SMART SPEAKER

EMBEDDED SYSTEMS PROJECT

Students:

Raul-Ionuț Buș
Alex-Virgil Crișan

Timișoara
May, 2024

Contents

1 Introduction **2**
- 1.1 Objectives . . . 2
- 1.2 Scope of the project . . . 2

2 Background Study **3**
- 2.1 Overview of Embedded Systems . . . 3
- 2.2 Development Boards . . . 3
  - 2.2.1 Arduino Uno . . . 3
  - 2.2.2 Raspberry Pi 2 . . . 3
- 2.3 Sensors . . . 3
  - 2.3.1 MQ135 Gas Sensor . . . 4
  - 2.3.2 SGP30 Gas Sensor . . . 4
  - 2.3.3 DHT11 Temperature and Humidity Sensor . . . 4
- 2.4 Actuators . . . 4
  - 2.4.1 DC Fan . . . 4
  - 2.4.2 Stepper Motor . . . 4

3 System Architecture **5**
- 3.1 Overview . . . 5
- 3.2 Hardware Components . . . 5
  - 3.2.1 Arduino Uno . . . 5
  - 3.2.2 Raspberry PI 2 . . . 6
  - 3.2.3 MQ135 Gas Sensor . . . 8
  - 3.2.4 SGP30 Gas Sensor . . . 9
  - 3.2.5 DHT11 Temperature and Humidity Sensor . . . 10
  - 3.2.6 DC Fan . . . 11
  - 3.2.7 Stepper Motor . . . 11
- 3.3 Software Components . . . 12

4 Design and Implementation **13**
- 4.1 Overview Design . . . 13
- 4.2 Sensor Integration with Arduino Uno . . . 14
- 4.3 Actuator Control . . . 18
- 4.4 Serial Communication Setup with Raspberry Pi 2 . . . 21
- 4.5 Coding and Programming Details . . . 23

5 Conclusions **31**
- 5.1 Summary of Achievements . . . 31
- 5.2 Improvements . . . 31

Bibliography **32**

Chapter 1

Introduction

1.1 Objectives

The primary objective of this project is to design and implement an embedded system which contain one microcontroller and multiple sensors that can monitor environmental parameters and based on this parameters to control actuators. In addition, this system will further communicate with a microprocesor for more complex operations.

1.2 Scope of the project

This project involves integrating multiple air quality sensors (MQ-135, SGP-30, DHT-11) and actuators (PWM-Fan, Stepper-Motor) with an Arduino One microcontroller. Establishing a serial communication between Arduino and a Raspberry Pi is another critical aspect of our project. This system addresses the need for a simple and efficient solution to monitor temperature, humidity and air quality and based on this parameters to control ventilation and movement within an environment.

Chapter 2

Background Study

2.1 Overview of Embedded Systems

Embedded systems are ubiquitous in modern technology, representing a combination that integrates hardware and software designed for a specific functionality. In some applications, embedded systems can be part of a more complex system.

2.2 Development Boards

Development Boards provide a platform for designing, deploying and testing hardware and software components. Various types of boards exists, each designed for specific requirements.

2.2.1 Arduino Uno

The Arduino Uno is a well-known and widely used microcontroller board based on the ATmega328P microcontroller. The reasons for choosing this board include the abundance of available resources, its simplicity to use, ensuring long-term reliability, and its characteristics that align with the project objectives.

2.2.2 Raspberry Pi 2

The Raspberry Pi 2 is a microprocessor based on an ARM Cortex-A7 CPU. Its energy-efficient architecture allows for maintaining high computational power, making it suited for complex systems. In the large system stands as the central component, offering us more sophisticated ways to interact with data, better communication lines, and enhanced processing. Raspberry tasks in our embedded system are handling data from Arduino, managing control signals for the actuators, and ensuring integration of the entire system.

2.3 Sensors

Sensors are devices designed to receive stimulus from the environment and transform that stimulus into an electric signal. In our system we are using multiple type of gas sensors and a sensor for temperature and humidity.

2.3.1 MQ135 Gas Sensor

MQ135 is a low-cost and easy to use gas sensor that can detect the presence of multiple hazardous gasses as ammonia, benzene, sulfide and toluene.

2.3.2 SGP30 Gas Sensor

SGP30 is a gas sensor that can detect a wide range of Volatile Organic Compounds (VOCs) and Hydrogen (H2) and is intended for indoor air quality monitoring. Also, this sensor calculates Equivalent Carbon Dioxide(eCO2) based on hydrogen concentration; it's not a "true CO2" sensor for laboratory use, but it offers valuable information.

2.3.3 DHT11 Temperature and Humidity Sensor

The DHT11 Temperature and Humidity Sensor offers efficiency and long-term reliability in a cost-effective design. The values returned by this sensor will be used to calibrate and correct the data obtained from the other sensors, ensuring more realistic readings across the system.

2.4 Actuators

An actuator is a device that creates movement by converting the power source, which can be either electric, hydraulic, or pneumatic, into a linear or rotary type of movement.

2.4.1 DC Fan

A fan is an actuator that transforms an electric signal into rotary movement. The purpose of a fan is to ventilate and cool the environment.

2.4.2 Stepper Motor

The stepper motor is an electromechanical actuator that converts electric signal pulses into precise mechanical movements. This motor is offering a precise control over movement because it is moving for a discrete number of steps, allowing for accurate positioning without an additional feedback sensor.

Chapter 3

System Architecture

3.1 Overview

In the architecture of the embedded system, we combine various hardware components such as sensors, actuators, and development boards, as presented previously. These components are working together to form the foundation of the system, allowing it to interact with the environment, handle the data obtained, and perform the tasks for which it was designed.

3.2 Hardware Components

3.2.1 Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P . It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button.

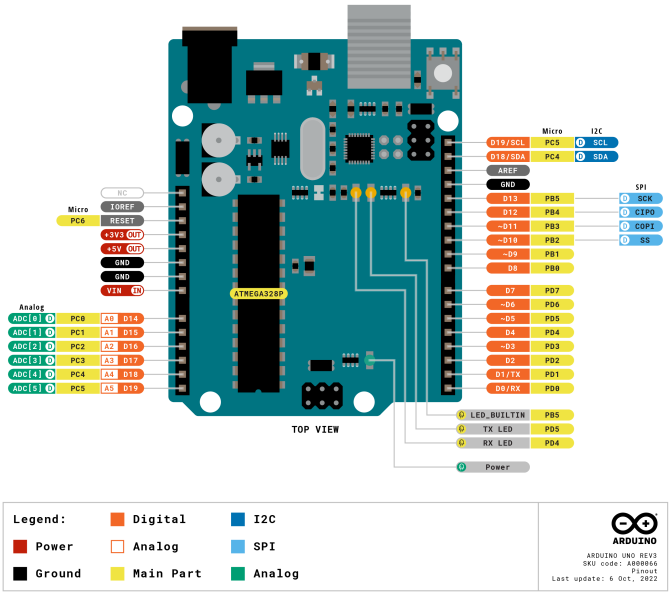


Figure 3.1: Arduino Uno

- **ATmega328P Microcontroller**: The central processing unit responsible for executing the instructions.

- **Digital I/O Pins**: 14 Pins that can be used as input or output. The Pins 3, 5, 6, 9, 10, 11 can also be used as PWM pins.
- **Analog Input Pins**: 6 Pins that can read analog voltages from sensors and convert them to a digital value.
- **16 MHz Ceramic Resonator**:Provides the clock signal for the microcontroller.
- **USB Connection**: Allows for programming the board and serial communication with a computer.
- **Power Jack**: Can be used to power the board with an external power supply.
- **ICSP HEADER**: Used for programming the microcontroller with an external programmer.
- **Reset Button**: Resets the microcontroller, restarting the currently loaded program.

The role of this board in the system is crucial because it is responsible for sensor integration, data acquisition, and data transmission to the Raspberry Pi through serial interface.

3.2.2 Raspberry PI 2

Raspberry Pi 2 is credit-card sized computer with an ARM processor that can run Linux. The Raspberry Pi 2 Model B is the second generation of Raspberry Pi.

