

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 `GPI02` 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 `GPI010` (`M5IOE1_PIN_10`) と `GPI02` (`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 `GPI02` を HIGH にして LoRa 電源を有効にし、M5IOE1 `GPI010` で SX1262 をリセットした後、送受信状態に応じて M5IOE1 `GPI02` 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

```

1  #include <Arduino.h>
2  #include <M5IOE1.h>
3  #include <M5PM1.h>
4  #include <M5Unified.h>
5  #include <RadioLib.h>
6  #include <SPI.h>
7
8  namespace {
9
10     constexpr float kFrequencyMhz = 868.0f;
11     constexpr float kBandwidthKhz = 125.0f;
12     constexpr uint8_t kSpreadingFactor = 12;
13     constexpr uint8_t kCodingRate = 5;
14     constexpr uint8_t kSyncWord = 0x34;
15     constexpr int8_t kTxPowerDbm = 22;
16     constexpr uint16_t kPreambleLength = 20;
17     constexpr uint32_t kSendIntervalMs = 3000;
18
19     constexpr int kMosiPin = 38;
20     constexpr int kMisoPin = 40;
21     constexpr int kSckPin = 39;
22     constexpr int kNssPin = 41;
23     constexpr int kIrqPin = 5;
24     constexpr int kBusyPin = 21;
25     constexpr uint32_t kSpiFrequencyHz = 8000000;
26
27     constexpr auto kResetPin = M5IOE1_PIN_10;
28     constexpr auto kAntennaSwitchPin = M5IOE1_PIN_2;
29
30     SPIClass loraSpi(HSPI);
31     SX1262 radio = new Module(
32         kNssPin, kIrqPin, RADIOLIB_NC, kBusyPin, loraSpi,
33         SPISettings(kSpiFrequencyHz, MSBFIRST, SPI_MODE0));
34     M5Canvas canvas(&M5.Display);
35     M5PM1 pm1;
36     M5IOE1 ioe1;
37
38     volatile bool transmissionFinished = false;
39     int transmissionState = RADIOLIB_ERR_NONE;
40     uint32_t messageNumber = 1;
41
42     void IRAM_ATTR onTransmissionFinished()
43     {
44         transmissionFinished = true;
45     }
46
47     void drawScreen(const String& state, const String& message,
48         const String& detail = "")
49     {
50         canvas.fillSprite(TFT_WHITE);
51         canvas.setTextDatum(middle_center);
52         canvas.setTextColor(TFT_BLACK, TFT_WHITE);
53
54         canvas.setFont(&fonts::FreeSansBold24pt7b);
55         canvas.drawString("LoRa TX", 240, 95);

```

```

56
57     canvas.drawFastHLine(40, 150, 400, TFT_BLACK);
58     canvas.setFont(&fonts::FreeSansBold18pt7b);
59     canvas.drawString(state, 240, 220);
60
61     canvas.setFont(&fonts::FreeSansBold12pt7b);
62     canvas.drawString(message, 240, 330);
63     canvas.setFont(&fonts::FreeSans12pt7b);
64     canvas.drawString(detail, 240, 405);
65     canvas.drawString("868.0 MHz | SF12 | 16 dBm", 240, 665);
66
67     canvas.pushSprite(0, 0);
68 }
69
70 [[noreturn]] void stopWithError(const String& message, int code)
71 {
72     Serial.printf("%s, code: %d\n", message.c_str(), code);
73     drawScreen("ERROR", message, "Code: " + String(code));
74     while (true) {
75         delay(1000);
76     }
77 }
78
79 bool enableLoRaHardware()
80 {
81     const m5pm1_err_t pm1Error =
82         pm1.begin(&M5.In_I2C, M5PM1_DEFAULT_ADDR, M5PM1_I2C_FREQ_100K);
83     if (pm1Error != M5PM1_OK) {
84         Serial.printf("M5PM1 init failed, code: %d\n", pm1Error);
85         return false;
86     }
87
88     if (pm1.gpioSetFunc(M5PM1_GPIO_NUM_2, M5PM1_GPIO_FUNC_GPIO) != M5PM1_OK ||
89         pm1.gpioSet(M5PM1_GPIO_NUM_2, M5PM1_GPIO_MODE_OUTPUT, HIGH,
90                     M5PM1_GPIO_PULL_NONE,
91                     M5PM1_GPIO_DRIVE_PUSHPULL) != M5PM1_OK) {
92         Serial.println("LoRa power enable failed");
93         return false;
94     }
95     Serial.println("LoRa power: enabled");
96
97     const m5ioe1_err_t ioeError = ioe1.begin(
98         &M5.In_I2C, M5IOE1_DEFAULT_ADDR_2, M5IOE1_I2C_FREQ_100K);
99     if (ioeError != M5IOE1_OK) {
100         Serial.printf("M5IOE1 init failed, code: %d\n", ioeError);
101         return false;
102     }
103
104     ioe1.pinMode(kResetPin, OUTPUT);
105     ioe1.pinMode(kAntennaSwitchPin, OUTPUT);
106     if (ioe1.setDriveMode(kResetPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK ||
107         ioe1.setDriveMode(kAntennaSwitchPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK) {
108         Serial.println("LoRa IO drive setup failed");
109         return false;
110     }

