PythonをRaspberry PiのIO制御

Raspberry PiでのIO制御

Raspberry PiにはIO（GPIO：General-purpose input/output）ピンがヘッダによって引き出されており，入出力の制御を行うことができる.

GPIOピンを利用することで，外部デバイスのON/OFFを取得・制御したり，外部のセンサやデバイスとI2CやSPI，UARTなどのプロトコルを用いて通信を行うことができる.

今回はRaspberry PiからGPIOを利用して，外部に接続したLEDの制御を行うためのプログラムの実装を行う.

> GPIOのON/OFF制御はハードウェア（CPU）に近いレイヤーで実現する機能であるため，CPUのほかMMU，チップセットなどを備えた基盤上で稼働するOSからアクセスすることは非常に難しく，一般的なPCでは実現しにくいアプリケーションである.
> （最近では見かけなくなったが）パラレルポートなどがIO制御を行うためのインターフェイスとしてかつて存在していた

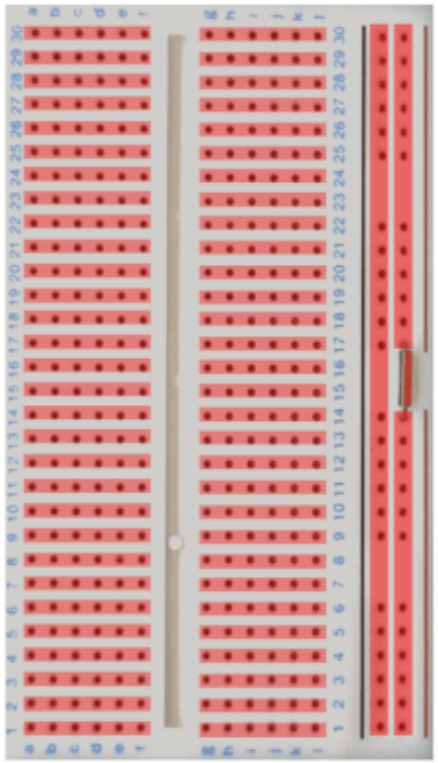
GPIO制御によるLチカ

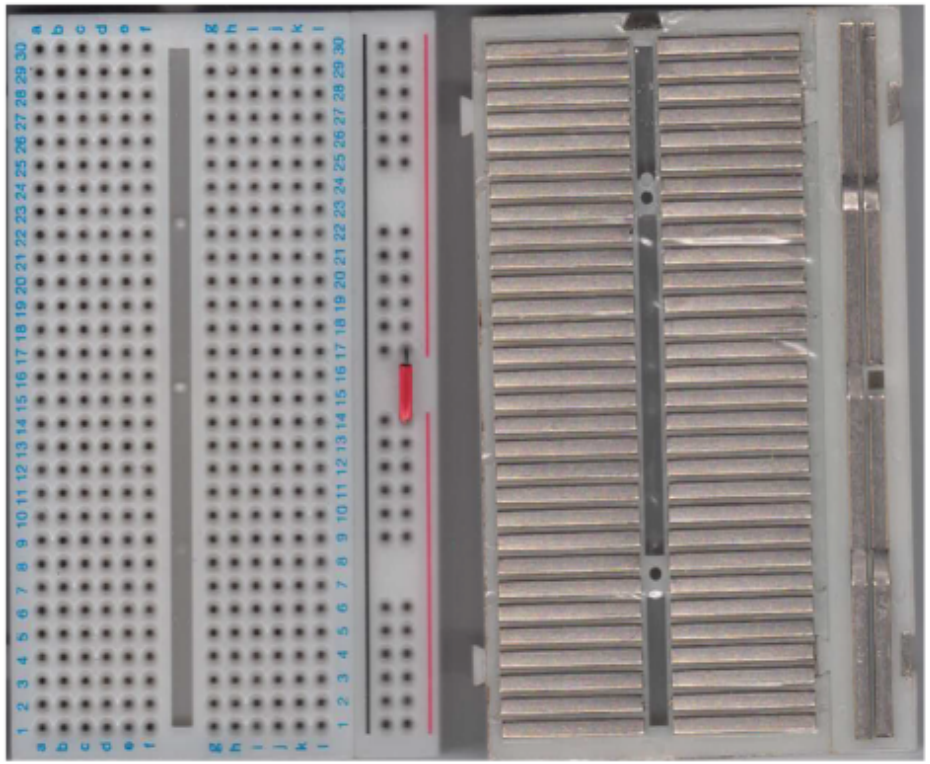
> Lチカ：
> LEDをつないで点滅させる作業のこと．新型のマイコンなどを購入した場合などにIO制御のコードを実装する練習としてよく行う．

