

RISC-V コア Wi-Fi マイコン ESP32-C3 搭載 M5Stamp C3 で 無線 × IO 制御プログラミング

国野 亘

RISC-V コア Wi-Fi マイコン ESP32-C3 搭載 M5Stamp C3 で 無線 × IO 制御プログラミング

内容

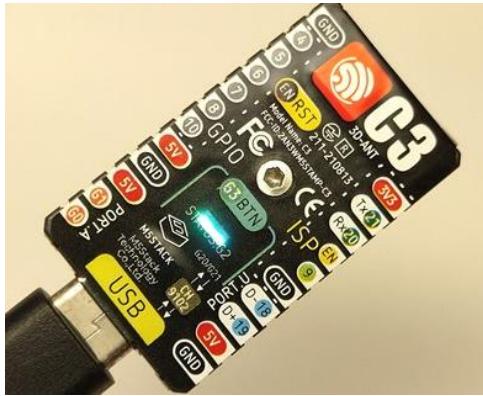
1 章 ESP32-C3 マイコンの特徴と開発環境の準備	6
Wi-Fi マイコン ESP32-C3 搭載 M5Stamp C3/C3U モジュール	6
M5Stamp C3 モジュールと C3U モジュールの違い	6
RISC-V 搭載 ESP32-C3 の特徴	7
ESP32-C3 の処理能力	8
ESP32-C3 用 Arduino 開発環境のセットアップ要件	9
準備① Arduino 開発環境をセットアップする	9
準備② 本書用のサンプル・プログラムのダウンロードする	11
準備③ ブレッド・ボードで IO 実験用ボードを作成する	11
準備④ IO 実験用ボードの動作確認	12
プログラム① IO 実験用ボードで IO 制御	12
2 章 ESP32-C3 マイコン無線 × IO 制御サンプル・プログラム集	14
プログラム① Wi-Fi コンシェルジェ照明担当（ワイヤレス LED）	14
HTTP サーバ搭載 LED 制御プログラム ex01_led.ino の内容	15
プログラム② Wi-Fi ボタン・送信機	17
アルカリ乾電池による長期間動作を実現するスリープ機能	17
LINE Notify 用のトークンを取得し、設定する	18
M5Stamp C3/C3U (2 台) を使ったリモート LED 制御	18
スリープ機能を使ったボタン送信プログラム ex02_sw.ino の内容	18
製作した機器の安全や信頼性に関する注意点	21
コラム 1：タクト・スイッチのチャタリング	21

3 章 ESP32-C3 マイコン無線×センサ活用サンプル・プログラム集	22
プログラム [3] Wi-Fi 照度計・送信機	22
Wi-Fi 照度計・送信機のハードウェアの製作方法	22
照度センサ NJL7502L の接続方法	22
超低消費電力を実現するディープ・スリープ機能	23
Ambient 用のチャネル ID とライトキーを取得し、設定する	24
センサ用プログラムの基本プログラム ex03_lum.ino の内容	24
プログラム [4] Wi-Fi コンシェルジェ掲示板担当・LCD/表示器	26
LCD/表示器の製作方法	26
LCD モジュール AE-AQM0802 の概要	27
LCD にメッセージを送信する方法	27
UDP ブロードキャストを受信する	28
HTTP サーバ搭載 LCD/表示器のプログラム ex04_lcd.ino の内容	28
プログラム [5] Wi-Fi 温湿度計・送信機	30
Wi-Fi 温湿度計・送信機のハードウェアの製作方法	30
温湿度センサ AE-SHT31/SHT35 の接続方法	30
最新の温度と湿度の LCD 表示と推移のグラフ表示	32
温湿度センサ用プログラム ex05_hum.ino の内容	32
コラム 2：温湿度センサ SHT3x 用 I ² C インタフェース部プログラム	34
4 章 ESP32-C3 マイコンを使用した応用システム・プログラム	35
応用例 [1] センサ・ネットワーク・システム	35
システムの概要	35
UDP センサ用モニタ・プログラム udp_monitor_chart.py の実行方法	35
プログラム udp_monitor_chart.py の便利機能	36
応用例 [2] 自分だけの My ホーム・オートメーション・システム	37
M5Stack 既製品とサンプル・プログラムで簡単にハードウェアを製作	37
プログラム srv_myhome.py の機能	38
My ホーム・オートメーション用プログラム srv_myhome.py の動作例	39

Appendix 試作に便利。ブレッド・ボード上に周辺回路を実装してみる	40
Wi-Fi コンシェルジェ [音声アナウンス担当]	40
Wi-Fi コンシェルジェ [カメラ担当]	40
むすび	41
参考文献	41
権利情報	41
履歴	41

RISC-V コア Wi-Fi マイコン ESP32-C3 搭載 M5Stamp C3 で無線×IO 制御プログラミング

国野 亘



RISC-V コア ESP32-C3 搭載 M5Stamp C3

切手サイズ (34mm × 20mm × 4.5mm) の M5Stamp C3 を使って無線を活用した IO 制御プログラムを作成する

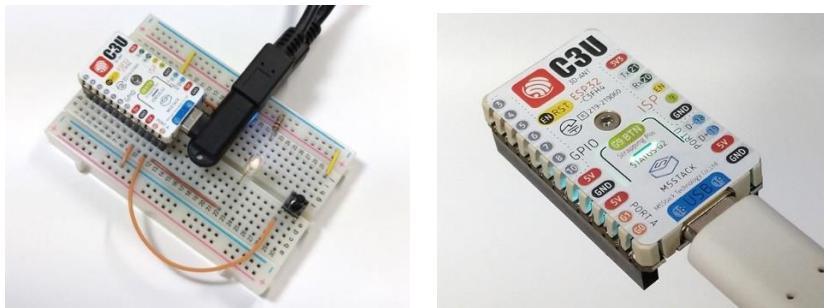
使用機器(第 1 章～3 章)：

- M5Stamp C3 または C3U Mate × 2
- ブレッド・ボード × 2
- LED, タクト・スイッチ
- 照度センサ NJL7502L
- 温湿度センサ SHT31 または SHT35
- 液晶モジュール AE-AQM0802
- (C3U の場合)USB シリアル変換モジュール AE-FT234X
- 単 3 アルカリ乾電池, ケースほか

Wi-Fi 搭載マイコン ESP32 シリーズに RISC-V コアの ESP32-C3 が登場しました。本書では ESP32-C3 を搭載した M5Stamp C3/C3U を用い、無線による IO 制御プログラムを作成します。

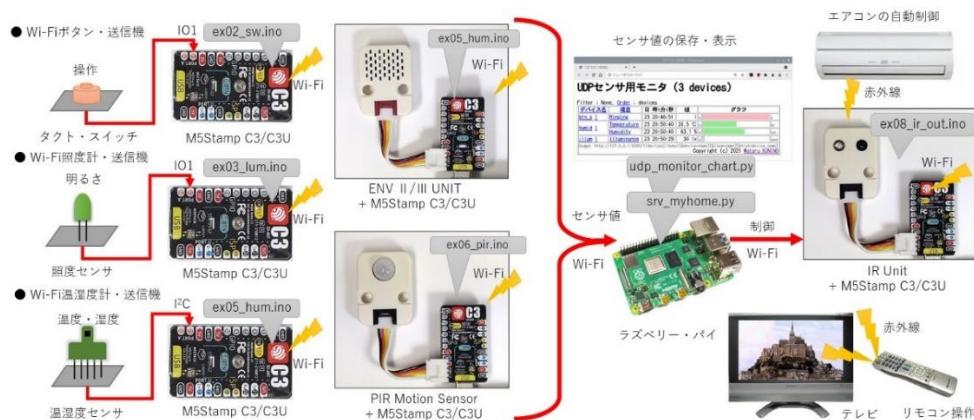
ESP32-C3 は、現 ESP8266 の後継にあたり、ESP32 の廉価版の位置づけ（後述）です。チップ単価は ESP32 の 66% 前後で売られており、IoT 機器を量産するときのコスト低減に有利です。廉価版といつても Wi-Fi を使った IO 制御が可能な IoT 用デバイスです。IO ポート数も多すぎないので、ブレッド・ボード上に実装したときに実験用の回路スペースを十分に確保することができます。予めピン・ソケットを半田づけしておけば、実証用の試作や学習用教材などに活用しやすいでしょう。

第 1 章では、ESP32-C3 による IO 実験用ボードをブレッド・ボードで製作し、第 2 章で無線による IO 制御プログラムの基礎を解説し、第 3 章でより多機能な機器を製作します。最後の 4 章では、無線 IO 制御の活用システムを試作します。1 章～3 章で製作した機器からの情報をラズベリー・パイで収集し、保存、可視化するシステムと、M5Stamp C3/C3U の Grove 互換端子に取り付けた拡張ユニットを活用したホーム・オートメーション・システムの製作例を紹介します。



M5Stamp C3U を使った IO 実験用ボードとモジュール単体での動作のようす

IO ポート数が多すぎないので、ブレッド・ボード上に実装したときに実験用の回路スペースを確保することができる（左）。単体でも動作する（右）



無線 IO 制御・活用システム(第 4 章)

センサ値をラズベリー・パイで収集し、保存、可視化するシステムやホーム・オートメーション・システムの製作例を紹介する

1章 ESP32-C3 マイコンの特徴と開発環境の準備

本章では、M5Stamp C3/C3U および ESP32-C3 マイコンの特徴と、従来の ESP8266 や ESP32 とのパフォーマンス比較例、ESP32-C3 用 Arduino 開発環境のセットアップ方法について説明します。

Wi-Fi マイコン ESP32-C3 搭載 M5Stamp C3/C3U モジュール

M5Stamp C3/C3U は、切手サイズ（34mm×20mm×4.5mm）の Wi-Fi マイコン ESP32-C3 内蔵モジュールです。写真 1-1 のようにアンテナ、フルカラーLED、ボタン、IO 端子を搭載しており、また国内の電波法に基づく認証番号も取得できているので、手っ取り早くプログラムを作成し、IoT 機器を製作することができます。

M5Stamp C3/C3U には 4MB フラッシュを内蔵した ESP32-C3-FN4 が用いられ、Wi-Fi 動作に必要な全機能が一つの QFN32 パッケージ内に集約されています。主な仕様を表 1-1 に示します。

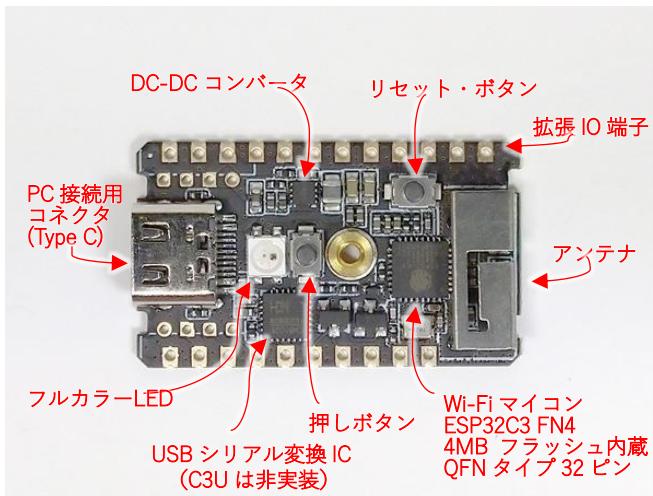


写真 1-1 M5Stamp C3 モジュールの内部

Wi-Fi マイコン ESP32-C3、DC-DC コンバータ、USB シリアル変換 IC といった IC と、アンテナ、フルカラーLED、ボタン、IO 端子などを搭載。マイコンの QFN32 パッケージ内に 4MB フラッシュを内蔵。ピン・ヘッダやピン・ソケットが付属する M5Stamp C3 Mate (写真 1-2) や C3U Mate は、製造元の M5Stack Technology Co., Ltd. から購入できるほか、国内では、スイッチ・サイエンス (<https://www.switch-science.com/>) から購入できます。

M5Stamp C3 モジュールと C3U モジュールの違い

M5Stamp C3 と C3U との違いは、本体の色と USB シリアル変換 IC の有無です。実験用には USB シリアル変換 IC を内蔵した M5Stamp C3 のほうが便利です。M5Stamp C3U でもマイコン内蔵の USB 機能が使えますが、開発時のシリアル・モニタを使った動作確認には、別途、USB シリアル変換モジュール (AE-FT234X、写真 1-3・右) または USB シリアル出力プログラムが必要です。

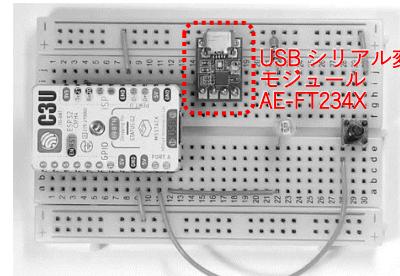


写真 1-3 M5Stamp C3(黒)と C3U(白)
開発時は USB シリアル変換 IC を内蔵した C3(左)が便利。C3U の場合は、USB シリアル変換モジュールが必要(右)

製品には予め LED とボタンの動作確認用プログラムが書き込まれています。入手したら USB ケーブルをパソコンまたは USB 出力用 AC アダプタに接続し、LED が点灯することを確認してください。また、中央のボタン G3 BTN を押して、LED の色が変化することを確認してください。

M5Stamp C3/C3U 本体の樹脂カバーは、リフロー装置による半田付け実装が可能な耐熱性能を有しております。また、ピン配列シールも耐熱素材で作られています。内部発熱などによって樹脂が発火するリスクの低減効果も期待できるでしょう（但し難燃性能は示されていない）。

表 1-1 M5Stamp C3 仕様表（※U は M5Stamp C3U）

項目	機能・性能（筆者による実測値を含む）
マイコン	ESP32-C3 32 ビット RISC-V シングル・コア・プロセッサ（最大動作周波数 160 MHz）
マイコン内蔵メモリ	384KB ROM, 400KB SRAM, 8KB RTC SRAM, 4MB フラッシュ
マイコン内蔵 Wi-Fi	2.4 GHz（バンド幅 20 MHz / 40 MHz），IEEE 802.11 b/g/n 最大データレート 150 Mbps
マイコン内蔵 Bluetooth	Bluetooth 5, Bluetooth mesh, データレート 125 Kbps / 500 Kbps / 1 Mbps / 2 Mbps
電源電圧・電流	5V・500mA, 通信時消費電流(実測) 約 57 mA, 待機時消費電流(実測) 約 0.4 mA / 約 0.3mA ^{※U}
ユーザ・インターフェース	押しボタン(IO3/IO9 ^{※U}) x 1, リセット・ボタン x 1, フルカラーLED SK6812(IO2) x 1
PC 接続インターフェース	Type C
内蔵アンテナ	2.4G 板金 3D 曲げ加工アンテナ（逆 F 給電）
汎用インターフェース	ADC, GPIO, SPI, UART, I2C, I2S, PWM, RMT, DMA, USBシリアル・ポート, TWAI
汎用 IO インタフェース	IO21, IO20, IO9, IO18, IO19, IO1, IO0, IO10, IO8, IO7, IO6, IO5, IO4, IO3 ^{※U}
USB シリアル変換 IC	M5Stamp C3 のみ内蔵（M5Stamp C3U は非搭載 ^{※U} ）
サイズ・質量	34 * 20 * 4.6mm, 3.8g
耐熱性能	245°C以下（最大 70 秒）・リフロー対応
電波法に基づく認証	工事設計認証 211-210813 (M5Stack Technology Co., Ltd, 令和 3 年 9 月 3 日) / 219-219060 ^{※U}

参考資料：https://docs.m5stack.com/en/core/stamp_c3 および stamp_c3u

RISC-V 搭載 ESP32-C3 の特徴

M5Stamp C3/C3U に搭載されている Wi-Fi マイコン ESP32-C3 の特徴について、説明します。

ESP32-C3 には、カリфорニア大学バークレイ校が開発を始めた RISC-V（リスク・ファイブ）と呼ばれるプロセッサが搭載されています。RISC-V は、オープン・ソース・ライセンスで提供されているので、従来の ESP シリーズで採用されていた Tensilica Xtensa プロセッサ・コアよりも低価格化が可能です。発売直後にもかかわらず、流通価格（執筆時点）は ESP32 の 66% 前後、ESP8266 と同等価格です。しかも、ESP32 と同様、台湾 TSMC 社の 40nm プロセスで生産することで、高速かつ高機能、低消費電力な Wi-Fi 搭載マイコンを実現しています。

従来の ESP シリーズの MPU 仕様との比較結果を表 1-2 に示します。ESP32-C3 は、コア数 1, FPU（浮動小数点数演算ユニット）なし、QFN 32 ピンなどから、ESP8266（ESP-WROOM-02）の後継にあたります。また、現行の ESP32 の後継には ESP32-S3 が用いされることになるでしょう。

表 1-2 ESP32-C3 の主な MPU 仕様（従来モデル ESP8266, ESP32, ESP32-S2 との比較）

マイコン	Bit	MPU コア	コア数	FPU	動作周波数	RAM	GPIO	パッケージ	参考単価
ESP32-C3	32	RISC-V	1	–	160 MHz	400 KB	22	QFN 32 5x5	約 133 円
ESP32-S3	32	Tensilica Xtensa LX7	2	○	240 MHz	512KB	45	QFN 56 7x7	—
ESP8266	32	Tensilica Xtensa LX106	1	–	160 MHz	160 KB	17	QFN 32 5x5	約 119 円
ESP32	32	Tensilica Xtensa LX6	2	○	240 MHz	520 KB	34	QFN 48 5x5	約 203 円

参考資料：<https://www.espressif.com/en/products/socs>, 単価は流通価格を参考にした

ESP32-C3 の処理能力

ESP32-C3 の処理速度に関するベンチマーク評価を行い、従来の Tensilica Xtensa プロセッサを搭載した ESP32 や ESP8266 と比較してみました。

表 1-3 に、整数演算速度、浮動小数点数演算速度、GPIO 制御処理速度、ADC (A/D コンバータ) の読み取り処理速度の測定結果の一例を示します。括弧内は ESP8266 との比較です。総合的なパフォーマンスは ESP8266 と ESP32 の中間あたりに位置するものと思います。

Wi-Fi マイコンとして最も基本となる整数演算速度については、ESP8266 の約 1.8 倍となる 172 DMIPS が得られました。動作周波数 240 MHz の ESP32 には及ばないものの、ESP32 の動作周波数を 160 MHz に換算した処理能力 (167 DMIPS) を上回っており、コア単体の構造は同レベルと思われます。ただし、デュアル・コアの ESP32 の場合は Wi-Fi の処理を別コアで行い、シングル・コアの ESP32-C3 は一つのコアで実行するため、音声やビデオのようなリアル・タイム処理は苦手です。

