農業分野における「まち・ひと・しごと創生」の実現を
支援する農業 IT 人材育成テキスト
（農業 IT 編）

平成３０年２月

学校法人三橋学園
船橋情報ビジネス専門学校

まえがき

初めての技術は、王道を進む

初めてのマイコンを利用し始めるときの【王道】と考えている道筋があります。第一分冊もそのとおりに進みました。第二分冊も同じです。これまでに多くのシステムに臨みましたが、初めてのマイコンとの共同作業は、その【王道】を進むのが良いと思います。その【王道】とは、2冊のテキストの前半部分です。私のテキストは、どれも必ずLEDの点滅から初めています。2冊のテキストの目次を見比べながら【王道】を考えつつ、道を逸れないように慎重に進むことにしましょう。しばらくの間、パートナーとペアを組んで道を進んでいると、相手の得意技や性格が見えてきます。そのとき【王道】を進んできた意味が理解されるでしょう。

テキスト後半では、パートナーの得意技とWEBサービスというIoT Carrierを使い、あらゆる情報をどこにでも、どこからでも届けることのできる【王様】に近づく夢を描いてテキストを作りました。

wiseman　原田賢一

【なんでもつながる！！IoT 基礎講座】

WiFi マイコン編

目次

【なんでもつながる！！IoT 基礎講座】

WiFi マイコン編

実習の範囲と WiFi マイコン

図　1

この写真は、千葉県柏市の北部を流れる利根川の河川敷にある農地です。奥に見えているのは、つくばエクスプレスと常磐高速の橋です。右手に進むと利根川を渡って筑波方面、左手は都心方面に向かいます。黄金色になった実りの秋に、本格的に収穫が始まる圃場の様子が写っています。

さて、このテキストの前編では無線マイコンを利用した実験を、いろいろと行ってきました。写真のような広大なフィールドでも、多くのセンサー等を利用することで、無線を通じたネットワークを形成して、環境情報を収集するモデルを考えていました。使用していたのは無線マイ

コンだったのですが、それを【WiFi機能を持つマイコン】で置き替えて、さらに実験を進めようと考えています。

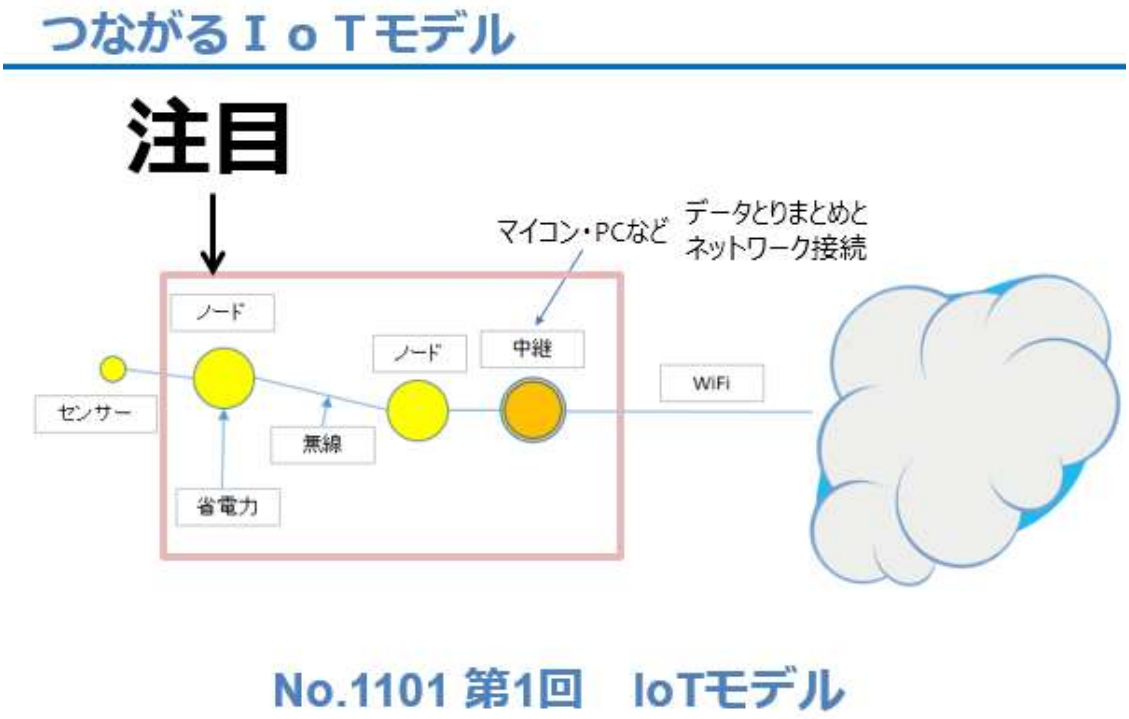


図　2

上の図を覚えていますか。テキストの第一分冊で、はじめに出てきた「つながるIoTモデル」の図です。左側にあるセンサーは、広大なフィールドに配置されて、環境情報を検出します。無線機能を持ったノードが、センサーの計測値を取り込み、無線で対抗するノードに送信します。対向ノード（受信ノード）は、中継機能と接続していて、受信した計測値などを取りまとめて、WiFi・アクセスポイントを経由してWEBに情報を通知しています。これから始める一連の実験では、図の枠で囲んだ部分に、文字通り【注目】してみます。

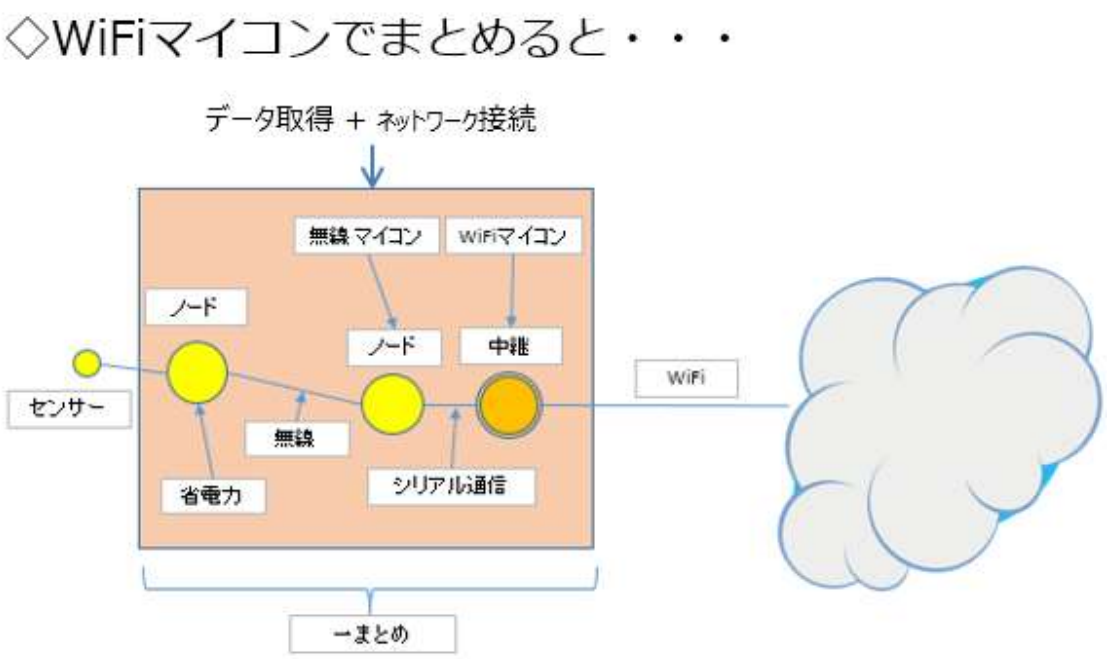


図　3

この部分を【WiFi機能を持つマイコン】で一まとめにすると、すっきりとした図になります（下図）。

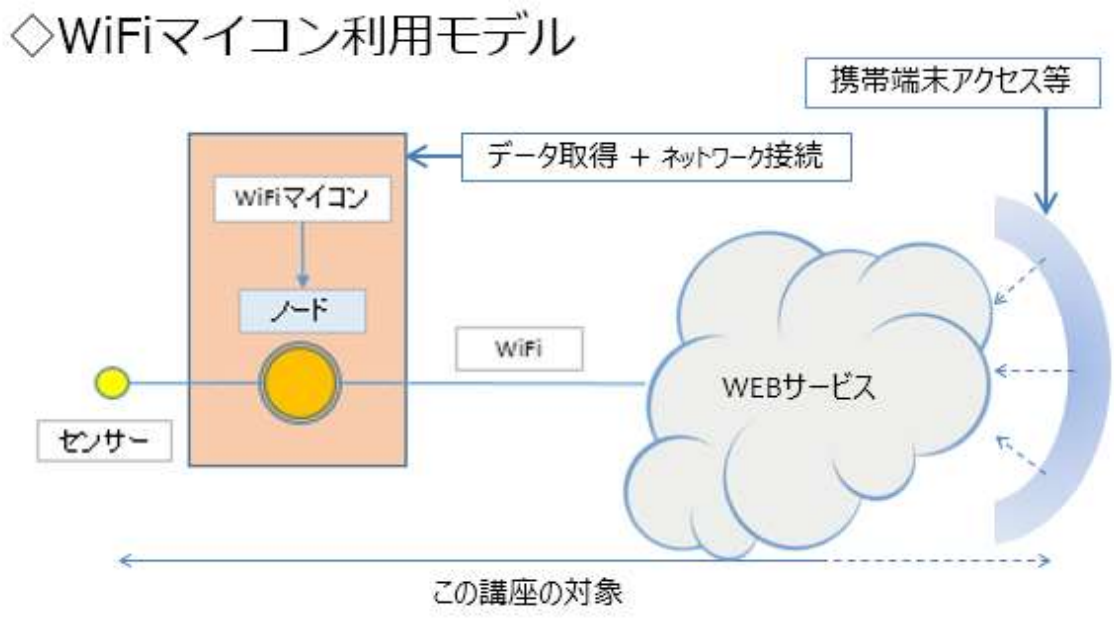


図　4

ノードとしてWiFiマイコンを利用することで、センサーの情報を取得・変換計算・WEBサービスへのUp Load等が【1Stop】で出来るようになります。これで、WEBの利用がとても身近なものになります。これから始める14回の講座では、WiFiマイコンを利用して、センサーで計測したデータを、WEBサービスを経由して携帯端末でアクセスしてみるところまでを対象範囲として、様々な実験を行います。

図 5

上の図のは、WiFi機能を持つESP-WROOM-02というマイコンで、中央のWiFiマイコンモジュール本体は、20×18mmと郵便切手のような大きさです。モジュールの両端に見える信号を取り出す端子のピッチが、ブレッドボードには合わないので、標準と云える2.54mmのピッチに変換する基板に取り付けたものが右側のモジュールです。実験にはUSB-シリアルI/Fを搭載しているもの（左側）が便利です。

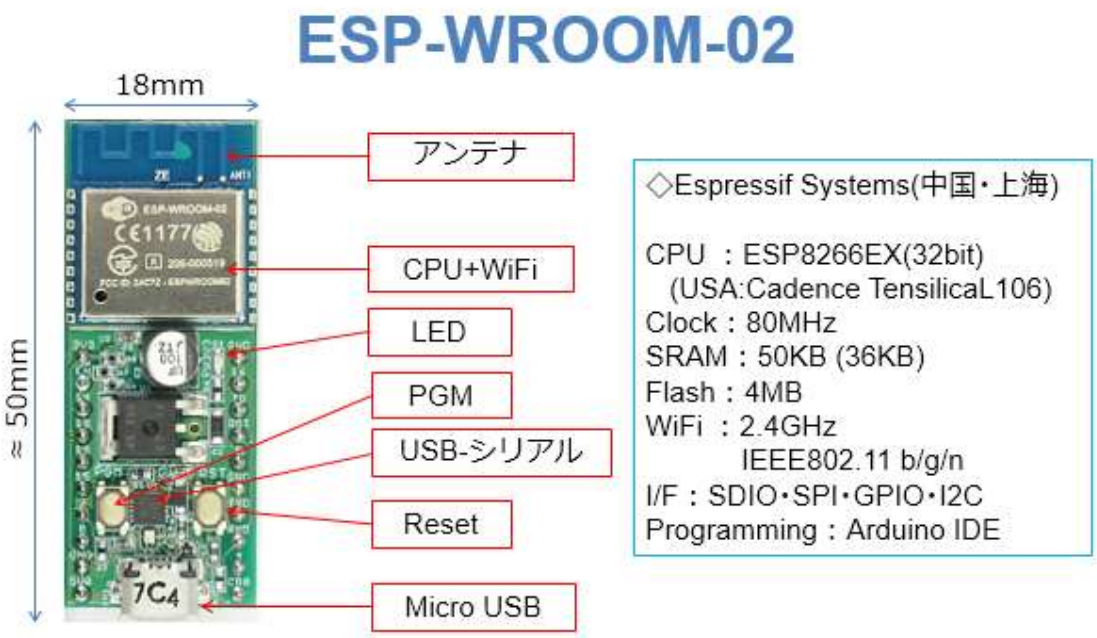


図 6

この基板は、約 50×18mm の大きさです。上図では、上部にアンテナのパターンが見えています。CPU と WiFi 機能は金属のケースに収められて、WiFi と型番 ESP-WROOM-02 の刻印が見えます。電源 LED は写真では小さくて見えませんが、赤く点灯します。中央にある黒い四角形の IC は電源の IC で、5V から 3.3V を作り出しています。さらにその下にある、小さな黒い IC は USB-シリアル I/F の IC です。その IC の両側にある楕円形の白いボタンは、Reset と Programming の SW で、プログラムをマイコンのフラッシュメモリに書き込む際に操作をします。基板の一番下に見えるのは microUSB コネクタです。　PC との接続や外部から 5V 電源を供給する際に使用します。

金属ケース内の CPU モジュールは、中国・上海の Espressif Systems 社が開発した ESP8266EX(32bit)です。ユーザが使用できる SRAM は 36KB、Flash Memory は 4MB あります。WiFi は 2.4GHz 帯

IEEE802.11 b/g/n で、多くのアクセスポイントに対応しています。I/F は、SDIO・SPI・GPIO・I2C と広く利用されているマイコンと変わりません。特徴は Programming の環境で、世界中で大変多く利用されている Arduino と云うマイコンボードの IDE（Integrated Development Environment：統合開発環境）を使ってソフトウエア開発とマイコンへのプログラム書込みができるという点です。そのために microUSB コネクタがあると考えて良いでしょう。このコネクタは、外部とシリアル通信を行う用途にも使えます。

次の図は、データシートの一部を抜粋したものです。操作電圧が 3.0～3.6V となっていて、電池駆動も可能です。操作電流は 80mA です。

ハードウエア要点

Categories	Items	Values
Hardware Paramaters	Peripheral Bus	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	5x5mm
	External Interface	N/A

図　7

ソフトウエア要点

Categories	Items	Values
Software Parameters	WiFi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP

図 8

ソフトウエアの要点を上図に示します。Firmware Upgrade の OTA とは、Over-The-Air のことで、WiFi マイコンですから無線を通じて Firmware Upgrade を行うことも可能です。クラウド環境での開発もできるようですが、この講座では利用しません。様々なユーザーアプリケーションの開発（custum firmware development）に利用できる SDK やライブラリが、数多く公開されています。対応するネットワークの種類を見ても、WEB 連携に使うためのマイコンと言えます。

では、これから 14 回の講座をトレースしていただきながら、この WiFi マイコンの基本的な使いかたと、WEB サービス連携システムの開発の方法を実習してください。

第1回　LED 点滅

新しいマイコンを使い始めるときの最初のテーマは、LED 点灯です。まずはこのテーマで新しいマイコンの DO（Digital Output：デジタル出力）を使ってみるのが良いと思います。

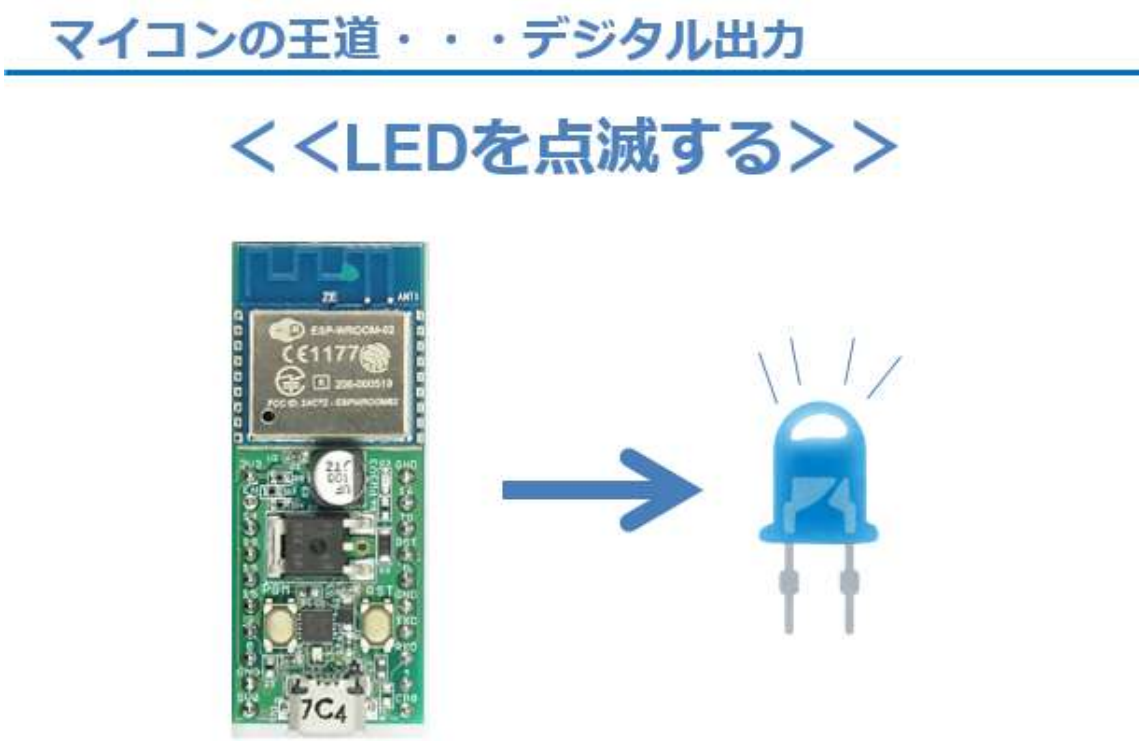


図　9

このマイコンは GPIO という汎用の入出力ポートをいくつか持っています。それを使って LED を点滅させてみましょう。

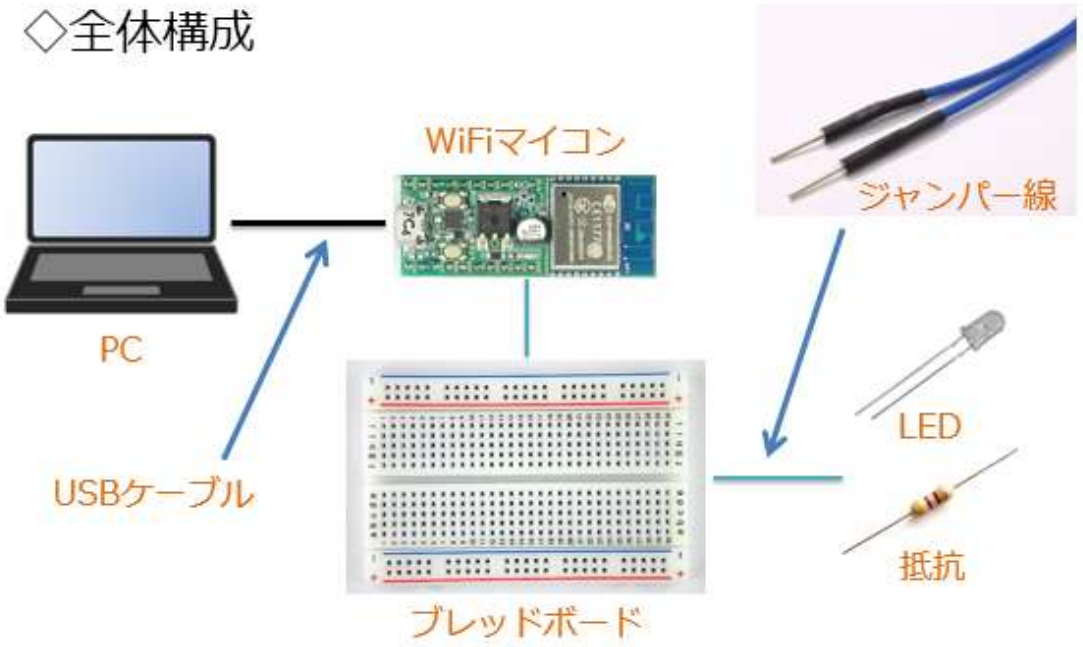


図　10

システムの全体構成を上図に示します。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個

各パーツは 1 個でも入手できます。通販でも購入が可能です。WiFi マイコンモジュールは ESP8266 という型番で容易に見つかるでしょう。ブレッドボードはジャンパー線と一緒に使うことで、半田付けせずに配線ができるので大変便利です。このような実験に適しています。

◇マイコンと周辺デバイスの接続に使います。

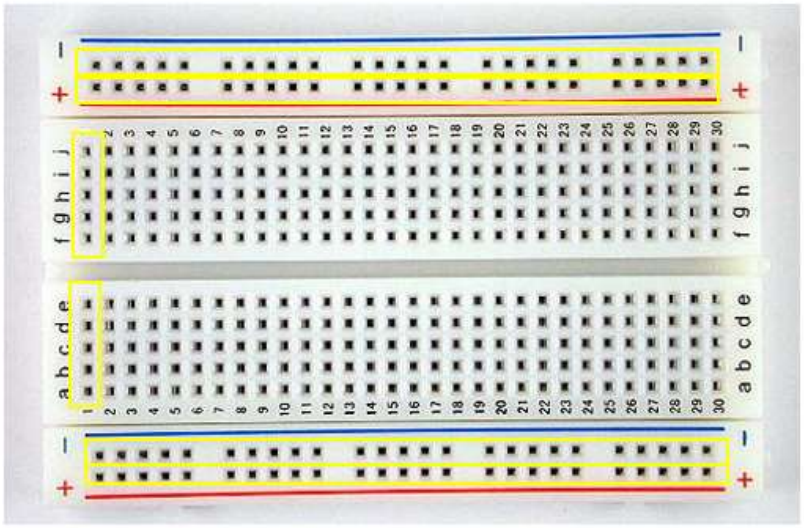


図　11

◇マイナス(－)の線、プラスの線(＋)、
A～E、F～Jの線はブレッドボード内部で
繋がっています。

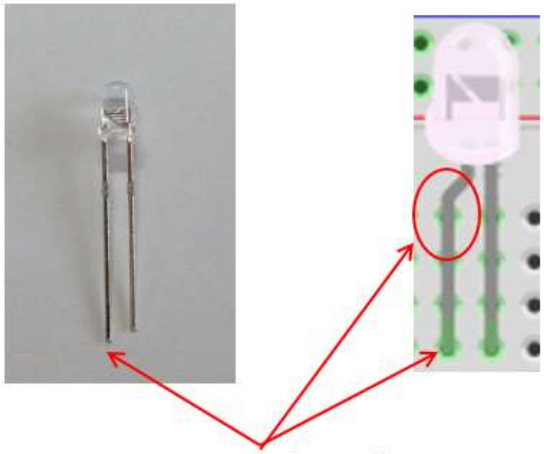
図　12

上の写真のように、ブレッドボードの内部で穴が接続されています。中央部分の溝の上下は繋がっていません。この溝を跨ぐように IC などを配置します。青色の線がある部分の穴には、GND（－）側を、赤色の線がある部分の穴には、電源（＋）側を接続して利用します。ブレッドボードの上下の赤・青の電源（＋）・GND（－）を両方使う場合は、上下の同じ色の線の穴を、ジャンパー線でつないで使用します。

図　13

配線用ジャンパー線（上図）は、両端にピンが取り付けられています。このピンをブレッドボードの穴に差し込み配線します。

LED は、次の図のように足の長さで極性を示しています。長い方の脚から電流が流れ込んで、短い方の脚から流れ出てきます。反対に接続すると電流が流れずに光りません。実体配線図では足の長さが分りにくいので、長い方の脚を曲げて表現しています。



◇図では、長いほうのピンがわかりにくいので
曲げて表現しています。気を付けて配線してください。

図　14

抵抗のカラーコード

図 15

抵抗器は、カラーコードで抵抗値を示しています。今回使用する抵抗器は 470Ω のもので、LED に電流が流れすぎないようにするために使います。抵抗の両端で足を直角に曲げて使用します。

【重要】

LED に大きな電流が流れると、明るく光りますが、大きすぎる電流が流れると、寿命が短くなってしまいます。LED の最大定格電流としては 20~30mA の物が多く使われています。この最大定格電流の半分程度で、視認できる明るさが確保できますので、オームの法則に従って、E=5V,R=470Ω で計算すると、A=10.6mA となり、最大定格以下となります。この抵抗は電流制限抵抗と言って、水道でいうとバルブに相当します。バルブの絞り具合が抵抗値ということです。

◇抵抗は写真のように足を曲げて使います。
◇抵抗の値を書いたものを付けておくと、間違えにくくなります。

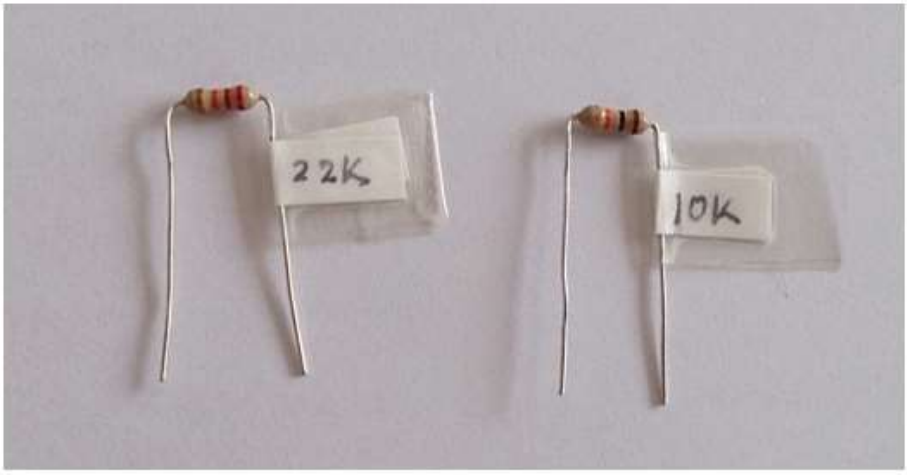


図 16

上の写真ように紙片に抵抗の値を書いたものをテープなどで貼っておくと、値の違う抵抗を間違いなく利用できると思います。回路を作成する前に、このような準備を念入りに行うことは、後の作業の効率化や誤り排除などの面で、とても効果があります。

実習キット【Appendix 参照】にはすべてのパーツがそろっていますが、個別に求めたパーツを利用する場合と同様に、極性や抵抗値などを十分に確認してから使用して下さい。

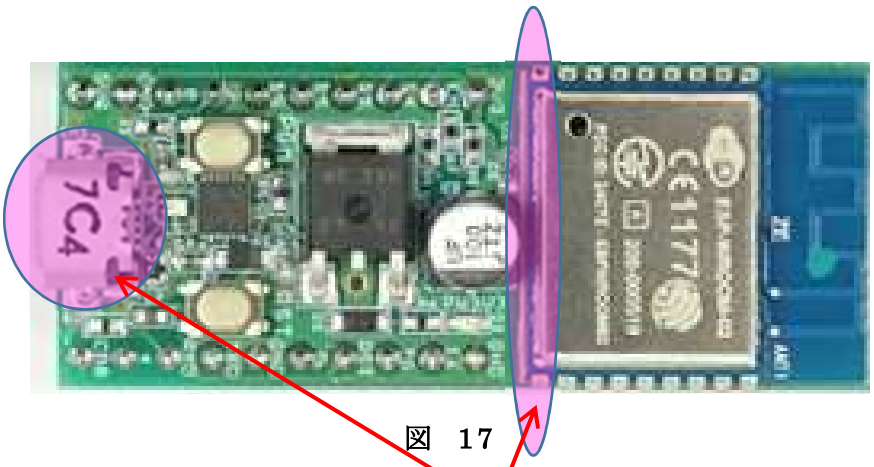
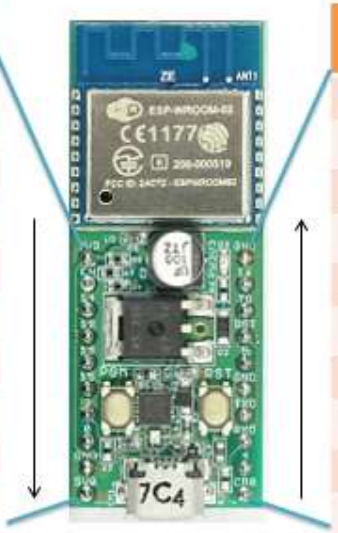


図　17

WiFi マイコンモジュールは写真の矢印で示す、マイコンモジュールと基板の境界部分と microUSB コネクタの部分が弱いので、ブレッドボードへの取付け・取り外し（特に取り外し）と microUSB コネクタの挿抜時（特に抜くとき）には強い力が掛からない様に、注意して作業を行ってください。ピンセットなどの工具を準備しておくと良いでしょう。

番号	名称	機能
1	3V3	3.3V(In/Out)
2	EN	イネーブル端子
3	14	GPIO14
4	12	GPIO12
5	13	GPIO13
6	15	GPIO15
7	2	GPIO2
8	0	GPIO0
9	GND	GND
10	5V0	5V In



番号	名称	機能
20	GND	GND
19	16	GPIO16
18	TO	ADC（max1V）
17	RST	リセット
16	5	GPIO5
15	GND	GND
14	TXD	TXD
13	RXD	RXD
12	4	GPIO4
11	CB0	通信モニタ用

図　18

上の図は、WiFi マイコンモジュールのピン配置を示しています。

WiFiマイコンモジュールのアンテナパターンを上に向けて、左上のピンから1,2,3,4・・・と番号が付いています。

----------<< 各ピンに配置されている信号 >>----------

1番：3.3Vの電源で駆動するときは、このピンに直接3.3Vを接続します。USB経由または、外部5Vで駆動する場合は、このピンから3.3Vの電源を取り出すことができます。

2番：イネーブル端子で、このモジュール全体の有効・無効を設定します。通常は解放で有効です。無効（Disable）とするときは、GNDに接続します。

3～8番：汎用入出力端子です。デジタル入出力を取り扱います。

12,16,19番：同上。

9,15,20番：GND

10番：外部5Vの入力。【注意】外部への5V取り出しや3.3V入力との併用はできません。

11番：USB-シリアルI/F ICの設定により、通信状態の表示などに使用します。通常は使用しません。

13番：RXDは、シリアル通信の受信信号です。

14番：TXDは、シリアル通信の送信信号です。

17番：RSTはこのマイコンモジュールのReset信号です。

18番：TO（TOUT）は、アナログ電圧を計測出来るADC（A/D Converter）が接続されているポートです。最大1Vまで測定できます。

いよいよ無線マイコンモジュールの実験回路を配線します。配線と言っても線の数や使用するデバイスの数が少ないので、じっくりと確認しながら行ってもすぐに終わります。各パーツには、くれぐれも余計な力がかからないようにして、回路の配線を行ってください。手順としては、半田付けする基盤の場合は、高さの低いものから順に基板に取り付けてゆくのが王道ですが、ブレッドボードとジャンパー線による配線なので、作業しやすいと思った順に行えばよいでしょう。回を重ねれば、自ずと最適な作業手順が身についてくると思います。

まず、ブレッドボードを小さい数字が左、アルファベットの A が下になるように置いて、次の図の様に配線をしてください。WiFi マイコンモジュールは、microUSB コネクタが左側に向いています。

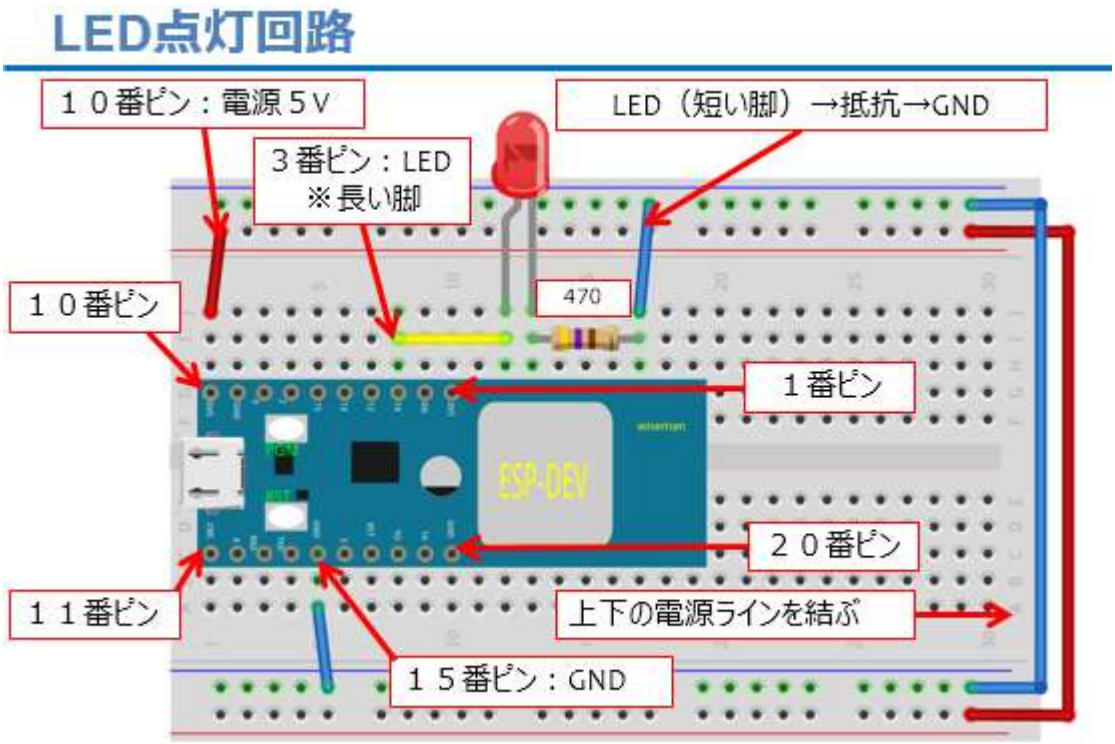


図 19

使用しているジャンパー線の色は、特に指定ではありませんが、電源（＋）側は赤系統、GND（－）側には青や黒を使うことが多いので、覚えておくと別の回路を見たときに役立ちます。WiFi マイコンモジュールの 10 番ピンから 5V を取り出し、15 番ピンは GND に接続しています。ブレッドボードの上下の電源ラインは、一番右側で上下をジャンパー線で接続しています。3 番ピンから LED の長い方の脚に配線して、LED の短い方の脚は 470Ω の抵抗を介して GND に接続します。このとき、抵抗の脚をそのまま GND に配線してもかまいません。そのようにして配線に使用するジャンパー線が少なくなると、基板上がすっきりして、全体を良く見る事ができるようになります。

GPIO14 を High（=1）にすると、3 番ピンから電流が LED の長い方の脚に流れ込み、LED を点灯させて、短い方の脚から 470Ω の抵抗を経て GND に流れ出るという回路です。

実際に配線した様子が次の写真です。

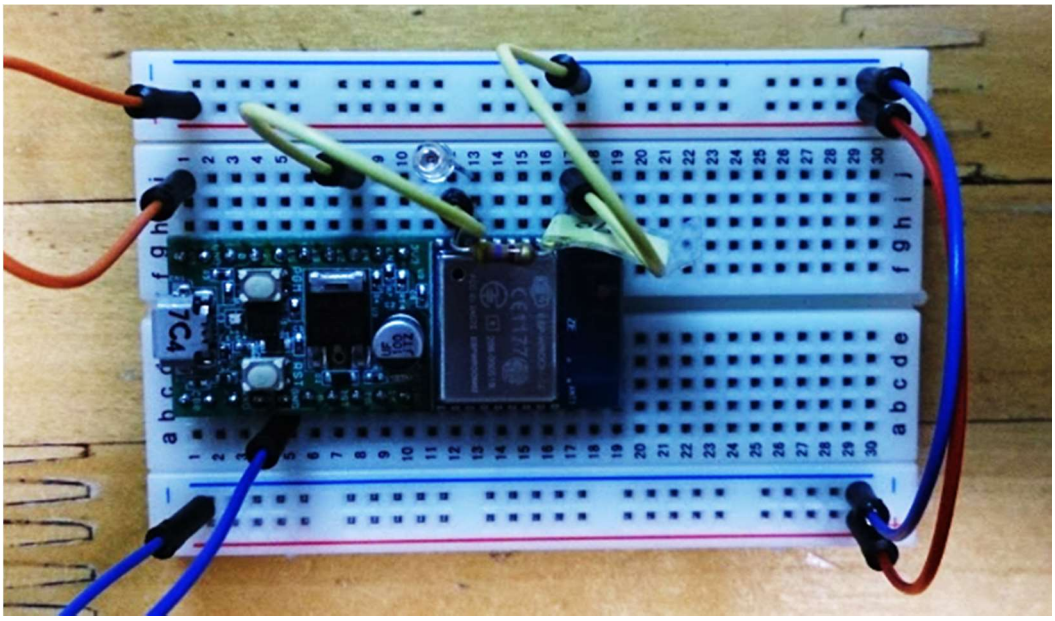


図 20

Arduino IDE

図 21

ソフトウエア開発には Arduino IDE を使用します。WEB で https://www.arduino.cc/を指定すると上のページが見つかります。ページ上部の SOFTWARE をクリックすると次のページに移動します。

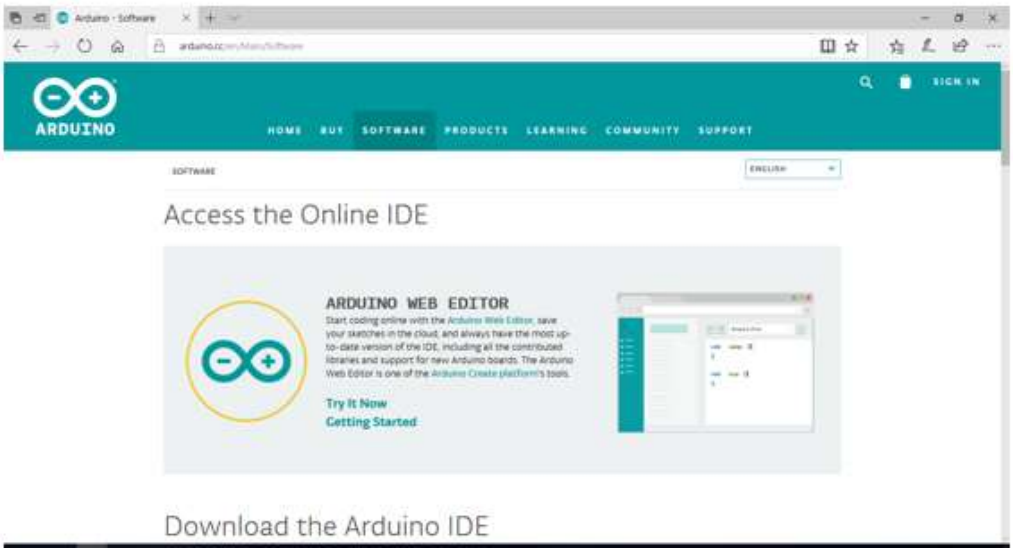


図 22

図 23

さらに、すこし下に移動すると上図のページになります。ここで、Windows Installer をクリックすると、下図のページに移動します。

図 24

ここで、寄付される方は CONTRIBUTE & DOWNLOAD を、そのままダウンロードを続ける方は JUST DOWNLOAD を選択します。

適当なフォルダにダウンロードファイルを置き、そのファイルをダブルクリックして Install します。途中で USB-シリアルドライバなどの Install 確認メッセージが表示されることがありますが、全て Install してください。また Java が通信するという確認のメッセージが表示されることがありますが、全ての通信を OK（またはチェック）してください。Install が終わるとデスクトップに下図左のような眼鏡マークのショートカットができています。それをダブルクリックすることで、下図右のような Arduino IDE が起動されます。プログラムソースコードは、ウインドウ中央の白い部分に記述します。

図 25

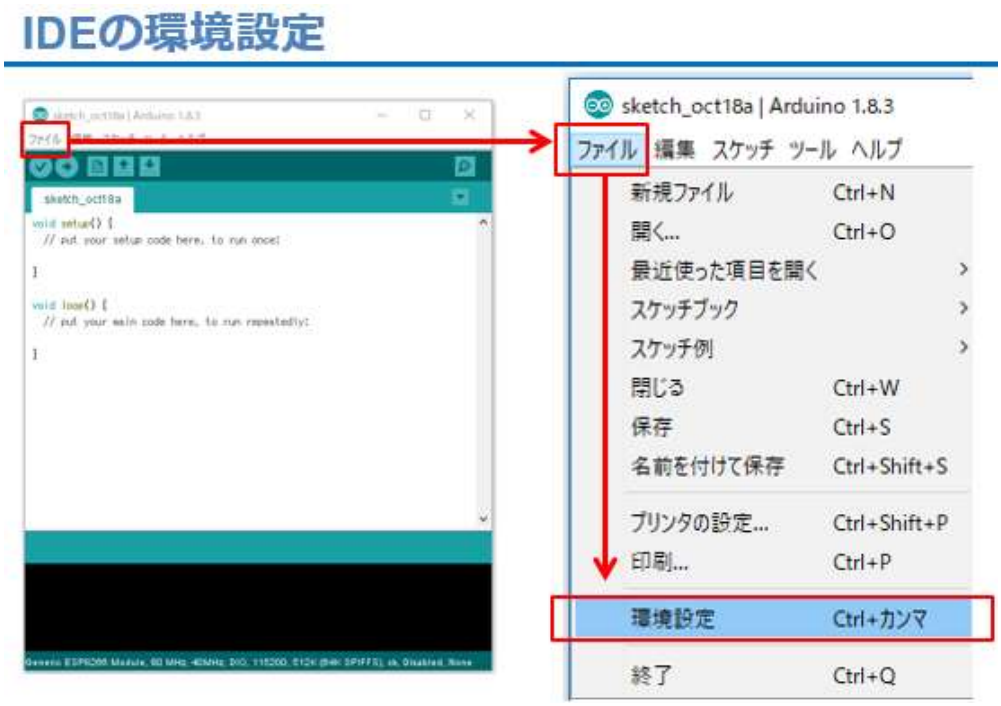


図　26

これから使用する WiFi マイコンモジュール専用のライブラリなどを準備します。上図で、【ファイル→環境設定】 と辿ると、次のウインドウが開きます。

図　27

ウインドウ下部にある【追加のボードマネージャの URL】に次の URL を入力して、OK ボタンを押してください。

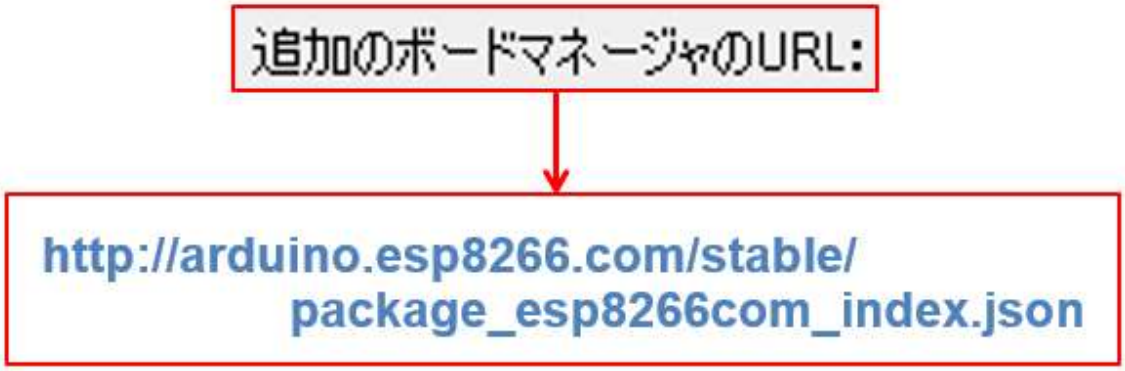


図　28

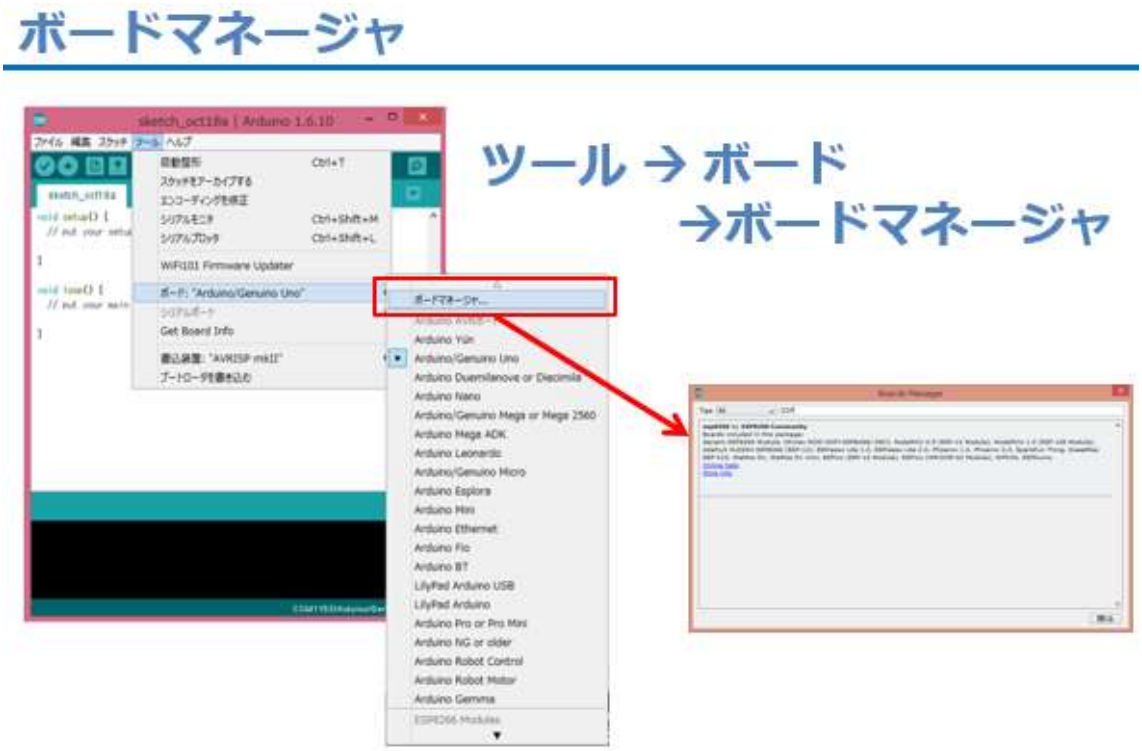


図　29

【ボード→ボードマネージャ】と辿り、ボードマネージャのウインドウを開きます。（次の図）

図　30

右上にある検索フィールドに ESP と入力し、表示される esp8266 by ESP8266 Community を選択し、Install します。（下図）

図　31

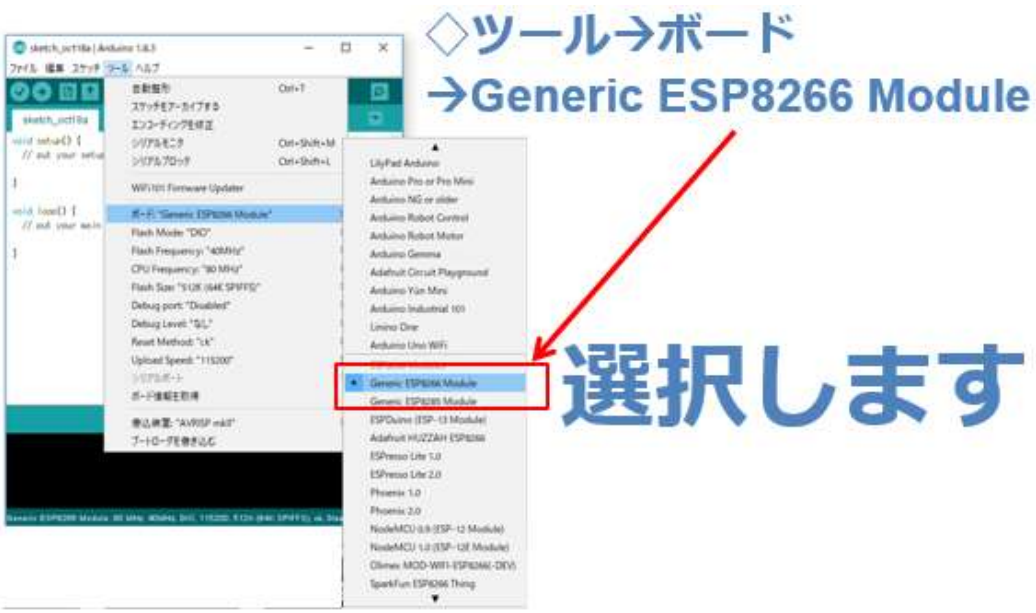


図　32

元のウインドウに戻り、【ツール→ボード】と辿り、一覧の中から、Generic ESP8266 Module を選択します。

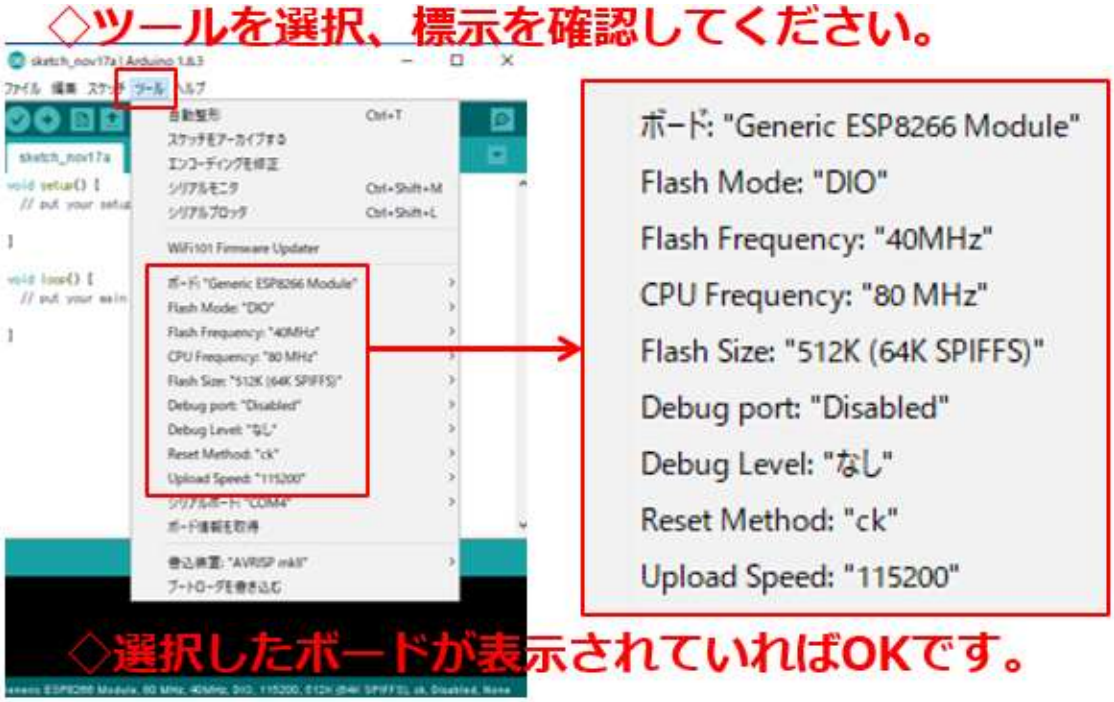


図　33

ツールで選択したボードが表示されていれば OK です。（上図）

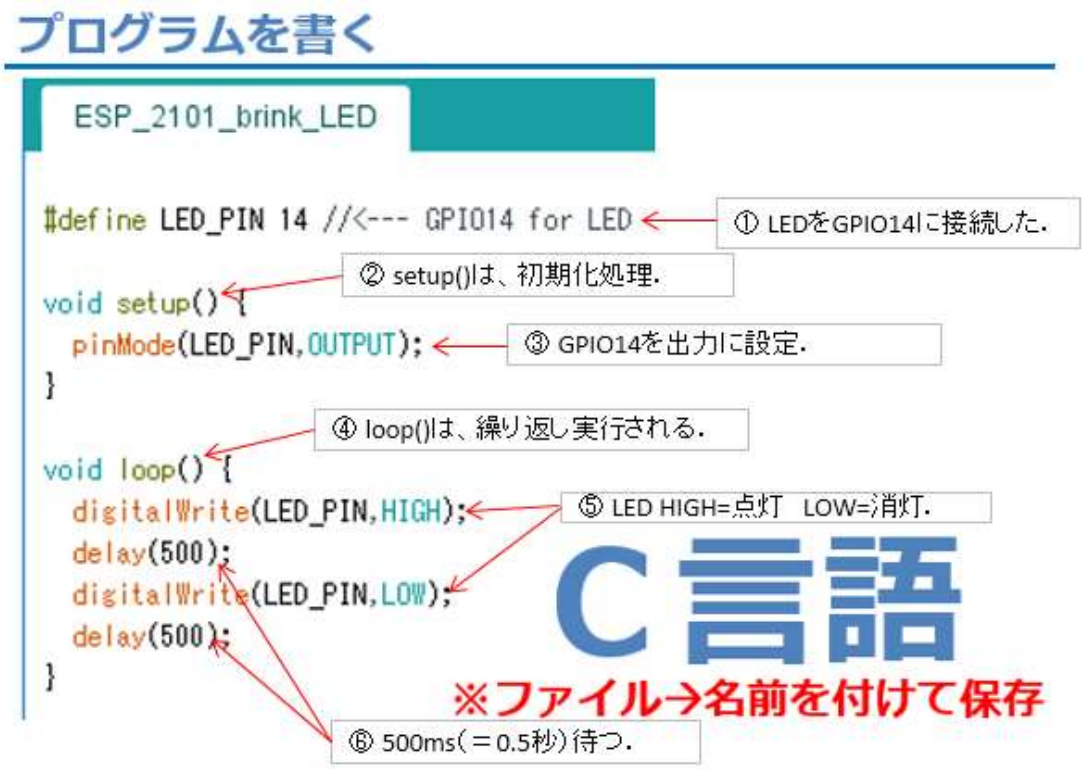


図 34

IDE ウインドウ中央の背景が白い部分がソースコードエディタになっていますので、そこにプログラムを記述します。

【重要】

基本的に Arduino マイコンと同様の言語体系となっています。Arduino 言語は C/C++をベースにして、C 言語のすべての構造と、いくつかの C++の機能をサポートしています。

今回の LED 点滅プログラムは、上の図の通り 10 行程度ですから、容易に入力できるでしょう。もしコメントを入力する場合は、スラッシュ 2 つ【//】やスラッシュ＋アスタリスク【/*】とアスタリスク＋スラッシュ【*/】を利用します。詳細は C または C++言語の仕様を調べて下さい。

特に、今回はGPIO14にLEDを接続したので、①の部分で定義しています。また②setup()は、どのプログラムでも共通の初期化処理をまとめて記述する関数として名称が決まっています。④loop()は、繰り返し呼び出される関数で通常の処理を全てここで行います。(※割込みなどは別途記述するのですが、この講座では割込みを使用しないので、別の機会に解説をしたいと思います。) ③pinMode()は、LEDへの制御用としてマイコンの汎用入出力GPIO14を出力に設定しています。⑤digitalWrite()は、GPIOポートに（※このマイコンでは入出力や通信の為にアクセスする部分をポートと呼びます。）HIGH(=1)かLOW(=0)を書き込みます。これにより、LEDが点灯(GPIO=1)したり消灯（GPIO=0）したりします。⑥deley()は、指定の時間、プログラムを一時停止します。単位はms（1/1000秒）です。

初期化が行われたマイコンシステムは、LEDへの出力が設定されて、loop()が繰り返し実行されることにより、0.5秒ごとにLEDが点滅を繰り返すという動作をします。