```

```

111     delay(200);
112     m5ioe1_err_t ioeWriteError = M5IOE1_OK;
113     ioe1.digitalWriteWithRes(kResetPin, LOW, &ioeWriteError);
114     delay(100);
115     ioe1.digitalWriteWithRes(kResetPin, HIGH, &ioeWriteError);
116     delay(200);
117     if (ioeWriteError != M5IOE1_OK) {
118         Serial.printf("LoRa reset failed, code: %d\n", ioeWriteError);
119         return false;
120     }
121     pinMode(kBusyPin, INPUT);
122     Serial.printf("LoRa reset: released, BUSY=%d\n", digitalRead(kBusyPin));
123     return true;
124 }
125
126 void startPacket()
127 {
128     String message = "Hello PaperMono #" + String(messageNumber);
129     Serial.printf("[LoRa TX] Sending: %s\n", message.c_str());
130
131     ioe1.digitalWrite(kAntennaSwitchPin, HIGH);
132     transmissionState = radio.startTransmit(message);
133     if (transmissionState != RADIOLIB_ERR_NONE) {
134         transmissionFinished = true;
135     }
136 }
137
138 } // namespace
139
140 void setup()
141 {
142     Serial.begin(115200);
143     delay(100);
144     Serial.println("PaperMONO LoRa sender");
145
146     auto config = M5.config();
147     config.clear_display = false;
148     config.internal_rtc = false;
149     config.internal_imu = false;
150     config.internal_mic = false;
151     M5.begin(config);
152
153     M5.Display.setRotation(0);
154     M5.Display.setBrightness(160);
155     M5.Display.setEpdMode(epd_mode_t::epd_fast);
156     canvas.setColorDepth(16);
157     if (!canvas.createSprite(M5.Display.width(), M5.Display.height())) {
158         Serial.println("Display canvas allocation failed");
159         while (true) delay(1000);
160     }
161     drawScreen("INITIALIZING", "SX1262");
162
163     if (!enableLoRaHardware()) {
164         stopWithError("LoRa power init failed", -1);
165     }

```

```

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

```

1  #include <Arduino.h>
2  #include <M5IOE1.h>
3  #include <M5PM1.h>
4  #include <M5Unified.h>
5  #include <RadioLib.h>
6  #include <SPI.h>
7
8  namespace {
9
10 constexpr float kFrequencyMhz = 868.0f;
11 constexpr float kBandwidthKhz = 125.0f;
12 constexpr uint8_t kSpreadingFactor = 12;
13 constexpr uint8_t kCodingRate = 5;
14 constexpr uint8_t kSyncWord = 0x34;
15 constexpr int8_t kTxPowerDbm = 22;
16 constexpr uint16_t kPreambleLength = 20;
17
18 constexpr int kMosiPin = 38;
19 constexpr int kMisoPin = 40;
20 constexpr int kSckPin = 39;
21 constexpr int kNssPin = 41;
22 constexpr int kIrqPin = 5;
23 constexpr int kBusyPin = 21;
24 constexpr uint32_t kSpiFrequencyHz = 8000000;
25
26 constexpr auto kResetPin = M5IOE1_PIN_10;
27 constexpr auto kAntennaSwitchPin = M5IOE1_PIN_2;
28
29 SPIClass loraSpi(HSPI);
30 SX1262 radio = new Module(
31     kNssPin, kIrqPin, RADIOLIB_NC, kBusyPin, loraSpi,
32     SPISettings(kSpiFrequencyHz, MSBFIRST, SPI_MODE0));
33 M5Canvas canvas(&M5.Display);
34 M5PM1 pm1;
35 M5IOE1 ioe1;
36
37 volatile bool packetReceived = false;
38 uint32_t receivedCount = 0;
39
40 void IRAM_ATTR onPacketReceived()
41 {
42     packetReceived = true;
43 }
44
45 void drawScreen(const String& state, const String& message,
46     const String& rssi = "", const String& snr = "",
47     const String& detail = "")
48 {
49     canvas.fillSprite(TFT_WHITE);
50     canvas.setTextDatum(middle_center);
51     canvas.setTextColor(TFT_BLACK, TFT_WHITE);
52
53     canvas.setFont(&fonts::FreeSansBold24pt7b);
54     canvas.drawString("LoRa RX", 240, 95);
55

```

