ESP32 講習会

2018 年度生用

梶本研究室

ver.1 2017.12.16

Change Log

ToDo

BLE 関係

1　はじめに

ESP32-DevKitC を活用していく（本稿では今後便宜上 ESP32 と呼ぶ．厳密には ESP32 はコアチップの名称）．2017 年冬の時点で比較的新しく，**Arduino ライブラリも日々更新されているので本資料のやりかたは古い可能性が常にある．** PC 側のプログラムは Processing を用いる．

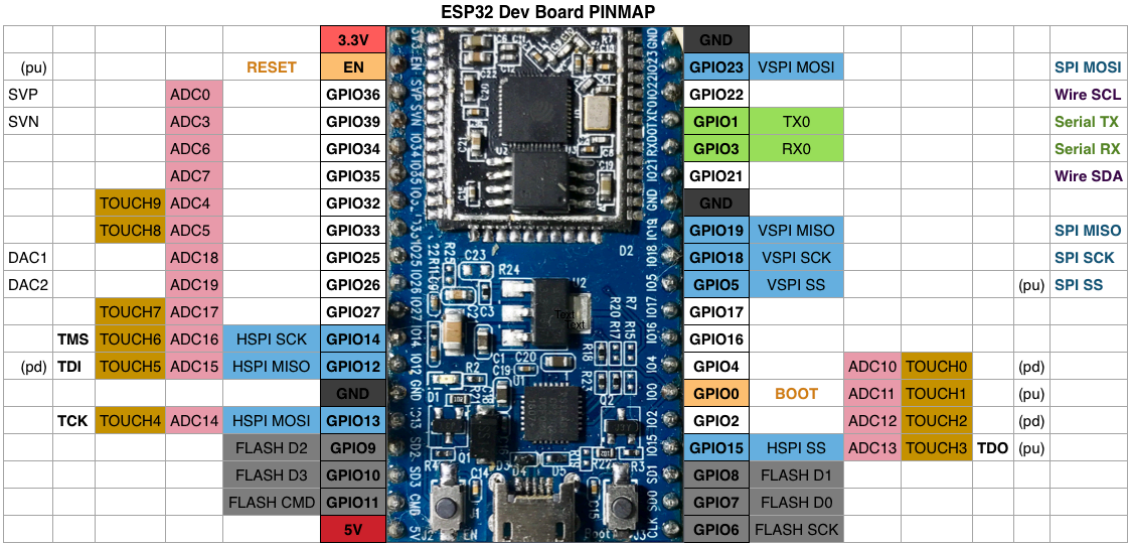


図 1　ESP32-DevKitC ピン配置

(https://github.com/espressif/arduino-esp32/blob/master/docs/esp32_pinmap.png)

入手先：http://akizukidenshi.com/catalog/g/gM-11819/

ブレッドボード：http://akizukidenshi.com/catalog/g/gP-12366/　（通常の 5 列ではなく 6 列のもの．このキットは幅が広いため）

1.1 課題の進め方

- 2 章「LED の点滅」：LED を点滅させる．

- 3 章「外付けスイッチ」: 外付けスイッチによる入力を行う．割込みの練習も．
- 4 章「アナログ入力」: 加速度センサを例にアナログ入力を使う．
- 5 章「RC サーボモータの駆動」: RC（ラジコン）サーボモータを動かす．
- 6 章「アナログ出力」: 内蔵の DA 変換器を使う．
- 7 章「SPI 通信」: SPI 通信によるポートの拡張例として DA 変換ポートの増設を行う．
- 8 章「DC モータ制御」: H ブリッジ IC とモータ付属のエンコーダで DC モータを PD 制御する．
- 9 章「データロギング」: Processing を用い，実用的なデータロギングを行う．
- 10 章「無線通信」: WiFi および Bluetooth 通信のサンプルを動かす．

一週目：一日目 2〜3 章，二日目〜5 章，三日目 6 章，四日目 7 章，五日目予備日
二週目：一日目〜二日目 8 章，三日目〜四日目 9〜10 章
というペースで進める．
なお本資料中でたびたび「〇〇については自習すること」と言及しているが，これはすべて知っておくべきことなので本当に自習すること．

1.2 レポート

最後にレポートを書く．添削します．

- Word を標準とします．Word 講習を兼ねていますので，図表番号への参照などの**参照機能**は必ず使うこと． Word 使い方講習は講習会の途中に行う．
- 実験中に取得した図（特にオシロスコープの画面キャプチャ）を貼る．
- **回路図など，テキストにある内容を繰り返す必要はない**．
- コツは**レポートを書きながら実験を行う**こと．これは研究の場面でも同じ．データはリアルタイムに文書化．常に写真，動画を撮るようにする．「論文を書くときに初めて写真を撮る」のは普段の研究態度として誤り．
- 提出されたレポートの添削と回覧は研究室全体で行う．

2　LED の点滅

開発環境を整え，外付け LED を駆動する．

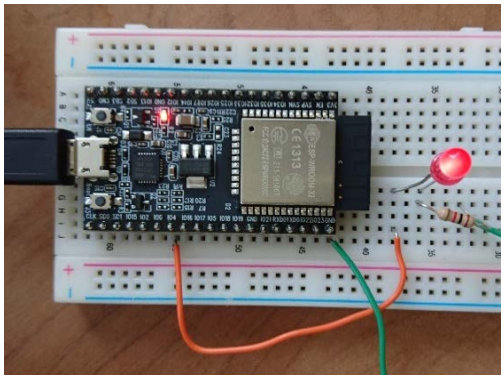
- Arduino IDE インストール https://www.arduino.cc/en/main/software
- https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/windows.md を読みながら，Git インストール→レポジトリクローン（Arduino のスケッチ保存場所を確認する必要あり）などなど
- Arduino IDE で選ぶボードは ESP32 Dev Module

図 2 を見ながら配線する．ブレッドボードを真似るのではなく回路図から読み解くこと．

- ESP32 のデジタル入出力については以下の点に注意
  - GPIO6～11 は内部で使用されているため使用非推奨．
  - GPIO34,35,36,39 は入力専用
- LED の極性はテスタで確認すること．
- LED に直列に接続する抵抗（制限抵抗）はここでは 1kΩ としているが，電源電圧が変わる場合等もあるので計算できなければならない．抵抗値の計算方法は下記リンクなどに記載されているので自習する．特殊なものでなければ大体 1～10mA 流す．
  http://www.rank-a.com/html/led.html
  http://www.ops.dti.ne.jp/~ishijima/sei/letselec/letselec11.htm
- 抵抗のカラーコードは読めるようになるとよい．
- ここではブレッドボードを使用．外す際には USB コネクタ等に力が加わると破損するのでマイナスドライバ等を用いる．
- 回路図中の同じネット名は接続されていることを表す．例えば図 2 の場合は 2 箇所の「GND」は接続されることに注意．

[課題1]　LED を流れる電流を計測する（抵抗間の電圧を計測し，オームの法則により算出する）．その値は前述の計算方法で求めた予想と比較して妥当か．

```
void setup() {
  pinMode(4, OUTPUT);
}

void loop() {
  digitalWrite(4, HIGH);
  delay(250);
  digitalWrite(4, LOW);
  delay(250);
}
```



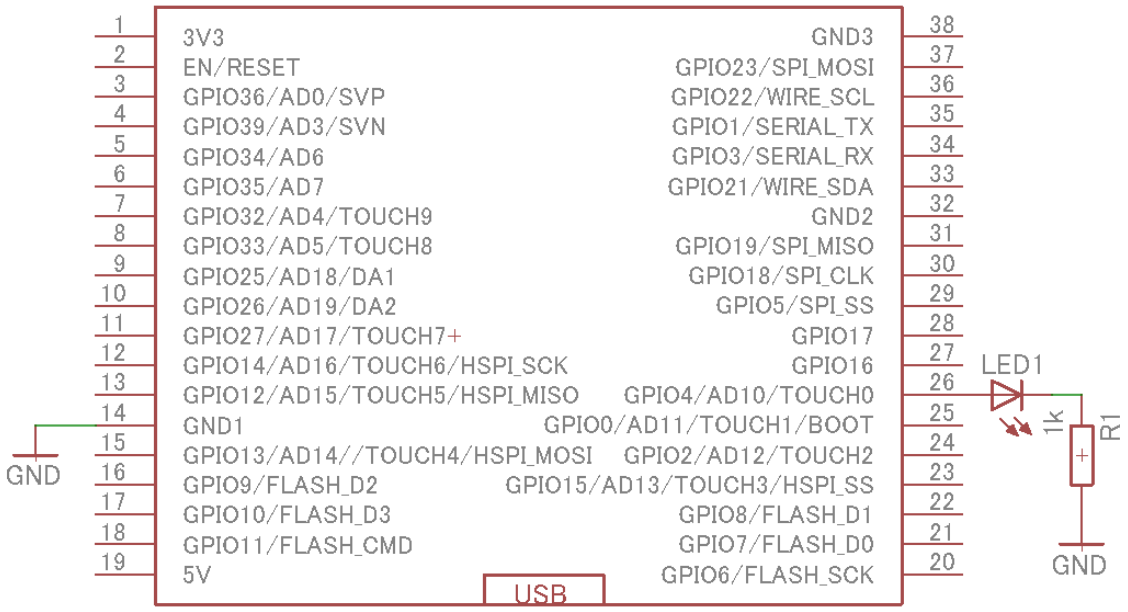


図 2　LED 点滅のソースと回路

3　外付けスイッチ

外部からスイッチ入力を受け付けるようにする（なお ESP32 基板には入力用スイッチもある）．図 3 参照．タクトスイッチは押下時どこが導通するかテスタでチェック．

[課題2]　スイッチ回路に抵抗が必要な理由を考察する．下図のサンプル回路ではボタンを押したときにどのようなロジック入力になるか．

[課題3]　タクトスイッチを押すと LED が光るようにする(digitalRead 関数)

