

6コマ目  
Pythonによる  
ロボット制御プログラミング

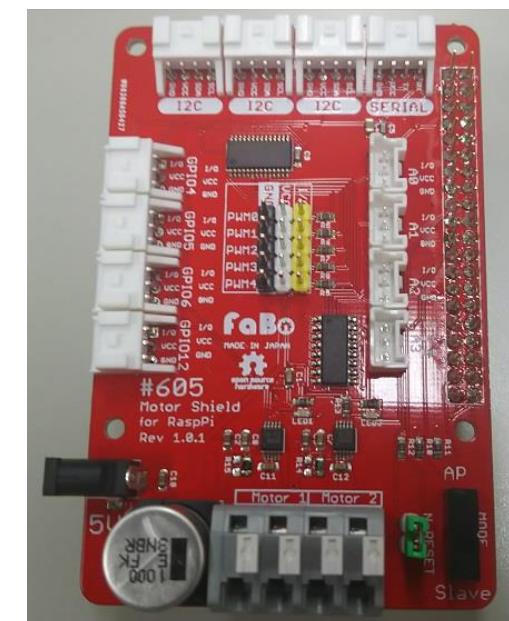


# FaBoについて



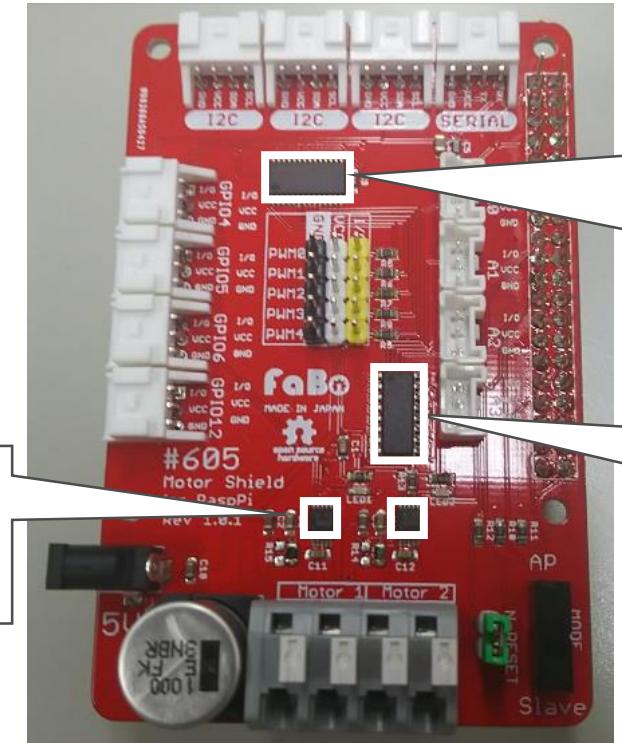
# FaBo概要

- モータシールド（拡張シールド）
  - Raspberry Piに接続することによって、サーボモータ，DCモータ，センサを簡単に接続し，操作できる
  - ハードウェアに簡単に接続できるので，電子工作が容易になる
  - 幅広いコンピュータに対応できる
    - Arduino
    - Raspberry Pi
    - IchigoJam



