令和2年度「専修学校による地域産業中核的人材養成事業」
（Society5.0等対応カリキュラムの開発・実証）

富山県をモデルとした「モノづくり」現場に IoT を導入する中核的人材育成

＜ 製造IoT応用演習 テキスト教材 ＞

本紙は、文部科学省の生涯学習振興事業委託費による委託事業として学校法人浦山学園富山情報ビジネス専門学校が実施した令和２年度「専修学校による地域産業中核的人材養成事業」の成果物です。

学校法人浦山学園 富山情報ビジネス専門学校

このテキストについて

このテキストでは、とてもコンパクトなマイコンを用いた、ミニマムモデルの工場【μ-Factory】の開発手順を解説しています。μ-Factoryでは、サーボモータを駆動して振動が発生します。その振動を測るセンシングシステムも開発し、両方を同時に稼働させて計測値を行います。センサが得た情報を蓄積するData Lake機能も開発します。これらの教材を良く理解して実習すれば、現場での情報収集システムが自力で開発できるようになるでしょう。

製造 IoT 応用テキスト

目次

第1章 生産現場が求めているIoT

IoTでやりたいこと

◇独立している制御システムをIoTによってスマート工場化すれば、個々の分析だけでは見出しにくい、ラインや工場全体の適正が把握しやすくなり、大きな効果が期待されている

◇しかし、IoTを活用した制御システムでは、セキュリティに対する対応策がまだ明確ではなく、次のような現状であることを、まず理解しておく必要がある

①. 決定版というものがない
②. セキュリティ対策を満足するレベルで行うには費用が大きい
③. 対策費用対効果を考えた場合、現状レベルで十分という考えが大きい

◇この講座では、IoTによる制御も実習するが、主としてIoTによる情報収集に重点を置く

◇モノを製造している現場で、IoTを利用して収集したい情報は、次のようなことである

①. 製品に係る情報を得ること
②. 設備に係る情報を得ること
③. 人に係る情報を得ること

1

図 1-1

情報を得るには

◇IoT利用の目的は、大きな費用を掛けずに実施できるという点が重要である
→ 費用を掛ければ、どのようなことでも実現できる時代になっている

◇情報収集システムを素早く構築して稼働させるためには、ネットワークに接続できるマイコンの利用が必須である
→ ネットワーク機能を持つマイコンは、表示器やバッテリを内蔵したものを安価に入手できる

◇図に示すのは、M5StickC（M5Stick-Cとハイフンを付ける場合もある）というマイコンである
・80mAhリチウムイオン電池内蔵
・フルカラーTFT液晶
・WiFi・Bluetooth
・ SW内蔵
・I2C、SPI による外部デバイス拡張
・6軸（加速度＋ジャイロ）センサ内蔵
・240MHz ESP32 CPU
・2000円以下で入手可能

◇内蔵センサ、外部拡張センサにより情報収集し、WiFiあるいはBluetoothでホストに情報を送り記録する
→ このような処理を行うマイコンシステムをIoTデバイスと呼ぶことにする

2

図 1-2

収集できる情報

◇IoTデバイスを用いて収集できる情報には次のようなものがある

①. 製品に係る情報
→ 数量や品質（大きさ・重さ・表面状態・・・etc）

②. 設備に係る情報
→ 稼働時間や状態（稼働・停止・故障）

③. 人に係る情報
→ 人の作業時間や作業内容

◇ IoTデバイスで情報を得るために利用できるセンサは、とても多くの種類があり、内蔵センサや外部取付のモノを使う ※以下に3例を挙げる

①. SW → 人が設備を稼働・停止・故障で停止の際に押せば、設備の稼働状態と稼働時間が分かる
また、製造する製品の通過する場所にSWを取り付ければ、製造数量が把握できる
設備稼働状態と時間、製造数量が分かれば、設備稼働率や生産効率も把握できるので、
SWは大きな役割を果たす

②. 加速度センサ → 金属加工設備では、振動が製品精度に与える影響が大きいが、紹介した
M5StickCには加速度センサとジャイロセンサが6軸センサとして内蔵されている

③. カラーセンサ → 光の三原色の波長帯と白に感度を持つセンサを1チップにし、色を検出するもの
試料に充てた白色光の反射光を検出して、表面加工状態を検出することもできる

3

図 1-3

第2章 利用されている設備

利用されている設備　旋盤

◇工場では様々な設備が稼働している（詳細は機械工作法などのテキストを参照のこと）
　それらの設備は、ほぼすべてが主動力に電動機（モータ）を用いている
◇ここでは、金属加工に用いられる設備をいくつか取り上げる

①. 旋盤　→　工作物（ワーク）を回転させて、バイトと呼ばれる刃物でその表面を削ったり、穴あけ加工
　ねじ山加工などを行う

・小型の卓上旋盤や、コンピュータ制御のNC旋盤などもある
・右図は正面旋盤（大型のワーク加工に用いる）
・中図は、バイトの例（写真左側が刃先）
・左図は、刃物部分だけが交換できるスローアウェイ
　チップを使用するバイト

4

図 2-1

利用されている設備　フライス盤

②. フライス盤　→　工作物は固定して、刃物を回転させてその表面の切削や穴あけ加工を行う

・小型の卓上旋盤や、コンピュータ制御のNC旋盤もある
・右図は、縦型フライス盤
・左図は、フライス（刃物）の例
　これをフライス盤の主軸に取り付けて使用する

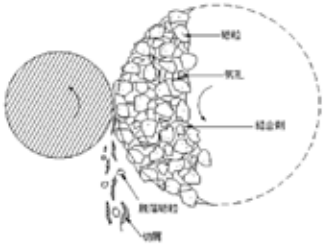
5

図 2-2

利用されている設備　研削盤

③. 研削盤　→　回転する砥石でワークの表面を削る工作機械で大きく分けて次の3種類がある
- ・円筒研削盤　→　円筒素材の外側を削る
- ・内面研削盤　→　円筒素材の内側を削る
- ・平面研削盤　→　平らな平面を削る

・右図は、平面研削盤
・左図は、砥石の例
・　結合剤で成形されている砥粒が、研削加工を行う際に脱落して、常に新しい刃物（砥粒）が現れる



6

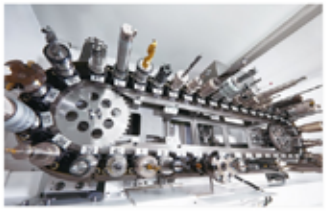
図 2-3

利用されている設備　マシニングセンタ

④. マシニングセンタ　→　複数の刃物を自動で交換できる装置を持ち、NCのプログラミング制御に従って穴開けや平面削りなどを1台でこなせる機械

・日本工業規格（JIS）では工作物の取り付け替えなしに、多種類の加工を行うNC工作機械と定義機械の構造によって主軸が水平の横型マシニング、垂直の立型マシニング、門型構造の門型マシニングなどがある

・右図は縦型マシニングセンタの例
・下図は多くの刃物を保持する複雑なツールマガジンの例



7

図 2-4

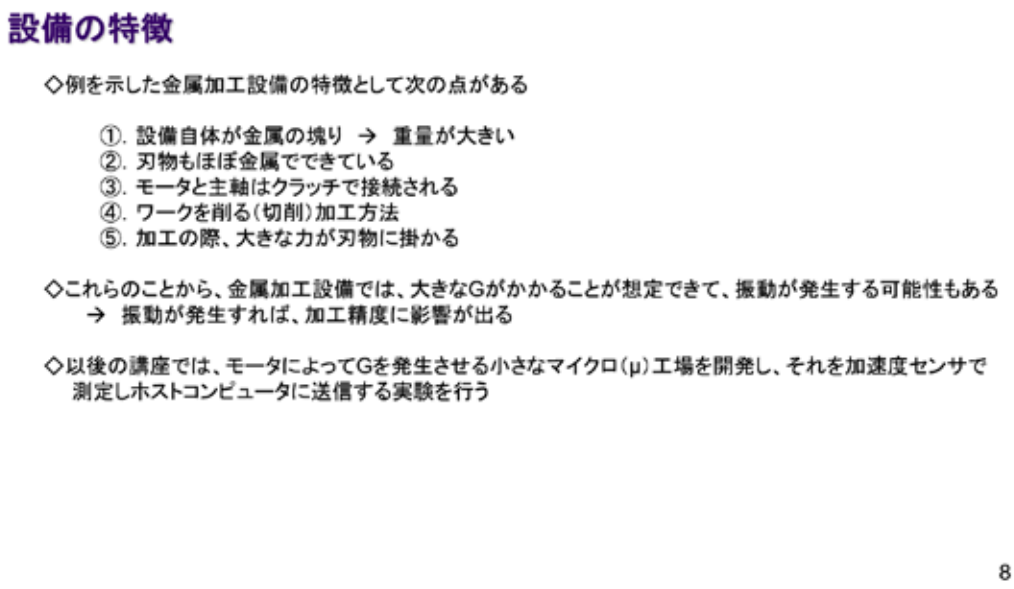


図 2-5

このテキストでは、開発体験型教材として実習キットを用いた、ミニマム工場モデルを通じて、IoT によるスマート工場構築への足場づくりを行っている。

図 2-6

第3章 マイクロ工場（μ Factory）の開発

3．1 実習で用いるマイコン

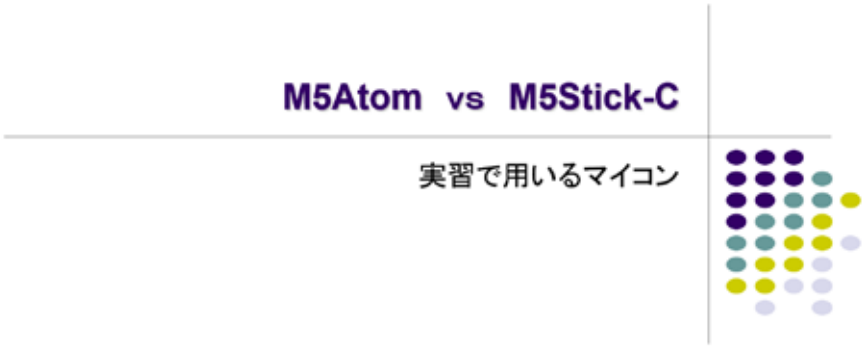


図 3-1

これからしばらくの間、IoT を実現するために必要なマイコンの開発実習を行う。ここでは 2 種類のマイコンを使用する。そのマイコンの概要について以下説明し、開発環境の構築も行う。

図 3-2

実習で使用するのは図に示す 2 種類のマイコンで、いずれも大変コンパクトである。今回は制御とセンシングを同時に行うにあたり別の種類のマイコンを用いた。

図 3-3

M5Atom は、24mm 角とたいへん小さいが、32bit の WiFi・Bluetooth 機能を搭載している。残念ながらバッテリを内蔵していないので外部電源による駆動が必要である。このテキストではこのマイコンを制御用として用いる。

図 3-4

M5Stick-C（M5StickC と書く場合もある）は、M5Atom に LiPo バッテリを加え、フルカラーLED に代わって、フルカラーTFT 液晶を備えたものになっている。このテキストでは、このマイコンをセンシング用として用いる。

CPU ESP32-PICO-D4

◇M5ATOM、M5Stick-Cはいずれも同じCPU ESP32-PICO-D4チップを使用している
◇CPUメーカから下に示すデータシートが公開されている （右は目次の一部）

Contents

目次

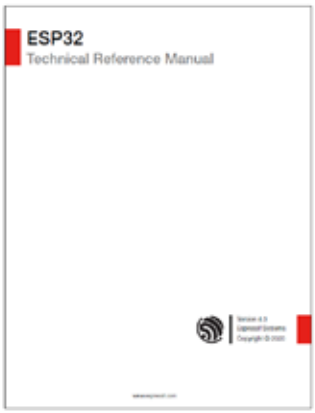
1. 概要
2. ピン配置
3. 機能説明
4. 入出力とセンサ
5. 電気的特性
・・・etc

4

図 3-5

CPU ESP32

◇ESP32-PICO-D4チップの詳細は、このチップのベースになった【ESP32を参照せよ】と記述がある
※詳細は【ESP32 Technical Reference Manual】を参照する（右は目次の一部、700頁以上ある）



5

図 3-6

図 3-7

M5Atom、M5Stick-C は、同じ開発環境が利用できる。このテキストでは、Arduino IDE を用いたシステム開発を説明する。

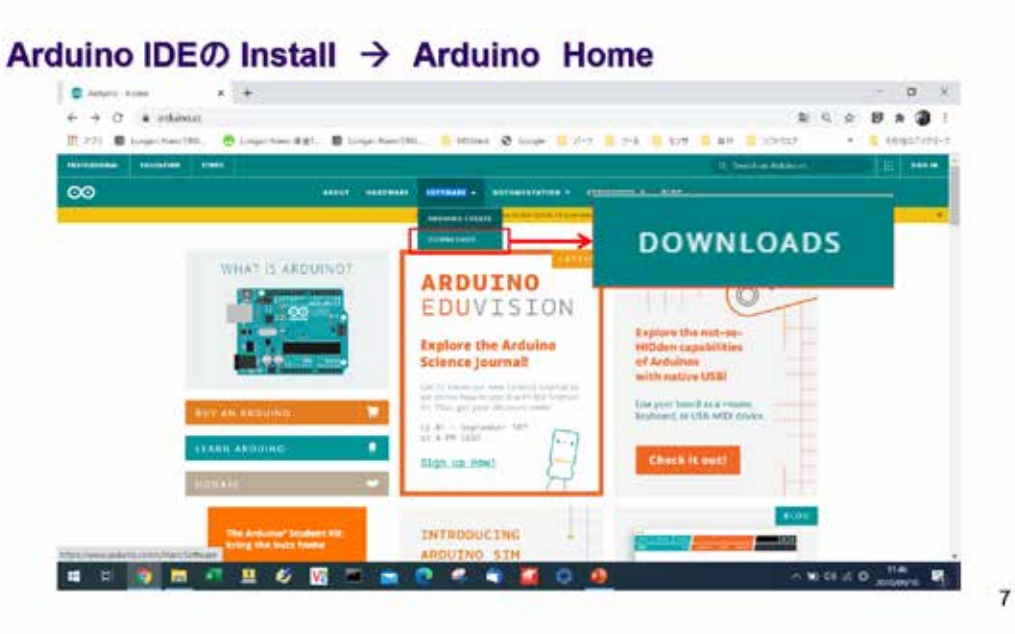


図 3-8

Arduino IDE は、WEB から誰でもダウンロードができて、インストールも容易である。WEB で【Arduino】を検索すると、Arduino Home（上図）がヒットする。

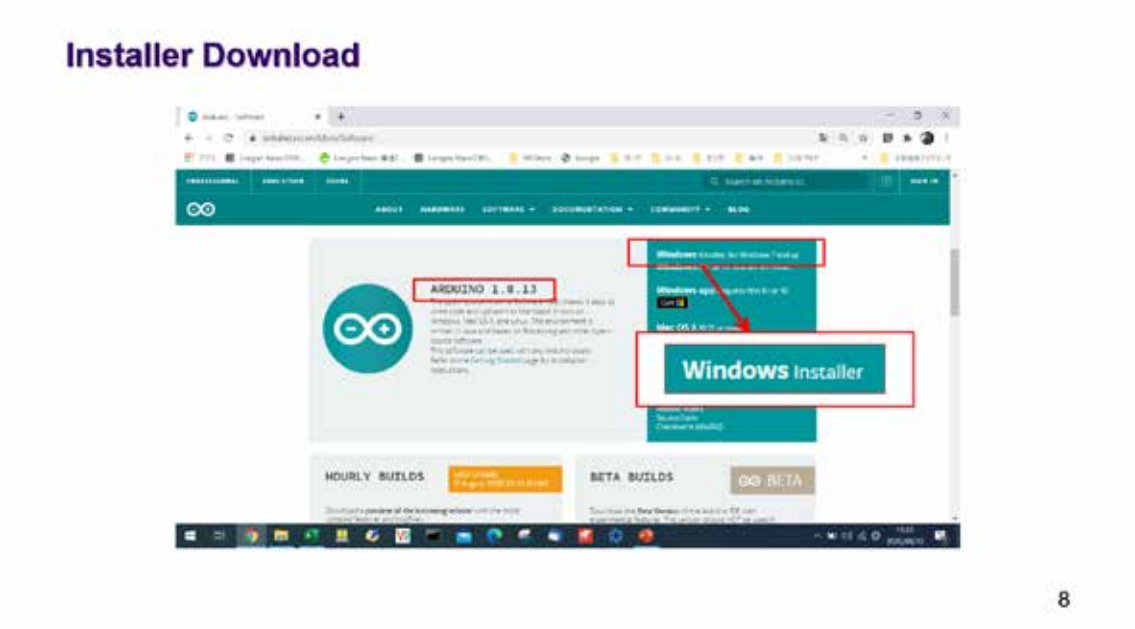


図 3-9

そのページから Windows Installer をダウンロードする（執筆時点のバージョンは、Ver.1.8.13)。ダウンロードしたファイルを実行して、表示されるメッセージには、「すべてをインストール」を選択すること。そうしないと、USB シリアルドライバがインストールされないことがある。

図 3-10

インストールが完了すると、上図に示す眼鏡マークのアイコンがデスクトップに作られる。IDE を起動すると、図右に示すウインドウが開く。中央の白い部分にソースコードを記述して、プログラムを開発する。

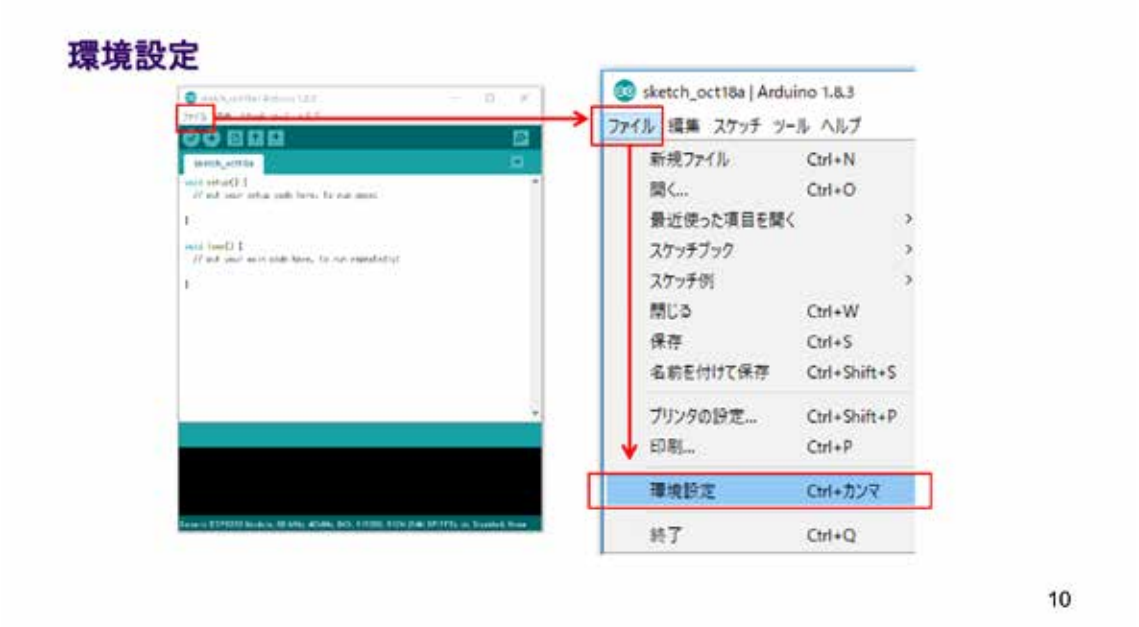


図 3-11

開発前に環境設定が必要である。図のように【環境設定】を選択する。

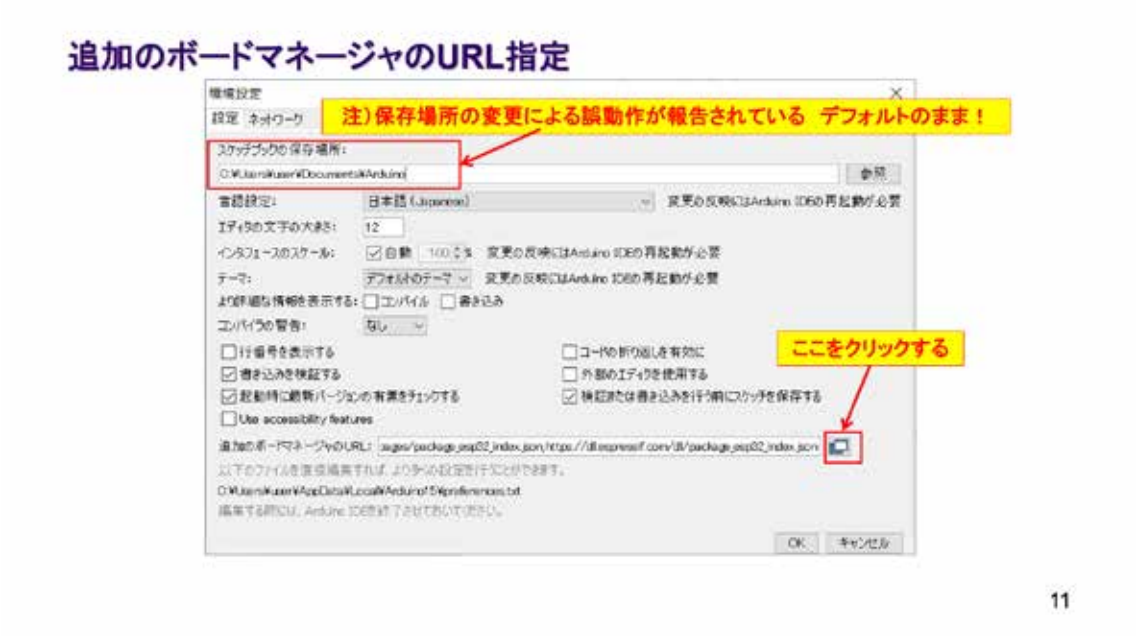


図 3-12

開いたウインドウで、追加のボードマネージャの URL の右にあるアイコンをクリックする。

図 3-13

開いたウインドウに、上図の URL を追加する。これはマイコンを開発したメーカが提供している専用のボードマネージャが登録されている URL である。IDE は、この URL から利用できるボードマネージャの一覧を取得する。

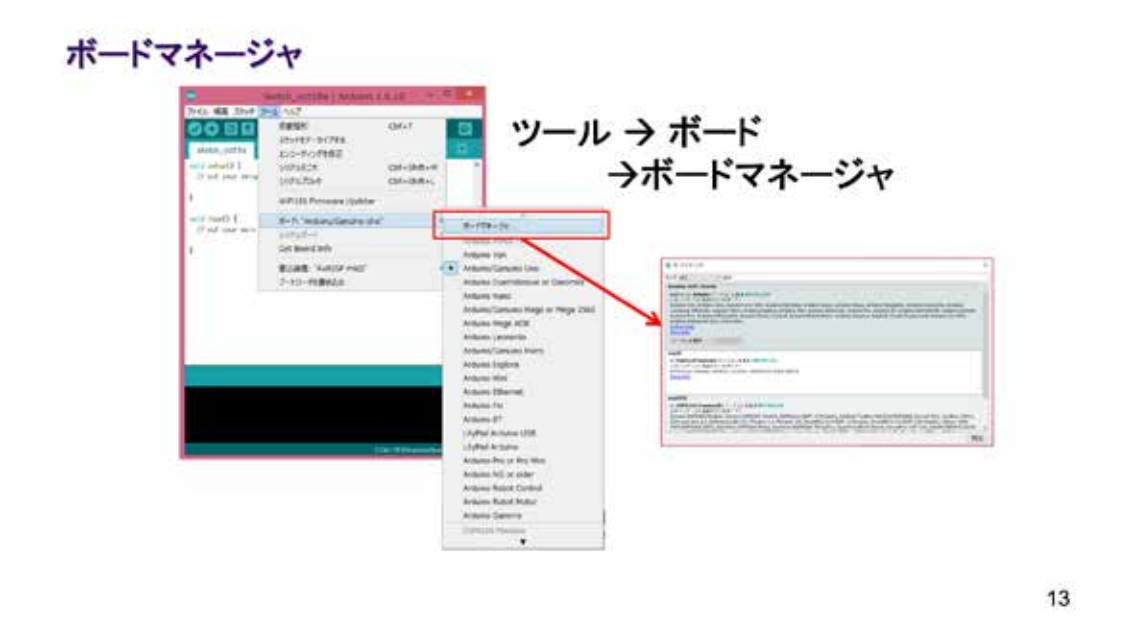


図 3-14

使用するマイコン（ここではマイコンボードと表現している）用のマネージャをインストールする。上図に従いボードマネージャを開く。

図 3-15

検索ボックスに ESP と入力して少し待つと、関連するボードマネージャが一覧表示されるので、esp323 by Espressif Systems を選択してインストールする。

図 3-16

次に対象マイコン用のライブラリをインストールする。図の手順で【ライブラリを管理】を選択する。

図 3-17

ライブラリマネージャが開くので、図のように【M5Atom】【FastLED】【M5StickC】を検索してインストールする。順番は関係ない。

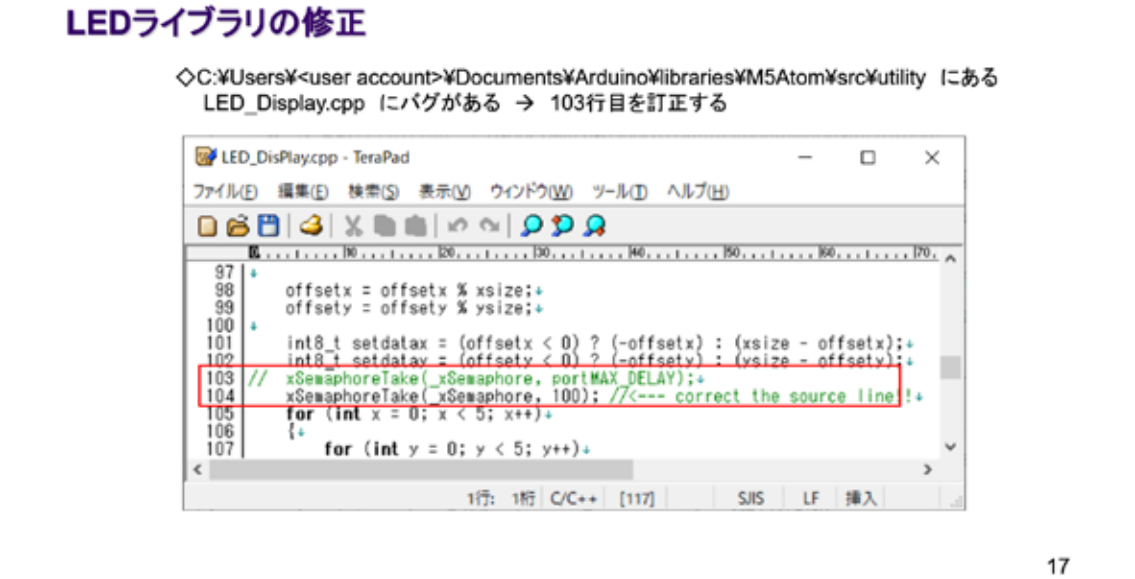


図 3-18

FastLED ライブラリは M5Atom が内蔵しているフルカラーLED マトリクス用のライブラリだが、バグが報告されているので、修正を行う。図に示すフォルダのライブラリファイルをエディタソフトで開き、ソースコードを訂正する。上書き保存することを忘れないように。

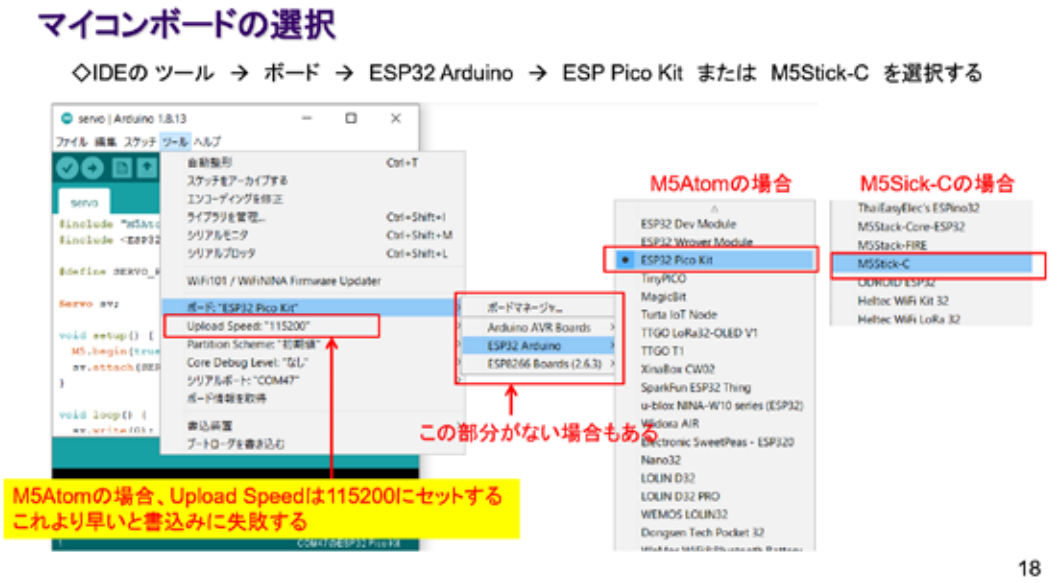


図 3-19

開発対象のマイコンボードを選択する。このテキストでは、2 種類のマイコンを用いるが、ソースコードを入力する前に、該当するマイコンボードを選択しておく。

図 3-20

IDE のインストールがうまく行われていれば、USB シリアルドライバがインストールされているはずなので、マイコンを認識するかどうか事前に確かめておく。マイコンを USB ケーブルで PC と接続して、COM ポートが認識されることを確かめる。

３．２ M5Atom ー LED

図 3-21

いよいよ、マイコンを使った実習に入る。最初に小さなマイコン M5Atom の LED 点滅制御を行う。M5Atom に 5×5＝25 個内蔵されている LED は、一つずつ個別のマイコンで制御されており、M5Atom 本体のマイコンと I2C I/F で接続されている。水平方向と垂直方向に LED のアドレスを指定して個別の制御が行えるので、中図のようにパターンを表現することもできる。フォントデータを用意すれば、字幕スクロールを行うことができる。小さなマイコンユニットは、情報を外部に伝える表現能力が低いが、このような LED マトリクスがあると、相当な情報を外部に伝えることができ、人も理解しやすい。LED チップのデータシートが公開されているので、次頁に直訳と合わせて掲載する。

LEDチップ(WS2812C-2020)データシート

General description

WS2812C-2020 is an intelligent control LED light source, its exterior adopts the latest MOLDING packaging technology, the control circuit and RGB chips are integrated in a package of **2020** component. Its internal includes intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a voltage programmable constant current control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

RESET time **>280μs** , it won't cause wrong reset while interruption, it supports the lower frequency and inexpensive MCU.

Refresh Frequency updates to **2KHz**, Low Frame Frequency and No Flicker appear in HD Video Camera, it improve excellent display effect.

LED with low driving voltage, environmental protection and energy saving, high brightness, large scattering angle, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

2

図 3-22

概要説明 General description

◇WS2812C-2020はインテリジェント(高機能)制御LED光源で、その外装は最近のモールディングパッケージ技術を採用し、制御回路とRGBチップは2020コンポーネントのパッケージに集積されている。
◇高機能デジタルポートのデータラッチと信号整形アンプ駆動回路を内蔵している。
◇ピクセルが示す光色の高度な調和を効果的に確実にする内部発信回路と、電圧プログラム可能な電流制御部も含んでいる。
◇データ転送プロトコルは、単独のNZR通信モードを用いている。
◇ピクセルパワーオンリセット後、DIN(Digital IN)ポートはコントローラからデータを受け取り、1つ目のピクセルは最初の24bitデータを集めて、そのデータをラッチ(確保)し、内部信号再整形増幅回路により再整形される他のデータは、DO(Digital OUT)ポートを通じて、次の段のピクセルに送られる。
◇各ピクセル送信後、信号を24bitに減少する。
◇ピクセルは自動的に再整形送信技術を採用し、接続段数の作りは信号送信速度にだけ依存して信号送信を制限しない
◇ 280μsを超えるリセット時間は、割込みの間の誤ったリセットは引き起こさず、低周波数で低価格なMCUをサポートする
◇リフレッシュ周期は2KHzで更新され、HDビデオカメラでもちらつきが現れず、優れた表示効果の改善をしている
◇低駆動電圧、環境保護そしてエネルギー保護、高輝度、大きな散乱角度、良好な整合性、低電力、長寿命かつ他の優位性を伴うLED
◇上記LED中に集積された制御チップは、シンプルで小型、設置が容易な回路になっている

3

図 3-23

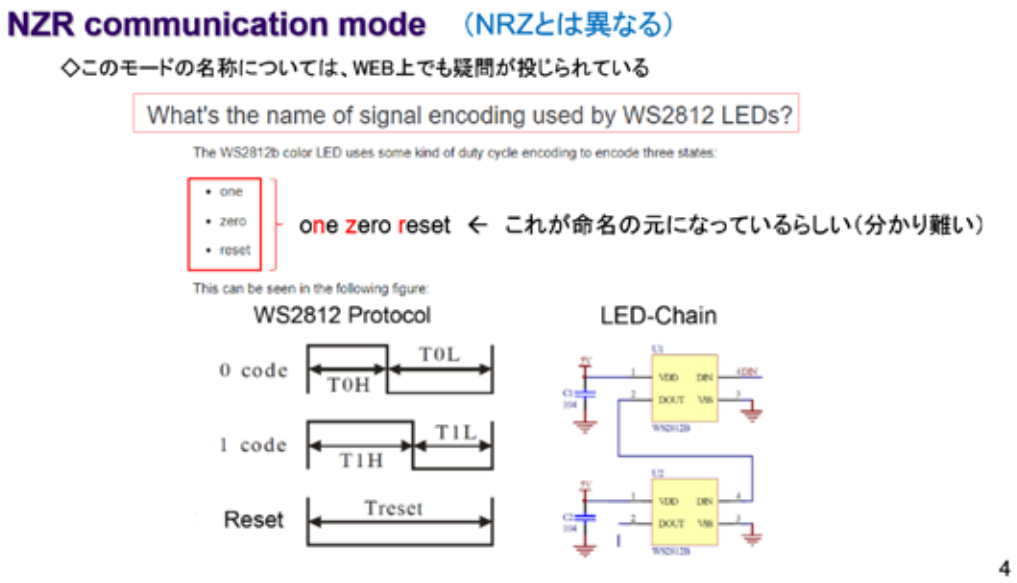


図 3-24

データシート中【NZR】という用語が用いられている。その説明を図に示す。よく NRZ（Non Return to Zero：ゼロに戻らない信号のこと）という用語が使われるが、これとは違うことを示している。

以下、データシートの一部抜粋を掲載する。

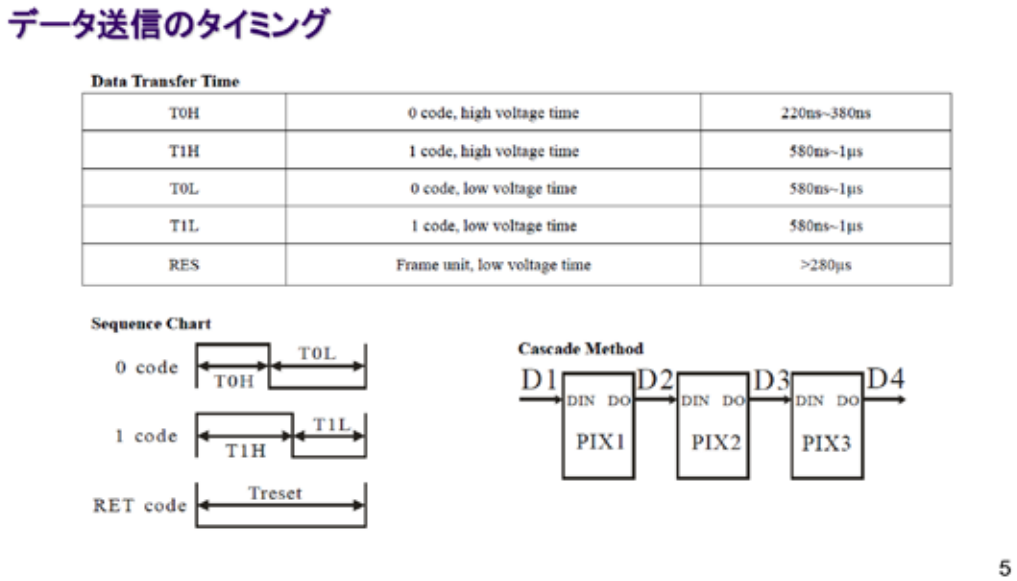


図 3-25

データ送信方法　Data Transmission Method

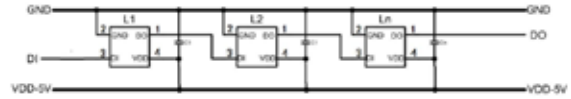
Note: The data of D1 is send by MCU, and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Composition of 24bit Data

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Note: Data transmit in order of GRB, high bit data at first.

Typical Application Circuit



Remarks: C1 is the filter capacitor for VDD, its value of 100nF.

M5ATOMのLEDもこのように接続されている

6

図 3-26

上図のように、LED 同士が通信を行うために CPU が搭載されている。高密度な集積である。

LEDの番号

◇M5ATOM Matrix には、5×5個のフルカラーLEDが実装されている

◇希望するLEDを制御するには、LEDの番号と色を指定して次の関数を呼ぶ
M5.dis.drawpix(LED番号, 色)

◇LED番号はUSBコネクタを下に向けて、マトリクス左上から右に0,1,・・・, 4、下の段左から5,6,・・・と続き、右下最後のLEDは24番に割り当てられている

◇LEDの点灯色は、色コードを指定する
色コードは、GRB（緑赤青）の順に0x00～0xff（0～255）で指定する．※0x00は消灯。1ドットに上から順に青赤緑の順で光源LEDが並んでいる

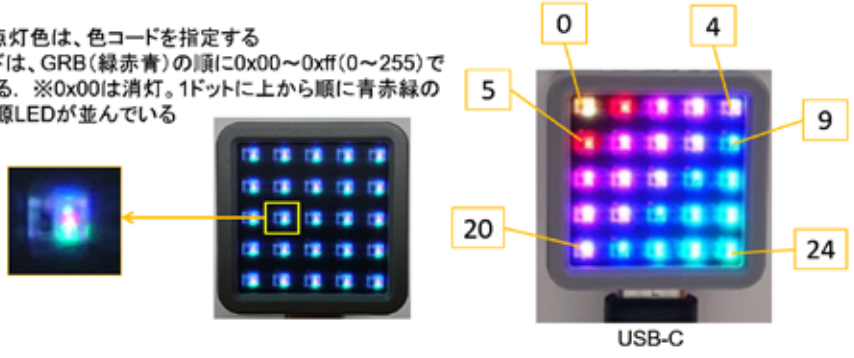


図 3-27

LED を 1つずつG･R･Bで順に点灯してみよう！

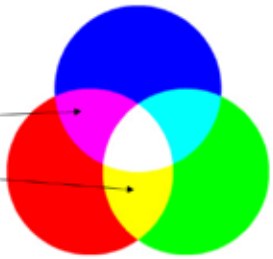
LED巡回点灯

8

図 3-28

システム構想

◇関数 M5.dis.drawpix(LED番号, 色) を呼び出す
① LEDの番号は 0～24 で指定すれば良い
② 色番号は、0xff0000が緑、0x00ff00が赤、0x0000ffが青になっている
③ 0～0xff(0～256)の範囲でLED各色の明るさを調整する
④ 実際に関数に渡すときは、これらを3バイトにして
緑の場合 → 0xff0000
赤の場合 → 0x00ff00
青の場合 → 0x0000ff
として渡す(実際は4バイトで渡される)中間色の場合は、各色の明るさを次のようにする
紫の場合 → 0x00ffff (赤＋青)
黄の場合 → 0xffff00 (赤＋緑)
⑤ 渡す値を小さくすれば各色の明るさが変わる(ここでは0xff固定)
◇LEDを番号順に、緑→赤→青→消灯として1巡したら次のLEDに移る
① LED Indexを0～24で繰り返す
② 色 Indexを0～3で繰り返す
③ 光色は、0x00ff0000の4バイトを(色 Index×8bit)右シフトして渡す
※色Indexが0(最初)の時は緑。3回繰り返してIndexが3の時は24bit右シフトして0x00000000となりLEDは消灯(黒)になる
※このプログラムは、全LEDが正しく動くかどうか、確認するテストも兼ねている



光の三原色

9

図 3-29

この LED は、I2C I/F で M5Atom 本体の CPU と接続されているので、普通の制御プログラムであれば、1wire ライブラリを用いてコマンドの送信を行って点灯制御するが、上図冒頭で説明するように M5.dis.drawpix(LED 番号, 色)というライブラリ関数が用意されているので、コントロールプログラムが作りやすい。図で説明されている色を変えるアルゴリズムをソースコードでよく理解したうえで、システムを作ろう。

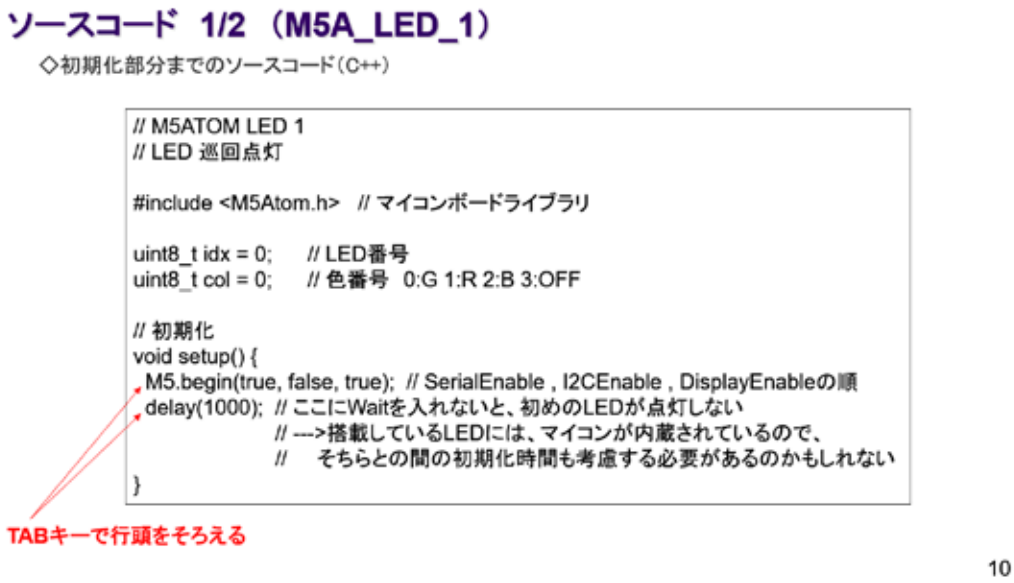


図 3-30

システムの初期化を行うのは setup()関数と決まっている。この関数内の M5.begin()は、M5 シリーズ共通のマイコン初期化関数である。色番号は、通常 RGB の順に並ぶが、このマイコンのライブラリでは、GRB の順になっている。これは、ハードウエアの設計によるもののようだ。ライブラリを修正して RGB の順にすることもできる。

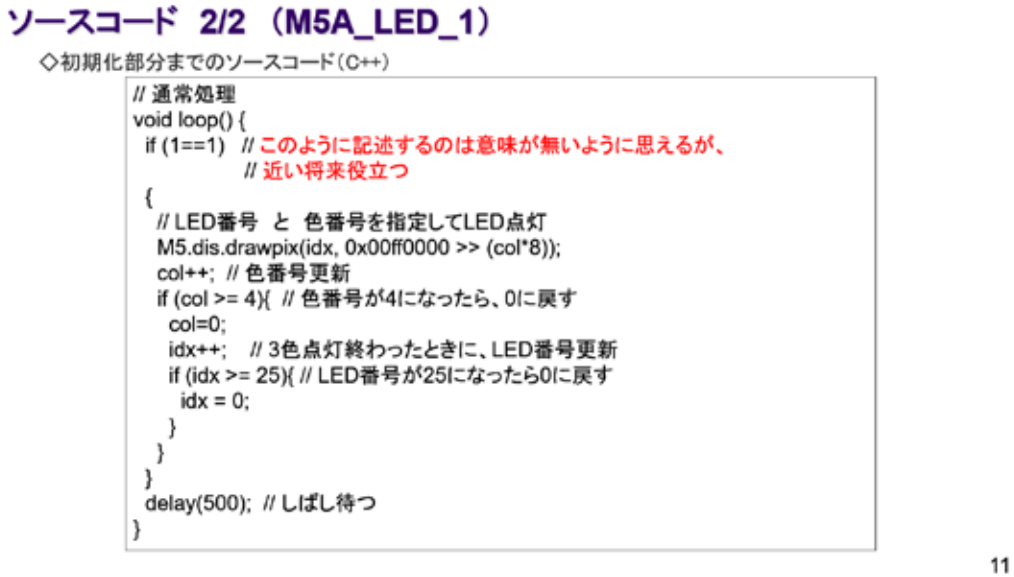


図 3-31

If(1==1)と記述している部分の意味は、後の方のシステム開発でその理由が分かる。

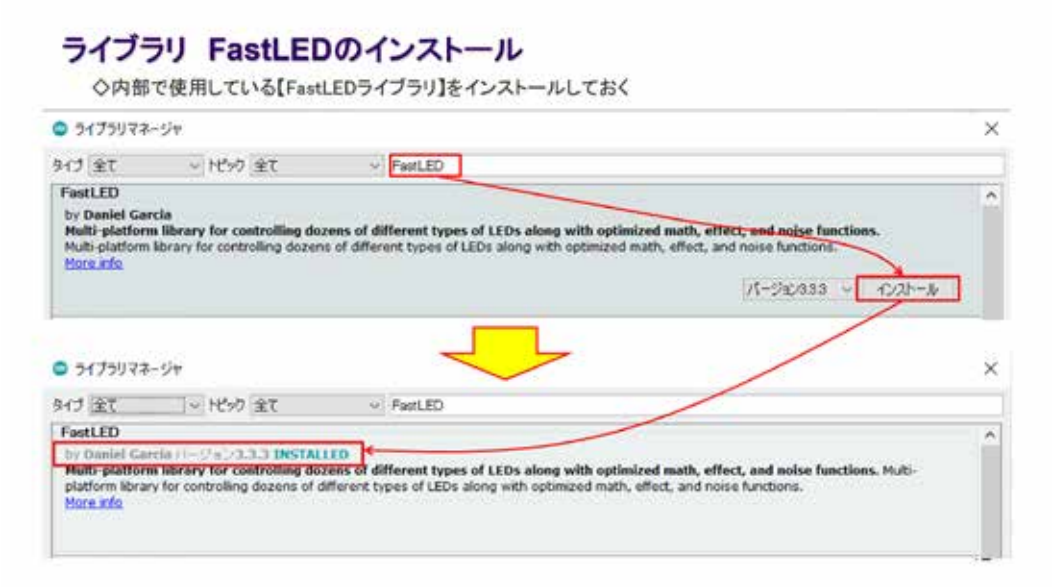


図 3-32

公開されている FastLED は、I2C I/F で LED と通信を行うライブラリと考えてよい。このようなライブラリが公開されていること自体はシステム開発を素早く進めるうえで、とてもありがたいことだが、公開されて間もないライブラリにはバグが含まれていることがある。初めて用いる場合は、周辺の情報を調べることが必要だ。図 3-18 LED ライブラリの修正を参照のこと。

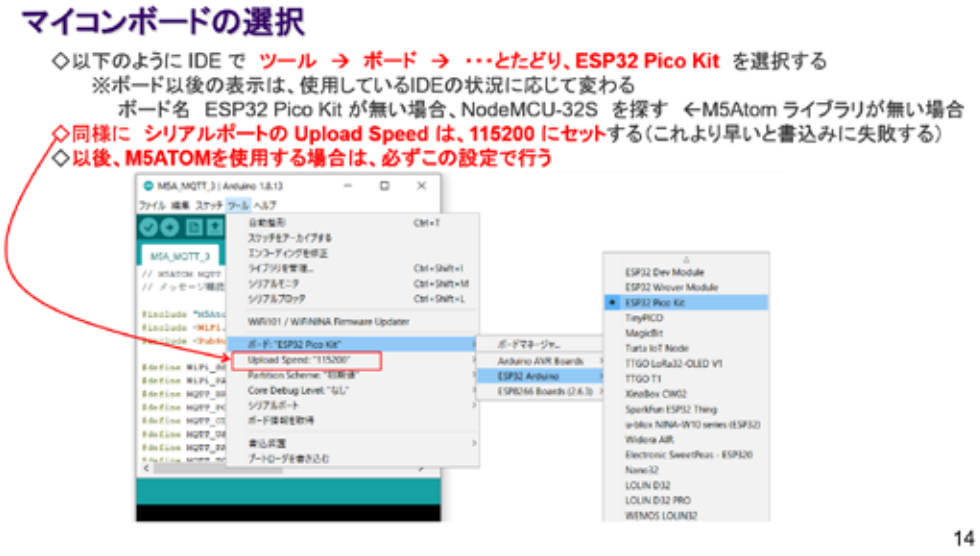


図 3-33

この IDE は、非常に多くのマイコンボードに対応しているので、開発対象のマイコンボードを選択する必要がある。一度選択すれば変更しない限り有効である。この作業は、コンパイルする都度解説するが、同じ内容である。

図 3-34

ソースコードが完成したら、コンパイルからマイコン内部への書き込みまでを一気に行う仕組みとなっている。コンパイル前にマイコンと PC を図に従って USB ケーブル接続する。USB ケーブルは PC とマイコン間の通信ケーブルであるが、電源を供給する役割も持っている。マイコンが PC に接続されるとシリアルポート（COM ポート）が認識されるので、デバイスマネージャでその番号を確認して、IDE のツールで選択する。同じマイコンと PC であれば、COM ポート番号は変わらないので、ほとんどの場合最初に確認した COM ポート番号を今後も使い続けることになる。この作業はコンパイルする都度解説するが、同じ内容である。この後に続く動作確認はシステムによって異なるので、慎重に行うこと。

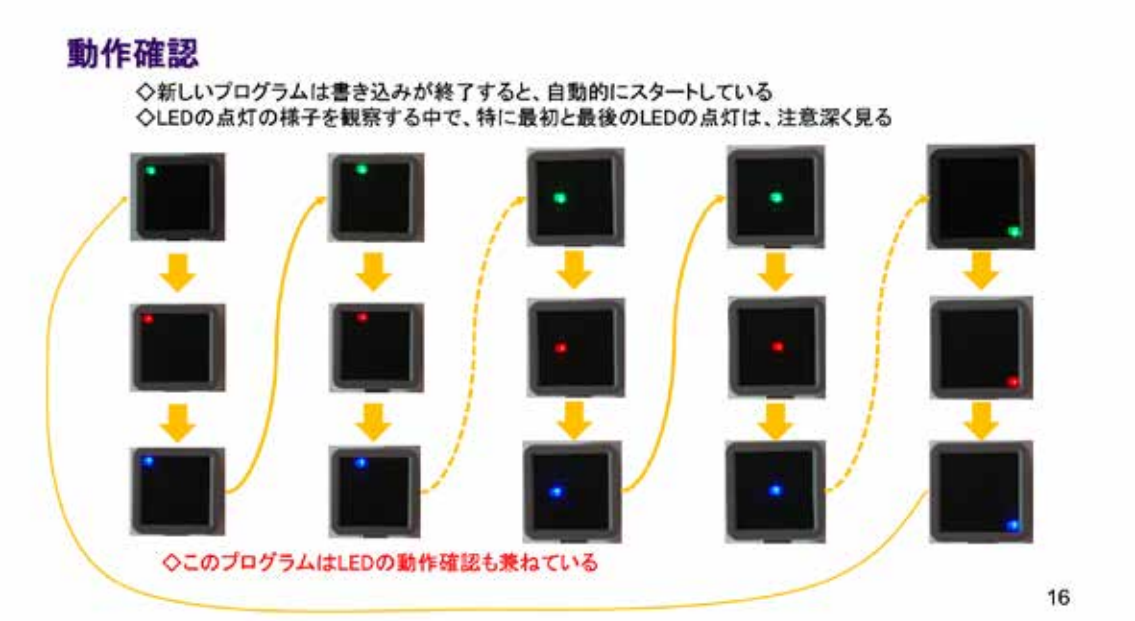


図 3-35

フルカラーLED であることを活かしてグラデーション表示を行ってみよう。これを現場で行うと人の注意を引くことができる。以下のソースコードを記述すれば実現できる。

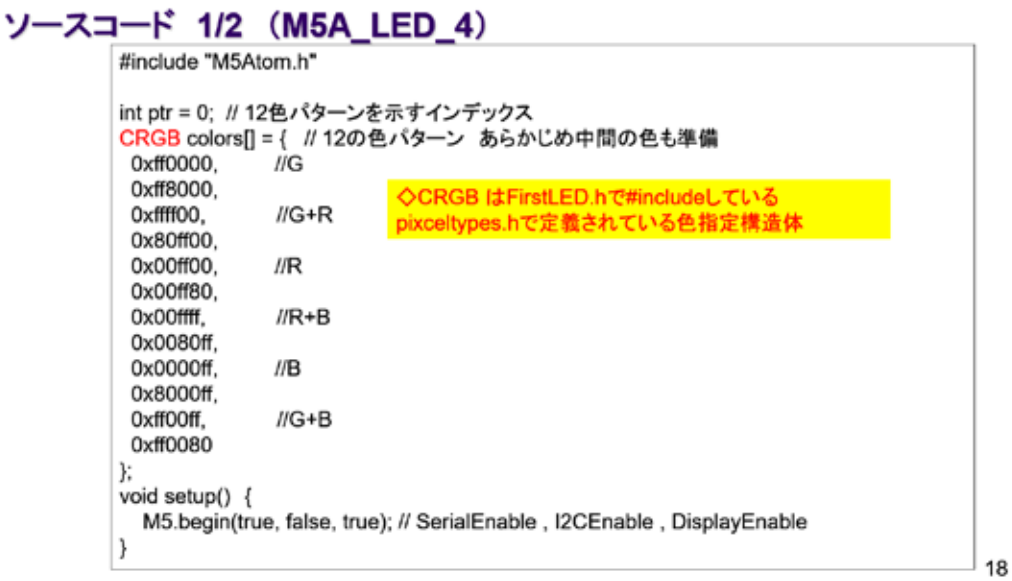


図 3-36

0x で始まるのは、16 進数の色コードである。3byte で、上位 2 桁でそれぞれ G・R・B に対応している。初期化は M5.begin()を呼び出すだけでよい。

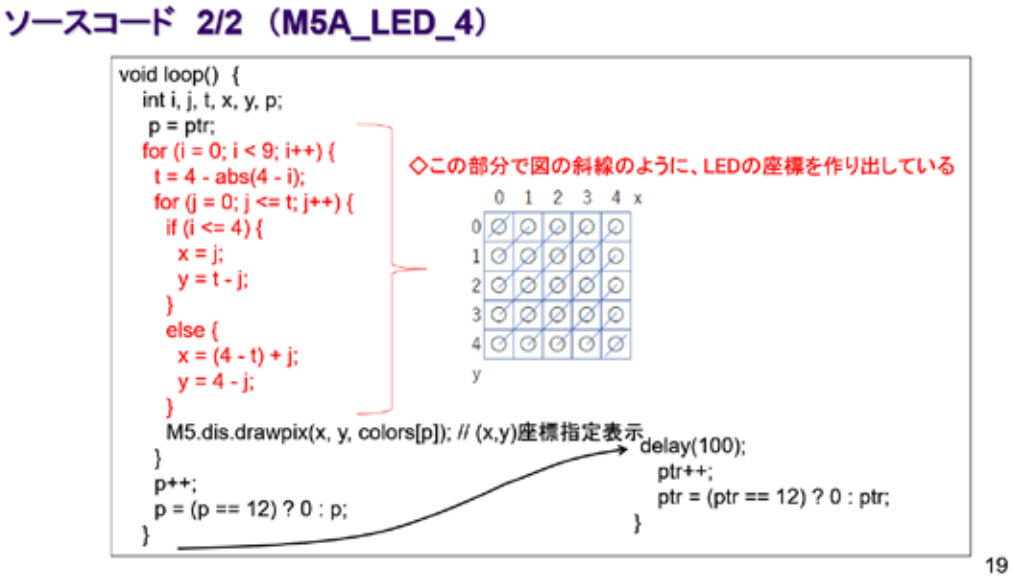


図 3-37

赤字で示した部分は、右に描いた LED マトリクス上での座標を作り出している。ソースコードでは、10 行程度だが、このアルゴリズムを何もないところから考え出すのは時間がかかる。行数が少ないからシステム開発が簡単という訳ではないことを示す良い例である。

図 3-38

ソースコードが完成したら、コンパイルからマイコン内部への書き込みまでを一気に行う仕組みとなっているので、コンパイル前には、マイコンと PC を図に従い、USB ケーブル接続する。USB ケーブルは、PC とマイコン間の通信ケーブルであるが、電源を供給する役割も持っている。マイコンが PC に接続されると、シリアルポート（COM ポート）が認識されるので、デバイスマネージャでその番号を確認して、IDE のツールで選択する。同じマイコンと PC であれば、COM ポート番号は変わらないので、ほとんどの場合最初に確認した COM ポート番号を今後も使い続けることになる。

図 3-39

書込み完了と同時に、マイコンに書き込まれたプログラムが動作を始める。図のように、グラデーション表示が行われることを確認しよう。

図 3-40

文字を表示させてみよう。

図 3-41

ATOM PIXEL TOOL を使えば、マトリクス用のパターンをスクロールに対応したデータとして、作成することができる。

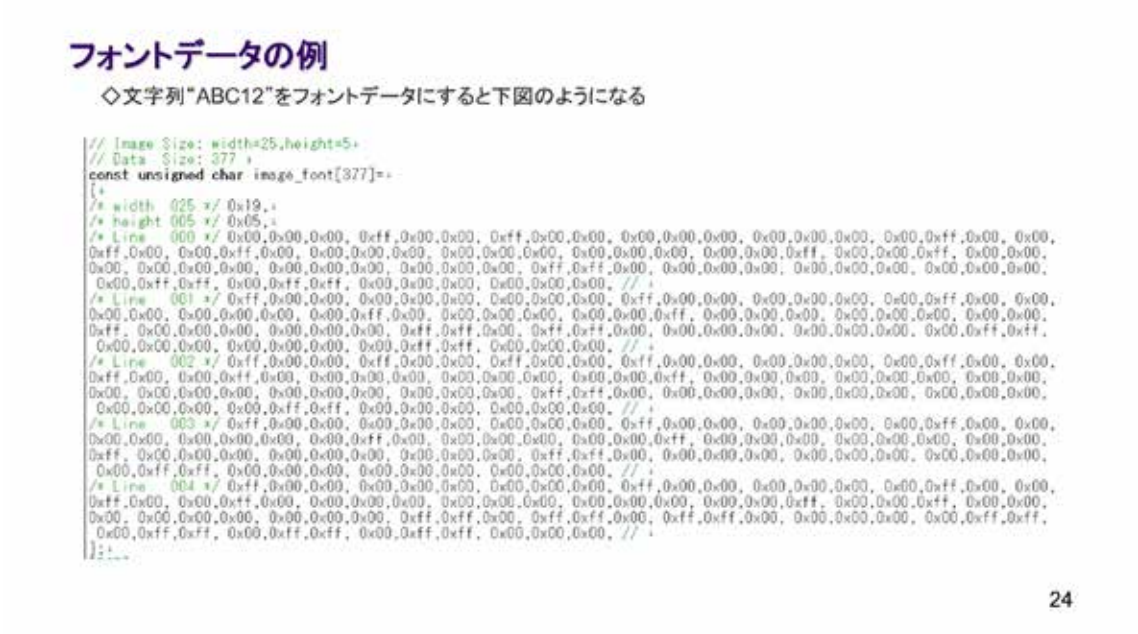


図 3-42

“ABC12”という文字列のパーターンデータを図に示す。これをそのままソースコードに埋め込んで使用できる。以下のソースコードでその部分を示している。

図 3-43

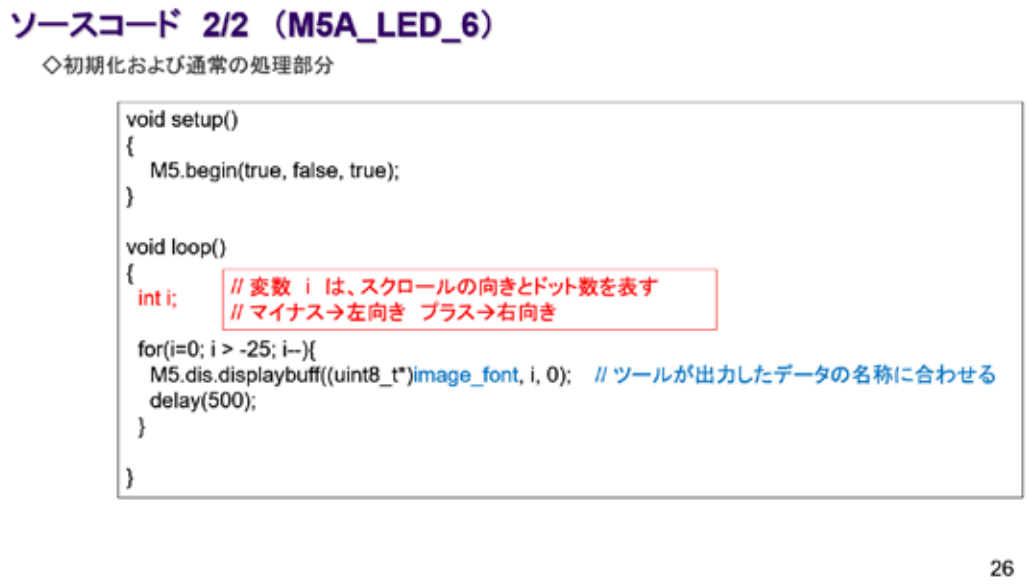


図 3-44

M5.dis.displaybuf()という関数が、LED の表示部分である。パラメータ i でスクロールの向きと 1 回あたりのスクロールドット数を指定している。