また、GPIO 制御や ADC (A/D コンバータ) の読み取り処理速度も向上していました。

一方、浮動小数点数演算では、ESP8266 よりも 15 ポイントほど劣る結果でした。ESP32-C3、ESP8266 ともにハードウェアによる浮動小数点数演算機能 (FPU) が搭載されていないので、もともと浮動小数点数演算が多いエッジ・コンピューティングには向いていません。浮動小数点演算の少ない IO 制御やセンサ値の単位変換程度の演算では何ら支障ないでしょう。また、ソフトウェアによる浮動小数点数演算速度は、今後のコンパイラやライブラリの進化で向上する場合もあります。

なお、測定結果は特定の機器や環境で測定した一例です。条件によって異なる結果となる場合があります。

表 1-3 ESP32-C3 の実測処理速度の一例 (括弧内は従来モデル ESP8266、ESP32 との比較)

項目	測定方式	ESP32-C3	ESP8266	ESP32
整数演算	Dhrystone	172.35 DMIPS (1.79)	96.26 DMIPS (1.00)	250.82 DMIPS (2.61)
小数演算	Whetstone	11.56 MWIPS (0.85)	13.68 MWIPS (1.00)	323.62 MWIPS (23.66)
GPIO 処理	digitalWrite	2.86 meg/s (1.34)	2.14 meg/s (1.00)	7.88 meg/s (3.68)
ADC 処理	analogRead	20.69 kilo/s (1.69)	12.26 kilo/s (1.00)	15.88 kilo/s (1.30)

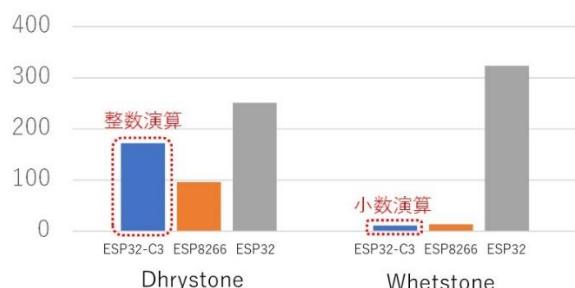


図 1-1 ESP シリーズの整数演算速度の比較結果
ESP32-C3 は、従来モデル ESP8266 の約 1.8 倍となる 172 DMIPS の整数演算速度を得た。ESP32 の動作周波数 (240 MHz) を 160 MHz に換算した処理能力 167 DMIPS を上回る。ESP32-C3 の GPIO 制御の処理速度と ADC (A/D コンバータ) の読み取り処理速度についても、従来モデル ESP8266 より向上した

ESP32-C3 用 Arduino 開発環境のセットアップ要件

M5Stamp C3/C3U のプログラムを開発するのに必要な、ESP32-C3 用の Arduino 開発環境をセットアップし、本書用のサンプル・プログラムをダウンロードする方法について説明します。

開発用のパソコンには、Windows や macOS, Linux を搭載した PC (ラズベリー・パイを含む) が使用できます。ただし、書き込みツール esptool の動作要件の問題で、作成したソフトウェアを M5Stamp C3/C3U に書き込めない場合があります。

本書執筆時点では、古い PC などで使用されている 32 ビット版 Windows 10 や、Python 2 がインストールされていない最新の Linux 環境で動作しない問題がありました。32 ビット版 Windows 10 で動作する ESP32-C3 対応の esptool のバイナリは配布されていないので、自分で esptool をビルドする必要がありました。また、Linux 環境では Python 2 や pip 2, pip 2 による pyserial のインストールが必要な場合があります。Windows 11 や 64 ビット版の Widows 10 であれば、問題ないでしょう。

準備 1 Arduino 開発環境をセットアップする

図 1-2 は、Windows 11 (または 64 ビット版 Windows 10) の場合のセットアップ手順です。他の環境でも似たような手順でインストールできます。

- ① Arduino IDE のインストール：下記の URL にアクセスして Arduino IDE をダウンロードし、PC にインストールします。

Arduino IDE : <https://www.arduino.cc/en/software/>

- ② ダウンロードする際に、Arduino の開発などに充てられる寄付金額の選択画面が表示されます。寄付は任意で、PayPal かクレジットカードで簡単に支払えます。この機会に支払っておくと良いでしょう。寄付済みや寄付したくない場合は[JUST DOWNLOAD]を選択してください。

- ③ ボードマネージャの設定：Arduino IDE を起動し、[ファイル]メニュー内の[環境設定]を開き、「追加のボードマネージャの URL」の欄に下記の URL を追加してから[OK]をクリックします。
追加のボードマネージャの URL :

https://raw.githubusercontent.com/espressif/arduino-esp32/ghpages/package_esp32_index.json

- ④ Arduino 用 ESP32 開発環境：Arduino IDE の [ツール]メニュー内の[ボード]でボードマネージャを開き、検索窓に「esp32」を入力し、esp32 by Espressif Systems をインストールします。

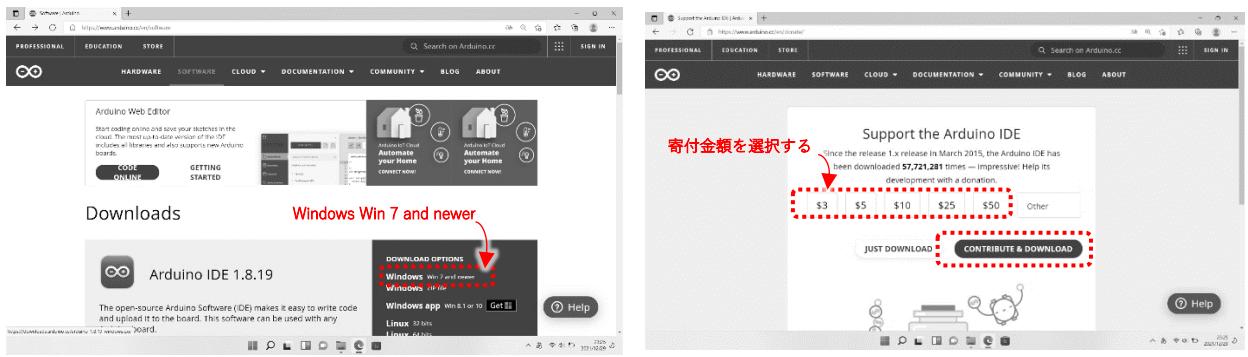
- ⑤ ボード設定：[ツール]メニュー内の[ボード]で ESP32 Arduino を選択後、ESP32C3 Dev Module を選択してください。

- ⑥ COM ポート番号の確認：M5Stamp C3U モジュールの場合は本体中央のボタンを押した状態で、M5Stamp C3 の場合はそのままの状態で、PC の USB 端子に接続してください。

Windows はドライバを自動設定し、COM ポート番号を割り振ります。モジュール COM ポート番号は、スタートメニュー（画面上の Windows ロゴ）を右クリックして[デバイスマネージャー]を選択し、図 1-2 の⑥の「ポート (COM と LPT)」をダブルクリックすると、M5Stamp C3 の場合は「USB-Enhanced-SERIAL CH9102 (COM○)」の丸括弧内に、M5Stamp C3U の場合は「USB シリアル デバイス (COM○)」の丸括弧内に表示されます。

- ⑦ Arduino IDE の[ツール]メニュー内の[シリアルポート]を選択し、⑥で確認した COM ポート番号を設定してください。

- ⑧ プログラムを入力後、Arduino IDE の画面左上の右矢印(⇒)ボタンでコンパイルと M5Stamp C3/C3U に書き込みを行います。エラーが発生するとステータス表示部が橙色に変化します。



- ① ダウンロードページ (<https://www.arduino.cc/en/software>) の「DOWNLOAD OPTION」で OS を選択する



- ③ [ファイル]メニュー内の[環境設定]を開き、「追加のボードマネージャの URL」の欄に URL を追加してから[OK]を押す



- ⑤ [ツール]メニュー内の[ボード]で ESP32 Arduino → ESP32C3 Dev Module を選択



- ⑦ [ツール]メニュー内の[シリアルポート]を選択し、⑥で確認した COM ポート番号を設定する

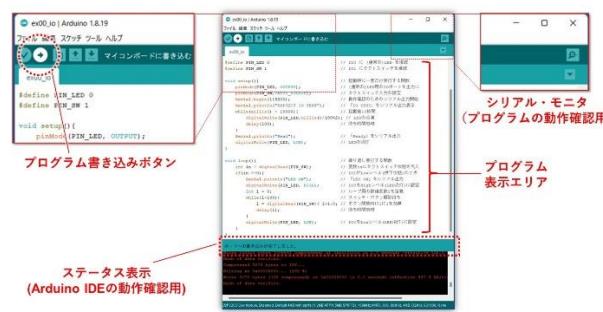
- ② 寄付金額を選択して PayPal かクレジットカードで支払う。寄付しない場合は JUST DOWNLOAD を選択する



- ④ [ツール]メニューの[ボード]内のボードマネージャで、「esp32」を入力後、esp32 をインストールする



- ⑥ デバイスマネージャー]で「ポート (COM と LPT)」をダブルクリックし、CH9102 のポート番号を確認する



- ⑧ 画面左上の右矢印(→)ボタンでプログラムを書き込む。エラーが発生するとステータス表示部が橙色に変化する

図 1-2 Arduino 開発環境のセットアップ手順 ①Arduino サイトから②IDE をダウンロードし、③ボードマネージャの URL を追加し、④esp32 をインストールする。⑤ESP32C3 用の設定を行い、⑥COM ポートの確認と⑦設定後、⑧M5Stamp C3/C3U へのプログラムの書き込みができるようになる

準備2 本書用のサンプル・プログラムのダウンロードする

筆者が本書用に作成したサンプル・プログラムは、下記からダウンロードすることができます。

本書用プログラム集：

<https://bokunimo.net/git/esp32c3/>

ZIP 形式ダウンロード：

<https://bokunimo.net/git/esp32c3/archive/master.zip>

ZIP 形式でダウンロード後、PC 内に展開（ZIP フォルダ内の esp32c3-master フォルダを任意の場所にコピー）してから、拡張子 ino のプログラムを Arduino IDE で開きます。同じフォルダ内に複数の ino ファイルがあった場合、どれか一つを開けば、自動的に全てのプログラムが読み込まれます。

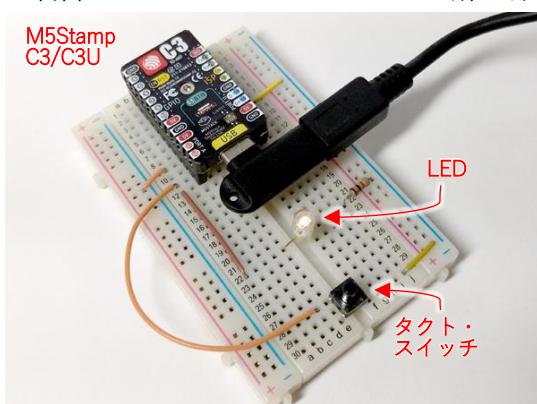
esp32c3-master フォルダ内には、複数のフォルダが含まれており、本書では主に learning フォルダ内のサンプル・プログラムを使用します。表 1-4 に主要なフォルダの内容と、本書で使用する主なサンプル・プログラムの一覧表を示します。

表 1-4 本書で使用する主なサンプル・プログラム

フォルダ・ファイル名	説明
learning/ex00_io	後述の IO 実験用ボードの動作確認用プログラム
learning/ex01_led_io	LED 制御用プログラム。HTTP サーバ機能によりブラウザから制御可能
learning/ex02_sw_io	押しボタンの送信プログラム。ex01_led の LED の制御や LINE への送信が可能
learning/ex03_lum	照度センサの送信プログラム。照度値をクラウド(Ambient)に送信しグラフ化が可能
learning/ex04_lcd	小型液晶への表示プログラム。ex02, 03, 05 の送信データの表示が可能
learning/ex05_hum	温度 + 湿度センサの送信プログラム。家じゅうの部屋に設置すれば居住環境の監視が可能
learning/ex06_pir	人感センサ・ユニット (PIR Motion Sensor) を使った Wi-Fi 人感センサ用プログラム
learning/ex08_ir_out	赤外線リモコン・ユニット (IR Unit) を使った Wi-Fi 赤外線・リモコン用プログラム
tools/udp_logger.py	PC やラズベリー・パイ上でセンサ値を受信する Python プログラム。ex02, 03, 05 に対応
tools/udp_monitor_chart.py	センサ値の棒グラフ化、CSV 保存、履歴表示などに対応した Python プログラム
tools/srv_myhome.py	自分だけの My ホーム・オートメンション・システム用サンプル・プログラム

準備3 ブレッド・ボードで IO 実験用ボードを作成する

本書で解説する IO 制御プログラムで使用する IO 実験用ボード（写真 1-4）の作成方法について説明します。本ボードが無くても本書の実験を進めることができます。実際に製作してみると IO 制御のハードウェアについての理解も深まるので、ぜひ作ってみてください。



ブレッド・ボードでの実験にちょうど良い
切手サイズ

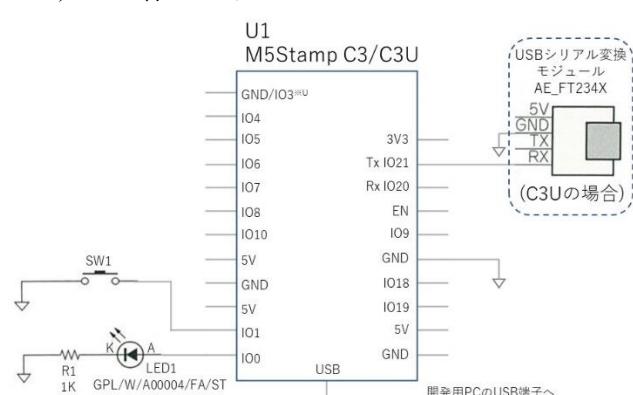


図 1-3 IO 実験用ボードの回路図

IO0 に LED と電流制限抵抗を、IO1 にタクト・スイッチを接続する。C3U の場合はデバッグ（動作確認）用に USB シリアル変換モジュールを実装する

図 1-3 に IO 実験ボードの回路図を示します。本例では、IO0 ピン（ESP32-C3 の GPIO0）に LED と電流制限抵抗を、IO1 ピン（ESP32-C3 の GPIO1）にタクト・スイッチを接続しました。M5Stamp C3/C3U では各 GPIO のピン名を G0, G1 と表記していますが、本書では ESP32-C3 の製造元 Espressif Systems 社が販売するモジュール製品に合わせて IO0, IO1 と記します。

M5Stamp C3/C3U Mate に付属のピン・ソケット（メス側コネクタ）は、**写真 1-5** のように、モジュールの裏面（部品が実装されていない面）に取り付け、表面で半田付けを行います。半田付け前に、一度、電源を入れて動作確認を行い、確認後は、電源を切ってから、端子近くの部品に注意しながら半田付けしてください。モジュール側にピン・ヘッダを実装するよりも、静電気リスクの低減や、単体動作時の短絡リスク低減、リード部品の直接接続のほか、複数のメリットがあります。

ブレッド・ボード上には、M5Stamp C3/C3U 用のピン・ヘッダ（オス側コネクタ）、LED、抵抗、タクト・スイッチを実装し、**写真 1-6** のように配線してください。ピン・ヘッダは、ピンの長い方をブレッド・ボード側に接続します。M5Stamp C3U の場合は、シリアル・モニタ用に Tx(IO21)端子を USB シリアル変換モジュール AE-FT234X の RXD 端子に接続してください。さらに Rx(IO20)端子を AE-FT234X の TXD 端子に接続すると、AE-FT234X 経由でのプログラム書き換えも出来ます。

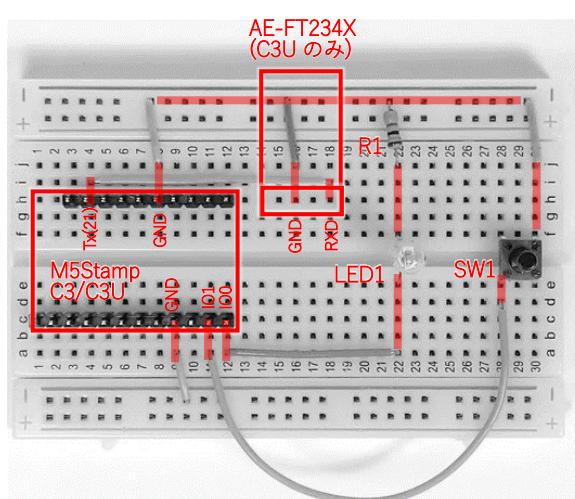


写真 1-6 ブレッド・ボードで製作する IO 実験用ボードの配線例

M5Stamp C3/C3U モジュールを取り付けるブレッド・ボード上の位置に、ピン・ヘッダを接続する。ピンの長い方をブレッド・ボード側にすると、モジュールの抜き差しがしやすくなる

準備④ IO 実験用ボードの動作確認

ブレッド・ボードが完成したら、M5Stamp C3/C3U モジュールを取り付け、ダウンロードしたサンプル・プログラムの learning フォルダ内の ex00_io.ino を Arduino IDE で書き込みます。

M5Stamp C3U モジュールの場合は、USB を接続するときにモジュール中央のボタンを押しておき、ダウンロード・モードに手動で設定します。また、起動時はモジュール本体の[(EN)RST]ボタンを押してください。M5Stamp C3 の場合は、本体の操作が不要です。

M5Stamp C3/C3U 上に書き込んだプログラム ex00_io.ino が起動すると、10 秒間、LED が点滅します。その後、タクト・スイッチを押下しているときだけ LED が点灯し、開放すると消灯します。

プログラム① IO 実験用ボードで IO 制御

M5Stamp C3/C3U 用のサンプル・プログラム ex00_io.ino と、IO 実験用ボードの動作内容について、リスト 0 を用いて説明します。



写真 1-5 M5Stamp C3 にコネクタを取り付ける
M5Stamp C3 Mate に付属のピン・ソケット（メス側コネクタ）を M5Stamp C3 モジュールの裏面に取り付け、表面で半田付けした

