TAKE FREE

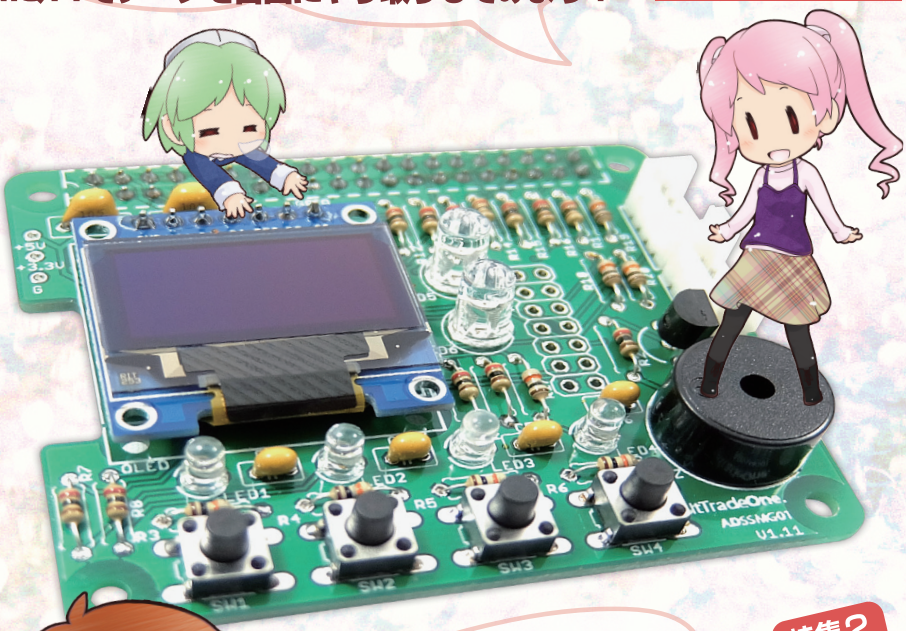
AdB

AssemblyDesk Books

特集1

ラズパイで IoTをはじめよう!

MQTTでデータを自由にやり取りしてみよう！



特集2

ラズパイ入門ボード 応用編 その3.4

OLEDへの画像表示と
赤外線リモコンをつないでみよう。

vol.03
201905

ラズパイではじめる電子制御と IoT

本誌について

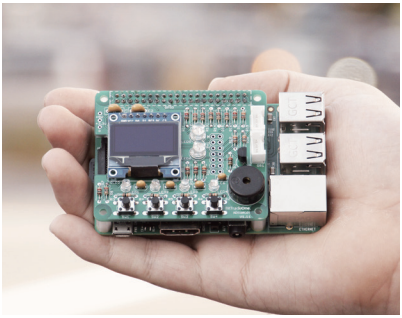
Assembly Desk Books はビット・トレード・ワンが発行するエンジニアやエンジニアを目指す方々に向けたフリーペーパーです。シェルスクリプトマガジンとの連動企画による「ラズパイ入門ボード」の解説や昨今話題の IoT を手軽に試すことができる情報が満載です。
Assembly Desk Books Vol.3 では特集記事として、このラズパイ入門ボードに関して OLED や光センサーの使い方の一部を紹介するほか、Node-RED と MQTT を利用した IoT 入門をご紹介します。
ラズパイ入門ボードに関するソースコードについては右記ページにて公開されております。
本書とあわせてご活用ください。

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

INDEX

ラズパイ入門ボードの入手方法

ラズパイ入門ボードは、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。



ADSSMG01

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。
価格(税別)は、完成品が 3980 円、組み立てキットが 2780 円です。(送料別)

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/

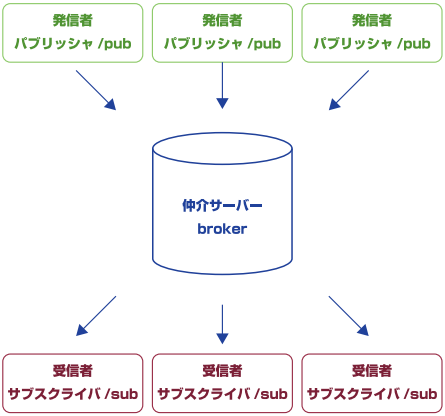
共立エレショップ
http://eleshop.jp/

ラズパイではじめる電子制御と IoT

はじめに

Assembly Desk Books Vol.2 では Node-RED の基本的な使い方を解説しました。今回はいよいよ MQTT を使い、インターネット上でデータをやり取りしてみます。

MQTT とは、発信機器 (パブリッシャ /Pub) と受信機器 (サブスクライバ /Sub)、その間でデータを仲介するサーバー (ブローカ /broker) から成り立つ、インターネット経由で遠隔地間の情報をやり取りする通信プロトコルです。



まず最初に、Publisher から受信したデータを仕分けし、Subscriber へ送信する MQTT 界の親分"Broker"のセットアップを行います。

MQTT brokerの準備

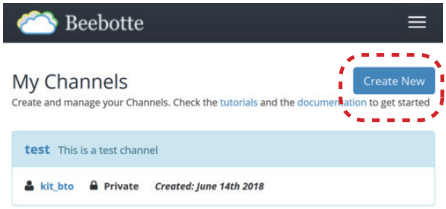
[クラウドMQTTブローカーのアカウント作成]

今回は broker として、クラウドの MQTT broker サービスの "beebotte"を利用します。

beebotte ホームページ(https://beebotte.com/)

