

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
(IoT 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

学成す者体系を学ぶ

要を得る者実を成す

皆さんはどちらのタイプでしょうか。平成 29 年度文部科学省専修学校委託事業として、IoT に関連する印刷教材となるテキストを作成することとなりました。IoT とは近年よく耳にするようになった言葉です。意味はもちろんご承知のことでしょう。私は長年いろいろなコンピュータシステムの開発に携わってきましたが、それらのどのシステムもお手本がありませんでした。すべて独自開発というやつで、とても手間がかかりました。使用してきた手法は、説明では巧く行きそうでしたが、実際にやってみるといろいろな問題に直面してきました。解決するには創意工夫と時間が必要でした。

最近では、WEB 上にあらゆる情報が公開されていて、手元にお手本が無くても、教科書が無くてもいろいろと勉強できる環境になりました。特に IT 関連は、情報公開の速度とボリューム、幅・奥行き全てにおいて抜群な環境です。そのような時代でも、探しても見つからない【要を得る】なにかがほしいものです。それは、実際にやってみて、巧く行く記録です。【要を得る】には、巧く行く記録が必要で、それを自分でトレースすることで、その情報を提供してくれた方の思いに応え、自身にスキルを付けて成長する。そんなことを考えながら、このテキストをつくりました。皆さんの【要】に応えられれば幸いです。 wiseman 原田賢一

【なんでもつながる！！IoT 基礎講座】

入門編

目次

実習の範囲	1
第1回 無線通信	7
第2回 双方向無線通信	25
第3回 スマートフォン連携	33
第4回 PC 連携	49
第5回 PWM	59
第6回 双方向 PWM	69
第7回 スマートフォン連携 PWM	75

第8回	PC 連携 PWM	83
第9回	温度センサー	99
第10回	液晶表示器(LCD)	121
第11回	デジタル温度計	135
第12回	SW 状態検出	145
第13回	WEB 連携①(MQTT)	155
第14回	WEB 連携②(MQTT)	171

【なんでもつながる！！IoT基礎講座】

入門編

実習の範囲



図 1

写真は、千葉県柏市・我孫子市などにまたがる手賀沼に隣接する農地の様子を 2017 年 8 月初旬に撮影したものです。大きく広がる水田に、元気に育つ稲の様子が写っています。稲穂も太くなっていて、良い収穫が見込まれます。写真をよく見ると、右側奥手にはいくつかのハウスも写っていて、稲だけではなく様々な野菜などが生産されていることが分かります。この写真では奥行き 3km ほどの田園風景ですが、ちょうど反対側（背中側）も同じように広い水田が広がっています。

農業分野の IT では、このような広いフィールドや環境をコントロールするハウスなどに、様々なセンサーなどを数多く配置して、それらが

取り込んでくれる環境情報を収集して、統計・解析などを行い、次の農業生産に活かす取り組みが盛んに進められています。



図 2

上の図は、政府が開催している会議【高度情報通信ネットワーク社会推進戦略本部（IT戦略本部）データ流通環境整備検討会 AI、IoT時代におけるデータ活用ワーキンググループ（第6回）】（2016年度開催）で配布された資料の一部です。これを見ると、第4次産業革命では、ロボット、ビッグデータ、人工知能そしてIoTへの取り組みが農業に関連して重要なテーマであることが分かります。ロボットの分野では、GPSセンサーによる測位情報の取得、人工知能では、画像センサーによる農産物の生育状態分析、ビッグデータとして得られる圃場（農

業生産を行う場所) の環境情報の AI 手法による解析・分析・統計、IoT として市場動向のタイムリーな把握・利用など、IT 技術 (特にセンサーの利用) なしでは実現できないことばかりです。このような取り組みは、最近始まったことではなく、すでに長年の研究開発と実証を繰り返したうえで、活用ができるようになってきた技術ですが、近年 IoT (Internet of Things) として、ネットワークを利用した情報収集・分析・活用技術に発展したものが、実際に利用される時期に入ってきたことの証です。

◇センサーなどをネットワーク接続する例

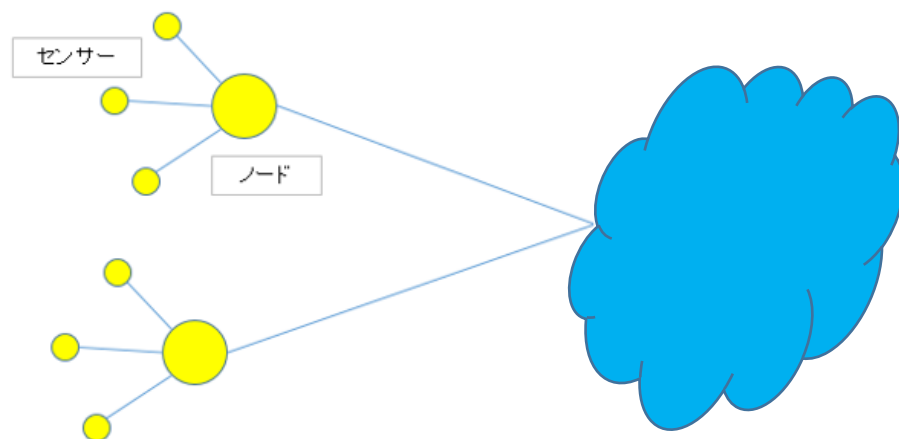


図 3

上の図は、センサーなどをネットワークに接続する簡単な例示です。フィールドで環境情報を取得するセンサーが繋がっているノードというもの考えてみましょう。図では、2 組しか描かれていませんが、実際は無数のノードが存在します。そして、ノードの上流にはネットワークが繋がっています。分かり易いようにとても簡単に描いたものなので、

実際とは違っています。もう少し詳しく描いたものが下の図になります。

◇中継を考慮する

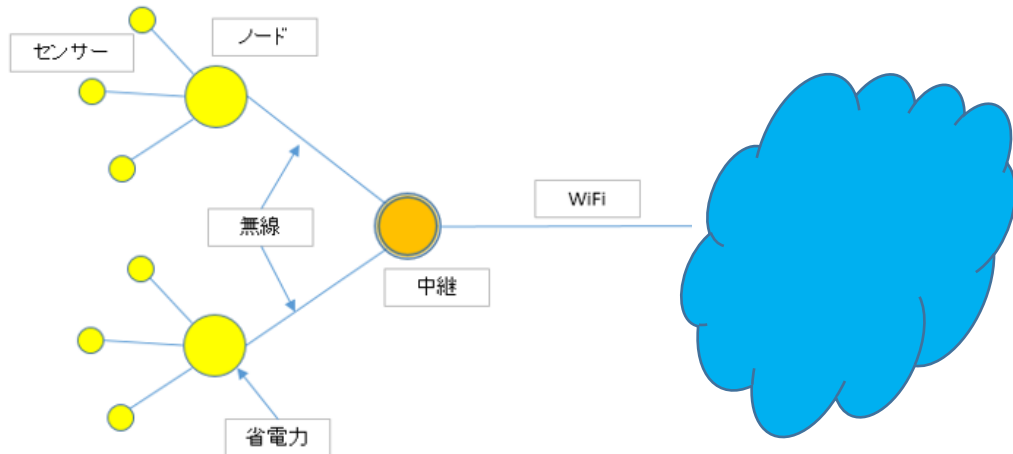


図 4

上の図では、センサーが繋がっている多数のセンサーノードを、直接ネットワークに接続することは現実的に難しいので、センサーノードからの情報を整理して取り纏める【中継機能】を配置することを考えています。センサーは広いフィールドに数多く配置されますから、そのような場所での電源確保を考えると省電力（低消費電力）であることが望まれます。当然そのようなノードを数多く有線で接続することは困難ですから、無線機能が欲しくなります。センサーノードにはセンサー制御機能と無線通信機能を持っていれば実現することができます。（従来は、マイコンと無線通信モジュールを組み合わせてこのようなユニットが実現されていました。）中継機能は、センサーノードからの情報を取りまとめてネットワークに流しますが、この接続も LAN ケーブルなどでの

接続では不便ですから、WiFiなどを用いて無線で接続することで圧倒的に利便性が高くなります。現在ではWiFiは近い距離での通信に限られますから、WiFiの接続先（AP：Access Point）も近くて電源も安定している場所を確保しようとする、屋内または建物のすぐ近くになるでしょう。

◇無線の対向ノードを考慮したモデル

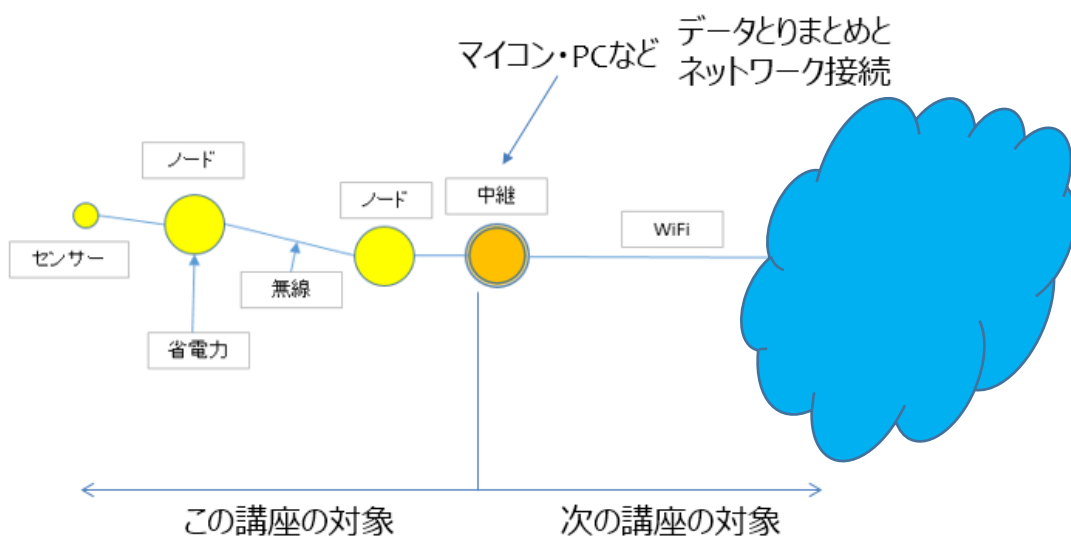


図 5

実習のできる、実際のユニットを考慮したモデルを考えたものが上の図です。

センサーノードはフィールドで運用されるので、電源には電池などが使えるものが良いはずですが。センサーノードの無線到達距離は数十メートル～数キロメートルが望まれますが、実際に利用できるモジュールでフィールドに沢山配置する低消費電力のものでは、数十メートル～数百メートルのものが現実になります。ノードに利用できる無線モジュールがWiFi機能を兼ね備えていると便利ですが、そのようなモジュールは

まだありませんので、センサーノードの無線を受ける対向ノードを配置することにします。その対向ノードを、中継機能を持つ機器に直接接続（有線接続）して、センサーデータがネットワークに流れる仕組みを実現します。中継機能は PC で開発します。

このテキストでは、中継デバイスからフィールド側を対象として実習し、中継デバイスから上流側（インターネット側）は、次の講座（第二分冊）の対象とします。しかし、中継機能を PC で開発しますので、講座の終盤では、PC を利用して WEB サービスとの連携実習も行います。

第1回 無線マイコン

この実習【なんでもつながる！！IoT 基礎講座】では、まず末端の部分の実験を行うことで、無線マイコンの機能を確認したいと思います。

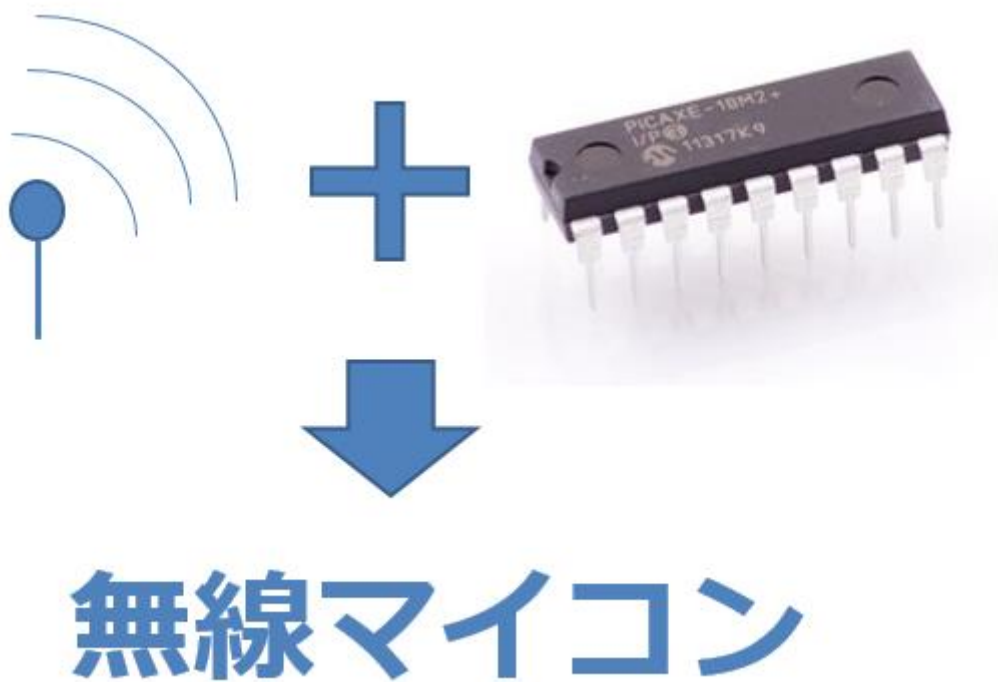


図 6

数年前までは、無線でマイコンを繋ごうとすると、上の図のように、無線ユニットとマイコンユニットを別々に用意して、それらを電氣的に接続し、マイコンにはプログラミングを行って、アプリケーションを書込んで無線マイコンを実現していたのですが、数年前から、それらが一つになった（無線モジュールとマイコンモジュールが1パッケージになった）無線マイコンモジュールが開発されて、色々な場所で実際に利用されるようになってきています。



TWE-Lite

図 7

上の写真は、TWE-Lite（と書いてトワイライトと読む）無線マイコンモジュールです。金属のパッケージ内部に無線ユニットとマイコンユニットを内蔵しているので、このモジュール1台で無線通信のできるマイコンとして機能するように設計されています。写真左側のものは、白くプリントされた部分（アンテナの絵）の裏側にアンテナパターンが配置されているので、写真右側のようにアンテナ突起物がなくて、コンパクトになっています。

- ◇TWE-Lite – トワイライト
- ◇東京コスモス電機(株)で開発
- ◇現在は、モノワイヤレス(株)
- ◇マイコン内蔵の無線モジュール
- ◇マイコンに独自アプリを書き込める



図 8

- ◇出荷時に標準アプリ書き込み済み
- ◇配線するだけで使える
- ◇中継機機能あり

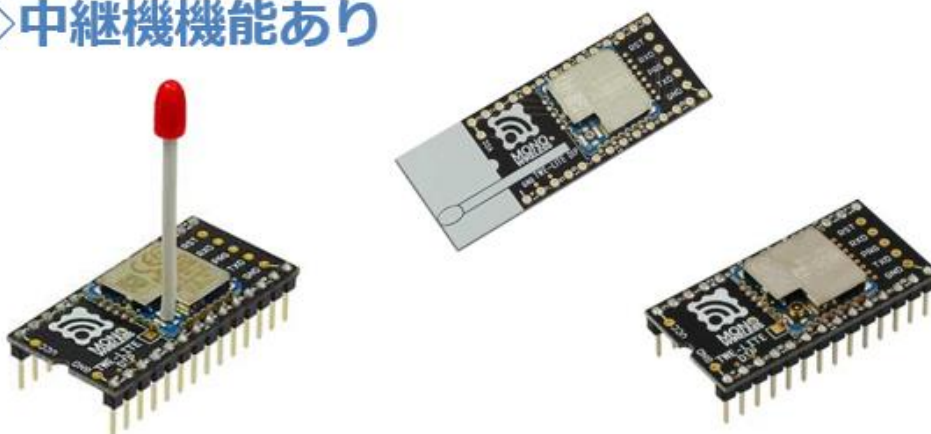


図 9

このモジュールは、東京コスモス電機(株)という会社で開発されたものですが、現在は、ワイヤレス部門が移管されてモノワイヤレス(株)という会社が関連商品を含めていろいろなモジュールを開発しています。マイコンを内蔵しているので、独自のアプリケーションを開発してユニットに書き込むことができますので、専用モジュールの開発も可能です。

出荷時には、標準の機能を持つアプリケーションが書き込み済みとなっ

ています。この標準アプリケーションの機能範囲での利用であれば、プログラミングレスで、配線をするだけで無線マイコンモジュールとして使用することができます。無線というと送信機と受信機の間での通信を行うわけですが、このモジュールは、ある設定を行うと、送信機と受信機の間に入って、中継器としての役割を果たしてくれるようになり、全体での通信距離が伸ばせるという機能があります。中継機としては、送信、受信機の間の中継器を 3 台（3hop）まで設定できるそうです。

<<無線を使う>>

・・・配線するだけです・・・

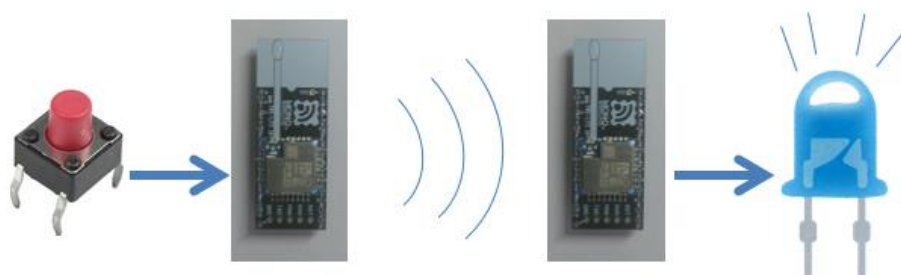


図 10

無線機能を使うとどのようなことができるのか、まず試してみることにしましょう。上の図のように、無線マイコンモジュールを 2 台使い、一方に SW、他方に LED を接続します。SW を押すとその状態が対向機のモジュールに無線で通知されて LED の点灯制御が、遠隔でおこなえるようになります。この機能は工場出荷時に書き込まれている標準アプリケーション機能の範囲内ですので、プログラミングレスで実現できます。

◇システムの全体構成

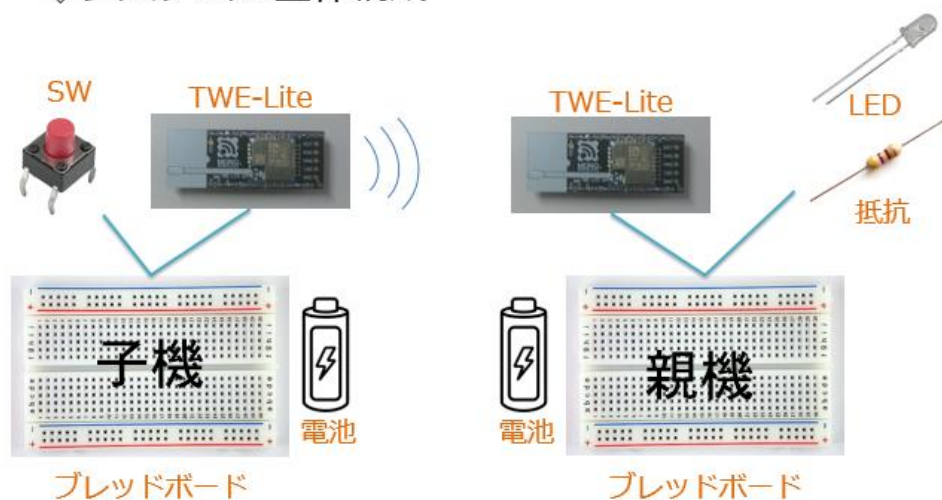


図 11

システム全体構成を上図に示します。説明の都合上 SW を接続する側を子機、LED を接続する側を親機とします。必要なパーツは下記です。

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. 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. LED×1 個
6. 抵抗器（470Ω）×1 個

各パーツは、1個でも購入することができ、通販などでも入手できます。無線マイコンモジュールは、親機・子機ともに全く同じものです。SWは「タクトSW」として販売されています。SW全体の大きさや押しボタン部分の色や長さの違うものがありますが、どれでも利用できます。ブレッドボードは、半田付けせずに配線ができるので大変便利です。

◇マイコンと周辺デバイスの接続に使います。

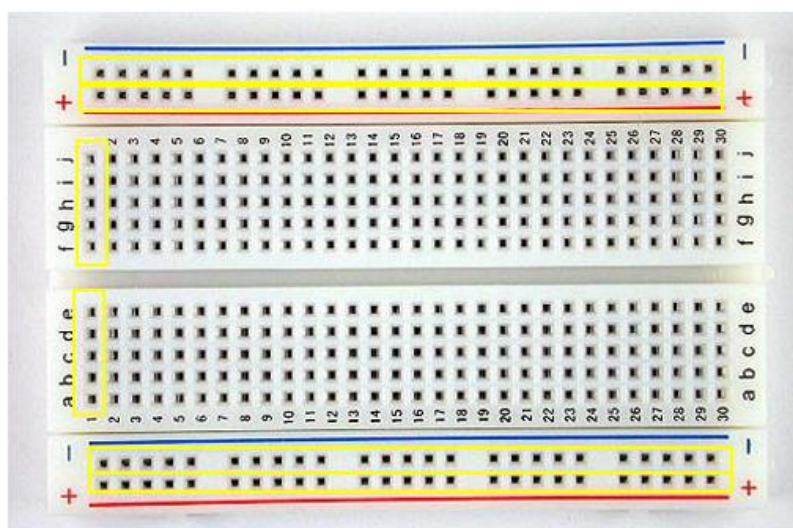


図 12

◇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。

上の写真のように、ブレッドボード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につないで使用します。

電源は、1.5V 乾電池を 2 個直列に接続して、3V として使用します。この実習で使用する無線マイコンモジュールは 3V で駆動できます。実習キットに含まれている電池ボックスは 2 種類あり、単 4 乾電池を 2 本使用するもの（下の写真）と 3 本使用するものがあります。この実験で使用する電池ボックスは、単 4×2 本のもので、間違えないようにして下さい。3 本のものを使ってしまうと、電圧が 4.5V になり、無線マイコンモジュールの電源電圧 (2.3~3.6V) の範囲を超えてしまうので、注意してください。また、ケースには SW が付いていますので、動作確認を行うまでは、SW を OFF にしておいてください。



図 13



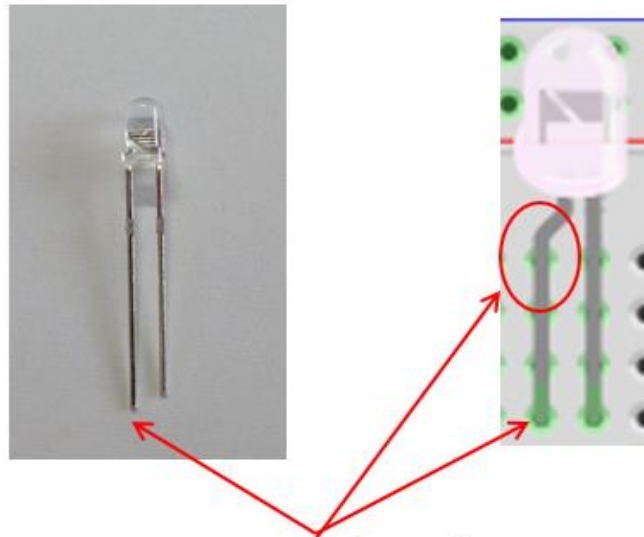
図 14



図 15

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 16

抵抗器は、カラーバー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。

- ◇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。

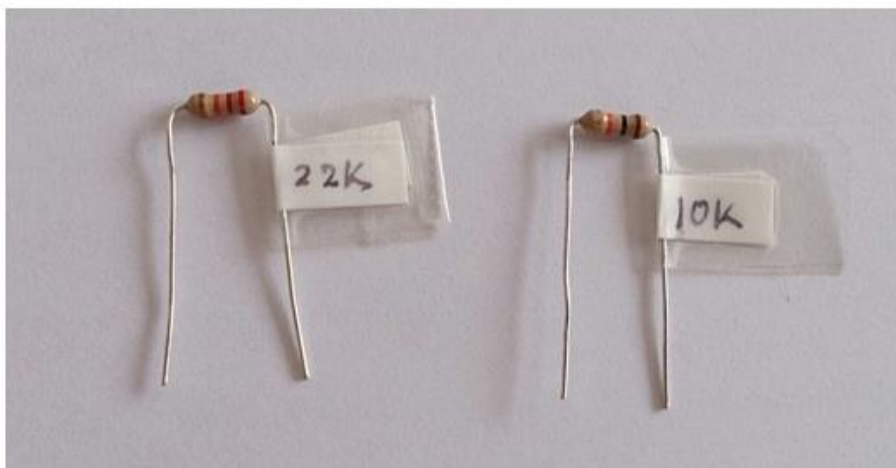


図 17

上の写真の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、種類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。ぜひ心がけてください。

実習キットにはすべてのパーツがそろっていますが、個々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。

無線マイコンモジュールは、次の写真の矢印の部分が弱いので、ブレッドボードへの取付け・取り外し（特に取り外し）の際に、強い力がかからないように、差し込むときはすこしずつ、取り外しには、専用工具やピンセットなどを使い少しずつ抜いて取り外してください。

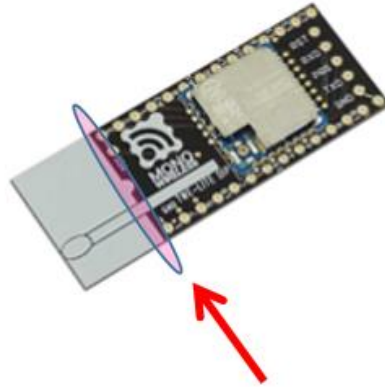


図 18

機能	信号名	シルク	ピン	ピン配置表			
電源グランド	GND	GND	1	28	VCC	VCC	電源(2.3~3.6V)
I2Cクロック	SCL	14	2	27	3	M3	モード設定ビット3
UART受信	RX	7	3	26	2	M2	モード設定ビット2
PWM出力1	PWM1	5	4	25	1	AI4	アナログ入力4
デジタル出力1	DO1	18	5	24	A2	AI3	アナログ入力3
PWM出力2	PWM2	C	6	23	0	AI2	アナログ入力2
PWM出力3	PWM3	I	7	22	A1	AI1	アナログ入力1
デジタル出力2	DO2	19	8	21	R	RST	リセット入力
デジタル出力3	DO3	4	9	20	17	BPS	UART速度設定
UART送信	TX	6	10	19	15	SDA	I2Cデータ
PWM出力4	PWM4	8	11	18	16	DI4	デジタル入力4
デジタル出力4	DO4	9	12	17	11	DI3	デジタル入力3
モード設定ビット1	M1	10	13	16	13	DI2	デジタル入力2
電源グランド	GND	GND	14	15	12	DI1	デジタル入力1

図 19

上の図は、標準アプリケーションが書込まれている無線マイコンモジュールの信号配置の表です。モジュールの半円形の切り欠き部分を上に

向けて、左側の上のピンから 1,2,3,4・・・と番号がついています。上の表の「ピン」の項目がピン番号です。シルクというのは、モジュールに印刷されている文字です。

-----<< 各ピンに配置されている信号 >>-----

1 番：電源グランドです。電池駆動の際はマイナス側に接続します。同じ機能のピンが 14 番にもあります。(GND)

2 番：I2C クロック。アイ・ツー・シーやアイ・スクエアード・シーなどと呼ばれる、2 線式シリアルインターフェースのクロック信号です。

I2C のデータは 19 番ピンに配置されています。

3 番：UART 受信。シリアル通信の受信信号です。送信信号は 10 番ピンに配置されています。

4 番：PWM 出力 1。Pulse Width Modulation という、一定周期で発生するパルスの幅で、デバイスを制御する信号です。全部で 4ch あり、4,6,7,11 番ピンに配置されています。

5 番：デジタル出力 1。ON/OFF の制御を行うデジタル信号なので、全部で 4ch あり、5,8,9,12 番ピンに配置されています。

13 番：モード設定ビット 1。このモジュールが電源投入後、起動する際に、このモード設定ビットの信号を GND (マイナス) に接続することで立ち上がった後の動作を決める信号です。全部で 3bit あり、13,26,27 番ピンに配置されています。

15 番：デジタル入力 1。SW などのデジタル入力信号を取り扱うピンです。全部で 4ch あり 15,16,17,18 番ピンに配置されています。

20 番：UART 速度設定。このモジュールのシリアル通信の通信速度は内部設定で変更することができます。設定変更した場合、この信号を

GND（マイナス）に接続することでその設定が有効になります。解放している場合は、通常の設定で動作します。

21番：Reset 信号。モジュールのリセット信号です。

22番：アナログ入力 1。センサーなどの出力電圧を測定する A/D 変換器を内蔵しているピンです。前部で 4ch あり、22,23,24,25 番ピンに配置されています。使用しないときは、この信号がふら付かない様に、電源の+または GND に接続しておきます。

28番：電源の+側です。この講座では電池の+（3V）に接続します。

.....

前掲のピン配置表は、無線マイコンモジュールに書込むアプリケーションで柔軟に変更ができるようになっていまして、書込まれたアプリケーションが異なると、ピンの機能も変わります。ですから、あくまでも【標準アプリケーションでのピン配置】と考えておいてください。

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・ジャンパ線での配線なので、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。

◇親機の配線

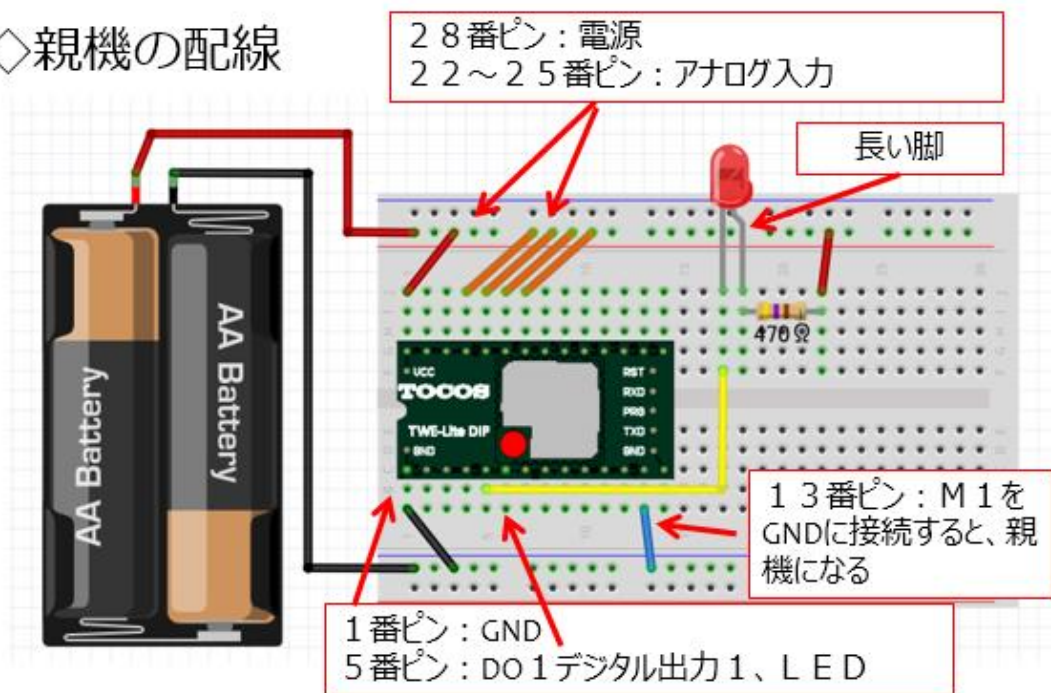


図 20

まずは、親機の配線です。ブレッドボードを小さい数字が左、アルファベットのAが左下になるように置いて、親機を上の方のように配線してください。電池ケースのSWがOFFの状態であることを確認して配線しましょう。上の図ではアンテナパターンが印刷されている白い部分が省略されていますので注意してください。実際にブレッドボードに無線マイコンモジュールを取り付けると、アンテナパターンの部分が左側にはみ出します。ピン配置は同じです。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。

1番ピンは、電源のGND（-）に、28番ピンは電源の（+）に接続します。22～25番ピンは電源の（+）に接続しておきます。このマイコン

モジュールはアナログ信号の変化により状態が変わると、対向機に状態が変化したことを知らせる無線通信が発生してしまうため、これを防ぐ目的で、アナログ入力のレベルを固定しておくためです。

13番ピンは、GNDに接続します。このようにすると、起動後のモジュールは親機として機能します。

LEDは、5番ピンからジャンパー線でLEDの短い方の脚に接続します。LEDは極性がありますので、誤って長い方の脚に接続すると意図したように動作しませんので注意して下さい。LEDの長い方の脚は抵抗器を経由して電源の(+)側に接続します。LEDには、5番ピンの信号レベルがLowになったときに、電源(+)側から、抵抗器を経由してLEDの長い方の脚から電流が流れ込み(LEDが光る)、短い方の脚から流れ出て、5番ピンに吸い込まれると考えて下さい。

実際に配線した親機の様子が次の写真です。

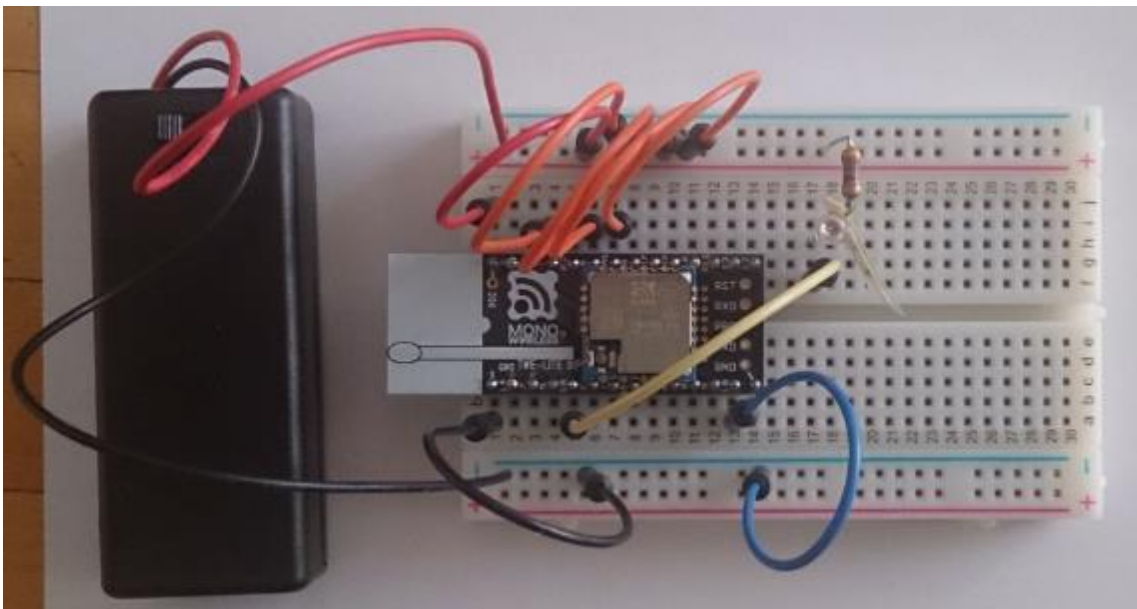


図 21

◇子機の配線

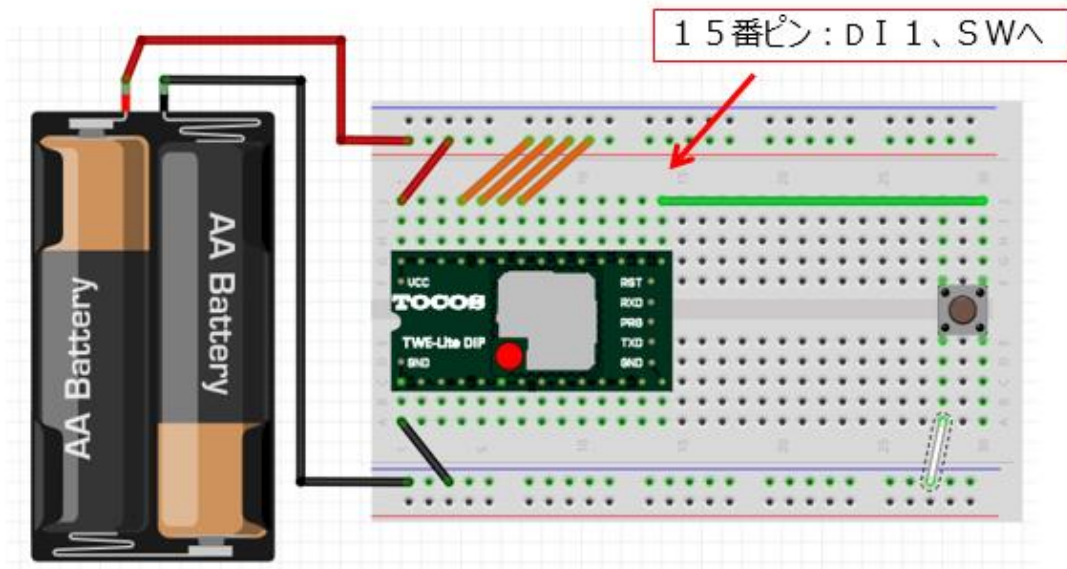


図 22

次に子機の配線を行います。子機は親機に比べて、とてもシンプルですが、侮ると配線間違いをしてしまいます。落ち着いて配線してください。親機と似ている（というより同じ）部分が多いですね。図の左半分は親機と全く同じです。親機の 13 番ピン（M1）の信号を GND に接続するジャンパー線がありません。繰り返しですが 13 番ピンを解放するか GND 接続するかによって、親機として振舞うか、子機として動作するかが決まります。