図 3-45

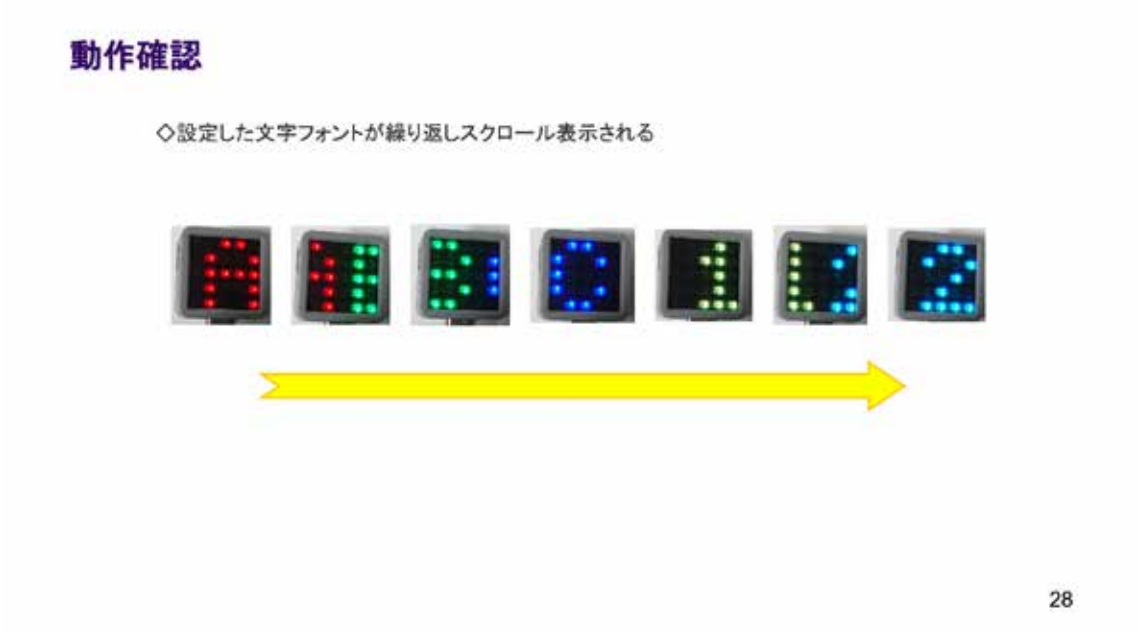


図 3-46

解説のソースコードで動作確認出来たら、M5.dis.displaybuf()のパラメータ i を変更してみると面白い。

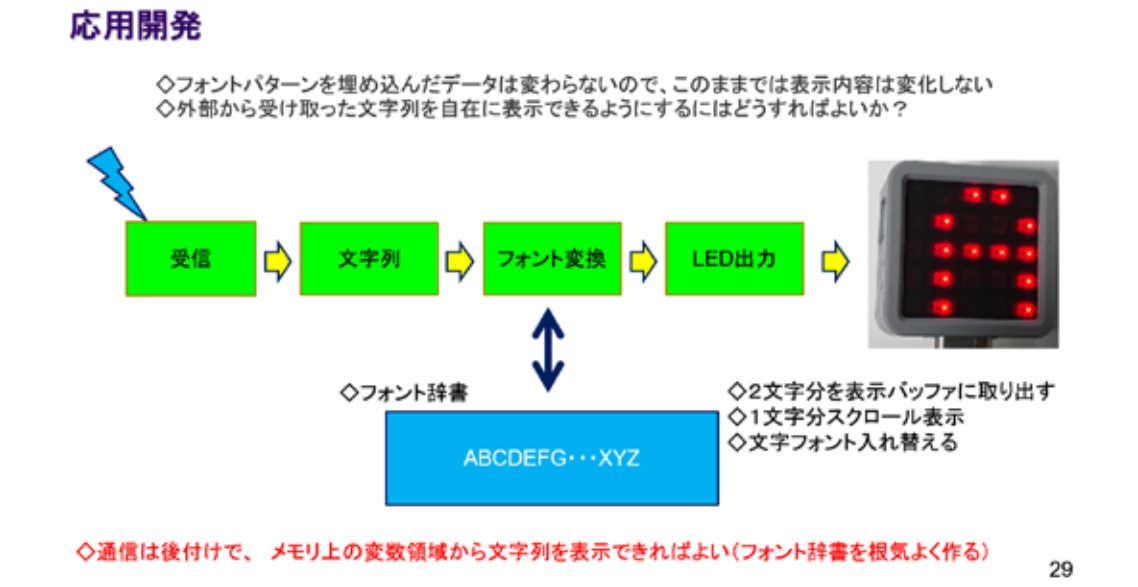


図 3-47

応用開発として、任意の文字表示を行うこともできる。

３．３　M5Atom　－　押しボタンスイッチ

図 3-48

M5Atom は、LED マトリクスの奥に SW を内蔵している。LED 表示の部分を指で押すとクリック感が伝わる。

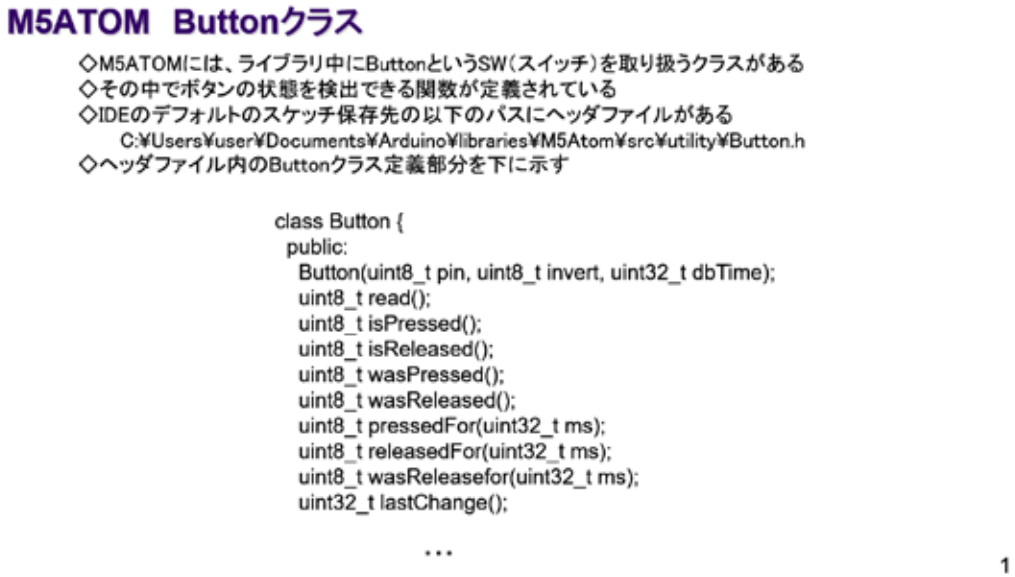


図 3-49

図 3-50

M5 シリーズマイコンのライブラリでは、Button クラスが定義されていて、シリーズすべてのマイコンで共通のソースコードが使えるようになっている。

図 3-51

実習で行いたいことは、この SW を利用した LED の順送りである。

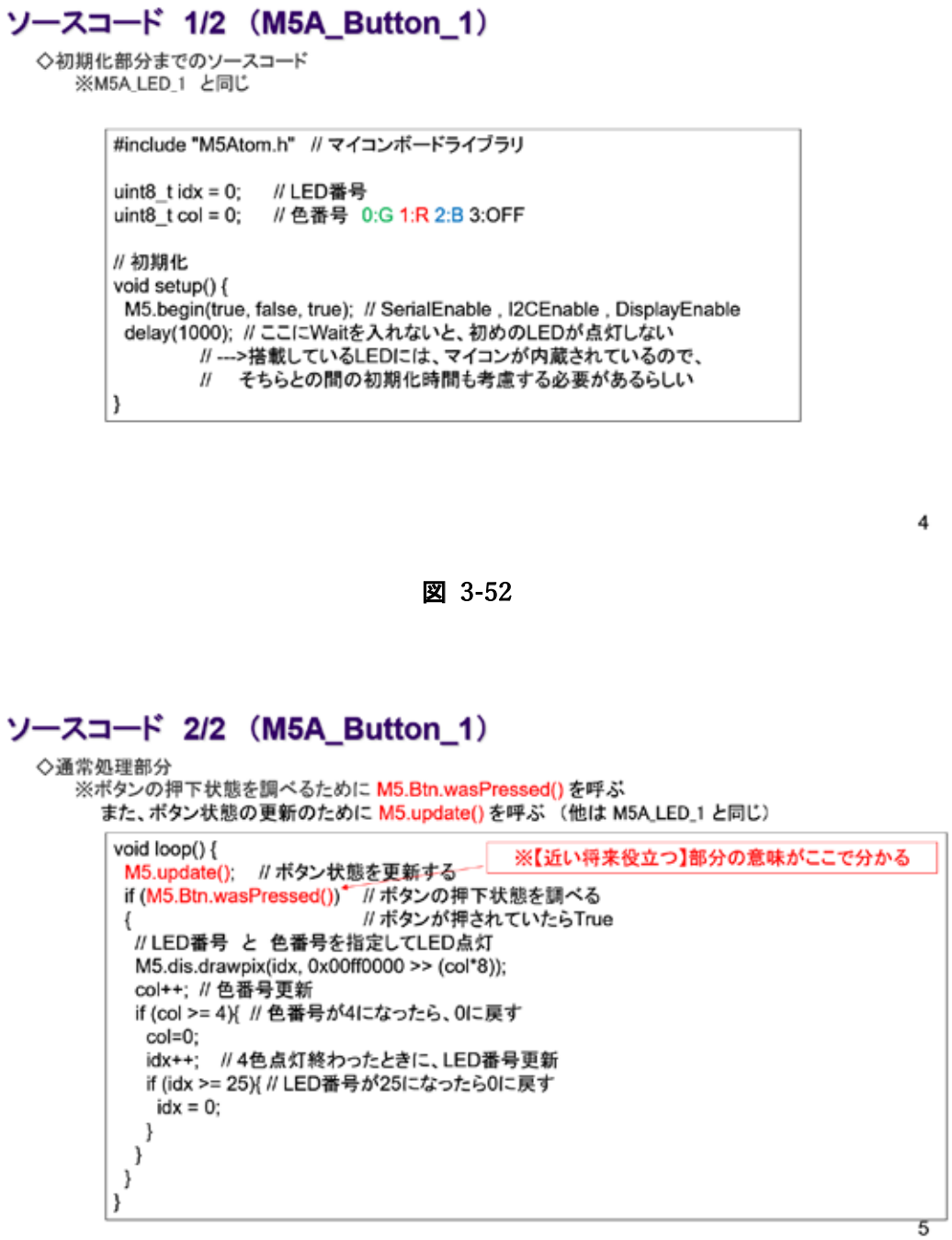


図 3-52

図 3-53

SW 押下に応じた、LED の順送り点灯を行うのは、loop()関数の役割である。この IDE で開発されるシステムには、必ず loop()が必要で、この関数は IDE で選択したマイコンボードの専用の OS から繰り返し呼び出される。Loop()が終了すると、OS は、割込みなどの処理を行い、再び loop()を呼び出す。

図 3-54

ここまで実習を続けていると、このマイコン選択は解説の必要がないかもしれないが、共用の PC を用いている場合は、他の開発者が他のマイコンボードのプログラムをコンパイルしたかもしれないので、確認だけは怠らないようにしよう。

図 3-55

マイコンを USB 接続しコンパイルする。

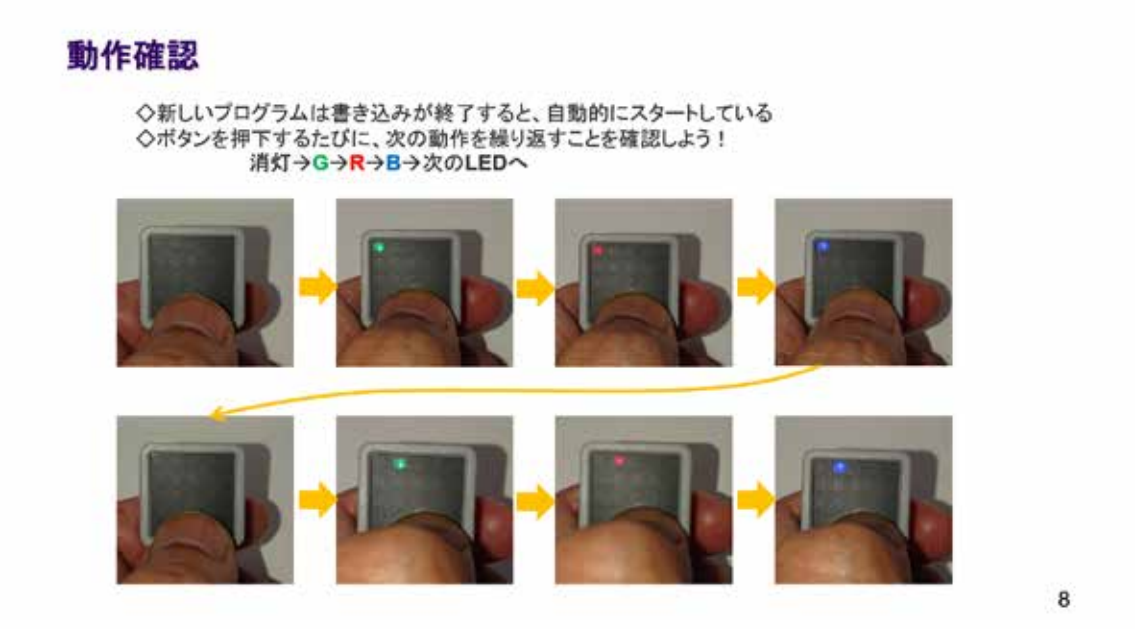


図 3-56

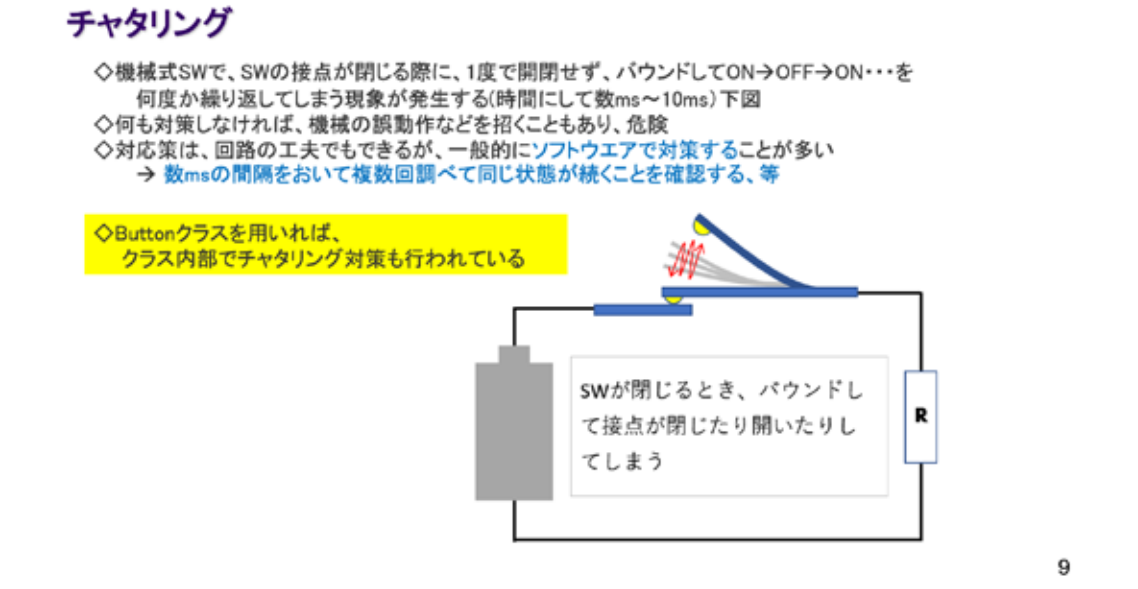


図 3-57

マイコンの能力が低かった時代、チャタリングによるシステム誤動作(機械の誤動作)による事故も発生した。CPU の能力とライブラリが、それを吸収して見えなくしているが、誤動作が発生する理由を熟知しておくことは、いつまでも伝える必要がある。


応用開発
◇LEDの文字フォントスクロール表示をボタンを押すたびに1回行うようにするにはどうすればよいか？
※このようなプログラムが開発できれば、来訪者が受付前に立った時に【ようこそ】と表示ができるし、
通信でスクロール表示をトリガーすることもできる！！
◇ボタンを押下 → スクロール表示 ・・・
10

図 3-58

３．４　M5Atom　－　シリアル通信

図 3-59

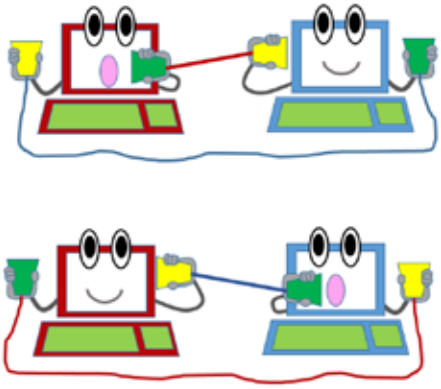
シリアル通信を送信と受信に分けて実習する。

図 3-60

IDE は USB ケーブル経由（COM ポート経由）でマイコンとシリアル通信している。

シリアル通信のイメージ

◇シリアル通信のイメージは【糸電話】



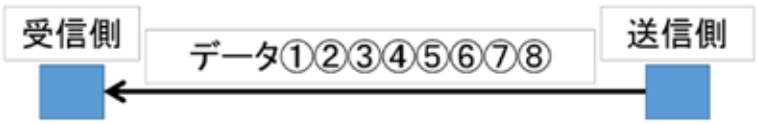
2

図 3-61

シリアル通信は・・・

◇最も多用されている通信方式です。

- ✔ PC間、PC・マイコン・装置間など、多く利用される
- ✔ 1本の信号線で1ビットずつデータを伝送する
- ✔ 基本的に1対1の通信方式

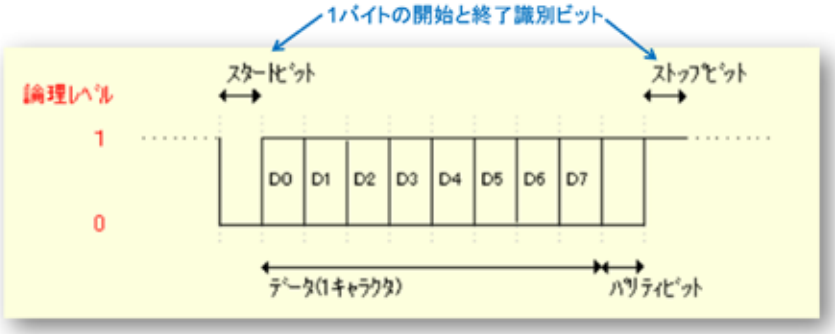


3

図 3-62

信号線の中のデータ

◇1バイトのデータ通信（調歩同期式通信を示す）
※左に向かってデータが送られる様子
複数バイトの場合でも、以下が繰り返される



◇誤り検出のためパリティビットが付加できる

4

図 3-63

文字 A（0x41）のデータ

◇半角アルファベット「A」（ASCIIコードで0x41）のデータを
偶数パリティで送信すると、図のようになる

D0	D1	D2	D3	D4	D5	D6	D7	P
1	0	0	0	0	0	1	0	0

'A' のデータ（偶数パリティ）

※下位（D0）のビットから送られる
実際のデータはD7が上位なので、（パリティ＋0x41）となる
※偶数パリティ → パリティビットも含めて全ビットをビット加算した
結果が0になるようにパリティビットをセットする

5

図 3-64

図 3-65

まず送信から始める。

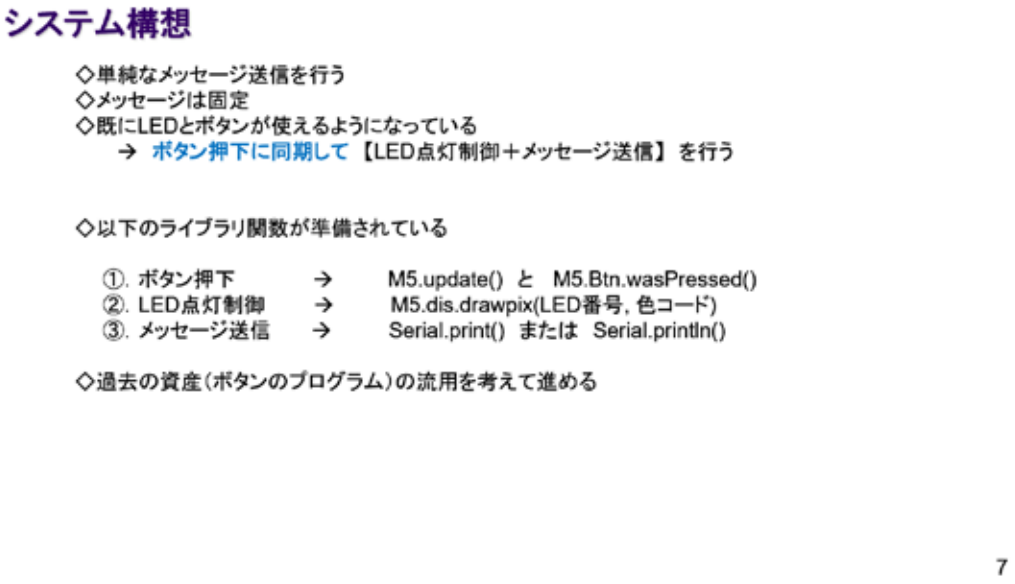


図 3-66

メッセージの送信には、Serial.print()関数と Serial.println()関数が使える。後者は、改行付きでメッセージを送信する。

シリアル通信のデフォルト速度

◇M5ATOMは、シリアルポートの初期化処理も、デフォルトでライブラリが処理をする
◇C:¥Users¥user¥Documents¥Arduino¥libraries¥M5Atom¥src¥M5Atom.cpp の該当部分を示す

```
void M5Atom::begin(bool SerialEnable , bool I2CEnable , bool DisplayEnable )
{
        if( _isInited ) return;

        _isInited = true;

        if( I2CEnable )
        {
                Wire.begin(25,21,10000);
        }
        if( SerialEnable )
        {
                Serial.begin(115200);
                Serial.flush();
                delay(50);
                Serial.print("M5Atom initializing...");
        }
```

※この部分でシリアルポートの初期化が行われている Serial.begin()のパラメータが通信速度
初期メッセージは改行コードを出力していないので Serial.println() に変更しても良い

8

図 3-67

デフォルト設定で、115200bps の速度が設定されている。これでメッセージを送信するので、動作確認では、受信側もこの速度に合わせる必要があることを記憶しておく。

ソースコード 1/2 （M5A_Serial_1）

◇初期化処理まで （この部分は、ボタンのプログラムと同じ）

```
#include <M5Atom.h>"  // マイコンボードライブラリ

uint8_t idx = 0;     // LED番号
uint8_t col = 0;     // 色番号  0:G 1:R 2:B 3:OFF

// 初期化
void setup() {
  M5.begin(true, false, true);  // SerialEnable , I2CEnable , DisplayEnable
  delay(1000);  // ここにWaitを入れないと、初めのLEDが点灯しない
              // --->搭載しているLEDには、マイコンが内蔵されているので、
              //    そちらとの間の初期化時間も考慮する必要があるらしい
}
```

9

図 3-68

ソースコード 2/2 （M5A_Serial_1）

◇通常処理では Serial.println("ABC123abc!"); でメッセージを送信している

```
// 通常処理
void loop() {
  M5.update(); // ボタン状態を更新する
  if (M5.Btn.wasPressed()) { // ボタンの押下状態→押されていたらTrue
    Serial.println("ABC123abc!"); // メッセージ送信　改行コード付き
    M5.dis.drawpix(idx, 0x00ff0000 >> (col*8)); / / LED番号と色番号を指定してLED点灯
    col++; // 色番号更新
    if (col >= 4){  // 色番号が4になったら、0に戻す
      col=0;
      idx++;    // 3色点灯終わったときに、LED番号更新
      if (idx >= 25){ // LED番号が25になったら0に戻す
        idx = 0;
      }
    }
  }
}
```

10

図 3-69

マイコンボードの選択

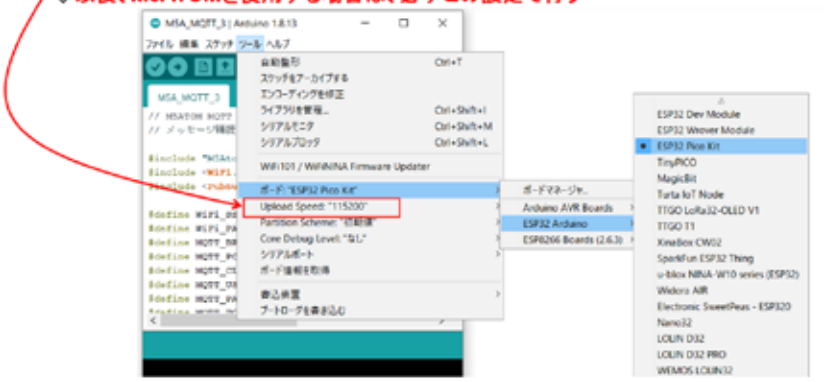
◇以下のように IDE で **ツール → ボード → ・・・とたどり、ESP32 Pico Kit** を選択する

※ボード以後の表示は、使用しているIDEの状況に応じて変わる

ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合

◇同様に **シリアルポートの Upload Speed は、115200 にセット**する（これより早いと書込みに失敗する）

◇**以後、M5ATOMを使用する場合は、必ずこの設定で行う**



11

図 3-70

図 3-71

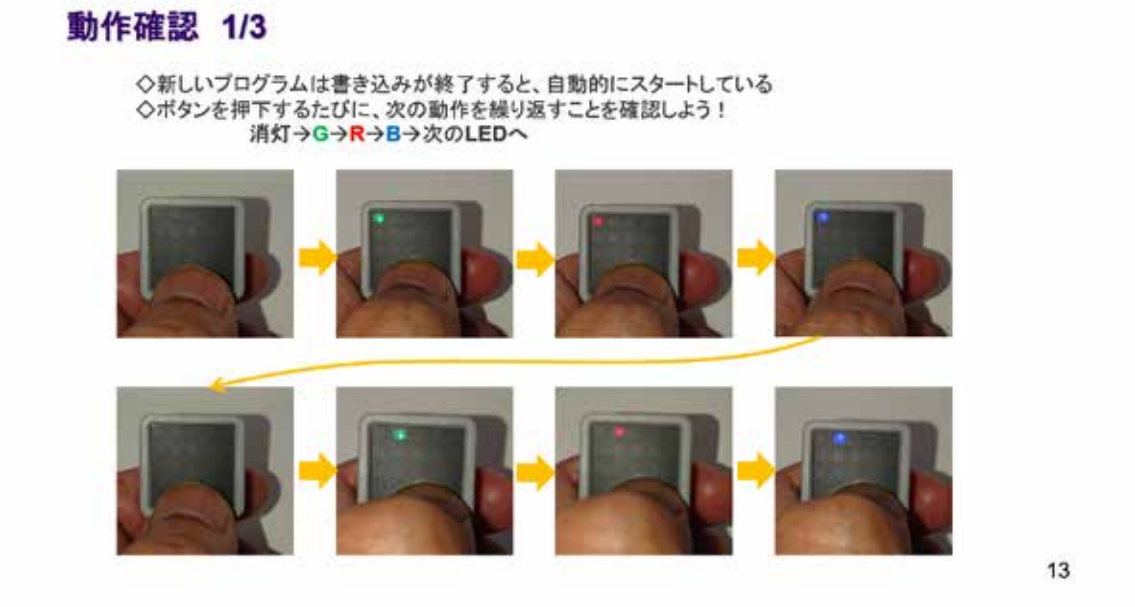


図 3-72

まず、LED 点灯制御の動作確認を行う。

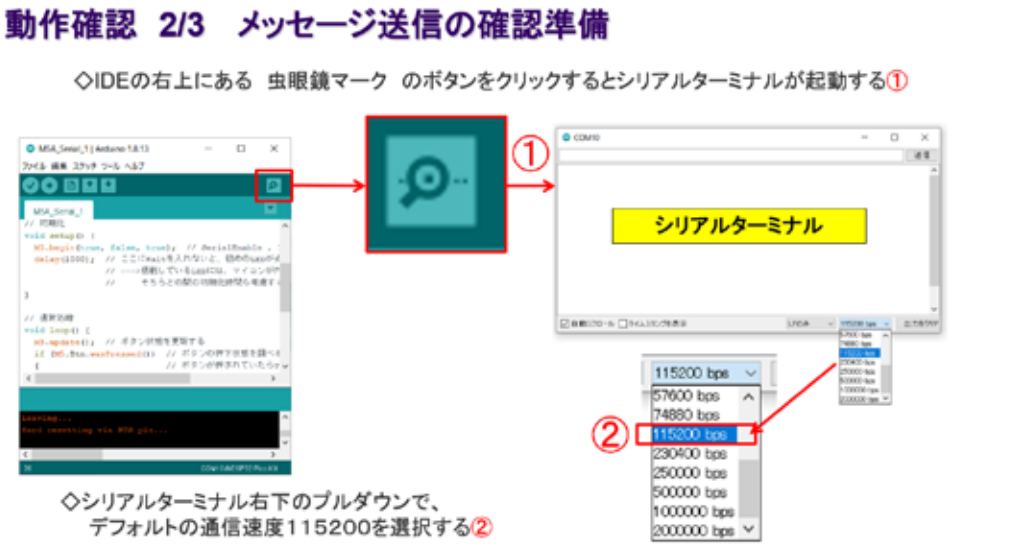


図 3-73

受信側もライブラリがデフォルトで設定している通信速度 115200bps に設定する。

図 3-74

複数の機能を盛り込んだシステムの動作確認は、機能ごとに行うので長くなるが、単独の機能が目論見通り、仕様を満たしているかを慎重に確認する。

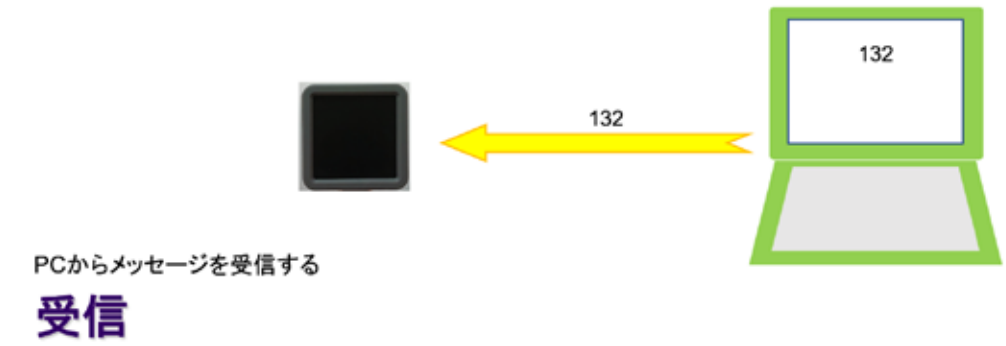


図 3-75

次は受信を行ってみよう。

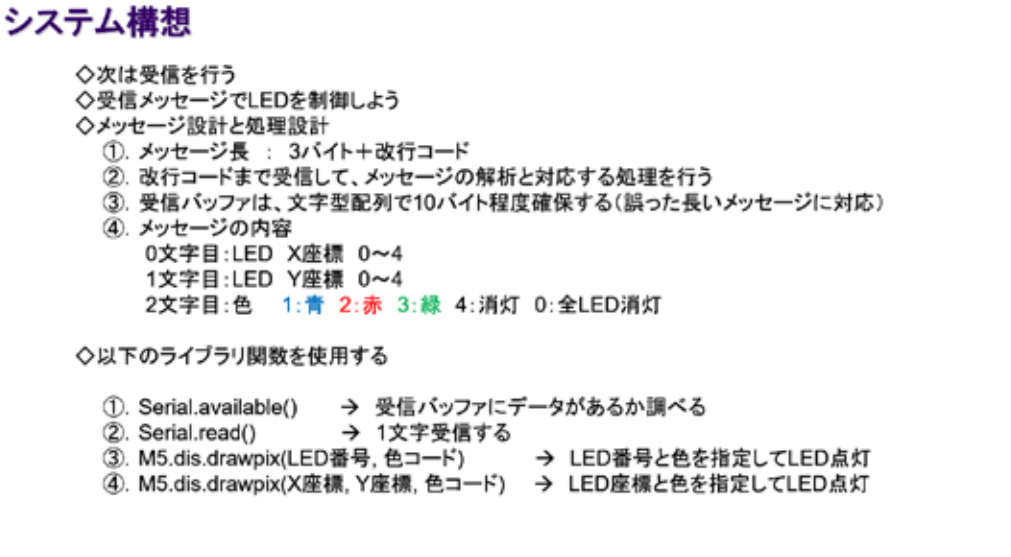


図 3-76

これまで実習したすべての機能を盛り込むので、それだけソースコードは増える。

ソースコード 1/5 （M5A_Serial_2）

◇初期化処理まで

```
#include <M5Atom.h>  // マイコンボードライブラリ

int i;        // 1文字受信時のバッファインデックス
char buf[10]; // メッセージバッファ

void fill_led(unsigned long col){  // LEDをすべて同じ色にする  全消灯に利用する
  for(int i=0; i<25; i++){
    M5.dis.drawpix(i, col);   // LED番号を0～24まで指定して色を設定する
  }
}

void setup() {   // 初期化
  M5.begin(true, false, true);  // SerialEnable , I2CEnable , DisplayEnable
  delay(1000);  // ここにWaitを入れないと、初めのLEDが点灯しない
                // --->搭載しているLEDには、マイコンが内蔵されているので、
                //    そちらとの間の初期化時間も考慮する必要があるらしい
 i=0;         // バッファインデックス初期化
}
```

18

図 3-77

ソースコード 2/5 （M5A_Serial_2）

◇通常処理 最も外側部分

```
void loop() {  // 通常処理
   char c;      // 受信データ  1文字分
   int x;       // LED  X座標
   int y;       // LED  Y座標
   int col;     // LED  色指定
   unsigned long color; // 色コード

   // 受信処理
   //座標・色コード計算
   //エラーチェック
   //色コード計算・LED点灯
}
```

19

図 3-78

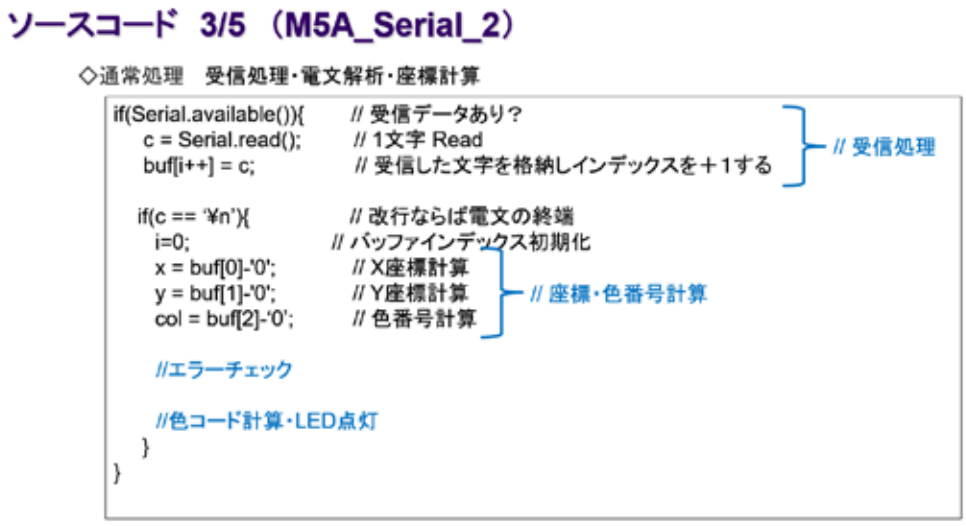


図 3-79

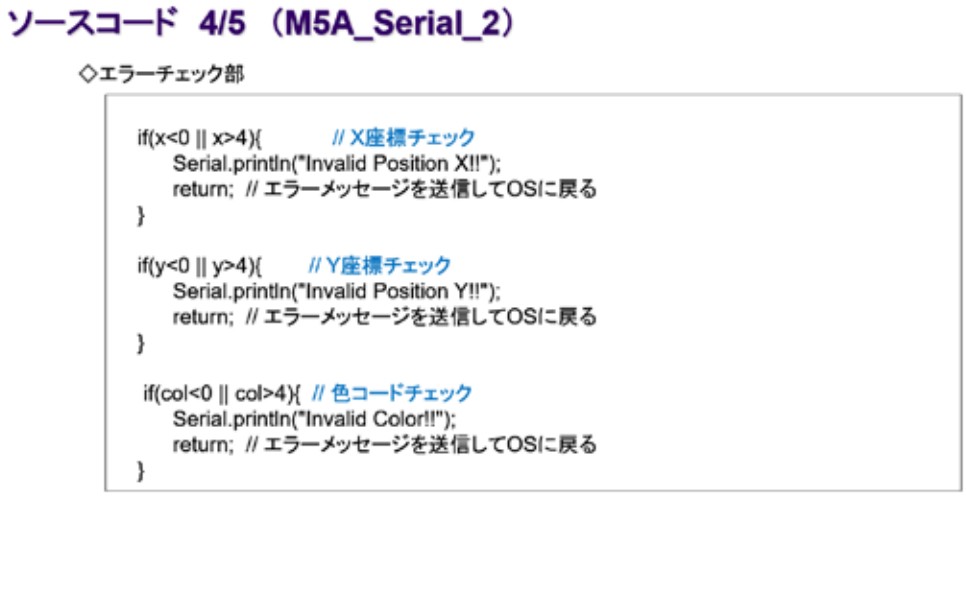


図 3-80

エラーチェックを丁寧に行うことで、システムテストにも役立ち、信頼性も増す。動作確認では、わざわざエラーを起こすことも必要になる。

ソースコード 5/5 （M5A_Serial_2）

◇色コード計算・LED点灯処理

```
if(col == 0){
      fill_led(color = 0x000000);   // ← 冒頭で宣言した関数をCallして、LED全消灯する
      return;                       // LEDをすべて消灯
} else if(col == 4) {
      color = 0x000000;             // 色コード　黒＝消灯
} else {
      color = 0x0000ff << (col-1)*8;  // 他の色コード計算
}

M5.dis.drawpix(x, y, color);      // X・Y座標 ・ 色コード指定LED点灯
```

22

図 3-81

マイコンボードの選択

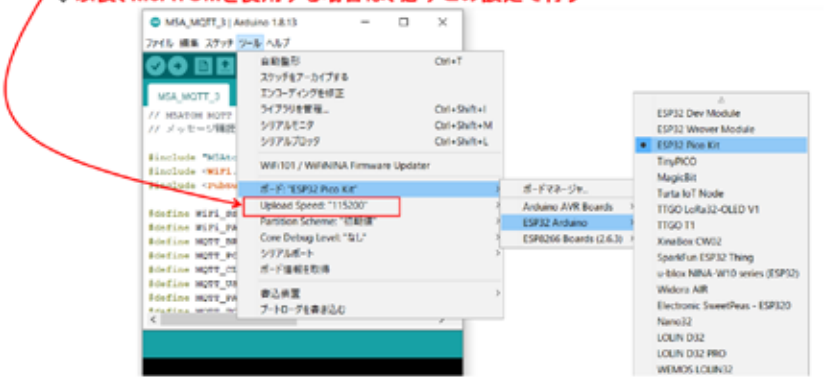
◇以下のように IDE で **ツール → ボード → ・・・とたどり、ESP32 Pico Kit** を選択する

※ボード以後の表示は、使用しているIDEの状況に応じて変わる

ボード名　ESP32 Pico Kit が無い場合、NodeMCU-32S　を探す　←M5Atom ライブラリが無い場合

◇同様に　**シリアルポートの Upload Speed は、115200 にセット**する（これより早いと書込みに失敗する）

◇**以後、M5ATOMを使用する場合は、必ずこの設定で行う**



23

図 3-82


マイコンをPCと接続
①. M5ATOMをPCと接続
PCのUSBポートに接続
開発用PC
M5ATOM
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
24

図 3-83

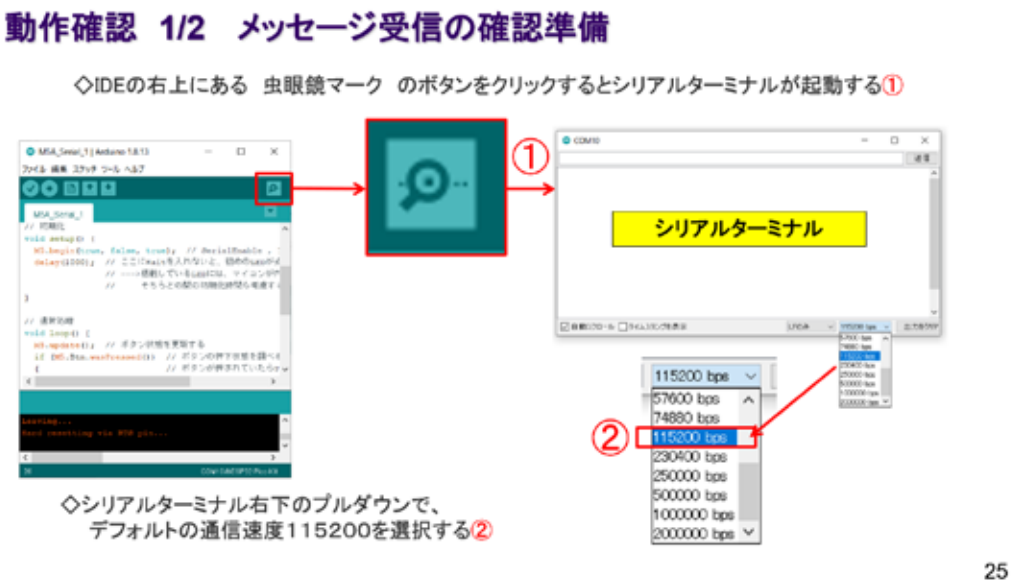

動作確認 1/2 メッセージ受信の確認準備
◇IDEの右上にある 虫眼鏡マーク のボタンをクリックするとシリアルターミナルが起動する①
①
シリアルターミナル
115200 bps
57600 bps
74880 bps
115200 bps
230400 bps
250000 bps
500000 bps
1000000 bps
2000000 bps
②
◇シリアルターミナル右下のプルダウンで、
デフォルトの通信速度115200を選択する②
25

図 3-84

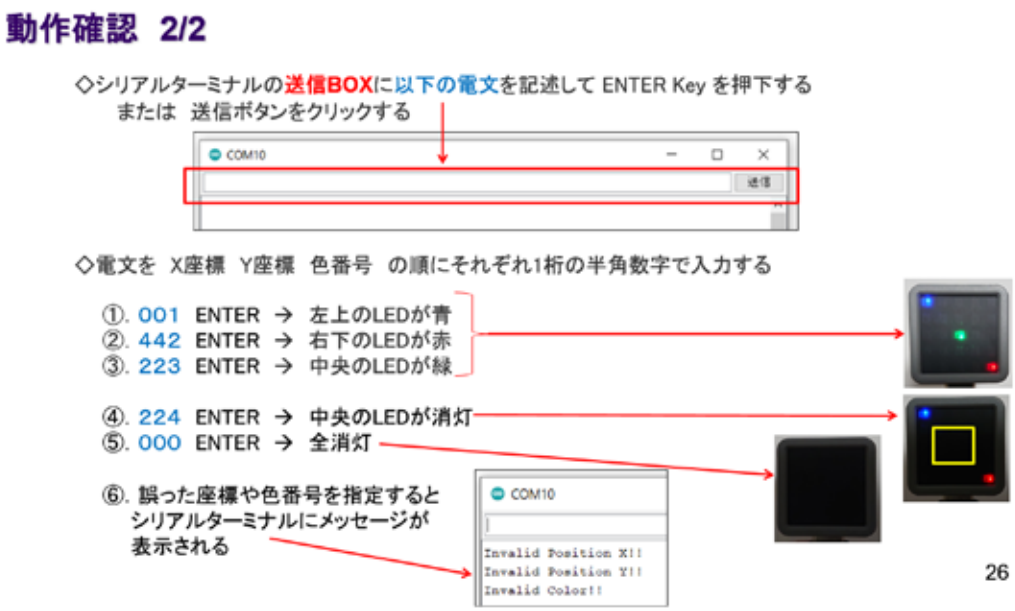

動作確認 2/2
◇シリアルターミナルの送信BOXに以下の電文を記述して ENTER Key を押下する
または 送信ボタンをクリックする
COM10
送信
◇電文を X座標 Y座標 色番号 の順にそれぞれ1桁の半角数字で入力する
①. 001 ENTER → 左上のLEDが青
②. 442 ENTER → 右下のLEDが赤
③. 223 ENTER → 中央のLEDが緑
④. 224 ENTER → 中央のLEDが消灯
⑤. 000 ENTER → 全消灯
⑥. 誤った座標や色番号を指定すると
シリアルターミナルにメッセージが
表示される
COM10
Invalid Position X!!
Invalid Position Y!!
Invalid Color!!
26

図 3-85

３．５　M5Atom　－　MQTT（WEB 経由通信）

図 3-86

IoT では、インターネットを経由した情報交換が必須である。コンパクトな M5Atom でもそれが行えることを確認する。

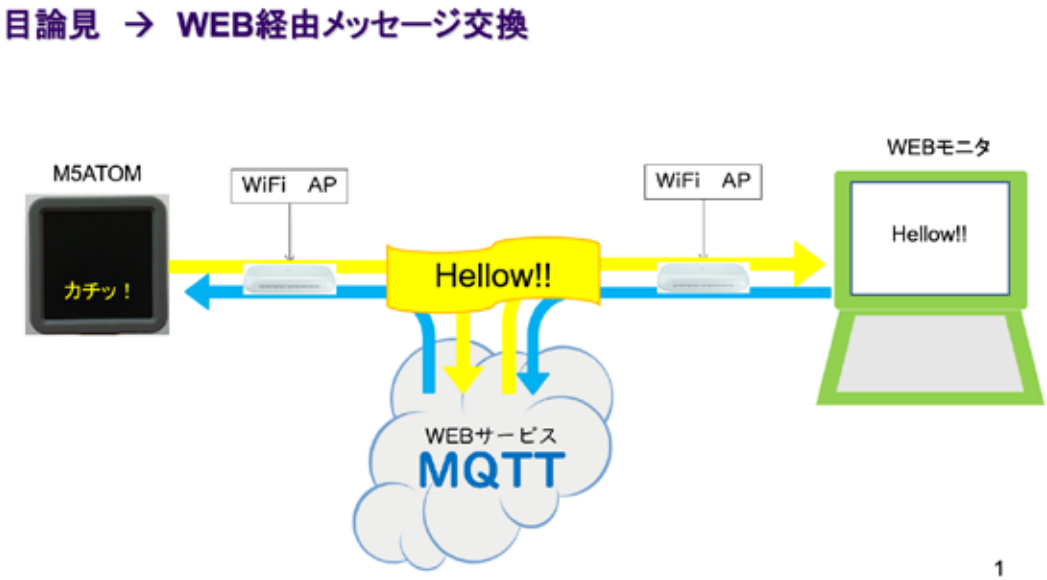


図 3-87

MQTTサービス

◇MQTTは、Message Queue Telemetry Transport の頭文字
◇サービスプロバイダ(MQTT Broker)が世界中にある
◇短いメッセージ交換に特化したWEBサービス
◇登録不要で即利用できる Broker が多い

◇WEBを経由するメッセージ交換 → インターネットに接続できれば場所を選ばない
◇マイコン⇔マイコン間、PC⇔PC間、マイコン⇔PC間でのメッセージ交換が可能
◇言語依存しない(マイコン向けC＋＋・MicroPython・PC用Python・同C++・・・etc)
ライブラリが必要 → 本講座では PubSubClient を利用するのでIDEにインストールする

◇ROS(Robot OS)でnode間通信にも採用されている仕組み
◇ルールが簡単

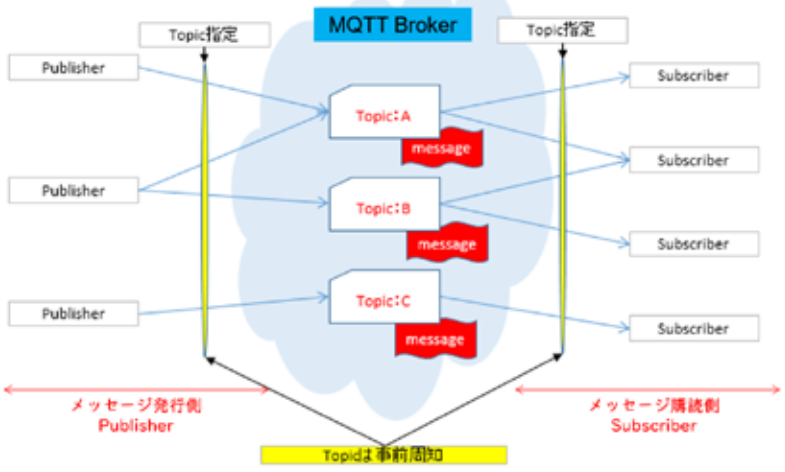
2

図 3-88

WEB 経由でメッセージ交換を行うためのライブラリが公開されている。

MQTTの仕組み

◇トピック名付きメッセージを発行すると、購読登録しているクライアントに通知される



3

図 3-89

メッセージの発行側とそれを購読する側であらかじめ決めた Topic 名が、MQTT の大切な要素である。

図 3-90

次に紹介する broker.shiftr.io は、比較的新しいが、仕組みを理解するうえでシステム開発上、とても使いやすい MQTT Broker である。

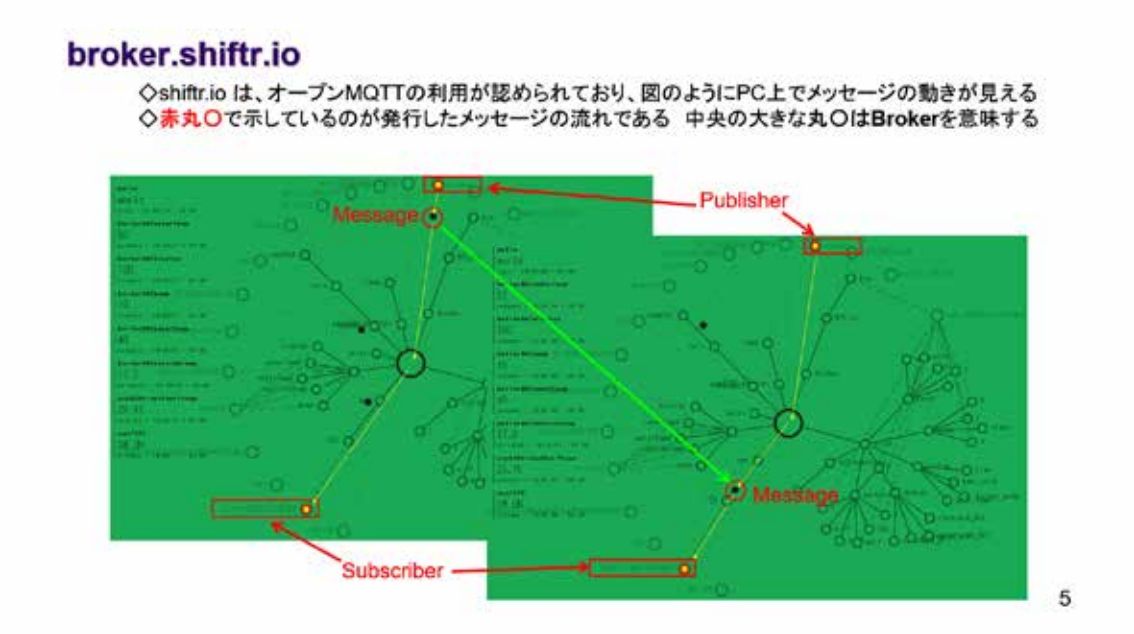


図 3-91

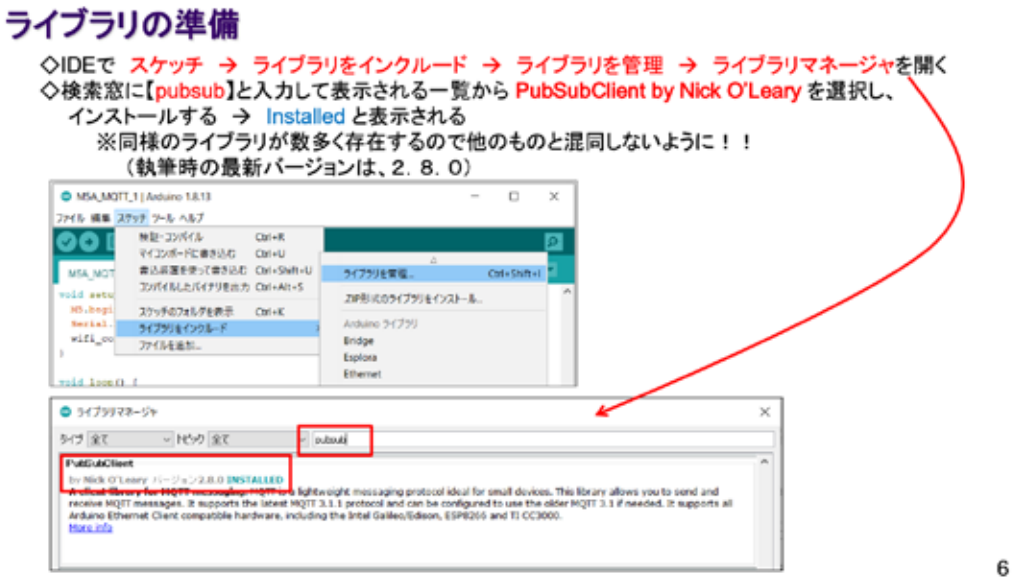


図 3-92

先に説明した、MQTT によるメッセージ交換のためのライブラリをここでインストールしておく。図の手順に従って、PubSubClient ライブラリを選択する。MQTT に関係するライブラリは非常に多いので、他のライブラリと間違えないようにインストールする。異なるライブラリを利用しても MQTT を活用できるが、関数のパラメータが異なる、関数そのものが無い、などの問題が発生するので注意すること。但し、いつかは自力で新しいライブラリを利用してシステム開発ができるようにならなければいけない。

図 3-93

まず、メッセージの発行から始める。

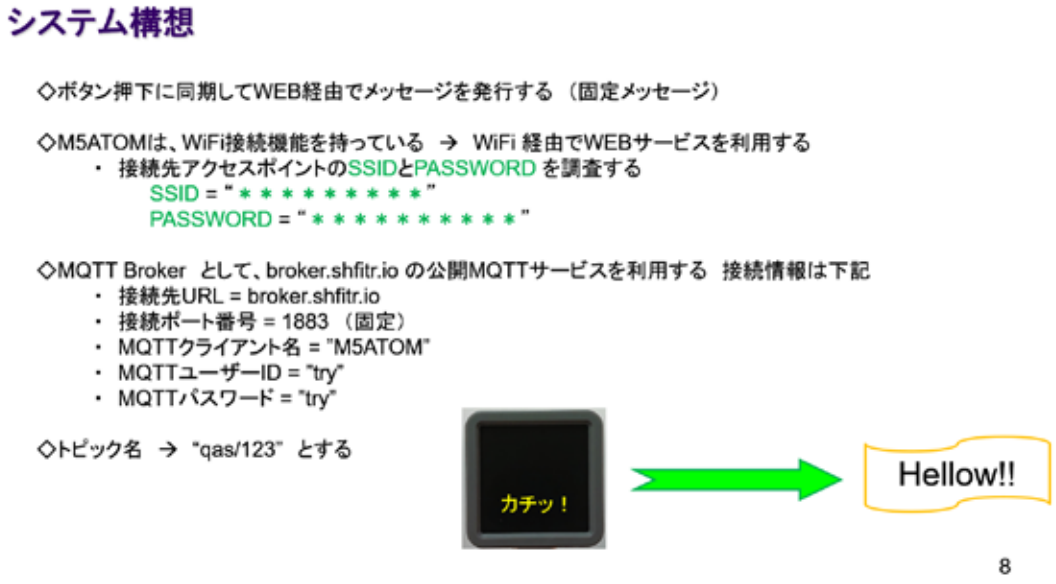


図 3-94

シリアル通信の送信と同様にメッセージを送信するシステムを作る。

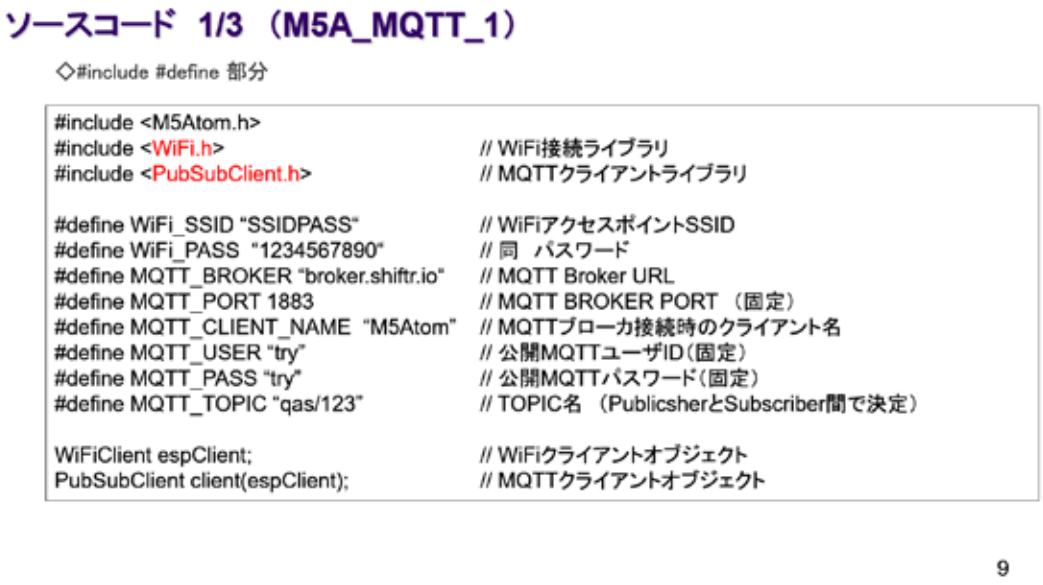


図 3-95

冒頭に並ぶ多くの #define で定義している内容を理解して、開発者自身の環境に合わせた変更が必要になる。利用する WiFi アクセスポイントや MQTT Broker もここで指定している。

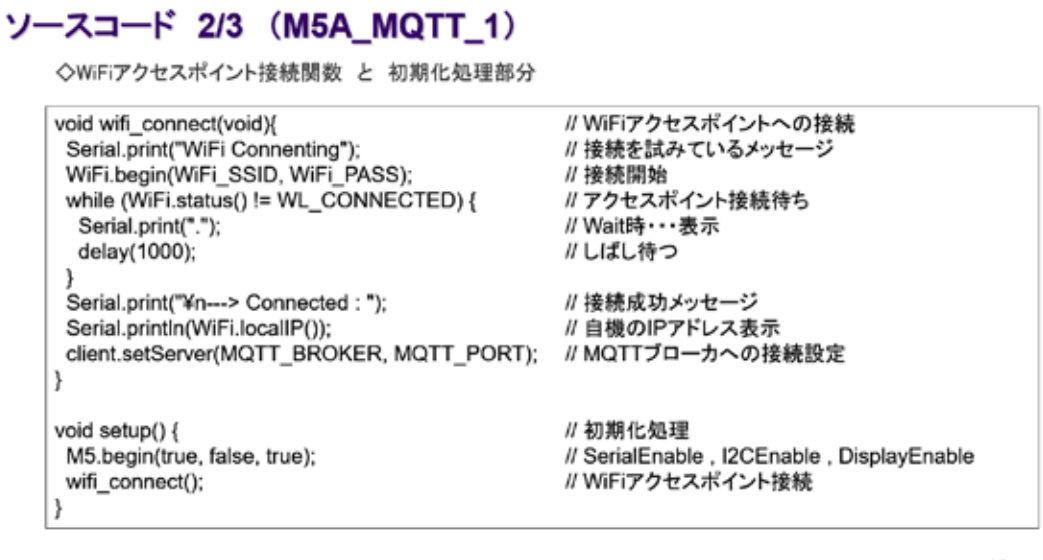


図 3-96

WiFi アクセスポイントへの接続手続きは、標準のアプリケーション例に従っている。Wifi_connect()関数は、このソースコードの後の方で、記述されている。

ソースコード 3/3 （M5A_MQTT_1）

```
void loop() {                                          // 通常処理
  client.loop();                                       // MQTT接続状況更新
  while(!client.connected()){                          // MQTT接続
    Serial.println("Mqtt Reconnecting");               // MQTT接続 試みているメッセージ
    if( client.connect(MQTT_CLIENT_NAME, MQTT_USER, MQTT_PASS) ){
      Serial.println("Mqtt Connected");                // MQTT接続成功メッセージ
      break;
    }
    delay(3000);                                       // しばし待ってから再接続
  }

  M5.update();                                         // M5ATOMボタン状況更新
  if (M5.Btn.wasPressed()){                            // ボタンが押されていたか？
    client.publish(MQTT_TOPIC, "Button was pressed !!"); // ここでボタン押下メッセージを送信
    Serial.println("Send message");                    // シリアルターミナルにも出力（動作確認に必須）
  }
}
```

