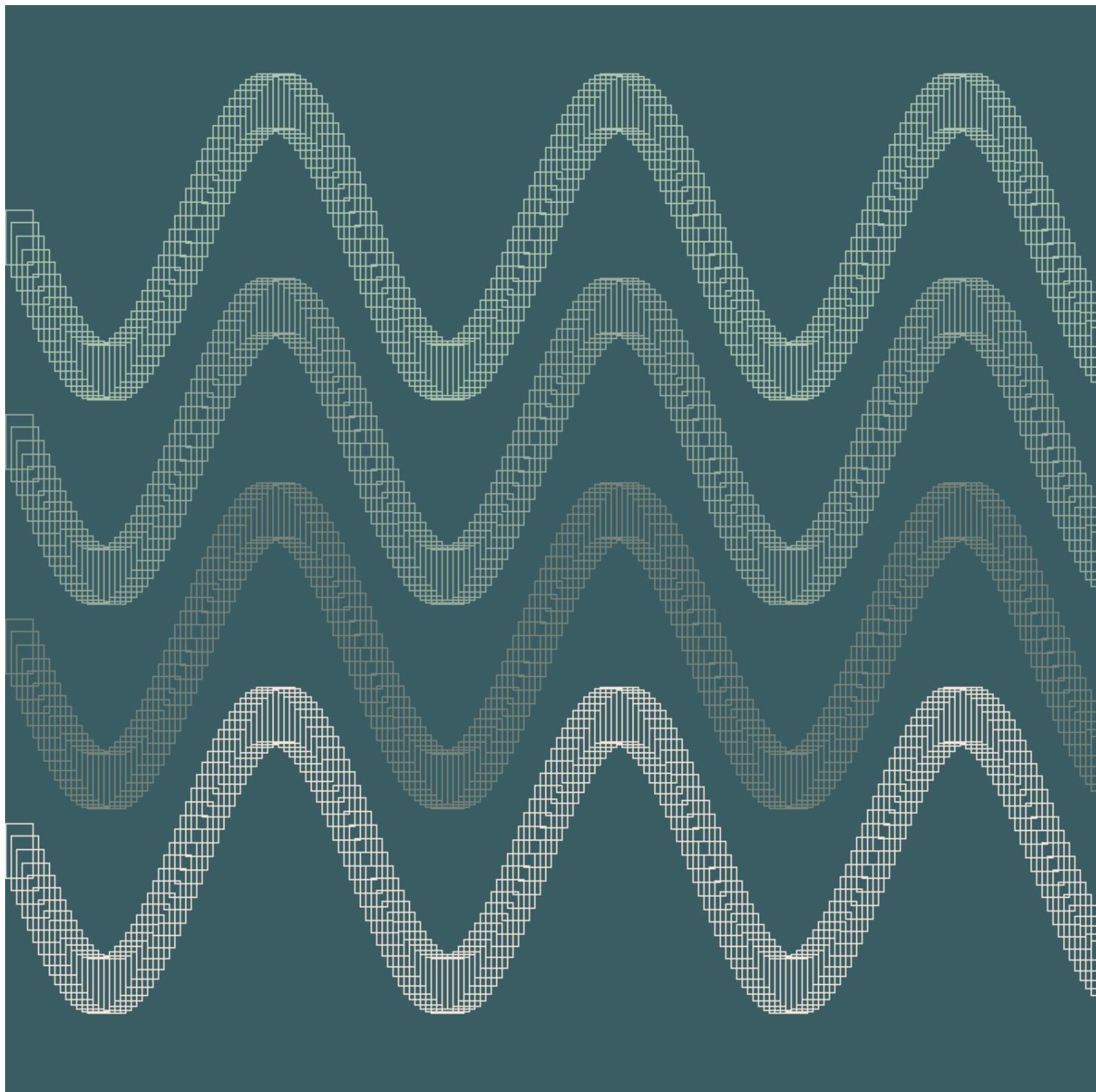


ESP32ではじめる初めてのIoT

First IoT with ESP32



電子計算機研究会

ESP32 ではじめる初めての IoT

THEToilet、raimu 著

2021-08-12 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。IoT (Internet of Things) はインターネットにすべてを繋ぎま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

^{*1} 電子計算機研究会 HP <http://den3.net>

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

前提とする知識

- 何らかのプログラム言語の基礎知識

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	iii
前提とする知識	iii
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	6
2.1　　ESP32 とは	6
2.2　　ESP32 の開発環境	7
2.3　　Arduino IDE のインストール	7
2.4　　ESP32 用ボードマネージャーのインストール	13
2.5　　Hello ESP32!!	19
シリアル通信とは	32
第3章 電子部品を使ってみよう	35
3.1　　部品説明	35
3.2　　L チカしよう！	46

目次

チャタリング	55
3.3 応用問題: 状態遷移	56
第 4 章 センサのデータを Web 上に公開しよう	62
4.1 センサを使おう	62
4.2 Web に公開しよう	72
第 5 章 WebAPI を使おう	82
5.1 Weather API を使う	82
5.2 ディスプレイを使う	105
5.3 Weather API から得たデータを表示	112
5.4 応用 1: センサのデータを表示する	117
5.5 応用 2: 時計を表示してみる	123
第 6 章 応用編	127
6.1 Web サーバからの操作	127
6.2 ESP32 でアクセスポイントを設定	138
付録 A トラブルシューティング	143
A.1シリアルモニタで文字化けがする	143
A.2プログラムが書き込めない	144
A.3プログラムが反映されない	145
A.4error: redefinion	145
A.5接続ポートに ESP32 がない	146
A.6COM ポートが表示されない	149
A.7うまく書き込めない	149
A.8LED の光り方が弱い	150
A.9回路図どうりなのにつかない	150
A.10Failed to execute script esptool	150

目次

参考文献	151
著者紹介	152

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: 必要な電子部品

品名	個数	参考価格	画像
ESP32DevKitC	1 個	1230 円	
microUSB Type-B	1 本	約 300 円	
ブレッドボード	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	

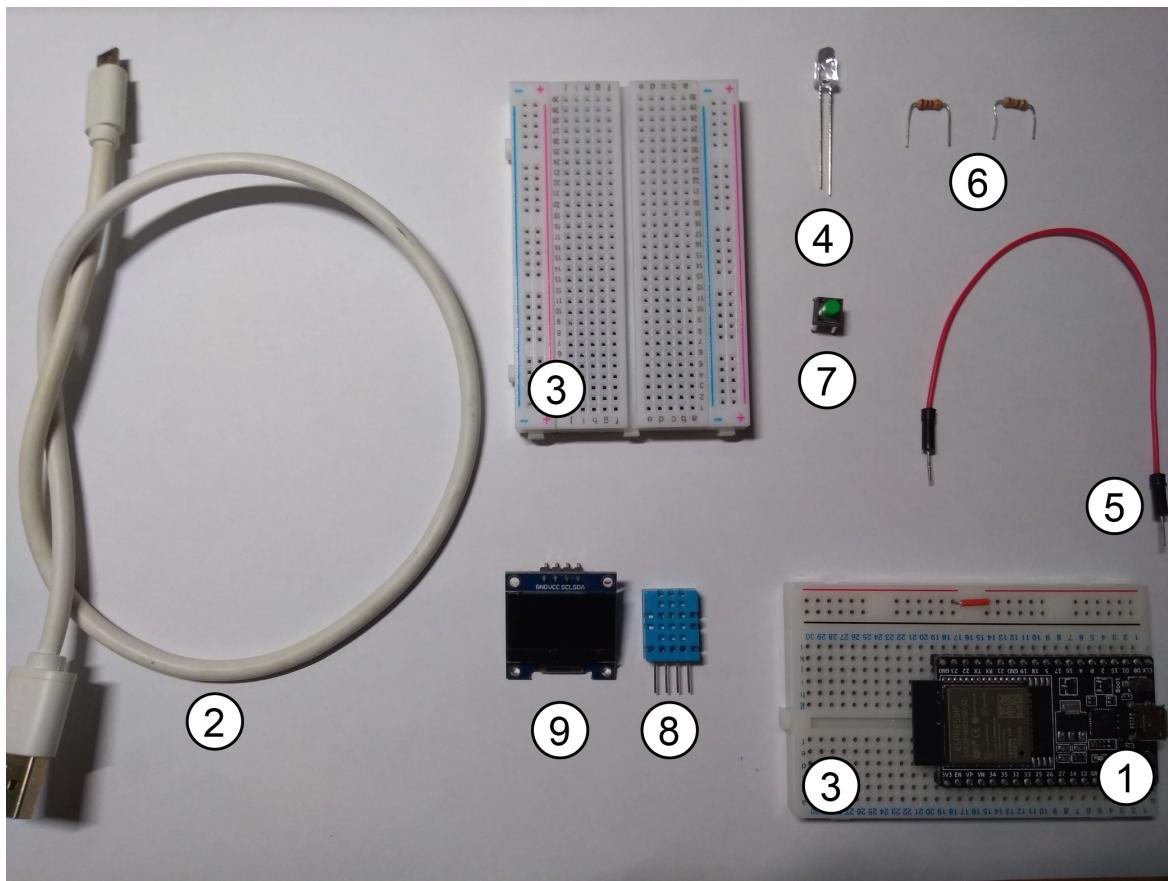


図 1.1: 電子部品一覧

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

E S P 3 2 - D e v K i t C - 3 2 E E S P 3 2 - W R O O M - 3 2
E 開発ボード 4 M B

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

プレッドボード

プレッドボード 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穴のプ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)

プレッドボード・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。

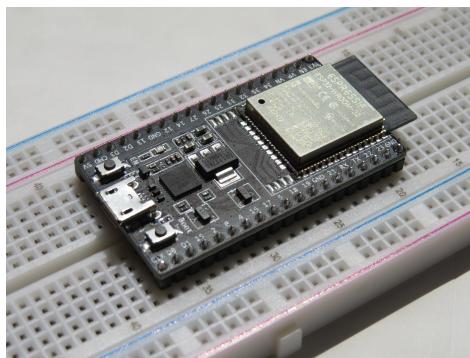


図 2.1: ESP32DevKitC-32E

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。

Arduino IDE Download: <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。

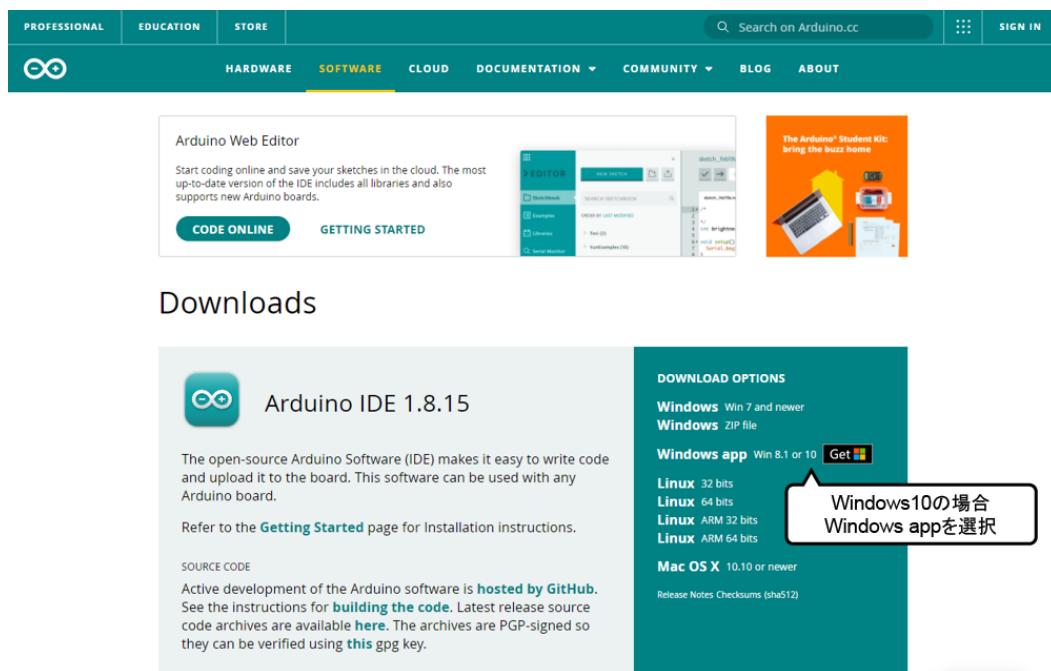


図 2.2: ArduinoIDE のダウンロード画面

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。

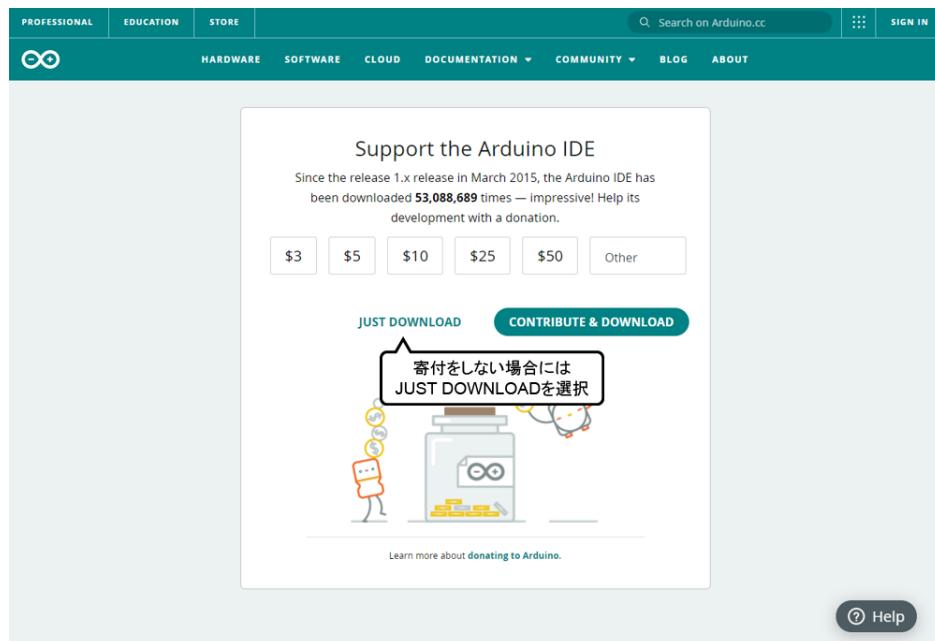


図 2.3: 寄付金の金額選択画面

JUST DOWNLOAD を選択するとブラウザ内で MicorsoftStore 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 MicorsoftStore

