# ESP32-SLIM-R1
# (ESP-WROOM-32 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 4 月

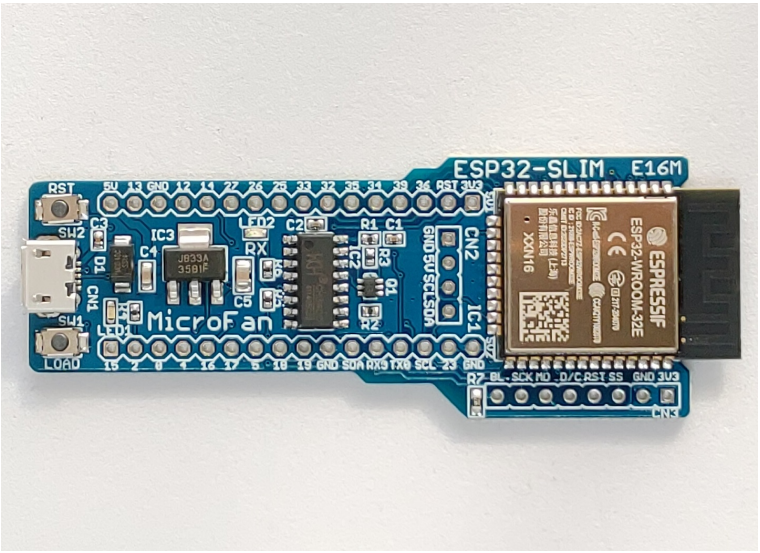
# 目次

# 表目次

# 図目次

# 第 1 章

# ESP32-SLIM-R1 の紹介




## 1.1 製品概要

近年様々なモノをインターネットに接続してサービスの高度化を図るモノのインターネット「IoT [*1]」が注目されており、IoT サービスを実現するための様々な開発や実験が、企業はもちろん個人でも行われています。その IoT 装置を実現する中核部品として、WiFi や Bluetooth が標準装備された ESP-WROOM-32 が広く使用されています。

ESP32-SLIM-R1 は ESP-WROOM-32 を利用した IoT 機器の開発や実験を、ブレッドボード上で手軽に行うための開発ボードとして開発されました。ESP32-SLIM-R1 は Arduino の基本機能を一通り習得し、無線 LAN 機能を活用した応用に取り組みたい人に最適な開発ボードです。

ESP32-SLIM-R1 は以下のような特徴を持っています。

- ESP32 の E バージョンの 16M 版を搭載しています。

[*1] Internet of Things

- 高性能の 32 ビットマイクロプロセッサを搭載することで、Arduino UNO R3 などと比較して高速な処理が行えるとともに、大容量の FLASH(16M) と RAM を利用できます。
- 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。
- 電子工作で広く利用されている Arduino などの無償、便利、高機能な開発環境を利用してソフトウェアを開発できます。
- Arduino IDE で作成したスケッチを書き込むための 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 [*2]の電圧レギュレータを搭載し、ESP-WROOM-32 に安定した電源を供給できます。
- ESP-WROOM-32 の信号線がピンヘッダーを取り付け可能な端子列に引き出されており、ブレッドボードに挿して利用できます。
- 端子列の幅はブレッドボードを効果的に活用できるよう狭く（40 ピン DIP と同じ間隔）設計されています。
- 様々な情報を表示できる OLED ディスプレイ (別売) や TFT ディスプレイを搭載することができます。

## 1.2　購入・利用上の注意

ESP32-SLIM-R1 をご購入の際には、下記項目をご確認ください。

- ESP-WROOM-32 の未接続端子
  内部のフラッシュメモリに接続されている ESP-WROOM-32 の 17-22 ピンは、使用上注意が必要なため未接続となっています。
- OLED ディスプレイは別売りです。
  - `https://store.shopping.yahoo.co.jp/microfan/oled096-128x64-i2c-blue.html`
  - `https://www.amazon.co.jp/dp/B06Y4TKL1F`
- TFT ディスプレイは別売りです。
  - `https://store.shopping.yahoo.co.jp/microfan/TFT144-128x128.html`
  - `https://www.amazon.co.jp/dp/B0BN5XGRTW`
- ピンヘッダーは別売りです。
  `https://store.shopping.yahoo.co.jp/microfan/pin-header-40px2.html`
- ブレッドボードは別売りです。
  `https://store.shopping.yahoo.co.jp/microfan/breadboard-63.html`

[*2] 放熱の制限で、継続的に 1.5A の電流を使用することはできません

## 1.3　マニュアルの記載内容に関して

ESP-WROOM-32 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-SLIM-R1 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-SLIM-R1 の特徴

## 2.1 USB インターフェース

Arduino IDE で作成したスケッチを ESP-WROOM-32 に書き込むために、USB インターフェース (microB) を備えています。

USB インターフェースは以下のような用途で使用されます。

- ESP32-SLIM-R1 への電力供給。
  USB からは 5V の電力が供給され、ESP32-SLIM-R1 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-SLIM-R1 へのスケッチ（プログラム) の書き込み。
  スケッチの書き込み時や、USB を介した受信時には、赤色の LED2（RX）が点滅します。
- ESP32-SLIM-R1 と PC 間のシリアル通信。

## 2.2 電源回路

ブレッドボード上で回路の試作や実験を行うためには、電源回路が必要になります。ESP32-SLIM-R1 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。

ESP-WROOM-32 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、ESP-WROOM-32 の動作が不安定になることがあります。

### 2.2.1 電圧レギュレータ

ESP-WROOM-32 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-SLIM-R1 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 ESP-WROOM-32 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。ESP-WROOM-32 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。

### 2.2.2　外部への電力供給と注意点

ESP32-SLIM-R1 が組み込まれているブレッドボード上の回路や、試作基板上の消費電力の小さな回路に対しては、ESP32-SLIM-R1 の 5V 端子や 3.3V 端子からそれぞれ電力を取得して使用することができます。

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-SLIM-R1 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。

### 2.2.3　外部からの電力供給と注意点

USB コネクタを通して電力供給を行わない場合には、5V の端子に電源を接続してください。

- 5V 端子に接続する電源の電圧は 6V を超えないようにしてください。6V を超えると電圧レギュレータをはじめとする回路が破損する可能性があります。
- 外部電源にそれ自身に対する保護回路がついている様であれば、USB コネクタが PC 等に接続された状態でも、5V 端子に外部電源の接続を行って構いません。（逆流防止用のダイオードがついているので、PC 等に外部電源からの電圧・電流が逆流することはありません。）
- 3.3V の端子は出力専用で、外部から 3.3V の電源を接続しないでください。3.3V 端子に外部から電力を供給すると回路が破損する可能性があります。

## 2.3　ブレッドボードの利用

ESP32-SLIM-R1 の 2 列の信号の引き出し端子は、ブレッドボードの部品配置領域を効果的に利用できるように、他の ESP32 開発ボードと比較して狭い間隔（40 ピン DIP IC と同じ）で配置されています。他の ESP32 開発ボードでは部品の配置が難しかった一般的なブレッドボードでも、ESP32-SLIM-R1 では図 2.1 に示すように基板の両横に余裕をもって部品を配置し利用することができます。

[*1] ただし PC に接続された 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。

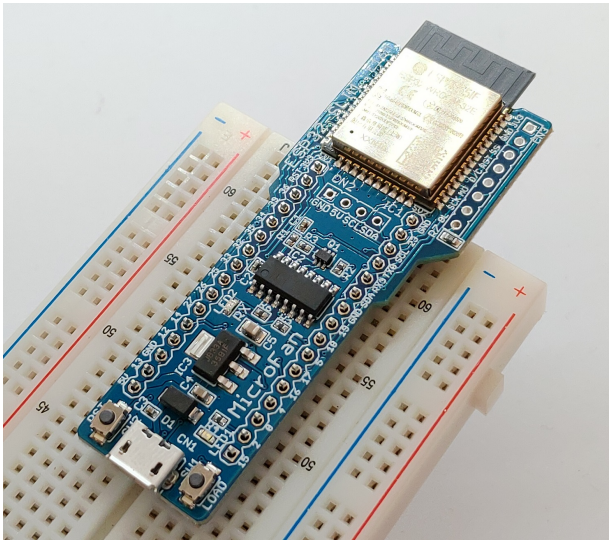


図 2.1　ブレッドボードに乗せた ESP32-SLIM-R1

## 2.4　ディスプレイの利用

ESP32-SLIM-R1 には OLED ディスプレイや TFT ディスプレイの接続端子が装備されているため、それらを基板に搭載して手軽に使用することができます。

ネット上などで公開されている ESP-WROOM-32 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 ESP-WROOM-32 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-SLIM-R1 では、面倒な配線等を行うことなく開発ボード上に OLED ディスプレイや TFT ディスプレイを搭載できるため、PC と切り離して単独で運用している場合でも、様々な情報をそれらのディスプレイに表示し確認することができます。

