

CODIGO DE GAS METANO

```
#include "Arduino.h"
#include "LoRaWan_APP.h"
#include <Wire.h>
#include "HT_SSD1306Wire.h"
#include <SPI.h>
#include <SD.h>
#include <time.h>
#include <WiFi.h>

#define SD_CS 39

const char* ssid = "Jair";
const char* password = "123498765";

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -18000;
const int daylightOffset_sec = 0;
int begin();

uint32_t appTxDutyCycle = 5000;

const unsigned long intervaloCSV      = 5000;
const unsigned long intervaloLectura = 200;

unsigned long ultimaInteraccion = 0;
bool pantallaEncendida = true;

uint8_t devEui[] = { 0x70,0xB3,0xD5,0x7E,0xD0,0x07,0x3F,0x19 };
uint8_t appEui[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
uint8_t appKey[] = {
0x74,0xD6,0x6E,0x63,0x45,0x82,0x48,0x27,0xFE,0xC5,0xB7,0x70,0xBA,0x2B,0
x50,0x45 };

uint8_t nwkSKey[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
uint8_t appSKey[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };

uint32_t devAddr = (uint32_t)0x00000000;

LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;
```

```

DeviceClass_t loraWanClass = CLASS_A;
bool overTheAirActivation = true;
bool isTxConfirmed = true;
uint8_t appPort = 2;

bool loraWanAddr = true;
uint8_t confirmedNbTrials = 4;
uint16_t userChannelsMask[6] = {
0xFF00,0x0000,0x0000,0x0000,0x0000,0x0000 };

struct tm timeinfo;
bool rtcConfigurado = false;

void VextON(void) { pinMode(Vext, OUTPUT); digitalWrite(Vext, LOW); }
void VextOFF(void){ pinMode(Vext, OUTPUT); digitalWrite(Vext, HIGH); }

void resetearEstadisticas();
void mostrarPantalla();
void mostrarGraficaGas();
void mostrarDatosActuales();
void mostrarEstadisticas();
void mostrarAlertas();
void mostrarMemoriaSD();
void leerSensor();
void controlarAlertas();
void conectarWiFiYSincronizarHora();
String obtenerFechaHora();
uint32_t obtenerEpoch();
void apagarPantalla();
void encenderPantalla();

SSD1306Wire factory_display(0x3c, 500000, SDA_OLED, SCL_OLED,
GEOMETRY_128_64, RST_OLED);

#define GAS_ANALOGICO 1
#define GAS_DIGITAL 3
#define LED_RED 4
#define LED_GREEN 7
#define LED_BLUE 6
#define BUZZER 5
#define ENCODER_CLK 48
#define ENCODER_DT 47
#define ENCODER_SW 2

```

```

volatile int menuActual = 0;
int opcionAnterior = -1;
const int numMenus = 5;
String menuItems[] = {"Grafica Gas", "Datos
Actual", "Estadisticas", "Alertas", "Memoria SD"};
```

```

#define MAX_DATOS 128
int valoresGas[MAX_DATOS];
int indiceDatos = 0;
int totalDatos = 0;

int gasActual = 0;
int gasDigital = 0;
String nivelActual = "Normal";
String estadoActual = "Normal";
int gasMin = 9999;
int gasMax = 0;
int alertasTotal = 0;

#define UMBRAL_PRECAUCION 55
#define UMBRAL_PELIGRO 70
```

```

SPIClass spiSD(HSPI);
File logFile;
const char* rutaArchivo = "/logs/datos_gas.csv";
int datosGuardados = 0;

volatile bool encoderMoved = false;
void IRAM_ATTR encoderISR() {
    static unsigned long ultima = 0;
    if (millis() - ultima < 5) return;
    ultima = millis();
    encoderMoved = true;
}

static void prepareTxFrame(uint8_t port) {
    uint32_t epoch = obtenerEpoch();
    appDataSize = 7;
    appData[0] = highByte(gasActual);
    appData[1] = lowByte(gasActual);
    appData[2] = (uint8_t)gasDigital;
    appData[3] = (uint8_t)((epoch >> 24) & 0xFF);
```

```

    appData[4] = (uint8_t)((epoch >> 16) & 0xFF);
    appData[5] = (uint8_t)((epoch >> 8) & 0xFF);
    appData[6] = (uint8_t)(epoch & 0xFF);
}

void conectarWiFiYSincronizarHora() {
    Serial.println("\n==== Conectando WiFi para sincronizar hora ====");
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    factory_display.clear();
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(10, 20, "Conectando WiFi...");
    factory_display.display();
    int intentos = 0;
    while (WiFi.status() != WL_CONNECTED && intentos < 20) {
        delay(500);
        Serial.print(".");
        intentos++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\nWiFi conectado!");
        Serial.print("IP: ");
        Serial.println(WiFi.localIP());
        factory_display.clear();
        factory_display.drawString(10, 20, "WiFi OK!");
        factory_display.drawString(10, 35, "Sincronizando...");
        factory_display.display();
        configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
        delay(2000);
        struct tm timeinfo;
        if (getLocalTime(&timeinfo)) {
            rtcConfigurado = true;
            Serial.println("Hora sincronizada: " + obtenerFechaHora());
            factory_display.clear();
            factory_display.drawString(10, 20, "Hora sincronizada!");
            factory_display.drawString(10, 35, obtenerFechaHora());
            factory_display.display();
            delay(2000);
        } else {
            Serial.println("Error al sincronizar hora");
            factory_display.clear();
            factory_display.drawString(10, 25, "Error sync hora");
            factory_display.display();
        }
    }
}

```

```

    delay(2000);
}

WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
Serial.println("WiFi desconectado");
} else {
    Serial.println("\nNo se pudo conectar a WiFi");
    factory_display.clear();
    factory_display.drawString(10, 25, "WiFi FAIL");
    factory_display.display();
    delay(2000);
}
}

unsigned long tiempoAnteriorCSV = 0;
unsigned long ultimaLectura = 0;

void encenderPantalla() {
    if (!pantallaEncendida) {
        factory_display.displayOn();
        pantallaEncendida = true;
    }
    ultimaInteraccion = millis();
}

void setup() {
    Serial.begin(115200);
    delay(500);

    VextON();
    delay(100);

    factory_display.init();
    factory_display.clear();
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(15, 20, "Sistema de");
    factory_display.drawString(10, 35, "Deteccion Metano");
    factory_display.display();

    pinMode(GAS_DIGITAL, INPUT);
    pinMode(LED_RED, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
    pinMode(LED_BLUE, OUTPUT);
}