```

56 canvas.drawFastHLine(40, 150, 400, TFT_BLACK);
57 canvas.setFont(&font::FreeSansBold18pt7b);
58 canvas.drawString(state, 240, 215);
59
60 canvas.setFont(&font::FreeSansBold12pt7b);
61 canvas.drawString(message, 240, 310);
62 canvas.setFont(&font::FreeSans12pt7b);
63 canvas.drawString(rssi, 240, 390);
64 canvas.drawString(snr, 240, 445);
65 canvas.drawString(detail, 240, 515);
66 canvas.drawString("868.0 MHz | SF12 | CR 4/5", 240, 665);
67
68 canvas.pushSprite(0, 0);
69 }
70
71 [[noreturn]] void stopWithError(const String& message, int code)
72 {
73     Serial.printf("%s, code: %d\n", message.c_str(), code);
74     drawScreen("ERROR", message, "Code: " + String(code));
75     while (true) {
76         delay(1000);
77     }
78 }
79
80 bool enableLoRaHardware()
81 {
82     const m5pm1_err_t pm1Error =
83         pm1.begin(&M5.In_I2C, M5PM1_DEFAULT_ADDR, M5PM1_I2C_FREQ_100K);
84     if (pm1Error != M5PM1_OK) {
85         Serial.printf("M5PM1 init failed, code: %d\n", pm1Error);
86         return false;
87     }
88
89     if (pm1.gpioSetFunc(M5PM1_GPIO_NUM_2, M5PM1_GPIO_FUNC_GPIO) != M5PM1_OK ||
90         pm1.gpioSet(M5PM1_GPIO_NUM_2, M5PM1_GPIO_MODE_OUTPUT, HIGH,
91                     M5PM1_GPIO_PULL_NONE,
92                     M5PM1_GPIO_DRIVE_PUSHPULL) != M5PM1_OK) {
93         Serial.println("LoRa power enable failed");
94         return false;
95     }
96     Serial.println("LoRa power: enabled");
97
98     const m5ioe1_err_t ioeError = ioe1.begin(
99         &M5.In_I2C, M5IOE1_DEFAULT_ADDR_2, M5IOE1_I2C_FREQ_100K);
100     if (ioeError != M5IOE1_OK) {
101         Serial.printf("M5IOE1 init failed, code: %d\n", ioeError);
102         return false;
103     }
104
105     ioe1.pinMode(kResetPin, OUTPUT);
106     ioe1.pinMode(kAntennaSwitchPin, OUTPUT);
107     if (ioe1.setDriveMode(kResetPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK ||
108         ioe1.setDriveMode(kAntennaSwitchPin, M5IOE1_DRIVE_PUSHPULL) != M5IOE1_OK) {
109         Serial.println("LoRa IO drive setup failed");
110         return false;

```

