平成 30 年度「専修学校による地域産業中核的人材養成事業」

【IoTへのドリル2】
ネットワーク設計・構築

情報通信技術に対応した組込みシステム開発技術者育成のモデルカリキュラム開発と実証事業

平成 30 年度「専修学校による地域産業中核的人材養成事業」

【IoTへのドリル2】
ネットワーク設計・構築

情報通信技術に対応した組込みシステム開発技術者育成のモデルカリキュラム開発と実証事業

この講座を進める前に

このテキストはIoTへのドリルとして作成しました。特にこの第二分冊は、Internetに接続してWEBサービスを活用したIoTシステムを開発するためのトレーニングドリルと認識していただくのが良いと思います。どのテーマも第一分冊に続く内容となっているので、必要に応じて、第一分冊を参照していただくのが良いと思います。

このテキストの内容を実習するには、インターネットに接続されたPCとWiFiアクセスポイントが必用です。また、Androidスマートフォン（またはタブレット）を利用してIoTシステムを構築する講座も含まれています。また、Arduino IDEと必要なドライバ、Python開発環境とそれぞれのライブラリは、第一分冊で準備が済んでいることが前提となっています。特にプロキシサーバーを経由したインターネット接続環境では、テキストで意図したWEB接続ができないことがありますので、学校等の環境で実習する際は、事前に充分確認を行ってください。

各回で解説しているソースコードは実習キット付属のCDで提供しています。

【IoT へのドリル2】

ネットワーク設計・構築

目次

【無線マイコン(TWE-Lite)編】

第12回　SW 状態検出

今回は、SW 状態を検出してみましょう。SW は ON/OFF 状態をマイコンに入力します。その状態は、デジタル入力(DI) の信号を読めばわかります。

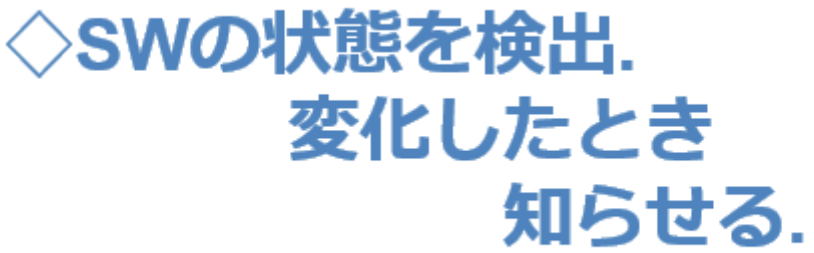

図　1

第 12 回で開発するシステムは、SW 状態が変化したことを検出して、通知するというもので、DI（デジタル入力）の基本的な使い方を実習します。第 13 回ではこの応用を行いますので、そのための準備運動です。

図　2

システム概要は、子機側は第 1 回目と同じです。SW のデジタル入力の状態を親機に通知します。親機は PC と接続していて、子機から送信される電文を解析します。そしてデジタル入力の状態に変化があったとき、PC にメッセージを表示します。

なぜ、ここで既に行ったことを再び実習するのかというと、次の第 13 回から本格的な IoT の中心的領域に踏み込む予定があるからです。ここで確実に子機側（フィールドと読み替えても良いです。）の状態が変化したことを検出できている保証があれば、WEB サービスなどを利用する本格的 IoT システムを構築する際に、問題の発生場所を限定できるからです。第 12 回では、その環境整備をしっかりとしておき、後の第 13、14 回をスムースに運ぶという狙いがあります。

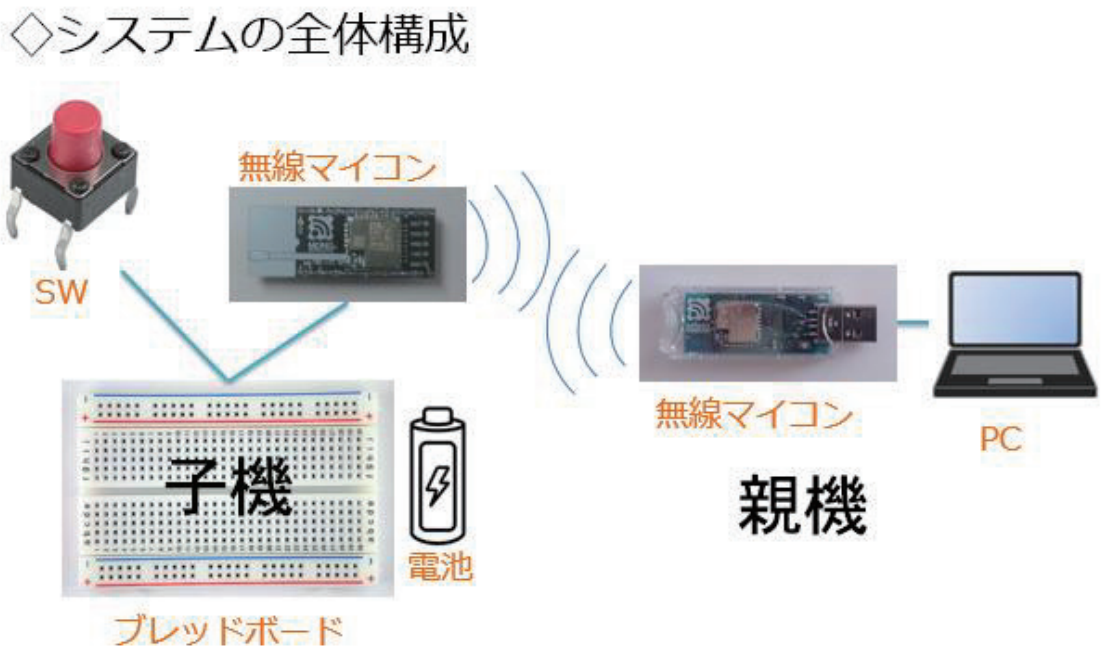

図　3

システムの全体構成は、上図の様に子機側は第 1 回目と同じです。親機側は、MoNoSTICK を PC にセットして、Python のプログラムで処理を行います。

データ受信コマンドでSW状態を知る

◇先頭はコロン【：】で始まるテキストデータ

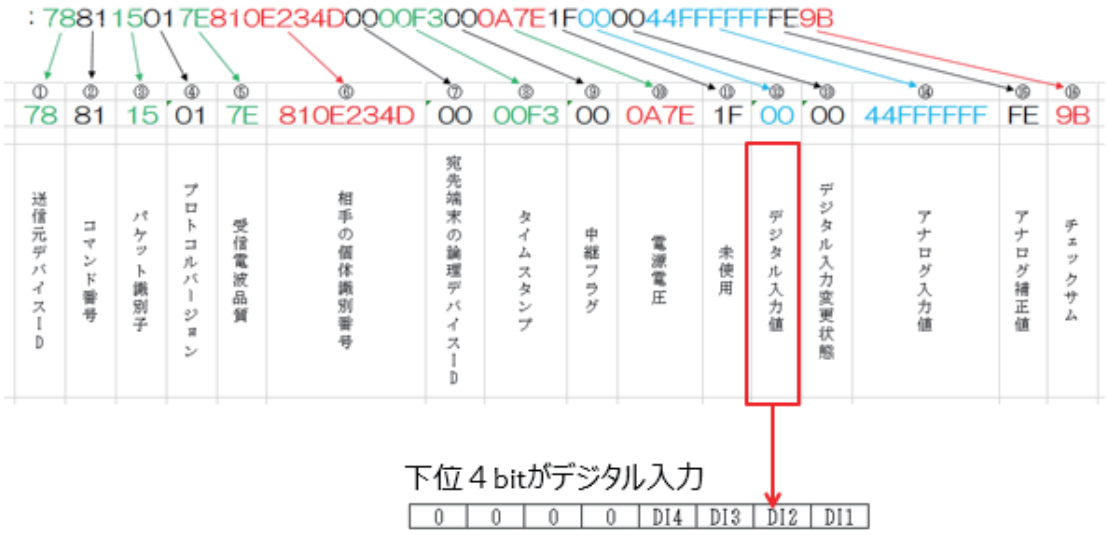

図　4

子機からはSWの状態がデータ受信コマンドという電文で親機に送られてきます。その中のデジタル入力値という箇所が上図に示されています。このデジタル入力値は1byteの16進数ですが、その下位4bitにDI1～DI4が割り当てられていて、デジタル入力の4chを賄っています。受信した電文のこの部分を解析すれば、子機のSWがどのような状態かが分かります。

デジタル入力値

◇8bit のデータ中、下位 4bit でデジタル入力の状態を表している.

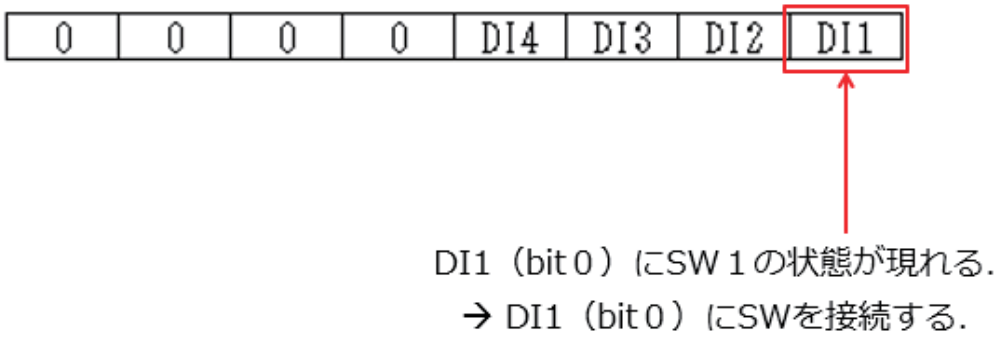

図 5

今回は、デジタル入力 1 に SW を接続しようと考えていますので、上図の DI1 に着目して解析するプログラムを作ります。

使用するパーツは、下記のとおりです。

子機側：

1. 無線マイコンモジュール×1 台
2. ブレッドボード×1 個
3. 電池（単 4 乾電池×2 個＋SW 付電池ケース×1 個）×1 セット
4. 配線用ジャンパー線
5. SW×1 個

親機側：

1. MoNoSTICK×1 台
2. PC×1 台（Windows10）
3. Python 開発・実行環境×1 セット

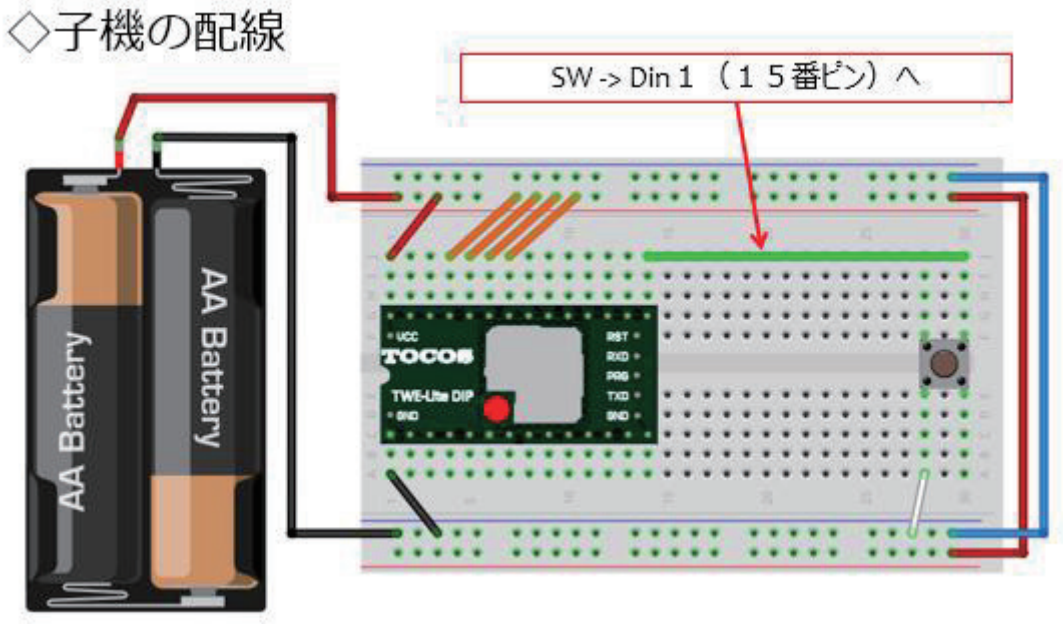

図 6

子機の配線は、第 1 回目で行ったものです。SW の端子の一つ（図の SW の右上端子）はデジタル入力 1 番（15 番ピン）に接続します。他端は GND に接続します。SW を押すと 15 番ピンが Low レベルになるという回路です。実際に配線が済んだ子機は次の様になりました。

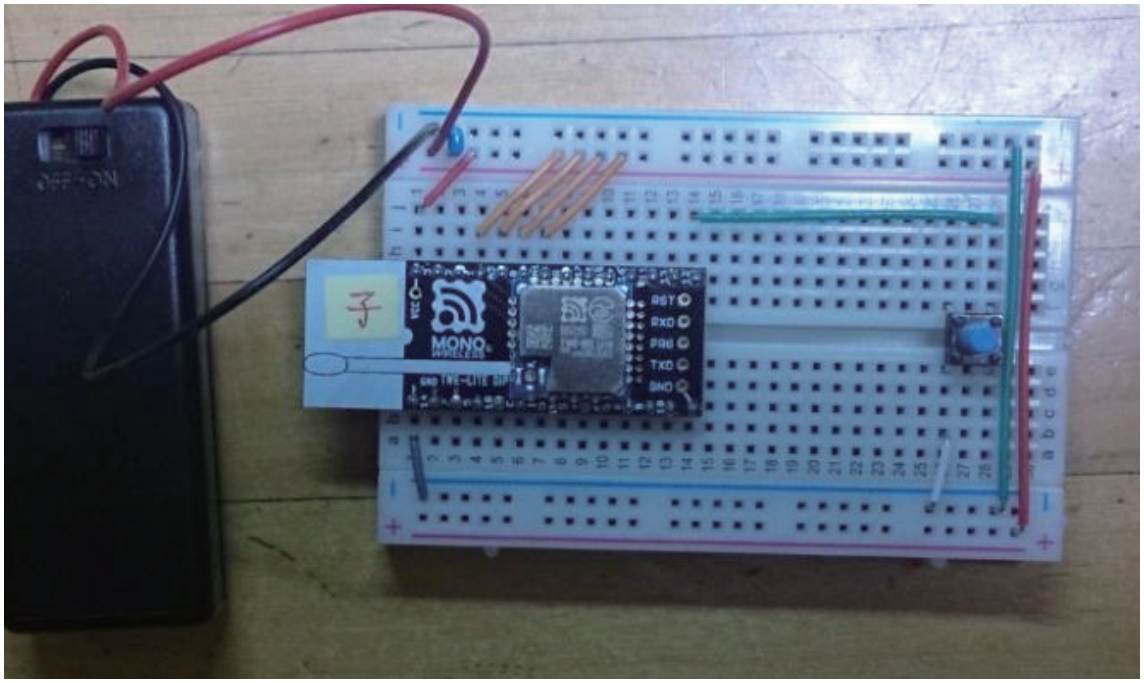

図　7

PC 側 Python のソースコードは次のとおりです。

◇モジュールなどの取り込み

```
import struct       # バイナリ<--->文字列相互変換モジュール
import binascii     # バイナリ<--->ASCII相互変換モジュール
import serial       # シリアル通信パッケージ
```

◇受信電文解析処理

◇電文（子機→親機）を解析する関数

```
def denbunKaiseki(data):   # 受信電文を解析する
    if data[0] != ":":
        return False       # 先頭が「:」でなければ、対象のデータではない
    data = data[1:]        # 先頭の「:」を取り除く

    # バイトデータに変換する
    ss = struct.Struct(">BBBBBIBHBHBBBBBBBBB")  # フォーマット文字列に従って
                                                # バイナリデータを読み書きする
    data = binascii.unhexlify(data.rstrip())    # 右端のチェックサムを取り除く
    parsed = ss.unpack(data)                    # 文字列をバイナリとして解釈
                                                # バイトデータの配列として解釈

    # デジタル入力／アナログ入力の値を計算する
    digital = [0] * 4                           # デジタル入力値4ch分
    digitalchanged = [0] * 4                    # デジタル入力変更状態4ch分
    analog = [0xffff] * 4                       # アナログ値4ch分
```

◇受信電文解析処理　4ch分のデータ取り出し

◇デジタル・アナログ各4chあり.

```
for i in range(4):                      # 各4ch分繰り返し
    # デジタル入力
    if parsed[11] & (1 << i):           # バイトデータ配列の11番目が真か？
        digital[i] = 1                  #   真（＝１）ならデジタル入力を１にする
    else:                               # そうでなければ（偽）
        digital[i] = 0                  #   デジタル入力を０にする
    if parsed[12] & (1 << i):           # デジタル入力変更状態が真か？
        digitalchanged[i] = 1           #   真（＝１）ならば１にする
    else:                               # そうでなければ（偽）
        digitalchanged[i] = 0           #   偽（＝０）にする

    # アナログ入力
    if parsed[13 + i] == 0xff :         # アナログ入力値が0xFFならば
        analog[i] = 0xffff              #   max値とする
    else:                               # そうでなければ
        analog[i] = (parsed[13 + i] * 4 + ((parsed[17] >> (2 << i)) & 3)) * 4
                                        #   補正値も含めて元の値を復元する
```

◇受信電文解析処理　結果を【辞書】にする

◇項目にキーを付けた【辞書】.

```
# 結果を返すために、辞書データを作る
result = {
    "from" : parsed[0],                  # 送信元デバイスID
    "lqi" : parsed[4],                   # 受信電波品質
    "fromid" : parsed[5],                # 相手の個体識別番号
    "to": parsed[6],                     # 宛先端末の論理デバイスID
    "timestamp": parsed[7],              # タイムスタンプ
    "isrelay": parsed[8],                # 中継フラグ
    "baterry" : parsed[9],               # 電源電圧
    "digital" : digital,                 # デジタル入力値
    "digitalchanged" : digitalchanged,   # デジタル入力変更状態
    "analog" : analog                    # アナログ入力値
}
return result  # 呼出元に戻る
```

◇処理の本体

```
s = serial.Serial('COM5', 115200)   # COM5を開く
f = 0                               # SWの状態を覚えるフラグ

while 1:   # ずっと繰り返し
    data = s.readline()             # TWE子機から1行の電文受信
    parsed = denbunKaiseki(data) # 受信電文を解析して辞書を作る

    sw = parsed["digital"][0]       # デジタル入力の変化を調べる
    if sw != f:    # SW の状態が前回と変わったか？
        # 変化しているので、メッセージ表示.
        print "changed!!  " + str(f) + "--->" + str(sw)
        f = sw   # SW の状態を記憶

s.close()      # COMを閉じる
```

【重要】上のソースコードで、‘COM5’と書いた部分は、実際に親機を接続した PC での仮想 COM ポート番号に書き換えてください。

ソースコードが完成したら、【pi】という拡張子を付けて保存して下さい。

動作確認のための準備をします。（前講座参照）

◇MoNoStickと子機の準備

◇MoNoStickをPCにセットする.
◇子機の電源をON！！
◇Python PG 実行

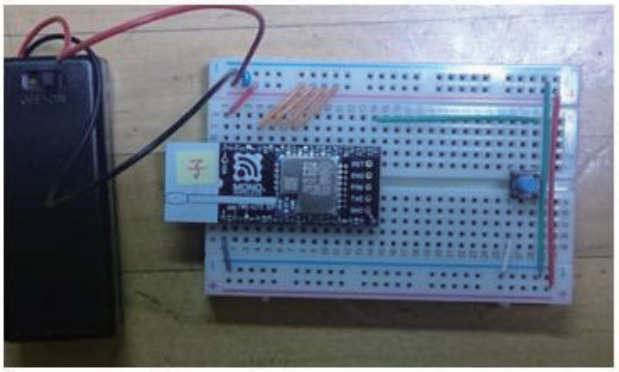

図　8

◇動作確認

◇SWをON/OFF．状態が変化したとき表示.

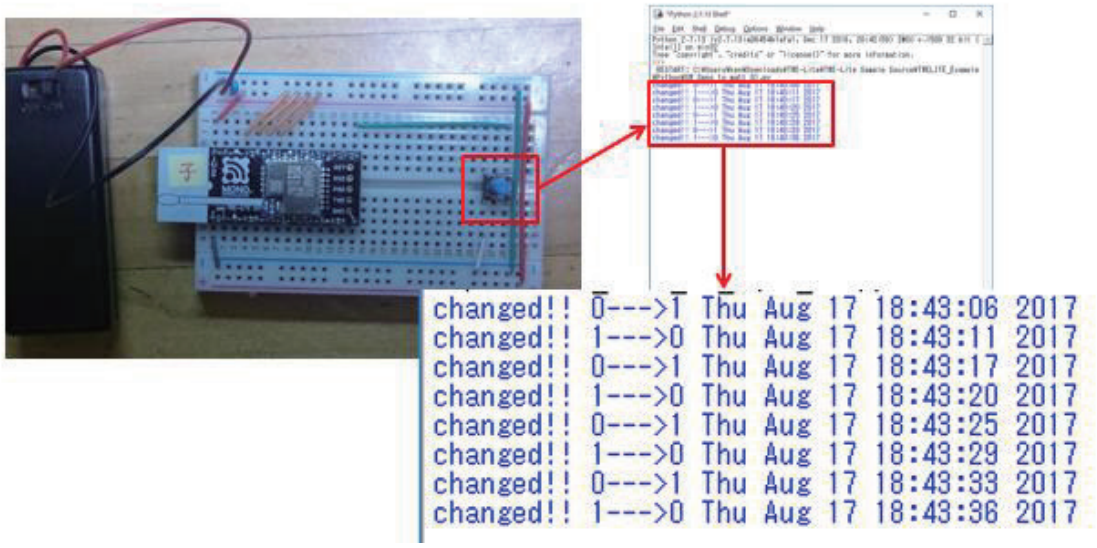

図　9

Python のプログラムを起動して、子機の SW を ON/OFF すると、SW の状態が変化したときだけ、メッセージが表示されます。次回以後の

WEB 連携のために、変化を検出した日付・時刻も表示しています。

これで、いよいよ WEB サービスと連携する準備が整いました。後の講座は本格的な IoT、WEB 連携です。

第13回　WEB 連携①(MQTT)

ついに本格的な IoT に踏み込みます。

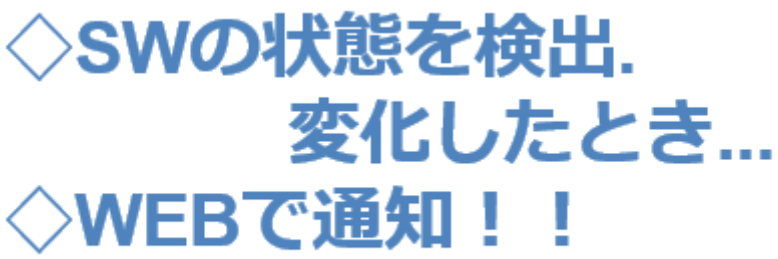

図　10

SW の状態を検出したとき通知するのは、すでに前回（第 12 回）で行いました。今回の講座ではこの通知の部分に WEB サービスを利用します。

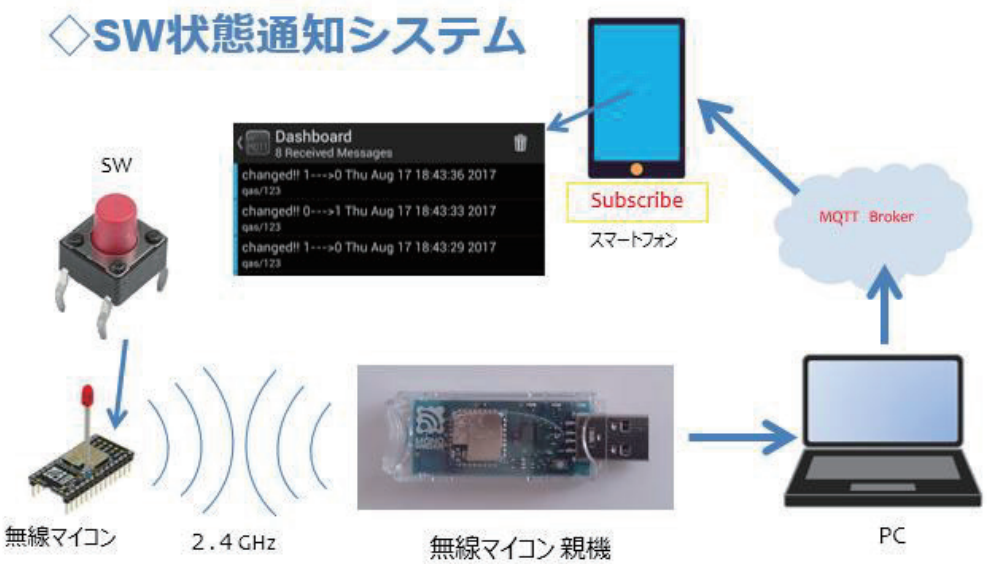

図　11

子機側の SW から、親機、PC までは、第 12 回と同じ構成ですが、今

回はその先があります。PC 内部で Python のプログラミングが動き、SW の状態変化を検出していましたが、検出したタイミングで MQTT という WEB サービスと連携して、メッセージを WEB 上に送ります。送られたメッセージは、MQTT Broker のメッセージ配信サービスにより、購読者（Subscriber）としてあらかじめ登録してあるスマートフォンなどの携帯端末で読みだすことができます。このシステムが稼働すると、いつでも離れたところから SW の状態を監視することができるようになります。まさに【IoT なシステム】となります。

下図はシステムの全体構成です。

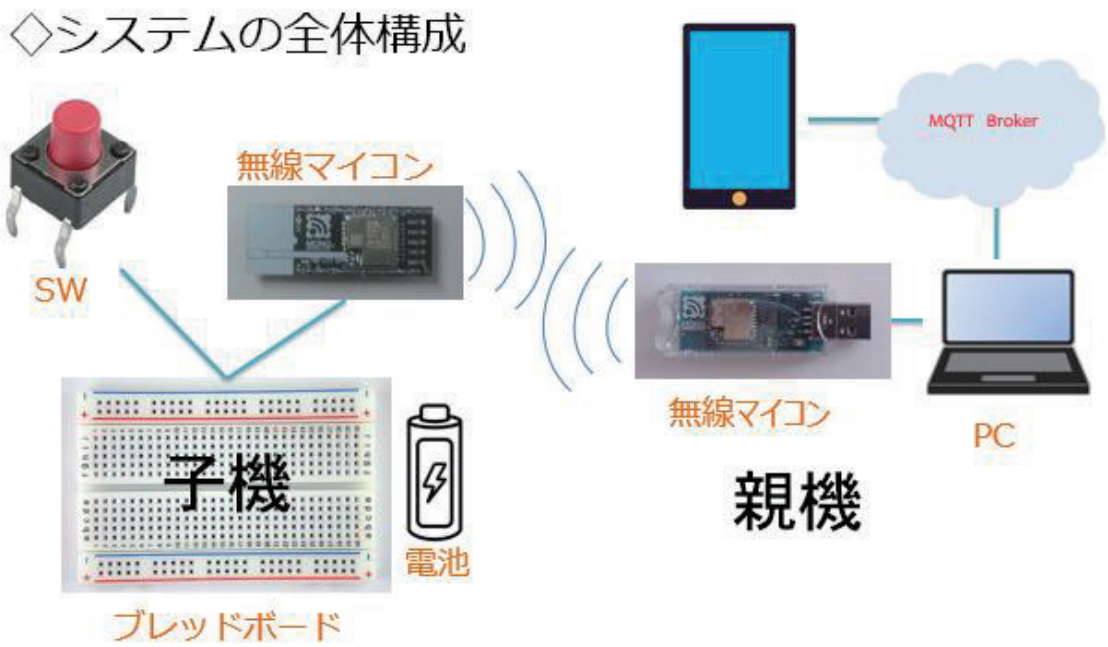

図 12

子機・親機とも第 12 回と同じです。SW の状態変化は PC のウインドウで確認できますが、同時に今回はスマートフォンを利用して WEB に送られた SW の状態変化メッセージを受け取ります。この講座では Android フォンを使いました。PC と携帯端末の間に MQTT という WEB サービスを使用します。