# FaBoに組み込まれているドライバ

- 本講習で使用するのはMCP3008とDRV8830



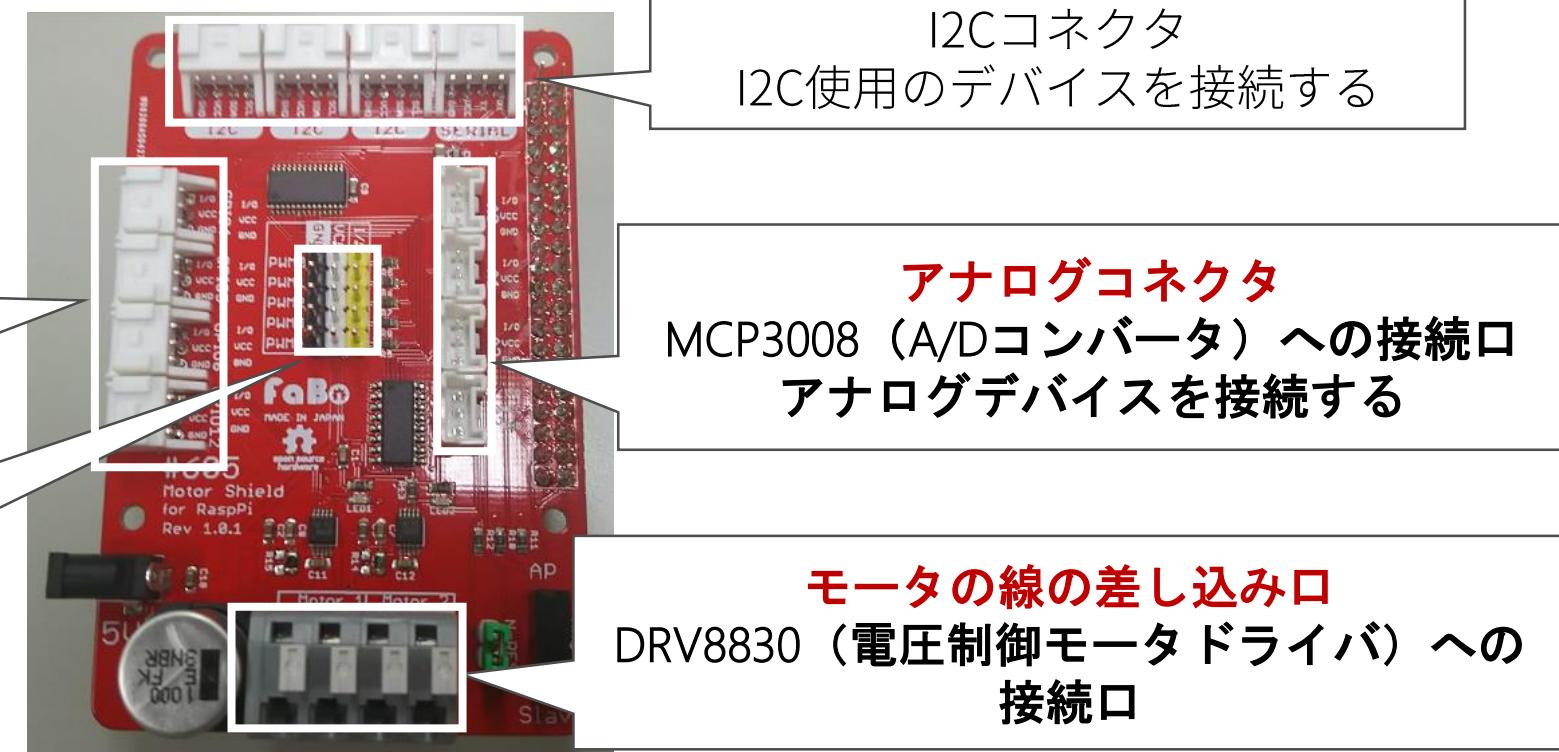
DRV8830 : 電圧制御モータドライバ  
モータにかける電圧を制御

PCA9865 :  
サーボモータを制御するPWMコントローラ  
PWMとはオンとオフの繰り返し  
スイッチングを行い、  
出力される電力を制御する方法

MCP3008 : A/Dコンバータ  
アナログ値をデジタル値に変換

# デバイス差し込み口

- 本講習で使用するのは**GPIOコネクタ**と**アナログコネクタ**と**モータの線の差し込み口**



# FaBo回路図

- 今回使用する箇所は赤枠で囲まれた所

Raspberry Pi

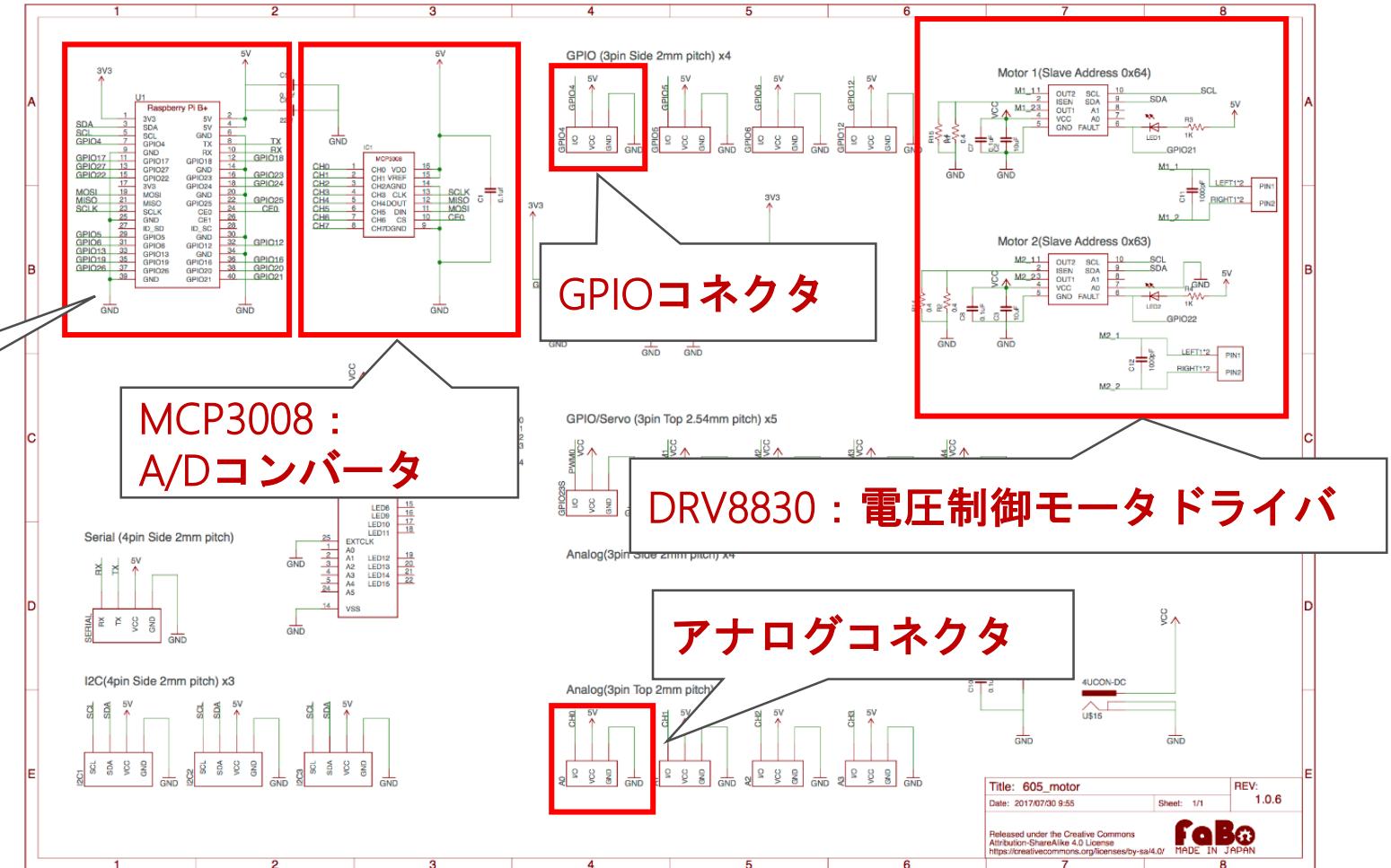
MCP3008 :  
A/Dコンバータ

GPIOコネクタ

DRV8830 : 電圧制御モータドライバ

アナログコネクタ

FaBo回路図：  
<http://www.fabo.io/605.html>



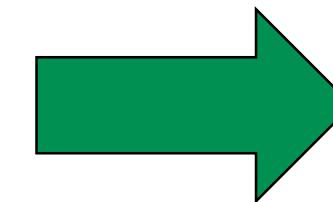
# FaBoとRaspberry Piの接続方法



FaBo



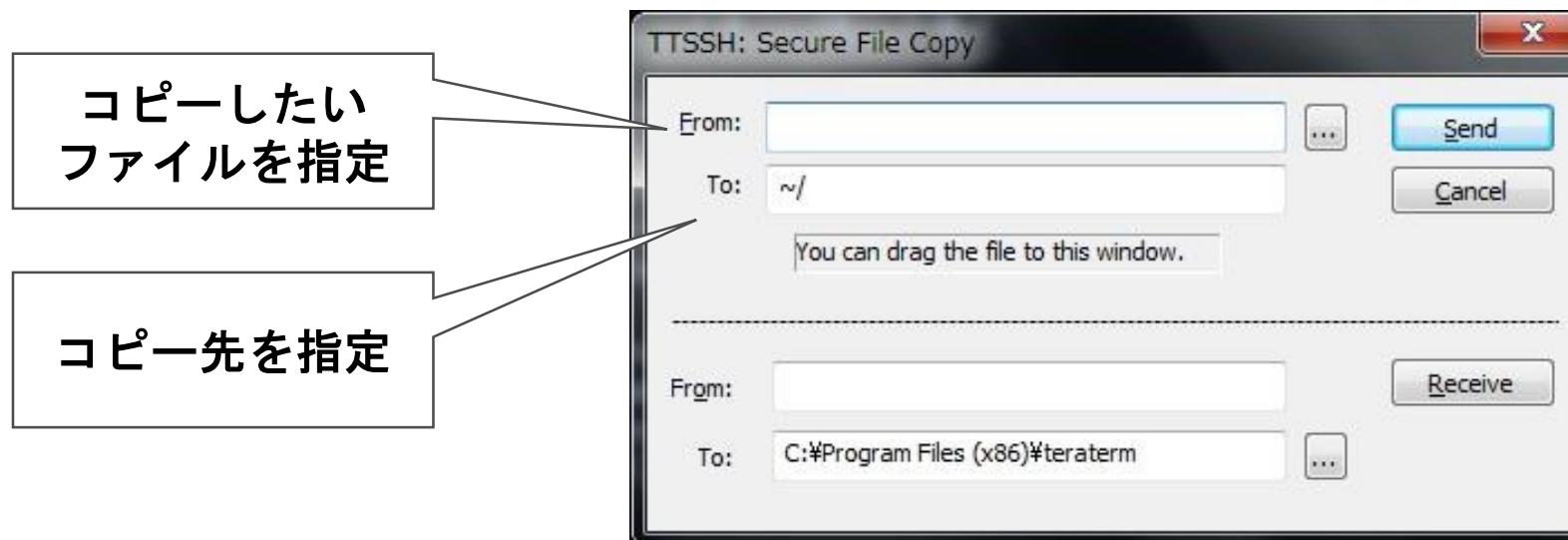
Raspberry Pi



Raspberry Pi

# サンプルプログラム

- 以下のURLからzipファイルをダウンロード  
<https://rtc-fukushima.jp/wp/wp-content/uploads/2018/11/06Sample.zip>
- SCPでファイル転送
  - TeraTermの「ファイル」→「SSH SCP …」を使用して、ダウンロードファイルをRaspberry Piへコピーする



# サンプルプログラム

- コピーを行ったら以下コマンドでzipファイルを解凍し,  
ディレクトリの中身を確認すること

```
$ sudo apt-get -y --force-yes install zip unzip  
$ unzip 06Sample.zip  
$ cd 06Sample  
$ ls
```

- 以下ファイルがあることを確認すること
  - DCMotor.py
  - Distance.py
  - LED.py

# LEDモジュールの操作

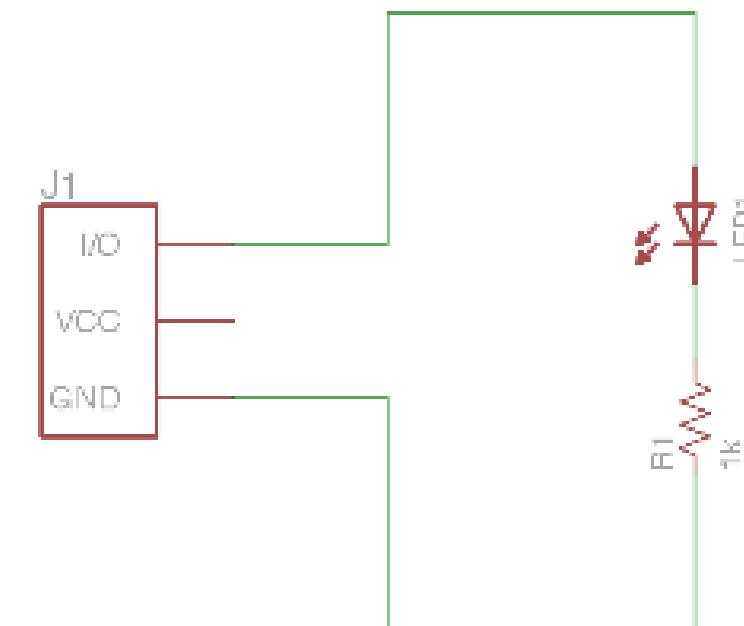


# 目的

- LEDモジュールをFaBoに接続して、  
Raspberry Pi上のプログラムで2秒点灯させる

# LEDモジュールの概要

- 電気が流れるとあらかじめ決められた色のLEDを点灯
  - 発光色は5色（青・緑・赤・白・黄）
  - 1KΩの抵抗が付いている

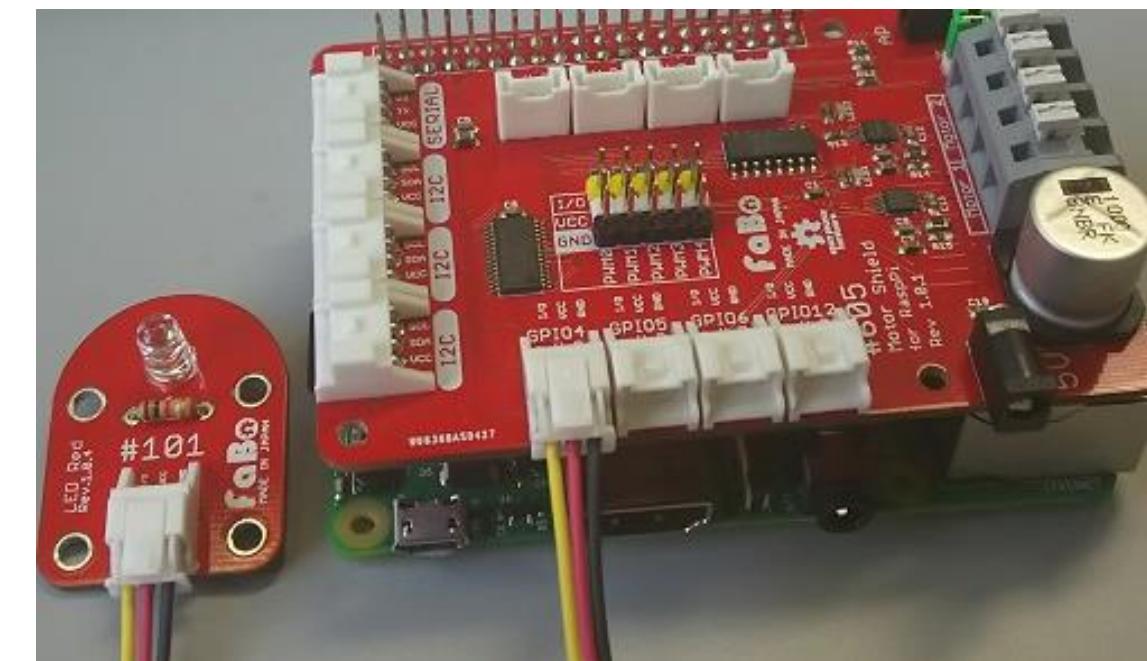


LED回路：[http://www.fabo.io/101\\_A.html](http://www.fabo.io/101_A.html)

