ระบบตรวจจับและแจ้งเตือนการรั่วไหลของก๊าซติดไฟ

วัตถุประสงค์ของโครงการ

เพื่อสร้างระบบตรวจจับและแจ้งเตือนการรั่วไหลของก๊าซติดไฟอย่างรวดเร็วและมีประสิทธิภาพ โดยใช้เซ็นเซอร์ตรวจจับแก๊ส MQ-2 เพื่อป้องกันอันตรายที่อาจเกิดจากก๊าซรั่วไหล ระบบจะสามารถแจ้งเตือนผ่านทางเสียงด้วย Buzzer และแจ้งเตือนผ่านทางแอปพลิเคชัน LINE Notify เมื่อก๊าซที่ตรวจจับได้เกินระดับที่กำหนด อีกทั้งยังบันทึกข้อมูลการตรวจจับเพื่อการวิเคราะห์ในอนาคตโดยใช้ SD Card และแสดงข้อมูลที่เกี่ยวข้องผ่านจอ OLED

การทำงานของระบบ

การทำงานของระบบ ได้แก่ มีการใช้เซ็นเซอร์ตรวจจับควันและก๊าซที่มีสถานะเป็นเชื้อเพลิงในการติดไฟ จากนั้นจะทำการแจ้งเตือนด้วยระบบเสียง และส่งแจ้งเตือนไปอีกทีทางไลน์ พร้อมทั้งแสดงผลค่าก๊าซติดไฟบน OLED Display

ฟังก์ชันของระบบ

1. การแจ้งผ่านทางไลน์ด้วย LINE Notify และ แจ้งเตือนผ่านทางเสียงด้วย Buzzer เมื่อเซ็นเซอร์ตรวจพบก๊าซเกินระดับที่กำหนด
2. จอ OLED แสดงค่าความหนาแน่นก๊าซ,อุณหภูมิ, วันที่และเวลา
3. ใช้ LED เพื่อแสดงสถานะ เมื่อเซ็นเซอร์ตรวจพบก๊าซเกินระดับที่กำหนด
4. ใช้ SD Card เก็บข้อมูลประวัติการตรวจสอบก๊าซเพื่อการวิเคราะห์ และใช้ข้อมูลที่บันทึกเพื่อตรวจสอบเหตุการณ์ที่ผ่านมาและปรับปรุงการตอบสนองในอนาคต

อุปกรณ์ที่ใช้

1. Board Node32s: ใช้ในการควบคุมหลักในระบบ
2. DS18B20: เป็นเซ็นเซอร์ใช้ในการวัดอุณหภูมิ
3. Resistor ขนาด 4.7k ohm ¼ Watt: ต่อระหว่างสายข้อมูล (Data Line) กับแหล่งจ่ายไฟ (VCC) เพื่อดึงแรงดันขึ้น
4. SD Card Reader Module: ใช้สำหรับอ่านและเขียนข้อมูลลงใน SD Card
5. SD Card: สำหรับในการบันทึกข้อมูลของก๊าซ
6. ชุดสาย Jumper: ใช้เชื่อมต่ออุปกรณ์ต่างๆ
7. Protoboard: เป็นอุปกรณ์ที่ใช้ในการสร้างวงจรอิเล็กทรอนิกส์ชั่วคราวโดยไม่ต้องบัดกรีสายไฟ
8. Sensor MQ-2: เซ็นเซอร์ใช้ตรวจจับก๊าซไวไฟ
9. Buzzer: ใช้สำหรับส่งสัญญาณเสียงเตือนเมื่อมีการตรวจจับก๊าซติดไฟ
10. LED: ใช้แจ้งสถานะ เมื่อมีการตรวจจับก๊าซติดไฟ
11. OLED Display: จอแสดงผลข้อมูลของก๊าสที่ตรวจจับ
12. Real-Time Clock Module: เพื่อบันทึกและติดตามเวลาแนบลงในการส่งแจ้งเตือน
13. Push button switch module: สำหรับตรวจจับการกดปุ่มและเปลี่ยนความถี่ในส่งข้อมูลผ่านการแจ้งเตือน

ตารางการเชื่อมต่ออุปกรณ์กับ Pin ของ Board NodeMCU32s

อุปกรณ์ภายนอก	Pin ของ Board NodeMCU32s
1.เซ็นเซอร์ DS18B20	VCC ->3V3 GND -> GND Data Pin -> GPIO16 Resistor 4.7k ohm -> ต่อระหว่าง Data Pin และVCC
2.การต่อ LED สีเขียว	ANODE(+) -> GPIO26 Cathode(-) -> Resistor 1.1k ohm -> GND

3.การต่อ LED สีแดง	ANODE(+) -> GPIO27 Cathode(-) -> Resistor 1.1k ohm -> GND
4.การต่อ OLED Display (I2C)	VCC -> 3V3 GND -> GND SDA -> GPIO21 SCK -> GPIO22
5.การต่อ Real-Time Clock Module (RTC)	VCC -> 3V3 GND -> GND SDA -> GPIO21 SCL -> GPIO22
6.เซ็นเซอร์ MQ-2	VCC -> 5V AO -> GPIO36 D0 -> GPIO32 GND -> GND Resistor 1.1k ohm -> GND
7.SD Card Reader Module	VCC -> 5V MOSI -> GPIO23 MISO -> GPIO19 SCK -> GPIO18 CS -> GPIO5 GND -> GND
8.การต่อ Buzzer	VCC -> 3V3 I/O -> GPIO25 GND -> GND
9.การต่อ Switch Module	VCC -> 3V3 GND -> GND OUTPUT -> GPIO23

