

上級者向け講習会課題 1

会津大学 R T ミドルウェア講習会

サーボモータの動かし方の確認。サーボモータを動かすプログラムの作成。

目次

1	課題概要.....	1
1.1	サンプルプログラムの実行(課題 1-1).....	1
1.2	プログラムの作成(課題 1-2).....	1
1.3	コンポーネントの作成(課題 1-3).....	1
1.3.1	作成コンポーネント仕様.....	1
2	接続.....	3
2.1	Raspberry Pi との接続方法.....	3
2.2	サーボモータと FaBo の接続方法.....	4
2.3	モバイルバッテリーについて.....	5
2.4	モバイルバッテリーと RaspberryPi の接続	5
2.5	FaBo と電源アダプターの接続.....	6
3	FaBo の仕様.....	7
3.1	Fabo について	7
3.2	FaBo とサーボモータの接続確認.....	7
3.3	PCA9685 について	7
3.3.1	概要.....	7
3.3.2	デバイスアドレス	8
3.3.3	命令レジスタ番号	8
3.3.4	必要情報	8
4	ロボットアームの仕様	9
4.1	ロボットアームについて.....	9
4.2	サーボモータ	9
4.2.1	使用サーボモータ	9
4.3	ロボットアームの角度.....	11
4.4	必要情報	13

5	I2C	15
5.1	I2C(Inter Integrated Circuit; アイ スクエア シー)について	15
5.2	I2C の使い方	15
5.2.1	Python での使い方	15
6	ロボットアーム（サーボモータ）の動かし方	17
6.1	手順	17
6.2	I2C を使用出来る様に sumbus を import	17
6.3	プレスケール値を計算して設定	18
6.4	PWM 出力値を計算	19
6.5	I2C で PWM 出力値を出力し、サーボを動作させる	19
6.6	補足説明	19
6.7	サンプルプログラム	21
7	課題のヒント	25
7.1	サンプルプログラムの実行(課題 1-1)	25
7.1.1	注意事項	25
7.2	プログラムの作成（課題 1-2）	27
7.2.1	注意事項	28
7.3	コンポーネントを作成(課題 1-3)	29
7.3.1	注意事項	29

1 課題概要

1.1 サンプルプログラムの実行(課題 1-1)

サンプルプログラムを実行し、サーボモータの動きを確認

以下の URL からサンプルプログラムをダウンロードし、Raspberry Pi 内にコピーして実行してください。

- ・ <https://rtc-fukushima.jp/wp/wp-content/uploads/2017/11/SampleProgram.zip>
- ・ 01_ServoMotorSample1.py

実行しましたら角度の値を変えてみたり、動かすサーボモータを変えたりして色々実験してみてください。

1.2 プログラムの作成(課題 1-2)

指定した速度でサーボモータが動作するプログラムを作成

以下 URL からプログラムをダウンロードしてください。

穴埋め式になっているので、記載してプログラムを完成させてください。

<https://rtc-fukushima.jp/wp/wp-content/uploads/2017/11/SampleProgram.zip>

- ・ 02_ServoMotorSample1.py

実行結果は台座のサーボが 0 度へ動作しその後 0～180 度に動作。その後グリッパーが開き、閉じし中部のサーボが上下に動きます。

1.3 コンポーネントの作成(課題 1-3)

ロボットアームの 2 軸とグリッパーが動作するコンポーネントを作成

1.3.1 作成コンポーネント仕様

コンポーネント名
ServoMotor
概要

上級者向け講習会課題 1

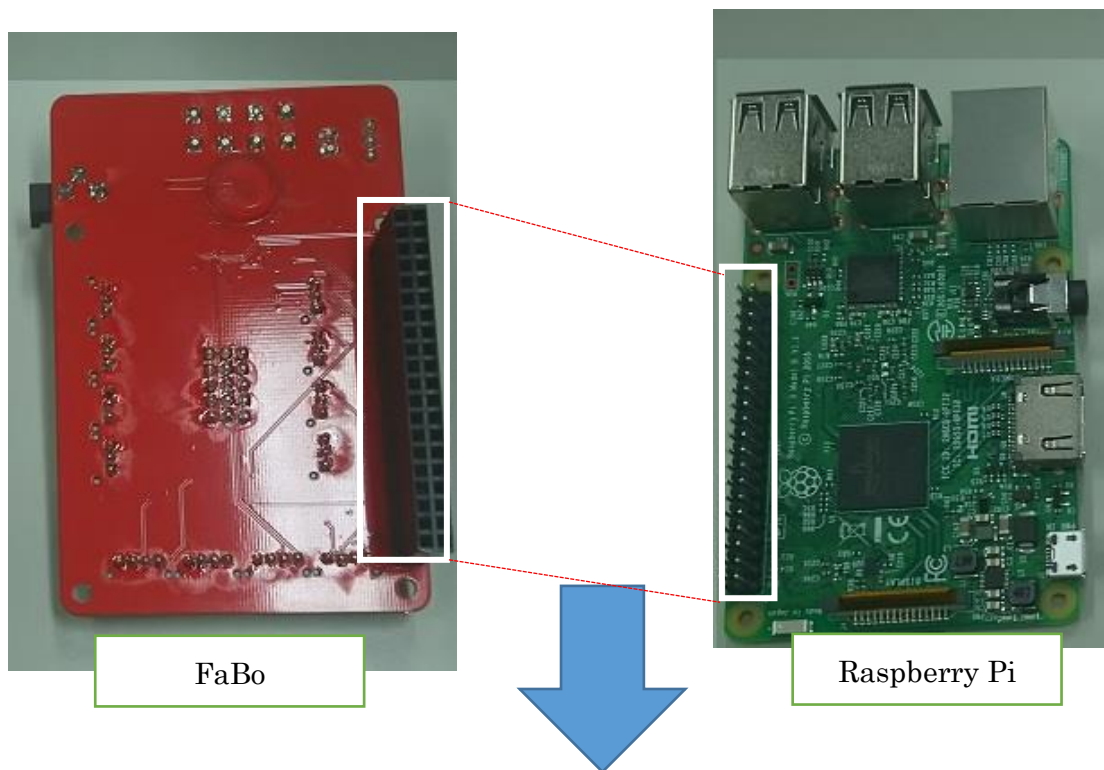
FaBo に接続してあるロボットアームが動作するコンポーネント。起動するとグripper、中間部分、台座部分との順で動作するプログラム。			
ポート名	フローポート	変数	意味
なし	なし	なし	なし
言語		Python	