15 番ピンは、デジタル入力 1 の信号です。ここにジャンパー線で SW を接続します。SW の反対側は、ジャンパー線で GND に接続してやると、デジタル入力 1 の信号は SW を押したときには Low レベルになります。これをマイコンのプログラムで読み込むと該当するビットが 0（ゼロ）になります。この状態を対向機に知らせて、LED を点灯したいので、

【SW を押す→デジタル入力が Low→デジタル出力が 0→LED 点灯】と

なります。親機はデジタル出力ピンで電気を吸い込んで LED が光るように配線した理由が分かりますね。

図のジャンパー線の色は適当なものを使用して配線して構いません。大切なことは、どんなに簡単な回路でも、念入りに確認をすることです。

実際に配線した子機の様子が次の写真です。

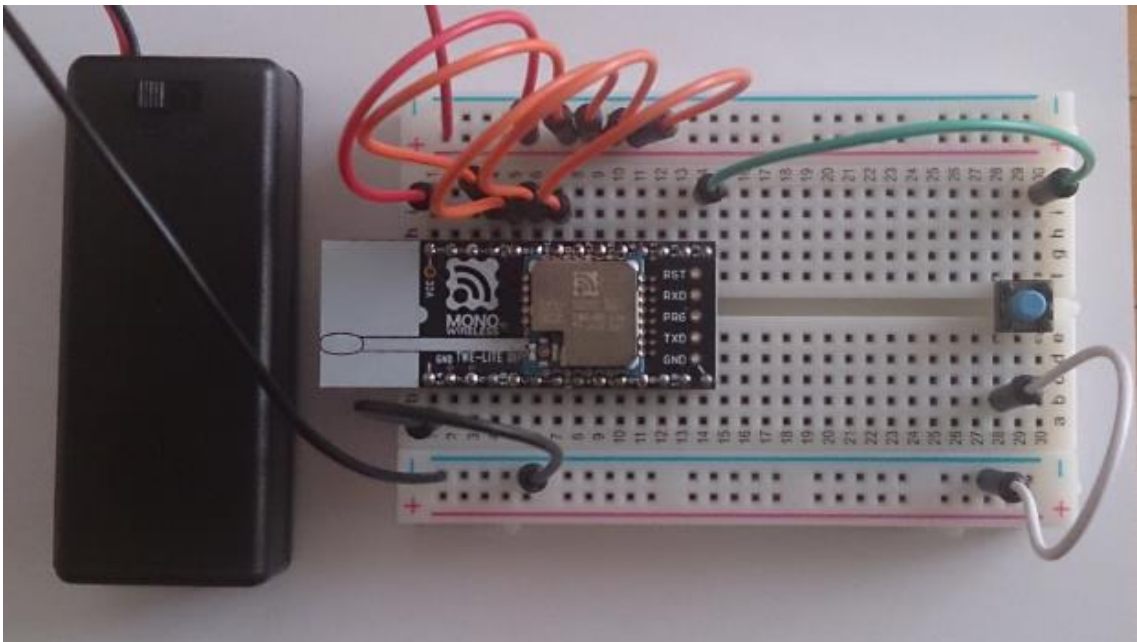


図 23

ジャンパー線を使うと、どうしても写真写りがごちゃごちゃしてしまいますので、しつこい様ですが、よく確認をしてください。

配線の確認ができたなら、いよいよ動作確認です。親機・子機の電池ボックスにある SW を ON 側にスライドさせてください。次に子機の SW を押してください。SW を押すと同時に親機の LED が光るでしょうか。SW から指を離すと LED は消灯するはずですが、巧くいかない様でしたら、配線をチェックしましょう。配線が間違っていないなくても、しっかりと差し込まれていないと接触不良で正しく動作しません。電池ボックス

の中の乾電池の向きはいかがでしょうか。確認することはいっぱいありますが、一つ一つ自分で行った作業ですから、自分で確認をして正しい動作をするように修正をしましょう。動作の様子を写真に示します。

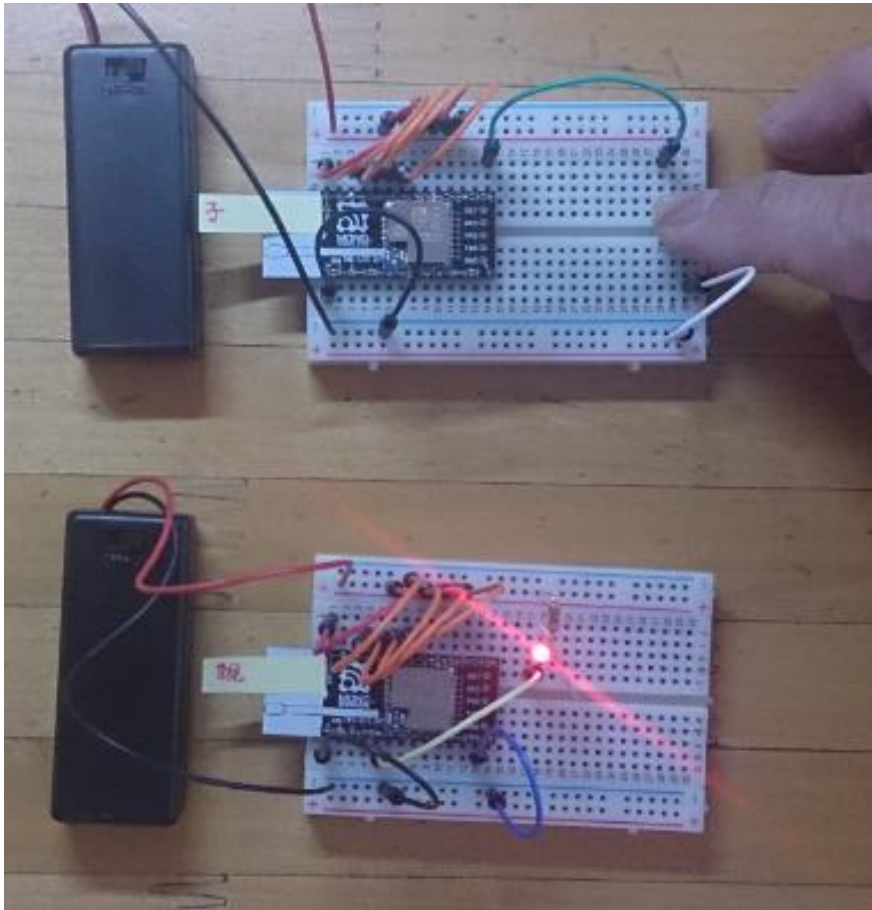


図 24

この回路が作れると、LED の代わりに例えばガレージの扉開閉や、ハウスの天井にある換気窓の開閉、ファンの起動などがリモートコントロールできるようになります。LED の ON/OFF 信号で AC (交流) を制御するときはリレーや SSR (ソリッド・ステート・リレー) などを使います。

第2回 双方向無線通信

次は双方向の無線通信で、機器の制御を行ってみましょう。第1回では、子機と親機を設定して子機側のSWの情報を無線通信で親機に送り、その内容に応じてLEDを点滅させることができました。第2回は、同じことを双方向で行います。

<<双方向で無線通信>>

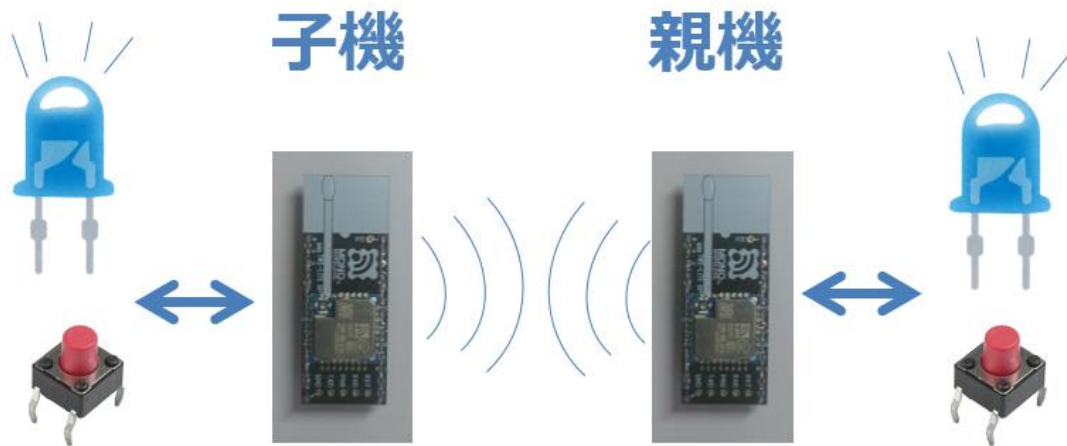


図 25

上の図は、今回の実習の内容を描いたものです。前回は親機と子機で担当する機能が違っていたので、無線マイコンモジュールに取り付けたデバイスは別のものでしたが、今回は双方向で同じことを行います。子機側のSWの状態を親機側のLEDに反映し、親機側のSWの状態を子機側のLEDに反映します。この機能も工場出荷時にマイコンに書き込まれている標準アプリケーションを使って実現しますので、プログラミ

ングレスが可能です。と、言うことは回路の配線だけでおこなえますので、配線の正確性が求められます。

◇システムの全体構成（親子同一）

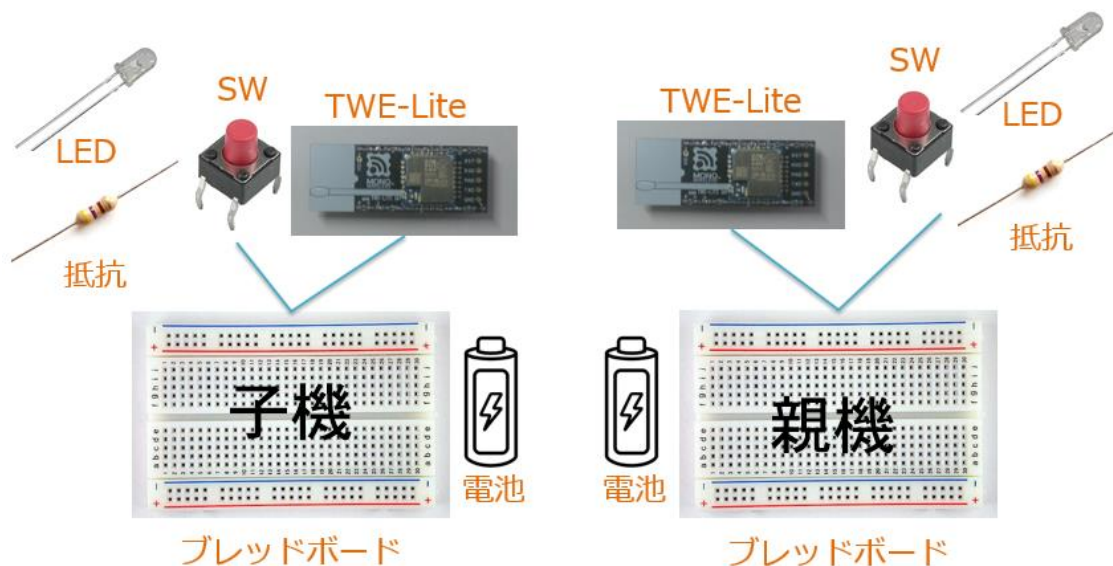


図 26

システム全体構成を上図に示します。親機・子機ともに同じパーツを使用しますので、必要なパーツは各々2組です。（下記）

子機側＋親機側：

1. 無線マイコンモジュール×2台
2. ブレッドボード×2個
3. 電池（単4乾電池×2個＋SW付電池ケース×1個）×2セット
4. 配線用ジャンパー線
5. SW×2個
5. LED×2個
6. 抵抗器（470Ω）×2個（LED電流制限用）

各パーツの内容は第1回の説明を参照してください。

◇親機の配線

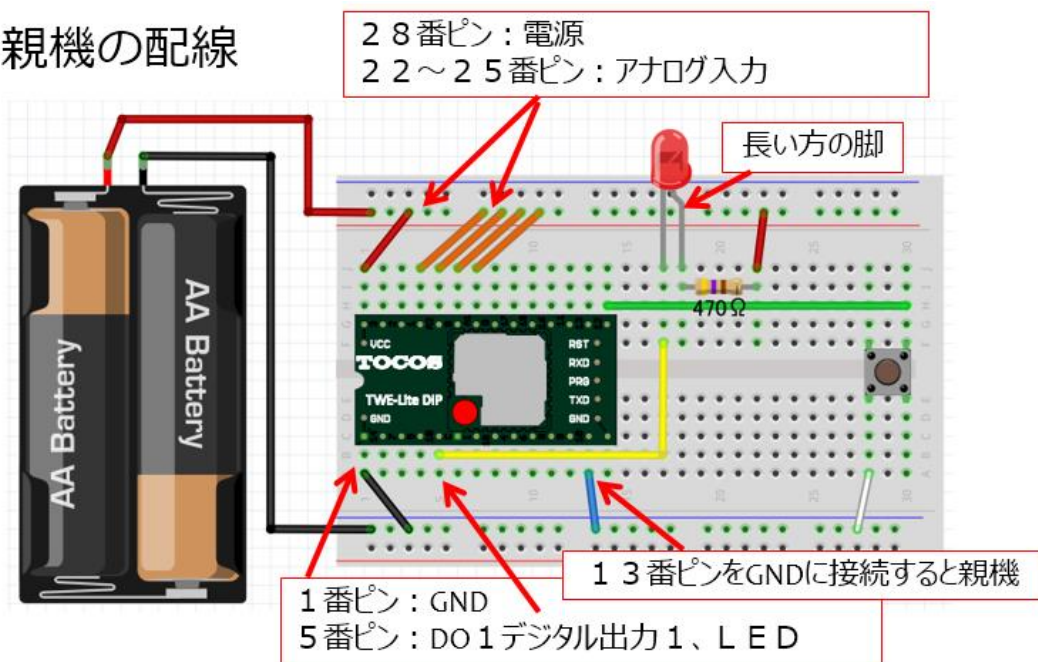


図 27

まず、上の図に従って親機を配線しましょう。第 1 回で配線した親機と子機をミックスした回路になっています。回路がそのまま残っているのであれば、親機には子機の SW を、子機には親機の LED と抵抗器を取り付けて配線します。新たに作成する場合は、第 1 回目の説明を思い出して配線してください。

実際に配線した親機の様子が次の写真です。LED と抵抗器、SW が同じブレッドボード上に配線されたので少しごちゃごちゃしていますが、配線に誤りはないでしょうか。写真中央下の青いジャンパー線が親機として機能する配線です。(第 1 回を参照)

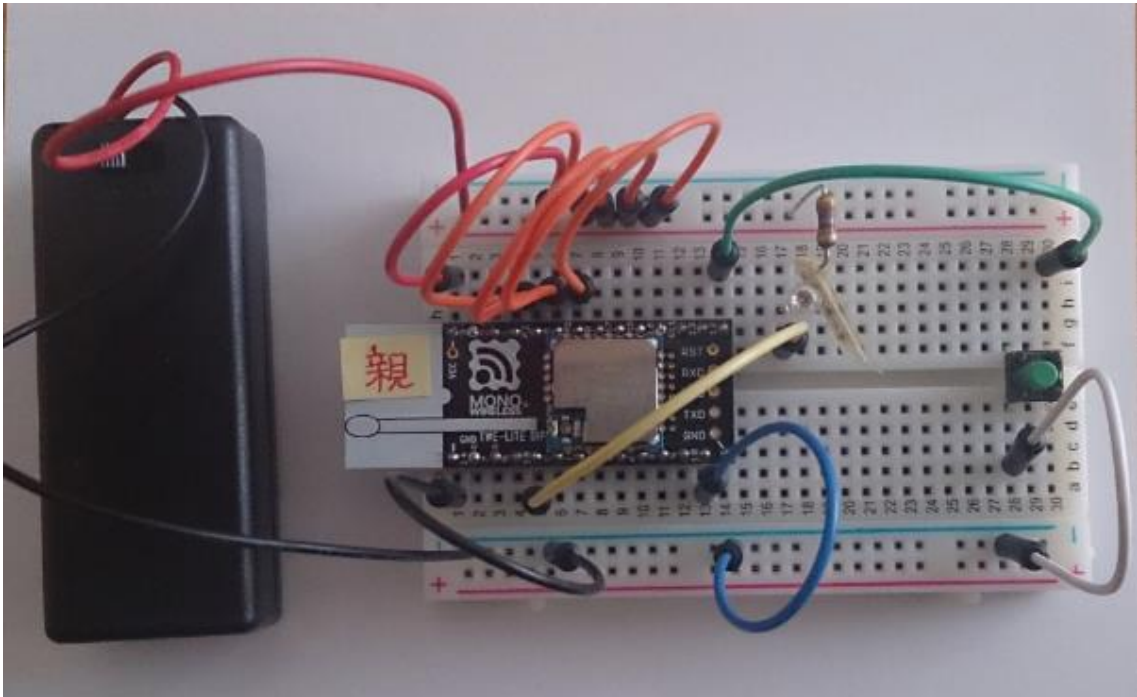


図 28

◇子機の配線

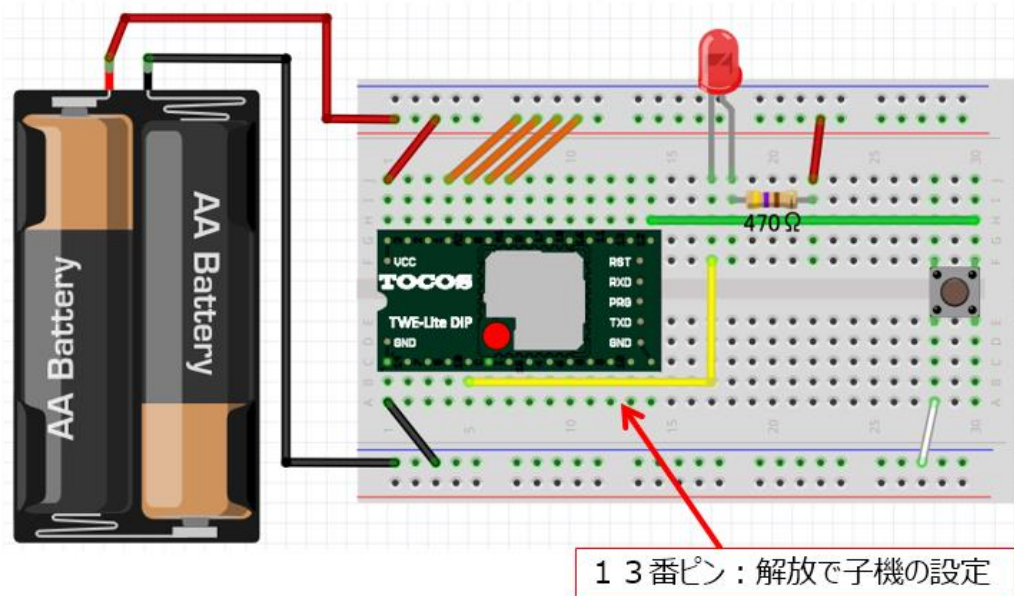


図 29

次に子機の配線を行います。上の図に従って配線します。すでに気づいたと思いますが、子機側の配線は親機の回路の 13 番ピンのジャンパ

一線が無いものです。13番ピンをGNDに接続するとこの無線マイコンモジュールは親機として機能し、13番ピンが解放ならば子機として機能します。ですから2組の同じ回路を作り、13番ピンのジャンパー線の有無で回路の区別をしても構いません。実際に配線した子機の様子が次の写真です。親機との違いはジャンパー線1本だけです。

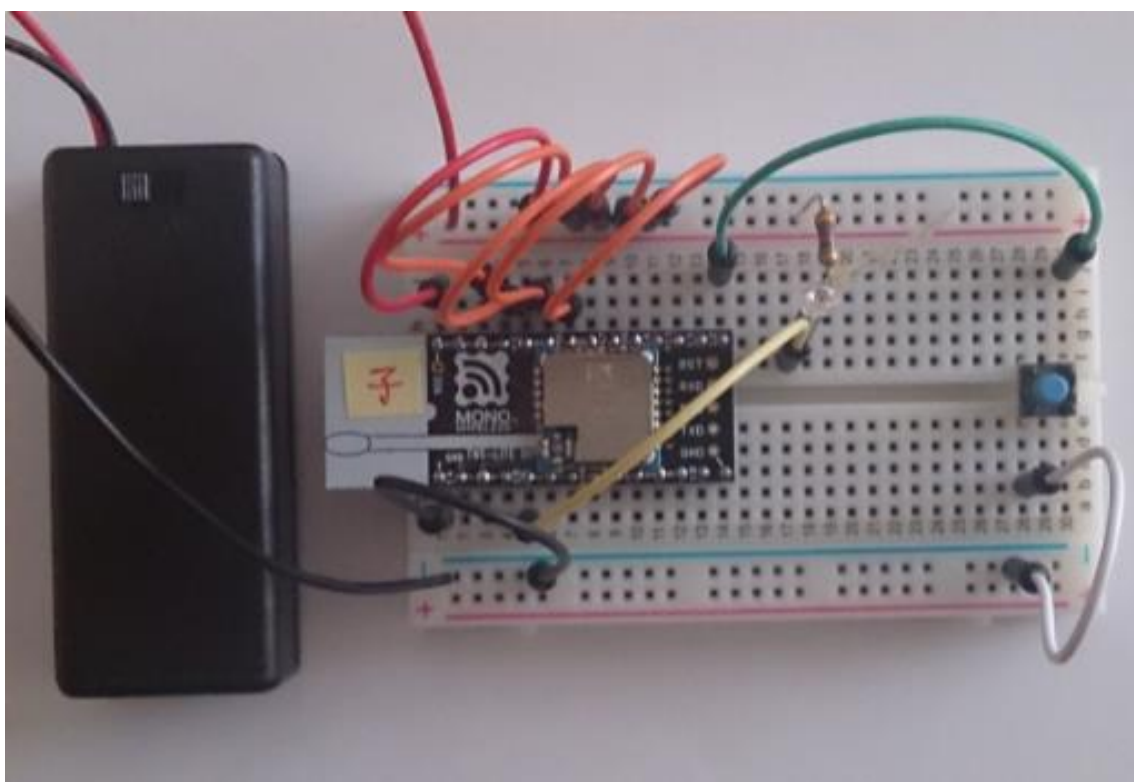


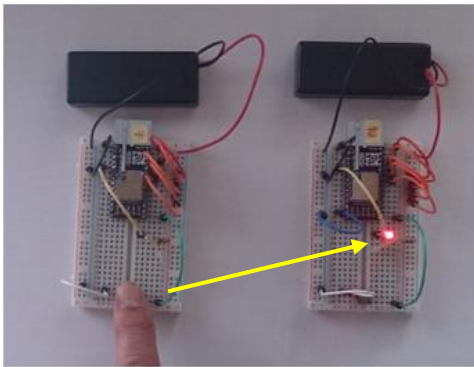
図 30

回路の作成ができたなら、油断せずに良く配線を確認してください。第1回の回路に配線を付加して作成した場合は、ジャンパー線やSWの緩みも確認して下さい。電池ボックスからの配線も要点検個所です。

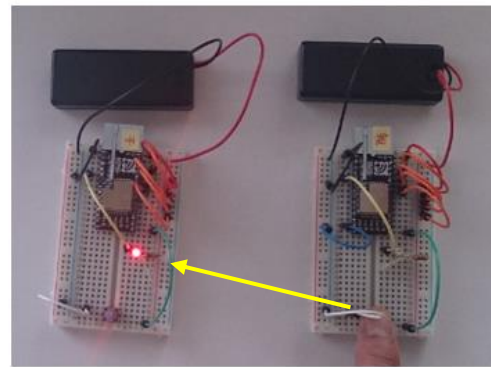
◇動作確認：

配線の確認が済んだら、電池ボックスの SW を ON にスライドして電源を入れます。まず子機の SW を押してみましよう。親機の LED が反応するでしょうか。次に親機の SW を押してみましよう。同様に子機の LED の点灯が確認できるでしょうか。両方の SW を細かく押したり離したりするとどうでしょうか。

◇とても簡単に双方向の無線通信ができました。
◇デジタル制御の通信がプログラミングレス！



子機から親機へ



親機から子機へ

図 31

◇無線到達距離について

今回のような双方向の無線通信ができると、二人で親機・子機それぞれを持ち、だんだん離れながらどのくらいの距離まで、通信ができるか測ることができますね。この講座で使用している無線マイコンモジュールは、アンテナの状態や周囲の電波環境が良ければ 1km ほどの到達距離があるとされています。無線通信の到達距離を伸ばすには次の点を確認しながら実験を行ってみてください。河川敷などの広い場所で他の電波の影響を受けないような環境で実験すると良いでしょう。この実験では携帯電話を機内モードにするなどとした手順を考えてみてください。

- ◇アンテナを親機と子機で平行になるように
- ◇アンテナ高さをできるだけ高くする
- ◇電波ノイズの少ない環境がよい
 - ※2.4GHz帯を使用しているので、
同じ周波数帯の機器との併用は要配慮
- ◇高さ3 m以上で、電波環境の良いところでは
およそ1 kmの距離で通信が可能

図 32

第3回 スマートフォン連携

この講座で使用している無線マイコンモジュールには、親機としてスマートフォンに USB 接続のできる機器が開発されています。このモジュールを使用することで、無線マイコンモジュールとスマートフォンが連携するシステムを作ることができます。

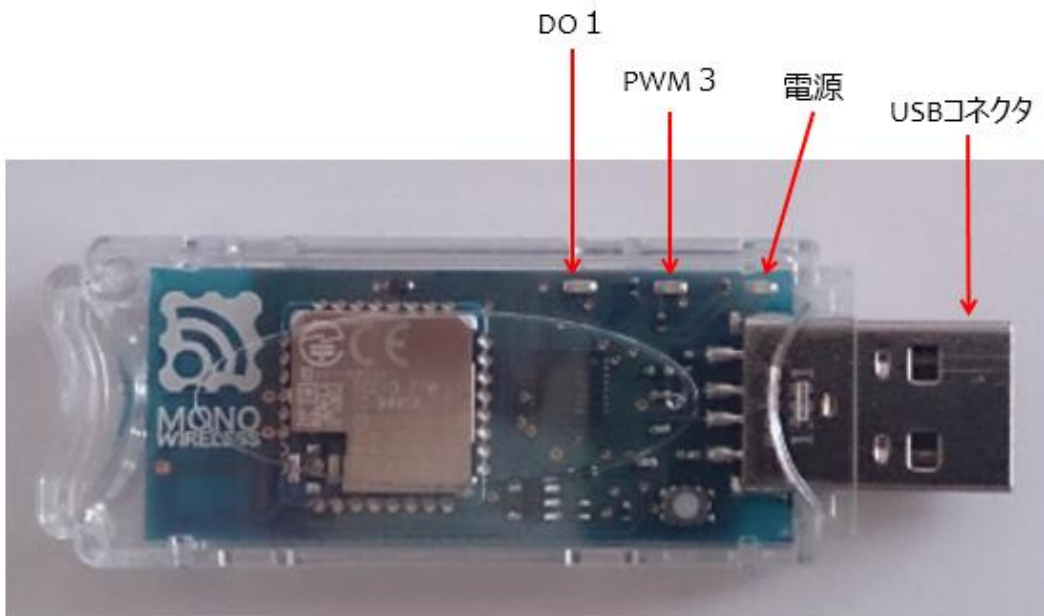


図 33

上の写真が MoNoSTICK というモジュールです。写真のモデルは、ケースが透明で、中の様子がよくわかります。USB コネクタを持っていて、これでスマートフォンに接続をします。一見 USB メモリと間違ってしまうような外観です。内部には基盤があり、これまで使用した無線マイコンモジュールと同様に金属ケースが中央にあります。この中にマイコンユニットと無線ユニットが入っています。基板左側のメーカーロゴが

印刷されているあたりの裏側にアンテナパターンが配置されていて、無線通信が行えるようになっていました。USB コネクタの付け根部分の上側には、小さい LED が基板直付けで配置されています。それぞれ、電源、PWM3 (PWM 出力の 3 番)、DO1 (デジタル出力の 1 番) の LED です。スマートフォンとの連携は下の図のようになります。

◇ USB I/Fでスマートフォンと接続



図 34

左側無線マイコンモジュールの子機と USB I/F を持つモジュール (親機) との間で双方向の無線通信をおこないます。スマートフォンで子機の様子をモニターしたり、あるいはスマートフォンから子機をコントロールしたりすることができます。スマートフォンには、そのままでは連携用のソフトウェアがありませんので、専用のアプリをインストールして使います。

◇システムの全体構成

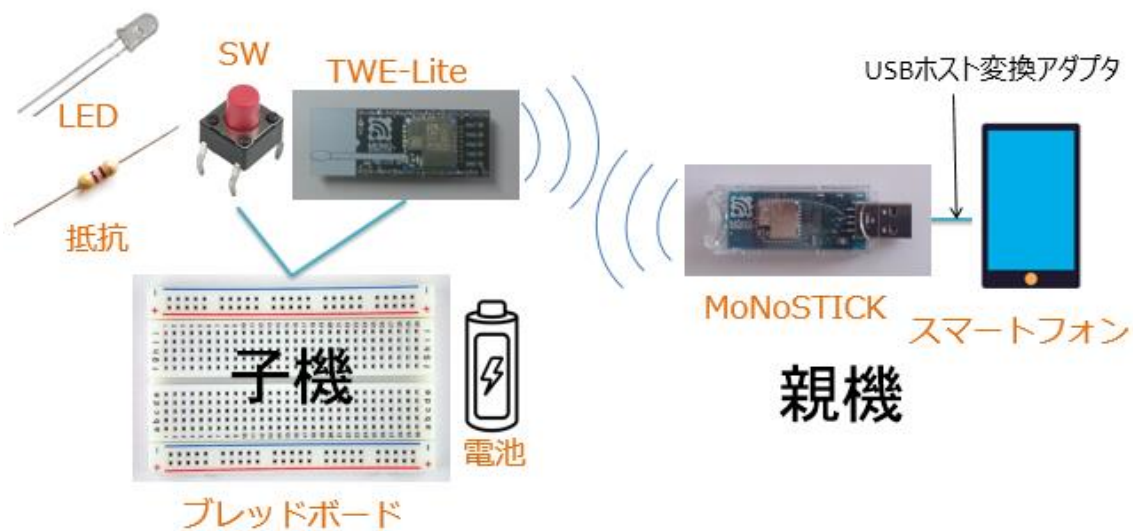


図 35

上の図がシステム全体構成です。今回の実習では、前回使用した子機と全く同じものを使います。子機側に配置した SW の状態をモニターしたり、LED を点滅させたりするので、子機は親機である MoNoSTICK と双方向無線通信を行います。親機である MoNoSTICK はわずかな LED しかありませんので、モニターや操作はスマートフォンの画面を通じて行います。MoNoSTICK の USB コネクタはそのままではスマートフォンに接続できませんので、次の写真に示す USB ホスト変換アダプターというものを使います。



図 36

USB ホスト変換アダプターは、一方に microUSB オスコネクタがあり、こちらをスマートフォンに接続します。他方には USB メスコネクタがあるので、こちらに MoNoSTICK モジュールを接続します。

使用するパーツは下記です。

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
5. LED×1 個
6. 抵抗器（ 470Ω ）×1 個（LED 電流制限用）

親機側：

1. MoNoSTICK×1 台
2. USB ホスト変換アダプター×1 本
3. スマホアプリ TWE Control×1 セット

◇子機の配線

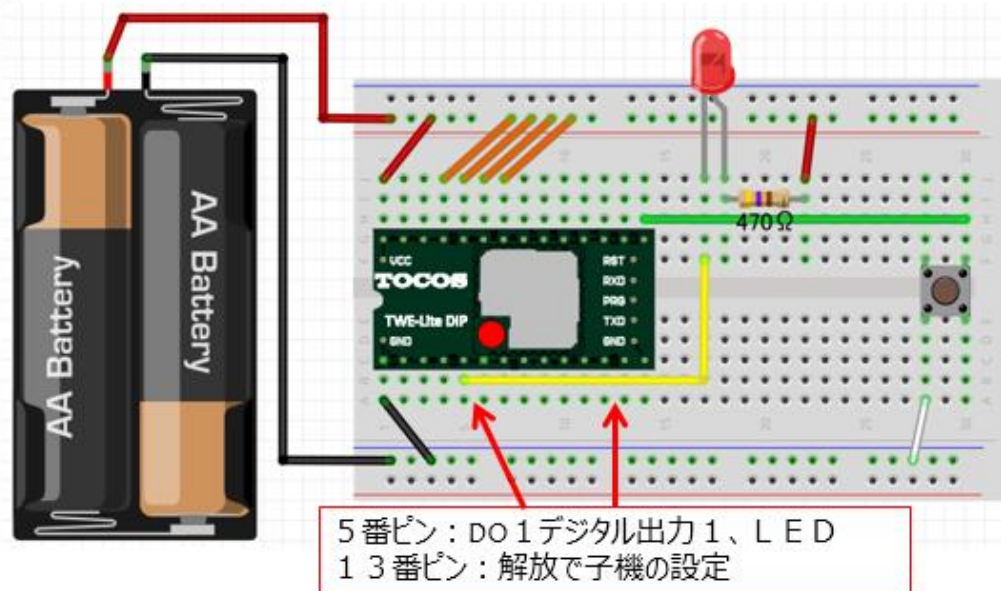


図 37

この実習で使用する子機は、第 2 回で使用した子機と同じものです。すでに作成済みの子機を利用される方は、親機と子機を間違えないようにしてください。13 番ピンが解放されているものが子機です。新たに作成される方は上の図をよく見て間違いの内容に配線を行ってください。実際に作製した子機の様子が下の写真です。

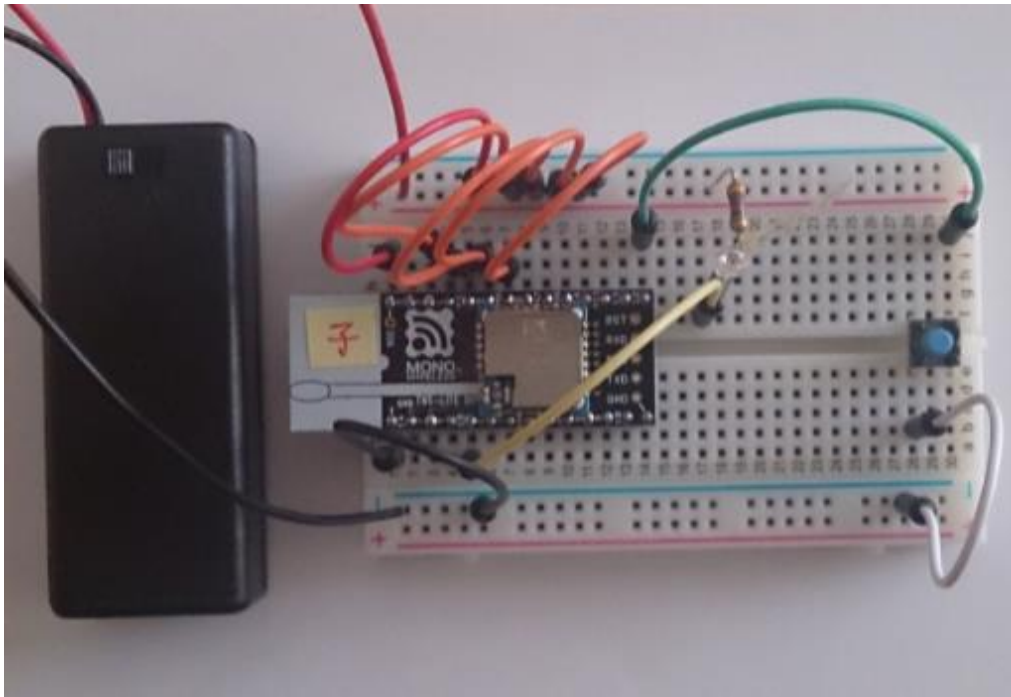


図 38

今回の実習では Android フォンを使用します。メーカーが開発したアプリで【TWE Control】というソフトウェアが公開されています。Google Storeなどで検索すると容易に見つかりますので、これをダウンロード、インストールしてください。(※この原稿執筆時にメーカーが公開しているアプリは Android 向けのものになっています。)

- ◇実験ではAndroidフォンを使用
- ◇アプリ【TWE Control】をインストール
※Google Storeなどで配布



図 39

次に、MoNoSTICK を準備します。次の写真のように、MoNoSTICK を USB ホスト変換アダプターで Android フォンに接続してください。写真の左下にあるのは子機です。