ソースコードを入力したら、名前を付けて保存してください。

PCと接続
【USBケーブル使用】

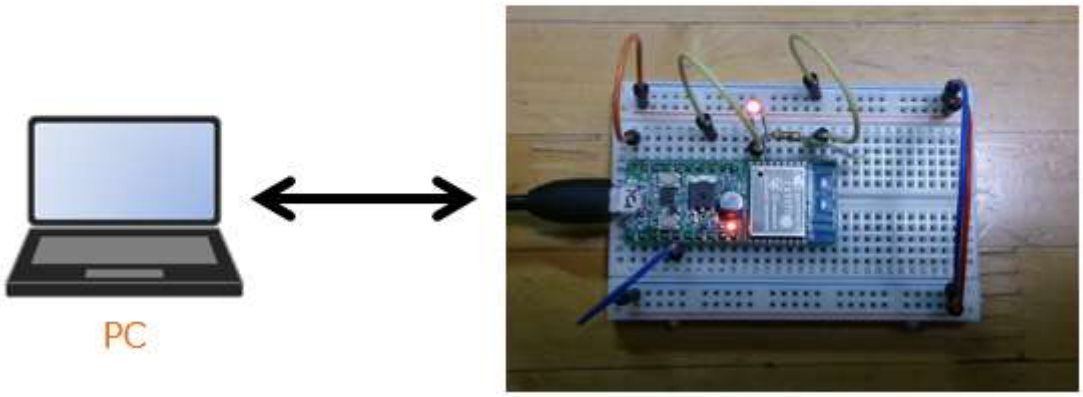


図 35

プログラムを保存したら、USB ケーブルで PC と WiFi マイコンモジュールを接続します。

【注意】

上の図右の写真では、LED が 2 つ点灯していますが、これは分かり易いように点灯しているところを撮影したものです。実際は、WiFi マイコンモジュールの金属ケース付近にある小さな電源 LED だけが赤く点灯して電源が投入されていることを示します。

COMポート番号確認

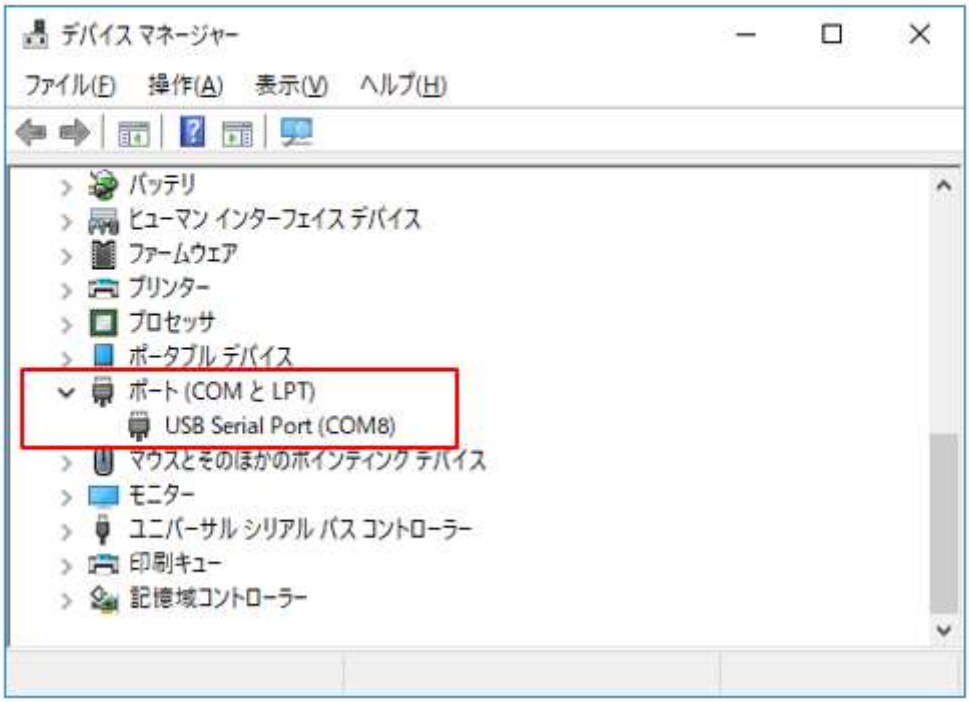


図 36

初めて、USB ケーブルで PC と WiFi マイコンモジュールを接続したとき、仮想 COM ポートドライバーのインストールが行われますので、終了するまで待ちます。終わりましたら、デバイスマネージャのウインドウで USB Serial Port の COM 番号を確認してください。ドライバーがインストールされない場合は、次頁の【注意】を参照して下さい。

【重要】

ここで確認した COM ポート番号は、他の WiFi マイコンモジュールを接続すると、異なる番号に設定されます。PC と接続するのが初めて（または、かなり時間が経過している）であれば、デバイスマネージャで番号を確認してください。同じものを繰り返し使用するときは、COM ポート番号は変わりません。

シリアルポートの設定

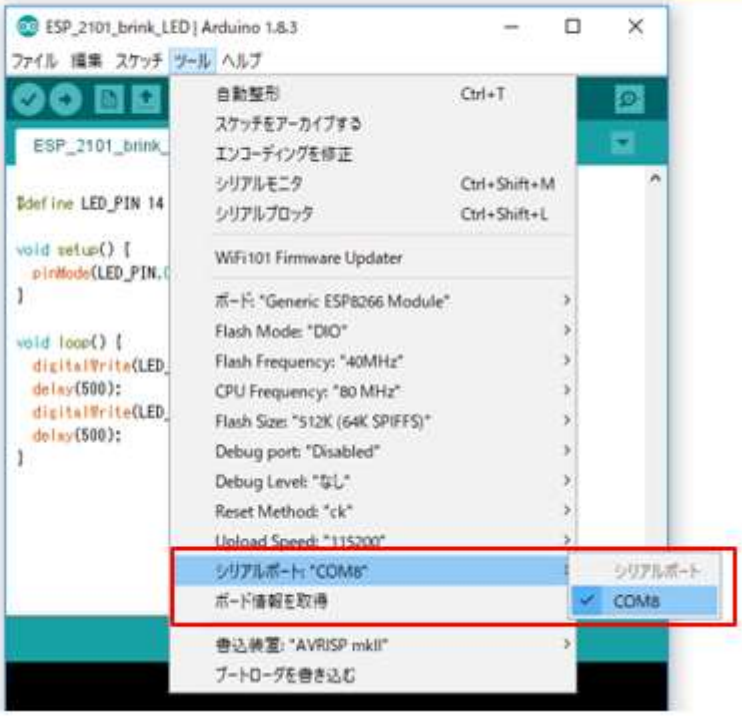


図 37

【ツール→シリアルポート】と辿り、確認した COM ポート番号を選択して、チェックを入れます。（上図）この番号は、Arduino IDE と WiFi マイコンモジュールが通信する場所（シリアルポート）です。一度設定すると次回以後も有効になっているのですが、WiFi マイコンモジュールへのプログラム書き込みが巧くいかない場合などは、確認をして再度設定することもあります。

次は WiFi マイコンモジュールへの書込み準備です。

【注意】仮想 COM ポートドライバーがインストールされていない場合は、FTDI 社（http://www.ftdichip.com/Drivers/VCP.htm）にある VCP ドライバーをインストールして下さい。

書き込み可能にする

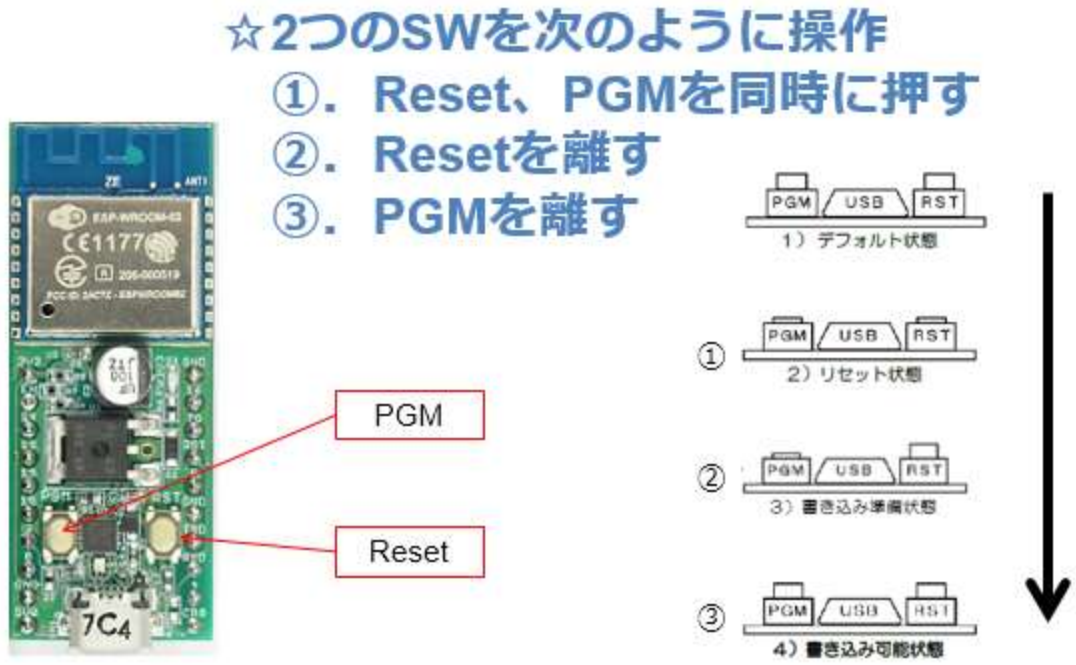


図　38

WiFi マイコンモジュールは、通常は書き込まれているプログラムを実行しているので、IDE からの書込みに対して、なにも反応しませんので【これからプログラムを書込むゾ！！】と教えてやらなければいけません。その際、WiFi マイコンモジュール上の 2 つの SW を使います。

その手順は次のとおりです。（上図）

① まず、PGM と Reset SW を同時に押します。

② 次に、Reset SW だけ離します。

③ 最後に PGM SW を離します。

これで、WiFi マイコンモジュールは、IDE からのプログラム書込みモードになります。

◇プログラムのコンパイルと書込み

図 39

これから、プログラムのコンパイル・リンクと WiFi マイコンモジュールへの書込みを行います。IDE の上の方にある、右向き矢印ボタンをクリックしてください。(上図)

図 40

ボタンを押すと上の図の様に、コンパイルから書込みまで、一連の流れで実行されて、最後に書込み完了のメッセージが表示されます。

動作の様子

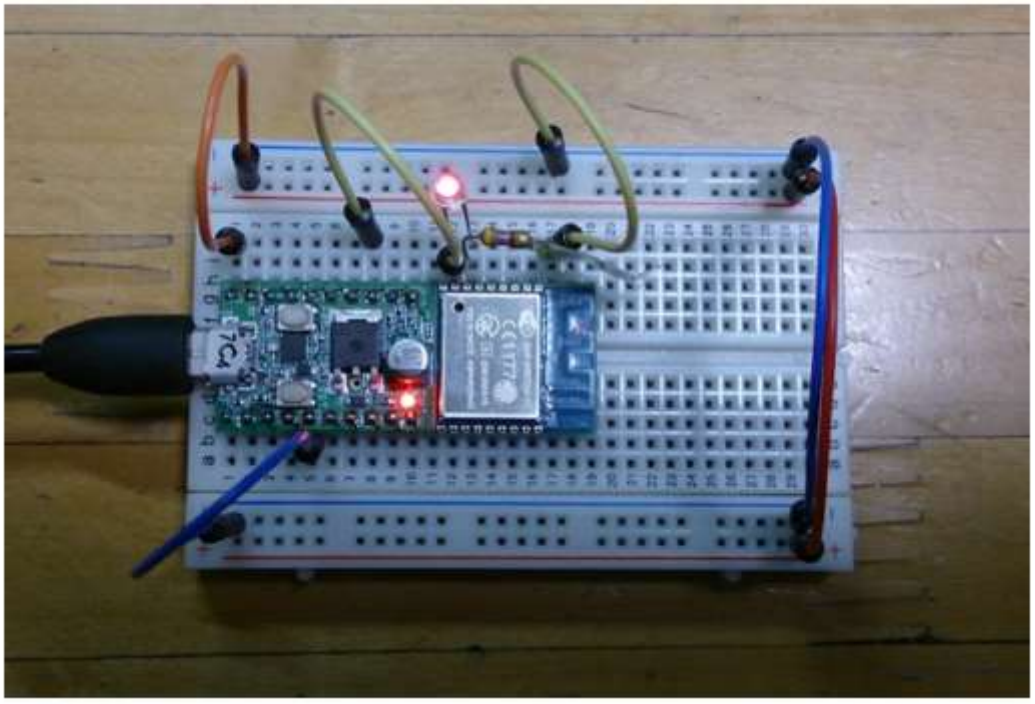


図 41

書込みが終了すると、マイコンは自動で Reset されて、プログラムの実行が開始されます。配線とソースコードの記述に間違いが無ければ、配線した LED がおよそ 0.5 秒間隔で点滅を繰り返します。

これが、第 1 回目の実験の結果です。LED は小さいので、これを点滅させることの意味を理解しにくいのですが、LED の ON/OFF 信号を取り出して外部機器・装置の ON/OFF 制御が行えます。また ON/OFF の周期を短くして制御すると PWM（Pulse Width Modulation）などに対応できます。制御対象が AC（交流）でしたら、リレーや SSR（Solid State Relay：ソリッドステートリレー）を使うことにより、ON/OFF 制御できます。

最後に、WiFiマイコンモジュールを使った開発全体の手順を振り返っておきましょう。

----------<< WiFiマイコン開発手順 >>----------

1. 回路の制作
2. IDEのダウンロード・インストール
3. ボードマネージャのダウンロード・インストール
4. WiFiマイコンモジュールのソースコード記述
5. PCとWiFiマイコンモジュールの接続（USBケーブル）
   ※この際、USB-シリアルドライバがインストールされる
6. COMポート番号の確認とIDEへの設定
7. コンパイル・リンク・書込み
8. 動作確認

上の手順は、初めてWiFiマイコンモジュールのシステム開発を行う際の手順です。次の開発からは、2，3の手順が省略できます。使用するWiFiマイコンモジュールが同じものであれば6の手順もIDEの設定を確認するだけで大丈夫です。別のWiFiマイコンモジュールを使用する場合は、同じシステムを作る場合でもCOMポート番号が変わります（前述）ので6番の手順を行います。また、開発対象のデバイスやシステムによっては、専用のライブラリをインストールする必要がありますので、その場合は、手順3と4の間にライブラリをライブラリマネージャでインストールします。今後の実験でも、概ね必要な手順は説明しますので、全体は上の1～8の手順だと理解しておいてください。

第2回　SW

第１回はLEDの点滅でした。これはLEDに対して一方的にWiFiマイコンからON/OFF信号を出力していました。第２回はSWの状態を読み込んで、その状態に対応してLEDが点灯・消灯するシステムを作りましょう。

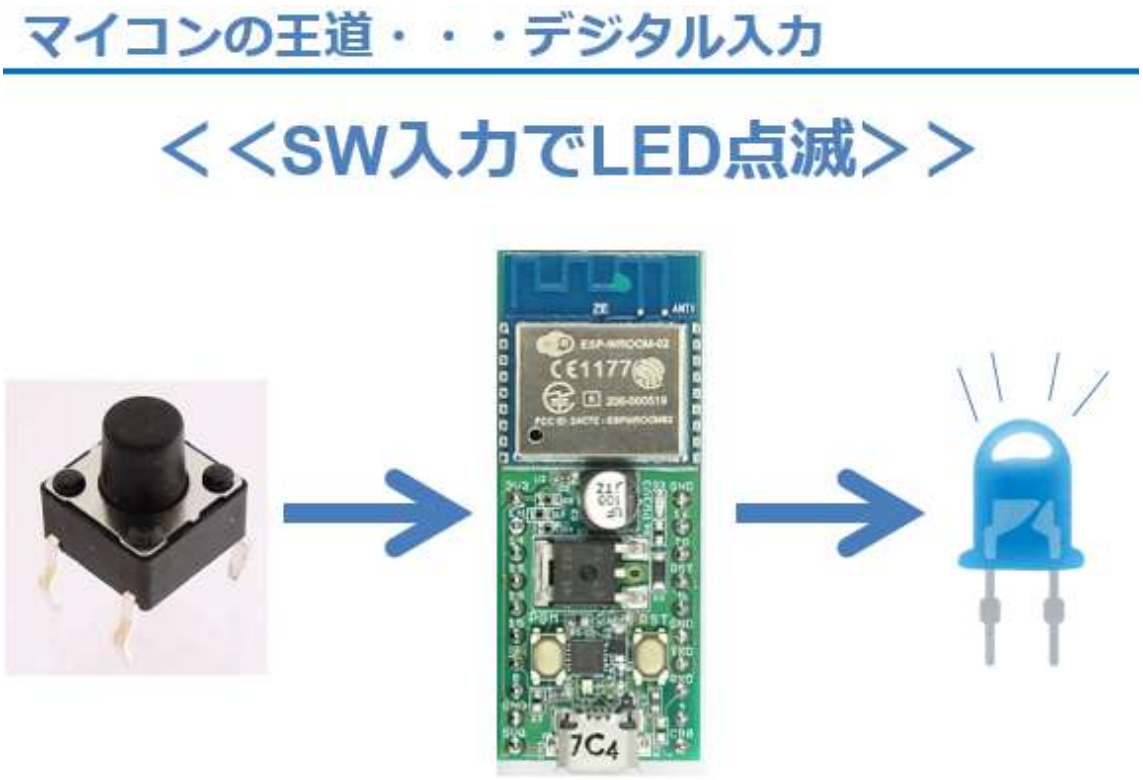


図　42

LEDのON/OFF信号がデジタル出力（DO）であったのに対して、SWからの入力はデジタル入力（DI）になります。システムが動作するとSW操作にWiFiマイコンが反応しているように見えます。

◇全体構成とパーツ

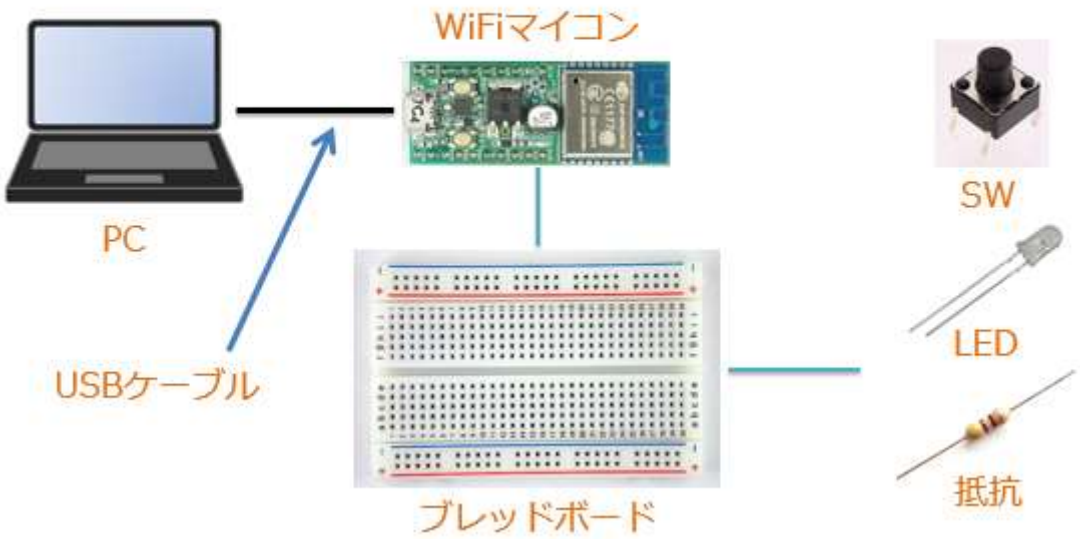


図 43

システムの全体構成を上図に示します。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個
8. SW×1 個

前回のパーツに SW が加わってしまします。ここで、SW の構造について少し説明します。

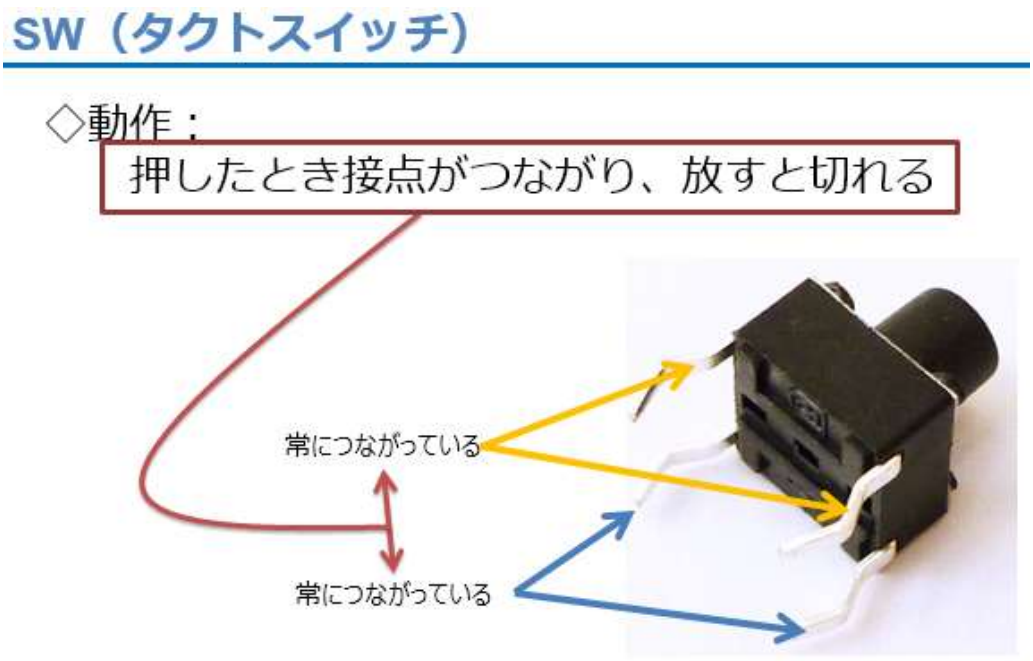


図　44

今回使用するSWは【タクトスイッチ】という名称の小さなSWです。動作は単純で、押したときに接点が繋がり、放すと切れるというものです。上図で湾曲した脚ピンがありますが、湾曲している向かい合った対になるピンが内部で接続されています。これが2対あり、ボタンを押すと、その2対が内部で繋がる仕組みです。

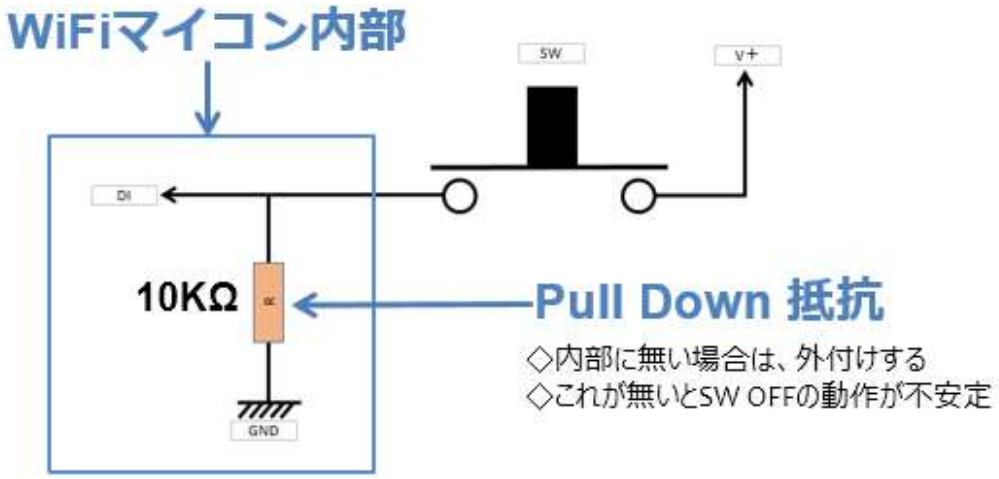


図　45

今回のSWの接続は上図のように描けます。SWを横から見た様子と考えて下さい。SWの脚ピンは湾曲している1対がSW下の○印です。片側がV+に接続されています。V+とは電源の＋側を指します。反対側はマイコン内部で10KΩの抵抗とマイコンのDI（デジタル入力の意味）に接続されています。10KΩの抵抗はGNDに接続されています。まずSWを押したときの状態をマイコン側から読み込むと、DIの部分はSWを介してV+に接続された状態となっているので、電圧レベルはV+(=HIGH)となり=1です。反対にSWを離した状態の場合は、DIの部分は電圧レベルが0V（※抵抗でGNDに接続されている）で読み込んでも=0(LOW)です。ここで10KΩの抵抗は何のためにあるかというと、なにも接続されていない解放状態のDIの信号は不安定になる可能性があるので、解放状態でも確実にLOWレベルにするために抵抗を介してGNDにつなぎます。抵抗が無いと、SWが押されたときV+とGNDが直接つながってしまい、ショート状態になります。

【重要】

この様に確実にLOWレベルにしておく為の抵抗をプルダウン(PULL DOWN)抵抗と云います。

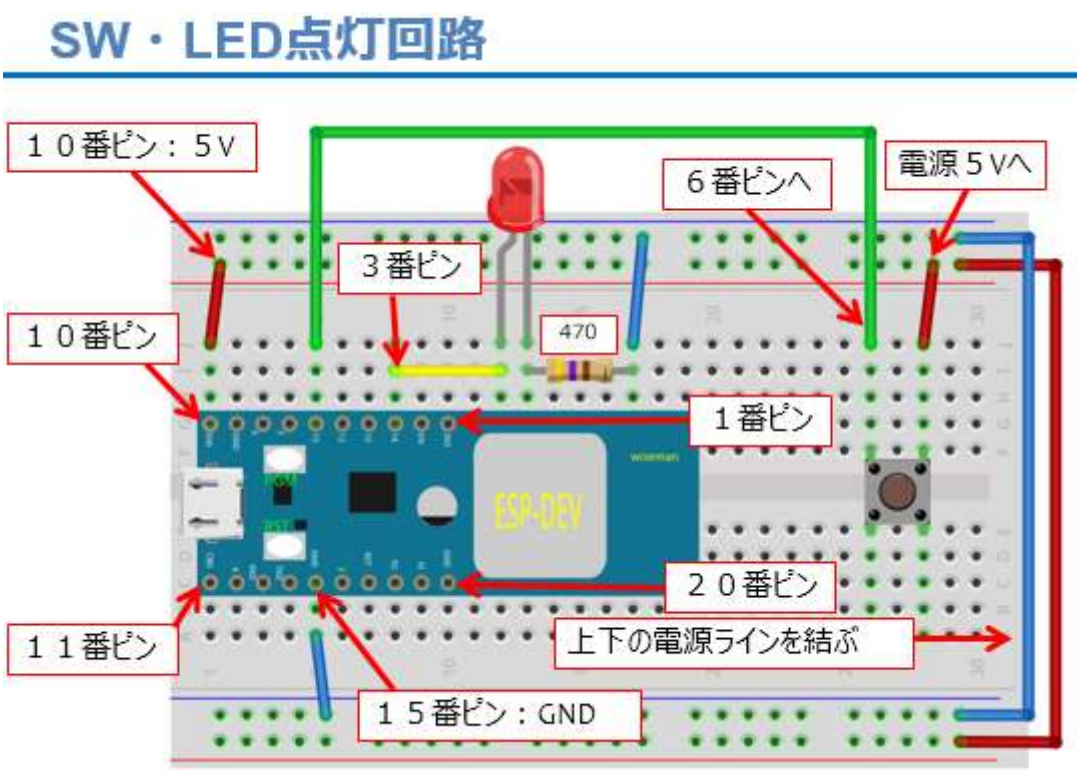


図　46

まず、上の図に従って配線しましょう。前回の LED 点滅の回路に SW が追加になっています。SW は湾曲した脚で、ブレッドボード中央の溝を跨ぐように配置してください。図で SW 右上のピンは赤いジャンパー線で 5V に配線します。SW 左上のピンは緑色のジャンパー線で 6 番ピンの GPIO15 に配線します。ジャンパー線は 2 本増えただけですが、油断せず確実に配線してください。新しい配線をすると、既存の配線が外れたり緩んだりすることもありますので、確認を行ってください。

実際に配線した様子

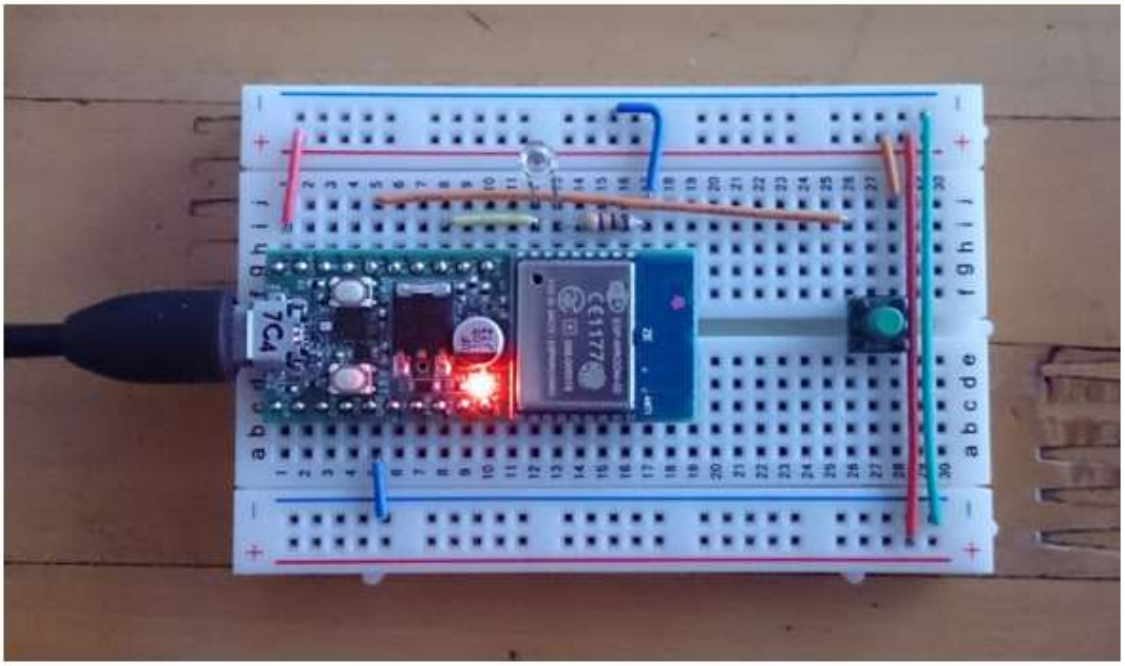


図　47

実際に配線した様子が上の写真です。

【重要】

この回路は、直後の講座でも使いますので、今回の動作確認が終わっても保存しておいて下さい。

回路ができたら、次はプログラムの作成です。

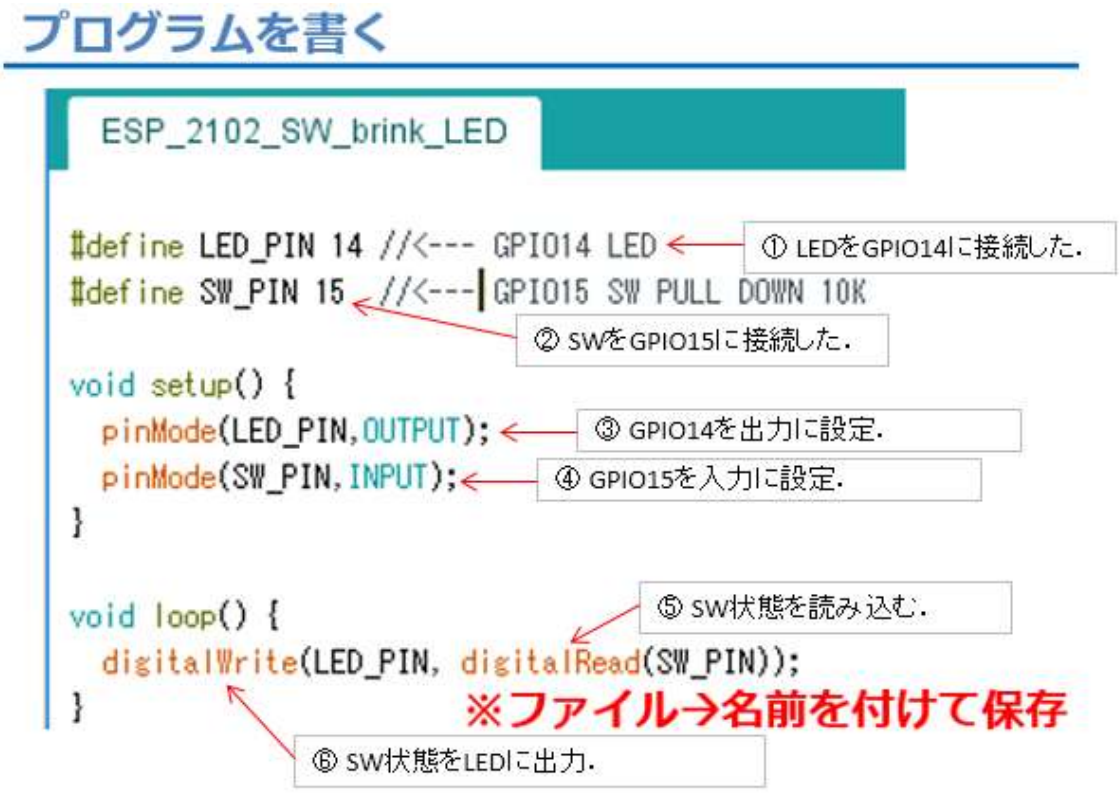


図　48

IDE のファイルメニューで新規を選択して、新しいスケッチ（Arduino IDE ではプログラムをスケッチと呼びます。）を開きます。ウインドウの中央、白い部分に上の図のソースコードを入力してください。

①LED を GPIO14 に接続したので 14 を定義しています。②SW は GPIO15 に接続しました。③GPIO14 を出力に設定。④GPIO15 を入力に設定。⑤SW 状態を読み込みます。状態は degitalRead()の戻り値として返されますが、次の⑥digitalWrite ()で LED の接続されている GPIO14 にそのまま出力します。

プログラムができたら、名前を付けて保存しておきましょう。

次は、マイコンと PC を USB ケーブルで接続します。

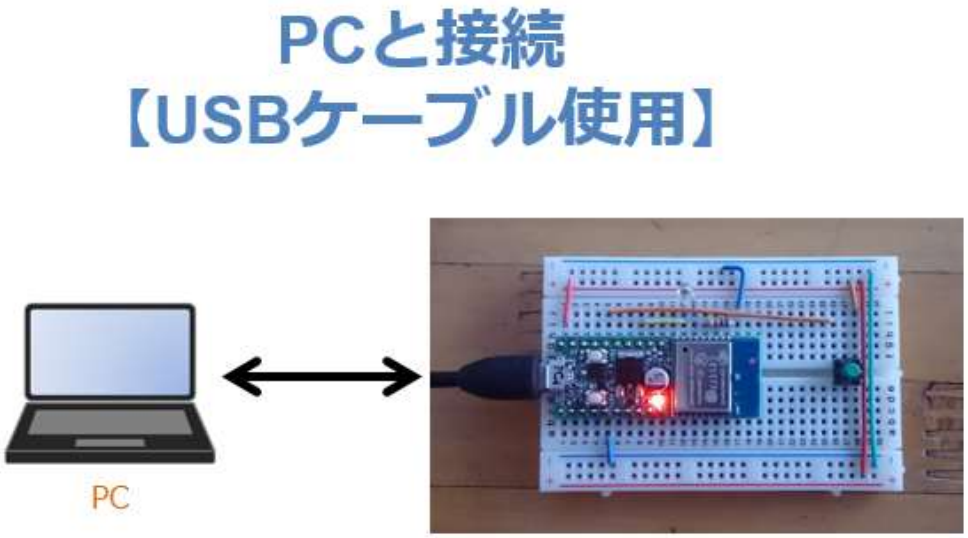


図　49

すでに、第１回目で USB-シリアルドライバがインストールされていて、COM ポート番号も設定済みだと思いますが、ここでは念のためにデバイスマネージャで確認して、IDE の COM ポート番号を確認・設定しておきます。

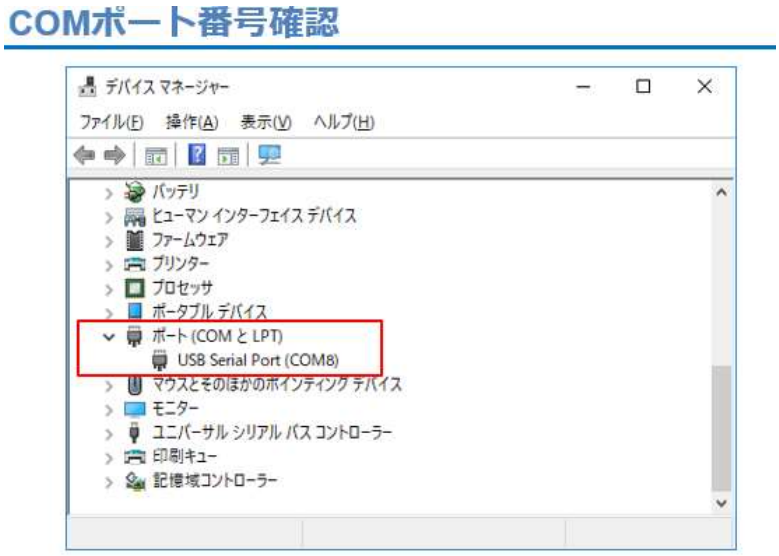


図　50

図　51

【ツール→シリアルポート】で COM ポート番号を確認・設定します（上図）。下図に従いマイコンへの書込み準備を行います。

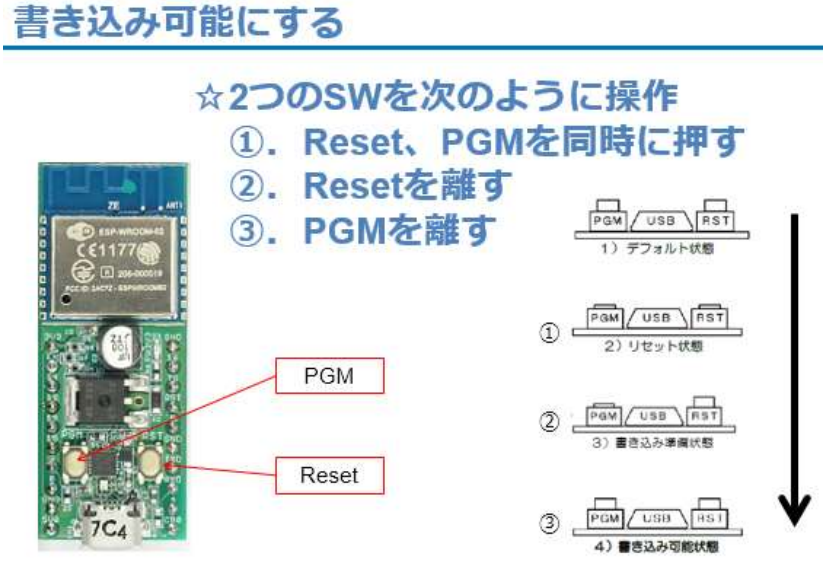


図　52

◇プログラムのコンパイルと書込み

図 53

上図に示す右向き矢印ボタンを押すと、コンパイルからマイコンへの書込みまでが自動で行われます。

プログラムの書込み

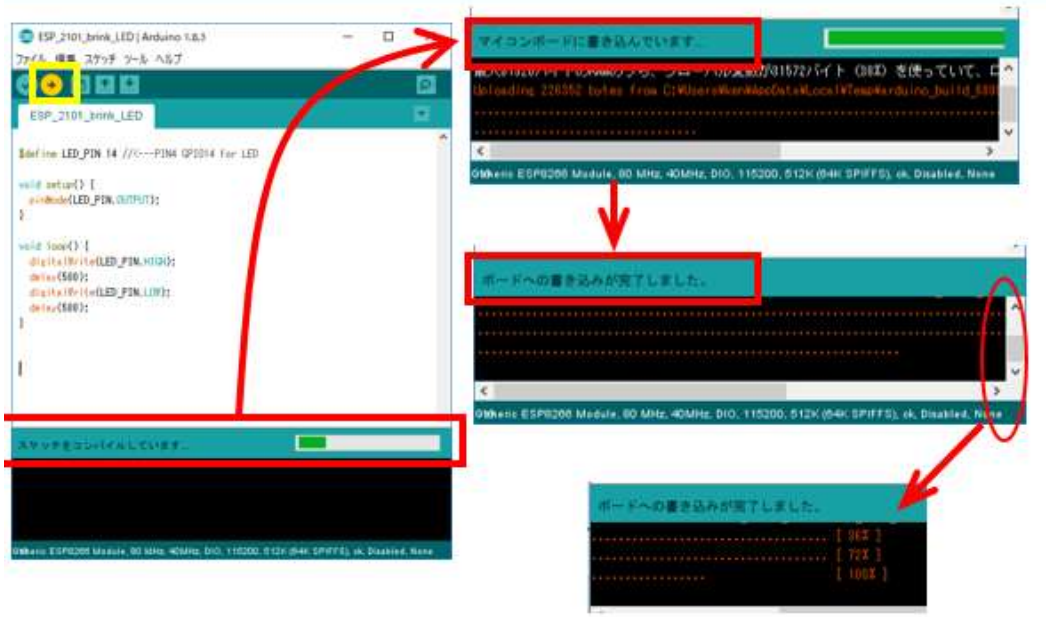


図 54

書込みが終了するとメッセージが表示されて、マイコンに Reset が掛かりプログラムは開始されています。出来上がったシステムの動作確認をしましょう。

動作の様子

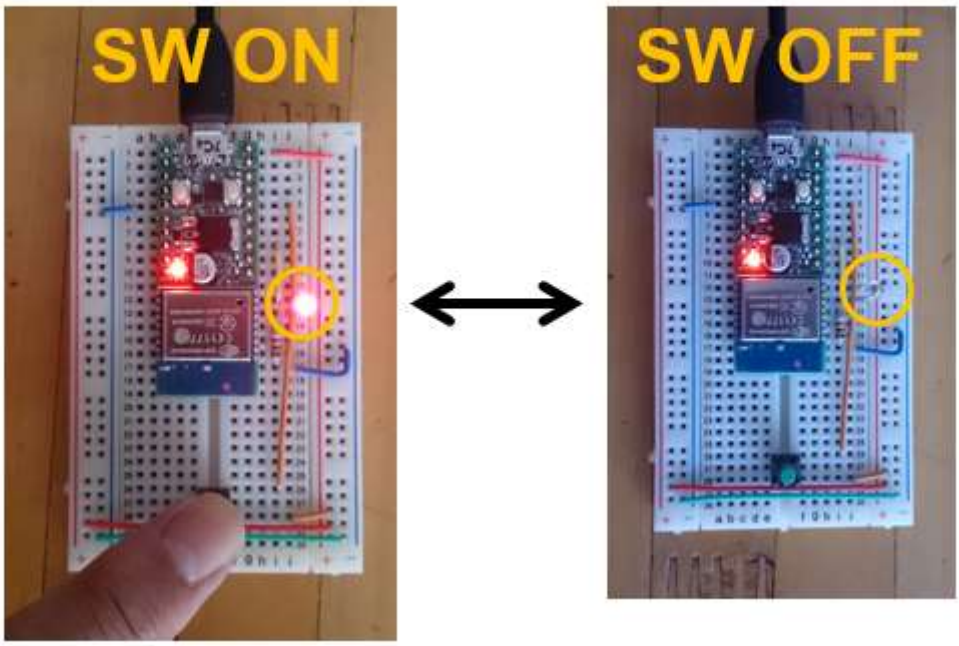


図 55

まずは、SW を押してみます。同時に LED が点灯します。（写真左）SW を解放すると LED は消灯します（写真右）。いかがでしょうか、SW 状態に反応する（または SW 状態を反映する）システムができたでしょうか。これで、この WiFi マイコンモジュールのデジタル入力・出力が使えるようになりました。今回の講座の中で解説した SW の入力を読むポートに接続されている PULL DOWN 抵抗を覚えておいてください。

第3回　シリアル通信【送信】

今回は、開発のために使用しているPCに対してシリアル通信を行い、WiFiマイコンモジュールからメッセージを送信してみましょう。

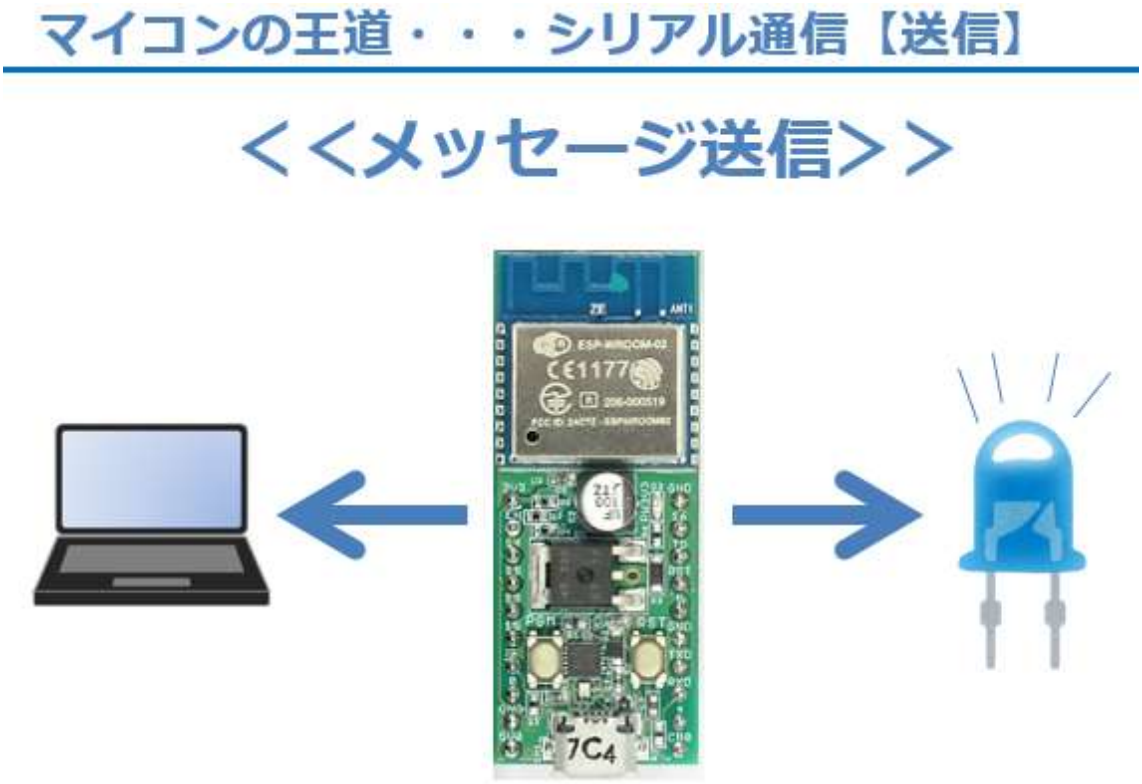


図　56

USBケーブルで接続されているPCは、プログラムをWiFiマイコンモジュールのフラッシュメモリに書き込むために、通信を行っていますが、その通信ポートを使って、マイコン側で作るメッセージを送信すると同時にLEDの点灯を行ってみます。

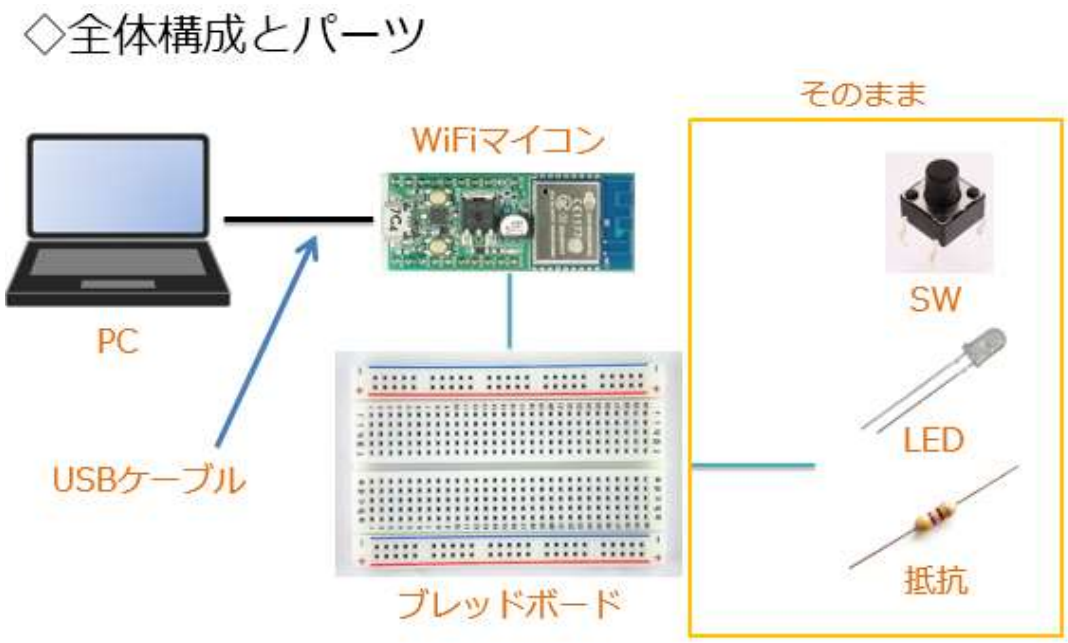


図　57

上図にシステムの全体構成を示します。前回のものと同じ内容です。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個
8. SW×1 個

回路を次の図に示します。こちらも前回と同じです。

SW・LED点灯回路

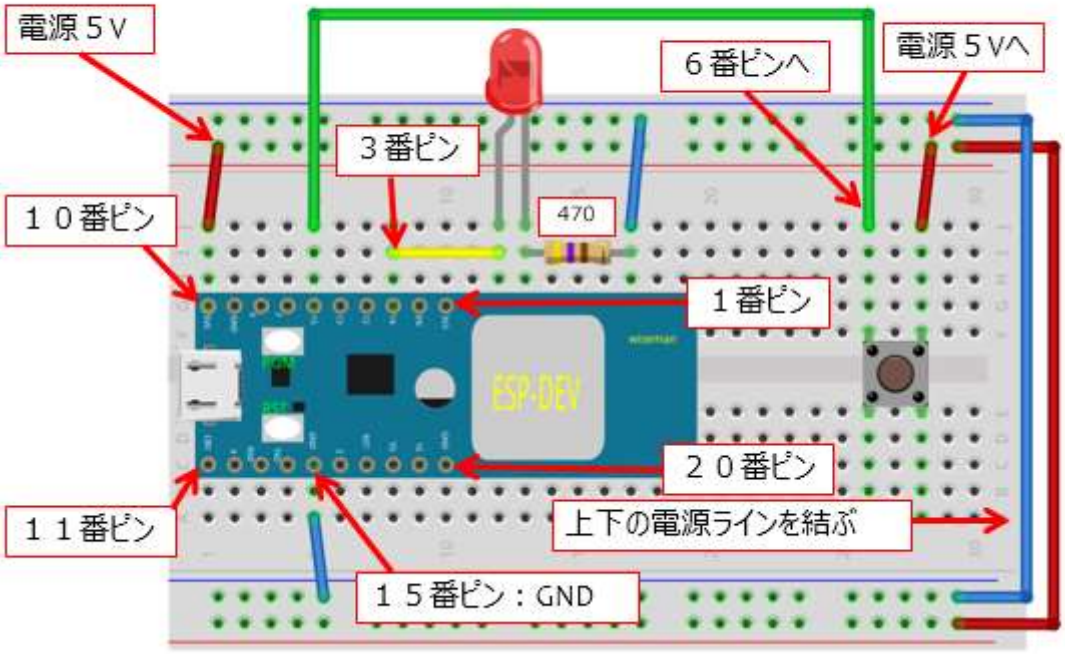


図　58

実際に配線した様子

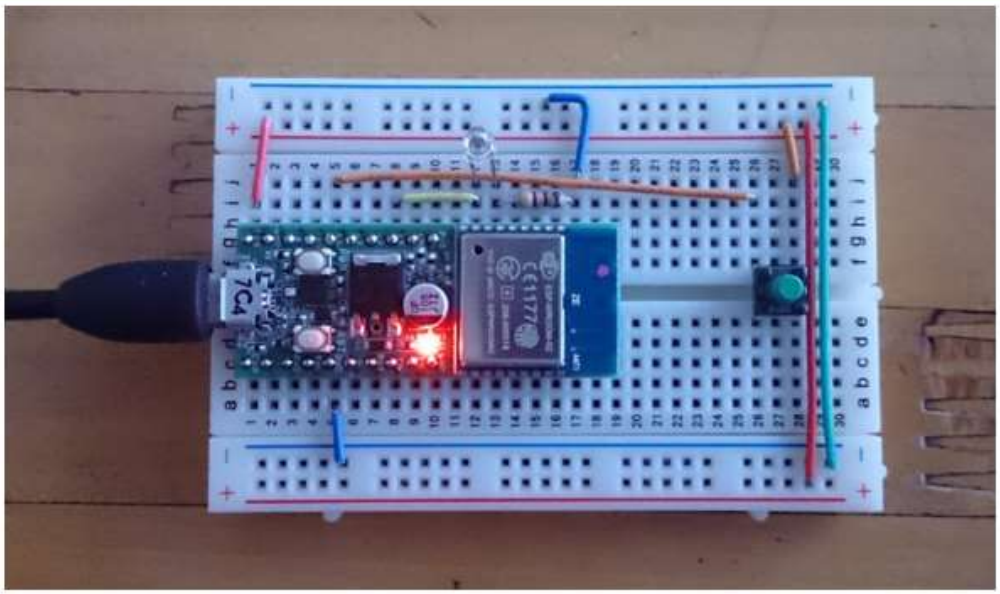


図　59

実際に配線した様子を上の図に示します。前回の講座後に、改めて回路を作られる方は、第 2 回 SW を参照して作成して下さい。

【重要】
この回路は、直後の講座でも使いますので、今回の動作確認が終わっても保存しておいて下さい。

回路ができたら、次はプログラムの作成です。プログラムの作成に必要な IDE の準備についての詳細は、第 1 回 LED 点滅を参照して下さい。

◇Arduino 統合開発環境 IDE

図 60

上図デスクトップ上の IDE ショートカットにより、IDE を起動します。

図　61

ツールメニューのプルダウンで、Generic ESP8266 Module が選択されていることを確認してください。異なるボードの場合は、上図に従い、ボードを選択してください。

IDE の中央にソースコードを入力します。（下図）

図 62

IDE に上のソースコードを入力してください。新たな部分は次の 2 点です。①シリアルポートの初期化です、シリアル通信は通信速度を指定できます。ここでは 9600bps（bps：bit/sec）の速度で初期化しています。②シリアル通信で文字列を送信します。()内部のパラメータで【""】ダブルクォーテーションで挟んだ文字列を、PC に向けて改行付きで送信します。Serial.println()の関数名の最後の 2 文字に【ln】が付いているのが、改行コード付きの送信関数名です。

loop()内では、固定のメッセージを PC に送信して、LED を 1 秒点灯し、1 秒消灯しています。この結果、1 秒ごとに LED 点滅を行いながら、固定メッセージをシリアル送信する動作を繰り返します。

ソースコードを入力したら、名前を付けて保存しておきましょう。

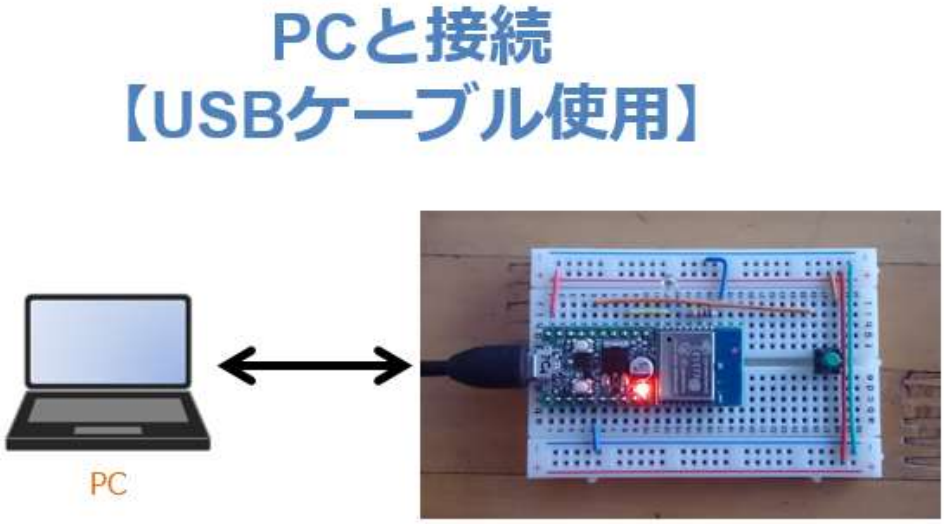


図　63

すでに、第１回目で USB-シリアルドライバがインストールされていて、COM ポート番号も設定済みだと思いますが、ここでは念のためにデバイスマネージャで確認して、IDE の COM ポート番号を確認・設定しておきます。

