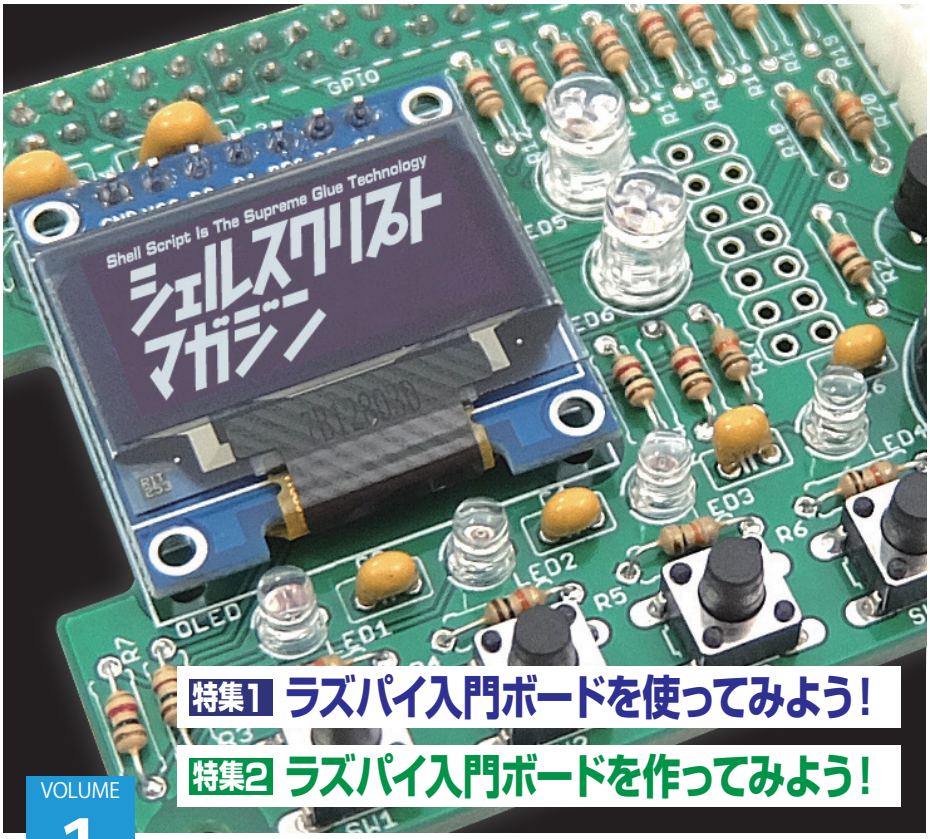
ラズパイ制御 丸わかり!!!

TAKE FREE

シェルスクリプトマガジン連動 " ラズパイ入門ボード ではじめる電子制御 "

超入門!! 学んでおきたい制御方法 基礎の基礎



特集1 ラズパイ入門ボードを使ってみよう!

特集2 ラズパイ入門ボードを作ってみよう!

VOLUME 1

本記事はUSP研究所 シェルスクリプトマガジン2018年Vol.52の記事 "ラズパイ入門ボードを使ってみよう。"を再構成したものです。

AssemblyDesk. BOOKS

"ラズパイ入門ボード ではじめる電子制御 "

超入門!! 学んでおきたい制御方法 基礎の基礎

■ 本誌について

シェルスクリプトマガジンとの連動企画で生まれたラズパイ学習向け、入門ボード「ラズパイ入門ボード」４個の赤色 LED、２個のカラー LED、４個の押しボタンスイッチ、単色有機 EL ディスプレイ、圧電スピーカーなど、初めてラズパイを使う初心者にとって制御方法を知っておくべき電子部品が搭載されています。Assembly Desk Books Vol.1 では特集記事として、このラズパイ入門ボードに関する基本的な使い方の一部を紹介します。

本製品向けのソースコードについては以下ページにて公開されております。
本書とあわせご活用ください。

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（http://bit-trade-one.co.jp/）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。

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）、Amazon.co.jp、 共 立 エ レ ショップ（http://eleshop.jp/）および、シリコンハウス、千石電商、デジット、マルツエレック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）

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」。
本特集では、このラズパイ入門ボードに関する基本的な使い方の一部を紹介します。

※本記事は USP 研究所 シェルスクリプトマガジン 2018 年 Vol. 52 の記事”ラズパイ入門ボードを使ってみよう。”を再構成したものです。

ラズパイ入門ボードで始める電子制御

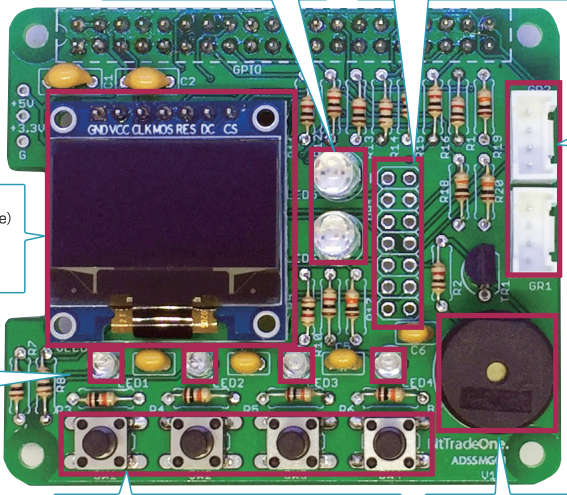
ガジェット 米田聡

まず、ラズパイ入門ボードに搭載されている機能を一通り見ておくことにします。ラズパイ入門ボードには、初めてラズパイを使う人にとって制御方法を知っておきたい電子部品が搭載されています（図 1）。具体的には、4 個の赤色 LED、2 個のカラー LED、4 個の押しボタンスイッチ、単色有機 EL ディスプレイ（OLED パネル）、圧電スピーカー（ブザー）です。

また、センサーなどを追加実装できるように Grove コネクターや拡張用スルーホールも用意しています。

以降では、これらの中から赤色 LED、カラー LED、押しボタンスイッチ、圧電ブザーを制御するためのプログラミング方法を解説します。OLED パネルや Grove コネクターに搭載するセンサーなどを制御するプログラミング方法は、次号でご紹介いたしますのでお楽しみに。

図 1 ラズパイ入門ボードの主な機能
LED、押しボタンスイッチ、OLED、圧電スピーカーなどが実装されている。



ボードの取り付けとRaspbianのセットアップ

ラズパイへのラズパイ入門ボードの取り付けは簡単です。図 2 のようにラズパイ入門ボードの 40 ピンソケットを、ボードが積み重なるようにラズパイの 40 ピンヘッダーに差し込みます。1 ピンずれて取り付けるなどすると最悪、壊れることもあり得るので、その点だけ注意します。

続いて OS をセットアップします。本特集では、ラズパイ向けの標準 Linux ディストリビューションである「Raspbian」を利用します。「https://www.raspberrypi.org/downloads/noobs/」から入手できるインストーラー「NOOBS」と microSD カードを使って Raspbian を導入しておいてください。