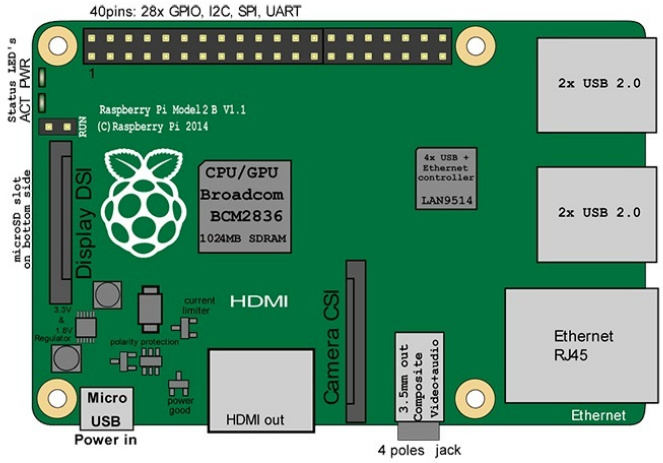


Figure 3.2: Raspberry Pi

- **900 MHz quad-core ARM Cortex-A7 CPU**
- **1 GB RAM**
- **VideoCore IV 3D graphics core**
- **Ethernet port**
- **Four USB ports**
- **Full-size HDMI output**
- **Four-pole 3.5 mm jack**: supports audio and composite video output
- **40-pin GPIO header**: 0.1'-spaced male pins
- **CSI**: camera interface
- **DSI**: display interface
- **Micro SD card slot**
- **Operating voltage**: 5V

There are 26 GPIO pins, two 5V PINS, two 3.3V pins, eighth ground pins(GND), and two ID EEPROM pins among the total 40 Raspberry Pi pins. A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

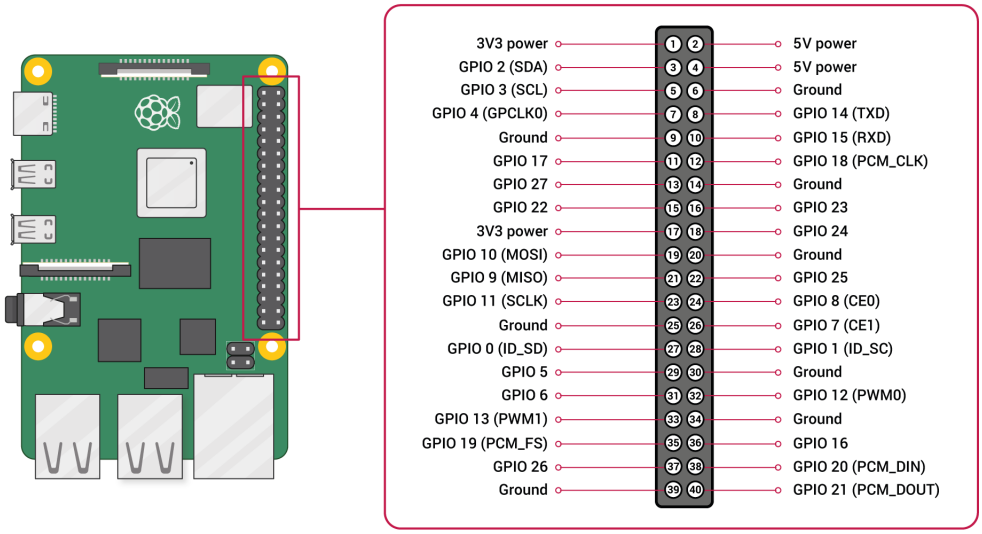


Figure 3.3: GPIO Pinout

A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

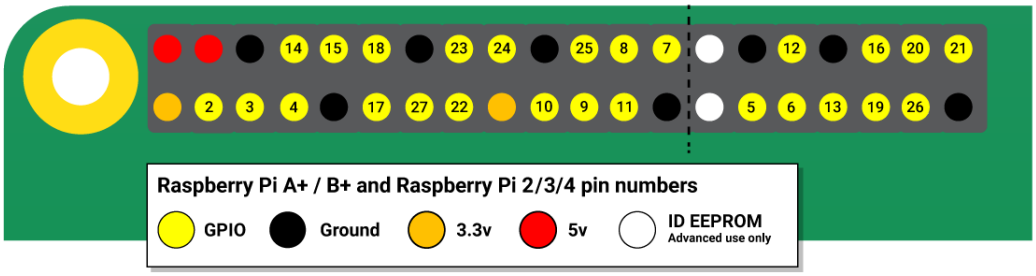


Figure 3.4: GPIO Diagram

GPIO functions:

- **PMW**: Pulse Width Modulation
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- **SPI**: Serial Peripheral Interface
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- **I2C**: two-wire serial communication protocol using a serial data line (SDA) and a serial clock line (SCL)
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- **Serial**
  - TX (GPIO14); RX (GPIO15)

3.2.3 MQ135 Gas Sensor

The MQ135 Gas sensor is one type gas sensor used to detect and monitor a wide range of gases present in air like ammonia, alcohol, benzene, smoke, carbon dioxide. It operates at a 5V supply with 150mA consumption. Preheating of 20 seconds is required before the operation, to obtain the accurate output.

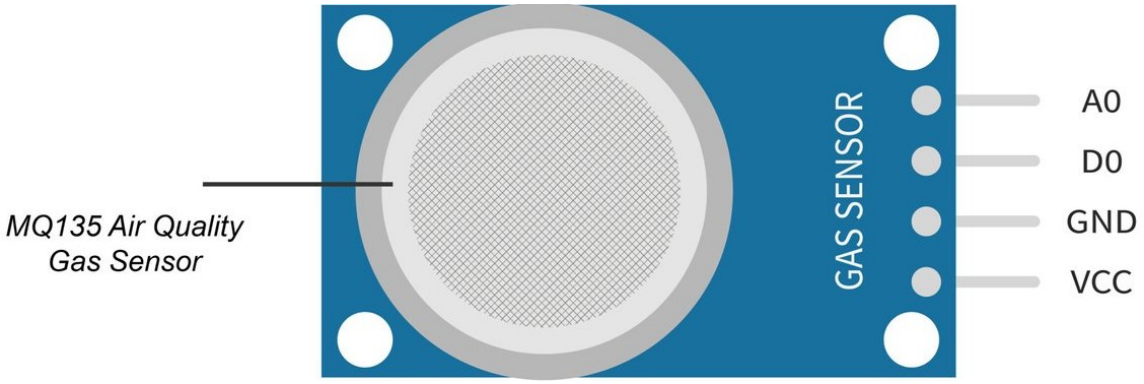


Figure 3.5: MQ135 Pin Diagram (Top View)

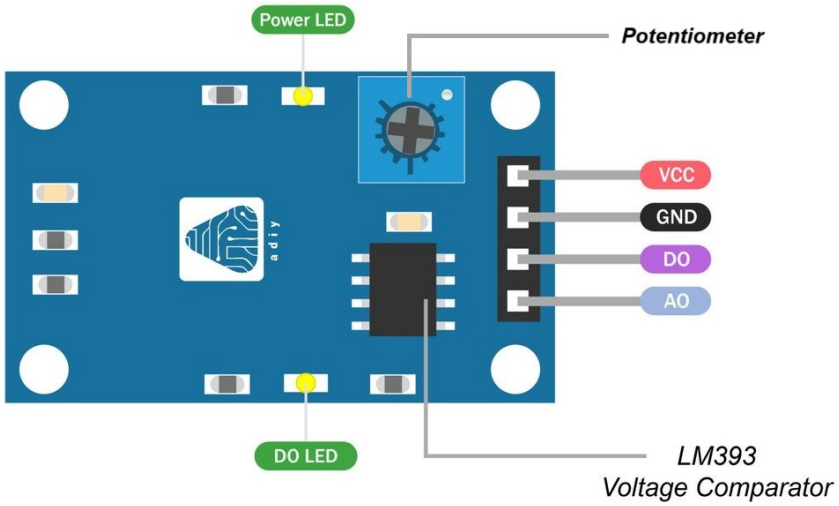


Figure 3.6: MQ135 Pin Diagram (Back View)

The sensor has 4 pins:

- **Pin 1 -VCC:** This pin refers to a positive power supply of 5V that power up the MQ135 sensor module.
- **Pin 2 -GND:** This is a reference potential pin, which connects the MQ135 sensor module to the ground.
- **Pin 3 -D0:** This pin refers to the digital output pin that gives the digital output by adjusting the threshold value with the help of a potentiometer.
- **Pin 4 -A0:** This pin generates the analog output signal of 0V to 5V and it depends on the gas intensity.This analog output signal is proportional to the gas vapor concentration, which is measured by the MQ135 sensor module.

