ESP32-C3M-TRY
(ESP32-C3-MINI-1 開発ボード)
取扱説明書

マイクロファン
http://www.microfan.jp/
https://store.shopping.yahoo.co.jp/microfan/
https://www.amazon.co.jp/s?me=A28NHPRKJDC95B

2023 年 7 月

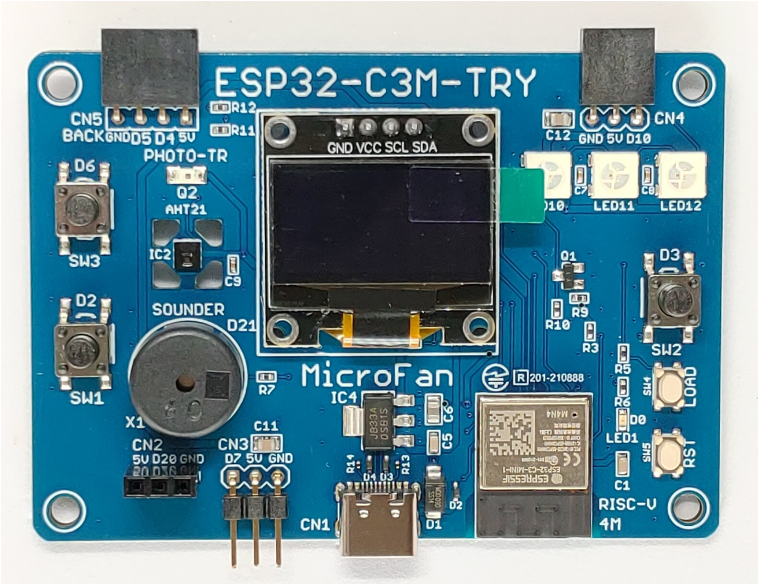
目次

表目次

図目次

第 1 章

ESP32-C3M-TRY の紹介



1.1 製品概要

組込み系のプログラミングの学習や、各種のセンサーを使用した実験など、様々な分野で ESP32 系の開発ボードを使用したプログラミングが行われています。このような用途では、ブレッドボード上に開発ボードとセンサー類を配置し、ジャンプワイヤーで配線して回路を構成することが多いのですが、回路の作成が面倒であったり、配線を間違えたり、接触が不安定だったりと、様々な問題に遭遇します。

ESP32-C3M-TRY は典型的なプログラミングの学習や実験に使用される、表示装置、センサー、スピーカー、スイッチなどを基板上に装備し、ESP32-C3M-TRY だけで多用な用途で利用することができるように設計されています。

ESP32 系のプログラミングに慣れて、様々なアプリケーションの構築を考えている方や、これから様々なセンサーなどを使用した ESP32 系のプログラミングを学びたい方に、ESP32-C3M-TRY

は最適な開発ボードとなっています。

ESP32-C3M-TRY は以下のような特徴を持っています。

- ESP32-C3-MINI-1(RISC-V の 4M Flash 版) を搭載しています。
- ESP32-C3-MINI-1 は従来の ESP32(Xtensa) よりも消費電力が少なく、ESP8266 よりも性能が高いモジュールとなっています。
- ネットワークと接続するための WiFi や Bluetooth のネットワーク機能を利用できます。
- 出力として OLED ディスプレイやカラー LED、圧電スピーカーを搭載しています。
- 温度、湿度、明るさ、加速度センサーを搭載しています。
- 複数のタクトスイッチを搭載しています。
- MicroPython や Arduino でプログラミングを楽しむことができます。
- Type-C の USB インターフェースを装備しています。
- ドロップアウトが 300mV と少ない 1.5A の電圧レギュレータを搭載し、ESP32-C3-MINI-1 に安定した電源を供給できます。
- 超音波距離センサー、人感センサー、RC サーボ、カラー LED マトリックスなどを接続できる接続端子を備えています。

1.2　購入・利用上の注意

ESP32-C3M-TRY には、CN2-CN5 に接続される超音波センサー、人感センサー、RC サーボ、カラー LED ボードなどは含まれていませんので別途お買い求めください。

1.3　マニュアルの記載内容に関して

ESP32-C3-MINI-1 やそれに関連するハードウェアやソフトウェアは、機能の追加や改良が頻繁に行われているため、本文書で提供している情報は、ESP32-C3M-TRY の購入者の利用時にはすでに古い情報になっている可能性があります。そのため、本文書で示している内容と異なる部分があったり、本文書で示している手順ではうまく動作しないことがあることと、その場合には、各自で対処方法を調査・確認していただく必要があることをご承知おきください。

本マニュアルの記載内容と、ご提供するソフトウェア、ハードウェアに差異がある場合には、ご指摘によりマニュアルの迅速な訂正を心がけますが、ご提供するソフトウェア、ハードウェアの現品の仕様が優先されます。

お伝えする内容と本質的な問題がない場合には、本マニュアルには、旧バージョンの製品の写真や他製品の写真などがそのまま使用されている場合がありますのでご承知おきください。

本書に記載されている内容に基づく作業、運用などにおいて、いかなる損害が生じても、弊社および著者をはじめとする本文書作成関連者は、一切の責任を負いません。

本文書に記載されている製品名などは、一般的にそれぞれの権利者の登録商標または商標です。

第 2 章

ESP32-C3M-TRY の機能概要

2.1　マイクロコントローラ

従来の ESP32 には、CPU として一般的にはあまり知られていない Xtensa の 32bit コアが使われていました。一方、ESP32-C3M-TRY に使用されている ESP32-C3-MINI-1 には、最近注目され次世代の主要な CPU になることが期待されている RISC-V が使用されています。したがって、ESP32-C3M-TRY を使用することで、ESP32 やその通信機能の利用法の習得や実験を行えるだけでなく、RISC-V の利用法の習得や実験を行うこともできます。

なお、アセンブラではなく、Arduino 言語 (C++) や MicroPython でプログラミングを行う場合には、CPU が Xtensa から RISC-V に変わっても従来の ESP32 と同様に利用することができます。

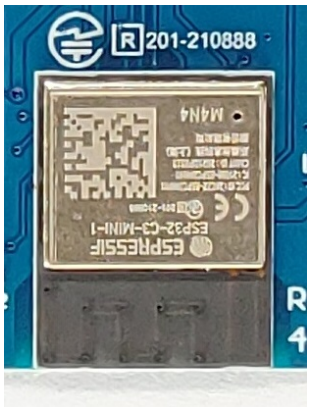


図 2.1　マイクロコントローラ ESP32-C3-MINI-1

ESP32-C3-MINI-1 は最高で 160MHz で稼働し、384KB ROM と 400KB RAM を内蔵しています。また、4MB の Flash を装備しています。

ESP32-C3-MINI-1 は従来の ESP32 同様、WiFi や BlueTooth などの無線機能を内蔵するだけでなく、新たな通信機能として、USB インターフェースを備えています。ESP32-C3M-TRY は、PC との USB 接続に、UART を USB 変換器で利用する従来方式ではなく、この新しく内蔵され

た USB インターフェースを利用する方式を採用しています。

2.2　表示装置

2.2.1　OLED ディスプレイ

ESP32-C3M-TRY の基板中央部には、図 2.2 に示すように OLED ディスプレイが装備されており、その表示能力を手軽に使用することができます。

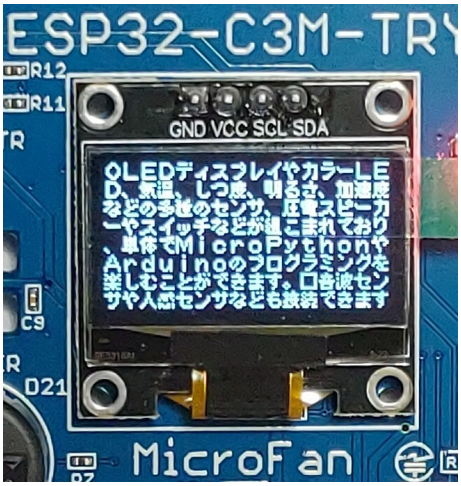


図 2.2　OLED ディスプレイ

OLED ディスプレイは 128x64 ピクセルのグラフィックディスプレイになっており、基本的にはピクセル単位の点灯、消灯機能しか持ちませんが、Arduino や MicroPython のライブラリを使用することにより、英数記号や日本語などの文字表示や、直線、四角、円など図形のグラフィックス表示を行うことができます。

ネット上などで公開されている ESP32-C3-MINI-1 のサンプルスケッチでは、IP アドレスや様々な情報を PC 上でシリアルモニタに表示する例が多いですが、実際の運用では ESP32-C3-MINI-1 を PC に接続して使用することは少ないため、運用時に必要な情報を確認することができないという問題があります。

ESP32-C3M-TRY では、開発ボード上に OLED ディスプレイが搭載されているため、PC と切り離して単独で運用している場合でも、ボードの稼働状態や様々な情報を OLED に表示し確認することができます。

2.2.2　カラー LED

ESP32-C3M-TRY の基板右上には、図 2.3 に示すように 3 個のカラー LED が装備されており、様々な色で発色させることができます。

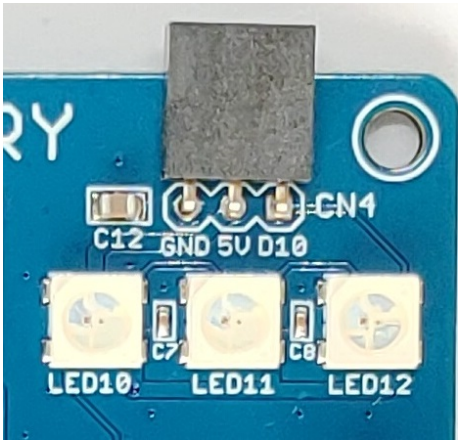


図 2.3　カラー LED

カラー LED の中には、LED の色や明るさを制御する IC が組み込まれており、一本の信号線で３個の LED の色や明るさの制御をすることができます。

カラー LED 上部の CN4 を使用すると、これらのカラー LED の延長として外部にもカラー LED ボードなどを接続して、カラー LED 表示機能を拡張することができます。

2.3　圧電スピーカー

ESP32-C3M-TRY には図 2.4 に示すように圧電スピーカーが装備されており、様々な音を出すことができます。

圧電スピーカーは、電圧をかけると変形する圧電素子を使用したスピーカーで、鳴らしたい音の周波数の交流信号を与えると、その周波数の音を出力します。

図 2.4　圧電スピーカー

2.4　センサー

2.4.1　温度、湿度センサー

ESP32-C3M-TRY の基板左上部には、温度と湿度を測れるセンサーとして、図 2.5 に示す AHT21 が装備されています。

AHT21 は I2C 信号で値を読み取ることができます。

図 2.5　温度、湿度センサー

2.4.2　明るさセンサー

ESP32-C3M-TRY の基板左上部には図 2.6 に示す明るさセンサーが装備されており、周囲の（正確には、センサーが向いた方向の）明るさを調べることができます。

図 2.6　カラー LED

明るさセンサーには、フォトトランジスタを使用しており、周囲の明るさの明暗が電圧の高低で出力されるように回路が構成されており、AD 変換で、その電圧を読み取って明るさを得ます。

2.4.3　加速度センサー

ESP32-C3M-TRY には図 2.7 に示す加速度センサー KXTJ3-1057 が装備されており、基板にかかる加速度を調べることができます。加速度センサーは基板の中央部、OLED ディスプレイの下に装備されているため、通常は見ることができません。

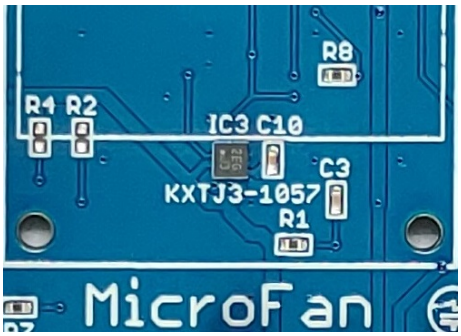


図 2.7　加速度センサー

加速度センサーは、3 軸の加速度（あるいは重力加速度）を計測することができます。基板が静止している場合には重力加速度、基板に動きがある場合には加速度と重力加速度の合成値が計測されます。

加速度センサーの計測軸とその正方向を示します。

- X 軸
  基板の縦方向に並行な軸で、基板の上側から下側に向かう方向が正
- Y 軸
  基板の横方向に平行な軸で、基板の右側から左側に向かう方向が正
- Z 軸
  基板に垂直な軸で、基板の表側から裏側に向かう方向が正

加速度センサーは I2C 信号で値を読み取ることができます。

2.5　入力装置

ESP32-C3M-TRY には、基本的な入力装置として、図 2.8 に示すタクトスイッチが 3 個が装備されています。

タクトスイッチの出力はデジタル入力で読み取ることができます。タクトスイッチが押された状態は 0、押されてない状態は 1 となる負論理構成となっています。

図 2.8　タクトスイッチ

このほかにも、形状と用途の異なるタクトスイッチが 2 個装備されており、それらは ESP32-C3M-TRY のリセットと、BootLoader の制御に使用されます。

2.6　外部接続装置

ESP32-C3M-TRY にはオプションとして、様々な入出力装置を接続して使用することができます。

2.6.1　超音波距離センサー

ESP32-C3M-TRY には基板左上の CN5 に、図 2.9 に示す超音波距離センサー HCSR-04 を接続することができ、計測対象物と基板との距離を測ることができます。

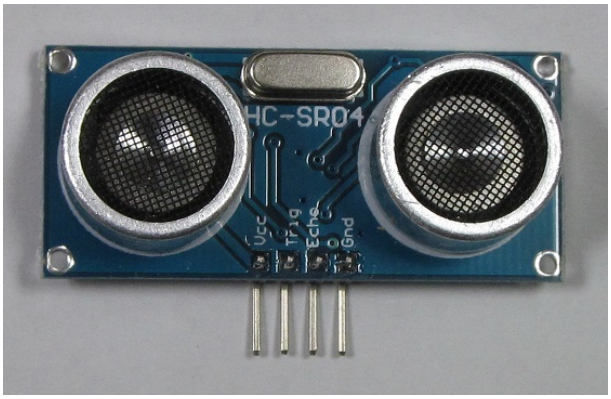


図 2.9　超音波距離センサー HC-SR04

超音波距離センサー HCSR-04 の CN5 への取り付けは、図 2.10 に示すように基板の背面方向に向けて取り付けます。

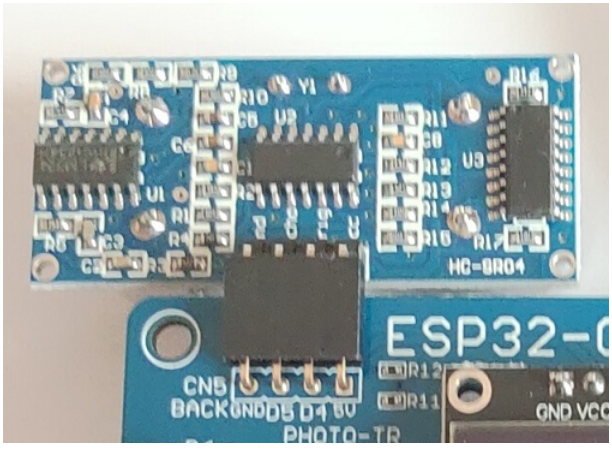


図 2.10　背面方向に向けた取り付け

2.6.2　人感センサー

ESP32-C3M-TRY には基板左下の CN2 に、図 2.11 に示す人感センサー:PIR（Passive Infrared Ray）を接続することができ、周囲に人間がいればそれを検知することができます。