```

```

pinMode(BUZZER, OUTPUT);

pinMode(ENCODER_DT, INPUT);
pinMode(ENCODER_CLK, INPUT);
pinMode(ENCODER_SW, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(ENCODER_DT), encoderISR,
CHANGE);

Mcu.begin();

for (int i=0;i<MAX_DATOS;i++) valoresGas[i]=0;
digitalWrite(LED_RED, HIGH); delay(80); digitalWrite(LED_RED, LOW);
digitalWrite(LED_GREEN, HIGH); delay(80); digitalWrite(LED_GREEN,
LOW);
digitalWrite(LED_BLUE, HIGH); delay(80); digitalWrite(LED_BLUE,
LOW);
digitalWrite(BUZZER, HIGH); delay(50); digitalWrite(BUZZER, LOW);
delay(1000);
conectarWiFiYSincronizarHora();
Serial.println("\n==== Montando SD ====");
spiSD.begin(26, 33, 34, SD_CS);
delay(100);
if (!SD.begin(SD_CS, spiSD, 4000000, "/sd", 5)) {
  Serial.println("SD FAIL - Reintentando...");
  delay(500);
  if (!SD.begin(SD_CS, spiSD, 1000000, "/sd", 5)) {
    Serial.println("SD NO MONTADA - Continuando sin SD");
  } else {
    Serial.println("SD montada en segundo intento!");
  }
} else {
  Serial.println("SD montada correctamente!");
}
if (SD.cardType() != CARD_NONE) {
  uint8_t cardType = SD.cardType();
  Serial.print("Tipo SD: ");
  if (cardType == CARD_MMC) Serial.println("MMC");
  else if (cardType == CARD_SD) Serial.println("SDSC");
  else if (cardType == CARD_SDHC) Serial.println("SDHC");
  else Serial.println("UNKNOWN");
  uint64_t cardSize = SD.cardSize() / (1024 * 1024);
  Serial.printf("Tamaño: %lluMB\n", cardSize);
  if (!SD.exists("/logs")) {

```

```

        Serial.println("Creando /logs...");
        SD.mkdir("/logs");
    }

    if (!SD.exists(rutaArchivo)) {
        Serial.println("Creando archivo CSV...");
        File f = SD.open(rutaArchivo, FILE_WRITE);
        if (f) {
            f.println("FechaHora,Epoch,GasAnalogico,GasDigital,Estado");
            f.close();
            Serial.println("Archivo CSV creado!");
        } else {
            Serial.println("Error creando CSV");
        }
    } else {
        Serial.println("Archivo CSV ya existe");
    }
}

#if (LORAWAN_DEVEUI_AUTO)
LoRaWAN.generateDeveuiByChipID();
#endif

LoRaWAN.init(loraWanClass, loraWanRegion);
LoRaWAN.setDataRateForNoADR(3);

deviceState = DEVICE_STATE_JOIN;

ultimaInteraccion = millis(); // Iniciar contador de inactividad

Serial.println("\n== Sistema listo ==\n");
}

void loop() {
    unsigned long now = millis();
    if (encoderMoved) {
        encoderMoved = false;
        encenderPantalla();
        int dt = digitalRead(ENCODER_DT);
        int clk = digitalRead(ENCODER_CLK);
        static int ultimoDT = LOW;
        if (dt == LOW && ultimoDT == HIGH) {

```

```

    if (clk == HIGH) menuActual = (menuActual - 1 + numMenus) % numMenus;
    else             menuActual = (menuActual + 1) % numMenus;
}
ultimoDT = dt;
}

if (now - ultimaLectura >= intervaloLectura) {
    ultimaLectura = now;
    leerSensor();
    controlarAlertas();
}

if (SD.cardType() != CARD_NONE && (now - tiempoAnteriorCSV >= intervaloCSV)) {
    tiempoAnteriorCSV = now;
    logFile = SD.open(rutaArchivo, FILE_APPEND);
    if (logFile) {
        uint32_t epoch = obtenerEpoch();
        logFile.print(obtenerFechaHora()); logFile.print(",");
        logFile.print(epoch);           logFile.print(",");
        logFile.print(gasActual);      logFile.print(",");
        logFile.print(gasDigital);     logFile.print(",");
        logFile.println(estadoActual);
        logFile.close();
        datosGuardados++;
        Serial.println("SD OK: escrito registro #" +
String(datosGuardados));
    } else {
        Serial.println("SD FAIL: no se pudo abrir para escribir");
    }
}

if (pantallaEncendida) {
    if (menuActual != opcionAnterior) {
        mostrarPantalla();
        opcionAnterior = menuActual;
    }
    if (menuActual == 0) {
        mostrarPantalla();
    }
}

static unsigned long ultimaPulsacion = 0;
if (now - ultimaPulsacion > 300) {
    if (digitalRead(ENCODER_SW) == LOW) {
        resetearEstadisticas();
    }
}

```

```

        ultimaPulsacion = now;
    }

}

switch (deviceState) {
    case DEVICE_STATE_INIT:
#ifndef LORAWAN_DEVEUI_AUTO
    LoRaWAN.generateDeveuiByChipID();
#endif
    LoRaWAN.init(loraWanClass, loraWanRegion);
    LoRaWAN.setDataRateForNoADR(3);

    deviceState = DEVICE_STATE_JOIN;
    break;
    case DEVICE_STATE_JOIN:
    LoRaWAN.join();
    break;
    case DEVICE_STATE_SEND:
    prepareTxFrame(appPort);
    LoRaWAN.send();
    deviceState = DEVICE_STATE_CYCLE;
    break;
    case DEVICE_STATE_CYCLE:
    txDutyCycleTime = appTxDutyCycle + randr(-APP_TX_DUTYCYCLE_RND,
APP_TX_DUTYCYCLE_RND);
    LoRaWAN.cycle(txDutyCycleTime);
    deviceState = DEVICE_STATE_SLEEP;
    break;
    case DEVICE_STATE_SLEEP:
    LoRaWAN.sleep(loraWanClass);
    break;
    default:
    deviceState = DEVICE_STATE_INIT;
    break;
}
delay(5);
}

void leerSensor() {
    int lecturaADC = analogRead(GAS_ANALOGICO);
    gasDigital = digitalRead(GAS_DIGITAL);
    int adcMin = 300;
    int adcMax = 700;
}

