ラズベリーパイの初期設定から徹底解説‼ 各種センサーの使い方をイチから学べます！

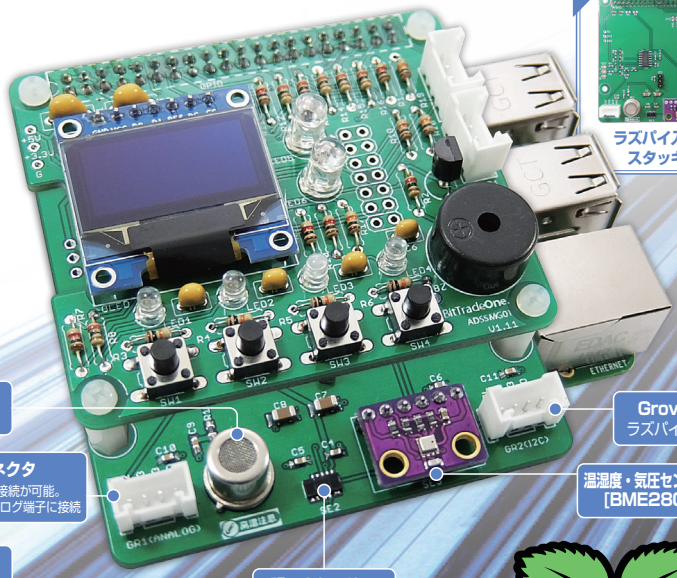
ラズパイセンサーボード総力特集!!

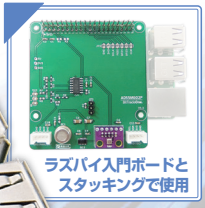
TAKE FREE

AdB
Assembly Desk Books

特集 1 最初の一歩！Raspbianのインストールと初期設定！

特集 2 明るさ・近接センサーを使って近づく物体を検出しよう！





特集 3 アナログ入力電圧を測定してみよう！

vol.04
201908
Assembly Desk Books

ラズパイセンサーボード総力特集!!

本誌について

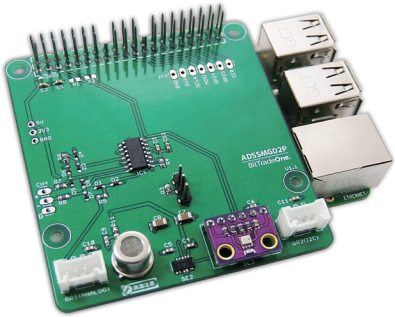
Assembly Desk Books はビット・トレード・ワンが発行するエンジニアやエンジニアを目指す方々に向けたフリーペーパーです。シェルスクリプトマガジンとの連動企画による「ラズパイセンサーボード」の解説や昨今話題の IoT を手軽に試すことができる情報が満載です。
Assembly Desk Books Vol.4 では特集記事として、このラズパイセンサーボードに関して Raspbian のインストールと初期設定からアナログ入力電圧の測定までをご紹介します。
ラズパイセンサーボードに関するソースコードについては右記ページにて公開されております。本書とあわせてご活用ください。

https://shell-mag.com/raspi_original_code/

INDEX

ラズパイセンサーボードの入手方法

ラズパイセンサーボードは、ビット・トレード・ワンとシェルスクリプトマガジン編集部が共同で制作したラズパイ用ハードウエアです。



ADSSMG02P

完成品と組み立てキットの２種類を用意します。
いずれもビット・トレード・ワンの公式オンラインショップ（http://btoshop.jp/）、
シェルスクリプトマガジン発行元である
ユニバーサル・シェル・スクリプト研究所のサイト
（https://www.usp-lab.com/ORDER/CGI/PUB.ORDER.CGI）
マルツエレック
（https://www.marutsu.co.jp/http://eleshop.jp/）
共立エレショップ
（http://eleshop.jp/）
および、Amazon.co.jp、シリコンハウス、デジット、千石電商、の各店舗から購入できます。

ビット・トレード・ワン公式オンラインショップ
http://btoshop.jp/

ユニバーサル・シェル・スクリプト研究所
https://www.usp-lab.com/ORDER/CGI/PUB.ORDER.CGI

マルツエレック
https://www.marutsu.co.jp/

共立エレショップ
http://eleshop.jp/

特集1

最初の一歩！ Raspbianのインストールと初期設定！

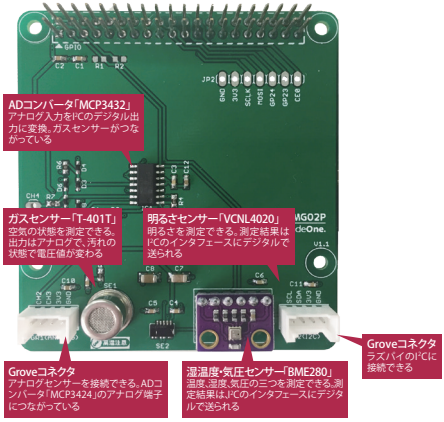
ガジェット **米田聡**

ラズパイセンサーボードはラズパイ入門ボードの第二弾として製作されました。サンプルプログラムでは、搭載されている湿温度・気圧センサー、ガスセンサー、照度センサーの使い方を紹介しますが、まずは最初の一歩、OSのインストールと初期設定をおこないましょう！

何かを測定するために、「センサー」は欠かせない電子部品です。シェルスクリプトマガジンでは、小型コンピュータボード「Raspberry Pi」(以下、ラズパイ)のオリジナル拡張ボード第2弾として、ビット・トレード・ワンと「ラズパイセンサーボード」(図1)を製作しました。湿温度・気圧センサー、ガスセンサー、明るさセンサーの3種類を搭載しています。また、アナログセンサー用とI2C用に、二つのGroveコネクタ*1があります。ガスセンサーなどのアナログセンサーを扱うために、ADコンバータを実装しています。

ラズパイセンサーボード(以下、センサーボード)は単体で使用できるほか、第1弾の「ラズパイ入門ボード」と組み合わせても利用可能です(図2)。

図1　ラズパイセンサーボードの主な機能
湿温度・気圧センサー、ガスセンサー、明るさセンサーを搭載する。



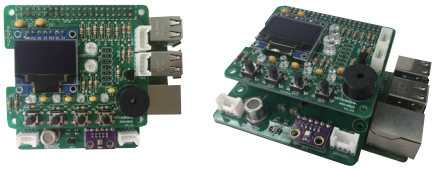
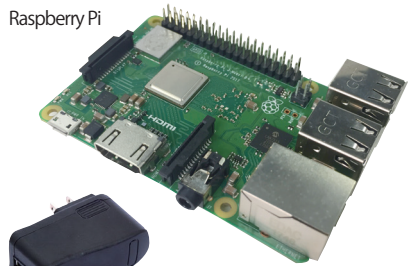


図2　ラズパイ入門ボードとも組み合わせた状態
ラズパイセンサーボードの上にラズパイ入門ボードを実装する。

*1　中国Seeed Studio社が展開している規格「Grove System」に対応したセンサーやディスプレイ、そのほかの表示装置などを接続するためのコネクタ

図3　必要な機材



USB電源アダプタ

Micro-USBケーブル

microSDカード

HDMIケーブル

パソコン用ディスプレイ

パソコン用キーボードとマウス

必要な機材

センサーボードを動かす準備をします。必要な機材は、ラズパイ、空の microSD カード（8G バイト以上）、USB 電源アダプタ（5V2.5A 以上を推奨）、Micro-USB ケーブル、USB 接続のキーボードとマウス、HDMI ケーブル、HDMI 端子付きのディスプレイです（図 3）。USB 接続のキーボードとマウス、HDMI ケーブル、HDMI 端子付きのディスプレイがなくてもパソコンなどからリモートで操作できますが、今回はラズパイのデスクトップ画面から各種操作を実施するので必要です。

図4　インストーラをダウンロード

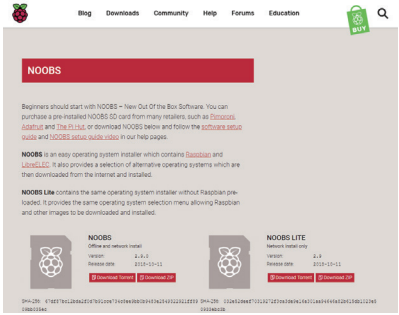


図5　microSDカードにインストーラのファイルをコピー

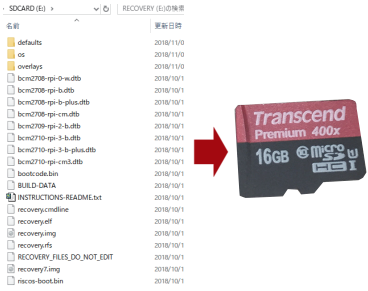


図6　Raspbianのインストールを実行する

Raspbianのインストールと初期設定

最初に、microSD カードを読み書きできるカードリーダーが付いたパソコンや Mac などを使って、ラズパイの標準 OS「Raspbian」を導入するためのメディアを作成します。

