Unit MQ Arduino 使用チュートリアル

1. 準備作業

- 環境構築： Arduino IDE スタートガイド を参考に IDE をインストールし、使用する開発ボードに応じたボードマネージャおよび必要なドライバライブラリをインストールしてください。
- 使用するドライバライブラリ：
  - M5Unified
  - M5GFX
  - M5Unit-MQ
- 使用するハードウェア製品：
  - Core2 v1.1
  - Unit MQ

2. 注意事項

ピン互換性

各ホストのピン設定は異なるため、ユーザーがより便利に使用できるよう 、M5Stack 公式はピン互換性表を提供しています。実際のピン接続状況に応じてサンプルコードを修正してください。

3. サンプルコード

このチュートリアルでは、Core2 v1.1 を主な制御デバイスとし、Unit MQ を組み合わせて使用します。本電流電圧測定ユニットは I2C 方式で通信します。実際の回路接続に応じてコード内のピン定義を修正してください。デバイス接続後の対応 I2C ピンは `G33（SCL）`、`G32（SDA）` です。

説明

1.**高レベルを20秒間**維持した後に、センサーデータが有効になります。低レベルに切り替えると、データ有効フラグは即座に0になります。データ有効フラグと加熱制御ピンの High/Low レベルの関係については図解を参照してください。
2.検出された可燃性ガスの**濃度が高い**ほど、Unit MQ が出力する**電圧は高く**なります。電圧と濃度 ppm に関する情報は、MQ-5 データシートを参照してください。

- Unit MQ は連続加熱モード（`HEAT_MODE_CONTINUOUS`）と間欠加熱モード（`HEAT_MODE_PIN_SWITCH`）の 2 種類の加熱モードを設定できます。取得できるデータは、センサー電圧、ADC 基準電圧、デバイス温度などがあり、必要に応じて選択してください。
- データ取得 API は以下の通りです：

```
getValidTags()                              // データ有効タグ
getReferenceVoltage()                       // 基準電圧
getNTCADC12bit()                            // NTC ADC
getNTCVoltage()                             // NTC 電圧
getNTCResistance()                          // NTC 抵抗値
getNTCTemperature(uint16_t ntcResistance)   // NTC 温度
getMQADC12bit()                             // センサー ADC
getMQVoltage()                              // センサー電圧
```

連続加熱モード

cpp

```cpp
#include <M5Unified.h>
#include <M5GFX.h>
#include "m5_unit_mq.hpp"

#define I2C_SDA   (32)     /**< I2C SDA pin number */
#define I2C_SCL   (33)     /**< I2C SCL pin number */
#define I2C_SPEED (400000) /**< I2C bus speed in Hz */

M5UnitMQ unitMQ; /**< Instance of the Unit MQ gas sensor */

/* Sensor status and data variables */
static led_status_t ledStatus        = LED_WORK_STATUS_OFF;  /**< Current LED state */
static heat_mode_t heatMode          = HEAT_MODE_CONTINUOUS; /**< Current heating mode */
static mq_adc_valid_tags_t validTags = VALID_TAG_INVALID;    /**< MQ ADC data validity flag */
static uint8_t highLevelTime         = 0;                    /**< High-level heating time (not used
static uint8_t lowLevelTime          = 0;                    /**< Low-level heating time (not used
static uint16_t mqADC12bit           = 0;                    /**< MQ sensor raw ADC value (12-bit)
static uint16_t ntcADC12bit          = 0;                    /**< NTC sensor raw ADC value (12-bit)
static uint16_t ntcResistance        = 0;                    /**< Calculated NTC resistance in Ohms
static uint16_t referenceVoltage     = 0;                    /**< Reference voltage in millivolts */
static uint16_t mqVoltage            = 0;                    /**< Calculated MQ sensor voltage in m
static uint16_t ntcVoltage           = 0;                    /**< Calculated NTC sensor voltage in
float temperature                    = 0.0f;                 /**< Calculated temperature in degrees
static uint8_t firmwareVersion       = 0;                    /**< Sensor firmware version */

/**
 * @brief Arduino setup function, runs once at startup.
 *
 * Initializes the M5 hardware and Unit MQ sensor with I2C configuration.
 * Retries sensor initialization until successful.
 * Sets sensor heating mode and turns on the sensor LED.
 */
void setup()
{
    M5.begin();
    M5.Display.setRotation(1);
    M5.Display.clear(TFT_WHITE);
    M5.Display.setFont(&fonts::FreeMonoBold9pt7b);
    M5.Display.setTextColor(TFT_BLACK);
    Serial.begin(115200);

    Serial.println("========== Unit MQ Initialization ==========");

    // Initialize Unit MQ sensor on I2C bus, retry until success
    while (!unitMQ.begin(&Wire, UNIT_MQ_I2C_BASE_ADDR, I2C_SDA, I2C_SCL, I2C_SPEED)) {
        Serial.println("[ERROR] Unit MQ initialization failed! Retrying...");
        delay(1000);
    }
    Serial.println("[INFO] Unit MQ initialization succeeded.");