Raspberry Pi本体とLEDを配線したブレッドボードを接続し，プログラムを用いて点灯させる．

ブレッドボードの仕組み配線

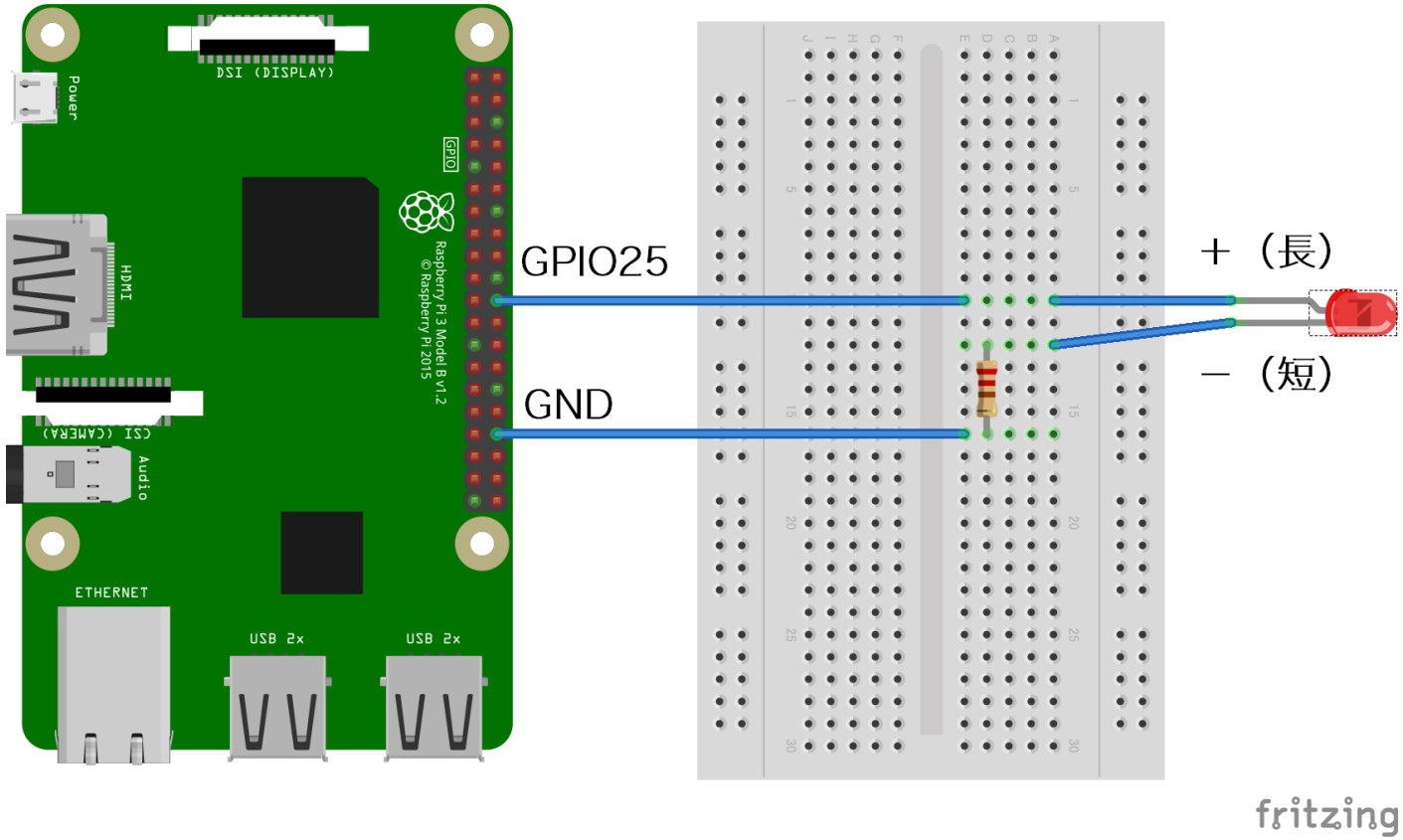
ブレッドボードは表面にジャンパ線やピンを挿すことによって容易に回路を構成する事ができるキットのこと．プロトタイピングなどの際に回路を簡単に構成するために用いられる．
（裏面を見ると導体の入り方がわかる）