โปรแกรมการทำงาน

C/C++

```
#include <Wire.h>              // ไลบรารีสำหรับการสื่อสาร I2C
#include <Adafruit_GFX.h>      // ไลบรารีสำหรับการจัดการกราฟิกบนจอแสดงผล
#include <Adafruit_SH110X.h>  // ไลบรารีสำหรับจอ OLED ชนิด SH1106/SH1107
#include <OneWire.h>           // ไลบรารีสำหรับการสื่อสารแบบ OneWire ใช้ใน Lab 9
#include <DallasTemperature.h>  // ไลบรารีสำหรับการใช้งานเซ็นเซอร์วัดอุณหภูมิ DS18B20
#include <ESP32TimerInterrupt.h>  // ไลบรารีสำหรับการใช้งานการขัดจังหวะ interrupt บน ESP32
#include <SD.h>   // ไลบรารีสำหรับการจัดการ SD card
#include <SPI.h>  // ไลบรารีสำหรับการสื่อสาร SPI จำเป็นสำหรับ SD card
#include <TridentTD_LineNotify.h>  // ไลบรารีสำหรับการแจ้งเตือนผ่าน LINE Notify

// กำหนดค่าชื่อ WIFI SSID และรหัสผ่าน PASSWORD เพื่อเชื่อมต่อกับเครือข่าย WiFi
#define SSID "17nim"
#define PASSWORD "77499981"
// กำหนดค่าคีย์ LINE Notify Token สำหรับส่งข้อความแจ้งเตือน
#define LINE_TOKEN "4EhUI3x7LEYvZFZJa1EcOoZTWuUoq4r7dmDq2BiPJxm"
// กำหนดหมายเลขพินที่เชื่อมต่อกับ LED บนบอร์ด LED สีฟ้า ซึ่งอยู่ที่ GPIO2
#define LED_PIN 2  // On board LED --> GPIO2
// กำหนดหมายเลขพินสำหรับการสื่อสาร I2C เชื่อมต่อ OLED และ DS3231 RTC
#define I2C_SDA 21  // OLED and DS3231: SDA --> GPIO21
#define I2C_SCL 22  // OLED and DS3231: SCL --> GPIO22
// กำหนดค่าเกณฑ์สำหรับอุณหภูมิ TEMP_THRESHOLD เมื่อเกิน 60 องศาเซลเซียส จะเกิดการแจ้งเตือน
#define TEMP_THRESHOLD 60
```

```
// กำหนดค่าแรงดันไฟฟ้าเกณฑ์สำหรับเซ็นเซอร์แก๊ส GAS_VOLTAGE_THRESHOLD เมื่อแรงดันเกิน 3000 mV จะมีการแจ้งเตือน
#define GAS_VOLTAGE_THRESHOLD 3000
// กำหนดหมายเลขพินสำหรับ Buzzer และไฟ LED สีเขียวกับสีแดง
#define BUZZER_PIN 25  // เชื่อมต่อ Buzzer ที่ GPIO25
#define GREEN_LED 26   // เชื่อมต่อไฟ LED สีเขียวที่ GPIO26
#define RED_LED 27     // เชื่อมต่อไฟ LED สีแดงที่ GPIO27
// กำหนดหมายเลขพินสำหรับเซ็นเซอร์แก๊ส ทั้งแบบอนาล็อกและดิจิทัล
#define GAS_ANALOG 36   // อ่านค่าแรงดันจากเซ็นเซอร์แก๊สแบบอนาล็อกที่ GPIO36
#define GAS_DIGITAL 32  // อ่านค่าจากเซ็นเซอร์แก๊สแบบดิจิทัลที่ GPIO32
// กำหนดพินสำหรับการใช้งาน SD card
#define CS 5       // Chip Select ที่ GPIO5
#define SD_SCK 18  // Clock SCK ที่ GPIO18
#define MISO 19    // Master In Slave Out (MISO) ที่ GPIO19
#define MOSI 23    // Master Out Slave In (MOSI) ที่ GPIO23

SPIClass spi = SPIClass(VSPI);  // กำหนดการใช้งาน SPI โดยใช้ VSPI บน ESP32

// กำหนดหมายเลขพินสำหรับการใช้งาน OneWire ใช้ในการเชื่อมต่อเซ็นเซอร์วัดอุณหภูมิ DS18B20
#define ONE_WIRE_BUS 16  // เชื่อมต่อเซ็นเซอร์อุณหภูมิที่ GPIO16

// กำหนดหมายเลขพินสำหรับสวิตช์ SW_PIN ซึ่งเชื่อมต่อที่ GPIO23
#define SW_PIN 23

// ประกาศตัวแปรชนิด string array of characters สำหรับเก็บข้อมูลเวลา ต้องกำหนดขนาดเพียงพอที่จะเก็บข้อมูลทั้งหมด
char timetext[100];

// ตัวแปรแบบ boolean สำหรับใช้ในการสลับสถานะ LED
bool LEDtoggle = false;

// กำหนดที่อยู่ I2C ของจอ OLED และขนาดของหน้าจอ OLED
#define OLED_I2C_ADDRESS 0x3c  // ที่อยู่ I2C ของจอ OLED
#define SCREEN_WIDTH 128       // ความกว้างของหน้าจอ OLED เป็นพิกเซล
#define SCREEN_HEIGHT 64       // ความสูงของหน้าจอ OLED เป็นพิกเซล
#define OLED_RESET -1          // กำหนดการรีเซ็ตของ OLED -1 หมายถึงไม่ใช้พินรีเซ็ต

// ประกาศตัวแปร `display` สำหรับการใช้งานจอ OLED โดยใช้ไลบรารี Adafruit_SH1106G
Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// กำหนดที่อยู่ I2C ของโมดูล DS3231 สำหรับการใช้งานนาฬิกาจริง RTC
#define DS3231_I2C_ADDRESS 0x68  // ที่อยู่ I2C ของ DS3231