図　13

MQTT は Message Queue Telemetry Transport の略で、短いメッセージを頻繁にやり取りすることに特化したサービスです。

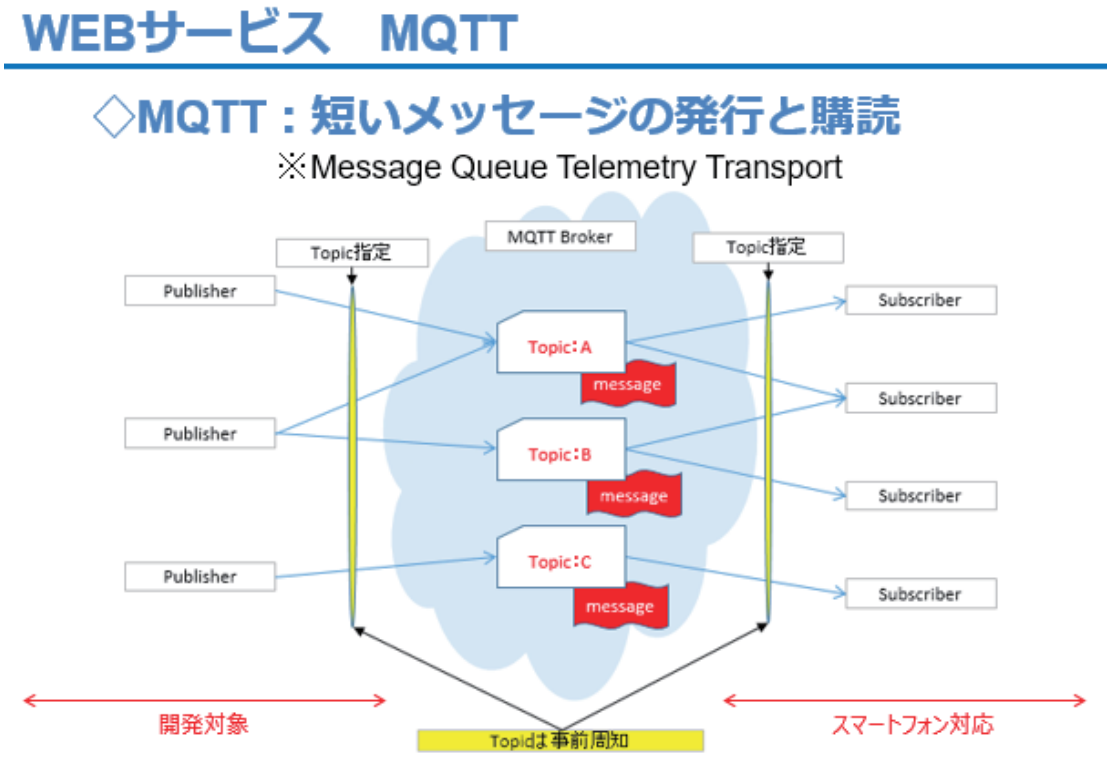

図　14

上図は MQTT の利用方法を描いたものです。

まず、メッセージの発行者側（Publisher）とそのメッセージの読者側（Subscriber）で、特定のメッセージを交換するために Topic というキ

ーワードを決めます。読者側は購読したい Topic をあらかじめ MQTT サービスに登録しておきます。メッセージ発行者がメッセージを Topic と共に MQTT サービスに送ると、MQTT Broker は、その Topic のメッセージ購読希望者に対してメッセージを配信するというものです。実際には、MQTT Broker が登録されている全ての Topic と購読者に対してメッセージを送るのではなく、購読者側の端末プログラムが MQTT Broker に問い合わせをかけて、新しいメッセージが発行されると、それを読み出す処理をしているようです。購読者が何人いても良く、これまで実験したところではタイムラグもほとんど感じません。ユーザ登録も不要で無料利用できる MQTT Broker は、たいへんありがたい存在です。短いメッセージに特化しているからできることなのでしょう。

この講座では、メッセージの発行側（Publisher）を開発対象としています。購読者側（Subscriber）はスマートフォンアプリを利用します。

図 15

この講座では、接続が安定している **broker.hivemq.com** という MQTT Broker を使いました。そして、スマートフォン側は、公開されている MQTT アプリの中で、連続して届くメッセージを見やすい MyMQTT を使用しています。

図　16

MyMQTT は Android 版ですが、他の携帯端末をお使いの方は他の OS 用アプリも公開されていますので探してみてください。アプリをダウンロードしてあらかじめインストールしておいて下さい。ここで、このアプリの使い方を説明しておきます。

図　17

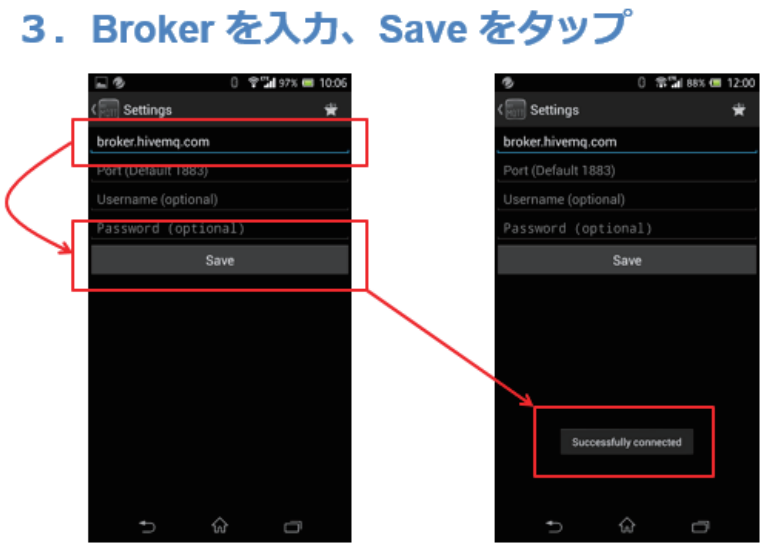

図　18

この時、MQTT Broker に接続が行われます。

図　19

図　20

Subscribe する Topic 名を登録します。トピック名は、マイコンのプログラムで発行するトピック名と一致している必要があります。

図　21

◇Subscribeの様子

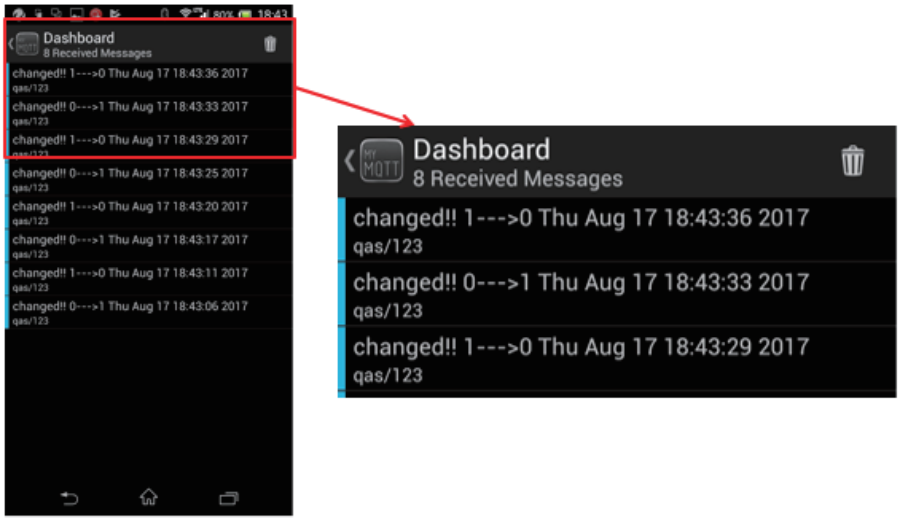

図　22

Subsribe していると、新しいメッセージが Publish される都度、そのメッセージが Dashboard の最上位にスタックされていきます。

今回の子機側の SW 状態を通知する電文は、既に第 12 回で説明しました。

使用するパーツは、下記のとおりです。

子機側：

1. 無線マイコンモジュール×1 台
2. ブレッドボード×1 個
3. 電池（単 4 乾電池×2 個＋SW 付電池ケース×1 個）×1 セット
4. 配線用ジャンパー線
5. SW×1 個

親機側：

1. MoNoSTICK×1 台
2. PC×1 台（Windows10＋Internet 接続）
3. Python 開発・実行環境×1 セット

4. スマートフォン（アプリインストール）

子機の配線は第 1 回（第 12 回）を参照して作成してください。

次に Python プログラムですが、今回は MQTT パッケージを使用しますので、あらかじめインストールしておきます。

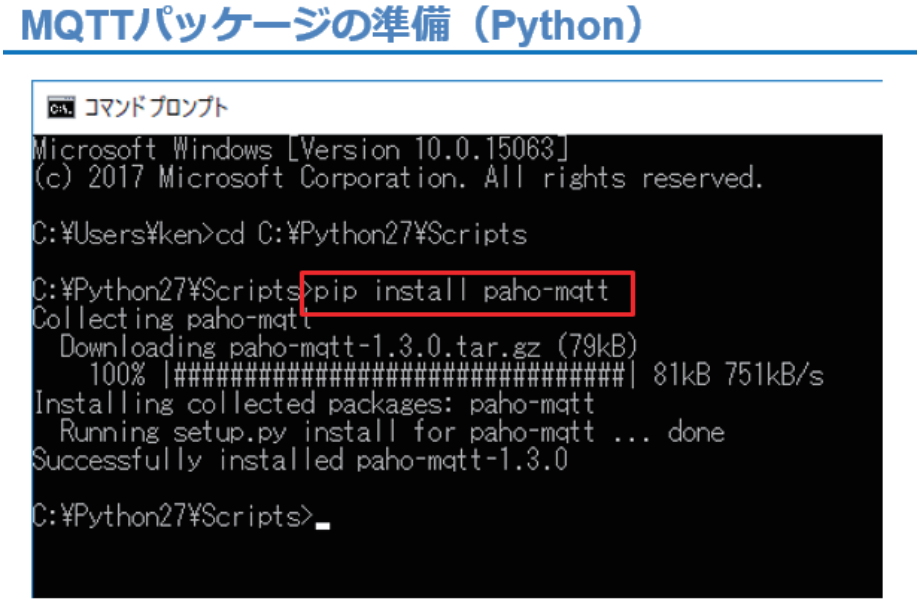

図 23

PC が Internet に接続している状態でコマンドプロンプトに上図の様に

pip install paho-mqtt

と入力して Enter キーを押下します。ダウンロード、インストールが行われて、Successfully～の文字が表示されます。

Python のソースコードを下記します。

◇モジュールなどの取り込み

```
#---------------------------------
#--- 2017.08.17
#--- SWの ON/OFF 状態を調べて、
#--- 変化したときメッセージ表示する
#--- 同時に MQTT でメッセージを送る
#---------------------------------
#--- No.1113
#---------------------------------


import paho.mqtt.client as mqtt    # MQTT パッケージ

import struct      # バイナリ<--->文字列相互変換モジュール
import binascii    # バイナリ<--->ASCII相互変換モジュール
import serial      # シリアル通信パッケージ
import time        # 日付・時刻を取り扱う
```

◇MQTTの準備

◇MQTT Broker接続時の振る舞い

```
# MQTT Broker 接続するときの振る舞い（ここではメッセージを表示するだけ）
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))

# Brokerにメッセージを上げるときの振る舞い（ここでもメッセージを表示するだけ）
def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.payload))
```

◇受信電文解析処理

◇電文（子機→親機）を解析する関数

```
def denbunKaiseki(data):    # 受信電文を解析する
    if data[0] != ":":
        return False        # 先頭が「:」でなければ、対象のデータではない
    data = data[1:]         # 先頭の「:」を取り除く

    # バイトデータに変換する
    ss = struct.Struct(">BBBBBIBHBHBBBBBBBBB")  # フォーマット文字列に従って
                                                # バイナリデータを読み書きする
    data = binascii.unhexlify(data.rstrip())    # 右端のチェックサムを取り除く
    parsed = ss.unpack(data)                    # 文字列をバイナリとして解釈
                                                # バイトデータの配列として解釈

    # デジタル入力／アナログ入力の値を計算する
    digital = [0] * 4                   # デジタル入力値4ch分
    digitalchanged = [0] * 4            # デジタル入力変更状態4ch分
    analog = [0xffff] * 4               # アナログ値4ch分
```

◇受信電文解析処理　4ch分のデータ取り出し

◇デジタル・アナログ各4chあり.

```
for i in range(4):                      # 各4ch分繰り返し
    # デジタル入力
    if parsed[11] & (1 << i):           # バイトデータ配列の11番目が真か？
        digital[i] = 1                  #   真（＝１）ならデジタル入力を１にする
    else:                               # そうでなければ（偽）
        digital[i] = 0                  #   デジタル入力を０にする
    if parsed[12] & (1 << i):           # デジタル入力変更状態が真か？
        digitalchanged[i] = 1           #   真（＝１）ならば１にする
    else:                               # そうでなければ（偽）
        digitalchanged[i] = 0           #   偽（＝０）にする

    # アナログ入力
    if parsed[13 + i] == 0xff :         # アナログ入力値が0xFFならば
        analog[i] = 0xffff              #   max値とする
    else:                               # そうでなければ
        analog[i] = (parsed[13 + i] * 4 + ((parsed[17] >> (2 << i)) & 3)) * 4
                                        #   補正値も含めて元の値を復元する
```

◇受信電文解析処理　結果を【辞書】にする

◇項目にキーを付けた【辞書】.

```
# 結果を返すために、辞書データを作る
result = {
    "from" : parsed[0],                    # 送信元デバイスID
    "lqi" : parsed[4],                     # 受信電波品質
    "fromid" : parsed[5],                  # 相手の個体識別番号
    "to": parsed[6],                       # 宛先端末の論理デバイスID
    "timestamp": parsed[7],                # タイムスタンプ
    "isrelay": parsed[8],                  # 中継フラグ
    "baterry" : parsed[9],                 # 電源電圧
    "digital" : digital,                   # デジタル入力値
    "digitalchanged" : digitalchanged,     # デジタル入力変更状態
    "analog" : analog                      # アナログ入力値
}
return result  # 呼出元に戻る
```

◇処理の本体　前半

◇COMを開き、MQTTの準備.

```
s = serial.Serial('COM5', 115200)   # COM5を開く
f = 0                               # SWの状態を覚えるフラグ

#host = 'test.mosquitto.org'   # 利用するブローカー
host = 'broker.hivemq.com'     # 利用するブローカー
port = 1883                    # 接続するポート番号
topic = 'qas/123'              # MQTT Topic name

client = mqtt.Client()
client.on_connect = on_connect    # mqtt 接続できた時の処理
client.on_message = on_message    # mqtt メッセージが配信されたときの処理
client.connect(host, port=port, keepalive=60)   # mqtt broker 接続
```

【重要】上のソースコードで、‘COM5’と書いた部分は、実際に親機を接続した PC での仮想 COM ポート番号に書き換えてください。

◇処理の本体　後半

◇COMを開き、MQTTの準備.

```
while 1:   # ずっと繰り返し
    data = s.readline()           # TWE子機から1行の電文受信
    parsed = denbunKaiseki(data) # 受信電文を解析して辞書を作る

    # デジタルの変化を調べる
    sw = parsed["digital"][0]    # デジタル入力の0番目がSW.
    if sw != f:   # SW は前回と状態が変化？
        # 変化しているので、メッセージ表示.
        print "changed!!  " + str(f) + "--->" + str(sw) + "  " + time.ctime()
        # mqtt 用メッセージを作製 (topicの内容)
        msg = "changed!!  " + str(f) + "--->" + str(sw) + "  " + time.ctime()
        #client.publish(topic, msg, 0, True)
        client.publish(topic, msg)   # mqtt broker にtopic を 発行
        f = sw   # SW の状態を記憶

s.close()     # COMを閉じる
```

◇プログラムが完成したら、動作確認を行います。Python のプログラムを起動して、スマーフォンアプリで Dashboard を開くと、SW の ON/OFF に対応して、新たなメッセージが表示されます。

動作確認

◇SWをON/OFF．状態が変化の表示.

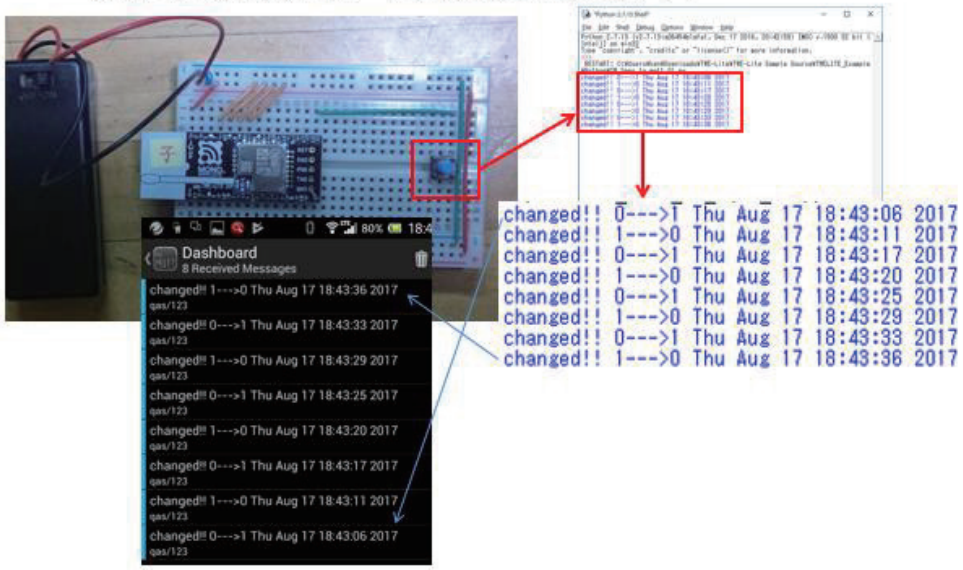

図　24

第14回　WEB 連携②(MQTT)

前回は、SWの状態変化を検出して、その状況をメッセージとしてWEB経由で、遠隔地からモニター出来るシステムを開発しました。この SWの状態が変化することを検出する代わりに、温度を計測する。そして、ON/OFF などのメッセージではなく、計測値をメッセージにしてPublish すれば、現場で実際に使えるリモート温度センサーのシステムが開発できます。一連の講座の集大成として、今回のテーマは、無線による温度通知システムです。

図　25

図 26

無線による温度通知システムといっても、前回のシステムとあまり変わらない構成で開発できます。第 13 回の SW の代わりに温度センサーを使用します。この温度センサーは、第 9 回、第 11 回で使用したものです。そして子機で温度を表示できるように液晶表示器を使います。その他は前回、第 13 回と同じ構成です。液晶表示器は、第 10 回、第 11 回で使用しました。

情報の流れとしては、次の様になります。まず温度センサーで温度を計測します。計測した温度に対応する電圧が無線通信の電文に収容されて子機から親機へと送信されます。親機は PC と接続されていて、PC 内部で稼動する Python プログラムで、通信電文を受け取り、その内容を解析します。計測電圧から温度を計算して、PC の画面に表示するとともに、子機への通信電文を作成して送り返します。子機側は親機からの通信電文に載っている温度表示のメッセージを液晶表示器に表示します。親機は、子機への電文送信を行うとともに、MQTT Broker にも温度の

値をメッセージとして Publish（発行）します。あらかじめ、購読 Topic を登録していたスマートフォンアプリで Dashboard を見ていると、刻々と変化する温度の様子が、メッセージとして Subscribe（購読）される。という仕組みです。温度計測は子機側のタイミングで繰り返し行われるので、スマートフォン側で見ていると、ほぼ一定間隔で温度が表示されます。この結果、センサーのある場所（子機）でも PC の画面（親機側）でも、離れた場所（スマートフォン）でも温度の変化を観察できるシステムとなります。

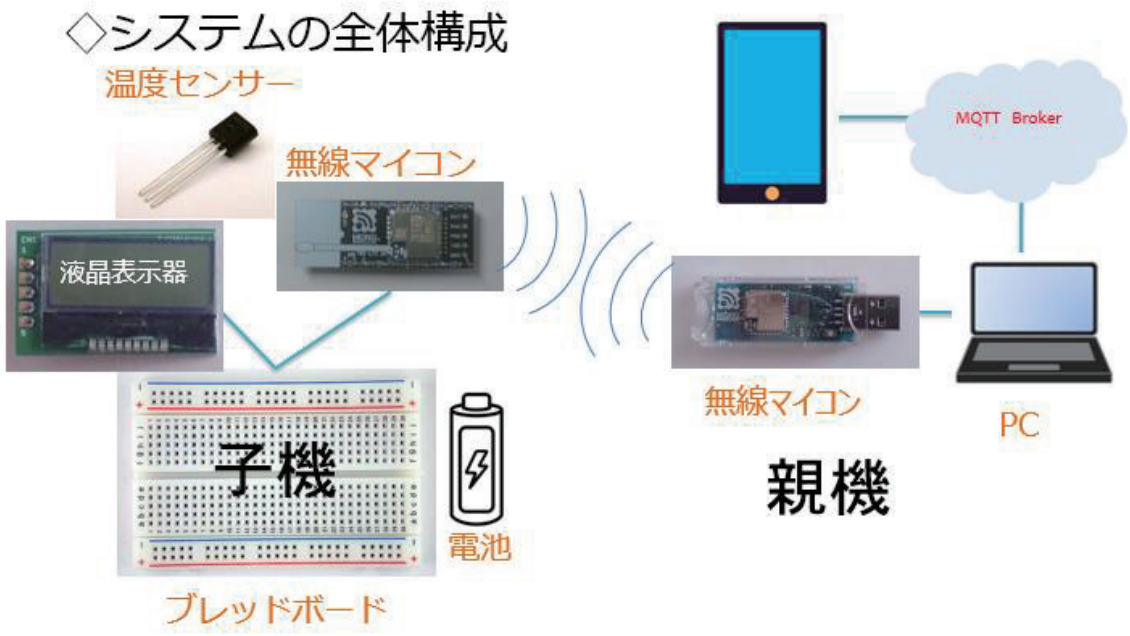

図 27

全体の構成は上図の様になります。子機も親機も、第 11 回のデジタル温度計の開発と同じです。第 11 回から時間が経ってしまった方は、もう一度、第 11 回を振り返っておくと良いと思います。WEB サービスの MQTT については、仕組みや使い方を第 13 回で解説しました。

図　28

利用できる MQTT Broker は沢山ありますが、登録などが不要で無料で使用でき、接続が安定している broker.hivemq.com を使いました。

図　29

スマートフォンアプリは、MyMQTT（Android 版）を使用します。メッセージを Subscribe する画面表示が見易いのが特徴です。使用方法は第 13 回で解説しました。温度センサー計測値は、下図の電文に利用されて親機に送られます。第 9 回で解説しました。

データ受信コマンド（センサー検出値）

◇先頭はコロン【：】で始まるテキストデータ

図　30

I2Cデータ書込みコマンド（LCD表示）

◇先頭はコロン【：】で始まるテキストデータ

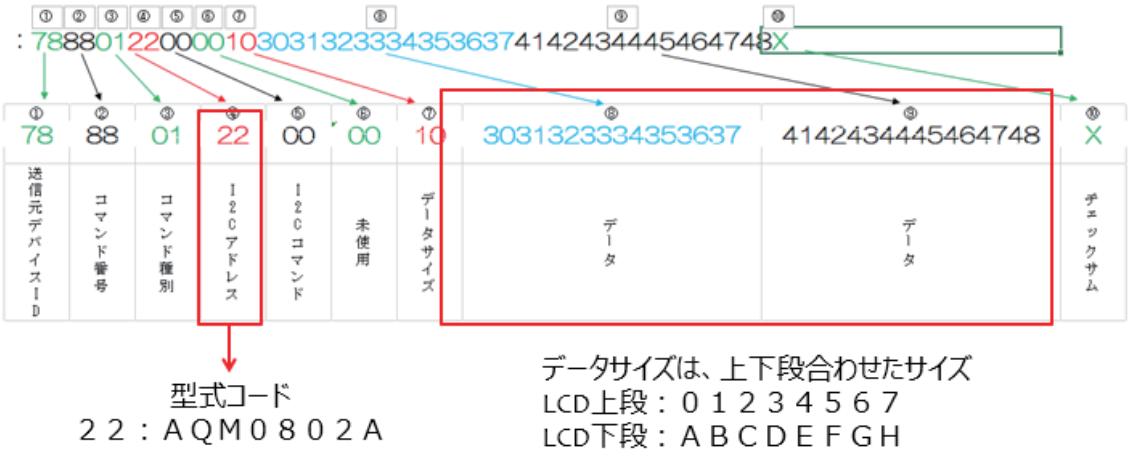

図　31

子機の液晶表示器への表示は、上図の電文によって行います。こちらは第 10 回で解説しました。温度センサーと液晶表示器の詳細仕様につ

いても第 9 回、第 10 回で解説しましたので参照して下さい。

今回使用するパーツは以下の通りです。

子機側：

1. 無線マイコンモジュール×1 台
2. ブレッドボード×1 個
3. 電池（単 4 乾電池×2 個＋SW 付電池ケース×1 個）×1 セット
4. 配線用ジャンパー線
5. LCD（AQM0802A）×1 個
6. 温度センサー（LM61CIZ）×1 個