[課題4]　割込みプログラミングについて自習し，割込みを用いたプログラムに改変する．スイッチを押すたびに LED が点滅するようにする（attachInterrupt 関数）

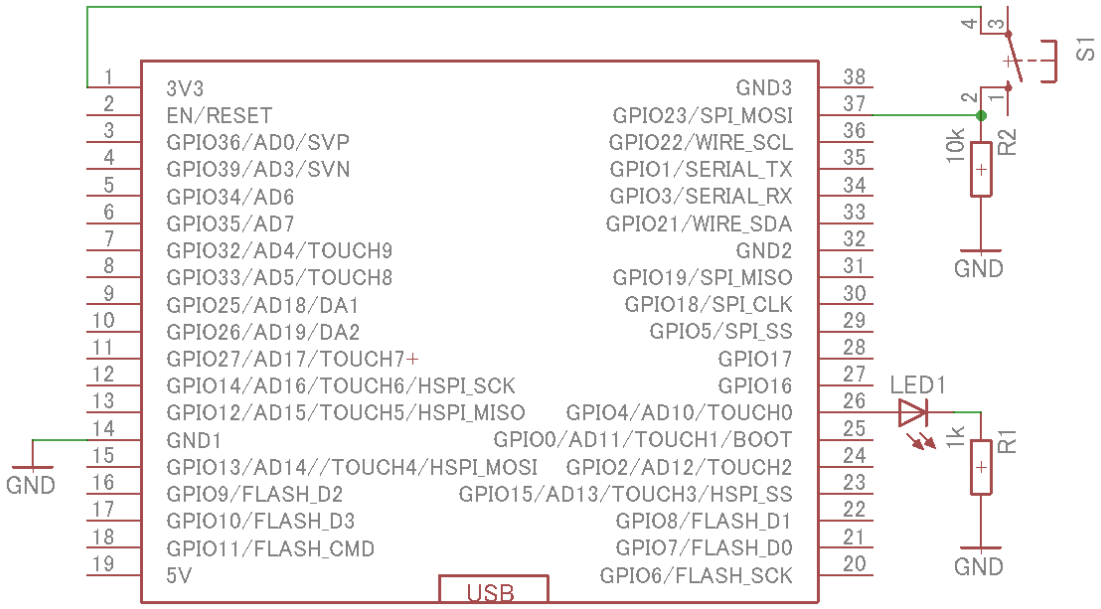
参考：http://gammon.com.au/interrupts

この際，スイッチの「チャタリング」によって点滅が予想通りに動作しないと思われる．「チャタリング防止」でしらべ，コンデンサによる方法またはソフトウエアによる方法を試してみる．

[課題5]　スイッチのステータスをシリアル通信で PC に伝えるようにする．例えばスイッチが押されている間は"a"を，押されていない間は"b"を送信し続けるなど．PC 側はシリアル通信用ソフトを用いる（RealTerm，TeraTerm など．Arduino IDE のモニタ機能を用いても良い）

RealTerm http://realterm.sourceforge.net/

TeraTerm http://sourceforge.jp/projects/ttssh2/



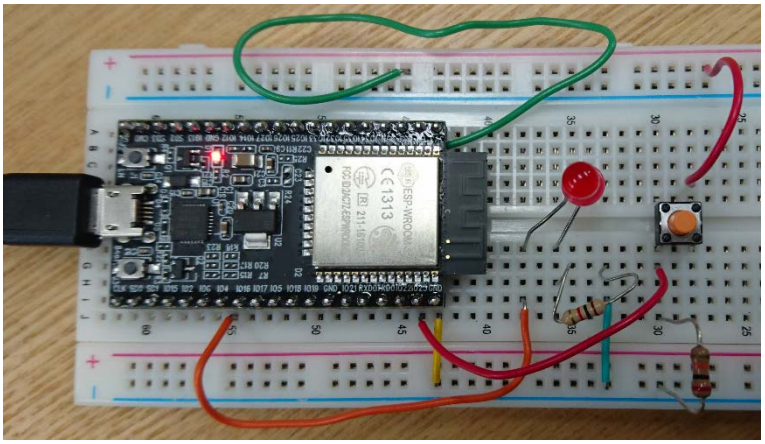
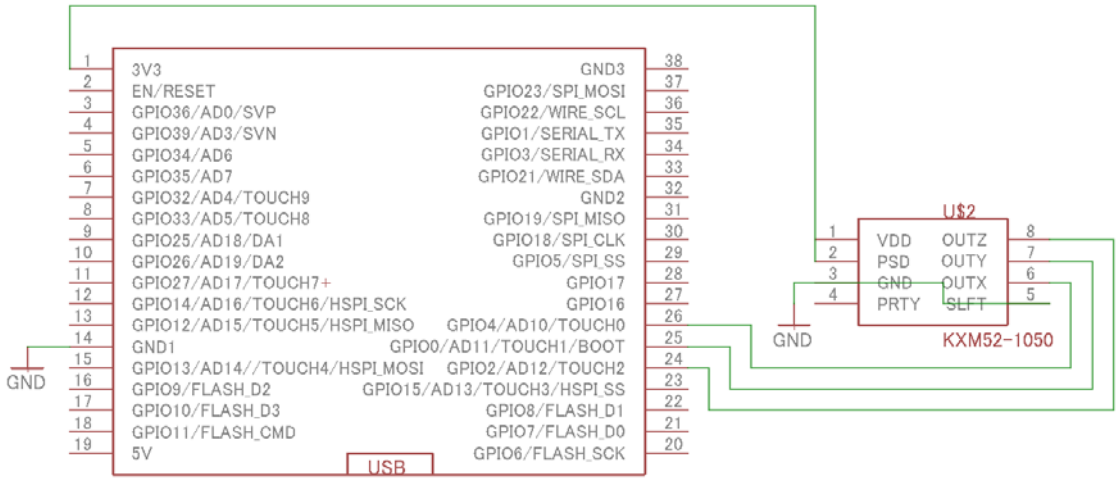


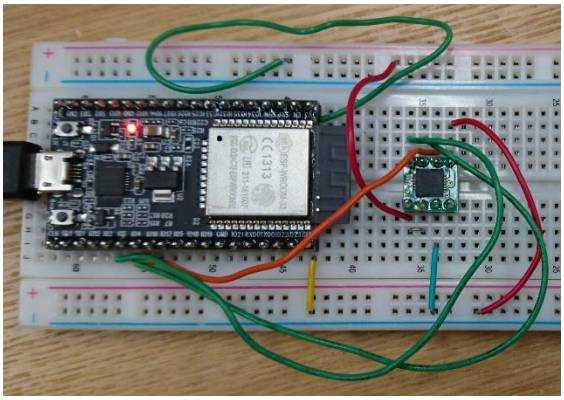
図 3　タクトスイッチ回路部分

4　加速度センサ

KXR94-2050 モジュール（http://akizukidenshi.com/catalog/g/gM-05153/ ）を用いる．このモジュールは 3 軸加速度をアナログ出力するので，3 つの AD ポートで受け取れば良い．関数は analogRead．関数の引数は AD のチャンネル番号(0-19 番)ではなく GPIO の番号となる．例えば AD10 は 10 番ではなく 4 番となる（これは DA や SPI 等でも同様）．
この加速度センサは 3.3V 電源に接続した場合，オフセット 1.65V，1G 加わった際に 0.66V の変化がある．つまり重力加速度の範囲では大体 0.99V～2.31V 出力が変化することになる．
ESP32 の AD 変換はデフォルトで 0～3.6V を 0～4095(12bit)に割り当てて出力する．これは通常の Arduino 出力(10bit)とは異なるので注意．
ESP32 は AD 変換に関して幾つかの制約があり，例えば WiFi 機能を使用する場合に使用できなくなるチャンネルが有る．また attenuation 機能によって測定電圧レンジを変えることも出来る．http://trac.switch-science.com/wiki/esp32_tips

[課題6]　PC で読み取った値が妥当であることを確認する．重力加速度が存在するため，ある軸のセンサ値は，ある向きで+1G，逆向きで-1G となるはずである．実際にどのような値で変化しているか．それは妥当か．





```
int ax,ay,az;

void setup() {
  Serial.begin(9600);
}

void loop() {
  ax = analogRead(4);
  ay = analogRead(0);
  az = analogRead(2);
  Serial.print(ax, DEC);  Serial.print(" ");
  Serial.print(ay, DEC);  Serial.print(" ");
  Serial.println(az, DEC);
  delay(500);
}
```

図 4　加速度センサ回路とソースサンプル

5　RC サーボモータ駆動（PWM）

PWM 出力を利用して RC（ラジコン）サーボモータを動かす.
ここから基板に配線していく.RC サーボモータの駆動には大電流が必要でこれまでの USB 給電では限界があるため外部電源（5V）をつなげ，RC サーボの電源はここから取る.

- 配線は錫メッキ線，あるいはジュンフロン線を用いる．無用に太い線は使わない．ただしモータを駆動する部分や電源部分には太い線を用いる.
  - ハンダ付けの方法は下記リンクや「電子工作の素」などを参照する.
  - ハンダ付けの方法　http://www.youtube.com/watch?v=S5f7jueQHr8
  - 良いハンダ付け形状など　http://homepage1.nifty.com/x6/elecmake/solder.htm

PWM および RC サーボモータの制御方法は例えば下記で自習.

- https://toshiba.semicon-storage.com/jp/design-support/e-learning/brushless_motor/chap3/1274512.html
- http://berry.sakura.ne.jp/technics/servo_control_p1.html

ESP32 における RC サーボの扱いは下記などを参照．Arduino デフォルトの Servo ライブラリは 2017 年 11 月の時点では対応していない.