// สร้างตัวแปร `ITimer0` สำหรับใช้งานการขัดจังหวะโดยใช้ไลบรารี ESP32Timer ใช้ใน Lab 9
ESP32Timer ITimer0(0);  // สร้างการขัดจังหวะในตัวจับเวลาที่ 0

// การใช้งาน OneWire สำหรับการเชื่อมต่อกับเซ็นเซอร์วัดอุณหภูมิ DS18B20 ที่พิน ONE_WIRE_BUS GPIO16
OneWire oneWire(ONE_WIRE_BUS);

// ใช้ไลบรารี DallasTemperature เพื่อจัดการกับเซ็นเซอร์ DS18B20 ที่เชื่อมต่อกับ oneWire
DallasTemperature sensors(&oneWire);

// ตัวแปร `tempC` ใช้เก็บค่าอุณหภูมิที่อ่านได้จากเซ็นเซอร์ DS18B20
float tempC;

bool FileOpenFlag = false;     // ตัวแปร flag เพื่อบ่งชี้ว่าไฟล์ถูกเปิดหรือไม่
File myFile;                   // ตัวแปรสำหรับจัดการไฟล์ ใช้สำหรับการเขียนหรืออ่านข้อมูลจาก SD card
unsigned long n;               // ตัวแปรสำหรับเก็บค่าเริ่มต้นของ n
unsigned long nsample = 1000;  // กำหนดจำนวนตัวอย่างที่ต้องการ เช่น การเก็บข้อมูล 1000 ตัวอย่าง

// กำหนดค่าแรงดันไฟฟ้าของแหล่งจ่ายไฟ Vcc 5V
float Vcc = 5;

// กำหนดตัวแปรสถานะ LED และสวิตช์ให้เป็น `static` เพื่อคงค่าระหว่างการเรียกฟังก์ชันต่าง ๆ
static bool LED_toggle = false;  // กำหนดสถานะ LED เริ่มต้นเป็นปิด false --> LED ปิด
static bool SW_Flag = false;     // ตัวแปร flag สำหรับการควบคุมการสุ่มตัวอย่างสวิตช์ sampling frequency
static bool ReadFlag = false;    // ตัวแปร flag สำหรับบ่งบอกสถานะการอ่านข้อมูล

// กำหนดช่วงเวลาในการทำงานของตัวจับเวลา timer interval เป็น 1000 มิลลิวินาที 1 วินาที
```

```
unsigned int timer_interval = 1000;  // ตัวจับเวลาในหน่วยมิลลิวินาที 1000ms = 1s

// กำหนดตัวแปรเวลา เพื่อเก็บค่าเวลาที่ผ่านไปตั้งแต่ครั้งก่อนหน้าและเวลาปัจจุบัน
unsigned long previousMillis = 0;  // เก็บค่าเวลาในช่วงก่อนหน้า
unsigned long currentMillis;       // เก็บค่าเวลาปัจจุบัน

// ตัวแปรที่ใช้สำหรับการคำนวณค่าแรงดันและความต้านทานของเซ็นเซอร์แก๊ส
float gasAnalog, Vout, Rs, ratio, R0, RL;  // gasAnalog: ค่าที่อ่านจากเซ็นเซอร์แก๊ส, Vout: แรงดันเอาต์พุต, Rs: ความต้านทานเซ็นเซอร์, ratio: อัตราส่วน Rs/R0, R0: ความต้านทานพื้นฐาน, RL: ความต้านทานโหลด

// ค่าความเข้มข้นของแก๊สชนิดต่าง ๆ ที่เซ็นเซอร์วัดได้ เช่น แก๊สมีเทน (CH4), LPG, คาร์บอนมอนอกไซด์ (CO), ไฮโดรเจน (H2), และแอลกอฮอล์
float ch4, lpg, co, h2, alcohol;

// ฟังก์ชันสำหรับคำนวณค่าความเข้มข้นของแก๊ส CH4 (มีเทน)
// รับพารามิเตอร์ ratio ซึ่งเป็นอัตราส่วนของ Rs/R0 และคำนวณโดยใช้สูตรยกกำลัง
// pow(ratio, -1.67) หมายถึงการยกกำลัง ratio ด้วย -1.67 แล้วคูณด้วย 10 เพื่อให้ได้ค่าความเข้มข้นของแก๊ส CH4
float calculateCH4(float ratio) {
  return pow(ratio, -1.67) * 10.0;
}

// ฟังก์ชันสำหรับคำนวณค่าความเข้มข้นของแก๊ส LPG
// ใช้สูตร pow(ratio, -2.0) ซึ่งยกกำลัง ratio ด้วย -2.0 แล้วคูณด้วย 20 เพื่อให้ได้ค่าความเข้มข้นของแก๊ส LPG
float calculateLPG(float ratio) {
  return pow(ratio, -2.0) * 20.0;
}

// ฟังก์ชันสำหรับคำนวณค่าความเข้มข้นของคาร์บอนมอนอกไซด์ (CO)
// pow(ratio, -1.70) หมายถึงการยกกำลัง ratio ด้วย -1.70 แล้วคูณด้วย 15 เพื่อให้ได้ค่าความเข้มข้นของ CO
float calculateCO(float ratio) {
  return pow(ratio, -1.70) * 15.0;
}

// ฟังก์ชันสำหรับคำนวณค่าความเข้มข้นของไฮโดรเจน (H2)
// ใช้สูตร pow(ratio, -1.41) ซึ่งยกกำลัง ratio ด้วย -1.41 แล้วคูณด้วย 18 เพื่อให้ได้ค่าความเข้มข้นของ H2
float calculateH2(float ratio) {
  return pow(ratio, -1.41) * 18.0;
}

// ฟังก์ชันสำหรับคำนวณค่าความเข้มข้นของแอลกอฮอล์
// ใช้สูตร pow(ratio, -1.45) ซึ่งยกกำลัง ratio ด้วย -1.45 แล้วคูณด้วย 25 เพื่อให้ได้ค่าความเข้มข้นของแอลกอฮอล์
float calculateAlcohol(float ratio) {