- ① Arduino のプログラムには、起動時に実行する `setup` 関数と、その後に、繰り返し実行する `loop` 関数が必要です。M5Stamp C3/C3U の起動時に、`setup` 関数を自動実行します。
- ② IO 制御の対象ピン名と入出力を設定する関数 `pinMode` を使って、LED を接続した IO0 を出力 (OUTPUT) に設定します。
- ③ タクト・スイッチ SW1 を接続した IO1 を入力 (INPUT_PULLUP) に設定します。丸括弧内の第 1 引数は IO 番号です。プログラム冒頭の `#define PIN_SW 1` で IO1 を示します。第 2 引数の INPUT に続く _PULLUP は、ESP32-C3 マイコン内でのプルアップ抵抗（約 45 kΩ）です。この設定により IO1 は IC 内のプルアップ抵抗を経由して電源 3.3 V に接続されます。
- ④ LED の点滅処理部です。関数 `millis` は起動してからの経過時間を示し、単位はミリ秒です。10,000 ミリ秒 (10 秒間)、`digitalWrite` 命令 (⑧参照) を使って LED を点滅制御します。
- ⑤ `setup` 関数の処理後に繰り返し処理を行う `loop` 関数です。⑥以下の処理を繰り返します。
- ⑥ 処理③で設定した IO1 の入力状態を取得し、変数 `in` に代入します。丸括弧内の引数は IO 番号 (1) です。タクト・スイッチが開放状態のときは、プルアップ抵抗を経由して約 3.3V が入力されます。この電圧を H レベルと呼び数値 1 が変数 `in` に代入されます。タクト・スイッチを閉じると約 0.0V (GND の電位) の L レベルが入力され、数値 0 が変数 `in` に代入されます。
- ⑦ 変数 `in` が 0 のとき、すなわちタクト・スイッチが押下状態のときに⑧～⑩の処理を行います。
- ⑧ 処理②で設定した IO0 に約 3.3V の H レベルを出力します。`digitalWrite` 命令の丸括弧内の最初の引数は IO 番号 (0) です。また、2 番目の引数は出力値で、ここでは H レベル (HIGH) を設定し、3.3V を出力します。このとき、LED と抵抗に電流が流れ、LED が点灯します。
- ⑨ タクト・スイッチを開放するまで待ち続ける処理部です。また、操作直後は、チャタリングと呼ばれる値が不安定になる状態が生じます。ここでは `in` が 1 となる開放状態が 100ms 繼続するまで待機してチャタリングを防止します (コラム参照)。
- ⑩ `digitalWrite` 命令の 2 番目の引数に L レベル (LOW) を設定し、LED を消灯します。

```

#define PIN_LED 0 // IO0 に (通常の) LED を接続
#define PIN_SW 1 // IO1 にタクトスイッチを接続

void setup() { // ① // 起動時に一度だけ実行する関数
    pinMode(PIN_LED, OUTPUT); // ② // (通常の) LED 用の IO ポートを出力に
    pinMode(PIN_SW, INPUT_PULLUP); // ③ // タクトスイッチ入力の設定
    Serial.begin(115200); // 動作確認のためのシリアル出力開始
    Serial.println("ESP32C3 IO TEST"); // 「IO TEST」をシリアル出力表示
    while(millis() < 10000){ // 起動後 10 秒間
        digitalWrite(PIN_LED, millis() / 100%2); // LED の点滅
        delay(100); // 待ち時間処理
    }
    Serial.println("Ready"); // 「Ready」をシリアル出力
    digitalWrite(PIN_LED, LOW); // LED の消灯
}

void loop() { // ⑤ // 繰り返し実行する関数
    int in = digitalRead(PIN_SW); // ⑥ // 変数 in にタクトスイッチ状態を代入
    if(in ==0){ // ⑦ // IO1 が Low レベル (押下状態) のとき
        Serial.println("LED ON"); // 「LED ON」をシリアル出力
        digitalWrite(PIN_LED, HIGH); // ⑧ // IO0 を High レベル (LED 点灯) に設定
        int i = 0; // ループ用の数値変数 i を定義
        while(i<100){ // スイッチ・ボタン解除待ち
            i = digitalRead(PIN_SW)? i+1:0; // ボタン開放時に i に 1 を加算
            delay(1); // 待ち時間処理
        }
        digitalWrite(PIN_LED, LOW); // ⑩ // IO0 を Low レベル (LED 消灯) に設定
    }
}

```

リスト 0 プログラム

`ex00_io.ino`

起動すると、10 秒間、LED が点滅し (④)、タクト・スイッチを押下しているときだけ (⑥) LED が点灯し (⑦)、開放すると消灯する (⑧)

2章 ESP32-C3 マイコン無線×IO 制御サンプル・プログラム集

プログラム① Wi-Fi コンシェルジェ 照明担当（ワイヤレス LED）



図 2-1 Wi-Fi コンシェルジェ 照明担当
HTTP サーバ機能を搭載した Wi-Fi コンシェルジェが LED を制御

HTTP サーバ機能を搭載した Wi-Fi コンシェルジェが LED を制御するプログラム ex01_led.ino を製作します。同じ LAN 内に接続したパソコンやスマートフォンのインターネット・ブラウザから、LED の点灯状態を取得して表示したり、LED の点灯状態を制御したりすることができます。

ハードウェアは、準備④で製作した IO 実験用ボードを使用しますが、M5Stamp C3/C3U モジュール単品でも実験が行えます。

プログラムは、準備②でダウンロードした learning→ex01_led→ex01_led.ino ファイルを Arduino IDE で開き、図 2-2 の(a)の箇所にお手持ちの無線 LAN アクセスポイント（ホーム・ゲートウェイ）の SSID とパスワードを記入してから、(b)右矢印ボタンをクリックして M5Stamp C3/C3U モジュールに書き込んでください。

M5Stamp C3/C3U モジュールの IP アドレスは、(c)シリアル・モニタを起動すると、(d)のように表示されます。確認した IP アドレスを(e)インターネット・ブラウザのアドレス・バーに入力すると、LED の点灯状態と、点灯／消灯制御ボタンが表示されます。なお、M5Stamp C3U の場合は、Arduino IDE の[ツール]メニュー内の[シリアルポート]を選択して、USB シリアル変換モジュール AE-FT234X の COM ポート番号に設定を変更してから(c)シリアル・モニタを起動してください。

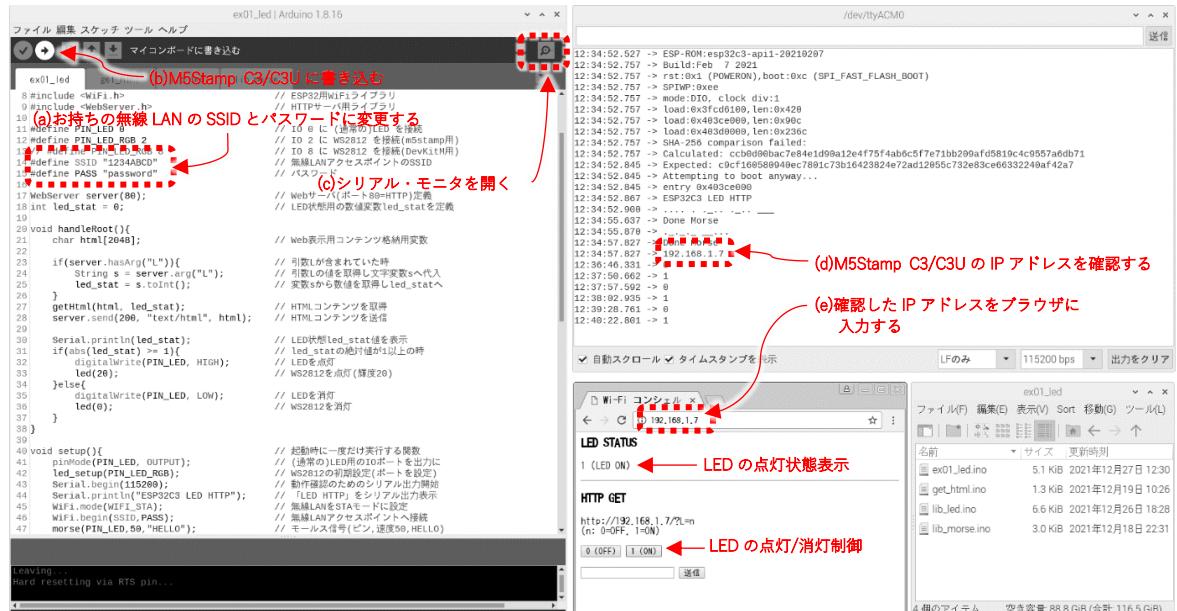


図 2-2 LED を制御するプログラム ex01_led.ino を実行したときの様子

(a)に無線アクセスポイントの SSID とパスワードを記入し、(b)M5Stamp C3/C3U モジュールにプログラムを書き込む。(e)ブラウザでアクセスすると、LED の制御画面が表示された

HTTP サーバ搭載 LED 制御プログラム ex01_led.ino の内容

HTTP サーバによる IO 実験ボード上の LED の制御機能を切り出したサンプル・プログラム ex01_led.ino の処理の手順について、**リスト 1** を用いて説明します。また、シリアル・モニタの表示例を**図 2-3** に示します。

- ① LED 制御用の IO ポート番号を#define を使って PIN_LED と定義します。
- ② お手持ちの無線 LAN アクセスポイント（ホーム・ゲートウェイ）の SSID とパスワードを、それぞれ SSID, PASS と定義します。
- ③ ライブラリ WebServer で定義された HTTP サーバを名称 server で生成します。以降、server. に続けて関数名を付与することで、ライブラリ WebServer 内の機能にアクセスできます。
- ④ 整数 int 型の変数 led_stat を初期値 0 で定義します。LED の状態は ON を示す 1 または OFF を示す 0 の 2 値なので、拡張性を考えなければ boolean 型を用いたほうがよいでしょう。
- ⑤ HTTP サーバにアクセスがあったときに実行する関数部です。本プログラム内から実行するのではなく、ライブラリ WebServer（の実体 server）から実行されるコールバック関数です。
- ⑥ HTTP サーバ server の関数 hasArg を使って、HTTP アクセスを受けた時のクエリ項目「L」が含まれているかどうかを確認し、含まれていた時に関数 arg を使って L の値を取得し、変数 led_stat に代入する処理部です。例えば、ブラウザに「http://(IP アドレス)/?L=1」のように入力すると、末尾の 1 が変数 led_stat に代入されます。
- ⑦ ブラウザに応答するための HTML コンテンツを取得し、文字列変数 tx に代入します。HTML コンテンツ生成部は別ファイル get_html.ino として同じフォルダに保存してあります。Arduino IDE 画面上のタブ「get_html」をクリックすると内容を確認できます。
- ⑧ HTTP サーバ server を使って、アクセス元のブラウザに、HTTP リザルト 200 (OK) と文字列変数 tx に代入された HTML コンテンツを送信します。
- ⑨ 変数 led_stat の絶対値が 1 以上、すなわち 0 以外のときに LED を点灯し、0 のときに LED を消灯する処理部です。前章の**リスト 0** と同様に digitalWrite 命令で IO を制御します。
- ⑩ WiFi.mode を使って、ステーション・モード（無線 LAN アクセスポイントに接続して使用するモード）に設定します。
- ⑪ WiFi.begin に処理②の SSID と PASS を渡して、無線 LAN アクセスポイントに接続します。
- ⑫ Wi-Fi 状態を取得する WiFi.status() が接続状態 WL_CONNECTED になるまで待機します。
- ⑬ HTTP サーバ server の関数 on を使って、HTTP アクセスを受けた時に呼び出すコールバック関数名を設定します。ここでは、ルート・ディレクトリ "/" へのアクセスに対して、処理⑤の関数名 handleRoot を設定します。
- ⑭ HTTP サーバ server の関数 begin を使って、HTTP サーバを起動します。
- ⑮ HTTP サーバ server の関数 handleClient を使って、外部からアクセスがあったかどうかを確認します。本例では、処理⑬で設定した通りルート・ディレクトリにアクセスがあると、処理⑤の関数 handleRoot が実行されます。

なお、前節で動作確認に使用したプログラム ex01_led.ino には、本説明の HTTP サーバによる IO 実験用ボード上の LED 制御に加え、M5Stamp C3/C3U 内蔵 LED を制御できるほか、起動後に LED の点滅によるモールス信号で IP アドレスを表示する機能などを追加しました。追加部の動作内容については、プログラムの各行の右側に記したコメントを参考にしてください。

```

#include <WiFi.h> // ESP32 用 WiFi ライブラリ
#include <WebServer.h> // HTTP サーバ用ライブラリ

#define PIN_LED 0 // I/O 0 に (通常の)LED を接続
#define SSID "1234ABCD" // 無線 LAN アクセスポイントの SSID
#define PASS "password" // パスワード

WebServer server(80); // Web サーバ(ポート 80=HTTP)定義
int led_stat = 0; // LED 状態変数 led_stat を定義

void handleRoot() { // 受信用, 送信用文字列
    String rx, tx;

    if(server.hasArg("L")){
        rx = server.arg("L");
        led_stat = rx.toInt(); // 引数 L の値を取得し文字変数 rx へ代入
        // 変数 s から数値を取得し led_stat へ
    }
    tx = getHtml(led_stat); // HTML コンテンツを取得
    server.send(200, "text/html", tx); // HTML コンテンツを送信

    Serial.println(led_stat);
    if(abs(led_stat) >= 1){ // LED 状態 led_stat 値を表示
        digitalWrite(PIN_LED, HIGH); // led_stat の絶対値が 1 以上の時
        // LED を点灯
    }else{
        digitalWrite(PIN_LED, LOW); // LED を消灯
    }
}

void setup(){
    pinMode(PIN_LED, OUTPUT); // (通常の)LED 用の I/O ポートを出力に
    Serial.begin(115200); // 動作確認のためのシリアル出力開始
    Serial.println("ESP32C3 LED HTTP"); // 「LED HTTP」をシリアル出力表示
    WiFi.mode(WIFI_STA); // 無線 LAN を STA モードに設定
    WiFi.begin(SSID, PASS); // 無線 LAN アクセスポイントへ接続
    while(WiFi.status() != WL_CONNECTED){ // 接続に成功するまで待つ
        digitalWrite(PIN_LED, millis()/100%2); // LED の点滅
        delay(50); // 待ち時間処理
    }
    server.on("/", handleRoot); // HTTP 接続時のコールバック先を設定
    server.begin(); // Web サーバを起動する
    Serial.println(WiFi.localIP()); // 本機の IP アドレスをシリアル出力
}

void loop(){
    server.handleClient(); // 繰り返し実行する関数
} // クライアントから Web サーバ呼び出し

```

リスト 1 プログラム ex01_led_io.ino

HTTP サーバ (⑬⑭) がクエリ「?L=」を受信 (⑯) すると、値に応じて LED を点灯／消灯制御 (⑯) する。※プログラムを書き込む前に②の設定が必要

```

11:46:44.116 -> ESP-ROM:esp32c3-api1-20210207
11:46:44.149 -> Build:Feb 7 2021
11:46:44.149 -> rst:0x1 (POWERON), boot:0xc (SPI_FAST_FLASH_BOOT)
11:46:44.149 -> SPIWP:0xee
11:46:44.149 -> mode:DIO, clock div:1
11:46:44.149 -> load:0x3fc6100, len:0x420
11:46:44.149 -> load:0x403ce000, len:0x90c
11:46:44.149 -> load:0x403d0000, len:0x236c
11:46:44.149 -> SHA-256 comparison failed:
11:46:44.149 -> Calculated: XXXXXXXXXXXXXXXXXXXXXXXXX
11:46:44.202 -> Expected: XXXXXXXXXXXXXXXXXXXXXXXXX
11:46:44.202 -> Attempting to boot anyway...
11:46:44.202 -> entry 0x403ce000
11:46:44.381 -> ESP32C3 LED HTTP <----- 【起動メッセージ】
11:46:49.387 -> 192.168.1.7 <----- 【本機の IP アドレス】
11:46:55.725 -> 1 <----- 【LED ON を実行】
11:47:00.006 -> 0 <----- 【LED OFF を実行】

```

**図 2-3 プログラム
ex01_led.ino の実行結果**
起動メッセージと本機の IP アドレスを表示する。HTTP サーバ受信した指示に応じて LED の点灯／消灯制御を行った

プログラム② Wi-Fi ボタン・送信機

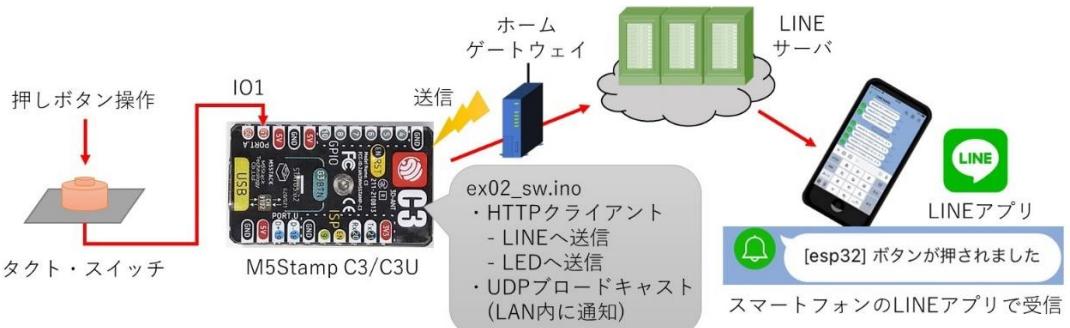


図 2-4 Wi-Fi ボタン・送信機

HTTP クライアント機能を搭載した Wi-Fi 送信機が LINE アプリにメッセージを送信する

HTTP クライアント機能を搭載した Wi-Fi ボタン送信機のプログラム ex02_sw.ino を製作します。スマートフォン用 LINE アプリにメッセージを送信したり、前節で作成した Wi-Fi コンシェルジェ照明担当（ワイヤレス LED）の LED をリモート制御したりすることができます。また、LAN 内に UDP ブロードキャストで文字列 Ping を送信し、プログラム④ の Wi-Fi コンシェルジェ掲示板担当（ワイヤレス LCD）ex04_lcd にメッセージを表示することもできます。

ハードウェアは M5Stamp C3/C3U と IO 実験用ボードを使用します。M5Stamp C3 単品でも本体中央のボタンを使った実験ができますが、M5Stamp C3U の場合は中央ボタンが使用できません。

プログラムは、ダウンロードした learning フォルダ内の ex02_sw フォルダ内のものを使用します。プログラム内に無線 LAN 接続用の SSID と PASS を記入し、LINE トークンを LINE_TOKEN に、リモート制御対象の M5Stamp の IP アドレスを LED_IP に記入してください（詳細は後述）。