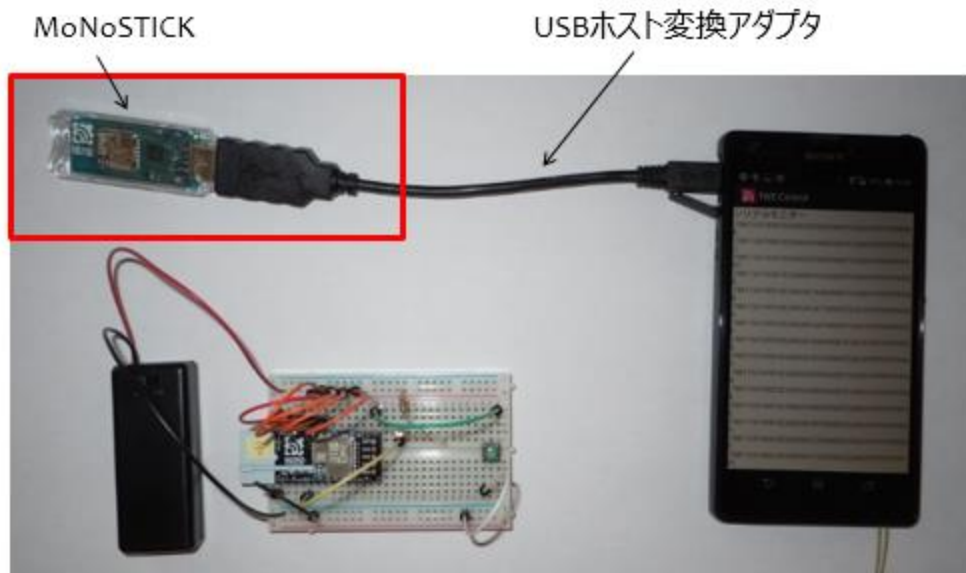


図 40

次に、子機の電池ボックスにある SW を ON にスライドします。子機の電源を入れて数秒後 Android フォンにインストールしたアプリ TWE Control を起動します。画面中央の【接続】をタップします。
(下図) 接続できると【未接続】から【接続中】に表示が変わります。



図 41

◇アプリ起動 接続、遠隔操作を選択

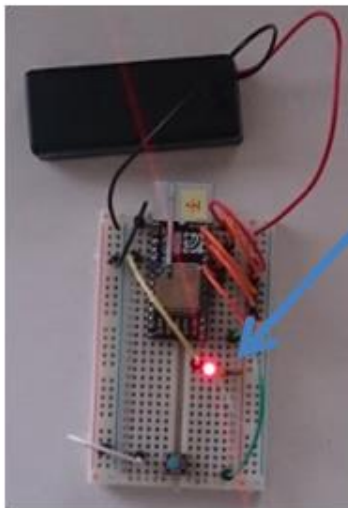


図 42

メニュー画面の下の部分のボタンが有効になりますので、【遠隔操作 (リモートコントロール)】をタップします。(上の図) すると、画面が右のように変わります。この画面でいろいろな操作をおこないます。

スマートフォンで操作

◇ DO1
ON/OFFをタップ



LEDが点滅



図 43

まず上の図の右側、スマートフォン画面の上にある DO1 ON をタップすると子機の LED が点灯します。次に DO1 OFF をタップすると LED は消灯します。いかがでしょうか。プログラミングレスでスマートフォンから子機を容易にコントロールできたと思います。画面のボタンを眺めると DO2~DO4 というのがありますね。もうお分かりの通り、無線マイコンモジュールのデジタル出力の 1~4 (DO1~4) のピンを使用すると、デジタル 4ch の無線コントロールが可能なのです。

次にシリアルモニターという機能がありますので、それを使ってみます。

シリアルモニタの準備

◇アプリで、シリアルモニタを選択

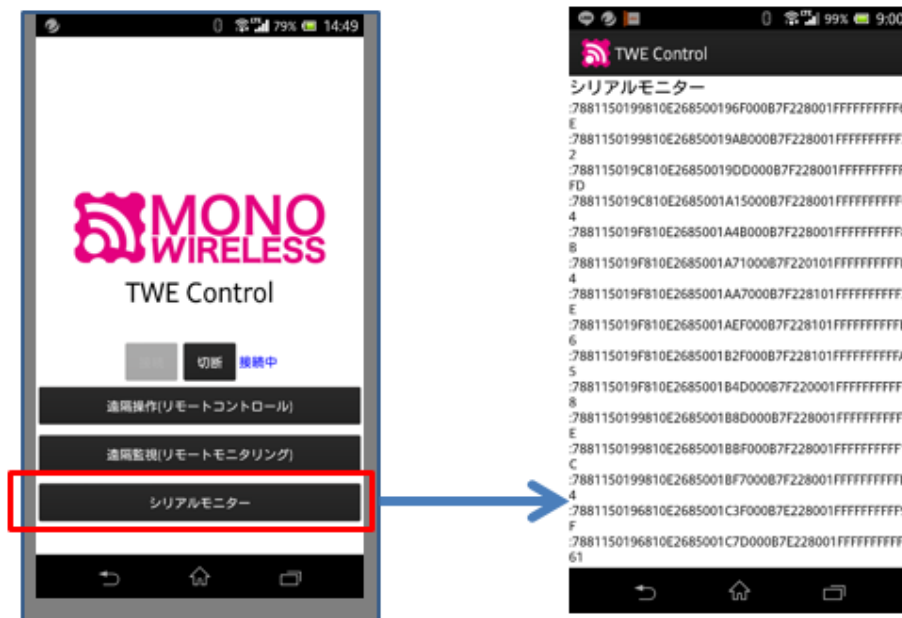
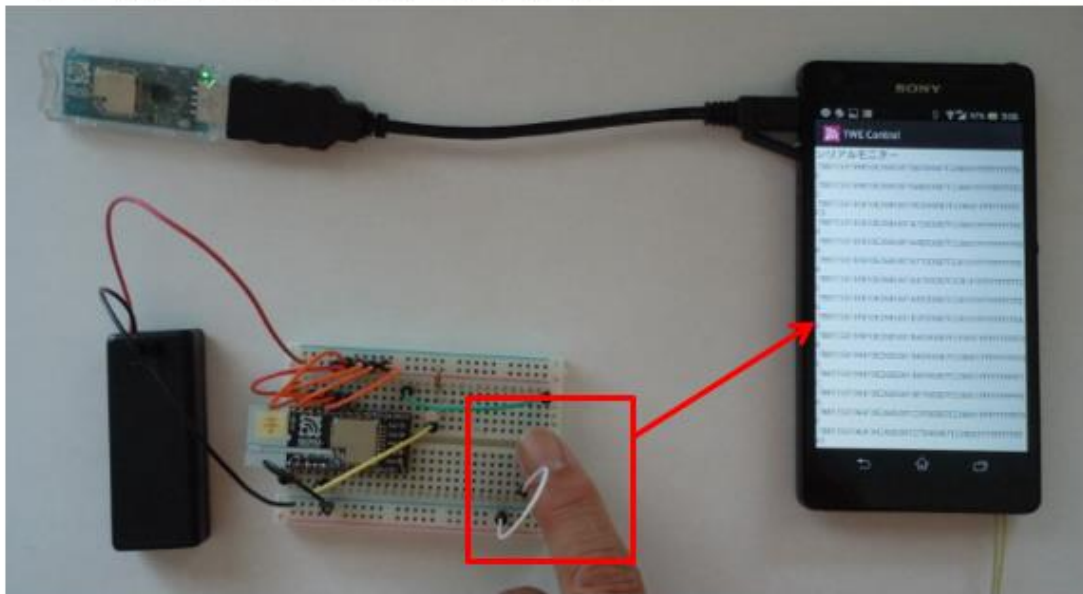


図 44

上の図、メニューで【シリアルモニター】をタップします。画面は右のように変わります。見ていると連続する 16 進数の文字列が順次表示されていきます。この 16 進数文字列の中に子機の情報が含まれています。子機の SW を ON/OFF するとシリアルモニターの表示文字列に変化が起こります。

◇子機のSWをON/OFFする



◇シリアルモニタに変化が見える

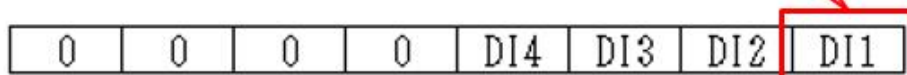
図 45

上の写真では文字列の変化が分かりませんので、その部分を拡大して切り出したのが次の図です。

デジタル入力の変化

◇デジタル入力を表すビットが変化

```
:788115017B810E2685000011000B6B208000FFFFFFFFF1A <---起動↓
:788115017B810E2685000053000B6D208000FFFFFFFFFD6↓
:7881150184810E268500008B000B71208000FFFFFFFFF91↓
:7881150187810E26850000D7000B70208000FFFFFFFFF43↓
:7881150187810E26850000F9000B702001FFFFFFFFF9F <---SW ON↓
:7881150184810E2685000131000B6F208101FFFFFFFFFEA↓
:7881150184810E2685000167000B6F208101FFFFFFFFFB4↓
:7881150184810E268500019F000B70208101FFFFFFFFF7B↓
:788115017E810E26850001BB000B6F20001FFFFFFFFFE7 <---SW OFF↓
:7881150187810E2685000207000B6F208001FFFFFFFFF11↓
:7881150184810E268500023D000B6F218001FFFFFFFFFDD↓
:7881150187810E2685000273000B70218001FFFFFFFFFA3↓
```



デジタル入力の値 レイアウト

子機のSWに対応する

図 46

図のように【:78】で始まる文字列が出力されています。これが子機から親機に無線で送られたメッセージです。子機の SW を押した（ON した）とき、離れた（OFF した）ときに変化している（赤枠）ことがわかります。この赤枠内の右側の桁は 8bit の下位 4bit であることがわかります。この下位 4bit で子機のデジタル入力 1～4 の変化を通知しています。より詳細にモニターして子機の SW 状態に対応するコントロールを行いたいときは、この文字列を解析して対応する制御を行うソフトウェアを開発すればよいのです。デジタル入力の情報以外にも多くの文字が並んでいますが、これらは子機側からの情報通知に使われています。

メーカーの WEB 情報でこれらの内容は公開されていますので、参照してみると良いと思います。

◇親機のDO 1、LEDが点灯！

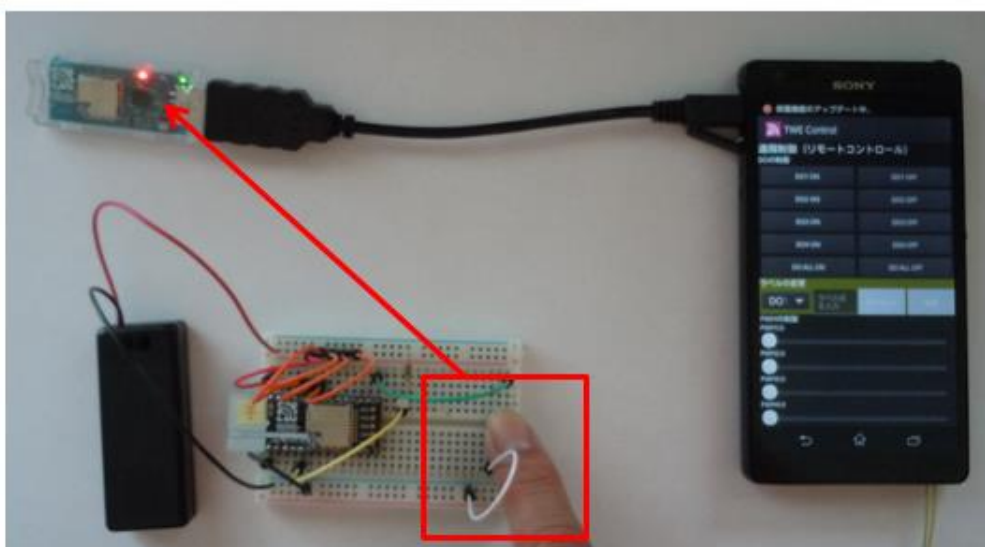


図 47

子機の SW を ON/OFF すると親機である MoNoSTICK の中にある LED が点滅していることに気が付かれたでしょうか。(上の図) スマートフォンのボタンをタップすることにより子機の LED を制御しながら、子機の SW の状態に応じて親機 LED の制御も行われているということです。特別なことをしなくても、双方向で無線通信が常時行われていて、その機能を簡単な配線を行うだけで利用できるのは、IoT にもってこいのモジュールと言えるのではないのでしょうか。

とても簡単、スマホモニタ

- ◇スマートフォンから子機をリモートコントロール
- ◇子機から親機をリモートコントロール
- ◇子機の様子をシリアル通信で受信



図 48

子機のリモートコントロール、親機のリモートコントロール（LED）、子機の様子をシリアルモニターすることなどをスマホアプリで行っていましたが、このアプリには、他にも機能がありますので、紹介しておきます。

他にも便利な機能があります



図 49

上の図、左はこのアプリのメニューから【遠隔監視】をタップした時の画面です。電圧グラフというボタンがあります。これは、子機のアナログ入力に与えられた電圧の変化をグラフにして表示する機能で、上の図、中央のように電圧変化をグラフ表示してくれる機能です。また、右側には温度グラフというボタンがありますが、これは LM61BIZ という温度センサーをアナログ入力ピンに接続したとき、この機能を使うと、子機が置かれた場所のセンサーで計測した温度変化をグラフ表示してくれます。

第4回 PC 連携

前回行ったスマートフォンとの連携を PC に置き換えて行ってみましょう。PC との連携は次の図のようになります。

◇USB I/FでPCと接続



図 50

子機としての機能は第 3 回と変わる部分はありませんので、前回と同じ回路が使えます。システム全体の構成もスマートフォンが PC に置き換わっただけです。PC には USB コネクタが備わっているので、変換ケーブルなども必要ありません。

◇システムの全体構成

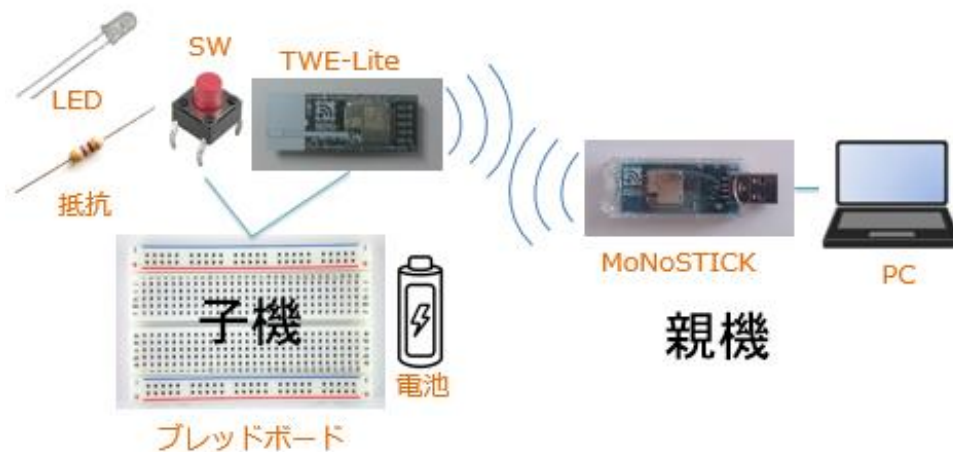


図 51

親機を通じて行う子機のモニターや操作は、PCの画面でおこないます。使用するパーツなどは下記の通りです。

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. SW×1個
5. LED×1個
6. 抵抗器（470Ω）×1個（LED電流制限用）

親機側：

1. PC×1台（Windows10＋Internet接続）
2. Windows専用アプリ×1セット

子機の配線（次の図）は、前回と同じものです。回路が残っていればそのまま使えます。前回の回路をそのまま使う方は、配線やデバイスの緩みなどが無いかよく確認して下さい。新しく作られる方は図を参考に作成して下さい。

◇子機の配線

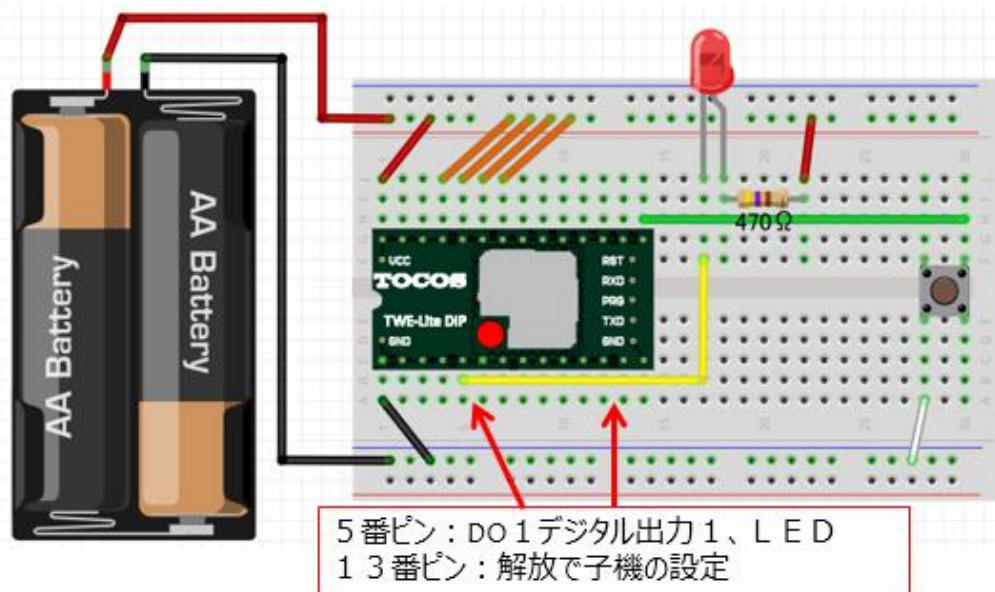


図 52

実際に作成した回路が次の写真です。

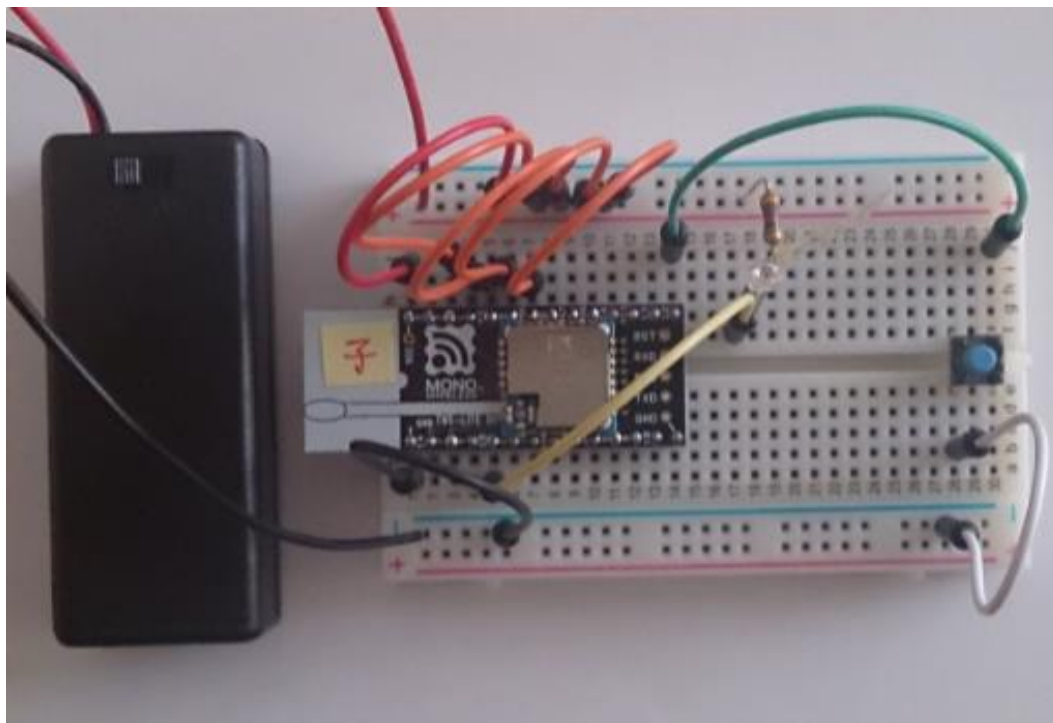


図 53

次に、メーカーWEB ページから、Windows 専用アプリをダウンロードします。

(<https://mono-wireless.com/jp/products/TWE-APPS/index.html>)

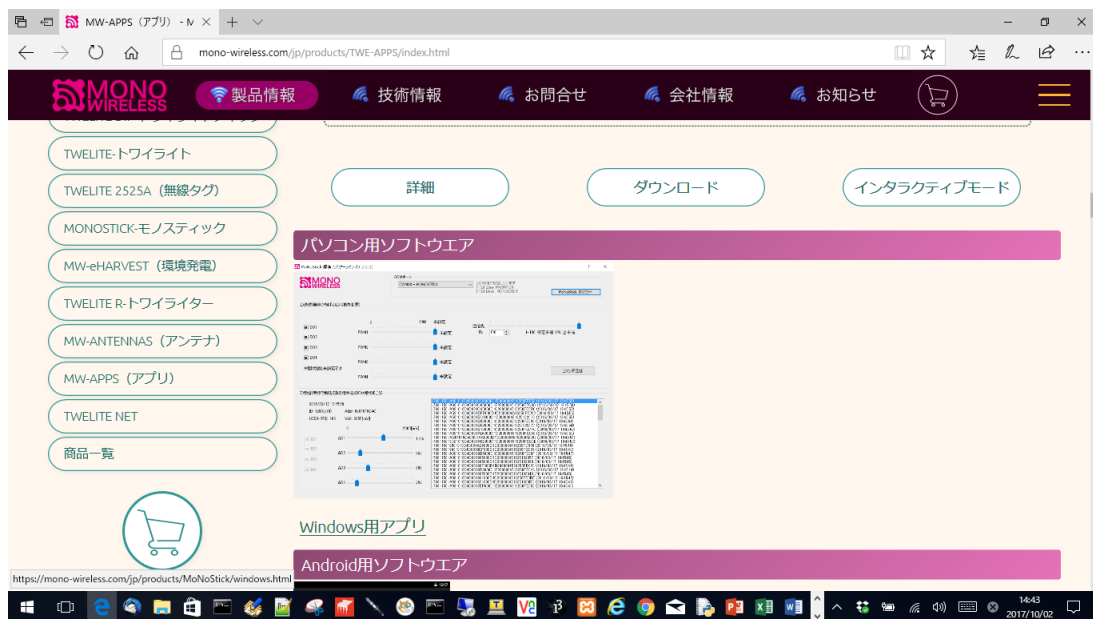


図 54

圧縮ファイルがダウンロードされますので、適当なフォルダに解凍します。その中に含まれる EXE ファイルが Windows アプリです。(下の図)



図 55

インストールは必要なくこの実行形式 EXE ファイルを開けばアプリケーションが起動するようになっています。

PC の USB コネクタに MoNoSTICK モジュールを差し込むと、仮想 COM ポートというシリアル通信のポートが設定されます。デバイスドライバで確認をすると COM ポート番号が確認できます。

◇ アプリ起動 接続COMポート確認



図 56

Windows アプリを起動して最初に行うことは、この COM ポートの確認と設定です。上の画面の上部にある【COM ポート】に仮想 COM ポート番号が設定されていなければ、プルダウンリストの中から、該当する COM ポート番号を選択します。

PCアプリで操作

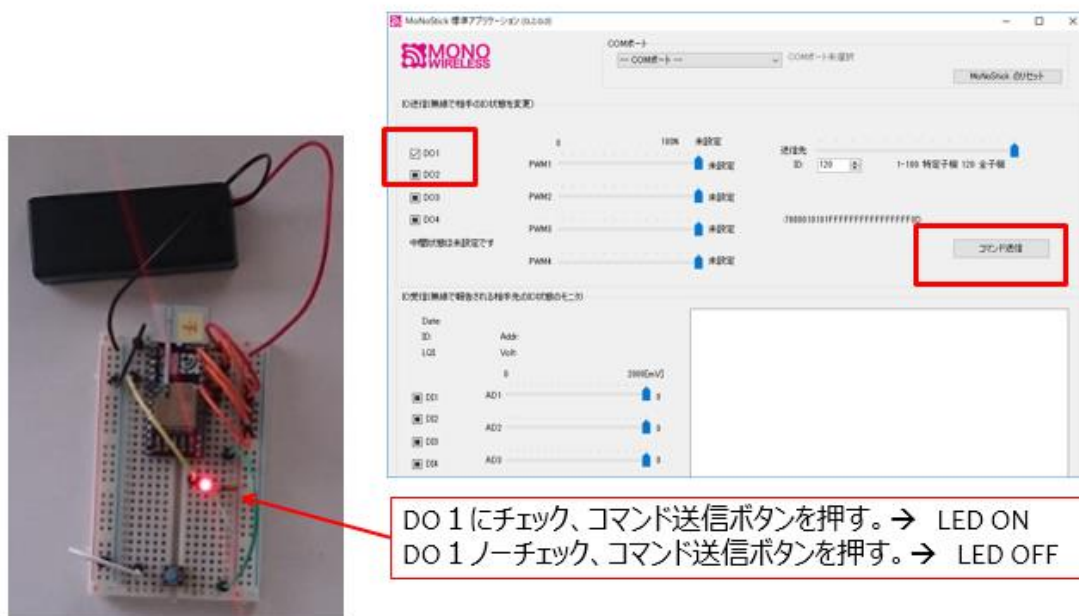


図 57

PC アプリで操作をしてみましょう。子機の電源 SW を ON にスライドしてください。そして、アプリケーションウインドウの左上にある【DO1】というチェックボックスにチェックを入れます。そして、右側中央部にある【コマンド送信】のボタンをクリックします。すると子機の LED が点灯します。次に DO1 のチェックをはずしてコマンド送信ボタンをクリックすると、子機の LED が消灯します。この時に送信された電文（16 進数文字列）がコマンド送信ボタンの左上に表示されているので、どの部分が DO1 に対応するのかが分かります。無線マイコンモジュールの制御用通信ソフトを開発するときにはたいへん役立つと思います。

◇子機のSWをON/OFFする



◇シリアルモニタに変化が見える

図 58

次に、子機側の SW を ON/OFF してみます。Windows アプリケーションのウィンドウ右下にシリアルモニターという白い窓があり、そこに刻々と子機から受信した電文が表示されていきます。子機の SW の状態が変化 (ON/OFF) するとこの電文に変化が現れます。この変化の様子を切り出したものが次の図です。(再掲) これは第 3 回で詳しく説明したのですが、子機の SW が変化すると、その時に電文中のデジタル入力 1 のビットも対応して変化します。この変化する 16 進数の下の桁 4bit で子機のデジタル入力 1~4 を通知しています。なおこの部分は子機の SW 状態に変化があったときだけ通知されるようになっています。これは無線マイコンモジュールに書き込まれている標準アプリケーションの仕様です。

デジタル入力の変化

◇デジタル入力を表すビットが変化

```
:788115017B810E2685000011000B6B208000FFFFFFFFF1A <---起動↓
:788115017B810E2685000053000B6D208000FFFFFFFFFD6↓
:7881150184810E268500008B000B71208000FFFFFFFFF91↓
:7881150187810E26850000D7000B70208000FFFFFFFFF43↓
:7881150187810E26850000F9000B702001FFFFFFFFF9F <---SW ON↓
:7881150184810E2685000131000B6F208101FFFFFFFFFEA↓
:7881150184810E2685000167000B6F208101FFFFFFFFFB4↓
:7881150184810E268500019F000B70208101FFFFFFFFF7B↓
:788115017E810E26850001BB000B6F20001FFFFFFFFFE7 <---SW OFF↓
:7881150187810E2685000207000B6F208001FFFFFFFFF11↓
:7881150184810E268500023D000B6F218001FFFFFFFFFDD↓
:7881150187810E2685000273000B70218001FFFFFFFFFA3↓
```

0	0	0	0	DI4	DI3	DI2	DI1
---	---	---	---	-----	-----	-----	-----

デジタル入力の値 レイアウト

子機のSWに対応する

図 59

子機の SW の ON/OFF 状態でシリアルモニターの受信電文の内容が変化しますが、それと同時にウインドウ左下の DI1 というチェックボックスにも変化が現れます。子機の SW を ON すると DI1 にチェックが入り、SW を OFF するとチェックが消えることが確認できると思います。

同時にD I 1の状態が変化！



子機のSWを押すと、その状態がD I 1に現れる

図 60

現在、アプリケーションは Windows 用のみが公開されているのですが、これらアプリは VS (Visual Studio)・VB (Visual Basic) で開発されていて、そのプロジェクトソースコードが提供されています。ここで実習に使用したアプリケーション以外にも、温度センサーの情報を表示する簡易ビューアなどがソースコードとともに公開されています。TWE-Lite に使用するのであれば、商利用も可能ということですので、皆さんもアプリケーション開発をしてみたいはいかがでしょうか。

アプリはソース公開

- ◇VS・VBで開発されている
- ◇プロジェクトファイルが提供されている
- ◇TWE-Lite用であれば商使用可能



温度センサーの情報を表示する簡易ビューア

図 61

第5回 PWM

PWM 制御という言葉をお聞きになったことがありますか。これは、Pulse Width Modulation の頭文字をとった言葉で、パルス幅変調制御というものです。次の図を見てください。

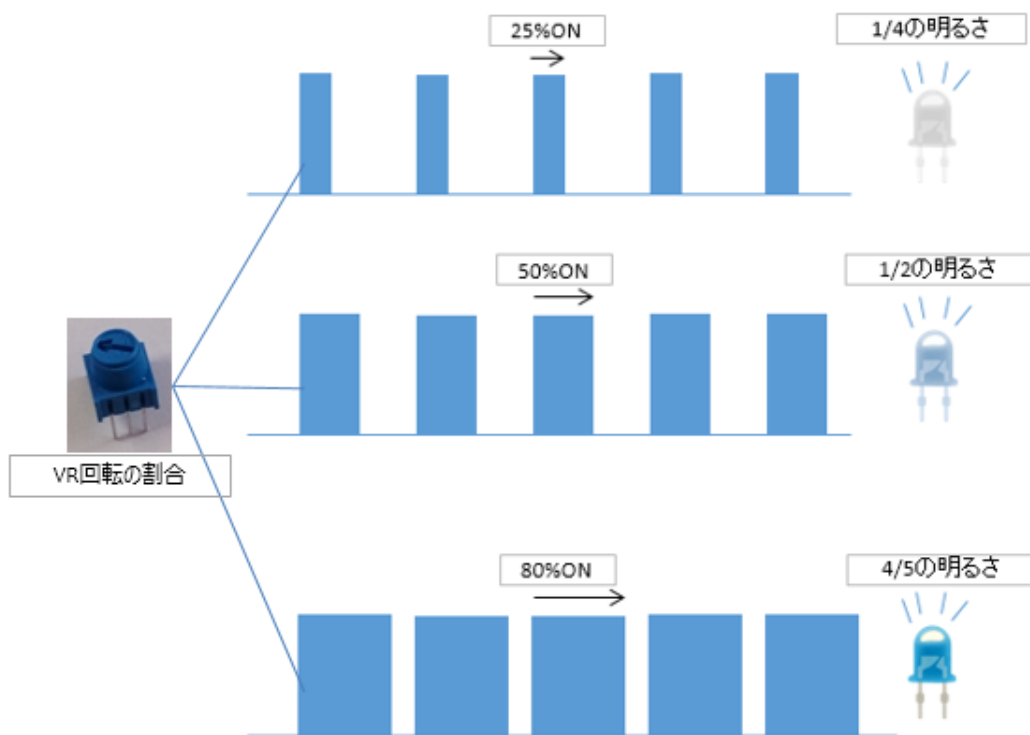


図 62

LED を点滅する信号があります。OFF では LED の明るさは 0% で、ON した時 100% の明るさだとします。この ON/OFF 信号を一定周期で繰り返すパルスとして発生させます。そして、ON の割合が例えば周期の 25% の時は LED が 1/4 の明るさで光ります。次にこの ON の幅を 50% にしてあげると LED は半分の明るさで光ります・・・と云うように、ON の幅（パルス幅）をコントロールすることで、ある対象の動作を制御す

る仕組みが PWM 制御（パルス幅変調制御）と言われるものです。ここでは例として LED を取り上げましたが、モーターの回転数や回転角、ヒーターの温度などを対象に、身近なところで利用されている制御方式です。今回は、無線マイコンモジュールの子機に VR（ボリューム：可変抵抗器）を取り付けて、VR の抵抗に対応する PWM 制御信号で、親機の LED の明るさを変化させてみましょう。今回も配線するだけでプログラミングレスです。

<<VRで明るさ変化>>

・・・配線するだけです・・・

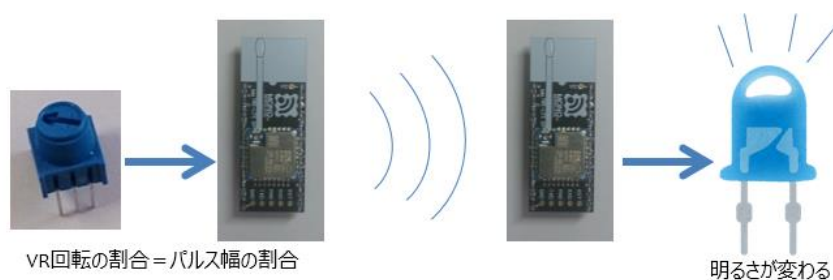


図 63

上の図左側が子機です。VR の両端に電圧を加えて VR を回転させると、回転の割合に応じた電圧が VR から取り出せます。この可変電圧を無線マイコンモジュールのアナログ入力ピンに接続すると、電圧が変化するとき、子機は無線通信でその状態を右側の親機に伝えます。親機はこの無線で通知されたアナログ入力ピンに加えられた電圧に対応する PWM 信号を PWM 出力ピンに出力しますので、ここに LED を接続しておくと、子機の VR の回転の度合いに応じて親機の LED の明るさが変化するという具合です。

◇システムの全体構成

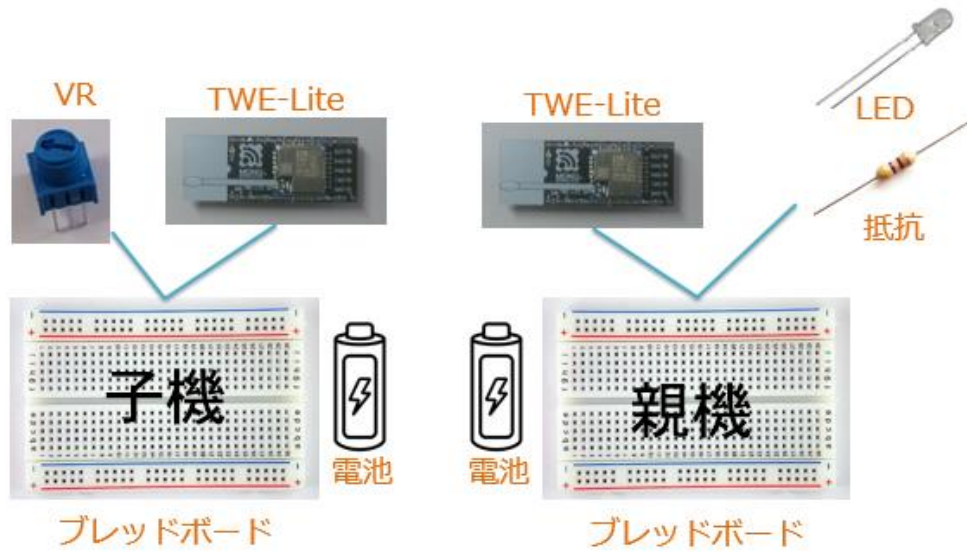


図 64

上の図がシステムの全体構成です。親機には明るさを変化させる対象として LED を使います。今回初めて VR (ボリューム：可変抵抗器) を使いますが、その構造・役割を見ておきましょう。(次の図)

VR (可変抵抗器) の役割

電圧を分ける → 分圧

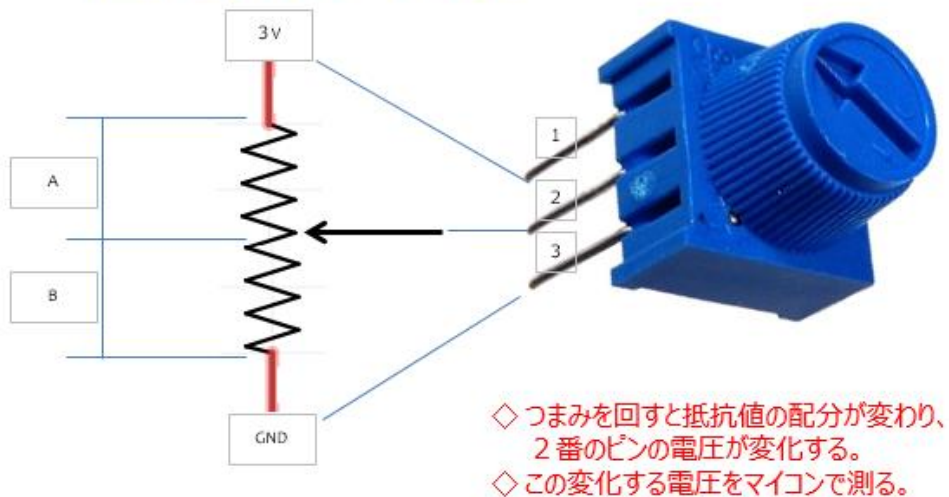


図 65

VRには3本のピンが出ています。このピンに図のように1~3の番号を付けます。1番ピンに3V、3番ピンにGNDを接続すると、2番ピンからは、VRの回転の割合に応じて内部の抵抗がAとBの部分に分割されて、それに対応する電圧が取り出せます。VRのつまみを回すと、図の矢印の部分が上下に移動します。一番上に移動すると2番ピンは3Vになります。反対に一番下に移動すると2番ピンは0Vになります。このように内部の抵抗値の配分で両端にかかっている電圧を配分するので、このことを抵抗値による分圧と云います。この2番ピンをマイコンのアナログ入力ピンに接続して電圧を測ります。その値がPWM制御の元になるパルス幅へと変換されるわけです。この電圧計測からパルス幅への変換は、この無線マイコンモジュールに書き込まれている標準アプリケーションが行ってくれるので、プログラミングは必要がありません。

