

序文

SunFounderについて

SunFounderはロボット用のソフトウェアプラットフォーム、開発ボード、STEAMキット、モジュール、ツール、その他の世界中に配布しているスマートデバイスなどの製品のSTEAM(科学・技術・工学・数学)教育に焦点を合わせている会社である。SunFounderは、STEAM教育を通じて、小学生や中学生、愛好家を支援し、実践能力と問題解決能力を強化するよう努めている。このように、知識を広め、喜びに満ちた方法でスキルトレーニングを提供することより、プログラミングと創作への趣味を促進させ、あなたを科学と工学の魅力的な世界にさらけ出すことを望んでいる。人工知能の未来を受け入れるために、豊富なSTEAM知識を学ぶことは非常に重要なことである。

Da Vinci キットについて

この Da Vinci キットは、Raspberry Pi 世代 4 モデル B、世代 3 モデル A+、世代 3 モデル B+、世代 3 モデル B、世代 2 モデル B、世代 1 モデル B+、世代 1 モデル A+、ゼロ W およびゼロに適用される。さまざまな興味深い現象を作成するのに役立つさまざまな部品とチップが含まれており、実験の指示に従って操作すると達成できる。このプロセスでは、プログラミングに関する基本的な知識を習得できる。また、より多くのアプリケーションを自分で探索することもできる。さあ、未知の世界を探そう！

無料サポート



技術的な質問がある場合は、当社のウェブサイトの掲示板にトピックを追加してください。できるだけ早く返信させていただく。



注文や発送の問題などの非技術的な質問については、直接に [service @ sunfounder.com](mailto:service@sunfounder.com) にメールを送信してください。貴方が掲示板でプロジェクトを共有することを期待している。

Contents

部品一覧	1
前書き	5
何が必要なのか?	6
必要な部品	6
準備	8
モニターをお持ちの場合	8
モニターを持っていない場合	14
必要な部品	14
書き込みシステム	14
Raspberry Pi をインターネットに接続する	15
SSH を起動する	16
IP アドレスを取得する	16
SSH リモートコントロールを使用する	17
Linux または/ Mac OS X ユーザーの場合	17
Windows ユーザーの場合	19
リモートデスクトップ	20
VNC	20
XRDP	25
ライブラリ	28
RPi.GPIO	28
WiringPi	29
GPIO 拡張ボード	31
コードをダウンロードする	33
1 出力	34

1.1 ディスプレイ	34
1.1.1 Blinking LED	34
1.1.2 RGB LED	46
1.1.3 LED Bar Graph	56
1.1.4 7-segment Display	64
1.1.5 4-Digit 7-Segment Display	74
1.1.6 LED Dot Matrix	88
1.1.7 I2C LCD1602	101
1.2 音声	107
1.2.1 Active Buzzer	107
1.2.2 Passive Buzzer	114
1.3 ドライバー	123
1.3.1 Motor	123
1.3.2 Servo	134
1.3.3 Stepper Motor	143
1.3.4 Relay	157
2 入力	165
2.1 コントローラー	165
2.1.1 Button	165
2.1.2 Slide Switch	173
2.1.3 Tilt Switch	181
2.1.4 Potentiometer	189
2.1.5 Keypad	204
2.1.6 Joystick	218
2.2 センサー	226
2.2.1 Photoresistor	226

2.2.2 Thermistor.....	233
2.2.3 DHT-11.....	243
2.2.4 PIR.....	255
2.2.5 Ultrasonic Sensor Module.....	264
2.2.6 MPU6050 Module.....	273
2.2.7 MFRC522 RFID Module.....	286
3 拡張.....	292
3.1 アプリケーション.....	292
3.1.1 Counting Device.....	292
3.1.2 Welcome.....	298
3.1.3 Reversing Alarm.....	305
3.1.4 Smart Fan.....	317
3.1.5 Battery Indicator.....	324
3.1.6 Motion Control.....	329
3.1.7 Traffic Light.....	335
3.1.8 Overheat Monitor.....	343
3.1.9 Password Lock.....	352
3.1.10 Alarm Bell.....	359
3.1.11 Morse Code Generator.....	367
3.1.12 GAME– Guess Number.....	375
3.1.13 GAME– 10 Second.....	385
3.1.14 GAME– Not Not.....	391
3.2 付録.....	402
3.2.1 I2C 設定.....	402
3.2.2 SPI 設定.....	407

部品一覽

Resistor (220R)

20 pcs



Resistor (1K)

10 pcs



Resistor (10K)

10 pcs



1N4007 Diode

2 pcs



Zener Diode

2 pcs



Green LED

4 pcs



Red LED

10 pcs



Yellow LED

4 pcs



Blue LED

4 pcs



RGB LED

1 pcs



S8050 Transistor

4 pcs



S8550 Transistor

4 pcs



Capacitor 0.1 uF

4 pcs



Capacitor 10uF

4 pcs



Photoresistor

1 pcs



Thermistor

1 pcs



L293D

1 pcs



ADC0834

1 pcs



74HC595

2 pcs



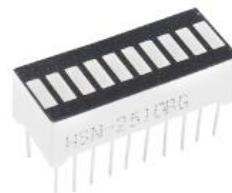
7-segment Display

1 pcs



LED Bar Graph

1 pcs



LED Matrix

1 pcs



Active Buzzer

2 pcs



4-Digit 7-segment Display

1 pcs



Tilt Switch

1 pcs



Potentiometer

3 pcs



Passive Buzzer

1 pcs



Button

4 pcs



Motor

1 pcs



Slide Switch

2 pcs



Keypad

1 pcs



I2C LCD 1602

1 pcs



9G Servo

1 pcs



Breadboard Power Module

1 pcs



MFRC522 RFID Module

1 pcs



Relay

1 pcs



Stepping Motor Driver

1 pcs



Stepping Motor

1 pcs



Ultrasonic Ranging Module

1 pcs



Joystick

1 pcs



Infrared Motion Sensor

1 pcs



DHT-11

1 pcs



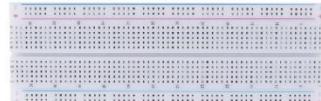
T-shape Extension Board

1 pcs



Breadboard

1 pcs



MPU6050 Module

1 pcs



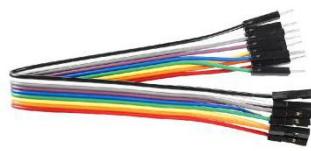
40 Pin GPIO Cable

1 pcs



Jump Wire F/M

10 pcs



9V Battery Cable

1 pcs



Jump Wire F/F

10 pcs



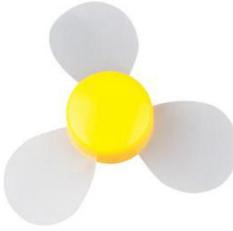
Jump Wire M/M

65 pcs



Fan

1 pcs



ご注意：パッケージを開封した後、部品の数量が製品の説明に一致しているかどうか、およびすべての部品が良好な状態にあるかどうかを確認してください。

前書き

Da Vinci キットはプロジェクトのスケジュールを立てたインテリジェントな初心者に適した基本的なキットである。26 の汎用の入力および出力部品とモジュール、およびプログラミング学習で友好的な支援を提供できる多くの基本的な電子装置（抵抗器、コンデンサなど）が含まれている。

このキットをマスターすることで、Raspberry Pi の実装方法、Bash shell と GPIO についての知識など、Raspberry Pi の基本的な知識を学ぶことができる。これらの知識を理解した上、プログラミングを開始できる。

ハードウェアに関する知識がない場合は、キットに関するこのドキュメントでは、26 回の基本的な I/O レッスンと 4 つの簡単な実例を含む 30 回のレッスンを参照と学習のために提供する。これらのコースの配置は難易度ではなく、実際の機能に基づいていることに注意してください。ニーズに応じて、対応するコースを見つけることができる。つまり、コース全体を読み終えていないく、言及した部品の使用方法を習得していない場合でも、このドキュメントは将来の実用的なプロジェクトを完了させるためのガイドとして重要な役割を果たす。

我々はあなたのプロジェクトを楽しみにしており、このドキュメントを読んでいる間にフォーラムであなたの成果や創造を共有できることを願っている。

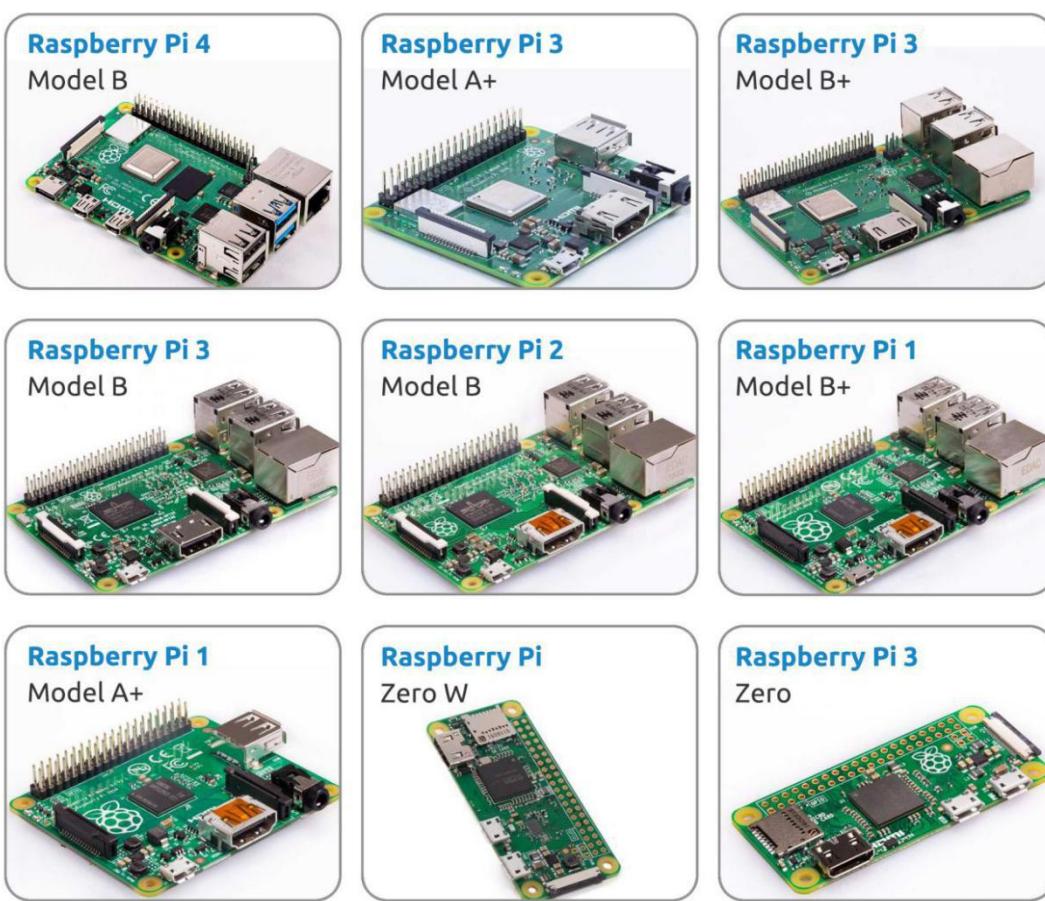
何が必要なのか？

必要な部品

Raspberry Pi

Raspberry Pi は、コンピューター モニターまたはテレビに接続し、標準のキーボードとマウスを使用する、低価格のクレジットカードサイズのコンピューターである。これは、あらゆる年齢の人々がコンピューティングを探求させたり、Scratch や Python などの言語でプログラミングする方法を学習させたりする小さなデバイスである。

このキットは Raspberry Pi 製品の以下のバージョンに適用される：



AC アダプター

Raspberry Pi には、電源ソケットに接続する用のマイクロ USB ポートがある（多くの携帯電話で見られるものと同じである）。2.5A 以上の電源が必要である。

マイクロ SD カード

すべてのファイルと Raspberry Pi OS オペレーティングシステムを保存するため、Raspberry Pi に Micro SD カードを取り付けてください。8GB 以上の容量のマイクロ SD カードが必要である。

選択可能な部品

画面

Raspberry Pi のデスクトップ環境を表示するには、テレビ画面またはコンピューターモニターの画面を使用する必要がある。画面にスピーカーが内蔵されている場合、Pi はそれらを介して音声を再生する。

マウスとキーボード

画面を使用する場合は、USB キーボードと USB マウスも必要である。

HDMI

Raspberry Pi には、ほとんどの最新のテレビおよびコンピューターモニターの HDMI ポートと互換性のある HDMI 出力ポートが搭載されている。画面に DVI または VGA ポートのみがある場合、適切な変換回線を使用してください。

保護ケース

デバイスを保護するために、Raspberry Pi を保護ケースに入れることができます。

サウンドまたはイヤホン

使用できる約 3.5 mm のオーディオポートが装備されている。

準備

使用するさまざまなデバイスに応じて、さまざまな方法で Raspberry Pi を起動できる。Raspberry Pi 用に別の画面がある場合は、この章の指示に従ってください。それ以外の場合は、次の章で対応する手順を見つけてください。

モニターをお持ちの場合

モニターをお持ちの場合は、NOOBS (New Out Of Box System) を使用して Raspberry Pi OS システムを実装できる。

必要な部品

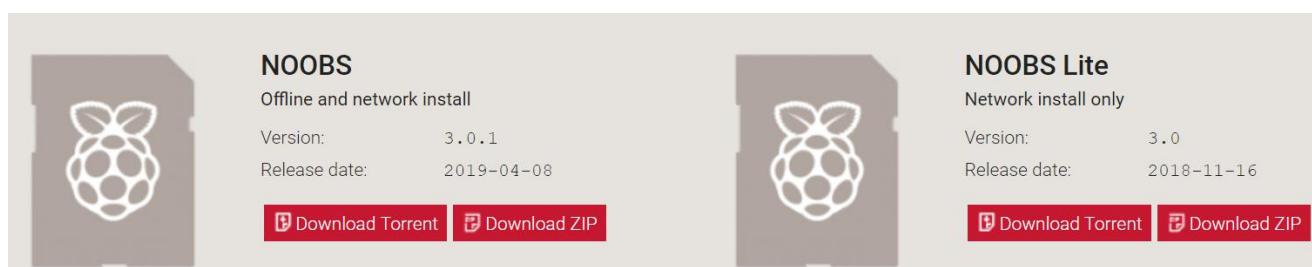
Raspberry Pi 本体	AC アダプター*1
モニター*1	モニター用 AC アダプター*1
HDMI ケーブル*1	マイクロ SD カード*1
マウス*1	キーボード*1
パソコン*1	

手順

ステップ 1

PC から NOOBS をダウンロードするには、NOOBS または NOOBS LITE を選択できる。この二つの唯一の違いは、NOOBS には内蔵式オフライン Raspberry Pi OS インストーラーが搭載されており、NOOBS LITE はオンラインでしか操作できないことである。ここでは、前者を使用することをお勧めする。Noobs のダウンロードアドレス：

<https://www.raspberrypi.org/downloads/noobs/>



ステップ 2

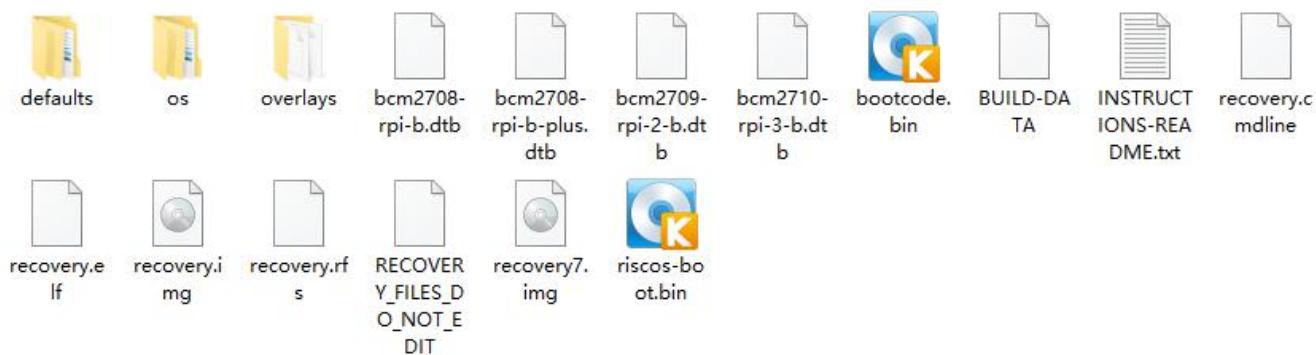
SD リーダーを挿入し、SD フォーマッターで Micro SD カードをフォーマットする（<https://www.sdcard.org/downloads/formatter/index.html>）。Micro SD カードに重要なファイルが保存されている場合は、まずバックアップを取ってください。

ステップ 3

次に、Raspberry Pi Web サイトからダウンロードした NOOBS zip 圧縮フォルダを解凍する。

- ダウンロードした圧縮ファイルを見つける —デフォルトでは、ダウンロードフォルダにあるはずである。
- それをダブルクリックしてファイルを抽出し、Explorer/Finder ウィンドウを開いたままにする。

最後に、NOOBS フォルダー内のファイルをすべて選択し、Micro SD カードにコピーする。



ステップ 4

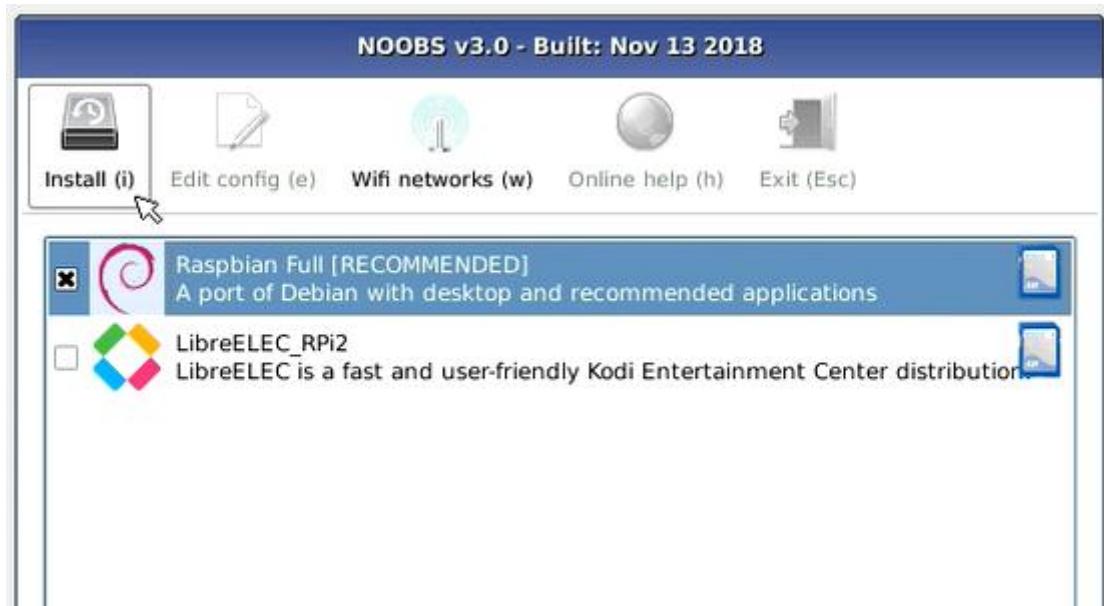
すべてのファイルが転送され後、Micro SD カードがポップアップする。

ステップ 5

Micro SD カードを Raspberry Pi に挿入する。それからモニター、キーボード、マウスを接続する。最後に、AC アダプターを使用して Raspberry Pi の電源を入れる。

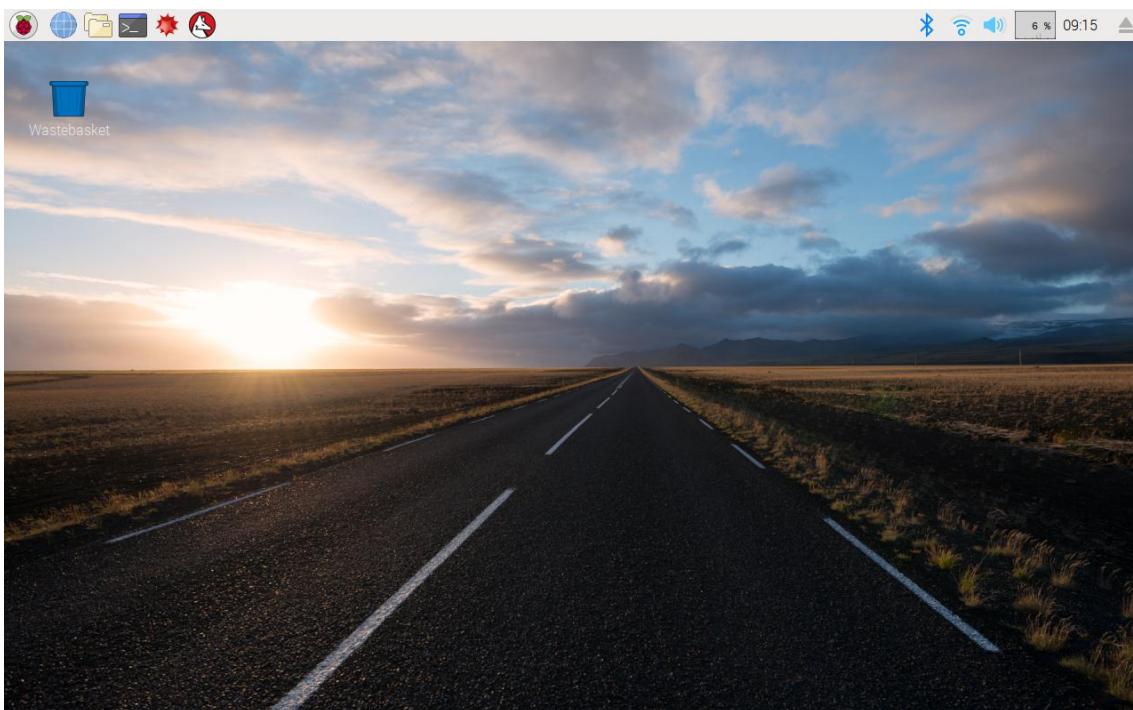
ステップ 6

起動後、NOOBS インターフェイスに入る。NOOBS LITE を使用する場合は、最初に Wi-Fi ネットワーク (w) を選択してください。Raspberry Pi OS のチェックボックスをオンにして、左上の「INSTALL」をクリックする。 NOOBS は実装を自動的に実行することに役立つ。このプロセスには数分かかる。



