# 概要

DX；Digital Transformation

デジタル技術の活用して世の中を変える事。

IoT；Internet of Things

なんでもかんでもWebにつなげてしまおう，という概念

実体のあるもののデジタル化とも言える。

# センサ系

## OCR；Optical Character Recognition

### 概要

光学文字認識

### Tesseract

テッセラクト

#### インストール

apt tesseract

参考：https://qiita.com/ku\_a\_i/items/93fdbd75edacb34ec610

AI-OCR

## ARタグ

### 概要

#### ArUco

#### カメレオンコード

参考

[https://www.womnet.com/news/tech/ar%E3%83%9E%E3%83%BC%E3%82%AB%E3%83%BC%E3%81%AE%E4%BD%BF%E3%81%84%E3%81%A9%E3%81%93%E3%82%8D/](https://www.womnet.com/news/tech/arマーカーの使いどころ/)

### OpenCVでaruco

<https://python-academia.com/opencv-aruco/>

<https://www.youtube.com/watch?v=HaQ3yXVyPjY>

計算式？

<https://qiita.com/KYoshiyama/items/3499bae69342b89230f3>

gyro holo

### ArKit

iOSで動かすAr

参考：<https://ken-it.world/it/2021/04/pronopointsscan-advanced.html>

LiDarはアンドロイドでもできるらしい

<https://www.taisei.co.jp/about_us/wn/2021/211227_8627.html>

# IoTゲートウェイ

## OpenBlocks

概要

直接クラウドサービスに接続できる

内部でBIツール、データベース、UPSを持っている

特徴

Modbusプロトコルに対応

OSをDebian, Windows IoT Enterpriseから選べる。

Node-RED対応

# 可視化

## Streamlit

pythonの可視化ライブラリ

JavaScriptはもちろんHTMLやCSSの知識すら不要で、pythonのみでダッシュボードが作成できる

詳細は別ドキュメント「python」

## Plotly Dash

# デジタルサイネージ

メモ

Philipsの大型モニタはUSB内の画像ファイルをスライドショーで表示する機能がある。

リモコン操作するらしい



# スマートホーム

## スマートスピーカー

### Google Home(Nest)

#### 概要

2023-01-07調査中

Google AssistantはWebhookを使う

何かと連動させるためにはWebhookに対応したWebサービスが必要となる

#### 基本

##### VoiceMatch

各個人の声紋特徴を登録しておくだけで、Google Assistantに話しかけた人を特定する機能。

例えば「今日の予定は？」と聞くだけでその人の予定を教えてくれる

登録方法は「OK,Google ねぇ Google」と４回続けてスピーカーに対して話しかける。

登録されているかどうかは「OK,Google　私はだれ？」で確認できる。

注意点

|  |  |  |
| --- | --- | --- |
|  | Conversation Actions | Direct Actions |
| 概要 |  |  |
| 利用対象者 | VoiceMatchを登録した人がいる場合、登録者のみ。登録者がいない場合は全員。 |  |

具体例

Google Keepなど

##### ルーティン

例えば「家に帰ってきた時に行う事」を複数登録しておき、「ただいま」と言う事でそれらを順番に実行してくれる機能。

カスタム ルーティンをスケジュール設定する

公式：<https://support.google.com/googlenest/answer/7029585?co=GENIE.Platform%3DAndroid&hl=ja>

#### 音声を再生する

##### 概要

##### 共通情報

castv2-clientを使うらしい

##### castv2-client

意外と情報が少ない

参考：https://qiita.com/nori-dev-akg/items/751e7d9bf2728afda28a

##### google-home-notify

概要

公式、かつ一番Web上で情報がある方法だが、不安定で色々苦労する為おすすめしない。

ESP32を使った例

https://qiita.com/horihiro/items/4ab0edf415916a2cd542

##### Cloud Text-to-Speech

https://qiita.com/odetarou/items/df5bc6d00494ea14f33f

##### VoiceText

HOYAが提供するText-To-Speechソフトウェア。

Node.jsのライブラリが存在する

npm install voicetext

HOYA公式：<https://cloud.voicetext.jp/webapi>

更に nori-dev-akgという人が作ったgitレポジトリがあるので利用する

参考：<https://qiita.com/nori-dev-akg/items/751e7d9bf2728afda28a#voicetext-web-api>

# リポジトリをクローン

git clone https://github.com/nori-dev-akg/castv2-googlehome

# ディレクトリを移動

cd castv2-googlehome

# npm初期化

npm init --yes

# npm パッケージインストール

npm install voicetext castv2-client express fs

systemdのUnitファイルを用意

[ /etc/systemd/system/castv2-googlehome.service ]（例）

[Unit]

Description=castv2-googlehome

After=syslog.target network.target

[Service]

Type=simple

User=pi

ExecStart=node /home/pi/castv2-googlehome/api.js

WorkingDirectory=/home/pi/castv2-googlehome

KillMode=process

Restart=always

StandardOutput=file:/tmp/castv2-googlehome.log # 必要であれば，標準出力のログ

StandardError=append:/tmp/castv2-googlehome\_error.log # 標準エラー出力

[Install]

WantedBy=multi-user.target

サービスの有効化

sudo systemctl start castv2-googlehome # 開始する

sudo systemctl enable castv2-googlehome # OS起動時にサービスが自動起動するようにする

補足

nodejsのforeverを使う方法もあるが、OS起動時の動きが不安定なので非推奨

# npm パッケージインストール

sudo npm install -g forever

# 運用ではsystemdのサービスで起動するのが良い

forever start api.js

次ページへ続く

～VoiceText　つづき

[ ~/castv2-googlehome/speech.js ]（例：時報）

const Castv2GoogleHome = require('./castv2-googlehome.js');

const rapsberrypi\_ip = '192.168.11.92';

const googlehome\_ip = '192.168.11.100';

const voicetext\_key = '\*\*\*\*\*\*\*\*\*\*\*\*';

const c2gh = new Castv2GoogleHome(rapsberrypi\_ip, googlehome\_ip, voicetext\_key);

var now = new Date();

var hour = now.getHours();

var minuites = now.getMinutes();

if (minuites == 0) {

c2gh.speech(`${hour}時です`, 'haruka');

} else{

c2gh.speech(`${hour}時${minuites}分です`, 'haruka');

}

実行（単発）

[ terminal ]

node ~/castv2-googlehome/speech.js

cronで行う場合

[ crontab -e]（例）

0,15,30,45 6-8 \* \* 1-5 /usr/local/bin/node ~/castv2-googlehome/speech.js # 平日の早朝は15分刻みで通知

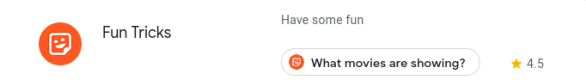
0 9-21 \* \* \* /usr/local/bin/node ~/castv2-googlehome/speech.js # 9~21時は1時間刻みで通知

#### アプリ（プラグイン）

独自の「アプリ」を入れる事でGoogle Assistant をカスタマイズできる。

アプリ選択：<https://assistant.google.com/explore/>

例

Fun Tricks

映画などのイベントを探す

LINE

UberEats

#### Actions on Google

##### 概要

Google アシスタント向けのアクション（対話アプリ）を作成するためのプラットフォーム。

URL：<https://console.actions.google.com/> Google Chromeで開いた方が無難？

チュートリアル？：<https://developers.google.com/assistant/smarthome/develop/create>

チュートリアル？：https://developers.google.cn/assistant/smarthome/overview?hl=ja

注意点：

・使いたいGoogle Homeにログインしているアカウントと同一アカウントにする。

##### テンプレート

Smart Home スマートホーム家電の制御？

Food ordering

Game

Custom

Story telling

Education

##### チュートリアル

###### おおまかな流れ

プロジェクト作成 プロジェクト名などの入力

Overview 最低限の設定項目をまとめたページ

├ Quick setup Display Nameなどの設定

├ Build your Action(s)

Develop 文字通り、開発機能

├ Invocation

├ Actions

├ Account linking

Test

├ Simulator

├ Test suite

###### プロジェクト作成

New Project → プロジェクト名入力 → a languageを Japanese、country or regionをJapanに

→ Display Name入力

###### Overview

Decide how your Action is invoked

Invocation

・Display name

Develop - Invocation - Display name アシスタント上での名前？2023-01-13

・Add Action(s)

Build your first Action → Get started → Custom intentが選択されたままをクリック

→ Dialogflow （自然言語理解プラットフォーム）の画面が表示されるので、Googleアカウントでサインイン

→ CREATE AGENT → DEFAULT LANGUAGEをJapanese - jaにして「CREATE」

左側メニュー「Integrations」から「Google Assistant」を選択

###### Develop

Invocation

・Display name アシスタント上での名前？2023-01-13

Device registration

##### メモ

Develop

Fulfillment URL

Invocation

Invocation



> Project settings

プロジェクト名変更 ：Delete project（下のほう） > ProjectIDを入力

プロジェクト削除

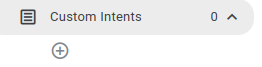
メモ

Display name

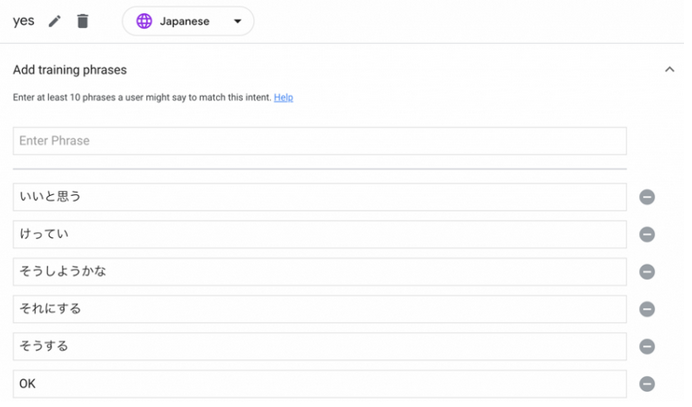
適当な呼び名を指定

Actions directory（公開リポジトリ的な）内で一意である必要がある？

Intentsの編集



Decide how your Action is invoked

Intents

→ Custom Intents の＋をクリック

一つの「単語」を登録するイメージ。

右図の例ではyesという単語として処理されるフレーズを複数登録している。

firebaseを使うと簡単にWebアプリ開発ができる（らしい）

npm install -g firebase-tools@6.5.0

npm install actions-on-google

##### Dialogflow

##### Conversational Actions終了(sunset)関連

代わりに「App Actions」というフレームワークが利用可能に

関連：https://developer.android.com/guide/app-actions/overview

出来ること

・Androidアプリの操作 (App Actions)

・WEBサイト上の構造化データの提供 (Actions from web content)

・スマートホーム操作 (Smart home)

・Media Actions Spotifyの曲送りとか？

参考

<https://qiita.com/H-Kz/items/0efe54d17f31a4500604>

Actions Builder を使え、と言う情報もあるが？

##### 資料

少し情報は古いが：

https://www.moyashi-koubou.com/blog/make\_google\_home\_app\_2018/

Actions on Googleチュートリアル

<https://qiita.com/knoxrd/items/f3044dd447178b94cbf9>

音声操作によるゲーム体験：

https://labs.gree.jp/blog/2022/01/21573/

#### MQTT

公式？

https://www.home-assistant.io/docs/mqtt/

参考

[Google Home + IFTTT + Beebotte(MQTT Broker) + ESP32](https://qiita.com/mascii/items/1db06be0950a47e6c720)でLチカしてみた

https://qiita.com/mascii/items/1db06be0950a47e6c720

#### ESP32対応

結構大変そう

参考：https://qiita.com/poruruba/items/337722ef13cf44986b37

#### 用語

Intent Home デバイスを起動する為の一つの「単語」。

Dialogflow

Converse.AI

### 対応デバイス

#### NatureRemo

#### LinkJapan

#### SmartBot

#### Philips Hue

参考：<https://www.benrilife.com/googlehome-smartremocon/#i-5>

## 電力監視

### スマートメーター

#### 基本モジュール

|  |  |  |  |
| --- | --- | --- | --- |
| メーカー | Rohm （日本) | Rohm （日本) | Rohm （日本) |
| 型番 | BP35C5 | BP35C0 | BP35C2 |
| 価格[円] | 8167 | 4276 | 15800 |
| 無線規格 | Wi-SUN FAN | Wi-SUN Bルート Wi-SUN EHAN | Wi-SUN Bルート Wi-SUN HAN |
| I/F | UART GPIO | UART GPIO | USB2 |
| ANT | 外付 | 外付 | 内蔵 |
| 技適 | ◯ | ◯ | ◯ |
| 寸法[mm] | 19.0 x 15.0 x 2.6 | 15 x 19 x 3 | 49.7 x 21.4 x 8.5 |
| 動作温度[℃] | -30 ～ 85 | -30 ～ 85 | -20 ～ 50 |
| 動作湿度[%] | 40 ～ 60 | 40 ～ 60 | 40 ～ 60 |
| 説明 | ・920MHz ・FCC/ARIB STD-T108 | ・920MHz ・ARIB STD-T108 | ・920MHz ・ARIB STD-T108 |

参考：<https://www.cynaps.jp/iot-tech/wi-sun/#Wi-SUN-4>

BP35A1が最安

公式：<https://www.rohm.co.jp/products/wireless-communication/specified-low-power-radio-modules#parametricSearch>

#### シリアル通信対応

参考：<https://nosuke.net/archives/279>

### 家庭内の電力測定

#### はじめに

プログラム的にも，電子的にもちょっと難しいので初心者がここから入るのはお勧めしない。

以下の項目への理解が必要：

・交流電源への理解（家庭用電源は交流）

・アナログとADコンバーター

（オペアンプによるレベルシフト） ※負電圧に対応していないADコンバーターを使う時

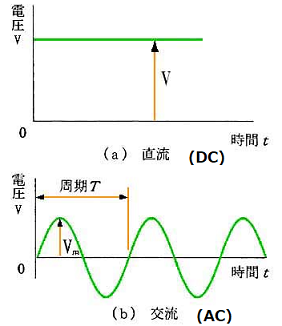
・ラズパイにおける定時間処理

・電力計算

などなど。一つ一つコツコツ進める必要がある。

#### 交流の基礎知識

家庭で使われている電源は100Vの交流。



乾電池などの直流電源（右図上）は常に既定の電圧なので分かりやすい。

反面，交流電源（右図下）は常に電圧が変化していて理解しにくい。

0V → 最大値+141V（※） → 0V → 最低値-141V → 0Vを１サイクル

として，西日本では1秒間に60回(=60Hz)，東日本では1秒間に50回(=50Hz)の周期でプラスとマイナスを行き来している。

家庭内で電力モニタしようとする時は，交流電流を少し理解しておく。

※なぜ最大値は100Vではないのか，と思った人はこちらを参照　<https://towatowa.net/rms-value/>

#### 電力測定に必要な物

|  |  |
| --- | --- |
| 項目 | 備考 |
| クランプ式電流センサ  ￥1000～1500くらい | ・CTセンサが良い（但しCT方式は交流しか測れないらしい） |
| ADコンバーター  ￥250～1000くらい | ・**負電圧の入力に対応しているものを強く推奨**。  ・シリアル通信はI2Cの方が少しだけ楽。  ・複数の電流センサを使いたいならch数にも気をつける。  *補足：使用するマイコンにアナログ入力が有れば必要ない。（ラズパイには無い）* |
| オペアンプ  ￥20～60くらい | ADコンバーターが負電圧に対応していない場合は必要。  単電源のものが良い。 |
| 抵抗  1個≒￥1  但し最小ロットが100個くらい。 | ・10Ωくらいのが1つ。電流センサに使用。金属皮膜の方が良いかも。（1個20円くらいの高精度抵抗）  ・オペアンプを使う場合，10kΩくらいの抵抗5個くらい。こちらは普通のカーボン抵抗で良いと思う。 *※抵抗値はあくまで例で，場合により抵抗値は異なる。* |
| 可変抵抗 | ・ADコンバーターを使った事が無い人は，[テスト運転](#_STEP1_ADコンバーター)（後述）の為に買っておいた方が良い。10kΩくらいのものがお勧め。  （参考）<https://akizukidenshi.com/catalog/g/gP-08012/> |
| wifi対応のマイコン  （ラズパイ） | wifi対応は必須じゃないが，置く場所の事を考えると対応しているものが良い。  ESP なども割と簡単にあつかえるマイコンでwifi対応。 |

#### C言語

概要

電力測定では高速な処理が必要なので，pythonよりも高速に処理ができるC言語を使った方が良い。

必要ありそうな基本情報をここに書く。

どんなプログラム言語でも最終的には「実行ファイル」が作られるが，pythonの「一行ずつの通訳」に対してC言語は「まとめて翻訳」して実行ファイルを作成する。これを「コンパイル」と言う。

ちなみにpythonのような一行ずつ実行ファイルに変換する言語を「インタープリタ言語」と言う。

事前準備

terminal:

sudo apt install build-essential # ビルド・ツールをインストール

touch hello.c # 今からソースコードを書くファイルを新規作成。

サンプルソース

例によってhello world。

エディタはGeany, nano, mousepad, vi, vscode　などなど。文字が編集できれば何でも良い。

[ hello.c ]