Web ブラウザを開いて、「https://www.raspberrypi.org/downloads/noobs/」にアクセスします。開いたページ（図 4）を「NOOBS」の「Download ZIP」をクリックします。最新のインストーラ（2018 年 11 月初旬時点のファイル名は「NOOBS_v2_9_0.zip」）がダウンロードできます。このファイルを展開し、中身のファイルをすべて空の microSD カードにコピーします（図 5）。

ラズパイにセンサーボードと、持っていればラズパイ入門ボードを挿します。キーボード、マウス、ディスプレイも接続して microSD カードを挿し、最後に USB 電源アダプタをつなぎます。これでインストーラが起動します。画面中央にダイアログ、下に言語やキーボードの選択バーが開きます。下のバーにある「言語」で「日本語」を選びます。中央のダイアログで「Raspbian [RECOMMENDED]」にチェックを付けて左上の「インストール」ボタンをクリックします（図 6）。SD カードを上書きする警告ダイアログの「はい」ボタンをクリックすると、Raspbian のインストールが始まります。

しばらく待って、「OS がインストールされました」のダイアログが表示されたら、「はい」ボタンをクリックします（図 7）。再起動後に Raspbian のデスクトップ画面が表示されます。中央に初期設定ウィザードが開きます（図 8）。

インターネットへ接続できるように、デスクトップ画面右上にある赤い二つの × 印が付いたアイコンをクリックし、無線 LAN に接続します。あるいは、ラズパイの LAN 端子にネットワークケーブルを挿します。インターネットにつながったら「Next」ボタンをクリックして初期設定を施します。

センサーボードとラズパイ入門ボードが利用するので、SPI と I²C のシリアル通信を有効にします。まず、左上にあるラズパイのロゴをクリックして、「設定」-「Raspberry Pi の設定」を選びます。「Raspberry Pi の設定」ダイアログの「インターフェイス」タブをクリックし、「SPI」と「I2C」で「有効」を選択して「OK」ボタンをクリックします（図 9）

図7　Raspbianのインストール完了

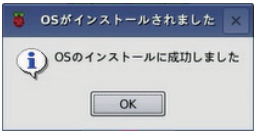


図8　Raspbianのデスクトップ画面　初期設定ウィザードが起動する。

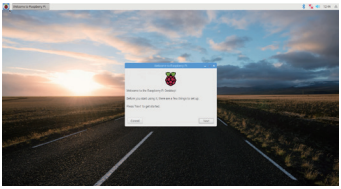


図9　SPIとI2Cを有効にする

各センサーを動かす

センサーボードとラズパイ入門ボードを動かす環境が整いました。サンプルプログラムを利用して、センサーボードに搭載されている湿温度・気圧センサー、ガスセンサー、明るさセンサーを動かします。

各種操作には、コマンドを使います。左上にあるラズパイのロゴをクリックし、「アクセサリ」-「LXTerminal」を選んで端末を開いてください。

次のコマンドで、サンプルプログラムをホームディレクトリにダウンロードします。

```
$ cd
$ git clone https://github.com/shellscript-magazine/
raspi_sensor.git
```

ラズパイ入門ボードの単色有機 EL ディスプレイ（OLED）が使えるように、「Python Package Index」（PyPI）のリポジトリで提供されている、米 Adafruit Industries 社（https://www.adafruit.com/）製「SSD1306 OLD モジュール」用の Python ライブラリを導入します。さらに、日本語の文字を表示するための日本語フォント（ここでは「Takao フォント」）もインストールします。

```
$ sudo apt -y install python3-pip
$ sudo pip3 install Adafruit_SSD1306
$ sudo apt -y install fonts-takao
```

湿温度・気圧センサーを動かす

センサーボードには、独 Bosch Sensortec 社（https://www.bosch-sensortec.com/）製の「BME280」という湿温度・気圧センサーが実装されています（図 10）。BME280 は、I2C または SPI のインタフェースに直接接続できます。センサーボードでは、ラズパイの I2C 端子につないでいます（図 11）。

RPi.BME280モジュールを導入する

BME280 を扱うために Python の「RPi.BME280」モジュール（https://github.com/rm-hull/bme280）を利用します。次のコマンドを実行し、RPi.BME280 モジュールをインストールしてください。

```
$ sudo pip3 install RPi.BME280
```

サンプルプログラムを動かす

サンプルプログラムは、BME280 の湿温度・気圧を取得するメインの「sample.py」（図 12）と、OLED に表示するクラスの「EbOled.py」（p.12 の図 13）に分かれています。

sample.py の 28 行目で、RPi.BME280 モジュールを使って湿温度・気圧の数値を取得しています。29〜31 行目で、取得した湿温度・気圧を表示する形式に加工しています。それらを、32 行目で「EbOled」クラスを使って OLED に表示しています。以上の処理を 27 行目の「while True:」と、34 行目の「time.sleep(1)」により 1 秒間隔で繰り返しています。

それでは、「raspi_sensor/BME280」ディレクトリ内にある実際のサンプルプログラムを動かしてみましょう。次のコマンドを実行します＊2。

```
$ cd ~/raspi_sensor/BME280
$ python3 sample.py
```

＊2　ラズパイ入門ボードがない場合は「sample.py」の代わりに「sampletest.py」を使用してください。

図10　湿温度・気圧センサー「BME280」

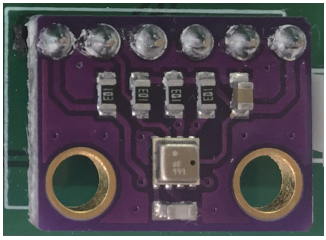


図11　湿温度・気圧センサーの回路図

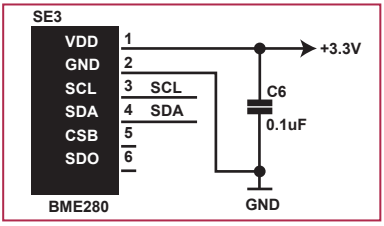


図12　BME280の湿温度・気圧を取得するメインプログラム（sample.py）

```
#!/usr/bin/env python3
#
# apt install python3-pip
# sudo pip3 install RPi.BME280
#

import time
import smbus2
import bme280

from EbOled import EbOled

BME280_ADDR = 0x76
BUS_NO = 1

# BME280
i2c = smbus2.SMBus(BUS_NO)
bme280.load_calibration_params(i2c, BME280_ADDR)

# OLEDパネル
oled = EbOled()
oled.begin()
oled.clear()
oled.display()

try:
  while True:
    data = bme280.sample(i2c, BME280_ADDR)
    oled.drawString('気温 :' +
str(round(data.temperature,1)) + '℃', 0)
    oled.drawString('湿度 :' +
str(round(data.humidity,1)) + '%', 1)
    oled.drawString('気圧 :' +
str(round(data.pressure,1)) + 'hPa', 2)
    oled.display()

    time.sleep(1)
except KeyboardInterrupt:
  pass
```

図13　OLEDに文字を表示するクラスファイル（EbOled.py）

```
import time
import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont


class EbOled(Adafruit_SSD1306.SSD1306_128_64):

  WIDTH = 128
  HEIGHT = 64

  __RST = 24
  __DC = 23
  SPI_PORT = 0
  SPI_DEVICE = 0

  DEFAULT_FONT =
'/usr/share/fonts/truetype/fonts-japanese-gothic.ttf'
  FONT_SIZE = 14
  _LINE_HEIGHT = 16

  def __init__(self):
    self.__spi = SPI.SpiDev(self.SPI_PORT,
self.SPI_DEVICE, max_speed_hz=8000000)
    super().__init__(rst=self.__RST, dc=self.__DC,
spi=self.__spi)
    self._image = Image.new('1', (self.WIDTH,
self.HEIGHT) ,0)
    self._draw = ImageDraw.Draw(self._image)
    self._font = ImageFont.truetype(self.DEFAULT_FONT,
self.FONT_SIZE, encoding='unic')

  def image(self, image):
    self._image = image
    super().image(self._image)
```

図 14 のように OLED 上に「気温」「湿度」「気圧」が表示されます＊3。BME280 の部分を軽く触ってみると、気温や湿度が上がることを確認できます。プログラムを停止するには、端末上で［Ctrl］キーを押しながら［C］キーを押してください。