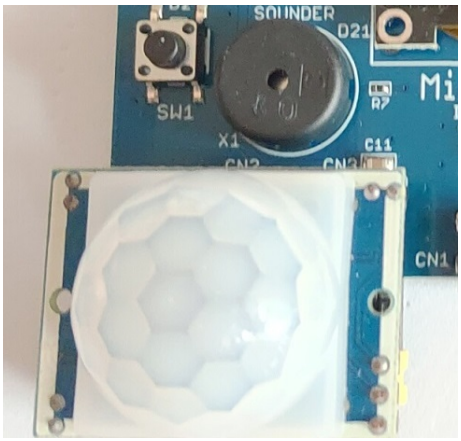


図 2.11　背面方向に向けた取り付け

人感センサーにはいくつかの方式がありますが、RIP は赤外線を利用して人間を感知するセンサーで、人間が発する赤外線が感知します。また、単純に赤外線の有無で人の有無を感知すると、人間と同様な温度の物があると人間と誤認するため、赤外線を放射している物に動きがある場合に人として検知するようになっています。このため、そばに人がいても、動きがなければ人間はいないと判断するという問題もあります。

2.6.3　RC サーボ

ESP32-C3M-TRY には基板左下の CN3 に、RC サーボを接続することができ、その回転量を制御することができます。

2.6.4　カラー LED

ESP32-C3M-TRY には基板右上に装備しているカラー LED に加え、CN4 に外部のカラー LED を接続しすることができ、基板上の LED と連携・独立して使用することができます。

2.7　USB インターフェース

ESP32-C3M-TRY は ESP32-C3/S3 で新たに内蔵された USB/JTAG 機能を使用した USB インターフェースを装備しています。

USB インターフェースは以下のような目的で使用されます。

- ESP32-C3M-TRY への電力供給。
  USB からは 5V の電力が供給され、ESP32-C3M-TRY では、基板上の電圧レギュレータで 3.3V に変換され使用されます。
- ESP32-C3M-TRY へのスケッチやファームウェアの書き込み。
  Arduino のスケッチの書き込み時や、MicroPython のファームウェアの書き込みに使用します。
- ESP32-C3M-TRY と PC 間のシリアル通信。
  ESP32-C3M-TRY で実行する Arduino のスケッチの実行時入出力や、MicroPython のスクリプトの書き込みやスクリプトの実行時の入出力で使用されます。

2.7.1　USB インターフェースの使用上の注意点

スケッチやファームウェアの書き込みで USB/JTAG ポートを使用する場合には、ESP32-C3-MINI-1 の状態により書き込みがうまく行えない場合があります。この様な場合には RST と LOAD スイッチを一緒に押し、まず RST スイッチを離し次に LOAD スイッチを離す事により、ESP32-C3-MINI-1 を BOOT ローダーモードに移行させることにより、ファームウェア等の書き込みが行えるようになります。

なお、RST スイッチを押すと ESP32-C3-MINI-1 の初期化により、一旦 USB 接続が失われます。したがって、RST スイッチを押すたびに IDE では再度 USB ポートの設定を行う必要があります。

上記の手順で ESP32-C3-MINI-1 を BOOT ローダーモードにしてファームウェア等を書き込んだあとは、書き込み終了後に RST スイッチを押して ESP32-C3-MINI-1 を通常の状態に戻してやる必要があります。

2.8　電源回路

ESP32-C3M-TRY には USB から必要な電力を取得する電源回路が組み込まれており、USB で PC に接続して開発を行う場合には、外部に別途電源を用意する必要がありません。

なお、ESP32-C3M-TRY にプログラムを書き込み PC と独立して動作させる場合には、USB コネクタから 5V の電力の供給をしてください。

2.8.1　電圧レギュレータ

ESP32-C3-MINI-1 は WiFi 機能を稼働させる際に、突入電流として多くの電流を消費することが知られています。このため、電源が貧弱だと、ESP32-C3-MINI-1 の動作が不安定になることがあります。

ESP32-C3-MINI-1 は無線機能の利用時に 300mA 程度の電流を消費します。さらに、瞬間的ではありますが、突入電流として 1A 以上を消費することもあるようです。ESP32-C3M-TRY で

利用している電圧レギュレータ BL8071 は、少なくとも 1.5A 以上の電流を供給[*1]できますので ESP32-C3-MINI-1 を余裕をもって稼働させることができます。

また、BL8071 の入力電圧から出力電圧のドロップダウンは 300mV 程度で、USB から電力を取得する場合、ショットキーダイオードの順方向電圧降下と合わせると電圧低下は 0.8V 程度となります。ESP32-C3-MINI-1 が瞬間的に大きな電流を必要としている際に、USB からの供給電圧が定格の 5V をある程度下回っても、安定した電源電圧 3.3V を維持することができます。

2.8.2　外部への電力供給と注意点

CN2-CN5 に接続する外部装置への電力の供給は、それぞれのコネクタを通して ESP32-C3M-TRY から行われます。

これらのコネクタの基本的な用途は想定されており、その用途での利用に関して注意すべき点を以下に示します。なお、これらのコネクタを基本的な用途外に転用することも可能ですが、その際には、信号線の利用法や電力供給量などが適切に行われるように、十分に配慮をお願いします。PC に接続された USB 端子からは、500mA あるいは 900mA 程度の電流しか取り出すことができないので、ESP32-C3M-TRY の使用分も含めて、使用する電力の使用量が過大にならないようにご注意ください。

CN2 に接続される人感センサーと、CN5 に接続される超音波距離センサーは、電力消費量が小さいので、その使用時の電力使用量に注意する必要はありません。

一方、CN3 に接続する RC サーボと CN5 に接続するカラー LED に関しては、その電力使用量に注意する必要があります。

RC サーボもカラー LED も、USB コネクタから供給される 5V の電源が直接接続されており、消費されます。

[*1] ただし USB2.0 からの供給電流は最大で 500mA、USB3.0 からは 900mA です。また、放熱の制限で継続して 1.5A の電流を使用することはできません。

第 3 章

MicroPython プログラム環境の整備

Python は深層学習やデータ科学の分野で成功をおさめ、その使いやすさや機能の高さから、急速に幅広い分野で使用されるようになってきました。

Python そのものは、メモリ容量など多くのリソースを使用するため、そのままリソースに制限の大きい MCU に組み込むことは困難でした。そこで、MCU で Python を使用できるように、2013 年頃に Python のサブセットとして MicroPython が開発されました。当初は ARM のみへの対応でしたが、現在は対応する MCU が広がっており、その中に ESP32 が含まれています。

3.1 MicroPython の WEB ページ

MicroPython のサイトを以下に示します。
`https://micropython.org/`

- `https://micropython-docs-ja.readthedocs.io/ja/latest/esp32/tutorial/intro.html`
  ESP32 への MicroPython のインストール法が紹介されています。
  ESP32-C3M-TRY は ESP32-C3-MINI-1 を使用しているので、使用するファームウエアなど異なる点がありますが、インストール手順の大きな流れを確認できます。
- `https://micropython-docs-ja.readthedocs.io/ja/latest/esp32/quickref.html`
  ESP32 用のクイックリファレンスがあり、ESP32 で MicroPython を使用する際に大変役立ちます。ただし、現在の記述は ESP32-C3-MINI-1 ではなく、基本の ESP32 を対象にした記述になっているので、ピン番号など読み替えが必要な部分があるのでご注意ください。
- `https://micropython-docs-ja.readthedocs.io/ja/latest/library/index.html`
  MicroPython に固有なライブラリに関しては、このページにまとめられています。
- `https://micropython.org/download/`
  ESP32 をはじめとする様々なマイクロコントローラに移植された MicroPython のファームウェアを選択することができます。

3.2　Thonny: IDE のインストール

ESP32-C3M-TRY に MicroPython のファームウェアを書き込み PC から操作する場合には、PC に TeraTerm などのシリアル通信用のアプリケーションをインストールして使用することができます。

しかしながらこの方法では、MicroPython の利用には不便な点が多いので、MicroPython を手軽に利用するための様々な支援機能が組み込まれた IDE を利用しましょう。MicroPython の IDE にはいくつかの候補がありますが、ここでは Raspberry Pi のソフトウェアパッケージに Python 用 IDE としてあらかじめ組み込まれている Thonny を紹介します。

3.2.1　Thonny の概要

Thonny の情報サイトを以下に示します。

- `https://thonny.org/`

Thonny には様々な機能がありますが、 例えば以下のような機能が、ESP32-C3M-TRY 上の MicroPython を快適に使用するために大変役立ちます。

- Python の文法を理解した組込みエディタが使える。
- MicroPython が ESP32-C3-MINI-1 上に作成しているファイルシステムの操作を IDE のエディタなどと連携して行える。これにより、Python プログラムの ESP32-C3-MINI-1 への書き込み・保存や、ライブラリ・モジュール等のデバイスへの登録が簡単に行える。
- PC 上で稼働する Python がインストールされるため、ESP32-C3-MINI-1 の MicroPython だけでなく、PC の標準的な Python を使用できる。また、この Python を利用して esptool などを使用できる。

3.2.2　Thonny のインストールパッケージのダウンロードとインストール

Thonny の TOP ページを開くと図 3.1 のような内容が表示されます。

- `https://thonny.org/`

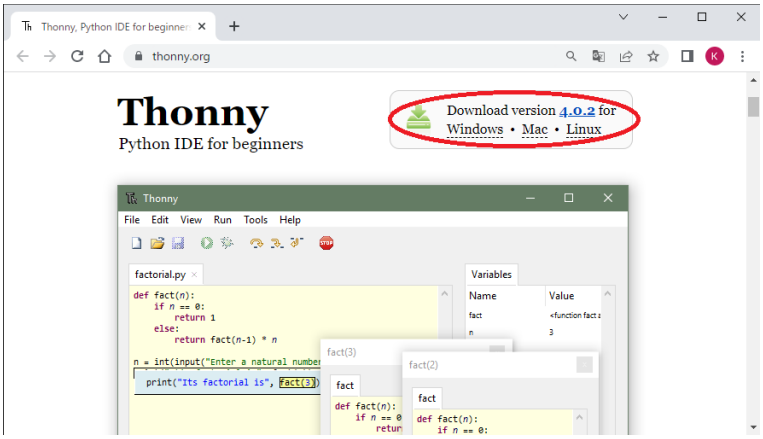


図 3.1　Thonny の TOP ページ

このページの右上の赤丸部分の Windows の部分をクリックすると、図 3.2 の様なダウンロード用のパネルがポップアップします。

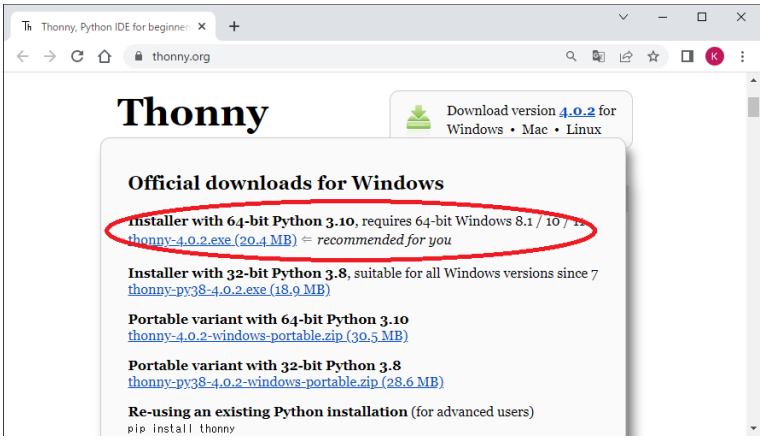


図 3.2　Thonny のダウンロードリンク

この中にはいくつかのインストールパッケージが表示されますが、特段の意向がなければ、一番上の 64bit 版の.exe 形式のインストールパッケージをダウンロードしてください。このパッケージには IDE だけでなく PC 用の Python も含まれており、PC 用の Python もこの IDE で使用できるようになっています。

ダウンロードしたインストールパッケージを起動して Thonny をインストールしてください。インストール途中で、デスクトップに Thonny のアイコンを設定するか聞いてくるので、それをチェックしてインストールすると Thonny の起動を簡単に行えるようになります。

3.2.3　Thonny の起動

Thonny のインストールができたら、デスクトップ上のアイコンなどをクリックして起動することができます。Thonny を最初に起動すると、IDE のメニュー等の言語設定を選択できるので、日本語を選択するとよいでしょう。

Thonny の起動画面を図 3.3 に示します。

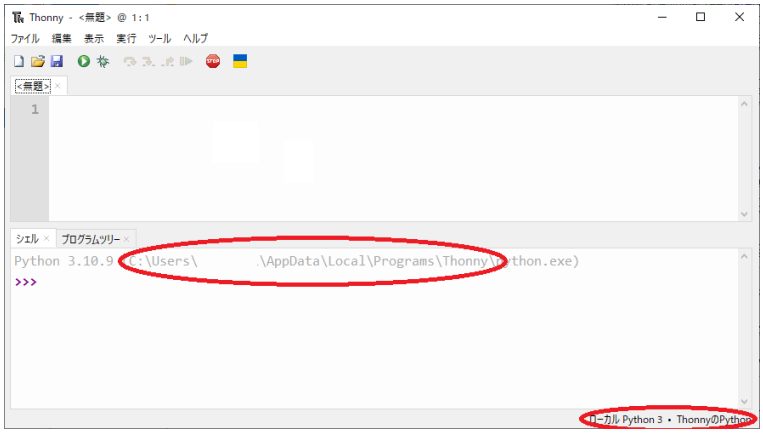


図 3.3　Thonny の起動画面

Thonny を起動すると、最初は Thonny と共にインストールされた PC 用の Python を実行対象として起動されます。初期設定では、Thonny の画面は上側のペインがプログラムの編集画面、下側のペインが Python の対話的実行画面になっています。