  return pow(ratio, -1.45) * 25.0;
}

// ฟังก์ชัน Interrupt Service Routine สำหรับตัวจับเวลา Timer บน ESP32
// ฟังก์ชันนี้จะถูกเรียกทุกครั้งเมื่อถึงเวลาที่กำหนดใน timer_interval
// เมื่อถูกเรียก ฟังก์ชันจะเปลี่ยนค่า ReadFlag เป็น true เพื่อบ่งบอกว่าควรอ่านข้อมูล
// IRAM_ATTR ระบุว่าเป็นฟังก์ชันที่สามารถรันได้ใน Internal RAM จำเป็นสำหรับ ISR บน ESP32
bool IRAM_ATTR TimerHandler(void *timerNo) {
  ReadFlag = true;  // ตั้งค่าสถานะ ReadFlag เพื่อบอกให้ระบบอ่านข้อมูล
  return true;      // ส่งค่ากลับเป็น true เพื่อบอกว่า ISR ทำงานสำเร็จ
}

// ISR สำหรับการกดสวิตช์ภายนอก
// ฟังก์ชันนี้จะถูกเรียกเมื่อมีการกดสวิตช์
// ใช้ IRAM_ATTR เพื่อให้ฟังก์ชันสามารถรันได้ใน Internal RAM จำเป็นสำหรับ ISR บน ESP32
void IRAM_ATTR SW_Press() {
  SW_Flag = true;  // ตั้งค่าสถานะ SW_Flag เป็น true เพื่อบ่งบอกว่ามีการกดสวิตช์
}

// ฟังก์ชันเพื่อแปลงเลขฐานสิบธรรมดาเป็นเลขฐานสองที่เข้ารหัส BCD
// รับพารามิเตอร์ val ซึ่งเป็นเลขฐานสิบ และทำการแปลง
// การคำนวณนี้ใช้การหารและการหารเอาเศษเพื่อนำมาแปลงเป็น BCD
byte decToBcd(byte val) {
  return ((val / 10 * 16) + (val % 10));  // แปลง val เป็น BCD
}

// ฟังก์ชันเพื่อแปลงเลขฐานสองที่เข้ารหัส (BCD) กลับเป็นเลขฐานสิบ
```

```
// รับพารามิเตอร์ val ซึ่งเป็นเลข BCD และทำการแปลง
// การคำนวณนี้ใช้การหารและการหารเอาเศษเพื่อนำมาแปลงเป็นเลขฐานสิบธรรมดา
byte bcdToDec(byte val) {
  return ((val / 16 * 10) + (val % 16));  // แปลง val จาก BCD เป็นเลขฐานสิบ
}

// ฟังก์ชันเพื่อกำหนดเวลาและวันที่ให้กับ DS3231
// รับพารามิเตอร์ ได้แก่ วินาที นาที ชั่วโมง วันในสัปดาห์ วันของเดือน เดือน และปี
void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte month, byte year) {
  Wire.beginTransmission(DS3231_I2C_ADDRESS);  // เริ่มการส่งข้อมูลไปยัง DS3231
  Wire.write(0);                               // กำหนดให้ข้อมูลถัดไปเริ่มต้นที่รีจิสเตอร์วินาที
  Wire.write(decToBcd(second));                // กำหนดวินาที
  Wire.write(decToBcd(minute));                // กำหนดนาที
  Wire.write(decToBcd(hour));                  // กำหนดชั่วโมง
  Wire.write(decToBcd(dayOfWeek));             // กำหนดวันในสัปดาห์ (1=อาทิตย์, 7=เสาร์)
  Wire.write(decToBcd(dayOfMonth));            // กำหนดวันที่ (1 ถึง 31)
  Wire.write(decToBcd(month));                 // กำหนดเดือน
  Wire.write(decToBcd(year));                  // กำหนดปี (0 ถึง 99)
  Wire.endTransmission();                      // ส่งข้อมูลและปิดการเชื่อมต่อ
}

// ฟังก์ชันอ่านเวลาและวันที่จาก DS3231
// รับพารามิเตอร์เป็นตัวชี้ไปยังตัวแปรเพื่อเก็บค่าเวลาที่อ่านมา
void readDS3231time(byte *second, byte *minute, byte *hour, byte *dayOfWeek, byte *dayOfMonth, byte *month, byte *year) {
  Wire.beginTransmission(DS3231_I2C_ADDRESS);  // เริ่มการส่งข้อมูลไปยัง DS3231
  Wire.write(0);                               // ตั้งค่าชี้ไปยังรีจิสเตอร์ที่ 00h วินาที
  Wire.endTransmission();                      // ปิดการเชื่อมต่อ

  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);  // ขอข้อมูลเจ็ดไบต์จาก DS3231 เริ่มจากรีจิสเตอร์ 00h
  // อ่านข้อมูลและแปลงจาก BCD เป็นเลขฐานสิบ
  *second = bcdToDec(Wire.read() & 0x7f);  // อ่านวินาที
  *minute = bcdToDec(Wire.read());         // อ่านนาที
  *hour = bcdToDec(Wire.read() & 0x3f);    // อ่านชั่วโมง
  *dayOfWeek = bcdToDec(Wire.read());      // อ่านวันในสัปดาห์
  *dayOfMonth = bcdToDec(Wire.read());     // อ่านวันที่
  *month = bcdToDec(Wire.read());          // อ่านเดือน