3.2.4 SGP30 Gas Sensor

The SGP30 is a digital multi-pixel gas sensor, this multi-pixel technology allows the sensor to perceive its surroundings using various receptors that detect the type and concentration of gases.

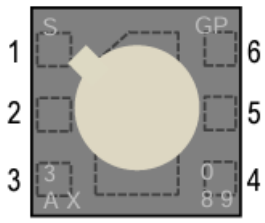


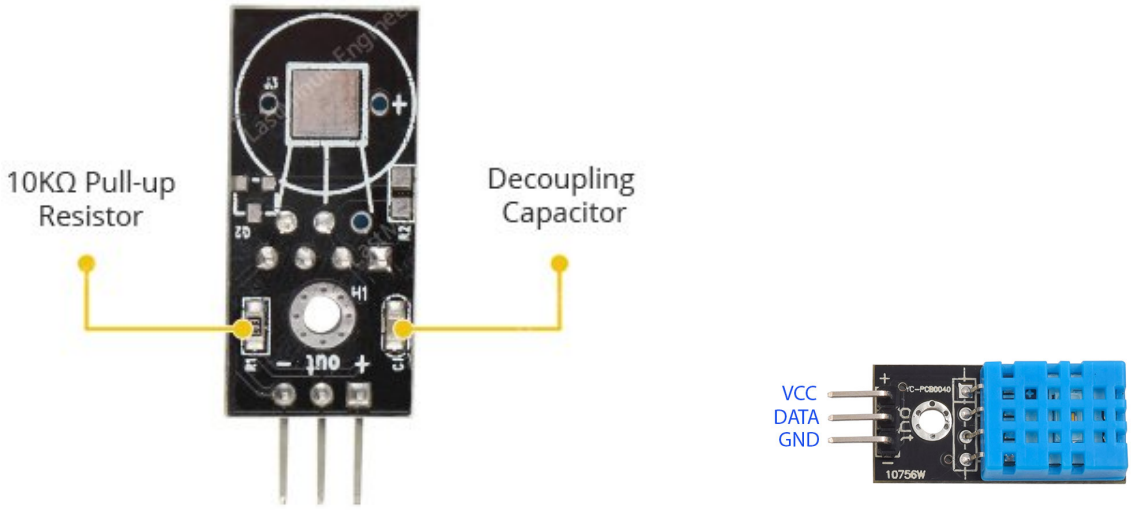
Figure 3.7: SGP30 Pin Configuration

The sensor has 6 pins:

- **Pin 1 -VDD:**This pin refers to a positive power supply which can vary from 2 to 5V and power the SGP30 sensor.
- **Pin 2 -VSS:**This is a reference potential pin, which connects the SGP30 sensor to the ground.
- **Pin 3 -SDA:**The SDA pin is used to transfer data to and from the sensor.
- **Pin 4 -R:** Connect to ground (no electrical function)
- **Pin 5 -VDDH:**This pin refers to a positive power supply which can vary from 2 to 5V and power the SGP30 sensor.
- **Pin 6 -SCL:**SCL is used to synchronize the communication between the microcontroller and the sensor.

3.2.5 DHT11 Temperature and Humidity Sensor

The DHT11 Temperature and Humidity Sensor is used to get the temperature and Humidity from the environment. This sensor operates at supply voltages ranging from 3.3 to 5V. The module comes with all the essential supporting circuitry, so it should be ready to run without any extra components. The DHT11 sensors usually require external pull-up resistor of 10KOhm between VCC and Out pin for proper communication between sensor and the Arduino.



(a) (b)

Figure 3.8: (a) DHT11 Back. (b) DHT11 Front.

The sensor has 3 pins:

- **Pin 1 -VCC:**This pin refers to a positive power supply which can vary from 3.3V to 5V and power the DHT11 sensor moduele. The recommended supply voltage is 5V to ensure more accurate results.
- **Pin 2 -DATA:**This pin is used to communication between the sensor and the Arduino.
- **Pin 3 -GND:**This is a reference potential pin, which connects the DHT11 sensor to the ground.

3.2.6 DC Fan

DC Fan is a simple actuator that transforms an electric energy into electromagnetic energy through DC voltage and electromagnetic induction,then into mechanical energy, and finally into kinetic energy, so as to make the fan blades rotate. Using a 12V Fan we used a Relay JQC-3FF-SZ.The relay acts as an electrically operated switch, allowing safe and efficient control of the fan.

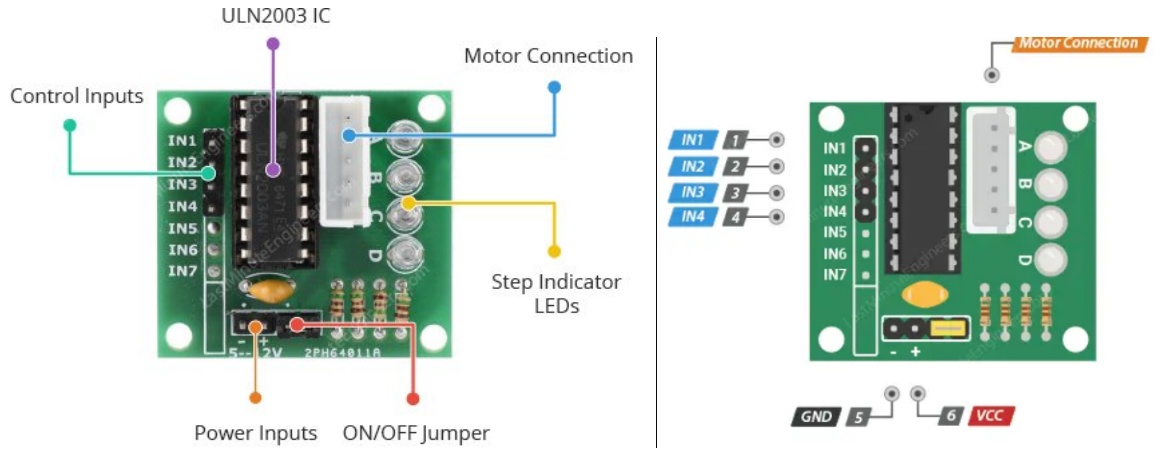
Figure 3.9: Relay Pins

The relay has the following pins:

- **Pin -VCC:**This voltage is consistent with the operating voltage on the relay.
- **Pin -GND:**This is a pin, which connects the relay to the ground.
- **Pin -IN:**This pin is low-level trigger, the voltage value between 0-1.2V". So pulling this pin low will trigger the relay. Setting the GPIO pin HIGH will untrigger the relay.
- **Pin -NC:**In the default state (when the relay is not energized), this pin is connected to the COM pin. When the relay is activated, the connection between NC and COM is broken.
- **Pin -COM:** This pin is the common terminal of the switch inside the relay. It connects to either the Normally Open (NO) or Normally Closed (NC) terminal depending on the state of the relay. On this pin we connect the + of 12V Power Supply.
- **Pin -NO:**In the default state, this pin is not connected to the Common pin. When the relay coil is energized, this pin is connected to the Common pin.On this pin we connect the + of DC Fan.