この状態で、下部のペインのプロンプト [>>>]　に Python のプログラムを入力すれば、そのプログラムは PC 上の Python で実行されます。

ちなみに、図 3.3 の中ほどの赤丸で括った対話実行用のペインの最初の部分には、Thonny と PC 用の Python がインストールされたパスが表示されます。（パスの一部に利用者のアカウント名（ログイン名）が表示されているので、図からは削除しています）また、PC 用の Python が実行対象となっていることは、右下部の赤丸で括った部分に表示されています。

Thonny と共にインストールされた PC 用の Python をコマンドプロンプト (CMD.exe) で実行したい場合には、Thonny の [ツール] ⇒ [システムシェルを開く...] メニューを選択してください。コマンドプロンプト（CMD.EXE）が起動されます。

図 3.4 に示すように、コマンドプロンプトで python と入力してエンターキーを押すと、Thonny と共にインストールされた python をコマンドプロンプトで起動することができます。

```
> python
Python 3.10.9 (tags/v3.10.9:1dd9be6, Dec  6 2022, 20:01:21) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello Python")
Hello Python
>>> quit()
>
```

図 3.4　Thonny と共にインストールされた Python の実行

Thonny からコマンドプロンプトを起動すると、そのコマンドプロンプトでは、Thonny と共にインストールされた Python のパスが環境変数 PATH に設定されているので、コマンドプロンプトで cd コマンドなどを使用して任意のパスに移動して Python を起動して利用できます。環境変数 PATH の内容は、[set PATH] コマンドの実行で表示することができます。

3.3　MicroPython ファームウエアの書き込み

ESP32-C3M-TRY に MicroPython のファームウェアを書き込みには、書き込み用のツール esptool を準備する必要があります。

esptool を準備する方法は以下に示す 3 種類の方法があります。

- ArduinoIDE の ESP32 開発環境に組み込まれている esptool を使用する。
  Arduino IDE と ESP32 に開発環境をすでにインストールしている場合に、直ちに使うことができるのでお勧めです。
- Thonny と共にインストールされた Python を使用して esptool を利用できるようにする。
  Thonny と共にインストールされた Python は単独でインストールされた Python と少し設定が異なるため、それを使用した esptool の導入は、ネット上で紹介されている操作手順とは少し手順が異なるので注意が必要です。しかしながら、Thonny の Python とは別の Python を 2 重インストールせずに準備できるので、PC 上で Python を使う予定がないのであれば価値のある方法です。
- Python を新たにインストールして esptool を利用できるようにする。
  Thonny の Python と 2 重に Python をインストールすることになり少々気が引けますが、ネット上で紹介されている操作手順で準備できるので、分かりやすい方法です。

上記の 3 種類の方法から、利用者の環境や目的に合った方法を選択して esptool の準備を行ってください。以下に 3 種類の方法を個別に説明しますが、その前に、ESP32-C3M-TRY に書き込む MicroPython のファームウェアのダウンロード法を示します。

3.3.1　MicroPython のファームウェアのダウンロード

ESP32-C3M-TRY に書き込む MicroPython のファームウェアは、以下のページから取得します。

ESP32-C3-MINI-1 用の MicroPython のファームウェアは、PC との通信に使用する USB インターフェースによって別々のファームウェアが提供されています。

- `https://micropython.org/download/esp32c3/`
  PC との通信に UART-USB を使用するファームウェア
- `https://micropython.org/download/esp32c3-usb/`
  PC との通信に USB/JTAG を使用するファームウェア

ESP32-C3M-TRY は、後者の USB/JTAG を使用したファームウェアを使用します。

USB/JTAG を使用する MicroPython のファームウェアのダウンロードページを図 3.5 に示します。ページの中ほどには、esptool を使用した標準的なファームウェアの書き込み方法が例示されています。ページの下部にファームウェアの色々な形式でのダウンロードリンクが示されています。ダウンロードするファイルは、拡張子が `.bin` 形式です。

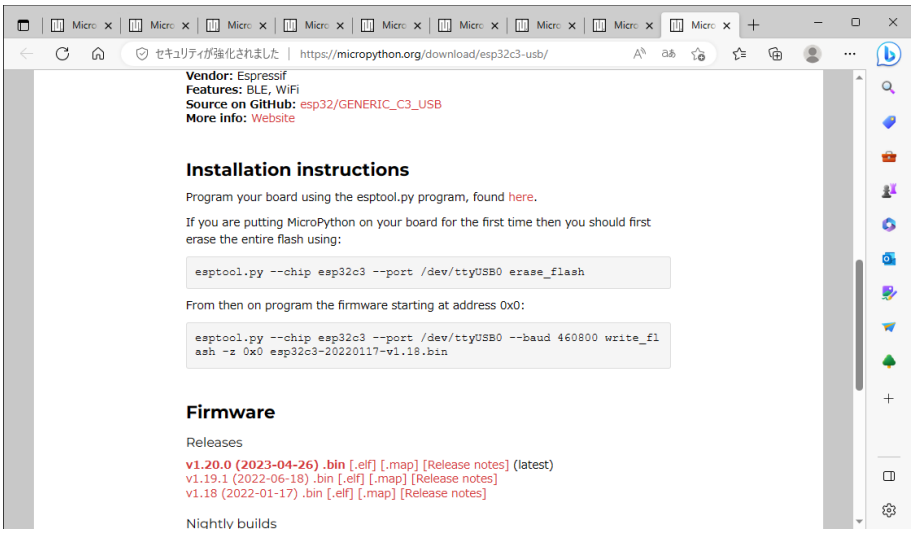


図 3.5　MicroPython のファームウェアのダウンロードページ

この文書の作成時点で取得できた最新のファームウェアのファイル名は以下の通りです。

- `esp32c3-usb-20230426-v1.20.0.bin`

次節以降のファームウェアの書き込み操作を行う前に、上記のファームウェアのダウンロードを行っておいてください。

3.3.2　Arduino の IDE のツールを利用した書き込み

Arduino IDE に組み込んだ ESP32 用の開発環境に esptool が含まれています。

Arduino でスケッチを作成しコンパイルすると、プログラムの開発ボードへのアップロード時に、IDE のメッセージ領域に沢山表示されるメッセージの最初の方に、esptool.py という文字列を確認することができます。このように、Arduino のスケッチを開発ボードにアップロードする処理に esptool が使われており、すでに利用できる状態にありますので、この esptool を利用する方法を紹介します。

esptool の格納場所の確認

著者の現在の開発環境では、以下のようなパスに esptool.exe という実行ファイルが格納されています。なお、パスの中の username は利用者のアカウント名です。

- `C:\Users\username\AppData\Local\Arduino15\packages\esp32\tools\esptool_py\4.2.1`

ファイルエクスプローラーでフォルダを順次見回って探すのが面倒であれば、ファイルエクスプローラーで Arduino をインストールしたドライブ（一般的には C）を esptool を検索すると、少し時間がかかるかもしれませんが esptool.exe を見つけることができます。

esptool の動作確認

esptool.exe を見つけたら、カレントディレクトリを esptool.exe が格納されているパスに移動しましょう。まず、コマンドプロンプト (CMD.EXE) を開きます。次に コマンドプロンプトに cd とスペースを入力した後に、ファイルエクスプローラー上で見つけた esptools.exe をコマンドプロンプトにドラッグアンドドロップします。コマンドプロンプトにはコマンドのパスを示す文字列が入力され、図 3.6 のように表示されます。なお、パス内の username は利用者のアカウント名です。

```
> cd C:\Users\username\AppData\Local\Arduino15\packages\esp32\tools\esptool_py\4.2.1\esptool.exe
```

図 3.6　exptool.exe のコマンドパス

次に、バックスペースキーで esptool.exe を削除すると表示は図 3.7 のようになります。

```
> cd C:\Users\username\AppData\Local\Arduino15\packages\esp32\tools\esptool_py\4.2.1\
```

図 3.7　exptool.exe が保存されているパス

この状態でエンターキーを押すと、カレントディレクトリを esptool.exe の格納場所に移動でき

ます。これ以降は、単純に esptool と入力すると、esptool を実行できるようになります。

ESP32-C3M-TRY の USB 接続

ここで、ESP32-C3M-TRY を USB で PC に接続してください。5.2 節を参照して ESP32-C3M-TRY を BOOT ローダーモードに設定してください。

もし、ESP32-C3M-TRY 以外の開発ボードを接続している様であれば、それらをすべて USB から外してください。

この状態で図 3.8 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-C3M-TRY の各種の情報を USB 経由で取得して画面に表示します。

```
> esptool flash_id

esptool.py v4.2.1
Found 1 serial ports
Serial port COM17
Connecting...
Detecting chip type... ESP32-C3
Chip is ESP32-C3 (revision 3)
Features: Wi-Fi
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

図 3.8　開発ボードの情報取得

esptool の標準的な操作法の説明では、開発ボードのチップの種類やシリアルポートを指定する様に書かれていますが、esptool は自動的に識別してくれるので、図 3.8 の例のように面倒であればそれらの指定を行う必要はありません。

esptool を使用したファームウェアの書き込み

まず、図 3.9 に示すように、erase_flash コマンドで ESP32-C3M-TRY の ESP32-C3-MINI-1 モジュールのフラッシュメモリの内容を消去します。この処理は少し時間がかかります。

```
> esptool erase_flash
```

図 3.9　フラッシュメモリの消去

次に、図 3.10 に示すように write_flash コマンドを使用して MicroPython のファームウェアを書き込みます。図の [PATH] の部分には、ファームウェアをダウンロードした場所のパスを記入してください。（例えば C:\HOME\ など。）

```
> esptool write_flash -z 0 [PATH]esp32c3-usb-20230426-v1.20.0.bin
```

図 3.10　MicroPython のファームウェアの書き込み

この書き込み処理には、１ - ２分程度かかります。PC のプロンプトが返ってきたらファームウェアの書き込み成功です。ファームウェアの書き込み完了後に RST スイッチを押して ESP32-C3M-TRY を再起動してください。

3.3.3　Thonny の Python を利用した書き込み

MicroPython のファームウェアの ESP32-C3M-TRY への書き込み操作は、Thonny と共にインストールされた Python をコマンドプロンプトで起動して行います。

コマンドプロンプトの起動

Thonny を起動して、まず、図 3.11 の赤丸で示すように PC 用の Python を実行対象として選択してください。

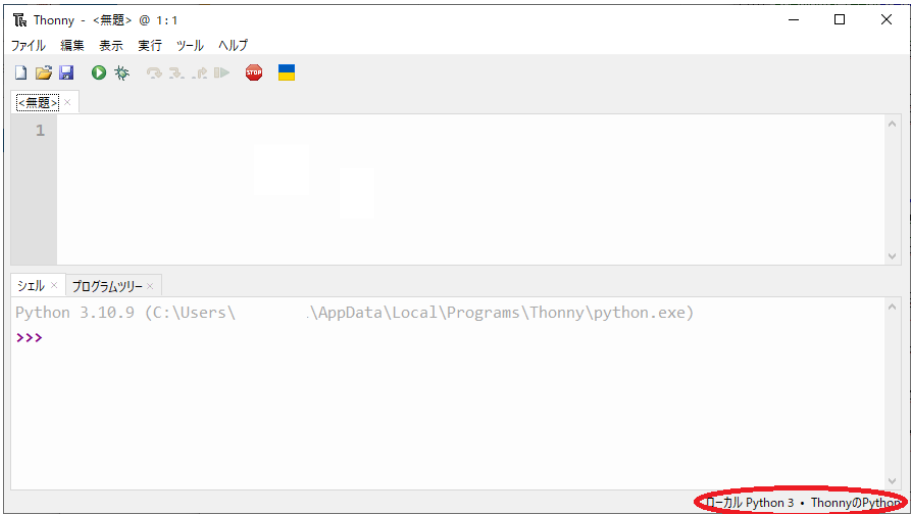


図 3.11　Thonny の起動

次に、Thonny の [ツール] ⇒ [システムシェルを開く...] メニューを選択してください。コマンドプロンプト（CMD.EXE）が起動されます。

esptool のインストール

コマンドプロンプトに図 3.12 に示すコマンドを入力してエンターキーを押すと、esptool をインストールできます。

```
> pip install esptool
```

図 3.12　esptool のインストール

esptool がインストールできたら、図 3.13 に示すコマンドを入力すると esptool の起動オプションなどが表示され、動作確認することができます。

```
> python -m esptool
```

図 3.13　esptool の動作確認

ESP32-C3M-TRY の USB 接続

ここで、ESP32-C3M-TRY を USB で PC に接続してください。5.2 節を参照して ESP32-C3M-TRY を BOOT ローダーモードに設定してください。

もし、ESP32-C3M-TRY 以外の開発ボードを接続している様であれば、それらをすべて USB から外してください。

この状態で図 3.14 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-C3M-TRY の各種の情報を USB 経由で取得して画面に表示します。

```
> python -m esptool flash_id

esptool.py v4.4
Found 1 serial ports
Serial port COM17
Connecting...
Detecting chip type... ESP32-C3
Chip is ESP32-C3 (revision v0.3)
Features: WiFi, BLE
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

図 3.14　開発ボードの情報取得

esptool の標準的な操作法の説明では、開発ボードのチップの種類やシリアルポートを指定する様に書かれていますが、esptool は自動的に識別してくれるので、図の例のように面倒であればそれらの指定を行う必要はありません。

esptool を使用したファームウェアの書き込み

まず、図 3.15 に示すように erase_id コマンドで ESP32-C3M-TRY の ESP32-C3-MINI-1 モジュールのフラッシュメモリの内容を消去します。この処理は少し時間がかかります。

```
> python -m esptool erase_flash
```

図 3.15　フラッシュメモリの消去

次に、図 3.16 に示すように write_flash コマンドを使用して MicroPython のファームウェアを書き込みます。図の [PATH] の部分には、ファームウェアが格納されている場所のパスを記入してください。（例えば C:\HOME\ など。）

```
> python -m esptool write_flash -z 0 [PATH]esp32c3-usb-20230426-v1.20.0.bin
```