ステップ 7

実装が完了すると、システムが自動的に再起動し、システムのデスクトップが表示される。



ステップ 8

Raspberry Pi を初めて実行する場合は、「Welcome to Raspberry Pi」というアプリケーションがポップアップされ、初期設定の実行をガイドする。



ステップ 9

国/地域、言語、タイムゾーンを設定し、「NEXT」をクリックする。



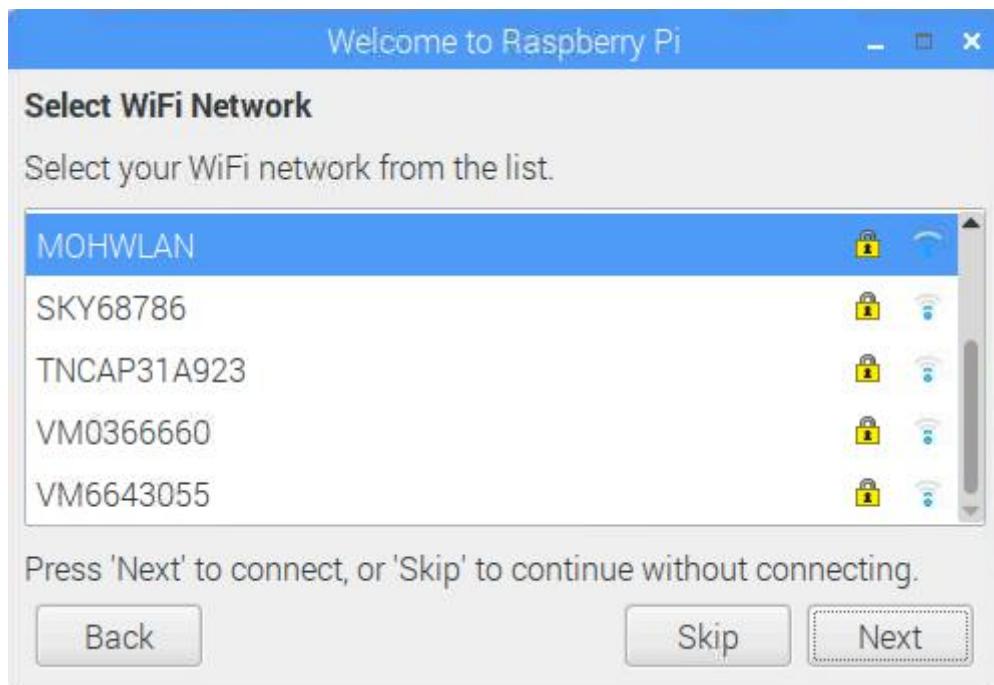
ステップ 10

Raspberry Pi の新しいパスワードを入力し、「NEXT」をクリックする。



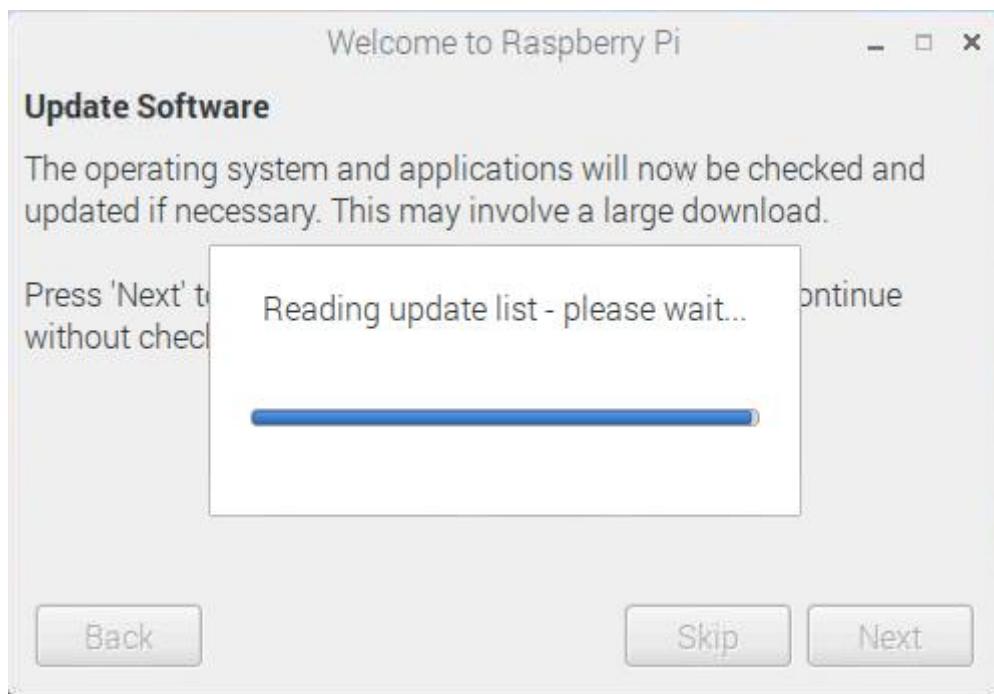
ステップ 11

Raspberry Pi を WIFI に接続し、「NEXT」をクリックする。



ステップ 12

更新を実行する。



ステップ 13

「DONE」をクリックして設定を完了する。



これで、Raspberry Pi を実行できる。

ご注意: Raspberry Pi の公式 Web サイトで NOOBS の完全なチュートリアルを確認できる：<https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up>

モニターを持っていない場合

モニターを持っていない場合、Raspberry Pi OS システムを Micro SD カードに直接書き込むことができ、Micro SD カードのネットワーク設定の構成ファイルを直接変更することで、PC 上の Raspberry Pi をリモートで制御できる。

必要な部品

Raspberry Pi 本体	AC アダプター*1
マイクロ SD カード*1	パソコン*1

書き込みシステム

ステップ 1

イメージ書き込みのツールを準備する。ここでは Etcher を使用する。次のリンクからソフトウェアをダウンロードできる：<https://www.balena.io/etcher/>

ステップ 2

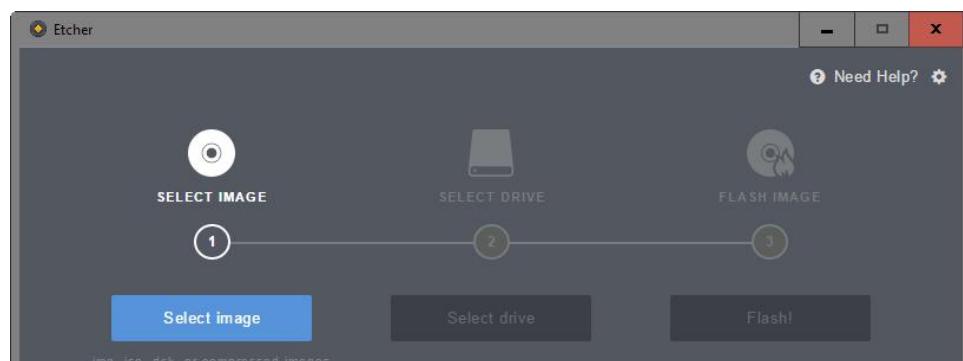
次のリンクをクリックして、公式 Web サイトで完全なイメージをダウンロードしてください：<https://www.raspberrypi.org/downloads/raspberry-pi-os/>。利用可能な Raspberry Pi OS システムには 3 種類あります。バージョンをインストールすることをお勧めします：Raspberry Pi OS とデスクトップおよび推奨ソフトウェアをインストールします。

ステップ 3

ダウンロードしたパッケージを解凍すると、中に.img と名前付けたファイルが表示される。

ステップ 4

Etcher を使用して、画像ファイル Raspberry Pi OS を Micro SD カードにフラッシュする。



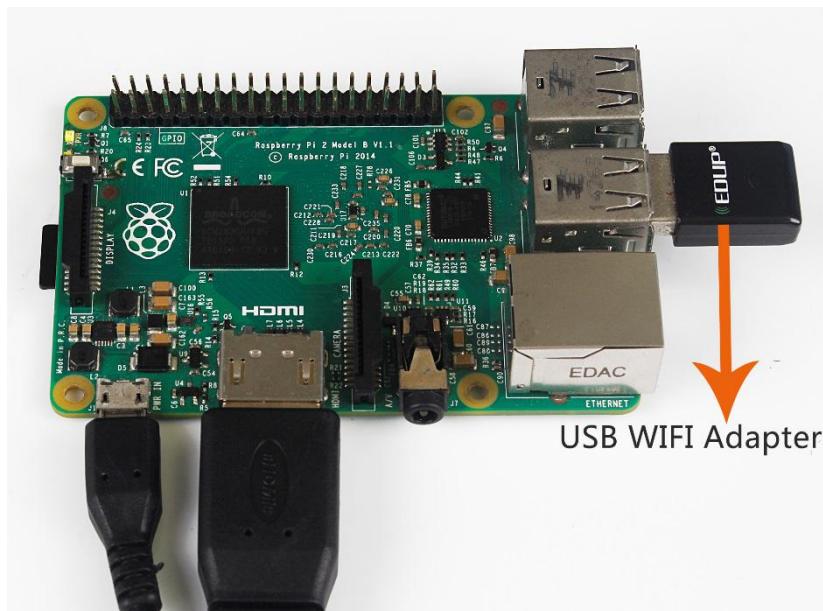
ステップ 5

この時点で、Raspberry Pi OS が実装された。ただし、それを活用する場合、下記の設定を完了する必要がある。

Raspberry Pi をインターネットに接続する

Raspberry Pi をネットワークに接続するには、二つの方法がある：一つ目はネットワークケーブルを使用する、もう一つは WIFI を使用することである。以下のように、WIFI を介して接続する方法について詳しく説明する。

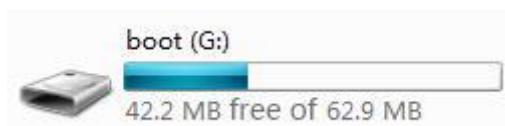
3B とそれ以降のバージョンの Raspberry Pi には Wifi 機能が搭載されている。使用しているのが Raspberry Pi の初期バージョンの場合、USB WIFI アダプターが必要である。詳細については、https://elinux.org/RPi_USB_Wi-Fi_Adapters にログインしてください。



WIFI 機能を使用する場合は、PC にあるディレクトリ /etc/wpa_supplicant/ で Micro SD カードの WIFI 構成ファイル wpa_supplicant.conf を変更する必要がある。

パソコンが Linux システムで動作している場合は、ディレクトリに直接アクセスして構成ファイルを変更できる。ただし、PC が Windows システムを使用している場合は、ディレクトリにアクセスできないため、次に必要なのはディレクトリにアクセスすることである。

/boot/を使用して、wpa_supplicant.conf と名前付ける新しいファイルを作成することである。



ファイルに次の内容を入力する。

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=COUNTRY
network={
ssid="SSID"
psk="PASSWORD"
key_mgmt=WPA-PSK
priority=1
}
```

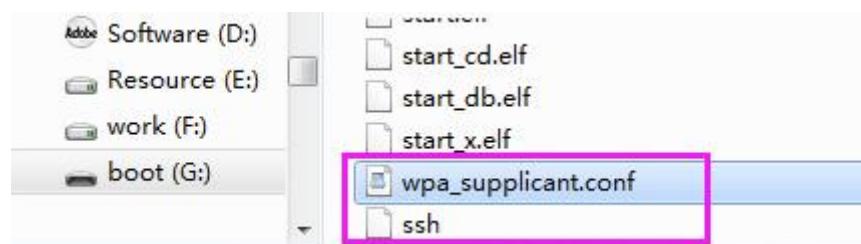
COUNTRY は、Raspberry Pi を使用している国の ISO/IEC alpha2 コードを 2 文字設定する必要があります。

https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements

「SSID」を WiFi のカスタマイズ名に、「PASSWORD」をパスワードに置き換える必要がある。これらを行うことにより、Raspberry Pi OS システムはこのファイルをターゲットディレクトリに自動的に移動し、次回の実行時に元の WIFI 構成ファイルを上書きする。

SSH を起動する

Raspberry Pi のリモートコントロール機能を使用するには、リモートログインセッションやその他のネットワークサービスのセキュリティを提供する、より信頼性の高いプロトコルである SSH を最初に起動する必要がある。通常、Raspberry Pi の SSH は無効な状態である。さらに、実行する場合は、ディレクトリ/boot/の下に SSH という名前のファイルを作成してください。



これで、Raspberry Pi OS システムの構成は完了した。Micro SD カードを Raspberry Pi に挿入すると、すぐに使用できる。

IP アドレスを取得する

Raspberry Pi が WIFI に接続されたら、その IP アドレスを取得する必要がある。IP アドレスを知る方法はたくさんあるが、そのうちの 2 つを以下のように示す。

1. ルーター経由で確認する

ルーター（ホームネットワークなど）にログインする権限がある場合は、ルーターの管理インターフェイスで Raspberry Pi に割り当てられたアドレスを確認できる。

システムのデフォルトのホスト名 - Raspberry Pi OS は raspberrypi であり、それを見つける必要がある。（ArchLinuxARM システムを使用している場合は、alarmpiを見つけてください。）

2. ネットワークセグメントスキャン

ネットワークスキャンを使用して、Raspberry Pi の IP アドレスを検索することもできる。ソフトウェア、アドバンスド IP スキャナーなどを適用可能である。

IP 範囲セットをスキャンすると、接続されているデバイスの名前がすべて表示される。同様に、Raspberry Pi OS システムのデフォルトのホスト名は raspberrypi であり、今ホスト名を見つけてください。

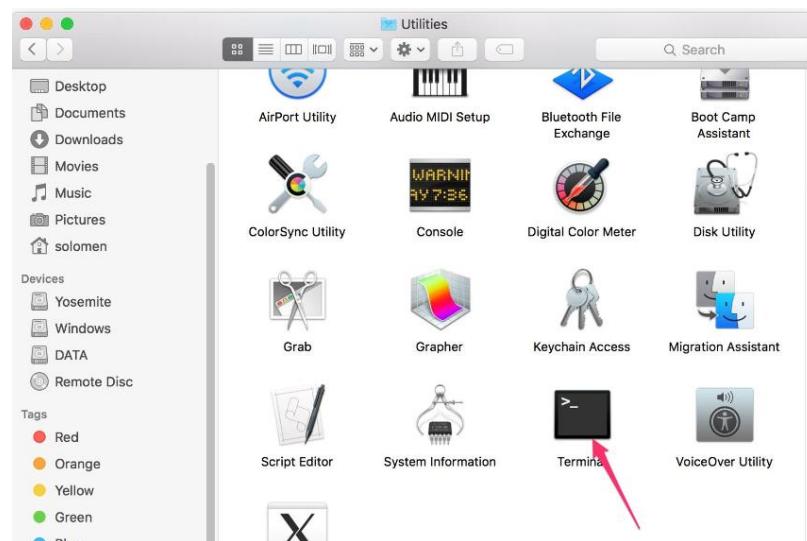
SSH リモートコントロールを使用する

SSH を適用することにより、Raspberry Pi の Bash Shell を開くことができる。Bash は Linux の標準のデフォルトシェルである。シェル自体は、お客様と Unix / Linux をリンクするブリッジである「C」で書き込まれたプログラムである。さらに、必要な作業のほとんどを完了することに役立ち。

Linux または/ Mac OS X ユーザーの場合

ステップ 1

アプリケーション->ユーティリティに入り、ターミナルを見つけてから開く。



ステップ 2

ssh pi @ ip_address と入力する。「pi」はユーザー名で、「ip_address」はの IP アドレスである。例えば：

```
ssh pi@192.168.18.197
```

ステップ 3

「yes」と入力する。

```
1. ssh pi@192.168.18.197 (ssh)  
Last login: Fri Apr 12 16:56:20 on ttys000  
  
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]  
$ ssh pi@192.168.18.197  
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.  
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.  
Are you sure you want to continue connecting (yes/no)?
```

ステップ 4

デフォルトのパスワード「raspberry」を入力する。

```
1. ssh pi@192.168.18.197 (ssh)  
Last login: Fri Apr 12 16:56:20 on ttys000  
  
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]  
$ ssh pi@192.168.18.197  
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.  
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.  
pi@192.168.18.197's password:
```

ステップ 5

これで、Raspberry Pi が接続され、次のステップに進む準備ができた。

```
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126  
  
SSH is enabled and the default password for the 'pi' user has not been changed.  
This is a security risk - please login as the 'pi' user and type 'passwd' to set  
a new password.  
  
pi@raspberrypi:~ $
```

ご注意： パスワードを入力すると、ウィンドウに文字が表示されないが、これは正常である。必要なのは、正しいパスコードを入力するだけである。.

Windows ユーザーの場合

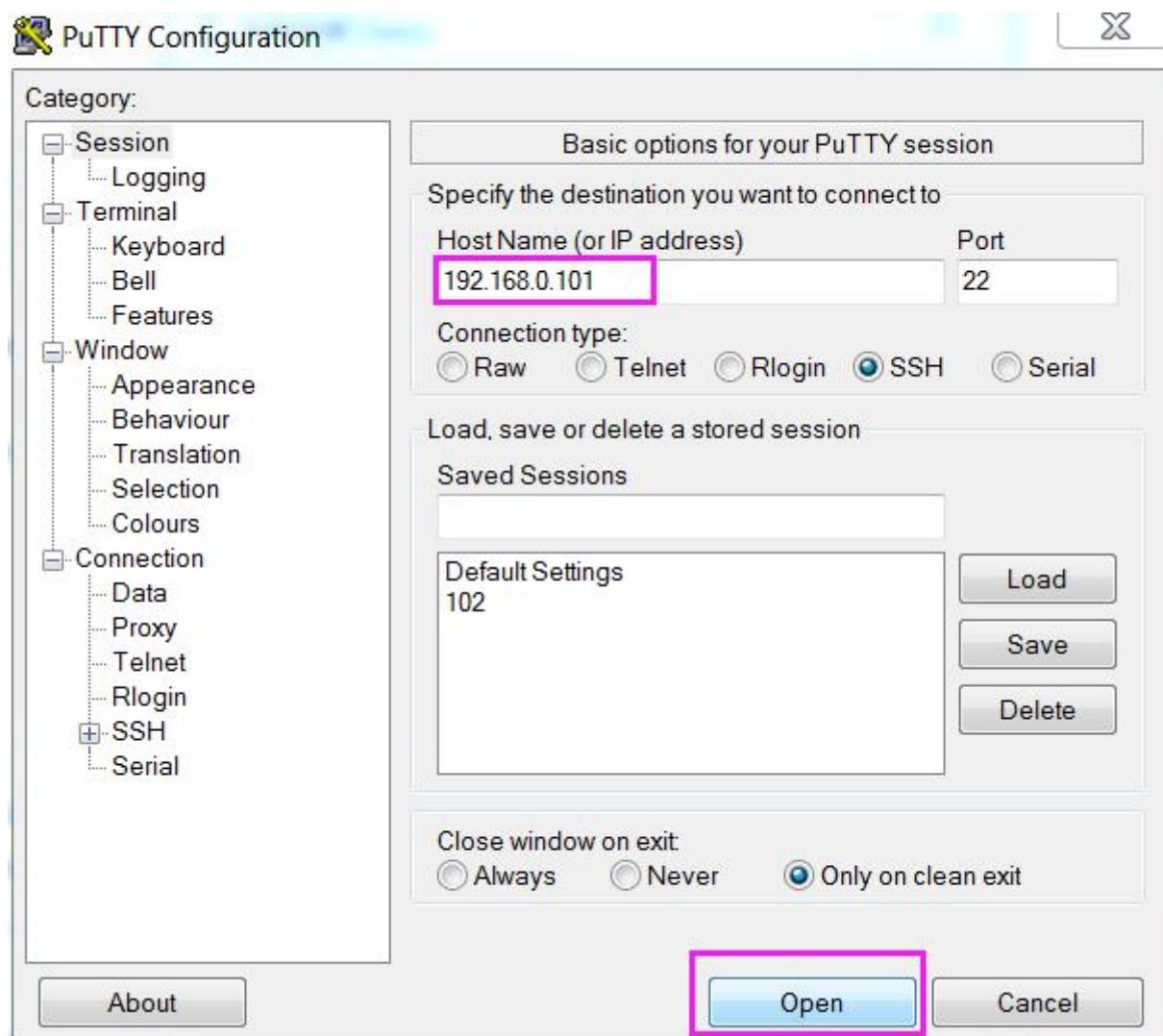
Windows ユーザーの場合、いくつかのソフトウェアのアプリケーションで SSH を使用できる。ここでは、PuTTY をお勧めする。

ステップ 1

PuTTY をダウンロードする。

ステップ 2

PuTTY を開き、左側のツリー構造にあるセッションをクリックする。「ホスト名」（または IP アドレス）の下のテキストボックスに RPi の IP アドレスを入力し、ポートに 22（デフォルトでは 22）を入力する。

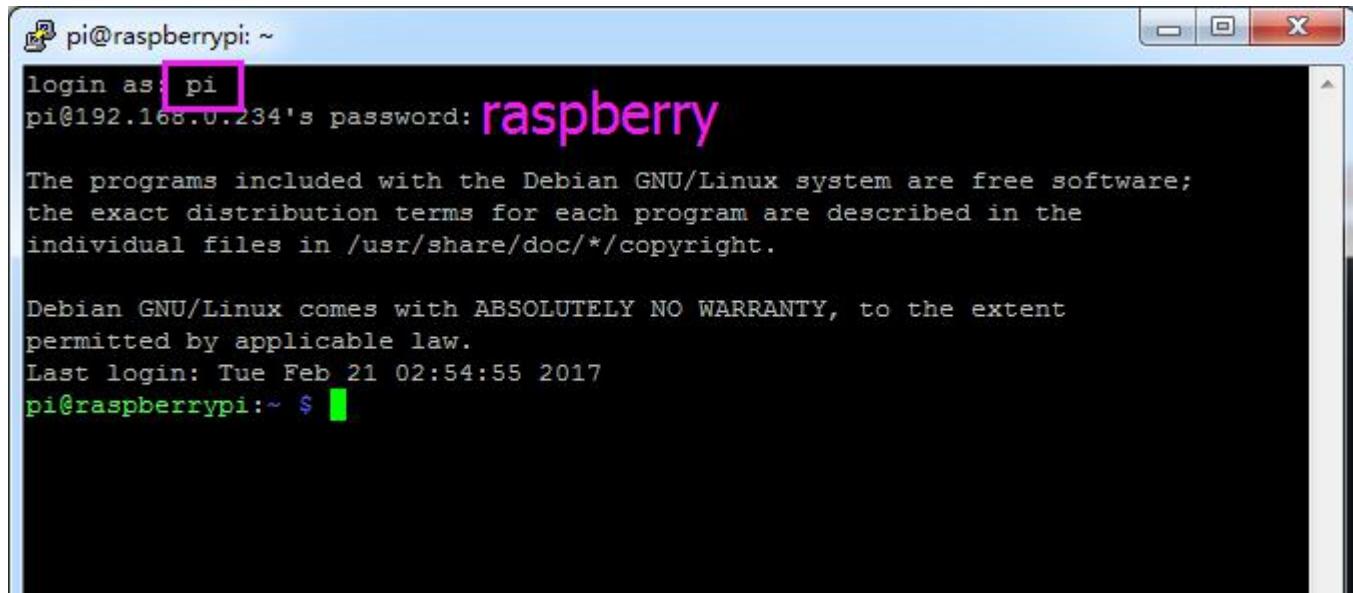


ステップ 3

開くをクリックする。IP アドレスを使用して Raspberry Pi に初めてログインすると、安全上の指示が表示されることに注意してください。その時、はいをクリックしてください。

ステップ 4

PuTTY ウィンドウに「login as:」と表示されたら、「pi」（RPi のユーザー名）とパスワード「raspberry」（変更していない場合はデフォルトのパスワードである）を入力する。



