Ahmedabad University

Project Report-3
Group-5
Deadline: 10/10/2025

Instructor: Anurag Lakhlani **Teaching Assistant:** Priyesh Chaturvedi

Enrolment No	Name
AU2240118	Tirth Teraiya
AU2240080	Namra Maniar
AU2240244	Saumya Shah
AU224075	Henish Trada

Chapter 05

Arduino UNO/Mega OR Raspberry Pi Features

This chapter presents the detailed electronic circuit design for the IoT-based Environmental Air Quality Monitoring and Alerting System. It outlines the overall system schematic, provides a comprehensive justification for the selection of each hardware component, and analyzes the technical inputs and outputs of the integrated system.

5.1 Raspberry Pi 3 Model B: Core Features and Project Role

The Raspberry Pi 3 Model B is a credit card-sized single-board computer (SBC) that offers a powerful combination of processing capability, memory, and integrated connectivity, making it a popular choice for complex IoT projects.

5.1.1 Key Features Utilized in Project

Processor: The board is equipped with a Quad-Core 1.2GHz Broadcom BCM2837 64-bit ARMv8 processor.

Memory: It includes 1GB of LPDDR2 SDRAM.

Connectivity: The on-board BCM43438 wireless LAN (Wi-Fi 802.11n) and Bluetooth Low Energy (BLE) are cornerstone features for this IoT project.

GPIO (General Purpose Input/Output): The 40-pin extended GPIO header provides the physical interface to all sensors and actuators. This project utilizes the header's support for multiple protocols:

- **I2C pins (SDA, SCL):** For the 16x2 LCD module.
- **UART pins (TXD, RXD):** For the PMS5003 sensor.
- **Standard Digital I/O pins:** For Active Buzzer.

USB Ports: The four USB 2.0 ports are used for interfacing with the Arduino UNO, which acts as an ADC and provides data over a serial connection.

5.1.2 Strengths and Weaknesses for this Project

Strengths:

- **Full-fledged Operating System:** Running Raspberry Pi OS (a Debian derivative) greatly simplifies development. It allows for easy installation of programming languages (Python), libraries (e.g., RPi.GPIO, smbus), remote access via SSH for debugging, and robust management of network connections.
- **Vast Community and Library Support:** The Raspberry Pi has one of the largest and most active communities in the maker space. This translates to extensive, well-maintained Python libraries for virtually every common sensor and peripheral, including every component used in this project.
- **High Computing Power:** The quad-core processor enables not only data acquisition but also on-device data processing. This allows for the implementation of more complex AQI calculation algorithms.

Weaknesses and Mitigation:

- **Lack of Analog Inputs:** The most significant weakness of the Raspberry Pi for electronics projects is its inability to read analog signals directly. This limitation is directly addressed by using an Arduino UNO as an external ADC. The Arduino reads the analog signal from the MQ135, and the Raspberry Pi reads the digitized value from the Arduino over a standard USB serial connection. This is a common and well-documented pattern for interfacing analog sensors with a Raspberry Pi.
- **Higher Power Consumption:** Compared to a microcontroller like the Arduino, the Raspberry Pi is significantly more power-hungry, consuming around 700 mA or more under load. This makes it less suitable for battery-powered, portable applications. For this project, a stationary monitoring node powered by a stable 5V/2.5A DC adapter is assumed, rendering this weakness irrelevant to the primary use case.