3.2.7 Stepper Motor

A stepper motor is an accurate actuator that transforms electrical energy into rotary motion through a series of discrete steps, offering accurate control of speed and position. To ensure the necessary voltage for operating in an efficient way, we use the ULN2003AN stepper driver.



(a) (b)

Figure 3.10: (a)Stepper Motor Driver Architecture. (b) Stepper Motor Driver Pins

The board architecture consists of the following:

- **ULN2003 Driver IC:** consists of an array of seven Darlington transistor pairs, each of which can drive a load of up to 500mA and 50V. This board utilizes four of the seven pairs.
- **LEDs:** that indicate activity on the four control input lines.
- **ON/OFF jumper:** for disabling the stepper motor if needed.
- **Motor Connector** is where the motor plugs in. The connector is keyed, so it will only go in one way.
- **Control Inputs**
- **Power Inputs**

This drivers has the following pins:

- **Pin 1 to 4 -IN:** This 4 pins are motor control input pins and are connected to Arduino Digital Output pins.
- **Pin 5 -GND:**This is a pin, which connects the motor driver to the ground.
- **Pin 6 -VCC:**This pin refers to a positive power supply of 5V.

3.3 Software Components

In our project, software components are represented by Arduino IDE and python scripts on Raspberry PI.

The Arduino Integrated Development Environment (IDE) is a software platform used to write and upload code to Arduino boards, including the Arduino Uno. Also,its offering a rich-library to help users get started. The programming languages used in this IDE are C and C++.

Python scripts running on Raspberry Pi offers a large set of options, including serial communication to data processing.

Chapter 4

Design and Implementation

4.1 Overview Design

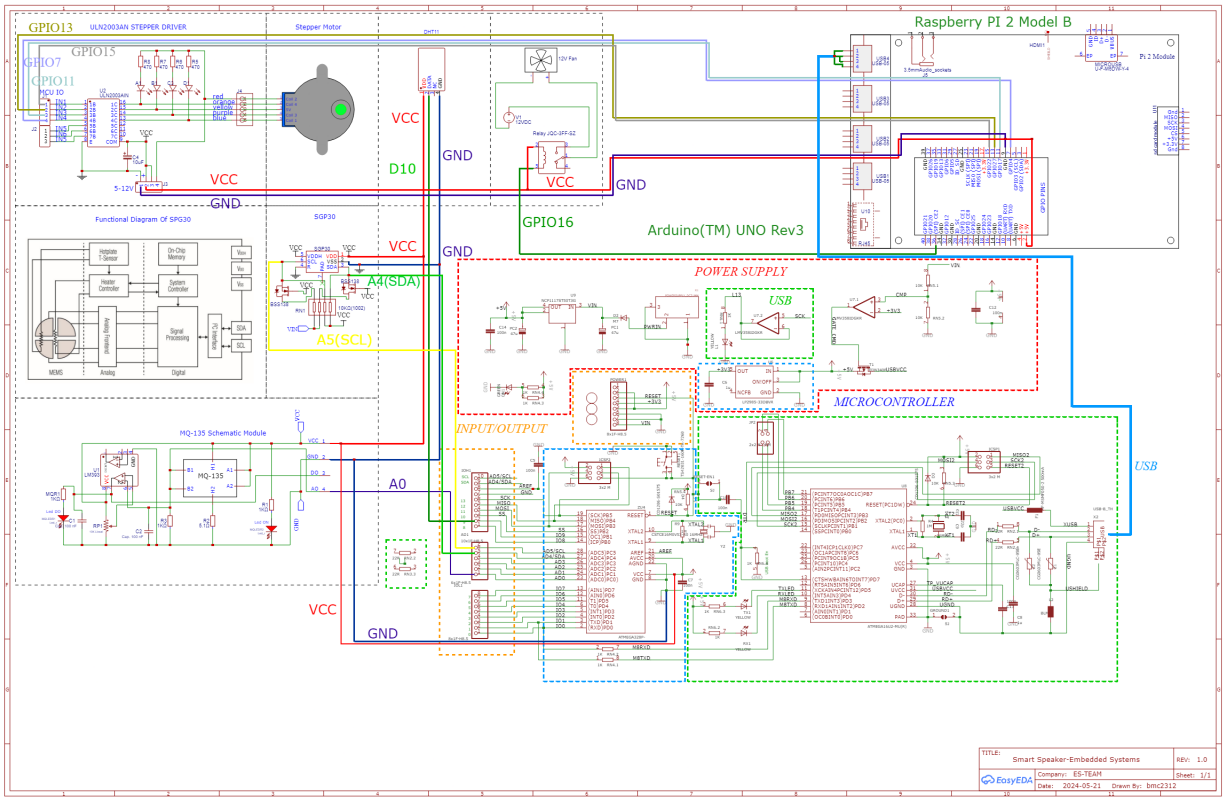


Figure 4.1: Design

4.2 Sensor Integration with Arduino Uno

In our project, incorporating analog sensors like the MQ-135 and SGP30 was necessary. We needed an Analog-to-Digital Converter to obtain digital values useful for the Raspberry Pi. These converters are integral components of the Arduino Uno's Analog Inputs, making the Arduino a hub for receiving sensor data.

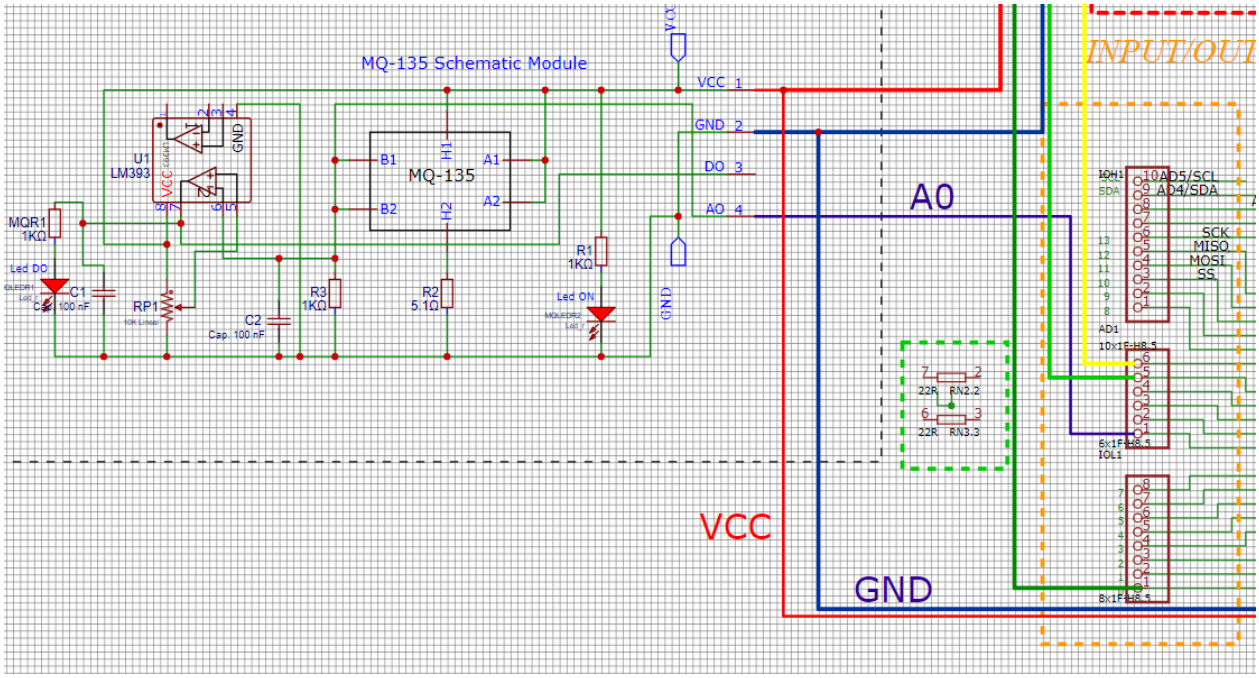


Figure 4.2: MQ-135 Design

The MQ135 sensor is connected to a 5V pin to receive the necessary voltage, and to read data from the sensor, we use Analog Pin 0, which uses its Analog-to-Digital Converter to transform environmental stimulus into digital values.