### 2.4.1　OLED ディスプレイ

ESP32-SLIM-R1 には OLED ディスプレイの接続端子が装備されているため、図 2.2 に示すように手軽に接続し、利用することができます。

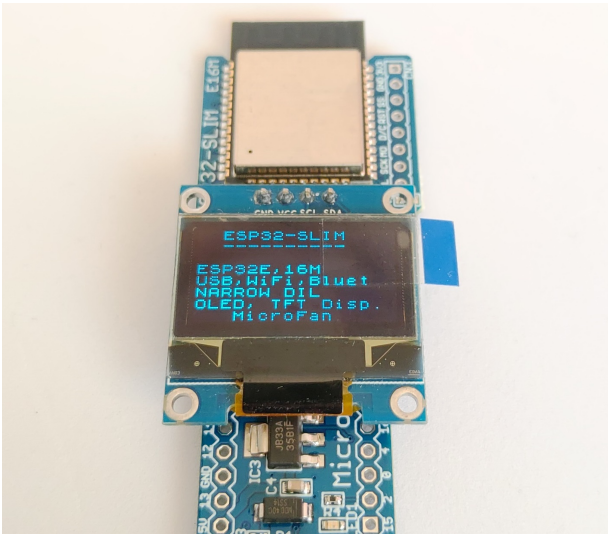



図 2.2　OLED ディスプレイを搭載した ESP32-SLIM-R1

OLED ディスプレイは、128x64 ドットのグラフィックディスプレイになっており、ボードの稼働状態や利用者に伝えたい情報を、画像や文字で分かり易く表示できるようになります。

OLED ディスプレイの端子と表示モジュールに関しては、6.4 節をご参照ください。

### 2.4.2　TFT ディスプレイ

ESP32-SLIM-R1 には TFT ディスプレイの接続端子が装備されているため、図 2.3 に示すように手軽に接続し、利用することができます。

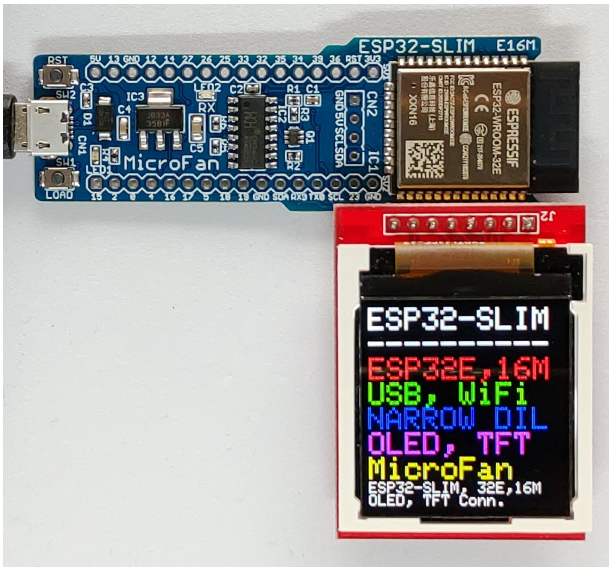



図 2.3　TFT ディスプレイを搭載した ESP32-SLIM-R1

TFT ディスプレイは、128x128 ドットのカラーグラフィックディスプレイになっており、ボードの稼働状態や利用者に伝えたい情報を、カラー画像や文字で分かり易く表示できるようになります。

TFT ディスプレイの端子と表示モジュールに関しては、6.4 節をご参照ください。

# 第 3 章

# 利用の準備

ESP32-SLIM-R1 の利用に先立って、必要に応じて、ピンヘッダやコネクタ類のはんだ付けを行います。

ピンヘッダやコネクタ（ピンソケット）等は、必要に応じて別途ご入手ください。

## 3.1 部品表

ESP32-SLIM-R1 の部品表を表 3.1 に示します。表 3.1 には関連部品を含めて示していますが、ESP32-SLIM-R1 の基本的な商品構成は開発基板 1 つのみです。基板が破損している場合には、ご利用になる前にマイクロファンにお問い合わせください。

表 3.1 部品表

| 部品 | シンボル | 規格等 | 個数 |
|---|---|---|---|
| プリント基板 | ESP32-SLIM | Rev.1 | 1 |
| OLED ディスプレイ | CN2 | 4 ピン | 別売り |
| TFT ディスプレイ | CN3 | 8 ピン | 別売り |
| ピンヘッダ | SV1, SV2 | 1x16PIN X2 | 別売り |

## 3.2 ESP32-SLIM-R1 の動作確認

ESP32-SLIM-R1 は、製造時の基本的な動作確認として、LED1（青色）を点滅させるスケッチを書き込んで動作確認を行っています。

ESP32-SLIM-R1 をご購入なさったら、利用に先立ち基板を USB ケーブルで PC 等に接続してください。LED1 が点滅し、ESP32-SLIM-R1 が稼働することが確認できます。

ここで問題があれば、マイクロファンにお問い合わせください。

## 3.3　ディスプレイの取り付け

ESP32-SLIM-R1 には、OLED ディスプレイや TFT ディスプレイを取り付けて使用することができます

開発基板へのディスプレイの取り付けは、基本的には、ディスプレイのピンを基板に直接はんだ付けするか、ピンソケットをまず基板の端子部分に取り付け、そのピンソケットにディスプレイのピンを接続するかの方法を取ります。

ディスプレイのピンを直接基板にはんだ付けしても、ピンソケットをはんだ付けしても、基板が少しかさばる様になります。それが望ましくない場合には、必ずしも安定した接続法ではありませんが、次の節に示すようにディスプレイのピンを少し加工することにより、はんだ付けなしに基板に実用的に接続できるようになるので、必要に応じて試してみてください。

## 3.4　ディスプレイの簡易取り付け

### 3.4.1　概要と注意点

OLED ディスプレイや TFT ディスプレイのピンを少し加工することにより、多くの場合はんだ付けせずに ESP32-SLIM-R1 の CN2, CN3 端子にディスプレイを取り付けて利用できるようになります。

以下に示す方法は、少し変則的な方法になりますが、はんだ付けもピンソケットも必要のない便利な方法ですので、必要に応じてご活用ください。この方法では、端子をはんだ付けしていないので不必要な時にはディスプレイを取り外せますし、取り外したときには基板にピンソケットなどが残ってかさばることもありません。

ただし、基板のスルーホールはこのような使い方を想定して作られてはいないため、安定して使用できないことも考えられますし、さらには、ピンの挿抜を繰り返すと、接触不良やスルーホールの銅箔が剥離する可能性もあることをご承知[*1]の上ご活用ください。

### 3.4.2　ピンの加工と基板への取り付け

具体的には、ディスプレイのピンの先端部分を図 3.1 に示すように、ピンの並びの中心線から交互に上下に少しずつずらすように、ピンを根元から少し曲げます。

[*1] 弊社はこの方法でディスプレイを接続し利用できることや、この方法で端子等に何らかの問題が発生しないことを推奨・保証するものではありません

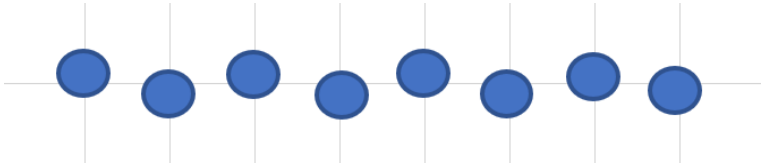


図 3.1　ディスプレイのピンの加工

上下への変位量が大きすぎると、CN2, CN3 にピンを差し込むことが非常に難しくなりますし、変位量が小さいと、ピンがばねの効果でスルーホールの壁面に押し付けられて接触する力が弱くなり、接触不良となる可能性が高くなるので少し慣れや調整が必要です。

ピンを加工したディスプレイを基板の端子に取り付ける際には、図 3.2 に示すように、ピンの先が基板の端子の裏側に少し抜けた程度の位置で止めてください。ディスプレイのピンの根元まで基板の端子差し込むと、ピンの根元でばピンを曲げて得られたばねの効果がないので、接触不良となる可能性が高くなります。

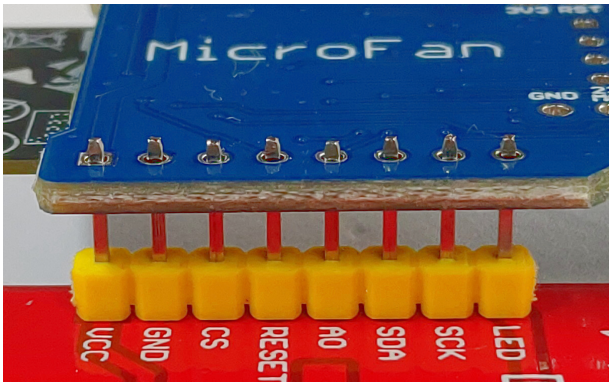



図 3.2　ディスプレイのピンの挿入

前章の図 2.3 は、この様にして接続した TFT ディスプレイに表示を行うスケッチを動かした例です。

ここで説明した例は TFT ディスプレイですが、OLED ディスプレイも同様の手順で取り付けることができます。

## 3.5　端子等のはんだ付け

### 3.5.1　半田ごての状態の管理

開発基板にピンヘッダをはんだ付けされる方のために、まず最初に、はんだ付けを行う際の一般的な半田ごての状態の管理に関して示します。

はんだ付けを行う直前に、スポンジなどのこて先クリーナーで半田ごてをクリーニングしてフラックスや酸化膜などの汚れを取り除き、こて先が銀色に輝く状態ではんだ付けを行います。また、こて先にほとんどはんだが乗っておらず乾いていると、こて先から部品のピンや基板のパター

