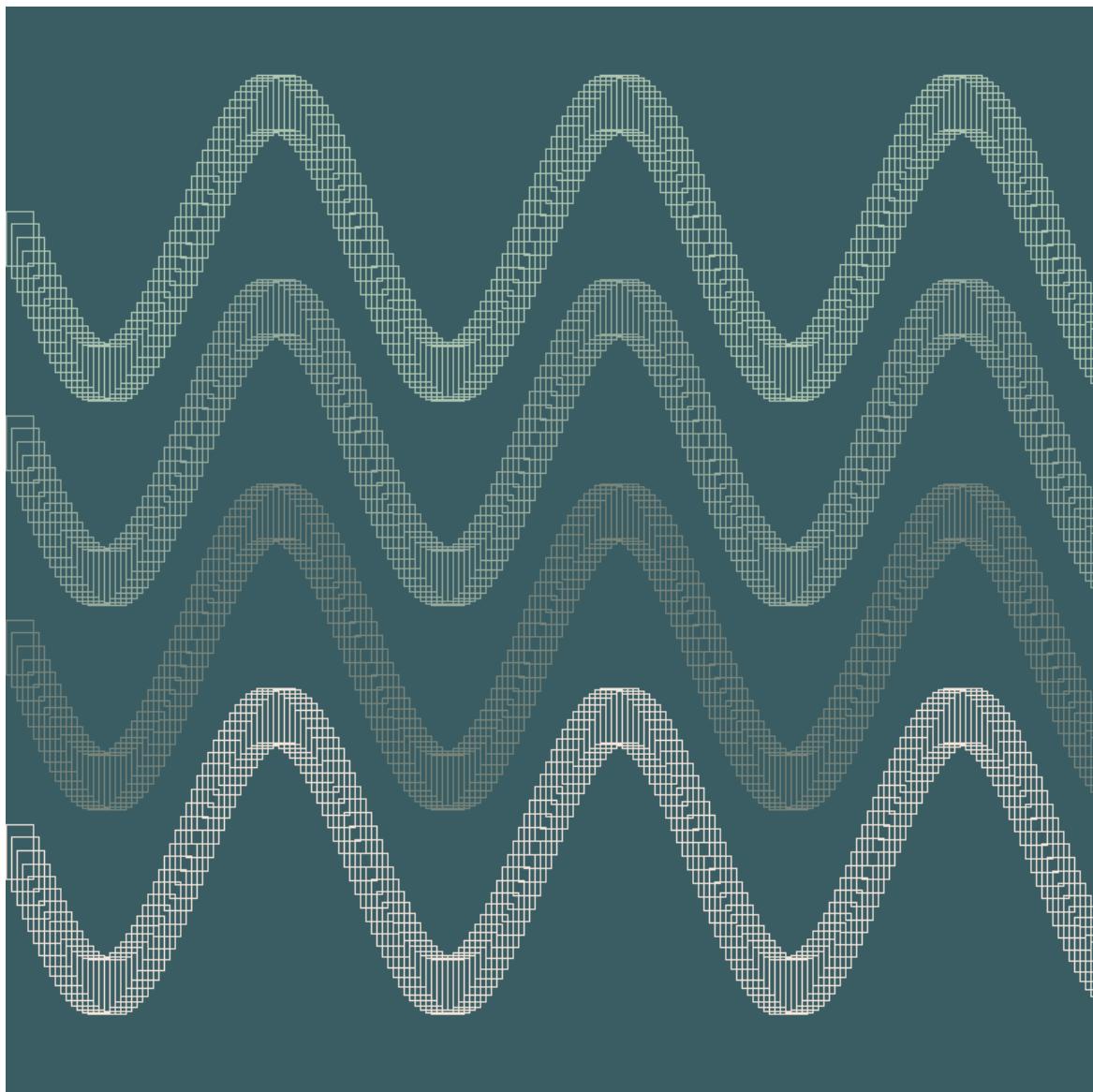


ESP32ではじめる初めてのIoT



電子計算機研究会

ESP32 ではじめる初めての IoT

THEToilet 著

2021-08-17 版 発行

はじめに

これは電子計算機研究会の IoT 講座用に作った技術同人誌です。

サークルに参加するメリットの一つに、興味があることについて学べる機会がある。これがあげられるとおもいます。自分も一年の時にサークルの先輩から、いろいろな勉強会を開催していただき。自分の知見をひろげることができました。本誌が少しでも役にたてば幸いです。

(@THEToilet)

電子計算機研究会とは

芝浦工業大学公認の技術サークルです^{*1}。主にゲームや Web アプリの制作活動やコンピューターサイエンスの勉強を行っています。

お問い合わせ先

本誌に関するお問い合わせ : toileito.wc.benki@gmail.com

想定読者

- IoT に興味はあるがなかなか手をだせない人
- 通信に興味がある人
- 電子計算機研究会に所属している人

^{*1} 電子計算機研究会 HP <http://den3.net>

目次

はじめに	ii
電子計算機研究会とは	ii
お問い合わせ先	ii
想定読者	ii
第 1 章 電子部品の準備	1
1.1 電子部品の購入の方法	1
1.2 本誌で利用する電子部品	2
第 2 章 環境構築	5
2.1 ESP32 とは	5
2.2 ESP32 の開発環境	6
2.3 Arduino IDE のインストール	6
2.4 ESP32 用ボードマネージャーのインストール	12
2.5 Hello ESP32!!	18
シリアル通信とは	29
第 3 章 電子部品を使ってみよう	32
3.1 部品説明	32
3.2 L チカ力しよう！	41
チャタリング	46
3.3 応用問題: 状態遷移	47
第 4 章 センサのデータを Web 上に公開しよう	51

目次

4.1	センサを使おう	51
4.2	Web に公開しよう	58
付録 A	トラブルシューティング	66
A.1	シリアルモニタで文字化けがする	66
A.2	プログラムが書き込めない	67
A.3	プログラムが反映されない	68
A.4	error: redefinition	68
A.5	接続ポートに ESP32 が反映されない	69
A.6	COM ポートがデバイスマネージャーに表示されない	72
A.7	うまく書き込めない	72
A.8	LED の光り方が弱い	72
A.9	回路図どうりなのにつかない	73
著者紹介		74

第1章

電子部品の準備

本章では本誌のサンプルを進めるにあたって必要な電子部品および、その購入方法について紹介します。

1.1 電子部品の購入の方法

電子部品の販売店が近くにあれば直接商品を見ながら購入するのが一番ですが、お店が近くになかったり、コロナ渦の問題などで直接行くことが難しい場合は、通販での購入をおすすめします。下記の5つは電子部品を通販で購入できるサイトです。特に秋月電子通商、千石電商そしてaitendoは秋葉原に店舗があるので、機会があれば行くことをおすすめします。

- 秋月電子通商
 - <https://akizukidenshi.com/catalog/>
- 千石電商
 - <https://www.sengoku.co.jp/>
- スイッチサイエンス
 - <https://www.switch-science.com/>
- Amazon
 - <https://www.amazon.co.jp/>
- aitendo
 - <https://www.aitendo.com/>

1.2 本誌で利用する電子部品

筆者が本誌に使用するサンプルを作成するにあたって購入した商品を紹介します（表1.1）。本誌のサンプルを進めるにあたって必要になるため、参考にしてください。

表 1.1: 必要な材料

品名	個数	参考価格	詳細情報
ESP32DevKitC	1 個	1230 円	
microUSB Type-B	1 本	約 300 円	
プレッドボード	2 個	280 円 × 2	
LED	1 袋	150 円	
ジャンプワイヤセット（オス・オス）	1 セット	220 円	
抵抗 100 & 10k	100 : 1 袋 10k : 1 袋	100 円 × 2	
タクトスイッチ	1 個	10 円	
温湿度センサ	1 個	300 円	
ディスプレイ	1 個	580 円	
計		約 3550 円	

おすすめ製品

今回筆者はすべて秋月の通販にて電子部品を購入をしましたが、同じ製品であればどの店舗で購入しても差し支えありません。しかし、本誌は以下の製品で動作確認をしているため基本的には以下の製品を購入することをおすすめします。

ESP32DevKitC

ESP32 - DevKitC - 32E ESP32 - WROOM - 32E 開発ボード
4 MB

<https://akizukidenshi.com/catalog/g/gM-15673/>

ブレッドボード

ブレッドボード 6穴版 E I C - 3 9 0 1

<https://akizukidenshi.com/catalog/g/gP-12366/>

備考: ESP32DevKitC は幅が広いため、6穴のブレッドボードを使うことをおすすめします。

LED

5mm赤色LED 625nm 7cd 60度 (10個入)

<https://akizukidenshi.com/catalog/g/gI-01318/>

ジャンプワイヤセット(オス・オス)

ブレッドボード・ジャンパワイヤ(オス-オス)セット 各種 合計60本以上

<https://akizukidenshi.com/catalog/g/gC-05159/>

抵抗

カーボン抵抗(炭素皮膜抵抗) 1 / 4W 10k (100本入)

<https://akizukidenshi.com/catalog/g/gR-25103/>

カーボン抵抗(炭素皮膜抵抗) 1 / 4W 100 (100本入)

<https://akizukidenshi.com/catalog/g/gR-25101/>

備考: 上記の抵抗は100本単位からしか購入できません。実際に使用するのはどちらの抵抗値とも3本以下なので必ずしも100本買う必要はありません。

タクトスイッチ

タクトスイッチ(緑色)

<https://akizukidenshi.com/catalog/g/gP-03651/>

備考: 色の選択は自由です。

温湿度センサ

温湿度センサ モジュール DHT11

<https://akizukidenshi.com/catalog/g/gM-07003/>

ディスプレイ

0.96インチ 128×64ドット有機ELディスプレイ（OLED）白色

<https://akizukidenshi.com/catalog/g/gP-12031/>

第 2 章

環境構築

この章では ESP32 にプログラムを書き込む際に必要な環境構築の手順を紹介します。本誌は、Windows 環境を想定しており Mac 環境の方は手順が異なる可能性があります。

2.1 ESP32 とは

ESP32 とは Espressif Systems 社が開発した SoC (System on a Chip) シリーズの名前です。ESP32 という名前の使われ方には様々あり今回使用する ESP32DevKitC-32E (図 2.1) は、ESP32 をユーザが利用しやすい形にした製品ですが、通称として ESP32 と呼ばれことがあります。そのため、本誌では ESP32DevKitC-32E も含めて ESP32 と呼んでいます。ESP32 の特徴としては Bluetooth や Wi-Fi モジュールがついている点やマルチコアな点が挙げられます。

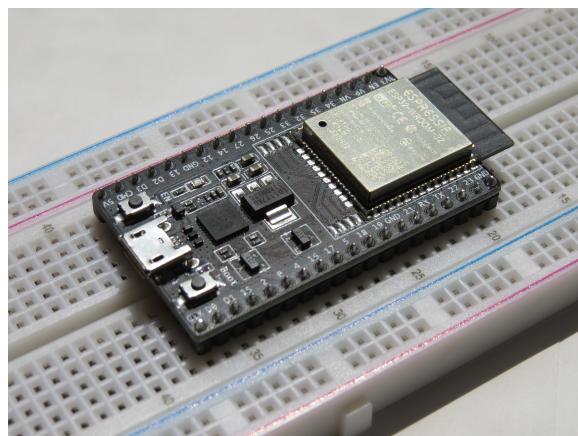


図 2.1: ESP32DevKitC-32E

2.2 ESP32 の開発環境

ESP32 の開発環境には主に以下の 3 つが挙げられます。

- Arduino IDE
 - Arduino 互換ボード用統合開発環境 (C/C++)
- ESP-IDF
 - ESP32 専用の開発環境 (C/C++)
- MicroPython
 - C 言語で作られた Python3 と互換性がある言語処理系

今回は利用者が多く、関連情報がネット上に多く見られる Arduino IDE を用いて開発を進めていきたいと思います。

2.3 Arduino IDE のインストール

Arduino IDE をインストールするために以下のリンクにアクセスしてください。

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

ダウンロード画面（図 2.2）ではご自身の PC 環境にあったダウンロードリンクを選択してください。ここからの手順では、Windows10 でのダウンロードを想定しています。

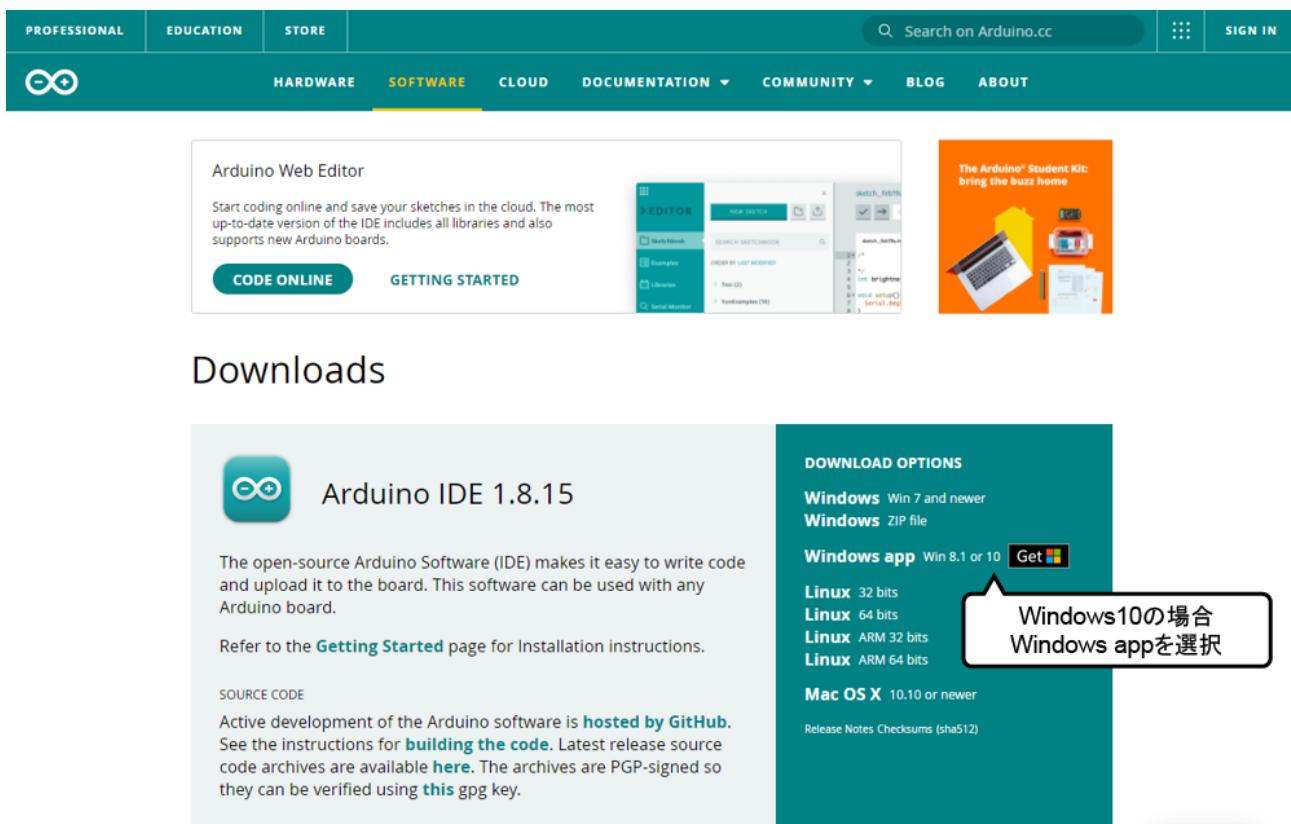


図 2.2: ArduinoIDE のダウンロード画面

ダウンロードリンクにアクセスすると、寄付金の金額選択画面に遷移します（図 2.3）。可能であれば寄付もできますが、JUST DOWNLOAD を選択することでつぎの画面に遷移します。

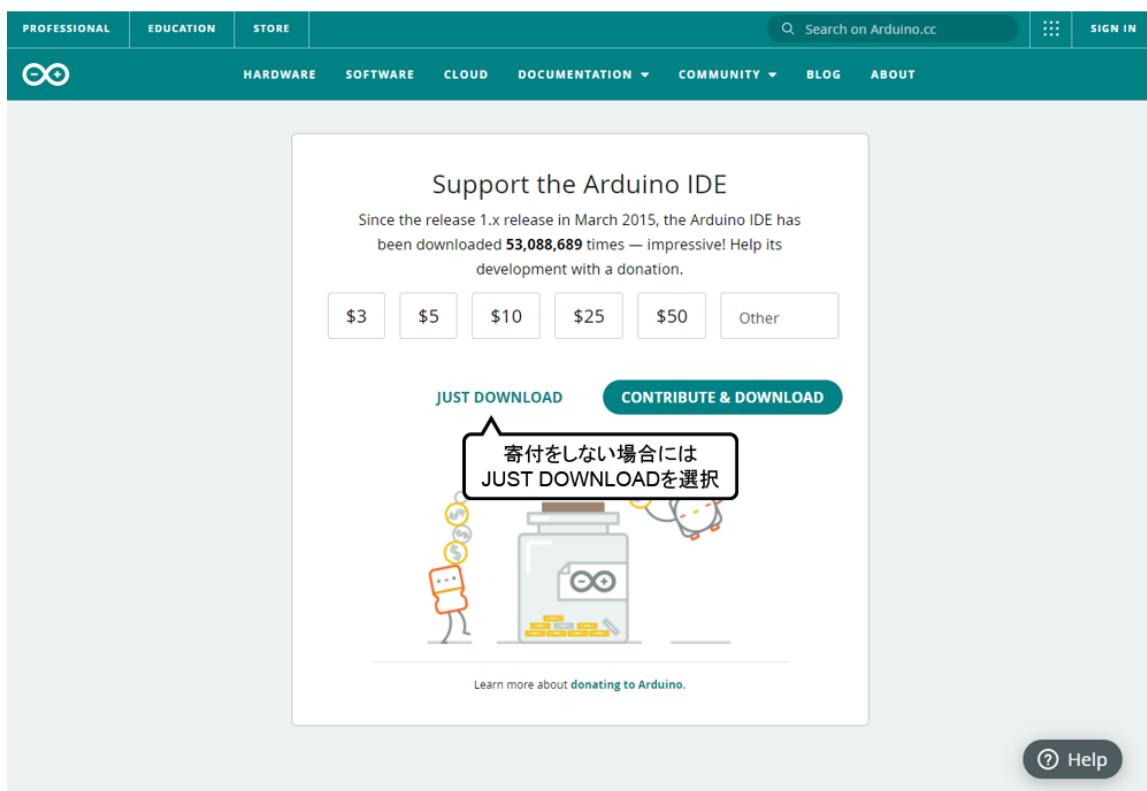


図 2.3: 寄付金の金額選択画面

JUST DOWNLOAD を選択するとブラウザ内で MicrosoftStore の画面に遷移します（図 2.4）。つぎに入手を選択すると、ブラウザのポップアップが表示され Windows 上で MicrosoftStore を開く許可を求められるので許可を選択してください。