#include <stdio.h> # printf を使う為に必要なツールを使います，という宣言

int main(int argc, char \*argv[]){ # エントリ・ポイント　プログラムがここから始まる，という「お約束」。

printf(“hello world\n”); # 最後の ; を忘れないように注意。　\n は改行の意。

}

コンパイル

terminal:

gcc *hello*.c -o *hello* # コンパイルする → 実行ファイルができあがる。

./hello # コンパイルした実行ファイルを起動。 ./ は現在ディレクトリの意味

-o オプションで出来上がる実行ファイル名を指定できる。

-o オプションを省略すると a.out という実行ファイルが作成される。

追加情報

・#includeするヘッダファイル（.h）は /usr/include/ に格納されている。ソースを直接見る事で，各ヘッダの使い方の助けになるかも。

・ビットシフト（マイコンのプログラミングでは良く出てくる）

unsigned char original = 0b01001010; // unsigned char は8bit（2進数で8桁）

unsigned char right\_shifted = original >> 1; // 00100101　右シフト

//　　original を真ん中に挟むと分かりやすい // 01001010　オリジナル

unsigned char left\_shifted = original << 1; // 10010100　左シフト

printf("right shifed:%d\n", right\_shifted); // 10進数で値を確認。（標準では2進数のprintfはできない）

### 具体例

#### 概要

電力測定は難易度が高めなので，何段階かのSTEPに分けて進んだ方が良いように思う。

選定部品：

|  |  |  |
| --- | --- | --- |
| 項目 | 型番 | 備考 |
| 電流センサ  ￥980 | SR-3704-150N | 変流比　3000:1　　負荷抵抗は100Ωを使用する。 |
| ADコンバータ  ￥540 | AE-ADS1015 | 12bit　4ch　　シリアル通信：I2C  ※なんでも良いが，**負電圧の入力に対応したもの**を強く推奨。 |
| マイコン | RaspberryPi zero | OSはラズパイOSのbuster-lite:2021-05-07バージョンとする  [Webリンク](https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2021-05-28/2021-05-07-raspios-buster-armhf-lite.zip) |

lsb\_release -a

> Distributor ID: Raspbian

> Description: Raspbian GNU/Linux 10 (buster)

> Release: 10

> Codename: buster

uname -a

> Linux raspberrypi 5.10.52+ #1440 Tue Jul 27 09:53:08 BST 2021 armv6l GNU/Linux

資料：

条件が近く参考になる（但しArduino）：<https://qiita.com/yoroyasu/items/d4eea6ed952a589ca780>

補足

/usr/include にmcp3400.h とかがあるけど 2021-08-06

#### STEP1. ADコンバーター

##### 概要

ADコンバーターの使い方のまとめ ＋ ADコンバーターを使った事が無い人向きに，可変抵抗（ボリューム）を使ったテスト運転の仕方を記述。

購入品

可変抵抗は10kΩ程度が良い。 （参考）<https://akizukidenshi.com/catalog/g/gP-08012/>

この辺も持ってなかったら，あった方が便利

ブレッドボード <https://akizukidenshi.com/catalog/g/gP-05294/>

ジャンパー（オス-オス）<https://akizukidenshi.com/catalog/g/gC-05159/>

ジャンパー（オス-メス）<https://akizukidenshi.com/catalog/g/gC-08932/>

<https://www.marutsu.co.jp/pc/i/69682/>

ハンダごてが必要だけど，ラズパイの場合これがあると大分楽になる。

<https://akizukidenshi.com/catalog/g/gK-08892/>

##### AE-ADS1015(I2C)の場合

配線

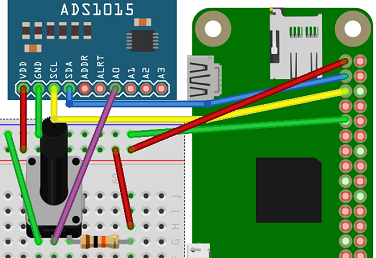
AE-ADS1015への接続を想定

[](https://akizukidenshi.com/catalog/g/gK-13838/)https://akizukidenshi.com/catalog/g/gK-13838/

ラズパイのピンは

3v3, GND,SDA, SCL を使用

右図はFritzingで描いた配線図。

この機種の場合は参考電圧を入れるピンが無いので，VDD（ADコンバーターの電源電圧）と同じ値を参考電圧で使用している（と思う）。

*※ADS1015は初期設定では入力電圧が±2.048Vなので，3.3Vを入力する事は控えたい。*

*⇒ 右図のように10kΩの抵抗で分圧して ～1.67V*

*くらいに落としておく。*

準備

terminal:

sudo apt install i2c-tools # インストール（最初から入ってる場合が多い）

sudo apt install python3-smbus # python 用のモジュールをインストール

pip3 install smbus2 # pipでも良いが，結局aptの方が良い，という声がある。

pip3 list | grep ‘smbus’ # インストール済みかどうか確認できる

sudo raspi-config # Interfacing Options でI2C を有効にする。

sudo i2cdetect -y 1 # 現在I2C接続している機器とそのI2Cアドレスを調べる。

# 0 1 2 3 4 5 6 7 8 9 a b c d e f # このような形で，有効なアドレスのみが表示される

# 00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- # それ以外の表示をする場合は

# 10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- # 通信が上手く行ってないかも。

# 20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

# 30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

# 40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --

# 以下略

sudo i2cget -y 1 0x48 # 通信できるか試す（任意）

sudo i2cget -y 1 0x48 0x00 b # i2cアドレス0x48のレジスタ0x00をバイト単位で読む

pipが入っていない場合

pip3 --version # pip(3)が入っているかどうか確認できる。

sudo apt install python3-pip # pip3のインストール。結局aptが一番良いという声がある。

（参考）pipよりaptの方が良い，という資料<https://karaage.hatenadiary.jp/entry/2018/10/05/073000>

サンプルプログラム（python）

[ ads1015\_smbus\_test.py ]

import smbus2 # 補足 smbusとsmbus2の違いは良く分からない。とりあえず新しい方にした。

import time

# 定数的設定

I2C\_ADDRESS=0x48 # 対象チップのI2Cアドレス

ADC\_CH=0 # ADコンバータの対象チャンネル

# 書込みデータの設定

os=1 # 運転状態 0:何もなし 1:変換開始

mux=0b100 | ADC\_CH # 対象chの選択 100: シングルエンド - ch0

pga=0b010 # オペアンプゲイン 010: ±2.048V

mode=1 # 変換モード 0: 連続変換 1:シングルショット[def.]

dr=0b100 # DataRate 100:1600SPS

# 2byte のlist にまとめる　MSB→LSBの順番

config\_data = [(os << 7 | mux << 4 | pga << 1 | mode), (dr <<5 | 0x03)]

bus = smbus2.SMBus(1) # bus番号を指定して通信モジュールをオープン。※bus番号は大体の場合1番。

try:

while True:

bus.write\_i2c\_block\_data(I2C\_ADDRESS, 0x01, config\_data) # 第2引数=0x01 → 対象がconfigに

read\_bytes = bus.read\_i2c\_block\_data(I2C\_ADDRESS, 0x00, 2) # (i2cアドレス, 先頭レジスタ, byte数)

ival=int.from\_bytes(read\_bytes, 'big', signed=True) >> 4 # int.from\_bytesはpython3以降

print(ival)

time.sleep(1)

except KeyboardInterrupt: # Ctrl + C で whileループから抜けれる

pass

bus.close()

⇒ 実行したら，ボリュームを回して値が変わるのを確認する。

*補足　pythonでは定数(const) は好まれない。特にプログラムが大きい時はsetting.pyなどの方法を取ると良い。*

資料

ADS1015の仕様：[ADS1015](#_ADS1015)（本ドキュメント内）

smbusの資料（英語）：<https://smbus2.readthedocs.io/en/latest/index.html>

pigpioを使った例：<https://tomosoft.jp/design/?p=10579>

補足：メーカーのgitレポジトリがあって，pythonで簡単にADS1015を使えるようになるらしい（未確認）

git clone https://github.com/adafruit/Adafruit\_Python\_ADS1x15.git # リポジトリをクローン（ダウンロード）

sudo python Adafruit\_Python\_ADS1x15/setup.py install # インストール

サンプルプログラム（C言語）

#include <fcntl.h> // O\_RDWR

#include <unistd.h> // close

#include <sys/ioctl.h> // ioctl

#include <stdio.h> // printf

#include <unistd.h> // usleep

#include <linux/i2c-dev.h> // <i2c/smbus.h> という情報も？

#include <linux/i2c.h>

#define ADC\_CH 0 // ADコンバータの対象チャンネル

int main(int argc, char \*argv[]){

int fd = open("/dev/i2c-1", O\_RDWR); // I2Cのバス1をオープン

ioctl(fd, I2C\_SLAVE, 0x48); // I2Cアドレスの設定。ioctlはデバイスの各種設定。

unsigned char os=1; // 運転状態 0:何もなし 1:変換開始

unsigned char mux=0b100 | ADC\_CH; // 対象 ch の選択 100: シングルエンド - ch0

unsigned char pga=0b010; // オペアンプゲイン 010: ±2.048V

unsigned char mode=1; // 変換モード 0: 連続変換 1:シングルショット[def.]

unsigned char dr=0b100; // DataRate 100:1600SPS

unsigned char cfg\_lsb = dr<<5 | 0x03; // config用のlsb（低い方のbyte）

unsigned char cfg\_msb = os<<7 | mux <<4 | pga<<1 | mode; // config用のmsb（高い方のbyte）

unsigned char config\_bytes[] = {0x01, cfg\_msb, cfg\_lsb}; // config用の送信byte配列

unsigned char addr\_ptr\_bytes[] = {0x00}; // アドレスポインタ用の送信byte配列

while(1){

write(fd, config\_bytes, 3); // config の書き込み

write(fd, addr\_ptr\_bytes, 1); // アドレスポインタの書き込み。

unsigned char read\_data[2]; // 読込み値を格納する変数

read(fd, &read\_data, 2); // readメソッドには &を付けてメモリアドレスを渡す

int ival = read\_data[0] << 4 | read\_data[1] >> 4; // 値の加工

printf("%d\n", ival);

usleep(500\*1000); // 500mSec待つ。（usleepの単位はμSec）

}

close(fd);

}

補足：C言語にはbyteという型が無い。1byte扱うのにちょうど良いのはunsigned char。

（参考１）<https://penkoba.hatenadiary.org/entry/20131222/1387689702>　色々な方法が示されている。

（参考２）<https://qiita.com/iwatake2222/items/d6645880c5bb91ce8a85>

（公式）<https://www.kernel.org/doc/html/latest/i2c/dev-interface.html>

i2cgetのソースコード　<https://github.com/omapconf/omapconf/blob/master/i2c-tools/i2cget.c>

WiringPiでやる場合（未確認）

#include <wiringPi.h>

#include <wiringPiI2C.h>

<https://www-makky.com/raspi-i2c-1/>

<http://wiringpi.com/reference/i2c-library/>

トラブルシューティング

No such file or directory: '/dev/i2c-0'

spiのバス番号に0番を選択してしまっている。1番を選択し直す。

ラズパイには2系統のspiがあるが，通常使用するにはbus1番の方。

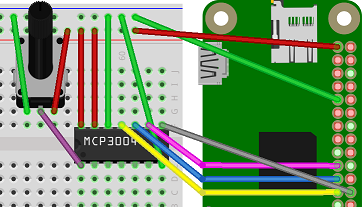
##### MCP3004(SPI)の場合

配線

MCP3004-I/Pへの接続を想定。

<https://akizukidenshi.com/catalog/g/gI-11987/>

SPIシリアル通信で使用するラズパイのピンはI2Cよりも多い。

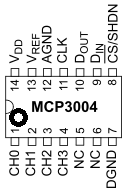
右図はFritzingで描いた配線図。

（但しMCP3004のFritzingパーツが無かった為，代わりに汎用ICを使っている）

VDD ADコンバータ電源

VREF 参考電圧

AGND Ch0~3のマイナス側

DGND VDDのマイナス側

注意点：このADコンバーターで電流センサの値を測りたい時は，オペアンプによるレベルシフトが必要。

準備

terminal:

sudo raspi-config # Interfacing Options でSPI を有効にする。

lsmod | grep spi # （再起動後）有効になったかどうかの確認

ls -la /dev/spi\* # 有効かどうかの確認はこれでも良い

sudo pip3 install spidev # 最初から入っている事も多い。python2を使う時はpip

pip3 list | grep ‘smbus’ # インストール済みかどうか確認できる

サンプルプログラム（python）

[ mcp3004\_spidev\_test.py ]

import spidev

import time

spi = spidev.SpiDev()

spi.open(0, 0)

spi.max\_speed\_hz = 100000 # 必ず設定する。

def read\_mcp\_10bit(ch): # 10bitのMCP ADコンバータの現在値を返すメソッド

ctrl\_bits = 0b11000 | ch # | は論理和，要するに足し算

sending\_data = [ctrl\_bits,0x00,0x00] # 送信データ 3byte(=24bit)分

ret = spi.xfer2(sending\_data) # 3byteのデータを受信する

return (ret[1] << 6) | (ret[2] >> 4) # 受信データの加工　12bitの場合はまた異なるので注意

try:

while True:

ch0\_val = read\_mcp\_10bit(ch=0) # chは0～3 ※chは0から始まる数値

print("value of ch0: %s" % (ch0\_val))

time.sleep(1) # sleepメソッドの単位は[秒]

except KeyboardInterrupt: # Ctrl + C で whileループから抜けれる

pass

spi.close()

⇒ 実行したら，ボリュームを回して値が変わるのを確認する。

補足①：ctrl\_bits

最初の1はスタートビット，次の1は「chをシングルエンドで使う」の意。残りの3bitで対象chを選択する。

（参考）<https://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>：(p19～)

補足②：受信データの加工

通常なら，［ 8bit左シフトした第1byte ＋ 第2byte ］で数値化できるが（数字の10の位，1の位みたいな感じ）

第1バイトの**最初の2bitは常に0**だった為（※），相殺した分6bit（8-2=6）の左シフトとした。

MCP3004の分解能は10bitなので残りは4bit。1byte(8bit)の残り4bitは必要ないので右シフトする。

（※）補足③を参照

補足③：受信データの詳細

MCP3004はマスターからの電文終了後の何クロック間は返信せず，この間のマスター側が受信するbitは全て0になる。確認した時はこの間1クロックだったが，これは[通信モード](#_通信モード)によって変わってくると思う。

更にMCP3004からの返信データの第1bitは0(Null Bit) になると[資料](https://akizukidenshi.com/download/ds/microchip/mcp3008.pdf)（FIGURE5-1 p20）に記載がある。

結果的に**受信データの最初2bitは常に0**。例えば，返信第1byteが0xFF(=255)でも受信データは0011から始まる事になる。（と思われる 2021-07-27）

サンプルプログラム（C言語）

（編集中）

[ mcp3004\_spidev\_test.c ]

#include <sys/ioctl.h> // ioctl

#include <fcntl.h> // O\_RDWR

#include <stdint.h> // uint32\_t, uint8\_t

#include <unistd.h> // open, close, write, read, usleep

#include <stdio.h> // printf

#include <linux/spi/spidev.h>

int fd =0;

int transfer(unsigned char tx[], void \*rx, int n){ // pythonのxferに似たmethod

struct spi\_ioc\_transfer tr = { // 転送に使われる構造体

.tx\_buf = (unsigned long)tx,

.rx\_buf = (unsigned long)rx,

.len = n,

.delay\_usecs = 0,

.speed\_hz = 100000,

.bits\_per\_word = 8,

};

int ret = ioctl(fd, SPI\_IOC\_MESSAGE(1), &tr);

return 0;

}

int read\_mcp\_10bit(unsigned char ch){

unsigned char ctrl\_bits = 0b11000 | ch;

unsigned char sending\_bytes[] = {ctrl\_bits, 0x00, 0x00};

unsigned char read\_bytes[3];

unsigned char tx[] = { 0xF7,0x00,0x00 }; // dummy

transfer(tx, &read\_bytes, 3)

int ival=read\_bytes[0] << 6 | read\_bytes[1] >> 4;

return ival;

}

int main (int argc, char \*argv[]){

fd = open("/dev/spidev0.0", O\_RDWR) // bus0 chip0をオープン

while(1){

int an\_val = read\_mcp\_10bit(0);

printf("%d\n", an\_val);

usleep(500\*1000);

}

close(fd);