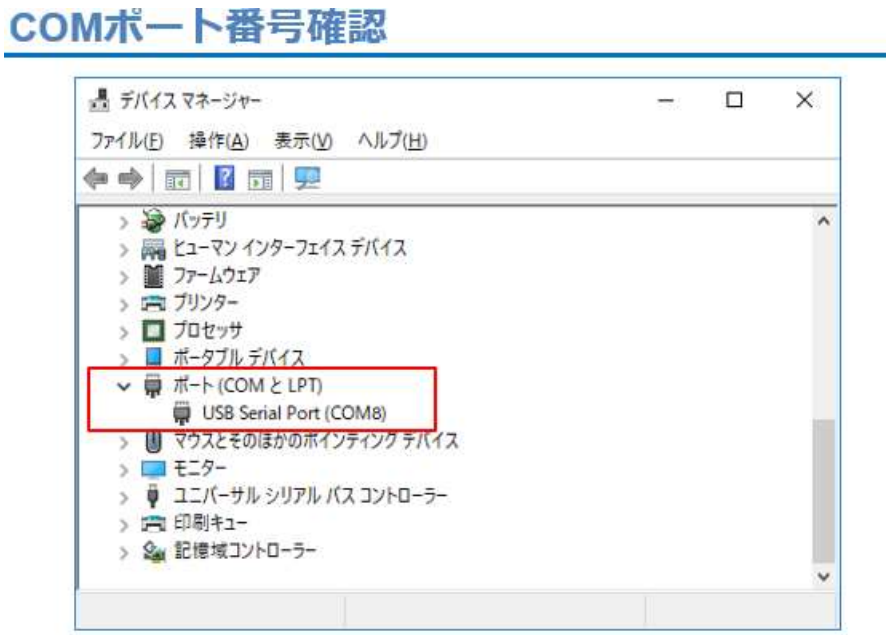


図　64

図　65

【ツール→シリアルポート】で COM ポート番号を確認・設定します（上図）。

次の図に従いマイコンへの書込み準備を行います。

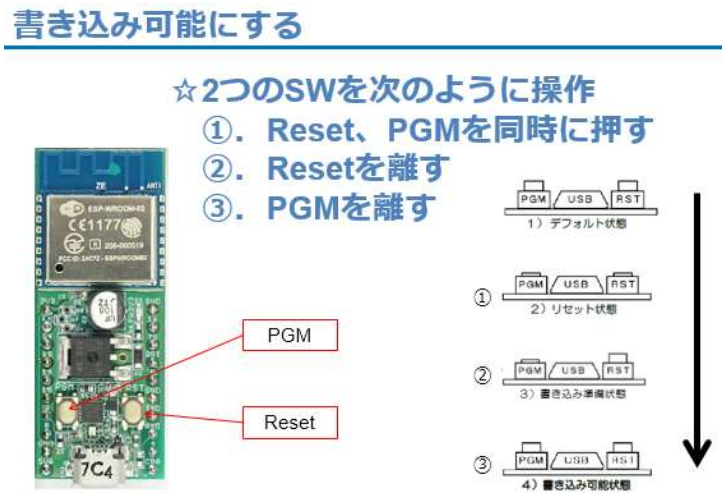


図　66

◇プログラムのコンパイルと書込み

図 67

上図に示す右向き矢印ボタンを押すと、コンパイルからマイコンへの書込みまで自動で行われます。

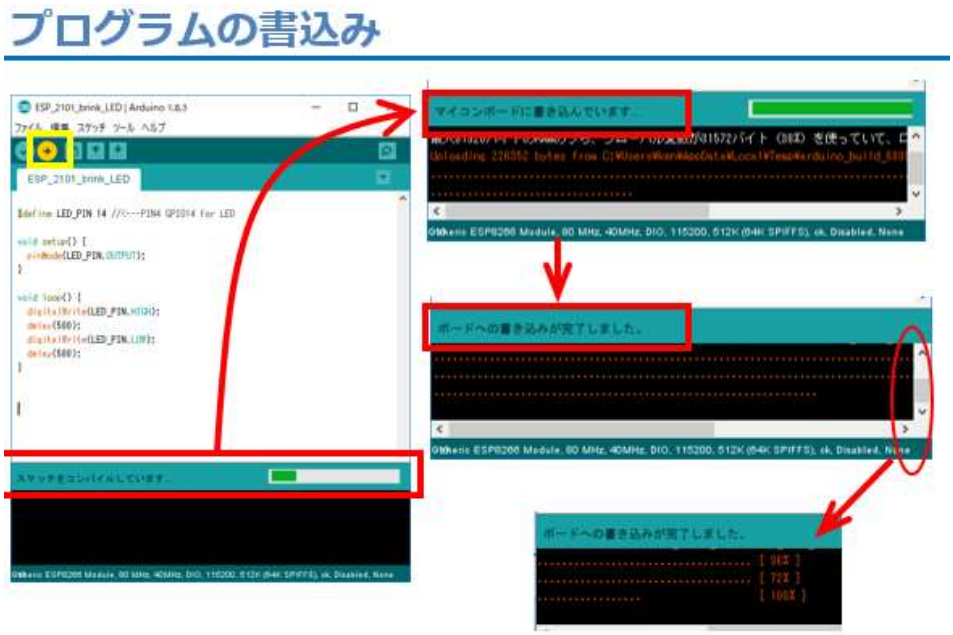


図 68

書込みが終了すると、メッセージが表示されてマイコンに Reset が掛かりプログラムは開始されています。出来上がったシステムの動作確認をしましょう。

動作確認

図　69

IDE ウインドウの右上にある虫眼鏡マークのボタンをクリックすると、シリアルモニターのウインドウが開きます。シリアルモニターウインドウの下部にある通信速度のプルダウンで、setup()関数で設定した 9600bps を選択します。通信速度が合っていないと、何も表示されなかったり文字化けを起こしたりしてしまいますので、忘れない様にしてください。

通信速度を合わせれば、WiFi マイコンが送信しているメッセージがウインドウに約 1 秒間隔で表示され続けます。

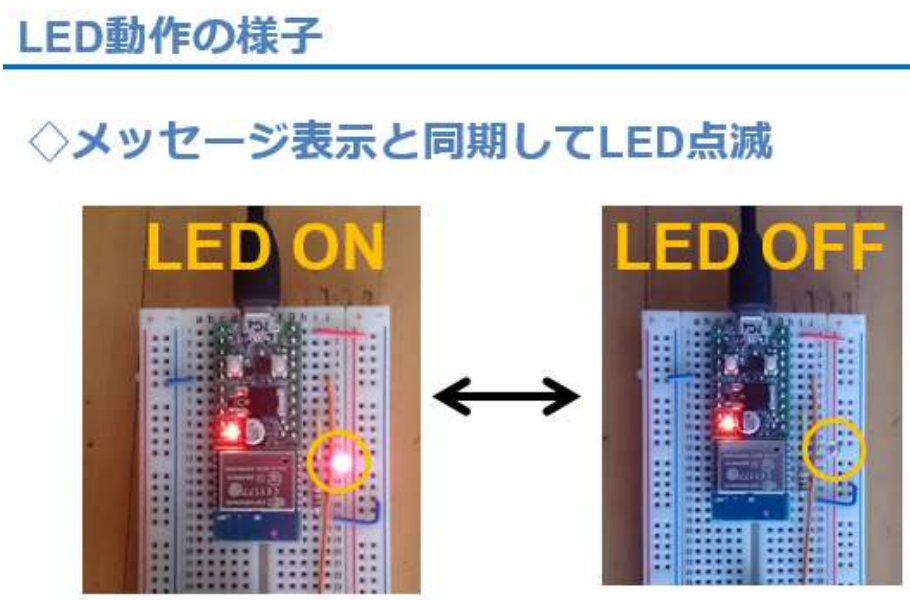


図 70

この時 WiFi マイコンの回路では LED が点滅をしていますが、シリアルモニターのメッセージと合わせて見ていると、メッセージ受信に同期して LED の点滅する動作を確認できます。

【重要】

ここでは、固定のメッセージを送信していますが、このメッセージ文字列の内容をダイナミックに変化するデータなどで編集すれば、マイコンが計測した情報を監視する用途などに、シリアル通信が使えるようになります。今回は送信でしたが、次の講座では受信を行ってみましょう。

第4回　シリアル通信【受信】

シリアル通信の受信が行えるようになると、受信したメッセージ（コマンド等とも言いますが）の内容に対応する動作や処理を行わせることができて、リモコン操作ができるような仕組みが構築できます。

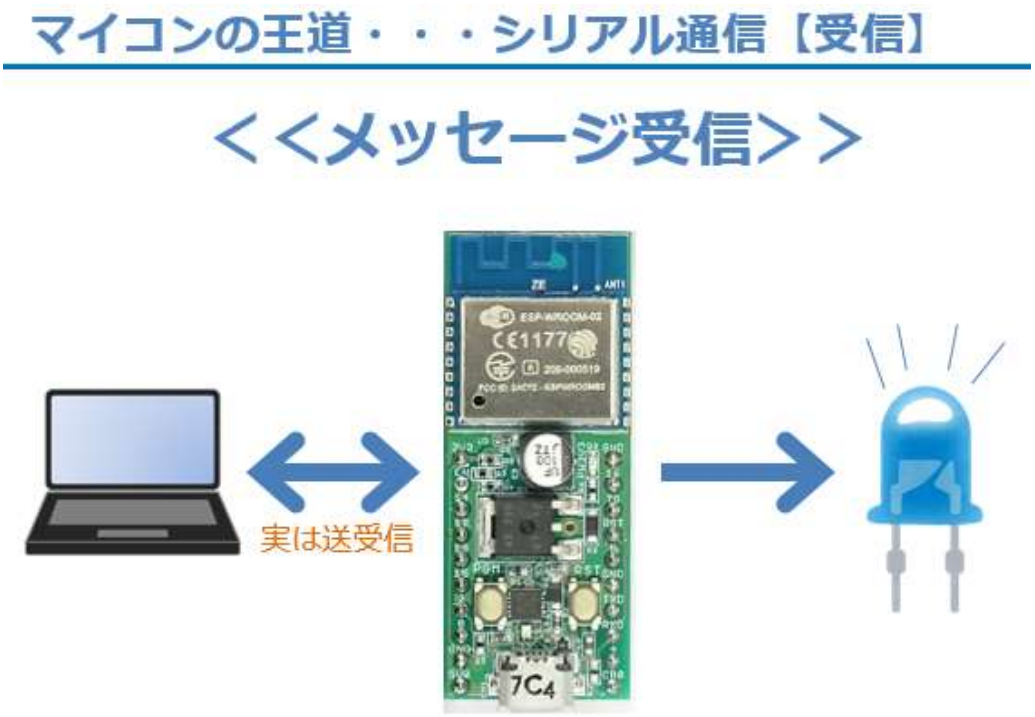


図　71

上の図は、前回と同じように見えますが、PC と WiFi マイコンが双方向の矢印で結ばれています。今回のテーマは、シリアル通信の【受信】ですが、せっかく送信もできるようになっているので、欲張って双方向の通信をしてみましょう。PC から送信するメッセージの電文内容によって、WiFi マイコンの外部に配線した LED を制御してみます。またその際の WiFi マイコンの振舞いを PC に送信します。

◇全体構成とパーツ

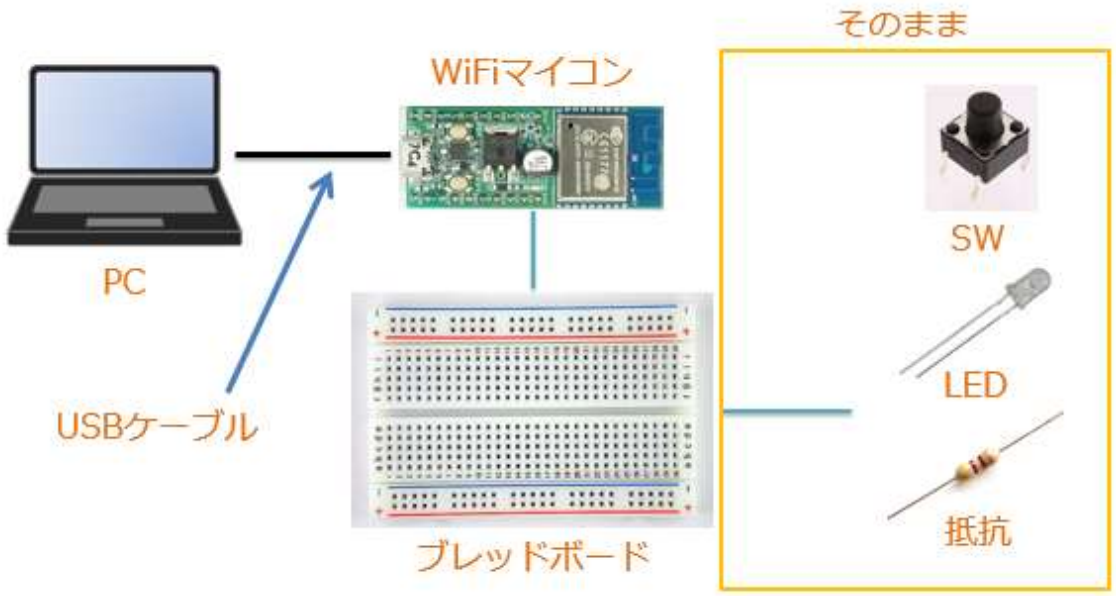


図　72

システムの全体構成は、上の図の通り、第２回と同じです。

必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個
8. SW×1 個

回路を次の図に示します。こちらも前回と同じです。

SW・LED点灯回路

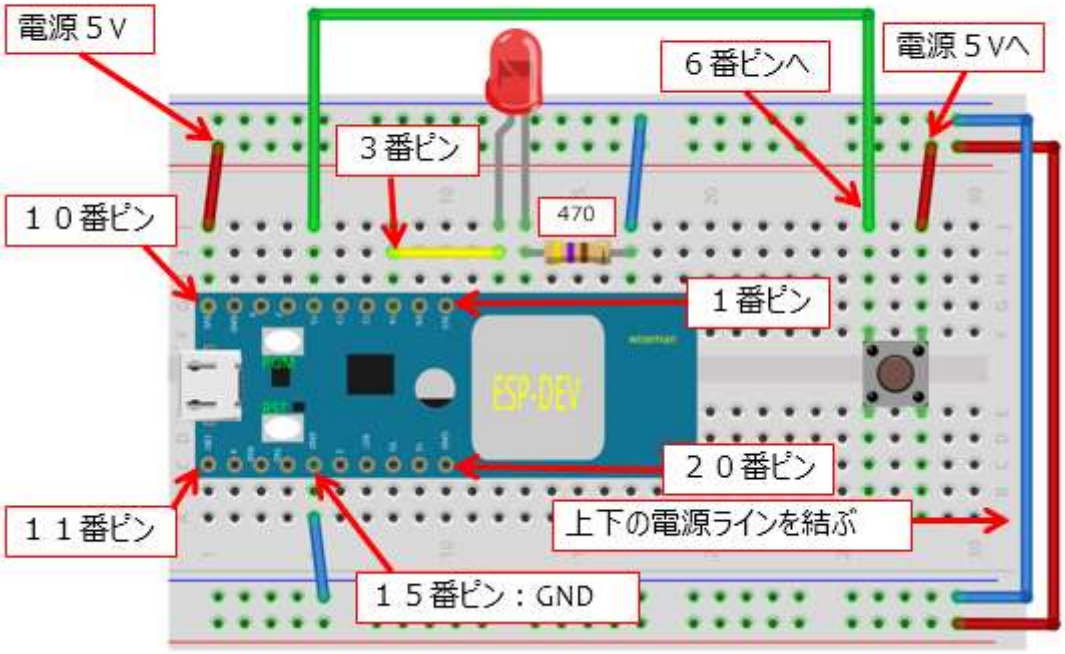


図　73

実際に配線した様子

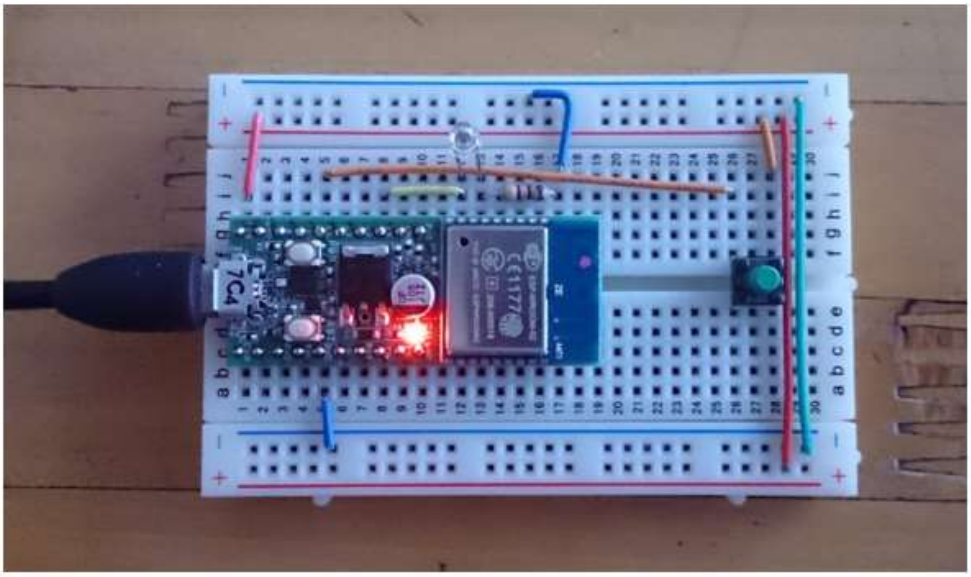


図　74

実際に配線した様子を上の図に示します。前回の講座後改めて回路を作られる方は、第 2 回 SW を参照して作成して下さい。

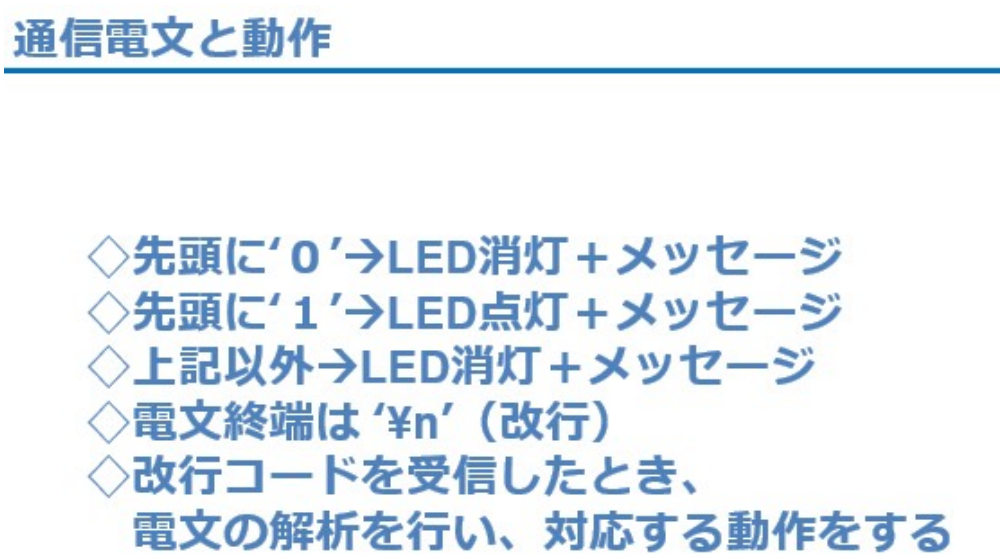


図 75

ここで、シリアル通信で使用する通信電文の設計をしておきましょう。上の図を見てください。通信に使用する電文は、文字（ASCII コード）により構成します。電文の先頭 1 文字目が数字の’0’か’1’またはそれ以外の 3 パターンで処理内容が変わります。電文の終端は’¥n’（改行）とし、’¥n’を受信したとき、電文の解析・対応処理を行います。

電文が設計出来たら、プログラムの作成です。必要な IDE の準備については第 1 回 LED 点滅を参照してください。

◇Arduino 統合開発環境 IDE

図　76

図　77

プログラム開発のための IDE の準備とボードの選択は、ここまでの講座で既に済んでいます。（上図）IDE の【ファイル→新規ファイル】と辿り、新しいスケッチにソースコードを記述しましょう。

プログラム

◇冒頭＋初期化

ESP_2104_Serial_Rx

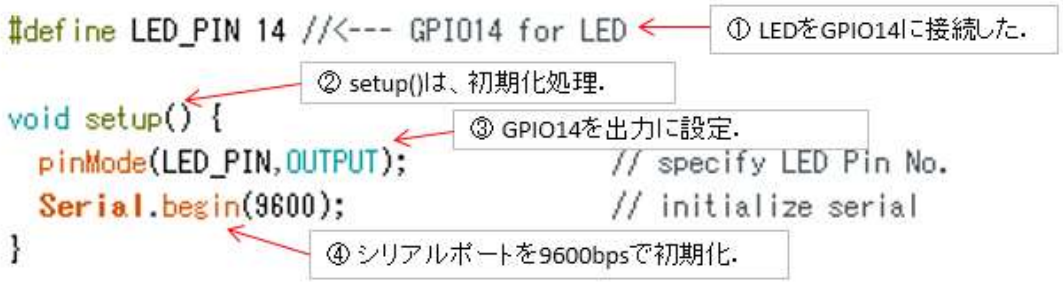


図　78

まず、冒頭の部分からです。

①LED 用の GPIO 番号定義

②初期化の専用関数は setup()という決まった名前

③LED 用に GPIO14 を出力に設定

④シリアルポートを 9600bps で初期化

次はメイン処理部分です。

プログラム　◇メイン処理部の全体

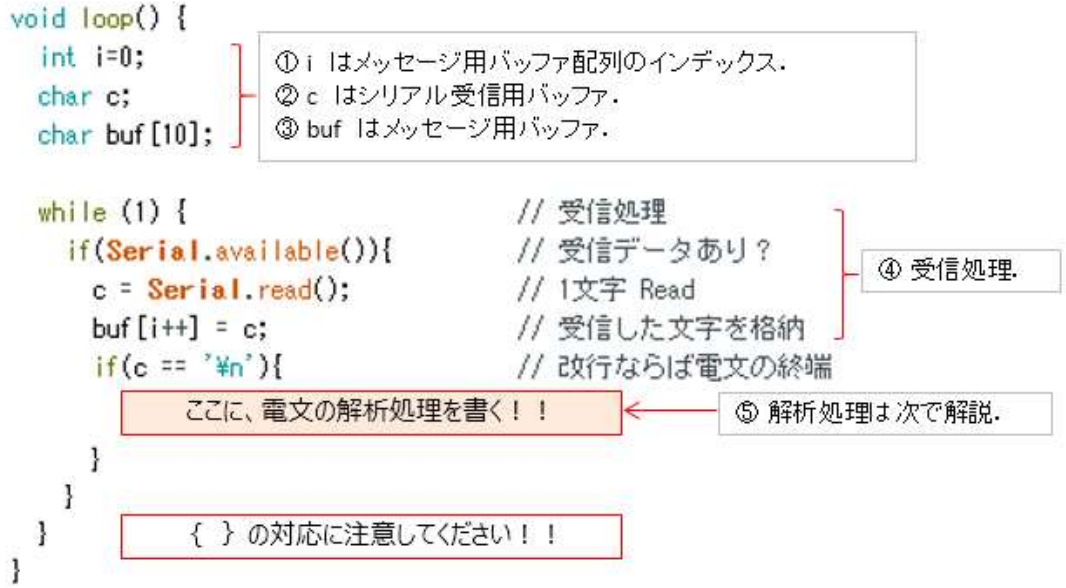


図　79

loop()関数は繰り返し実行される関数で、名称固定です。

①シリアル通信で受信される文字を③の配列に格納するインデックス

②シリアル通信で受信する 1 文字用のバッファ．10 文字分で十分

③受信した文字データを格納して電文として編成するバッファ

④受信処理部．

Serial.available()で受信しているデータがあるかを調べる

Serial.read()で 1byte を読み込む

受信した 1byte の文字を配列に格納．インデックス更新

受信したデータが'¥n'（改行）かどうか判断

⑤電文解析：次で説明

※この部分は、{}（括弧）が多いので対応に注意をしてください。

プログラム　◇電文の解析処理

```
switch(buf[0]){                  // 受信コマンド解析
  case '0':                      // ='0' LED 消灯
    digitalWrite(LED_PIN,LOW);
    Serial.print("LED OFF¥n");
    break;
  case '1':                      // ='1' LED 点灯
    digitalWrite(LED_PIN,HIGH);
    Serial.print("LED ON¥n");
    break;
 default:                        // '0','1'以外
    digitalWrite(LED_PIN,LOW);
    Serial.print("*** Unknown Command!!¥n");
    break;
}
```

① LED消灯＋メッセージ送信.

② LED点灯＋メッセージ送信.

③ LED消灯＋メッセージ送信.

図　80

上の部分は電文解析と対応処理の本体です。電文配列 buf[] 内の先頭の文字を判断して対応する処理を行います。

①先頭が'0'の場合：LED を消灯し【LED OFF】を送信

②先頭が'1'の場合：LED を点灯し【LED ON】を送信

③先頭が'0'の場合：LED を消灯し【*** Unknown Command!!】を送信

ソースコードを入力したら、名前を付けて保存しておきましょう。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

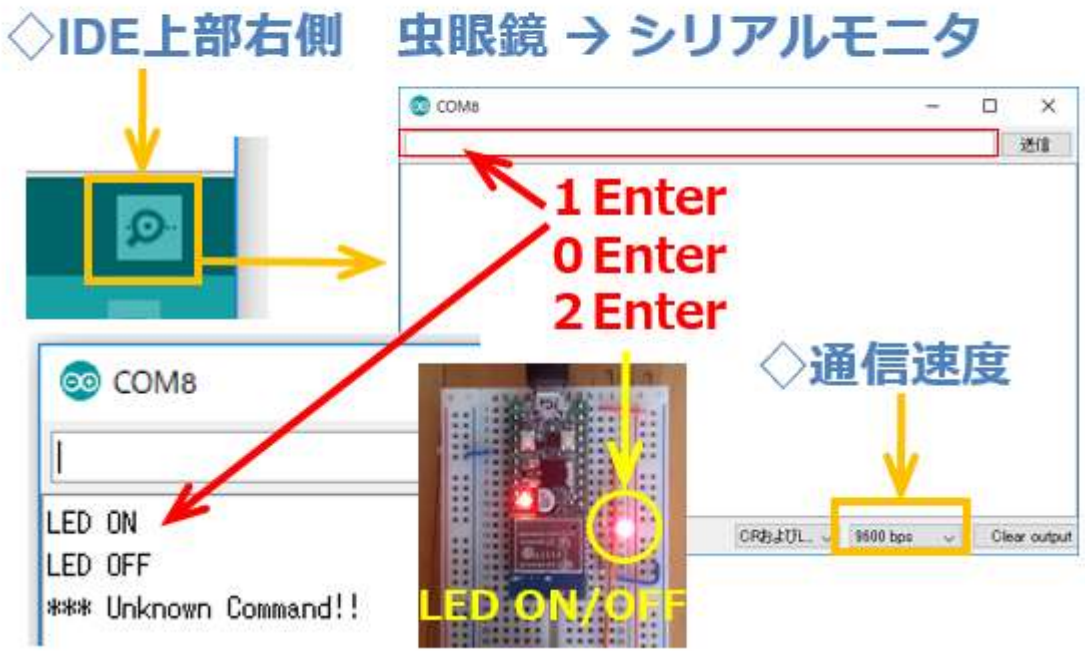


図　81

動作確認を行います。IDE 上部右側にある虫眼鏡マークのボタンをクリックしてシリアルモニターを起動しましょう。（上図右）
シリアルモニターの下部にある通信速度のセレクタで今回の通信速度 9600bps を選択します。メッセージは何も表示されません。

PC から WiFi マイコンに電文を送信してみましょう。シリアルモニター上部の COM ポート番号が表示されている直下に送信データを入力する部分があります。そこにカーソルを入れて、数字（半角）の'1'を入力し Enter キーを押下するか、右側にある送信ボタンをクリックしてください。この操作で 1 文字(='1')だけの電文が'¥n'(改行)付きで WiFi マイコンに送信されます。その時、LED が点灯してシリアルモニターには、LED ON のメッセージが表示されます。

次に送信データに'0'を入力して Enter キーを押下するか、送信ボタンをクリックして下さい。すると、LED が消灯して LED OFF のメッセージがシリアルモニターに表示されます。

今度は'2'を入力して Enter キーを押下するか、送信ボタンをクリックしてください。すると LED は消灯のままで、*** Unknown Meage!! と表示されます。

このシステムでは、'0', '1'以外の文字を送信すると全て Unknown になります。送信した側では、何を送ったかが分からないので、送信された文字を PC に送り返して、再送信を促すようにしてもよいですね。実際のマイコン・装置・設備・システムの間のシリアル通信では、そのように、現在の状況をお互いに送信しあいながら、連携をするような仕組みがプログラムされています。

いかがでしょうか、マイコンを利用したシステムを PC からリモートコントロールして、状況を返信させることができるようになりました。マイコンに複数のセンサーが接続されていて、その中から特定のセンサーの今の計測値を送信させるような使い方ができますね。PC 側では、受信したデータを CSV ファイルなどに記録したり、あるいはグラフ化したりして利用できます。どのような応用ができるか、考えてみてください。

第5回　VR(電圧測定)

なぜボリュームをVRと書くのでしょうか。ボリュームの正式名称は、可変抵抗器です。つまり抵抗の値が変えられる（可変）抵抗器ということでVariable Resistor、略してVRです。第5回目は、このVRで電圧を変化させて、それを計測してみましょう。

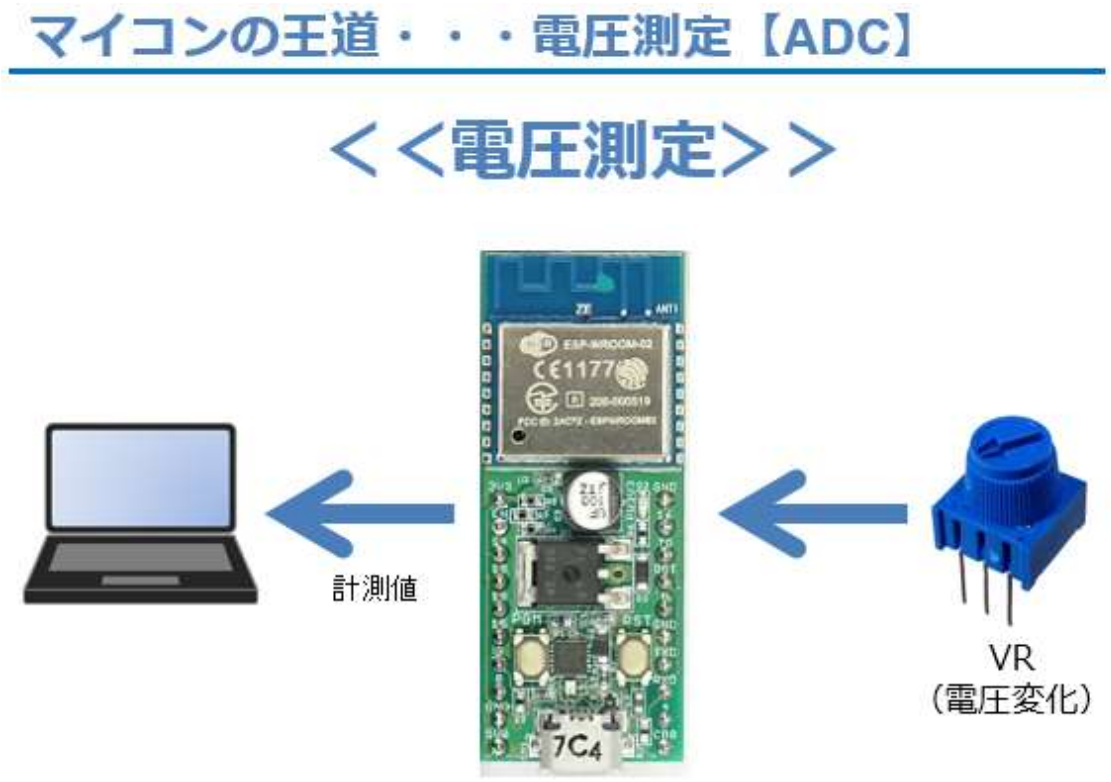


図　82

VRを使った電圧変化を測定してPCに送信します。第3回シリアル通信【送信】の最後で、送信するメッセージの内容を計測値で編集すれば、マイコンが計測した情報を監視する用途にシリアル通信が使えると解説しました。まさにその実証です。

◇全体構成とパーツ

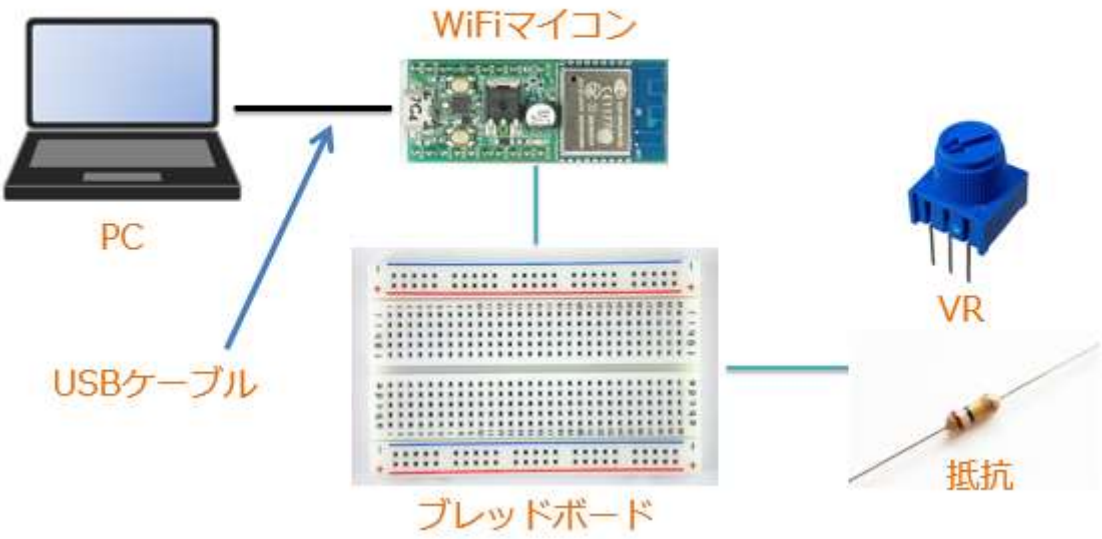


図 83

システムの全体構成を上図に示します。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. VR（可変抵抗器：50KΩ）×1 個
7. 抵抗器（100KΩ）×1 個

ここで、VR と固定抵抗器について解説します。

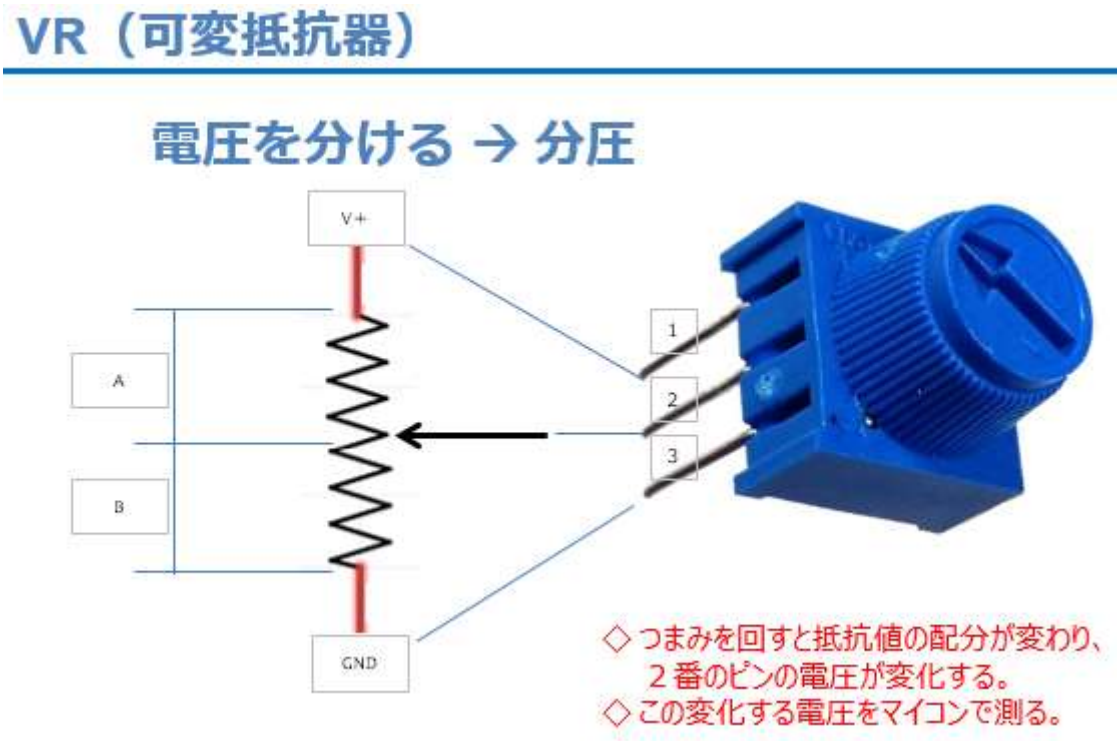
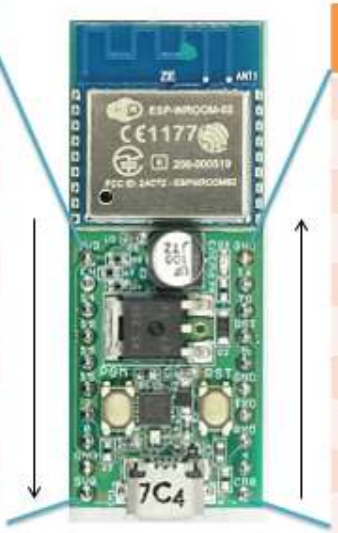


図 84

上図は、今回使用する VR の写真と内部構造を表しています。VR にはつまみ（矢印が刻印されていて周囲がギザギザになっている部分）が付いています。脚ピンが 3 本あり、便宜的に写真上から 1,2,3 番とします。1 番と 2 番ピンの間は、50KΩの抵抗値となっています。つまみは内部の接点（図左側の矢印部分）と連動しています。つまみを右左に回すと、図の矢印部分が上下に移動する仕組みです。ここで、1 番ピンを V+（電源の+側）に、3 番ピンを GND に接続して、つまみをいっぱいに回して、図の矢印が V+に一番近くなった時、2 番ピンと 3 番ピンの間には、V+の電圧がかかっていることになります。反対に矢印部分が一番下がったときは、矢印部分は GND と接続していることになり、2 番ピンと 3 番ピンの間の電圧は 0V になります。このように、VR は、2

番ピンの矢印の位置(つまみの回し具合)によって、図の A と B の部分に電圧を分けてくれる役割を果たします。このことを分圧と云います。この分圧で得られる電圧値は、GND から V+の間の大きさになります。VR の 2 番ピンから出力される電圧を計測するのが今回の目的です。

番号	名称	機能
1	3V3	3.3V(In/Out)
2	EN	イネーブル端子
3	14	GPIO14
4	12	GPIO12
5	13	GPIO13
6	15	GPIO15
7	2	GPIO2
8	0	GPIO0
9	GND	GND
10	5V0	5V In



番号	名称	機能
20	GND	GND
19	16	GPIO16
18	TO	ADC（max1V）
17	RST	リセット
16	5	GPIO5
15	GND	GND
14	TXD	TXD
13	RXD	RXD
12	4	GPIO4
11	CB0	通信モニタ用

図 85

ここで、上図を見てください。この図は第 1 回で説明した WiFi マイコンモジュールのピン配置です。上図の 18 番ピン TO には ADC(max1V)と記載があります。VR の 2 番ピンと接続します。ADC とは Analog to Digital Converter のことで、アナログ値である電圧をデジタル値に変換する便利な機能です。電圧計測の手段は ADC の機能そのものです。

ところで、max1V と記載があります。これは、1V までしか測れないことを意味していますので、VR からの出力が 1V になるように調整しなければいけませんので、少し工夫が必要です。

ADCへの対応

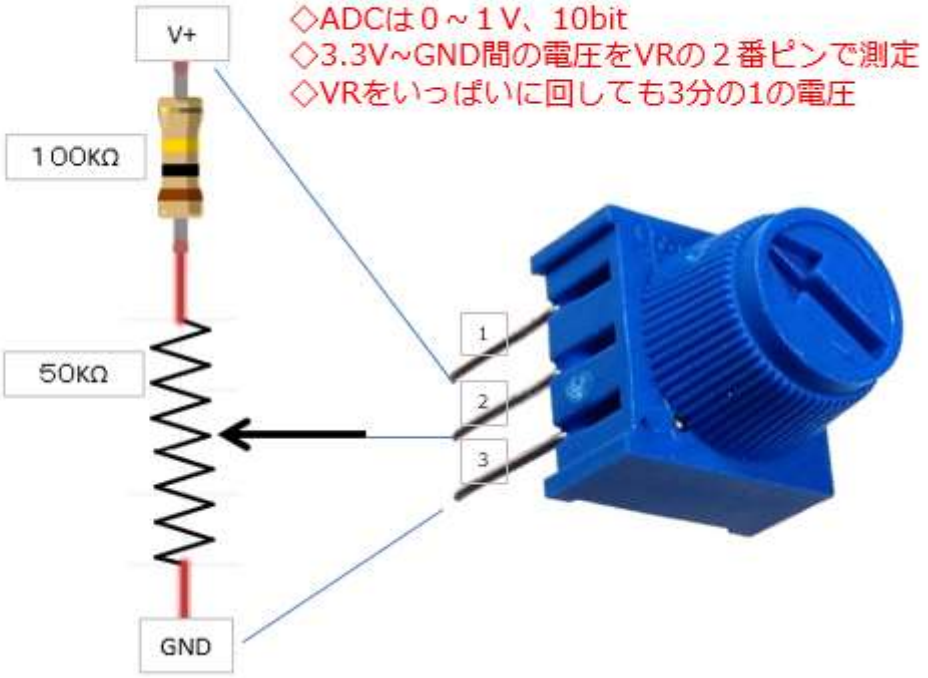


図　86

今回は、上の図に示すように、50KΩの VR に 100KΩの抵抗を直列に接続して、VR をいっぱいに回しても 2 番ピンの電圧が V+の 1/3 になるようにします。V+に 3.3V を加えると 2 番ピンは最大で 1.1V の電圧になります。これは 1V よりも少し大きいのですが、本格的な計測で精度や信頼性に重点を置く場合は、この部分を丁寧に設計します。

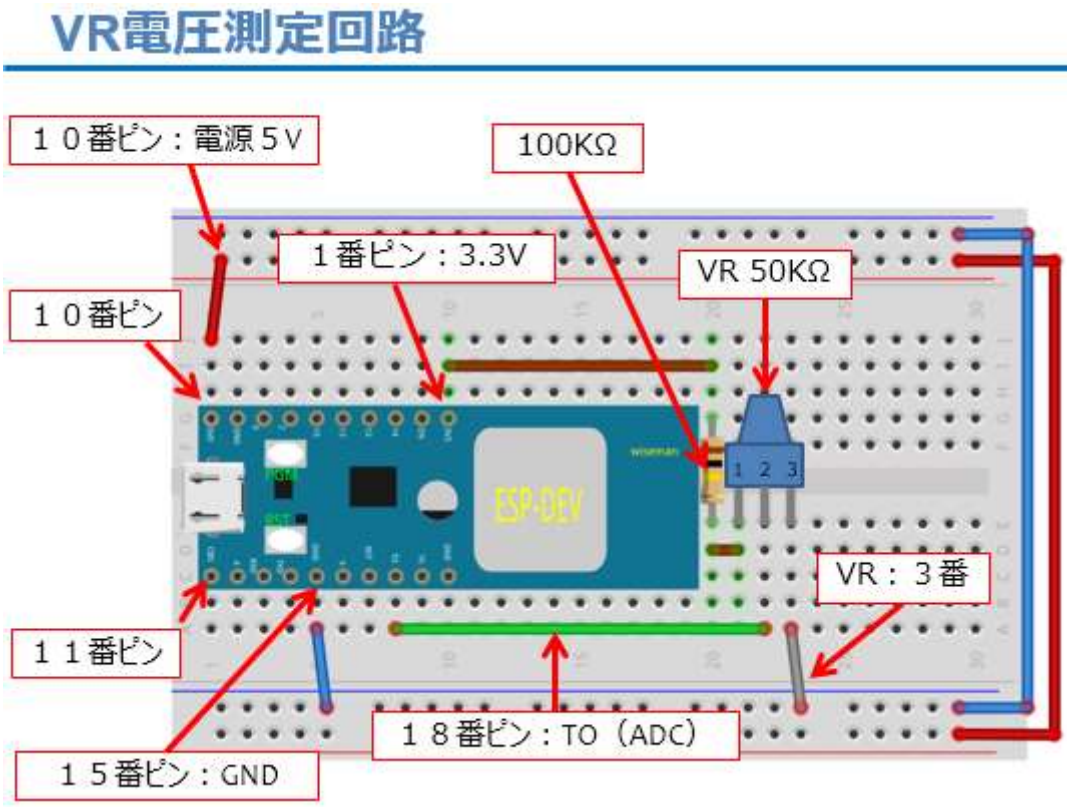


図　87

図に従い、配線をします。実際に配線したものが下の写真です。

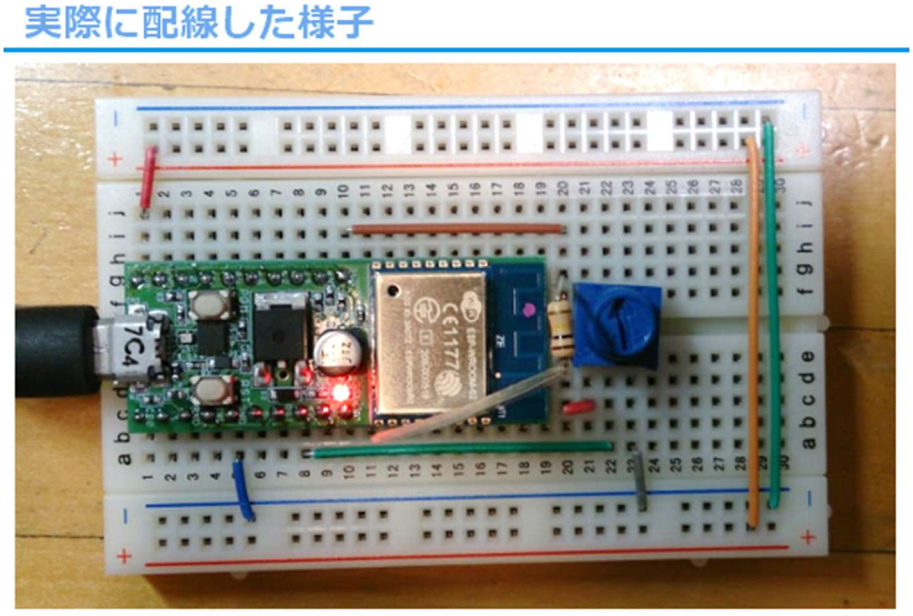


図　88

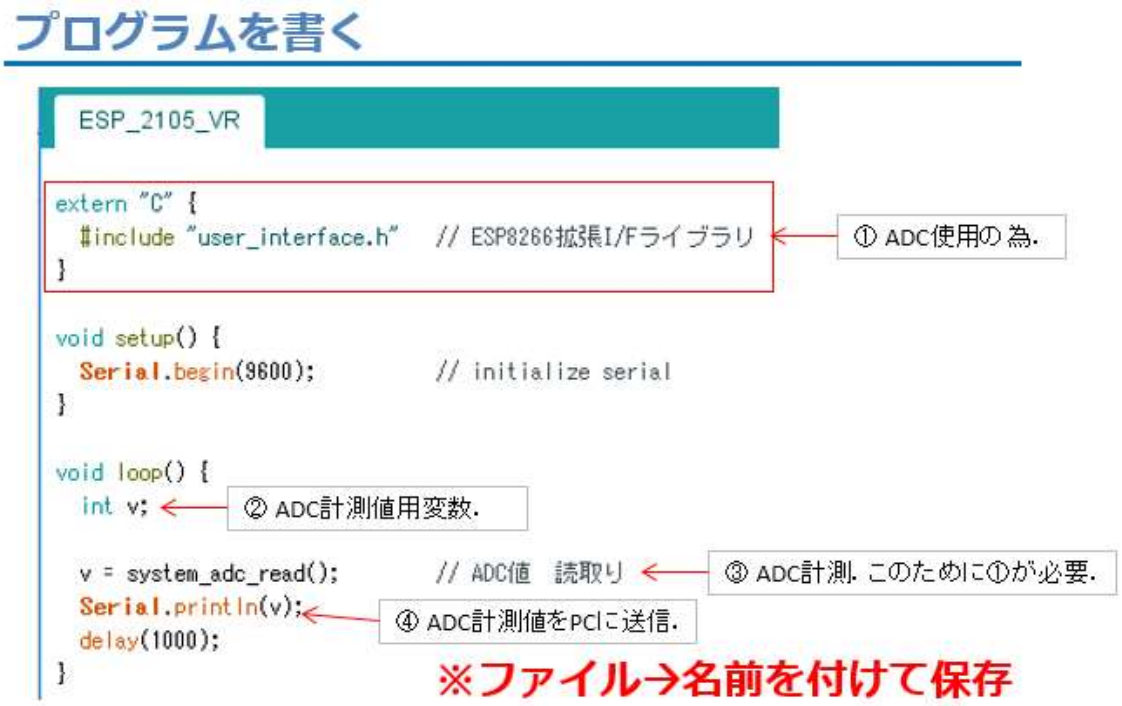


図 89

IDE を開き、新しいスケッチとして上図のソースコードを入力します。

①ADC 利用のためヘッダ取り込み.