親機側：

1. MoNoSTICK×1 台
2. PC×1 台（Windows10＋Internet 接続）
3. Python 開発・実行環境×1 セット
4. スマートフォン×1 台（アプリインストール）

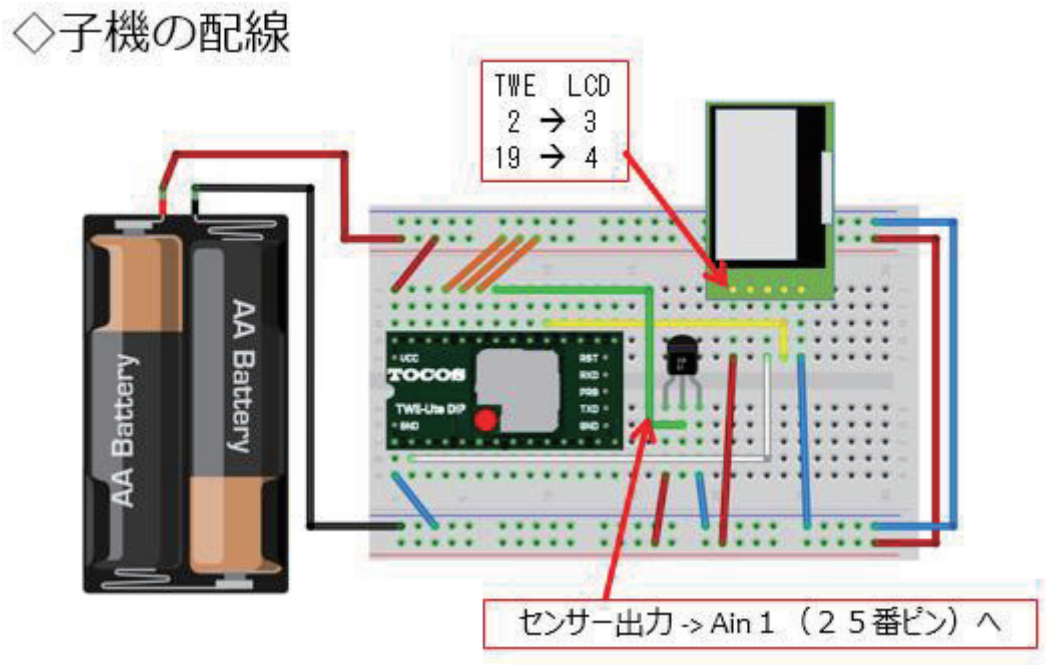

図 32

子機は、第 11 回で開発したデジタル温度計と全く同じものです。実際に配線した様子は下の写真のようになっています。

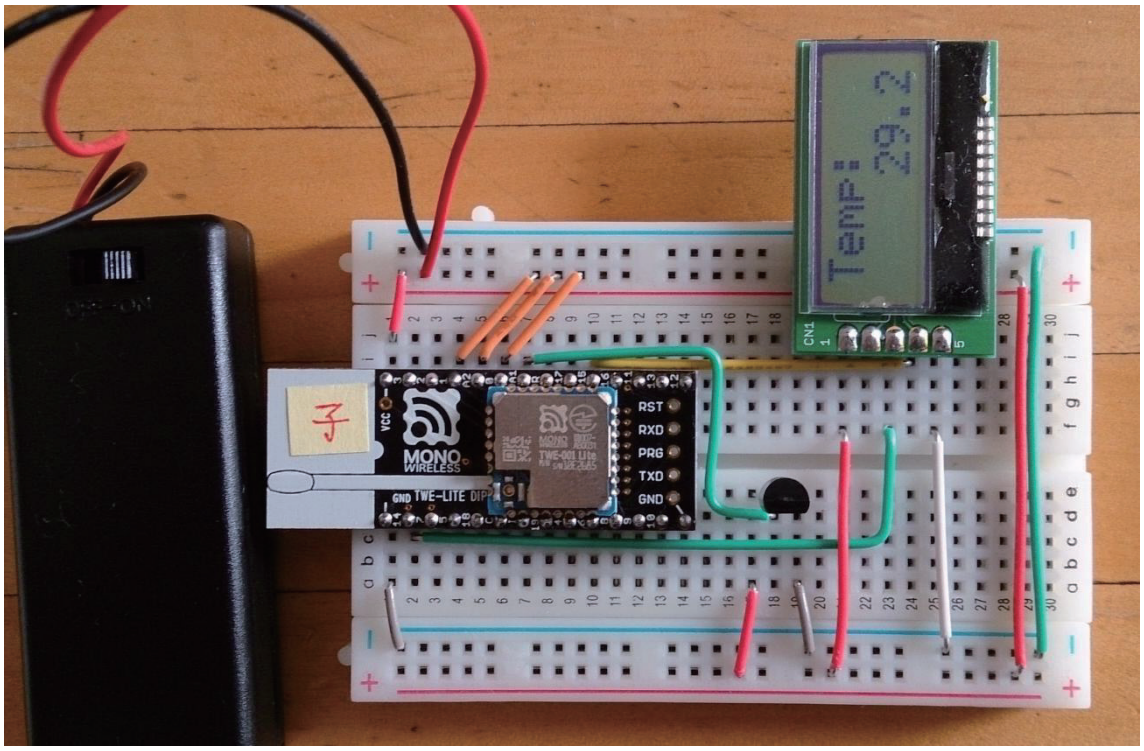

図 33

以下に Python のソースコードを示します。第 13 回と同様に MQTT パッケージを使います。インストールが済んでいない方は、第 13 回を参照して準備してください。

◇モジュールなどの取り込み

◇モジュール、パッケージなどの取り込み

```
#---------------------------------
#--- 2017.08.30
#--- Ondo + LCD(AQM0802A) + MQTT
#--- 温度センサー計測値をLCD表示する
#--- 同時に MQTT でメッセージを送る
#---------------------------------
#--- No.1114
#---------------------------------

import paho.mqtt.client as mqtt    # MQTT パッケージ

import struct      # バイナリ<--->文字列相互変換モジュール
import binascii    # バイナリ<--->ASCII相互変換モジュール
import serial      # シリアル通信パッケージ
import time        # 日付・時刻データを取り扱う
```

◇MQTTの準備

◇MQTT Broker接続時の振る舞い

```
# MQTT Broker 接続するときの振る舞い（ここではメッセージを表示するだけ）
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))

# Brokerにメッセージを上げるときの振る舞い（ここでもメッセージを表示するだけ）
def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.payload))
```

◇LCD制御電文送信　序盤

◇LCDを制御　序盤

```
# I2Cを制御する関数
def I2CLCD(s, sendto = 0x78, reqno = 0x00, command = 0x01,
        i2caddress = 0x00, i2ccommand = 0x00,
        data = [], readbyte = -1):

        # データを作成する
        if readbyte == -1:
                # dataを書き込む
                sendbytes = [sendto, 0x88, reqno,  i2caddress, i2ccommand, 0x00, len(data)]
                # dataを加える
                # dataを1文字ずつリストにする
                buf = list(data)
                # buffを文字コードの数値に変換する
                buff = map(ord,buf)
                # 送信データの後ろに、表示するメッセージの文字コードのリストを追加する
                sendbytes.extend(buff)
        else:
                # readbyteだけ読み取る（dataは利用しない）
                sendbytes = [sendto, 0x88, reqno,  i2caddress, i2ccommand, 0x00, readbyte]
```

◇LCD制御電文送信　中盤　電文完成～送信

```
# 16進数文字列に変換する
bytelen = len(sendbytes)

ss = struct.Struct(str(bytelen) + "B")
outstring = binascii.hexlify(ss.pack(*sendbytes)).upper()

# TWE-Lite子機に送信する
# チェックサム省略バージョンなので、"X"を追加しておく
s.write(":" + outstring + "X" + "¥r¥n")
```

◇LCD制御電文送信　終盤　子機応答チェック

```
# 応答を待つ
# 10回繰り返す
for i in range(10):
        status = s.readline()
        if status[0:9] == ":" + outstring[0:2] + "89" + outstring[4:8]:
                # 対応する応答結果が戻った
                status = status[1:].rstrip()      # 行頭の「:」と行末の改行を取り除く
                ss = struct.Struct(">BBBBBB")     # バイトデータに変換する
                parsed = ss.unpack(binascii.unhexlify(status[0:12]))
                if status[4]:
                        # I2Cへのアクセスに成功
                        # 戻り、１バイトを返す
                        ss = struct.Struct(str(parsed[5]) + "B")
                        result = ss.unpack(binascii.unhexlify(status[12:len(status) - 2]))
                        return result
                else:
                        # 失敗のときは、偽
                        return False
                break
return False
```

◇受信電文解析処理

◇電文（子機→親機）を解析する関数

```
def denbunKaiseki(data):    # 受信電文を解析する
    if data[0] != ":":
        return False        # 先頭が「:」でなければ、対象のデータではない
    data = data[1:]         # 先頭の「:」を取り除く

    # バイトデータに変換する
    ss = struct.Struct(">BBBBBIBHBHBBBBBBBBB")  # フォーマット文字列に従って
                                                # バイナリデータを読み書きする
    data = binascii.unhexlify(data.rstrip())    # 右端のチェックサムを取り除く
    parsed = ss.unpack(data)                    # 文字列をバイナリとして解釈
                                                # バイトデータの配列として解釈

    # デジタル入力／アナログ入力の値を計算する
    digital = [0] * 4                           # デジタル入力値4ch分
    digitalchanged = [0] * 4                    # デジタル入力変更状態4ch分
    analog = [0xffff] * 4                       # アナログ値4ch分
```

◇受信電文解析処理　4ch分のデータ取り出し

◇デジタル・アナログ各4chあり.

```
for i in range(4):                      # 各4ch分繰り返し
    # デジタル入力
    if parsed[11] & (1 << i):           # バイトデータ配列の11番目が真か？
        digital[i] = 1                  #   真（＝１）ならデジタル入力を１にする
    else:                               # そうでなければ（偽）
        digital[i] = 0                  #   デジタル入力を０にする
    if parsed[12] & (1 << i):           # デジタル入力変更状態が真か？
        digitalchanged[i] = 1           #   真（＝１）ならば１にする
    else:                               # そうでなければ（偽）
        digitalchanged[i] = 0           #   偽（＝０）にする

    # アナログ入力
    if parsed[13 + i] == 0xff :         # アナログ入力値が0xFFならば
        analog[i] = 0xffff              #   max値とする
    else:                               # そうでなければ
        analog[i] = (parsed[13 + i] * 4 + ((parsed[17] >> (2 << i)) & 3)) * 4
                                        #   補正値も含めて元の値を復元する
```

◇受信電文解析処理　結果を【辞書】にする

◇項目にキーを付けた【辞書】.

```
# 結果を返すために、辞書データを作る
result = {
    "from" : parsed[0],                 # 送信元デバイスID
    "lqi" : parsed[4],                  # 受信電波品質
    "fromid" : parsed[5],               # 相手の個体識別番号
    "to": parsed[6],                    # 宛先端末の論理デバイスID
    "timestamp": parsed[7],             # タイムスタンプ
    "isrelay": parsed[8],               # 中継フラグ
    "baterry" : parsed[9],              # 電源電圧
    "digital" : digital,                # デジタル入力値
    "digitalchanged" : digitalchanged,  # デジタル入力変更状態
    "analog" : analog                   # アナログ入力値
}
return result  # 呼出元に戻る
```

◇2行のメッセージをまとめて送信

◇LCDの上段・下段のメッセージをまとめて
１回で送信

```
# AQM0802A-RN-GBWに文字列を出力する
def writeAQM0802Msg(s, msg1, msg2):
    # 初期化は、標準アプリ側で行っている

    # 出力文字列を１つにまとめる
    msg = msg1.ljust(8)+msg2.ljust(8)
    # 子機に送信
    I2CLCD(s, i2caddress = 0x22, i2ccommand = 0x80, data=list(msg))
    return
```

◇処理の本体　前半

◇COMを開き、MQTTの準備.

```
s = serial.Serial('COM5', 115200)    # COM5を開く
writeAQM0802Msg(s, "", "")           # LCDを消去する
n = 0                                # 測定回数
avr = 0                              # 平均温度

#host = 'test.mosquitto.org'         # 利用するブローカー
host = 'broker.hivemq.com'           # 利用するブローカー
port = 1883                          # 接続するポート番号
topic = 'qas/123'                    # MQTT Topic name

client = mqtt.Client()
client.on_connect = on_connect       # mqtt 接続できた時の処理
client.on_message = on_message       # mqtt メッセージが配信されたときの処理
client.connect(host, port=port, keepalive=60)   # mqtt broker 接続
```

※ブローカーと通信を行っている間の最大時間（秒）

【重要】上のソースコードで、‘COM5’と書いた部分は、実際に親機を接続した PC での仮想 COM ポート番号に書き換えてください。

◇処理の本体　後半

```
while 1:   # 繰り返し
    data = s.readline()              # 1行受信
    parsed = denbunKaiseki(data)     # 1行を分析して、項目ごとの値を求める
    t = (parsed["analog"][0] - 600.0) / 10.0   # 電圧を温度に変換する
    print t   # 温度、画面表示

    n += 1    # 加算<---平均の為
    avr += t

    if n >= 10:   # 平均計算
        avr /= n
        n = 0

        writeAQM0802Msg(s, "Temp:", "    %2.1f" % avr)   # データを出力する

        # mqtt 用メッセージを作製 (topicの内容)
        msg = "ondo---> %2.1f %s" % (avr, time.ctime())
        print msg
        client.publish(topic, msg)   # mqtt broker にtopic を 発行
        avr = 0

s.close()   # COMを閉じる
```

ソースコードが完成したら、【pi】という拡張子を付けて保存して下さい。

動作確認のための準備をします。（前講座参照）

子機、親機の準備が整ったら、Python のプログラムを実行します。

動作確認の要点は次の 3 つです。

1. PC のウインドウに温度データが表示されるか。

2. 液晶表示器に温度が表示されるか。

3. スマートフォンアプリで MQTT Broker のメッセージが逐次更新されるか。

以上を確認してください。

MonoStickと子機の準備

◇MonoStickをPCにセットする.
◇子機の電源をON！！
◇Python PG 実行

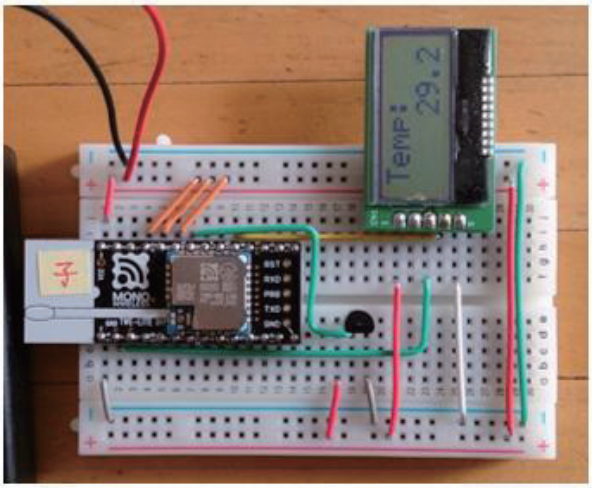

図　34

動作確認

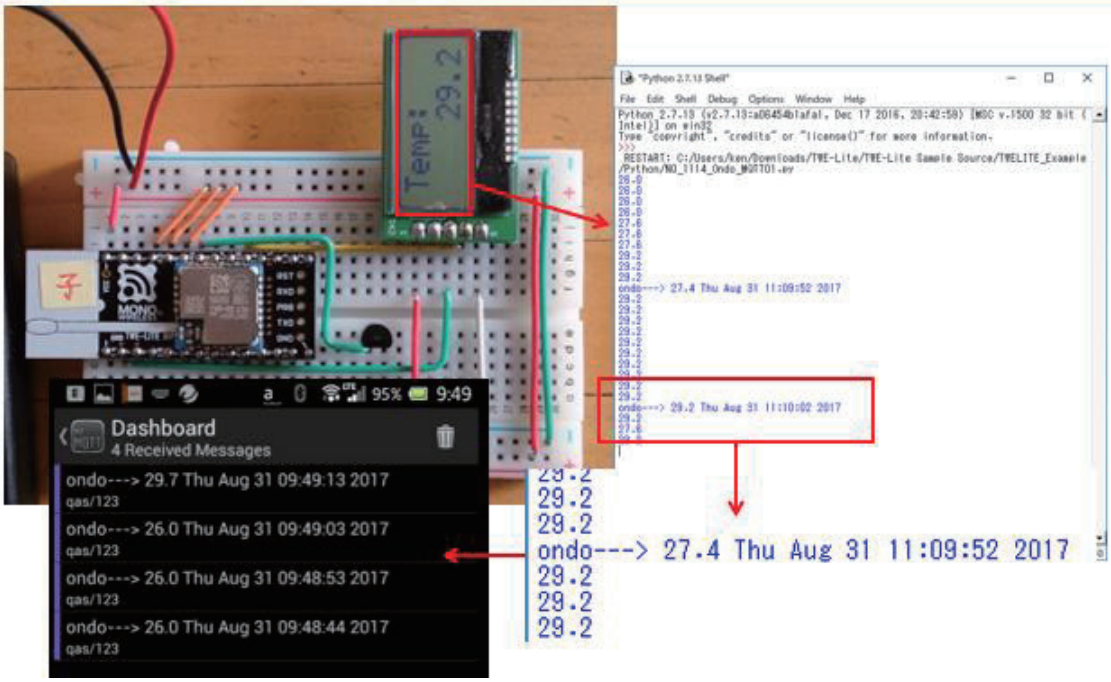

図　35

【WiFiマイコン(ESP-8266)編】

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。

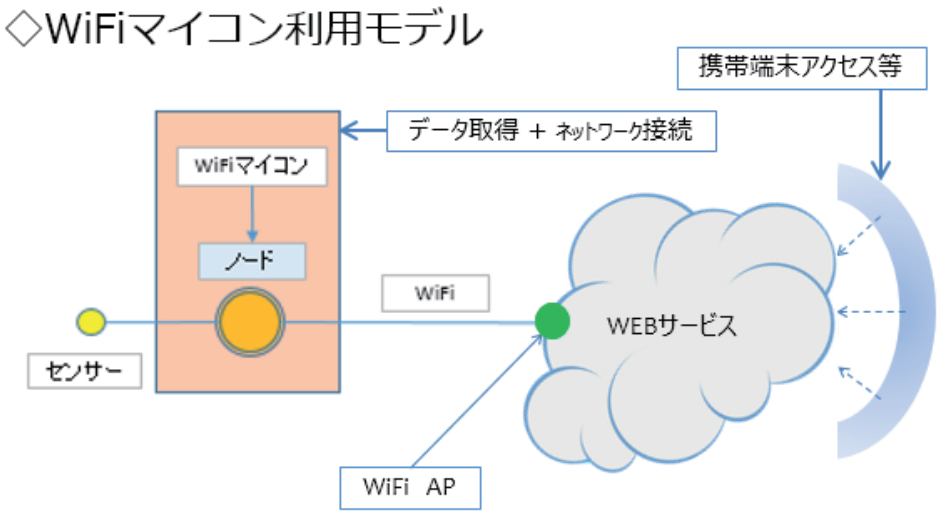

図　36

上図は、WiFi マイコンを利用した WEB サービス連携モデルです。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。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できます。

「これぞ IoT！！」と言えます。

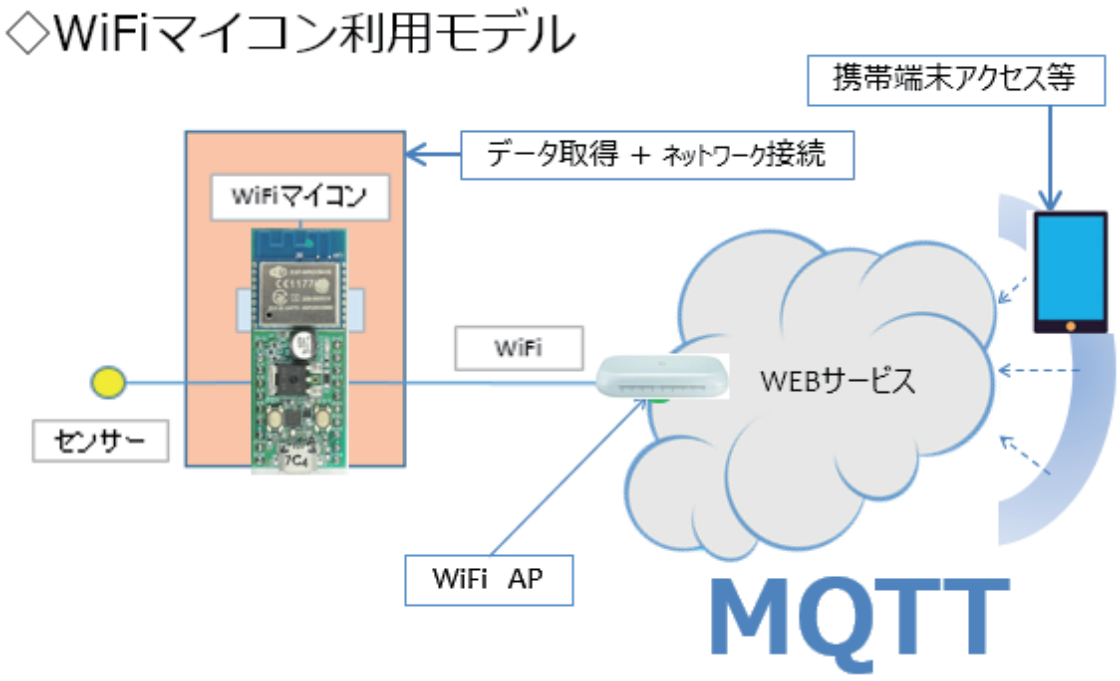

図 37

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 を利用します。

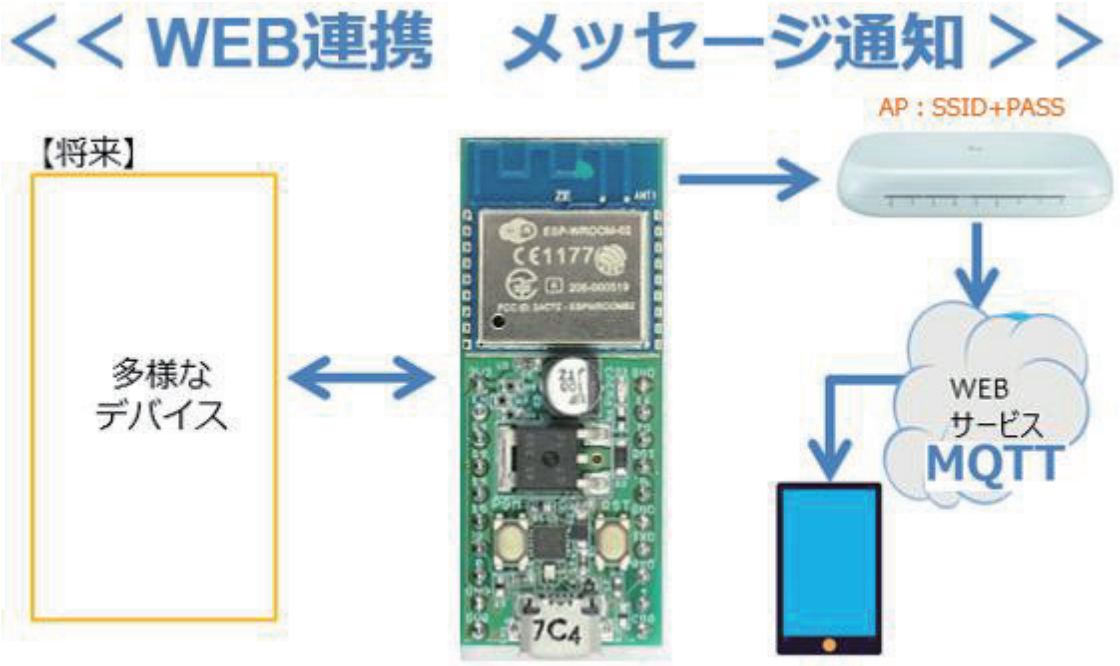

図　38

概要を示したものが上図です。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、**以後の講座では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携帯端末（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、上図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開発します。

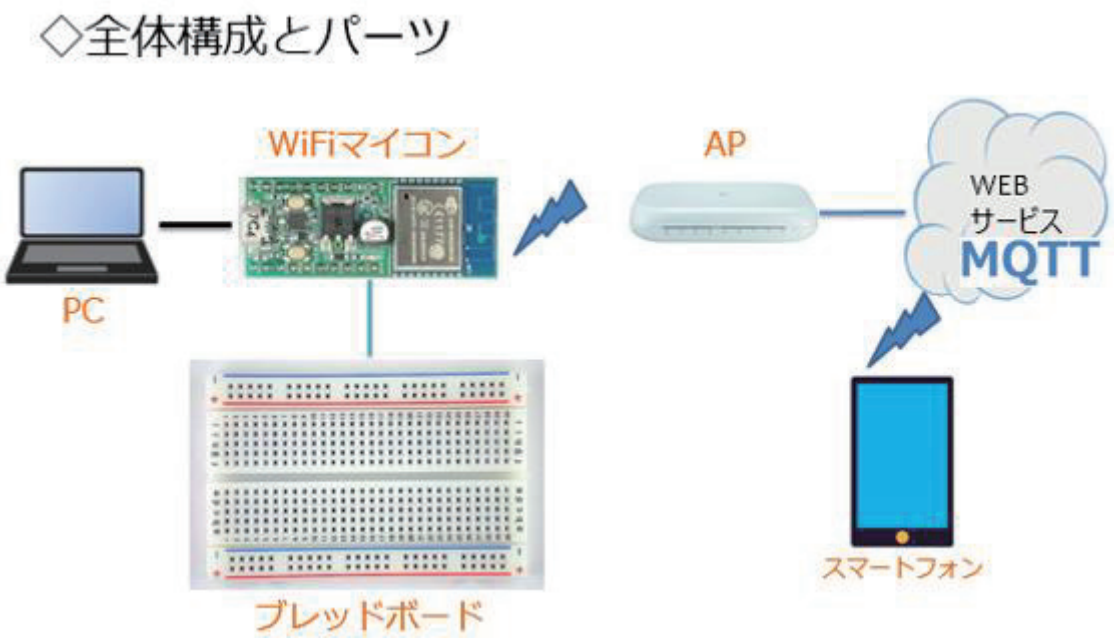

図 39

全体の構成は上図の通りです。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

※以下の MQTT とスマホアプリについては、【無線マイコン（ESP-8266）編】第 13 回で解説した内容を再掲しています。理解されている方は、先に読み進めて下さい。

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。下図で MQTT Broker を通じてメッセージの交換を行う様子を示します。

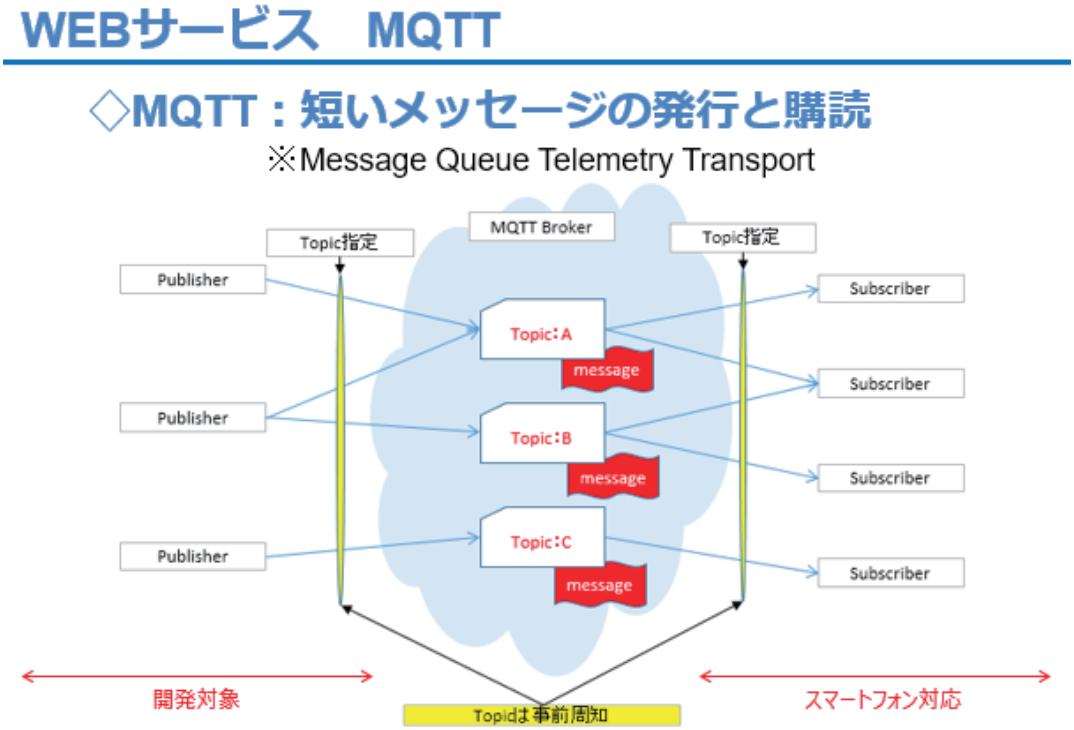

図　40

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に【Topic】を付けて 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、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　41

