Smart Kitchen and Alarm System

Ehatasham Haque
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
ehatasham.haque@northsouth.edu

Hasibul Hasan
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
hasibul.hasib21@northsouth.edu

Tawhid Avik
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
tawhid.avik@northsouth.edu

Sunjana Tarannum
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
sunjana.tarannum@northsouth.edu

***Abstract*— In the consumer electronic space, Internet of Things (IoT) has increasingly curved a niche market for itself. However, that space is somewhat governed by gimmicks as opposed to practicality. This project aims to remedy that by introducing a smart system to monitor, secure, and automate a kitchen environment in real-time. This is done through the use of microcontrollers such as an Arduino Uno R3 and sensors like IR-Flame Sensor, MQ-2 Smoke Sensor, MQ-135 Gas Sensor, DHT11 Temperature and Humidity Sensor. Furthermore, for all-time online connectivity NodeMCU Wi-Fi module is used to monitor the sensors via an app designed in the Blynk IoT platform. This project's objective is to pave a pathway to more practical applications for IoT devices in the future. Hence, this paper can be taken as a case study on how to design a proper, operational IoT device.**

Keywords*— *Arduino Uno R3, MQ-2 Sensor, MQ-135 Sensor, IR-Flame Sensor, DHT-11 Sensor, NodeMCU Wi-Fi Module, Blynk IoT platform.

I. Introduction

In an age of lightspeed computation, we are compelled to think and react at the speed of light. However, whilst digital computation reaches lightspeed, mankind's mental computation is still cumbersome and analog in nature; prone to negligence. Thus, in recent years, Internet of Things (IoT) has somewhat mitigated human error in a plethora of places like offices and homes, the latter is briefly discussed in this paper.

IoT encompasses many low-powered devices that aid in the day-to-day lives of various people and in various places. One such place is the humble kitchen, a homemade factory that sustains and maintains our biology. Just like any factory, a kitchen is a dangerous place. If not maintained or monitored, catastrophic failure is imminent.

Furthermore, aside from cooking disasters and health hazards, a poorly ventilated kitchen may damage utensils. At the heart of this topic human negligence stands solus. However, that being said, no machine can change that, but mitigate it. Hence, this paper deals with smart kitchen and alarm system, where monitoring and alarm are given priority over automation. This is because, we believe technology should be there as our co-pilot; not our pilot.

In contrast to our vision, prior projects similar to ours have often dealt with just one or two sensors, neglecting to monitor the entire kitchen environment. At most, they have covered Flame and Humidity. Moreover, they have used Passive-IR sensors (PIR) to detect motion, but we have opted to ignore it due to its lack of value in a practical setting. Furthermore, in all prior projects; automation, monitoring, and alarm have been done discretely. We have opted to aggregate monitoring, automation, and alarm into a single integrated system – as in one trigger another.

In this work we took four sensors that measure different environmental aspects of a home kitchen. We then used a microcontroller to read sensor data, then setup output devices such as an OLED Display and a passive buzzer according to the coded logic compiled in the microcontroller. Moreover, we used a Wi-Fi module to send data to the Blynk IoT platform for online monitoring of the kitchen sensors via a kitchen app and a web dashboard.

II. A Brief Description of the Components

A. Arduino Uno R3

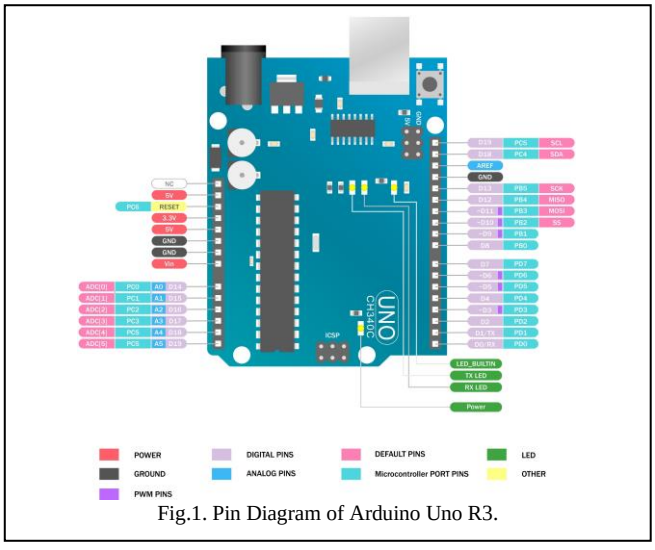

Fig.1. Pin Diagram of Arduino Uno R3.

The Arduino Uno R3 rests comfortably at the heart of our project. It is a PCB-mounted microcontroller that can be programmed to do simple tasks and mathematical computations [2].