```

```
    // Set the heating mode to continuous heating
    unitMQ.setHeatMode(HEAT_MODE_CONTINUOUS);
    Serial.println("[INFO] Heating mode set to CONTINUOUS.");

    // Turn on the sensor's LED indicator
    unitMQ.setLEDState(LED_WORK_STATUS_ON);
    Serial.println("[INFO] LED status set to ON.");
}

/**
 * @brief Arduino main loop function, runs repeatedly.
 *
 * Reads sensor data including LED status, valid tags, firmware version,
 * NTC sensor ADC and temperature, and MQ sensor ADC and voltage if valid.
 * Prints all the relevant data to Serial for monitoring.
 */
void loop()
{
    // Get the current heating mode
    heatMode = unitMQ.getHeatMode();

    // Retrieve current LED state
    ledStatus = unitMQ.getLEDState();

    // Retrieve validity tag for MQ sensor data
    validTags = unitMQ.getValidTags();

    // Retrieve sensor firmware version
    firmwareVersion = unitMQ.getFirmwareVersion();

    // Always read NTC sensor data and calculate temperature
    ntcADC12bit      = unitMQ.getNTCADC12bit();
    referenceVoltage = unitMQ.getReferenceVoltage();
    ntcVoltage       = unitMQ.getNTCVoltage();
    ntcResistance    = unitMQ.getNTCResistance();
    temperature      = unitMQ.getNTCTemperature(ntcResistance);

    // Print sensor status and readings
    Serial.println("======== Unit MQ Status ========");
    Serial.printf("Firmware Version   : %d\n", firmwareVersion);
    Serial.printf("Heating Mode       : %s\n", heatMode == HEAT_MODE_CONTINUOUS ? "CONTINUOUS" : "SI
    Serial.printf("LED Status         : %s\n", ledStatus == LED_WORK_STATUS_ON ? "ON" : "OFF");
    Serial.printf("NTC ADC (12-bit)   : %u\n", ntcADC12bit);
    Serial.printf("Reference Voltage  : %u mV\n", referenceVoltage);
    Serial.printf("NTC Voltage        : %u mV\n", ntcVoltage);
    Serial.printf("NTC Resistance     : %u Ohm\n", ntcResistance);
    Serial.printf("Temperature        : %.2f °C\n", temperature);

    M5.Display.clear(TFT_WHITE);
    M5.Display.drawCenterString("Unit MQ Status", M5.Display.width()/2, 0);
    M5.Display.setCursor(0, 20);
    M5.Display.printf("Firmware Version : %d\n", firmwareVersion);
    M5.Display.printf("Heating Mode     : %s\n", heatMode == HEAT_MODE_CONTINUOUS ? "CONTINUOUS" : '
```

```
    M5.Display.printf("LED Status       : %s\n", ledStatus == LED_WORK_STATUS_ON ? "ON" : "OFF");
    M5.Display.printf("NTC ADC (12-bit) : %u\n", ntcADC12bit);
    M5.Display.printf("Reference Voltage: %u mV\n", referenceVoltage);
    M5.Display.printf("NTC Voltage      : %u mV\n", ntcVoltage);
    M5.Display.printf("NTC Resistance   : %u Ohm\n", ntcResistance);
    M5.Display.printf("Temperature      : %.2f C\n", temperature);

    // Print MQ sensor readings only if data is valid
    if (validTags == VALID_TAG_VALID) {
        mqADC12bit = unitMQ.getMQADC12bit();
        mqVoltage  = unitMQ.getMQVoltage();

        Serial.printf("Valid Tags         : 0x%02X (MQ data valid)\n", validTags);
        Serial.printf("MQ ADC (12-bit)    : %u\n", mqADC12bit);
        Serial.printf("MQ Voltage         : %u mV\n", mqVoltage);

        M5.Display.printf("Valid Tags : 0x%02X (valid)\n", validTags);
        M5.Display.printf("MQ ADC (12-bit)  : %u\n", mqADC12bit);
        M5.Display.printf("MQ Voltage       : %u mV\n", mqVoltage);
    } else {
        Serial.printf("Valid Tags         : 0x%02X (MQ data invalid)\n", validTags);
        Serial.println("[WARNING] MQ data invalid. Skipping MQ readings.");

        M5.Display.printf("Valid Tags : 0x%02X (invalid)\n", validTags);
        M5.Display.println("[WARNING] MQ data invalid. Skipping MQ readings.");
    }

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

    delay(1000);  // Delay 1 second before next reading
}
```