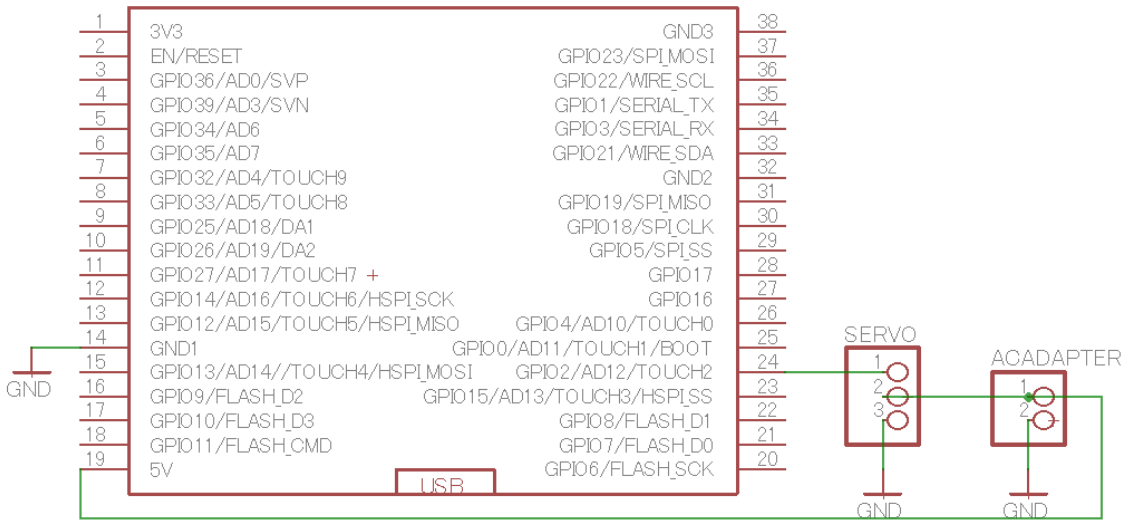
- http://www.instructables.com/id/Interfacing-Servo-Motor-With-ESP32/
- https://hackaday.com/2016/10/31/whats-new-esp-32-testing-the-arduino-esp32-library/
- https://github.com/RoboticsBrno/ESP32-Arduino-Servo-Library

今回は LED を PWM 駆動するためのライブラリを転用する．まず図 5 のソースの動作をオシロスコープで確認する.
http://www.instructables.com/id/Interfacing-Servo-Motor-With-ESP32/

```
//http://www.instructables.com/id/Interfacing-Servo-Motor-With-ESP32/ を一部改変
#define TIMER_WIDTH 16  //16bitで指定する
#define COUNT_LOW 0
#define COUNT_HIGH 6554 //16bitで6554は 20ms * 6554/65536=2.0msに相当

#include "esp32-hal-ledc.h"

void setup() {
   ledcSetup(1, 50, 16); // channel 1, 50 Hz, 16-bit width
   ledcAttachPin(2, 1);   // GPIO 2 assigned to channel 1
}

void loop() {
   for (int i=COUNT_LOW ; i < COUNT_HIGH ; i=i+100)
   {
      ledcWrite(1, i);       // sweep servo 1
      delay(50);
   }
}
```

図 5 RC サーボ用サンプルソース

次に図 6 に示すように RC サーボモータと電源を接続する．この回路図では AC アダプタの 5V 出力を ESP32 の 5V 入力端子に繋いでいる．これによって USB 給電しなくても動作するようになる． AC アダプタの GND と ESP32 の GND の接続を忘れないこと．



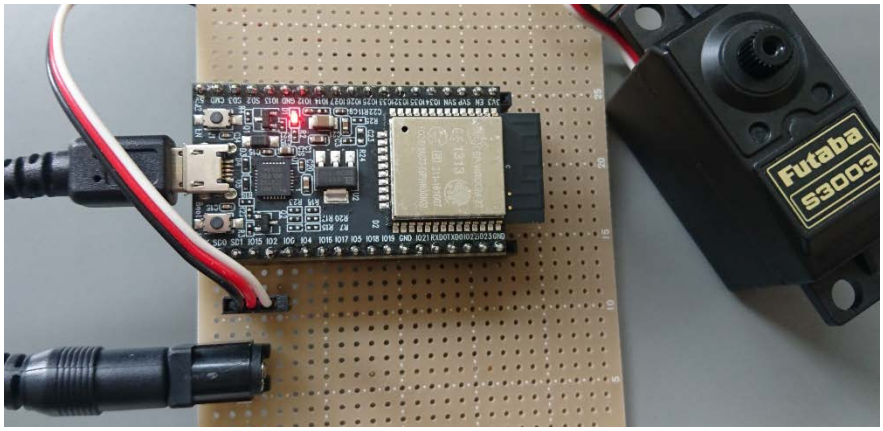


図 6 RC サーボ回路

[課題7] PC からのシリアル通信でサーボモータの姿勢を制御する．例えば 0-9 の数値を送る．PC 側はシリアル通信ソフトで良い．オシロスコープで PWM 信号を観察する．パルス幅，パルス間隔は計算どおりになっているか（以降オシロスコープで観察と書いてある場合はその画面をオシロの機能でキャプチャし，レポートに添付すること）

シリアル通信で PC からの入力を待つためには available 関数を用いる．典型的には図 7 のようになる．

```
//http://www.instructables.com/id/Interfacing-Servo-Motor-With-ESP32/ を一部改変
#include "esp32-hal-ledc.h"
#define TIMER_WIDTH 16  //16bitで指定する
#define COUNT_LOW 0
#define COUNT_HIGH 6554 //16bitで6554は 20ms * 6554/65536=2.0msに相当

void setup() {
   ledcSetup(1, 50, 16); // channel 1, 50 Hz, 16-bit width
   ledcAttachPin(2, 1);   // GPIO 2 assigned to channel 1
   Serial.begin(9600);
}

void loop() {
  int incomingByte;

  if(Serial.available()>0){
    incomingByte = Serial.read();
    ここで送られてきたキーに応じた処理
  }
  delay(15);
}
```

図 7 RC サーボとシリアル通信

6　アナログ出力（＋オペアンプ）

ESP32 は 8bit の DA 変換器を 2ch 持っている．通常の Arduino 環境にはアナログ出力として analogWrite 関数が用意されているがこれは PWM による擬似的なアナログ出力である．これに対して ESP32 は実際に DA 変換出力を行うことができる．

[課題8]　図 8 のように dacWrite 関数を使って DA 出力を行い，出力波形をオシロスコープで確認する．この例では GPIO25（DAC1）から出力される．どのような波形が出力されるか．また波形の周期および周波数と，そこからわかる loop の周期を求めよ．

```
int i=0;

void setup() {
}

void loop() {
  i=(i+1)%255;
  dacWrite(25, i);
}
```

図 8　dacWrite 関数によるアナログ出力

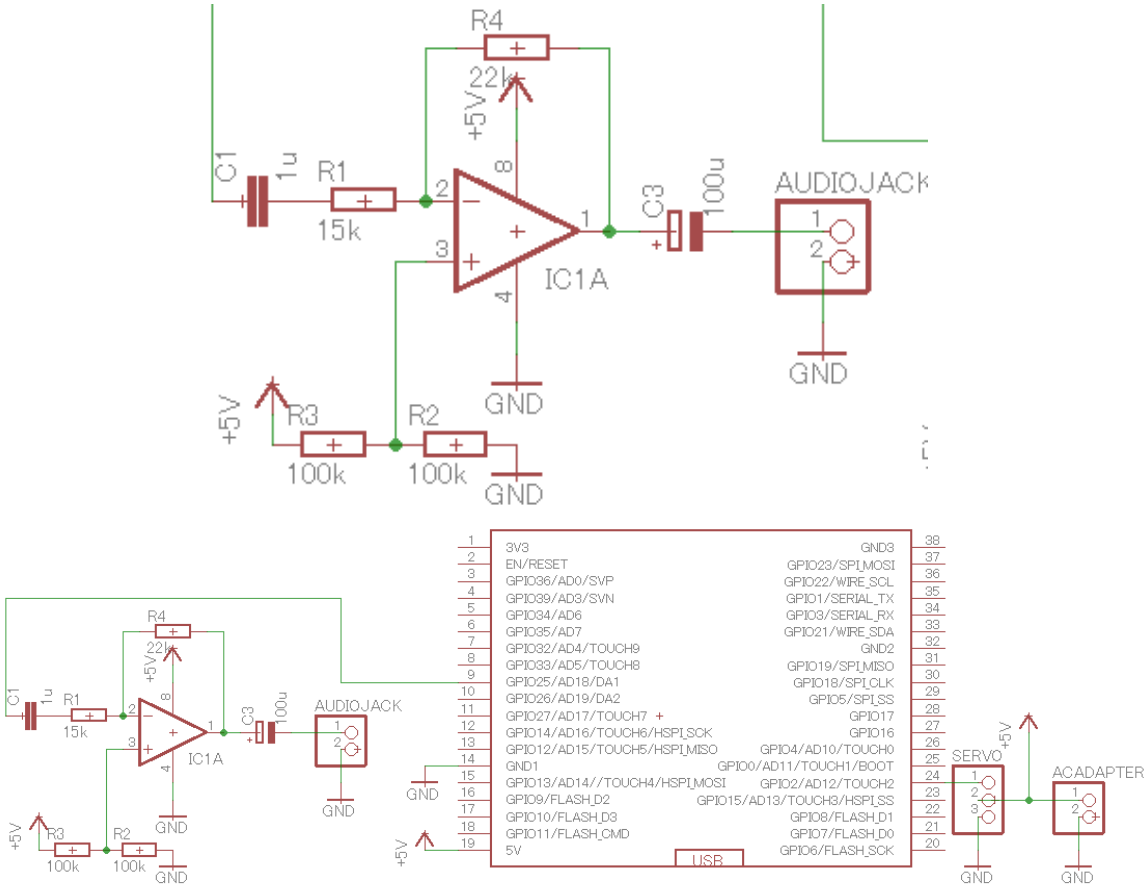
オペアンプ回路の練習としてイヤフォンを駆動する（図 9）．この回路では次のことが行われている．