  *year = bcdToDec(Wire.read());           // อ่านปี
}

// ฟังก์ชันแปลงเวลาให้เป็นข้อความ
// รับพารามิเตอร์เป็นตัวชี้ไปยังสตริงเพื่อเก็บค่าที่แปลงแล้ว
void TimetoText(char *p) {
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;                    // ตัวแปรสำหรับเก็บค่าที่อ่านจาก DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);  // เรียกฟังก์ชันอ่านเวลา
  // ใช้ sprintf เพื่อจัดรูปแบบข้อความเป็น DD/MM/YY HH:MM:SS
  sprintf(p, "%02d/%02d/%02d %02d:%02d:%02d", dayOfMonth, month, year, hour, minute, second);
}

void setup() {
  n = 0;  // เริ่มต้นค่าตัวแปร n

  // ตั้งค่าพินสำหรับ Buzzer, LEDs และสวิตช์
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(SW_PIN, INPUT_PULLUP);  // ใช้ตัวต้านทานดึงภายในสำหรับสวิตช์

  // ตั้งค่าพินสำหรับการวัดก๊าซ
  pinMode(GAS_ANALOG, INPUT);
  pinMode(GAS_DIGITAL, OUTPUT);
  R0 = 10;  // ค่าความต้านทาน R0 เริ่มต้น
  RL = 10;  // ค่าความต้านทาน RL เริ่มต้น

  // เชื่อมต่อ Wi-Fi
  WiFi.begin(SSID, PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
```

```
  Serial.print(".");  // แสดงจุดระหว่างรอการเชื่อมต่อ
  delay(400);         // รอสั้นๆ ก่อนตรวจสอบสถานะอีกครั้ง
}

// ตั้งค่า LINE Notify
LINE.setToken(LINE_TOKEN);
LINE.notify("Connected!");  // แจ้งเตือนเมื่อเชื่อมต่อสำเร็จ

Serial.begin(115200);               // เริ่มการสื่อสารผ่าน Serial ที่ baud rate 115200
while (!Serial && millis() < 1000)  // รอให้ Serial เชื่อมต่อ
  ;
delay(1000);
Serial.println("Start OLED Display...");  // ข้อความเริ่มต้นสำหรับ OLED

// เริ่มต้นการสื่อสาร I2C
Wire.begin(I2C_SDA, I2C_SCL);

// เริ่มต้น OLED Display
if (!display.begin(OLED_I2C_ADDRESS, true)) {
  Serial.println("OLED allocation failed");  // แจ้งเตือนหากการเริ่มต้นล้มเหลว
  for (;;)
    ;  // หยุดการทำงานในลูปตลอดไป
}
display.setContrast(0);             // ปรับความเข้มของจอแสดงผล
display.clearDisplay();             // ล้างจอแสดงผล
display.setTextSize(0.5);           // ขนาดตัวอักษร
display.setTextColor(SH110X_WHITE);  // สีตัวอักษร
display.setCursor(0, 0);            // ตั้งตำแหน่งเคอร์เซอร์
display.println("Initializing...");  // แสดงข้อความการเริ่มต้น
display.display();                  // อัปเดตจอแสดงผล

Serial.println("Start DS3231 Realtime Clock...");  // ข้อความเริ่มต้นสำหรับ DS3231
setDS3231time(0, 0, 0, 1, 1, 1, 24);              // ตั้งเวลาเริ่มต้น
sensors.begin();  // เริ่มต้นเซ็นเซอร์วัดอุณหภูมิ

delay(250);  // รอสั้นๆ

display.clearDisplay();                  // ล้างจอแสดงผลอีกครั้ง
display.setTextSize(1);                  // ขนาดตัวอักษรเพิ่มขึ้น
display.setTextColor(SH110X_WHITE);      // สีตัวอักษร
display.setCursor(0, 0);                 // ตั้งตำแหน่งเคอร์เซอร์
display.println("MPI09 ESP32 Interrupt");  // แสดงข้อความการทดสอบ
display.display();                       // อัปเดตจอแสดงผล

// แสดงข้อมูลเริ่มต้นเกี่ยวกับการทดสอบ Timer Interrupt
Serial.print(F("\nStarting TimerInterruptTest on "));
Serial.println(ARDUINO_BOARD);
Serial.println(ESP32_TIMER_INTERRUPT_VERSION);
Serial.print(F("CPU Frequency = "));
Serial.print(F_CPU / 1000000);
Serial.println(F(" MHz"));

// เริ่มต้น SD card
if (!SD.begin(CS)) {
  Serial.println("Initialization failed!");  // แจ้งเตือนหากการเริ่มต้นล้มเหลว
  return;                                    // ออกจากฟังก์ชัน
}
spi.begin(SD_SCK, MISO, MOSI, CS);          // เริ่มต้น SPI สำหรับ SD card
