2020

GPIO (Raspberry pi, Arduino)

S.Matoike

目次

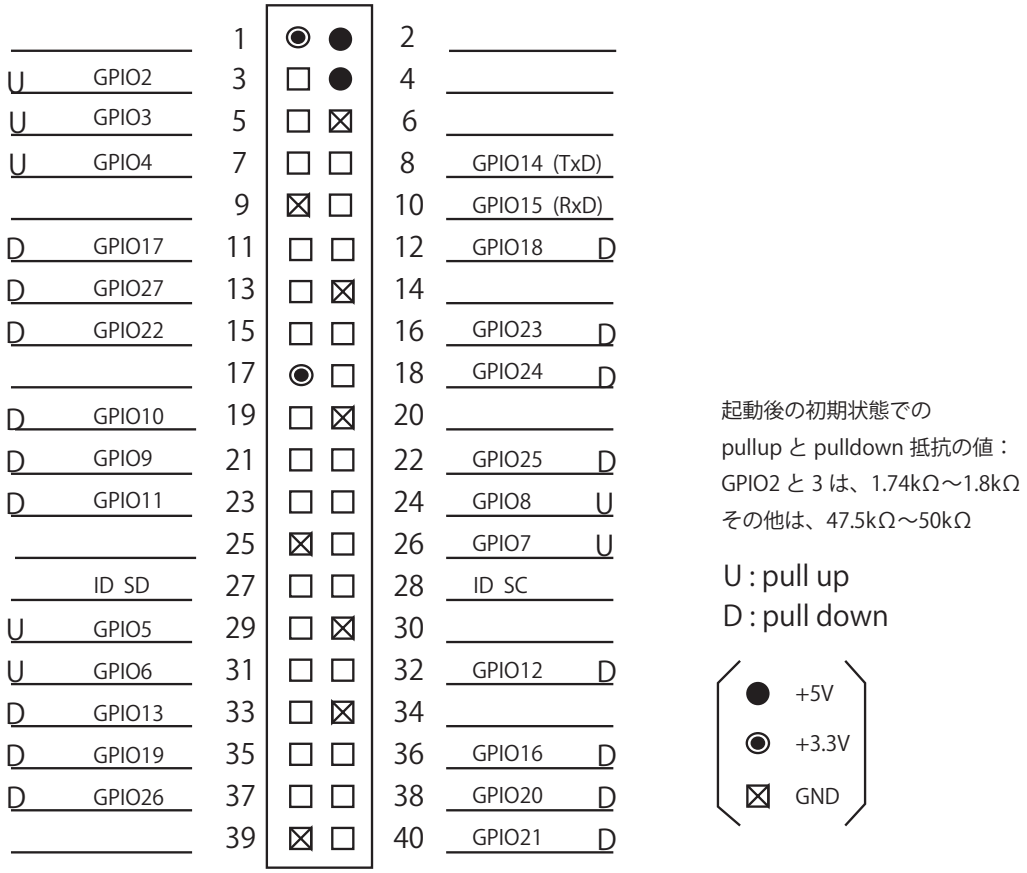
第 1 章

はじめに

第 2 章

GPIO

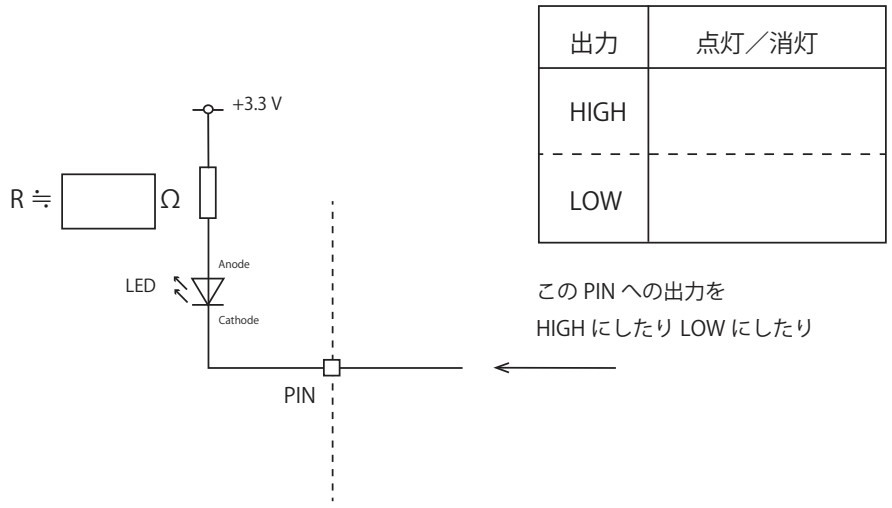
2.1　Raspberry pi



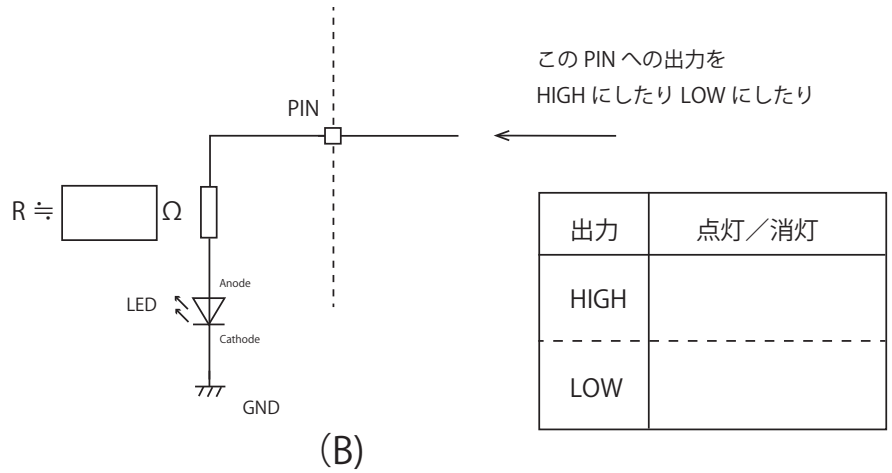
第 3 章

デジタル入出力

3.1　LED（出力）



（A）



（B）

※注意：GPIO の各ピンには、全てプルアップあるいはプロダウン抵抗が内蔵されているので、外付けの LED のための電流制限抵抗は無くても動作します

【演習 1】次のことを確認してみよう（PIN 番号は各自で任意に選択すること）

(1) 配布された外付け用の電流制限抵抗が何 Ω かテスターで調べてみよう
(2) Raspberry Pi の PIN で提供されている電源 3.3V と GND を、ブレッドボード上の赤と青のライン上に引き出し、実際の電圧をテスターで調べてみよう
(3) (A) の結線の場合には、GPIO の PIN に HIGH(=1) を出力した時に LED は消灯し、LOW(=0) を出力した時に LED は点灯することをプログラムで確認しよう
(4) (B) の結線の場合には、GPIO の PIN に HIGH(=1) を出力した時に LED は点灯し、LOW(=0) を出力した時に LED は消灯することをプログラムで確認しよう

ソースコード 3.1　LED(C 言語)

```
1  #include <stdio.h>
2  #include <wiringPi.h>
3  #define PIN 17
4  void main(void) {
5     if( wiringPiSetupGpio() == -1) return;
6     pinMode(PIN, OUTPUT);
7     int a;
8     do{
9         printf("1 or 0 : ");
10        scanf("%d", &a);
11        if ( a==1 )
12            digitalWrite(PIN, 1);
13        else if ( a==0 )
14            digitalWrite(PIN, 0);
15        else
16            break;
17    } while( TRUE );
18    digitalWrite(PIN, 0);
19 }
```