図 3.16　MicroPython のファームウェアの書き込み

この書き込み処理には、１ - ２分程度かかります。PC のプロンプトが帰ってきたらファー

ムウェアの書き込み成功です。ファームウェアの書き込み完了後に RST スイッチを押して ESP32-C3M-TRY を再起動してください。

3.3.4　PATH 設定された Python を利用した書き込み

Python のインストール

以下のサイトからインストールパッケージをダウンロードし、インストール時に PATH も設定されている Python が PC にインストールされているのであればそれを使用します。

- `https://www.python.org/`

PC に Python がインストールされていない場合には、上記のサイトから Python のインストールパッケージをダウンロードしてインストールしてください。

Python をインストールする際には、インストーラの下部に表示されている [Add Pyhon 3.XX to PATH] をチェックして、Python をコマンドプロンプトで利用しやすいようにしておいてください。

esptool のインストール

Python がインストールできたら、「Windows システムツール] からコマンドプロンプト (CMD.EXE) を開きます。図 3.17 に示すコマンドを入力してエンターキーを押すと、esptool をインストールできます。

```
> pip install esptool
```

図 3.17　esptool のインストール

esptool のインストール後図 3.18 に示すコマンドを入力してエンターキーを押します。

```
> esptool.py
```

図 3.18　esptool の動作確認

esptools.py のバージョンや使用法が表示されればインストールは成功です。

ESP32-C3M-TRY の USB 接続

ここで、ESP32-C3M-TRY を USB で PC に接続してください。5.2 節を参照して ESP32-C3M-TRY を BOOT ローダーモードに設定してください。

もし、ESP32-C3M-TRY 以外の開発ボードを接続している様であれば、それらをすべて USB から外してください。

この状態で図 3.19 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-C3M-TRY の各種の情報を USB 経由で取得して画面に表示します。

```
> esptool.py flash_id

esptool.py v4.5.1
Found 1 serial ports
Serial port COM17
Connecting...
Detecting chip type... ESP32-C3
Chip is ESP32-C3 (revision v0.3)
Features: WiFi, BLE
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

図 3.19　ボードとの接続確認

esptool の標準的な操作法の説明では、開発ボードのチップの種類やシリアルポートを指定する様に書かれていますが、esptool は自動的に識別してくれるので、図の例のように面倒であればそれらの指定を行う必要はありません。

esptool を使用したファームウェアの書き込み

まず、図 3.20 に示すように erase_id コマンドで ESP32-C3M-TRY の ESP32-C3-MINI-1 モジュールのフラッシュメモリの内容を消去します。この処理は少し時間がかかります。

```
> esptool.py erase_flash
```

図 3.20　フラッシュメモリの消去

次に、図 3.21 に示すように write_flash コマンドを使用して MicroPython のファームウェアを書き込みます。図の [PATH] の部分には、ファームウェアが格納されている場所のパスを記入してください。（例えば `C:\HOME\` など。）

