LAMPIRAN

Lampiran 1 Source Code *Arduino*

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiManager.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>

// ==========================================
// 1. CONFIGURATION & SETTINGS
// ==========================================
namespace Config {
   constexpr int PIN_MQ2_ANALOG = 1;
   constexpr int PIN_MQ2_DIGITAL = 11;
   constexpr int PIN_SENSOR_POWER = 12;
   constexpr int PIN_LED_STATUS = 13;
   constexpr int PIN_BUZZER = 4;

   constexpr int GAS_THRESHOLD_SOFT = 850;
   constexpr int HYSTERESIS = 200;
   constexpr unsigned long READ_INTERVAL_MS = 1000;
   constexpr unsigned long WARMUP_TIME_MS = 60000; // 1 Menit

   constexpr uint32_t SERIAL_BAUD = 115200;

   const char* TELEGRAM_BOT_TOKEN = "change-me";
   const char* TELEGRAM_CHAT_ID = "change-me";
}

enum class SystemState {
```

```
    BOOTING,
    WIFI_SETUP,
    WARMUP,
    IDLE,
    DANGER
};

// ==========================================
// 2. HARDWARE SERVICES
// ==========================================
class AlertController {
private:
    const int _pinLed;
    const int _pinBuzzer;
    unsigned long _lastLedToggle;
    unsigned long _lastBuzzerBeep;
    int _ledState;
    SystemState _currentState;

    // Logika Normal: HIGH = Nyala, LOW = Mati
    const int BUZZER_ON = HIGH;
    const int BUZZER_OFF = LOW;

public:
    AlertController(int pinLed, int pinBuzzer)
        : _pinLed(pinLed), _pinBuzzer(pinBuzzer), _lastLedToggle(0),
          _lastBuzzerBeep(0), _ledState(LOW), _currentState(SystemState::BOOTING)
{}

    void init() {
```

```
    pinMode(_pinLed, OUTPUT);
    digitalWrite(_pinLed, LOW);
    if (_pinBuzzer != -1) {
        pinMode(_pinBuzzer, OUTPUT);
        digitalWrite(_pinBuzzer, BUZZER_OFF);
    }
}

void setMode(SystemState state) {
    _currentState = state;
    _lastLedToggle = millis();
    _lastBuzzerBeep = millis();

    if (state == SystemState::IDLE) {
        digitalWrite(_pinLed, LOW);
        if (_pinBuzzer != -1) digitalWrite(_pinBuzzer, BUZZER_OFF);
    }
    else if (state == SystemState::WIFI_SETUP) {
        // Saat setting WiFi, cuma LED yang nyala (Buzzer diam biar gak berisik)
        digitalWrite(_pinLed, HIGH);
        if (_pinBuzzer != -1) digitalWrite(_pinBuzzer, BUZZER_OFF);
    }
}

void update() {
    unsigned long now = millis();

    if (_currentState == SystemState::WARMUP) {
        // LED Kedip stabil tiap 500ms
        if (now - _lastLedToggle >= 500) {
```

```
            _lastLedToggle = now;
            _ledState = !_ledState;
            digitalWrite(_pinLed, _ledState);
        }

        // Buzzer Bip singkat (100ms) setiap 5 detik
        if (_pinBuzzer != -1) {
            if (now - _lastBuzzerBeep >= 5000) {
                _lastBuzzerBeep = now;
                digitalWrite(_pinBuzzer, BUZZER_ON);
            } else if (now - _lastBuzzerBeep >= 100) {
                digitalWrite(_pinBuzzer, BUZZER_OFF); // Matikan setelah 100ms
            }
        }
    }
    else if (_currentState == SystemState::DANGER) {
        // Bahaya: LED & Buzzer nyala berbarengan kedip sangat cepat (150ms)
        if (now - _lastLedToggle >= 150) {
            _lastLedToggle = now;
            _ledState = !_ledState;
            digitalWrite(_pinLed, _ledState);
            if (_pinBuzzer != -1) digitalWrite(_pinBuzzer, _ledState ? BUZZER_ON :
BUZZER_OFF);
        }
    }
  }
};