ソースコード 3.2　LED(Python)

```
1  import RPi.GPIO as GPIO
2
3  PIN = 17
4  GPIO.setwarnings(False)
5  GPIO.setmode(GPIO.BCM)
6  GPIO.setup(PIN, GPIO.OUT)
7
8  while True:
9      str = input('1 or 0 : ')
10     a = int( str )
11     if a==1:
12         GPIO.output(PIN, GPIO.HIGH)
13     elif a==0:
14         GPIO.output(PIN, GPIO.LOW)
15     else:
16         break
17
18 GPIO.cleanup()
19 print('\n 終わり!')
```

C 言語のプログラムでは、ソースプログラムを led01.c という名称のテキストファイルとして保存し、以下のように gcc コマンドでコンパイルして実行します

```
$ gcc -o led01 led01.c -I/usr/local/include -L/usr/local/lib -ℓwiringPi
$ ./led01
```

物理的なピン配置の番号で PIN を指定する場合には、
C 言語では、wiringPiSetupGpio(void); に代えて、wiringPiSetupPhys(void); を使います。
Python では、GPIO.setmode(GPIO.BCM) を、GPIO.setmode(GPIO.BOARD) に代えます。
【演習 2】演習 1 のプログラムを、GPIO 番号から物理的 PIN 配置番号に変更して動作を確認しよう

3.1.1　シェル（コマンド）による GPIO の操作

wiringPi が使えるようになっているか否かを、次のようにバージョンを表示させて確認します

```
$ gpio -v
gpio version: 2.50
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty
Raspberry Pi Details:
.....
```

wiringPi を使うと、プログラムを書かなくても、次のようにコマンドを打つだけで GPIO を ON/OFF することができます

```
$ gpio -g mode 17 out
$ gpio -g write 17 1
$ gpio -g write 17 0
```

$-g$ という指定は、ピン番号に GPIO のピン番号を使う場合に指定します（この例では GPIO17）
この指定をしなかった場合のデフォルトは、物理的な配置番号（コネクタの番号）になります
bash のシェルスクリプトにまとめるとすると、次のように書くことができます

ソースコード 3.3　LED(shell)

```
1  #!/bin/bash
2  set -euo pipefail
3  LED_PIN = 23
4  gpio export $LED_PIN out
5  trap 'gpio unexport $LED_PIN' SIGINT
6  while true
7  do
8      gpio -g write $LED_PIN 1
9      sleep 1
10     gpio -g write $LED_PIN 0
11     sleep 1
12 done
```

これを、led01.sh という名前のテキストファイルに保存し、chmod コマンドを使ってファイルに実行権限を与え、その上で実行します
「ls -l」として、ファイルの属性を表示させてみると、実行権限が与えられたことを確認できます
ファイルに実行権限を与えない場合でも、「sh ./led01.sh」としてシェルを実行させることも可能です

```
$ chmod +x ./led01.sh
$ ./led01.sh
```

（参考：bash のオプション）

set -e： 実行されたコマンドの 1 つが 0 でないステータスで終了した場合、スクリプトを終了させ、エラーシェルスクリプト実行時に実行内容が表示させるようにするオプション

