PaperMono LoRa Arduino チュートリアル

PaperMono LoRa のサンプルプログラムです。

サンプルプログラム

ビルド要件

- M5Stack ボードマネージャのバージョン >= 3.3.9
- ボードオプション = `M5PaperMono`
- M5Unified ライブラリのバージョン >= 0.2.20
- M5GFX ライブラリのバージョン >= 0.2.27
- RadioLib ライブラリのバージョン >= 7.2.1
- M5PM1 ライブラリのバージョン >= 1.0.7
- M5IOE1 ライブラリのバージョン >= 1.0.9

> **注意**
>
> PaperMono に搭載された SX1262 は、専用の SPI ピン `SCK=39`、`MISO=40`、`MOSI=38`、`NSS=41` を使用します。RadioLib の `Module` を作成するときは、`loraSpi` と対応する `SPISettings` を渡し、`radio.begin()` を呼び出す前に `loraSpi.begin(kSckPin, kMisoPin, kMosiPin, kNssPin)` を実行する必要があります。これらの設定を省略すると、RadioLib がデフォルトの SPI ピンを使用し、SX1262 と通信できず初期化に失敗する場合があります。代表的なエラーコードは `RADIOLIB_ERR_CHIP_NOT_FOUND`（`-2`）です。

PaperMono の LoRa 電源イネーブル信号 `LoRa_EN` は M5PM1 `GPIO2` に接続されています。SX1262 のリセット信号 `SX_NRST` とアンテナスイッチ信号 `SX_ANT_SW` は、それぞれ M5IOE1 `GPIO10`（`M5IOE1_PIN_10`）と `GPIO2`（`M5IOE1_PIN_2`）に接続されています。サンプルプログラムは、まず M5PM1 `GPIO2` を HIGH にして LoRa 電源を有効にし、M5IOE1 `GPIO10` で SX1262 をリセットした後、送受信状態に応じて M5IOE1 `GPIO2` でアンテナスイッチの経路を制御します。これらのピンが正しく設定されていない場合、SX1262 が正常に初期化または送受信できない可能性があります。

送信側

このプログラムは M5PM1 を制御して LoRa の電源とアンテナスイッチを有効にし、SX1262 をリセットします。SX1262 の初期化後、連番付きの `Hello PaperMono #X` メッセージを 3 秒ごとに送信し、送信結果を画面とシリアルモニターに表示します。