ラズパイ入門ボードでは、GPIO（General-Purpose Input/Output：汎用入出力）のほかに OLED パネルが SPI（Serial Peripheral Interface）を利用しています。また、Grove コネクターに I²C（Inter-Integrated Circuit）を利用します。Raspbian では標準状態で GPIO は使えるものの、SPI、I²C ともに無効になっています。次の手順で有効にしておきましょう。

Raspbian のデスクトップ画面左上にあるメインメニュー（ラズベリーのアイコン）から「設定」-「Raspberry Pi の設定」を選択します。図 3 のように SPI と I²C を「有効」にセットします。また、SSH も有効にしておくと他のホストからログインできるので便利でしょう。

設定を変えたら「OK」ボタンをクリックして Raspberry Pi の設定ダイアログを閉じて、Raspbian を再起動して設定を反映します。

図2　ラズパイ入門ボードを取り付ける
ラズパイ入門ボードの40ピンソケットを、ラズパイの40ピンヘッダーに差し込む。

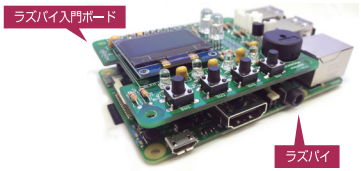


図3　I²CとSPIを有効にする

単色LEDとスイッチを使おう

まずは基本的なところで、「スイッチを押したら単色 LED が点灯する」という簡単なプログラムを完成させてみましょう。

スイッチと単色 LED は表 1 に示した GPIO に接続されています。GPIO は、デジタルで入出力できる汎用のインタフェースです。GPIO14 を出力(OUT)に設定し、「1」を書き込めば LED が点灯します。GPIO4 を入力(IN)にすれば、スイッチの状態(ステータス)を読み取ることができます。

表1　単色LEDとスイッチ、GPIOの対応

基板上のシルク	GPIO番号
SW1	GPIOP4
SW2	GPIO17
SW3	GPIO5
SW4	GPIO6
LED1	GPIO14
LED2	GPIO15
LED3	GPIO12
LED4	GPIO16

GPIO の制御には、プログラミング言語「Python」と、ラズパイ向けの GPIO ライブラリ「RPi.GPIO」が広く利用されています。Raspbian には双方ともあらかじめインストールされていますが、端末を開いて次のコマンドを実行し、念のためアップデートしておいた方がよいでしょう。

```
$ sudo apt-get update
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

図4 Pythonの定数定義ファイル(entryboard.py)

```
1 LED1 = 14
2 LED2 = 15
3 LED3 = 12
4 LED4 = 16
5
6 SW1 = 4
7 SW2 = 17
8 SW3 = 5
9 SW4 = 6
```

LED やスイッチは、ラズパイ入門ボードを使っていく上で頻繁にアクセスすることになります。したがって、次のような Python の定数定義ファイルを作っておくと便利です。テキストエディタで図 4 のコードを入力し、「entryboard.py」という名前でホームディレクトリーに保存しておいてください。以降のプログラムはすべて、entryboard.py と同じディレクトリーに保存して実行します。

LEDを光らせてみる

では、LED を光らせてみます。図 5 の「led.py」は LED1 を 2 秒間点灯させるプログラムです。次のように実行してください。
ボード上の LED1 が 2 秒間点灯します(図 6)。

```
$ python led.py
```

それではプログラムを見ていきましょう。RPi.GPIO を利用する場合、まず「GPIO.setmode()」を使ってライブラリの動作モードを設定する必要があります(8 行目)。GPIO.setmode() の引数には、「GPIO.BCM」と「GPIO.BOARD」の 2 通りがあります。GPIO.BCM を指定した場合、搭載されている SoC(System on a Chip)の GPIO 番号を利用します。本特集ではこちらを利用します。

一方、GPIO.BOARD を指定した場合は、ピンヘッダー内のピン番号を利用します。例えば、LED1 がつながっている GPIO14 は 8 番ピンなので、GPIO.BOARD による GPIO の指定には「8」を使うことになるわけです。

自作のプログラムでは、どちらを使っても構いませんが、ラズパイ入門ボードの場合はコネクターをピンヘッダーに挿すだけなのでピン番号よりは GPIO 番号の方が分かりやすいでしょう。

ライブラリの動作モードに続いて「GPIO.setup()」で GPIO の入出力モードを設定します(10 行目)。LED への信号は GPIO から出力することになるので、

```
GPIO.setup(LED1, GPIO.OUT, initial=GPIO.LOW)
```

とします。「initial」は必ずしも指定する必要はありませんが、上のように指定しておくと GPIO 初期化時に「LOW」(=0)が出力された状態になります。

GPIO への出力は「GPIO.output()」です(12、16 行目)。

```
GPIO.output(LED1, GPIO.HIGH)
```

これで「HIGH」(=1)が出力され、LED が点灯します。プログラムでは 14 行目の「time.sleep()」を使って 2 秒待ち、16 行目で LOW を出力して消灯させています。

最後に、18 行目で「GPIO.cleanup()」で使用した GPIO をクリーンアップして終わらせます。クリーンアップをしないでプログラムを終わらせると、次に同じ GPIO を GPIO.setup() で初期化するときに「その GPIO は使用中」という警告(Warning)が表示されます。通常は実害ありませんが、あまり気分がよいものではありません。できるだけ GPIO.cleanup() を実行してプログラムを終わらせておきましょう。

図5　LED1を2秒間点灯させるプログラム（ led.py）

```
# -*- coding: utf-8 -*-
# entryboard.pyを読み込む
from entryboard import *
import time
import RPi.GPIO as GPIO

# 動作モード設定
GPIO.setmode(GPIO.BCM)
# GPIOモード設定
GPIO.setup(LED1, GPIO.OUT, initial=GPIO.LOW)
# LED1に1を出力
GPIO.output(LED1, GPIO.HIGH)
# 2秒待つ
time.sleep(2)
# LED1に0を出力
GPIO.output(LED1, GPIO.LOW)
# 終了処理
GPIO.cleanup(LED1)
```

図6　led.pyの実行結果

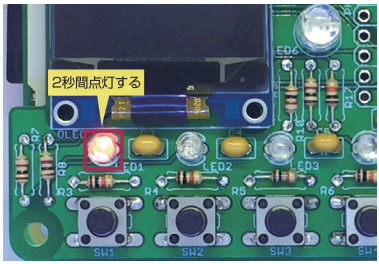


図7　4個のLEDが順に点灯するプログラム（leds.py）

```
# -*- coding: utf-8 -*-
from entryboard import *
import time
import RPi.GPIO as GPIO