① DA からの入力信号を C1 と R1 によるハイパスフィルタで直流成分カット．
② オペアンプ（新日本無線製 NJM4580）の反転増幅回路で増幅．増幅率は R1，R4 で指定．
③ 増幅時のオフセット電圧を R2 と R3 で設定し，オペアンプの V+に入力．
④ C3 とイヤフォンの抵抗によるハイパスフィルタで出力の直流成分カット．
⑤ イヤフォンジャックからイヤフォンやスピーカに出力．

- 電解コンデンサの極性に注意．＋側をオペアンプの出力側とする（これはなぜか）
- イヤフォンジャックは基板に直付けできるタイプを用いる．ステレオジャックの場合極性に注意．
- コンデンサの数値コードは読めるようにする．
  http://www.jarl.org/Japanese/7_Technical/lib1/konden.htm

オペアンプ回路については下記などで自習する．
http://picavr.uunyan.com/op_amp.html
http://kaji-lab.jp/ja/index.php?plugin=attach&pcmd=open&file=OpAmpIntro.pdf&refer=people%2Fkaji

なお ESP32 の DAC からの出力で直接イヤフォンを鳴らすことも可能であり，それだけが目的であればオペアンプ回路は不要.



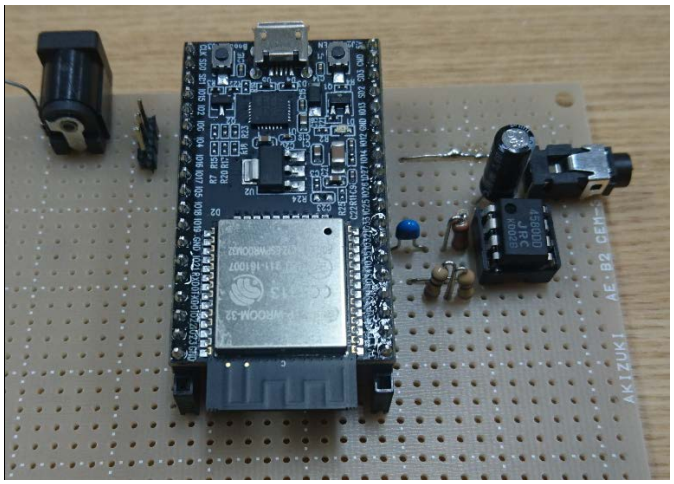


図 9　アナログ出力とオペアンプ回路

[課題9] 図 9 に示した抵抗値，コンデンサ容量のとき，この反転増幅器の増幅率はいくつか．これによって 0～3.3V の電圧入力はどのようになると予想されるか．C1 と R1 によるハイパスフィルタによるカットオフ周波数はいくつか（例えば

http://sim.okawa-denshi.jp/CRhikeisan.htm 利用)．イヤフォンの抵抗値が 32Ωのとき，C3 とイヤフォンによるハイパスフィルタによるカットオフ周波数はいくつか．

なお**このような典型的なオーディオ回路は，イヤフォン，スピーカなどの負荷が接続されてはじめてハイパスフィルタの特性が決定されることに注意．**負荷を接続せずに計測して波形が想定と異なるのは典型的な誤り．

[課題10]　実際に回路を作成し，[課題 8] の入力波形に対してイヤフォンを取り付け聞いてみる（高価なイヤフォンは使わないこと！）．またオシロスコープで出力波形を確認する．このとき波形は DA ポートの波形とは異なっているはずである．どのように異なっているか，またなぜか（ヒント：増幅率の正負，非 Rail-to-Rail）

[課題11]　PC からのシリアル通信でイヤフォンの音を制御する．1-8 の数値を送ると「ドレミファソラシド」が鳴るようにする．PC 側はシリアル通信用ソフトで良い．[課題 7] のソースを参考にすれば良い．ここでは**正確な周波数の波を実現するため，micros 関数を使う．** micros 関数はプログラム起動時から現在までの時間をマイクロ秒単位で返す関数であり，これを 1/1000000 すれば現在の時刻が秒単位で得られる．例えば f[Hz] の周波数の音を出したい場合，現在の時刻が t[s]であれば，$\sin(2\pi ft)$の波を出力すれば良い．正弦波を出力するサンプルを図 10 に示す．オシロスコープで波形の変化を確認する．詳細に観察すると階段状の波形が観察されるはずである．ここから loop 関数の一周期にかかっている時間を求めよ．

```
#define PI 3.141592653589793

unsigned long time;
float freq = 1000.0;

void setup() {
}

void loop() {
  time=micros();
  dacWrite(25, (int)((1.0+sin(2.0*PI* freq * time/1000000.0))*127.0));
}
```

図 10　micros 関数を用いた正弦波出力

7　外付けの DA（SPI 通信によるポートの拡張）

ESP32 には SPI 通信モジュールが複数用意されている．これを使うことでポートを拡張することが出来る．ここでは 8ch の 10bit D/A ポートを実現する．
SPI 通信については下記で自習．シフトレジスタを理解していることが前提．
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

外付けの DA 変換器として 8ch10bit D/A コンバータ LTC1660CN を用いる（図 11）．**この演習はデータシートをきちんと読む練習を兼ねているので手元にデータシートを用意．**
DA 変換器の電源とグランドの間に電圧安定化のためのコンデンサを入れる．10pin 端子を使い，余った 2pin はグランドと 5V にしている．
http://akizukidenshi.com/catalog/g/gI-02794/

ESP32 には SPI，VSPI，HSPI の 3 種類の SPI 通信モジュールが搭載されている．このうち SPI は内部フラッシュメモリとの通信用に使われ，Arduino 環境ではのこりのどちらか(VSPI，HSPI)を使うことになる．このライブラリは特に更新頻度が高いようである（2017.12 現在）
http://trac.switch-science.com/wiki/esp32_tips
https://www.mgo-tec.com/blog-entry-esp32-oled-ssd1331.html
https://www.mgo-tec.com/blog-entry-esp32-spimode-hspi-vspi-hispeed.html
https://github.com/espressif/arduino-esp32/commit/3198f25c19105ff933da4fdb33f132304bb3afaf

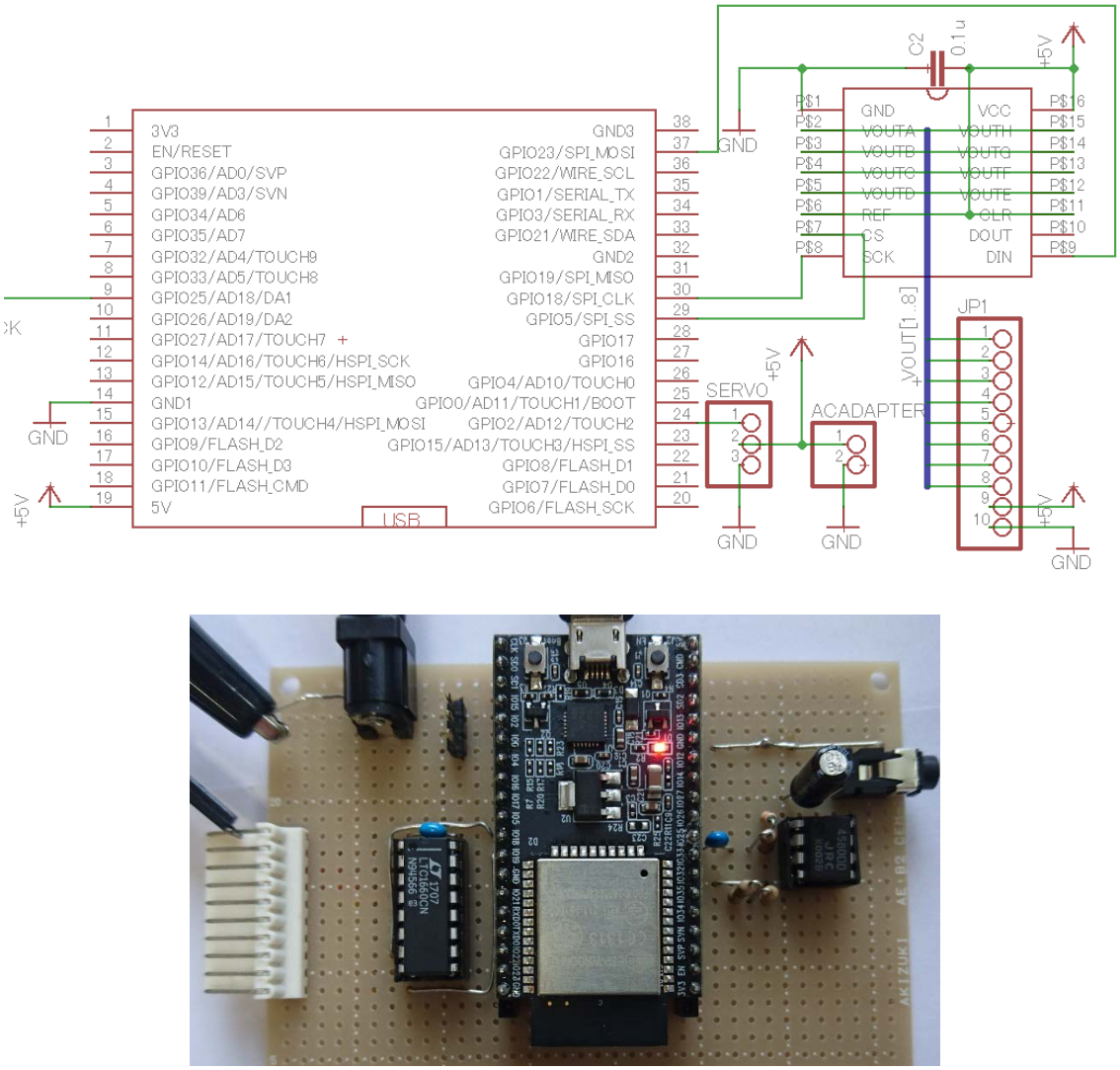


図 11　8ch DA ポート回路（10pin の端子を使い余った 2pin はグランドと 5V にしている）

[課題12]　　下記のサンプルプログラムを動かし，オシロスコープを使って波形を観察する．つぎにクロック信号とデータ信号を同時に観察する（CS ピンをトリガとする．**オシロスコープのトリガの使い方の練習も兼ねている**．２つのプローブを使い，CS ピンをトリガ信号とする． http://download.tek.com/document/55Z-17291-3.pdf http://download.tek.com/document/3GZ-24924-0.pdf ）．