ガスセンサーを動かす

センサーボードには、中国 Shenzhen Dovelet Sensors Technology 社製の「TP-401T」というガスセンサーが実装されています＊4（図 15）。このセンサーはアナログ出力であるため、ラズパイに直接接続できません。

よって、センサーボードでは 4 チャネルの AD コンバータ「MCP3424」を使って I2C で接続するようにしています（図 16）。I2C には、複数のデバイスを接続できます。I2C に接続したそれぞれのデバイスには、異なる I2C アドレスが割り当てられていて、それを使って個々のデバイスへのアクセスを識別します。

＊3　エラーになるようなら「sudo pip3 install smbus2」で「smbus2」モジュールをインストールしてください。
＊4　TP-401Tのデータシートは「http://www.szdovelet.com/Private/Files/63566350095433649839361729 5.pdf」です。

図14　ラズパイ入門ボードのOLEDに湿温度・気圧を表示

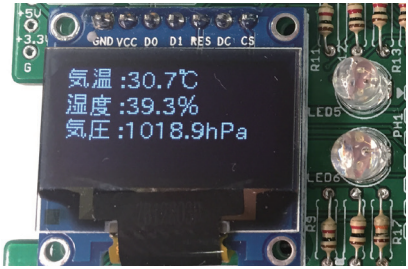


図15　ガスセンサー「TP-401T」

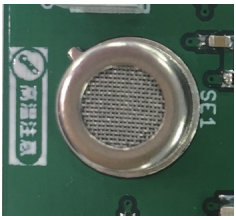
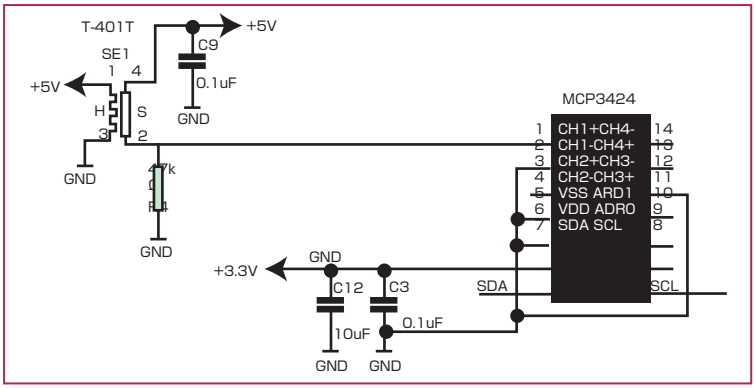


図16　ガスセンサーの回路



カーネルドライバの読み込み

サンプルプログラムでは、MCP3424 からの出力値をカーネルドライバを通じて読み込みます。サンプルプログラムを実行する前に、次のコマンドでカーネルドライバをロードします。

```
$ sudo sh -c "echo 'mcp3424 0x68' > /sys/bus/i2c/devices/i2c-1/new_device"
```

MCP3424 は最大 18 ビットで値を表現する精度を備えていますが、ラズパイではせいぜい 16 ビットの精度までが限界です。それより精度を上げてもノイズのために有意なデータは拾えません。カーネルドライバのデフォルトは 12 ビットの精度ですが、サンプリングレートを変えることで精度を変更できます＊5。

次のように、カーネルの仮想ファイルシステム（sysfs）内の「/sys/bus/i2c/devices/1-0068/iio\:device0/in_voltage_sampling_frequency」にサンプリングレートを書き込むと精度を切り替えられます。

```
$ sudo sh -c "echo 15 > /sys/bus/i2c/devices/1-0068/iio\:device0/in_voltage_sampling_frequency"
```

これで 15 サンプリング / 秒に設定されて精度が 16 ビットになります。この設定を常用するには、「/etc/rc.local」に次の 2 行を追加しておくとよいでしょう。

```
echo 'mcp3424 0x68' > /sys/bus/i2c/devices/i2c-1/new_device
echo 15 > /sys/bus/i2c/devices/1-0068/iio\:device0/in_voltage_sampling_frequency
```

図17　MCP3424の出力値からOLEDに正常・異常を表示したりブザーを鳴らしたりするメインプログラム（sample.py）

```
#!/usr/bin/env python3
import time
import RPi.GPIO as GPIO

from EbOled import EbOled
from TP401T import TP401T

BUZZER = 18

sensor = TP401T()
oled = EbOled()
oled.begin()
oled.clear()
oled.display()

# BUZZER
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
bz = GPIO.PWM(BUZZER, 1000)
bz.stop()

try:
  sensor.start()
  oled.drawString('待機中です')
  oled.display()
  while sensor.state == TP401T.WAITING:  # 測定開始待ち
    time.sleep(3)

  while True:
    if sensor.state == TP401T.NORMAL:
      oled.drawString('空気は正常です')
    else:
      oled.drawString('汚染されています!!')

    if sensor.state == TP401T.ALERT:
      bz.start(50)
    else:
      bz.stop()

    oled.display()
    time.sleep(3)

except KeyboardInterrupt:
  sensor.stop()  # センサー停止

GPIO.cleanup()
```

＊6　ラズパイ入門ボードがない場合は「sample.py」の代わりに「sampletest.py」を使用してください。「adctest.py」は、ラズパイ入門ボードがない場合の、MCP3424の出力値を取得するクラスファイルです。

図18　ガスセンサーを制御するクラスファイル（TP401T.py）

```
import threading
import time
from MCP3424 import MCP3424

class TP401T(MCP3424):

  WAITING = -1 # 測定開始待ち
  NORMAL  = 0  # 空気に問題はない
  ALERT   = 1  # 汚染されている
  WARNING = 2  # 汚染されているが減少中

  term = False
  __current_state = -1

  def __init__(self, ch = 0):
    super().__init__()
    self.tp401_ch = ch
    self.term = False
    self.prev_value = self.getVoltage(self.tp401_ch)
    self.normal_value = self.prev_value

  def start(self):
    self.worker = threading.Thread(target=self.__measure)
    self.term = False
    self.worker.start()

  def stop(self):
    self.term = True
    self.worker.join()

  def __sleep(self, sec):
    for i in range(sec * 100):
      if self.term == True:
        return False
      time.sleep(1/100)

    return True

  def __measure(self):
    self.__current_state = self.WAITING

    sum = 0.0
    for i in range(10):
      sum += self.getVoltage(self.tp401_ch)
      if self.__sleep(3) == False:
        return
    # 30秒間の平均値を平時の値として採用する
    self.normal_value = sum / 10
    self.prev_value = self.normal_value
    self.__current_state = self.NORMAL

    while self.__sleep(3):
      value = self.getVoltage(self.tp401_ch)
      if value < self.normal_value * 1.5:
        self.__current_state = self.NORMAL
      else:
        if (self.prev_value - value) < 0:  # 汚染が増加中
          self.__current_state = self.ALERT
        else:
          self.__current_state = self.WARNING

      self.prev_value = value

  @property
  def state(self):
    return self.__current_state

  def __del__(self):
    self.term = True
```

サンプルプログラムを動かす

サンプルプログラムは、MCP3424 の出力値から OLED に正常・異常を表示したりブザーを鳴らしたりするメインの「sample.py」（図 17）、ガスセンサーを制御するクラスの「TP401T.py」（図 18）、MCP3424 の出力値を取得するクラスの「MCP3424.py」（図 19）、OLED に表示するクラスの「EbOled.py」で構成されます＊6。EbOled.py の中身は、湿温度・気圧センサーで用いたものと同じです。

MCP3424 の出力値（電圧）は、「/sys/bus/i2c/devices/1-0068/iio\:device0/in_voltageN_raw」ファイル（「N」はチャンネル番号）と「/sys/bus/i2c/devices/1-0068/iio\:device0/in_voltageN_scale」ファイル（N はチャンネル番号）の値をかけ合わせると算出できます。MCP3424.py の「MCP3424」がカーネルドライバを通して MCP3424 の出力値を取り込むクラスです。例えば、

```
adc = MCP3424()
```

のように記述すると、「adc.ch0」「adc.ch1」「adc.ch2」プロパティから各チャンネルの電圧値を取得できます。TP401T.py の「TP401T」が、MCP3424 のチャンネル「0」につながっている TP-401T を扱うクラスです。

待機中にカーボン（空気の汚れ）がないときには、おおむね 0.4～0.6V 程度の電圧がチャンネル「0」で測定されます。空気の状態や個体差にもよるので、TP401T クラスでは 30 秒の間、10 回チャンネル「0」を測定して、その平均値を、大気が正常な状態の値に設定しています。