For our purposes, the Arduino Uno R3 will take-in analog input value from the sensors and convert it to digital input value through the use of its Analog-to-Digital Converter (ADC). This converted digital input value is then processed by the at ATmega328P microcontroller. The microcontroller then outputs a signal to the buzzer or the motor to turn on or off depending upon the previously taken sensor readings, and coded instructions [10].

In our project the Arduino Uno R3 will be powered through the barrel jack port. Input and output will be taken from the analog pins (A0-A5) and digital pins (D0-D13 & D18-19).

B. IR-Flame Sensor

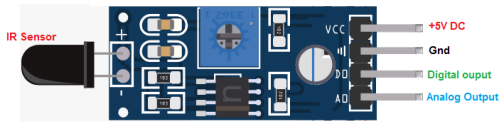

Fig. 2. Pin Diagram of IR-Flame Sensor.

The IR Flame sensor module has four pins and a built-in potentiometer to calibrate flame detection accuracy. Power is provided to the module through the V_{CC} pin, GND pin is connected to the ground and output is taken from the D0 pin.

C. DHT11 Temperature and Humidity Sensor

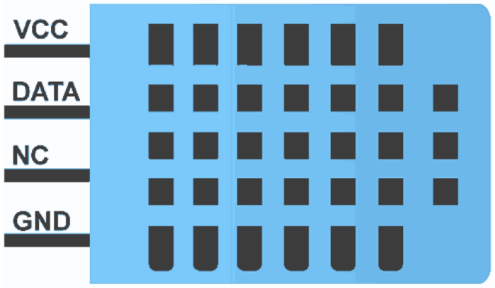

Fig. 3. Pin Diagram for DHT11 Sensor.

DHT11 sensor consists of a capacitive humidity sensing component coupled with a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels.

For measuring temperature, this sensor uses a NTC thermistor. The term NTC refers to Negative Temperature Coefficient, which means that the resistance decreases with an increase of the temperature. To get a noticeable change in resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramic or polymer [4].

D. MQ-2 Smoke sensor

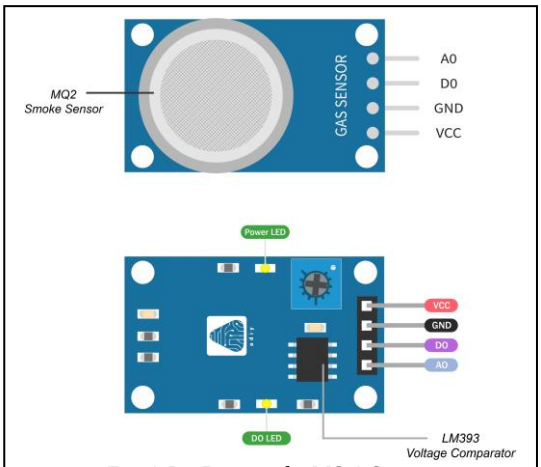

Fig. 4. Pin Diagram for MQ-2 Sensor.

The MQ-2 sensor module consists of a sensing component attached to a PCB that amplifies and processes the signal. The sensing component is made up of a tin dioxide (SnO2) semiconductor that is sensitive to the presence of certain gases and smokes. When gas or smoke molecules come into contact with the SnO2 surface, the conductivity of the sensor changes, and the output voltage of the PCB is modified accordingly [5].

The MQ-2 smoke sensor module has four pins and a built-in potentiometer to calibrate smoke detection accuracy. Power is provided to the module through the V_{CC} pin, GND pin is connected to the ground and output is taken from the D0 pin.

E. MQ-135 Smoke sensor

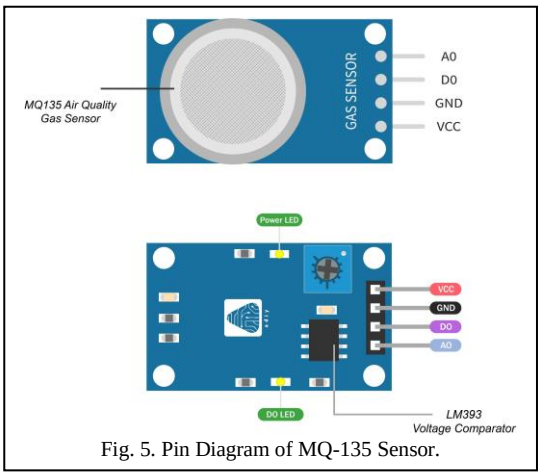

Fig. 5. Pin Diagram of MQ-135 Sensor.