ンなどに熱が伝わりにくいので、こて先に少し（薄く）はんだを付けてこて先がはんだで濡れた状態にしてはんだ付けを行います。

### 3.5.2　実装時のヒント

ピンソケット、ピンヘッダーなどの複数の端子を持つ部品のはんだ付けは、端子の端の 1 ピン、もしくは両端か対角上の 2 ピンをはんだ付けし、部品の取り付け姿勢などを必要に応じて修正してから残りの端子をはんだ付けすると、部品の姿勢をきれいに整えて取り付けることができます。

ESP32-SLIM-R1 のプリント基板はべたアースになっており、熱容量が大きくなっております。このため、各部品の GND 端子をはんだ付けする際には基板の端子部分（ランド）の温度が上がりはんだが融けるまで少し時間がかかるため、他の端子と比較して長めにはんだごてを当てておく必要がありますのでご注意ください。

### 3.5.3　ピンヘッダー SV1, SV2

ピンヘッダーはブレッドボードに挿せるように、SV1, SV2 の基板の裏側に取り付けます。ピンヘッダーは基板裏面からピンの短いほうをプリント基板に取り付け、プリント基板の表面（おもてめん）ではんだ付けします。

ピンヘッダーを開発ボードにはんだ付けする際に、はんだごてが基板上の部品に接触しないようにご注意ください。

### 3.5.4　ディスプレイ CN2, CN3

ディスプレイをピンソケットを介して ESP32-SLIM-R1 に接続する場合には、CN2 もしくは CN3 に 4 ピンもしくは 8 ピンのピンソケットを取り付けます。

ディスプレイを ESP32-SLIM-R1 に直接取り付ける場合には、OLED ディスプレイの場合には、裏側のコンデンサや抵抗の端子が基板上の部品に接触しないように、必要であれば少し浮かせた状態ではんだ付けするようご注意ください。

# 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`

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　ESP32 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-SLIM-R1 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 [ESP32 Dev Module] を選択してください。

また、[ポート:] は、 ESP32-SLIM-R1 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。

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、16M のフラッシュサイズに適合させます。

- Flash Size
  16MB を選択
- Partition Scheme
  ２種類ある　[16M Flash (—-)]　のどちらかを選択

## 4.3　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-SLIM-R1 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-SLIM-R1 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。

### 4.3.1　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-SLIM-R1 の LED は 2 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 2 に変更します。

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 2 as an output.
  pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(2, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(2, 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

### 4.3.2　スケッチのコンパイルと ESP32-SLIM-R1 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。

この時、ESP32-SLIM-R1 の LED2:RX が赤色で点滅し、スケッチの書き込みのための受信が行われていることが示されます。

スケッチが ESP32-SLIM-R1 に正しく書き込まれたら、ボード上の LED が点滅します。

LED2:RX が点滅し、スケッチが更新されたことは確認できますが、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　OLED ディスプレイの利用

### 4.4.1　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 4.2 のように表示されます。

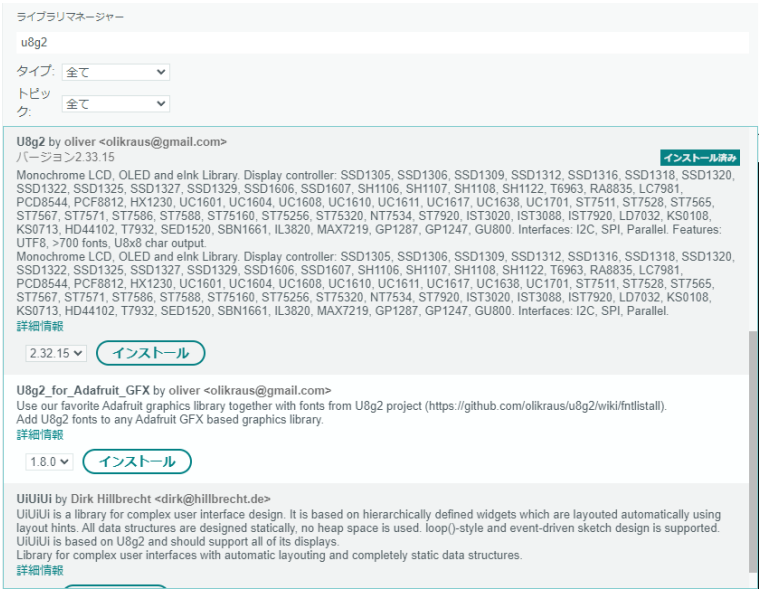



図 4.2　ライブラリマネージャを利用した U8g2 ライブラリの導入

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`

### 4.4.2　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 4.3　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。

## 4.5　TFT ディスプレイの利用

### 4.5.1　Adafruit-ST7735 ライブラリのインストール

TFT ディスプレイを利用するためのライブラリとして、Adafruit-ST7735 ライブラリを使用する例を示します。Adafruit-ST7735 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 [ST7735] を入力して絞り込むと、図 4.4 のように表示されます。

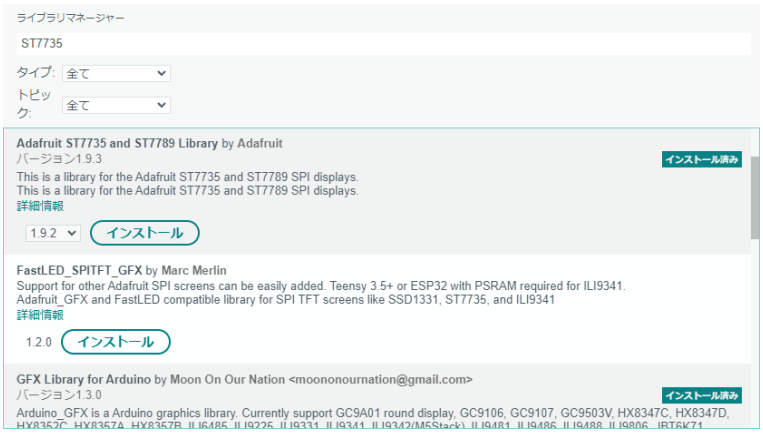



図 4.4　ライブラリマネージャを利用した Adafruit-ST7735 ライブラリの導入

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/adafruit/Adafruit-ST7735-Library`

### 4.5.2　Adafruit-ST7735 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 [Adafruit ST7735 and ST7789 Library] フォルダが追加されているの

が確認できます。この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 [Adafruit ST7735 and ST7789 Library] ⇒ [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、ESP32-SLIM-R1 に合わせた端子の設定や、使用するディスプレイの設定、向きなどを変更する必要があります。

まず、スケッチの 53 行あたりの端子指定の内容を以下のように書き換えます。

```
// もともとの指定内容
  #define TFT_CS         10
  #define TFT_RST         9 // Or set to -1 and connect to Arduino RESET pin
  #define TFT_DC          8

// ESP32-SLIM-R1 用の指定内容
  #define TFT_CS          5
  #define TFT_RST         -1
  #define TFT_DC          17
```

図 4.5　端子指定の変更

次に、ここで利用する TFT ディスプレイは、コントローラとして ST7735 の GREENTAB 仕様のものを使用し、サイズが 1.44 インチなので、サンプルスケッチの 89 行あたりの初期設定のメソッド呼び出しを、少し下の行（95 行）あたりでコメントアウトされているものに変更します。

```
// もともとの指定内容
  tft.initR(INITR_BLACKTAB);      // Init ST7735S chip, black tab

// ESP32-SLIM-R1 用の指定内容
  tft.initR(INITR_144GREENTAB); // Init ST7735R chip, green tab
  tft.setRotation(2) ; // ディスプレイの表示向きを変更
```

図 4.6　端子指定の変更

さらに、setRotation() のメソッド呼び出しを追加しディスプレイの表示の向きを変更します。引数は、0-3 を指定できます。ご自身が TFT ディスプレイを使用する状況に合わせて、表示の向きを調整してください。

これらの設定で、TFT ディスプレイを使用できるようになります。

# 第 5 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。

## 5.1 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。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。
- `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。

## 5.2　Thonny: IDE のインストール

ESP32-SLIM-R1 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。

### 5.2.1　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-SLIM-R1 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 ESP-WROOM-32 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 ESP-WROOM-32 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、ESP-WROOM-32 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。

### 5.2.2　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 5.1 のような内容が表示されます。

- `https://thonny.org/`

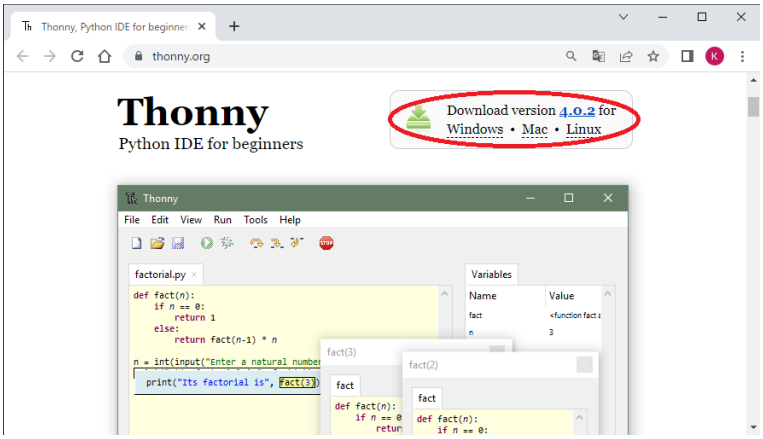



図 5.1　Thonny の TOP ページ

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 5.2 の様なダウンロード用のパネルがポップアップします。