# LEDが接続されているGPIOのリスト
leds = [LED1, LED2, LED3 ,LED4]

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

for i in range(3):
  for l in leds:
    GPIO.output(l, 1)
    time.sleep(1)
    GPIO.output(l, 0)
GPIO.cleanup(leds)
```

四つのLEDを点灯させる

続いて４個のLEDを点灯させてみましょう。図７の「leds.py」は、LED1からLED4までを順番に１秒ずつ点灯させ、それを３回繰り返すプログラムです。次のように実行してください。LED1から順に点灯します（図８）。

```
$ python leds.py
```

このプログラムのポイントは、GPIO.setup() やGPIO.cleanup() の引数として「リスト」が取れる点です（7行目）。リストは、データをひとまとめにする配列のようなものです。

例えば、次のようにすれば四つのGPIOをまとめて出力に初期化できます。

```
GPIO.setup([LED1, LED2, LED3 ,LED4], GPIO.OUT, init
ial=GPIO.LOW)
```

プログラムを簡潔にできるので、ぜひ覚えておいてください。

図8　leds.pyの実行結果
左から順に4個のLEDが点灯する。

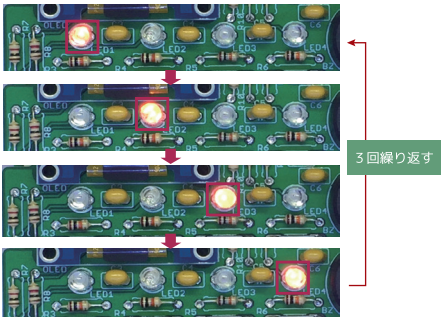
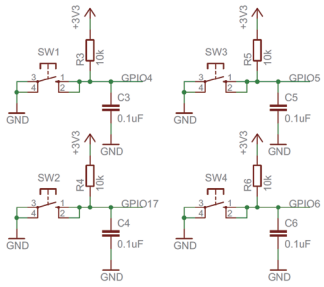


図9　スイッチ部分の回路図



スイッチを制御する

続いてスイッチを使ってみましょう。スイッチはLEDとは逆に、GPIOから入力します。ラズパイ入門ボードでは、図９の回路図が示すように「スイッチをオンにするとGPIOがGND（±0V）につながる」ように配線されています。したがって、スイッチが押されていないときにはGPIOからHIGHが、スイッチが押されるとLOWが読み取られます。

では、スイッチが押されたらその上にあるLEDが点灯するプログラム（switch1.py）を作成してみます。図10ができです。

図10　スイッチとLEDを連動させたプログラム　（switch1.py）

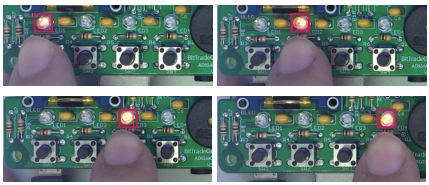
```
# -*- coding: utf-8 -*-
from entryboard import *
import time
import sys
import RPi.GPIO as GPIO

switches = [SW1, SW2, SW3, SW4]
leds = [LED1, LED2, LED3, LED4]
GPIO.setmode(GPIO.BCM)
# スイッチのGPIOを入力に設定
GPIO.setup(switches, GPIO.IN)
GPIO.setup(leds, GPIO.OUT, initial=GPIO.LOW)

try:
  while True:
    for i in range(4):
      if GPIO.input(switches[i]) == GPIO.LOW:
        GPIO.output(leds[i], GPIO.HIGH)
      else:
        GPIO.output(leds[i], GPIO.LOW)

except KeyboardInterrupt:
  GPIO.cleanup(switches)
  GPIO.cleanup(leds)
  sys.exit()
```

図11　switch1.pyの実行結果
押しボタンスイッチを押すと、その上にあるLEDが点灯する。



次のように switch1.py を実行してください。

```
$ python switch1.py
```

スイッチを押すと、その上に取り付けられている LED が点灯します（図11）。このプログラムは永久ループになっているので、終わらせるときには端末上で[Ctrl]キーを押しながら[C]キーを押します。これらのキーが押されると「KeyboardInterrupt」例外を補足して「GPIO.cleanup()」でクリーンアップして終了するようにしています（22 ～ 25 行目）。

GPIO を入力として使うときには、GPIO.setup() で該当する GPIO に対して「GPIO.IN」をセットします（11 行目）。入力に設定された GPIO からは、「GPIO.input()」で現在のステータスを読み出すことができます（17 行目）。

```
value = GPIO.input(GPIO番号) # valueはGPIO.HIGHまたは
GPIO.LOW
```

switch1.py ではスイッチが接続された GPIO が GPIO.LOW なら、その上にある LED につながっている GPIO を GPIO.HIGH にし、そうでないなら逆にする処理を永久ループ内で実行しています。

ちなみに、switch1.py では処理の分岐に if ～ else を使っています（17 ～ 20 行目）。ただ、ライブラリ内で GPIO.HIGH は「1」、GPIO.LOW は「0」が定義されているので、if ～ else を次の 1 行に置き換えることができます。

```
GPIO.output(leds[i], GPIO.input(switches[i]) ^
GPIO.HIGH)
```

このように XOR 演算（^）で GPIO.input() の値を反転させ、LED の GPIO に出力すれば if ～ else と同じことができるわけです。ただし、ほとんどあり得ないのですが将来、GPIO.HIGH と GPIO.LOW の値がライブラリ内で変更された場合、XOR 演算は機能しなくなる可能性があります。定数として隠ぺいされている生の値を前提にしたコーディングは非互換性を生じさせる原因になるので、好ましいとはいえないのも確かです。煩雑でも if ～ else を使っておく方が安全です。

図12　GPIO割り込みを使ったスイッチ・LED連動プログラム（switch2.py）

```
# -*- coding: utf-8 -*-
from entryboard import *
import time
import sys
import RPi.GPIO as GPIO

switches = [SW1, SW2, SW3, SW4]
leds = [LED1, LED2, LED3, LED4]

# スイッチのGPIOが変化したらこの関数が呼ばれる
def switch_event(channel):
  global leds
  global switches

  for i in range(4):
    if channel == switches[i]:
      if GPIO.input(channel) == GPIO.LOW:
        GPIO.output(leds[i], GPIO.HIGH)
      else:
        GPIO.output(leds[i], GPIO.LOW)

# プログラムはここから
if __name__ == "__main__":

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

  for i in range(4):
    # イベントを登録する
    GPIO.add_event_detect(switches[i], GPIO.BOTH,
bouncetime=60)
    GPIO.add_event_callback(switches[i], switch_event)

  try:
    # 1時間何もしない
    time.sleep(3600)

  except KeyboardInterrupt:
    GPIO.cleanup(switches)
    GPIO.cleanup(leds)
    sys.exit()
```

GPIO割り込みを使おう

先のプログラムでは、スイッチが接続されている GPIO を永久ループで読み取り続けました。スイッチと LED という単純なプログラムならそれでも対応できますが、少し複雑なプログラムになると常にスイッチを読み続ける処理は実施しづらくなります。

RPi.GPIO ライブラリには「GPIO のステータスに変化が起きたら何かをする」というイベント処理を実現する「GPIO 割り込み」をサポートしています。この機能を使えば、スイッチを延々と読み取り続ける必要がなくなり、スイッチに変化が起きたときだけ、それに対応するプログラムが作成できます。

そのサンプルプログラム(switch2.py)が図 12 です。switch2.py を実行すると、switch1.py と同じ動作をします。