The MQ-135 alcohol sensor consists of a tin dioxide (SnO2), a perspective layer inside aluminum oxide microtubes (measuring electrodes), and a heating element inside a tubular casing. The end face of the sensor is enclosed by a stainless-steel net and the backside holds the connection terminals. Ethyl alcohol present in the air is oxidized into acetic acid passing through the heating

element. When the ethyl alcohol cascade on the tin dioxide sensing layer, the resistance decreases. By using the external load resistance, the resistance change is converted into a suitable voltage variation that is compared against a set threshold value [6].

The MQ-135 gas leakage sensor module has four pins and a built-in potentiometer to calibrate gas detection accuracy. Power is provided to the module through the V_{CC} pin, GND pin is connected to the ground and output is taken from the pin 11.

F. OLED Display

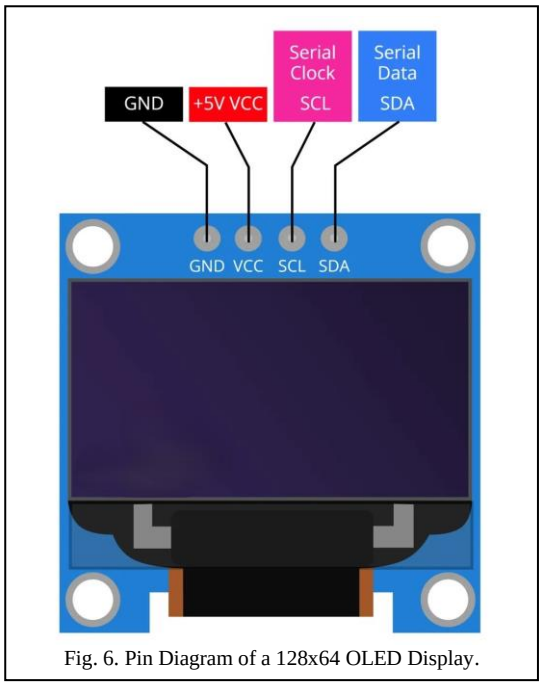

Fig. 6. Pin Diagram of a 128x64 OLED Display.

The 128x64 OLED Display is a type of 1.3-inch LED matrix display that can show 12,864 characters in either white or cyan; 64 characters high and 128 characters wide [7].

G. NodeMcu Lua V3 ESP8266 Wi-Fi Module

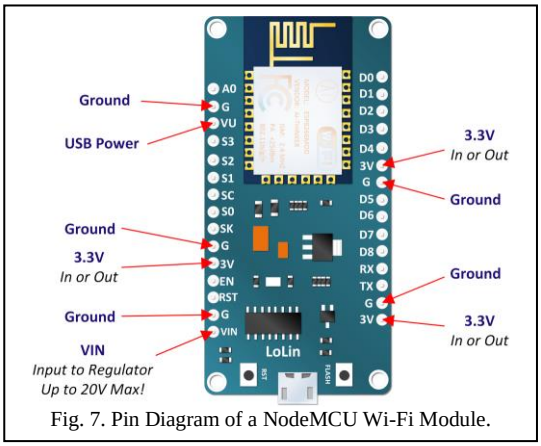

Fig. 7. Pin Diagram of a NodeMCU Wi-Fi Module.

NodeMCU Lua V3 ESP8266 Wi-Fi Module is a type of microcontroller similar to Arduino Uno R3. It is a Wi-Fi development board with an embedded ESP8266 chip to process Wi-Fi connectivity. For the purpose of this project, we will only use the TX and RX pin of the NodeMCU board [9].

H. Passive Buzzer

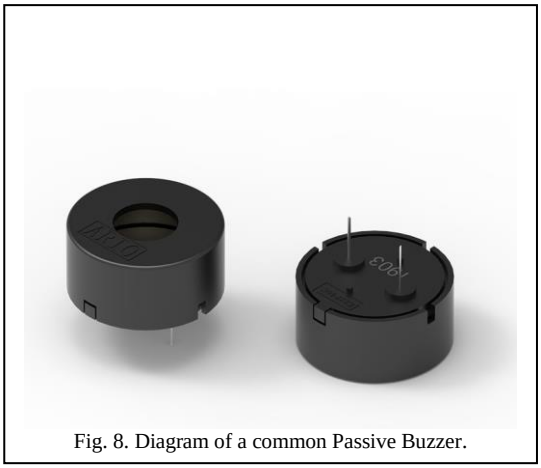

Fig. 8. Diagram of a common Passive Buzzer.

Passive buzzer is type of magnetic buzzer. There is a coil of wire inside the buzzer that is attached to its pins. In addition, the wire coil is encircled by a circular magnet. Above the circular magnet and wire coil is a thin, flexible ferromagnetic metal disc with a little metal weight fastened to the top. The metal weight and metal disc vibrate up and down as current pulses are given to the wire coil due to magnetic inductance. Sound waves are produced by the metal disk's oscillation [8].