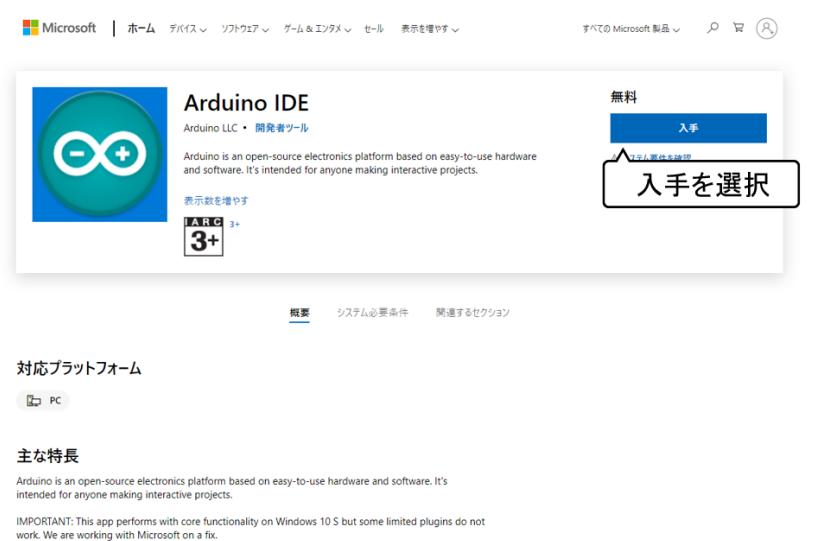


図 2.4: ブラウザで見る MicrosoftStore

Windows 上で開かれた MicrosoftStore です（図 2.5）。再度、入手を選択してください。

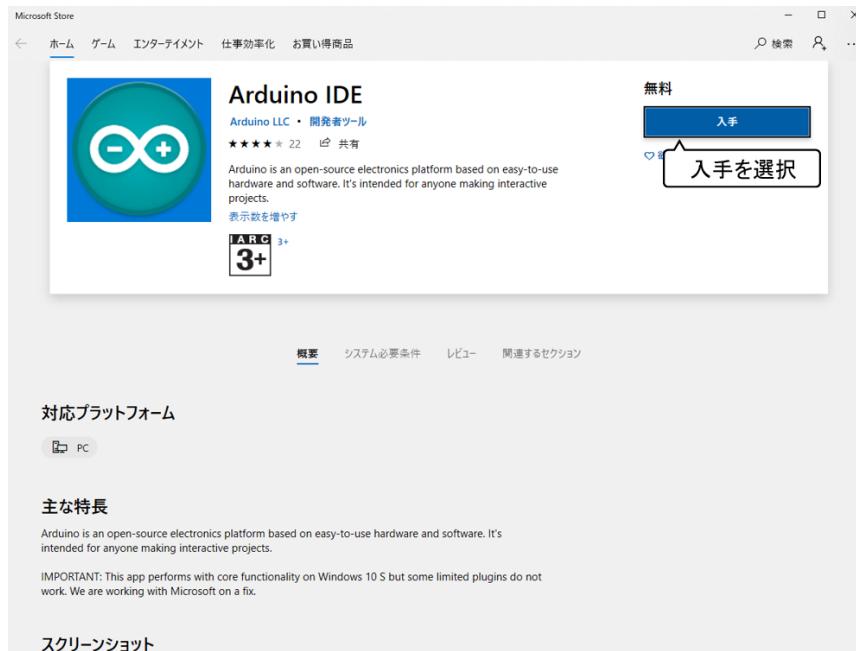


図 2.5: Windows で開いた MicrosoftStore

サインインについて尋ねられますが（図 2.6） 必要ありませんを選択した場合もダウンロードは開始されます。

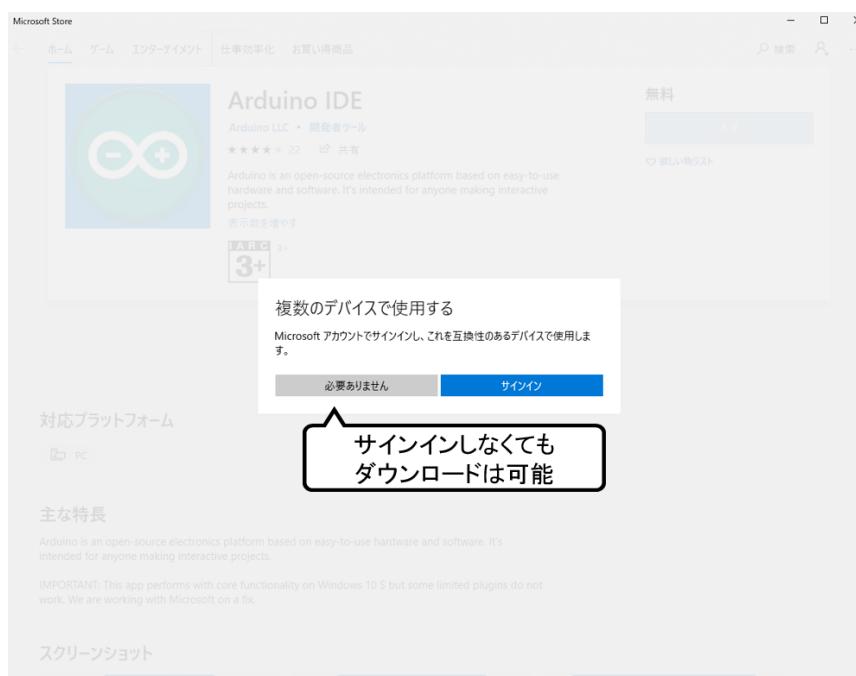


図 2.6: サインインの確認画面

図 2.7 では Arduino IDE のダウンロード状況を確認できます。



図 2.7: ダウンロードのキュー画面

ダウンロードが完了した後、検索窓にて Arduino IDE を検索し開いてください(図 2.8)。

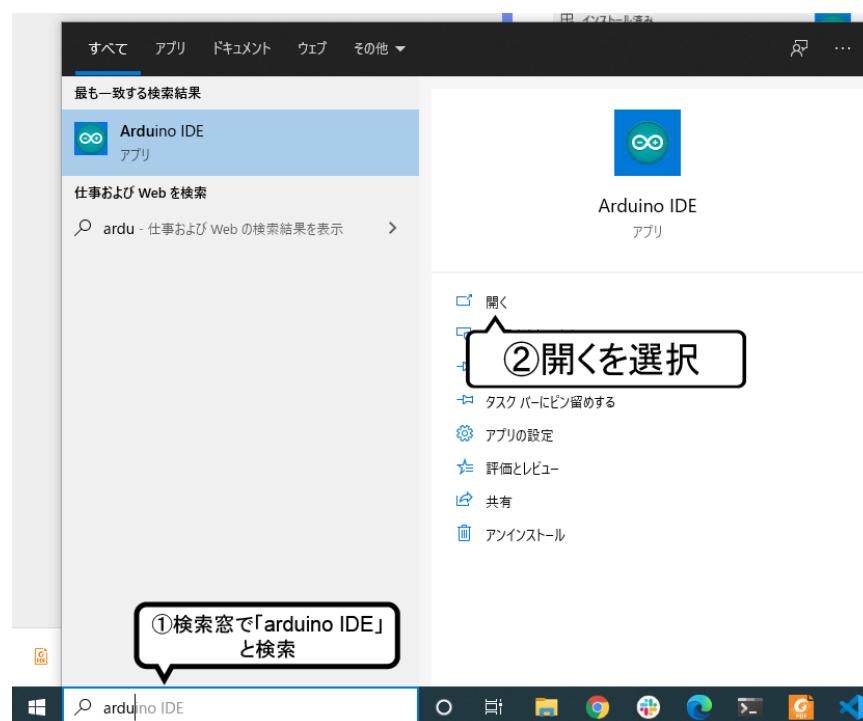


図 2.8: ArduinoIDE の検索

開いた際、セキュリティについての許可を求められるので(図 2.9) アクセスを許可するを選択してください。

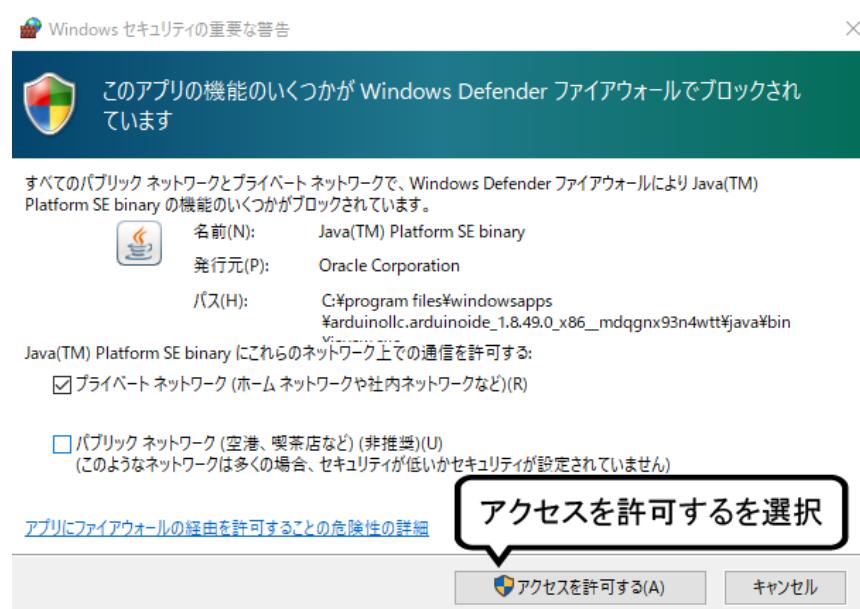


図 2.9: セキュリティの確認画面

Arduino IDE が起動すると、デフォルトの画面が表示されます（図 2.10）。

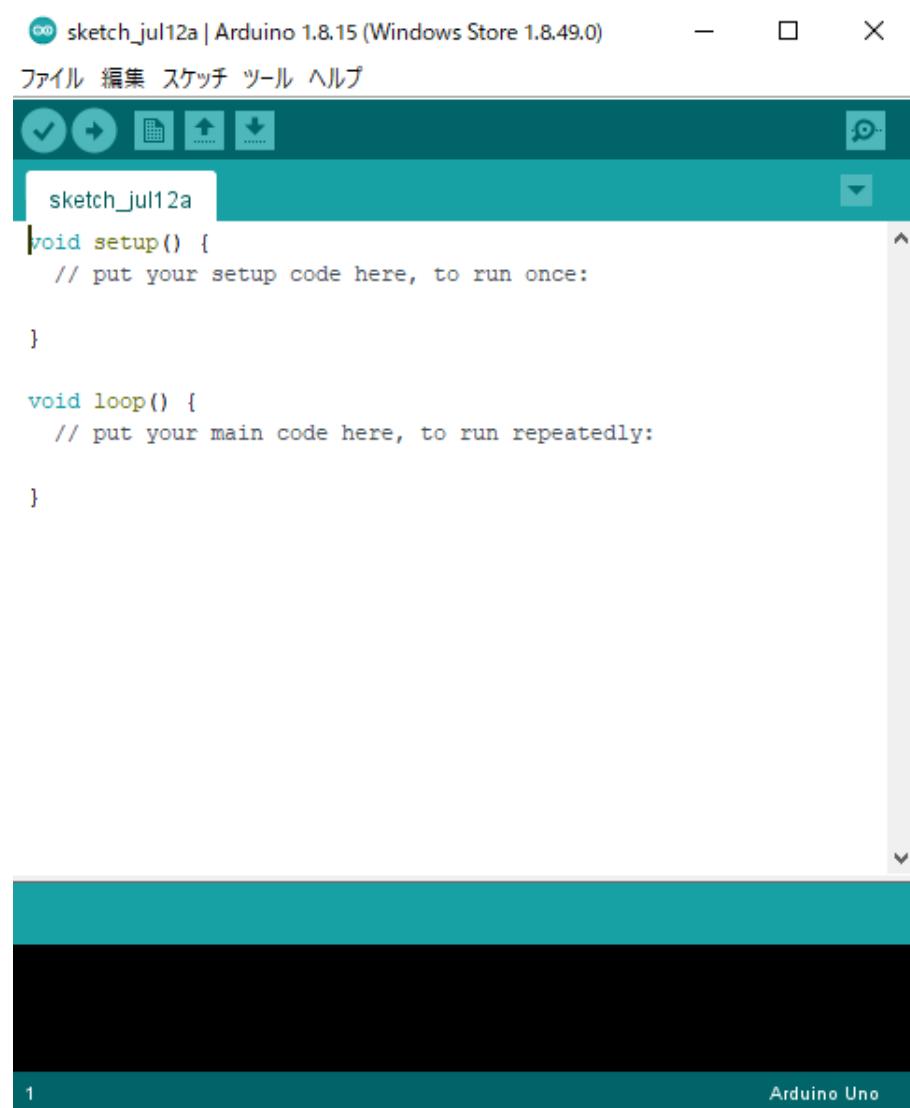


図 2.10: デフォルトのスケッチ画面

以上で Arduino IDE のインストールは完了です。

2.4 ESP32用ボードマネージャーのインストール

Arduino IDE にて ESP32 を使うために必要なボードマネージャーのインストール方法を紹介します。

図 2.11 は ESP32 のボードマネージャーを追加するための手順であり、以下のリンクに記載されています。

https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md

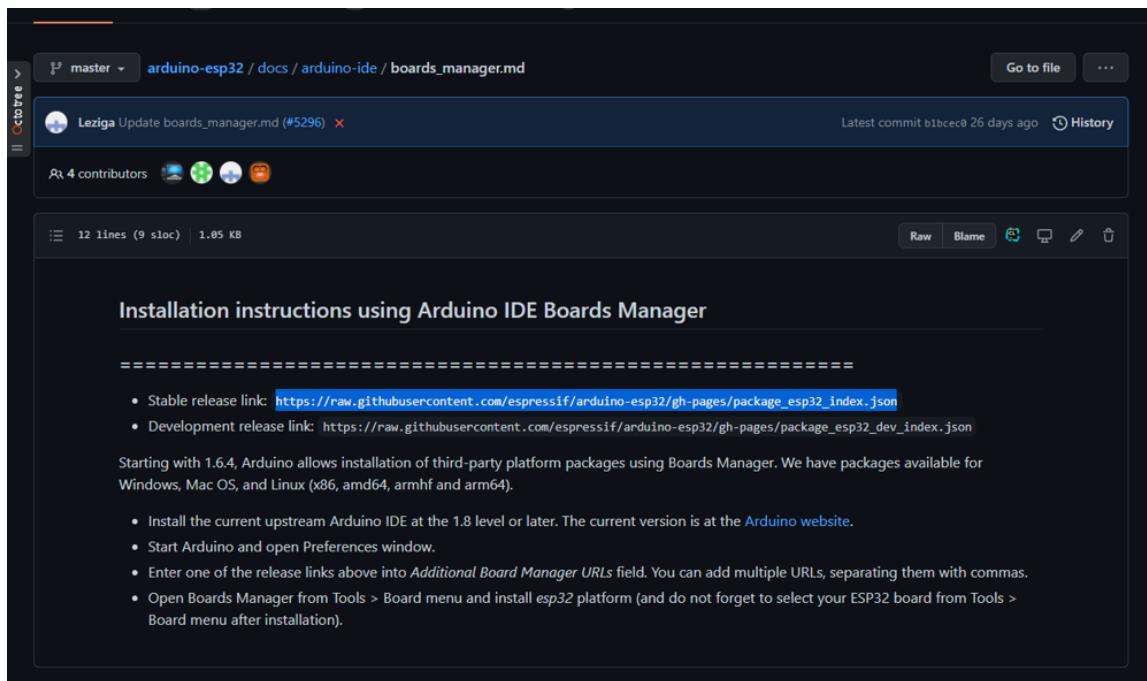


図 2.11: ESP32 を ArduinoIDE で使うための設定

手順に従い以下のリンクをコピーしてください(リスト 2.1)。以下のリンクには、図 2.12 のような情報が記載されています。以下のリンクでは改行をしていますが実際は一文のため注意してください。

リスト 2.1: ボードマネージャーのリンク

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/燐  
package_esp32_index.json
```