②ADC により電圧の直読値を格納する変数

③実際に A/D 変換を行い、計測巣する関数

④計測した値をそのままシリアル通信で送信（改行付き）

このプログラムの動作としては、loop()の最後にあるように、1 秒間隔で電圧計測を行い、その値を PC にシリアル通信で送信するという動作を繰り返します。

ソースコードを入力したら、名前を付けて保存しておきましょう。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

図　90

動作確認をしましょう。IDE の虫眼鏡マークのボタンをクリックしてシリアルモニターを起動します。通信速度を Serial.begin()で設定した速度に合わせます。すると、約 1 秒ごとに数値が表示され続けます。

これが VR によって分圧された電圧の A/D 変換値です。VR を回転してみると、表示される値が変化することが分かります。

ここで表示されるのは数値だけですので、【電圧であるという実感】が湧きませんね。そこで次の測定を行いました。

◇TO端子電圧とAD値の関係（実測）

V	AD
0.000	9
0.085	89
0.157	159
0.286	278
0.400	387
0.500	482
0.599	579
0.701	670
0.802	773
0.905	872
0.994	950

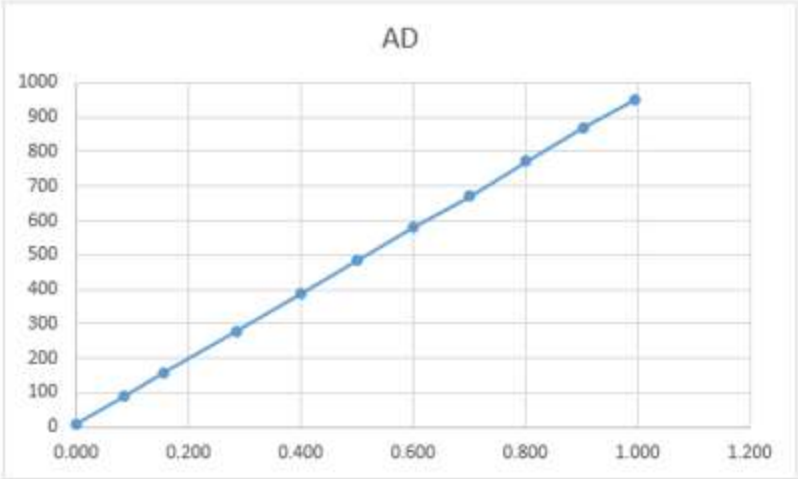


図 91

上図は、VR の 2 番ピンと GND 間の電圧をテスターで測りながら、その時シリアルモニターに表示される値を記録した表(図左)です。
その値をグラフにしたものが図右です。当然ですが直線になっていて【A/D 変換値とテスターで測った電圧が比例関係】にあることが分かります。

電圧への換算

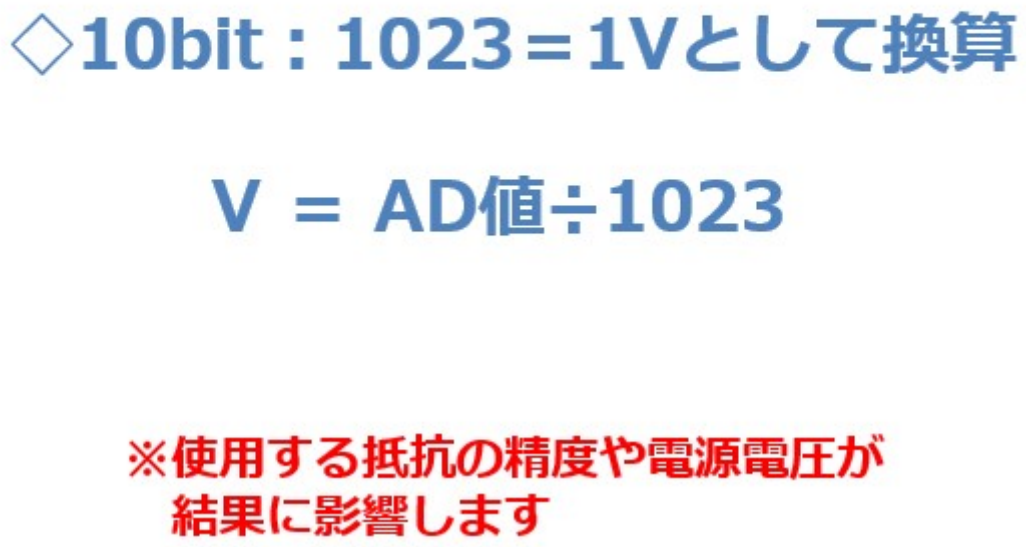


図 92

【重要】

このA/D変換器は10bit仕様です。ADC直読値から電圧値を求めるためには、上図のような変換が必要になります。今回のシステムは、ADC直読値の表示だけでしたが、計算を行って電圧を表示するようにプログラムを改善すると電圧計になります。電圧が計測できるようになると、様々なセンサーを利用することができるようになります。

次の講座では、そのことを実証します。

第6回　温度センサー(アナログ)

センサー活用はIoT必須の要素です。温度センサーを利用した環境計測では【温度を測ること】は【分圧した電圧を測ること】と同じことです。電圧測定ではA/D変換値をそのままPCに送信していましたが、今回はA/D変換値を電圧に変換し、さらに温度にして通知します。

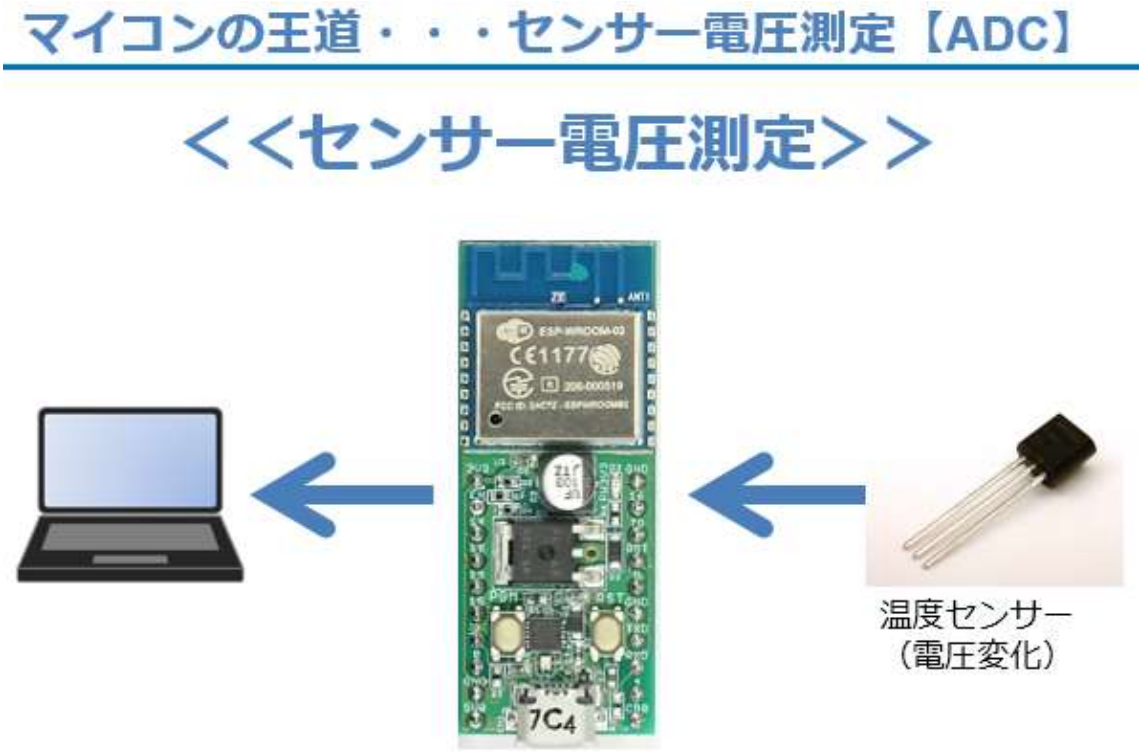


図　93

上の図を見れば明らかなように、システムの外観としてはVRと温度センサーを取り換えたシステムを作ります。

◇全体構成とパーツ

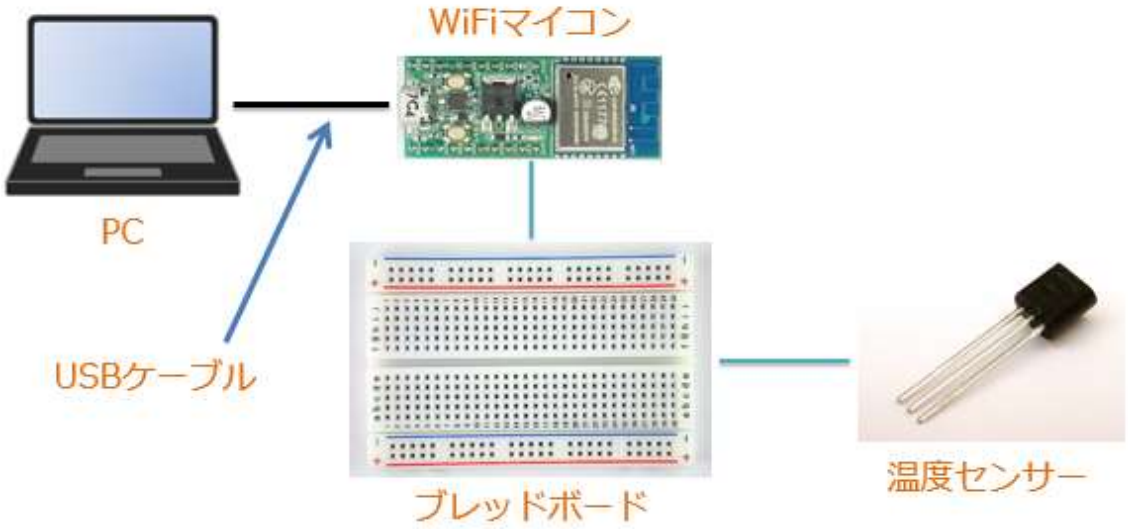


図 94

システムの全体構成を上図に示します。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. 温度センサー（LM61CIZ）×1 個

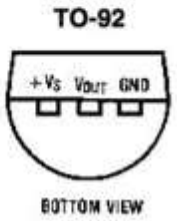
温度センサー

◇LM61CIZ　リニアな特性

◇測定範囲：－30℃～100℃
－30℃＝300mV
～
0℃＝600mV
～
100℃＝1600mV

◇温度係数：＋10mV／℃

◇動作電圧範囲：＋2．7～＋10V



温度センサー（LM61CIZ）

図　95

使用する温度センサーの Data Sheet を抜萃したものが上図です。測定範囲は-30℃から 100℃と広範囲ですが、今回の実験では気温程度が測定できれば良いので十分すぎる測定範囲です。温度係数が+10mV/℃で温度と出力電圧の間に比例関係がありますので、センサー出力電圧から温度を求めるには、次のような計算を行えば求まります。

温度＝（センサー出力電圧－600mV）÷１０mV

図　96

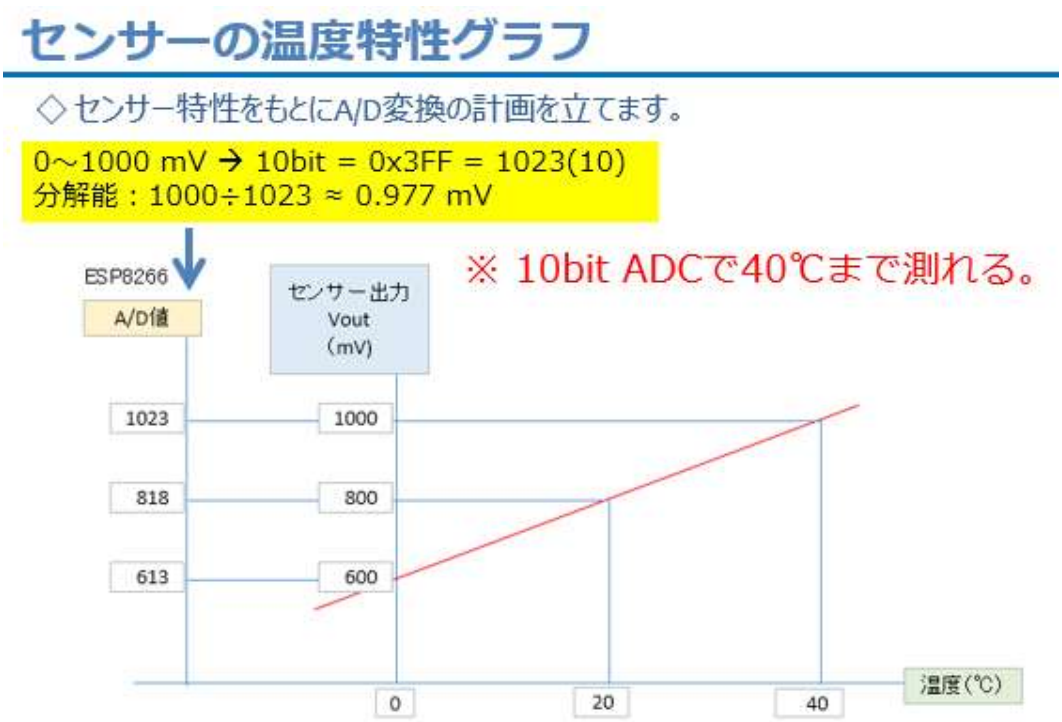


図 97

センサーの温度特性をグラフにすると上図のようになります。センサーの計測範囲はもっと広いのですが、今回の用途は気温計測なので、高くても 40℃です。室内で実験しますので氷点下にはなりませんので、上図の範囲で十分です。この時のセンサー出力電圧は、600~1000mV で、WiFi マイコンの ADC の仕様にちょうどマッチします。ADC の仕様から 1bit 当たりの分解能は、およそ 0.977mV です。これを基にして A/D 変換値から温度を求めるには次の計算を行います。

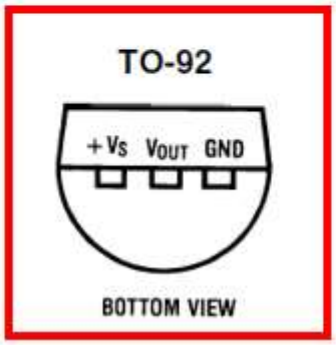
（AD値×分解能 － 600mV）÷10 ＝ 温度（℃）
☆1　☆2

☆1　0.977mV
☆2　0℃のときの出力電圧

図　98

データシート

ピン配置で配線が分かる



+Vs：3.3V
Vout：出力
GND：0V

図　99

Data Sheetには、上図の様にピン配置が描かれています。BOTTOM VIEWと記載のあることで分かるように、センサーを脚ピン側から見た図です。蒲鉾を逆さにしたような形の平らな部分に型番などが印刷されています。図の左から順に

+Vs：電源--->3.3V駆動が可能です。

Vout：出力--->WiFiマイコンのTO（18番ピン）に接続します。

GND：0V（GND）

このピン配置と前回使用したVRとを比較したものが次の図です。

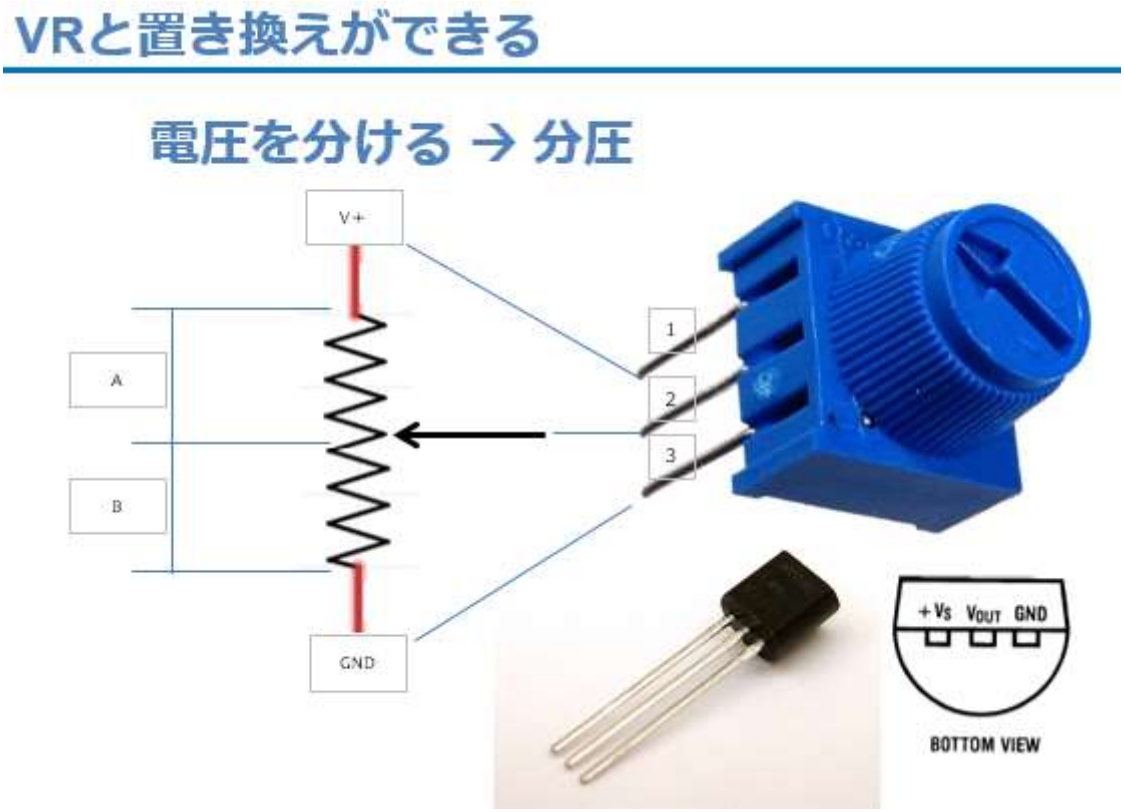


図　100

この温度センサーは、上図に示すように、前回使用した VR とピン配置が同じなので、差し替えが可能です。（向きに注意）そして計測対象の温度範囲では、出力電圧が 1V に収まるので、前回使用した 100KΩ の固定抵抗も必要ありません。直接 WiFi マイコンに接続できます。

センサー電圧測定回路

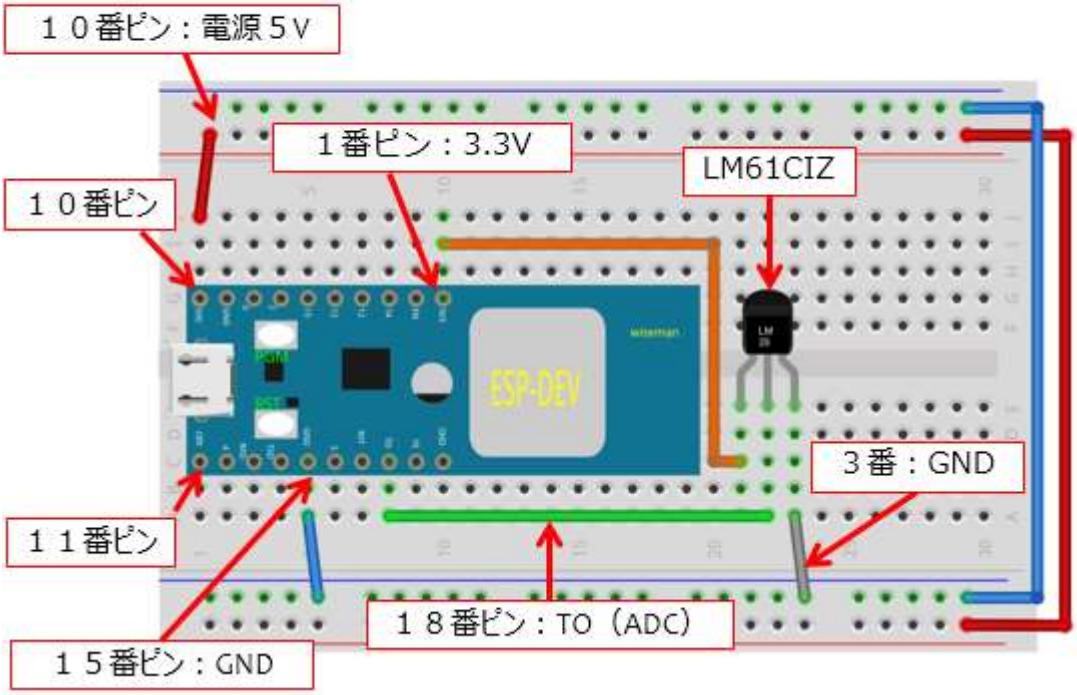


図　101

図に従い配線をします。

実際に配線した様子

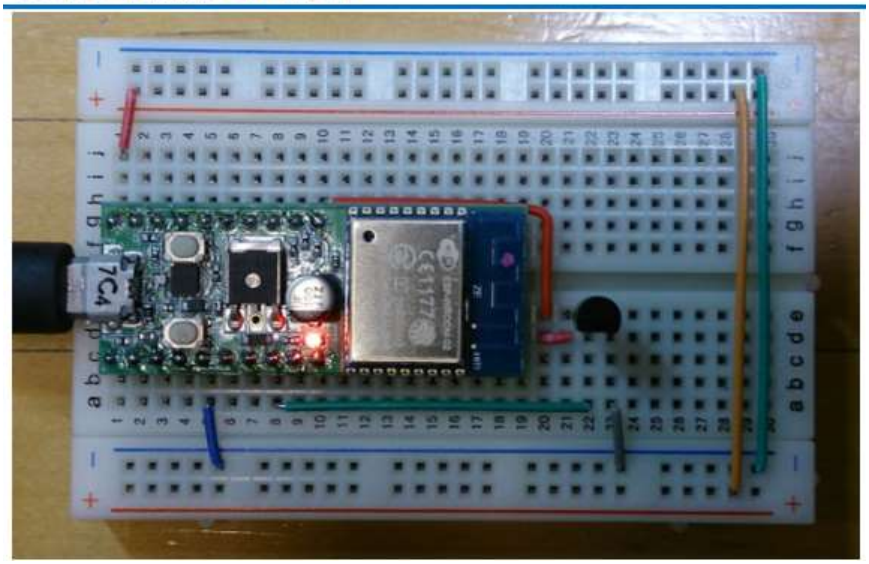


図　102

プログラムを書く

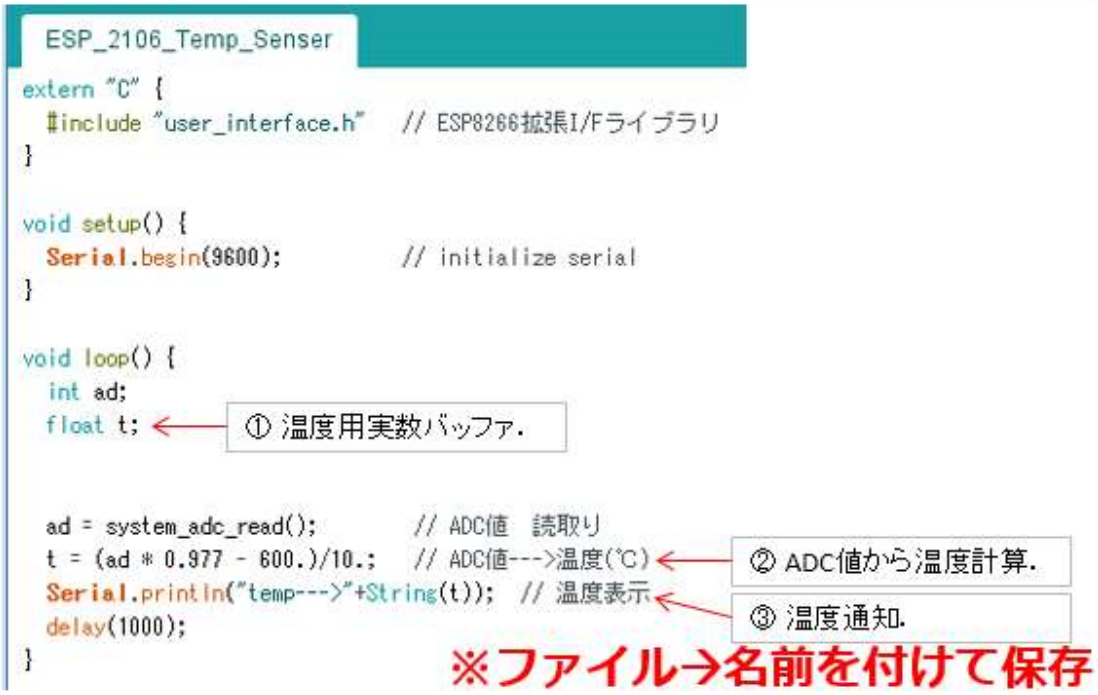


図 103

IDE を使い、ソースコードを入力します。

特に解説する部分は以下の通りです。

①温度は小数点以下まで求めているので実数型変数とする

②温度換算計算式は前に示した

③温度を文字列に変換して通知する

ソースコードを入力したら、名前を付けて保存してください。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

図　104

IDE からシリアルモニターを起動して、通信速度を合わせます。モニターに温度が 1 秒間隔で表示されています。今回使用した温度センサーは精度があまり高くありません。ですから、実際に使用する場合は何度か計測して平均値表示したり、小数部の桁数を考慮したりするなどの工夫をすると良いでしょう。プログラムを改善してみてください。ここまで進んだ皆さんなら、容易にできるはずです。

次回のテーマも温度計測ですが、使用する温度センサーの使い方が全く異なる、デジタル温度センサーを使います。

第7回　温度センサー(デジタル)

前回使用した温度センサーは出力が電圧でした。出力される温度に対応した電圧はアナログ値なのでアナログセンサーと呼ばれたりもします。温度センサーの出力電圧を計測出来る A/D 変換器は、WiFi マイコンでは、18 番ピンだけなので、1 個のセンサーしか使えません。それでは、マイコン 1 個で 1 計測となってしまいます。

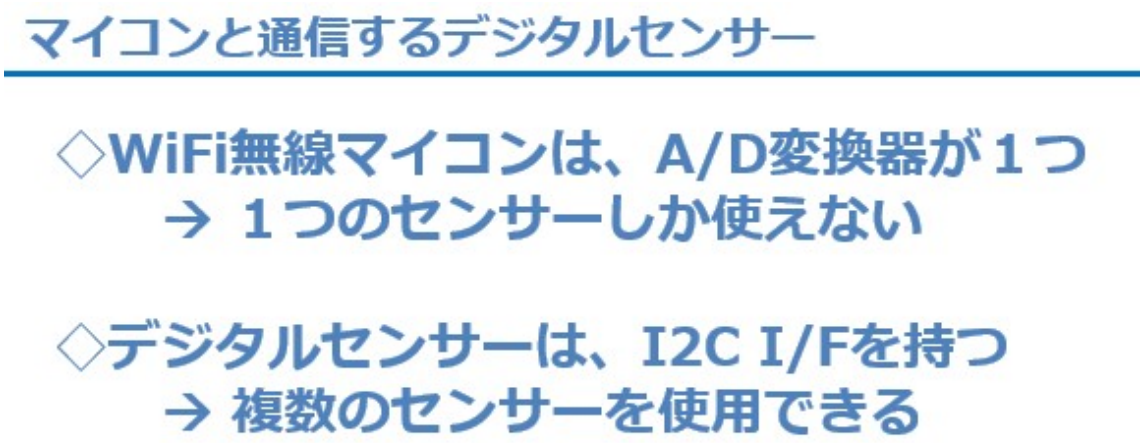


図　105

今回使用するデジタルセンサーは、I2C I/F を持つ温度センサーです。I2C は、日本ではアイ・ツー・シーと呼ぶことが多いのですが、正式には Inter-Integrated Circuit の略で、I-squared-C（アイ・スクエアド・シー）のことです。I2C I/F は、フィリップス（オランダ）という会社が開発したものです。シリアルデータ（SDA）とシリアルクロック（SCL）という 2 本の信号線に、複数（同じ種類でも異なる種類でも可）のデバイスを接続できる、シリアル通信の仲間です。これを利用すれば、1 つのマイコンに多数のセンサーが使えます。

マイコンの王道・・・デジタル温度センサー

＜＜温度直読＞＞

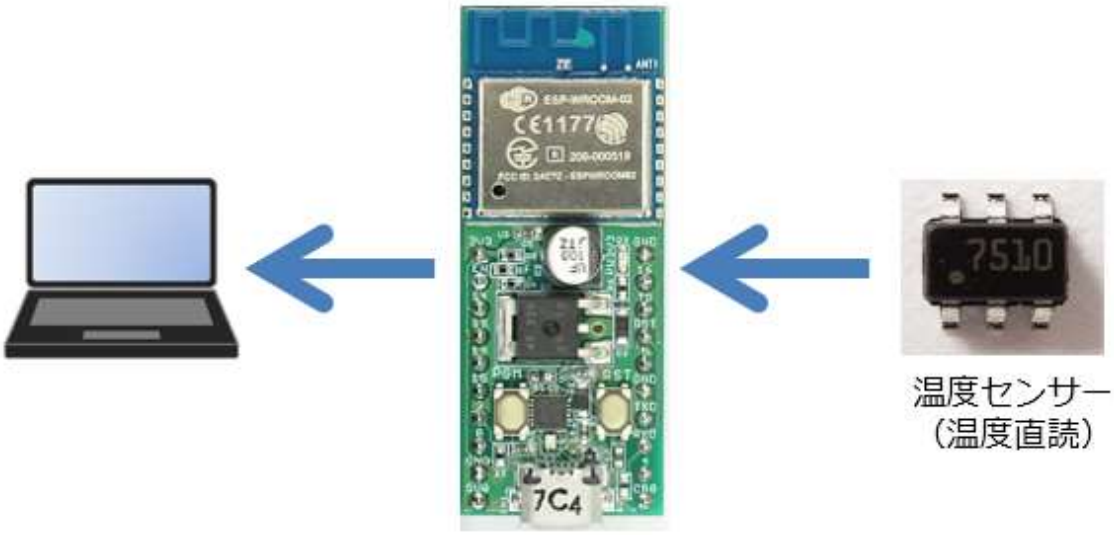


図　106

このシステムで利用するデジタル温度センサーは、図右の様に、足が 6 本あります。アナログ温度センサーと違い、直接温度を読むことができるので、電圧から温度への換算などは行わなくてよいという利点があります。このデジタル温度センサーから温度を直接読み込んで、PC に通知するものです。大がかりなデジタル温度計と考えて良いと思います。

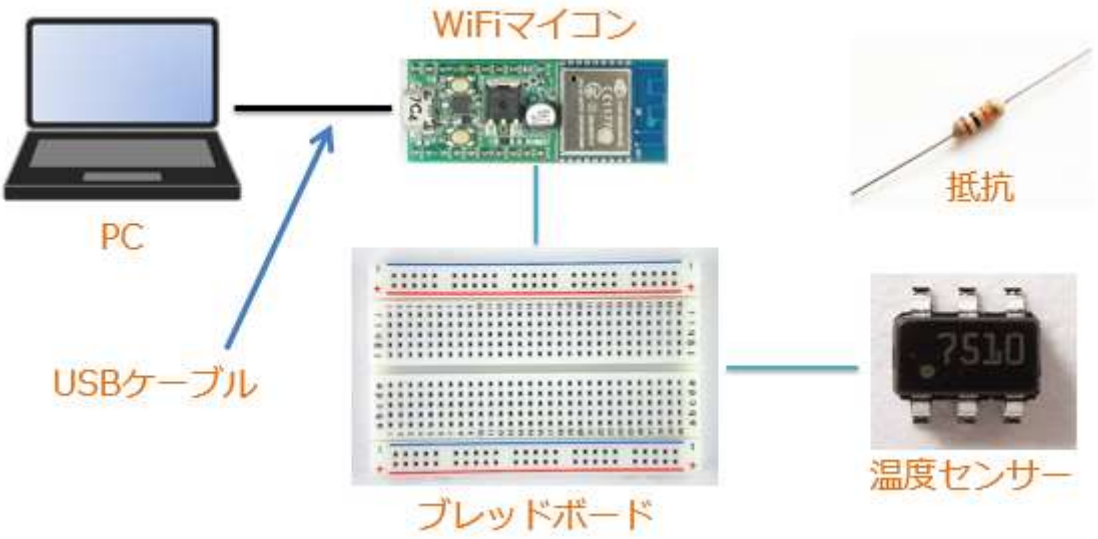


図 107

システムの全体構成を上図に示します。必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. デジタル温度センサー（STTS751）×1 個
7. 抵抗（10KΩ）×2 個（I2C I/F の Pull Up に使用）

以下、デジタルセンサー：STTS751 を少し詳しく説明します。

デジタル温度センサー　STTS751

STTS751

2.25 V low-voltage local digital temperature sensor

Features

- Operating voltage 2.25 V to 3.6 V
- Operating temperature –40 °C to +125 °C
- Programmable
  - 10 different conversion rates
    0.0625 to 32 conversions/sec.
    1 conversion/sec. - default
  - 4 different resolutions
    9-bit (0.5 °C/LSB) to12-bit (0.0625 °C/LSB)
    10-bit (0.25 °C/LSB) - default
- Low supply current
  - 50 µA (typ) for 8 conversions/sec.
  - 20 µA (typ) for 1 conversion/sec.
  - 3 µA (typ) standby
- Accuracy
  - ±1.0 °C (typ) 0 °C to +85 °C
  - ±2.0 °C (typ) –40 °C to +125 °C
- One-shot mode for power saving
- Fast conversion time 21 ms (typ) 10-bit
- Pull-up resistor value allows single pin to select one of four slave addresses
- Supports 400 kHz serial clock

UDFN-6L

SOT23-6L

- SMBus 2.0 compatible
  - SMBus ALERT (ARA) support
  - SMBus timeout
- RoHS/green

Applications

- Solid state drives
- Portable electronics
- Notebook computers
- Smart batteries
- Servers
- Telecom

図　108

上図は、今回使用するデジタル温度センサー（STTS751）の Data Sheet です。駆動電圧は 2.25～3.6V で WiFi マイコンの 3.3V を利用することができます。また計測範囲は-40℃～+125℃と広範囲です。プログラムによって 4 種類の分解能が選択できます。また、1 秒間に 8 回の計測で流れる電流は 50μA と大変小さなものとなっています。

デジタル温度センサー　STTS751

Figure 2.　Pinout - SOT23-6L and UDFN-6L

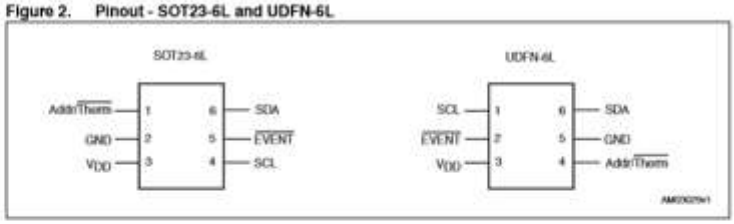


Table 3.　Pin descriptions

Pin		Name	Description
SOT23-6L	UDFN-6L		
1	4	Addr/$\overline{\text{Therm}}$	Open-drain output that can be used to turn on/off a fan or throttle a CPU clock in the event of an overtemperature condition. The pin at power-up determines the SMBus slave address according to the pull-up resistor value as shown in *Table 1*. This pin must have a pull-up resistor connected to the same voltage as V_{DD} or tied to GND (pin cannot float). Total capacitance on this pin must be <100 pF. Note: By tying Addr/$\overline{\text{Therm}}$ to ground, the device functions as one address device only. The $\overline{\text{Therm}}$ functionality is then not available. The address for device STTS751-0 is 72h and the address for device STTS751-1 is 76h.
2	5	GND	GND
3	3	V_{DD}	Power supply V_{DD}
4	1	SCL	SMBus clock
5	2	$\overline{\text{EVENT}}$	Open-drain interrupt output. Output supports the SMBus Alert (ARA). Note: This pin may not float.
6	6	SDA	SMBus data input/output

マイコン
16Pin
12Pin

図　109

今回使用するセンサーのパッケージは図上左のものです。センサーの 4 番ピン（SCL：I2C のクロック）と 6 番ピン（SDA：I2C のデータ）をそれぞれ WiFi マイコンの 16 番ピン、12 番ピンに接続します。

Figure 4.　Application hardware hookup

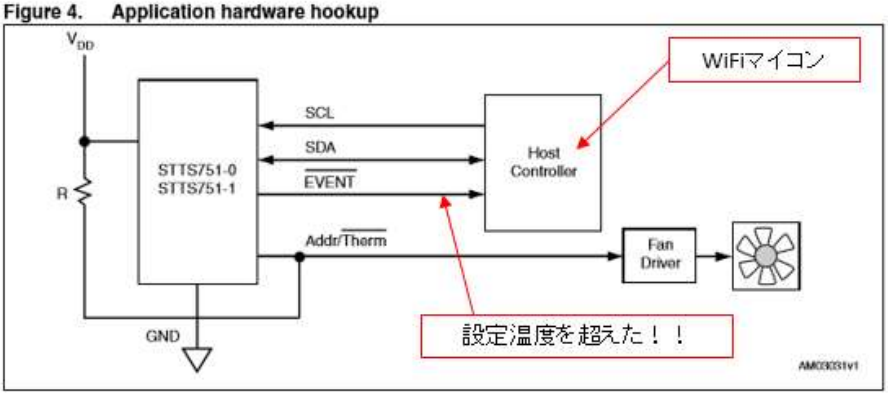


図　110

設定温度に対する判断の信号(EVENT)も準備されています。（上図）

4　STTS751 register summary

The STTS751 uses 8-bit registers. Variables longer than 8 bits are managed in byte pairs. For example, when reading a 10-bit temperature value (10 bits is the default resolution.) the application must read two registers and then concatenate the upper byte with the 2 most significant bits of the lower byte.

Table 9 below summarizes the register map for the device. Accessing any invalid address results in indeterminate data.

Table 9.　Registers/pointers

Address pointers (h)	STTS751 register map			Power-up default values binary (dec)
	Device registers name	Size	Type	
00	Temperature value high byte	8	R	undefined
01	Status	8	R	undefined
02	Temperature value low byte	8	R	undefined
03	Configuration	8	R/W	0000 0000
04	Conversion rate	8	R/W	0000 0100

図　111

このセンサーはレジスタが 5 つありますが、そのうち次の 3 つを使います。

00 番：温度の整数部を取り出すレジスタです。

02 番：温度の小数部を取り出すレジスタです。

03 番：設定用レジスタ。初期化で使用します。

4.2 Temperature register format

The temperature data is a 12-bit number and is stored in two's complement format spanning the high byte and low byte registers as shown in *Table 11*.

Table 11. Temperature register (two's complement)

ADDR (hex)	R/W	Register	b7	b6	b5	b4	b3	b2	b1	b0	Power-up default (hex)
00	R	Temperature - high byte	sign	64 °C	32 °C	16 °C	8 °C	4°C	2 °C	1 °C	00
02	R	Temperature - low byte	½ °C	¼ °C	$^{1}/_{8}$ °C	$^{1}/_{16}$ °C	0	0	0	0	00

The integer portion of the temperature is stored in the high byte, and the fractional portion in the low byte. The lower four bits of the low byte will always read 0. At power-up, the STTS751 defaults to 10-bit resolution. Thus, bits b5 and b4 of the lower byte will also read 0 until the device is configured to a higher resolution (via the Tres bits in the configuration register).

図　112

00 番レジスタは、1bit あたり 1℃の温度を通知してくれますので、読んだ値をそのまま使用できるのですが、最上位の b7 が sign bit となっていて、two's complement（2 の補数）ですから、氷点下の場合は、そのことに配慮が必要ですが、今回の計測対対象は室温ですので、直読値をそのまま利用できます。

02 番レジスタは、小数部の温度を通知してくれますが、下位 4bit が 0 となっているので、読み込んだのち右に 4bit シフトして、LSB（Least Significant Bit：最下位 bit または分解能を意味する）の 1/16℃を掛け算して、小数点以下の温度を求める処理が必要です。

4.6 Configuration register

The STTS751 configuration register is read/write and controls the functionality of temperature measurements. It is located at address 03h. The configuration register bits function as described below.

Table 15. Configuration register

ADDR (hex)	R/W	Register	b7	b6	b5	b4	b3	b2	b1	b0	Power-up default (hex)
03	R/W	Configuration	MASK1	RUN/STOP	0	RFU	Tres1	Tres0	RFU	RFU	00
			1	0	0	0	1	1	0	0	← 今回の初期設定値

Description

MASK1: [bit 7]

0: EVENT is enabled. Any out-of-limit condition asserts the EVENT pin (active low).

1: EVENT is disabled.

RUN/STOP: [bit 6]

0: Device is running in continuous conversion mode.

1: Device is in standby mode drawing minimum power.

The RUN/STOP bit controls temperature conversions by the ADC. When this bit is 0, the ADC converts temperatures in continuous mode, at a rate as selected by the Conversion Rate register (*Section 4.7*). When the RUN/STOP bit is 1, the ADC will be in standby mode, thus reducing current supply significantly.

Tres1:Tres0 [bits 3 and 2]

These bits select one of the four programmable resolutions for temperature data on the STTS751 providing resolutions down to 0.0625 °C/LSB. The default resolution is 10 bits, 0.25 °C/LSB.

図　113

Configuration レジスタは、センサーの使い方を指示するものです。図に今回の設定値を書いておきました。1 にする bit を説明します。

b7：MASK1 は、1 にすると、EVENT 信号（指定した温度を超えたかどうかを判断する信号）を使わない設定です。今回は使用しません。（※実は配線しなければ 0 でも 1 でもどちらでも良いのですが。）

b3,b2：Tres1:0 となっています。この bit は、4 つのプログラムで設定可能な分解能の内 1 つを選ぶと書いてあります。この bit を 11 とするとどうなるかは次の資料を見てください。

Table 16.　Conversion resolution

Tres1:Tres0	Temperature resolution	LSB step size (°C)
00	10 bits (default)	0.25
01	11 bits	0.125
11	12 bits	0.0625
10	9 bits	0.5

Table 11.　Temperature register (two's complement)

ADDR (hex)	R/W	Register	b7	b6	b5	b4	b3	b2	b1	b0	Power-up default (hex)
00	R	Temperature - high byte	sign	64 °C	32 °C	16 °C	8 °C	4°C	2 °C	1 °C	00
02	R	Temperature - low byte	½ °C	¼ °C	1/8 °C	1/16 °C	0	0	0	0	00

図　114

Tres1:0 を 11 にセットすると、図の下に示すように 12bit の分解能となります。（Tres は、Temperature Resolution の短縮形らしい）

これらにより温度処理は次の様になります。

◇12bit で温度測定
◇温度は、整数部と小数部に分かれている
　→整数部と小数部は別に処理
◇小数部は、b7~b4 の4bit
◇小数部は、 1 LSB＝1/16℃（0.0625℃）
　→読取り後、4bit右シフトして×0.0625する

① 11110000　→　00001111
② 00001111×0.0625＝℃（小数部）

図　115

【重要】I2C I/F では、多くのデバイスが同じ信号線につながるので、通信するデバイスを識別するために、【スレーブアドレス】が決められます。このアドレスをプログラムで使用します。

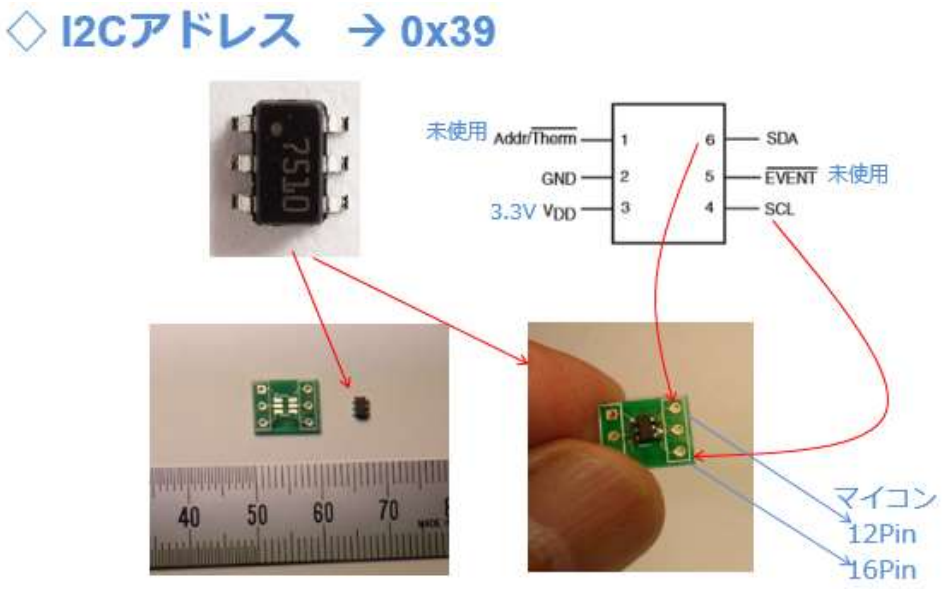


図　116

実際のデジタル温度センサー（STTS751）は図のような小さなものです。単体では 2×1mm 程度の大きさで、そこに 6 本の端子が出ていますが、そのままではブレッドボードでは使えませんので、図右下のように小さな基板に半田付けしたものを使用します。よく見るとセンサーの背面左上に小さな丸印があり、基板の左上にも白い丸印があるので、これを合わせて使用します。回路を作成する際も、この小さな丸印を基準にピン配置を考えながら配線します。

【重要】次に示す配線図では、I2C I/F の SCL と SDA の信号は、10KΩの抵抗で 3.3V に Pull Up する部分があります。見落とさないでください。Pull Up というのは、第 2 回で解説した Pull Down の反対です。今回は常時 High レベルにしておきたい信号に使用しています。

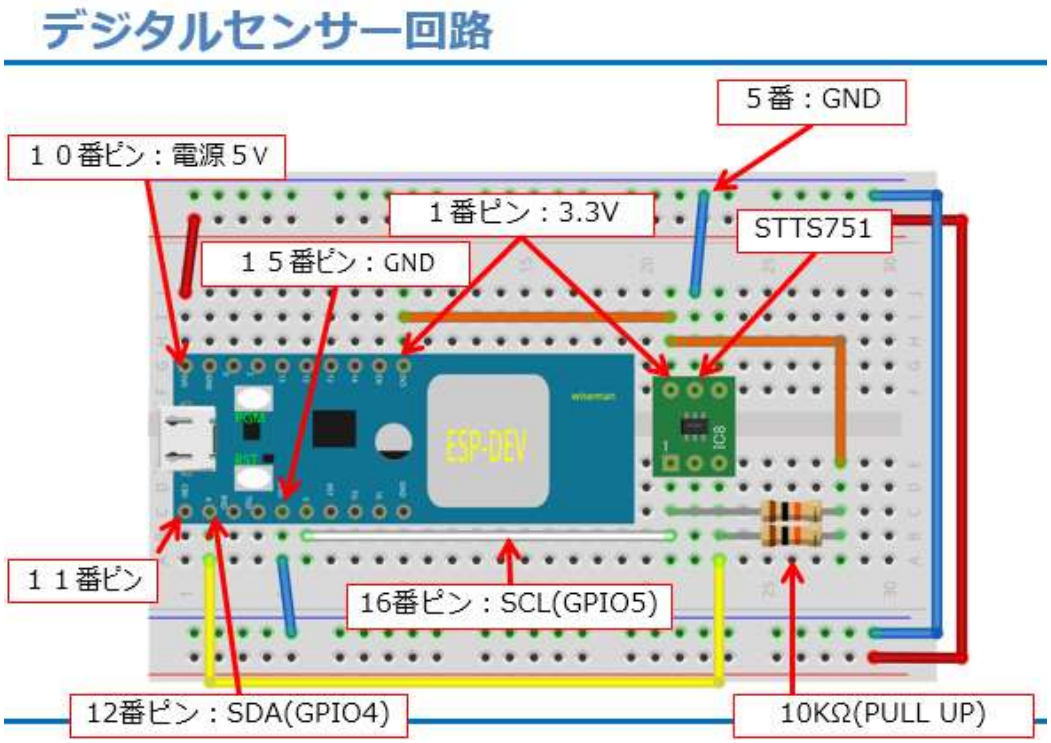


図　117

上図を見て回路を作成してください。完成したものが下図です。

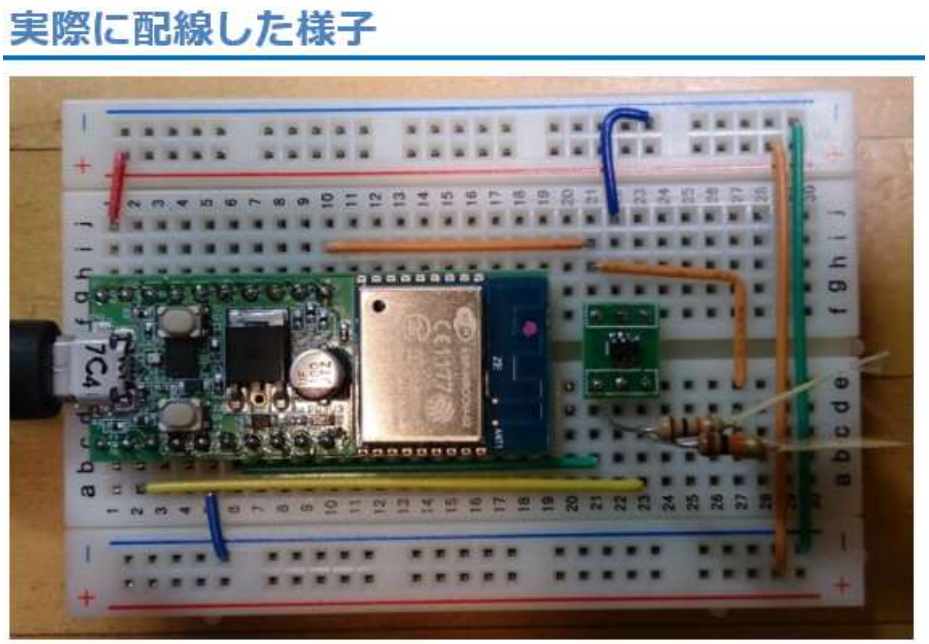


図　118

プログラムを書く

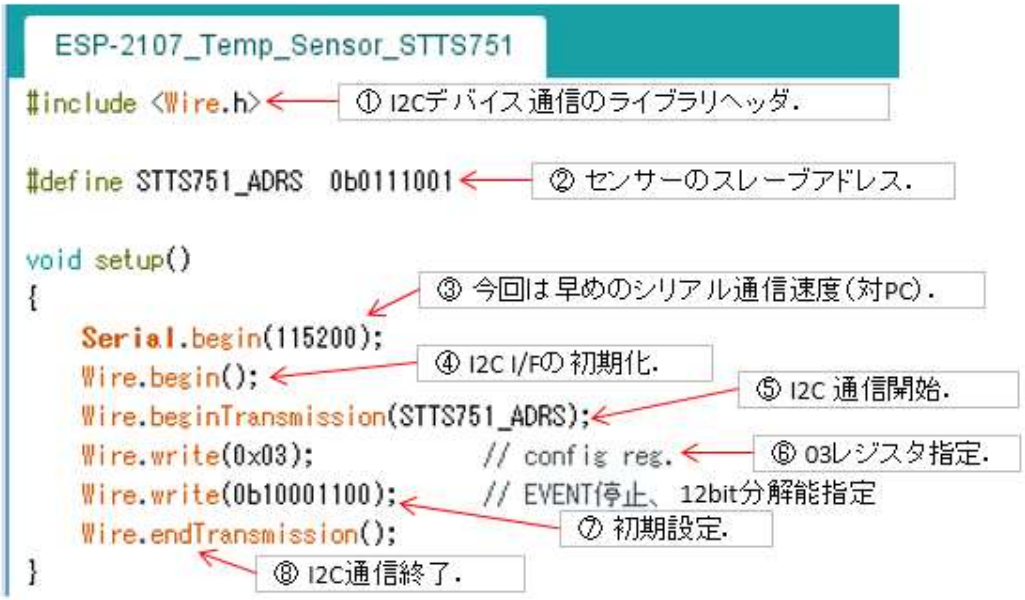


図 119

IDE にプログラムを入力して下さい。

①I2C でバイスとの通信に必要なライブラリを利用するためのヘッダ

②使用するセンサーの I2C スレーブアドレス（0x39）

③シリアル通信初期化(速度 115200bps)

④I2C I/F 初期化

⑤センサー初期化のため I2C 通信開始

⑥03 レジスタ設定

⑦初期設定

⑧I2C 通信終了

【重要】

※上の様に I2C デバイスと通信を行う際は、初めに I2C スレーブアドレスを指定して通信を開始します。

プログラムを書く　loop()　前半

図　120

loop()での処理は 2 つに分けて説明します。

①温度を整数部と小数部別々に読み込むための変数

②小数部を PC に通知する際、1 桁ごとに処理するための変数

③I2C 通信開始。スレーブアドレス指定。

④00 番レジスタ指定（温度整数部）

⑤I2C 通信終了

⑥I2C 通信で 1byte（温度整数部）読込リクエスト

⑦I2C 通信で 1byte 読込

⑧整数部温度を PC に通知（改行無し）

⑨続けて小数点を通知（改行無し）

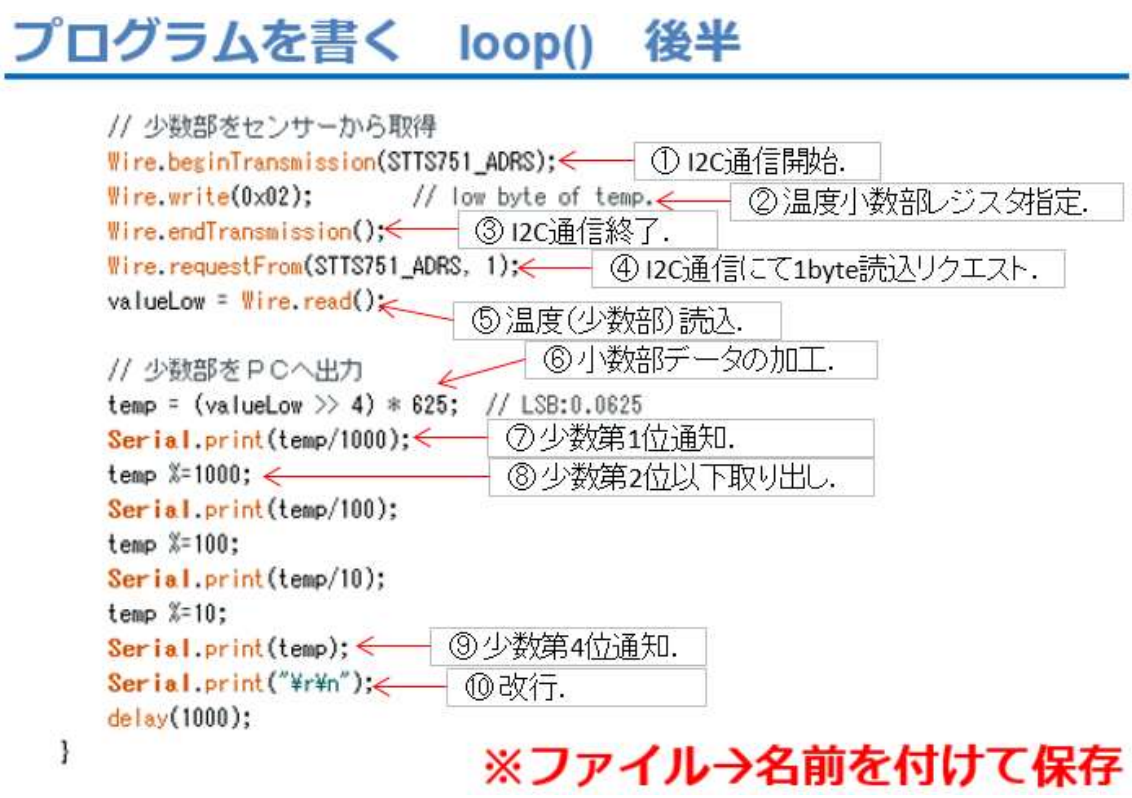


図　121

上記で特に説明をする部分は、少数部を PC へ出力している部分です。

⑥小数部温度のデータを 4bit 右シフトして分解能 625 を掛ける

⑦少数第 1 位出力

⑧余りを採って以下の桁の処理を続ける

⑨少数第 4 位出力

⑩改行を出力

ソースコードを入力したら、名前を付けて保存してください。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

図　122

シリアルモニターを起動して、通信速度を合わせてください。今回は少し早めの 115200bps としました。通信速度を合わせると、1 秒毎に温度が表示されていきます。温度センサーの背中に軽く指を触れてみてください。温度が変化することが分かるでしょう。

今回は、センサーの分解能である 1/16℃まで表示していますが、実際に使用する場合は、必要な表示桁数などの検討をしてください。

【重要】

このセンサーは、デジタルセンサーなので、I2C I/F 上に複数デバイスを接続できます。次の講座では、I2C I/F で制御する液晶表示器を同じ I/F に接続します。

第8回　液晶表示器

これまで、WiFiマイコンからの表示装置としてPCのシリアルモニターを使っていたわけですが、これからはWiFiマイコン単体でも表示ができるように、液晶表示器（LCD：Liquid Crystal Display）を使ってみましょう。前回使用したデジタル温度センサーはI2C I/Fを使用して温度を読み込む入力デバイスでしたが、今回は表示データを書込んで目に見える文字情報を表示する出力デバイスとして液晶表示器を使います。

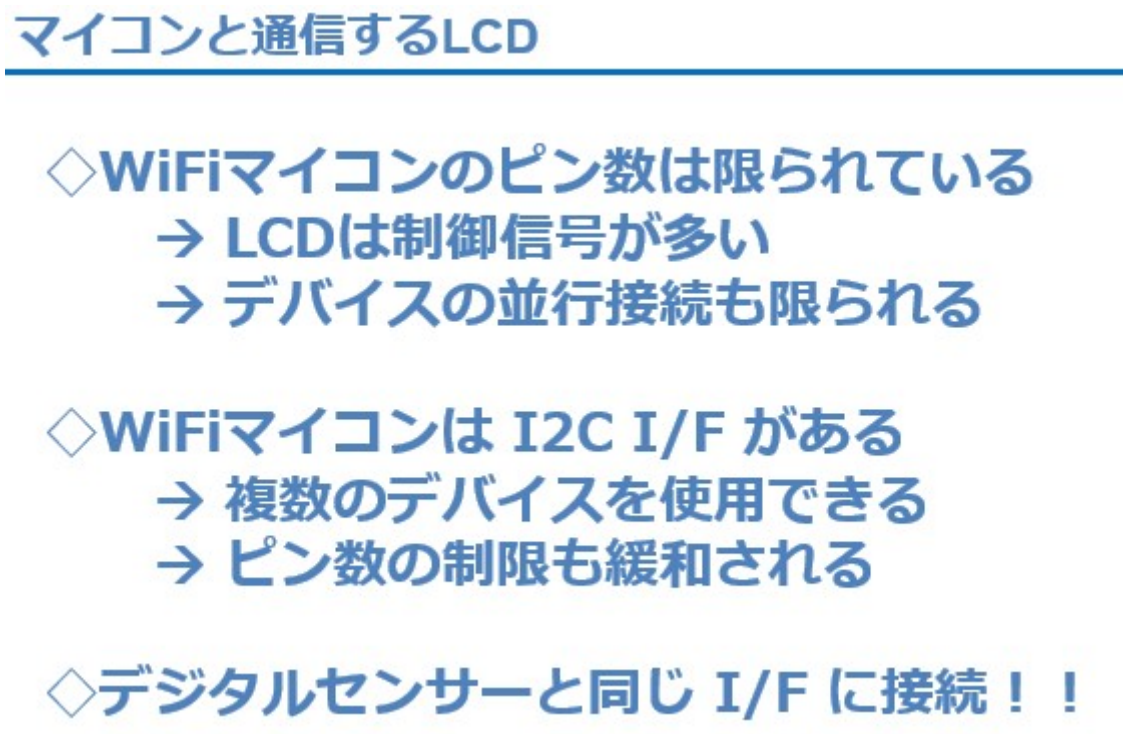


図　123

マイコンの王道・・・表示機能【LCD】

＜＜LCD＞＞

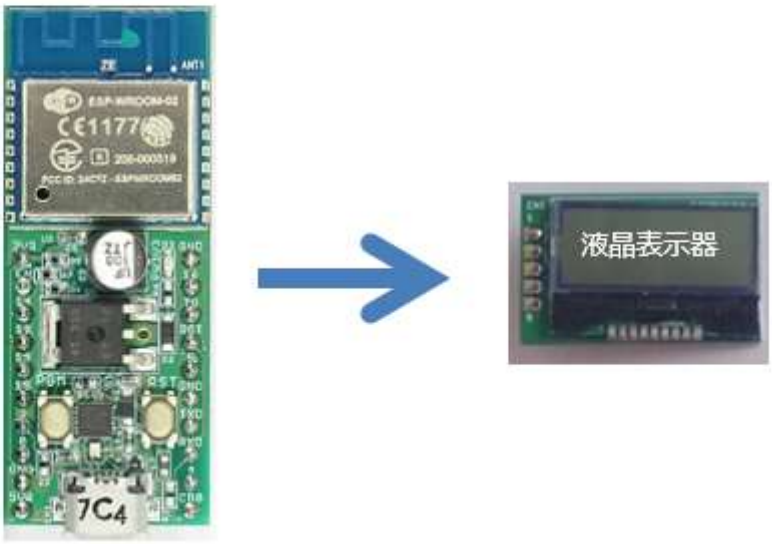


図　124

このシステムで利用する液晶表示器は、図右の様なものです。今回は使い方を学ぶことに重点を置いて、まずは固定の文字列を表示してみましょう。シリアル通信の【送信】と同じ考え方です。固定の文字列が表示できれば、標示する文字列をダイナミックに変化するデータで編集することで、【色々な情報を外部に伝える重要な仕事】が行えるようになります。

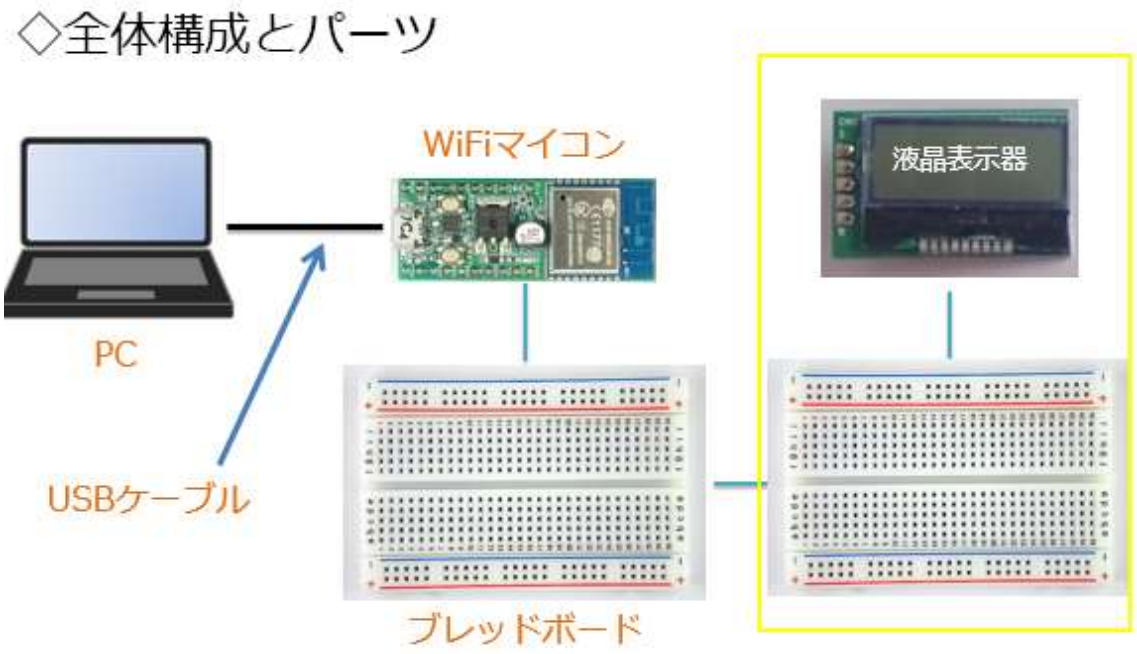


図　125

システムの全体構成を上図に示します。前回使用したデジタル温度センサーの回路を残したままま、液晶表示器を取り扱います。ブレッドボードが混雑するので、液晶表示器は別のブレッドボードに液晶表示器ユニットとして作成することにしました。