class GasSensorService {
private:
```

```
    const int _pinAnalog;
    const int _pinDigital;
    const int _pinPower;
    int _filteredAnalogValue;
    bool _lastDigitalState;
    unsigned long _powerOnTime;

    int readFilteredAnalog() {
        long sum = 0;
        constexpr int samples = 10;
        for (int i = 0; i < samples; i++) {
            sum += analogRead(_pinAnalog);
            delayMicroseconds(50);
        }
        return sum / samples;
    }

public:
    GasSensorService(int pinA, int pinD, int pinP)
        : _pinAnalog(pinA), _pinDigital(pinD), _pinPower(pinP),
          _filteredAnalogValue(0), _lastDigitalState(false), _powerOnTime(0) {}

    void init() {
        pinMode(_pinPower, OUTPUT);
        digitalWrite(_pinPower, LOW);
        pinMode(_pinAnalog, INPUT);
        pinMode(_pinDigital, INPUT);
    }

    void powerOn() {
```

```
        digitalWrite(_pinPower, HIGH);
        _powerOnTime = millis();
    }

    void update() {
        _filteredAnalogValue = readFilteredAnalog();
        _lastDigitalState = (digitalRead(_pinDigital) == LOW);
    }

    int getAnalogValue() const { return _filteredAnalogValue; }
    bool isHardwareTriggered() const { return _lastDigitalState; }

    bool isWarmedUp() const {
        if (_powerOnTime == 0) return false;
        return (millis() - _powerOnTime) >= Config::WARMUP_TIME_MS;
    }

    unsigned long getRemainingWarmup() const {
        if (_powerOnTime == 0) return Config::WARMUP_TIME_MS / 1000;
        unsigned long elapsed = millis() - _powerOnTime;
        if (elapsed >= Config::WARMUP_TIME_MS) return 0;
        return (Config::WARMUP_TIME_MS - elapsed) / 1000;
    }
};

// ============================================
// 3. NETWORK & NOTIFICATION SERVICES
// ============================================
class TelegramNotifier {
private:
```

```
    const char* _token;
    const char* _chatId;
    WiFiClientSecure _client;

    struct AlertPayload {
        TelegramNotifier* instance;
        int gasValue;
        bool isHwTriggered;
        unsigned long uptimeSec;
    };

    static void asyncTask(void* parameter) {
        AlertPayload* payload = static_cast<AlertPayload*>(parameter);
        payload->instance->executeSend(payload->gasValue,  payload->isHwTriggered,
payload->uptimeSec);
        delete payload;
        vTaskDelete(NULL);
    }

    void executeSend(int gasValue, bool isHwTriggered, unsigned long uptimeSec) {
        if (WiFi.status() != WL_CONNECTED) {
            Serial.println("[TELEGRAM] Aborted: WiFi Disconnected.");
            return;
        }

        HTTPClient https;
        char url[128];
        snprintf(url, sizeof(url), "https://api.telegram.org/bot%s/sendMessage", _token);

        String message;
```

```
    message.reserve(256);
    message = "⚠️ *BAHAYA KEBOCORAN GAS* ⚠️\n\n";
    message += "📈 Level Sensor: " + String(gasValue) + "\n";
    message += "🎛️ HW Status: " + String(isHwTriggered ? "BAHAYA (HW Trigger)" : "Tidak (SW Only)") + "\n";
    message += "📍 Lokasi: Dapur\n";
    message += "⏱️ Uptime: " + String(uptimeSec) + "s";

    String jsonPayload = "{\"chat_id\": \"" + String(_chatId) + "\", \"text\": \"" + message + "\", \"parse_mode\": \"Markdown\"}";

    if (https.begin(_client, url)) {
      https.addHeader("Content-Type", "application/json");
      int httpCode = https.POST(jsonPayload);

      if (httpCode == 200 || httpCode == 201) {
        Serial.println("[TELEGRAM] Notifikasi berhasil dikirim!");
      } else {
        Serial.printf("[TELEGRAM] Gagal kirim. HTTP Code: %d\n", httpCode);
      }
      https.end();
    }
  }

public:
  TelegramNotifier(const char* token, const char* chatId)
    : _token(token), _chatId(chatId) {
    _client.setInsecure();
  }
```