```
$ python switch2.py
```

ただし、スイッチを早く押したりすると LED が点灯しなかったり消灯しなかったりするかもしれません。その理由は後述します。

GPIO へのイベントは次のように登録します。

```
GPIO.add_event_detect(GPIO番号, トリガー, bouncetime =m 秒)

GPIO.add_event_callback(GPIO番号, コールバック関数)
```

「GPIO.add_event_detect()」で指定した GPIO 番号にイベントを登録します。イベントの引き金(トリガー)として表 2 の三つが登録できます。switch2.py では「GPIO.BOTH」を指定しているので、スイッチが押されたか離されたときにイベントが発生します(31 行目)。

パラメーターとして「bouncetime」を設定しますが、これはスイッチに付きものの「チャタリング」を避けるための処置です。メカニカル接点を使ったスイッチは、スイッチがオンになるとき、あるいはオフになるときに μ 秒ほどの時間でオンとオフを激しく繰り返してしまう現象を起こします。これがチャタリングです。

人間にとっては μ 秒ほどの時間は認識できませんが、コンピュータにとっては有意な時間です。チャタリングへの対応を怠ると後述するコールバック関数が短時間に繰り返し呼び出されるという不都合が生じます。

GPIO.add_event_detect() で bouncetime パラーターに m 秒単位の時間を設定すると、その時間だけオンとオフの繰り返しを無視します(31 行目)。switch2.py では「60」を設定していますが、これで 60m 秒の間に起きるオンとオフを無視するようになります。

表2　登録できるトリガー

トリガー	内容
GPIO.RISING	GPIOの入力がLOWからHIGHに変化した
GPIO.FALLING	GPIOの入力がHIGHからLOWに変化した
GPIO.BOTH	GPIOの入力がLOWからHIGHまたはHIGHからLOWに変化した

難しいのは bouncetime に設定する時間で、これはスイッチの種類や品質、あるいはスイッチの用途によって調整する必要があります。bouncetime の値は読者自身でいろいろと変更して動作を理解し、確かめてほしい点です。

なお、この bouncetime が設定されているため、前述のように LED が点灯しなかったり消灯しなかったりすることがある点が switch1.py と異なるわけです。

GPIO.add_event_detect() でイベントを設定した GPIO に対しては、「GPIO.add_event_callback()」でイベント発生時に呼び出す関数を設定できます。スイッチが押されるか離されるときに、switch2.py では「switch_event()」という関数が呼び出されます(32 行目)。

コールバック関数にはイベントが起きた GPIO 番号が引数として渡されます。switch_event() 関数内では、イベントが起きた GPIO 番号を調べて対応する LED を点灯または消灯させています(11 ～ 20 行目)。

圧電スピーカーを使おう

続いてはラズパイ入門ボード上の圧電スピーカーを使ってみましょう。スピーカーは GPIO18 につながっています。なので、先に作成した entryboard.py の末尾に次の 1 行を追加してください。

```
BUZZER=18
```

スピーカーから音を出すには、スピーカーを振動させなければなりません。つまりスピーカーに対してオンとオフを短時間に繰り返す信号を与えます。

ラズパイでは、GPIO に対して設定した周期でオン(HIGH)、オフ(LOW)を与える「PWM」(Pulse Width Modulation)がサポートされています。この PWM を使うとスピーカーを振動させて音を出すことが簡単にできます。

RPi.GPIOのPWM機能を使用する

まずは、これまで利用してきた RPi.GPIO に実装されている PWM の機能を使って圧電スピーカーを鳴らしてみます。サンプルプログラム(buzzer1.py)は図 13 です。

buzzer1.py を実行すると、2 秒間、スピーカーから「ピー」と音が鳴ります。音の周波数は約 1kHz です。

図13　圧電スピーカーを2秒間鳴らすプログラム　(buzzer1.py)

```
 1 # -*- coding: utf-8 -*-
 2 from entryboard import *
 3 import time
 4 import RPi.GPIO as GPIO
 5
 6 GPIO.setmode(GPIO.BCM)
 7 GPIO.setup(BUZZER, GPIO.OUT)
 8 bz = GPIO.PWM(BUZZER, 1000)
 9
10 bz.start(50)
11 time.sleep(2)
12 bz.stop()
13 GPIO.cleanup(BUZZER)
```

```
$ python buzzer1.py
```

GPIO に PWM を設定するのが「GPIO.PWM()」です(8 行目)。

```
bz = GPIO.PWM(GPIO番号, 周波数)
```

周波数は Hz 単位で指定します。switch1.py では「1000」Hz(＝1kHz)を設定しています。GPIO.PWM() は「PWM」オブジェクトを返します。

PWM は次のように出力します(10 行目)。

```
bz.start(デューティー比)
```

「デューティー比」というのは、指定した周期（ここでは 1kHz）のうちオンになっている時間の割合で、パーセント（%）で設定します。swtich1.py では 50% を設定しているので、オンとオフが同じ時間となり 1kHz の音が出ます。

ハードウエアPWMを使用する

実際に buzzer1.py を実行すると分かりますが、1kHz といっても安定した音程で出力されません。例えば「ピプピー」のように聞こえるでしょう。

これは RPi.GPIO の PWM 機能がソフトウエア的に実装されているためです。オンとオフの繰り返しをソフトウエアのみで実現しているので正確な 1kHz にはならず、音の高さが安定しないという結果になります。

ラズパイはハードウエアによる PWM 機能を備えています。これを「ハードウエア PWM」と呼びますが、ハードウエア PWM を使うと正確な PWM 出力が可能になります。

ただし、ハードウエア PWM は、GPIO19 と GPIO18 でしか使えません。また、RPi.GPIO はハードウエア PWM をサポートしていませんので、別のライブラリを使う必要があります。本特集では、Python からも使える高性能な「pigpio」ライブラリでハードウエア PWM を実現します。

pigpio ライブラリー式は Raspbian にあらかじめ導入されていますが、次のように念のため更新しておいてください。

```
$ sudo apt-get update
$ sudo apt-get install pigpio python-pigpio python3-p
igpio
```

ハードウエア PWM を使って 1kHz の音を 2 秒間、出力するプログラム（buzzer2.py）は図 14 です。pigpio はクライアント / サーバー式になっているため、buzzer2.py を入力しただけでは実行できません。実行前に pigpio サーバー（pigpiod）を常駐させておく必要があります。次のように root 権限で常駐させてください ＊1。