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）。

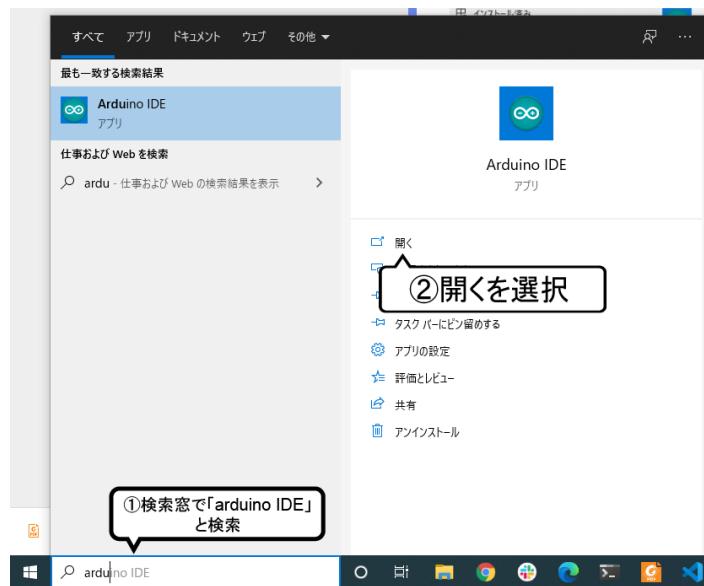


図 2.8: ArduinoIDE の検索

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）。

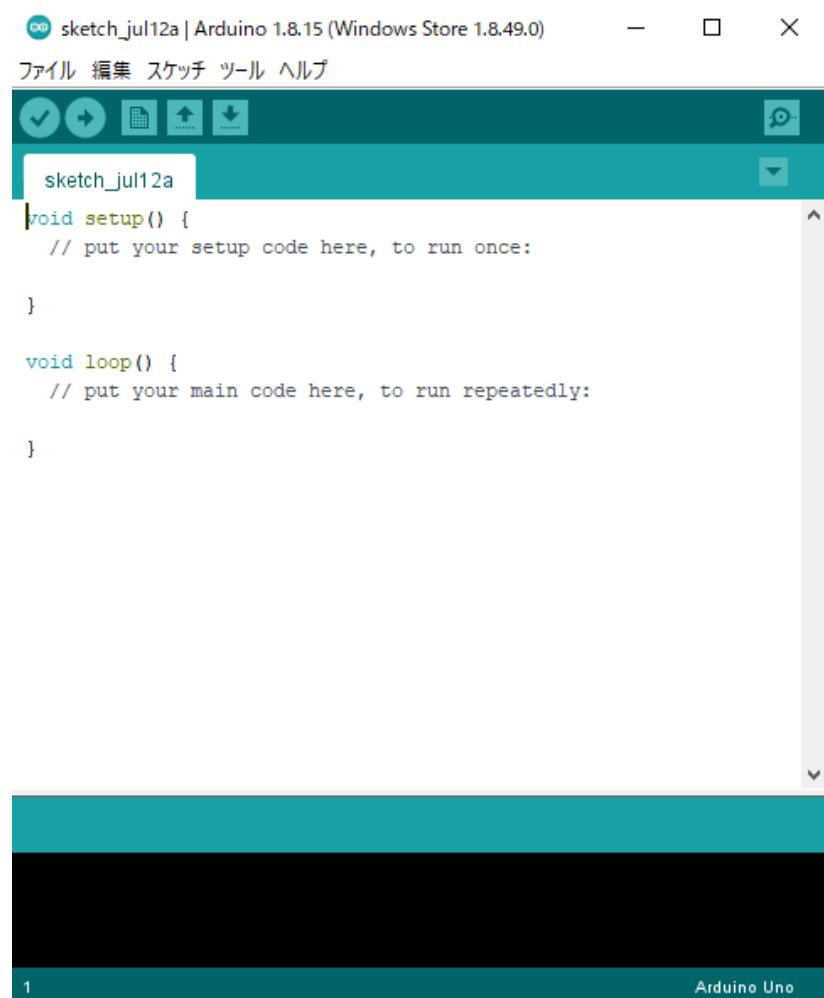


図 2.10: デフォルトのスケッチ画面

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

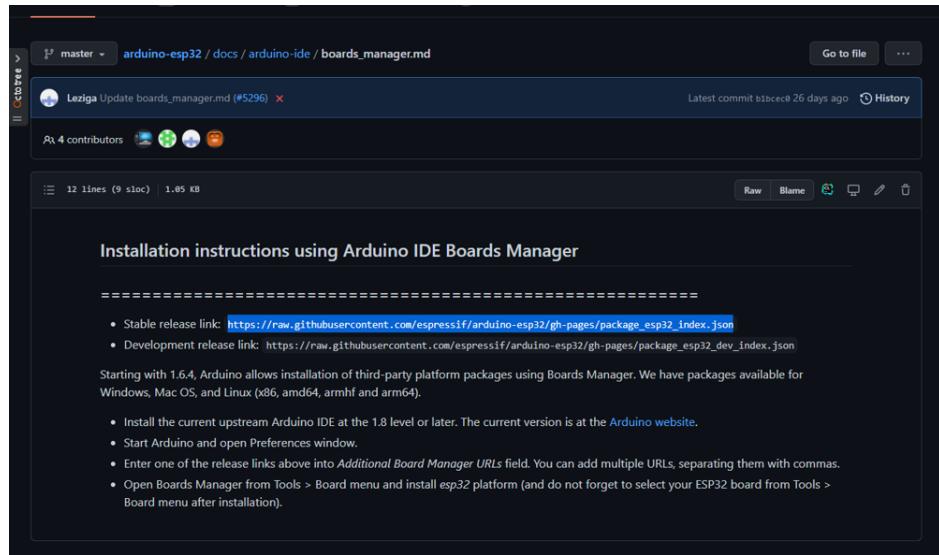


図 2.11: ESP32 を ArduinoIDE で使うための設定

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/package_esp32_index.json



The screenshot shows a JSON object representing the 'esp32' package. It includes fields for name, version, URL, help, platforms, boards, and toolsDependencies. The 'platforms' section lists four boards: 'ESP32 Dev Module', 'NEMO LoLin32', 'NEMO DI MINI ESP32', and 'NEMO DI WiFi'. The 'toolsDependencies' section lists three tools: 'xtensa-esp32-elf-gcc', 'espTool_py', and 'mkspiffs'.

```
{  
  "name": "esp32",  
  "version": "1.0.6",  
  "url": "https://github.com/espressif/arduino-esp32/releases/download/1.0.6/esp32-1.0.6.zip",  
  "checksum": "SHA-256:982d9aa181d8cbcb92dd49822bd022ecc0d1e9aa0c5b70428ccc3cb4556b",  
  "size": "51128602",  
  "help": {  
    "online": ""  
  },  
  "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",  
     "archiveName": "esp32-1.0.6.zip",  
     "checksum": "SHA-256:982d9aa181d8cbcb92dd49822bd022ecc0d1e9aa0c5b70428ccc3cb4556b",  
     "size": "51128602",  
     "help": {  
       "online": ""  
     },  
     "boards": [  
       {"name": "ESP32 Dev Module"},  
       {"name": "NEMO LoLin32"},  
       {"name": "NEMO DI MINI ESP32"},  
       {"name": "NEMO DI WiFi"}  
     ],  
     "toolsDependencies": [  
       {"packager": "esp32",  
        "name": "xtensa-esp32-elf-gcc",  
        "version": "1.22.0-97-ac759ad5-5.2.0"},  
       {"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)。

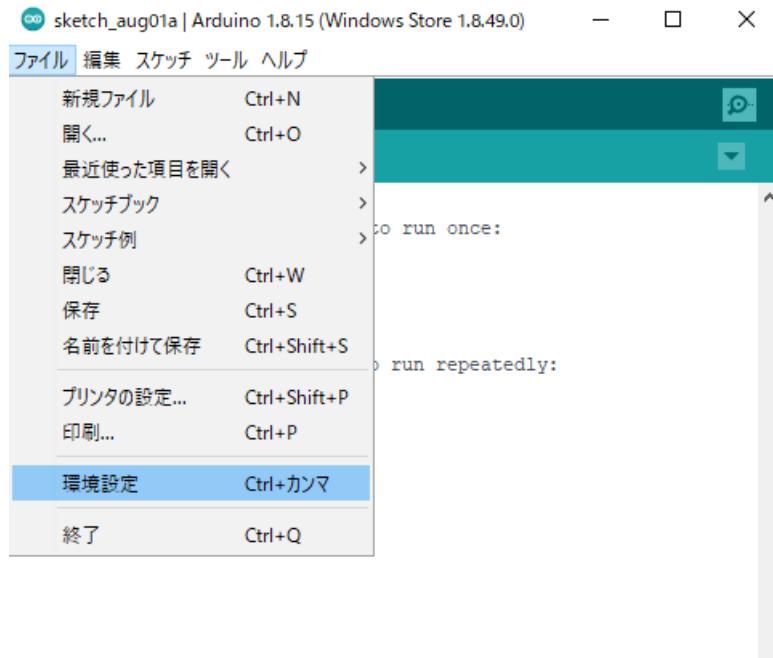


図 2.13: 環境設定を選択

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）。

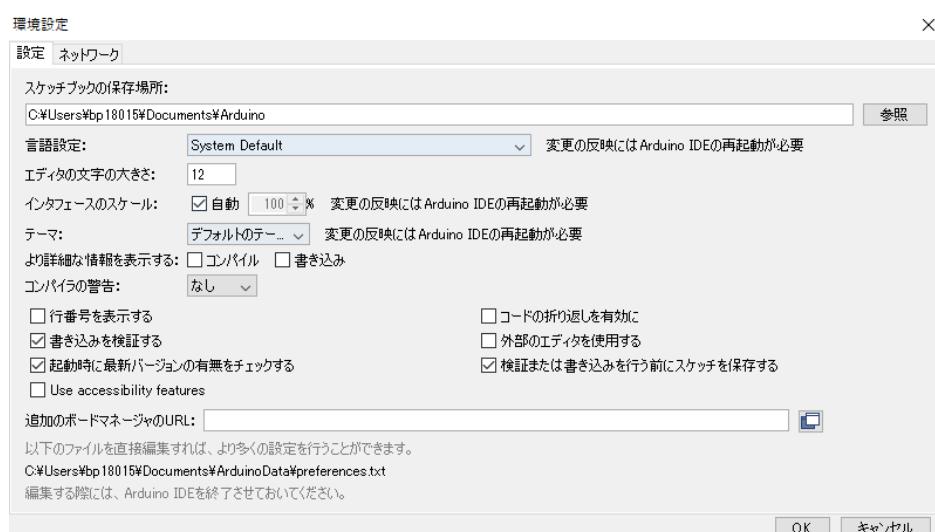


図 2.14: 環境設定の画面

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)。

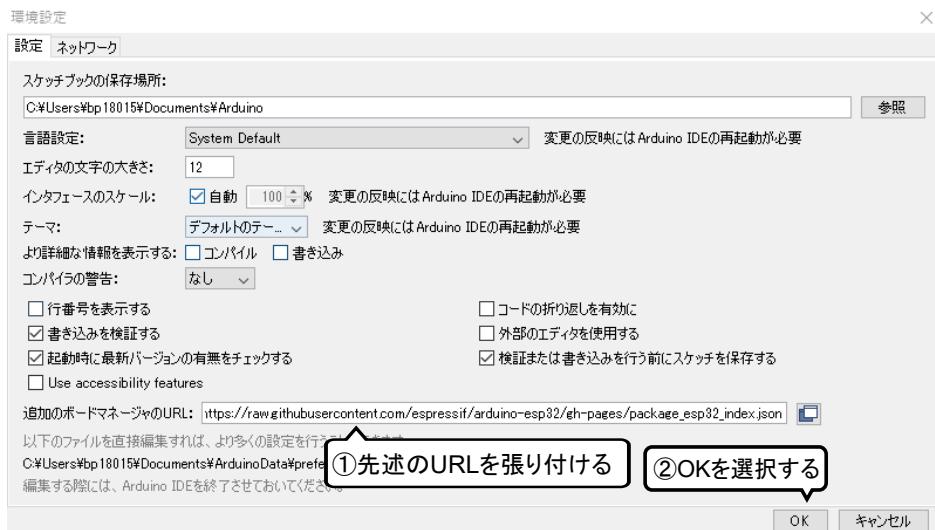


図 2.15: 追加ボードマネージャーの URL に貼り付ける

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。

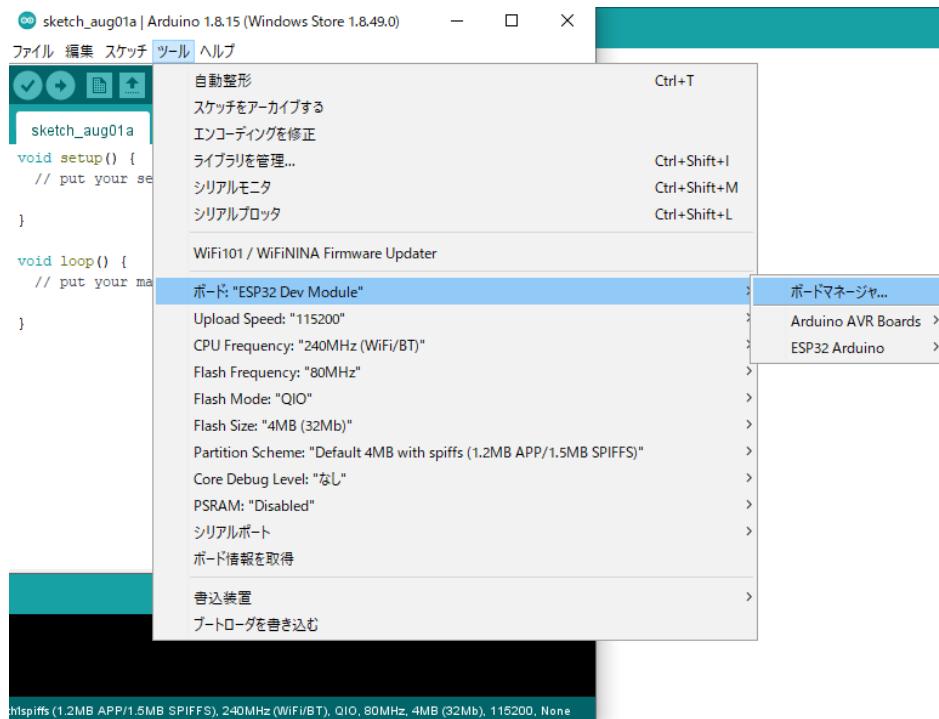


図 2.16: ボードマネージャーを開く

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。

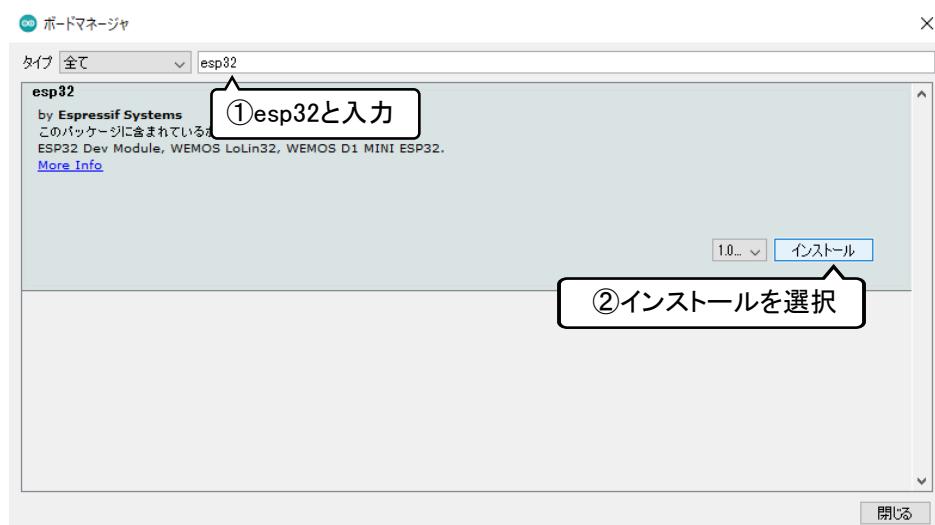


図 2.17: ESP32用ボードマネージャーのインストール

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。

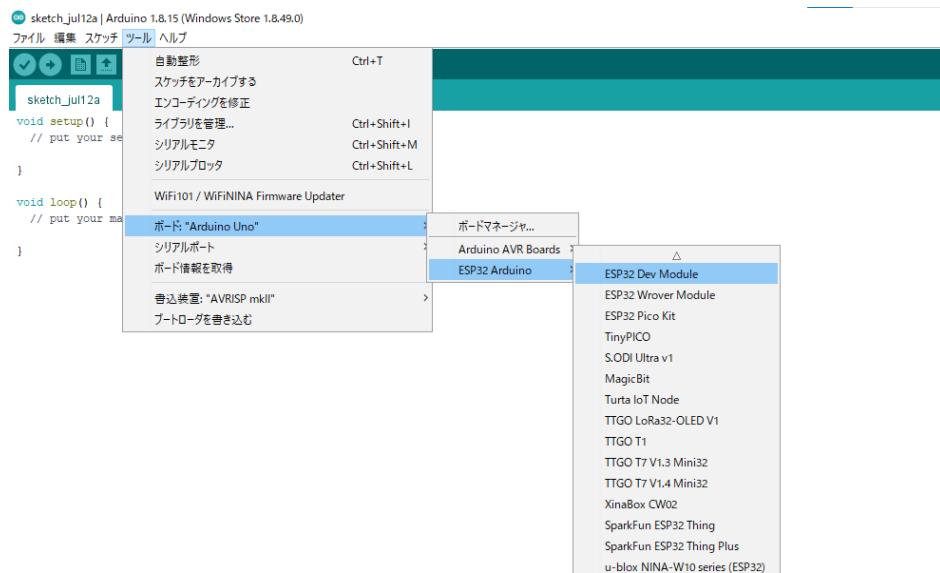


図 2.18: ボード ESP32 Dev Module の選択

2.5 Hello ESP32!!

ここで動作確認を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。

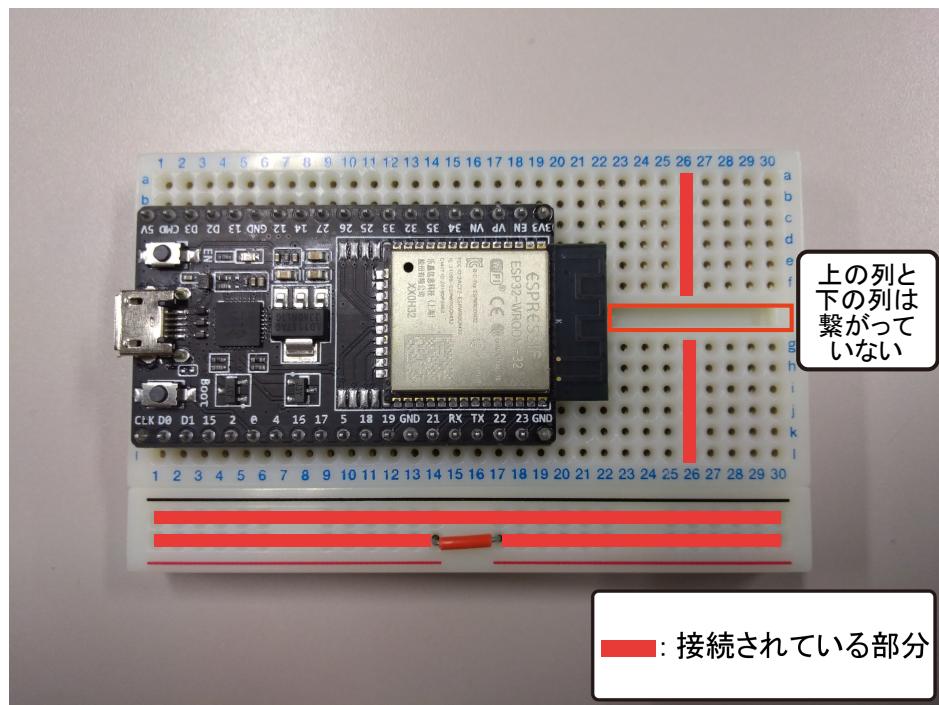


図 2.19: ブレッドボード

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。

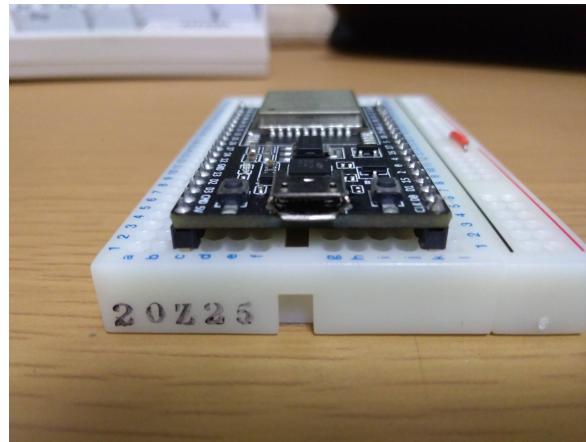


図 2.20: microUSB type-B 差し込み口

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）。

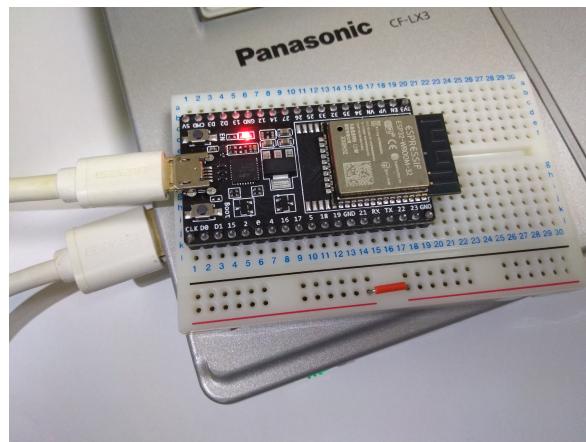


図 2.21: PC との接続

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。

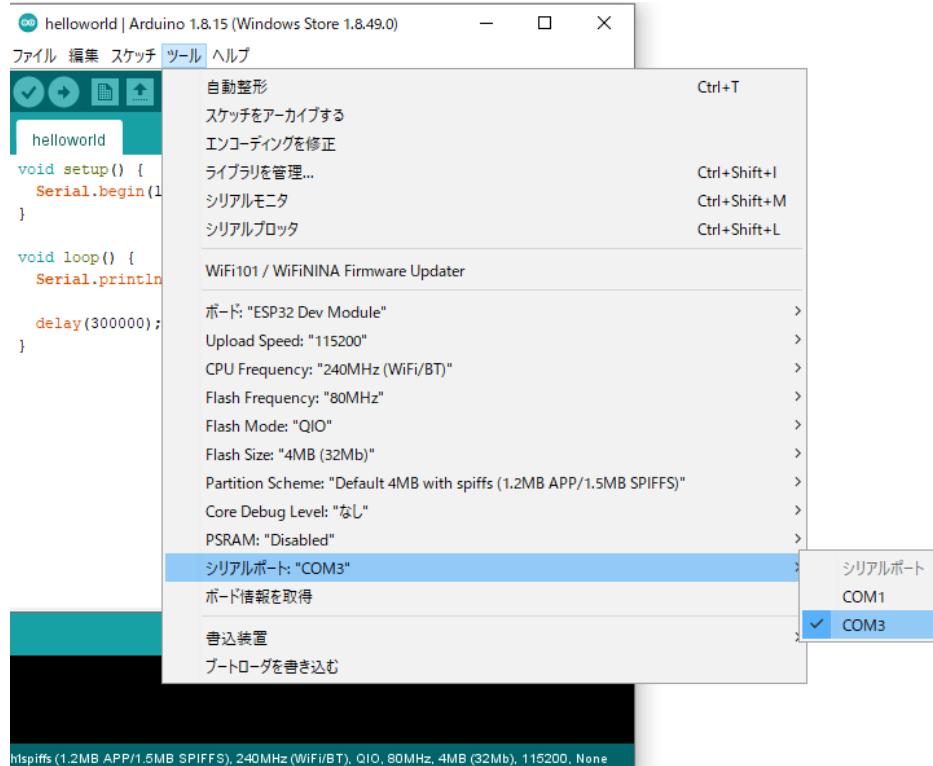


図 2.24: 接続ポートの反映

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。

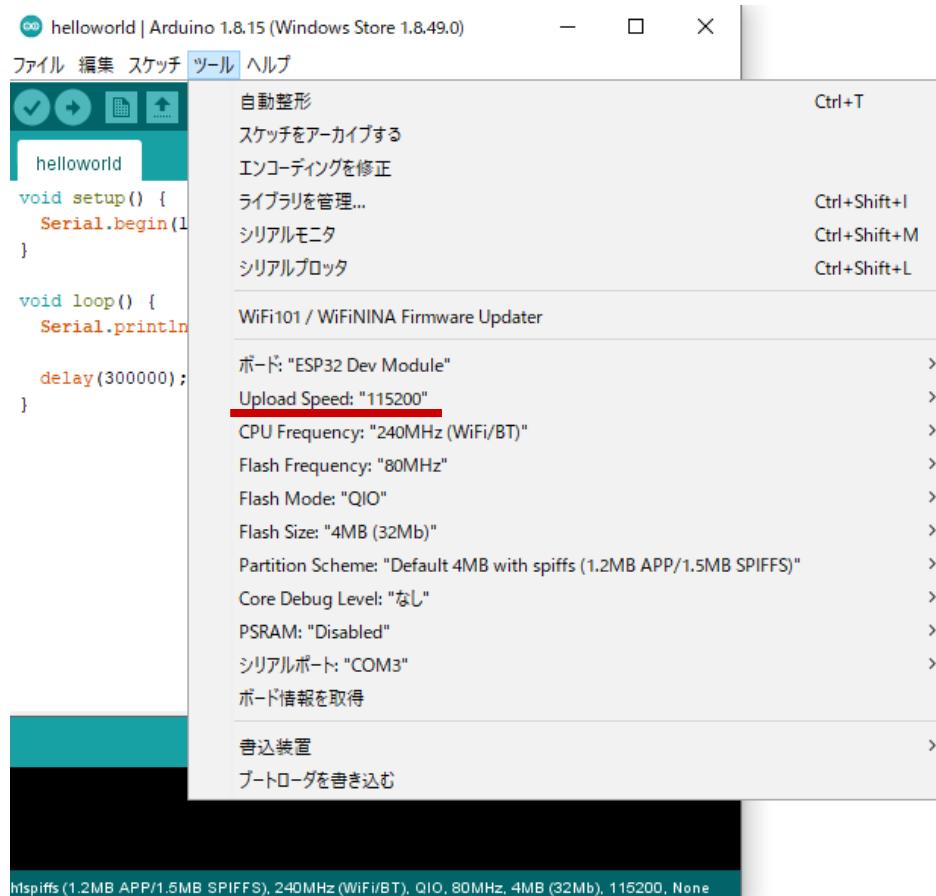


図 2.25: ボードの設定

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)。

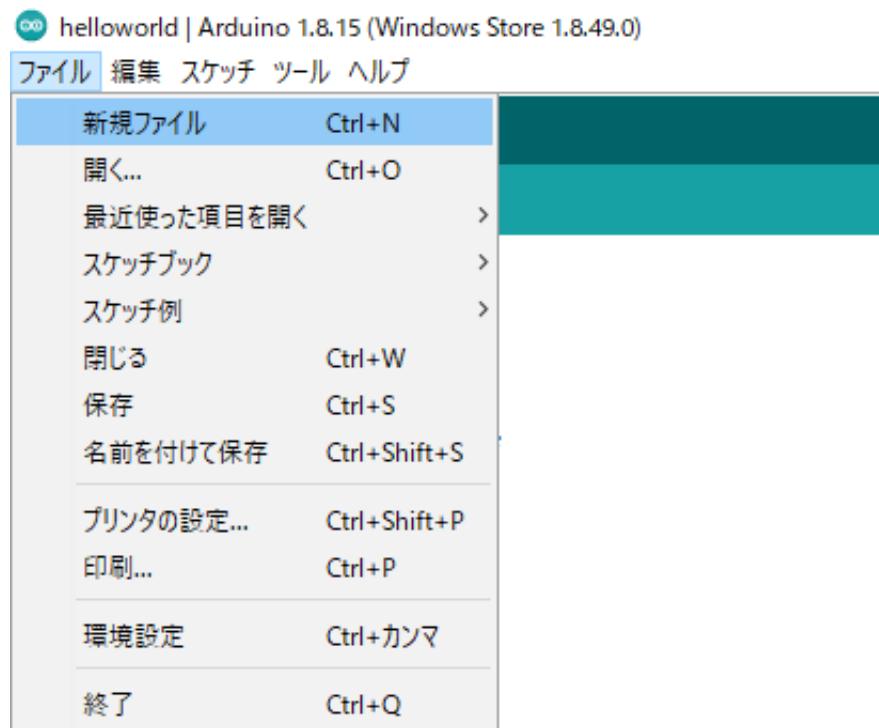


図 2.26: 新規ファイルの作成

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）。

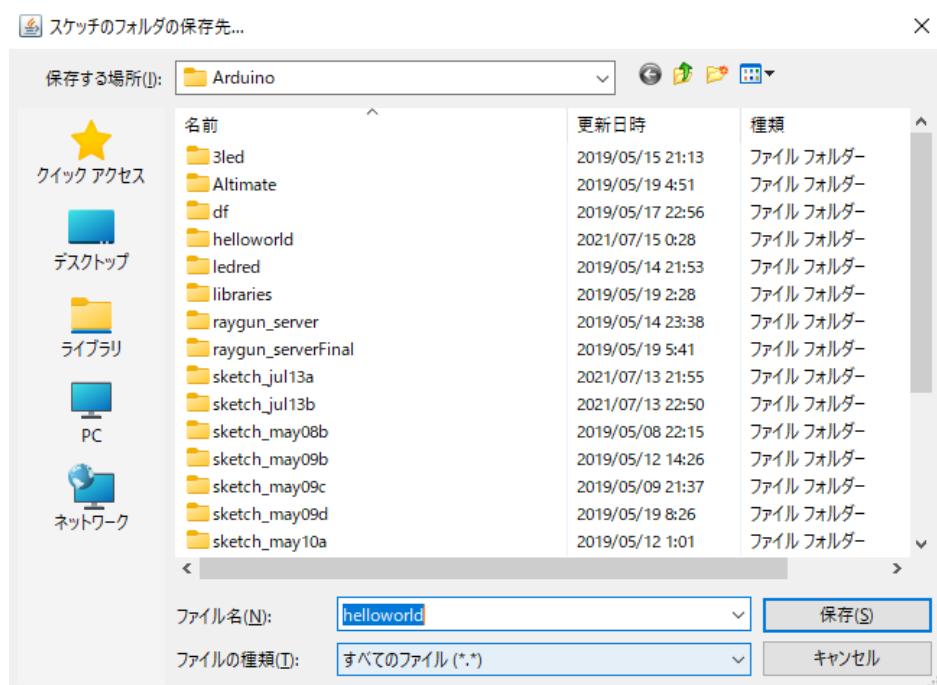


図 2.27: 新規ファイルの名前決定

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)". The menu bar includes "ファイル", "編集", "スケッチ", "ツール", and "ヘルプ". Below the menu is a toolbar with icons for save, upload, and refresh. The main area displays the code for the "helloworld" sketch:

```
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）。

```
helloworld | Arduino 1.8.15 (Windows Store 1.8.49.0)
ファイル 編集 スケッチ ツール ヘルプ
✓ + 📁 ⬆️ ⬇️ 検証
helloworld
void setup() {
    Serial.begin(115200);
}

void loop() {
    Serial.println("Hello,World");
    delay(300000);
}

ボードへの書き込みが完了しました。
Writing at 0x00020000... (71 %)
Writing at 0x00024000... (85 %)
Writing at 0x00028000... (100 %)
Wrote 204608 bytes (106238 compressed) at 0x00010000 in 9.4 seconds (effective 173.3 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 1170.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 115200, None
```

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）。

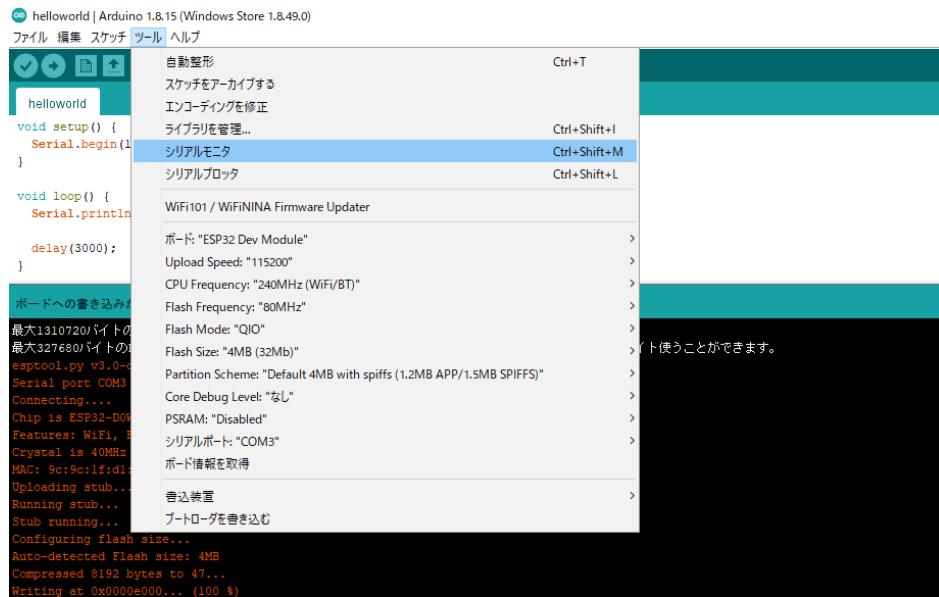


図 2.31: シリアルモニタの選択

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

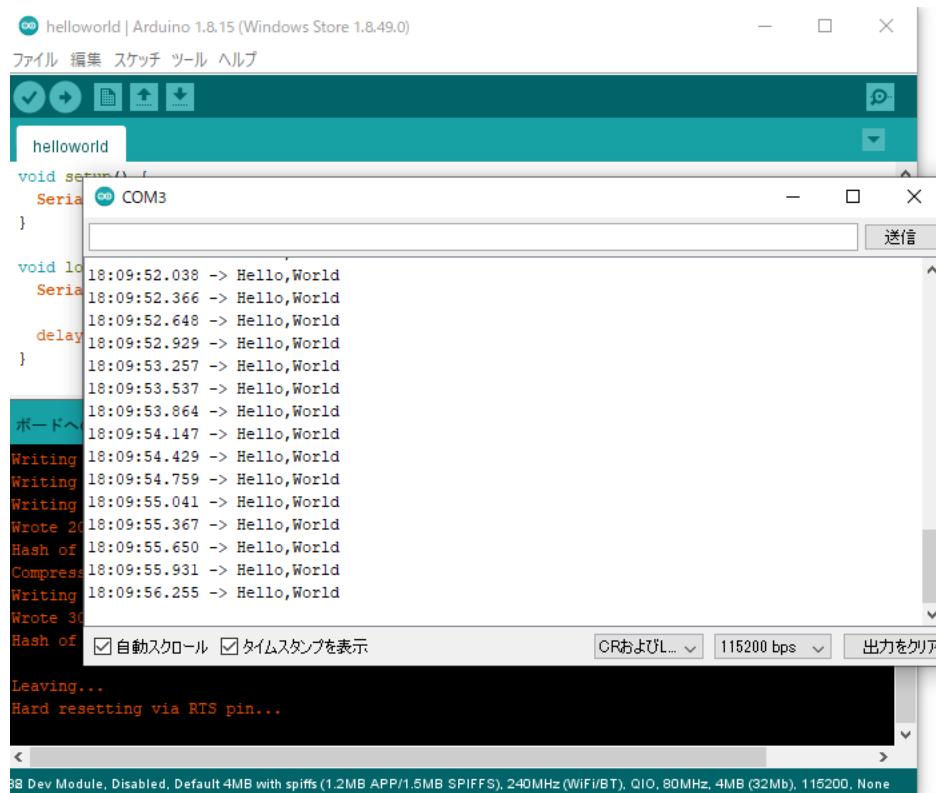


図 2.32: helloworld の表示成功

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

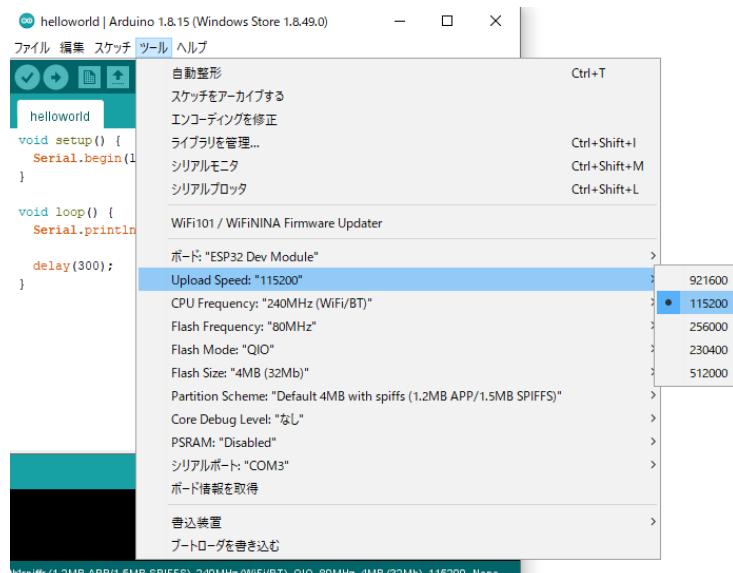
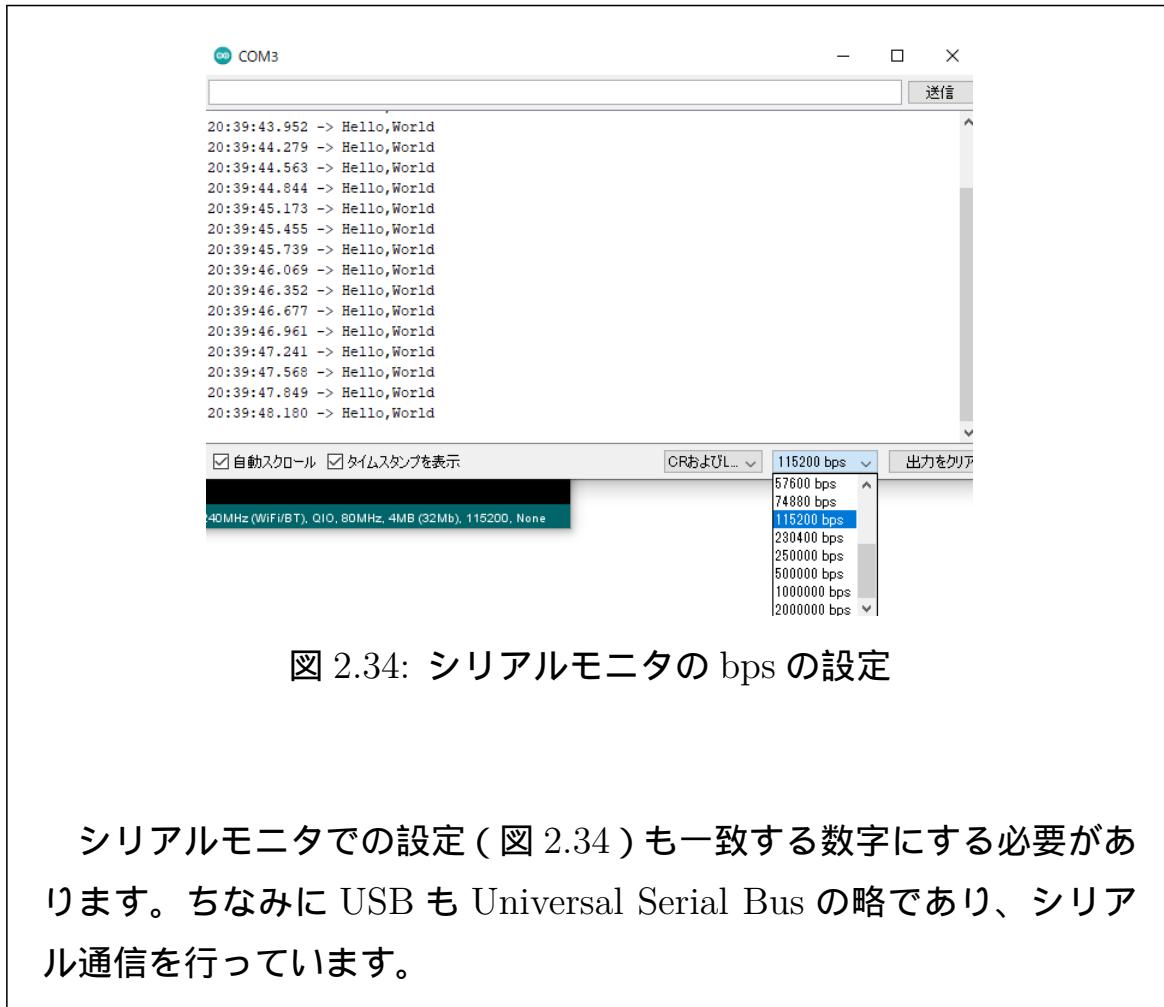