```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.234's password: raspberry

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 21 02:54:55 2017
pi@raspberrypi:~ $
```

ステップ 5

ここで、Raspberry Pi を接続し、次の手順を実行する。

ご注意: パスワードを入力すると、ウィンドウに文字が表示されないが、これは正常である。必要なのは、正しいパスコードを入力するだけである。

リモートデスクトップ

コマンドウィンドウを使用して Raspberry Pi を制御することに慣れない場合は、リモートデスクトップ機能も使用できる。これにより、Raspberry Pi 内のファイルを簡単に管理できる。Raspberry Pi のデスクトップをリモートで制御するには、VNC と XRDP.という二つの方法がある。

VNC

VNC を介してリモートデスクトップの機能を使用できる。

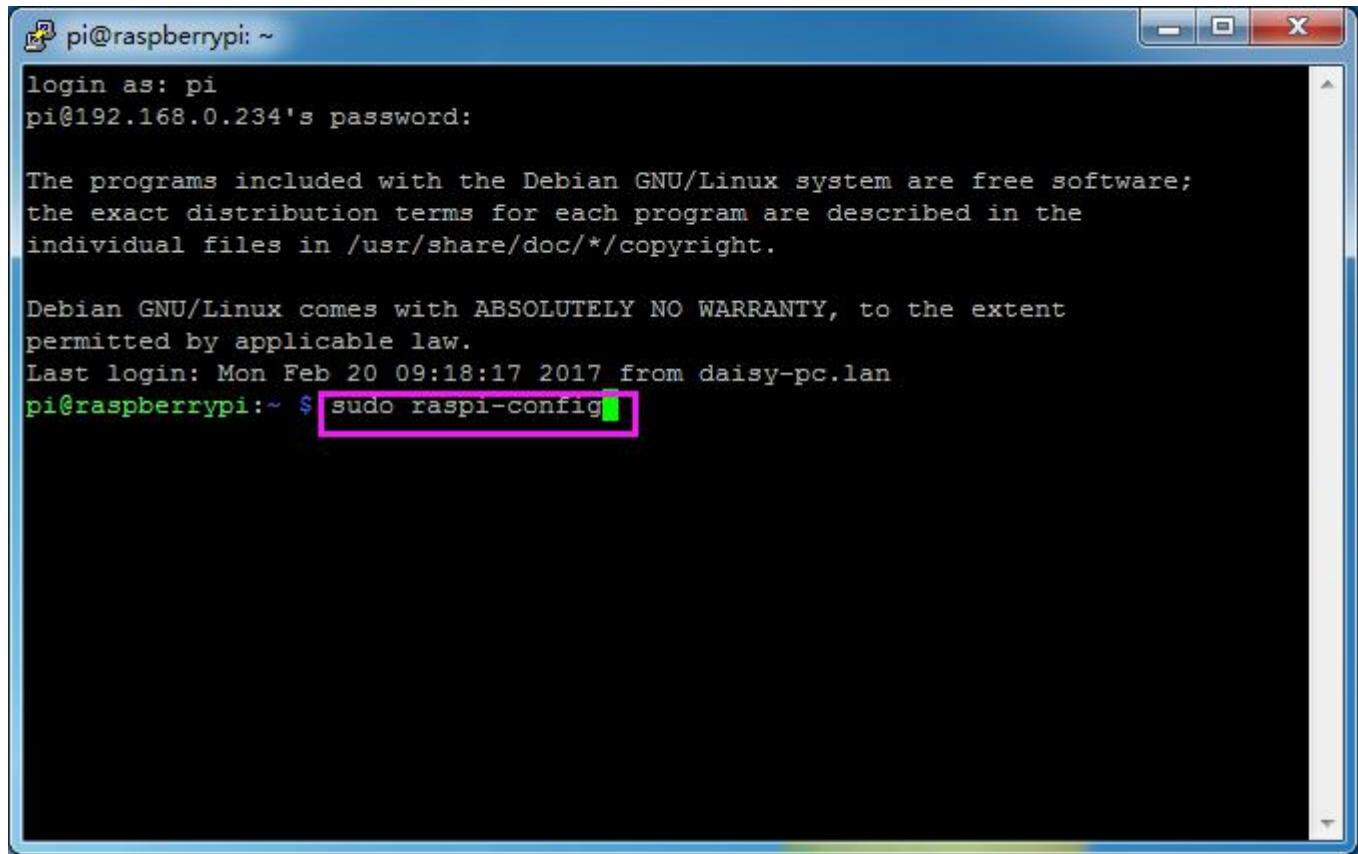
VNC サービスを有効にする

VNC サービスがシステムに実装された。デフォルトでは、VNC は無効である。設定で有効にする必要がある。

ステップ 1

次のコマンドを入力する：

```
sudo raspi-config
```



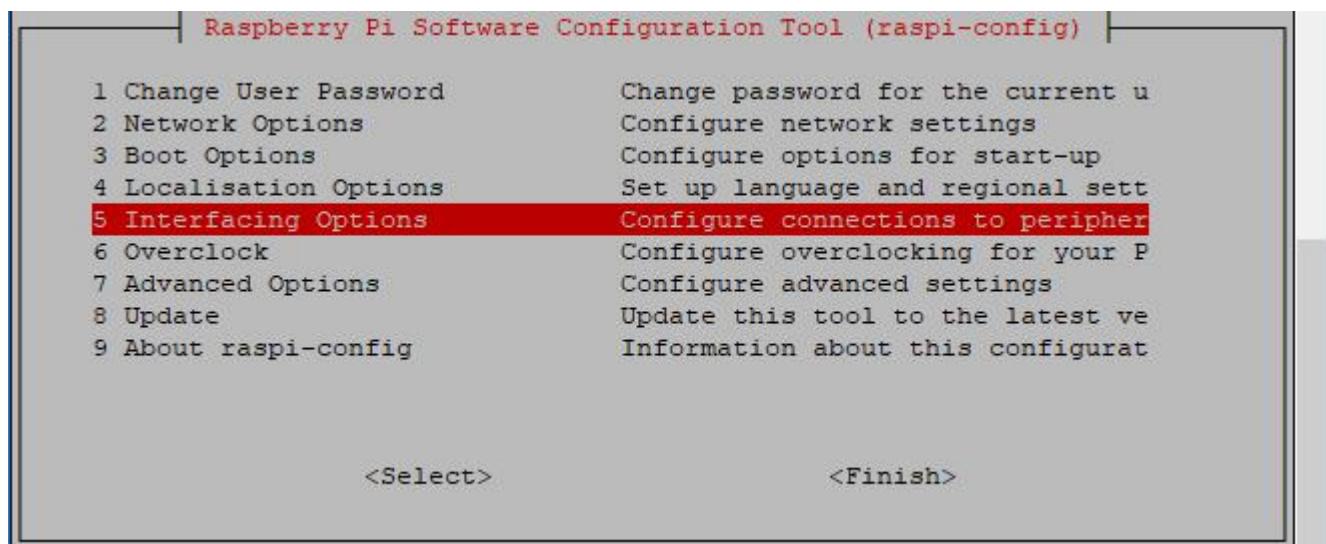
```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.234's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 20 09:18:17 2017 from daisy-pc.lan
pi@raspberrypi: ~ $ sudo raspi-config
```

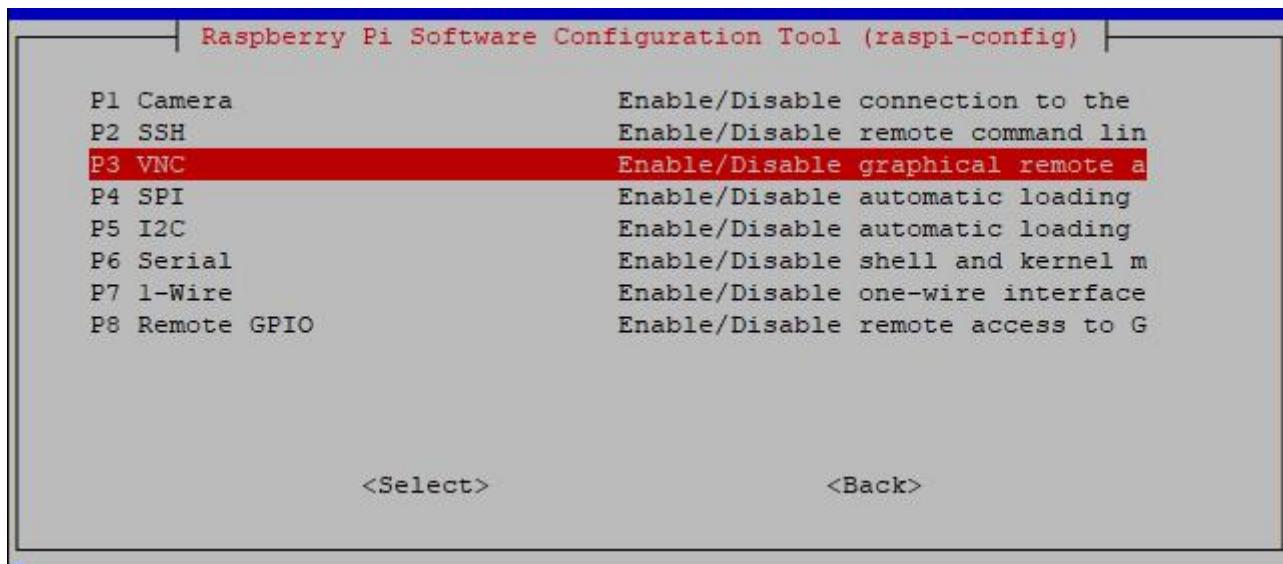
ステップ 2

構成インターフェイスで、キーボードの上下左右のキーを使用して「インターフェイスオプション」を選択する。



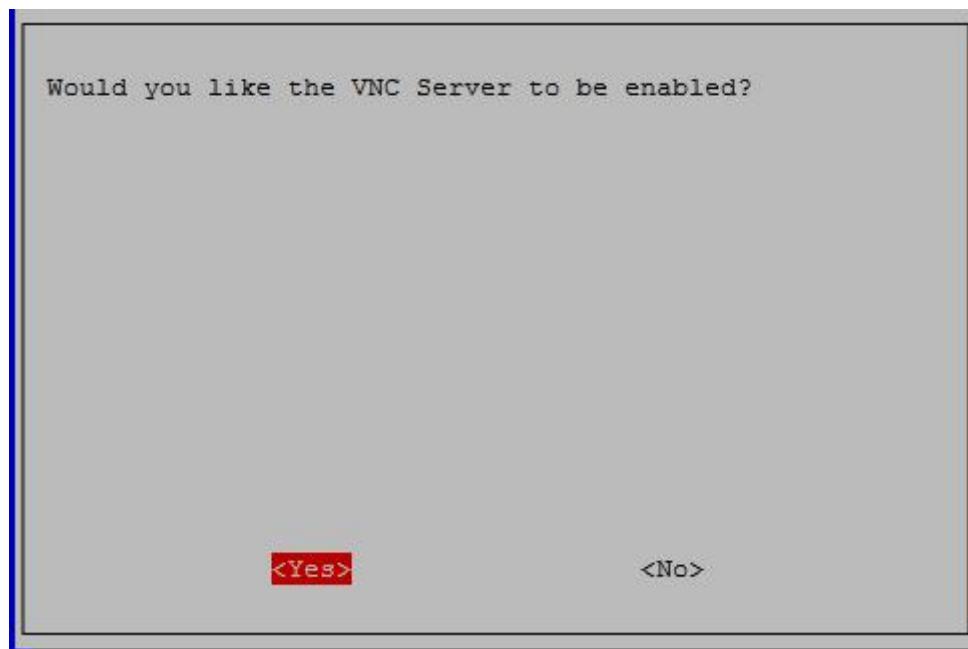
ステップ 3

VNC を選択する。



ステップ 4

「YES」 -> 「OK」 -> 「FINISH」を選択して、構成を終了する。



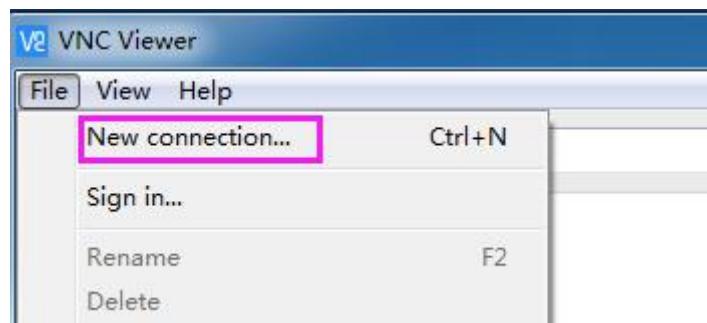
VNC にログインする

ステップ 1

VNC Viewer をパソコンに実装する必要がある。 実装が完了したら、それを開く。

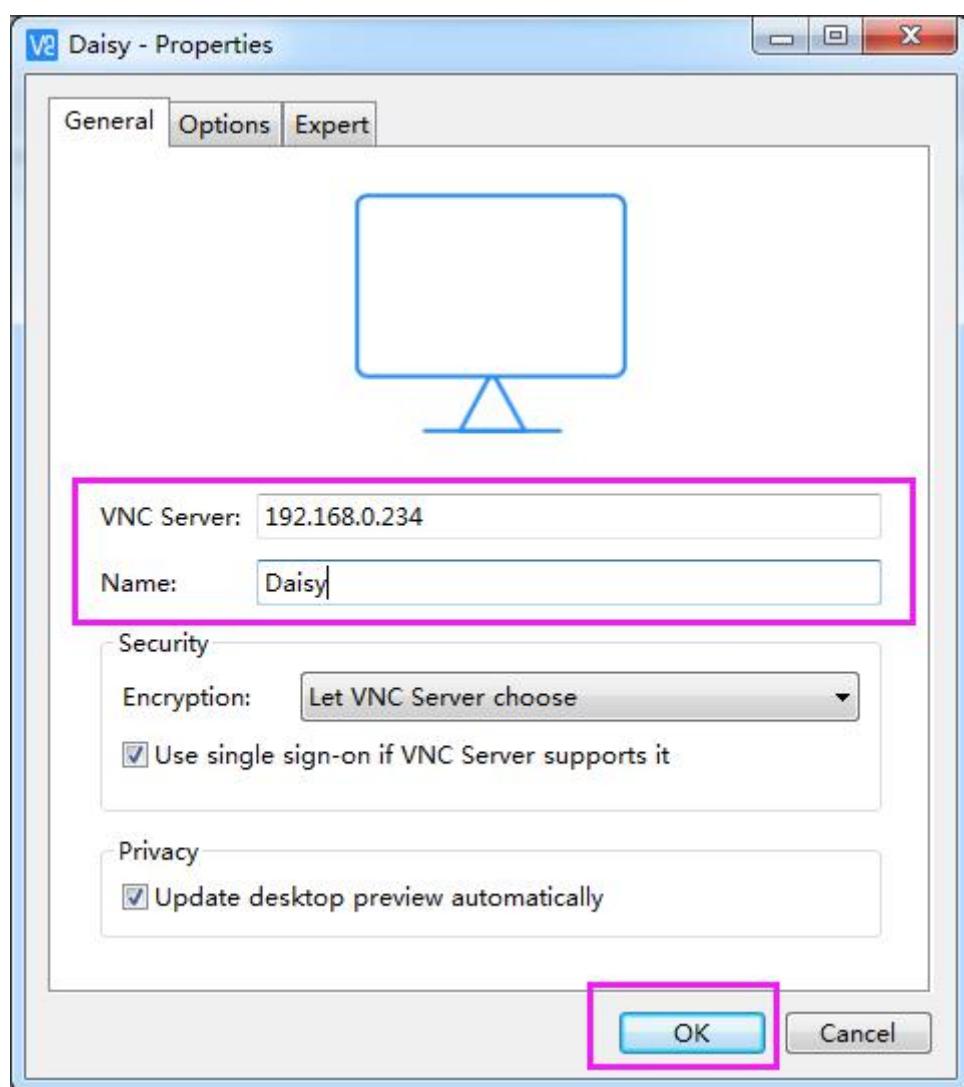
ステップ 2

次に、「New connection」を選択する。



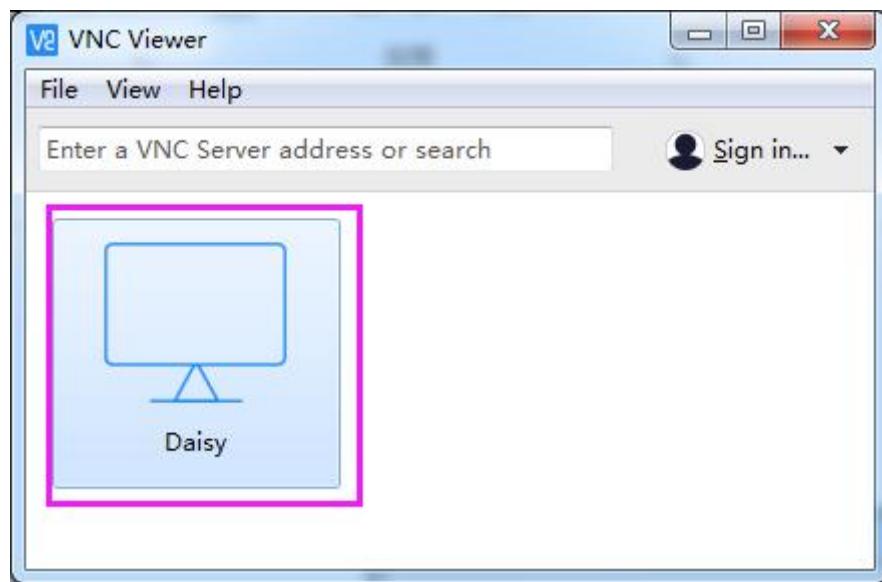
ステップ 3

Raspberry Pi の IP アドレスと任意の名前を入力する。



ステップ 4

作成した接続をダブルクリックする:



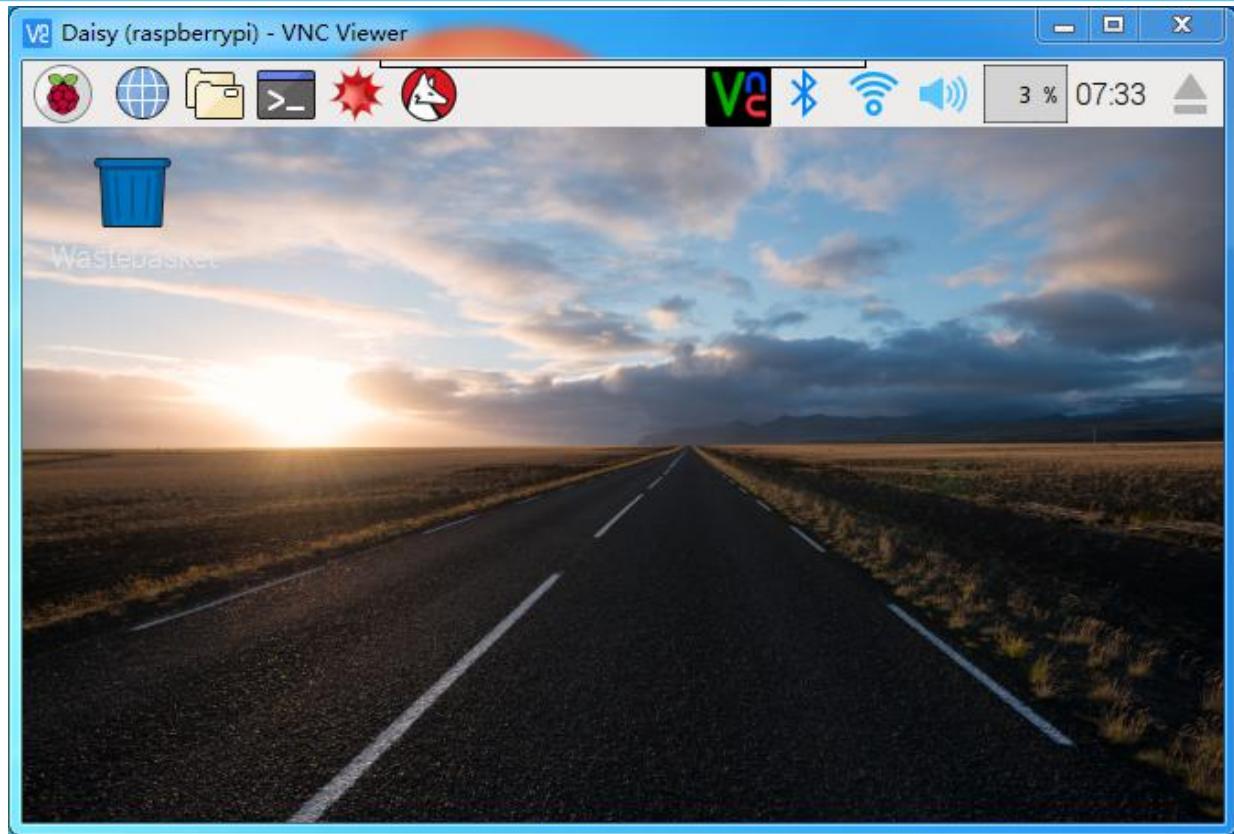
ステップ 5

ユーザー名 (pi) とパスワード (デフォルトでは raspberry である) を入力する。



ステップ 6

これで、Raspberry Pi のデスクトップが表示される:



XRDP

xrdp は、RDP（Microsoft リモートデスクトッププロトコル）を使用してリモートマシンにグラフィカルログインを提供する。

XRDP を実装する

ステップ 1

SSH を使用して Raspberry Pi にログインする。

ステップ 2

以下の指示を入力して、XRDP を実装する。