```
> esptool.py write_flash -z 0 [PATH]esp32c3-usb-20230426-v1.20.0.bin
```

図 3.21　MicroPython のファームウェアの書き込み

この書き込み処理には、１ - ２分程度かかります。PC のプロンプトが帰ってきたらファームウェアの書き込み成功です。ファームウェアの書き込み完了後に RST スイッチを押して ESP32-C3M-TRY を再起動してください。

3.4　Thonny を利用した MicroPython でのプログラミング

3.4.1　Thonny の起動

Thonny のインストール時に作成されたデスクトップの Thonny のアイコンをダブルクリックするなどして Thonny を起動してください。

図 3.22 のようなウインドウが開きます。

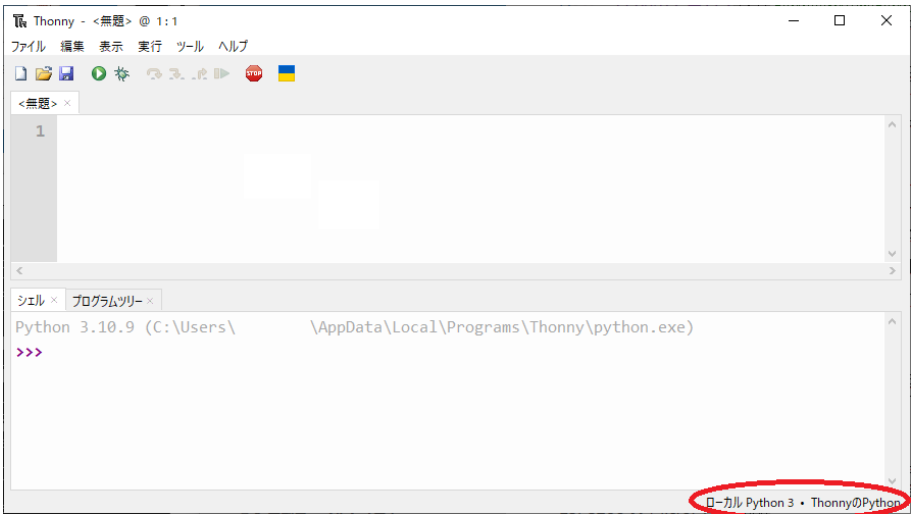


図 3.22　Thonny の起動画面

ウインドウの上のペインがプログラムの編集用で、下のペインが Python の実行用のペインです。この例では、下のペインで PC にインストールされた Python が起動されていますが、Thonny の前回の終了時の設定状況により、PC の Python が起動されるか、ESP32-C3M-TRY の Python が起動されるかが変わります。

どちらの Python が選択されているかは、ウィンドウ右下の赤丸で囲った部分に表示されます。

また、ESP32-C3M-TRY の MicroPython が起動される設定となっていた場合に、ESP32-C3M-TRY が USB で PC に接続されていない場合には、MicroPython の起動ができず図 3.23 の様な表示となります。

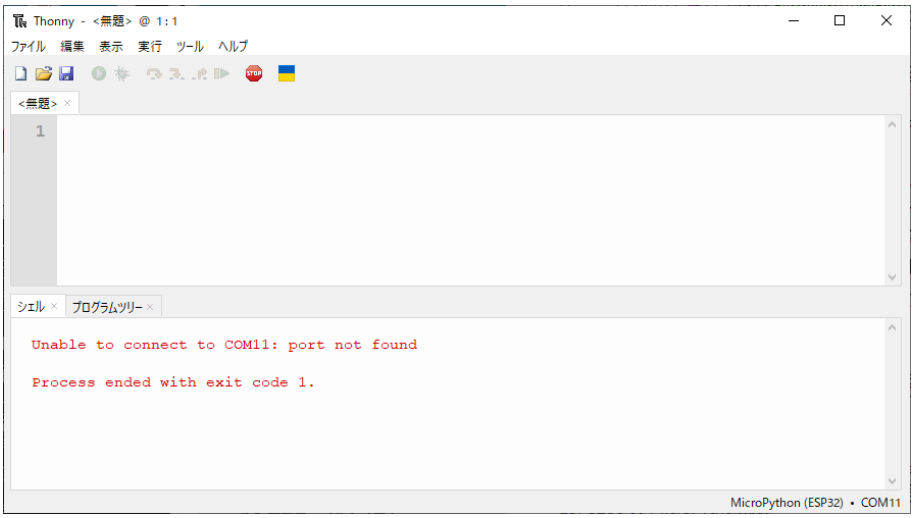


図 3.23　MicroPython 起動の失敗

3.4.2　MicroPython の起動

Thonny の下部のペインで MicroPython が起動されていない場合には、MicroPython の起動を行います。

ESP32-C3M-TRY が PC に接続されていない場合には、まず USB で PC に接続してください。

その後、Thonny のウィンドウの右下の [ローカル Python3・Thonny の Python] あるいは [MicroPython(XXX)・COMX] と書かれている部分をクリックしてください。すると、使用する Python を選択できるメニュー項目が表示されるので、その中で [MicroPython(ESP32)・COMX] という項目を選択してください。

図 3.24　Thonny で MicroPython の起動

この操作により、図 3.24 に示すように Thonny の下部のペインで MicorPython が起動され使用できるようになります。Thonny の下部のペインに、MicroPython のバージョンや、実行されているモジュールが ESP32C3 であることが起動メッセージとして表示されているのを確認できます。

3.4.3　プログラムの対話的な実行

ESP32-C3M-TRY での MicroPython の対話的な実行は、Thonny の下部のペインにプログラムを入力しエンターキーを押すことで行えます。

まずは、起動メッセージに表示されている help() を入力してエンターキーを押してみてください。簡単な歓迎メッセージや、machine モジュールや network モジュールの使用例などが表示されます。

次に、簡単な演算例を入力してみましょう。

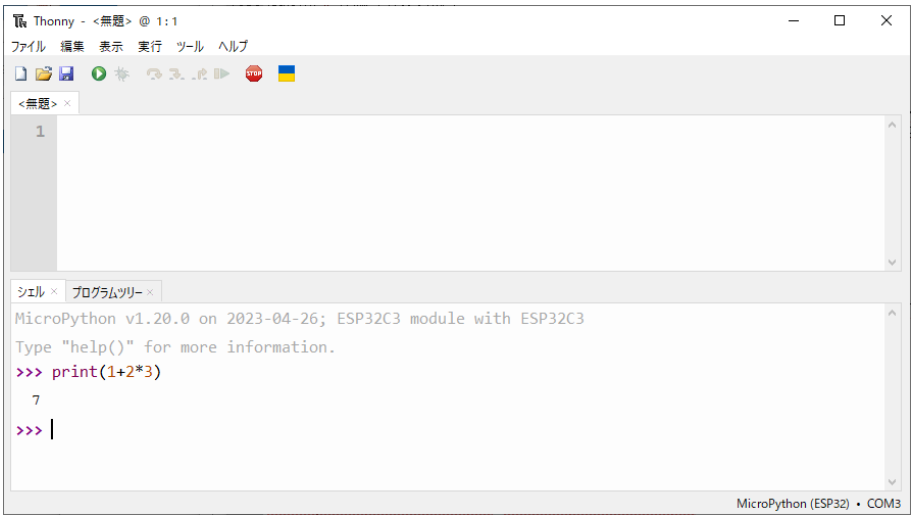


図 3.25　演算結果の出力

図 3.25 に示すように、入力した処理が ESP32-C3M-TRY 上で即座に実行され、演算結果が表示されるのを確認できます。

3.4.4　プログラムの編集と実行

対話的な実行環境で、少し行数のあるプログラムを入力して実行しようとすると、書き間違いや書き忘れがあって、試行錯誤でプログラムの再入力が面倒なことがことがよくあります。この様な場合には、まずエディタで正しいプログラムを作成・編集し、プログラムに間違いがないことを確認してそれを実行させるのが便利です。

この様にプログラムを編集、実行したい場合には、まずプログラムエディタを使用して、プログラムを入力・編集し、間違いのないプログラムを作成します。Thonny のプログラムエディタは、

図 3.26 に示すように、Thonny の上側のペインにあるので、そこにプログラムを入力します。

プログラム例は、よく L チカと呼ばれる、LED を点滅させる電子工作の入門プログラムです。

以降の Thonny のスクリーンショットは、都合により ESP32-C3M-SLIM 用に作成されたものをそのまま用いているため、LED が D7 に接続されていることが前提になっています。ESP32-C3M-TRY では、LED1 は D0 に接続されているので、LED 操作用のピン番号指定 7 は 0 に読み替えてくださるようお願いいたします。

図 3.26　プログラムの編集

プログラムの入力が完了したら、図 3.27 の上側の赤丸で囲まれた、緑色の丸の中に右向きの三角が表示されたアイコンをクリックします。

図 3.27　プログラムの実行

すると、ウィンドウ下部のシェルペインに赤丸で囲ったような表示が出力され、プログラムエディタに入力したプログラムが実行されます。この例では、プログラムの実行とともに ESP32-C3M-TRY の LED1 が点滅し始めたのを確認することができます。

このプログラム例は、print() などで計算結果を出力する部分がないのでシェルペインへの表示がありませんが、そのような処理が書かれているプログラムでは、計算結果などがシェルペインに表示されます。

なお、このプログラムは、無限ループになっており終了しませんが、シェルペインで CTRL-C を入力することで中断させることができます。プログラムを中断させた例を図 3.28 に示します。

図 3.28　プログラムの中断

3.4.5　作成したプログラムの保存

Thonny のエディタで作成したプログラムは、ファイルとして保存することができます。

MicroPython のファームウェアには、ESP32 のフラッシュメモリ上に小さなファイルシステムを作成する機能も含まれています。このため、ファイルは PC 上だけでなく、ESP32-C3M-TRY 上のファイルシステムに保存することができます。

ファイルを ESP32-C3M-TRY 上のファイルシステムに保存した場合には、その ESP32-C3M-TRY を他の PC で使用する場合でも使用することができるようになります。

まず、Thonny の [表示] ⇒ [ファイル] メニューを選択してください。図 3.29 のように、ウィンドウの左側にファイルペインが表示されます。

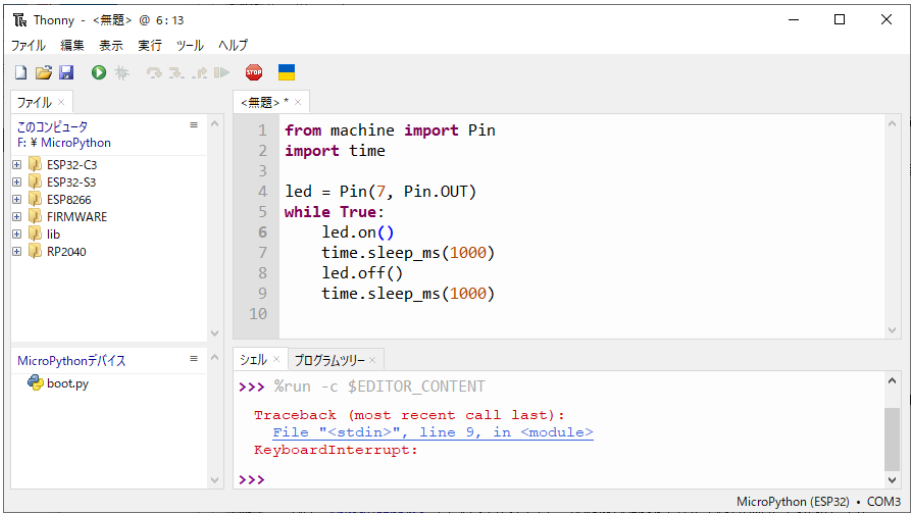


図 3.29　ファイルペインの表示

ファイルペインの上側は、PC 上のファイル、下側は ESP32-C3M-TRY 上のファイルを示しています。この例では、 ESP32-C3M-TRY にすでに boot.py が保存されていることが示されています。上下のファイルペインでは共に、エクスプローラーで行うように、フォルダの作成や削除、フォルダ間の移動などを行うことができます。

ここでは、現在編集しているプログラムを blink.py という名前で保存してみましょう。

まず、Thonny の [ファイル] ⇒ [名前を付けて保存...] メニューを選択してください。図 3.30 のようなダイアログが出てくるので、適切な保存先を選択してください。

ここでは、ESP32-C3M-TRY 上に保存する事とし、[MicroPython デバイス] をクリックして選択します。

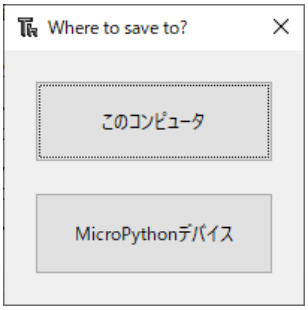


図 3.30　ファイルの保存先の選択

すると、図 3.31 に示すようなファイル名を入力するダイアログが表示されるので、保存するプログラムにファイル名を付けて入力してください。ここでは、blink.py と名付けて保存します。

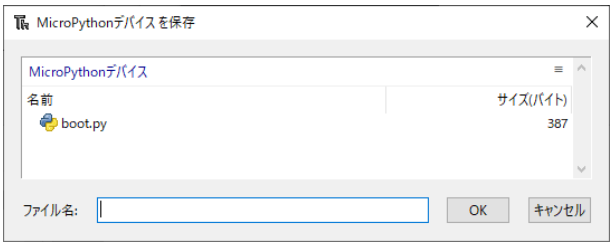


図 3.31　ファイルの名の入力

プログラムをファイルとして保存した後の Thonny の画面を図 3.32 に示します。

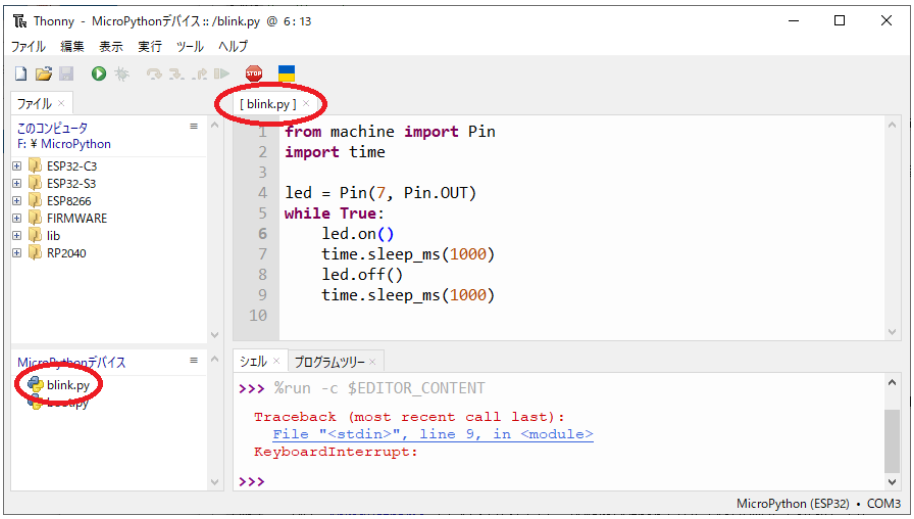


図 3.32　ファイルの保存後

図 3.32 では、エディタのペインの上部タブ部部分の [無題] となっていた部分が [blink.py] に変更されたことと、ファイルペインの下側に、[blink.py] が追加されたことを確認できます。

この後に、プログラムの内容を変更すると図 3.33 に示すように、変更されたファイルを再保存するためのアイコンが有効化されることが確認できます。このアイコンをクリックすると、変更内容がファイルに保存されます。

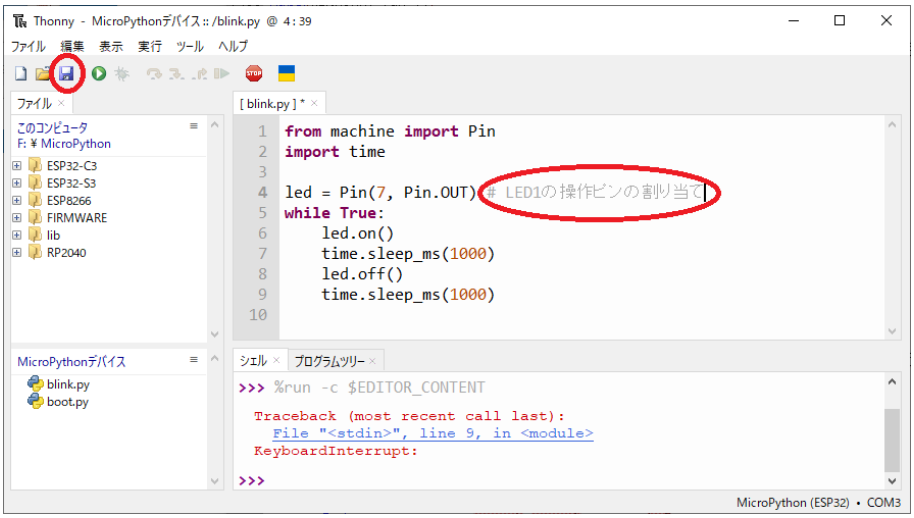


図 3.33　ファイルの変更

ファイルとして保存されたプログラムは、再度呼び出して編集、実行することができます。例えば、blink.py のファイルペインの上部タブの右側の×をクリックすると、編集処理を終了させることができます。編集結果が保存されていない場合には、ファイルとして保存するかどうかの確認が表示されるので、保存するか否かを指示してください。

ファイルの編集を修了すると編集ペインの内容が削除されて、図 3.34 のような画面になります。

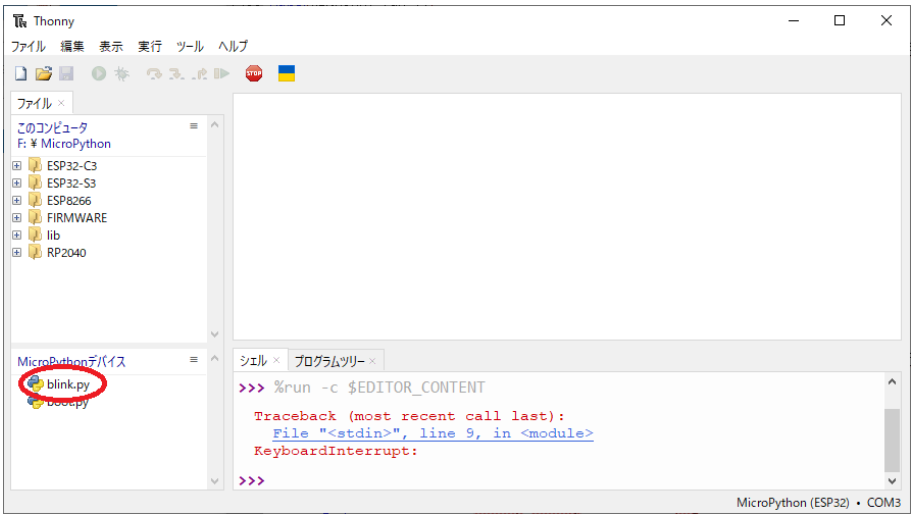


図 3.34　編集の終了

ファイルの編集を終了しても、ファイルペインのファイルをダブルクリックすることで、図 3.35 に示すように Thonny のエディタにプログラムを読み込み再度編集・実行することができます。

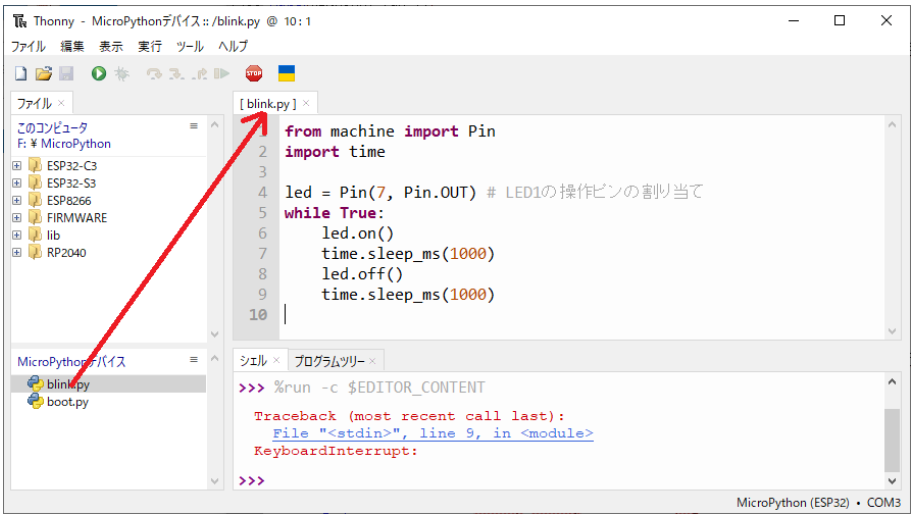


図 3.35　プログラムの再編集

3.4.6　MicroPython のライブラリの導入

python で使用するライブラリは、PC 上の Python では pip を利用してインストールするのが一般的ですが、MicroPython のライブラリは、PC 上の Python ではないので pip を使用してインストールすることはできません。MicroPython でのライブラリのインストールは、Thonny のパッケージ管理ツールを使用するか、blink.py の編集・保存と同様な方法でライブラリのプログラムを ESP32-C3-MINI-1 上のファイルシステムに保存することでインストールできます。

パッケージ管理ツールの利用

ここでは、MicroPython 用に作成されたライブラリのインストール法を紹介します。ESP32-C3M-TRY では開発ボード上に OLED ディスプレイを搭載できるようになっています。ESP8266 用の MicroPython には OLED ディスプレイ用のライブラリが ssd1306 という名前で標準で組み込まれており、単にインポートして使用することができます。しかしながら、ESP32 用の MicroPython には、OLED を操作するためのライブラリが標準で組み込まれていないので、それを見つけてインストールする必要があります。幸いにして、ESP8266 用の MicroPython に組み込まれている ssd1306 は Thonny のパッケージ管理ツールで ESP32-C3-MINI-1 にインストールして使用することができます。

まず、Thonny の [ツール] ⇒ [パッケージを管理...] メニューを選択してください。図 3.36 のようなダイアログが出てくるので、適切な保存先を選択してください。

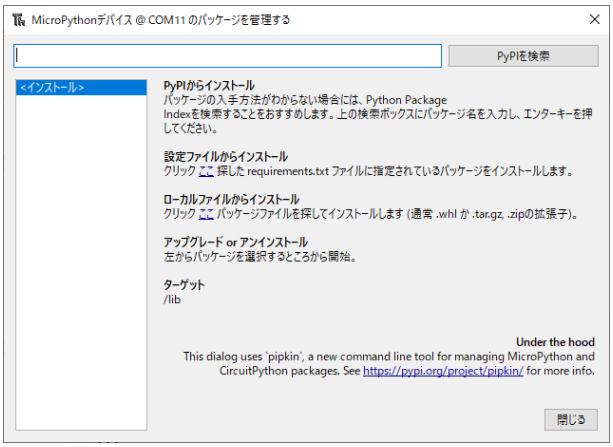


図 3.36　パッケージ管理ダイアログ

ssd1306 を入力して検索すると、候補のパッケージ（ライブラリ）が複数リストアップされます。ここでは、赤丸で囲んだ micropython-ssd1306 をクリックして選択してください。

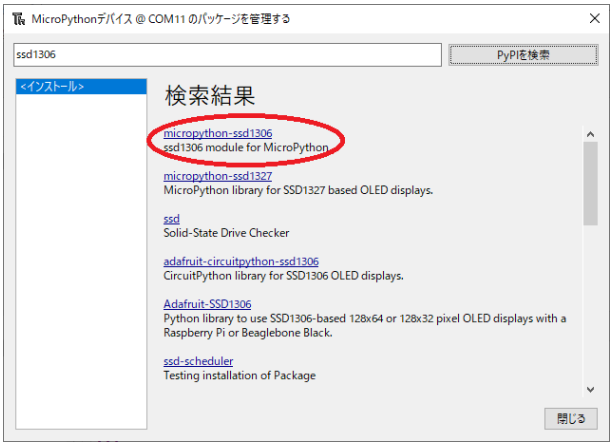


図 3.37　ssd1306 の検索

ダイアログの内容が図 3.38 の様に切り替わるので、下部のインストールボタンをクリックしてください。

図 3.38　ssd1306 パッケージ

インストールができたら、パッケージ管理ダイアログの右下の [閉じる] ボタンをクリックしてを閉じてください。

ライブラリ（パッケージ）インストール後の Thonny のファイルペインの MicroPython デバイスの部分を見ると、図 3.39 に示すように、[lib] フォルダが追加されていることが確認できます。また、[lib] フォルダの左側の [+] をクリックすると [lib] フォルダの中が表示され、ssd1306 ライブラリがインストールされていることを確認できます。

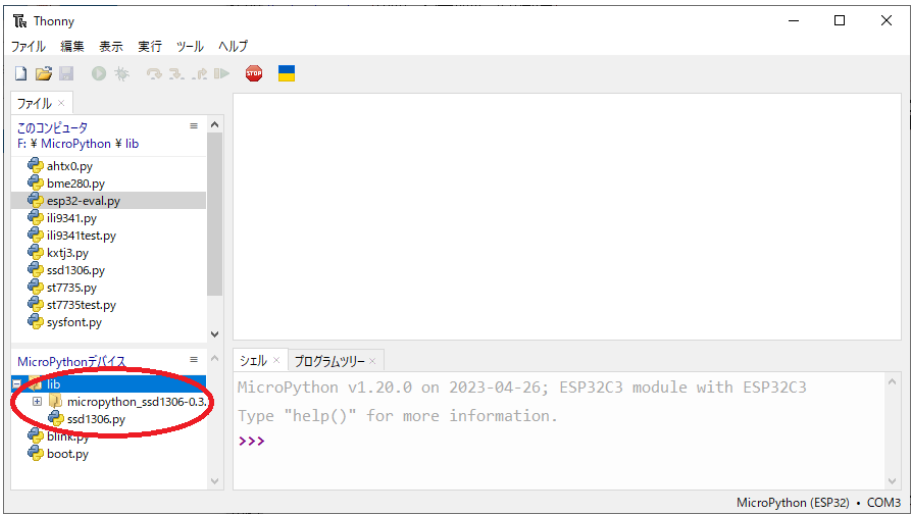


図 3.39　インストールされたライブラリ

MicroPython のファイルシステム上にライブラリがインストールされると、そのライブラリを import 文でプログラムに読み込んで使用することができます。

新しくインストールした ssd1306 ライブラリをインポートして使用するプログラム例を図 3.40 に示します。

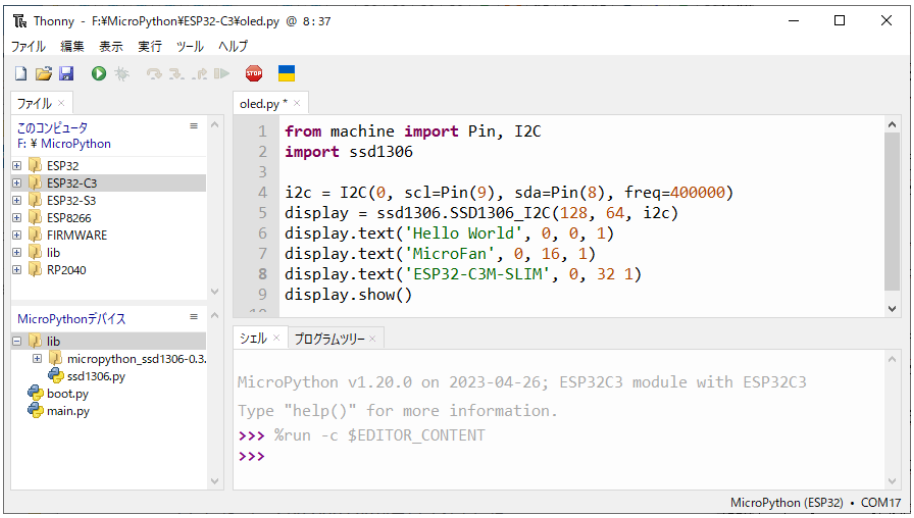


図 3.40　ssd1306 ライブラリの利用

このプログラムを実行すると、ESP32-C3M-TRY に接続している OLED ディスプレイに ‘Hello World’ から始まる 3 行のメッセージが表示されます。

3.5　MicroPython の起動時の特殊ファイル

MicroPython のファイルシステムには、様々な Python プログラムファイルやデータファイルを保存することができますが、boot.py と main.py という 2 つの特別な Python ファイルを保存することができます。

MicroPython では、デバイスがリセットされ再起動された際に、ファイルシステムに boot.py が保存されていると最初に boot.py の内容が自動的に実行され、ファイルシステムに main.py が保存されていると次に main.py の内容が自動実行されます。

boot.py と main.py は一般的に以下の様に使い分けます。

- boot.py には、その開発ボードを使用するときに汎用的に使える基本的な初期化処理や、便利に使える関数やクラスを記述する。
- main.py には、開発ボードの具体的な用途ごとに異なるアプリケーションプログラムを記述する。

なお、MicroPython のファームウェアを ESP32-C3-MINI-1 などに書き込んだ時点では、中身はコメントのみで実質的なコードは書かれていませんが、デフォルトで boot.py がファイルシステムに登録されています。

3.5.1　boot.py

ESP32-C3M-TRY の基本的な出力の設定等を boot.py に記述して、デバイスのファイルシステムに設定しておくと、その後のプログラミングで、具体的なピン番号など気にする必要がなくなる

ので便利です。