set -u： 未定義変数が参照されたときに処理を終了します（ーの代わりに＋を指定すると意味は逆になります）

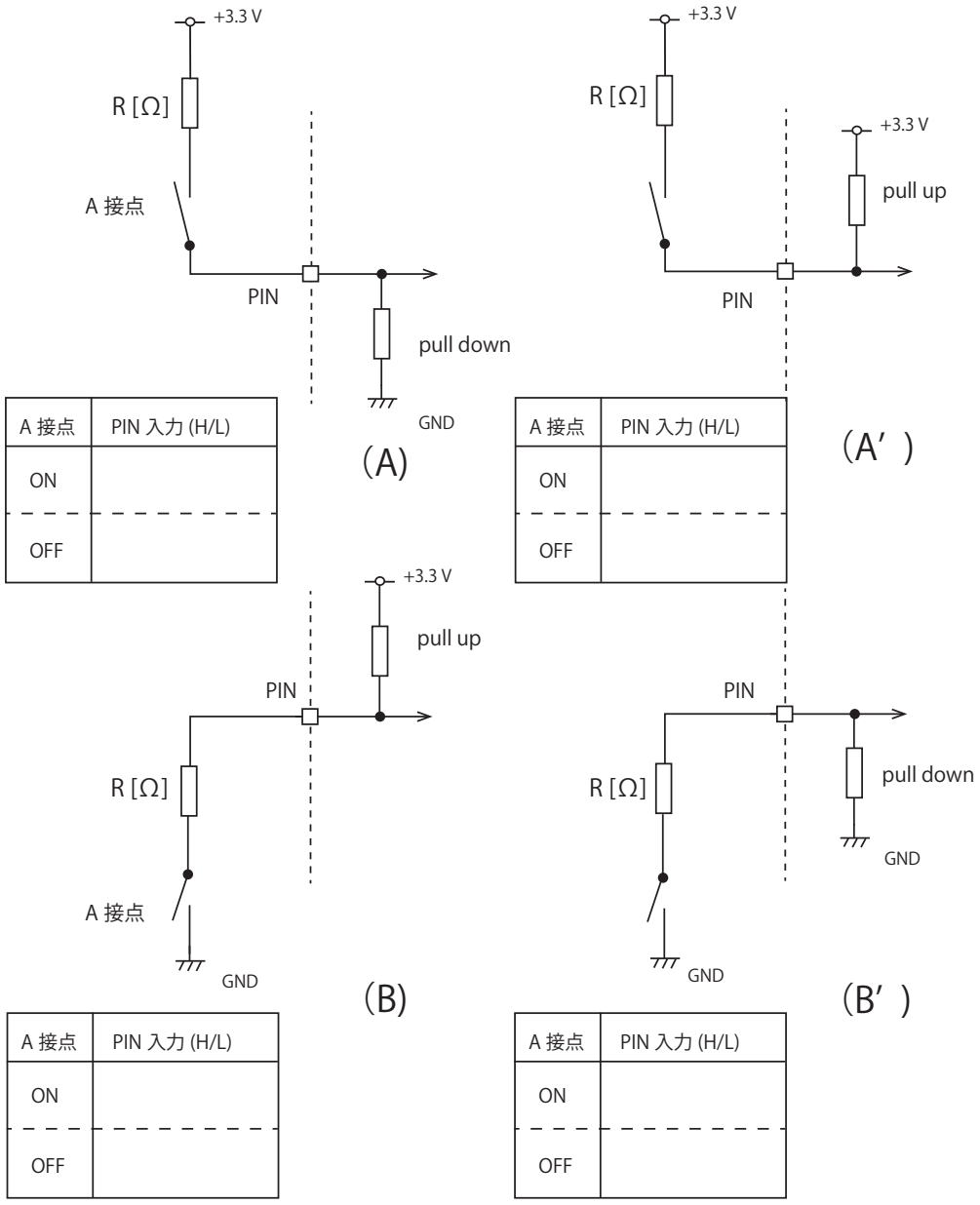
set -o： 続けて pipefail を指定します。パイプでつないだ各コマンドの中で、終了ステータスが 0（正常終了）以外だった場合に、最後に 0 以外だったコマンドの終了ステータスが返されます

set -n： スクリプト自体は実行されず、文法チェックのみが行われる（問題なければ何も出力されません）

set -v： シェルスクリプト内でこれから実行されるコマンドを表示します。これは、スクリプト内に記述されているコマンドが出力されるだけなので、実際の使用に当たっては、「-x」オプションと一緒に指定して、実行する内容と、実行された内容を表示させるようにして使います