図 2.33: UploadSpeed の設定

設定の UploadSpeed (図 2.33) や



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。

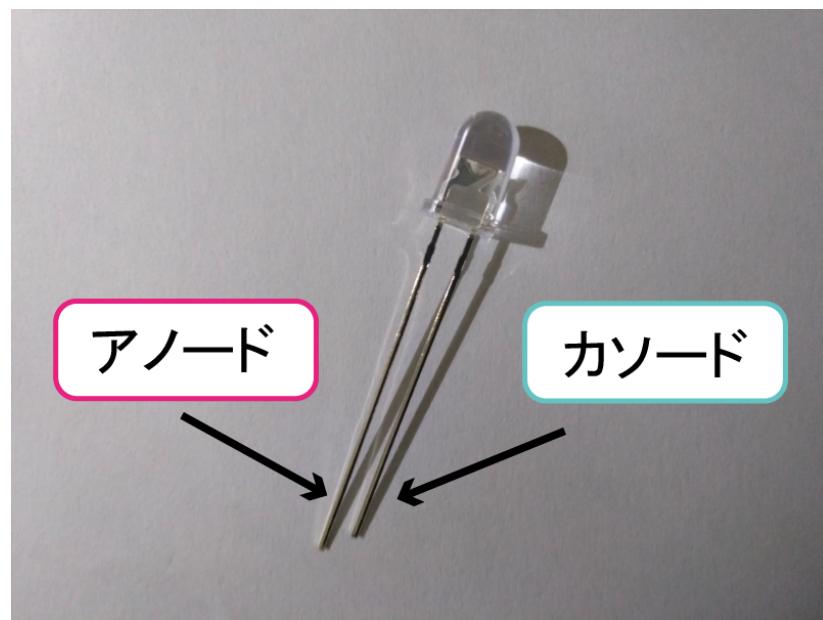


図 3.1: LED の端子の違い

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）。

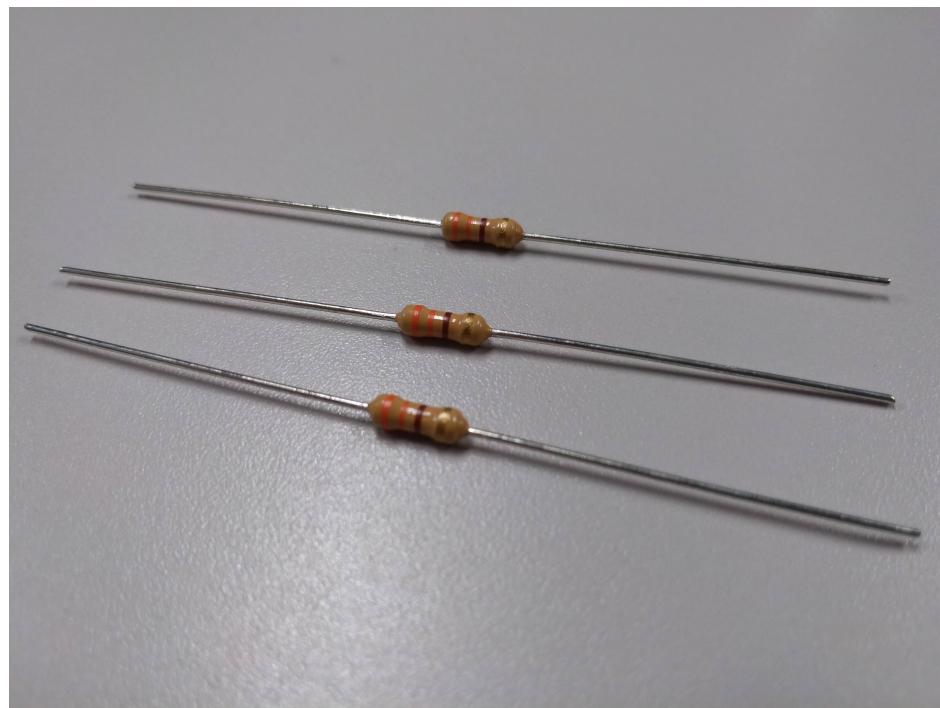


図 3.5: 330 抵抗

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。

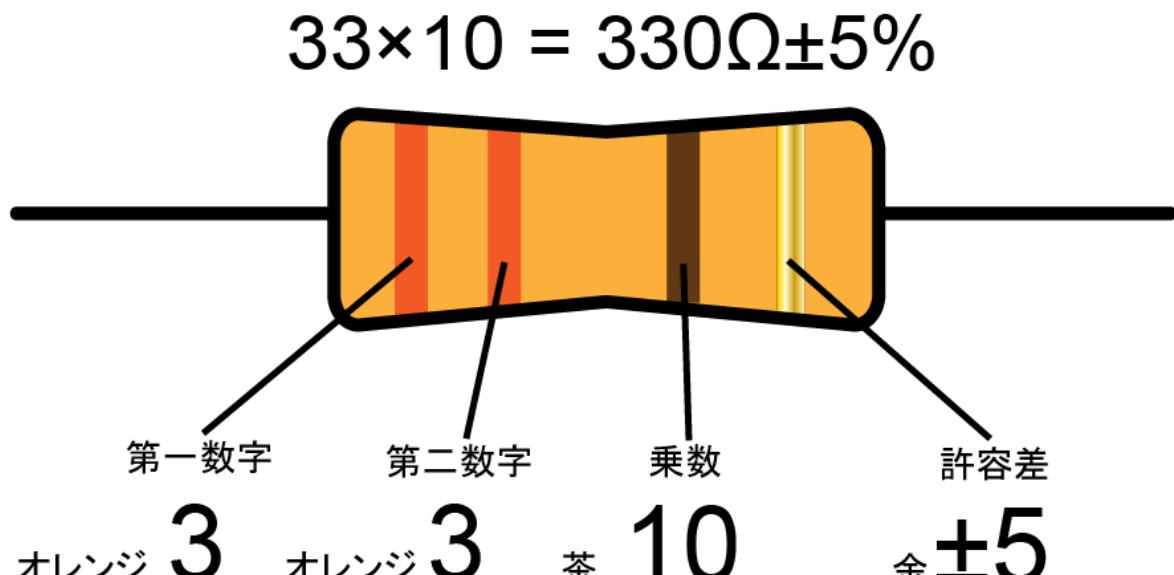


図 3.6: 抵抗値の読み取り

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。

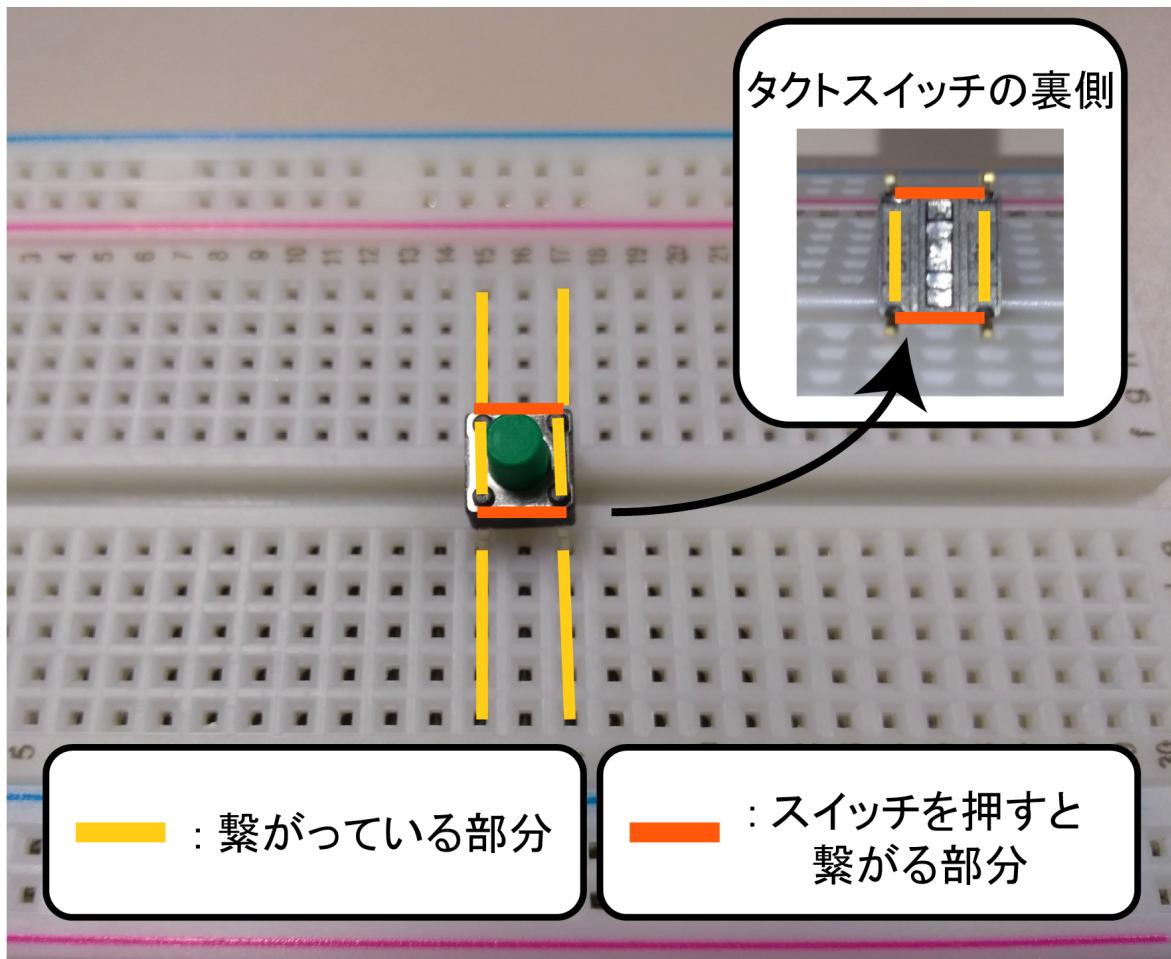


図 3.8: タクトスイッチの特徴

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）。

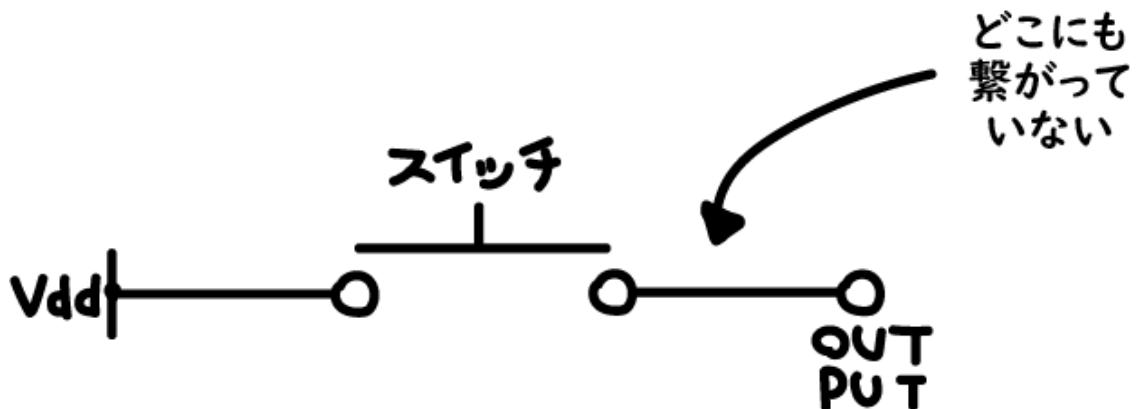


図 3.9: オフ状態のスイッチ

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。

これら二つの方法を用いることで安定した回路を組むことができます。

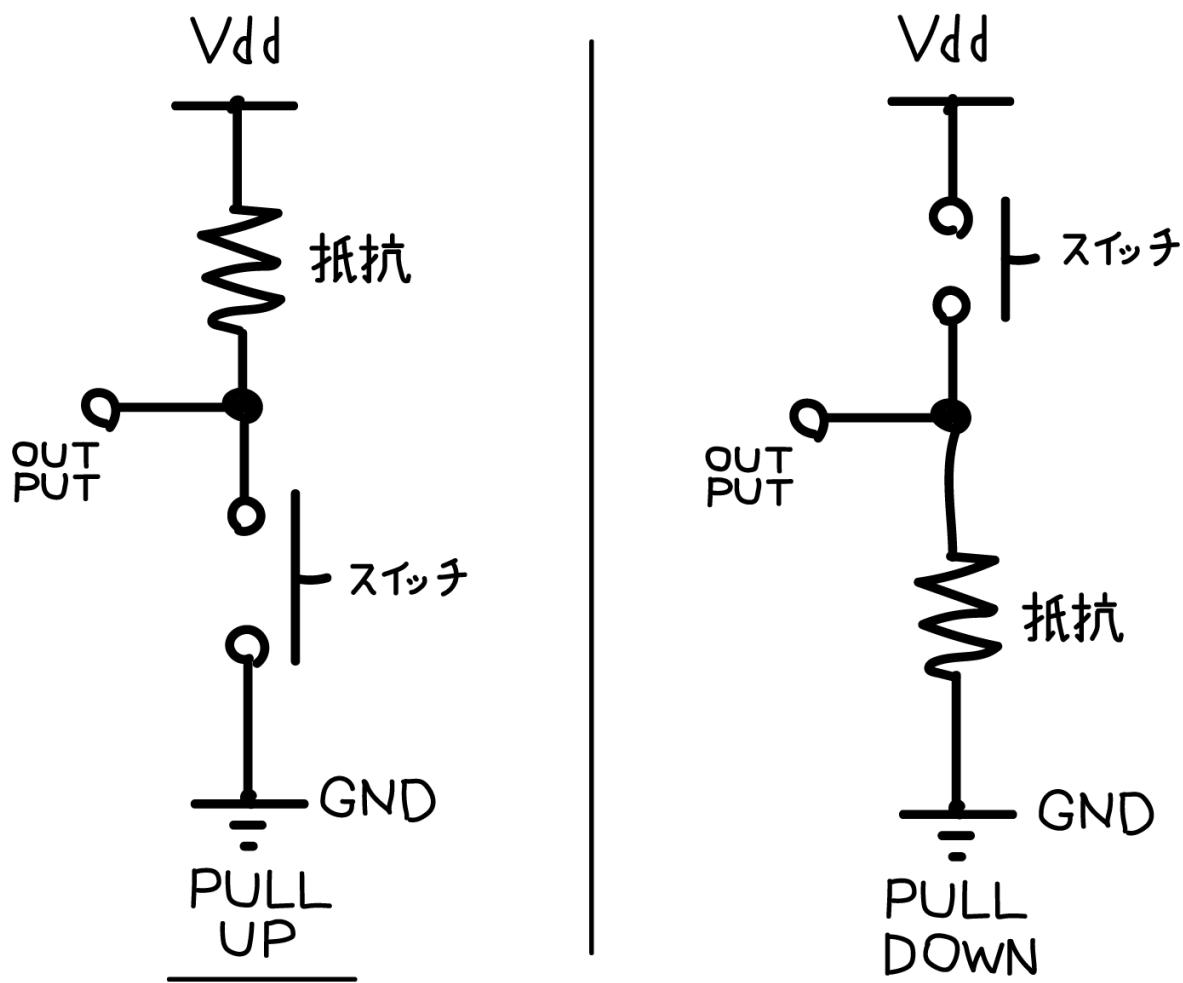


図 3.10: PULL UP と PULL DOWN

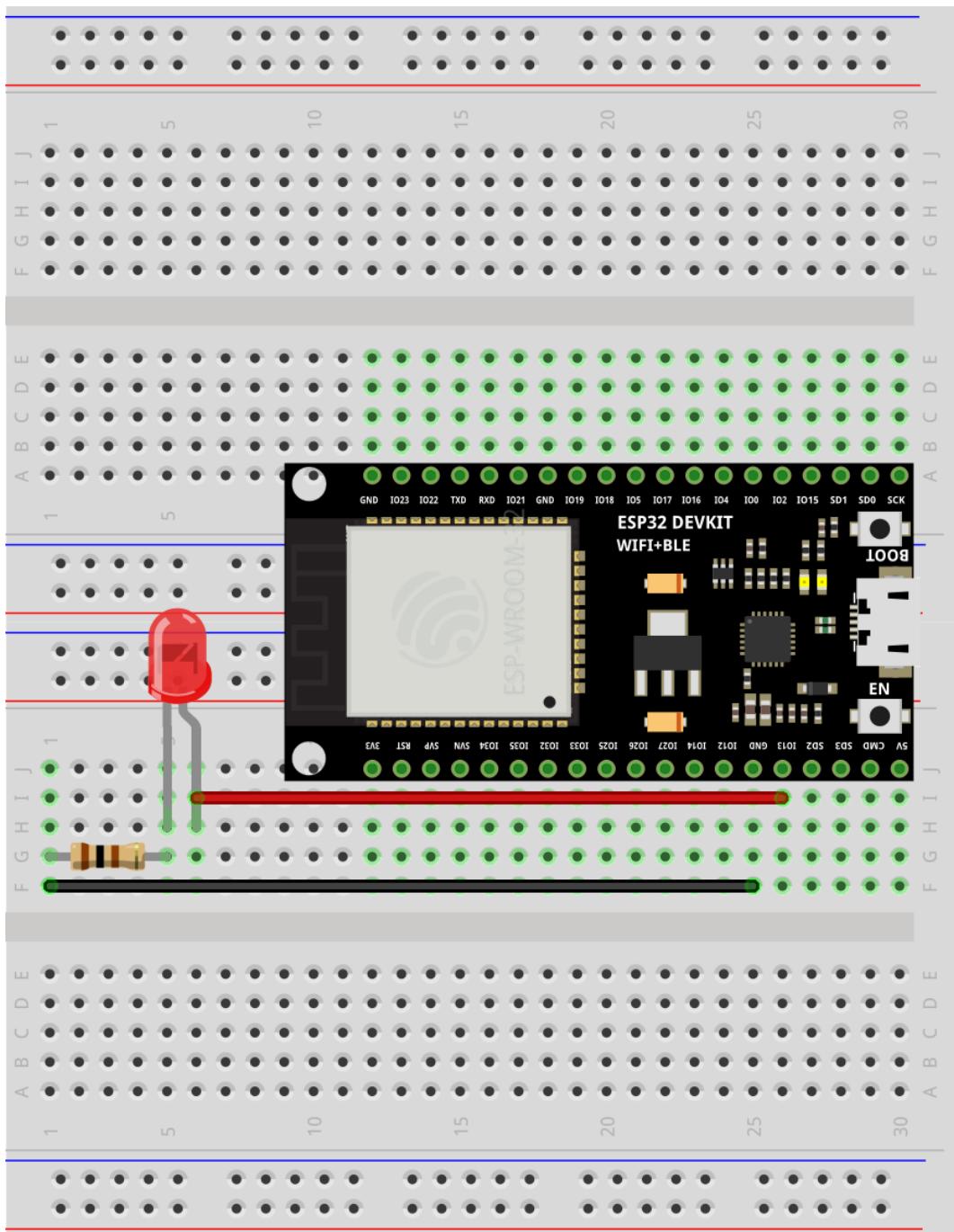
3.2 Lチカしよう！

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）。

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 1
 - 100Ω × 1
 - LED × 1
 - ジャンプワイヤ



fritzing

図 3.11: L チカの回路図

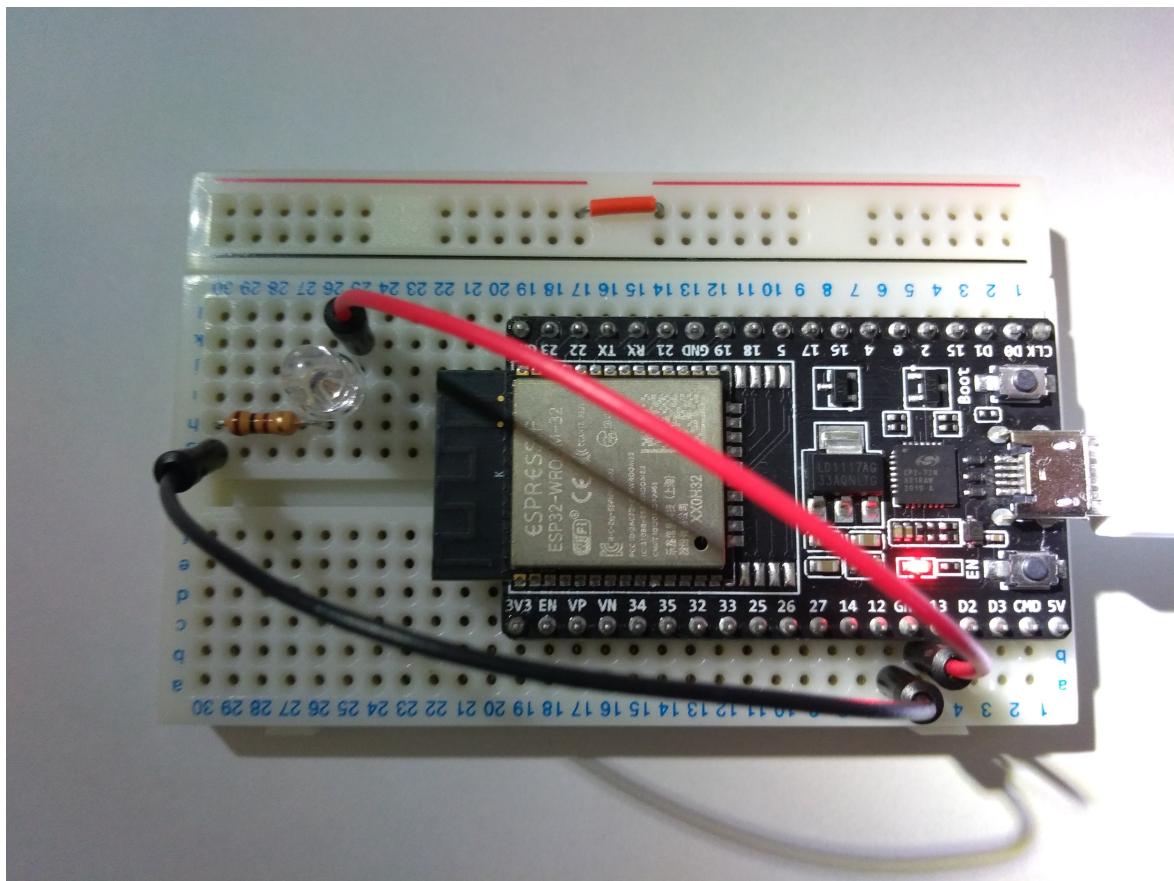


図 3.12: Lチカの回路を配置した図

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
}
```

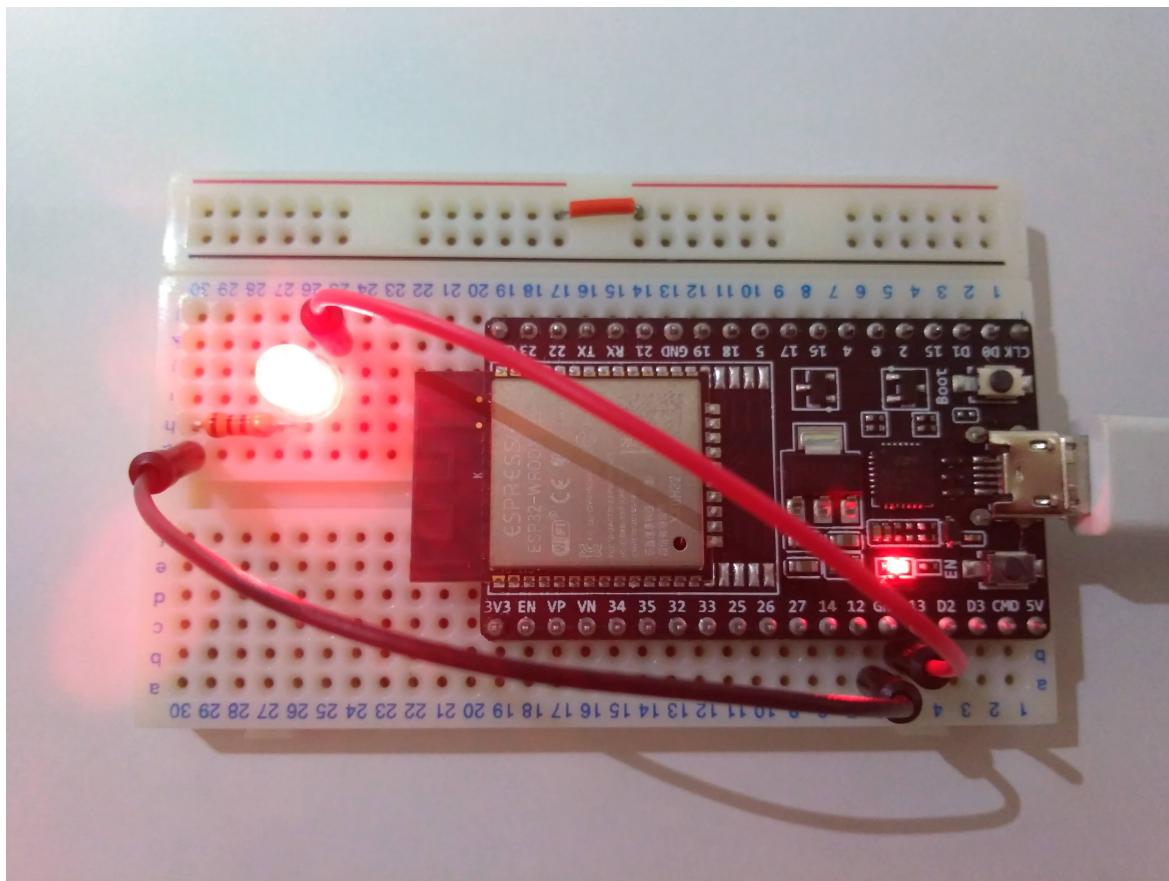


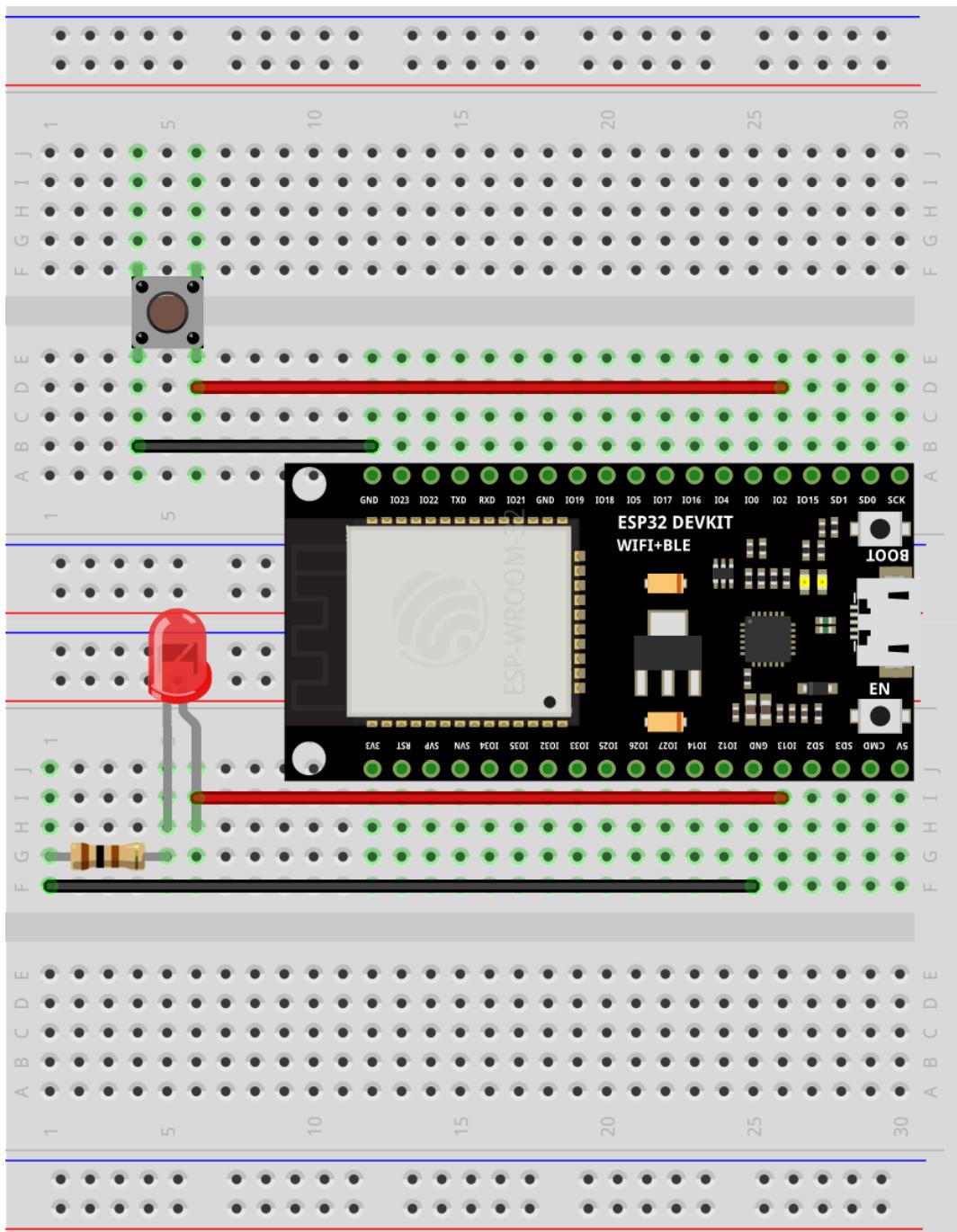
図 3.13: LED 点滅図

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。

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 1

- 100 × 1
- LED × 1
- タクトスイッチ
- ジャンプワイヤ



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
    // GPIO2を用いるまた、ESP32の内部でPULL UPをする
    pinMode(2, INPUT_PULLUP);
    pinMode(13, OUTPUT); // GPIO13を出力端子として用いる
}

void loop()
{
    // 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
    if (digitalRead(2) == LOW)
    {
        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!");
    }
}
```