# LEDモジュールとFaboとの接続方法

- LEDモジュールをケーブルで接続
    - GPIOコネクタに接続（※ここではGPIO4に接続）

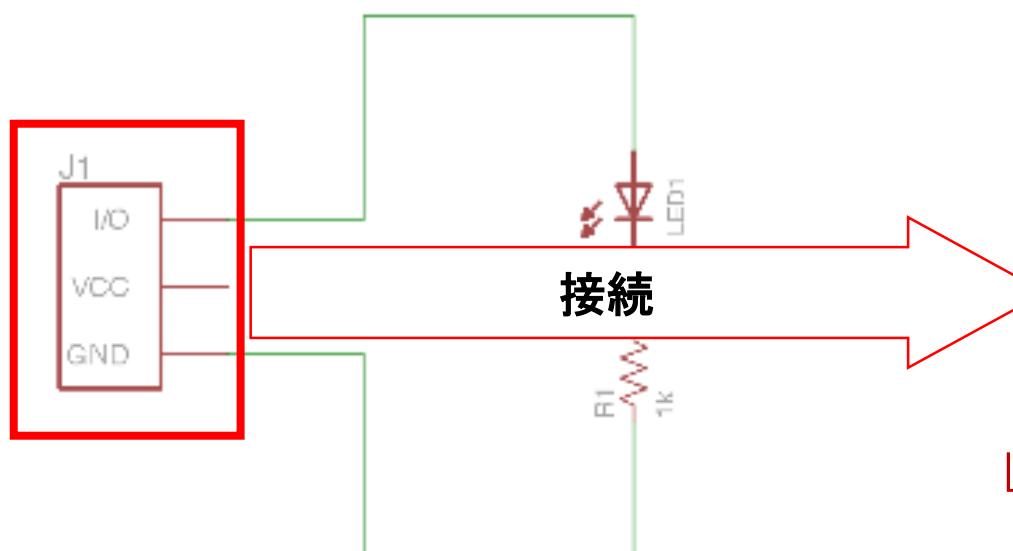
I/O	VCC	GND
黄色	赤	黒



# 回路から見た接続

- LED回路

[http://www.fabo.io/101\\_A.html](http://www.fabo.io/101_A.html)

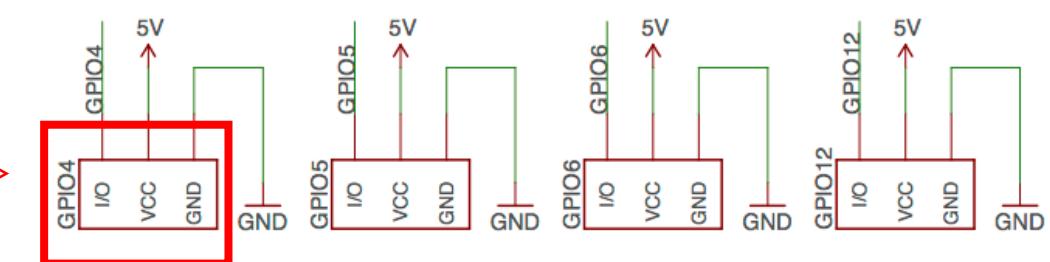


- FaBo回路 (1部抜粋)

<http://www.fabo.io/605.html>



GPIO (3pin Side 2mm pitch) x4



LEDがRaspberry PiのGPIO4に接続した状態なる

GPIO4を通して電流が送られ、LEDを点灯する

# LEDモジュール操作サンプルプログラム

- GPIO4に接続したLEDを2秒点灯させるプログラムを作成
  1. GPIOモジュールの導入  
Raspberry PiのGPIOを使用するためのライブラリの導入
  2. 初期化処理  
どのGPIOをどのモードで使用するかの設定
  3. 点灯, 消灯処理  
電気を2秒流して点灯後, 電気を流すのをやめて, LEDを消灯させる
- ダウンロードしたファイルLED.pyを参考にする

# モジュールの導入と初期化処理

- import文でGPIOの導入

```
import RPi.GPIO as GPIO # GPIOを使用するためのモジュール  
import time # sleepを使用するためのモジュール
```

- 初期化処理

```
LEDPIN = 4 # 使用するFaBo接続時のGPIO番号  
GPIO.setmode(GPIO.BCM) # GPIOピンの設定の仕方  
GPIO.setup(LEDPIN, GPIO.OUT) # GPIO4を出力として設定
```

# RPi.GPIO

- GPIOを使用するためのモジュール
- Raspbianに標準でインストールされている Python ライブライ

# GPIOの番号について

- GPIOの各ピンは図のようにそれぞれ役割を持つ
  - **GPIO.BRD** : 左上から1~40までの番号で使用
  - **GPIO.BCM** : 役割に記載されたGPIOの数字で指定

<https://elinux.org/File:Pi-GPIO-header.png>

	Pin 1	Pin 2	
+3V3			+5V
GPIO2 / SDA1			+5V
GPIO3 / SCL1			GND
GPIO4			TXD0 / GPIO 14
GND			RXD0 / GPIO 15
GPIO17			GPIO 18
GPIO27			GND
GPIO22			GPIO 23
+3V3			GPIO 24
GPIO10 / MOSI			GND
GPIO9 / MISO			GPIO 25
GPIO11 / SCLK			CE0# / GPIO8
GND			CE1# / GPIO7
GPIO0 / ID_SD			ID_SD / GPIO1
GPIO5			GND
GPIO6			GPIO12
GPIO13			GND
GPIO19 / MISO			CE2# / GPIO16
GPIO26			MOSI / GPIO20
GND			SCLK / GPIO21

# LEDモジュールの操作と終了処理

- LED点灯，消灯

```
GPIO.output(LEDPIN, True) # LEDをONにする
```

```
GPIO.output(LEDPIN, False) # LEDをOFFにする
```

- GPIO使用終了

```
GPIO.cleanup() # GPIOの使用を終了
```

- 命令の停止（停止することでLEDのON/OFFを維持する）

```
time.sleep(2.0) # 2秒間停止する(timeをimportする)
```

# LED.pyを実行

- 以下コマンドでLED.pyを実行して、LEDが点灯することを確認してください

```
$ cd 06Sample
```

```
$ python LED.py
```

# モータの操作



# 目的

- FaBoに接続されたモータを動かし、ロボットカーを制御する

# ギヤボックス概要

- ダブルギヤボックス（左右独立4速タイプ）

ギヤ比	回転数	回転トルク
12.7 : 1	1039rpm	94gf-cm
38.2 : 1	345rpm	278gf-cm
114.7 : 1	115rpm	<b>809gf-cm</b>
344.2 : 1	38rpm	2276gf-cm



# ギヤボックス概要

- 使用モータ：FA-130

限界電圧	1.5~3.0V
適正電圧	1.5V
適正負荷	0.39mN・m (4.0g・cm)
回転数	6,400r/min
消費電流	500mA
シャフト径	2.0mm
重量	18g
外観寸法	25.0×20.1mm

# モータとギヤボックス

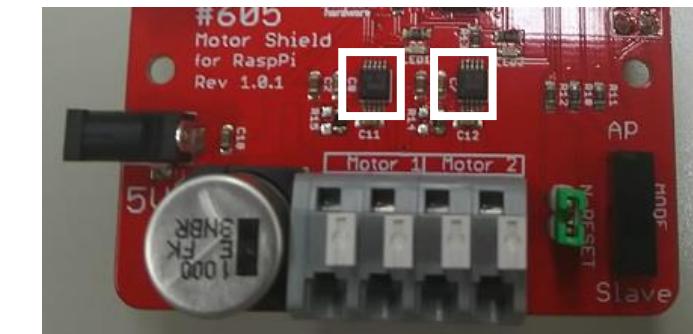
- 回転数と電圧の関係
  - モータの回転数はかけられた電圧と比例するところでは仮定する
  - 1.5V : 6400r/min の場合, 電圧を2倍にすると, 回転数も2倍になる
  - 3.0V : 12800r/min となる
- 回転数とギヤ比
  - モータの回転数が6400r/min, ギヤ比が344.2:1の場合,  
 $6400\text{r}/\text{min} \div 344.2 = 18.5$  となるため,  
回転数は18.5r/minとなる

# ギヤボックスとFaBoとの接続方法

- 初日に接続方法を確認したため、ここでは説明を割愛

# DRV8830

- モータドライバ
  - モーターを駆動・制御する装置
  - モータにかける電圧をプログラムから制御することを可能にする
  - I2C方式で制御を行う
    - 周辺デバイスとのシリアル通信の方式
- FaBoには2つ搭載されている
- Motor1, Motor2に接続したモータをそれぞれ制御する



# データシート内容確認

- プログラムでモータを制御するために以下のものを確認する
- データシート  
[http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830\\_manual.pdf](http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830_manual.pdf)
  - 秋月電子通商に掲載されているものを確認
- 回路図  
<http://www.fabo.io/605.html>
  - FaBoを販売しているGClue記載の回路図