Chapter 06

Program Flow Chart

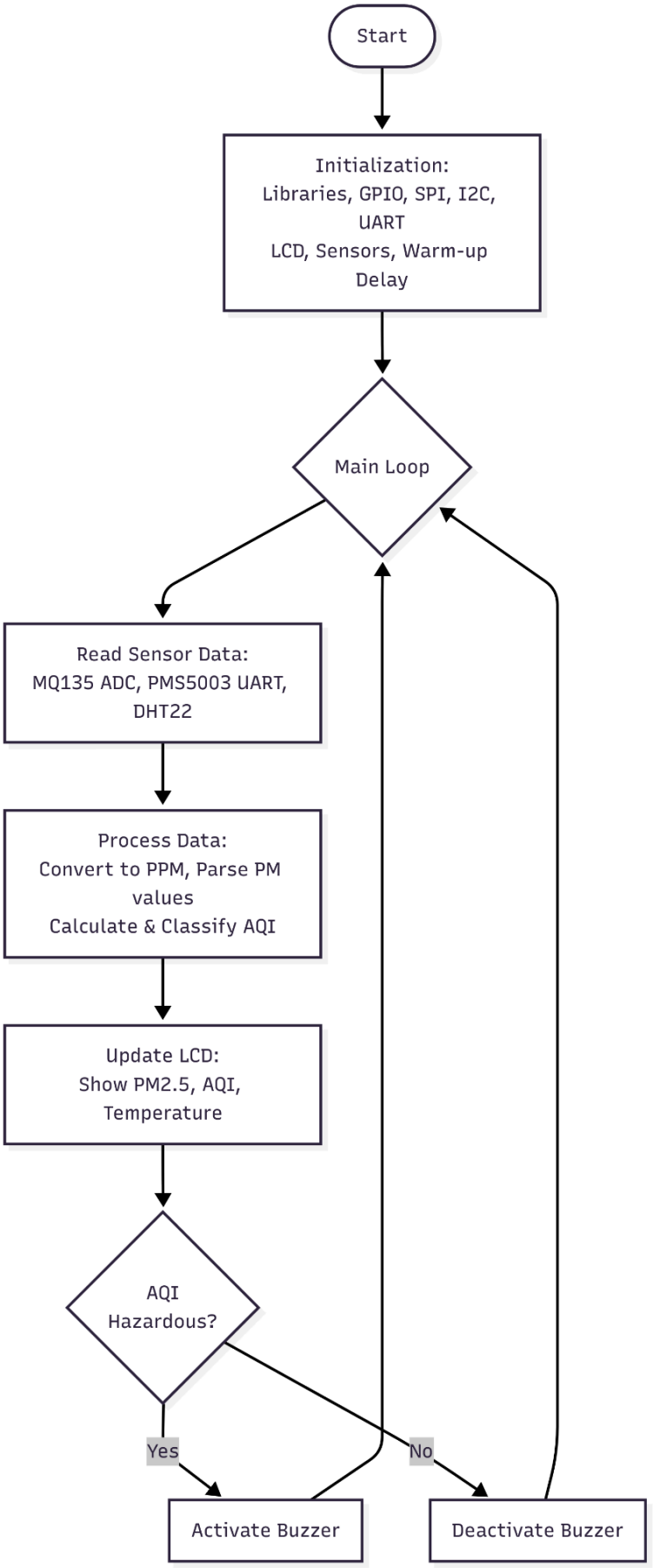
This chapter outlines the logical flow of the software designed to run on the Raspberry Pi. The program is responsible for initializing hardware, continuously reading data from all sensors, processing this data to determine air quality, displaying the results locally, and triggering alerts when necessary.

6.1 High-Level Program Logic Flowchart

The program operates on a continuous loop, orchestrating the various hardware components. A significant aspect of the software architecture is its ability to manage multiple, disparate communication interfaces concurrently—a task well-suited to the Raspberry Pi's multi-threaded Linux environment. The logical sequence of operations is as follows:

1. **Start:** The program execution begins.
2. **Initialization:** This is a critical one-time setup phase.
    - **Import Libraries:** Load necessary Python libraries.
    - **Initialize Interfaces:** Configure the GPIO pins for their respective modes (input/output). This includes setting up the I2C bus for the LCD and the serial (UART) port for the PMS5003. A separate serial object is created for the USB connection to the Arduino.
    - **Initialize Peripherals:** Create objects for each hardware component (e.g., the LCD object, the ADC object).
    - **Display Startup Message:** Write an initial message like "System Initializing..." to the LCD to provide user feedback.
    - **Sensor Warm-up:** Implement a mandatory delay to allow sensors, particularly the MQ135 with its internal heater, to reach a stable operating temperature. This is crucial for accurate initial readings.
3. **Enter Main Loop:** The program enters an infinite `while` loop to perform continuous monitoring.
4. **Read All Sensor Data:** In each iteration of the loop, the program polls each sensor to gather fresh data.
    - Read the serial data line from the Arduino UNO, which contains the 10-bit digitized value from the MQ135 sensor.
    - Read the 32-byte data frame from the PMS5003 via the UART serial buffer and validate the checksum.

5. **Process and Analyze Data:** The raw data from the sensors is converted into meaningful environmental metrics.
    - Convert the raw ADC value received from the Arduino into a voltage and then, using the sensor's characteristic curve from its datasheet, estimate the gas concentration in Parts Per Million (PPM).
    - Parse the PMS5003 data frame to extract the PM2.5 and PM10 concentration values (in μg/m³).
    - Calculate a composite Air Quality Index (AQI) using a weighted algorithm based on the measured PM2.5 and gas concentration values.
    - Classify the calculated AQI into a human-readable category (e.g., "Good," "Moderate," "Hazardous") based on standard thresholds.
6. **Update Local Display:** The processed information is displayed on the 16x2 I2C LCD.
    - The LCD is cleared.
    - Formatted strings showing the most critical data (e.g., PM2.5 value, AQI category, temperature) are written to the two lines of the display.
7. **Check Alert Condition:** A decision point checks if the air quality has reached a critical level.
    - The program evaluates: `if AQI_category == "Hazardous"`.
8. **Manage Alert System:**
    - **If True (Hazardous):** The program sends a signal to the GPIO pin connected to the active buzzer, causing it to emit an audible alarm.
    - **If False (Not Hazardous):** The program ensures the buzzer's GPIO pin is in a state that keeps the buzzer silent.
9. **Loop:** The execution flow returns to the "Read All Sensor Data" step to begin the next monitoring cycle.


Start
Initialization:
Libraries, GPIO, SPI, I2C,
UART
LCD, Sensors, Warm-up
Delay
Main Loop
Read Sensor Data:
MQ135 ADC, PMS5003 UART,
DHT22
Process Data:
Convert to PPM, Parse PM
values
Calculate & Classify AQI
Update LCD:
Show PM2.5, AQI,
Temperature
AQI
Hazardous?
Yes
No
Activate Buzzer
Deactivate Buzzer