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 42

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。

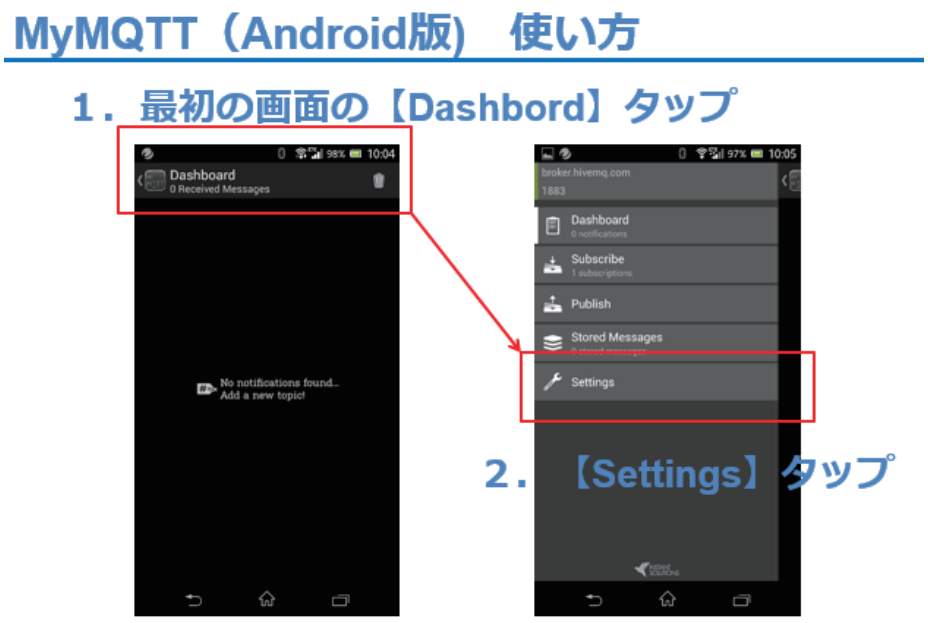

図 43

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。

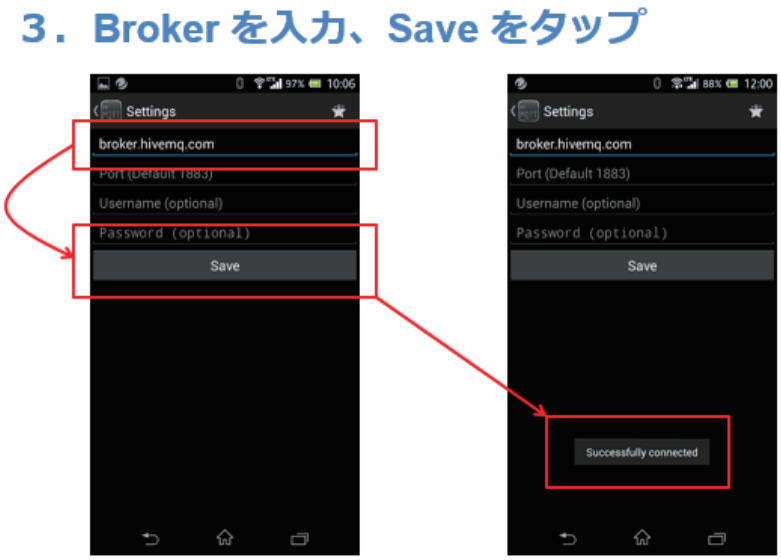

図 44

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 45

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。

【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。

図 46

６．Topic を入力して Add をタップして登録します。（図の７，８）

図　47

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。

使い方の細かな点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

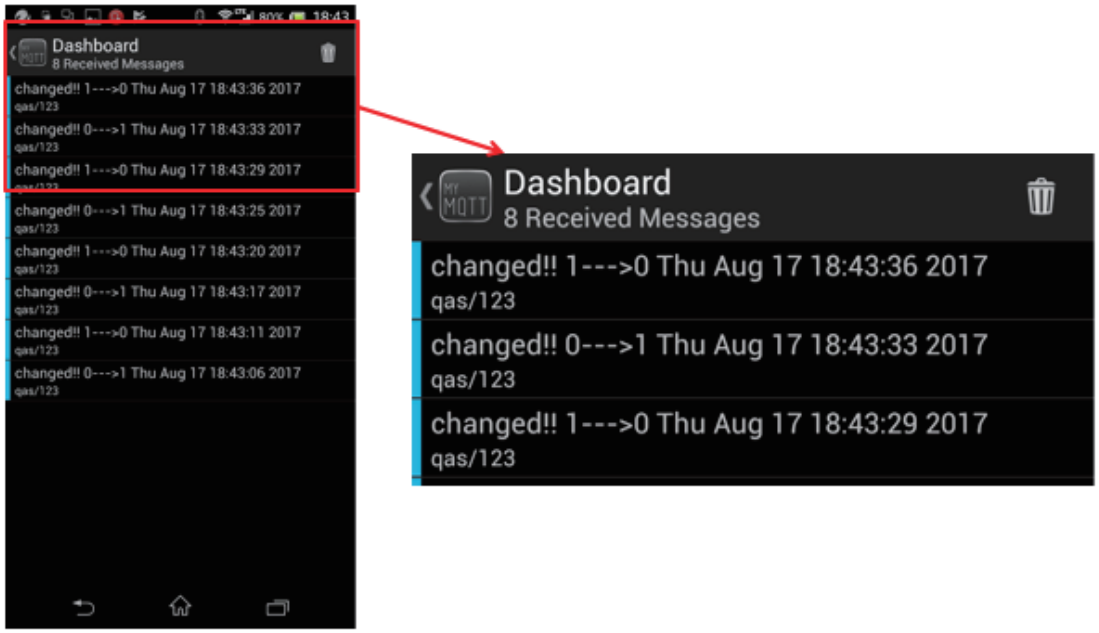

図　48

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

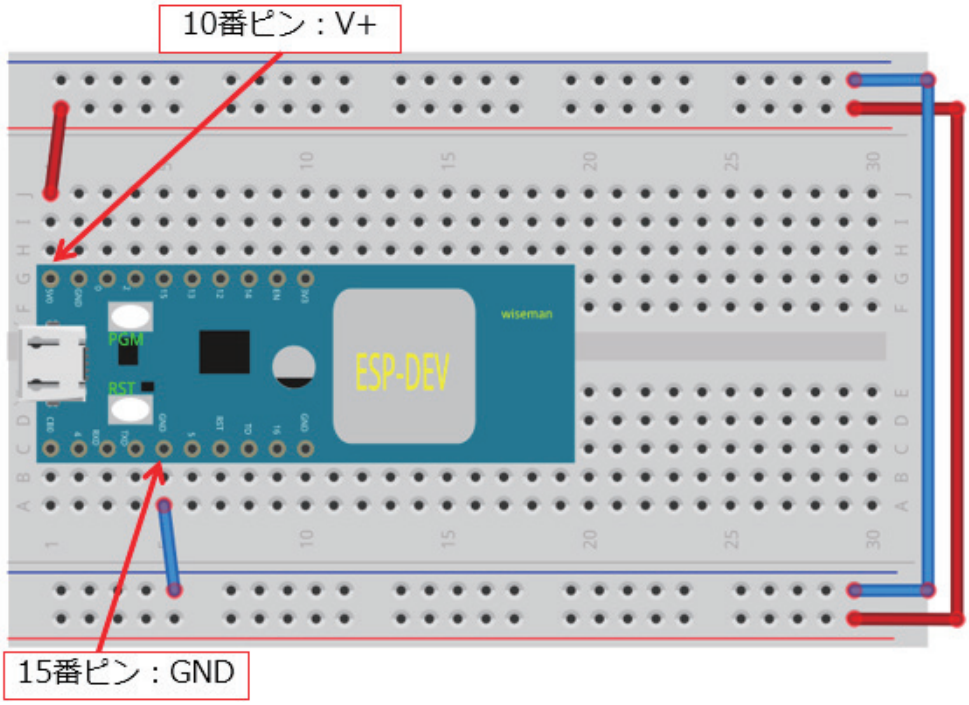

図　49

実際に配線した様子

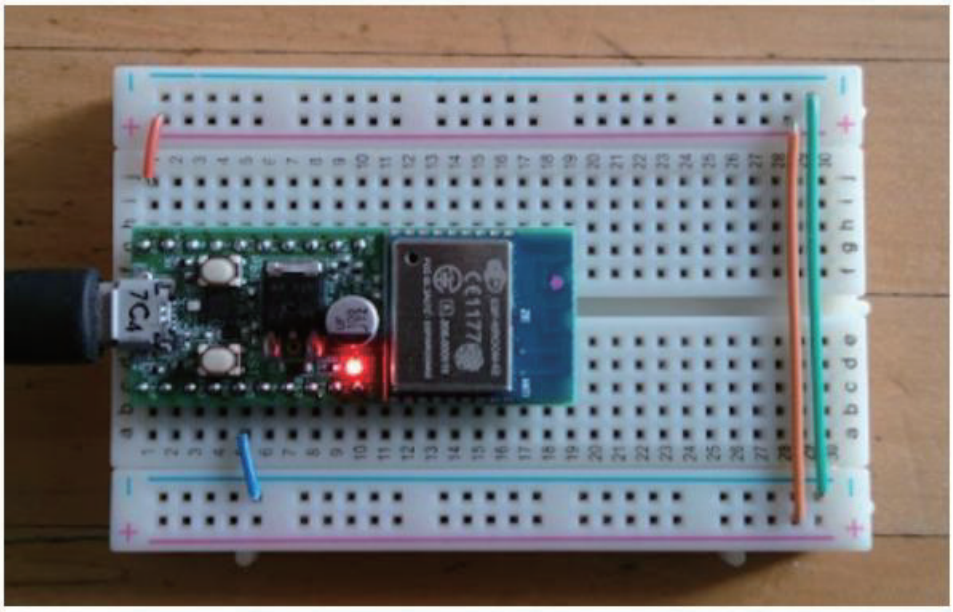

図　50

MQTTライブラリの準備

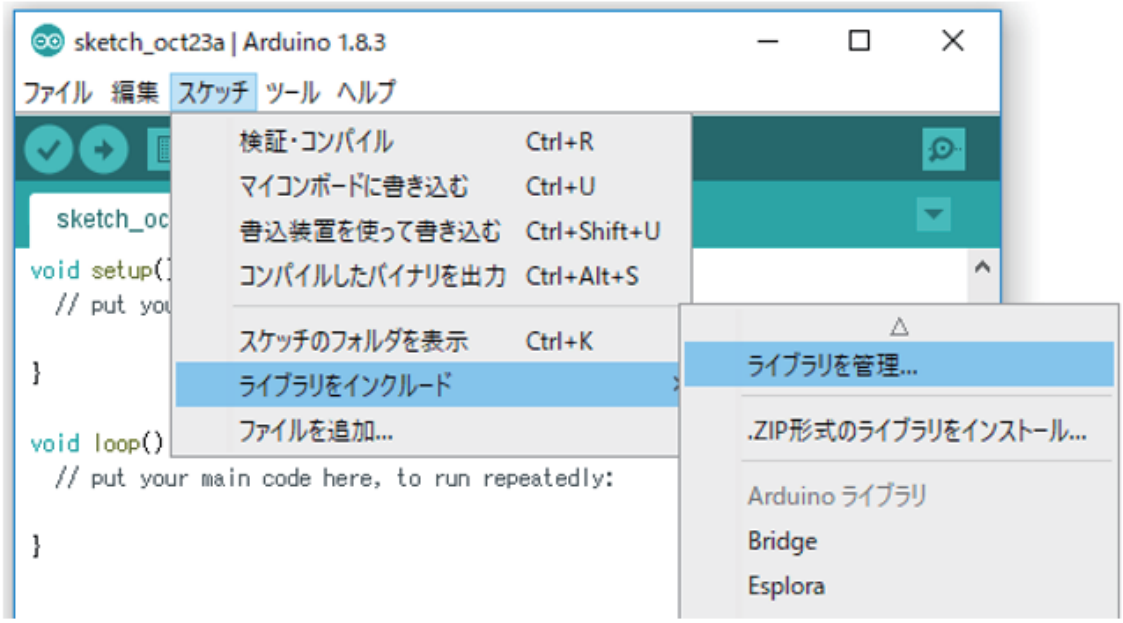

図 51

今回は、ソフトウエア開発に必要なMQTTライブラリをここで準備します。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。

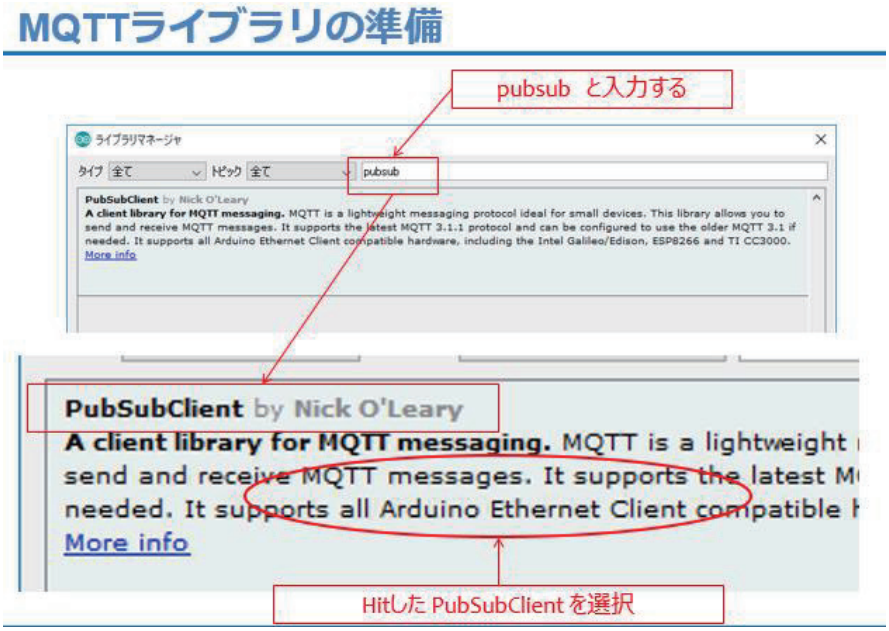

図　52

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。

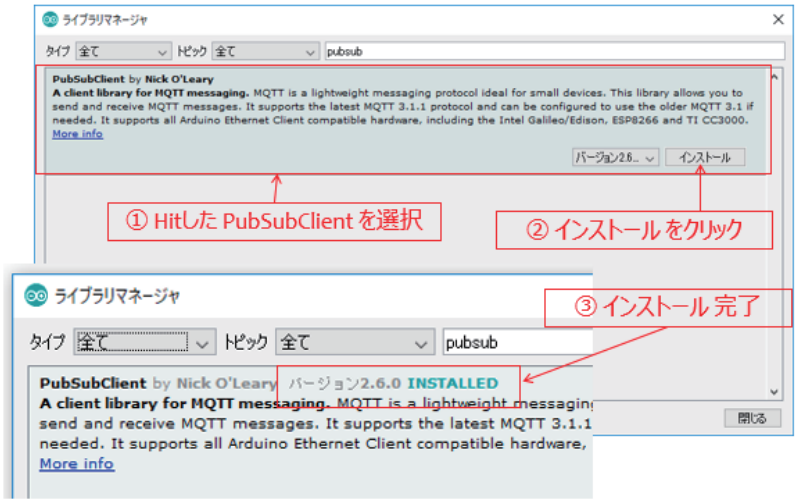

図　53

MQTTライブラリの準備

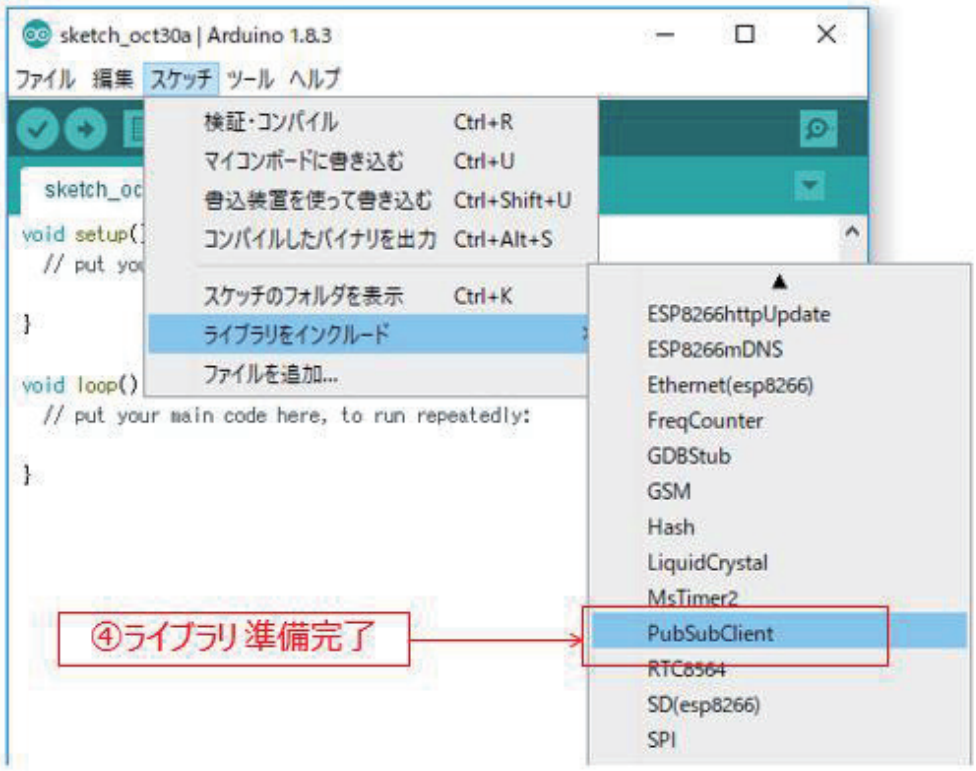

図 54

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。

プログラムを書く

ESP_2110_MQTT_1_publish

```
#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.

const char* ssid = "SSID";        ← ③ アクセスポイント SSID.
const char* pass = "PASS";        ← ④ アクセスポイント Pass Word.
const char* broker = "broker.hivemq.com"; ← ⑤ MQTT Broker.
const int port = 1883;            ← ⑥ MQTT 接続ポート番号 固定.
const char* topic = "Topic";      ← ⑦ Topic名 スマートフォンアプリで登録.

WiFiClient espClient;             ← ⑧ WiFi接続オブジェクト.
PubSubClient client(espClient);   ← ⑨ MQTT Client オブジェクト.
long lastMsg = 0;                 ← ⑩ メッセージ発行時刻.
char msg[50];                     ← ⑪ メッセージ編集用バッファ.
int value = 0;                    ← ⑫ メッセージ回数カウンター.
```

図 55

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

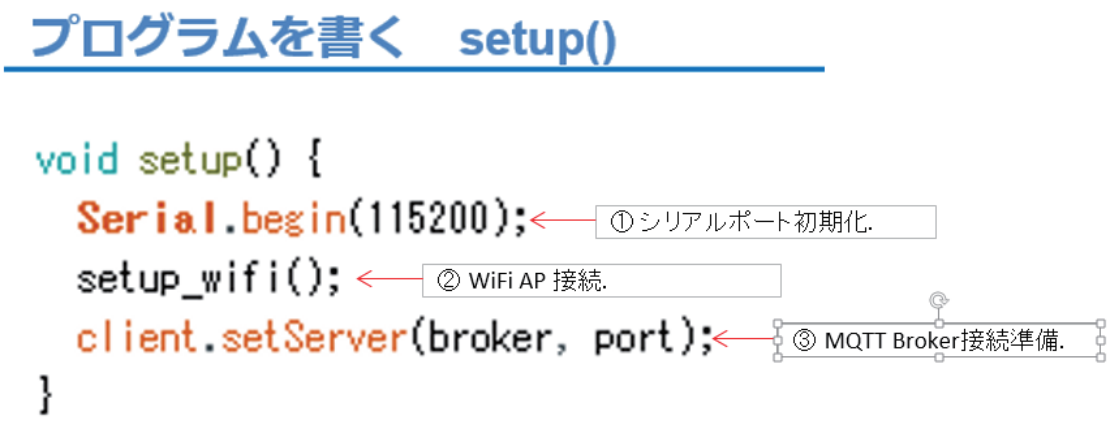

図　56

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　57

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　58

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

プログラムを書く　reconnect()

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {          ← ① MQTT 接続確認.
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("","","")) {      // (clientID, username, password)  ← ② MQTT 接続実行.
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish(topic, "hello world");  ← ③ 初メッセージ発行.
      // ... and resubscribe
      //client.subscribe(mqtt_sub_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");   ← ④ PCに状況通知.
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

※ファイル→名前を付けて保存

図　59

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

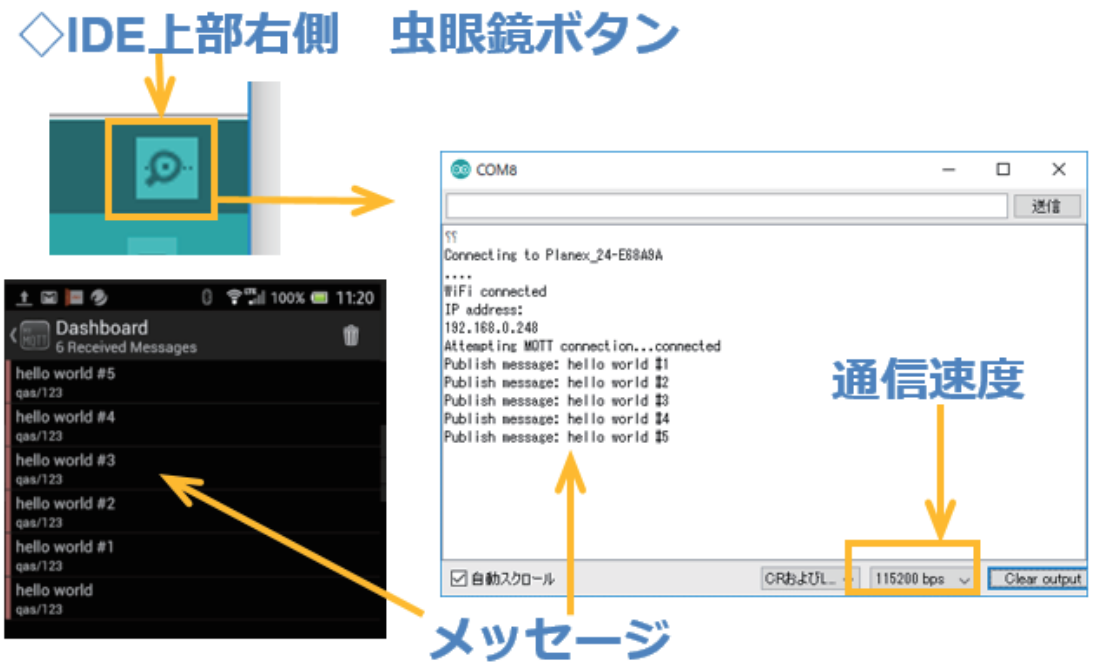

図　60

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。

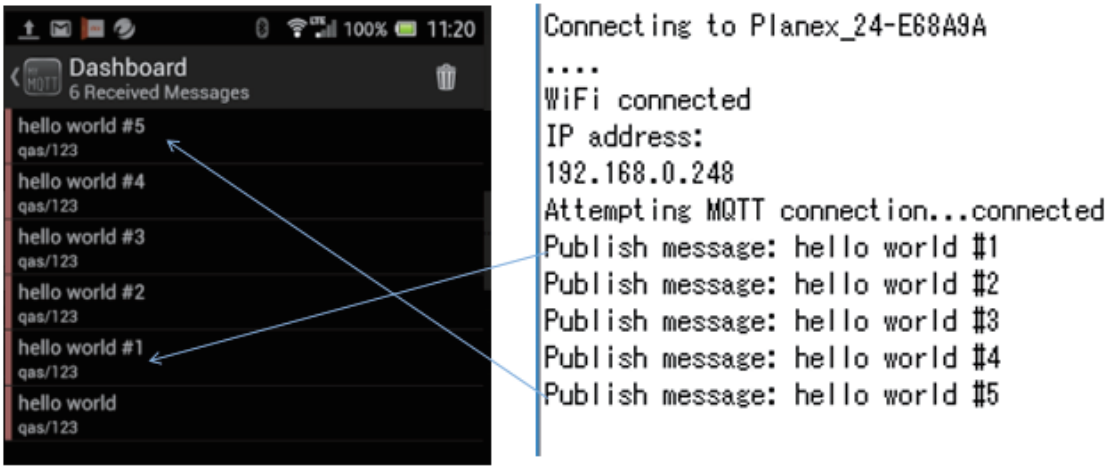

図　61

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を開発します。

第11回　WEB 連携②(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。下図をご覧ください。

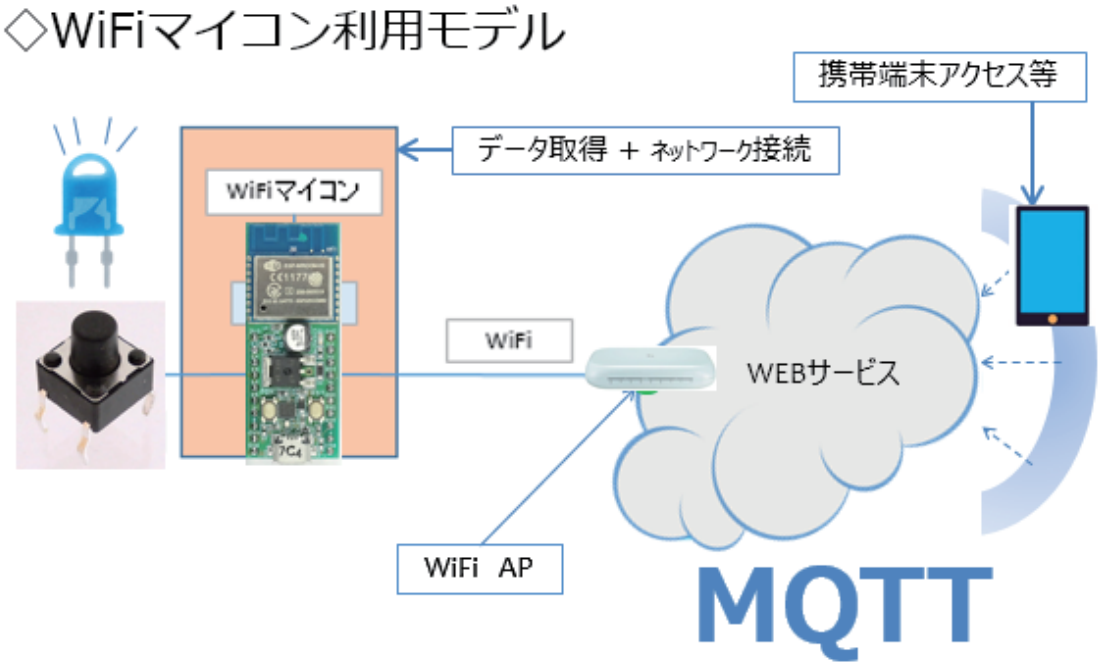

図　62

WiFi マイコンの外部デバイスとして SW と LED を使用します。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。工場などで設備の稼働・停止を把握する稼動状況収集システムや、製品が 1 個 1 個出来上がっている状況を把握する生産実績収集システム等に応用できます。