アルカリ乾電池による長期間動作を実現するスリープ機能

本例ではボタンが押された時だけ動作するスリープ機能を使ったので、乾電池駆動時に長期間のボタン待ち受けが可能です。図 2-5 の Sleep 表示でスリープし、ESP32-C3 チップ単体の消費電流を約 0.01mA（実測）まで低減できました。写真 2-1 のように、M5Stamp C3/C3U モジュールの 5V 端子にアルカリ乾電池 3 本で供給することもできます。ただし、M5Stamp C3 モジュール内の電源回路等を含めた電流は約 0.40mA（実測）、M5Stamp C3U は 0.32mA（実測）でした。それでも単 3 電池（2000mAh）で半年間以上の待ち受け期間（約 200 日）が見込めます。

```
/dev/ttyACM0
04:26:07.311 -> ESP-ROM:esp32c3-api1-20210207
04:26:07.344 -> Build:Feb 7 2021
04:26:07.344 -> rst:0x5 (DSLEEP),boot:0xc (SPI_FAST_FLASH_BOOT)
04:26:07.344 -> SPIWP:0xe
04:26:07.344 -> mode:DIO, clock div:1
04:26:07.344 -> load:0x3fc6d100, len:0x420
04:26:07.344 -> load:0x493ce000, len:0x90c
04:26:07.344 -> load:0x493d0000, len:0x236c
04:26:07.344 -> SHA-256 comparison failed:
04:26:07.344 -> Calculated: ccb6d00bac7e84e1d90a12e4f75f4ab6c5f7e71bb209af5819c4c95
04:26:07.344 -> Expected: c9cf160580940ec7801c73b16423824e72ad12055c732e83ce66332240
04:26:07.344 -> Attempting to boot anyway...
04:26:07.344 -> entry 0x493ce000
04:26:07.477 -> ESP32C3 SW UDP LINE LED
04:26:07.477 -> Wake = 7
04:26:10.496 -> 192.168.1.255
04:26:10.695 -> SW = 1
04:26:10.794 -> Sleep...
GPIO 割込みによる起動時に 7 を得る
UDP ブロードキャスト送信先 IP アドレス
ESP32-C3 をスリープ状態に設定する
```

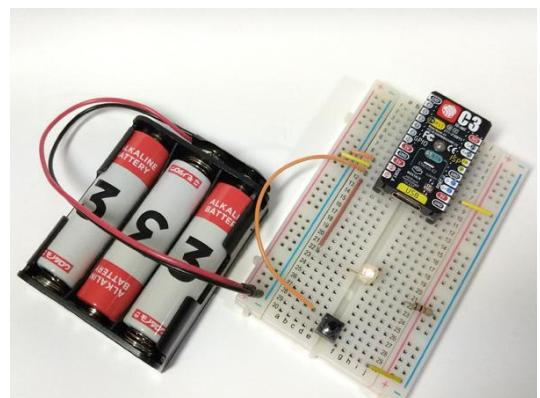


図 2-5 Wi-Fi ボタン送信機を作動させたときのシリアル・モニタの様子

Sleep 表示でスリープし、ESP32-C3 チップの消費電流を約 0.01mA（実測）まで低減できる

写真 2-1 乾電池で駆動するときの一例

ボタンが押されたときだけ動作するスリープ機能を使うことで、乾電池駆動時に約半年間（試算）の待ち受けが可能になる

LINE Notify 用のトークンを取得し、設定する

スマートフォンの LINE アプリにメッセージを送信するのに必要な LINE Notify 用のトークンの取得手順を、以下に示します。

1. インターネット・ブラウザを使って、<https://notify-bot.line.me/> にアクセスし、ブラウザに表示された画面右上の【ログイン】から、LINE アカウントの入力と本人認証をしてください。
2. ログイン後、画面右上のアカウント・メニューから【マイページ】を選択し、【トークンを発行する】を選択すると、図 2-6 のトークン発行ダイヤログが開きます。
3. トークン発行ダイヤログでトークン名を入力します。本例では「esp32」を入力しました。
4. 送信先のトグルームに「1:1 で LINE Notify から通知を受け取る」を選択します。
5. 【発行する】ボタンでトークンが発行されるので、【コピー】ボタンでクリップボードへコピーし、プログラム ex02_sw.ino の LINE_TOKEN のダブルコート(")内に貼り付けてください。

M5Stamp C3/C3U (2 台) を使ったリモート LED 制御

プログラム ex02_sw.ino を書き込んだ M5Stamp C3/C3U と、プログラム ex01_led.ino を書き込んだ M5Stamp C3/C3U を用意すれば、ex02_sw.ino の押しボタンから ex01_led.ino の LED の点灯／消灯制御ができます。

プログラム ex02_sw.ino 内の #define LED_IP に続くダブルコート(")内に、ex01_led.ino 側の M5Stamp C3 /C3U の IP アドレスを記入してください。写真 2-2 のようにタクト・スイッチを押すと LED が点灯し、タクト・スイッチを押し続けると LED が消灯します。

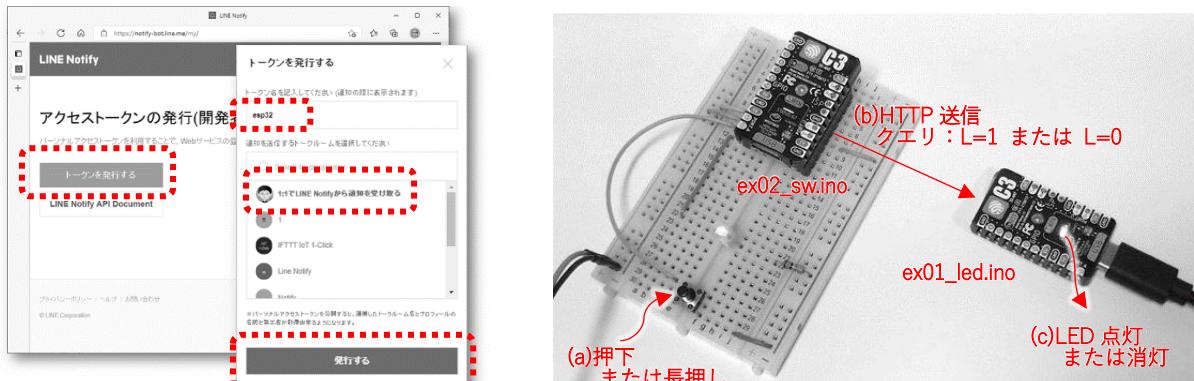


図 2-6 トークン発行ダイヤログ

LINE Notify (<http://notify-bot.line.me/>) の【マイページ】で、【トークンを発行する】を選択すると表示される

写真 2-2 M5Stamp C3 を使ったリモート LED 制御

ex02_sw.ino から ex01_led.ino の LED の点灯／消灯制御ができる。タクト・スイッチを押すと LED が点灯し、タクト・スイッチを押し続けると LED が消灯する

スリープ機能を使ったボタン送信プログラム ex02_sw.ino の内容

HTTP クライアント機能によるメッセージ送信機能と、UDP 送信機能、スリープ機能を切り出したサンプル・プログラム ex02_sw_ino.ino の処理の手順について、リスト 2 を用いて説明します。

- ① (LINE) 取得した LINE Notify 用トークンを#define を使って LINE_TOKEN に定義します。
LINE Notify を使用しない場合は、your_token のままにしておきます。
- ② (LED) 同じ LAN 内で動作する Wi-Fi コンシェルジェ照明担当（ワイヤレス LED）の IP アドレスを LED_IP として定義します。使用しない場合は、192.168.1.0 のままにしておきます。
- ③ (スリープ) タクト・スイッチを接続した IO ポート番号を PIN_SW に定義します。
- ④ (Wi-Fi) お手持ちの無線 LAN アクセスポイントの SSID とパスワードを定義します。

- ⑤ (UDP) UDP ブロードキャストの宛先ポート番号を定義します。
- ⑥ (UDP) UDP ブロードキャスト用アドレスを格納する変数（オブジェクト）を生成します。
- ⑦ (スリープ) プログラムが起動した理由を取得します。ここでは戻り値の理由名を整数値に変換して変数 wake に代入しました。理由名は esp_sleep_wakeup_cause_t で定義されているので、互換性を考慮する場合は、列挙値 (ESP_SLEEP_WAKEUP_GPIO) を使ってください。
- ⑧ (スリープ) 起動理由が 7 (GPIO からの起動) 以外のときに末尾の sleep 関数⑭を呼びます。
- ⑨ (Wi-Fi) 本機を無線 LAN アクセスポイントに接続します。
- ⑩ (UDP) 本機の IP アドレスを取得し、アドレスの 4 番目の数値を 255 に変更し、ブロードキャスト IP アドレス（一般的な家庭用 LAN で用いられるクラス C の場合）に変換します。
- ⑪ (UDP) UDP ブロードキャスト送信部です。ライブラリ WiFiUDP の関数 beginPacket で UDP 通信用の設定を行い、関数 println で送信データを渡し、関数 endPacket で送信します。
- ⑫ (LINE) Line Notify への送信部です。ライブラリ HTTPClient の関数 begin でリクエスト先を設定し、関数 addHeader で HTTP ヘッダ部に Line Notify 用のトークンを追加し、関数 POST で HTTP 本文を追加して LINE にメッセージを送信します。
- ⑬ (LED) Wi-Fi コンシェルジュ照明担当への送信部です。URL 内にクエリ「L=制御値」を追加し、HTTPClient の関数 get でリクエストを送信します。
- ⑭ (スリープ) ESP32-C3 チップの消費電流を約 0.01mA（実測）まで低減可能な GPIO 割込み起動用スリープ状態に遷移するための関数 sleep の定義部です。
- ⑮ (スリープ) 処理③のスリープ解除用のタクト・スイッチの開放待ち処理です。IO 入力値が H レベル（1, ボタン開放）のまま 100ms 継続するまで待機し、チャタリングを防止します。
- ⑯ (スリープ) GPIO 割込み起動が可能なスリープ機能の設定部です。変数 pin には各 IO ポートの割込み要否を代入します。ポート番号+1 桁目のビットを 1 にすると有効、0 で無効です。本例のポート番号 1 の場合、2 進数 2 桁目が 1, すなわち 10b となり、10 進数の 2 が代入されます。また、変数 val には起動する入力レベルを代入します。本例では L レベルで起動します。

```
#include <WiFi.h> // ESP32 用 WiFi ライブラリ
#include <WiFiUdp.h> // UDP 通信を行うライブラリ
#include <HTTPClient.h> // HTTP クライアント用ライブラリ
#include "esp_sleep.h" // ESP32 用 Deep Sleep ライブラリ
#define LINE_TOKEN "your_token" // LINE Notify トークン★要設定 ①
#define LED_IP "192.168.1.0" // LED 搭載子機 IP アドレス★要設定 ②
#define PIN_LED 0 // IO 0 に (通常の)LED を接続
#define PIN_SW 1 // IO1 にタクトスイッチを接続 ③
#define SSID "1234ABCD" // 無線 LAN アクセスポイント SSID
#define PASS "password" // パスワード ④
#define PORT 1024 // 送信のポート番号 ⑤
IPAddress IP_BROAD; // ブロードキャスト IP アドレス ⑥
int wake = (int)esp_sleep_get_wakeup_cause(); // 起動理由を変数 wake に保存 ⑦
void setup() {
    pinMode(PIN_LED, OUTPUT); // (通常の)LED 用 IO ポートを出力に
    pinMode(PIN_SW, INPUT_PULLUP); // タクトスイッチ入力の設定
    Serial.begin(115200); // 動作確認のためのシリアル出力
    Serial.println("ESP32C3 SW UDP LINE LED"); // 「SW UDP」をシリアル出力表示
    Serial.print(" Wake = "); // 「wake =」をシリアル出力表示
    Serial.println(wake); // 起動理由 no をシリアル出力表示
    if(wake != 7) sleep(); // ボタン以外で起動時にスリープ ⑧
    WiFi.mode(WIFI_STA); // 無線 LAN を STA モードに設定
    WiFi.begin(ssid, pass); // 無線 LAN アクセスポイント接続
    while(WiFi.status() != WL_CONNECTED) { // 接続に成功するまで待つ
        digitalWrite(PIN_LED, millis()/100%2); // (通常の)LED の点滅
        if(millis() > 30000) sleep(); // 30 秒超過でスリープ
        delay(50); // 待ち時間処理
    }
}
```

リスト 2 プログラム ex02_sw_io.ino

HTTP クライアント（⑫⑬）が LINE Notify や LED に HTTP リクエストを送信する。送信後、GPIO 割込みによる起動が可能なスリープ状態に移行する（⑭～⑯）。※プログラムを書き込む前に①②④の設定が必要

```

    }
    digitalWrite(PIN_LED, HIGH);
    IP_BROAD = WiFi.localIP();
    IP_BROAD[3] = 255;
    Serial.println(IP_BROAD);
}

void loop() {
    WiFiUDP udp; // 繰り返し実行する関数
    udp.beginPacket(IP_BROAD, PORT); // UDP 通信用のインスタンス定義
    udp.print("Ping"); // UDP 送信先を設定
    udp.endPacket(); // メッセージ"Ping"を送信
    delay(200); // UDP 送信の終了(実際に送信)
    // 送信待ち時間
    HTTPClient http; // HTTP リクエスト用インスタンス
    http.setTimeout(15000); // タイムアウトを 15 秒に設定する
    String url; // URL を格納する文字列変数を生成
    if(strlen(LINE_TOKEN) > 42){ // LINE_TOKEN 設定時
        url = "https://notify-api.line.me/api/notify"; // LINE の URL を代入
        Serial.println(url); // 送信 URL を表示
        http.begin(url); // HTTP リクエスト先を設定する
        http.addHeader("Content-Type", "application/x-www-form-urlencoded");
        http.addHeader("Authorization", "Bearer " + String(LINE_TOKEN));
        http.POST("message=ボタンが押されました"); // メッセージを LINE へ送信する
        http.end(); // HTTP 通信を終了する
    }
    if(strcmp(LED_IP, "192.168.1.0")){ // 子機 IP アドレス設定時
        url = "http://" + String(LED_IP) + "?L="; // アクセス先 URL
        url += String(digitalRead(PIN_SW)); // ボタン開放時に LED を ON
        Serial.println(url); // 送信 URL を表示
        http.begin(url); // HTTP リクエスト先を設定する
        http.GET(); // ワイヤレス LED に送信する
        http.end(); // HTTP 通信を終了する
    }
    sleep(); // 下記の sleep 関数を実行
}

void sleep(){ // スリープ実行用の関数
    Serial.println(digitalRead(PIN_SW)); // タクトスイッチ状態を表示
    int i = 0; // ループ用の数値変数 i
    while(i<100){ // スイッチ・ボタン解除待ち
        i = digitalRead(PIN_SW) ? i+1 : 0; // ボタン開放時に i に 1 を加算
        delay(1); // 待ち時間処理
    }
    digitalWrite(PIN_LED, LOW); // (通常の)LED を消灯
    Serial.println("Sleep..."); // 「Sleep」をシリアル出力表示
    delay(100); // 待ち時間処理
    uint64_t pin = 1ULL << PIN_SW; // 起動用 IO ポートのマスク作成
    esp_deepsleep_gpio_wake_up_mode_t val = ESP_GPIO_WAKEUP_GPIO_LOW;
    esp_deep_sleep_enable_gpio_wakeup(pin, val); // スリープ解除設定
    esp_deep_sleep_start(); // Deep Sleep モードへ移行
}

```

```

10:58:38.084 -> ESP-ROM:esp32c3-api1-20210207
10:58:38.084 -> Build:Feb 7 2021
10:58:38.084 -> rst:0x5 (DSLEEP), boot:0xc (SPI_FAST_FLASH_BOOT)
10:58:38.084 -> SPIWP:0xee
10:58:38.084 -> mode:DIO, clock div:1
10:58:38.084 -> load:0x3fc6100, len:0x420
10:58:38.134 -> load:0x403ce000, len:0x90c
10:58:38.134 -> load:0x403d0000, len:0x236c
10:58:38.134 -> SHA-256 comparison failed:
10:58:38.134 -> Calculated: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10:58:38.134 -> Expected: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10:58:38.134 -> Attempting to boot anyway...
10:58:38.134 -> entry 0x403ce000
10:58:38.349 -> ESP32C3 SW UDP LINE LED -----【起動メッセージ】
10:58:38.349 -> Wake = 7 -----【ボタン起動時 7】
10:58:42.367 -> 192.168.1.255 -----【UDP 送信】
10:58:42.599 -> http://192.168.1.7/?L=1 -----【HTTP 送信】
10:58:43.497 -> Sleep... -----【スリープモードへ移行】

```

図 2-7 プログラム ex01_led.ino の実行結果
起動メッセージと本機の IP アドレスを表示する。HTTP サーバ受信した指示に応じて LED の点灯／消灯制御を行った

製作した機器の安全や信頼性に関する注意点

本書で製作した機器は、実験を目的としたものです。例えば、電源が短絡し、電線やブレッド・ボード、乾電池等が発熱、出火したとしても周囲に燃え広がらないような場所で実験を行って下さい。

また、本機で得られたセンサ値は目安値です。個体差や電圧、温度などの条件によって変化するので、センサ機器として販売することや、得られた値を測定値として使用することはできません。

コラム1：タクト・スイッチのチャタリング

チャタリングとは、スイッチの切り替え時に発生する図 2-8 のようなノイズのことで、タクト・スイッチを押下したときや開放したときに発生することがあります。

Wi-Fi ボタン・送信機のスイッチ入力回路は、マイコンに内蔵されているプルアップ抵抗を利用して、タクト・スイッチを IO ポートに接続するだけで実現できますが、チャタリングが発生した場合に、誤動作しないようにする必要があります。

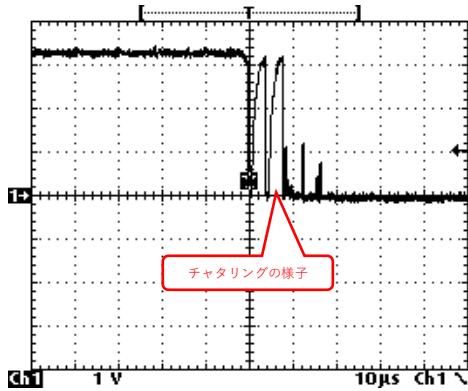


図 2-8 チャタリングが発生したときの様子

タクト・スイッチの接点の経年劣化や消耗などにより、スイッチの押下時 (HIGH レベルから LOW レベルへの遷移中) にノイズが発生した。スイッチを開放するときに発生することもある

例えば、デジタル時計の時刻を合わせようとした際に、1分だけ進めるつもりが3分進んでしまうといった誤作動を経験したことがないでしょうか。タクト・スイッチの利用形態としては、スイッチ押下時に非操作時（平常時）とは異なる機能を動作させることが多く、チャタリングによって当該機能が何度も繰り返し実行してしまうことがあります。

最も簡単な対策方法は、スイッチ状態の変化後、一定の時間（一例として 10ms～100ms 程度）、スイッチ状態の変化を無視することです。スイッチ状態を `digitalRead` 命令で読み取り後、スイッチ状態に変化が生じたときに `delay(100)` 命令で 100ms の待機を行う処理を入れることで、簡易的な対策が行えます。

しかし、この場合、長押しのように 100ms を超える押下があった場合、スイッチの解放時のチャタリングの影響を受けてしまう場合があります。例えば、Wi-Fi ボタン・送信機では、メッセージ送信後にスリープに遷移したにも関わらず、チャタリングによって、すぐに起動してしまい、再送信してしまうというケースが考えられます。