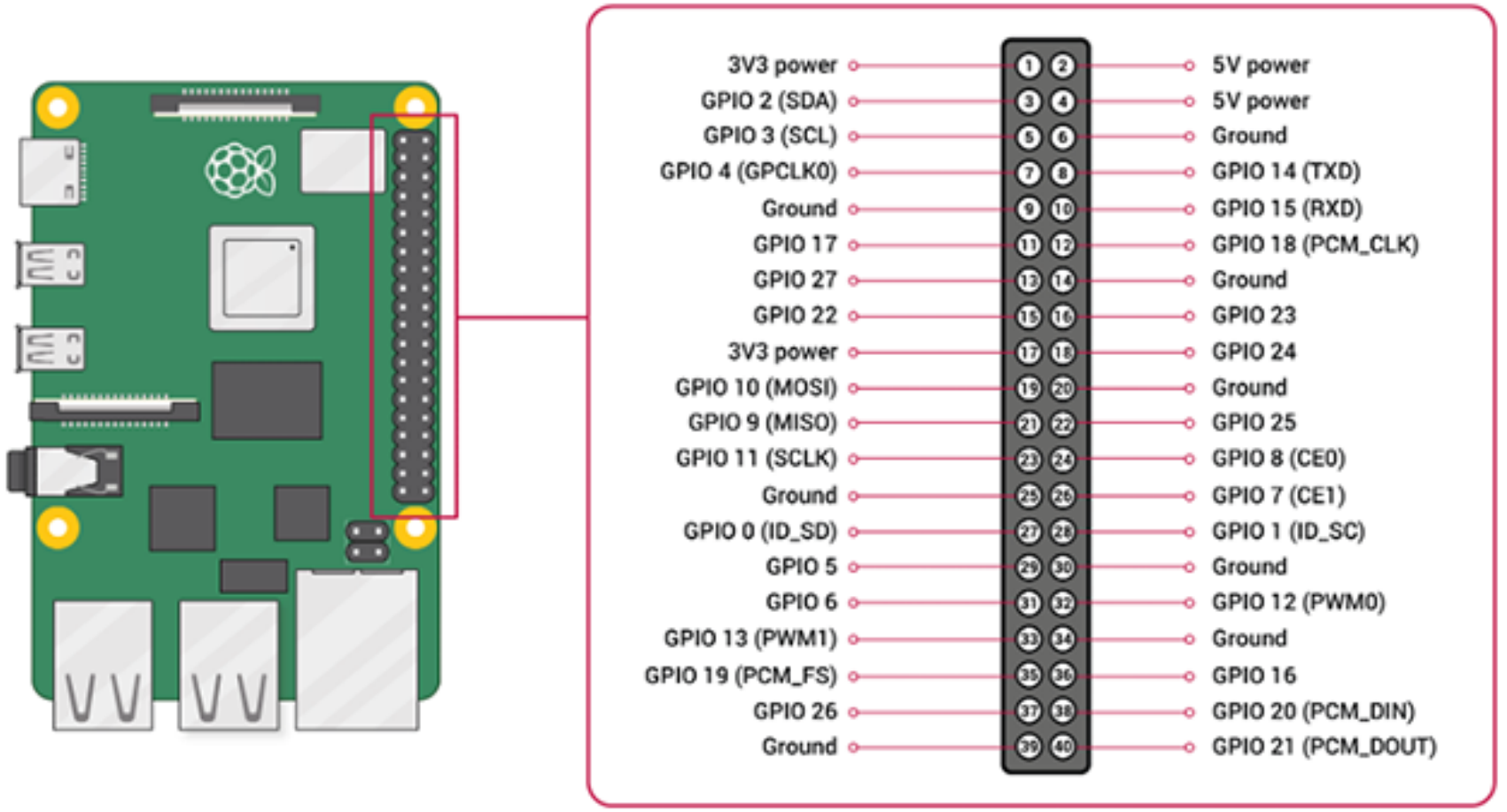


Raspberry Piとブレッドボードの接続

図の通り結線する



【参考】Raspberry PiのGPIOピンアサイン



GPIO出力制御プログラム

```
import RPi.GPIO as GPIO
from time import sleep

if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(25, GPIO.OUT)     #GPIO25を出力に設定
    try:
        while True:
            GPIO.output(25, GPIO.HIGH)   # Highを0.5s出力
            sleep(0.5)
            GPIO.output(25, GPIO.LOW)    # Lowを0.5s出力
            sleep(0.5)
    except KeyboardInterrupt:
        # KeyboardInterruptをハンドリングして
        # 終了時にGPIO.cleanup()が実行されるようにする.
        pass
    GPIO.cleanup()
```

GPIO出力制御プログラムの解説

GPIO25を出力に設定し，Highを0.5S，Lowを0.5s交互に出力することで，22番ピンに接続されたLEDが点滅する.

プログラム実行後，`Ctrl+C`でアプリケーションを停止することを考慮して`try/except`文を用いて`KeyboardInterrupt`をハンドリングして，終了時に必ず`GPIO.cleanup()`が実行されるように設計している.

GPIOのその他の応用

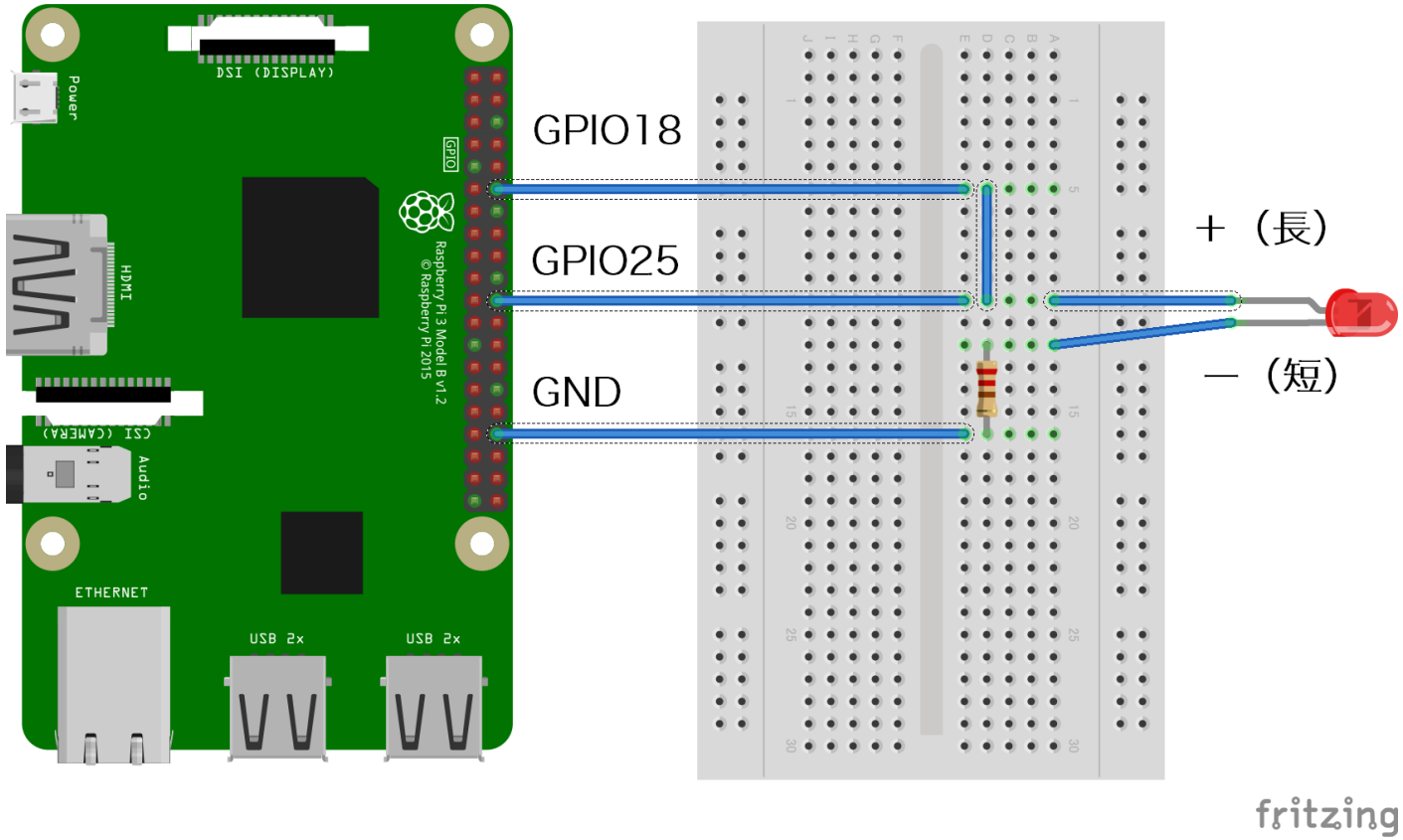
今回はGPIOピンを出力に設定して外部の機器を駆動させたが，入力に設定して外部機器のステータスを取得することも可能．入力ピンへの入力電圧がVcc（3.3V）を超えると本体が故障するため注意が必要となる．