2 接続

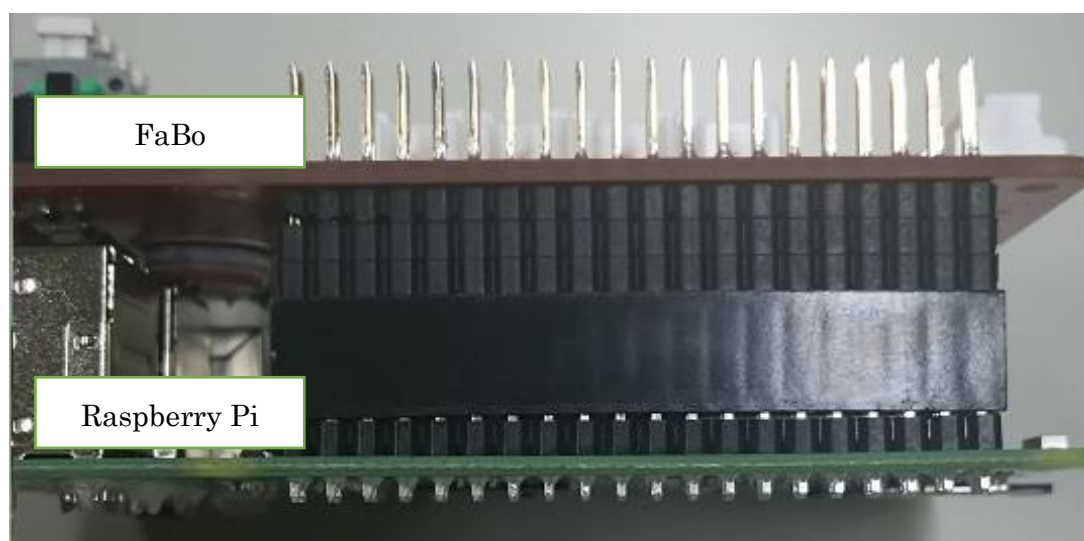
2.1 Raspberry Pi との接続方法

今回は FaBo を Raspberry Pi に接続して使用します。

Raspberry Pi と FaBo を下図のように接続します。



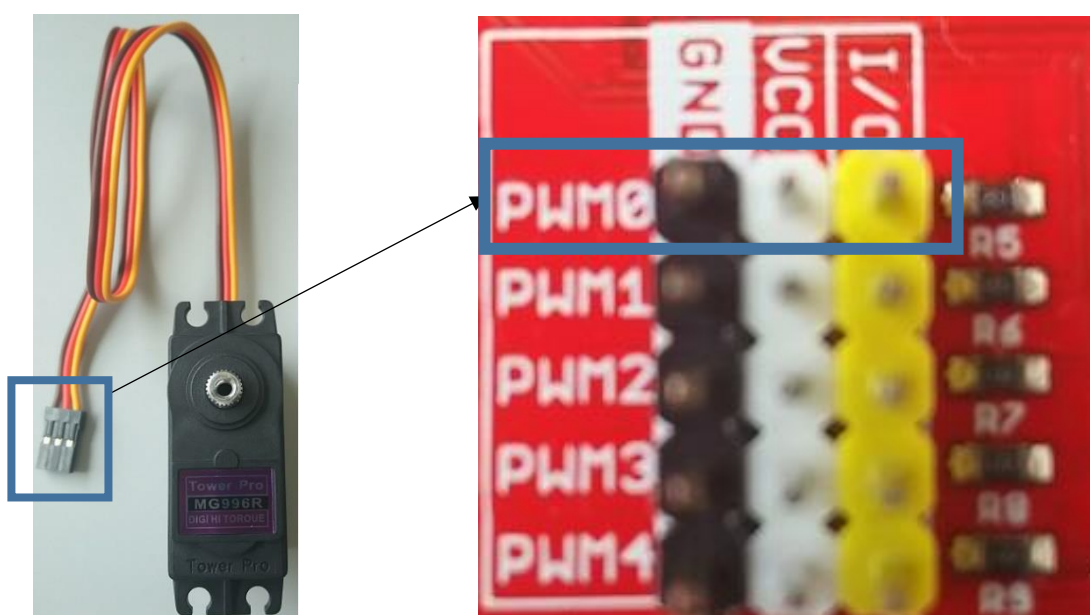
横から見た形



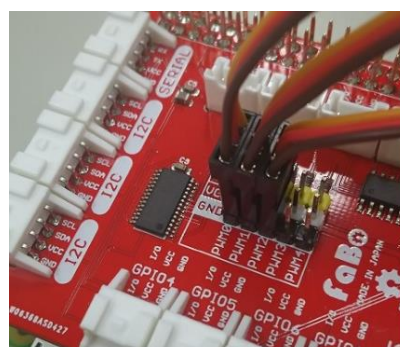
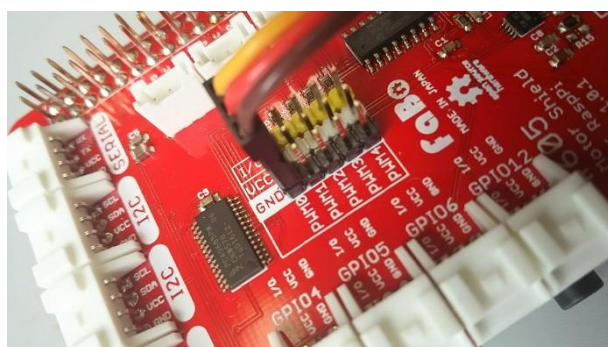
2.2 サーボモータと FaBo の接続方法

サーボモータのコードが黄色、赤、茶色の三色に分かれているのを確認してください。
そのコードの先のプラグを PWM のピンに差し込みます。差し込む場所は茶色：GND 赤：VCC 黄色：I/O になります。