Arduino Code:

```
// Arduino Uno sketch
#include <SoftwareSerial.h>
#include <PMS.h>            // include PMS library (install via Library Manager)
#include <ArduinoJson.h>    // for JSON formatting (optional but convenient)

#define MQ_PIN A0
#define BUZZER_PIN 8

SoftwareSerial pmsSerial(2, 3); // RX, TX  -> PMS5003 TX->2
PMS pms(pmsSerial);
PMS::DATA pmsData;

unsigned long lastSend = 0;
const unsigned long sendInterval = 5000; // ms

void setup() {
  Serial.begin(115200);       // USB Serial to Raspberry Pi
  pmsSerial.begin(9600);      // PMS5003 default baud
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);
  delay(1000);
  // wake PMS if needed:
  pms.wake();
}

float readMQ135() {
  int raw = analogRead(MQ_PIN); // 0-1023
  // Convert raw to approximate voltage (0-5V)
  float voltage = raw * (5.0 / 1023.0);
  // Simple mapping to "AQ index" or ppm — placeholder: calibrate properly
  // Many MQ135 modules require Ro/Rs calibration. We'll send raw ADC and voltage.
  return voltage;
}

void loop() {
  unsigned long now = millis();
  if (now - lastSend >= sendInterval) {
    lastSend = now;

    // Read PMS5003 (trigger a read)
    if (pms.read(&pmsData)) {
```

```
    // successful read
  } else {
    // failed read or no update yet
  }

  float mqVoltage = readMQ135();

  // Create a JSON object string manually (to keep memory small)
  // Example JSON: {"pm1":12,"pm2_5":20,"pm10":30,"mq_v":2.34}
  int pm1 = pmsData.PM_AE_UG_1_0;   // PM1.0 ug/m3 (auto-equivalent)
  int pm25 = pmsData.PM_AE_UG_2_5;  // PM2.5 ug/m3
  int pm10 = pmsData.PM_AE_UG_10;   // PM10 ug/m3

  // Optional: local alert logic
  if (pm25 > 100 || mqVoltage > 3.5) {
    digitalWrite(BUZZER_PIN, HIGH); // sound buzzer when very bad
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }

  // Send JSON over Serial (one line)
  Serial.print("{");
  Serial.print("\"pm1\":"); Serial.print(pm1); Serial.print(",");
  Serial.print("\"pm2_5\":"); Serial.print(pm25); Serial.print(",");
  Serial.print("\"pm10\":"); Serial.print(pm10); Serial.print(",");
  Serial.print("\"mq_v\":"); Serial.print(mqVoltage, 3);
  Serial.println("}");
 }

 // Let PMS run; short delay
 delay(100);
}
```

Explanation of the Code:

7.1.6 Flow of the Code

The Arduino sketch shown above forms the core logic for collecting, processing, and transmitting air quality data to the Raspberry Pi. It integrates readings from both the MQ135 gas sensor and the PMS5003 particulate matter sensor while providing real-time alerts through a buzzer. The following section describes the complete flow and functionality of the code.

1. Library Inclusion and Pin Configuration

The program begins by including three key libraries:

- SoftwareSerial.h – Allows creation of a secondary serial communication channel, essential for interfacing with the PMS5003 particulate sensor since the hardware serial port is used for communication with the Raspberry Pi.

- PMS.h – A dedicated library for the PMS5003 sensor that simplifies reading particulate matter values such as PM1.0, PM2.5, and PM10.

- ArduinoJson.h – Facilitates JSON-formatted data generation, making it easier for the Raspberry Pi to parse and process the readings.

Two pins are then defined:

- **MQ_PIN (A0): Analog input pin for the MQ135 sensor's output.**

- **BUZZER_PIN (8): Digital output pin used to drive the buzzer for alert notifications.**

Additionally, the PMS5003 is connected to the Arduino using pins 2 (RX) and 3 (TX) through a `SoftwareSerial` interface.

2. Initialization in `setup()`

During setup, three primary tasks are executed:

1. Serial Communication Setup

- `Serial.begin(115200)` initializes the main USB serial port for communication with the Raspberry Pi at 115200 baud rate.

- `pmsSerial.begin(9600)` initializes the PMS5003 communication channel at its default baud rate of 9600.

2. **Hardware Initialization**

    - **T**he buzzer pin is configured as an output and initially turned off using `digitalWrite(BUZZER_PIN, LOW)`.

    - The PMS sensor is "woken up" using the `pms.wake()` function to ensure it starts sending readings after power-up.

3. **Stabilization Delay**

    - **A one-second delay is added to allow hardware to stabilize before the main loop starts execution.**

3. Reading the MQ135 Sensor (`readMQ135()` Function)

This function handles the acquisition of analog data from the MQ135 gas sensor:

- The analog signal is read using `analogRead(MQ_PIN)`, producing a raw digital value between 0 and 1023.

- This value is converted to an approximate voltage using the formula:
  $$
  \text{Voltage} = \text{Raw Value} \times \frac{5.0}{1023.0}
  $$

- This voltage acts as a relative indicator of air pollution levels (the higher the voltage, the poorer the air quality).