# DRV8830 | 特徴

## ■特徴■

- ★DCモータモータドライブICのDRV8830を使用しています。
- ★マイコンのI2CインターフェイスでDCモータの正転・逆転・ブレーキ・惰走の切り替、速度コントロールが出来ます。当社FA-130RA-2270モーターに適しています。
- ★I2Cアドレスの設定で、最大9キットまで制御が出来ます。
- ★幅広い動作電源電圧 2.75V～6.8V(マイコンの電源電圧と同じにする必要があります。)
- ★センサー抵抗による電流制限回路(1A)内蔵です。
- ★1過電流保護・短絡保護・低電圧誤動作防止・加熱保護機能内蔵です。
- ★内部レジスタを参照する事で、障害内容を知ることが出来ます。またモニター用 FAULT LED付きです。
- ★電池駆動などで電源電圧が変化しても、モータ速度を一定に保ちます。

データシート：[http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830\\_manual.pdf](http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830_manual.pdf)

- 通信方式はI2Cだと判明する

# I2C | 確認項目

- 通信方式がI2Cと判別
- I2Cで通信するためには、以下の項目を確認する必要がある
  - 端子又は回路
  - スレイブアドレス
  - レジスタ
  - 入出力の値

# DRV8830 | 端子

CN1モータ接続端子

CN1	OUT1	正転時に「+」
	OUT2	正転時に「-」

モータへの電圧供給

CN2モータ接続端子

CN2	VCC	電源入力
	GND	GND

ドライバへの電圧供給

CN3マイコン接続端子

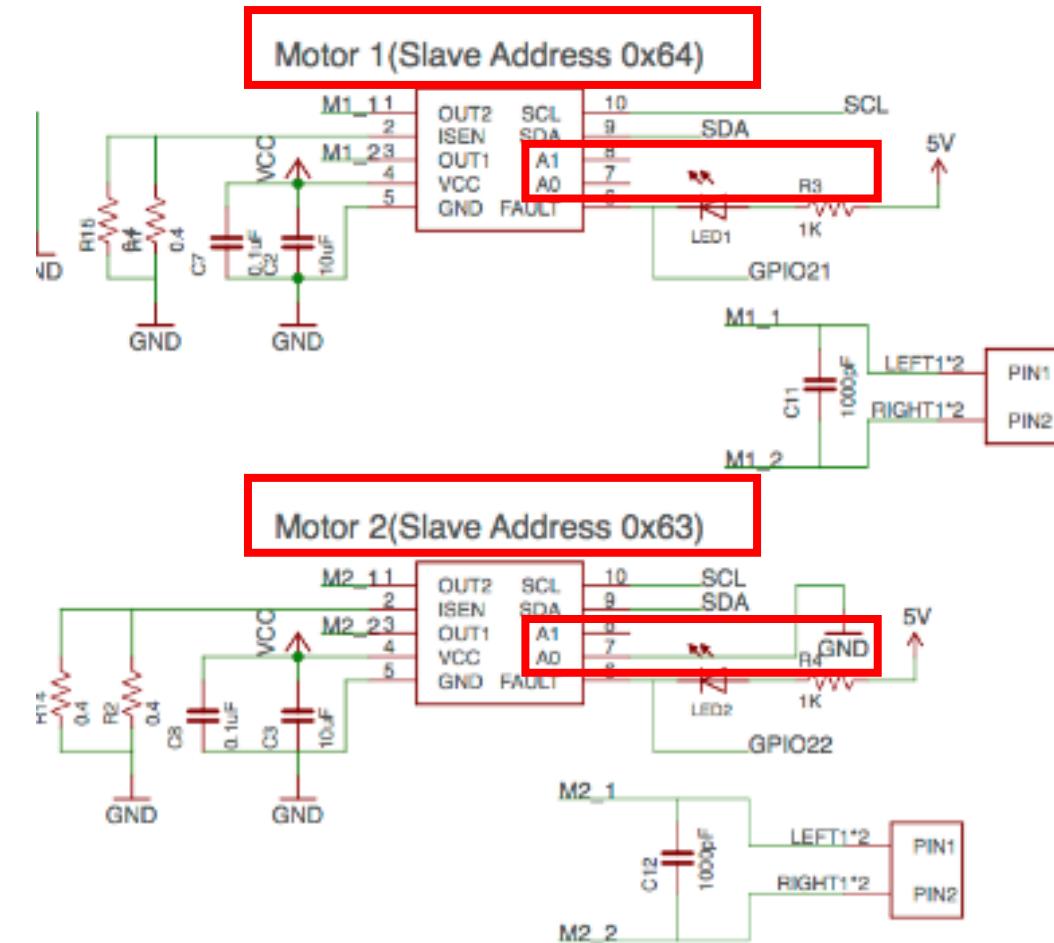
CN3	SCL	クロックライン
	SDA	データライン
	A1	アドレス設定1
	A2	アドレス設定2
	FAULTh	障害状態の出力
	GND	GND

アドレス設定の  
端子がある

ドライバとマイコンの  
データのやり取り

# DRV8830 | スレイブアドレス

- FaBoの回路図から
- Motor1
  - A0 : オープン
  - A1 : オープン
  - スレイブアドレス : **0x64**
- Motor2
  - A0 : GND(LOW)
  - A1 : オープン
  - スレイブアドレス : **0x63**



回路 : <http://www.faboi.io/605.html>  
会津大学ロボット教材 ©2017-2019 The University of Aizu

# DRV8830 | スレイブアドレス

- [A0]端子をVCC  
 [A1]端子をGNDに接続した場合,  
 • 回路の設定の仕方でアドレスが変わる

Motor	A1	A0	書き込みアドレス	読み取りアドレス
Motor1	オープン	オープン	0xc8	0xc9
Motor2	オープン	GND(Low)	0xc6	0xc7

H L	H L	H L	
1	0	オープン	
A1	A0	書き込みアドレス	読み取りアドレス
0(L)	0(L)	0xC0h	0xC1h
0(L)	オープン	0xC2h	0xC3h
0(L)	1(H)	0xC4h	0xC5h
オープン	0(L)	0xC6h	0xC7h
オープン	オープン	0xC8h	0xC9h
オープン	1(H)	0xCAh	0xCBh
1(H)	0(L)	0xCCh	0xCDh
1(H)	オープン	0xCEh	0xCFh
1(H)	1(H)	0xD0h	0xD1h

データシート：[http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830\\_manual.pdf](http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830_manual.pdf)

# DRV8830 | スレイブアドレス

- データシート記載のアドレスは、書き込み／読み出しの判定ビットを含んでいる
  - したがって、実際のスレイブアドレスは下位ビットを含めない値となる
- Motor1
  - 16進数 :  $0xC6 \rightarrow 0x63$
  - 2進数 :  $11000110 \rightarrow 1100011$
- Motor2
  - 16進数 :  $0xC8 \rightarrow 0x64$
  - 2進数 :  $11001000 \rightarrow 1100100$

# DRV8830 | レジスタ

- 値を格納するためのアドレスを確認

サブアドレス	レジスタ名	デフォルト値	説明
0x00	CONTROL	0x00	出力の状態 および出力電圧の設定
0x01	FAULT	0x00	障害状態の読み取り およびクリア

- モータに電圧を加えて、モータを動かしたい場合、  
CONTROL 0x00を使用

# DRV8830 | 入出力の値

- モータドライバーへの値は8bitで入力される
- 上位6bitが出力，下位2bitが出力の状態

VSET[5..0]	IN2	IN1
出力	出力の状態（ブリッジ制御）	

- 出力：電圧を設定する
- 出力の状態：出力の状態を設定

# 出力状態 | ブリッジ制御

- 出力の状態でモータの正負の向きが決まる
- 2bitの対応表は以下の通り

IN2	IN1	モータ（ロボット）の状態
0	0	スタンバイ / 惰走
0	1	正転（前進）
1	0	逆転（後退）
1	1	ブレーキ

- 惰走：電圧の出力は止まるが、モータの慣性で動いてしまうこと

# 出力電圧値

- 入力値によって出力電圧が決まる
- 入力値と出力電圧値の関係は右図の対応表の通り

電圧表

VSET[5..0]	出力電圧	VSET[5..0]	出力電圧
0x00h	予約済み	0x20h	2.57
0x01h	予約済み	0x21h	2.65
0x02h	予約済み	0x22h	2.73
0x03h	予約済み	0x23h	2.81
0x04h	予約済み	0x24h	2.89
0x05h	予約済み	0x25h	2.97
0x06h	0.48	0x26h	3.05
0x07h	0.56	0x27h	3.13
0x08h	0.64	0x28h	3.21
0x09h	0.72	0x29h	3.29
0x0Ah	0.80	0x2Ah	3.37
0x0Bh	0.88	0x2Bh	3.45
0x0Ch	0.96	0x2Ch	3.53
0x0Dh	1.04	0x2Dh	3.61
0x0Eh	1.12	0x2Eh	3.69
0x0Fh	1.20	0x2Fh	3.77
0x10h	1.29	0x30h	3.86
0x11h	1.37	0x31h	3.94
0x12h	1.45	0x32h	4.02
0x13h	1.53	0x33h	4.10
0x14h	1.61	0x34h	4.18
0x15h	1.69	0x35h	4.26
0x16h	1.77	0x36h	4.34
0x17h	1.85	0x37h	4.42
0x18h	1.93	0x38h	4.50
0x19h	2.01	0x39h	4.58
0x1Ah	2.09	0x3Ah	4.66
0x1Bh	2.17	0x3Bh	4.74
0x1Ch	2.25	0x3Ch	4.82
0x1Dh	2.33	0x3Dh	4.90
0x1Eh	2.41	0x3Eh	4.98
0x1Fh	2.49	0x3Fh	5.06