set -x： デバッグ情報の出力指示です。シェルスプリプと実行の際に、変数への代入や実行されたコマンドなど、シェルスクリプト内で処理された内容が表示されます

3.2　(Limit)Switch（入力）



A 接点	PIN 入力 (H/L)
ON	
OFF	

A 接点	PIN 入力 (H/L)
ON	
OFF	

A 接点	PIN 入力 (H/L)
ON	
OFF	

A 接点	PIN 入力 (H/L)
ON	
OFF	

※注意：GPIO の各ピンには、全てプルアップあるいはプロダウン抵抗が内蔵されているので、外付けの抵抗 R[Ω] は無くても動作します

GPIO の PIN を入力（INPUT）のモードで使うときには、プルアップ（PUD_UP）あるいはプルダウン（PUD_DOWN）を指定して、PIN の初期状態を切り替えることができます（なお GPIO の PIN を出力（OUTPUT）のモードで使うときには、この指定はできません）

（A’) の結線では、Switch の状態とは無関係に、PIN の状態が LOW(=0) になることはありません

（B’) の結線では、Switch の状態とは無関係に、PIN の状態が HIGH(=1) になることはありません

つまり、（A’) や（B’) の結線では Switch の状態を読むことはできません（このような使い方は無い）

【演習 3】次のことを確認してみよう（PIN 番号は各自で任意に選択すること）

(1) 配布した Limit Switch には端子が 3 本あります。A 接点と B 接点が内蔵されているのですが、どの端子とどの端子を使えば A 接点になるのか、どの端子とどの端子を使えば B 接点になるのか、テスターで調べてみましょう
(2) (A) の結線の様に PIN をプルダウンしている場合、A 接点が ON なら PIN の入力は HIGH(=1) に、A 接点が OFF なら PIN の入力は LOW(=0) になることをプログラムで確認しよう
(3) (B) の結線の様に PIN をプルアップしている場合、A 接点が ON なら PIN の入力は LOW(=0) に、A 接点が OFF なら PIN の入力は HIGH(=1) になることをプログラムで確認しよう
(4) (A') や (B') の結線では Switch の状態を読みとれないことを、プログラムで確認しましょう

ソースコード 3.4　Switch(C 言語)

```
1  #include <stdio.h>
2  #include <wiringPi.h>
3
4  #define PIN 27
5
6  void main(void) {
7     if( wiringPiSetupGpio() == -1)
          return;
8     pinMode(PIN, INPUT);
9     pullUpDnControl(PIN, PUD_UP);
10
11    do{
12       int a = digitalRead(PIN);
13       if ( a==1 )
14          printf("%d:Switch ON\n", a);
15       else
16          printf("%d:Switch OFF\n", a);
17       delay(1000);
18    } while( TRUE );
19 }
```

ソースコード 3.5　Switch(Python)

```
1  import RPi.GPIO as GPIO
2  from time import sleep
3
4  PIN = 27
5  GPIO.setwarnings(False)
6  GPIO.setmode(GPIO.BCM)
7  GPIO.setup(PIN, GPIO.IN,\
8             pull_up_down=GPIO.PUD_UP)
9  try:
10   while True:
11     a = GPIO.input(PIN)
12     if a==1:
13        print( str(a) + ':Switch ON' )
14     else:
15        print( str(a) + ':Switch OFF' )
16     sleep(1.0)
17 except KeyboardInterrupt:
18     pass
19 finally:
20     GPIO.cleanup()
```

wiringPi を使うと、特別なプログラムを書かずに、次のようにコマンドを並べるだけで GPIO から PIN の値を読み取ることができます

```
$ gpio -g mode 3 in
$ gpio -g read 3
```