無料プランでは 1 アカウントにつき一日当たり 5 万メッセージまで処理でき、暗号化通信にも対応していますので個人ユーザーが試しに使ってみるには十分な処理数です。アカウントはメールアドレスと、設定したいユーザ名 / パスワードを入力するだけで作成できます。

登録したメールアドレスにメールが届くので、メールの URL にアクセスすると認証が通りアカウント作成完了となります。この時点でログインが可能になります。アカウントを作成しログインすると、画像のような画面が表示されます。



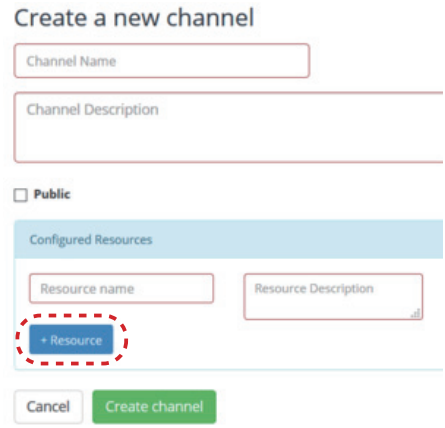
チャンネル/リソースの作り方

MQTT では、流れる情報に"トピック"を指定でき、特定のサブスクライバだけに情報を流す・特定のパブリッシャの情報だけを受け取るといった操作が出来ます。

今回使用する beebotte ではトピックを"チャンネル / リソース"の形で指定でき、データの内容はリソースに格納されます。リソースは一つのチャンネルに複数作成できます。

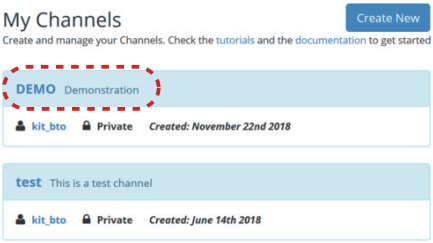
パブリッシャ / サブスクライバはチャンネルごとに発行される"トークン"を目印にメッセージをやり取りするため、一つのシステムに一つのチャンネルを割り当てるとよいでしょう。

右上の"Create New"ボタンをクリックすると、新しいチャンネルを作成できます。

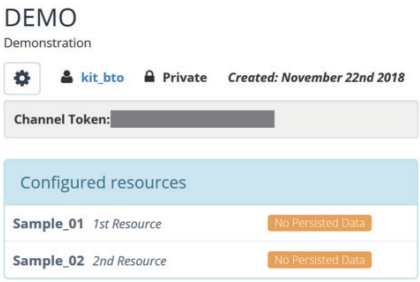


チャンネル / リソースの名前と説明を入力します。複数のリソースを用いる場合は"+ Resource"をクリックしてリソース欄を増やします。必要な情報を入力したら、"Create channel"をクリックしてチャンネルを作成します。

チャンネルを作成するとチャンネル一覧ページに遷移します。
ここで新しいチャンネルが追加されていることがわかります。



チャンネル名をクリックし、詳細情報を見てみます。

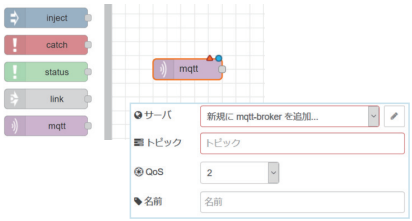


このページにある"Channel Token"が、
このチャンネルにアクセスするために必要な目印となります。

続いて、broker とデータをやり取りする Publisher、Subscriber を作っていきます。Node-RED 上に Publisher、Subscriber を作成してみます。

Node-REDからMQTT brokerへのアクセス
[Node-REDにMQTT brokerの情報を登録する]

Node-RED には MQTT にアクセスしデータをやり取りするノードが標準で装備されています。今回はこのノードに必要な情報をセットし、ブローカと通信します。
まず MQTT からのデータを受け付けるフローを作成します。
左側のパレットの"入力"グループより、"MQTT"ノードを配置しダブルクリックして設定画面を開きます。



"サーバ"欄の"新規に mqtt-broker を追加"の右にあるボタンから、MQTT ブローカの設定に入ります。

今回は以下のように設定します。
サーバ：mqtt.beebotte.com
ポート：8883
SSL/TLS 接続を使用：チェックする
また、"セキュリティ"タブを開き、"ユーザー名"の欄に

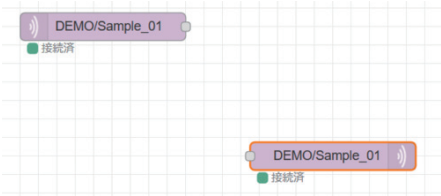
`token:token_`（ブローカのチャンネル設定で取得したトークン文字列）

の形式で入力します。

ここまで完了したら右上の"追加"ボタンをクリックし、ノードの設定画面に戻ります。トピック欄に（チャンネル名）/（リソース名）の形式でデータの受信先を設定します。

以上で入力ノードの設定は終了です。ここでデプロイを行い、画像のようにノードの左下に緑色の四角と"接続済"の文字が見えれば、設定成功です。

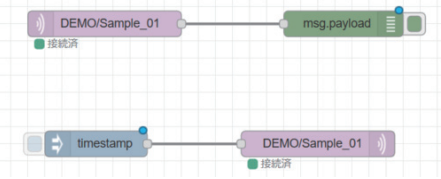
次に出力ノードを配置します。右側のパレットの"出力"グループより"MQTT"ノードを配置してください。
この時、ブローカ関連の設定は共有され、既に設定済みとなっています。ダブルクリックし、トピック欄に入力ノードと同様にデータの出力先を設定すれば設定完了です。