```
$ sudo pigpiod
```

図14　ハードウエアPWMを使って音を2秒間鳴らす

```
 1 # -*- coding: utf-8 -*-
 2 from entryboard import *
 3 import time
 4 import pigpio
 5
 6 gpio = pigpio.pi()
 7 gpio.set_mode(BUZZER, pigpio.OUTPUT)
 8 gpio.hardware_PWM(BUZZER, 1000, 500000) # 1kHz, 50%
 9 time.sleep(2)
10 gpio.hardware_PWM(BUZZER,0 ,0)
11 gpio.stop()
```

これで pigpio サーバーが常駐しました。その上で次のように実行してください。先ほどと異なり、クリアーなピーというと音がスピーカーから出力されます。

```
$ python buzzer2.py
```

pigpio を使用する場合、まずサーバーに接続して pigpio オブジェクトを得る必要があります（6 行目）。これで gpio に pigpio オブジェクト格納されます。

```
gpio = pigpio.pi()
```

「gpio.set_mode()」は、RPi.GPIO の GPIO.setmode() に相当する関数で、GPIO の入出力モードを設定します（6 ～ 7 行目）。ハードウエア PWM で GPIO を初期化して出力するのが「gpio.hardware_PWM()」です（8 行目）。

＊1　pigpioサーバーを停止しないと、Raspbianがシャットダウンできませんでした。シャットダウン時には「sudo kill -9 pigpiod」を実行します。

図15　簡易オルガンのプログラム　（buzzer3.py）

```
 1 # -*- coding: utf-8 -*-
 2 from entryboard import *
 3 import time
 4 import pigpio
 5
 6 # 1オクターブの音階周波数リスト
 7 scale = [ 523.251, 587.33, 659.255, 698.456,783.991,
880, 987.767, 1046.502 ]
 8 switches = [SW1, SW2, SW3, SW4]
 9
10 bouncetime = 5 *1000 # 5ms
11 prev_tick = 0
12
13 # コールバック
14 def switch_event(channel, edge, tick):
15   global bouncetime
16   global prev_tick
17   global scale
18   global switches
19
20   if (tick-prev_tick) < bouncetime:
21     return
22   prev_tick = tick
23
24   for i in range(4):
25     if switches[i] == channel:
26       if edge == pigpio.LOW:
27         gpio.hardware_PWM(BUZZER, scale[i], 500000)
28       else:
29         gpio.hardware_PWM(BUZZER, 0, 0)
30
31 if __name__ == "__main__":
32   gpio = pigpio.pi()
33   gpio.set_mode(BUZZER, pigpio.OUTPUT)
34   for i in range(4):35
35     # モードとコールバック
36     gpio.set_mode(switches[i], pigpio.INPUT)
37     gpio.callback(switches[i], pigpio.EITHER_EDGE, s
witch_event)
38   try:
39     time.sleep(3600)
40   except KeyboardInterrupt:
41     gpio.hardware_PWM(BUZZER, 0, 0)
42     gpio.stop()
```

表3　登録できるトリガー

トリガー	内容
EITHER_EDGE	GPIOがHIGHまたはLOWになった
FALLING_EDGE	GPIOがHIGHからLOWになった
RISING_EDGE	GPIOがLOWからHIGHになった

```
gpio.hardware_PWM(GPIO番号,周波数,デューティー比)
```

pigpio ではデューティー比としてパーセントを 1 万倍した値で設定します。また、RPi.GPIO とは異なり、gpio.hardware_PWM() を呼び出すとすぐに PWM 信号が GPIO から出力します。 停止させるには、

```
gpio.hardware_PWM(GPIO番号,0 ,0)
```

を呼び出します(10 行目)。最後に、pigpio サーバーとの接続を終了させます(11 行目)。

```
gpio.stop()
```

簡易オルガンを作る

正確でクリアーな音が出せるようになりました。そこで、4 個のスイッチを使った簡易オルガンを作ってみます。スイッチが押されたらドレミファが鳴るものです。プログラム(buzzer3.py)は図 15 です。

buzzer3.py を実行してみてください。SW1 ～ SW4 のスイッチを順番に押すとドレミファが鳴るでしょう。たった 4 音ですが一応オルガンとして機能します。

buzzer3.py では、GPIO の制御に pigpio を使ってみました。スイッチのオンとオフについては前節で RPi.GPIO を使った例を取り上げており、pigpio と RPi.GPIO を混在させることもできるのですが、二つの同じようなライブラリを混在させるのは余り見た目がよくないので、pigpio で統一してみたわけです。

pigpio でも GPIO のステータス変化に対してコールバック関数を登録できます(37 行目)。イベントには表 3 のトリガーが指定できます。

```
gpio.callback(GPIO番号,イベント, コールバック関数)
```

buzzer3.py では「EITHER_EDGE」を指定しているので、スイッチが押されたときと離されたときにコールバック関数が呼び出されます。

コールバック関数に渡されるパラメーターは次のようになっています(14 行目)。

```
def switch_event(GPIO番号,ステータス,システムティック)
```

GPIO 番号はイベントが発生した GPIO の番号です。ステータスは現在の GPIO の入力値で、「pigpio.LOW」または「pigpio.HIGH」です。

最後のシステムティックは、イベントが発生したシステム時刻がμ秒で格納されています。pigpio には RPi.GPIO で利用した bouncetime の設定に相当する機能がないので、スイッチのチャタリングを避けるには、コールバック関数に渡されるシステムティックを用いる必要があります。

具体的には、一つ前のイベントのシステムティックを保存しておき、現在のシステムティックと比較して極端に短い時間しか経過していなければチャタリングと判断します。

buzzer3.py では、「prev_tick」変数に一つ前のシステムティックを保存して、システムティックから prev_tick を引き、5m 秒(bouncetime)以内ならチャタリングとして無視しています(20 ～ 21 行目)。bouncetime を 5m 秒に設定したのは、オルガンとして切れのよい反応が必要だからです(10 行目)。

なお、システムティックは 32 ビットの値なので、約 72 分でオーバーフローして「0」に戻ってしまう点に注意が必要です。buzzer3.py では特に対策していませんが、場合によってはオーバーフロー対策が必要になります。

なお、スイッチが 4 個しかないのでドレミファまでしか音が出ませんが、音階リスト(scale)には 1 オクターブ分の周波数を記しておきました。コードを書き換え、音を変えるなどして楽しんでみてください。

図16　デューティー比の変更

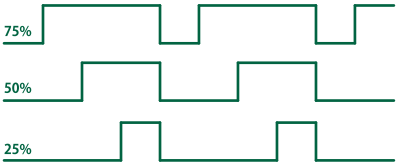


図17　カラーLEDの定義を追記

```
LED5_R = 20
LED5_G = 22
LED5_B = 27

LED6_R = 26
LED6_G = 19
LED6_B = 13
```

図18　スイッチでカラーLED(LED6)の色を変えるプログラム
(color_led.py)
最初はRGB3色が強く光っているので、SW4を押しながら減光するとよい。