```
from machine import Pin, I2C

i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000) # i2c 回線の設定

led1 = Pin(0, Pin.OUT) # D0

sw1 = Pin(2, Pin.IN) # D2
sw2 = Pin(3, Pin.IN) # D3
sw3 = Pin(6, Pin.IN) # D6
```

図 3.41　boot.py の記述例

3.5.2　main.py

ESP32-C3M-TRY の起動時に実行すべき処理を main.py に記述して MicroPython のファイルシステムに設定しておくと、起動時に毎回行う処理などを自動化することができます。

図 3.42 の例では、先の boot.py の初期化処理と組み合わせて ESP32-C3M-TRY の起動時に必ず LED の点灯が行われるようになります。この例では、無限ループになっていないので、処理は１回のみ行われます。

```
led.on()
```

図 3.42　main.py の記述例

min.py に無限ループのプログラムが書き込まれていない場合には、ESP32-C3M-TRY が単独で使用されると、main.py の内容の実行が完了した後は、開発ボードは機能が停止したのと同じ状況になってしまいます。一方、ESP32-C3M-TRY が PC に接続され Thonny で MicroPython が利用できるようになっている場合には、この main.py のプログラムの実行後の処理を引き続き Thonny を使って入力・実行できます。

3.6　MicroPython のプログラム例

以下に、ESP32-C3M-TRY 上での MicroPython のプログラム例を示します。

ESP32-C3-MINI-1 の信号線を操作するために、PC 上で Python を使用している方にはなじみのないライブラリ・モジュールばかりを使用していますが、以下の文書を参考にしてみてください。

- https://micropython-docs-ja.readthedocs.io/ja/latest/esp32/quickref.html

ESP32 用のクイックリファレンスがあり、ESP32 で MicroPython を使用する際に大変役立ちます。ただし、現在の記述は ESP32-C3-MINI-1 ではなく、基本の ESP32 を対象にした記述になっているので、ピン番号など読み替えが必要な部分があるのでご注意ください。

- `https://micropython-docs-ja.readthedocs.io/ja/latest/library/index.html`
  MicroPython のライブラリに関しては、このページにまとめられています。

ESP32-C3M-TRY の入出力を操作するための信号端子に関しては、5.3 節をご参照ください。

3.6.1　LED の点滅

電子工作で定番の LED の点滅プログラムです。ESP32-C3M-TRY に搭載されている LED1 を点滅させるプログラムです。

LED1 は 0 番ピンに接続されています。

```
from machine import Pin
import time

led = Pin(0, Pin.OUT) # D0 を出力に設定

while True:
    led.on()
    time.sleep_ms(1000) # 1000ms(1 秒) 待つ
    led.off()
    time.sleep_ms(1000)
```

図 3.43　LED の点滅

プログラムは無限ループですが、CTRL-C の入力で中断させることができます。

Arduino を使用したプログラミングでは、開発ボードにアップロードしたプログラムは、次にプログラムをアップロードするまで処理を中断させることができませんでしたので、少し不思議な感覚かもしれません。

MicroPython では、プログラムの実行を中断した後に、少しプログラムを追加変更して実行することもできますし、全く異なるプログラムを入力して実行することもできます。

3.6.2　カラー LED の点灯

ESP32-C3M-TRY には、WS2812 タイプのカラー LED が 3 個搭載されています。

ESP32-C3M-TRY に搭載された 3 つのカラー LED の点灯プログラムです。WS2812 タイプの LED の操作ライブラリ neopixel は、MicroPython の基本的なライブラリとしてあらかじめ組み込まれていますので、個別のインストール作業など行う必要がありません。ただ単にライブラリを import して使用できます。

カラー LED は 10 番ピンに接続されています。

```
from machine import Pin
from neopixel import NeoPixel

# NeoPixel の初期化
rgb = NeoPixel(Pin(38, Pin.OUT), 3) # D38 に 3 個の LED

# 各 LED の色の設定と表示
rgb[0] = (50,0,0) # LED3: 赤
rgb[1] = (0,50,0) # LED4: 緑
rgb[2] = (0,0,50) # LED5: 青
rgb.write() # 色の設定を全ての LED の表示に反映
```

図 3.44　WS2812 の点灯

neopixel ライブラリの利用法は、下記のページを参照してください。

- https://micropython-docs-ja.readthedocs.io/ja/latest/esp8266/tutorial/neopixel.html

3.6.3　スイッチと圧電スピーカー

スイッチを押すと圧電スピーカーが鳴るプログラムです。圧電スピーカーには、鳴らしたい音の周波数の交流信号を与える必要がありますが、それは PWM 機能を利用して作成しています。また、音の発声を簡単に操作できるように、TONE クラスを作成して利用しています。SW1, SW2 をそれぞれぞ個別に押した場合と同時に押した場合に、それぞれ異なる周波数の音が出力されます。

圧電スピーカーは 21 番ピンに接続されています。また、今回使う SW1, SW2 はそれぞれ 2 番、3 番ピンに接続されています。SW1, SW2 は負論理となっているので、それぞれの値が 0 の場合に、スイッチが押されたものと判断しています。

```
from machine import Pin, PWM
import time

class TONE:
    def __init__(self, pin):
        self._tone = PWM(Pin(pin), duty=512)
        self._tone.deinit()

    def on(self, hz):
        self._tone.init(hz)

    def off(self):
        self._tone.deinit()

snd=TONE(21) # 圧電スピーカー

sw1=Pin(2, Pin.IN)
sw2=Pin(3, Pin.IN)

while True:
    hz = 0
    if (sw1.value() == 0):
        hz += 440
    if (sw2.value() == 0):
        hz += 880

    if (hz == 0):
        snd.off()
    else:
        snd.on(hz)

    time.sleep_ms(100)
```

図 3.45　圧電スピーカーの発音

3.6.4　OLED ディスプレイへの出力

ESP32-C3M-TRY の CN2 に OLED ディスプレイを搭載することができます。OLED ディスプレイを使用するためには、ssd1306 ライブラリをボードに登録します。

OLED ディスプレイの使用に先立つ初期化は以下の 2 ステップで行います。

- I2C 回線の設定が行われていなければ、その設定を行う。
- 使用する i2c 回線を指定した OLED ディスプレイの設定を行う。

上記の初期化が終わったら OLED への文字等の表示を行います。

文字等の表示は以下の 2 ステップで行います。

- 表示する内容を OLED(に対応したフレームバッファ) に書き込む。(必要な内容を必要な回数)
- フレームバッファの内容を OLED の表示に反映させる。

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

# OLED の設定
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000) # i2c 回線の設定
oled = SSD1306_I2C(128, 64, i2c) # i2c を使用する OLED ディスプレイの設定

# OLED への文字列の出力
oled.text("Hello World", 0, 0, 1) # 表示文字列の書き込み
oled.text("MicroFan", 0, 16, 1)
oled.text("ESP32-C3M-TRY", 0, 32, 1)
oled.show() # 表示内容の OLED への反映
```

図 3.46　OLED ディスプレイへの出力

ssd1306 ライブラリの利用法は、下記のページを参照してください。

- https://micropython-docs-ja.readthedocs.io/ja/latest/esp8266/tutorial/ssd1306.html

3.6.5　温度、湿度、明るさセンサー

温度、湿度、明るさを調べて OLED に表示するプログラムです。温度と湿度は、AHT21 センサーから読み取り、明るさはフォトトランジスタから読み取ります。

AHT21 のライブラリに関しては、以下の URL をご参照ください。

- https://www.microfan.jp/2023/01/aht21/

AHT21 は OLED と同じ I2C 制御線に接続されています。フォトトランジスタは、1 番ピンに接続されており、ADC 機能を利用して出力されたアナログの電圧値をデジタル値に変換して読み取っています。

```
from machine import Pin, I2C, ADC
import time
from ssd1306 import SSD1306_I2C
from ahtx0 import AHT20

i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)
aht21 = AHT20(i2c)
brt = ADC(Pin(1))

while True:
    oled.fill(0)
    oled.text('TEMP: {:.1f}\''.format(aht21.temperature),0,0,1)
    oled.text('HUM: {:.1f}%'.format(aht21.relative_humidity),0,16,1)
    oled.text('BRT: {:.3f}'.format(brt.read()/4095),0,32,1)
    oled.show()
    time.sleep_ms(1000)
```

図 3.47　温度・湿度・明るさの計測

3.6.6　加速度センサー

基板にかかる加速度（重力）を調べて OLED に表示するプログラムです。加速度は、KXTJ3-1057 センサーから読み取ります。基板の縦方向が X 軸、横方向が Y 軸、前後方向が Z 軸になっています。

KXTJ3-1057 のライブラリに関しては、以下の URL をご参照ください。

- https://www.microfan.jp/2023/05/kxtj3-1057/

KXTJ3-1057 の初期化状態では、各軸は 2g がフルスケールになっており、フルスケールの 2g が 32767 で表現されます。計測値をその二分の一相当の 16384 で割ると、加速度が 1g(9.8m/s2) で正規化された形で得られます。

最後に表示されている角度 DEG は、基板の Z 軸を地面と水平にし、Z 軸を回転軸として基板を回転させた場合の基板の Y 軸と地面との傾き角を示します。

。。。動かして表示を確認するのが簡単です。

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import time
from kxtj3 import KXTJ3
import math

# i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
i2c = I2C(0)
oled = SSD1306_I2C(128, 64, i2c)
acc = KXTJ3(i2c)

while True:
    oled.fill(0)
    vacc = acc.getXYZ()
    oled.text('X: {:.2f}'.format(vacc[0] / 16384), 0, 0, 1)
    oled.text('Y: {:.2f}'.format(vacc[1] / 16384), 0, 16, 1)
    oled.text('Z: {:.2f}'.format(vacc[2] / 16384), 0, 32, 1)
    t = math.degrees(math.atan2(-vacc[1], vacc[0]))
    oled.text('DEG: {:.1f}'.format(t), 0, 48, 1)
    oled.show()
    time.sleep_ms(100)
```

図 3.48　加速度センサーの利用

数字の表示だけではわかりにくいので、グラフィックス機能を利用して、水平儀として表示する例を次に示します。

```
from machine import Pin, I2C
import time
from ssd1306 import SSD1306_I2C
from kxtj3 import KXTJ3
import math

i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)
acc = KXTJ3(i2c)

while True:
    oled.fill(0)
    vacc = acc.getXYZ()
    x = int(vacc[0] / 16384 * 40)
    y = int(vacc[1] / 16384 * 40)
    oled.line(64-x,32-y,64+x,32+y,1)
    oled.ellipse(64,32,40,40,1)
    t = math.degrees(math.atan2(-vacc[1], vacc[0]))
    oled.text('{:4.1f}'.format(t), 45, 36, 1)
    oled.show()
    time.sleep_ms(100)
```

図 3.49　加速度センサーを使用した重力加速度の計測

3.6.7　ヒープメモリの容量

MicroPython では、実行時のデータの格納領域として、ヒープメモリと呼ばれる領域を使用します。ヒープメモリの容量により、その開発ボードで大きなデータを扱えるかどうかが決まります。

ヒープメモリの容量を確認する方法を示します。