■ データを通信してみる

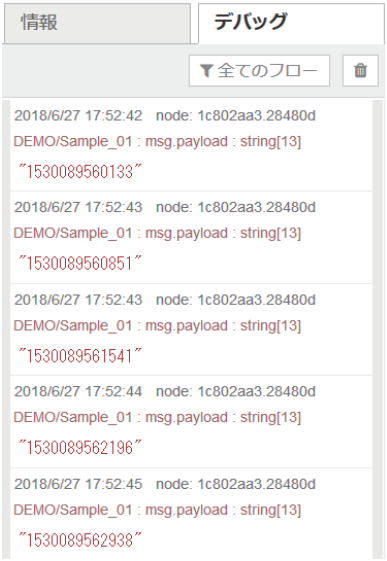
では、MQTT を経由してデータをやり取りしてみます。MQTT 出力ノードには、inject ノードを接続しデータを送信します。inject ノードは、デフォルトでは左側のトグルをクリックするとタイムスタンプ（'70 年 1 月 1 日 0 時丁度からの累計ミリ秒時間）を送信するようになっています。
MQTT 入力ノードからのデータは、"debug"ノードを使って観察します。左側のパレットの"出力"グループから debug ノードを配置してください。このノードには特別な設定は不要で、配置し結線してすぐに使えます。

以下のようなフローになります。



debug ノードに流れ込んだデータは、フローの右側にある情報欄の"デバッグ"タブに出力されます。ここに、inject ノードからのデータが表示されれば成功です。デプロイを行い、inject ノード左側のトグルをクリックして下さい。

次に出力ノードを配置します。右側のパレットの"出力"グループより"MQTT"ノードを配置してください。
この時、ブローカ関連の設定は共有され、既に設定済みとなっています。ダブルクリックし、トピック欄に入力ノードと同様にデータの出力先を設定すれば設定完了です。

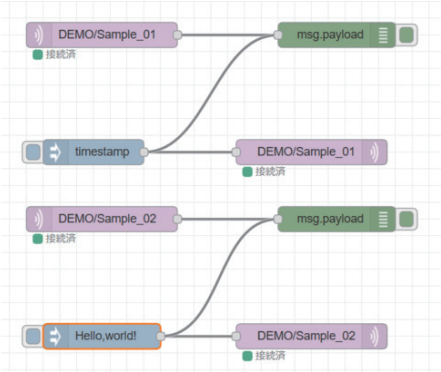


このように、クリックした回数分の出力がデバッグタブに表示されます。 実際に inject ノードの出力値を観察し、通信したデータが正しいか確認したいときは、inject ノードから debug ノードへ直接結線します。

この状態で inject ノードのトグルを入れると、一回のクリックでデバッグ欄にデータが二つ現れます。データ型が異なりますが、中身の値は一緒であることがわかります。

■ いろいろなデータを流してみる

inject ノードからは、様々な種類・内容のデータを発信することができます。
先ほど作成したフローを範囲選択と Ctrl+C でコピーし、Ctrl+V で貼り付けて下さい。貼り付けたフローの MQTT IN/OUT ノードの設定を開き、トピック欄を (チャンネル名)/(リソース名その 2) としてください。リソースを 1 つしか作成していない場合は、beebotte のコンソールよりチャンネルを編集しリソースを追加しましょう。



リソースを編集したら、inject ノードを編集します。
inject ノードをダブルクリックし、設定画面を開いてください。

“ペイロード”の欄をクリックし、”文字列”を選択してください。すると、右側の欄に任意の文字列を入力可能になります。ノードから出力させたい文字列を入力してください。
設定が終了したらデプロイを行い、文字列を設定した inject ノードのトグルを入れます。

MQTT 経由で文字列データも送信できていることがわかります。

このように、MQTT を利用することで様々なデータをインターネット上で自由にやり取りすることができます。
また、Node-RED にも多くのライブラリがあり、グラフや操作ボタンを作成できるものもあります。
皆様も Node-RED と MQTT、そして Raspberry Pi で実用できる IoT の第一歩を踏み出してはいかがでしょうか。

解説：Issou kitahara

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

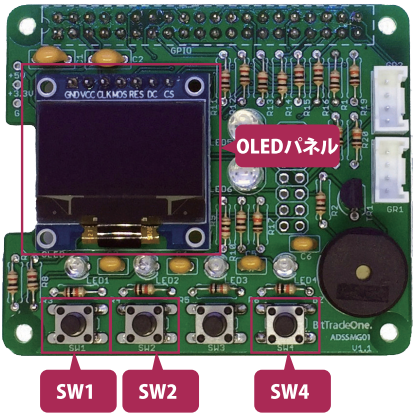
ラズパイ入門ボードで学ぶ電子回路の制御 その3

OLEDに画像を表示する

ガジェット 米田聡

小型コンピュータボード「Raspberry Pi」(ラズパイ)のプログラミングが楽しめる拡張ボード「ラズパイ入門ボード」を使った電子回路制御を取り上げていきます。第３回は、OLED に画像を表示します。