また，ポートから出力可能な電流はすべてのポートをあわせて50mAであり，モータなどの電流を要する負荷は直接接続することができないため，モータドライバなどを用いる必要がある．
（駆動電圧の異なる負荷はリレーユニットやフォトカプラ，ソリッドステートリレーを噛ませる）

またパルス幅変調などの（PWM:Pulse Width Modulation）機能も利用できるのでモータの速度やLEDの照度調整を行う際に利用できる．

GPIO入力制御プログラム

GPIO18を入力として利用する．（今回はGPIO25からの出力をそのまま入力にする）



GPIO制御プログラム

以下の２つの実装方法がある.

（１）入力状態を常時監視（取得）

入力状態監視ループの中で，常に何かしらの処理が行われる.

（２）イベントドリブン

入力状態遷移があった場合のみ処理を行う.

GPIO制御プログラム

（1）入力状態を常時監視（取得）

```
import time
import RPi.GPIO as GPIO

if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP) #GPIO18を入力に設定
    try:
        while True:
            sw_status = GPIO.input(18) #GPIO18の状態取得
            if sw_status == 0:
                print('GPIO18 ON!')
            time.sleep(0.3)
    except KeyboardInterrupt:
        pass
    GPIO.cleanup()
```

（2）イベントドリブン（入力のみ）

```
import time
import RPi.GPIO as GPIO
if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP) #GPIO18を入力に設定
    try:
        while True:
            GPIO.wait_for_edge(18, GPIO.FALLING) #GPIO18押下待ち
            print('GPIO18 ON')
            #time.sleep(0.3) #チャタリング対策（sleepでソフトウェア的に解決するより，HWで解決したほうが○）
    except KeyboardInterrupt:
        pass
    GPIO.cleanup()
```

【参考】アプリケーションが終了できない場合

端末上でプログラムを実行後に，`Ctrl+C`を入力してもプログラムを終了できない場合は下記の方法で終了します.

下記のようにプログラムを実行したとします

```
$ python3 gpio_input.py
```

別の端末を開いて`ps -aux | grep python3`コマンドを実行し，現在Raspberry Pi上で実行中のpythonプログラムの一覧を表示する.

例）

```
$ ps -aux | grep python3
pi         768  0.0  0.6  52088 27200 ?        S    10:17   0:01 /usr/bin/python3 /usr/share/system-config-printer/applet.py
pi       17338  0.2  0.1  14796  7628 pts/0    S+   15:15   0:00 python3 gpio_input.py
pi       17402  0.0  0.0   7452   584 pts/1    S+   15:16   0:00 grep --color=auto python3
```

出力されたリストは

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

の順に並んでいるので，COMMANDから実行済みの `python3 gpio_input.py` を探してPIDを調べる．PIDをを調べた上で，下記のコマンドを実行する．

```
$ kill -9 [調べたPID]
```