VRの電圧を測るADCの仕様

◇アナログ入力はAD変換器 (ADC)

ADCの仕様

TWELITEには10bit, 4ch のADCが搭載されています。(ADC2はVREF入力と共用です。ADC3,4はDIOと共用になっています)

- ADCは、0-2.4Vレンジです。(0-VCCの相対スケールではありません)
- ADCの内部Vrefは約1.2Vで、外部への出力はありません。また温度特性等の情報は公開されておりません。
- 精度を要求される場合は外部のADCの利用を推奨します。
- ADCは、2Mhz, 1Mhz, 500khz, 250khz (500khz推奨)でサンプリング回路のサンプリングクロックを設定可能です。実際にこのクロックでサンプリングできるわけではなく、内部回路の周波数です。一定周期でサンプルしたい場合は、周期タイマー (TickTimer や TIMER0/1/2) 割り込みを起点にサンプル値の取得をソフトウェアで行います。ただし高周期ではタイマーの時間軸のブレ (ジッター) を考慮する必要があります。
- 1サンプル取得に (2, 4, 6, 8) x 3 + 14 サンプリングクロックが必要です。
- 500khzで2クロックの場合、2x3+14 = 20 サンプリングクロック = 40 usec となります。

※メーカー資料抜粋

図 66

上の資料は、メーカーが公開しているアナログ入力ピンにつながっている A/D 変換器 (ADC) の仕様です。これを見ると、【ADC は 0-2.4V】との記述があります。先の VR でつまみを回して矢印が一番上に移動すると 2 番ピンには 3V が出力されて、アナログ入力ピンに 3V がかかってしまいます。これは、上の仕様を超えてしまうので、少し工夫をしなければいけません。

VRの部分を少し工夫

- ◇VRに加える電圧を**2.4V以下**にします。
- ◇50kΩのVRに15kΩの抵抗を加えると、VRの最大出力電圧は、

$$\frac{50}{50+15} \times 3 \approx 2.3 \text{ [V]}$$

となります。

- ◇15kΩの抵抗をVRの電源側に接続します。

図 67

15kΩの抵抗をVRの電源(3V+)側に接続すれば、上の計算のようにVRを回し切ってしまうと最大の電圧が出力されても2.3VとなってADCの仕様の範囲内に収まりますので、VRは次の図のようにします。

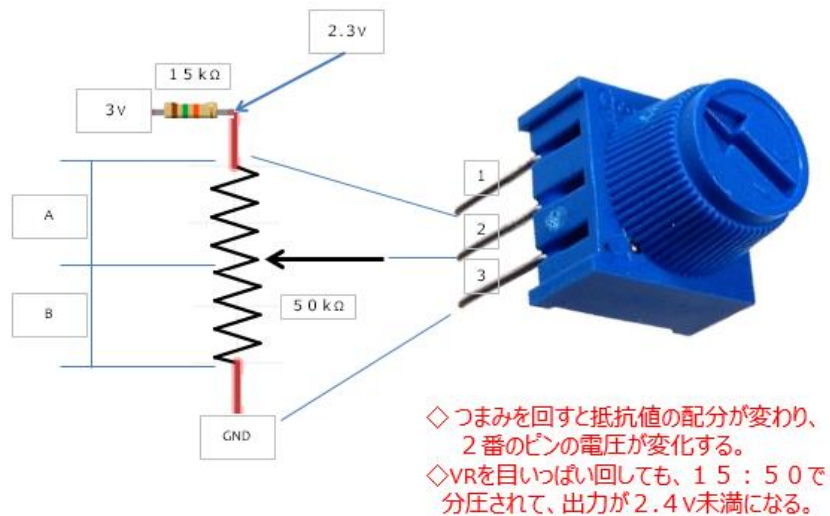


図 68

使用するパーツは次のようになります。

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. VR（50kΩ）×1個
6. 抵抗器（15kΩ）×1個

親機側：

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. LED×1個
6. 抵抗器（470Ω）×1個

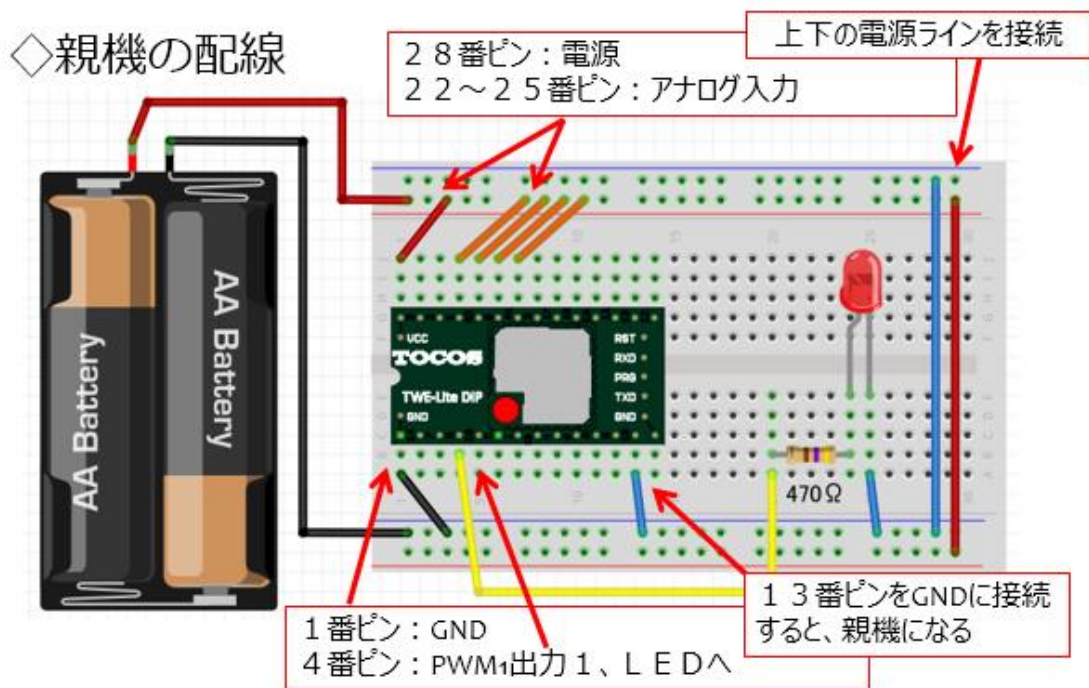


図 69

では親機の配線を上の図のように行ってください。親機には LED と抵抗を使用しますが、これまでの講座で使用したピンとは違う場所に LED を接続していることに注意してください。これまでデジタル出力 1 (DO1) に LED を接続しましたが、今回は 4 番ピンの PWM1 の信号に LED をつなぎます。LED の極性 (長い方の脚の向き) にも注意をしてください。反対向きに接続すると電流が流れず LED は光りません。親機ですから、13 番ピンを GND 接続するのも忘れないようにしてください。今回からブレッドボードの右端に上下の電源ラインを接続する赤・青のジャンパー線が加わっています。今回の実習では実際には使用されていませんが、このようにするとブレッドボードの上下どちらからでも電源 (+) と GND(-) が使えます。

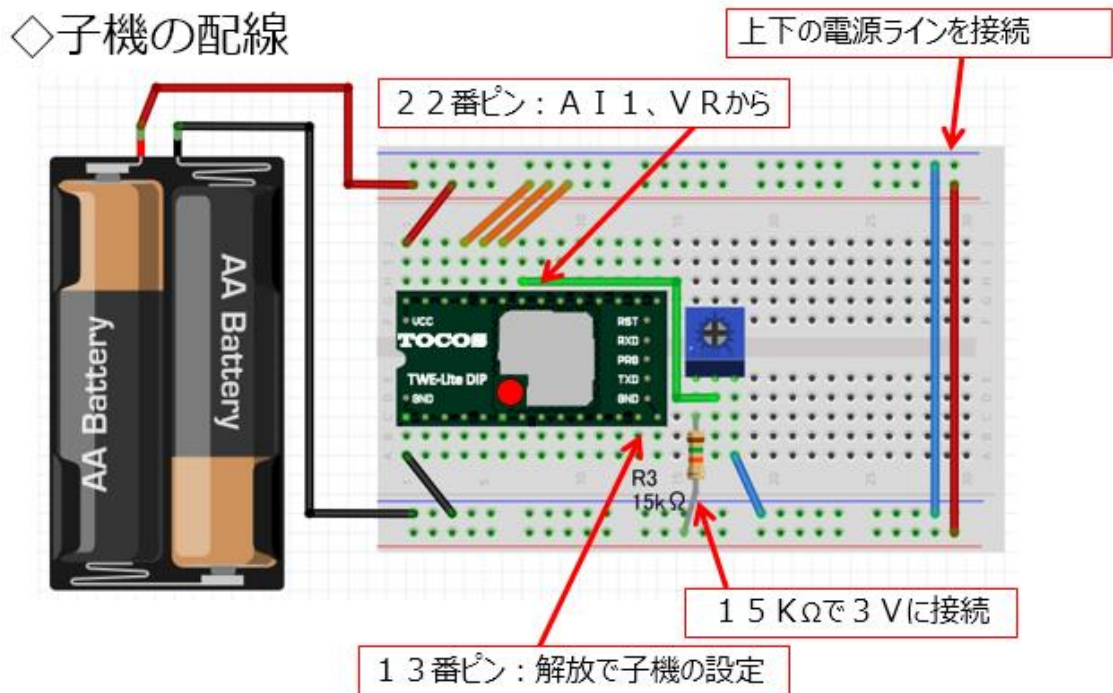


図 70

次は子機を配線しましょう。(上の図) 子機は既に説明した VR と抵抗器の配線がありますね。VR の 2 番ピンはマイコンの AI1 の信号である 22 番ピンに接続します。子機も右側に上下の電源ラインを接続するジャンパー線を追加してあります。

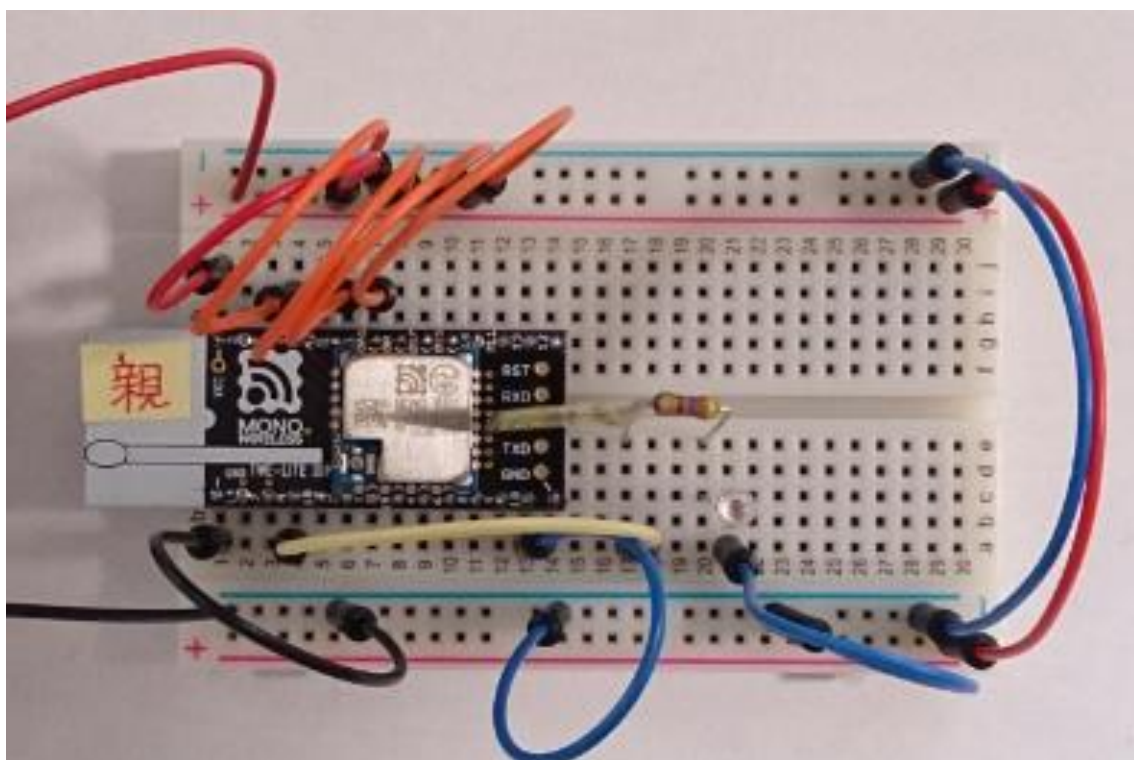


図 71

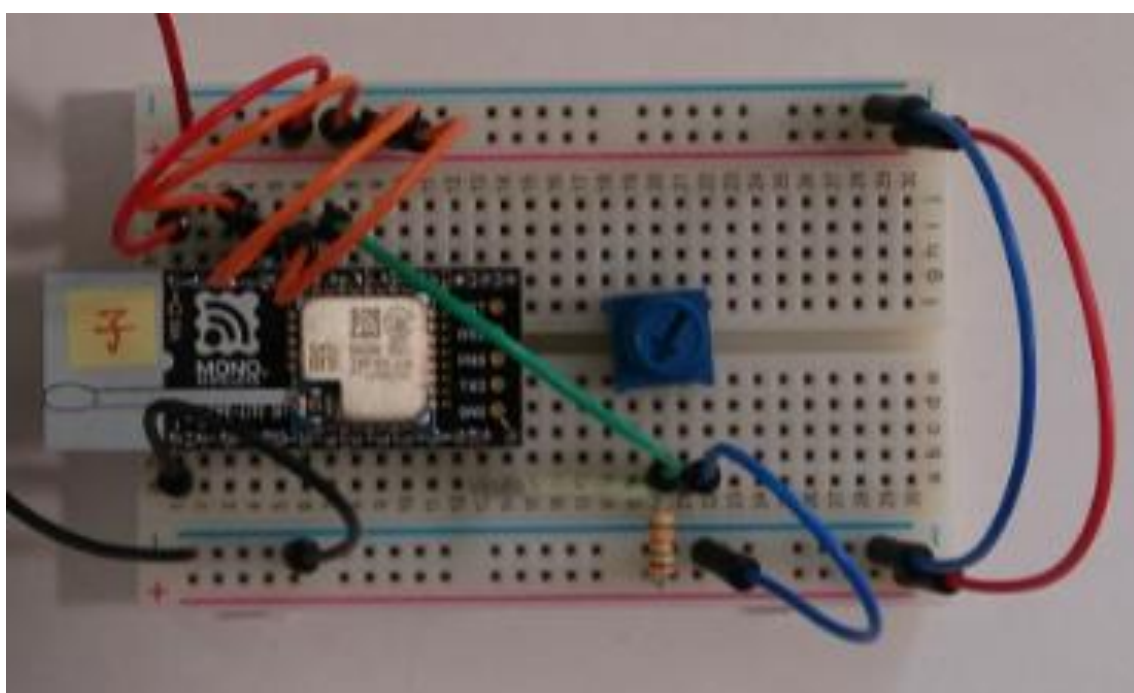


図 72

実際に配線したものが上の写真です。

もう配線にも慣れてきたと思いますが、油断大敵です。注意深く確認をしてください。子機の AI1 ピンはこれまで電源 (+) に配線されていましたが、今回はそれが VR からの配線になっています。

◇動作確認

さて、親機・子機の電源 SW を ON にスライドしてください。そして、子機の VR のつまみを少しずつ回してみましょ。親機の LED の明るさが変化しているのが確認できると思います。

次の写真は、パルスの幅をオシロスコープで観測して、その時の LED の明るさを撮ったものです。

パルス幅とLEDの明るさ

◇繰り返し出力するパルスの幅に比例して、LEDの明るさが変化します。

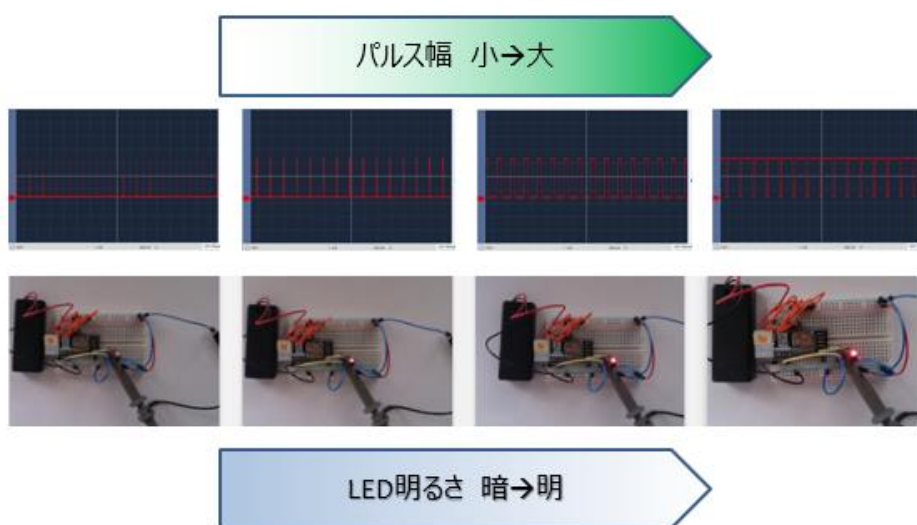


図 73

VR のつまみを回転することに対応して、パルス幅が変化していることが分かります。また、それに応じて LED の明るさも変化しています。これが PWM 制御と呼ばれている制御方式です。

第6回 双方向 PWM

今回は、第 5 回で行った PWM 制御を双方向で行っていきましょう。



図 74

第 5 回では、VR のつまみ位置で設定する電圧を子機側マイコンモジュールで読み込んで、それを親機に無線送信して、親機側の PWM 制御のパルス幅に反映させて LED の明るさを制御しましたが、今回は同時に、親機でも VR で設定される電圧を読み込み、それを子機に無線送信して、子機の PWM 制御パルス幅に変えて LED の明るさ制御を行います。双方向同時 PWM 制御です。

◇システムの全体構成

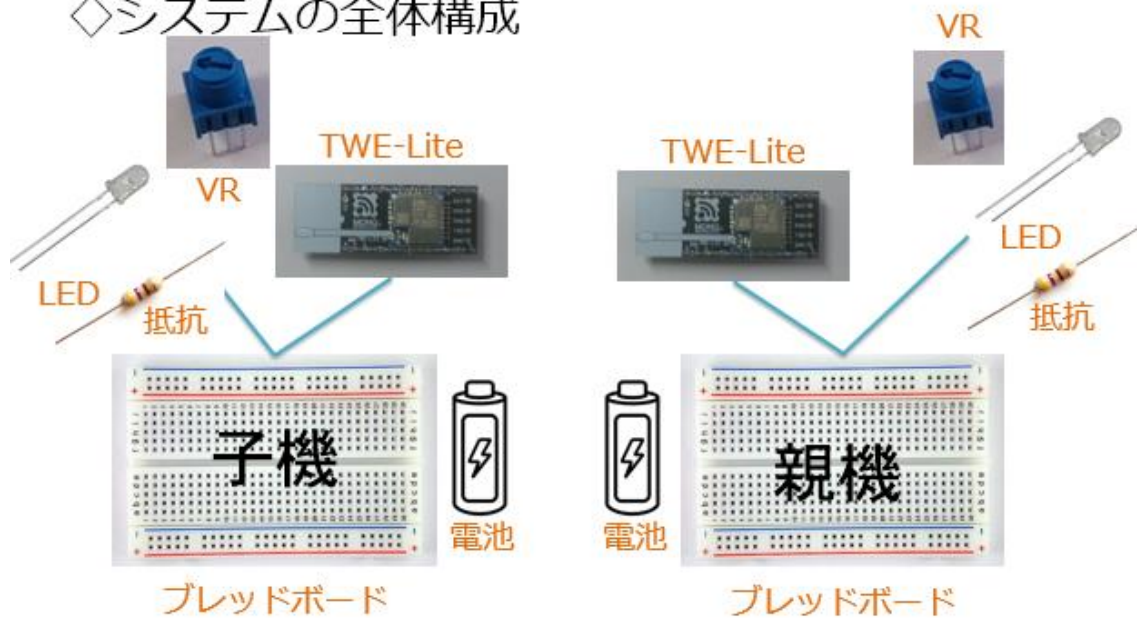


図 75

親機・子機ともに全く機器構成ですから、使用するパーツは各々2組です。

子機側＋親機側：

1. 無線マイコンモジュール×2台
2. ブレッドボード×2個
3. 電池（単4乾電池×2個＋SW付電池ケース×1個）×2セット
4. 配線用ジャンパー線
5. VR（50k Ω ）×2個
6. LED×2個
7. 抵抗器（470 Ω ）×2個（LED電流制限用）
8. 抵抗器（15 Ω ）×2個

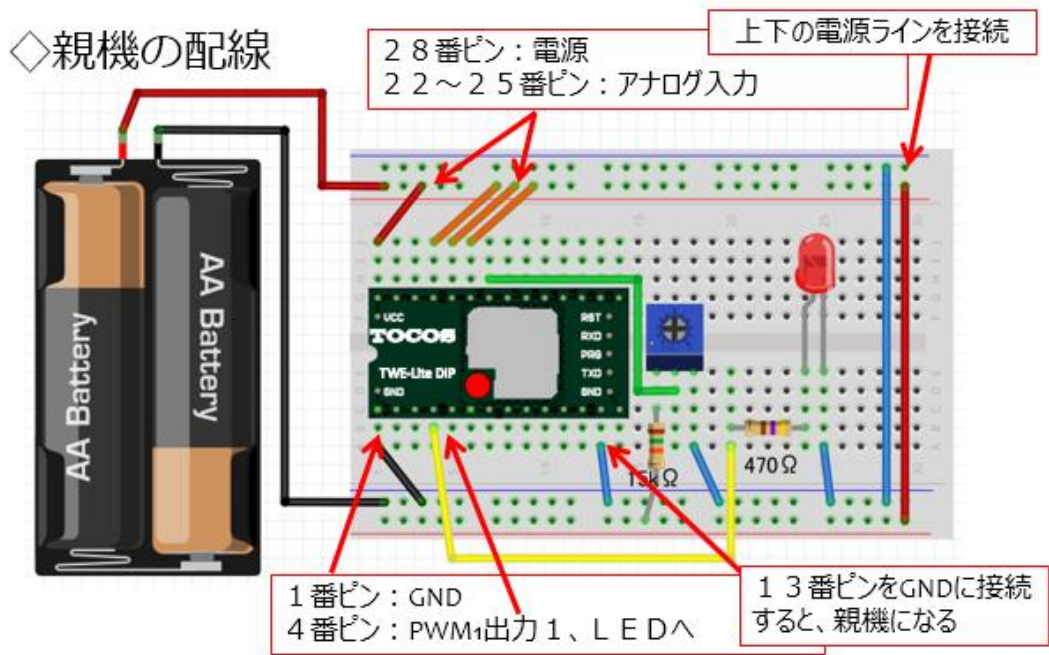


図 76

上の図に従って親機を配線します。子機も同様に次の図に従って配線します。

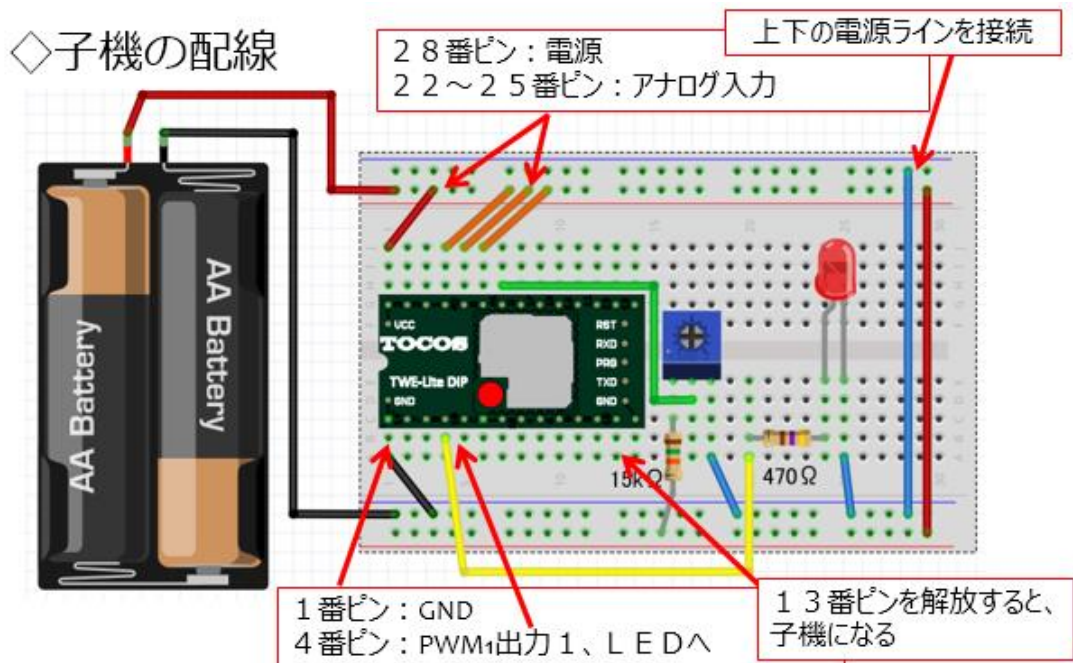


図 77

親機の 13 番ピンを GND に接続するジャンパー線を配線する以外は、親機・子機共に同じ配線ですので、確認しやすいですね。

◇実際に配線した様子 【親機】

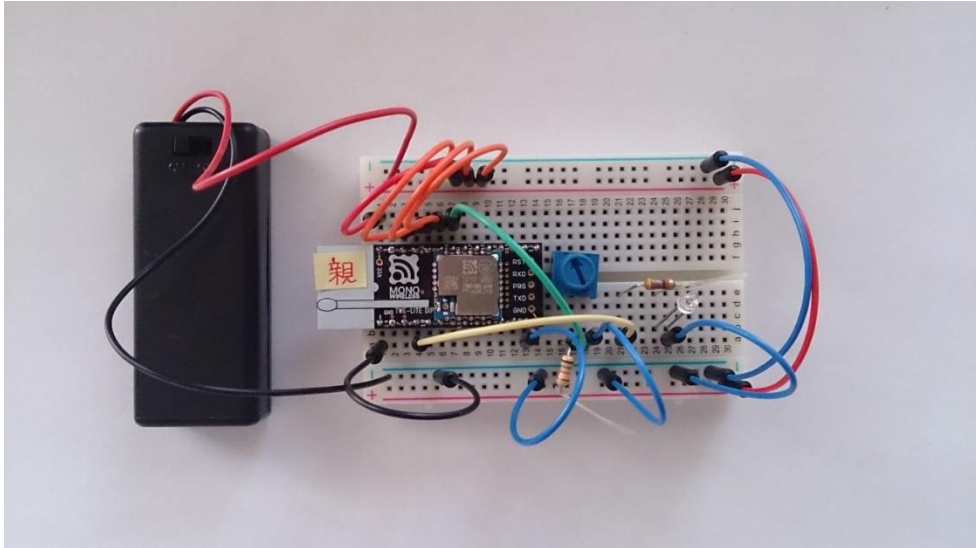


図 78

◇実際に配線した様子 【子機】

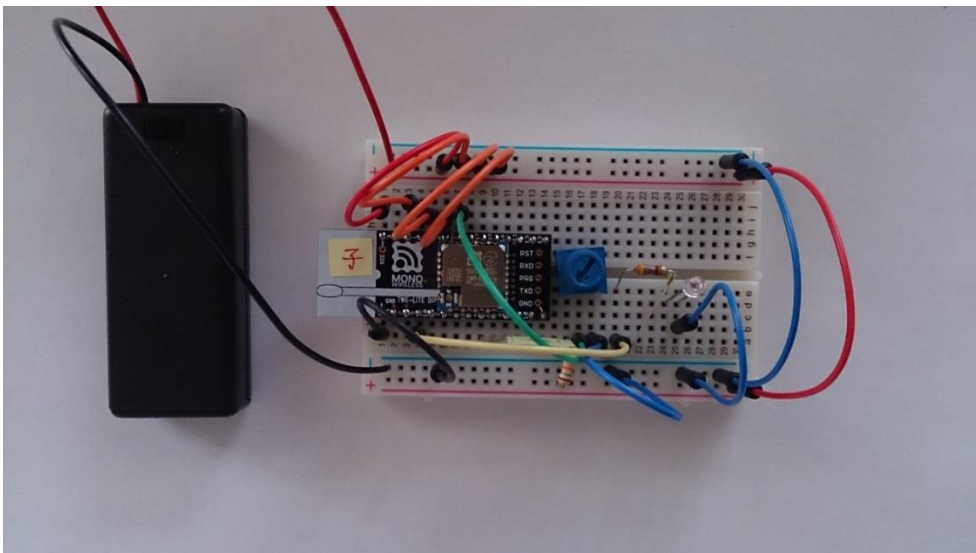


図 79

違いは、13 番ピンのモード設定信号のジャンパー線だけです。

◇動作確認

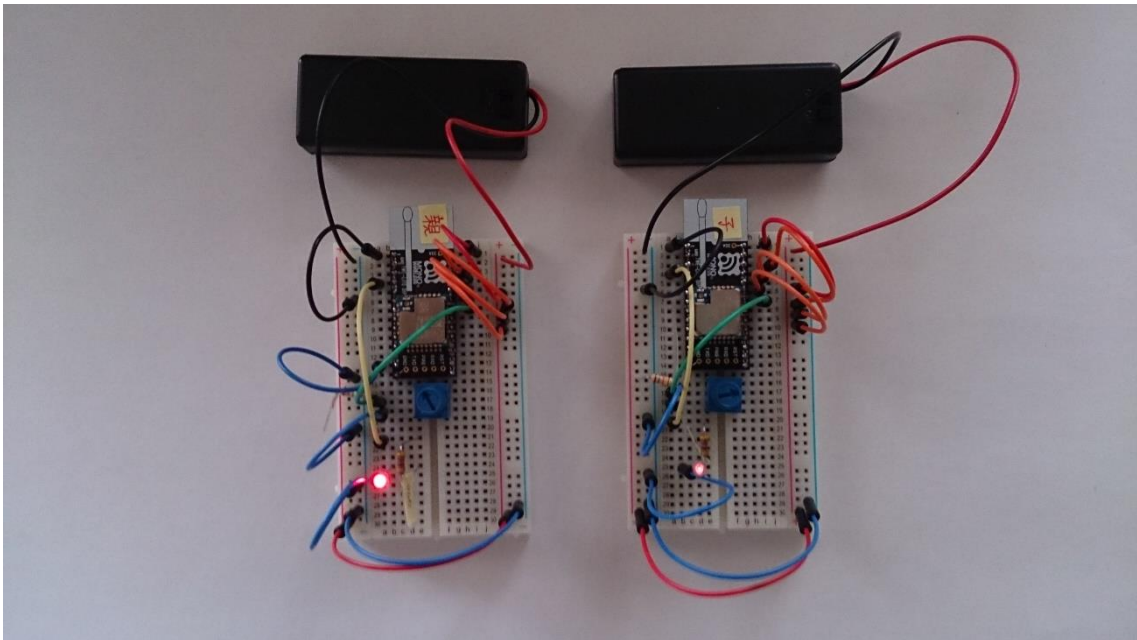


図 80

親機・子機共に電源 SW を ON にしてください。親機の VR を回転すると、その状況が子機側 LED に反映されますか。同じように子機の VR の回転位置が親機の LED の明るさとなって現れますか。

双方向 PWM 制御は、とても簡単に実現できました。

【注意】今回作成した子機は、次の回でもそのまま使用しますので、保存しておいてください。

第7回 スマートフォン連携 PWM

第5回、第6回で行った PWM 制御では、VR によって無線通信の相手側の LED の明るさをコントロールしたのですが、今回はその VR の代わりにスマートフォンを使うという実習です。

◇ USB I/Fでスマートフォンと接続



図 81

上の図の様に第3回のスマートフォン連携での考え方をそのまま使います。ただし、今回はスマートフォンから子機への送信のみです。スマートフォンの画面に配置されているスライダーを VR と見立てて、子機の LED を PWM 制御します。

◇システムの全体構成

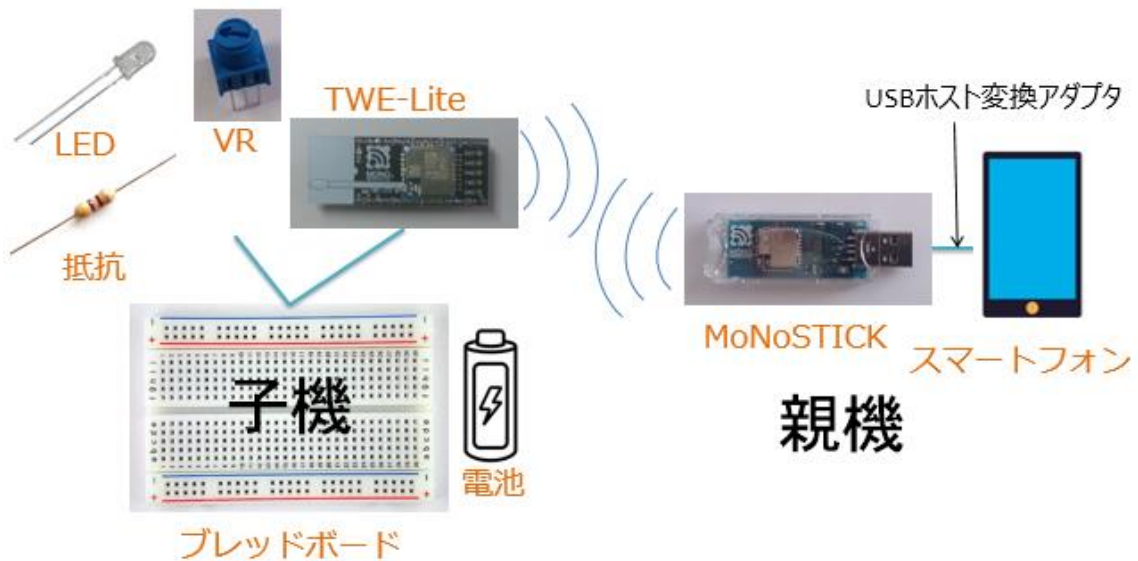


図 82

第 6 回の子機をそのまま使用しますが、子機に配置されている VR は、使いません。

◇USB ホスト変換アダプター



図 83

MoNoSTICK モジュールをスマートフォンに接続するときは、上の写真の USB ホスト変換アダプターを使用します。

親機・子機ともに全く機器構成ですから、使用するパーツは各々2組です。

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. VR（50k Ω ）×1個（実験には使っていません）
6. LED×1個
7. 抵抗器（470 Ω ）×1個（LED電流制限用）
8. 抵抗器（15k Ω ）×1個（実験には使っていません。）

親機側：

1. MoNoSTICK×1台
2. USBホスト変換アダプター×1本
3. スマホアプリ TWE Control×1セット

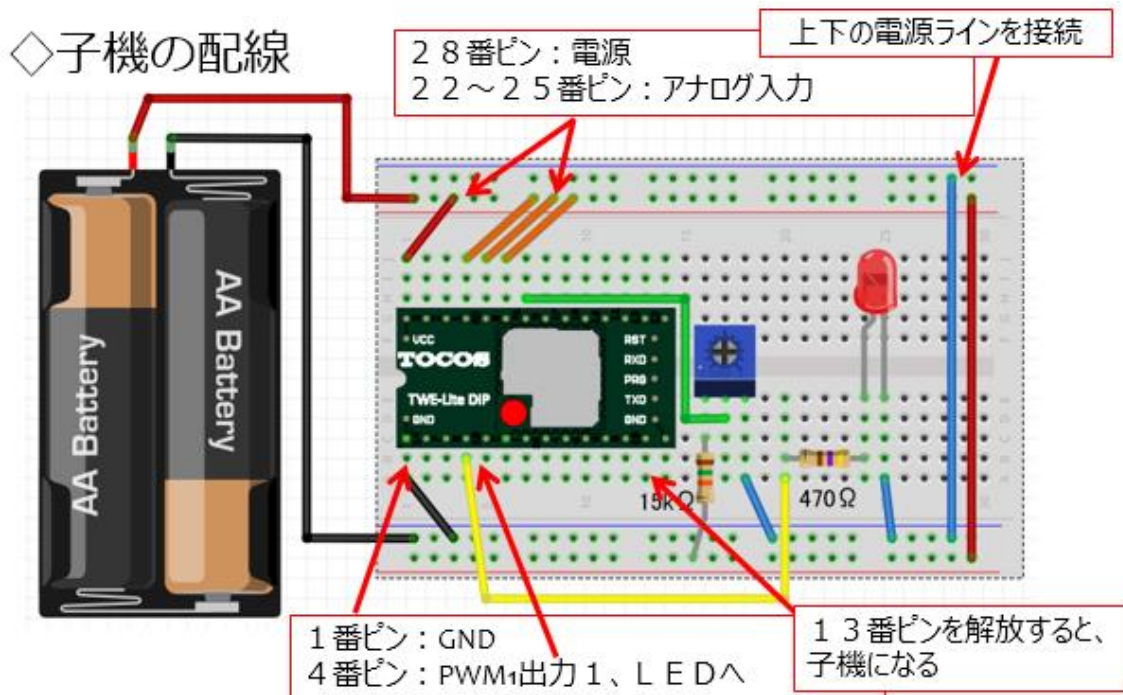


図 84

前回の子機をそのまま使いますので、保管してある方は配線の緩みなどが無いことを良く確認しておいてください。新たに作る方は上の図を参考にして配線を行ってください。実際に配線すると次の写真のようになります。VR と付随する抵抗（ $15\text{k}\Omega$ ）は配線されていますが未使用です。