図1　今回使用するOLEDパネルとスイッチ



ラズパイ入門ボードに搭載されている 128×64 ドットの単色有機 EL ディスプレイ(OLED)パネル(図 1)は、白黒などの２値画像しか表示できないので、カラー写真などの表示には向いていません。それでもアイコンや絵、ちょっとしたアニメーションを表示したい人は多いようです。

今回は、OLED パネルに JPEG 形式の画像ファイルを表示する方法を紹介します。

PILを使って画像ファイルを表示する

米 Adafruit Industries 社(https://www.adafruit.com/)が公開している Python ライブラリを使うと、OLED パネルに画像を簡単に表示できます。このライブラリは、AssemblyDesk Books volume1 でも文字表示のために導入しました。
もし導入したことがなければ、次のように Raspbian 上でインストールします。なお、OLED パネルは「SPI」のシリアルインタフェースに接続されているので、Raspbian の設定画面で「SPI」を有効にしてください。

```
$ sudo apt update
$ sudo apt install python3-pip
$ sudo pip3 install Adafruit_SSD1306
```

このライブラリでは、「Python Imaging Library」(PIL)の「Image」オブジェクトを OLED に表示します＊1。第 1 回でも Image オブジェクトを新規作成し、そこに文字を描画しました。PIL の「open()」メソッドを使えば、画像ファイルから Image オブジェクトを作成できます。JPEG や PNG などのポピュラーな画像形式をサポートしているので、任意の画像ファイルから Image オブジェクトを作成可能です。

ほとんどの手持ちの画像ファイルはカラーでしょう。カラーでも問題ありません。PIL には高機能かつ高速な画像変換メソッドが用意されています。そのメソッドで、カラー画像を２値画像に簡単に変換できます。

図２はカレントディレクトリーにある画像ファイルを OLED パネルに表示するサンプルスクリプト(oledimage.py)です。次のように実行権を与えてください。

```
$ chmod +x oledimage.py
```

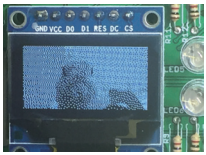
＊1　2018年7月上旬時点において、PILから派生した「PILLOW」(https://pillow.readthedocs.io/en/5.1.x/)というライブラリが使われています。ここでは、一般に通りが良いPILという名を使いました。

図2　JPEG形式の画像ファイルを表示するスクリプト（oledimage.py）

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import time
import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306
import RPi.GPIO as GPIO

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

class EB_OLED(Adafruit_SSD1306.SSD1306_128_64):

  WIDTH = 128
  HEIGHT = 64

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

  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)


if __name__ == "__main__":

  oled = EB_OLED()
  # 初期化
  oled.begin()
  oled.clear()
  oled.display()

  jpg = Image.open("sample.jpg")
  # 画像を縮小
  scale = float(float(EB_OLED.WIDTH)/float(jpg.size[0]))
  jpg = jpg.resize((int(jpg.size[0] * scale), int(jpg.size[1]*scale)))
  jpg = jpg.crop(box=(0, 0, EB_OLED.WIDTH, EB_OLED.HEIGHT))
  # 2値に変換
  image = jpg.convert('1')
  oled.image(image)
  oled.display()
```

図3　元画像とOLEDの表示

元画像（sample.jpg）



OLED パネルの表示

```
$ ./oledimage.py
```

　すると、画像ファイルが適切なサイズで OLED パネルに表示されます（図 3）。36 行目の「Image.open()」が画像ファイルから Image オブジェクトを作成するメソッドです。作成された Image オブジェクトは、画像ファイルの色数、サイズの属性を持っています。

　OLED パネルの解像度は 128×64 ドットです。まず、39 行目の「resize()」メソッドで、Image オブジェクト幅を「128」ドットに変換しています。リサイズ時にアスペクト比率を維持するように、リサイズするスケールを「scale」変数に格納しています（38 行目）。

　その上で画像を切り抜く「crop()」メソッドを使い、画像の上部 128×64 ドットを切り出しています（40 行目）。crop() メソッドの box パラメーターには、切り出す矩形の座標をタプル形式で渡します。次のようにすれば、画像上部の 128×64 ドットを切り出せます。

```
img = img.crop(box=(0 ,0 , 128,64))
```

　2 値画像への変換は「convert()」メソッドを使います（42 行目）。convert() メソッドの第 1 パラメーターが画像形式の指定です。「'1'」は 2 値画像を意味します。convert() メソッドは変換先が 2 値画像の場合、デフォルトでディザー＊処理を実施しますが次のように記述すればディザーなしの変換も可能です。単純な記号などはディザーがない方がよい場合もあるので適宜、使い分けてください。

```
img = img.convert('1', dither=Image.NONE)
```

　以上のようにして表示可能な Image オブジェクトが作成できます。第 1 回に紹介した文字表示と同様に、Adafruit Industries 社のライブラリを通じて Image オブジェクトを貼り付ければ、OLED パネルに表示されます。

【ディザー】中間色をランダムなドットに変換すること。中間色はドットの密度で表現される。

画像をスクロールさせてみよう

　先の例では画像の一部である、128×64 ドットしか表示されません。そこで、スイッチを組み合わせて画像をスクロールさせます。OLED パネルは高速な SPI バスで接続されている上、データ量が極めて小さい 2 値のビットマップです。Adafruit Industries 社のライブラリでも連続して異なる画像を表示させればアニメーションのように動かせます。スクロールも同様で、crop() メソッドで切り出す画像の位置を変えると、スムーズにスクロールするように表示できます。

　図 4 が、そのサンプルスクリプト（imgscroll.py）です。oledimage.py に記述した「EB_OLED」クラスはそのまま流用するので、「if __name__ == "__main__":」より前に書かれたコードを図 4 に追加して実行してください。「SW1」「SW4」のスイッチが上下のスクロール、「SW2」がスクロールの終了です。

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

図4　画像をスクロールできるスクリプト（imgscroll.py）