```

```

gasActual = map(lecturaADC, adcMin, adcMax, 0, 100);
gasActual = constrain(gasActual, 0, 100);

valoresGas[indiceDatos] = gasActual;
indiceDatos = (indiceDatos + 1) % MAX_DATOS;
if (totalDatos < MAX_DATOS) totalDatos++;
if (gasActual < gasMin) gasMin = gasActual;
if (gasActual > gasMax) gasMax = gasActual;
Serial.print(obtenerFechaHora());
Serial.print(" | Gas: ");
Serial.print(gasActual); Serial.print("%");
Serial.print(" | Digital: ");
Serial.print(gasDigital);
Serial.print(" | Estado: ");
Serial.println(estadoActual);

}

void controlarAlertas() {
    if (gasActual < UMBRAL_PRECAUCION) {
        nivelActual = "Bajo";
        estadoActual = "Normal";
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, HIGH);
        digitalWrite(LED_BLUE, LOW);
        digitalWrite(BUZZER, LOW);

    } else if (gasActual < UMBRAL_PELIGRO) {
        nivelActual = "Medio";
        estadoActual = "Precaucion";
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_BLUE, HIGH);
        digitalWrite(BUZZER, LOW);

    } else {
        nivelActual = "Alto";
        estadoActual = "PELIGRO";
        digitalWrite(LED_RED, HIGH);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_BLUE, LOW);
        digitalWrite(BUZZER, HIGH);
        alertasTotal++;

    }
}

```

```

void mostrarPantalla() {
    factory_display.clear();

    factory_display.setFont(ArialMT_Plain_10);
    String horaActual = obtenerFechaHora();
    factory_display.drawString(0, 0, horaActual.length() >= 16 ?
horaActual.substring(11, 16) : "SIN_HORA");

    factory_display.drawRect(100, 1, 20, 8);
    factory_display.fillRect(101, 2, 18, 6);
    factory_display.fillRect(120, 3, 2, 4);
    factory_display.drawString(88, 0, "100%");

    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(0, 12, "< " + menuItems[menuActual] + " "
>");
    factory_display.drawLine(0, 22, 128, 22);

    switch(menuActual) {
        case 0: mostrarGraficaGas(); break;
        case 1: mostrarDatosActuales(); break;
        case 2: mostrarEstadisticas(); break;
        case 3: mostrarAlertas(); break;
        case 4: mostrarMemoriaSD(); break;
    }
    factory_display.display();
}

void mostrarMemoriaSD() {
    factory_display.setFont(ArialMT_Plain_10);
    if (SD.cardType() != CARD_NONE) {
        uint64_t cardSize = SD.cardSize() / (1024 * 1024);
        factory_display.drawString(5, 32, "SD OK");
        factory_display.drawString(5, 45, "Tamano: " + String(cardSize) + " "
MB);
        factory_display.drawString(5, 58, "Guardados: " +
String(datosGuardados));
    } else {
        factory_display.drawString(5, 32, "SD NO detect");
    }
}

```

```

void mostrarGraficaGas() {
    if (totalDatos == 0) {
        factory_display.setFont(ArialMT_Plain_10);
        factory_display.drawString(10, 35, "Iniciando..."); 
        factory_display.display();
        return;
    }

    int minGrafica = 0;
    int maxGrafica = 100;
    int rango = maxGrafica - minGrafica;
    int alturaGrafica = 35;
    int yBase = 60;

    factory_display.drawLine(10, 25, 10, yBase);
    factory_display.drawLine(10, yBase, 125, yBase);
    int precaucionPct = (int)(UMBRAL_PRECAUCION / 4095.0 * 100);
    int peligroPct = (int)(UMBRAL_PELIGRO / 4095.0 * 100);
    int yPrecaucion = yBase - (int)((precaucionPct - minGrafica) /
(float)rango * alturaGrafica);
    int yPeligro = yBase - (int)((peligroPct - minGrafica) /
(float)rango * alturaGrafica);
    factory_display.drawLine(10, yPrecaucion, 125, yPrecaucion);
    factory_display.drawLine(10, yPeligro, 125, yPeligro);
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(0, 24, "100%");
    factory_display.drawString(0, yBase - 6, "0%");
    int numPuntos = min(totalDatos, 60);
    for (int i = 0; i < numPuntos - 1; i++) {
        int idx1 = (indiceDatos - numPuntos + i + MAX_DATOS) % MAX_DATOS;
        int idx2 = (indiceDatos - numPuntos + i + 1 + MAX_DATOS) %
MAX_DATOS;
        int x1 = 15 + i * 2;
        int x2 = 15 + (i + 1) * 2;
        int y1 = yBase - (int)((valoresGas[idx1] - minGrafica) /
(float)rango * alturaGrafica);
        int y2 = yBase - (int)((valoresGas[idx2] - minGrafica) /
(float)rango * alturaGrafica);
        y1 = constrain(y1, 25, yBase);
        y2 = constrain(y2, 25, yBase);
        factory_display.drawLine(x1, y1, x2, y2);
    }
    factory_display.drawString(90, 25, String(gasActual) + "%");
}

```

```

factory_display.display();

}

void mostrarDatosActuales() {
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 27, "Gas: " + String(gasActual) + "%");
    factory_display.drawString(70, 27, "(" + nivelActual + ")");
    int barWidth = map(gasActual, 0, 100, 0, 118);
    factory_display.drawRect(5, 38, 118, 8);
    factory_display.fillRect(5, 38, barWidth, 8);
    factory_display.setFont(ArialMT_Plain_16);
    factory_display.drawString(20, 50, estadoActual);
    factory_display.display();
}

void mostrarEstadisticas() {
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 27, "Maximo: " + String(gasMax));
    factory_display.drawString(5, 37, "Minimo: " + String(gasMin));
    factory_display.drawString(5, 47, "Actual: " + String(gasActual));
    factory_display.drawString(5, 57, "Muestras: " + String(totalDatos));
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(80, 57, "Btn=Reset");
}

void mostrarAlertas() {
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 27, "Total Alertas:");
    factory_display.setFont(ArialMT_Plain_16);
    factory_display.drawString(45, 42, String(alertasTotal));
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 58, "Estado: " + estadoActual);

    if (estadoActual == "Normal") {
        factory_display.drawString(95, 27, "OK");
    } else if (estadoActual == "Precaucion") {
        factory_display.drawString(85, 27, "PREC");
    } else if (estadoActual == "PELIGRO") {
        factory_display.drawString(75, 27, "PELIGRO!");
    }
}

String obtenerFechaHora() {