間欠加熱モード

cpp

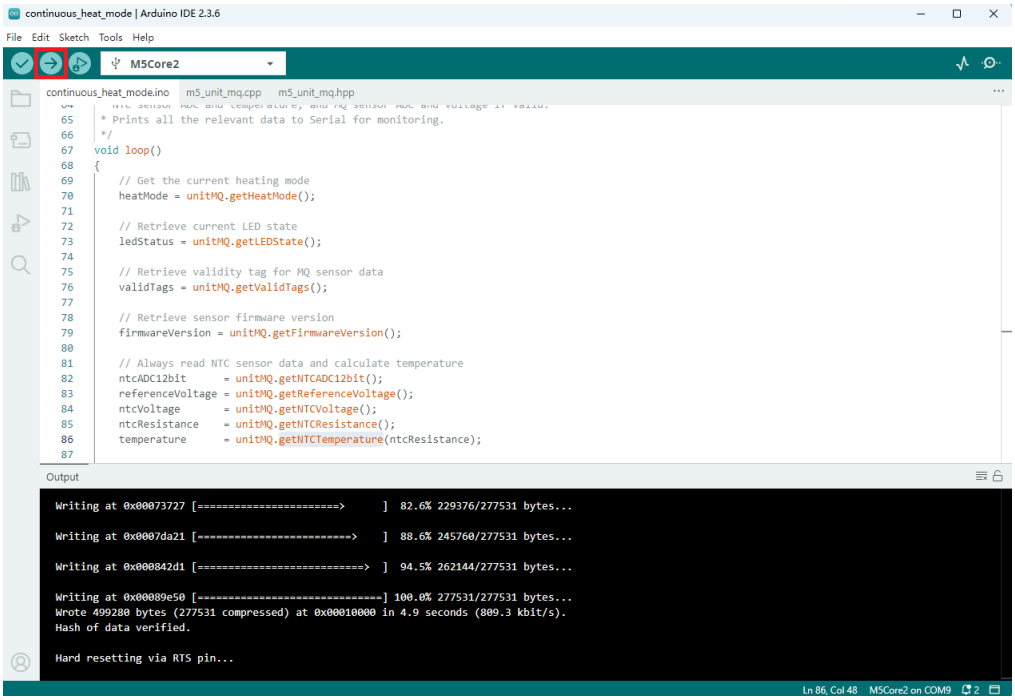
```cpp
#include <M5Unified.h>
#include <M5GFX.H>
#include "m5_unit_mq.hpp"

#define I2C_SDA   (32)     /**< I2C SDA pin number */
#define I2C_SCL   (33)     /**< I2C SCL pin number */
#define I2C_SPEED (400000) /**< I2C bus speed in Hz */

#define HIGH_LEVEL_TIME (30) /**< High level duration for pin switch mode in seconds */
#define LOW_LEVEL_TIME  (10)  /**< Low level duration for pin switch mode in seconds */

M5UnitMQ unitMQ; /**< Instance of the Unit MQ gas sensor class */

/* Sensor and system state variables */
static led_status_t ledStatus        = LED_WORK_STATUS_OFF;  /**< Current LED status */
static heat_mode_t heatMode          = HEAT_MODE_PIN_SWITCH; /**< Heating mode of the sensor */
static mq_adc_valid_tags_t validTags = VALID_TAG_INVALID;    /**< Validity tag for MQ sensor ADC da
static uint8_t highLevelTime         = 0;                    /**< High-level time setting for pin s
static uint8_t lowLevelTime          = 0;                    /**< Low-level time setting for pin sw
static uint16_t mqAdc12bit           = 0;                    /**< Raw 12-bit ADC reading from MQ se
static uint16_t ntcAdc12bit          = 0;                    /**< Raw 12-bit ADC reading from NTC t
static uint16_t ntcResistance        = 0;                    /**< Calculated resistance of NTC sens
static uint16_t referenceVoltage     = 0;                    /**< Reference voltage value (mV) */
static uint16_t mqVoltage            = 0;                    /**< Calculated voltage of MQ sensor o
static uint16_t ntcVoltage           = 0;                    /**< Calculated voltage of NTC sensor
float temperature                    = 0.0f;                 /**< Calculated temperature in Celsius
static uint8_t firmwareVersion       = 0;                    /**< Firmware version of the Unit MQ s

void setup()
{
    M5.begin();
    M5.Display.setRotation(1);
    M5.Display.clear(TFT_WHITE);
    M5.Display.setFont(&fonts::FreeMonoBold9pt7b);
    M5.Display.setTextColor(TFT_BLACK);
    Serial.begin(115200);

    Serial.println("========== Unit MQ Initialization ==========");

    // Initialize the Unit MQ sensor with I2C settings.
    // Retry initialization until successful.
    while (!unitMQ.begin(&Wire, UNIT_MQ_I2C_BASE_ADDR, I2C_SDA, I2C_SCL, I2C_SPEED)) {