```
    void sendAlertAsync(int gasValue, bool isHwTriggered) {
        AlertPayload* payload = new AlertPayload{this, gasValue, isHwTriggered, millis() / 1000};
        xTaskCreatePinnedToCore(asyncTask, "TeleTask", 8192, payload, 1, NULL, 0);
    }
};

// ==========================================
// 4. MAIN APPLICATION
// ==========================================
WiFiManager wifiManager;
TelegramNotifier telegram(Config::TELEGRAM_BOT_TOKEN, Config::TELEGRAM_CHAT_ID);
GasSensorService sensor(Config::PIN_MQ2_ANALOG, Config::PIN_MQ2_DIGITAL, Config::PIN_SENSOR_POWER);
AlertController alertSystem(Config::PIN_LED_STATUS, Config::PIN_BUZZER);

SystemState currentState = SystemState::BOOTING;
unsigned long lastSensorReadTime = 0;

void setup() {
    Serial.begin(Config::SERIAL_BAUD);
    delay(2000);
    Serial.println("\n\n[SYSTEM] INIT PHASE START");

    alertSystem.init();
    sensor.init();

    wifiManager.setDebugOutput(false);
```

```
    Serial.println("[SYSTEM] Starting WiFiManager...");

    // Set state ke WIFI_SETUP biar buzzer diam dan LED nyala statis
    alertSystem.setMode(SystemState::WIFI_SETUP);

    if (!wifiManager.autoConnect("ESP32-S3-GAS")) {
        Serial.println("[FATAL] WiFi Failed! Rebooting...");
        delay(3000);
        ESP.restart();
    }

    Serial.println("[SYSTEM] WiFi Connected. Turning ON Sensor Power...");
    sensor.powerOn();

    // Masuk ke mode Warmup (1 menit)
    currentState = SystemState::WARMUP;
    alertSystem.setMode(currentState);
}

void loop() {
    // Fungsi ini yang membuat pola kedip dan bip jalan tanpa nge-block sistem
    alertSystem.update();

    unsigned long currentMillis = millis();
    if (currentMillis - lastSensorReadTime >= Config::READ_INTERVAL_MS) {
        lastSensorReadTime = currentMillis;

        sensor.update();
        int analogVal = sensor.getAnalogValue();
```

```
    bool hwTrigger = sensor.isHardwareTriggered();

    if (currentState == SystemState::WARMUP) {
      if (!sensor.isWarmedUp()) {
        Serial.printf("[WARMUP] %lu s remaining | Val: %d\n",
sensor.getRemainingWarmup(), analogVal);
        return; // Tunggu sampai panas
      } else {
        Serial.println("[SYSTEM] Warm-up Selesai. Masuk mode Siaga (IDLE).");
        currentState = SystemState::IDLE;
        alertSystem.setMode(currentState);
      }
    }

    Serial.printf("[SENSOR] Val: %4d | HW: %d | State: %d\n", analogVal,
hwTrigger, static_cast<int>(currentState));

    if (analogVal > Config::GAS_THRESHOLD_SOFT && currentState !=
SystemState::DANGER) {
      Serial.println("[ALERT] BAHAYA! Melewati ambang batas.");
      currentState = SystemState::DANGER;
      alertSystem.setMode(currentState);
      telegram.sendAlertAsync(analogVal, hwTrigger);
    }
    else if (analogVal < (Config::GAS_THRESHOLD_SOFT -
Config::HYSTERESIS) && currentState == SystemState::DANGER) {
      Serial.println("[SYSTEM] Kondisi Aman. Mengembalikan ke IDLE.");
      currentState = SystemState::IDLE;
      alertSystem.setMode(currentState);
    }
```