For passive buzzers the anode and cathode are interchangeable. Hence, we can take one pin to be VCC and the other to be ground.

III. METHODS

Our project primarily relies on the Arduino Uno R3, for onboard control and decision-making process. Arduino Uno R3 is powered by an external 5V, 2A DC phone charger, connected to the AC power outlet.

Four sensors (IR Flame sensor, MQ-2 smoke sensor, DHT11 temperature and humidity sensor, and MQ-135 gas leakage detection sensor) are connected to the Arduino Uno R3 inputs pins (~3, ~9, ~6, ~11) respectively. Three output components (OLED display 128x64, passive buzzer, and a 5V rated motor) are connected to the Arduino Uno R3 output pins (A4, A5; ~10; ~9) respectively. All the sensors along with the OLED display take power from the 5V power trail of the Arduino Uno R3.

A 5V rated motor acts as an exhaust fan. When kitchen humidity exceeds 60%, Arduino Uno R3 turns on the motor; else it remains turned off. In addition to the motor, a passive buzzer is turned on, for every time a sensor detects its respective particulate. All the sensors are active low sensors, meaning they indicate the presence of their respective particulates by sending a low to zero voltage signal to the Arduino Uno R3. Moreover, as we are using a passive buzzer, so each sensor sounds a different tone of alarm. Hence, these unique tones indicate which sensor is turned on at that time.

Our project secondarily relies on NodeMCU ESP8266 Wi-Fi Module and a Phone App, for off-board control and decision-making process. ESP8266 Wi-Fi Module's receive pin (RXD0) is connected to the transfer pin of Arduino Uno R3 (TX), and Arduino Uno R3's receive pin (RX) is connected to the transfer pin (TXD0) of the NodeMCU ESP8266 Wi-Fi Module [11].

Whenever a sensor detects its respective particulate, a signal is sent via Wi-Fi to a cloud server that in turn updates the sensor reading shown in the user's Phone App.

A. *Hardware*

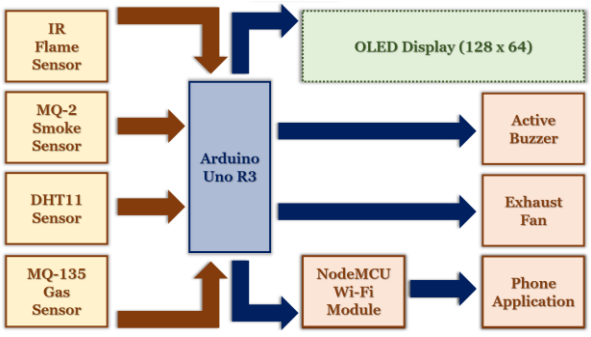

Fig. 9. Hardware Block Diagram.

According to Fig. 9, the yellow boxes are all sensor inputs going into the Arduino Uno R3 (represented by the blue rectangle). The Arduino Uno R3 then takes decisions according to the coded logic in the microcontroller memory chip. These decisions are then passed onto the output devices (shown in red boxes) to either sound and alarm, turn on the exhaust fan and show sensor status on the phone app. Moreover, a display (shown in green rectangle) is also used as an output module to showcase the sensor readings at all time.

B. *Software*

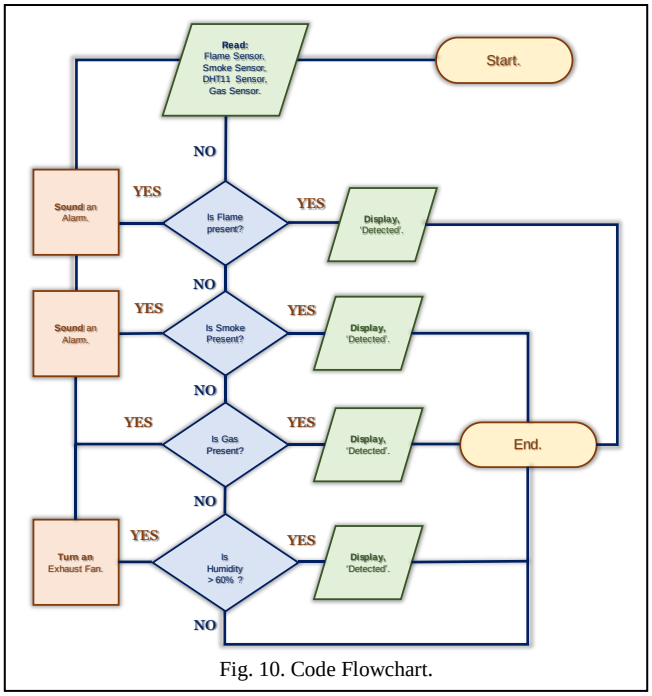

Fig. 10. Code Flowchart.