よって、サンプルプログラムを実行するときには大気が正常な状態である必要があります。

```
sensor = TP401()
sensor.start()  ← これで測定が開始
```

測定を開始した後は「sensor.state」プロパティを読むことで大気の状態を取り出すことができます。

「sensor.state」が「TP401T.WAITING」のときは、まだ平均値集計中で測定ができないことを意味します。「TP401T.NORMAL」のときは大気の状態が正常、「TP401.ALART」のときは汚染状態、「TP401.WARNING」のときは汚染されているが汚染度が低下しつつある状態となります。TP401 クラスでは、3 秒おきにチャネル「0」の出力値を参照し、起動直後の 30 秒間の平均より出力値が 1.5 倍を超えたら汚染されていると判断しています。また、3 秒前の値と比較して低下しているなら TP401.WARNING と判断します。

それでは、次のようにサンプルプログラムを実行してみましょう。

```
$ cd ~/raspi_sensor/TP-401T
$ python3 sample.py
```

を実行すると 30 秒待機した後に、測定を始めます（図 20）。空気が汚染されると、OLED に通知を表示し、ブザーが鳴ります。ガスセンサーのテストには「エアダスター」を使うと便利です。エアダスターを吹くと汚染されていると判断してブザーが鳴ります。

サンプルプログラムを終了するには、端末上で［Ctrl］キーを押しながら［C］キーを押してください。

明るさセンサーを動かす

センサーボードには、米 Vishay Intertechnology 社製の「VCNL4020」という明るさセンサーが実装されています＊7(図21)。VCNL4020 は、I2C のインタフェースに直接接続できます。センサーボードでは、BME280 や MCP3424 と同様に、ラズパイの I2C 端子につないでいます(図 22)。VCNL4020 には割り込み端子(INT)があり、ジャンパピン(JP1)を経由してラズパイの汎用 I/O(GPIO)の「6」番に接続しています。

サンプルプログラムを動かす

サンプルプログラムは、明るさの値を OLED に表示したりブザーを鳴らしたりするメインの「sample.py」(図 23)、明るさセンサーから値を取得するクラスの「VCNL4020.py」(P.18 の図24)、OLED に表示するクラスの「EbOled.py」で構成されます。EbOled.py の中身は、湿温度・気圧センサーやガスセンサーで用いたものと同じです。

VCNL4020.py に明るさセンサーの値を取り出す「VCNL4020」クラスを実装しました。このクラスでは、ライブラリやドライバを使っていません＊8。

```
sensor = VCNL4020()
```

としたら、以降は「sensor.luminance」プロパティで明るさの値(LUX 値)、「sensor.proximity」プロパティから近接センサーの値を読み取ることができます。

それでは、サンプルプログラムを実行しましょう＊9。

```
$ cd ~/raspi_sensor/VCNL4020
$ python3 sample.py
```

OLED に現在の明るさを表示します＊10(図 25)。また、手や物をセンサーに接近させるとブザーが鳴ります。サンプルプログラムを終了するには、端末上で[Ctrl]キーを押しながら[C]キーを押してください。

図19　MCP3424の出力値を取得するクラスファイル(MCP3424.py)

```
import os

class MCP3424():

  SYSFS_PATH = '/sys/bus/i2c/devices/i2c-1/1-0068'
  SYSFS_IIO  = '/iio:device0/'
  __enable = False

  def __init__(self):
    if not os.path.exists(self.SYSFS_PATH):
      os.system('sudo /bin/bash -c "echo \'mcp3424 0x68\' > /sys/bus/i2c/devices/i2c-1/new_device"')

    if not os.path.exists(self.SYSFS_PATH):
      raise Exception('sysfs error')

    self.__enable = True

  def getVoltage(self, ch):
    if not self.__enable:
      return 0

    raw = 0.0
    scale = 0.0

    with open(self.SYSFS_PATH+self.SYSFS_IIO+'in_voltage{}_raw'.format(ch), "r") as f:
      raw = float(f.read())
    with open(self.SYSFS_PATH+self.SYSFS_IIO+'in_voltage{}_scale'.format(ch), "r") as f:
      scale = float(f.read())

    return (raw * scale)

  @property
  def ch0(self):
    return self.getVoltage(0)

  @property
  def ch1(self):
    return self.getVoltage(1)

  @property
  def ch2(self):
    return self.getVoltage(2)

  @property
  def ch3(self):
    return self.getVoltage(3)
```

図20　ラズパイ入門ボードのOLEDに空気の状態を表示

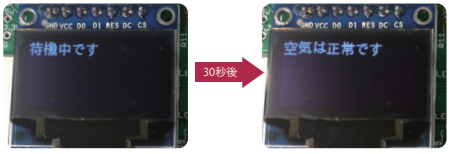


図21　明るさセンサー「VCNL4020」

図22　明るさセンサーの回路

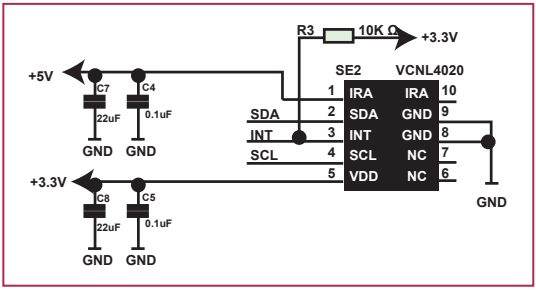
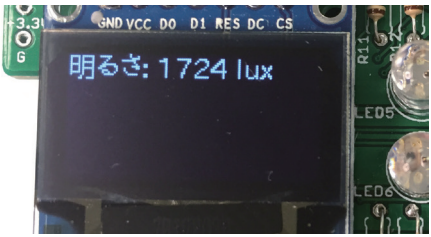


図23　明るさの値をOLEDに表示したりブザーを鳴らしたりするメインプログラム(sample.py)

```
#!/usr/bin/env python3
import time
import RPi.GPIO as GPIO
import threading

from EbOled import EbOled
from  VCNL4020 import  VCNL4020

BUZZER = 18

term = False
sensor = VCNL4020()
oled = EbOled()
oled.begin()
oled.clear()
oled.display()

def prox():
  global term

  GPIO.setmode(GPIO.BCM)
  GPIO.setup(BUZZER, GPIO.OUT)
  bz = GPIO.PWM(BUZZER, 1000)

  while term == False:
    if sensor.proximity > 4000:
      bz.start(50)
    else:
      bz.stop()
    time.sleep(0.1)

  GPIO.cleanup(BUZZER)


t = threading.Thread(target=prox)
t.start()

try:
  while True:
    oled.drawString('明るさ: ' + str(sensor.luminance) +' lux')
    oled.display()
    time.sleep(0.2)
except KeyboardInterrupt:
  term = True
```

*7　VCNL4020のデータシートは「https://www.vishay.com/docs/83476/vcnl4020.pdf」。

図25　ラズパイ入門ボードのOLEDに明るさを表示



*8　今回は割り込みを使っていません。連載「ラズパイ入門ボードで学ぶ電子回路の制御」で割り込みを使う方法を解説したいと考えています。
*9　ラズパイ入門ボードがない場合は「sample.py」の代わりに「sampletest.py」を使用してください。
*10　表示している結果は「Lux」(ルクス)ではなく、Ambient Light Signal(cts)値かもしれません。