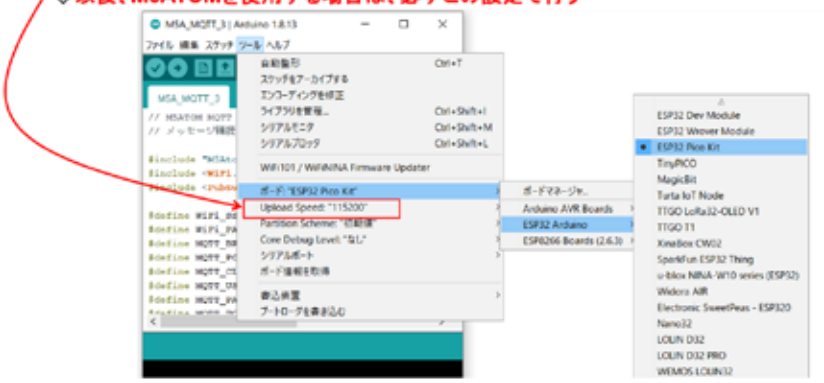
11

図 3-97

WEB 経由の通信を行う MQTT の利用も、ライブラリの恩恵を受けて、たいへん短いソースコードで記述できる。

マイコンボードの選択

◇以下のように IDE で **ツール → ボード → ・・・とたどり、ESP32 Pico Kit** を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に **シリアルポートの Upload Speed は、115200 にセット**する（これより早いと書込みに失敗する）
◇**以後、M5ATOMを使用する場合は、必ずこの設定で行う**



12

図 3-98

図 3-99

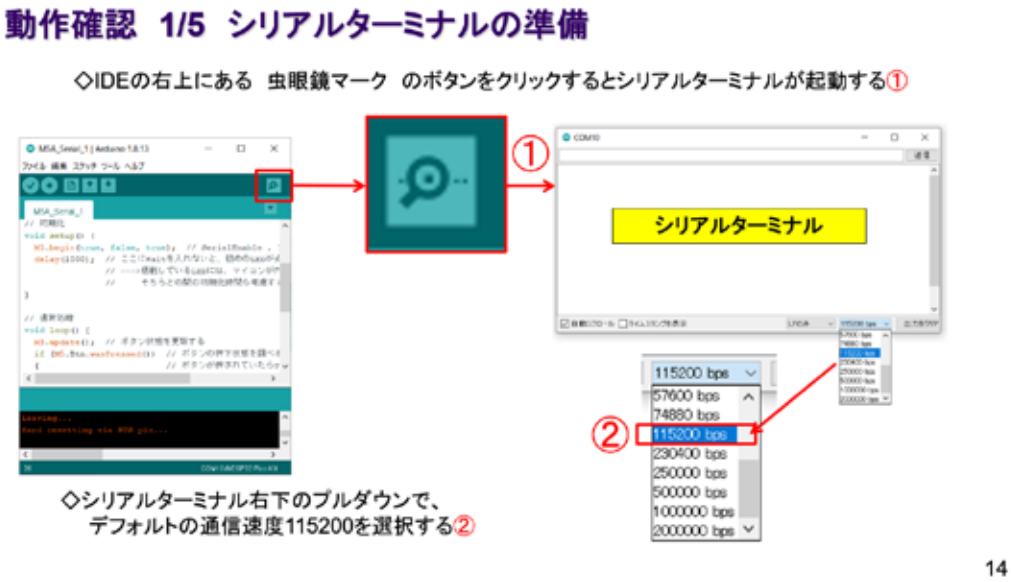


図 3-100

MQTT メッセージと同期してシリアル出力も行っている。その機能も動作確認する。

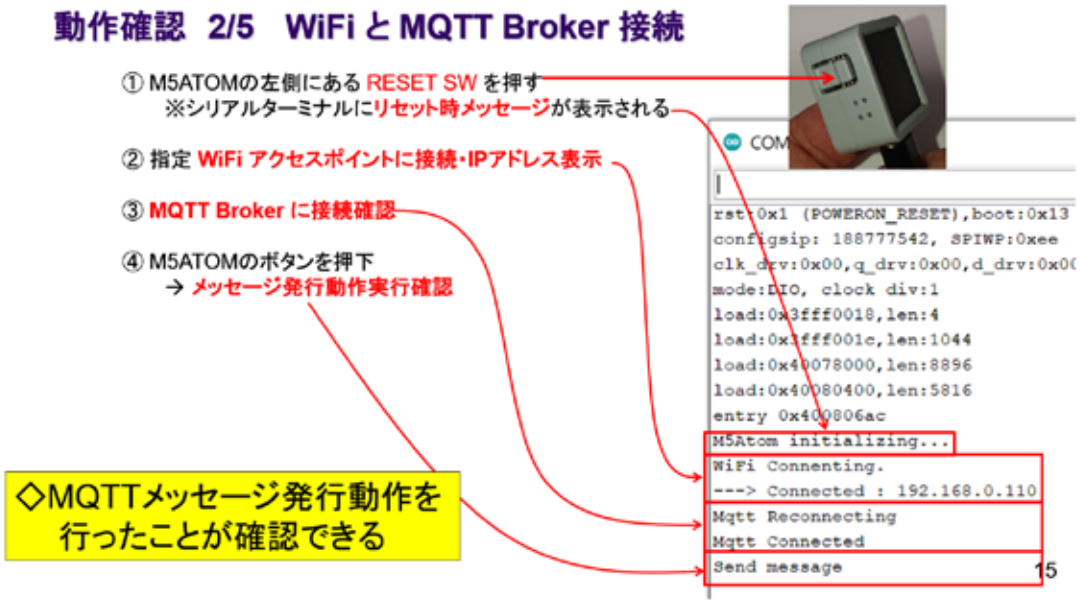


図 3-101

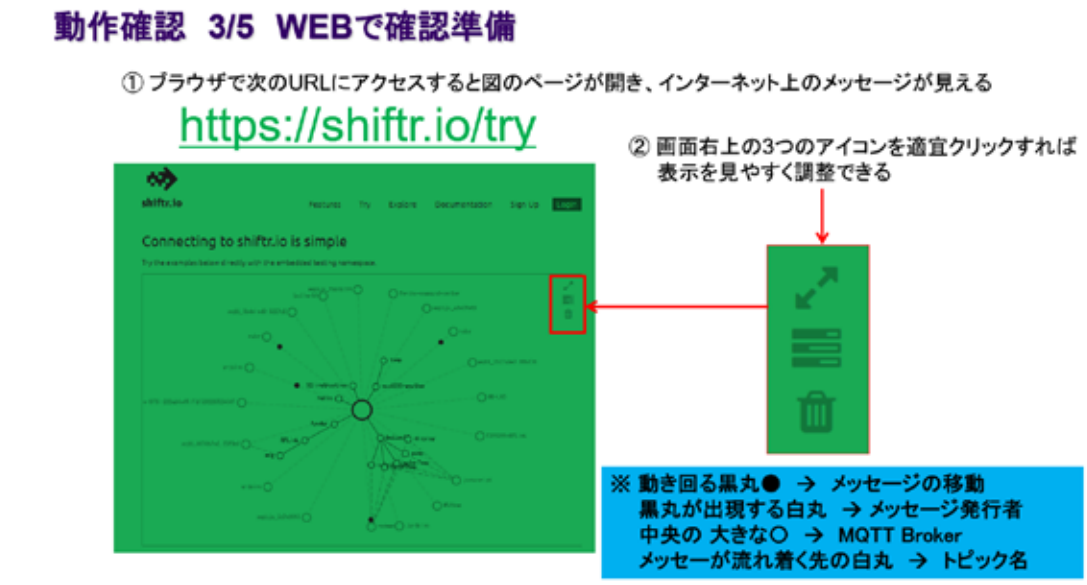


図 3-102

ここからは、MQTT でしか味わえないインターネットの世界に向けてメッセージが送信されていることを確認する。

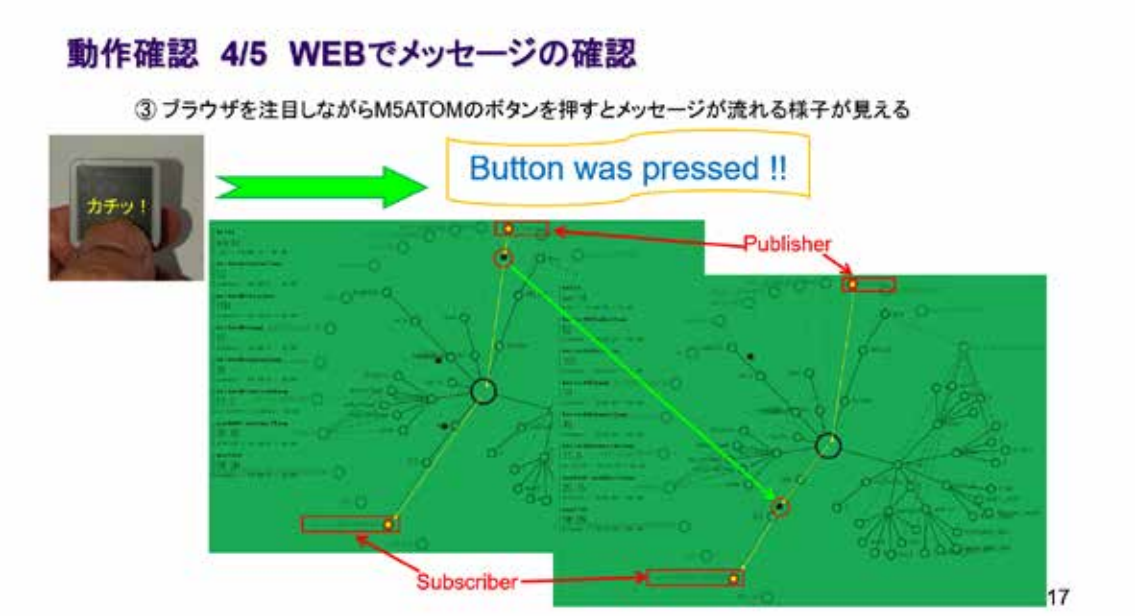


図 3-103

多くのメッセージが画面上を飛び交っている。

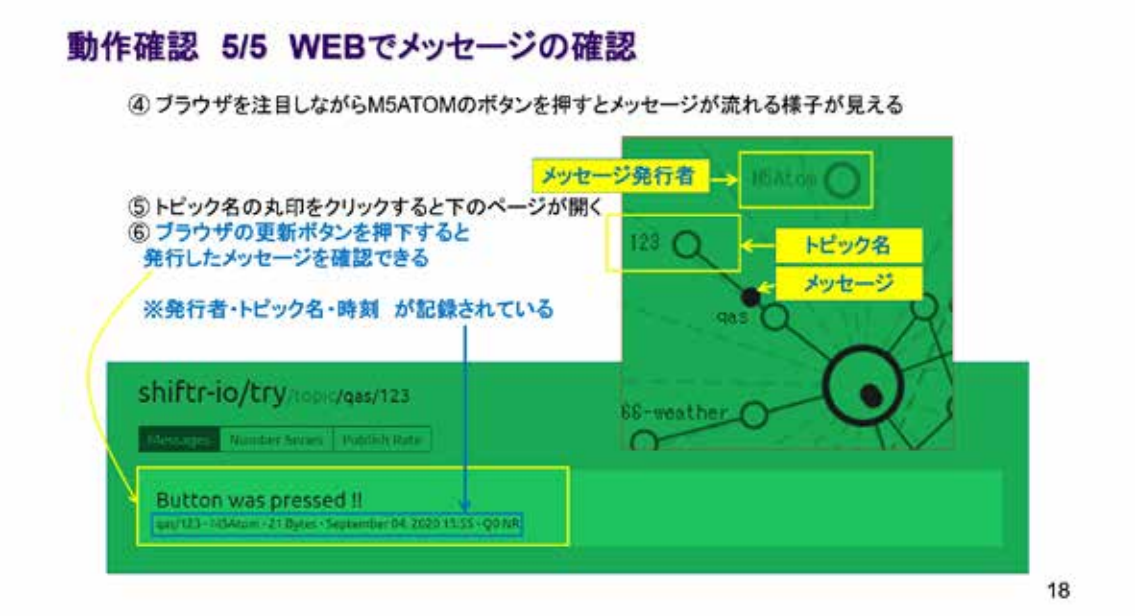


図 3-104

自身のマイコンによるメッセージが確認できるだろうか？同じ Topic 名を使用しているので、他の開発者と重なって分かり難いだろう。そのような場合は、参加者全員でミーティングし、順番に一人ずつ確認する（協調開発）ことや、ソースコードを変更して開発者独自の Topic 名を使用するなどの対応策が必要になる。MQTT は公共的なサービスなので、モニタされるメッセージが多いことを理解しておこう。

図 3-105

次はメッセージの購読を行う。

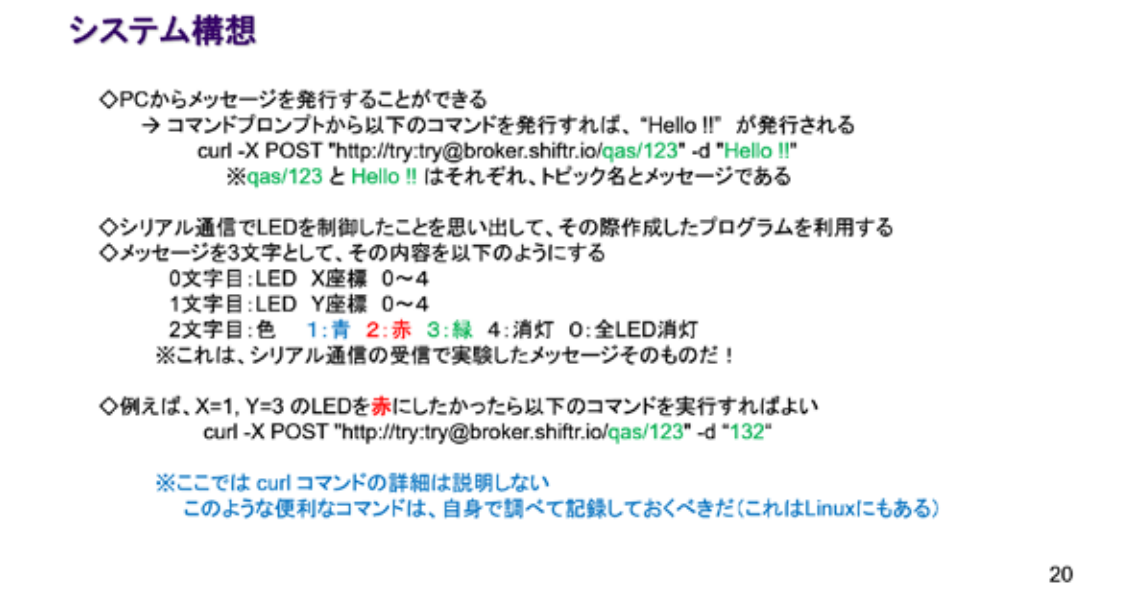


図 3-106

購読（受信）したメッセージに応じた処理を行うシステムは、インターネット経由でシステムをリモートコントロールするものだ。これは、メッセージ発行側と比べるととても長いソースコードになる。

ソースコード 1/7 （M5A_MQTT_2）

◇#include と #define 、通信関連オブジェクト部分

```
#include <M5Atom.h>                    // マイコンボードライブラリ
#include <WiFi.h>                      // WiFiライブラリ
#include <PubSubClient.h>              // MQTTクライアントライブラリ

#define WiFi_SSID "Planex_24-E68A9A"   // WiFiアクセスポイントSSID
#define WiFi_PASS  "7D438B6945"        // 同　パスワード
#define MQTT_BROKER "broker.shiftr.io" // MQTTブローカーURL
#define MQTT_PORT 1883                 // MQTT BROKER PORT
#define MQTT_CLIENT_NAME  "M5Atom"     // MQTTブローカ接続時のクライアント名
#define MQTT_USER "try"                // 公開ユーザー名（固定）
#define MQTT_PASS "try"                // 公開パスワード（固定）
#define MQTT_TOPIC "qas/123"           // TOPIC名
#define MQTT_QOS 0                     // Quality of Service（サービスの品質）
WiFiClient espClient;                  // WiFiコントロールオブジェクト
PubSubClient client(espClient);        // MQTTクライアントコントロールオブジェクト
```

21

図 3-107

ソースコード 2/7 （M5A_MQTT_2）

◇グローバル変数 と WiFiアクセスポイント接続関数

```
char flg=0;                                  // メッセージ受信フラグ　0:未受信　1:メッセージ到着
char msg[10];                                // 受信メッセージ格納用（少し大きめに確保した）

void wifi_connect(void){                     // WiFiアクセスポイントへの接続
  Serial.print("WiFi Connenting");           // 接続を試みているメッセージ
  WiFi.begin(WiFi_SSID, WiFi_PASS);          // 接続開始
  while (WiFi.status() != WL_CONNECTED) {    // アクセスポイント接続待ち
    Serial.print(".");                       // Wait時・・・表示
    delay(1000);                             // しばし待つ
  }
  Serial.print("¥n---> Connected : ");       // 接続成功メッセージ
  Serial.println(WiFi.localIP());            // 自機のIPアドレス表示
  client.setServer(MQTT_BROKER, MQTT_PORT); // MQTTブローカへの接続設定
}
```

22

図 3-108

ソースコード 3/7 （M5A_MQTT_2）

◇MQTTブローカー接続関数

```
void mqtt_connect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(MQTT_CLIENT_NAME, MQTT_USER, MQTT_PASS)) {       // MQTT接続実行！
      Serial.println("connected");                                      // つながったメッセージ
      client.subscribe(MQTT_TOPIC, MQTT_QOS);                         // メッセージ購読登録
      Serial.println("Subscribing!");                                   //  購読しているよメッセージ
    } else {                                                            // うまく行かなかった場合
      Serial.print("failed, rc=");                                     // 失敗原因コードの表示
      Serial.print(client.state());                                     // 同上
      Serial.println(" try again in 5 seconds");                        // 5秒待ちますメッセージ
      // Wait 5 seconds before retrying
      delay(5000);                                                      // しばし待つ
    }
  }
}
```

23

図 3-109

ソースコード 4/7 （M5A_MQTT_2）

◇メッセージが発行された際に呼び出される（コールバック）関数

```
void callback(char* topic, byte* payload, unsigned int length) {
  int i;

  Serial.print("Message arrived [");   // メッセージが到着したよメッセージ
  Serial.print(topic);                // 念のためトピック名表示
  Serial.print("] ");                  // 括弧閉じ
  for (i = 0; i < length; i++) {      // （メッセージ文字長文）メッセージ取り出し
    msg[i] = (char)payload[i];
    Serial.print((char)payload[i]);   // メッセージ表示
  }
  Serial.println();                   // 改行を付けて！！
  flg = 1;                            // メッセージ到着フラグをONする  loop()内でこのフラグを見て処理する
}
```

24

図 3-110

ソースコード 5/7 （M5A_MQTT_2）

◇LEDを全消灯する際に利用する関数 と 初期化

```
void fill_led(unsigned long col){                    // LEDをすべて同じ色にする
  for(int i=0; i<25; i++){
    M5.dis.drawpix(i, col);                          //
  }
}

void setup() {                                       // 初期化
  M5.begin(true, false, true);                       // SerialEnable , I2CEnable , DisplayEnable
  wifi_connect();                                    // WiFiアクセスポイント接続
  client.setCallback(callback);                      // メッセージ発行時に呼び出される関数を登録する
}
```

25

図 3-111

ソースコード 6/7 （M5A_MQTT_2）

◇通常処理全体 エラー処理とLED制御は次で説明）

```
void loop() {                                        // 通常処理
  int i;
  int x;                                             // LED X座標
  int y;                                             // LED Y座標
  int col;                                           // LED 色指定
  unsigned long color;                               // 色コード

  mqtt_connect();                                    // MQTT 接続が切れているといけないので、調べて必要なら再接続
  client.loop();                                     // MQTT接続状況更新 ・・・ これがなかなか難しい

  if(flg==1){                                        // 受信メッセージあり？
    flg = 0;                                         // フラグクリア
    x = msg[0]-'0';                                  // X座標計算
    y = msg[1]-'0';                                  // Y座標計算
    col = msg[2]-'0';                                // 色番号計算

    // エラーチェックブロック
    // LED制御ブロック
}
```

26

図 3-112

ソースコード 7/7 （M5A_MQTT_2）

◇エラーチェックブロック と LED制御ブロック

```
//エラーチェックブロック

if(x<0 || x>4){   // X座標チェック
  Serial.println("Invalid Position X!!");
  return; // エラーメッセージを送信してOSに戻る
}
if(y<0 || y>4){   // Y座標チェック
  Serial.println("Invalid Position Y!!");
  return; // エラーメッセージを送信してOSに戻る
}
if(col<0 || col>4){  // 色番号チェック
  Serial.println("Invalid Color!!");
  return; // エラーメッセージを送信してOSに戻る
}
```

```
// LED制御ブロック

if(col == 0){ // 色番号指定が0なら全LED消灯
  fill_led(color = 0x000000);  // 既に宣言した関数をCall
  return;   // LEDをすべて消灯して正誤をOSに戻す
} else if(col == 4) { // 色番号4なら指定のLEDだけ消灯
  color = 0x000000;   // 色コード　黒＝消灯
} else {
  color = 0x0000ff << (col-1)*8;  // 色コード計算　※
}
M5.dis.drawpix(x, y, color);   // 座標・色コード指定LED点灯
}

※色番号　1なら0x0000ff
          2なら0x00ff00
          3なら0xff0000
   のように、ffの位置が8ビットずつずらされる
```

27

図 3-113

マイコンボードの選択

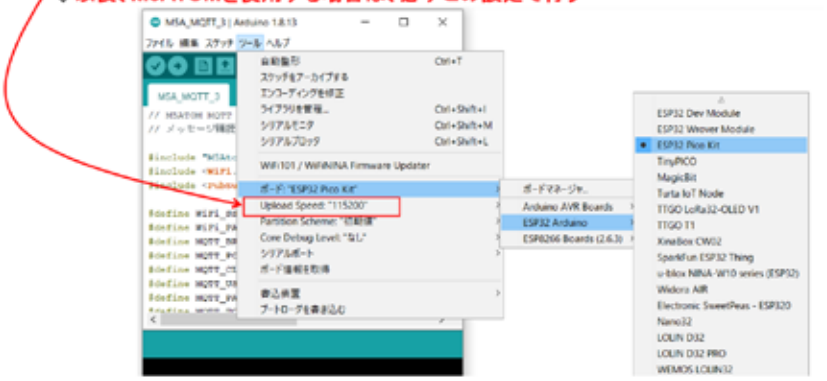
◇以下のように IDE で **ツール → ボード → ・・・とたどり、ESP32 Pico Kit** を選択する

※ボード以後の表示は、使用しているIDEの状況に応じて変わる

ボード名　ESP32 Pico Kit が無い場合、NodeMCU-32S　を探す　←M5Atom ライブラリが無い場合

◇同様に　シリアルポートの Upload Speed は、115200 にセットする（これより早いと書込みに失敗する）

◇以後、M5ATOMを使用する場合は、必ずこの設定で行う



28

図 3-114


マイコンをPCと接続
①. M5ATOMをPCと接続
PCのUSBポートに接続
開発用PC
M5ATOM
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
29

図 3-115

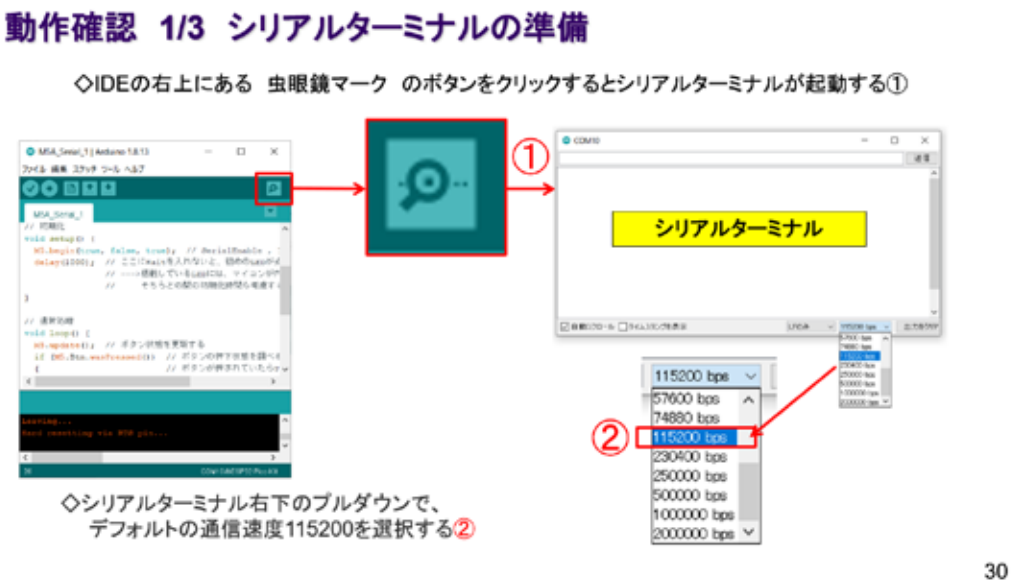

動作確認 1/3 シリアルターミナルの準備
◇IDEの右上にある 虫眼鏡マーク のボタンをクリックするとシリアルターミナルが起動する①
①
シリアルターミナル
115200 bps
57600 bps
74880 bps
115200 bps
230400 bps
250000 bps
500000 bps
1000000 bps
2000000 bps
②
◇シリアルターミナル右下のプルダウンで、
デフォルトの通信速度115200を選択する②
30

図 3-116

図 3-117

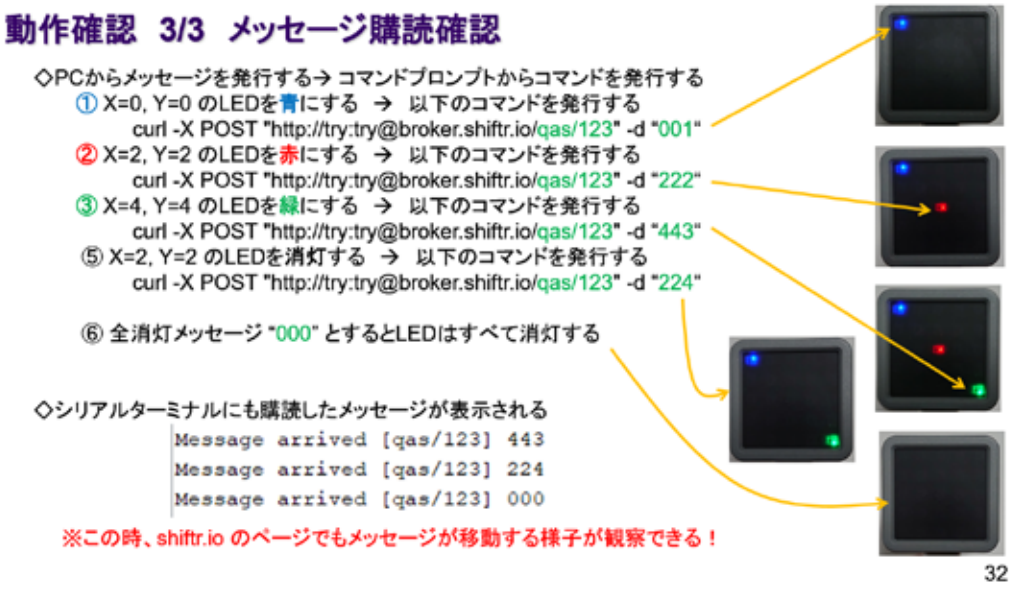


図 3-118

うまく確認できただろうか。大変多くのメッセージが混在する MQTT 環境だが、Topic 名を共通にするというたいへんシンプルなルールで、開発できるシステムの幅と奥行きが広がる。

３．６　M5Atom　－　サーボモータ

図 3-119

工場の要として、位置決めに用いられるサーボモータを制御しよう。

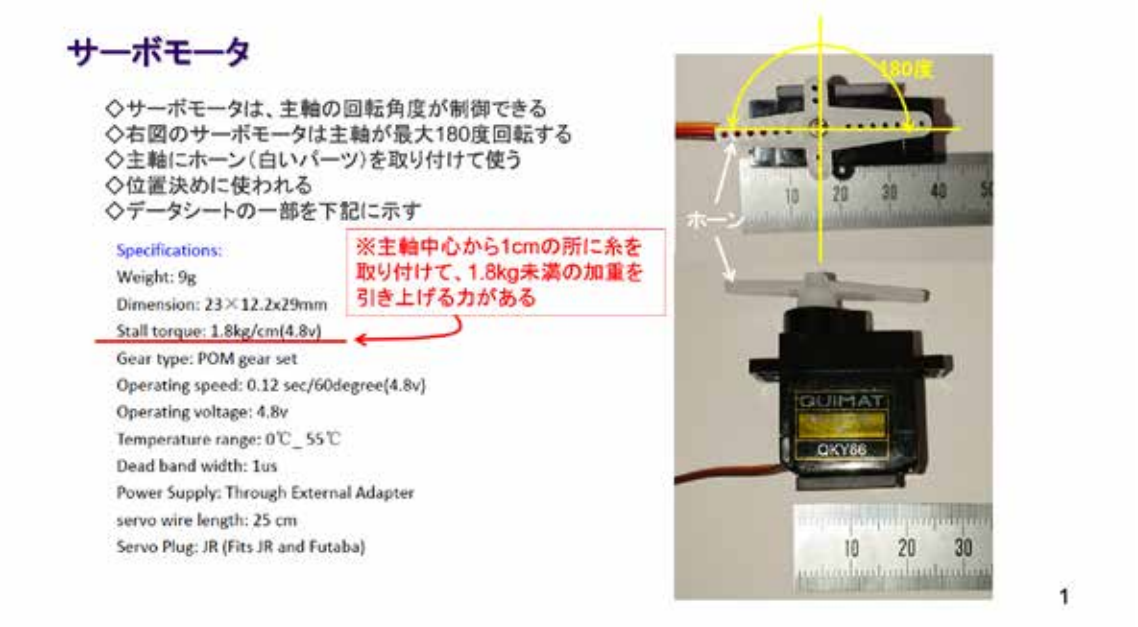


図 3-120

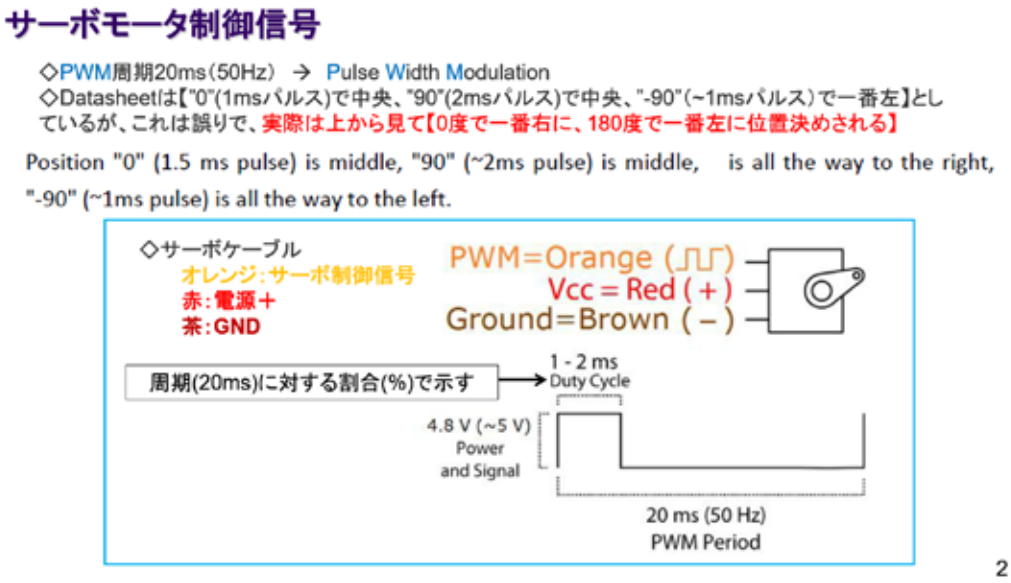


図 3-121

サーボモータの回転角度は、図の PWM 周期に対する Duty 割合で決まる。マイコンは周期的な動作が得意なので、PWM 制御には応用しやすい。※Duty には【仕事量】の意味がある。

図 3-122

モータは、一般に大きな電流が流れるので、マイコンの回路とは電源を分けることが多い。今回は、細かな配線を行わないように配慮して、電源とサーボモータドライバを兼ねるコントローラ（上図）を用いて、制御を行う。自習キットのケーブルはあらかじめ配線しておいたが、外れやすいので注意して取り扱うこと。

サーボモータを制御する

SERVO MORTOR CONTROL

4

図 3-123

μ-Factory の重要な部分を開発する。

目論見 → モデル工場の設備振動シミュレーション

◇M5ATOMからサーボコントローラを介してPWMパルスを出力し、サーボモータを左右に回転運動させる

→ 加速度センサを載せて振動を測定する

→ **モデル工場の設備振動シミュレーション**

M5ATOM　　サーボコントローラ　　サーボモータ

5

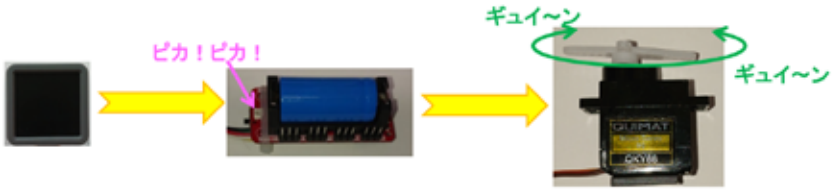
図 3-124

システム構想

◇M5ATOMによるサーボモータ制御 → 0度から180度の往復回転運動を繰り返す

①. IIC_servo ライブラリ (servo.cpp + servo.h) が提供されている
※ソースコードと同一フォルダで同時にコンパイル → ライブラリインストールの必要なし
※別マイコン(M5Stick-C)用のライブラリをM5ATOM用に変更したもの

②. 次の関数が使える
- IIC_Servo_Init() → コントローラの初期化
- Servo_angle_set(CH, 角度) → サーボモータ角度制御 CHは1~8、角度は0~180で指定
- RGB_set(R,G,B) → LED点灯・・・R,G,Bの明るさは、それぞれ0~255で指定



6

図 3-125

ここで用いるサーボライブラリは、実習キットに含まれているソースコードの同一フォルダに含まれているので、別にインストールする必要はない。使用している IDE では、同じフォルダに存在するソースコードファイルは、コンパイルからリンクまで同時に行われる。もし、既存のライブラリは残しつつ、ライブラリの変更をしたい場合は、ライブラリのソースコードファイルを、開発対象ソースコードと同じフォルダにコピー（移動ではなく）して、変更すれば、元のライブラリはそのまま残すことができる。その場合は、#include のヘッダファイル名を”　”で囲む。

ソースコード 1/2 （M5A_Servo_1）

◇初期化処理部

```
#include "M5Atom.h"        // " "は< >でも良い
#include "IIC_servo.h"     // ヘッダファイル名を" "で囲むこと

// 内蔵加速度センサが利用するGPIOポートを開放する → コントローラICとの通信に使用する
bool IMU6886Flag = false; //sda:25 and scl:21 are free.

void setup(){  // 初期化
  M5.begin(true, true, false); //serial, I2C, LED
  Serial.printf("¥n  IIC_servo¥n");
  IIC_Servo_Init();     //sda:21  scl:25 Servo Controler用にIICを設定（初めに開放したポート）
                        // IICというのは、マイコンとデバイスを通信で結ぶI/F
}
```

7

図 3-126

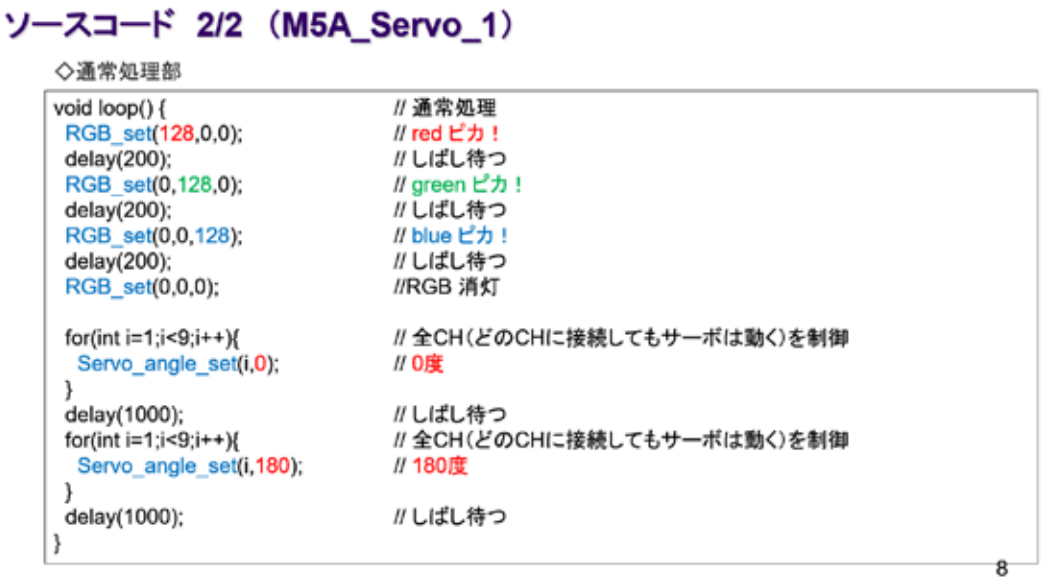


図 3-127

今回利用しているのは、Servo_angle_set（角度）関数で、直接角度を指定できる。使用するサーボモータは、0～180 度の回転位置が指定できるので、その範囲で往復回転させている。

図 3-128

コンパイルからマイコンへの書き込みまでは、連続して行われる。書込みが完了すると自動的に動作が始まるので、あらかじめ全体を接続してサーボモータの動作に備えておく。

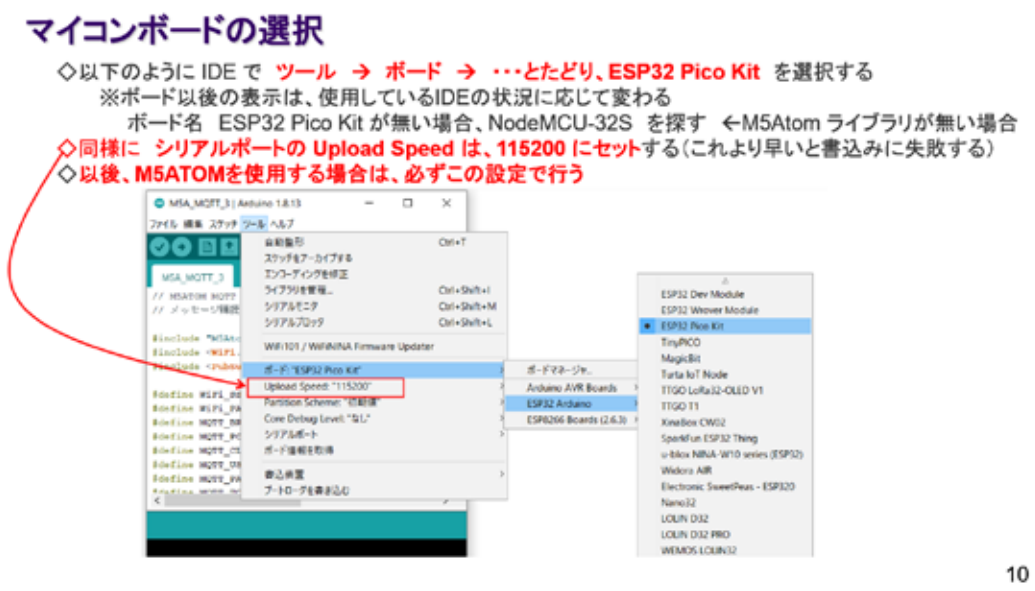

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、ESP32 Pico Kit を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に シリアルポートの Upload Speed は、115200 にセットする（これより早いと書込みに失敗する）
◇以後、M5ATOMを使用する場合は、必ずこの設定で行う
10

図 3-129


マイコンをPCと接続
①. M5ATOMをPCと接続
PCのUSBポートに接続
開発用PC
M5ATOM
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
11

図 3-130

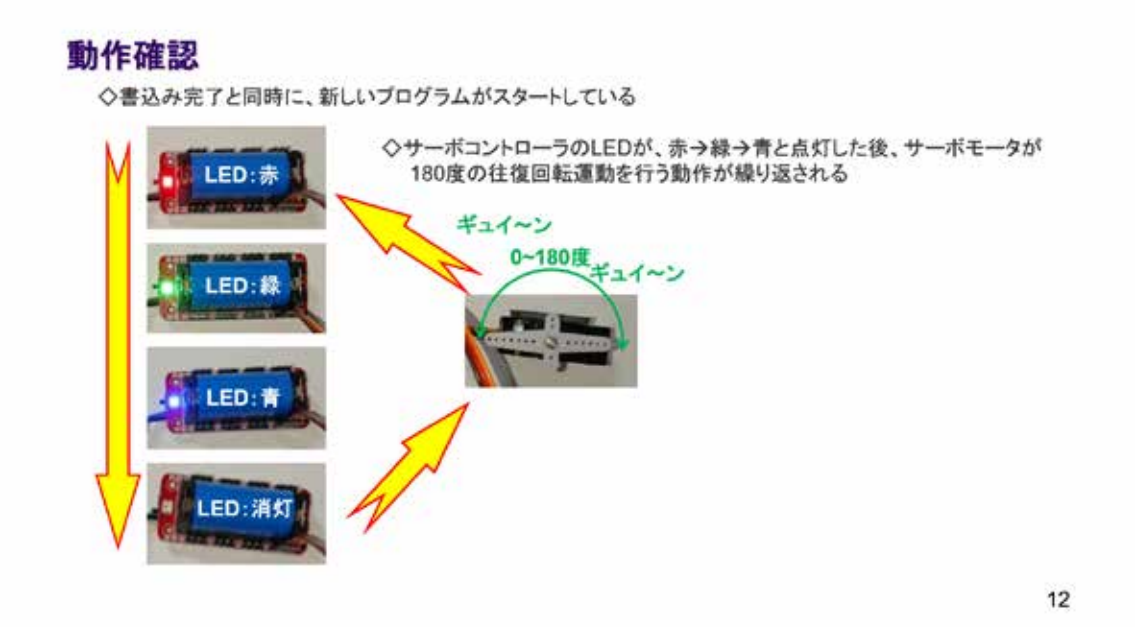


図 3-131

実習キットには、スチールプレートが含まれている。サーボモータには、小さなマグネットが貼り付けてある。これらを利用してサーボモータを簡易固定すると、動作確認が行いやすい。一定の時間間隔でサーボモータが往復回転運動することが確認できる。

図 3-132

MQTT の技術を生かして、インターネット経由でサーボモータをリモートコントロールすることを実証しよう。

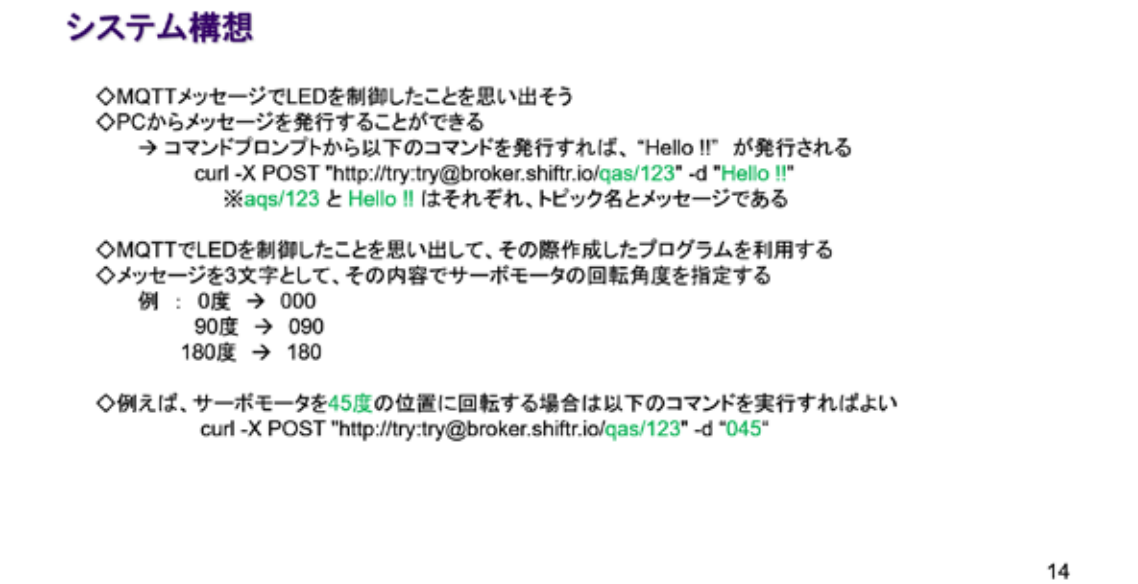


図 3-133

メッセージを購読（受信）することになるので、ソースコードは長くなる。

ソースコード 1/6 （ M5A_Servo_2 ）

◇#include と #define 、通信関連オブジェクト部分

```
#include <M5Atom.h>                          // マイコンボードライブラリ
#include <WiFi.h>                            // WiFiライブラリ
#include <PubSubClient.h>                    // MQTTクライアントライブラリ
#include "IIC_servo.h"                       // サーボ制御ライブラリ  " " でヘッダファイル名を囲むこと
#define WiFi_SSID "Planex_24-E68A9A"         // WiFiアクセスポイントSSID
#define WiFi_PASS  "7D438B6945"              // 同  パスワード
#define MQTT_BROKER "broker.shiftr.io"       // MQTTブローカーURL
#define MQTT_PORT 1883                       // MQTT BROKER PORT
#define MQTT_CLIENT_NAME  "M5Atom"           // MQTTブローカ接続時のクライアント名
#define MQTT_USER "try"                      // 公開ユーザー名（固定）
#define MQTT_PASS "try"                      // 公開パスワード（固定）
#define MQTT_TOPIC "qas/123"                 // TOPIC名
#define MQTT_QOS 0                           // Quality of Service（サービスの品質）
WiFiClient espClient;                        // WiFIコントロールオブジェクト
PubSubClient client(espClient);              // MQTTクライアントコントロールオブジェクト
// 内蔵加速度センサが利用するGPIOポートを開放する
bool IMU6886Flag = false; //sda:25 and scl:21 are free.
```

15

図 3-134

よく見ると、以前目にした記述がほとんどである。

ソースコード 2/6 （ M5A_Servo_2 ）

◇グローバル変数 と WiFiアクセスポイント接続関数

```
char flg=0;                                  // メッセージ受信フラグ  0:未受信  1:メッセージ到着
char msg[10];                                // 受信メッセージ格納用（少し大きめに確保した）

void wifi_connect(void){                     // WiFiアクセスポイントへの接続
  Serial.print("WiFi Connenting");           // 接続を試みているメッセージ
  WiFi.begin(WiFi_SSID, WiFi_PASS);          // 接続開始
  while (WiFi.status() != WL_CONNECTED) {    // アクセスポイント接続待ち
    Serial.print(".");                       // Wait時・・・表示
    delay(1000);                             // しばし待つ
  }
  Serial.print("¥n---> Connected : ");       // 接続成功メッセージ
  Serial.println(WiFi.localIP());            // 自機のIPアドレス表示
  client.setServer(MQTT_BROKER, MQTT_PORT); // MQTTブローカへの接続設定
}
```

16

図 3-135

ソースコード 3/6 （ M5A_Servo_2 ）

◇MQTTブローカー接続関数

```
void mqtt_connect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(MQTT_CLIENT_NAME, MQTT_USER, MQTT_PASS)) {       // MQTT接続実行！
      Serial.println("connected");                                      // つながったメッセージ
      client.subscribe(MQTT_TOPIC, MQTT_QOS);                          // メッセージ購読登録
      Serial.println("Subscribing!");                                   //  購読しているよメッセージ
    } else {                                                           // うまく行かなかった場合
      Serial.print("failed, rc=");                                     // 失敗原因コードの表示
      Serial.print(client.state());                                    // 同上
      Serial.println(" try again in 5 seconds");                       // 5秒待ちますメッセージ
      // Wait 5 seconds before retrying
      delay(5000);  // しばし待つ
    }
  }
}
```

17

図 3-136

ソースコード 4/6 （ M5A_Servo_2 ）

◇メッセージが発行された際に呼び出される（コールバック）関数

```
void callback(char* topic, byte* payload, unsigned int length) {
  int i;

  Serial.print("Message arrived [");   // メッセージが到着したよメッセージ
  Serial.print(topic);                // 念のためトピック名表示
  Serial.print("] ");                  // 括弧閉じ
  for (i = 0; i < length; i++) {      // （メッセージ文字長文）メッセージ取り出し
    msg[i] = (char)payload[i];
    Serial.print((char)payload[i]);   // メッセージ表示
  }
  Serial.println();                   // 改行を付けて！！
  flg = 1;   // メッセージ到着フラグをONする  loop()内でこのフラグを見て処理する
}
```

18

図 3-137

ソースコード 5/6 （ M5A_Servo_2 ）

◇初期化関数

```
void setup() {                          // 初期化
  M5.begin(true, true, true);           // SerialEnable , I2CEnable , DisplayEnable
  wifi_connect();                        // WiFiアクセスポイント接続
  client.setCallback(callback);         // メッセージ発行時に呼び出される関数を登録する
  mqtt_connect();                       // MQTT 再接続・・・（接続が切れていたら）

  Serial.printf("¥n  MQTT_Servo Control-2¥n"); // シリアルポートにメッセージ出力
  IIC_Servo_Init();                      //sda:21  scl:25 Servo Controler用にIICを設定
}
```

19

図 3-138

ソースコード 6/6 （ M5A_Servo_2 ）

◇通常処理全体　エラー処理とLED制御は次で説明）

```
void loop() {                           // 通常処理
  int i;
  int a;                                // サーボモータ角度

  mqtt_connect();                       // MQTT 接続が切れているといけないので、調べて必要なら再接続
  client.loop();                        // MQTT接続状況更新　・・・　これがなかなか難しい

  if(flg==1){                           // 受信メッセージあり？
    flg = 0;                            // フラグクリア

    a = (msg[0]-'0')*100 +(msg[1]-'0')*10 +(msg[2]-'0'); // サーボモータ回転角度計算

    // エラーチェックブロック
    // LED点滅ブロック
    // サーボモータコントローラ出力 ブロック
  }
}
```

20

図 3-139

購読したメッセージからモータの回転角度を計算している。図の青字で記述したコードは、後のページで示す。

図 3-140

ソースコードは長いが、ブロックで処理内容を見ると、難しいことは行っていない。記述した通りにシステムは動く。

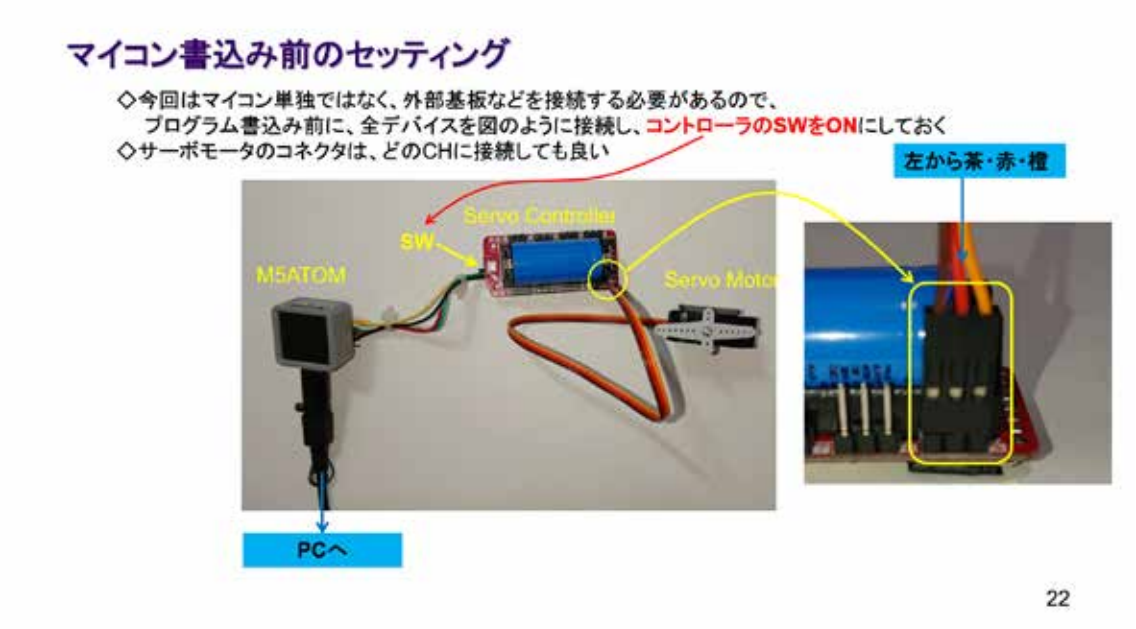


図 3-141

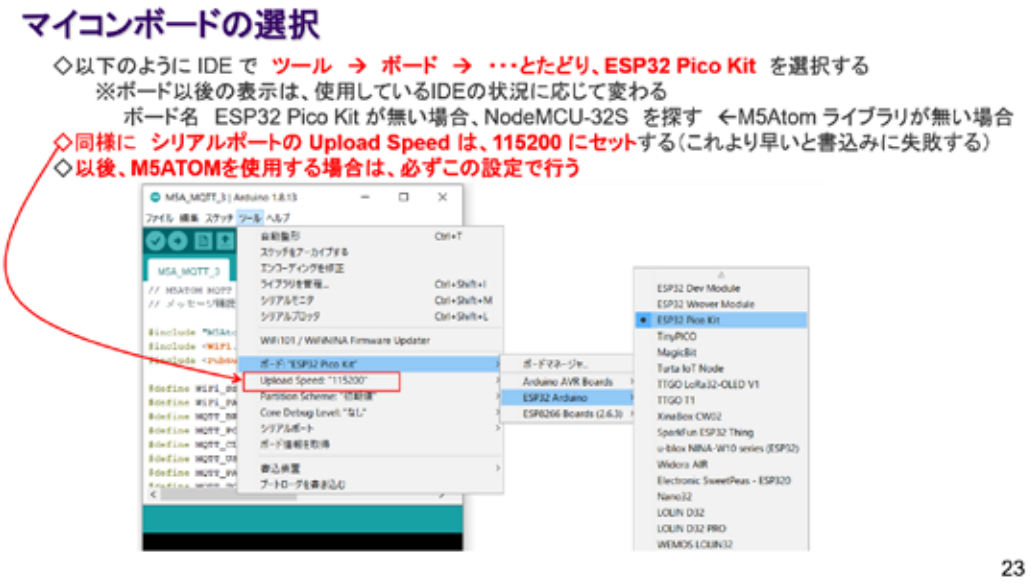

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、ESP32 Pico Kit を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に シリアルポートの Upload Speed は、115200 にセットする（これより早いと書込みに失敗する）
◇以後、M5ATOMを使用する場合は、必ずこの設定で行う
23

図 3-142


マイコンをPCと接続
①. M5ATOMをPCと接続
PCのUSBポートに接続
開発用PC
M5ATOM
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
24

図 3-143

図 3-144

curl コマンド（MQTT Broker の WEB ページで示されている）を用いてメッセージを発行し、サーボモータが意図したように動作することを確認する。この際、0～180 度の範囲内で、自由に角度指定できることも確認しておこう。

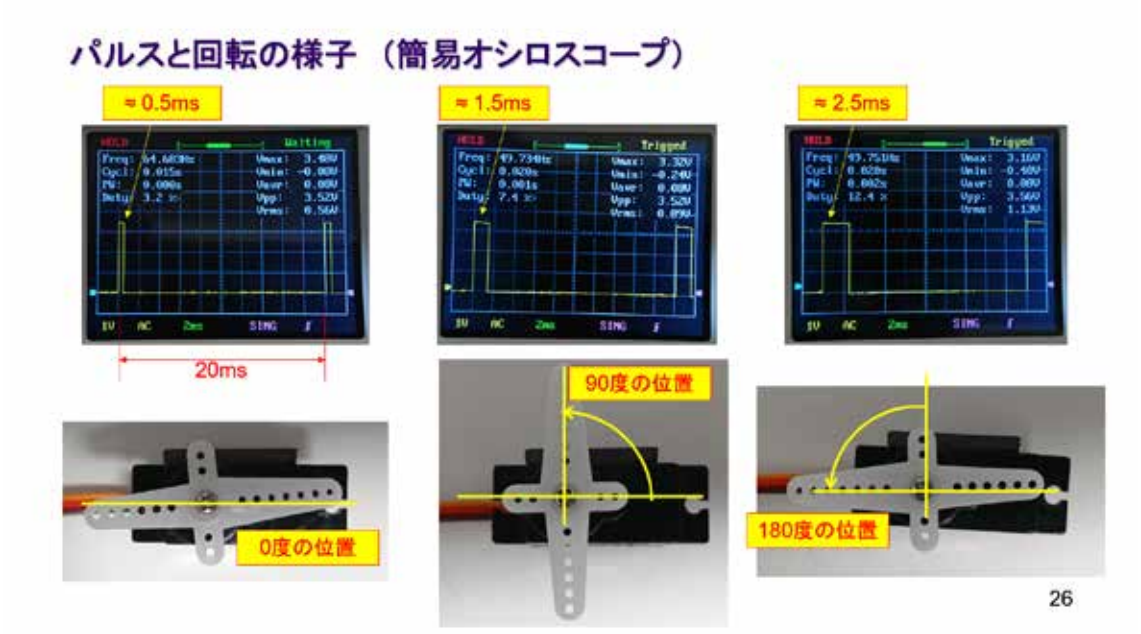


図 3-145

簡易オシロスコープで、サーボモータコントローラの出力信号を観察した様子を図に示している。PWM で制御される回転角度と Duty の対応がおおよそ観察できる。

３．７　M5Atom　－　Bluetooth

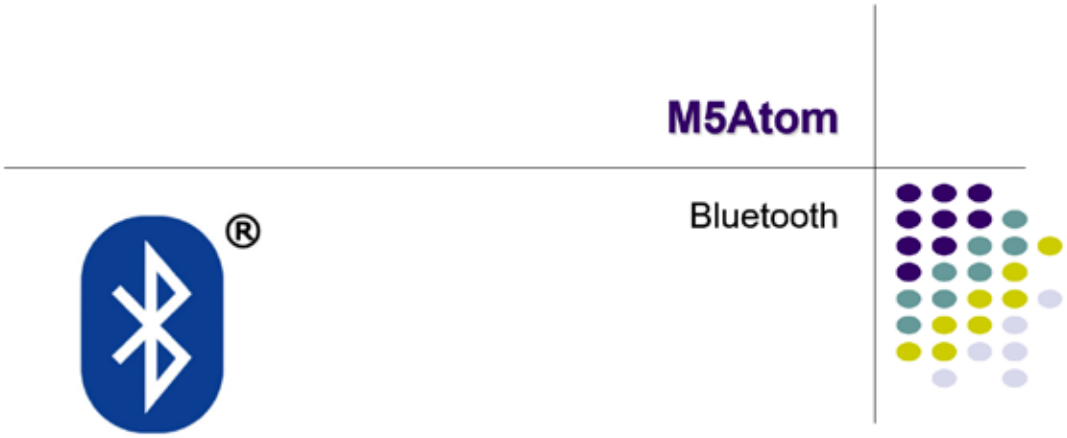


図 3-146

近距離の無線通信に Bluetooth が使えると IoT では大変有利である。

図 3-147

Bluetooth 規格には Class があるが、M5 シリーズマイコンは Class1（約 100m の通信距離）に対応している。

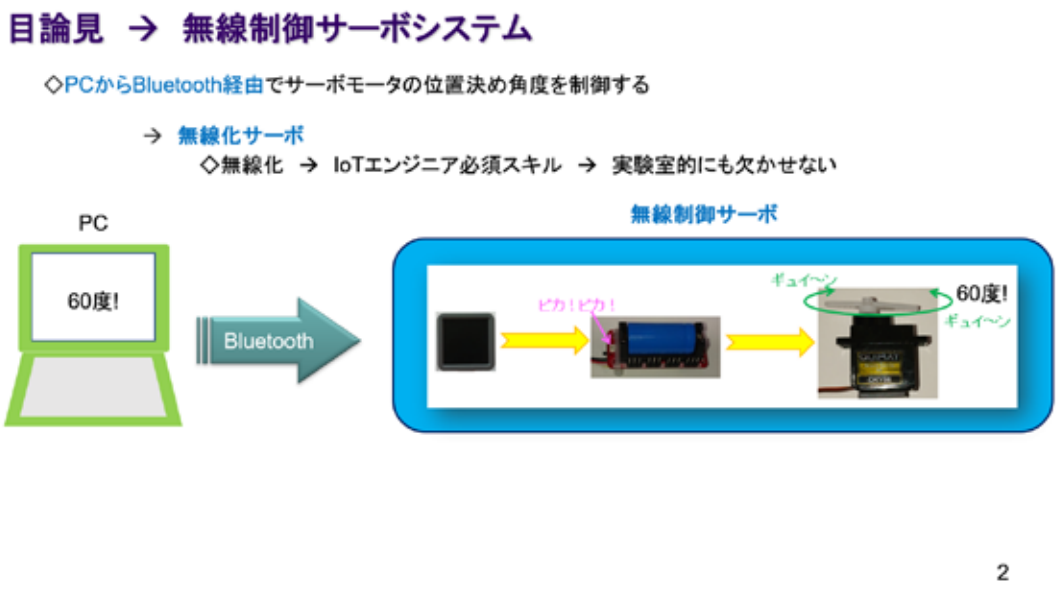


図 3-148

Bluetooth 通信で PC からサーボモータをコントロールしてみよう。

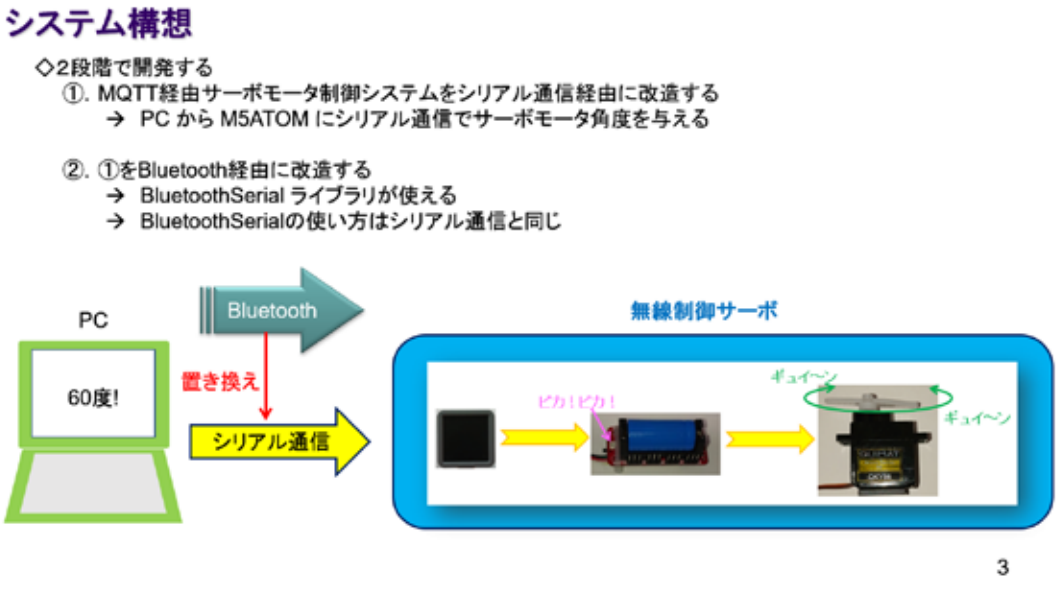


図 3-149

Bluetooth 用に、BluetoothSerial ライブラリが利用できる。過去の開発資産を活用しながら段階的に開発を進めることにする。

図 3-150

まず、シリアル通信を利用してサーボモータをリモートコントロールする。
ここまで実習を続けていれば、以下のソースコードに対する解説は不要だと思う。

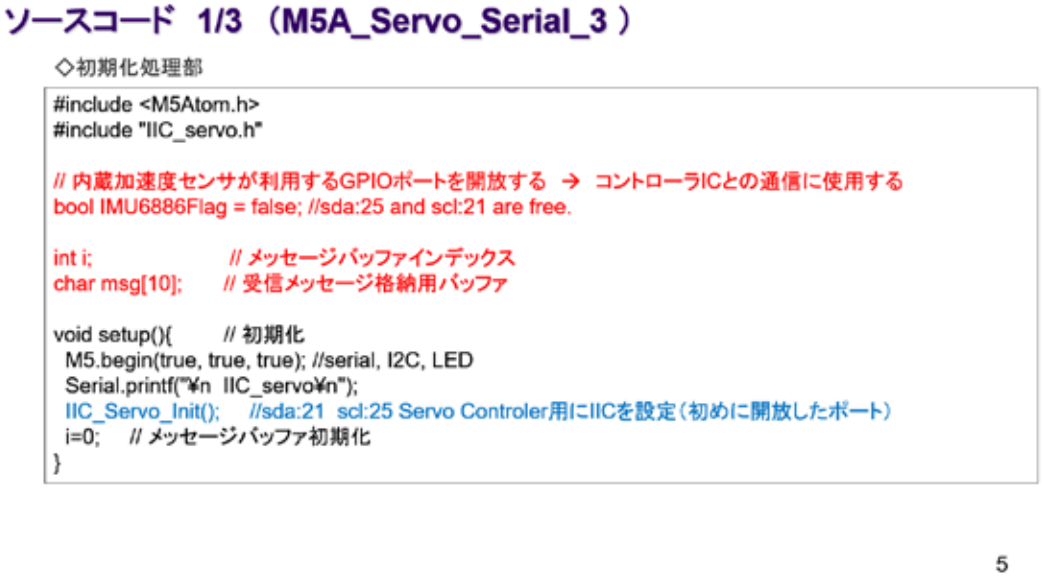


図 3-151

ソースコード 2/3 （M5A_Servo_Serial_3）

◇通常処理部

```
void loop() {                                   // 通常処理
  int a;                                        // サーボモータ角度
  char c;                                       // 受信データ 1文字分

  if(Serial.available()){                       // 受信データあり？
    c = Serial.read();                          // 1文字 Read
    msg[i++] = c;                               // 受信した文字を格納
    if(c == '¥n'){                              // 改行ならば電文の終端
      Serial.print("Received message ---> ");   // 受信した角度表示
      Serial.print(msg);                        // 同上 角度部
      i=0;                                      // メッセージバッファ初期化
      a = (msg[0]-'0')*100 +(msg[1]-'0')*10 +(msg[2]-'0'); // 角度計算

      //エラーチェック部
      //LED点滅部
      //サーボ制御部
    }
  }
}
```