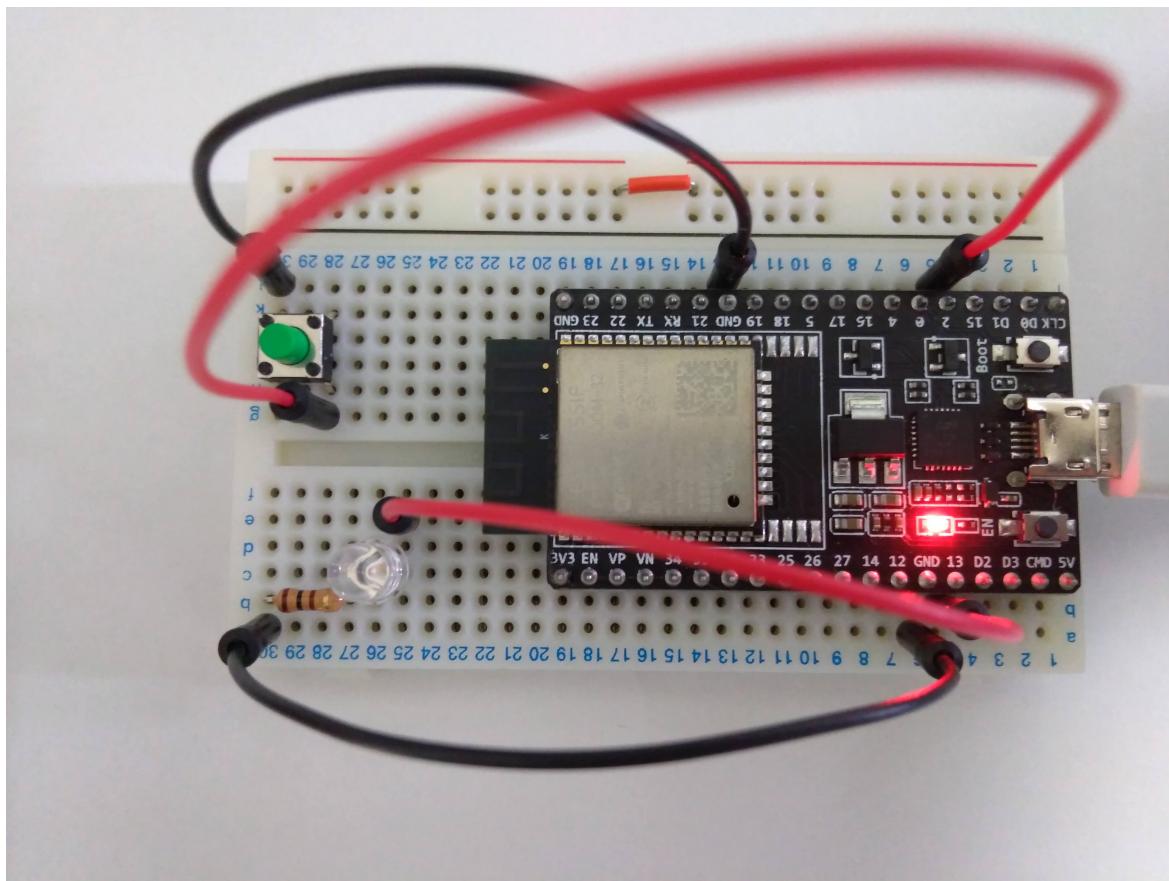


図 3.15: 回路をブレッドボード上で配置した図

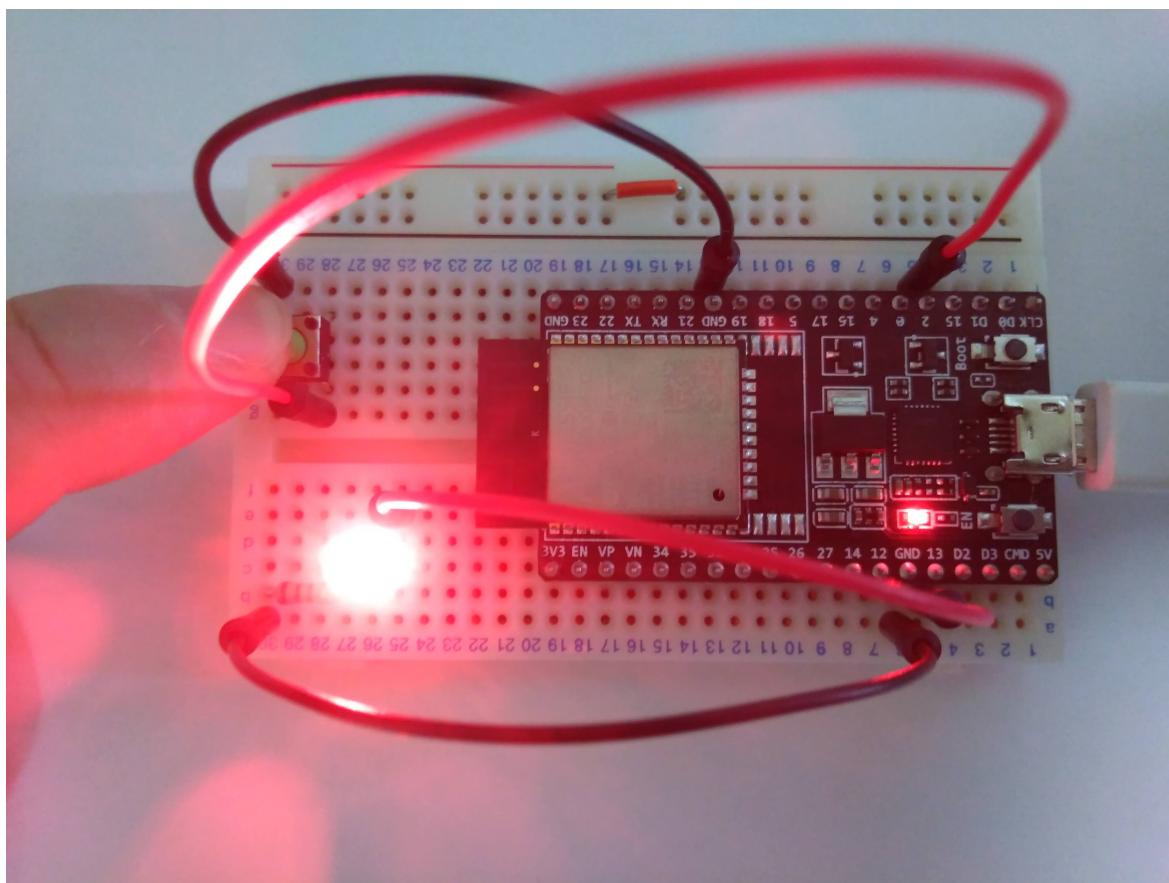


図 3.16: スイッチで LED を点滅させる図

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

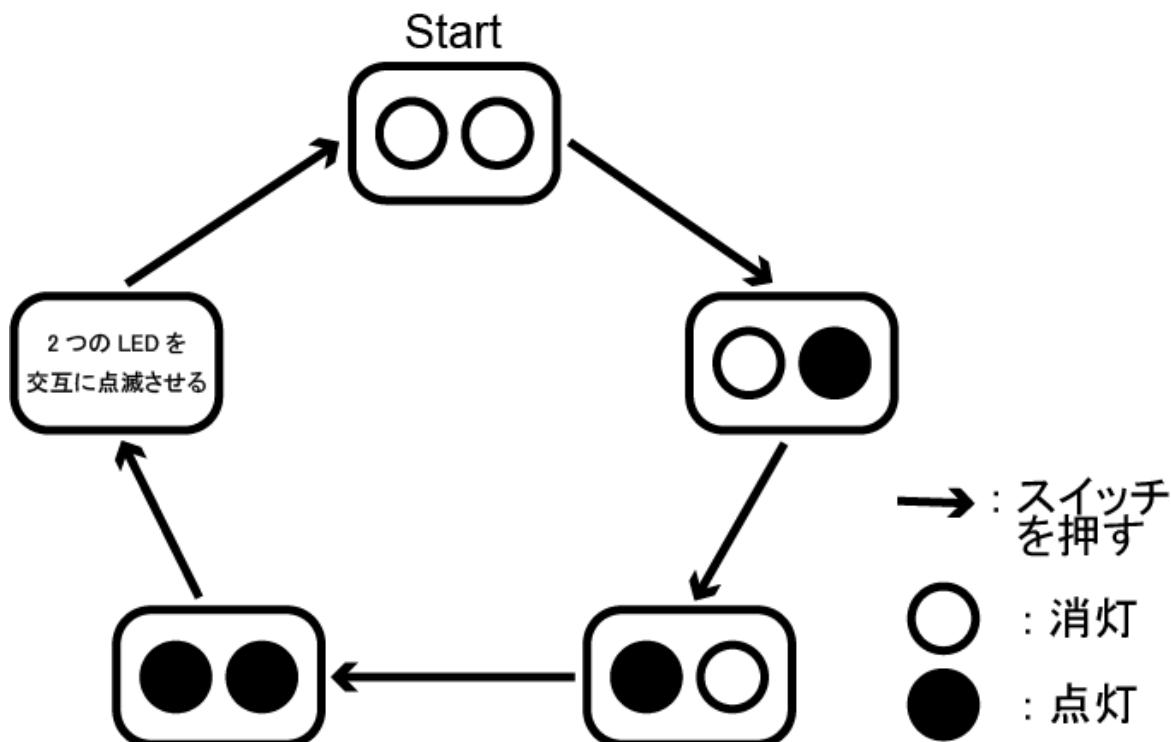


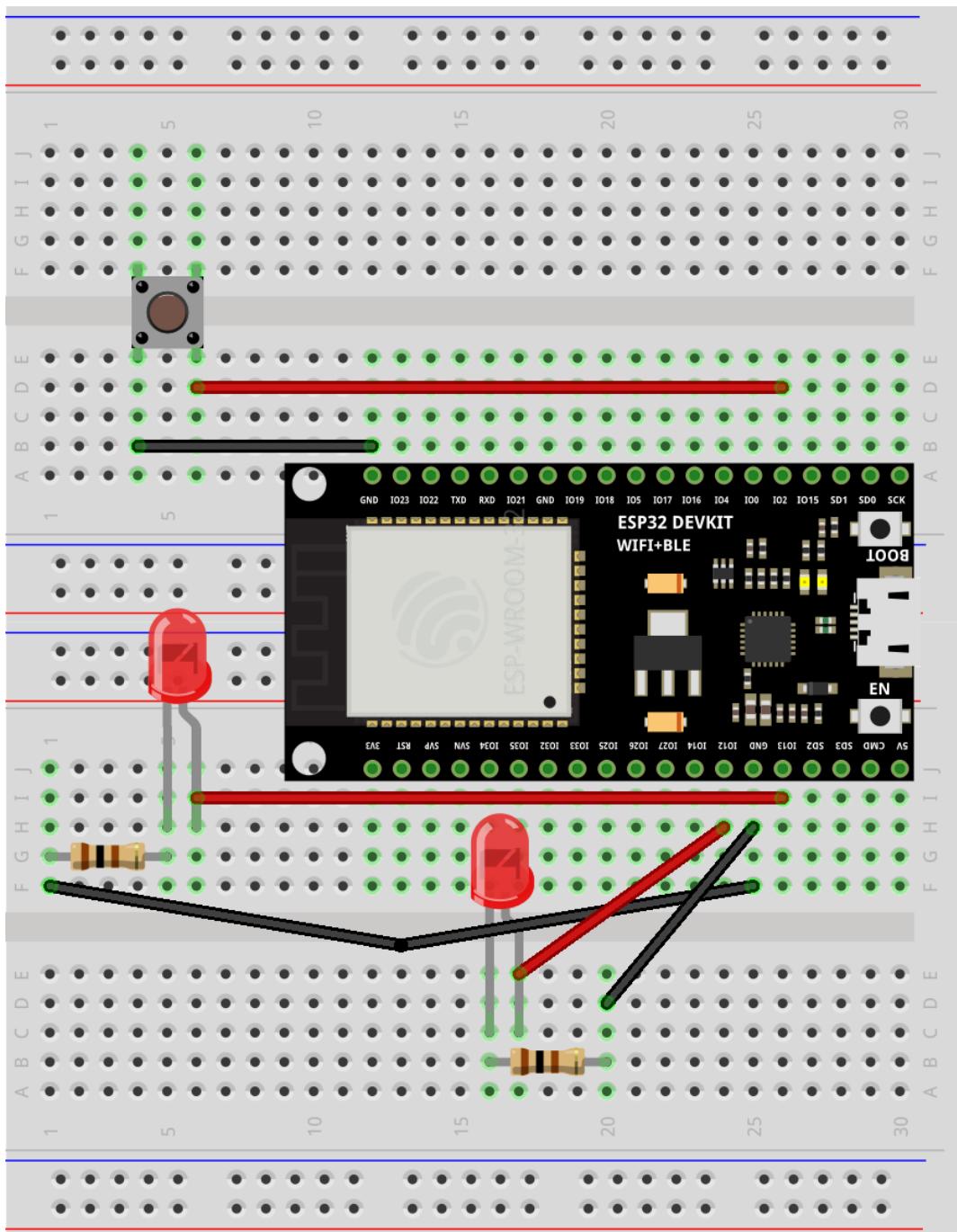
図 3.17: 状態遷移図

2019年度組み込み制作講座より引用

応用問題解答

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 2
 - 100 × 2
 - LED × 2
 - タクトスイッチ
 - ジャンプワイヤ

回路図 & 配置図



fritzing

図 3.18: 應用問題回路図

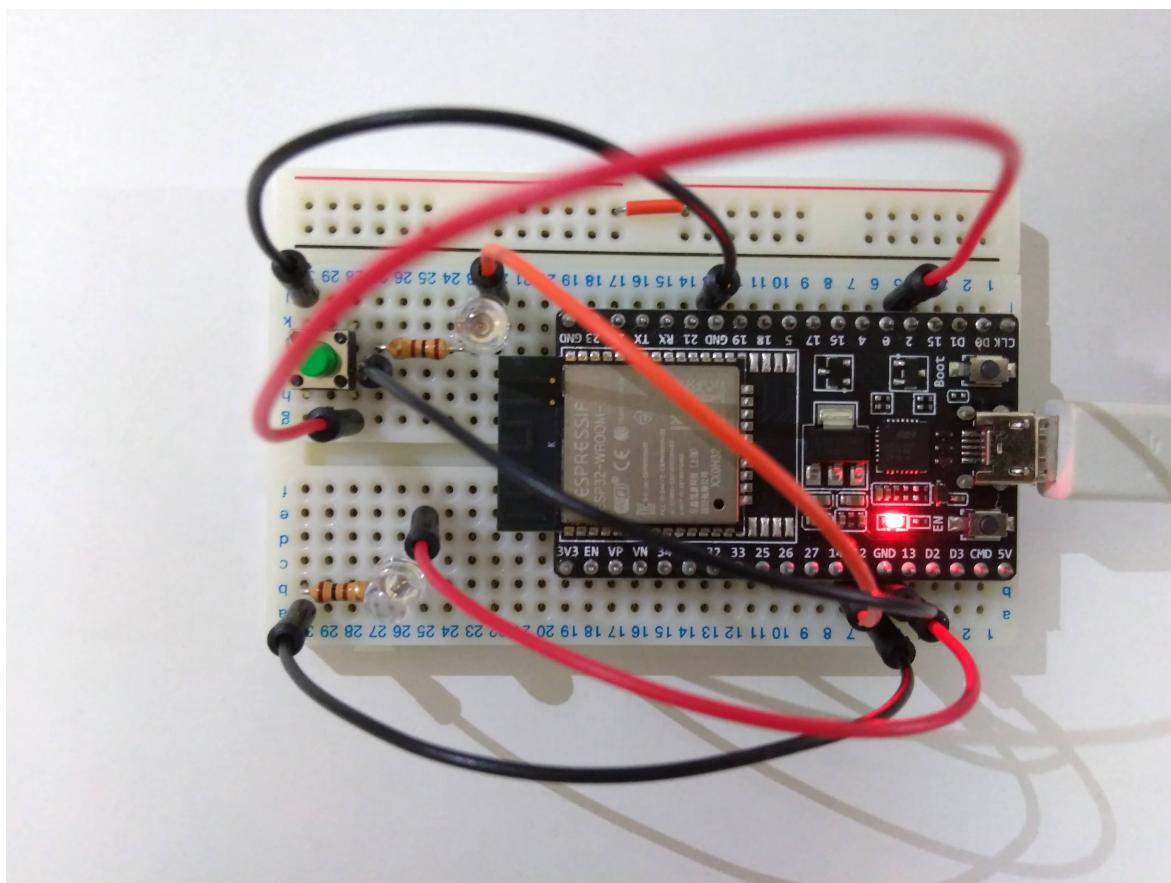


図 3.19: 応用問題回路配置図

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。

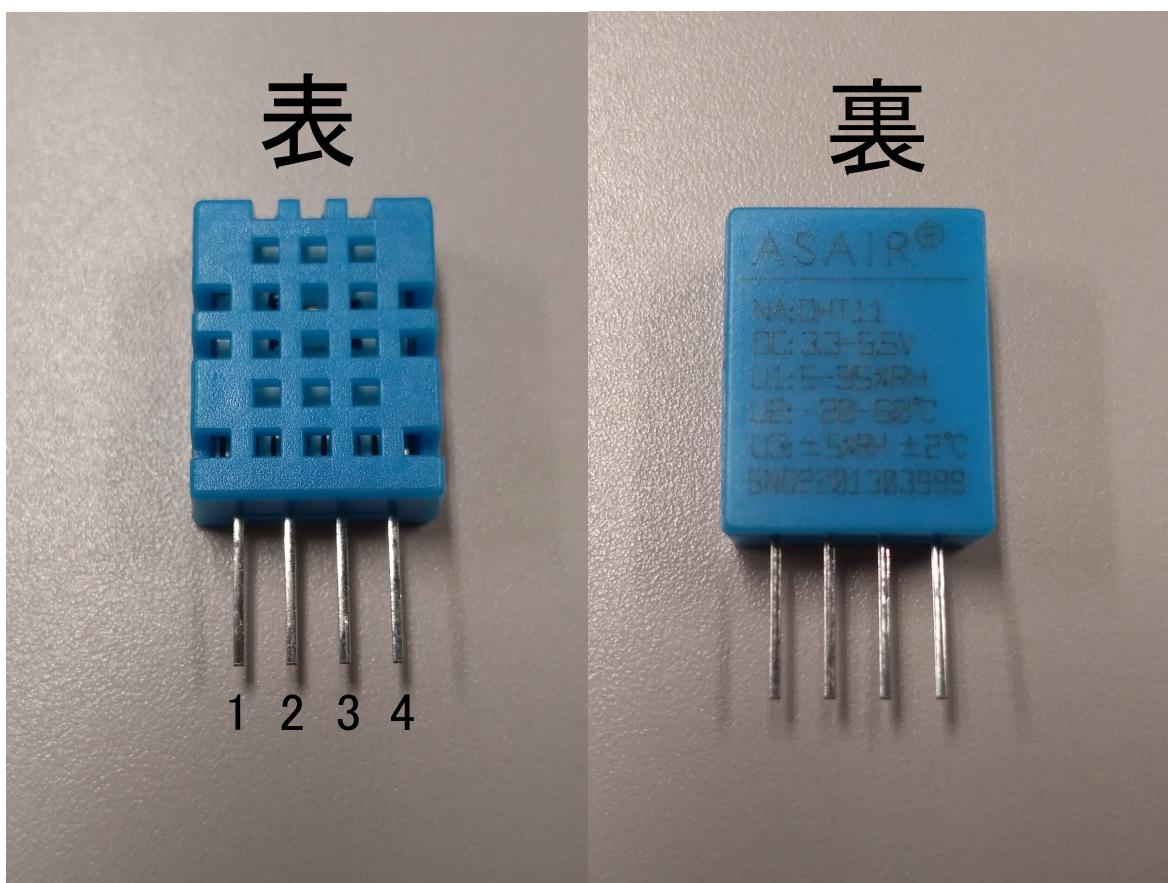


図 4.1: DHT11

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。

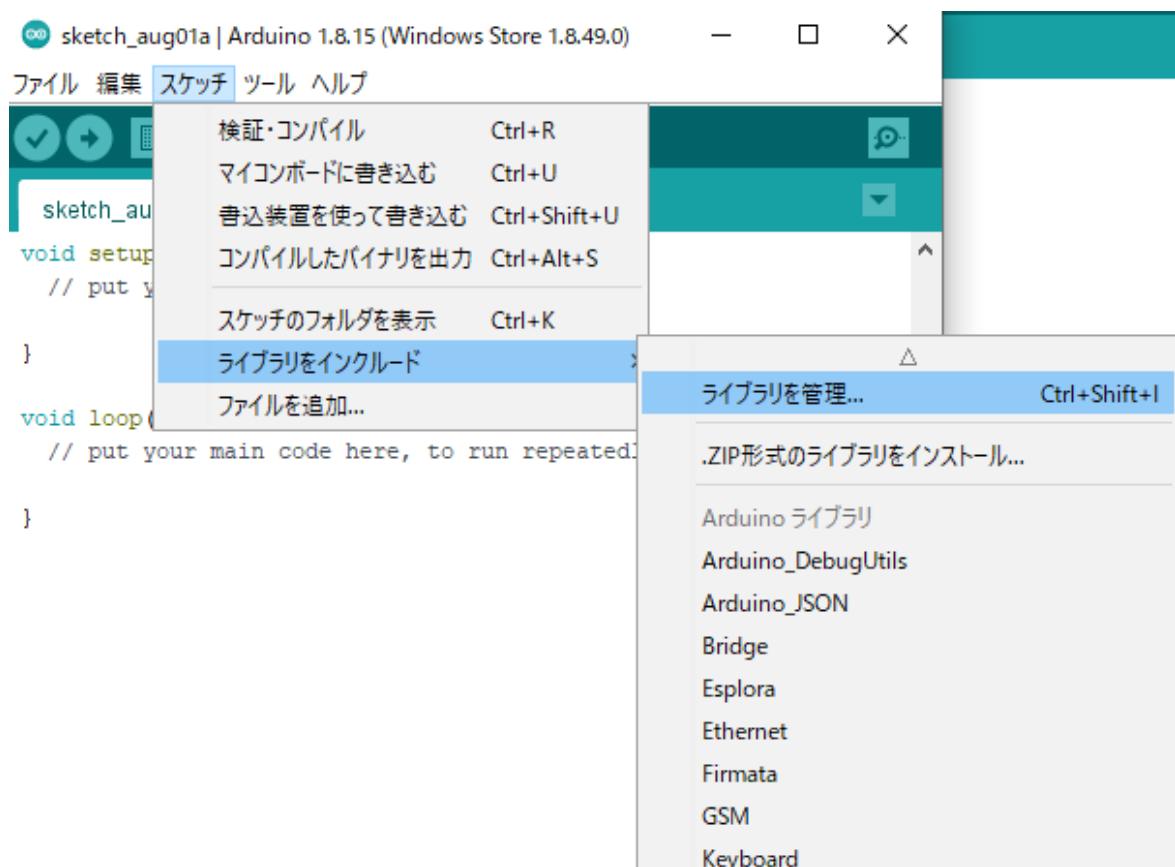


図 4.2: ライブラリの管理の選択

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。

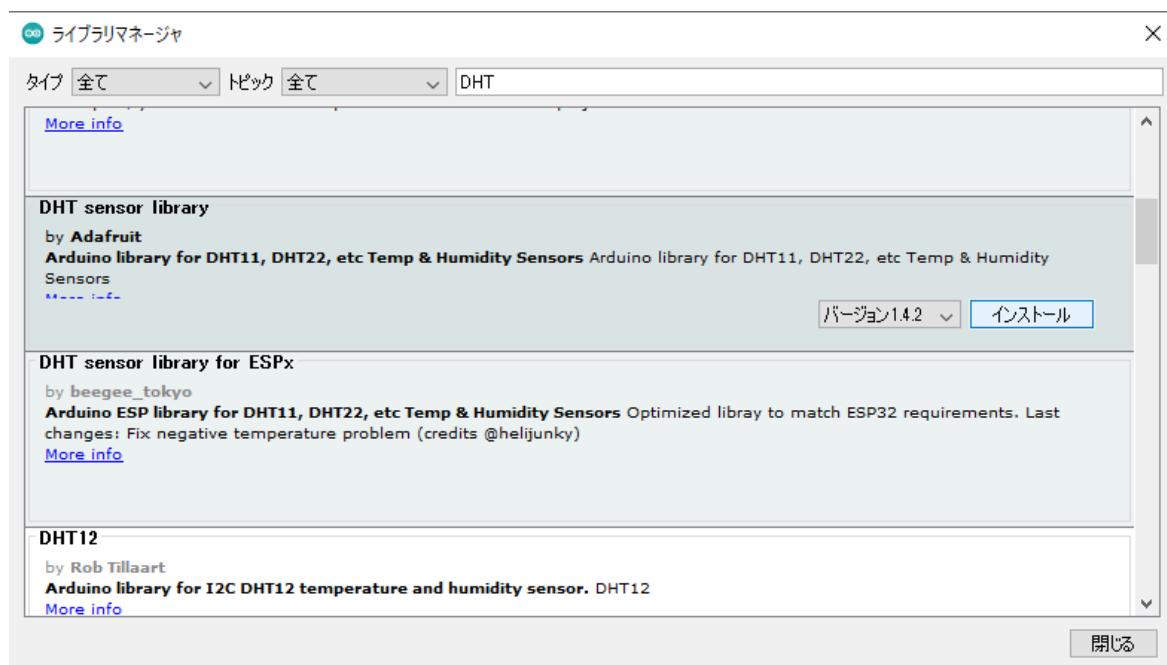


図 4.3: DHT11 用ライブラリのインストール

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)。

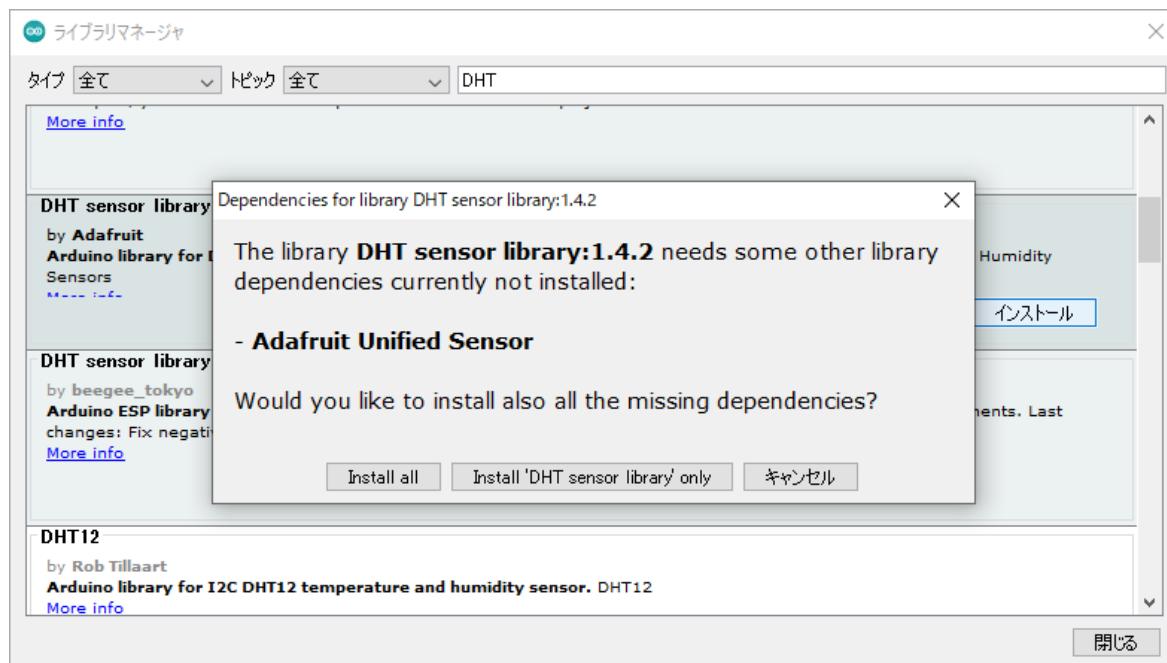
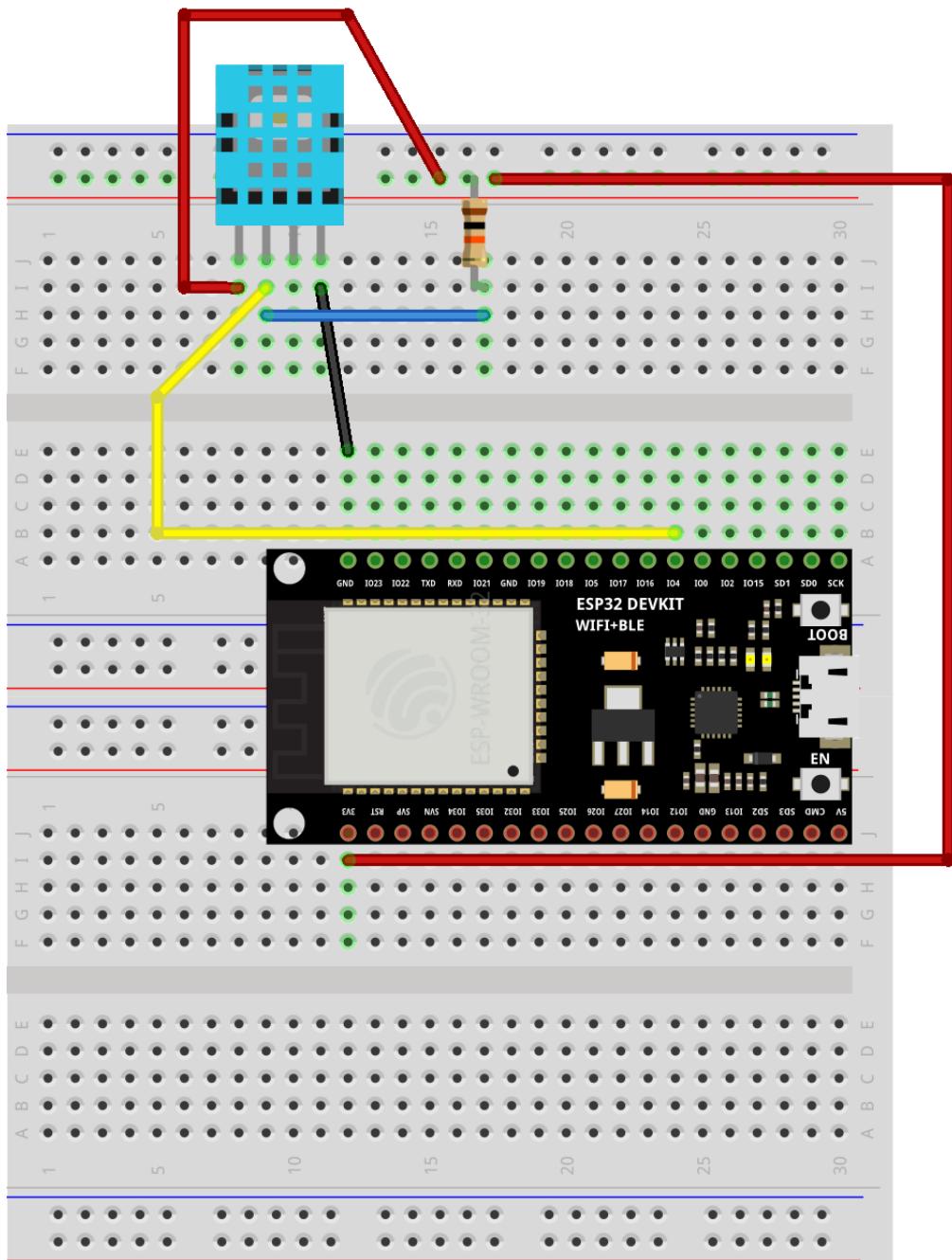


図 4.4: 依存ライブラリのインストール

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

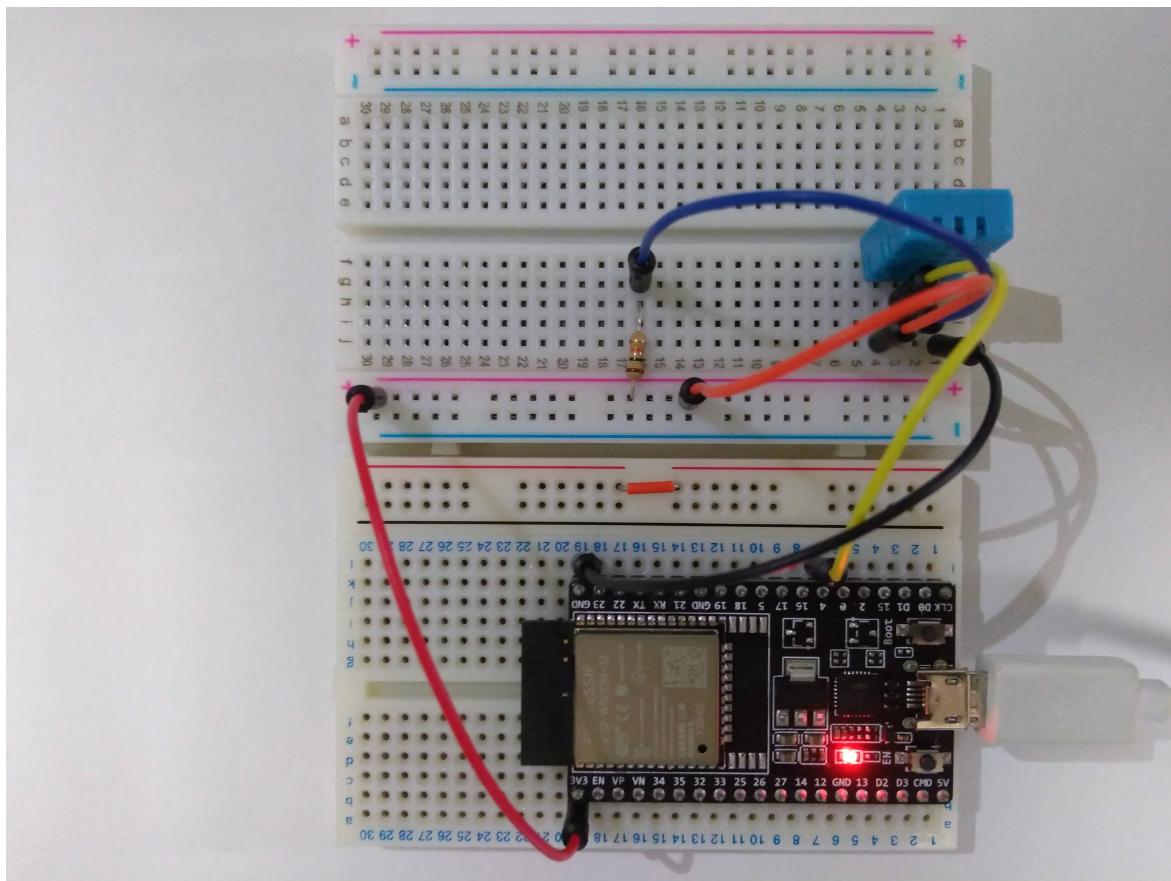


図 4.6: DHT11 回路配置図

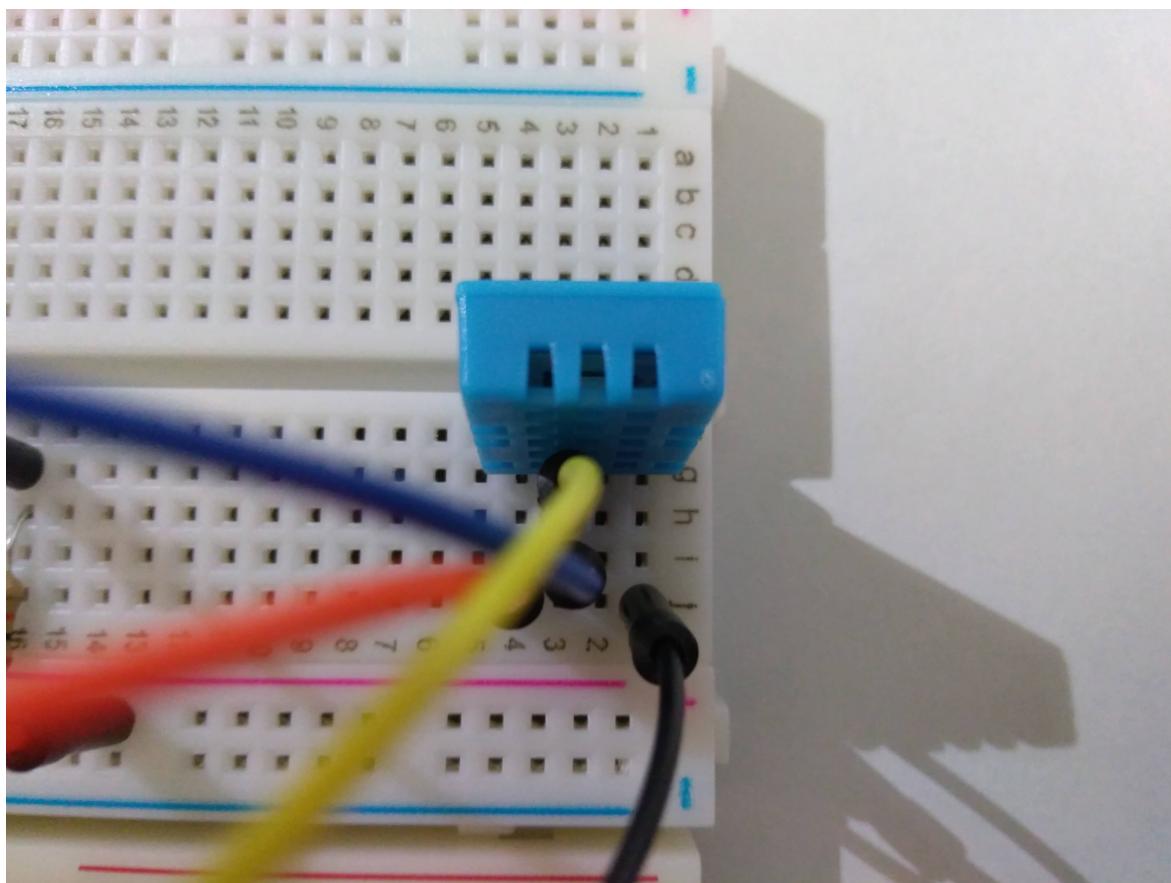


図 4.7: DHT11 アップ図

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てきます。「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: 59.00% 体感温度: 18.65
温度: 23.80 湿度: 59.00% 体感温度: 18.65
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/>

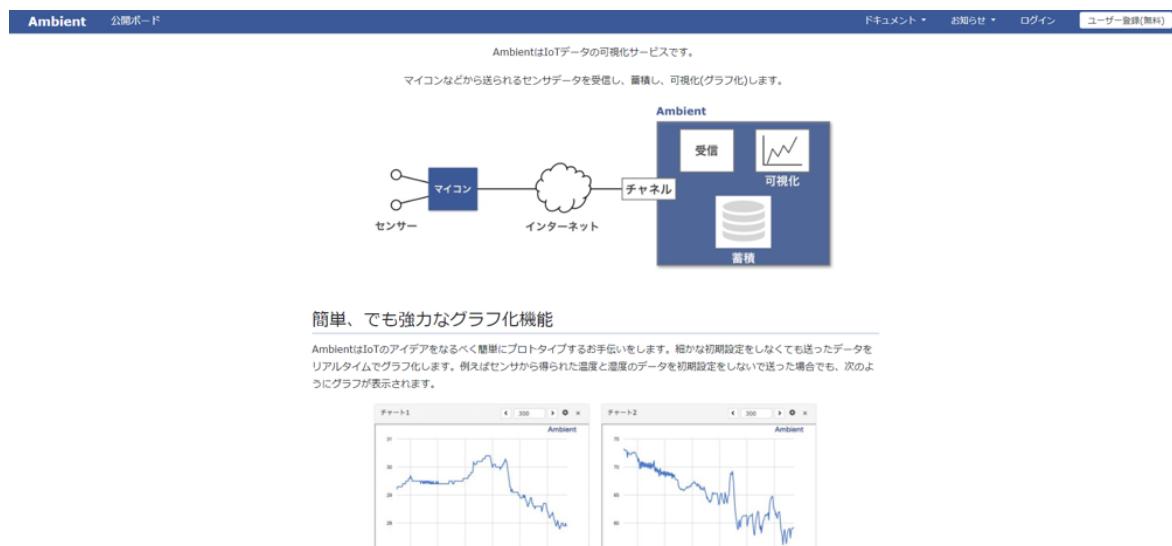


図 4.8: Ambient のトップページ

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

Ambient を利用するために、まずユーザ登録をします。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。

The screenshot shows the Ambient user registration form. At the top, there is a navigation bar with 'Ambient' and '公開ボード' on the left, and 'ドキュメント' (Documentation), 'お問い合わせ' (Contact), 'ログイン' (Login), and 'ユーザ登録(無料)' (User Registration Free) on the right. The main area contains 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 button labeled 'ユーザ登録(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 and conditions of the Ambient Use Policy). At the bottom of the page, there are links for 'AmbientData Inc.' and '利用規約' (Terms of Use) and '会社概要' (About Us).