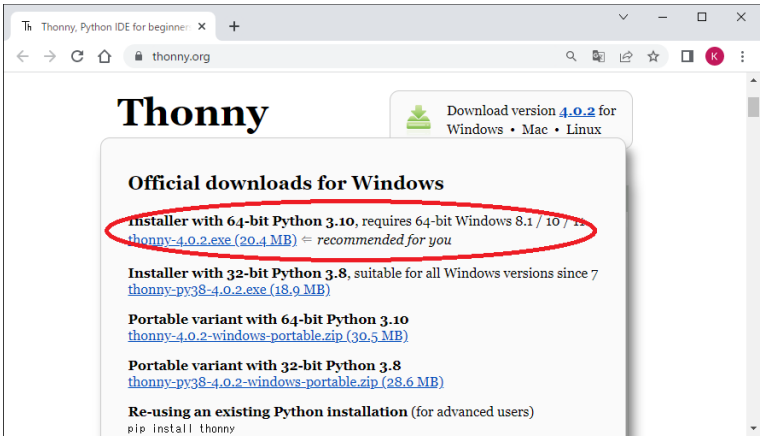



図 5.2　Thonny のダウンロードリンク

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。

### 5.2.3　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 5.3 に示します。

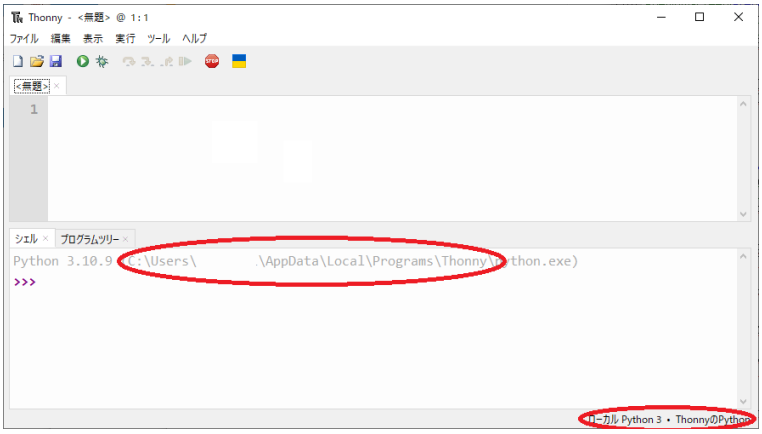



図 5.3　Thonny の起動画面

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 5.3 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 5.4 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 5.4　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。

## 5.3　MicroPython ファームウエアの書き込み

ESP32-SLIM-R1 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-SLIM-R1 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。

### 5.3.1　MicroPython のファームウェアのダウンロード

ESP32-SLIM-R1 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。

- `https://micropython.org/download/esp32/`

ダウンロードページを図 5.5 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。

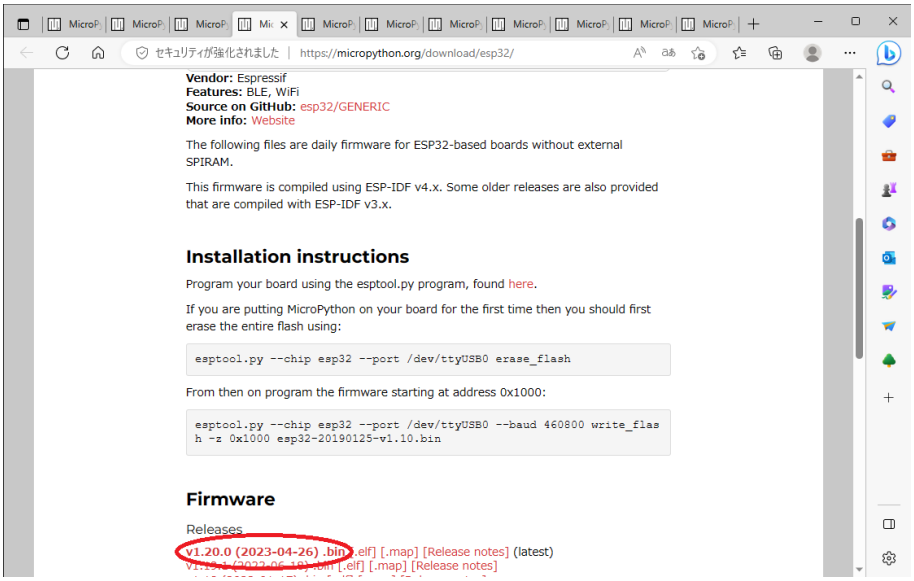



図 5.5　MicroPython のファームウェアのダウンロードページ

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。

- `esp32-20230426-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。

### 5.3.2　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 5.6 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 5.6　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 5.7 のようになります。

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

図 5.7　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-SLIM-R1 の USB 接続

ここで、ESP32-SLIM-R1 を USB で PC に接続してください。もし、ESP32-SLIM-R1 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 5.8 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-SLIM-

R1 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 COM3
Connecting.....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting......
Detecting chip type... ESP32
Chip is ESP32-D0WD-V3 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4018
Detected flash size: 16MB
Hard resetting via RTS pin...
```

図 5.8　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 5.8 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 5.9 に示すように、erase_flash コマンドで ESP32-SLIM-R1 の ESP-WROOM-32 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 5.9　フラッシュメモリの消去

次に、図 5.10 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 0x1000 [PATH]esp32-20230426-v1.20.0.bin
```

図 5.10　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。

### 5.3.3　Thonny の Python を利用した書き込み

MicroPython のファームウェアの ESP32-SLIM-R1 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 5.11 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。

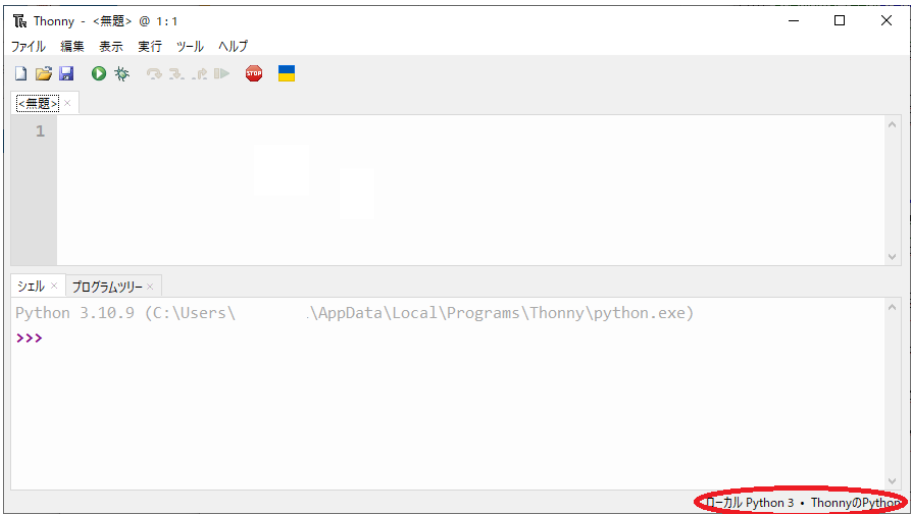



図 5.11　Thonny の起動

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 5.12 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 5.12　esptool のインストール

esptool がインストールできたら、図 5.13 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 5.13　esptool の動作確認

### ESP32-SLIM-R1 の USB 接続

ここで、ESP32-SLIM-R1 を USB で PC に接続してください。もし、ESP32-SLIM-R1 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 5.14 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-SLIM-R1 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 COM3
Connecting.....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting....
Detecting chip type... ESP32
Chip is ESP32-D0WD-V3 (revision v3.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4018
Detected flash size: 16MB
Hard resetting via RTS pin...
```

図 5.14　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 5.15 に示すように erase_id コマンドで ESP32-SLIM-R1 の ESP-WROOM-32 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 5.15　フラッシュメモリの消去

次に、図 5.16 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 0x1000 [PATH]esp32-20230426-v1.20.0.bin
```

図 5.16　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。

### 5.3.4　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 5.17 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 5.17　esptool のインストール

esptool のインストール後図 5.18 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 5.18　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-SLIM-R1 の USB 接続

ここで、ESP32-SLIM-R1 を USB で PC に接続してください。もし、ESP32-SLIM-R1 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 5.19 に示すように flash_id を引数に指定して esptool を実行すると、ESP32-SLIM-R1 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 COM3
Connecting......
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting.......
Detecting chip type... ESP32
Chip is ESP32-D0WD-V3 (revision v3.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: XX:XX:XX:XX:XX:XX
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4018
Detected flash size: 16MB
Hard resetting via RTS pin...
```

図 5.19　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 5.20 に示すように erase_id コマンドで ESP32-SLIM-R1 の ESP-WROOM-32 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 5.20　フラッシュメモリの消去

次に、図 5.21 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 0x1000 [PATH]esp32-20230426-v1.20.0.bin
```