今回の例ではPIDは `17338` であったので下記に示すコマンドとなる．

```
$ kill -9 17338
```

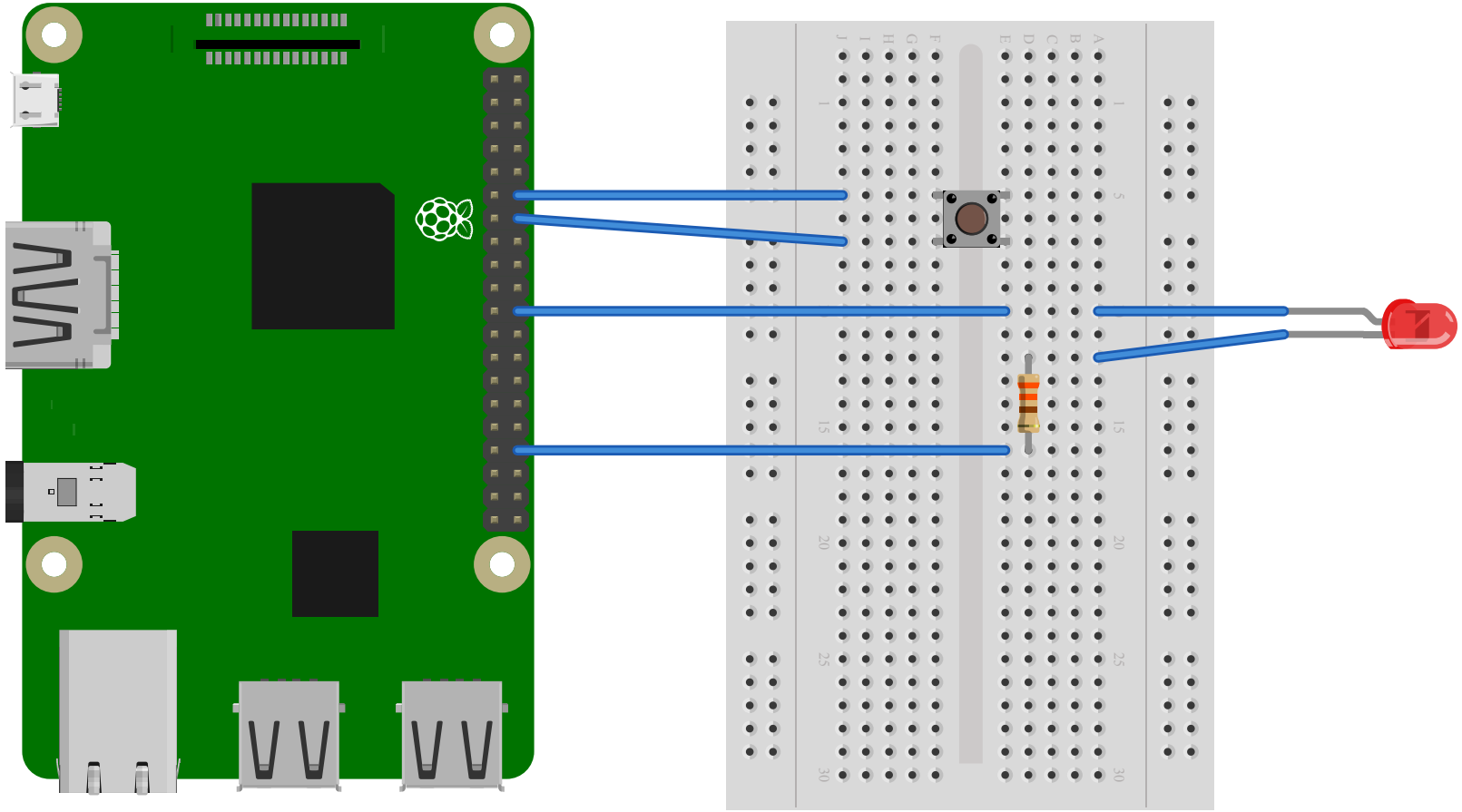
上記の `kill` コマンド実行後，`Ctrl+C` を入力してもプログラムが終了しない端末側に戻ると，`Killed` と表示されて再度操作ができるようになっている．