データシート：[http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830\\_manual.pdf](http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830_manual.pdf)

# DRV8830 | 確認項目

- データシートから、以下のことが判明する
  - 通信方式：I2C
  - スレイブアドレス：Motor1 0x63, Motor2 0x64
  - レジスタ：0x00
  - 入出力値：上位6bitが電圧の値、下位2bitがモータの向き
- これらの値を使用して、デバイスにアクセスする

# I2Cでのプログラミングの仕方



# I2Cの使い方

- PythonプログラムでI2Cを使用するにはsmbusをimport  
`import smbus # smbusをインポート`
- デバイスアドレス取得時に使用したバス番号を引数にして初期化  
`bus = smbus.SMBus(1) # 初期化, 引数はBusNumber`

# I2Cの使い方

- 命令レジスタ番号を先頭に複数の命令レジスタ番号に値を書き込む

`write_i2c_block_data(デバイスアドレス, 命令レジスタ番号, 値)`

デバイスアドレス	接続している機器のアドレス (ここではFaBoのDRV8830)
命令レジスタ番号	どのレジスタに書き込むかの番号(0x00)
値	出力の状態設定と出力電圧

- 使用例：

`bus.write_i2c_block_data(0x63, 0x00, [93])`

# 出力状態の例

- ・値が93の場合、以下のようになる

10進数	93		
2進数値	01011101		
VSET[5..0]	010111(0x17)	IN2, IN1	01
出力電圧	1.85V	モータ向き	正転

- ・つまり、正の方向に1.85Vの力で回転する

# DCモータサンプルプログラム

- 必要なimportについて

```
import smbus # PythonでI2Cを使用するために必要
```

```
import time # Sleep関数を使用するため
```

```
from time import sleep
```

```
import math # 円周率piを使用するため
```

# DCモータサンプルプログラム

- モータドライバのアドレス

```
bus = smbus.SMBus(1)          # I2Cバス番号  
SLAVE_ADDRESS_LEFT = 0x64    # 左モータのアドレス  
SLAVE_ADDRESS_RIGHT = 0x63   # 右モータのアドレス
```

- I2Cバス番号とアドレスの取得を指定する

- 確認の仕方は以下コマンドを使用する

```
$ sudo i2cdetect 0  
$ sudo i2cdetect 1
```

# DCモータサンプルプログラム

- ・バス番号とデバイスのアドレスを確認：**sudo i2cdetect 1**を実行
- ・エラーが出ない方がバス番号
  - ・エラーが出ない場合、以下のような結果となる

```
pi@raspberrypi:~ $ sudo i2cdetect 1
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-1.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:   - - - - - - - - - - - - - - - - - - - - - -
10:   - - - - - - - - - - - - - - - - - - - - - -
20:   - - - - - - - - - - - - - - - - - - - - - -
30:   - - - - - - - - - - - - - - - - - - - - - -
40: 40 - - - - - - - - - - - - - - - - - - - - -
50:   - - - - - - - - - - - - - - - - - - - - - -
60:   - - - 63 64 - - - - - - - - - - - - - -
70: 70 - - - - - - - - - - - - - - - - - - - -
pi@raspberrypi:~ $
```

- ・この場合アドレスは、**0x63, 0x64**となる

# DCモータサンプルプログラム

- 2.5秒間前進し、2.5秒間後退する
  1. I2Cの導入  
DRV8830にアクセスするためのライブラリーの導入
  2. 初期化処理  
アドレス、レジスタの設定
  3. 前進、後退消処理  
2.5秒前進後、2.5秒後退後停止
- ダウンロードしたファイルDCMotor.pyを開いてください

# DCモータサンプルプログラム

- import

```
import smbus # I2Cを使用するためのモジュール  
import time # sleepを使用するためのモジュール
```

- 初期化

```
bus = smbus.SMBus(1)          # I2Cを使用するための宣言  
SLAVE_ADDRESS_RIGHT = 0x63 # 右モータ用DRV8830のアドレス  
SLAVE_ADDRESS_LEFT = 0x64 # 左モータ用DRV8830のアドレス  
CONTROL = 0x00                # 出力の設定  
STOP = 0x00                  # モータ停止の値
```

# DCモータサンプルプログラム

- 前進

```
left_sval = 93 # 左モータへの出力の値  
right_sval = 93 # 右モータへの出力の値
```

# 左モータ用DRV8830に前進の値を入力

```
bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT, CONTROL, [left_sval])
```

# 右モータ用DRV8830に前進の値を入力

```
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT, CONTROL, [right_sval])
```

```
sleep(2.5)
```

# DCモータサンプルプログラム

- 後退

```
left_sval = 94 # 左モータへの出力の値  
right_sval = 94 # 右モータへの出力の値
```

# 左モータ用DRV8830に後退の値を入力

```
bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT, CONTROL, [left_sval])
```

# 右モータ用DRV8830に後退の値を入力

```
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT, CONTROL, [right_sval])
```

```
sleep(2.5)
```

# DCモータサンプルプログラム

- 停止

```
# 左モータ用DRV8830に停止の値を入力  
bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT, CONTROL, [STOP])  
  
# 右モータ用DRV8830に停止の値を入力  
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT, CONTROL, [STOP])  
  
sleep(2.5)
```

# DCMotor.pyを実行

- 以下コマンドでDCMotor.pyを実行してロボットカーが動くことを確認してください

```
$ cd 06Sample
```

```
$ python DCMotor.py
```

# 距離センサの操作



# 目的

- 距離センサを使い距離を取得する

# 距離センサ概要

- GP2Y0A21YKを使用した赤外線による距離センサ



# データシート内容確認

- GP2Y0A21YKを使用した赤外線による距離センサ  
[http://akizukidenshi.com/download/ds/sharp/gp2y0a21yk\\_e.pdf](http://akizukidenshi.com/download/ds/sharp/gp2y0a21yk_e.pdf)

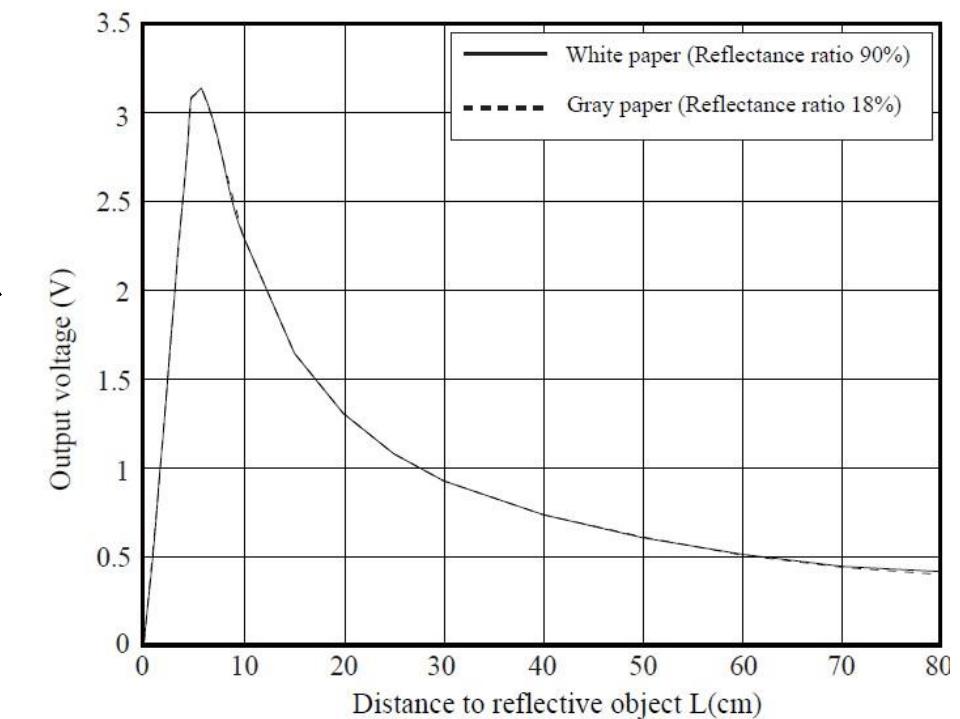
## ■ Features

- Distance measuring range : 10 to 80 cm
- Analog output type
- Package size : 29.5×13×13.5 mm
- Consumption current : Typ. 30 mA
- Supply voltage : 4.5 to 5.5 V