図24　明るさセンサーから値を取得するクラスファイル(VCNL4020.py)

```
# https://www.vishay.com/docs/83476/vcnl4020.pdf

import smbus
import time
from threading import BoundedSemaphore

class VCNL4020():

  _ALS_OD       = 0b00010000  # オンデマンド明るさ計測スタート
  _PROX_OD      = 0b00001000  # オンデマンド近接計測スタート
  _ALS_EN       = 0b00000100  # 明るさ繰り返し計測有効
  _PROX_EN      = 0b00000010  # 近接繰り返し計測有効
  _SELFTIMED_EN = 0b00000001  # 内蔵タイマー有効

  _CONT_CONV    = 0b10000000  # Continue Conversion有効
  _AMBIENT_RATE = 0b00010000  # 明るさの計測レート(default:2sample/s)
  _AUTO_OFFSET  = 0b00001000  # 自動オフセットモード有効
  _AVERAGING    = 0b00000101  # 平均化(default:32conv)

  _COMMAND_REG       = 0x80  # コマンドレジスタ
  _PID_REG           = 0x81  # プロダクトIDレジスタ
  _PROX_RATE_REG     = 0x82  # 近接測定レートジスタ
  _IR_CURRENT_REG    = 0x83  # 近接測定用赤外線LED電流設定レジスタ(default=20mA)
  _AMBIENT_PARAM_REG = 0x84  # 明るさセンサーパラメータレジスタ

  _AMBIENT_MSB       = 0x85  # 明るさ上位バイト
  _AMBIENT_LSB       = 0x86  # 明るさ下位バイト

  _PROX_MSB          = 0x87  # 近接上位バイト
  _PROX_LSB          = 0x88  # 近接下位バイト

  def __init__(self, i2c_addr = 0x13, busno = 1):
    self.addr = i2c_addr
    self.i2c = smbus.SMBus(busno)

    self._write_reg(self._COMMAND_REG, self._ALS_OD  |\
                    self._PROX_OD |\
                    self._ALS_EN  |\
                    self._PROX_EN |\
                    self._SELFTIMED_EN )

    self._write_reg(self._IR_CURRENT_REG, 2 )  # 20mA

    self._write_reg(self._AMBIENT_PARAM_REG, self._CONT_CONV    |\
                    self._AMBIENT_RATE |\
                    self._AUTO_OFFSET  |\
                    self._AVERAGING )
    self.semaphore = BoundedSemaphore()
    time.sleep(0.6)      # 初回測定まで待つ

  def _write_reg(self, reg, value):
    self.i2c.write_byte_data(self.addr, reg, value)

  @property
  def luminance(self):
    self.semaphore.acquire()
    d = self.i2c.read_i2c_block_data(self.addr, self._AMBIENT_MSB, 2)

    self.semaphore.release()
    return (d[0] * 256 + d[1])

  @property
  def proximity(self):
    self.semaphore.acquire()
    d = self.i2c.read_i2c_block_data(self.addr, self._PROX_MSB, 2)
    self.semaphore.release()
    return (d[0] * 256 + d[1])
```

かながわ経済新聞
KANAKEI

ディープな製品情報満載
マニアのための情報紙。

編集長　千葉龍太

ご購読はこちらから⇒⇒⇒

Office365・G Suiteとシームレスに連携
クラウド型ビジネスメールセキュリティサービス

TRIHIKO_AJ

形骸化したメールセキュリティを一新。
特定の機密情報を検知し、
メールの誤送信による情報漏洩を防ぐ。

メール誤送信による情報漏洩を自動で防ぐ！

送信済みメール取消機能

機密情報を暗号化送信

64%

の人がメール誤送信の経験があると回答しています。

宛先間違い・添付ファイル間違いなどの誤送信。

取り消せれば・・・

と思った経験ありませんか？

TRIHIKO_AJの特徴

01

オペレーション変更不要

普段お使いのOffile365・G Suite
からいつも通りメールを送信するだけ。
インストールや初期設定等面倒な作業は必要
ありません

02

自動検知・自動暗号化

本文・件名・添付ファイル内に
機密情報などの重要なワードが含まれていない
かをチェックし検知されたメールは暗号化送
信。TRIHIKO_AJが自動でサポート。
個人のリテラシーに依存しません。

03

送信済みメール取消し

送信後のメールも自在にコントロール可能。
送信済みメール取消機能で万が一の誤送信の際
も安心。閲覧期限を設定することでセキュアな
情報を必要な時にだけ共有。

TRIHIKO

トリヒコ株式会社
東京都江東区青海２－７－４
https://trihiko.com

平日10:00から19:00
03-6428-0737

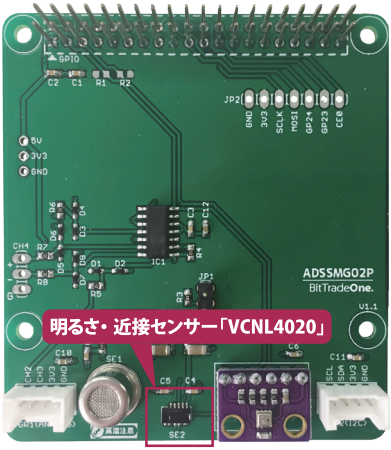
特集2 明るさ・近隣センサーを使って近づく物体を検出しよう！

ガジェット 米田聡

特集2では、このボードを使った電子回路制御を取り上げます。具体的には、明るさ・近隣センサーに近づく物体の検出です。

ラズパイセンサーボードに搭載されている米 Vishay Intertechnology 社製の明るさ・近隣センサー「VCNL4020」(図1)は、プログラム可能な「割り込み」機能を持っています。前号(Vol.57)の特集1に掲載したVCNL4020 ライブラリ(VCNL4020.py)では、この割り込み機能を使っていませんでした。割り込みを使うことで近接センサー機能などを効率的に活用できます。

図1　ラズパイセンサーボードの明るさ・近接センサー



VCNL4020の割り込み機能

VCNL4020 のマニュアルは、Vishay Intertechnology 社の公式サイトからダウンロードできます。「https://www.vishay.com/docs/83476/vcnl4020.pdf」と「https://www.vishay.com/docs/84361/designingvcnl4100.pdf」の2冊に分かれています。VCNL4020 の扱い方はこれらのマニュアルを見れば分かるのですが、それだけでは不親切なので割り込み機能について補足説明を加えておきます。

VCNL4020 の割り込みピンは、大きく分けて2種類の割り込み要因によりアクティブになります。

一つは、近接センサーまたは明るさセンサーの値が「しきい値レジスタ」の値をまたいだときです。しきい値レジスタには「低値用」(Low threshold)と「高値用」(High threshold)の二つがあります。

例えば、近接センサーの値が低い方から高い方に変化し、高値用レジスタの値を超えたら割り込みがアクティブになります。近接センサーの値は、センサーに物体が近づくと低い方から高い方に変化します。よって、高値用レジスタの値を適切に設定することで、物体が近づいたら割り込みをアクティブすることが可能になるわけです。

低値用レジスタの方は、明るさセンサー向きです。例えば、周囲が暗くなり明るさセンサーの値が低値用レジスタをまたいで小さくなったら、割り込みをアクティブすることができます。

もう一つは、明るさセンサーや近接センサーに新しい値が準備できたときです。VCNL4020 は一定の周期でセンサーの値を更新しています。新しい値の準備ができたら割り込みがアクティブになる機能を使うことで、新しい値を割り込み関数で受け取ることができます。

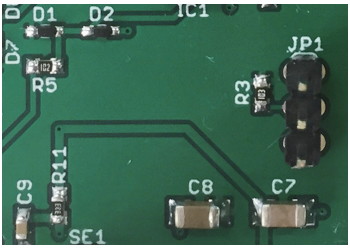
なお、「Raspberry Pi」(ラズパイ)のようにマルチスレッド処理が可能な Linux を搭載できる高性能なコンピュータボードなら、割り込みを使用せずにセンサーの値を短い間隔のポーリングで読み込むことができます。そのため新しい値の準備ができたら割り込みをアクティブにする機能はあまりありがたみがないかもしれません。この機能は、どちらかというと非力なマイコンボード向きといえます。

センサーボードの割り込みの使い方と注意点

ラズパイセンサーボードでは VCNL4020 の割り込みピンがボード上のジャンパスイッチ「JP1」の「1」番ピンと「2」番ピンを介してラズパイの「GPIO6」に接続されています。VCNL4020 の割り込みを使いたいときには、あらかじめ JP1 の1番ピンと2番ピンをショートさせておく必要があります(図2)。付属のジャンパピンを取り外していなければ、1番ピンと2番ピンに挿さっているので問題ありません。

なお、ラズパイセンサーボードと、シェルスクリプトマガジン製のオリジナル拡張ボード「ラズパイ入門ボード」を併用している場合、GPIO6 がラズパイ入門ボードのスイッチで使用されている点に注意してください。VCNL4020 の割り込みを使用するときにはラズパイ入門ボード上のスイッチ「SW4」が使えなくなります。

図2　ジャンパピンで1番ピンと2番ピンをショート



割り込みに対応したVCNL4020ライブラリ

前回の特集 1 に掲載した VCNL4020.py に若干の改変を加えて割り込み対応にしたコードを図 3 に示しました。

VCNL4020 の割り込みピンは通常「HIGH」で、アクティブになると「LOW」になります。

この VCNL4020 ライブラリでは、近接センサーまたは明るさセンサーの値がしきい値レジスタの値をまたいだときに割り込みがアクティブになる機能を利用しています。高値用レジスタのしきい値を書き込む関数が「set_high_threshold()」、低値用レジスタのしきい値を書き込む関数が「set_low_threshol
d()」です。