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。

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



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。

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



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。

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

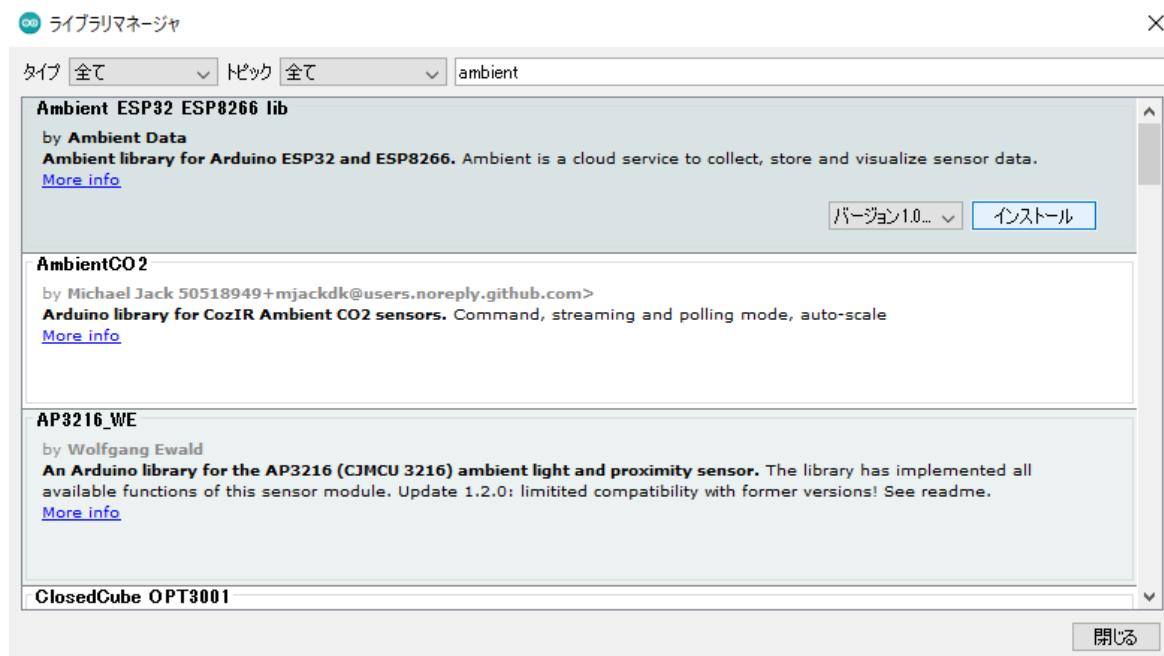


図 4.13: ambient 用ライブラリのインストール

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

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

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、30秒ごとにDHT11より取得したデータをAmbientに送信します。

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

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。

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



図 4.14: Ambient に表示されるグラフ

第 5 章

WebAPI を使おう

この章では WebAPI を利用して天気情報をディスプレイに表示することを目的とします。

WebAPI とは？

API とは Application Programming Interface の略です。API はコンテキストによって様々な意味合いで用いられますが、主にアプリケーションが何らかの情報を得たり、渡したりする際に用いる窓口を指す際に用いられています。そのため WebAPI は Web を通してやり取りを行う API を指しています。身近な例としては「Twitter に共有」というリンクが設置されていることがあります、これはアプリケーションが Twitter の API を通して Twitter 側に情報を送っています。

5.1 Weather API を使う

今回は WebAPI の中でも天気を知ることができる WebAPI を用います。天気を知ることができる WebAPI は複数ありますが、この章では「Weather API」というものを利用します。以下のリンクにアクセスして

Weather API のトップページに移動してください（図 5.1）。

Weather API: <https://www.weatherapi.com/>

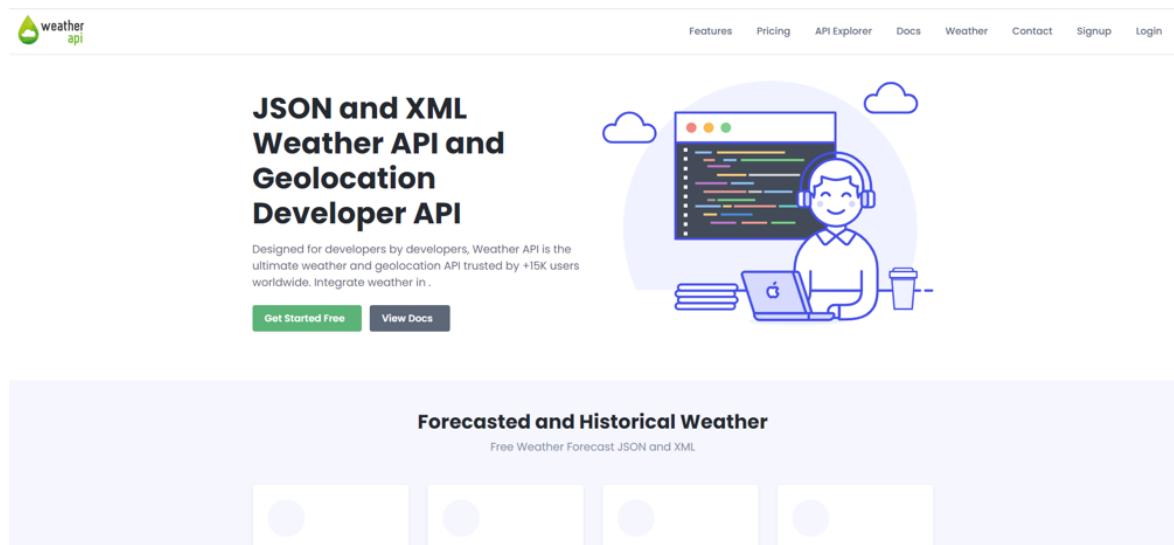
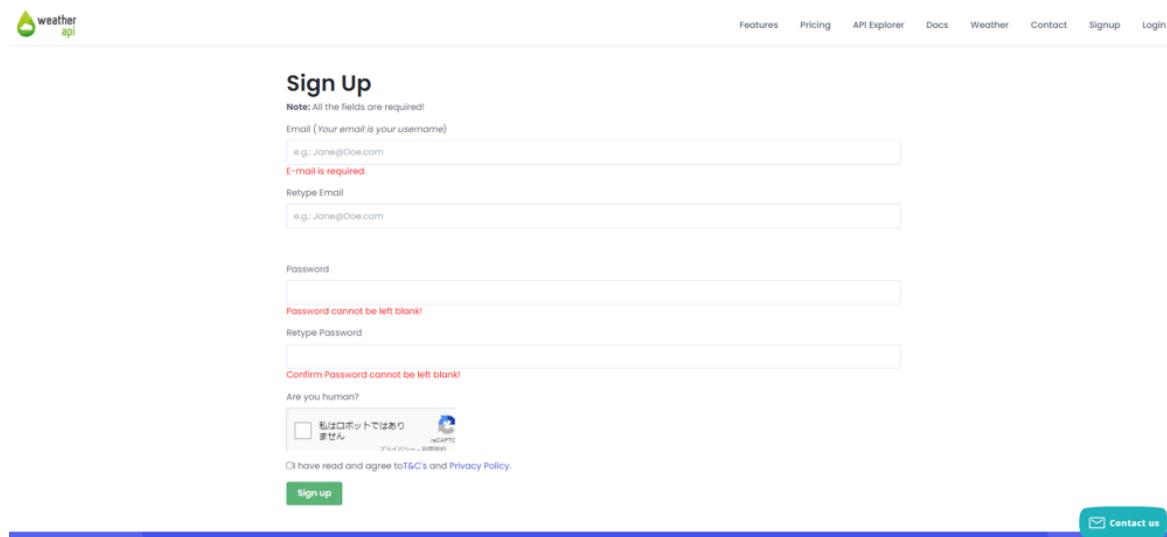


図 5.1: WeatherAPI のトップページ

Weather API を利用するためには、ユーザ登録をします。Signup を選択してユーザ登録画面に移動してください（図 5.2）。登録をするために、メールアドレスとパスワードを入力し Sign up を選択してください。

第5章 WebAPI を使おう

5.1 Weather API を使う



The screenshot shows the 'Sign Up' form for the Weather API. It includes fields for Email, Password, Retype Password, and a CAPTCHA section. A note at the top states 'Note: All the fields are required!'. Error messages appear in red for empty fields: 'Email (Your email is your username)' and 'Password cannot be left blank!'. Below the password fields, another error message says 'Confirm Password cannot be left blank!'. The CAPTCHA asks 'Are you human?' with two options: '私はロボットではありません' (checkbox) and 'I have read and agree to T&C's and Privacy Policy.' (checkbox). A 'Sign up' button is at the bottom, and a 'Contact us' link is on the right.

図 5.2: 登録画面

ユーザ登録が完了すると登録メールアドレスに認証メールが届くので（図 5.3）リンク先にアクセスをしてユーザ登録を完了してください。

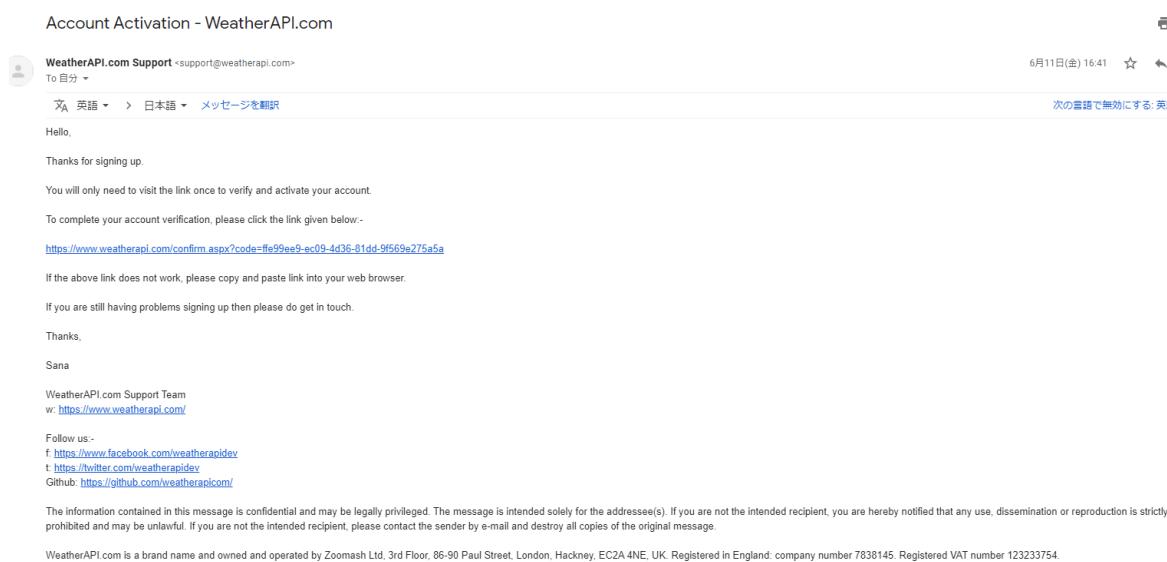


図 5.3: 認証メール

その後、登録情報をもとにログインをしてください（図 5.4）。

第5章 WebAPI を使おう

5.1 Weather API を使う

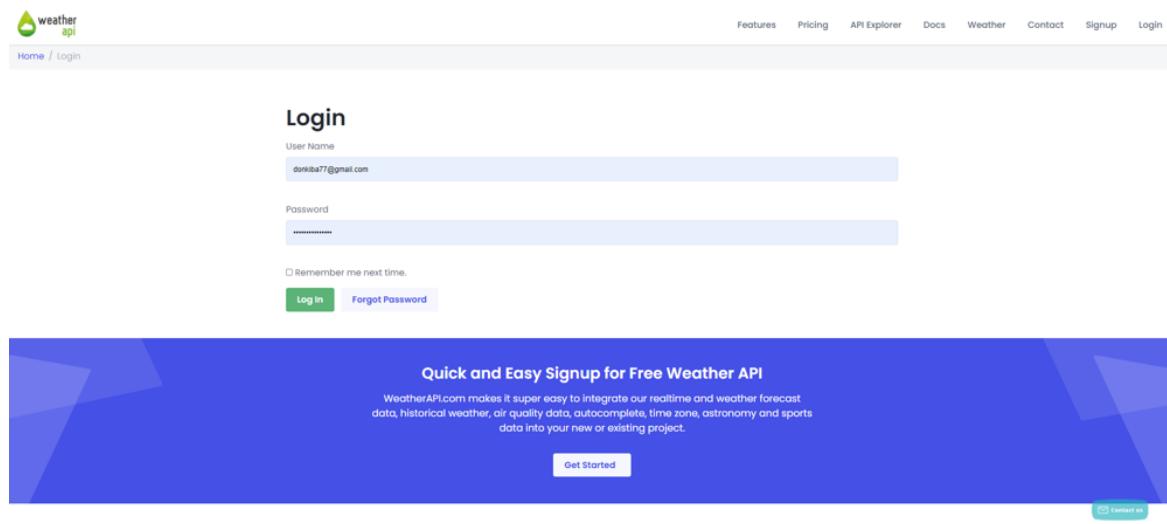


図 5.4: ログイン画面

ログイン後はマイページに遷移します（図 5.5）。この際ページ上部の

- API key

は Weather API を使用する際の認証に使うのでメモをしておいてください。次に Weather API の仕様を確認するために API Explorer を選択してください。

第5章 WebAPI を使おう

5.1 Weather API を使う

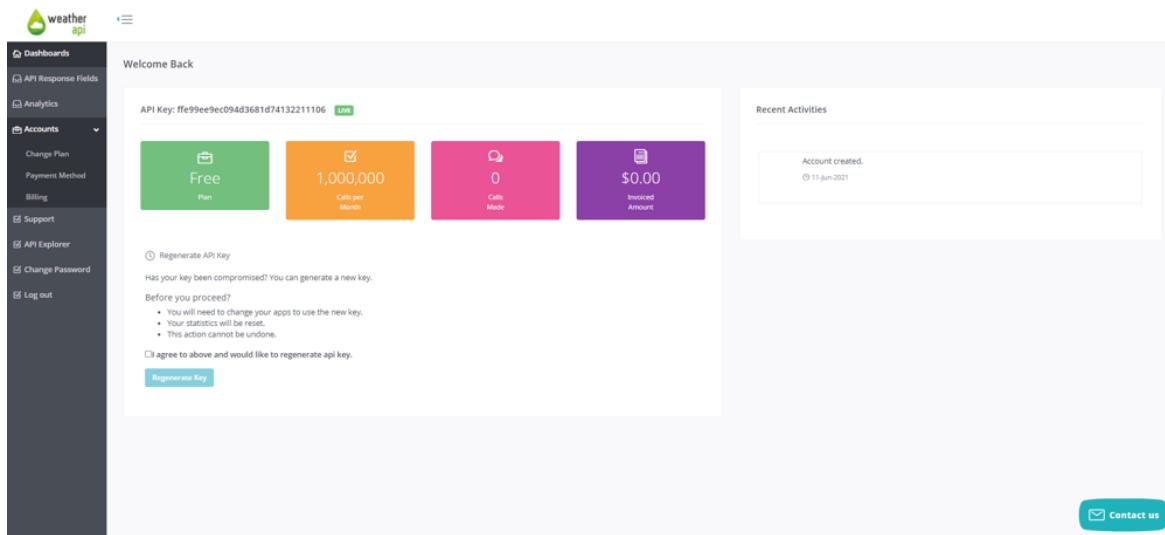


図 5.5: マイページ

選択した後、図 5.6 のような画面に遷移します。

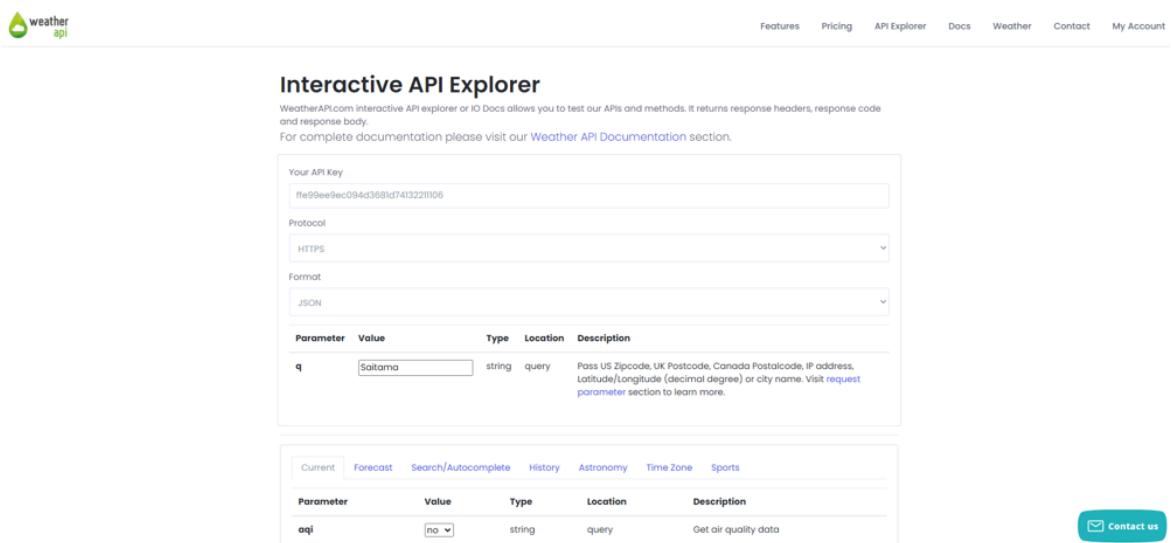


図 5.6: API の試行ページ

この画面では以下の 5 つのパラメータを設定することができます。以下を参考にパラメータを設定してください。

- Your API Key

- 先ほどメモした API key
- Protocol
 - HTTP
- Format
 - JSON
- Parameter q
 - London
- Parameter api
 - no

Protocol は API との通信に用いるプロトコルを指定して Format はやり取りをする情報の形式を表しています。HTTP と JSON の詳細については後に説明します。パラメータを選択したのち Show Response を選択すると以下のレスポンスが帰ってきます。リスト 5.1 は API を呼び出すときに用いる URL です。key の部分には Weather API で得られた API Key を入れてください。リスト 5.2 とリスト 5.3 とリスト 5.4 は API との通信で HTTP というプロトコルを使用した際に帰ってくるもの（レスポンス）です。これについては後に説明します。また、リスト 5.3 とリスト 5.4 は API で呼び出した指定した地域の天気情報です。これは先ほど指定した JSON という形式で送られています。これもまた後に説明します。

レスポンス

リスト 5.1: Call

```
http://api.weatherapi.com/v1/current.json?key=*****&q=Saitama&aqi=no
```

リスト 5.2: ResponseCode

```
200
```

リスト 5.3: ResponseHeader

```
{
    "Transfer-Encoding": "chunked",
    "Connection": "keep-alive",
    "Vary": "Accept-Encoding",
    "CDN-PullZone": "93447",
    "CDN-Uid": "8fa3a04a-75d9-4707-8056-b7b33c8ac7fe",
    "CDN-RequestCountryCode": "FI",
    "CDN-EdgeStorageId": "615",
    "CDN-CachedAt": "2021-07-12 14:05:36",
    "CDN-RequestPullSuccess": "True",
    "CDN-RequestPullCode": "200",
    "CDN-RequestId": "a45be49d32c7a76559a3f3920d337f53",
    "CDN-Cache": "MISS",
    "Cache-Control": "public, max-age=180",
    "Content-Type": "application/json",
    "Date": "Mon, 12 Jul 2021 12:05:36 GMT",
    "Server": "BunnyCDN-FI1-615"
}
```

リスト 5.4: ResponseBody

```
{
    "location": {
        "name": "Saitama",
        "region": "Saitama",
        "country": "Japan",
        "lat": 35.91,
```

```
"lon": 139.66,
"tz_id": "Asia/Tokyo",
"localtime_epoch": 1626091536,
"localtime": "2021-07-12 21:05"
},
"current": {
    "last_updated_epoch": 1626087600,
    "last_updated": "2021-07-12 20:00",
    "temp_c": 29.4,
    "temp_f": 84.9,
    "is_day": 0,
    "condition": {
        "text": "Partly cloudy",
        "icon": "//cdn.weatherapi.com/weather/64x64/night/116.png",
        "code": 1003
    },
    "wind_mph": 7.6,
    "wind_kph": 12.2,
    "wind_degree": 162,
    "wind_dir": "SSE",
    "pressure_mb": 1010.0,
    "pressure_in": 30.3,
    "precip_mm": 0.0,
    "precip_in": 0.0,
    "humidity": 61,
    "cloud": 47,
    "feelslike_c": 32.1,
    "feelslike_f": 89.8,
    "vis_km": 10.0,
    "vis_miles": 6.0,
    "uv": 7.0,
    "gust_mph": 9.2,
    "gust_kph": 14.8
}
}
```

JSON (JavaScript Object Node)

JSON はデータを記述する際に用いられるデータ記述言語の一つで、JavaScript でオブジェクトを記述する際の記法をもとにしています。

リスト 5.5: JSON データの例

```
{  
    "THEToilet" : unko,  
    "DNE3" : "芝浦"  
}
```

JSON で使えるデータ型は文字列、数値、配列、Boolean 等が挙げられます。API の通信ではデータのやり取りで JSON を使うことが多く、またゲームのセーブデータや 3D モデルのデータファイル形式としても使われことがあります。

HTTP (Hypertext Transfer Protocol)

HTTP とは TCP/IP と呼ばれるをインターネットなどのネットワークで使われている、通信プロトコル上で Web 情報をやり取りする際に用いられるプロトコルです。例えば電子計算機研究会(通称: DEN3)のホームページにアクセスした際も図 5.7 のように HTTP 通信が行われています。

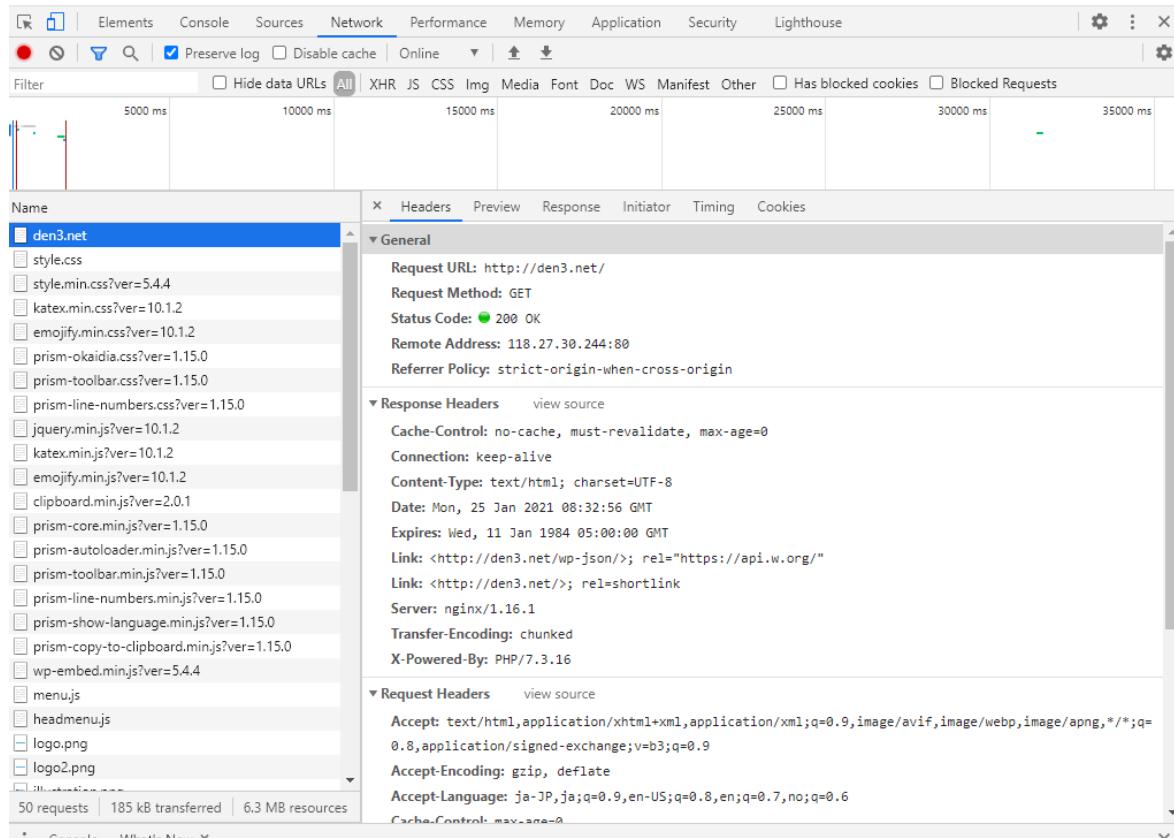


図 5.7: DEN3 のホームページにアクセスした際の通信

HTTP はリクエスト/レスポンス型の通信プロトコルであり、クライアント側は通信相手にリクエスト（要求）を出すと、通信相手からのレスポンス（応答）が返ってくるまで待機します。ちなみに HTTPS は HTTP の通信を暗号化しているプロトコルです。また、HTTP の通信のやり取りには HTTP メッセージというものが使われてあり Weather API とのやり取りでも出てきた

- ステータスコード
- ヘッダ
- ボディ

の 3 つと

- HTTP メソッド

が主に重要になるので、これらの紹介を行います。

ステータスコード

ステータスコードとは HTTP 通信にてリクエストを送った後、そのリクエストが成功したかどうかを表している数字です。主に 3 衔の数字で表され、それぞれの数字の意味としては

- 2xx 200 番台
 - リクエストが成功したことを表す。
 - EX: 200 OK (リクエスト成功)
- 3xx 300 番台
 - リダイレクト (別のアクセスへの転送) を行うときに返ってくる。
 - EX: 301 Moved Permanently (アクセス先の URL が変更したため転送する)
- 4xx 400 番台
 - クライアント側のエラーを表す。
 - クライアントが送った処理が間違っている時に返ってくる。
 - EX: 404 not found (指定したコンテンツが見つかりませんでした)
- 5xx 500 番台
 - サーバ側のエラーを表す。
 - サーバ内でエラーが発生している。
 - EX: 500 Internal Server Error (サーバ内部エラー)

を表しています。

ヘッダ

ヘッダには HTTP メッセージについてのメタデータ（データについての付加情報）が書かれています。例としてはボディに記述されているコンテンツの種類（HTML、CSS（Web ページを表しているデータ）や JSON など）やコンテンツのサイズ、後述する HTTP メソッド等が記述されています。

ボディ

やり取りを行うコンテンツ（データ）が入っています、主に JSON 化したデータや HTML、CSS などです。

HTTP メソッド

HTTP 通信にはメソッドと呼ばれている HTTP リクエストの種類を表すものがあります。ここでは CURD（Create, Update, Read, Delete）と呼ばれる性質を満たす代表的なメソッド

- GET
- POST
- PUT
- DELETE

の四つを紹介したいと思います。

GET コンテンツの取得の際に使われます。図 5.8 はブログの記事を取得した例で、リクエストメソッドとして GET が使われています。

▼ General

Request URL: http://den3.net/activity_diary/2021/01/31/2752/

Request Method: GET

Status Code: 200 OK

Remote Address: 118.27.30.244:80

Referrer Policy: strict-origin-when-cross-origin

図 5.8: HTTP 通信で GET

POST コンテンツの追加、作成時に使われます。

DEN3 ブログにコメントを投稿（図 5.9）した際は、リクエストメソッドとして POST が使われ（図 5.10）、HTTP リクエストのボディにはコメントの内容が書かれており（図 5.11）、コメントの内容とともにサーバにリクエストを送信しています。