Fig. 10 demonstrates the programming logic behind the project. According to this diagram when the program starts the Arduino Uno R3 takes in sensor values (shown in the larger parallelogram at the top). It then computes these values according to the Boolean logic shown inside the blue diamond shapes. Based on the Boolean logic the microcontroller displays text on the screen (shown by the smaller parallelograms) and sounds an alarm or turns on the fan (shown via the red square shapes). This loop runs as long as the device is active and functional, to continuously monitor the sensor readings.

IV. Results and Discussion

This project can be divided into two parts; the discrete part and the online part. In the discrete part we showed the sensor readings via an OLED display connected to the main circuit of the project. The offline part of the project is shown via a phone app interface and a web dashboard interface. Fig. 11 and Fig. 12, Fig.13 showcases these respectively.

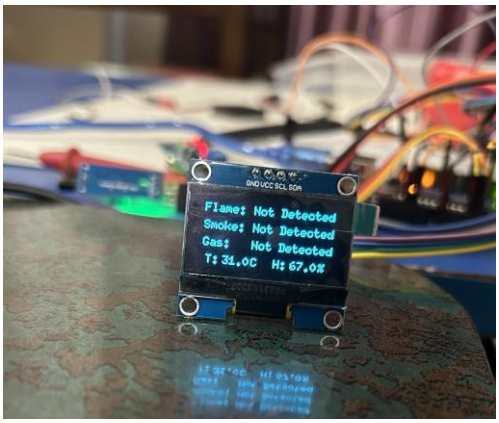

Fig. 11. Discrete Part: OLED Display.

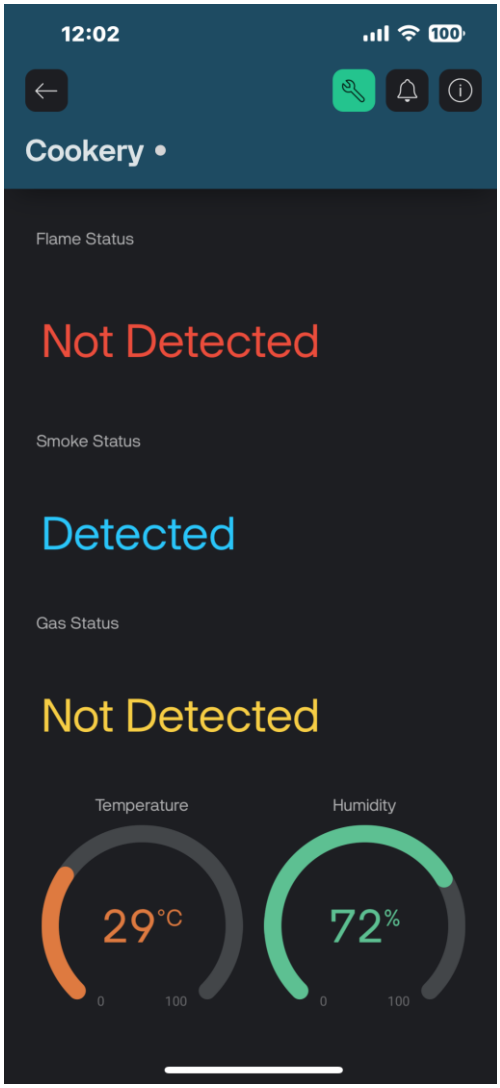

Fig. 12. Online Part: Phone App Interface.

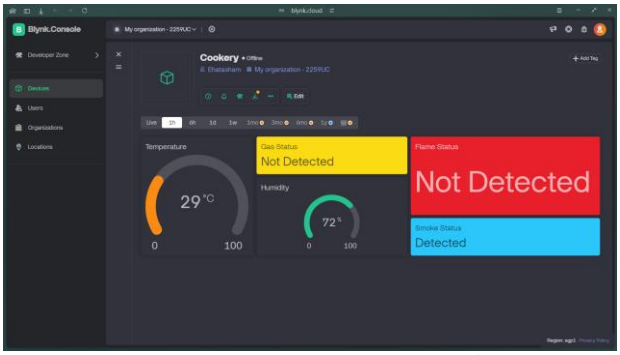

Fig. 13. Online Part: Web Dashboard Interface.

This project has been tested theoretically through proteus simulation (shown in Fig. 14). Furthermore, it has been tested practically through physical implementation of the concept (shown in Fig. 15 and Fig.16).