- 測定範囲が10cm～80cm
- アナログ値出力なので  
**ADコンバータ**が必要
- 値を電圧で出力

距離を電圧に  
変換して出力

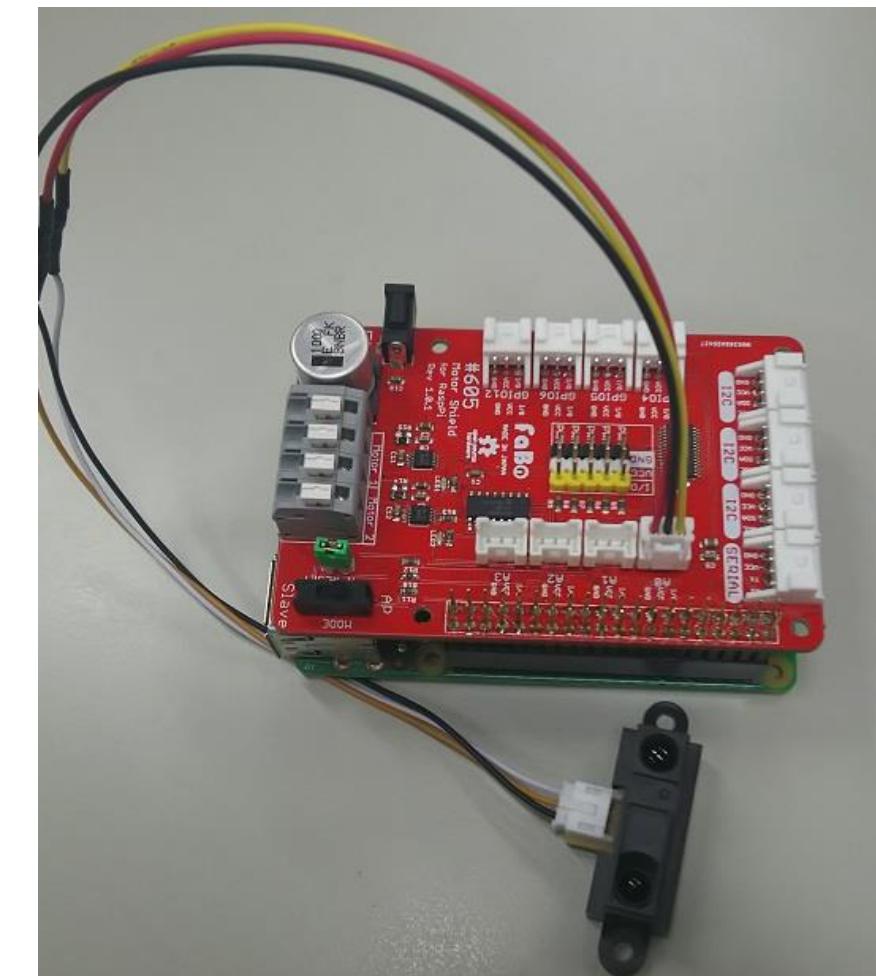
Fig. 2 Example of distance measuring characteristics(output)



# 距離センサとFaboとの接続方法

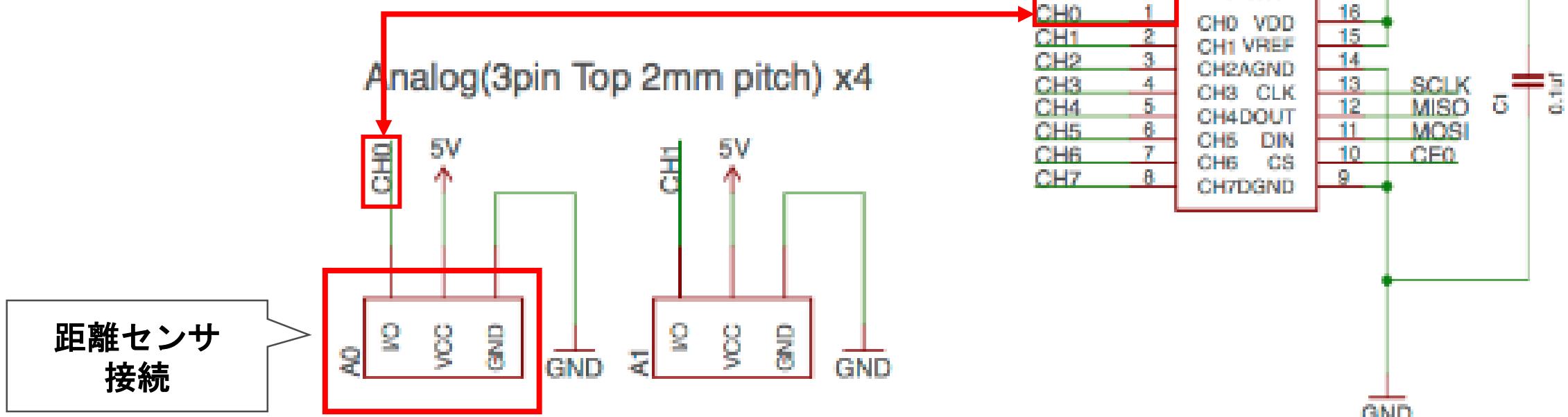
- A0コネクタに接続する場合

I/O	VCC	GND
黄色	赤	黒



# 回路から見た接続

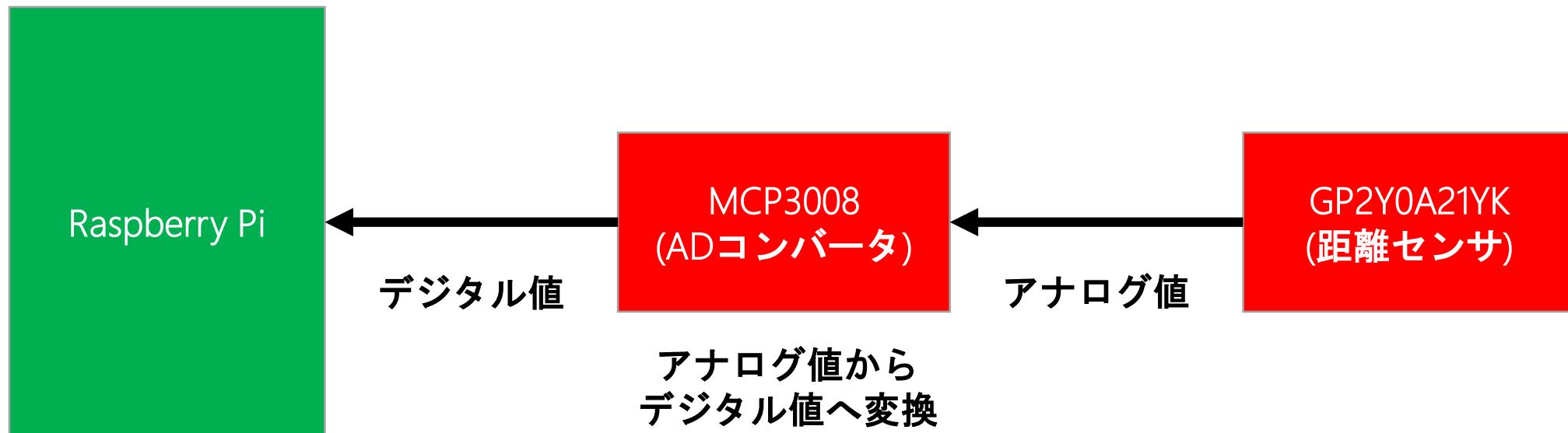
- 距離センサをA0に接続
  - A0のI/OがMCP3008のCH0と接続



回路図 : <http://www.fabo.io/605.html>

# MCP3008

- MCP3008 : ADコンバータ
  - 入力されたアナログ値をデジタル値に変換するデバイス



# MCP3008 | データシート内容確認

- アナログ値をデジタル値へ変換のため、以下のものを確認する
  - データシート  
<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>
    - 販売元などが作成している説明書で、扱い方が記載されている
    - 秋月電子通商に記載されているものを確認
  - 回路図  
<http://www.fabo.io/605.html>
    - FaBoを販売しているGClue記載の回路図

# MCP3008 | 特徴

- 分解能は10bit (1024段階)
  - 出力する値：0～1023
- チャンネル数は8
- 通信方式はSPI

## Features

- 10-bit resolution
- $\pm 1$  LSB max DNL
- $\pm 1$  LSB max INL
- 4 (MCP3004) or 8 (MCP3008) input channels
- Analog inputs programmable as single-ended or pseudo-differential pairs
- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 200 ksps max. sampling rate at  $V_{DD} = 5V$
- 75 ksps max. sampling rate at  $V_{DD} = 2.7V$
- Low power CMOS technology
- 5 nA typical standby current, 2  $\mu$ A max.
- 500  $\mu$ A max. active current at 5V
- Industrial temp range:  $-40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$
- Available in PDIP, SOIC and TSSOP packages

<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>

# SPI | 確認項目

- 通信方式がSPIと判明
- SPIで通信するためには、以下の項目を確認する必要がある
  - 端子又は回路
  - CE番号
  - コンフィギュレーションビット又はレジスタ
  - 入出力の値