```
sudo apt-get update
sudo apt-get install xrdp
```

ステップ 3

その後、実装が開始される。

「Y」を入力し、「Enter」キーを押して確認する。

```
pi@raspberrypi:~ $ sudo apt-get install xrdp
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
Suggested packages:
  vnc-java mesa-utils x11-xfs-utils
The following NEW packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
  xrdp
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,468 kB of archives.
After this operation, 17.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

ステップ 4

実装が完了したら、Windows リモートデスクトップアプリケーションを使用して Raspberry Pi にログインする必要がある。

XRDP にログインする

ステップ 1

Windows ユーザーの場合、Windows に付属のリモートデスクトップ機能を利用できる。Mac ユーザーの場合は、APP Store から Microsoft リモートデスクトップをダウンロードして使用できる。両者に特に大きな違いはない。Windows リモートデスクトップについて、次の例をご参照ください。

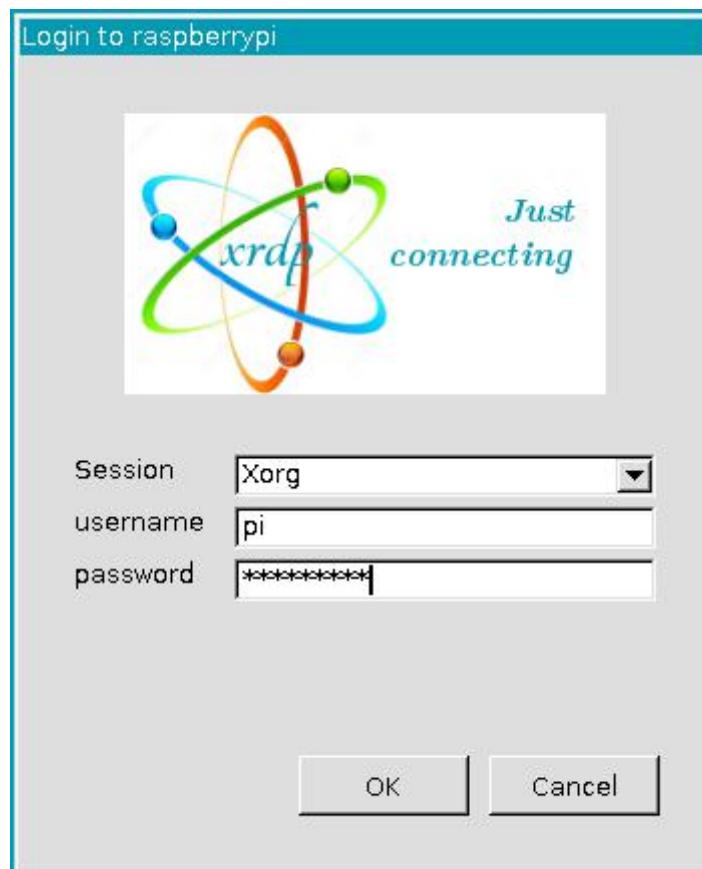
ステップ 2

[実行 (WIN + R)]に「mstsc」と入力してリモートデスクトップ接続を開き、Raspberry Pi の IP アドレスを入力して、「接続」をクリックする。



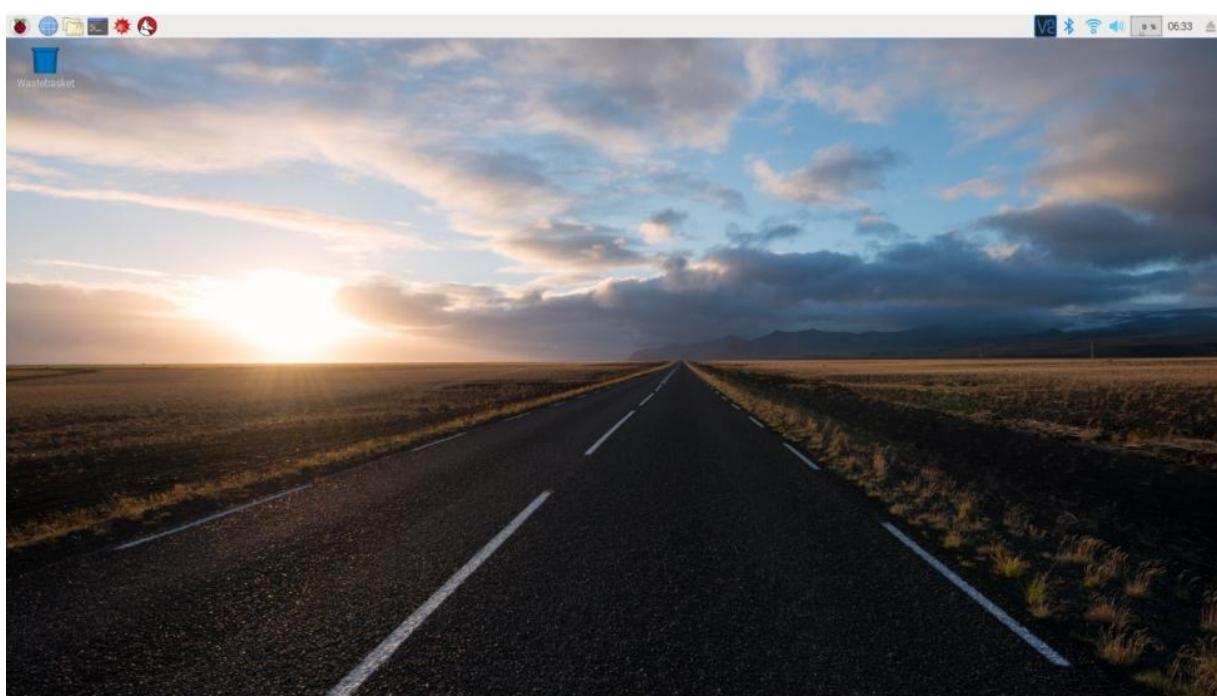
ステップ 3

次に、xrdp ログインページが表示される。ユーザー名とパスワードを入力してください。その後、「OK」をクリックしてください。初めてログインする場合は、ユーザー名は「pi」、パスワードは「raspberry」である。



ステップ 4

ここでは、リモートデスクトップを使用して RPi に正常にログインできる。



ライブラリ

Raspberry Pi を使用したプログラミングでは、wiringPi と RPi.GPIO という 2 つの重要なライブラリが使用されている。Raspberry Pi OS イメージはデフォルトでそれらを実装するため、直接使用できる。

RPi.GPIO

Python ユーザーの場合、RPi.GPIO が提供する API を使用して GPIO をプログラミングできる。RPi.GPIO は、Raspberry Pi GPIO チャンネルを制御するモジュールである。このパッケージは Raspberry Pi で GPIO を制御するクラスを提供する。例とドキュメントについては、次の URL にアクセスしてください：<https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>

RPi.GPIO が実装されているかどうかをテストするには、Python を入力する：

```
python
```

```
pi@raspberrypi:~ $ python
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Python CLI で、「import RPi.GPIO」と入力する。エラーが表示されない場合は、RPi.GPIO の実装が完了したと意味する。

```
import RPi.GPIO
```

```
pi@raspberrypi:~ $ python
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO
>>> 
```

Python CLI を終了する場合は、次を入力する：

```
exit()
```

```
>>> exit()
pi@raspberrypi:~ $ 
```

WiringPi

wiringPi は、Raspberry Pi プラットフォームに適用される C 言語の GPIO ライブラリである。GUN Lv3 に準拠している。wiringPi の機能は、Arduino の配線システムの機能と似ている。Arduino に精通したユーザーは、wiringPi をより簡単に使用できる。

wiringPi には、Raspberry Pi のあらゆる種類のインターフェイスを制御できる多数の GPIO コマンドが含まれている。以下の手順により、wiringPi ライブラリが正常に実装されたかどうかをテストできる。

```
gpio -v
```

```
pi@raspberrypi:~/davinci-kit-for-raspberry-pi/c/1.1.1 $ gpio -v
gpio version: 2.52
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 4B, Revision: 01, Memory: 2048MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 4 Model B Rev 1.1
  * This Raspberry Pi supports user-level GPIO access.
```

ご注意:

Raspberry Pi 4B を使用している場合、GPIO のバージョンは 2.50 なので、C 言語コードを実行した後応答が来ないです。つまり、GPIO ピンは機能しません。解決策では、手動で GPIO バージョンを 2.52 に更新する必要があります。更新手順は次の通りです：

次のコードを実行し、GPIO バージョンは自動的にアップグレードされます。

```
cd /tmp
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

アップグレードが完了した後、「gpio -v」をチェックして、

GPIO バージョンが 2.52 に更新したかいないか確認してください。

gpio readall

Pi 3													
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM			
		3.3v			1	2		5v					
2	8	SDA.1	ALT0	1	3	4		5V					
3	9	SCL.1	ALT0	1	5	6		0v					
4	7	GPIO. 7	IN	0	7	8	1	IN	TxD	15	14		
		0v			9	10	1	IN	RxD	16	15		
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18		
27	2	GPIO. 2	IN	0	13	14		0v					
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23		
		3.3v			17	18	0	IN	GPIO. 5	5	24		
10	12	MOSI	ALT0	1	19	20		0v					
9	13	MISO	ALT0	1	21	22	0	IN	GPIO. 6	6	25		
11	14	SCLK	ALT0	0	23	24	1	OUT	CE0	10	8		
		0v			25	26	1	OUT	CE1	11	7		
0	30	SDA.0	IN	1	27	28	1	OUT	SCL.0	31	1		
5	21	GPIO.21	IN	0	29	30		0v					
6	22	GPIO.22	IN	0	31	32	0	IN	GPIO.26	26	12		
13	23	GPIO.23	IN	1	33	34		0v					
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16		
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20		
		0v			39	40	0	IN	GPIO.29	29	21		

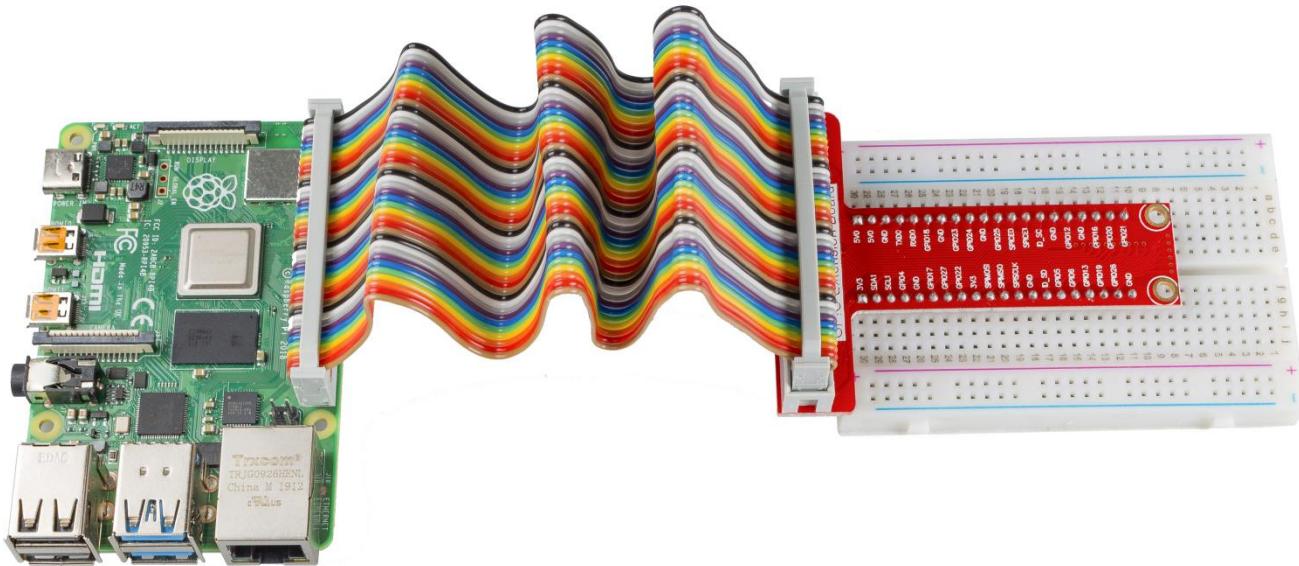
wiringPi の詳細については、以下の URL を参照ください。

<http://wiringpi.com/download-and-install/>

GPIO 拡張ボード

コマンドの学習を開始する前に、最初に Raspberry Pi のピンについて詳しく知る必要がある。これは、その後の研究の鍵となる。

GPIO 拡張ボードによって Raspberry Pi のピンをブレッドボードに簡単に引き出して、頻繁に挿入または引き抜きによる GPIO の損傷を防止できる。これは、Raspberry Pi モデル B+、世代 2 モデル B および世代 3、4 モデル B に適用する 40 ピン GPIO 拡張ボードと GPIO ケーブルである。



ピン番号

Raspberry Pi のピンには、命名方法が三つあり、つまり、wiringPi、BCM、およびボードである。これらの命名方法の中で、40 ピン GPIO 拡張ボードは命名方法 BCM を使用している。ただし、I2C ポートや SPI ポートなどの一部の特別なピンでは、付属の名前を使用する。次の表は、WiringPi、ボード、および GPIO 拡張ボード上の各ピンの固有の命名方法を示している。たとえば、GPIO17 の場合、ボードの命名方法によると 11 で、wiringPi の命名方法によると 0 で、固有の命名方法によると GPIO0 である。

ご注意:

- 1) C 言語では、使用されている命名方法は WiringPi である。
- 2) Python 言語では、使用される命名方法はボードおよび BCM であり、GPIO.setmode () 機能を使用してそれらを設定する。

Name	WiringPi	Board	BCM		Board	WiringPi	Name
		GPIO Extention Board					
3.3V	3V3	1	3V3	5.0V	2	5.0V	5V
SDA	8	3	SDA	5.0V	4	5.0V	5V
SCL	9	5	SCL	GND	6	GND	0V
GPIO7	7	7	GPIO4	TXD	8	15	TXD
0V	GND	9	GND	RXD	10	16	RXD
GPIO0	0	11	GPIO17	GPIO18	12	1	GPIO1
GPIO2	2	13	GPIO27	GND	14	GND	0V
GPIO3	3	15	GPIO22	GPIO23	16	4	GPIO4
3.3V	3.3V	17	3.3V	GPIO24	18	5	GPIO5
MOSI	12	19	MOSI	GND	20	GND	0V
MISO	13	21	MISO	GPIO25	22	6	GPIO6
SCLK	14	23	SCLK	CEO	24	10	CEO
0V	GND	25	GND	CE1	26	11	CE1
IN_SDA	30	27	EED	EEC	28	31	ID_SCL
GPIO21	21	29	GPIO5	GND	30	GND	0V
GPIO22	22	31	GPIO6	GPIO12	32	26	GPIO26
GPIO23	23	33	GPIO13	GND	34	GND	0V
GPIO24	24	35	GPIO19	GPIO16	36	27	GPIO27
GPIO25	25	37	GPIO26	GPIO20	38	28	GPIO28
0V	GND	39	GND	GPIO21	40	29	GPIO29

コードをダウンロードする

コードをダウンロードする前に、サンプルコードは Raspberry Pi OS のみでテストすることに注意してください。ダウンロードには二つの方法がある：

方法 1: git clone を使用する (推奨)

Raspberry Pi にログインし、ディレクトリを /home/pi に変更する。

```
cd /home/pi/
```

ご注意: 現在のパスから目的のディレクトリに移動する cd。非公式には、パス /home /pi/ に移動する。

GitHub からリポジトリを複製する

```
git clone https://github.com/sunfounder/davinci-kit-for-raspberry-pi.git
```

方法 2: コードをダウンロードする

github からソースコードをダウンロードする：

<https://github.com/sunfounder/davinci-kit-for-raspberry-pi>

This repository is for SunFounder Da Vinci Kit for Raspberry Pi.

Branch: master ▾ New pull request

5 commits 1 branch 0 releases 1 contributor GPL-2.0

Clone with HTTPS ⓘ Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/sunfounder/davinci-kit-for-raspberry-pi>

Open in Desktop Download ZIP

5 days ago

File	Action
c	Upload code
python	Upload code
LICENSE	Upload code
README.md	upload codes
show	Upload code

1 出力

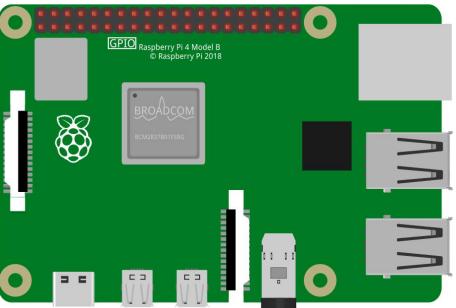
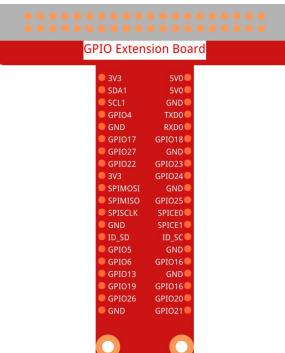
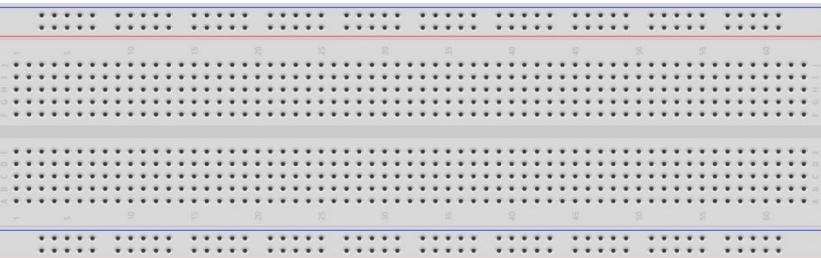
1.1 ディスプレイ

1.1.1 Blinking LED

前書き

このレッスンでは、プログラミングによって LED を点滅させる方法を学習する。設定により、LED は一連の興味深い現象を生成できる。今、行動しよう。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	LED*1
	 GPIO Extension Board Pinout: ● 3V3 SVO ● SDA1 SVO ● SCL1 GND ● GPIO4 TXD0 ● GND RXD0 ● GPIO17 GPIO18 ● GND GND ● GPIO27 GND ● GPIO22 GPIO23 ● 3V3 GPIO24 ● SPI MOSI GND ● SPI MOSI GPIO25 ● SPI CLK SPI CE0 ● GND ID_SCK ● GPIO5 GPIO26 ● GPIO13 GND ● GPIO19 GPIO16 ● GPIO26 GPIO20 ● GND GPIO21	
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		抵抗器 (220Ω) *1
		

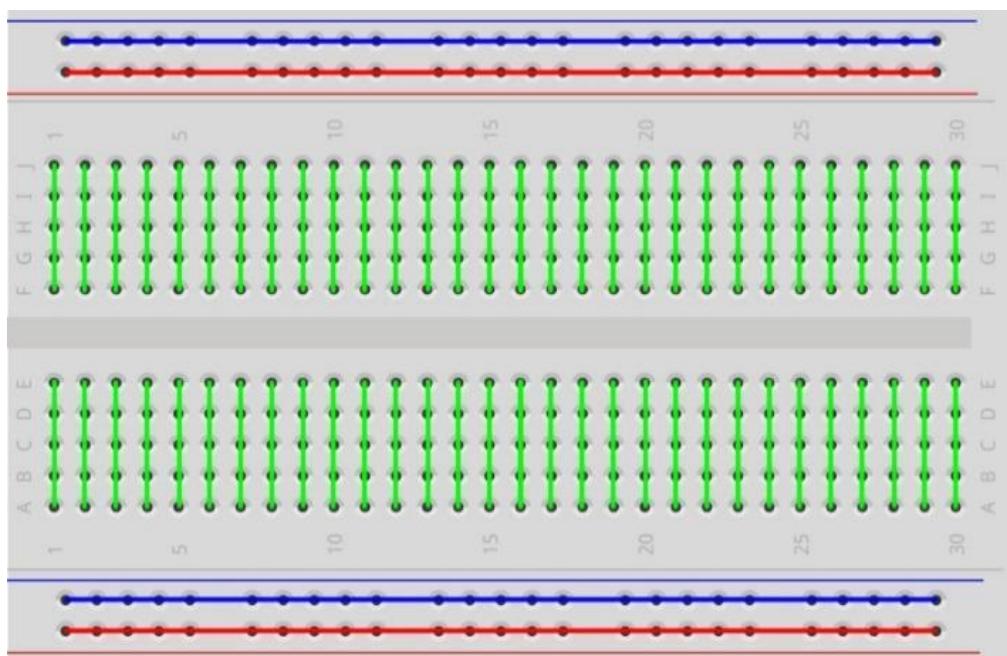
ご注意: スムーズに進めるには、独自の Raspberry Pi、TF カード、および Raspberry Pi の電源を用意する必要がある。

原理

ブレッドボード

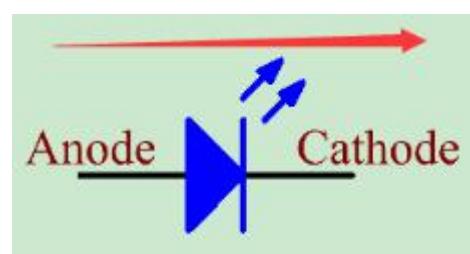
ブレッドボードは、電子装置のプロトタイピングの構築基盤である。それは回路設計を完了する前に、回路を迅速に作って、テストするために使用される。また、上記の部品を IC や抵抗器、ジャンパー線などのように挿入できる多くの穴が搭載されている。ブレッドボードを使用すると、部品を簡単に差し込んだり、取り外したりすることができる。

この写真は、full + ブレッドボードの内部構造を示している。ブレッドボード上のこれらの穴は互いに独立しているように見えるが、実際には内部で金属ストリップを介して互いに接続されている。



LED

LED は一種のダイオードである。LED の長いピンが正極に接続され、短いピンが負極に接続されている場合にのみ、LED が点灯する。

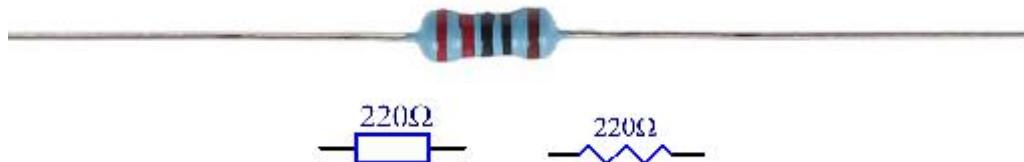


LED を電源に直接接続することはできないため、部品に損傷を与える可能性がある。160Ω 以上の抵抗 (5V 動作電圧) は、LED の回路に直列に接続する必要がある。