必要な機材・パーツは、下記です。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×2 個
5. 配線用ジャンパー線×適宜
6. LCD（液晶表示器）×1 個
7. デジタル温度センサー（STTS751）×1 個（前回のまま）

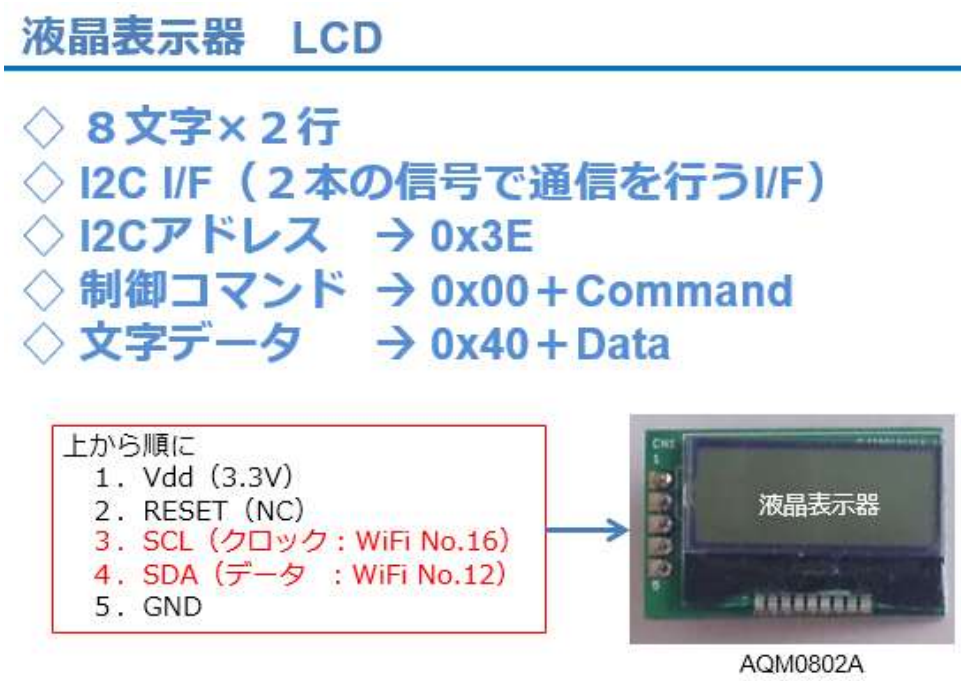


図　126

使用するLCDは、AQM0802Aという、８文字×2行の表示器です。デジタルセンサーと同様のI2 I/Fを用いるので、スレーブアドレス（0x3E）が設定されています。基板の左端には５本のピンが取り付けられていて、上から順に3.3V、RESET、SCL、SDA、GNDとなっていますが、このうちRESET信号は使用しません。LCDの初期化は、WiFiマイコンのプログラムで行います。

I2Cの通信は２種類あって、初期化や消去などの制御コマンドを送る場合と、表示する文字データの送信に分かれています。制御コマンドの場合は、最初に0x00を送り、続けてコマンドコードを、文字データの場合は最初に0x40を送り、続けて文字コードを送る仕組みです。これらはプログラムで処理します。

デジタルセンサー回路

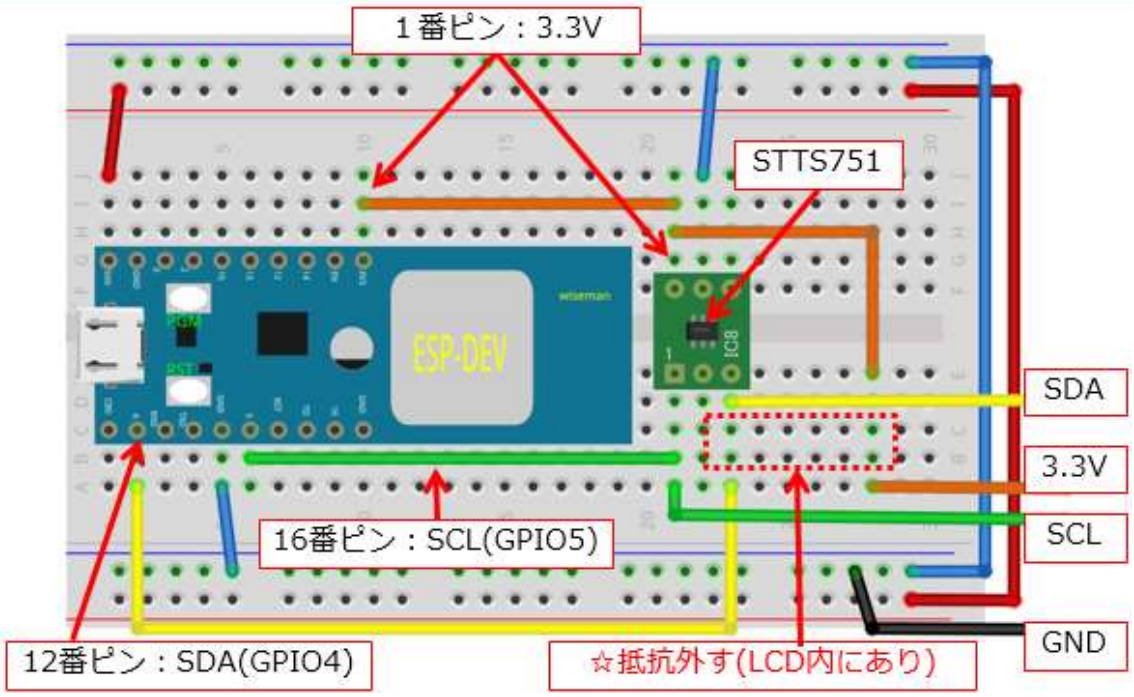


図　127

回路図はWiFiマイコン側とLCDユニット側に分かれています。

WiFiマイコン側は、前回のデジタル温度センサー回路を流用します。少し変更する点があります。【10KΩのPull Up抵抗は外します。】なぜかというとI2C I/Fを持つLCDの内部に同じ目的の抵抗が内蔵されているので、その他のPull Upが不要になるからです。LCDユニットには図の右側に出ているSDA、3.3V、SCL、GNDの4本のジャンパー線で接続します。

【重要】

この Pull Up 抵抗を取り外さなくても動くと思います。が、さらに Pull Up 抵抗を内蔵している I2C デバイスを増やすと、抵抗の並列接続となり I2C I/F に流れる電流が多くなって、I/F が不安定になります。

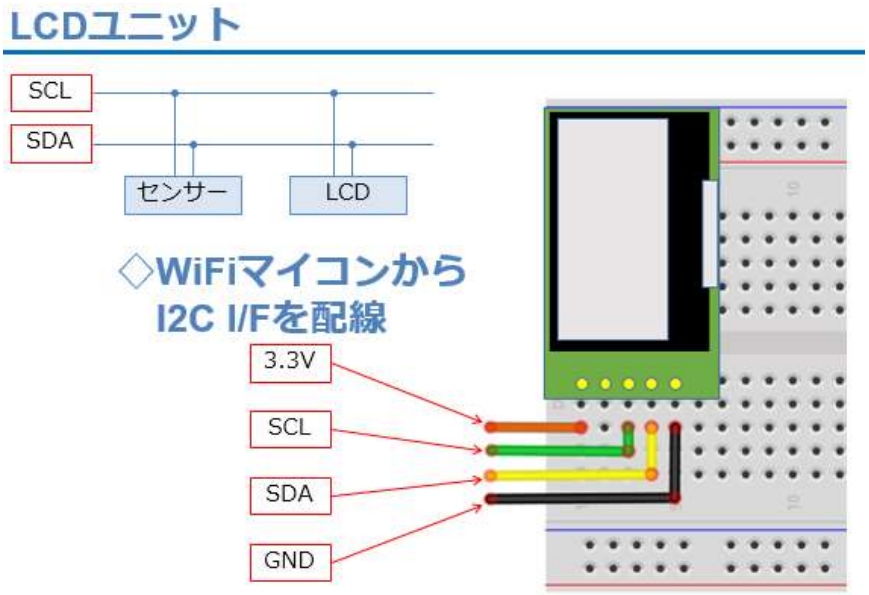


図　128

LCD ユニットは上図の様に配線してください。今回の回路では、デジタルセンサーも併設しているので、図左上に描いたように SCL,SDA にセンサーと LCD がぶら下がっている形となります。

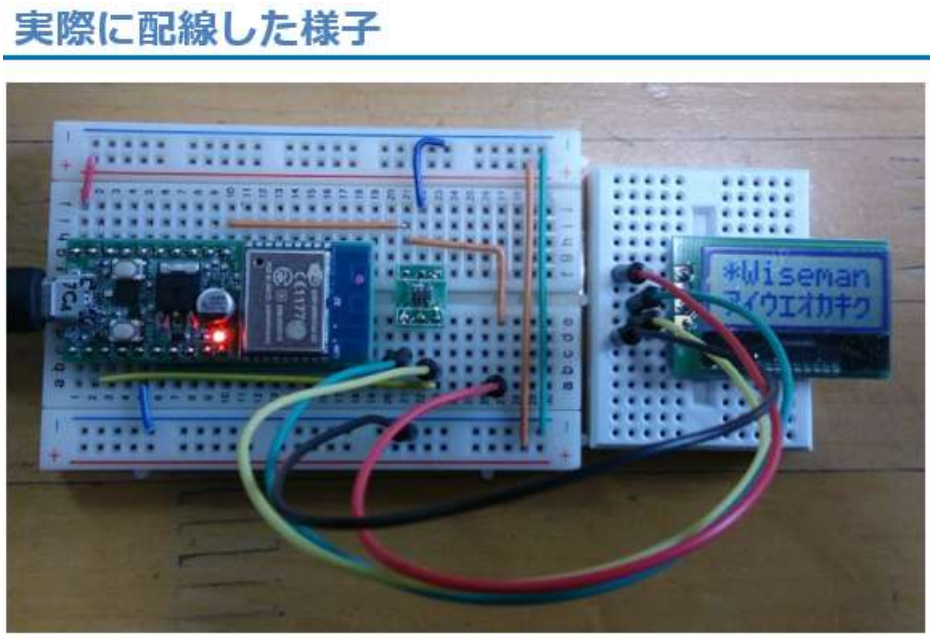


図　129

プログラムを書く

ESP_2108_LCD_8X2

```
#include <Wire.h>          ← ① I2Cデバイス通信のライブラリヘッダ.
#define LCD_ADRS 0x3E      ← ② LCDスレーブアドレス.
char moji[]="*Wiseman";    ← ③ 表示文字列.

//SCL=16:LCD No.3  SDA=12:LCD No.4

void setup() {
  Wire.begin();            ← ④ I2C I/F初期化.
  init_LCD();              ← ⑤ LCD初期化.
}
```

図 130

①I2C 通信ライブラリ用ヘッダ

②LCD の I2C スレーブアドレス

③LCD に表示する固定文字列バッファ（8 文字）

④I2C I/F 初期化

⑤LCD 初期化。関数の中身は後で説明

プログラムを書く　loop()

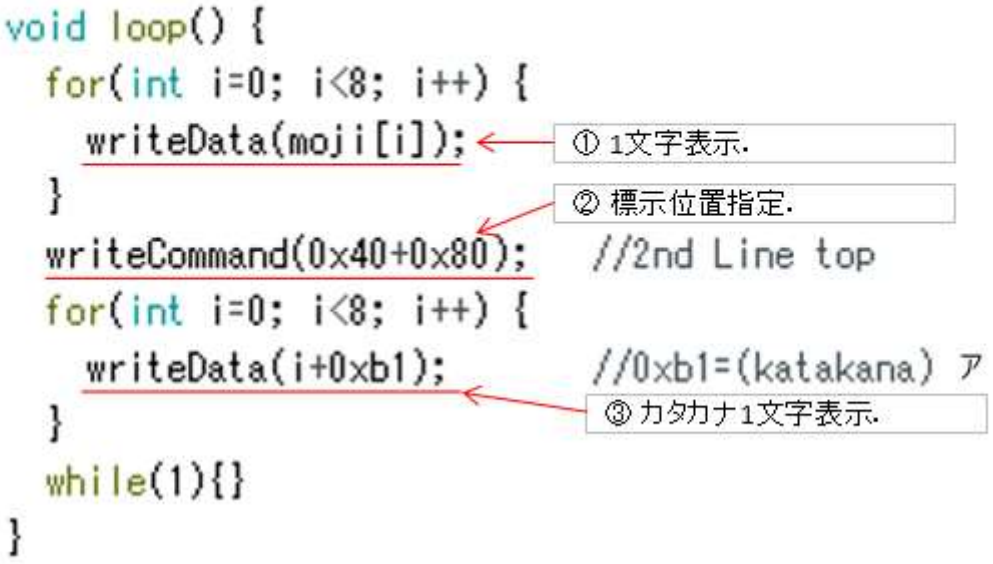


図　131

①バッファ内文字列を１文字表示

②LCD の２行目先頭を指定（LCD 内の表示メモリアドレスを指定）

③１文字表示（0xb1 は【ア】）アイウエオ…

表示は１回行えばよいので、最後に while(1)で永久ループとします。

プログラムを書く　文字送信・コマンド送信

```
void writeData(byte t_data) {          ① I2C通信開始.
  Wire.beginTransmission(LCD_ADRS);
  Wire.write(0x40);                    ② 0x40送信.
  Wire.write(t_data);                  ③ 表示文字コード送信.
  Wire.endTransmission();              ④ I2C通信終了.
  delay(1);
}

void writeCommand(byte t_command){
  Wire.beginTransmission(LCD_ADRS);    ⑤ I2C通信開始.
  Wire.write(0x00);                    ⑥ 0x00送信.
  Wire.write(t_command);               ⑦ コマンドコード送信.
  Wire.endTransmission();              ⑧ I2C通信終了.
  delay(10);
}
```

図　132

①I2C 通信開始

②表示文字通知指定

③文字コード送信

④I2C 通信終了

⑤I2C 通信開始

⑥コマンド通知指定

⑦コマンドコード送信

※コマンドコードは沢山種類があります。詳細はデータシートを入手して参照してください。

⑧I2C 通信終了

プログラムを書く　LCD初期化

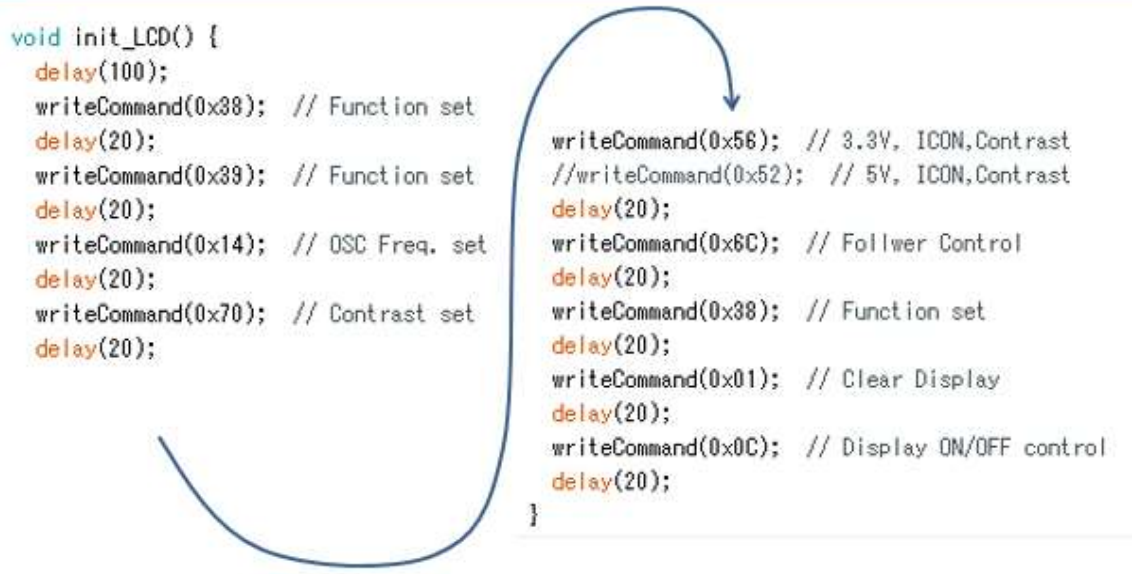


図　133

上記関数 init_LCD()は、writeCommand()で LCD 初期化のためのコマンドコードを連続して送信して、LCD 内部を初期化します。初期化用コマンドは、デバイスのデータシートに記載があります。サンプルコードが提供されることもあります。この LCD は AQM0802A という型番です。WEB で検索すると多くの情報とサンプルがヒットします。調べてみてください。

ソースコードを入力したら、名前を付けて保存してください。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

図 134

動作確認は、LCD を見れば一目瞭然です。プログラムで設定した文字列が 1 行目に表示され、2 行目はアイウエオ・・・とカタカナが並びます。

次回は、デジタル温度センサーと LCD を同時に使用して、デジタル温度計を開発します。

第9回　デジタル温度計

第７回ではデジタル温度センサー、第８回では液晶表示器（LCD）を使ってみたわけですが、第８回は、その両者を合体して、単体で温度を計測・表示するデジタル温度計を開発します。

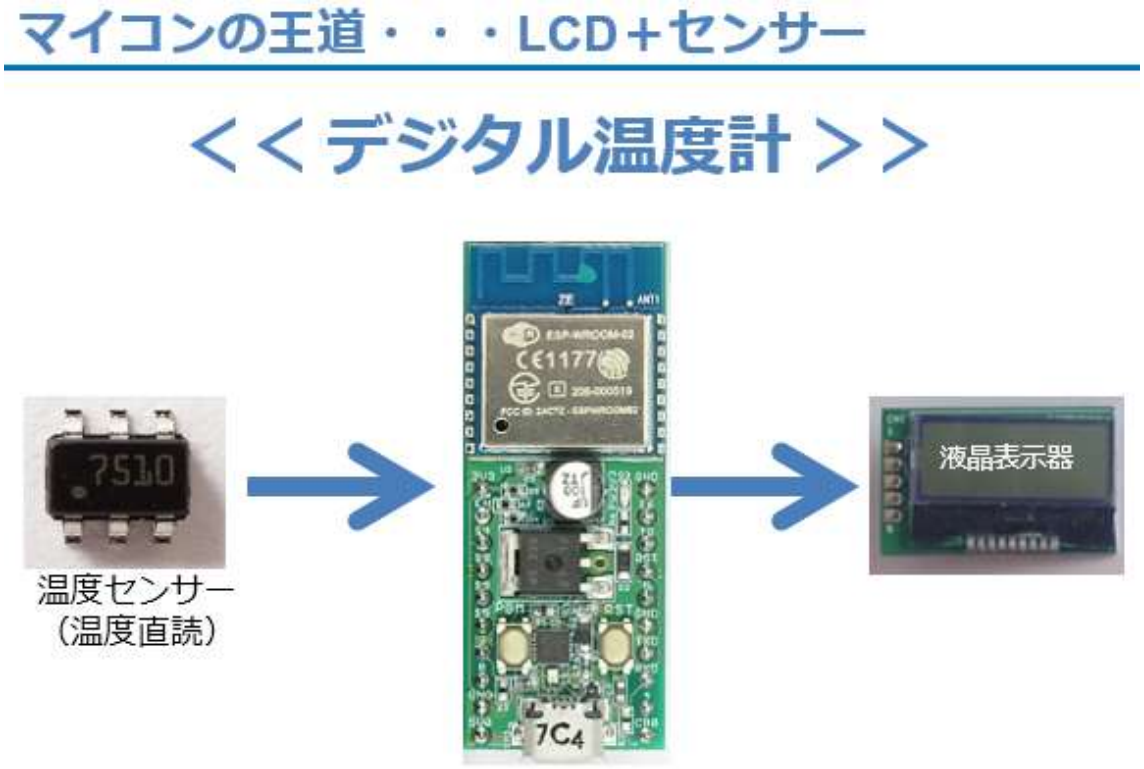


図　135

デジタル温度センサー(STTS751)は、最小計測温度が 1/16℃という、相当精度の高い温度センサーでした。計測した温度を文字列にして、PC に通知していましたが、その文字列を LCD 用に編集してあげることで、デジタル温度計が開発できることは、容易に想像できます。PC にも通知しながら、LCD にも表示するデジタル温度計を作りましょう。

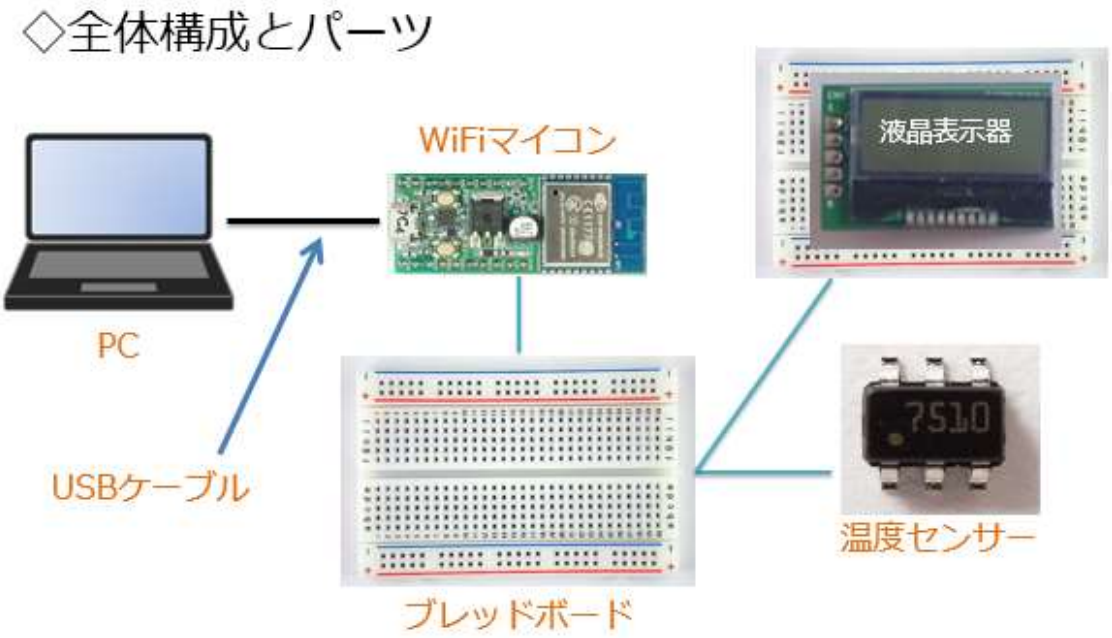


図 136

図で分かるように、直前 2 回の講座で使用したデバイスを使用します。そのために、前回もデジタル温度センサーを残したままの回路としました。

必要な機材・パーツは、下記です。（前回のまま）

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×2 個
5. 配線用ジャンパー線×適宜
6. LCD（液晶表示器）×1 個
7. デジタル温度センサー（STTS751）×1 個（前回のまま）

使用する LCD とデジタル温度センサーの仕様については、直前 2 回の内容を参照してください。回路も同じですが再掲します。

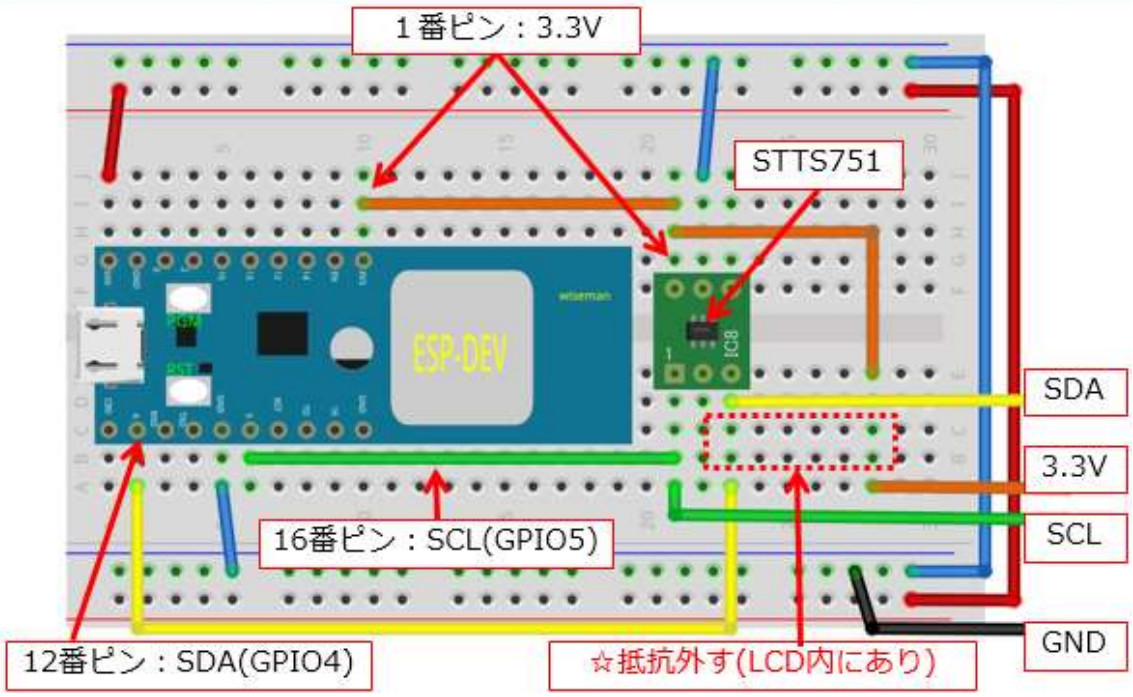

デジタルセンサー回路
1番ピン：3.3V
STTS751
ESP-DEV
SDA
3.3V
SCL
GND
16番ピン：SCL(GPIO5)
12番ピン：SDA(GPIO4)
☆抵抗外す(LCD内にあり)

図　137

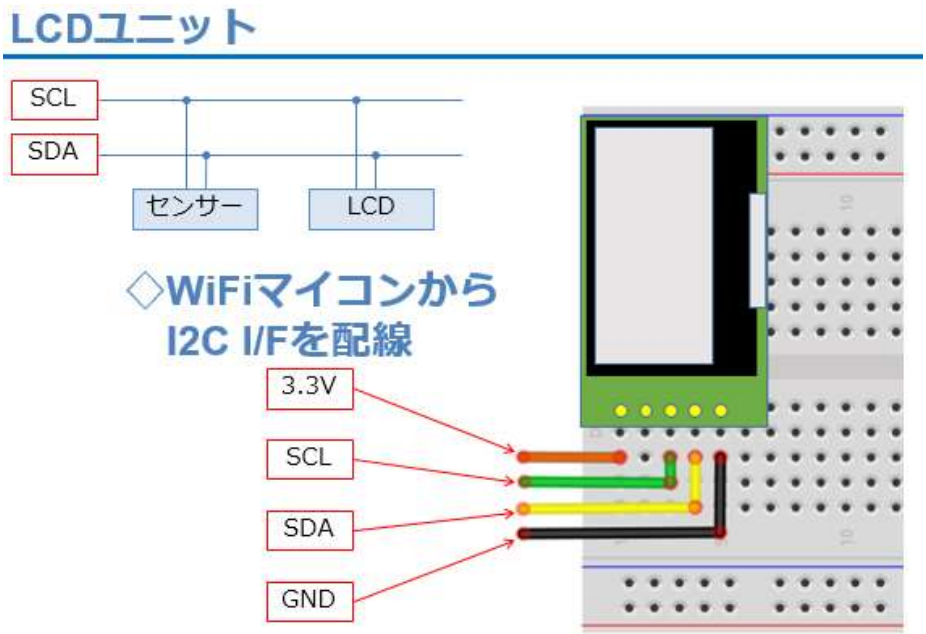

LCDユニット
SCL
SDA
センサー
LCD
◇WiFiマイコンから
I2C I/Fを配線
3.3V
SCL
SDA
GND

図　138

実際に配線した様子

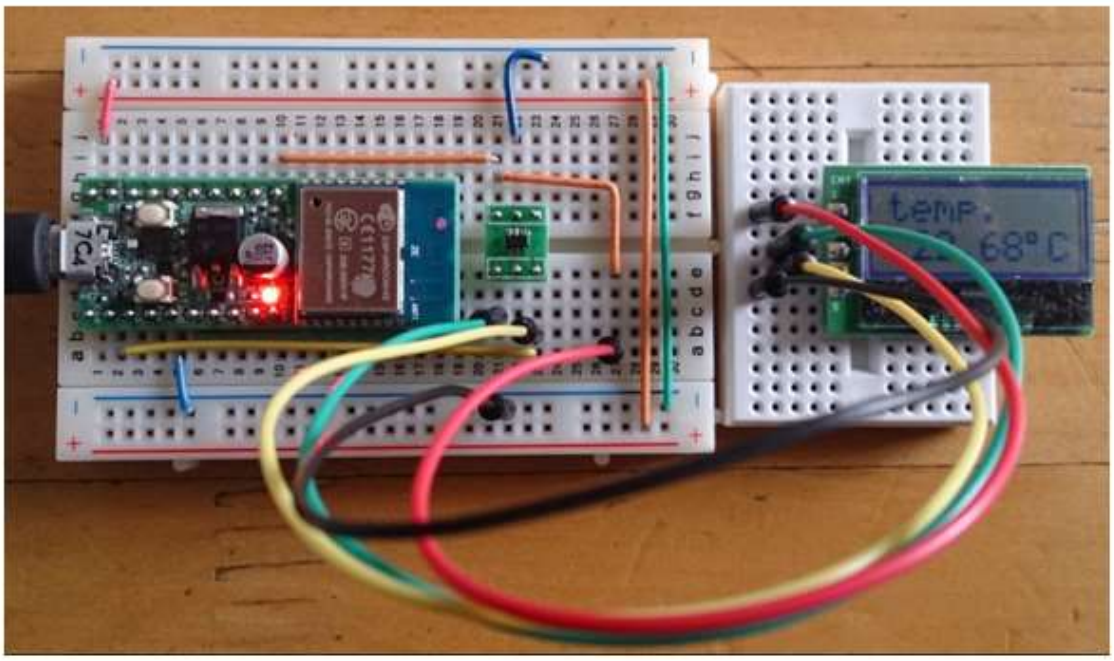


図 139

上の写真はすでに動作をしている様子です。今回は既に 1 度、回路も完成しているのでソフトウエア主体の開発です。しかし、開発と言っても直前 2 回分のソースコードを合体させて、LCD 表示文字列の編集部分のみ作成すればよいので、後に示すソースコード全体量で感じるほどの新規開発分はありません。ほんの僅かです。

【重要】

【これら】のことは、しっかりとした基礎技術を身に付ければ、長く・有効に活用できて、応用開発も楽だということを示唆しています。

※【これら】とは、どのようなことでしょう？

次にプログラムを作成しますが、既に説明したようにデジタル温度センサーのプログラムと、LCD のプログラムを合体したものになるので、少し長めのソースコードになります。

プログラムを書く

ESP_2109_LCD_Temp

```
#include <Wire.h>                    ← ① I2Cデバイス通信のライブラリヘッダ.

#define STTS751_ADRS  0x39           ← ② 温度センサー スレーブアドレス.
#define LCD_ADRS      0x3E           ← ③ LCD スレーブアドレス.

char u_moji[]="temp.   ";            ← ④ 表示文字列(上段用).
char l_moji[]=" **.** C";            ← ⑤ 表示文字列(下段用).

//SCL=16:LCD No.3  SDA=12:LCD No.4

void setup() {
    Serial.begin(115200);            ← ⑥ シリアルポート 初期化.
    Wire.begin();                    ← ⑦ I2C通信初期化.
    init_STTS751();                  ← ⑧ 温度センサー初期化.
    init_LCD();                      ← ⑨ LCD初期化.
}
```

図　140

①I2C I/F を使用するライブラリ用ヘッダ

②温度センサーの I2C デバイス識別用スレーブアドレス

③LCD の I2C デバイス識別用スレーブアドレス

④表示文字列用文字配列（上段：1 行目）

⑤表示文字列用文字配列（下段：2 行目）

⑥シリアルポート初期化

⑦I2C 初期化

⑧温度センサー初期化（関数内容は後に解説します。）

⑨LCD 初期化（関数内容は後に解説します。）

プログラムを書く　loop()　前半

```
void loop() {
    byte valHigh,valLow;
    int temp;

    valHigh = readUpp();          ①// 整数部をセンサーから取得
    Serial.print(valHigh);          // 整数部を出力
    Serial.print(".");              // 小数点
    valLow = readLow();           ②// 少数部をセンサーから取得
    temp = (valLow >> 4) * 625;     // LSB:0.0625 小数部温度計算
    Serial.print(temp/1000);        // 0.1の位
    temp %=1000;
    Serial.print(temp/100);         // 0.01の位
    temp %=100;
    Serial.print(temp/10);          // 0.001の位
    temp %=10;
    Serial.println(temp);           // 0.0001の位
```

図　141

①温度の整数部をセンサーから取得し valHigh に格納

※readUpp()関数は後で解説

②温度の少数部をセンサーから取得し valLow に格納

※readLow()関数は後で解説

上の 2 つの部分は、第 7 回デジタル温度センサーの講座では、loop()の中に直に記述されていましたが、今回は 2 つの関数に分けました。

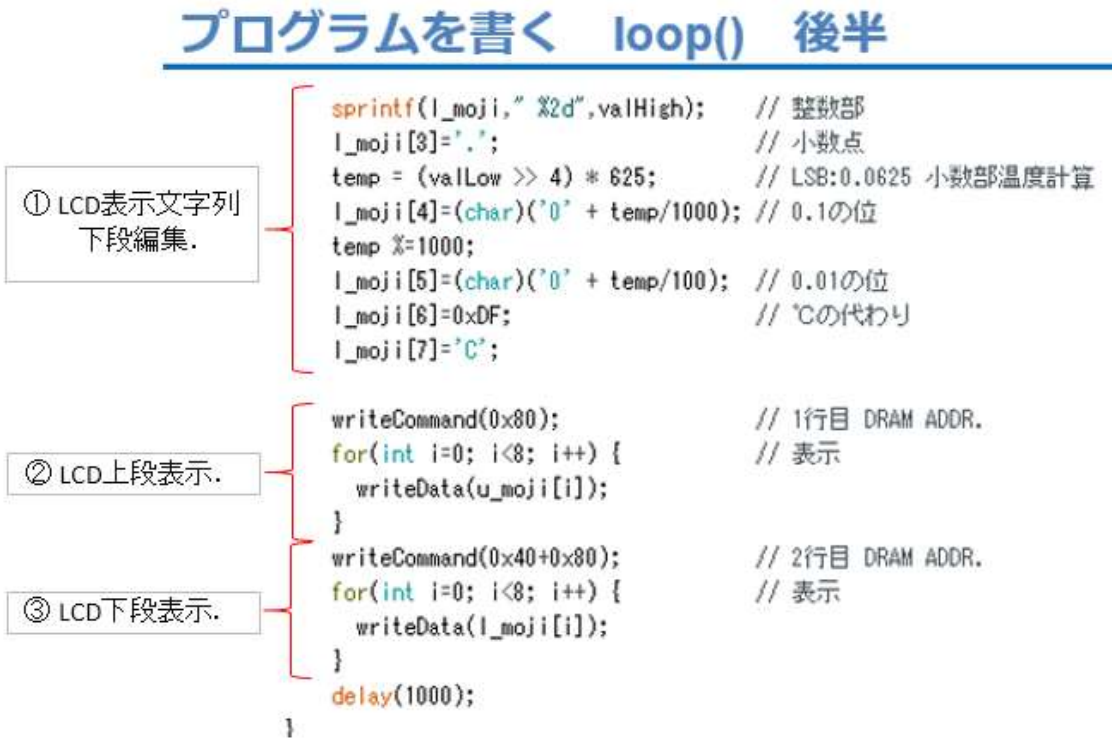


図　142

①LCD 下段に表示する温度表示用文字列の編集を行う。

文字列用バッファは、次の様に編集されます。

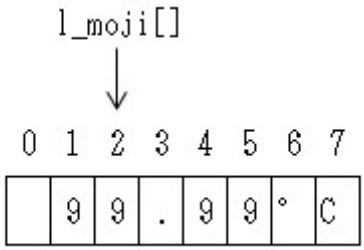


図　143

②LCD の上段（1 行目）に固定文字列を表示

③LCD の下段（2 行目）に温度文字列表示

delay(1000)ですので、1 秒おきに動作が繰り返されます。

プログラムを書く　温度読込

```
byte readUpp() {
    // 整数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);  ← ① I2C通信開始.
    Wire.write(0x00);  ← ② 温度整数部レジスタ指定.
    Wire.endTransmission();  ← ③ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);  ← ④ I2C通信にて1byte読込リクエスト.
    return(Wire.read());  ← ⑤ 温度(整数部)読込. 戻り値にセット.
}

byte readLow() {
    // 少数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);  ← ⑥ I2C通信開始.
    Wire.write(0x02);  ← ⑦ 温度小数部レジスタ指定.
    Wire.endTransmission();  ← ⑧ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);  ← ⑨ I2C通信にて1byte読込リクエスト.
    return(Wire.read());  ← ⑩ 温度(少数部)読込. 戻り値にセット.
}
```

図　144

①I2C 通信開始

②00 番（温度整数部）レジスタ指定

③I2C 通信終了

④I2C にて 1byte 読込リクエスト

⑤1byte（温度整数部）読込。そのまま戻り値としてセット

⑥I2C 通信開始

②02 番（温度小数部）レジスタ指定

③I2C 通信終了

④I2C にて 1byte 読込リクエスト

⑤1byte（温度小数部）読込。そのまま戻り値としてセット

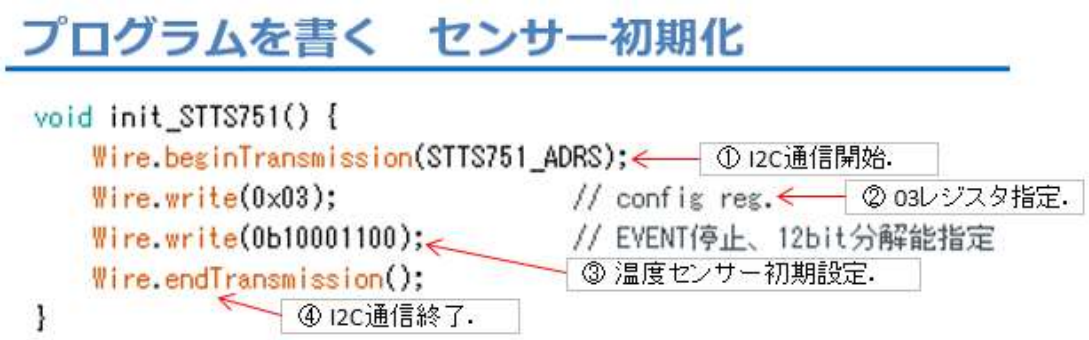


図　145

①I2C 通信開始

②03 番レジスタ(設定レジスタ)指定

③初期設定

④I2C 通信終了

図　146

①I2C 通信開始

②表示文字通知指定

③文字コード送信

④I2C 通信終了

⑤I2C 通信開始

⑥コマンド通知指定

⑦コマンドコード送信

※コマンドコードは沢山種類があります。詳細はデータシートを入手して参照してください。

⑧I2C 通信終了

各々の最後の delay()は、I2C I/F が切り替わるための待ち時間です。

プログラムを書く　LCD初期化

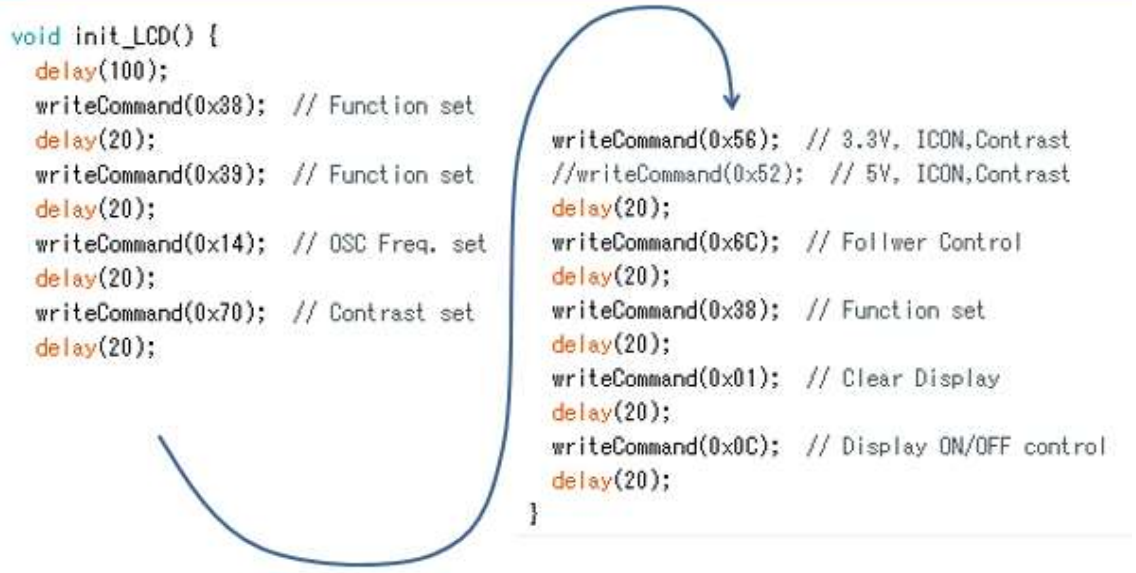


図　147

上記関数 init_LCD()は、writeCommand()で LCD 初期化のためのコマンドコードを連続して送信して、LCD 内部を初期化します。初期化用コマンドは、デバイスのデータシートに記載があります。サンプルコードが提供されることもあります。この LCD は AQM0802A という型番です。WEB で検索すると多くの情報とサンプルがヒットします。調べてみてください。

各コマンドの間にある delay()はコマンドが有効になるための待ち時間です。

ソースコードを入力したら、名前を付けて保存してください。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

図　148

動作確認です。

まず、LCD に図の様に整数部、小数部、℃の表示が行われます。次に、IDE からシリアルモニターを起動します。

図　149

通信速度を合わせると、温度が 1 秒ごとに表示されます。

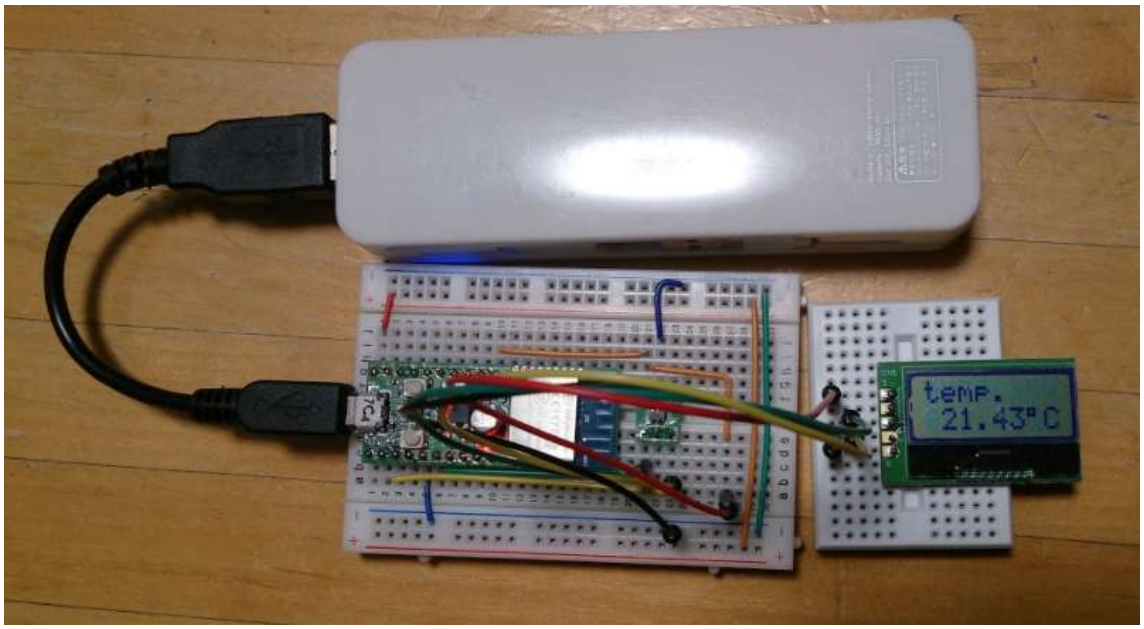


図 150

図の様に、USB ケーブルで電源（写真は携帯電話の充電用）を供給すれば、単独で稼動するデジタル温度計となります。

これまで、沢山の実習をしてきました。これで、WiFi マイコンの【マイコン機能】の使い方は終了です。次回から、いよいよ WiFi 機能を使った WEB 連携システムへの利用を実習します。

第10回　WEB連携①(MQTT)

いよいよ、WEB連携システムの開発に入ります。WEB連携の初めは、WiFiマイコンが発信するメッセージを、WEBを通じて携帯端末に届けてみましょう。

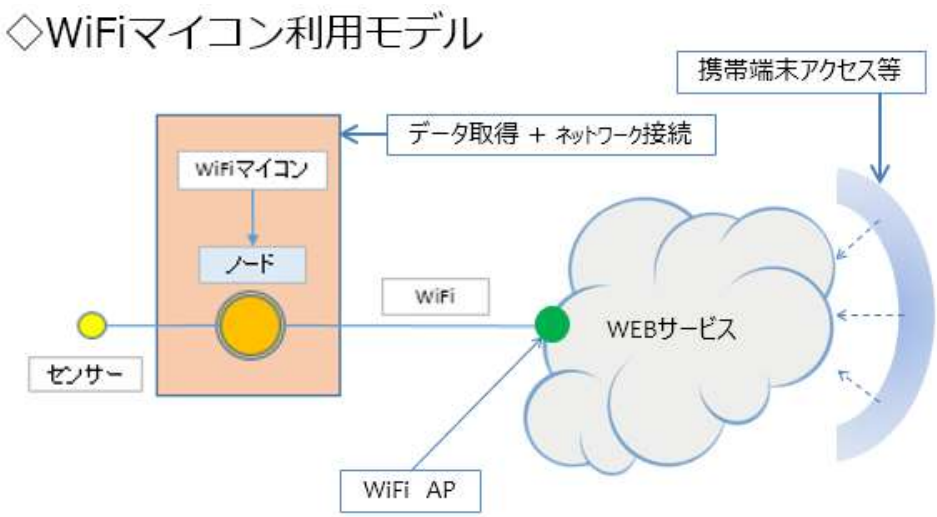


図　151

上の図を覚えていますか。第1回で説明した図です。フィールドに配置したセンサーはノードであるWiFiマイコンに接続されて、ノードで収集した環境情報は、WiFi機能でアクセスポイントを通じて、WEBに送られます。それを離れたところにある携帯端末などでアクセスするというモデルです。WEBサービスと説明しているのはホームページのようなものと考えておけば良いと思います。この仕組みであれば、遠隔地にある携帯端末は台数に関係なく、ネット環境が整えばどこからでも、いつでも、何人でもフィールドの状態を監視することができます。

「これぞIoT！！」と言えます。

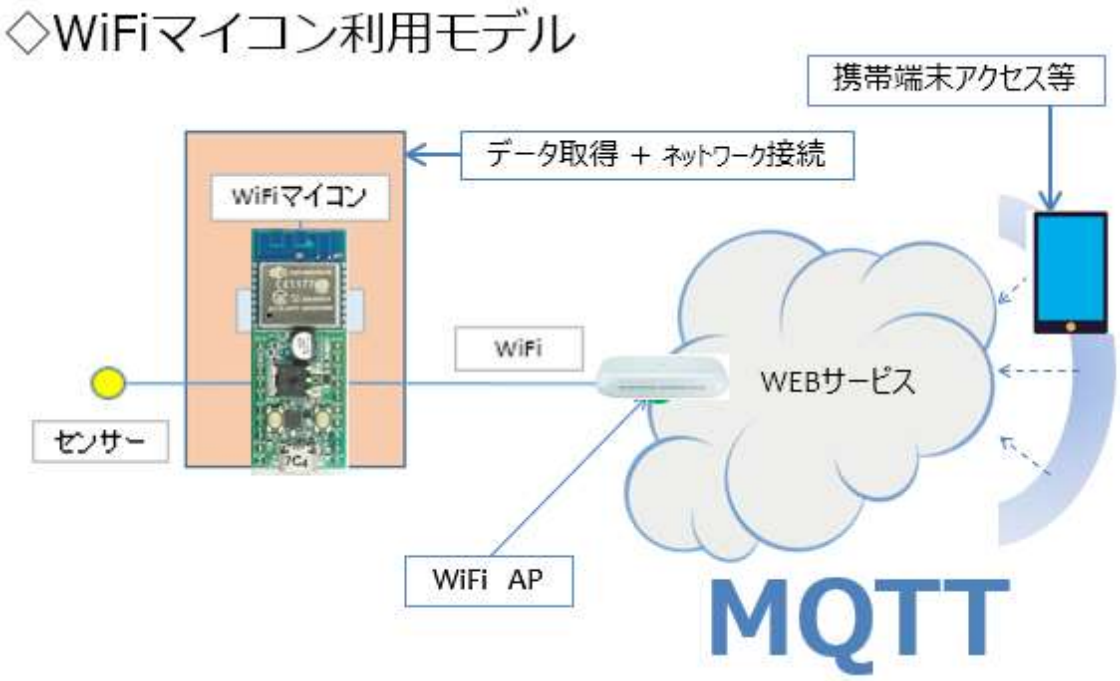


図 152

このモデルに、これから使用する機材等を重ねたものが上の図です。WiFi マイコンは自ら、アクセスポイント（AP）に接続します。このアクセスポイントは、普通にオフィスや学校、家庭などで利用されているアクセスポイントです。アクセスポイントは Internet に接続しています。WiFi マイコンは、アクセスポイント経由で Internet 上の WEB サービスに接続して情報を通知します。WEB サービスは、通知された情報を預かり、Internet 経由の情報アクセスの要求に対して、預かっている情報を提供する、という仕組みです。

今回は初めての WEB サービス利用なので、固定のメッセージを遠隔地に通知するシステムを開発します。シリアル通信の【送信】に相当することを WEB 経由で行います。

使用する WEB サービスは MQTT(後で説明します。)というものです。

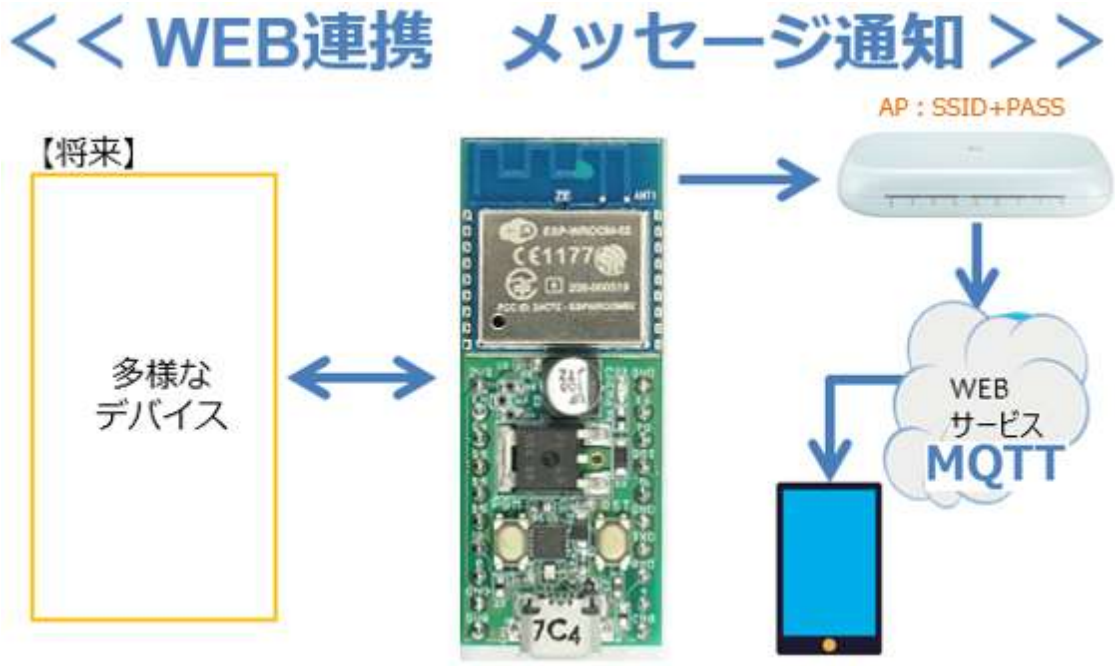


図　153

概要を示したものが上の図です。WiFi マイコンはアクセスポイントを通じて WEB サービスに接続しますので、利用するアクセスポイントの SSID と Pass Word が必要になります。調べて準備しておいて下さい。当然ですが、使用するネットワークは ISP（Internet Service Provider）に接続できる必用があります。WiFi マイコンが通知する固定メッセージは、WEB サービスで一時預かりとなり、外部から要求のあったスマートフォンなどの形態端末（PC でも可。但し専用アプリが必要。）に提供します。今回は固定のメッセージを通知しますが、メッセージの内容をダイナミックに変化する情報で編集すれば、様々な情報が遠隔地で受け取れるシステムとなります。そのダイナミックに変化する情報の元になるものは、図の左側で示す多様なデバイスです。将来は WEB に通知するだけでなく、WEB 経由でマイコンが情報を得ることもできるシステムの基礎となるモデルを今回は開発します。

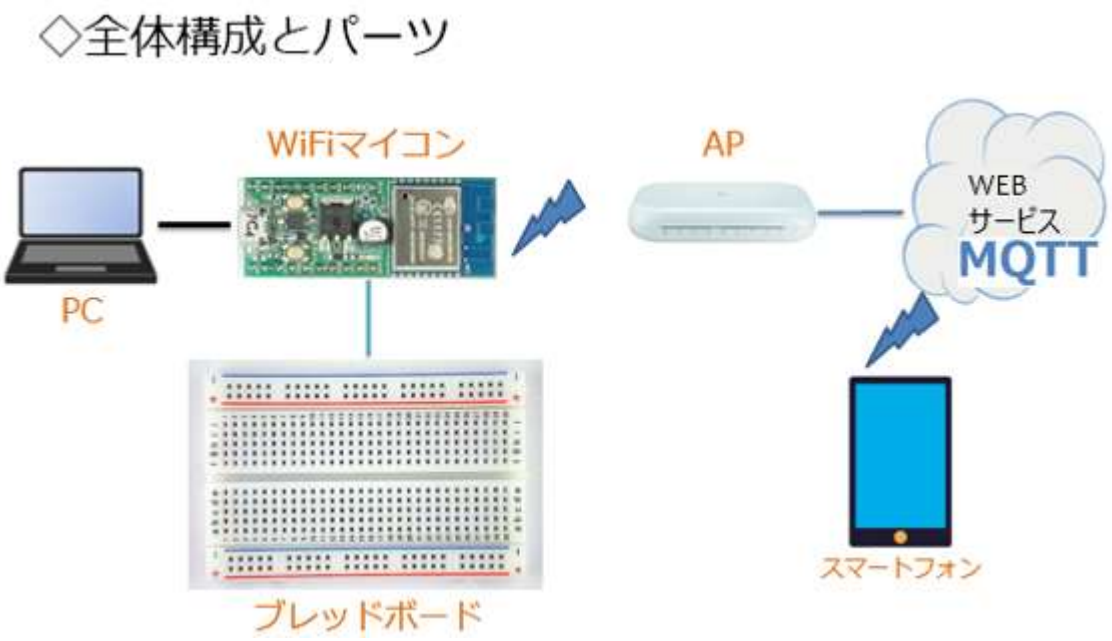


図 154

全体の構成は上の図の通りです。携帯端末としてはスマートフォンを利用します。必要な機材・パーツは、下記です。

1. スマートフォン×1 台（解説は Android 携帯）
2. WiFi マイコン×1 台
3. PC（プログラム開発・書込）×1 台
4. USB ケーブル（マイコンとの接続）×1 本
5. ブレッドボード×2 個
6. 配線用ジャンパー線×適宜