        Serial.println("[ERROR] Unit MQ initialization failed! Retrying...");
        delay(1000);
    }
    Serial.println("[INFO] Unit MQ initialization succeeded.");

    // Set heating mode to PIN SWITCH mode, which alternates heating between HIGH and LOW levels.
    unitMQ.setHeatMode(HEAT_MODE_PIN_SWITCH);
    Serial.println("[INFO] Heating mode set to PIN SWITCH mode.");
```

```
51
52      // Configure the durations for high and low heating levels in PIN SWITCH mode.
53      unitMQ.setPulseTime(HIGH_LEVEL_TIME, LOW_LEVEL_TIME);
54      Serial.println("[INFO] Pin level switch time set to 30s HIGH and 5s LOW.");
55
56      // Turn on the sensor LED indicator.
57      unitMQ.setLEDState(LED_WORK_STATUS_ON);
58      Serial.println("[INFO] LED status set to ON.");
59  }
60
61  void loop()
62  {
63      // Get the current heating mode
64      heatMode = unitMQ.getHeatMode();
65
66      // Read the current LED status from the sensor.
67      ledStatus = unitMQ.getLEDState();
68
69      // Retrieve validity tag indicating whether MQ sensor data is reliable.
70      validTags = unitMQ.getValidTags();
71
72      // Get the firmware version from the sensor.
73      firmwareVersion = unitMQ.getFirmwareVersion();
74
75      // Always read NTC-related sensor data regardless of MQ data validity.
76      ntcAdc12bit      = unitMQ.getNTCADC12bit();                  // Raw ADC for NTC sensor
77      referenceVoltage = unitMQ.getReferenceVoltage();             // Reference voltage for ADC
78      ntcVoltage       = unitMQ.getNTCVoltage();                   // Calculated voltage across NTC
79      ntcResistance    = unitMQ.getNTCResistance();                // Calculated resistance of NTC th
80      temperature      = unitMQ.getNTCTemperature(ntcResistance);  // Convert resistance to temperatu
81
82      Serial.println("======== Unit MQ Status ========");
83      Serial.printf("Firmware Version   : %d\n", firmwareVersion);
84      Serial.printf("Heating Mode       : %s\n", heatMode == HEAT_MODE_CONTINUOUS ? "CONTINUOUS" : "SI
85      Serial.printf("LED Status         : %s\n", ledStatus == LED_WORK_STATUS_ON ? "ON" : "OFF");
86
87      // Print detailed NTC sensor parameters
88      Serial.printf("NTC ADC (12-bit)   : %u\n", ntcAdc12bit);
89      Serial.printf("Reference Voltage  : %u mV\n", referenceVoltage);