}

（参考）<https://qiita.com/eggman/items/7d0b35c3ed86e8983a50>

<https://qiita.com/iwatake2222/items/d6645880c5bb91ce8a85>

#### STEP2. 電流センサのテスト

##### 概要

・電流量の瞬時値を返すread\_ampere(int ch)と言うメソッドを作る

・まずは1mSec毎で適当な交流電流のアナログ値を測定してみる

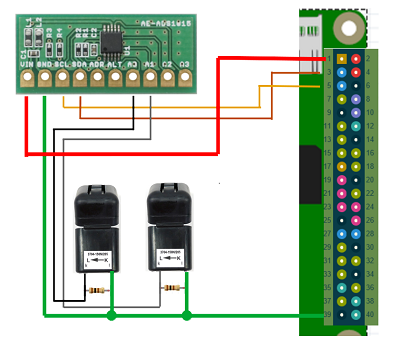
右図は大体こんな感じというイメージ。

・オペアンプの使用が必要であれば，その対応も行う。

補足

rms電流値への計算はSTEP3で行い，ここでは電流量の瞬時値を取るだけにする。

配線図

電流センサ2つ接続の例。（1つでも良い）

以下のSTEPでもこの配線を前提とする。

電流センサ

L: ライブ？（プラス）

K: カソード（マイナス）

抵抗を並列で入れる

⇒ 100Ω抵抗を入れた時：

電流センサへの入力　：±60A（※１）

電流センサからの出力：±2.0V

となり，ADS1015のデフォルトの入力電圧範囲である±2.048Vにほぼ一致する。

ラズパイ

（参考）<https://pinout.xyz/pinout>

1: 3v3(out) 他にも17が3v3

3: DATA　シリアルデータ

5: CLOCK　シリアルクロック

39: GND

他にも6,9,14,20,25,30,34がGND

※１　実効値40Aの時，最大値が56.57Aになる。電流センサを40Aのブレーカーに付けるような想定

##### 電流センサの接続

1つ1000円以上する比較的高価な電子部品はできるだけ使いまわしたい。

また，持ち運びの事などを考えても，電流センサを基板から簡単に取り外せるようにしておけると理想。

更に，電流センサは買ってきたままではブレッドボードに挿せない為，なにかしらのコネクタが必要。

ここでは，電流センサ用の接続方法のアイディアをまとめてみる。

基盤に直接ハンダ付けで良い，と言う人は読み飛ばしてOK。

|  |  |  |
| --- | --- | --- |
| 名称 | 写真 | 備考 |
| コネクタ |  | 電子分野で最も一般的なコネクタ。  専用の圧着ペンチが必要，コネクタの種類が多く迷う，など面倒な面もある。コネクタは迷ったらQIコネクタが良いかも。 |
| PA-09  （圧着ペンチ） |  | 上に挙げた，コネクタの圧着ペンチ  大体 \4,000 くらい。  安物を買うと痛い目に合うらしい。エンジニア　というメーカーのものが有名。 |
| オーディオプラグ |  | 3.5mmのモノラルのプラグ（メス）とジャック（オス）を使う。（ステレオ用のものでも可）　合計で￥120くらいから。ハンダ付けは多少必要だが，電流センサ自体は抜き差し簡単で専用の工具も不要。  ただ，ブレッドボードに対応したプラグは無い。 |
| ピン |  | 特にブレッドボードで使う時。  電流センサのケーブルとピンを，少ない面積分だけハンダ付けする。実験が終わったらハンダを外せばよい。 |
| ワニ口クリップ |  | 苦肉の策  あまりお勧めしない。ただ，一番楽な方法でもある。 |

※むしろ，他に良い方法があったら教えて欲しい。2021-08-09

##### アナログ値を記録

概要

STEP1のソースコードをクラス化する。そうする事で後のSTEPで使いやすくなる。

まずはそのままアナログ値を記録

※pythonでは処理速度に限界があるのでsleepなしでwhileループをそのまま行う。

電流量瞬時値に変換するメソッド

get\_instantaneous\_ampr()

サンプルプログラム（python）

（編集中）

ads1015\_i2c.py

ADコンバーターの処理を任せるモジュール。以降のSTEPでもこれを用いる

import smbus2

from enum import IntEnum

class ConfigPga(IntEnum): # pgaの値を分かりやすくする為のEnum

GAIN\_6144 = 0b000

GAIN\_4096 = 0b001

GAIN\_2048 = 0b010

GAIN\_1024 = 0b011

GAIN\_0512 = 0b100

GAIN\_0256 = 0b101

class Ads1015\_I2C():

def \_\_init\_\_(self, i2c\_address=0x48, bus\_number=1):

self.\_i2c\_address=i2c\_address

self.\_bus\_number=bus\_number

self.\_os= 1 # 運転状態 0:何もなし 1:変換開始

self.pga= ConfigPga.GAIN\_2048 # オペアンプゲイン 0b010 ±2.048V

self.mode=1 # 変換モード 0:連続変換 1:シングルショット[def.]

self.dr=0b100 # DataRate 0b100:1600SPS

self.\_bus = smbus2.SMBus(1)

def \_\_del\_\_(self):

self.\_bus.close()

def \_read\_analog(self, mux):

config\_data = \

[(self.\_os << 7 | mux << 4 | self.pga << 1 | self.mode), (self.dr <<5 | 0x03)]

self.\_bus.write\_i2c\_block\_data(self.\_i2c\_address, 0x01, config\_data)

read\_bytes = self.\_bus.read\_i2c\_block\_data(self.\_i2c\_address, 0x00, 2)

return int.from\_bytes(read\_bytes, 'big', signed=True) >> 4 # int.from\_bytes はpython3以降

def read\_analog\_single(self, ch):

if ch < 0 or ch > 3:

raise ValueError("range of ch must be in [0-3]") # 例外エラーを発生させる

mux=0b100|ch # 0b100 シングルエンド

return self.\_read\_analog(mux)

def \_clac\_millivolts\_from\_analog(self, analog\_value): # アナログ値からmVを算出するメソッド

milli\_volt\_max=2048.0

if self.pga < 1:

milli\_volt\_max=6144.0

elif self.pga < 0b101:

milli\_volt\_max=2048.0\*(2 \*\*(2-self.pga)) # アナログ値 × { 2 の (2 - pga) 乗}

else:

milli\_volt\_max=256.0

milli\_volt= (analog\_value \* milli\_volt\_max) / 2048.0 # ミリボルトに変換

return milli\_volt

def read\_millivolts\_single(self, ch):

if ch < 0 or ch > 3:

raise ValueError("range of ch must be in [0-3]") # 例外エラーを発生させる

ana\_val=self.read\_analog\_single(ch) # アナログ値取得

return self.\_clac\_millivolts\_from\_analog(ana\_val)

サンプルプログラム（C言語）

（編集中）

※C言語ではある程度の高速処理が見込めるので，1mSec待つ処理を加える。

但しC言語はファイルを複数に分ける時のやり方が分からない 2021-08-06現在：

.hファイルだけでやるのが理想

makefileを使う必要がある？？

ここでADコンバーターの処理を任せるヘッダファイルも作成する。以降のSTEPでもこれを使用する。

（日時取得の参考資料）<https://8ttyan.hatenablog.com/entry/2015/02/03/003428>

##### 電流値瞬時値

サンプルプログラム（python）

サンプルプログラム（C言語）

メモ

２つの.cファイルをコンパイルする方法。（C++じゃないとできないんだっけ？）

mvolt\_ampere\_coeffient = (TRANS\_RATIO / (RESIST\_OHM \* RESOLUTION\_MAX \* 1000.0)) \* 1.0;

#### STEP 2α オペアンプによるレベルシフト

##### 概要

MCP3204を使う場合に必要な，レベルシフトに関して記載

##### 部品選定

NJM2747D

|  |  |  |  |
| --- | --- | --- | --- |
| 項目 | 公称値 | 具体値 |  |
| 差動入力電圧範囲 | ±15V or V+ | 0〜3.3V |  |
| 最大出力電圧 | 4.75～4.9V | 3.05～3.2V | VIN=5.0の時 |

オペアンプ選定は，計算とかが色々めんどくさくならないように，レールtoレール（フルスイング）品を選定するのを推奨。（それでも実際にGND〜VCCまで入れれるものは少ない）

MCP3204

概要：12bit 4ch ADコンバータ

|  |  |  |  |
| --- | --- | --- | --- |
| 項目 | 公称値 | 具体値 |  |
| 入力電圧 | -0.6V ～ VDD+0.6V | -0.6～4.2V |  |

負荷抵抗

電流センサ用の負荷抵抗は10Ωを想定。 ⇒ 電流センサからの出力電圧は±200mV。

差動減算回路（右下の配線図）の出力計算式

VOUT = (Vin2 – Vin1) × (R2 / R1)

補足　Vin1はマイナス側。電流センサからの交流信号。　Vin2はプラス側。直流のレベルシフト電圧。

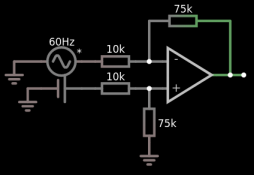
オペアンプは0.1〜3.05Vまで出力できるものとする（レールtoレールでも少し余裕を見る）

⇒2.95Vの範囲が使える。

⇒ 1.575Vを中心とした片範囲1.475Vになる

⇒ 電流センサが出す±200mVを±1475mVに増幅したい。そのゲイン比は1475/200 = 7.375倍

・抵抗は11kΩと82kΩで増幅率7.45倍とする。

　⇒ 1.575Vを中心とした片範囲1.49Vに修正

⇒ 最低値のときに85mVになるレベルシフト電圧が必要

・シフト電圧200+11.4(= 85 ÷ 7.45) =211.4mV

　⇒ 3.3Vからは7.5kΩと510Ωで作る。=210.11mV

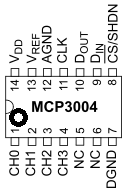
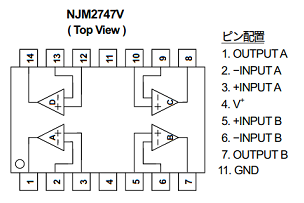
⇒最終的な電力出力は75mV〜3075mVとなる。

（1575mVが真ん中）

右の図は[Webツール](https://www.falstad.com/circuit/circuitjs.html)でのシミュレーションの図。

補足10kΩと75kΩが良かったが，75kΩは手に入りにくい？

##### 配線



サンプルプログラム（python）

まずはオフセット電圧を見る

サンプルプログラム（C言語）

#### STEP3. 定周期処理

##### 概要

・1秒間のRMS電流量を計算するプログラムとする。（後々の計算が楽になるので）

・まず，実験的にラズパイの定周期処理の限界と言うか，能力を見てみる。

・

補足

結論を先に言うと，ラズパイではどうあがいても正確な定周期処理ができない為，正確な電力値計算は無理。それでも，大体の電力量は取れるであろうという考えでこの先は進める。

##### 高速の定周期処理

サンプルプログラム(python)

まずは何mSecくらいがラズパイ（python）の処理限界なのか見てみる。

[ thread.py ]

import signal

import time

last\_time=time.time() # グローバル変数を「宣言だけ」しておく

def test\_handler(arg1, arg2):

global last\_time # globalがないとグローバル変数をmethod内で使用できない。

current\_time = time.time()

interval = current\_time - last\_time

print(interval)

last\_time = time.time()

signal.signal(signal.SIGALRM, test\_handler) # 送るsignalの種類と対象メソッドの指定

signal.setitimer(signal.ITIMER\_REAL,1,0.01) # ここの**第3引数で時間を指定**。単位は秒。

time.sleep(10) # sleepメソッドの単位は「秒」

結果：

・2mSec周期ではエラーが出て止まった。

・5~10mSec間隔くらいまでは動作する。2021-07-16 ⇒ただ、他の処理が加わるとどうなるか。

但し，けっこう大きな時間周期のばらつきがある

設定10mSec →　6.153～12.161 mSec 設定5mSec → 　4.138～6.914 mSec

（公式情報）<https://docs.python.org/ja/3/library/signal.html> signal.setitimer

（補足）Windows環境ではtime.perf\_counter()を使った方が良い，という情報あり

情報元：<https://qiita.com/takeopy/items/170d0e1ddbf02ef9fbb9>

⇒ 恐らくpythonでは5mSecくらいが限界ではないか。

サンプルプログラム(C言語)

まずは，実験。pythonの例と同じく実行時間間隔の限界を調べる

[ thread.c ]

#include <stdio.h> // printf

#include <unistd.h> // usleep, useconds\_t

#include <sys/time.h> // timeval

#include <pthread.h>

void \*thread\_function1( void \*ptr ); // メソッドの宣言

int main(void){

useconds\_t tick1 = 2000; //**ここで時間間隔を指定**。　単位は[μSec] =1秒÷1000÷1000

pthread\_t thread1;

pthread\_create(&thread1, NULL, thread\_function1, (void \*) &tick1);

pthread\_join( thread1, NULL); // thread1 の終了を待つ

}

void \*thread\_function1(void \*ptr){ // メソッドの定義

useconds\_t tick = \*( int \* )ptr;

struct timeval tv;

while(1){

gettimeofday(&tv, NULL);

printf("diff\_tv:%lf\n",tv.tv\_sec, tv.tv\_usec);

usleep(tick);

}

}

（参考）<https://www.ishikawa-lab.com/RasPi_wattmeter.html>

差分だけ見たい時は

while(1){

gettimeofday(&tv, NULL);

double current\_tv = ((double)(tv.tv\_sec) + (double)(tv.tv\_usec) \* 0.001 \* 0.001);

if(last\_tv > 0){

double diff\_tv = current\_tv - last\_tv;

printf("%lf\n", diff\_tv);

}

last\_tv=current\_tv;

usleep(tick);

}

コンパイル

terminal:

gcc -o thread thread.c -l pthread # 別ライブラリ(pthread)を使う時ので -l オプションが必要

./thread # 実行

結果：

設定10mSec → 10.154～10.269 mSec 設定5mSec → 5.170～5.502 mSec

設定2mSec → 2.112～2.24mSec 設定1mSec → 1.180～1.380 mSec

但しごく稀に大きなずれも起こる。それでもpythonよりはずれが小さい。

また，設定値以下の周期にはならない模様。

##### Armsの算出

##### 資料（定周期処理）

割込みの資料１ <https://blog.boochow.com/article/458880693.html>

割込みの資料２ <https://tomosoft.jp/design/?p=5647>

タイマー情報１ <https://hensa40.cutegirl.jp/archives/1268>

#### STEP4. 電力値計算

##### 概要

STEP4は「5分毎に1分間の電力測定を行う」と言う仮の仕様に基づいて進める。

タイマーは以下の2種類が必要

・「1分間の測定」の部分を受け持つタイマー。精度はそこまで求められない。

・「5分毎に1回」の部分を受け持つタイマー。ラズパイでは色々と方法がある。精度は一番大雑把で良い。

測定の時間周期

交流1周期につき8～10点以上は電力値を計測したい。もちろん頻度は多ければ多いほど良い。

例えば1周期に8点とすると：

東日本50Hz：50×8 = 400Hz （2.5mSec周期）

西日本60Hz：60×8 = 480Hz （2.08mSec周期）

⇒ 2mSec周期を目指してみようと思う。

##### 「5分毎に１回」の処理

cronでやる場合

おそらくcronが一番簡単に実装できるのでお勧め。

ファイル構成

/home/*user\_name*/ 扱いやすさの為 /homeにしたが、/usr/localなどでも良い

│└ watt\_mon 今回作成するサービスのファイルを格納するディレクトリ

│　└ watt\_clac.py 本番では電力量計算を行うスクリプトをここに置く

補足：watt\_clac.py　の内容は，下の「サービスを作成する場合」を参考にして下さい。

terminal:

crontab -e # crontab の設定を開始する。→ 設定ファイルが開かれる。

補足

・初めてcrontabを使う場合はテキストエディタを選択する必要がある。→ 迷ったらnanoがお勧め。

・crontabはユーザー権限でも良いらしい。（sudoが要らない）

crontabの書き方（フォーマット）

分 時 日 月 曜日 実行するファイルやbashコマンド

具体例

\*/5 \* \* \* \* python /home/pi/watt\_mon/watt\_clac.py # 5分毎に実行する例

⇒ 厳密には「時計の分の値が5の倍数の時」であって，後述の「サービスを利用する方法」とは結果が異なる。

*補足：最初は動作確認しやすいように1分毎で動くようにcrontabを書込んだ方が良い。*

他の例

|  |  |
| --- | --- |
| 書式 | 意味 |
| 33 14 \* \* \* | 毎日14:33に実行。（アスタリスクは「すべての条件に一致」の意） |
| 0 04 \* \* 1 | （月）の午前4:00に実行（毎週）　曜日は0（日）～6（土） |
| 0 21 \* \* 1-6 | （月）〜（土）まで21:00に実行。（ハイフンで範囲指定） |
| 0,10,20,30,40,50 \* \* \* \* | 10分おきに実行（毎時） |
| \*/10 \* \* \* \* | 10分おきに実行（毎時） |
| \* 1 \* \* \* | 1:00から1:59まで1分おきに実行（毎日） |
| 0 \*/1 \* \* \* | 0分に1時間おきに実行（毎時） |
| 0 \* \* \* \* | 0分に1時間おきに実行（毎時） |
| 2 8-20/3 \* \* \* | 8:00から20:00までの間に3時間おきに実行。8:02などに実行される。 |
| 30 5 1,15 \* \* | 毎月1日と15日の5:30に実行。（コンマはorの意らしい。） |
| @reboot | cron デーモンが起動した時に実行される |

サービスを作成する場合

ファイル構成

/ ルートディレクトリ

├/home/*user\_name*/ 扱いやすさの為 /homeにしたが、/usr/localなどでも良い

│└ watt\_mon 今回作成するサービスのファイルを格納するディレクトリ

│　└ watt\_clac.py 本番では電力量計算を行うスクリプトをここに置く

├/etc/systemd/system

│├ watt\_mond.service

│└ watt\_mond.timer

[ watt\_clac.py ]

※仮のスクリプト。色々な事を確認する為に，現在時間を追記していくスクリプト

import os

from datetime import datetime as dt

with open('/home/*user\_name*/watt\_mon/test.txt', 'a',encoding='utf-8') as f:

f.lineterminator='\n'

f.writelines(f'datetime{dt.now()}\n')

f.close()

[ wattmond.service ]

[Unit]

Description=watt monitoring service using CT Sensor # ここの内容はなんでも良い

[Service]

Type=simple

ExecStart=/usr/bin/python3 /home/*user\_name*/*watt\_mon*/*watt\_clac*.py # 実行されるコマンド

[Install]

WantedBy=multi-user.target

*補足：コマンドはフルパスで書いた方が良い。相対アドレスで書くと失敗しやすい。*

[ wattmond.timer ]

[Unit]

Description=timer of watt\_mond service

[Timer]

OnUnitActiveSec=5min # 5分毎に動く設定。（Secと書いてあるのにminも設定できる）

Unit=*wattmond*.service

[Install]