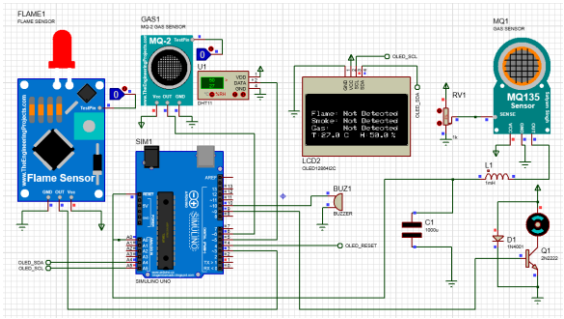

Fig. 14. Proteus Simulation.

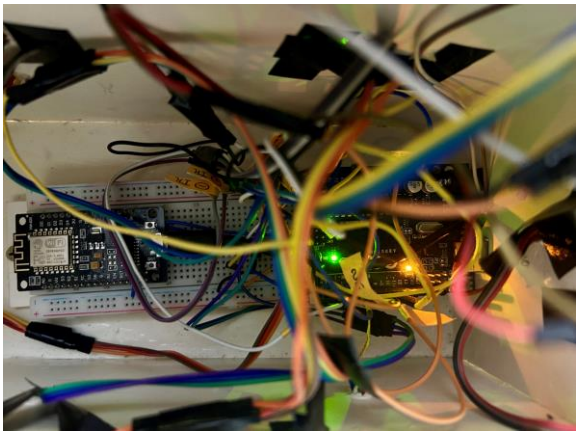

Fig. 15. Internal Circuit Implementation.

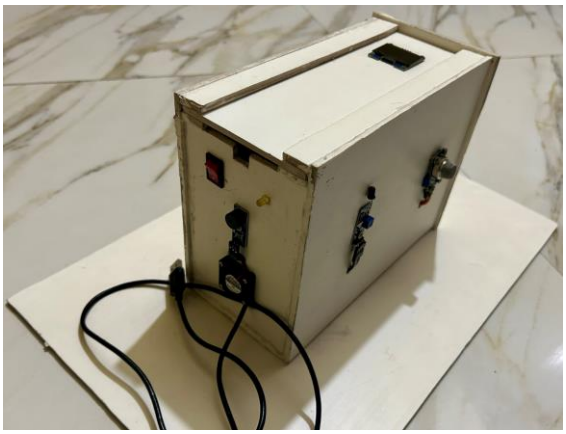

Fig. 16. External Circuit Implementation.

A. *Project Limitations and Hurtles*

In our project we used MQ-2, MQ-135 sensors to detect smoke and gas leakage respectively [12]. These two sensors are analog sensors; they are prone to errors and often give inaccurate data. They cannot even detect the exact type of smoke or gas. Furthermore, in this project neither smoke concentration nor gas concentration is shown. This is done to keep the project as simple and cost effective as possible. Hence, this project is by no means suitable for professional cooking places like bustling bistros or restaurants. This project is more suitable for use in homes.

Also, using ESP8266 Wi-Fi Module, instead of the much-advanced ESP32 Wi-Fi & Bluetooth Module, was a point of contention for some of us. ESP32 has twice the speed of a ESP8266 Wi-Fi Module but also costs thrice the price of just one ESP8266 Wi-Fi Module. So, even though ESP8266 is extremely cost efficient it may turn out to be a bottleneck while uploading code.

On a different note, other more established projects have used a PIR sensor (passive infra-red sensor) or an ultrasonic sensor to monitor the presence of people in kitchen [1]. However, we have opted not to use such sensors as we are disinterested in such a functionality. Whilst this decreases one functionality of the project. Its absence has minimal effect on our final goals for this project.

Lastly, I repeat, the sensors we used are neither quite accurate, nor reliable. The prime difficulty faced in this project was sensor-calibration. While the concept behind this project has a propitious prospect in the near future, the sensors used here could be the main bottleneck of this project. Hence, more expensive sensors like DHT22 and higher models of the MQ-series of sensors could be used to make this project more practical and robust for day-to-day, commercial use.

V. Conclusion

The main motive of this project was to set aside the cooking chores and set our sights on life. For this to happen automation has to be approachable, practical and meaningful. This project aims were to motive others to design their projects according to our design philosophy. Thus, we hope this project can pave a path for practical designs as the world of automation gleams over us.

References

[1] Nugroho, Fibry & Pantjawati, Arjuni. (2018). Automation and Monitoring Smart Kitchen Based on Internet of Things (IoT). IOP Conference Series: Materials Science and Engineering. 384. 012007. 10.1088/1757-899X/384/1/012007.