そこで、リスト 0 の⑨や、リスト 2 の⑯の開放待ち処理部では、`digitalRead` 命令による IO 入力の取得値が H レベル (1, ボタン開放) のまま 100ms 繼続するまで待機する対策を行いました。ハードウェアで対策する方法もありますが、回路部品の追加が必要になります。実験目的であれば、ソフトウェアによる対策のほうが簡単です。

3章 ESP32-C3 マイコン無線×センサ活用サンプル・プログラム集

プログラム③ Wi-Fi 照度計・送信機

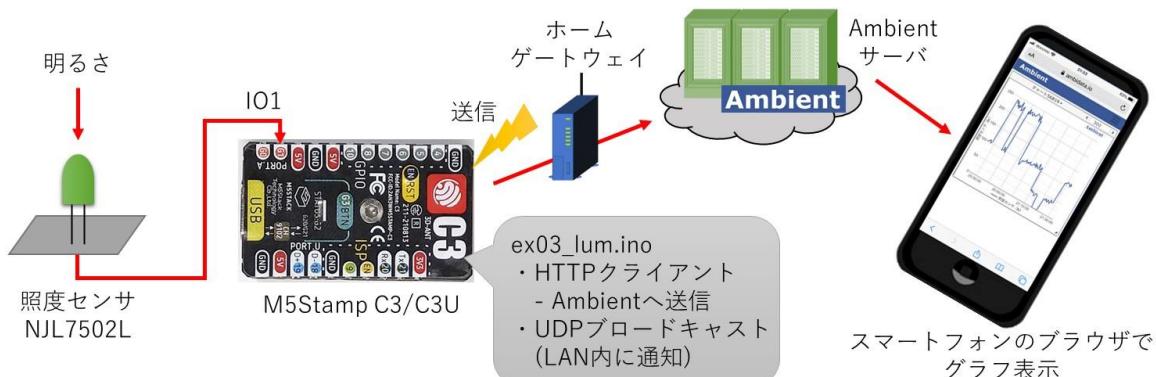


図 3-1 Wi-Fi 照度計・送信機

HTTP クライアント機能を搭載した Wi-Fi 送信機がセンサ値を Ambient に送信。スマホで表示

HTTP クライアント機能を搭載した Wi-Fi 照度計・送信機のプログラム ex03_lum.ino を製作します。IoT センサ用クラウド・サービス Ambient に照度センサ値を送信することで、パソコンやスマートフォンのブラウザで照度値のグラフ表示ができます。また、UDP でも照度値を送信します。

Wi-Fi 照度計・送信機のハードウェアの製作方法

ハードウェアは写真 3-1 のように M5Stamp C3 /C3U と照度センサ NJL7502L、負荷抵抗 $1k\Omega$ をブレッド・ボード上に実装して製作します。電源は単 3 型アルカリ乾電池 3 本を 5V 端子に入力しました。後述のソフトウェアにより、測定時以外は ESP32C3 をディープ・スリープ状態に設定し、また照度センサにも電源を供給しないことで、乾電池による長期間動作に対応しています。製作した Wi-Fi 照度計・送信機の回路図を図 3-2 に、配線図を写真 3-3 に示します。なお、C3U の場合は、Tx (IO21) 端子を USB シリアル変換モジュールの RXD 端子に接続してください（動作確認用）。

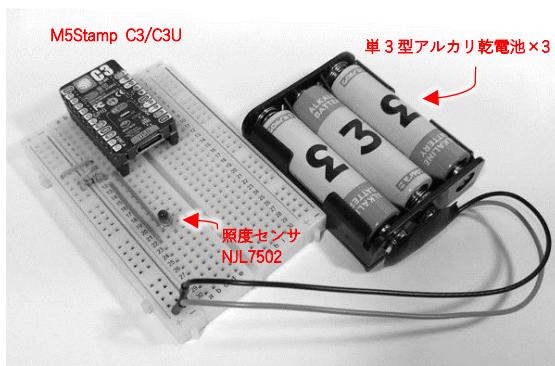


写真 3-1 照度計・送信機の製作例

乾電池による長期間動作に対応するハードウェアをブレッド・ボード上に構成する

照度センサ NJL7502L の接続方法

照度センサには、写真 3-2 示す日清紡マイクロデバイス製の NJL7502L を使用しました。一見、LED のようですが、作用はその反対で、受光した光量に応じた電流を出力します。リード線の長い方がコレクタ (C) 端子、樹脂モールド部に切り欠きがある方がエミッタ (E) 端子です。

照度センサの電源供給用にコレクタ (C) 端子を M5Stamp C3/C3U モジュールの IO0 へ接続し、測定中以外は照度センサが動作しないようにしました。照度センサが検出した電流は、エミッタ

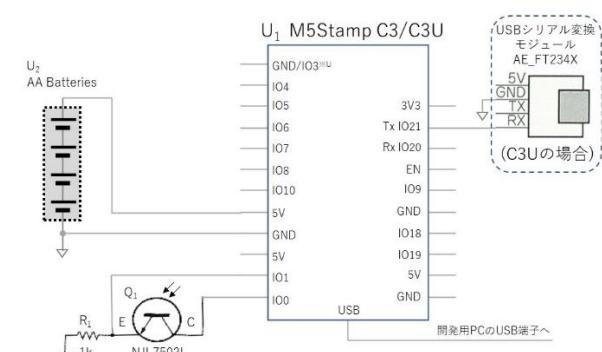


図 3-2 照度計・送信機の回路図

照度センサのコレクタを IO0 に、エミッタを IO1 に接続し、エミッタには負荷抵抗を接続する

(E) 側に接続した負荷抵抗 R_1 で電圧へ変換し、M5Stamp C3/C3U の IO1 に入力します。

この照度センサ NJL7502L は照度 100 lx につき約 $33\mu A$ の電流を流します。この電流が $1k\Omega$ の負荷抵抗 R_1 へ流れると、 $33mV$ の電圧が IO1 に入力されるので、照度値は下式のように計算します。

照度値の計算方法(負荷 $1k\Omega$ 時) :

$$\text{照度 (lx)} = 100(\text{lx}) \times \frac{\text{入力電圧(mV)}}{33(\text{mV})}$$

参考文献：新日本無線（現・日清紡マイクロデバイス）NJL7502L データシート Ver.2013-08-07

なお、得られた照度値は明るさの目安を示す値です。また、製作例は IoT システムの実験用です。



写真 3-2 照度センサ NJL7502L の拡大図

LED に似た照度センサ。リード線の長い方がコレクタ端子、樹脂モールドに切り欠きがある方がエミッタ端子

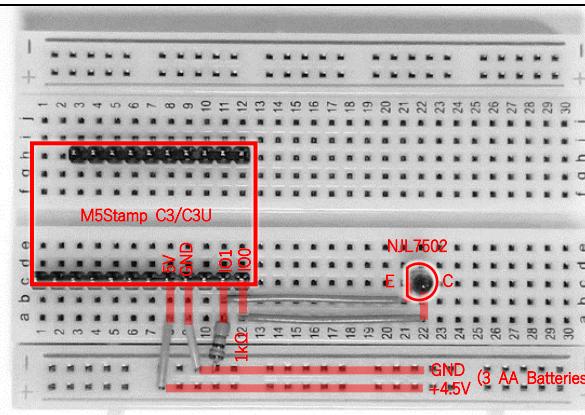


写真 3-3 ブレッド・ボードで製作する Wi-Fi 照度計・送信機の配線例

M5Stamp C3/C3U モジュールと照度センサ NJL7502L、負荷抵抗 ($1k\Omega$) を配線する

超低消費電力を実現するディープ・スリープ機能

本例では ESP32-C3 チップ内のタイマー割込みで起動可能なディープ・スリープ機能を使用することで、チップ単体のデータシート上で $5\mu A$ 、筆者の実測で $6\mu A$ の消費電力で待機することができます。ただし、M5Stamp C3 モジュール内の電源回路などの合計電流は約 $0.40mA$ （実測）でした。単3電池 3 本での動作期間は、1 時間に 1 度の送信であれば、5 か月程度になるでしょう。

図 3-3 に、照度値を Ambient へ HTTP 送信し、30 秒間のディープ・スリープを設定したときの動作の一例を示します。30 秒経過後は、再起動し、以降、送信とスリープを繰り返し実行します。

送信した照度値は、Ambient に蓄積され、時間経過にともなう変化の推移をグラフで表示することができます。図 3-4 にインターネット・ブラウザに表示した照度値の変化の一例を示します。

```
/dev/ttyACM0
01:33:10.060 -> ESP-ROM:esp32c3-api1-20210207
01:33:10.093 -> Build:Feb 7 2021
01:33:10.093 -> rst:0x5 (DLEEP),boot:0xc (SPI_FAST_FLASH_BOOT)
01:33:10.093 -> SPIWP:0xee
01:33:10.093 -> mode:DIO, clock div:1
01:33:10.093 -> load:0x3fc0d6100, len:0x420
01:33:10.093 -> load:0x403ce000, len:0x90c
01:33:10.093 -> load:0x403d0000, len:0x236c
01:33:10.093 -> SHA-256 comparison failed:
01:33:10.093 -> Calculated: ccb0d00bac7e84e1d90a12e4f75f4ab6c5f7e71bb209afdf5819c4c
01:33:10.140 -> Expected: c9cf160580940ec7801c73b16423824e72ad12055c732e83ce663322
01:33:10.140 -> Attempting to boot anyway...
01:33:10.140 -> entry 0x403ce000
01:33:10.358 -> ESP32C3 LUM
01:33:13.872 -> 192.168.1.255
01:33:13.971 -> illum_1,664
01:33:13.971 -> http://ambidata.io/api/v2/channels/46178/data
01:33:14.270 -> Sleep...
```

照度値 664 lx
HTTP 送信
ESP32-C3 をスリープ状態に設定する

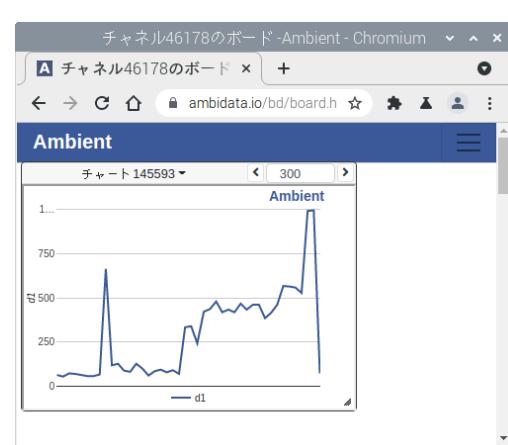


図 3-3 Wi-Fi 照度計・送信機の動作の様子

照度値 664 lx を Ambient へ HTTP 送信後、30 秒間のスリープを設定した。スリープ中は ESP32-C3 チップの消費電流を約 $6\mu A$ （実測）まで低減できる

図 3-4 Ambient 上での照度値の表示例

送信した照度値は、Ambient に蓄積され、時間経過にともなう推移をグラフで表示することができる

Ambient用のチャネルIDとライトキーを取得し、設定する

Ambientは、センサ値などの数値データを蓄積し、スマートフォンやパソコンのインターネット・ブラウザでグラフ表示できるクラウド・サービスです。アンビエントデーター社が提供しており、無料アカウントで最大8つのチャネルID（送信機）まで利用できます。

Ambientを利用するには、ユーザ登録後、チャネルIDと、ライトキーを取得し、それらをプログラム内に設定する必要があります。チャネルIDは送信機毎に割り当てられた番号で、ライトキーはデータを送信するときの16進数16桁の認証キーです。

以下にAmbient用のチャネルIDとライトキーを取得し、設定する手順を示します。

1. Ambientのウェブサイト (<https://ambidata.io/>) にアクセスしてください。
2. 右上の[ユーザ登録(無料)]ボタンから、メールアドレス、パスワードを設定し、アカウントを登録します。
3. チャネル一覧画面で、[チャネルを作る]ボタンを押下し、チャネルIDを新規作成します。
4. 「チャネルID」をリスト3の③のAmb_Idのダブルコート(")内に入力してください。
5. 「ライトキー」をリスト3の④のAmb_Keyに入力してください。

センサ用プログラムの基本プログラム ex03_lum.ino の内容

HTTP クライアント機能によるセンサ値の送信機能と、UDP 送信機能、スリープ機能を備えたサンプル・プログラム ex03_lum.ino の処理の手順について、リスト3を用いて説明します。

- ① (照度センサ) センサの負荷抵抗 R1 に生じた電圧を入力する IO ポート番号を定義します。
- ② (Wi-Fi) お手持ちの無線 LAN アクセスポイントの SSID とパスワードを定義します。
- ③ (Ambient) Ambient で取得した送信機ごとのチャネル ID (整数値) を、ダブルコート(")内に記入してください。
- ④ (Ambient) 認証用ライトキー (16進数16桁) を、ダブルコート(")内に記入してください。
- ⑤ (照度センサ) 処理①で定義した IO1 をアナログ入力に設定します。
- ⑥ (照度センサ) digitalWrite 命令で IO0 を H レベルに設定することで照度センサに電流を流し、analogRead 命令で ESP32-C3 内蔵の 12 ビット A/D コンバータで IO1 の電圧を取得し (処理⑦)、照度値に変換 (処理⑧) してから、IO0 を L レベルに戻して電流を止めます。
- ⑦ 12 ビット A/D コンバータは、入力電圧に応じて 0~4095 の値を出力します。また、初期状態での最大入力電圧は 2.5V なので、およその入力電圧を下式で算出できます。

A/D コンバータの電圧値：

$$\text{電圧(mV)} = \text{ADC 値} \times 2500(\text{mV}) \div 4095(\text{mV})$$

ただし、本 A/D コンバータの入出力特性は、0V を入力しても 0 にならない点や直線的な変換ではない点から、電圧値は目安です。

- ⑧ (照度センサ) 処理⑦で得た電圧値を前述 (照度値の計算方法) の式で照度値に変換します。
- ⑨ (UDP) センサ値を LAN 内に UDP ブロードキャスト送信する処理部です。
- ⑩ (Ambient) センサ値を Ambient へ HTTP POST 送信する処理部です。
- ⑪ (Ambient) センサ値を JSON 形式に変換します。項目 d1 は Ambient に送信するデータ番号です。ここではデータ番号 d1 に照度値を渡しました。
- ⑫ (スリープ) 引数 SLEEP_P(μ秒)で設定した時間、ディープ・スリープを実行します。本コマンド実行後、ディープ・スリープに移行し、約 30 秒後に、復帰 (起動) します。

```

#include <WiFi.h>
#include <WiFiUdp.h>
#include <HTTPClient.h>
#include "esp_sleep.h"
#define PIN_LED_RGB 2
#define PIN_EN 0
#define PIN_AIN 1 ←①
#define SSID "1234ABCD"
#define PASS "password" }②
#define PORT 1024
#define SLEEP_P 30*1000000ul
#define DEVICE "illum_1."
#define Amb_Id "00000" ←③
#define Amb_Key "0000000000000000" ←④

IPAddress IP_BROAD; ←⑤

void setup() {
    led_setup(PIN_LED_RGB);
    pinMode(PIN_AIN, ANALOG); ←⑥
    pinMode(PIN_EN, OUTPUT);
    Serial.begin(115200);
    Serial.println("ESP32C3 LUM");
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, PASS);
    while(WiFi.status() != WL_CONNECTED) {
        led(millis()/50) % 10;
        if(millis() > 30000) sleep();
        delay(50);
    }
    led(0, 20, 0);
    IP_BROAD = WiFi.localIP();
    IP_BROAD[3] = 255;
    Serial.println(IP_BROAD);
}

void loop() {
    digitalWrite(PIN_EN, HIGH); ←⑦
    delay(100);
    float mv = analogRead(PIN_AIN) * 2500. / 4095.; // センサ電圧を変数 mv に保持
    float lux = 100.* mv / 33.; ←⑧
    digitalWrite(PIN_EN, LOW); // センサ用の電源を OFF に
    String S = String(DEVICE) + String(lux, 0); // 送信データ S にデバイス名を代入
    Serial.println(S); // 送信データ S をシリアル出力表示
    WiFiUDP udp; ←⑨
    udp.beginPacket(IP_BROAD, PORT);
    udp.print(S);
    udp.endPacket(); // UDP 送信の終了(実際に送信する)
    if(strcmp(Amb_Id, "00000") == 0) sleep(); // Ambient 未設定時に sleep を実行
    S = "{\"writeKey\":\""+String(Amb_Key)+"\""; // (項目)writeKey, (値)ライトキー
    S += ", \"d1\":"+String(lux)+"}"; // (項目)d1, (値)照度
    HTTPClient http; ←⑩
    http.setTimeout(15000); // タイムアウトを 15 秒に設定する
    String url = "http://ambidata.io/api/v2/channels/"+String(Amb_Id)+"/data";
    http.begin(url); // HTTP リクエスト先を設定する
    http.addHeader("Content-Type", "application/json"); // JSON 形式を設定する
    Serial.println(url); // 送信 URL を表示
    http.POST(S); // センサ値を Ambient へ送信する
    http.end(); // HTTP 通信を終了する
    sleep(); // 下記の sleep 関数を実行
}

void sleep() {
    delay(200);
    led_off(); // (WS2812)LED の消灯
    Serial.println("Sleep..."); // 「Sleep」をシリアル出力表示
    esp_deep_sleep(SLEEP_P); ←⑫ // Deep Sleep モードへ移行
}

```

リスト 3 プログラム

ex03_lum.ino

HTTP クライアント (⑩) が、センサ値 (⑥) を Ambient に HTTP 送信する。送信後、タイマー起動可能なディープ・スリープ状態に移行する。※プログラムを書き込む前に②③④の設定が必要

プログラム④ Wi-Fi コンシェルジ掲示板担当・LCD/表示器

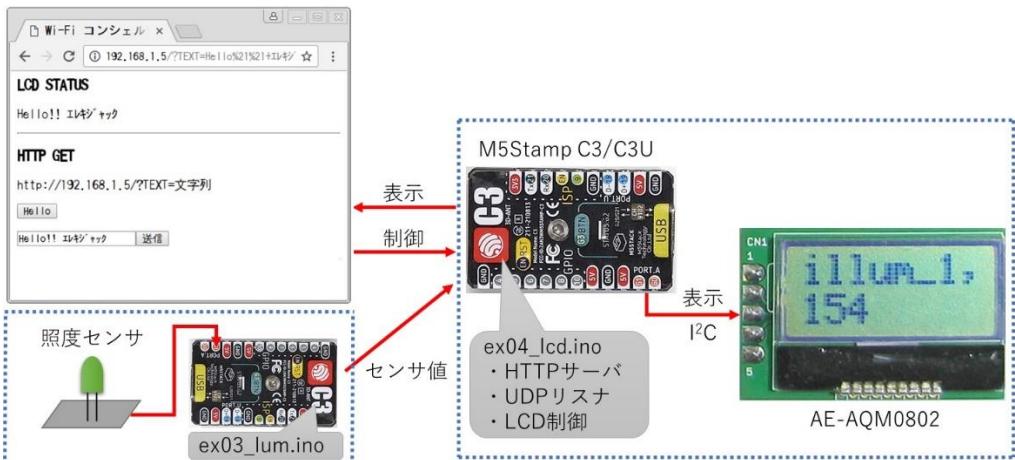


図 3-5 Wi-Fi コンシェルジ掲示板担当・LCD/表示器

HTTP サーバと UDP 受信機能を搭載した Wi-Fi コンシェルジが LCD に受信メッセージを表示

HTTP サーバ機能と UDP 受信機能を搭載した Wi-Fi コンシェルジが LCD (液晶表示器) に受信メッセージを表示するプログラム ex04_lcd を製作します。