```

{
  "packages": [
    {
      "name": "esp32",
      "maintainer": "Espressif Systems",
      "websiteURL": "https://github.com/espressif/arduino-esp32",
      "email": "christo@espressif.com",
      "help": {
        "online": "http://esp32.com"
      }
    }
  ],
  "platforms": [
    {
      "name": "esp32",
      "architecture": "esp32",
      "version": "1.0.6",
      "category": "ESP32",
      "url": "https://github.com/espressif/arduino-esp32/releases/download/1.0.6/esp32-1.0.6.zip",
      "archiveFileName": "esp32-1.0.6.zip",
      "checksum": "SHA-256:982da9aa181b8cb9c892dd4c9822b022ecc0d1e3aa0c5b70428ccc3c1b4556b",
      "size": "51126602",
      "help": {
        "online": ""
      }
    }
  ],
  "boards": [
    {
      "name": "ESP32 Dev Module"
    },
    {
      "name": "WEMOS LoLin32"
    },
    {
      "name": "WEMOS D1 MINI ESP32"
    }
  ],
  "toolsDependencies": [
    {
      "packager": "esp32",
      "name": "xtensa-esp32-elf-gcc",
      "version": "1.22.0-97-ac792ad5-5.2.0"
    },
    {
      "packager": "esp32",
      "name": "esptool_py",
      "version": "3.0.0"
    },
    {
      "packager": "esp32",
      "name": "mkspiffs"
    }
  ]
}

```

図 2.12: ESP32用のボードマネージャ情報

Arduino IDE 側では、(ファイル > 環境設定) を選択してください (図 2.13)

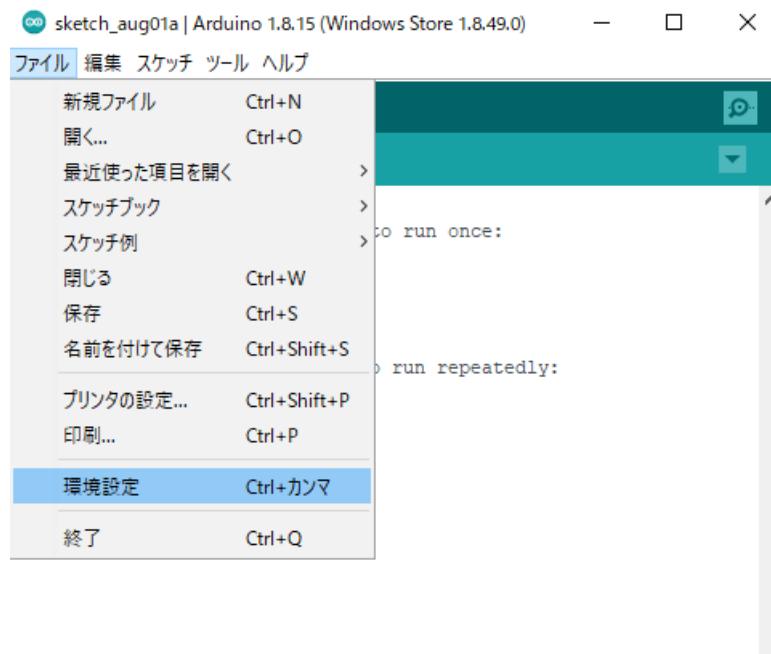


図 2.13: 環境設定を選択

選択した後、環境設定の画面が表示されていることを確認してください（図 2.14）。



図 2.14: 環境設定の画面

次に、先ほどコピーしたリンク（リスト 2.1）を追加ボードマネージャーの URL の欄にペーストしてください（図 2.15）。

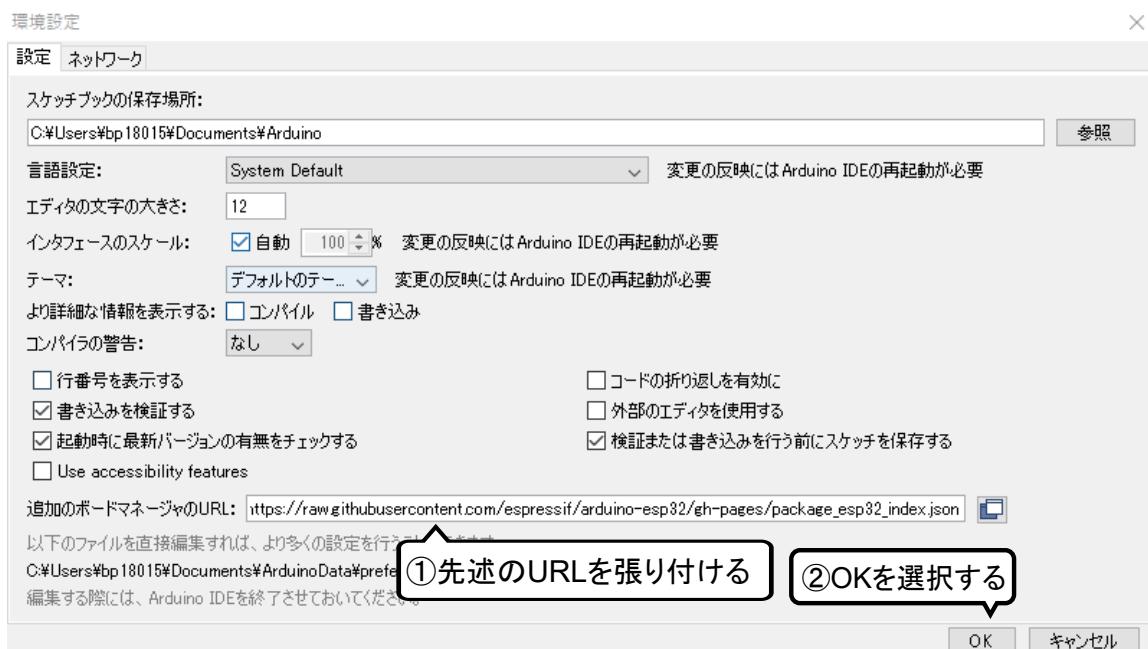


図 2.15: 追加ボードマネージャーの URL に貼り付ける

その後、OKを選択してください。

次に、(ツール>ボード>ボードマネージャー)を開いてください。

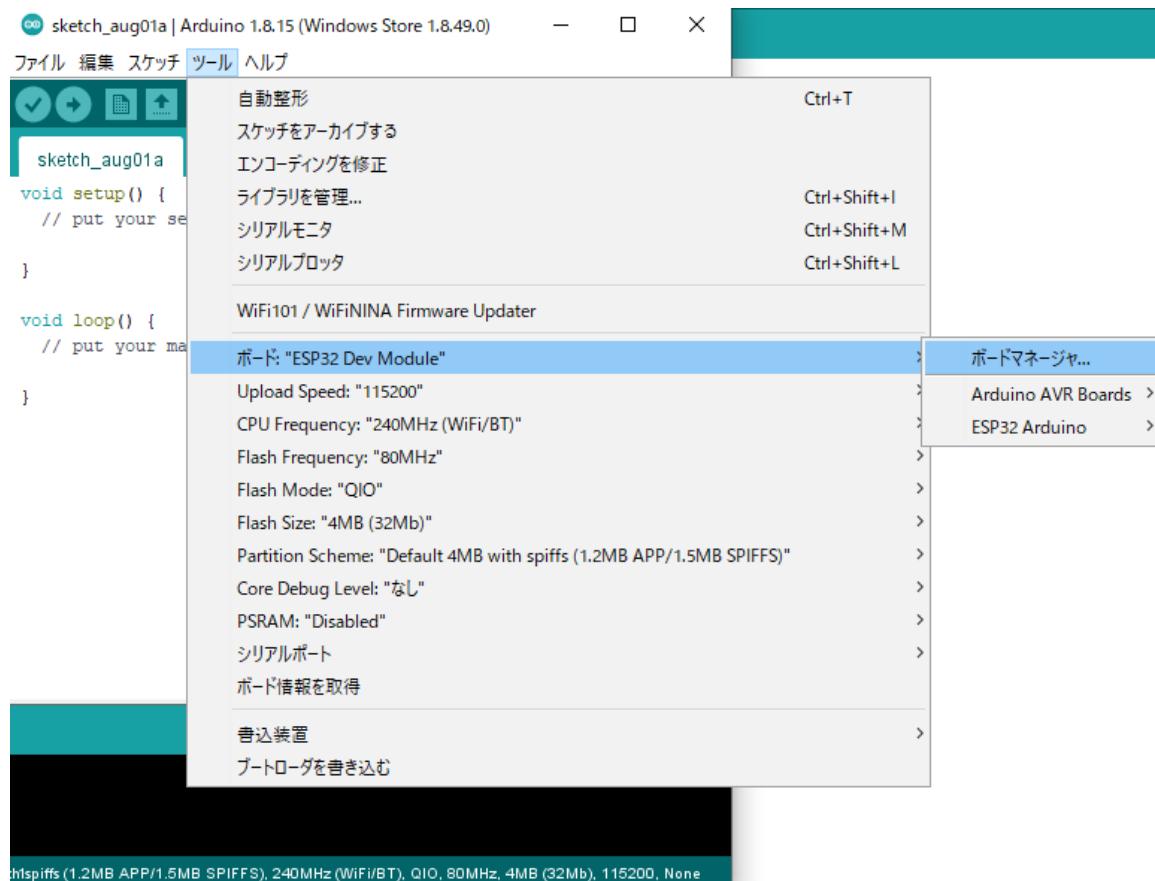


図 2.16: ボードマネージャーを開く

開かれたボードマネージャーの検索窓に「ESP32」を入力しインストールをしてください。

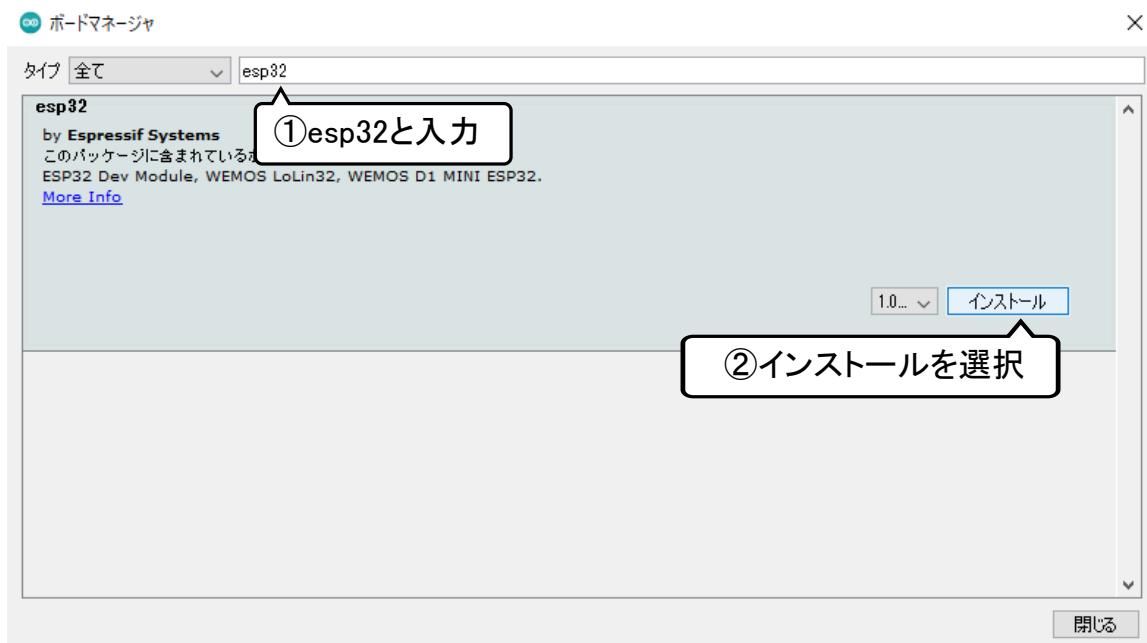


図 2.17: ESP32用ボードマネージャーのインストール

インストールが完了した後、(ツール > ボード > ESP32 Arduino > ESP32 Dev Module) を選択してください。

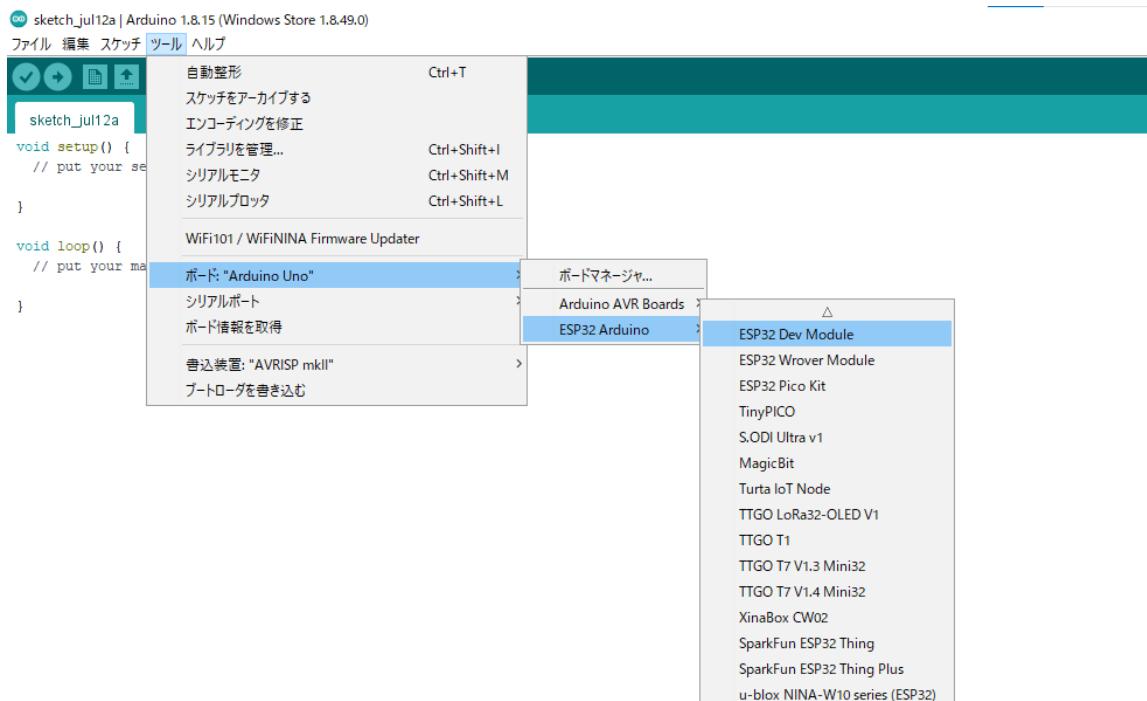


図 2.18: ボード ESP32 Dev Module の選択

2.5 Hello ESP32!!

ここで動作確認するためにプログラミングでは定番の HelloWorld を ESP32 でやってみましょう。

ブレッドボード

これから作業のために ESP32 をブレッドボードにさします。図 2.19 のように、esp32 をブレッドボード中央あたりに差し込んでください。ブレッドボードの説明をします。ブレッドボードは電子回路を仮組みする際によく使われます。ブレッドボードにさした部品は再利用できるため、いろいろな回路を試すことができます。ブレッドボードの最大の特徴として図 2.19 のように、回路的につながっている部分とつながっていない部分に分かれているところがあげられます。最初のうちは、回路的につながっている黄色の部分を忘れて、ショートする回路を作ってしまうことがあるので注意してください。

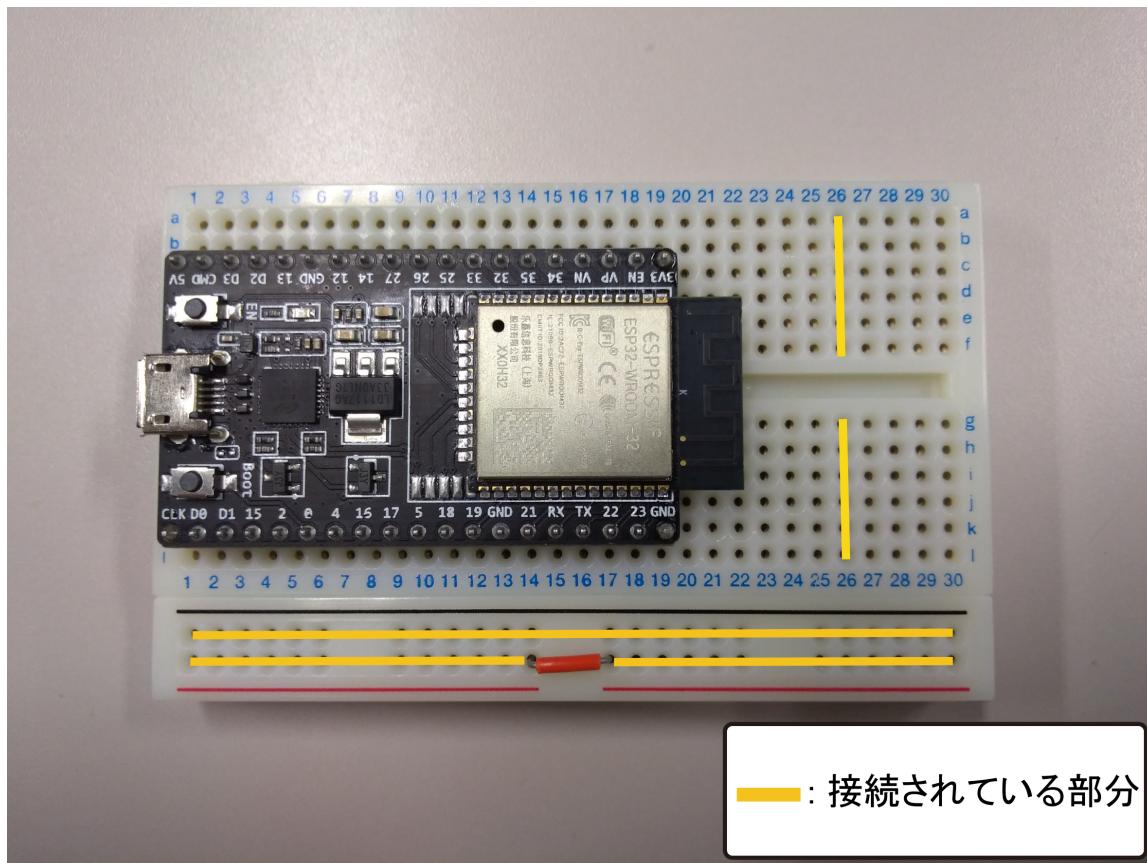


図 2.19: ブレッドボード

PCとの接続

つぎに、ESP32をPCと接続します。まずmicroUSB Type-Bの差し込み口に(図2.20) microUSB Type-B端子を差し込んでください。

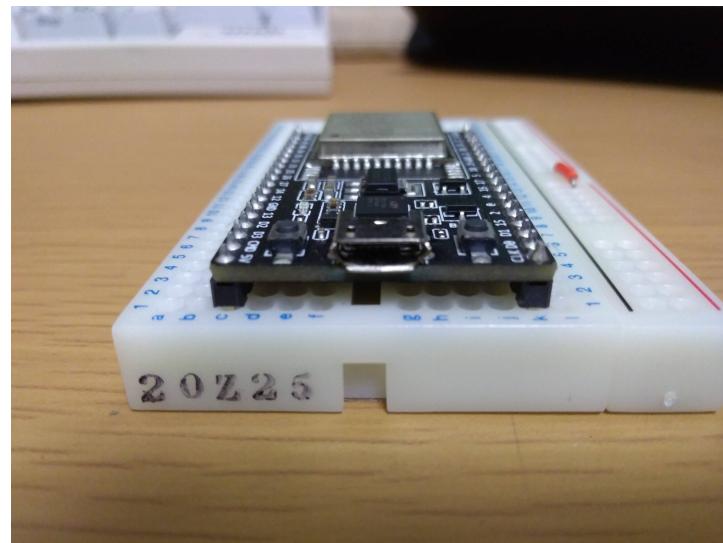


図2.20: microUSB type-B 差し込み口

その後、PCとesp32を接続してください。接続が完了するとesp32上のLEDが光ります(図2.21)。

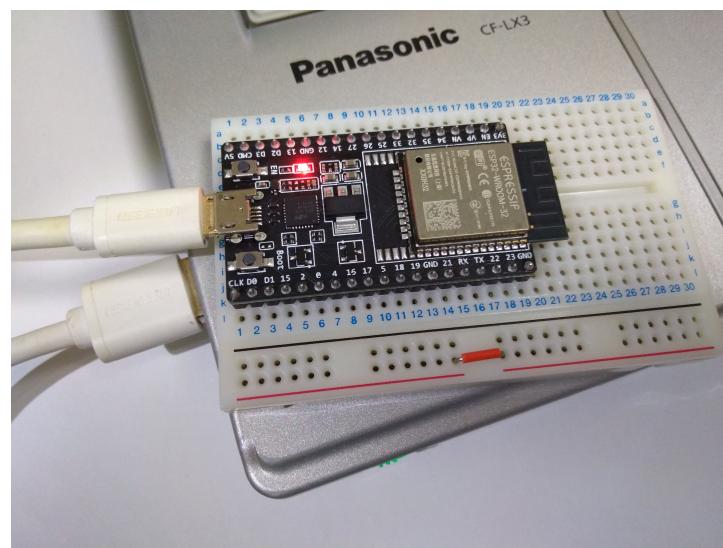


図2.21: PCとの接続

次にデバイスマネージャーを用いて、ESP32 がつながっているポート番号を調べます。デバイスマネージャーを開いてください（図 2.22）。



図 2.22: デバイスマネージャーの検索

ESP32 は Silicon Labs CP210x USB to UART Bridge という名前で COM3 につながっていることがわかります（図 2.23）。接続ポートは環境によって異なります。接続ポートに ESP32 にがない場合は「A.5 接続ポートに ESP32 が反映されない」を参照してください。



図 2.23: ESP32 の接続ポートを調べる

先ほど調べた接続ポートを反映するため ツール>シリアルポートを選択し変更してください。

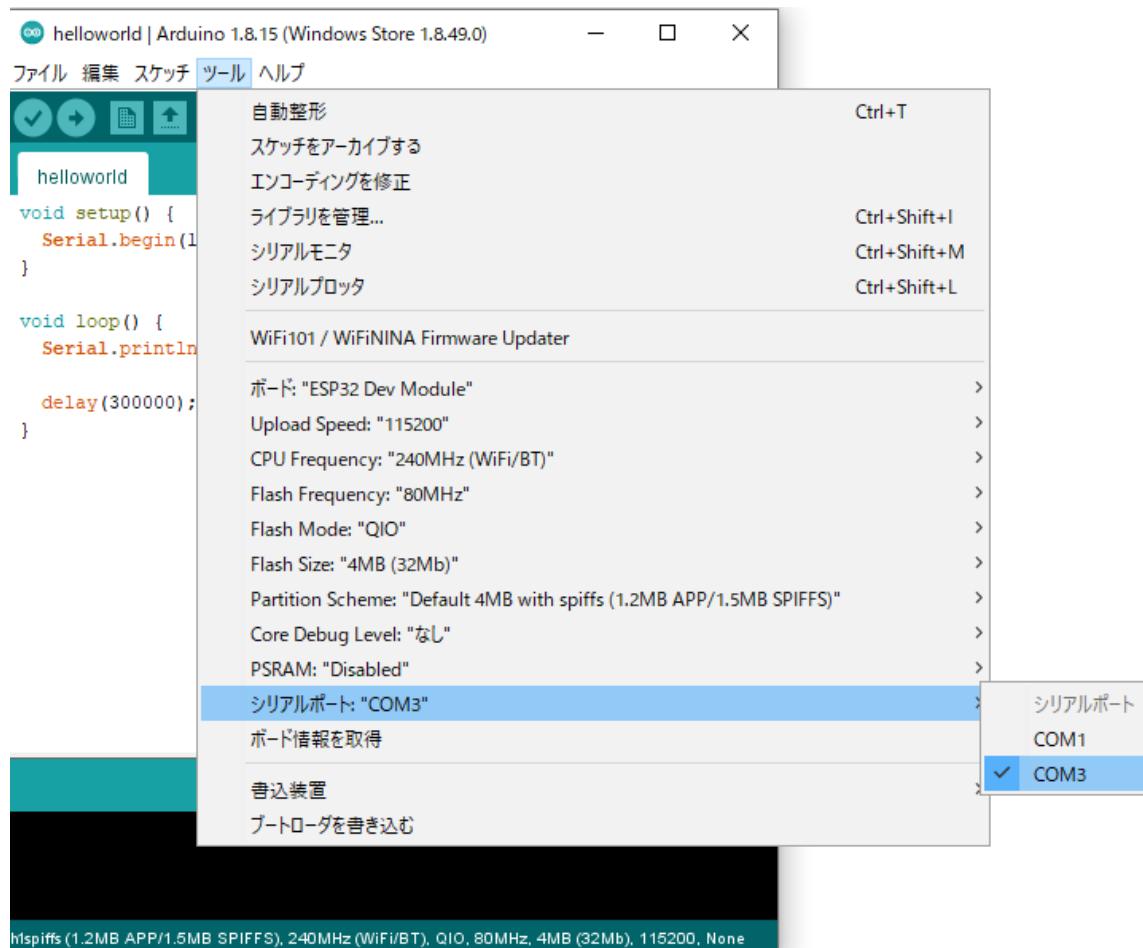


図 2.24: 接続ポートの反映

設定を確認します。ツールを開いて UploadSpeed が 115200 であることを確認してください。

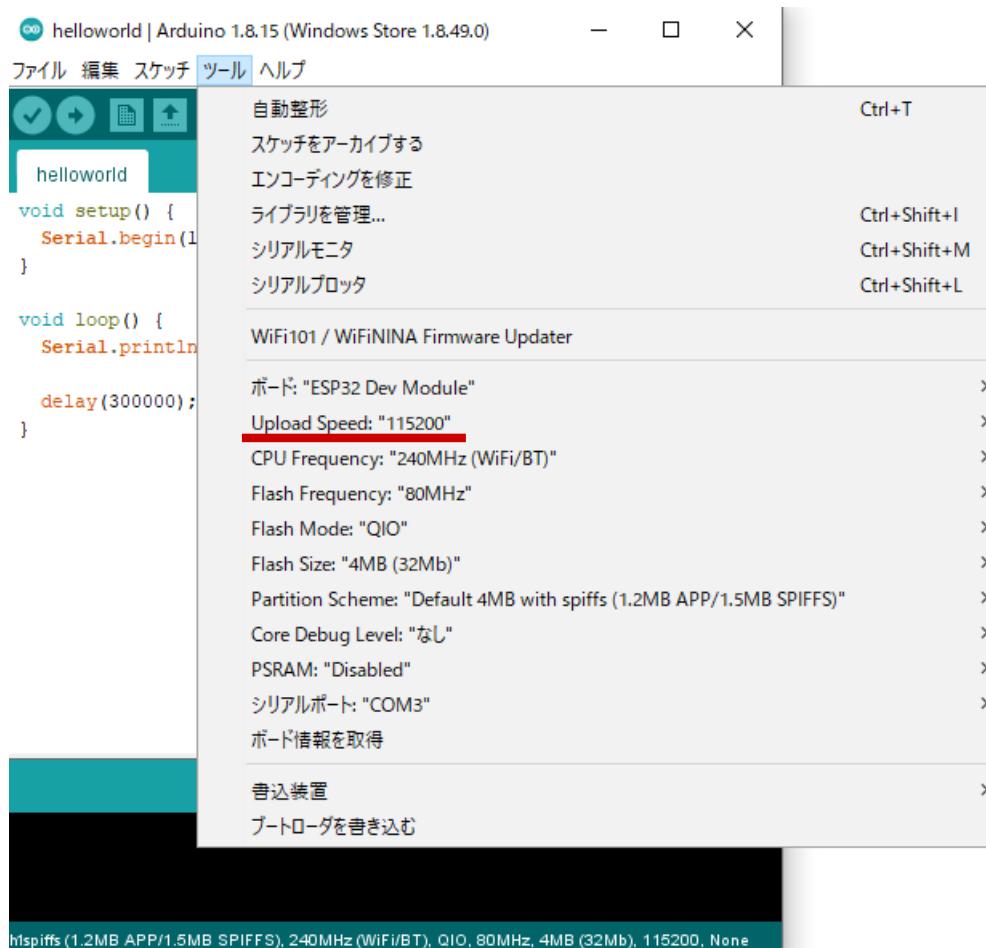


図 2.25: ボードの設定

プログラムの記述

HelloWorld を実行するため、新しくファイルを作成します。ファイル>新規ファイルを選択してください（図 2.26）。

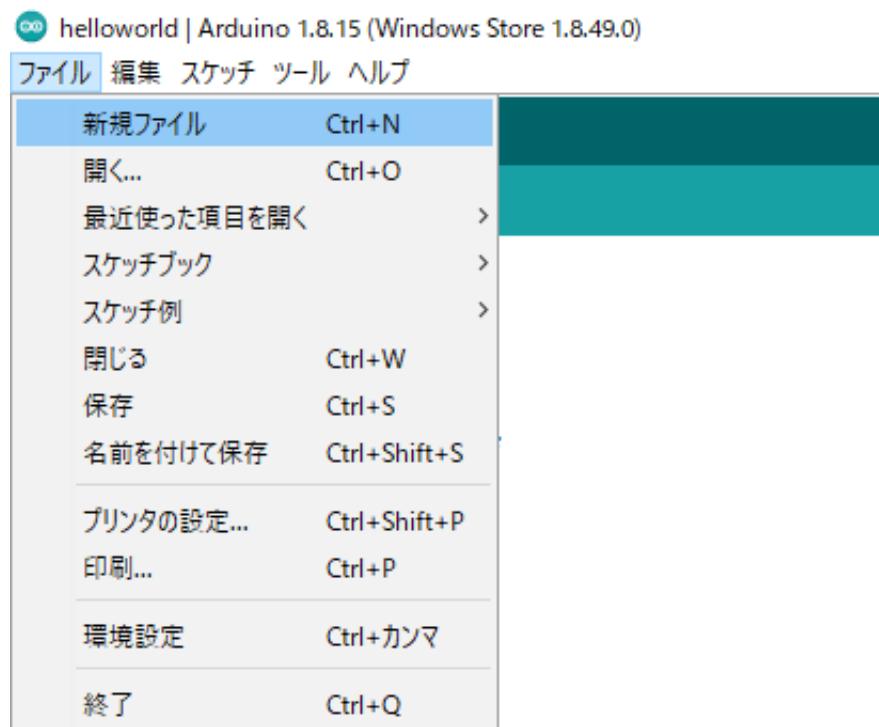


図 2.26: 新規ファイルの作成

ファイルエクスプローラーが開かれるので、ファイル名に `helloworld` と入力して保存してください（図 2.27）。

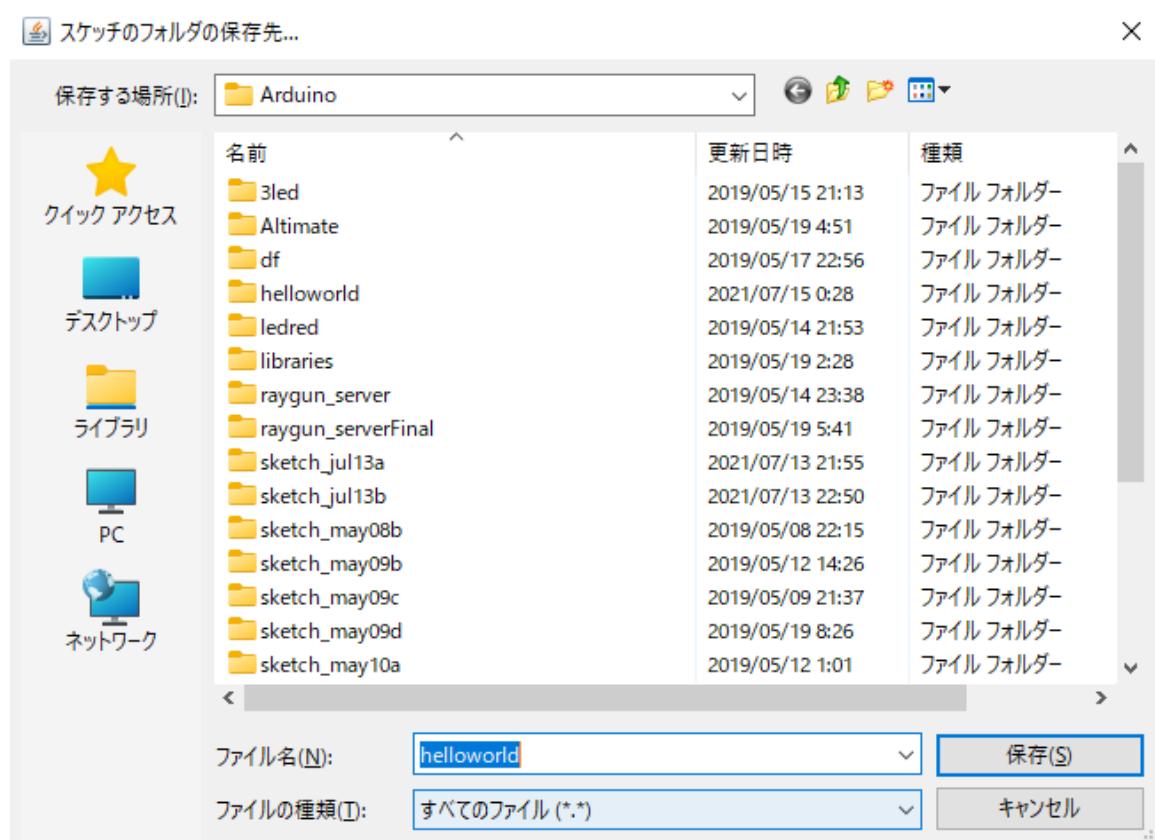


図 2.27: 新規ファイルの名前決定

つぎに、リスト 2.2 を参考にして図 2.28 のようにプログラムを記述してください。

リスト 2.2: HelloWolrd

```
void setup() {
    Serial.begin(115200); // シリアル通信をUploadSpeed 115200bpsで開始
}

void loop() {
    Serial.println("Hello,World"); // シリアル通信で"Hello,World"を送信する
    delay(3000); // 3000ms (3秒) 停止する
}
```



The screenshot shows the Arduino IDE interface with the title bar "helloworld | Arduino 1.8.15 (Windows Store 1.8.49.0)". Below the title bar is a menu bar with "ファイル" (File), "編集" (Edit), "スケッチ" (Sketch), "ツール" (Tools), and "ヘルプ" (Help). A toolbar with icons for file operations (checkmark, arrow, file, up, down) is located above the code editor. The code editor window has a teal header bar with the project name "helloworld". The main area contains the following C++ code:

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.println("Hello,World");
  delay(3000);
}
```

図 2.28: HelloWorld のプログラムを記述

プログラムの説明

ここで、先ほど記述したプログラムの説明をします。まず、ESP32 のプログラムは大枠として

- `setup()`
- `loop()`

の二つに分類されます。`setup()` は起動時に一回だけ実行され、`loop()` は `setup()` の実行後、無限に繰り返されます。そのため、`setup()` 内には初期化などの処理を書き、`loop()` 内にはセンサーの値取得など逐次取得したい内容を書きます。