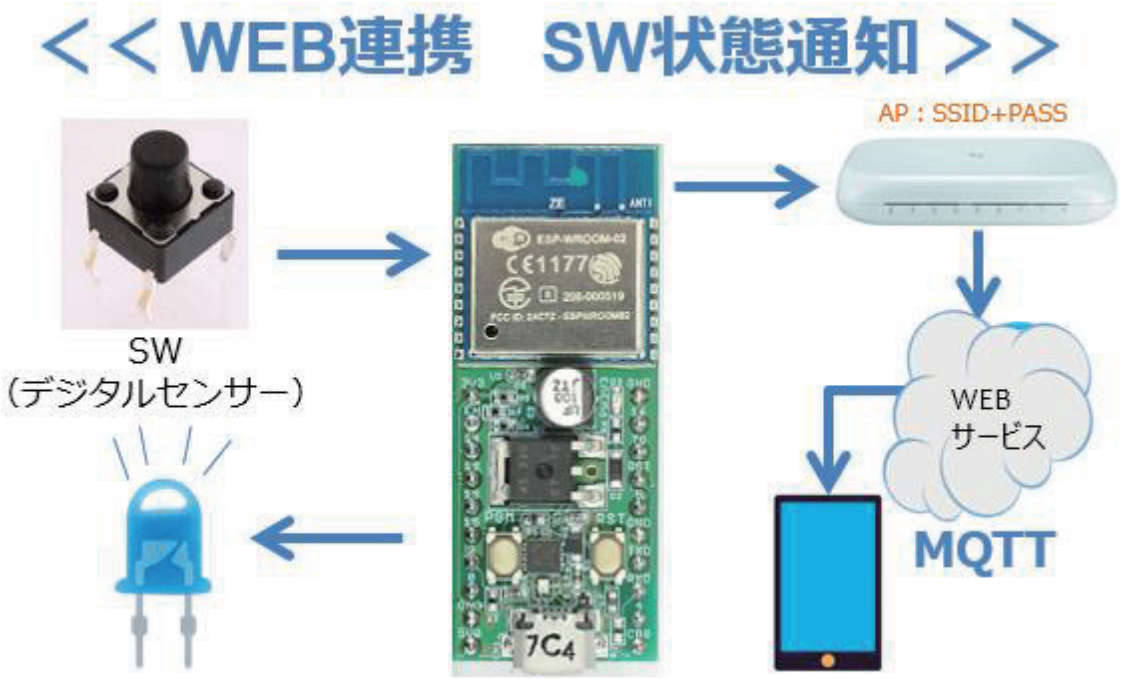

図　63

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】だと考えれば分かり易いでしょう。LED で SW の ON/OFF 状態を表示します。今回は【SW の ON/OFF 状態が変化した】ことを検出してメッセージを発行することにしましょう。

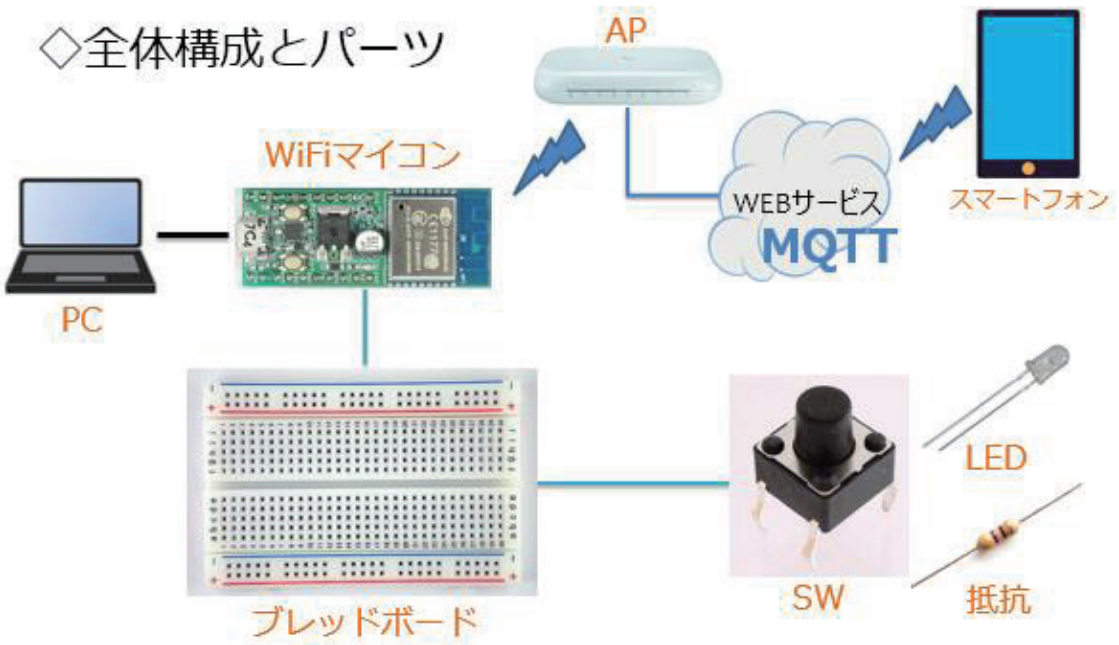

図 64

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
9. スマートフォン×1 台（アプリインストール）

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

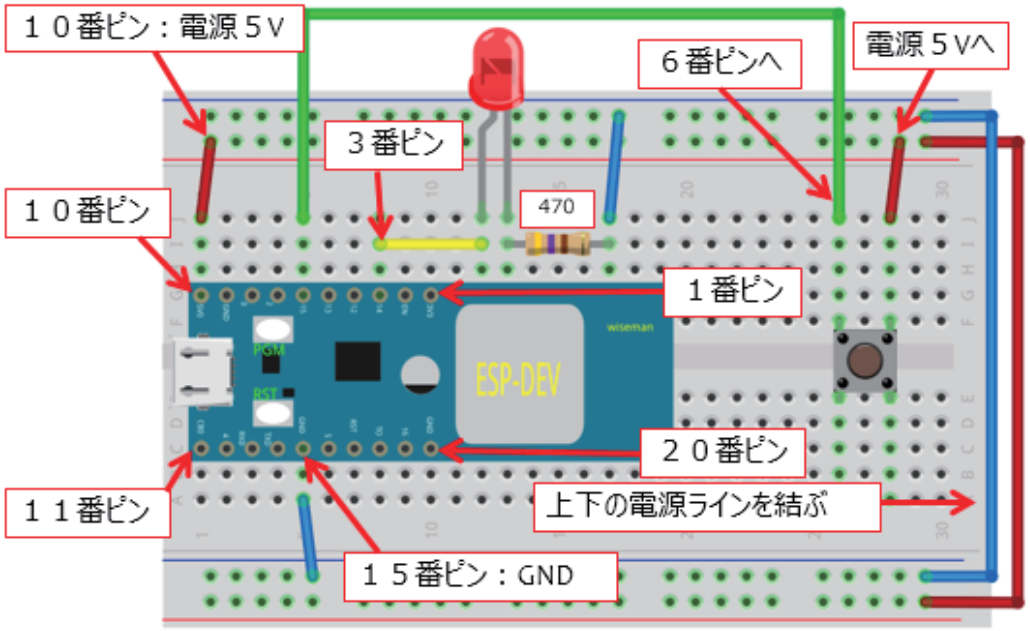

図　65

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

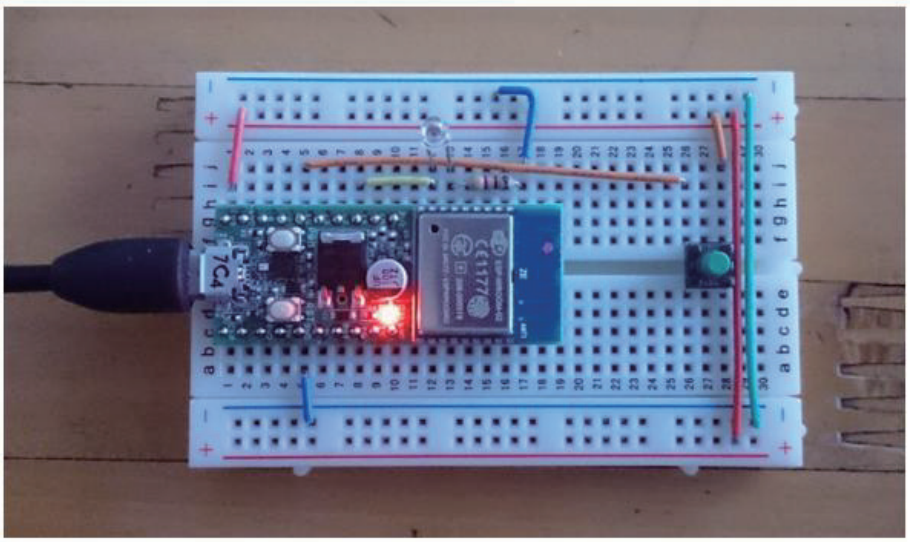

図　66

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

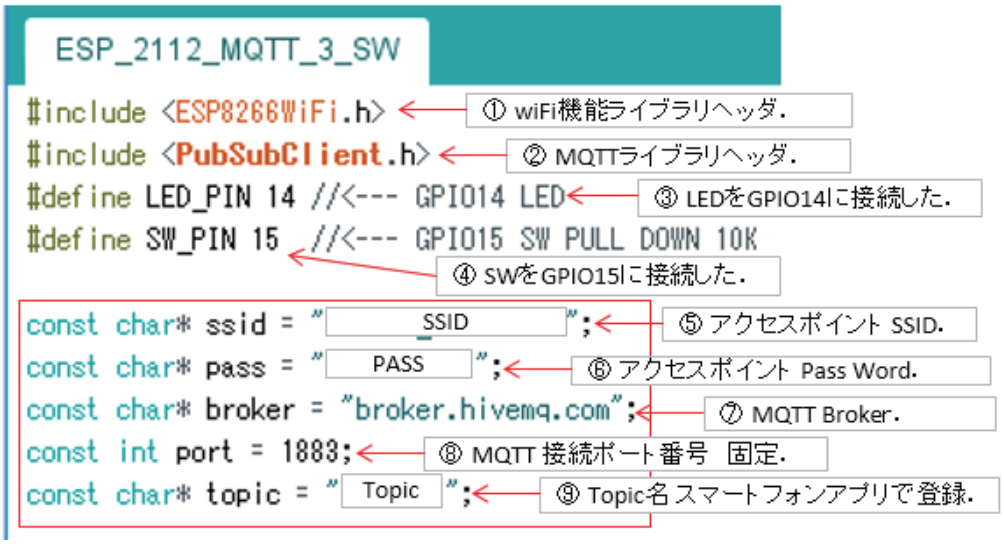

図 67

プログラムを書く

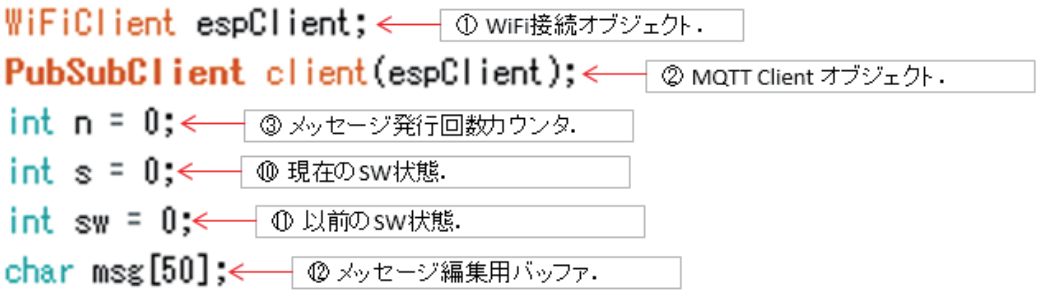

図 68

プログラムを書く　setup()

```
void setup() {
  pinMode(LED_PIN,OUTPUT);
  pinMode(SW_PIN,INPUT);
  Serial.begin(115200);
  init_wifi();
  client.setServer(broker, port);
}
```

① GPIO14を出力に設定.
② GPIO15を入力に設定.
③ シリアルポート初期化.
④ WiFi AP 接続.
⑤ MQTT Broker接続準備.

図　69

プログラムを書く　loop()

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  s = digitalRead(SW_PIN);
  digitalWrite(LED_PIN, s);
  if(s != sw) {
    ++n;
    switch(s) {
      case 1:  // OFF--->ON
        snprintf(msg, sizeof(msg), "changed!! OFF--->ON #%d", n);
        break;
      case 0:  // ON--->OFF
        snprintf(msg, sizeof(msg), "changed!! ON--->OFF #%d", n);
        break;
    }
    sw = s;
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish(topic, msg);
  }
}
```

① MQTT 接続確認.
② MQTT 接続.
③ MQTT 接続情報更新.
④ 現在のSW状態読込.
⑤ SW状態をLEDに反映.
⑥ SW状態変化したか？.
⑥ メッセージカウンタ更新.
⑦ SW ONした.
⑧ メッセージ編集.
⑨ SW OFFした.
⑩ メッセージ編集.
⑪ SW状態更新.
⑫ PCに通知.
⑬ PCにメッセージ通知.
⑭ メッセージ発行.

図　70

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　71

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　72

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

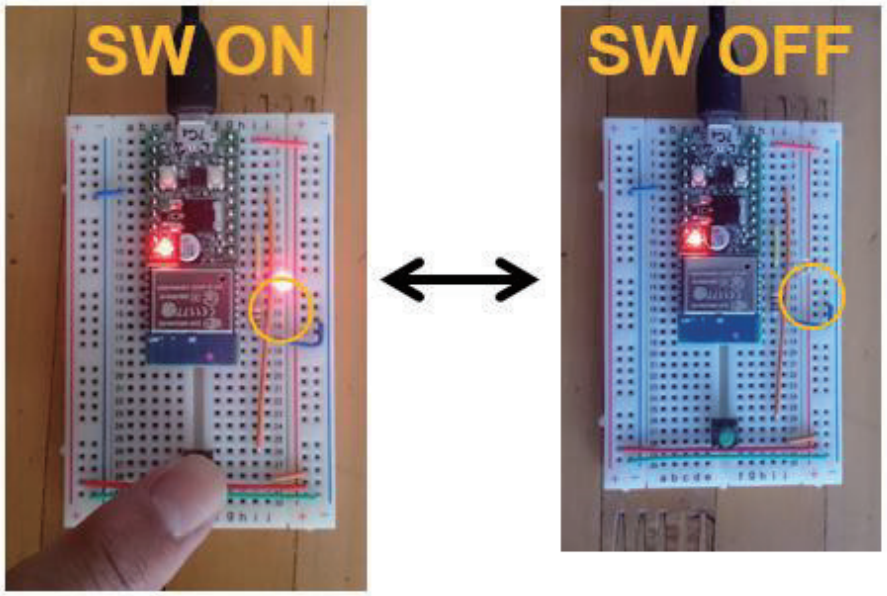

図 73

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　74

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。

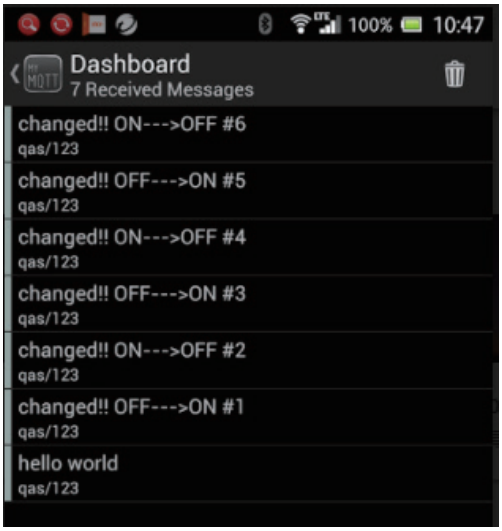

図　75

第12回　WEB連携③(MQTT)

まず下図を見てください。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。

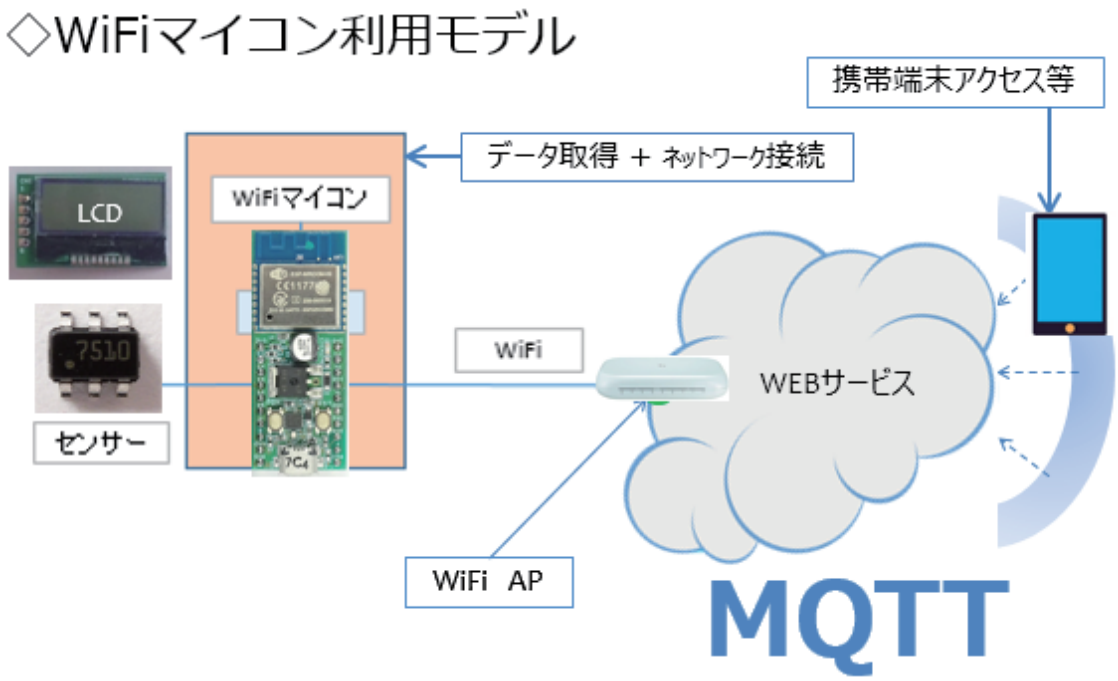

図　76

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が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　77

図を見て分かるように、具体的な機器類の関係でも、WiFi マイコンの左側に【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。

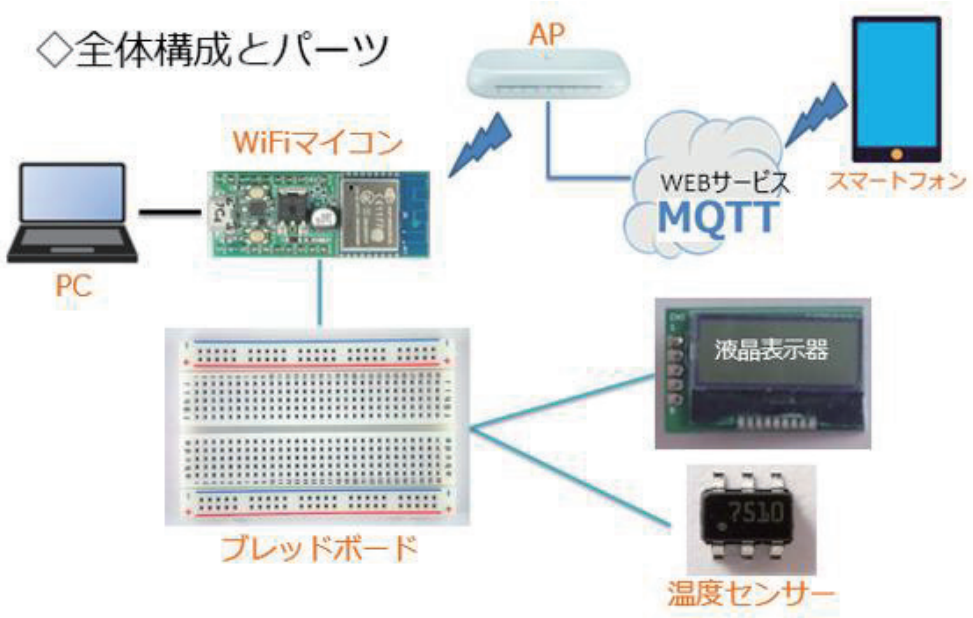

図 78

全体の構成は上図の通りです。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。

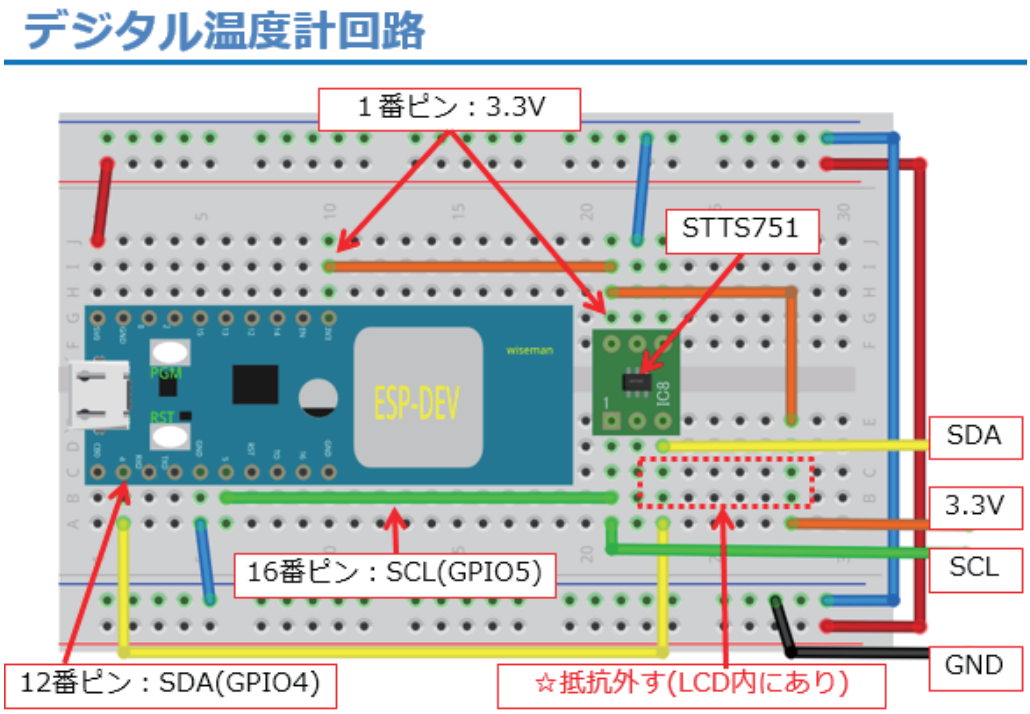

図　79

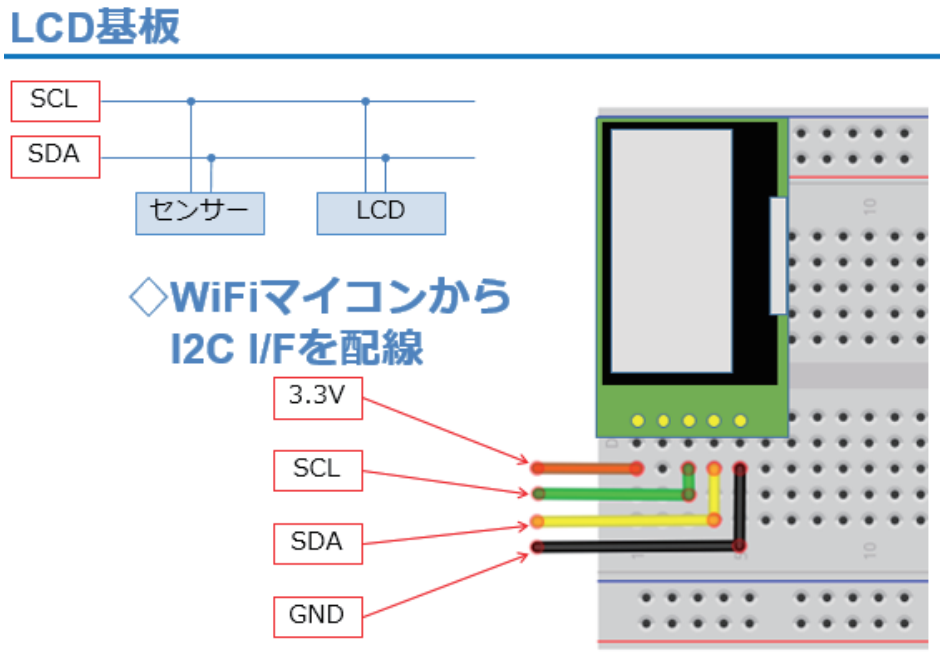

図　80

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。新たに配線される方は、**USB ケーブルを必ず外して配線を行って下さい。**

実際に配線した様子

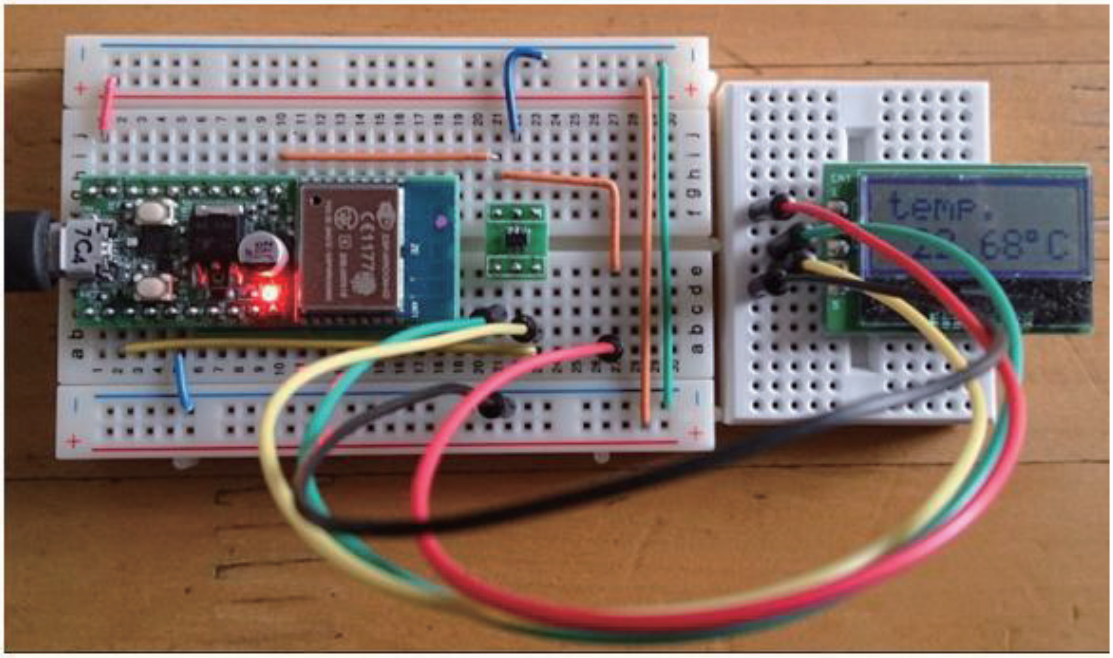

図 81

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　82

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　83

プログラムを書く　loop()