PWM		
GND	VCC	I/O
茶色	赤	黄色



1つ差し込むと左のようになります。右が3つ差し込んだ状態です。



以下の図の様にサーボモータを差しこんでください。

PWM 番号	差し込むサーボモータ
PWM0	グリッパ部分
PWM1	中間部分
PWM2	台座部分

2.3 モバイルバッテリーについて

モバイルバッテリーは RaspberryPi と FaBo に電源を供給するために使用します。

使用モバイルバッテリー：EasyAcc (PB13000MS) 13000mAh



2.4 モバイルバッテリーと RaspberryPi の接続

下図の様にモバイルバッテリーと Raspberry Pi を接続します。接続する場所は、Raspberry Pi の MicroUSB コネクタです。接続するケーブルはモバイルバッテリー付属の MicroUSB ケーブルです。



2.5 FaBo と電源アダプターの接続

下図の様にアダプターと FaBo を接続します。接続する場所は、FaBo のコネクタ部分です。
アダプターの先端のサイズは外径 5.5 mm、内径 2.1 mm ですが、FaBo のコネクタ部分は
外径 4.0 mm、内径 1.7 mm なので 1 つ DC ジャックでサイズを変換しています。



3 FaBo の仕様

3.1 Fabo について

FaBo とは株式会社 GClue が製造販売している、プロトタイプツールです。Raspberry Pi に接続することが出来、センサーやモータ類を簡単に使用することが出来ます。

3.2 FaBo とサーボモータの接続確認

FaBo とサーボモータが接続されているかの確認をします。

下記コマンドで Raspberry Pi と FaBo が正しく接続されているか確認します。

```
$ sudo i2cdetect 1
```

[y/n]を聞かれたら[y]を選択してください。

下図の内容が出力され、[0x40]が表示されることを確認してください。

```
pi@rasp4:~$ 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@rasp4:~$
```

3.3 PCA9685 について

3.3.1 概要

FaBo に組み込まれているサーボモータドライバーです。16 チャンネルの PWM を出力可能で最大 16 個のサーボモータを操作出来ます。通信には I2C を使用します。

以下仕様書に必要な情報が記載されています。

<https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf>

3.3.2 デバイスアドレス

I2C で PCA9865 を制御するには、値を送るためのデバイスアドレスが必要になります。

これは[FaBo とサーボモータの接続確認]で確認している[0x40]です。

3.3.3 命令レジスタ番号

PWM に接続したサーボモータに値を送るには命令レジスタ番号が必要になります。

PWM	命令レジスタ番号			
PWM0	6(0x06)	7(0x07)	8(0x08)	9(0x09)
PWM1	10(0x0A)	11(0x0B)	12(0x0C)	13(0x0D)
PWM2	14(0x0E)	15(0x0F)	16(0x10)	17(0x11)

3.3.4 必要情報

	値	説明
デバイスアドレス	0x40	PCA9865 のデバイスアドレス。I2C で通信時に使用。
分解能	4096	分解能とは何 step けられるかということです。ここでは周期を 4096step にわけられるということになります。
osc clock	25MHz	処理能力
プリスケール値	$= \text{Round}\left(\frac{\text{osc clock}}{4096 * \text{update rate}} - 1\right)$	プリスケール値の求め方 PWM 周波数を設定するための値 update rate はサーボモータの PWM 周期になります。

4 ロボットアームの仕様

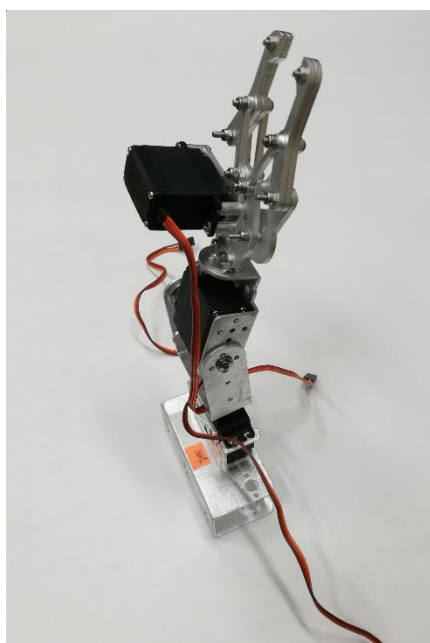
4.1 ロボットアームについて

- ・ サインスマート 3 軸 パレタイジングロボット ロボットアーム

<https://www.sainsmart.com/products/3-axis-desktop-robotic-arm>

性能

回転半径	240mm
高さ	290mm
グripper最大幅	55mm



4.2 サーボモータ

4.2.1 使用サーボモータ

- ・ MG995 メタル ギア デジタル ハイトルク サーボ

<http://www.towerpro.com.tw/product/mg995/>

http://www.electronicoscaldas.com/datasheet/MG995_Tower-Pro.pdf

性能

サイズ	40.7 x 19.7 x 42.9 ミリメートル
重量	55 グラム
角度	180 度
トルク	9.4 キロ/ センチメートル (4.8V)
速度	0.20/60° (4.8V)
動作電圧	4.8V-6.6V
動作温度	0°C - 55°C
PWM 周期	20 ms(50Hz)
デューティサイクル	記載なし

・ MG945 メタル ギア デジタル ハイトルク サーボ

<http://www.towerpro.com.tw/product/mg945/>


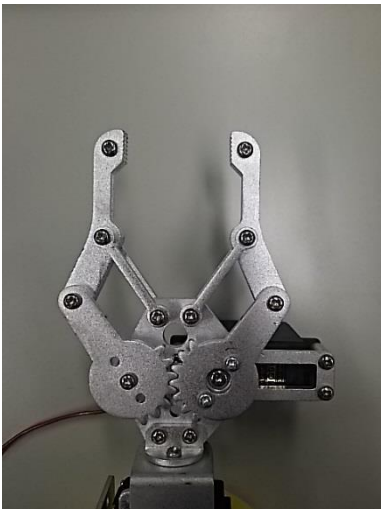

性能

サイズ	40.7 x 19.7 x 42.9 ミリメートル
重量	55 グラム
角度	120 度
トルク	12 キロ/ センチメートル (4.8V)
速度	0.23/60° (4.8V)
動作電圧	4.8V-6.6V
動作温度	0°C - 55°C
PWM 周期	記載なし
デューティサイクル	記載なし