同じ LAN 内に接続したパソコンやスマートフォンのインターネット・ブラウザから、メッセージを送信して LCD に表示することや、UDP で送られてきたセンサ値を表示することができます。

LCD/表示器の製作方法

ハードウェアは図 3-5 のようにブレッド・ボード上に I²C インタフェース搭載 LCD モジュール AE-AQM0802 (秋月電子通商製) を実装して製作しました。I²C 接続用に M5Stamp C3/C3U の IO0 を SCL (クロック), IO1 を SDA (データ) として使用します。これらの信号ラインにはプルアップ抵抗が必要ですが、AE-AQM0802 完成品の場合は、あらかじめ 10kΩ でプルアップされているのでそのまま使えます。AE-AQM0802 キットの場合は、背面の PU 部 (2箇所) に半田を盛ってショートし、プルアップを有効にしてください。製作した LCD/表示器の回路図を図 3-6 に、配線図を写真 3-5 に示します。なお、C3U の場合は動作確認用に USB シリアル変換モジュールが必要です。

ソフトウェアは、リスト 4 のプログラム ex04_lcd の①に SSID とパスワードを入力後、M5Stamp C3/C3U に書き込みます。書き込み後、LCD のリセットと初期設定を行うために、一度、M5Stamp C3/C3U モジュールに接続した USB ケーブルを抜いてから指し直してください。

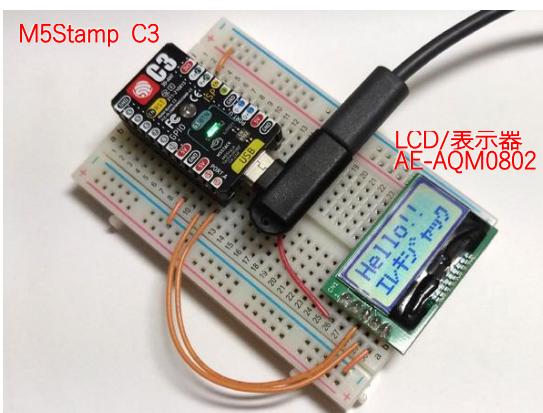


写真 3-4 LCD/表示器の製作例

ブレッド・ボード上に M5Stamp C3 と LCD モジュール AE-AQM0802 を実装した

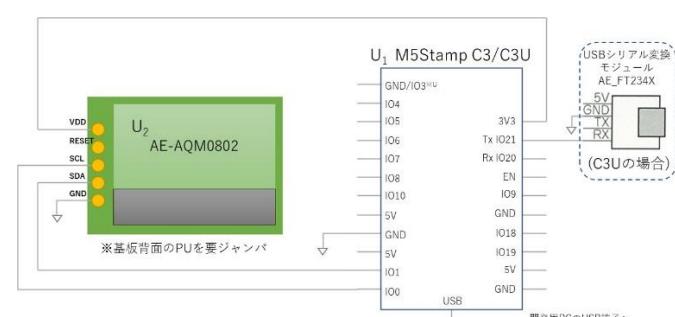


図 3-6 LCD/表示器の回路図

LCD モジュールとは 2 本の信号線で通信が行える I²C インタフェースで接続する。

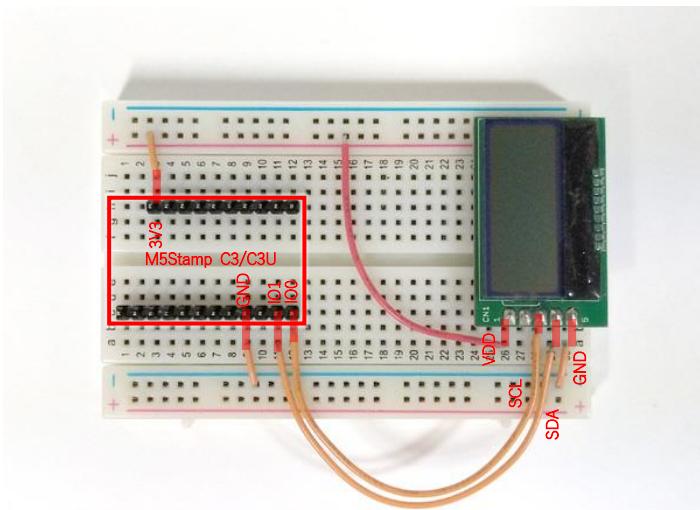


写真 3-6 ブレッド・ボードで製作する LCD/表示器の配線例

本 LCD モジュールは I²C インタフェースを搭載している。M5Stamp C3 の IO0 を SCL (クロック), IO1 を SDA (データ) として通信を行う。信号ラインにはプルアップ抵抗が必要

LCD モジュール AE-AQM0802 の概要

使用する LCD モジュールは、I²C インタフェースを搭載した 8 文字×2 行のキャラクタ表示液晶です。アルファベット、数字のほか、カタカナを含む計 256 文字のフォントを保持しており、表示したい文字コード 0～255 を送るだけで表示できるので、マイコン側の処理負担が少ないのが特徴です。また、8 文字までの自作のキャラクタを作成して表示することもできます。文字コードは、表示可能な ASCII 文字コード 96 文字中 93 文字に互換性があり、カタカナはシフト JIS の半角カナ 64 文字全てに互換性があります。このため、多くの既存システムから容易に文字列を表示できます。

LCD モジュールのガラス基板上には、LCD ドライバ ST7032i (台湾・Sitronix 製) が実装されています。台湾製にも関わらずカタカナ表示に対応しているのは、かつて日本製のドライバ IC が世界中に普及した名残です。例えば、LCD に逆スラッシュ (\) を表示しようとすると ¥マークが表示される点は、日本語版 Windows を使っていない国にとっては紛らわしい仕様です。

LCD にメッセージを送信する方法

起動すると、本機の LCD 画面上に本機の IP アドレスが表示されるので、インターネット・ブラウザのアドレス・バーに「http://(IP アドレス)」を入力します。図 3-7 のような画面が表示されたら、表示したいメッセージを入力して[送信]ボタンをクリックして下さい。半角で「エレキシック IoT CQpb」の 16 文字を入力すると、写真 3-7 のように 8 文字×2 行の文字列として表示します。

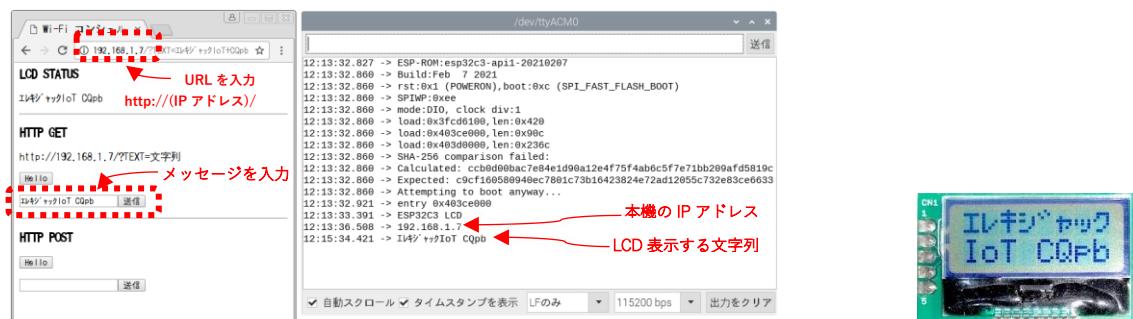


図 3-7 LCD/表示器にメッセージを送信したときの様子

ブラウザのアドレス・バーに「http://(IP アドレス)」を入力して表示される画面でメッセージを入力した

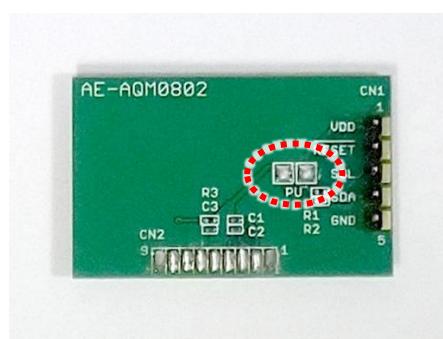


写真 3-5 AE-AQM0802 完成品の裏面

完成品は PU の部分に半田が盛ってあり、10kΩ でプルアップされている。キット品は未処理なので同じように半田を盛ってプルアップする

UDP ブロードキャストを受信する

本書で製作した Wi-Fi ボタン・送信機, Wi-Fi 照度計・送信機, 次節の Wi-Fi 温湿度計・送信機が送信する UDP ブロードキャストのメッセージを受信すると, LCD に表示します。

写真 3-8 は, 左から順に, Wi-Fi 照度計・送信機から照度値 154 lx を受信したときと, Wi-Fi ボタン・送信機からメッセージ「Ping」を受信したとき, Wi-Fi 温湿度計・送信機から温度値 20.3°C と湿度値 59% を受信したときの表示例です。受信するたびに, 表示が切り替わります。図 3-8 は各機器から受信したメッセージを LCD に表示したときの動作ログです。



写真 3-8 UDP ブロードキャスト受信動作の様子

本書で製作した Wi-Fi ボタン・送信機, Wi-Fi 照度計・送信機, Wi-Fi 温湿度計・送信機に対応する

HTTP サーバ搭載 LCD/表示器のプログラム ex04_lcd.ino の内容

HTTP サーバ機能によるセンサ値の送信機能と, UDP 送信機能, スリープ機能を備えたサンプル・プログラム ex04_lcd.ino の処理の手順について, **リスト 4** を用いて説明します。

- ① (Wi-Fi) お手持ちの無線 LAN アクセスポイントの SSID とパスワードを定義します。
- ② (UDP) UDP を受信するリスナーの待ち受けポート番号を定義します。
- ③ (HTTP) HTTP サーバを名称 server で生成します。
- ④ (HTTP) HTTP サーバにアクセスがあったときに実行する関数部です。
- ⑤ (HTTP) アクセス時のクエリに項目 TEXT が含まれていた場合に, TEXT=に続くクエリ値を文字列 String 型の変数 rx に保持します。
- ⑥ (HTTP) ブラウザに応答するための HTML コンテンツを取得し, 文字列 String 型の変数 tx に代入します。コンテンツは同じフォルダの get_html.ino に書かれています。文字列変数には文字 char 型を配列化した型と, String 型があります。文字列を扱う点では同じですが, 使い方は異なり, String 型の方が使いやすい反面, Char の配列型よりもメモリ等を消費します。
- ⑦ (HTTP) HTML コンテンツをインターネット・ブラウザへ送信(応答)します。
- ⑧ (LCD) 文字列変数 rx の内容を LCD に表示します。表示部は別ファイル soft_i2c.ino です。
- ⑨ (UDP) 処理②で定義した UDP ポートを開き, udp を生成します。
- ⑩ (LCD) char 配列型の変数 lcd を定義します。括弧内の数値 49 は配列数で, 最大 48 バイトまで格納できます。文字列の末尾には, 末尾を示す 0 である必要があるので, 格納できる容量は配列数よりも 1 つ減ります。UTF-8 を考慮し, LCD 表示文字数 16 文字よりも多めにしました。
- ⑪ (UDP) 受信した UDP メッセージを文字列変数 lcd に代入します。第 2 引数は最大長です。
- ⑫ (LCD) 文字列変数 lcd の内容を LCD に表示します。

図 3-8 シリアル・モニタの表示例

各機器から受信した UDP メッセージが表示された

```

#include <WiFi.h>
#include <WiFiUdp.h>
#include <WebServer.h>
#define PIN_LED_RGB 2
#define SSID "1234ABCD"
#define PASS "password"
#define PORT 1024 // ①

WebServer server(80); // ③

void handleRoot() {
    String rx, tx;
    led(20, 0, 0);
    if(server.hasArg("TEXT")) { // ④
        rx = server.arg("TEXT"); // ⑤
    }
    tx = getHtml(rx); // ⑥
    server.send(200, "text/html", tx);
    Serial.println(rx); // ⑦
    lcdPrint(rx); // ⑧
    led(0, 20, 0);
}

WiFiUDP udp; // UDP通信用のインスタンスを定義

void setup() {
    led_setup(PIN_LED_RGB);
    lcdSetup(8, 2, 1, 0);
    Serial.begin(115200);
    Serial.println("ESP32C3 LCD");
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, PASS);
    while(WiFi.status() != WL_CONNECTED) {
        led((millis()/50) % 10);
        delay(50);
    }
    led(0, 20, 0);
    lcdPrintIp(WiFi.localIP());
    server.on("/", handleRoot);
    server.begin();
    Serial.println(WiFi.localIP());
    udp.begin(PORT); // ⑨
}

void loop() {
    server.handleClient();
    char lcd[49]; // ⑩
    memset(lcd, 0, 49);
    int len = udp.parsePacket();
    if(len==0) return;
    led(20, 0, 0);
    udp.read(lcd, 48); // ⑪
    udp.flush();
    Serial.print(lcd);
    lcdPrint(lcd); // ⑫
    led(0, 20, 0);
}

```

リスト 4 プログラム

ex04_lcd.ino

HTTP サーバ (③) がクエリ「?TEXT=」を受信 (⑤) すると、LCD を表示制御 (⑧) し、受信した文字列を表示する。また、UDP を受信 (⑪) したときも表示する (⑫)

プログラム⑤ Wi-Fi 温湿度計・送信機

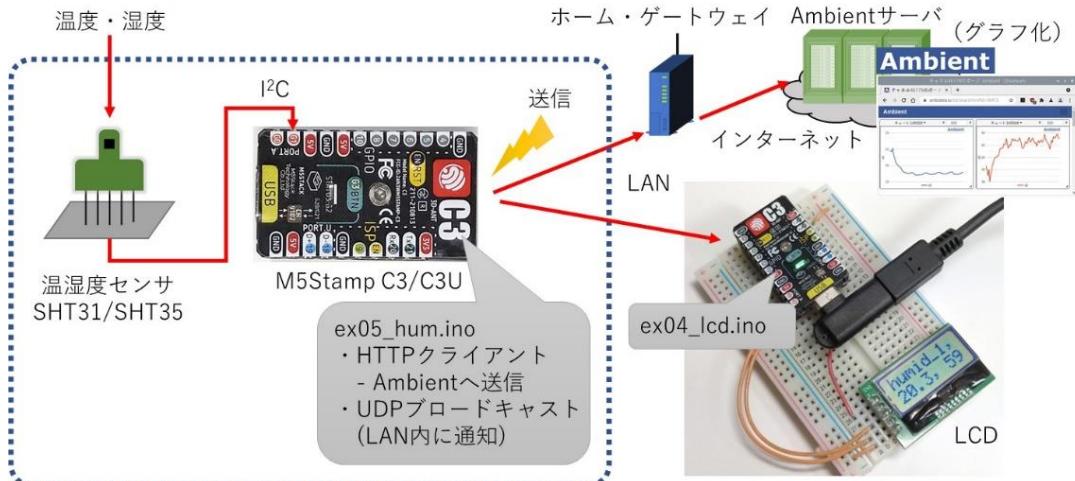


図 3-9 Wi-Fi 温湿度計・送信機

HTTP クライアント機能を搭載した Wi-Fi 送信機がセンサ値を Ambient や LCD (表示器) に送信

HTTP クライアント機能を搭載した Wi-Fi 温湿度計・送信機のプログラム ex05_hum.ino を製作します。Ambient でグラフ表示したり、プログラム④ ex04_lcd.ino の LCD (表示器) にセンサ値を表示したりすることができます。測定したい場所の温度と湿度を手元で確認できるようになります。

Wi-Fi 温湿度計・送信機のハードウェアの製作方法

ハードウェアは、写真 3-9 のように M5Stamp C3 /C3U と温湿度センサ AE-SHT31 をブレッド・ボード上に実装して製作します。電源には単 3 型アルカリ乾電池 3 本を使用し、ディープ・スリープによる節電動作に対応しました。製作した Wi-Fi 温湿度計・送信機の回路図を図 3-10 に示します。

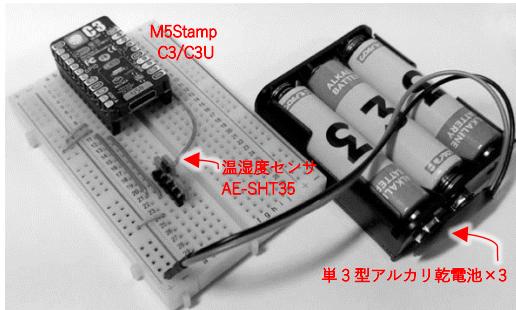


写真 3-9 温湿度計・送信機の製作例

乾電池による長期間動作に対応するハードウェアをブレッド・ボード上に構成する

温湿度センサ AE-SHT31/SHT35 の接続方法

温湿度センサ・モジュールには、センシリオン社（スイス）の SHT35 を実装した AE-SHT35（秋月電子通商製）を使用しました（写真 3-10）。旧製品の SHT31 でも動作するほか、I²C アドレスの変更により M5Stack 製 ENV II / III Unit (SHT30 搭載) にも対応できます。これら SHT3x シリーズの精度は、表 3-1 のように SHT30, 31, 35 の順に高くなります（SHT35 が最も高精度）。

温湿度センサ SHT31 や SHT35 は、センサ出荷時に校正されており、また温度や電圧で自動補正されます。一般的な環境センサの中でも高精度かつ高信頼との定評があり、筆者も同シリーズの製品を 10 年以上前から使い続けていますが、実測値が良く一致している点で信頼感があります。とはいえ、これらは部品の一つであり、測定器ではありません。得られた値の信頼性については他の一般的