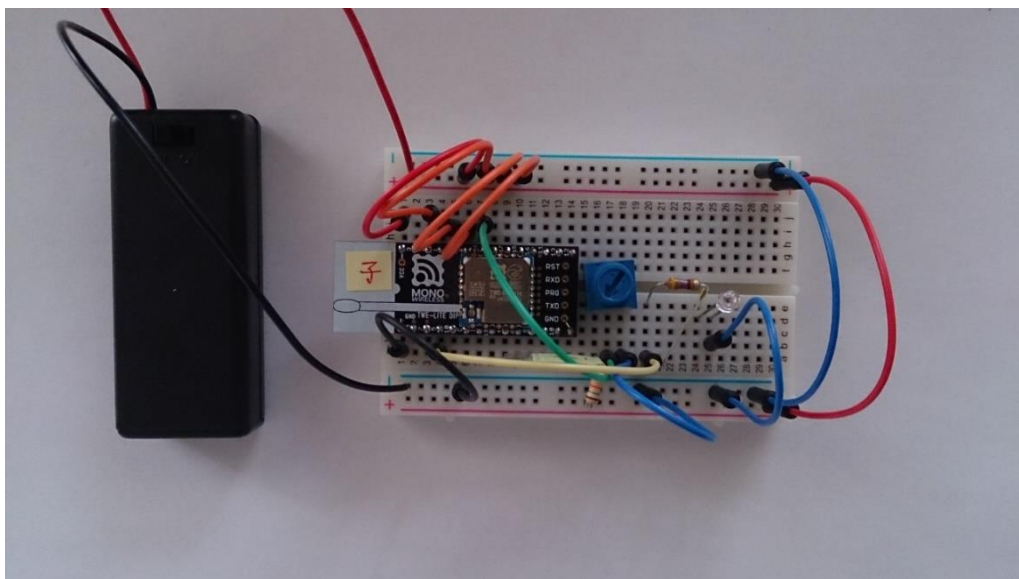


図 85

◇ MoNoSTICK モジュールの接続と子機、アプリの準備

USB ホスト変換アダプターを使い、下の写真のように、MoNoSTICK モジュールをスマートフォンに接続してください。

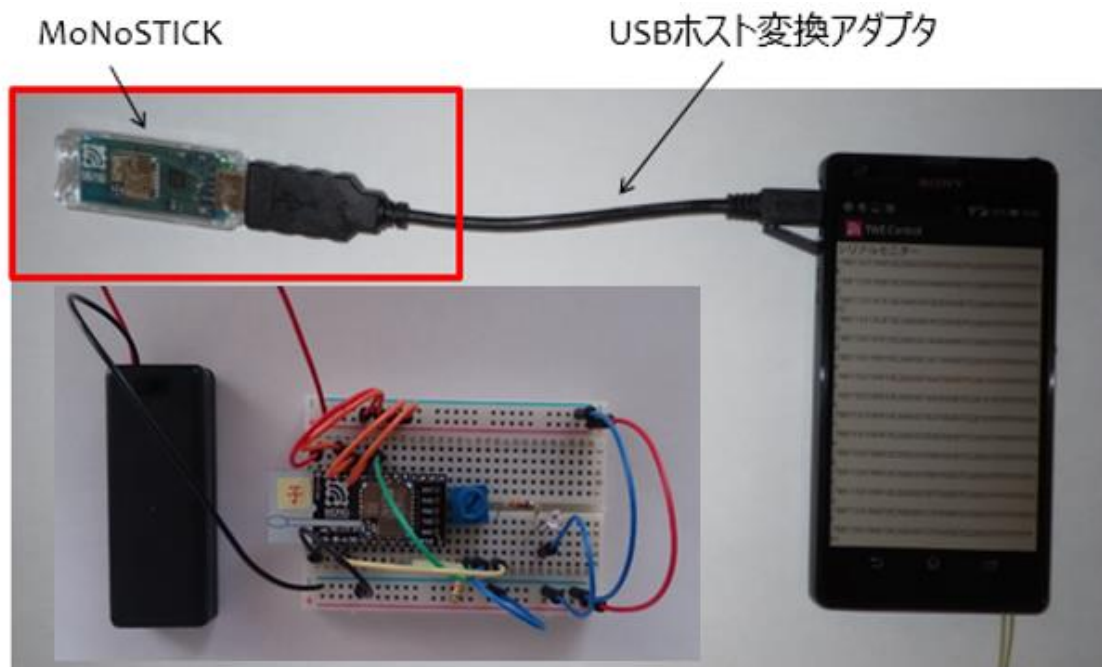


図 86

MoNoSTICK モジュールが接続されるとモジュール内の電源 LED が点灯します。次に、子機の電池ボックスにある SW を ON にスライドします。

第 3 回でインストールしたスマートフォンアプリ【TWE Control】を今回も使用します。まだインストールされていない方は第 3 回を参照してください。

◇スマートフォンアプリの起動

子機の電源を入れて数秒後 Android フォンにインストールしたアプリ TWE Control を起動します。画面中央の【接続】をタップします。

(次の図) 接続できると【未接続】から【接続中】に表示が変わります。



図 87

◇アプリ起動 接続、遠隔操作を選択



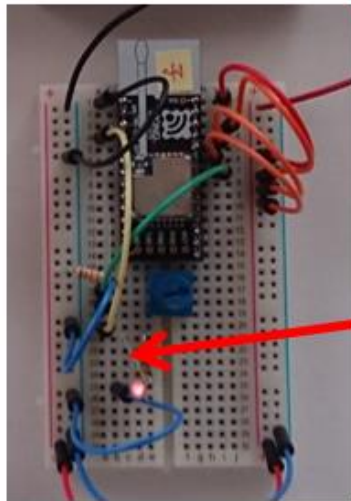
図 88

メニュー画面の下の部分のボタンが有効になりますので、【遠隔操作 (リモートコントロール)】をタップします。(上の図) すると、画面が右のように変わります。この画面で操作をおこないます。

◇動作確認

スマートフォンで操作

◇PWM1をスライド



LED明るさが
変化



図 89

上の図で示すように、スマートフォン画面の PWM1 のスライダーをスライドしてみましょう。スライダー位置に対応して子機の LED の明るさが変化するでしょう。

この PWM 出力信号を取り出して、様々な機器を制御することが容易にできます。皆さんなら、どのような使い方を考えますか。

第8回 PC 連携 PWM

第 7 回で行ったスマートフォンに代わって、PC で PWM 制御の操作を行ってみましょう。少しの時間があれば試すことができます。

◇ USB I/FでPCと接続



図 90

前回と同様の事を PC と行います。PC へのアプリケーションは既に第 4 回でインストール済みですが、まだ準備が済んでいない方は第 4 回を参照していただき、下記メーカーWEB ページからダウンロードしてインストールしておきましょう。

(<https://mono-wireless.com/jp/products/TWE-APPS/index.html>)

◇システムの全体構成

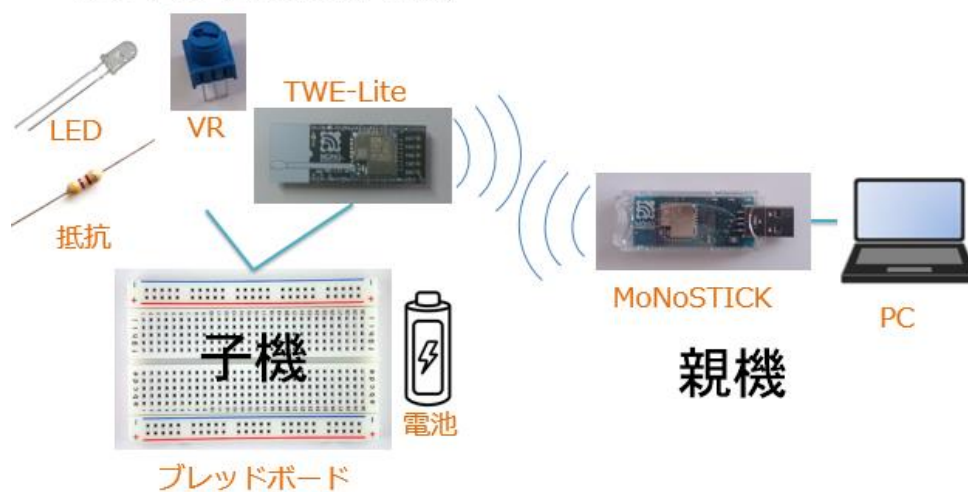


図 91

上の図はシステムの全体構成です。第 7 回とほぼ同じです。
MoNoSTICK モジュールは USB コネクタでそのまま PC に接続できます。
使用するパーツなど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. VR（50k Ω ）×1 個
6. LED×1 個
7. 抵抗器（470 Ω ）×1 個（LED 電流制限用）
8. 抵抗器（15k Ω ）×1 個

親機側：

1. PC×1 台（Windows10）
2. Windows 専用アプリ×1 セット

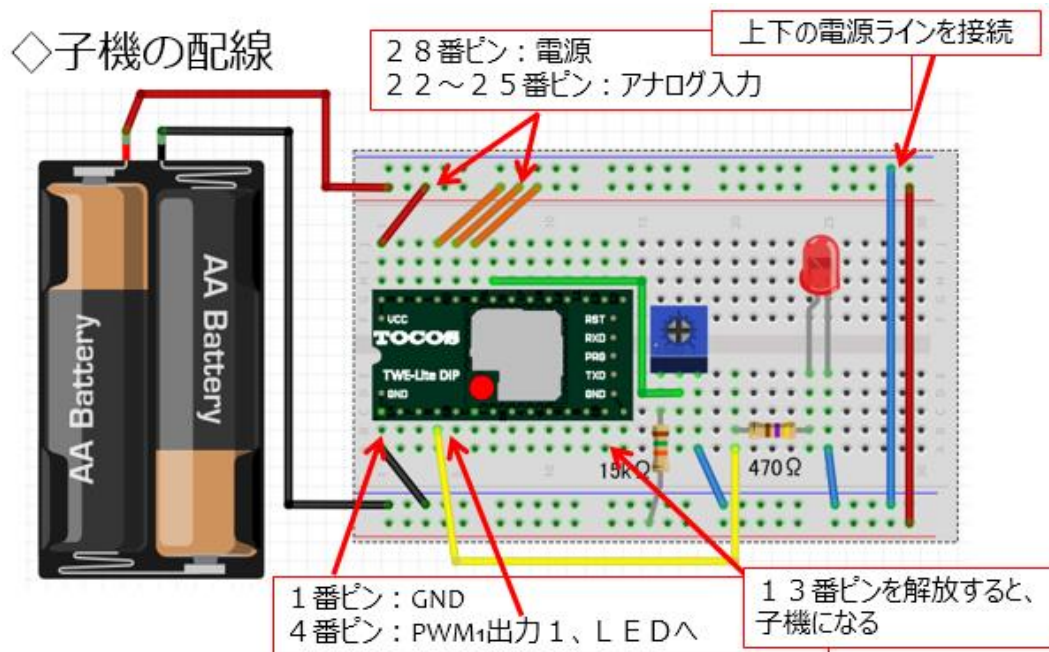


図 92

第 7 回の子機をそのまま使いますので、保管してある方は配線の緩みなどが無いことを良く確認しておいてください。実際の子機は下の写真のようになっています。確認したら子機の電源を入れておいてください。

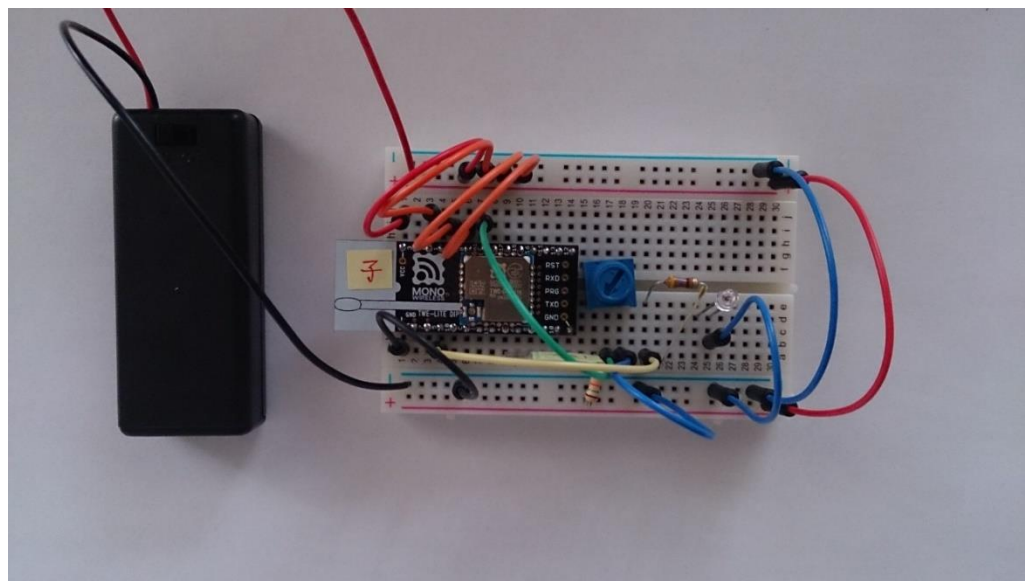


図 93

◇ MoNoSTICK を PC にセット



図 94

MoNoSTICK を PC の USB コネクタに差し込んで、OS に認識させます。デバイスマネージャで仮想 COM ポート番号を確認してください。第 4 回で使用したものと同一 PC 環境であれば COM ポート番号はもう分かっているはずです。

◇ Windows アプリケーションの準備が整っていれば、起動します。

◇ アプリ起動



図 95

アプリケーションのウィンドウが開いた際に中央上部に COM ポート番号が選択表示されていない場合は、デバイスドライバで確認した COM ポート番号をリストから選択してください。(子機の電源は、既に入っていると思いますが、ここで ON にしてもかまいません。)

◇ 動作確認

起動したアプリケーションの中央やや左の上段にある PWM1 と表示されているスライダーを中央にスライドして右側中程にあるコマンド送信というボタンをクリックします。すると、LED が点灯します。次に、スライダーをさらに右に移動して再びコマンド送信ボタンをクリックすると、LED は先ほどより明るくなります。反対に最初の状態より

左側にスライドさせてコマンド送信ボタンをクリックすると LED は暗くなります。Windows アプリではスライダの位置を決めてからコマンド送信ボタンをクリックすることで、通信電文が送られるような仕様になっているので、スマートフォンアプリと少し使い勝手が違っていますが、同じように PWM 制御が行えます。次に、子機の VR を回転してみます。



図 96

子機の VR を右に回転すると、Windows アプリの下の方にある AD1 というスライダーが右に移動し、その右にある数字が大きくなります。また、VR を左に回転するとスライダーが左の方に移動しながら、数字が小さくなっていきます。先ほど LED の明るさを PC 側からコントロールしたのに対して、子機の VR で作られる電圧によって、親機側 (PC 側) のアプリ上のスライダー位置をコントロールしているということになります。このシステムでは親機と子機の間で双方向無線通信を行い、それぞれの状態を通知しあうことで、互いをコントロールしているのです。

第9回 温度センサー

今回は、温度センサーを使用します。計測値を離れたところに無線通信で送ります。主な内容は次の通りです。

- ◇温度を測定、無線で送り、PCで処理.
- ◇PC側の処理プログラムを作る.
- ◇処理系 → Python 使用.
- ◇PC側 開発環境の説明あり.

図 97

温度は電圧として温度センサーから得られますが、それはそのまま PC に送信して、PC 側の処理プログラムを作り、電圧から温度に変換計算を行って温度を求めます。PC 側プログラムには Python という言語を使うことにします。Python は、最近ビッグデータの処理システムや AI などに使われてきています。豊富なライブラリの公開などにより、あらゆるシステム開発を少ない時間で行うことができ、コスト削減につながるの産業分野でも利用されています。ここで Python を経験しておくことは、皆さんのスキルアップに有意に働くでしょう。

◇システム概要

◇温度を無線通信 + PC表示



図 98

今回の新たなデバイスは温度センサーです。使用している無線マイコンモジュールは、温度センサーを取り扱うための機能が標準アプリケーションに含まれているので、実習にはもってこいのテーマです。計測値から温度を計算して PC で表示します。

◇システムの全体構成

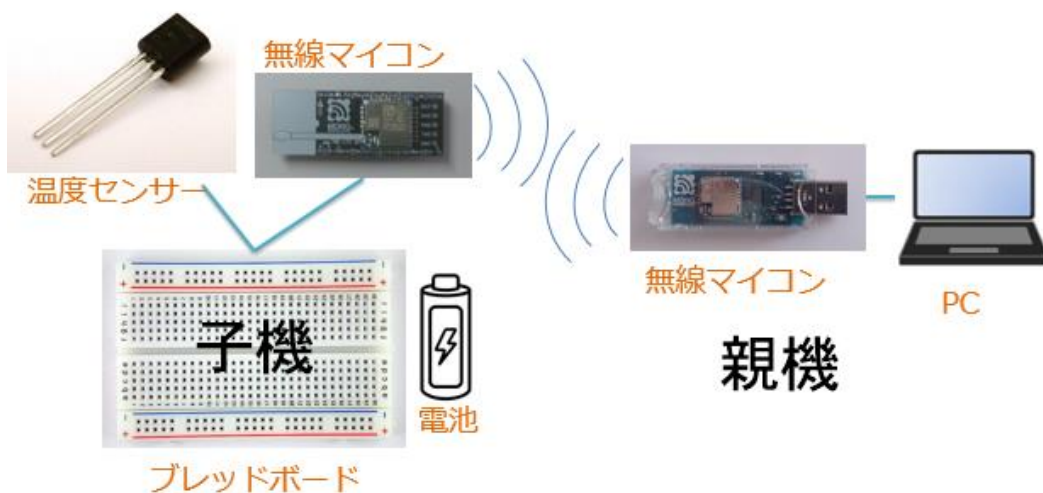


図 99

システムの全体構成は、上の図の通りです。前回の実習の子機の VR の代わりに温度センサーが使われていると思えばよいでしょう。

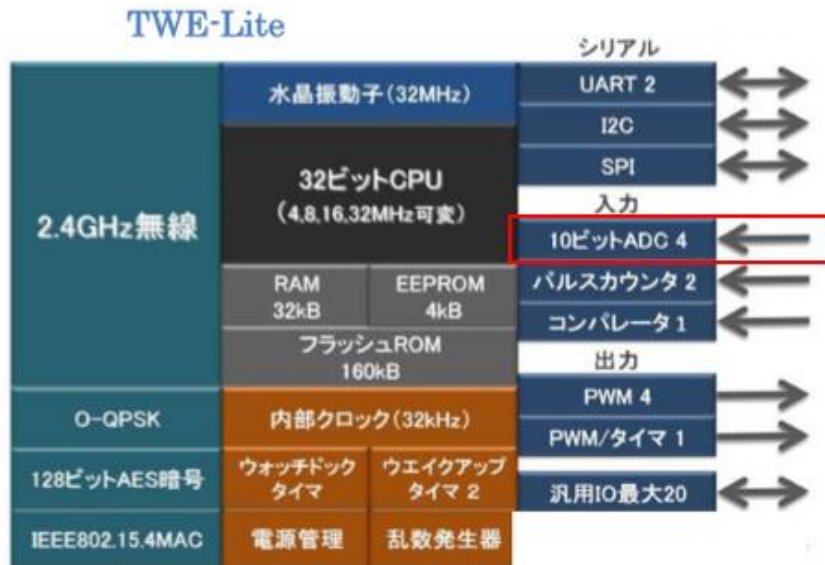
◇温度センサーはアナログ電圧出力

- ◇【基本】センサーは測定結果を電圧で出力.
- ◇その電圧を測るには A/D変換 が必要.
- ◇デジタル値になればマイコンで取り扱える.
- ◇そのために、ADC (AD変換器) を少し…

図 100

一般にセンサーは計測結果を電圧で出力します。電圧はアナログ値ですからそのままではデジタルのコンピュータでは計算などができません。そのために電圧のデジタル値を得るために A/D 変換 (アナログ→デジタル変換) を行います。デジタル値になって取り込めればマイコンでも容易にいろいろな計算に使えるようになります。そのために、使用している無線マイコンモジュールの A/D 変換器について少し知っておきましょう。

無線マイコンモジュールブロック図 ADC



※メーカー資料抜粋

図 101

図は、無線マイコンモジュールブロック図です。この中に【10ビットADC】と書かれているのが A/D 変換器 (A/D Converter) です。4 という数字は 4ch (4 系統) あるという意味です。

電圧を測るADCの仕様

◇アナログ入力はAD変換器（ADC）

ADCの仕様

TWELITEには10bit, 4ch のADCが搭載されています。(ADC2 は VREF 入力と共用です。ADC3,4 は DIO と共用になっています)

- ADCは、0-2.4V レンジです。(0-VCC の相対スケールではありません)
- ADCの内部 Vref は約 1.2V で、外部への出力はありません。また温度特性等の情報は公開されておられません。
- 精度を要求される場合は外部のADCの利用を推奨します。
- ADC は、2Mhz, 1Mhz, 500khz, 250khz (500khz 推奨) でサンプリング回路のサンプリングクロックを設定可能です。実際にこのクロックでサンプリングできるわけではなく、内部回路の周波数です。一定周期でサンプルしたい場合は、周期タイマー (TickTimer や TIMER0/1/2) 割り込みを起点にサンプル値の取得をソフトウェアで行います。ただし高周期ではタイマーの時間軸のブレ (ジッター) を考慮する必要があります。
- 1サンプル取得に (2, 4, 6, 8) x 3 + 14 サンプリングクロックが必要です。
- 500khz で 2 クロックの場合、2x3+14 = 20 サンプリングクロック = 40 usec となります。

※メーカー資料抜粋

図 102

ADC の仕様が上の資料（再掲）に書かれています。【ADC の入力レンジは 0-2.4V】と書かれています。これは、その範囲内の電圧を測れるということです。前に示したブロック図の内容と合わせて ADC の内容を整理すると次のようになります。

- ◇レンジ：0～2.4V → 10bit (0～1023)
- ◇10bitのAD変換値を1byteにまとめている。
- ◇ADC補正值も付加している。
 $2\text{bit} \times 4\text{ch} = 8\text{bit}$

※標準アプリで通信を行いやすくするためか。
※2bit 補正は、AD 値を 1/4 しているため。

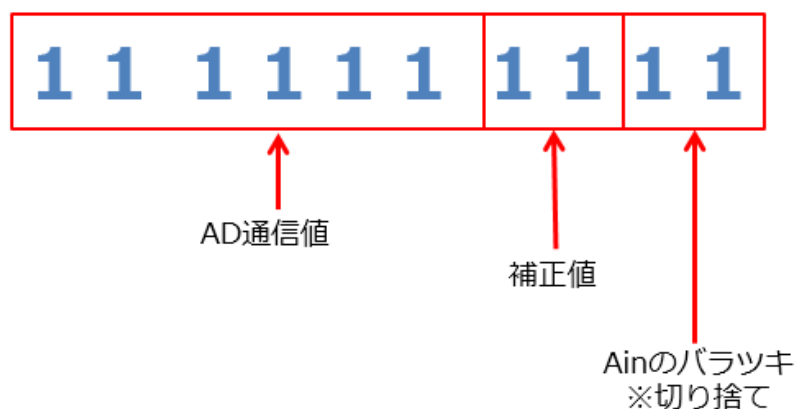
図 103

まず 1ch 分の A/D 変換の値は 10bit にまとめられているので、その内容

は 0~1023 という数値になります。この 10bit の値が 4ch 分あるので、無線通信の効率にも配慮して、1ch あたり 8bit+2bit の補正值として、全体で 5byte (【8bit×4ch=4byte】 + 【2bit×4ch=1byte】 = 5byte) の ADC の値を子機から親機に通知するような設計になっています。具体的にどのように取り扱っているかについては、これから説明します。

アナログ値の取り扱い

◇10bitのAD変換値をどう扱っているか.



※メーカー資料・標準アプリのプログラムから推測

図 104

10bit の AD 変換値の下位 2bit は、バラツキがあるので切り捨てて 0 と考えます。残りの 8bit の下位 2bit を補正值とします。補正值は 4 分の 1 した値と考えられます。残りの 6bit を右に Shift して 8bit として 1ch 分のデータとします。こうしておくで将来マイコンモジュールの ADC の能力が変更になり 12bit となっても対応ができます。このデータを無線通信の電文中に配置して子機から親機に通知しています。この電文を見るとこの通信は、普通のシリアル通信と同じであることが分かります。

データ受信コマンド

◇先頭はコロン【:】で始まるテキストデータ

: 788115017E810E234D0000F3000A7E1F000044FFFFFFFE9B															
①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯
78	81	15	01	7E	810E234D	00	00F3	00	0A7E	1F	00	00	44FFFFFF	FE	9B
送信元デバイスID	コマンド番号	パケット識別子	プロトコルバージョン	受信電波品質	相手の個体識別番号	宛先端末の論理デバイスID	タイムスタンプ	中継フラグ	電源電圧	未使用	デジタル入力値	デジタル入力変更状態	アナログ入力値	アナログ補正值	チェックサム

図 105

実際の通信電文は【データ受信コマンド】として上の図の様に説明されています。先頭がコロン【:】で始まるテキストデータで、電文の中に含まれる情報は、上の図の様に区切って解釈を行う約束です。右側に太枠で囲んだアナログ入力値と補正值があります。電文のこの位置の数値を解析すれば、子機が計測した温度センサーの出力電圧が分かり、それを元に温度の値が変換計算できるという仕組みです。

アナログ入力値

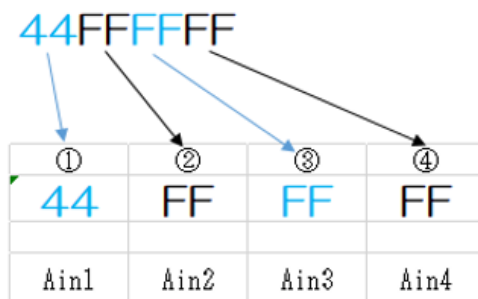


図 106

アナログ入力値の部分は上の図の様に区切られていて、16進数2桁ずつ（1byte）の文字列となって4ch分送信されます。図で Ain1 の部分がアナログ入力1であり、今回温度センサーを接続する部分のデータとなります。

補正值データ

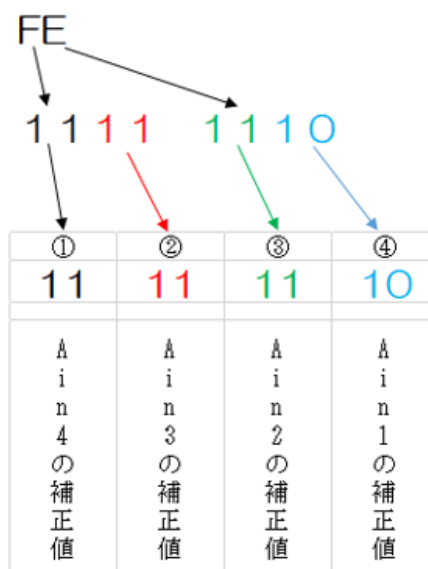


図 107

補正值データは、16進数2桁で通知されているものを2bitずつに区切って解釈をします。各チャンネルの補正值は図の様に配置されています。ch1の補正を行うときは、下位2bitを使用します。これら電文中の情報から、測定電圧を得る仕組みは、次の様に考えられます。

補正を含めた電圧計算

- ◇AD変換の値は標準アプリの中で処理されている。
- ◇計測した電圧の計算は次による。

$$\begin{aligned} & \# \text{ 補正ビットを含めた計算} \\ \text{電圧} & = ((\text{AD値} \times 4) + \text{ch補正值}) \times 4 \end{aligned}$$

※ この計算は、PC内プログラムで処理する。

図 108

受信電文の中に配置されたAD値と補正值を取り出して上のような計算を行えば、元の電圧を知ることができます。この電圧を基にして、次は温度を知る計算を行うのです。それには、温度センサーの特性を調べる必要があります。

温度センサー

◇LM61CIZ リニアな特性

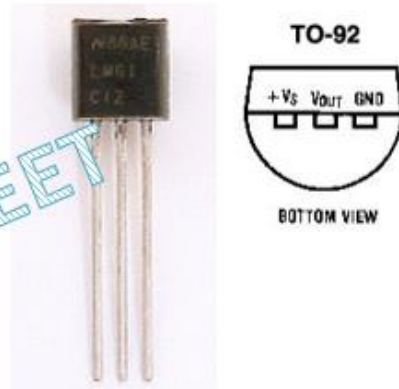
◇測定範囲：-30°C~100°C
-30°C=300mV

~
0°C=600mV

~
100°C=1600mV

◇温度係数：+10mV/°C

◇動作電圧範囲：+2.7~+10V



温度センサー(LM61CIZ)

$$\text{温度} = (\text{センサー出力電圧} - 600\text{mV}) \div 10\text{mV}$$

図 109

今回使用する温度センサーは、一般に広く使われているものです。LM61 シリーズとしていろいろな型番のものが作られています。その中の LM61CIZ というセンサーを使用します。次の資料はデータシートの抜粋です。今回は LM61 シリーズの中の C グレードのセンサーを使用します。

精度が載っている

主な仕様

■ 精度@ 25℃	±2.0℃、±3.0℃(最大)
■ Cグレード精度(-30℃~+100℃)	±4.0℃(最大)
■ Bグレード精度(-25℃~+85℃)	±3.0℃(最大)
■ 検出感度	+10mV/℃
■ 動作規定温度範囲	+2.7V~+10V
■ 待機時消費電流@ 25℃	125μA(最大)
■ 非線形性	±0.8℃(最大)
■ 出力インピーダンス	800Ω(最大)

図 110

データシートには、細かな情報がたくさん書かれていて、難しく思えるところもありますが、次の資料の様に計算方法などの記載もあり、これを使って温度換算を行いますので、役立つ資料です。

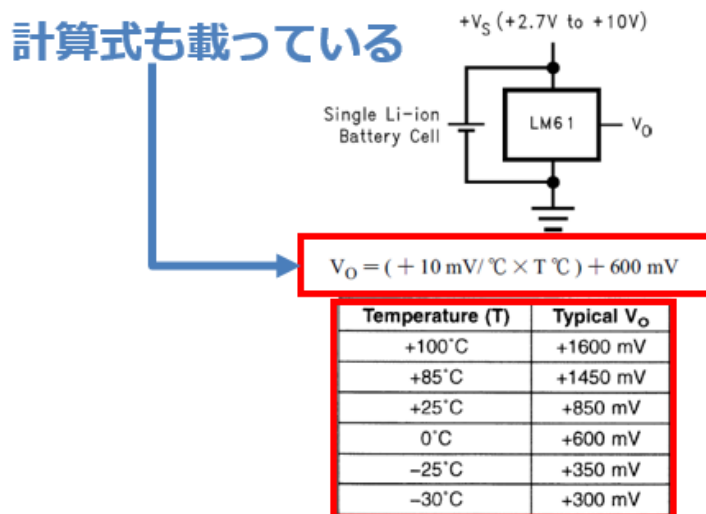


FIGURE 1. Full-Range Centigrade Temperature Sensor (-30°C~+100°C) Operating from a Single Li-Ion Battery Cell

図 111

センサーの温度特性グラフ

◇ センサー特性をもとにA/D変換のレンジを確認。

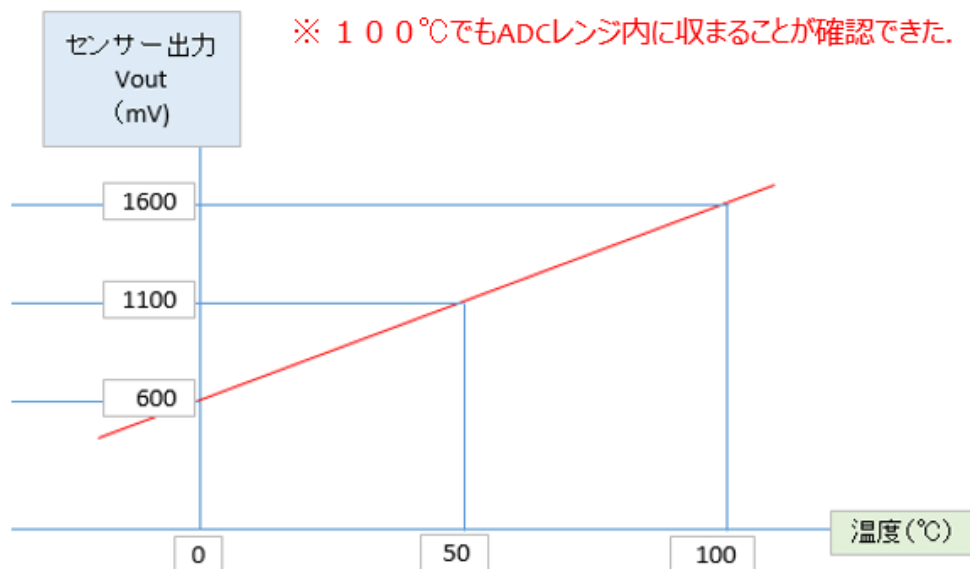


図 112

データシートに記載されているセンサーの特性をもとにして温度と出力電圧の関係をグラフにしたものが上の図です。リニアな特性という表現が出ていましたが、温度と出力電圧は直線で描くことができる正比例の関係であることが理解できます。このグラフで見ると 100°C の温度でも出力電圧は 1600mV となっていて、無線マイコンモジュールの ADC のレンジ (0-2.4V) 内に収まっていますので、安心して使えます。

この出力電圧から温度を求める計算は、つぎの様に行えばよいわけです。

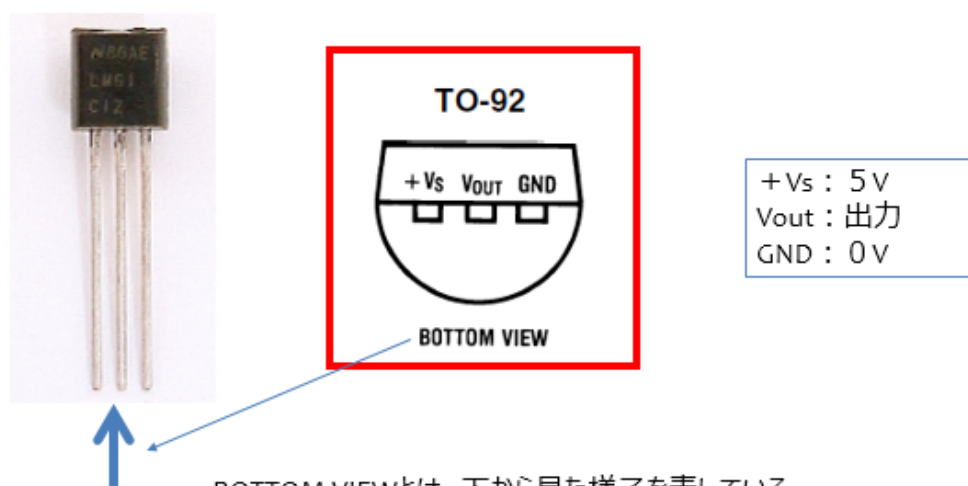
◇電圧値から温度を計算

$$\text{温度〔℃〕} = (\text{電圧〔mv〕} - 600 \text{〔mv〕}) / 10$$

図 113

ピンの誤りは高熱破壊！！

ピン配置を十分確認.



BOTTOM VIEWとは、下から見た様子を表している.

