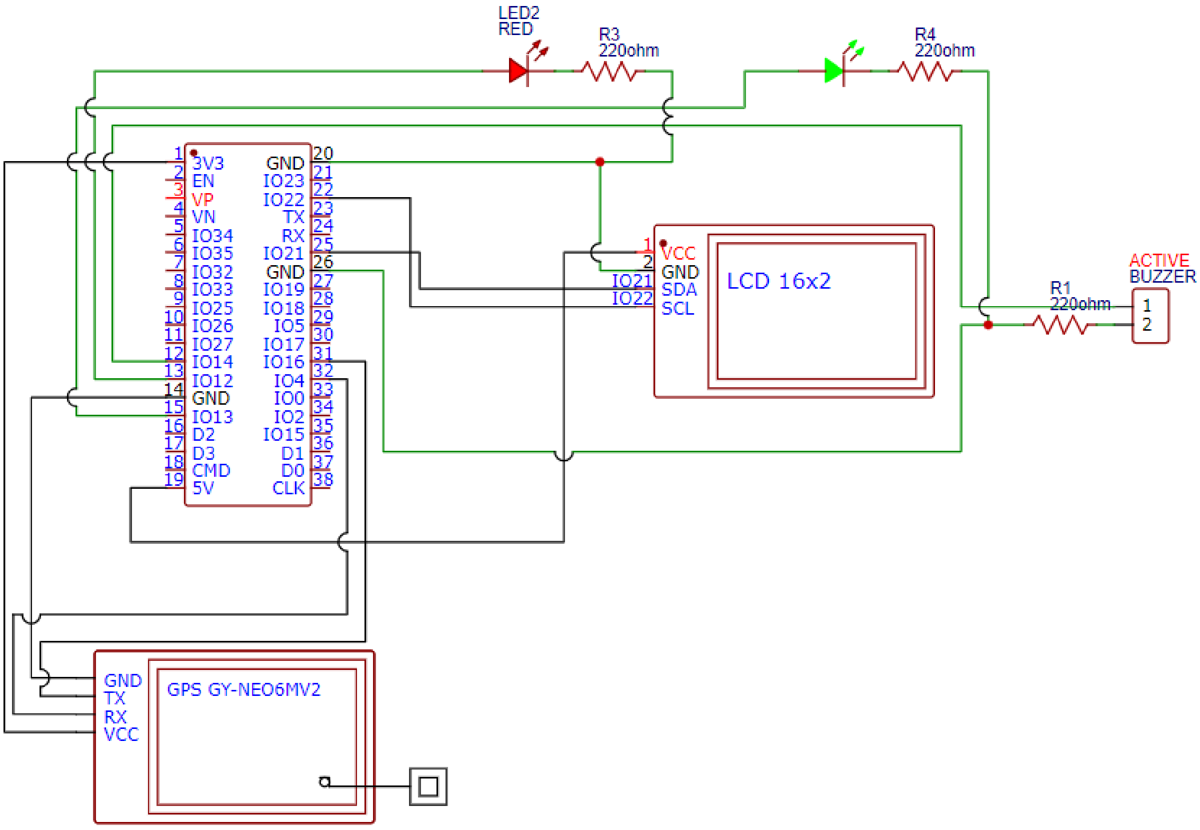


Figure 1.1: Schematic Diagram for Over speed Detection

The schematic diagram shows the configuration of the Automated Overspeeding Detection System, which is built around the ESP32 microcontroller. The system uses a GPS module (GY-NEO6MV2) to gather real-time location and speed data. The module transmits this data to the ESP32 via its TX pin. The microcontroller processes this data and compares the current speed with a preset speed limit stored in its memory. If the vehicle is operating within the safe speed range, the ESP32 activates a green LED as an indicator. However, if the speed exceeds the limit, it turns on a red LED and an active buzzer to alert the driver about overspeeding.

The system also includes a 16x2 LCD display with an I2C interface, which is connected to the ESP32 via the SDA (GPIO21) and SCL (GPIO22) pins. This display shows the current speed of the vehicle in real-time. The red and green LEDs, as well as the buzzer, are each connected in series with 220-ohm resistors to limit the current and protect the components. All devices are powered through the ESP32's onboard power pins (3.3V and GND). Additionally, the ESP32 is programmed to send speed notifications through the Blynk IoT platform when overspeeding occurs, allowing remote monitoring and helping improve road safety.