二つの信号のタイミングはどのような関係にあるか．それらは LTC1660 のデータシートのタイミングチャートと比較して妥当か．

16bit のデータ信号はプログラムとどのような関係にあるか．またそれらは LTC1660 のデータシートと比較して妥当か．

```
#include <SPI.h>
// VSPI用設定
#define SCLK 18
#define MOSI 23
#define MISO 19
#define CS 5

short spiData,DA;
char channel = 0;
int t=0;

void setup() {
  Serial.begin(9600);
  SPI.begin(SCLK, MISO, MOSI, CS);
  SPI.setFrequency(1000000);
  SPI.setDataMode(SPI_MODE0);
  SPI.setHwCs(true);
}

void loop() {
  t = (t+1)%2;
  if(t==0){
    DA = 0x3FF;
  }else{
    DA = 0;
  }
  spiData = (((channel&0x07)+1)<<12)|((DA&0x3FF)<<2);
  SPI.transfer16(spiData);
}
```

図 12 8ch D/A 変換器 LTC1660 を動かすサンプル.

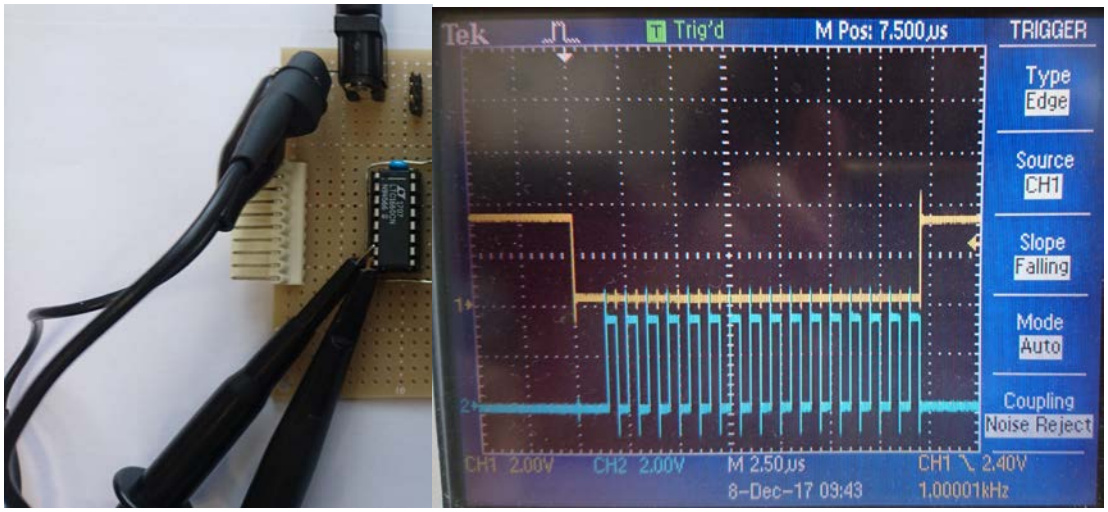


図 13 制御信号の観察

[課題13]　　チャンネルを指定して電圧を出力できる関数を作成する.

void DAout(char channel, float voltage)

LTC1660 のデータシートを見ながらチャンネル指定の方法を考える．電圧を 0.0 から 5.0 までの float で指定出来るようにする．その上で，8ch それぞれから指定した異なる周波数の正弦波を出力し，オシロスコープで波形を観察する．正確な周波数で出力するために[課題 11] の方法を用いる．
ただし 8ch 分の sin 関数をリアルタイムに計算するには時間がかかり loop 周期が遅くなるかもしれないしならないかもしれない．この問題を解決するためにはあらかじめ setup 関数中で出力データのテーブル（配列）を作成しておく．

8　DC モータの制御

MAXON 社製 DC モータ(RE25, 10W, 118746)を駆動する．使用する DC モータにはエンコーダ(HEDS5540，500CPR (CPR: count per revolution))が付いており，位置を計測しながら制御する事ができる．出力には H ブリッジ IC の BD6222HFP を使用する．

エンコーダについては例えば下記で自習する．特に 4 逓倍モードについて理解する．

http://ednjapan.com/edn/articles/1203/16/news012_2.html

http://www.geocities.jp/horie_ryu/page010.html

Arduino 環境のエンコーダについては下記に大量のリストがある．

https://playground.arduino.cc/Main/RotaryEncoders

今回は下記のものを用いる．

https://github.com/PaulStoffregen/Encoder

ライブラリは IDE のメニューから，「スケッチ」→「ライブラリをインクルード」→「ライブラリを管理」で表示されるライブラリマネージャで，”Encoder”で検索すれば出てくる．しかし 2017 年 12 月現在, この方法でインストールすると ESP32 ボードをサポートしていないバージョンがインストールされてしまうので，手動で ZIP ダウンロードする．

上記にアクセスし，「Clone or download」で ZIP ダウンロード，その後 Arduino の IDE からインポートする．

https://www.mgo-tec.com/arduino-ide-lib-zip-install

図 14 を参考に 5V，GND,チャンネル A,B を接続する．なお各出力チャンネルは 5V 電源と 3.3k オームの抵抗で接続すること（これを**プルアップ抵抗**という：自習すること）．プルアップ抵抗無しでも動くが高速回転の際に不具合を生じることが多い．

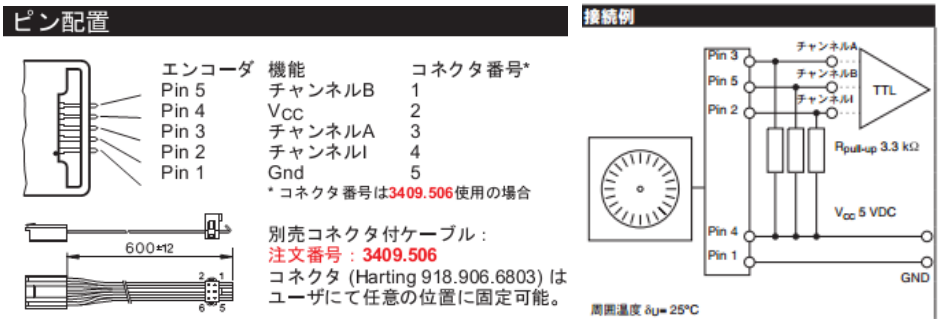


図 14 エンコーダピン配置（HEDS5540 のマニュアルより）

ソースコードは次のようになる(GPIO ピン 21,22 を用いている)．500 パルス／一周のエンコーダを 4 逓倍モードで利用している（一回転何パルスか）．シリアル通信は表示の頻度が多いためこれまでより高速に設定している．

```
/* Encoder Library - Basic Example
 * http://www.pjrc.com/teensy/td_libs_Encoder.html
 *
 * This example code is in the public domain.
 */

#include <Encoder.h>

Encoder myEnc(21, 22);

void setup() {
  Serial.begin(921600);
}

long oldPosition  = 0;

void loop() {
  long newPosition = myEnc.read();
  if (newPosition != oldPosition) {
    oldPosition = newPosition;
    Serial.println(newPosition);
  }
}
```

図 15　エンコーダを読むサンプルプログラム

[課題14]　　エンコーダを用い，モータの回転角度を計測，シリアル通信によって PC 画面に表示し続けるようにする．角度はラジアンに変換して表示する．正転・逆転方向に一周させて±2πの数値が表示されるか確認する．

（参考）マイコンによるエンコーダ値計測は高速な回転になるとカウントの取りこぼしが発生し，ずれを生じることがある．ESP32 でどの程度生じるか未検証であるが，もし生じる場合はカウント専用 IC(LS7366 等)を用いるなどの対策が望ましい．

次にモータを駆動する．H ブリッジ回路については例えば下記で自習する．
http://www.picfun.com/motor03.html
モータ駆動（H ブリッジ駆動）は第 5 章で扱った PWM 制御により明示的に制御してみる．今回使用する BD6222HFP の足は 1.27mm ピッチなので通常の基板ピッチ（2.54mm）に変換する必要がある（図 16）．ピッチ変換基板を使う際は BD6222 の放熱面が各端子に接触しないようにすること（放熱面自体が GND に接続されている）

図 16　BD6222HFP のピッチ変換

BD6222HFP の内部モジュールと端子名，真理値表は図 17 のとおり． Fin,Rin 番ピンにデジタル入力を加えることで，正転，逆転，ブレーキ，ストップを実現する．
電源と GND の間に 10uF のコンデンサを挟む．コンデンサは電源電圧の 2 倍以上の耐圧のあるものを選ぶこと．
後述するようにモータ駆動用の電源は ESP32 の 5V 電源とは別に用意する．GND 同士の接続を忘れないこと．