- In a calibrated setup, this voltage could be converted into precise parts-per-million (ppm) readings for gases like CO_2 or NH_3.

4. Main Program Logic (`loop()` Function)

The `loop()` function runs continuously, performing sensor reads and communication tasks at defined time intervals.

1. **Timed Execution Control**
    - The program ensures that sensor readings and data transmission occur every 5 seconds, controlled by `sendInterval = 5000` milliseconds.
    - This avoids excessive data transmission while maintaining real-time monitoring.
2. **Reading PMS5003 Data**
    - The PMS5003 sensor is polled using `pms.read(&pmsData)`.
    - Upon successful read, particulate matter concentrations are extracted:
        - `pmsData.PM_AE_UG_1_0` → PM1.0 concentration (µg/m³)
        - `pmsData.PM_AE_UG_2_5` → PM2.5 concentration (µg/m³)
        - `pmsData.PM_AE_UG_10` → PM10 concentration (µg/m³)
3. **Local Alert Logic**
    - After reading both sensors, the code checks if pollution exceeds safe thresholds:
        - If PM2.5 > 100 µg/m³ or MQ135 voltage > 3.5V, the buzzer is activated.
        - Otherwise, the buzzer remains off.
    - This feature provides immediate feedback about deteriorating air quality.
4. **Data Packaging and Transmission**
    - The collected readings are structured into a lightweight JSON object for easy parsing by the Raspberry Pi.
      Example output:

JSON

```
{"pm1":12,"pm2_5":20,"pm10":30,"mq_v":2.34}
```

- This JSON string is sent to the Raspberry Pi via USB serial using `Serial.print()` statements.

5. **Delay and Loop Continuation**
    - A short `delay(100)` at the end of each loop iteration ensures stable sensor communication and prevents rapid redundant polling.

5. Overall Workflow Summary

1. **Arduino initializes sensors and serial communication.**
2. **Every 5 seconds:**
    - Reads gas concentration from the MQ135 (via analog input).
    - Reads PM values (PM1.0, PM2.5, PM10) from PMS5003.
    - Activates the buzzer if pollutant levels exceed thresholds.
    - Transmits all readings to the Raspberry Pi in JSON format.
3. **The Raspberry Pi receives, stores, and visualizes these readings for continuous air quality monitoring.**

6. Key Functional Highlights

- Multi-sensor integration: Combines gas detection (MQ135) and particulate sensing (PMS5003).
- Data standardization: Uses JSON formatting for structured, machine-readable output.

- Local alert mechanism: Provides immediate environmental feedback through a buzzer.
- Efficient communication: Maintains steady data flow between Arduino and Raspberry Pi.

Chapter 07

Sensors

This chapter provides a detailed technical overview of the sensors employed in the air quality monitoring system. For each sensor, the operating principle, physical specifications, power ratings, and method of interfacing with the Raspberry Pi are described.

7.1 MQ135 Air Quality Gas Sensor

The MQ135 is a semiconductor-type gas sensor designed to detect a wide range of gases, making it suitable for general air quality monitoring. It is particularly sensitive to ammonia (NH3), nitrogen oxides (NOx), alcohol, benzene, smoke, and carbon dioxide (CO2).

7.1.1 Operating Principle

The sensing mechanism of the MQ135 is based on a chemoresistive material, specifically Tin Dioxide (SnO2). The core principle is as follows:

1. **Sensing Material:** The sensor contains a micro-tubular ceramic element coated with a layer of SnO2. In clean air, oxygen molecules are adsorbed onto the surface of the SnO2, trapping electrons from the material and creating a potential barrier that results in high electrical resistance.
2. **Gas Detection:** When pollutant gases are introduced, they react with the adsorbed oxygen molecules on the sensor's surface. This reaction releases the trapped electrons back into the SnO2, lowering the potential barrier and thereby decreasing the material's overall resistance.
3. **Signal Output:** This change in resistance is proportional to the concentration of the target gas. An external load resistor (RL) is used in a voltage divider circuit. As the sensor's resistance (RS) changes, the voltage across the load resistor (VRL) also changes, providing an analog voltage output that corresponds to the gas concentration.
4. **Heating Element:** The sensor includes an internal heating element made of Ni-Cr alloy, which is necessary to bring the SnO2 material to its optimal operating temperature (typically 100-200°C). This is why the sensor requires a pre-heating period of at least 24 hours for optimal performance and stabilization before accurate measurements can be taken.