プログラムの書き込み

ここで、esp32 にプログラムを書き込みます。矢印を選択し、プログラミングを書き込んでください（図 2.29）



図 2.29: ESP32 にプログラムを書き込む

矢印を選択するとプログラムの書き込みが開始します。書き込みの様子はコンソール画面にて確認できます（図 2.30）。

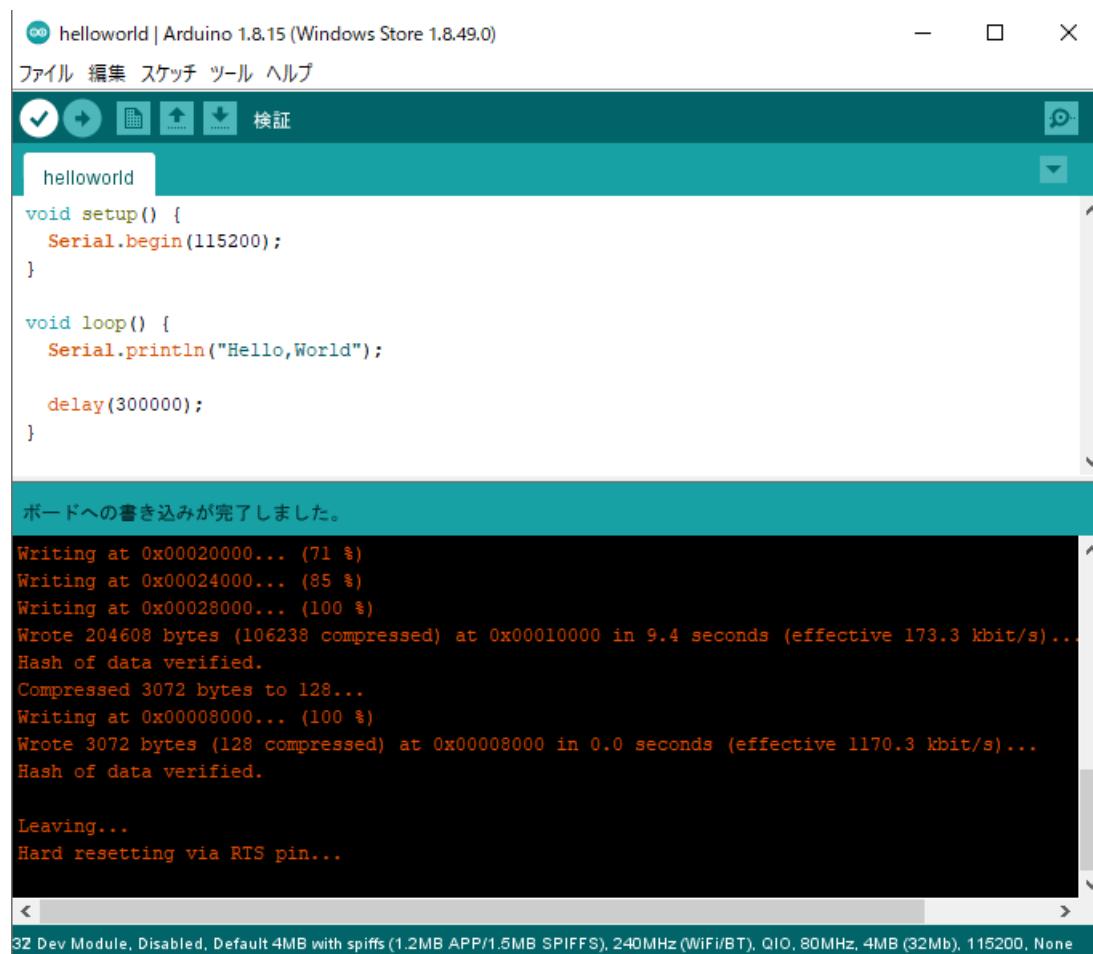


図 2.30: コンソール画面

動作確認

ESP32 からの HelloWorld を表示するために、シリアルモニタを開きます。ツール > シリアルモニタを選択してください（図 2.31）。

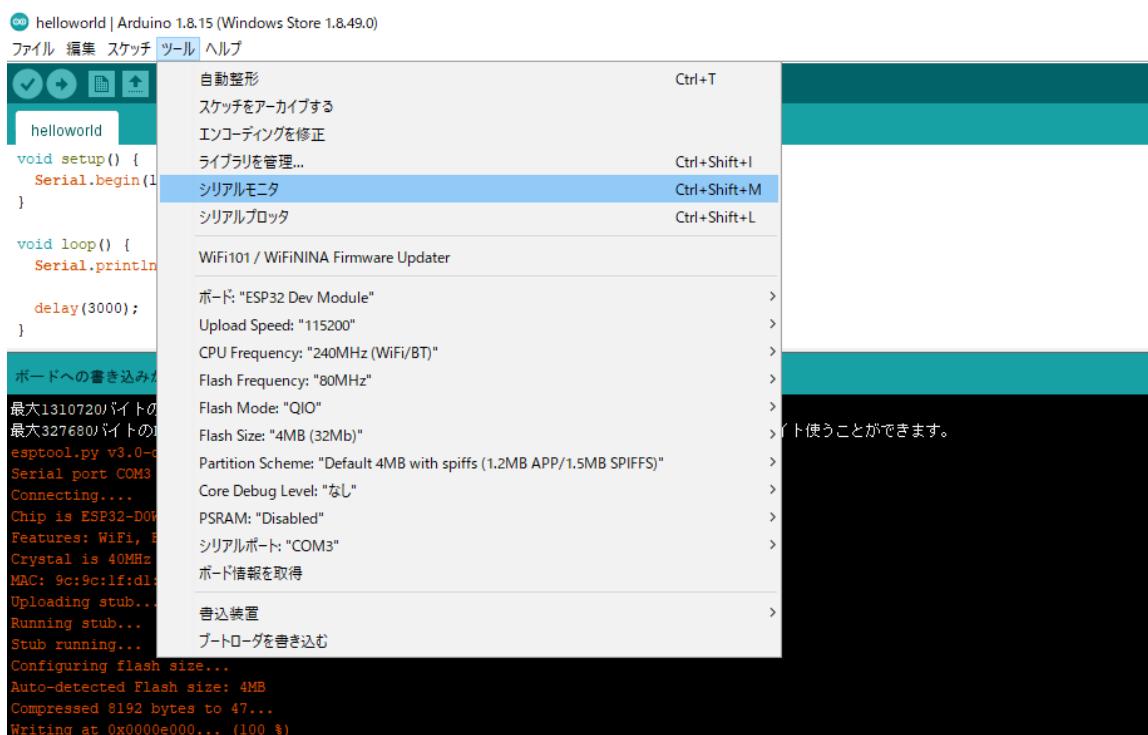


図 2.31: シリアルモニタの選択

ESP32 から HelloWorld が送られてくることを確認できました（図 2.32）。

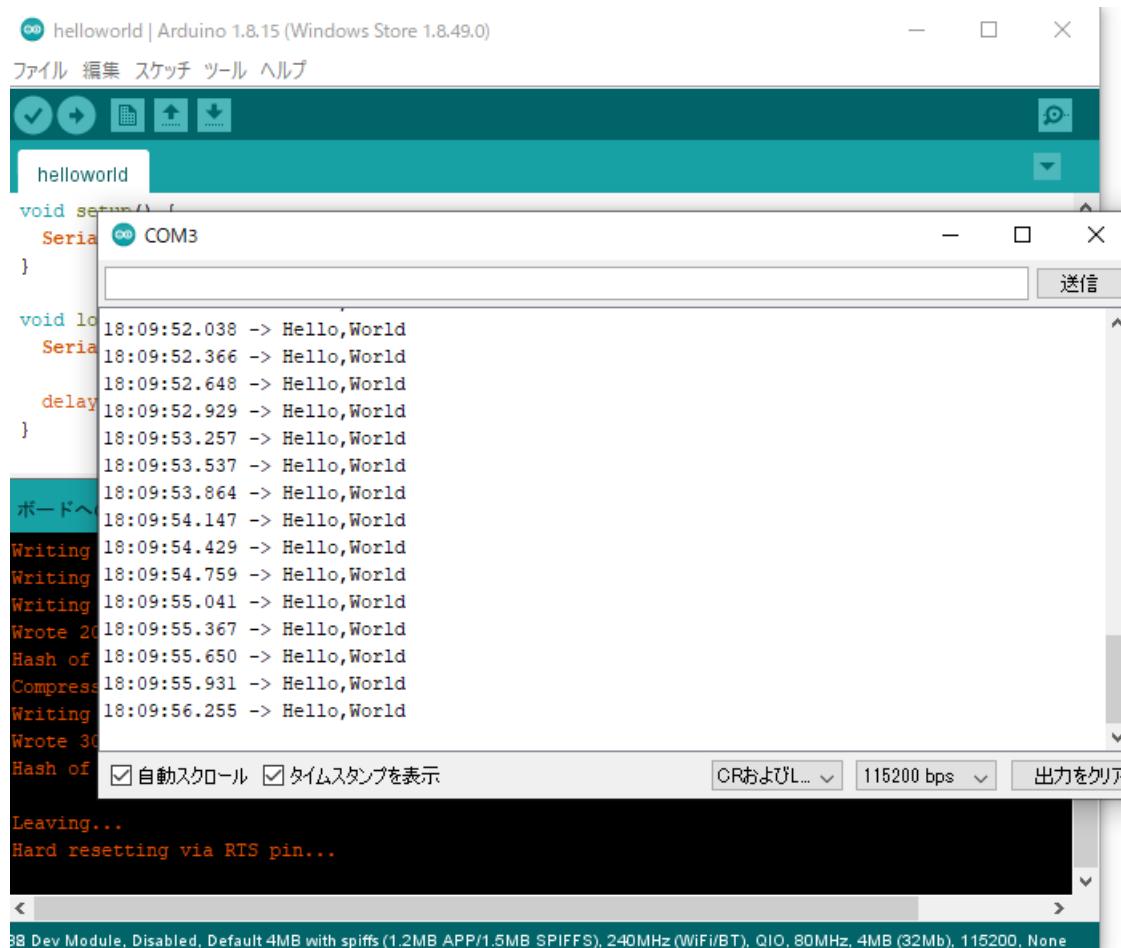


図 2.32: helloworld の表示成功

コラム: シリアル通信とは

シリアル通信とは通信線を用いて信号を HIGH と LOW の 1/0 の組み合わせの連続（シリアル）的に情報を送信するものです。HelloWorld を受信した際に使用したシリアルモニタは ESP32 から送られてきた情報を表示したり送信したりする機能です。またシリアル通信では送信速度と受信速度を一致させる必要があり、これを一秒あたりのビット数 (bps) として表します。プログラムで記載した

```
Serial.begin(115200);
```

も esp32 と PC との間の通信速度を 115200bps として設定しています。ほかにも

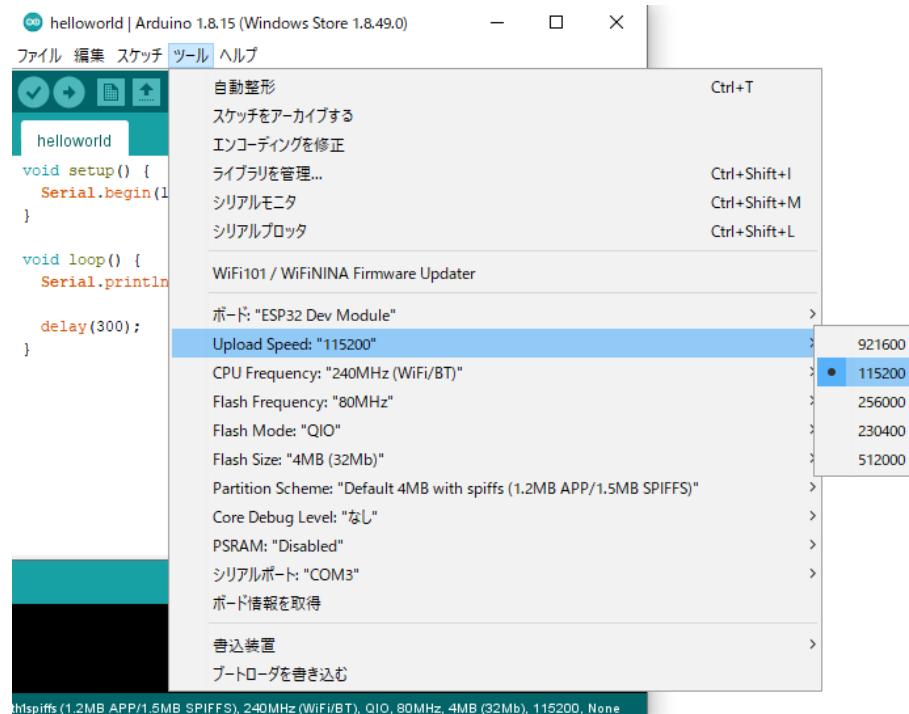


図 2.33: UploadSpeed の設定

設定の UploadSpeed (図 2.33) や

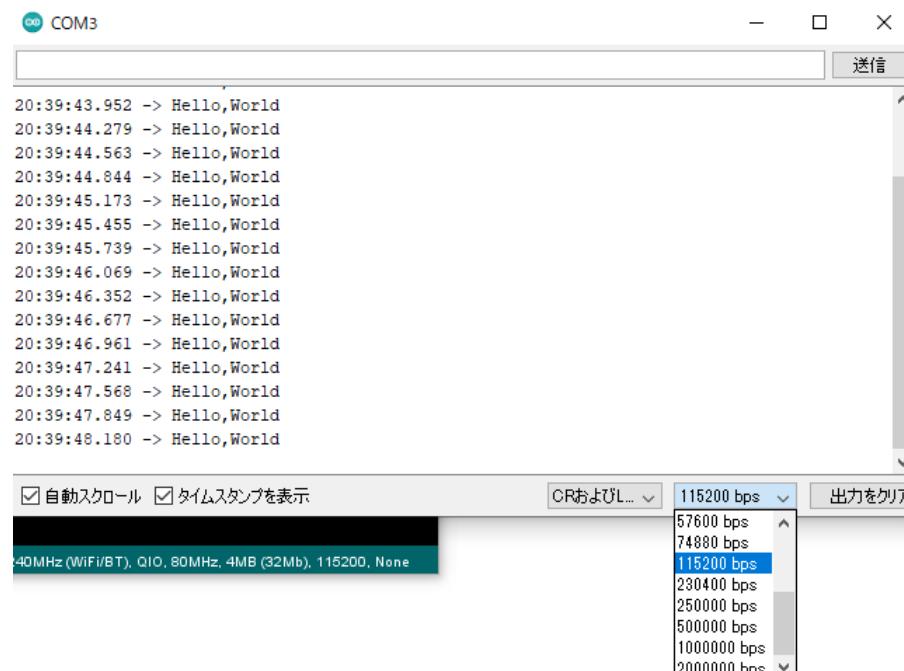


図 2.34: シリアルモニタの bps の設定

シリアルモニタでの設定（図 2.34）も一致する数字にする必要があります。ちなみに USB も Universal Serial Bus の略であり、シリアル通信を行っています。

第3章

電子部品を使ってみよう

3.1 部品説明

ESP32 で電子部品を用いた回路を組む前に、それぞれの部品の概要を紹介します。

LED

LED (発光ダイオード) は決まった方向に電圧を加えることで、発光する半導体素子です。 LED には極性があり、以下の二つに分けられます (図 3.1)。

- アノード
 - 端子の長いほうをアノードと呼び電源の + に接続する
- カソード
 - 端子の短いほうをカソードと呼ぶ GND (マイナス) に接続

極性を逆に繋ぐと光らないので注意してください。

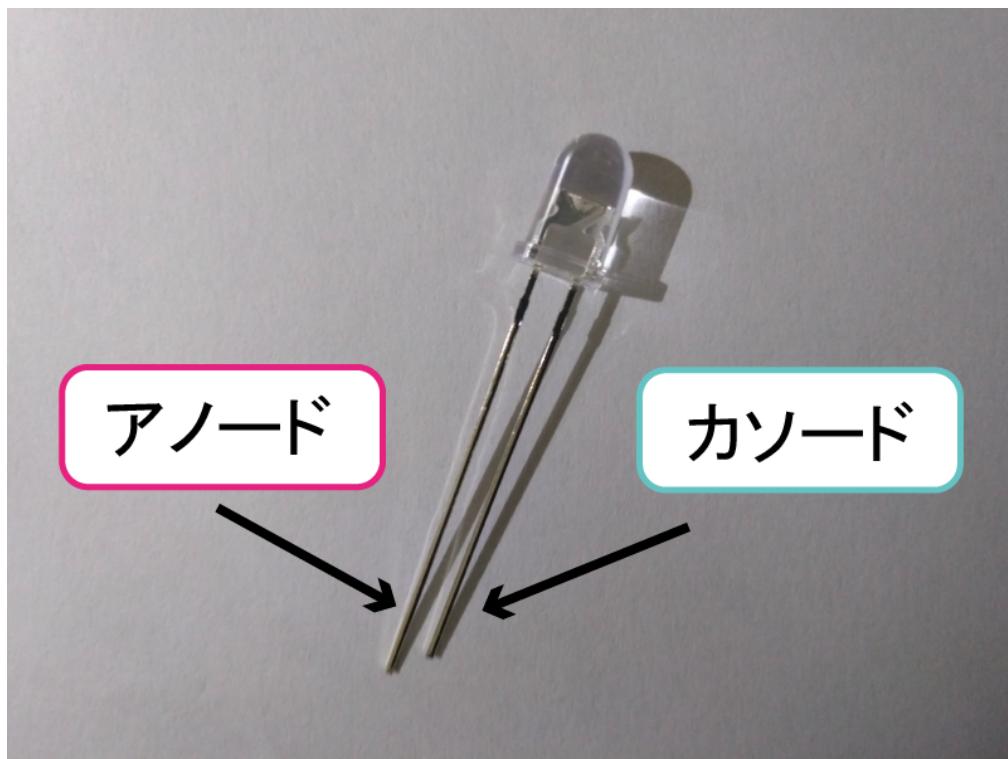


図 3.1: LED の端子の違い

ちなみに電流を流しすぎると下記の画像（図 3.2）の右側 LED のように燃えて使えなくなってしまいます。



図 3.2: 燃えてしまった LED

極性のほかにも点灯のために必要な情報があり、代表的なものに以下の二つがあります。

- 順電圧 (Vf)
 - LED は発光するため一定値以上の電圧をかける必要があります。これを順電圧 (Vf) と呼びます。
- 順電流 (If)
 - LED は流す電流の大きさによって明るさが変わりますが、電流を流しすぎると壊れてしまいます。そこで順電流 (If) を用いて適切な電流値を表します。

抵抗値の求め方

先述の二つを用いて LED と ESP32 の間に接続する抵抗値の選択をします。今回使う LED は

順電圧 (Vf) : 2.1V 順電流 (If) : 20mA です。

まず LED にかかる抵抗にかかる電圧を求めます。GPIO から出力される電圧は 3.3V、LED にかかる電圧は順電圧を用いて 2.1V とします。そこから LED にかかる電圧を求める

$$3.3V - 2.1V = 1.2V$$

より 1.2V が抵抗にかかる電圧だと分かりました。次に抵抗にかかる電流値を求めます。ESP32 の GPIO にかけてもいい最大電流値は 20mA (おそらく) なのでオームの法則を用いて抵抗値を求める

$$1.2V / 0.02A = 60$$

60 Ω が必要ということが分かりました。しかし、今回は 100 Ω の抵抗を用意したので

$$1.2V / 100 \Omega = 0.012A (12mA)$$

より 100 Ω の抵抗を使っても流れる電流は 12mA であり ESP32 の許容範囲内です。

ジャンプワイヤ

主にブレッドボード上で、電子回路を仮組する際に使われるものがジャンプワイヤです。ジャンプワイヤには、いくつかの種類があり主に以下の二つがあります。

- オス・オス

- 両端子とも基盤にさして使う（図 3.3）
- オス・メス
 - 片方が端子を差し込めるようになっている（図 3.4）



図 3.3: ジャンプワイヤ オス・オス

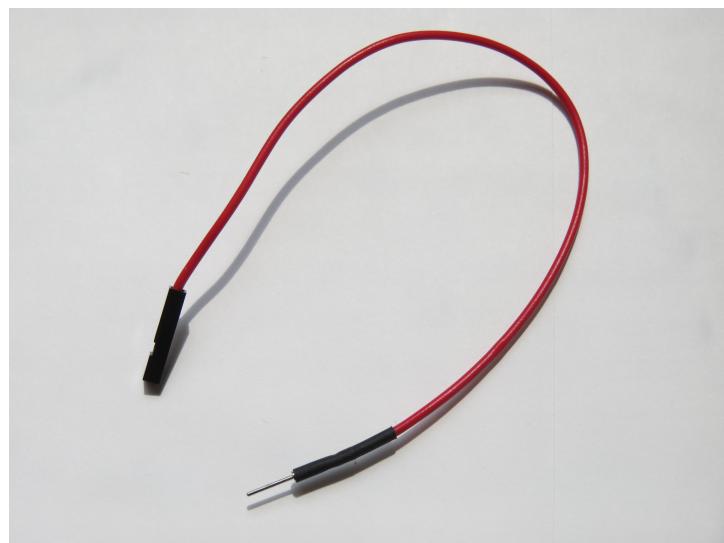


図 3.4: ジャンプワイヤ オス・メス

抵抗

電子回路上を流れる電流を調整するために使われるのが抵抗であり（図3.5）抵抗値によって様々なものがあります。これを見分けるための規格があり以下の表のように決められています（表3.1）。

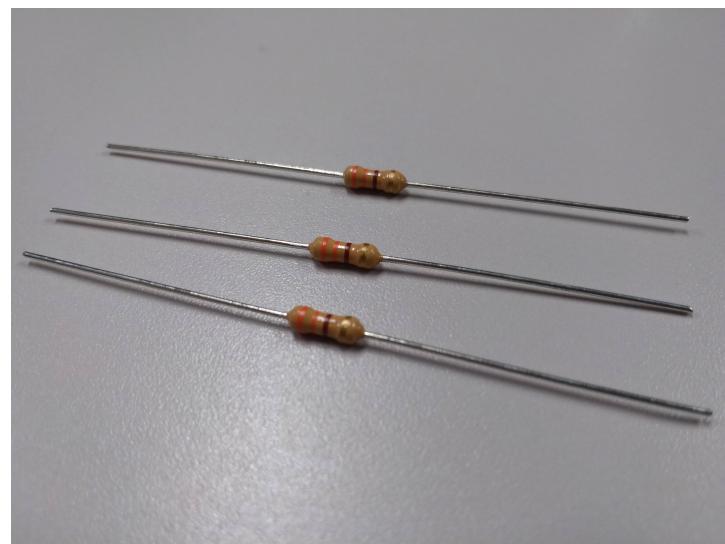


図3.5: 330 抵抗

許容差とは抵抗のばらつき度を表しています。

表 3.1: 抵抗のカラーコード

色	有効数字	乗数	許容差 [%]
黒	0	$10^0(1)$	
茶	1	$10^1(10)$	± 1
赤	2	$10^2(100)$	± 2
オレンジ	3	$10^3(1000)$	± 0.05
黄	4	$10^4(10000)$	
緑	5	$10^5(100000)$	± 0.5
青	6	$10^6(1000000)$	± 0.25
紫	7	$10^7(10000000)$	± 0.1
灰	8	$10^8(100000000)$	
白	9	$10^9(1000000000)$	
金		$10^{(-1)}(0.1)$	± 5
銀		$10^{(-2)}(0.01)$	± 10
無色			± 20