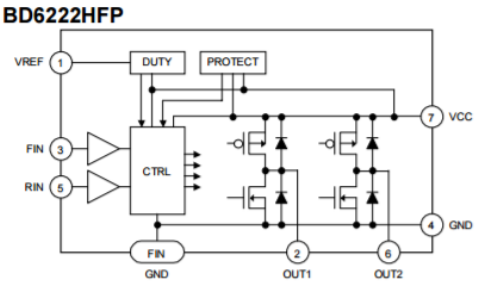


Fig.3 BD6222HFP

Table 2　BD6222HFP

番号	端子名	機能
1	VREF	VREF 可変電圧入力
2	OUT1	出力端子
3	FIN	制御入力(正)
4	GND	GND
5	RIN	制御入力(逆)
6	OUT2	出力端子
7	VCC	電源
FIN	GND	GND

Table 5　真理値表

	FIN	RIN	VREF	OUT1	OUT2	動作(OPERATION)
a	L	L	X	Hi-Z*	Hi-Z*	スタンバイ(空転)
b	H	L	VCC	H	L	正転(OUT1→OUT2)
c	L	H	VCC	L	H	逆転(OUT2→OUT1)
d	H	H	X	L	L	ブレーキ(停止)
e	PWM	L	VCC	H	$\overline{\text{PWM}}$	正転(PWM 制御 A)
f	L	PWM	VCC	$\overline{\text{PWM}}$	H	逆転(PWM 制御 A)
g	H	PWM	VCC	$\overline{\text{PWM}}$	L	正転(PWM 制御 B)
h	PWM	H	VCC	L	$\overline{\text{PWM}}$	逆転(PWM 制御 B)
i	H	L	Option	H	$\overline{\text{PWM}}$	正転(VREF 制御)
j	L	H	Option	$\overline{\text{PWM}}$	H	逆転(VREF 制御)

*Hi-Z とは、出力トランジスタが OFF の状態です。メカ・リレーとは異なり、ダイオードが接続された状態になっていますので、ご注意ください。
X : Don't care

図 17　内部モジュールと端子名，真理値表(BD6222 マニュアルより)

モータを駆動する電源は実験用の安定化電源を用いる．この電源は電圧の設定，および電流のリミットの設定が可能である(**CVCC 電源**: Constant Voltage Constant Current power supply)．一般的な働きとして，制限電流未満のときは定電圧電源として働き，それを上回ったときは定電流電源になる．これにより電流のリミットを低めに設定しておくことでショート等による回路の損傷が防げる．また電流リミット機能を積極的に使うことで，定電流源として使う場合もある．

本実験では当初は電源電圧 12V，電流リミット 0.3A 程度に設定する．その後回路上の問題がなければ電流リミットを 1.5A 程度に上げる．

CVCC 電源については例えば下記で自習する．

http://okwave.jp/qa4105361.html

http://www.kikusui.co.jp/knowledgeplaza/powersupply1/powersupply1_j.html

図 18 のサンプルでモータがたしかに動くことを確認する．

```
int t=0;

void setup() {
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);
}

void loop() {
  t=(t+1)%2;
  if(t==0){
    digitalWrite(32, 0);
    digitalWrite(33, 1);
  }else{
    digitalWrite(32, 1);
    digitalWrite(33, 0);
  }
  delay(1000);
}
```

図 18　モータドライバを動かすサンプルプログラム

[課題15]　　H ブリッジ回路におけるストップとブレーキの違いを原理を含めて説明せよ．

図 17 のファンクションテーブルと，Fin と Rin に PWM 端子をつなげる事とを合わせて考えると，図 19 のようなセットアップおよび関数で出力を制御できると考えられる．

```
void motor(float p)
{
  if(p>0){
    ledcWrite(1, 0);
    ledcWrite(2, (int)(p*65535.0));
  }else{
    ledcWrite(2, 0);
    ledcWrite(1, (int)(-p*65535.0));
  }

}
void setup() {
  ledcSetup(1, 20000, 16); // channel 1, 20kHz, 16-bit width
  ledcAttachPin(32, 1);   // GPIO 32 assigned to channel 1
  ledcSetup(2, 20000, 16); // channel 2, 20kHz, 16-bit width
  ledcAttachPin(33, 2);   // GPIO 33 assigned to channel 2
}
```

図 19　H ブリッジ回路を PWM 駆動するサンプルプログラム（一部）

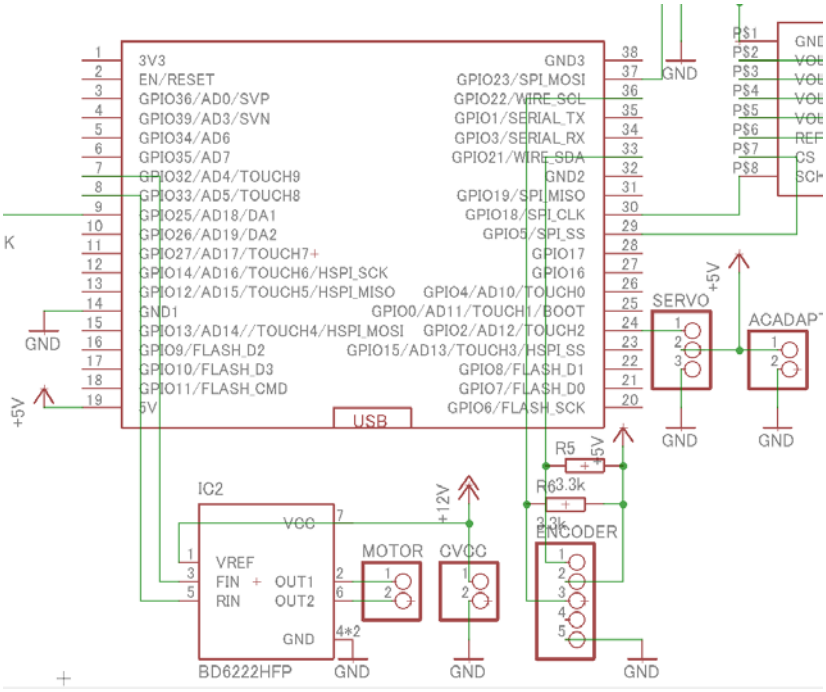


図 20　エンコーダ読み取り回路および H ブリッジ回路

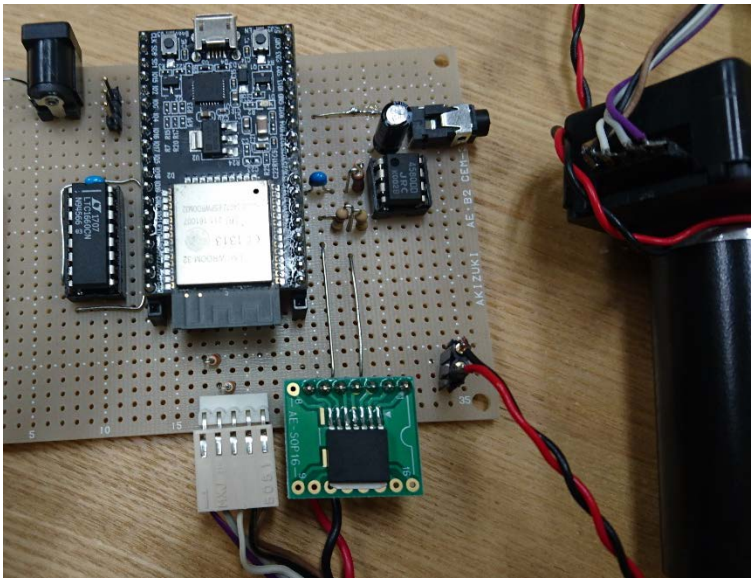


図 21　エンコーダ読み取り回路および H ブリッジ回路写真

[課題16]　　シリアル通信で指令を送り，例えば f(orward)，b(ackward)を入力すると正転／逆転が切り替わり，h(igh)，l(ow)を入力するたびに速度が変更されるようにせよ．PWM の制御周波数を 50Hz，1kHz，20kHz と変化させ，振る舞いの変化，モータから発する音を観察し，考察せよ．20kHz に設定し，２つの PWM 出力が予想通りの出力になっているかどうかをオシロスコープで 2ch 同時に計測して確認せよ．この後は 20kHz の設定で用いる．

[課題17]　　一方向に回転させ続けている時，電流はどの程度流れているか．またこのモータの回転を手で止めた時（気をつける！），電流はどの程度流れるか．この違いはなぜ生じるか調べ，説明せよ（キーワード：DC モータ，逆起電力）．

[課題18]　　アルミ加工により，DC モータの軸に取手をつける．（この部分は下記に従い個別に指導する）

アルミ加工の講習内容

1. ケガキの仕方
    - ①　ノギスの使い方
    - ②　ハイトゲージの使い方
    - ③　ポンチの打ち方
2. バンドソーの使い方
3. ドリルの使い方
    - ①　台座の緩め方，上下させ方