6

図 3-152

ソースコード 3/3 （M5A_Servo_Serial_3）

◇エラーチェック部

```
if(a<0 || a>180){ // 角度チェック 0° から180° の範囲外はエラー
      Serial.println("Over Rotation Angle!!");
      return; // エラーメッセージを送信してOSに戻る
}
```

◇LED点滅部

```
RGB_set(128,0,0);       // red ピカ！
delay(200);             // しばし待つ
RGB_set(0,128,0);       // green ピカ！
delay(200);             // しばし待つ
RGB_set(0,0,128);       // blue ピカ！
delay(200);             // しばし待つ
RGB_set(0,0,0);         //RGB 消灯
```

◇サーボ制御部

```
for(int j=1;j<9;j++){           // 全CH（どのピンに接続してもサーボが動く）
      Servo_angle_set( j, a );  // a=指定角度
}
```

7

図 3-153

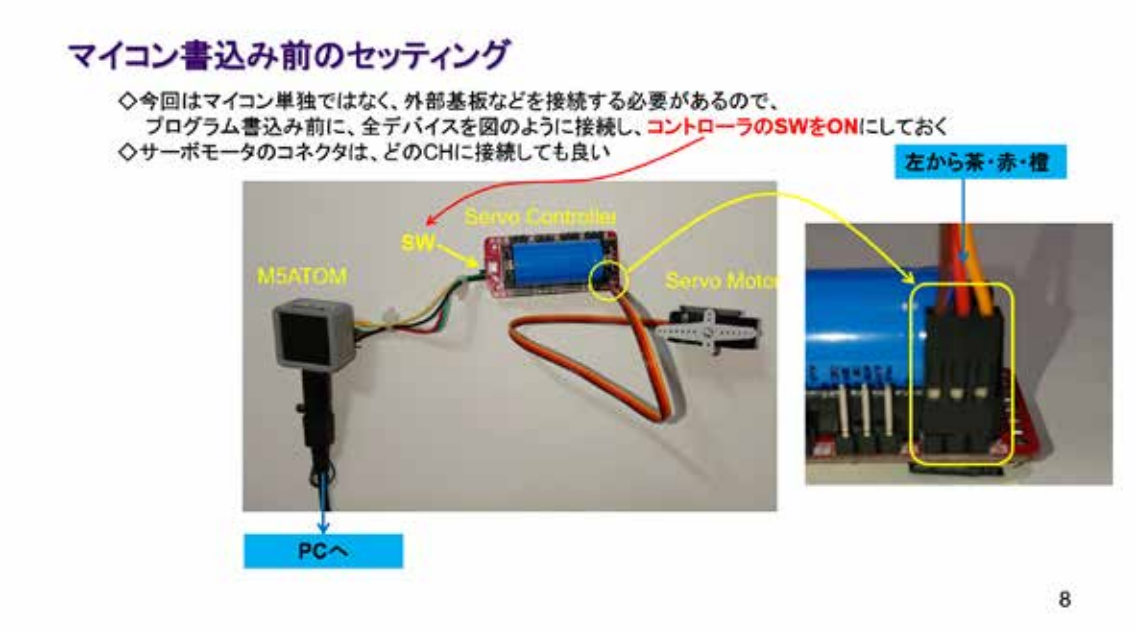

マイコン書込み前のセッティング
◇今回はマイコン単独ではなく、外部基板などを接続する必要があるので、
プログラム書込み前に、全デバイスを図のように接続し、コントローラのSWをONにしておく
◇サーボモータのコネクタは、どのCHに接続しても良い
左から茶・赤・橙
Servo Controller
SW
M5ATOM
Servo Motor
PCへ
8

図 3-154

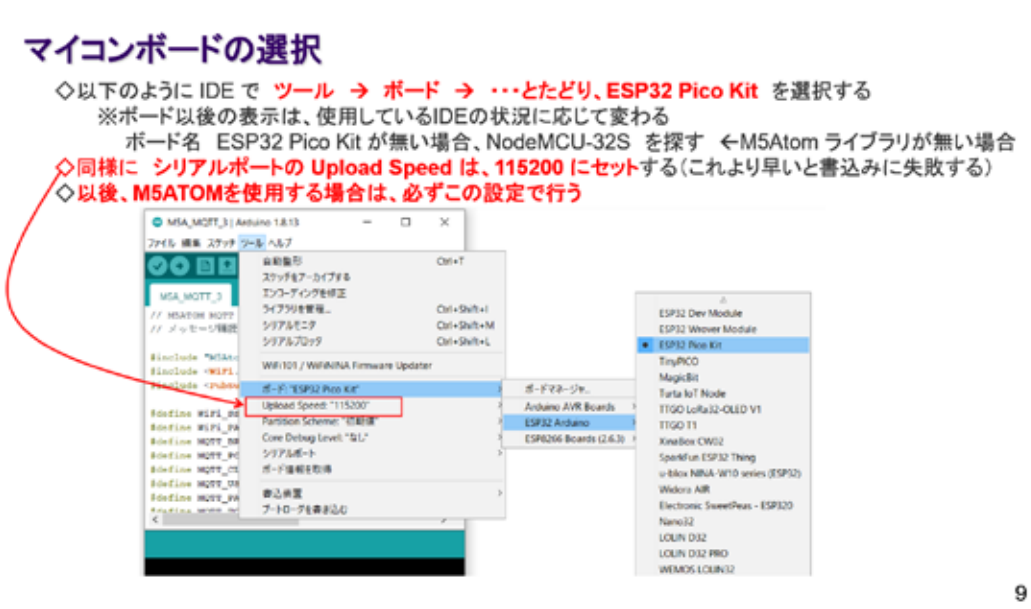

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、ESP32 Pico Kit を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に シリアルポートの Upload Speed は、115200 にセットする(これより早いと書込みに失敗する)
◇以後、M5ATOMを使用する場合は、必ずこの設定で行う
Upload Speed: "115200"
ESP32 Pico Kit
9

図 3-155

図 3-156

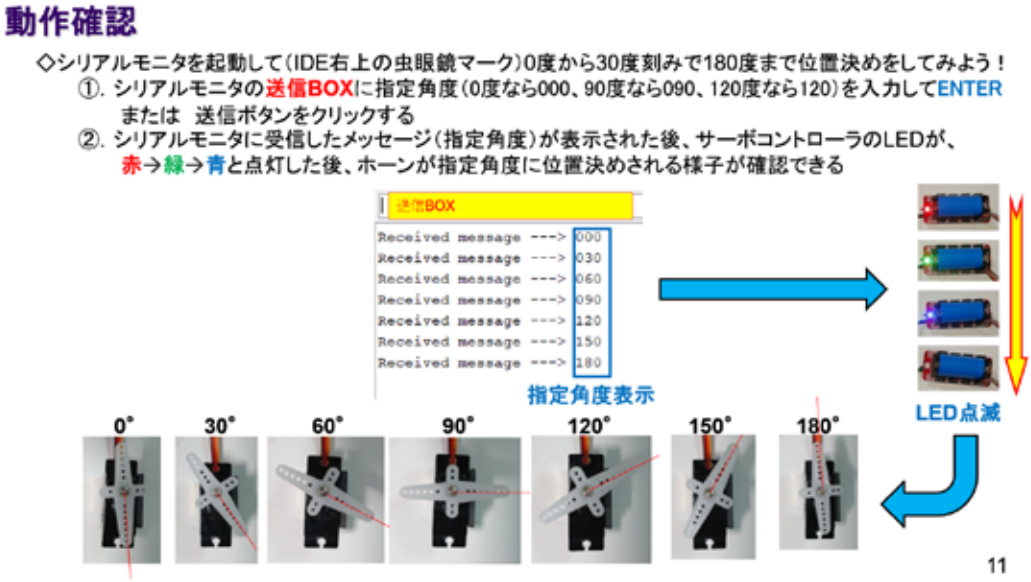


図 3-157

PC からの指令（メッセージ）に応じたモータの回転角度制御ができた。

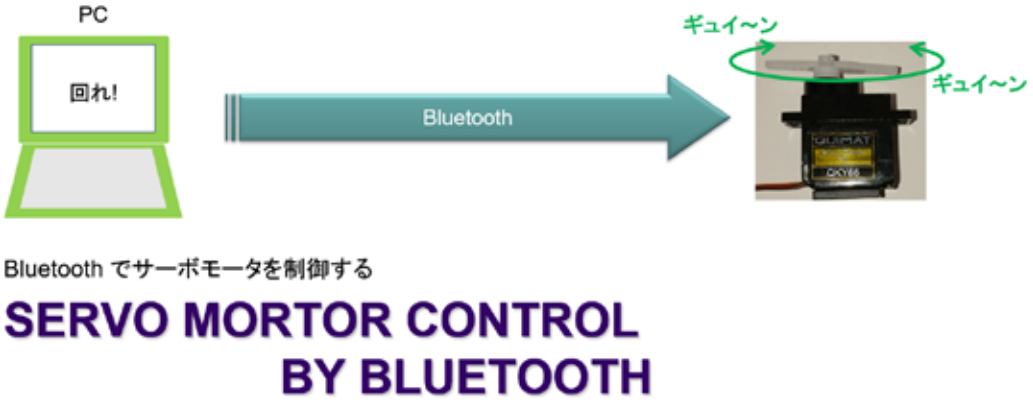


図 3-158

マイコンがシリアル通信で行ったのは、受信したメッセージに応じた処理である。これを Bluetooth に置き換える。

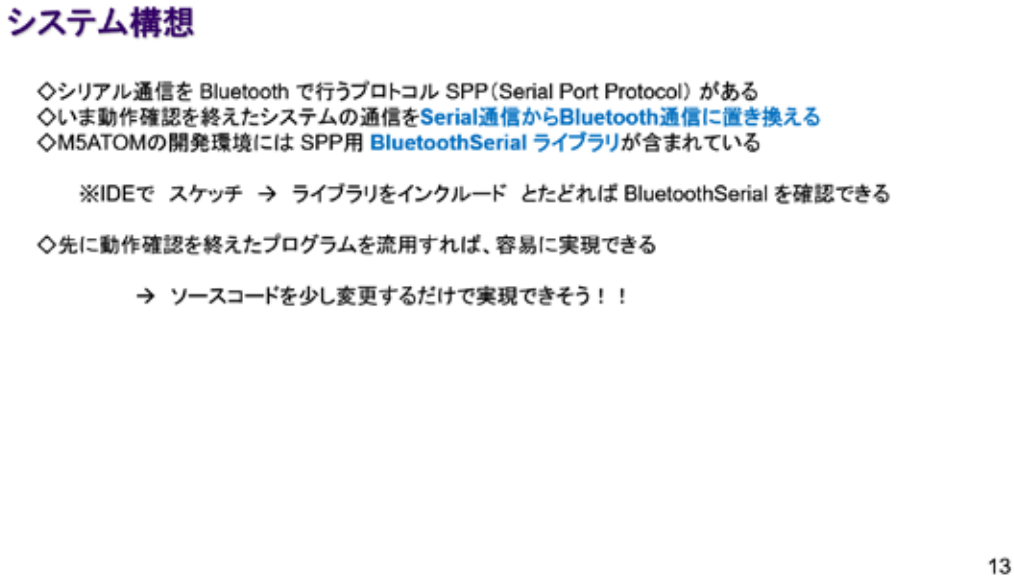


図 3-159

直前に開発したソースコードを一部変更して、Bluetooth 対応システムを開発する。以下、追加・変更するソースコード部分のみ青字で示している。

ソースコード 1/3 （M5A_Servo_Bluetooth_Serial_4 ）

◇初期化処理部

※青字部分を追加･変更する

```
#include <M5Atom.h>
#include "IIC_servo.h"
#include <BluetoothSerial.h>
BluetoothSerial bts;                    // Bluetooth Serial Object
// 内蔵加速度センサが利用するGPIOポートを開放する → コントローラICとの通信に使用する
bool IMU6886Flag = false;               //sda:25 and scl:21 are free.
int i;                                  // メッセージバッファインデックス
char msg[10];                           // 受信メッセージ格納用バッファ

void setup(){                           // 初期化
  M5.begin(true, true, true);            //serial, I2C, LED
  Serial.printf("¥n  IIC_servo¥n");
  IIC_Servo_Init();           //sda:21  scl:25 Servo Controler用にIICを設定（初めに開放したポート）
  i=0;                                  // メッセージバッファインデックス初期化
  bts.begin("＊＊＊＊＊"); // PC側でペアリングするデバイス名称  ※各自ユニークな名称にする！！
  delay(1000);                          // しばし待つ
  Serial.print("  Servo Bluetooth Serial Control-4¥n"); // Serial Terminal Message
  bts.print("  Servo Bluetooth Serial Control-4¥n");   // Bluetooth Terminal Message
}
```

14

図 3-160

ソースコード 2/3 （M5A_Servo_Bluetooth_Serial_4 ）

◇通常処理部

```
void loop() {                                   // 通常処理
  int a;                                        // サーボモータ角度
  char c;                                       // 受信データ 1文字分
   if(bts.available()){                         // Bluetoothに受信データあり？
    c = bts.read();                             // Bluetooth から1文字 Read
    msg[i++] = c;                               // 受信した文字を格納
    if(c == '¥n'){                              // 改行ならば電文の終端
      Serial.print("Received message ---> ");   // 受信した角度表示
      Serial.print(msg);                        // 同上 角度部
      bts.print("Received message ---> ");      // Bluetooth Terminal にも表示
      bts.print(msg);
      i=0;                                      // メッセージバッファ初期化
      a = (msg[0]-'0')*100 +(msg[1]-'0')*10 +(msg[2]-'0'); // 角度計算（文字コード→数値）
      //エラーチェック部
      //LED点滅部
      //サーボ制御部
    }
  }
}
```

15

図 3-161

追加・変更するソースコード部分は、ここまで。他は、そのまま。

ソースコード 3/3 （M5A_Servo_Bluetooth_Serial_4 ）

◇エラーチェック部

```
if(a<0 || a>180){                       // 角度チェック 0° から180° の範囲外はエラー
    bts.println("Over Rotation Angle!!");  // Bluetooth にエラーメッセージ出力
    return;                             // エラーメッセージを送信してOSに戻る
}
```

◇LED点滅部

```
RGB_set(128,0,0);          // red ピカ！
delay(200);                // しばし待つ
RGB_set(0,128,0);          // green ピカ！
delay(200);                // しばし待つ
RGB_set(0,0,128);          // blue ピカ！
delay(200);                // しばし待つ
RGB_set(0,0,0);            //RGB 消灯
```

◇サーボ制御部

```
for(int j=1;j<9;j++){      // 全CH（どのピンに接続してもサーボが動く）
    Servo_angle_set( j, a );  // a=指定角度
}
```

16

図 3-162

マイコン書込み前のセッティング

◇今回はマイコン単独ではなく、外部基板などを接続する必要があるので、
プログラム書込み前に、全デバイスを図のように接続し、コントローラのSWをONにしておく
◇サーボモータのコネクタは、どのCHに接続しても良い

17

図 3-163

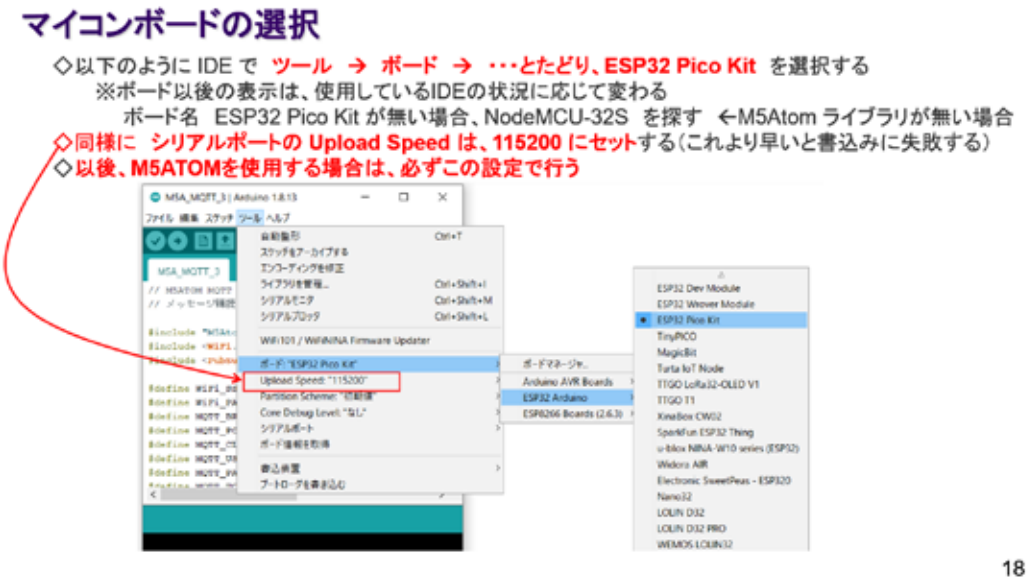

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、ESP32 Pico Kit を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に シリアルポートの Upload Speed は、115200 にセットする（これより早いと書込みに失敗する）
◇以後、M5ATOMを使用する場合は、必ずこの設定で行う
18

図 3-164


マイコンをPCと接続
①. M5ATOMをPCと接続
PCのUSBポートに接続
開発用PC
M5ATOM
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
19

図 3-165

図 3-166

Bluetooth は、通信を行う双方でペアリングが必要である。図に従い、PC 側でペアリング設定を行うこと。

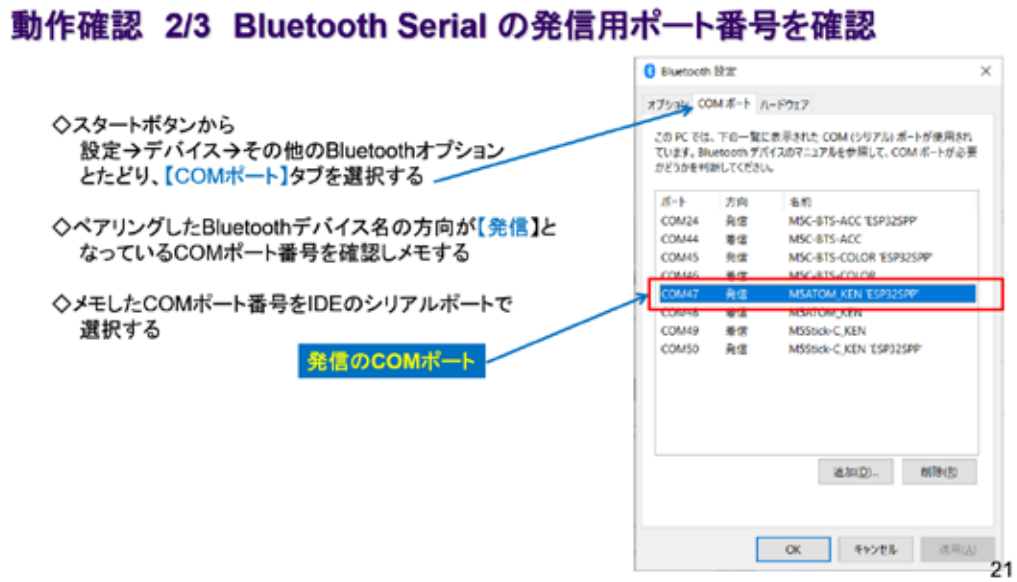


図 3-167

Bluetooth のペアリングが完了すると、COM ポート番号が割り当てられる。その番号を用いて動作確認しなければならない。

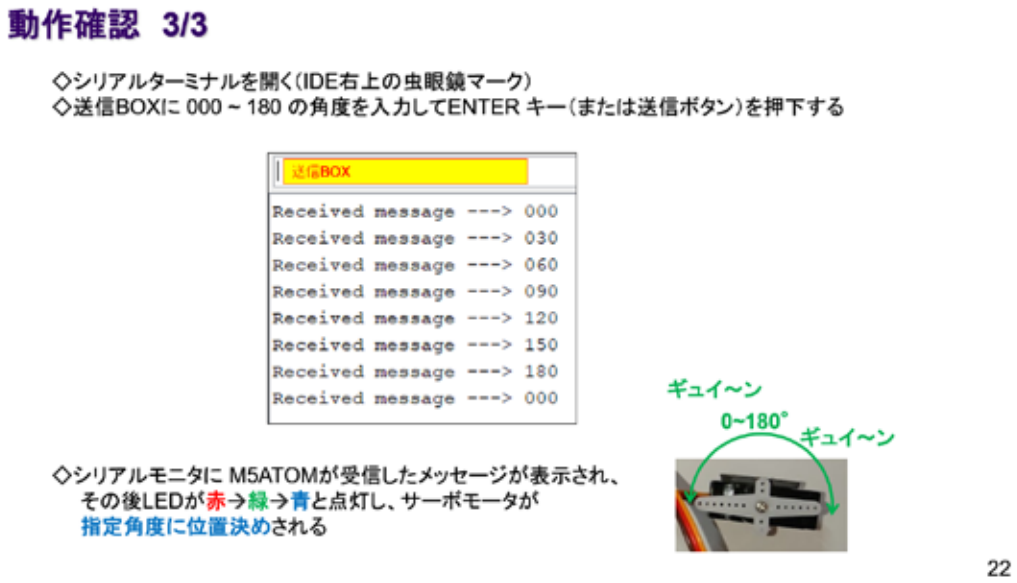


図 3-168

ここまで、M5Atom を用いた IoT システムに必要な開発を行ってきた。この小さなマイコンがインターネットにもつながり、近距離無線通信 Bluetooth にも対応した制御システムを構築できることが実証された。

3．8　M5Stick-C　－　LED

ここからしばらくの間、センシング用マイコンとして使用する M5Stick-C を用いた実験を行う。

図 3-169

図 3-170

いろいろなパッケージが続々登場しているが、このテキストでは、図の左に示すものを利用している。どのパッケージも本体は同じである。

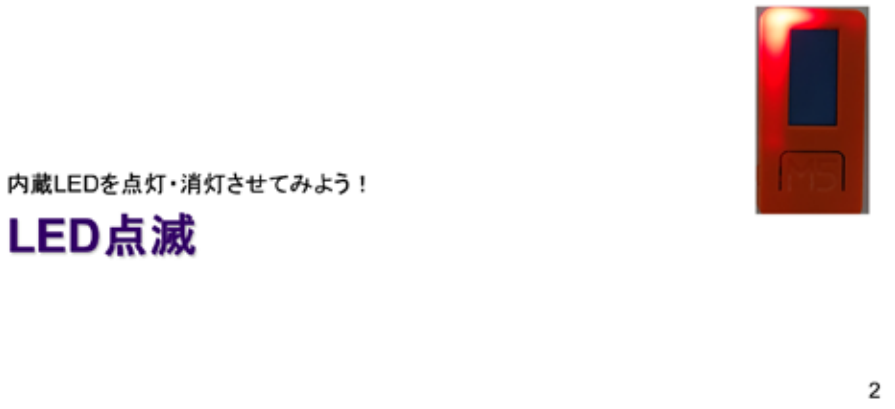


図 3-171

M5Stick-C の基本機能確認を行う。

図 3-172

LED を内蔵していることは、外側から判断できない。

システム構想

◇内蔵LEDを単純に点滅させる → これはどのマイコンでも、初めにやってみること
◇M5Stck-Cのマイコンボードライブラリは、マイコン開発環境の準備でIDE内にインストールされている
→ ソースコードの冒頭で以下のようにライブラリを取り込む
#include <M5StickC.h>
◇マイコンボードを初期化するために、M5.begin() 関数を setup() 関数内で呼び出す

◇LEDを制御するピンは、ライブラリ中で以下のように定義されている
M5_LED または GPIO_NUM_10
※前の図で示したM5Stick-C裏面の LED が G10 の表記は GPIO_NUM_10 を意味する
なお、GPIOとは、汎用の入出力（General Purpose Input Output）を意味している

※GPIOは入出力どちらにでも使える信号
→ 入力か出力かは、プログラムで指定する → 通常これは setup() 関数の中で行う
※通常の処理中にI/Oを切り替えることもある

◇内蔵LEDは、制御信号が負論理なので、LEDの制御では注意する
→ 正論理 ： High→点灯、Low→消灯
→ 負論理 ： High→消灯、Low→点灯

4

図 3-173

ソースコード 1/1 （M5C_LED_1）

◇初期化部分までのソースコード（C++）

```
#include <M5StickC.h>                 // 対象マイコンのライブラリ

void setup(){                         // 初期化部
  M5.begin();

  // LED → GPIO_NUM_10 または M5_LEDでアクセスできる
  pinMode(GPIO_NUM_10, OUTPUT);

}                                     ※両方を使ってみただけ

void loop() {                         // 通常処理部
  digitalWrite(GPIO_NUM_10, LOW); // GPIO_NUM_10で点灯（負論理）
  delay(2000);                        // しばし待つ ( 2000ms = 2sec )
  digitalWrite(M5_LED, HIGH);         // M5_LEDで消灯（負論理）
  delay(2000);  // しばし待つ
}
```

TABキーまたはスペースで行頭をそろえる

5

図 3-174

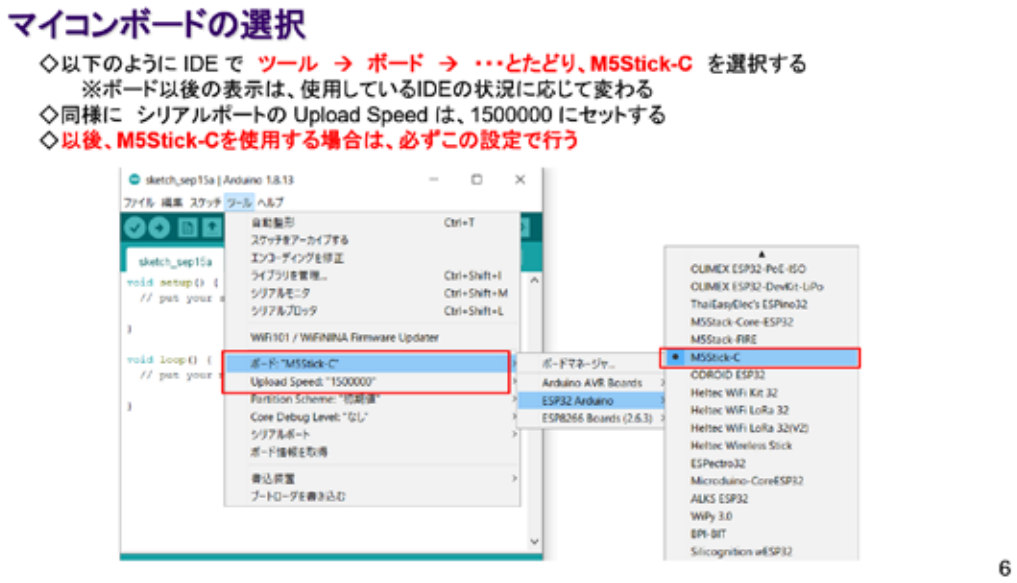


図 3-175

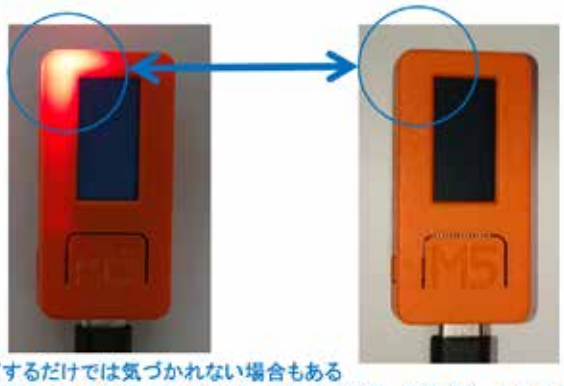
この IDE は、非常に多くのマイコンボードに対応しているので、開発対象のマイコンボードを選択する必要がある。一度選択すれば、マイコンを変更しない限り有効である。

図 3-176

ソースコードが完成したら、コンパイルからマイコン内部への書き込みまでを一気に行うので、コンパイル前にマイコンと PC を USB ケーブル接続しておく。マイコンが PC に接続されると、シリアルポート（COM ポート）が認識されるので、デバイスマネージャでその番号を確認して IDE のツールで選択する。同じマイコンと PC であれば、COM ポート番号は変わらず、ほとんどの場合最初に確認した COM ポート番号を今後も使い続けることになる。

動作確認

◇新しいプログラムは書き込みが終了すると、自動的にスタートする
◇LEDはプログラムで指定した delay() の間隔で点滅を繰り返す



◇LEDは、点灯するだけでは気づかれない場合もある
→ 短い間隔で点滅を繰り返したのち、点灯するなど、点灯パターンを変えることで
十分な表示機能が得られる ※点灯パターンにより内部エラーを知らせる等

8

図 3-177

３．９　M5Stick-C　ー　Button（押しボタンスイッチ）

図 3-178

M5Stick-C は複数の SW を持っている。

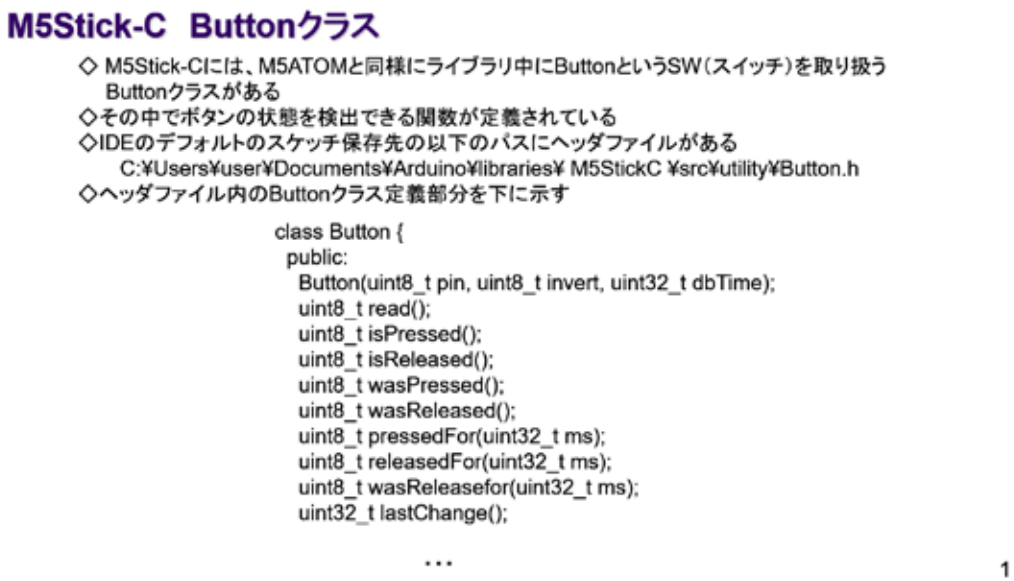


図 3-179

M5Stick-Cの2つのボタン

◇ M5Stick-Cには、プログラムで使用できる【Aボタン】と【Bボタン】がある
→ プログラムからはそれぞれ M5.BtnA と M5.BtnB でアクセスできる
◇この BtnA と BtnB は、IDEのデフォルトのスケッチ保存先の以下のパスにあるヘッダファイル で定義されている
C:¥Users¥user¥Documents¥Arduino¥libraries¥ M5StickC ¥src¥M5StickC.h
→ 右に該当部分を示す


```
class M5StickC {

 public:
    M5StickC();
        void begin(bool LCDEnable=true, bool PowerEnable=tru
    void update();

    //!LCD
    M5Display Lcd = M5Display();

    //!Power
    AXP192 Axp = AXP192();

    #define DEBOUNCE_MS 10
    Button BtnA = Button(BUTTON_A_PIN, true, DEBOUNCE_MS);
    Button BtnB = Button(BUTTON_B_PIN, true, DEBOUNCE_MS);
    //!RTC
    RTC  Rtc;
```

2

図 3-180

本体左側面の SW は電源用で、長押しして電源を OFF する。

Buttonクラスの機能

◇Buttonの関数は、以下の機能がある
◇ここでは、M5.BtnA.wasPressed() のようにしてボタンの押下状態を調べる
◇ボタンの状態は M5.update() を用いて更新されるので、通常処理の中でこの関数をCallする

関数	機能
M5.update()	ボタンの状態を更新する関数 ※loop()関数内で必ず実行する
isPressed()	ボタンを押しているかどうかを返す ※ボタンを押している間は常にTRUEが戻る
isReleased()	ボタンを離しているかどうかを返す ※ボタンを押していない間は常にTRUEが戻る
wasPressed()	ボタンを押してから最初に呼び出した時だけ、TRUEを返す
wasReleased()	ボタンを押して、離してから最初に呼び出した時だけTRUEを返す
pressedFor(ms)	ボタンを指定時間以上押している場合にTRUEが返される
releasedFor(ms)	ボタンを離してから指定時間以上経過している場合にTRUEが返れる
wasReleasefor(ms)	指定時間以上ボタンを押し、離してから最初に呼び出した時だけTRUEを返す
lastChange()	最後にボタンの状態が変更された時の millis() の値が返却される 現在のmillis()からの差分が経過時間になりる

3

図 3-181

M5 シリーズのマイコンで共通の Button クラスが利用できる。

図 3-182

単純な、SW による LED の点灯・消灯制御である。以下にソースコードを示す。

図 3-183

ソースコード 2/2 （M5C_Button_1）

◇通常処理部分

※ボタンの押下状態を調べるために M5.BtnA.wasPressed() M5.BtnB.wasPressed() をCallする
また、ボタン状態の更新のために M5.update() をCallする

```
void loop() {
  M5.update();                          // M5Stick-C 状態を更新

  if(M5.BtnA.wasPressed()== true) {
    digitalWrite(GPIO_NUM_10, LOW); // GPIO_NUM_10で点灯
  }
                                        // どちらでもよい
  if(M5.BtnB.wasPressed()== true) {
    digitalWrite(M5_LED, HIGH);       // M5_LEDで消灯
  }
}
```

6

図 3-184

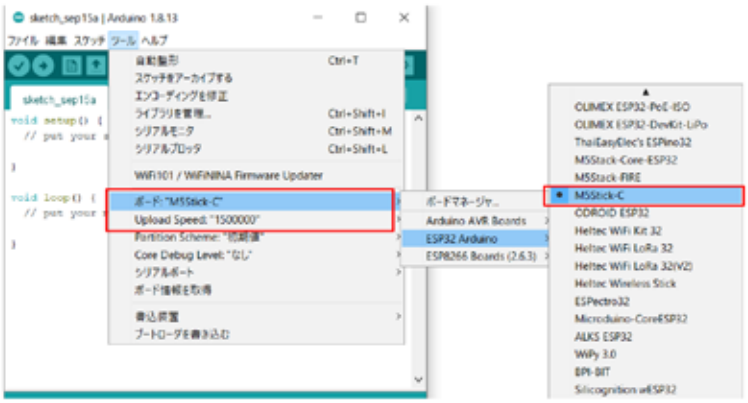
マイコンボードの選択

◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する

※ボード以後の表示は、使用しているIDEの状況に応じて変わる

◇同様に シリアルポートの Upload Speed は、1500000 にセットする

◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



7

図 3-185


マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
PCのUSBポートに接続
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
8

図 3-186


動作確認
◇新しいプログラムは書き込みが終了すると、自動的にスタートしている
◇Aを押下するとLEDが点灯し、Bボタンを押下するとLEDが消灯することを確認しよう！
カチッ！
Aボタン
Bボタン
カチッ！
◇これで2つのボタンが使えるようになった
※左側面のボタンもプログラムで使用することができるが、電源制御が絡む
利用する際は、慎重に調査してからシステムを開発すること
9

図 3-187

３．１０　M5Stick-C　ー　シリアル通信

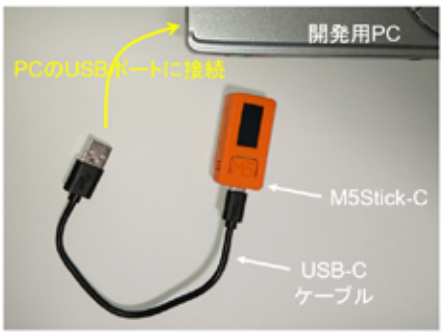
M5Stick-C

シリアル通信

図 3-188

M5Stick-CもPCとUSB接続される

◇M5Stick-Cは、プログラム書き込みの際にはPCとUSB接続します
◇マイコンはUSB経由のシリアル通信によって受け取ったプログラムをフラッシュメモリに書き込みます
◇今回は、このシリアル通信を利用して、PCとの間で通信を行います



1

図 3-189

固定のメッセージをPCに向けて送信する

送信

2

図 3-190

システム構想

◇単純なメッセージ送信を行う
◇メッセージは固定
◇既にLEDとボタンが使えるようになっている
→ ボタン押下に同期して【LED点灯制御＋メッセージ送信】を行う
M5Stick-CにはA、B 2つのボタンがある

◇以下のライブラリ関数が準備されている

①. ボタン押下 → M5.update() と M5.BtnA.wasPressed()
M5.BtnB.wasPressed()
②. LED点灯 → digitalWrite(M5_LED, LOW) ※負論理なのでLOWで点灯
③. LED消灯 → digitalWrite(M5_LED, HIGH)

③. メッセージ送信 → Serial.print() または Serial.println()
※押されたボタンをメッセージで伝える

◇過去の資産（ボタンのプログラム）の流用を考えて進める

3

図 3-191

シリアル通信のデフォルト速度

◇M5Stick-Cは、シリアルポートの初期化処理も、デフォルトでライブラリが処理をする
◇C:¥Users¥user¥Documents¥Arduino¥libraries¥M5Atom¥src¥M5StickC.cpp の該当部分を示す

```
10 void M5StickC::begin(bool LCDEnable, bool PowerEnable, bool SerialEnable){
11
12         //! Correct init once
13         if (isInited) return;
14         else isInited = true;
15
16         //! UART
17         if (SerialEnable) {
18                 Serial.begin(115200);
19                 Serial.flush();
20                 delay(50);
21                 Serial.print("M5StickC Initializing...");
22         }
```

※この部分でシリアルポートの初期化が行われている Serial.begin()のパラメータが通信速度
初期メッセージは改行コードを出力していないので Serial.println() に変更しても良い

4

図 3-192

ソースコード 1/2 （M5C_Serial_1）

◇初期化処理まで

```
#include <M5StickC.h>                  // マイコンボードライブラリ

void setup(){                          // 初期化部
  M5.begin();

  // LED ON(GPIO_NUM_10 or M5_LED)
  pinMode(M5_LED, OUTPUT);             // LEDピンを出力に設定する
  digitalWrite(M5_LED, HIGH);          // LEDをあらかじめ消灯しておく
}
```

5

図 3-193

ソースコード 2/2 （M5C_Serial_1）

◇通常処理では 押されたボタンを通知するメッセージを送信している

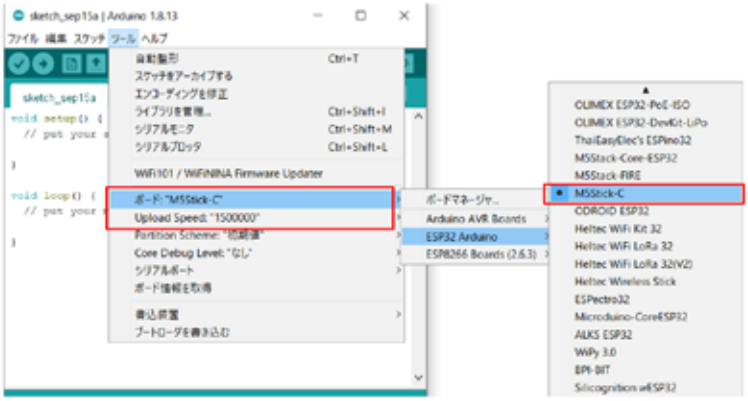
```
void loop() {                                  // 通常処理部
  M5.update();                                 // M5Stick-C 状態を更新

  if(M5.BtnA.wasPressed()== true) {            // Aボタンが押下されたか？
    digitalWrite(M5_LED, LOW);                 // LED点灯（負論理）
    Serial.println("Button A was Pressed!!");  // Aボタンが押された
  }

  if(M5.BtnB.wasPressed()== true) {            // Bボタンが押下されたか？
    digitalWrite(M5_LED, HIGH);                // LED消灯（負論理）
    Serial.println("Button B was Pressed!!");  // Bボタンが押された
  }
}
```

6

図 3-194

マイコンボードの選択

◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



7

図 3-195


マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
8

図 3-196


動作確認 1/3
◇新しいプログラムは書き込みが終了すると、自動的にスタートしている
◇Aを押下するとLEDが点灯し、Bボタンを押下するとLEDが消灯することを確認しよう！
カチッ！
Aボタン
Bボタン
カチッ！
◇これで2つのボタンが使えるようになった
※左側面のボタンもプログラムで使用することができるが、電源制御が絡む
利用する際は、慎重に調査してからシステムを開発すること
9

図 3-197

動作確認　2/3　メッセージ送信の確認準備

◇IDEの右上にある　虫眼鏡マーク　のボタンをクリックするとシリアルターミナルが起動する①

◇シリアルターミナル右下のプルダウンで、
デフォルトの通信速度 11520 を選択する②

10

図 3-198

動作確認　3/3　メッセージ送信の確認

◇Aボタンを押すとLEDが点灯し、メッセージ(Button A was pressed!!)が表示される
◇Bボタンを押すとLEDが消灯し、メッセージ(Button B was pressed!!)が表示される

11

図 3-199

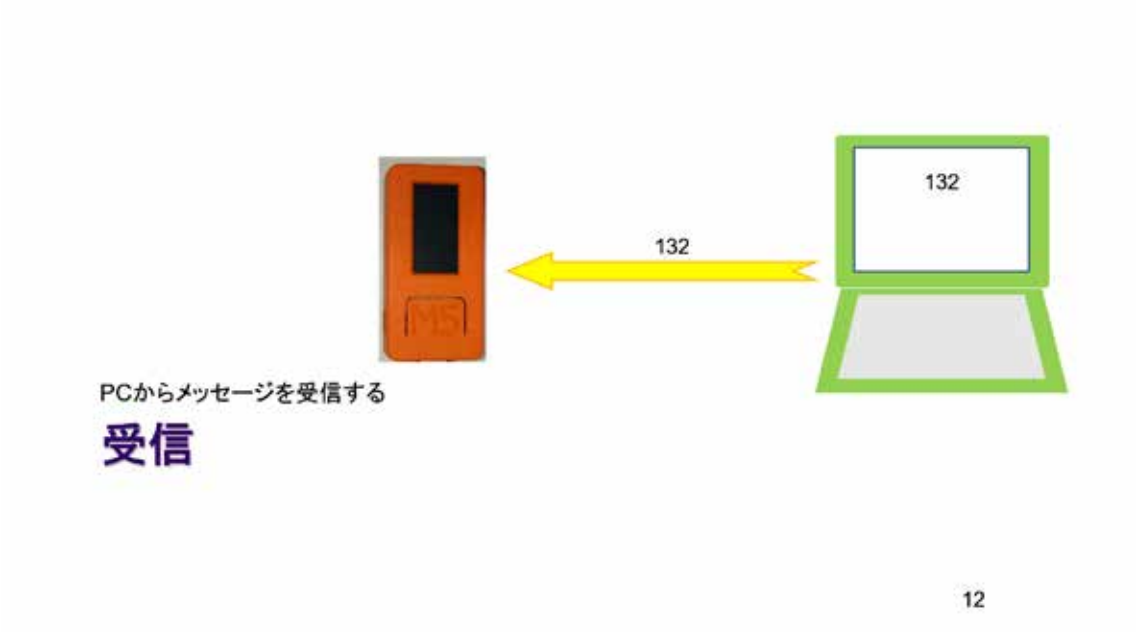


図 3-200

システム構想

◇次は受信を行う
◇受信メッセージでLEDを制御しよう
◇メッセージ設計と処理設計
　①. メッセージ長 ： 1バイト＋改行コード
　②. 改行コードまで受信して、メッセージの解析と対応する処理を行う
　③. 受信バッファは、文字型配列で10バイト程度確保する（誤った長いメッセージに対応）
　④. メッセージの内容
　　0文字目：LED ON / OFF 指示 0：消灯 1：点灯 それ以外：点滅

◇以下のライブラリ関数を使用する

　①. Serial.available() → 受信バッファにデータがあるか調べる
　②. Serial.read() → 1文字受信する
　③. digitalWrite(M5_LED, LOW) → LED点灯
　④. digitalWrite(M5_LED, HIGH) → LED消灯

13

図 3-201

ソースコード 1/3 （M5C_Serial_2）

◇初期化処理まで

```
#include <M5StickC.h>              // マイコンボードライブラリ

int i;                             // 1文字受信時のバッファインデックス
char buf[10];                      // メッセージバッファ

void setup(){                      // 初期化部
  M5.begin();

pinMode(M5_LED, OUTPUT);           // LEDピンを出力に設定する
  digitalWrite(M5_LED, HIGH);      // LEDをあらかじめ消灯しておく
  i=0;                             // バッファインデックス初期化
}
```

14

図 3-202

ソースコード 2/3 （M5C_Serial_2）

◇通常処理 最も外側部分

```
void loop() {                      // 通常処理部
  char c;                          // 受信データ 1文字分

  if(Serial.available()){          // 受信データあり？
    c = Serial.read();             // 1文字 Read
    buf[i++] = c;                  // 受信した文字を格納
    if(c == '¥n'){                 // 改行ならば電文の終端
      i=0;                         // バッファインデックス初期化

      // LED点灯制御部
    }
  }
}
```

15

図 3-203

ソースコード 3/3 （M5C_Serial_2）

◇LED点灯制御部

```
switch (buf[0]){
    case '0':      // 0→LED消灯
      digitalWrite(M5_LED, HIGH);          // LED消灯（負論理）
      Serial.println("LED OFF!!");         // LED消灯メッセージ
      break;
    case '1':      // 1→LED点灯
      digitalWrite(M5_LED, LOW);           // LED点灯（負論理）
      Serial.println("LED ON!!");          // LED点灯メッセージ
      break;
    default:       // 不明なメッセージ
      digitalWrite(M5_LED, LOW);           // LED点灯（負論理）
      delay(100); // しばし待つ
      digitalWrite(M5_LED, HIGH);          // LED消灯（負論理）
      delay(100); // しばし待つ
            ・・・ ・・・
      Serial.println("Unknown command!!");  // 不明なコマンドメッセージ
      break;
  }
```

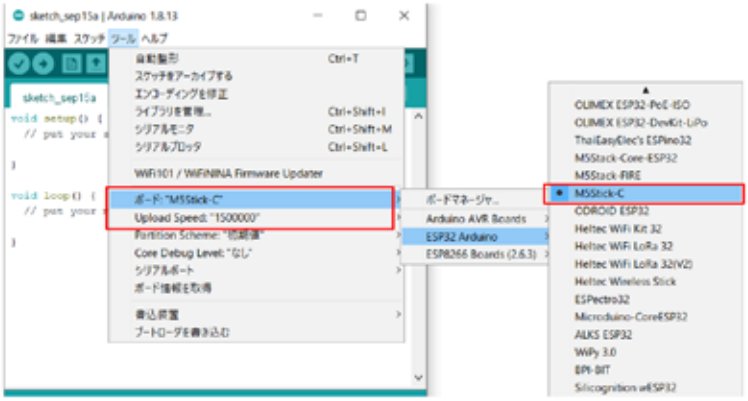
LEDピカピカ

16

図 3-204

マイコンボードの選択

◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



17

図 3-205


マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
18

図 3-206

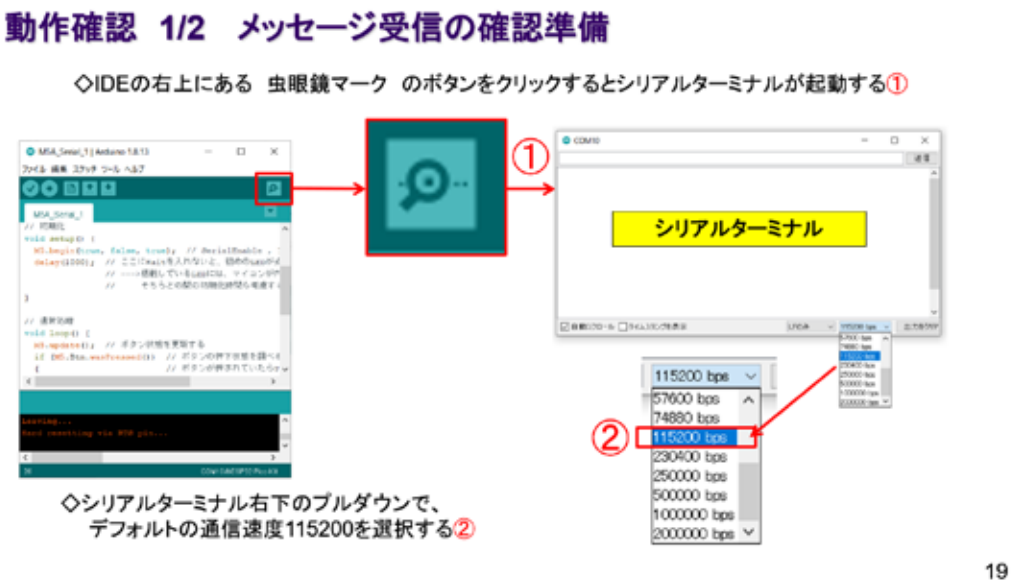
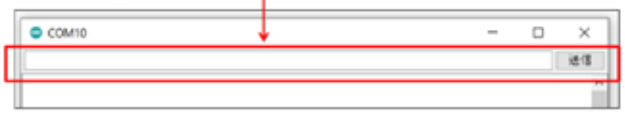

動作確認 1/2 メッセージ受信の確認準備
◇IDEの右上にある 虫眼鏡マーク のボタンをクリックするとシリアルターミナルが起動する①
①
シリアルターミナル
115200 bps
57600 bps
74880 bps
115200 bps
230400 bps
250000 bps
500000 bps
1000000 bps
2000000 bps
②
◇シリアルターミナル右下のプルダウンで、
デフォルトの通信速度115200を選択する②
19

図 3-207

動作確認 2/2

◇シリアルターミナルの送信BOXに以下の電文を記述して ENTER Key を押下する
　または 送信ボタンをクリックする



◇電文を 1桁の半角数字で入力する

①. 1 ENTER → LEDが点灯しLED ONメッセージ
②. 0 ENTER → LEDが消灯しLED OFFメッセージ
③. 0, 1以外 ENTER → LEDが点滅しUnknown command!!メッセージ

20

図 3-208

３．１１　M5Stick-C　ー　MQTT（WEB 経由通信）

図 3-209

M5Stick-C は WiFi に接続でき、MQTT というインターネットを利用した通信が行える。

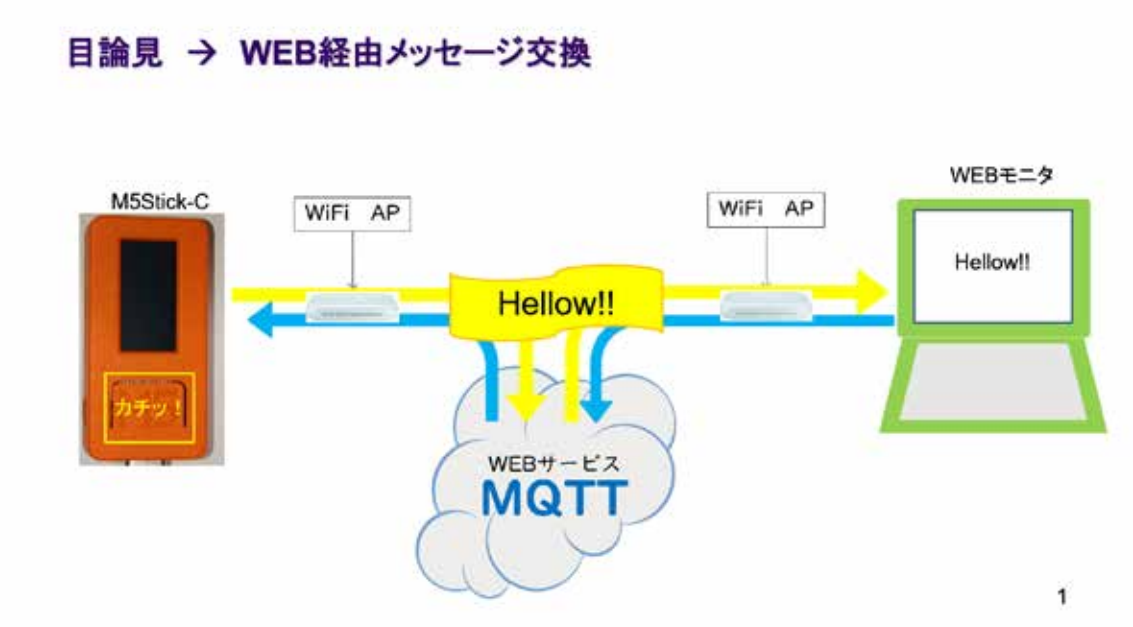


図 3-210

MQTTサービス

◇MQTTは、Message Queue Telemetry Transport　の頭文字
◇サービスプロバイダ（MQTT Broker）が世界中にある
◇短いメッセージ交換に特化したWEBサービス
◇登録不要で即利用できる Broker が多い

◇WEBを経由するメッセージ交換　→　インターネットに接続できれば場所を選ばない
◇マイコン⇔マイコン間、PC⇔PC間、マイコン⇔PC間でのメッセージ交換が可能
◇言語依存しない（マイコン向けC＋＋・MicroPython・PC用Python・同C++・・・etc）
　　ライブラリが必要　→ 本講座では PubSubClient を利用するのでIDEにインストールする

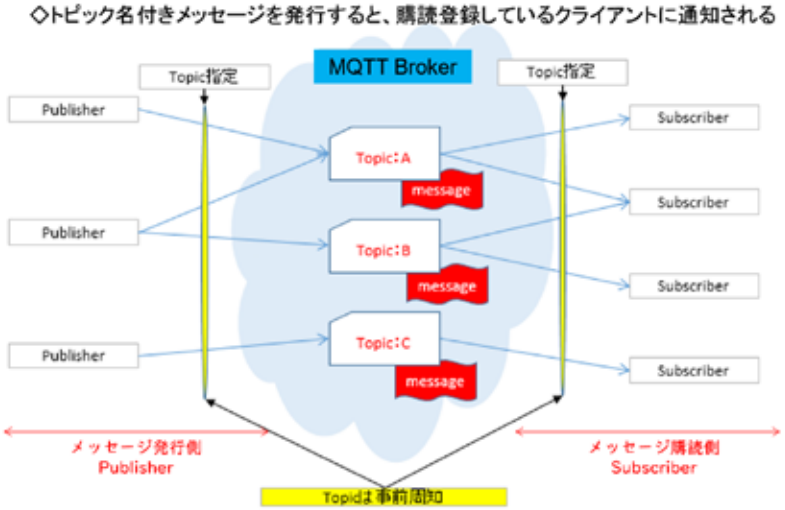
◇ROS（Robot OS）でnode間通信にも採用されている仕組み
◇ルールが簡単

2

図 3-211

MQTTの仕組み

◇トピック名付きメッセージを発行すると、購読登録しているクライアントに通知される



3

図 3-212

MQTT Broker

◇サービスを提供している MQTT Broker は世界中に沢山ある

- test.mosquito.org
- broker.hivemq.com
- broker.shiftr.io ← 今回利用する
- broker.mqttdashboard.com
- iot.eclipse.org

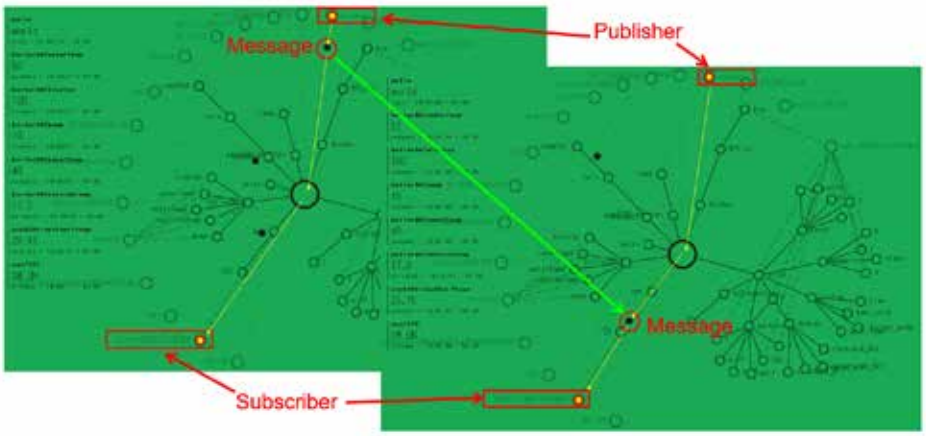
・・・ etc

4

図 3-213

broker.shiftr.io

◇shiftr.io は、オープンMQTTの利用が認められており、図のようにPC上でメッセージの動きが見える
◇赤丸○で示しているのが発行したメッセージの流れである 中央の大きな丸○はBrokerを意味する