7.1.2 Physical Dimensions and Pin Diagram

The MQ135 is commonly available as a module mounted on a small PCB for ease of use.

- **Physical Dimensions:** The module typically measures approximately 35 mm x 22 mm x 23 mm (L x W x H).
- **Pinout:** The module has a 4-pin header for connection:
    - **VCC:** Power supply input (5V).
    - **GND:** Ground connection.
    - **D0 (Digital Output):** Provides a HIGH or LOW signal based on a threshold set by an on-board potentiometer. This pin is not used in this project.
    - **A0 (Analog Output):** Provides the analog voltage signal proportional to gas concentration. This is the primary output used for this project.

7.1.3 Power Ratings and Technical Specifications

1. **Heater Voltage (VH):** 5.0V ± 0.1V AC or DC.
2. **Circuit Voltage (VC):** The sensor module operates at 5V DC.
3. **Heater Consumption (PH):** Approximately ≤950mW.
4. **Load Resistance (RL):** Adjustable via on-board potentiometer, but typically around 10 kΩ.
5. **Detection Range:** 10 – 1000 ppm for gases like ammonia and benzene.

7.1.4 Interfacing with Raspberry Pi

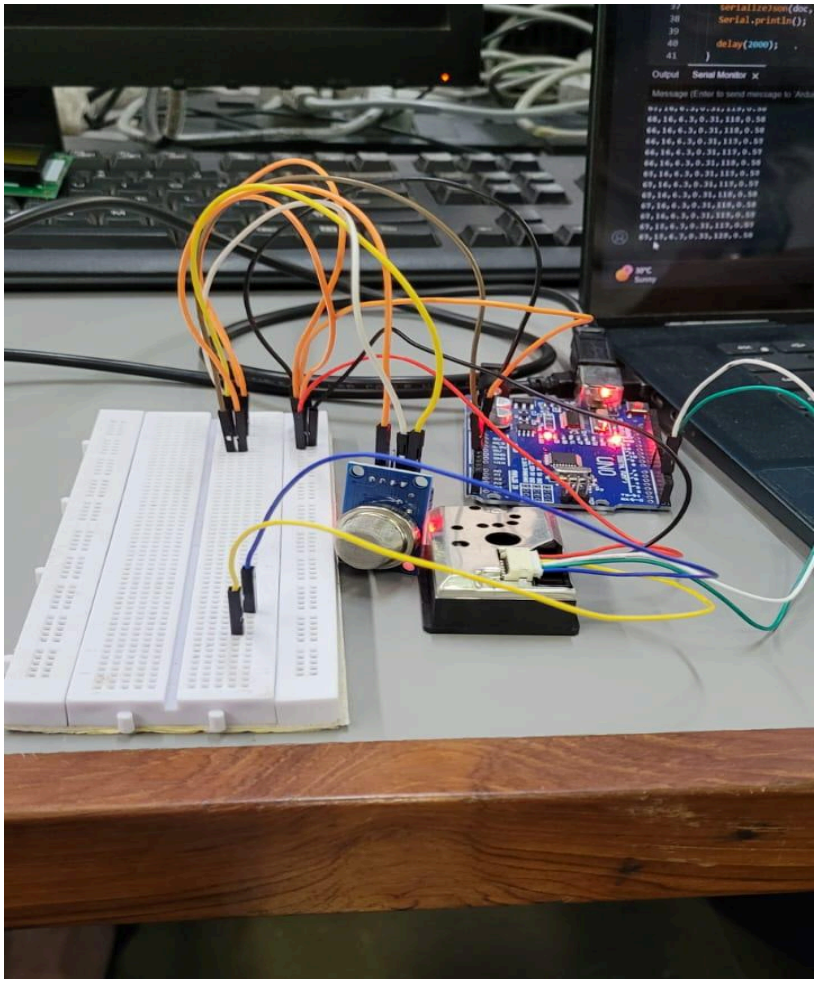
As the Raspberry Pi cannot process analog signals, the A0 pin of the MQ135 cannot be connected directly. An Arduino UNO serves as the necessary intermediary, acting as an Analog-to-Digital Converter (ADC).