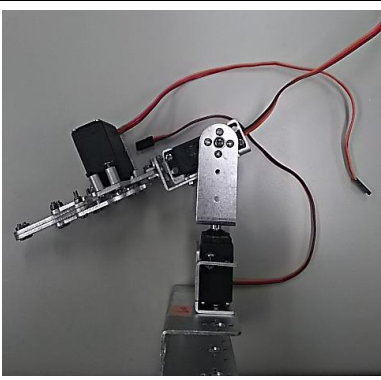
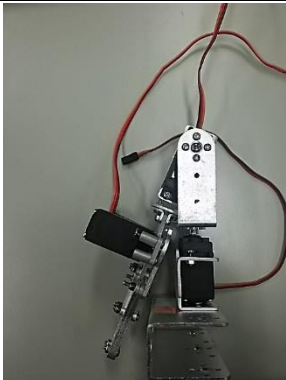
4.3 ロボットアームの角度

サーボモータは 0～180 度まで動作しますが実際の所、ロボットアームはハード上の問題により稼働角度は小さいです。以下に実際の角度を記載します。

- ・ グリッパ部分

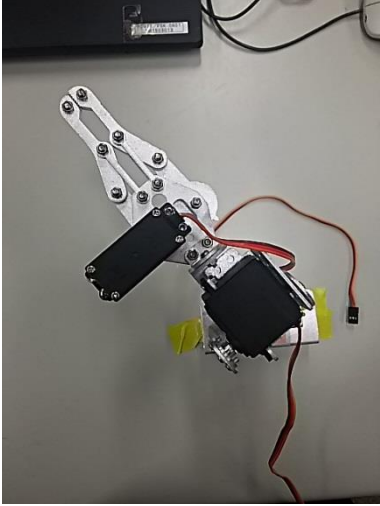
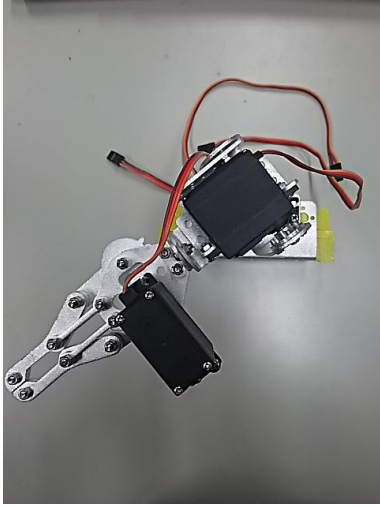
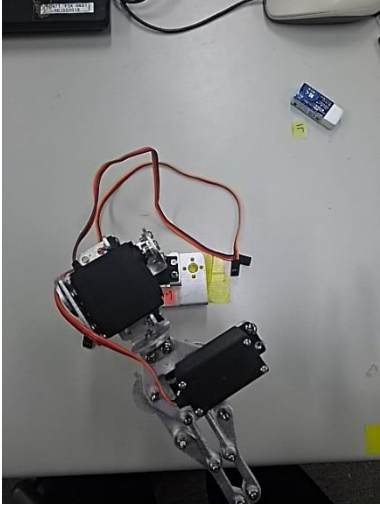
最大(140 度)	中間(110 度)	最小 (80 度)
		

- ・ 中間部分

最大(180 度)	中間(75 度)	最小(35 度)
		

上級者向け講習会課題 1

- ・ 台座部分

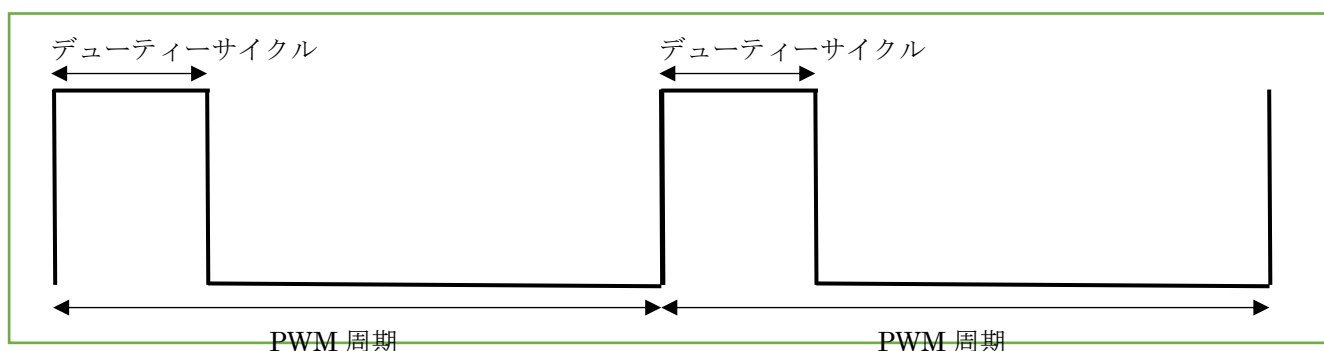
最大(180 度)	中間(90 度)	最小(0 度)
		

4.4 必要情報

サーボモータの PWM 制御に必要な情報は[PWM 周期]と[デューティサイクル値]です。

・ PWM 制御

PWM 制御とはモータにかける電圧を変化させるのではなく、一定の電圧をかけている時間 (ON) とかけていない時間 (OFF) の割合を変化させることによってモータの回転数を制御する方法です。



	値	説明
PWM 周期	50Hz	ON の時間と OFF の時間の周期 今回は MG995 の 50Hz を使用します。
デューティサイクル	0.5～ 2.4ms	電圧をかけている時間。電圧をかけていることが出来る時間は基本的にサーボモータごとに決まっています。 本来は各サーボのデータシートを参考にするのですが、記載がないので今回使用サーボモータと同社のサーボモータの SG90 とこちらが手で確認した値を参考にしたいと思います。 SG90 ・ http://akizukidenshi.com/download/ds/towerpro/SG90_a.pdf