実際に抵抗の値をカラーコードから読み取ってみましょう。図 3.6 を見てください。今回は 4 本線の場合を考えます。まず左側の二つの線の色を見るとどちらもオレンジなので表 3.1 から 33 ということが分かります。次に右側の二つ線の色を見ると茶色と金色なので乗数が 10 許容差が $\pm 5\%$ ということが分かります。これらのことから、この抵抗は抵抗値 330 で許容差 $\pm 5\%$ だということが分かりました。

$$33 \times 10 = 330\Omega \pm 5\%$$

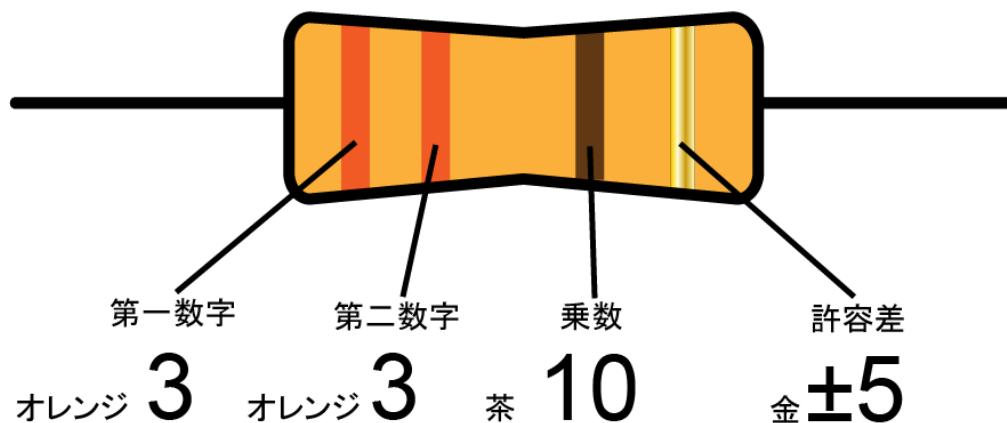


図 3.6: 抵抗値の読み取り

タクトスイッチ

タクトスイッチはスイッチを押すことで、電流の止めたり流したりできる部品です。主にプログラムのリセットやメッセージ送信スイッチなどに使われます。

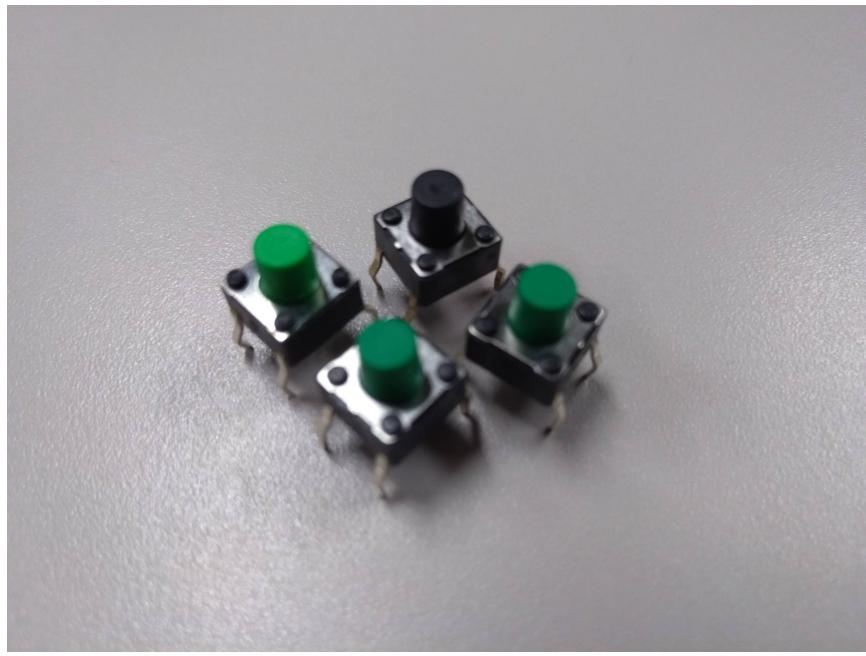


図 3.7: タクトスイッチ

タクトスイッチには通常時繋がっている部分と繋がっていない部分があります。図 3.8を見てもらうと分かるように黄色の線の部分は通常時繋がっていて、オレンジ色の線の部分はタクトスイッチを押すと繋がります。

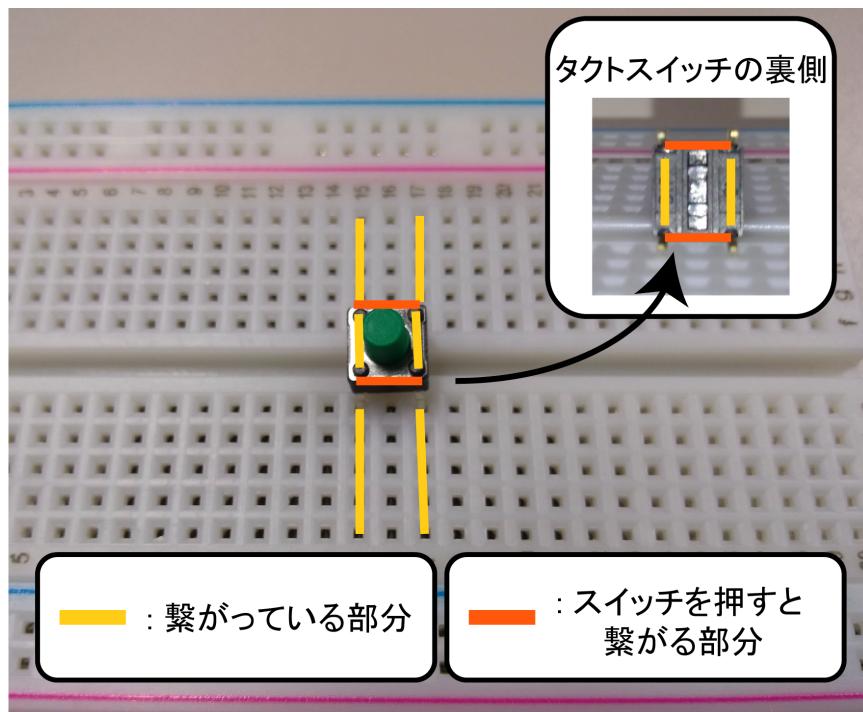


図 3.8: タクトスイッチの特徴

プルアップとプルダウン

タクトスイッチなどのスイッチを用いることで、電流を流したり止めたりすることができます。しかし、出力する端子に何もつながっていない状況（図 3.9）では不安定になってしまいノイズなどの影響を受けやすくなってしまいます。そこで、これを解決するために以下の二つの方法を用います（図 3.10）。

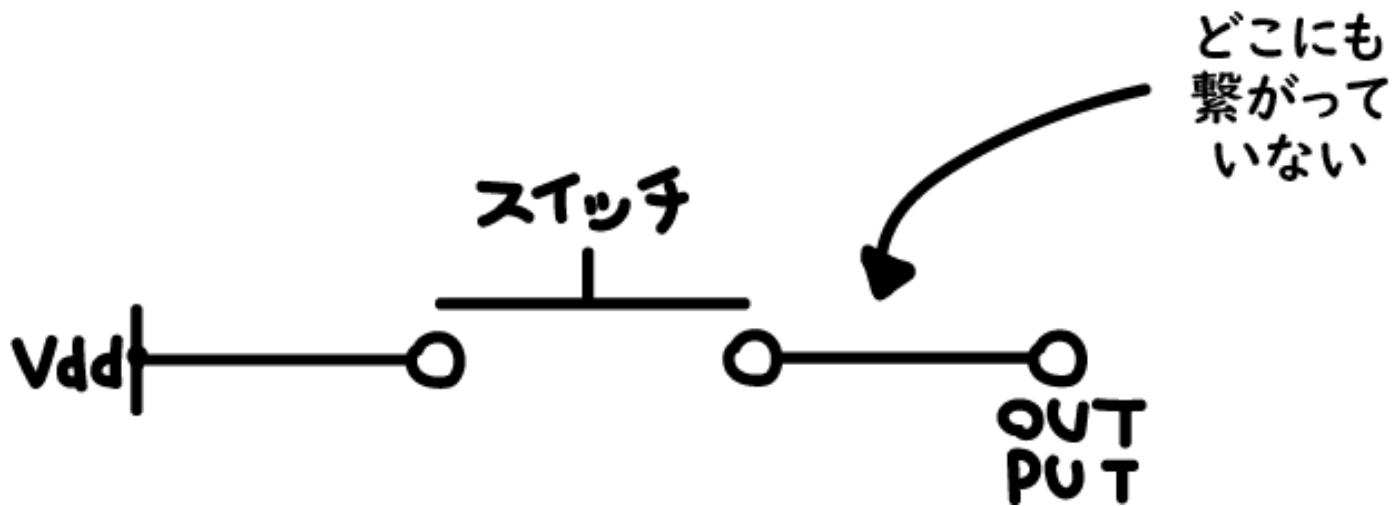


図 3.9: オフ状態のスイッチ

まず前提として、Vdd は電気の + 側を GND は電気の - 側を表しています。また OUT PUT は ESP32 でいうところの入出力が可能なピンである GPIO (General Purpose Input/Output: 汎用入出力) をさしています。

まず図 3.10 の左側に記載してある PULL UP は常に OUT PUT に電圧をかけ、スイッチが押された際には OUT PUT への電圧が 0 になる回路です。これによりノイズの影響を少なくしています。

つぎに図 3.10 の右側に記載してある PULL DOWN は OUT PUT を GND につないでおくことでかかる電圧を 0 に維持しています。スイッチが押された際には電圧が OUT PUT にかかります。これによりノイズの影響を少なくしています。

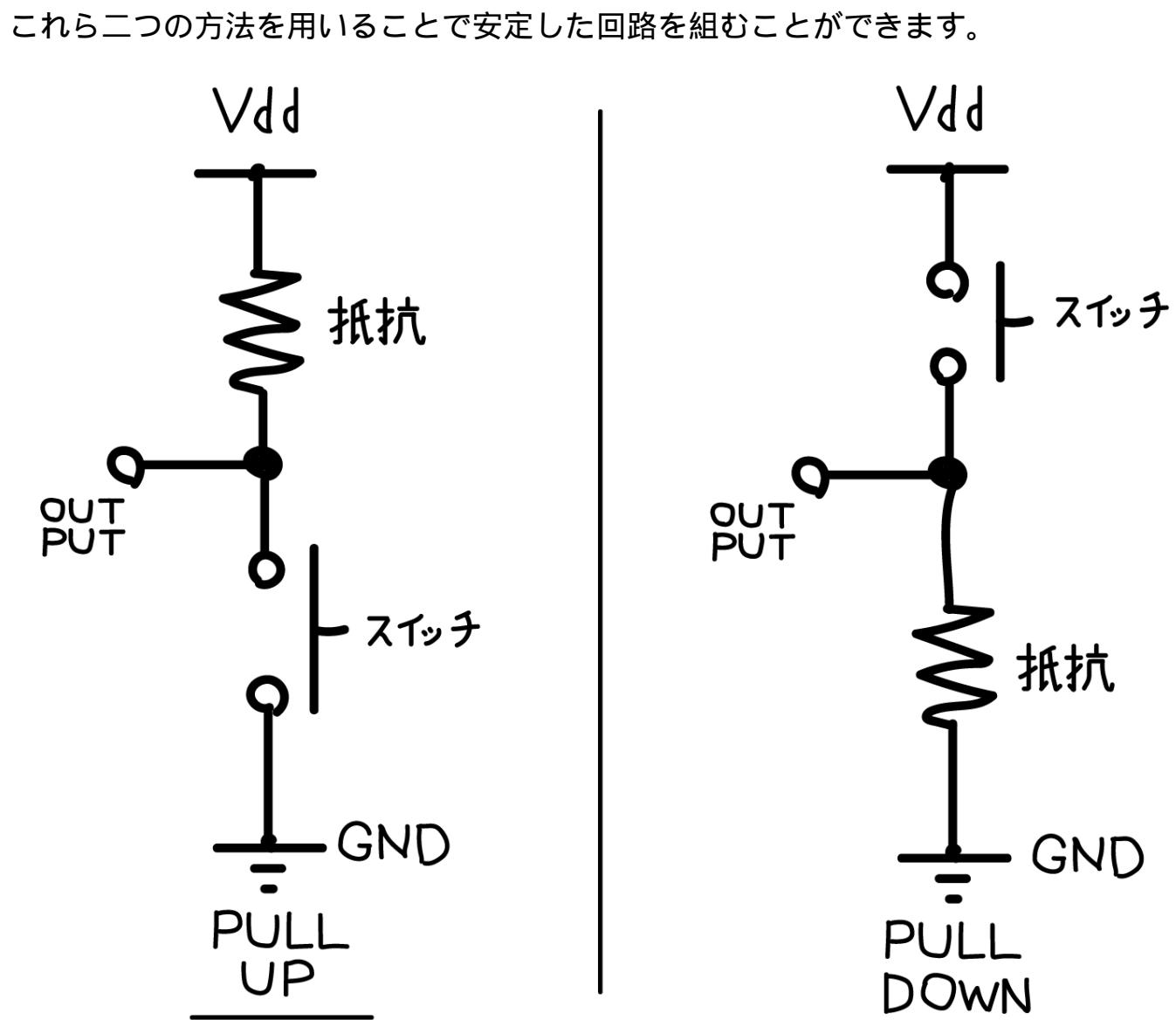


図 3.10: PULL UP と PULL DOWN

3.2 Lチカしよう！

Lチカとは、さまざまな言語のプログラムでのHelloWorldに相当するハードウェア操作のプログラムです。LEDをチカチカ点滅させてみましょう！！

プログラムでLチカ

Lチカですが、ESP32を使用することで容易に実現できます。Arduino IDEから新規作成を選択し、新たなファイルを作成して以下のプログラムを貼り付けてください（リスト3.1）。回路図ではブレッドボードを二枚用いていますが（図3.11）、今回使用するブレッドボードでは一枚の中に収まるようになっています（図3.12、図3.13）。

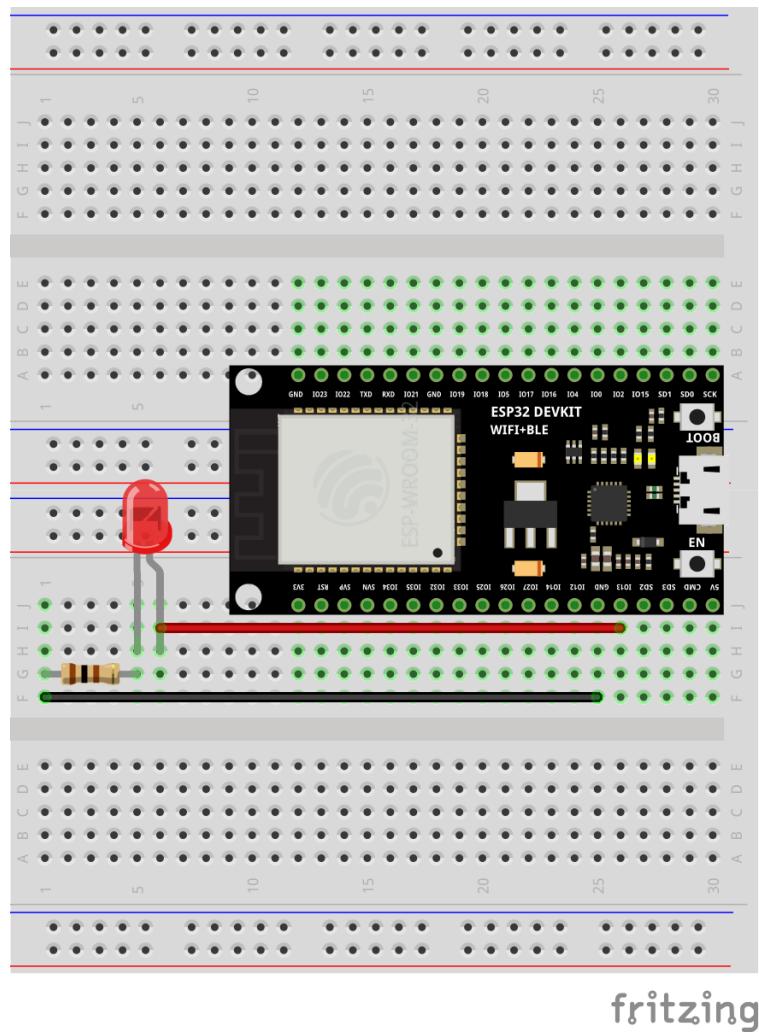


図3.11: Lチカの回路図

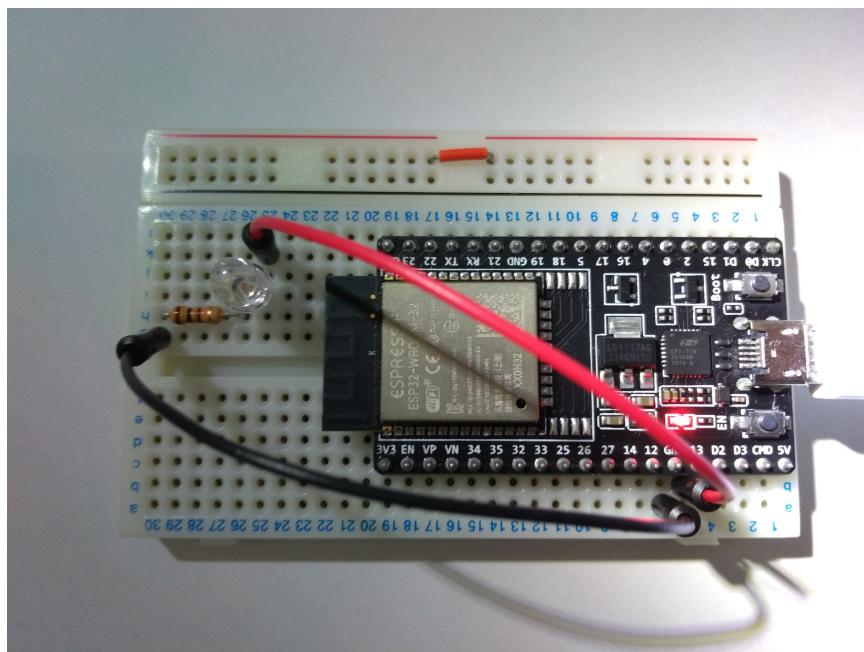


図 3.12: Lチカの回路を配置した図

リスト 3.1: Lチカのプログラム

```
void setup() {
    pinMode(13, OUTPUT); // GPIO13を出力端子として用いる
}
void loop() {
    digitalWrite(13, HIGH); // GPIO13に電流を流しLEDを光らせる
    delay(100); // 0.1s停止
    digitalWrite(13, LOW); // GPIO13の電流を止める
    delay(100); // 0.1s停止
}
```

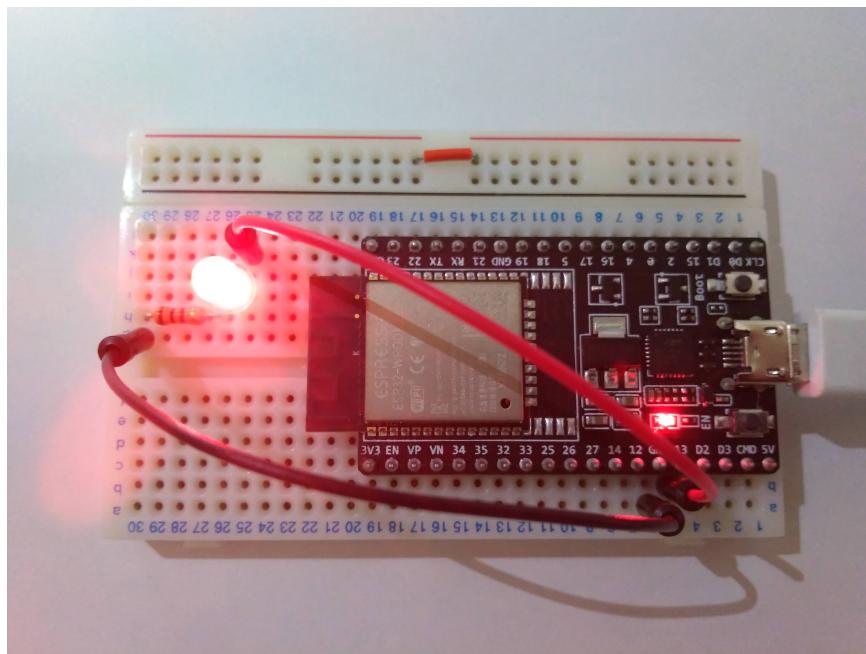
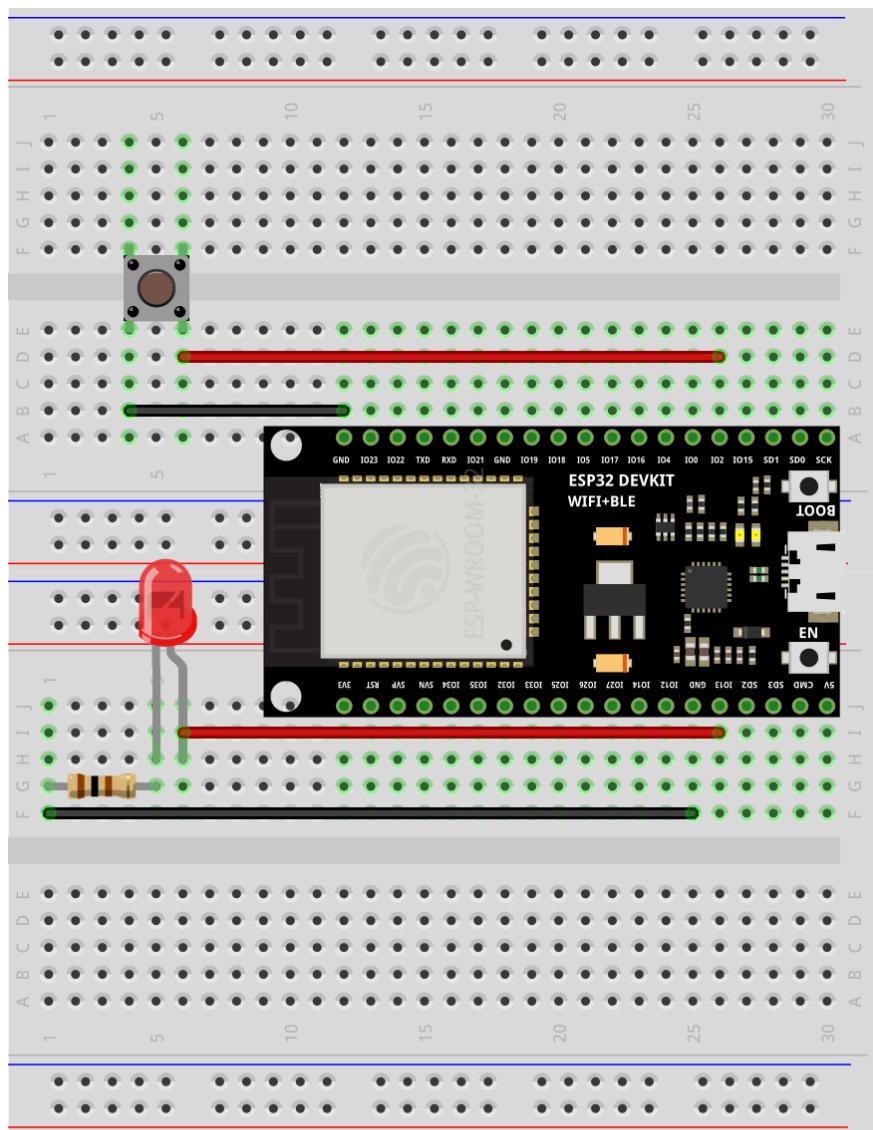


図 3.13: LED 点滅図

タクトスイッチでLチカ

せっかくなので、スイッチを使用して、LED を光らせましょう同様に以下の画像（図 3.14、図 3.15）を参考に電子回路を組み、プログラム（リスト 3.2）を参考にして、ESP32 に書き込んでください。



fritzing

図 3.14: タクトスイッチで L チカをする回路図

リスト 3.2: タクトスイッチで L チカをするプログラム

```
void setup()
{
    Serial.begin(115200); // シリアル通信を115200bpsで開始する
    pinMode(2, INPUT_PULLUP); // GPIO2を用いるまた、ESP32の内部でPULL UPをする
    pinMode(13, OUTPUT); // GPIO13を出力端子として用いる
}

void loop()
{
```

```
if (digitalRead(2) == LOW) // PULL UPを用いているので電圧が低いと、  
    // スイッチが押されていると判定する  
{  
    delay(100); // チャタリング防止のため0.1s停止  
    digitalWrite(13, HIGH); // GPIO13に電流を流しLEDを光らせる  
    Serial.println("ON!");  
}  
if (digitalRead(2) == HIGH) // スイッチを押していない場合  
{  
    delay(100); // チャタリング防止のため0.1s停止  
    digitalWrite(13, LOW); // GPIO13の電流を止める  
    Serial.println("OFF!");  
}  
}
```