```
>>> import gc
>>> print(gc.mem_free())
124816
>>>
```

図 3.50　ヒープメモリ容量の確認

gc ライブラリの利用法は、下記のページを参照してください。

- https://micropython-docs-ja.readthedocs.io/ja/latest/library/gc.html

プログラムを作成する上でヒープメモリが不足するようであれば、大容量の PSRAM を搭載した以下の開発ボードを使用することをお勧めします。

- ESP32-S3-KEY-R2: 8MB 程度のヒープメモリを使用できます。
  https://www.amazon.co.jp/dp/B0C3XLHB5D/
- ESP32-WROVER-KEY-R2: 4MB 程度のヒープメモリを使用できます。(MicroPython のファームウェアの設定を変更して再コンパイルすれば 8MB 程度のヒープメモリを使用可能)
  https://www.amazon.co.jp/dp/B0BPWP5GKS/

第 4 章

Arduino スケッチ環境の整備

4.1 ESP32 用 Arduino 開発環境のインストール

Arduino の開発環境のインストールは以下の 2 段階の手順で行います。

- 基本となる Arduino IDE のインストール
- ESP32 用の開発機能の追加

この後は、必要に応じて、各種のライブラリの追加インストールを行います。

下記のインストール法がわかりにくい様であれば、WEB で検索をするとインストール法を示したページが複数見つかるので、ご自身がわかりやすいと思うページを参照してインストールを行ってください。

4.1.1 基本となる Arduino IDE のインストール

以下のページからダウンロードオプションで、ご自身が使用している OS 用のインストールパッケージを選択しダウンロードしインストールしてください。

- `https://www.arduino.cc/en/software`

Arduino IDE がインストールできたら起動してください。

メニュー等を日本語化するために、Arduino IDE の [File] ⇒ [Preferences...] ⇒ [Settings] タブの [Language:] を日本語に設定してください。

4.1.2 ESP32 用の開発機能の追加

ESP32 用の Arduino は以下の WEB ページで公開されています。

- `https://github.com/espressif/arduino-esp32`

ESP32-C3 用の開発機能もこのページの手順でインストールされるパッケージに含まれています。

インストール方法も示されているので、示されている手順に従って ESP32 用の Arduino のインストールを行ってください。

4.2　USB ケーブルの接続

USB ケーブルで ESP32-C3M-TRY を PC に接続します。

ESP32-C3M-TRY は USB インターフェースに ESP32-C3-MINI-1 内蔵の USB/JTAG ポートを使用しているため、スケッチやファームウェアの書き込みで書き込みがうまく行えない場合があります。この様な場合には RST と LOAD スイッチを一緒に押し、まず RST スイッチを離し次に LOAD スイッチを離す事により、ESP32-C3-MINI-1 を BOOT ローダーモードに移行させることにより、ファームウェア等の書き込みが行えるようになります。

なお、RST スイッチを押すと ESP32-C3-MINI-1 の初期化により、一旦 USB 接続が失われます。したがって、RST スイッチを押すたびに IDE では再度 USB ポートの設定を行う必要があります。

上記の手順で ESP32-C3-MINI-1 を BOOT ローダーモードにしてファームウェア等を書き込んだあとは、書き込み終了後に RST スイッチを押して ESP32-C3-MINI-1 を通常の状態に戻してやる必要があります。

4.3　ESP32-C3 用 Arduino 開発環境の設定

Arduino IDE のメニューの「ツール」を選択してメニューを表示してください。このメニューの中から、まず、開発用のボードと PC と ESP32-C3M-TRY の接続を行うポートを設定します。

開発ボードの選択は、[ボード:] メニューの esp32 の中から、初めの方に表示されている [**ESP32C3 Dev Module**] を選択してください。

ESP32-C3M-TRY が PC に USB で接続されていることを確認してください。

RST と LOAD スイッチを使用して、ESP32-C3-MINI-1 を BOOT ローダーモードに移行させてください。

[ポート:] は、 ESP32-C3M-TRY が接続されているポートが **COMX(X は数値)** の形式で表示されているので、それを選択して設定してください。

ESP32-C3M-TRY では RST ボタンを押すと USB 接続がいったん解除されるため、RST ボタンを押した場合には、再度 [ポート:] を選択する必要があるので注意してください。

MAC の場合には、ポートの選択方法は WEB などで確認してください。

最後に、フラッシュサイズに関連する以下の設定をメニューから行ってください。他の設定は既定の状態で問題ありません。

基本的には、4M のフラッシュサイズに適合させます。

- Flash Size
  4MB を選択
- Partition Scheme

2 種類ある　[Default 4M —- (—-)]　のどちらかを選択

4.4　サンプルスケッチの実行

ESP32 用の Arduino をインストールすると、Arduino IDE の [ファイル] ⇒ [スケッチの例] に、ESP32 用の多くのサンプルスケッチが追加されます。これらのサンプルスケッチを試すことで、ESP32 のプログラミングを学ぶことができます。

ESP32-C3M-TRY に初めから書き込まれている LED の点滅スケッチとかち合いますが、ここでは、ESP32-C3M-TRY の動作確認のために、LED 点滅スケッチの実行（更新）を試してみましょう。

4.4.1　BLINK:LED の単純な点滅

電子工作界の hello world、LED の点滅スケッチを実行しましょう。

Arduino IDE の [ファイル] ⇒ [スケッチの例] ⇒ [01.Basics] から Blink を選択してください。ESP32-C3M-TRY の LED は 0 番ピンに接続されているので、スケッチの pinMode(),digitalWrite() の第 1 引数の LED_BUILTIN を 0 に変更します。

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 7 as an output.
  pinMode(0, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(0, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(0, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

図 4.1　BLINK:LED の単純な点滅

4.4.2　スケッチを書き込む際の注意

ESP32-C3M-TRY では、スケッチの書き込み時に LED などは点滅しません。

ESP32-C3M-TRY では、次の節の操作でスケッチを ESP32-C3M-TRY に書き込む際に、エラーが発生して書き込めない場合があります。

特に、初めて ESP32-C3M-TRY にスケッチを書き込む際には、必ずエラーになります。このため、初めてスケッチを書き込む際やスケッチの書き込み時にエラーが発生する場合には、まず

RST ボタンと LOAD ボタンを押した状態にし、次に RST ボタンを離し、最後に LOAD ボタンを離す操作をしてください。

この操作で、ESP32-C3M-TRY はスケッチの書き込み用の状態に設定されるので、エラーなくスケッチを書き込むことができるようになります。なお、このボタンの操作をした場合、IDE の [ポート:] の設定が未設定になってしまうので、スケッチの書き込み処理の前に、再度適切なポート番号を設定するようにしてください。

また、上記のボタンの操作をして ESP32-C3M-TRY にスケッチを書き込んだ場合には、書き込み終了後にスケッチが自動的に起動しないので、RST ボタンを押してスケッチが起動するようにしてください。

4.4.3　スケッチのコンパイルと ESP32-C3M-TRY への書き込み

スケッチの入力・修正が終わったら、まず問題なくコンパイルを行えるかどうか、Arduino IDE の左上部のチェックマーク [検証] のアイコンをクリックして、スケッチをコンパイルします。

問題なくコンパイルできたならば、先ほどのアイコンの右隣の右矢印マーク [書き込み] のアイコンをクリックします。スケッチの再コンパイルの後に、Arduino IDE の下部のメッセージエリアに白色の文字で多数の行のメッセージが出て、スケッチの書き込みが開始されます。

スケッチが ESP32-C3M-TRY に正しく書き込まれたら、ボード上の LED が点滅します。

新たにアップロードしたスケッチは、先に書き込まれていたスケッチと同じなので、代り映えがせず本当にスケッチが更新されたか分かりにくいと思われる場合には、delay() の引数を変更して、LED の点滅の間隔などを変えてみるとよいでしょう。

4.4.4　スケッチの書き込みに失敗した場合

5.2 節に示す BOOT ローダーモードへの移行を行い、再度スケッチをアップロードしてください。なお、開発ボードを BOOT ローダーモードに移行させた後に、IDE のシリアルポートを再設定することを忘れないようにしてください。

4.5　カラー LED WS2812 の利用

ESP32-C3M-TRY にはカラー LED:WS2812 が 3 個搭載されています。WS2812 には、LED の点灯制御用の IC が組み込まれており、この IC のおかげで、複数のカラー LED の色や明るさを 1 本の信号線で制御できるのですが、その制御信号として、高速かつ正確なタイミングの制御信号を生成する必要があります。このようなプログラムを作成するためには、MCU のマシン語を用いたプログラミングの知識なども必要になり少々面倒なのですが、Adafruit 社がオープンソースライセンスで公開している NeoPixel ライブラリを利用すると、WS2812 を簡単に制御できます。

NeoPixel ライブラリは以下の URL で公開されています。

- `https://github.com/adafruit/Adafruit_NeoPixel`

Adafruit の NeoPixel に関する Web ページは次のとおりです。

- https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels

4.5.1　NeoPixel ライブラリのインストール

NeoPixel ライブラリは、Arduino IDE のライブラリマネージャーを利用して簡単にインストールすることができます。

Arduino IDE のメニューバーから、[ツール] ⇒ [ライブラリを管理...] を選択します。IDE の左側にライブラリマネージャが開き、最上部にライブラリの検索文字を入力することができます。ここに [neopixel] と入力すると、該当するライブラリが絞り込まれます。

この状態ではまだ候補が 20 程度ありますが、[Adafruit NeoPixel by Adafruit] の項目を見つけてインストールボタンをクリックします。

4.5.2　NeoPixel ライブラリの利用

ライブラリがインストールされると、メニューバーの [ファイル] ⇒ [スケッチ例...] を選択すると、メニューの下の方の [カスタムライブラリのスケッチ例] の中にインストールされた NeoPixel ライブラリに関する項目が追加されていることがわかります。その項目を選択すると、いくつかのデモプログラムが表示されます。

カラー LED の動作テストとしては、カラー LED の鮮やかな色の変化を楽しめる strandtest がよいでしょう。スケッチを開いたら、図 4.2 に示す様に 2 つの定数を ESP32-C3M-TRY に合わせて設定します。

```
#define LED_PIN    10
#define LED_COUNT 3
```

図 4.2　カラー LED のピンと個数の設定

strandtest をコンパイル・実行させると、ESP32-C3M-TRY 上の 3 個のカラー LED が、様々に色を変えながら点滅します。

4.5.3　カラー LED の追加利用

WS28212 を搭載したカラー LED ボードは、カラー LED がマトリックス状に配置されたもの、円環上に配置されたもの、一列に配置されたものなど、様々な製品が販売されています。それらの製品は,CN4 に接続し、追加された個数を含めて LED_COUNT にカラー LED の個数を設定することで、開発ボード上のカラー LED と同様に制御することができます。

CN4 にカラー LED を接続して利用する際には、追加したカラー LED の消費電流が課題にならないように注意することと、もし消費電流が大きくなる場合には、追加したカラー LED には別途

5V の電源を接続するようにしてください。

4.6　OLED ディスプレイの利用

ESP32-C3M-TRY は、多様な情報の表示装置として OLED ディスプレイを搭載することができます。

4.6.1　U8g2 ライブラリのインストール

OLED ディスプレイを利用するためのライブラリとして、U8g2 ライブラリを使用する例を示します。この他にも、よく利用されるライブラリとして Adafruit の SSD1306 などがあります。

U8g2 ライブラリは、Arduino IDE のライブラリマネージャを利用してインストールすることができます。ライブラリマネージャの検索フィルタに [U8g2] を入力して絞り込むと、図 4.3 のように表示されます。

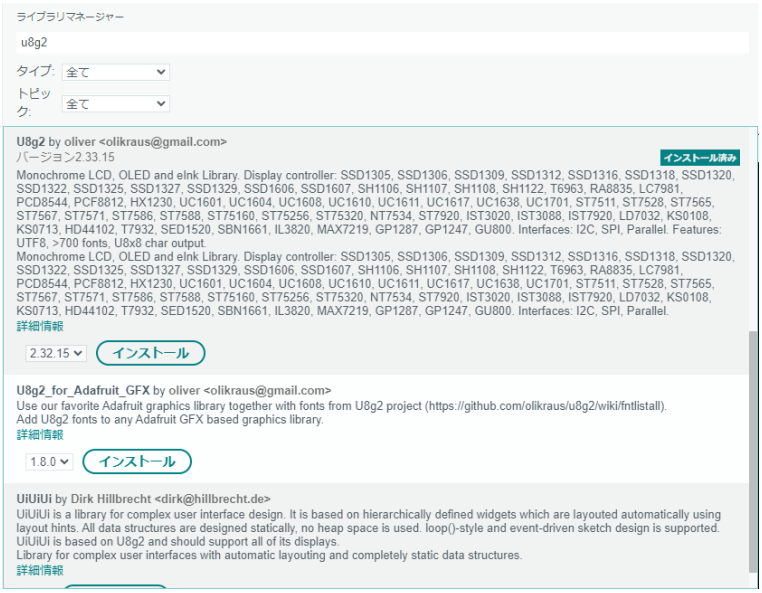


図 4.3　ライブラリマネージャを利用した U8g2 ライブラリの導入

ライブラリマネージャのダイアログ上でインストールするライブラリの欄をクリックすると、インストールボタンが表示されるので、最新バージョンを選択して、ライブラリをインストールします。関連するライブラリも合わせてインストールするか聞かれた場合には、それらもすべてインストールしてください。

このライブラリは、以下の URL で取得することもできます。

`https://github.com/olikraus/u8g2`

また、マニュアルは、以下の URL で参照することができます。

`https://github.com/olikraus/u8g2/wiki`

4.6.2　U8g2 ライブラリの利用

ライブラリのインストール後、Arduino IDE メニューから [ファイル] ⇒ [スケッチの例] を選択すると、リストの下の方に U8g2 フォルダが追加されているのが確認できます。U8g2 フォルダの中を確認するといくつかのサンプルスケッチがあり、選択して実行することができます。

例として [u8g2] ⇒ [full_buffer] ⇒ [GraphicsTest] を選択します。このスケッチは、デモ用の簡単なグラフィック表示を行うものです。

このサンプルスケッチをコンパイル・実行するためには、コメントアウトされたリストの中から適切な u8g2 コンストラクタを選択するか、自分自身で追加する必要があります。ここで利用する OLED ディスプレイは、コントローラとして SSD1306 を使用しており、I2C インターフェースで接続されているので、サンプルスケッチの 65 行あたりの、以下のコンストラクタを有効にしてください。