```

```

if (!rtcConfigurado) return "SIN_FECHA";
struct tm timeinfo;
if (!getLocalTime(&timeinfo)) return "SIN_FECHA";

char buffer[25];
sprintf(buffer, "%04d-%02d-%02d %02d:%02d:%02d",
        timeinfo.tm_year + 1900,
        timeinfo.tm_mon + 1,
        timeinfo.tm_mday,
        timeinfo.tm_hour,
        timeinfo.tm_min,
        timeinfo.tm_sec);
return String(buffer);
}

uint32_t obtenerEpoch() {
    time_t now;
    time(&now);
    return (uint32_t)now;
}

void resetearEstadisticas() {
    gasMin = 9999;
    gasMax = 0;
    alertasTotal = 0;
    factory_display.clear();
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(20, 28, "Estadisticas");
    factory_display.drawString(30, 40, "Reseteadas");
    factory_display.display();
    delay(300);
}

```

Inclusión de librerías

El programa comienza cargando las librerías necesarias. Estas permiten manejar el hardware (Arduino), la comunicación LoRaWAN, la pantalla OLED, la tarjeta SD, el tiempo y la conexión WiFi. Sin ellas, el código no podría usar esas funciones.

```

#include "Arduino.h"
#include "LoRaWan_APP.h"
#include <Wire.h>
#include "HT_SSD1306Wire.h"

```

```
#include <SPI.h>
#include <SD.h>
#include <time.h>
#include <WiFi.h>
```

Definir pin de tarjeta SD:

```
#define SD_CS 39
```

Configuración inicial

Las credenciales de la red WiFi para conectarse a internet para sincronizar la hora:

```
const char* ssid = "Jair";
const char* password = "123498765";

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -18000;
const int daylightOffset_sec = 0;
int begin();
```

Intervalos de trabajo

Se establecen los tiempos de ejecución: cada 200 ms se lee el sensor y cada 5 segundos se guarda un registro en la tarjeta SD. Esto organiza el flujo de trabajo del sistema:

```
uint32_t appTxDutyCycle = 5000;

const unsigned long intervaloCSV      = 5000;
const unsigned long intervaloLectura = 200;
```

Variables de control de pantalla

Estas variables permiten saber si la pantalla está encendida y cuándo fue la última interacción. Sirven para manejar la interfaz y ahorrar energía

```
unsigned long ultimaInteraccion = 0;
bool pantallaEncendida = true;
```

Credenciales LoRaWAN

Aquí se configuran las claves necesarias para conectarse a la red LoRaWAN. Son como la “identidad” del dispositivo en la red.

```

uint8_t devEui[] = { 0x70,0xB3,0xD5,0x7E,0xD0,0x07,0x3F,0x19 };
uint8_t appEui[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
uint8_t appKey[] = {
0x74,0xD6,0x6E,0x63,0x45,0x82,0x48,0x27,0xFE,0xC5,0xB7,0x70,0xBA,0x2B,0
x50,0x45 };

uint8_t nwkSKey[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
uint8_t appSKey[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
uint32_t devAddr = (uint32_t)0x00000000;

```

Variables para LoRaWAN

En esta sección se configuran parámetros de la red LoRaWAN: la región, la clase de dispositivo, si se usa activación por aire (OTAA), si los mensajes requieren confirmación, el puerto de aplicación y la máscara de canales. Todo esto define cómo el dispositivo se comunica con la red.

```

LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;
DeviceClass_t loraWanClass = CLASS_A;
bool overTheAirActivation = true;
bool isTxConfirmed = true;
uint8_t appPort = 2;

bool loraWanAdr = true;
uint8_t confirmedNbTrials = 4;
uint16_t userChannelsMask[6] = {
0xFF00,0x0000,0x0000,0x0000,0x0000,0x0000 };

```

Variables de tiempo y energía

Aquí se prepara la estructura para manejar fecha y hora, y se definen funciones para encender o apagar la alimentación externa (Vext), que controla la pantalla OLED.

```

struct tm timeinfo;
bool rtcConfigurado = false;

void VextON(void) { pinMode(Vext, OUTPUT); digitalWrite(Vext, LOW); }
void VextOFF(void){ pinMode(Vext, OUTPUT); digitalWrite(Vext, HIGH); }

```

Prototipos de funciones

Se declaran las funciones que se usarán más adelante. Esto organiza el código y permite que el compilador sepa que esas funciones existen aunque estén definidas después.

```

void resetearEstadisticas();

void mostrarPantalla();

void mostrarGraficaGas();

void mostrarDatosActuales();

void mostrarEstadisticas();

void mostrarAlertas();

void mostrarMemoriaSD();

void leerSensor();

void controlarAlertas();

void conectarWiFiYSincronizarHora();

String obtenerFechaHora();

uint32_t obtenerEpoch();

void apagarPantalla();

void encenderPantalla();

```

Inicialización de la pantalla OLED

Se crea el objeto `factory_display` que representa la pantalla OLED, indicando su dirección I2C, velocidad y geometría.

```
SSD1306Wire factory_display(0x3c, 500000, SDA_OLED, SCL_OLED,
GEOMETRY_128_64, RST_OLED);
```

Definición de pines

Aquí se asignan los pines del ESP32 para el sensor de gas, los LEDs, el buzzer y el encoder rotatorio

```

#define GAS_ANALOGICO 1

#define GAS_DIGITAL    3

#define LED_RED       4

#define LED_GREEN     7

```

```
#define LED_BLUE 6  
  
#define BUZZER 5  
  
#define ENCODER_CLK 48  
  
#define ENCODER_DT 47  
  
#define ENCODER_SW 2
```

Menú

Se define el sistema de menús que se mostrará en la pantalla OLED.

```
volatile int menuActual = 0;  
  
int opcionAnterior = -1;  
  
const int numMenus = 5;  
  
String menuItems[] = {"Grafica Gas", "Datos  
Actual", "Estadisticas", "Alertas", "Memoria SD"};
```

Buffers y estado de medición

Se crean variables para almacenar las lecturas del gas, calcular mínimos y máximos, y contar alertas. Esto permite mostrar estadísticas y gráficas en la pantalla.

```
#define MAX_DATOS 128  
  
int valoresGas[MAX_DATOS];  
  
int indiceDatos = 0;  
  
int totalDatos = 0;  
  
  
int gasActual = 0;  
  
int gasDigital = 0;  
  
String nivelActual = "Normal";  
  
String estadoActual = "Normal";  
  
int gasMin = 9999;  
  
int gasMax = 0;
```

```

int alertasTotal = 0;

#define UMBRAL_PRECAUCION 55
#define UMBRAL_PELIGRO    70