図 114

上の写真は温度センサーのピンを下にして正面（型番が印刷してある面）から見た様子です。中央に【BOTTOM VIEW】という蒲鉾を逆さまにしたような図が描いてあります。これは、センサーのピン側からみた図です。このように見たとき、「左側のピンを電源（+）、中央のピンを出力電圧、右側のピンを GND(0V)に接続する」、ということを表しています。図の右側に +Vs : 5V と書いてありますが、今回の電源電圧は 3V です。このセンサーの動作電圧は +2.7V ~ +10V という記載がデータシー

トにあるので、この 5V は 3V に読み替えてください。

もしこの図を読み間違えて、センサーを反対向きに配線すると、電源と GND が逆になってしまいます。すると、電源を投入したとたんに火傷するほどの高熱を発生してセンサーに致命的なダメージを与えてしまいます。くれぐれも向きを間違えないようにしてください。

今回の実習で必要なパーツは次の通りです。

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. 温度センサー（LM61CIZ）×1個

親機側：

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

今回は Python という言語を使って PC の処理プログラムを作ります。

この環境準備については、後ほど説明します。

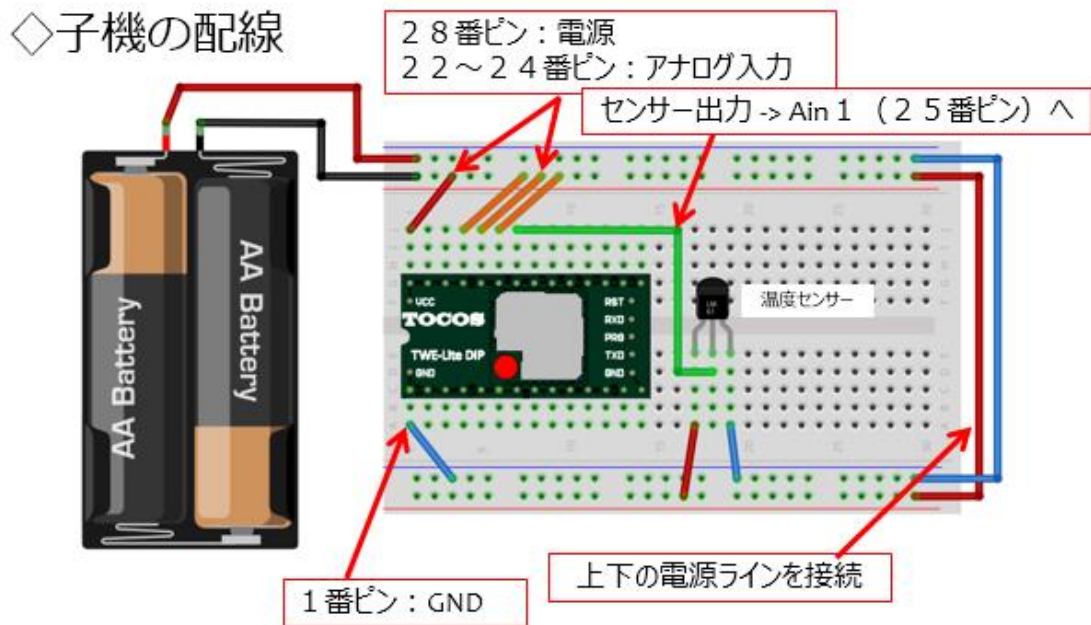


図 115

パーツの準備が出来たら上の図に従い子機を配線します。温度センサー

に必要なジャンパー線はわずか 3 本ですから、間違えることはないと思いますが、センサーの向きには十分注意してください。

実際に配線した様子 子機

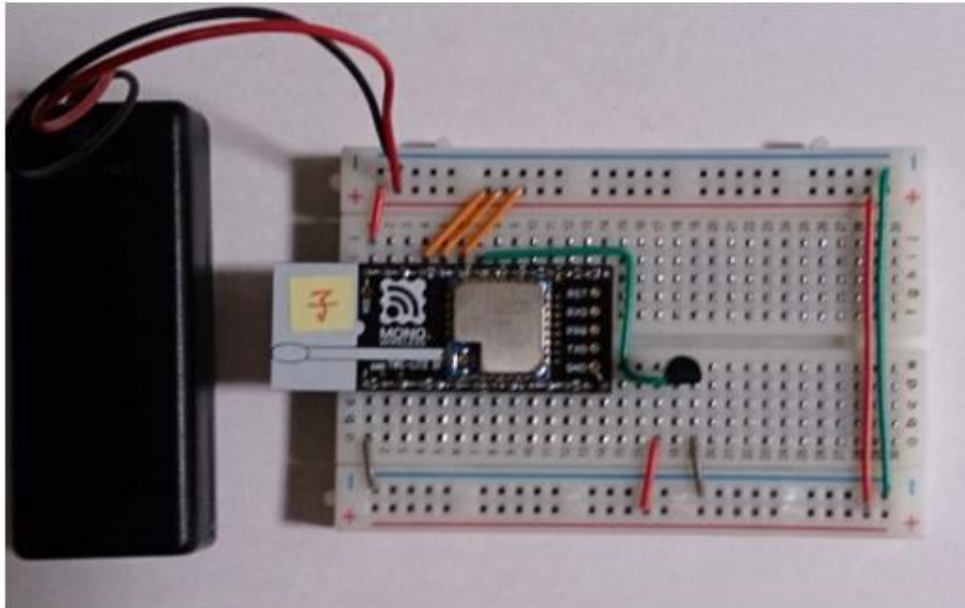


図 116

実際に配線したものは上の写真の様になりました。温度センサーの中央のピンは、ジャンパー線によって無線マイコンモジュールの AI1 (25 番ピン) に接続されています。

Pythonとは

- とてもクリーンで読みやすい文法
- 強力な内省(イントロスペクション)機能
- 直感的なオブジェクト指向
- 手続き型のコードによる、自然な表現
- パッケージの階層化もサポートした、完全なモジュール化サポート
- 例外ベースのエラーハンドリング
- 高レベルな動的データ型
- 事実上すべてのタスクをこなせる、広範囲に及ぶ標準ライブラリとサードパーティのモジュール
- 拡張とモジュールはC/C++で書くのが容易(JythonではJava、IronPythonでは.NET言語を利用)
- アプリケーションに組み込んでスクリプトインタフェースとして利用することが可能

図 117

今回、PC 側処理系では Python を使うと説明しました。皆さんは Python (パイソン) という言語を既にご存知で利用されている方もいらっしゃるのではないのでしょうか。お使いでない方でも、書店のコンピュータ関連書籍の棚に Python という見出しの雑誌や書籍を見かけていると思います。最近では AI やビッグデータの処理によく使われて、それらの特集記事が雑誌やメルマガで手元に届くようになってきました。

環境を選ばない

Pythonはどこでも実行可能

Pythonは、Windows、Linux/Unix、OS/2、Mac、Amigaなど多くのメジャーなオペレーティング・システムで使うことができます。これ以外にも.NETやJava仮想マシン、Nokia Series 60携帯電話で動くバージョンもあります。一度書いたソースコードが、変更なしにすべての環境で動くことを知ると、うれしくなってくるでしょう。

あなたのお気に入りのシステムが登録されていない？もし、その環境でCコンパイラが利用できるのであれば、おそらくPythonが動作するでしょう。ぜひ、news:comp.lang.pythonに質問してみてください。

◇Androidもiosでも、動きます。

図 118

Pythonは、環境を選ばずどこでも実行することができ、Androidやiosでも動きます。最近ではMicro Pythonと言って、マイコン上で稼動する環境も開発されています。技術革新は高速ですから、私が行うマイコン制御の実習もPythonで行ようになる 때가1,2年後には到来するかもしれません。次は環境の準備です。

Python開発環境

◇この講座では・・・

1. Python2.7系を使用.
2. Pyserialを使用.

◇下記サイトから、Pythonのファイルをダウンロードします。

<https://www.python.org/>

図 119

この講座では、Python2.7系を使用し、PCと無線マイコン親機間でシリアル通信を行うので、Pyserialというライブラリも使います。

上記のWEBサイトからPythonのファイルをダウンロードします。

PythonのWebサイト

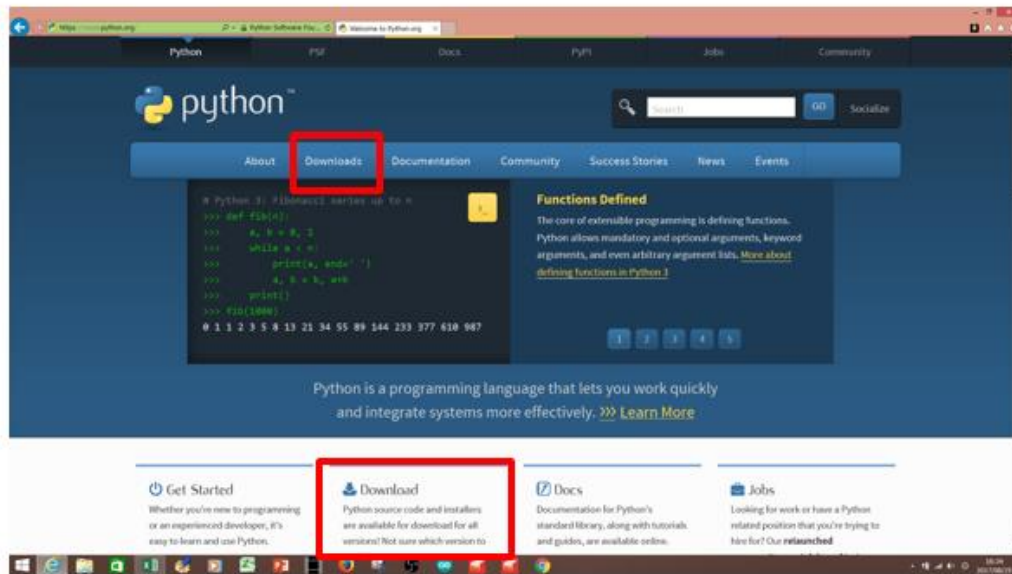


図 120

Python2.7系

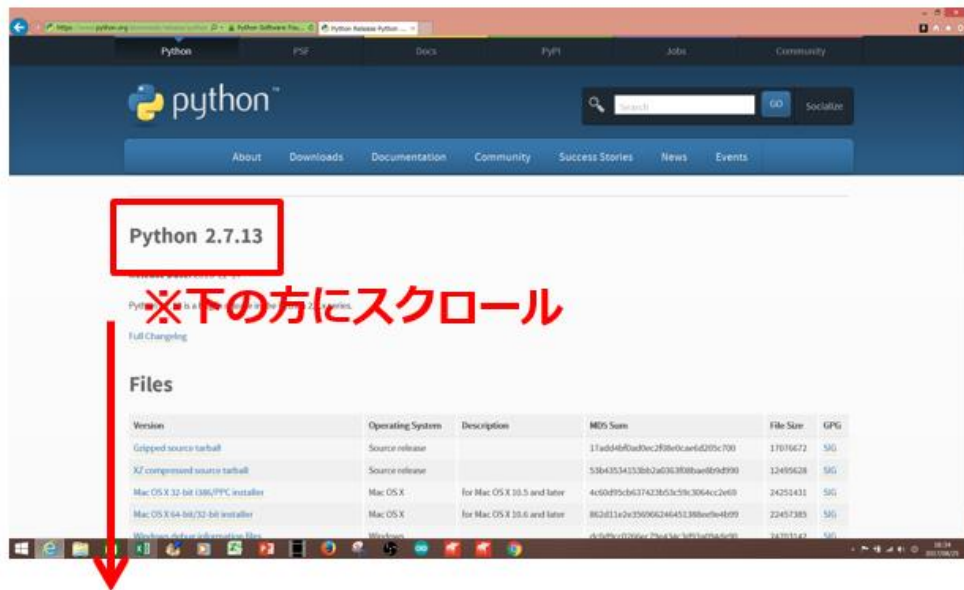


図 121

Windows x86 MSI Installer

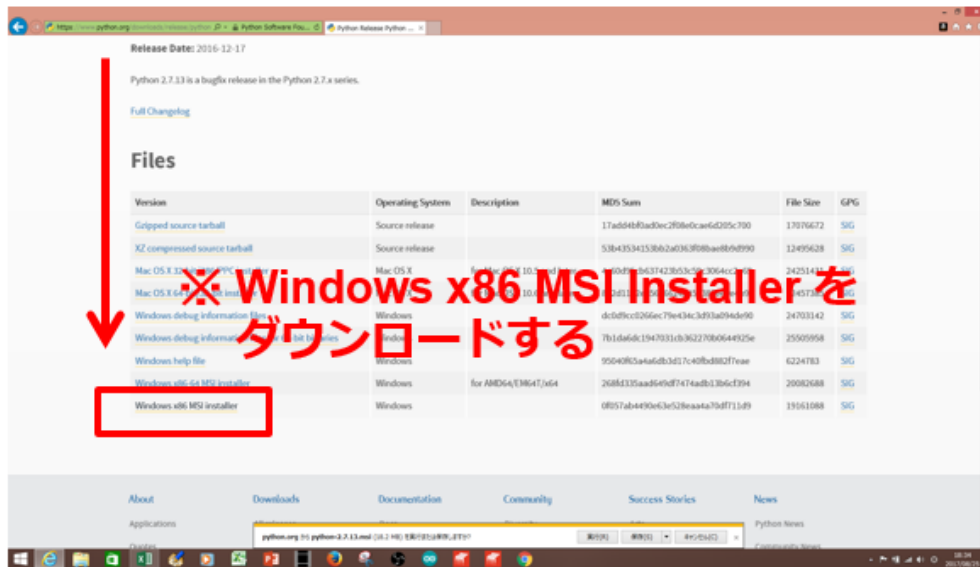


図 122

Install → パスの確認

- ◇ダウンロードしたファイルをInstall.
- ◇環境変数にPython27のパスが追加されている.

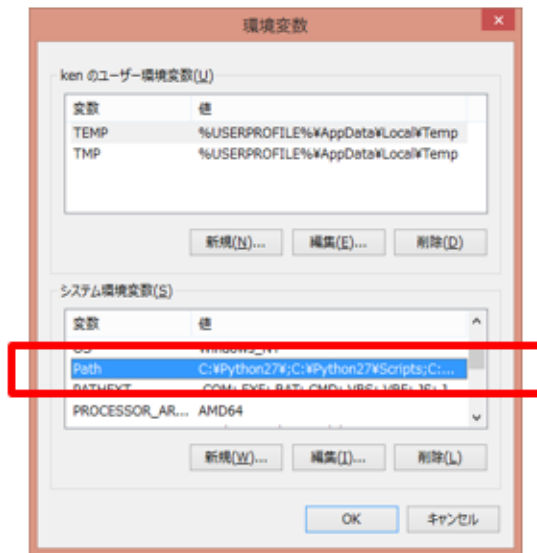
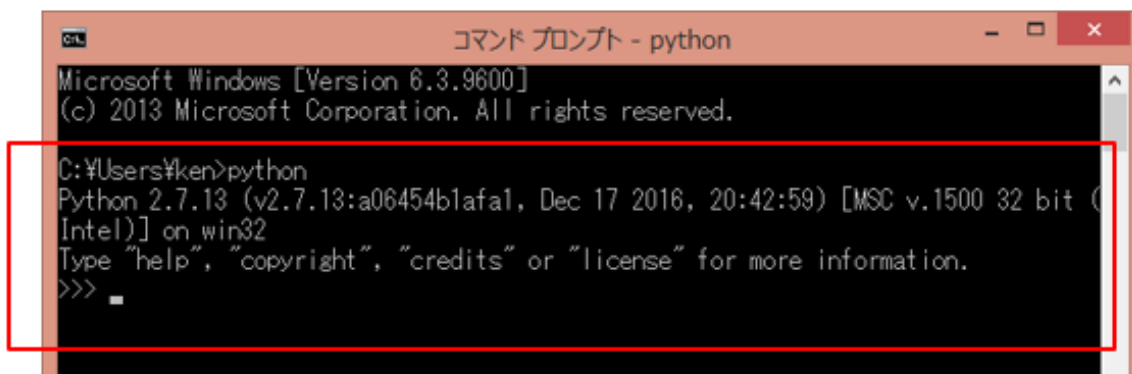


図 123

ダウンロードしたファイルをインストールすると、Windows の環境変数に Python27 の path が追加されているはずです。

起動テスト

- ◇ コマンドプロンプトにpythonと入力し、pythonが起動すればOK.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\ken>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> .
```

図 124

上の図のように起動テストを行って下さい。コマンドプロンプトに

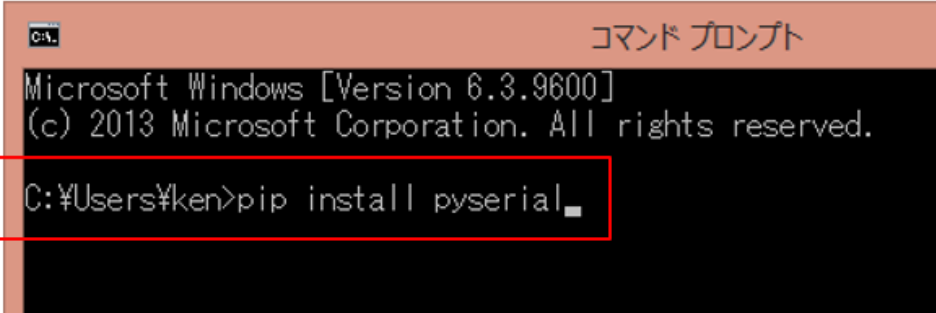
python

と小文字で入力して Enter キーを押下します。>>>というプロンプトが表示されれば OK です。

次に PC と親機間の通信で使用する Pyserial ライブラリをインストールします。

Pyserialライブラリ

- ◇PCと親機の通信に シリアル通信を使用.
- ◇Pyserial を Install .
 - コマンドプロンプトで
 - > pip install pyserial
 - と打つ.



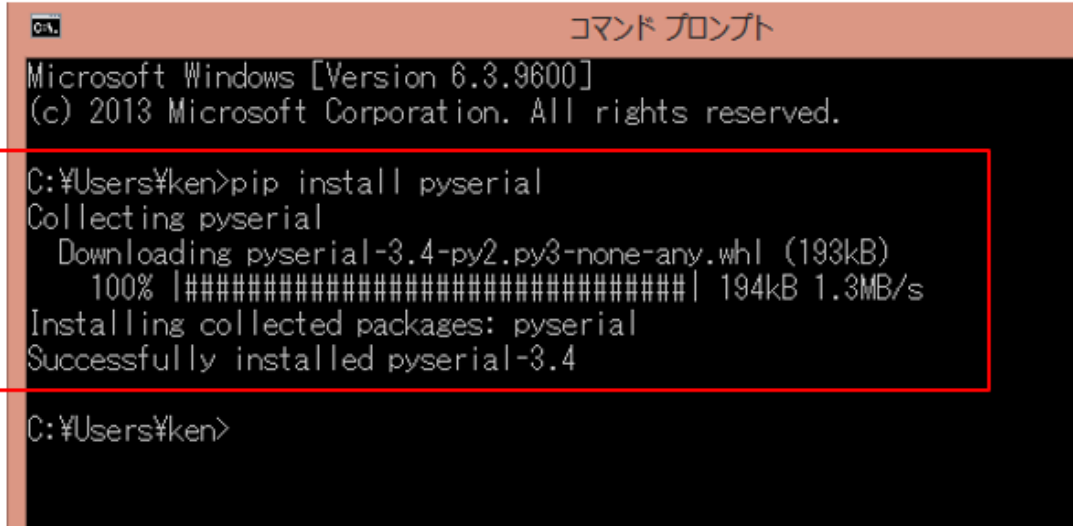
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\ken>pip install pyserial
```

図 125

上の図の様にコマンドを入力すると、下の様にインストールができます。Successfully の文字が表示されればライブラリも準備完了です。

◇こんな感じでInstallができる。



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\ken>pip install pyserial
Collecting pyserial
  Downloading pyserial-3.4-py2.py3-none-any.whl (193kB)
    100% |#####| 194kB 1.3MB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.4

C:\Users\ken>
```

図 126

念のために、ライブラリが使えるか確認をしておきます。

Python をインストールすると IDLE というプログラムが追加されていますので、それを起動して、そのプロンプトに次の Python コマンドを入力してみます。

```
import serial
```

Enter キーを押下したとき、エラー表示も無く、プロンプト>>>が表示されれば、ライブラリはセットアップされています。

◇ライブラリが使えるか、確認.

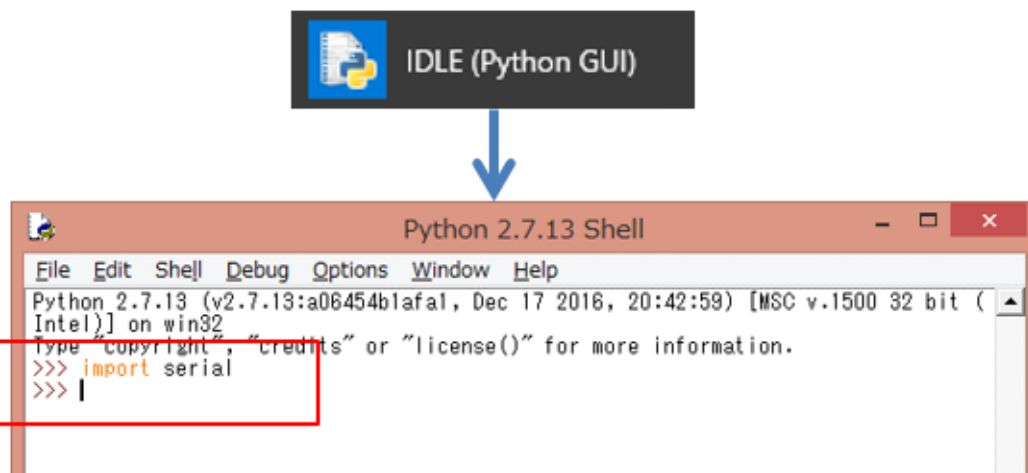


図 127

◇プログラム作成

- ◇テキストエディタでソースコードを入力.
- ◇*.py と拡張子に【py】を付けて保存.

図 128

いよいよ Python のプログラム作成です。プログラムは使い慣れたテ

キストエディタ（私は TeraPad というフリーのテキストエディタを長年利用しています。）で、ソースコードを入力し、拡張子に【py】を付けて保存して下さい。ソースコードは後に示しますが、この講座の Python のプログラムは大まかに次のような構造をしています。

プログラム 全体構造


モジュールなどの取り込み	<pre> ----- 2017.08.16 ----- LM61C1Z,B1Z 温度センサー ----- 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 </pre>
関数の定義	<pre> 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 </pre>  <pre> return result # 呼出元に戻る </pre>
処理のエントリーポイント メイン処理部	<pre> # ここから、処理開始 # COM5を開く<---自分の環境に合わせてCOMポート番号を指定する s = serial.Serial('COM5', 115200) # COMポート番号、通信速度 while 1: # ずっと繰り返し data = s.readline() # シリアルポートから1行読取り 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 # COMを閉じる s.close() </pre>

図 129

最初にモジュールなどの取り込み、次に関数定義の部分、そして処理のエントリーポイントとメイン処理部分となっています。#より右側はコメントです。

では以下にソースコードを示します。

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

```
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
```

電文処理関数

◇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(">BBBBIBHHBBBBBBBB") # 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
```

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
```

結果をまとめた【辞書】を作る

◇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
    "battery" : 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

◇main() 関数のような部分は、一番下を書く。

```
# ここから、処理開始
# COM5を開く<---自分の環境に合わせてCOMポート番号を指定する
s = serial.Serial('COM5', 115200) # COMポート番号、通信速度

while 1: # ずっと繰り返し
    data = s.readline() # シリアルポートから1行読取り
    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 # 温度を表示する

# COMを閉じる
s.close()
```

※厳密には、もっと書いた方が良いところもあるが・・・

【注】ソースコードは、WEBよりダウンロードできる環境が整いましたら、そちらからダウンロードしてお使いください。ソースコードができましたら、【py】という拡張子を付けて、適当な場所に保存してください。

【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。

これから動作確認を行いますが、手順を間違えると正しく動作しません。

1. MoNoSTICKをPCにセットして認識させる。2. 子機の電源をONする。3. Pythonのプログラムを実行する。です。

親機と子機の準備

- ◇ MonoStickをPCにセットする。
- ◇ 子機の電源をON！！

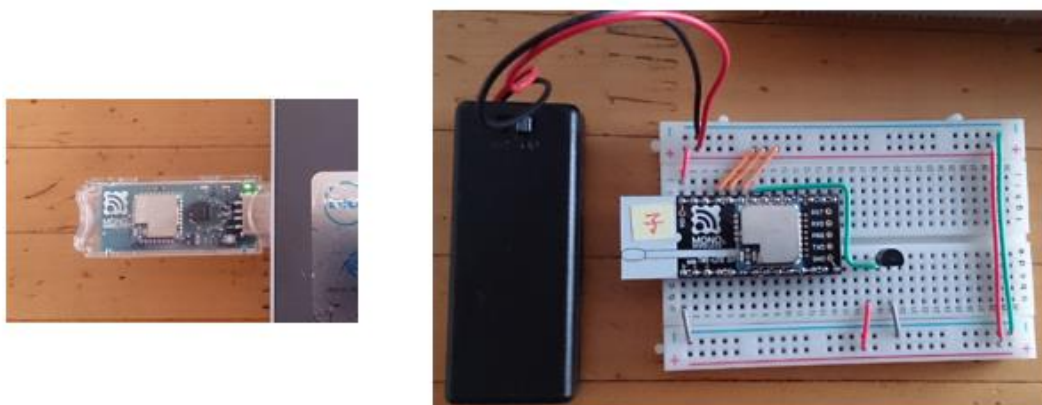


図 130

IDLE を起動し、Python のプログラムを開きます。

Pythonプログラム実行方法

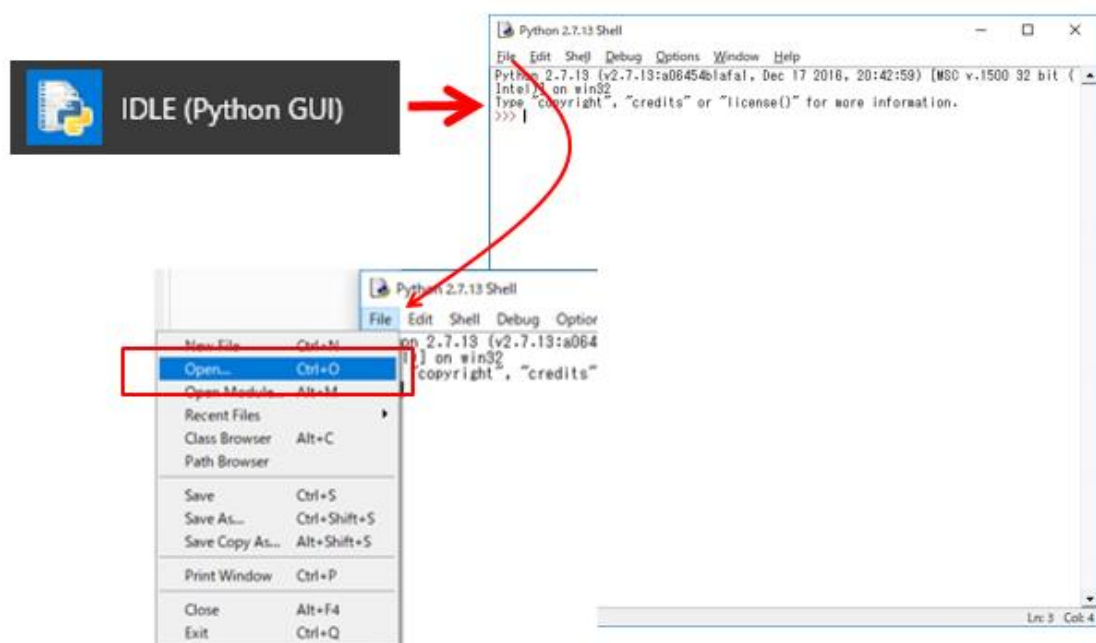


図 131

開いたソースコードファイルのメニューで **Run...>Run Module** と選択すると、別のウインドウが開いて実行が始まります。

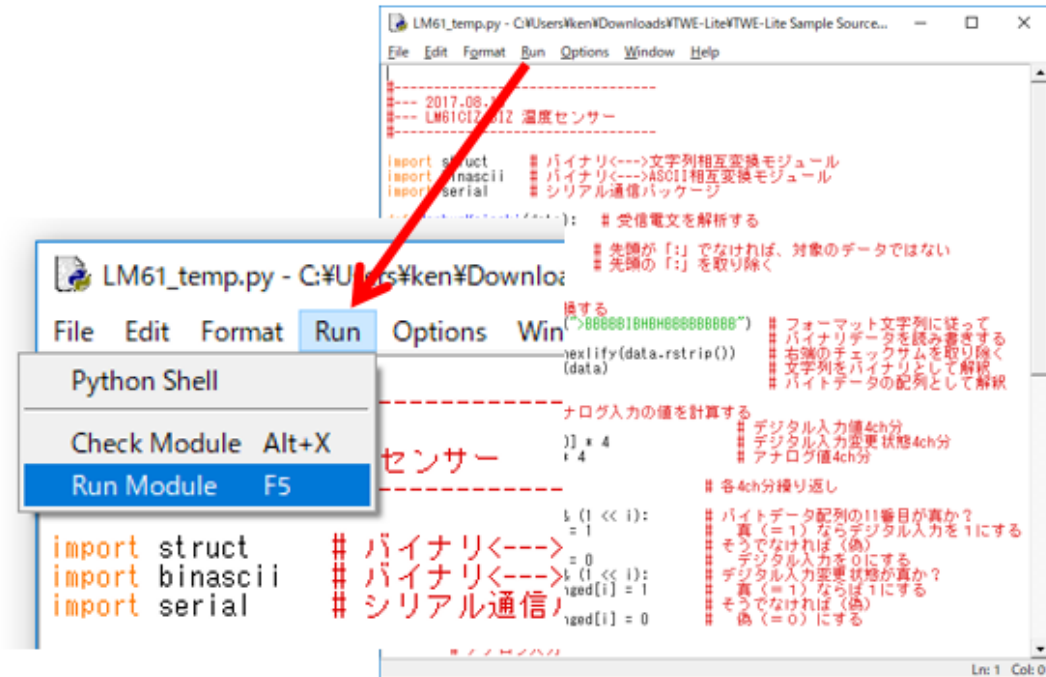


図 132

動作確認

- ◇温度が標示される.
- ◇感度や精度について考えてみる.

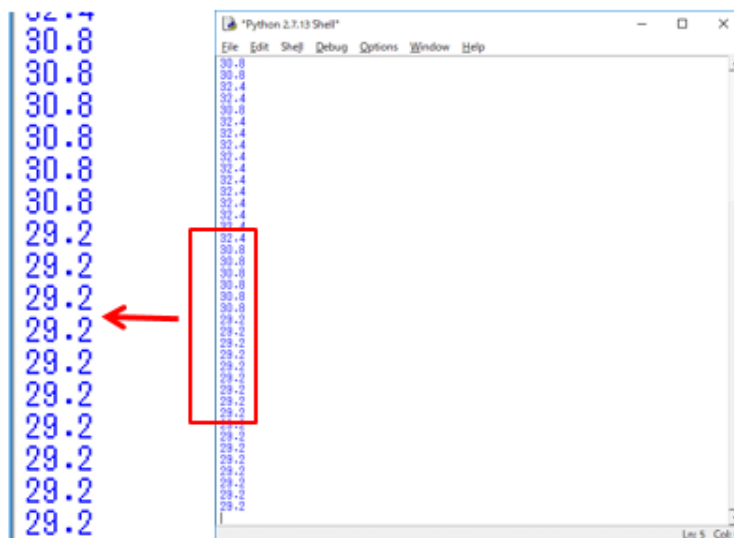


図 133

ここまでの準備が完璧なら、次のようにウインドウに温度が表示されていきます。温度センサーの頭に指を触れてみて下さい。体温が伝わり表示温度が上昇します。指を離せば温度は低下します。

子機は電池駆動で親機とは無線通信をしていますから、測定場所を移動することが容易にできます。これが無線マイコンモジュールの便利なところですよ。

今回の講座では、温度表示を PC 画面に行っているのので、測定場所では温度が分かりません。この点を改善するために、次の講座で表示器の使い方を学びます。

第10回 液晶表示器(LCD)

今回の講座では液晶表示器（LCD：Liquid Crystal Display）の使い方を学びます。無線で離れたところにメッセージを表示することができます。

◇メッセージを無線通信 液晶表示



図 134

PC でメッセージを作りそれを無線通信の電文に埋め込んで、親機から子機に送ります。子機は受信した電文に含まれるメッセージを液晶表示器に表示します。第 9 回で、温度センサーの情報を子機から親機に送り、PC で処理して温度を求めて表示したのは、情報の流れる方向が逆になります。

◇システムの全体構成

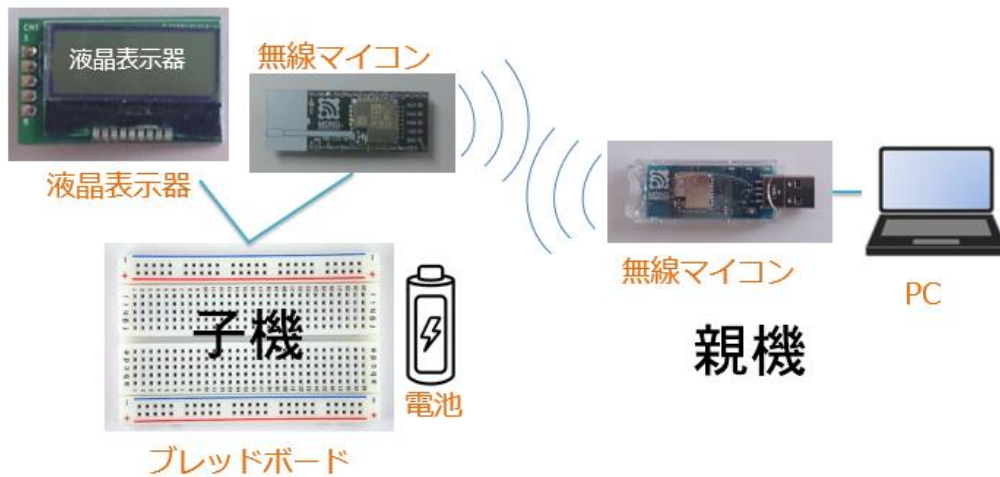


図 135

第 9 回の温度センサーに代わって、液晶表示器（LCD）が子機で使われます。温度センサーは子機に対する入力デバイスでしたが LCD は出力デバイスとなります。今回は PC で固定のメッセージを作りそれを子機の LCD に 2 行で表示してみます。PC 内部のプログラムは Python で開発します。

◇I2C I/F を持つ LCD

- ◇I2C I/F というシリアル通信規格.
- ◇LCDの初期化などは、標準アプリ内で処理.
- ◇標準アプリの可能な範囲で液晶表示.
- ◇細かな制御は、ここでは考えない.

図 136

今回使用する LCD は I2C I/F という 2 線式のシリアル通信規格を使用するものです。I2C というのはクロックとデータを伝える電線を準備

して、デバイスはその 2 本の信号線にぶら下がる形で種類の違うデバイスでも複数台が接続できます。デバイスごとの識別は I2C スレーブアドレスという番号がデバイスごとに決まっています、その番号で通信相手(スレーブ)を指定して通信を行います。スレーブに対するマスターは、無線マイコンモジュールの子機が担当します。この様を書くとは何やら難しそうですが、実際はそうでもありません。この講座で使用している無線マイコンモジュールに、最初から書き込まれている標準アプリケーションの機能として、サポートされている液晶表示器があります。それを使うことで、容易にシステムを開発できます。利用できる液晶表示器は型式コードが決まっています。

型式コードが決まっている

- ◇標準アプリで、LCD (AQM0802A)は、
型式コードが **0x22** と決められている。
※これはI2Cスレーブアドレスとは異なる、
独自の番号である。
- ◇この型式コードを付与した表示文字列データを
標準アプリにシリアル通信で
渡すと、LCDの初期化から、
表示まで、その都度実行される。



AQM0802A

※標準アプリは AQM0802A と ACM1602 の2機種のみ対応。

LCD は上に示す AQM0802A を使います。異なる LCD は使用できません。但し、無線マイコンモジュールのアプリケーションを作成して独自にサポートする LCD 制御プログラムを内蔵させれば、別のものでも使用できます。

I2Cデータ書込みコマンド

◇先頭はコロン【:】で始まるテキストデータ

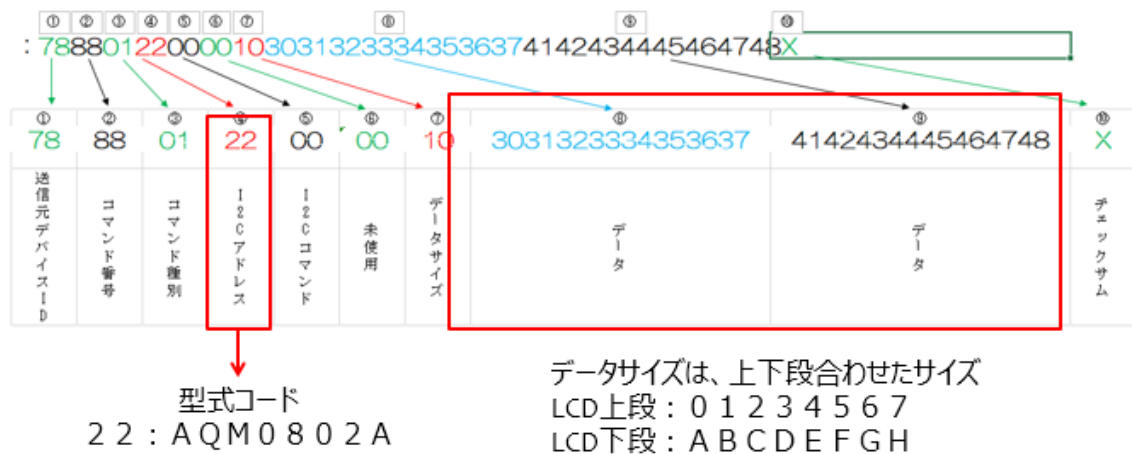


図 138

親機側から子機に対して制御を行う無線通信の電文に I2C データ書き込みコマンドというものがあります。マイコンモジュール内蔵の標準アプリでは、この電文をサポートしています。

【注意】この電文の解説がメーカーWEB ページで公開されています。また、書籍「TWE-Lite ではじめるカンタン電子工作」(I/O BOOKS) でも解説されていますが、標準アプリのバージョンが更新されたためか、解説が古いものとなっていて、実際の無線マイコンモジュールの電文とは一部解説の内容が違っているようですので、注意してください。

上記電文で I2C アドレスと表記してある箇所が実は液晶表示器の内部型式コードとなっています。ここに 16 進数で 22 という型式コードを埋め込むと、無線マイコンモジュールの標準アプリは LCD として AQM0802A を制御対象とした操作をするように動きます。LCD に表示するメッセージはデータと書かれた箇所に 8 文字×2 列 (LCD 上段と下段) のデータとして埋め込みます。合計で 16byte のデータになるので、データサイズ (データの左側) は 16 進で 10 となっています。この電文を送信すると、子機はその都度 LCD をリセットして表示を消去し、必要な初期化処理を行い上下 2 段の文字列表示を行う動作をします。

上の電文を作り、子機に対して送信するプログラムを Python で開発し PC で動かすことで、無線通信で子機の LCD にメッセージを表示できます。使用する液晶表示器の詳細は次のようになっています。データの部分に埋め込む文字コードを変化させることで、自由なメッセージを表示することができます。

液晶表示器 LCD

- ◇ 8文字×2行
- ◇ I2C I/F (2本の信号で通信を行うI/F)
- ◇ I2Cアドレス → 0x7C
- ◇ 制御コマンド → 0x00 + Command
- ◇ 文字データ → 0x40 + Data

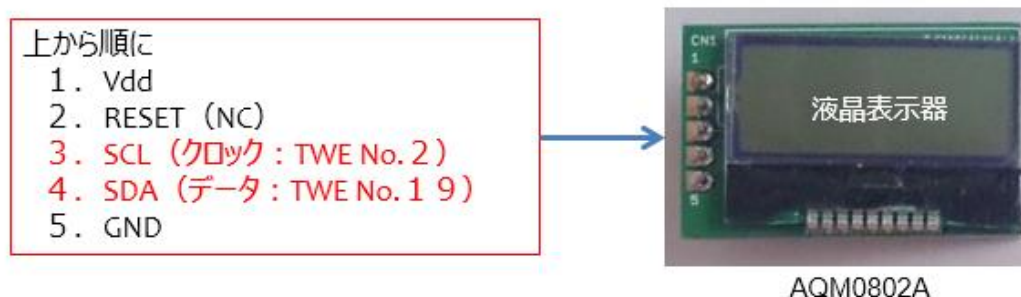


図 139

液晶表示器には、上の図の様に、5本のピンが取り付けられています。写真では上から順に番号が付いています。1番がVddと記載がありますが、これは電源(+)のことです。ここでは3Vになります。2番はLCDのREST信号ですが、(NC:Not Connect)とありますので、なにもつなぎません。3番はI2C I/Fのクロック信号で無線マイコンモジュールの2番ピンに接続します。4番はI2C I/Fのデータで無線マイコンモジュールの19番ピンに接続します。5番はGNDです。

今回の実習で必要なパーツは、次の通りです。

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

親機側：

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

下の図に従って配線を行います。LCDとマイコンのピン接続を枠内に記載してありますので確認をしてください。

◇子機の配線

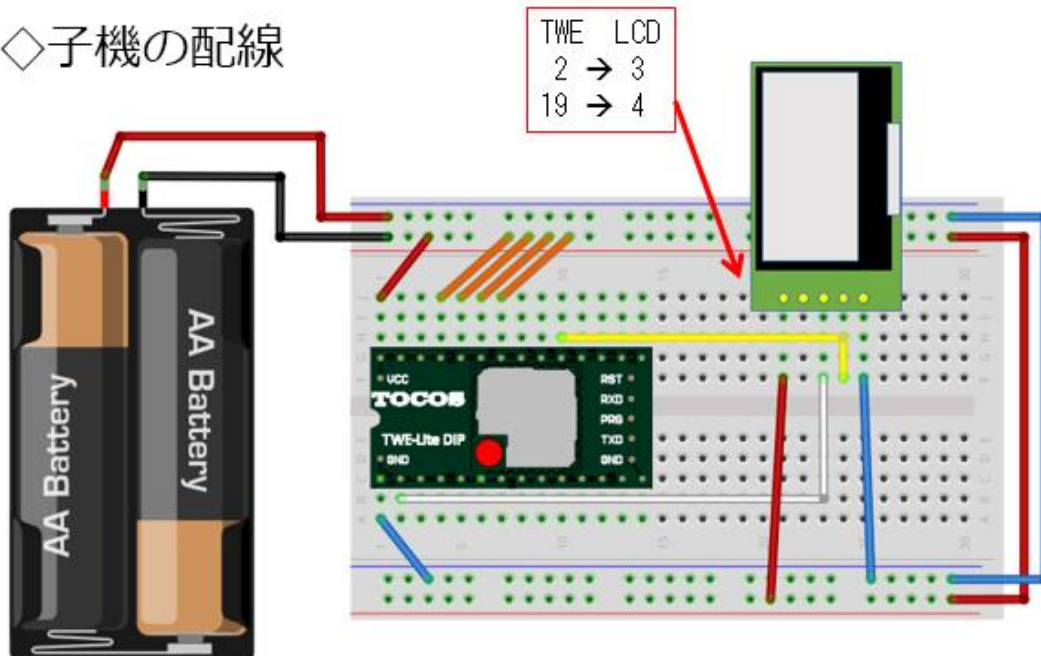


図 140

実際に配線をした子機は次の写真のようになっていきます。

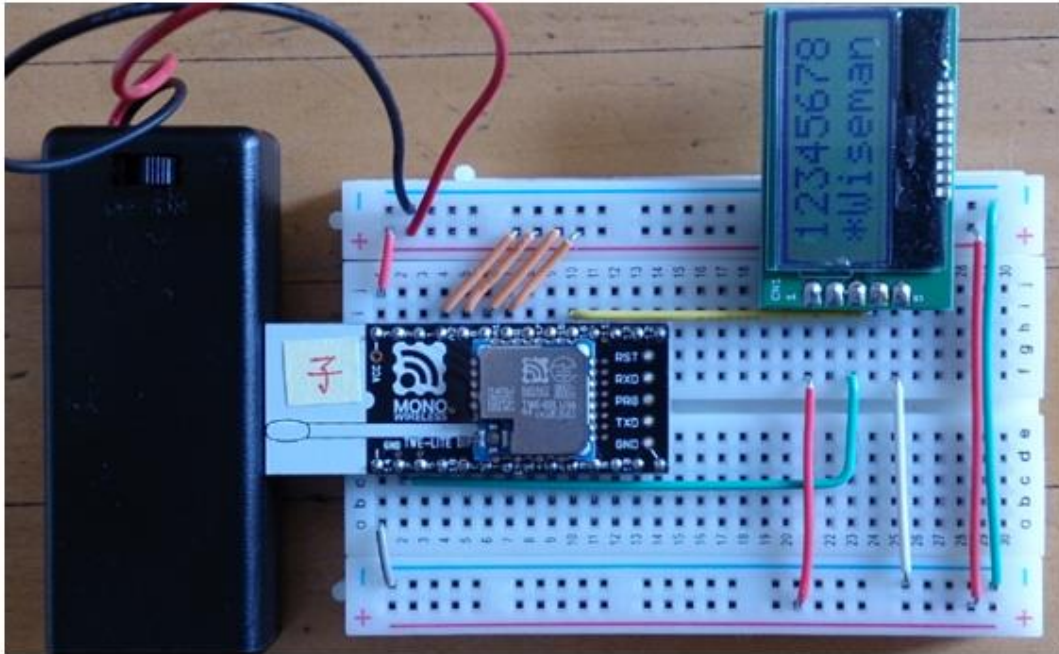


図 141

この写真は既に動かしているものですが、正しく動作すれば写真の様に LCD の上下 2 段にメッセージが表示されます。

既に Python 開発・実行環境は整っているはずですから、次はプログラムの作成です。ソースコードを下記します。

◇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
import time        # 日付・時刻データを取り扱う
```