```
void loop() {
  long now = millis();          ← ① マイコン現在時 取得.
  if (!client.connected()) {    ← ② MQTT 接続確認.
    reconnect();                ← ③ MQTT 接続.
  }
  client.loop();                ← ④ MQTT 接続情報更新.
  if ((now - lastMsg) % 1000 == 0) {   ← ⑤ 1秒経過確認.
    read_disp();                ← ⑥ 温度読込＋LCD表示.
  }
  if (now - lastMsg > 10000) {  ← ⑦ 10秒経過確認.
    lastMsg = now;              ← ⑧ メッセージ発行時刻更新.
    ++value;                    ← ⑨ メッセージ発行回数更新.
    snprintf(msg, 30, "hello world #%ld", value);   ← ⑩ メッセージ編集.
    Serial.print("Publish message: ");   ← ⑪ PCにメッセージ発行通知.
    Serial.println(msg);        ← ⑫ PCにメッセージ内容通知.
    l_moji[6]='¥0';             ← ⑬ LCD下段にNULL追加.
    client.publish(topic, l_moji);   ← ⑭ メッセージ発行.
  }
}
```

図　84

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　85

※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 初メッセージ発行.
④ PCに状況通知.

図　86

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;                  ┐
    l_moji[5]=(char)('0' + temp/100);  // 0.01の位  ├ ④ 小数部温度 LCDバッファ格納.
    l_moji[6]=0xDF;               // °C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　87

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　88

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　89

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　90

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　91

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。

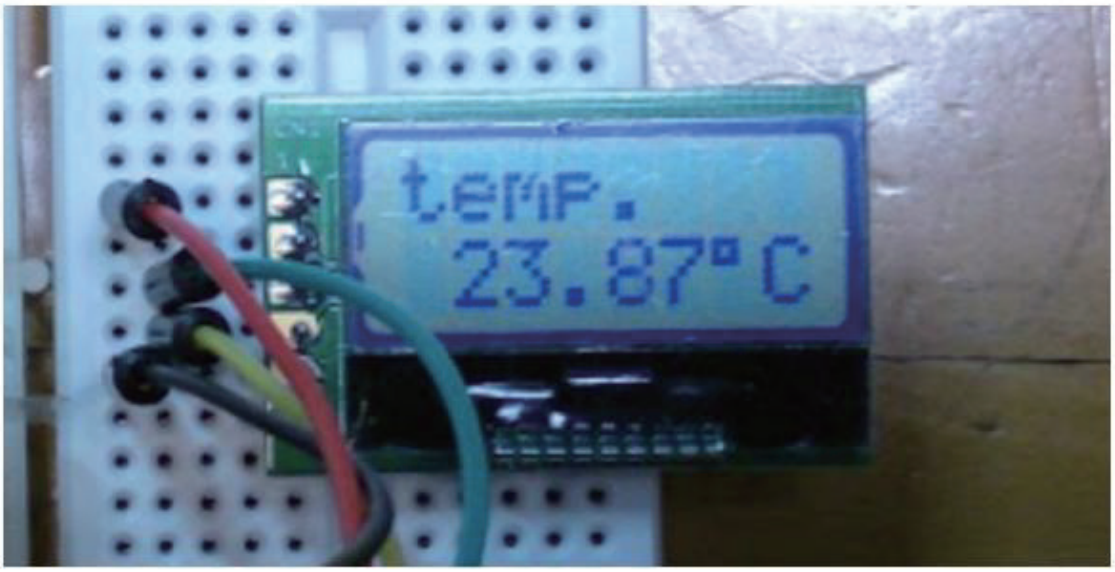

図 92

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。

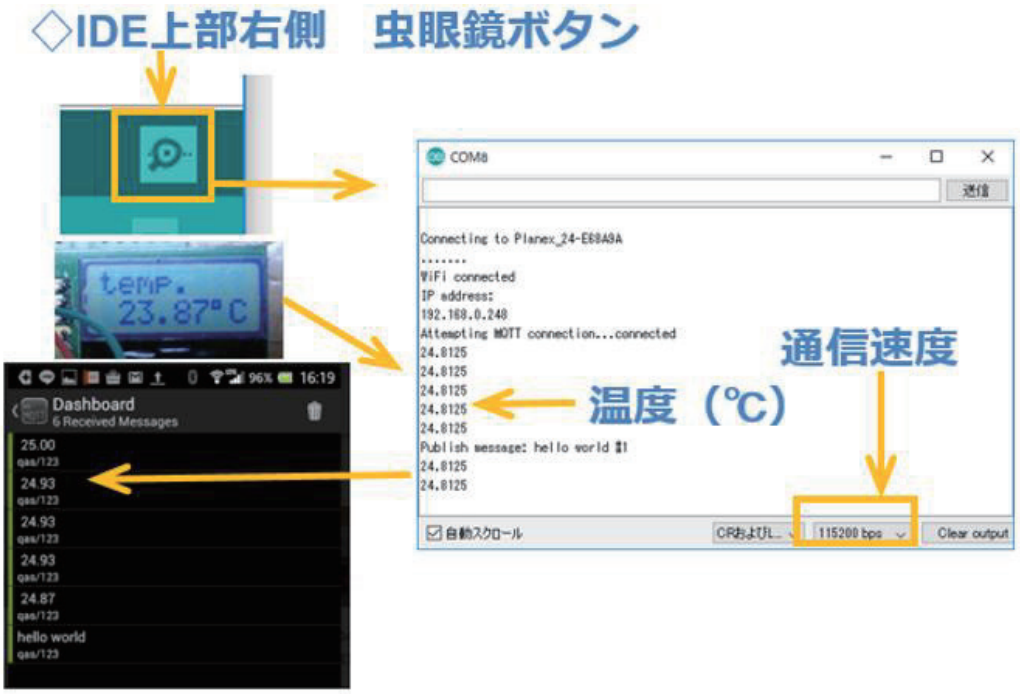

図　93

いかがでしょうか。約 1 秒ごとに LCD の温度表示が更新されて、約 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、外出先等の遠隔地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。ぜひ考えてみて下さい。

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 12 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）。

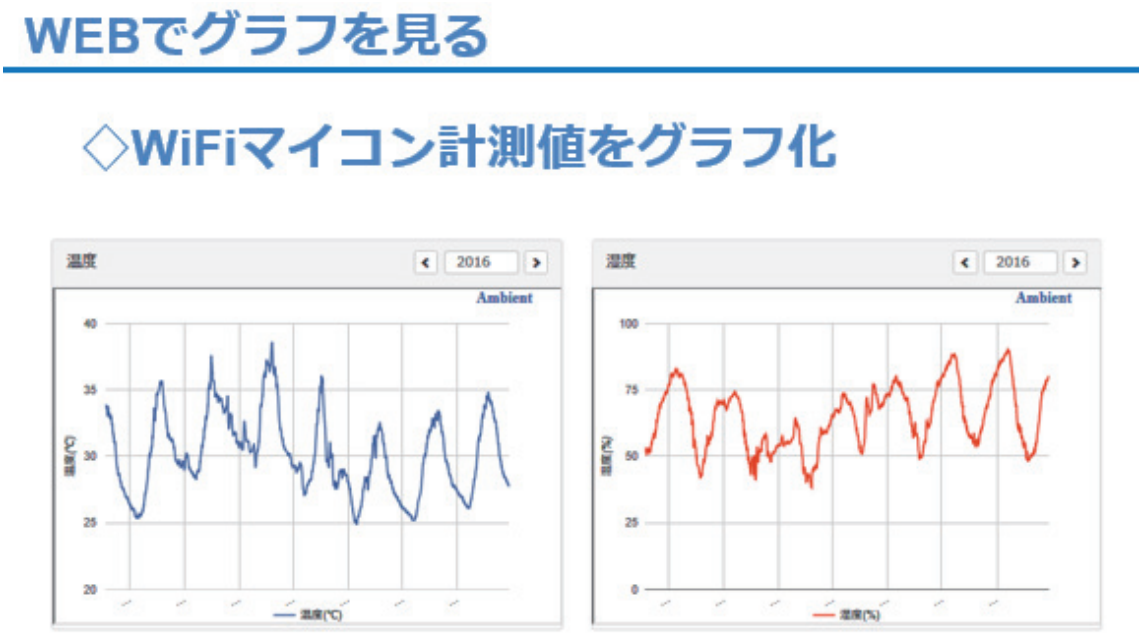

図　94

今回使用するのは、Ambient という WEB サービスです。上図は、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。

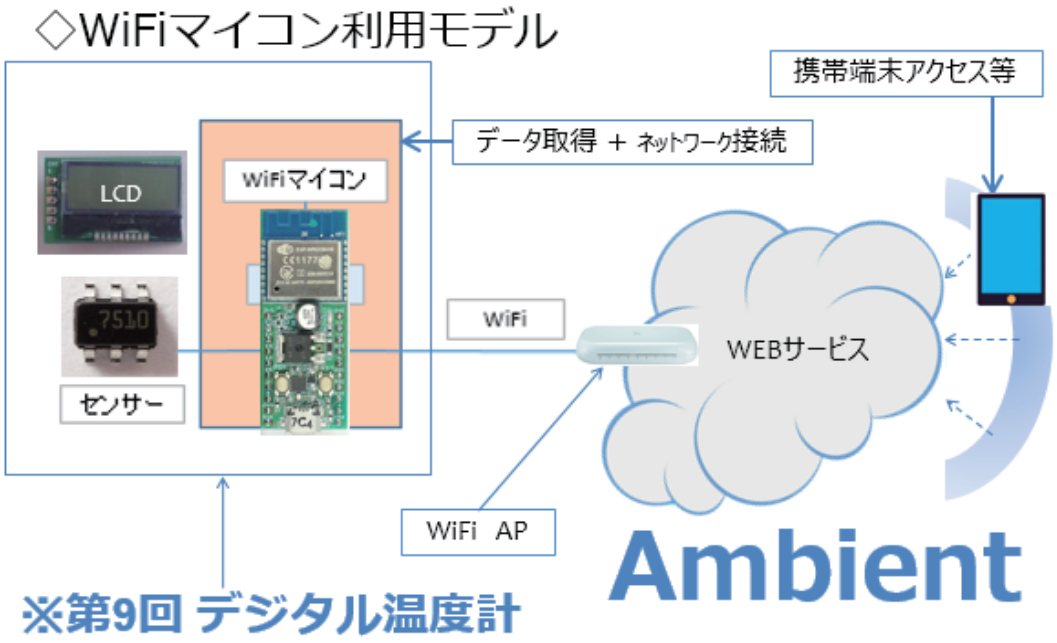

図 95

今回のモデルは、第 12 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。

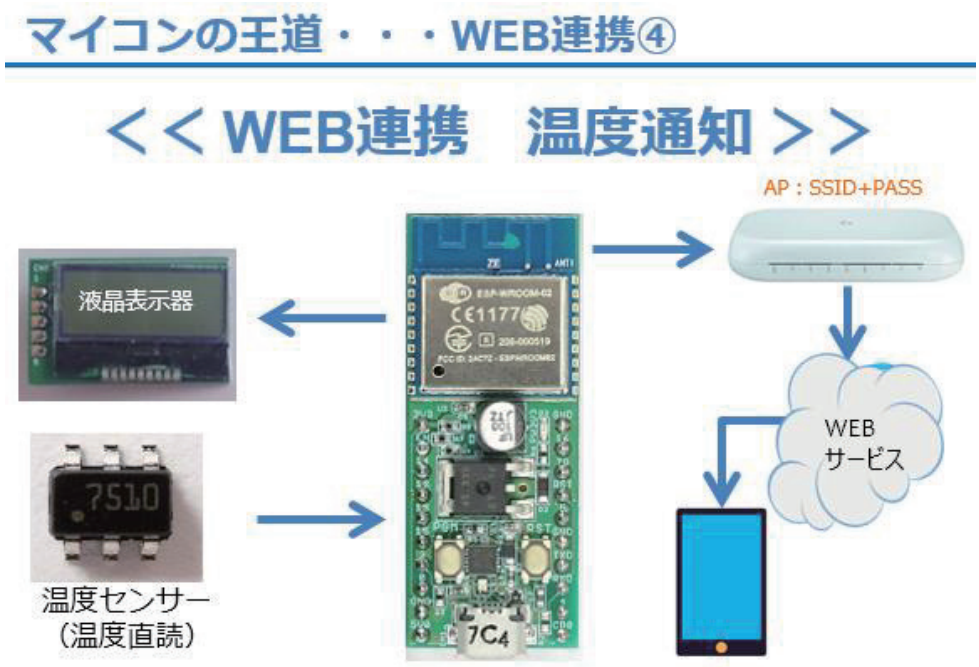

図 96

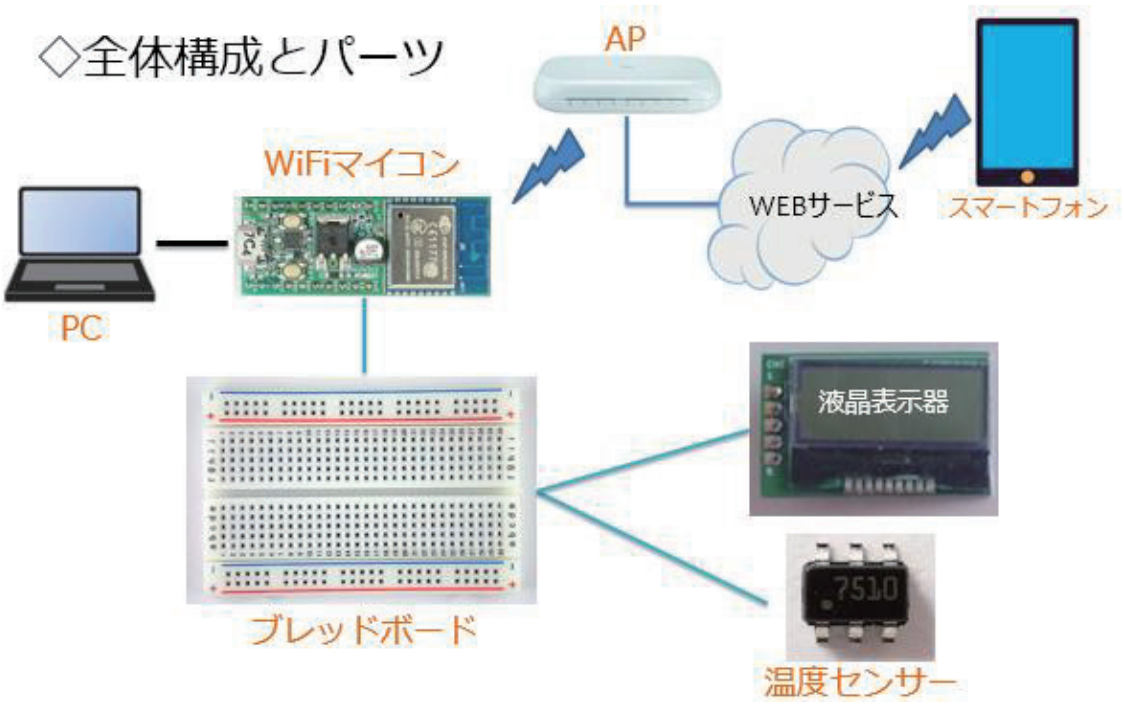

図 97

全体の構成は上図の通りです。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）
9. メールアカウント（Ambient：WEB サービス登録用）

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 98

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。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。

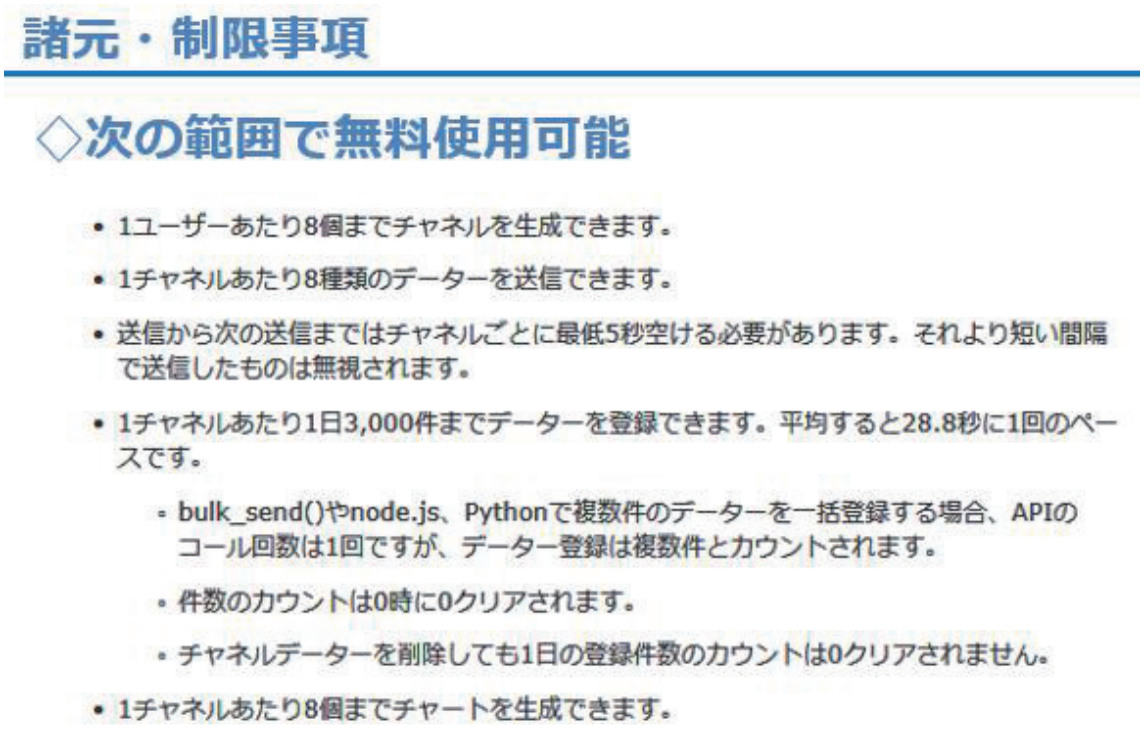

図 99

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間に合います。

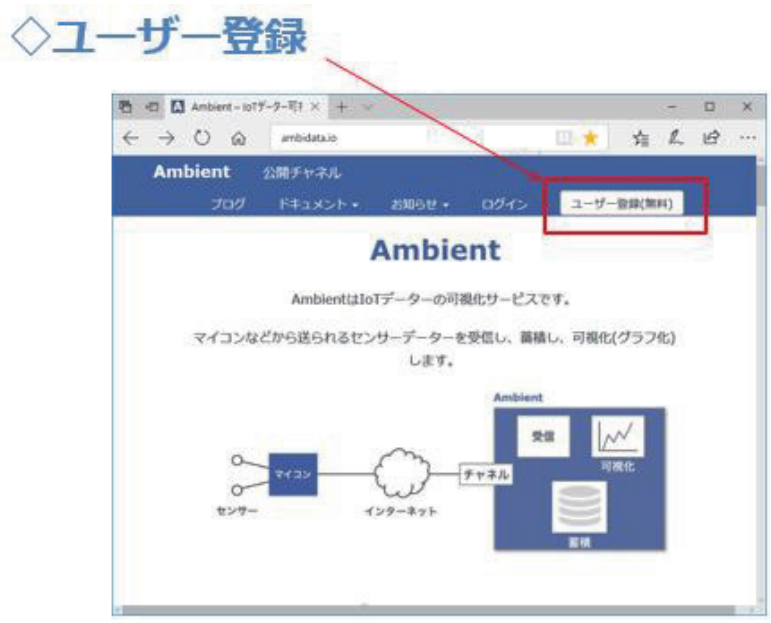

図　100

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。

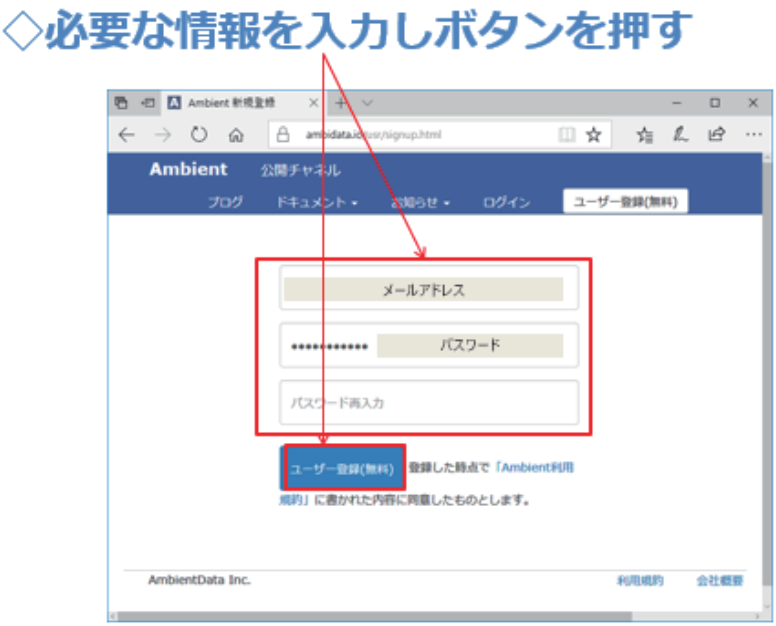

図　101

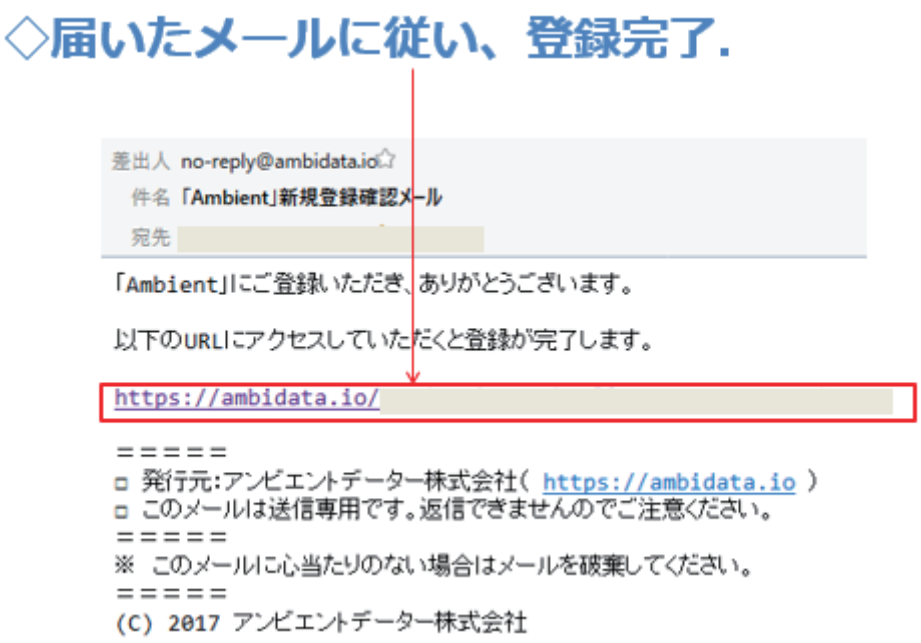

図 102

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 103

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（埋込）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。

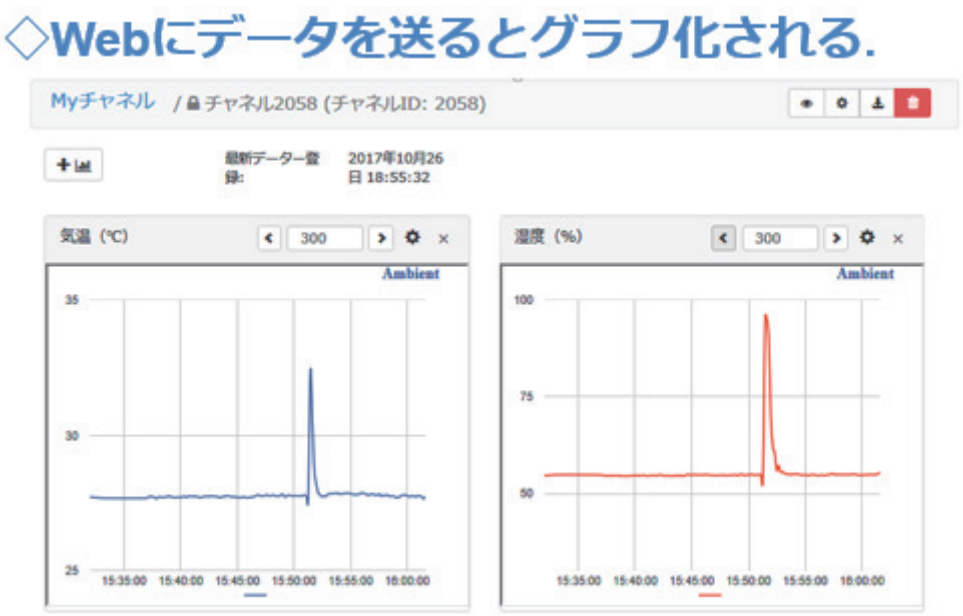

図 104

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。

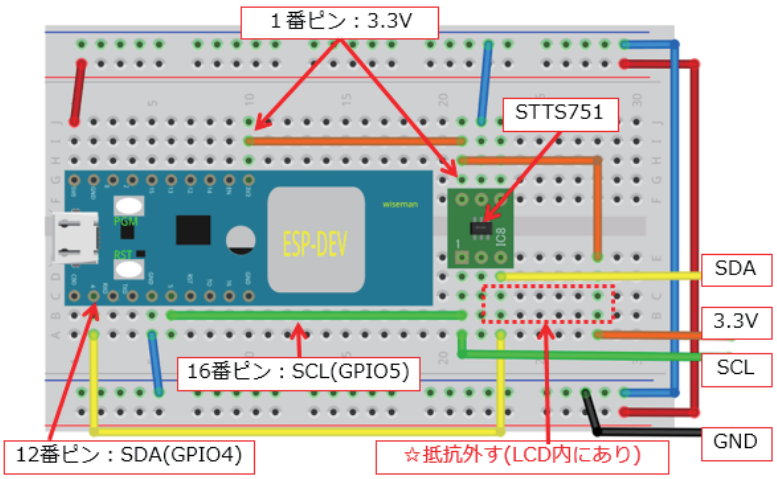

図　105

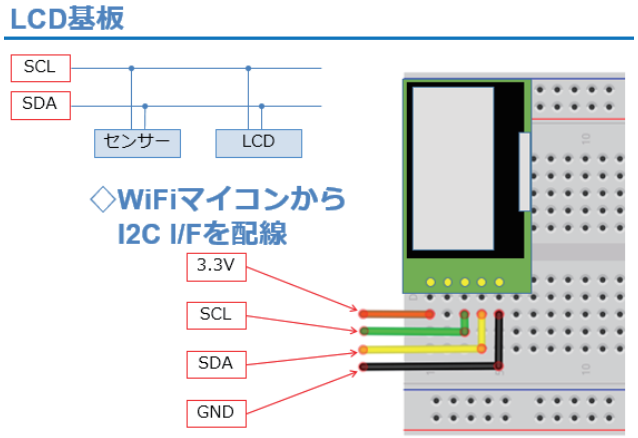

図　106

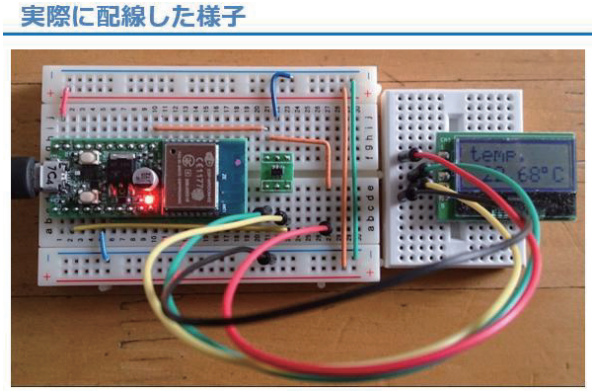

図　107

Ambientライブラリの準備

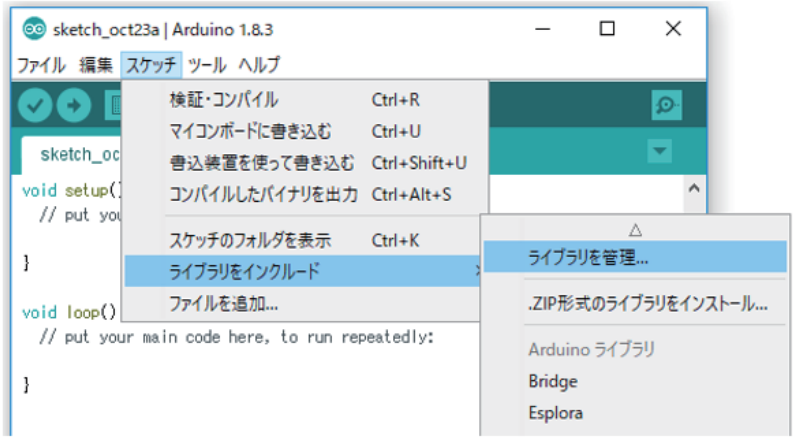

図　108

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。

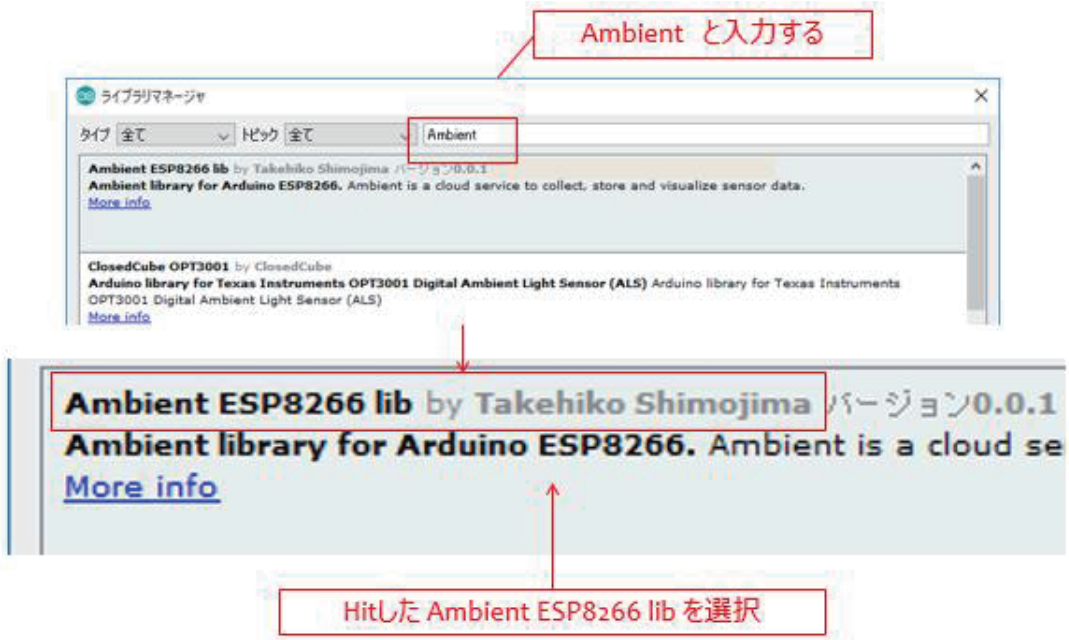

図　109

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　110

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。

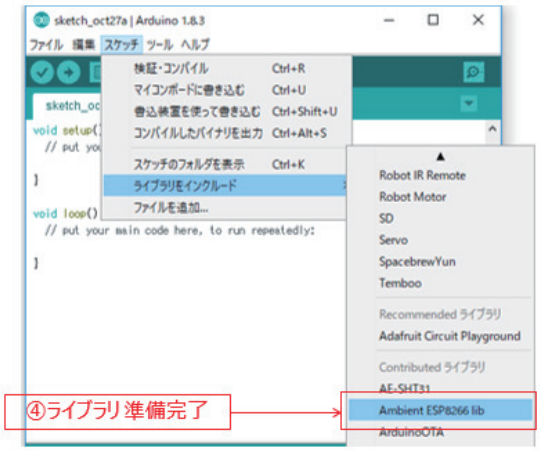

図　111

上図に従い、【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。

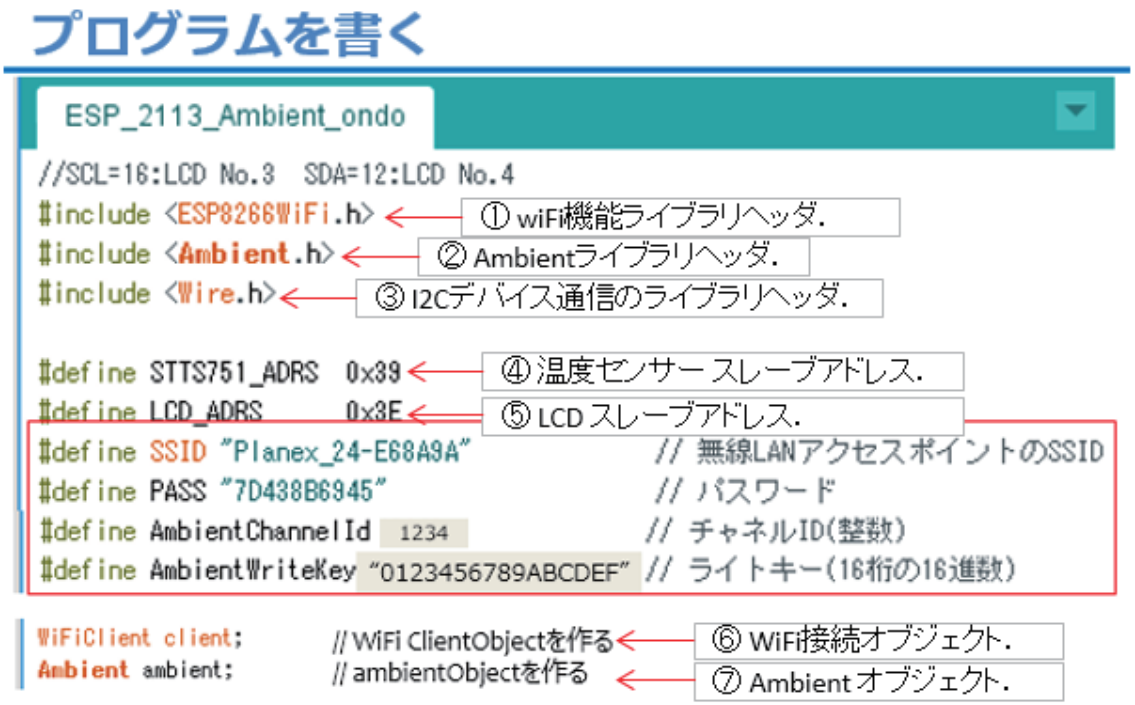

図 112

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　113

①Ambient を初期化・開始します.

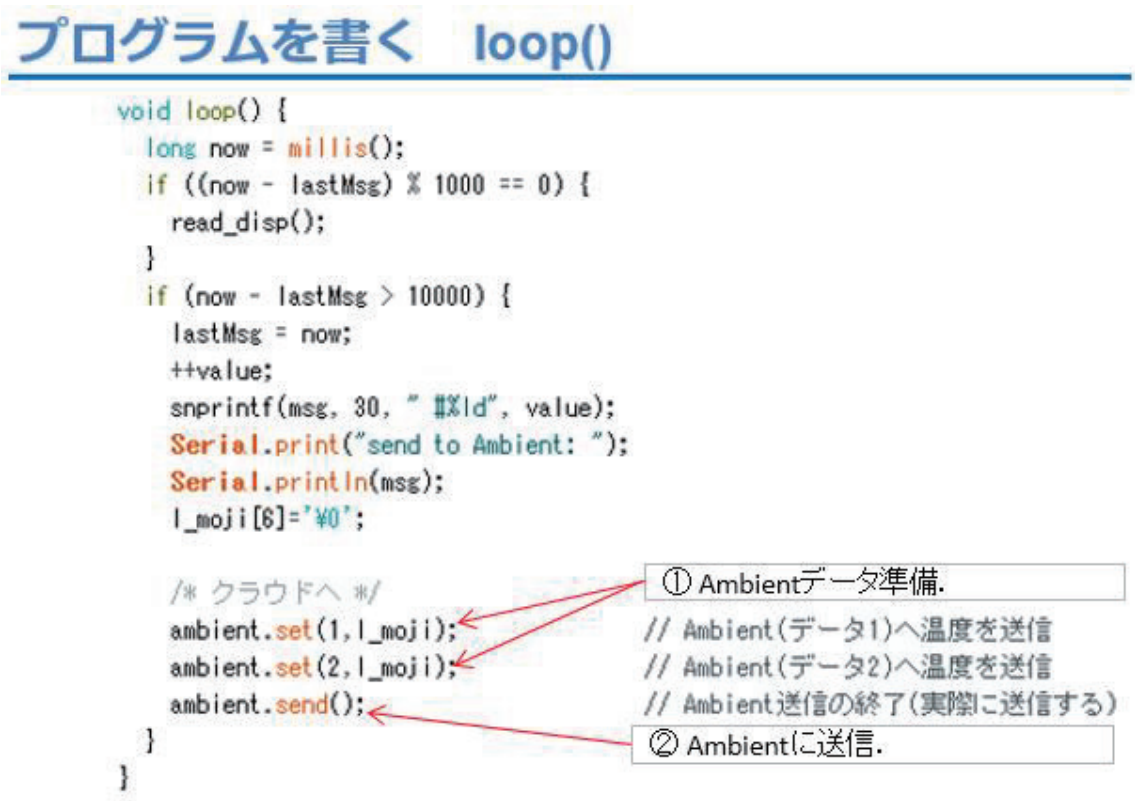

図　114

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 つ準備しているのは、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.

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　115

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　116

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　117

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
                                            ← ③ 温度センサー初期設定.
    Wire.endTransmission();
}                                           ← ④ I2C通信終了.
```