WantedBy=multi-user.target

*補足：最初は動作確認しやすいように1分毎で動くような設定にした方が良い。*

～サービスを作成する場合　つづき

terminal:

sudo systemctl start *wattmond*.timer # タイマーを開始する。

#sudo systemctl start *wattmond* # サービス本体は開始しなくて(inactiveのままで) 良いらしい。

sudo systemctl enable *wattmond*.timer # OS起動時にタイマーが自動起動するようにする。

sudo systemctl stop *wattmond*.timer # タイマーを停止する。

# 補足コマンド

sudo systemctl daemon-reload # .service ファイルを書き換えたら実行する必要がある

sudo systemctl list-unit-files --type=service | grep *wattmond* # サービスが認識されたかどうか

トラブルシューティング

上手く行かない時は動作確認の基本，システムログの確認。

cat /var/log/syslog | grep *wattmond* # ファイルを直接見る方法。わりと見やすいと思う。

journalctl | grep *wattmond* # こちらでも良い。比較的新しいやり方（らしい）

サービスの状態の確認

sudo systemctl status *wattmond* # こちらの方が細かいログが確認できる。（.serviceの方の状態確認）

sudo systemctl status *wattmond*.timer # タイマーの状態を確認

##### 実際の電力測定

（編集中）

最後に，実際に電力量を計算する為のプログラム。STEP3.で算出した「1秒間の電力量」を利用する

算出するのは；

1分間の電力量平均

1分間の電力総量 （補足）Wh（ワットアワー）を算出する為に必要

「1分間」の測定は単純にtime.sleep()メソッドを使えば良い。

（補足）threading.Timerを使う例もあるらしい？2021-07-28

サンプルプログラム(python)

（編集中）

サンプルプログラム(C言語)

（編集中）

「1分間」の測定は単純にdelay()メソッドを使えば良い。

include <time.h>

wait\_nanosecond

#### STEP5. ソケット通信

（編集中）

ソケット通信はpythonに任せる。

参考コード<http://openlabtools.eng.cam.ac.uk/Resources/Datalog/RPi_ADS1115/code/ADS1115_sample.c.html>

割込みに関してのみ１<https://ryusakura.wordpress.com/2016/01/27/469/>

### 詳細情報

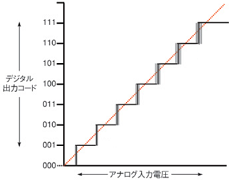
#### 概要

上の「具体例」の詳しい内容を記述。

けっこう難しい内容になっているので読み飛ばしても良い。

#### ADコンバータ

##### 概要

通常マイコンなどのデジタル機器は，電流量などの「アナログ値」を直接扱えない為に，デジタル信号に変換する機器が必要になる。

大まかなイメージは右図の通り。デジタル変換後の値は，細かく見るとカクカクとなってしまう。

測定された値を様々な方法でデジタル化し，デジタル化された値はADコンバータの内部メモリにて保持される。それを利用側マイコンからI2Cなどの「シリアル通信」で読み出す。

要点：

・少し高くても，**負電圧の入力に対応**したADコンバーターを強く推奨する。

・複数のch（チャンネル）を持ち，複数の機器からの入力を受け付けるADコンバータもある。

（例）１つのADコンバータで4つの電流センサの値を読み取る　など。

・マイコンによってはアナログ入力を受け付けるIOピンが最初から付属している場合もある。

ADC（ADConvertor）がアナログからデジタル。DAC（DAConvertor）が付いているマイコンもある。

・シリアル通信がI2C対応のものとSPI対応の物がある。

##### データシート（ADコンバータ）

（例）MCP3004-I/P <https://akizukidenshi.com/catalog/g/gI-11987/>

bit数

別名：Resolution（分解能）

アナログ値をデジタル化する時の縦軸の目盛りの細かさの事。よくあるのは8,10,12bit。

（補足）bit数から実際の目盛り数は2の何乗かで算出。例えば10bitであれば210=1024なので縦軸の目盛りは1024段階。なお，11bitでは211=2048段階なので，1bit変わるだけでデータの質が大きく異なる。

※符号付き　は+/- を示すビットが更に1bit追加される。

サンプリング回数

別名：Conversion Time（変換時間）；Sampling Rate；Data Rate；Settling Time

アナログ値をデジタル化する時の横軸（時間軸）の細かさの事。

単位がclock cycles の場合は，別項目「処理速度」と併せてみる。

またSPSという単位はSamples per Secondの意で，1秒間に何回デジタル値に変換されるか，を指す。

※厳密にはsettling timeはデジタルデータ出力が収束するまでの時間の事。latencyよりも長い。

待ち時間

別名：latency

ADコンバータがアナログ信号を取り込んでからデジタルデータを出力するまでの時間。

処理速度

別名：Throughput Rate

一秒間に何サンプルの変換が可能かを示す。

ch数

データシートではなく，「特徴(features)」や，回路図にChがいくつあるかを読むと分かる。

IOピン

|  |  |  |
| --- | --- | --- |
| 名前 | 別名 | 説明 |
| CH0 |  | 複数Chある場合の測定電圧のプラス側。  （例えば電流センサのプラスをここに繋ぐ） |
|  | VIN+, +IN, T+ | 測定Chが１つの場合の電圧のプラス側 |
| AGND | VIN-, -IN, T-, GND | 測定電圧のマイナス側。 |
| VCC | +V | ADコンバータの動作電源のプラス側。 |
| VDD | VREF | 参考電圧，またはADコンバータの動作電源（又は兼用）のプラス側。 |
| VSS | GND | 〃　　　　　　　　　　　　　　のマイナス側。 |
| SDA | DATA | シリアル通信がI2Cの場合に使用する，データピン。 |
| SCK | CLK, SCL | 通信クロックピン。  ※I2CでもSPIでも似た名前の事が多いのでまとめる |
| MOSI | DIN | シリアル通信がSPIの場合にマイコンからのデータ受信ピン。 |
| MISO | SDO, SO, DOUT | シリアル通信がSPIの場合のマイコンへのデータ送信ピン。 |
| SS | CS（Chip Select） | シリアル通信がSPIの場合のチップセレクト　Slave Selectの意。 |

（補足） 　*などと上に棒線がある場合があるが，これはB接点（Normal Close）で信号を入れろ，と言う意味。常時HIGH（例えば3.3V印加）にしておき，LOW（例えば0V）にすると通信を開始する。*

##### 具体的な使い方

###### 参考電圧

最終的にADコンバーターが出力するデジタル値に密接に関連する**重要な電圧値**。

この「参考電圧」は通常ADコンバータを動作させるための電源電圧と同じ事が多い。

（マイコンであればマイコンの動作電圧）

【 重要 】

この参考電圧がふらふらと安定していないと，最終的なデジタル値も変動してしまう。

参考電圧をしっかり安定させる事がADコンバータを使う上では大切。

ADコンバーター内部で保持するデジタル値は以下のようになる：

デジタル値 ＝ ×分解能

（例）分解能10bit (210=1024) のADコンバータで「参考電圧」として3,3Vを印加。

測定電圧，例えば電流センサの出力電圧が0.1V(100mV) だった場合

（0.1÷3.3）×1024　=31.03

⇒ 内部では31　という値が保持される。

##### カタログ（ADコンバータ）

|  |  |  |
| --- | --- | --- |
| 商品情報 | 参考写真 | 備考 |
| AE-MCP4726  取扱店  秋月電子通商  コード: K-07995  [Webリンク](https://akizukidenshi.com/catalog/g/gK-07995/)  値段 ￥150 |  | 12bit　1ch　　　※MCP4726が載っている  シリアル通信：I2C  入力電圧2.7-5.5V  　Settling Time　6μsec(=166kHz) |
| MCP3004-I/P  コード: I-11987  値段 ￥180 |  | 10bit　4ch　　シリアル通信：SPI  安いが，10bitでは少し厳しいかも。 |
| MCP3204-CI/P  コード: I-11131  [Webリンク](https://akizukidenshi.com/catalog/g/gI-11131/)  値段 ￥300 |  | 12bit　4ch　シリアル通信：SPI  電源電圧2.7-5.5V  入力電圧 0.25V ～ VDD  サンプリングレート  100ksps@5.0V 　50ksps@2.7V  Bグレード品（低誤差）MCP3204-BI/P　¥340 |
| AE-ADS1015  取扱店  秋月電子通商  コード: I-00238  [Webリンク](https://akizukidenshi.com/catalog/g/gK-13838/)  値段 ￥540 |  | 12bit　4ch　　シリアル通信：I2C　　**マイナス電圧対応可**  電源電圧範囲：2～5V  入力電圧±256mV　～　±6.144V  詳細情報：<https://www.tij.co.jp/product/jp/ADS1015>  （補足） switch science の1136がこれとほとんど同じだが，値段が￥1562なのでこっちの方が良い。 |
|  |  |  |

12bitじゃないと精度的に厳しいかも 2021-08-11

##### 用語（ADコンバータ）

PGA：Programable Gain Amp

増幅率（ゲイン）をソフト的に変更できる機能の事。ADコンバーター内部にオペアンプ（増幅回路）が内蔵されている。

シングルエンド

1つのchとGND間の電圧差をADコンバーターが取り込む方式の事。（対：差動入力）

差動入力

チャンネルを2つ使い，それぞれの電圧差をADコンバーターが取込む方式の事。（対：シングルエンド）

チャージポンプ

コンデンサとスイッチを組み合わせた昇圧回路。基本的に2倍の電圧を得られる。

DNL；Differential Non-Linearity

微分非直線性誤差。±*n*LSBで表される。このLSBは測定できる最低単位いくつ分の誤差になる，という意味。例えば，分解能1024(10bit)，Vref=5Vで±4LSBの誤差の時，±4×(5÷1024)=±19.53mVの誤差。

Pseudo-Differential

疑似差動入力。AN+とAN-を同タイミングではなく，別々のタイミングで測定するモードの事。

ノイズを低減し，より高精度の測定を可能にする。

Common-Mode Rejection

同相信号除去。差動増幅回路などでの2つの入力に共通する入力信号を除去する事。

#### シリアル通信

##### 概要

シリアル通信の詳細情報で必要そうな情報があればここに書きたい。

シリアル通信は，コンピューター間でモールス信号のようなオン／オフだけの電気信号を連続で（＝シリアル）送る通信方式。

シリアル通信の種類

|  |  |  |
| --- | --- | --- |
| 名称 | 用途 | 備考 |
| I2C | マイコン同士 | ワイヤ線が少ない。シンプルな通信。 |
| SPI | マイコン同士 | I2Cより高速。 |
| JTAG | マイコン | 開発時にデバッグで使われる。 |
| CAN | 車とマイコン | 車検用の機器などと通信する。 |
| RS232C | 様々 | 最も広く使われている。歴史は古く，legacy deviceなどと揶揄されながらも未だに現役。  誤用だが，RS422/485などと共にUARTとも呼ばれる。 |
| RS422/485 | 様々 | 強化版のRS232Cといった立ち位置。  ノイズに強く，高速。ethernetと酷似。 |
| CCLink | 工業機器とPLC | ノイズに強く，高速，の工業用版。 |
| MODBUS | 工業機器 | 物理層はRS-485かethernetらしい。実はよく知らない。 |

補足

・ArduinoStudioのSerial.begin()の「Serial」はRS232C通信の事を指している。

##### SPI

4W(4線式)と3W(3線式)がある。

I2Cに比べると通信速度が早いが，SPIの方が通信線が多かったり，プログラム処理がめんどくさい事が多い。

###### 通信モード

|  |  |  |  |
| --- | --- | --- | --- |
| mode | クロック極性(CPOL) | クロック位相(CPHA) | 備考 |
| 0(0b00) | 0:正論理 | 0:クロックが0→1の時 | mode0が一般的（らしい） |
| 1(0b01) | 0:正論理 | 1:クロックが1→0の時 |  |
| 2(0b10) | 1:負論理 | 0:クロックが0→1の時 |  |
| 3(0b11) | 1:負論理 | 1:クロックが1→0の時 |  |

（補足）CPOL：Clock Polarity CPHA ：Clock Phase

##### I2C

プルアップ抵抗

#### 電流センサ

##### 基礎知識

###### 種類

クランプ型

電流が流れる時に発生する磁束を利用した測定方法。

ケーブルを輪っかの内側に入れれば良いだけなので安全に使える。

（補足）細かく言うとクランプ型の中にも色々種類はある。

（補足）クランプ型には直流電流は測れないものがあるので注意。

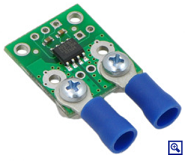
注意：

コンセントごとに電流センサを設けたい場合には以下の点に注意する。

家庭用の電源ケーブルは2本1対となっていて，電流を測る時にはその片方1本を対象としなければならない。（右図参照）

2本まとめてクランプすると必ず0Aになり，電流値を計測する意味が無い。

（補足）アース線込みで3本の物もあるかも。



直結型

電気の流れている導線を直接繋いで計測する方式。（右図参照）

クランプ型より安価な事が多いが色々危険なので購入しない事。

###### 接続方法

各コンセントに付ける場合

右図のように線が1本ずつになっているコンセントを手に入れる必要がある。

まず売ってないので2本に分けれそうなケーブルを買って来て自力で引き裂くか，自力でコンセントのパーツを買って来て配線して作るしかない。

（配線自作はあまりお勧めしないが，理解すれば簡単でもある）

ブレーカーに付ける場合

配電盤内はむき出しの100，200Vの端子があったりするので，十分に注意する事！！

**たかが100Vだと甘く見ない事**。交流はけっこう怖い。あくまで自己責任で。

基本的に金属部分（絶縁被膜が無いところ）を避ければ大丈夫。

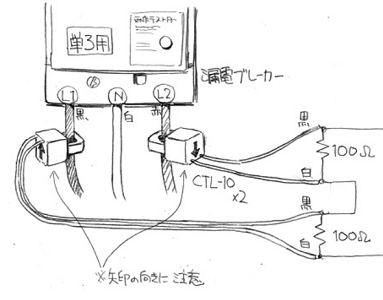
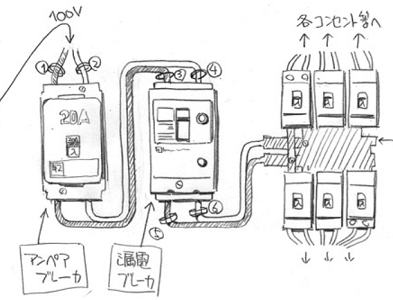
単相2線式の場合 単相3線式の場合

①～⑥のどこでも良いが，⑤か⑥が良いと思う 家庭内に200V機器が有ったりすると

※少し見にくいが漏電ブレーカのすぐ下流 こんな感じで3本出ている事がある。

各コンセント（群）に分岐している小さいブレーカに

１つずつ電流センサを設けても良い。



（情報元）<http://siesta.la.coocan.jp/zk/PwrChk/pwrchk.html>

##### 具体的な使い方

###### 負荷抵抗

概要

電流センサからの出力電流を，ADコンバーターで計測できるように電圧に変換する為に抵抗を使う。

だいたいセンサーのデータシートに推奨の抵抗値の記載があり，その抵抗値を使うのが一般的。

SR-3704-150Nの場合は10Ω（以下）とある。

（右図参照）

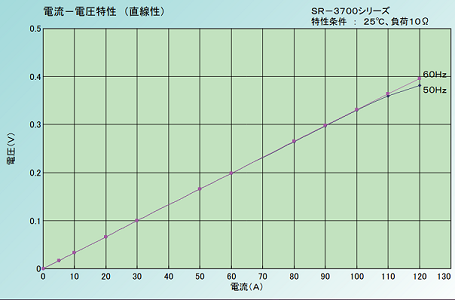
（情報元）<https://akizukidenshi.com/download/ds/sarah/SR-3704-150N_20200619.pdf>

詳細

EOUT=K・IOUT・RL/n

EOUT：出力電圧[Vrms]　K：結合係数　　IOUT：貫通電流[Arms]　　RL：負荷抵抗[Ω]　　n：電流センサのコイル巻き数

（式の情報元）<http://www.u-rd.com/technical/tech1.html>

上の式自体はそれほど重要では無いが，重要なのは一般ユーザーにはK：結合係数とn：コイル巻き数は分からない，という事。

その為，メーカーのデータシートには右図のような，指定の負荷抵抗を用いた場合の出力電圧値の記載がある。

⇒ 右図により，一次電流が60Aの時には出力電圧は0.2V(=200mV) になる事が分かる。

（図の情報元）：<https://akizukidenshi.com/download/ds/sarah/SR-3700series_data.pdf>

付随情報（やや難しい）

計算式

電流センサの変流比が分かるだけでも入力電流（一次電流）値と出力電圧値の関係は分かる。

変流比が3000:1で一次電流が60Aの場合，二次電流は ⇒ 60÷3000 = 0.02A(20mA) となる。

オームの法則により　電圧[V]= 抵抗[Ω]×電流[A]　なので10Ωの時は　10[Ω]×0.02[A]=0.2[V](200mV) になる。

※並列で繋ぐと分流されて，電流量が半分になるのでは？ 2021-07-26

補足

一般家庭で80Aもの大電流が流れる事はまずない。

一方，電流センサの値を読むADコンバーターは，0〜3.3Vと言う電圧幅に対し最大値1024の「目盛り」を割り振る。（※１）（※２）　電流センサが出力する電圧が±200mVしかない場合，せっかくの3.3V分もある測定精度を存分に使えなくなり，測定精度が悪くなってしまう。

対策：

方法１：電流センサにつける負荷抵抗を上げる。100Ωくらいまでは負荷抵抗を上げても問題ないらしい。

（情報元）<http://dededemio.blog.fc2.com/blog-entry-45.html>

負荷抵抗100Ωを用いて60A（※３）まで測定するとした時の，電流センサが出力する電圧範囲は：

　100[Ω]×0.02[A]（※４） = 2.0[V] ⇒ 測定レンジ2V。3.3Vには届かないが±200mVよりはずいぶんマシ。

方法２：オペアンプで電圧を増幅（レベルシフト）する。

詳細は後述。ただ，こちらの方法は少し難しい。

※１　マイコンで使うADコンバーターの場合大体これくらい。もちろん参考電圧や分解能（後述）に依る。

※２　負電圧に対応したADコンバーターの場合。非対応のADコンバーターの場合はオペアンプなどを使ってレベルシフトして，3.3Vの真ん中，1.65Vを中心に0V〜3.3Vの範囲を使う。