```

111     }
112     ioel.digitalWrite(kAntennaSwitchPin, LOW);
113     delay(200);
114     m5ioe1_err_t ioeWriteError = M5IOE1_OK;
115     ioel.digitalWriteWithRes(kResetPin, LOW, &ioeWriteError);
116     delay(100);
117     ioel.digitalWriteWithRes(kResetPin, HIGH, &ioeWriteError);
118     delay(200);
119     if (ioeWriteError != M5IOE1_OK) {
120         Serial.printf("LoRa reset failed, code: %d\n", ioeWriteError);
121         return false;
122     }
123     pinMode(kBusyPin, INPUT);
124     Serial.printf("LoRa reset: released, BUSY=%d\n", digitalRead(kBusyPin));
125     return true;
126 }
127
128 void resumeReceive()
129 {
130     ioel.digitalWrite(kAntennaSwitchPin, LOW);
131     const int state = radio.startReceive();
132     if (state != RADIOLIB_ERR_NONE) {
133         stopWithError("Listen failed", state);
134     }
135 }
136
137 } // namespace
138
139 void setup()
140 {
141     Serial.begin(115200);
142     delay(100);
143     Serial.println("PaperMONO LoRa receiver");
144
145     auto config = M5.config();
146     config.clear_display = false;
147     config.internal_rtc = false;
148     config.internal_imu = false;
149     config.internal_mic = false;
150     M5.begin(config);
151
152     M5.Display.setRotation(0);
153     M5.Display.setBrightness(160);
154     M5.Display.setEpdMode(epd_mode_t::epd_fast);
155     canvas.setColorDepth(16);
156     if (!canvas.createSprite(M5.Display.width(), M5.Display.height())) {
157         Serial.println("Display canvas allocation failed");
158         while (true) delay(1000);
159     }
160     drawScreen("INITIALIZING", "SX1262");
161
162     if (!enableLoRaHardware()) {
163         stopWithError("LoRa power init failed", -1);
164     }
165

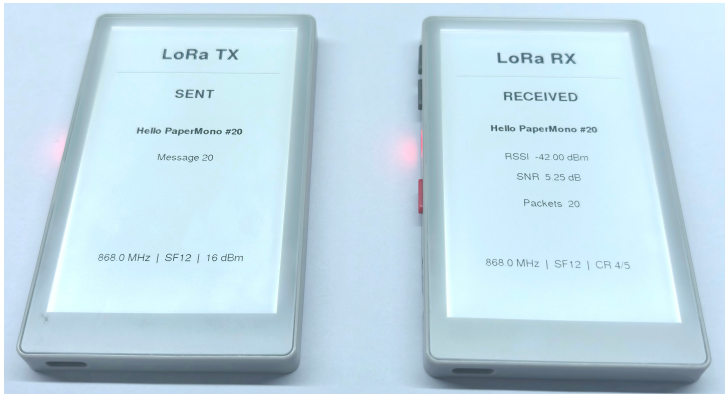
```

```

166 loraSpi.begin(kSckPin, kMisoPin, kMosiPin, kNssPin);
167 Serial.println("LoRa SPI: HSPI/SPI3 at 8 MHz");
168 const int state = radio.begin(kFrequencyMhz, kBandwidthKhz,
169                               kSpreadingFactor, kCodingRate, kSyncWord,
170                               kTxPowerDbm, kPreambleLength, 3.0f, true);
171 if (state != RADIOLIB_ERR_NONE) {
172     stopWithError("SX1262 init failed", state);
173 }
174
175 radio.setCurrentLimit(140);
176 radio.setPacketReceivedAction(onPacketReceived);
177 resumeReceive();
178 Serial.println("[LoRa RX] Listening...");
179 drawScreen("LISTENING", "Waiting for a packet");
180 }
181
182 void loop()
183 {
184     if (!packetReceived) {
185         delay(5);
186         return;
187     }
188
189     packetReceived = false;
190     String message;
191     const int state = radio.readData(message);
192     if (state == RADIOLIB_ERR_NONE) {
193         ++receivedCount;
194         const float rssi = radio.getRSSI();
195         const float snr = radio.getSNR();
196         const float frequencyError = radio.getFrequencyError();
197
198         Serial.printf("[LoRa RX] Data: %s\n", message.c_str());
199         Serial.printf("[LoRa RX] RSSI: %.2f dBm, SNR: %.2f dB\n", rssi, snr);
200         Serial.printf("[LoRa RX] Frequency error: %.2f Hz\n", frequencyError);
201         drawScreen("RECEIVED", message,
202                   "RSSI  " + String(rssi, 2) + " dBm",
203                   "SNR   " + String(snr, 2) + " dB",
204                   "Packets " + String(receivedCount));
205     } else if (state == RADIOLIB_ERR_CRC_MISMATCH) {
206         Serial.println("[LoRa RX] CRC mismatch");
207         drawScreen("CRC ERROR", "Packet discarded");
208     } else {
209         Serial.printf("[LoRa RX] Read failed, code: %d\n", state);
210         drawScreen("READ FAILED", "Code: " + String(state));
211     }
212
213     resumeReceive();
214 }

```

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](#)