cpp

```cpp
#include <Arduino.h>
#include <M5IOE1.h>
#include <M5PM1.h>
#include <M5Unified.h>
#include <RadioLib.h>
#include <SPI.h>

namespace {

constexpr float kFrequencyMhz = 868.0f;
constexpr float kBandwidthKhz = 125.0f;
constexpr uint8_t kSpreadingFactor = 12;
constexpr uint8_t kCodingRate = 5;
constexpr uint8_t kSyncWord = 0x34;
constexpr int8_t kTxPowerDbm = 22;
constexpr uint16_t kPreambleLength = 20;
constexpr uint32_t kSendIntervalMs = 3000;

constexpr int kMosiPin = 38;
constexpr int kMisoPin = 40;
constexpr int kSckPin = 39;
constexpr int kNssPin = 41;
constexpr int kIrqPin = 5;
constexpr int kBusyPin = 21;
constexpr uint32_t kSpiFrequencyHz = 8000000;

constexpr auto kResetPin = M5IOE1_PIN_10;
constexpr auto kAntennaSwitchPin = M5IOE1_PIN_2;

SPIClass loraSpi(HSPI);
SX1262 radio = new Module(
    kNssPin, kIrqPin, RADIOLIB_NC, kBusyPin, loraSpi,
    SPISettings(kSpiFrequencyHz, MSBFIRST, SPI_MODE0));
M5Canvas canvas(&M5.Display);
M5PM1 pm1;
M5IOE1 ioe1;

volatile bool transmissionFinished = false;
int transmissionState = RADIOLIB_ERR_NONE;
uint32_t messageNumber = 1;

void IRAM_ATTR onTransmissionFinished()
{
    transmissionFinished = true;
}

void drawScreen(const String& state, const String& message,
                const String& detail = "")
{
    canvas.fillSprite(TFT_WHITE);
    canvas.setTextDatum(middle_center);
    canvas.setTextColor(TFT_BLACK, TFT_WHITE);

    canvas.setFont(&fonts::FreeSansBold24pt7b);
    canvas.drawString("LoRa TX", 240, 95);
```

```
    canvas.drawFastHLine(40, 150, 400, TFT_BLACK);
    canvas.setFont(&fonts::FreeSansBold18pt7b);
    canvas.drawString(state, 240, 220);

    canvas.setFont(&fonts::FreeSansBold12pt7b);
    canvas.drawString(message, 240, 330);
    canvas.setFont(&fonts::FreeSans12pt7b);
    canvas.drawString(detail, 240, 405);
    canvas.drawString("868.0 MHz  |  SF12  |  16 dBm", 240, 665);

    canvas.pushSprite(0, 0);
}

[[noreturn]] void stopWithError(const String& message, int code)
{
    Serial.printf("%s, code: %d\n", message.c_str(), code);
    drawScreen("ERROR", message, "Code: " + String(code));
    while (true) {
        delay(1000);
    }
}

bool enableLoRaHardware()
{
    const m5pm1_err_t pm1Error =
        pm1.begin(&M5.In_I2C, M5PM1_DEFAULT_ADDR, M5PM1_I2C_FREQ_100K);
    if (pm1Error != M5PM1_OK) {
        Serial.printf("M5PM1 init failed, code: %d\n", pm1Error);
        return false;
    }

    if (pm1.gpioSetFunc(M5PM1_GPIO_NUM_2, M5PM1_GPIO_FUNC_GPIO) != M5PM1_OK ||
        pm1.gpioSet(M5PM1_GPIO_NUM_2, M5PM1_GPIO_MODE_OUTPUT, HIGH,
                    M5PM1_GPIO_PULL_NONE,
                    M5PM1_GPIO_DRIVE_PUSHPULL) != M5PM1_OK) {
        Serial.println("LoRa power enable failed");
        return false;
    }
    Serial.println("LoRa power: enabled");

    const m5ioe1_err_t ioeError = ioe1.begin(
        &M5.In_I2C, M5IOE1_DEFAULT_ADDR_2, M5IOE1_I2C_FREQ_100K);
    if (ioeError != M5IOE1_OK) {
        Serial.printf("M5IOE1 init failed, code: %d\n", ioeError);
        return false;
    }

    ioe1.pinMode(kResetPin, OUTPUT);
    ioe1.pinMode(kAntennaSwitchPin, OUTPUT);
    if (ioe1.setDriveMode(kResetPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK ||
        ioe1.setDriveMode(kAntennaSwitchPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK) {
        Serial.println("LoRa IO drive setup failed");
        return false;
    }
```

```
    delay(200);
    m5ioe1_err_t ioeWriteError = M5IOE1_OK;
    ioe1.digitalWriteWithRes(kResetPin, LOW, &ioeWriteError);
    delay(100);
    ioe1.digitalWriteWithRes(kResetPin, HIGH, &ioeWriteError);
    delay(200);
    if (ioeWriteError != M5IOE1_OK) {
        Serial.printf("LoRa reset failed, code: %d\n", ioeWriteError);
        return false;
    }
    pinMode(kBusyPin, INPUT);
    Serial.printf("LoRa reset: released, BUSY=%d\n", digitalRead(kBusyPin));
    return true;
}

void startPacket()
{
    String message = "Hello PaperMono #" + String(messageNumber);
    Serial.printf("[LoRa TX] Sending: %s\n", message.c_str());

    ioe1.digitalWrite(kAntennaSwitchPin, HIGH);
    transmissionState = radio.startTransmit(message);
    if (transmissionState != RADIOLIB_ERR_NONE) {
        transmissionFinished = true;
    }
}

}  // namespace

void setup()
{
    Serial.begin(115200);
    delay(100);
    Serial.println("PaperMONO LoRa sender");

    auto config = M5.config();
    config.clear_display = false;
    config.internal_rtc = false;
    config.internal_imu = false;
    config.internal_mic = false;
    M5.begin(config);

    M5.Display.setRotation(0);
    M5.Display.setBrightness(160);
    M5.Display.setEpdMode(epd_mode_t::epd_fast);
    canvas.setColorDepth(16);
    if (!canvas.createSprite(M5.Display.width(), M5.Display.height())) {
        Serial.println("Display canvas allocation failed");
        while (true) delay(1000);
    }
    drawScreen("INITIALIZING", "SX1262");

    if (!enableLoRaHardware()) {
        stopWithError("LoRa power init failed", -1);
    }
```

```
166
167     loraSpi.begin(kSckPin, kMisoPin, kMosiPin, kNssPin);
168     Serial.println("LoRa SPI: HSPI/SPI3 at 8 MHz");
169     const int state = radio.begin(kFrequencyMhz, kBandwidthKhz,
170                                   kSpreadingFactor, kCodingRate, kSyncWord,
171                                   kTxPowerDbm, kPreambleLength, 3.0f, true);
172     if (state != RADIOLIB_ERR_NONE) {
173         stopWithError("SX1262 init failed", state);
174     }
175
176     radio.setCurrentLimit(140);
177     radio.setPacketSentAction(onTransmissionFinished);
178     drawScreen("READY", "Sending every 3 seconds");
179     startPacket();
180 }
181
182 void loop()
183 {
184     if (!transmissionFinished) {
185         delay(5);
186         return;
187     }
188
189     transmissionFinished = false;
190     const String message = "Hello PaperMono #" + String(messageNumber);
191     if (transmissionState == RADIOLIB_ERR_NONE) {
192         Serial.printf("[LoRa TX] Sent: %s\n", message.c_str());
193         drawScreen("SENT", message,
194                    "Message " + String(messageNumber));
195     } else {
196         Serial.printf("[LoRa TX] Failed, code: %d\n", transmissionState);
197         drawScreen("SEND FAILED", message,
198                    "Code: " + String(transmissionState));
199     }
200
201     radio.finishTransmit();
202     delay(kSendIntervalMs);
203     ++messageNumber;
204     startPacket();
205 }
```