```
    }
}
```

MQ-2 Semiconductor Sensor for Combustible Gas

Sensitive material of MQ-2 gas sensor is SnO_2, which with lower conductivity in clean air. When the target combustible gas exist, The sensor's conductivity is more higher along with the gas concentration rising. Please use simple electrocircuit, Convert change of conductivity to correspond output signal of gas concentration.

MQ-2 gas sensor has high sensitity to LPG, Propane and Hydrogen, also could be used to Methane and other combustible steam, it is with low cost and suitable for different application.

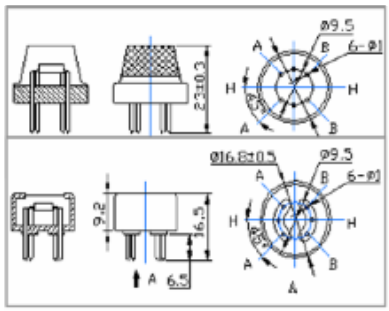
Character

*Good sensitivity to Combustible gas in wide range
* High sensitivity to LPG, Propane and Hydrogen
* Long life and low cost
* Simple drive circuit

Application

* Domestic gas leakage detector
* Industrial Combustible gas detector
* Portable gas detector

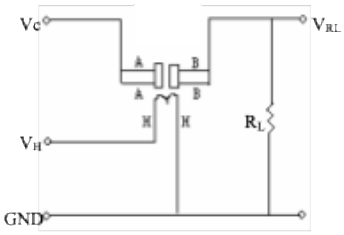
Configuration



Technical Data

Model No.			MQ-2
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Combustible gas and smoke
Concentration			300-10000ppm (Combustible gas)
Circuit	Loop Voltage	V_C	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V ACorDC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω (Room Tem.)
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_s	2KΩ-20KΩ(in 2000ppm C_4H_{10})
	Sensitivity	S	Rs(in air)/Rs(1000ppm isobutane)≥5
	Slope	α	≤0.6($R_{5000ppm}/R_{3000ppm}$ CH_4)
Condition	Tem. Humidity		20℃±2℃; 65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V; V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

Basic test loop



The above is basic test circuit of the sensor. The sensor need to be put 2 voltage, heater voltage(VH) and test voltage(VC). VH used to supply certified working temperature to the sensor, while VC used to detect voltage (VRL) on load resistance (RL) whom is in series with sensor. The sensor has light polarity, Vc need DC power. VC and VH could use same power circuit with precondition to assure performance of sensor. In order to make the sensor with better performance, suitable RL value is needed:

Power of Sensitivity body(Ps):

$Ps=Vc^2\times Rs/(Rs+RL)^2$

Lampiran 3 ESP32-S3 Super Mini Datasheet

ESP32-S3 Supermini

The ESP32-S3 SuperMini is an IoT mini development board based on the Espressif ESP32-S3 WiFi/ Bluetooth dual-mode chip. The ESP32-S3 is a 32-bit RISC-V CPU that contains the FPU (floating point unit), which can perform 32-bit single-precision operations and has powerful computing power. It has excellent RF performance and supports IEEE 802.11b /g/n WiFi and Bluetooth 5 (LE) protocols. The board comes with an external antenna to enhance signal strength for wireless applications. It also has a small and delicate form factor combined with a single-sided surface mount design. It is equipped with a wealth of interfaces, with 11 digital I/ OS that can be used as PWM pins and 4 analog I/ OS that can be used as ADC pins. It supports four serial interfaces: UART, I2C and SPI. The board also has a small reset button and a boot loader mode button.

Based on the above features, ESP32S3SuperMini is positioned as a high-performance, low-power and cost-effective iot mini development board suitable for low-power iot applications and wireless wearable applications.

Product parameter

- **Powerful CPU: ESP32-S3, 32-bit RISC-V single-core processor, running at up to 160 MHz**
- **WiFi: 802.11b/g/n protocol, 2.4GhHz, support Station mode, SoftAP mode, SoftAP+Station mode, hybrid mode**
- **Bluetooth: Bluetooth 5.0**
- **Ultra-low power consumption: deep sleep power consumption of about 43μA**
- **Rich board resources: 400KB SRAM, 384KB ROM built-in 4Mflash.**
- **Chip model: ESP32S3FH4R2**
- **Ultra-small size: As small as the thumb (22.52x18mm) classic shape, suitable for wearables and small projects**
- **Reliable security features: Encryption hardware accelerators that support AES-128/256, hashing, RSA, HMAC, digital signatures, and secure startup**
- **Rich interface: 1xI2C, 1xSPI, 2xUART, 11xGPIO(PWM), 4xADC**
- **Onboard RGB, blue LED: common GPIO48 pin**

Pin diagram