② ドリルの交換方法

③ 万力による材料の固定

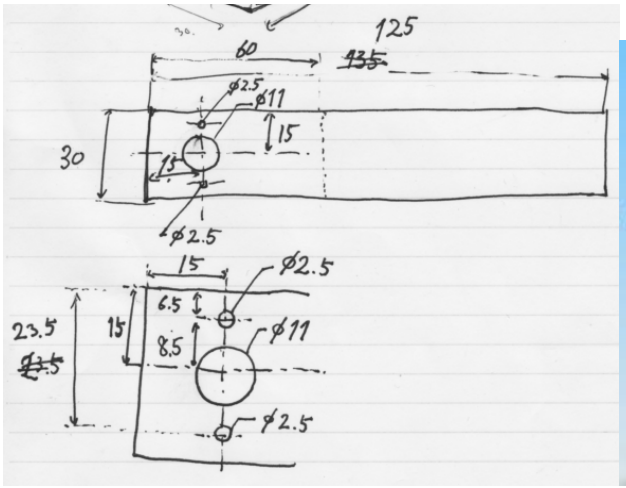
④ 穴あけ，小さい穴に関するバリとり

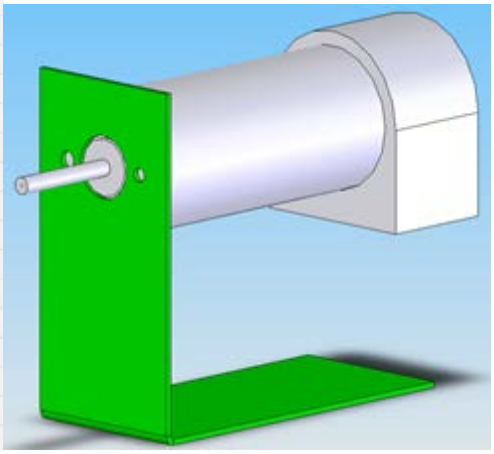
⑤ 特殊ドリル１：ステップドリルの使い方

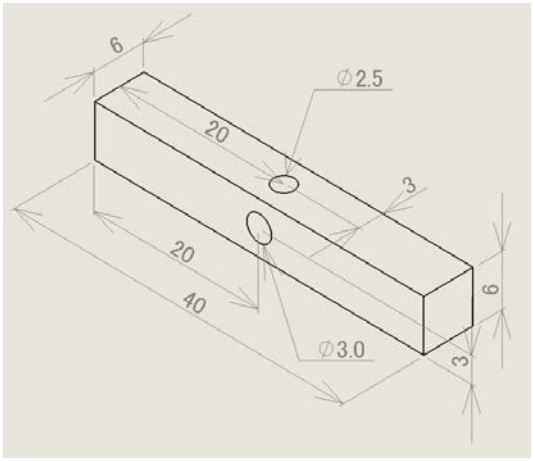
⑥ 特殊ドリル２：バリ取り用ドリルの使い方

4. タップの切り方

5. 後片付け：掃除の仕方（工具周りを刷毛で．床を掃除機で）







これ以降はアルミ台座を机にガムテープで固定する（図 22）.

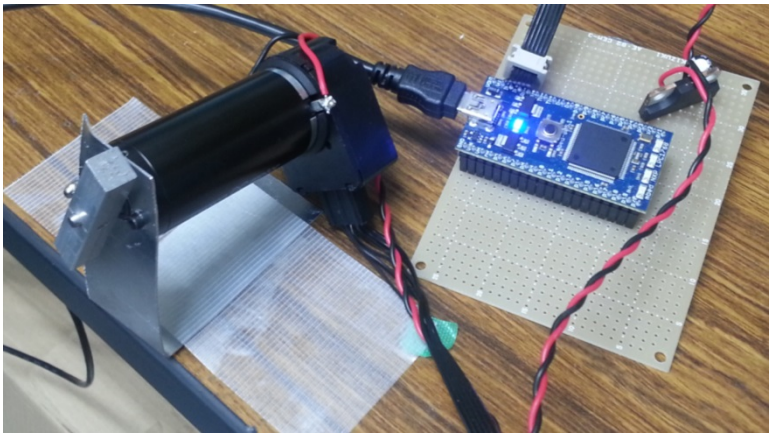


図 22　ノブ付きモータと制御基板

[課題19]　　P 制御により目的の位置（初期位置で良い）に向かうようにし，電源を付け，取手を手で回転させてみる．どうなるか．ゲインを上げ，なるべく早く戻るようにする．

[課題20]　　PD 制御により目的の位置（初期位置で良い）に向かうようにし，電源を付け，取手を手で回転させてみる．どうなるか．また PC から位置指令を送り（例えば r(ight), l(eft)），キーボードを押すたびに移動するようにせよ．

[課題21]　　DC モータの制御により「カチカチ感」のあるロータリースイッチを擬似的に再現する．カチカチの正体を周期的な抵抗感とすれば，これは PD 制御の目標位置を周期的に設定することで実現できると考えられる．

9　PC プログラムによるデータロギング・Processing の導入

4 章で加速度センサの 3 軸の値を読み取る方法を練習した．本章では PC プログラムを用いたデータロギングの方法を考える．

ここからは PC 側のプログラミング環境として Processing を用いる（https://processing.org/）．研究室の実験環境でも多用するため，使用したことがなければ自習したうえで以下の課題に進むこと．4 章の回路を基板上に作成してから進める．加速度センサへの電源は ESP32 の 3.3V 電源を用いる．

[課題22]　図 23，図 24 のサンプルプログラムを写経して ESP32 に接続された加速度センサの電圧をシリアル通信にて PC で表示する．

その後，Processing のプログラムを変更し，データを CSV ファイル形式で保存する．PrintWriter クラスを使用する．Processing のヘルプを参照．キーボード関係の関数も調べ，どれかキーを押したら記録を開始し，100 回記録したら保存して終了するようにする．

```
int ax,ay,az;

void setup() {
  Serial.begin(921600);
}

void loop() {
  ax = analogRead(4);
  ay = analogRead(0);
  az = analogRead(2);
  Serial.write(ax>>8);
  Serial.write(ax&0xFF);
  Serial.write(ay>>8);
  Serial.write(ay&0xFF);
  Serial.write(az>>8);
  Serial.write(az&0xFF);
  delay(20);//wait for 20ms
}
```

図 23　加速度センササンプルプログラム（ESP32 側）

```
import processing.serial.*;

Serial myPort;
String COM_PORT="COM36"; //COM番号. 変更する
int time=0;
int ax=0,ay=0,az=0;

void setup() {  // setup() runs once
  size(256, 256);
  frameRate(60);
  //シリアルポートに接続. 速度は921.6kbpsに設定
  myPort = new Serial(this, COM_PORT, 921600);
}

void draw() {  // draw() loops forever, until stopped
  color cx = color(255,0,0);
  color cy = color(0, 255,0);
  color cz = color(0, 0, 255);

  //シリアル通信
  if(myPort.available()>=6){
    ax = myPort.read() * 256;
    ax = ax + myPort.read();
    ay = myPort.read() * 256;
    ay = ay + myPort.read();
    az = myPort.read() * 256;
    az = az + myPort.read();
  }
  //描画
  time = (time+1)%256;
  fill(cx);ellipse(time,ax/16, 5, 5);//x座標
  fill(cy);ellipse(time,ay/16, 5, 5);//y座標
  fill(cz);ellipse(time,az/16, 5, 5);//z座標
  println(ax," ", ay," ",az);
}
```

図 24　加速度センサ値読み出しサンプルプログラム（Processing 側）

上記サンプルでは Processing 中の Draw 関数中でシリアル通信を行った．しかし Draw 関数は通常 60fps で読み出される関数である．より高頻度ないし正確なデータ記録をする場合には限界がある．

Processing のシリアルライブラリはこうした場合に有用な仕組みをもっている．まず serialEvent 関数を定義する．また bufferUntil 関数を用いて，特定の文字が来た時にだけ serialEvent が発生するようにする．これらは関数のヘルプで使い方を調べること．この仕組を使うためには，データの末尾に特定の文字（以後フッタと呼ぶ）を指定する必要があるため，一回あたりの送受信量は 1byte 増える（図 25）．

またこの時，フッタを示す数値がデータ中に偶然現れることを避ける必要がある．今回の AD 変換は 1ch あたり 12bit であるので，この上位，下位 6bit ずつ送信すれば，データはすべて 0-63 の範囲に収まる．フッタの数値を例えば 0xFF(8bit すべて 1)とすれば，データと指定文字が重なる心配はなくなる．

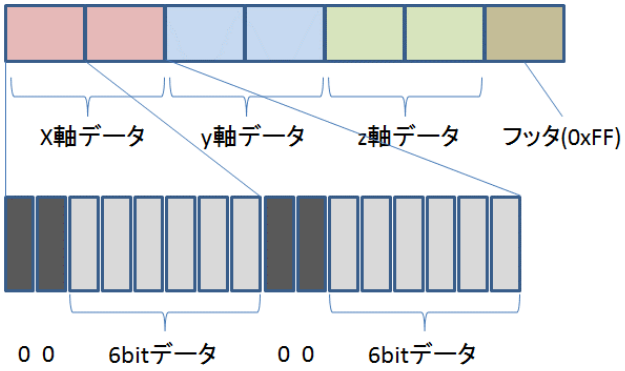


図 25　データ列とフッタの例

複数バイトのデータを読み出すに際して，データの「切れ目」が正しいことを保証する事が重要である． mbed マイコンからのデータは，mbed マイコンを起動するタイミング，PC のプログラムを起動するタイミング，などにより，Y 軸データから始まっているかもしれないし，Z 軸データの２バイト目から始まっている可能性もある．
この問題を解決するためには，第一に PC 側からデータ送信のリクエスト信号を mbed 側に送る方法が考えられる．これが最も標準的だが，図 26 のように PC 側のプログラムでフッタを明示的に探索するという方法もある．これによって初回の読み出し時にデータの切れ目が正しく認識される．

```
30 void serialEvent(Serial p) {
31
32   ax = p.read() * 64; //upper 6bit
33   ax = ax + p.read();
34   ay = p.read() * 64;
35   ay = ay + p.read();
36   az = p.read() * 64;
37   az = az + p.read();
38   while(p.read() != 0xFF);//フッタをチェック.
39
40   if(WriteToFile==true){
41     output.println(ax+","+ay+","+az);
42   }
43 }
```

図 26　シリアルイベント内でのフッタの探索（Processing）

bufferUntil による割り込み関数の定義は，こうした割り込み関数の常として，serialEvent 関数がいつ呼び出されるかわからないために再現性のないバグを生むことがある．例えば serialEvent 関数は，bufferUntil が呼ばれた次の瞬間に（まだ Processing の setup 関数が終わらないうちに）呼ばれるかもしれない．例えばこれによって，まだ書き込むファイルが準備されていない状態でファイル書き込みが生じる等のバグが生じる．こうしたバグは典型的に「何回かに一回」発生する．こうしたバグに対してアドホックな解決は取らないこと．