抵抗器

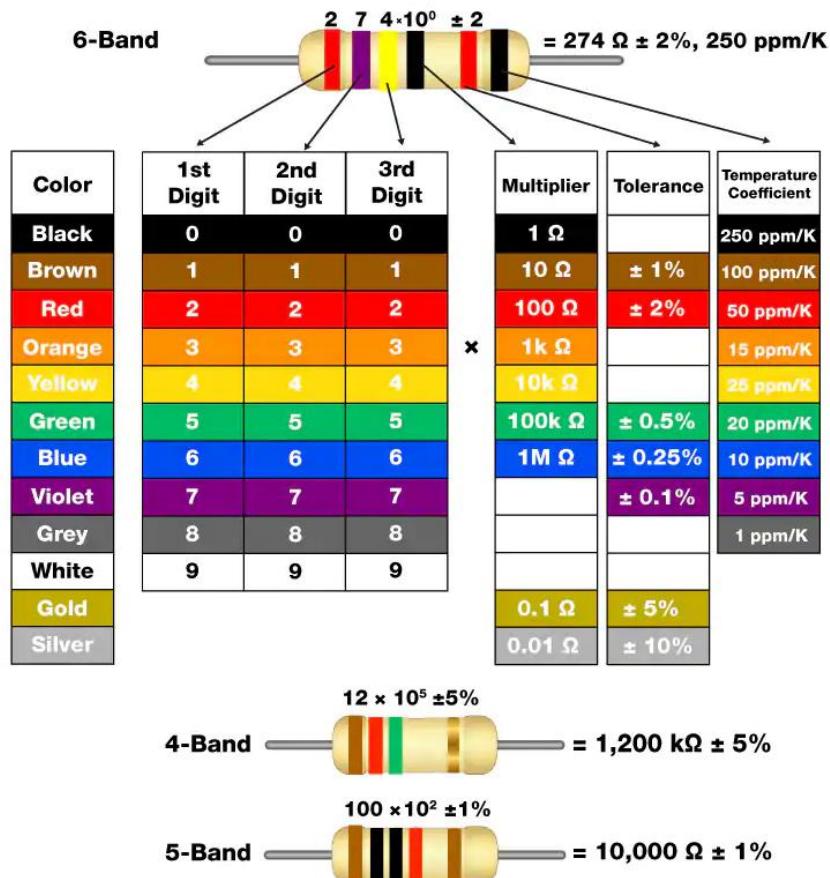
抵抗器は、分岐電流を制限できる電子素子である。固定抵抗器は抵抗値を変更できない抵抗器の一種であるが、ポテンショメータまたは可変抵抗器の抵抗値は調整できる。

このキットには固定抵抗器が適用される。回路では、接続された部品を保護することは重要である。次の図は、実際の物体、220Ω抵抗器、および抵抗器汎用の二つの回路記号を示している。Ωは抵抗値の単位で、より大きな単位は KΩ、MΩなどである。それらの関係は次のように示している: 1M= 1000KΩ、1KΩ=1000Ω。通常、抵抗値はマークされている。したがって、回路にこれらの記号が表示される場合、抵抗があることを意味する。



抵抗器を使用する場合、まず抵抗値を知る必要がある。以下の二つの方法がある: 抵抗の帯域を観察すること、またはマルチメーターを使用して抵抗を測定すること。より便利で迅速なので、一番目の方法をお勧めする。値を測定するには、マルチメーターを使用してください。

カードに示されているように、各色は数字を表している。



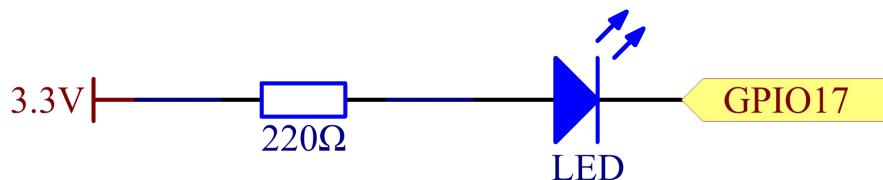
回路図

この実験では、まず 220Ω の抵抗器を陽極（LED の長いピン）に接続し、それから抵抗器を $3.3V$ の電源に接続し、LED の陰極（短いピン）を Raspberry Pi の GPIO17 に接続する。したがって、LED をオンにするには、GPIO17 を低（ $0V$ ）レベルにする必要がある。プログラミングを通じてこの現象を取得できる。

ご注意: Pin11 は Raspberry Pi の左から右への 11 番目のピンを指し、対応する wiringPi と BCM のピン番号は次の表に示されている。

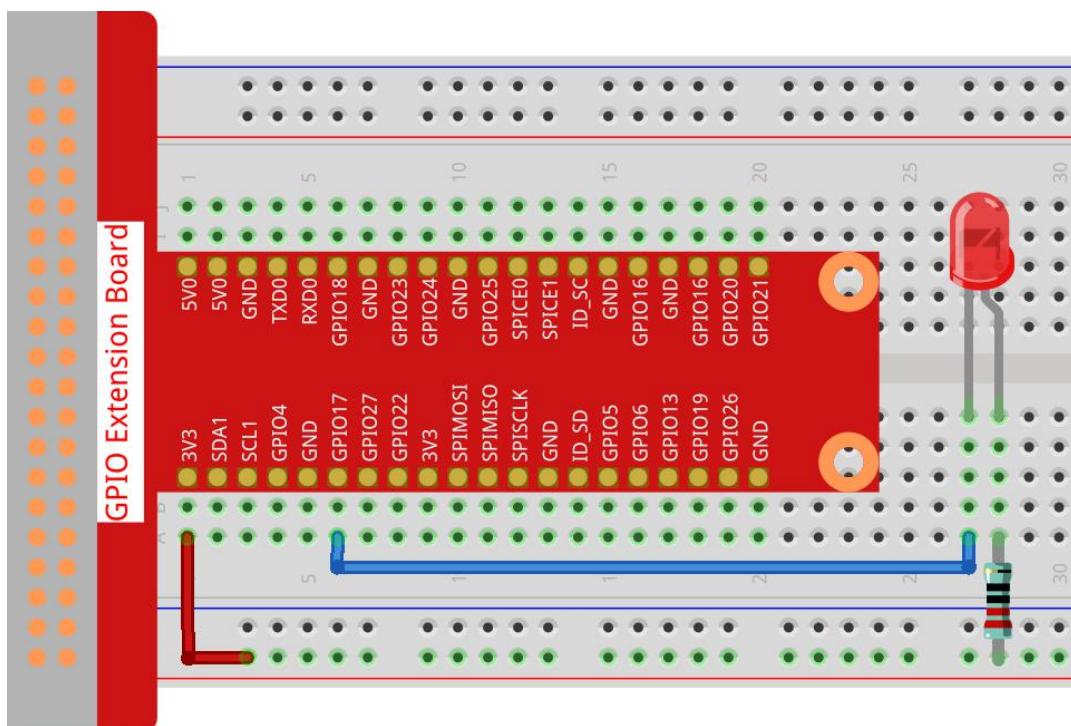
C 言語関連のコンテンツでは、wiringPi で GPIO0 を 0 と同等にする。Python 言語関連のコンテンツの中で、BCM 17 は次の表の BCM 列の 17 である。同時に、それらは Raspberry Pi の 11 番目のピン-ピン 11 と同じである。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ 1: 回路を作る。

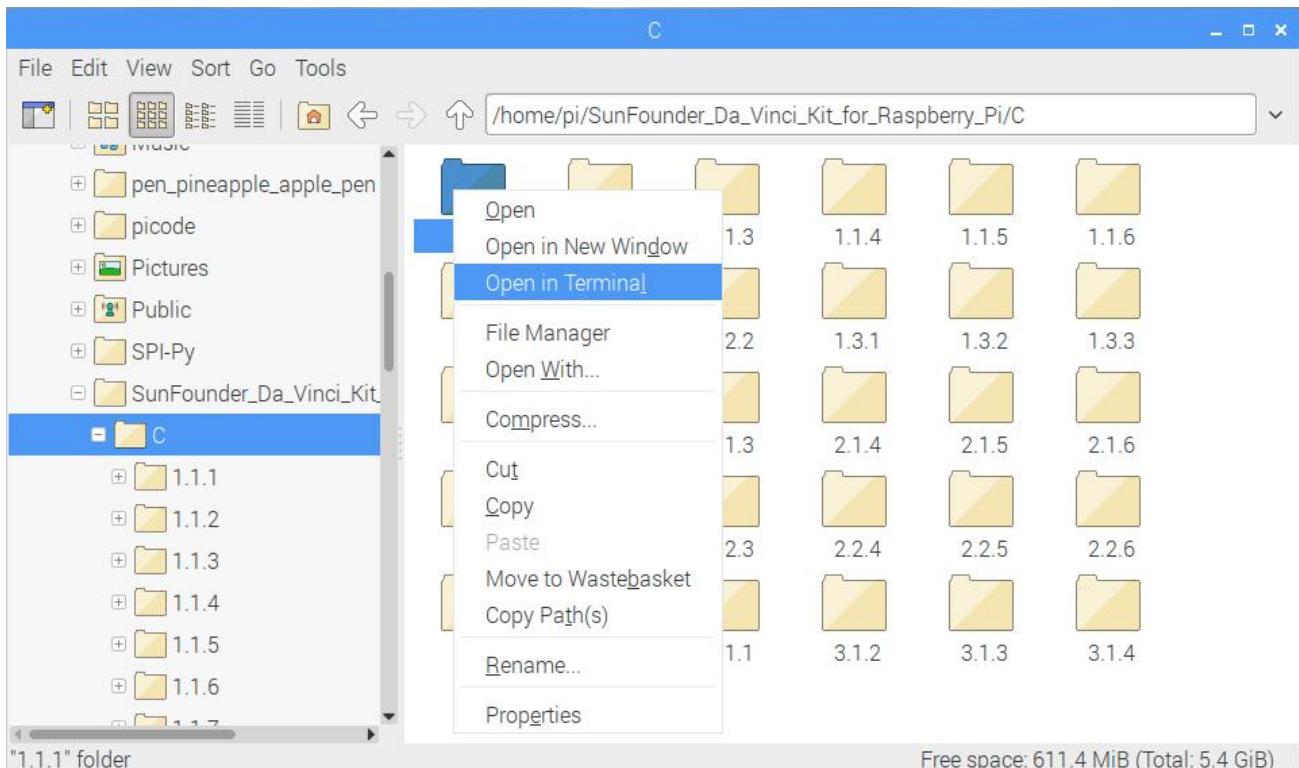


➤ C 言語ユーザー向け

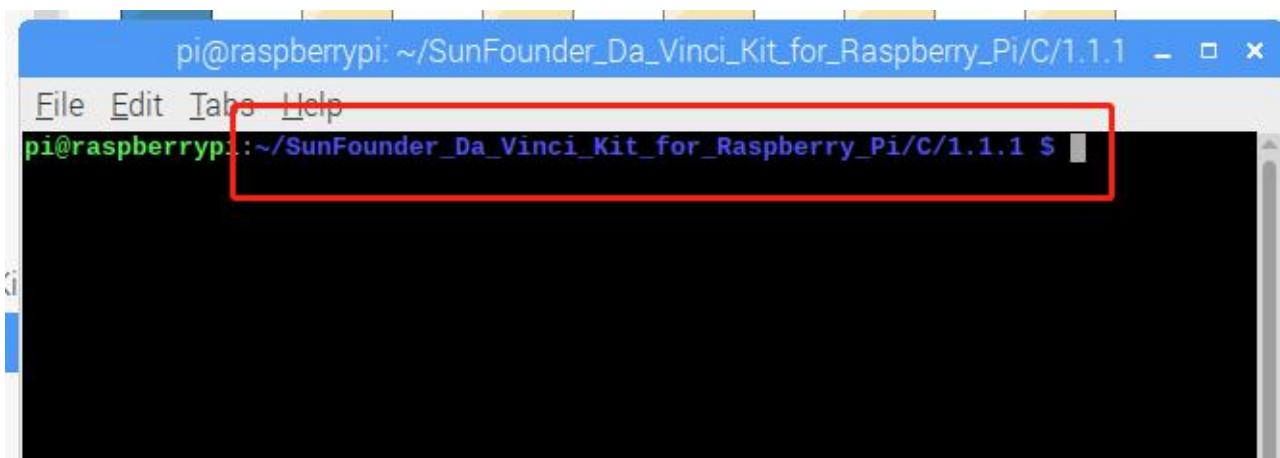
ステップ 2: コードのフォルダーに入る。

モニターを使用する場合は、次の手順を実行することをお勧めする。

/home/pi/に入り、davinci-kit-for-raspberry-pi フォルダーを見つけてください。フォルダ内に C を見つけて右クリックし、「Open in Terminal」を選択する。



それから以下のようなウィンドウがポップアップされる。これで、コード 1.1.1_BlinkingLed.c のパスに入っていた。



次のレッスンでは、右クリックの代わりにコマンドを使用してコードファイルを入力する。ただし、お好みの方法を選択可能である。

Raspberry Pi にリモートでログインする場合、「cd」を使用してディレクトリを変更する：

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.1/
```

ご注意: cd を使用して、この実験のコードのパスにディレクトリを変更してください。

いずれにしても、現在はフォルダー C にいる。これら二つの方法に基づく以降の手順は同じである。次へ移りましょう。

ステップ 3: コードをコンパイルする

```
gcc 1.1.1_BlinkingLed.c -o BlinkingLed -lwiringPi
```

ご注意: gcc は GNU コンパイラコレクションの略語である。ここでは、それは C 言語ファイル 1_BlinkingLed.c をコンパイルして EXE ファイルを出力するように機能している。

コマンドでは、-o は出力を意味し (-o の直後の文字はコンパイル後のファイル名出力であり、BlinkingLed という名前の EXE ファイルがここで生成される) 、 -lwiringPi はライブラリーの wiringPi をロードする (l は library の省略形である) 。

ステップ 4: 前のステップで出力された EXE ファイルを実行する。

```
sudo ./BlinkingLed
```

ご注意: GPIO を制御するには、コマンド sudo (superuser do) でプログラムを実行してください。コマンド "./" は現在のディレクトリを示している。コマンド全体は、現在のディレクトリで BlinkingLed を実行することである。



```
pi@raspberrypi: ~/davinci-kit-for-raspberry-pi/c/1.1.1
pi@raspberrypi: ~ $ cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.1/
pi@raspberrypi:~/davinci-kit-for-raspberry-pi/c/1.1.1 $ gcc 1.1.1_BlinkingLed.c
-o BlinkingLed -lwiringPi
pi@raspberrypi:~/davinci-kit-for-raspberry-pi/c/1.1.1 $ sudo ./BlinkingLed
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
```

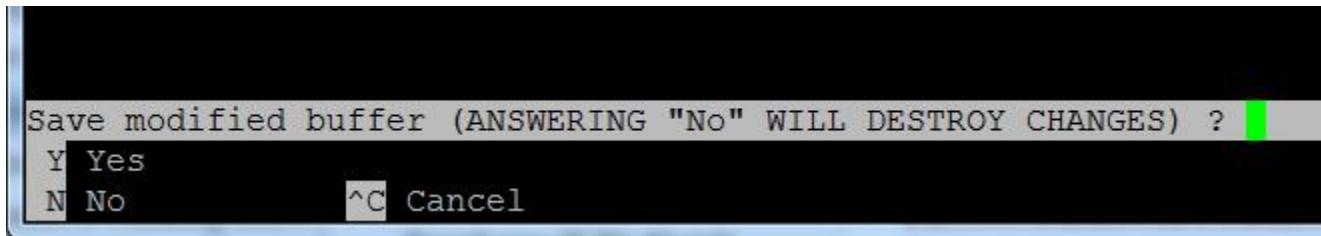
コードの実行後、LED が点滅する。

コードファイル 1.1.1_BlinkingLed.c を編集する場合は、Ctrl + C を押してコードの実行を停止する。次に、次のコマンドを入力してそれを開く：

```
nano 1.1.1_BlinkingLed.c
```

ご注意: nano はテキストエディターツールである。このコマンドは、このツールを通して、コードファイル (1.1.1_BlinkingLed.c) を開くために使用される。

Ctrl + X を押して終了する。コードを修正した場合、変更を保存するかどうかを尋ねるプロンプトが表示される。Y (保存) または N (保存しない) を入力する。次に、Enter キーを押して終了する。ステップ 3 とステップ 4 を繰り返して、修正後の効果を確認する。.



コード

プログラムコードは次のように表示される：

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin      0
int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(LedPin, OUTPUT); // Set LedPin as output to write value to it.
    while(1){
        // LED on
        digitalWrite(LedPin, LOW);
        printf("...LED on\n");
        delay(500);
        // LED off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
        delay(500);
    }
    return 0;
}
```

コードの説明

```
#include <wiringPi.h>
```

ハードウェアドライブライブラリは、Raspberry Pi の C 言語用に設計されている。このライブラリを追加すると、ハードウェアの初期化、および I/O ポート、PWM などの出力に役立つ。

```
#include <stdio.h>
```

標準 I/O ライブラリ。画面に表示されるデータの印刷に使用される printf 機能は、このライブラリによって実現される。他にも多くのパフォーマンス機能がある。

```
#define LedPin 0
```

T_Extension Board の GPIO17 ピンは、wiringPi の GPIO0 に対応している。GPIO0 を LedPin に割り当て、LedPin は後のコードで GPIO0 を表す。

```
if(wiringPiSetup() == -1){  
    printf("setup wiringPi failed !");  
    return 1;
```

これにより、wiringPi が初期化され、呼び出しのプログラムが wiringPi ピン番号スキームを使用することになると想定される。

この関数を呼び出すには、ルート権限が必要である。配線の初期化に失敗すると、画面にメッセージが表示される。関数「return」は現在の関数から飛び出すために使用される。main () 関数で関数「return」を使用すると、プログラムが終了する。

```
pinMode(LedPin, OUTPUT);
```

LedPin を出力として設定し、値を書き込む。

```
digitalWrite(LedPin, LOW);
```

GPIO0 を 0V (低レベル) に設定する。LED の陰極は GPIO0 に接続されているため、GPIO0 が低レベルに設定されると LED が点灯する。それに反して、GPIO0 を高レベルに設定すると (digitalWrite (LedPin, HIGH)): LED が消灯する。

```
printf("...LED off\n");
```

printf 関数は標準ライブラリ関数であり、その関数プロトタイプはヘッダーファイル「stdio.h」にある。呼び出しの一般的な形式は、printf ("format control string", output tables columns) である。フォーマット制御文字列は、出力形式を指定するために使用され、

フォーマット文字列と非フォーマット文字列に分けられる。フォーマット文字列は、「%」で始まり、その後に10進整数出力用の「%d」などのフォーマット文字が続く。フォーマットしていない文字列はプロトタイプとしてプリントされる。ここで使用されているのは非フォーマット文字列で、その後に改行文字「\n」が続き、これは、文字列をプリントした後の自動行の折り返しを表す。

```
delay(500);
```

Delay (500) は、現在の HIGH または LOW 状態を 500ms 維持する。

これは、プログラムを一定期間中断する機能である。また、プログラムの速度はハードウェアによって決まる。ここで、LED をオンまたはオフにする。遅延機能がない場合、プログラムはプログラム全体を非常に高速で実行し、継続的にループする。そのため、プログラムの作成とデバッグに役立つ遅延機能が必要である。

```
return 0;
```

通常、メイン関数の後に配置され、関数が正常に実行されると 0 を返すことを示す。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入り、それを実行する。

モニターを使用する場合は、次の手順を実行することをお勧めする。

1.1.1_BlinkingLed.py を見つけて、ダブルクリックして開く。今、ファイルに入った。ウィンドウで「Run ->Run Module」をクリックすると、次の内容が表示される。

実行を停止するには、右上の「X」ボタンをクリックして閉じるだけで、コードに戻る。コードを変更する場合は、「Run Module (F5)」をクリックする前に、まず保存しなければならない。その後、結果を確認できる。

Raspberry Pi にリモートでログインする場合、次のコマンドを入力する:

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ご注意: cd を使用して、この実験のコードのパスにディレクトリを変更できる。

ステップ 3: コードを実行する

```
sudo python3 1.1.1_BlinkingLed.py
```

ご注意: ここでは sudo-superuser do と python は、Python でファイルを実行することを意味する。

コードの実行後、LED が点滅する。

ステップ4: コードファイル 1.1.1_BlinkingLed.py を編集する場合は、Ctrl + C を押してコードの実行を停止してください。それから次のコマンドを入力して 1.1.1_BlinkingLed.py を開く：

```
nano 1.1.1_BlinkingLed.py
```

ご注意: nano はテキストエディターツールである。このツールは、コマンドを使用してコードファイル 1.1.1_BlinkingLed.py を開く。

Ctrl + X を押して終了する。コードを修正した場合、変更を保存するかどうか尋ねるプロンプトが表示される。Y (保存) または N (保存しない) を入力する。

次に、Enter を押して終了する。変更後の効果を確認するには、nano 1.1.1_BlinkingLed.py をもう一度入力してください。

コード

以下はプログラムコードである：

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
LedPin = 17
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output, and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
# Define a main function for main process
def main():
    while True:
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)
        print ('LED OFF...')
        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)
# Define a destroy function for clean up everything after the script finished
def destroy():
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)
    # Release resource
```

```
GPIO.cleanup()  
# If run this script directly, do:  
if __name__ == '__main__':  
    setup()  
    try:  
        main()  
    # When 'Ctrl+C' is pressed, the program destroy() will be executed.  
    except KeyboardInterrupt:  
        destroy()
```

コードの説明

```
#!/usr/bin/env python3
```

システムがこれを検出すると、env 設定で python の実装パスを検索し、対応するインタープリターを呼び出して操作を完了させる。その目的は、ユーザーが Python を/usr/bin のデフォルトパスに実装することを防止することである。

```
import RPi.GPIO as GPIO
```

この方法で、RPi.GPIO ライブラリをインポートし、変数 GPIO を定義して、次のコードで RPi.GPIO を置き換える。

```
import time
```

次のプログラムの時間遅延機能を行うために、時間パッケージをインポートしなければならない。

```
LedPin = 17
```

LED は T 字型拡張ボードの GPIO17、つまり BCM 17 に接続している。

```
def setup():  
    GPIO.setmode(GPIO.BCM)  
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
```

LedPin のモードを出力に設定し、初期レベルを高 (3.3v) に設定する。

RPi.GPIO 内の Raspberry Pi の IO ピンに番号を付けるには、BOARD と BCM 二つの番号付与方法がある。レッスンでは、使用しているのは BCM 番号である。入力または出力として使用しているチャンネルをすべて設定する必要がある。