※３ 実効値40A交流のピーク電流が56.57[A]なので60Aくらいまで対応したい。　式：40[A]×√2 ＝ 56.57[A]

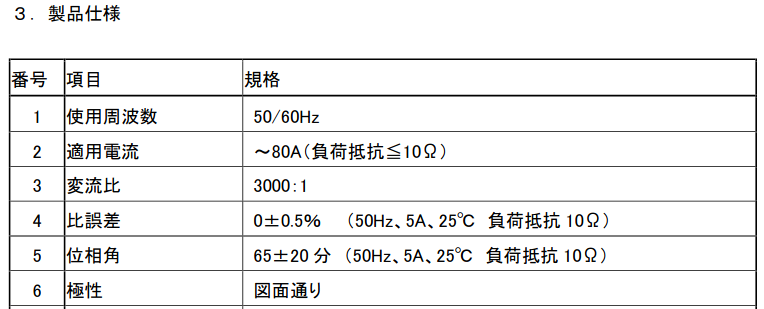
※４ 変流比が3000:1な為。　計算式：60÷3000=0.02A(20mA)

##### カタログ（電流センサ）

|  |  |  |
| --- | --- | --- |
| 商品情報 | 参考写真 | 備考 |
| SR-3704-150N  取扱店  秋月電子通商  コード: P-08960  [Webリンク](https://akizukidenshi.com/catalog/g/gP-08960/)  値段　￥980 |  | 80Aまで測定可能。  変流比　3000:1  負荷抵抗　10Ω  直流測定不可。 |
| 取扱店  switch science  コード: 813  [Webリンク](https://www.switch-science.com/catalog/813/)  値段　￥1,045 |  | 5Aまでしか測定できない。  変流比　2000:1  負荷抵抗　800Ω（内臓らしい）  直流測定不可。  （参考）<https://wiki.seeedstudio.com/Grove-Electricity_Sensor/>  元々＄5らしいので，もっと安く買えるところがあるかも？ |

##### データシート（電流センサ）

SR-3704-150Nを例にする。 <https://akizukidenshi.com/download/ds/sarah/SR-3704-150N_20200619.pdf>



使用周波数

文字通り使用できる周波数。日本は富士川（静岡県）と糸魚川（新潟県）を境に西側は60Hz，東側は50Hz。

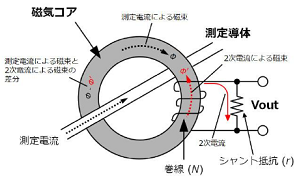
適用電流

文字通り測定可能な電流値の範囲。

家庭用電源はAC100Vなので：　80A ⇒ 100[V]×80[A] =8000[W](8kW) 　まで測定できる事になる。

電力[w] = 電圧[V]×電流[A]　　また，電流量[A] = 電力[W]÷電圧[V]　⇒（例）1500W÷100V=15A

（補足）家庭では40Aも測れれば十分すぎるので，実際には80Aも必要ない。



変流比（Transformation Ratio）

一次電流（測定電流）と二次電流（出力電流）の比の事。

例えば変流比が1000:1の電流センサで10Aを測定する時，出力電流量は0.01[A] (10mA) となる。

計算式：10[A] ÷1000（比率）= 0.01[A]

（参考）<http://www.u-rd.com/technical/tech1.html>

極性

直流を測定する時は，クランプ内に入れたケーブルの電流の流れる方向をセンサが指定する方向に合わせる必要がある。交流を測定できるセンサの場合は間違えても全く問題ない（はず）。

他の項目は理解できなくても，とりあえず使ってみる分には問題ない事が多い。（と思う 2021-07-06）

##### 資料（電流センサ）

詳細<https://tmi.yokogawa.com/jp/library/resources/measurement-tips/basic_knowledge_of_ac_power_and_mechanism_of_power_measurement_equipment/>

#### オペアンプ

##### 概要

通常ADコンバーターには負電圧（マイナスの電圧）を入れる事ができない。

CT電流センサからの出力は，例えば±200mVといった負電圧を含んだ範囲となる為，そのままではADコンバーターで測定する事ができない。

対策は以下のうちのいずれか

１　負圧を印加できるADコンバーターを使用する。 **【 この方法が１番お勧め。】**

参考ADコンバータ：<https://akizukidenshi.com/catalog/g/gK-13838/>

２　オペアンプによるレベルシフト（電圧の底上げ）を行う。

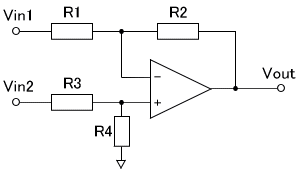
３　電流センサのマイナス側に，例えば1.65Vを印加する。

*理屈としてはこれで良いはずだが，なんか問題がある気がしてモヤモヤする* 2021-07-25

以下は　【 ２　オペアンプによるレベルシフト 】を選択した場合の資料。

それ以外の場合は読み飛ばして良い。

##### 要点

オペアンプを使った「加算回路」で電圧を底上げする。

下記サイトの「減算（差動回路）」を参考に

<http://okazu.air-nifty.com/blog/2013/03/op-325d.html#diff>

この回路の出力は以下の式のようになる

VOUT = (Vin2 – Vin1) × (R2 / R1)

具体的には

仮に，電流センサからの出力電圧±1VをVin1に入力，R1：R2＝1:1とした時に，Vin2に1.65Vを印加。

　Vinが-1Vの時は　{ 1.65-(-1) } ×1 =2.65 Vinが1Vの時は　{ 1.65-(1) } ×1 = 0.65V

⇒ 1.65Vを中心に±1V変化する**反転波形**（プラスの時低く，マイナスの時高い）が出力される。

この時のVin2に入れる1.65Vは抵抗を使った分圧で良い。（レールスプリッタなどは使わないで良いらしい。）

また，せっかくオペアンプを使うので，ADコンバーターに合わせてR1, R2でゲインを調整すると良い。

（但しR1：R2 = R3：R4が良いらしい；まだ完全には理解できていない。2021-08-14）

また，Vin1の直後にハイパス用のコンデンサを入れると良いらしい

CIN1[F] = 1 ÷｛2π×fc×（R1/R2）｝ ※fc=カットオフ周波数

（情報元）<http://www.picfun.com/f1/f08.html>　(4) 交流増幅回路

電流センサの出力値からマイナスを取り除く方法。いくつかある：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 番号 | CTセンサ  負荷抵抗 | オペアンプ  回路 | 加算電圧と  分圧抵抗 | オペアンプ  回路抵抗 | 最終的な  出力 |
| １ | 10Ω  （±200mV） | 差動減算 | +200mV  330Ωと5.1kΩ | 10kΩと75kΩ | 0～3.0V |
| ２ | 75Ω  （±1500mV） | 差動減算 | +1.5V  1.2kΩと1kΩ | 1:1であれば何でも可 | 0～3.0V |
| ３ | 10Ω  （±200mV） | 非反転加算 | +200mV  330Ωと5.1kΩ | 10kΩと140kΩ | 0～3.0V |
| ４ | 10Ω  （±200mV） | 差動減算 | +236mV | 10kΩと56kΩ | 0.201～2.44V |
| 5 | 51Ω  （±1020mV） | 差動減算 | +1.4V | 1:1であれば何でも可 | 0.20〜2.4V |

方法１

SR-3704-150Nのマニュアルに負荷抵抗は10Ω以下と書いてあったのでそれを忠実に守ることに焦点をおいている。

方法２

負荷抵抗は10Ω以下という条件を無視すれば，負荷抵抗で電圧をある程度上げておくのが一番楽。

（SR-3704-150Nには実は100Ωくらいまで付けられる，という情報もある）

方法３

この方法以外は電流センサからの信号をオペアンプの反転入力端子（マイナス側）に入れているため，例えば-200mVは+3Vに，+200mVは0Vにと上下反転して出力されてしまう。そこでどうにかして非反転入力端子（プラス側）に入力できないか，と考えた方法。

但し，オペアンプに用いる抵抗値の計算など，却ってめんどくさいと感じた。

方法４

入力範囲が0.2〜2.4Vのオペアンプの場合。まずオペアンプのゲインを算出。±200mV，つまり400mVの範囲を0.2〜2.4つまり2.2V範囲に増幅したいので，増幅率は2.2/0.4=5.5。オペアンプの抵抗値は片方を10kΩで考えると良い（らしい）ので10kΩと55kΩとなる。ただし55kΩは存在しないので56kΩとする。

更に，最終的な電圧の最低値を0.2(200m)Vにしたい。

　VOUT = （Vin2 – Vin1） × (R2 / R1)　⇨単純に（Vin2 – Vin1） を5.6倍した時に200mVになる値を考える。

200/5.6=35.714...mV　で（Vin2 – Vin1）が最低値になるのはVin1=200mVの時なのでVin2=235mV

計算はめんどうだが，オペアンプに入力制限がある場合はこの方法を取るしかない。

方法５

方法４の計算がめんどくさいので，やはり電流センサの負荷抵抗値を上げてやろう，という発想。

オペアンプの抵抗比でゲインを微妙に調整できる為，この方法が良いのかもしれない。

それぞれWebツールでシミュレートしてみると良い。　https://www.falstad.com/circuit/circuitjs.html

2021-08-16 テストの結果

Inn 0 40.3 141.1 161.3

Inp 217.5 217.5 217.5 217.5

OUT 1031 842 371.8 277.0

計算上 1022.25 832.84 359.08 261.14

差異

※この時のバイアス電流（OUT-GND：64.1mA）

##### カタログ（オペアンプ）

|  |  |  |
| --- | --- | --- |
| 商品情報 | 参考写真 | 備考 |
| LM358N  取扱店  秋月電子通商  コード: I-15053  [Webリンク](https://akizukidenshi.com/catalog/g/gI-15053/)  ￥20 |  | ２回路　単電源　バイポーラ　　※両電源でも使える  電源電圧（Vcc）：3～32V（±1.5～16V）  同相入力電圧：〜(Vcc-1.5)V  　出力電圧：～5V（Vcc5V時）  単電源では有名なオペアンプらしい。  但しVcc=3.3の時，1.8Vまでしか入力できない点に注意 |
| MCP602-I/P  取扱店  秋月電子通商  コード：I-16192  [Webリンク](https://akizukidenshi.com/catalog/g/gI-16192/)  ￥50 |  | ２回路　単電源　CMOS　レールtoレール  　電源電圧(Vcc)：2.7~6.0V  同相入力電圧：(Vss-1.0)〜(Vcc+1.0)V  　出力電圧：(Vss-0.3)〜(Vcc+0.3)V　※Vssは通常0V  　SR：2.3V/us |
| NJU7062  コード：I-03755  [Webリンク](https://akizukidenshi.com/catalog/g/gI-03755/)  ￥80 |  | ２回路　単電源  ※NJU7022＋オフセットピンがあるタイプ  NJU7022の情報が無いが？？ |
| NJM2747-Z  コード: I-09117  [Webリンク](https://akizukidenshi.com/catalog/g/gI-09117/)  ￥100 |  | ４回路　単電源　CMOS　フルスイング（レールtoレール）  電源電圧(Vcc)：0～15V  同相入力電圧：0〜15V  　出力電圧：0.1～VCC-0.25  SR：3.5V/us |
| NJU7044D  コード: I-02371  [Webリンク](https://akizukidenshi.com/catalog/g/gI-09117/)  ￥150 |  | ４回路　単電源　CMOS　フルスイング（レールtoレール）  電源電圧(Vcc)：2.2～5.5V  同相入力電圧：0〜7V  　出力電圧：0.1～VCC-0.05  SR：0.8V/us |

補足　今回は単電源で使うのでVss≒GND つまり0Vと思えば良い。

##### データシート（オペアンプ）

（例）NJM2902N <https://akizukidenshi.com/catalog/g/gI-06329/>

電源電圧

別名：Suppy Voltage 記号：Vcc，V+ など

差動電源電圧

別名：Differential Input Voltage 記号：VID など

差動電圧で使う時の入力可能なVcc-Vss(≒GND)の範囲。

入力電圧

別名：(広)同相入力電圧(範囲)，Input Voltage 記号：VID，VOH（Hi）， VOL（Lo） など

入力可能な電圧範囲。電源電圧に大きく影響を受ける。

（最大）出力電圧

別名：Output Voltage Swing 記号：Vopp など

電源電圧に関係なく，出力できる電圧の限界値

入力オフセット電圧

別名：バイアス電圧

入力0Vの時に印加される電圧成分。この分の電圧をオフセットしないと測定に影響が出る。

スルーレート

別名：SR

時間当たりの出力電圧の変化度合い。0.6V/μSecくらいが標準。

##### 資料

レベルシフター 入力信号と同じ波形で，異なる電圧値を出力する回路。

Webブラウザで動く電子シミュレータ <https://www.falstad.com/circuit/circuitjs.html>

（オペアンプはシミュレータが無いと理解しにくい）

#### 電力値計算

##### 概要

・下の「アンペア計算方法」を参考に，現在の電流値（瞬時値）を算出。

・Root Mean Square（二乗平均平方根※１）の為に　電流値2　の値を加算し続ける。

・どこかのタイミングで平均値を算出。その値の平方根を取る。（Arms ；電流実効値が算出される。）

・Arms ；電流実効値　×　電圧（家庭用なので100V）= 電力値[W]　となる。

※１　なぜそんな計算が必要なのか，の参考資料：<https://cognicull.com/ja/nzok9whj>

アンペアの計算方法

ADコンバーターの値 → 測定電圧[V]

　ADコンバータ値 ＝ ADコンバータのMAX値（例えば1024）の時に、測定電圧が参考電圧（例えば3.3V）と等しいはずなので

　測定電圧[V] = ADコンバーター値　×｛ 参考電圧 ÷ ADコンバータ分解能値(MAX値) ｝

測定電圧[V] → 一次電流値[A]

　二次電流値に変流比を掛ければ一次電流値になる。二次電流値 ＝測定電圧値[V] ÷ 抵抗値[Ω]（オームの法則より）なので：

一次電流量 ＝｛ 測定電圧値[V] ÷ 負荷抵抗[Ω] ｝× 変流比

一気に計算する為の変換係数　＝

この係数を算出しておけば ⇒ 　ADコンバータ値 × この係数　＝　一次電流量（測定電流値）　となる。

（補足）

普通の平均と，二乗平均平方根では値が結構変わってくる。

1+3 = 4 平均化　⇒ 4÷n(=2) = 2 結果：2

12+32=10 平均化　⇒ 10÷n(=2) = 5 平方根⇒2.2362 結果：2.23

最初は疑問に感じたが，どうやら標準偏差を取るような統計的な手法でもあるらしい。2021-07-15

#### 各種資料

##### ADS1015

###### 仕様まとめ

|  |  |  |  |
| --- | --- | --- | --- |
| 項目 | 記号 | 内容 | 備考 |
| 分解能 |  | 12bit | 11bitと正負1bitなので，数値の範囲は±2048 |
| 電源電圧 | VDD | -0.3～7V |  |
| 入力電圧 | AIN | GND-0.3～VDD+0.3 |  |

I2CのアドレスはADDRポートと以下のポート繋ぐ事で変更できる。

|  |  |  |  |
| --- | --- | --- | --- |
| 相手 | アドレス | 2進数 | 備考 |
| GND | 0x48 | 0100 1000 | デフォルト |
| VDD | 0x49 | 0100 1001 | 変える必要が出た時は，まずここを検討する。 |
| SDA | 0x4A | 0100 1010 | SCLがLOWになって100nSec間は，SDAをLOW状態に保つ必要がある。面倒なので，この選択は**止むを得ない時だけ**にする。 |
| SCL | 0x4B | 0100 1011 |  |

情報元：　[公式資料](https://www.tij.co.jp/jp/lit/gpn/ads1015)　8.5.1.1　I2C Address Selection　より

通信速度

SCLの通信速度（周期）は以下の３つの内のいずれかを選べる。

Standard Mode 0.01M(10k)？～0.1 M(100k )Hz

Fast Mode 0.01M(10k)～0.4 M(400k )Hz

High-Speed Mode 0.01M(10k)～3.4MHz

StandardやFastモードを使う時は特に設定は必要ない（らしい）。

START conditionに続いて0000 1xxx という特殊レジスタに

xxxの部分はHs-capable master特有bit⇒意味不明

情報元：　[公式資料](https://www.tij.co.jp/jp/lit/gpn/ads1015)　8.5.1.3　I2C Speed Modes　より

2byte目にスレーブアドレス 2021-08-13追加情報

第一ビットに特別データを追加するだけ？ https://www.i2c-bus.org/highspeed/

HSPI はhs mode のことでは無い。

###### 通信プロトコル

先にマニュアルが示す通信プロトコルは以下のような形

|  |  |  |  |
| --- | --- | --- | --- |
| byte番号 | 内容 | 具体例(2進) | 備考 |
| 第1byte | I2Cアドレス とR/ | 1001 0000 | R/が第1bitでI2Cアドレスは1bit左シフト。 |
| 第2byte | アドレスポインタ | 0000 0001 | 通信対象のレジスタを選択する。(Config) |
| 第3byte | データMSB | 1000 0101 | MSBが先。 |
| 第4byte | データLSB | 1000 0011 | データが1byteの時は未使用。 |