ここで、今回利用する WEB サービス【MQTT】について説明します。

図　155

MQTT とは Message Queue Telemetry Transport の頭文字を並べたものです。これは、短いメッセージのやり取りに特化した仕組みです。利用することがとても簡単で、各種コンピュータや言語向けのライブラリも多く公開されています。マイコンでも利用可能な WEB サービスの代表格となっています。WEB 上では、これを利用してメッセージをやり取りするのですが、そのメッセージの仲立ちをしてくれるサービスの本体が MQTT Broker というものです。次の図で MQTT Broker を通じてメッセージの交換を行う様子を示します。

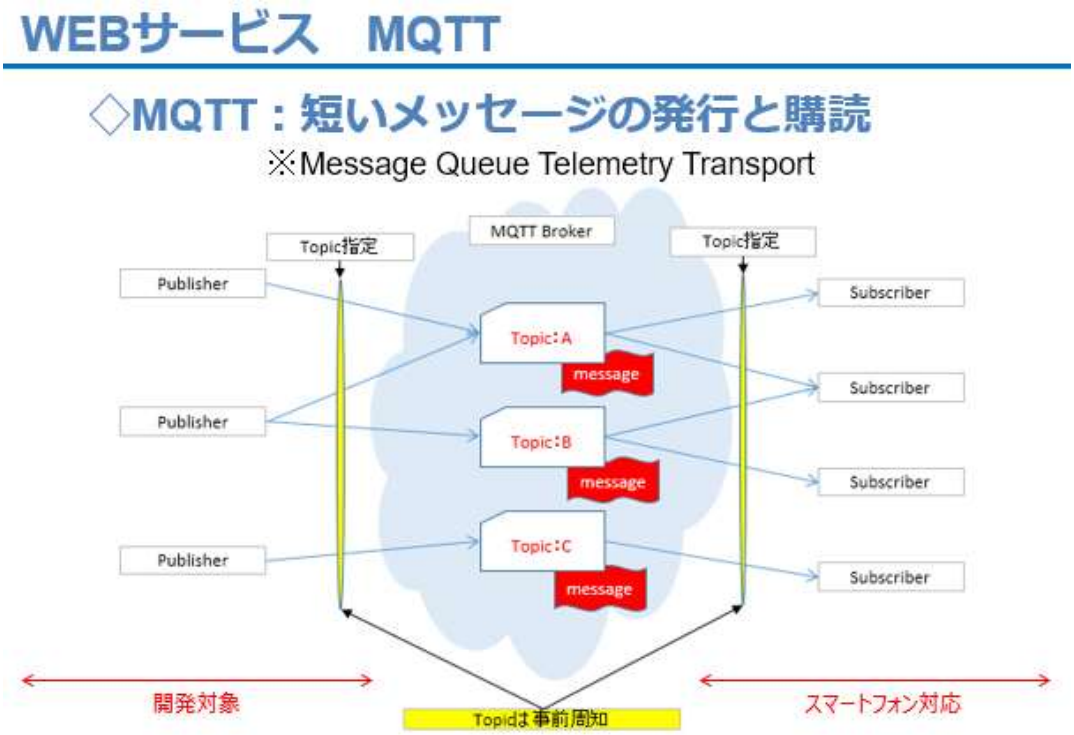


図　156

図左がメッセージを発行する側【Publisher：発行者】、図右がメッセージの読み手【Subscriber：購読者】を示しています。【Publisher】と【Subscriber】は事前に交換するメッセージについて取り決めをして、メッセージの見出し【Topic】を決めておきます。【Publisher】は、発行するメッセージができたら、【Topic】を付けて、メッセージを MQTT Broker に発行【Publish】します。【Subscriber】はあらかじめ、購読したいメッセージの【Topic】を登録しておきます。新しいメッセージが発行されると、MQTT Broker は購読者が登録している【Topic】が同じであれば該当の登録者にメッセージの発行を通知し、購読者は購読するという仕組みで、メッセージが届きます。【Publisher】【Subscriber】ともに、使用する【Topic】は複数使用することができるので、相手を知らなくてもメッセージの交換ができます。

今回は、【Piblisher】側を開発対象として【Subscriber】側はスマートフォンの公開アプリを利用してメッセージ通知を行います。

図 157

公開されている MQTT Broker は、とてもたくさんありますが、その中で、今回は上図の【broker.hivemq.com】を利用します。

図 158

MQTT サービスに発行したメッセージを読むためには、スマートフォン側でアプリケーションを準備します。沢山のアプリケーションが公開されていますが、その中で MyMQTT（Android 版）を使用します。（※Android 以外のものも沢山公開されていますので、皆さんの環境にあったものを見つけてください。）

MyMQTT をインストールすると、上図右のようなショートカットができます。このアプリの使いかたを説明します。

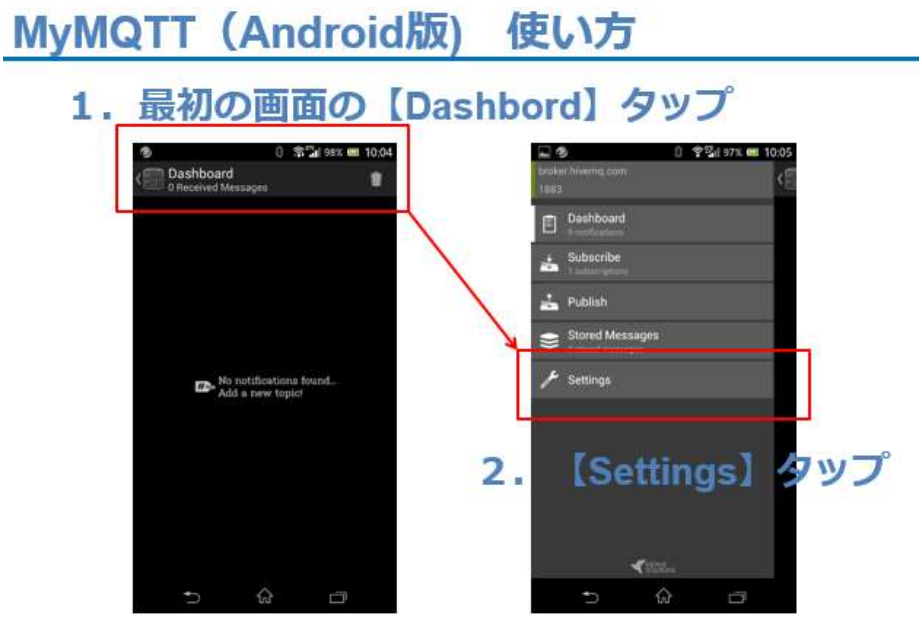


図 159

MyMQTT を起動すると図左の画面になります。

1．Dashboard をタップすると、右側の画面になります。

2．Settings をタップします。次の画面に変わります。

図 160

３．Broker 名 【broker.hivem.com】入力して、save をタップします。

※この時、指定した MQTT Broker に接続が行われます。Successfully conneted とメッセージが表示されない場合は、入力した MQTT Broker を確認して下さい。このアプリケーションは、起動するたびに設定した MQTT Broker への接続を行って、接続の状況をメッセージで知らせてくれるので、それを確認してから使用するようにしてください。

図 161

４．Settings をタップして、前の画面にもどります。

５．Subscribe をタップします。

図 162

【重要】

ここで登録する Topic 名はどのような名称でも構いません。後で作成するプログラムに記述しますので、記録しておいてください。

６．Topic を入力して Add をタップして登録します。（図の７，８）

図 163

９．Subcribe をタップして元に戻ります。

１０．Dashboard で Subscribe します。

※通常、アプリを起動して MQTT Broker に接続確認したら、Dashboard を使用して発行されメッセージを購読する、という流れで使用します。

細かな点の使いかたは省略していますが、使用しながら操作に慣れて下さい。また、他のアプリも試してみて、皆さんの使い易いものを発見していただくのも面白いと思います。

◇Subscribeの様子

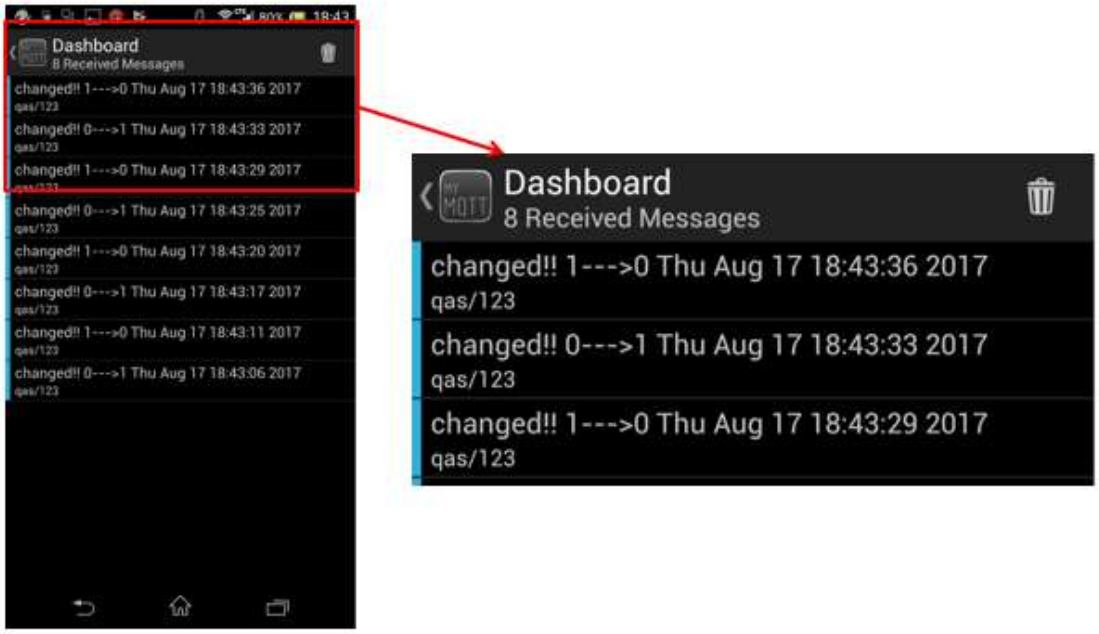


図 164

上図に Subscribe の様子を示します。Dashboard を見ていると、メッセージが発行される度に Dashboard にその内容が表示されていきます。

購読したメッセージは、新しいものを上に積上げられ、古いものは下に下がっていきますので、動作確認を行う際は、Dashboard の最上部に注目していて下さい。

次に配線図と制作した様子を示します。回路は大変シンプルです。

今回のテーマでは WiFi マイコンが機能すれば良く、周辺デバイスは不要ですから、動作確認をした前回の回路を、そのまま使用してもかまいません。

メッセージ発行回路

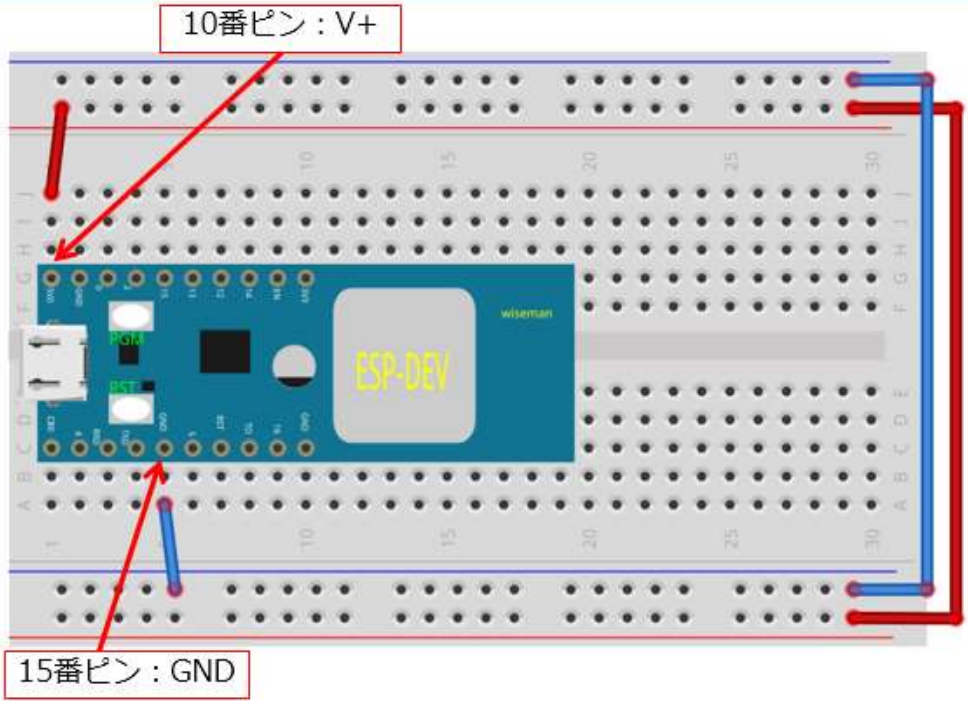


図　165

実際に配線した様子

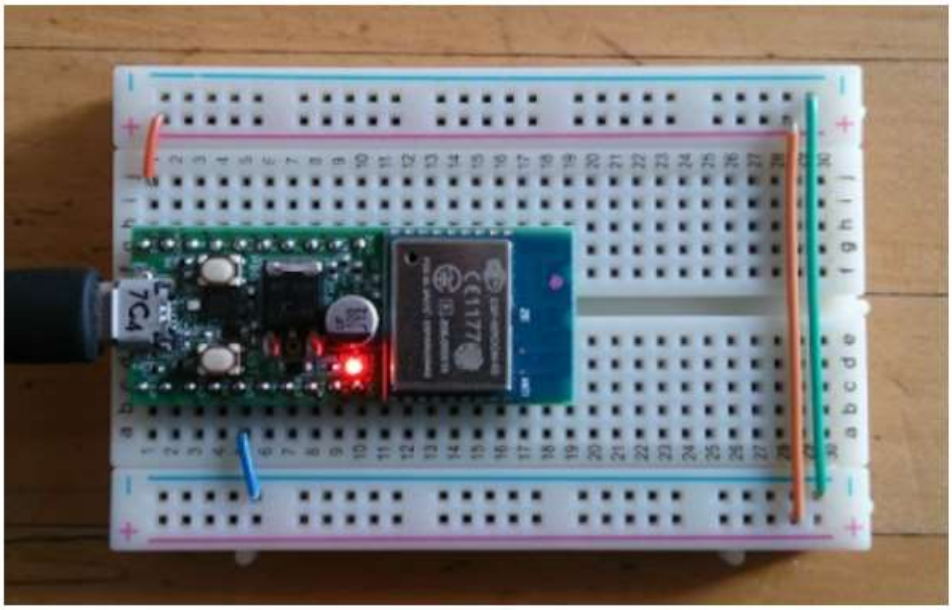


図　166

MQTTライブラリの準備

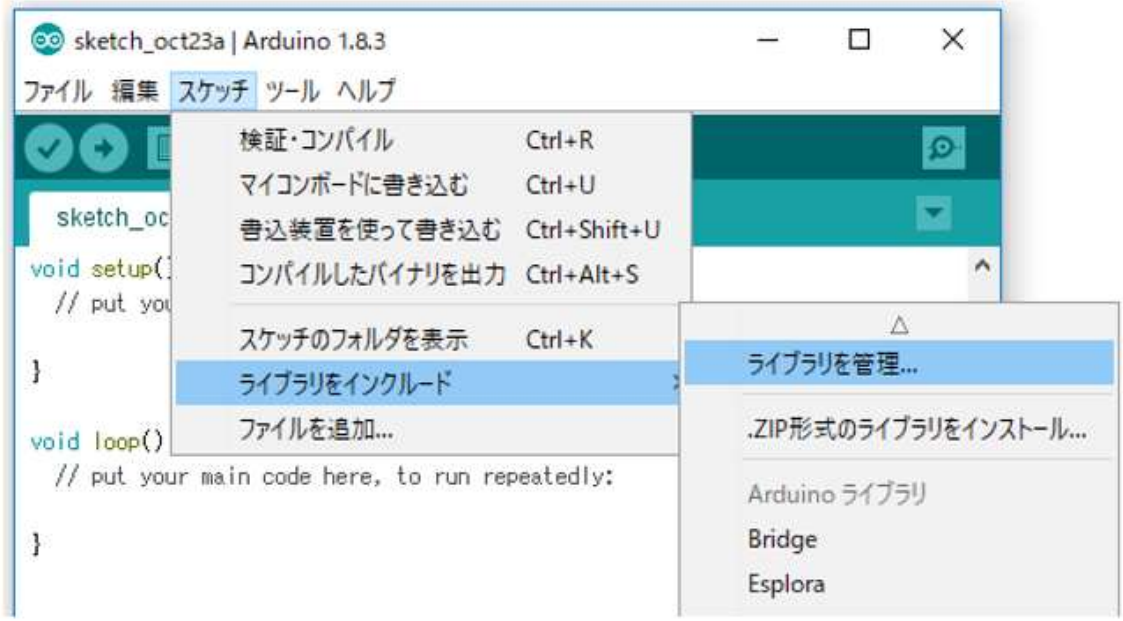


図 167

今回は、ソフトウエア開発に当たり、MQTT ライブラリが必要ですので、ここで準備します。上図の手順、【スケッチ→ライブラリをインクルード→ライブラリを管理】を辿りライブラリマネージャを開いてください。

【注意】

この時、PC は Internet に接続された状態であることが必要です。ネットへの PC 接続を確認してください。

図 168

ライブラリマネージャの上部にある検索テキストに【pubsub】(半角小文字で可)と入力します。すると、該当のライブラリがネット上で検索されて【PubSubClient】がヒットします。この中央部分をクリックして選択すると、右下にインストールボタンが現れます。(下図)これをクリックして、インストールして下さい。

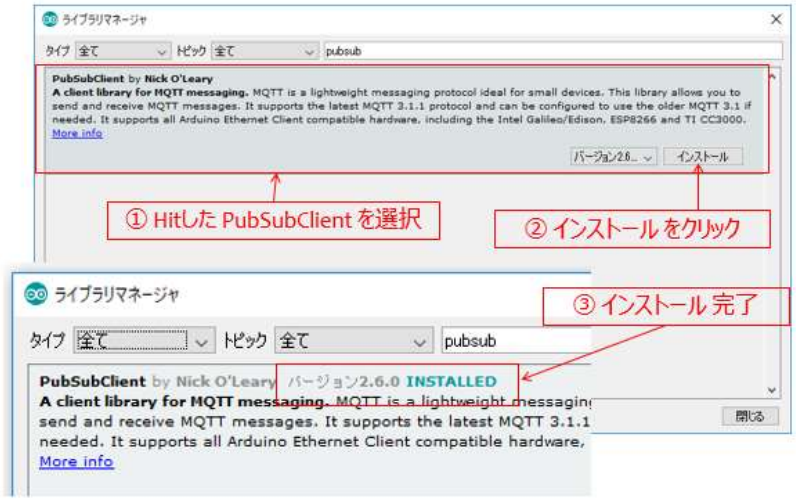


図 169

MQTTライブラリの準備

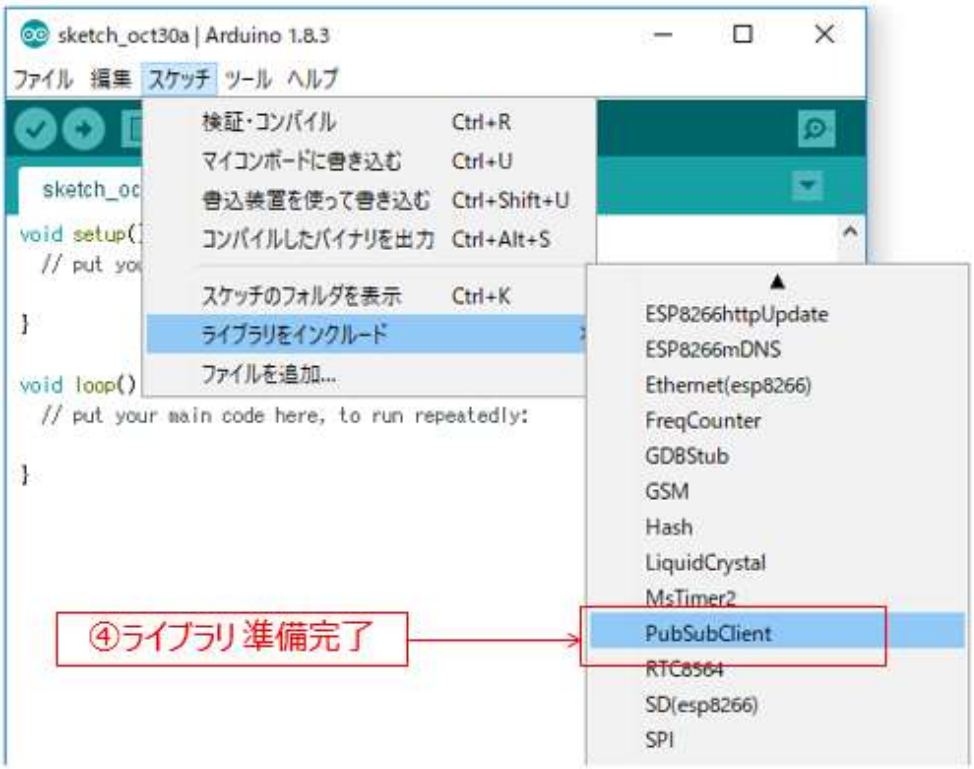


図 170

ライブラリが正しく準備できたことを確認するには【スケッチ→ライブラリをインクルード】と辿って表示される一覧の中に【PubSubClient】が表示されていれば、準備完了です。

以下、ソースコードを示します。

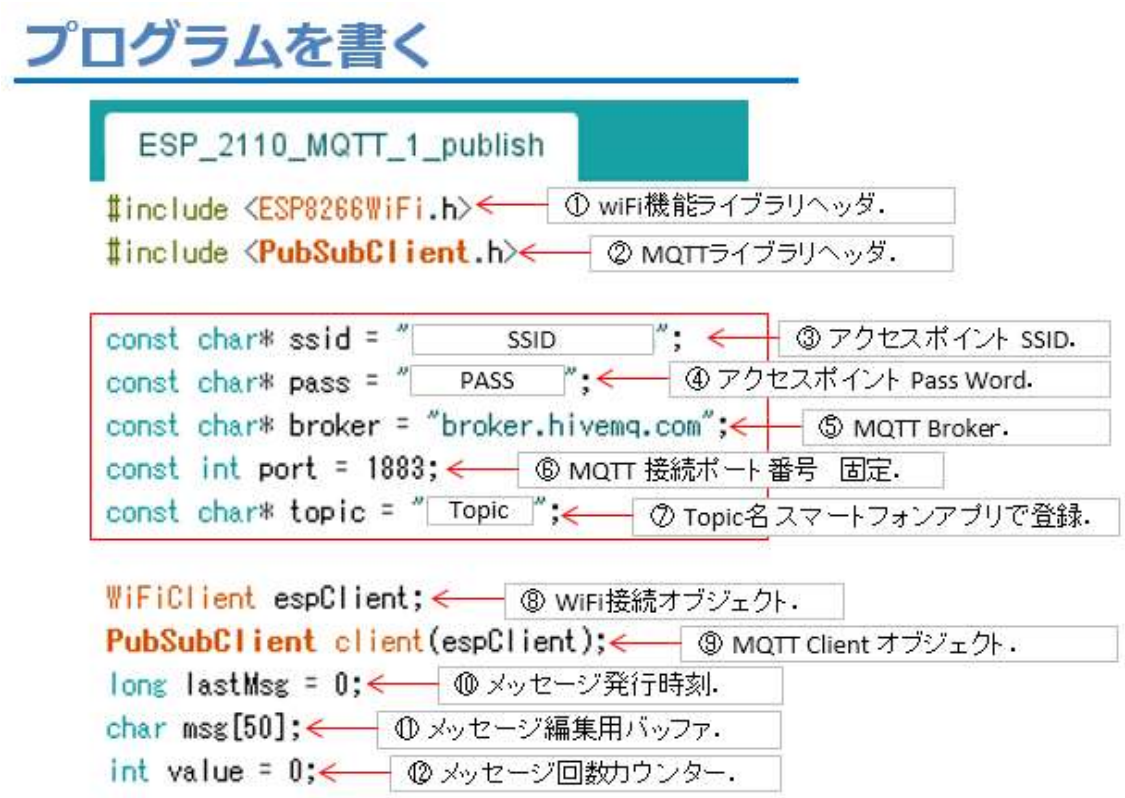


図 171

①WiFi 機能を使用するためのライブラリヘッダ

②MQTT 用ライブラリヘッダ

③アクセスポイント SSID を文字列で記述

④同 PassWord を記述

⑤使用する MQTT Broker

⑥ポート番号は固定で 1883.

⑦スマホアプリで設定した Topic 名を記述

※Topic はスラッシュ【/】で区切って階層構造化できます。また、ある階層以下を全部指定することも可能です。その時は、親の Topic 名に【/+】を付加します。

⑧アクセスポイント接続用 WiFi 機能オブジェクト

⑨MQTT 接続用オブジェクト

⑩メッセセージ発行時刻用変数

※マイコン内部のタイマーから ms（1/1000 秒）単位で読み込む。

⑪発行するメッセージ編集用バッファ

⑫メッセージ発行回数カウンタ

図　172

①シリアル通信 115200bps で初期化

②WiFi アクセスポイントに接続（関数内部は後に解説）

③MQTT Broker 接続情報を準備

プログラムを書く　loop()

```
void loop() {
  if (!client.connected()) {   ← ① MQTT 接続確認.
    reconnect();               ← ② MQTT 接続.
  }
  client.loop();               ← ③ MQTT 接続情報更新.

  long now = millis();         ← ④ マイコン現在時 取得.
  if (now - lastMsg > 10000) { ← ⑤ 10秒経過確認.
    lastMsg = now;             ← ⑥ メッセージ発行時刻更新.
    ++value;                   ← ⑦ メッセージ発行回数更新.
    snprintf(msg, sizeof(msg), "hello world #%ld", value);  ← ⑧ メッセージ編集.
    Serial.print("Publish message: ");                      ← ⑨ PCにメッセージ発行通知.
    Serial.println(msg);                                    ← ⑩ PCにメッセージ内容通知.
    client.publish(topic, msg);                             ← ⑪ メッセージ発行.
  }
}
```

図　173

①MQTT 接続状況確認。未接続であれば MQTT 接続

②MQTT 接続実行

③MQTT 接続状況とメッセージ発行状況の更新

※この関数内部でこのプログラムと MQTT Broker が情報交換。

loop()内で必ずこの関数を呼ぶ。

④マイコン内部のタイマーを読み込む

⑤前回から 10 秒経過したか

⑥処理時刻の記録を更新

⑦メッセージの発行回数を更新（＋１）

⑧メッセージ内容を編集して msg[] 配列に格納

⑨PC にシリアル通信で、メッセージ発行通知

⑩同、メッセージの内容通知

⑪MQTT Broker に実際のメッセージを【Topic】を付けて発行【Publish】

プログラムを書く　setup_wifi()

```
void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

① PCに状況通知.

② WiFi AP 接続実行.

③ WiFi AP 接続確認.

④ インジケータ.

⑤ PCに接続結果通知.

図　174

①シリアル通信で PC に WiFi 接続開始通知

②WiFi アクセスポイントに接続実行

③接続状況確認

④インジケータとして【.】ピリオド表示

⑤PC に接続結果と IP アドレス通知

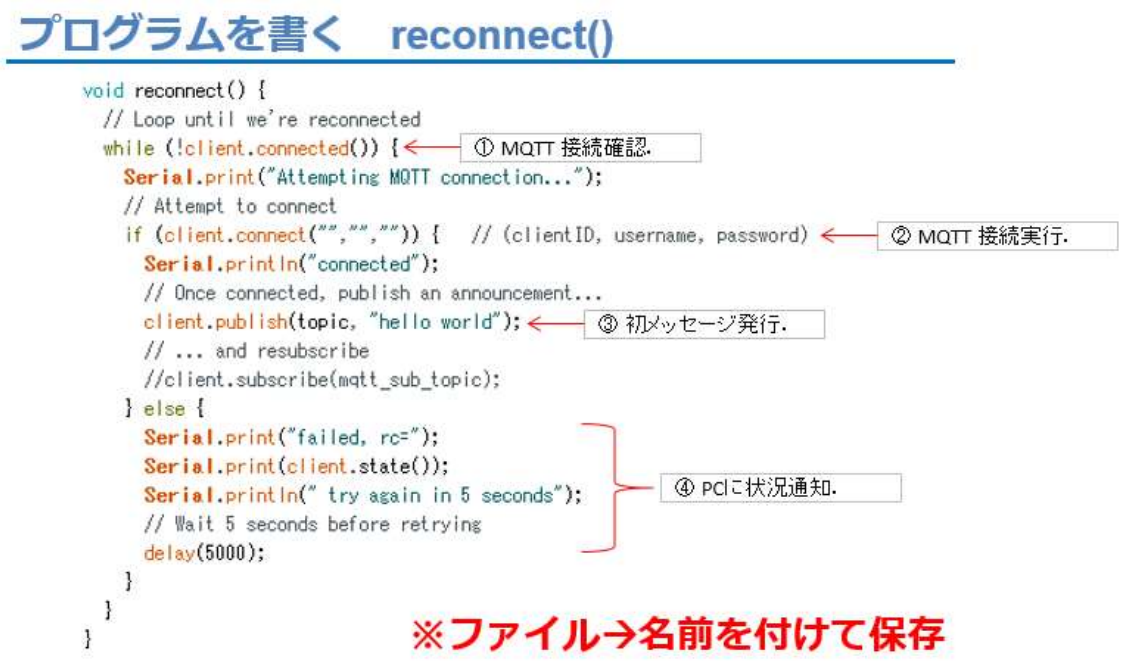


図　175

①MQTT 接続状況確認

②MQTT 接続実行

③接続ができた場合、初めてのメッセージを発行

④接続が巧く行かなかったとき、PC に状況を通知

この reconnect()は、MQTT PuSubClient ライブラリのサンプルプログラムの関数をそのまま流用しています。

ソースコードを入力したら、名前を付けて保存してください。

【重要】

このソースコードの一部は、今後の講座でも流用します。

必ず保存をしてください。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作確認

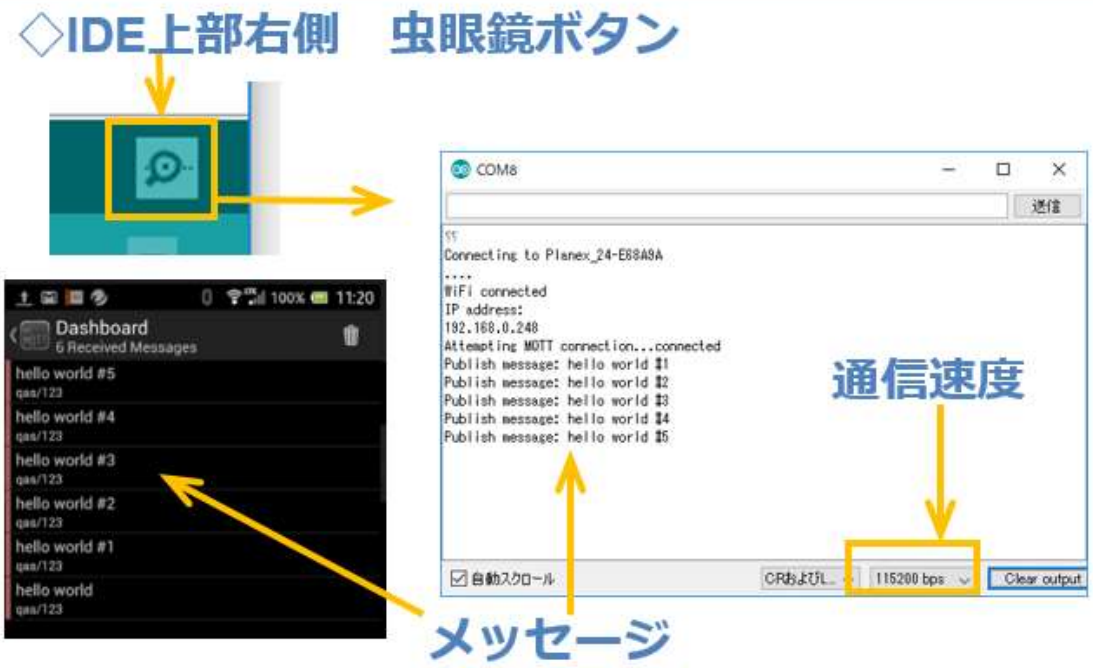


図　176

シリアルモニターを起動して、通信速度 115200bps に合わせ、携帯アプリも起動します。10 秒毎にシリアルモニターへのメッセージ表示と同期して、携帯画面にもメッセージが通知されます。

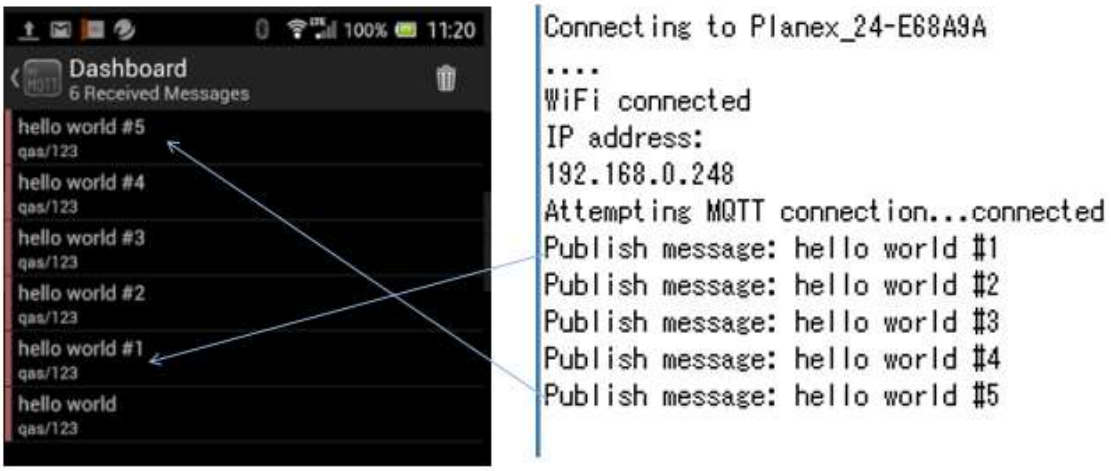


図　177

シリアルモニターでは、新しいメッセージは下に追加されて画面は上に移動して行きますが、携帯アプリの方は、上に最新のメッセージが積み重なるように表示されて、古いメッセージは下の方に移動します。Dashboardの一番上のメッセージに注意して、観察してください。

これで、遠隔地とフィールドを結ぶ、MQTTによるメッセージ交換システムが開発できました。

次は、すぐに利用できる温度通知システムの開発です。

第11回　WEB 連携②(MQTT)

まず次の図を見てください。第 10 回の冒頭で示した図の【多様なデバイス】部分に LCD とセンサー（デジタル温度センサー）を配置しています。

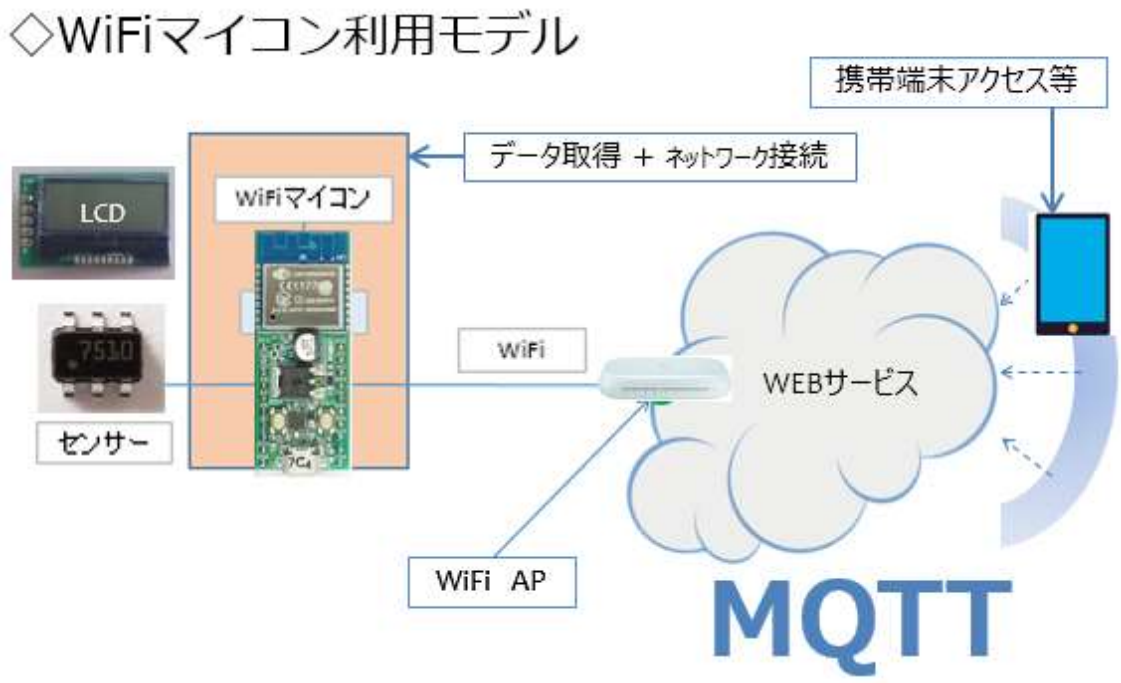


図　178

デジタル温度センサーで計測した温度を LCD に表示し、WEB サービス（MQTT）を通じて遠隔地に通知します。図から推測できると思いますが、今回は第 9 回で開発したデジタル温度計と第 10 回で開発したメッセージ通知システムの合体版になります。

この進め方は、温度センサーと液晶表示器を単独で制御できるようにした後、デジタル温度計を開発したのと同じ流れなので、今回の開発全体の流れも同様であることも類推できるでしょう。MQTT の仕組みは第 10 回を参照してください。

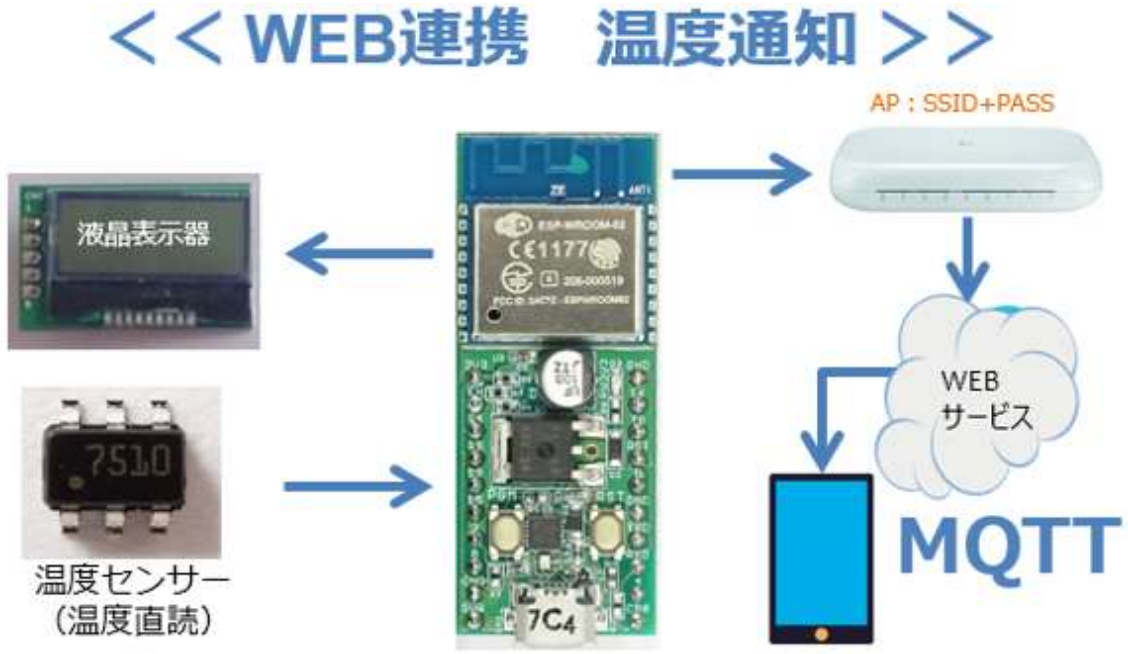


図 179

図を見て分かるように、具体的な機器類の関係でも、WiFi マイコン左側に【第 9 回：デジタル温度計システム】を、右側に【第 10 回：メッセージ通知システム】を配置したものです。それぞれのデバイス単体の制御や通信ができれば、それらを組み合わせながら、ステップ・アップでシステムを拡張できるという、分かり易い例となっています。

ここでは、携帯端末としては Android スマートフォンを利用します。またスマートフォンアプリとして MyMQTT を使用します。第 10 回で使用方法を説明しましたので、インストールして使いかたを練習しておいてください。

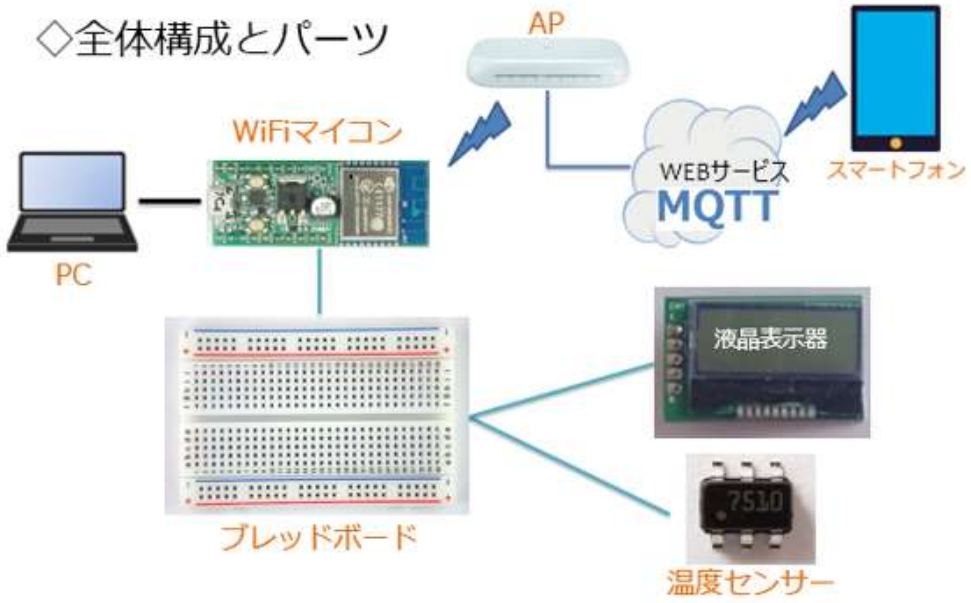


図 180

全体の構成は上の図の通りです。必要な機材・パーツは、下記です。

1. スマートフォン×1 台（解説は Android 携帯）
2. WiFi マイコン×1 台
3. PC（プログラム開発・書込）×1 台
4. USB ケーブル（マイコンとの接続）×1 本
5. ブレッドボード×2 個
6. 配線用ジャンパー線×適宜
7. LCD（液晶表示器）×1 個（AQM0802A）
8. デジタル温度センサー（STTS751）×1 個（前回のまま）

上記の機材・パーツは第 9 回デジタル温度計の開発に使用したものにスマートフォンが加わったものになっています。デジタル温度センサーは第 7 回で、LCD は第 8 回で解説をしましたので、該当部分を参照してください。

回路図を示します。第９回デジタル温度計の回路そのものです。

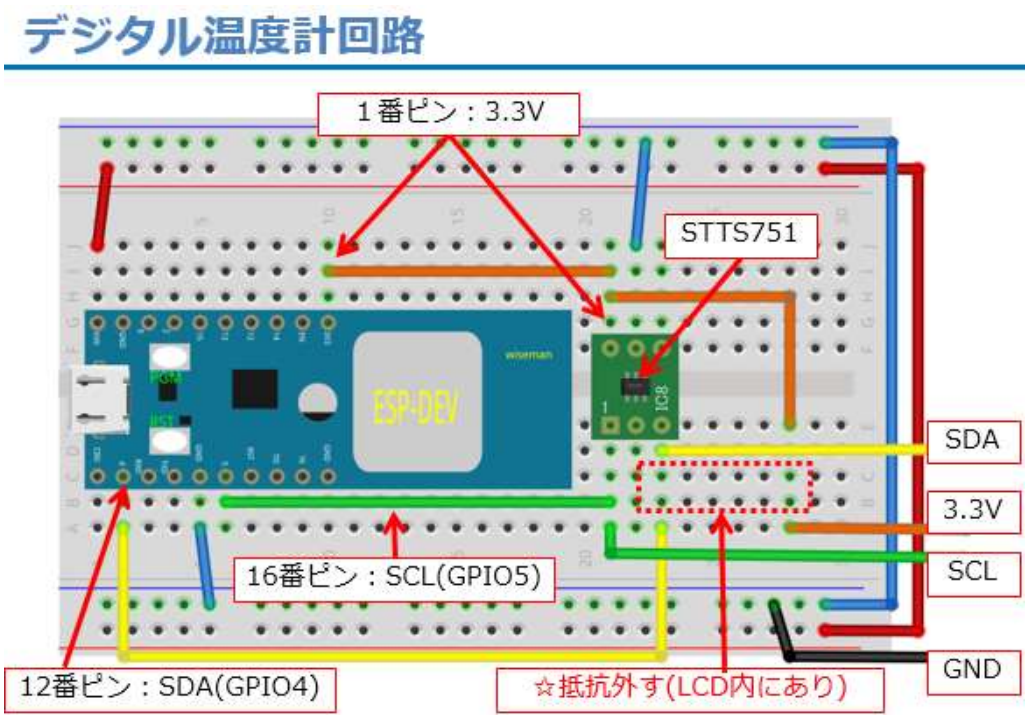


図　181

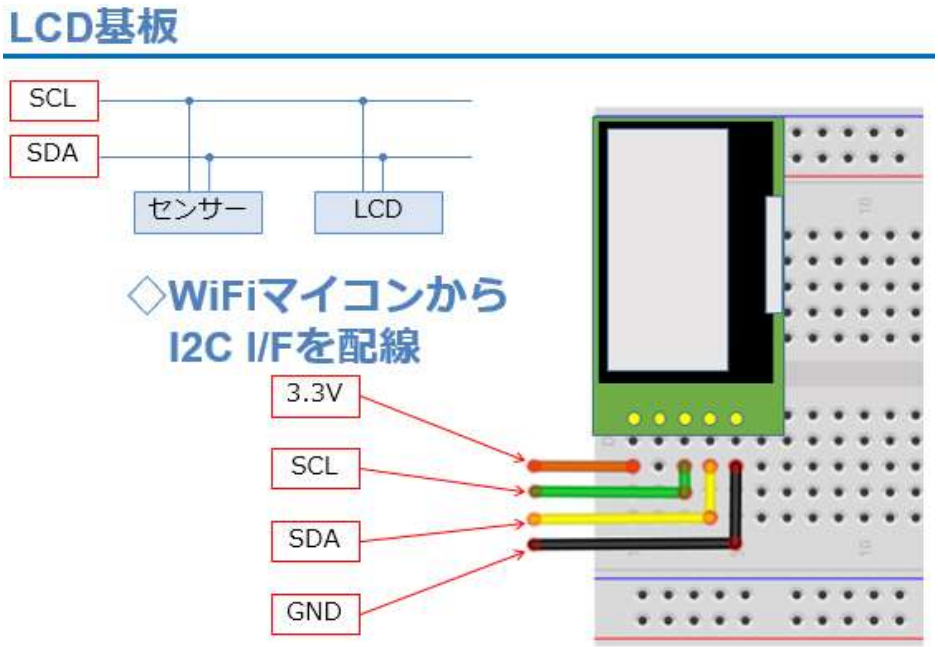


図　182

該当の回路が残っている方も、配線抜けやセンサー、LCD などの緩みが無いか、もう一度よく確認をして下さい。実際に配線したものは図のようになります。

実際に配線した様子

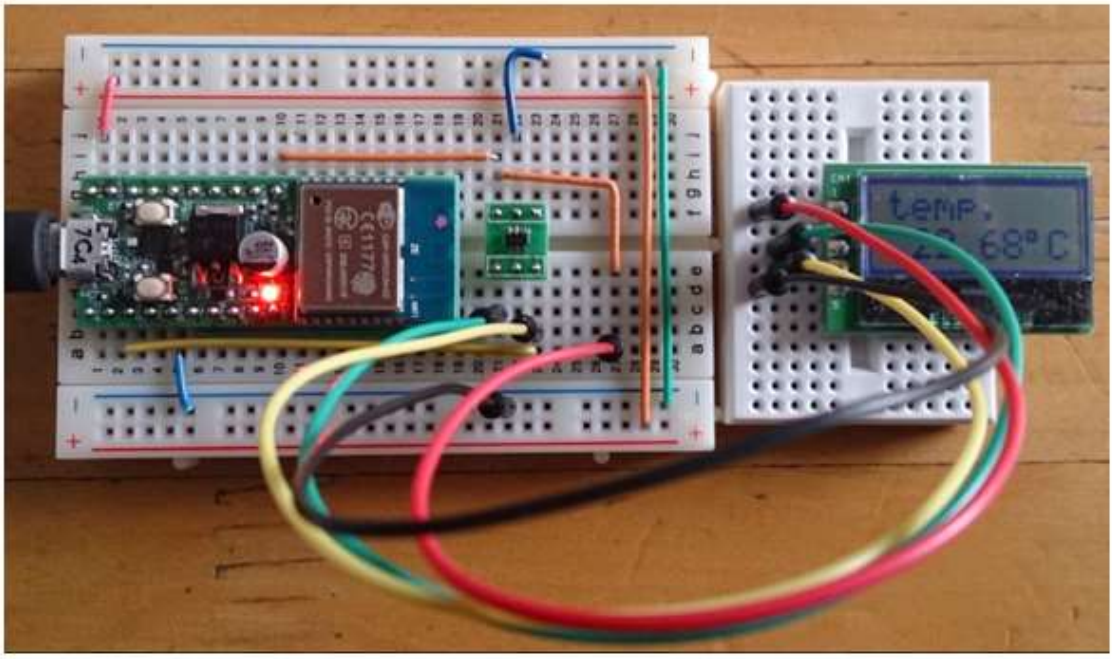


図 183

以後に示すソースコードは、【そのままの順番】で IDE に入力してください。これまでに、詳細に説明した部分は解説を省略します。これまでの講座の解説を参照してください。温度通知システム特有の部分について、説明を加えます。

プログラムを書く

ESP_2111_MQTT_2_ondo

```
#include <ESP8266WiFi.h>          ← ① WiFi機能ライブラリヘッダ.
#include <PubSubClient.h>         ← ② MQTTライブラリヘッダ.
#include <Wire.h>                 ← ③ I2Cデバイス通信 のライブラリヘッダ.
#define STTS751_ADRS  0x39        ← ④ 温度センサー スレーブアドレス.
#define LCD_ADRS      0x3E        ← ⑤ LCD スレーブアドレス.

//SCL=16:LCD No.3  SDA=12:LCD No.4

const char* ssid = "SSID";                          ← ⑥ アクセスポイント SSID.
const char* pass = "PASS";                          ← ⑦ アクセスポイント Pass Word.
const char* broker = "broker.hivemq.com";           ← ⑧ MQTT Broker.
const int port = 1883;                              ← ⑨ MQTT 接続 ポート番号 固定.
const char* topic = "Topic";                        ← ⑩ Topic名 スマートフォンアプリで登録.

char u_moji[]="temp.    ";                          ← ⑪ 表示文字列(上段用).
char l_moji[]=" **.** C";                           ← ⑫ 表示文字列(下段用).
long lastMsg = 0;                                   ← ⑬ メッセージ発行時刻.
char msg[50];                                       ← ⑭ メッセージ編集用バッファ.
int value = 0;                                      ← ⑮ メッセージ回数カウンター.

WiFiClient espClient;                               ← ⑯ WiFi接続オブジェクト.
PubSubClient client(espClient);                     ← ⑰ MQTT Client オブジェクト.
```

図　184

プログラムを書く　setup()

```
void setup() {
  Serial.begin(115200);  // start selial      ← ① シリアルポート初期化.
  Wire.begin();          // start i2c         ← ② I2C I/Fの初期化.
  init_STTS751();        // initialize sensor ← ③ 温度センサーの初期化.
  init_LCD();            // initialize lcd    ← ④ LCD初期化.
  init_wifi();           // initialize wifi   ← ⑤ WiFi AP 接続.
  client.setServer(broker, port);  // setting server inf. ← ⑥ MQTT Broker接続準備.
}
```

図　185

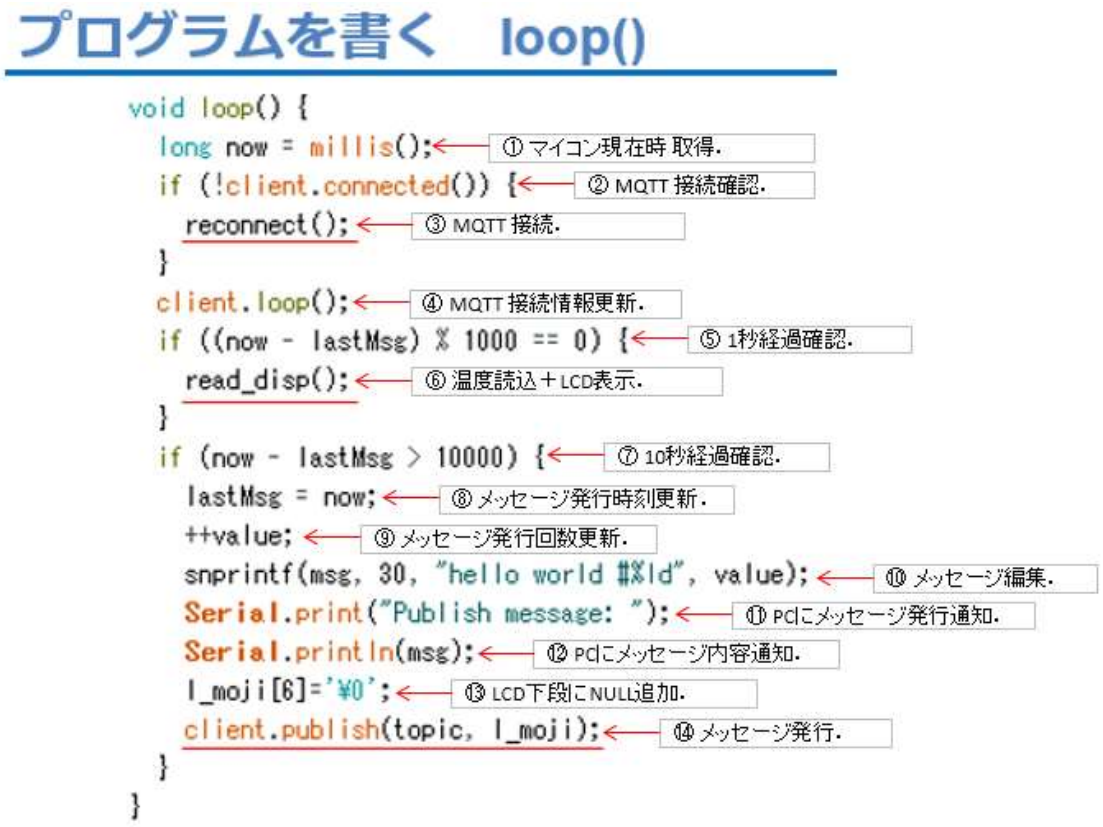


図　186

⑤⑥で、1秒ごとに温度をセンサーから読取り、PCに通知するとともにLCDに表示しています。

⑦で10秒経過したか判断しています。これは、あまり短い間隔で遠くにあるWEBサービスにメッセージを送っても、途中の通信回線の状況やサーバー処理の混雑具合などによって、スムースな通知ができない可能性があるので、適当な時間間隔をとっています。実験なので、10秒としましたが、実用システムでは、目的に合わせた時間間隔の検討が必要です。

⑬でLCDに温度を表示するためのバッファにNULL【'¥0'】を格納しているのは、このバッファを文字列として取り扱い、MQTT Brokerに【Publish】するためです。

プログラムを書く　init_wifi()

```
void init_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

① PCに状況通知.
② WiFi AP 接続実行.
③ WiFi AP 接続確認.
④ インジケータ.
⑤ PCに接続結果通知.