```
GPIO.output(LedPin, GPIO.LOW)
```

GPIO17 (BCM17) を 0V (低レベル) に設定する。LED のカソードは GPIO17 に接続されているため、LED が点灯する。

```
time.sleep(0.5)
```

0.5 秒の遅延。ここで、ステートメントは C 言語の遅延機能に似ており、単位は秒である。

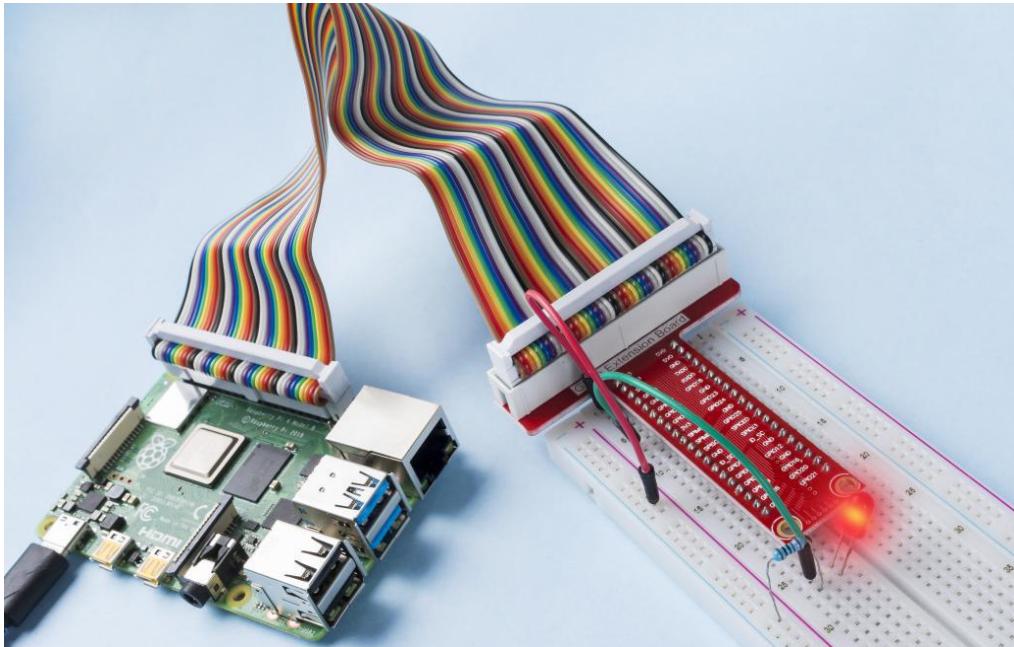
```
def destroy():
    GPIO.cleanup()
```

スクリプトの終了後にすべてを消去する破棄関数を定義する。

```
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

これは、コードの一般的な実行構造である。プログラムの実行が開始されると、setup () を実行してピンを初期化し、main () 関数でコードを実行してピンを高レベルと低レベルに設定する。「Ctrl + C」を押すと、プログラム destroy () が実行される。

現象画像

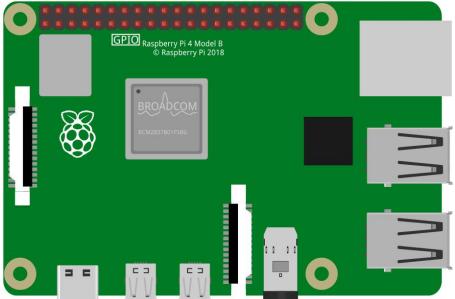
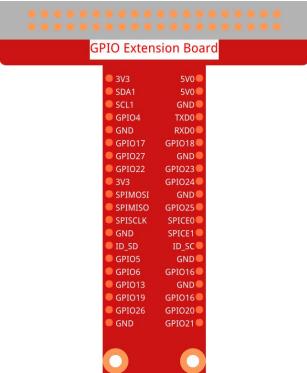
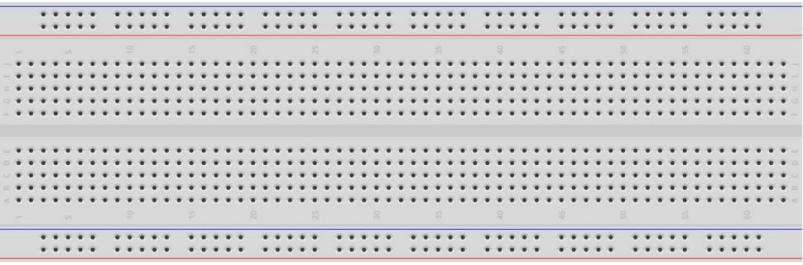


1.1.2 RGB LED

前書き

このレッスンでは、これを使用して RGB LED を制御し、さまざまな種類の色を点滅させる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	RGB LED*1																																																																	
	 <table border="1"> <tr><th></th><th>3V3</th><th>SDA1</th><th>SCL1</th><th>GND</th><th>GPIO4</th><th>TXD0</th><th>GND</th><th>GPIO17</th><th>GPIO18</th><th>GPIO27</th><th>GPIO22</th><th>3V5</th><th>SPI_MOSI</th><th>GND</th><th>SPI_MISO</th><th>GND</th><th>SPI_SCLK</th><th>GND</th><th>ID_SD</th><th>ID_SC</th><th>GPIO5</th><th>GND</th><th>GPIO6</th><th>GPIO16</th><th>GPIO13</th><th>GPIO16</th><th>GPIO20</th><th>GND</th><th>GPIO2</th><th>GPIO21</th></tr> <tr><td>GPIO</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>		3V3	SDA1	SCL1	GND	GPIO4	TXD0	GND	GPIO17	GPIO18	GPIO27	GPIO22	3V5	SPI_MOSI	GND	SPI_MISO	GND	SPI_SCLK	GND	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GPIO16	GPIO20	GND	GPIO2	GPIO21	GPIO																																		
	3V3	SDA1	SCL1	GND	GPIO4	TXD0	GND	GPIO17	GPIO18	GPIO27	GPIO22	3V5	SPI_MOSI	GND	SPI_MISO	GND	SPI_SCLK	GND	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GPIO16	GPIO20	GND	GPIO2	GPIO21																																					
GPIO																																																																			
40 ピンケーブル*1		何本のジャンパー線																																																																	
																																																																			
ブレッドボード*1		抵抗器 (220Ω) *3																																																																	
																																																																			

原理

PWM

パルス幅変調 (PWM) は、デジタル手段でアナログ結果を取得するための技術である。デジタル制御は、オンとオフを切り替える信号である方形波を作成するために使用される。このオン/オフパターンは、信号がオンになる時間と信号がオフになる時間の部分を変更することにより、完全にオン (5 ボルト) である時と完全にオフ (0 ボルト) である時の間の電圧をシミュレートできる。「オンタイム」の期間はパルス幅と呼ばれる。さまざまなアナログ値を取得するには、その幅を変更または変調できる。このオン/オフパターン

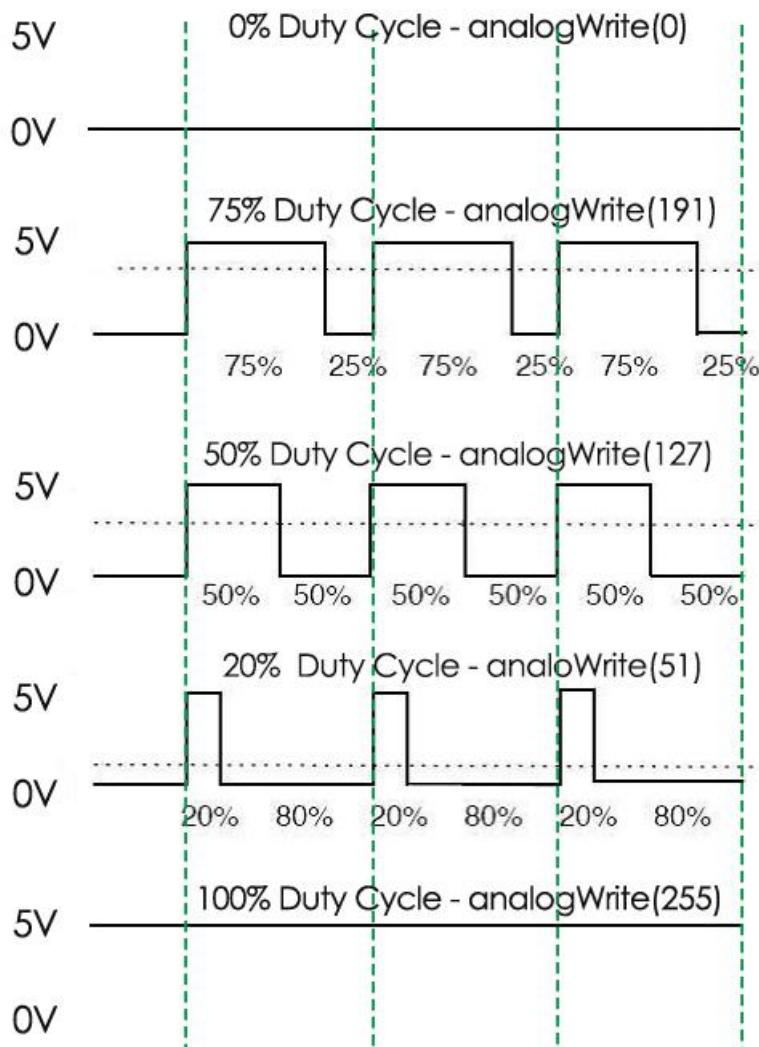
を何らかのデバイス、たとえば LED で十分に速く繰り返すと、結果は次のようになる：信号は LED の輝度を制御する 0~5v の安定した電圧である。

デューティサイクル

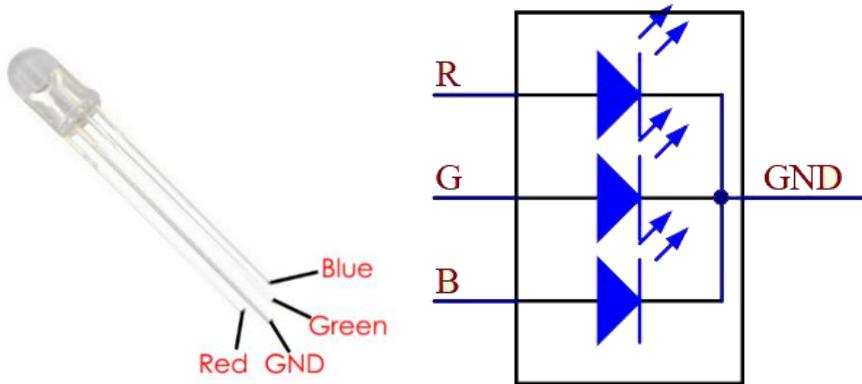
デューティサイクルは、信号が有効である 1 周期の割合である。期間とは、信号がオンとオフのサイクルを完了するのにかかる時間である。式として、デューティサイクルは次のように表示される：

$$D = \frac{T}{P} \times 100\%$$

ここで、D はデューティサイクル、T は信号の有効時間、P は信号の合計周期である。したがって、60%のデューティサイクルは、信号が時間の 60%にわたってオンであるが、時間の 40%にわたってオフであることを意味する。60%のデューティサイクルの「オン時間」は、期間の長さに応じて、1秒、1日、または 1 週間の分数である場合がある。



RGB LED

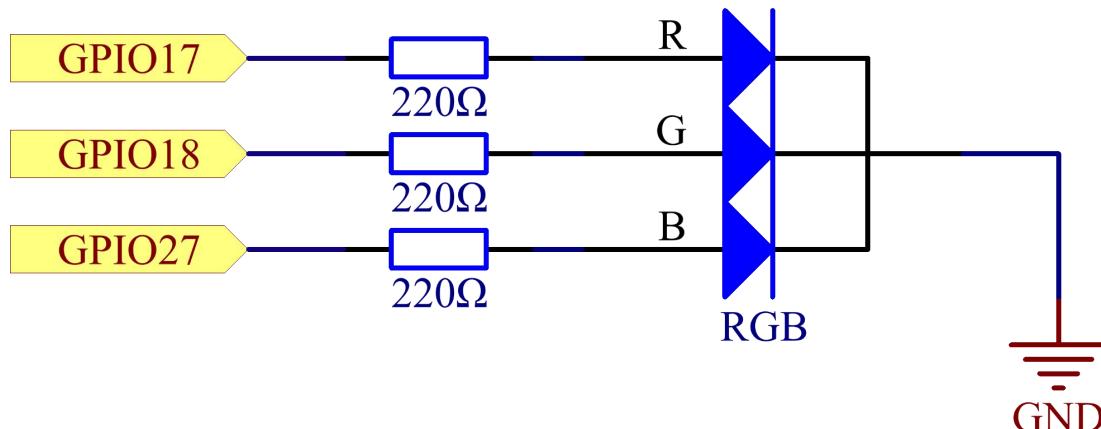


RGB LED の 3 つの原色は、輝度によってさまざまな色に混ぜることができる。LED の輝度は PWM で調整可能である。Raspberry Pi にはハードウェア PWM 出力用のチャネルが 1 つしかないが、RGB LED を制御するには 3 つのチャネルが必要である。つまり、Raspberry Pi のハードウェア PWM で RGB LED を制御することは困難である。幸いなことに、softPwm ライブラリはプログラミングによって PWM (softPwm) をシミュレートする。ヘッダーファイル softPwm.h (C 言語ユーザー向け) をインクルードし、提供される API を呼び出して、マルチチャネル PWM 出力によって RGB LED を簡単に制御するだけで、あらゆる種類の色を表示できる。

回路図

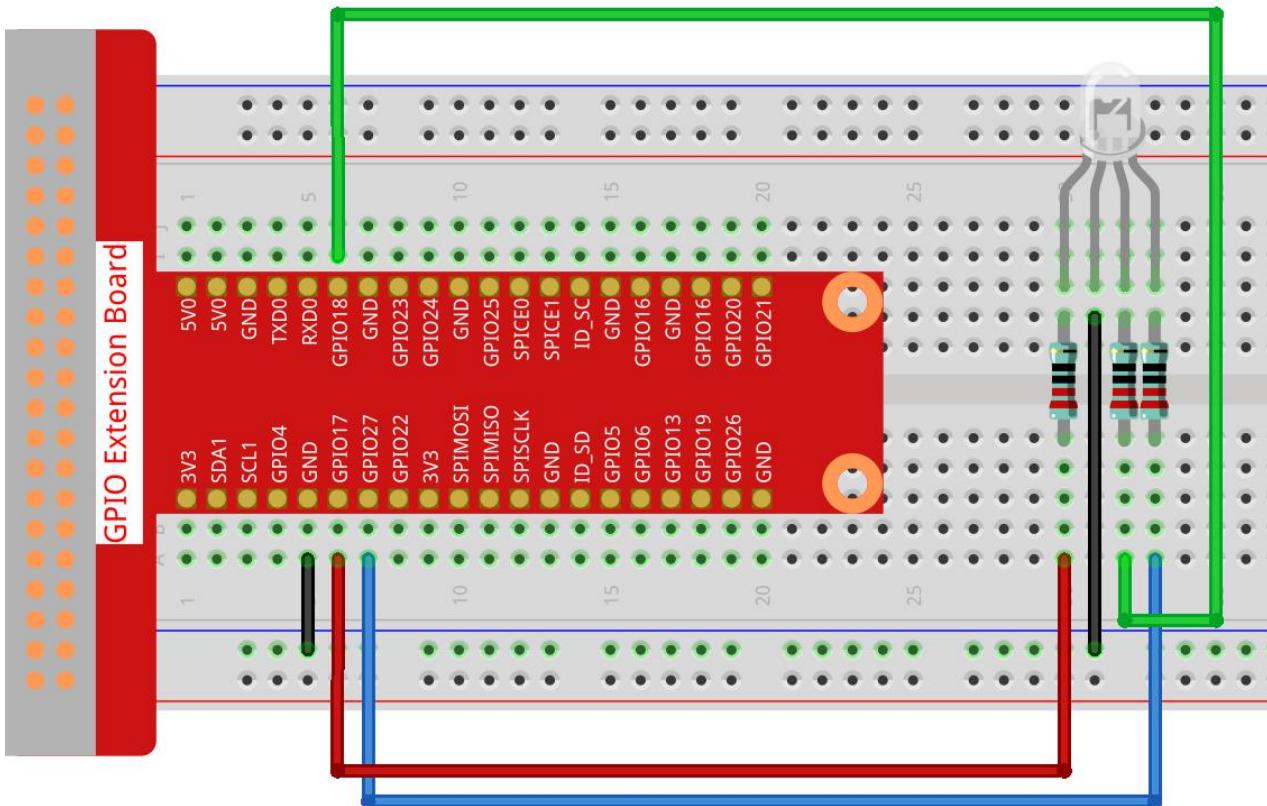
R、G、および B のピンを電流制限抵抗器に接続した後、それぞれ GPIO17、GPIO18、および GPIO27 に接続してください。LED の最も長いピン (GND) は、Raspberry Pi の接地に接続する。3 つのピンに異なる PWM 値が与えられると、RGB LED は異なる色を表示する。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.2/
```

ステップ 3: コードをコンパイルする。

```
gcc 1.1.2_rgbLed.c -lwiringPi
```

ご注意: 命令「gcc」が実行されるときに「-o」が呼び出されない場合、EXE ファイルは「a.out」と命名する。

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードを実行すると、RGB が赤、緑、青、黄色、ピンク、およびシアンを表示する。

コード

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>

#define uchar unsigned char
#define LedPinRed    0
#define LedPinGreen   1
#define LedPinBlue   2

void ledInit(void){
    softPwmCreate(LedPinRed,  0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
}

void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(LedPinRed,   r_val);
    softPwmWrite(LedPinGreen, g_val);
    softPwmWrite(LedPinBlue,  b_val);
}

int main(void){

    if(wiringPiSetup() == -1){ //when initialize wiring failed, printf message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    ledInit();
    while(1){
        printf("Red\n");
        ledColorSet(0xff,0x00,0x00);    //red
        delay(500);
        printf("Green\n");
        ledColorSet(0x00,0xff,0x00);    //green
        delay(500);
        printf("Blue\n");
        ledColorSet(0x00,0x00,0xff);    //blue
        delay(500);
        printf("Yellow\n");
        ledColorSet(0xff,0xff,0x00);    //yellow
    }
}
```

```

delay(500);
printf("Purple\n");
ledColorSet(0xff,0x00,0xff); //purple
delay(500);
printf("Cyan\n");
ledColorSet(0xc0,0xff,0x3e); //cyan
delay(500);
}
return 0;
}

```

コードの説明

```
#include <softPwm.h>
```

ソフトウェアの pwm 機能を実現するために使用されるライブラリ。

```

void ledInit(void){
    softPwmCreate(LedPinRed, 0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
}

```

この機能は、ソフトウェアを使用して PWM ピンを作成し、その周期を 0x100us ~ 100x100us に設定することである。

関数 softPwmCreate (LedPinRed、0、100) のプロトタイプは次のとおりである：

```
int softPwmCreate(int pin,int initialValue,int pwmRange);
```

Parameter pin: Raspberry Pi の GPIO ピンは、PWM ピンとして設定できる。

Parameter initialValue: 初期パルス幅は initialValue に 100us を掛けたものである。

Parameter pwmRange: PWM の周期は、pwmRange に 100us を掛けたものである。

```

void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(LedPinRed, r_val);
    softPwmWrite(LedPinGreen, g_val);
    softPwmWrite(LedPinBlue, b_val);
}

```

この機能は LED の色を設定する。RGB を使用すると、仮パラメータは赤の輝度の r_val、緑の輝度の g_val、青の輝度の b_val を表す。

関数 softPwmWrite(LedPinBlue, b_val)のプロトタイプは次のとおりである：

```
void softPwmWrite (int pin, int value);
```

Parameter pin: Raspberry Pi の GPIO ピンは、PWM ピンとして設定できる。

Parameter Value: PWM のパルス幅は value に 100us を掛けたものである。値は以前に定義された pwmRange よりも小さくすることができ、pwmRange よりも大きい場合、値には固定値 pwmRange が与えられることに注意してください。

```
ledColorSet(0xff,0x00,0x00);
```

前に定義した関数を呼び出す。LedPinRed に 0xff を、LedPinGreen と LedPinBlue に 0x00 を書き込む。このコードを実行すると、赤色の LED のみが点灯する。他の色の LED を点灯させる場合は、パラメーターを変更するだけである。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: 実行する。

```
sudo python3 1.1.2_rgbLed.py
```

コードを実行すると、RGB が赤、緑、青、黄色、ピンク、およびシアンを表示する。

コード

```
import RPi.GPIO as GPIO
import time

# Set up a color table in Hexadecimal
COLOR = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]

# Set pins' channels with dictionary
pins = {'Red':17, 'Green':18, 'Blue':27}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    for i in pins:
        GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.HIGH)

    p_R = GPIO.PWM(pins['Red'], 2000)
    p_G = GPIO.PWM(pins['Green'], 2000)
    p_B = GPIO.PWM(pins['Blue'], 2000)
    p_R.start(0)
    p_G.start(0)
```

```

p_B.start(0)

# Define a MAP function for mapping values. Like from 0~255 to 0~100
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

# Define a function to set up colors
def setColor(color):
    # configures the three LEDs' luminance with the inputted color value.
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0

    # Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

    # Change the colors
    p_R.ChangeDutyCycle(R_val)
    p_G.ChangeDutyCycle(G_val)
    p_B.ChangeDutyCycle(B_val)

    print ("color_msg: R_val = %s, G_val = %s, B_val = %s"%(R_val, G_val, B_val))

def main():
    while True:
        for color in COLOR:
            setColor(color)# change the color of the RGB LED
            time.sleep(0.5)

def destroy():
    # Stop all pwm channel
    p_R.stop()
    p_G.stop()
    p_B.stop()
    # Turn off all LEDs
    GPIO.output(pins, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

```

```
if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
p_R = GPIO.PWM(pins['Red'], 2000)
p_G = GPIO.PWM(pins['Green'], 2000)
p_B = GPIO.PWM(pins['Blue'], 2000)

p_R.start(0)
p_G.start(0)
p_B.start(0)
```

GPIO.PWM() 関数を呼び出して、赤、緑、青を PWM ピンとして定義し、PWM ピンの周波数を 2000Hz に設定してから、Start() 関数を使用して初期デューティサイクルをゼロに設定する。

```
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

値をマッピングするための MAP 関数を定義する。たとえば、 $x = 50$ 、 $in_min = 0$ 、 $in_max = 255$ 、 $out_min = 0$ 、 $out_max = 100$ 。マップ関数のマッピング後、 $(50-0) * (100-0) / (255-0) + 0 = 19.6$ を戻す。つまり、0-255 の 50 は 0-100 の 19.6 に相当する。