図 5.21　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。

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

### 5.4.1　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 5.22 のようなウインドウが開きます。

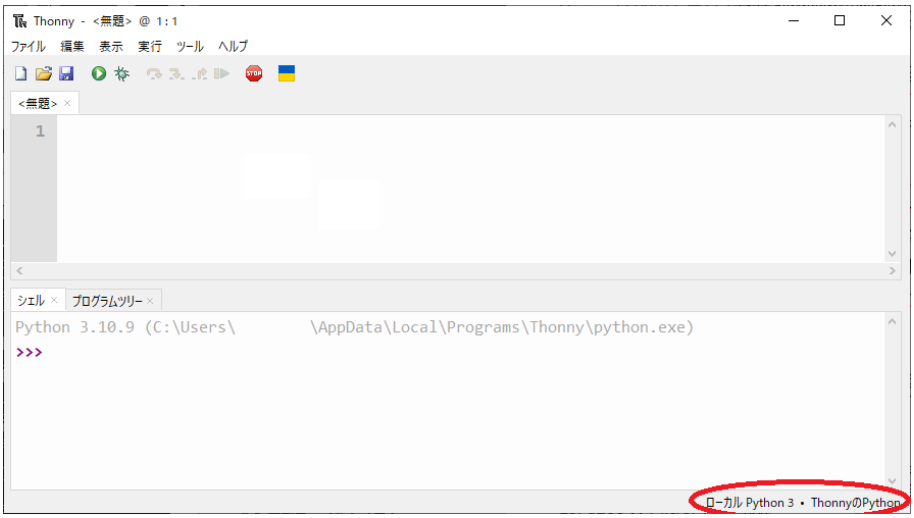



図 5.22　Thonny の起動画面

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-SLIM-R1 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-SLIM-R1 の MicroPython が起動される設定となっていた場合に、ESP32-SLIM-R1 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 5.23 の様な表示となります。

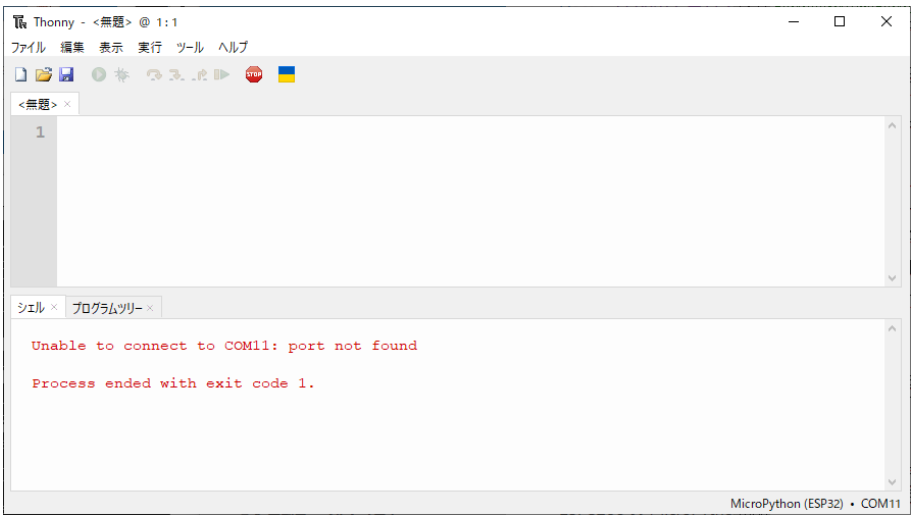



図 5.23　MicroPython 起動の失敗

### 5.4.2　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-SLIM-R1 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。

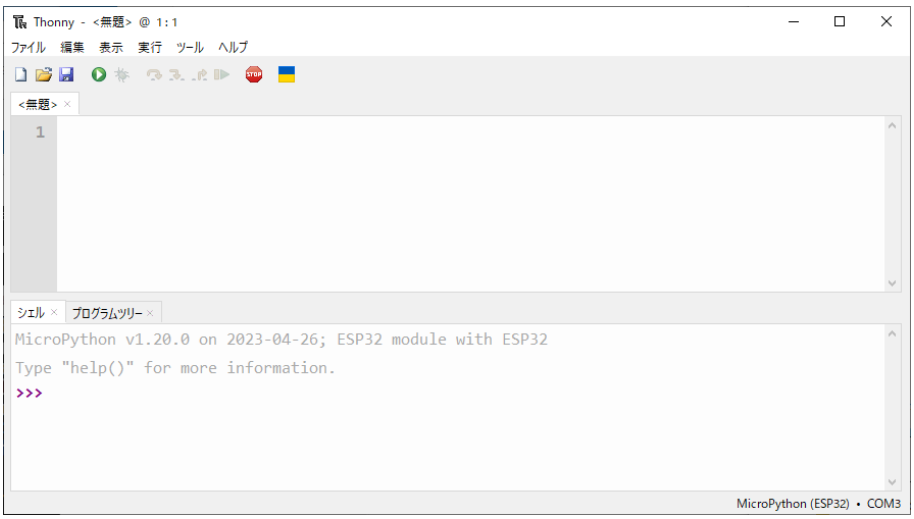



図 5.24　Thonny で MicroPython の起動

この操作により、図 5.24 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 ESP32 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。

### 5.4.3　プログラムの対話的な実行

ESP32-SLIM-R1 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 5.25　演算結果の出力

図 5.25 に示すように、入力した処理が ESP32-SLIM-R1 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。

### 5.4.4　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 5.26 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。


図 5.26　プログラムの編集

プログラムの入力が完了したら、図 5.27 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。

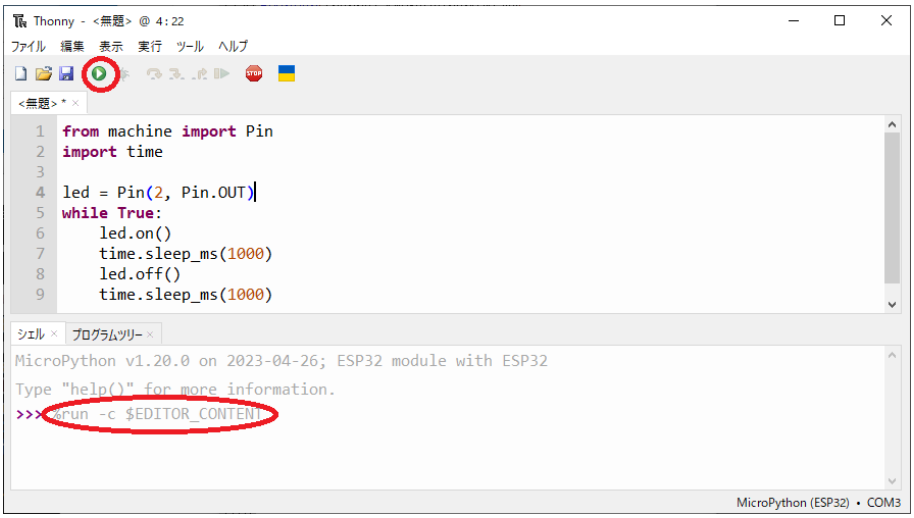



図 5.27　プログラムの実行

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-SLIM-R1 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 5.28 に示します。


図 5.28　プログラムの中断

### 5.4.5　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-SLIM-R1 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-SLIM-R1 上のファイルシステムに保存した場合には、その ESP32-SLIM-R1 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 5.29 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。

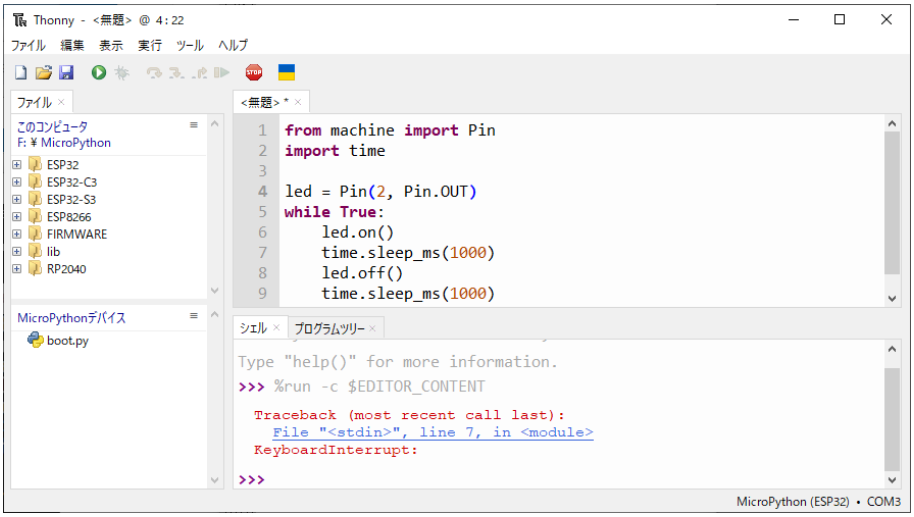



図 5.29　ファイルペインの表示

ファイルペインの上側は、PC 上のファイル、下側は ESP32-SLIM-R1 上のファイルを示しています。この例では、 ESP32-SLIM-R1 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 5.30 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-SLIM-R1 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。