```
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
```

図 4.4　OLED ディスプレイ (SSD1306) 用のコンストラクタ

このコンストラクタを有効にすることにより、スケッチのコンパイル、実行ができるようになります。

4.7　気温・湿度・明るさセンサー

ESP32-C3M-TRY の気温・湿度センサーには、AHT21 が使用されています。

AHT21 の制御には、AHT20 用のライブラリを使用することができます。ここでは、Adafruit AHTX0 ライブラリを使用した例を示します。ライブラリマネージャを使用して Adafruit AHTX0 ライブラリを導入して使用してください。

図 4.5 に OLED ディスプレイに気温・湿度センサー AHT21 で得られた気温と湿度、明るさセンサー（フォトトランジスタ）で得られた明るさを表示するスケッチ例を示します。

```
#include <U8x8lib.h>
#include <Adafruit_AHTX0.h>

U8X8_SSD1306_128X64_NONAME_HW_I2C u8x8(U8X8_PIN_NONE);
Adafruit_AHTX0 aht;

void setup() {
  pinMode(21, OUTPUT) ; // 圧電スピーカーのノイズ止め

  u8x8.begin();
  u8x8.setFont(u8x8_font_amstrad_cpc_extended_r);
  u8x8.clearDisplay();
  aht.begin(); // AHT21 の初期化
}

char tstr[20];

void loop() {
  sensors_event_t humidity, temp;

  aht.getEvent(&humidity, &temp); // AHT21 から湿度と温度の取得

  u8x8.draw1x2String(0, 0, "TEMP: ");
  dtostrf(temp.temperature, 4, 1, tstr);  // sprintf は実数の変換ができない
  u8x8.draw1x2String(6, 0, tstr);
  u8x8.draw1x2String(10, 0, "'C");
  sprintf(tstr, "%2d%%", (int)humidity.relative_humidity);
  u8x8.draw1x2String(0, 2, "HUM: ");
  u8x8.draw1x2String(5, 2, tstr);
  u8x8.draw1x2String(0, 4, "BRT: ") ;
  dtostrf(analogRead(2)/4095.0, 4, 2, tstr);  // sprintf は実数の変換ができ
ない
  u8x8.draw1x2String(5, 4, tstr);

  delay(1000);
}
```

図 4.5　気温・湿度・明るさセンサー使用のスケッチ例

4.8　加速度センサー

ESP32-C3M-TRY の加速度センサーには、XKTJ3-1057 が使用されています。

ここでは、XKTJ3-1057 ライブラリを使用した例を示します。ライブラリマネージャを使用して XKTJ3-1057 ライブラリを導入して使用してください。

図 4.6 に OLED ディスプレイに 3 軸の加速度と、基板の傾きを表示するスケッチ例を示します。

```
#include <U8x8lib.h>

U8X8_SSD1306_128X64_NONAME_HW_I2C u8x8(U8X8_PIN_NONE);

// Accelerometer provides different Power modes by changing output bit resolution
#define LOW_POWER
//#define HIGH_RESOLUTION

#include <kxtj3-1057.h>
#include <Wire.h>
#include <math.h>

float sampleRate = 6.25;  // HZ - Samples per second - 0.781, 1.563, 3.125, 6.25, 12.5
uint8_t accelRange = 2;   // Accelerometer range = 2, 4, 8, 16g

KXTJ3 myIMU(0x0E);  // Address can be 0x0E or 0x0F

void setup() {
  u8x8.begin();
  u8x8.setFont(u8x8_font_amstrad_cpc_extended_r);
  u8x8.clearDisplay();

  pinMode(21, OUTPUT) ; // 圧電スピーカーのノイズ止め

  myIMU.begin(sampleRate, accelRange);
  // Detection threshold, movement duration and polarity
  myIMU.intConf(123, 1, 10, HIGH);
}

char tstr[20];

void loop() {
  float x, y, t ;

  myIMU.standby(false);

  u8x8.draw1x2String(0, 0, "X: ");
  dtostrf(x = myIMU.axisAccel(X), 6, 3, tstr);  // sprintf は実数の変換がで
きない
  u8x8.draw1x2String(3, 0, tstr);
  u8x8.draw1x2String(0, 2, "Y: ");
  dtostrf(y = myIMU.axisAccel(Y), 6, 3, tstr);
  u8x8.draw1x2String(3, 2, tstr);
  u8x8.draw1x2String(0, 4, "Z: ");
  dtostrf(myIMU.axisAccel(Z), 6, 3, tstr);
  u8x8.draw1x2String(3, 4, tstr);

  t = atan2(-y, x) * 180 / 3.14 ;
  u8x8.draw1x2String(0, 6, "DEG: ");
  dtostrf(t, 6, 1, tstr);
  u8x8.draw1x2String(5, 6, tstr);

  myIMU.standby(true);

  delay(1000);
}
```

第 5 章

資料

5.1　ESP32-C3M-TRY の回路図

ESP32-C3M-TRY の回路図を図 5.1、部品表を表 5.1 に示します。

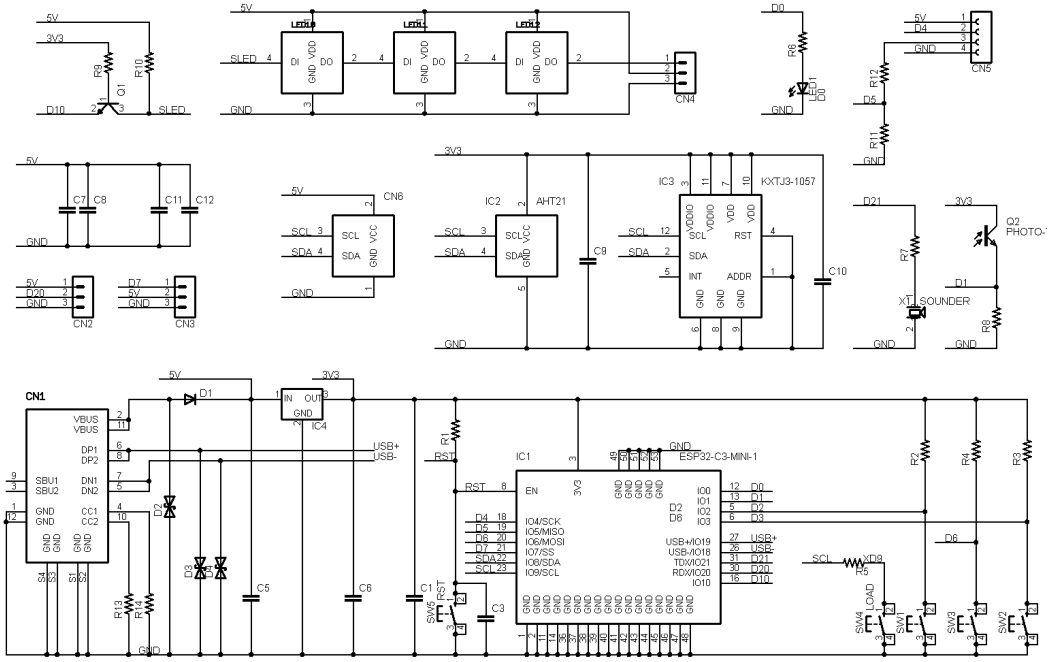


図 5.1　ESP32-C3M-TRY の回路図

表 5.1　部品表

部品	シンボル	規格等	1
プリント基板	ESP32-C3M-TRY	Rev.1	1
IC	IC1	ESP32-C3-MINI-1(RISC-V, 4M)	1
	IC2	AHT21	1
	IC3	KXTJ3-1057	1
	IC4	BL8071	1
トランジスタ	Q1	SS8050	1
フォトトランジスタ	Q2	SMD1206-30	1
ショットキーダイオード	D1	SS14	1
ESD ダイオード	D2-D4	LESD5D5.0CT1G	3
発光ダイオード	LED1	青	1
カラー LED	LED10-LED12	WS2812	3
抵抗	R1-R4,R8-R10	10K Ω	7
	R5	220 Ω	1
	R6,R7,R12	1K Ω	3
	R11	2.2K Ω	1
	R13,R14	5.1K Ω	2
セラミックコンデンサ	C1, C5	10 μ F	2
	C3	1 μ F	1
	C4,C7-C10	100nF	5
	C6	47 μ F	1
	C11,C12	22 μ F	2
タクトスイッチ	SW1, SW2, SW3	4 端子	3
	SW4, SW5	4 端子	2
OLED ディスプレイ	CN6	4 ピン	1
圧電スピーカー	X1		1
USB	CN1	Type-C	1
拡張端子	CN2	3 ピン ピンソケット	1
	CN3	3 ピン ピンヘッダー	1
	CN4	3 ピン ピンソケット	1
	CN5	4 ピン ピンソケット	1

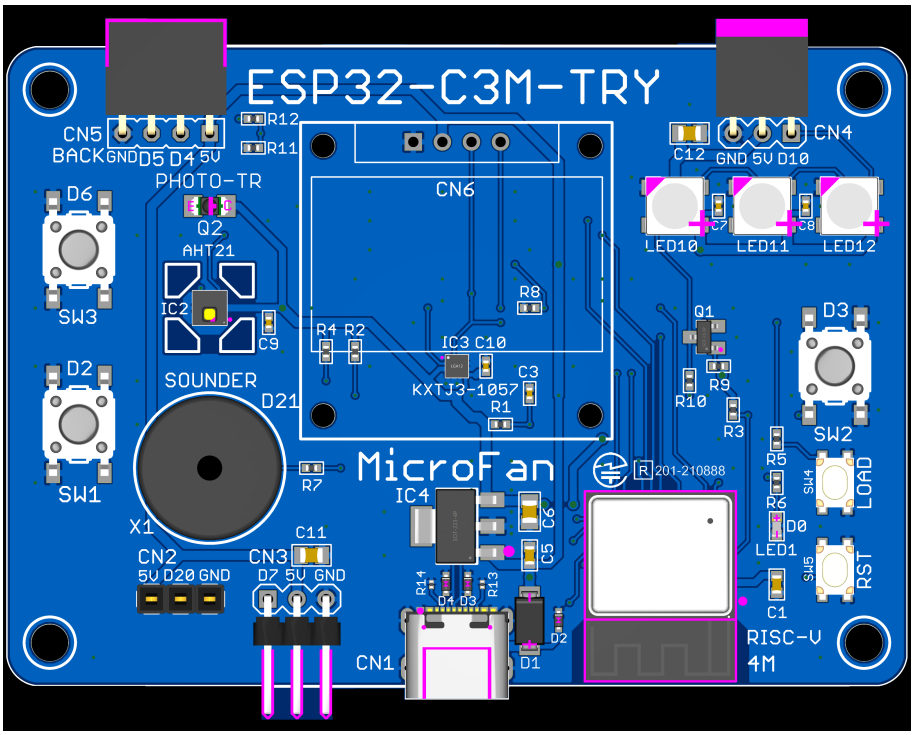


図 5.2　ESP32-C3M-TRY の部品配置

5.2　RST および LOAD スイッチ

ESP32-C3M-TRY には、リセット用のスイッチ RST と、ESP32-C3-MINI-1 をスケッチやファームウェアのアップロードを行う BOOT モードに移行させるための LOAD スイッチが搭載されています。

5.2.1　リセット

単に RST スイッチを押し、その後離します。

ESP32-C3-MINI-1 はリセットされ、フラッシュに記録されているスケッチもしくはファームウェアの実行を開始します。

5.2.2　BOOT ローダーモードへの移行

スケッチやファームウェアのアップロードを行う際は、以下に説明する操作は行う必要がない場合が多いのですが、ESP32-C3-MINI-1 の状態により、PC からの操作だけスケッチやファームウェアのアップロードがうまくいかない場合に使用してください。

まず、RST と LOAD の両方のスイッチを押した状態にします。その後、まず RST スイッチを離し、次に LOAD スイッチを離します。この操作により、ESP32-C3-MINI-1 はスケッチや

ファームウェアのアップロードを行う BOOT ローダーモードに移行します。

なお、RST スイッチを押すと ESP32-C3-MINI-1 の初期化により、一旦 USB 接続が失われます。したがって、RST スイッチを押すたびに IDE では USB ポートの再設定を行う必要があります。

また、この様に ESP32-C3-MINI-1 を BOOT ローダーモードにしてファームウェア等を書き込んだ場合には、書き込み終了後に RST スイッチを押して、ESP32-C3-MINI-1 を再起動して通常の状態に戻す必要があります。

5.3　開発ボード上の入出力

ESP32-C3M-TRY の入出力の信号線を表 5.2 に示します。

SW4 は、リセット時のブートモード（Arduino IDE などからのスケッチ書き込み）の切り替え用ですが、信号線が I2C の SCL 信号と共用のため、プログラムが走り始めたあとは、一般的な入力用のスイッチとして利用するこはできません。

表 5.2　入出力の信号線

シンボル	信号線	備考
SW1	D2	負論理
SW2	D3	負論理
SW3	D6	負論理
SW4	D9/SCL	BOOT ローダーモード移行用
SW5	RST	リセット
LED1	D0	正論理
LED10-LED12	D10	WS2812B, CN4
SCL0	D9	OLED ディスプレイ、温度センサ、加速度センサ
SDA0	D8	
圧電スピーカー	D21	
明るさセンサー	D1	アナログ値
CN2	D20	人感センサー
CN3	D7	RC サーボ
CN5 TRIG	D4	超音波距離センサー
CN5 ECHO	D5	

CN2-CN5 のコネクタは、標準的な使用法は決まっていますが、電源や信号線を適切に使用するならば、標準的な使用法以外の拡張用端子として使用することができます。

第 6 章

購入および問い合わせ先

6.1　ご協力のお願い

製品をより良くし、多くの方々にお楽しみいただけるよう、製品の向上に努めて参ります。問題点やお気づきの点、あるいは製品の企画に対するご希望などございましたら、microfan_shop@yahoo.co.jp までご連絡いただけますようよろしくお願いいたします。末永くご愛顧いただけますよう、お願いいたします。

6.2　販売：ネットショップ

製品の販売はネットショップで行っています。対面販売は行っておりません。

- マイクロファン Yahoo!ショップ
  WEB アドレス：https://store.shopping.yahoo.co.jp/microfan/
- アマゾン
  WEB アドレス：https://www.amazon.co.jp/s?merchant=A28NHPRKJDC95B

6.3　製品情報

マイクロファン ラボ
WEB アドレス：http://www.microfan.jp/
マイクロファンの製品情報や活用情報を紹介しています。

6.4　問い合わせ先

株式会社ピープルメディア　マイクロファン事業部
E-Mail: microfan_shop@yahoo.co.jp
TEL: 092-938-0450
お問い合わせは基本的にメイルでお願いいたします。

6.5　所在地

株式会社ピープルメディア　マイクロファン事業部
〒811-2316 福岡県糟屋郡粕屋町長者原西 2-2-22-503