```
 1 # -*- coding: utf-8 -*-
 2 from entryboard import *
 3 import time
 4 import RPi.GPIO as GPIO
 5
 6 led6 = [LED6_R, LED6_G, LED6_B]
 7 duty = [50.0, 50.0, 50.0]
 8 switches = [SW1, SW2, SW3, SW4]
 9 pwm = []
10
11 GPIO.setmode(GPIO.BCM)
12 GPIO.setup(led6, GPIO.OUT)
13 GPIO.setup(switches, GPIO.IN)
14
15 for i in range(3):
16   # 100Hzに設定
17   pwm.append(GPIO.PWM(led6[i], 100))
18   pwm[i].start(duty[i])
19 try:
20   while True:
21     for i in range(3):
22       if GPIO.input(switches[i]) == GPIO.LOW:
23         if GPIO.input(SW4) == GPIO.LOW:
24           duty[i] = duty[i] - 5.0
25           if duty[i] < 0.0:
26             duty[i] = 0.0
27           else:
28             duty[i] = duty[i] + 5.0
29             if duty[i] > 100.0:
30               duty[i] = 100.0
31       pwm[i].ChangeDutyCycle(duty[i])
32       time.sleep(0.1)
33 except KeyboardInterrupt:
34   GPIO.cleanup()
```

表4　カラーLEDとGPIOの対応

LED	色	GPIO番号
LED5	赤	GPIO20
	緑	GPIO22
	青	GPIO27
LED6	赤	GPIO26
	緑	GPIO19
	青	GPIO13

カラーLEDを使おう

最後にカラー LED を制御してみましょう。カラー LED には、内部に「赤」「緑」「青」(RGB)の LED が組み込まれています。光の三原色を混ぜてカラーを表現しているわけです。RGB3 色の LED をそれぞれ点灯または消灯させるだけだと 7 色のカラーしか表現できませんが、光の割合を変えることでフルカラーを表現できます。

どうやって光の割合を変えるかというと、前節の PWM を用います。前節の音ではデューティー比 50%を使いましたが、図 16 のようにデューティー比を変えることによってオンの時間を任意に設定できます。LED の場合、オンの時間が長いほど明るく見え、逆にオンの時間が短いほど暗く見えます。

LED が点滅しているわけですが十分に高速に点滅させればちらつきとして認識することはありません。実際、蛍光灯は 1 秒間に 50 または 60 回(50Hz または 60Hz)で点滅していますが、日常生活でちらつきを認識することはめったにないでしょう。また PWM を使った輝度調節は、LED 照明やディスプレイのバックライト＊2 など身の回りに機器に広く利用されています。

カラー LED の RGB それぞれに PWM を与え、デューティー比を変えれば自由な色の設定が可能なフルカラーとして使えるわけです。

＊2　フリッカーフリーのバックライトはPWMを使わずに電流で輝度を制御しています。PWMで輝度を下げるとオフ時間が長くなり、ディスプレイ表示と干渉するとちらつきが発生し目の疲れの原因になると言われているためフリッカーフリーが増えてきています。

スイッチでLEDの色を変える

4 個のスイッチで三原色のデューティー比率を変えて，任意の色に調節できる LED のプログラムを作ってみましょう。LED が接続されている GPIO は表 4 の通りです。ここまでと同様に、entryboard.py に図 17 の 6 行を追加します。

ここでは PWM に RPi.GPIO を使うことにします。前述の通り、RPi.GPIO では正確な周波数が得られませんが、音と違って視覚は点滅に対して鈍感ですから周波数が厳密である必要はまったくありません。

また、ちらつきが認識できない周波数はおおよそ 50Hz 以上です。50Hz を割り込むとチラチラすると感じる人が多くなります。ならば高い周波数を設定すればよいではないかと思うかもしれませんが、LED の点灯と消灯は残光があるため、あまり高速にはできません。PWM の周波数を上げすぎると LED がほぼ常時点灯しっぱなしの状態になってしまい、デューティー比による明るさの変化が得られなくなります。一般に 50 ～ 100Hz の間を設定しておくのが安全でしょう。

それらを考慮したサンプルプログラム(colorled.py)は図 18 のようになります。colorled.py を実行してみてください。

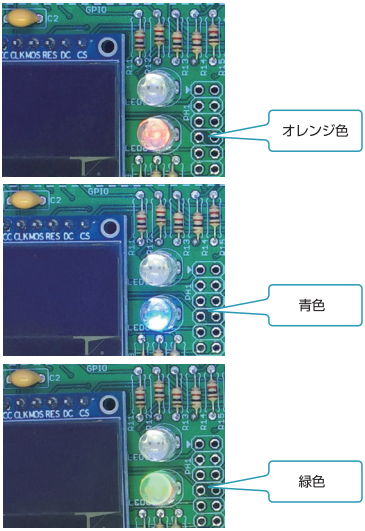
```
$ python colorled.py
```

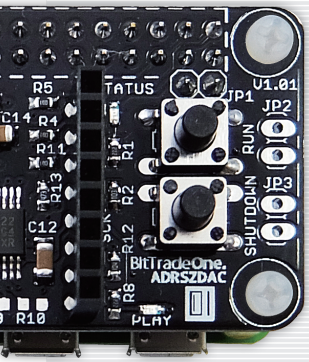
SW1、SW2、SW3 が LED6 の赤青緑に対応し、押すとそれぞれの成分が強くなります。また、SW4 と合わせて押すと、それぞれの成分が減っていきます。このように四つのスイッチを使って、LED6 の色を自在に変更できるでしょう(図 19)。

「ChangeDutyCycle()」にパーセントで比率を渡すと、PWM のデューティー比を変えることができます(31 行目)。注意が必要なのは、ChangeDutyCycle() を呼び出した後、100ms 程度のディレイが必要な点です。ChangeDutyCycle() を十分なディレイなしに呼び出してしまうと、デューティー比が正しく切り替わらず連続点灯になってしまいます。これは RPi.GPIO の仕様のようです。