```
SW1=4
SW2=17
SW3=5
SW4=6

pos_y = 0
img_height = 0
f_exit = False
jpg = None

def sw_event(channel):

  global pos_y
  global f_exit
  global jpg
  global img_height

  if channel == SW1:
    if (img_height - pos_y - EB_OLED.HEIGHT) >= 0:
      pos_y += 1
  elif channel == SW4:
    if pos_y <= 0:
      return
    pos_y -= 1
  elif channel == SW2:
    f_exit = True

  image = jpg.crop(box=(0, pos_y, EB_OLED.WIDTH, EB_OLED.HEIGHT+pos_y))
  oled.image(image)
  oled.display()


if __name__ == "__main__":

  oled = EB_OLED()
  # OLED 初期化
  oled.begin()
  oled.clear()
  oled.display()

  # スイッチ初期化
  GPIO.setmode(GPIO.BCM)
  GPIO.setup([SW1, SW2, SW4], GPIO.IN)
  GPIO.add_event_detect(SW1, GPIO.FALLING, bouncetime=60)
  GPIO.add_event_detect(SW2, GPIO.FALLING, bouncetime=60)
  GPIO.add_event_detect(SW4, GPIO.FALLING, bouncetime=60)

  GPIO.add_event_callback(SW1, sw_event )
  GPIO.add_event_callback(SW2, sw_event )
  GPIO.add_event_callback(SW4, sw_event )

  jpg = Image.open("sample.jpg")
  scale = float(float(EB_OLED.WIDTH)/float(jpg.size[0]))
  jpg = jpg.resize((int(jpg.size[0] * scale), int(jpg.size[1]*scale)))
  # 2 値に変換
  jpg = jpg.convert('1')
  img_height = jpg.size[1]

  image = jpg.crop(box=(0, 0, EB_OLED.WIDTH, EB_OLED.HEIGHT))
  oled.image(image)
  oled.display()

  while f_exit == False:
    time.sleep(0.1)

  GPIO.cleanup()
```

ラズパイ入門ボードで学ぶ電子回路の制御 その4

赤外線レシーバでラズパイを制御 ガジェット 米田聡

小型コンピュータボード「Raspberry Pi」（ラズパイ）のプログラミングが楽しめる拡張ボード「ラズパイ入門ボード」を使った電子回路制御を取り上げていきます。第４回は、赤外線レシーバを扱います。

今回は Grove コネクタに接続できる赤外線レシーバを使います。Raspberry Pi に赤外線レシーバを搭載することで、普段利用している赤外線リモコンから Raspberry Pi を制御できます。

赤外線センサーを接続する

赤外線リモコンはテレビやエアコンでおなじみでしょう。このリモコンには赤外線 LED が組み込まれていて、押されたボタンに対応する赤外線パルスを出力します。

赤外線リモコンは幅広い機器で利用されているので専用の受信用センサーが商品化されています。一般的には、三つの端子を持つセンサーで、電源と GPIO（汎用 I/O）端子に直結できるのが特徴です。

図1　赤外線レシーバの「Grove - Infrared Receiver」

受信用センサーを搭載する Grove System 対応ハードエアは赤外線レシーバ「Grove - Infrared Receiver」という製品です（図 1）。この赤外線レシーバは海外製品ですが、日本でも千石電商やスイッチサイエンスなどから購入できます。価格は 410 ～ 583 円です。

Grove コネクタには四つのピンがあり、二つのピンが電源と GND に、残りの二つが GPIO につながっています。前述のように、赤外線センサーは３端子の部品です。そのため、Grove - Infrared Receiver では電源と GND、黄色のラインだけが使われていて、残りのピンは未使用になっています。

ラズパイ入門ボード側では Grove コネクタの「GR1」が GPIO 接続用、「GR2」が I²C 接続用です。よって、Grove - InfraredReceiver は GR1 コネクタに接続します（図 2）。

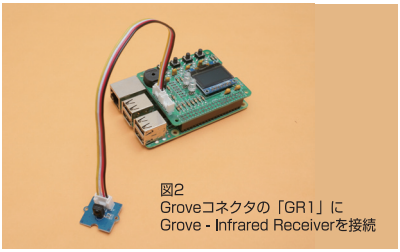


図2
Groveコネクタの「GR1」に
Grove - Infrared Receiverを接続

赤外線センサーの動作チェック

Linux では、赤外線リモコンを扱うソフトウエアとして「LIRC*1」（Linux Infrared Remote Control）が標準的に使われています。LIRC には細かな問題もありますが、さまざまなソフトウエアでサポートしています。最初に、この LIRC で赤外線センサーを使ってみるべきでしょう。

Raspbian を導入した Raspberry Pi を起動したら、端末を開いて次のコマンドを実行し、LIRC を導入します。

```
$ sudo apt update
$ sudo apt install lirc
```

*1　LIRCの公式サイトは「http://www.lirc.org/」。

インストール後、Raspbian 起動時の設定ファイル（/boot/config.txt）を「nano」エディタで開いてください。