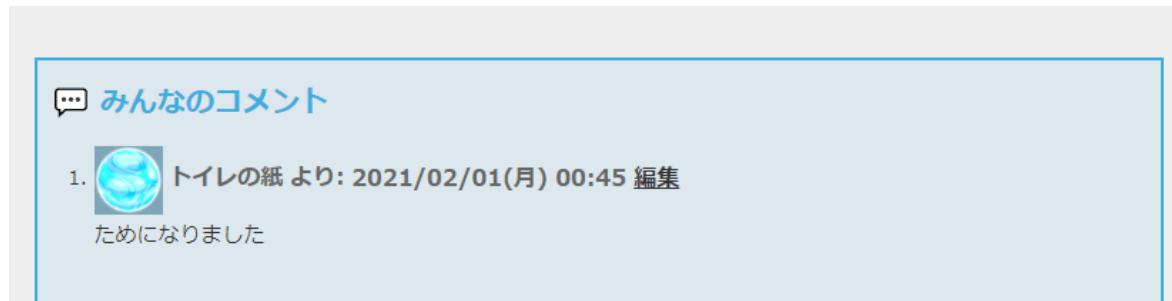


図 5.9: ブログに送信したコメント

▼ General

Request URL: <http://den3.net/wp-comments-post.php>

Request Method: POST

Status Code: 302 Found

図 5.10: HTTP 通信で POST

▼ Form Data [view source](#) [view URL encoded](#)

comment: ためになりました
submit: 送信

図 5.11: コメントを送信した際のボディ部

PUT コンテンツの作成、更新時に使われ、ブログで例えるとブログを更新したいときに使われています。

DELETE コンテンツの削除時に使われ、ブログで例えるとブログの投稿などを削除した時に使われます。

ESP32 で JSON を利用する

JSON 形式のデータが WeatherAPI から返ってくるため、これを ESP32 側で解釈してデータとして扱えるようにする必要があります。そこで、公開されているライブラリである ArduinoJSON を利用します。ArduinoJSON を使用する場合には JSON データを ESP32 で扱えるデータに変換する際のキャパシティの計算をする必要があります。そのため以下のリンクにアクセスして設定を行ってください。ただし、今回使う Weather API 用のキャパシティの計算結果は用意してあるので参考として紹介します。

ArduinoJson Assistant: <https://arduinojson.org/v6/assistant/>

リンクにアクセスすると図 5.12 に遷移します。

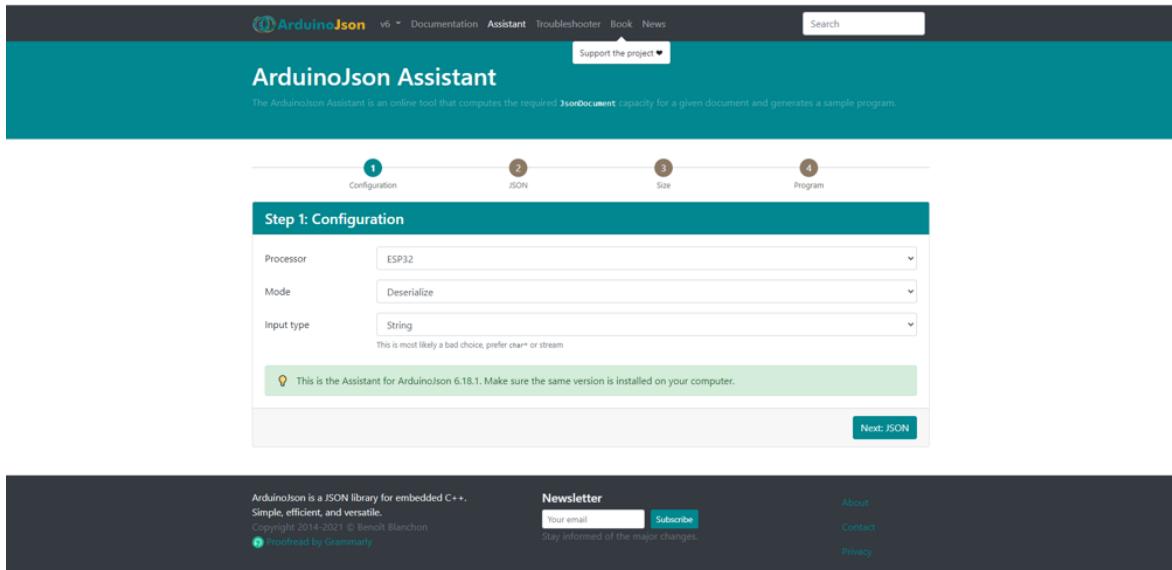


図 5.12: ArudinoAssistant のトップページ

ここで設定には

- Processor
 - ESP32
- Mode
 - Deserialize (データ構造を復元する処理)
- Input Type
 - String (入力時のデータ形式)

を設定してください。その後、Next: JSON を選択してください。

遷移後は図 5.13 のような画面になるので Input に使用する JSON データを入力してください。次に Next: Size を選択してください。

第5章 WebAPI を使おう

5.1 Weather API を使う

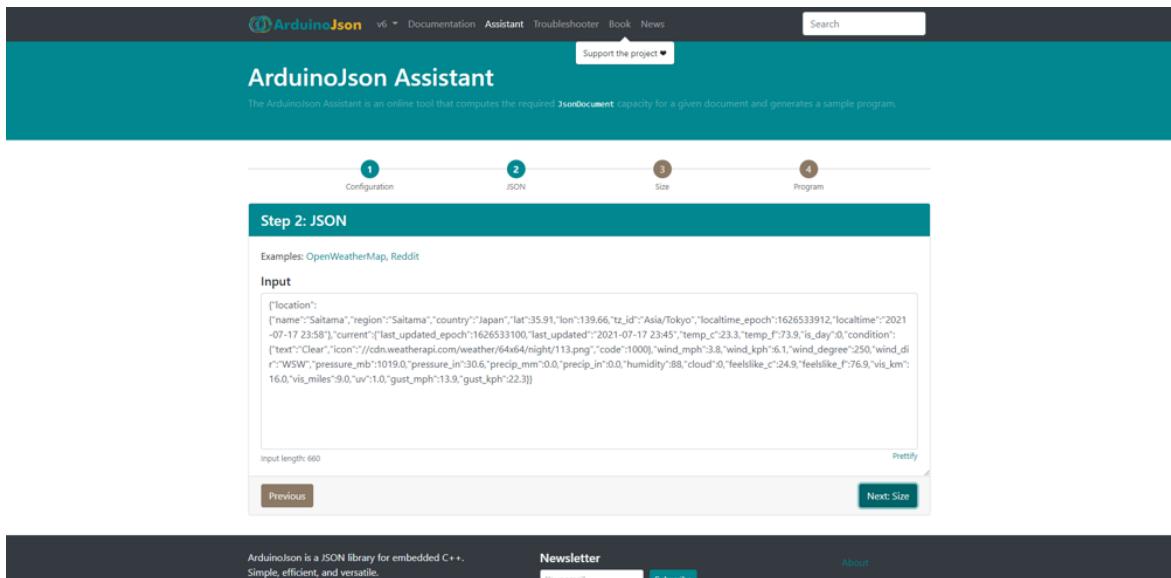


図 5.13: JSON の大きさ設定

遷移後の図 5.14 でデシリアライズに必要なサイズが分かります。

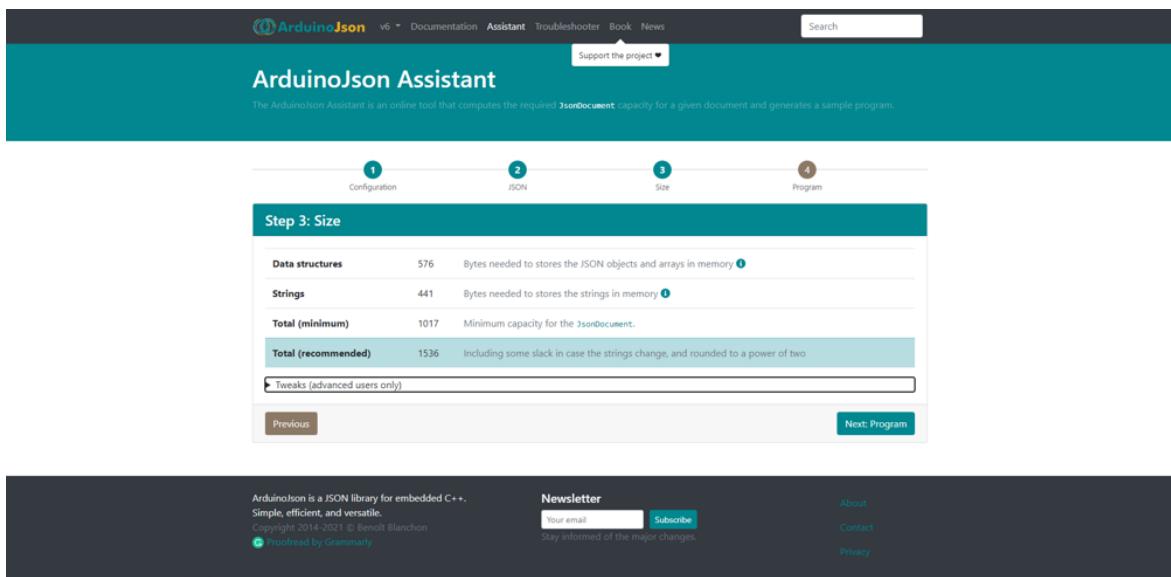


図 5.14: JSON のサイズ確認画面

リスト 5.6 が Weather API を ESP32 で使用するためのコードです。input には Weather API から送られてきたデータを代入してください。

リスト 5.6: デシリアライズコード

```
// String input;

StaticJsonDocument<1536> doc;

DeserializationError error = deserializeJson(doc, input);

if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return;
}

JsonObject location = doc["location"];
const char* location_name = location["name"]; // "Saitama"
const char* location_region = location["region"]; // "Saitama"
const char* location_country = location["country"]; // "Japan"
float location_lat = location["lat"]; // 35.91
float location_lon = location["lon"]; // 139.66
const char* location_tz_id = location["tz_id"]; // "Asia/Tokyo"
// 1626533912
long location_localtime_epoch = location["localtime_epoch"];
// "2021-07-17 23:58"
const char* location_localtime = location["localtime"];

JsonObject current = doc["current"];
// 1626533100
long current_last_updated_epoch = current["last_updated_epoch"];
// "2021-07-17 23:45"
const char* current_last_updated = current["last_updated"];
float current_temp_c = current["temp_c"]; // 23.3
float current_temp_f = current["temp_f"]; // 73.9
int current_is_day = current["is_day"]; // 0

JsonObject current_condition = current["condition"];
// "Clear"
const char* current_condition_text = current_condition["text"];
const char* current_condition_icon = current_condition["icon"];
int current_condition_code = current_condition["code"]; // 1000
```

```
float current_wind_mph = current["wind_mph"]; // 3.8
float current_wind_kph = current["wind_kph"]; // 6.1
int current_wind_degree = current["wind_degree"]; // 250
const char* current_wind_dir = current["wind_dir"]; // "WSW"
int current_pressure_mb = current["pressure_mb"]; // 1019
float current_pressure_in = current["pressure_in"]; // 30.6
int current_precip_mm = current["precip_mm"]; // 0
int current_precip_in = current["precip_in"]; // 0
int current_humidity = current["humidity"]; // 88
int current_cloud = current["cloud"]; // 0
float current_feelslike_c = current["feelslike_c"]; // 24.9
float current_feelslike_f = current["feelslike_f"]; // 76.9
int current_vis_km = current["vis_km"]; // 16
float current_gust_mph = current["gust_mph"]; // 13.9
float current_gust_kph = current["gust_kph"]; // 22.3
```

JSON のライブラリをインストールする

JSON を ESP32 上で使うためにライブラリを Arduino IDE にインストールします。図 5.15 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。

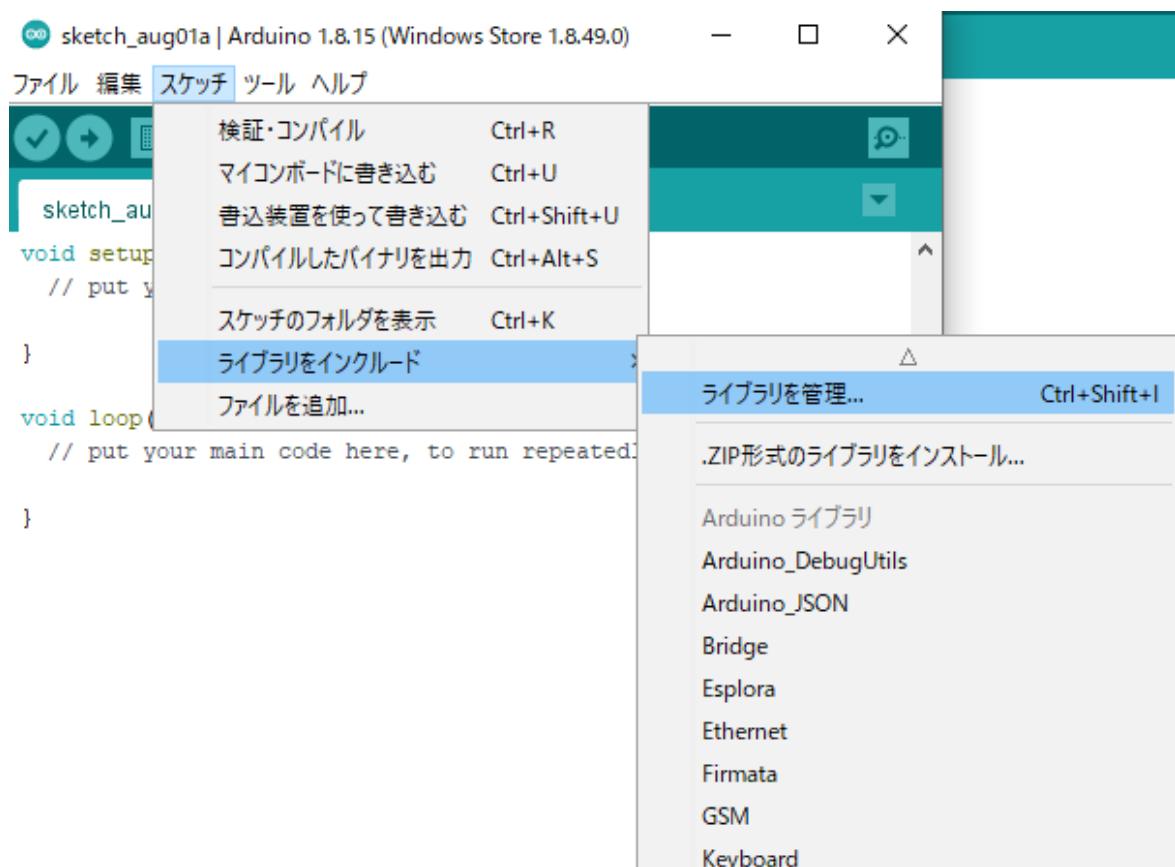


図 5.15: ライブライの管理の選択

選択するとライブライマネージャーが開かれるので、検索窓に「ArduinoJSON」を入力してください(図 5.16)。その後、「ArduinoJson」をインストールしてください

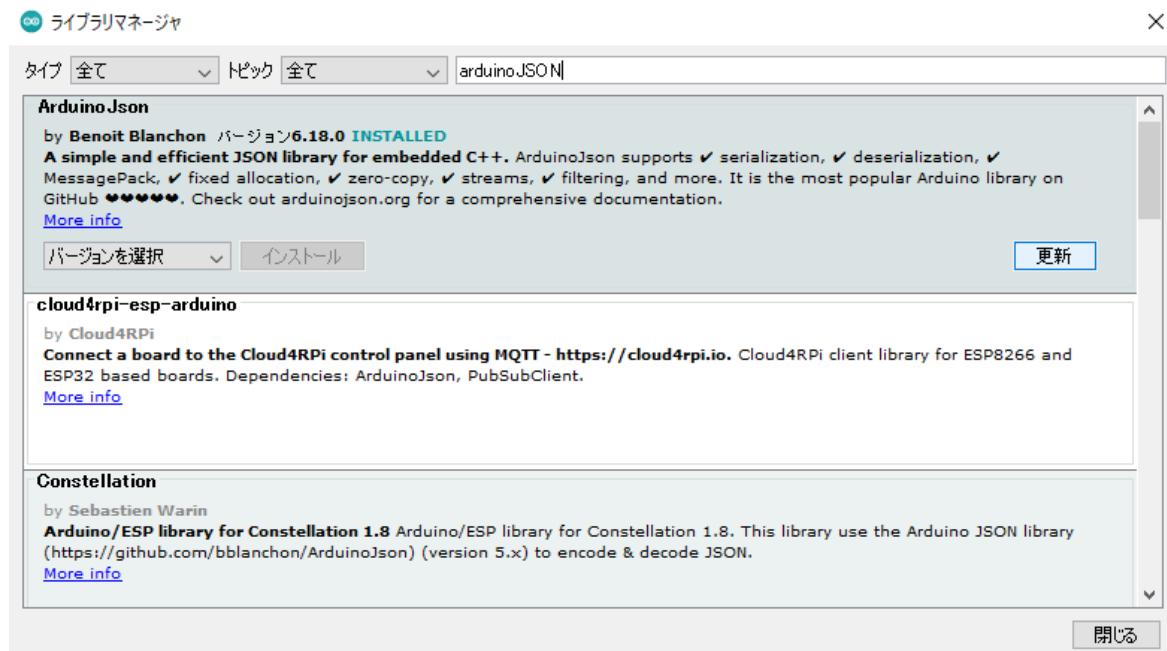


図 5.16: ArduinoJson 用ライブラリのインストール

Weather API からデータを取得する

ここで、実際に WeatherAPI からデータを取得してみます。リスト 5.7 のプログラムを ESP32 に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
- API Key
 - Weather API に用いる API Key
 - 変数名: api_key
- Location (地名)
 - Weather API で取得したい地名

- 変数名: location

リスト 5.7: Weaher API との通信

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// WiFi接続用変数
const char *ssid = "elecom-3230d1-s";
const char *password = "fwp342pf3c3u";

// WeatherAPI用変数
const String api_key = "ffe91ee9ec093d313683d7433221106";
const String location = "Saitama";

struct Weather {
    const char *region;
    float temperature;
    int humidity;
};

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");
}

void loop()
{
    HTTPClient http;
    String target_url = "https://api.weatherapi.com/v1/current.json";
```

```
target_url += ("?key=" + api_key + "&q=" + location + "&aqi=no");
http.begin(target_url); // HTTP通信を開始する

int http_code = http.GET(); // HTTP通信でGETする

Serial.printf("status code : %d\n", http_code);
if (http_code > 0) // HTTP通信が失敗すると負値になる
{
    if (http_code == HTTP_CODE_OK) // HTTPコードが200の場合成功
    {
        // HTTPのレスポンスボディを取得
        String payload = http.getString();
        Serial.println(payload);
        // WeatherAPIのJSONをパースする
        Weather weather = parse(payload);
        Serial.println("-----weather-----");
        Serial.println(weather.region);
        Serial.println(weather.temperature);
        Serial.println(weather.humidity);
    }
    else if (http_code > 500) {
        Serial.printf("Server Error: %d", http_code);
    }
    else if (http_code > 400) {
        Serial.printf("Client Error: %d", http_code);
    }
}
else
{
    Serial.println(http.errorToString(http_code).c_str());
}
http.end(); // HTTP通信の終了
delay(60000);
}

Weather parse(String input)
{
    Serial.println("parse.....");
    // JSONをパースするための領域を作成
    StaticJsonDocument<1536> doc;
```

```
// JSONをパースする
DeserializationError error = deserializeJson(doc, input);

if (error) // パースに失敗すると呼ばれる
{
    Serial.print(F("deserializeJson() failed: "));
    // F()マクロは、指定した文字列分がSRAMからFlashメモリに移動する。
    Serial.println(error.f_str());
    Weather weather = {"", 0, 0};
    return weather;
}

JsonObject location = doc["location"];
const char *location_region = location["region"]; // "Saitama"

JsonObject current = doc["current"];
float current_temp = current["temp_c"]; // 23.3
int current_humidity = current["humidity"]; // 88%

Weather weather={location_region,current_temp,current_humidity};

return weather;
}
```

一分ごとに Weather API にアクセスし情報を取得しています。取得した情報をパースした後にリスト 5.8 のように表示しています。

リスト 5.8: Weather API との通信をシリアルモニタに表示

```
Waiting for Wi-Fi connection...
Connected to Wi-Fi
status code : 200
{"location": {"name": "Saitama", "region": "Saitama", "country": ...省略
parse.....
-----weather-----
Saitama
30.00
```

```
59
2021-08-11 17:00
status code : 200
{"location": {"name": "Saitama", "region": "Saitama", "country": ...省略
parse.....
-----weather-----
Saitama
30.00
59
2021-08-11 17:00
status code : 200
{"location": {"name": "Saitama", "region": "Saitama", "country": ...省略
parse.....
-----weather-----
Saitama
30.00
59
2021-08-11 18:15
```

5.2 ディスプレイを使う

ここで、取得した情報を手軽に確認するためにディスプレイを使ってみます。今回使うディスプレイ（図 5.17）は ESP32 との通信に I2C という通信方式を利用しているので、ますそちらを紹介します。

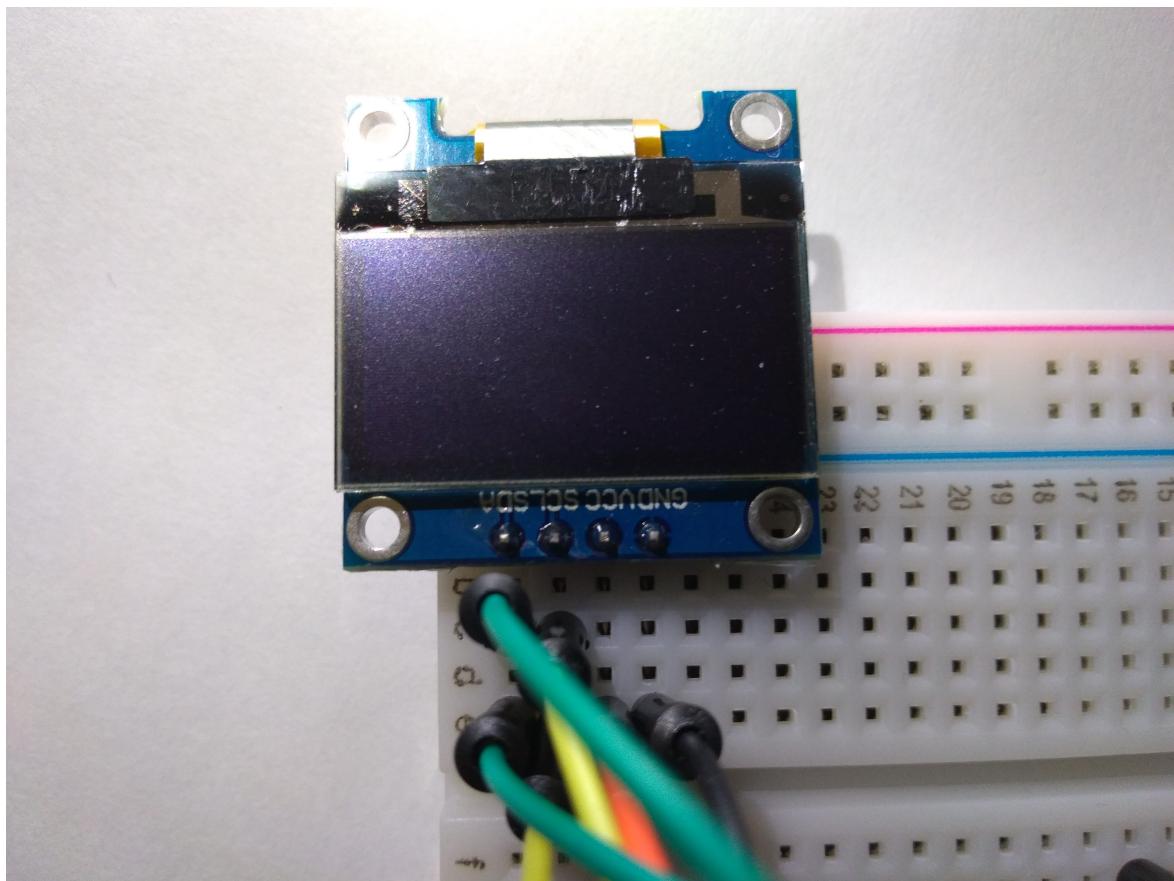


図 5.17: ディスプレイ

I2C (Inter-Integrated Circuit) とは

I2C はフィリップ社が開発したシリアル通信方式です。I2C に必要な通信線は 4 本で使用目的は電源 (Vdd)、GND、SDA (シリアルデータ)、SCL (シリアルクロック) です。SDA はデータの書き込みと読み込みを行い、SCL は通信先との同期をとる際に使用されます。I2C では通信するデバイスはマスタとスレーブに分類され、マスタがスレーブとの通信を管理します。ここでは ESP32 がマスタに相当し、ディスプレイがスレーブに相当します。また、マスタがスレーブを認識するためにスレーブにはそれぞれ認識アドレスが割り当てられます。

ライブラリのインストール

ディスプレイを ESP32 上で使うためにライブラリを Arduino IDE にインストールします。

図 5.15 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。

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「ssd1306 esp32」を入力してください（図 5.16）。その後、「ESP8266 ans ESP32 OLED driver for SSD1306 displays」をインストールしてください

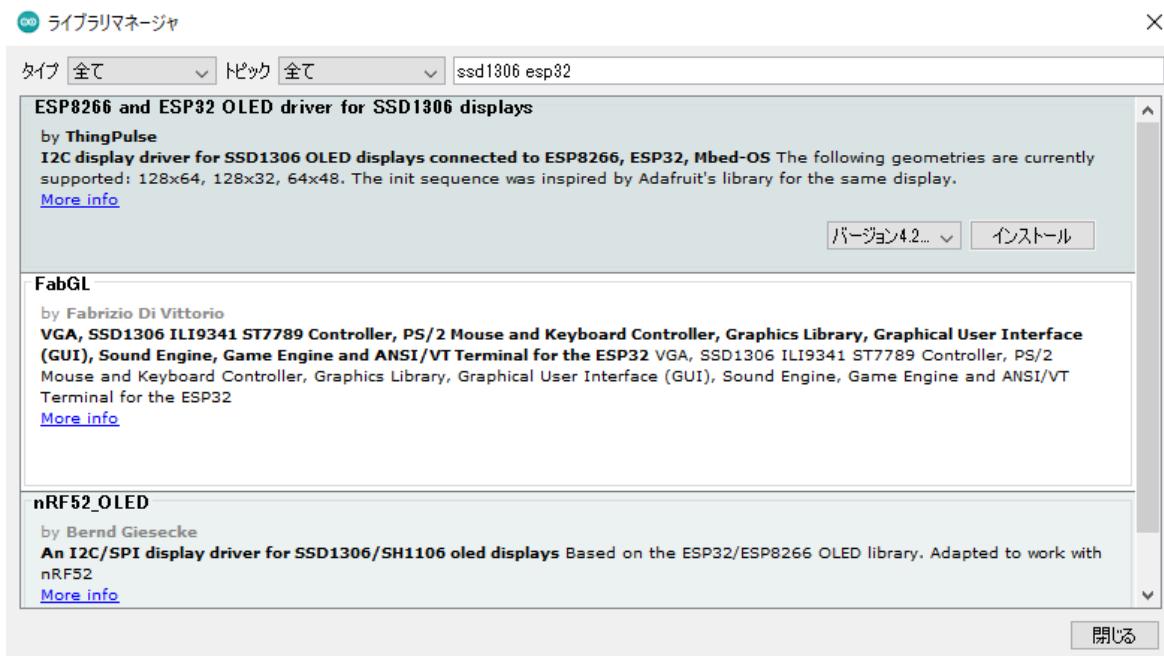


図 5.18: SSD1306 用ライブラリのインストール

ディスプレイの表示

ここで実際にディスプレイを表示させてみましょう。今回は Hello,World を表示させてみたいと思います。回路図（図 5.19）を

参考に電子回路を組み、プログラム（リスト 5.9）を書き込んでください。

回路図

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 2
 - 10k × 2
 - SSD1306 display × 1
 - ジャンプワイヤ

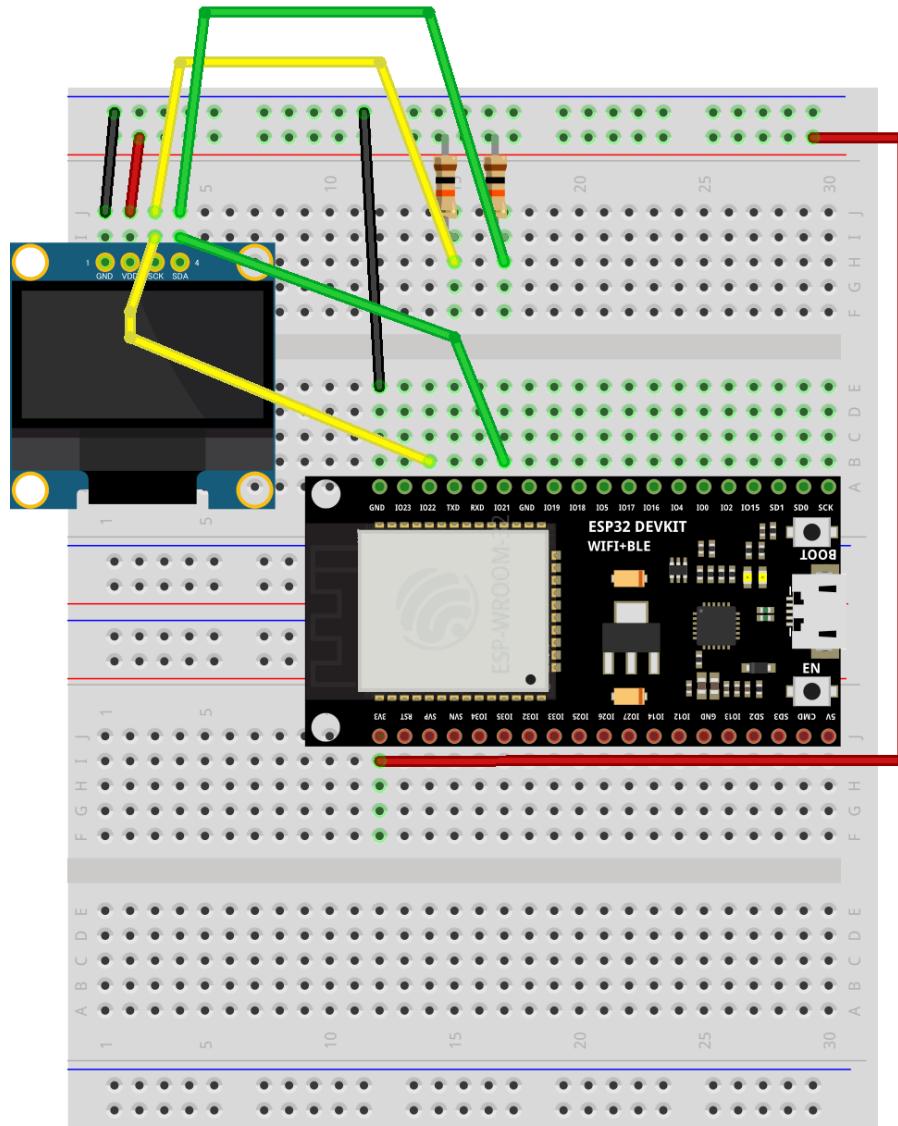


図 5.19: ディスプレイ表示回路図

リスト 5.9: ディスプレイ表示プログラム

```
#include <Wire.h> // I2Cを利用するためのライブラリ
#include "SSD1306.h"