[2] J. Joseph, “Everything you need to know about the Arduino Hardware,” circuitdigest.com, Mar. 21, 2022. https://circuitdigest.com/article/everything-you-need-to-know-about-arduino-uno-board-hardware

[3] MACFOS, “IR sensor Working Principle and Applications | Robu.in,” Robu.in | Indian Online Store | RC Hobby | Robotics, May 19, 2020. https://robu.in/ir-sensor-working/

[4] N. kalaburgi, “Working of DHT sensor - DHT11 and DHT22,” NerdyElectronics, Jan. 27, 2021. https://nerdyelectronics.com/working-of-dht-sensor-dht11-and-dht22/

[5] “All About MQ Series Gas Sensor,” Robocraze. https://robocraze.com/blogs/post/mq-series-gas-sensor

[6] “MQ135 Alcohol Sensor Circuit and Its Working,” ElProCus - Electronic Projects for Engineering Students, May 04, 2016. https://www.elprocus.com/mq-135-alcohol-sensor-circuit-and-working/

[7] tonygo2, “Arduino and the SSD1306 OLED I2C 128x64 Display,” Instructables. https://www.instructables.com/Arduino-and-the-SSD1306-OLED-I2C-128x64-Display

[8] “Understanding Difference between Active and Passive Buzzer and How to use it with Arduino,” circuitdigest.com. https://circuitdigest.com/microcontroller-projects/understanding-difference-between-active-and-passive-buzzer-with-arduino

[9] M. J. Jayakumar, “Quick Start to Nodemcu (ESP8266) on Arduino IDE,” Instructables. https://www.instructables.com/Quick-Start-to-Nodemcu-ESP8266-on-Arduino-IDE/

[10] C. A. U. Hassan et al., “Design and Implementation of Real-Time Kitchen Monitoring and Automation System Based on Internet of Things,” Energies, vol. 15, no. 18, p. 6778, Sep. 2022, doi: https://doi.org/10.3390/en15186778.

[11] A. Parajuli, “ESP8266 Based Smart Kitchen Automation & Monitoring System,” IoT Projects Ideas, Feb. 26, 2024. https://iotprojectsideas.com/esp8266-based-smart-kitchen-automation-monitoring-system/ (accessed Jun. 17, 2024).

[12] More, Shubham & Shelar, Shridhar & Randhave, Vaibhav & Bagde, Prof. (2021). IoT Based Smart Kitchen System. International Journal of Scientific Research in Science, Engineering and Technology. 479-485. 10.32628/IJSRSET2183198.