```
$ sudo nano /boot/config.txt
```

LIRC ドライバ用の「Device Tree Overlay*2」の指定行がコメントアウトされているので、有効にして図３のように書き換えます。

*2　「Device Tree」はカーネルにハードウエア構成を教えるための仕組みです。「Overlay」を使って現在のハードウエア構成に別のハードウエアを追加できます。

図3　/boot/config.txtファイルの書き換え箇所

```
（略）
# Uncomment this to enable the lirc-rpi module
dtoverlay=lirc-rpi:gpio_in_pin=21,gpio_in_pull=down
↑「#」を削除して変更する
（略）
```

GR1 の Grove コネクタは、白が「GPIO25」、黄色が「GPIO21」につながっています。したがって、赤外線レシーバの GPIO ポートを設定する「gpio_in_pin」の右辺に「21」を記述します。また、「gpio_in_pull」でプルアップ（up）かプルダウン（down）を設定できます。実は、Grove - Infrared Receiver は基板上でプルダウンされているので、通常「gpio_in_pull=down」の指定は必要ありません。ただし、Raspbian の仕様で、gpio_in_pull を使ってプルダウンを指定しないと正常に動作しません。ここではプルダウンの指定を追加しています。

以上の変更を加えたら、/boot/config.txt を保存してください。Raspbarry Pi を再起動し、適当な赤外線リモコンを用意します。赤外線リモコンなら何でも構いません。

再起動後にログインして端末を開き、次のコマンドを実行します。

```
$ mode2 --device /dev/lirc0 --driver default
```

そして、リモコンのどれかのボタンを押してみてください。図４のような表示が端末に現れれば、受信は成功しています。ちなみに、「space」はパルスとパルスの間隔を、「pulse」はパルスの幅を意味しています。最初の「space」はボタンを押すまでの時間になるので、非常に大きな数字です。

図4　リモコンからの受信

```
Using driver default on device /dev/lirc0
Trying device: /dev/lirc0
Using device: /dev/lirc0
space 16777215
pulse 9001
space 4429
pulse 607
space 509
pulse 599
（略）
```

表示を確認できたら、[Ctrl] キーを押しながら [C] キーを押し、先ほど実行したコマンドを停止します。

リモコンを学習させる

LIRC では、「lircd」というデーモン（常駐プログラム）を通じて、受信した赤外線信号をユーザープログラムに伝える仕組みを提供しています。この仕組みを使うには、リモコンのボタンとパルス信号の対応を lircd に設定する必要があります。

設定するためのツール「irrecord」が用意されています。irrecord を使うには、まず lircd を停止します。次のコマンドを実行して停止させましょう。

```
$ sudo systemctl stop lircd
```

ここでは、例としてボタン二つだけを設定します。図５にその設定方法を示しました。図５の操作によって「button_a」と「button_b」という名前で二つのボタンを記憶した設定ファイル（remocon.lircd.conf）がカレントディレクトリに作成されます。

図5　リモコンを学習させる

```
$ irrecord -f -n -d /dev/lirc0 --driver=default
（略）
Press RETURN to continue.  ← [Enter]キーを押す
（略）
Enter name of remote (only ascii, no spaces) :
remocon  ← リモコン名(ここでは「remocon」)を入力する
（略）
Press RETURN now to start recording.  ← [Enter]キーを押す

ここでリモコンのボタンをランダムに押していく。受信するとドット(.)が
表示されるのでドットが1 行以上になるまでランダムに押し続けると終了する

Please enter the name for the next button (press<ENTER> to finish recording)
button_a  ← 記憶させるボタン名(ここでは「button_a」)を入力する
Now hold down button "button_a".
ここで「button_a」に割り当てたいリモコンボタンを 1 度だけ押す
Please enter the name for the next button (press<ENTER> to finish recording)
button_b  ← 次に記憶させたいボタン名(ここでは「button_b」)を入力する
Now hold down button "button_b".
ここで「button_b」に割り当てたいリモコンボタンを 1 度だけ押す
Please enter the name for the next button (press<ENTER> to finish recording)
2 個のボタンを記憶させたので[Enter]キーだけ押して学習を終了する
```

このファイルを「/etc/lirc/lircd.conf.d」ディレクトリ以下にコピーすると機能しますが、その前にデフォルトの設定を次のようにして削除します。

```
$ sudo rm /etc/lirc/lircd.conf.d/devinput.lircd.conf
```

その上で remocon.lircd.conf をコピーします。

```
$ sudo cp remocon.lircd.conf /etc/lirc/lircd.conf.d/
```

最後に、LIRC の設定ファイル（/etc/lirc/lirc_options.conf）を Raspberry Pi 向けに変更します。nano エディタで /etc/lirc/lirc_options.conf ファイルを開いてください。

```
$ nano /etc/lirc/lirc_options.conf
```

図６に従って内容を変更してください。書き換えたら、lircd を再起動します。

```
$ sudo systemctl restart lircd
```

図6　/etc/lirc/lirc_options.confファイル

```
（略）
[lircd]
nodaemon = False
driver = default  ← 右辺を「default」変更する
device = /dev/lirc0  ← 右辺を「/dev/lirc0」に変更する
（略）
```

図7　「rctest.lircrc」ファイル

```
1 begin
2 button = button_a  ← ボタンのラベル名
3 prog = rctest  ← プログラムの識別文字列
4 config = button_a  ← プログラムに渡す文字列
5 end
6
7 begin
8 button = button_b
9 prog = rctest
10 config = button_b
11 end
```

自作スクリプトとリモコンを連動させる

最後に、自作スクリプトとリモコンを連動させる例を紹介します。まずは、LIRC と Python を連動させるライブラリを導入しましょう。