情報元：　[公式資料](https://www.tij.co.jp/jp/lit/gpn/ads1015)　9.1.7　Quickstart Guide　より

但し，pythonにせよC言語にせよ第1byteの「I2Cアドレスとread/write」の部分は通信モジュールがやってくれるので意識する必要は無い。実質には**常に3byteの通信**，と思っておけば良い。

アドレスポインタ（Address Pointer Register）はbit[1:0](2bit) のみを使用して残りの bit2～bit7は常に0

|  |  |  |
| --- | --- | --- |
| 値 (2進数) | 対象になるレジスタメモリ | 備考 |
| 00 | Conversion register | AD変換された値。 |
| 01 | Config register | 各種設定。 |
| 10 | Lo\_thresh register | エラー下限の事。今回は使わない。 |
| 11 | Hi\_thresh register | エラー上限の事。今回は使わない。 |

エラー上下限値について：この上下限値を超えるとALERTピンがオンする。（2021-07-30未確認）

未確認事項：

・第8bitはread/write状態を示す？ 8.5.1.3の4行目辺り　0:write 1:read

・General Callのアドレスは0x00。その後0x06 を受信でリセット？　8.5.1.2の2行目

###### Config

太字がデフォルト（0x8583；0b1000 0101 1000 0011 が初期値）

|  |  |  |  |
| --- | --- | --- | --- |
| bit | 名称 | 詳細 | |
| 15(1bit) | OS (Operational State)  運転状態 | 書き 0：なにもなし 1：変換開始  読み0：変換処理中  **1：何もしていない** | |
| 14:12(3bit) | MUX  対象chの選択 | シングル  100　ch0  101　ch1  110　ch2  111　ch3 | 差動  **000　プラスch0 マイナスch1**  001　プラス ch0 マイナスch3  010　プラス ch1 マイナスch3  011　プラス ch2 マイナスch3 |
| 11:9(3bit) | PGA  内蔵オペアンプのゲイン設定 | 000 : FS = ±6.144V(2/3倍)　精度：3mV  001 : FS = ±4.096V(1倍) 　精度：2mV  **010 : FS = ±2.048V(2倍)** 　精度：1mV  011 : FS = ±1.024V(4倍) 　精度：0.5mV  100 : FS = ±0.512V(8倍) 　精度：0.25mV  101，それ以外: FS = ±0.256V(16倍) 　精度：0.125mV | |
| 8(1bit) | MODE | 0:連続変換　　　**1:シングルショット** | |
| 7:5(3bit) | DR(Data Rate) | 000：128SPS　　001：250SPS　　010：490SPS　　011：1600SPS  **100：1600SPS** 101：2400SPS 110：3300SPS 111：3300SPS | |
| 4(1bit) | COMP\_MODE | 0がデフォルト　（良く分からないのでデフォルトのまま） | |
| 3(1bit) | COMP\_POL | 0がデフォルト　（良く分からないのでデフォルトのまま） | |
| 2(1bit) | COMP\_LAT | 0がデフォルト　（良く分からないのでデフォルトのまま） | |
| 1:0(2bit) | COMP\_QUE | 11がデフォルト　閾値データの使用/未使用。使うならその方法。 | |

情報元：　[公式資料](https://www.tij.co.jp/jp/lit/gpn/ads1015)　8.6.2 Conversion Register，8.6.3　Config Register　より

（参考）<http://zattouka.net/GarageHouse/micon/circuit/A_D2.htm>

（全体の参考資料）<https://qiita.com/yoroyasu/items/d4eea6ed952a589ca780>

通信の仕方が分からなかった時の分析資料（2021-07-27）

（分析対象）<https://github.com/adafruit/Adafruit_ADS1X15/blob/master/Adafruit_ADS1X15.cpp>

readADC\_SingleEnded　の分析をすると良い

// 以下は Adafruit\_ADS1X15.h で設定されているものの抜き出し

#define ADS1X15\_REG\_CONFIG\_OS\_SINGLE 0x8000

#define ADS1X15\_REG\_CONFIG\_MUX\_SINGLE\_0 0x4000 // ch0

#define ADS1X15\_REG\_CONFIG\_MUX\_SINGLE\_1 0x5000 // ch1の場合

#define ADS1X15\_REG\_POINTER\_CONFIG 0x01

private uint8\_t buffer[3]; // つまりbyteの配列

// まず初期化（ここから Adafruit\_ADS1X15.cpp）

uint16\_t config =

ADS1X15\_REG\_CONFIG\_CQUE\_NONE | // 0x0003 Disable the comparator (default val)

ADS1X15\_REG\_CONFIG\_CLAT\_NONLAT | // 0x0000 Non-latching (default val)

ADS1X15\_REG\_CONFIG\_CPOL\_ACTVLOW | // 0x0000 Alert/Rdy active low (default val)

ADS1X15\_REG\_CONFIG\_CMODE\_TRAD | // 0x0000 Traditional comparator (default val)

ADS1X15\_REG\_CONFIG\_MODE\_SINGLE; // 0x0100 Single-shot mode (default)

// ⇒ config = 0x0103

config |= m\_gain; // 設定値なので放置

config |= m\_dataRate; // 同上

switch(channel) // 中略

config |= ADS1X15\_REG\_CONFIG\_MUX\_SINGLE\_0 // =0x4000　つまり ⇒　0x4103

config |= ADS1X15\_REG\_CONFIG\_OS\_SINGLE; // =0x8000　つまり ⇒　0xC103　　※0x8 + 0x4 = 0xC

// このメソッドで書き込む

writeRegister(ADS1X15\_REG\_POINTER\_CONFIG, config);

実際の書き込みメソッド

void Adafruit\_ADS1X15::writeRegister(uint8\_t reg, uint16\_t value) {

buffer[0] = reg; // 第1byte　= ADS1X15\_REG\_POINTER\_CONFIG = 0x01

buffer[1] = value >> 8; // 第2byte 実際のデータの第1byte

buffer[2] = value & 0xFF; // 第3byte 実際のデータの第2byte

m\_i2c\_dev->write(buffer, 3);

}

これにより通信フォーマットは

第1 byte 今回の通信でアクセスする対象レジスタを指定。

第2 byte 対象のch番号と

第3 byte 実際の設定データ

となっている事が読み取れる。

###### コーディング資料

<https://github.com/adafruit/Adafruit_ADS1X15/>

##### MCP3004

Max 10 kHzとあるが，100kHz設定でも通信できた。

電源電圧2.7-5.5V　なのでVrefも2.7以上

Throughput Rate VDD2.7V時に75ksps

##### タイマーに関して

###### ラズパイでやる難しさ

pythonが高速処理向きでない※ 為にC言語でのコーディングが必要となる事。

※目安として500Hzよりも早いものはpythonでは難しいらしい。　ソース不明（リンク切れ）なので500Hzの妥当性は疑問。

但し，何kHz単位とかの超高速処理ではないし，大体の電力測定で良いと割り切る事もできるので，pythonとC言語のソースを両方載せたいと思う。

（補足）

厳密に言えばOSが載っている環境では，C言語を使った所で正確な定周期処理は不可能。

いっその事ラズパイpico，Arduino，ESP WROOM など純粋なマイコンでやるべき分野で，初心者の為に，とラズパイにこだわる事でむしろ難しくなるのかもしれない。

資料

pythonの定周期処理の方法：<https://qiita.com/montblanc18/items/05715730d99d450fd0d3>

pythonの定周期処理の限界に関する資料：<https://teratail.com/questions/249758>

GPIOの割り込み限定だけど，これも面白そうな情報（蛇足）：<https://team-ebi.com/arc/290>

###### マルチスレッド

通常プログラム（≒CPU）は細かく見ると処理を１つずつしか実行できない。その為，複雑な処理は細分化されて「小さな処理の行列」を作って１つずつ処理される。これの一連の処理の列を「スレッド」と言う。

定周期処理では，例えば2.0mSecという「一定時間待ってからアナログ値を読む」という処理を繰り返すが，この「待つ」と言うのも立派な処理で，待っている間はCPUは他の処理ができなくなってしまう。

その為，待つ処理を専用で行うスレッドと他の処理を行うスレッドと２つ（以上）のスレッドに分けて実行する。これをマルチスレッドと呼ぶ。

定周期処理のような時間が絡む処理はマルチスレッドを使わないと実現が難しい。

ちなみにマルチスレッドはシングルコア（CPUが1つ）でも実装が可能。

##### 用語（総合）

MSB；Most Significant Bit

最上位ビット。例えば0x0A(=10)を2進数にすると1010だが，一番左の1がMSB

LSB；

最下位ビット。例えば0x0A(=10)を2進数にすると1010だが，一番右の0がLSB

### ESPを使う場合

#### 概要

ESP32はADコンバーター内臓。内蔵なのでシリアル通信の必要が無い上（※）定周期処理はラズパイよりはるかに簡単な為，総合してラズパイよりもESP32の方が楽になる。

（※）実際にはCPU ⇔ 内部ADコンバーターはI2C通信をしているとかどっかで読んだ。2021-07-15

但し，ESP32内臓のADコンバーターの精度は一般的なADコンバーターより荒いらしい。

また，そのADコンバーター部分の仕様が**やや難解**ではある。

（参考）ESP32では±23～60mVの誤差 （市販品10bitのADコンバーター [ 誤差1LSBの品] 約±6.5mV）

先に分かり難い部分を箇条書きにする

・実際に使えるpinを見分ける必要がある。

・パラメータattenuationで測定レンジが変化する。

#### Arduino Studio

##### 使い方

（編集中）

改行したい場合はこのうちのいずれかを使う

Serial.println('hello world');

Serial.print('hello world');

Serial.print('\n');

##### トラブルシューティング

###### 共通

‘LED\_BUITIN’ was not declared in this scope

⇒ LED\_BUILTINを定義していない，という意味のエラー。

サンプルプログラムの「blink」を試していると思われるが，このサンプルではLED\_BUILTINを定義していない。以下の用に定義してやる。

#define LED\_BUILTIN 4 // ピン番号はこのように指定できる。

// ※{}内では無いところで，上の方に書くと良い

void setup() { // 以下略

もしくはLED\_BUILTINの部分を全て具体的な数値（4とか）に変える。

stray '\342' in program

⇒ ASCII引用符の代わりにUnicode引用符 を用いている。「 “ 」などを「 " 」に直す　など。

C++のソースコードにUnicode文字を使ってはいけないらしい。

「WiFi.h」に対して複数のライブラリが見つかりました

⇒ 同名の.h（ヘッダ）ファイルがあり，どちらを使えばよいのかわからない　という意味。

ただ，このエラーは他のエラーに誘発されているだけ　というパターンが多いらしいので，まず他の問題を疑った方が良いらしい。

Brownout detector was triggered

電圧低下

###### Linux

exec: “python”: executable file not found in $PATH

⇒ pythonの実行ファイルへのパスが通っていない。

which python // まずはこのコマンドの実行結果を確認

⇒パスが表示されない：python(2)がインストールされていないので apt install python-is-python2

⇒表示される：export でそのパスを追加する　例：export PATH=$PATH:/usr/bin/python

ImportError: No module named serial

⇒ python(2) にpyserialが入っていない。Arduino Studio内部ではシリアル通信にpyserialを使っているらしい。しかもpython3では無い。

sudo pip -m install pyserial

could not open port /dev/ttyUSB0 permission denied

⇒ /dev/ttyUSB0 に読み書き許可が無い

ls /dev/ttyUSB0 -l # 現在の権限を確認

sudo chmod o+rw /dev/ttyUSB0 # その他ユーザーに読書き権限を付与（所有者はroot）

sudo chmod 777 /dev/ttyUSB0 # めんどうならばこれでも良い

###### Windows

##### コーディング・レファレンス

公式レファレンス（英語）

<https://www.arduino.cc/reference/en/> 英語，オランダ語，ポルトガル語しか対応していない。

<https://www.arduino.cc/reference/en/libraries/> ライブラリのレファレンス

<https://github.com/espressif/arduino-esp32> サンプルスケッチのgithub版

有志の人が作ったArduinoStudioのコーディング・レファレンス：

レファレンス１：<http://www.musashinodenpa.com/arduino/ref/index.php>

レファレンス２：<https://garretlab.web.fc2.com/arduino_reference/language/functions/index.html>

#### STEP1,2 A 内部ADコンバータを使う場合

##### 概要

ESP内臓の12bitのADコンバーターを使う

|  |  |
| --- | --- |
| 長所 | 短所 |
| ・シリアル通信の配線，プログラム実装が必要ない為，簡単。 | ・一般的なADコンバーターより測定精度が悪い。  ・関連する仕様が複雑で，理解しにくい。 |

##### 必要なもの

抵抗，可変抵抗

2kΩ，200Ω可変 簡易0〜200mV電源用 安定化電源などがあれば必要なし

　10kΩ，56kΩ オペアンプ5.6　増幅用

　390Ω，5.1kΩ 3.27→232mV用

オペアンプ できるだけレールtoレール（フルスイング）のもの

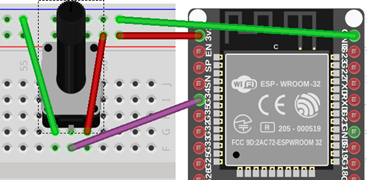
テスター 電圧を測る必要がある

##### A-1　ADコンバーターのテスト

概要

ESP内臓の12bitのADコンバーターのテスト

|  |  |
| --- | --- |
| 長所 | 短所 |
| ・シリアル通信の配線，プログラム実装が必要ない為，簡単。 | ・一般的なADコンバーターより測定精度が悪い。  ・関連する仕様が複雑で，理解しにくい。 |

配線

右図はFritzingで描いた配線図。

GPIO34番にアナログ電圧を入れる事を想定。

ここでは10kΩの可変抵抗（ボリューム）を使用する。

初期設定のままでも3.3Vの入力は問題ない。

（3.9Vまで入力OKらしいが本当か？）

使うピンは以下のいずれかが良さそう

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| pin# | 名称 | GPIO# | A番号 | 備考 |
| 4 | ADC1\_CH0 | 36 | A0 |  |
| 5 | ADC1\_CH3 | 39 | A3 |  |
| 6 | ADC1\_CH6 | 34 | A6 |  |
| 7 | ADC1\_CH7 | 35 | A7 |  |

（参考）ADコンバーター

サンプルプログラム

void setup() {

**Serial**.begin(115200); // Serial.println()を行う為に必要

pinMode(34, ANALOG); // この行は必要無い。後学の為に残す。

}

void loop() {

int an\_val = analogRead(A6); // analogRead(34) としても結果は全く同じ

**Serial**.println(an\_val); // シリアルモニタに値を出力

delay(500); // 500mSec待つ → 最終的に1mSecでも問題ないか確認

}

（補足）シリアルモニタは　ツール → シリアルモニタ　又は　Ctrl+Shift+M で表示される。

（補足）Arduino Studio内で使用するピン番号は，GPIOの番号で良いらしい。

##### A-2　分圧で0〜200mVを作る

概要

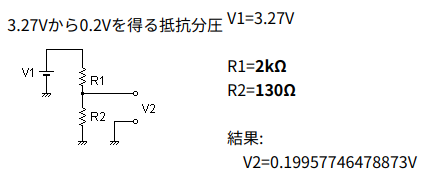
次ステップでオペアンプの動作テストを行う為に，電圧を自由に変更できる電源を簡易的につくる。

具体的には，電流センサが出す事になる−200〜200mVの範囲で変更できる電源が欲しい。

簡易的に可変抵抗（ボリューム）とカーボン抵抗による分圧で0〜200mVの電源を作る。

（補足：この方法で作る電源は，精密用途では全く使い物にならないので他の用途では使わないように）

元電圧はESP32に供給している3.3Vを使う。但し電圧は3.3Vピッタリでは無く，3.27Vなど少し低い場合があるので注意。テスターでしっかり電圧を確認する。

今回は電圧が3.27Vだったと仮定して

→ 抵抗値の選定　2kΩと130Ω。

3.27 × ｛ 130 ÷ （2000+130）｝=0.19957 ≒200mV

抵抗値の計算はここが使いやすかった：

<http://sim.okawa-denshi.jp/teikokeisan.htm>

このR2の抵抗を可変抵抗にして電圧を調整したいが，130Ωの可変抵抗は売ってないと思うので，この場合は200Ωの可変抵抗を購入する。（今回は抵抗値が少し大き目のほうが良い）

この抵抗値はきっちりでなくて良い 2021-08-16

→200Ωと2.7kΩがちょうど良い？225mVまで

##### A-3　オペアンプによるレベルシフト

ESP32のアナログ入力範囲は150～2450mV（atten=3；デフォルトの時）。

⇒2.3Vの範囲が使える。

⇒ 1.3Vを中心とした片範囲1.15Vになる

⇒ 200mVを1150mVに増幅。5.75倍

・抵抗は10kΩと56kΩで増幅率5.6倍とする

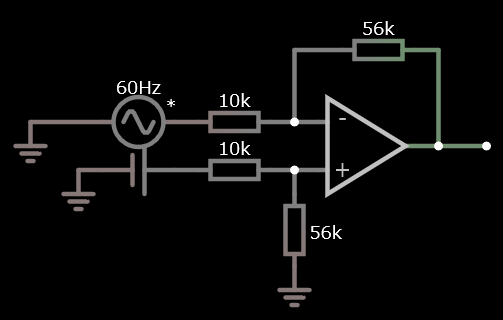
・1.3Vを中心とした片範囲1.12Vに修正→　180〜2420mVになる

⇒ 最低値のときに180mVになるレベルシフト

・シフト電圧200+32.142(=180÷5.6)=232.142mV

・3.27Vからは5.1kΩと390Ωで作る → 3.27×｛390 ÷（5100+390）｝=232.295mV

⇒最終的な電力出力は180.85mV〜2420.852mVとなる。

（1300mVが真ん中）

右図は [Webシミュレータ](https://www.falstad.com/circuit/circuitjs.html) による配線図

非反転加算回路

Vin（マイナス；Vinn 側） ±200mV交流

Vref（プラス；Vinp 側） +232.295V直流

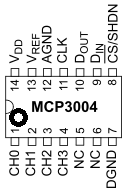
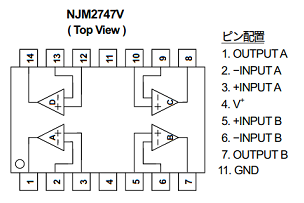
補足：ラズパイはしっかりと3.3V出ているが，ESP32は3.27Vだった。

補足：Vref計算がめんどくさい。最低電圧を180にしたいので（かつ，ゲインは5.6倍で決定済み）

（Vref – 200）×5.6＝180 ⇒　Vref = (180÷5.6)＋200 ⇒ 232.1428...