```

Configuración de la tarjeta SD

Aquí se inicializa la interfaz SPI para la tarjeta SD, se define la ruta del archivo CSV y se prepara el contador de registros guardados

```

SPIClass spiSD(HSPI);
File logfile;
const char* rutaArchivo = "/logs/datos_gas.csv";
int datosGuardados = 0;

```

Interrupción del encoder

Se configura una rutina de interrupción para detectar el movimiento del encoder rotatorio. Esto permite cambiar de menú en la pantalla OLED.

```

volatile bool encoderMoved = false;

void IRAM_ATTR encoderISR() {
    static unsigned long ultima = 0;

    if (millis() - ultima < 5) return;

    ultima = millis();

    encoderMoved = true;
}

```

Preparar trama LoRaWAN

Esta función empaqueta los datos del gas y la hora en un arreglo de bytes para enviarlos por LoRaWAN.

```

static void prepareTxFrame(uint8_t port) {
    uint32_t epoch = obtenerEpoch();

```

```

appDataSize = 7;

appData[0] = highByte(gasActual);

appData[1] = lowByte(gasActual);

appData[2] = (uint8_t)gasDigital;

appData[3] = (uint8_t)((epoch >> 24) & 0xFF);

appData[4] = (uint8_t)((epoch >> 16) & 0xFF);

appData[5] = (uint8_t)((epoch >> 8) & 0xFF);

appData[6] = (uint8_t)(epoch & 0xFF);

}

```

Función para conectar WiFi y sincronizar hora

Esta función se encarga de conectar el ESP32 a la red WiFi, mostrar el estado en la pantalla OLED y sincronizar la hora con un servidor NTP. Si la conexión falla, también lo informa en pantalla y en el monitor serial.

- Activa el modo estación WiFi.
- Intenta conectarse con el SSID y contraseña definidos.
- Si logra conectarse, muestra la IP y sincroniza la hora con `pool.ntp.org`.
- Si la hora se sincroniza correctamente, marca `rtcConfigurado = true`.
- Finalmente, desconecta el WiFi para ahorrar energía.

```

void conectarWiFiYSincronizarHora() {
    Serial.println("\n==== Conectando WiFi para sincronizar hora ====");

    WiFi.mode(WIFI_STA);

    WiFi.begin(ssid, password);

    factory_display.clear();

    factory_display.setFont(ArialMT_Plain_10);

    factory_display.drawString(10, 20, "Conectando WiFi...");

    factory_display.display();

    int intentos = 0;

    while (WiFi.status() != WL_CONNECTED && intentos < 20) {

```

```

delay(500);

Serial.print(".");

intentos++;

}

if (WiFi.status() == WL_CONNECTED) {

Serial.println("\nWiFi conectado!");

Serial.print("IP: ");

Serial.println(WiFi.localIP());

factory_display.clear();

factory_display.drawString(10, 20, "WiFi OK!");

factory_display.drawString(10, 35, "Sincronizando...");

factory_display.display();

configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

delay(2000);

struct tm timeinfo;

if (getLocalTime(&timeinfo)) {

rtcConfigurado = true;

Serial.println("Hora sincronizada: " + obtenerFechaHora());

factory_display.clear();

factory_display.drawString(10, 20, "Hora sincronizada!");

factory_display.drawString(10, 35, obtenerFechaHora());

factory_display.display();

delay(2000);

} else {

Serial.println("Error al sincronizar hora");
}
}

```

```

factory_display.clear();

factory_display.drawString(10, 25, "Error sync hora");

factory_display.display();

delay(2000);

}

WiFi.disconnect(true);

WiFi.mode(WIFI_OFF);

Serial.println("WiFi desconectado");

} else {

Serial.println("\nNo se pudo conectar a WiFi");

factory_display.clear();

factory_display.drawString(10, 25, "WiFi FAIL");

factory_display.display();

delay(2000);

}
}

```

Variables de tiempo

Estas variables sirven para controlar cada cuánto se guarda un registro en la SD y cada cuánto se lee el sensor. También ayudan a manejar la interacción con la pantalla.

- **tiempoAnteriorCSV** → última vez que se guardó un registro en CSV.
- **ultimaLectura** → última vez que se leyó el sensor.

```

unsigned long tiempoAnteriorCSV = 0;

unsigned long ultimaLectura = 0;

```

Función para encender la pantalla

Esta función asegura que la pantalla OLED esté encendida y actualiza el tiempo de la última interacción. Se usa cuando el usuario mueve el encoder o presiona el botón.

- Si la pantalla estaba apagada, la enciende.
- Marca `pantallaEncendida = true`.
- Actualiza `ultimaInteraccion` para saber que hubo actividad.

```
void encenderPantalla() {
    if (!pantallaEncendida) {
        factory_display.displayOn();
        pantallaEncendida = true;
    }
    ultimaInteraccion = millis();
}
```

Setup del sistema

En esta función se inicializa todo el hardware y software antes de que el programa empiece a funcionar. Se configura la comunicación serial, la pantalla OLED, los pines de los LEDs, buzzer y encoder, se monta la tarjeta SD y se prepara la conexión LoRaWAN. También se sincroniza la hora vía WiFi y se crea el archivo CSV si no existe.