[課題23]　　加速度センサの３軸の値を 1kHz で記録したい．上記の解説を参考に，ESP32 側から 1kHz（1ms 周期）で AD 変換，シリアル通信を行い，データを記録せよ．ESP 側のデータ取得間隔を正確にするため図 27 を参考にタイマー割り込みを行う． 1ms 周期で計測できていることを確認するため，適当なデジタル出力端子の状態を毎回反転させ，オシロスコープで観察する．

ESP32 は高時間分解能のタイマー割り込みを利用出来る．
http://www.iotsharing.com/2017/06/how-to-use-interrupt-timer-in-arduino-esp32.html
https://esp32.com/viewtopic.php?f=19&t=2481
https://techtutorialsx.com/2017/10/07/esp32-arduino-timer-interrupts/
https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/Timer/RepeatTimer/RepeatTimer.ino

```
//http://www.iotsharing.com/2017/06/how-to-use-interrupt-timer-in-arduino-esp32.html
//hardware timer
hw_timer_t * timer = NULL;

void IRAM_ATTR onTimer(){
  Serial.println("here");
}

void setup() {
  Serial.begin(921600);
  timer = timerBegin(0, 80, true);             //タイマーモジュールの設定．80MHz/80で1tickが1usに対応
  timerAttachInterrupt(timer, &onTimer, true); //呼び出し関数の設定
  timerAlarmWrite(timer, 1000000, true);       //呼出し周期の設定．1秒=1000000
  timerAlarmEnable(timer);                     //タイマー設定
  Serial.println("start timer");
}

void loop() {
}
```

図 27　タイマー割り込みのサンプルプログラム

上記は Processing 側の Draw 関数の周期に依存しない計測の方法だが，そもそもシリアル通信を USB2.0 経由で行なっているため，通信は実際には 500Hz 程度の頻度で行われている．このためマイコンの段階では 1kHz でサンプリングができても，PC での取得時刻は正確な 1kHz ループではない．PC を入れた制御ループを構成する際には問題となる．高速な制御ループはマイコン内で完結させるべきである．
振動子の応答特性計測などではより速い計測が必要となるが，リアルタイム性が必要でない場合には，マイコン上のメモリに保存することもよく行われる．

10 無線通信

今回使用しているボードは WiFi および Bluetooth Low Energy (BLE)用のモジュールを持つ．これらをとりあえず使ってみる．この部分は 2017 年冬現在変化が早い．

10.1　WiFi

https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi

[課題24]　上記ページの SimpleWiFiServer（LED を web ブラウザ経由で点滅制御）を動かしてみる．SSID およびパスワードは研究室の WiFi をとりあえず使用する（実際のデモ等での使用ではモバイルルータやスマートフォンのテザリングを使用）．シリアル通信によって割り振られた IP アドレスが分かるので，そのアドレスに Web ブラウザからアクセスしてみる（図 28）．

その上で Processing からの通信を行う．Processing 側は Network ライブラリの write 関数のサンプルを改変し，特定の文字を送信するようにする．ESP32 側は特定の文字を受信したら LED を点滅させるようにする．

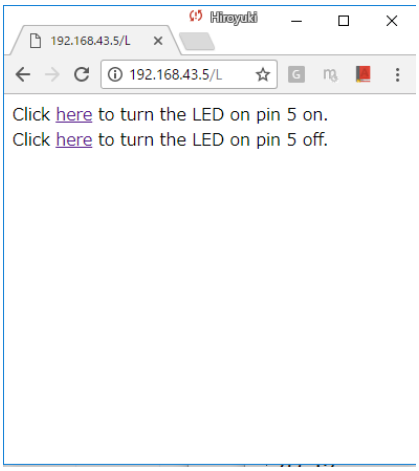


図 28　Web ブラウザによる動作確認

10.2　BLE

下記のライブラリを用いる．8 章で説明した方法でライブラリを Arduino IDE 環境で利用できるようにする（2017 年 12 月現在は zip ダウンロードしてインストール）．

https://github.com/nkolban/ESP32_BLE_Arduino/

ライブラリの examples フォルダーの中に，BLE_write のプロジェクトを試しに書き込む．

PC との通信を行う場合：PC の設定→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→デバイス追加（デバイスが見つからない場合 Bluetooth が有効

になっていることを確認．PC を再起動すると直ることがある）
BLE テスト用アプリとして Bluetooth LE Lab をインストール→デバイスを選択，接続→Unknown Characteristics をクリックして Writing のフォームに入力すると通信できることが確認できる．送信形式は UTF-8.
https://www.microsoft.com/ja-jp/store/p/bluetooth-le-lab/9n6jd37gwzc8

Android との通信を行う場合：BLE テスト用アプリとして nRF Connect アプリをインストールして利用．現時点で特定機種で接続できない現象を確認.

[課題25]　ESP32 に接続した LED を BLE 通信で On/Off 出来るようにする．上記 BLE テスト用プログラムから文字列を送信する.

11 指導者向け TIPS

次の指導が必要です．放置して野生力をつけるという方針もありえますが，この講習に限って前半は集中的に指導し，後半も適切な苦労をさせてください．

- 初日冒頭，2～5 章の説明．
  - 開発環境導入
  - ハンダ付けの方法
  - スイッチ回路の動作の原理
  - LED 回路の抵抗計算の原理（LED の特性），テスタで LED の極性を知る方法
  - RC サーボの駆動の前提となる PWM の説明
  - オシロスコープによる測定（トリガの概念，外部トリガを用いる方法）
- ３日目冒頭，6～7 章の説明
  - オペアンプ入門
  - 圧着端子の使い方説明
  - SPI 入門，シフトレジスタの説明から，LTC1660 のマニュアルを見ながらタイミングチャートと送信データ構造の理解
  - オシロスコープによる測定（外部トリガを使う方法）
- ２週間目冒頭，8 章の説明
  - エンコーダの説明，モータドライバ IC の説明
  - アルミ加工の解説（これは個別に教える）
- ２週間目中頃，9 章の説明
  - Processing の簡単な説明．通信遅延の重要さについて解説．
- ２週間目終盤，10 章の説明
  - WiFi，Bluetooth モジュールの簡単な説明．

この講習は研究室的技能の中心となりますので，集中的に指導しましょう．講習資料は不完全ですので，プログラミング言語のヘルプや IC のマニュアルを隅々まで理解しないと出来ません．**しっかり理解すればできる（あいまいな理解ではできない）という体験に持っていく**ようコントロールしてください．

問題解決（デバッグ）の効率的な手順も指導してください．例えば望ましい値が計測値として出ない場合，最初は必ずフリーズします．これを，まずテスタで確かに電圧が出ていることを確認して「ハードの問題かソフトの問題かを切り分ける」，次に通信ソフトで表示して「通信の問題かそれ以外かを切り分ける」，という**二分岐による可能性探索**を体験させてください．**初めてだとテスタやオシロが開発／デバグに使うものという概念がありませ**

ん（レポート用のデータを取るものという認識）.**「テスタ／オシロで測ってみた？」は最初に聞く**ようにしてください.

プログラムに関しては， if 文，for 文，while 文は理解していても，ある課題に対してどのようなプログラム構造を作るべきかという視点は持っていないのが普通です．**ホワイトボードで擬似的なプログラムを書くという感じで議論**しながら，プログラムの**構造を作っていく**感覚を身につけさせてください．これはフローチャートを書くというのではありません．ホワイトボードでの議論に慣れさせるためにも意図的に使ってください.

最後に**わかっていない YES は問い詰めてください**．講習会の場で追求すれば，それは課題を解くことに直結するので利益が見えやすいです．つまり**わかっていないことをごまかさないことが実際に有益であることを体験できる機会**になります.

以上のように本講習の指導の際は

- プログラムソースとハードウエアマニュアルを完全に理解する必要があるという感覚
- デバグ（二分岐）の概念，テスタやオシロがデバグ用ツールであるという常識
- プロジェクトマネジメントの概念，ソースコードは構造の設計が大切という体験
- わかっていないことをごまかさない習慣

をつけるように促すことが大切になります.

12 参考文献

文中で取り上げたもの以外で参考になるものを挙げます.

資料作成で特に参考にしたリンク

[1] ESP32-DevKitC Getting Started Guide
http://esp-idf.readthedocs.io/en/latest/get-started/get-started-devkitc.html

[2] ESP-WROOM-32 https://ht-deko.com/arduino/esp-wroom-32.html

[3] ESP-WROOM-32 DataSheet
https://www.espressif.com/sites/default/files/documentation/esp-wroom-32_datasheet_en.pdf

[4] ESP-IDF 環境．Arduino 環境とは異なり ESP32 のすべての機能を利用できるようになります． https://www.mgo-tec.com/esp32-idf-howto-01

書籍

[1] 「オペアンプ基礎回路再入門」標準的な教科書.

[2] 「すぐ使える！オペアンプ回路図」

[3] トランジスタ技術 SPECIAL「OP アンプによる実用回路設計」

[4] トランジスタ技術 SPECIAL「OP アンプ IC 活用ノート」

[5] トランジスタ技術 SPECIAL「徹底図解 ディジタル・オシロスコープ活用ノート」CQ 出版の SPECIAL シリーズ. 内容は標準的で，入門を卒業した人の次のステップ用で，長く座右に置けるタイプです． これ以外にも研究室に置いてあるトランジスタ技術 SPECIAL はどれも良い参考書.

[6] トランジスタ技術 研究室で毎月購読しています.

[7] PIC とセンサの電子工作 PIC を使った電子工作本ですが，センサ関係の解説が豊富.

[8] 日経エレクトロニクス 研究室で毎月購読しています.