5 I2C

5.1 I2C(Inter Integrated Circuit; アイ スクエア シー)について

フィリップ社が開発した周辺デバイスとのシリアル通信の方式です。

SDA,SCL の 2 本信号線だけでデバイスを制御し、複数のデバイスに並列して接続し使用出来ます。

5.2 I2C の使い方

5.2.1 Python での使い方

- ・ import

Python プログラムで I2C を使用するには[smbus]を import します。

```
import smbus # smbus をインポート
```

- ・ 宣言

デバイスアドレス確認時に使用したバス番号を引数にして初期宣言をします。

```
bus = smbus.SMBus(1) #初期宣言 引数は BusNumber ([sudo i2cdetect 1]の 1)
```

今回の講習会では以下の関数を使用して値のやり取りを行います。

- ・ read_byte_data(デバイスアドレス , 命令レジスタ番号)

命令レジスタ番号に書き込まれている値を読み取る

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 PCA9865）
命令レジスタ番号	どのレジスタに書き込むかの番号

以下の様に使用します。

```
value = bus.read_byte_data(0x40, 0x00)# PCA9865(0x40)の 0x00 から値を読み取る
```

上級者向け講習会課題 1

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

命令レジスタ番号に値を書き込む

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 PCA9865）
命令レジスタ番号	どのレジスタに書き込むかの番号
値	書き込む値

```
bus.write_byte_data(0x40, 0x00, 0x00) # PCA9865(0x40)の 0x00 に 0x00 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, 1 つ以上の値)

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

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 PCA9865）
命令レジスタ番号	どのレジスタに書き込むかの番号
1 つ以上の値	複数の値を書き込むことによって複数の命令レジスタに値を書き込める。

```
bus.write_i2c_block_data(0x40, 0x06, [1,2,3,4])  
# PCA9865(0x40)の 0x06 に 1、0x07 に 2 、0x08 に 3 、0x09 に 4 を書き込む
```

6 ロボットアーム（サーボモータ）の動かし方

ロボットアームを動かすために必要な説明をします。

6.1 手順

以下の手順で行います。

1. I2C を使用出来る様に `smbus` を `import`
2. プレスケール値を計算して設定
3. PWM 出力値を計算
4. I2C で PWM 出力値を出力し、サーボを動作させる

6.2 I2C を使用出来る様に `smbus` を `import`

・ `import`

Python プログラムで I2C を使用するには `[smbus]` を `import` します。

```
import smbus # smbus をインポート
```

・ 宣言

デバイスアドレス確認時に使用したバス番号を引数にして初期宣言をします。

```
bus = smbus.SMBus(1) #初期宣言 引数は BusNumber ([sudo i2cdetect 1]の 1)
```

6.3 プレスケール値を計算して設定

・プレスケール値

プレスケール値の計算式は以下になります。

$$\text{プレスケール値} = \text{Round}\left(\frac{25\text{MHz}}{4096 \cdot f[\text{Hz}]} - 1\right) \quad f = \text{PWM 周期なので}$$

$$\text{プレスケール値} = \text{Round}\left(\frac{25\text{MHz}}{4096 \cdot 50\text{Hz}} - 1\right)$$

$$\text{プレスケール値} = \text{Round}\left(\frac{25000000}{4096 \cdot 50} - 1\right)$$

$$\text{プレスケール値} = 121$$

となります。

・プレスケール値設定

プレスケール値は命令レジスタ番号[0xFE]に書き込みます。設定に時は Sleep モードを設定する必要があります。

```
oldmode = bus.read_byte_data(0x40, 0x00)#現在のモード取得
newmode = (oldmode & 0x7F) | 0x10 #Sleep モードの値作成
bus.write_byte_data(0x40, 0x00, newmode)#Sleep モード設定
bus.write_byte_data(0x40, 0xFE, 121)
# PCA9865(0x40)の 0xFE にプレスケール値を書き込む
bus.write_byte_data(0x40, 0x00, oldmode)#Sleep モード解除
```

Sleep モードを設定するには命令レジスタ番号：0x00 の 4bit 目に 1 を入れる必要があります。

従って[&0x7f]で 8bit 目以降を 0 に変換、[|0x10]で 4bit 目を 1 にしています。

式	oldmode (値不明)&7F	式	0000xxxxxxxx 0x10
論理積 両方が 1 の時 1	xxxxxxxxxxxx 000011111111	論理和 片方が 1 の時 1	0000xxxxxxxx 00010000
上位 4bit が 0 になる	0000xxxxxxxx	4bit 目が 1 になる	xxxx1xxxx

6.4 PWM 出力値を計算

サーボモータを動かすために PWM 出力値を計算します。この計算に必要な情報は以下になります。

- ・ PWM 周期
 - ・ 20ms

- ・ デューティサイクル値
 - ・ 2.4ms

※デューティサイクル値は 0.5～2.4ms の間なら問題のですが、ここでは最大値を使用します。

- ・ 分解能
 - ・ 4096

計算式は以下になります。

PWM 出力値 = デューティサイクル ÷ PWM 周期 × 分解能

各パラメータに値を入れると以下の結果になります。

PWM 出力値 = $2.4 \div 20 \times 4096 = 491$

6.5 I2C で PWM 出力値を出力し、サーボを動作させる

求めた PWM 出力値を使用してサーボモータを動かします。今回は台座部分のサーボモータを動かします。その場合の値の与え方は以下になります。