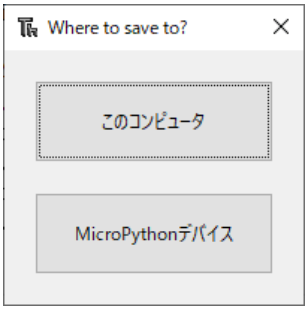



図 5.30　ファイルの保存先の選択

すると、図 5.31 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。

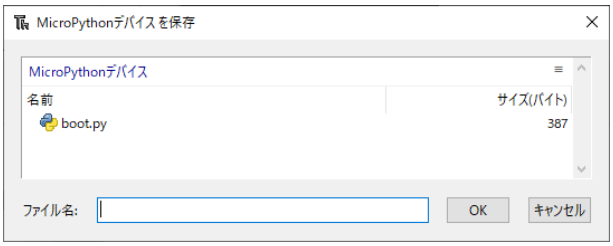



図 5.31　ファイルの名の入力

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 5.32 に示します。

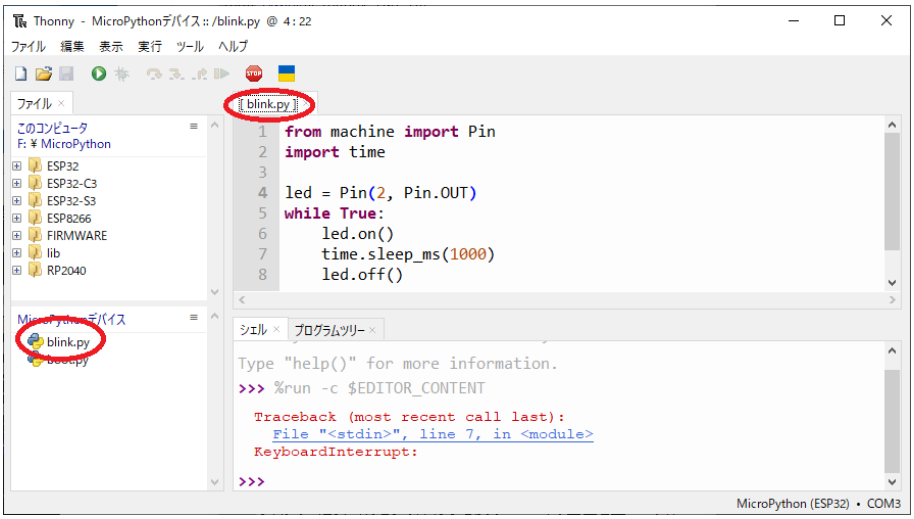



図 5.32　ファイルの保存後

図 5.32 では、エディタ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 5.33 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。

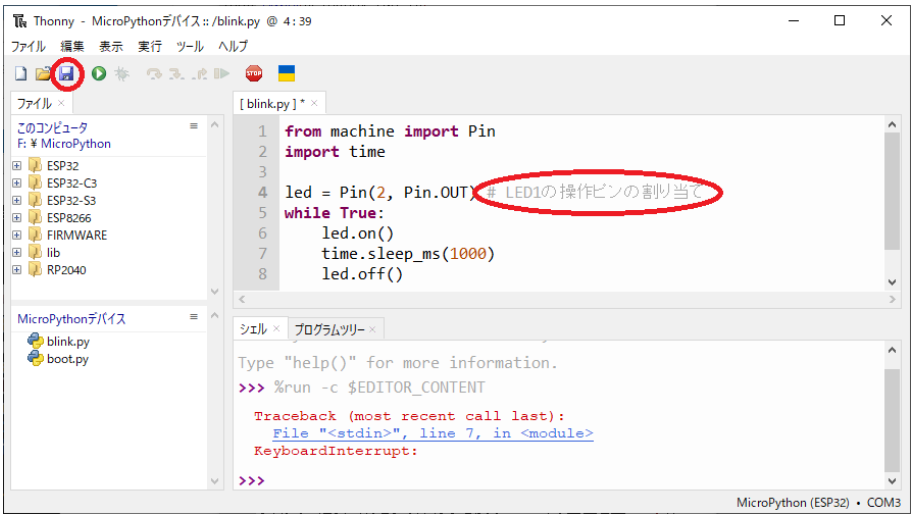



図 5.33　ファイルの変更

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 5.34 のような画面になります。

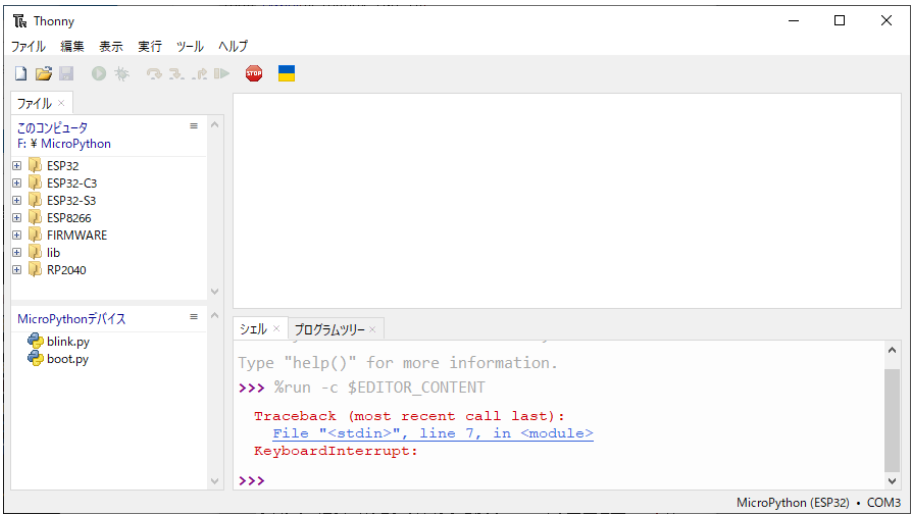



図 5.34　編集の終了

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 5.35 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。

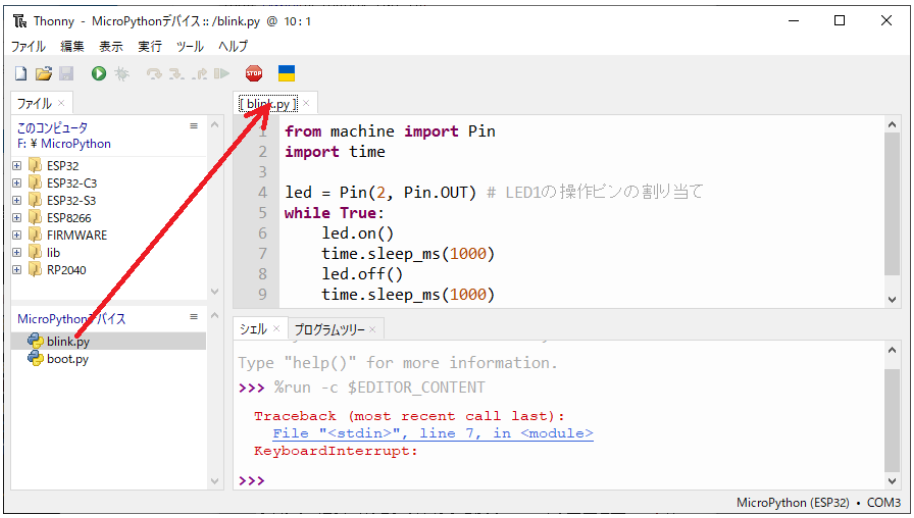



図 5.35　プログラムの再編集

### 5.4.6　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 ESP-WROOM-32 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-SLIM-R1 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 ESP-WROOM-32 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 5.36 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。

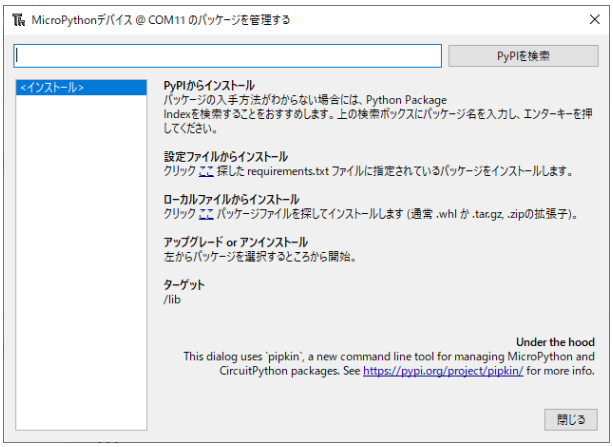



図 5.36　パッケージ管理ダイアログ

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。

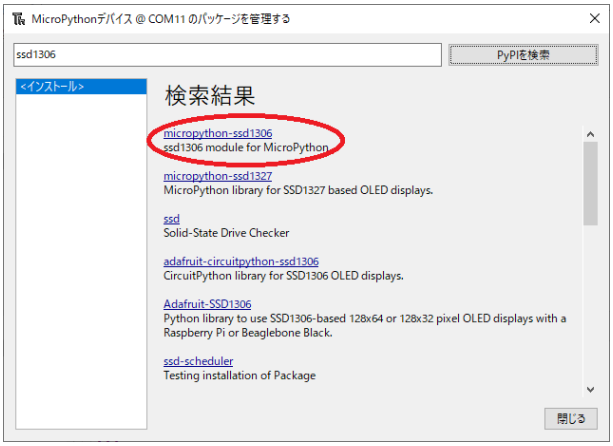



図 5.37　ssd1306 の検索

ダイアログの内容が図 5.38 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 5.38　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 5.39 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。