図　187

※上記 init_wifi()は、第 9 回の setup_wifi()と同じ内容です。

プログラムを書く　reconnect()

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("","","")) {   // (clientID, username, password)
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish(topic, "hello world");
      // ... and resubscribe
      //client.subscribe(mqtt_sub_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

① MQTT 接続確認.
② MQTT 接続実行.
③ 初ﾒｯｾｰｼﾞ発行.
④ PCに状況通知.

図　188

プログラムを書く　温度読込と表示

```
void read_disp() {
    byte valHigh,valLow;          ← ① 温度用変数.
    int temp;                     ← ② 小数部温度処理用変数.

    valHigh = readUpp();          // 整数部をセンサーから取得
    Serial.print(valHigh);        // 整数部を出力
    Serial.print(".");            // 小数点
    valLow = readLow();           // 少数部をセンサーから取得
    temp = (valLow >> 4) * 625;   // LSB:0.0625 小数部温度計算
    Serial.print(temp/1000);      // 0.1の位
    temp %=1000;
    Serial.print(temp/100);       // 0.01の位
    temp %=100;
    Serial.print(temp/10);        // 0.001の位
    temp %=10;
    Serial.println(temp);         // 0.0001の位
    sprintf(l_moji," %2d",valHigh);   // 整数部 ← ③ 整数部温度 LCDバッファ格納.
    l_moji[3]='.';                // 小数点
    temp = (valLow >> 4) * 625;   // LSB:0.0625 小数部温度計算
    l_moji[4]=(char)('0' + temp/1000); // 0.1の位
    temp %=1000;                                  ④ 小数部温度 LCDバッファ格納.
    l_moji[5]=(char)('0' + temp/100);  // 0.01の位
    l_moji[6]=0xDF;               // ℃の代わり
    l_moji[7]='C';

    writeCommand(0x80);           // 1行目 DRAM ADDR.
    for(int i=0; i<8; i++) {      // 表示
      writeData(u_moji[i]);
    }
    writeCommand(0x40+0x80);      // 2行目 DRAM ADDR.
    for(int i=0; i<8; i++) {      // 表示
      writeData(l_moji[i]);
    }
}
```

図　189

プログラムを書く　温度読込

```
byte readUpp() {
    // 整数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);   ← ① I2C通信開始.
    Wire.write(0x00);                       ← ② 温度整数部レジスタ指定.
    Wire.endTransmission();                 ← ③ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);      ← ④ I2C通信にて1byte読込リクエスト.
    return(Wire.read());                    ← ⑤ 温度(整数部)読込. 戻り値にセット.
}

byte readLow() {
    // 少数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);   ← ⑥ I2C通信開始.
    Wire.write(0x02);                       ← ⑦ 温度小数部レジスタ指定.
    Wire.endTransmission();                 ← ⑧ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);      ← ⑨ I2C通信にて1byte読込リクエスト.
    return(Wire.read());                    ← ⑩ 温度(少数部)読込. 戻り値にセット.
}
```

図　190

プログラムを書く　センサー初期化

```
void init_STTS751() {
    Wire.beginTransmission(STTS751_ADRS);   ← ① I2C通信開始.
    Wire.write(0x03);                       // config reg.  ← ② 03レジスタ指定.
    Wire.write(0b10001100);                 // EVENT停止、12bit分解能指定
    Wire.endTransmission();                 ← ③ 温度センサー初期設定.
}                                           ← ④ I2C通信終了.
```

図　191

プログラムを書く　文字送信・コマンド送信

```
void writeData(byte t_data) {
  Wire.beginTransmission(LCD_ADRS);
  Wire.write(0x40);
  Wire.write(t_data);
  Wire.endTransmission();
  delay(1);
}

void writeCommand(byte t_command){
  Wire.beginTransmission(LCD_ADRS);
  Wire.write(0x00);
  Wire.write(t_command);
  Wire.endTransmission();
  delay(10);
}
```

① I2C通信開始.
② 0x40送信.
③ 表示文字コード送信.
④ I2C通信終了.
⑤ I2C通信開始.
⑥ 0x00送信.
⑦ コマンドコード送信.
⑧ I2C通信終了.

図　192

プログラムを書く　LCD初期化

```
void init_LCD() {
  delay(100);
  writeCommand(0x38);  // Function set
  delay(20);
  writeCommand(0x39);  // Function set
  delay(20);
  writeCommand(0x14);  // OSC Freq. set
  delay(20);
  writeCommand(0x70);  // Contrast set
  delay(20);
  writeCommand(0x56);  // 3.3V, ICON,Contrast
  //writeCommand(0x52);  // 5V, ICON,Contrast
  delay(20);
  writeCommand(0x6C);  // Follwer Control
  delay(20);
  writeCommand(0x38);  // Function set
  delay(20);
  writeCommand(0x01);  // Clear Display
  delay(20);
  writeCommand(0x0C);  // Display ON/OFF control
  delay(20);
}
```

※ファイル→名前を付けて保存

図　193

以上がソースコードです。これまでに開発してきたプログラムの流用が多く、新しい部分はそれほどありません。入力が終わりましたら、名前を付けて保存しましょう。以下の手順は、これまでと同じです。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

図 194

動作確認です。まず、LCD に図の様に整数部、小数部、℃の表示が行われます。シリアルモニターの通信速度を合わせ、携帯端末のアプリも起動してください。

図　195

いかがでしょうか。1秒ごとにLCDの温度表示が更新されて、10秒ごとに小数点以下2桁までの温度が、メッセージとして配信されることが確認できましたか。

このシステムをUSBケーブルで5Vを供給して部屋に置き、外出先から携帯端末で現在の温度はどうなっているのか、メッセージを購読【Subscribe】してみましょう。

実際の応用も難しくないですね。アイデア次第です。ぜひ考えてみましょう。

第12回　WEB 連携③(MQTT)

今回は、WEB サービスを通じて SW の状態を通知するモデルです。次の図をご覧ください。

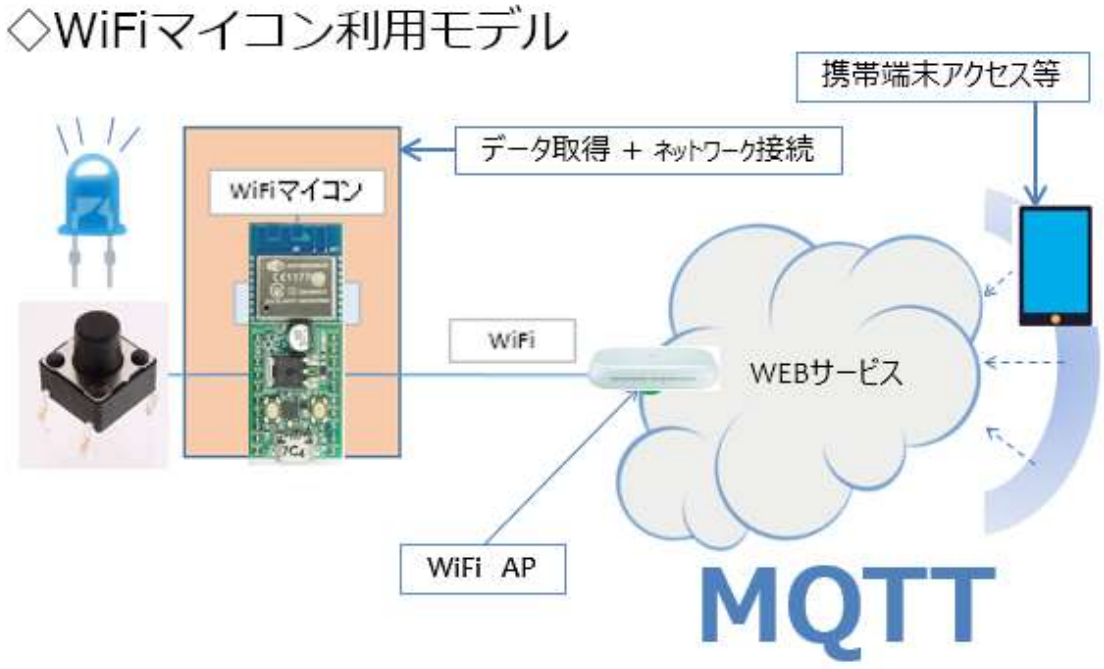


図　196

第 11 回に WiFi マイコンの外部デバイスとして使用していたデジタル温度センサーに代えて SW を使用します。また LCD に代えて LED を使います。SW の状態に応じて LED の点灯・消灯を制御しながら、状態が変化したときだけ、MQTT サービスを利用してメッセージを発行するシステムを作ります。このモデルは、お年寄りの【見守りシステム】などで既に実用化されているものと同様です。

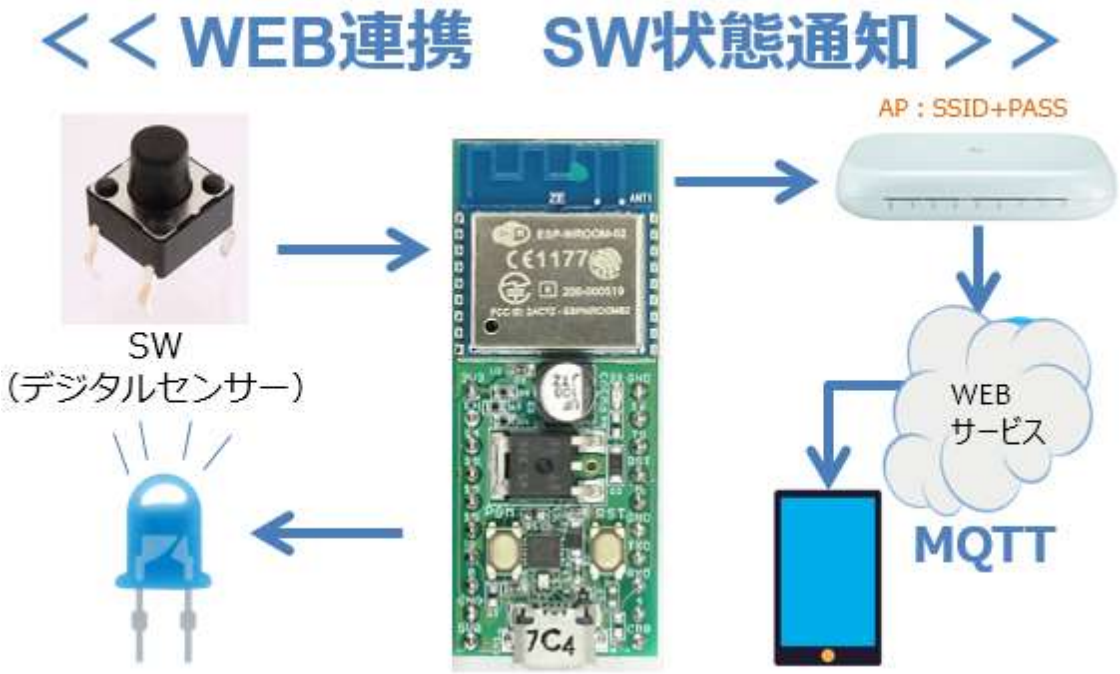


図　197

図に記載しましたが、SW は ON/OFF 状態を検出する【デジタルセンサー】だと考えれば、前回のシステムと同じものとして理解できます。文字情報表示器と LED が入れ替わり、ON/OFF 状態を表示していると考えます。前回は【温度】をメッセージとして発行していましたが、今回は【SW の ON/OFF 状態が変化した】という内容のメッセージを通知します。それだけでは、あまりに簡単ですから、SW の状態が変化したときだけメッセージを発行することにしましょう。

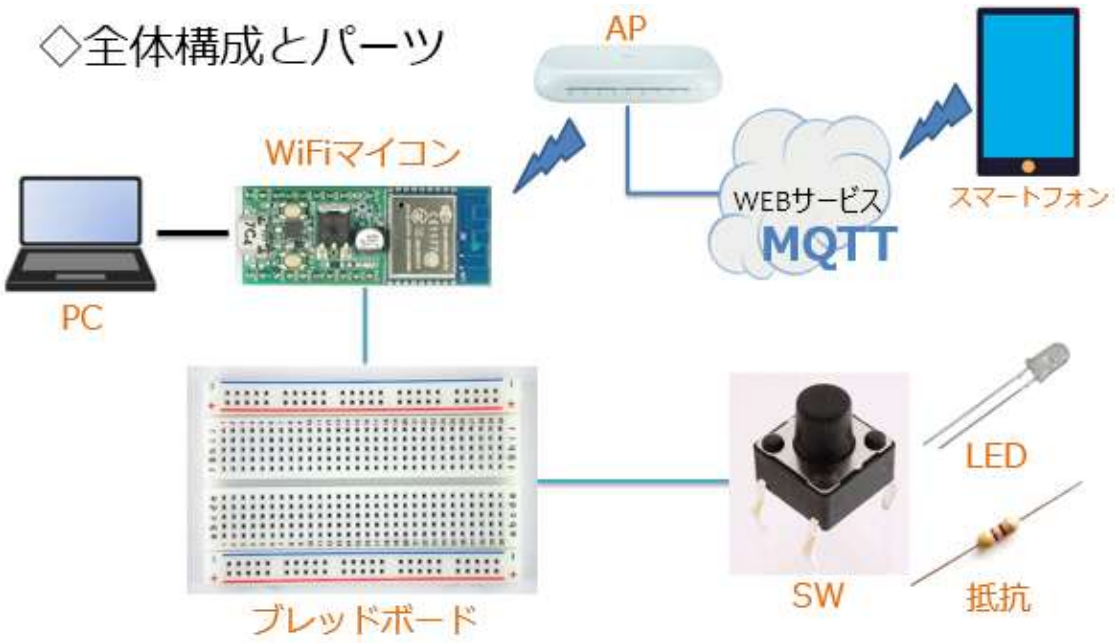


図 198

全体の構成は上図の通りです。必要な機材・パーツを下記に示します。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個
8. SW×1 個

上記の機材・パーツは第 2 回の開発に使用したものと同じです。回路も第 2 回と同じです。配線図を次に示します。

SW・LED点灯回路

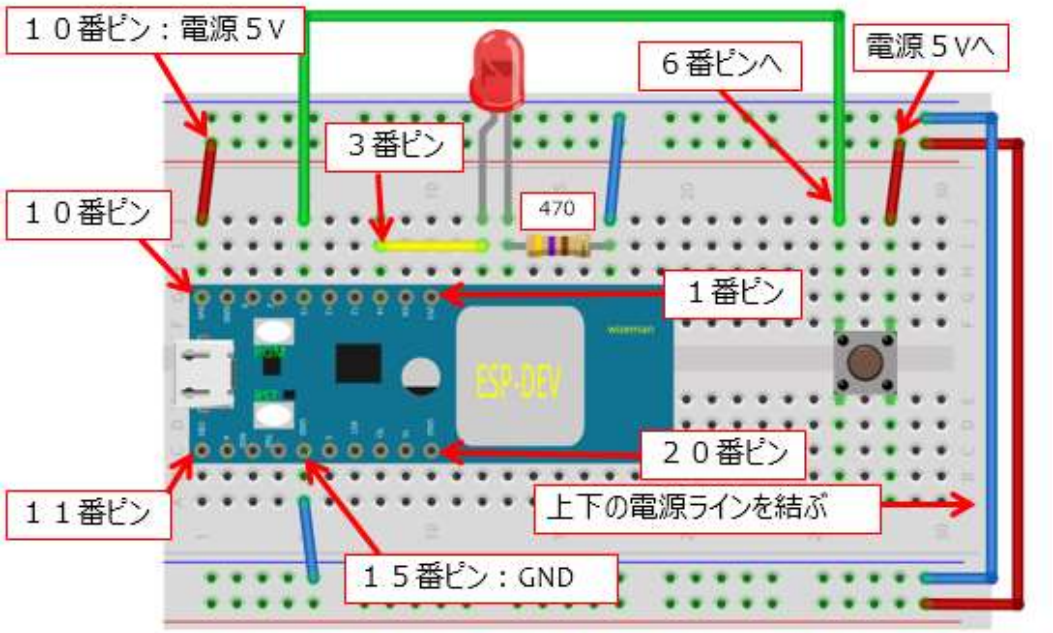


図　199

新たに配線を行う場合は、LEDの向き（脚の長さの違い）に気を付けましょう。完成すると下図の様になります。

実際に配線した様子

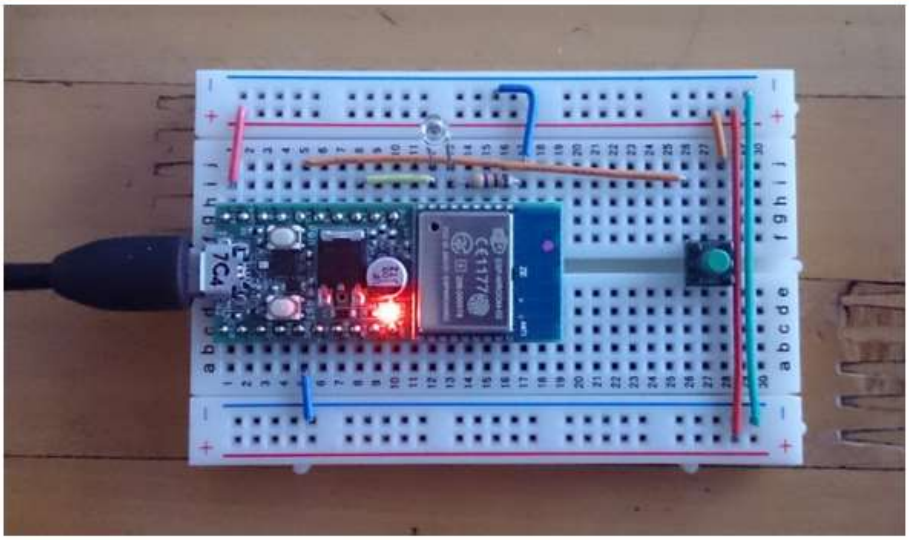


図　200

プログラムを作りましょう。以後に示すソースコードは、そのままの順番で、IDE に入力してください。既に詳細に説明した部分は解説を省略しています。これまでの講座の解説を参照してください。

プログラムを書く

ESP_2112_MQTT_3_SW

```
#include <ESP8266WiFi.h>          ← ① wiFi機能ライブラリヘッダ.
#include <PubSubClient.h>         ← ② MQTTライブラリヘッダ.
#define LED_PIN 14 //<--- GPIO14 LED   ← ③ LEDをGPIO14に接続した.
#define SW_PIN 15  //<--- GPIO15 SW PULL DOWN 10K
                                  ← ④ SWをGPIO15に接続した.
const char* ssid = "SSID";        ← ⑤ アクセスポイント SSID.
const char* pass = "PASS";        ← ⑥ アクセスポイント Pass Word.
const char* broker = "broker.hivemq.com";   ← ⑦ MQTT Broker.
const int port = 1883;            ← ⑧ MQTT 接続ポート番号 固定.
const char* topic = "Topic";      ← ⑨ Topic名 スマートフォンアプリで登録.
```

図 201

プログラムを書く

```
WiFiClient espClient;                  ← ① WiFi接続オブジェクト.
PubSubClient client(espClient);        ← ② MQTT Client オブジェクト.
int n = 0;                             ← ③ メッセージ発行回数カウンタ.
int s = 0;                             ← ⑩ 現在のSW状態.
int sw = 0;                            ← ⑪ 以前のSW状態.
char msg[50];                          ← ⑫ メッセージ編集用バッファ.
```

図 202

プログラムを書く　setup()

```
void setup() {
  pinMode(LED_PIN,OUTPUT);   ← ① GPIO14を出力に設定.
  pinMode(SW_PIN,INPUT);     ← ② GPIO15を入力に設定.
  Serial.begin(115200);      ← ③ シリアルポート 初期化.
  init_wifi();               ← ④ WiFi AP 接続.
  client.setServer(broker, port);  ← ⑤ MQTT Broker接続準備.
}
```

図　203

プログラムを書く　loop()

```
void loop() {
  if (!client.connected()) {   ← ① MQTT 接続確認.
    reconnect();               ← ② MQTT 接続.
  }
  client.loop();               ← ③ MQTT 接続情報更新.
  s = digitalRead(SW_PIN);     ← ④ 現在のSW状態読込.
  digitalWrite(LED_PIN, s);    ← ⑤ SW状態をLEDに反映.
  if(s != sw) {                ← ⑥ SW状態変化したか？.
    ++n;                       ← ⑥ メッセージカウンタ更新.
    switch(s) {
      case 1:  // OFF--->ON     ← ⑦ SW ONした.
        snprintf(msg, sizeof(msg), "changed!! OFF--->ON #%d", n);
        break;                 ← ⑧ メッセージ編集.
      case 0:  // ON--->OFF     ← ⑨ SW OFFした.
        snprintf(msg, sizeof(msg), "changed!! ON--->OFF #%d", n);
        break;                 ← ⑩ メッセージ編集.
    }
    sw = s;                    ← ⑪ SW状態更新.
    Serial.print("Publish message: ");   ← ⑫ PCに通知.
    Serial.println(msg);       ← ⑬ PCにメッセージ通知.
    client.publish(topic, msg);  ← ⑭ メッセージ発行.
  }
}
```

図　204

プログラムを書く　init_wifi()

```
void init_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

① PCに状況通知.
② WiFi AP 接続実行.
③ WiFi AP 接続確認.
④ インジケータ.
⑤ PCに接続結果通知.

図　205

プログラムを書く　reconnect()

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("","","")) {   // (clientID, username, password)
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish(topic, "hello world");
      // ... and resubscribe
      //client.subscribe(mqtt_sub_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

① MQTT 接続確認.
② MQTT 接続実行.
③ 初メッセージ発行.
④ PCに状況通知.

図　206

以上がソースコードです。これまでに開発してきたプログラムの流用が多く、新しい部分はほんの少しです。入力が終わりましたら、名前を付けて保存しましょう。以下の手順は、これまでと同じです。

ここから、WiFiマイコンへの書込みまで一気に進めましょう。

以下の手順でプログラムのコンパイルからWiFiマイコンへの書き込みを行います。

①PCとWiFiマイコンの接続：USBケーブルで接続

②シリアルポート（COM番号）の確認：デバイスマネージャで確認

③シリアルポート（COM番号）の設定：IDEで設定

④WiFiマイコンへの書込み準備：WiFiマイコンのSW操作

⑤スケッチ（プログラム）のコンパイルから書込み：IDEで操作

書込みが終了すると書込み完了のメッセージがIDEに表示される

※上記の手順についてはすでに身に付いていることと思います。不安がある方は、これまでの解説を読みながら、【正確】に手順をトレースしてください。手順を誤ると巧くゆきません。私の経験では①から③は、大丈夫ですが、④を時々忘れます。IDEの下部に、マイコンへの書込みが失敗した旨のメッセージが表示されても、驚くことなくWiFiマイコンのSW操作を行い、もう一度IDEの右向き矢印ボタンをクリックしてください。

書込みが終了すると、WiFiマイコンにResetが掛かり、プログラムの実行が始まります。

動作の様子

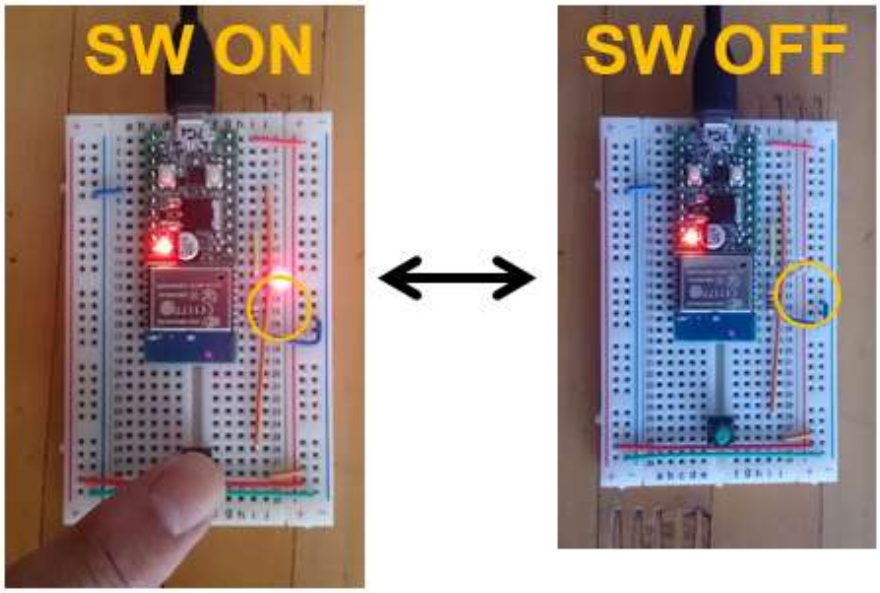


図 207

動作確認をします。まず、SW ON で LED が点灯し、SW OFF で LED が消灯することを確認しましょう。次にシリアルモニターを起動して、通信速度を合わせます。（下図）

この状態で、SW の ON/OFF に対応して、メッセージが表示されるでしょうか。また、SW を ON したまま、あるいは OFF したままの状態では、メッセージが表示されないことも、確認しておきましょう。

図　208

続いて、携帯端末でメッセージ購読【Subscribe】用のアプリケーションを起動しましょう。MQTT Broker に接続してメッセージ講読状態にしておきます。ここで、SW を ON/OFF したとき、携帯端末のアプリ画面に購読【Subscribe】したメッセージが表示されます。

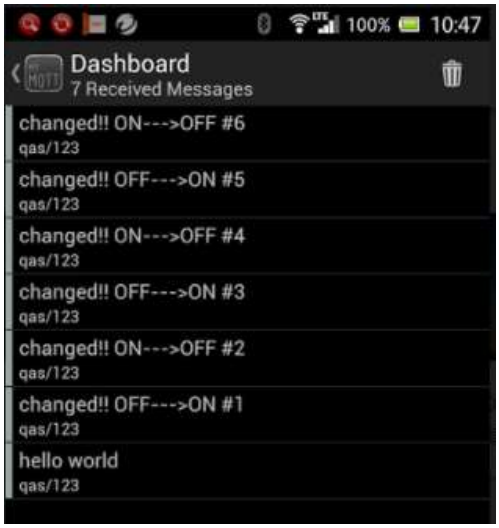


図　209

第13回　WEB 連携④(Ambient)

これまで利用してきた WEB サービスは、メッセージ通知に特化したものでした。計測した温度をメッセージとして WEB に送り、遠隔地でそれを観察するシステムを第 11 回で開発しました。そのメッセージは数字として読むことができましたが、そのような変化するデータがグラフとして見えると、とても便利ですね。今回は、WEB にメッセージを送るだけで、遠隔地からグラフとして観察できる WEB サービスを使ってみます。(下図)

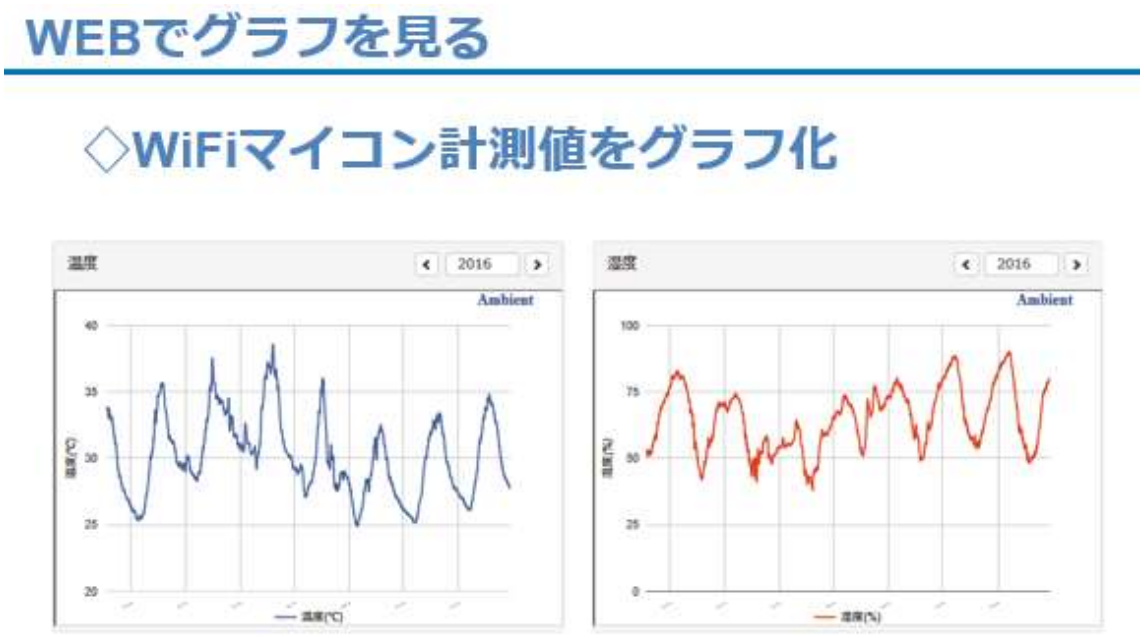


図　210

今回使用するのは、Ambient という WEB サービスです。上の図は、温度と湿度を同時に送って、外部からブラウザで観察した様子です。

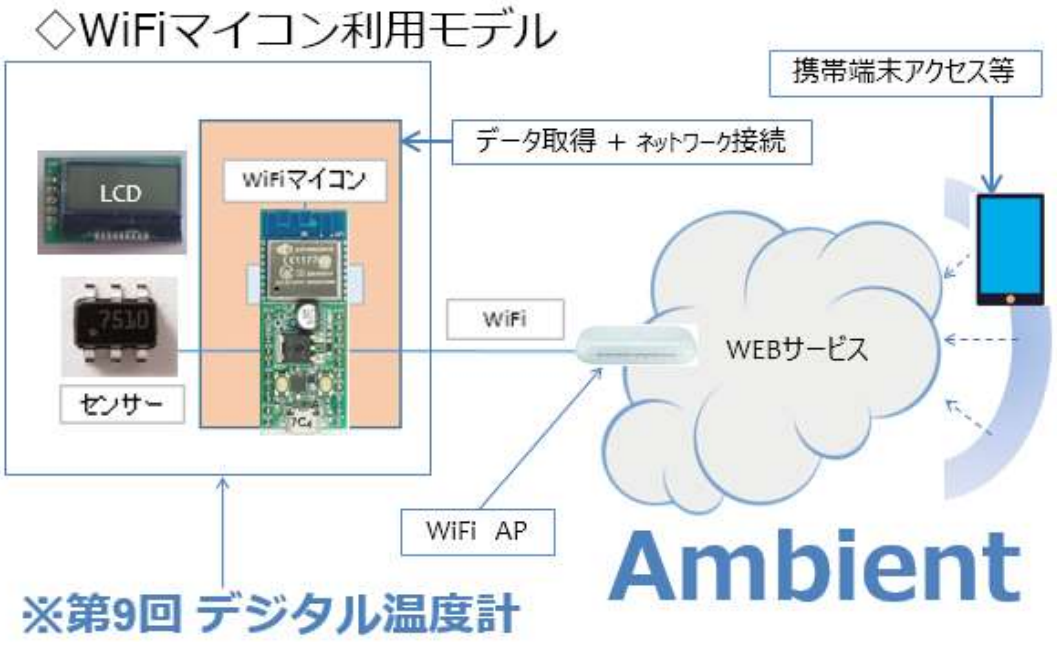


図 211

今回のモデルは、第 11 回で利用した WEB サービスを Ambient に変更しただけのモデルです。次に示す全体像について説明は省略します。

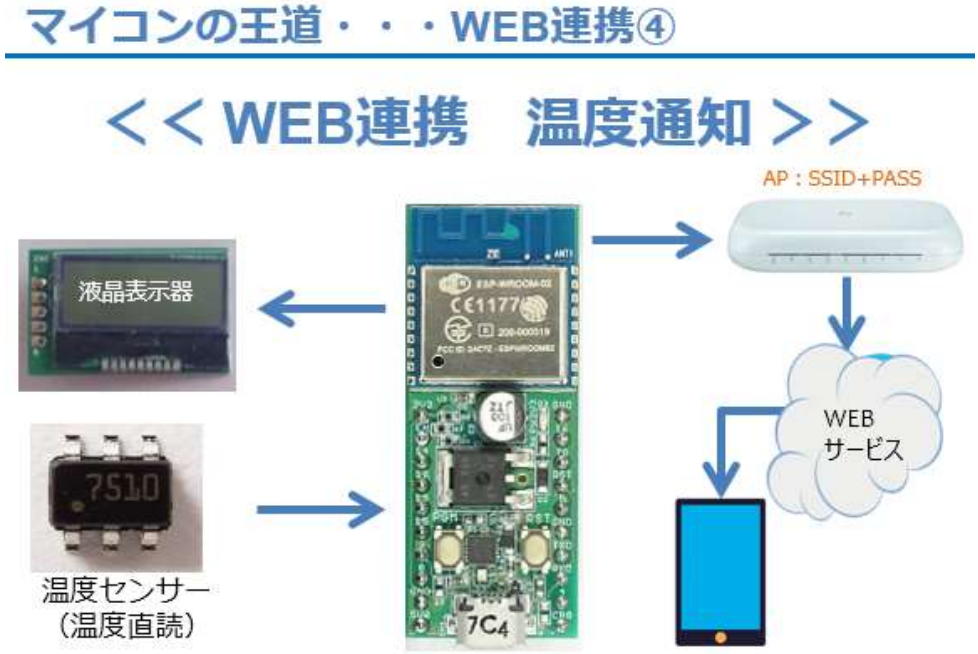


図 212

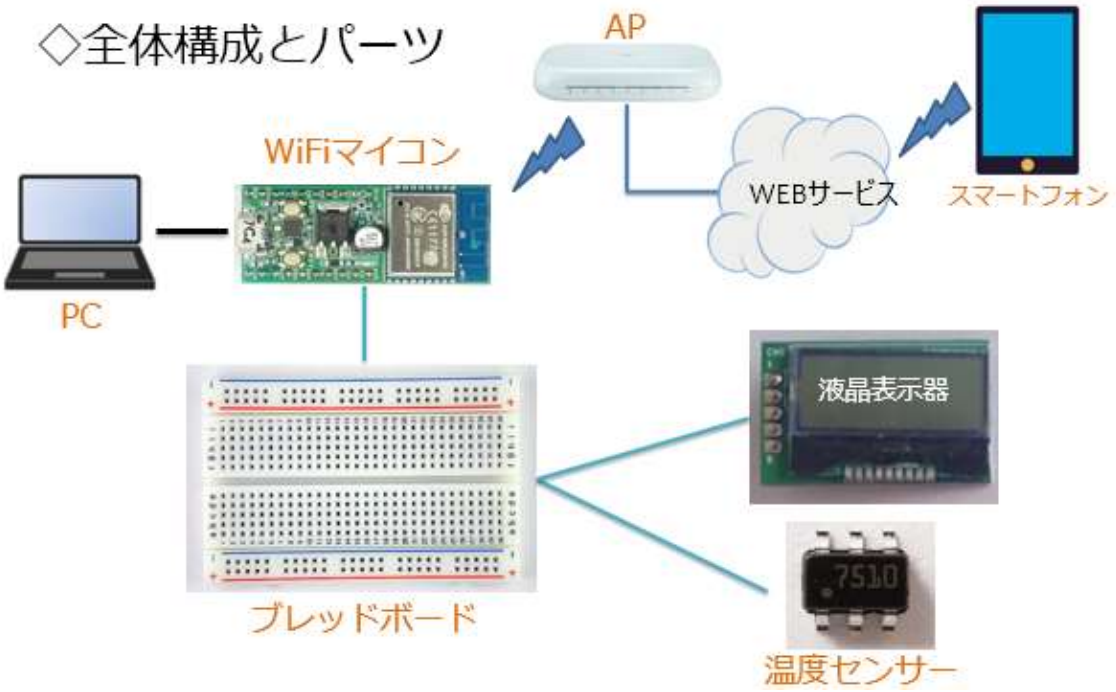


図 213

全体の構成は上の図の通りです。必要な機材・パーツは、下記です。

1. スマートフォン×1 台（解説は Android 携帯）
2. WiFi マイコン×1 台
3. PC（プログラム開発・書込）×1 台
4. USB ケーブル（マイコンとの接続）×1 本
5. ブレッドボード×2 個
6. 配線用ジャンパー線×適宜
7. LCD（液晶表示器）×1 個（AQM0802A）
8. デジタル温度センサー（STTS751）×1 個（前回のまま）

上記の機材・パーツは第 9 回デジタル温度計の開発に使用したものにスマートフォンが加わったものになっています。デジタル温度センサーは第 7 回で、LCD は第 8 回で解説をしましたので、該当部分を参照してください。

図 214

今回利用する Ambient は、チュートリアルがとても充実しています。私もこのシステムを開発する際、大いに参考にさせていただきました。

下図の範囲であれば無料で使用できます。

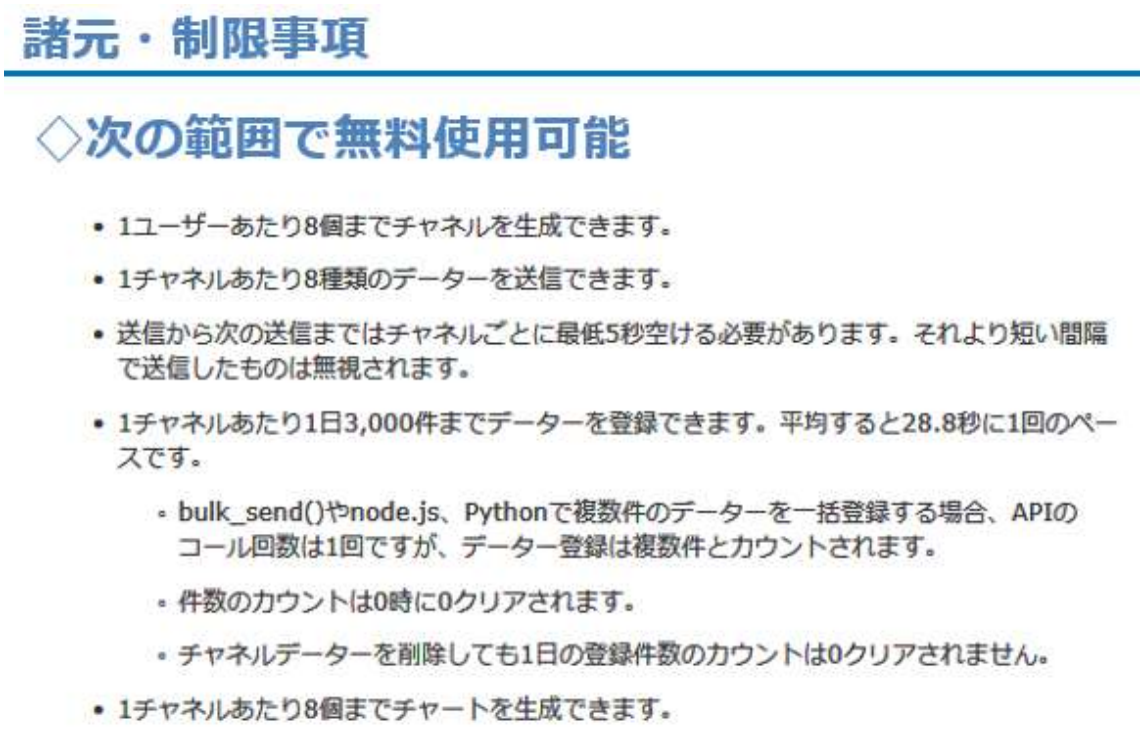


図 215

Ambient に送ったデータは、チャネル単位で管理されます。1 チャネルで 8 つのデータを送れますので、フルに使うと 8×8 = 64 個のグラフを観察することができます。温度などのデータは、数分に 1 回程度の保存で十分ですから、上記の無料で使用できる範囲で十分です。

図　216

Ambient を利用するために、先に示した WEB ページにアクセスして、ユーザ登録をします。必要な情報を入力してボタンを押します。

図　217

図 218

入力したアドレス宛にメールが届きますので、その指示に従い登録を完了します。URL にアクセスするだけですので簡単です。

図 219

登録が完了したら、早速ログインします。ログイン後、My チャネルのページに移動します。【チャネルを作る】ボタンをクリックして、チャネルを作ります。

【重要】
この時、上の図の様に、【チャネル ID】、【リードキー】、【ライトキー】が発行されますので、メモをしておいてください。この情報をソースコードに記述することにより、WEB 上の自分のチャネルにマイコンからアクセスしますので、メモを無くさない様にして下さい。

図 220

作成したチャネル宛に WiFi マイコンでデータを送り、携帯端末や PC のブラウザで My チャネルにアクセスすると、何も設定しない状態で、上の図の様なグラフが表示されます。もちろん、細かな設定を行うことで、希望の様式にすることができます。

以下に、回路図を示します。回路は第 11 回で作成したものと同じです。

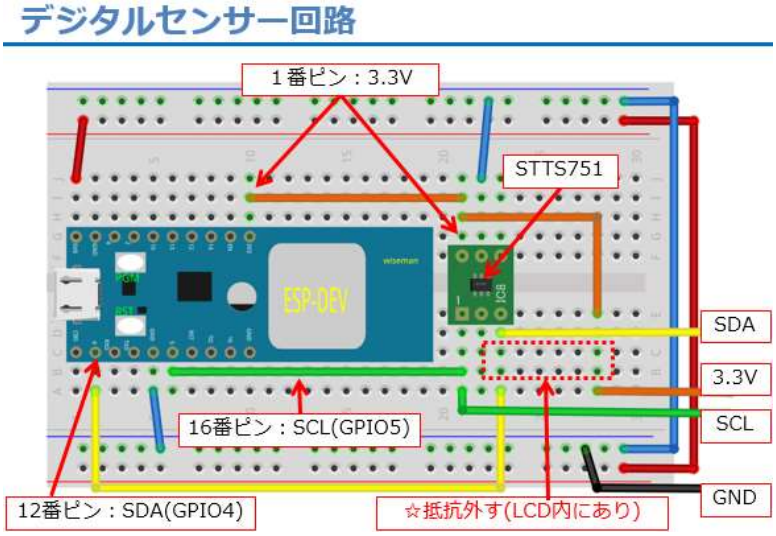


図　221

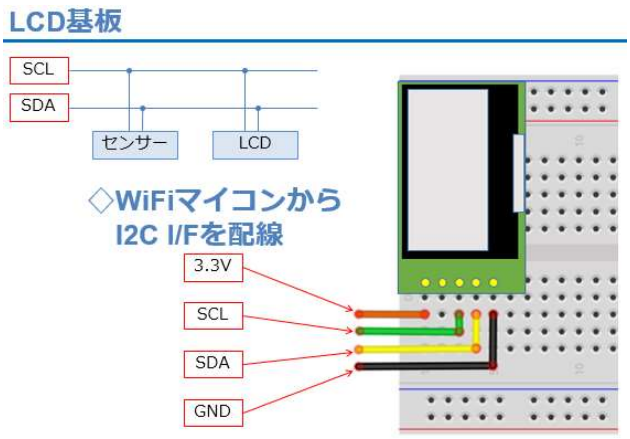


図　222

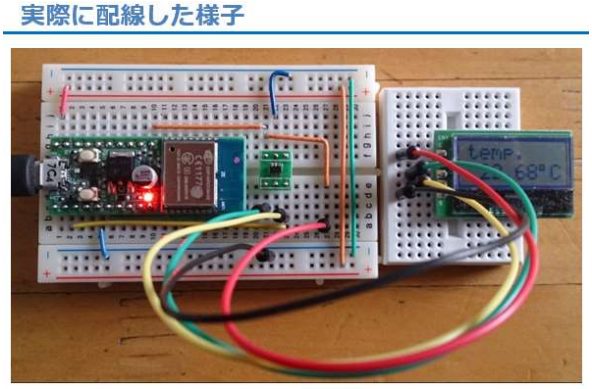


図　223

Ambientライブラリの準備

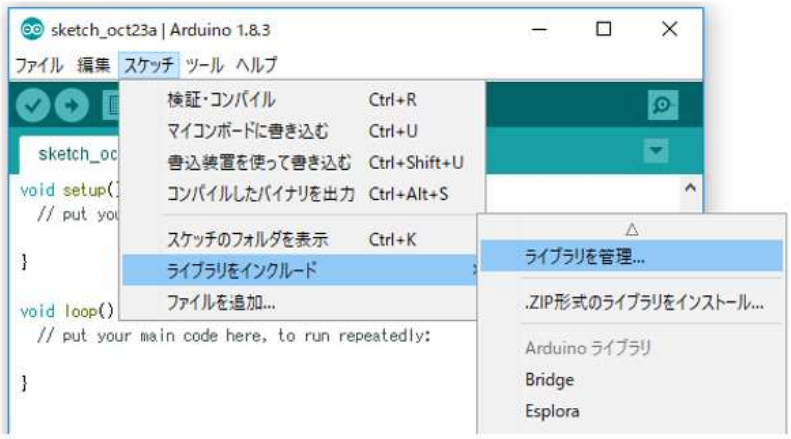


図　224

Ambient を WiFi マイコンで使うためには、ライブラリが必要ですので、ここで準備をします。上の図に従い、【スケッチ→ライブラリをインクルード→ライブラリを管理】と辿り、ライブラリマネージャを起動してください。

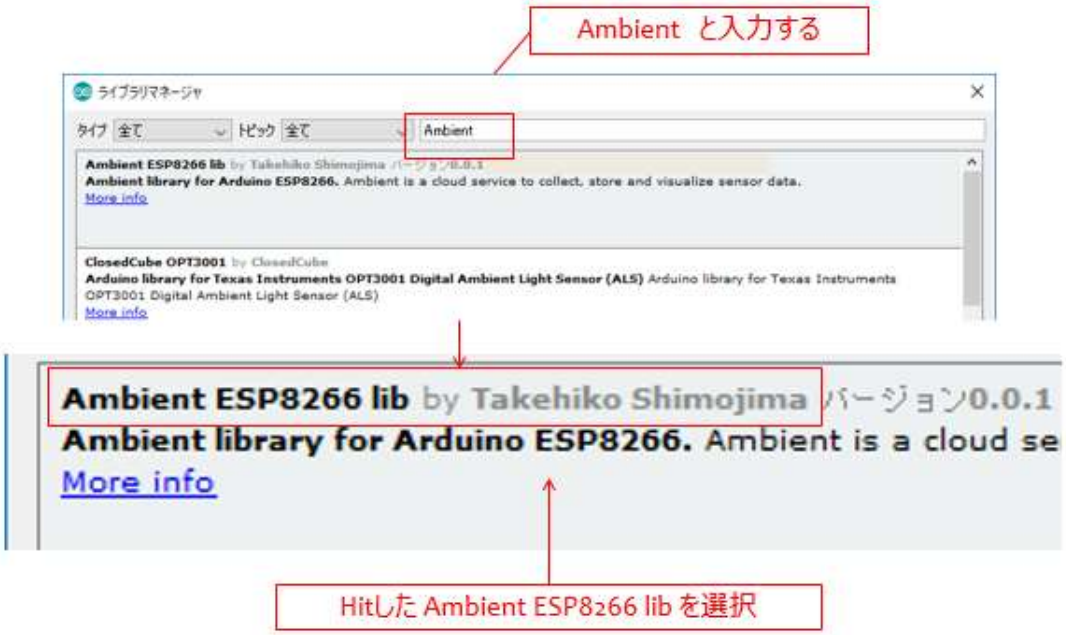


図　225

ライブラリマネージャの上部に Ambient と入力すると、使用している WiFi マイコン用のライブラリ【Ambient ESP8266 lib】がヒットします。

図 226

ヒットした【Ambient ESP8266 lib】を選択（中央部分をクリック）して、インストールボタンをクリックして下さい。しばらくすると、ライブラリ上部に INSTALLED と表示されます。

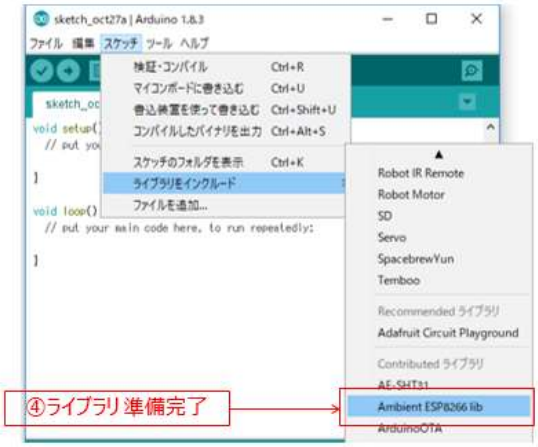


図 227

上の図に従い、【Ambient ESP8266 lib】が一覧に表示されれば、ライブラリの準備は完了です。

以下にソースコードを示します。全体としては、これまでに出てきたソースコードとほとんど同じですので、ここまで進んだ皆さんには、説明は不要だと思いますが、Ambient 特有の部分について、説明を付加しておきます。ソースコードは、下記の順番のとおりに IDE に入力してください。

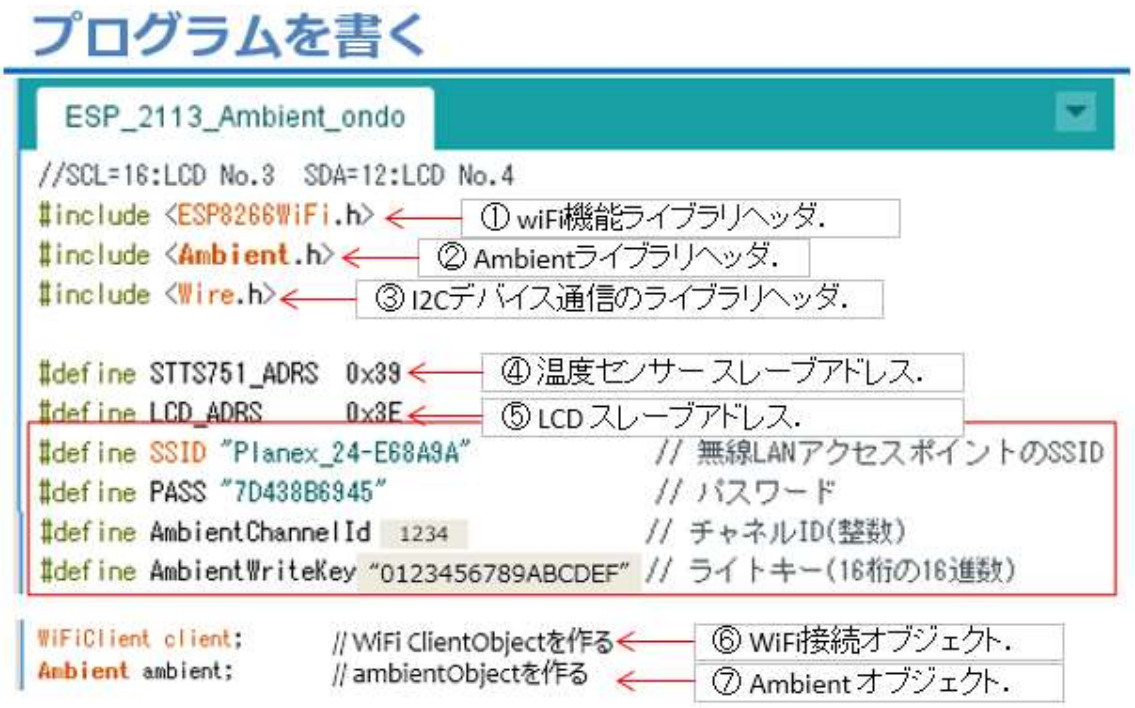


図 228

②Ambient ライブラリを利用するため、ヘッダを取り込む.

⑦Ambient オブジェクト作成.

【重要】

赤枠内の AmbientChannelID と AmbientWriteKey は Ambient のチャネルを作成した際に発行されたものです。

プログラムを書く　setup()

```
char u_moji[]="temp.    ";   // LCD用バッファ
char l_moji[]=" **.** C";    // LCD用バッファ
long lastMsg = 0;            // メッセージ表示間隔用カウンタ
char msg[50];                // メッセージ用バッファ
int value = 0;               // 発行回数用カウンタ

void setup() {
  Serial.begin(115200);  // start selial
  Wire.begin();          // start i2c
  init_STTS751();        // initialize sensor
  init_LCD();            // initialize lcd
  init_wifi();           // initialize wifi
  ambient.begin(AmbientChannelId, AmbientWriteKey, &client);  // Ambient開始
}
```

① Ambient開始.

図　229

①Ambient を初期化・開始します.

プログラムを書く　loop()

```
void loop() {
  long now = millis();
  if ((now - lastMsg) % 1000 == 0) {
    read_disp();
  }
  if (now - lastMsg > 10000) {
    lastMsg = now;
    ++value;
    snprintf(msg, 30, " %ld", value);
    Serial.print("send to Ambient: ");
    Serial.println(msg);
    l_moji[6]='¥0';

    /* クラウドへ */
    ambient.set(1,l_moji);
    ambient.set(2,l_moji);
    ambient.send();
  }
}
```

① Ambientデータ準備.

// Ambient(データ1)へ温度を送信

// Ambient(データ2)へ温度を送信

// Ambient送信の終了(実際に送信する)

② Ambientに送信.

図　230

①Ambient に送るデータを準備します。

【重要】

送るデータは、単位を付けないで文字列として準備します。

同じデータを 2 つ準備している訳は、複数のデータを送る場合に、どのように送ればよいかを示すためです。パラメータにシーケンス番号を入れるとその順番にグラフが描かれます。

②実際に Ambient に送信します.

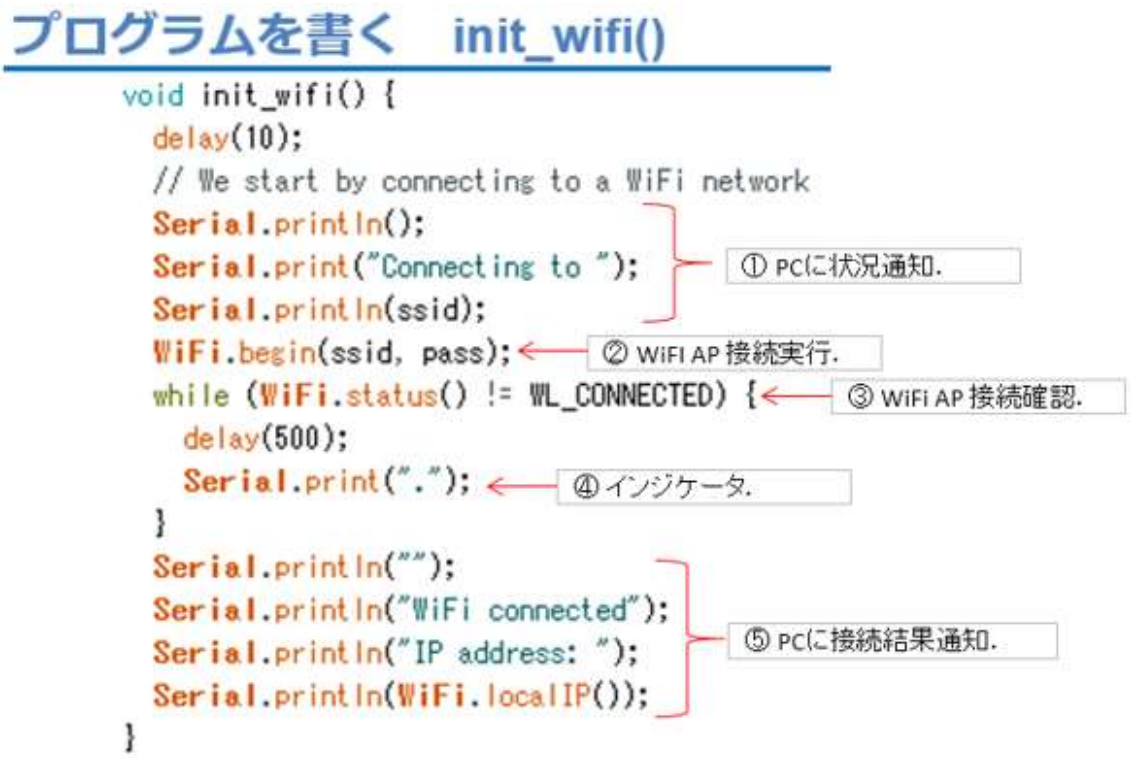


図　231

この WiFi 初期化の部分は、これまでに使ったコードです。

プログラムを書く　温度読込と表示

```
void read_disp() {
    byte valHigh,valLow;          ← ① 温度用変数.
    int temp;                     ← ② 小数部温度処理用変数.