5

図 3-214

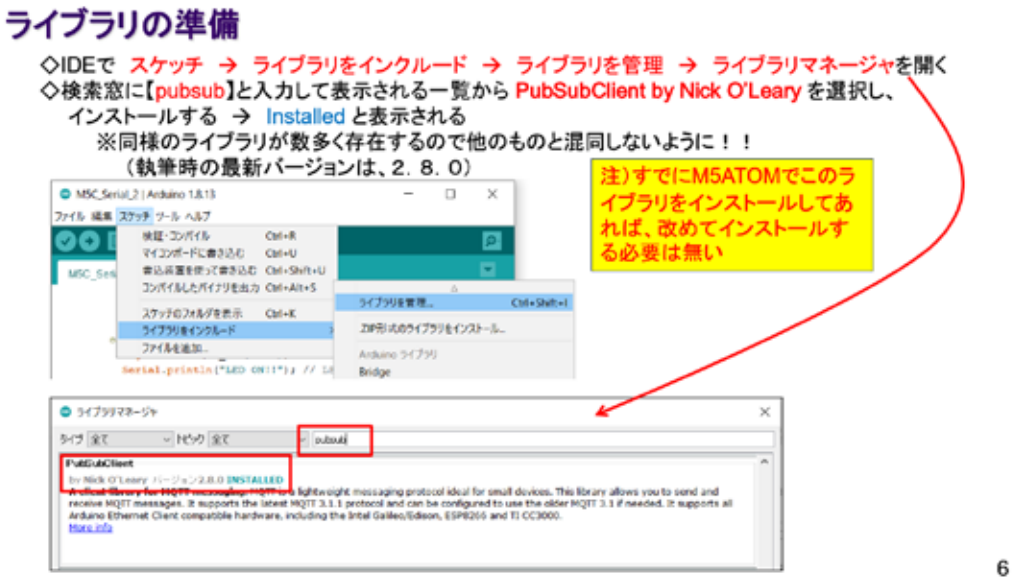


図 3-215

このライブラリは、M5 シリーズで共通に利用できる。既にインストール済みであれば、そのまま使用する。

図 3-216

2 つのボタン押下に応じて、別々のメッセージを発信する。

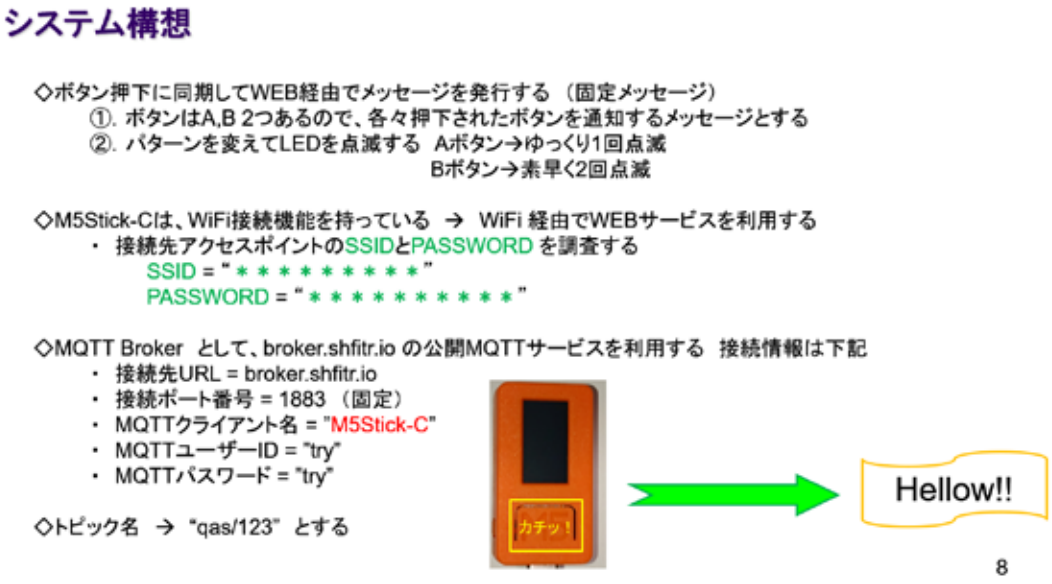


図 3-217

ソースコード 1/4 （M5C_MQTT_1）

◇#include #define 部分

```
#include <M5StickC.h>
#include <WiFi.h>
#include <PubSubClient.h>

#define WiFi_SSID "＊＊＊＊＊＊＊＊"        // 使用するWiFiのSSID
#define WiFi_PASS  "＊＊＊＊＊＊"           // 使用するWiFiのPassword
#define MQTT_BROKER "broker.shiftr.io"     // MQTT Broker  利用するブローカーのURL
#define MQTT_PORT 1883                     // MQTT BROKER PORT  固定
#define MQTT_CLIENT_NAME  "M5Stick-C"   // MQTTブローカ接続時のクライアント名
#define MQTT_USER "try"                    // 公開クライアントユーザー名  決まっている
#define MQTT_PASS "try"                    // 同、パスワード    決まっている
#define MQTT_TOPIC "qas/123"               // 事前に取り決めた TOPIC名

WiFiClient espClient;                      // WiFi接続クライアントオブジェクト
PubSubClient client(espClient);            // MQTTクライアント接続オブジェクト
```

9

図 3-218

ソースコード 2/4 （M5C_MQTT_1）

◇WiFiアクセスポイント接続関数 と 初期化処理部分

```
void wifi_connect(void){                          // WiFiアクセスポイントへの接続
  Serial.print("WiFi Connenting");                // 接続を試みているメッセージ
  WiFi.begin(WiFi_SSID, WiFi_PASS);               // 接続開始
  while (WiFi.status() != WL_CONNECTED) {         // アクセスポイント接続待ち
    Serial.print(".");                            // Wait時・・・表示
    delay(1000);                                  // しばし待つ
  }
  Serial.print("¥n---> Connected : ");            // 接続成功メッセージ
  Serial.println(WiFi.localIP());                 // 自機のIPアドレス表示
  client.setServer(MQTT_BROKER, MQTT_PORT);       // MQTTブローカへの接続設定
}

void setup() {                                    // 初期化処理
  M5.begin(true, false, true);                    // SerialEnable , I2CEnable , DisplayEnable
  wifi_connect();                                 // WiFiアクセスポイント接続
  pinMode(M5_LED, OUTPUT);                        // LED ピン出力に設定
  digitalWrite(M5_LED, HIGH);                     // LED消灯
}
```

10

図 3-219

ソースコード 3/4 （M5C_MQTT_1）

◇通常処理部

```
void loop() {                                   // 通常処理
  client.loop();                                // MQTT接続状況更新
  while(!client.connected()){                   // MQTT接続
    Serial.println("Mqtt Reconnecting");        // MQTT接続  試みているメッセージ
    if( client.connect(MQTT_CLIENT_NAME, MQTT_USER, MQTT_PASS) ){
      Serial.println("Mqtt Connected");         // MWTT接続成功メッセージ
      break;                                    // 接続が成功したので、while()ループから抜ける
    }

  // ボタン処理部  ←  M5ATOMと異なるボタン処理
}
```

11

図 3-220

ソースコード 4/4 （M5C_MQTT_1）

◇ボタン処理部

```
M5.update();                                                // M5Stick-Cボタン状況更新
 if (M5.BtnA.wasPressed()){                                 // Aボタンが押されていたか？
   client.publish(MQTT_TOPIC, "Button A was pressed !!");   // ここでAボタン押下メッセージを送信
   Serial.println("Send message (Button A was pressed!!)"); // シリアルターミナルにも出力
   digitalWrite(M5_LED, LOW);       // LED点灯
   delay(1000);                     // しばし待つ
   digitalWrite(M5_LED, HIGH);      // LED消灯          // 点滅は1回だけ
 }
 if (M5.BtnB.wasPressed()){                                 // Bボタンが押されていたか？
   client.publish(MQTT_TOPIC, "Button B was pressed !!");   // ここでBボタン押下メッセージを送信
   Serial.println("Send message (Button B was pressed!!)"); // シリアルターミナルにも出力
   digitalWrite(M5_LED, LOW);       // LED点灯
   delay(200);                      // しばし待つ
   digitalWrite(M5_LED, HIGH);      // LED消灯
   delay(200);                      // しばし待つ
   digitalWrite(M5_LED, LOW);       // LED点灯          // 点滅を2回繰り返す
   delay(200);                      // しばし待つ
   digitalWrite(M5_LED, HIGH);      // LED消灯
 }
```

12

図 3-221

シリアルターミナルの起動　（マイコンリセット時メッセージを見るためにシリアルモニタを起動する）

◇Windowsのデバイスマネージャで COMポート番号を確認し、
　ツール→シリアルポート　で、COMポートを選択する
◇IDEの右上にある　虫眼鏡マーク　のボタンをクリックするとシリアルターミナルが起動する①

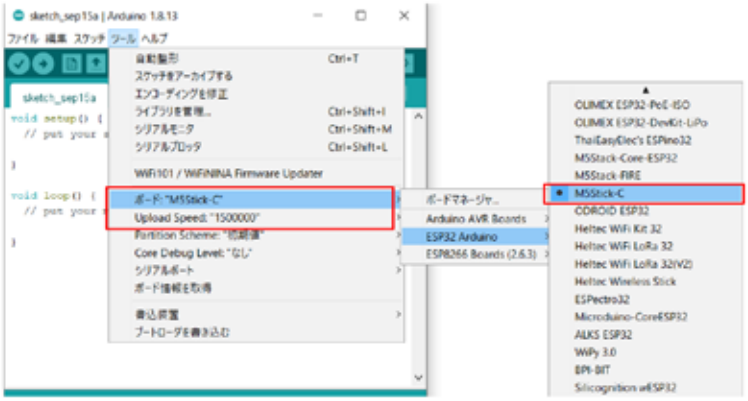
◇シリアルターミナル右下のプルダウンで、
　デフォルトの通信速度115200を選択する②

13

図 3-222

マイコンボードの選択

◇以下のように IDE で　ツール　→　ボード　→　・・・とたどり、M5Stick-C　を選択する
　※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に　シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



14

図 3-223

図 3-224

動作確認 1/4 メッセージ送信の確認

◇書込みが完了すると M5Stick-CはResetされ、その際のメッセージがシリアルモニタに表示される

◇ボタンを操作する
①. Aボタンを押下する → LEDがゆっくり1度点滅する(右図)
②. 同時にシリアルモニタにメッセージが表示される(下図)
③. Bボタンを押下する → LEDが素早く2回点滅する(右図)
④. 同時にシリアルモニタにメッセージが表示される(下図)

```
M5StickC initializing...OK
WiFi Connenting..
---> Connected : 192.168.0.72
Mqtt Reconnecting
Mqtt Connected
Send message (Button A was pressed!!)
Send message (Button B was pressed!!)
```

16

図 3-225

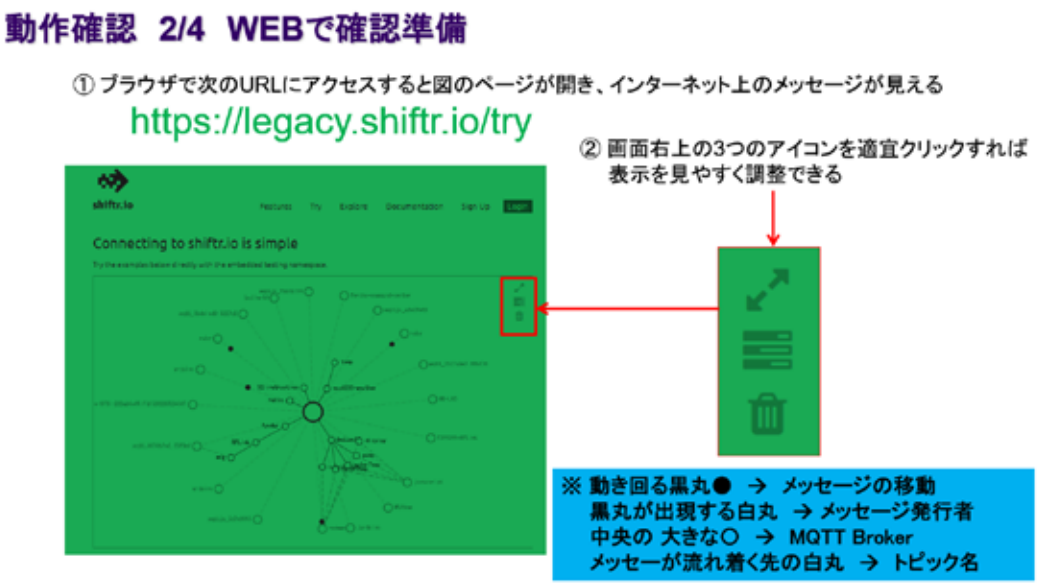

動作確認 2/4 WEBで確認準備
① ブラウザで次のURLにアクセスすると図のページが開き、インターネット上のメッセージが見える
https://legacy.shiftr.io/try
② 画面右上の3つのアイコンを適宜クリックすれば
表示を見やすく調整できる
Connecting to shiftr.io is simple
※ 動き回る黒丸● → メッセージの移動
黒丸が出現する白丸 → メッセージ発行者
中央の 大きな○ → MQTT Broker
メッセーが流れ着く先の白丸 → トピック名

図 3-226

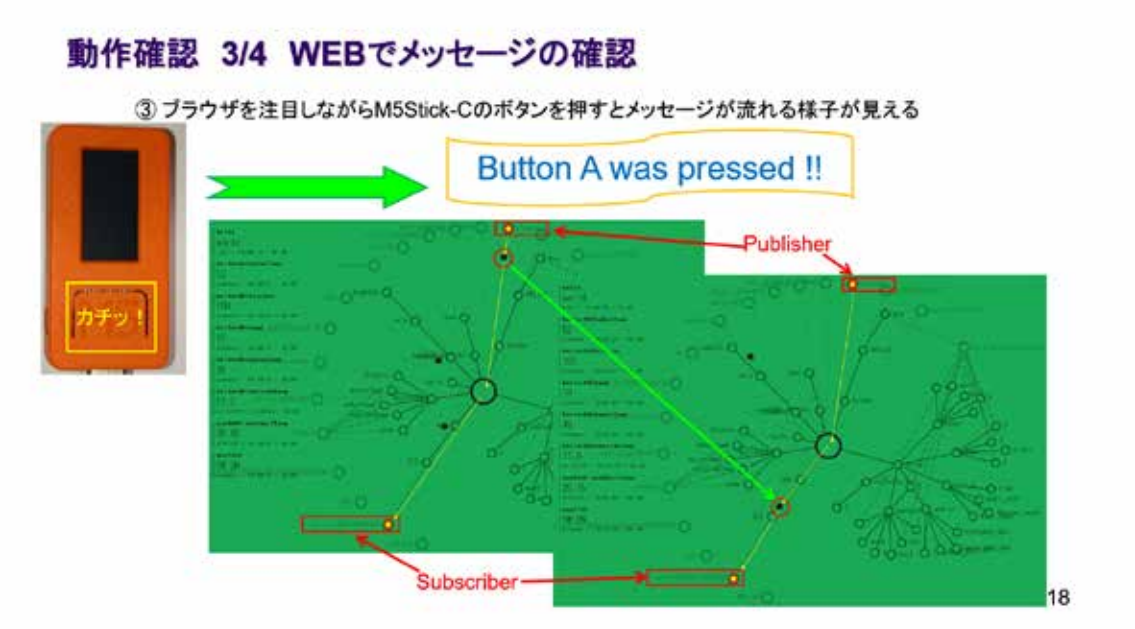

動作確認 3/4 WEBでメッセージの確認
③ ブラウザを注目しながらM5Stick-Cのボタンを押すとメッセージが流れる様子が見える
カチッ！
Button A was pressed !!
Publisher
Subscriber
18

図 3-227

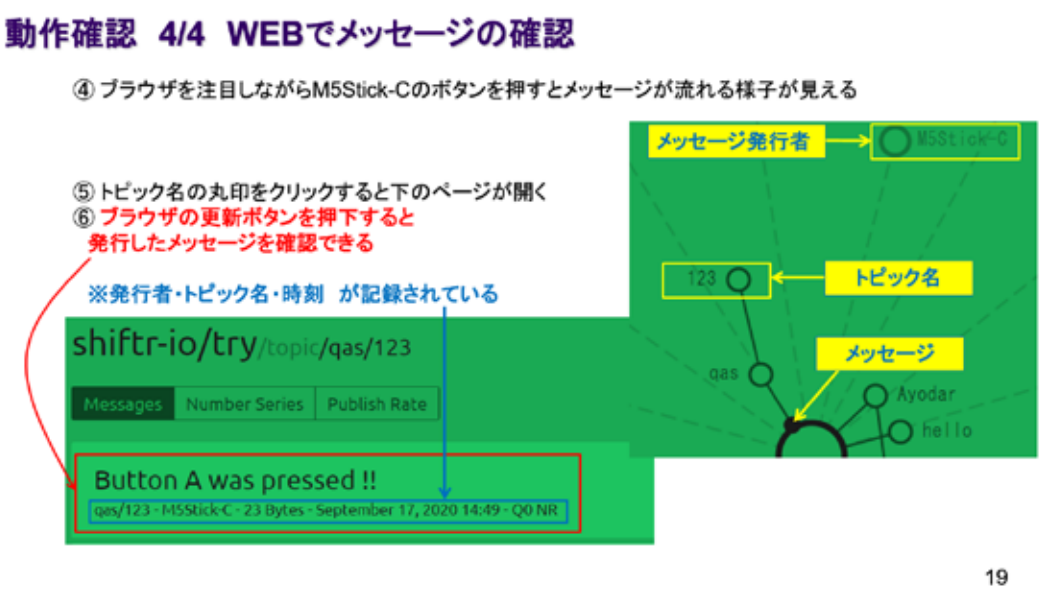

動作確認 4/4 WEBでメッセージの確認
④ ブラウザを注目しながらM5Stick-Cのボタンを押すとメッセージが流れる様子が見える
⑤ トピック名の丸印をクリックすると下のページが開く
⑥ ブラウザの更新ボタンを押下すると
発行したメッセージを確認できる
※発行者・トピック名・時刻 が記録されている
shiftr-io/try/topic/qas/123
Messages
Number Series
Publish Rate
Button A was pressed !!
qas/123 - M5Stick-C - 23 Bytes - September 17, 2020 14:49 - Q0 NR
メッセージ発行者
M5Stick-C
123
トピック名
qas
メッセージ
Ayodar
hello
19

図 3-228

図 3-229

システム構想

◇PCからメッセージを発行することができる
→ コマンドプロンプトから以下のコマンドを発行すれば、"Hello !!" が発行される
curl -X POST "http://try:try@broker.shiftr.io/qas/123" -d "Hello !!"
※qas/123 と Hello !! はそれぞれ、トピック名とメッセージである

◇シリアル通信でLEDを制御したことを思い出して、その際作成したプログラムを利用する
◇メッセージを1文字として、その内容を以下のようにする
0文字目:LED ON / OFF 指示 0：消灯 1：点灯 それ以外:点滅
※これは、シリアル通信の受信で実験したメッセージそのものだ！

◇例えば、**LEDを点灯**したければ以下のコマンドを実行すればよい
curl -X POST "http://try:try@broker.shiftr.io/qas/123" -d "1"

※ここでは curl コマンドの詳細は説明しない
このような便利なコマンドは、自身で調べて記録しておくべきだ（これはLinuxにもある）

21

図 3-230

ソースコード 1/6 （M5C_MQTT_2）

◇#include と #define 、通信関連オブジェクト部分

```
#include "M5StickC.h"                    // マイコンボードライブラリ
#include <WiFi.h>                        // WiFiライブラリ
#include <PubSubClient.h>                // MQTTクライアントライブラリ

#define WiFi_SSID "＊＊＊＊＊＊＊＊"      // WiFiアクセスポイントSSID
#define WiFi_PASS "＊＊＊＊＊＊＊＊"      // 同　パスワード
#define MQTT_BROKER "broker.shiftr.io"   // MQTTブローカーURL
#define MQTT_PORT 1883                   // MQTT BROKER PORT
#define MQTT_CLIENT_NAME  "M5Stick-C"    // MQTTブローカ接続時のクライアント名
#define MQTT_USER "try"                  // 公開ユーザー名（固定）
#define MQTT_PASS "try"                  // 公開パスワード（固定）
#define MQTT_TOPIC "qas/123"             // TOPIC名
#define MQTT_QOS 0                       // Quality of Service（サービスの品質）

WiFiClient espClient;                    // WiFiクライアントオブジェクト
PubSubClient client(espClient);          // MQTTクライアントコントロールオブジェクト
```

22

図 3-231

ソースコード 2/6 （M5C_MQTT_2）

◇グローバル変数 と WiFiアクセスポイント接続関数

```
char flg=0;                              // メッセージ受信フラグ　0:未受信　1:メッセージ到着
char msg[10];                            // 受信メッセージ格納用

void wifi_connect(void){                 // WiFiアクセスポイントへの接続
  Serial.print("WiFi Connenting");       // 接続を試みているメッセージ
  WiFi.begin(WiFi_SSID, WiFi_PASS);      // 接続開始
  while (WiFi.status() != WL_CONNECTED) { // アクセスポイント接続待ち
    Serial.print(".");                   // Wait時・・・表示
    delay(1000);                         // しばし待つ
  }
  Serial.print("¥n---> Connected : ");   // 接続成功メッセージ
  Serial.println(WiFi.localIP());        // 自機のIPアドレス表示
  client.setServer(MQTT_BROKER, MQTT_PORT); // MQTTブローカへの接続設定
}
```

23

図 3-232

ソースコード 3/6 （M5C_MQTT_2）

◇MQTTブローカー接続関数

```
void mqtt_connect() {
  // Loop until we're reconnected                          // 再接続するまでループするぞ！！
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");         // MQTT接続を行っているよ！メッセージ
    if (client.connect(MQTT_CLIENT_NAME, MQTT_USER, MQTT_PASS)) {   // MQTT 接続実行！
      Serial.println("connected");                        // つながった！メッセージ
      client.subscribe(MQTT_TOPIC, MQTT_QOS);             // メッセージ購読登録
      Serial.println("Subscribing!");                     // 購読しているよ！メッセージ
    } else {                                              // うまく行かなかった場合
      Serial.print("failed, rc=");                        // 失敗、その原因コードは！メッセージ
      Serial.print(client.state());                       // クライアントの状態（原因コードが表示される）
      Serial.println(" try again in 5 seconds");          // 5秒待って試行する！メッセージ
      delay(5000);                                        // 5秒待つ
    }
  }
}
```

24

図 3-233

ソースコード 4/6 （M5C_MQTT_2）

◇メッセージが発行された際に呼び出される（コールバック）関数

```
void callback(char* topic, byte* payload, unsigned int length) {
  int i;                                  // メッセージバッファのインデックス

  Serial.print("Message arrived [");      // メッセージが到着したよ！
  Serial.print(topic);                    // トピック名は！
  Serial.print("] ");
  for (i = 0; i < length; i++) {          // メッセージの内容は！
    msg[i] = (char)payload[i];
    Serial.print((char)payload[i]);       // 1文字ずつ表示する
  }
  Serial.println();                       // 改行しておく
  flg = 1;                                // メッセージ到着　loop()内でこのフラグを見て処理する
}
```

25

図 3-234

ソースコード 5/6 （M5C_MQTT_2）

◇LEDを全消灯する際に利用する関数 と 初期化

```
void setup() {                          // 初期化部
  M5.begin(true, false, true);          // M5Stick-C初期化
  wifi_connect();                       // WiFiアクセスポイント接続
  client.setCallback(callback);         // メッセージ購読時処理の登録
  pinMode(M5_LED, OUTPUT);              // LED ピン出力に設定
  digitalWrite(M5_LED, HIGH);           // LED消灯しておく
}
```

26

図 3-235

ソースコード 6/6 （M5C_MQTT_2）

◇通常処理全体

```
void loop() {                           // 通常処理
  int i;

  mqtt_connect();                       // MQTT 接続が切れているといけないので、調べて必要なら再接続
  client.loop();                        // MQTT接続状況更新 ・・・ これがなかなか難しい
  if(flg==1){                           // 受信メッセージあり？
    flg = 0;                            // メッセージ到着フラグクリア
    if(msg[0]=='0'){                    // '0'か？
      digitalWrite(M5_LED, HIGH);       // LED消灯
    }else if(msg[0]=='1'){              // '1'か？
      digitalWrite(M5_LED, LOW);        // LED点灯
    }else{                              // '0'でも'1'でもない！ 不明コマンド
      Serial.println("Invalid Message!!");   // エラーメッセージ
          //          ・・・       ・・・ 素早いLED点滅 3回
      return;                           // OSに戻る
    }
  }
}
```

27

図 3-236

シリアルターミナルの起動　（マイコンリセット時メッセージを見るためにシリアルモニタを起動する）

◇Windowsのデバイスマネージャで COMポート番号を確認し、
　ツール→シリアルポート　で、COMポートを選択する
◇IDEの右上にある　虫眼鏡マーク　のボタンをクリックするとシリアルターミナルが起動する①

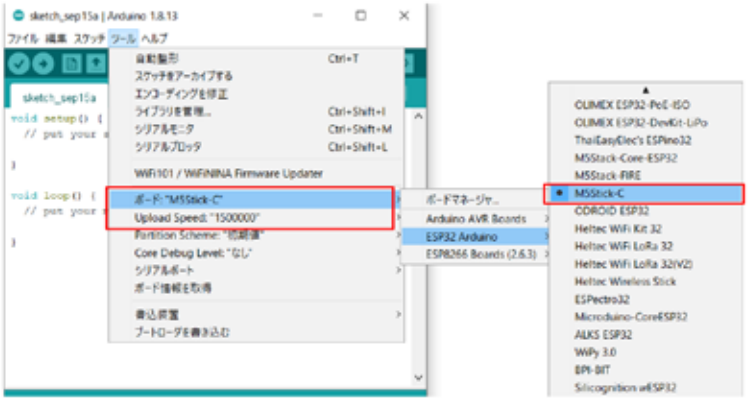
◇シリアルターミナル右下のプルダウンで、
　デフォルトの通信速度115200を選択する②

28

図 3-237

マイコンボードの選択

◇以下のように IDE で　ツール　→　ボード　→　・・・とたどり、M5Stick-C　を選択する
　※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に　シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



29

図 3-238


マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
30

図 3-239

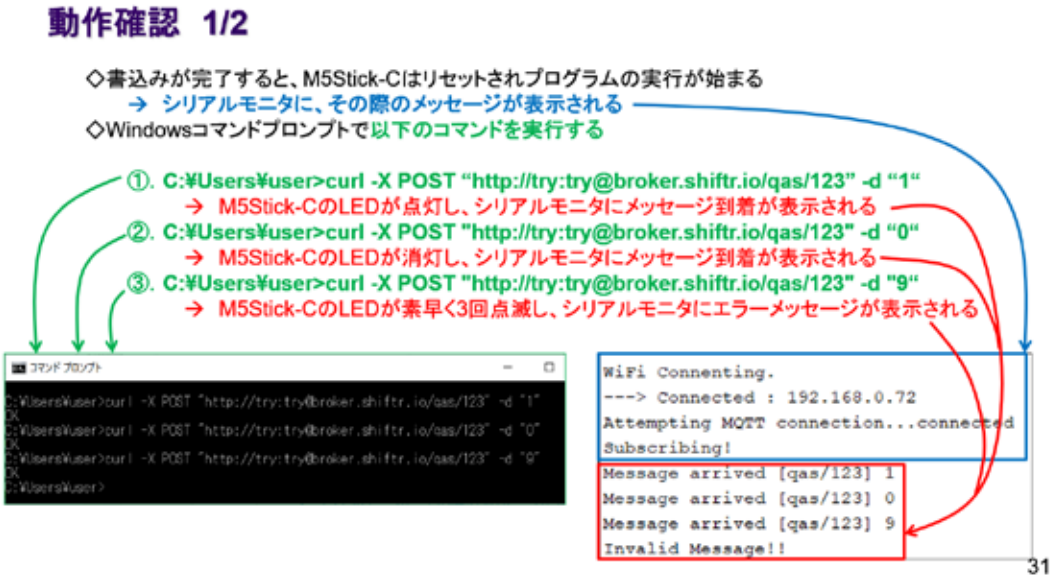

動作確認 1/2
◇書込みが完了すると、M5Stick-Cはリセットされプログラムの実行が始まる
→ シリアルモニタに、その際のメッセージが表示される
◇Windowsコマンドプロンプトで以下のコマンドを実行する
①. C:¥Users¥user>curl -X POST "http://try:try@broker.shiftr.io/qas/123" -d "1"
→ M5Stick-CのLEDが点灯し、シリアルモニタにメッセージ到着が表示される
②. C:¥Users¥user>curl -X POST "http://try:try@broker.shiftr.io/qas/123" -d "0"
→ M5Stick-CのLEDが消灯し、シリアルモニタにメッセージ到着が表示される
③. C:¥Users¥user>curl -X POST "http://try:try@broker.shiftr.io/qas/123" -d "9"
→ M5Stick-CのLEDが素早く3回点滅し、シリアルモニタにエラーメッセージが表示される
WiFi Connenting.
---> Connected : 192.168.0.72
Attempting MQTT connection...connected
Subscribing!
Message arrived [qas/123] 1
Message arrived [qas/123] 0
Message arrived [qas/123] 9
Invalid Message!!
31

図 3-240

動作確認 2/2 WEBで確認

① ブラウザで次のURLにアクセスすると図のページが開き、インターネット上のメッセージが見える

https://legacy.shiftr.io/try

② 画面右上の3つのアイコンを適宜クリックすれば
表示を見やすく調整できる

図 3-241

３．１２　M5Stick-C　ー　LCD（液晶表示器）

図 3-242

M5Stick-C にはフルカラーの液晶表示器が搭載されている。小さくても強力な情報伝達機能を提供している。

図 3-243

ソースコード 1/4 （M5C_Display_Test_1）

◇#include から初期化部

```
#include <M5StickC.h>

void setup() {      // 初期化部
 M5.begin();

 // Lcd display  色
 M5.Lcd.fillScreen(WHITE);          // 白
 delay(5000);
 M5.Lcd.fillScreen(RED);            // 赤
 delay(5000);
 M5.Lcd.fillScreen(GREEN);          // 緑
 delay(5000);
 M5.Lcd.fillScreen(BLUE);           // 青
 delay(5000);
 M5.Lcd.fillScreen(BLACK);          // 黒
 delay(5000);

          // 右に続く・・・
```

```
 // text print  文字列
 M5.Lcd.fillScreen(BLACK);    //背景  黒
 M5.Lcd.setCursor(0, 10);     //カーソル位置
 M5.Lcd.setTextColor(WHITE);  //文字色
 M5.Lcd.setTextSize(1);       //文字サイズ
 M5.Lcd.printf("Display Test!");  //文字列表示

 delay(5000);

 // draw graphic
 M5.Lcd.drawRect(15, 55, 50, 50, BLUE);//青矩形
 delay(5000);
 M5.Lcd.fillRect(15, 55, 50, 50, BLUE);  //塗りつぶし
 delay(5000);
 M5.Lcd.drawCircle(40, 80, 30, RED);  //赤円
 delay(5000);
 M5.Lcd.fillCircle(40, 80, 30, RED);    //塗つぶし
 delay(5000);
}
```

2

図 3-244

ソースコード 2/2 （M5C_Display_Test_1）

◇通常処理部

```
void loop(){    // 通常処理部

 //rand draw
 M5.Lcd.fillTriangle(      random(M5.Lcd.width()-1),      // 三角形の3頂点座標を指定
                           random(M5.Lcd.height()-1),     //
                           random(M5.Lcd.width()-1),      //
                           random(M5.Lcd.height()-1),     //
                           random(M5.Lcd.width()-1),      //
                           random(M5.Lcd.height()-1),     //
                           random(0xffffe)     );         // 塗の色指定
}
```

問題 ： 上記の各頂点のパラメータで -1 しているのは何故か？

3

図 3-245

問題を考えてみよう！

図 3-246

座標系を理解しておかないと、自由な作図ができない。システムに応じて座標系が回転できるのは、たいへん便利な機能である。

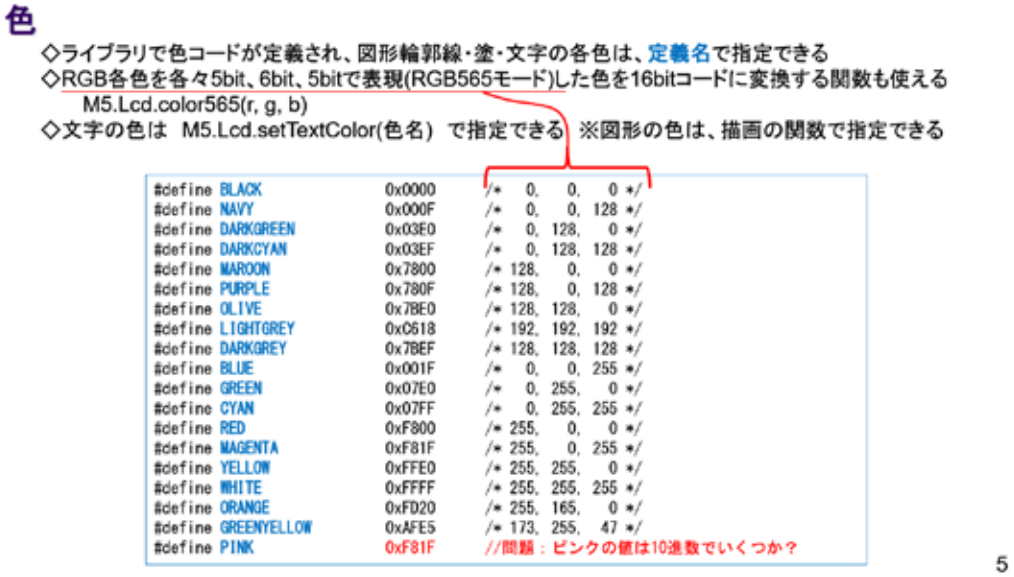


図 3-247

文字描画

◇文字描画は、次の関数が準備されている
・カーソル指定
①. M5.Lcd.print(string) → 文字列描画
②. M5.Lcd.println(string) → 改行付き
③. M5.Lcd.printf("%d", i) → 書式指定文字列描画

・座標指定
④. M5.Lcd.drawString(string, x, y) → 座標指定
⑤. M5.Lcd.drawString(string, x, y, font) → 座標・フォント指定

※文字列・文字描画関数は他にもある

◇文字サイズ、文字色を変えて M5.Lcd.println(string) を用いた描画例（下図）M5C_Display_Test_3 がある
上から順に文字サイズ1,2,3,4 各々色の名称を描画したもの

◇グラフの描画では、ドット単位の位置指定が重要なので、
座標指定文字列描画が威力を発揮する

6

図 3-248

画面塗りつぶし

◇画面全体の塗りつぶしには M5.Lcd.fillScreen(color) 関数が使える

図は左から、WHITE、RED、GREEN、BLUE、BLACK で塗りつぶした様子
※参考ソースコード M5C_Display_Test_1

7

図 3-249

図形描画

◇図形描画関数は以下のものがある
※color は色定義名、または色コード16bit）

①. M5.Lcd.drawPixel(x, y, color) → 点描画 x、yは点の座標
②. M5.Lcd.drawLine(x0, y0, x1, y1, color) → 線描画 x0、y0は開始位置座標、x1、y1は終了位置座標
③. M5.Lcd.drawCircle(x, y, r, color) → 円描画 x、yは円の中心座標、rは半径
④. M5.Lcd.drawRect(x, y, w, h, color) → 矩形描画 x、yは左上の座標、w、hは矩形の幅、高さ
⑤. M5.Lcd.drawTriangle(x0, y0, x1, y1, x2, y2, color) → 三角形描画 xn,ynは3つの頂点の座標
⑥. M5.Lcd.fillCircle(x, y, r, color) → 円塗りつぶし描画 x、yは中心座標、rは半径
⑦. M5.Lcd.fillRect(x, y, w, h, color) → 矩形塗りつぶし x、yは左上の座標、w、hは矩形の幅、高さ
⑧. M5.Lcd.fillTriangle(x0, y0, x1, y1, x2, y2, color) → 三角形描画 xn,ynは3つの頂点の座標

※参考ソースコード M5C_Display_Test_1

8

図 3-250

文字フォント

◇文字フォントは、次の関数で指定できる

①. M5.Lcd.setTextFont(font) → あらかじめフォントを指定する
②. M5.Lcd.drawString(string, x, y, font) → 描画時に指定する

font = 1 → Adafruit 8ピクセルASCIIフォント（defaultらしい）
2 → 16ピクセルASCIIフォント → 1のフォントを2倍の大きさにしたもの
4 → 26ピクセルASCIIフォント
6 → 26ピクセル数字フォント
} サイズが大きすぎて実用的ではない

※参考ソースコード M5C_Display_Test_4

9

図 3-251

実際に文字列を描画してみると、１または２のフォントサイズが適当だと分かる。

図 3-252

この例では、計測値の数値を表示していない。数値表示の場合は、文字サイズをあらかじめ実験で確認しておくとよい。

３．１３　M5Stick-C　ー　加速度センサ（Accelerometer）

図 3-253

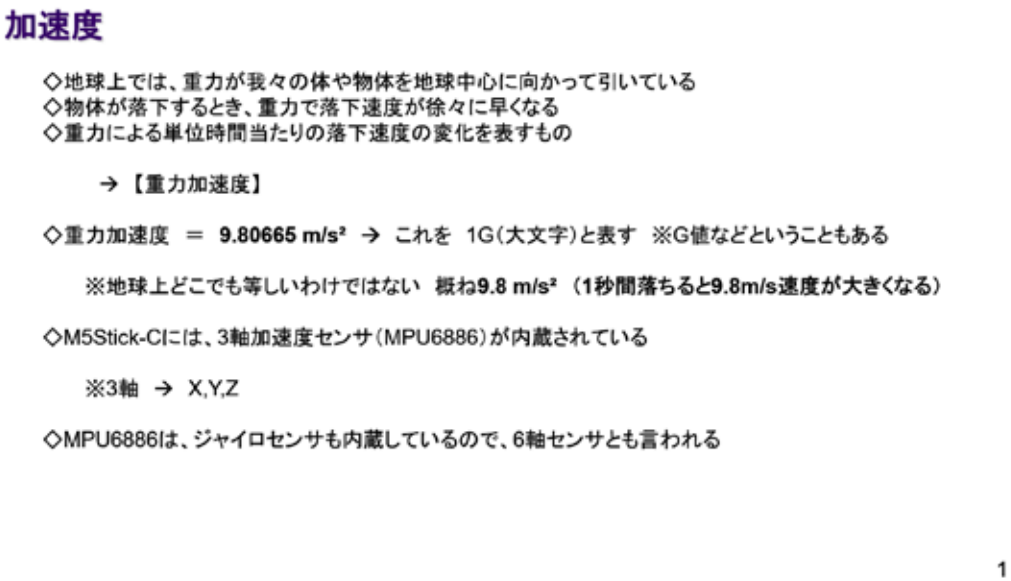


図 3-254

データシートから加速度センサ MPU6886 の特徴を読んでみよう。

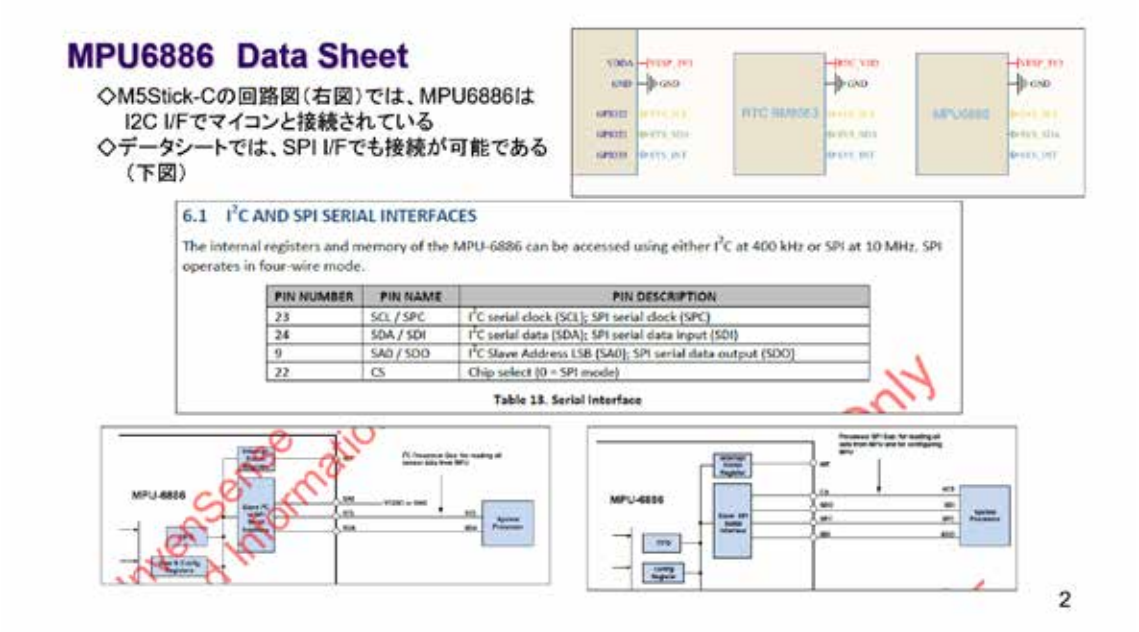


図 3-255

16bitADCを持つ3軸加速度計と信号調整

4.8 THREE-AXIS MEMS ACCELEROMETER WITH 16-BIT ADCS AND SIGNAL CONDITIONING

The MPU-6886's 3-Axis accelerometer uses separate proof masses for each axis. Acceleration along a particular axis induces displacement on the corresponding proof mass, and capacitive sensors detect the displacement differentially. The MPU-6886's architecture reduces the accelerometers' susceptibility to fabrication variations as well as to thermal drift. When the device is placed on a flat surface, it will measure 0*g* on the X- and Y-axes and +1*g* on the Z-axis. The accelerometers' scale factor is calibrated at the factory and is nominally independent of supply voltage. Each sensor has a dedicated sigma-delta ADC for providing digital outputs. The full scale range of the digital output can be adjusted to ±2*g*, ±4*g*, ±8*g*, or ±16*g*.

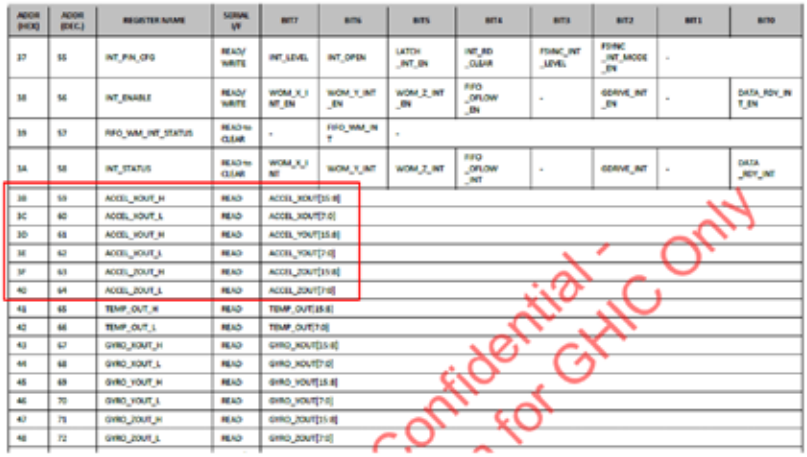
4.8 16bit ADCを持つ3軸加速度計と信号調整
MPU-6886の3軸加速度計は、各軸について別々の試験質量を用いている。特定の軸に沿った加速度は、対応する試験質量での変化を含み、容量センサが変化の差を検出している。MPU-6886の構造は、加速度計の構造変化に対する感受性を、熱ドリフトと同様に減少させている。**デバイスが平面に置かれている場合、X-軸とY-軸では0Gを、そしてZ-軸では+1Gを測定する。**この加速度計のスケール係数は、工場で校正されており、供給電圧とは表面上独立している。各センサはデジタル出力のための専用Σ-ΔADCを持っている。デジタル出力の**フルスケール範囲は±2g、±4g、±8g、または±16gに調整できる。**

3

図 3-256

MPU6886のコマンド

◇データシートには、MPU6886の初期化やデータ読み出し用レジスタが列挙されており、ライブラリではこれらをアクセスして、加速度の値を取得している



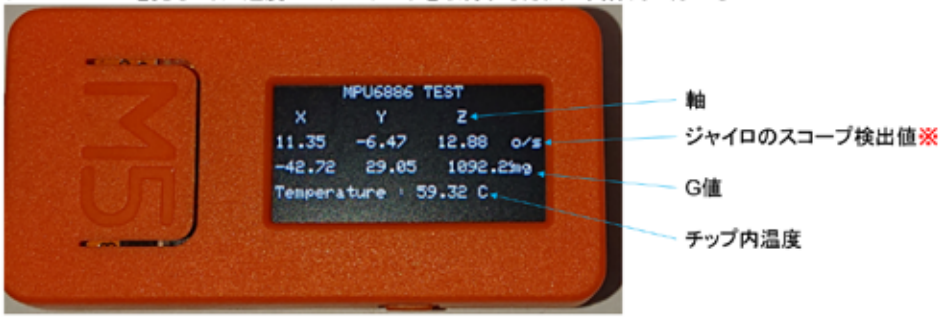
4

図 3-257

実際にシステムを開発する際には、センサ内部の各種レジスタのアドレスをライブラリ関数に渡すだけで、その内容を読み出したり、変更したりできる。センサチップとの通信は気にする必要がない。

サンプルプログラム

◇IDEには、M5Stick-C用の様々なサンプルコードが含まれている
◇IDEのメニューで ファイル → スケッチの例 → M5StickC → Basics → MPU6886 を選択すると6軸センサMPU6886ようのサンプルスケッチが開く
◇これをそのままコンパイルして実行してみよう(下図)
→ LCDにX,Y,Z軸ごとにジャイロセンサの検出値と加速度センサの検出値、センサチップ内温度が表示される
◇サンプルコードを見ると、加速度センサのデータを取得する方法が具体的に分かる



※単位にo/sとあるのは、deg/sのdegを○で表現しているらしい

5

図 3-258

IDE に含まれている次のサンプルコードを実際に動かしてみた様子を図に示している。

サンプルソースコード 1/2 （ MPU6886 ）

◇冒頭から初期化部
※6軸センサ用ライブラリはM5StickCのライブラリに含まれているので、
初期化はM5.MPU6886.Init()をCallするだけでよい

```
#include <M5StickC.h>

float accX = 0;    // 加速度・ジャイロ各3軸用
float accY = 0;
float accZ = 0;
float gyroX = 0;
float gyroY = 0;
float gyroZ = 0;
float temp = 0;    // チップ内部温度
```

```
void setup() {
  // put your setup code here, to run once:
  M5.begin();
  M5.Lcd.setRotation(3);
  M5.Lcd.fillScreen(BLACK);
  M5.Lcd.setTextSize(1);
  M5.Lcd.setCursor(40, 0);
  M5.Lcd.println("MPU6886 TEST");
  M5.Lcd.setCursor(0, 15);
  M5.Lcd.println("  X      Y      Z");
  M5.MPU6886.Init();    // 6軸センサの初期化
}
```

6

図 3-259

サンプルソースコード 2/2 （ MPU6886 ）

◇通常処理部

```
void loop() {
  M5.MPU6886.getGyroData(&gyroX,&gyroY,&gyroZ);  // ジャイロセンサデータ取得
  M5.MPU6886.getAccelData(&accX,&accY,&accZ);    // 加速度センサデータ取得
  M5.MPU6886.getTempData(&temp);                 // 温度センサデータ取得

  M5.Lcd.setCursor(0, 30);
  M5.Lcd.printf("%.2f   %.2f   %.2f     ", gyroX, gyroY,gyroZ);
  M5.Lcd.setCursor(140, 30);
  M5.Lcd.print("o/s");
  M5.Lcd.setCursor(0, 45);
  M5.Lcd.printf("%.2f   %.2f   %.2f     ",accX * 1000,accY * 1000, accZ * 1000);
  M5.Lcd.setCursor(140, 45);
  M5.Lcd.print("mg");
  M5.Lcd.setCursor(0, 60);
  M5.Lcd.printf("Temperature : %.2f C", temp);
  delay(100);
}
```

7

図 3-260

図 3-261

機械につきものの振動を測定してみよう。

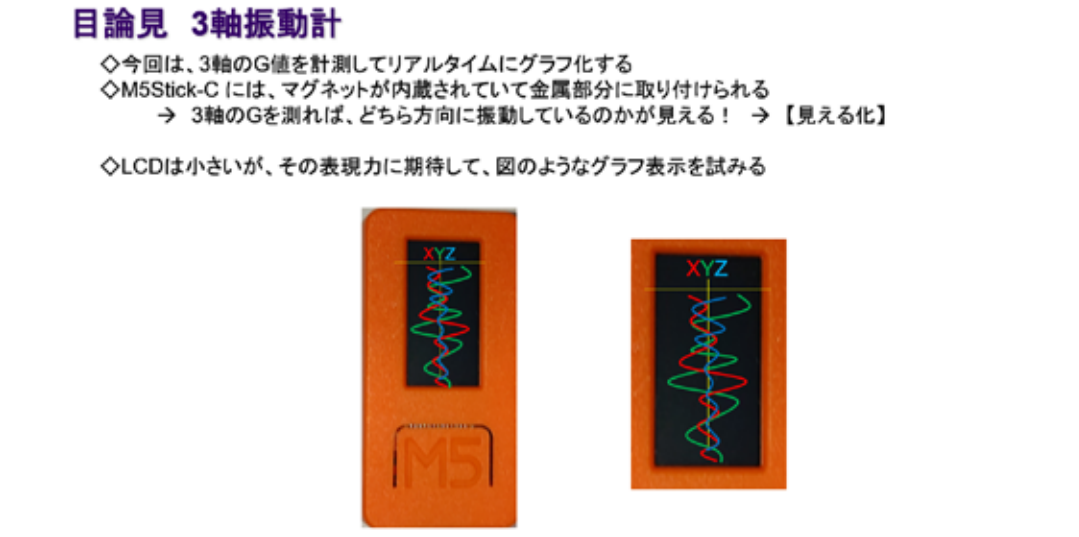
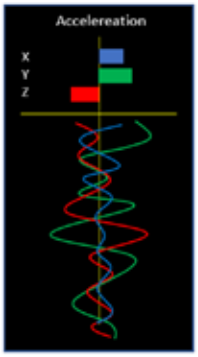
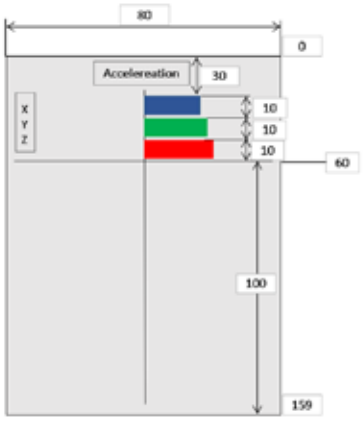


図 3-262

システム構想

◇画面設計 ： 以下のような画面にする
①. グラフ部分は棒グラフ部(中央の基準線の右側: G>=0、左側C<0)と折れ線グラフ部にする
②. 折れ線グラフ部は、最新のデータを水平基準軸下にプロットする
③. 折れ線グラフ部が下に流れるようにするために描画データを3軸分保存する必要があり
3軸×100ドットの配列を確保する





10

図 3-263

LCD で理解した座標系がここで活かせる！
以下に長いソースコードを示すが、ほとんどがグラフ描画部分である。

ソースコード 1/9 （M5C_ACC_GRAPH_1 ）

◇冒頭から変数宣言

```
#include <M5StickC.h>

#define maxG 30  // グラフにする際の1G当たりのドット数

float accX = 0.0F;   // X方向加速度
float accY = 0.0F;   // Y方向加速度
float accZ = 0.0F;   // Z方向加速度

int AX;              // グラフのためのX軸加速度
int AY;              // グラフのためのY軸加速度
int AZ;              // グラフのためのZ軸加速度
int i=0;             // 描画データ配列用インデックス
int acv[3][100];     // 3軸の加速度データ100プロット分
```

```
float gyroX = 0.0F;  // ジャイロX
float gyroY = 0.0F;  // ジャイロY
float gyroZ = 0.0F;  // ジャイロZ

float pitch = 0.0F;  // ピッチ
float roll  = 0.0F;  // ロール
float yaw   = 0.0F;  // ヨー

int mode = 1; // 出力モード
  //(0:ジャイロ  1:加速度  2:ピッチ・ロール・ヨー)
```

11

図 3-264

ソースコード 2/9 （M5C_ACC_GRAPH_1）

◇初期化部 前半

```
void setup() {                          // 初期化部
  M5.begin();                           // M5Stick-C初期化
  M5.IMU.Init();                        // 慣性測定装置（IMU：Inertial Measurement Unit）の初期化

  M5.Lcd.setTextDatum(0);               // LCDの向き設定（縦長）
  M5.Lcd.fillScreen(BLACK);             // 黒で塗りつぶし
  M5.Lcd.setCursor(3, 10);              // カーソル位置
  M5.Lcd.setTextColor(WHITE);           // 文字色　白
  M5.Lcd.setTextSize(1);                // 文字の大きさ
  M5.Lcd.printf("Acceleration");        // タイトル

  M5.Lcd.drawLine(10, 60, 70, 60, BLUE); //---  グラフの水平基準軸
  M5.Lcd.drawLine(40, 20, 40, 159, BLUE); // |   グラフの垂直基準軸

  M5.Lcd.setCursor(29, 20);             // ＋-表示のカーソル位置
  M5.Lcd.printf("-  +");                // ＋　－　表示
```

12

図 3-265

ソースコード 3/9 （M5C_ACC_GRAPH_1）

◇初期化部 後半

```
 //BAR (x, y, xw, yw, c)
  M5.Lcd.fillRect(41, 30, 10, 8, BLUE);   // 棒グラフ
  M5.Lcd.fillRect(41, 40, 10, 8, GREEN);
  M5.Lcd.fillRect(41, 50, 10, 8, RED);

  //加速度データ配列初期化
  for(i=0;i<100;i++){
    acv[0][i]=0;
    acv[1][i]=0;
    acv[2][i]=0;
  }
}
```

13

図 3-266

ソースコード 4/9 （ M5C_ACC_GRAPH_1 ）

◇通常処理部 序盤

```
void loop() {  // 通常処理部
  M5.update();  // M5Stick-C内部更新（ボタン押下状況）
  M5.IMU.getGyroData(&gyroX, &gyroY, &gyroZ);  // IMUから各データを読み出す
  M5.IMU.getAccelData(&accX, &accY, &accZ);
  M5.IMU.getAhrsData(&pitch, &roll, &yaw);

  if ( mode == -1 || M5.BtnA.wasReleased() ) {  // ボタンが押されていたら出力モード変更
    mode++;
    mode = mode % 3;

    // プロッタ用のタイトル出力
    if ( mode == 0 ) {
      Serial.printf("gyroX,gyroY,gyroZ¥n");
    } else if ( mode == 1 ) {
      Serial.printf("accX,accY,accZ¥n");
    } else if ( mode == 2 ) {
      Serial.printf("pitch,roll,yaw¥n");
    }
  }
```

14

図 3-267

ソースコード 5/9 （ M5C_ACC_GRAPH_1 ）

◇通常処理部 中盤

```
  if ( mode == 0 ) {                                          // データ出力
    Serial.printf("%6.2f,%6.2f,%6.2f¥n", gyroX, gyroY, gyroZ);  // 角加速度
  } else if ( mode == 1 ) {
    Serial.printf("%5.2f,%5.2f,%5.2f¥n", accX, accY, accZ);     // 重力加速度
  } else if ( mode == 2 ) {
    Serial.printf("%5.2f,%5.2f,%5.2f¥n", pitch, roll, yaw);     // ピッチ・ロール・ヨー
  }

  //BAR (x, y, xw, yw, c)
  M5.Lcd.fillRect(0, 30, 80, 30, BLACK);                       //棒グラフ領域のクリア

  AX = (int)(accX * maxG);                                     // グラフ用加速度値の計算
  AY = (int)(accY * maxG);
  AZ = (int)(accZ * maxG);
  acv[0][i]=AX;                                                // 現在の表示値格納
  acv[1][i]=AY;
  acv[2][i]=AZ;
  if(++i >= 100)(i = 0;}   // 配列はリングバッファ--->インデックス更新
```

15

ソースコード 6/9 （ M5C_ACC_GRAPH_1 ）

◇通常処理部 棒グラフ部

```
if(AX > 0){                                    // 正のX棒グラフ
  M5.Lcd.fillRect(41, 30, AX, 8, BLUE);
}else{                                         // 負のX棒グラフ
  M5.Lcd.fillRect(40+AX, 30, -(AX), 8, BLUE);
}

if(AY > 0){                                    // 正のY棒グラフ
  M5.Lcd.fillRect(41, 40, AY, 8, GREEN);
}else{                                         // 負のY棒グラフ
  M5.Lcd.fillRect(40+AY, 40, -(AY), 8, GREEN);
}

if(AZ > 0){                                    // 正のZ棒グラフ
  M5.Lcd.fillRect(41, 50, AZ, 8, RED);
}else{                                         // 負のZ棒グラフ
  M5.Lcd.fillRect(40+AZ, 50, -(AZ), 8, RED);
}
```

16

図 3-268

ソースコード 7/9 （ M5C_ACC_GRAPH_1 ）

◇通常処理部 基準軸と凡例

```
M5.Lcd.drawLine(40, 20, 40, 60, BLUE); // | 垂直基準軸　上書き
M5.Lcd.setCursor(2, 30);               // 凡例 X
M5.Lcd.printf("X");

M5.Lcd.setCursor(2, 40);               // 凡例 Y
M5.Lcd.printf("Y");

M5.Lcd.setCursor(2, 50);               // 凡例 Z
M5.Lcd.printf("Z");

accPlot(i);                            // 折れ線グラフ部（と言ってもドット）描画
delay(50);
}
```

17

図 3-269

ソースコード 8/9 （M5C_ACC_GRAPH_1）

◇折れ線グラフ部 前半

```
void accPlot(int i){   // 折れ線グラフ部描画関数
  int n;
  int y;

  //グラフ領域のクリア
  //BAR (x, y, xw, yw, c)
  M5.Lcd.fillRect(0, 61, 80, 99, BLACK);
  //グラフのXY軸
  M5.Lcd.drawLine(10, 60, 70, 60, BLUE);  //---
  M5.Lcd.drawLine(40, 20, 40, 159, BLUE); // |

  if(i==0){
    n=99;
  }else{
    n=i-1;
  }
```

18

図 3-270

ソースコード 9/9 （M5C_ACC_GRAPH_1）

◇折れ線グラフ部 後半

```
for(y=0;;y++){
    if(acv[0][n] >= 0){
      M5.Lcd.drawPixel(41 + acv[0][n], 61 + y, BLUE); // 正のX
    }else{
      M5.Lcd.drawPixel(40 + acv[0][n], 61 + y, BLUE); // 負のX
    }
    if(acv[1][n] >= 0){
      M5.Lcd.drawPixel(41 + acv[1][n], 61 + y, GREEN); // 正のY
    }else{
      M5.Lcd.drawPixel(40 + acv[1][n], 61 + y, GREEN); // 負のY
    }
    if(acv[2][n] >= 0){
      M5.Lcd.drawPixel(41 + acv[2][n], 61 + y, RED); // 正のZ
    }else{
      M5.Lcd.drawPixel(40 + acv[2][n], 61 + y, RED); // 負のZ
    }
    n--;
      if(n<0){n=99;}
      if(n==i){break;}
    }
}
```

19

図 3-271

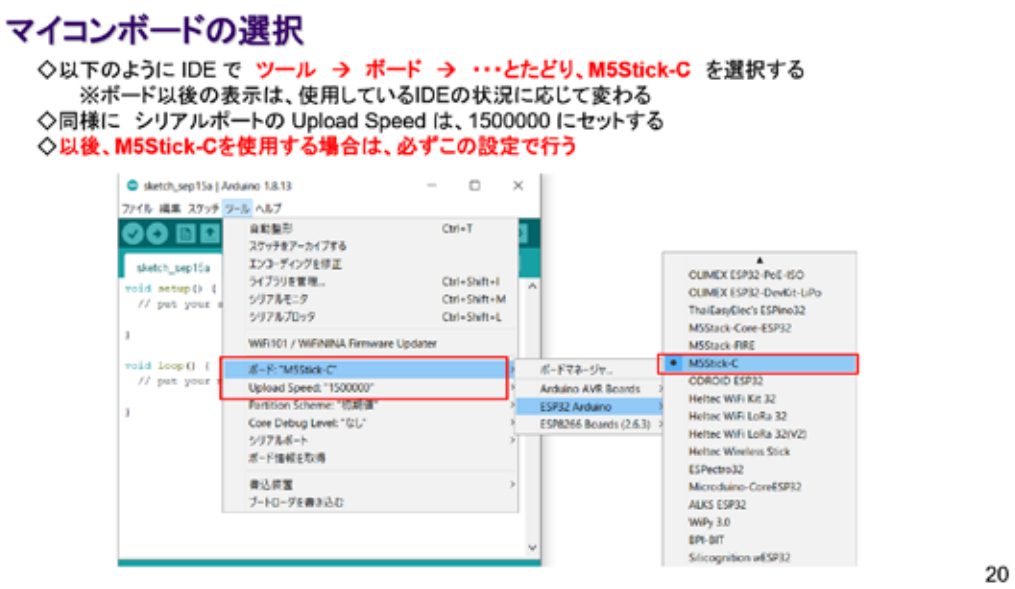

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う
20

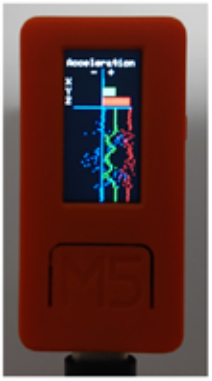
図 3-272

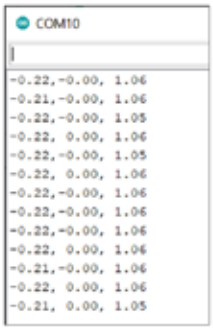

マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
21

図 3-273

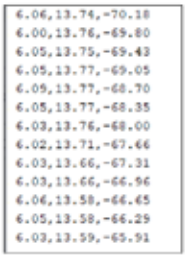
動作確認　1/2

◇プログラムが書き込まれると、マイコンが重力加速度Gを検出してグラフが左図のように描画される
◇シリアルモニタを起動すれば右図のように、計測した値が表示される
◇Aボタンを押下すると、シリアルモニタの表示が切り替わる（G値→角加速度→ピッチ・ロール・ヨー）