図　118

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　119

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　120

LCD 初期化もこれまでに作成したものです。

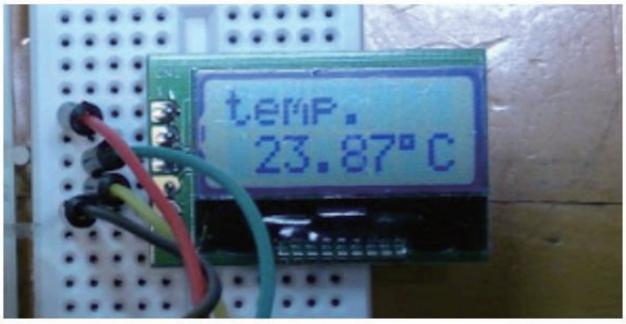

図　121

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。

動作確認

図　122

図 123

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。

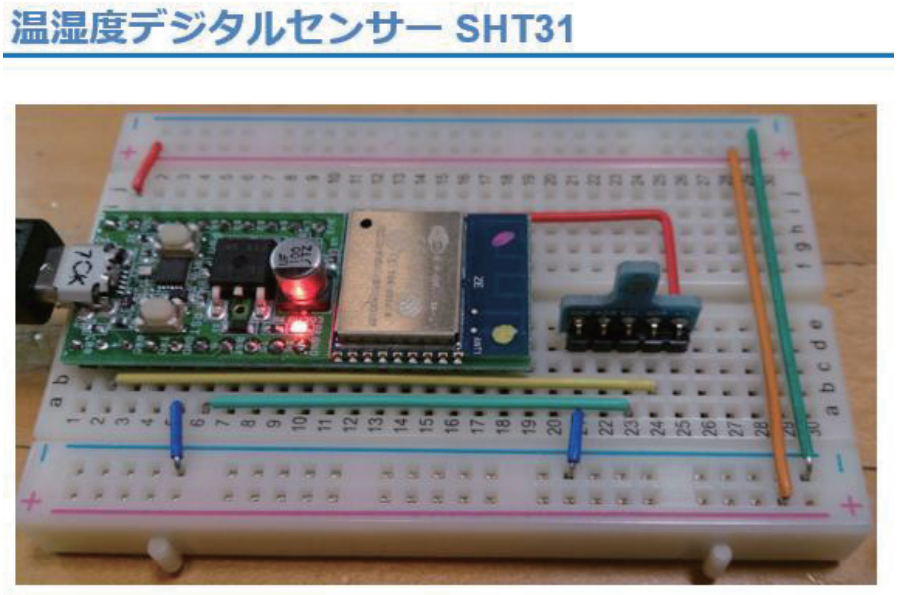

図　124

図　125

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　126

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。

つながるＩｏＴモデル

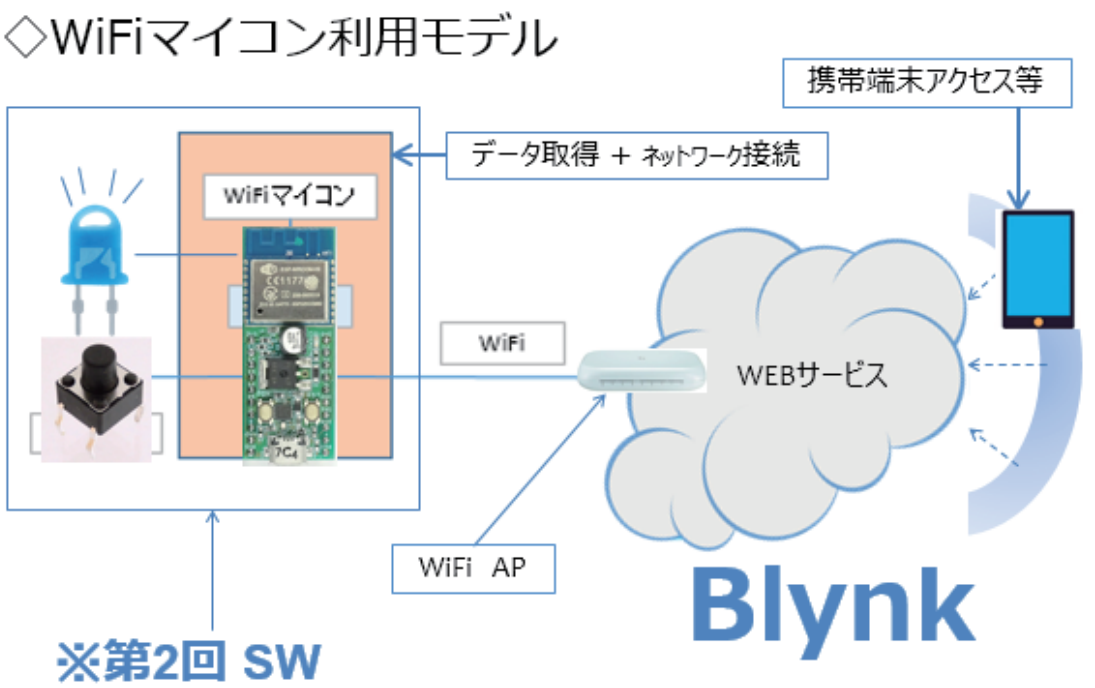

図　127

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。

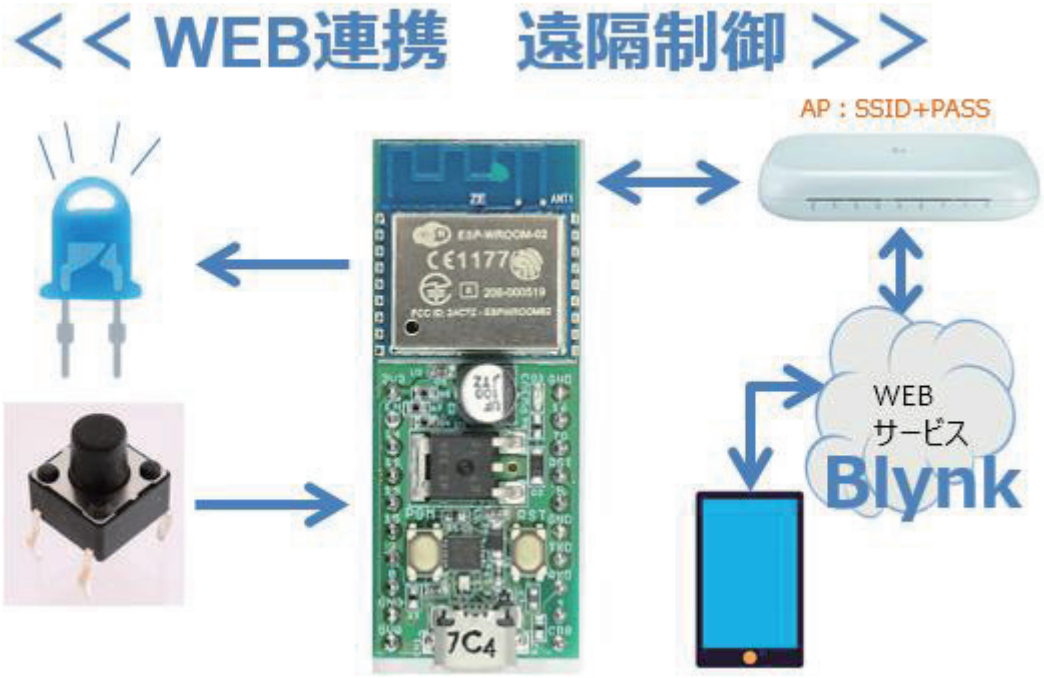

図　128

上図は第 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。

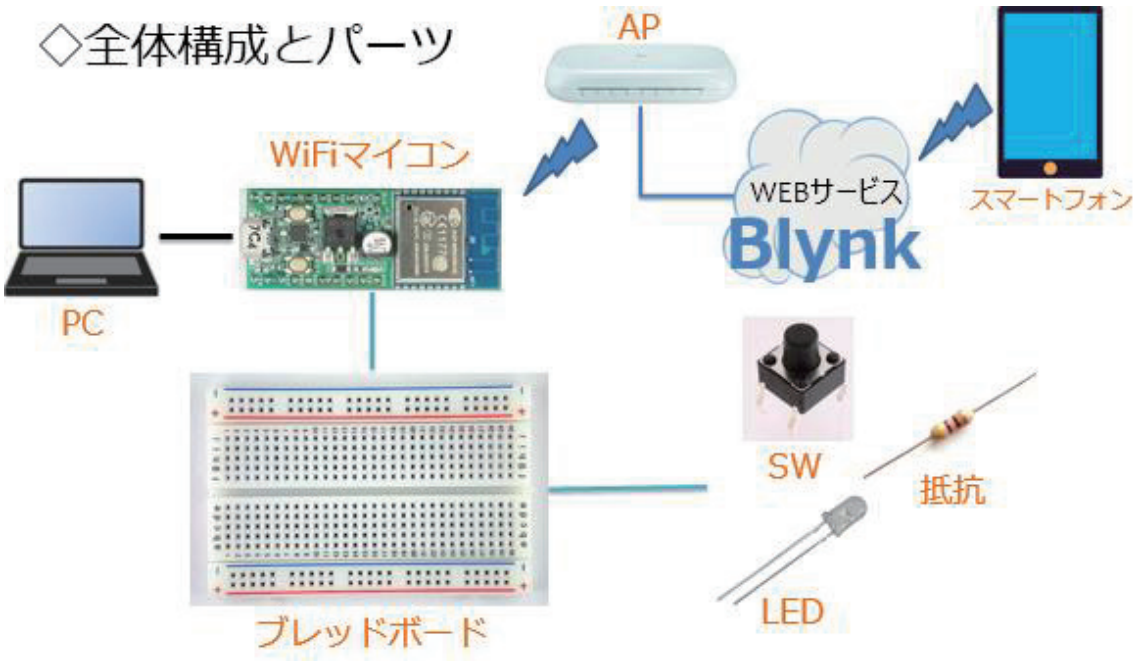

図 129

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
9. スマートフォン×1 台（アプリインストール）
10. メールアカウント（Blynk：WEB サービス登録用）

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 11 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

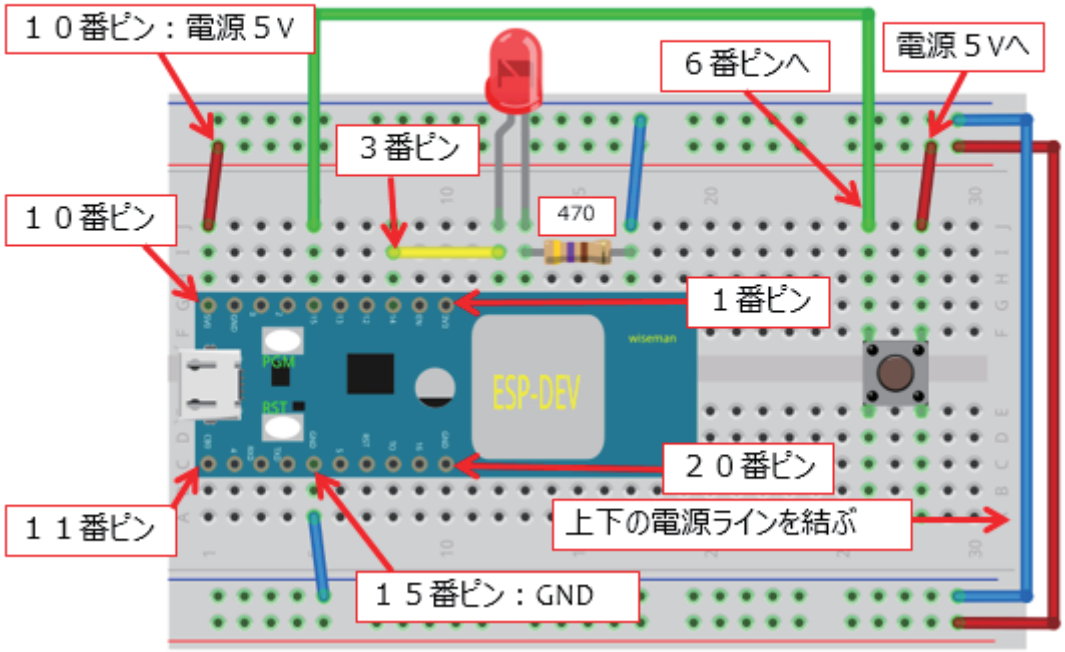

図　130

実際に配線した様子

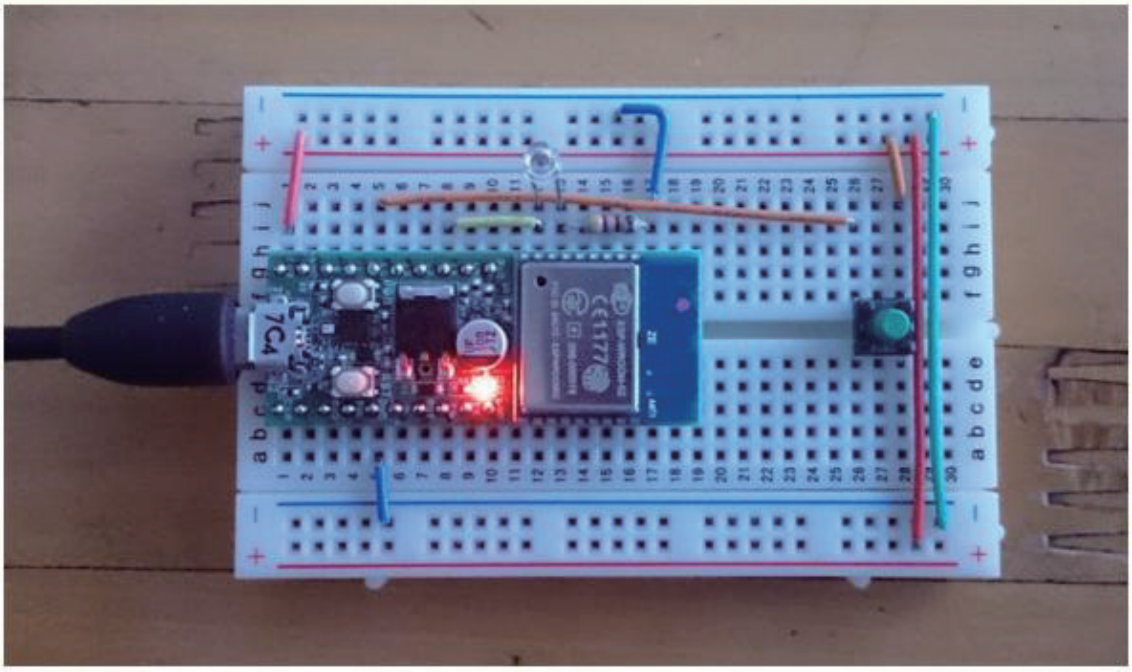

図　131

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。）がそろっています。

図 132

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

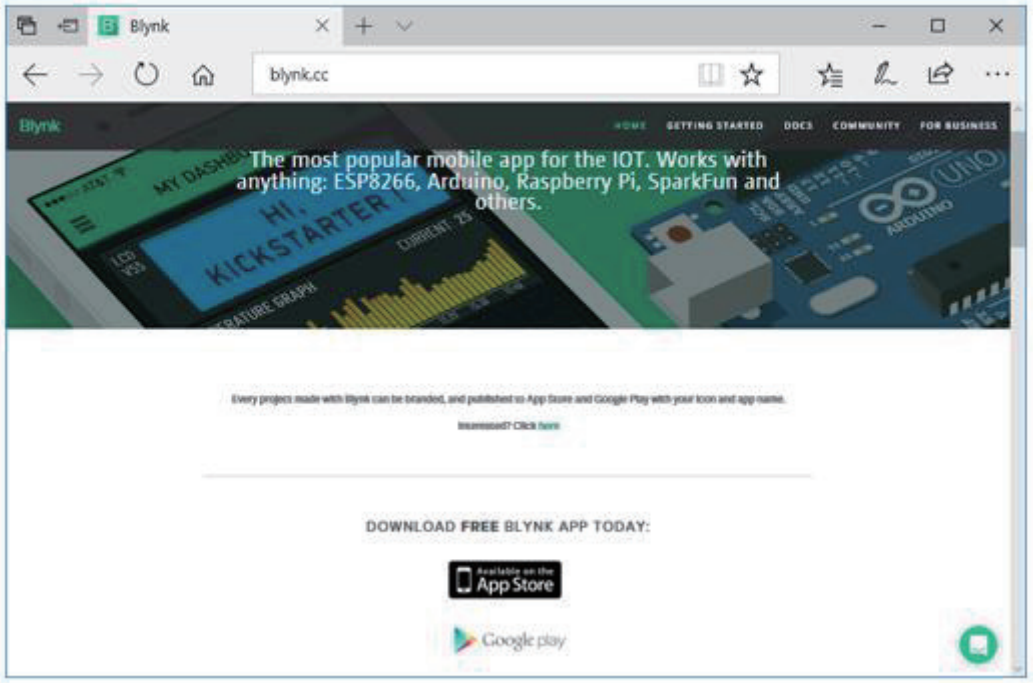

図　133

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　134

使用するスマートフォンで Blynk を検索すると、上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）