配線

（編集中）

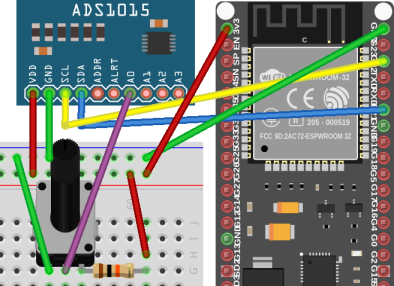


#### STEP1,2 B DS1015(I2C)

###### ADコンバーターのテスト

ラズパイの例と同じく，AE-ADS1015への接続を想定　　<https://akizukidenshi.com/catalog/g/gK-13838/>

配線

デフォルトではSDAがGPIO21，SCLがGPIO22。

10kΩの可変抵抗を使用。

*※ADS1015は初期設定では入力電圧が±2.048Vなので，*

*10kΩ抵抗で分圧して，入力電圧を ～1.67Vにしている。*

通信開始メソッドでSDAとSCLのピン番号を指定する為， ESPではI2Cに使うピンは**実はどれでも良い**。但し，GPIO34~39には内部pull-up，pull-down 抵抗が無いので，それらを使う時は外部に10kΩのpull-up抵抗が必要。

サンプルプログラム

#include <Wire.h>

short readAnalogADS1015(int ch){

// 補足。通信は最小限にしたいが，muxによるch選択しているので，configの通信も毎回行わなければならない。

byte os=1; // 運転状態 0:何もなし 1:変換開始

byte mux=0b100; // 対象 ch の選択 　値=0b100 → ch0シングルエンド

byte pga=0b010; // オペアンプゲイン 010: ±2.048V

byte op\_mode=0; // 変換モード 0: 連続変換 1:シングルショット[def.]　modeは予約語

byte dr=0b100; // DataRate 100:1600SPS

byte cfg\_lsb = dr <<5 | 0x03; // configのlsb（低い方のbyte）

byte cfg\_msb = os << 7 | mux << 4 | pga << 1 | op\_mode; // configのmsb（高い方のbyte）

Wire.beginTransmission(0x48); // スレーブのI2Cアドレスを引数に通信を開始

Wire.write(0x01); // 1byte目：アドレスポインタ（※１）でConfigを指定

Wire.write(cfg\_msb); // 2byte目：データ1　MSB→LSBの順

Wire.write(cfg\_lsb); // 3byte目：データ2

Wire.endTransmission(); // 通信終了

Wire.beginTransmission(0x48); // 通信開始

Wire.write(0x00); // アドレスポインタ（※１）でConversion Registerを指定

Wire.endTransmission(); // 通信終了

short anaVal; // short 符号付き16bit数値

Wire.requestFrom(0x48, 2); // I2Cアドレス0x48から2byteを要求

while(2 <= Wire.available()){

anaVal = Wire.read() << 4;

anaVal |= Wire.read() >>4;

}

return anaVal/16; // MSBが+/-符号ビットになっている為，シフトでなく割り算で対応

}

void setup(){

**Serial**.begin(115200); // Serial.println()を行う為に必要

pinMode(21, INPUT\_PULLUP); // 内部プルアップ抵抗を使用（デフォルトなので不要）

pinMode(22, INPUT\_PULLUP); // 内部プルアップ抵抗を使用（デフォルトなので不要）

Wire.begin(); // (SDA, SCL) だが，21,22の場合は引数不要らし

Wire.setClock(400000); // 通信速度変更。できるだけ高速で通信したい。

}

void loop(){

short analogVal=readAnalogADS1015(0); // ch 0 のアナログ値を読む

**Serial**.println(analogVal); // シリアルモニタに値を出力

delay(500); // 500mSec待つ→最終的に1mSecでも問題ないか確認

}

※１ ADS1015の[通信プロトコル](#_通信プロトコル)を参照。

（公式情報）<https://www.arduino.cc/en/Reference/WireRead>　⇒ 但し情報不足。

（参考資料）<https://nobita-rx7.hatenablog.com/entry/27669296>

###### 電流センサのテスト

概要

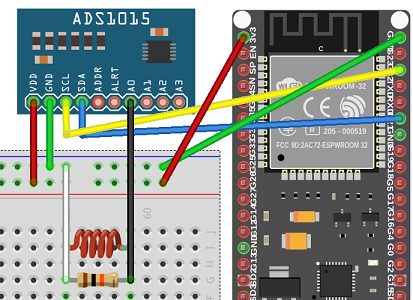
ラズパイの時と同様

・電流量の瞬時値を返すread\_ampere(int ch)と言うメソッドを作る

・まずは1mSec毎で適当な交流電流のアナログ値を測定してみる

補足

rms電流値への計算はSTEP3で行い，ここでは電流量の瞬時値を取るだけにする。

配線

電流線センサの負荷抵抗には100Ωを使用

補足

fritzingに電流センサが無かったので，コイルを利用した。

サンプルプログラム（抜粋）

#include <Wire.h>

const byte PGA=0b101;

float voltMax=2.048;

short readAmpereADS1015(int ch){

// ..省略 　途中までは　readAnalogADS1015　と同じ

short anaVal; // 16bitの数値

Wire.requestFrom(0x48, 2);

while(2 <= Wire.available()){

　anaVal = Wire.read() << 8;

　anaVal |= Wire.read();

}

anaVal/=16;

float voltVal = (anaVal / 2048.0f) \* voltMax; // 電圧値の算出

float ampereVal = (voltVal / RESIST\_OHM) \* TRANS\_RATIO; // 一次電流量を算出

return ampereVal;

}

void setup() {

**Serial**.begin(115200);

if (PGA < 1)voltMax=6.144; // PGAの設定値によって

else if(PGA < 0b101) voltMax = 2.048 \* pow(2 ,(2-PGA)); // 値=2048の時の電圧値が異なる為

else voltMax=0.256; // プログラム内でもそれに対応する

// 以下略　（Wire.beginとsetClockを行う）

}

void loop(){

short ampereVal=readAmpereADS1015(0); // ch 0 のアナログ値を読む

**Serial**.println(analogVal); // シリアルモニタに値を出力

delay(50); // 50mSec待つ→最終的に1mSecでも問題ないか確認

}

#### STEP 1,2 MCP3004(SPI)

##### ADコンバーターのテスト

概要

ラズパイの例と同じく，MCP3004-I/Pへの接続を想定。 https://akizukidenshi.com/catalog/g/gI-11987/

※MCP3004は足が簡単に曲がるので注意！

ESP32のSPIの仕様

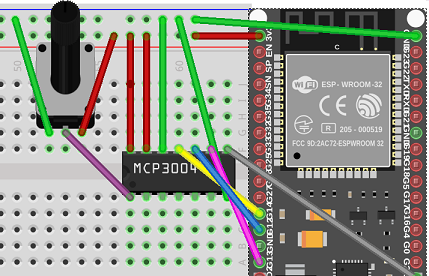
SPI, HSPI, VSPIとあり，SPIは内蔵フラッシュROMとの通信に使われている為使用不可。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SCK | MISO | MOSI | SS | HD | WP | 備考 |
| HSPI | 14 | 12 | 13 | 15 | 4 | 2 | 12:MTDI　13:MTCK |
| VSPI | 18 | 19 | 23 | 5 | 21 | 22 |  |

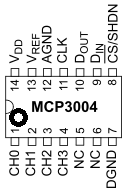
理由は分からないが，公式資料ではMISOはQ，MOSIはDと表現されている。

（参考）[https://trac.switch-science.com/wiki/esp32\_tips#SPI](https://trac.switch-science.com/wiki/esp32_tips#SPI通信)通信

（公式）esp32\_datasheet　4 Pripherals and Sensors



配線



※絵が切れてる

やりなおし

サンプルプログラム

（編集中）

#include <SPI.h>

// 通信設定 ( 通信速度, MSBが先か, SPIの通信モード )

SPISettings spi\_set = SPISettings(1000000, MSBFIRST, SPI\_MODE0);

void readMCP3004(int ch){

**SPI**.beginTransaction(spi\_set); // 通信開始

digitalWrite(15, LOW); // Slave Selectは手動で行う必要がある

**SPI**.transfer(0b11000 | ch);

byte ret\_msb = **SPI**.transfer(0); // msb受信。送信データはなんでもよい

byte ret\_lsb = **SPI**.transfer(0); // lsb受信

//byte ret[] = SPI.transfer(buffer, 3); // こんなメソッドもあるらしい　未確認 2021-08-11

digitalWrite(15, HIGH);

**SPI**.endTransaction(); // 通信終了

return ret\_msb << 4 | ret\_lsb >> 4; // 数値化

}

void setup(){

**Serial**.begin(115200);

Pinmode(15, OUTPUT);

**SPI**.begin(14, 12, 13, 15); // (SCK, MISO, MOSI, SS)

**//SPI**.setBitOrder(MSBFIRST); // SPISettingsの内容を個別設定するメソッドも存在する。

}

void loop(){

short ana\_val = readMCP3004(0);

**Serial**.println(ival); // シリアルモニタに値を出力

delay(500); // 500mSec待つ

}

2021-08-04 通信成功したらreadMCP3004(ch) というメソッドにする。

（参考１）<https://qiita.com/yoroyasu/items/3d15563947589801d34a>

（参考２）<https://blog.goo.ne.jp/namva/e/2733bdfc3ab15105cf3efabfb36ea29f>

（公式）<https://www.arduino.cc/en/Reference/SPI>（但し情報不足）

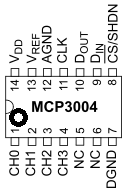
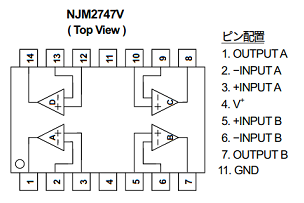
未確認情報

writeRegister(0x02, 0x2D); // データ書き込み

##### オペアンプによるレベルシフト

抵抗値などの設定はラズパイ版のものと同じにする。

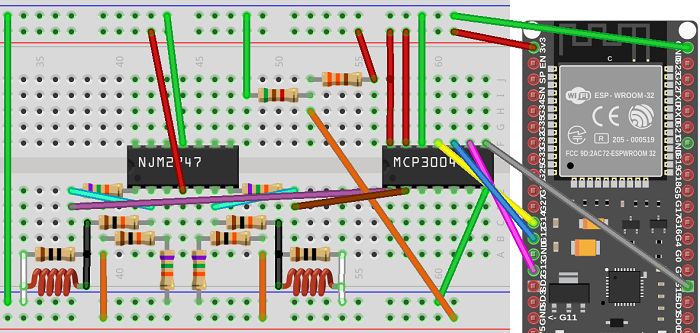
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CTセンサ  負荷抵抗 | オペアンプ  回路 | 加算電圧と  分圧抵抗 | オペアンプ  回路抵抗 | 最終的な  出力 | 備考 |
| 10Ω  （±200mV） | 差動減算 | +200mV  330Ωと5.1kΩ | 10kΩと75kΩ | 0～3.0V |  |

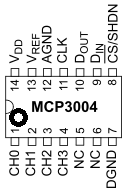
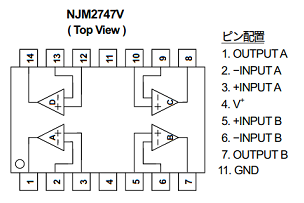


##### 電流センサのテスト

配線図

かなり複雑な回路になってしまう。





補足

電流センサが無かったので，コイルを利用した

#### STEP3. 定周期処理

##### タイマー割込み基本

まずは割込みタイマーの周期を設定する。

※このソースコードには問題があるが，理解しやすさに重点を置いている。

hw\_timer\_t \*timer = NULL; // NULLが無いとエラーになった 2021-08-09

volatile int isrCounter=0; // 割込み中にアクセスする変数にはvolatileを付ける

int secondCounter=0; // 秒数を格納する変数

void IRAM\_ATTR onTimer(){ // IRAM\_ATTR　高速RAMへの配置を指定する

isrCounter +=1; // 割込みハンドルのメソッドでは最低限の処理のみを行う

}

void setup() {

**Serial**.begin(115200);

timer = timerBegin(0, 80, true); // タイマー開始　(タイマー#，divider，カウントアップ)

timerAttachInterrupt(timer, &onTimer, true); // (タイマーハンドル，割込みメソッド名，エッジ割込み)

timerAlarmWrite(timer, 1000, true); // (タイマーハンドル，タイマーカウント数，継続タイマ)

timerAlarmEnable(timer); // タイマーアラーム（割込み）を有効化

}

void loop() {

if(isrCounter >= 1000){ // 1/1000秒割込みが1000回発生 ＝ 1秒経過

secondCounter+=1; // 秒数を保持する変数をカウントアップ

isrCounter=0; // 割込み回数　を保持する変数をリセット

**Serial**.println(secondCounter); // シリアルモニタに値を出力

}

}

タイマー分周の考え方

分周は英語ではdividerと言い　peripheral周波数÷divider = タイマー動作周波数　と，文字通り割り算（divide）でタイマー動作周波数を決める。ESP32は（デフォルトでは）peripheral周波数が80MHzなのでdividerを80にするとタイマー動作周波数は　80MHz÷80= 1MHz（=1μsec毎） となる。これを「タイマクロックは80MHzを80分周」などと表現する。

timerAlarmWriteの第２引数「カウンタ数」でタイマーのアラーム（≒割込み）の頻度を調整できる。

割り込み発生頻度 ＝ タイマー動作周波数 × カウンタ数

この例でいくと ⇒ 　　1μsec × 1,000 = 1mSec 補足1μ=10-6　1m=10-3

となり，1/1000秒(1mSec) 毎に割込みが発生する。

補足

getApbFrequency()というメソッドがあり，peripheral周波数を取得できる。例えば

divider = getApbFrequency() / 1000000(=106)

として，timerBeginの第２引数に入れれば，どんなCPUでもタイマー周期1μSecにできる。

（参考）<http://marchan.e5.valueserver.jp/cabin/comp/jbox/arc202/doc21105.html>

##### タイマー＋メモリ管理

サンプルプログラム（抜粋）

簡単に解説すると，1mSec毎にonTimerメソッドが実行される → isrCounterがカウントアップ → メインループ（loopメソッド）でisrCounterの値はモニタされていて，isrCounterが0以上ならメイン処理を開始。その際にisrCounterをカウントダウンする。

つまりonTimerではisrCounterを増やし続け，メインループ内ではisrCounterを減らし続ける。この２つのバランスが取れていればisrCounterは0→1→0→1 という変化を繰り返し続ける。

また，割込みとメインループで共用する変数がお互い意図しないタイミングで変更されるのを防ぐ為として， portENTER\_CRITICAL\_ISRやtimerSemaphoreなどを利用している。

volatile int isrCounter=0; // 処理の有効無効を判断する為の数値

portMUX\_TYPE timerMutex = portMUX\_INITIALIZER\_UNLOCKED;

hw\_timer\_t \* timer = NULL;

volatile SemaphoreHandle\_t timerSemaphore; // semaphoreを使わない例もみられる。必須ではないらしい。

float readAmpere(int ch){} // メソッドの内容はSTEP2に同じなので省略

void IRAM\_ATTR onTimer(){

portENTER\_CRITICAL\_ISR(&timerMutex); // 割り込み禁止区間　開始

isrCounter++; // isrCounterをカウントアップ＝ 処理を1回追加

portEXIT\_CRITICAL\_ISR(&timerMutex); // 割り込み禁止区間　終了

xSemaphoreGiveFromISR(timerSemaphore, NULL); // semaphoreは必須ではない。

}

void setup() {

timer = timerBegin(0, 80, true); //

timerAttachInterrupt(timer, &onTimer, true);　// (タイマーハンドル，割込みメソッド名，エッジ割込み)

timerAlarmWrite(timer, 1000, true); // (タイマーハンドル，タイマーカウント数，継続タイマ)

timerAlarmEnable(timer); // タイマーアラーム（割込み）を有効化

timerSemaphore = xSemaphoreCreateBinary();

}

void loop() {

if(xSemaphoreTake(timerSemaphore,0)==pdTRUE){// 上述のとおり，semaphoreまでは使わなくても動く

if(isrCounter > 0){ // メイン処理の必要有無判断

　　portENTER\_CRITICAL(&timerMutex); // 割り込み禁止区間　開始

　　isrCounter--; // isrCounterをカウントダウン＝ 処理が１回終わった

　　portEXIT\_CRITICAL(&timerMutex); // 割り込み禁止区間　終了

　　// ここでメイン処理を行う。ここならばI2CやSPI通信を行っても良い。

　　float ampere = readAmpere(0);