Code instructions used for initialzing Analog-to-Digital Converter and reading from it :

```
1
2
3
4  Function to initialize the ADC
5   void adc_init(void) {
6     // Set the reference voltage to AVcc (5V)
7     ADMUX |= (1 << REFS0);
8
9     // Set the ADC prescaler to 128 for a 125kHz ADC clock with a 16MHz system clock
10    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
11
12    // Enable the ADC
13    ADCSRA |= (1 << ADEN);
14  }
15
16
17
18 Function to read from the ADC
```

```
19  uint16_t adc_read(uint8_t channel) {
20    // Select the ADC channel (0-7)
21    ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);
22
23    // Start the conversion
24    ADCSRA |= (1 << ADSC);
25
26    // Wait for the conversion to complete
27    while (ADCSRA & (1 << ADSC))
28      ;
29
30
31    // Read the ADC result (ADCL must be read first)
32      uint16_t adc_result = ADCL;
33        adc_result |= (ADCH << 8);
34
35        return adc_result;
36  }
37
38
```

To obtain gas concentration from the ADC voltage, we need to get load resistor because the concentration of the gas make the resistance to change. The sensor resistance is normalized and the normalized value is used to calculate gas concentration in Parts-per-Million.

```
1  const float RLOAD = 10.0;         // Load resistance (kOhm)
2  const float RZERO = 76.63;        // Sensor resistance in clean air (kOhm)
3  const float PARA = 116.6020682;   // Constant for the equation
4  const float PARB = 2.769034857;   // Constant for the equation
5   // Function to calculate the resistance of the sensor
6 float getSensorResistance(int rawADC) {
7     float vrl = (rawADC / 1024.0) * 5.0;    // Convert ADC value to voltage
8     float rs = (5.0 - vrl) / vrl * RLOAD;   // Calculate sensor resistance
9     return rs;
10
11
12  Function to calculate the PPM of CO2 (or other gases) using the sensor resistance
13 float getPPM(float rs) {
14     float ratio = rs / RZERO;               // Ratio of Rs/Ro
15     float ppm = PARA * pow(ratio, -PARB);   // Calculate PPM using the equation
16     return ppm;
17
```

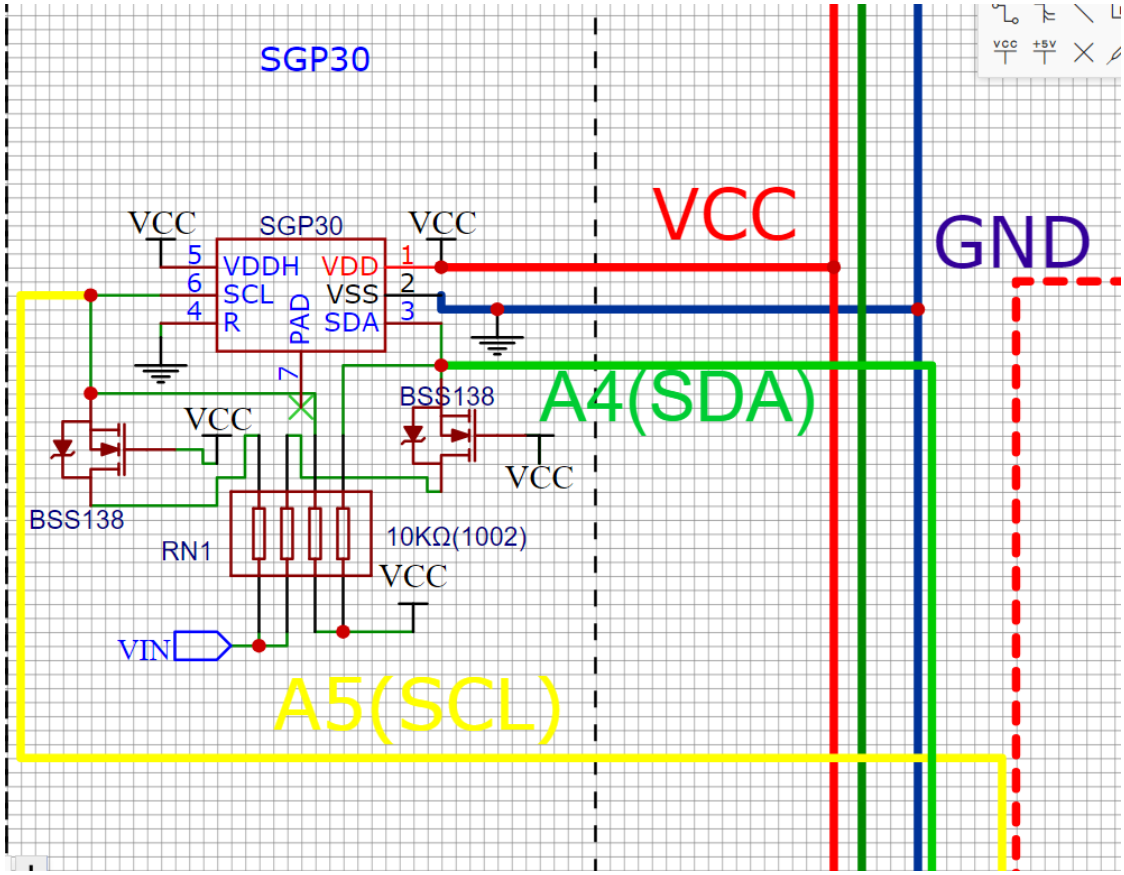


Figure 4.3: SGP30 Design

The SGP30 sensor uses the I²C (Inter-Integrated Circuit) communication protocol to send data to Arduino through Analog Pin 4(SDA-Data Line) and Analog Pin 5 (SCL-Clock Line) and to obtain the necessary voltage we use 5V from Arduino.

```
1
2   #include <Wire.h>
3   #define SGP30_I2C_ADDR 0x58  // Default I2C address of the SGP30 sensor
4   bool initSGP30() {
5     Wire.beginTransmission(SGP30_I2C_ADDR);
6     Wire.write(0x20);  // Command to initialize air quality readings
7     delay(10);
8     Wire.write(0x03);  // Command parameters
9     delay(10);
10    Wire.write(0x00);  // CRC
11    delay(10);
12    if (Wire.endTransmission() != 0) {
13      return false;  // Error initializing SGP30
14    }
15    delay(10);     // Wait for initialization to complete
16    return true;  // SGP30 initialized successfully
17  }
18  void softResetSGP30() {
19    Wire.beginTransmission(0x58); // SGP30 I2C address
20    Wire.write(0x00);
21    delay(10);
```

```
22    Wire.write(0x06);
23    Wire.endTransmission();
24    delay(10); // Delay to allow the sensor to reset
25  }
26  bool measureAirQuality(uint16_t& co2, uint16_t& tvoc) {
27    Wire.beginTransmission(SGP30_I2C_ADDR);
28    Wire.write(0x20);  // Command to measure air quality
29    delay(10);
30    Wire.write(0x08);  // Command parameters
31    int endTransmissionResult = Wire.endTransmission();
32
33    if (endTransmissionResult != 0) {
34      Serial.print("Transmission Error: ");
35      Serial.println(endTransmissionResult);  // Print the error code
36      return false;                           // Error sending command
37    }
38    delay(12);  // Wait for measurement to complete
39    // Read air quality data
40    Wire.requestFrom(SGP30_I2C_ADDR, 6);  // Request 6 bytes of data
41
42    if (Wire.available() < 6) {
43      return false;  // Error reading data
44    }
45    co2 = Wire.read() << 8 | Wire.read();   // CO2 data (MSB + LSB)
46    Wire.read();                            // Discard CRC byte
47    tvoc = Wire.read() << 8 | Wire.read();  // TVOC data (MSB + LSB)
48    Wire.read();                            // Discard CRC byte
49    return true;                            // Air quality measurement successful
50  }
51
52
```