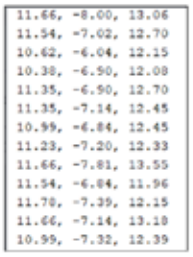




G値



角加速度



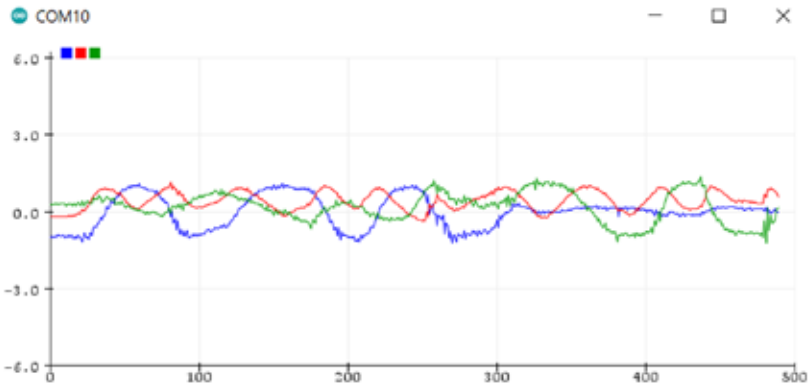
ピッチ・ロール・ヨー

22

図 3-274

動作確認　2/2

◇シリアルモニタウインドウを閉じ、 IDEメニューで ツール → シリアルプロッタ を選択する
◇プログラムで各軸のG値を【 , 】カンマで区切って出力すればPCでも簡易グラフを描画できる
※ただし、データの記録はできない



23

図 3-275

シリアルプロッタは、その仕様に従ったマイコン側からの送信が必要だが、理解していればセンサ検出値を確認するのにとても役立つ機能である。

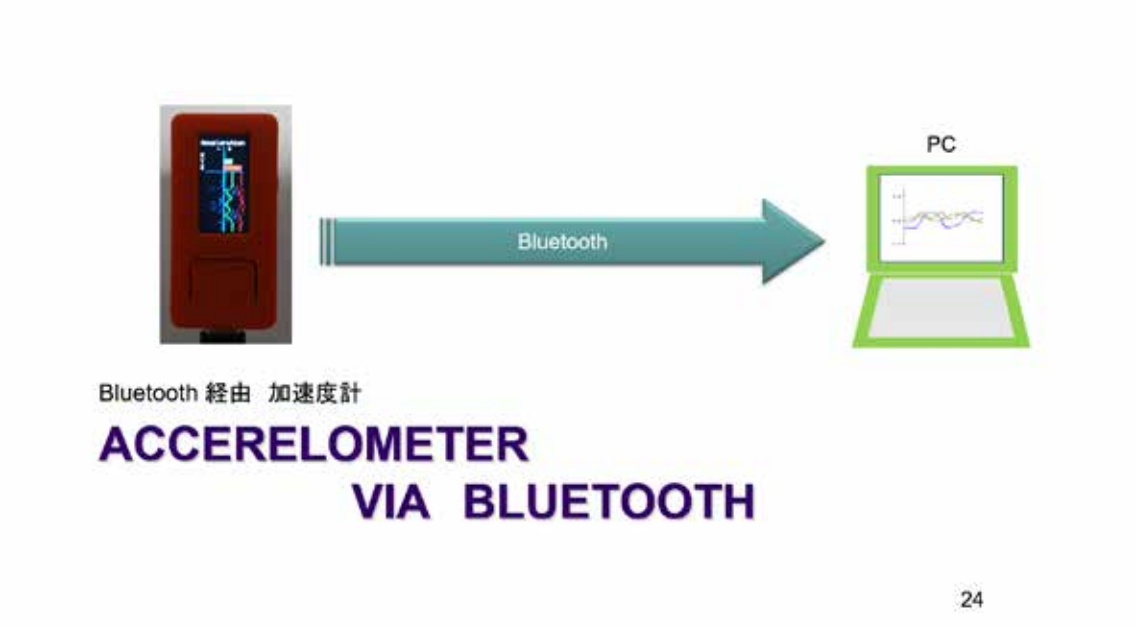


図 3-276

Bluetooth を活用してみる。

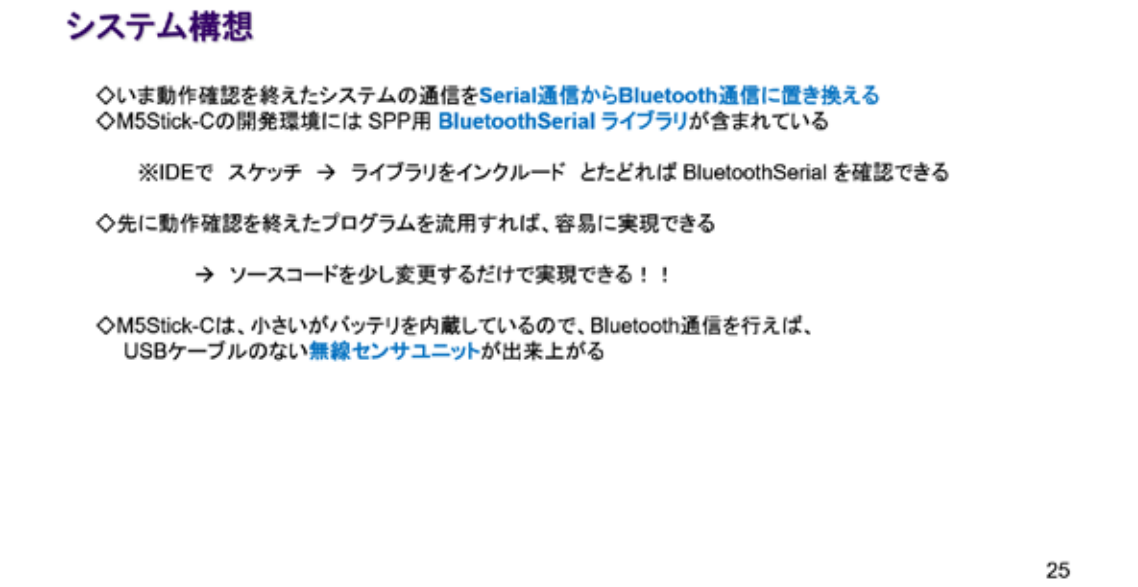


図 3-277

以下に示すのは、追加・変更部分だけである。

ソースコード 1/2 （ M5C_ACC_GRAPH_Bluetooth_2 ）

◇初期化処理部 追加･変更する部分のみ下記する
◇冒頭部分

※青字部分を追加･変更する

```
#include <M5StickC.h>
#include <BluetoothSerial.h>      // Bluetooth Serial ライブラリ

#define maxG 30                   // G値をグラフにする際の1G当たりのドット数

BluetoothSerial bts;              // Bluetooth Object

float accX = 0.0F;                // X方向加速度
float accY = 0.0F;                // Y方向加速度
float accZ = 0.0F;                // Z方向加速度
                ・・・      ・・・

void setup() {                    // 初期化部
  M5.begin();                     // M5Stick-C初期化
  bts.begin("＊＊＊＊＊");        // PC側でペアリングするデバイス名称（ユニークな名称）
  M5.IMU.Init();                  // 慣性測定装置（IMU：Inertial Measurement Unit）の初期化
                ・・・      ・・・
```

26

図 3-278

ソースコード 2/2 （ M5C_ACC_GRAPH_Bluetooth_2 ）

◇通常処理部 追加･変更する部分のみ下記する

※青字部分を追加･変更する

```
  // プロッタ用のタイトル出力
  if ( mode == 0 ) {
    bts.printf("gyroX,gyroY,gyroZ¥n");          // Serial→btsに変更
  } else if ( mode == 1 ) {
    bts.printf("accX,accY,accZ¥n");             // Serial→btsに変更
  } else if ( mode == 2 ) {
    bts.printf("pitch,roll,yaw¥n");             // Serial→btsに変更
  }
}
 // データ出力
if ( mode == 0 ) {
  bts.printf("%6.2f,%6.2f,%6.2f¥n", gyroX, gyroY, gyroZ);      // Serial→btsに変更
} else if ( mode == 1 ) {
  bts.printf("%5.2f,%5.2f,%5.2f¥n", accX, accY, accZ);         // Serial→btsに変更
} else if ( mode == 2 ) {
  bts.printf("%5.2f,%5.2f,%5.2f¥n", pitch, roll, yaw);         // Serial→btsに変更
}
                    ・・・      ・・・
```

27

図 3-279

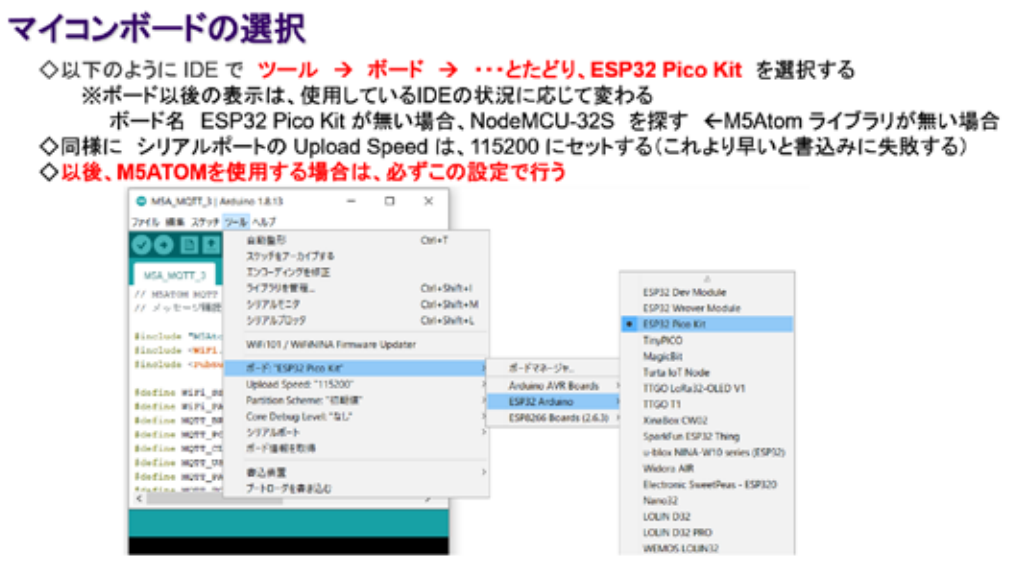

マイコンボードの選択
◇以下のように IDE で ツール → ボード → ・・・とたどり、ESP32 Pico Kit を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
ボード名 ESP32 Pico Kit が無い場合、NodeMCU-32S を探す ←M5Atom ライブラリが無い場合
◇同様に シリアルポートの Upload Speed は、115200 にセットする（これより早いと書込みに失敗する）
◇以後、M5ATOMを使用する場合は、必ずこの設定で行う
28

図 3-280


マイコンをPCと接続
①. M5Stick-CをPCと接続
開発用PC
M5Stick-C
USB-C
ケーブル
②. デバイスマネージャでCOMポート確認
③. シリアルポート設定
④. コンパイル
⑤. 書込み完了
29

図 3-281


動作確認 1/4 Bluetooth Device の追加（ペアリング）
◇書込みが完了したら、数秒待って、以下の手順でデバイスのペアリングを行う
①スタート→設定
②デバイス
③Bluetoothとその他のデバイス
デバイス
Bluetooth、プリンター、マウス
Bluetooth とその他のデバイス
Bluetooth またはその他のデバイスを追加する
Bluetooth
オン
④デバイスを追加する
デバイスを追加する
⑤プログラム指定のデバイス名選択
デバイスを追加する
M5Stick-C_KEN
⑥ペアリング完了
デバイスの準備が整いました!
30

図 3-282

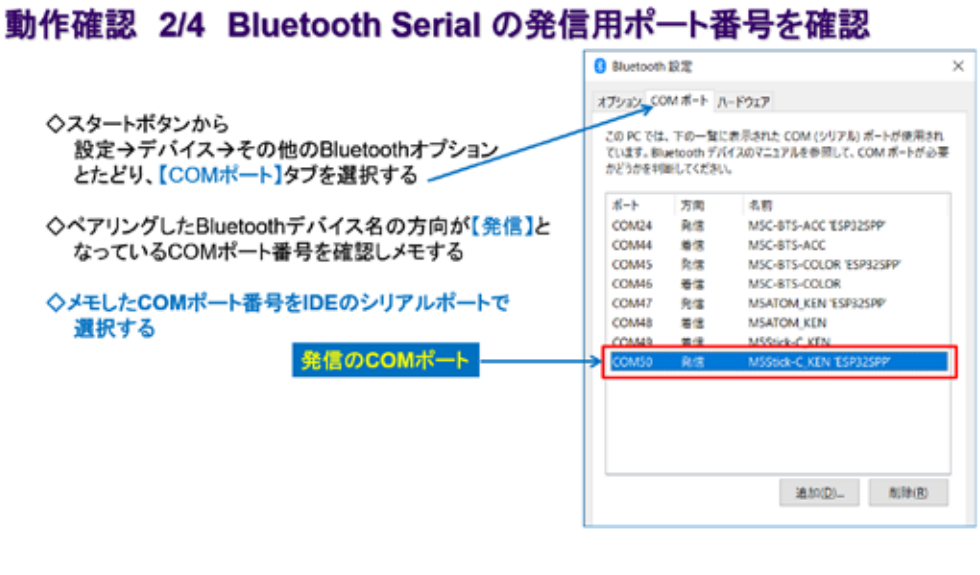
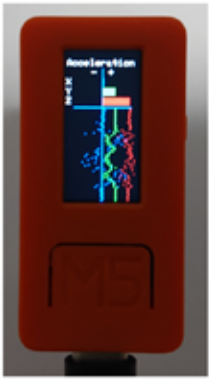
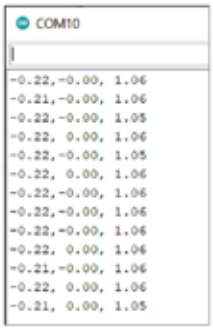

動作確認 2/4 Bluetooth Serial の発信用ポート番号を確認
◇スタートボタンから
設定→デバイス→その他のBluetoothオプション
とたどり、【COMポート】タブを選択する
◇ペアリングしたBluetoothデバイス名の方向が【発信】と
なっているCOMポート番号を確認しメモする
◇メモしたCOMポート番号をIDEのシリアルポートで
選択する
発信のCOMポート
Bluetooth 設定
オプション
COM ポート
ハードウェア
ポート
方向
名前
COM24 発信 MSC-BTS-ACC 'ESP32SPP'
COM44 着信 MSC-BTS-ACC
COM45 発信 MSC-BTS-COLOR 'ESP32SPP'
COM46 着信 MSC-BTS-COLOR
COM47 発信 MSATOM_KEN 'ESP32SPP'
COM48 着信 MSATOM_KEN
COM50 発信 M5Stick-C_KEN 'ESP32SPP'
追加(D)...
削除(R)
31

図 3-283

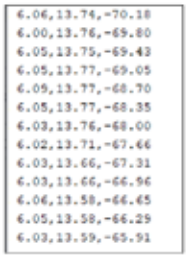
動作確認　3/4

◇プログラムが書き込まれると、マイコンが重力加速度Gを検出してグラフが左図のように描画される
◇シリアルモニタを起動すれば右図のように、計測した値が表示される
◇Aボタンを押下すると、シリアルモニタの表示が切り替わる（G値→角加速度→ピッチ・ロール・ヨー）

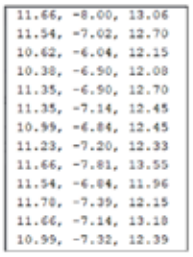




G値



角加速度



ピッチ・ロール・ヨー

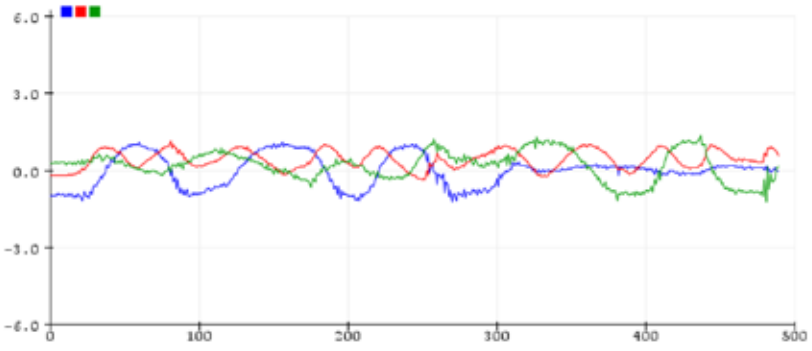
32

図 3-284

動作確認　4/4

◇シリアルモニタウインドウを閉じ、 IDEメニューで ツール → シリアルプロッタ を選択する
◇プログラムで各軸のG値を【 , 】カンマで区切って出力すればPCでも簡易グラフを描画できる
※ただし、データの記録はできない

※M5Stick-Cはバッテリを内蔵しているので、USBケーブルを外しても、このプロットは続けることができる



33

図 3-285


μ Factory の実現
μF
マイクロ工場
振動
工作機械
光
マイコン
自動機械
Bluetooth
WiFi
34

図 3-286


スチールプレート & マグネット
◇実習キットには、次のものが含まれれいる
①. スチールプレート
②. マグネット付きサーボモータ
※すでにサーボモータ制御で利用したかもしれないが、マグネットで
サーボモータをスチールプレートに固定すると、動作確認しやすい
◇M5Stick-Cの背面にもマグネットが内蔵されているので、
これを利用すると、右上図のように、M5Stick-Cをサーボホーンに
固定することができる
◇この状態でサーボモータを動作させれば、加速度センサで往復回転時の
振動計測ができる
→ 工場設備の振動を測定する極小モデル = μ工場 となる
スチールプレート
マグネット
35

図 3-287

３．１４　M5Stick-C　－　カラーセンサ

図 3-288

光学系のセンサを使ってみよう。

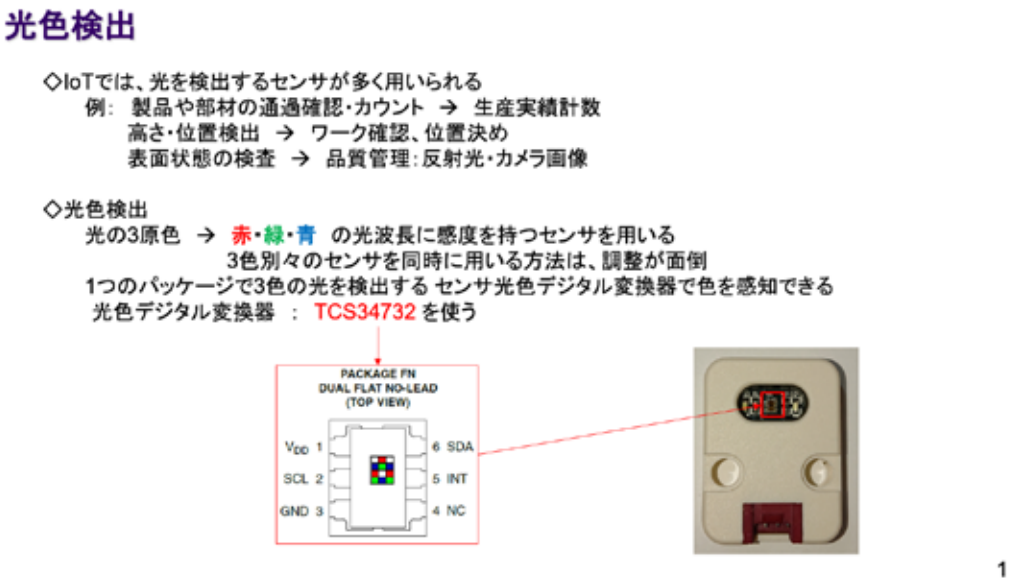


図 3-289

TSC3472 Data Sheet Description

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials.

The TCS3472 color sensor has a wide range of applications including RGB LED backlight control, solid-state lighting, health/fitness products, industrial process controls and medical diagnostic equipment. In addition, the IR blocking filter enables the TCS3472 to perform ambient light sensing (ALS). Ambient light sensing is widely used in display-based products such as cell phones, notebooks, and TVs to sense the lighting environment and enable automatic display brightness for optimal viewing and power savings. The TCS3472, itself, can enter a lower-power wait state between light sensing measurements to further reduce the average power consumption.

説明

TCS3472デバイスは、赤・緑・青(RGB)そしてクリアの光感知値を提供する。色検知光ダイオードのためにチップ上で集積されて限定された赤外光遮蔽フィルタが入射光の赤外光成分を最小化して、色測定を高精度化できる。高感度で広いダイナミックレンジ、そして赤外光遮蔽フィルタがTCS3472を、変化する照明条件や光を減衰させる物質を通した利用に対する理想的な色センサの解決策にする。

RGB LEDのバックライト制御、固体照明、健康/フィットネス製品、工業プロセス制御、医療診断装置を含む応用の広い幅を持っている。さらに、赤外光遮蔽フィルターは、環境光感知(ALS)性能をTCS3472にもたらしている。環境光感知は、光学的な鑑賞(見た目)や電力制限のために、携帯電話、ノートPC、そしてTVのような表示に主な機能をもつ製品に幅広く用いられており、自動表示輝度調整を可能にする。TCS3472 は平均電力消費の削減を進めるために、光感知測定の間、低電力待ち状態に入ることができる。

2

図 3-290

上記はデータシート冒頭の概略説明。詳細説明は、データシートを少し読み進んだ部分に記載されている。

TSC3472 Detailed Description 1/2

The TCS3472 light-to-digital converter contains a 3 × 4 photodiode array, four analog-to-digital converters (ADC) that integrate the photodiode current, data registers, a state machine, and an I^2C interface. The 3 × 4 photodiode array is composed of red-filtered, green-filtered, blue-filtered, and clear (unfiltered) photodiodes. In addition, the photodiodes are coated with an IR-blocking filter. The four integrating ADCs simultaneously convert the amplified photodiode currents to a 16-bit digital value. Upon completion of a conversion cycle, the results are transferred to the data registers, which are double-buffered to ensure the integrity of the data. All of the internal timing, as well as the low-power wait state, is controlled by the state machine.

詳細説明

TCS3472 光デジタル変換器は3×4光ダイオードアレイと光ダイオード電流を積分する 4個のアナログ-デジタル変換器(ADC)、光データレジスタ、ステートマシン、および I2C I/F を持っている。3×4光ダイオードアレイは、赤、緑、青フィルタの掛けられたものと、クリア(フィルタなし)の光ダイオードから構成されている。さらに、光ダイオードは赤外光遮断フィルタでコートされている。 4つの集積ADCは、増幅された光電流を16bitデジタル値に同時変換する。変換サイクルの完了で、その結果は、データレジスタに送られる。それは、データの蓄積を保証するためダブルバッファになっている。低電力待ち状態と同じように、すべての内部タイミングはステートマシンで制御されている。

3

図 3-291

TSC3472 Detailed Description 2/2

Communication of the TCS3472 data is accomplished over a fast, up to 400 kHz, two-wire I²C serial bus. The industry standard I²C bus facilitates easy, direct connection to microcontrollers and embedded processors.

In addition to the I²C bus, the TCS3472 provides a separate interrupt signal output. When interrupts are enabled, and user-defined thresholds are exceeded, the active-low interrupt is asserted and remains asserted until it is cleared by the controller. This interrupt feature simplifies and improves the efficiency of the system software by eliminating the need to poll the TCS3472. The user can define the upper and lower interrupt thresholds and apply an interrupt persistence filter. The interrupt persistence filter allows the user to define the number of consecutive out-of-threshold events necessary before generating an interrupt. The interrupt output is open-drain, so it can be wire-ORed with other devices.

TC3472データの通信は、400kHzに上る高速の2線式I2Cシリアルバスで達成される。工業規格I2Cバスは、マイクロコントローラと組み込まれている計算機の直接接続を簡単に実現する。
さらに、I2Cバスに対してTXS34732は、別の割込み信号出力を提供する。割込みが可能な場合、そしてユーザー定義閾値を超える場合に、アクティブロウ割込みが宣言されて、それがプロセッサによってクリアされるまでは、そのまま残っている。この割込みの特徴は、TCS3472問い合わせの必要を無くすことで、システムソフトウエアの効率を向上させている。ユーザは上位、低位の割込み閾値を決めて、割込み持続フィルタに適用することができる。その割込み持続フィルタは、ユーザーが割り込み生成前に必要な、連続する閾値外イベントの数を決められるようにしている。割り込み出力は、オープンドレインなので、他のデバイスとワイヤードオアすることができる。

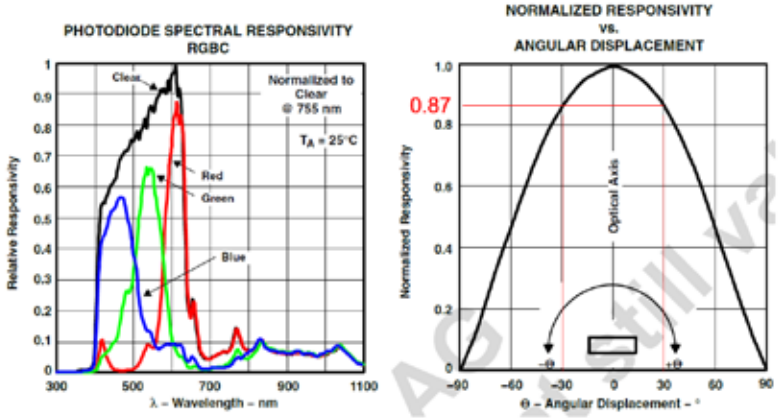
4

図 3-292

TSC3472 Data Sheet

◇光ダイオードスペクトル感度および規格化感度対角度変異

①. およそ650nm波長まで感知する性能がある
②. ±30度の角度範囲で80%以上の応答性能



5

図 3-293

左図に示すグラフで、センサに用いている光ダイオードは、RGB 各色に感度を持つことが分かる。センサユニットの正面から±30 度の角度範囲で利用するのが良さそう（右図）。

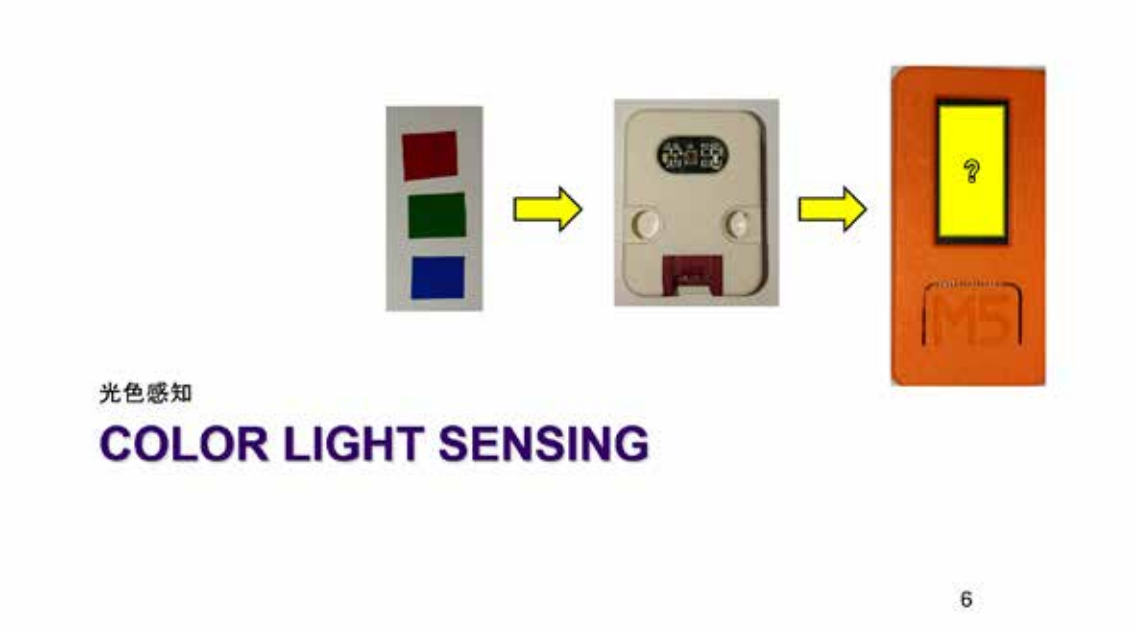


図 3-294

図 3-295

実習キットでは黄色を含む 4 色のセロファン紙を準備した。

図 3-296

このセンサチップのライブラリも公開されている。ソースコードを記述する前に、あらかじめインストールしておく。

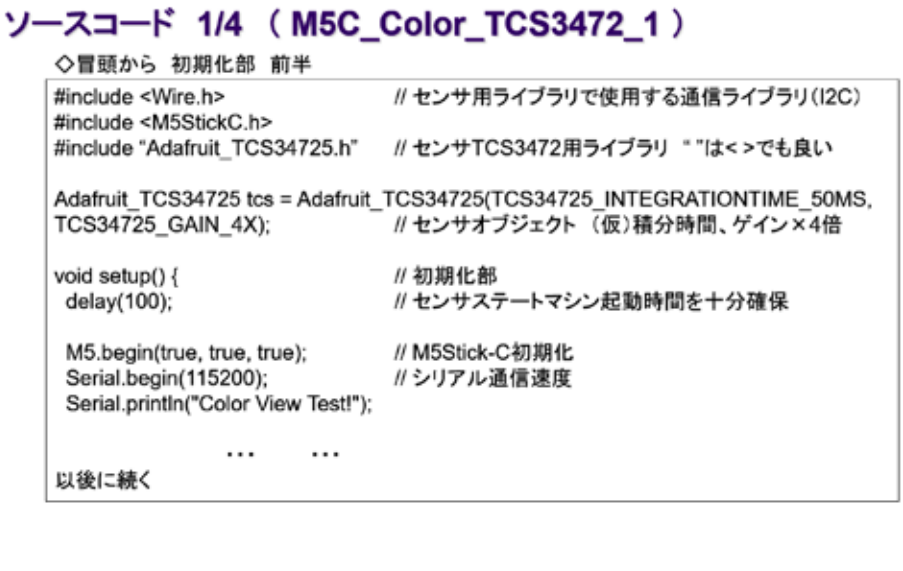


図 3-297

ソースコード 2/4 （ M5C_Color_TCS3472_1 ）

◇初期化部 後半

```
続き
                    …      …

while(!tcs.begin()){     // 初期化に失敗した場合、センサーが接続されていないと判断する
    Serial.println("No TCS34725 found ... check your connections");
    M5.Lcd.setTextFont(1);
    M5.Lcd.setTextColor(WHITE);
    M5.Lcd.drawString("No Found sensor.",0,0);
    delay(1000);       // さらに1秒待つ
 }
 tcs.setIntegrationTime(TCS34725_INTEGRATIONTIME_154MS); // 積分時間154MS
 tcs.setGain(TCS34725_GAIN_4X);                          // ゲイン×4倍
}
```

10

図 3-298

ソースコード 3/4 （ M5C_Color_TCS3472_1 ）

◇通常処理部 前半

```
void loop() {
 uint16_t clear, red, green, blue;

 delay(60);  // takes 50ms to read

 tcs.getRawData(&red, &green, &blue, &clear);     // 生データ取り込み

 // Figure out some basic hex code for visualization
 uint32_t sum = clear;
 float r, g, b;
 r = red; r /= sum;                               // クリア光の値で各色を規格化する（最大＝1.0）
 g = green; g /= sum;
 b = blue; b /= sum;
 r *= 256; g *= 256; b *= 256;                    // 各色を8bit（最大256）にする
 uint16_t c = M5.Lcd.color565((int)r, (int)g, (int)b);   // 各色256を16bit値（565）に変換
 M5.Lcd.fillScreen(c);                            // 変換したRGB565コードでLCDを塗りつぶす

                    …      …
続く
```

11

図 3-299

ソースコード 4/4 （ M5C_Color_TCS3472_1 ）

◇通常処理部 後半

```
続き
                    …      …

 Serial.print("RGB=");
 Serial.print(",");          // なぜここに(,)カンマを入れているかは、後で分かる
 Serial.print(r);            // 赤の値(Max256)
 Serial.print(",");
 Serial.print(g);            // 緑の値(Max256)
 Serial.print(",");
 Serial.print(b);            // 青の値(Max256)
 Serial.println("");

 delay(100);                 // しばし待つ
}
```

12

図 3-300

センサの接続

◇プログラム書込み前に、あらかじめセンサを
M5Stick-Cと接続しておく

◇右図のように、センサ付属の専用ケーブル両端を
センサとM5Stick-CのGroveコネクタに差し込む

※コネクタには向きがあり、反対向きには
差し込めないようになっている

注意）実験が終わって、ケーブルを取り外す際は、
ケーブルにテンションを掛けず、コネクタを引いて
外すこと！

◇センサを接続後、USBケーブルでM5Stick-Cと
PCを接続する

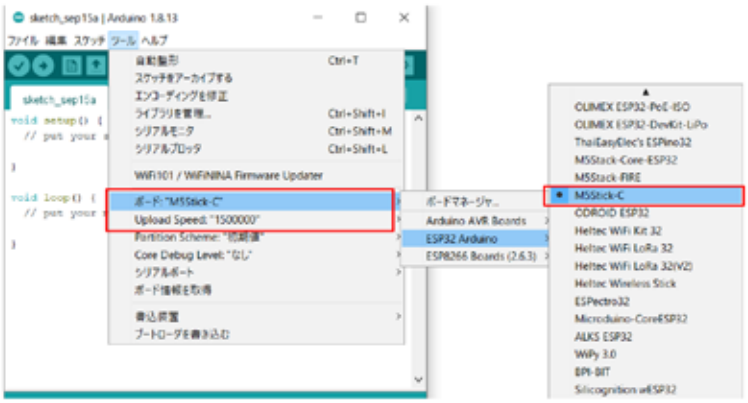
13

図 3-301

センサユニットとマイコンは、専用の Grove ケーブル（4 線）で接続する。コネクタには向きがあるので、接続の際は、確認する。※取り外す際はコネクタを引く。ケーブルを直に引っ張らないこと。

マイコンボードの選択

◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



14

図 3-302

15

図 3-303

図 3-304

上右図のように、感知窓にセロファン紙を載せて光色を感知させる。感知した色で、そのまま LCD 塗りつぶしを行っている。人の目で感じる色と、センサーが感知する色は、違いがある。セロファン紙は半透明で、環境光を通すが、他の物ではどうか？センサユニットには、反射光を検出するための LED が点灯している。

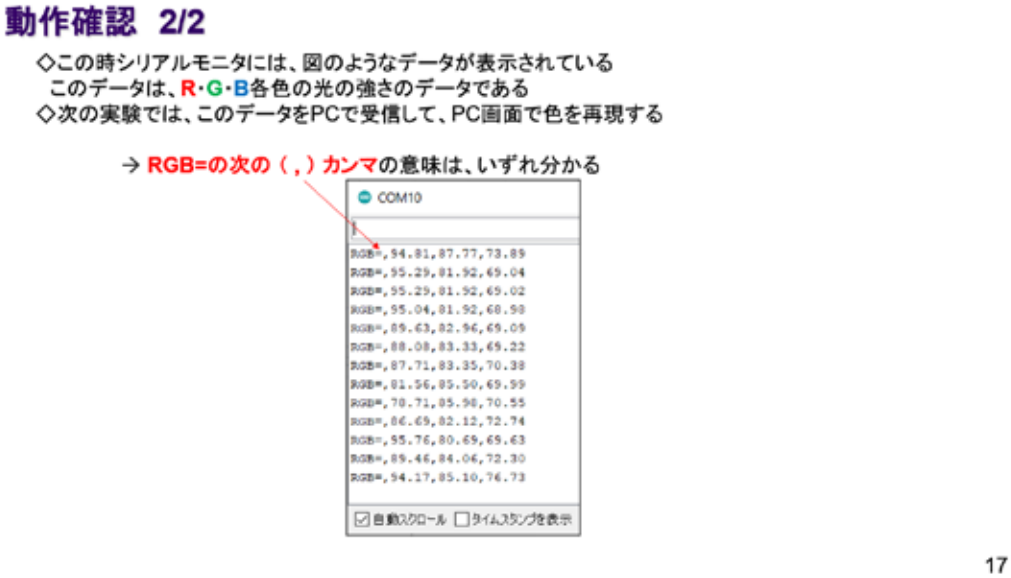


図 3-305

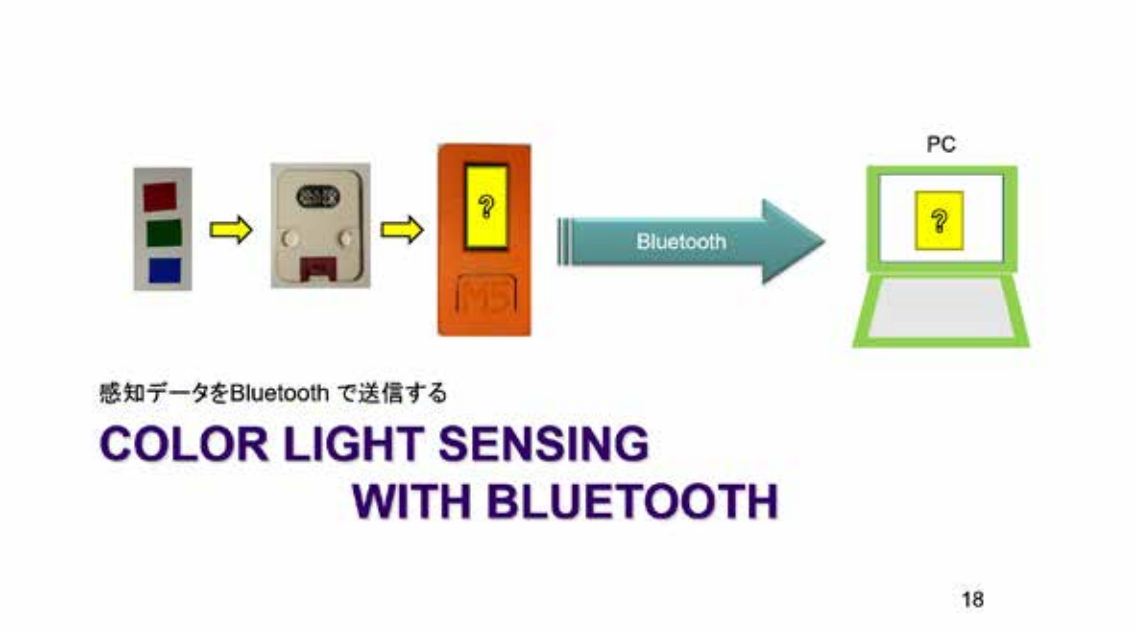


図 3-306

ここで検出した色情報を Bluetooth でホストコンピュータに送信してみよう。

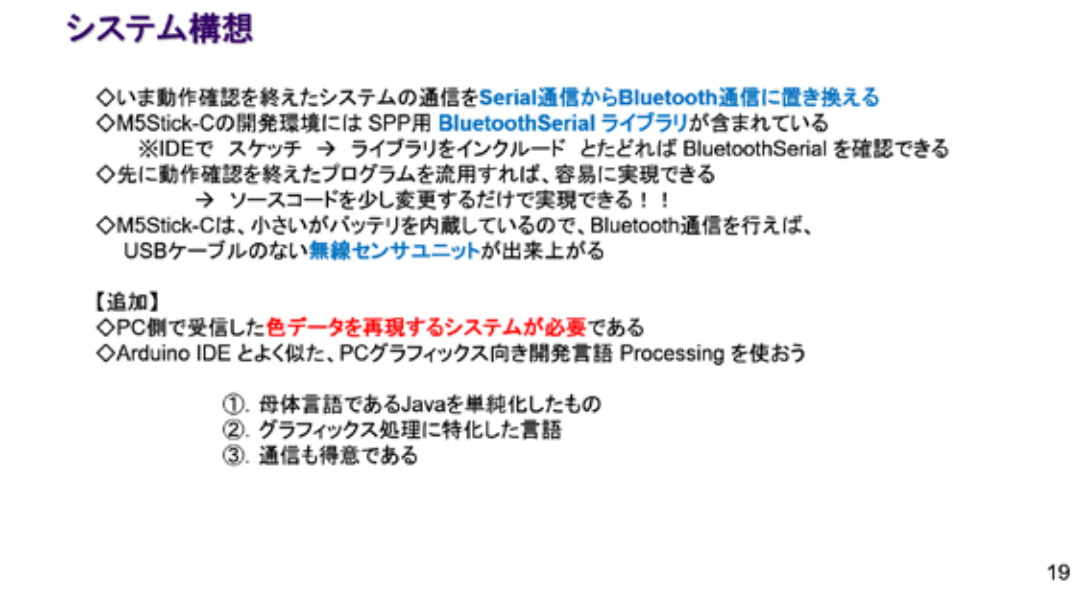


図 3-307

基になるシステムは出来上がっているので、Bluetooth に対応する部分だけを追加・変更すればよい。

ソースコード 1/2 （M5C_COLOR_TCS3472_BLT_2）

◇初期化処理部 追加・変更する部分のみ下記する
◇冒頭部分

※青字部分を追加・変更する

```
#include <Wire.h>
#include <M5StickC.h>
#include "Adafruit_TCS34725.h"   // センサTCS3472用ライブラリ  " "は< >でも良い
#include <BluetoothSerial.h>  //

Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);
BluetoothSerial bts;  //


void setup() {   // 初期化部
  bts.begin("＊＊＊＊＊"); // ペアリングに必要なデバイス名。各自ユニークな名称にすること！！
  delay(100);    // ステートマシン起動時間を十分確保

                    …          …
```

20

図 3-308

ソースコード 2/2 （M5C_COLOR_TCS3472_BLT_2）

◇通常処理部の最後 追加・変更する部分のみ下記する

※青字部分を追加・変更する

```
void loop() {
                    …          …
                    …          …

  bts.print("RGB=");
  bts.print(",");            // なぜここに(,)カンマを入れているかは、後で分かる
  bts.print(r);
  bts.print(",");
  bts.print(g);
  bts.print(",");
  bts.print(b);
  bts.println("");

  delay(100);                // しばし待つ
}
```

21

図 3-309

センサの接続

◇プログラム書込み前に、あらかじめセンサを
M5Stick-Cと接続しておく

◇右図のように、センサ付属の専用ケーブル両端を
センサとM5Stick-CのGroveコネクタに差し込む

※コネクタには向きがあり、反対向きには
差し込めないようになっている

注意）実験が終わって、ケーブルを取り外す際は、
ケーブルにテンションを掛けず、コネクタを引いて
外すこと！

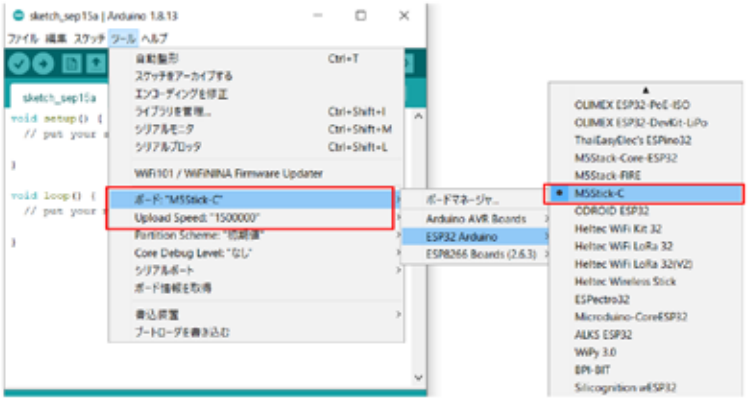
◇センサを接続後、USBケーブルでM5Stick-Cと
PCを接続する

22

図 3-310

マイコンボードの選択

◇以下のように IDE で ツール → ボード → ・・・とたどり、M5Stick-C を選択する
※ボード以後の表示は、使用しているIDEの状況に応じて変わる
◇同様に シリアルポートの Upload Speed は、1500000 にセットする
◇以後、M5Stick-Cを使用する場合は、必ずこの設定で行う



23

図 3-311

図 3-312

図 3-313

既にペアリングを行っていれば、そのまま使用できる。

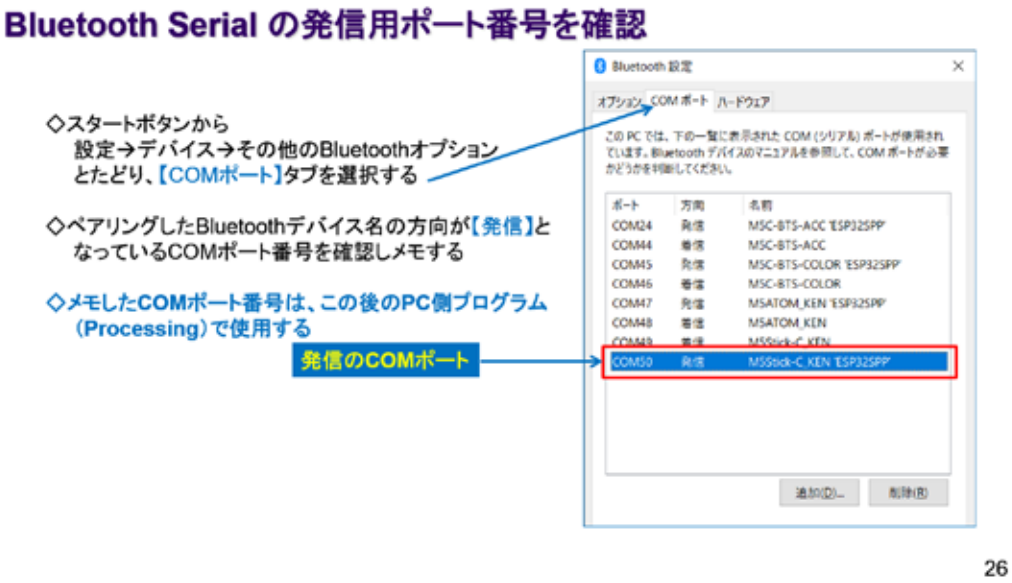


図 3-314

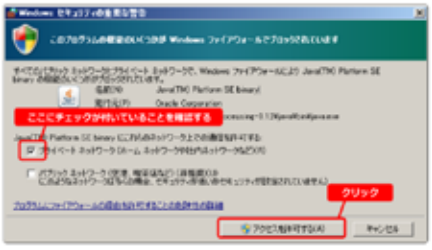
2 回目以降の Bluetooth 利用であれば、COM ポート番号に変更はないが、PC やマイコンが変わると、ペアリングと確認が必要である。

図 3-315

M5Stick-C の LCD は小さいので、この情報を PC でも確認できるようにしてみよう。そのために PC 側に必要なプログラムの環境を準備する。

Processing起動時の警告と日本語環境設定

◇左図のような警告ウインドウが表示された場合は、チェックボックスを確認して、【アクセスを許可する】をクリックする
◇うまく起動したら、IDEメニューの ファイル → 設定 とたどり、右図の言語を日本語に設定し、スケッチブックの場所、フォントを確認する
※言語を変更した場合は再起動する



28

図 3-316

これまでマイコンのシステムはC++で開発してきたが、Processingは Javaによる開発になる。言語が2つ出てきたわけだが、以下に示すソースコードを見れば内容はすぐに理解できる。

ソースコード 1/2 （ P_M5C_Color_TSC3472_2 ）

◇冒頭から初期化部（Processing）

```
import processing.serial.*;                    // Serial通信を利用するためのライブラリ

Serial myPort;                                 // シリアルポートオブジェクト
String str_get_data = null;                    // 色データ文字列受信用変数
String buf[];                                  // 色データを保管する文字列変数

void setup()                                   // 初期化部
{
  size(500, 500);                              // 画面に矩形ウインドウを描く
  myPort = new Serial(this, "COM50", 115200);  // Bluetooth接続で使用するCOMポート番号を指定
                                               // COMポートを速度115200で開く
}
```

29

図 3-317

ソースコード 2/2 （ P_M5C_Color_TSC3472_2 ）

◇通常処理部（Processing）

```
void draw()        // 通常処理部
{
  str_get_data = myPort.readStringUntil(10);          // シリアルポートで文字コード10が来るまで受信
                                                      // 10 = 0x0A = ラインフィードコード
  if (str_get_data != null){                          // 順データがNullでなければ処理する
    str_get_data = trim(str_get_data);                // 改行コード取り除き
    buf = split(str_get_data,",");                    // (,)カンマで分離してbuf[ ]に保管

    //色を設定
    fill(int(buf[1])*2, int(buf[2])*3, int(buf[3])*2);   // 保管したRGBデータを整数に変換して、
                                                         // 矩形塗りつぶし色の指定
    rect(100 ,100, 300, 300);                         // ウインドウ内に指定色の矩形を描画する
    println(int(buf[1]),int(buf[2]),int(buf[3]));     // 念のため受け取ったデータを表示する
  }
}
```

念のため受け取ったデータを表示する

30

図 3-318

動作確認 1/2

◇M5Stick-Cはプログラムが書き込まれて、動作スタートしている → ペアリングできれば、動作している
◇Processingのソースコードが準備できたら、IDEの左上にある(▸)をクリックして実行する(左図)
◇しばらくすると右図のウインドウが表示される

31

図 3-319

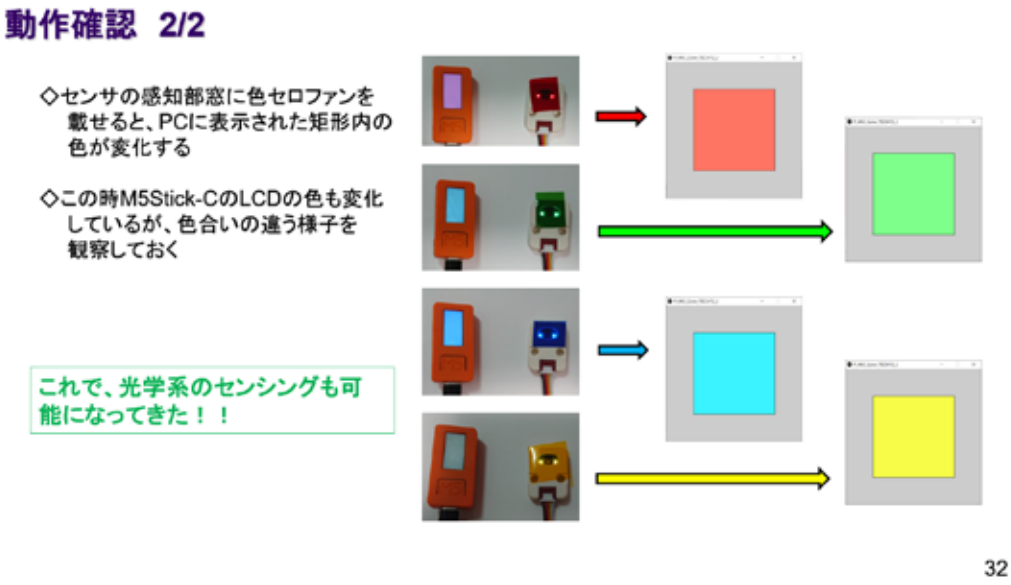

動作確認 2/2
◇センサの感知部窓に色セロファンを載せると、PCに表示された矩形内の色が変化する
◇この時M5Stick-CのLCDの色も変化しているが、色合いの違う様子を観察しておく
これで、光学系のセンシングも可能になってきた！！
32

図 3-320

３．１５　M5Stick-C　ー　IoT クラウドモデル

図 3-321

センサシステムが開発できるようになったので、検出情報をクラウド側で受け取るモデルを開発してみよう。

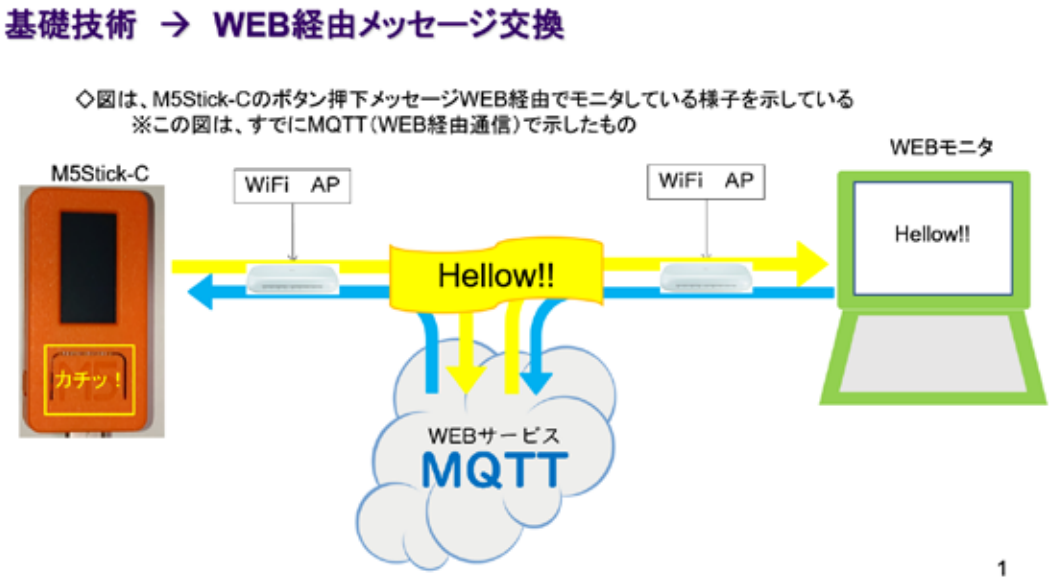


図 3-322

先行して実験した MQTT を使う。

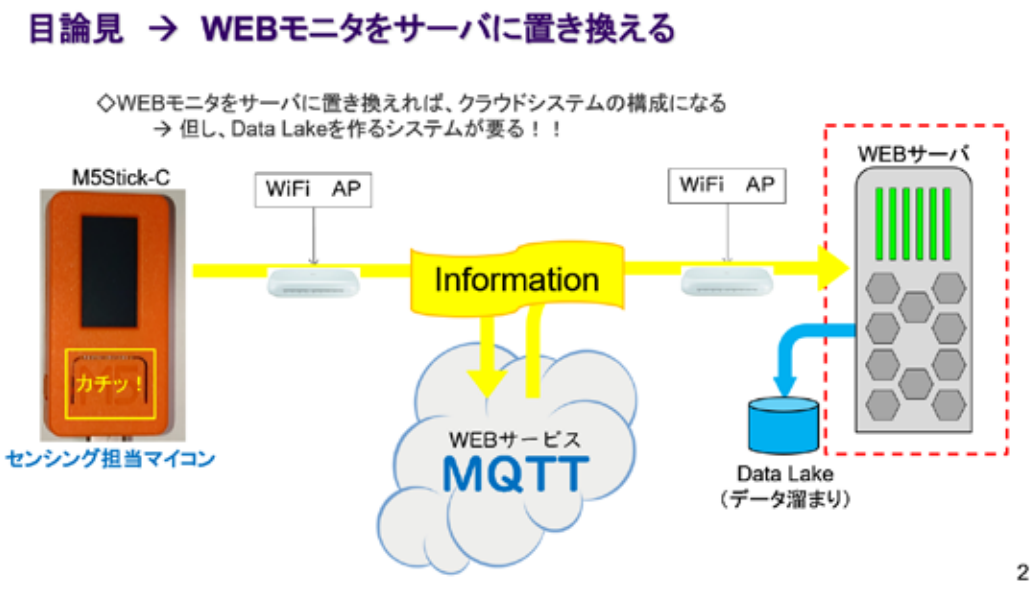


図 3-323

WEB ブラウザで確認していた MQTT のメッセージを、直に受信してファイルに保存することができれば、この PC がクラウドシステムになっていると、理解できる。データ溜まりが Data Lake に相当する。

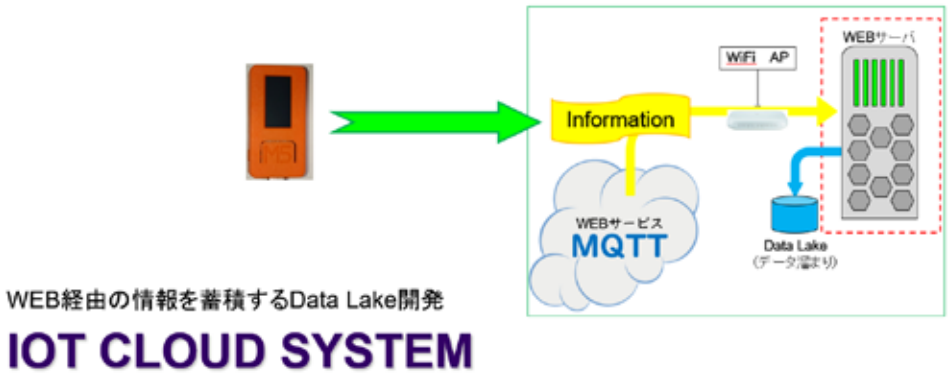


図 3-324

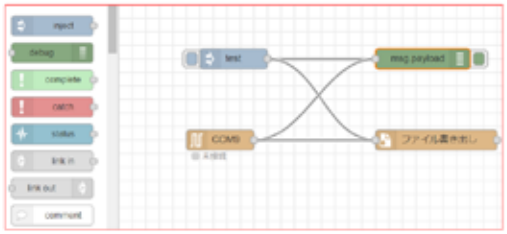
システム構想

◇IoTクラウドシステムには、WEB上でクライアントから送られた情報を蓄積する役割が必要
◇ここで、WEBから取得するメッセージを蓄積するシステムを開発する
→ 蓄積するメッセージは、MQTTで受信した生のメッセージを蓄積する
→ Data Lakeと言われる
→ 釣った魚を活かしておく【生簀】のようなもの
（魚＝Data、 後の調理＝データ加工 と考える）

◇ソフトウエアの開発には、Node-REDを用いる ← IoTのための【ビジュアルツール】とも呼ばれる
→ 複雑なプログラミングが不要、エッジマイコンでも稼働する
→ 習得に時間がかからない などメリットあり
◇開発手順は

①. フローエディタ（右図）に、カプセルを配置
②. カプセルのプロパティを設定
③. カプセル間を接続する

※このカプセルのことを【ノード】と呼ぶ



4

図 3-325

ここで用いる Node-RED は、これまでとは異なる系統のシステムになるので、専用の環境を構築する。

Node-RED環境構築

◇Node-REDを稼働させるために、2つのシステムをインストールする
①. Node.js ← WEBからダウンロードしてインストールする → https://nodejs.org/ja/
②. Node-RED ← Node.jsの後に、コマンドプロンプトからインストールする