colorled.py では、SW1 ～ SW4 を読み取り、その押下状態に応じて 3 色のデューティー比を増減し、ChangeDutyCycle() で切り替え、呼び出すたびに 100m 秒(0.1 秒)のディレイを入れています。

図19　colorled.pyの実行結果
四つのスイッチでLED6の色が変わる。



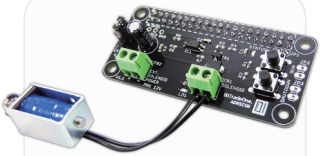


01 ゼロワン

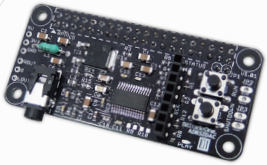
- 単機能の拡張基板
- pHAT サイズ
- 基板上から電源操作
- ライブラリ公開中

ラズパイゼロに必要な機能を一つだけ追加する拡張基板 "ゼロワン" シリーズ

単一機能を使い易くコンパクトに。[ゼロをイチに] あなたもまずは1つの機能の追加から初めてみませんか？



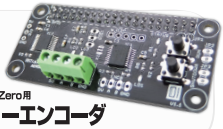
RaspberryPi Zero用
ソレノイド拡張基板
ADRSZSN



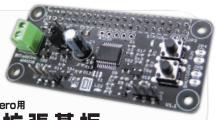
RaspberryPi Zero用
ハイレゾDAC拡張基板
ADRSZDAC

RaspberryPi Zero用
焦電型赤外線センサ
拡張基板 ADRSZPY

RaspberryPi Zero用
温湿度・気圧センサ
拡張基板 ADRSZBM



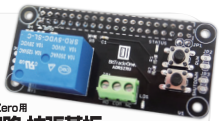
RaspberryPi Zero用
ロータリーエンコーダ
拡張基板 ADRSZRE



RaspberryPi Zero用
サーボ拡張基板
ADRSZSB

RaspberryPi Zero用
赤外線受信 拡張基板
ADRSZIRR

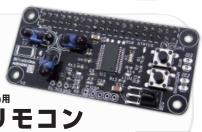
RaspberryPi Zero用
明るさセンサ 拡張基板
ADRSZLX



RaspberryPi Zero用
リレー回路 拡張基板
ADRSZRU

RaspberryPi Zero用
照光スイッチ 拡張基板
ADRSZSW

RaspberryPi Zero用
9軸センサ拡張基板
ADRSZGR

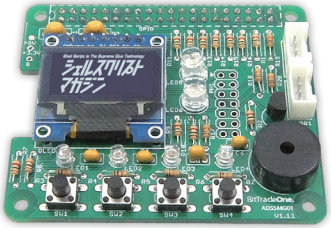


RaspberryPi Zero用
赤外線リモコン
拡張基板 ADRSZIRS

RaspberryPi Zero用
OLED 拡張基板
ADRSZLD

キット版ユーザーのためにラズパイ入門ボードの作り方を解説！はんだごてを握ってレッツ電子工作！

■ キット作成の前に〜内容物と道具の確認



キット作成の前に、
・半田セット (はんだごて、はんだなど)
・ニッパー

完成写真を確認しましょう。
背の低い部品から順におこなうと作業が楽になります。

■部品表の確認

以下の部品表と併せ内容物の確認を行ってください。部品表 2 を取り付ける前に、部品表1の部品をすべて取り付けて　ください。

◆ **部品表1**

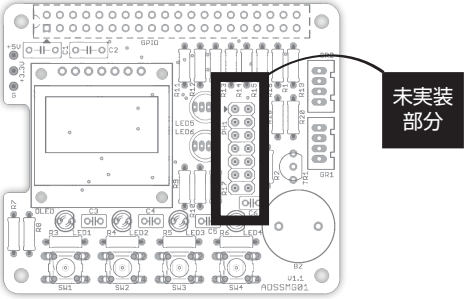
部品名	個数	部品番号	備考
基板	1	---	上下方向は印字ロゴを参照してください
カーボン抵抗 10kΩ	8	R3〜6,R17〜20	茶黒橙金
カーボン抵抗 1kΩ	12	R1〜2,R7〜16	茶黒赤金
リードコンデンサー1uF50V	2	C1,C2	---
リードコンデンサー0.1uF10V	4	C3〜6	---
LEDリード赤	4	LED1〜4	取り付け向き有り／透明
フルカラーLED	2	LED5〜6	取り付け向き有り
トランジスタ	1	TR1	取り付け向き有り

◆ **部品表2**

部品名	個数	部品番号	備考
圧電スピーカー	1	BZ	半田面から取り付ける
Groveコネクター	2	GR1〜2	取り付け向き有り／
モジュール(7ピン)	1	OLED	取り付け向き有り／R7,R8,R9取り付け後に取り付けます
ピンソケット	1	GPIO	半田面から取り付ける
タクトスイッチ 黒	4	SW1〜4	---

■ 基板シルクと部品番号の確認

部品表の部品番号と対比して基板上に各部品の配置が記載されています。シルク印字と部品番号に対応個所に部品を搭載します。



2. ラズパイ入門ボードを作ってみよう（2. 部品を順番に取り付けよう）

基板への半田付けは背の低い部品から順に行うと作業が楽になります。下記の表に従って計 20 本ある抵抗から取り付けていきましょう。

カーボン抵抗 10kΩ

取付箇所
R3～6,R17～20

キット版には10kΩと1kΩの2種類の抵抗があります。見た目が似ているので混在しないよう気を付けて下さい。

カーボン抵抗 1kΩ

取付箇所
R1,2,R7～16

キット版には10kΩと1kΩの2種類の抵抗があります。見た目が似ているので混在しないよう気を付けて下さい。

リードコンデンサ 1μF (105)(大)

取付箇所
C1,2

リードコンデンサに取り付け方向はありません。

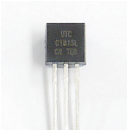
リードコンデンサ 0.1μF (104)(小)

取付箇所
C3～6

リードコンデンサに取り付け方向はありません。

トランジスタ

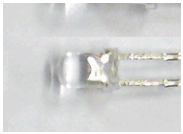
取付箇所
TR1 (C1815)



基板に印刷されているシルクとトランジスタの形が同じになるように取り付けます。

LEDリード赤

取付箇所
LED1～4



キット版には10kΩと1kΩの2種類の抵抗があります。見た目が似ているので混在しないよう気を付けて下さい。

フルカラーLED

取付箇所
LED5,6

キット版には10kΩと1kΩの2種類の抵抗があります。見た目が似ているので混在しないよう気を付けて下さい。

圧電スピーカー

取付箇所
BZ

スピーカー上に"+"（プラス）の記号が印字されています。この印字とシルク文字の方向が合うように取り付けて下さい。

Groveコネクター

取付箇所
GR1～2

2本の切込みが基板の外側を向くように取り付けてください。

モジュール（7ピン）

取付箇所
OLED

基板上シルクに従い取り付けてください。液晶は割れ易いので力をいれないよう気を付けて作業を行ってください。