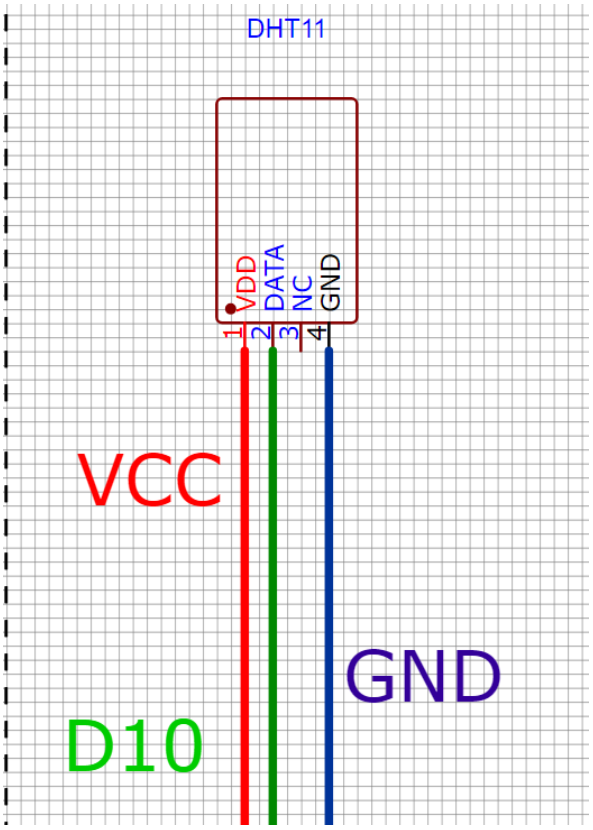


Figure 4.4: DHT11 Design

To connect DHT 11 we use 5V Pin from Arduino and the Digital Pin 10 to receive the data obtained by the DHT.

```

  #include "DHT.h"
  #define DHTPIN 10
  #define DHTTYPE DHT11
  // Initialize DHT sensor
  DHT dht_sensor(DHTPIN, DHTTYPE);
  humidity = dht_sensor.readHumidity();
  temperature = dht_sensor.readTemperature();
```

For obtaining the temperature and humidity we are using dht library which contains the functions to read both data.

4.3 Actuator Control

The actuators are controlled by the Raspberry Pi, which receives normalized sensor data and makes decisions based on the processed data.

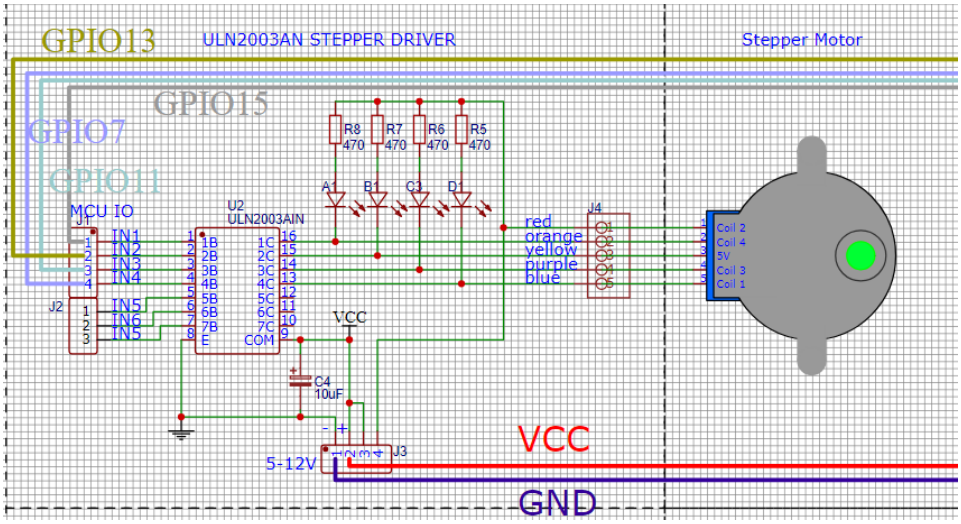


Figure 4.5: Stepper Motor Design

The Stepper Motor Driver is connected to the 5V Pin and the control inputs are connected to the GPIO Pins(7,11,13,15) from Raspberry Pi. Stepper Motor is connected to the Driver through motor connection port.

```
import RPi.GPIO as GPIO
import time
 # Define the GPIO pins connected to the stepper motor
ControlPin = [17, 18, 27, 22]

# Set up each pin as an output and initialize it to a low state (0)
for pin in ControlPin:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, 0)

# Function to control the windows by opening or closing them using the stepper motor
def actionWindows(action):
    # Initialize a sequence array for the stepper motor control
    seq = [[0]*4]*8

    # Define the sequence for opening the windows
    if action == "open":
        print("Opening windows")
        seq = [
            [1, 0, 0, 0],
            [1, 1, 0, 0],
            [0, 1, 0, 0],
            [0, 1, 1, 0],
            [0, 0, 1, 0],
            [0, 0, 1, 1],
            [0, 0, 0, 1],
            [1, 0, 0, 1]
        ]

```

```
30        # Define the sequence for closing the windows
31        elif action == "close":
32            print("Closing windows")
33            seq = [
34                [1, 0, 0, 1],
35                [0, 0, 0, 1],
36                [0, 0, 1, 1],
37                [0, 0, 1, 0],
38                [0, 1, 1, 0],
39                [0, 1, 0, 0],
40                [1, 1, 0, 0],
41                [1, 0, 0, 0]
42            ]
43
44        # Loop to repeat the stepper motor steps
45        for i in range(512):  # Adjust this value to control the total rotation
46            # Loop through each half-step in the sequence
47            for halfStep in range(8):
48                # Loop through each pin and set the corresponding state from the sequence
49                for pin in range(4):
50                    GPIO.output(ControlPin[pin], seq[halfStep][pin])
51                # Small delay to control the speed of the stepper motor
52                time.sleep(0.001)
53
```

The DC Fan is connected to the Relay JQC-3FF-SZ through the COM PIN and 12V Power Supply. The Relay JQC-3FF-SZ VCC Pin is connected to the 5V Pin from the Raspberry Pi and IN Pin to GPIO16 Pin. NC Pin (Normally Closed) is connected to + from the 12V power supply.

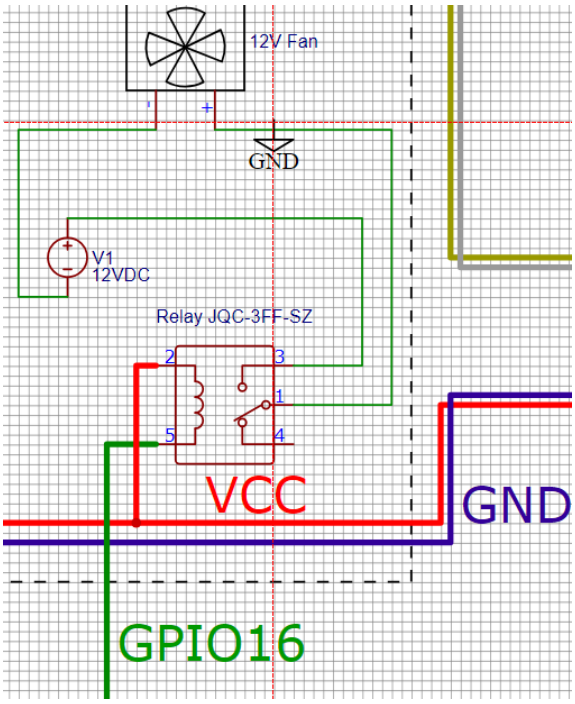


Figure 4.6: DC Fan Design

```
    import RPi.GPIO as GPIO

    # Initialize GPIO settings
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(36, GPIO.OUT)

    fan_running = False

    # Function to start the fan
    def start_fan():
        global fan_running
        GPIO.output(36, GPIO.HIGH)  # Turn on the fan
        fan_running = True

    # Function to stop the fan
    def stop_fan():
        global fan_running
        GPIO.output(36, GPIO.LOW)  # Turn off the fan
        fan_running = False
        print("Stopping fan")  # Print a message indicating fan is stopped
```

4.4 Serial Communication Setup with Raspberry Pi 2

Serial communication is a way to tranfer data. The data will be sent sequentially, one bit at a time.