SSD1306 display(0x3c, 21, 22);
```

```
// ディスプレイのインスタンスを作成する。  
// アドレス、 SDA、 SCLを指定  
  
void setup()  
{  
    display.init(); // ディスプレイの初期化  
    display.setFont(ArialMT_Plain_24); // フォントサイズ24pxで表示  
    // 左上を原点とした座標で  
    // (0,0) に"Hello,World"表示  
    display.drawString(0, 0, "Hello,World");  
    display.display(); // 指定した文字列を表示させる  
}  
  
void loop(){}

```

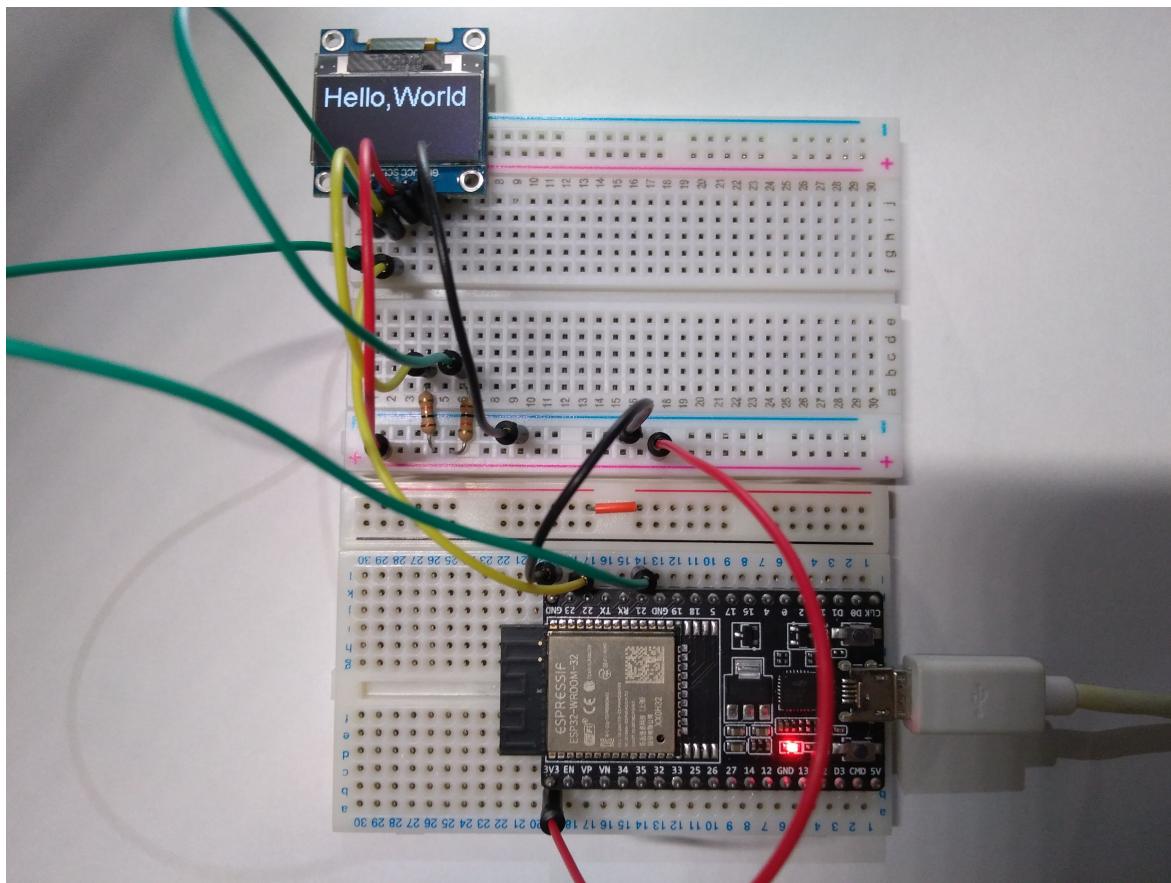


図 5.20: ディスプレイ回路配置図



図 5.21: Hello! ディスプレイ

5.3 Weather API から得たデータを表示

先ほど試した、Weather API からのデータをディスプレイに表示します。リスト 5.10 を参考にしてプログラム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

- API Key
 - Weather API に用いる API Key
 - 変数名: api_key
- Location (地名)
 - Weather API で取得したい地名
 - 変数名: location

リスト 5.10: I のデータをディスプレイに表示するプログラム

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

#include <Wire.h> // I2Cを利用するためのライブラリ
#include "SSD1306.h" // ディスプレイのライブラリ

SSD1306 display(0x3c, 21, 22);
// ディスプレイのインスタンスを作成する。
// アドレス、SDA、SCLを指定

// WiFi接続用変数
const char *ssid = "eledom-b2839f-g";
const char *password = "fad2d42pf313u";

// WeatherAPI用変数
const String api_key = "ffe92ee93c394d3681d74133211206";
const String location = "Saitama";

struct Weather {
    const char *region;
    float temperature;
    int humidity;
    const char *last_updated;
};

void setup()
{
```

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

// Wi-Fiアクセスポイントへ接続するまで待機
while (WiFi.status() != WL_CONNECTED)
{
    Serial.println("Waiting for Wi-Fi connection....");
    delay(500);
}
Serial.println("Connected to Wi-Fi");

display.init(); // ディスプレイの初期化
}

void loop()
{
    display.clear(); // ディスプレイの文字をすべて消す

    HTTPClient http;
    String target_url = "https://api.weatherapi.com/v1/current.json";
    target_url += ("?key=" + api_key + "&q=" + location + "&aqi=no");
    http.begin(target_url); // HTTP通信を開始する

    int http_code = http.GET(); // HTTP通信でGETする

    Serial.printf("status code : %d\n", http_code);
    if (http_code > 0) // HTTP通信が失敗すると負値になる
    {
        // HTTPコードが200の場合成功
        if (http_code == HTTP_CODE_OK)
        {
            // HTTPのレスポンスボディを取得
            String payload = http.getString();
            Serial.println(payload);
            // WeatherAPIのJSONをパースする
            Weather weather = parse(payload);
            Serial.println("-----weather-----");
            Serial.println(weather.region);
            Serial.println(weather.temperature);
            Serial.println(weather.humidity);
        }
    }
}
```

```
Serial.println(weather.last_updated);
// フォントサイズを10pxに設定
display.setFont(ArialMT_Plain_10);
// (x座標, y座標, 表示したい文字列)
display.drawString(0, 0, "region:");
display.drawString(0, 12, "temperature:");
display.drawString(0, 24, "humidity:");
display.drawString(0, 36, "last_updated:");
display.drawString(65, 0, weather.region);
display.drawString(65, 12, String(weather.temperature)+" °C");
display.drawString(65, 24, String(weather.humidity) + "%");
display.drawString(30, 48, weather.last_updated);

display.display(); // 設定した文字列をディスプレイに表示させる
}
else if (http_code > 500) {
    Serial.printf("Server Error: %d", http_code);
}
else if (http_code > 400) {
    Serial.printf("Client Error: %d", http_code);
}
else
{
    Serial.println(http.errorToString(http_code).c_str());
}
http.end(); // HTTP通信の終了
delay(60000);
}

Weather parse(String input)
{
    Serial.println("parse.....");
    StaticJsonDocument<1536> doc; // JSONをパースするための領域を作成
    // JSONをパースする
    DeserializationError error = deserializeJson(doc, input);

    if (error) // パースに失敗すると呼ばれる
    {
        Serial.print(F("deserializeJson() failed: "));
    }
}
```

```
// F()マクロは、指定した文字列分がSRAMからFlashメモリに移動する。
Serial.println(error.f_str());
Weather weather = {"", 0, 0};
return weather;
}

JsonObject location = doc["location"];
const char *location_region = location["region"]; // "Saitama"

JsonObject current = doc["current"];
// "2021-07-17 23:45"
const char *current_last_updated = current["last_updated"];
float current_temp = current["temp_c"]; // 23.3
int current_humidity = current["humidity"]; // 88%

Weather weather = {location_region, current_temp,
current_humidity, current_last_updated};

return weather;
}
```

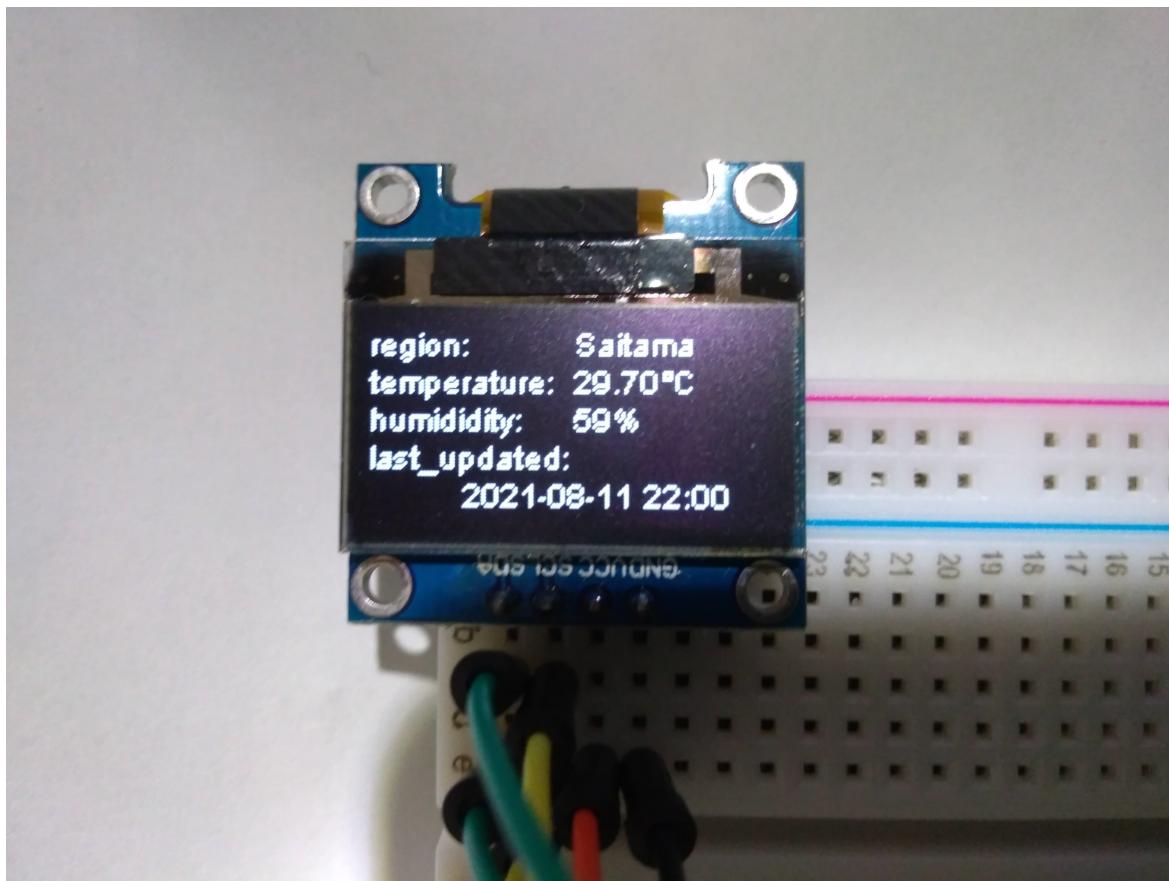


図 5.22: Weather API のデータをディスプレイに表示

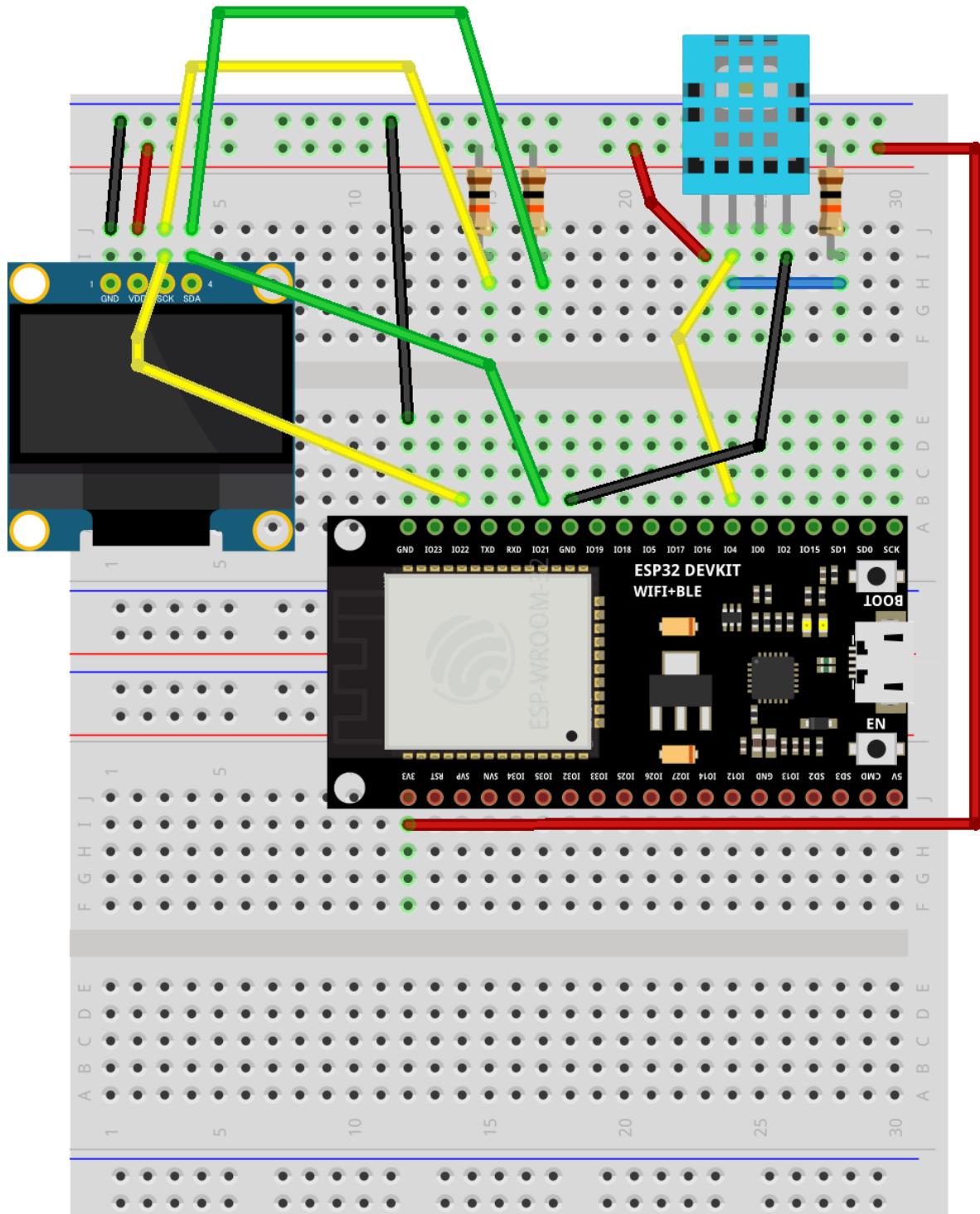
Weather API は 15 分に一回更新されますがここでは一分間に一回 Weather API へのリクエストを送っています。

5.4 応用 1: センサのデータを表示する

応用問題として、温湿度センサで得られた値をディスプレイに表示してみます。回路図(図 5.23)とプログラム(リスト 5.11)を参考にしてみてください。

- 必要材料

- ESP32 × 1
- ブレッドボード × 2
- 10k × 3
- SSD1306 display × 1
- DHT11 × 1
- ジャンプワイヤ



fritzing

図 5.23: 温湿度データをディスプレイに表示する回路図

リスト 5.11: DHT11 のデータをディスプレイに表示すえうプログラム

```
#include "DHT.h"

#include <Wire.h> // I2Cを利用するためのライブラリ
#include "SSD1306.h" // ディスプレイのライブラリ

#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

SSD1306 display(0x3c, 21, 22);
// ディスプレイのインスタンスを作成する。
// I2Cアドレス、SDA、SCLを指定

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
    display.init(); // ディスプレイの初期化
}

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);
    display.clear(); // ディスプレイの文字をすべて消す

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

    // 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);

display.setFont(ArialMT_Plain_10); // フォントサイズを10pxに設定
// (x座標, y座標, 表示したい文字列)
display.drawString(0, 0, "temperature");
display.drawString(0, 25, "humidity");
display.setFont(ArialMT_Plain_24); // フォントサイズを24pxに設定
// String()で文字列に変換
display.drawString(50, 10, String(temperature));
display.drawString(50, 30, String(humidity));
display.setFont(ArialMT_Plain_10); // フォントサイズを10pxに設定
display.drawString(110, 22, "° C");
display.drawString(110, 42, "%");

display.display(); // 設定した文字列をディスプレイに表示させる
}
```

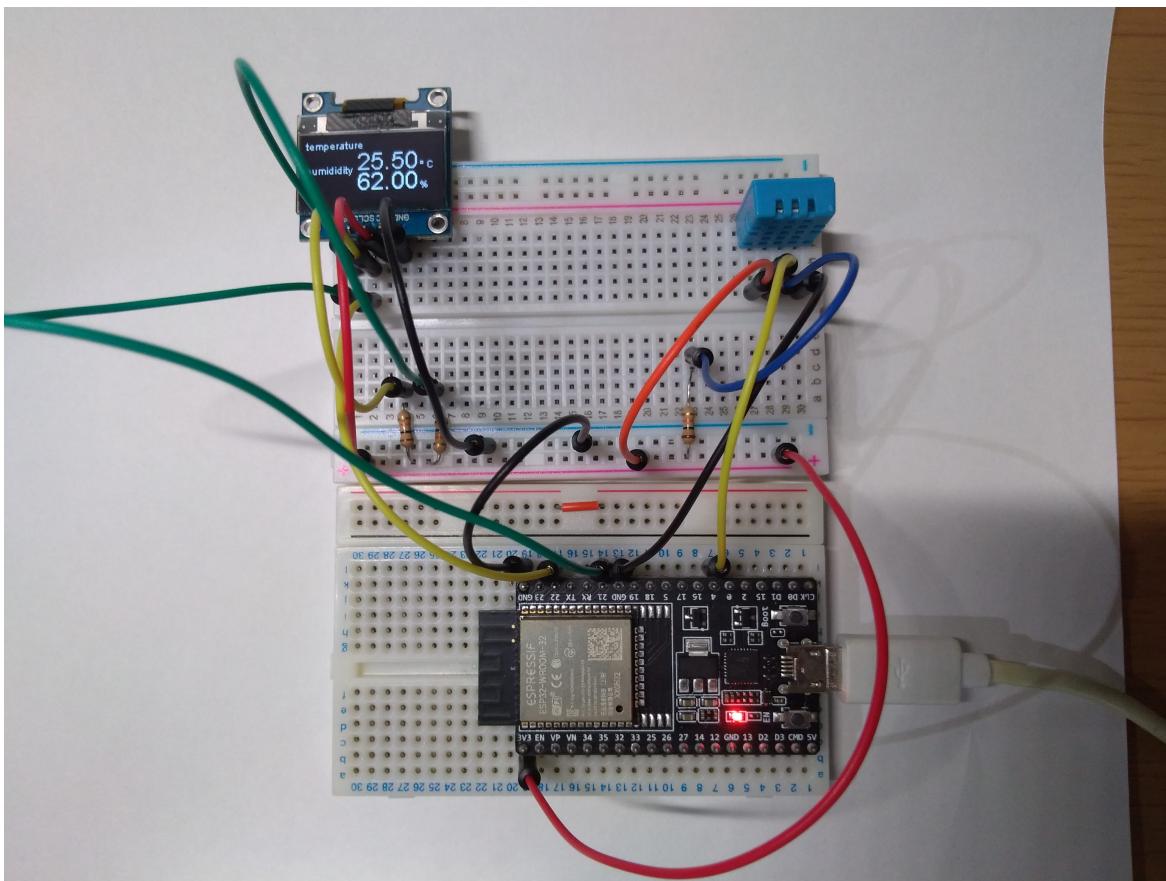


図 5.24: ディスプレイとセンサ配置図

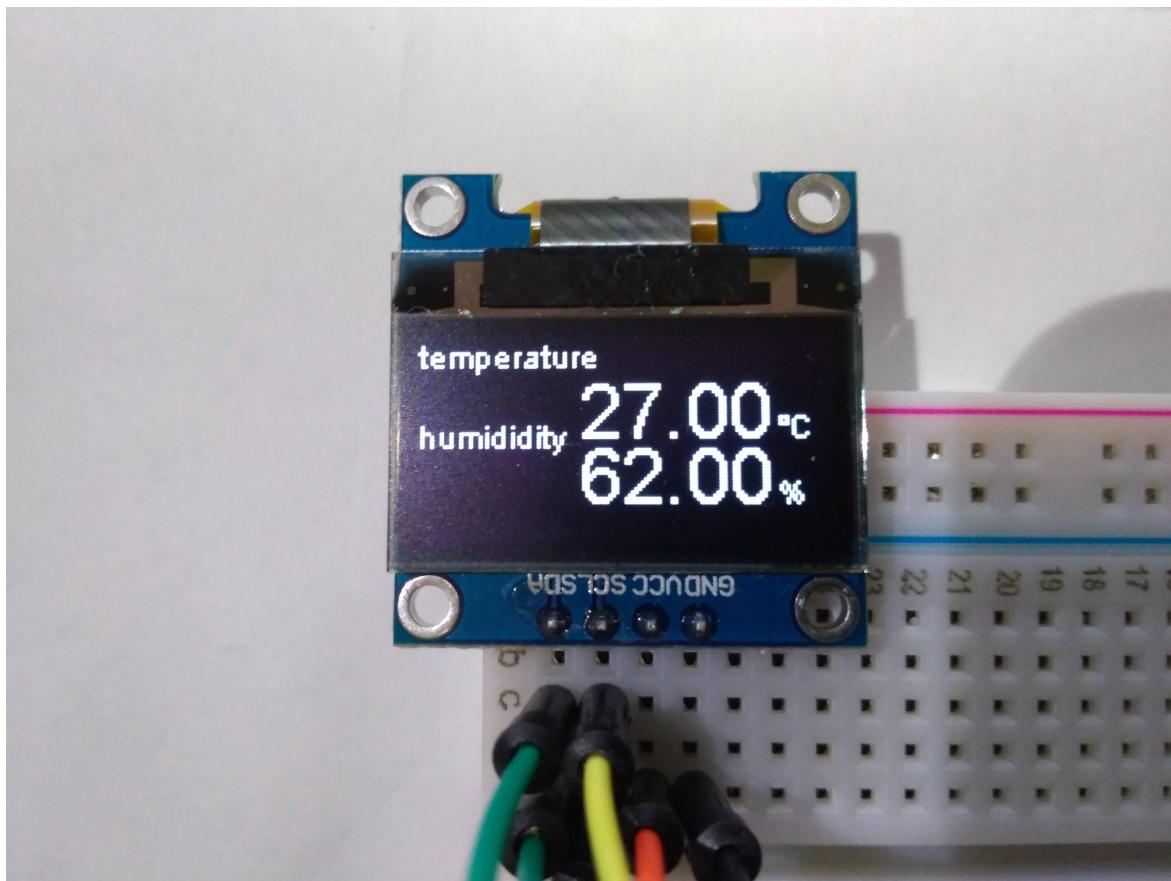


図 5.25: 温湿度をディスプレイに表示

5.5 応用 2: 時計を表示してみる

応用として時計を作成します。正確な時刻情報を取得するためには、NTP サーバにアクセスする必要があります。そのためまずは NTP サーバについての紹介を行います。

NTP (Network Time Protocol)

NTP は時刻の同期に用いられるプロトコルです。クライアントは主に原子時計や GPS を用いて時刻を管理している、NTP サーバに問い合わせを送ることで時刻を取得することができます。以下のプログラム（リスト

5.12) を参考にしてみてください。

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 2
 - 10k × 2
 - SSD1306 display × 1
 - ジャンプワイヤ

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
- API Key

リスト 5.12: 時刻表示プログラム

```
#include <WiFi.h>

#include <Wire.h> // I2Cを利用するためのライブラリ
#include "SSD1306.h" // ディスプレイのライブラリ

SSD1306 display(0x3c, 21, 22);
// ディスプレイのインスタンスを作成する。
// アドレス、SDA、SCLを指定

// WiFi接続用変数
const char *ssid = "el3c3m-223033f-g";
const char *password = "fa2d4rpaa33u";

int JST = 3600 * 9; // 日本標準時間
```

```
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");

    configTime(JST, 0, "ntp.nict.jp", "0.jp.pool.ntp.org",
    "time1.google.com");
    // 標準時間, サマータイム, ntpサーバ
    display.init(); // ディスプレイの初期化
}

// 時刻情報を入れる構造体
struct tm timeInfo;

void loop() {
    delay(1000);

    display.clear(); // ディスプレイの文字をすべて消す

    // 構造体のtimeInfoに時刻情報を書き込む
    getLocalTime(&timeInfo);

    char date[12], now_time[7];
    sprintf(date, "%04d/%02d/%02d",
    timeInfo.tm_year + 1900, timeInfo.tm_mon + 1, timeInfo.tm_mday);
    sprintf(now_time, "%02d:%02d", timeInfo.tm_hour, timeInfo.tm_min);
    Serial.println(date);

    display.setFont(ArialMT_Plain_16); // フォントサイズを10pxに設定
    display.drawString(2, 2, String(date));
    display.setFont(ArialMT_Plain_24); // フォントサイズを24pxに設定
    display.drawString(35, 25, String(now_time));
```