bash のシェルスクリプトにこれらのコマンドをまとめると、次のようにしてプログラムを書いた場合と同じようにしてシェルスクリプトを使うことができます

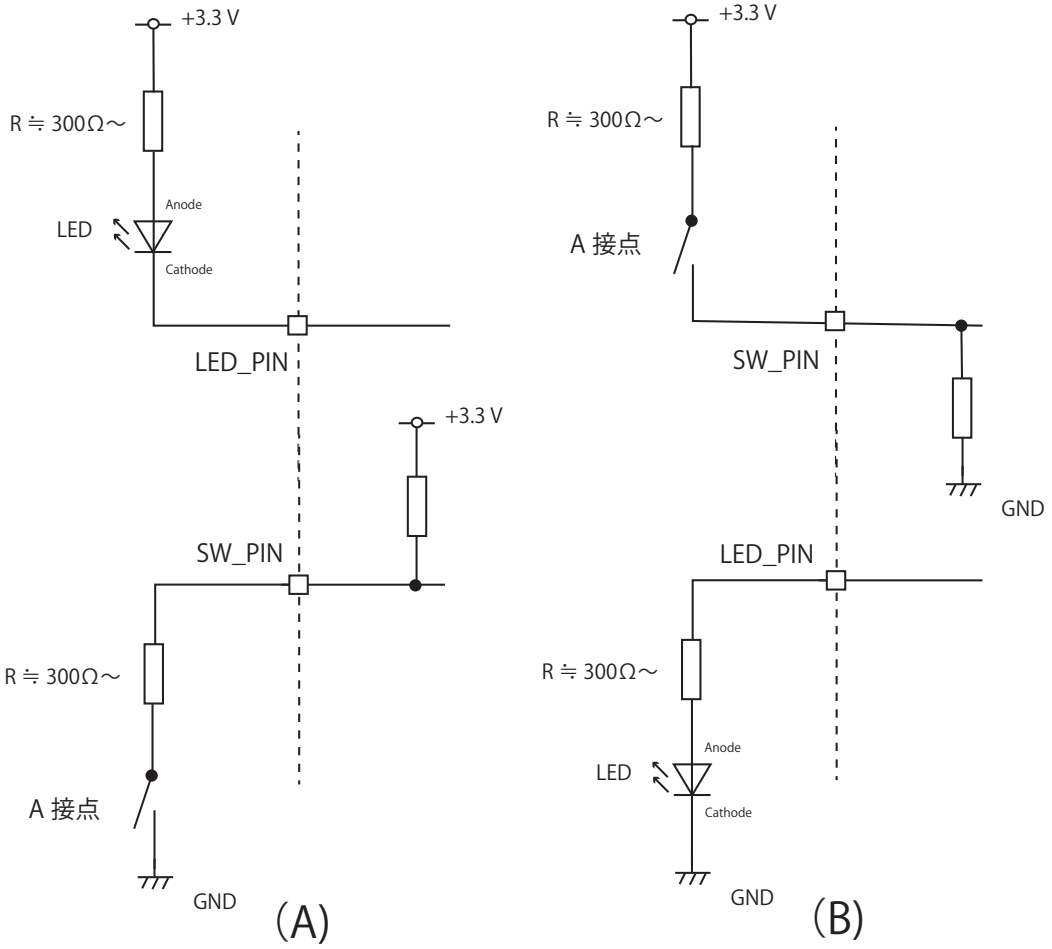
ソースコード 3.6　SW(shell)

```
1  #!/bin/bash
2  set -euo pipefail
3  SW_PIN = 3
4  gpio export $SW_PIN in
5  trap 'gpio unexport $SW_PIN' SIGINT
6  while true
7  do
8     gpio -g read $SW_PIN
9     sleep 1
10 done
```