UART protocol: Universal Asynchronous Reception and Transmission. It is an asynchronous multi-master protocol based on Serial communication, which will allow you to communicate between 2 boards. Multi-master means that all connected devices will be able to send data when they want. This is not the case for out project because we will send air quality data only from Arduino to Raspberry Pi. There are 2 ways to obtain a connection over UART: using the RX TX pins or using the USB Ports. We connected the Arduino's USB port to one of Raspberry Pi's USB ports and that was it, now we are able to use serial communication to send data from Arduino to Raspberry Pi.

In order to detect the Arduino board on Raspberry Pi we had to run the following command once before connecting the boards and once again after.

```
$ls /dev/tty*
```

From the Arduino side we simply get the reading from each sensor, we convert the float into string and we print them all in a char array. And at the end we send this char array over Serial.

```
    char hum_str[20];
    char temp_str[20];
    char ppm_str[20];
    char stringResult[100];
    dtostrf(humidity, 8, 2, hum_str);
    dtostrf(temperature, 8, 2, temp_str);
    dtostrf(ppm, 8, 2, ppm_str);
    sprintf(stringResult, "%s %d %d %s %s", ppm_str, co2, tvoc, temp_str, hum_str);

```

```
    // Send the air quality data over Serial
    Serial.println(stringResult);
```

```
    # Initialize serial communication
    serial1 = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
    print("Reading data from serial port")

```

We used **pySerial** library in order to be able to use the Serial interface with Python. We create and object called serial1 and use it to read line by line from the serial.

```
if serial1.is_open:  # Check if serial is opened
        serial_line = serial1.readline()  # Read the data from the serial port
        air_data = format_message(serial_line.decode('utf-8'))
```

4.5 Coding and Programming Details

Arduino Code:

```
#include <Wire.h>
#include "DHT.h"

#define SGP30_I2C_ADDR 0x58  // Default I2C address of the SGP30 sensor
#define DHTPIN 10
#define DHTTYPE DHT11

const float RLOAD = 10.0;         // Load resistance (kOhm)
const float RZERO = 76.63;        // Sensor resistance in clean air (kOhm)
const float PARA = 116.6020682;   // Constant for the equation
const float PARB = 2.769034857;   // Constant for the equation

// Initialize DHT sensor
DHT dht_sensor(DHTPIN, DHTTYPE);

// Function to initialize the ADC
void adc_init(void) {
  // Set the reference voltage to AVcc (5V)
  ADMUX |= (1 << REFS0);

  // Set the ADC prescaler to 128 for a 125kHz ADC clock with a 16MHz system clock
  ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

  // Enable the ADC
  ADCSRA |= (1 << ADEN);
}

// Function to read from the ADC
uint16_t adc_read(uint8_t channel) {
  // Select the ADC channel (0-7)
  ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);

  // Start the conversion
  ADCSRA |= (1 << ADSC);

  // Wait for the conversion to complete
  while (ADCSRA & (1 << ADSC))
    ;

  // Read the ADC result (ADCL must be read first)
  uint16_t adc_result = ADCL;
  adc_result |= (ADCH << 8);

  return adc_result;
}
```

```
47
48  // Simple delay function
49  void delay_ms(uint16_t ms) {
50    while (ms) {
51      // This loop is roughly calibrated to take 1 ms
52      // on a 16 MHz system clock
53      for (uint16_t i = 0; i < 4000; i++) {
54        asm volatile("nop");
55      }
56      ms--;
57    }
58  }
59
60  // Function to calculate the resistance of the sensor
61  float getSensorResistance(int rawADC) {
62    float vrl = (rawADC / 1024.0) * 5.0;    // Convert ADC value to voltage
63    float rs = (5.0 - vrl) / vrl * RLOAD;   // Calculate sensor resistance
64    return rs;
65  }
66
67  // Function to calculate the PPM of CO2 (or other gases) using the sensor resistance
68  float getPPM(float rs) {
69    float ratio = rs / RZERO;               // Ratio of Rs/Ro
70    float ppm = PARA * pow(ratio, -PARB);   // Calculate PPM using the equation
71    return ppm;
72  }
73
74  bool initSGP30() {
75    Wire.beginTransmission(SGP30_I2C_ADDR);
76    Wire.write(0x20);  // Command to initialize air quality readings
77    delay(10);
78    Wire.write(0x03);  // Command parameters
79    delay(10);
80    Wire.write(0x00);  // CRC
81    delay(10);
82    if (Wire.endTransmission() != 0) {
83      return false;  // Error initializing SGP30
84    }
85    delay(10);     // Wait for initialization to complete
86    return true;  // SGP30 initialized successfully
87  }
88
89  void softResetSGP30() {
90    Wire.beginTransmission(0x58); // SGP30 I2C address
91    Wire.write(0x00);
92    delay(10);
93    Wire.write(0x06);
94    Wire.endTransmission();
95    delay(10); // Delay to allow the sensor to reset
96  }
```

```
97
98  void setup() {
99    // Initialize the ADC
100   adc_init();
101   Wire.begin();
102   dht_sensor.begin();
103   // Initialize Serial communication at 9600 baud
104   Serial.begin(9600);
105   delay_ms(1000);
106   if (!initSGP30()) {
107     Serial.println("SGP_ERROR");
108     while (1)
109       ;  // Hang indefinitely if initialization fails
110   } else {
111     Serial.println("SGP_OKAY");
112     softResetSGP30();
113   }
114 }
115
116 bool measureAirQuality(uint16_t& co2, uint16_t& tvoc) {
117   Wire.beginTransmission(SGP30_I2C_ADDR);
118   Wire.write(0x20);  // Command to measure air quality
119   delay(10);
120   Wire.write(0x08);  // Command parameters
121   int endTransmissionResult = Wire.endTransmission();
122
123   if (endTransmissionResult != 0) {
124     Serial.print("Transmission Error: ");
125     Serial.println(endTransmissionResult);  // Print the error code
126     return false;                           // Error sending command
127   }
128   delay(12);  // Wait for measurement to complete
129   // Read air quality data
130   Wire.requestFrom(SGP30_I2C_ADDR, 6);  // Request 6 bytes of data
131
132   if (Wire.available() < 6) {
133     return false;  // Error reading data
134   }
135   co2 = Wire.read() << 8 | Wire.read();   // CO2 data (MSB + LSB)
136   Wire.read();                            // Discard CRC byte
137   tvoc = Wire.read() << 8 | Wire.read();  // TVOC data (MSB + LSB)
138   Wire.read();                            // Discard CRC byte
139   return true;                            // Air quality measurement successful
140 }
141
142
143
144
145 void loop() {
146   // Variable to store ADC result
```

```
147     uint16_t a0_adc_value, co2 = 0, tvoc = 0;
148     float temperature, humidity;
149     bool airQualityMeasured = false;
150 
151     char buffer[10];
152     airQualityMeasured = measureAirQuality(co2, tvoc);
153 
154     //Measure co2 and tvoc using SGP30
155 
156     // Read the ADC value from channel 0
157     a0_adc_value = adc_read(0);
158 
159     // Convert the ADC value to a string
160     itoa(a0_adc_value, buffer, 10);
161     float rs = getSensorResistance(a0_adc_value);  // Calculate the sensor resistance
162     float ppm = getPPM(rs);
163     // Calculate the PPM of the gas
164     humidity = dht_sensor.readHumidity();
165     temperature = dht_sensor.readTemperature();
166 
167     char hum_str[20];
168     char temp_str[20];
169     char ppm_str[20];
170     char stringResult[100];
171     dtostrf(humidity, 8, 2, hum_str);
172     dtostrf(temperature, 8, 2, temp_str);
173     dtostrf(ppm, 8, 2, ppm_str);
174     sprintf(stringResult, "%s %d %d %s %s", ppm_str, co2, tvoc, temp_str, hum_str);
175 
176     // Send the air quality data over Serial
177     Serial.println(stringResult);
178 
179     // Delay for 1000 milliseconds
180     delay_ms(1000);
181 }
```