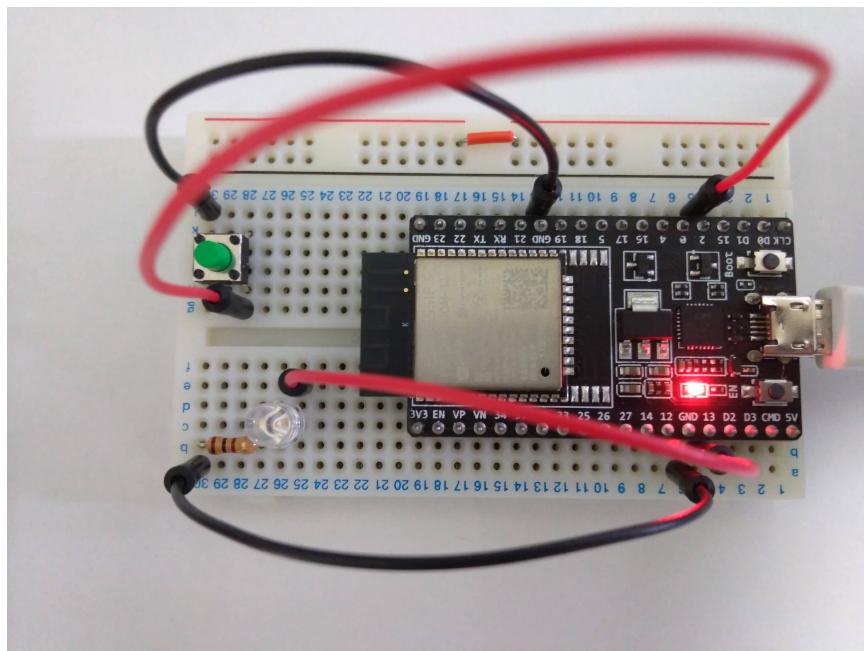


図 3.15: 回路をブレッドボード上で配置した図

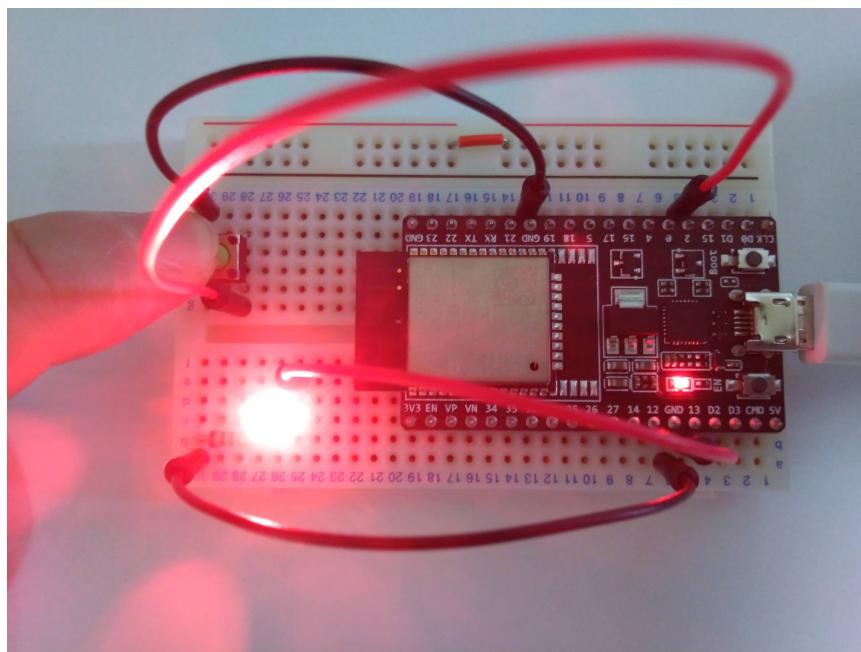


図 3.16: スイッチで LED を点滅させる図

コラム: チャタリング

スイッチを押した際、内部の金属板の接触によって電流が流れます。しかし、人間がスイッチを押すときには一回だけ押したつもりでも内部的には金属板が何回も接触し複数回の ONOFF が発生する可能性があります。そのためプログラム側での対策として、delay を何秒か挟むことで複数回の ONOFF を防ぐことができます（リスト 3.3）。

リスト 3.3: チャタリング防止のプログラム

```
if (digitalRead(2) == LOW) // PULL UPを用いているので電圧が低いと、スイッチが押されていると判定する
{
    delay(100); // チャタリング防止のため0.1s停止
    Serial.println("ON!");
}
```

3.3 応用問題: 状態遷移

二つの LED とスイッチを使用して、二つの LED の状態を以下のように変更してください

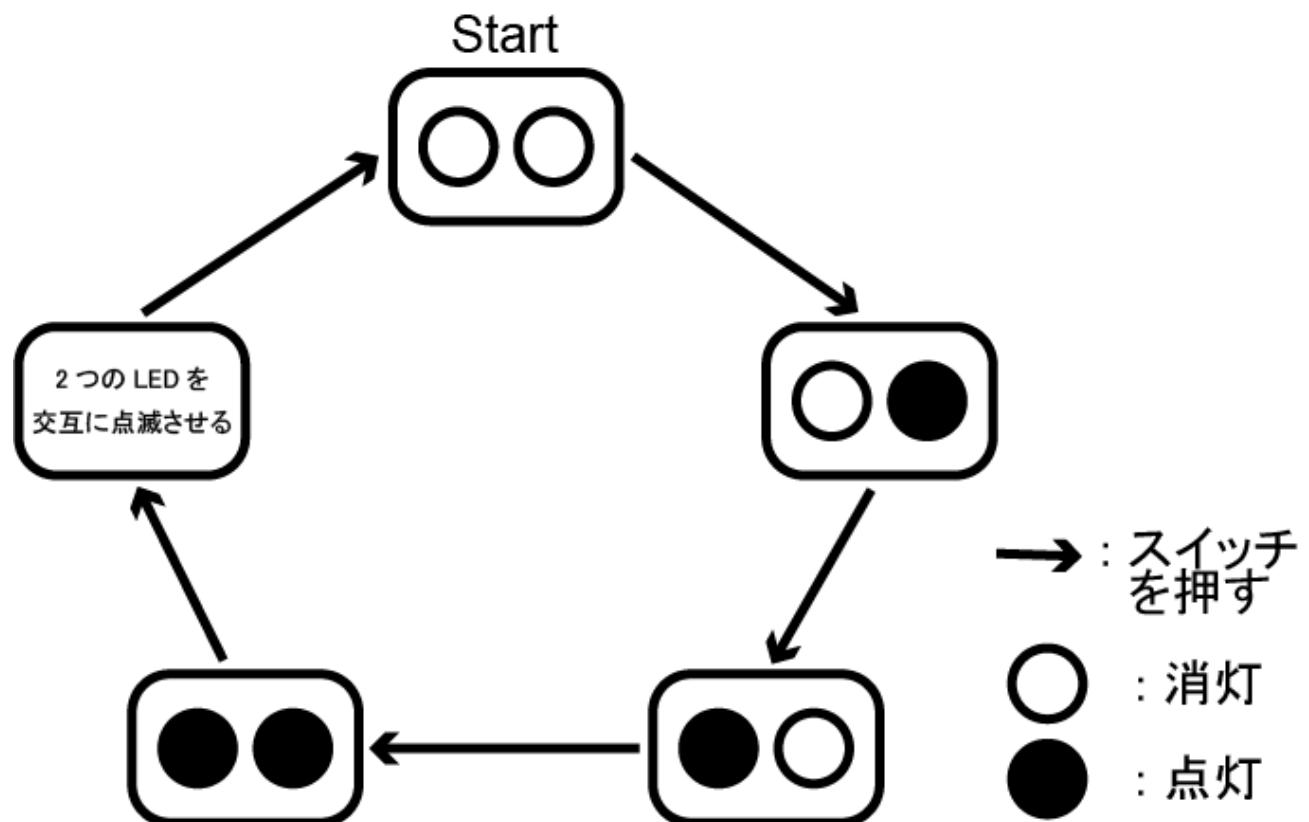
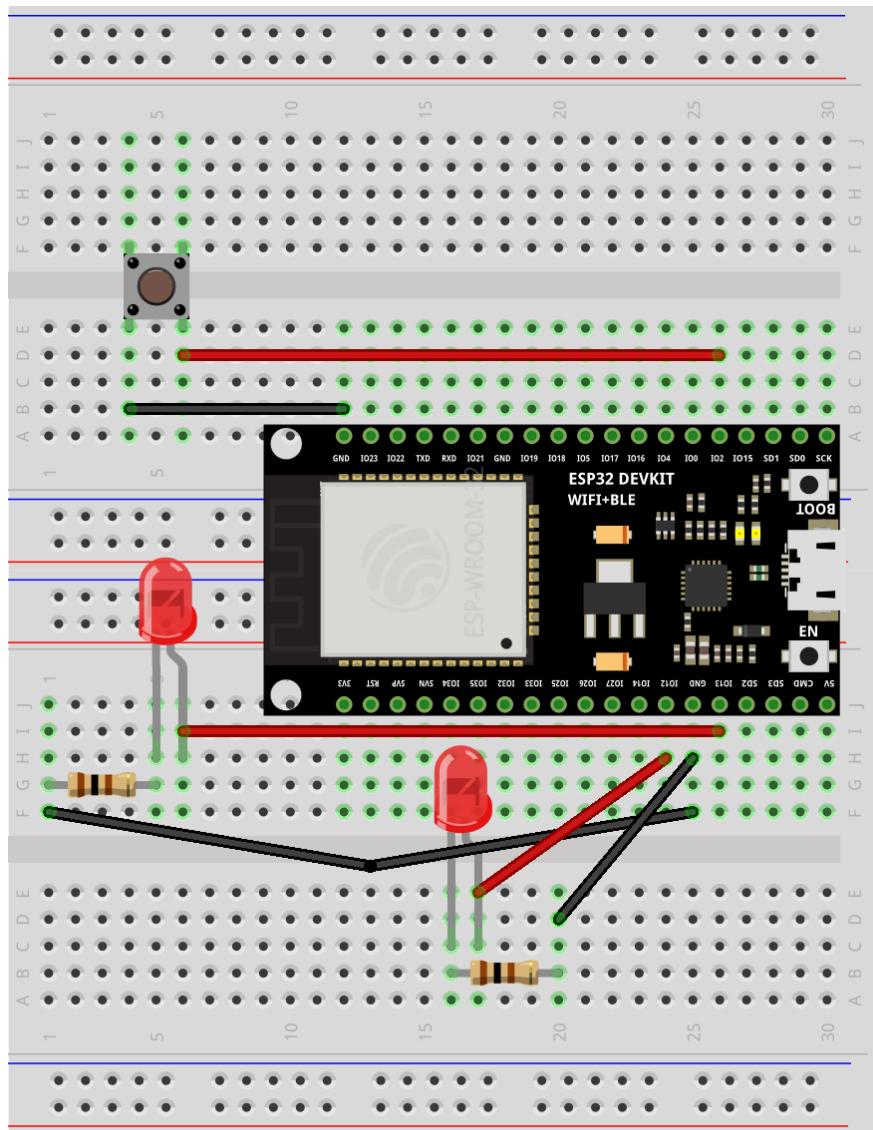


図 3.17: 状態遷移図

2019 年度組み込み制作講座より引用

応用問題解答

回路図 & 配置図



fritzing

図 3.18: 応用問題回路図

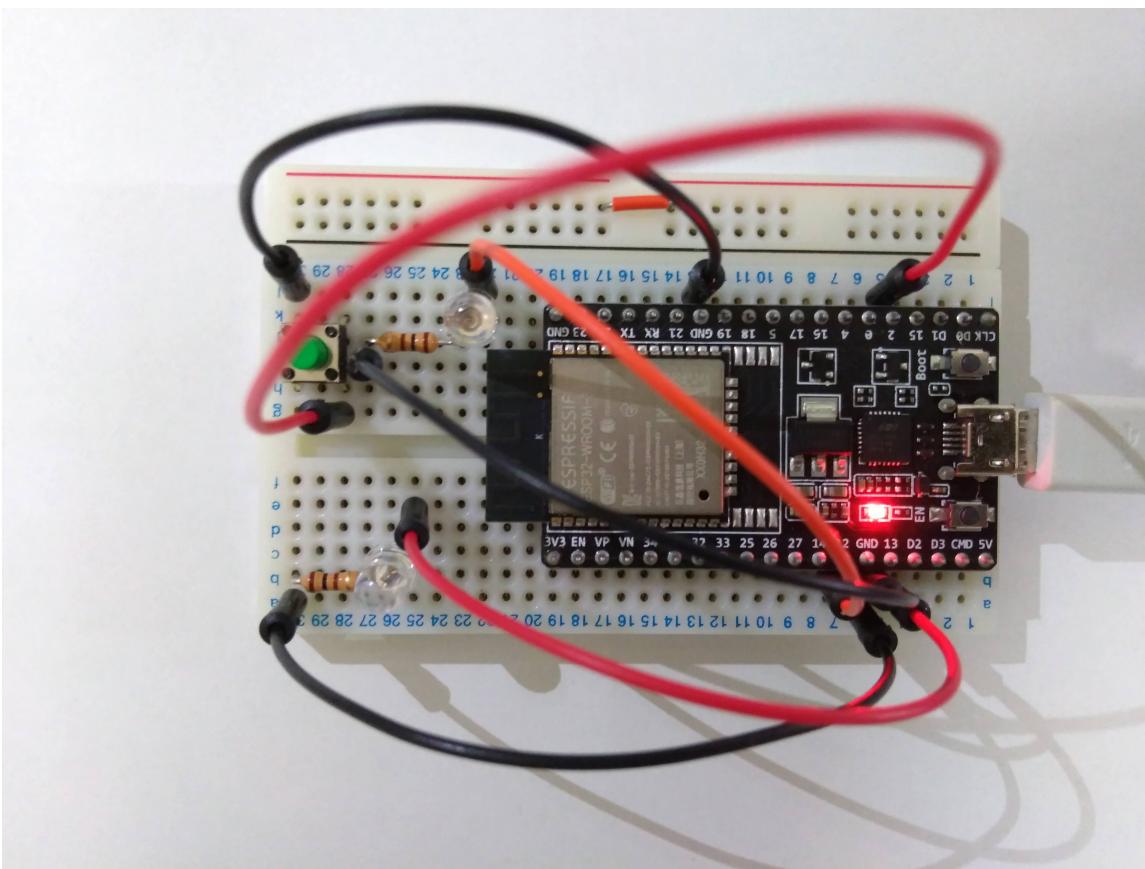


図 3.19: 応用問題回路配置図

プログラム

リスト 3.4: 状態遷移 L チカプログラム

```
int state = 0;
bool is_tica = false;

void setup() {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(2, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(2) == LOW)
    {
        delay(100);
        Serial.println("ON!");
        state = state + 1;
        Serial.println(state);
    }
}
```

```
if (state == 5) {
    state = 0;
    is_tica = false;
}
switch (state) {
    case 0:
        digitalWrite(12, LOW);
        digitalWrite(13, LOW);
        break;
    case 1:
        digitalWrite(12, HIGH);
        digitalWrite(13, LOW);
        break;
    case 2:
        digitalWrite(12, LOW);
        digitalWrite(13, HIGH);
        break;
    case 3:
        digitalWrite(12, HIGH);
        digitalWrite(13, HIGH);
        break;
    case 4:
        is_tica = true;
        break;
    default:
        break;
}
delay(200);
}
if (is_tica) {
    digitalWrite(13, HIGH);
    digitalWrite(12, LOW);
    delay(100);
    digitalWrite(13, LOW);
    digitalWrite(12, HIGH);
    delay(100);
}
}
```

第4章

センサのデータを Web 上に公開しよう

この章では IoT の定番であるセンサを使ってデータを取得します。目標として、温湿度センサで得たデータを Web 上に公開します。

4.1 センサを使おう

センサとは温度や湿度、匂いなどの様々な情報を信号化して機械が使いやすいようにしてくれるモノです。この章では、身近な温湿度を手軽に計測できる温湿度センサを使用します。

温湿度センサ

温湿度センサはその名の通り温度と湿度を計測してくれます。使用するセンサは DHT11 (図 4.1) というもので、取得した温湿度データをデジタル出力してくれます。

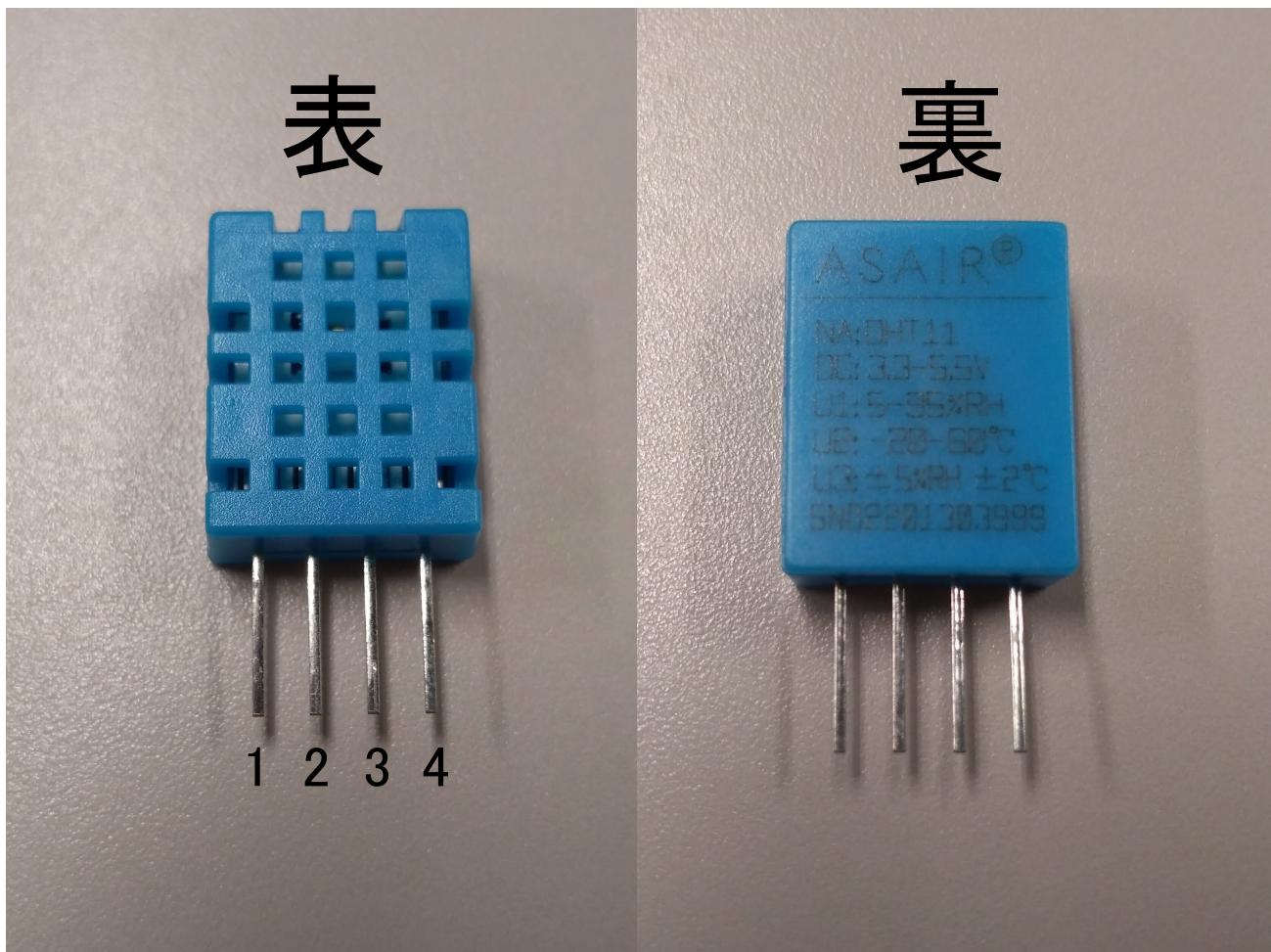


図 4.1: DHT11

DHT11 の主な仕様は以下の通りです（表 4.1）。

表 4.1: DHT11 の主な仕様

温度範囲	-20 ~ -
湿度範囲	5 ~ 95 %
サンプリング間隔	2 秒に一回

4 本あるピン（図 4.1）はそれぞれ表 4.2 のような用途で使われます。

表 4.2: DHT11 のピンについて

ピンの番号	ピンの用途
1	Vdd: 3.3 ~ 5.5V の直流を流す
2	データ出力用ピン
3	なにも接続しない
4	GND

DHT11 用ライブラリのインストール

DHT11 を ESP32 上で使うために DHT11 ライブラリを Arduino IDE にインストールします。

図 4.2 のように（スケッチ > ライブラリのインクルード > ライブラリを管理）を選択してください。

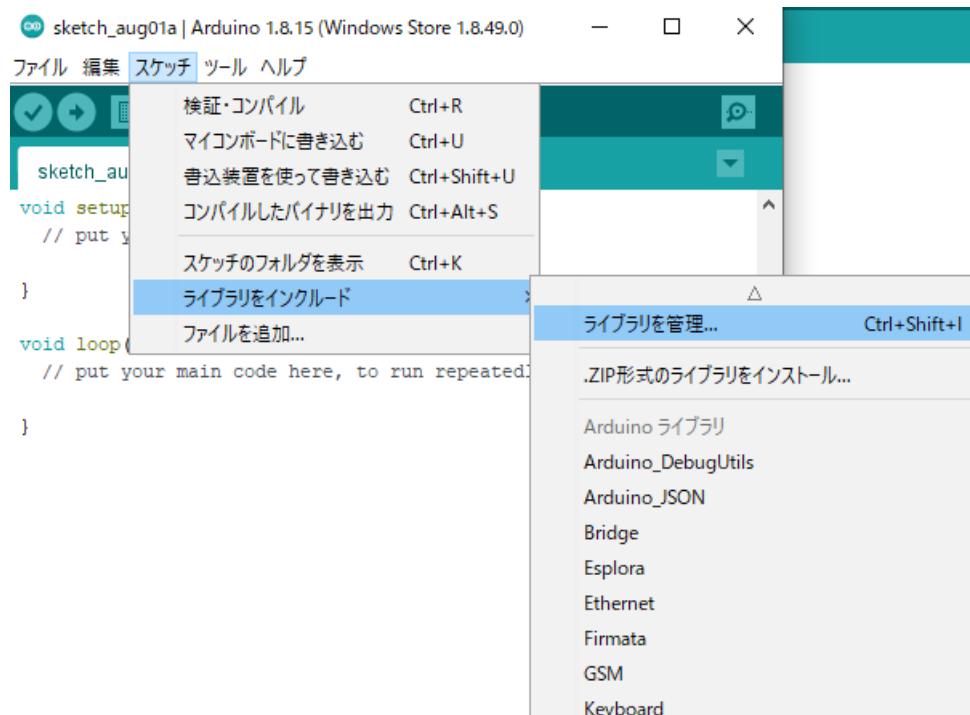


図 4.2: ライブラリの管理の選択

選択するとライブラリマネージャーが開かれるので、検索窓に「DHT11」を入力してください（図 4.3）。その後、DHT sensor library をインストールしてください。