```
    display.display(); // 設定した文字列をディスプレイに表示させる  
}
```



図 5.26: 時刻を表示したディスプレイ

第 6 章

応用編

6.1 Web サーバからの操作

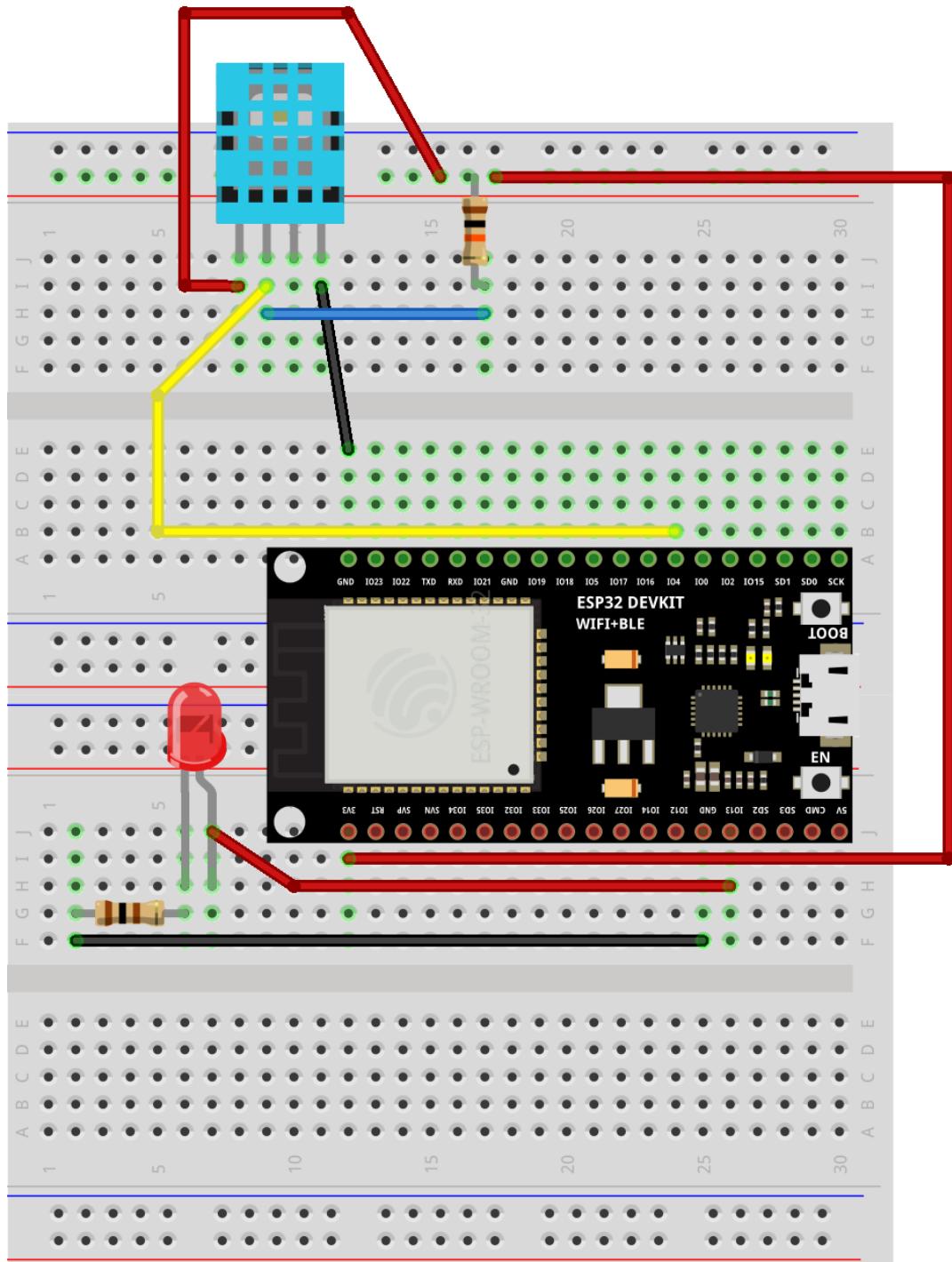
ESP32 上で Web サーバを起動させて、LED の操作とセンサの値読み取りを行います。

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 2
 - 10k × 1
 - 100 × 1
 - LED × 1
 - DHT11 × 1
 - ジャンプワイヤ

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



fritzing

図 6.1: 回路図

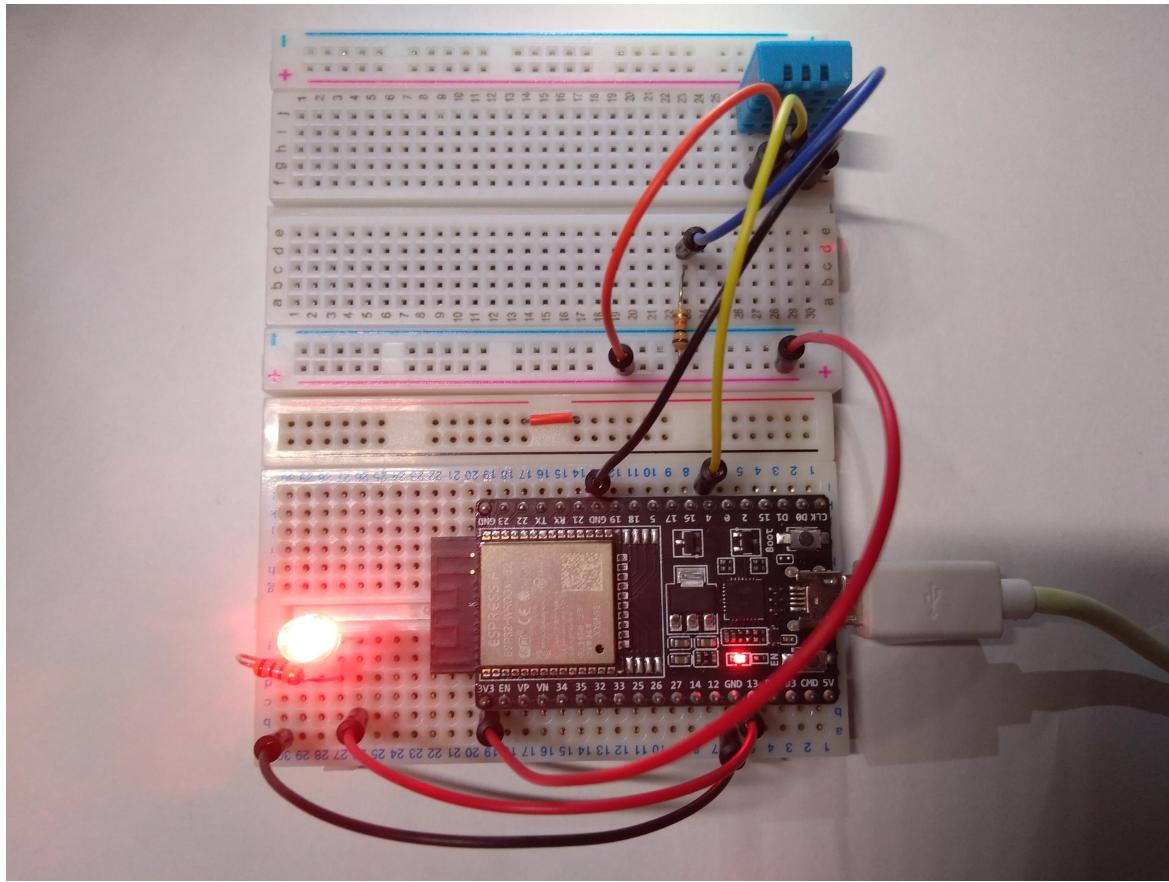


図 6.2: 回路配置図

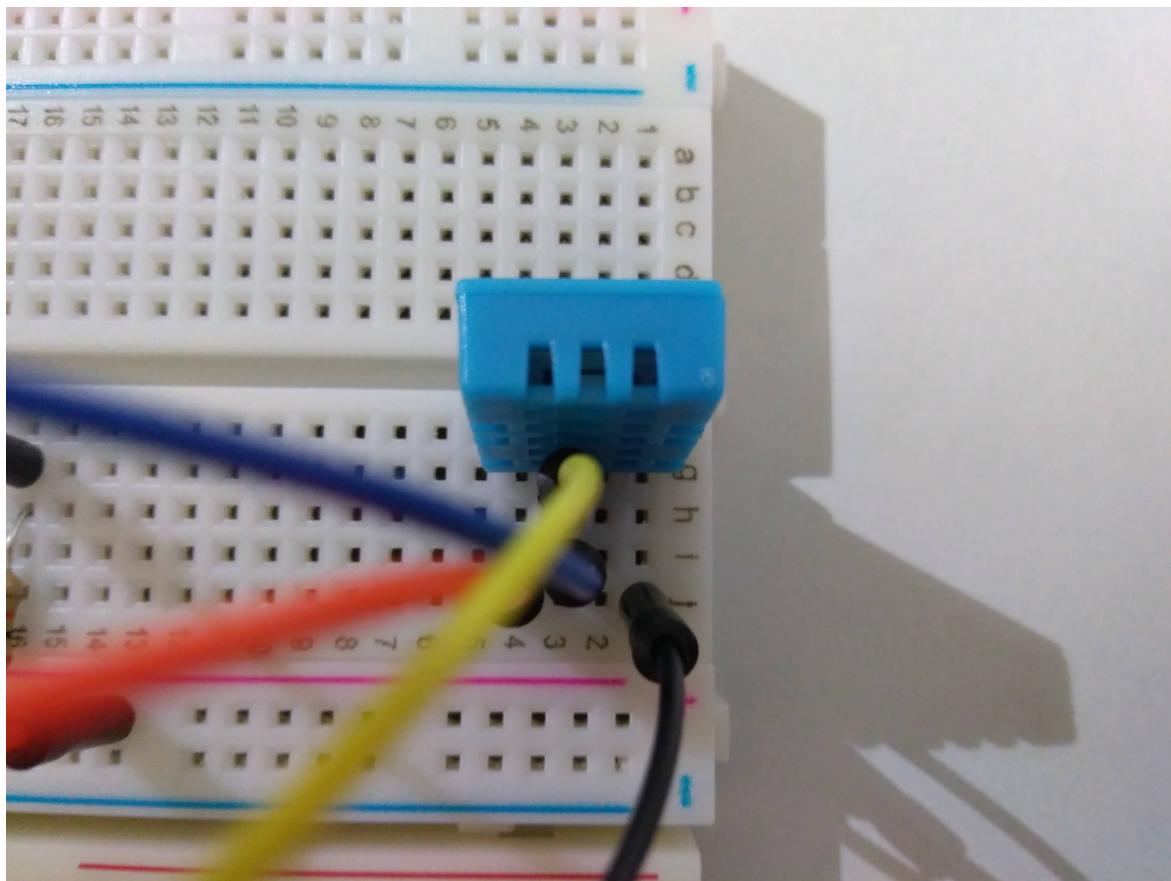


図 6.3: DHT11 アップ図

リスト 6.1: d

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#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

// WiFi接続用変数
```

```
const char *ssid = "ele2o3-b2309f-g";
const char *password = "fedd4123as3u";

String common_html = "\
<!DOCTYPE html>\n\
<html>\n\
<head>\n\
    <meta charset=\"utf-8\" />\n\
    <title>ESP32 DashBoard</title>\n\
    <link rel=\"stylesheet\" href=\"/stylesheet.css\" />\n\
</head>\n\
<body>\n\
    <header>\n\
        <div class=\"container\"\>\n\
            <div class=\"header-title\"\>\n\
                <div>\n\
                    <a id=\"top-btn\" class=\"header-logo\" href=\"/\">ESP32 DashBoard</a>\n\
                </div>\n\
            </div>\n\
            <div class=\"header-menu\"\>\n\
                <ul class=\"header-menu-right\"\>\n\
                    <li><a href=\"/blink\">Blink</a></li>\n\
                    <li><a href=\"/sensor\">Sensor</a></li>\n\
                </ul>\n\
            </div>\n\
        </div>\n\
    </header>\n\
\n";\n\nWebServer server(80);\n\nvoid handleNotFound() {\n    server.send(200, \"text/plain\", \"hello from esp32!\");\n}\n\nvoid handleRoot() {\n    String message = \"\\\n        <div class=\"top-wrapper\"\>\n            <h1>Here is the ESP32 DashBoard</h1>\n        </div>\n    \"\n    server.send(200, \"text/html\", message);\n}
```

```
</body>\n\
</html>\n\
\n";
server.send(200, "text/html", common_html + message);
}

void handleLedBlink() {
String message = "\n
<div class=\"top-wrapper\">\n\
<h1>LED Blink</h1>\n\
<button onclick=\"location.href='/blink/on'\">ON</button>\n\
<button onclick=\"location.href='/blink/off'\">OFF</button>\n\
</div>\n\
</body>\n\
</html>\n\
\n";
server.send(200, "text/html", common_html + message);
}
void handleDHT11() {
// 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);
String message = "\n
<div class=\"top-wrapper\">\n\
<h1>DHT11 Sensor</h1>\n\
<h2>Temperature: " + String(temperature) + " °C</h2>\n\
<h2>Humidity: " + String(humidity) + "%</h2>\n\
<h2>ApparentTemperature: " + String(apparent_temperature) + "%</h2>\n\
<button onclick=\"location.href='/sensor'\">UPDATE</button>\n\
</div>\n\
</body>\n\
</html>\n\
\n";
server.send(200, "text/html", common_html + message);
}
```

```
</div>\n\
</body>\n\
</html>\n\
\n";
server.send(200, "text/html", common_html + message);
}

void handleCSS() {
    String message = "\n
@charset \"UTF-8\";* {min-height: 0;min-width: 0;}\n\n
body { margin: 0; font-weight: 400; -webkit-font-smoothing: antialiased; }\n\n
font-size: 14px; color: #888; }\n\n
a { text-decoration: none; color: #323a45; }\n\n
-webkit-transition: all 0.3s; transition: all 0.3s; }\n\n
a a:hover { color: #5983ff; text-decoration: none; }\n\n
ul { margin: 0; padding: 0; list-style: none; }\n\n
li { margin: 0; padding: 0; list-style: none; }\n\n
.container { width: 1170px; padding-right: 15px; }\n\n
padding-left: 15px; margin-right: auto; margin-left: auto; }\n\n
h2,h3,h4,h5,h6 { margin-top: 10px; }\n\n
margin-bottom: 10px; font-weight: 400; }\n\n
/*----- ここからheader -----*/\n\n
header { height: 50px; position: fixed; top: 0; }\n\n
left: 0; right: 0; background-color: #2489c4; line-height: 50px; }\n\n
-webkit-box-shadow: -1px 1px 1px rgba(0, 0, 0, 0.1); }\n\n
box-shadow: -1px 1px 1px rgba(0, 0, 0, 0.1); }\n\n
z-index: 10; }\n\n
.header-title { float: left; }\n\n
.header-logo { color: white; font-weight: 700; }\n\n
font-size: 22px; font-family: \"Dosis\", sans-serif; cursor: pointer; }\n\n
.header-menu-right { float: right; }\n\n
.header-menu-right li { float: left; }\n\n
.header-menu-right a { color: #ffffff; font-weight: 700; }\n\n
padding: 10px; text-decoration: none; }\n\n
.header-menu-right a:hover { color: #104600; }\n\n
-webkit-transition: color 0.3s; transition: color 0.3s; }\n\n
.heading { padding-top: 60px; padding-bottom: 30px; color: #5f5d60; }\n\n
text-align: center; }\n\n
.heading h2 { font-weight: normal; }\n\n
/*----- ここまでheader -----*/
```

```
.top-wrapper { padding-top: 120px; padding-bottom: 100px; \n\
color: #5f5d60; text-align: center; }\n\
\n";\n\n    server.send(200, "text/css", message);\n}\n\nvoid setup() {\n    Serial.begin(115200);\n    // WiFi.mode(WIFI_STA);\n    WiFi.begin(ssid, password); // Wi-Fi接続開始\n    while (WiFi.status() != WL_CONNECTED) // Wi-Fiアクセスポイントへ接\n    続するまで待機\n    {\n        Serial.println("Waiting for Wi-Fi connection....");\n        delay(500);\n    }\n    Serial.println("Connected to Wi-Fi");\n\n    // ESP32のIP\n    Serial.print("ESP32 IP: ");\n    Serial.println(WiFi.localIP());\n    // ネットワークゲートウェイのIP\n    Serial.print("Network Gateway IP: ");\n    Serial.println(WiFi.gatewayIP());\n    // DNSServerのIP\n    Serial.print("Network DNS Server IP: ");\n    Serial.println(WiFi.dnsIP());\n\n    while (!MDNS.begin("ESP32"))\n    { // mDNSでESP32という名前を登録する\n        Serial.println("Waiting for mDNS to be configured....");\n        delay(100);\n    }\n    Serial.println("Complete mDNS configuration.");\n\n    pinMode(13, OUTPUT); // GPIO13を出力端子として用いる\n    dht11.begin(); // DHT11を始動させる
```

```
server.on("/", handleRoot);
server.on("/blink", handleLedBlink);
server.on("/blink/on", []() {
    digitalWrite(13, HIGH); // GPIO13に電流を流しLEDを光らせる
    handleLedBlink();
});
server.on("/blink/off", []() {
    digitalWrite(13, LOW); // GPIO13に電流を流しLEDを光らせる
    handleLedBlink();
});
server.on("/sensor", handleDHT11);
server.on("/stylesheet.css", handleCSS);

server.onNotFound(handleNotFound);
server.begin();
Serial.println("WebServer 起動");
}

void loop() {
    server.handleClient();
    delay(5);
}
```

<https://en.wikipedia.org/wiki/.local>

リスト 6.2: WebServer 起動時のシリアルモニタ

```
Waiting for Wi-Fi connection....
Connected to Wi-Fi
ESP32 IP: 192.168.2.159
Network Gateway IP: 192.168.2.1
Network DNS Server IP: 192.168.2.1
Complete mDNS configuration.
```

WebServer 起動
ESP32アクセスポイントのIPアドレスは192.168.4.1

<http://esp32.local>

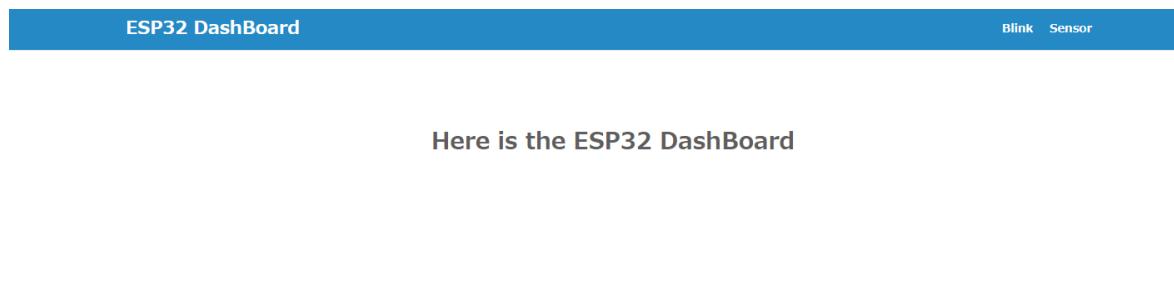


図 6.4: Web サーバのトップページ

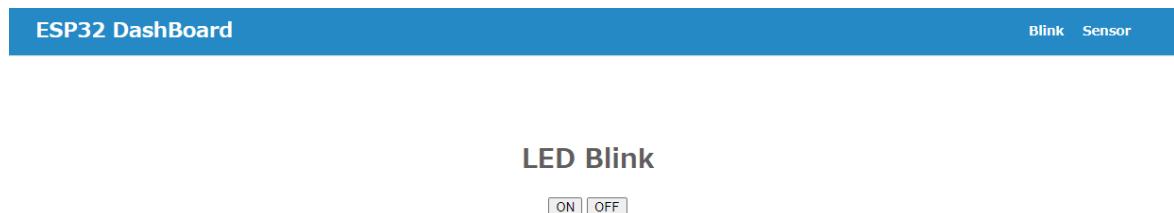


図 6.5: LED 操作画面

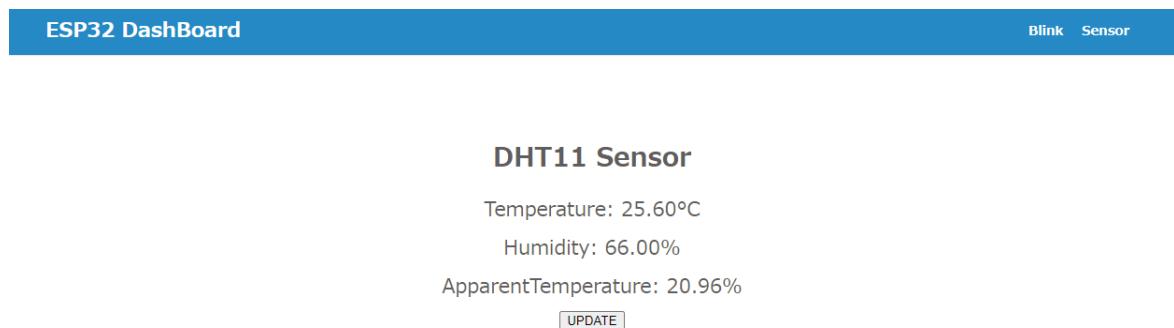


図 6.6: センサ読み取り画面

6.2 ESP32 でアクセスポイントを設定

サーバクライアント Interface 2018.9 より Wi-Fi ネットワークはアクセ・ポイント(AP)を中心としたネットワークアクセスポイントは多くの場合、インターネットなどの他のネットワークに接続しており、その場合はルータとも呼ばれるアクセス・ポイントに接続する端末をステーション(STA)という。ESP32 を AP モードするには > WiFi.softAP(ssid, password); STA モードでアクセスポイントに接続するには > WiFi.begin(ssid, password);

WiFi のアクセスポイントがなくても ESP32 が 2 庁あれば、片方をアクセスポイントにして通信できる

- 必要材料
 - ESP32 × 1
 - ブレッドボード × 2

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

リスト 6.3: アクセスポイントプログラム

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiAP.h>
#include <WebServer.h>
#include <ESPmDNS.h>

// ESP32アクセスポイントのSSIDとパスワード（名称自由）
const char *ssid = "ssid";
const char *password = "password";

// ポート80番につなぐWebServerを設定
WebServer server(80);

// エラー時に呼ばれる関数
void handleNotFound()
{
    server.send(400, "text/plain", "not found");
}

void setup()
{
    Serial.begin(115200);
    WiFi.mode(WIFI_AP);           // ESP32をアクセスポイントに設定
    WiFi.softAP(ssid, password); // アクセスポイントモード設定
    delay(100);

    Serial.print("ESP32アクセスポイントのIPアドレスは");
    Serial.print(WiFi.softAPIP());
    Serial.println("です。");

    while (!MDNS.begin("esp32"))
    { // mDNSでESP32という名前を登録する
        Serial.println("Waiting for mDNS to be configured....");
    }
}
```

```
    delay(100);
}
Serial.println("Complete mDNS configuration.");

// WebServerの設定（"/"にアクセスした際の挙動を設定）
server.on("/", []()
{
    server.send(200, "text/plain", "ok");
});
// WebServerの設定（エラー時の挙動を設定）
server.onNotFound(handleNotFound);

server.begin(); // サーバを起動する
Serial.println("WebServerを起動します");
}

void loop()
{
    // サーバへのアクセスを受け付ける
    server.handleClient();
    delay(5);
}
```

リスト 6.4: アクセスポイント起動時のシリアルモニタ

```
ESP32アクセスポイントのIPアドレスは192.168.4.1です。
Complete mDNS configuration.
WebServerを起動します
dhcps: send_nak>>udp_sendto result 0
```



図 6.7: アクセスポイント表示画面



図 6.8: アクセスポイント接続時画面



図 6.9: ESP32 のアクセスポイント経由で接続



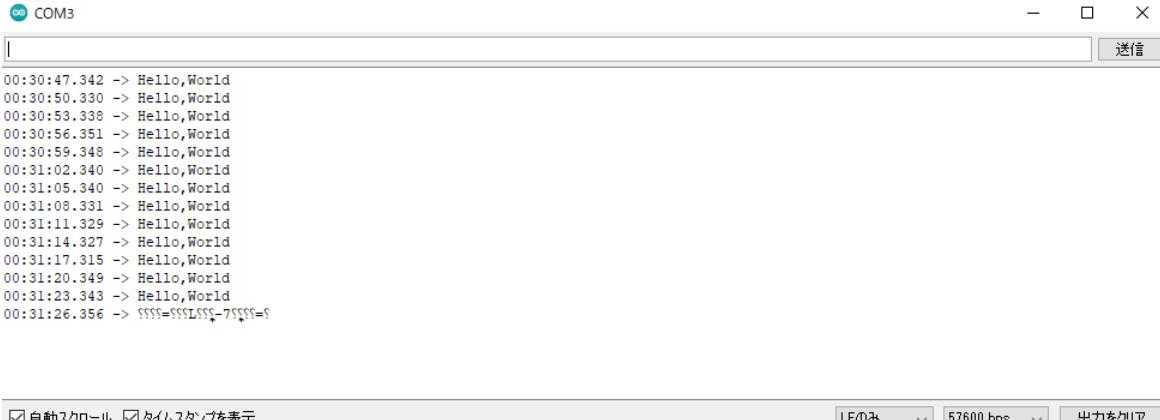
図 6.10: ESP32 のアクセスポイント経由でのエラー画面

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

ここでは、筆者が ESP32 で開発をしていた際に遭遇したエラーとその解決法を紹介します。



```
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§§§-§§§§§=§
```

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: 文字化けしたシリアルモニタ



図 A.2: Upload Speed を変更する

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

A.3 プログラムが反映されない



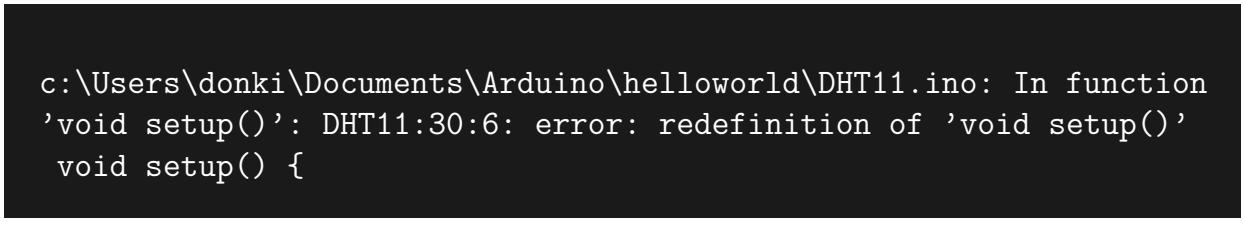
The screenshot shows the Arduino IDE interface. The title bar reads "helloworld | Arduino 1.8.15 (Windows Store 1.8.49.0)". The menu bar includes "ファイル" (File), "編集" (Edit), "スケッチ" (Sketch), "ツール" (Tools), and "ヘルプ" (Help). Below the menu is a toolbar with icons for back, forward, refresh, and file operations.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

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

図 A.3: プログラムが保存されていない

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 がない

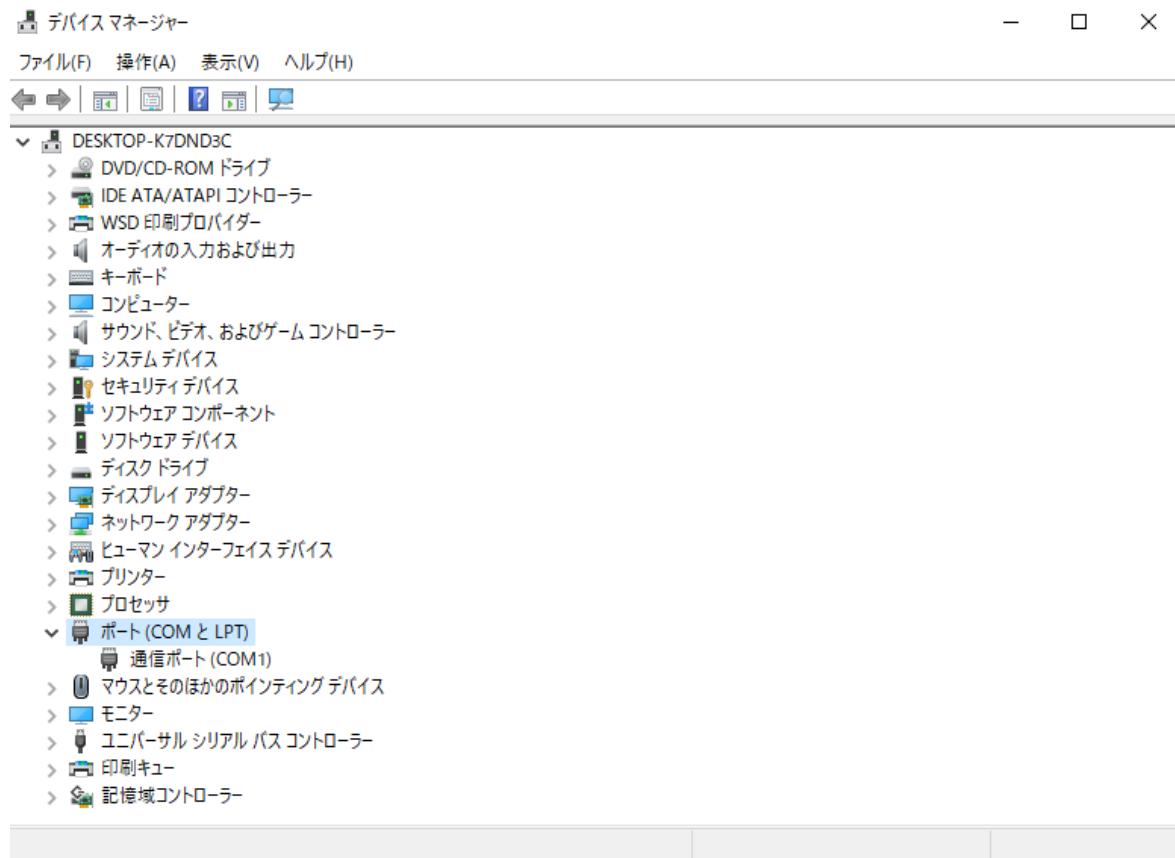


図 A.4: デバイスマネージャー

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: ドライバのインストールページ

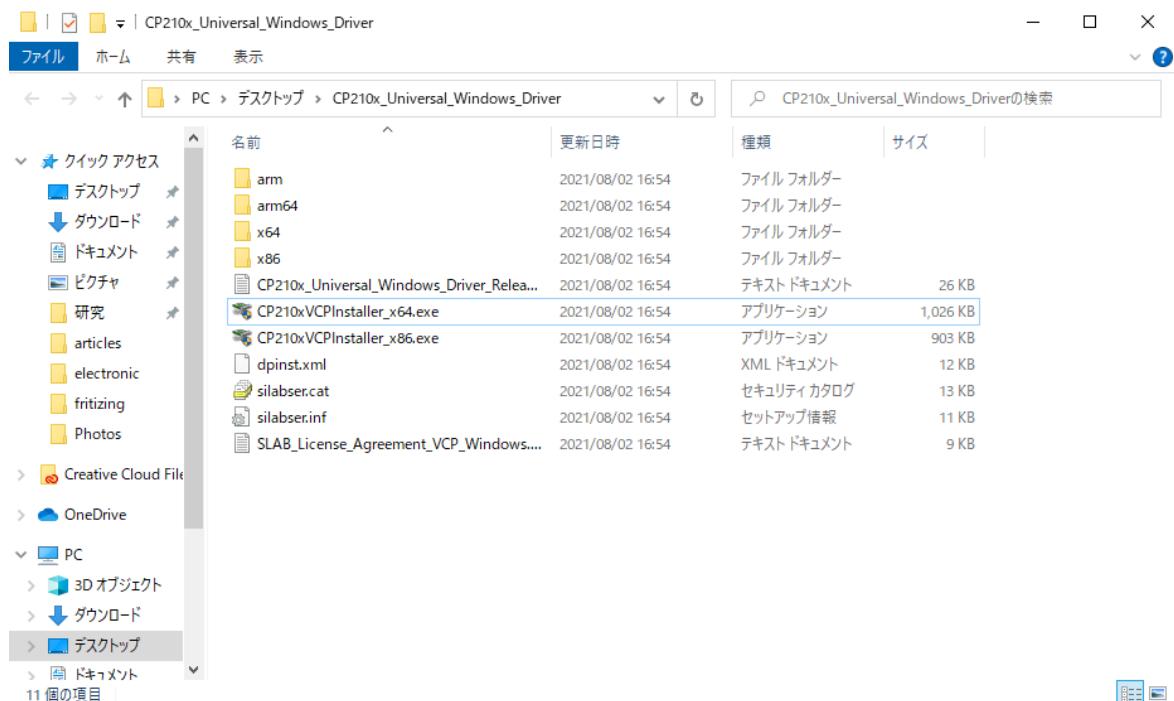


図 A.6: インストールしたドライバファイル

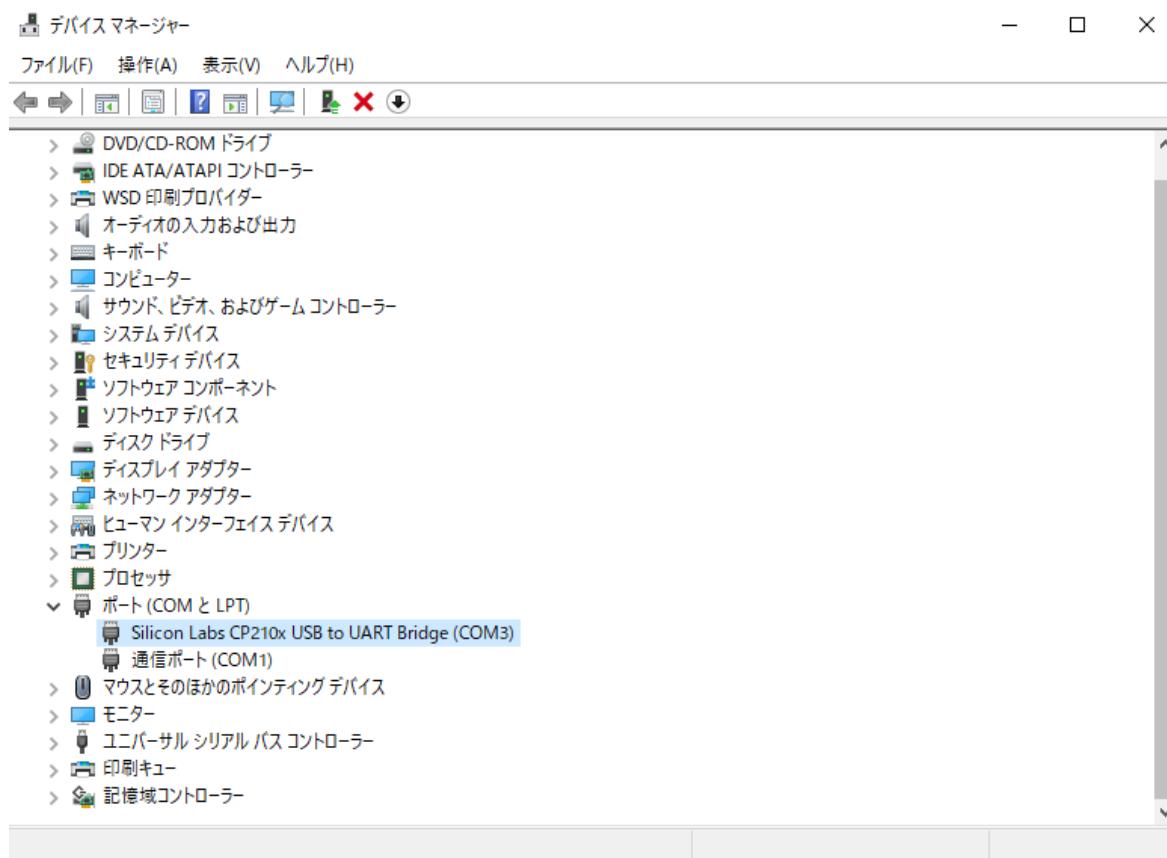


図 A.7: デバイスマネージャーの反映が成功

A.6 COM ポートが表示されない

デバイスマネージャー 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

A.10 Failed to execute script esptool

シリアルポート「Failed to execute script esptool」が選択されていますが、そのポートは存在しないか、ボードが接続されていません。

シリアルモニタを閉じる

参考文献

山本 陽平. Web を支える技術 HTTP ,URI ,HTML ,そして REST
WEB+DB PRESS plus 株式会社技術評論社.

著者紹介

THEToilet / @THEToilet

この本を執筆していたら夏がおわりました。

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 8 月 12 日 初版第 1 刷 発行

著 者 THEToilet、raimu