```
bus.write_i2c_block_data(0x40, 0x0E, [0, 0, 491 & 0xFF, 491 >> 8])
```

デューティサイクル値の最大を使っているのでサーボモータは 180 度を差します。アームは真っ直ぐになるはずですが。

6.6 補足説明

今回値は 0x10 と 0x11 に値を入れています。さらに値を入れる時に [&0xFF]、[>>8] をしています。理由としては 0x0E, 0x0F は今回使用しないからです。これは他の PWM の命令レジスタ番号の 0x06, 0x07, 0x0A, 0x0B も同じです。

上級者向け講習会課題 1

PWM	命令レジスタ番号			
PWM0	6(0x06)	7(0x07)	8(0x08)	9(0x09)
PWM1	10(0x0A)	11(0x0B)	12(0x0C)	13(0x0D)
PWM2	14(0x0E)	15(0x0F)	16(0x10)	17(0x11)

[&0xFF]、[>>8]をしている理由ですが、サーボモータの値は 4096（12bit）まで使用できます。しかし、0x10 に入れられる値は 256(8bit)までです。それ以上の値を入れた場合は上位 4bit を 0x11 の下位 4bit に入れることになります。

式	491&FF	式	491>>8
論理積 両方が 1 の時 1	000111101011 000011111111	右に 8 シフト	000111101011
上位 4bit を 0	11101011	上位 4bit の値のみ	0001

命令レジスタ番号	値	
0x10	0000 0000	8bit すべて使用可能
0x11	xxx* 0000	下位 4bit のみ使用可能

6.7 サンプルプログラム

サーボモータを動かすサンプルプログラムを使い説明します。

01_ServoMotorSample1.py

```
import RPi.GPIO as GPIO
from time import sleep
import smbus # I2C module import
import math

bus = smbus.SMBus(1) #初期宣言

PCA9685_ADDRESS = 0x40 #FaBo のデバイスアドレス宣言

bus.write_byte_data(PCA9685_ADDRESS, 0x00, 0x00) #初期化
#プレスケール値の計算
freq = 50
resolution = 4096.0          # 12-bit
prescaleval = 25000000.0     # 25MHz
prescaleval /= resolution
prescaleval /= float(freq)
prescaleval -= 1.0
#丸め処理
prescale = int(math.floor(prescaleval+0.5))
oldmode = bus.read_byte_data(PCA9685_ADDRESS, 0x00) #現在の値を読み込む
newmode = (oldmode & 0x7F) | 0x10 #4bit 目を 1 にする
#プレスケール値の書き込み
bus.write_byte_data(PCA9685_ADDRESS, 0x00, newmode) #Sleep モード設定
bus.write_byte_data(PCA9685_ADDRESS, 0xFE, prescale) #プレスケール値の書き込み
bus.write_byte_data(PCA9685_ADDRESS, 0x00, oldmode) #Sleep モード解除
sleep(0.005)
bus.write_byte_data(PCA9685_ADDRESS, 0x00, oldmode | 0xa1) #7,5,0bit 目を 1 にする
#PWM 出力値の計算
DutyCycle_Max=2.4 #デューティサイクル最大値
DutyCycle_Min=0.5 #デューティサイクル最小値
PwmPeriod = 20.0 #PWM 周期

Max=float((DutyCycle_Max/PwmPeriod)*resolution)
Min=float((DutyCycle_Min/PwmPeriod)*resolution)

angle1= 120 #角度 1
angle2= 90 #角度 2
duty_Servo1_1 = int(Min + (Max - Min)/180 * angle1) #角度を PWM 出力値に変換(120 度
を変換)
duty_Servo1_2 = int(Min + (Max - Min)/180 * angle2) #角度を PWM 出力値に変換(90 度
を変換)
```

次のページに続きます。

上級者向け講習会課題 1

```
#サーボモータの値を書き込む
Servo_Adress=6#動かすサーボモータを指定

bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_2 &
0xFF , duty_Servo1_2 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
```

このサンプルプログラムを動かしますとアームのグリッパー部分が動作します。

・imPort とアドレス

必要なライブラリーのインポートと FaBo (PCA9685) のデバイスアドレスを宣言しています。

```
import RPi.GPIO as GPIO
from time import sleep
import smbus # I2C module import
import math

bus = smbus.SMBus(1) #初期宣言

PCA9685_ADDRESS = 0x40 #FaBo のデバイスアドレス宣言
```

・プレスケール値の計算と設定

プレスケーラ値の計算とセットを行っています。プログラムを作成するときはこのまま使用しても問題ありません。

```
bus.write_byte_data(PCA9685_ADDRESS, 0x00, 0x00) #初期化
#プレスケール値の計算
freq = 50
resolution = 4096.0          # 12-bit
prescaleval = 25000000.0     # 25MHz
prescaleval /= resolution
prescaleval /= float(freq)
prescaleval -= 1.0
#丸め処理
prescale = int(math.floor(prescaleval+0.5))
oldmode = bus.read_byte_data(PCA9685_ADDRESS, 0x00) #現在の値を読み込む
newmode = (oldmode & 0x7F) | 0x10 #4bit 目を 1 にする
#プレスケール値の書き込み
bus.write_byte_data(PCA9685_ADDRESS, 0x00, newmode) #Sleep モード設定
bus.write_byte_data(PCA9685_ADDRESS, 0xFE, prescale) #プレスケール値の書き込み
bus.write_byte_data(PCA9685_ADDRESS, 0x00, oldmode) #Sleep モード解除
sleep(0.005)
bus.write_byte_data(PCA9685_ADDRESS, 0x00, oldmode | 0xa1) #7,5,1bit 目を 1 にする
```

- ・ サーボモータへ与える値を計算

PWM 出力値の計算です。角度の 120 や 90 を別の角度に変更すれば、その角度で PWM 出力値を計算します。

```
DutyCycle_Max=2.4 #デューティサイクル最大値
DutyCycle_Min=0.5 #デューティサイクル最小値
PwmPeriod = 20.0 #PWM 周期

#PWM 出力の計算
Max=float((DutyCycle_Max/PwmPeriod)*resolution) #PWM 出力の最大値
Min=float((DutyCycle_Min/PwmPeriod)*resolution) #PWM 出力の最小値
angle1= 120 #角度 1
angle2= 90 #角度 2

duty_Servo1_1 = int(Min + (Max - Min)/180 * angle1) #角度を PWM 出力値に変換(120 度
を変換)
duty_Servo1_2 = int(Min + (Max - Min)/180 * angle2) #角度を PWM 出力値に変換(90 度
を変換)
```

- ・ サーボモータの値を書き込む

サーボモータに値を書き込んでいます。今回は PWM0 に接続しているので命令レジスタ番号は 6 です。PWM1 に接続しているサーボモータを動かしたいときは 6 を 10 に、PWM2 に接続しているサーボモータ

上級者向け講習会課題 1

タを動かしたければ 6 を 14 に変えればが動きます。