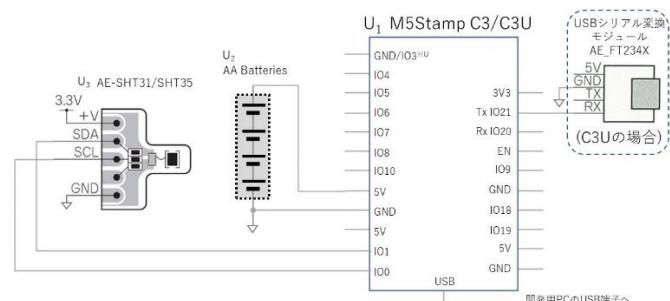


図 3-10 温湿度・送信機の回路図

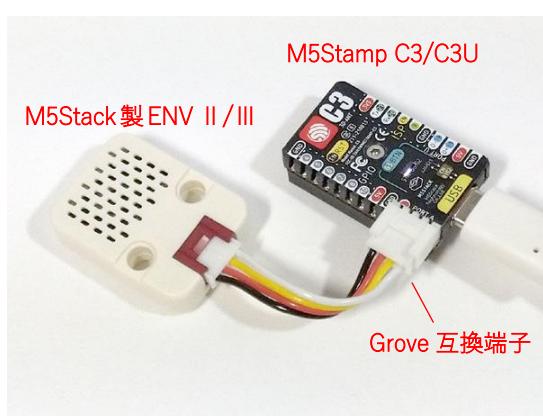
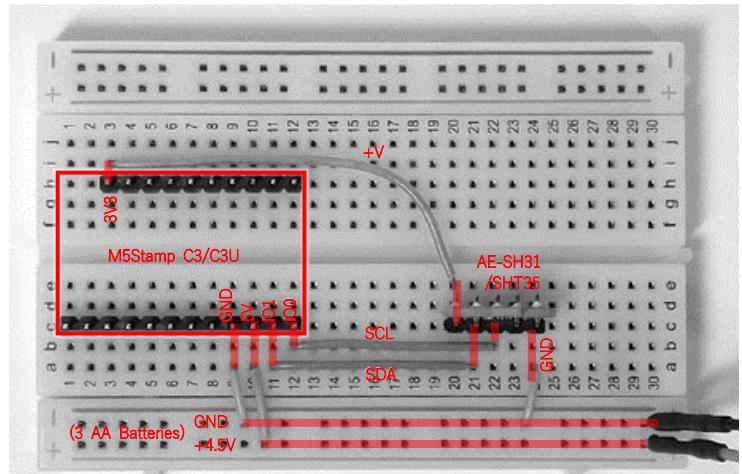
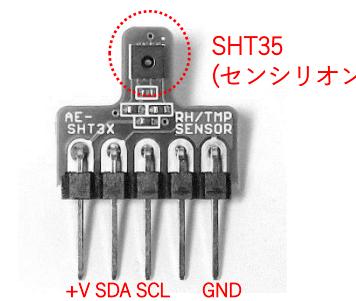
温湿度センサ AE-SHT31 または SHT35 と M5Stamp C3/C3U を I²C インタフェースで接続する

なデバイスと同様に自己責任となる点に注意してください。

I²C 接続用に、M5Stamp C3/C3U の IO0 を SCL (クロック), IO1 を SDA (データ) として使用します。プルアップ抵抗は、各モジュールに内蔵されています。AE-SHT35 または AE-SHT31 を使った時のブレッド・ボードでの配線図を写真 3-11 に示します。なお、シリアル・モニタで M5Stamp C3U の動作確認を行うには USB シリアル変換モジュールが必要です。

M5Stack 製 ENV II / III Unit を使った場合は、写真 3-12 のように、M5Stamp C3/C3U の PORT A に Grove 互換端子を取り付け、Grove ケーブルで接続してください。I²C アドレスを ENV II / III Unit 用に変更するには、プログラム用フォルダ ex05_hum 内の i2c_sht31.ino に書かれている 「#define I2C_sht 0x45」 のアドレス部 0x45 を 0x44 に書き換えます。

図 3-11 は、Wi-Fi 温湿度計が温度値と湿度値を取得し、それらを送信したときの様子です。



```
/dev/ttyACM0
12:11:02.337 -> ESP-ROM:esp32c3-api1-20210207
12:11:02.371 -> Build:Feb 7 2021
12:11:02.371 -> rst:0x5 (DSLEEP), boot:0xc (SPI_FAST_FLASH_BOOT)
12:11:02.371 -> SPIWP:0xee
12:11:02.371 -> mode:DIO, clock div:1
12:11:02.371 -> load:0x3fc0d100, len:0x420
12:11:02.371 -> load:0x403ce000, len:0x990
12:11:02.371 -> load:0x403d0000, len:0x236c
12:11:02.371 -> SHA-256 comparison failed:
12:11:02.371 -> Calculated: ccb0d00bac7e84e1d90a12e4f75f4ab6c5f7e1bb209afdf5819c4
12:11:02.371 -> Expected: c9cf160580040ec7801c73b16423824e72ad12055c732e83ce66332
12:11:02.371 -> Attempting to boot anyway...
12:11:02.473 -> entry 0x403ce000
12:11:02.635 -> ESP32C3 HUM 起動表示
12:11:05.654 -> 192.168.1.255
12:11:05.719 -> humid_1, 24.8, 47.9 24.8°C, 47.9%を（LCD へ） UDP 送信
12:11:05.885 -> Sleep... ESP32-C3 をスリープ状態に設定する

```

写真 3-11 Wi-Fi 温湿度計・送信機の動作の様子
温度値 24.8°C, 湿度値 47.9%を LCD へ UDP 送信後、ESP32-C3 を 30 秒間のスリープ状態に設定した

表 3-1 センシリオン製 SHT3x シリーズの性能比較

項目	SHT30	SHT31	SHT35
湿度測定精度許容差（標準）	±3 %RH	±2 %RH	±1.5 %RH
温度測定精度許容差（標準）	±0.3 °C	±0.3 °C	±0.2 °C
温度の標準条件範囲	0～65 °C	-40～90 °C	-40～90 °C

参考文献：センシリオン SHT3x-DIS データシート Version 3

最新の温度と湿度の LCD 表示と推移のグラフ表示

製作した Wi-Fi 温湿度計を起動すると、30 秒ごとに温度値と湿度値を UDP ブロードキャスト送信します。送信したセンサ値は、同じ LAN 上で動作するプログラム④ ex04_lcd.ino Wi-Fi コンシェルジエ掲示板担当 LCD/表示器で受信して図 3-12 のように表示することができます。ここでは、温度値 21.0°C と湿度 63% が表示されました。

また、温度値と湿度値を Ambient サーバに送信すると、これらの値がクラウド上に保存され、図 3-13 のように、パソコンやスマートフォンでセンサ値の推移をグラフ表示することができます。



図 3-12 最新の温度と湿度の LCD 表示例
Wi-Fi コンシェルジエ掲示板担当 LCD/表示器に温度値 21.0°C と湿度 63% が表示された



図 3-13 温度と湿度の推移のグラフ表示例
温度値と湿度値を Ambient に保存することで、それらの推移をグラフ表示することができる

温湿度センサ用プログラム ex05_hum.ino の内容

サンプル・プログラム ex05_hum.ino の処理の手順について、リスト 5 を用いて説明します。

- ① (Wi-Fi) お手持ちの無線 LAN アクセスポイントの SSID とパスワードを記入してください。
- ② (Ambient) Ambient で取得した送信機ごとのチャネル ID (整数値) と、認証用ライトキー (16 進数 16 衔) を、ダブルコート ("") 内に記入してください。
- ③ (温湿度センサ) SHT3x との I²C 通信設定の処理部です。第 1 引数は I²C インタフェースの SDA 信号用のポート番号 1、第 2 引数は SCL 信号用のポート番号 0 です。Arduino IDE 上のタブ「i2c_sht31」をクリックすると shtSetup を含むプログラムが表示されます (コラム参照)。
- ④ (温湿度センサ) SHT3x との I²C 通信設定の処理部です。取得した温度値を変数 temp に、湿度値を変数 hum に代入します。
- ⑤ (UDP) センサ値を LAN 内に UDP ブロードキャスト送信する処理部です。
- ⑥ (Ambient) センサ値を Ambient へ HTTP POST 送信する処理部です。
- ⑦ (スリープ) 引数 SLEEP_P(μ秒) で設定した時間、ディープ・スリープに移行します。

```

#include <WiFi.h>
#include <WiFiUdp.h>
#include <HTTPClient.h>
#include "esp_sleep.h"

#define PIN_LED_RGB 2
#define SSID "1234ABCD" ]①
#define PASS "password"
#define PORT 1024
#define SLEEP_P 30*1000000ul
#define DEVICE "humid_1."
#define Amb_Id "00000"
#define Amb_Key "0000000000000000" ]②

IPAddress IP_BROAD;

void setup() {
    led_setup(PIN_LED_RGB);
    shtSetup(1,0); ←③
    Serial.begin(115200);
    Serial.println("ESP32C3 HUM");

    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID,PASS);
    while(WiFi.status() != WL_CONNECTED) {
        led((millis()/50) % 10);
        if(millis() > 30000) sleep();
        delay(50);
    }
    led(0,20,0);
    IP_BROAD = WiFi.localIP();
    IP_BROAD[3] = 255;
    Serial.println(IP_BROAD);
}

void loop() {
    float temp = getTemp(); ]④
    float hum =getHum(); ]④
    if(temp < -100. || hum < 0.) sleep();

    String S = String(DEVICE);
    S += String(temp,1) + ", ";
    S += String(hum,1);
    Serial.println(S);
    WiFiUDP udp;
    udp.beginPacket(IP_BROAD, PORT);
    udp.println(S);
    udp.endPacket();
    if(strcmp(Amb_Id,"00000") == 0) sleep(); ]⑤

    S = "{$writeKey":""+String(Amb_Key); // (項目)writeKey, ( 値)ライトキー
    S += "{$d1":$temp,2}; // (項目)d1,(値)温度
    S += "{$d2":$hum,2}"; // (項目名)d2,(値)湿度
    HTTPClient http; // HTTP リクエスト用インスタンス
    http.setConnectTimeout(15000); // タイムアウトを 15 秒に設定する
    String url = "http://ambidata.io/api/v2/channels/" +String(Amb_Id)+"/data";
    http.begin(url); // HTTP リクエスト先を設定する
    http.addHeader("Content-Type", "application/json"); // JSON 形式を設定する
    Serial.println(url); // 送信 URL を表示
    http.POST(S); // センサ値を Ambient へ送信する
    http.end(); // HTTP 通信を終了する
    sleep(); // 下記の sleep 関数を実行
}

void sleep(){
    delay(200);
    led_off();
    Serial.println("Sleep..."); // 「Sleep」をシリアル出力表示
    esp_deep_sleep(SLEEP_P); ←⑦
}

```

リスト 5 プログラム

ex05_hum.ino

温湿度センサ値 (④) を UDP 送信する (⑤). 送信後、タイマー起動可能なディープ・スリープ状態に移行する (⑦). ※プログラムを書き込む前に①②の設定が必要

コラム 2：温湿度センサ SHT3x 用 I²C インタフェース部プログラム

温湿度センサ用プログラム ex05_hum.ino には、shtSetup, getTemp, getHum の 3 つの I²C インタフェース用の関数が使われています。これらの関数のプログラムは、フォルダ ex05_hum 内に i2c_sht31.ino として収録しました。本プログラムの通信部について、以下に説明します。

- ⑧ 温湿度センサ SHT3x の I²C アドレス 0x45 を定義します。
- ⑨ 温度を取得する関数 getTemp を定義します。SHT3x は一度の通信で温度値と湿度値を取得することができるので、本プログラムでは getTemp で両方の値を取得するようにしました。
- ⑩ I²C アドレス 0x45 を送信し、SHT3x との通信を開始します。
- ⑪ SHT3x の測定コマンド 2C 06 を送信します。
- ⑫ 送信を終了します。処理⑩と⑪の送信が実際に行われるのは、このタイミングです。
- ⑬ 測定待ち時間です。SHT3x が測定を行う 15ms 以上の時間、待機する必要があります。
- ⑭ 測定の完了した頃を見計らって、測定結果データ 6 バイトの受信を要求します。
- ⑮ 受信した 6 バイトのうち先頭 2 バイトは温度です。変数 temp に代入します。
- ⑯ 4~5 バイト目が湿度です。変数 hum に代入します。
- ⑰ temp を温度値に変換して応答します。湿度値は変数 _i2c_sht31_hum に保持します。
- ⑱ 処理⑰で保持した湿度値を取得するための関数 getHum です。
- ⑲ I²C インタフェースを Wire.begin 命令で初期化します。引数は処理③のポート番号です。

なお、収録したプログラム i2c_sht31.ino には、これらの処理に加え、I²C 通信に失敗したときの処理が含まれています。

```
#include <Wire.h>
#define I2C_sht 0x45 ←⑧
float _i2c_sht31_hum;
float getTemp() { ←⑨
    int temp, hum;
    Wire.beginTransmission(I2C_sht); ←⑩
    Wire.write(0x2C); ←⑪
    Wire.write(0x06);
    Wire.endTransmission(); ←⑫
    delay(18); ←⑬
    Wire.requestFrom(I2C_sht, 6); ←⑭
    temp = Wire.read(); ←⑮
    temp <= 8;
    temp += Wire.read(); ←⑯
    Wire.read();
    hum = Wire.read(); ←⑯
    hum <= 8;
    hum += Wire.read(); ←⑯
    Wire.read();
    _i2c_sht31_hum = (float)hum / 65535. * 100.; ←⑰
    return (float)temp / 65535. * 175. - 45.; ←⑰
}
float getHum() { ←⑱
    return _i2c_sht31_hum;
}

void shtSetup(int sda, int scl) { ←⑲
    delay(2);
    Wire.begin(sda, scl);
    delay(18);
}
```

プログラム i2c_sht31.ino (コラム用)
フォルダ ex05_hum 内の i2c_sht31.ino の異常処理を省いたプログラム

4章 ESP32-C3 マイコンを使用した応用システム・プログラム

応用例① センサ・ネットワーク・システム

1章～3章で製作したWi-Fiボタン・送信機、Wi-Fi照度計・送信機、Wi-Fi温湿度計・送信機が送信する各種センサ値を、ラズベリー・パイで受信し、CSV形式で保存したり、HTTPでLAN内に情報を共有したりすることができるセンサ値データ活用システムを製作します（図4-1）。

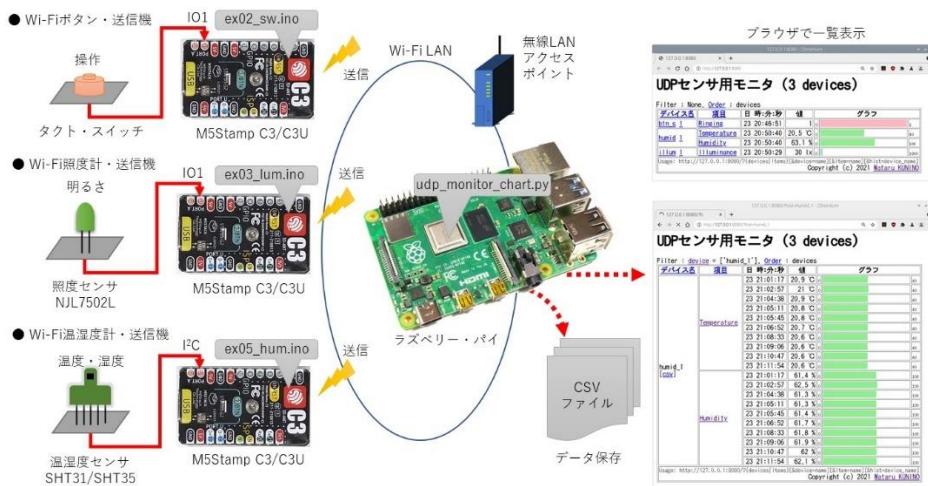


図 4-1 センサ・ネットワーク・システム

1章～3章で製作した送信機のセンサ値を受信し、CSV形式で保存したり、LAN内に情報共有したりすることができるセンサ値データ活用システム

システムの概要

Wi-Fiボタン・送信機（プログラム② ex02_sw.ino）や、Wi-Fi照度計・送信機（プログラム③ ex03_lum.ino）、Wi-Fi温湿度計・送信機（プログラム⑤ ex05_hum.ino）が送信するセンサ値を、ラズベリー・パイで受信し、図4-2のような情報をLAN内に共有しつつ、CSVファイルに保存するシステムを製作します。

作成したCSVファイルは、Microsoft Excelなどの表計算ソフトに入力することで、時刻を横軸にした図4-3のようなグラフが作成でき、センサ値データの活用範囲が広がります。

ラズベリー・パイ上では、プログラム udp_monitor_chart.py で各機器が送信するセンサ値を受信し、センサ値を棒グラフ表示機能したり、複数のデバイスから得られた情報を一覧表示したり、同じセンサを搭載した複数のデバイスから同じ測定項目のみを比較表示したり、受信履歴を表示したり、CSVデータをダウンロードしたりすることができます。

UDPセンサ用モニタ・プログラム udp_monitor_chart.py の実行方法

ラズベリー・パイ用プログラム udp_monitor_chart.py は tools フォルダに収録しました。ラズベリー・パイにダウンロードして実行するには、LXTerminal を起動し、以下のコマンドを入力します。

```
git clone https://bokunimo.net/git/esp32c3 ↵
cd esp32c3/tools ↵
./udp_monitor_chart.py ↵
```

ラズベリー・パイ上で、インターネット・ブラウザを起動し、下記の URL（localhost のポート 8080）をアドレス・バーに入力すれば、受信したセンサ値を確認することができます（図4-2）。

```
http://127.0.0.1:8080/
```

また、同じ LAN 内の他のインターネット・ブラウザ（スマホなど）からアクセスしたい場合は、上記の IP アドレス「127.0.0.1」の部分をラズベリー・パイの IP アドレスに変更してください。

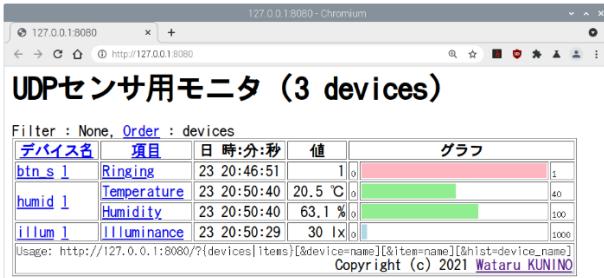


図 4-2 UDP で受信したセンサ値データを HTTP サーバで LAN 内に共有する応用システム
同じ LAN 内のスマートフォンなどのインターネット・ブラウザでもアクセスできる

プログラム udp_monitor_chart.py の便利機能

UDP センサ用モニタ・プログラム udp_monitor_chart.py の便利機能について説明します。

① 棒グラフ・複数デバイス一覧表示機能

図 4-2 のような本プログラムのトップ画面です。複数のデバイスから得られたセンサ値を棒グラフで一覧表示します。表示対象のデバイスは、プログラム内の辞書型変数 sensors に定義しています。Wi-Fi ボタン・送信機のプログラム② ex02_sw.ino は「btn_s」、Wi-Fi 照度計・送信機のプログラム③ ex03_lum.ino は「illum」、Wi-Fi 温湿度計・送信機のプログラム⑤ ex05_hum.ino は「humid」です。