◇LCD制御電文送信 序盤

```
# TWE-Liteの標準アプリ内のI2C関数をシリアル通信を経由して制御する.
def accessI2C(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: # 引数が-1のとき、I2Cに書き込むだけ
        # 送信データのリストを作る.
        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:
        # 引数 > 0 のとき、
        # readbyteだけ読み取る (dataは利用しない)
        # 送信データのリストを作る.
        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
```

◇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
    accessI2C(s, i2caddress = 0x22, i2ccommand = 0x80, data=list(msg))
    return
```

◇処理の本体

◇main() 関数のような部分は、一番下を書く。

```
# COM5を開く
s = serial.Serial('COM5', 115200)

# データを出力する
writeAQM0802Msg(s, "12345678", "*Wiseman")

# COMを閉じる
s.close()
```

◇*.py という名称でファイル保存

【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。

“12345678” ,”*Wiseman”の部分を書き換えれば、自由なメッセージを表示することができます。

※自由といっても、ASCII コードの範疇のみ可能で、漢字やひらがななどは表示できません。ソースコードは【py】という拡張子を付けて保存して下さい。

親機と子機の準備

- ◇親機をPCにセットする.
- ◇子機の電源をON！！
- ◇Python PG 実行

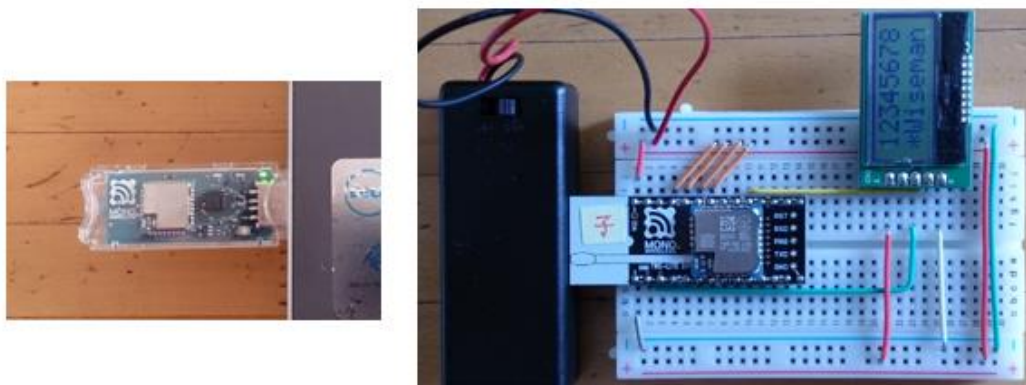


図 142

動作確認の準備です。親機を PC の USB コネクタに差込み、認識させます。IDLE を起動して Python のプログラムを開きます。

Pythonプログラム実行方法

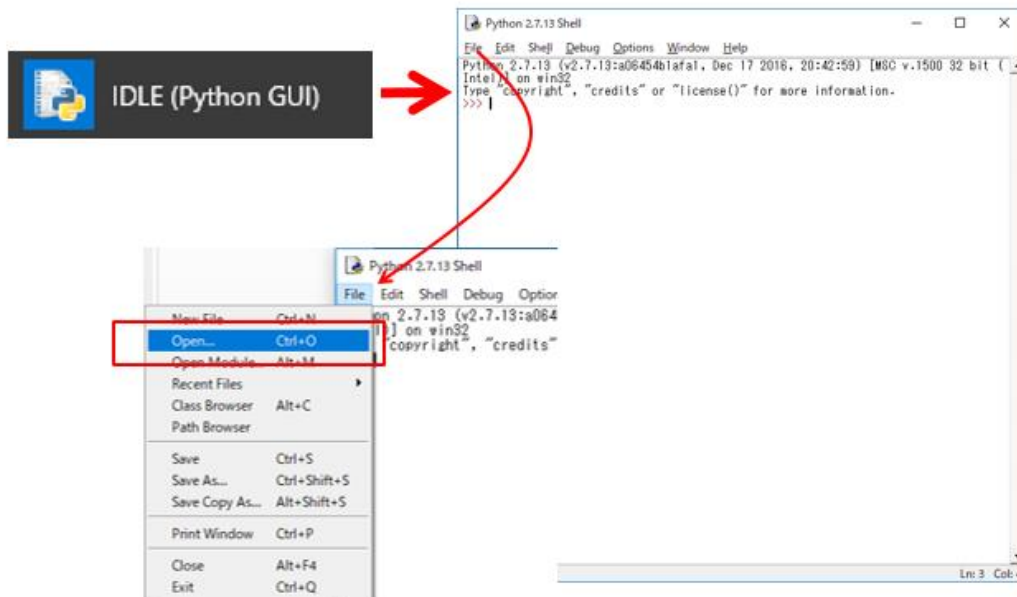


図 143

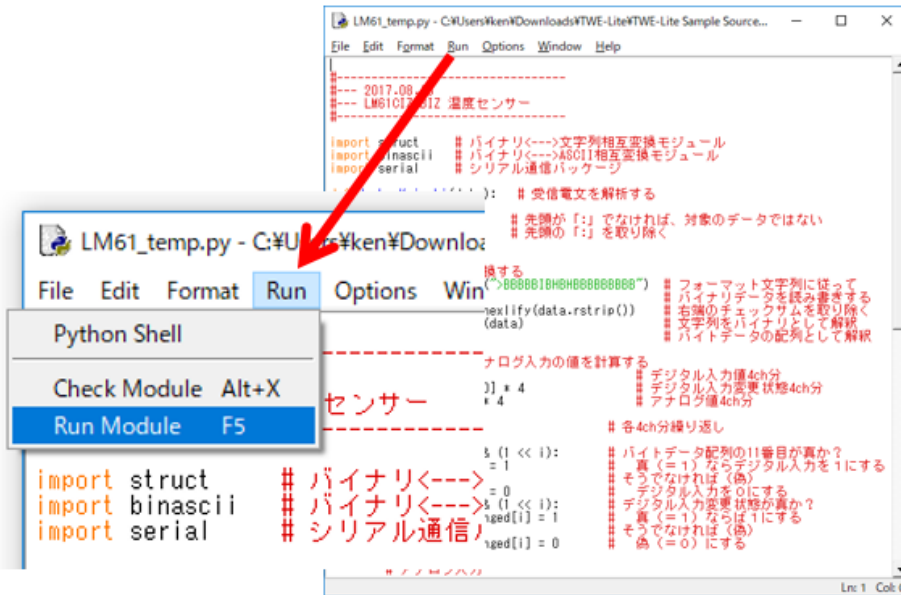
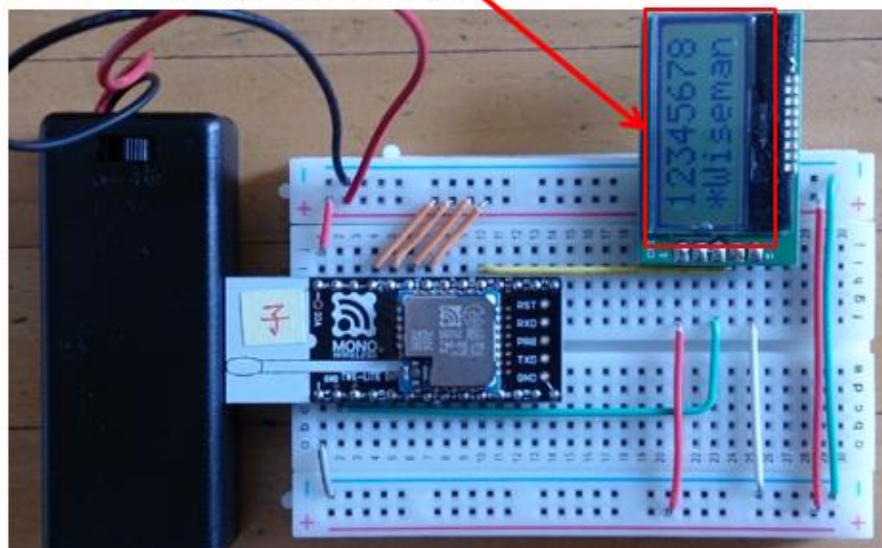


図 144

Run--->Run Module と選択すると別のウィンドウが開き、Python のプログラムの実行が始まります。

動作確認

◇メッセージが表示される.



◇動的なメッセージを表示するには...

図 145

プログラムに実行はすぐに終わり、少し遅れて LCD にメッセージが表示されます。ここまですると次は変化するメッセージを表示するにはどうすれば良いか、知りたくなります。

第11回 デジタル温度計

第11回の講座は、前2回分の講座の合体版です。温度を計測し LCD に表示するデジタル温度計です。

無線で温度を受信し液晶表示する

◇無線の双方向通信
受信：温度計測値
送信：液晶表示文字列。

図 146

子機側には、温度センサーと LCD が使われます。

◇温度データ送受信

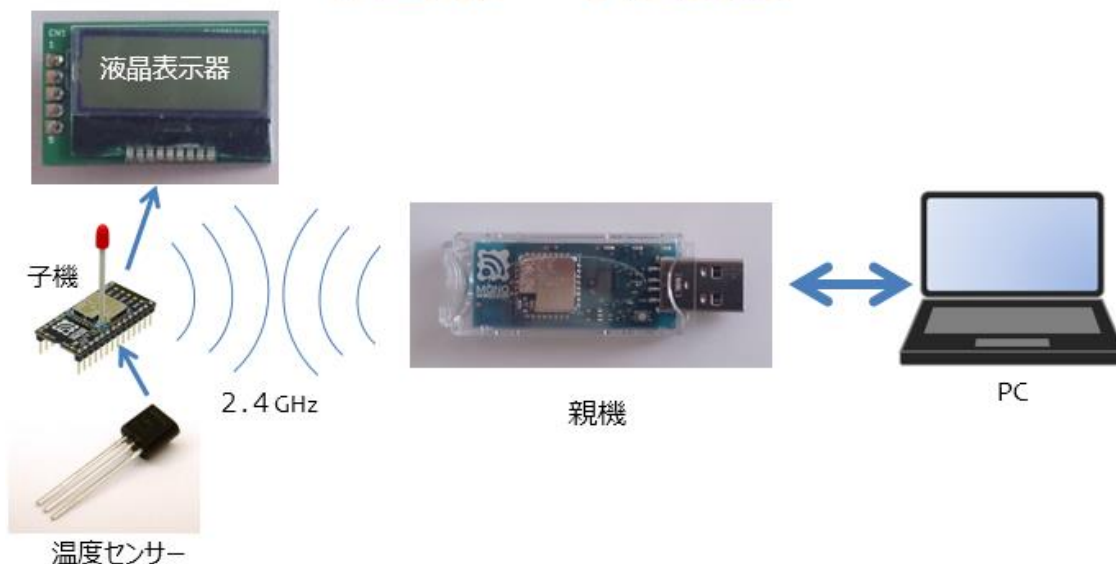


図 147

◇システムの全体構成

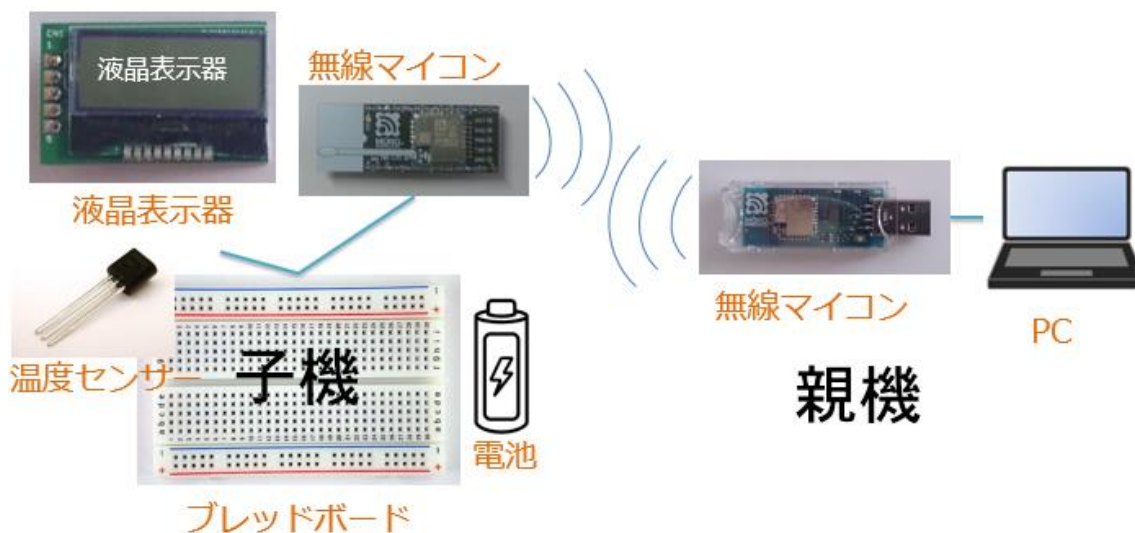


図 148

PC から見ると子機とやり取りするデータは、受信側は温度センサーの計測値、送信側は LCD に表示するメッセージです。

必要なパーツは、次の通りです。

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

下の図に従って配線を行います。LCD とマイコンのピン接続を枠内に記載してありますので確認をしてください。

◇子機の配線

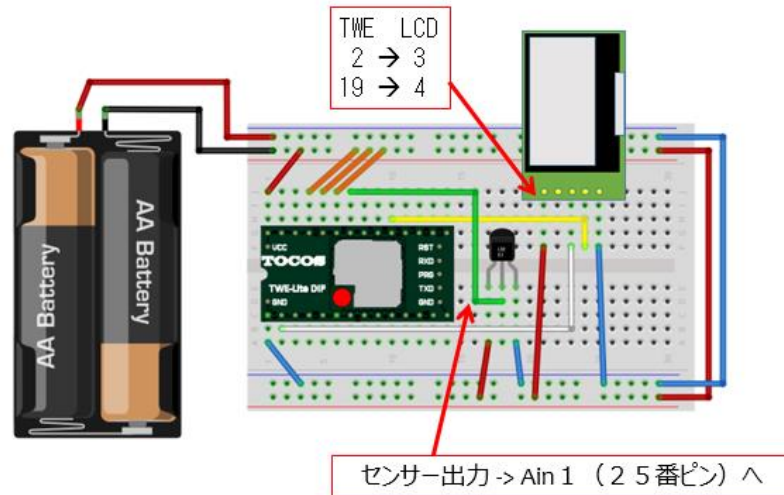


図 149

前 2 回分、第 9 回と第 10 回の子機が合体した回路になっていることが分かりますね。温度センサーと LCD の配線は、わずか 7 本ですから、簡単ですね。配線誤りがないか良く確認をしてください。

実際に配線した様子 子機

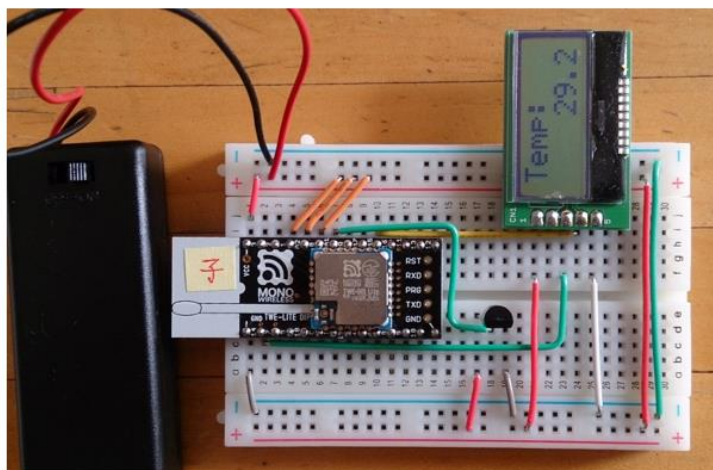


図 150

◇ソースコードを記載しますが、前 2 回分のものを合体したソースコードになっています。

◇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
import time        # 日付・時刻データを取り扱う
```

◇LCD制御電文送信 序盤

```
# TWE-Liteの標準アプリ内のI2C関数をシリアル通信を経由して制御する。
def accessI2C(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: # 引数が-1のとき、I2Cに書き込むだけ
        # 送信データのリストを作る。
        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: # 引数 > 0 のとき、
        # readbyteだけ読み取る。(dataは利用しない)
        # 送信データのリストを作る。
        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(">BBBBB") # 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(">BBBBBIBHGHBBBBBBBB") # 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 # 補正值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
    "battery" : 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
    accessI2C(s, i2caddress = 0x22, i2ccommand = 0x80, data=list(msg))
    return
```