```
def setColor(color):
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0
```

入力されたカラー値で三つの LED の輝度を構成し、16 進数の最初の二つの値を R_val に割り当て、中央の二つを G_val に割り当て、最後の二つの値を B_val に割り当てる。たとえば、 $color = 0xFF00FF$ 、 $R_val = (0xFF00FF \& 0xFF0000) >> 16 = 0xFF$ 、 $G_val = 0x00$ 、 $B_val = 0xFF$ の場合。

```
R_val = MAP(R_val, 0, 255, 0, 100)
G_val = MAP(G_val, 0, 255, 0, 100)
B_val = MAP(B_val, 0, 255, 0, 100)
```

マップ機能を使用して、0~255 の R、G、B 値を PWM デューティサイクル範囲 0~100 にマップする。

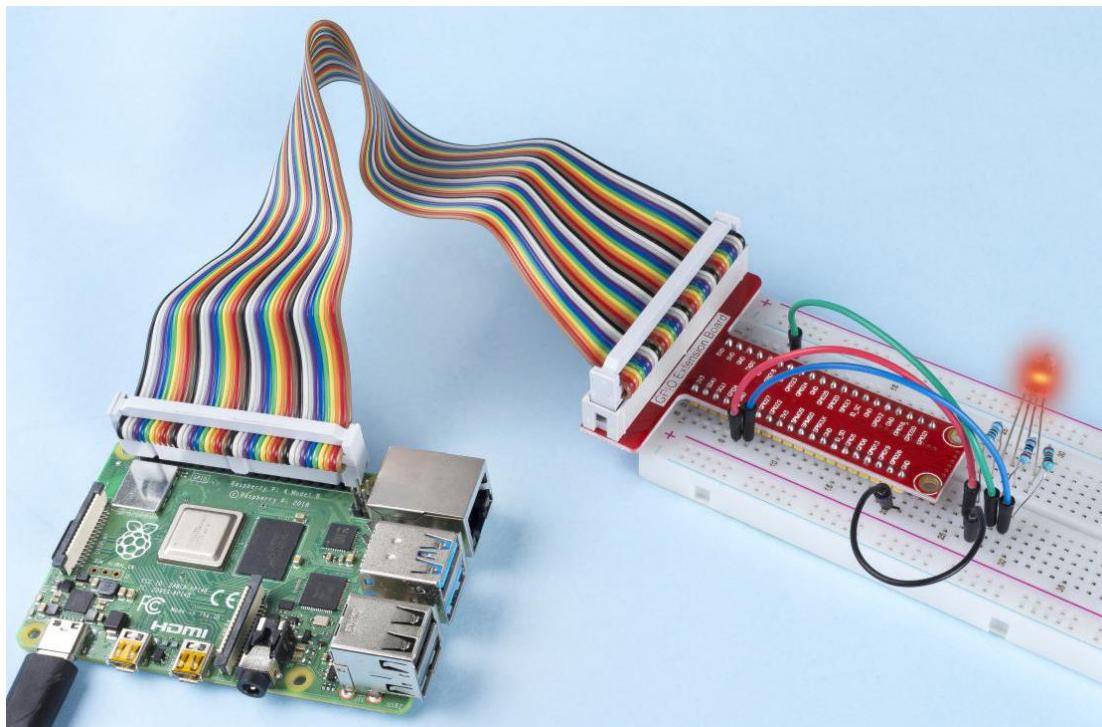
```
p_R.ChangeDutyCycle(R_val)  
p_G.ChangeDutyCycle(G_val)  
p_B.ChangeDutyCycle(B_val)
```

マッピングされたデューティサイクルを対応する PWM チャネルに割り当てて、輝度を変更する。

```
for color in COLOR:  
    setColor(color)  
    time.sleep(0.5)
```

COLOR リストのすべてのアイテムをそれぞれ色に割り当て、setColor() 関数を介して RGB LED の色を変更する。

現象画像

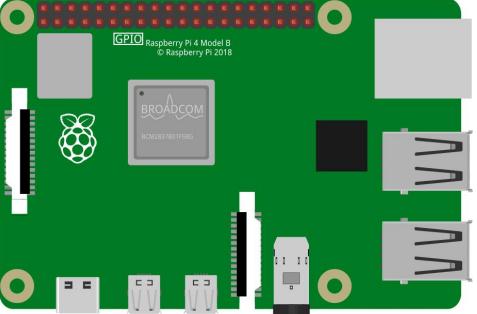
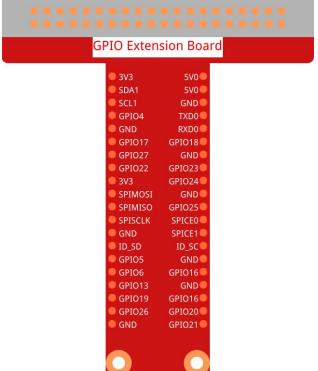
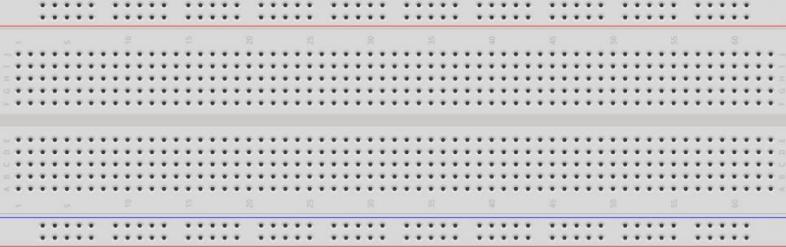


1.1.3 LED Bar Graph

前書き

このプロジェクトでは、LED 棒グラフのライトを順番に点灯させる。

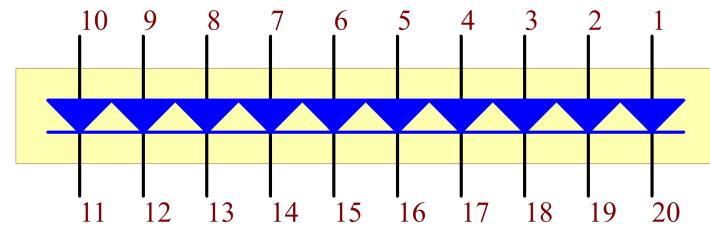
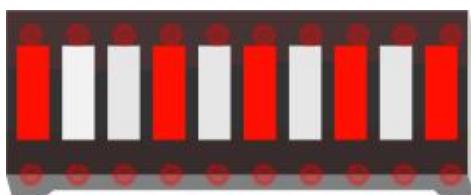
部品

Raspberry Pi 本体*1	T 拡張ボード*1	LED 棒グラフ*1
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">3V3SDA1SCL1GPIO4GNDEXD0GPIO17GPIO27GPIO223V2SPIMOSISPISCLKGNDID_SDGPIO5GPIO6GPIO13GPIO19GPIO26GNDSDA1SCL1GPIO4TXD0GNDEXD0GPIO17GPIO27GPIO22GPIO34GNDGPIO25SPICE0SPICE1GNDID_SCGPIO16GPIO10GPIO16GPIO20GPIO21GND	
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		抵抗器 (220Ω) *1
		

原理

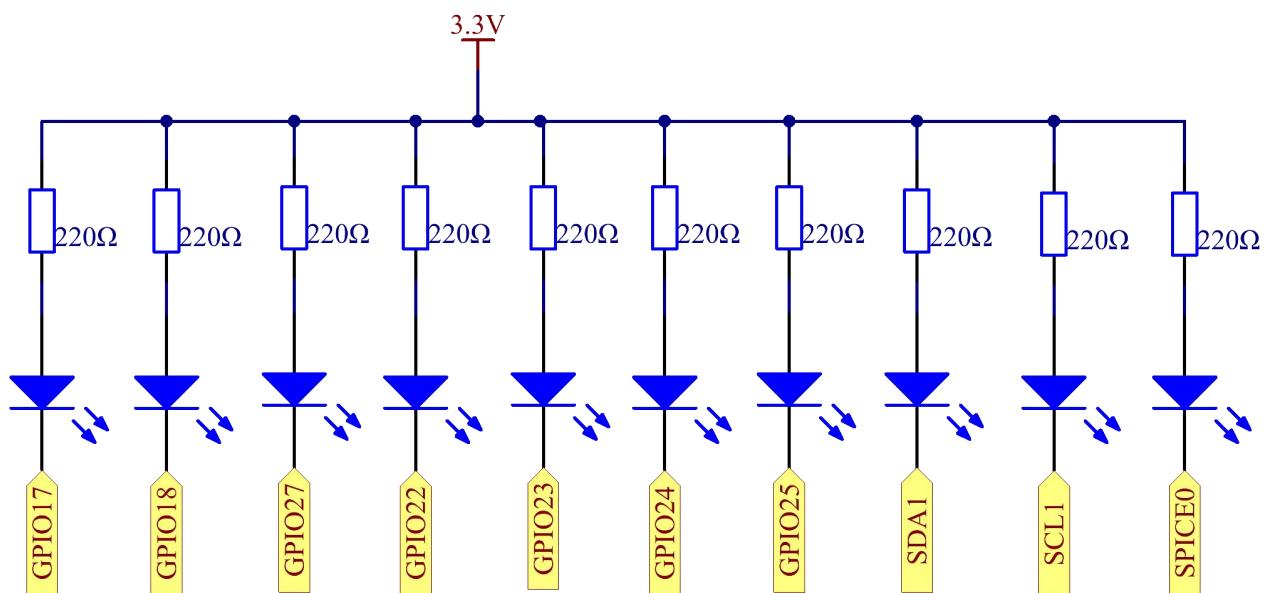
LED 棒グラフ

LED 棒グラフは、電子回路またはマイクロコントローラーとの接続に使用される LED 配列である。10 個の個別の LED を 10 本の出力ピンに接続するように、LED 棒グラフを回路に簡単に接続できる。通常、LED 棒グラフは、バッテリーレベルインジケーター、オーディオ機器、および産業用制御パネルとして使用できる。LED 棒グラフには他にも多くの用途がある。



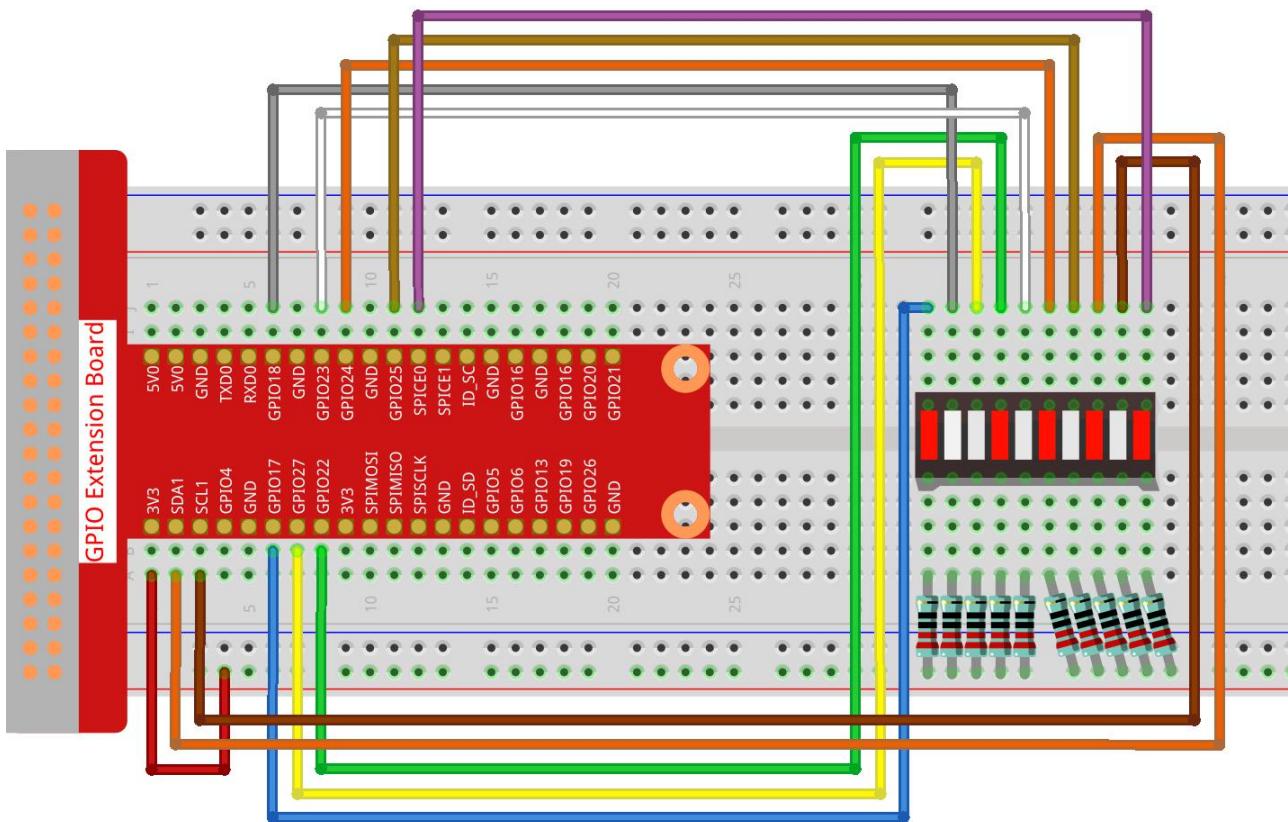
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3	8	2
SCL1	Pin 5	9	3
SPICE0	Pin 24	10	8



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd ~/davinci-kit-for-raspberry-pi/c/1.1.3/
```

ステップ 3: コードをコンパイルする。

```
gcc 1.1.3_LedBarGraph.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、LED バーの LED が定期的にオン/オフになる。

コード

```
#include <wiringPi.h>
#include <stdio.h>

int pins[10] = {0,1,2,3,4,5,6,8,9,10};
void oddLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2;
        digitalWrite(pins[j],HIGH);
    }
}
```

```

        delay(300);
        digitalWrite(pins[j],LOW);
    }
}

void evenLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2+1;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}

void allLedBarGraph(void){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
        delay(300);
        digitalWrite(pins[i],LOW);
    }
}

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to
screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(int i=0;i<10;i++){      //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],LOW);
    }
    while(1){
        oddLedBarGraph();
        delay(300);
        evenLedBarGraph();
        delay(300);
        allLedBarGraph();
        delay(300);
    }
    return 0;
}

```

コードの説明

```
int pins[10] = {0,1,2,3,4,5,6,8,9,10};
```

配列を作成し、LED 棒グラフ (0、1、2、3、4、5、6、8、9、10) に対応するピン番号に割り当て、配列は LED を制御するために使用される。

```
void oddLedBarGraph(void){  
    for(int i=0;i<5;i++){  
        int j=i*2;  
        digitalWrite(pins[j],HIGH);  
        delay(300);  
        digitalWrite(pins[j],LOW);  
    }  
}
```

LED 棒グラフの奇数桁の LED を順番に点灯させる。

```
void evenLedBarGraph(void){  
    for(int i=0;i<5;i++){  
        int j=i*2+1;  
        digitalWrite(pins[j],HIGH);  
        delay(300);  
        digitalWrite(pins[j],LOW);  
    }  
}
```

LED 棒グラフの偶数桁の LED を順番に点灯させる。

```
void allLedBarGraph(void){  
    for(int i=0;i<10;i++){  
        digitalWrite(pins[i],HIGH);  
        delay(300);  
        digitalWrite(pins[i],LOW);  
    }  
}
```

LED 棒グラフの LED を 1 つずつ点灯させる。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: Run the executable file.

```
sudo python3 1.1.3_LedBarGraph.py
```

コードの実行後、LED 棒の LED が定期的にオン/オフになる。

コード

```
import RPi.GPIO as GPIO
import time

ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]

def oddLedBarGraph():
    for i in range(5):
        j = i*2
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

def allLedBarGraph():
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(i,GPIO.LOW)

def setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BRD)          # Numbers GPIOs by physical location
    for i in ledPins:
        GPIO.setup(i, GPIO.OUT)     # Set all ledPins' mode is output
        GPIO.output(i, GPIO.LOW)    # Set all ledPins to high(+3.3V) to off led

def loop():
    while True:
        oddLedBarGraph()
        time.sleep(0.3)
        evenLedBarGraph()
        time.sleep(0.3)
        allLedBarGraph()
```

```

    time.sleep(0.3)

def destroy():
    for pin in ledPins:
        GPIO.output(pin, GPIO.LOW)      # turn off all leds
    GPIO.cleanup()                      # Release resource

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will
be executed.
    destroy()

```

コードの説明

```
ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]
```

配列を作成し、LED 棒グラフ (11、12、13、15、16、18、22、3、5、24) に対応するピン番号に割り当て、配列は LED を制御するために使用される。

```

def oddLedBarGraph():
    for i in range(5):
        j = i*2
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

```

LED 棒グラフの奇数桁の LED を順番に点灯させる。

```

def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

```

LED 棒グラフの偶数桁の LED を順番に点灯させる。

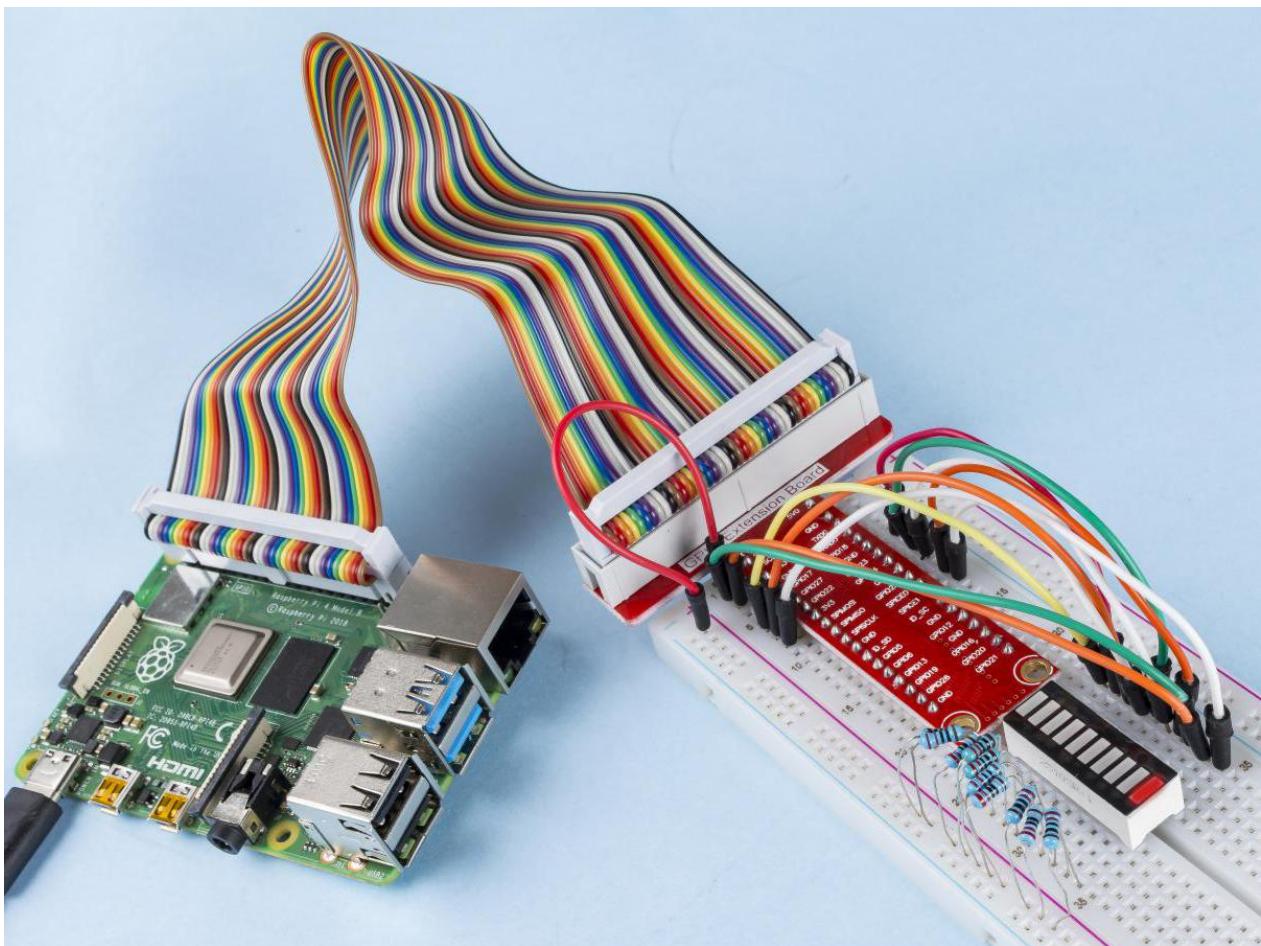
```