    valHigh = readUpp();          // 整数部をセンサーから取得
    Serial.print(valHigh);        // 整数部を出力
    Serial.print(".");            // 小数点
    valLow = readLow();           // 少数部をセンサーから取得
    temp = (valLow >> 4) * 625;   // LSB:0.0625 小数部温度計算
    Serial.print(temp/1000);      // 0.1の位
    temp %=1000;
    Serial.print(temp/100);       // 0.01の位
    temp %=100;
    Serial.print(temp/10);        // 0.001の位
    temp %=10;
    Serial.println(temp);         // 0.0001の位
    sprintf(l_moji," %2d",valHigh);   // 整数部 ← ③ 整数部温度 LCDバッファ格納.
    l_moji[3]='.';                // 小数点
    temp = (valLow >> 4) * 625;   // LSB:0.0625 小数部温度計算
    l_moji[4]=(char)('0' + temp/1000); // 0.1の位
    temp %=1000;                                  ④ 小数部温度 LCDバッファ格納.
    l_moji[5]=(char)('0' + temp/100);  // 0.01の位
    l_moji[6]=0xDF;               // ℃の代わり
    l_moji[7]='C';

    writeCommand(0x80);           // 1行目 DRAM ADDR.
    for(int i=0; i<8; i++) {      // 表示
      writeData(u_moji[i]);
    }
    writeCommand(0x40+0x80);      // 2行目 DRAM ADDR.
    for(int i=0; i<8; i++) {      // 表示
      writeData(l_moji[i]);
    }
}
```

図　232

デジタル温度センサーからデータを取り込み、LCD に表示するプログラムは、これまでに作成したものをそのまま利用します。

プログラムを書く　温度読込

```
byte readUpp() {
    // 整数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);   ← ① I2C通信開始.
    Wire.write(0x00);                       ← ② 温度整数部レジスタ指定.
    Wire.endTransmission();                 ← ③ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);      ← ④ I2C通信にて1byte読込リクエスト.
    return(Wire.read());                    ← ⑤ 温度(整数部)読込. 戻り値にセット.
}

byte readLow() {
    // 少数部をセンサーから取得
    Wire.beginTransmission(STTS751_ADRS);   ← ⑥ I2C通信開始.
    Wire.write(0x02);                       ← ⑦ 温度小数部レジスタ指定.
    Wire.endTransmission();                 ← ⑧ I2C通信終了.
    Wire.requestFrom(STTS751_ADRS, 1);      ← ⑨ I2C通信にて1byte読込リクエスト.
    return(Wire.read());                    ← ⑩ 温度(少数部)読込. 戻り値にセット.
}
```

図　233

温度読込の関数もこれまでに作成したものです。

プログラムを書く　センサー初期化

```
void init_STTS751() {
    Wire.beginTransmission(STTS751_ADRS);   ← ① I2C通信開始.
    Wire.write(0x03);                       // config reg.  ← ② 03レジスタ指定.
    Wire.write(0b10001100);                 // EVENT停止、12bit分解能指定
    Wire.endTransmission();                 ← ③ 温度センサー初期設定.
}                                           ← ④ I2C通信終了.
```

図　234

センサー初期化もこれまでに作成したものです。

プログラムを書く　文字送信・コマンド送信

```
void writeData(byte t_data) {
  Wire.beginTransmission(LCD_ADRS);
  Wire.write(0x40);
  Wire.write(t_data);
  Wire.endTransmission();
  delay(1);
}

void writeCommand(byte t_command){
  Wire.beginTransmission(LCD_ADRS);
  Wire.write(0x00);
  Wire.write(t_command);
  Wire.endTransmission();
  delay(10);
}
```

① I2C通信開始.
② 0x40送信.
③ 表示文字コード送信.
④ I2C通信終了.
⑤ I2C通信開始.
⑥ 0x00送信.
⑦ コマンドコード送信.
⑧ I2C通信終了.

図　235

LCD に文字送信・コマンド送信もこれまでに作成したものです。

プログラムを書く　LCD初期化

```
void init_LCD() {
  delay(100);
  writeCommand(0x38);  // Function set
  delay(20);
  writeCommand(0x39);  // Function set
  delay(20);
  writeCommand(0x14);  // OSC Freq. set
  delay(20);
  writeCommand(0x70);  // Contrast set
  delay(20);
  writeCommand(0x56);  // 3.3V, ICON,Contrast
  //writeCommand(0x52);  // 5V, ICON,Contrast
  delay(20);
  writeCommand(0x6C);  // Follwer Control
  delay(20);
  writeCommand(0x38);  // Function set
  delay(20);
  writeCommand(0x01);  // Clear Display
  delay(20);
  writeCommand(0x0C);  // Display ON/OFF control
  delay(20);
}
```

図　236

LCD 初期化もこれまでに作成したものです。

動作確認

図　237

動作確認は、まず LCD への温度表示を確かめます。（上図）

次にシリアルモニターを起動（下図）して、通信速度を合わせます。温度が順次表示され、10 回表示すると Ambient に送信した旨の表示が行われます。

図　238

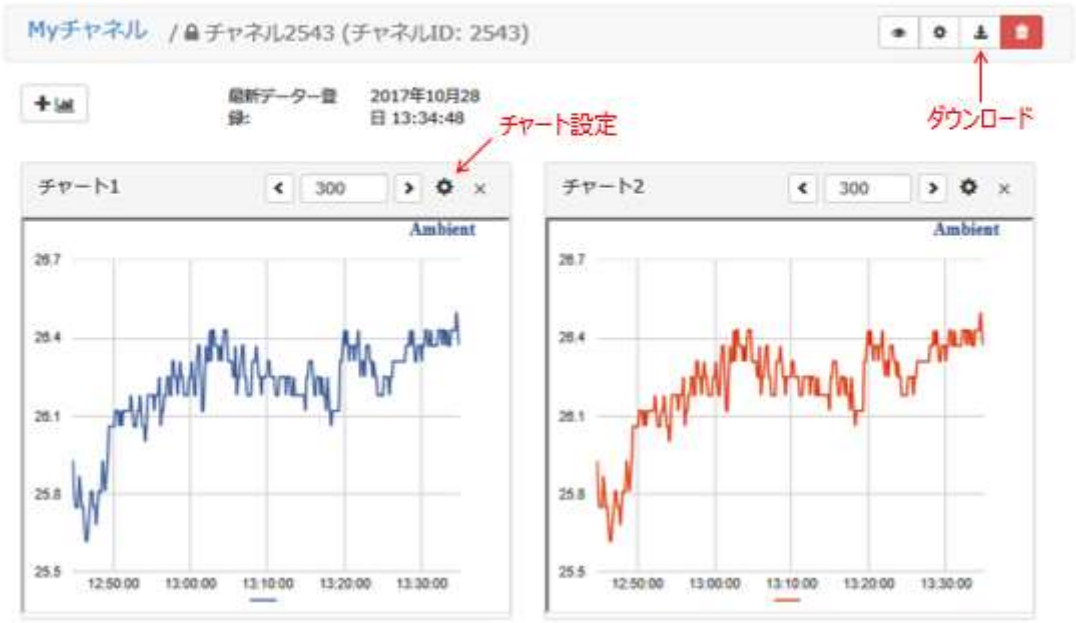


図 239

次に、携帯端末や PC で My チャネルにアクセスします。すると、上図の様に、グラフが表示されます。今回は同じデータを 2 つ送っているので、同じチャートが 2 つ表示されています。チャートの右上にある【歯車ボタン】で表示の設定変更ができます。ページの右上にある【ダウンロードボタン】でこのチャネルに送ったデータを CSV ファイルとしてダウンロードすることができます。送信したデータを別途、統計・分析などに利用することができるので、とても便利な WEB サービスです。

下の写真は、温度と湿度を同時に測定できるデジタルセンサー【SHT31】を利用した例です。

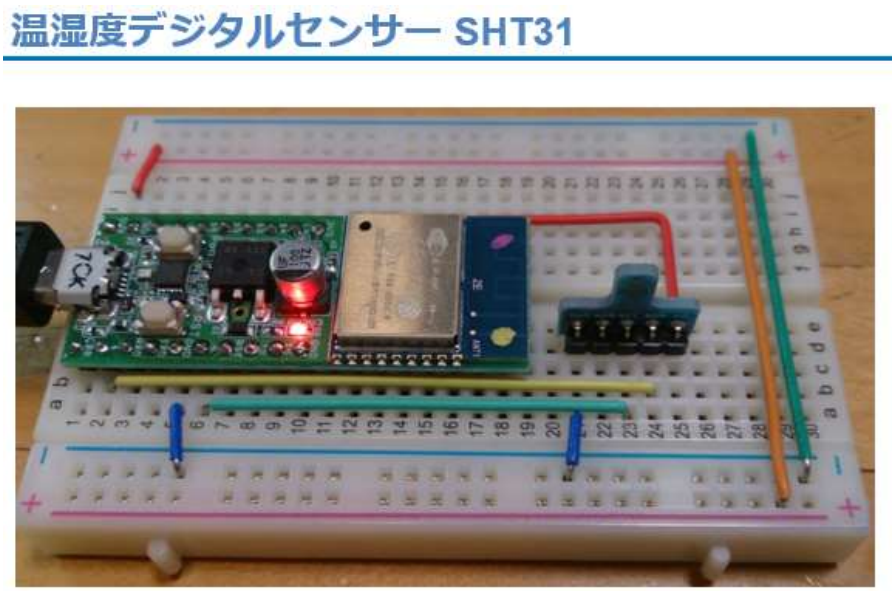


図　240

図　241

ソースコードの一部を示します。コメントを見れば、何をしているかが分かると思います。

ESP_2113_SHT31_Ambient

```
void loop()
{
    float temp,hum;                         // センサ用の浮動小数点数型変数
    char s[6];

    // SHT31から温湿度データを取得
    SHT31.GetTempHum();
    temp=SHT31.Temperature();               // 温度を取得(Float)して変数tempに代入
    hum =SHT31.Humidity();                  // 湿度を取得(Float)して変数humに代入
    Serial.println("-------------------------");
    Serial.println("Temperature ('C)");
    Serial.println(SHT31.Temperature());
    Serial.println("Humidity (%)");
    Serial.println(SHT31.Humidity());

    if( temp>-40. && hum>=0.){              // 適切な値の時

        /* クラウドへ */
        dtostrf(temp,5,2,s);                // 温度(Float)を文字列に変換
        ambient.set(1,s);                   // Ambient(データ1)へ温度を送信
        dtostrf(hum,5,2,s);                 // 湿度(Float)を文字列に変換
        ambient.set(2,s);                   // Ambient(データ2)へ湿度を送信
        ambient.send();                     // Ambient送信の終了(実際に送信する)
    }
    // 待ち時間
    delay(10000);
}
```

図 242

第14回　WEB 連携⑤(Blynk)

これまでに、解説をしてきた WEB サービスでは、情報が WiFi マイコンから WEB サービスを経由して遠隔地端末へ通知するという流れでした。それが可能になるのであれば、それとは反対向きに遠隔地端末から WiFi マイコンに通知というのも、考えられます。この一連の講座の最終回は、離れたところから WEB サービスを通じて、WiFi マイコンを操作する遠隔操作を行ってみます。Blynk というサービスを使うモデルです。

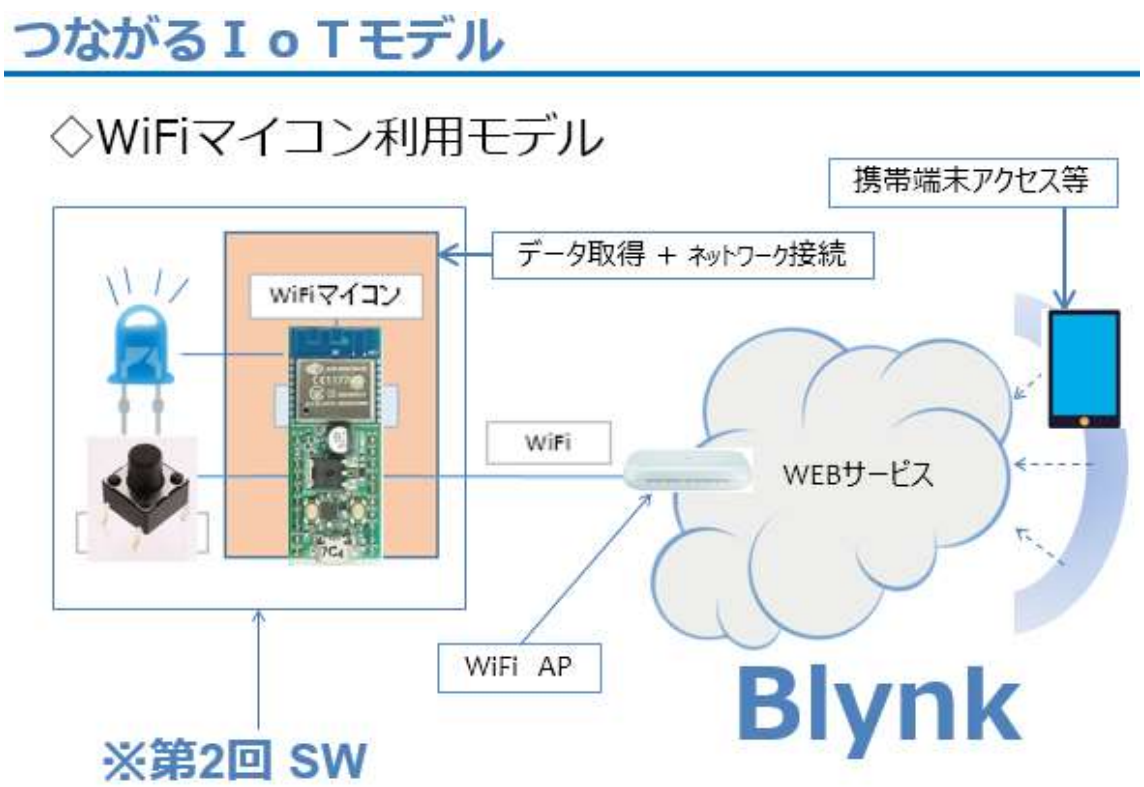


図　243

SW 状態を WEB 経由で携帯端末に通知し、携帯端末を操作することで WEB を通じて WiFi マイコン側の LED を点灯制御します。

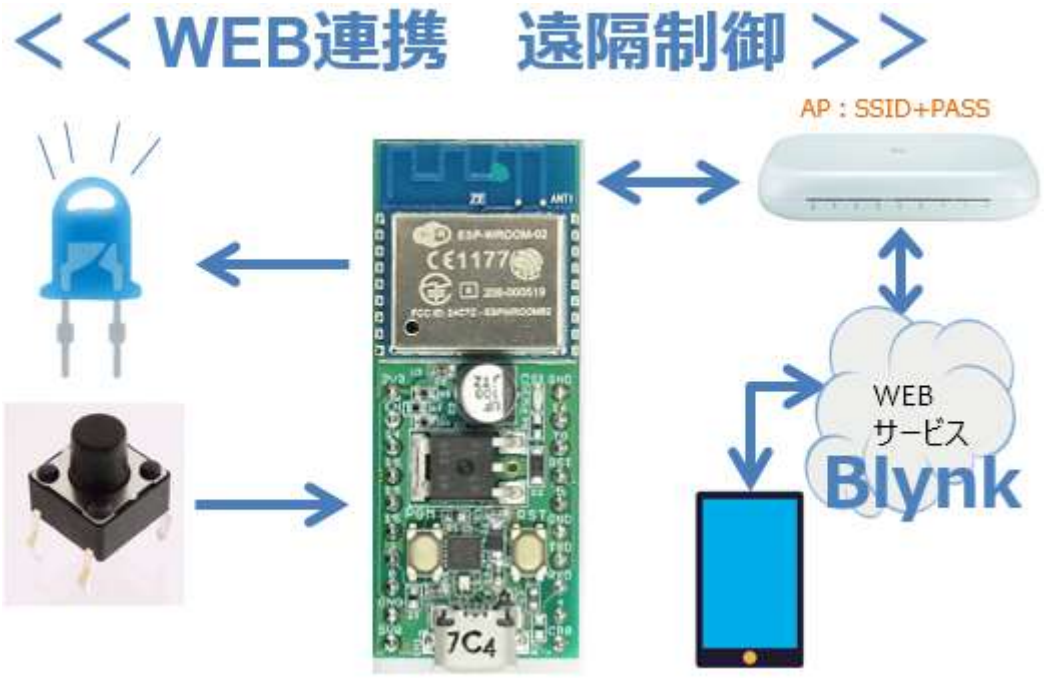


図 244

上の図は第 12 回で説明した図に良く似ていて、WEB サービスが MQTT から Blynk に変わっただけに見えます。しかし、内容には大きな違いがあります。第 12 回では LED の制御は WiFi マイコンが SW の ON/OFF 状態を判断して点灯/消灯していたのに対して、これから開発するシステムでは、SW 状態は携帯端末に通知して携帯端末の画面上に描いた LED が点灯/消灯します。同様に携帯端末の画面上に描いた SW ボタンの状態を WiFi マイコンに通知して、それに対応して実際の LED が点灯/消灯するというものです。ここでは、【あえて言葉で説明】しているので理解しにくいかもしれませんが、後の説明を読み進むと、「こんなに容易に WEB 経由のリモートコントロールができるのか！！」と驚くと思います。

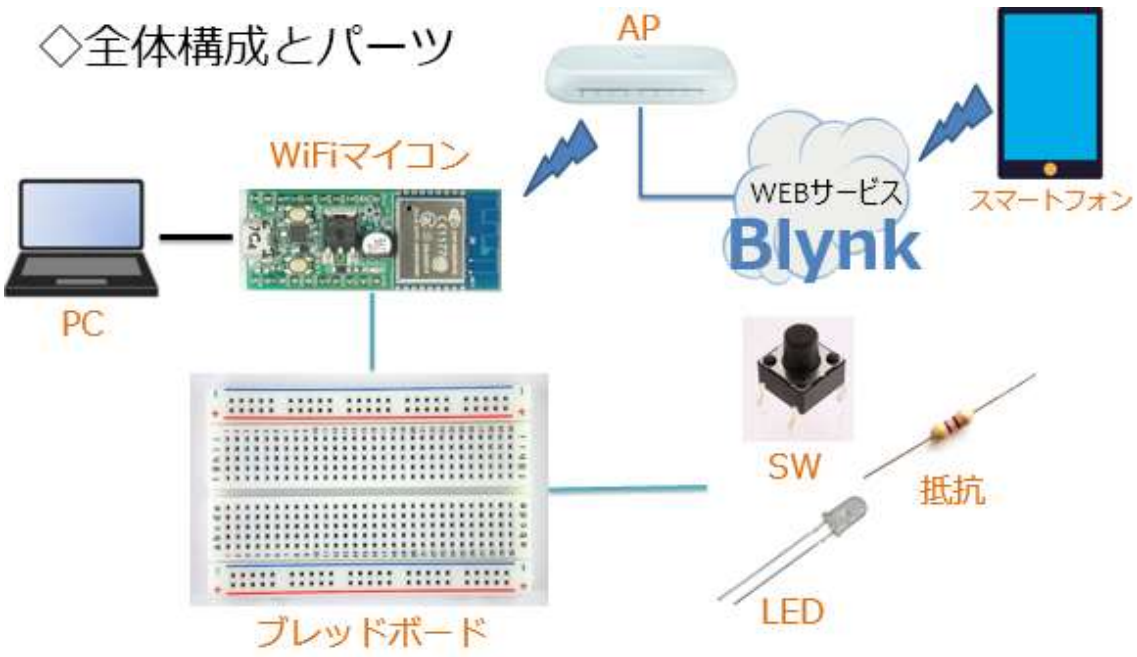


図 245

全体の構成は上図の通りです。必要な機材・パーツを下記に示します。

1. WiFi マイコン×1 台
2. PC（プログラム開発・書込）×1 台
3. USB ケーブル（マイコンとの接続）×1 本
4. ブレッドボード×1 個
5. 配線用ジャンパー線×適宜
6. LED×1 個
7. 抵抗器（470Ω）×1 個
8. SW×1 個

上に使用する機材・パーツを示します。

次に回路図と配線した様子を示します。この回路は第 2 回、第 12 回で作成したものと同じですので説明の必用はないでしょう。

SW・LED点灯回路

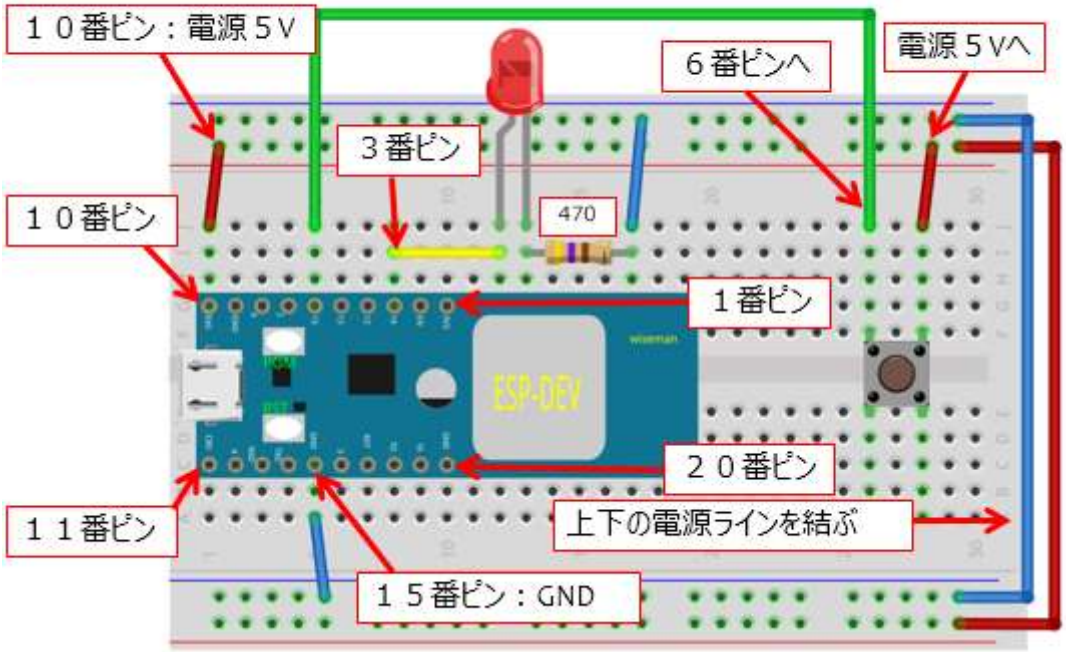


図　246

実際に配線した様子

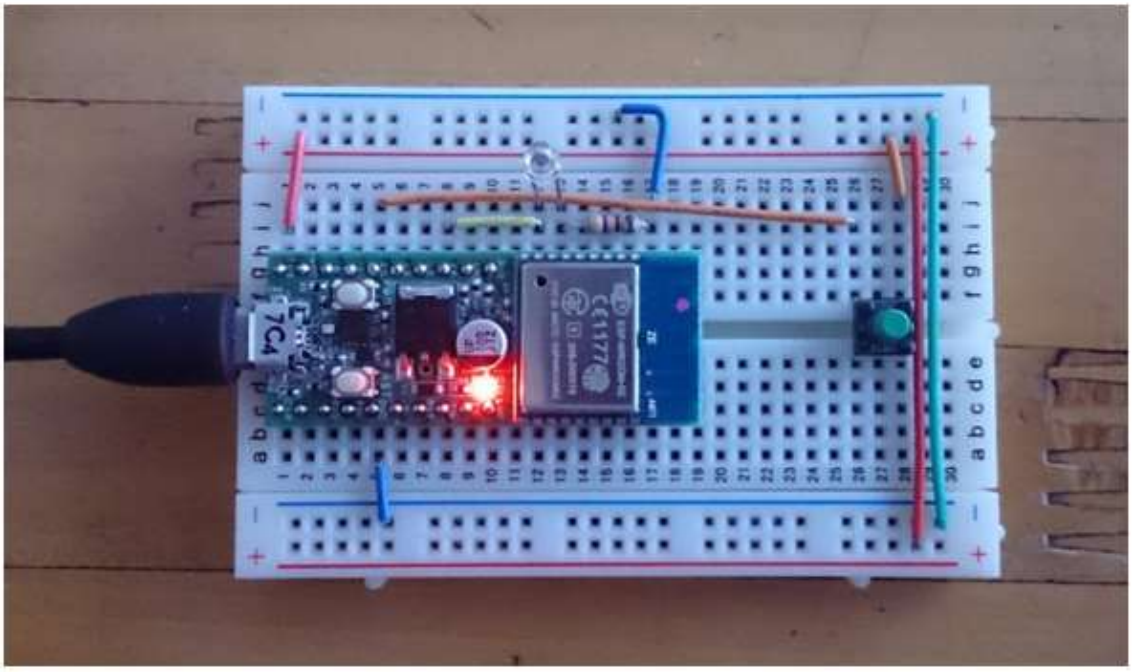


図　247

今回利用するWEBサービスBlynkは、スマートフォンとWiFiマイコンがWEB連携できますが、スマートフォン側の操作画面を自由に指先だけで作れます。SWやLEDなどの画面に配置できるパーツ（ウィジェットと呼ぶそうです。）がそろっています。

図 248

この講座では、スマートフォン側はAndroid用のアプリケーションを使用します。（iOS用のものもあります。）

まず、上のURLにアクセスしてみて下さい。次のようなページがヒットします。

WEBサービス　Blynk

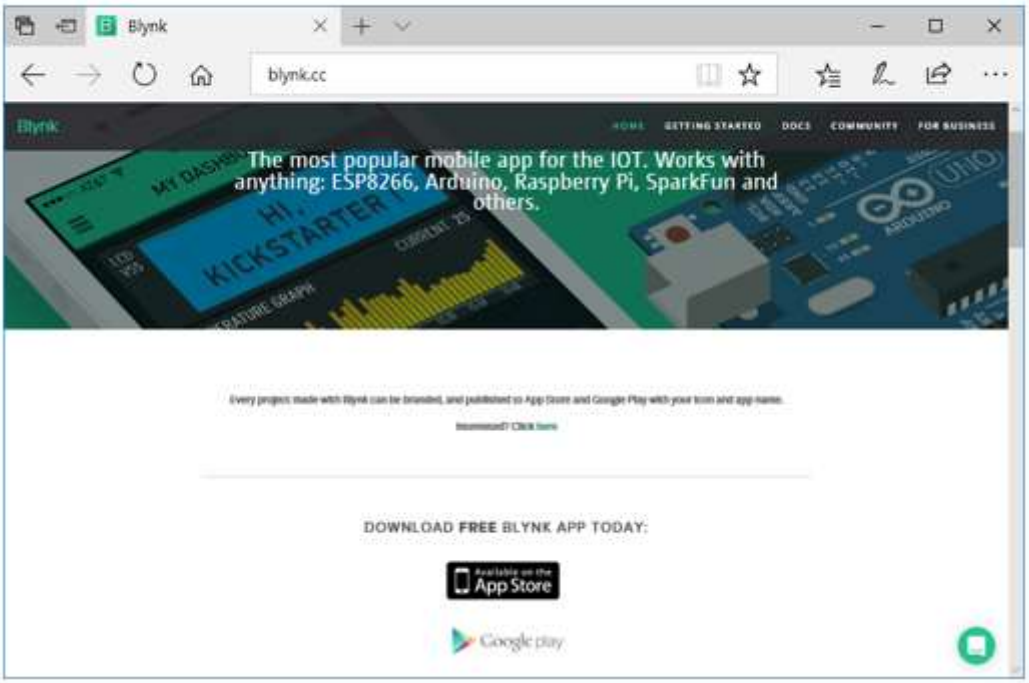


図　249

このページを見ると、上の方にはESP8266(WiFiマイコン)やArduino、Raspberry Pi、SparkFunなどに利用できることが書いてあります。また、下の方にはApp StoreやGoogle playなどへのリンクがあります。

このサービスを利用するには、まずスマートフォンアプリをインストールすることから始めます。

図　250

使用するスマートフォンで Blynk を検索すると、上の図のようなアプリがヒットします。これをインストールすると、右のようなボタンが作られます。これをタップしてアプリを起動してください。（下図）

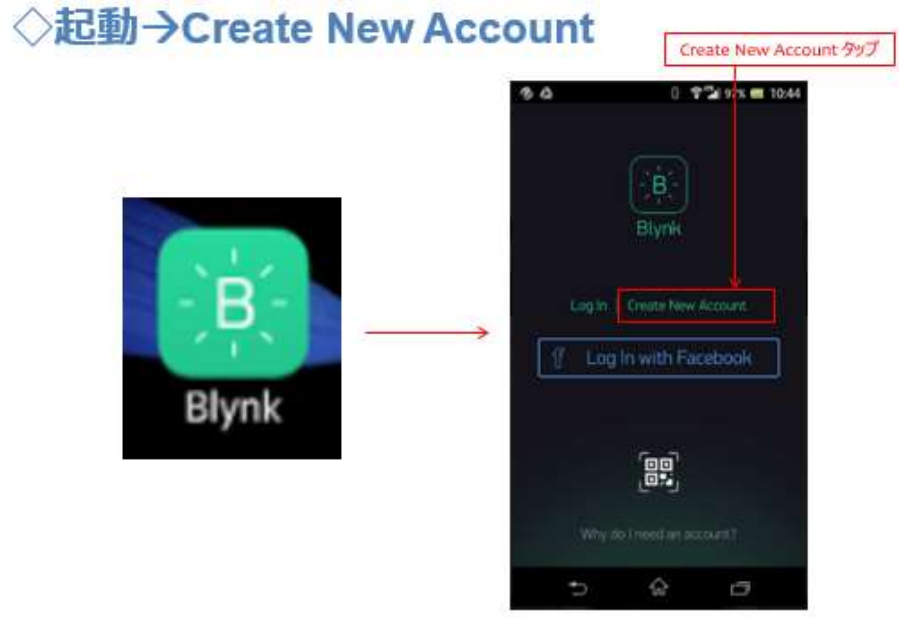


図　251

【Create New Account】という表示のある部分をタップします。

図 252

【Create New Acount】というタイトル画面に切り替わります。(上図左)ここに連絡の取れる Mail Address と指定する Pass Word を入力して、画面中央の【Sign Up】をタップすると、【Blynk】というタイトルの画面に切り替わり（上図中央）ます。

【注意】

Mail Address は、後に発行されるキーワードを送信できる、WiFi マイコンのプログラムを開発する PC で使える Mail Address が便利です。

タイトルの下に New Project と表示されています。この部分にはプロジェクト名が表示されます。この画面の中央部をタップします。するとタイトルが【New Project】の画面に変わります。

タイトルの右にナットマークのボタンがありますから、これをタップします。すると、次の図左の画面【Project Settings】に切り替わります。

図 253

New Project の表示のある場所（上図左）には、このプロジェクトの名称を入力してください。画面を下に移動して New Device をタップします。（上図中央）画面が【My Devices】に変わります。ここで【＋New Device】という表示をタップします。画面は次の図左の【My Devices】に変わります。

図 254

上図左で、New Device にはデバイスの適当な名称を入力してください。次のプルダウンからは、【ESP8266】を選択してください。これが、Blynk とつながる WiFi マイコンの機種になります。さらにその下のプルダウンから接続方法として【WiFi】を選びます。

【重要】

ここで【AUTH TOKEN】の部分に何か長い文字が標示されます。この AUTH TOKEN をキーワードとして、WiFi マイコンのソースコードに記述することで、WEB サービスと WiFi マイコンが繋がります。AUTH TOKEN はその下の【E-mail】をタップすることで、先に登録したメールアドレス宛に通知されます。これが WiFi マイコンのプログラムを開発する PC で使えるメールアドレスが便利という理由です。※表示されている AUTH TOKEN をタップすると、クリップボードにコピーされますので、それを参照しながらソースコードを入力することもできます。

次に【My Devices】の左にある【←】をタップして、前画面に戻ります。ここから、スマートフォン画面にボタンやLEDを配置して設定します。タイトルの右側にあるボルトの頭のマークをタップして下さい（前図中）。一番上に電池のような絵があり、Button・Slider・Timer・Joystick・・・などが並ぶ【Widget Box】画面が表示されます。まず、一番上にある【Button】をタップしてください。すると、北全の画面にButtonが配置されています。再び、ボルトの頭のマークをタップして【Widget Box】に切り替えて、次は下の方に移動すると、LEDがありますのでこれをタップして配置します（下図左）。画面左上には【Button】と【LED】が1個ずつ配置されています。配置された【Button】を指で、しばらく触れていると、【Button】が反応して、すこし大きくなります。そのまま、画面上で指を動かすことで、自由な位置に移動することができますので、好きな場所に配置してください。私は画面中央に配置しました（下図中）。

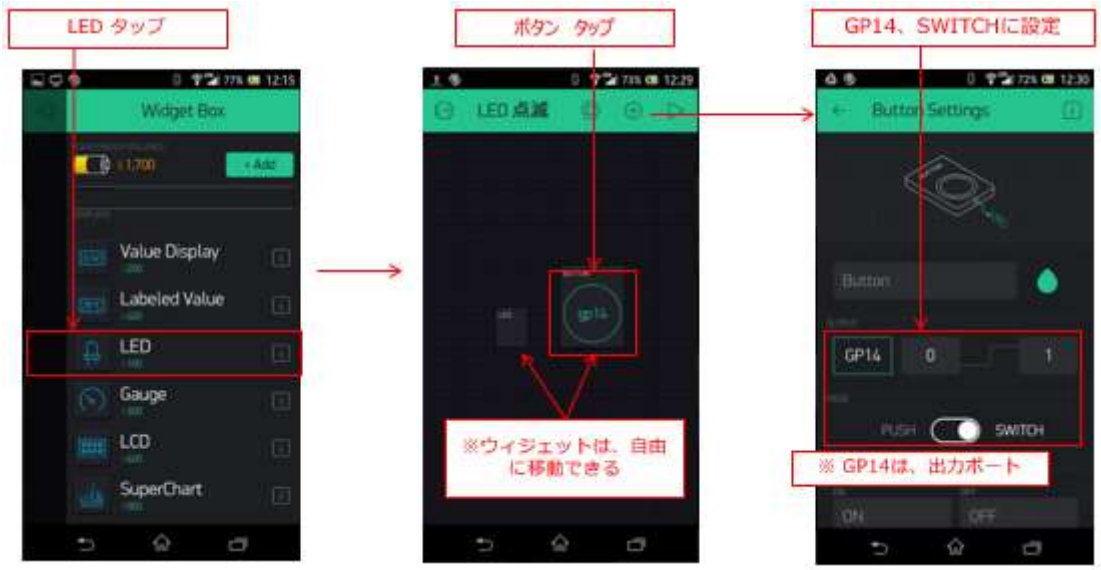


図 255

次に、個々の【Widget】について、設定をします。まず【Button】をタップして下さい（上図中）。すると【Button Settings】画面に変わります（上図右）。

図 256

画面の中央に【OUTPUT】と表示のある場所（最初は PIN が選択されているところ）をタップして【Select pin】画面にします（上図左）。PIN の表示を下に移動して【gp14】を選択します。これは、WiFi マイコンの GPIO14 を使用するという設定です。【OK】をタップして直前の画面委戻ります。GP14 の表示を確認して下さい。次に、【PUSH】と【SWITCH】の選択を【SWITCH】にスライドします。

【重要】

【PUSH】に設定すると、基板上に配置した SW と同じ動作になります。【Switch】に設定すると、一度【Button】を押すと ON になり、再度押すと OFF になる、トグル動作になります。

【Button Settings】の画面は最終的に上図右のようになります。

【Button Settings】の左側【←】部分をタップして、前画面に戻ります。

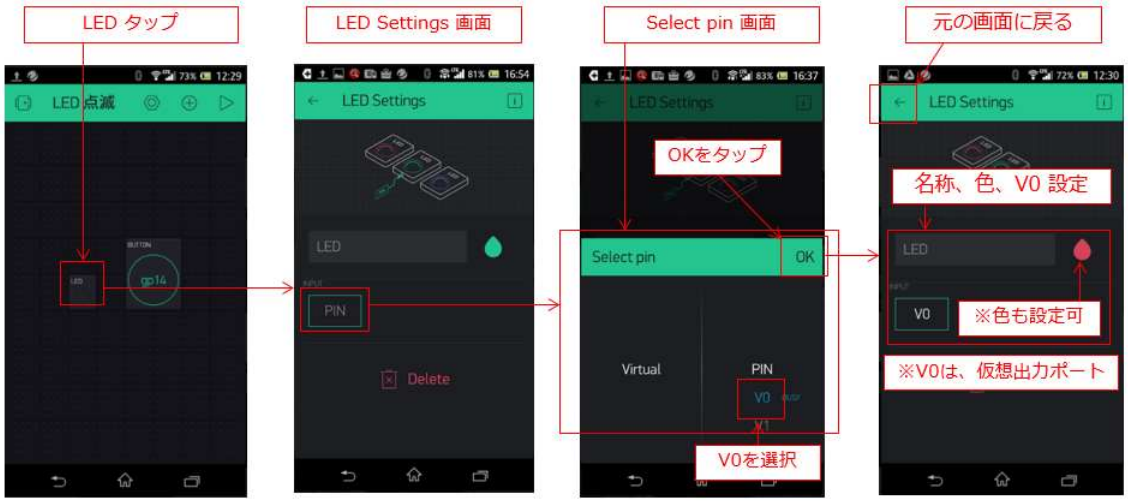


図 257

次は、【LED】を設定します。画面に配置した【LED】をタップします（上図左）。【LED Settings】画面に変わります。INPUT 表示の下【PIN】と表示されている部分をタップします。【Select pin】画面に切り替わりますので、PIN の下のリストから【V0】を選択して【OK】をタップします。

【重要】

この【V0】とは、スマートフォンに【LED】として配置した Widget に対する仮想デジタル出力と考えます。WiFi マイコンは、この【仮想デジタル出力ポート 0 番：V0】に SW の状態に応じた出力を行うソースコードを記述するのです。

設定が終わりましたら、元の画面に戻ります。

図 258

元の画面に戻ると、上図左のような画面になっています。私はプロジェクト名に【LED 点滅】と入力しました。タイトルの右側に三角印がありますが、このプロジェクトを WEB 上の Blynk サービスと連携するときには、この三角印をタップしてサービスと接続します。

その手順を下記します。

【重要】

①. WiFi マイコンの電源を入れ、システムを起動します。

②. Blynk アプリの三角印をタップして Blynk サービスに接続します。

この順番でシステムが連携します。Blynk アプリの三角印を先にタップすると【Device is offline】というメッセージが表示されて、巧く接続できません。

ソースコードを入力する前に、Blynk 用ライブラリの準備を行います。

Blynkライブラリの準備

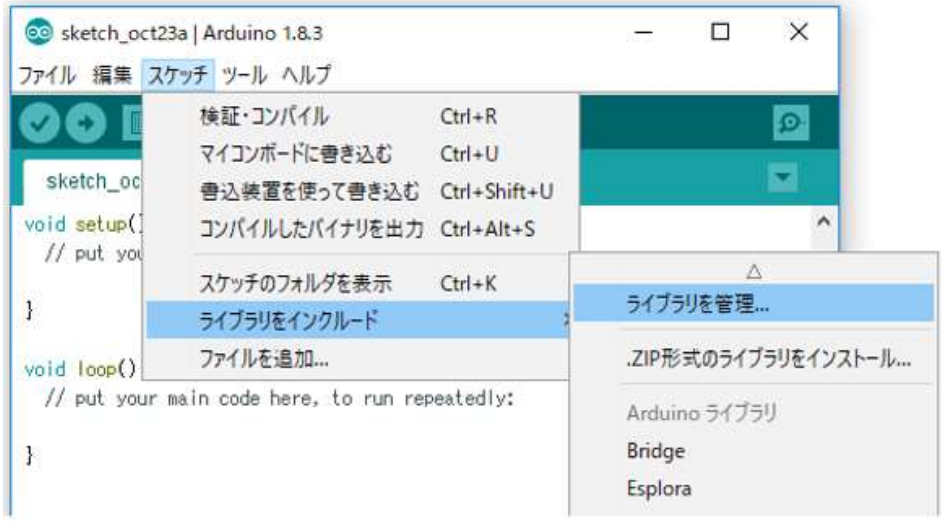


図 259

IDE メニューで【スケッチ→ライブラリをインクルード→ライブラリを管理】と辿り、ライブラリマネージャを起動します。

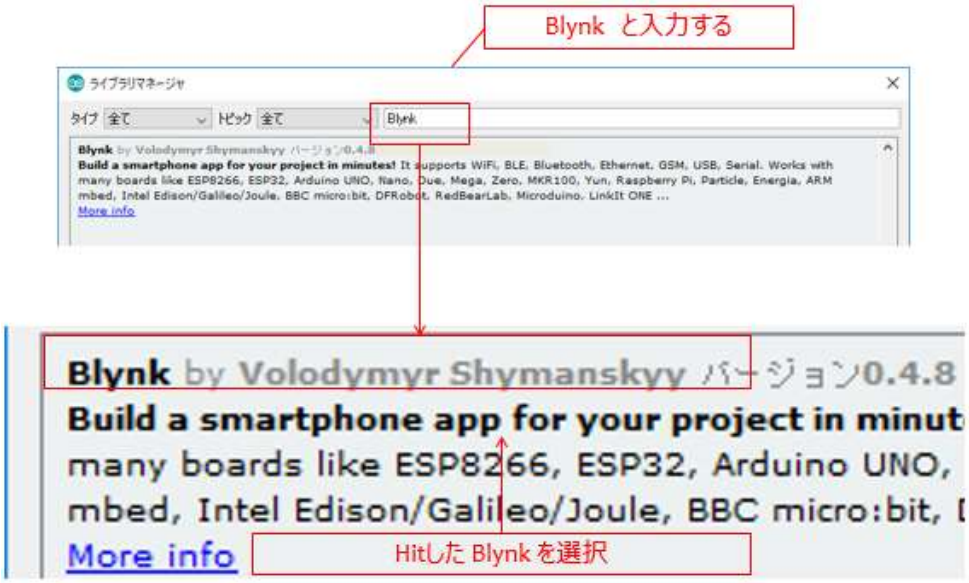


図 260

ライブラリマネージャの上部に【Blynk】と入力します。

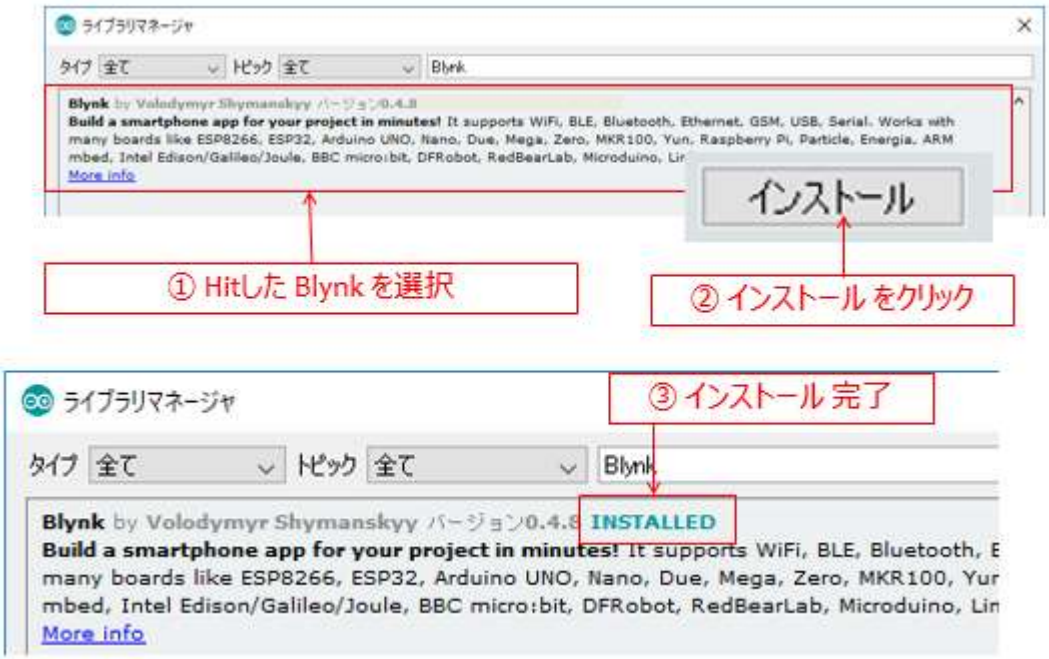


図　261

Blynk ライブラリを選択してインストールします。

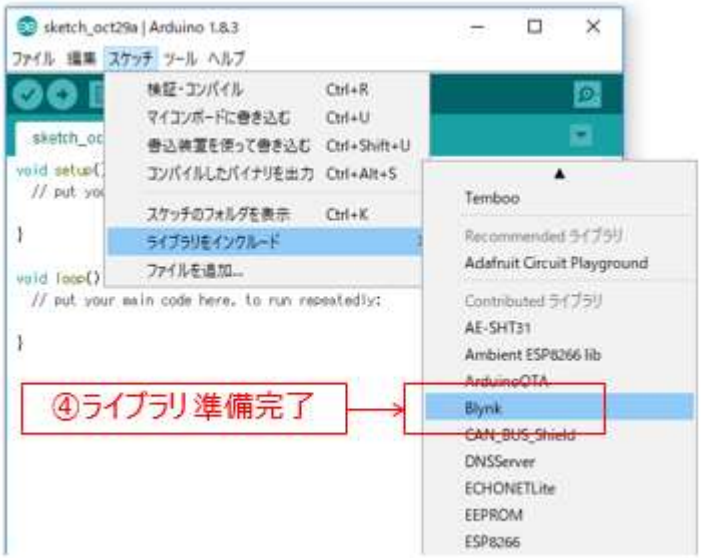


図　262

【スケッチ→ライブラリをインクルード】で表示される一覧に【Blynk】があれば、準備完了です。

ここまで準備に時間が要りましたが、ソースコードはシンプル過ぎて驚愕します。

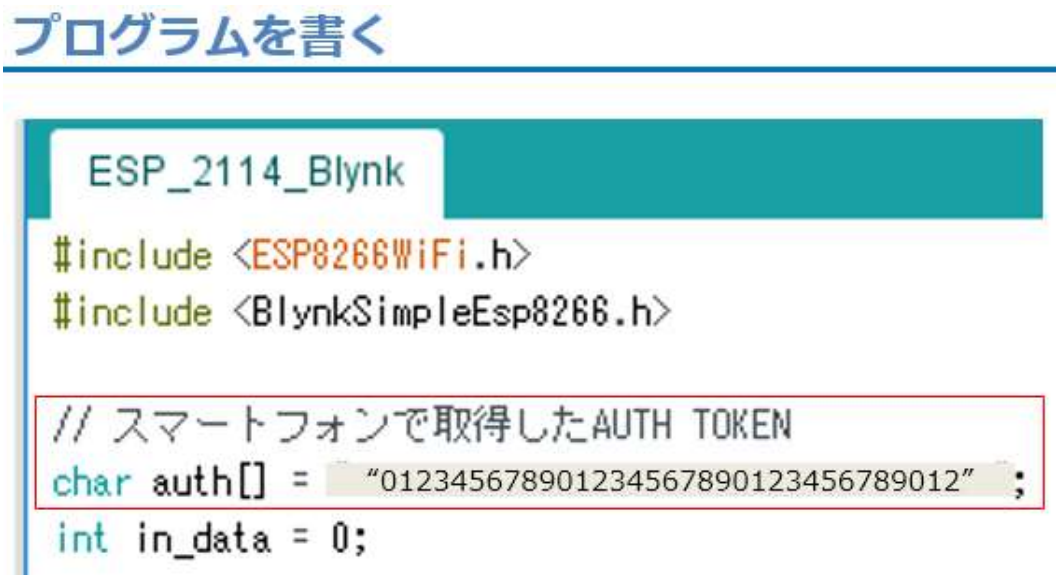


図 263

Blynk 用のヘッダの取り込みを忘れないようにしてください。

32 文字の Auth Token の記述は、カット＆ペーストが確実です。

図 264

Blynk.begin()関数で、Auth Token を指定して、WiFi 経由で Blynk に接続します。マイコン側が先に接続をしておく必要があります。

プログラムを書く　loop()

```
void loop()
{
  Blynk.run();
  in_data = digitalRead(15) * 255 ;
  Blynk.virtualWrite(0, in_data);
}
```

図 265

loop()関数は、僅かに 3 行です。1 行目の Blynk.run()は、WEB 上で双方向の情報のやり取りを行っています。loop()関数内で必ず呼び出すようにします。digitalRead()関数は、GPIO15 に接続した SW の状態を読み込んでいます。in_data には、SW 状態に 255 を掛けた値が格納されます。何故 255 を掛けているのかというと、この 255 はスマートフォン画面に配置した【LED】Widget の明るさを指定する値です。この値を 127 にすると【LED】表示の明るさが約 50%になります。PWM 制御を行っているようです。

Blynk.virtualWrite()は、パラメータに 0 と in_data が含まれていますが、この 0 が【LED】Widget に設定した仮想デジタル出力ポートの番号になります。

さあ、ソースコードが入力できたら、名前を付けて保存してください。そして WiFi マイコンと PC を USB ケーブルで接続して、コンパイル・リンク・書込みです。WiFi マイコンの Reset と PGM の SW 操作を忘れない様にしてください。書き込みが終了したら、動作確認です。

図 266

上図左の通り、WiFI マイコンは既に電源が入りスタートしています。スマートフォンで Blynk アプリを起動して、プロジェクトの三角印をタップしてください。何もエラーが出なければ Blynk と接続されています。まず、スマートフォン画面の【Button】をタップしてください。それが、WiFi マイコンの LED に反映されれば【半分成功】です。次に、WiFi マイコン側の実物の SW を ON/OFF してみてください。その状態が、【LED】Widget に反映されれば【大成功】です。

◇スマートフォン画面SW

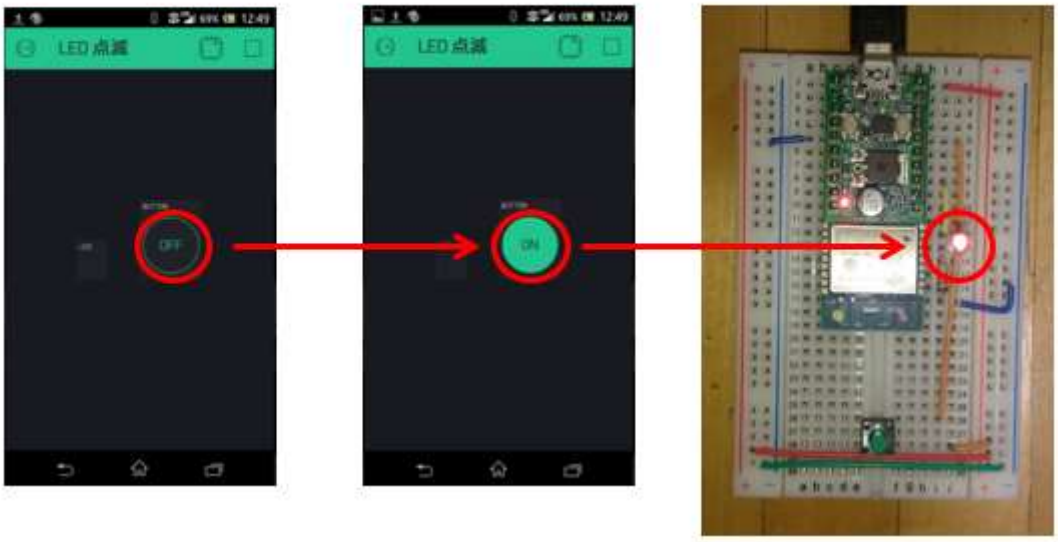


図　267

◇マイコンSW操作

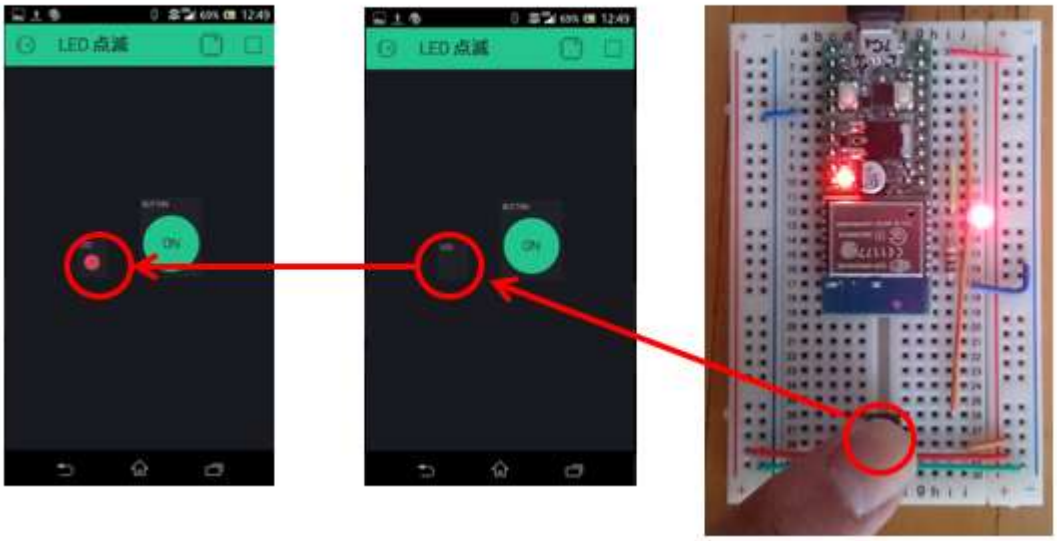


図　268

いかがでしょうか、スマートフォン側の設定作業が必要でしたが、難しくはなかったはずです。ソースコードは 10 数行ですから、驚きです。

Blynk アプリには、次の図のように多くの Widget が準備されています。Widget Box の上部にある電池マークの数字【エネルギー】の分だけ、無料で使用することができます。Wdget を配置すればエネルギーは減りますが、Delete すれば元に戻ります。複雑なシステム対応でエネルギーが不足するような場合は【＋Add】で追加購入できるようです。

色々なウィジェットと無料枠

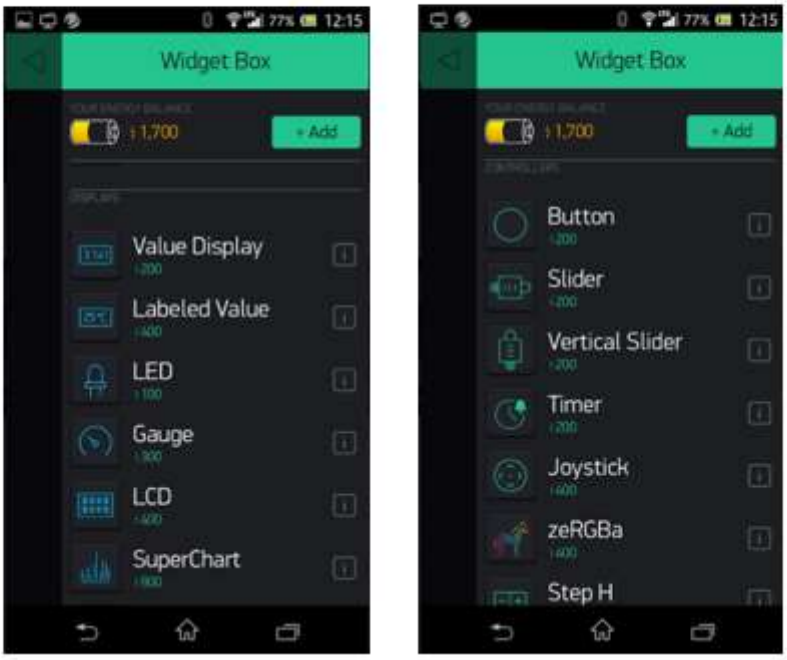


図 269

これで、全ての講座が終わりました。後は、皆さんの創意工夫で、色々なシステムに挑んでください。【王道】を基本にして、色々組み合わせて利用すれば、たくさんのバリエーションができると思います。

お疲れさまでした！！

Appendix　A：実習キット

図 270

2冊のテキストで使用する、全てのパーツが揃った実習キットを示します。

Appendix B：使用する全パーツ(第一分冊も含む)

図　271

図　272

最後に

最後まで進めていただき、ありがとうございます。すべての講座を実習された方は、WiFi マイコンモジュールを使った WEB サービス連携システムの設計・開発ができるようになっているでしょう。ここで取り上げることのできなかった他の WiFi マイコンや様々な WEB サービスが、まだまだ沢山あります。それらは今後も様々な特徴を持って改善・開発され、世の中に提供されてきます。その時、ここで身に着けた IoT デバイス開発や WEB 連携システム開発の技術をたたき台にして、新しい技術に臨んでみてください。同じようなステップを踏めば、どんな技術でも利用できる技量を自分の物にすることができるでしょう。

2017.12.1

有限会社ワイズマン　原田賢一

農業分野における「まち・ひと・しごと創生」の実現を支援する農業 IT 人材育成テキスト
（農業 IT 編）

このテキストは、平成２９年度文部科学省　「専修学校による地域産業中核的人材養成事業」「農業分野における『まち・ひと・しごと創生』の実現を支援する農業 IT 人材の育成」事業で開発されました。

平成３０年２月
学校法人三橋学園
船橋情報ビジネス専門学校