GY-NEO6MV2 GPS Module:

Compared to other components, the GY-NEO6MV2 GPS module is used because it offers an excellent balance of accuracy, simplicity, and cost-effectiveness. Unlike modules that combine GPS with cellular communication, such as the A9G, which add complexity and higher power consumption, the GY-NEO6MV2 focuses solely on providing reliable GPS positioning with a straightforward serial interface. This simplicity makes it easier to integrate into our project without the need to manage cellular network connections or additional protocols. Furthermore, while newer GPS modules may offer enhanced accuracy or multi-constellation support, the GY-NEO6MV2 provides sufficient precision for our application at a lower cost, making it ideal for efficient and budget-friendly development. Overall, the GY-NEO6MV2 is chosen because it meets our project's requirements for dependable location tracking while minimizing complexity and expense.

LCD DISPLAY (I2C):

We need the LCD I2C display to show key information, like the vehicle's speed and whether it's overspeeding, and the LCD does this job efficiently with clear text that's easy to read. LCD I2C display is a more practical and reliable choice for displaying essential information without compromising on performance or ease of use. We also chose the LCD I2C display because we have a background in using it, and we've already practiced using it for our laboratory report, so we're familiar with its setup and operation.

The reason we chose the LCD I2C display is its simplicity. It's widely used and compatible with the ESP32, making it easier to program and integrate into the system. Another reason is power consumption. Since our project aims to keep the system running efficiently over long periods, especially with the ESP32, the lower power consumption of the LCD I2C display becomes a key advantage, ensuring that the system remains power-efficient and sustainable.

Supply Voltage (V): 5 volts

Typical Current Draw: ~20 mA

POWER:

$$P = V \times I \qquad P = 5V \times 0.020A$$

$$P = 0.100 \text{ watts}$$

RED LED (OVERSPEEDING INDICATOR):

The red LED in our system serves as the overspeeding indicator and is powered by a 3.3V GPIO pin from the ESP32. Given its forward voltage of 2.0V, the remaining 1.3V drops across the 220Ω resistor. This results in a current of approximately 5.91 mA flowing through the circuit. From a power perspective, the LED consumes about 11.8 milliwatts (mW), while the resistor dissipates around 7.7 mW. These values show that the LED operates safely within limits, drawing minimal current from the ESP32 while still providing a clear visual alert for overspeeding detection.

Supply Voltage (Vs) = 3.3 V

Forward Voltage of LED (Vf) = 2.2 V

Resistor Value (R) = 220 Ω

Voltage Across Resistor (Vr):

$$Vr = Vs - Vf = 3.3V - 2.0V = 1.3V$$

Current Through LED:

$$I = \frac{Vr}{R} = \frac{1.3V}{220\ \Omega} \approx 0.00591\ A = 5.91\ \text{mA}$$

Power Dissipated by the LED:

$$P_{LED} = Vf \times I = 2.0V \times 0.00591A \approx 0.0118\ W = 11.8\ \text{mW}$$

Power Dissipated by the Resistor:

$$P_{Resistor} = Vr \times I = 1.3V \times 0.00591A \approx 0.0077\ W = 7.7\ \text{mW}$$

GREEN LED (NORMAL SPEED INDICATOR):

This green LED lights up to show that everything is running at normal speed. It's powered by a 3.3V supply, and a resistor helps control the flow of electricity. The voltage is split, 2.2V goes to the LED, and the remaining 1.1V drops across the resistor. This setup allows about 5 milliamps of current to flow. The LED uses 11 milliwatts of power to shine, while the resistor safely uses 5.5 milliwatts. It's a simple but efficient setup that gets the job done.

Supply Voltage (Vs) = 3.3 V

Forward Voltage of LED (Vf) = 2.2 V

Resistor Value (R) = 220 Ω

Voltage across the resistor (Vr):

$$V_r = V_s - V_f = 3.3V - 2.2V = 1.1V$$

Current through the circuit (I):

$$I = \frac{V_r}{R} = \frac{1.1V}{220\ \Omega} = 0.005\ A = 5.00\ mA$$

Power consumed by the LED (P_LED):

$$P_{LED} = V_f \times I = 2.2V \times 5.00\ mA = 11.0\ mW$$

Power consumed by the resistor (P_Resistor):

$$P_{Resistor} = V_r \times I = 1.1V \times 5.00\ mA = 5.5\ mW$$

BUZZER (FOR ALERT):

The buzzer operates on a 3.3V supply and is paired with a 220-ohm resistor to safely control current flow. The buzzer drops around 2.0V, while the resistor handles the remaining 1.3V. This setup results in a current of about 5.91 mA. The buzzer consumes roughly 11.8 milliwatts of power to produce sound, and the resistor dissipates 7.7 milliwatts to keep the circuit stable and safe.