◇処理の本体

```
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

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

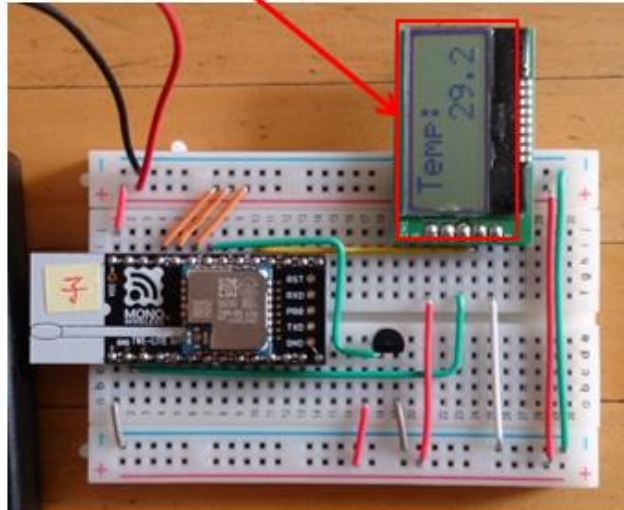
        writeAQM0802Msg(s, "Temp:", " %2.1f" % avr) # データを出力する
        avr = 0

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。

◇動作確認

◇温度が表示される.



◇センサーに指を触れて温度変化を見ましょう.

図 151

既に前 2 回の講座で、動作確認の手順は説明しましたので、省略します。必要でしたら第 9 回、第 10 回を参照してください。今回の Python のプログラムは、動作させると複数回計測した平均値を表示するようにしています。

センサーを持つノードが複数あり、色々なところの計測値をまとめて一か所の LCD で表示する。そして、その表示の内容が PC に蓄積されて後に統計、分析などに利用できるようなシステムも開発できそうですね。

第12回 SW 状態検出

さて次は、SW 状態を検出してみましょう。SW は ON/OFF の状態をマイコンに入力してくれます。その状態は、デジタル入力の信号を読めばわかります。

◇SWの状態を検出。
変化したとき
知らせる。

図 152

第 12 回では、この状態が変化したことを検出して、通知するというものです。



図 153

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、ここで既に行ったようなことを再び実習するのかというと、実は次の第 13 回から本格的な IoT の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 を構築する際の問題の発生場所を限定できるからです。第 12 回で、そのような環境整備をしっかりとしておくことで、後の第 13 回、第 14 回をスムーズに運ぶという狙いがあります。

◇システムの全体構成

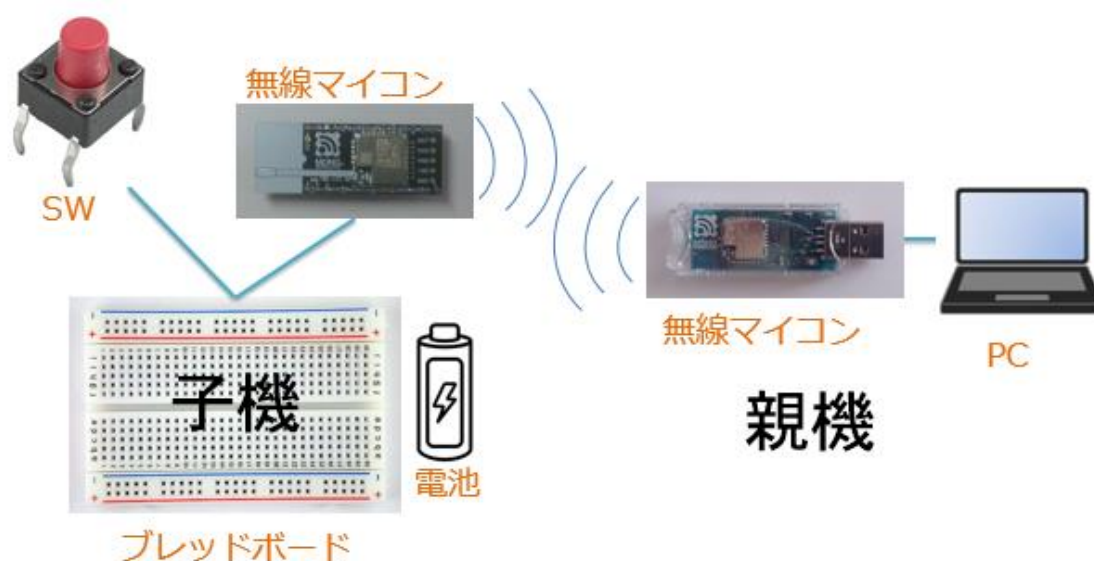


図 154

システムの全体構成は、上の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

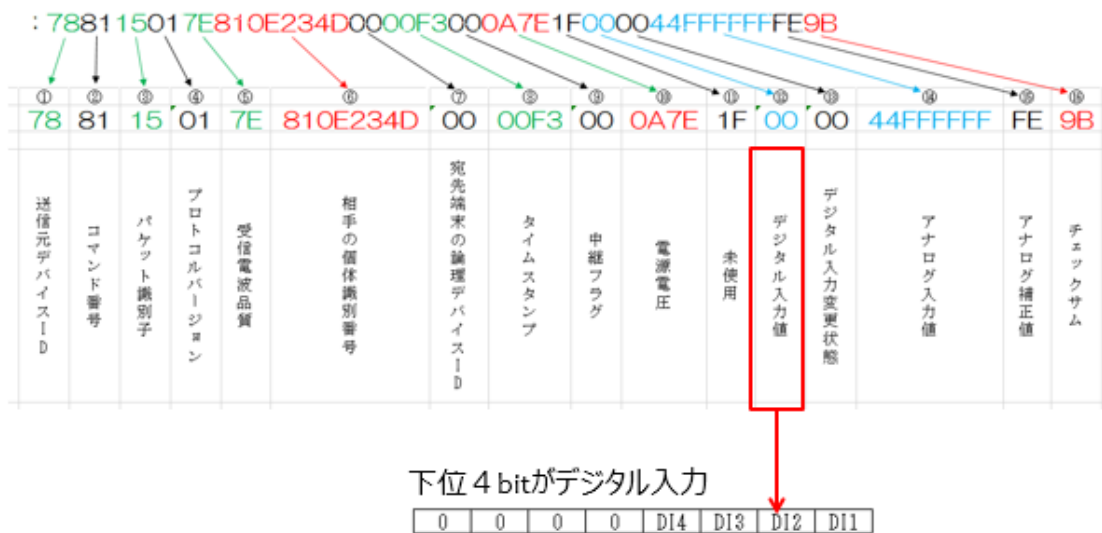
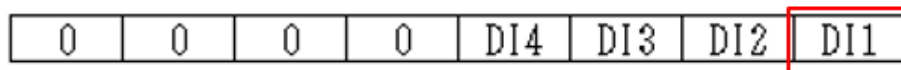


図 155

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の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。



DI1 (bit 0) に SW 1 の状態が現れる。
→ DI1 (bit 0) に SW を接続する。

図 156

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、上の図の 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. PC×1 台（Windows10）
2. Python 開発・実行環境×1 セット

◇子機の配線

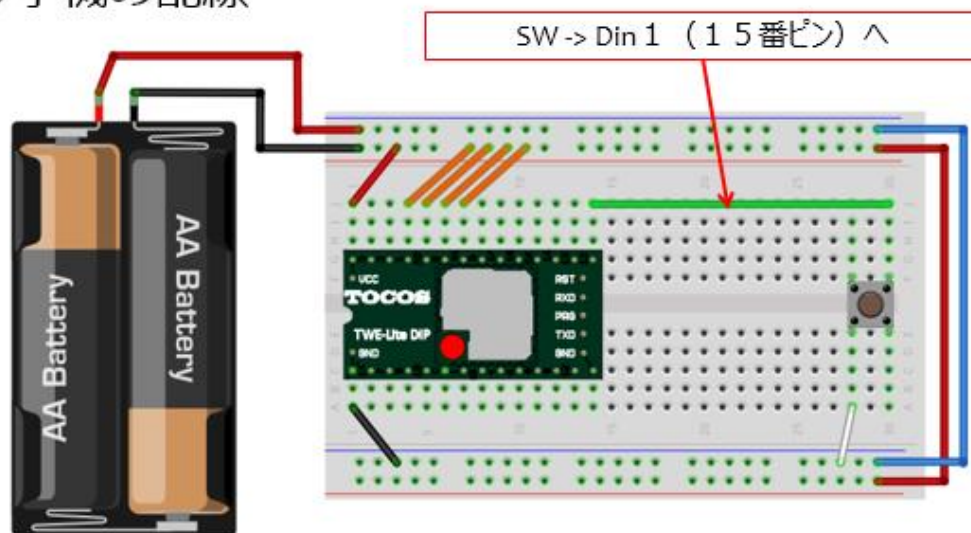


図 157

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）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。実際の子機は次の様になりました。

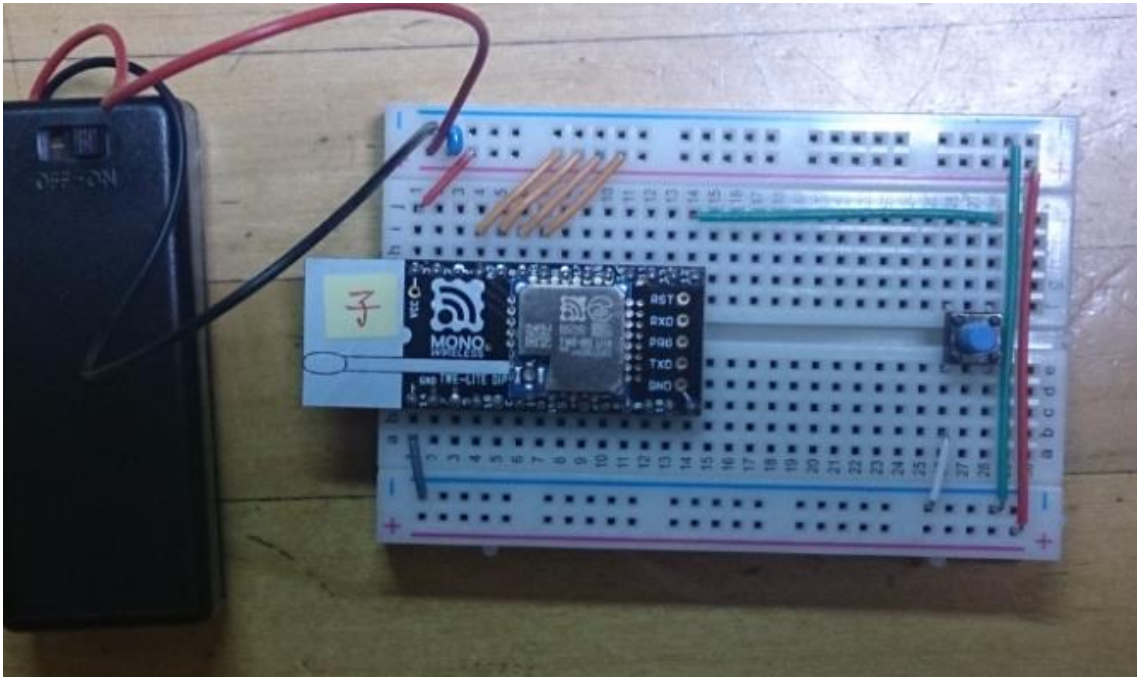


図 158

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(">BBBBBIBHBBBHHBBBBBB") # 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
    "battery" : 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 = denbunkKaiseki(data) # 受信電文を解析して辞書を作る

    sw = parsed["digital"][0]     # デジタル入力の变化を調べる
    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

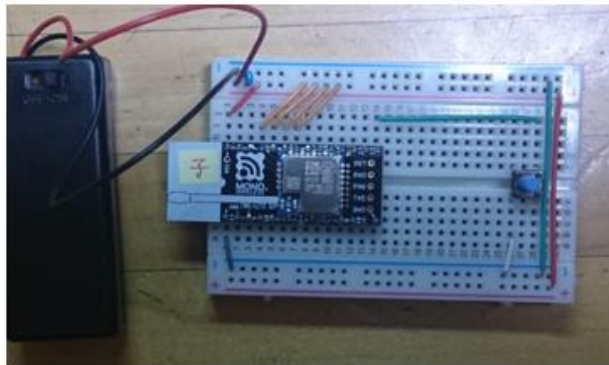


図 159

◇動作確認

- ◇SWをON/OFF. 状態が変化したとき表示.

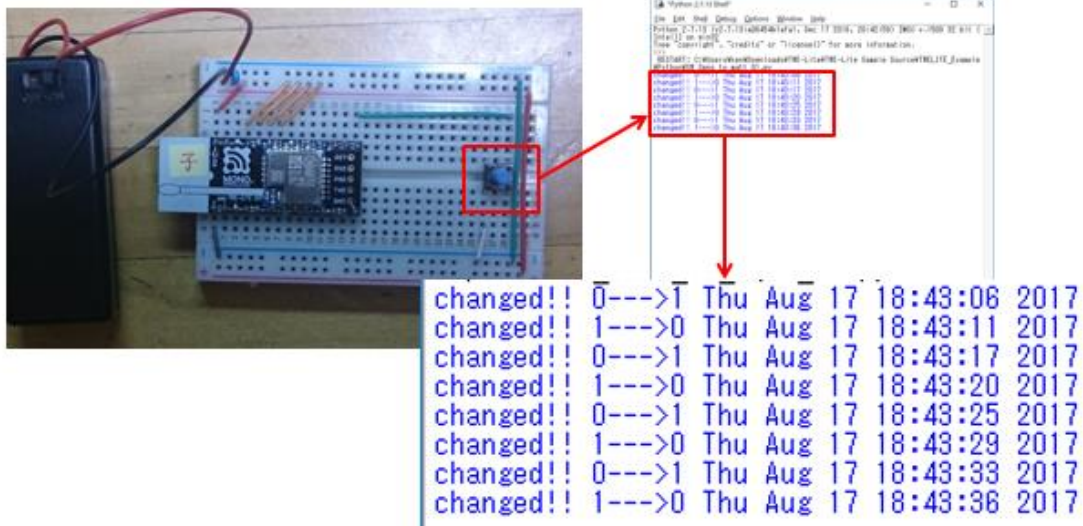


図 160

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。

◇SWの状態を検出。
変化したとき...
◇WEBで通知！！

図 161

SW の状態を検出したとき通知するのは、もう前回、第 12 回で行っています。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。

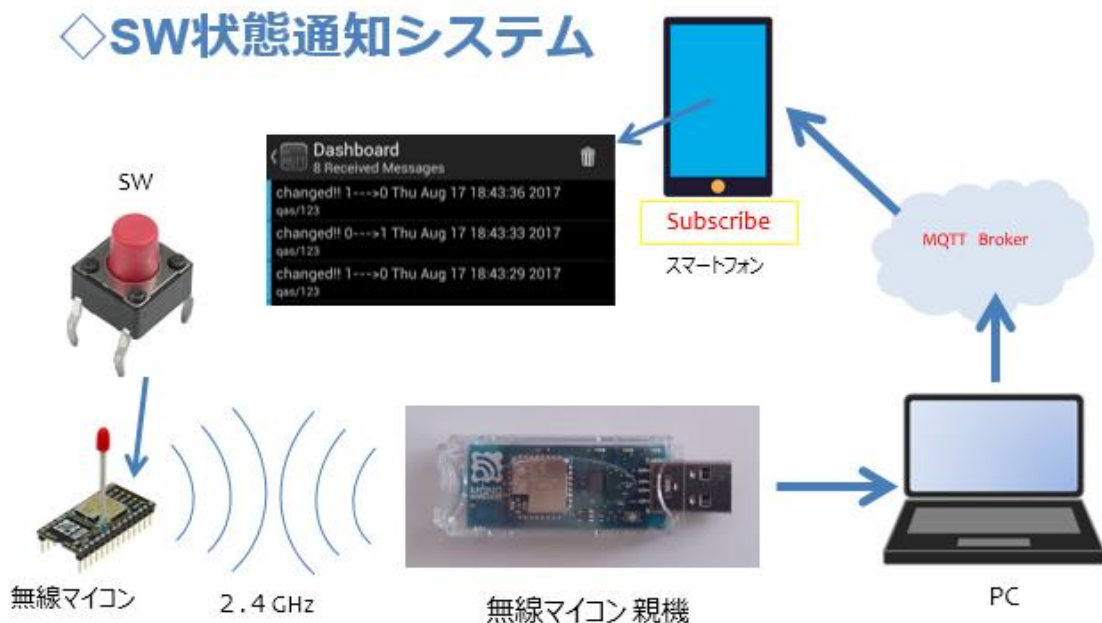


図 162

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。

次の図はシステムの全体構成です。

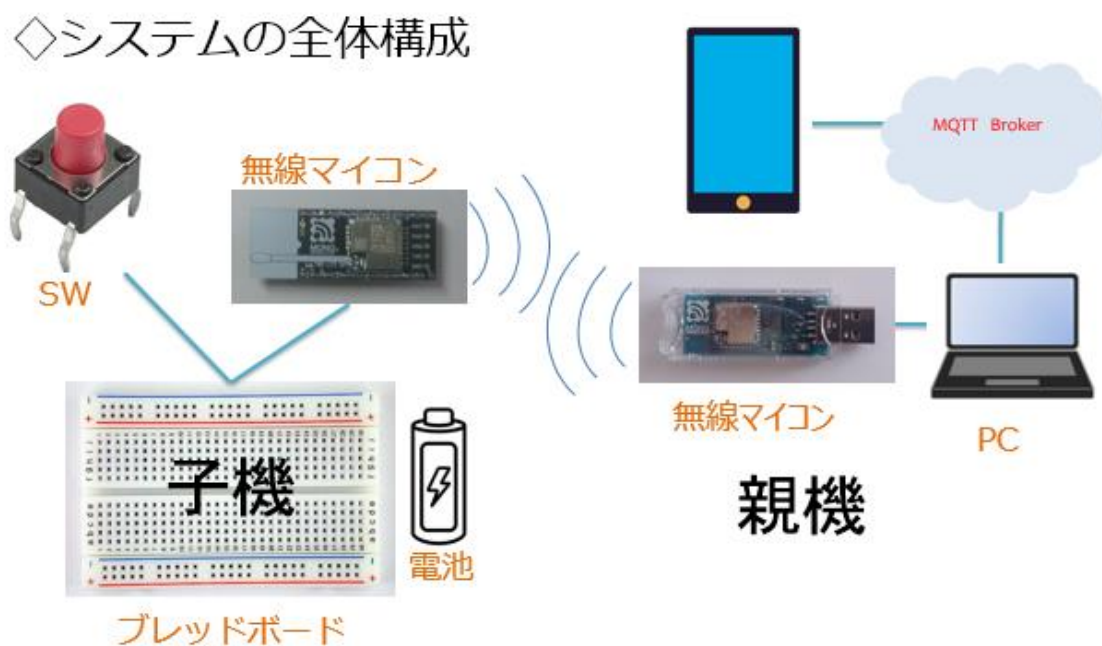


図 163

子機・親機とも第 12 回と同じです。SW の状態変化は PC のウィ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 フォンを使用しました。PC と携帯端末の間に MQTT という WEB サービスを使用します。

◇SWの状態を通知する

- ◇IoTに相応しく.
- ◇遠隔地への通知可能なWebサービス.
- ◇利用容易で無料.

WEBサービス MQTT

※Message Queue Telemetry Transport

図 164

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。

WEBサービス MQTT

◇MQTT : 短いメッセージの発行と購読

※Message Queue Telemetry Transport

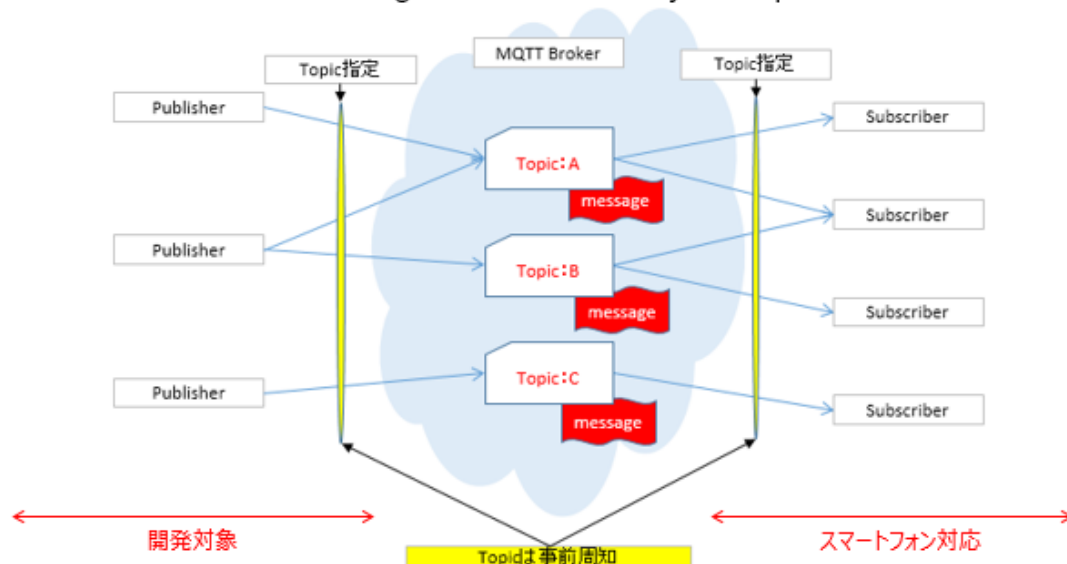


図 165

上の図は 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特定するために 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たい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。そして、無料

で利用できる MQTT Broker があるということは、とてもありがたいことです。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。

◇無料でテストできる MQTT Brokerがある.

1. test.mosquitto.org

2. broker.hivemq.com

図 166

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が見やすい MyMQTT を使用しています。

◇公開MQTTアプリ

1. MQTT Dashboard (Android)
2. MyMQTT (Android)



◇MyMQTT (Android版) を使用する.

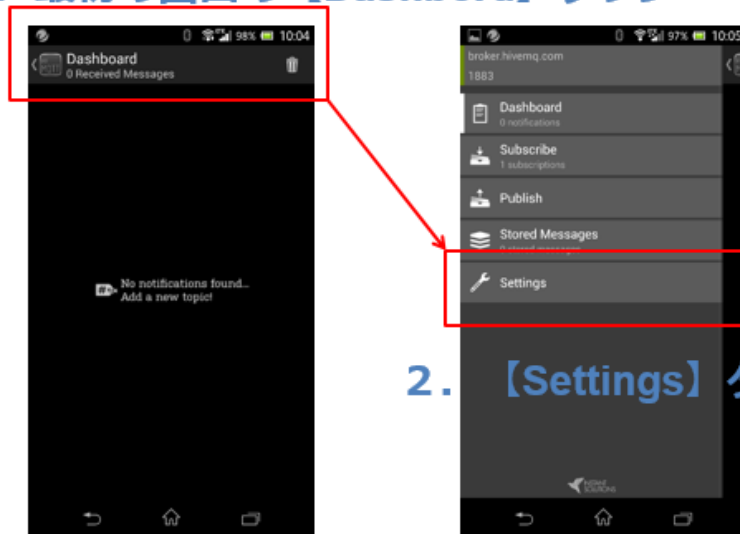
図 167

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るようで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。

MyMQTT (Android版) 使い方

1. 最初の画面の【Dashbord】タップ



2. 【Settings】タップ

図 168

3. Broker を入力、Save をタップ

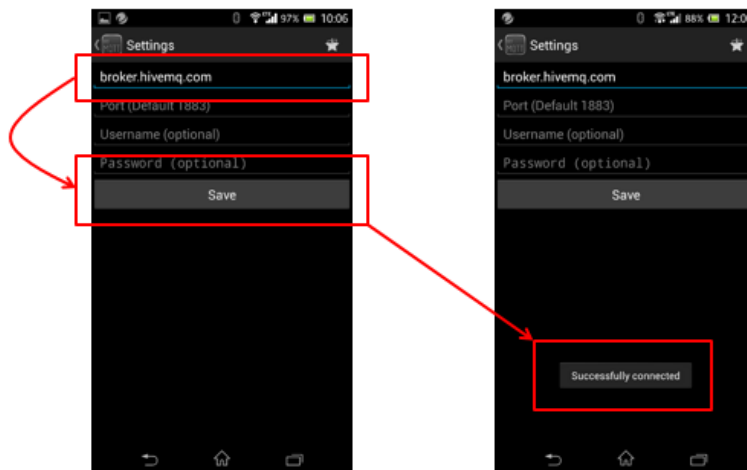


図 169

この時、MQTT Broker に接続が行われます。

4. Settings をタップ

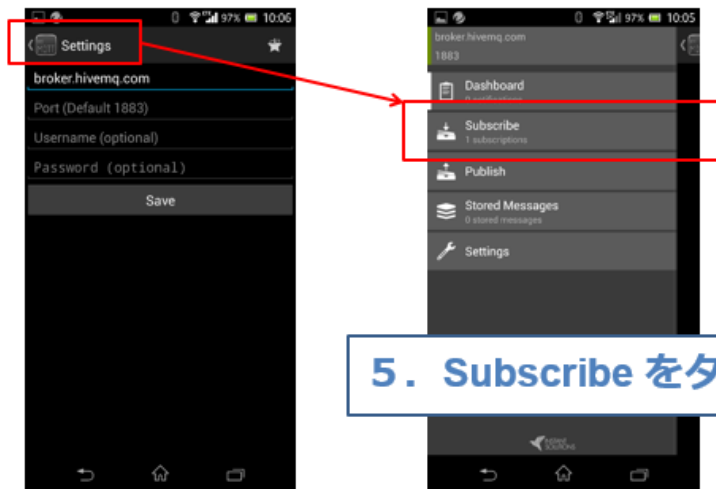


図 170

6. Settings をタップ



図 171

Subscribe する Topic 名を登録します。

9. <Subscribe タップで元に戻り

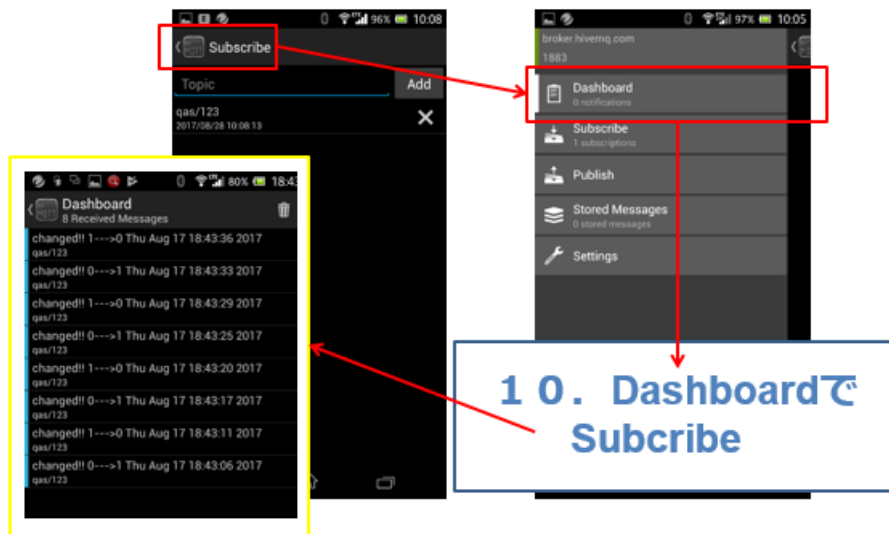


図 172

◇Subscribeの様子

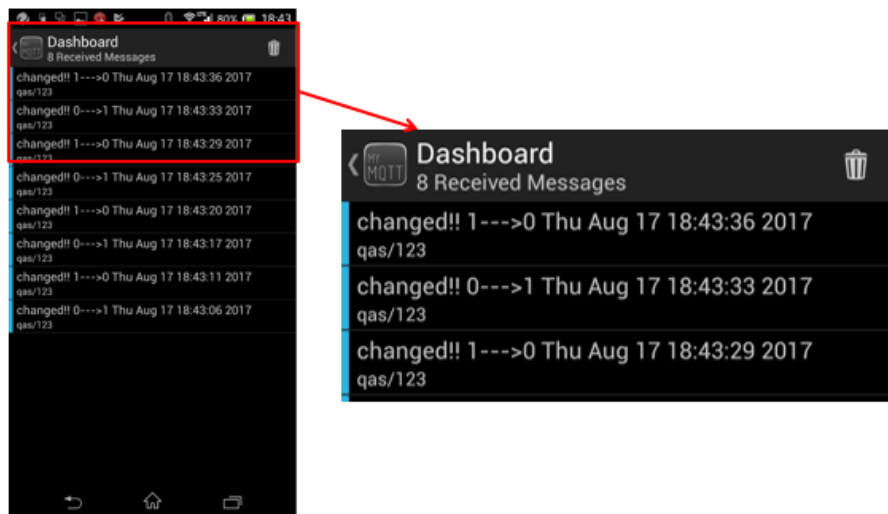


図 173

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

◇子機の配線

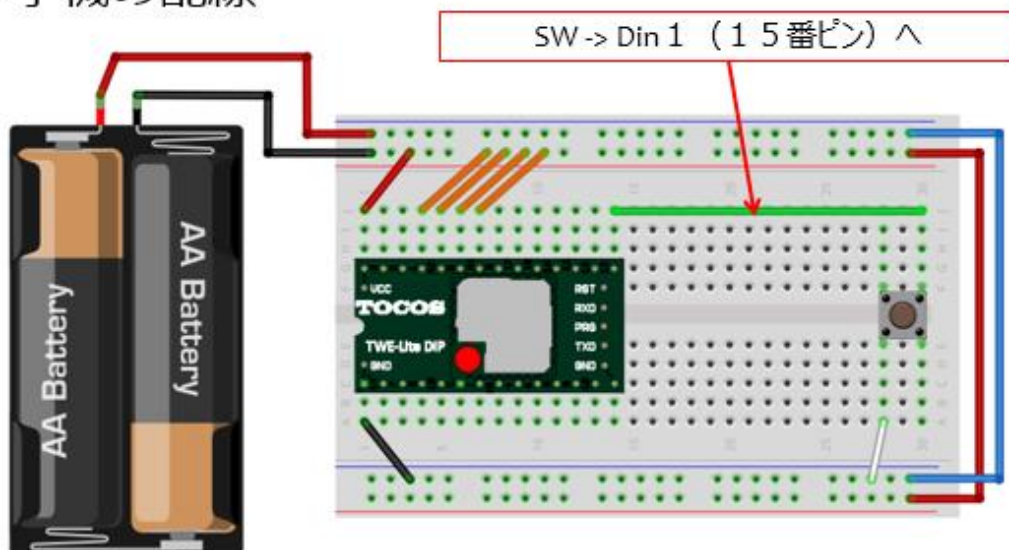


図 174

子機の配線も前回の第12回と同じです。

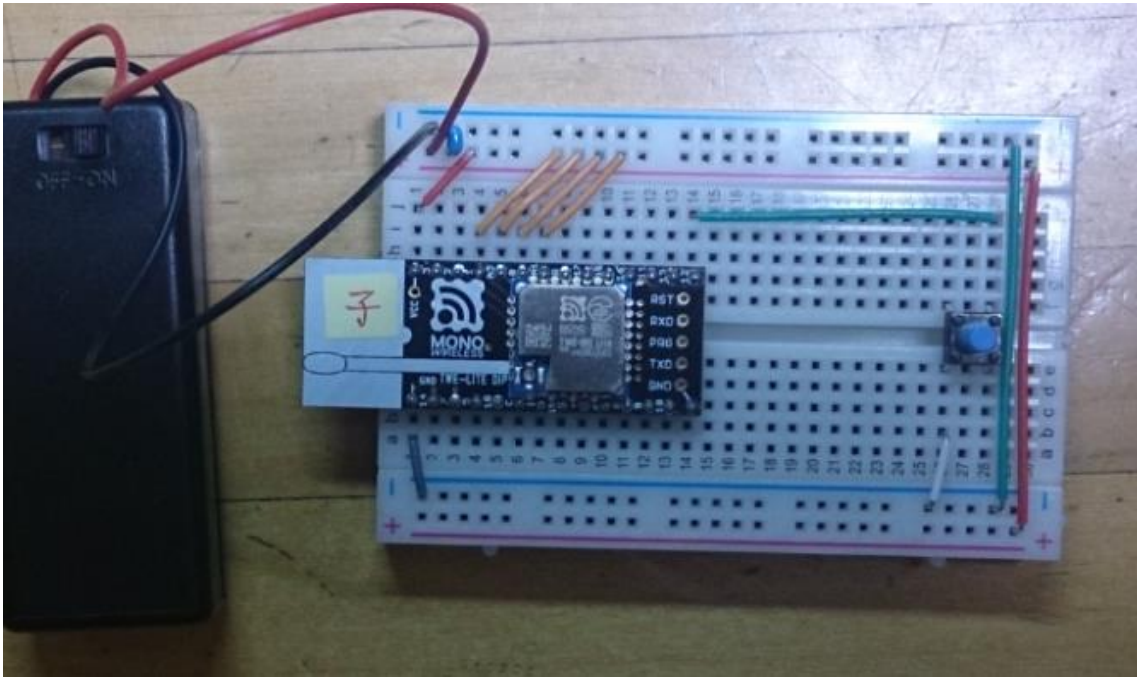
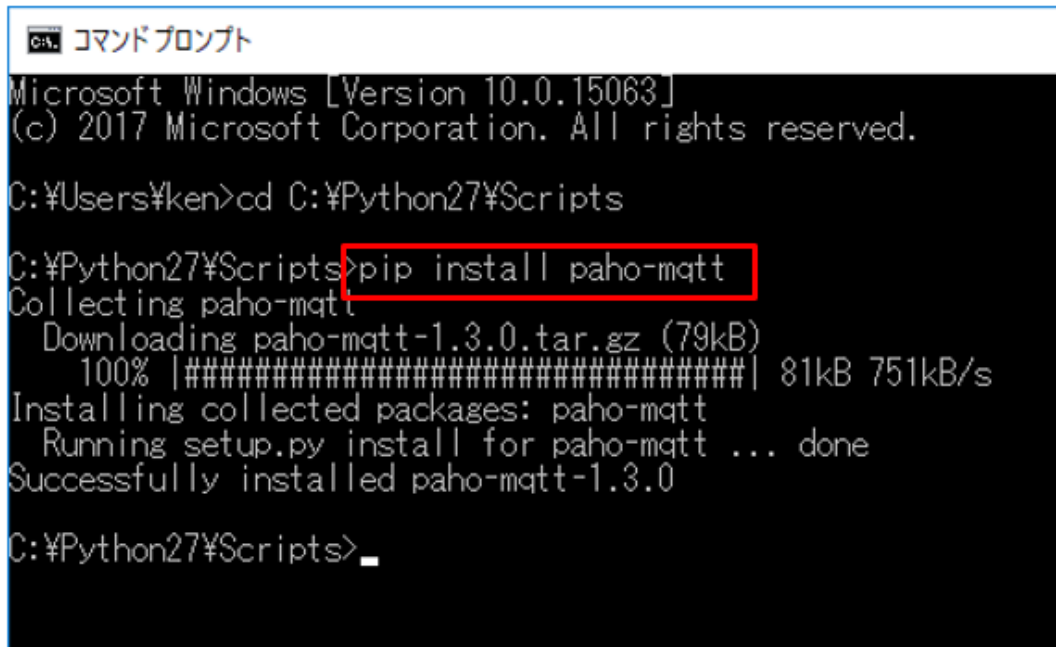


図 175

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。

MQTTパッケージの準備 (Python)



```
コマンドプロンプト
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:¥Users¥ken>cd C:¥Python27¥Scripts

C:¥Python27¥Scripts>pip install paho-mqtt
Collecting paho-mqtt
  Downloading paho-mqtt-1.3.0.tar.gz (79kB)
    100% |#####| 81kB 751kB/s
Installing collected packages: paho-mqtt
  Running setup.py install for paho-mqtt ... done
Successfully installed paho-mqtt-1.3.0

C:¥Python27¥Scripts>_
```

図 176

コマンドプロンプトに上の様に

```
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。この時、**Internet** に接続できる状態にしておいてください。

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(">BBBBBIBHBBBBBBBB") # 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
    "battery" : 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、スマートフォンアプリで 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.

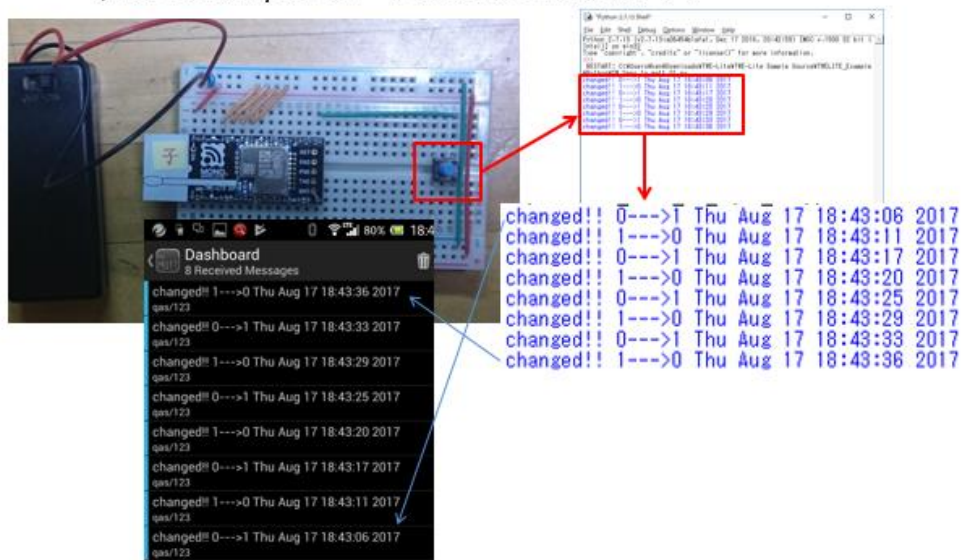


図 177

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、農業などで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、無線温度センサーです。

◇温度センサーで計測.
◇WEBで通知！！

図 178

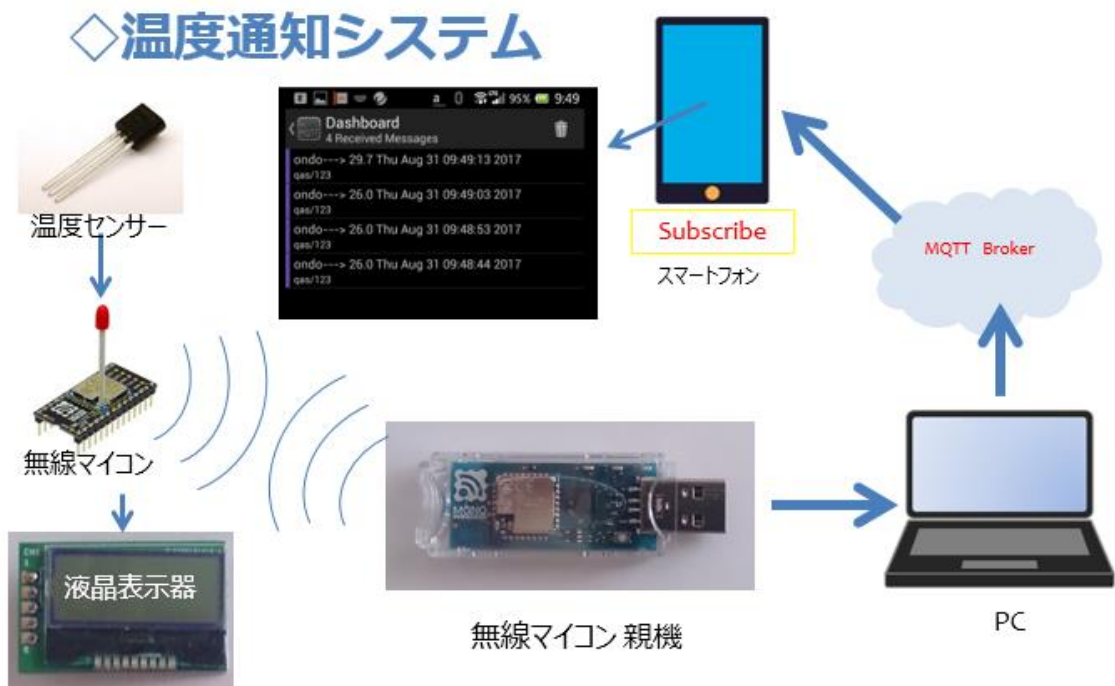


図 179

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を無線通信の電文に載せ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 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。

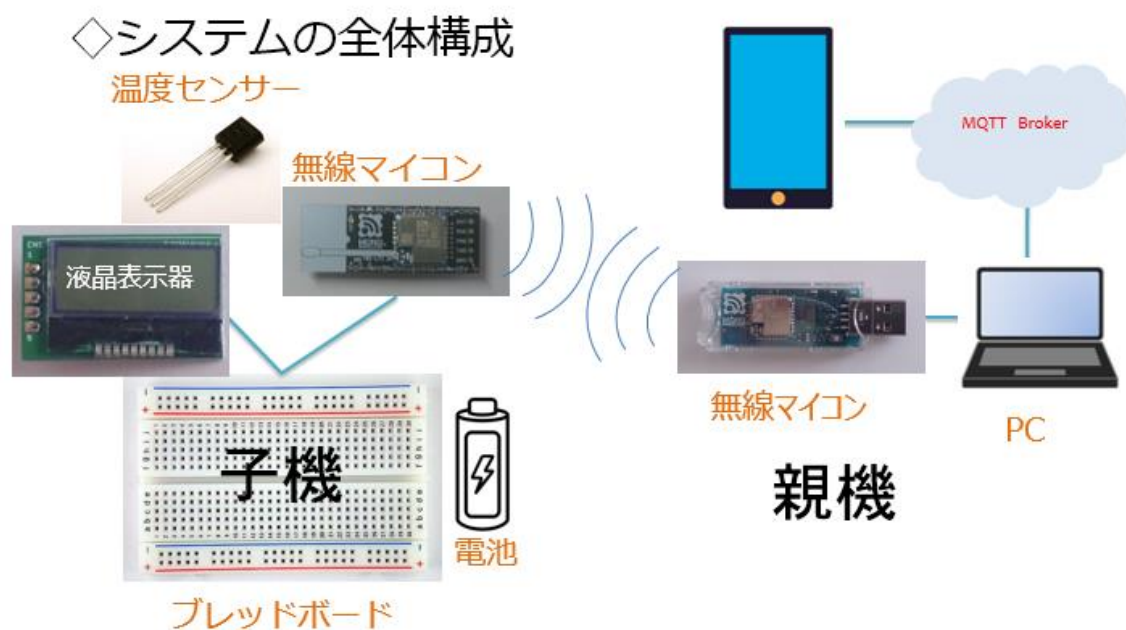


図 180

全体の構成は上の図の様になります。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 回から時間がたってしまった方は、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。

◇無料でテストできる MQTT Brokerがある.

1. test.mosquitto.org
2. broker.hivemq.com

図 181

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。

◇公開MQTTアプリ

1. MQTT Dashboard (Android)
2. MyMQTT (Android)



◇MyMQTT (Android版) を使用する.

図 182

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、下の図の電文に載っ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

: 788115017E810E234D0000F3000A7E1F000044FFFFFFFE9B

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯
78	81	15	01	7E	810E234D	00	00F3	00	0A7E	1F	00	00	44FFFFFF	FE	9B
送信元デバイスID	コマンド番号	パケット識別子	プロトコルバージョン	受信電波品質	相手の個体識別番号	宛先端末の論理デバイスID	タイムスタンプ	中継フラグ	電源電圧	未使用	デジタル入力値	デジタル入力変更状態	アナログ入力値	アナログ補正値	チェックサム

図 183

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

: 7888012200001030313233343536374142434445464748X

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯
78	88	01	22	00	00	10	3031323334353637	4142434445464748	X						
送信元デバイスID	コマンド番号	コマンド種別	I2Cアドレス	I2Cコマンド	未使用	データサイズ	データ	データ	チェックサム						

型式コード
22 : AQM0802A

データサイズは、上下段合わせたサイズ
LCD上段 : 0 1 2 3 4 5 6 7
LCD下段 : A B C D E F G H

図 184

子機の液晶表示器への表示は上の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. PC×1 台（Windows10 + Internet 接続）
2. Python 開発・実行環境×1 セット

◇子機の配線

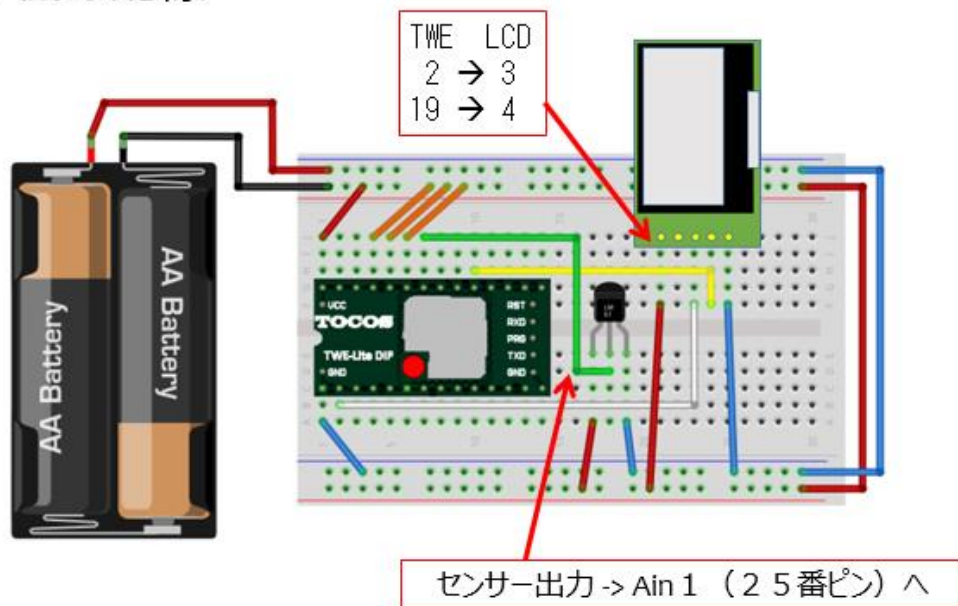


図 185

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。

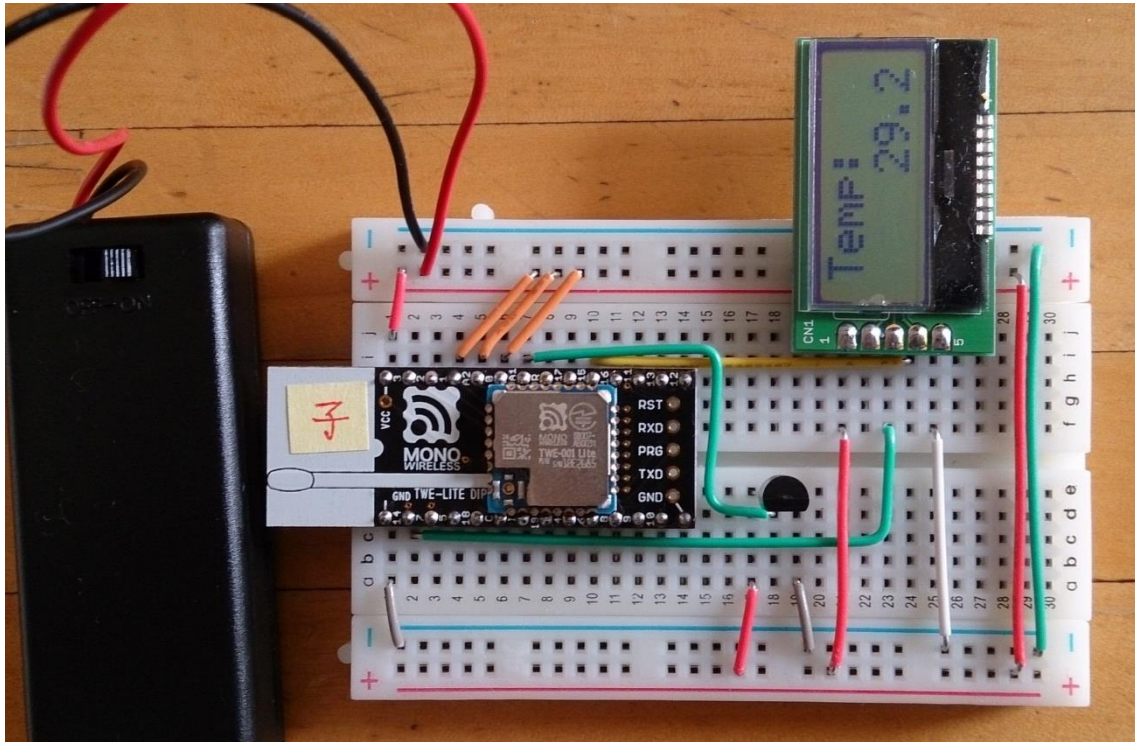


図 186

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(">BBBBBIBHBHBBBBBBBB") # 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 = 'gas/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。
-

最後に

最後まで読み進めていただき、ありがとうございます。すべての講座を実習された方は、もう WEB を利用した連携の入門編は身に付きました。次はぜひ第二分冊へと進んでください。第二分冊では、このテキストで使用したものとは別の無線マイコンモジュールを使い、直接 Internet に接続して、いろいろな WEB サービスを使用してみます。第二分冊を終了するころには、現場で使える IoT の WEB 連携技術が身に付いているでしょう。

2017.10.6

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
(IoT 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