myFile = SD.open("/MPI10.txt", FILE_WRITE);  // เปิดไฟล์เพื่อเขียน
if (myFile) {
  Serial.println("File is opened");      // แจ้งเตือนเมื่อเปิดไฟล์สำเร็จ
  FileOpenFlag = true;                   // ตั้งค่าธงว่าไฟล์เปิดอยู่
  myFile.println("Temp(C) & Gas(ppm)");  // เขียนข้อความหัวตารางในไฟล์
} else {
  Serial.println("Error open file");  // แจ้งเตือนหากเกิดข้อผิดพลาดในการเปิดไฟล์
}
```

```
  attachInterrupt(digitalPinToInterrupt(SW_PIN), SW_Press, FALLING);  // ตั้งค่า Interrupt สำหรับสวิตช์

  // ตั้งค่า Timer Interrupt
  if (ITimer0.attachInterruptInterval(timer_interval * 1000, TimerHandler)) {
    Serial.print(F("Starting ITimer0 OK, millis() = "));
    Serial.println(millis());
  } else {
    Serial.println(F("Can't set ITimer0. Select another freq. or timer"));  // แจ้งเตือนหากตั้งค่าไม่สำเร็จ
  }
  Serial.flush();  // ล้างข้อมูลใน Serial buffer
}
void loop() {
  currentMillis = millis();  // บันทึกเวลาในหน่วยมิลลิวินาทีตั้งแต่เริ่มโปรแกรม

  // อ่านค่าจากเซ็นเซอร์
  TimetoText(timetext);                 // ดึงเวลาปัจจุบันในรูปแบบข้อความและเก็บไว้ใน timetext
  sensors.requestTemperatures();        // ส่งคำขออ่านอุณหภูมิจากเซ็นเซอร์
  tempC = sensors.getTempCByIndex(0);  // อ่านค่าอุณหภูมิในเซลเซียสจากเซ็นเซอร์แรก
  gasAnalog = analogRead(GAS_ANALOG);  // อ่านค่าจากเซ็นเซอร์แก๊สในรูปแบบอนาล็อก

  // แปลงแรงดันไฟฟ้าที่อ่านได้เป็นความเข้มข้น
  Vout = gasAnalog * (Vcc / 4095.0);  // คำนวณแรงดันไฟฟ้าตามค่าที่อ่านได้จากเซ็นเซอร์
  Rs = ((Vcc - Vout) * RL) / Vout;    // คำนวณค่า Rs จากแรงดันไฟฟ้า
  ratio = Rs / R0;                    // คำนวณอัตราส่วนระหว่าง Rs และ R0

  // คำนวณความเข้มข้นของแก๊สแต่ละชนิด
  ch4 = calculateCH4(ratio);          // คำนวณความเข้มข้นของ CH4
  lpg = calculateLPG(ratio);          // คำนวณความเข้มข้นของ LPG
  co = calculateCO(ratio);            // คำนวณความเข้มข้นของ CO
  h2 = calculateH2(ratio);            // คำนวณความเข้มข้นของ H2
  alcohol = calculateAlcohol(ratio);  // คำนวณความเข้มข้นของแอลกอฮอล์

  // แสดงข้อมูลบน Serial Monitor
  Serial.println(timetext);  // แสดงเวลาปัจจุบันใน Serial Monitor

  Serial.print("Temp = ");  // แสดงข้อความ "Temp = "
  Serial.print(tempC);      // แสดงค่าอุณหภูมิ
  Serial.println(" C");     // แสดงหน่วย "C"

  Serial.print("Gas = ");   // แสดงข้อความ "Gas = "
  Serial.print(gasAnalog);  // แสดงค่าที่อ่านจากเซ็นเซอร์แก๊ส
  Serial.println(" ppm");   // แสดงหน่วย "ppm"

  // เตรียมเนื้อหาสำหรับแสดงผล
  display.clearDisplay();               // ล้างหน้าจอ OLED เพื่อเริ่มต้นใหม่
  display.setTextSize(1);               // กำหนดขนาดข้อความเป็น 1
  display.setTextColor(SH110X_WHITE);  // กำหนดสีข้อความเป็นขาว
  display.setCursor(0, 0);              // กำหนดตำแหน่งเคอร์เซอร์ที่มุมบนซ้าย

  // แสดงเวลาปัจจุบัน
  display.println(timetext);  // แสดงเวลาปัจจุบันในรูปแบบข้อความ

  // แสดงค่าอุณหภูมิ
  display.print("Temp = ");  // แสดงข้อความ "Temp = "
  display.print(tempC);      // แสดงค่าอุณหภูมิ
  display.println(" C");     // แสดงหน่วย "C"

  // แสดงค่าการอ่านจากเซ็นเซอร์แก๊ส
  display.print("Gas = ");     // แสดงข้อความ "Gas = "
  display.println(gasAnalog);  // แสดงค่าที่อ่านจากเซ็นเซอร์แก๊ส

  // แสดงความเข้มข้นของแก๊สแต่ละชนิด
  display.print("CH4: ");   // แสดงข้อความ "CH4: "
  display.print(ch4);       // แสดงความเข้มข้นของ CH4
  display.println(" ppm");  // แสดงหน่วย "ppm"