受信側

以下のプログラムを別の PaperMono に書き込みます。受信パラメータは送信側と一致させてください。

```cpp
```

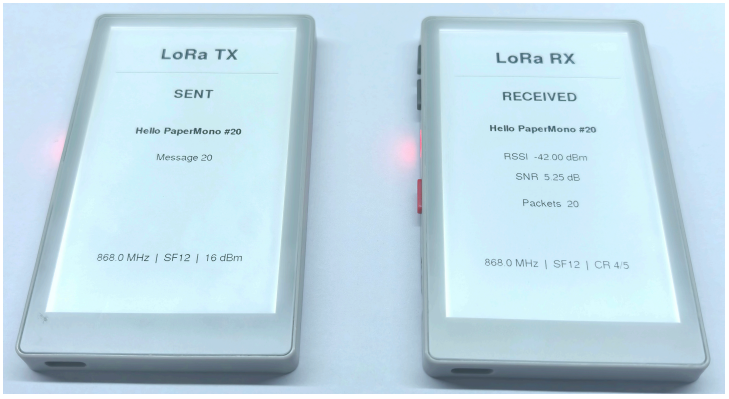
```cpp
#include <Arduino.h>
#include <M5IOE1.h>
#include <M5PM1.h>
#include <M5Unified.h>
#include <RadioLib.h>
#include <SPI.h>

namespace {

constexpr float kFrequencyMhz = 868.0f;
constexpr float kBandwidthKhz = 125.0f;
constexpr uint8_t kSpreadingFactor = 12;
constexpr uint8_t kCodingRate = 5;
constexpr uint8_t kSyncWord = 0x34;
constexpr int8_t kTxPowerDbm = 22;
constexpr uint16_t kPreambleLength = 20;

constexpr int kMosiPin = 38;
constexpr int kMisoPin = 40;
constexpr int kSckPin = 39;
constexpr int kNssPin = 41;
constexpr int kIrqPin = 5;
constexpr int kBusyPin = 21;
constexpr uint32_t kSpiFrequencyHz = 8000000;

constexpr auto kResetPin = M5IOE1_PIN_10;
constexpr auto kAntennaSwitchPin = M5IOE1_PIN_2;

SPIClass loraSpi(HSPI);
SX1262 radio = new Module(
    kNssPin, kIrqPin, RADIOLIB_NC, kBusyPin, loraSpi,
    SPISettings(kSpiFrequencyHz, MSBFIRST, SPI_MODE0));
M5Canvas canvas(&M5.Display);
M5PM1 pm1;
M5IOE1 ioe1;

volatile bool packetReceived = false;
uint32_t receivedCount = 0;

void IRAM_ATTR onPacketReceived()
{
    packetReceived = true;
}

void drawScreen(const String& state, const String& message,
                const String& rssi = "", const String& snr = "",
                const String& detail = "")
{
    canvas.fillSprite(TFT_WHITE);
    canvas.setTextDatum(middle_center);
    canvas.setTextColor(TFT_BLACK, TFT_WHITE);

    canvas.setFont(&fonts::FreeSansBold24pt7b);
    canvas.drawString("LoRa RX", 240, 95);