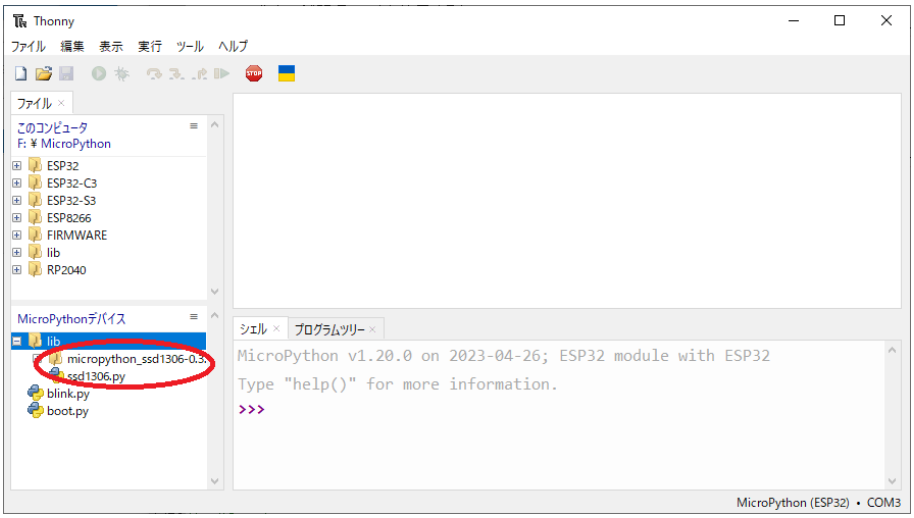



図 5.39　インストールされたライブラリ

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 5.40 に示します。

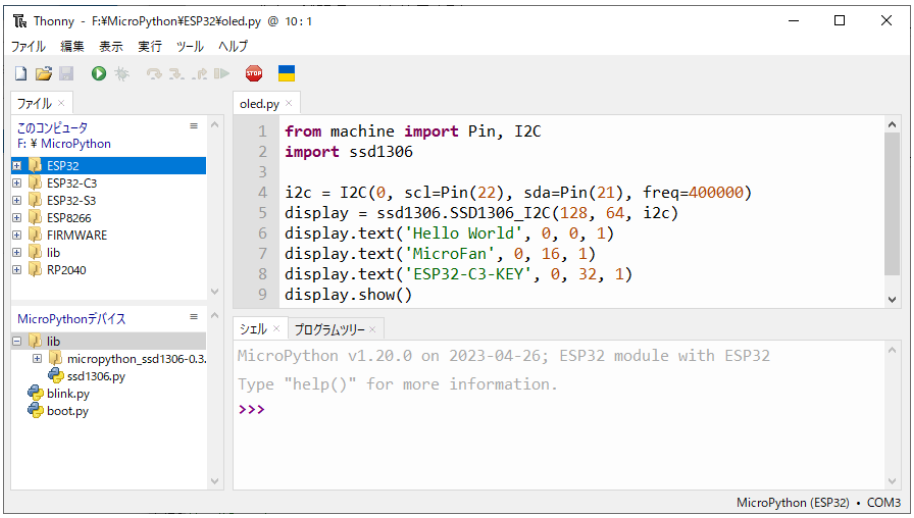



図 5.40　ssd1306 ライブラリの利用

このプログラムを実行すると、ESP32-SLIM-R1 に接続している OLED ディスプレイに ‘Hello World’ と表示されます。

## 5.5　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 ESP-WROOM-32 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。

### 5.5.1　boot.py

ESP32-SLIM-R1 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
from neopixel import NeoPixel

i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000) # i2c 回線の設定

led = Pin(2, Pin.OUT) # D2, LED1
```

図 5.41　boot.py の記述例

### 5.5.2　main.py

ESP32-SLIM-R1 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 5.42 の例では、先の boot.py の初期化処理と組み合わせて ESP32-SLIM-R1 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。

なお、PC と接続せずに ESP32-SLIM-R1 を単体で使用する場合には、main.py の内容は、図 5.43 の LED の点滅プログラムの様に、実行が終了しない無限ループのプログラムとなるのが一般的です。

```
led.on()
```

図 5.42　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-SLIM-R1 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-SLIM-R1 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・実行できることになります。

## 5.6　MicroPython のプログラム例

以下に、ESP32-SLIM-R1 上での MicroPython のプログラム例を示します。

ESP-WROOM-32 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。

- 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。

### 5.6.1　LED の点滅

電子工作で定番の LED の点滅プログラムです。ESP32-SLIM-R1 に搭載されている LED1 を点滅させるプログラムです。

```
from machine import Pin
import time

led = Pin(2, Pin.OUT) # D2 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 5.43　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。

### 5.6.2　OLED ディスプレイへの出力

ESP32-SLIM-R1 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
import ssd1306

# OLED の設定
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000) # i2c 回線の設定
oled = ssd1306.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-SLIM-R1", 0, 32, 1)
oled.show() # 表示内容の OLED への反映
```

図 5.44　WS2812 の点灯

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

### 5.6.3　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())
101840
>>>
```

図 5.45　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 6 章

# 資料

## 6.1　ESP32-SLIM-R1 の回路図

ESP32-SLIM-R1 の回路図を図 6.1、部品表を表 6.1 に示します。

内部のフラッシュメモリに接続されている ESP-WROOM-32 の 17-22 ピンは、取扱に注意を要するため未接続となっています。

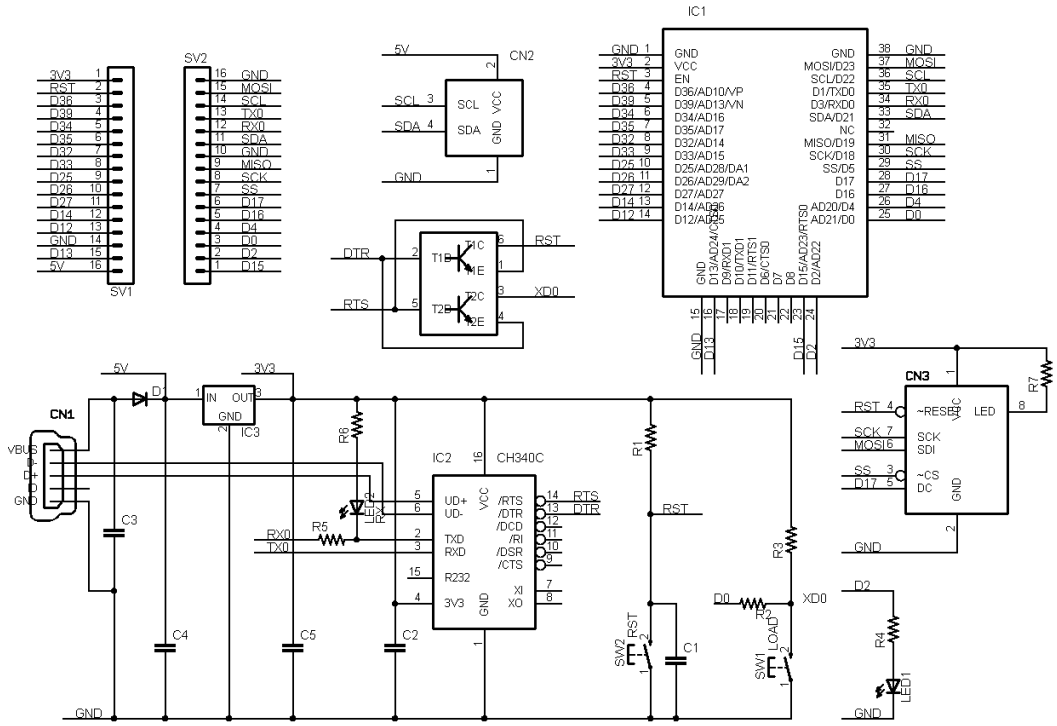



図 6.1　ESP32-SLIM-R1 の回路図

表 6.1　部品表

| 部品 | シンボル | 規格等 | 1 |
|---|---|---|---|
| プリント基板 | ESP32-SLIM | Rev.1 | 1 |
| IC | IC1 | ESP-WROOM-32(E, 16M) | 1 |
| | IC2 | CH340C | 1 |
| | IC3 | BL8071CLATR33 | 1 |
| トランジスタ | Q1 | UMH3N | 1 |
| ショットキーダイオード | D1 | SS14 | 1 |
| 発光ダイオード | LED1 | 青 | 1 |
| | LED2 | 赤 | 1 |
| 抵抗 | R1, R3 | 10K Ω | 2 |
| | R2, R6 | 1K Ω | 2 |
| | R4, R5 | 470 Ω | 2 |
| | R7 | 4.7 Ω | 1 |
| セラミックコンデンサ | C1, C2, C3 | 0.1 μ F | 3 |
| | C4 | 10 μ F | 1 |
| | C5 | 47 μ F | 1 |
| タクトスイッチ | SW1, SW2 | 2 端子 | 2 |
| USB | CN1 | microB | 1 |
| OLED ディスプレイ | CN2 | 4 ピン | 別売り |
| TFT ディスプレイ | CN3 | 8 ピン | 別売り |
| ピンヘッダ | SV1, SV2 | 16PIN X2 | 別売り |