Python Code:

```
# Import necessary libraries
import time
import serial
import RPi.GPIO as GPIO

# Set up GPIO
GPIO.cleanup()  # Clean up any previous GPIO configurations

GPIO.setmode(GPIO.BOARD)  # Use Board pin numbering
GPIO.setup(36, GPIO.OUT)  # Set Output mode for pin 36 (GPIO 16) to control the fan
GPIO.output(36, GPIO.LOW)  # Stop the fan
global fan_running  # Variable for storing the fan state
global win_open  # Variable for storing the window state
fan_running = False  # Assign false to the fan state variable initially
win_open = False  # Assign false to the window state variable initially
# Define the GPIO pins connected to the stepper motor
ControlPin = [17, 18, 27, 22]

 # Set up each pin as an output and initialize it to a low state (0)
for pin in ControlPin:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, 0)

# Function to control the windows by opening or closing them using the stepper motor
def actionWindows(action):
# Initialize a sequence array for the stepper motor control
seq = [[0]*4]*8

# Define the sequence for opening the windows
if action == "open":
    print("Opening windows")
    seq = [
        [1, 0, 0, 0],
        [1, 1, 0, 0],
        [0, 1, 0, 0],
        [0, 1, 1, 0],
        [0, 0, 1, 0],
        [0, 0, 1, 1],
        [0, 0, 0, 1],
        [1, 0, 0, 1]
    ]

# Define the sequence for closing the windows
elif action == "close":
    print("Closing windows")
    seq = [
        [1, 0, 0, 1],
```

```
            [0, 0, 0, 1],
            [0, 0, 1, 1],
            [0, 0, 1, 0],
            [0, 1, 1, 0],
            [0, 1, 0, 0],
            [1, 1, 0, 0],
            [1, 0, 0, 0]
        ]

    # Loop to repeat the stepper motor steps
    for i in range(512):  # Adjust this value to control the total rotation
        # Loop through each half-step in the sequence
        for halfStep in range(8):
            # Loop through each pin and set the corresponding state from the sequence
            for pin in range(4):
                GPIO.output(ControlPin[pin], seq[halfStep][pin])
            # Small delay to control the speed of the stepper motor
            time.sleep(0.001)

    # Define methods for handling high temperature, humidity, and air quality (unused in t

    # Function to start the fan
    def start_fan():
        global fan_running
        GPIO.output(36, GPIO.HIGH)  # Turn on the fan
        fan_running = True

    # Function to stop the fan
    def stop_fan():
        global fan_running
        GPIO.output(36, GPIO.LOW)  # Turn off the fan
        fan_running = False
        print("Stopping fan")  # Print a message indicating fan is stopped

    # Method to open the window
    def open_window():
        global win_open
        if not win_open:
            actionWindows("open")  # Open the window
            win_open = True

    # Method to close the window
    def close_window():
        global win_open
        if win_open:
            actionWindows("close")  # Close the window
            win_open = False

    # Method to check if the data is a string of 5 numbers separated by spaces
    def is_num_array(data):
```

```
98      try:
99          data_arr = data.split()
100         if len(data_arr) != 5:
101             return False
102         for item in data_arr:
103             float(item)
104         return True
105     except ValueError:
106         return False
107
108 # Method for formatting the data
109 def format_message(data):
110     data = data.strip()  # Remove leading and trailing whitespaces
111     if is_num_array(data):  # Check if the data is a string of 5 numbers
112     #separated by spaces
113         data_arr = data.split()  # Split the data into an array
114         return {
115             'gas': float(data_arr[0]),
116             'co2': float(data_arr[1]),
117             'tvoc': float(data_arr[2]),
118             'temp': float(data_arr[3]),
119             'hum': float(data_arr[4])
120         }
121     else:
122         print("SERIAL_NOT_OKAY")
123         return None
124
125     # Initialize serial communication
126     serial1 = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
127     print("Reading data from serial port")
128
129     # Define weights for air quality variables
130     w_gas = 1
131     w_co2 = 1.1
132     w_tvoc = 1
133     w_hum = 2.0
134     w_temp = 2.0
135     aqi = 0
136
137 try:
138   while True:
139     if serial1.is_open:  # Check if serial is opened
140      serial_line = serial1.readline()  # Read the data from the serial port
141      air_data = format_message(serial_line.decode('utf-8'))  # Format the data
142
143      if air_data is not None:  # Check
144         if the data is not None
145         print(air_data)
146         # Compute the AQI
147         aqi = (w_gas * air_data['gas']) + (w_co2 * air_data['co2']) +
```

```
148             (w_tvoc * air_data['tvoc']) + (w_temp * air_data['temp']) +
149             (w_hum * air_data['hum'])
150             print(aqi)
151 
152             # Control fan and window based on AQI thresholds
153             if aqi > 1600 and not fan_running: # If AQI is greater than 1600 and
154             # the fan is not running
155                 start_fan()
156 
157             elif aqi > 1600 and not win_open:  # If AQI is greater than 1600
158             # and the window is not open
159                 open_window()
160 
161             elif aqi < 1500 and fan_running:  # If AQI is less than 1500
162             # and the fan is running
163                 stop_fan()
164 
165             elif aqi < 1500 and win_open:  # If AQI is less than 1500
166             # and the window is open
167                 close_window()
168         else:
169             print('Serial is not open')
170 
171             time.sleep(1)  # Wait for 1 second
172     except KeyboardInterrupt:
173         print("Program interrupted. Exiting gracefully.")
174         serial1.close()  # Close serial communication
175         close_window()  # Close the window
176         stop_fan()  # Stop the fan
177 
```

Chapter 5

Conclusions

5.1 Summary of Achievements

Throughout the project, significant results have been achieved, presenting successful integration of sensors and actuators into the embedded system.

Firstly, various sensors(MQ135,SGP30 and DHT11) have been integrated, enabling data collecting for analysis and control.

Secondly, the setting of serial communication has been a significant success This structure enabled smooth data transmission between the system's components, providing accurate and rapid information flow for decision process.

Lastly, the embedded system demonstrated a good overall performance. It is utilising in an efficient way sensor data to drive actuators, in this way achieving the expected system behavior.

5.2 Improvements

While the project has achieved its objective, there are various area for future work and improvements.

Adding additional sensors to upgrade system's sensing of the environment.

Optimizing the algorithms represents another area for improvement, leading to better responsiveness and improved overall performance.
Using more capable sensors to get a higher precision and increased sensitivity.Also, higher-performing boards will enhance the system offering faster read and response time.

By addressing these areas, the embedded system can be further refined, offering more functionality and efficiency in future iterations.

Bibliography

[1] Arduino Uno Schematics. Retrieved from: https://docs.arduino.cc/resources/schematics/A000066-schematics.pdf

[2] MQ135 Gas Sensor Manual. Retrieved from: https://www.winsen-sensor.com/d/files/PDF/Semiconductor%20Gas%20Sensor/MQ135%20(Ver1.4)%20-%20Manual.pdf

[3] SGP30 Gas Sensor Manual. Retrived from: https://docs.arduino.cc/resources/schematics/A000066-schematics.pdf,

[4] DHT11 Module Arduino Tutorial. Retrived from: https://razecus.go.ro/lme/lastminuteengineers.com/dht11-module-arduino-tutorial/index.html,

[5] The Basic Arduino Schematic Diagram. Retrieved from: https://learn.circuit.rocks/the-basic-arduino-schematic-diagram

[6] 28BYJ-48 Stepper Motor Arduino Tutorial. Retrieved from: https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/

[7] One Channel Relay Module Arduino Tutorial. Retrieved from: https://lastminuteengineers.com/one-channel-relay-module-arduino-tutorial/

[8] Raspberry Pi Documentation. Retrieved from: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html