-Comunicación serial y pantalla de inicio

Se inicia el puerto serial para depuración y se enciende la pantalla OLED mostrando un mensaje de bienvenida

```
Serial.begin(115200);

Serial.begin(115200);

delay(500);

VextON();

delay(100);

factory_display.init();

factory_display.clear();
```

```
factory_display.setFont(ArialMT_Plain_10);

factory_display.drawString(15, 20, "Sistema de");
factory_display.drawString(10, 35, "Detección Metano");

factory_display.display();
```

Configuración de pines de sensores y actuadores

Se definen los pines como entradas o salidas: sensor digital, LEDs, buzzer y el encoder rotatorio.

```
pinMode(GAS_DIGITAL, INPUT);

pinMode(LED_RED, OUTPUT);

pinMode(LED_GREEN, OUTPUT);

pinMode(LED_BLUE, OUTPUT);

pinMode(BUZZER, OUTPUT);

pinMode(ENCODER_DT, INPUT);

pinMode(ENCODER_CLK, INPUT);

pinMode(ENCODER_SW, INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt(ENCODER_DT), encoderISR,
CHANGE);
```

Inicialización del microcontrolador con **Mcu.begin()**

- Esta instrucción **inicializa el microcontrolador** y los recursos básicos que necesita la librería **LoRaWan_APP**.
- Configura los relojes internos, periféricos y prepara la pila LoRaWAN para que el dispositivo pueda conectarse a la red.

```
Mcu.begin();
```

Prueba de hardware (LEDs y buzzer)

Se encienden los LEDs y el buzzer brevemente para comprobar que funcionan correctamente.

```

for (int i=0;i<MAX_DATOS;i++) valoresGas[i]=0;

digitalWrite(LED_RED, HIGH); delay(80); digitalWrite(LED_RED, LOW);

digitalWrite(LED_GREEN, HIGH); delay(80); digitalWrite(LED_GREEN,
LOW);

digitalWrite(LED_BLUE, HIGH); delay(80); digitalWrite(LED_BLUE,
LOW);

digitalWrite(BUZZER, HIGH); delay(50); digitalWrite(BUZZER, LOW);

delay(1000);

```

Conexión WiFi y sincronización de hora

Se conecta a la red WiFi y sincroniza la hora con el servidor NTP.

```
conectarWiFiYSincronizarHora();
```

Montaje de la tarjeta SD

Se inicializa la tarjeta SD, se detecta su tipo y tamaño, y se crea la carpeta `/logs` y el archivo CSV si no existen.

```

Serial.println("\n==== Montando SD ====");

spiSD.begin(26, 33, 34, SD_CS);

delay(100);

if (!SD.begin(SD_CS, spiSD, 4000000, "/sd", 5)) {

    Serial.println("SD FAIL - Reintentando...");

    delay(500);

    if (!SD.begin(SD_CS, spiSD, 1000000, "/sd", 5)) {

        Serial.println("SD NO MONTADA - Continuando sin SD");

    } else {

        Serial.println("SD montada en segundo intento!");

    }

} else {

```

```

Serial.println("SD montada correctamente!");

}

if (SD.cardType() != CARD_NONE) {

    uint8_t cardType = SD.cardType();

    Serial.print("Tipo SD: ");

    if (cardType == CARD_MMC) Serial.println("MMC");

    else if (cardType == CARD_SD) Serial.println("SDSC");

    else if (cardType == CARD_SDHC) Serial.println("SDHC");

    else Serial.println("UNKNOWN");

    uint64_t cardSize = SD.cardSize() / (1024 * 1024);

    Serial.printf("Tamaño: %lluMB\n", cardSize);

    if (!SD.exists("/logs")) {

        Serial.println("Creando /logs...");

        SD.mkdir("/logs");

    }

    if (!SD.exists(rutaArchivo)) {

        Serial.println("Creando archivo CSV...");

        File f = SD.open(rutaArchivo, FILE_WRITE);

        if (f) {

            f.println("FechaHora,Epoch,GasAnalogico,GasDigital,Estado");

            f.close();

            Serial.println("Archivo CSV creado!");

        } else {

            Serial.println("Error creando CSV");

        }

    }

}

```

```

    } else {

        Serial.println("Archivo CSV ya existe");

    }

}

```

Inicialización LoRaWAN

Se prepara la conexión LoRaWAN con la clase, región y tasa de datos. El dispositivo queda listo para unirse a la red.

```

#if (LORAWAN_DEVEUI_AUTO)

LoRaWAN.generateDeveuiByChipID();

#endif

LoRaWAN.init(loraWanClass, loraWanRegion);

LoRaWAN.setDataRateForNoADR(3);

deviceState = DEVICE_STATE_JOIN;

```

Finalización del setup

Se guarda el tiempo de la última interacción y se muestra un mensaje en el monitor serial indicando que el sistema está listo.

```

ultimaInteraccion = millis(); // Iniciar contador de inactividad

Serial.println("\n==== Sistema listo ===\n");

```

Loop del sistema

El `loop()` es la función que se ejecuta **de manera continua** mientras el sistema está encendido. Aquí se concentran todas las tareas repetitivas:

- Detectar movimiento del encoder para cambiar menús.
- Leer el sensor de gas periódicamente.
- Guardar registros en la tarjeta SD.
- Actualizar la pantalla OLED.
- Manejar el botón del encoder para resetear estadísticas.
- Controlar la máquina de estados LoRaWAN (unirse, enviar datos, dormir).

Manejo del encoder (cambio de menús)

Detecta si el encoder rotatorio se movió y cambia el menú en la pantalla OLED.

```
if (encoderMoved) {  
    encoderMoved = false;  
    encenderPantalla();  
    int dt = digitalRead(ENCODER_DT);  
    int clk = digitalRead(ENCODER_CLK);  
    static int ultimoDT = LOW;  
    if (dt == LOW && ultimoDT == HIGH) {  
        if (clk == HIGH) menuActual = (menuActual - 1 + numMenus) %  
numMenus;  
        else menuActual = (menuActual + 1) % numMenus;  
    }  
    ultimoDT = dt;  
}
```

Lectura del sensor y control de alertas

Cada 200 ms se lee el sensor de gas y se actualizan LEDs/buzzer según el nivel de riesgo.

```
if (now - ultimaLectura >= intervaloLectura) {  
  
    ultimaLectura = now;  
  
    leerSensor();  
  
    controlarAlertas();  
  
}
```

Registro en la tarjeta SD

Cada 5 segundos se guarda un registro en el archivo CSV con fecha, hora, valor de gas y estado.

```
if (SD.cardType() != CARD_NONE && (now - tiempoAnteriorCSV >=  
intervaloCSV)) {  
    tiempoAnteriorCSV = now;  
    logFile = SD.open(rutaArchivo, FILE_APPEND);  
    if (logFile) {  
        uint32_t epoch = obtenerEpoch();  
        logFile.print(obtenerFechaHora()); logFile.print(",");  
        logFile.print(epoch); logFile.print(",");  
        logFile.print(gasActual); logFile.print(",");  
        logFile.print(gasDigital); logFile.print(",");  
        logFile.println(estadoActual);  
        logFile.close();  
    }
```

```

    datosGuardados++;
    Serial.println("SD OK: escrito registro #" +
String(datosGuardados));
} else {
    Serial.println("SD FAIL: no se pudo abrir para escribir");
}
}