APPENDIX

A. *Implemented Arduino Code (Arduino IDE)*

```
#include <U8glib.h>   // Library for U8glib OLED display.
#include <dht.h>      // Library for DHT11 Sensor.
#include <ArduinoJson.h>



const int Flame_Sensor_Pin = 3;    // Connects to Arduino pin ~3.
const int Smoke_Sensor_Pin = 6;    // Connects to Arduino pin ~6.
const int Gas_Sensor_Pin = 11;     // Connects to Arduino pin ~11.
const int Buzzer_Pin = 10;         // Connects to Arduino pin ~10.
const int Dht_Pin = 5;             // Connects to Arduino pin ~5.
const int Exhaust_Fan_Pin = 9;     // Connects to Arduino pin ~9.

// Threshold values for gas detection & humidity detection.
const int Humidity_Threshold = 60;

// Initialization for OLED display.
U8GLIB_SH1106_128X64 u8g(U8G_I2C_OPT_NONE); // OLED display object
dht DHT;

void setup() {
  Serial.begin(9600);               // Setting Baud rate
  pinMode(Flame_Sensor_Pin, INPUT);  // Takes input from Arduino pin A0.
  pinMode(Smoke_Sensor_Pin, INPUT);  // Takes input from Arduino pin ~6.
  pinMode(Gas_Sensor_Pin, INPUT);    // Takes input from Arduino pin A0.
  pinMode(Buzzer_Pin, OUTPUT);       // Takes output from Arduino pin ~10.
  pinMode(Exhaust_Fan_Pin, OUTPUT);  // Takes output from Arduino pin ~9.
}

void loop() {
  // Reads sensor values.
  int Flame_Value = digitalRead(Flame_Sensor_Pin);  // Reads analog value from Flame sensor.
  int Smoke_Value = digitalRead(Smoke_Sensor_Pin);  // Reads digital value from Smoke sensor.
  int Gas_Value = digitalRead(Gas_Sensor_Pin);      // Reads analog value from Gas sensor.
  DHT.read11(Dht_Pin);

 // Create JSON object
  StaticJsonDocument<200> doc;
  doc["Flame"] = (Flame_Value == LOW) ? "Detected" : "Not Detected";
  doc["Smoke"] = (Smoke_Value == LOW) ? "Detected" : "Not Detected";
```

```
    doc["Smoke"] = (Smoke_Value == LOW) ? "Detected" : "Not Detected";
    doc["Gas"] = (Gas_Value == LOW) ? "Detected" : "Not Detected";
    doc["Temperature"] = DHT.temperature;
    doc["Humidity"] = DHT.humidity;

    // Serialize JSON to a string
    String jsonString;
    serializeJson(doc, jsonString);

    // Send JSON string over serial
    Serial.println(jsonString);

    delay(1000); // Adjust delay as needed.


    // Update the display with the new sensor readings.
    displayData(Flame_Value, Gas_Value, Smoke_Value);
    // Handle alarms based on sensor readings.
    handleAlarms(Flame_Value, Smoke_Value, Gas_Value);
}
```

```
void displayData(int Flame_Value, int Gas_Value, int Smoke_Value) {
  u8g.firstPage();
  do {
    // Display flame status.
    u8g.setFont(u8g_font_unifont);
    u8g.setFont(u8g_font_6x10);
    u8g.drawStr(7, 15, "Flame: ");
    u8g.drawStr(50, 15, Flame_Value == LOW ? "Detected" : "Not Detected");

    // Display gas status.
    u8g.drawStr(7, 45, "Gas: ");
    u8g.drawStr(50, 45, Gas_Value == LOW ? "Detected" : "Not Detected");

    // Display smoke status.
    u8g.drawStr(7, 30, "Smoke: ");
```

```
    u8g.drawStr(50, 60, "C");

    // Display humidity (H).
    u8g.drawStr(70, 60, "H: ");
    u8g.drawStr(85, 60, String(DHT.humidity, 1).c_str());
    u8g.drawStr(110, 60, "%");
  } while (u8g.nextPage());
}

void handleAlarms(int Flame_Value, int Smoke_Value, int Gas_Value) {
  // Handle gas alarm first since it has priority.
  if (Gas_Value == LOW) {
    tone(Buzzer_Pin, 2280);
    digitalWrite(Exhaust_Fan_Pin, HIGH); // Turn on exhaust fan if gas is detected.
  }
  // Handle other alarms.
  else if (Flame_Value == LOW) {
    tone(Buzzer_Pin, 1520);
  }
  else if (Smoke_Value == LOW) {
```

```
  // Handle gas alarm first since it has priority.
  if (Gas_Value == LOW) {
    tone(Buzzer_Pin, 2280);
    digitalWrite(Exhaust_Fan_Pin, HIGH); // Turn on exhaust fan if gas is detected.
  }
  // Handle other alarms.
  else if (Flame_Value == LOW) {
    tone(Buzzer_Pin, 1520);
  }
  else if (Smoke_Value == LOW) {
    tone(Buzzer_Pin, 760);
  }
  else {
    noTone(Buzzer_Pin); // Turn off buzzer if no alarms are triggered.
    digitalWrite(Exhaust_Fan_Pin, LOW); // Turn off exhaust fan.
  }

  // Turn on exhaust fan if gas is detected.
  digitalWrite(Exhaust_Fan_Pin, DHT.humidity > Humidity_Threshold ? HIGH : LOW);
}
```

B. *Implemented NodeMCU Wi-Fi Code (Arduino IDE)*

```

#define BLYNK_PRINT Serial
#define BLYNK_TEMPLATE_ID "TMPL6wCPp3lQi"
#define BLYNK_TEMPLATE_NAME "Smart Kitchen and Alarm System"
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include <BlynkSimpleEsp8266.h>

char auth[] = "4ndjb6z60OXw--cHT-XD8n677hMXyomb";  // Blynk authentication token
char ssid[] = "iPhone"; // WiFi network SSID
char pass[] = "Lead-Steel-BG9"; // WiFi network password
// Virtual pin numbers
#define FLAME_PIN V0
#define SMOKE_PIN V1
#define GAS_PIN V2
#define TEMP_PIN V3
#define HUMIDITY_PIN V4
void setup() {
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass); // Connect to Blynk server
```

```
  Blynk.begin(auth, ssid, pass); // Connect to Blynk server
  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  Blynk.run();
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    parseSensorData(data); // Parse and process sensor data
  }
}
void parseSensorData(String data) {
```

```
  }
}
void parseSensorData(String data) {
  Serial.println("Received data: " + data); // Print received data
  // Parse JSON data
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, data);
  if (error) {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
    return;
  }
  // Update Blynk widgets with sensor data
  Blynk.virtualWrite(FLAME_PIN, doc["Flame"].as<String>());
  Blynk.virtualWrite(SMOKE_PIN, doc["Smoke"].as<String>());
  Blynk.virtualWrite(GAS_PIN, doc["Gas"].as<String>());
  Blynk.virtualWrite(TEMP_PIN, doc["Temperature"].as<float>());
  Blynk.virtualWrite(HUMIDITY_PIN, doc["Humidity"].as<float>());
}

```