```
#サーボモータの値を書き込む
Servo_Adress=6#動かすサーボモータを指定

bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_2 &
0xFF , duty_Servo1_2 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
```

PWM1

```
#サーボモータの値を書き込む
Servo_Adress=10#動かすサーボモータを指定

bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_2 &
0xFF , duty_Servo1_2 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
```

PWM2

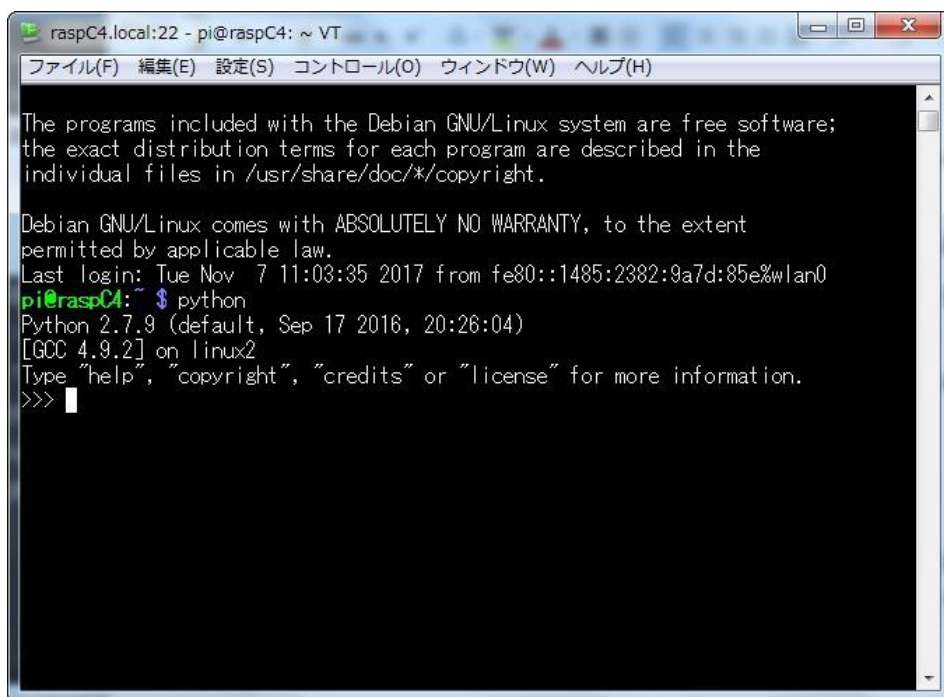
```
#サーボモータの値を書き込む
Servo_Adress=14#動かすサーボモータを指定

bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_2 &
0xFF , duty_Servo1_2 >>8])
sleep(1)
bus.write_i2c_block_data(PCA9685_ADDRESS, Servo_Adress ,[0, 0, duty_Servo1_1 &
0xFF , duty_Servo1_1 >>8])
sleep(1)
```

7 課題のヒント

7.1 サンプルプログラムの実行(課題 1-1)

プログラムを修正して再実行する場合、対話モードを使用してみてください。コマンドで [python] と打ち込みますと、下図の様に対話モードになります。



```
raspC4.local:22 - pi@raspC4: ~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov  7 11:03:35 2017 from fe80::1485:2382:9a7d:85e%wlan0
pi@raspC4:~$ python
Python 2.7.9 (default, Sep 17 2016, 20:26:04)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

ここにサンプルプログラムをコピーして貼り付けするとそのままプログラムの内容が実行されます。エラー内容や現在の値がすぐ確認できるので色々試すのには便利です。

7.1.1 注意事項

7.1.1.1 限界角度について

角度を変えてサーボモータを動かす時、誤ってハードの限界以上の角度を入れ、サーボモータに負荷がかかっている状態になる場合があると思います。その場合は速やかに値を入れ直してハードの許容内の角度に戻してください。対話モードのときは問題ない角度を入力出来る様にあらかじめ用意をしていてください。プログラム実行時の時は **Ctrl+C** でプログラム終了後、対話モードで問題ない角度を入れるか、問題ないプログラムを実行してください。