Supply Voltage (Vs) = 3.3 V

Voltage Drop Across Buzzer (Vbuzzer) = 2.0 V

Voltage Across Resistor (Vr) = 1.3 V

Resistor Value (R) = 220 Ω

1. **Current (I):**

Using Ohm's Law:

$$I = \frac{V_r}{R} = \frac{1.3V}{220\ \Omega} \approx 0.00591\ A = 5.91\ mA$$

---

2. **Power Consumed by Buzzer (P_buzzer):**

$$P = V \times I = 2.0V \times 5.91\ mA = 11.82\ mW \approx 11.8\ mW$$

3. **Power Dissipated by Resistor (P_resistor):**

$$P = V \times I = 1.3V \times 5.91\ mA = 7.683\ mW \approx 7.7\ mW$$

Computation of Speed from GPS Data Using TinyGPSPlus Library

This section describes the methodology used to compute the vehicle speed from GPS data acquired via the GY-NEO6MV2 GPS module. The speed data is extracted and converted using the TinyGPSPlus library, which processes NMEA sentences transmitted by the GPS module.

GPS Module Output and NMEA Sentences:

The GPS module outputs navigational data in the form of NMEA (National Marine Electronics Association) sentences. Among these, the Recommended Minimum Specific GPS/Transit Data sentence, denoted as $GPRMC , contains essential information including time, latitude, longitude, date, and speed. The speed component within the $GPRMC sentence is reported in units of knots, where one knot is defined as one nautical mile per hour.

An example of a $GPRMC sentence is as follows:

$GPRMC,123519,A,13.7841,N,121.0742,E,**60.0**,084.4,250525,003.1,W*6A

In this example, the speed value indicated is 60.0 knots.

Breakdown of $GPRMC sentence:

Field#	Example Value	Description
1	$GPRMC	Sentence type
2	123519	UTC time (12:35:19)
3	A	Status (A = Active, V = Void)
4	13.7841	Latitude
5	N	N/S Hemisphere
6	121.0742	Longitude
7	60.0	Speed over ground(in knots)
8	084.4	Track angle in degrees
9	250525	Date(25 May 2025)
10	003.1	Magnetic variation
11	W	Direction of variation
12	*6A	Checksum

Parsing NMEA Sentences Using TinyGPSPlus

The TinyGPSPlus library processes the incoming serial data stream character-by-character. Each received character “c” is passed to the parser function as follows:

gps.encode(c);

The parser accumulates these characters until a complete and valid NMEA sentence is formed and verified. Upon successful parsing, the library updates its internal data structures with the latest navigation parameters, including speed.

Extraction and Conversion of Speed Data

The speed extracted from the $GPRMC sentence is initially in knots. To convert this to a more commonly used unit, kilometers per hour (km/h), the following conversion factor is applied:

$$\mathbf{Speed_{km/h} = Speed_{knots} \times 1.852}$$

This conversion is conveniently handled by the TinyGPSPlus library via the function:

float actualSpeed = gps.speed.kmph();

Where actualSpeed represents the vehicle's speed in kilometers per hour as a floating-point value.
For example, a speed reading of 60.0 knots from the GPS translates to:

$$\mathbf{60.0_{(knots)} \times 1.852 = 111.12 km/h}$$

Thus, the vehicle is traveling at approximately 111.12 kilometers per hour.

Definition and Context of a Knot

A knot is a unit of speed that corresponds to one nautical mile per hour. The nautical mile is defined based on the Earth's circumference and is commonly used in maritime and aviation navigation. By definition,

$$\mathbf{1\ knot = 1.852\ km/h}$$

Hence, speed values reported by GPS modules in knots require conversion for standard road vehicle speed measurements.