出力したポート状態を入力ポートで確認する

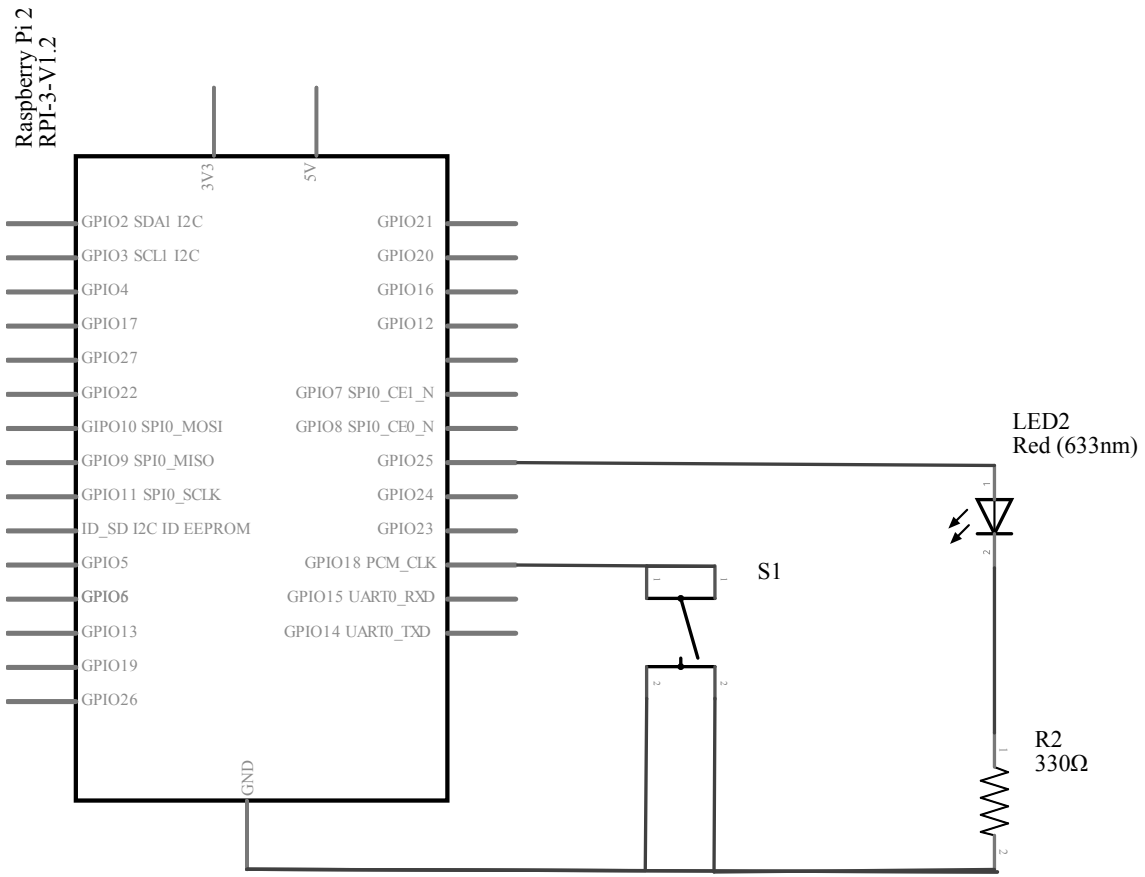
```
import RPi.GPIO as GPIO
from time import sleep
if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(25, GPIO.OUT)     #GPIO25を出力に設定
    GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP) #GPIO18を入力に設定
    try:
        while True:
            GPIO.output(25, GPIO.HIGH)   # Highを0.5s出力
            sw_status = GPIO.input(18) # GPIO18状態取得
            print('sw_status:', sw_status)
            sleep(0.5)
            GPIO.output(25, GPIO.LOW)    # Lowを0.5s出力
            sw_status = GPIO.input(18) # GPIO18状態取得
            print('sw_status:', sw_status)
            sleep(0.5)
    except KeyboardInterrupt:
        pass
    GPIO.cleanup()
```

スイッチのON／OFF状態を取得する

ブレッドボードの配線



回路図



サンプルコード（イベントドリブン）

```
import time
import datetime
import RPi.GPIO as GPIO
if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP) #GPIO18を入力に設定

    while True:
        GPIO.wait_for_edge(18, GPIO.FALLING) #スイッチ押下待ち
        print(datetime.datetime.strftime(datetime.now(), "%Y-%m-%d %H:%M:%S')'スイッチON!')

        time.sleep(0.3) #チャタリング対策
```