```

Actualización de pantalla OLED

Se muestra el menú seleccionado (gráfica, estadísticas, alertas, etc.). Si el menú es la gráfica, se repinta continuamente.

```

if (pantallaEncendida) {
    if (menuActual != opcionAnterior) {
        mostrarPantalla();
        opcionAnterior = menuActual;
    }
    if (menuActual == 0) {
        mostrarPantalla();
    }
}

```

Botón del encoder (reset de estadísticas)

Si se presiona el botón del encoder, se reinician las estadísticas de gas.

```

static unsigned long ultimaPulsacion = 0;
if (now - ultimaPulsacion > 300) {
    if (digitalRead(ENCODER_SW) == LOW) {
        resetearEstadisticas();
        ultimaPulsacion = now;
    }
}

```

Máquina de estados LoRaWAN

Controla el ciclo de conexión y transmisión de datos por LoRaWAN: inicialización, unión a la red, envío de datos y modo de bajo consumo.

```

switch (deviceState) {
    case DEVICE_STATE_INIT:
#ifndef LORAWAN_DEVEUI_AUTO
        LoRaWAN.generateDeveuiByChipID();
#endif
}

```

```

LoRaWAN.init(loraWanClass, loraWanRegion);
LoRaWAN.setDataRateForNoADR(3);

deviceState = DEVICE_STATE_JOIN;
break;

case DEVICE_STATE_JOIN:
LoRaWAN.join();
break;

case DEVICE_STATE_SEND:
prepareTxFrame(appPort);
LoRaWAN.send();
deviceState = DEVICE_STATE_CYCLE;
break;

case DEVICE_STATE_CYCLE:
txDutyCycleTime = appTxDutyCycle + randr(-APP_TX_DUTYCYCLE_RND,
APP_TX_DUTYCYCLE_RND);
LoRaWAN.cycle(txDutyCycleTime);
deviceState = DEVICE_STATE_SLEEP;
break;

case DEVICE_STATE_SLEEP:
LoRaWAN.sleep(loraWanClass);
break;

default:
deviceState = DEVICE_STATE_INIT;
break;
}

```

Pequeña pausa

Se agrega un `delay(5)` para dar estabilidad al ciclo y evitar saturar el procesador.

```
delay(5);
```

Funciones auxiliares

Leer sensor

Esta función lee el valor analógico del sensor de gas y lo convierte en un porcentaje (0–100%). También guarda el valor en un buffer para graficarlo, actualiza los mínimos y máximos registrados y muestra la lectura en el monitor serial.

```

void leerSensor() {
    int lecturaADC = analogRead(GAS_ANALOGICO);
    gasDigital = digitalRead(GAS_DIGITAL);
    int adcMin = 300;
    int adcMax = 700;

```

```

gasActual = map(lecturaADC, adcMin, adcMax, 0, 100);
gasActual = constrain(gasActual, 0, 100);
valoresGas[indiceDatos] = gasActual;
indiceDatos = (indiceDatos + 1) % MAX_DATOS;
if (totalDatos < MAX_DATOS) totalDatos++;
if (gasActual < gasMin) gasMin = gasActual;
if (gasActual > gasMax) gasMax = gasActual;
Serial.print(obtenerFechaHora());
Serial.print(" | Gas: ");
Serial.print(gasActual); Serial.print("%");
Serial.print(" | Digital: ");
Serial.print(gasDigital);
Serial.print(" | Estado: ");
Serial.println(estadoActual);
}

```

Control de alertas

Compara el valor de gas con los umbrales definidos. Según el nivel, enciende LEDs de colores y activa o desactiva el buzzer. También actualiza el estado textual (Normal, Precaución, Peligro).

```

void controlarAlertas() {
    if (gasActual < UMBRAL_PRECAUCION) {
        nivelActual = "Bajo";
        estadoActual = "Normal";
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, HIGH);
        digitalWrite(LED_BLUE, LOW);
        digitalWrite(BUZZER, LOW);

    } else if (gasActual < UMBRAL_PELIGRO) {
        nivelActual = "Medio";
        estadoActual = "Precaucion";
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_BLUE, HIGH);
        digitalWrite(BUZZER, LOW);

    } else {
        nivelActual = "Alto";
        estadoActual = "PELIGRO";
        digitalWrite(LED_RED, HIGH);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_BLUE, LOW);
    }
}

```

```

    digitalWrite(BUZZER, HIGH);
    alertasTotal++;
}
}

```

Mostrar pantalla

Dibuja la interfaz en la pantalla OLED. Muestra la hora, el menú actual y según el menú seleccionado llama a la función correspondiente (gráfica, datos actuales, estadísticas, alertas o memoria SD).

```

void mostrarPantalla() {
    factory_display.clear();

    factory_display.setFont(ArialMT_Plain_10);
    String horaActual = obtenerFechaHora();
    factory_display.drawString(0, 0, horaActual.length() >= 16 ?
horaActual.substring(11, 16) : "SIN_HORA");

    factory_display.drawRect(100, 1, 20, 8);
    factory_display.fillRect(101, 2, 18, 6);
    factory_display.fillRect(120, 3, 2, 4);
    factory_display.drawString(88, 0, "100%");

    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(0, 12, "< " + menuItems[menuActual] + " "
>);

    factory_display.drawLine(0, 22, 128, 22);

    switch(menuActual) {
        case 0: mostrarGraficaGas(); break;
        case 1: mostrarDatosActuales(); break;
        case 2: mostrarEstadisticas(); break;
        case 3: mostrarAlertas(); break;
        case 4: mostrarMemoriaSD(); break;
    }
    factory_display.display();
}

```

Mostrar Memoria SD

Informa sobre el estado de la tarjeta SD: si está montada correctamente, su tamaño en MB y cuántos registros se han guardado. Si no se detecta la tarjeta, muestra un mensaje de error.

```

void mostrarMemoriaSD() {
    factory_display.setFont(ArialMT_Plain_10);
    if (SD.cardType() != CARD_NONE) {
        uint64_t cardSize = SD.cardSize() / (1024 * 1024);

```

```

factory_display.drawString(5, 32, "SD OK");
factory_display.drawString(5, 45, "Tamano: " + String(cardSize) + "
MB");
factory_display.drawString(5, 58, "Guardados: " +
String(datosGuardados));
} else {
factory_display.drawString(5, 32, "SD NO detect");
}
}