割り込みを有効化する関数は「enable_interrupt()」で、三つの引数を取ります。

最初の引数「callbackfunc」は、割り込みがアクティブになると呼び出されるコールバック関数です。二つの引数を取る関数を設定します。

二つ目の引数「prox」には、しきい値をまたぐ種類を指定します。「True」を指定した場合、近接センサーの値がしきい値をまたぐと割り込みがアクティブになります。「False」を指定すると明るさセンサーの値がしきい値をまたいだときに割り込みがアクティブになります。

三つ目の引数「samples」には割り込みがアクティブになる値の測定回数を設定します。デフォルトは 1 回で、しきい値をまたぐ値が 1 回、測定されれば割り込みが発生します。回数は「1」～「128」回を 3 ビットで指定できます。指定方法は、VCNL4020 の「INTERRU
PT CONTROL REGISTER」の設定値をマニュアルで確認するとよいでしょう。なお、通常はデフォルトの 1 回で問題ありません。

物体が近づいたら割り込みを発生させる

割り込みを使用するサンプルプログラム(simpletest.py)を図 4 に示しました。プログラミングをする上で注意しなくてはならないのは、VCNL4020 の割り込み状態は「割り込み制御レジスタ」(INTERRUPT CONTROL REGISTER)を設定するまでクリアーされない点です。割り込み状態をクリアーしない限り、割り込みはアクティブのままなので、コールバック関数が 1 回しか呼び出されないことになります。VCNL4020 ライブラリを使う場合は、コールバック関数内で enable_interrupt() メソッドを呼び出して割り込み状態をクリアーします。

それでは、次のように simpletest.py に実行権限を与えて、実行してみましょう。

```
$ chmod +x simpletest.py
$ ./simpletest.py
```

VCNL4020 に手などを近づけると、図 5 のように表示されます＊1。これで明るさ・近接センサーに何かが近づいたことが分かります。

＊1　明るさ・近接センサーの横にあるにおいセンサーが高温になっていることに注意してください。不用意に触ってしまうと、やけどする可能性があります。

図3　割り込みに対応したVCNL4020ライブラリ(VCNL4020.py)

```
import smbus
import time
import RPi.GPIO as GPIO
from threading import BoundedSemaphore

class VCNL4020():

  _ALS_OD       = 0b00010000  # オンデマンド明るさ計測スタート
  _PROX_OD      = 0b00001000  # オンデマンド近接計測スタート
  _ALS_EN       = 0b00000100  # 明るさ繰り返し計測有効
  _PROX_EN      = 0b00000010  # 近接繰り返し計測有効
  _SELFTIMED_EN = 0b00000001  # 内蔵タイマー有効

  _CONT_CONV    = 0b10000000  # Continue Conversion 有効
  _AMBIENT_RATE = 0b00010000  # 明るさの計測レート
(default:2sample/s)
  _AUTO_OFFSET  = 0b00001000  # 自動オフセットモード有効
  _AVERAGING    = 0b00000101  # 平均化 (default:32conv)

  _COMMAND_REG       = 0x80  # コマンドレジスタ
  _PID_REG           = 0x81  # プロダクト ID レジスタ
  _PROX_RATE_REG     = 0x82  # 近接測定レートジスタ
  _IR_CURRENT_REG    = 0x83  # 近接測定用赤外線 LED 電流設定レジスタ (default=20mA)
  _AMBIENT_PARAM_REG = 0x84  # 明るさセンサーパラメータレジスタ

  _AMBIENT_MSB       = 0x85  # 明るさ上位バイト
  _AMBIENT_LSB       = 0x86  # 明るさ下位バイト

  _PROX_MSB          = 0x87  # 近接上位バイト
  _PROX_LSB          = 0x88  # 近接下位バイト

  _INT_CONTROL_REG   = 0x89  # 割り込み制御レジスタ

  _LOW_TH_MSB        = 0x8A  # Low しきい値 (MSB)
  _LOW_TH_LSB        = 0x8B  # Low しきい値 (LSB)
  _HIGH_TH_MSB       = 0x8C  # High しきい値 (MSB)
  _HIGH_TH_LSB       = 0x8D  # High しきい値 (LSB)

  _INT_STATUS_REG    = 0x8E  # 割り込みステータス

  _INT_NO            = 0x06  # int = GPIO6

  # コールバック
  __callbackfunc     = None

  def __init__(self, i2c_addr = 0x13, busno = 1):
    self.addr = i2c_addr
    self.i2c = smbus.SMBus(busno)

    self._write_reg(self._COMMAND_REG, self._ALS_OD  |\
                    self._PROX_OD |\
                    self._ALS_EN  |\
                    self._PROX_EN |\
                    self._SELFTIMED_EN )

    self._write_reg(self._IR_CURRENT_REG, 2 )  # 20mA

    self._write_reg(self._AMBIENT_PARAM_REG, self._CONT_CONV
|\
                     self._AMBIENT_RATE |\
                     self._AUTO_OFFSET  |\
                     self._AVERAGING )
    self.semaphore = BoundedSemaphore()

    # GPIO 設定
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(self._INT_NO, GPIO.IN,
pull_up_down=GPIO.PUD_UP)
    GPIO.add_event_detect(self._INT_NO, GPIO.FALLING,
callback=self.__interruptfunc)

    time.sleep(0.6)      # 初回測定まで待つ
```

```
  def _write_reg(self, reg, value):
    self.i2c.write_byte_data(self.addr, reg, value)

  def _read_reg(self, reg):
    return self.i2c.read_byte_data(self.addr, reg)

  # 高値用レジスタ設定
  def set_high_threshold(self, value):
    self.semaphore.acquire()
    h = (value & 0xFF00) >> 8
    l = value & 0x00FF
    self._write_reg(self._HIGH_TH_MSB, h)
    self._write_reg(self._HIGH_TH_LSB, l)
    self.semaphore.release()

  # 低値用レジスタ設定
  def set_low_threshold(self, value):
    self.semaphore.acquire()
    h = (value & 0xFF00) >> 8
    l = value & 0x00FF
    self._write_reg(self._LOW_TH_MSB, h)
    self._write_reg(self._LOW_TH_LSB, l)
    self.semaphore.release()

  # 割り込み有効化
  def enable_interrupt(self, callbackfunc=None, prox=True,
samples=1):
    self.semaphore.acquire()

    self.__callbackfunc = callbackfunc
    value = self._read_reg(self._INT_CONTROL_REG)

    if callbackfunc is not None:
      if prox:
        value |= 0b00000010
      else:
        value |= 0b00000011
    else:
        value &= 0b11111100

    # samples
    samples &= 0b00000111
    samples = samples << 5
    value &= 0b00011111
    value |= samples

    self._write_reg( self._INT_CONTROL_REG, value)
    self.semaphore.release()

  # 割り込み関数
  def __interruptfunc(self, ch):
    if ch != self._INT_NO:
      return
    if self.__callbackfunc is not None:
      self.__callbackfunc(self.luminance, self.proximity)


  @property
  def luminance(self):
    self.semaphore.acquire()
    d = self.i2c.read_i2c_block_data(self.addr,
self._AMBIENT_MSB, 2)
    self.semaphore.release()
    return (d[0] * 256 + d[1]) / 4

  @property
  def proximity(self):
    self.semaphore.acquire()
    d = self.i2c.read_i2c_block_data(self.addr,
self._PROX_MSB, 2)
    self.semaphore.release()
    return (d[0] * 256 + d[1])
```

図4　物体が近づいたら割り込みを発生させるサンプルプログラム(simpletest.py)

```
#!/usr/bin/env python3
import time
from  VCNL4020 import  VCNL4020

sensor = VCNL4020()

# コールバック関数
def callback(lux, prox):
  print(' センサーに何かが接近しています ')
  print(' 現在の明るさ :'+str(lux) )
  print(' 近接センサー :'+str(prox) )
  # 割り込み再設定
  sensor.enable_interrupt(callback)

# しきい値高を設定
sensor.set_high_threshold(2500)
# しきい値低を設定
sensor.set_low_threshold(0)
# 割り込み有効化
sensor.enable_interrupt(callback)

try:
  while True:
    time.sleep(1)

except KeyboardInterrupt:
  pass
```

図5　サンプルプログラムの実行例

```
yoneda@raspberrypi:~/sensorboard/VCNL4020_v2 $ ./simpletest.py
センサーに何かが接近しています
現在の明るさ :126.25
近接センサー :2720
センサーに何かが接近しています
現在の明るさ :126.5
近接センサー :2793
センサーに何かが接近しています
現在の明るさ :126.0
近接センサー :2968
センサーに何かが接近しています
現在の明るさ :127.5
近接センサー :2615
```