【演習 4】プログラムを作成して確認しましょう（各 PIN の GPIO 番号は任意に選択すること）

(1)（A）の結線の A 接点が ON の時は LED が点灯し、OFF の時は消灯するプログラム

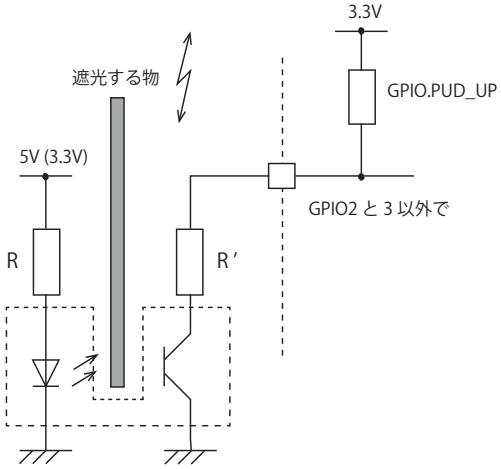
(2)（B）の結線の A 接点が ON の時は LED が点灯し、OFF の時は消灯するプログラム



3.3　Photo interrupter, reflector（入力）

フォトインタラプタは、光を遮光する箇所をリミットスイッチとして使ったり、遮光した回数を数えたりする場面で利用されます

フォトリフレクタは、物体に当てた光の反射を受信検知するもので、物体の通過を知るときなどに使われます。プログラム上での扱い方は、フォトリフレクタやインタラプタ、その他のスイッチでも同じです



注意：抵抗器 R' は、プルアップ抵抗が内蔵されているので必ずしも必要ではありません

動作確認のプログラムはスイッチの時と同様に記述できますが、Arduino 風に書くことに倣うと次のように書くことができます。setup() 関数と loop() 関数を中心に記述していくことができます

ソースコード 3.7　Photo interrupter(Python)

```
1  import RPi.GPIO as GPIO
2  import time
3  PIPin = 4
4  def setup():
5      GPIO.setwarnings( False )
6      GPIO.setmode( GPIO.BCM )
7      GPIO.setup( PIPin, GPIO_IN, pull_up_down=GPIO.PUD_UP )
8
9  def loop():
10     v = GPIO.input( PIPin )
11     print( v )
12     time.sleep( 1.0 )
13
14 if __name__ == '__main__':
15     setup()
16     try:
17         while True:
18             loop()
19     except KeyboardInterrupt:
20         pass
21     finally:
22         GPIO.cleanup()
```

【演習 5】上記プログラムを以下の課題に対応させるように書き換えてみましょう

(1) 演習 4 の回路（A) のスイッチをフォトインタラプタに代えて、遮光時に LED を点灯させる動作
(2) 遮光した回数をカウントし、その回数が 5 の倍数の時に LED を 5 秒間点灯させる動作
(3) C 言語でも動作を確認してみましょう

3.4　3.3V と 5V の世界の橋渡し

3.4.1　Photo coupler（出力・入力）

3.4.2　CMOS と TTL

3.4.3　Interrupt(割り込み)

ソースコード 3.8　Interrupt(Python)

```
1  import RPi.GPIO as GPIO
2
3  PIN1 = 23
4  PIN2 = 24
5  GPIO.setwarnings(False)
6  GPIO.setmode(GPIO.BCM)
7  GPIO.setup(PIN1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
8  GPIO.setup(PIN2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
9
10 def myCallback(channel):
11     print(str(PIN1) + ' was pushed:' + str(channel))
12
13 GPIO.add_event_detect(PIN2, GPIO.RISING, callback=myCallback, bouncetime=300)
14
15 try:
16     print('waiting for falling edge on port ' + str(PIN1))
17     GPIO.wait_for_edge(PIN1, GPIO.FALLING) # FALLING, RISING, BOTH
18     print(str(PIN1) + ' was pressed')
19 except KeyboardInterrupt:
20     pass
21 finally:
22     GPIO.cleanup()
```

【演習 6】演習 4 の課題を使って、次のことを試してしてみましょう

(1) 演習 4 の課題そのままを、割り込みを使って実現する処理
(2)（A）又は（B) の結線で、A 接点を ON（直後に OFF）にした時に LED が点灯し、再度 ON（直後に OFF）とした時に消灯する処理

3.4.4　チャタリング対策

第 4 章

おわりに

謝辞

参考文献

[1]