```

```
    canvas.drawFastHLine(40, 150, 400, TFT_BLACK);
    canvas.setFont(&fonts::FreeSansBold18pt7b);
    canvas.drawString(state, 240, 215);

    canvas.setFont(&fonts::FreeSansBold12pt7b);
    canvas.drawString(message, 240, 310);
    canvas.setFont(&fonts::FreeSans12pt7b);
    canvas.drawString(rssi, 240, 390);
    canvas.drawString(snr, 240, 445);
    canvas.drawString(detail, 240, 515);
    canvas.drawString("868.0 MHz  |  SF12  |  CR 4/5", 240, 665);

    canvas.pushSprite(0, 0);
}

[[noreturn]] void stopWithError(const String& message, int code)
{
    Serial.printf("%s, code: %d\n", message.c_str(), code);
    drawScreen("ERROR", message, "Code: " + String(code));
    while (true) {
        delay(1000);
    }
}

bool enableLoRaHardware()
{
    const m5pm1_err_t pm1Error =
        pm1.begin(&M5.In_I2C, M5PM1_DEFAULT_ADDR, M5PM1_I2C_FREQ_100K);
    if (pm1Error != M5PM1_OK) {
        Serial.printf("M5PM1 init failed, code: %d\n", pm1Error);
        return false;
    }

    if (pm1.gpioSetFunc(M5PM1_GPIO_NUM_2, M5PM1_GPIO_FUNC_GPIO) != M5PM1_OK ||
        pm1.gpioSet(M5PM1_GPIO_NUM_2, M5PM1_GPIO_MODE_OUTPUT, HIGH,
                    M5PM1_GPIO_PULL_NONE,
                    M5PM1_GPIO_DRIVE_PUSHPULL) != M5PM1_OK) {
        Serial.println("LoRa power enable failed");
        return false;
    }
    Serial.println("LoRa power: enabled");

    const m5ioe1_err_t ioeError = ioe1.begin(
        &M5.In_I2C, M5IOE1_DEFAULT_ADDR_2, M5IOE1_I2C_FREQ_100K);
    if (ioeError != M5IOE1_OK) {
        Serial.printf("M5IOE1 init failed, code: %d\n", ioeError);
        return false;
    }

    ioe1.pinMode(kResetPin, OUTPUT);
    ioe1.pinMode(kAntennaSwitchPin, OUTPUT);
    if (ioe1.setDriveMode(kResetPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK ||
        ioe1.setDriveMode(kAntennaSwitchPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK) {
        Serial.println("LoRa IO drive setup failed");
        return false;
```

```
    }
    ioe1.digitalWrite(kAntennaSwitchPin, LOW);
    delay(200);
    m5ioe1_err_t ioeWriteError = M5IOE1_OK;
    ioe1.digitalWriteWithRes(kResetPin, LOW, &ioeWriteError);
    delay(100);
    ioe1.digitalWriteWithRes(kResetPin, HIGH, &ioeWriteError);
    delay(200);
    if (ioeWriteError != M5IOE1_OK) {
        Serial.printf("LoRa reset failed, code: %d\n", ioeWriteError);
        return false;
    }
    pinMode(kBusyPin, INPUT);
    Serial.printf("LoRa reset: released, BUSY=%d\n", digitalRead(kBusyPin));
    return true;
}

void resumeReceive()
{
    ioe1.digitalWrite(kAntennaSwitchPin, LOW);
    const int state = radio.startReceive();
    if (state != RADIOLIB_ERR_NONE) {
        stopWithError("Listen failed", state);
    }
}

}  // namespace

void setup()
{
    Serial.begin(115200);
    delay(100);
    Serial.println("PaperMONO LoRa receiver");

    auto config = M5.config();
    config.clear_display = false;
    config.internal_rtc = false;
    config.internal_imu = false;
    config.internal_mic = false;
    M5.begin(config);

    M5.Display.setRotation(0);
    M5.Display.setBrightness(160);
    M5.Display.setEpdMode(epd_mode_t::epd_fast);
    canvas.setColorDepth(16);
    if (!canvas.createSprite(M5.Display.width(), M5.Display.height())) {
        Serial.println("Display canvas allocation failed");
        while (true) delay(1000);
    }
    drawScreen("INITIALIZING", "SX1262");

    if (!enableLoRaHardware()) {
        stopWithError("LoRa power init failed", -1);
    }