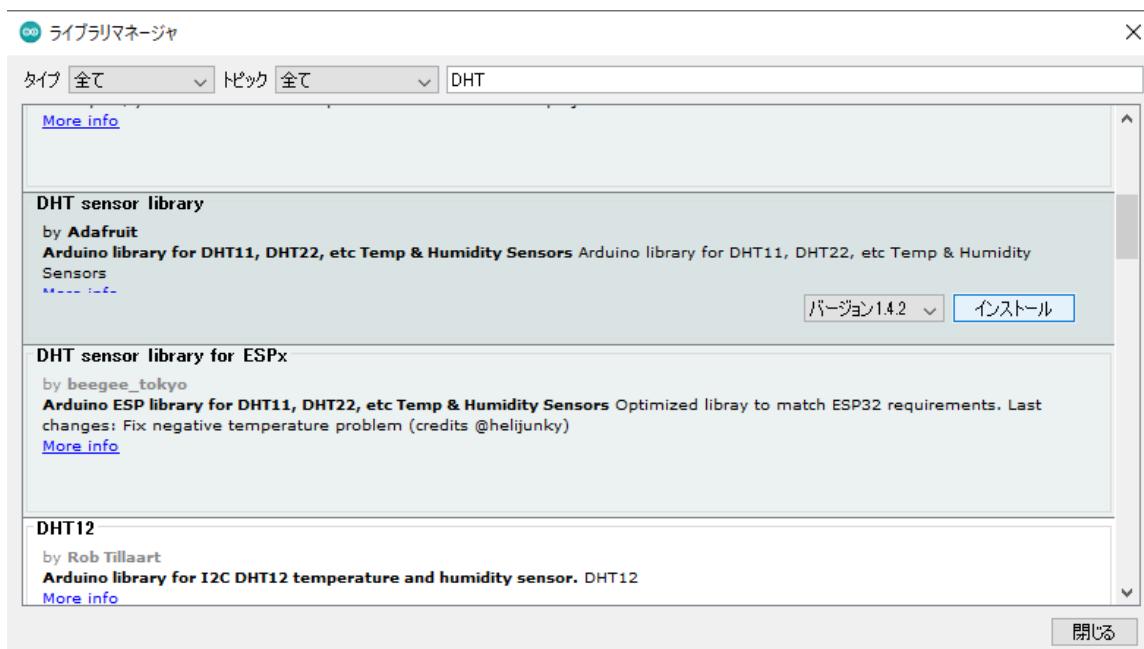


図 4.3: DHT11 用ライブラリのインストール

インストールを選択すると依存ライブラリの追加インストールについて聞かれるので
Install all を選択してください(図 4.4)。

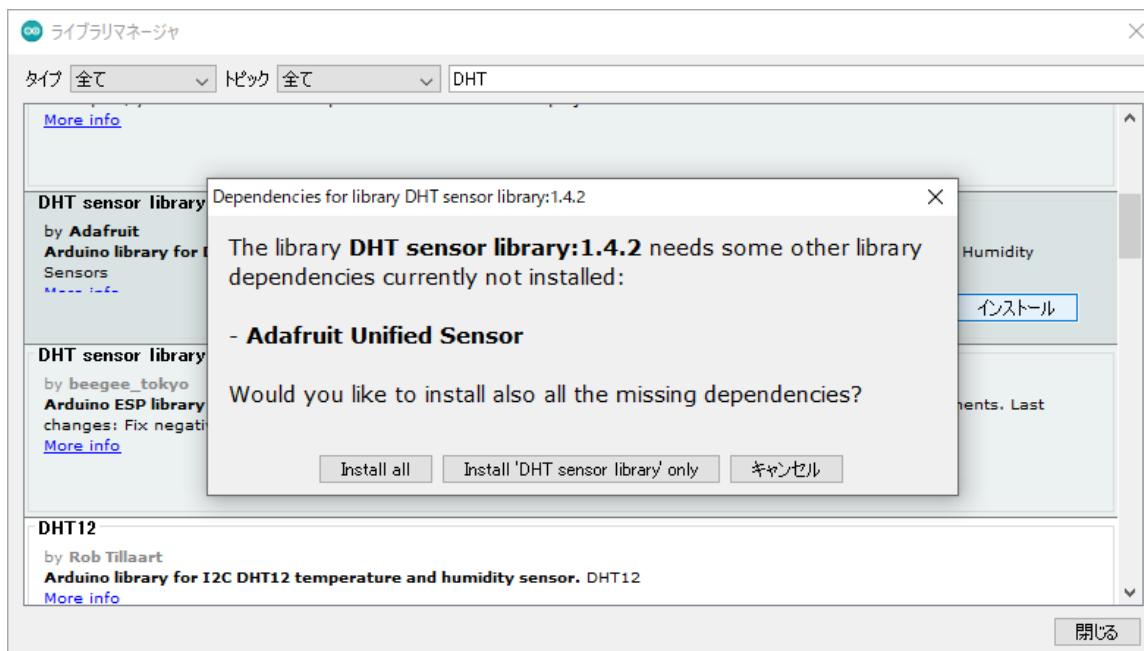
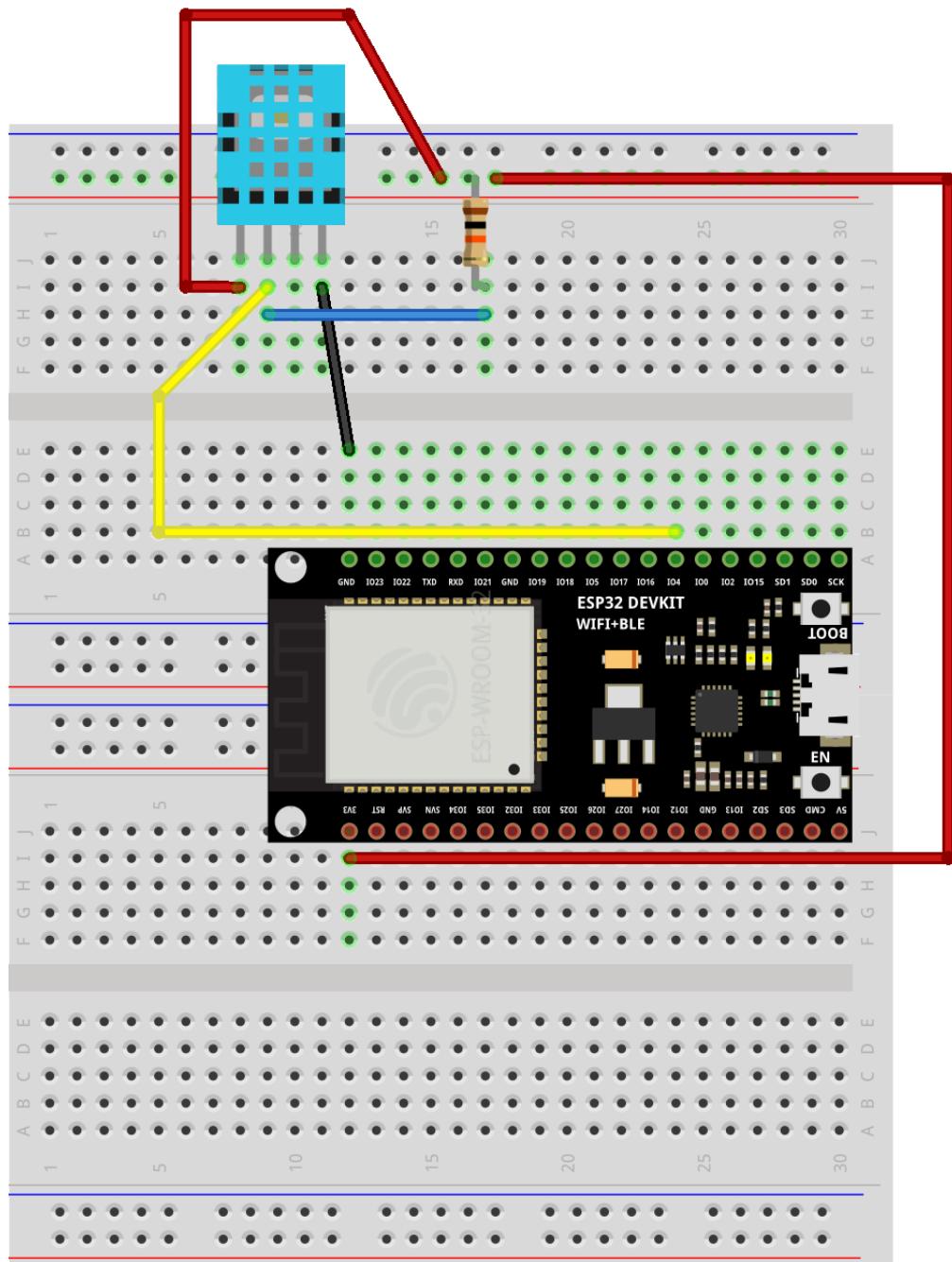


図 4.4: 依存ライブラリのインストール

DHT11 を使って温湿度を測る

ここで実際に DHT11 を使ってみましょう。図 4.5 と図 4.6 また図 4.7 を参考に回路を組んでください。抵抗は 10k Ω を使用しています。



fritzing

図 4.5: DHT11 回路図

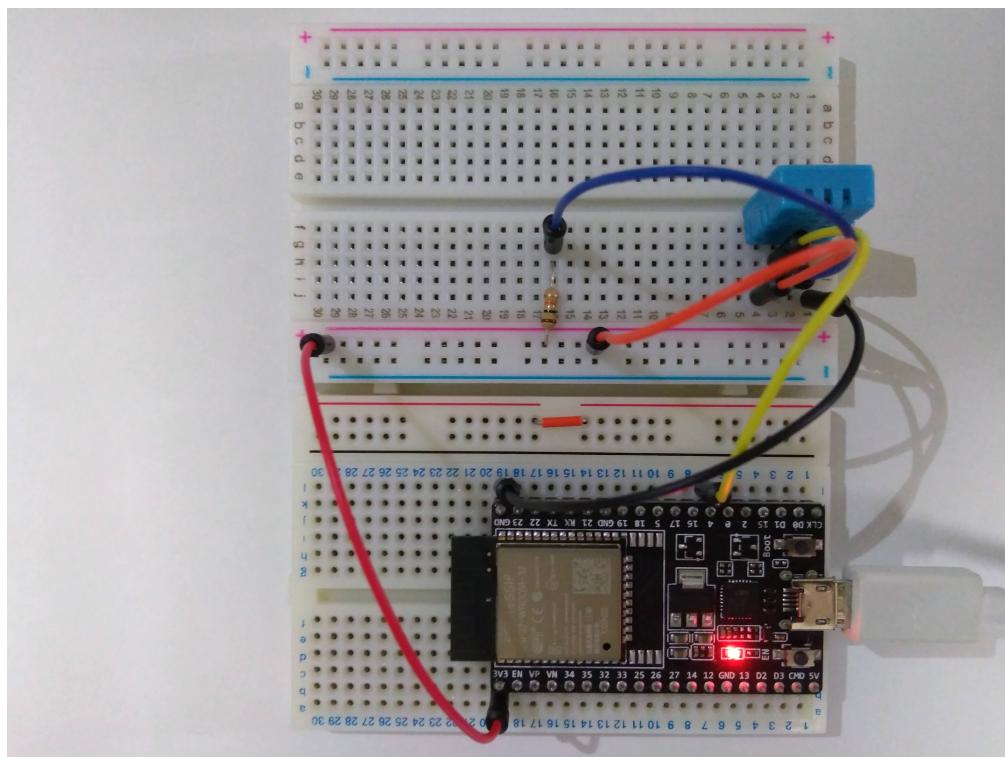


図 4.6: DHT11 回路配置図

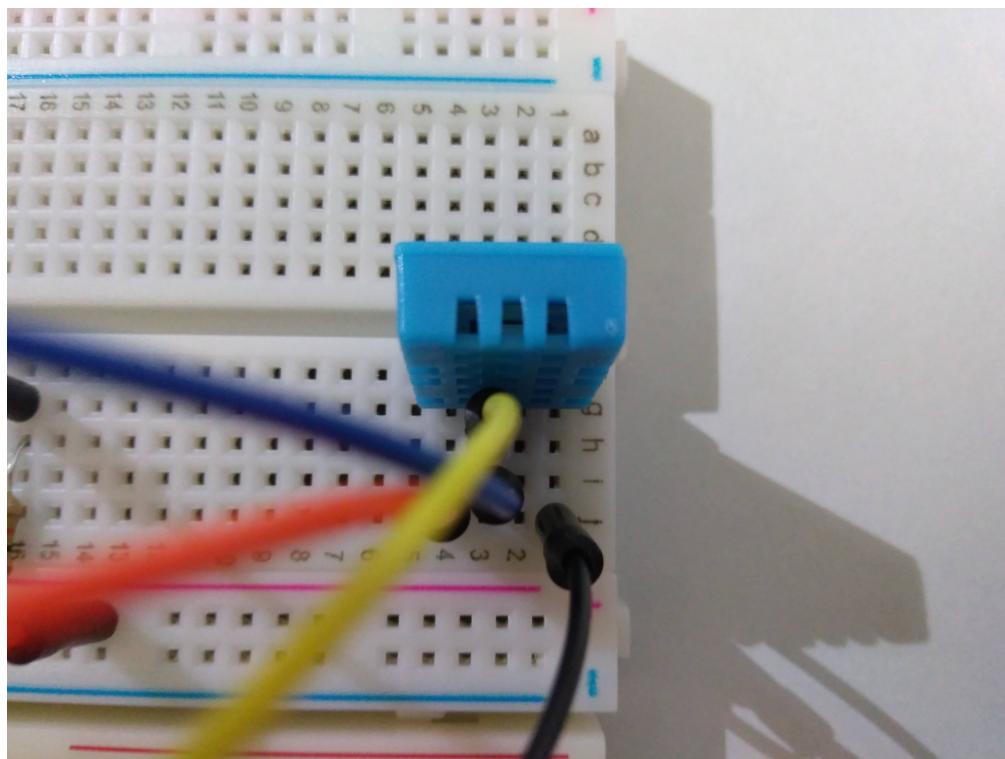


図 4.7: DHT11 アップ図

リスト 4.1: DHT11 実行プログラム

```
#include "DHT.h"
#define DHTPIN 4 // センサのデータを読み取るGPIOの番号を指定する

// DHTライブラリはDHT22/DHT11に対応しているので
// 使用するセンサを指定する
#define DHTTYPE DHT11

DHT dht11(DHTPIN, DHTTYPE); // DHT11のインスタンスを作成する

void setup() {
    Serial.begin(115200);
    dht11.begin(); // DHT11を始動させる
}

void loop() {
    // DHT11のサンプリング間隔が2秒なので
    // センサが値を読むまで2秒待機
    delay(2000);

    float humidity = dht11.readHumidity(); // 湿度取得
    float temperature = dht11.readTemperature(); // 温度取得（デフォルトでは摂氏=）

    // NaN (Not a Number) つまり数字を読み取れなかった場合再取得する
    // returnした場合loop()の最初に戻る
    if (isnan(humidity) || isnan(temperature)) {
        Serial.println("値が読み取れませんでした");
        return;
    }

    // 体感温度（湿度を含めた体感の温度指数）を計算する
    float apparent_temperature = dht11.computeHeatIndex(temperature, humidity);

    Serial.printf("温度: %.3lf\n", temperature);
    Serial.printf("湿度: %.3lf %\n", humidity);
    Serial.printf("体感温度: %.3lf\n", apparent_temperature)
}
```

プログラムの書き込みと、回路の配置に成功するとシリアルモニタにデータが送られます。「値が読み取れませんでした」が表示されている部分は試しにセンサを抜いたためです。

リスト 4.2: シリアルモニタ

```
温度: 24.00 湿度: 59.00% 体感温度: 18.87
温度: 23.80 湿度: 58.00% 体感温度: 18.61
温度: 23.80 湿度: 59.00% 体感温度: 18.65
温度: 23.80 湿度: 61.00% 体感温度: 18.75
値が読み取れませんでした
値が読み取れませんでした
値が読み取れませんでした
温度: 24.50 湿度: 83.00% 体感温度: 20.55
温度: 24.60 湿度: 75.00% 体感温度: 20.28
温度: 24.70 湿度: 71.00% 体感温度: 20.21
温度: 24.80 湿度: 67.00% 体感温度: 20.13
温度: 24.80 湿度: 64.00% 体感温度: 19.99
温度: 24.80 湿度: 62.00% 体感温度: 19.89
温度: 24.80 湿度: 61.00% 体感温度: 19.85
温度: 24.90 湿度: 59.00% 体感温度: 19.86
温度: 24.80 湿度: 58.00% 体感温度: 19.71
```

4.2 Web に公開しよう

外出時に自分の部屋の温湿度を知りたいことはありませんか？ 外部のサーバにデータを送信することで、外出時も自宅のデータを Web で見ることができます。

Wi-Fi と接続する

外部のサーバに接続するために、まず Wi-Fi に接続します。Wi-Fi に接続するために必要な情報は以下の二つです。プログラムを書き込む際に必要になるので準備をしておいてください。

- SSID (Service Set Identifier)
 - SSID とは Wi-Fi、無線 LAN の通信規格 (IEEE802.11) で定められているアクセスポイントのを識別するための名称。芝浦で言うところの「SRAS-WPA」

など。

- パスワード

- 指定したSSIDのアクセスポイントに接続する際に必要なパスワード。

Ambient

外出時に自宅のセンサのデータをWeb上で見るために、センサから取得したデータを外部のサーバに送信しますが、その外部のサーバとしてAmbientを使用します。

AmbientはIoTデータの可視化サービスです。データをグラフとして表示してくれるだけでなく、データを利用した様々なカスタマイズができます。

以下のリンクにアクセスしてAmbientのトップページに移動してください(図4.8)。

<https://ambidata.io/>

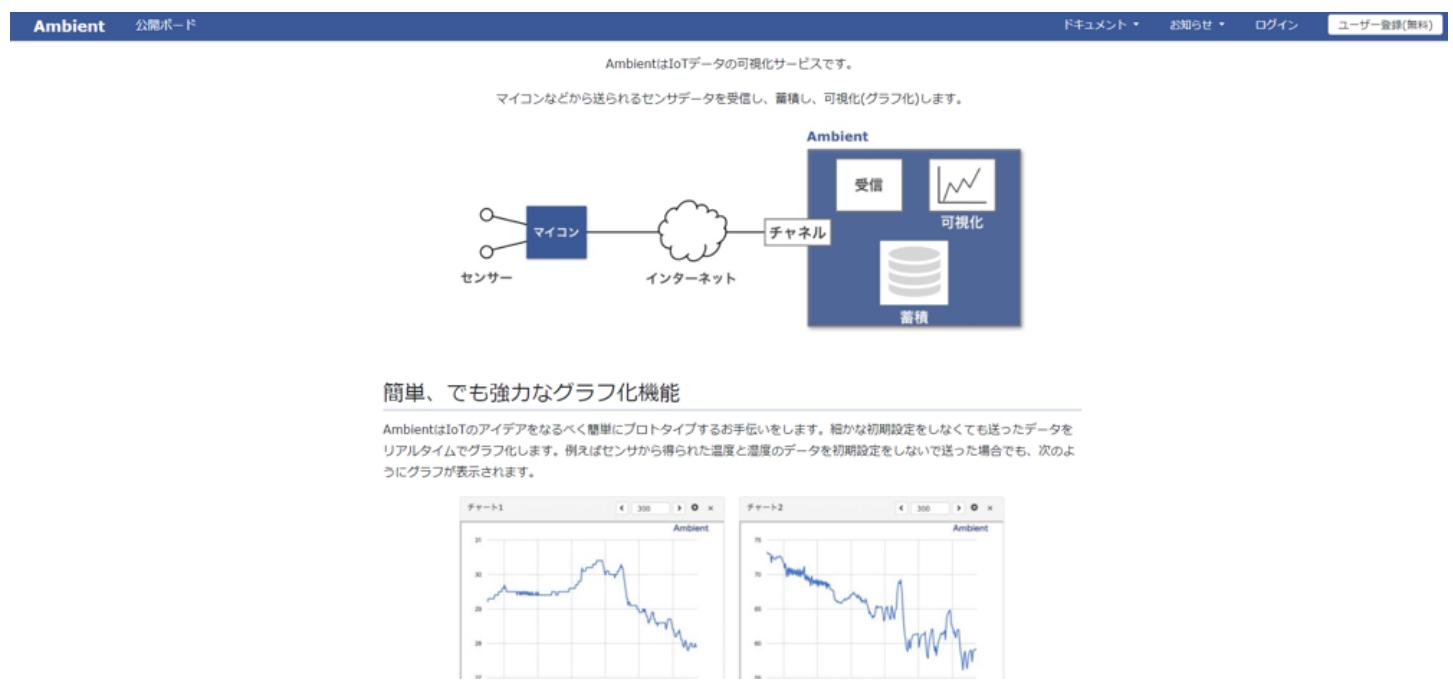


図4.8: Ambientのトップページ

Ambientを利用するためには、まずユーザ登録を行います。ユーザ登録(無料)を選択してユーザ登録画面に移動してください(図4.9)。その後、メールアドレスとパスワードを入力してユーザ登録をしてください。

第4章 センサのデータをWeb上に公開しよう 4.2 Webに公開しよう



The screenshot shows the user registration page for the Ambient platform. At the top, there is a navigation bar with links for 'Ambient' (highlighted in blue), '公開ボード', 'ドキュメント', 'お知らせ', 'ログイン', and 'ユーザー登録(無料)'. The main form consists of three input fields: 'メールアドレス' (Email Address), 'パスワード' (Password), and 'パスワード再入力' (Re-enter Password). Below these fields is a blue button labeled 'ユーザー登録(無料)' with the text '登録した時点で「Ambient利用規約」に書かれた内容に同意したものとします。' (By registering, you agree to the terms of service). At the bottom of the page, there are links for 'AmbientData Inc.', '利用規約', and '会社概要'.

図 4.9: ユーザ登録

ユーザ登録をすると登録したメールアドレスに登録完了メールが届きます（図 4.10）。このメールに添付してあるリンクにアクセスすることユーザ登録完了です。



図 4.10: 登録完了メール

ユーザ登録完了後、ログイン画面にアクセスしログインしてください（図4.11）。



図4.11: ログイン画面

ログイン後、**チャネルを作る**を選択し、チャネルを作成することでAmbientの設定は完了です。



図4.12: チャネル作成完了画面

プログラムを書き込む際に必要な情報として以下の二つがあるので、メモをしておいてください。

- チャネルID
- ライトキー

ライブラリのインストール

AmbientをESP32上で使うためにAmbientライブラリをArduino IDEにインストールします。

図4.2のように(スケッチ>ライブラリのインクルード>ライブラリを管理)を選択してください

ライブラリマネージャーの検索窓に「Ambient」と入力し、候補に出てくる「Ambient ESP32 ESP8266 lib」をインストールしてください。

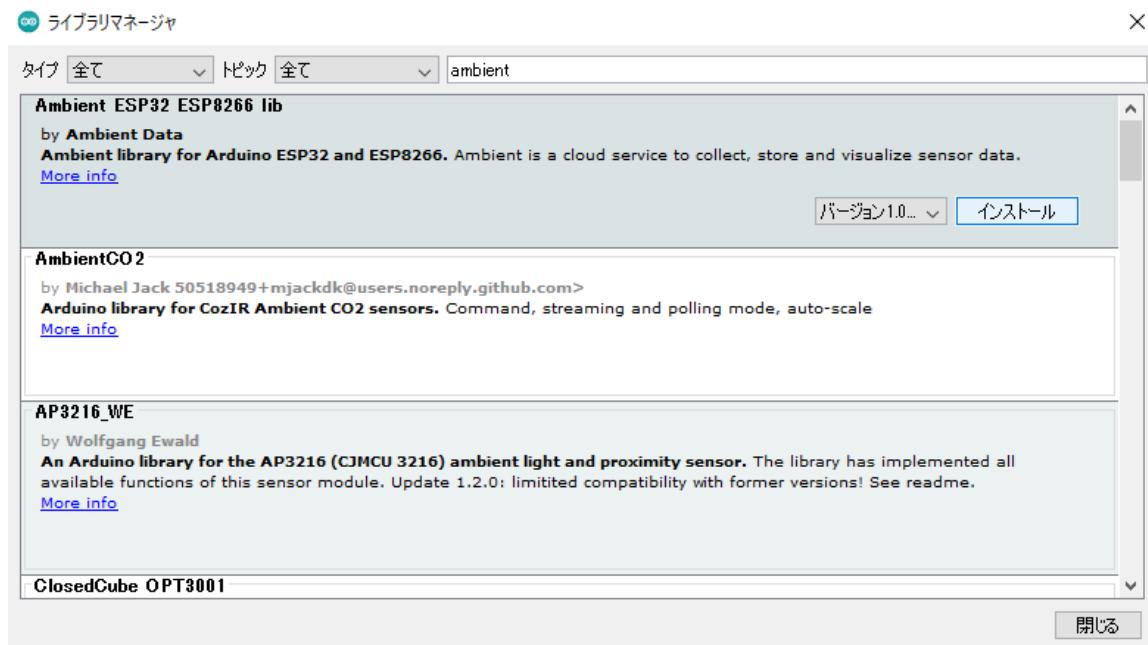


図 4.13: ambient 用ライブラリのインストール

Ambientにデータを送る

ここで実際に Ambient にデータを送ります。DHT11 を使用するので回路図は図 4.6 を利用してください。プログラムはリスト 4.3 を書き込んでください。各々の環境に合わせて変数を書き換える必要があります。以下の変数を書き換えてください。