# MCP3008 | 端子

- ・チャンネル数は8
- ・CH0はアナログインプット

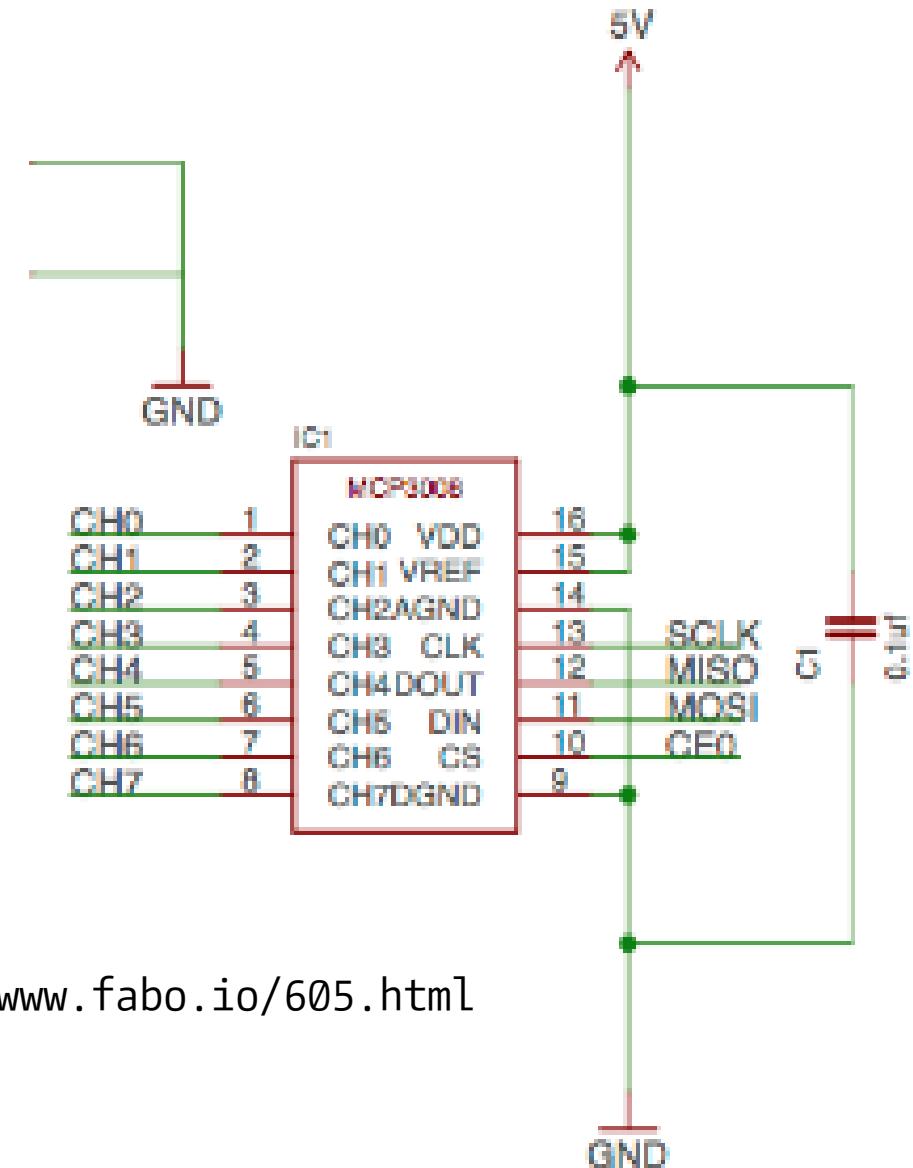
MCP3008 PDIP, SOIC	Symbol	Description
1	CH0	Analog Input
2	CH1	Analog Input
3	CH2	Analog Input
4	CH3	Analog Input
5	CH4	Analog Input
6	CH5	Analog Input
7	CH6	Analog Input
8	CH7	Analog Input
9	DGND	Digital Ground
10	CS/SHDN	Chip Select/Shutdown Input
11	D <sub>IN</sub>	Serial Data In
12	D <sub>OUT</sub>	Serial Data Out
13	CLK	Serial Clock
14	AGND	Analog Ground
15	V <sub>REF</sub>	Reference Voltage Input
16	V <sub>DD</sub>	+2.7V to 5.5V Power Supply
-	NC	No Connection

<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>

# MCP3008 | 端子

- FaBoの回路：  
MCP3008の端子の接続は以下になる

MCP3008	接続先
CLK	SCLK(Raspberry Pi)
DOUT	MISO(Raspberry Pi)
DIN	MOSI(Raspberry Pi)
CS	CE0(Raspberry Pi)
VDD	5V
VRED	5V



- CE番号は**0**になる

# MCP3008 | コンフィギュアービット

- コンフィギュアービット：  
どのCHに接続しているかの値
- CH0は[1000]になる

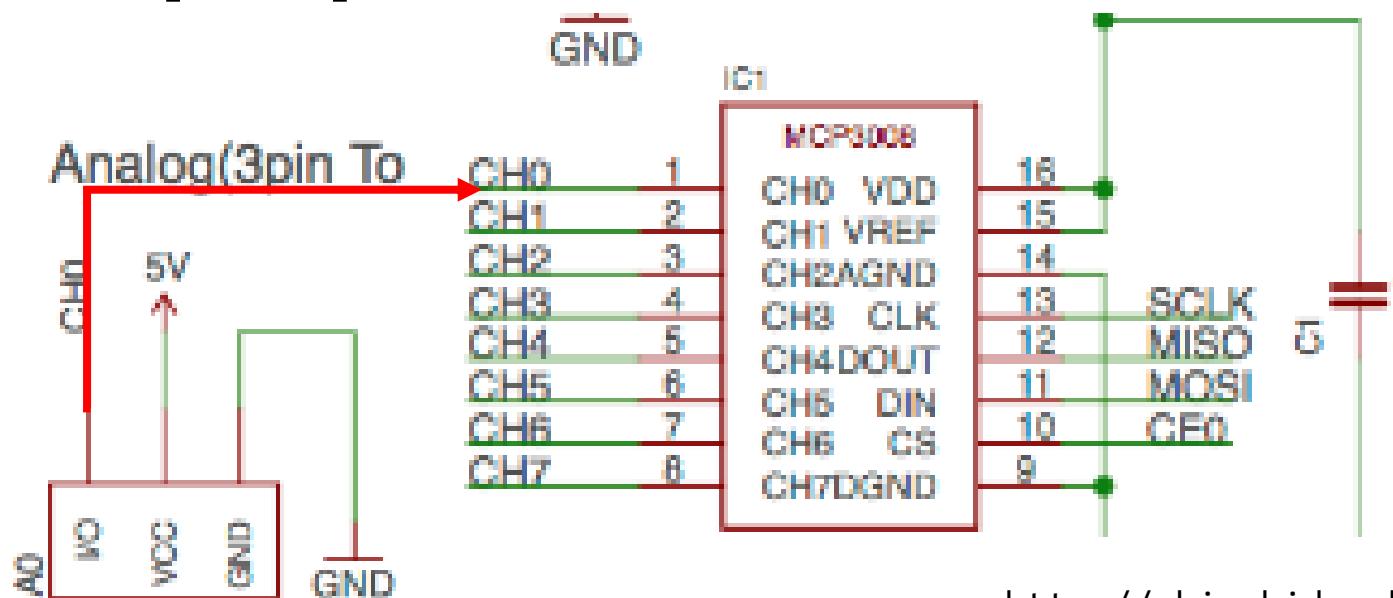


TABLE 5-2: CONFIGURE BITS FOR THE MCP3008

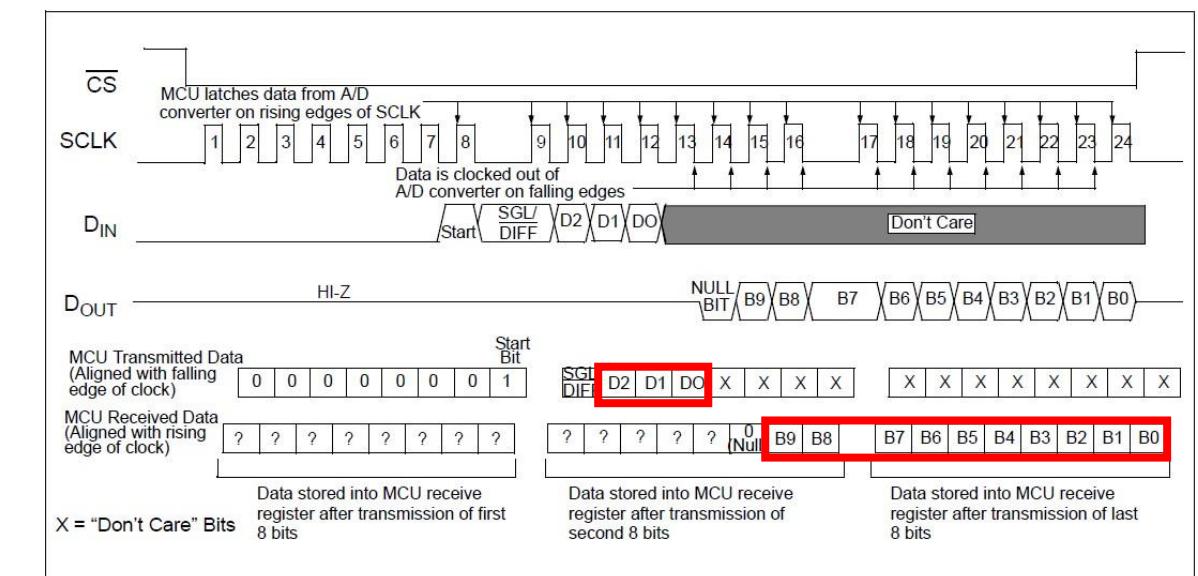
Control Bit Selections	Single /Diff	Input Configuration			Channel Selection
		D2	D1	D0	
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7

<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>

回路図：<http://www.fabo.io/605.html>

# MCP3008 | コンフィギュアービット送信

- コンフィギュアービットを送信して指定したチャンネルに接続されたデバイスの値を取得する
- 赤線で囲まれた所が使用するbit
- D2, D1, D0で使用するチャンネルを送信
- B9~B0の10bitで値を受信



データシート：

<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>

# MCP3008 | 値の出力

- MCP3008の値は以下の式で出力される

## EQUATION 4-2: DIGITAL OUTPUT CODE CALCULATION

$$\text{Digital Output Code} = \frac{1024 \times V_{IN}}{V_{REF}}$$

Where:

$V_{IN}$  = analog input voltage

$V_{REF}$  = analog input voltage

Digital OutPut Code : 出力する値  
 $V_{IN}$  : CHから入力された電圧  
 $V_{REF}$  : 端子VREFに接続された電圧  
1024 : 分解能

データシート : <http://akizukidensi.com/download/ds/microchip/mcp3008.pdf>

# MCP3008 | 値の出力

- 分解能
  - ADコンバータは、基準となる電圧を分解能の値で等分する
  - 入力の値を等分した一番近い値に変換して出力する
- 例：基準5V，分解能1024，入力された値が2Vの場合
  - 基準の電圧を分解能で割る： $5/1024 = 0.0048828125$
  - 入力された値を上記の値で割る： $2/0.0048828125 = 409.6$
  - ADコンバータの結果として409が出力される
- ADコンバータから値を受け取った場合，戻す計算が必要となる