  display.print("LPG: ");   // แสดงข้อความ "LPG: "
```

```
display.print(lpg);       // แสดงความเข้มข้นของ LPG
display.println(" ppm");  // แสดงหน่วย "ppm"

display.print("CO: ");    // แสดงข้อความ "CO: "
display.print(co);        // แสดงความเข้มข้นของ CO
display.println(" ppm");  // แสดงหน่วย "ppm"

display.print("H2: ");    // แสดงข้อความ "H2: "
display.print(h2);        // แสดงความเข้มข้นของ H2
display.println(" ppm");  // แสดงหน่วย "ppm"

display.print("Alcohol: "); // แสดงข้อความ "Alcohol: "
display.print(alcohol);     // แสดงความเข้มข้นของแอลกอฮอล์
display.println(" ppm");    // แสดงหน่วย "ppm"
display.display();  // อัปเดตหน้าจอ OLED เพื่อแสดงข้อมูลทั้งหมดที่เตรียมไว้ในรอบนี้

// สลับ LED เพื่อแสดงสถานะการอ่านอุณหภูมิ
digitalWrite(LED_PIN, LED_toggle);  // เขียนค่าลงไปที่ LED_PIN ตามสถานะของ LED_toggle
LED_toggle = !LED_toggle;           // เปลี่ยนสถานะของ LED_toggle

// ถ้า SW_Flag ถูกตั้งค่าให้เปลี่ยนอัตราการสุ่มตัวอย่าง
if (SW_Flag) {
  ITimer0.disableTimer();     // ปิด Timer0
  ITimer0.detachInterrupt();  // แยกการเชื่อมต่อ Interrupt

  // สลับระยะเวลาของ Timer ระหว่าง 1000ms และ 250ms
  if (timer_interval == 1000) {
    timer_interval = 5000;  // เปลี่ยนเป็น 5000ms
  } else {
    timer_interval = 1000;  // เปลี่ยนเป็น 1000ms
  }

  // เชื่อมต่อ Interrupt ใหม่ด้วยระยะเวลาใหม่
  if (ITimer0.attachInterruptInterval(timer_interval * 1000, TimerHandler)) {
    Serial.print(F("Timer interval changed to "));  // แสดงข้อความใน Serial Monitor
    Serial.print(timer_interval);                   // แสดงค่าระยะเวลาใหม่
    Serial.println(F(" ms"));                       // แสดงหน่วยเป็น ms
  } else {
    Serial.println(F("Failed to change timer interval."));  // แจ้งเตือนถ้าไม่สามารถเปลี่ยนระยะเวลาได้
  }

  ITimer0.enableTimer();  // เปิดใช้งาน Timer0 อีกครั้ง
  SW_Flag = false;        // รีเซ็ตธงของสวิตช์
}

// การควบคุม Buzzer และ LED
if (tempC >= TEMP_THRESHOLD || gasAnalog >= GAS_VOLTAGE_THRESHOLD) {
  digitalWrite(BUZZER_PIN, LOW);  // ปิด Buzzer ถ้ามีการตรวจพบอุณหภูมิ/แก๊สเกินเกณฑ์
  digitalWrite(GREEN_LED, LOW);   // ปิด LED สีเขียว
  digitalWrite(RED_LED, HIGH);    // เปิด LED สีแดง
  if (currentMillis - previousMillis >= timer_interval) {
    previousMillis = currentMillis;                                  // อัปเดตค่า previousMillis
    LINE.notify("The detected temperature/gas exceeds threshold!");  // ส่งการแจ้งเตือน
  }
} else {
  digitalWrite(BUZZER_PIN, HIGH);  // เปิด Buzzer ถ้าทุกอย่างปกติ
  digitalWrite(GREEN_LED, HIGH);   // เปิด LED สีเขียว
  digitalWrite(RED_LED, LOW);      // ปิด LED สีแดง
}

// เขียนข้อมูลลงใน SD Card
if (FileOpenFlag) {
  n++;                          // นับจำนวนการประมวลผลหลัก
  myFile.print(n);              // เขียนจำนวนการประมวลผลลงในไฟล์
  myFile.print(" ");            // เขียนช่องว่าง
  myFile.print(tempC);          // เขียนค่าอุณหภูมิ
  myFile.print(" ");            // เขียนช่องว่าง
```

```
    myFile.println(gasAnalog);  // เขียนค่าก๊าซลงในไฟล์
    LEDtoggle = !LEDtoggle;     // เปลี่ยนสถานะของ LEDtoggle

    if (n % 5 == 0) {
      myFile.flush();  // เขียนข้อมูลไปยัง SD Card ทุกๆ 5 รอบ
    }
  }

  // ถ้าถึงจำนวนการสุ่มตัวอย่างที่กำหนด
  if (n == nsample) {
    if (FileOpenFlag) {
      myFile.close();        // ปิดไฟล์
      FileOpenFlag = false;  // รีเซ็ตธงของการเปิดไฟล์
    }
  }
}
```