特集3 アナログ入力電圧を測定してみよう！

ガジェット 米田聡

この回では、外部の電子回路を接続するための Grove コネクタが備えるアナログ入力を試します。

ラズパイセンサボードには２個の Grove コネクタが搭載されています（図 1）。「GR1」がアナログ入力用、「GR2」が I2C デバイス接続用です。今回は二つのコネクタのうち、GR1 のアナログ入力を試します。

ドライバがある周辺LSIならそれを使おう

ラズパイセンサーボードには、I2C 接続の高精度４チャンネル ADC（Analog to Digital Converter）「MCP3424」が搭載されています（図 2）。MCP3424 のチャンネル「0」はオンボードのガス（臭い）センサー「TP-401T」に接続されていて、チャンネル「1」とチャンネル「2」が GR1 に接続されています。

MCP3424 のような周辺 LSI（ペリフェラル）を制御する場合、自力で制御するスクリプトを作成する方法と、既存のライブラリやドライバを探して利用する方法の２通りが考えられます。シェルスクリプトマガジン Vol.57 の特集 1 では後者を選択しました。

どちらが良いとは一概にいえませんが、既存のライブラリやドライバをまず探して、存在しないかを確認します。存在しない、あるいは既存のライブラリやドライバが提供する機能が自分の用途にそぐわない場合に限り自作のスクリプトで周辺 LSI を制御した方が効率は良いでしょう。自作のスクリプトで周辺 LSI を制御すれば自分の思い通りの機能を実装できる利点がありますが、場合によっては車輪の再発明になりかねないからです。

MCP3424 に対しては、Linux カーネルに「Linux IIO」（Industrial I/O Subsystem）ベースのドライバが同こんされています。Linux IIO は ADC や「DAC」（Digital to Analog Converter）と、各種センサー類を扱うための共通基盤で、ADC や DAC、センサーを Raspberry Pi で使うときはまず、標準 OS「Raspbian」内に Linux IIO のドライバがないかを調べてみるとよいでしょう。

Raspbian にインストール済みのドライバは「/lib/modules/ カーネルバージョン /kernel/drivers/iio」ディレクトリ以下を調べることで分かります（図 3）。/lib/modules/ カーネルバージョン /kernel/drivers/iio 以下にサブディレクトリに分けてドライバが納められています。「buffer」と「common」のディレクトリの下には Linux IIO で使われる共用の機能を提供するドライバが、「adc」ディレクトリ以下に ADC 用、「humidity」ディレクトリ以下に湿度センサー用、「imu」ディレクトリ以下に慣性センサー用、「light」ディレクトリ以下に光センサー用、「puressure」ディレクトリ以下に圧力センサー用のドライバが格納されています。

2019 年２月末時点の Raspbian では、図３のように adc ディレクトリの下に、SPI 接続の MPC320x 対応のドライバと、I2C 接続の MPC3422 のドライバが格納されていました。この二つのうち、MCP3422 用のドライバはセンサーボードに使われている MCP3424 にも対応しています。

図1　ラズパイセンサーボードに搭載された二つのGroveコネクタ

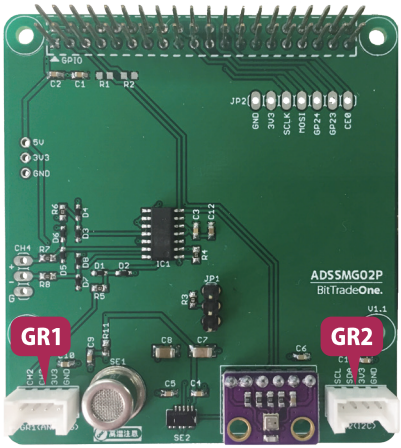


図2　I2C接続の高精度4チャンネルADC「MCP3424」

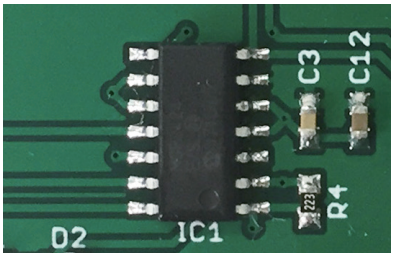


図3　Raspbianにインストール済みのドライバを調べる

```
$ ls -F /lib/modules/`uname -r`/kernel/drivers/iio/
adc/  buffer/  common/  humidity/  imu/  industrialio.ko
light/  pressure/
$ ls -F /lib/modules/`uname -r`/kernel/drivers/iio/adc/
mcp320x.ko  mcp3422.ko
```

I2Cドライバを組み込む

I2C アドレスは７ビットですが、Linux 上では上位１ビットに「0」を追加した８ビットのアドレスを使います。MCP3424 の I2C アドレスは、７ビット中の上位４ビットが固定で８ビットアドレスでは「0b01101xxx」（x は「0」または「1」）になります。末尾３ビットは MCP3424 の「A0」～「A2」ピンで設定が可能ですが、ラズパイセンサーボードでは A0～A2 ピンがすべて GND と接続されています。よって、ラズパイセンサーボードに搭載された MCP3424 の I2C アドレスを８ビットで表現したアドレスは「0b01101000」となり、16 進数で表すと「0x68」となります。

では、実際にラズパイセンサーボードに搭載された MCP3424 のアドレスを確認します。次のように「i2c-tools」パッケージをインストールし、I2C インタフェース上で接続されているアドレスを列挙する「i2cdetect」を実行してみましょう。図４のように「0x68」にデバイスが接続されていることが示されます。

```
$ sudo apt update
$ sudo apt -y install i2c-tools
$ i2cdetect -y 1
```

I2C 対応のドライバは「/sys/bus/i2c/devices/i2c-N/new_device」という仮想ファイルに「デバイス名 I2C アドレス」という文字列を書き込んで組み込みます。N は I2C インタフェースのバス番号で、Raspberry Pi ではユーザーに開放されているバス番号が「1」なので「/sys/bus/i2c/devices/i2c-1/new_device」となります。

よって、次のコマンドで MCP3424 のドライバを組み込むことができます。なお、「new_device」は特殊ファイルであり、管理者(root ユーザー)権限がないと書き込むことができません。

```
sudo bash -c "echo 'mcp3424 0x68' >/sys/bus/i2c/devices/i2c-1/new_device"
```

「カーネルドライバローダー」によって、デバイス名に指定した部分の名称が「/lib/modules/ カーネルバージョン /modules.alias」ファイルから検索され、該当するドライバがロードされる仕組みになっています。

図4　i2cdetectコマンドを実行した結果

```
iyoneda@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- 13 -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
yoneda@raspberrypi:~ $
```

コマンド実行後、i2cdetect コマンドを改めて実行してみると 0x68 の部分が「UU」に変わっていることが確認できます(図 5)。この UU は、その I2C アドレスのデバイスがカーネル内、つまりドライバで使用されているために汎用 I2C ドライバ(/dev/i2c-*)からは利用できないことを示しています。

I2C ドライバの組み込みで注意が必要なのは、I2C デバイスにはそのデバイスの種類を調べる統一的な方法がない点です。USB や PCI(PCI Express)はベンダーID とデバイス ID というデバイス固有の ID があるため、デバイスドライバを組み込むときに正しいデバイスかどうかをドライバ内でチェックして正しくなければ組み込まないという仕組みがありますが、I2C デバイスにはそれがありません。

結果として I2C のドライバは、最小限のデバイスのチェックしかしていないので、例えば、異なるデバイスに MCP3424 ドライバを割り当てるといったこともできてしまう可能性があります。

MCP3424ドライバでGR1コネクタの電圧を調べる

アナログ用 Grove コネクタのピンアサインは図 6 の通りです。1 番ピンと 2 番ピンがアナログ入力、3 番ピンが +3.3V の電源、4 番ピンが ±0V の GND となっています。Grove コネクタの「AN0」が MCP3424 のチャンネル「1」に、「AN1」は同チャンネル「2」に接続されています。

MCP3424 の機能を調べるために図 7 の回路図と図 8 の回路を作成して LED の順方向降下電圧を測ってみます。抵抗「R」は 100Ω前後が標準ですが、適当に変えて調べてみるのもよいでしょう。

利用にあたって注意が必要な点があります。デバイスドライバを利用した MCP3424 の制御「/sys/bus/i2c/devices/1-0068/iio:device0」ディレクトリ以下にある特殊ファイルを通じて実施します。1-0068 の部分は「バス番号 -I2C アドレス」で変化しませんが、iio:device0 の最後にある「0」は Linux IIO の番号で、MCP3424 より先に別の Linux IIO ドライバが組み込まれている場合は「0」以外(「1」など)になる点に注意してください。