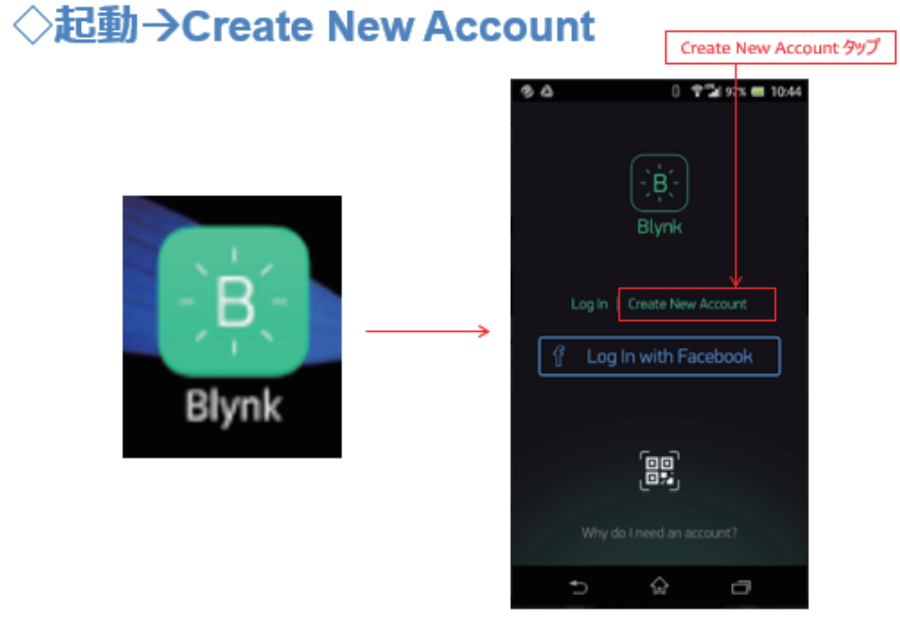

図　135

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

図　136

【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、下図左の画面【Project Settings】に切り替わります。

図 137

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。画面は下図左の【My Devices】に変わります。

図 138

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)。

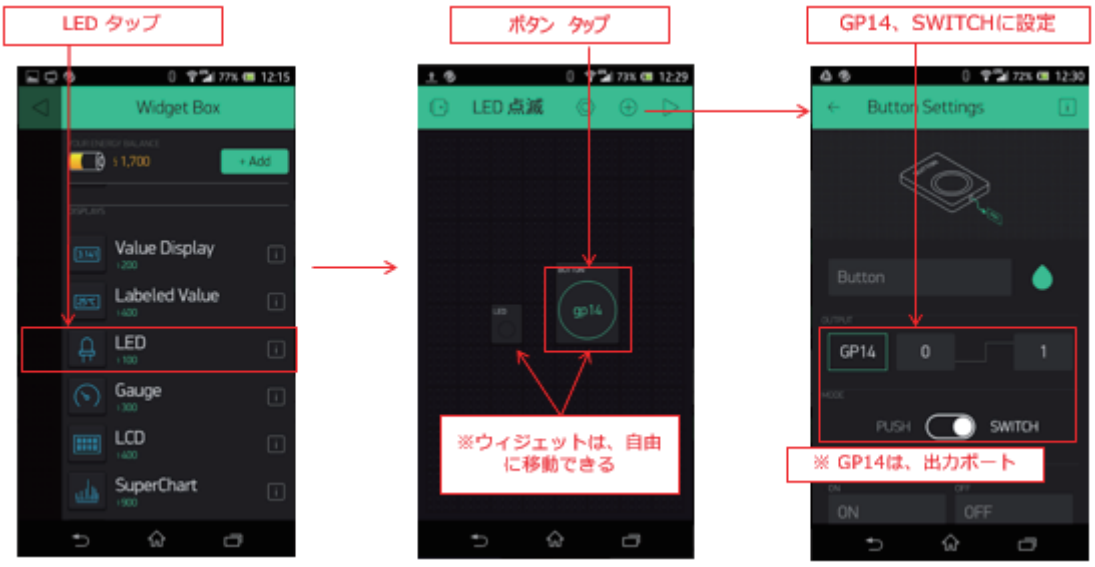

図 139

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）。

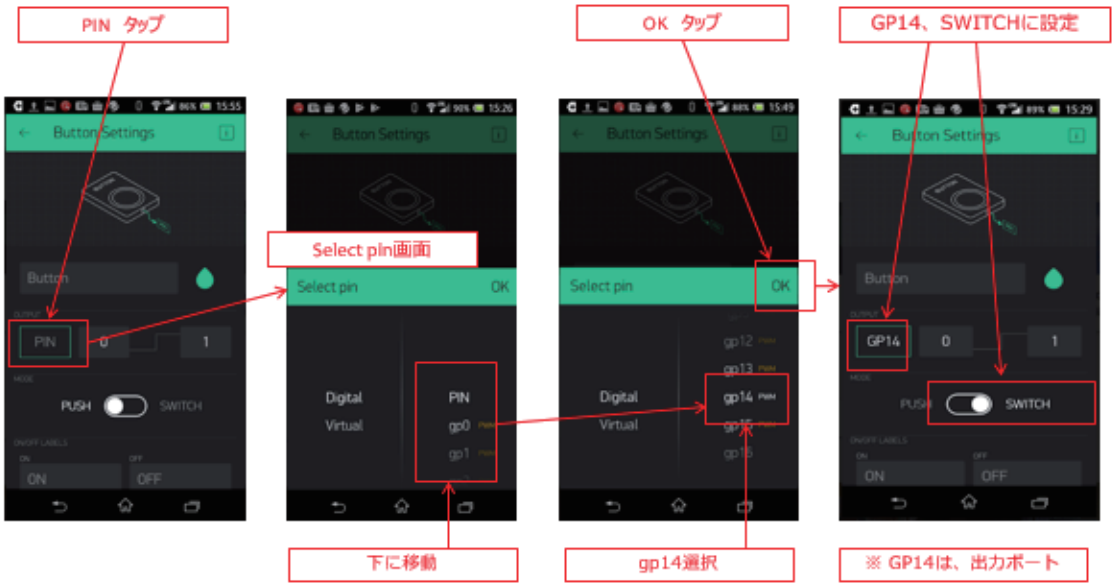

図 140

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。

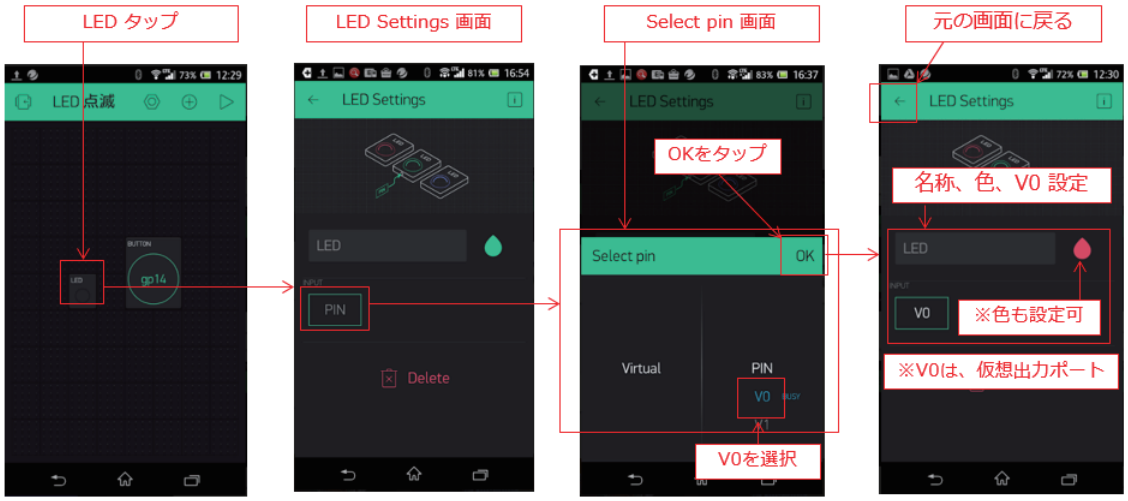

図 141

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。

◇スマートフォンアプリを起動・停止

図　142

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

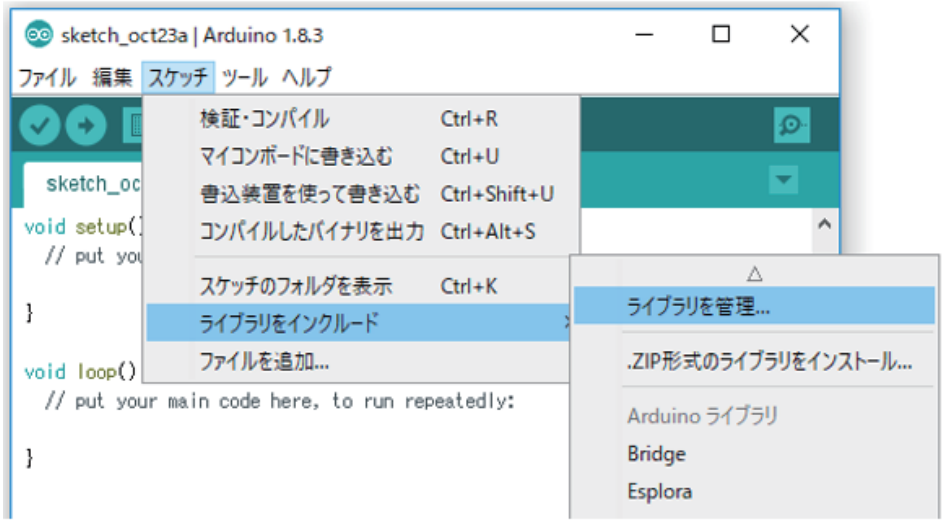

図　143

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。

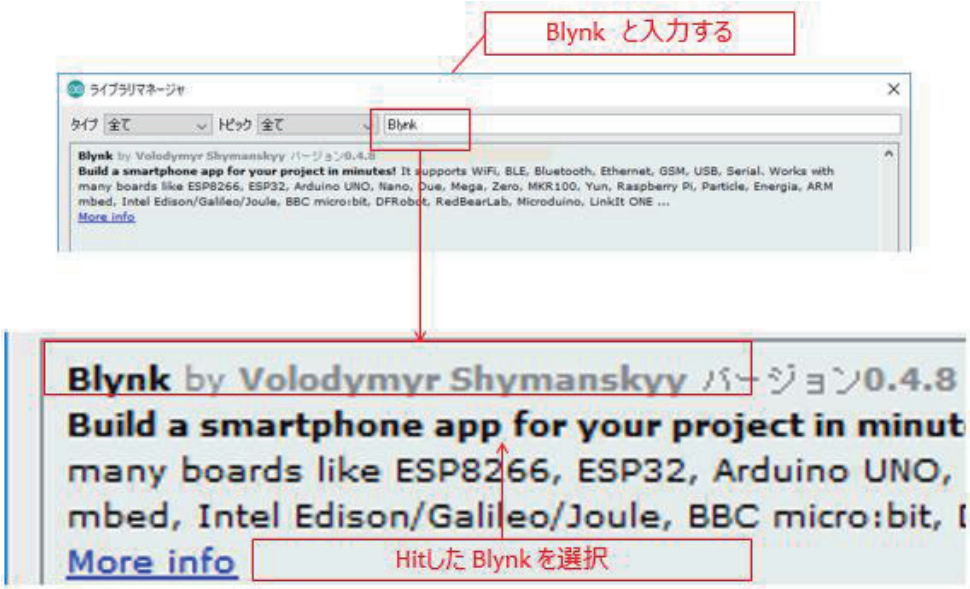

図　144

ライブラリマネージャの上部に【Blynk】と入力します。

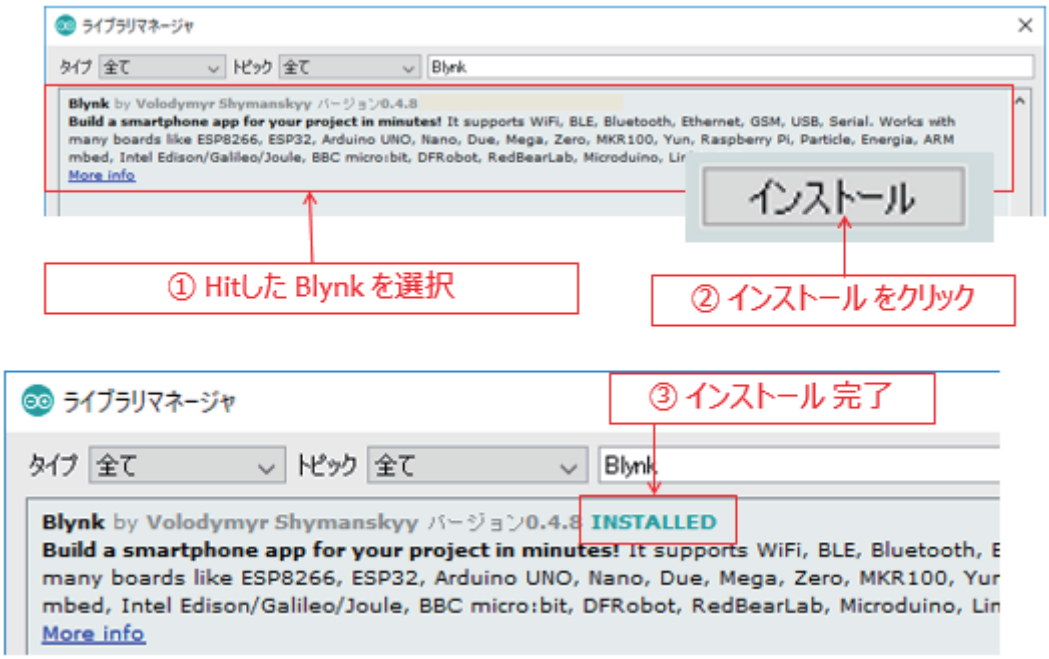

図　145

Blynk ライブラリを選択してインストールします。

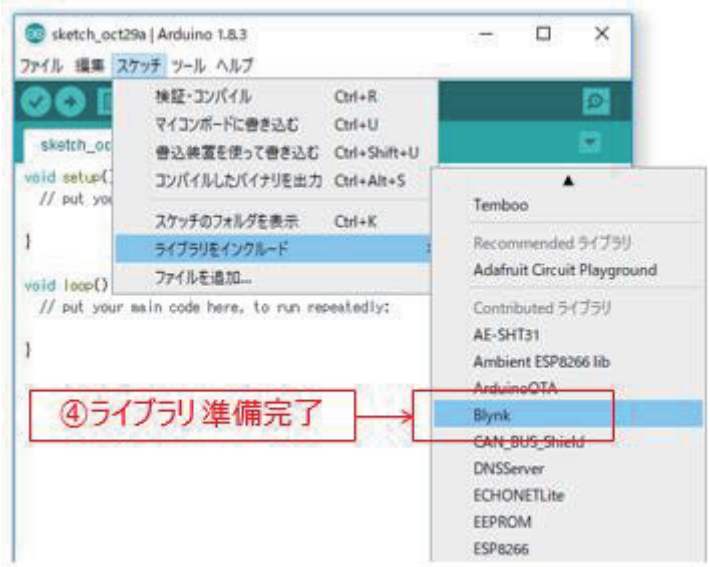

図　146

【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！！

プログラムを書く

ESP_2114_Blynk

```
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

// スマートフォンで取得したAUTH TOKEN
char auth[] = "01234567890123456789012345678901";
int in_data = 0;
```

図 147

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。

プログラムを書く　setup()

```
void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, "Planex_24-E68A9A", "7D438B6945");
  Serial.println("Start Blynk!!");
}
```

SSID　PASS

図 148

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　149

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。

動作確認

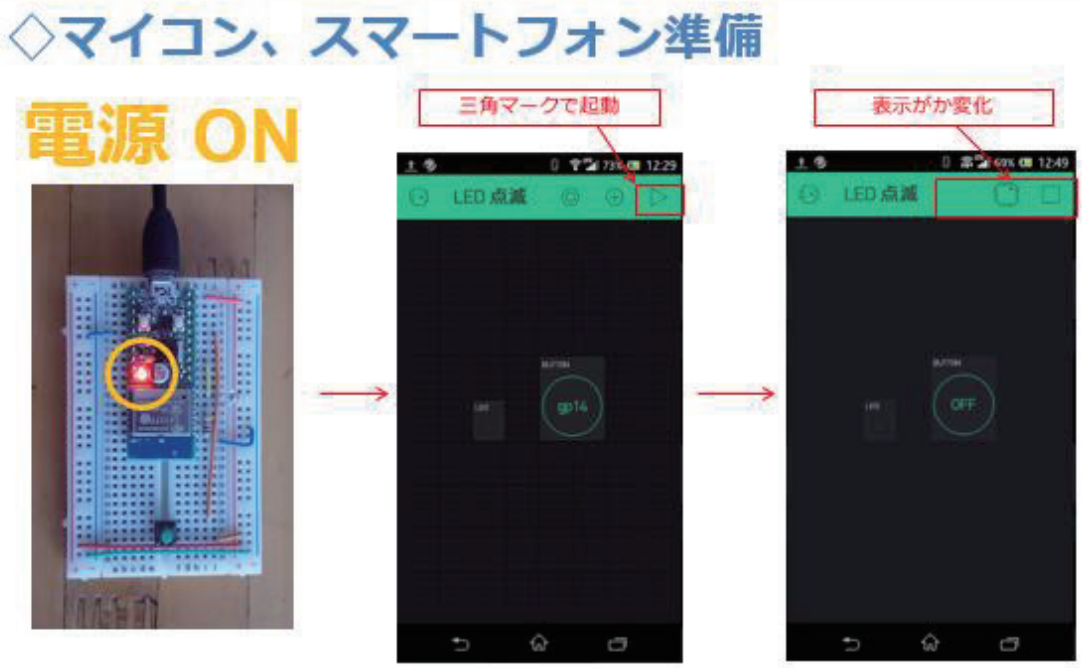

図　150

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

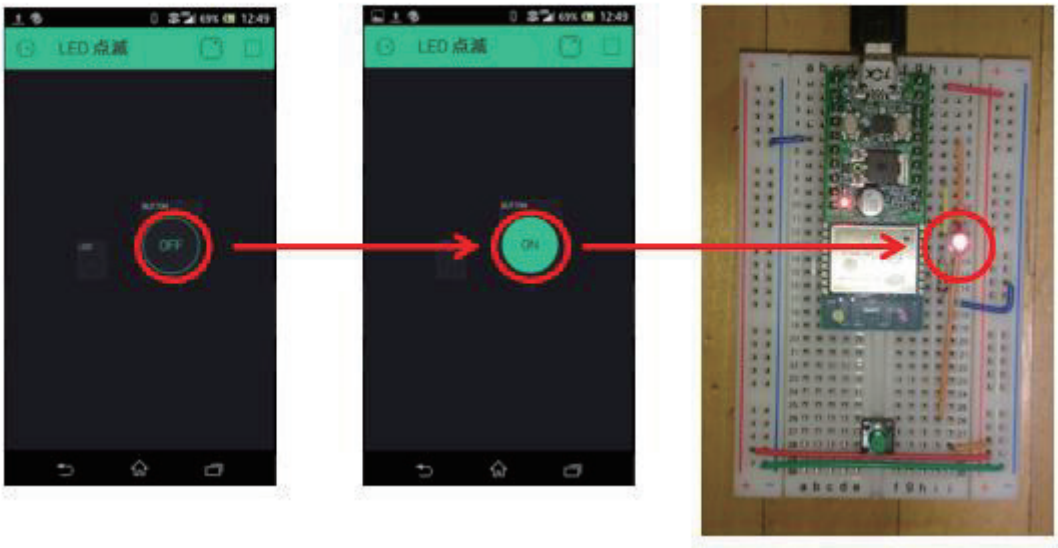

図　151

◇マイコンSW操作

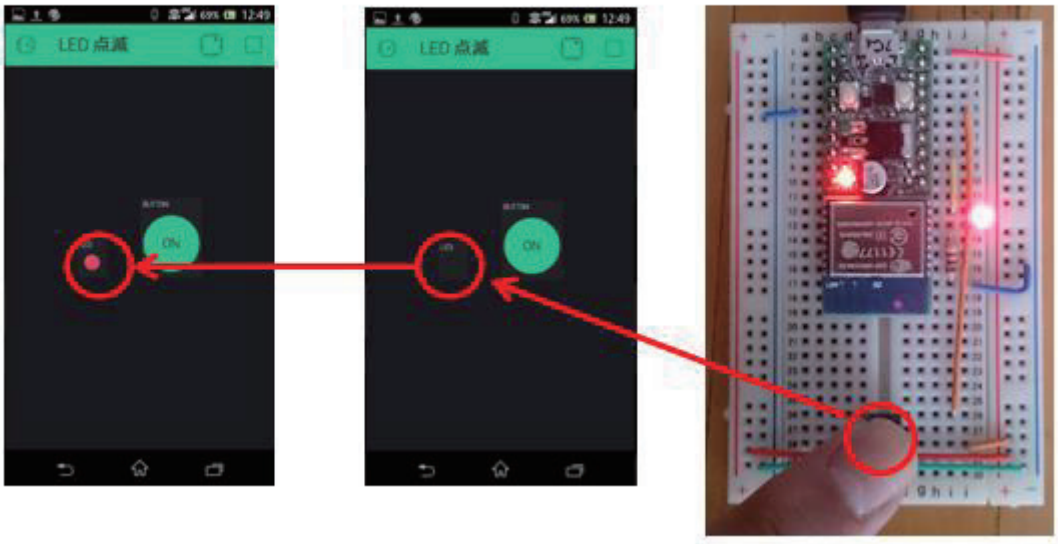

図　152

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、下図のように多くの 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

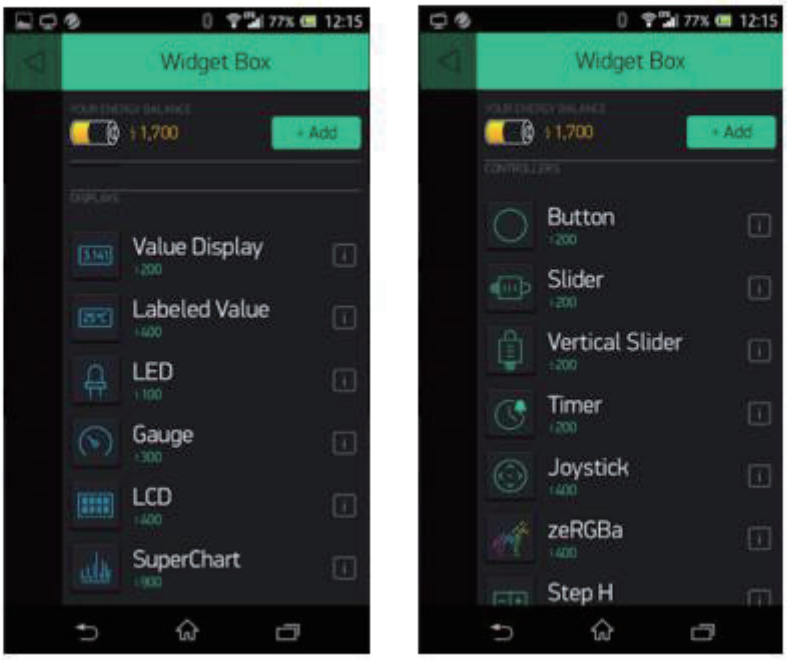

図 153

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。これまでの流れ【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

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。

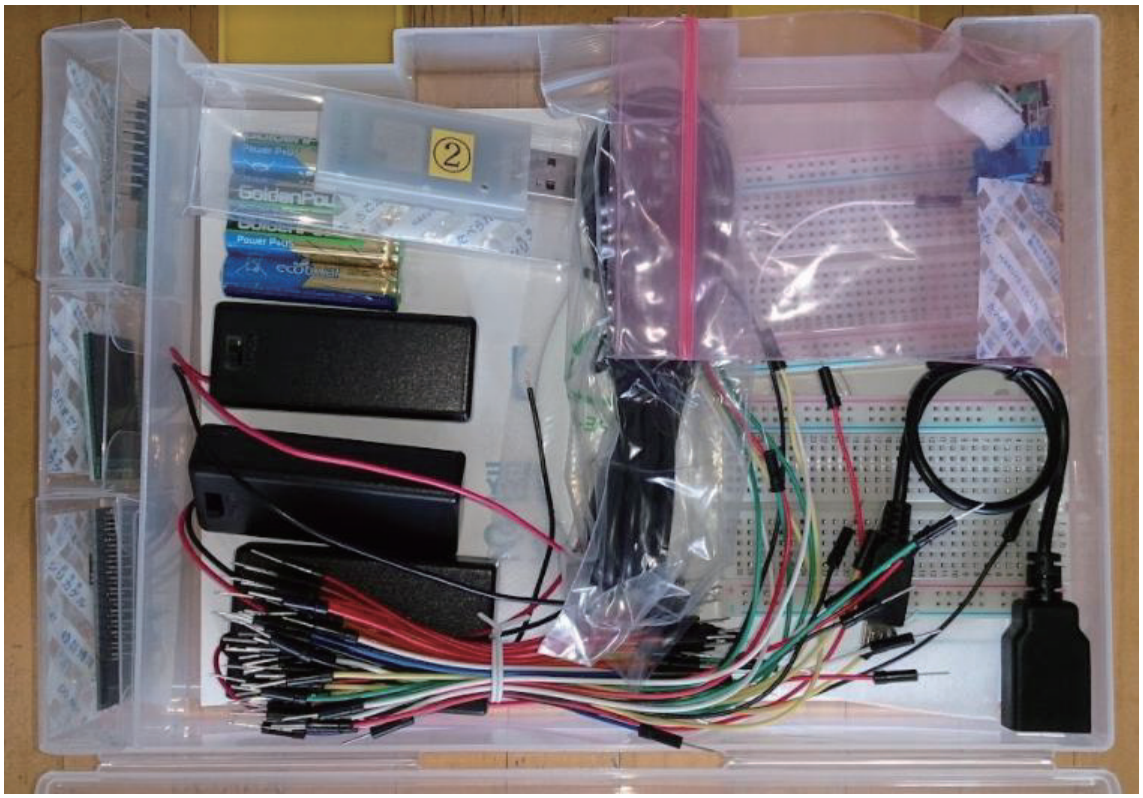

図 154

Appendix B: 使用する全パーツ

図　155

図　156

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、無線マイコン・WiFiマイコンを使った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。

平成 30 年度「専修学校による地域産業中核的人材養成事業」
情報通信技術に対応した組込みシステム開発技術者育成のモデルカリキュラム開発と実証事業

【IoT へのドリル２】ネットワーク設計・構築

平成 31 年 3 月

一般社団法人全国専門学校情報教育協会
〒164-0003　東京都中野区東中野 1-57-8　辻沢ビル 3F
電話：03-5332-5081　FAX　03-5332-5083