ピンソケット

取付箇所
GPIO

今まで取り付けてきた部品がある面から半田付けを行います。正確に取り付けないと接続先のラズパイのGPIOピンが曲がってしまう等の悪影響を及ぼす可能性があります。慎重に取り付けて下さい。

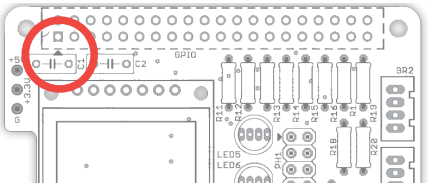
タクトスイッチ 黒

取付箇所
SW1～4

取り付け向き等は有りません。そのまま取り付けて下さい。

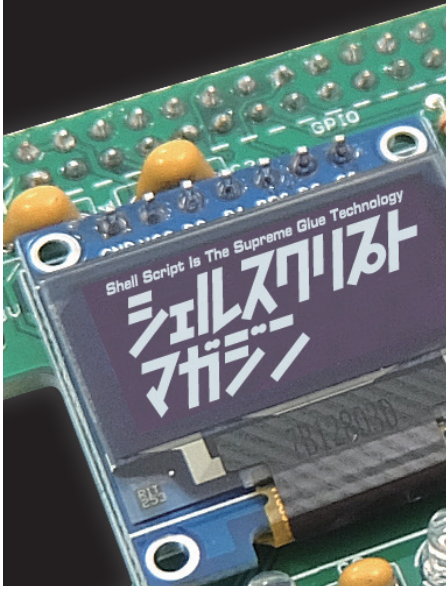
■ ラズベリーパイとの接続

ラズベリー・パイの GPIO1 番ピンと
ラズパイ入門ボードのシルク上の▲を合わせるように接続します。
付属のネジセットを使いラズパイ入門ボードと
ラズベリー・パイを固定してください。



以上で組み立て終了となります。お疲れ様でした。

超入門!! 学んでおきたい制御方法
基礎の基礎



■奥付

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
印刷・製本
　Gr@phic
発行日
　2018/10/05

K. 拉麺喰い師伝説

解ってないな。
ラーメンなんかな、列作って食べるもんじゃないんだよ。客がいるんだかいないんだか解らない店の、こ汚い暖簾をくぐって座るなりおばちゃんに「お姉さん、ビールと餃子。」もちろん生じゃなく瓶だ、この所作がわかってない18歳未満のお子様が多くて困るわ。
ビールをつぐグラスはもちろん冷タン。洗いが雑なため泡立ちはボチボチだ。新聞を読む店の親父がカッタルそうに立ち上がって餃子の準備をするところから戦いは始まっている。まずは注目すべきはその店のラインナップだ。ゴルゴ13か浮浪雲があれば合格。
ゴルゴ13は 14 巻の " 日本人・東研作 " あたりを流し読みするのがプロのラーメン喰いだ。半ばまで読んだあたりで餃子が来るはずだが、ここで大事なのはその視点、決して餃子を目で追うな。ゴルゴに集中しテーブルにコトリと置かれた餃子の音で、さも今気づいたというような目線を姉さんに送れ。
さて餃子が来たが、ここで最初に醤油をとったド素人は実戦ではすでに死んでいるので多いに反省すべきだ。まずは酢。あくまで酢を主体に醤油で風味をつけるわけだ。あとは辣油か辛子をつけるわけだが辛子がある場合には醤油の前に辛子を溶くことを忘れないように辛子は醤油では溶けづらい。
餃子は一口で食え。半分かじるなんて喰い方は、おんな子どもの喰い方だ。多少熱くてもビールがある。
イメージとしては餃子をビールで流し込む。もちろん熱いそぶりなんてこれっぽっちも見せずに食え。これはなかなか難しいので普段からイメージトレーニングを行っておくこと。餃子を半分喰ったあたりで、おもむろに「チャーハン一つ」オーダーは姉さんにかけるが目線はあくまで店の親父だ。
ここまでの所作で親父も自分が相手をしてる人間がド素人でないことは十分に解っているはず、まずは挑戦状を送ろう。
ここで初心者は大盛りにすべきかどうか迷うかもしれない。が大盛りを完食することがその店と自分との戦い。大盛りは " 生き方 " だ。
店とメニューで選ぶことではない。大盛りを選ぶ十字架を背負った生き方を貫くなら、行く店すべて大盛りが基本だ。潰なら時と場所で生き方をチャラチャラ変えるもんじゃない。さて本題に戻ろう。すでに厨房では炒飯のオーダーを受けた親父が中華鍋を振るっていると思う。中華鍋上で踊る米と鍋のリズミカルな動き、小刻みに動く親父の肩、スネアドラムのように入る味の素他、調味料、親父のパフォーマンスの見せ所だ。ここだけはお互い戦いを忘れ鑑賞につとめよう。前半、終了間際、中華鍋を振るう親父がお玉に炒飯を受ける箇所が最大の見どころだ、マスタークラスなら3回の動作でお玉への収納を完了し皿の上に見事な炒飯ドームを形成しているはず。最後のフィナーレを飾るのは中華鍋を叩く「チンチン」というお玉の音色。もちろん鍋の中に米の一粒も残っていない。さて先に餃子半分といったがこれは、基準値としてのタイマーだ。経験豊富な店なら残り半分の餃子が胃袋に消えるあたりで炒飯がやってくるだろうから、ビールと餃子どちらも欠けることなくスムーズに炒飯にバトンタッチしたい。言うまでもないことだが基本の「流れる水のように」の所作を心がけるように初心者はこれらのタイミングが中ン掴めず、途中でメンマをオーダーしたりあろうことかビールを追加注文したりしてしまうが、真にプロのラーメン喰いを目指すのであれば必須のスキルなので、焦らず身につけていってもらいたい。さあ炒飯だ。目の前に炒飯が到着したらゴルゴはできるだけスープが到着する前に置き、まずは炒飯のドーム形成を鑑賞しよう。米たちが潰れることなくお玉形状に整形されているか、周辺に余計な米粒が散乱していないか、米の焼き目は均等か?卵に焼き目はついていないか?食す前に見ただけで判断できることは多い。今、あなたが見ている炒飯が店の人となりを表している。
炒めるにあたってスピードがなければ卵に焼き目がついてしまうし、急げば米に均等に焼き目をつけることが難しい。
強大な火力を制御する鉄の意思と技術がなければ不格好な炒飯もどきが出来上がるだけだ。以上の所見が確認できれば後は、目の前にある炒飯を胃袋に収めるだけだが、ここで再度状況を確認しておこう。炒飯が提供されたことで今まで防戦一方だった親父が攻勢に転じる時だ。
親父がタバコを吸いながら「どうだ?」という体でこちらを見ているはず。一呼吸置いて肉を食うように一気に食え!
レンゲの動きはできるだけ止めず、小刻みに肩を震わせリズミカルに先ほどの親父の調理をトレースするかのように口元に運び、合いの手のようにスープもいただこう。
脳内 BGM はアートブレイキーとジャズ・メッセンジャーズの So Tired だ。
一気に炒飯を食べ終わったら一呼吸置き最後に残ったビールを飲み干す、ここで煙草を取り出す輩がいるが、長居は無用。辛抱し姉さんを呼び「ごちそうさん」だ。
姉さんの「はい!お釣り350万エ〜〜ン」のコールを聞き暖簾をくぐれば午後3時の日差しの中、爽やかな風が頬をなでる。
さあ仕事を探そう。

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⇒⇒⇒

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