このディレクトリの下には図 9 のように「in_voltageN_raw」と「in_voltageN_scale」という特殊ファイルが並んでいます。N の部分は MCP3424 のチャンネル番号で、Grove コネクタの AN0 には「in_voltage1_raw」と「in_voltage1_scale」が該当します。

電圧値は、in_voltage1_raw ファイルと in_voltage1_scale ファイルに書き込まれた値をかけ合せることで算出できます。単に数値のかけ算ができればよいので、シェル上でも「bc」コマンドなどで計算できます。それでは、次のように「bc」パッケージをインストールして bc コマンドを使えるようにしましょう。

```
$ sudo apt -y install bc
```

カレントディレクトリを /sys/bus/i2c/devices/1-0068/iio:device0 に移し、次のように bc コマンドを実行すると電圧値が得られます(図 10)。

```
$ cd /sys/bus/i2c/devices/1-0068/iio:device0
$ echo "`cat in_voltage1_raw` * `cat in_voltage1_scale`" | bc
```

同じディレクトリにある「in_voltage_sampling_frequency」ファイルにサンプリング周波数を書き込むと精度が変わります。サポートされているサンプリング周波数＊1 は表 1 の通りです。デフォルトは「240」、精度は 12 ビットですが、「3」までサンプリング周波数を落とすと 18 ビット精度が得られます。サンプリング周波数を切り替えると、in_voltageN_scale の中身が自動的に精度に応じて変わる仕組みになっています。

ちなみに、手持ちの LED で実際に順方向降下電圧を測ったところ、R が 100Ωのときに約 1.9V、R が 1.2kΩのときに約 1.8V でした。LED に流れている電流は R が 100Ωのとき約 14mA、R が 1.2kΩのとき 1.25mA となります。このように電流を変えても順方向降下電圧がほとんど変わらないのは、電子が PN 接合を乗り越えるエネルギーに起因しているためで、ダイオードを定電圧源として回路内で役立てることができる理由になっています。

＊1　単位は「sps」で、1秒間にアナログ値を読み取る回数です。

図5　ドライバ組み込み後のi2cdetectコマンドを実行した結果

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- 13 -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

図6　アナログ入力用のGroveコネクタ

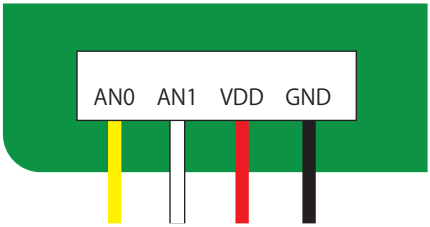


図7　MCP3424の機能を調べる回路図

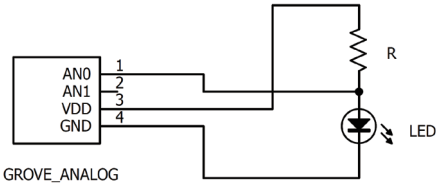


図8　MCP3424の機能を調べる回路
ブレッドボードや、ブレッドボードに挿さるピンが付いたGroveコネクタケーブル、LED、抵抗は千石電商や秋月電子などで購入できる。

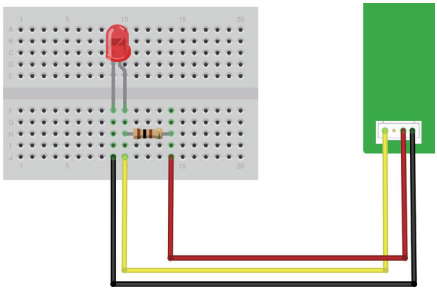


表1　サンプリング周波数と精度

サンプリング周波数（sps）	精度（ビット）
240	12
60	14
15	16
3	18

＊1　明るさ・近接センサーの横にあるにおいセンサーが高温になっていることに注意してください。不用意に触ってしまうと、やけどする可能性があります。

図9　MCP3424ドライバの特殊ファイル

```
yoneda@raspberrypi:/sys/bus/i2c/devices/1-0068/iio:device0 $ ls -F
dev                in_voltage2_scale              power/
in_voltage0_raw    in_voltage3_raw                sampling_frequency_available
in_voltage0_scale  in_voltage3_scale              subsystem@
in_voltage1_raw    in_voltage_sampling_frequency  uevent
in_voltage1_scale  in_voltage_scale_available
in_voltage2_raw    name
```

図10　アナログ入力の電圧値を取得した

```
yoneda@raspberrypi:/sys/bus/i2c/devices/1-0068/iio:device0 $ ls -F
dev                in_voltage2_scale              power/
in_voltage0_raw    in_voltage3_raw                sampling_frequency_available
in_voltage0_scale  in_voltage3_scale              subsystem@
in_voltage1_raw    in_voltage_sampling_frequency  uevent
in_voltage1_scale  in_voltage_scale_available
in_voltage2_raw    name
```

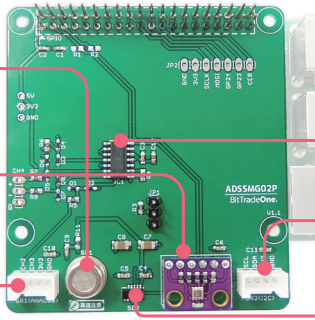
発　行　株式会社 ビット・トレード・ワン
企画構成　株式会社 ビット・トレード・ワン 東京デザイン室
協　力　ユニバーサル・シェル・プログラミング研究所
シェルスクリプトマガジン編集部
連 絡 先　info@bit-trade-one.co.jp
企画構成　Gr@phic
発 行 日　2019/08/19

AdB#04

AssemblyDesk Books Volume.04

冊子内使用ボードのご紹介

ラズパイセンサーボード



- ガスセンサー [T-401T]
- 温湿度・気圧センサー [BME280]
- **Groveコネクタ**
  アナログセンサーの接続が可能。
  ADコンバーターのアナログ端子に接続
- ADコンバーター [MCP3432]
- **Groveコネクタ**
  ラズパイのI^2Cに接続
- 明るさセンサー [VCNL4020]

■ 環境測定の定番３つのセンサーを搭載

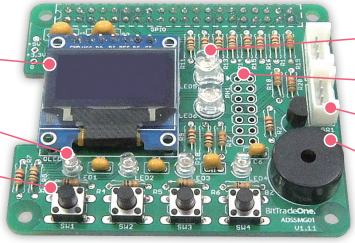
湿温度・気圧センサー、ガスセンサー、明るさセンサーなど、3種類のセンサーを搭載し様々な状況を測定可能です。
またアンモニア、水素、アルコール、CO、メタンなど有機揮発性気体を測定可能なガスセンサー用に、
高精度デルタシグマA／Dコンバータを実装しています。

▼ スタッキングで使用可能 ▲

ラズベリー・パイ 対応

ラズパイ入門ボード

ラズパイで制御方法を学ぶ基礎となる電子部品を搭載したラズパイ学習向け入門ボード。市販のGroveシステムに対応したセンサやアクチュエータなどを接続可能、更に拡張用スルーホールを備え、プロトタイピングにも最適です。



- 単色OLED
- 単色LED（赤）
- 押ボタンスイッチ
- カラーLED(赤・緑・青)
- 拡張用スルーホール
- Groveコネクター
- 圧電スピーカー

Linux/Unixシステム、プログラミング総合誌

シェルスクリプトマガジン

気軽に買える! 1コインITマガジン 気軽に読める!

Linux/Unix、シェルスクリプト/各種プログラミング言語、人工知能/深層学習、IoT/組み込み機器、IT開発/マーケティングの総合情報誌です。わずか500円(税別)で、さまざまなジャンルにおける注目の技術や手法を紹介しています。

- ★LinuxやUnixの知識が身に付く
- ★ラズパイや電子工作が始められる
- ★プログラムが書ける
- ★人工知能と機械学習が理解できる
- ★開発やマーケティングの手法が分かる
- ★レガシーシステムを移行できる

シェルスクリプトマガジン公式サイト
https://shell-mag.com/

サイズ:B5判
定価:本体500円+税
隔月(奇数月25日発売)

定期購読
年間料金:3,000円(送料サービス)
申し込みURL
https://www.usp-lab.com/pub.order.html

直販、Amazonおよび、20店舗以上の全国書店
- ●直販サイト
  https://www.usp-lab.com/pub.magazine.html
- ●取り扱い書店一覧
  https://www.usp-lab.com/pub.store.html