グラフ表示するときに必要なグラフ軸の最小値と最大値は、プログラム内の辞書型変数 csvs_range で定義しました。Wi-Fi 温湿度計・送信機のプログラム⑤ ex05_hum.ino の温度値 Temperature は最小値 0°C～最大値 40°C、湿度値 Humidity は最小値 0%～最大値 40%です。

② デバイス選択機能

一覧表示画面（図 4-2）の「デバイス名」に列に表示された文字（例えば humid）をクリックすると選択したデバイスのみを抽出して一覧表示します。ブラウザのアドレス・バーに、以下のように device= に続いてデバイス種別を入力することで、表示指示することも出来ます。

```
http://127.0.0.1:8080/?device=humid
```

③ 受信履歴を表示機能

一覧表示画面（図 4-2）の「デバイス名」に列に表示された数字（例えば humid_1 の「1」）をクリックすると、受信履歴を表示します。受信したセンサ値の履歴を棒グラフで表示します。表示する履歴件数はプログラム内の整数型変数 HIST_BUF_N で定義しており、初期値は 10 件です。以下のようにアドレス・バーにデバイス名を指定することも出来ます。

```
http://127.0.0.1:8080/?hist=temp_1
```

④ CSV ダウンロード機能

前記の受信履歴の表示画面の「デバイス名」内の[CSV]をクリックすると、選択したデバイスについて、これまでに受信した CSV 形式の全データをダウンロードすることができます。

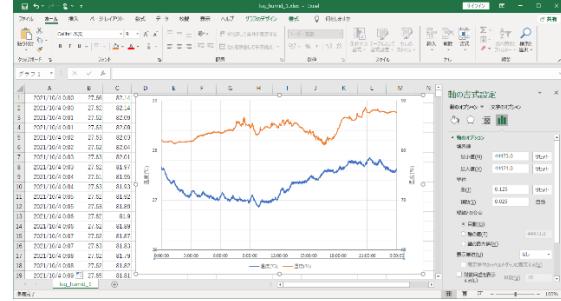
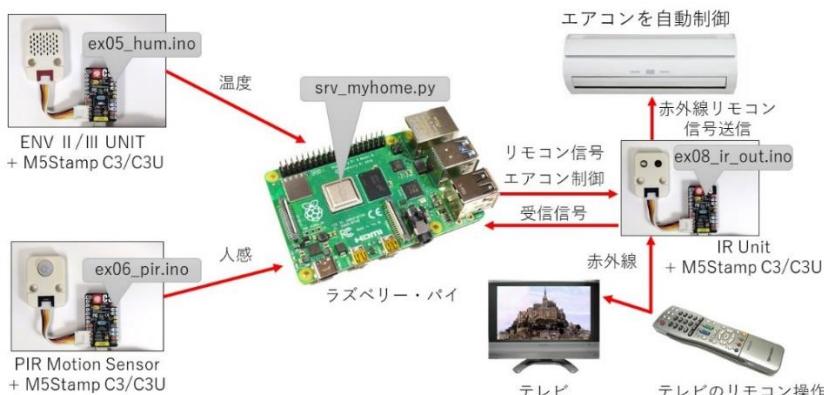


図 4-3 Microsoft Excel でグラフ化
表計算ソフトに入力することで、時刻を横軸にしたグラフが作成でき、センサ値データの活用範囲が広がる

応用例② 自分だけの My ホーム・オートメーション・システム



使用機器(本例のみ)

- M5Stamp C3 または C3U Mate × 3, ● ラズベリー・パイ × 1
- ENV II/III Unit, PIR Motion Sensor, IR Unit (M5Stack 製) 各 1
- 5V 電源 (USB 出力 AC アダプタなど) × 3

Wi-Fi 人感センサと Wi-Fi 温湿度センサを用い、居住者が在室中に、室内の温度が 28°C 以上または 15°C 以下になった時に、赤外線リモコン信号の送信によってエアコンの運転を開始する My ホーム・オートメーション・システムを製作します (図 4-4)。

家庭内の機器を自動制御するホーム・オートメーションは 1990 年ごろから注目され、2000 年代に通信機能が、2010 年代に統合制御機能が含まれ、実用化されています。また、技術のモジュール化によって安価に実現できるようになりましたが、普及が進んでいるように感じられません。

課題の一つは、多種多様な機器と用途に合わせた汎用的なアプリケーション・ソフトウェアの開発が難しいことです。ところが、自分用に特化すれば、汎用性の課題を回避することが出来、簡単にシステムを製作することができます。そこで、本書では、自分の機器や用途に合わせて、自分でソフトウェア設計する My ホーム・オートメーション・システムの製作例を紹介します。

M5Stack 既製品とサンプル・プログラムで簡単にハードウェアを製作

ハードウェアは、M5Stamp C3/C3U の Grove 互換端子に M5Stasck 社が販売している拡張ユニットを取り付けて製作します。写真 4-1 に、(a) M5Stack 製 ENV II/III Unit, (b) PIR Motion Sensor, (c) IR Unit を接続した M5Stamp C3 の様子を示します。

M5Stamp C3/C3U 用のソフトウェアについては、各写真の下部に記載したそれぞれのサンプル・プログラムに、無線 LAN アクセスポイントの SSID とパスワードを記入してから、書き込みます。



写真 4-1 M5Stack 既製品を M5Stamp C3/C3U に接続する

M5Stamp C3/C3U の Grove 互換端子に M5Stasck 社が販売している拡張ユニット (M5Stack 製 ENV II/III Unit, PIR Motion Sensor, IR Unit) を取り付けて製作した

図 4-4 My ホーム・オートメーション・システム

居住者が在室中に、室内の温度が 28°C 以上または 15°C 以下になった時にエアコンの運転を開始する My ホーム・オートメーション・システム

プログラム srv_myhome.py の機能

ラズベリー・パイで動作するホーム・オートメーション用プログラム srv_myhome.py は、 tools フォルダ内に収録しました。以下に本プログラムの主な機能と設定方法について説明します。

① 赤外線リモコン送信機能

写真 4-1 (c) の IR Unit とプログラム ex08_ir_out.ino を書き込んだ M5Stamp C3/C3U から、エアコンの運転を開始するリモコン信号を送信する機能です。

この機能を使用するには、 srv_myhome.py の ip_irrc に IP アドレスを設定します。IP アドレスは、 srv_myhome.py の動作ログ（図 4-5 の(C)）、または M5Stamp C3/C3U を起動したときのシリアル出力で表示されるので、どちらかを確認してください。

IP アドレスを初期値 ip_irrc = '192.168.0.XXX' のままにしておき、ラズベリー・パイの GPIO ポート 4 に赤外線 LED を接続して、リモコン信号を送信することもできます。

リモコン信号の方式は家製協方式 (AEHA) と NEC 方式に対応しており、IR_TYPE で 'AEHA' または 'NEC' を設定してください。

リモコン信号がわからないときは、 srv_myhome.py の動作ログで確認するか、インターネット・ブラウザで ex08_ir_out.ino が動作する M5Stamp C3/C3U にアクセスし、お手持ちのリモコンを操作してから [受信データの取得] ボタンで確認してください。初期値 AC_ON = 'AA,5A,CF,10,00,11,20,3F,18,B0,00,F4,B1' は、筆者が保有するエアコン（シャープ製）で確認しました。動作しない場合は、AC_ON の値を変更してください。

② 在室検知機能（人感センサ）

写真 4-1 (b) の PIR Motion Sensor（人感センサ）の接続とプログラム ex06_pir.ino を書き込んだ M5Stamp C3/C3U が、人体などの動きを検出したときに部屋に人が居る（在室）を検知する機能です。デバイス名 pir_s_1～pir_s_3 の人感センサからの送信に対応しています。

③ 在室検知機能（赤外線リモコン）

写真 4-1 (c) の IR Unit の接続とプログラム ex08_ir_out.ino を書き込んだ M5Stamp C3/C3U が、テレビのリモコン操作を検出したときに、在室を検知する機能です。

IR Unit は赤外線リモコン信号を受信することもでき、srv_myhome.py は、TV_CODE に設定したテレビのリモコン信号を検出したときに在室とみなします。

シャープ製のテレビの場合は '48,AA,5A,8F'、パナソニック製のテレビの場合には、'48,02,20,80' を設定してください。先頭の 48 は、リモコンのコード長（ビット）です。ここでは 48 ビットずなわち 6 バイトを指定しました。続く AA,5A,8A や 02,20,80 がリモコン信号の先頭 3 バイトです。受信したリモコン信号 6 バイトのうち先頭 3 バイトが一致すれば、残る 3 バイトに何が入っていてもテレビのリモコンであると判定します。

④ 温度管理機能

写真 4-1 (a) の ENV II/III Unit の接続とプログラム ex05_hum.ino を書き込んだ M5Stamp C3/C3U が送信する温度値が、28°C以上または15°C以下になった時に、エアコンの運転を開始する機能です。エアコンの制御判定に用いる温度値は、srv_myhome.py の ALLOWED_TEMP 値で設定します。初期値 ALLOWED_TEMP = [28,15] は、28°C以上と15°C以下のときにエアコンの運転する場合の設定例です。

⑤ 通知機能 (LINE)

在室かつエアコンの運転制御が必要になったときに、スマートフォンの LINE アプリに「室温が xx°Cになりました」と通知する機能です。 srv_myhome.py の LINE_TOKEN に LINE で取得したトークンを入力してください。初期値のままだと通知しません。トークンの取得方法はプログラムにコメントとして記載しました。

また、Gmail を使った通知機能も備えています。こちらも初期値では通知しません。

⑥ 対象デバイス番号の絞り込み機能

各部屋に機器を設置する場合、別の部屋に設置した人感センサ・送信機の情報を受信しても在室と判断したり制御したりしないようにする必要があります。本書で製作した送信機のデバイス名は、デバイス種別 5 文字 + 「_」 + デバイス番号の 7 文字で構成されており、末尾のデバイス番号は、同じ種類の複数の機器から個々を特定するのに使用します。

ラズベリー・パイ用プログラム srv_myhome.py には、検知したいデバイス番号を変数 ROOM に設定しています。初期値 ['1','2','3'] では、デバイス番号 1~3 を検知します。

各送信機のデバイス名は、各プログラム内で#define DEVICE で定義します。プログラム ex06_pir.ino のデバイス名の初期値は pir_s_1 で、デバイス種別 pir_s とデバイス番号 1 となっています。例えば、ex06_pir.ino のデバイス名に pir_s_5 を設定し、他の pir_s を受信したくない場合は、srv_myhome.py の変数 ROOM を['5']に変更してください。

⑦ エアコンつけっぱなし防止機能

エアコンを自動で ON した状態で、在室状態が 10 分以上、不在となっていた場合に、エアコンを停止します。本システムでは、自動で ON したあとに適温になっても自動では停止させず、不在が続いたときのみ自動で OFF する仕様にしました。

⑧ 温湿度センサの受信状態の通知機能

温湿度センサから（電池切れなどで）2 時間以上、受信できなかった場合に、メッセージ「センサの信号が○○時間ありません」を LINE またはメールで送信します。

My ホーム・オートメーション用プログラム srv_myhome.py の動作例

図 4-5 は動作結果の一例です。Wi-Fi 温湿度計・送信機から温度値と湿度値を受信(A)し、人感センサから在室状態を受信(B)すると、在室を示す変数 stay の値が True になります（テレビのリモコン信号を受信(C)しても True になる）。在室中に、Wi-Fi 温湿度計・送信機の温度値が 30°C を超過するとメッセージを表示し(D)，エアコンを ON する制御を行いました(E)。

なお、エアコンやテレビのリモコン信号コードは、ログ(C)のように、日時、デバイス名、IP アドレス、リモコン信号長（ビット）、リモコン信号の順で表示します。設定前に本プログラムを起動し、これらの値を参考に設定し、プログラムを起動し直すと効率的に設定することができます。

```
pi@raspberrypi:~/myMimamori $ ./srv_myhome.py
message=件名:i.myMimamoriHome, 起動しました
RC, Conditioner, http://192.168.1.8/?TYPE=0
Listening UDP port 1024 ...
2022/01/29 20:06, humid_1, 192.168.1.9, 18.6, 64.4, stay=False, bell=0, temp=18.6(0)
2022/01/29 20:07, humid_1, 192.168.1.9, 20.3, 60.9, stay=False, bell=0, temp=20.3(0)
2022/01/29 20:08, humid_1, 192.168.1.9, 20.3, 61.4, stay=False, bell=0, temp=20.3(0)
2022/01/29 20:08, pir_s_1, 192.168.1.7, 1, 1, stay=True, bell=1 (A)
2022/01/29 20:08, ir_rc_1, 192.168.1.8, 48, AA, 5A, 8F, 12, 15, E1, stay=True, bell=1 (B)
2022/01/29 20:09, humid_1, 192.168.1.9, 33.8, 63.4, stay=True, bell=2, temp=33.8(2)
message=件名:i.myMimamoriHome 警告レベル=2, 室温が 33.8°Cになりました (C)
RC, Conditioner, http://192.168.1.8/?IR=104, AA, 5A, CF, 10, 00, 11, 20, 3F, 18, B0, 00, F4, B1 (D)
(E)
```

図 4-5 ホーム・オートメーションの動作例

温度値と湿度値を受信(A)し、人感センサにより在室状態 stay が True に (B) なった状態で、30°C を超過すると(D)、エアコンを ON する制御を行う(E)

Appendix 試作に便利。ブレッド・ボード上に周辺回路を実装してみる

ESP32-C3 の IO ポート数は 22 本と、ESP32 の 34 本に比べて少ないので、ESP32-C3 用の開発ボードのピン数は、ESP32 よりも少い傾向があります。このため、ブレッド・ボード上に周辺回路を実装するスペースが確保しやすい利点があります。

一般的な 400 穴・最大 60 ピンのブレッド・ボード上に、M5Stamp C3（占有ピン数 22 ピン）を実装した場合、周辺回路用に 38 ピン分を充てることができます（うち 2 ピンは M5Stamp のアンテナ近く）。また、IO4（C3U の場合は IO4 と IO3）を使用しない場合は、ブレッド・ボード側に実装するピン・ヘッダを 10 ピン×2 本にすることで、連続した 40 ピンを周辺回路用に充てれます。

Wi-Fi コンシェルジェ [音声アナウンス担当]

写真 4-2 は、音声合成 LSI とオーディオ・アンプを M5Stamp C3/C3U の周辺回路として実装した Wi-Fi 搭載の音声アナウンス端末の製作例です。M5Stamp C3/C3U の Tx 端子を AquesTalk Pico LSI ATP3012 の RXD 端子（2 番ピン）に入力し、ATP3012 の音声出力（15 番ピン）をオーディオ・アンプ NJM386B で増幅してスピーカを鳴らします。ATP3012 と NJM386B の電源は、M5Stamp C3/C3U の 5V 端子から供給しました。スピーカからのノイズが気になる場合は、ブレッド・ボード左側の電源ラインに 1000~2000 μ F の低 ESR コンデンサを入れると改善できます。

ソフトウェアは、ex09_talk.ino を Arduino IDE で開いて、書き込んでください。HTTP サーバ機能を搭載しているので、インターネット・ブラウザから音声データを入力することができます。ESP-WROOM-02（旧チップ ESP8266 を搭載）用の回路図や使い方を書籍「超特急 Web 接続! ESP マイコン・プログラム全集」で紹介しているので、ぜひ参考にしてください。



Wi-Fi コンシェルジェ [カメラ担当]

写真 4-3 は LCD/表示器と、カメラの電源供給用の DC ジャックや FET を実装した Wi-Fi 搭載端末の製作例です。M5Stamp C3/C3U の IO7 をカメラの RXD 端子へ、IO8 を TXD へ接続し、IO10 を FET のゲートに入力します。LCD/表示器の接続方法は Wi-Fi LCD/表示器と同じです。

ソフトウェアは、ex10_cam.ino です。撮影した写真は、ブラウザで閲覧することができます。ESP-WROOM-02 用の回路図や使い方は、前述の書籍で紹介しています。



写真 4-2 音声のアナウンスを担当する Wi-Fi コンシェルジェ端末

ブラウザから入力した音声データをスピーカから出力する機器をブレッド・ボードで製作してみた

使用機器(本例のみ)

- M5Stamp C3 または C3U Mate
- AquesTalk Pico LSI ATP3012
- NJM386B, スピーカ, ほか

写真 4-3 カメラによる写真撮影を担当する Wi-Fi コンシェルジェ端末
カメラで撮影した写真をブラウザで閲覧する機器をブレッド・ボードで製作してみた

使用機器(本例のみ)

- M5Stamp C3 または C3U Mate
- Grove - Serial Camera Kit
- AE-AQM0802, ほか

むすび

本書では、IoT 対応製品の低価格に貢献できる Wi-Fi 搭載マイコン ESP32-C3 を使って、IO 制御プログラミングの基礎から IoT 化、システム化までの一例を説明しました。また、ESP32-C3 を実装した M5Stamp C3 を使い、ブレッド・ボード上に周辺回路を実装した試作方法についても説明しました。本書を読んでいただいた皆様が趣味や仕事に活かしていただくことで、様々な IoT 対応機器が登場し、また統合制御するシステム・アプリケーションが次々に登場し、いまよりも便利で、安全で、何より楽しい暮らしができる日が来るなどを願っています。

参考文献

本書の作成にあたり、下記の文献を参考にしました。

- ・スピード実習パーツセット説明書 (<https://shop.cqpub.co.jp/hanbai/books/I/I000215/setsumeisyo.pdf>)
- ・トランジスタ技術 2017 年 3 月号 (CQ 出版社)
- ・トランジスタ技術 2016 年 9 月号 (CQ 出版社)
- ・ESP32, ESPRESSIF SYSTEMS (<https://www.espressif.com/en/products/socs/esp32>)
- ・Arduino, arduino.cc (<https://www.arduino.cc/>)

権利情報

本書の著作権は筆者（国野 亘）に帰属します。

著者（国野 亘）に無断で本書の複製や改変、販売、配布などを行うことを禁止いたします（適切な引用や、筆者からの承諾を得ていた場合を除く）。

違反した場合、当方や本書の関係者が被った損害額に加え、調査・相談・訴訟等に要した全費用と将来に得られたと推測できる被害額等を弁償していただく場合があります。

本書の掲載事項によって生じたいかなる損害についても、著作者は一切の責任を負いません。すべて自己責任にてご利用ください。

本文中では、製品名、会社名の商標表示を省略しています。Wi-Fi™ は Wi-Fi Alliance の登録商標です。M5Stack, M5Stamp, LINE, Ambient, その他についても、各社の登録商標または商標等です。

Copyright (C) 2022-2023 国野 亘 (<https://bokunimo.net/>)

履歴

2021 年 12 月～2022 年 2 月	執筆期間
2022 年 2 月 6 日	執筆完了
2023 年 5 月 2 日	Web 公開

by bokunimo.net