```

```
    loraSpi.begin(kSckPin, kMisoPin, kMosiPin, kNssPin);
    Serial.println("LoRa SPI: HSPI/SPI3 at 8 MHz");
    const int state = radio.begin(kFrequencyMhz, kBandwidthKhz,
                                  kSpreadingFactor, kCodingRate, kSyncWord,
                                  kTxPowerDbm, kPreambleLength, 3.0f, true);
    if (state != RADIOLIB_ERR_NONE) {
        stopWithError("SX1262 init failed", state);
    }

    radio.setCurrentLimit(140);
    radio.setPacketReceivedAction(onPacketReceived);
    resumeReceive();
    Serial.println("[LoRa RX] Listening...");
    drawScreen("LISTENING", "Waiting for a packet");
}

void loop()
{
    if (!packetReceived) {
        delay(5);
        return;
    }

    packetReceived = false;
    String message;
    const int state = radio.readData(message);
    if (state == RADIOLIB_ERR_NONE) {
        ++receivedCount;
        const float rssi = radio.getRSSI();
        const float snr = radio.getSNR();
        const float frequencyError = radio.getFrequencyError();

        Serial.printf("[LoRa RX] Data: %s\n", message.c_str());
        Serial.printf("[LoRa RX] RSSI: %.2f dBm, SNR: %.2f dB\n", rssi, snr);
        Serial.printf("[LoRa RX] Frequency error: %.2f Hz\n", frequencyError);
        drawScreen("RECEIVED", message,
                   "RSSI  " + String(rssi, 2) + " dBm",
                   "SNR  " + String(snr, 2) + " dB",
                   "Packets  " + String(receivedCount));
    } else if (state == RADIOLIB_ERR_CRC_MISMATCH) {
        Serial.println("[LoRa RX] CRC mismatch");
        drawScreen("CRC ERROR", "Packet discarded");
    } else {
        Serial.printf("[LoRa RX] Read failed, code: %d\n", state);
        drawScreen("READ FAILED", "Code: " + String(state));
    }

    resumeReceive();
}
```

受信側は LoRa データを継続的に待ち受けます。データを受信すると、画面にメッセージ内容、累計受信数、RSSI、SNR が表示され、シリアルモニターには周波数誤差も出力されます。CRC エラーまたは読み取り失敗が発生した場合は対応する状態を表示し、その後自動的に受信を再開します。



ドライバライブラリ

- RadioLib