```
$ sudo apt install python3-lirc
```

次に、カレントディレクトリに図７の「rctest.lircrc」ファイルを作成します。rctest.lircrc は一つのセクションが「begin」で始まり「end」で終わる構造になっています。「button」の右辺がボタンのラベル名、「prog」の右辺がプログラムを識別する名前、「config」 の右辺がプログラムに渡す文字列です。この例では、button_a と button_b に登録したリモコンボタンが押されるとプログラムにはそれぞれ「button_a」と「button_b」の文字列が渡ります。

リモコンボタンと連動する Python プログラム（rctest.py）は図８の通りです。「lirc.nextcode()」関数では、次のリモコンボタンが押されるまで待ち、返値のインデックス０に「rctest.lircrc」に設定した文字列が返ります。このプログラムでは、button_a に登録したリモコンボタンが押されるたびにラズパイ入門ボードの「LED1」が点灯と消灯を繰り返します。button_b に登録したリモコンボタンが押されるとプログラムが終了します。次のように実行して動作を確かめてみてください。

```
$ python3 rctest.py
```

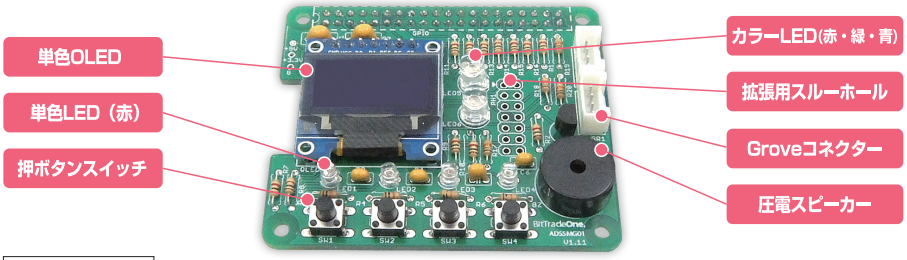
図8　ラズパイ入門ボードのLED1をリモコンで操作する Pythonプログラム（rctest.py）

```
import time
import lirc
import RPi.GPIO as GPIO

LED1 = 14
LED2 = 15
LED3 = 12
LED4 = 16

leds = [LED1, LED2, LED3 ,LED4]
flag_a = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(leds, GPIO.OUT, initial=GPIO.LOW)

socketid = lirc.init('rctest', './rctest.lircrc')