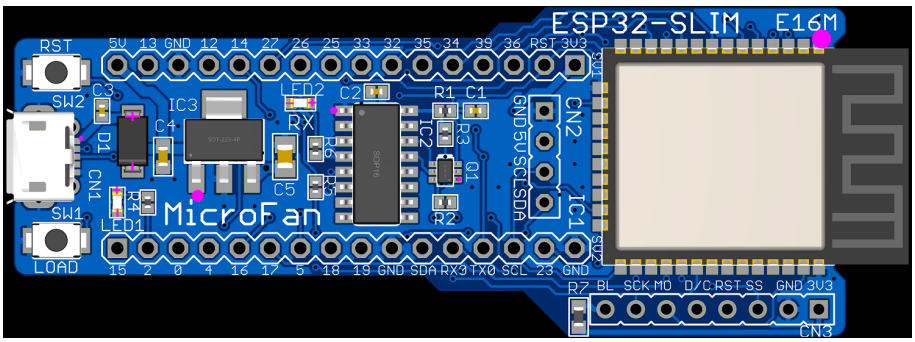



図 6.2　ESP32-SLIM-R1 の部品配置

## 6.2　基板上の入出力

ESP32-SLIM-R1 の基板上のスイッチと LED を表 6.2 に示します。

SW1 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、スケッチ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とができます。

表 6.2　スイッチと LED

| シンボル | 信号線 | 備考 |
|---|---|---|
| SW1 | D0 | BOOT ローダーモード移行用 |
| SW2 | EN | リセット用 |
| LED1 | D2 | 正論理 |

## 6.3　ブレッドボード用端子

ESP32-SLIM-R1 には、ブレッドボードに挿して利用するためのピンヘッダー用端子 SV1, SV2 が用意されています。SV1,SV2 のピン配置を表 6.3 に示します。

表 6.3　SV1,SV2 ピン配置

| 備考 | SV1 信号線 | ピン番号 | SV2 信号線 | 備考 |
|---:|---:|:---:|---|---|
| | 3.3V | 16 | GND | |
| EN | RST | 15 | D23 | MOSI |
| VP/A0 | D36 | 14 | SCL | D22 |
| VN/A3 | D39 | 13 | TX0 | D1 |
| A6 | D34 | 12 | RX0 | D3 |
| A7 | D35 | 11 | SDA | D21 |
| A4 | D32 | 10 | GND | |
| A5 | D33 | 9 | D19 | MISO |
| DA1/A18 | D25 | 8 | D18 | SCK |
| DA2/A19 | D26 | 7 | D5 | SS |
| A17 | D27 | 6 | D17 | |
| A16 | D14 | 5 | D16 | |
| A15 | D12 | 4 | D4 | A10 |
| | GND | 3 | D0 | A11 |
| A14 | D13 | 2 | D2 | A12 |
| | 5V | 1 | D15 | A13 |

表 6.3 に示されている I2C や SPI のピン配置は、Arduino でのデフォルト設定に準拠しています。

- `https://github.com/espressif/arduino-esp32/blob/master/variants/esp32/`

pins_arduino.h

## 6.4　ディスプレイ搭載用端子

### 6.4.1　OLED ディスプレイ

基板上に OLED ディスプレイを搭載するための CN2 端子を備えています。CN2 のピン配置を表 6.4 に、推奨する OLED ディスプレイを図 6.3 に示します。また、ネットショップ URL を以下に示します。

- https://store.shopping.yahoo.co.jp/microfan/oled096-128x64-i2c-blue.html
- https://www.amazon.co.jp/dp/B06Y4TKL1F

表 6.4　CN2(OLED ディスプレイ) ピン配置

| ピン番号 | 信号線 | 備考 |
|---|---|---|
| 1 | GND | |
| 2 | 5V | |
| 3 | D22 | SCL |
| 4 | D21 | SDA |

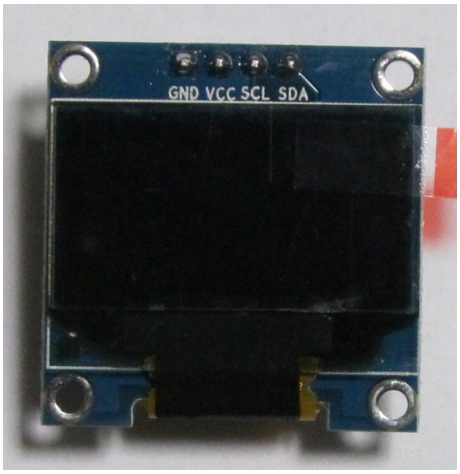



図 6.3　OLED ディスプレイ

OLED ディスプレイに要求される機能を以下に示します。

- モジュールを直接コネクタに刺すためには、信号線の並びが表 6.4 の順になっていること。
- SCL, SDA の信号線が 3.3V 対応であること。

- ESP32-SLIM-R1 には I2C 用のプルアップ抵抗が組み込まれていないため、SCL, SDA の信号線にプルアップ抵抗が付与されていること。
- ESP32-SLIM-R1 からの電源として 5V を供給しているため、3.3V の電圧レギュレータが内蔵されていること。
- 使用するライブラリにもよりますが、コントローラに SSD1306 か SH1106 を使用していること。

推奨する OLED ディスプレイの SDA, SCL 信号線には、4.7K-10K Ωのプルアップ抵抗が組み込まれています。このため、CN2 に OLED ディスプレイを接続している場合には、ブレッドボード上で I2C デバイスを使用する際に、SDA, SCL に別途プルアップ抵抗を接続する必要はありません。（付けた場合には、OLED ディスプレイのプルアップ抵抗との合成抵抗値となります。）

### 6.4.2　TFT ディスプレイ

基板上に TFT ディスプレイを搭載するための CN3 端子を備えています。CN3 のピン配置を表 6.5 に、推奨する TFT ディスプレイを図**??**に示します。また、ネットショップ URL を以下に示します。

- `http://store.shopping.yahoo.co.jp/microfan/tft144-128x128.html`
- `https://www.amazon.co.jp/dp/B0BN5XGRTW`

また、下記の 1.8 インチ 128x160 ピクセルの TFT ディスプレイもコントローラと端子の並びが共通しているので、スケッチでの初期化法を少し変更[*1]することにより利用可能です。

- `https://www.amazon.co.jp/dp/B0BN5XKFFL`

表 6.5　CN3(TFT ディスプレイ) ピン配置

| ピン番号 | 信号線 | 備考 |
|---|---|---|
| 1 | 3V3 | VCC |
| 2 | GND | |
| 3 | D3 | CS |
| 4 | RST | RESET |
| 5 | D17 | DC |
| 6 | D23 | MOSI |
| 7 | D18 | SCK |
| 8 | 3V3 | LED |

[*1] 実害はないのですが、画面の端に少しノイズが出ます。これを除去するためには、ライブラリのソースコードを少し変更する必要があります。

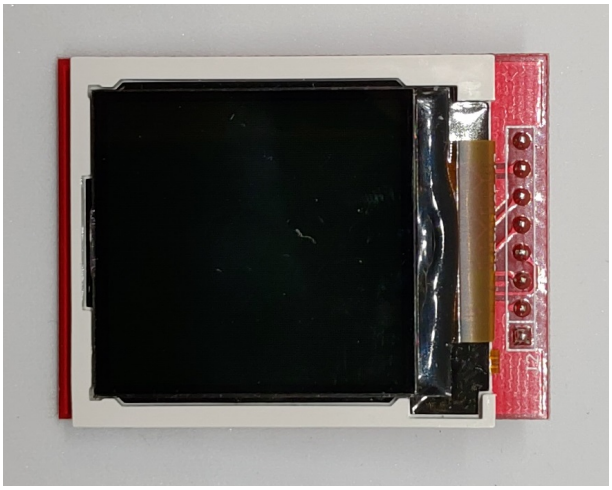


図 6.4　TFT ディスプレイ

TFT ディスプレイに要求される機能を以下に示します。

- モジュールを直接コネクタに刺すためには、信号線の並びが表 6.5 の順になっていること。
- 全ての信号線が 3.3V 対応であること。
- ESP32-SLIM-R1 からの電源として 3.3V を供給しているため、モジュールに内蔵の 3.3V の電圧レギュレータを無効にできること。（保証はできませんが、多くの場合、無効にしなくても問題なく動くようです）
- 使用するライブラリにもよりますが、コントローラに ST7735 を使用していること。

# 第 7 章

# 購入および問い合わせ先

## 7.1　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。

## 7.2　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

## 7.3　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。

## 7.4　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。

## 7.5　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