7.1.1.2 1度で動く角度について

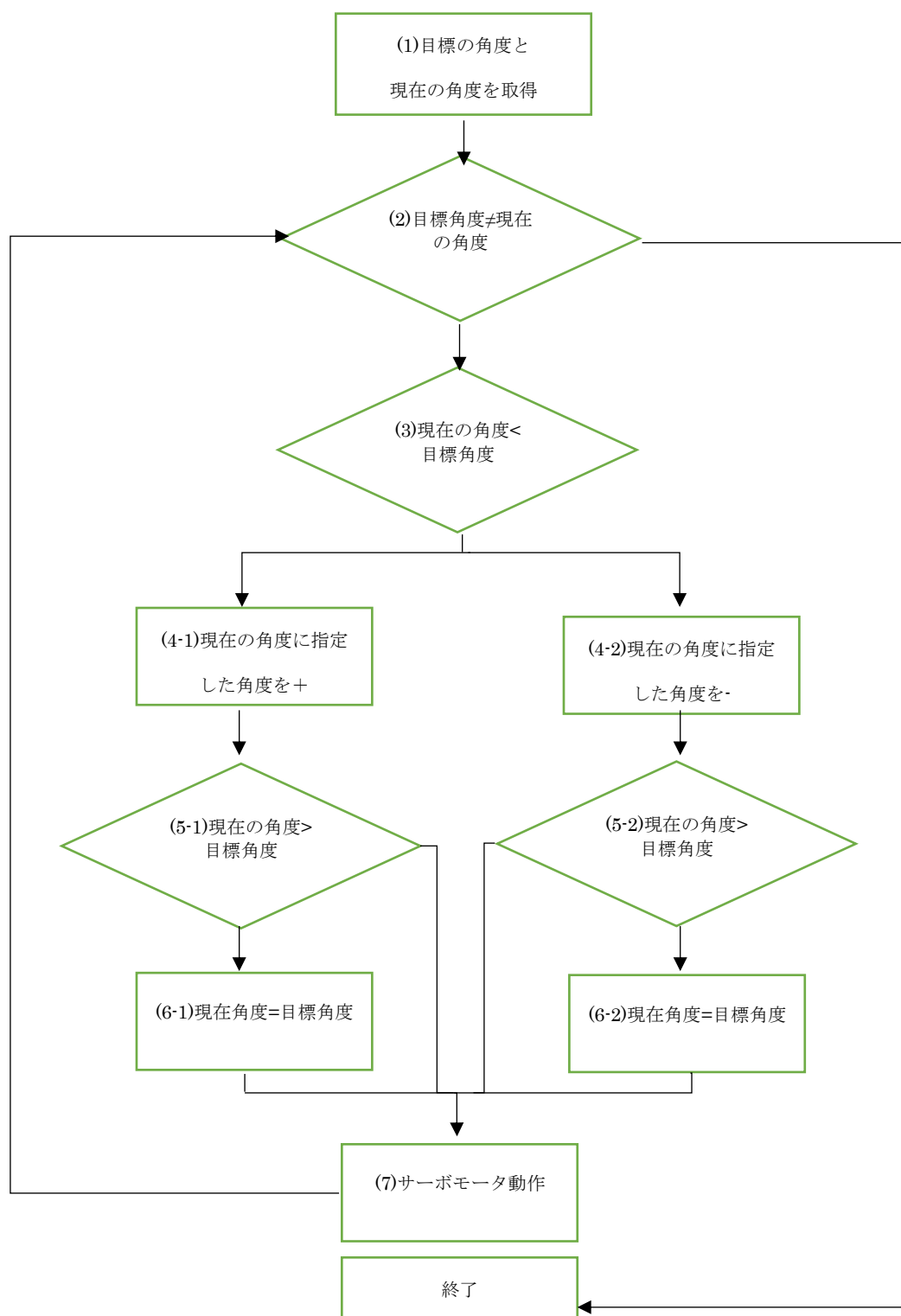
角度を与えてサーボモータを動かす時に、大きな変化量を連続で与えると電流不足になり、Raspberry Pi が落ちる、又は操作不能になる時があります。その場合は、サーボモータに触れてみて熱を帯びていないか確認してください。熱を帯びている場合は、再度アクセスして

上級者向け講習会課題 1

Raspberry Pi を[sudo halt]でシャットダウンしてください。その後、給電しているケーブルを抜いてサーボモータの熱が冷めるのを待ってください。

7.2 プログラムの作成（課題 1-2）

現在のサーボモータの値を取得後、目標角度に指定した角度ずつ動作するプログラムのフローチャートは以下になります。



1. サーボモータの現在の角度と目標の角度を取得。終了後(2)判定に移行。
2. ループの開始判定。現在の角度と目標の角度を比較して異なる値なら(3)判定に移行。同じなら(8)移行
3. 現在の角度と目標の角度を比較。現在の角度が目標の角度より小さい場合(4-1)に移行。大きい場合は(4-2)に移行。
4. -1 現在の角度に指定した角度をプラスする。終了後(5-1)に移行する。
-2 現在の角度に指定した角度をマイナスする。終了後(5-2)に移行する。
5. -1 現在の角度が目標の角度を上回っているか判定。上回っている場合(6-1)に移行。上回っていない場合、(7)に移行
-2 現在の角度が目標の角度を下回っているか判定。下回っている場合(6-2)に移行。下回っていない場合、(7)に移行
6. -1 現在の角度に目標の角度を代入する。終了後(7)に移行する。
-2 現在の角度に目標の角度を代入する。終了後(7)に移行する。
7. サーボモータに現在の角度を入力。入力後 1 秒間 `sleep` する。終了後(2)に移行する。
8. プログラムを終了

現在の角度の取得方法のサンプルプログラムをダウンロードし、**Raspberry Pi** 内にコピーして実行してください。

・ <https://rtc-fukushima.jp/wp/wp-content/uploads/2017/11/SampleProgram.zip>

02_ReadSample.py

7.2.1 注意事項

現在の角度の取得する時、サーボモータを 1 度も動かしていない状態だと以下の値を取得する。

duty: 4096.0

angle 1847.36842105

この値を現在の角度としてプログラムを走らせると、ハードの限界を超えた駆動をしようとして故障の原因になるので注意してください。

プログラムを走らせる場合は、事前に適当な角度を入れてサーボモータを動かしてください。

7.3 コンポーネントを作成(課題 1-3)

プログラムの作成(1-2)で作成したコンポーネントを元に作成してください。onInitialize、onActivated 部分に、宣言及び、プレスケーラ値の設定。onExecute に指令値を入れる様にしてください。

7.3.1 注意事項

7.3.1.1 指令値を与える周期

コンポーネントは基本的に実行周期が 1000.0 で設定されています。これは 1 秒間に 1000.0 回 onExecute が実行されるということです。ここで以下の様に 1 回実行される毎にアームが 60 度と 90 度の間を行き来するプログラムを実行いたします。

```
def onExecute(self, ec_id):
    self.count+=1 #onExecute の実行回数
    if self.count %2 ==1:
        duty=296 # 90 度の PWM 出力値
        self.bus.write_i2c_block_data(PCA9685_ADDRESS,6,[0, 0, duty &
0xFF , duty >>8])
    else:
        duty=232 # 60 度の PWM 出力値
        self.bus.write_i2c_block_data(PCA9685_ADDRESS,6,[0, 0, duty &
0xFF , duty >>8])
    return RTC.RTC_OK
```

その場合、1 秒間に 1000 回もアームを動かそうとして多大な負荷がかかり、Raspberry Pi が落ちたり、サーボモータが壊れたりする原因になります。

以下の方法で周期を変更してください。

- ① プログラムを書くときに指令値を与えた後に sleep 関数を使用
- ② コンポーネントを作成する時点で実行周期を変更
- ③ RTSystemEditor 上でアクティベート前に変更