Testing Procedure using Wokwi:
https://wokwi.com/projects/431755227981624321

A. Normal Speed Detection

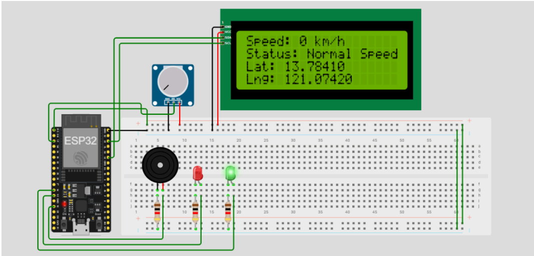


Fig. 1.1 Normal Speed Status

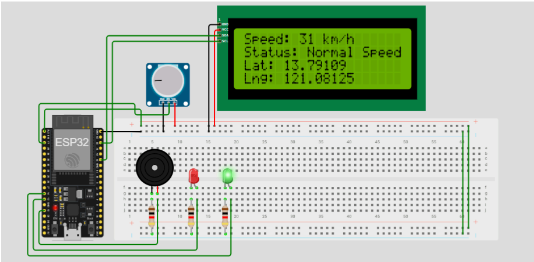


Fig. 1.2 Normal Speed Status

B. Overspeed Detection

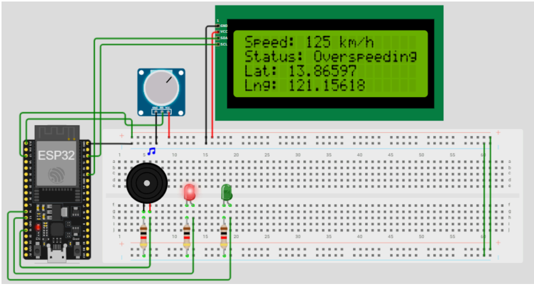


Fig. 1.3 Overspeed Status

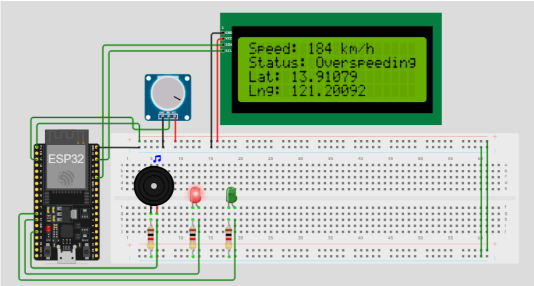
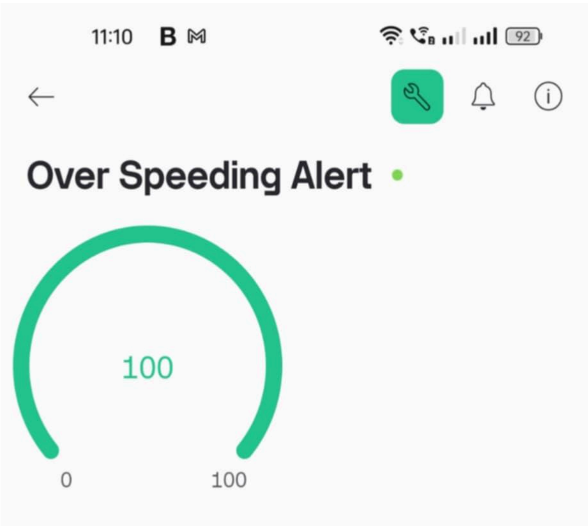
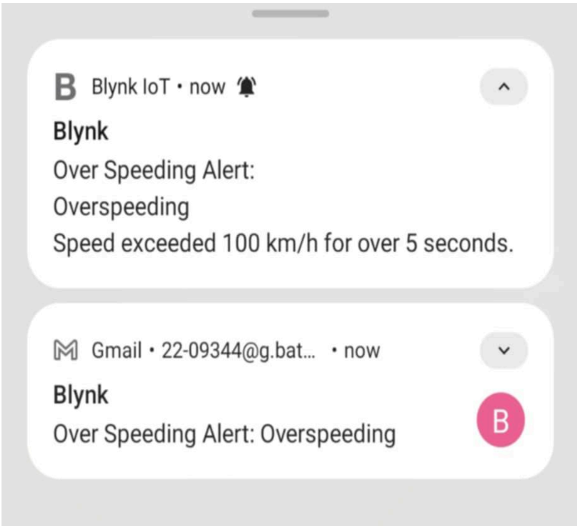


Fig. 1.4 Overspeed Status

Test result:

Speed Detected	Green LED	Red LED	Buzzer	LCD print	Location show	IoT Alert
<100 km/h	ON	OFF	OFF	NORMAL SPEED	YES	NO
≥ 100 km/h	OFF	ON	ON	OVERSPEED	YES	YES





The results of the functional test confirm the effectiveness of the developed overspeeding detection system in monitoring vehicle speed and issuing appropriate alerts based on predefined thresholds. The system was configured to identify speeds equal to or greater than 100 km/h as overspeeding events.