```

Mostrar GraficaGas

Dibuja una gráfica con los últimos valores de gas almacenados en el buffer. Incluye los ejes, las líneas de umbral de precaución y peligro, y la curva con los valores recientes. También muestra el valor actual en porcentaje.

```

void mostrarGraficaGas() {
if (totalDatos == 0) {
factory_display.setFont(ArialMT_Plain_10);
factory_display.drawString(10, 35, "Iniciando...");
factory_display.display();
return;
}

int minGrafica = 0;
int maxGrafica = 100;
int rango = maxGrafica - minGrafica;
int alturaGrafica = 35;
int yBase = 60;

factory_display.drawLine(10, 25, 10, yBase);
factory_display.drawLine(10, yBase, 125, yBase);
int precaucionPct = (int)(UMBRAL_PRECAUCION / 4095.0 * 100);
int peligroPct = (int)(UMBRAL_PELIGRO / 4095.0 * 100);
int yPrecaucion = yBase - (int)((precaucionPct - minGrafica) /
(float)rango * alturaGrafica);
int yPeligro = yBase - (int)((peligroPct - minGrafica) /
(float)rango * alturaGrafica);
factory_display.drawLine(10, yPrecaucion, 125, yPrecaucion);
factory_display.drawLine(10, yPeligro, 125, yPeligro);
factory_display.setFont(ArialMT_Plain_10);
factory_display.drawString(0, 24, "100%");
factory_display.drawString(0, yBase - 6, "0%");
int numPuntos = min(totalDatos, 60);
for (int i = 0; i < numPuntos - 1; i++) {

```

```

    int idx1 = (indiceDatos - numPuntos + i + MAX_DATOS) % MAX_DATOS;
    int idx2 = (indiceDatos - numPuntos + i + 1 + MAX_DATOS) %
MAX_DATOS;
    int x1 = 15 + i * 2;
    int x2 = 15 + (i + 1) * 2;
    int y1 = yBase - (int)((valoresGas[idx1] - minGrafica) /
(float)rango * alturaGrafica);
    int y2 = yBase - (int)((valoresGas[idx2] - minGrafica) /
(float)rango * alturaGrafica);
    y1 = constrain(y1, 25, yBase);
    y2 = constrain(y2, 25, yBase);
    factory_display.drawLine(x1, y1, x2, y2);
}
factory_display.drawString(90, 25, String(gasActual) + "%");
factory_display.display();
}

```

Mostrar Datos Actuales

Presenta el valor actual del gas en porcentaje, el nivel textual (Bajo, Medio, Alto) y el estado (Normal, Precaución, Peligro). Además, dibuja una barra horizontal que representa gráficamente el porcentaje.

```

void mostrarDatosActuales() {
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 27, "Gas: " + String(gasActual) + "%");
    factory_display.drawString(70, 27, "(" + nivelActual + ")");
    int barWidth = map(gasActual, 0, 100, 0, 118);
    factory_display.drawRect(5, 38, 118, 8);
    factory_display.fillRect(5, 38, barWidth, 8);
    factory_display.setFont(ArialMT_Plain_16);
    factory_display.drawString(20, 50, estadoActual);
    factory_display.display();
}

```

Mostrar Estadísticas

Muestra estadísticas acumuladas: el valor máximo, mínimo, actual y el número de muestras registradas. También indica que el botón del encoder sirve para resetear las estadísticas.

```

void mostrarEstadisticas() {
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(5, 27, "Maximo: " + String(gasMax));
    factory_display.drawString(5, 37, "Minimo: " + String(gasMin));
    factory_display.drawString(5, 47, "Actual: " + String(gasActual));
    factory_display.drawString(5, 57, "Muestras: " + String(totalDatos));
    factory_display.setFont(ArialMT_Plain_10);
    factory_display.drawString(80, 57, "Btn=Reset");
}

```

```
}
```

Mostrar Alertas

Presenta el total de alertas acumuladas y el estado actual. Según el estado, muestra etiquetas como “OK”, “PREC” o “PELIGRO!” en la pantalla OLED.

```
void mostrarAlertas() {  
  
    factory_display.setFont(ArialMT_Plain_10);  
    factory_display.drawString(5, 27, "Total Alertas:");  
    factory_display.setFont(ArialMT_Plain_16);  
    factory_display.drawString(45, 42, String(alertasTotal));  
    factory_display.setFont(ArialMT_Plain_10);  
    factory_display.drawString(5, 58, "Estado: " + estadoActual);  
  
    if (estadoActual == "Normal") {  
        factory_display.drawString(95, 27, "OK");  
    } else if (estadoActual == "Precaucion") {  
        factory_display.drawString(85, 27, "PREC");  
    } else if (estadoActual == "PELIGRO") {  
        factory_display.drawString(75, 27, "PELIGRO!");  
    }  
}
```

ObtenerFechaHora

Devuelve la fecha y hora actual en formato “YYYY-MM-DD HH:MM:SS”. Si la hora no está sincronizada, devuelve “SIN_FECHA”.

```
String obtenerFechaHora() {  
  
    if (!rtcConfigurado) return "SIN_FECHA";  
    struct tm timeinfo;  
    if (!getLocalTime(&timeinfo)) return "SIN_FECHA";  
  
    char buffer[25];  
    sprintf(buffer, "%04d-%02d-%02d %02d:%02d:%02d",  
            timeinfo.tm_year + 1900,  
            timeinfo.tm_mon + 1,  
            timeinfo.tm_mday,  
            timeinfo.tm_hour,  
            timeinfo.tm_min,  
            timeinfo.tm_sec);  
    return String(buffer);  
}
```

Obtener Epoch

Devuelve el tiempo actual en formato epoch (segundos desde 1970). Este valor se usa para guardar registros y transmitir datos por LoRaWAN

```
uint32_t obtenerEpoch() {  
    time_t now;  
    time(&now);  
    return (uint32_t)now;  
}
```

Resetear Estadísticas

Reinicia los valores mínimo y máximo de gas y el contador de alertas. Además, muestra un mensaje en la pantalla OLED indicando que las estadísticas fueron reseteadas.

```
void resetearEstadisticas() {  
    gasMin = 9999;  
    gasMax = 0;  
    alertasTotal = 0;  
    factory_display.clear();  
    factory_display.setFont(ArialMT_Plain_10);  
    factory_display.drawString(20, 28, "Estadisticas");  
    factory_display.drawString(30, 40, "Reseteadas");  
    factory_display.display();  
    delay(300);  
}
```