while True:
  code = lirc.nextcode()
  if code[0] == 'button_a':
    flag_a ^= 1
    GPIO.output(LED1, flag_a)
  elif code[0] == 'button_b':
    break
GPIO.cleanup()
```

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。



搭載デバイス

・米 Adafruit Industries 社製「SSD1306 OLD」互換の OLED×１・単色 LED×４・カラー LED×２・押しボタンスイッチ ×１・圧電スピーカー ×１・Grove System コネクタ ×２・拡張用スルーホール ×１

Raspberry Pi との接続

■OLED　SPI バス接続　CS：SPI_CE0　DC：GPIO23　RST：GPIO24　SDA：SPI_MOSI　SCL：SPI_CLK
■単色 LED(H：点灯、L：消灯)　LED1：GPIO14　LED2：GPIO15　LED3：GPIO12　LED4：GPIO16
■カラー LED(H：点灯、L：消灯)　[LED5]　R：GPIO20　G：GPIO27　B：GPIO22　[LED6] R：GPIO26　G：GPIO13　B：GPIO19
■押しボタンスイッチ(オン：L、オフ：H)　SW1：GPIO4　SW2：GPIO17　SW3：GPIO5　SW4：GPIO6
■圧電スピーカー(H：通電、L：休止)　GPIO18
■Grove System コネクタ　[GR1] Y：GPIO21　W：GPIO25　R：+3.3V　B：GND　[GR2] Y：I2C_SCL　W：I2C_SDA　R：+3.3V　B：GND
■拡張用スルーホール 1、2：+5V　3、4：+3.3V　5：ID_SD　6：ID_SC　7：I2C_SDA　8：I2C_SCL　9：SPI_MOSI　10：SPI_CE1　11：SPI_MISO　12：SPI_CLK　13、14：GND

AdB#03

AssemblyDesk Books Volume.03

POSTSCRIPT

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
挿　絵	信行(no-bu.net)
連 絡 先	info@bit-trade-one.co.jp
企画構成	Gr@phic
発 行 日	2019/03/15(初版)

５０年という時間と、うまくない焼きそば　作: 桃町正暗

両国国技館から東京都慰霊堂を抜けて横網方面へ、蔵前、両国と大相撲とは切っても切れない土地柄から、よく間違われるけど横網はヨコズナではなく、ヨコアミなんだよ。清澄通りをちょいと横にいって住宅街を入ったところ、何処？と聞かれるとちょっと困るあたり。

色あせた黄浅緑のテントに書かれた御食事処と藍染めの暖簾がなかったら、軒先の植木が店の入り口半分を占めて間違いなく見落とすレベル。周辺と同化する昭和感　というか昭和そのものの佇まい。その暖簾の奥に"味自慢、焼きそば"の旗がチラチラ。

小腹も空いていたので焼きそばでもと、古風なガラス戸をガラッと開けると、小さいおばあちゃんが「いらっしゃいませ」

カウンターだけのこれまた小さな店内を見回すと、そば、うどんの他、焼き魚定食にカツ丼や玉子丼なんかの丼ものも、取り揃えてあるみたい。勿論カレーライスも。

客は俺、一人か・・。しっかし、おばあちゃん　ちっさ！

壁に添えつけられた羽の青い扇風機、大相撲の番付表、カラーボックスに無造作に突っ込まれた漫画本。

あまりに昭和そのままに失神しそうになる。 こりゃ時間が止まってるね。

壁に貼られた手書きのメニューには　焼きそば　３５０円、大盛り４５０円。

「焼きそば　ひとつ」

「はい、焼きそばありがとうございます。」おばあちゃんが元気よく答えて、中華鍋をだして手際よく進めていく、体が小さいから、やけに中華鍋が大きく見える、アレに身を隠せばちょっとした銃撃くらいはやり過ごせそうだな。

そういえば表の看板にアサヒビールってあったな。。

「ビールください」

「ごめんなさい。前は置いてたんだけど、車でくる人が飲んじゃうからお酒置くのやめたのよ。よかったら向かいの酒屋さんで買ってください」

振り返って反対側を見ると確かに酒屋がある。確かに酒屋に見えるが営業している気配がまったく感じられない。「今の時間なら、まだ配達にいってないと思うから行ってみなよ」細かすぎる豚肉を炒めながら、おばあちゃんがいう。向かいの酒屋に行ってみると、入り口の前には空き瓶の入ったビールケースが山積みで、営業してませんオーラが漂っている。自動ドアの前にたってみても開く気配がない。ふと見ると"壊れています。手で開けてください"と貼り紙がある。

「あちゃー　ほんとにやってんの？」

やけに重い自動ドアを力づくであけると、暗い店内の奥に、これまた暗い感じのおっちゃんが一人座っている。

「向かいのおばちゃんが欲むなら、ここで買ってもってこいと」おっちゃん「はいはい　どうぞ、どうぞ」とすべて心得た感じ。

でかい冷蔵庫のなかにはアサヒ、サッポロ、エビス、キリンいちおうすべて揃っているが店のやる気を表すかのようにスカスカ、ここは先のアサヒの看板に敬意を評してアサヒにしておくかとスーパードライを一本買って店に戻る。

店ではおばあちゃんが小さな体で中華鍋を振るって焼きそばを絶賛制作中。味の素をドバッと投入　なんだか解らない謎の粉も投入　ソースがちょと焦げて　いい香りだね。スーパードライのプルタブをひねると水玉模様のコップを出してくれる。注ぐと盛泡みたいなでかい泡ができて、一口いただくと焼きそばが登場。

平たく皿に盛られた焼きそばは、最近はやりの真っ黒いタイプではなく、そこはかとないソース色、青のりはなく、端に添えられた紅しょうがの紅一点。

一口たべると、ガツンとくるソース味と味の素「ああ　なんだか懐かしいな・」でも、うまくない笑。

焼きそばなんて、気取った食いもんじゃないんだから、こんなのでいいんだよ。ビールと焼きそばを交互にやっつけながら、おばあちゃんと何となくおしゃべり。

「何年からやってるんですか？」

「もう何十年よ」

「まさか戦前からってことは無いですよね　笑」

「まさか！その頃、このあたりは焼け野原よ笑　蔵前も７０年なんていうから、かれこれ５０年くらいじゃないかしら？　蔵前に国技館があった時は三段目の子たちが、よく稽古のあとに来てたわよ。若い子は稽古が遅く終わるからちゃんこあまり食べられないのよね。」

へぇーさすが　相撲取り、聞けば一人うどん４杯にカツ丼２杯を食べていくそうだ。やがて話は北の湖の話に、大抵の力士は出世するとそっけなくなるみたいだけど、彼は出世するたびにキチンと挨拶に来てたそうで。「銀座で床山さんたちとボトル４本開けて、そのまま朝の稽古にいったこともあったわよ」なんて武勇伝も。

帰り際「こんどはカレーも食べって、評判なのよ。」 うん、また来るよ。カレーもだいたい想像つくけど。

"こういうところも、そのうち消えていっちまうんだろうな・・"

食べログだか、なんだかに昭和も知らないやつらに"昭和の雰囲気！最高の店です。"とか書かれてしまうんだろうか。それを見てきた連中に、あまりうまくない焼きそばについて薀蓄を書かれるんだろうか。

でも、そんなものはクソ食らえだ。

ここで５０年やってんだ。

このままでいいじゃないか。

marutsu®
ideas for making

Digi-Key社取扱い[※1]全製品を1個[※2]からマルツで購入できます

3,000円[※3] 以上
お買い上げで
送料無料
通常送料全国一律
240円～450円

マルツは
Digi-Key社の
正規代理店です

Digi-Key® ELECTRONICS

www.marutsu.co.jp　マルツオンライン

※1 マルツオンラインに一部未掲載製品がございますので、別途お見積りいたします。
※2 LOT 単位・複数個単位の製品もございます。
※3 マルツオンラインでの一度のご購入金額の合計。Digi-Key社製品に限らず、マルツオンライン掲載の全製品での合算になります。
● 価格はすべて税抜表記です。

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

スタッフ募集! システムエンジニア(SE)、プログラマー
URL https://www.usp-lab.com/recruit.html ユニバーサル・シェル・プログラミング研究所