- **Circuit:** The MQ135's VCC and GND pins are connected to the Arduino's 5V and GND pins, respectively. The sensor's A0 pin is connected to one of the Arduino's analog input pins (e.g., A0). The Arduino is then connected to the Raspberry Pi via a USB cable. The Arduino reads the analog voltage, converts it into a 10-bit digital value (0-1023), and sends this value over the serial connection to the Raspberry Pi, which can then read and process it.
- Preliminary connection of our sensors and raspberry pi is shown in the below image followed by explanation of the circuit.

7.1.5 Circuit Explanation

The above figure illustrates the preliminary circuit setup used for interfacing the MQ135 gas sensor with the Raspberry Pi through an Arduino UNO. The MQ135 sensor module is powered using a 5V supply from the Arduino, with the ground terminals of all components connected in common to ensure a stable reference. The sensor’s analog output (A0) is connected to the Arduino’s analog input pin (A0), enabling the Arduino to read the variable voltage corresponding to the gas concentration.

The Arduino functions as an Analog-to-Digital Converter (ADC), converting the analog voltage values from the MQ135 into 10-bit digital data (ranging from 0 to 1023). These readings are then transmitted serially to the Raspberry Pi through the USB connection. The Raspberry Pi receives and processes this data for further analysis, visualization, and storage.This arrangement allows continuous monitoring of air quality in real time while maintaining system flexibility. The setup also facilitates the integration of multiple sensors, as the same communication mechanism (Arduino-to-Raspberry Pi serial interface) can be extended to collect various environmental parameters simultaneously.



Chapter 08

Actuators and Displays

This chapter details the output components of the system: the devices that provide feedback to the user. It covers the operating principles, specifications, and interfacing methods for the local display and the audible alert mechanism.

8.1 16x2 I2C LCD Module

The 16x2 I2C LCD module is the primary visual interface for the system, providing real-time, human-readable information about the measured air quality parameters. It displays 16 characters per line across two lines.

8.1.1 Operating Principle

The module is a composite device consisting of two main parts: a standard character LCD and an I2C interface board.

- **Character LCD:** The display itself is typically based on the Hitachi HD44780 controller or a compatible equivalent. This controller requires a parallel interface (either 4-bit or 8-bit) to receive commands (like "clear display" or "move cursor") and data (the ASCII codes for characters to be displayed). Driving this directly would consume at least 6 GPIO pins from the Raspberry Pi.
- **I2C Interface Board:** To simplify wiring, an interface board is "piggy-backed" onto the LCD's 16-pin header. This board is built around a PCF8574 I/O expander chip. The PCF8574 acts as an I2C slave device. The Raspberry Pi, acting as the I2C master, sends commands and data over the two-wire I2C bus (SDA and SCL). The PCF8574 receives these serial commands and translates them into the parallel signals required by the HD44780 controller, managing the LCD's data and control lines. This elegant solution reduces the required GPIO pin count from six or more down to just two, which is a significant advantage in a pin-constrained project.