　　}

　}

}

（参考資料）<https://qiita.com/pontaman007/items/0540409fada8c3762ca7>

##### 1秒間Arms計算

int timerCounter=0;

int secondCounter=0;

double ampereSquareSum=0;

float armsValue=0;

void loop() {

　if(isrCounter > 0){

portENTER\_CRITICAL(&timerMutex);

isrCounter--;

portEXIT\_CRITICAL(&timerMutex);

timerCounter++; // タイマーカウンタ をカウントアップ

float ampere = readAmpere(0);

ampereSquareSum+= pow(ampereVal,2); // 電流値の２乗を加算していく

　}

　if(timerCounter >= 1000){ // 1秒経過：実際のタイマー分周によりカウンタ数は異なる

　　armsValue= sqrt((ampereSquareSum/timerCounter)); // 平均の平方根を算出=1秒間のArms

　　secondCounter+=1; // 秒数を保持する変数をカウントアップ

　　timerCounter=0; // 必要変数の初期化

　　ampereSquareSum=0;

　}

}

##### ADS1015の場合の補足

・I2C通信はそれほど高速で無いためか，タイマー周期が早めだとisrCounterの数がどんどん増えていってしまう。つまりI2C通信を終える前に次のタイマーの割込みが入ってしまい，処理がどんどん溜まっている状態。

対策1　readAmpereADS1015で行っていたconfigの通信をsetupメソッドに持っていく

void setup() {

//.. 略

byte mux=0b100;

byte cfg\_lsb = dr <<5 | 0x03;

byte cfg\_msb = os << 7 | mux << 4 | pga << 1 | mode\_;

Wire.beginTransmission(0x48);

//.. 略

→　200mSecf経過時のisrCounterの値が4108から1492に改善。

但し，読込みchはch0に固定されてしまう。

対策２　通信周波数を400kHzに

Wire.begin();

Wire.setClock(400000); // 100000(100k) → 400000(400k)に変更

→　200mSec時のisrCounter数（4回計測の平均）　100kの時：2040→　400kの時：1404

そこそこに効果有り。

対策３　ハイスピードモード

Wire.beginTransmission(0x48);

Wire.write(0x08); // hs-mode を有効化

→　200mSec時のisrCounter数（4回計測の平均）　1404→877

効果はあるようだが，そもそも送信バイト数が少ないのでそれほどの恩恵を受けられない。

対策４　タイマー周期を1mSec→4mSecに。

timerAlarmWrite(timer, 4000, true); // dividerは80なので，これで5mSec周期に

isrCounterの値が0になり，成功。逆に言えばここまでしないとisrCounterの値は0にまで落ちなかった。実際現状でもギリギリとおもわれる。

なお，Serial.printの回数を減らすと結構改善する。ただ，当然見たいデータは省けないので，デバッグによるモニタも最小限にすると良い。

完全版ソース

#### STEP4. 電力量計算

##### 概要

##### ソースコード

bool ctSensorEnable=false;

double ampereSecSum=0;

void IRAM\_ATTR onTimer(){ // 指定時間経過毎に実行されるメソッド

}

void loop() {

// wifi処理をloopメソッド内で行う。

}

（情報元）<https://lang-ship.com/blog/work/esp32-timer/>

（公式）<https://www.arduino.cc/reference/en/libraries/esp32timerinterrupt/>

（補足）hw\_timer \* の「\*」は掛け算では無くて，ポインタを示す。

##### 詳細情報

timerAlarmWrite

構文：hw\_timer\_t\* timerBegin(uint8\_t num, uint16\_t divider, bool countUp);

num 利用するタイマー番号。0～3。

divider 分周比(divider)。CPUクロック周波数何回で「1分周」とするか（クロック周波数は80MHｚ。）

countUpタイマをカウントアップさせる場合true　カウントダウンはfalse。

※CPUクロックは「CLKソース」によって異なる点に注意

|  |  |  |  |
| --- | --- | --- | --- |
| CLKソース | CPUクロック | ペリフェラル周波数 | 備考 |
| PLL\_CLK[320MHz] | 80~240 MHz | 80 MHz |  |
| XTL\_CLK[2~40MHz] | 2~40 MHz | 2~40 MHz |  |

timerAlarmWrite

構文：void timerAlarmWrite(hw\_timer\_t \*timer, uint64\_t alarm\_value, bool autoreload);

timer hw\_timerオブジェクト（タイマーハンドラ）へのポインタ

alarm\_value 割込み関数を呼び出すまでの分周数。　 実際の周期時間[sec] =｛分周比÷CPUクロック｝× 分周数

autoreload 自動再起動有無。=trueで再登録されて周期的に実行。

#### STEP5. ソケット通信

##### Wifi通信基本

#include <**WiFi**.h>

WebServer server(80); // コンストラクタの引数がポート番号になる

void setup(){

**Serial**.begin(115200);

**Serial**.print("Connecting");

**WiFi**.begin("your-ssid", "your-passwd"); // unicodeのダブルクオーテーション “ を使わないように！

while (**WiFi**.status() != WL\_CONNECTED) {

delay(500);

**Serial**.print(".");

}

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

server.begin();

}

void loop() {

{

**WifiClient** client = server.available(); // 接続したクライアントの情報を得る。

　if(client){

　　IPAddress ip = client.remoteIP(); // クライアント側のIPアドレス

　　Serial.print("new client:");

　　Serial.println(ip);

　　while(client.connected()){

　　　char msg = client.read(); // こちらの場合は　msg==’test’ となる

　　　//String msg = readStringUntil(‘\n’); // こちらの場合は　msg==”test” となる

　　　Serial.print("received msg:"); // 受信電文の表示

　　　Serial.println(msg);

　　　if (msg == 'p') { // 受信電文の表

　　　　client.println("test");

　　　　break;

　　　}

　　　else{

　　　　client.println("wrong request!!");

　　　　break;

　　　}

　　}

　　Serial.println("disconnecting.");

　　client.stop(); // 切断する

　}

}

※何やら割込みでやる方法もあるらしい

（参考）<https://kesokonpatata.hatenablog.jp/entry/2021/05/17/ESP32でブレーカーの電流値を測定>

xTaskCreateUniversal　でタスクが作れる？

https://lang-ship.com/blog/work/esp32-freertos-l02-taskcreate/

##### クライアント側（python）

上記ソースのESP32とソケット通信をするクライアント側のpythonコード例

import socket

host = '192.168.1.1' # ESP32のアドレス

port = 80 # ESP32のポート番号

soc = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

soc.connect((host,port))

soc.sendall(bytearray("p".encode())) # 文字列の送信

#soc.send(b'p') # この方法でも良い

rcv\_data = soc.recv(1024)

print('Received', repr(rcv\_data))

soc.close()

#### 資料（ESP32）

##### 内部ADコンバーター

###### 概要

12bitのADコンバーターが2つ内蔵。内蔵ADコンバーターの片方は8ch，もう片方は10ch。但しADC1\_CH1[pin6 GPIO37] とADC1\_CH2[pin7 GPIO38]　はマイコンでは使えなくなっている。

更にADC2はwifiを使う時は使用できない。

加えて下記によりADC1\_CH4[pin8 GPIO32]とADC1\_CH5[pin9 GPIO33] も積極的には使いにくい。

pin8,9は通常32kHzの水晶（クロック）と内側でつながっているので，水晶とそのコンデンサC13とC17を外し，R5とR6の抵抗をはんだ付けする必要がある　未確認情報2021-08-04

###### 対応ピン

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| pin | 名称 | A番号 | 備考 | pin | 名称 | A番号 | 備考 |
| 4 | ADC1\_CH0 | A0 | IO36　SENSOR\_VP  INのみ　ADC\_H | 26 | ADC2\_CH0 | A10 | IO4 |
| 25 | ADC2\_CH1 | A11 | IO0　 PGM |
| 5 | ADC1\_CH3 | A3 | IO39　SENSOR\_VN  INのみ　ADC\_H | 24 | ADC2\_CH2 | A12 | IO2 |
| 23 | ADC2\_CH3 | A13 | IO15 |
| 8 | ADC1\_CH4 | (A4) | IO32 | 16 | ADC2\_CH4 | A14 | IO13　？ |
| 9 | ADC1\_CH5 | (A5) | IO33 | 14 | ADC2\_CH5 | A15 | IO12 |
| 6 | ADC1\_CH6 | A6 | IO34　 INのみ | 13 | ADC2\_CH6 | A16 | IO14 |
| 7 | ADC1\_CH7 | A7 | IO35　INのみ | 12 | ADC2\_CH7 | A17 | IO27 |
|  |  |  |  | 10 | ADC2\_CH8 | A18 | IO25　？ |
|  |  |  |  | 11 | ADC2\_CH9 | A19 | IO26 |

###### 測定レンジ

atten（Attenuation）と言う値によって測定レンジが変わってくるらしい。

対応表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 値 | 列挙型(enum) | 減衰率 | full-scale | 測定レンジ | 誤差 | 備考 |
| atten=0 | ADC\_ATTEN\_DB\_0 | 0dB | 1.1V | 100 ∼ 950 mV | ±23mV |  |
| atten=1 | ADC\_ATTEN\_DB\_2\_5 | 2.5dB | 1.5V | 100 ∼ 1250 mV | ±30mV |  |
| atten=2 | ADC\_ATTEN\_DB\_6 | 6dB | 2.2V | 150 ∼ 1750 mV | ±40mV |  |
| atten=3 | ADC\_ATTEN\_DB\_11 | 11dB | 3.9V | 150 ∼ 2450 mV | ±60mV | デフォルト設定 |

厳密にはここで示した測定レンジ外の電圧も測定できるが，精度が落ちる，との事。

⇒ 測定レンジのフルスケールに対して2.5%程度も誤差がある。

###### コーディング

Arduino Studio

int a0\_value = analogRead(4); // pin#4(=GPIO36) の値を読む

（公式情報）<https://cdn.arduino.cc/reference/jp/language/functions/analog-io/analogread/>

C/C++

#include <esp\_adc\_cal.h>

int a0\_value = adc1\_get\_raw(ADC1\_CHANNEL\_0); // pin#4(=GPIO36) の値を読む

（公式情報）<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html#_CPPv412adc1_get_raw14adc1_channel_t>

補足：

VDD3P3\_RTC CPUピン#19：RTC IO用の電圧入力。マイコンボードではNCになっている

ADCピンはノイズに弱いので，0.1μFのコンデンサをつけろ，との事。

GPIO34~39には内部のpull-up，pull-down 抵抗が無い。

具体的な変更方法

void setup() {

adc1\_config\_channel\_atten(ADC1\_CHANNEL\_0, ADC\_ATTEN\_DB\_6) // ADC1\_CH0のattenを2にする

}

（補足）adc1\_get\_raw()メソッドを使う前にこのメソッドを実行する必要がある

※ArduinoStudioの話ではない気がする2021-07-15

補足情報

解像度（bit数）は9~12bitのいずれかを選択できるらしい。高速処理が必要な場合にはbit数を下げる。

void setup() {

adc1\_config\_width(ADC\_WIDTH\_BIT\_10) // 10bitに変更

}

※ArduinoStudioの話ではない気がする2021-07-15

関連する公式の関数レファレンス

関数(method) adc1\_config\_width()　<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html#_CPPv417adc1_config_width16adc_bits_width_t>

列挙型(enum) adc\_bits\_width\_t　<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html#_CPPv416adc_bits_width_t>

esp\_adc\_cal\_check\_efuse

calibration methods for ADC1

##### I2C

本家の情報ではこうなっていたが？？　2021-08-04

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | GPIO | pin#(CPU) | ADC(参考) | 備考 |
| I2C\_SDA | IO15 | 21 | ADC2\_CH3 |  |
| I2C\_SCL | IO2 | 22 | ADC2\_CH2 |  |
| I2C\_SDA | IO0 | 23 | ADC2\_CH1 |  |
| I2C\_SCL | IO4 | 24 | ADC2\_CH0 |  |

これは何？

|  |  |  |  |
| --- | --- | --- | --- |
| I2CEXT0\_SCL\_in | I2CEXT0\_SDA\_in | I2CEXT0\_SCL\_out | I2CEXT0\_SDA\_out |
| I2CEXT1\_SCL\_in | I2CEXT1\_SDA\_in | I2CEXT1\_SCL\_out | I2CEXT1\_SDA\_out |

補足

Wire.readに対し，Stream.read() というメソッドもあるらしい。

Arduino StudioでもADS1015.hをincludeする方法がある？

##### 用語メモ

RTC：Real Time Clock

DIG：Data Integrity Group

DNL：Differential Nonlinearity 微分非直線性誤差　1LSB以下が目安。

INL：Integral Nonlinearity 積分非直線性誤差　1LSB以下が目安。

LSB：Least Significant Bit 最下位ビット　参考電圧 ÷ (2×分解能)

SAR：Successive Approximation 随時比較型。ADコンバーターの測定方法の一種の事。

VDDA：アナログへの電源供給（ピン）の事

DMA：なぜか知らんがADC continuous mode の事らしい。

ULP：省電力モードの時に使用するプロセッサの事

ISR：Interrupt Service Routine 割込み処理を行う関数などの事。

（参考）<https://labo.mycabin.net/electronics/arduino-esp32/735/>

（公式；英語）<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html>

##### Espressif Systems公式資料

資料の説明

ESP-WROOM-32 Datasheet [Webリンク](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf) マイコンボードの説明。

ESP Datasheet CPUの説明。

##### その他資料

###### タイマーのサンプルプログラムの解説

ファイル→スケッチ例→ESP32→Timer→RepeatTimer にサンプルプログラムがある

ただ，あまり分かりやすく書いてはいない。Webに解説資料を見つけた。

（参考）スケッチ例の解説　<https://lang-ship.com/blog/work/esp32-timer/#toc8>

# ストレージ系

ImageStore

ライセンス：Apache-2.0

digiKam

### 資料

#### exifデータ

EXIF（Exchangeable image file format）

|  |  |
| --- | --- |
| ・撮影日時 | 日付と秒までの時刻が付加される。 |
| ・位置情報（ジオタグ） | GPS　緯度・経度・標高などが記録 |

など

from PIL.ExifTags import TAGS, GPSTAGS

関連ライブラリ：

smopyはOpenStreetMapのイメージタイルを取得する

参考：<https://www.vieas.com/en/exif23.html>

# アイディア

## カメラによる水位センサ

これの丸パクリ

<https://www.sanshin.co.jp/solution/iot/camera-ai/?lfpeid=h9hHRtvBr3Jn&lfmaid=1000247055-6>

# memo

## OPC UA　【補足】

マルチベンダー製品間や、異なるOSにまたがってデータ交換を可能にする高信頼の産業通信用の通信規格。

IEC62541。COM/DCOM技術がベース？SOAPベース、HTTPSベース、TCPベースのプロトコルが存在。

Native Binaryの場合、TCPでハンドシェイク→通信　　結局中継サーバーが必要らしい

python

pip install opcua

CODEYES はOPC UAサーバーにもなる。

https://misoji-engineer.com/archives/opc-ua.html

## profinet

世界で最も普及しているイーサネット（Ethernet）ベースの産業用ネットワーク

キーエンスが採用？

## micro-ROS

ROS（Robot OS）と通信するためのマイコンOS。

補足：RTOSと混同しないように

ROS2

## AWS IoT Greengrass 1.9.3

AWSの機能をローカルデバイスに拡張するソフトウェア

ラズパイでも利用可能らしい

## Google Coral Dev　【スピンオフ】

Arm NXP i.MX 8M SOC(4コア)　サポートOS：Debian

TPU(TensorFlow Processing Unit)「TensorFlow」に最適化させたカスタムASIC(application specific integrated circuit、特定用途向け集積回路）

公式：https://coral.ai/products/

## arduino industrial 101

Arduino YÚNという製品に近いらしい（現在は販売終了）

Linux（Linino）が動作し、SSH接続も可能。pythonも動作する。

Arduino 101 LGA moduleの評価基盤

イーサネットコネクタ

<https://akizukidenshi.com/catalog/g/gM-10445/>

⇨Lininoはあまり人気が無かったらしい

ディスコンも考慮する（マイコンにはディスコンの問題が付き物）IoTゲートウェイも同様。

## RevPi

https://www.takagishokai.co.jp/product-search/2019/12/12/297

BYOD；Bring Your Own Device

従業員が個人で所有する情報機器を業務のために使用すること

TCO；Total Cost of Ownership

システム導入から運用及び維持・管理までを含めた費用の総額

エンタープライズアーキテクチャは一般に，次の 4 つのカテゴリに分類されます。

① ビジネスアーキテクチャ（BA）

IT 機器やシステムの省電力化や熱対策な業務機能の構成。機能構成図（DMM），機能情報関連図（DFD）などが作成。

② データアーキテクチャ（DA）

業務機能に使われる情報の構成。実体関連ダイアグラム（ERD），データ定義表などが作成。

③ アプリケーションアーキテクチャ（AA）

業務機能と情報の流れをまとめたサービスの固まりの構成。情報システム関連図，情報システム機能構成図などが作成。

④ テクノロジアーキテクチャ（TA）

各サービスを実現するためのテクノロジの構成。ネットワーク構成図，ソフトウェア構成図などが作成。

## NASによるバックアップシステム

2022年6月23日　定例勉強会の五十川さんの発言より。

半日～一日分のcsvファイル？をNASに保存しておく事でシステムやネットワーク障害向きのバックアップシステムとする。かなりの良案に思えた為メモ。

→通常はRESTか何かでjsonを返すシステムにしておき、平行してNASにも同じjsonファイルを保存する仕組みは障害に強いし復旧も楽なように思える。

# 用語

## MES；Manufacturing Execution System

エムイーエス、メス