90      Serial.printf("NTC Voltage        : %u mV\n", ntcVoltage);
91      Serial.printf("NTC Resistance     : %u Ohm\n", ntcResistance);
92      Serial.printf("Temperature        : %.2f °C\n", temperature);
93
94      M5.Display.clear(TFT_WHITE);
95      M5.Display.drawCenterString("Unit MQ Status", M5.Display.width()/2, 0);
96      M5.Display.setCursor(0, 20);
97      M5.Display.printf("Firmware Version : %d\n", firmwareVersion);
98      M5.Display.printf("Heating Mode     : %s\n", heatMode == HEAT_MODE_CONTINUOUS ? "CONTINUOUS" : '
99      M5.Display.printf("LED Status       : %s\n", ledStatus == LED_WORK_STATUS_ON ? "ON" : "OFF");
100     M5.Display.printf("NTC ADC (12-bit) : %u\n", ntcAdc12bit);
101     M5.Display.printf("Reference Voltage: %u mV\n", referenceVoltage);
102     M5.Display.printf("NTC Voltage      : %u mV\n", ntcVoltage);
103     M5.Display.printf("NTC Resistance   : %u Ohm\n", ntcResistance);
```

```
    M5.Display.printf("Temperature      : %.2f C\n", temperature);

    // If MQ sensor data is valid, read and print MQ ADC and voltage values.
    if (validTags == VALID_TAG_VALID) {
        mqAdc12bit = unitMQ.getMQADC12bit();
        mqVoltage  = unitMQ.getMQVoltage();

        Serial.printf("Valid Tags         : 0x%02X (MQ data valid)\n", validTags);
        Serial.printf("MQ ADC (12-bit)    : %u\n", mqAdc12bit);
        Serial.printf("MQ Voltage         : %u mV\n", mqVoltage);

        M5.Display.printf("Valid Tags : 0x%02X (valid)\n", validTags);
        M5.Display.printf("MQ ADC (12-bit)  : %u\n", mqAdc12bit);
        M5.Display.printf("MQ Voltage       : %u mV\n", mqVoltage);
    } else {
        // If MQ data is invalid, log a warning and skip readings.
        Serial.printf("Valid Tags         : 0x%02X (MQ data invalid)\n", validTags);
        Serial.println("[WARNING] MQ data invalid. Skipping MQ readings.");

        M5.Display.printf("Valid Tags : 0x%02X (invalid)\n", validTags);
        M5.Display.println("[WARNING] MQ data invalid.\nSkipping MQ readings.");
    }

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

    delay(1000);  // Wait for 1 second before next update
}
```

4. コンパイルと書き込み

- デバイスポートを選択（詳細については プログラムのコンパイルと書き込み を参照）、Arduino IDE 左上のコンパイル & アップロードボタンをクリックし、コンパイルと書き込みが完了するのを待ちます。



5. 測定結果

Unit MQ の 2 つのモードの動作は下記の通りです:

- 連続加熱モード

- 間欠加熱モード