# MCP3008 | 確認項目

- データシートから、以下のことが判明する
  - 通信方式：SPI
  - CE番号：0
  - コンフィギレーションビット：1000
  - 入出力の関係：出力の値 =  $V_{IN} \times 1024 / V_{REF}$
- これらの値を使用して、デバイスにアクセスする

# SPIを使ったプログラミングの仕方



# SPIの使い方

- PythonプログラムでSPIを使用するにはspidevをimport

```
import spidev # spidevをインポート
```

- SpiDevの初期宣言

```
spi = spidev.SpiDev() # spiの初期宣言  
spi.open(0, 0)          # spiのデバイスへの接続
```

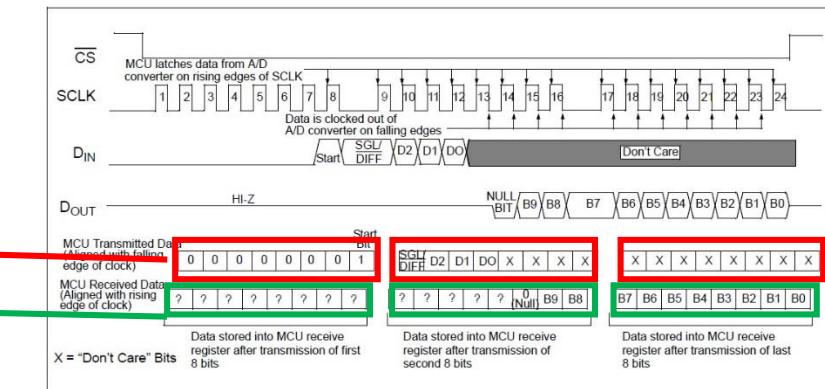
※open(spiの番号, CE番号) spiの番号はデフォルトでは0

# SPIの使い方

- データを送信し、結果を取得する  
**xfer2 ([値]) で、return [値]**

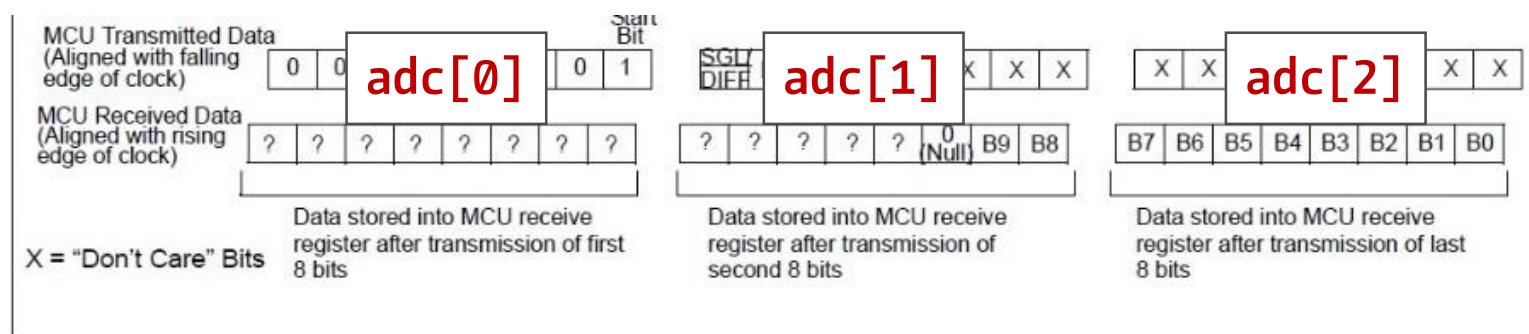
<b>xfer2 ([値])</b>	8bitのデータを送信する リストにすることによって、 8bitのデータを複数送信できる
<b>return [値]</b>	送信の結果を取得

- 使用例  
`adc = spi.xfer2([1, 128, 0])`  
 3要素のリストで格納



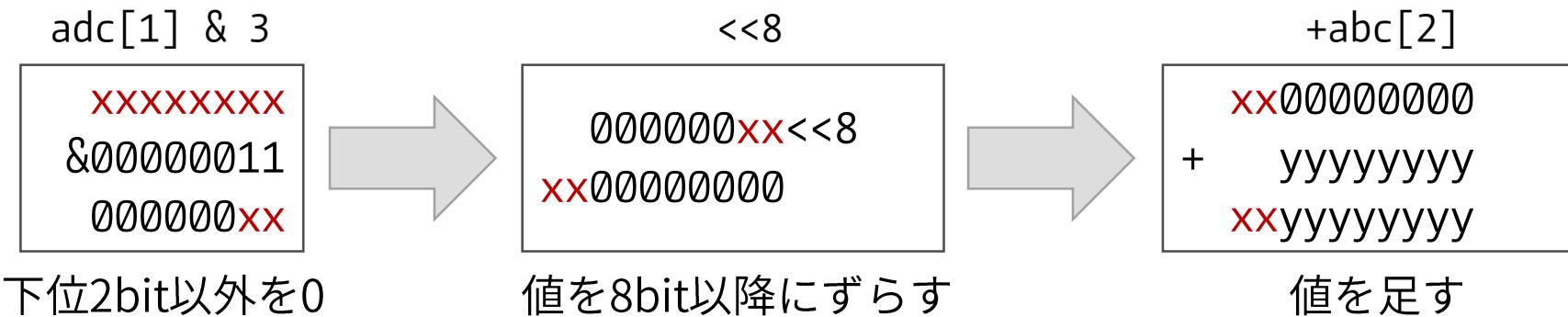
# 値の変換

- 値は以下のように、格納される



データシート：  
<http://akizukidenshi.com/download/ds/microchip/mcp3008.pdf>

- 受け取った値は以下の計算で一つの変数に格納される  
 $((adc[1] \& 3) << 8) + adc[2]$



# 距離センササンプルプログラム

- 距離センサの値を表示するプログラム
  1. SpiDevの導入  
MCP3008にアクセスするためのライブラリーの導入
  2. 初期化処理  
SpiDevの初期設定
  3. MCP3008から値を取得  
MCP3008にコンフィギュレーションビットを送信  
距離センサーからの値を取得
  4. 距離センサーからの値を距離に変換する
- ダウンロードしたファイルDistance.pyを開いてください

# 距離センササンプルプログラム

- import  
import spidev # spiを使用するためのモジュール  
import time # sleepを使用するためのモジュール
- 初期化  
spi = spidev.SpiDev() # spiの初期宣言  
spi.open(0, 0) # spiのデバイスへの接続  
DISTANCE\_PIN = 0 # A0コネクタにDistanceを接続  
SPI\_SPEED = 1000000  
spi.max\_speed\_hz = SPI\_SPEED # SCLK周波数を指定

# 距離センササンプルプログラム

- 距離の取得

try:

```
    while True:  
        # 指定したチャンネルから値を受け取る  
        adc = spi.xfer2([1, (8 + DISTANCE_PIN) << 4, 0])  
        # 受け取った値をアナログ値に変換  
        data = ((adc[1] & 3) << 8) + adc[2]  
        # アナログ値を距離に変換  
        if data <= 17:  
            data = 18  
        distance = 5461 / (data - 17) - 2 # http://appnote.tumblr.com/ を参考  
        print("data : {:3}, distance : {:3} ".format(data, distance))  
        time.sleep(0.05)
```

except:

```
    spi.close() # 終了処理
```

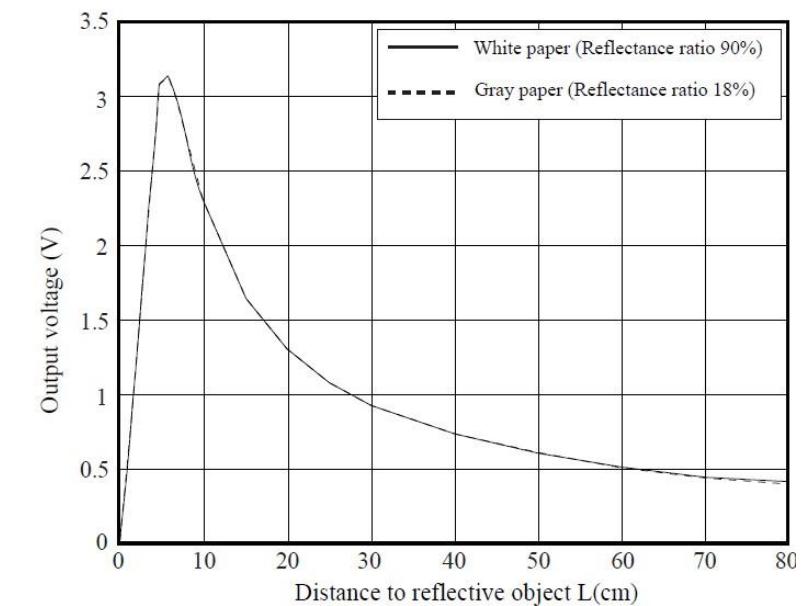
# アナログ値を距離の値に変換

- MCP3008からの距離センサーの値(1~1024)を電圧に変換し、その値を表に照らし合わせて距離に変換する
- 計算式は以下のサイトを参考した

<http://appnote.tumblr.com/>

$$d = 5461 / (V - 17) - 2$$

Fig. 2 Example of distance measuring characteristics(output)



[http://akizukidenshi.com/download/ds/sharp/gp2y0a21yk\\_e.pdf](http://akizukidenshi.com/download/ds/sharp/gp2y0a21yk_e.pdf)

# Distance.pyを実行

- 以下コマンドでDistance.pyを実行して距離センサから距離が取得できることを確認してください

```
$ cd 06Sample
```

```
$ python Distance.py
```

※CTR+Cでプログラムを終了