5

図 3-326

Nopde-RED には、2 つのプログラムをインストールする必要がある。以下の順番でインストールする。

図 3-327

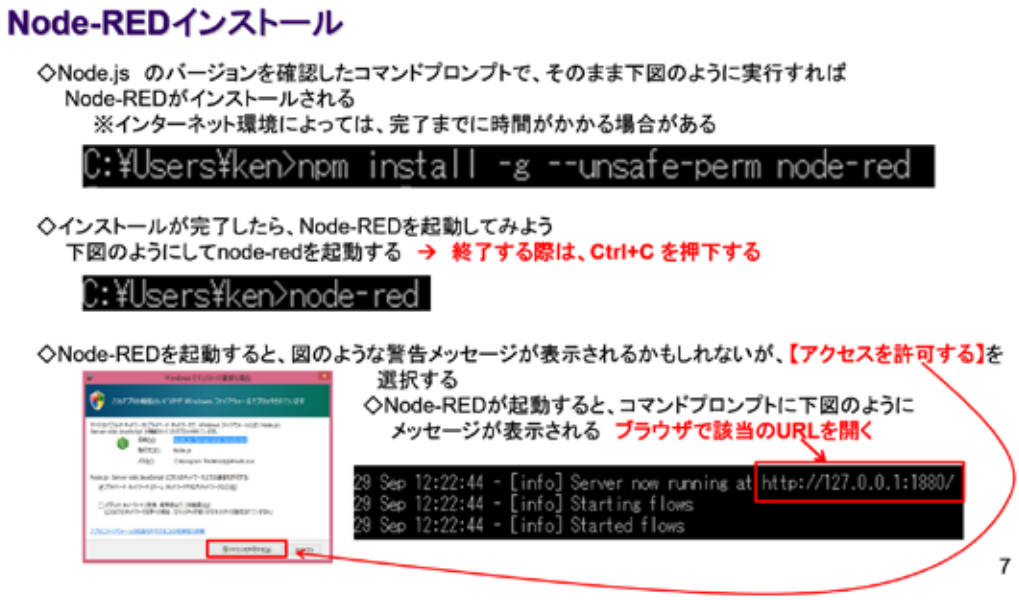


図 3-328

Node-RED の起動は、コマンドプロンプトで node-red【ENTER】と入力すればよい。終了は、Ctrl+C を押下する。

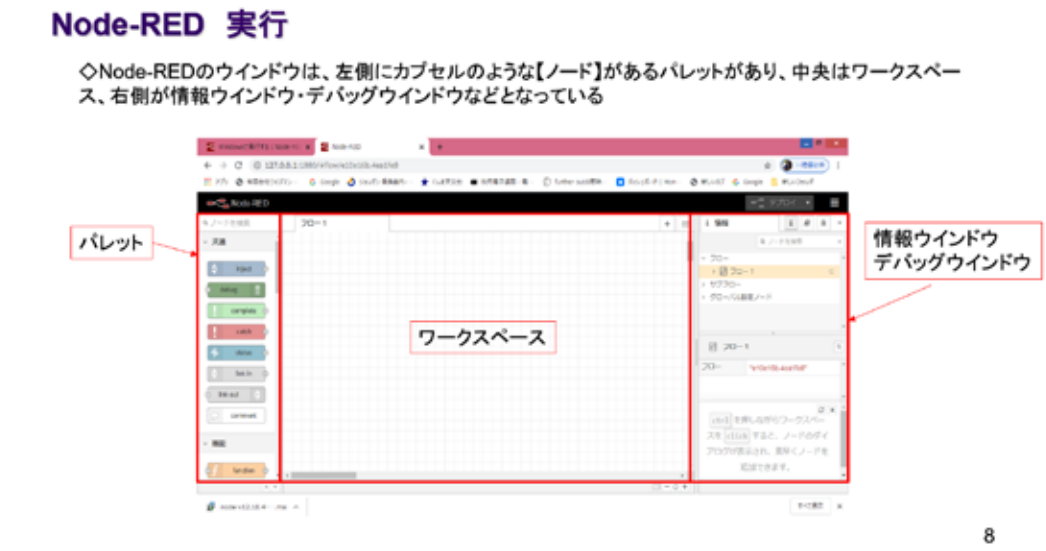


図 3-329

ブラウザを開き、URL に http://127.0.0.1:1880/ と入力してページを開くと、上図の画面が表示される。パレットに並ぶノードと呼ばれるカプセルのようなものを中央のワークスペース部分に配置する。ノードのプロパティを設定し、ノード間を接続すれば、フローと呼ばれる Node-RED のプログラムが開発できる。これを用いて、MQTT 経由のメッセージを購読して、ファイルに保存するシステムを開発する。

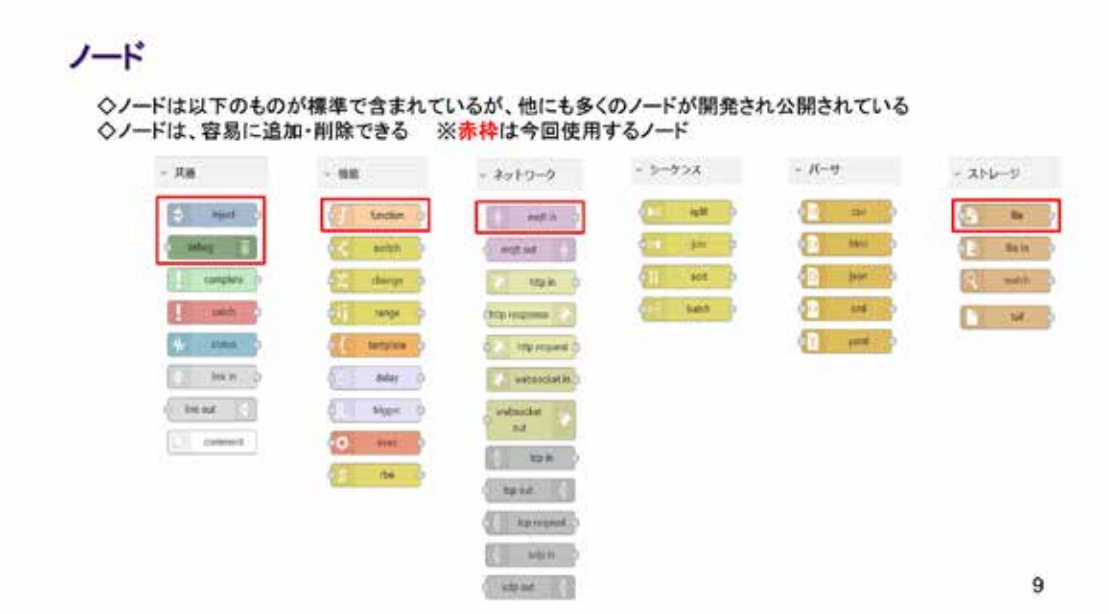


図 3-330

インストール直後にそのまま使用できるノードを図に示す。赤枠で囲んだノードは、今回使用するものである。上図以外に多くのノードが開発され、公開されている。それらの利用については、別の機会に説明する。

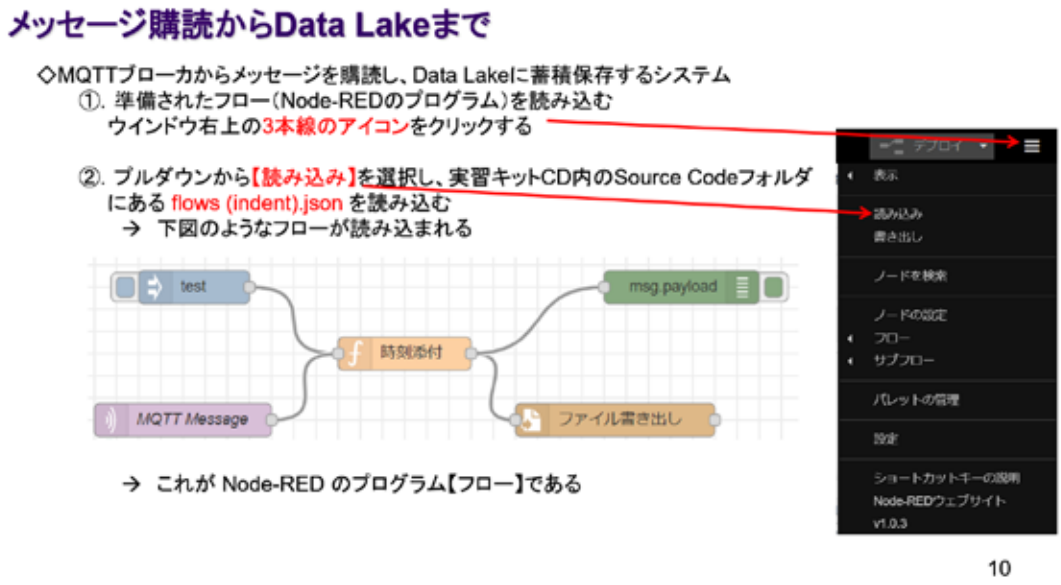


図 3-331

上図に示す手順で、実習キットに含まれているフロー(flows(indent).json)を読み込む。このフローが、メッセージを購読して記録するクラウドシステムである。

図 3-332

読み込んだフローのノードを、稼働環境に合わせて、適切に設定する。file ノードは、Data Lake にメッセージの内容を記録する機能を持つ。このシステムでは、テキストファイルにメッセージを追記するので、そのファイルのフルパスを設定する。ディレクトリを分ける場合は、それを作成するかどうかも設定できる。

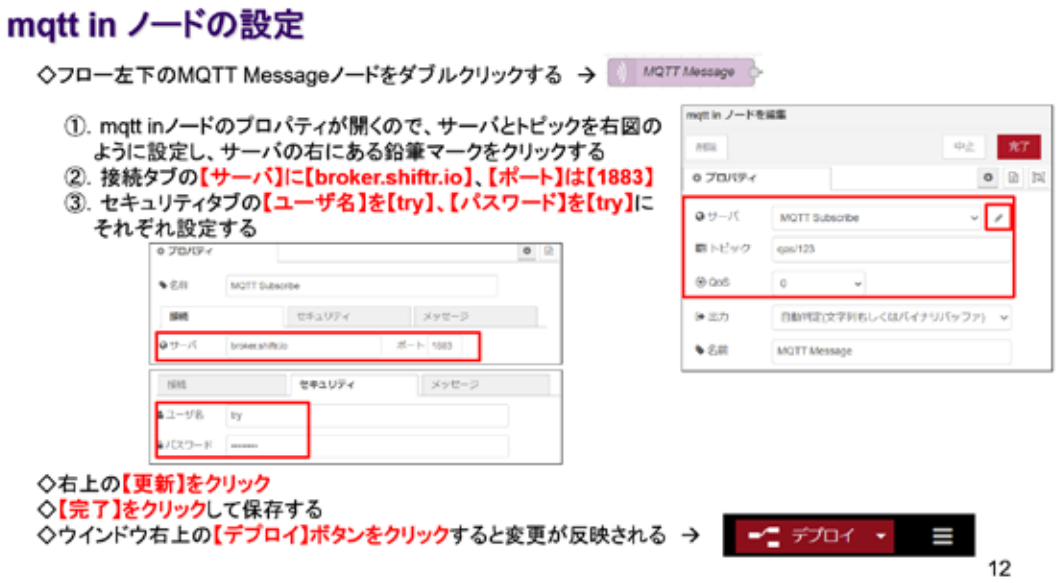


図 3-333

mqtt in ノードは、MQTT Broker からメッセージを購読するので、Broker URL、ユーザ名、パスワード、トピックなどを設定する。ノードの設定変更や、ノードの追加などを行うと、デプロイボタンが有効になるので、これをクリックする。エラーメッセージが表示されなければ、フローは正常に稼働している。

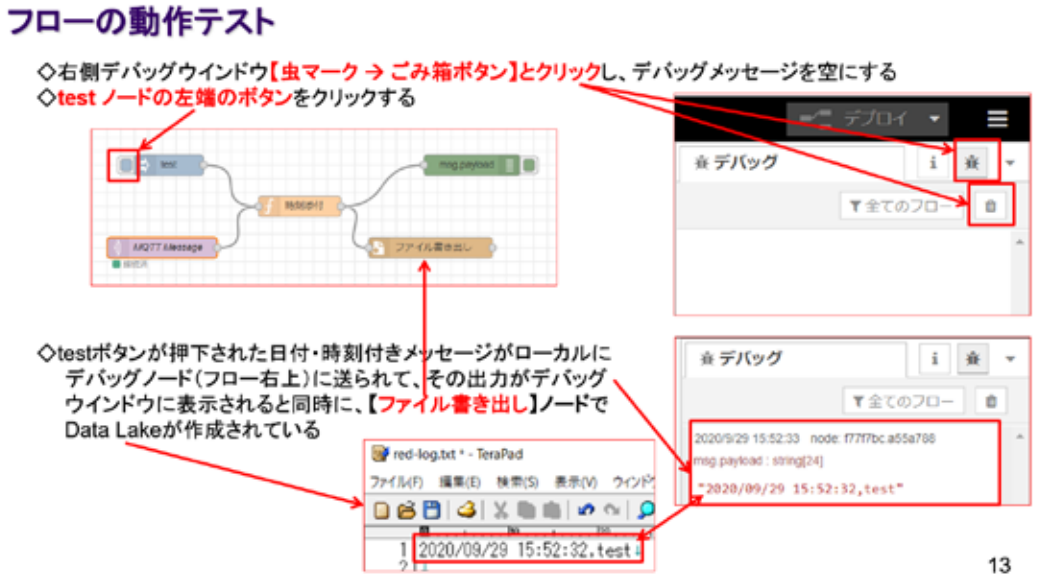


図 3-334

フローの動作を上図の手順で行う。初めに test ノードの左側ボタンをクリックして、デバッグメッセージを確認する。

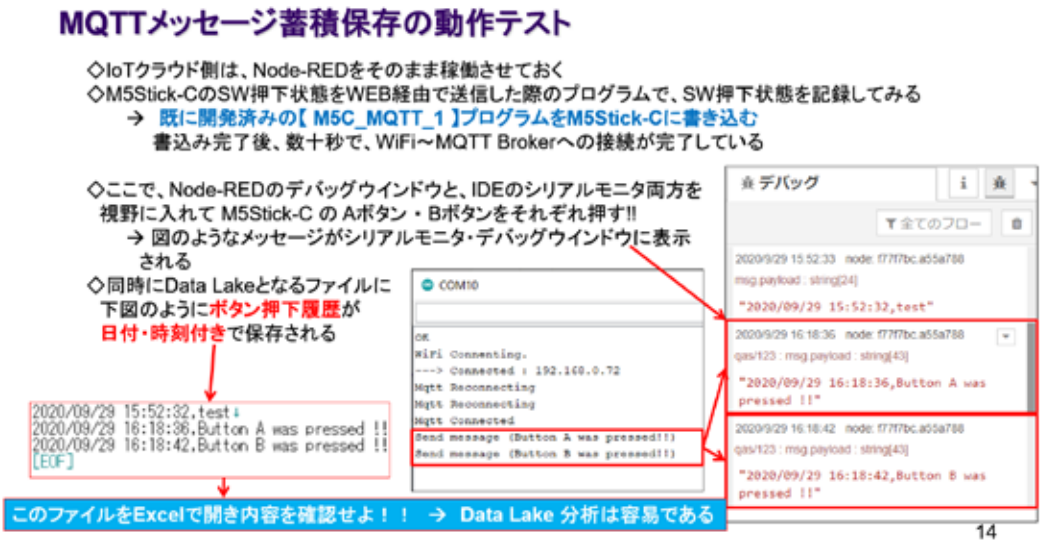


図 3-335

test ノードによるデバッグメッセージが確認出来たら、M5Stick-C で既に動作確認した、図の青字で示すプログラムをマイコンに書き込む。書き込みが完了して数十秒の間には、MQTT Broker への接続が完了している。デバッグウインドウのメッセージに注意しながら、M5Stick-C の A ボタン、B ボタンを押してみる。該当のメッセージが表示されれば、Node-RED で MQTT メッセージが購読されている。

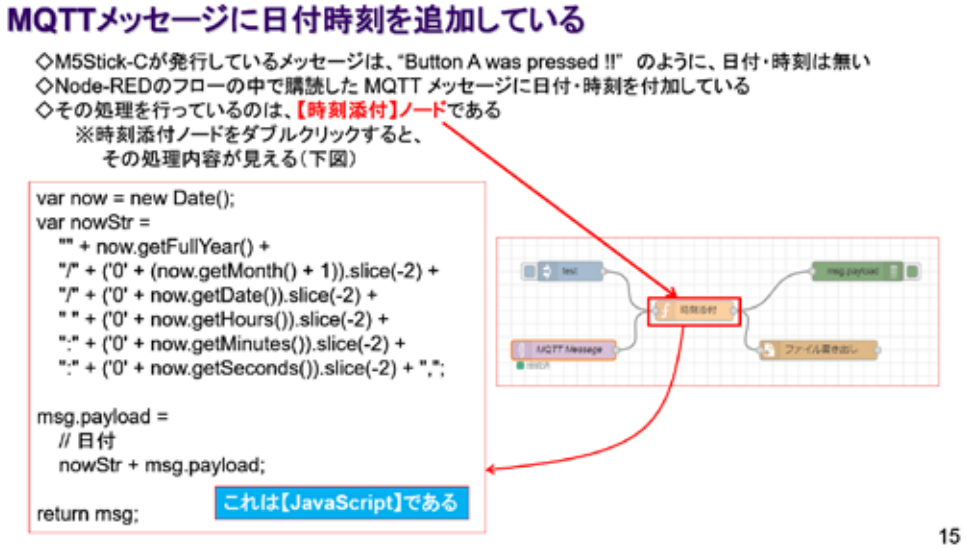


図 3-336

M5Stick-C には、日付時刻が分からないので、Node-RED のフロー側で受信した日付・時刻を添付している。その部分は、簡単なスクリプトで記述している。該当のノードは中脳の【時刻添付】ノードである。ダブルクリックすると、内容が確認できる。

３．１６　データ取り出しと分析

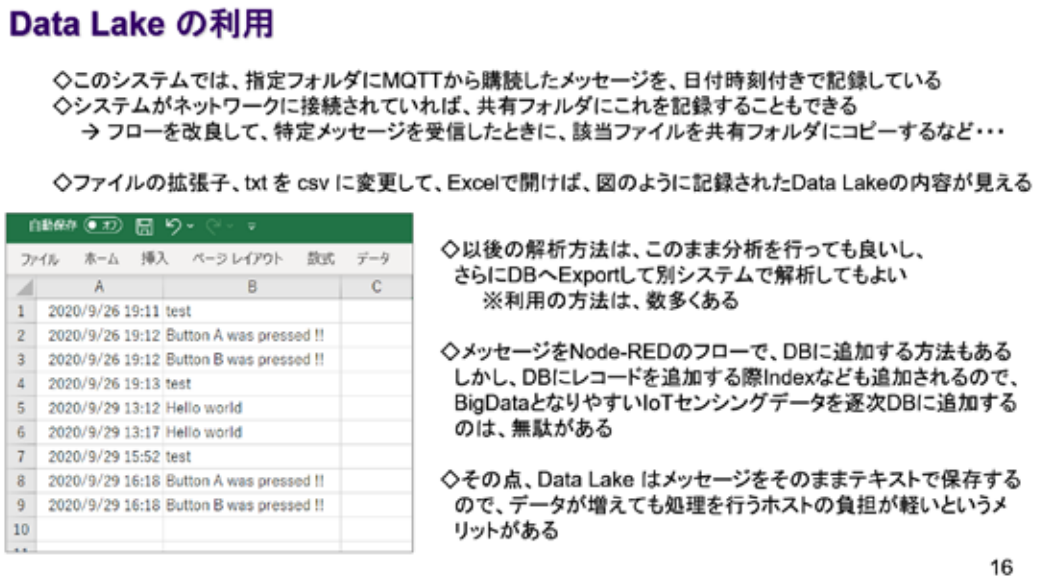


図 3-337

これまで開発したシステムでは、IoT デバイスであるマイコンから通知されたメッセージをインターネット経由で受け取り、Data Lake に保存した。このデータを必要に応じて取り出し、分析を行って将来の工場運営に活かすことが、モノつくり IoT の最大の目的である。ここで記録しているデータは、テキストファイルであるから、簡単に取り出せる。Node-RED のシステムが稼働するのは、ネットワークに接続されているコンピュータなので、共有が許されている PC からは容易にファイルの取り出しができる。ローカルに取り出したファイルは、IoT とクラウドの連携に束縛されず、自由に分析・解析を行うことができる。この講座では、解析手法につては触れない。取り出したデータを DB に登録することも容易である。Python を用いてテキストファイルをそのまま読み込み、AI システムに送って解析することも容易である。

まとめ

◇マイコン発生のメッセージをWEB経由（MQTT）でクラウドホストに伝達することができた
◇クラウドホストまでの間では、マイコンの無線通信の一手法として Bluetooth が利用できる
◇クラウドホストは、フロープログラミングで構築できる
◇Data Lake が単純な構造であることが分かった

※ Big Data となり得る IoTセンサ情報を、常時 Data Base に追加保存するのは無駄である
→ 実験したように、Data Lake に泳がせておいて、バッチなどで、必要なデータを抽出して Data Base に格納し、その後の作業を軽く行うのが理想的

◇MQTT メッセージは、マイコン側で購読も可能（M5ATOMで実験した）なので、M5Stick-Cで
メッセージの簡易モニタも開発できる
※ 搭載のカラーLCDは小さいが、小さくてもそこに表示ができれば、後で外付けの大型表示器を用いればよい

◇Big Dataとなりやすいの IoTのデータは、直接DBに追加せず、Data Lakeで活かしておき、必要な時に必要なデータだけをDBに取り出して処理するのが良い

17

図 3-338

第4章 IoT プラットフォーム

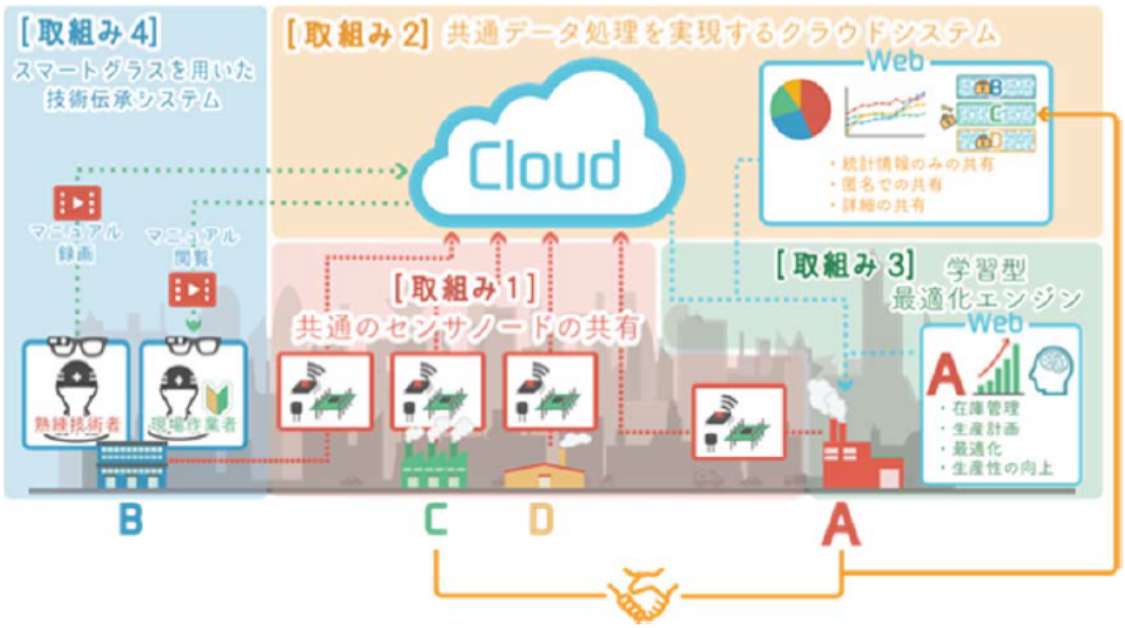


図 4-1

『IoT プラットフォーム』は、共有型モノづくり IoT プラットフォームとして、富山県立大学で開発された。図 4-1 にシステム開発時の提案概要を示す。このテキストでは、提案概要の取組 1 で開発されたセンサノードを使用する。『共有型とやまモノづくり IoT プラットフォーム』は『取組み 1』から『取組み 4』でシステムを開発して、複数企業で簡易的な IoT システムを共同利用する仕組みづくりを行い、中小企業で IoT システム導入促進を図り、生産性の拡大を目指しているものである。

第5章 IoT センサシステムの構成と必用機材

この章以後で、IoT センサシステムの設置から撤去までの作業を説明する。2019 年度、千葉県の樹脂加工を行う企業の協力を得て、このセンサユニットをテスト稼働した際の記録に基づき解説する。その際に IoT システムを設置した設備は樹脂成型機である。テキストでは、この設備を対象にして解説を進める。まず使用する IoT センサシステムとその他の機材を説明する。

５．１　IoT センサシステム

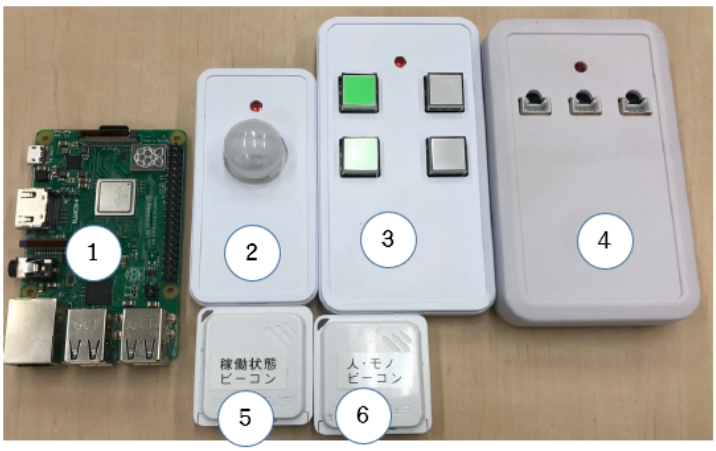


図 5-1

図 5-1 にすべてのセンサユニット、及びセンサノードとして稼動する IoT マイコンを示す。2019 年度に富山県立大学岩本研究室よりお借りした機材を下記に示している。

①．IoT マイコン（Rspberry Pi）
②．人感センサユニット（赤外線センサ）
③．スイッチユニット（ユニットあたり 4 個）
④．光センサユニット（ユニットあたり 3 個：三色タワーに対応）
⑤．稼働状態センサユニット（加速度センサ）

※⑥の人・モノビーコンは、使用していない（借用物には含まれていない）。

図 5-2

図 5-2 は、ケースに入れて電源アダプタを接続した IoT マイコンである。実際に使用する際はケースに蓋をして、WiFi アクセスポイント付近に配置する。IoT マイコンを置く場所は、各センサユニットとの距離を考慮する必要がある。センサユニットと IoT マイコン間で BLE（Bluetooth Low Energy）を用いた通信が行われるからだ。このマイコンは、稼動中に発熱を伴うので、通気の良い場所に置くことが望ましい。電源は 5V で電流が 3A 以上のものを使用している（借用物に含まれている）。

５．２　その他の機材

５．２．１　WiFi アクセスポイント

図 5-3

図 5-4

図 5-3、5-4 に使用した WiFi アクセスポイント（BUFFALO WSR-2533DHPL）と電源アダプタを示す。WiFi アクセスポイントは、社内 LAN に LAN ケーブルで接続し、IoT マイコンと無線 LAN で通信を行う。

５．２．２　電源ケーブルと LAN ケーブル

図 5-5

図 5-5 に、使用した電源ケーブル、LAN ケーブルを示す。電源ケーブルは、IoT マイコンと WiFi アクセスポイントそれぞれの電源アダプタ用に使用した。LAN ケーブルは WiFi アクセスポイントと社内 LAN を接続するために使用した。電源ケーブルの口数は、WiFi アクセスポイント用、IoT マイコン用と PC 用の電源も確保しておいた方が良い。

５．２．３　ノート PC

IoT マイコンの設定とセンサ閾値調整、データ確認・保存などを行うためにノート

PC（Windows10　Home）を使用した。PC はノート PC である必要はないが、現場や事務所などを移動して IoT マイコンの設定、センサに対する閾値調整やデータの確認を行う上では、可搬性のあるノート型が良い。

５．２．４　ソフトウエア（ノート PC 内）

IoT マイコン内のデータベースに保存された稼動状況のデータを取り出す際に、フリーのターミナルソフトウエア『Teara Term』を使用した。このソフトウエアの Log 機能を使えば、IoT マイコンに Tera Term で接続して操作する記録が取れる。IoT マイコン内で稼動するデータベース（MySQL）内のデータも取得することができる。

※このソフトウエアは WEB 上で『Tera Term』で検索すれば、容易にダウンロードサイトが見つかる。あらかじめダウンロードして、PC にインストールしておく。Appendix：C でダウンロード URL とインストールの方法を説明した。他のターミナルソフトウエアでも Log 機能のあるものであれば、『Tera Term』に代えて使用することができる。他に ssh コマンドを用いる。これは「Windows 10 April 2018 Update」で正式版となっており、デフォルトで OpenSSH クライアントが利用できる。そのため April 2018 Update 後の Windows 10 であれば、SSH クライアントのインストール作業は不要である。未対応の OS の場合は、OpenSSH クライアントを事前にインストールしておく必要がある（Tera Term で代用も可能。11 章参照)。また、Excel がインストールされていれば、稼動データの簡易解析に役立つ。

５．２．５　ディスプレイ

IoT マイコンを WiFi アクセスポイントに接続するための設定を行う際に使用する。

５．２．６　HDMI ケーブル

上記ディスプレイを接続するために使用する。

５．２．７　USB キーボード

IoT マイコンを WiFi アクセスポイントに接続するための設定を行う際に使用する。

5．3　全体構成

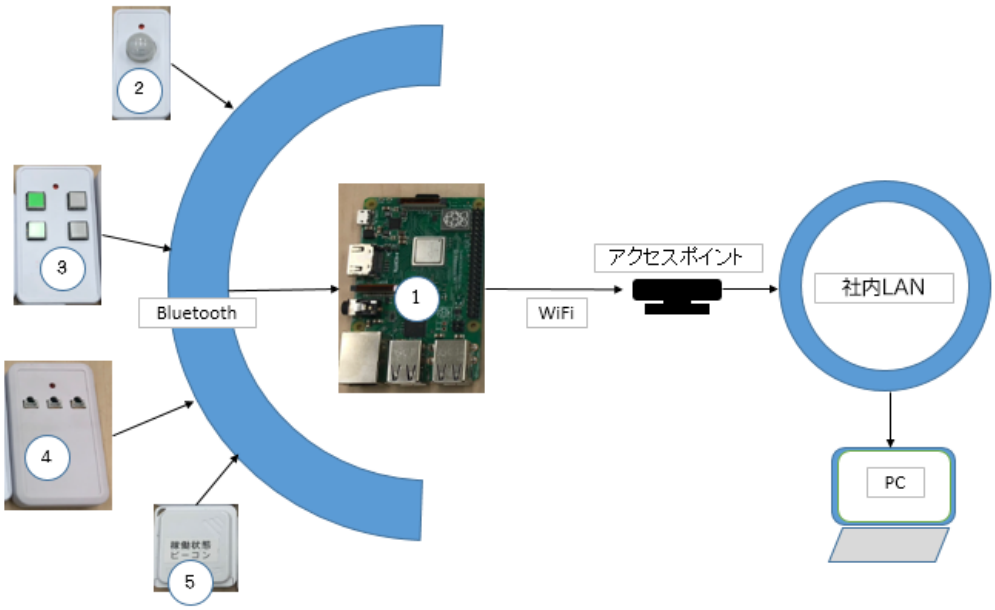


図 5-6

図 5-6 に全体の構成を示す。各センサユニットは設備及びその周辺に取り付けられ、稼動状況の元になるデータを取得し、Bluetooth（BLE）により、IoT マイコンに送信する。IoT マイコンは、データを稼働状態に変換しローカルデータベースに保存する。クラウドへの接続とデータ記録の準備を行えば、共通プラットフォーム上での実績表示、分析などが行える。今回は、クラウド環境は使用せず、ローカルデータベースに記録された稼動状況のデータを保存する所までを解説している。IoT マイコンは、WiFi によりアクセスポイントを通じて社内 LAN に接続するので、PC によって、IoT マイコンやセンサに対する調整、およびローカルデータベース内のデータを PC 側に取り出すことができる。それぞれの役割を下記する。

①．IoT マイコン（Rspberry Pi）

各センサが検出した情報を Bluetooth 経由で受信し、そのデータを元に、稼働状況を IoT マイコンのデータベースに記録する。同時に Cloud にも同じデータを送信する。（※但し、この解説では Cloud への送信は対象としていない。）センサ毎にあらかじめ設定した閾値によって、稼働状態を判断する。

②．人感センサユニット

制御盤付近に取り付ける。操作のために作業者が制御盤に近づいたことを検出する。このセンサは、稼動状態の変化に対して人的な介在の有無を記録するためのものである。

③. スイッチユニット

スイッチを稼動・正常停止・エラーによる停止に分類して、作業者が押下した内容を記録する。

④. 光センサユニット

制御盤前面に配置されているパイロットランプの点灯状態により、稼動中か停止中かを判断して記録する。光センサは予め閾値の設定が必要である。

※光センサユニットは三色灯に対応するセンサであるが、今回対象とした設備（真空成型機）では、制御盤面に設置されているパイロットランプ（電源、送り）を用いた。

⑤. 稼働状態センサユニット

内部に加速度センサを持ち、その検出値を IoT マイコンに送信する。主軸に近い付近に設置すれば、主電源が入っているかどうかが判断できる。今回は、成型機の搬送用メインモータのケースに取り付けた。このセンサには予め閾値設定が必用である。

第6章 環境確認

IoT センサシステムを使用する環境で、あらかじめ確認しておく点を説明する。

６．１　電源の確認

IoT センサシステムを使用するためには、下記の装置に電源が必要である。それらの電源が付近のコンセントから取れるかどうかを確認する。

①．IoT マイコン（Raspberry Pi 用電源アダプタ）

②．WiFi アクセスポイント（付属電源アダプタ）

（既存の WiFi アクセスポイントが利用できる時は不要。）

※センサユニットは電池駆動なので、電源は不要。

必要に応じて図 5-5 左に示す電源ケーブルを使用する。

６．２　ネットワーク環境の確認

このシステムでは、ネットワークを次の目的で使用する。

①．IoT マイコン本体の設定

②．センサ閾値の設定

③．データ取得状況の確認

④．データ取出し

⑤．データの整理

上記用途では、IoT マイコンと PC が同一ネットワーク上に存在する必用がある。利用しているネットワークが DHCP（Dynamic Host Configuration Protocol：機器に自動的に IP アドレスが割り当てられる環境）で稼動しているかどうかを確認する。装置毎に個別の IP アドレスを手動設定している環境では、ネットワーク管理者（または IP アドレスを設定している担当者）にヒアリングして、IoT マイコンと PC の IP アドレスを決める。また、使用する PC が既にネットワークに接続して利用しているのであれば、そのままで構わないが、別途 PC を準備する場合は PC 用の IP アドレスを決める。DHCP で稼動した方が、設定は容易である。

６．３　利用できるアクセスポイント

既に稼働している社内 LAN の WiFi アクセスポイントがあり、それを利用したい場合は、アクセスポイントを管理している担当者に IoT マイコンと PC を接続できるか、問い

合わせる必用がある。IoT マイコン付近に利用できる WiFi アクセスポイントが無い場合、新規に WiFi アクセスポイントを準備する。LAN ケーブルによるネットワーク接続は、想定していない。新規の WiFi アクセスポイントを準備する場合は、WiFi アクセスポイントを DHCP による IP アドレス設定か、または手動による固定 IP アドレスに設定する。これにより、IoT マイコン・WiFi アクセスポイント・PC が同じネットワーク内に存在して通信が行えることになり、センサ閾値の設定や、取得データの観察などを行う。

６．４　対象設備の情報をどこから取り出すか

設備の稼働率を知るための元の事象を、設備のどの部分からとらえるのが良いか事前検討が必要である。制御盤および設備付近にセンサユニットの養生テープによる固定が可能であれば、パイロットランプやタワー型ランプなどは便利な情報源である。また作業者がスイッチを操作して事象の入力ができれば、これも有力な情報源になる。振動する主軸等は加速度センサで設備稼働を捉えることができる。金属加工設備などは高精度が要求されるため、全体の振動が非常に小さいものである。主軸の振動を検出するよりも、主電源のパイロットランプを検出する方が容易な場合もある。現在、稼動センサ、スイッチ、人感センサ、光センサなどが提供されているが、稼動・非可動を記録できれば、すべてのセンサを使用しなくてもよい。対象とニーズに応じた選択が可能で、ユーザの要求に柔軟に対応できるユニットが揃っている。

センサユニットを取り付ける場所が高温になる場合や、粉塵・高湿度などの状況下では、正常な動作が保証されないので、環境について不安要素がある場合は、事前に開発元への確認を行うのが良い。

６．５　使用する PC の OS バージョンと、ソフトウエアの確認

OS（WIndows10）のバージョンについて確認をしておく（Windows 10 April 2018 Update 以後が望ましい)。OS が古いものであれば 3.2.4 ソフトウエア（ノート PC 内）で説明したように、ssh クライアント（またはこれに代わるソフトウエア）のインストールが必要になる。上記 OS が不具合なく稼動している PC であれば、能力やメモリ、ディスクの容量などが問題になることはない。

第7章 IoT センサユニットの設置手順

センサユニットにはコイン電池が内蔵されていて、常時通電状態である。対象設備への取り付けさえ行えば、各センサが状況を取得して IoT マイコンに送信を行う。センサユニットを固定するために再剥離可能な養生テープが使用できる。本テキストで解説している、樹脂シート成型機は、成型後のシートを方から外しやすくするために離型剤が使用されていて、養生テープの粘着力が効かない部分があったため、長いテープを利用した部分が多い。各ユニットの取付け順は特にない。恒久的に設置する場合は、マグネットシートなどをセンサユニットのケースに張り付け、その性質で設備周辺に取り付けることも可能である。(金属加工を行う工場設備の場合は、磁力により金属粉が吸着されることもあるので、定期的な清掃が必要になる。

7．1　稼動センサユニット

成型機の中央下部にある、樹脂シート搬送用メインモータのケース部分に固定した。1 辺が 4cm、厚さ 1cm ほどの小さなユニットは、固定するテープの幅に充分収まる。

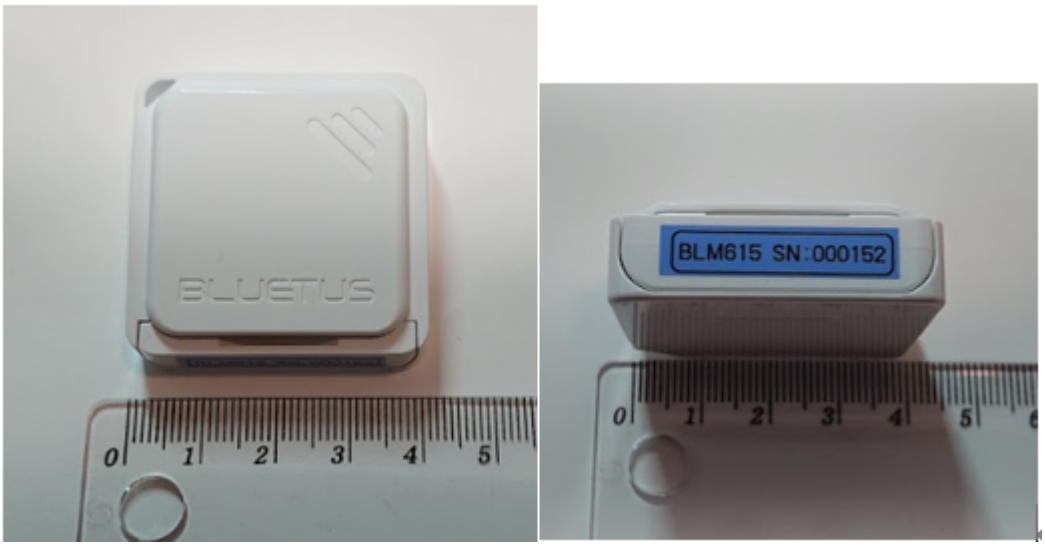


図 7-1

図 7-1 は稼動センサである。右図の SN で示される番号はセンサユニット ID となっている。このセンサユニット ID は、後に IoT マイコンの設定で必要になるのでメモしておく。

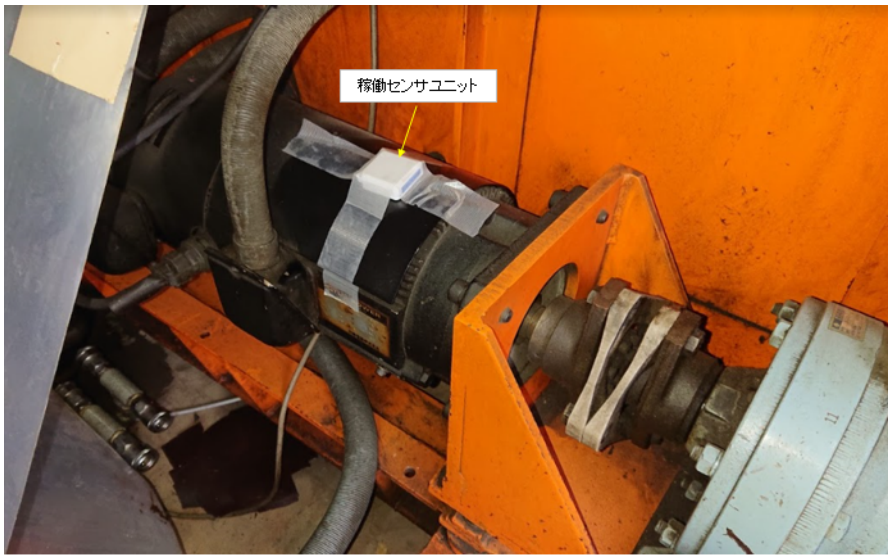


図 7-2

図 7-2 に、稼動センサユニットを取り付けたモータのケース部分の様子を示す。稼動中は、このモータが回転しており、常時小刻みに振動している。一定時間ごとにモータ右側のクラッチが繋がり、成型機の中の樹脂シートを搬送するチェーンを駆動する。このとき比較的大きな振動が発生する。IoT マイコンでは、電源投入と同時に発生する小さな振動で、稼動状態と判断するように閾値を設定する。この設備では、シート搬送方向に対して垂直方向に真空成型の型を動かすので、成型の前後にも比較的大きな振動が発生する。稼動センサでは、稼動中か否かを判断するので、小刻みな振動を検出するように閾値の値を決めることが必要である。この工場は、工業団地の中に在り、隣接している工場には建設資材などを製造しているプラントなどがある。隣接工場での設備稼働時や、資材搬入・搬出の際の大型車両の通過などでも振動が発生している。対象設備だけでなく、周辺の環境によっては、閾値の決定が難しい場合もあるが、時間をかけてじっくりと決める。

7．2　スイッチ

スイッチは、作業者が制御盤を操作する際、その要因を簡単に分類して記録することが目的である。3 つの要因 ①稼動、②通常の停止（ロット数完了、材料シート掛け代えなど)、③異常による停止、の区別を記録するために制御盤に取り付けた。今回は直接養生テープで固定したが、透明のシートなどを被せたうえで固定する等の方法もあり、固定後作業者がスイッチを操作することができれば良い。

図 7-3

図 7-3 はスイッチユニットである。左図ではスイッチの番号を示している。①②は共通で稼動開始、③は通常停止、④は異常による停止を記録する。右図の SN で示される番号はセンサユニット ID となっている。このセンサユニット ID は、後に IoT マイコンの設定で必要になるのでメモしておく。

図 7-4

図 7-4 にスイッチユニットを固定した様子を示す。長めのテープで固定しているのは、制御盤面にも離型剤の粒子が付着しているからか、長時間経過すると養生テープの剥離が見られたからである。左右のケーブルは、光センサのものである。

7．3　光センサユニット

光センサは、ユニット本体とセンサ部（3 個）に分かれている。

図 7-5

図 7-5 は光センサユニットである。左図ではコネクタに接続した光センサの番号を示している。この番号は、閾値設定の際、個々の光センサ ID となる。三色タワー用に 3 つのセンサがあるが、今回は①を『電源』（搬送用モータが回転する）②を『送り』（実可動）のパイロットランプに割り当てた。③は未使用として、ケーブルをまとめて、制御盤面に固定した。使用しない光センサは、ユニットにケーブルを接続せずに使用してもよいが、その際はコネクタ部に異物などが入らないように表面を養生しておく必要がある。右図の SN で示される番号はセンサユニット ID となっている。このセンサユニット ID は、後に IoT マイコンの設定で必要になるのでメモしておく。

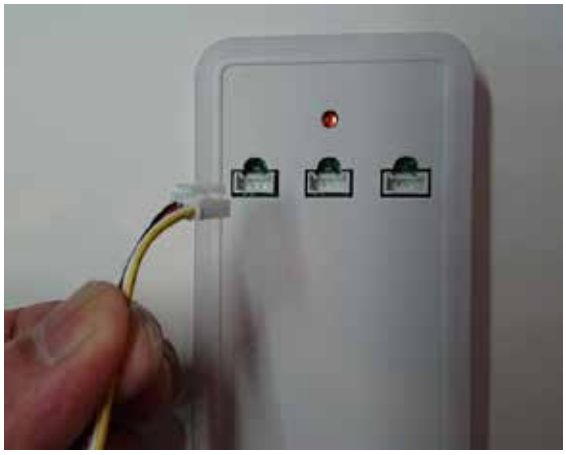


図 7-6

図 7-6 は、光センサケーブルの取付けの様子である。コネクタの上下を間違えぬように光センサケーブルをユニットのコネクタにパチッと音がするまで挿し込む。

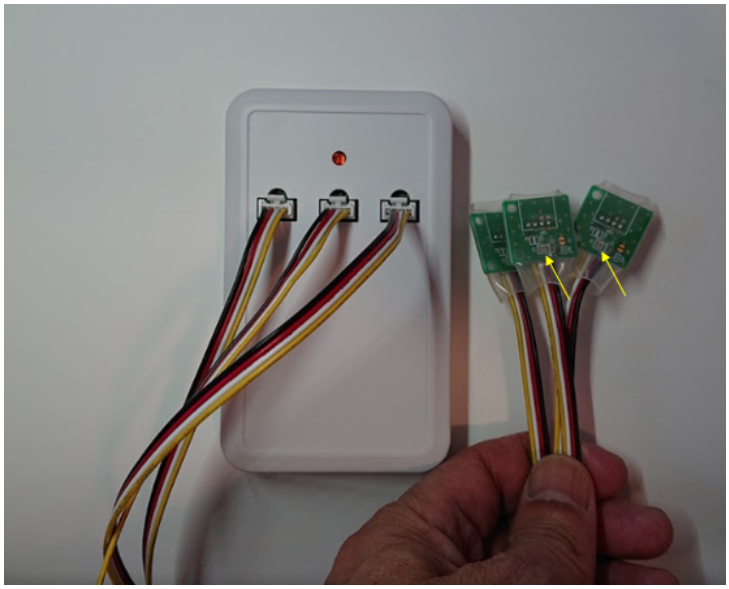


図 7-7

図 7-7 は 3 つの光センサケーブルを取り付けた様子である。ケーブルの先にある緑色の基板上に小さな光センサがある（矢印の先）。

光センサの取付け作業は、①ユニットの固定　と、②センサの固定　に分かれる。ユニットはスイッチユニットと同様の固定方法で制御盤面に取り付けた。標準ケーブルの長さが 60 ㎝弱なので、環境によっては、長いケーブルを提供していただく必要がある。

図 7-8

図 7-8 に光センサユニットと光センサを取り付けた様子を示す。未使用センサは、制御盤面にケーブルをまとめて固定した。センサケーブルは、制御盤操作中に手などが引っ掛からないように盤面に細かく固定した。(※未使用センサを取り付けずに、IoT マイコン内部の設定で処理できることが後で分かった。)

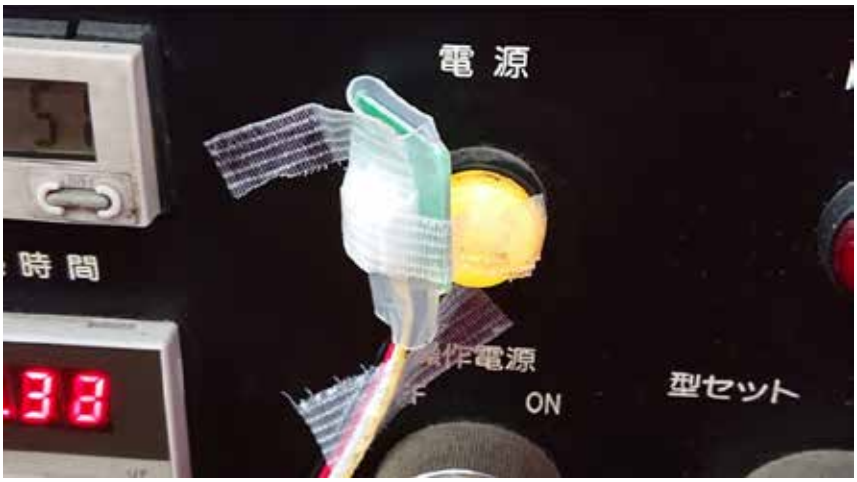


図 7-9

図 7-10

図 7-9、7-10 はパイロットランプへの光センサの取付けの様子を示している。現在作られている設備の制御盤では先端が平面形状のパイロットランプも多いが、工場には、図に示す電球式のパイロットランプも多く残っている（制御盤の横方向からの視認性を考慮した卵型のケース形状である)。今回は、試用期間も短く稼動データの取得はテスト的であるため、センサ基板を図のように養生テープで固定した。将来は、取り付け部分の形状に合わせたセンサ基板や、基板自体を小さくすることについて検討しておく必要がある。また、光センサにこだわらずパイロットランプ電源から電圧を取得して入力（DI）とするなどの検討も必要である。そうすれば、スイッチユニットが代わりに利用できる可能性もある。現状では、隙間ができるためパイロットランプの光以外もセンサに入り込む。光センサの閾値設定が重要である。

7．4　人感センサ

人感センサは、設備稼働状況の変化に人的な介在があったかどうかを記録する目的で使用する。

図 7-11

図 7-11 に人感センサを示す。白い半球体の部分が人体の発する赤外線を検出する部分になっている。図の白い球体はフレネルレンズと呼ばれる赤外線をセンサに導くためのレンズである。右図の SN で示される番号はセンサユニット ID となっている。このセンサユニット ID は、後に IoT マイコンの設定で必要になるのでメモしておく。

図 7-12

このセンサは、検出範囲が広いので制御盤の前に作業者が来たとき反応するように、設備

の付近にある工場建屋の鉄骨に取り付け、制御盤前の空間の赤外線を検出するように紙を丸めた筒状のフードを作り、検出範囲を限定した（図 7-12）。このセンサは、検出できる奥行きも大きいので、設定したい空間の向こう側を人が通過すると検出する場合がある。このような場合は、高い位置（上）から制御盤前の空間を狙うように取り付けると良いと思われるが、今回はそのような位置に設置することができなかった。

第8章 IoT センサシステムのセットアップ手順

この章では、WiFi アクセスポイント、PC、IoT マイコンに分けて手順を説明する。なお、IoT マイコンのセットアップは、『IoT システム利用マニュアル』（富山県立大学岩本研究室発行）を併読して行う必要が有る。

8．1　WiFi アクセスポイントの IP アドレス設定

WiFi アクセスポイントの IP アドレスを使用するネットワーク内のアドレスに設定する。設定の具体的手順は使用する機器により異なるので、機器付属のマニュアルによって手順を確認して設定する。なお、今回使用した WiFi アクセスポイント（BUFFALO WSR-2533DHPL）の設定の手順は、Appendix:A に記した。IP アドレスの設定を行った後、WiFi アクセスポイントを再起動する。(再起動の方法は、機器付属のマニュアルで確認すること)。再起動後にネットワーク内の PC から図 8-1 のように ping コマンドを発行して、WiFi アクセスポイントが応答することを確認しておく。

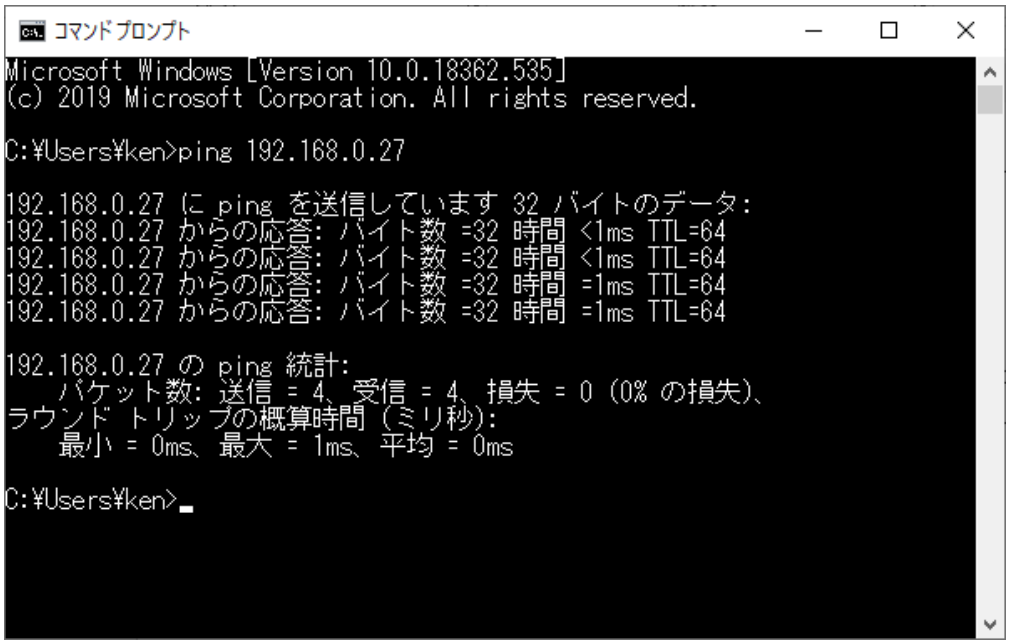


図 8-1

8．2　PC のネットワーク接続

PC を WiFi アクセスポイントに接続する。接続の手順は、スタートボタンから、設定 → ネットワークとインターネット → ネットワークと共有センター → アダプタの

設定の変更　→　Wi-Fi　とたどり、Wi-Fi の状態ウインドウのプロパティボタンで、インターネットプロトコルバージョン 4（TCP/IP）を図 8-2 に示すように設定し、OK ボタンをクリックする。

図 8-2

左図は DHCP の場合、右図は IP アドレスを指定する場合を示している。

8．3　IoT マイコン（Raspberry Pi）のネットワーク接続

IoT マイコンを WiFi アクセスポイントに接続して PC から接続確認を行う。

8．3．1　WiFi アクセスポイントに接続するための設定

まず初めに、IoT マイコンを WiFi アクセスポイントに接続しなければ何もできない。そのための設定を以下の手順で行う。

①．ディスプレイを HDMI ケーブルで IoT マイコンに接続する。
②．USB キーボードを IoT マイコンに接続する。
③．IoT マイコンに電源アダプタを接続し、電源を投入する。
④．ディスプレイに iot login: と表示されるまで待ち、ユーザ名 pi と入力し Enter キーを押す。
⑤．ディスプレイに password:と表示されるので、予め通知されたパスワードを入力して Enter キーを押すとプロンプトが pi@iot:~ $ に変わる（図 8-3）。

図 8-3

⑥. プロンプト以後に次のように入力して Enter キーを押す。

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

⑦. nano エディタが開くので、↑↓←→のキーでカーソルを移動して、図 8-4 で示すように、接続する WiFi アクセスポイントの SSID と Password を入力する。接続する WiFi アクセスポイントの SSID と Password は一つ追記すればよい。

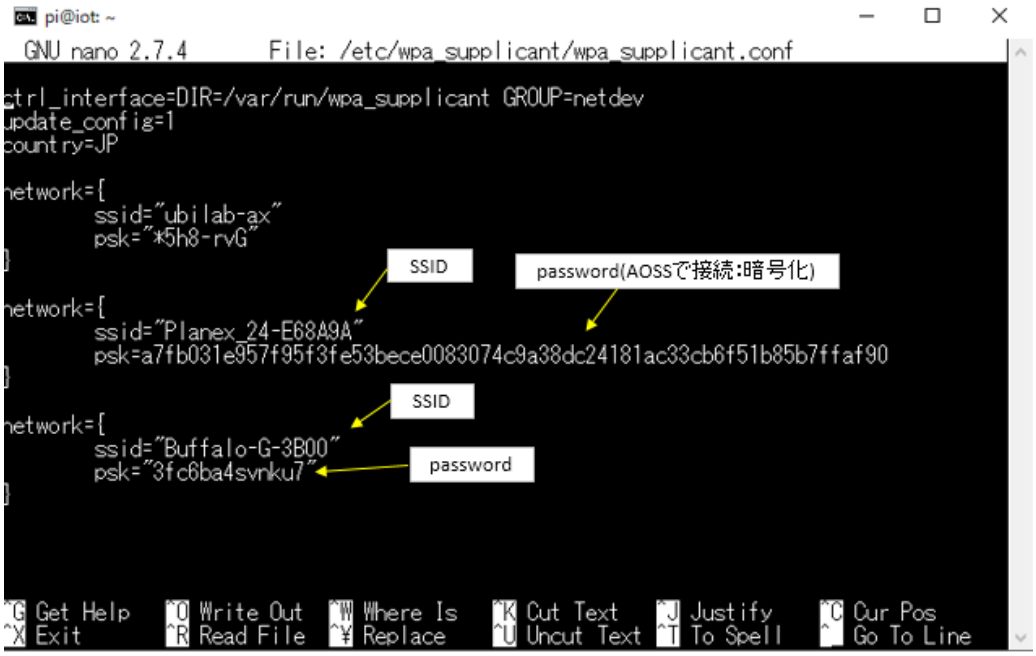


図 8-4

⑧. ^X（^は Ctrl キーを示す）を押すと書込み保存するかどうか問われる。y（Yes）キーを押下する。書込みするファイル名が表示されるので、そのまま Enter キーを押して書込み保存する。

⑨. 次に固定 IP アドレスで運用する場合の、IoT マイコンの IP アドレスを設定す

る。(※DHCP で運用する場合は⑨⑩の設定は行わない)。プロンプトに次のように入力し Enter キーを押して、設定ファイルを nano で開く。

sudo nano /etc/dhcpcd.conf

⑩. 下記の 4 行を追記して⑧のようにして保存する。
※下記の IP アドレスなどは、事前にネットワーク管理者と協議して決めておく。

interface wlan0 <--- ワイヤレス LAN の 0 番目を意味する
static ip_address=192.168.0.20/24 <--- 指定する IP アドレスとマスクビット数
static routers=192.168.0.1 <--- ゲートウェイアドレス
static domain_name_servers=192.168.0.1 <--- DNS アドレス

⑪. 図 8-5 のように入力し Enter キーを押して、IoT マイコンを再起動する。

図 8-5

⑫. 再び iot Login：が表示されたら、PC（IoT マイコンの USB キーボードではないことに注意）でコマンドプロンプトを起動する（図 8-6)。

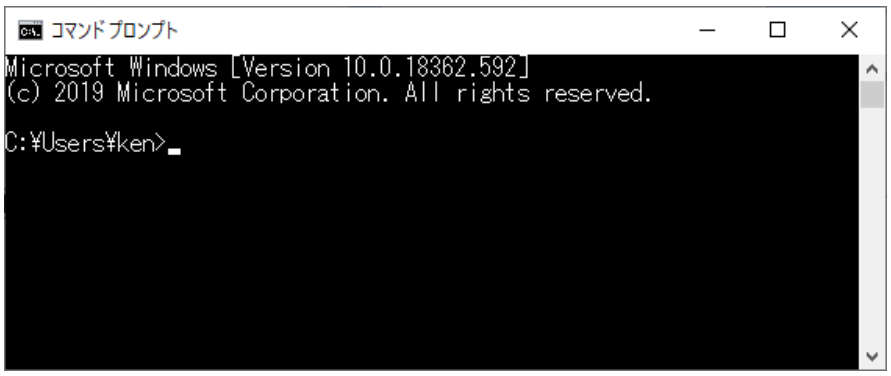


図 8-6

⑬. 次のように入力して Enter キーを押す。

ssh pi@iot.local

⑭. Password を求められるので、Login Password を入力して Enter キーを押すとプ

ロンプトが変わる（図 8-7）。

図 8-7

以上で、IoT マイコンは WiFi アクセスポイント経由で PC と接続できるようになった。ここで、ディスプレイと USB キーボードは取り外してよい。以後のキー入力操作は PC のターミナル接続で行える。なお、ログアウトする際はプロンプトに対して exit と入力して Enter キーを押す。

8．3．2 プログラム自動起動設定

次に、センサデータを処理するプログラムが電源投入時に自動起動するように設定する。

① ．６．２．１⑭の画面で、図 8-8 のように入力して設定ファイルを開く。

図 8-8

② ．図 8-9 に示すように、プログラムの自動起動部分の先頭にある#を削除して、記述を有効にする。

図 8-9

③. ^X（^は Ctrl キーを示す）を押す。書込み保存するかどうか問われるので、y（Yes）キーを押下する。書込みするファイル名が表示されるので、そのまま Enter キーを押せば書込み保存される。

④. 次のように入力して Enter キーを押す。

sudo reboot

これで、IoT マイコンが再起動し、設定が有効になる。

IoT マイコンには、各センサユニットからデータを受信するためのミドルウエアがインストールされている。上記設定を行うことでミドルウエアから値を受け取る機能が IoT マイコンの起動時に自動起動する。起動後のコンソールにはデータ取得時の Log 用メッセージが表示され続けるので、コマンド等の入力ができない。そのため、WiFi アクセスポイントへの接続設定などを行った後に上記設定を行った。以後、センサ閾値の設定は、ssh によるターミナル接続で行わなければならない。

８．４ ミドルウエアの設定と動作確認

ミドルウエアは、必用なセンサユニットからだけデータを受信するためのフィルタリング機能がある。これを設定し、ミドルウエアの動作確認を行う。

８．４．１ フィルタリングファイルの設定

IoT システムでは、5 種類のセンサユニットに対応するフィルタリングファイルがあり、その中にセンサユニット ID を記述することで、該当のセンサユニットからのデータだけを受信するようになる。以下にフィルタリングファイルの設定手順を説明する。

①．PC から ssh でログインする。

```
C:¥Users¥ken>ssh pi@iot.local
pi@iot.local's password:
Linux iot 4.19.42-v7+ #1219 SMP Tue May 14 21:20:58 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Linux iot 4.19.42-v7+ #1219 SMP Tue May 14 21:20:58 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jan 18 19:28:51 2020 from 2408:211:a807:c900:1431:2560:7192:c191
pi@iot:~ $
```

図 8-10

※ログイン方法は、Windows のコマンドプロンプトを起動して、６．２．１⑬以後の手順を行う。図 8-10 にコマンドプロンプトから ssh でログインした例を示す。

②．フィルターファイルのディレクトリ（Windows ではフォルダに該当する）に移動し、ファイル一覧を表示する（図 8-11）。

```
pi@iot:~ $ cd /opt/iotpf/config
pi@iot:/opt/iotpf/config $ ls
0x50_beacon_recv.filter  0x52_pir_motion.filter  0x54_light.filter
0x51_accel.filter        0x53_switch.filter      0x55_ope_status.filter
```