8.1.2 Physical Dimensions and Pin Diagram

- **Physical Dimensions:** The module's PCB is approximately 80 mm x 36 mm.
- **Pinout:** The I2C interface provides a simple 4-pin header for connection:
    - **GND:** Ground.
    - **VCC:** Power supply (5V).
    - **SDA (Serial Data):** The I2C data line.
    - **SCL (Serial Clock):** The I2C clock line.

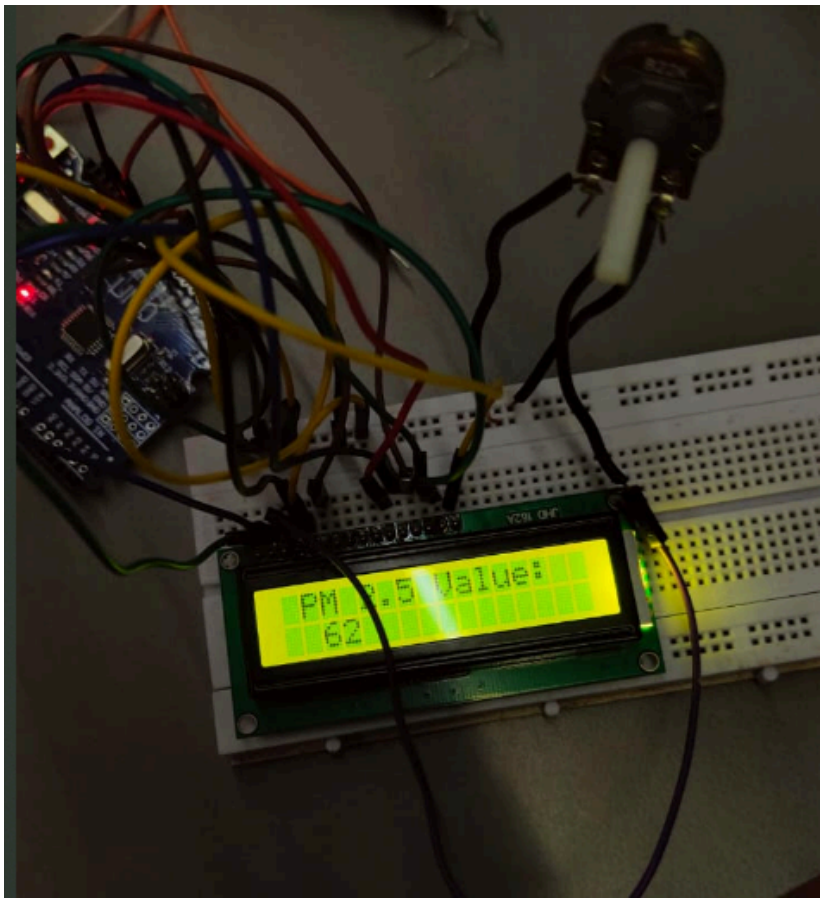
8.1.3 Power Ratings and Technical Specifications

- **Operating Voltage:** 5V DC.
- **Interface:** I2C (Inter-Integrated Circuit).
- **I2C Address:** The slave address is typically either 0x27 or 0x3F, depending on the specific I/O expander chip used on the interface board. The default address must be confirmed, often by running the `i2cdetect -y 1` command on the Raspberry Pi's terminal.
- **Display Format:** 16 characters x 2 lines.
- **Backlight:** Blue or green, controllable via software commands sent over I2C.
- **Contrast:** Adjustable via an on-board potentiometer.

8.1.4 Interfacing with Raspberry Pi

Interfacing the module with the Raspberry Pi is straightforward due to the I2C protocol.

- **Circuit:** The four pins of the LCD module (GND, VCC, SDA, SCL) are connected directly to the corresponding pins on the Raspberry Pi's GPIO header (GND, 5V, GPIO2/SDA, GPIO3/SCL). No other components are required for the connection. The Raspberry Pi's I2C interface must be enabled in the `raspi-config` utility.



Video Link to the circuit :

Circuit_Video.mp4