ผลการทดลองใช้งาน

จากการทดลองการสร้างระบบตรวจจับและแจ้งเตือนการรั่วไหลของก๊าซติดไฟ โดยเริ่มการทดลองจากการใช้ไฟแช็ครนไปยังเซ็นเซอร์ DS18B20 เพื่อใช้ในการตรวจจับอุณหภูมิ และในขณะเดียวกัน ทำการใช้ไฟแช็คอีกอันกดปล่อยก๊าซไปยังเซ็นเซอร์ MQ-2 เพื่อใช้ในการตรวจจับก๊าซไวไฟ จากการทำงานของระบบพบว่าตัว Detector สามารถตรวจจับอุณหภูมิและความเข้มข้นของก๊าซในอากาศได้ โดยเมื่อเซ็นเซอร์วัดอุณหภูมิได้มากกว่า 60 องศาเซลเซียส และตรวจจับค่าก๊าซที่เกินกว่า 3000 mv ระบบจะทำการแจ้งเตือนด้วยเสียงจาก Buzzer และแสดงสถานะไฟ LED เป็นสีแดง โดยข้อมูลต่าง ๆ ที่เกี่ยวข้องกับการตรวจจับก๊าซไวไฟจะถูกแสดงผลบนหน้าจอ OLED ซึ่งประกอบไปด้วย อุณหภูมิ ปริมาณก๊าซ CH4, LPG, CO, H2, Alcohol และเวลาในรูปแบบ Real-time clock จากนั้นระบบจะทำการส่งแจ้งเตือนการรั่วไหลของก๊าซผ่าน Line Notify เพื่อให้ผู้ใช้งานทราบถึงเหตุการณ์ แต่เมื่อปริมาณก๊าซอยู่ในระดับปกติหรือไม่เกินค่าที่กำหนด ระบบจะทำการปิด Buzzer และแสดงสถานะไฟLEDเป็นสีเขียวเพื่อบ่งบอกถึงความปลอดภัยอื่นๆ

ทั้งนี้ผู้ใช้สามารถปรับเปลี่ยนความถี่ในการแจ้งเตือนโดยการ Push Button Switch Module ซึ่งจะสลับความถี่ระหว่าง 0.25 วินาที และ 1 วินาที เพื่อให้เหมาะสมกับการใช้งานในสถานการณ์ต่าง ๆ

สรุป

จากการทดลองระบบตรวจจับและแจ้งเตือนการรั่วไหลของก๊าซติดไฟ โดยใช้เซ็นเซอร์ MQ-2 ร่วมกับเซ็นเซอร์ DS18B20 พบว่าระบบสามารถทำงานได้ตามวัตถุประสงค์ของโครงงานที่ได้ตั้งเป้าไว้ โดยนำรายละเอียดที่สำคัญมาสรุปเป็นข้อ ๆ ได้ดังนี้

1. ในด้านการตรวจจับอุณหภูมิและก๊าซ

   ตัวระบบสามารถตรวจจับอุณหภูมิได้เมื่อมีการใช้ไฟแช็คจุดที่เซ็นเซอร์ DS18B20 โดยหาก ตรวจจับอุณหภูมิและก๊าซได้เกินค่าที่กำนด ระบบจะทำการแจ้งเตือนทันที

2. ในด้านการแจ้งเตือน และการส่งข้อมูล

   เมื่อระบบตรวจพบอุณหภูมิและความเข้มข้นของก๊าซเกินค่าที่กำหนด ระบบสามารถแจ้งเตือนผ่านเสียและแสดงไฟ LED สีแดง เพื่อบ่งบอกถึงสถานการณ์ฉุกเฉินได้ รวมถึงส่งข้อความแจ้งเตือนผ่าน LINE Notify เพื่อให้ผู้ใช้งานได้รับข้อมูลในเวลาจริง

3. ในด้านการแสดงผลข้อมูล

   ข้อมูลที่ได้จากการตรวจจับอุณหภูมิและความเข้มข้นของก๊าซจะแสดงผลบนจอ OLED ซึ่งรวมถึงอุณหภูมิและปริมาณก๊าซในอากาศ รวมถึงข้อมูลวันที่และเวลาที่ตรวจจับได้

4. ในด้านการจัดการสถานะ

   หากค่าปริมาณก๊าซไม่เกินกำหนด ระบบจะปิด Buzzer และแสดงไฟ LED สีเขียว เพื่อบ่งบอกว่าสถานการณ์กลับเข้าสู่ภาวะปกติ

5. ในด้านฟังก์ชันการปรับความถี่ในการแจ้งเตือน

ตัวระบบมีฟังก์ชันการปรับความถี่ในการแจ้งเตือนผ่าน switch ซึ่งทำให้ผู้ใช้งานสามารถควบคุมการแจ้งเตือนได้ตามความต้องการ

จากการสำคัญทั้ง 5 ข้อ จะเห็นสรุปรายละเอียดที่การทำระบบตรวจจับได้ว่าโดยรวม ๆ แล้วไหลของก๊าซติดไฟในและแจ้งเตือนการรั่วเพื่อเพิ่มความปลอดภัยครั้งนี้ ถูกพัฒนาขึ้นมาที่อาจเกิดจากก๊าซรั่วจากอันตรายอันใกล้ตัวประสิทธิภาพไหลได้อย่างมี

รายชื่อสมาชิกกลุ่ม (sec 3)

นางสาวกมลนันท์	วงศ์พรมบุตร	653040117-6
นายจิรายุส	ทองยศ	653040120-7
นายธีรยุทธ	โสดายิ่ง	653040129-9
นางสาวเอื้ออารี	ดีพลงาม	653040149-3
นายศิรัสพล	แสงนาค	653040462-9
นายปฏิภาณ	สีดามาตย์	653040626-5
นางสาวพลอยรัตน์	ทองทับ	653040628-1

References

Dr. Md. Iqbal Bahar Chowdhury. (2023). Gas Detection Using ESP32 and Fire Alarm. สืบค้นเมื่อ 25 กันยายน 2567, จาก https://www.researchgate.net/publication/373824748_Gas_Detection_Using_ESP32_and_Fire_Alarm_Project_Report

Mas Guieta bt Aton, Nur Farhani Imelda binti Abdullah, Norazlina bt. Abd Muttoleb. (2022). Development of LPG Gas Leakage Detection using ESP 32. สืบค้นเมื่อ 25 กันยายน 2567, จาก https://www.researchgate.net/publication/377440979_Development_of_LPG_Gas_Leakage_Detection_using_ESP_32