- SSID
 - 変数名: ssid
- パスワード
 - 変数名: password
- チャネル ID
 - 変数名: channel_id
- ライトキー
 - 変数名: write_key

リスト4.3: Ambient利用プログラム

```
#include <WiFi.h>
#include "DHT.h"
#include "Ambient.h"
#define DHTPIN 4 // センサのデータを読み取るGPIOの番号を指定する

// DHTライブラリはDHT22/DHT11に対応しているので
// 使用するセンサを指定する
#define DHTTYPE DHT11

// Ambient用変数
unsigned int channel_id = 111111;
const char *write_key = "b7471121723ae295";

// WiFi接続用変数
const char *ssid = "elecom-b3506f-g";
const char *password = "*****";

DHT dht11(DHTPIN, DHTTYPE); // DHT11のインスタンスを作成する
Ambient ambient; // Ambientのインスタンスを作成する
WiFiClient wifi_client; // Ambientに接続するためのクライアントを用意

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password); // Wi-Fi接続開始

    while (WiFi.status() != WL_CONNECTED) // Wi-Fiアクセスポイントへ接続するまで待機
    {
        Serial.println("Waiting for Wi-Fi connection....");
        delay(500);
    }
    Serial.println("Connected to Wi-Fi");
    dht11.begin(); // DHT11を始動させる
    ambient.begin(channel_id, write_key, &wifi_client);
}

void loop() {
    // DHT11のサンプリング間隔は2秒ですが
    // Amibentのデータ送信間隔は最低でも5秒間隔を開ける
    // 必要があるので30秒待機
    delay(30000);

    float humidity = dht11.readHumidity(); // 湿度取得
    float temperature = dht11.readTemperature(); // 温度取得（デフォルトでは摂氏=）
```

```

// NaN (Not a Number) つまり数字を読み取れなかった場合再取得する
// returnした場合loop()の最初に戻る
if (isnan(humidity) || isnan(temperature)) {
    Serial.println("値が読み取れませんでした");
    return;
}

// 体感温度（湿度を含めた体感の温度指数）を計算する
float apparent_temperature = dht11.computeHeatIndex(temperature, humidity);

Serial.printf("温度: %.3lf\n", temperature);
Serial.printf("湿度: %.3lf %\n", humidity);
Serial.printf("体感温度: %.3lf\n", apparent_temperature)

ambient.set(1, temperature); // チャート1に温度データ登録
ambient.set(2, humidity); // チャート2に湿度データ登録
ambient.set(3, apparent_temperature); // チャート3に体感温度データ登録

ambient.send(); // 登録データ送信
Serial.println("Ambientにデータを送信しました");
}

```

プログラムの実行に成功するとシリアルモニタに以下のように表示されます（リスト4.4）。Wi-Fiのコネクションが完了した後、5秒ごとにDHT11より取得したデータをAmbientに送信します。

リスト4.4: シリアルモニタ画面

```

Waiting for Wi-Fi connection....
Connected to Wi-Fi
温度: 24.70 湿度: 63.00% 体感温度: 19.83
Ambientにデータを送信しました
温度: 24.70 湿度: 61.00% 体感温度: 19.74
Ambientにデータを送信しました
温度: 24.70 湿度: 61.00% 体感温度: 19.74
Ambientにデータを送信しました

```

```
温度: 24.60 湿度: 61.00% 体感温度: 19.63
Ambientにデータを送信しました
温度: 24.50 湿度: 61.00% 体感温度: 19.52
Ambientにデータを送信しました
温度: 24.40 湿度: 61.00% 体感温度: 19.41
Ambientにデータを送信しました
温度: 24.40 湿度: 61.00% 体感温度: 19.41
Ambientにデータを送信しました
```

Ambientとの通信に成功していると図4.14のように表示されます。ただし図4.14はプログラムを開始してから数分経過後のグラフです。

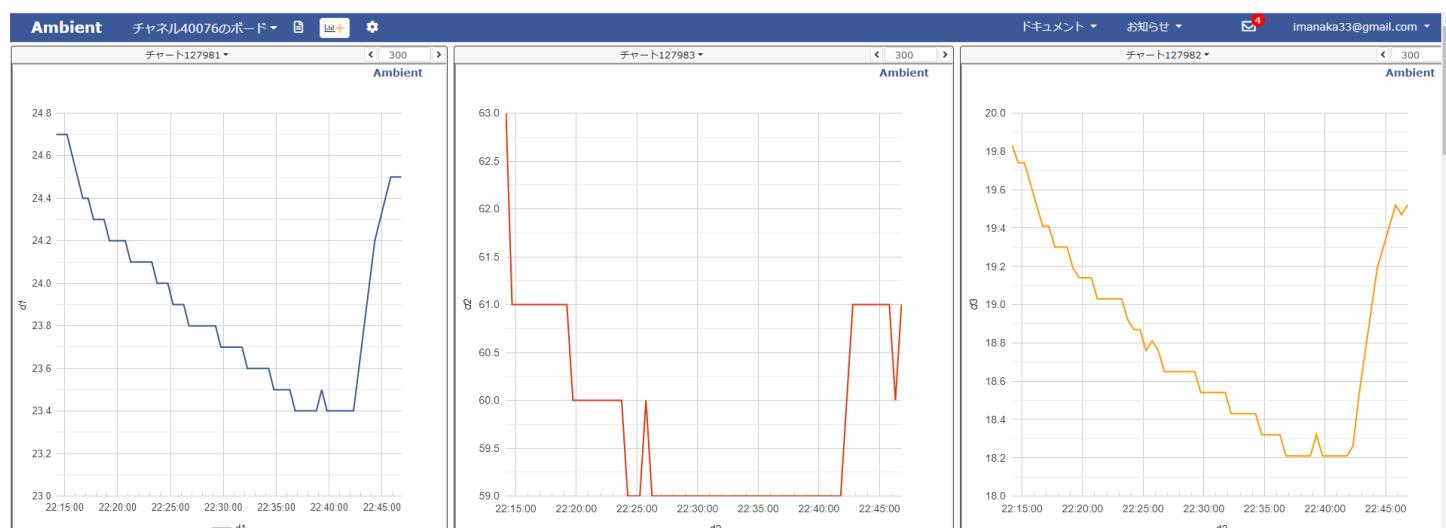
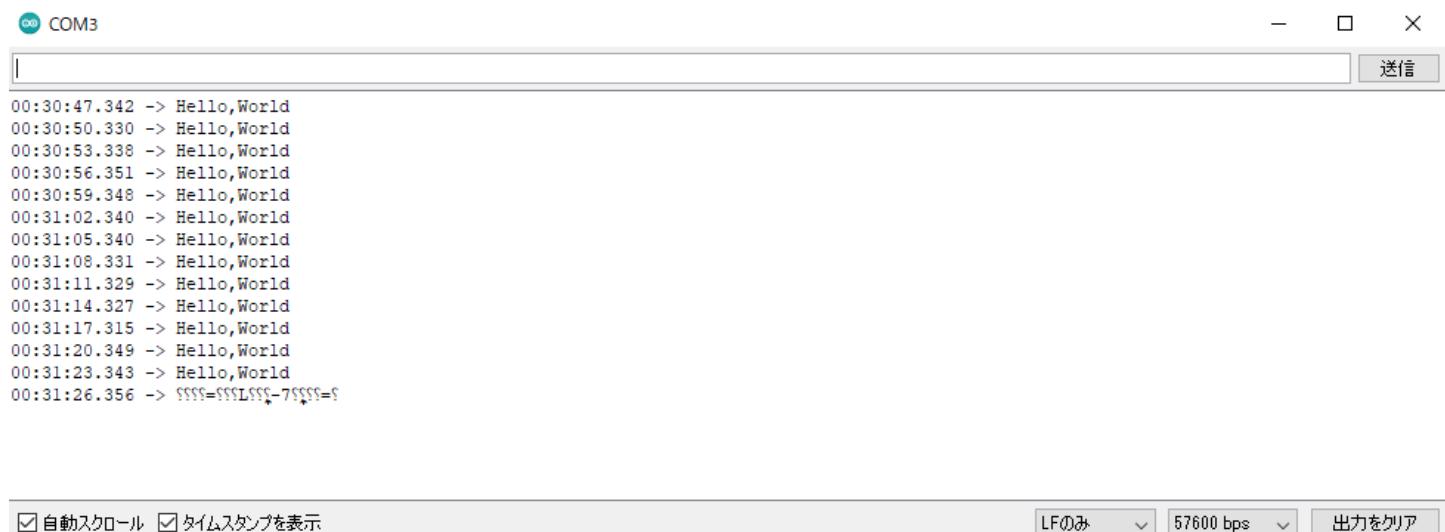


図4.14: Ambientに表示されるグラフ

付録 A

トラブルシューティング

A.1 シリアルモニタで文字化けがする



```
00:30:47.342 -> Hello,World
00:30:50.330 -> Hello,World
00:30:53.338 -> Hello,World
00:30:56.351 -> Hello,World
00:30:59.348 -> Hello,World
00:31:02.340 -> Hello,World
00:31:05.340 -> Hello,World
00:31:08.331 -> Hello,World
00:31:11.329 -> Hello,World
00:31:14.327 -> Hello,World
00:31:17.315 -> Hello,World
00:31:20.349 -> Hello,World
00:31:23.343 -> Hello,World
00:31:26.356 -> ?????=????L????-7????=?
```

□ 送信

自動スクロール タイムスタンプを表示

LFのみ 57600 bps 出力をクリア

図 A.1: 1

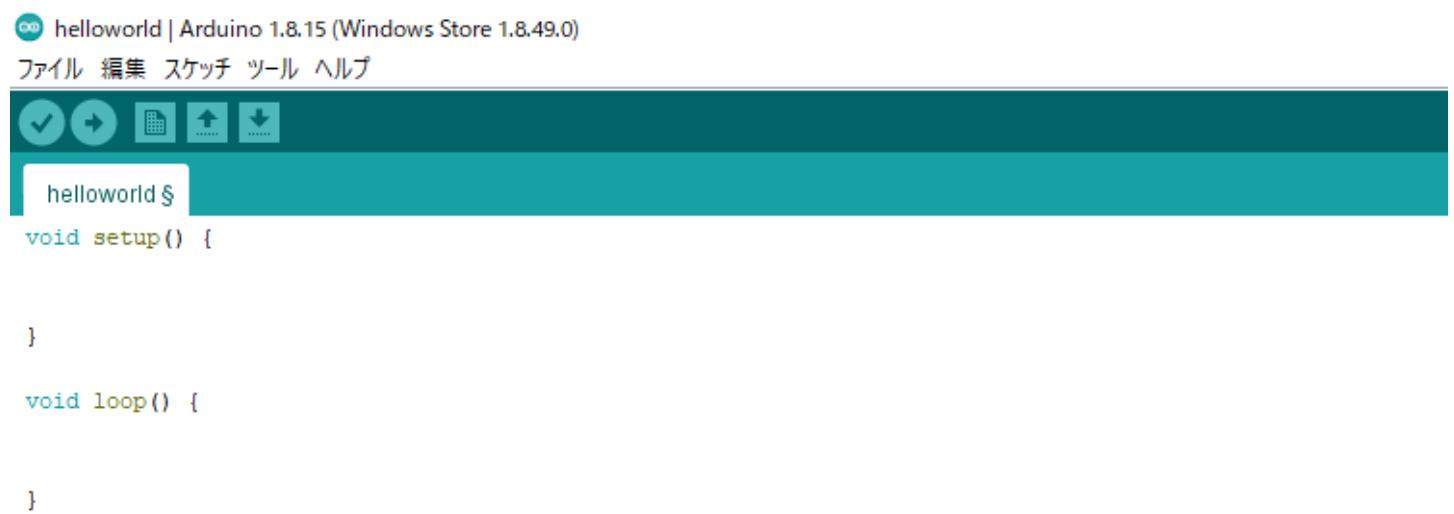
```
00:30:47.342 -> Hello,World  
00:30:50.330 -> Hello,World  
00:30:53.338 -> Hello,World  
00:30:56.351 -> Hello,World  
00:30:59.348 -> Hello,World  
00:31:02.340 -> Hello,World  
00:31:05.340 -> Hello,World  
00:31:08.331 -> Hello,World  
00:31:11.329 -> Hello,World  
00:31:14.327 -> Hello,World  
00:31:17.315 -> Hello,World  
00:31:20.349 -> Hello,World  
00:31:23.343 -> Hello,World  
00:31:26.356 -> ?????=????L?=?-7????=????L?=?????=????L?=?????=Hello,World  
00:31:53.335 -> Hello,World  
00:31:56.328 -> Hello,World
```

図 A.2: 2

Upload speed が間違っている可能性がある

A.2 プログラムが書き込めない

シリアルポートが間違っているかもしれない



The screenshot shows the Arduino IDE interface. The title bar reads "helloworld | Arduino 1.8.15 (Windows Store 1.8.49.0)". The menu bar includes "ファイル" (File), "編集" (Edit), "スケッチ" (Sketch), "ツール" (Tools), and "ヘルプ" (Help). Below the menu is a toolbar with icons for save, undo, redo, and upload. The code editor window contains the following code:

```
helloworld §  
void setup() {  
  
}  
  
void loop() {  
  
}
```

図 A.3: 3

プログラムの保存を忘れている Ctrl+S で保存してから読み込む

A.4 error: redefinition

```
c:\Users\donki\Documents\Arduino\helloworld\DHT11.ino: In function  
'void setup()': DHT11:30:6: error: redefinition of 'void setup()'  
void setup() {
```

```
c:\Users\donki\Documents\Arduino\helloworld\helloworld.ino:1:6:  
note: 'void setup()' previously defined here  
void setup() {  
^  
  
c:\Users\donki\Documents\Arduino\helloworld\DHT11.ino: In function  
'void loop()': DHT11:37:6: error: redefinition of 'void loop()'  
void loop() {  
^  
  
c:\Users\donki\Documents\Arduino\helloworld\helloworld.ino:5:6:  
note: 'void loop()' previously defined here  
void loop() {  
^  
  
exit status 1
```

- 解決法 Arduino コンパイルエラー (redefinition) 同じフォルダ内に setup() と loop() が重複している際に出るエラー Arduino はコンパイルをファルダ単位で行うため、このようなエラーが出る

A.5 接続ポートに ESP32 が反映されない

デバイスマネージャーに ESP32 の接続ポートが表示されない場合はデバイスドライバをインストールする必要があります。以下のリンクにアクセスしてください。<https://jp.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

付録 A トラブルシューティングA.5 接続ポートに ESP32 が反映されない



図 A.4: 3

付録 A トラブルシューティングA.5 接続ポートに ESP32 が反映されない

VCP ドライバのダウンロードとインストール

Windows、マックintosh、Linux を以下からダウンロードします。

*注：ドライバの Linux 3.x.x および 4.x.x バージョンは、www.kernel.org 内の最新 Linux 3.x.x および 4.x.x ツリー内に格納されています。

レガシ OS ソフトウェアバージョン

ドライバ・パッケージのダウンロード・リンクとサポート情報

ソフトウェア・ダウンロード

ソフトウェア (11)

ソフトウェア · 11

CP210x Universal Windows Driver	v10.1.10	1/13/2021
CP210x VCP Mac OSX Driver	v6.0.1	4/1/2021
CP210x VCP Windows	v6.7	9/4/2020
CP210x Windows Drivers	v6.7.6	9/4/2020
CP210x Windows Drivers with Serial Enumerator	v6.7.6	9/4/2020

その他 6 個を表示 ソフトウェア

シリアル列挙ドライバ

シリアル列挙ドライバとは、そしてなぜ必要なのか？

図 A.5: 3

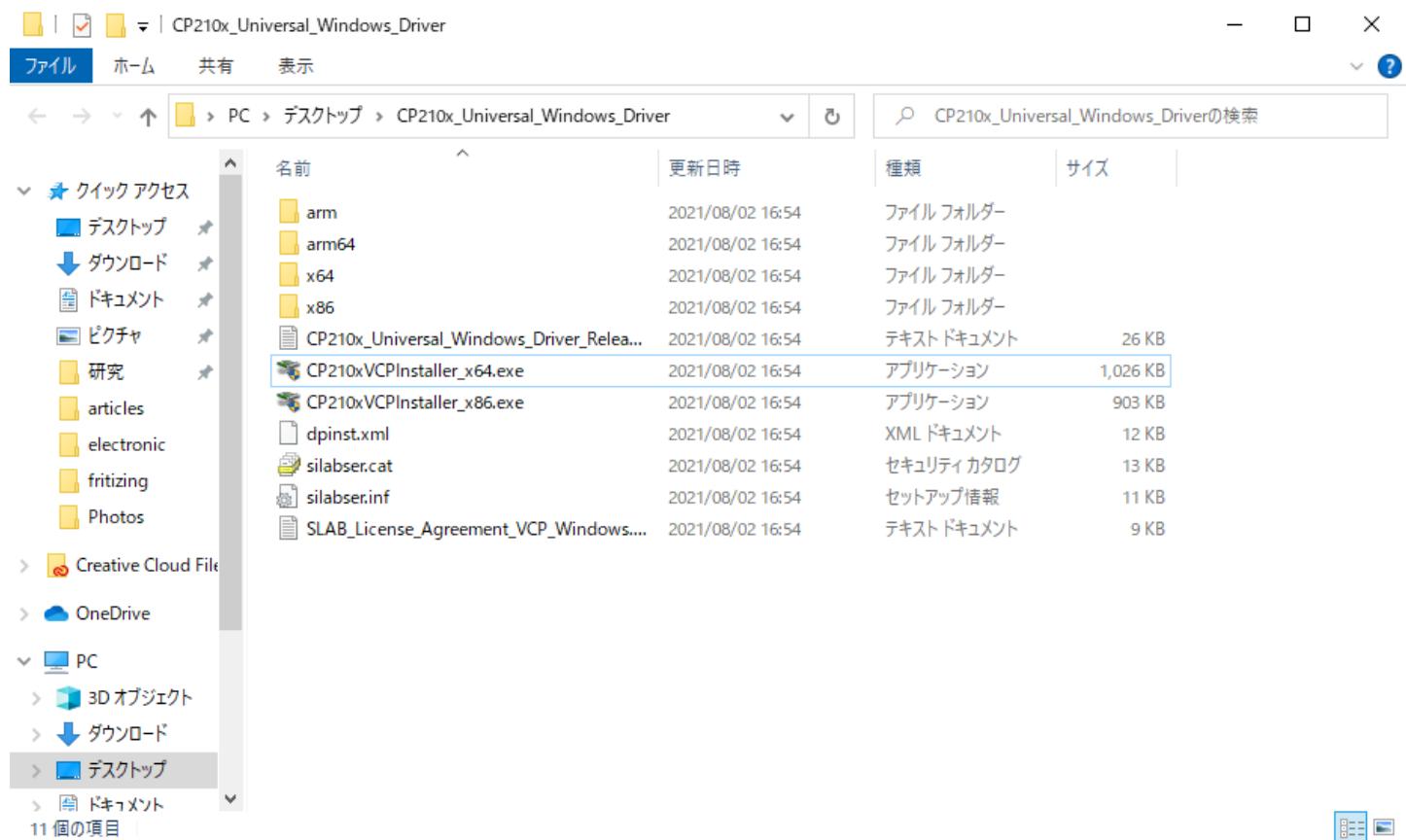


図 A.6: 3

付録 A トラブルシューティング COM ポートがデバイスマネージャーに表示されない

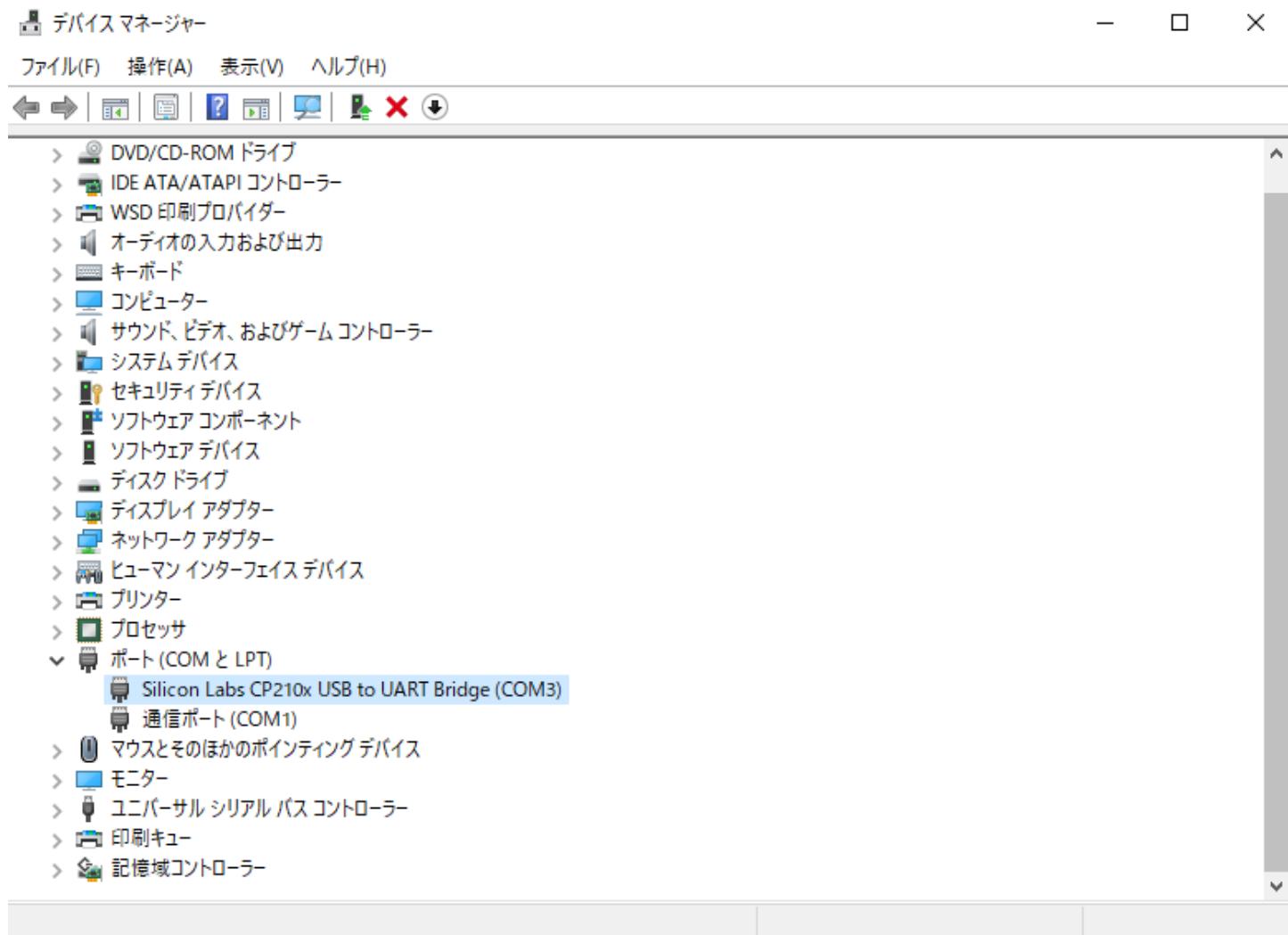


図 A.7: 3

A.6 COM ポートがデバイスマネージャーに表示されない

USB ケーブルに問題があるかもデバイスドライバの更新がうまくいっていない

A.7 うまく書き込めない

シリアルモニタがついているといけないデバイスドライバを更新する必要がある

A.8 LED の光り方が弱い

抵抗の大きさが違う可能性あり

A.9 回路図どうりなのにつかない

ジャンプワイヤがつかない可能性あり

著者紹介

THEToilet / @THEToilet

あとがきみたいなのにあこがれています。

執筆協力 / @raimu

少し校閲しただけで名前が載りました。

ESP32ではじめる初めてのIoT

2021年7月12日 初版第1刷 発行

著 者 THEToilet