From the test results:

- When the vehicle's speed was less than 100 km/h, the system accurately indicated normal speed conditions by turning on the green LED, keeping the red LED and buzzer off, displaying "NORMAL SPEED" on the LCD, and showing real-time GPS location. No IoT or remote alert was triggered.

- When the speed was 100 km/h or above, the system successfully activated the overspeeding alert. The red LED and buzzer were triggered to warn the driver, the LCD displayed "OVERSPEED", GPS location remained active, and an IoT-based alert (e.g., SMS or remote notification) was initiated if integrated.

These observations demonstrate that the prototype consistently and accurately distinguishes between normal and overspeeding scenarios and responds accordingly with both local and remote alerts. The integration of GPS data ensures accurate speed tracking and location monitoring, which is crucial for real-time and remote safety applications.

In summary, the system fulfills its design goals of providing real-time overspeed detection, driver alerting, and location-based reporting. It has strong potential for real-world deployment in vehicles to enhance road safety and enable effective speed monitoring, especially in areas where speed regulation is critical.

Recommendation:

To ensure the overspeeding detection system functions effectively in a real-world vehicular environment, the following recommendations are proposed:

- Ensure Reliable GPS Signal Acquisition
  In actual driving scenarios, GPS signal quality may vary due to environmental factors such as tunnels, tall buildings, or dense tree cover. To mitigate this, it is recommended to use a GPS module with an external antenna and install it in a location with maximum sky visibility, such as near the vehicle's windshield or on the roof.

- Integrate the System with Vehicle Ignition
  For practical use, the system should automatically power on and off with the vehicle ignition. This avoids draining the car battery when the vehicle is not in use and ensures the system is always active when the vehicle is running.

- Implement Real-Time Alerts for the Driver
  In addition to the LED and buzzer indicators, the system can be enhanced with a voice module or a graphical display to provide audible or visual warnings that are more noticeable to the driver, improving safety response time.

- Log Speed and Location for Recordkeeping
  Storing GPS and speed data on an SD card or transmitting it to a remote server provides a historical record of vehicle movements. This can be valuable for fleet monitoring, insurance assessments, or law enforcement verification.

- Conduct Environmental Durability Testing
  Before deployment, the system should be tested under various conditions (heat, vibration, humidity, etc.) to ensure it can withstand typical automotive environments without malfunctioning.

- Secure Mounting and Cabling
  Ensure the system is securely mounted inside the vehicle to prevent disconnection or damage due to vibration or sudden movements. All wiring should be insulated and routed safely to prevent electrical shorts or user injury.

- Regulatory Compliance and Safety Standards
  Ensure that the device complies with local automotive and safety regulations, especially if integrated into commercial or public service vehicles. Components should meet relevant automotive standards (e.g., ISO 7637 for electrical transients).

Conclusion

The overspeeding detection prototype developed using the ESP32, GY-NEO6MV2 GPS module, I2C LCD, LED indicators, buzzer, and the Blynk IoT platform successfully meets the objectives of real-time speed monitoring, local alerting, and remote notification.

Throughout testing, the system accurately detected vehicle speed based on GPS data and compared it against a defined threshold of 100 km/h. When the speed remained below the threshold, the system indicated a safe driving condition through the green LED, a “NORMAL SPEED” message on the LCD, and suppressed alert notifications. In contrast, when the speed exceeded or equaled the threshold, the system responded with a comprehensive alert mechanism: activating the red LED, sounding the buzzer, displaying an “OVERSPEED” warning on the LCD, and sending a real-time notification to the Blynk mobile application.

The integration of IoT via the Blynk platform adds a critical layer of functionality by enabling remote awareness and monitoring. This feature is particularly useful for fleet management, parental control, and enforcement applications, where real-time speed violations can be tracked even without direct access to the vehicle.

The project demonstrates a reliable, low-cost solution that leverages GPS and IoT technologies to promote safer driving behavior. It shows strong potential for real-world deployment in personal, commercial, or public transportation vehicles, especially in environments where overspeeding is a major safety concern.

In conclusion, this prototype not only provides effective local alerts to drivers but also expands the system’s reach through cloud-based notifications, enhancing the overall impact of speed monitoring on road safety.