図 8-11

表 8-1 フィルタリングファイル名とセンサユニットの対応

センサユニット		ファイル名
位置推定センサ		0x51_accel.filter
人感センサ		0x52_pir_motion.filter
スイッチ		0x53_switch.filter
光センサ		0x54_light.filter
稼働状態センサ		0x55_ope_status.filter

表 1 にフィルタリングファイル名とセンサユニットの対応を示す。

③.『５．IoT センサユニットの設置手順』で確認したセンサ ID を準備する。
④. 下記のように入力して該当ファイルを nano で開く。

nano 0x54_light.filter

⑤. センサ ID を記述し改行する。同じ種類のセンサユニットを複数使用する場合は、行末改行しながら複数のセンサ ID を記述する（図 8-12)。

図 8-12

⑥. ^X を押下して、上書き保存する。
⑦. 上記を必要なフィルタリングファイルすべてについて行う。
⑧. 設定を有効にするために次のように入力して Enter キーを押し、IoT マイコンを再起動する。

sudo reboot

ここでは、nano を用いて IoT マイコン内のファイルに直接書込みを行ったが、FileZilla というソフトウエアを用いると、IoT マイコンと PC 間でファイルのダウンロード・アップロードが自由にできる。この機能を使い、PC 側でファイルを編集し履歴として保存することが可能である。この FileZilla の入手とインストール、簡単な使用方法を Appendix

B.に掲載した。なお、PC 内で編集したファイルをアップロードする場合、文字コードに注意すること。IoT マイコン OS では UTF-8 を使用している。

８．４．２ ミドルウエアの動作確認

telnet を利用してミドルウエアに接続し、データ取得出来ているか確認する。

①．次のように入力して、データが表示されれば、ミドルウエアからセンサデータを受信している。

telnet localhost 8566

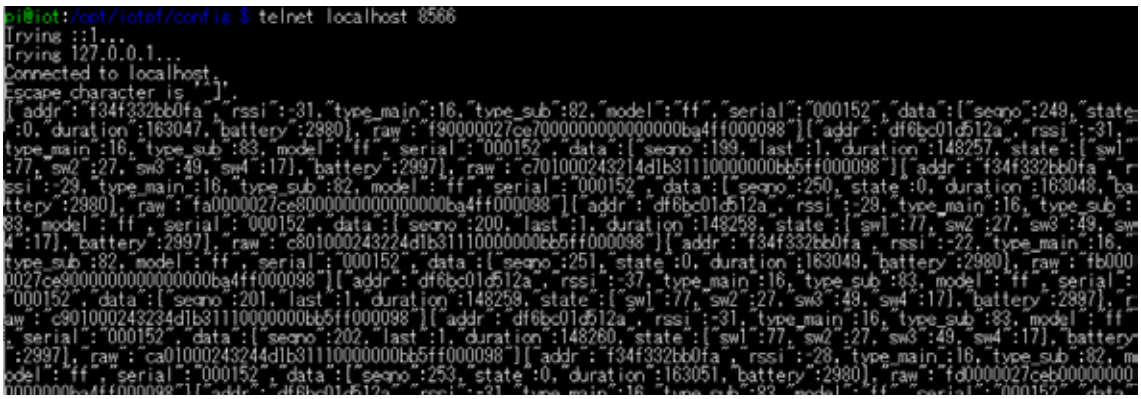


図 8-13

図 8-13 に telnet 接続してデータ表示している様子を示す。

②．telnet 接続を終了する。^］と入力して表示を止める。さらに^z と入力して telnet 接続を終了する（^ は、Ctrl キーを押しながら続く文字を押下する）と、プロンプト表示に戻る。

第9章 各種 ID とセンサ閾値の設定

Cloud 上で稼動する IoT 共通プラットフォーム内で、企業や工場を識別するための設定ファイル /iot/config/sensor.json がある。このファイル内にはセンサユニット ID とセンサ毎の閾値設定部が含まれている。以下、sensor.json の設定手順を説明する。nano で、または PC にダウンロードしてテキストエディタで設定後、アップロードする。

※ダウンロードは、FileZilla（Appendix：B）で行える。

９．１　sensor.json の設定例

図 9-1 に sensor.json ファイルの設定例を示す。この設定例では 000152 という数字がいくつか見られるが、これは使用したセンサユニットの ID、000152 を設定したものである。

図 9-1

以下、キーワード毎に内容を説明する。提供される元のファイルの『-』および数字部分を環境に合わせて変更する。

①. factory：設置する工場の ID を設定する。

設定例：FCH001　　CH は千葉、001 は 1 番目の意味。

この ID でクラウド上のデータを識別される。

②. machine：設置する機械の ID を登録する。

id：機械の ID

設定例では、センサ毎に機械の ID が異なるが、IoT システムが機械に 1 個の

センサを取り付ける前提になっているからである。この解説では光センサ 2 個を用いたので、2 つの設定が記述されている。

ope：使用する稼動センサユニットの ID

lig：使用する光センサユニットの ID

swi：使用するスイッチユニットの ID

ligtype：3 個の光センサのうち使用するものの番号（コネクタ左から 1,2,3）

※1,3 だけ使用して 2 は未使用という設定もできる。

threshold：光センサで稼動状態を推定する際の閾値

※後に説明する『取得データの確認』で認識結果を参照しながら、センサの閾値を設定する。

ave：稼動状態センサで稼動状態を推定する際の閾値の平均値

※後に説明する『取得データの確認』で認識結果を参照しながら、センサの閾値を設定する。

var：稼動状態センサで稼動状態を推定する際の閾値の分散値

※後に説明する『取得データの確認』で認識結果を参照しながら、センサの閾値を設定する。

③. area：作業員の介在状態を推定する場所の ID

pir：使用する人感センサユニットの ID

第10章 取得データの確認方法

センサから得たデータを元に推定した稼働状態は、IoT マイコン内部のデータベース（MySQL）に記録されている。このデータベースの記録を確認すれば閾値が適切かどうか分かる。本章では、このデータベースのデータを表示する手順を説明する。

１０．１　MySQL へのログイン

PC から ssh ログイン後、次のように入力して Enter キーを押し、パスワードを入力して MySQL にログインする。（パスワードは機材提供時にあらかじめ通知されている。）

mysql –D IoI –u iot -p

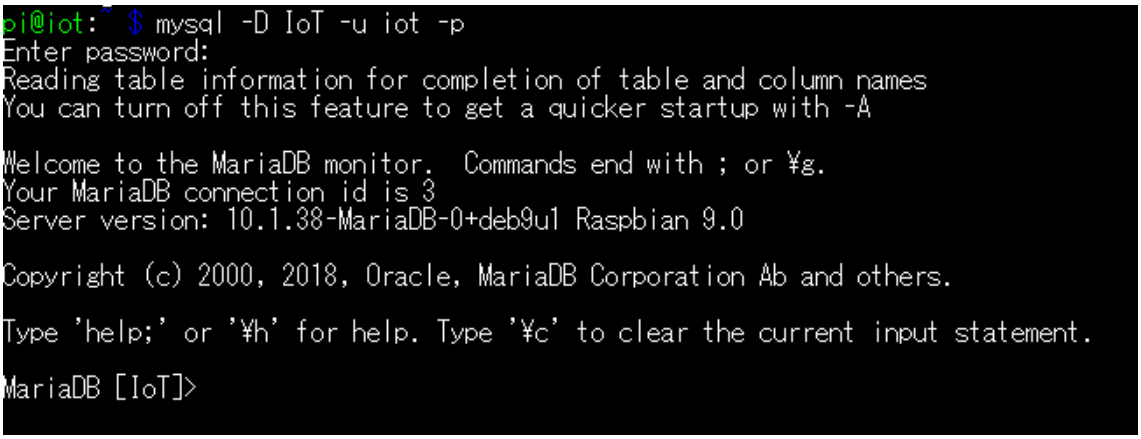


図 10-1

図 10-1 は、MySQL にログインしてプロンプトが表示されている様子である。

１０．２　データベースとテーブルの確認

IoT システムで使用するデータベースは IoT である。
次のように入力するとデータベースが確認できる。行末に ；（セミコロン）を付けて Enter キーを押す。

show databases;

テーブルを確認するには、次のように入力する。上と同ように行末に　；　を付けて Enter キーを押す。

show tables;

図 10-2 はデータベースおよびテーブルを表示した様子である。MariaDB [IoT]と表示されているのは、IoT というデータベースファイルを使用していることを示している。
MariaDB とは、MySQL の新しい名称である。

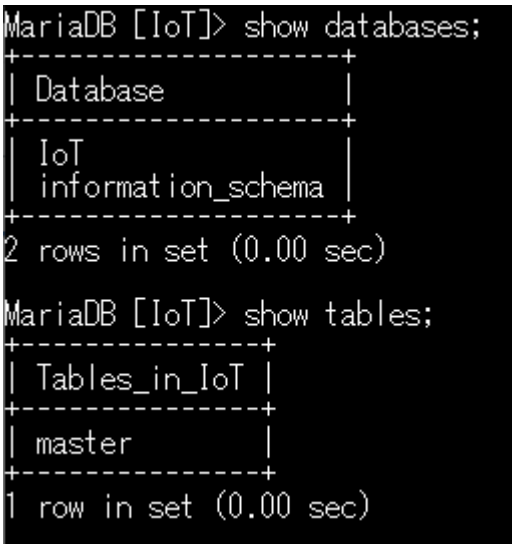


図 10-2

10. 3 テーブル内容の表示

上記で確認したテーブル master に記録されているレコードを表示するには、つぎのように入力して Enter キーを押す。

select * from master;

図 10-3 は、master テーブルの内容を表示した例である。

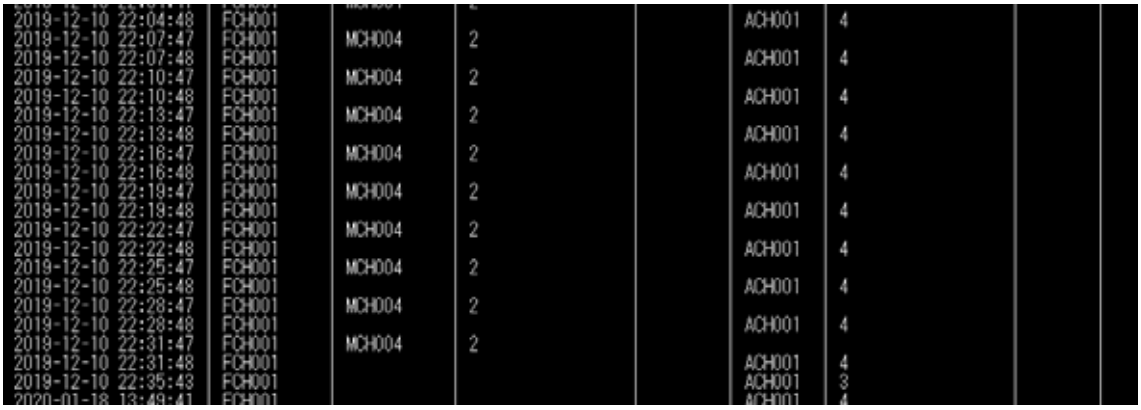


図 10-3

また、図 10-4 のように入力すると、日付時刻で表示するデータを限定できる。

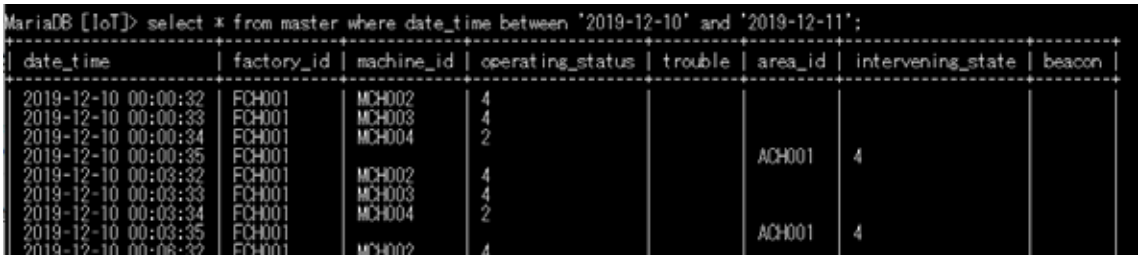


図 10-4

表 10-1 テーブルのカラムと内容

date_time	factory_id	machine_id	operating_status	trouble	area_id	intervening_status
推定時刻	工場 ID	機械 ID	稼働状態	トラブル	エリア ID	作業者介在状態

表 10-1 にテーブルのカラムと内容を示す。稼動状態の内容は表 10-2 のように記録される。

表 10-2 稼働状態

『稼働状態』の内容	意味
1	稼働開始
2	稼働中
3	停止開始
4	停止
5	トラブル開始
6	トラブル中

また、作業者介在状態は表 10-3 のように記録される。

表 10-3 作業者介在状態

『作業者介在状態』の内容	意味
1	介在開始
2	介在中
3	非介在開始
4	非介在中

10．4　ログアウト

MySQL からログアウトする際は、以下のように入力して Enter キーを押す。

exit

図 10-5

ログアウトすると、プロンプトが元に戻る（図 10-5)。

第11章 取得データの取り出し

取得データを取り出す際の入力操作は、『８．取得データの確認』と同じ操作を行うが、接続する際に ssh ではなく、予めインストールしておいた Tera Term を用いる。以下、このソフトウエアを使用した、取得データ取出し手順を説明する。

１１．１　Tera Term の起動と IoT マイコンへの接続

①．Tera Term を起動する。

③　．IoT マイコンに接続する。

Tera Term 起動時に開いたウインドウで、ホスト：pi@iot.local と入力し、サービス：ssh を選択して OK ボタンを押す（図 11-1)。

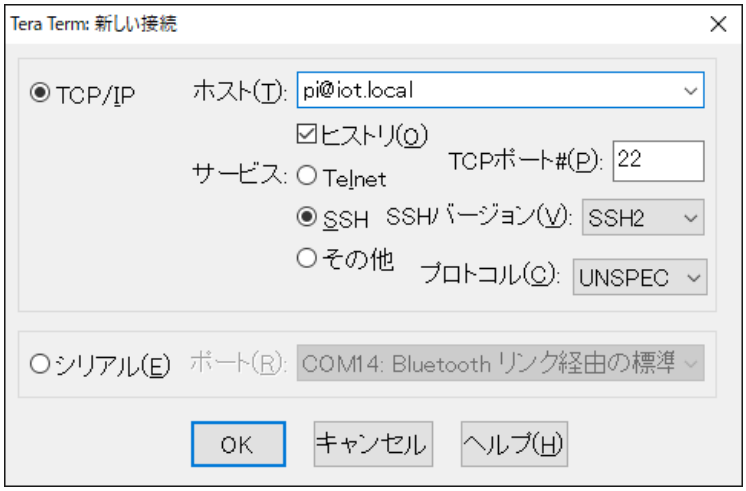


図 11-1

③．SSH 認証ウインドウに、ユーザ名：pi 、パスフレーズにログインパスワードを入力して OK ボタンを押す。

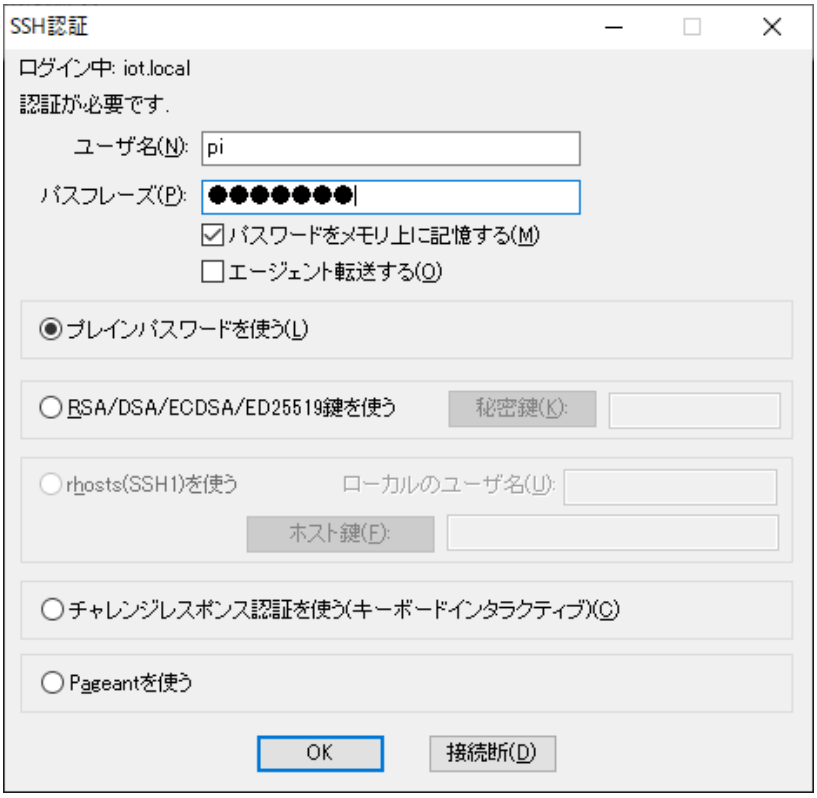


図 11-2

④ ．図 11-3 のように、ssh 接続時のプロンプトが表示される。

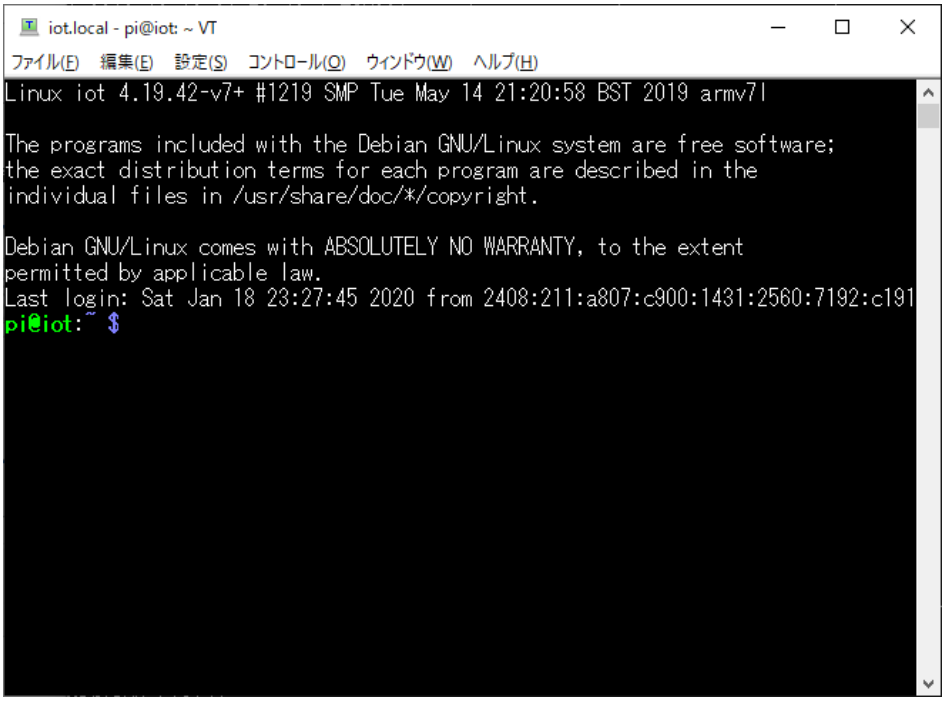


図 11-3

⑤ .『８．取得データの確認』と同じ手順で、MySQL にログインする。図 11-4 のように MySQL に接続される。

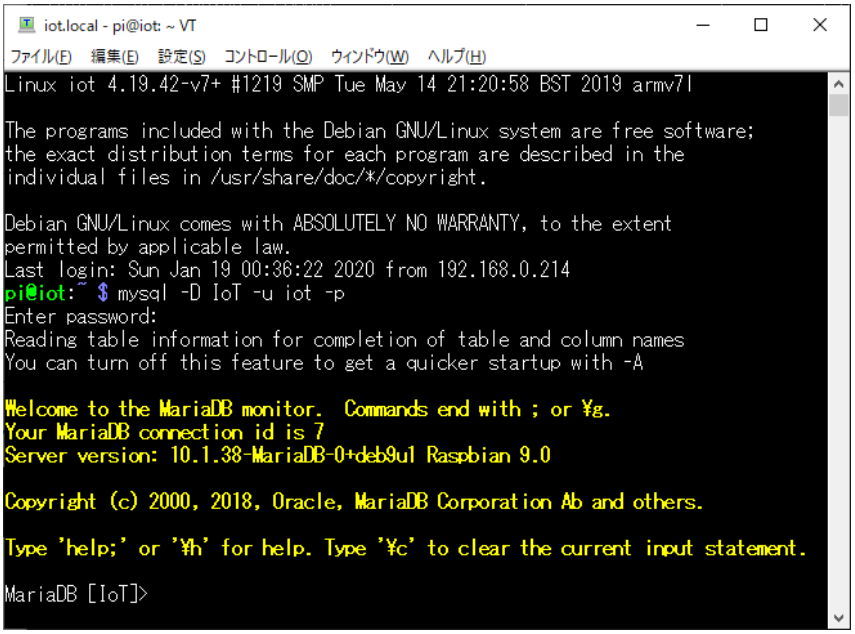


図 11-4

⑥ . ここで、ログファイルの設定を行う。図 11-5 に示すように、ファイル→ログと辿る。

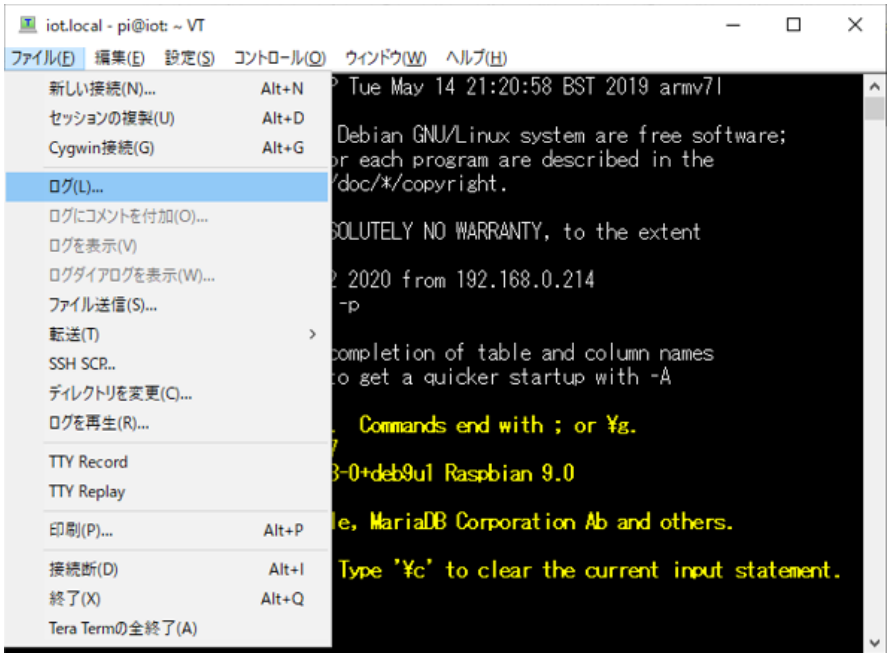


図 11-5

図 11-6 のように、ログ設定のウインドウが表示されるのでログファイル名と PC 内の保存する場所を指定して保存ボタンを押す。

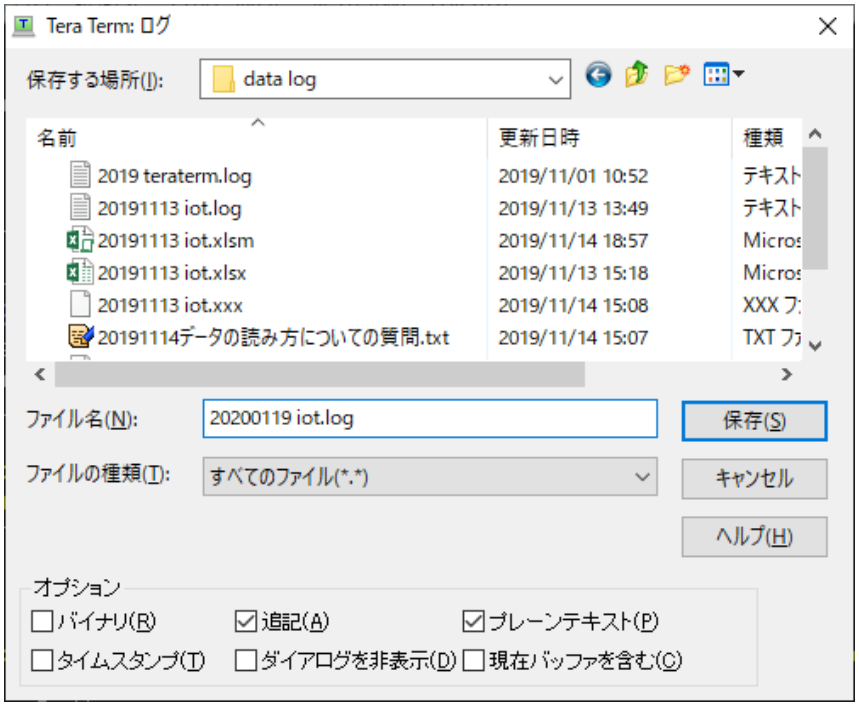


図 11-6

図 11-7 のウインドウが表示される。ログを終了する際は、閉じるボタンを押す。

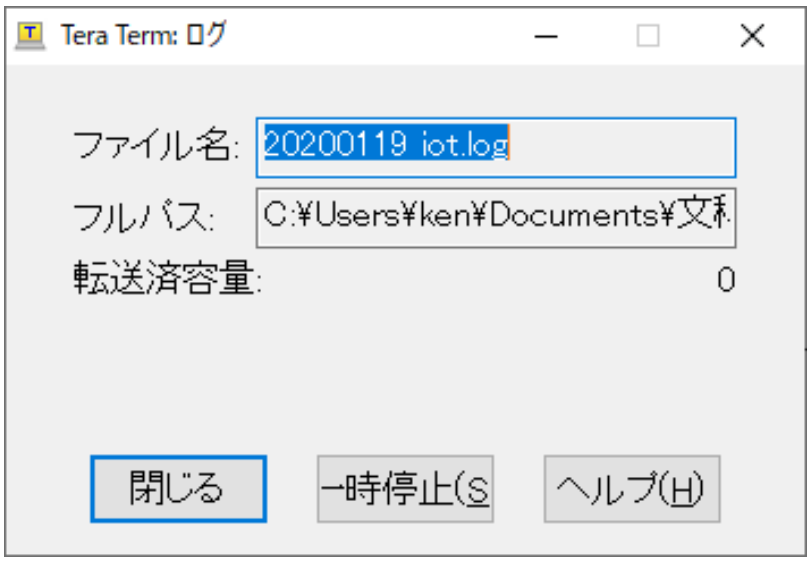


図 11-7

これ以後の操作と表示がログファイルに記録される。

⑦. master テーブル内容を表示させる。MySQL プロンプトに対して、以下のように入力して Enter キーを押す。

select * from master;

⑧. テーブル内容の表示が止まり、プロンプトが表示されたら、ログウインドウの閉じるボタンを押してログを終了する。これで、IoT マイコンに記録された稼動データが PC にログとして保存された。

⑨. Tera Term のウインドウを閉じて、Tera Term を終了する。

⑩. 適当なテキストエディタでログファイルを開く。図 11-8 のようにファイルの先頭には、MySQL に入力したコマンドや、テーブルカラムの罫線なども含まれている。

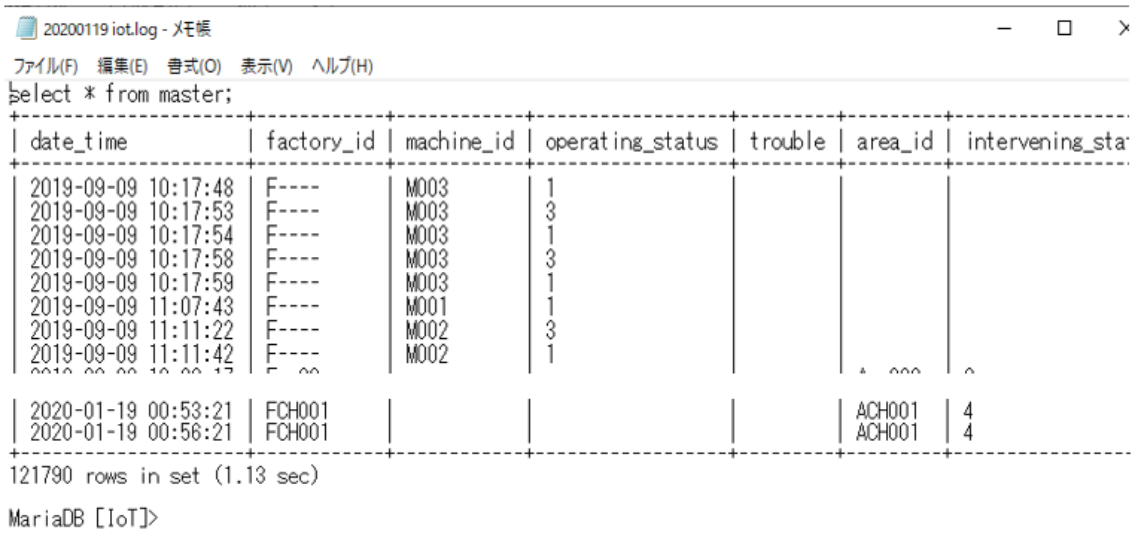


図 11-8

⑪. 先頭の 2 と最後の 4 行を削除して、ファイル名を指定して拡張子を txt として保存する。

第12章 データの簡易解析手順

保存したテキストファイルを Excel で解析するための手順について説明する。

※解析手法については、この報告では触れていない。

①. ファイル名を変えて保存した取得データファイルを Excel で開く。この際、テキストファイルを指定して開くと図 12-1 のウインドウが開く。『先頭行を見出しとして使用する』にチェックを入れて『次へ』進む。

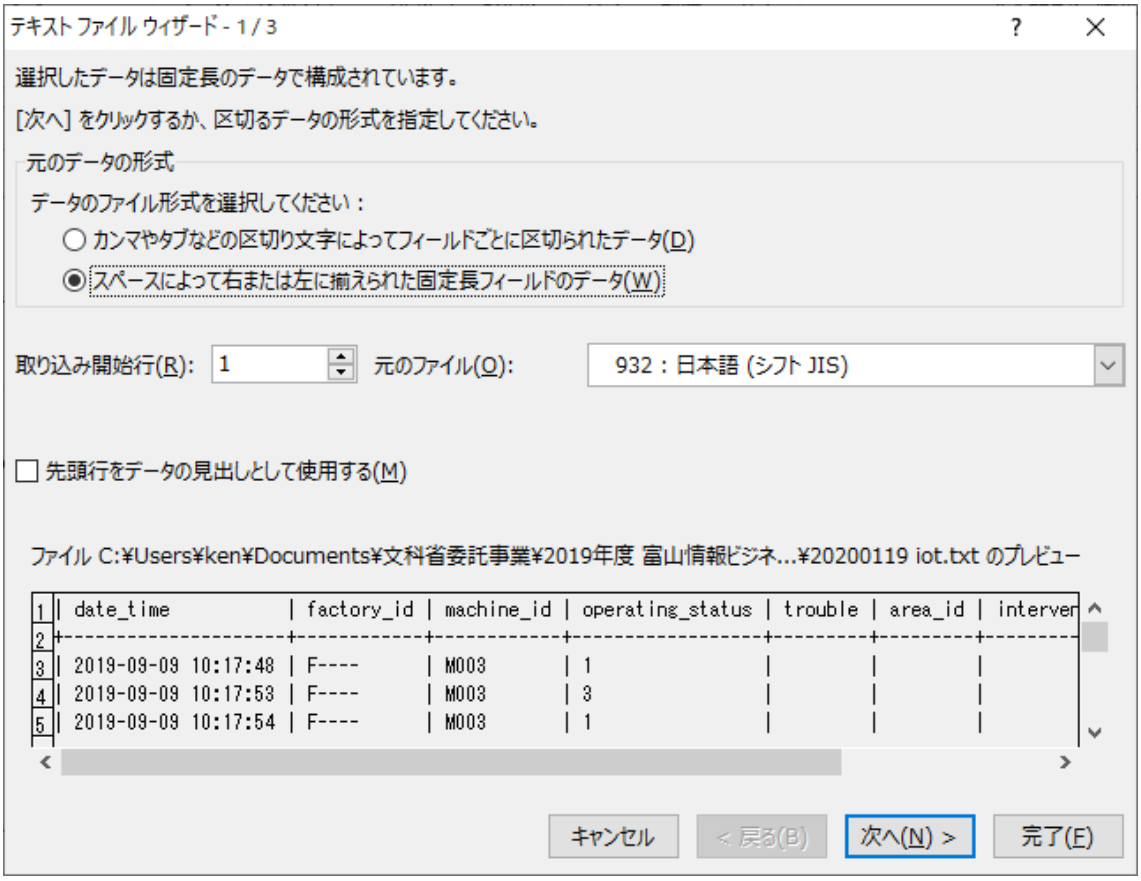


図 12-1

①．データのプレビューの trouble カラムの後の縦線の手前をクリックしてデータ区切りを指定する。同ように最後のカラムの縦線を挟むようにデータ区切りを指定する（図 12-2 の赤矢印で示す)。して、『次へ』『次へ』と進み、完了ボタンを押す。

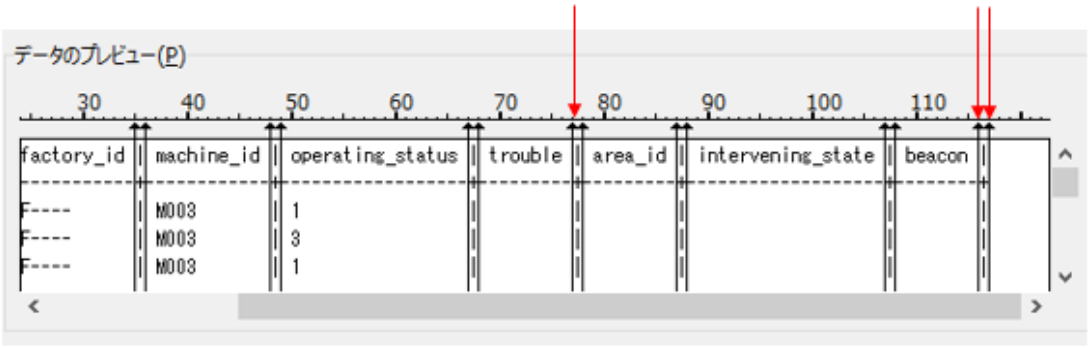


図 12-2

② ．しばらくすると図 12-3 のワークシートが開く。

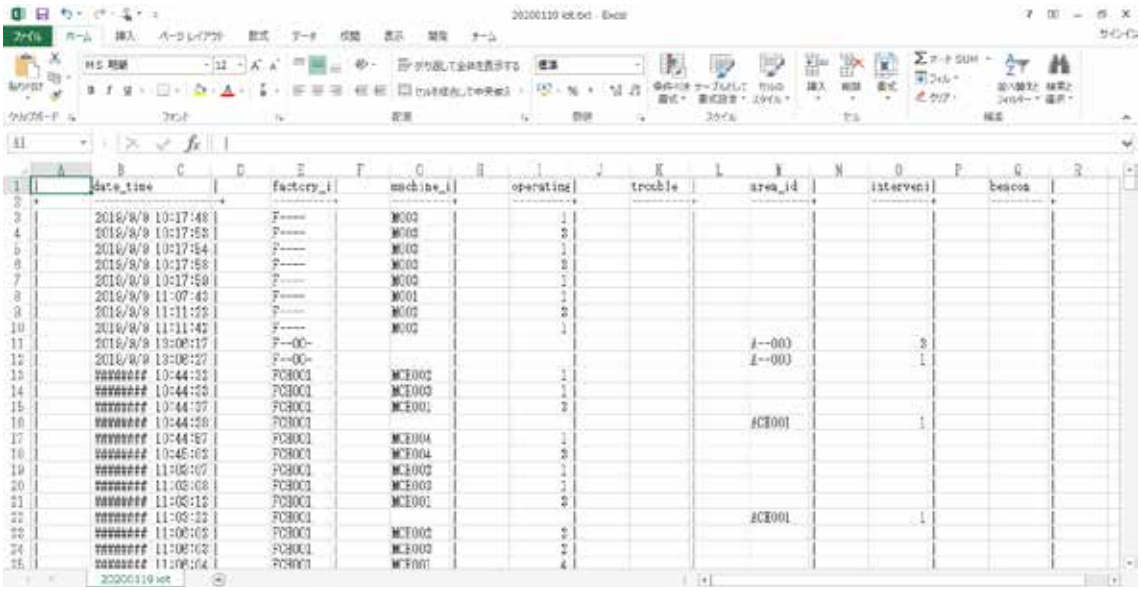
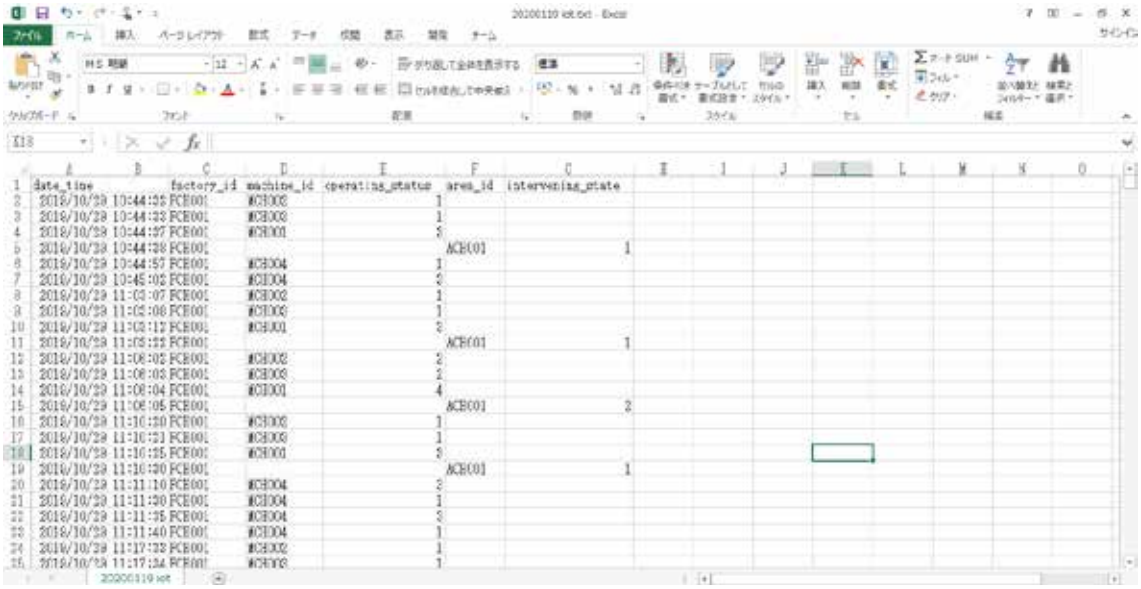


図 12-3

③ ．ワークシートで、縦横の罫線部分と不要なカラム、および不要なデータを削除すると図 12-4 のようになる。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	date_time		factory_id	machine_id	operating_status	area_id	interval_state								
2	2018/10/29 10:44:25		FCE001	#CE003		1									
3	2018/10/29 10:44:33		FCE001	#CE003		1									
4	2018/10/29 10:44:37		FCE001	#CE003		3									
5	2018/10/29 10:44:38		FCE001			ACE001	1								
6	2018/10/29 10:44:57		FCE001	#CE004		1									
7	2018/10/29 10:45:02		FCE001	#CE004		3									
8	2018/10/29 11:03:07		FCE001	#CE002		1									
9	2018/10/29 11:03:08		FCE001	#CE003		1									
10	2018/10/29 11:03:17		FCE001	#CE001		3									
11	2018/10/29 11:03:28		FCE001			ACE001	1								
12	2018/10/29 11:08:02		FCE001	#CE002		2									
13	2018/10/29 11:08:03		FCE001	#CE003		1									
14	2018/10/29 11:08:04		FCE001	#CE001		4									
15	2018/10/29 11:08:05		FCE001			ACE001	2								
16	2018/10/29 11:10:20		FCE001	#CE002		1									
17	2018/10/29 11:10:23		FCE001	#CE003		1									
18	2018/10/29 11:10:25		FCE001	#CE001		3									
19	2018/10/29 11:10:30		FCE001			ACE001	1								
20	2018/10/29 11:11:10		FCE001	#CE004		3									
21	2018/10/29 11:11:20		FCE001	#CE004		1									
22	2018/10/29 11:11:35		FCE001	#CE004		3									
23	2018/10/29 11:11:40		FCE001	#CE004		1									
24	2018/10/29 11:17:23		FCE001	#CE002		1									
25	2018/10/29 11:17:24		FCE001	#CE003		1									

図 12-4

⑤. このファイルを『名前を付けて保存』で Excel Book として保存する。保存したファイルの稼動状態（operating_status）カラムで分析して、稼動開始（1）から停止開始（3）までの時間を合計すれば総稼働時間となる。

この解説では、すべてのセンサを 1 台の機械に設置しているので、IoT システムでの稼動状態推定アルゴリズムが、かえって複雑に思える。実際に作業者が正確にスイッチを押してくれるのであれば、推定などは必要がない。また、近年の金属加工機械などは高精度になっていて、そもそも振動が発生しない。このような設備には、稼働センサユニットは不要で、光センサユニットでパイロットランプによる稼動状況の把握が良さそうである。

第13章 センサユニットの撤去手順

撤去は、完全に撤去する場合と、移設する場合の撤去がある。

１３．１　IoT システムを完全に撤去する場合

全体を別の場所、または別の設備に移設する場合もこれに該当する。以下手順を説明する。

①．IoT マイコンをシャットダウンする。ssh 接続して PC からつぎのように入力して Enter キーを押す。

```
sudo shutdown -h now
```

②．しばらくすると、IoT マイコンのアクセス LED（緑）が消灯し、電源 LED（赤）のみの点灯となる。
③．電源を切る（電源アダプタを取り外す）
④．センサユニットをすべて取り外し、固定に使用した養生テープなどは取り除く。順番は特にない。
⑤．WiFi アクセスポイントを取り外す。
⑥．LAN ケーブル、電源ケーブル等を回収する。

※撤去は以上で終了。

⑦．別の場所（設備）に移設する場合は、この後『5 章．センサユニットの設置手順』以後の作業を行う。

１３．２　近い設備に移設する場合

１１．１の①~④を行い『5 章．センサユニットの設置手順』以後の作業を行う。

※『近い設備』とは、WiFi アクセスポイントに IoT マイコンが接続できる範囲、または、センサユニットが BLE で IoT マイコンと通信できる範囲を指す。

第14章 トラブルシュート

表 5 にトラブルシュートの対応表を示す。

表 14-1

情況	対応
1. WiFi アクセスポイントに Raspberry Pi が繋がらない。	①. WiFi アクセスポイント機器でローカルな DHCP 環境を設定して、接続を試みる。 ②. LAN ケーブルで WiFi アクセスポイントに接続して、PC からアクセスできるか確認する。
2. 停電した。	停電した場合、設備も一定時間停止していると思われるので、給電開始後プログラム自動起動が行われるまで待つ。PC から ssh 接続で接続できれば、問題ない。
3. 特定のセンサデータが記録されなくなった。	センサユニットの電池切れが考えられる。ミドルウエア接続テストで、センサからのデータを取得すると、電池電圧が分かる。またセンサユニットの LED が点灯しないような場合はボタン電池を交換する。
4. 光センサが記録されなくなった。	光センサの取付け状態を確認する。初期の状態と変わっている場合は再取付けして、閾値の再設定を行う。
5. 記録されている時刻がずれている。	sudo nano /etc/rc.local で次の記述が有るか確認する。ntp で時刻を合わせている。 sudo service ntp restart sleep 10
6. 稼動状況の記録が取れない。	稼働状況センサの閾値が変化した可能性がある。閾値の再設定を行う。
7. データベースのレコードが増えた。	記録したデータを取り出した後、データベースのレコードを削除する。 delete from master;　で master テーブルのレコードを削除する。概ね 2 ヶ月に 1 度程度を行なえば良い。
8. 長期にわたり設備が止まる。 ※工事、夏季・年末年始休業など。	ssh 接続して次のように入力して IoT マイコンをシャットダウンして、電源を切る。

	sudo shutdown -h now
9. センサを別の設備に設置したい。	上記同様、シャットダウンして電源を切り、センサを移設後に、電源投入し閾値などを設定する。
10. 別の WiFi アクセスポイントに接続したい。	WiFi アクセスポイントを交換する前に、ssh 接続により次のようにして新しいアクセスポイントの ssid とパスワードを追加して再起動する。 sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
11. 別のセンサを追加したい（取り外したい)。	/iot/config/sensor.json を再設定して、再起動する。

Appendix:

A.　WiFi アクセスポイントの IP アドレス設定の例

今回使用した WiFi アクセスポイント（BUFFALO WSR-2533DHPL）の設定では、機器メーカの WEB サイトで提供されているソフトウエア『エアステーション設定ツール』を PC にインストールして行うのが容易である。該当のソフトウエアは、あらかじめダウンロードしてインストールしておく。参考までに、このテキスト執筆時点でのダウンロードサイトを下記する。

https://www.buffalo.jp/support/download/detail/?dl_contents_id=60749

以下に『エアステーション設定ツール』を用いた IP アドレス設定手順を説明する。

①.『エアステーション設定ツール』を起動すると開くウインドウの『次へ』をクリック。

②. 図 54 のウインドウに WiFi アクセスポイントが表示されるので、該当の機器をクリックして選択し『次へ』をクリックする。

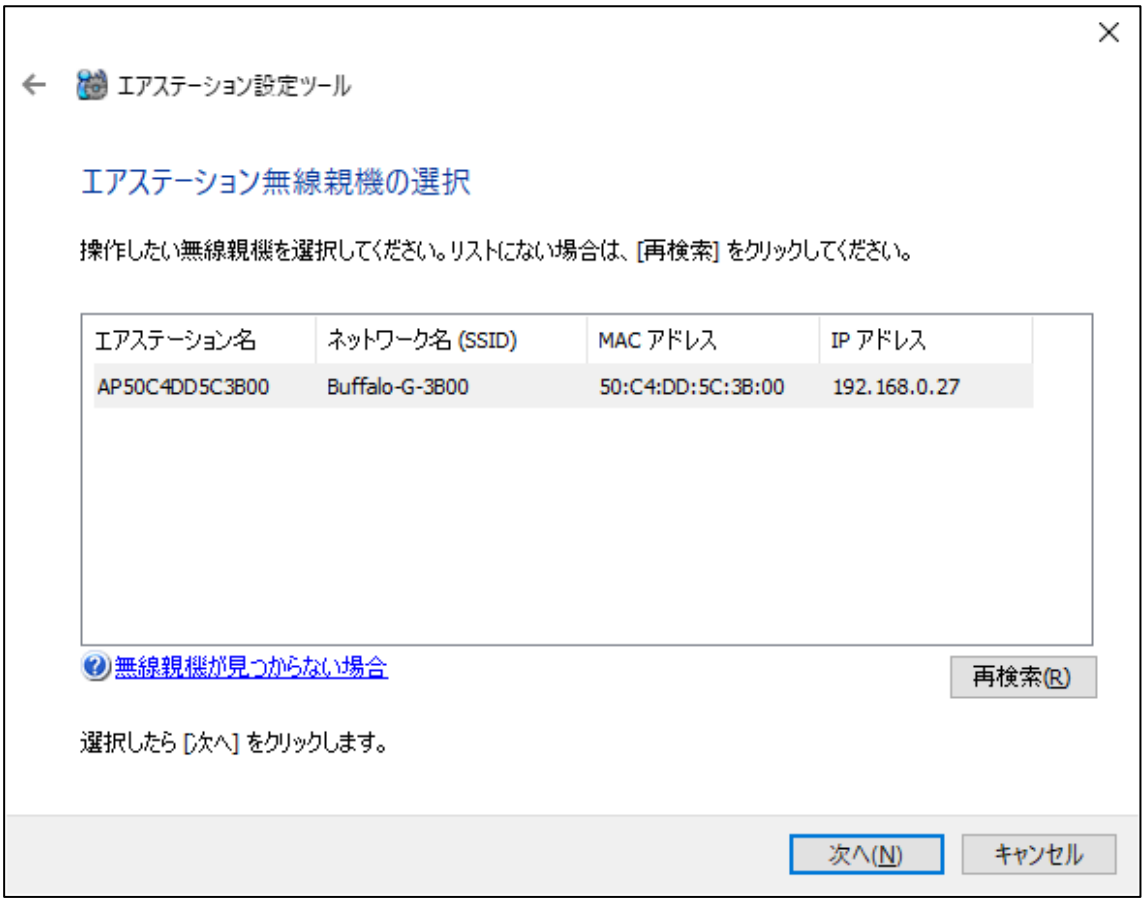


図 14-1

③.『操作の選択』ウインドウが表示されるので、『この無線親機の IP アドレスを指定する』をクリックする。

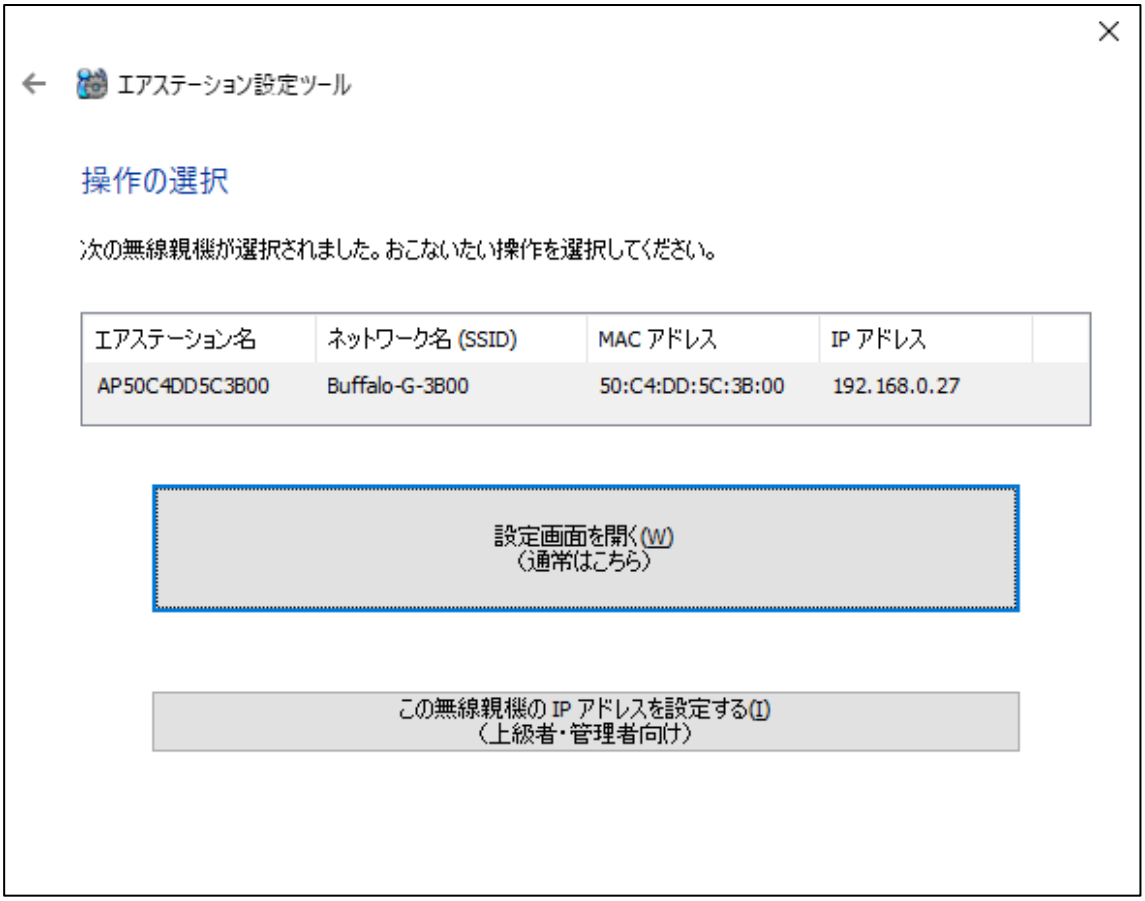


図 14-2

④.『無線親機の IP アドレス設定』ウインドウで、DHCP または IP アドレスを指定する。

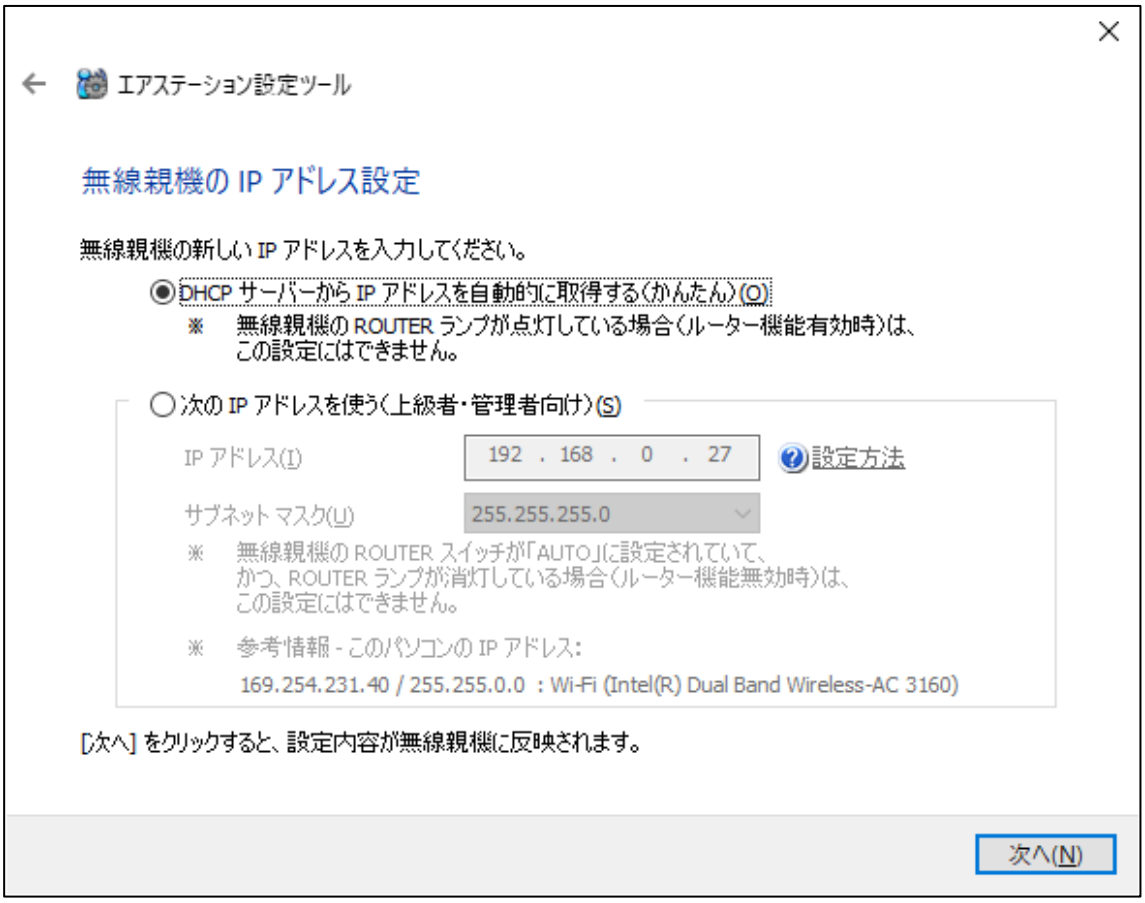


図 14-3

⑤. 最後のウインドウで、『次へ』をクリックすると、WiFi アクセスポイントが再起動する。再起動までには数分かかる。

WiFi アクセスポイントが再起動したら、ネットワーク内の PC から ping コマンドを発行して。WiFi アクセスポイントが応答する事を確認する。

B.　FileZilla のインストール

①. 次の URL から FileZillaClient for Windows をダウンロードしてインストールする。

https://filezilla-project.org/download.php?type=client

②. プログラムを起動して、以下の項目を入力しクイック接続ボタンを押すと図 57 のように、IoT マイコンの階層構造が右側に、PC 内のフォルダが左側に表示される。

ホスト（sftp://iot.local）
ユーザ名（pi）
パスワード（ログインパスワード）
ポート（未入力）

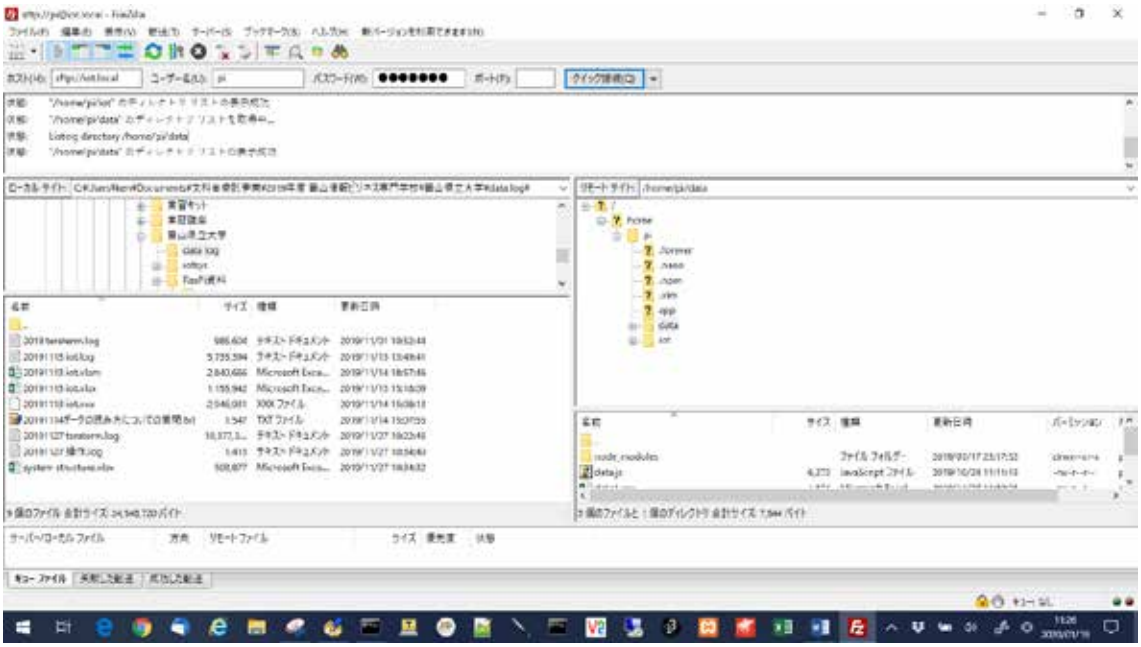


図 14-4

③. IoT マイコン内のファイルを右クリックしてプルダウンからダウンロードを選択すると、PC のカレントフォルダにファイルがダウンロードされる（図 58）。

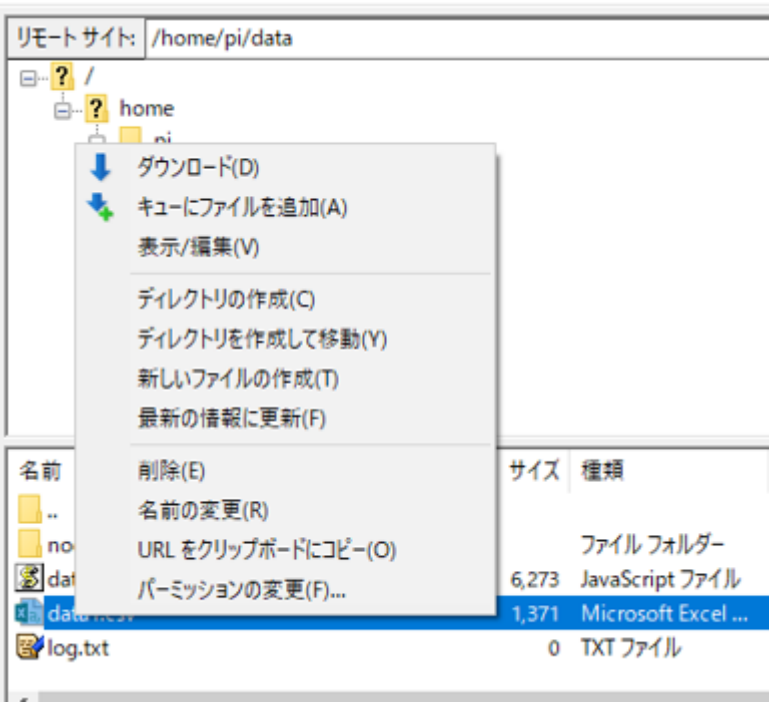


図 14-5

④. 同じように PC 内ファイルを右クリックすればアップロードができる。

C.　Tera Term のインストール

①. 次の URL から Tera Term をダウンロードしてインストールする。

https://forest.watch.impress.co.jp/library/software/utf8teraterm/

最新バージョンは、v4.105(19/12/07)となっている。

②. ダウンロードした実行形式ファイルをダブルクリックしてインストールする。

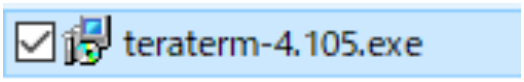


図 14-6

※設定はデフォルトのままで良い。

③. インストールは僅かな時間で終了し、デスクトップに図 60 に示すショートカットができる。

図 14-7

令和2年度「専修学校による地域産業中核的人材養成事業」
（Society5.0 等対応カリキュラムの開発・実証）

富山県をモデルとした「モノづくり」現場に
IoT を導入する中核的人材育成

製造 IoT 応用演習 テキスト教材

令和 3 年 2 月発行

学校法人 浦山学園 富山情報ビジネス専門学校
〒939-0341 富山県射水市三ヶ613
TEL:0766-55-1420 FAX:0766-55-0757