def allLedBarGraph():
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(i,GPIO.LOW)

```

LED 棒グラフの LED を 1 つずつ点灯させる。

現象画像

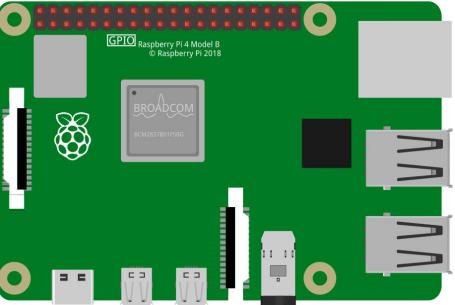
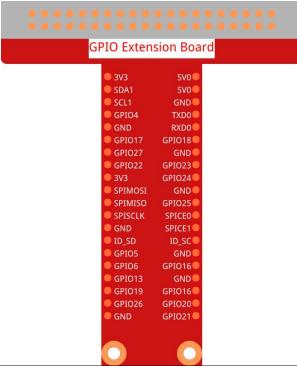


1.1.4 7-segment Display

前書き

7セグメントディスプレイを駆動して、0から9およびAからFの数字を表示してみましょう。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	7セグメントディスプレイ *1
		
40ピンケーブル*1		抵抗器 (220Ω) *1 
		何本のジャンパー線 
ブレッドボード*1		74HC595*1 

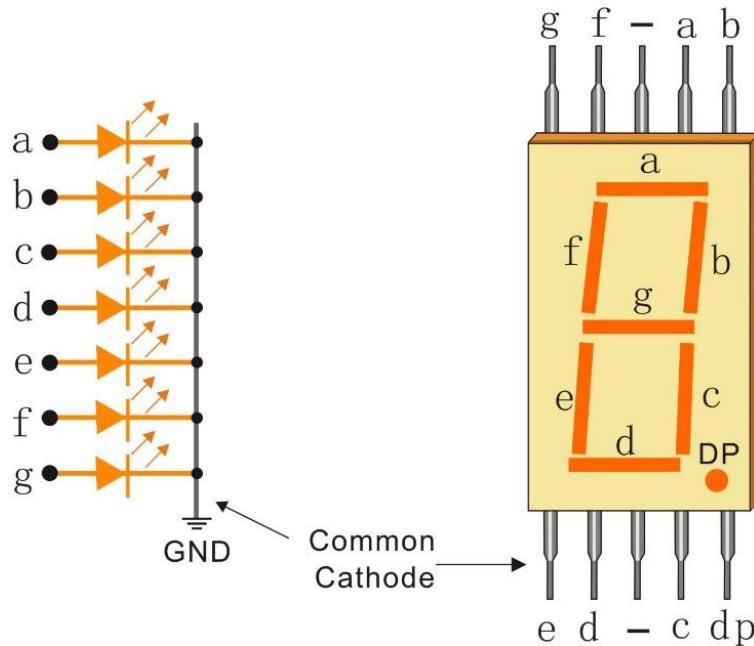
原理

7セグメントディスプレイ

7セグメントディスプレイは、LEDを7個パッケージした8字型の部品である。各LEDはセグメントと呼ばれる-通電されると、1つのセグメントが表示される数字の一部を形成する。

ピン接続には、カソードコモン (CC) とアノードコモン (CA) の2種類がある。名前が示すように、CAディスプレイには7セグメントのアノードがすべて接続されている場合、CC

ディスプレイには 7 つの LED のカソードがすべて接続されている。このキットでは、前者を使用する。



ディスプレイの各 LED には、長方形のプラスチックパッケージから接続ピンの 1 つが引き出された位置セグメントがある。これらの LED ピンには、個々の LED を表す「a」から「g」までのラベルが付いている。他の LED ピンは一緒に接続され、共通のピンを形成する。そのため、LED セグメントの適切なピンを特定の順序で順方向にバイアスすることにより、一部のセグメントが明るくなり、他のセグメントが暗くなり、ディスプレイに対応する文字が表示される。

表示コード

7 セグメントディスプレイ（カソードコモン）がどのように番号を表示するかを知るために、次の表を作成した。数字は、7 セグメントディスプレイに表示される数字 0～F である。（DP）GFEDCBA は、0 または 1 に設定された対応する LED を指す。たとえば、00111111 は、DP と G が 0 に設定され、他が 1 に設定されることを意味する。したがって、7 セグメントディスプレイには 0 が表示され、HEX コードは 16 進数に対応する。

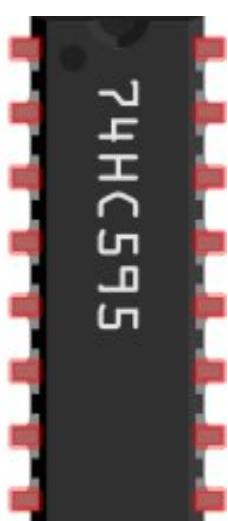
数字	カソードコモン		数字	カソードコモン	
	(DP)GFEDCBA	16 進コード		(DP)GFEDCBA	16 進コード
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39

3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d			
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

74HC595

74HC595 は、8 ビットのシフトレジスタと、3 段階の並列出力を備えたストレージレジスタで構成されている。MCU の IO ポートを節約できるように、シリアル入力を並列出力に変換する。

MR (ピン 10) が高レベルで、OE (ピン 13) が低レベルの場合、データは SHcp の立ち上がりエッジで入力され、SHcp の立ち上がりエッジを介してメモリレジスタに入力される。2 つのクロックが接続されている場合、シフトレジスタは常にメモリレジスタより 1 パルス早くなる。メモリレジスタには、シリアルシフト入力ピン (Ds) 、シリアル出力ピン (Q) 、非同期リセットボタン (低レベル) がある。メモリレジスタは並列 8 ビットで 3 つの状態のバスを出力します。OE が有効 (低レベル) の場合、メモリレジスタのデータがバスに出力される。



1	Q1	VCC	16
2	Q2	Q0	15
3	Q3	DS	14
4	Q4	CE	13
5	Q5	STcp	12
6	Q6	SHcp	11
7	Q7	MR	10
8	GND	Q7'	9

74HC595 のピンとその機能:

Q0-Q7: 8 ビットの並列データ出力ピン-8 個の LED または 7 セグメントディスプレイの 8 個のピンを直接制御できる。

Q7': 直列出力ピン-ほかの 74HC595 の DS を直列に複数の 74HC595 の DS に接続された。

MR: リセットピン - 低レベルで作動。

SHcp: シフトレジスタの時系列入力。立ち上がりエッジで、シフトレジスタのデータは連続して 1 ビット移動する。つまり、Q1 のデータは Q2 に移動し、以下同様に続く。立ち下がりエッジでは、シフトレジスタのデータは変更しない。

STcp: ストレージレジスタの時系列入力。立ち上がりエッジで、シフトレジスタのデータがメモリレジスタに移動する。

CE: 出力イネーブルピン、低レベルで作動する。

DS: 直列データ入力ピン

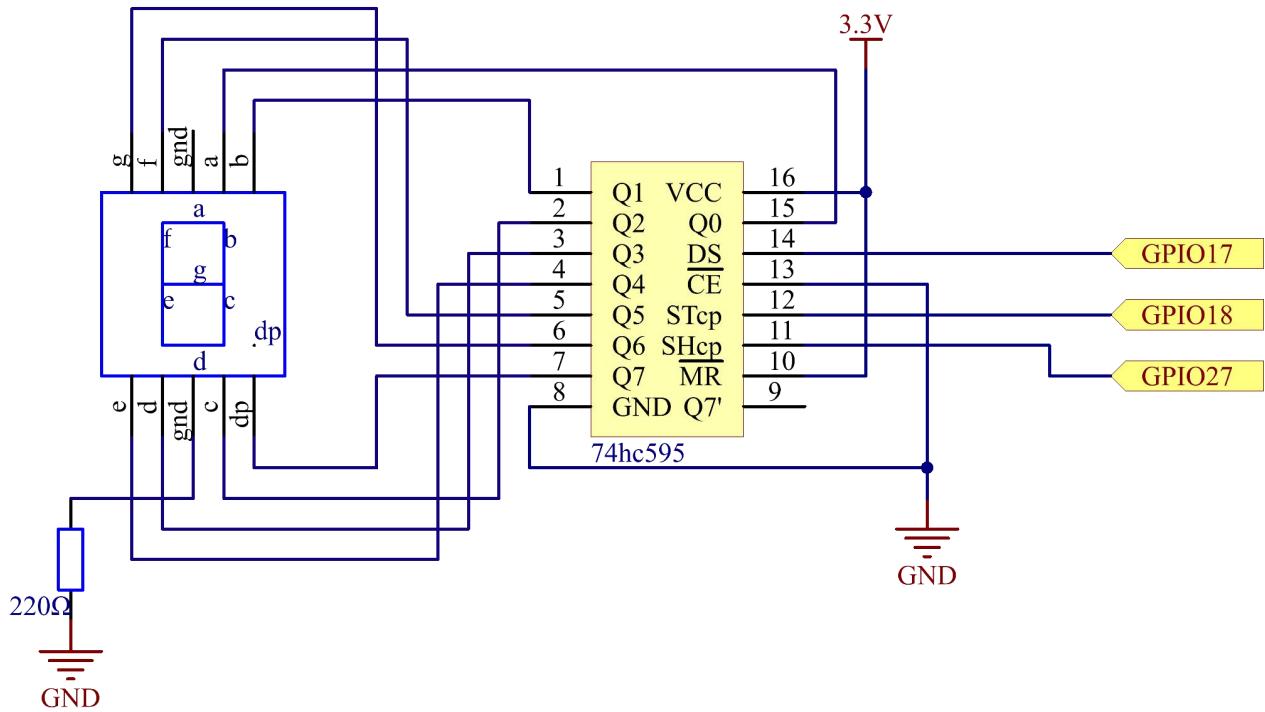
VCC: 正の電源電圧

GND: 接地

回路図

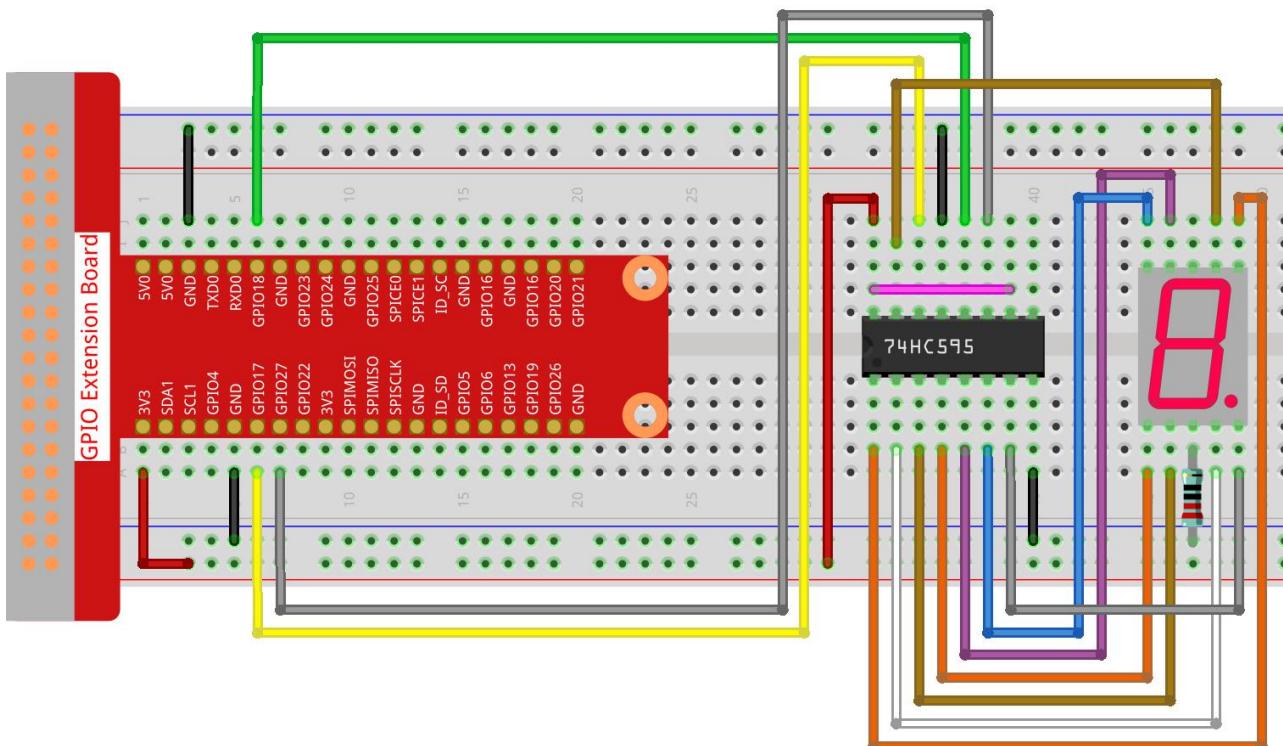
74HC595 のピン ST_CP を Raspberry Pi GPIO18 に、SH_CP を GPIO27 に、DS を GPIO17 に、並列出力ポートを LED セグメントディスプレイの 8 つのセグメントに接続する。SH_CP (シフトレジスタのクロック入力) が立ち上がりエッジにある場合は DS ピンのデータをシフトレジスタに入力し、ST_CP (メモリのクロック入力) が立ち上がりエッジにある場合はそのデータをメモリレジスタに入力する。次に、Raspberry Pi GPIO を介して SH_CP および ST_CP の状態を制御し、直列データ入力を並列データ出力に変換して、Raspberry Pi GPIO を保存したりディスプレイを駆動したりすることはできる。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.4/
```

ステップ3: コンパイルする。

```
gcc 1.1.4_7-Segment.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、7セグメントディスプレイに0~9、AFが表示される。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)
unsigned char SegCode[16] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

void hc595_shift(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
    }
    digitalWrite(RCLK, 1);
    delay(1);
    digitalWrite(RCLK, 0);
}
```

```

}

int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<16;i++){
            printf("Print %1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
    return 0;
}

```

コードの説明

```

unsigned char SegCode[16] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};

```

16進数（カソードコモン）の0からFまでのセグメントコード配列。

```

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

```

ds、st_cp、sh_cp の3つのピンをOUTPUTに設定し、初期状態を0に設定する。

```

void hc595_shift(unsigned char dat){}

```

8ビット値を74HC595のシフトレジスタに割り当てる。

```

digitalWrite(SDI, 0x80 & (dat << i));

```

ビットごとに dat データを SDI (DS) に割り当てる。ここでは、dat = 0x3f (0011 1111) を仮定し、i = 2 の場合、0x3f は左 (<<) 2 ビットにシフトする。1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000、真である。

```
digitalWrite(SRCLK, 1);
```

SRCLK の初期値は元々に 0 に設定されていたが、ここでは 1 に設定されている。これは、立ち上がりエッジパルスを生成し、DS の日付をシフトレジスタにシフトする。

```
digitalWrite(RCLK, 1);
```

RCLK の初期値は元々に 0 に設定されていたが、ここでは 1 に設定されている。これは、立ち上がりエッジパルスを生成し、データーをシフトレジスタからストレージレジスターにシフトする。

```
while(1){
    for(i=0;i<16;i++){
        printf("Print %1X on Segment\n", i); // %X means hex output
        hc595_shift(SegCode[i]);
        delay(500);
    }
}
```

.この for 文では、「%1X」を使用して、i を 16 進数として出力する。i を適用して、SegCode []配列内の対応するセグメントコードを見つけ、hc595_shift () を使用して SegCode を 74HC595 のシフトレジスタに渡す。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 1.1.4_7-Segment.py
```

コードの実行後、7 セグメントディスプレイに 0~9、AF が表示される。

コード

```
import RPi.GPIO as GPIO
import time

# Set up pins
SDI    = 17
RCLK   = 18
```

SRCLK = 27

```
# Define a segment code from 0 to F in Hexadecimal
segCode = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)

# Shift the data to 74HC595
def hc595_shift(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)

def main():
    while True:
        # Shift the code one by one from segCode list
        for code in segCode:
            hc595_shift(code)
            print ("segCode[%s]: 0x%02X"%(segCode.index(code), code)) # %02X means
double digit HEX to print
            time.sleep(0.5)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
segCode = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]
```

16進数（カソードコモン）の0からFまでのセグメントコード配列。

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
```

ds、st_cp、sh_cp の3つのピンを出力に設定し、初期状態を低レベルとして設定する。

```
    GPIO.output(SDI, 0x80 & (dat << bit))
```

ビットごとに dat データを SDI (DS) に割り当てる。ここでは、dat = 0x3f (0011 1111、bit = 2 を仮定し、0x3f は右 (<<) 2 ビットに切り替える。1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000、は真である。

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

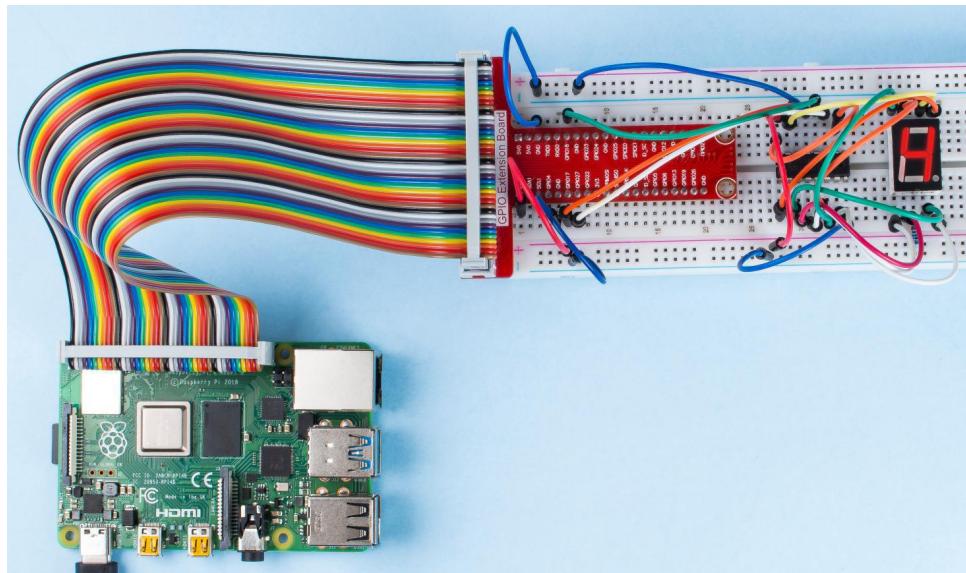
SRCLK の初期値は元々 LOW に設定されていたが、ここでは立ち上がりエッジを生成し、DS データをシフトレジスタに切り替えるために HIGH に設定されている。

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

SRCLK の初期値は元々 LOW に設定されていたが、ここでは立ち上がりエッジを生成し、DS データをシフトレジスタに切り替えるために HIGH に設定されている。

ご注意： 番号 0~15 の 16 進形式は：(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)。

現象画像

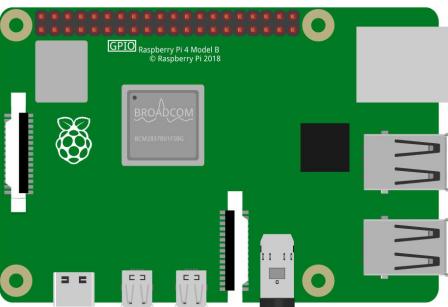
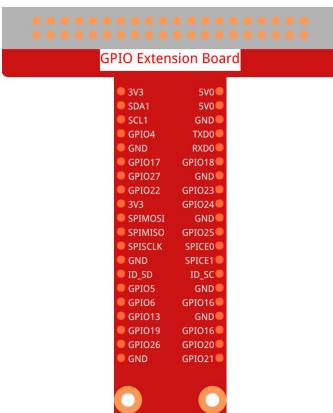
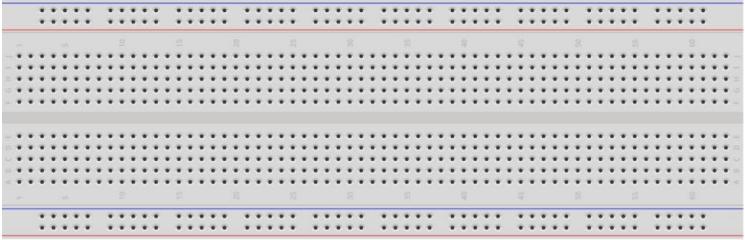


1.1.5 4-Digit 7-Segment Display

前書き

次に、4桁7セグメントディスプレイを制御してみてください。

部品

Raspberry Pi 本体*1	T拡張ボード*1	4桁7セグメントディスプレイ *1 
	 GPIO Extension Board Pinout: 3V3, SDA1, SDO, GND, GPIO4, GND, TXD0, GPIO17, GPIO18, GND, GPIO27, GPIO23, 3V3, GPIO24, SPI MOSI, GND, SPI MISO, GPIO25, SPI SCLK, SPI CE0, GND, ID_SD, ID_SC, GPIO5, GND, GPIO6, GPIO16, GPIO13, GND, GPIO19, GPIO16, GPIO26, GPIO20, GND, GPIO21	
40ピンケーブル*1		74HC595*1 
		抵抗器 (220Ω) * 4 
ブレッドボード*1		何本のジャンパー線 
		

原理

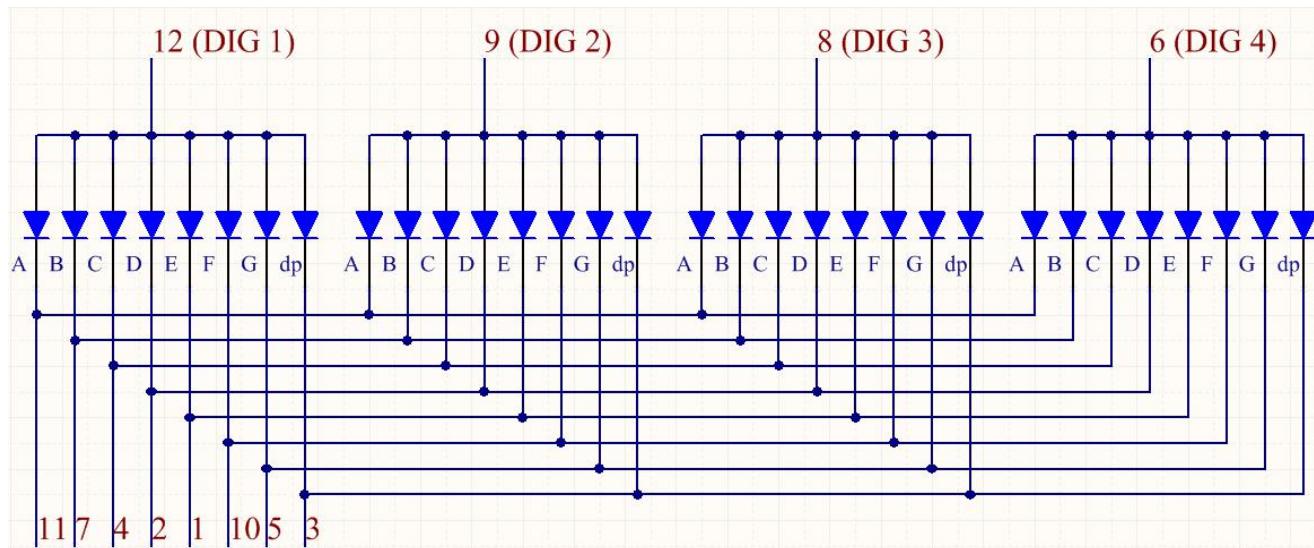
4桁7セグメントディスプレイ

4桁の7セグメントディスプレイは、連携して動作する4つの7セグメントディスプレイで構成されている。



4デジタル7セグメントディスプレイは独立して動作する。人間の視覚的持続性の原理を使用して、ループ内の各7セグメントの文字をすばやく表示し、連続した文字列を形成する。

つまり、ディスプレイに「1234」が表示されている場合、最初の7セグメントに「1」が表示され、「234」は表示されないということである。しばらくすると、2番目の7セグメントに「2」が表示され、7セグメントの1番目、3番目、4番目に表示されなくなり、4つのデジタルディスプレイショーが順番に表示される。このプロセスは非常に短く（通常5ms）、光学的残光効果と視覚的残留の原理により、同時に4つの文字を見ることができる。



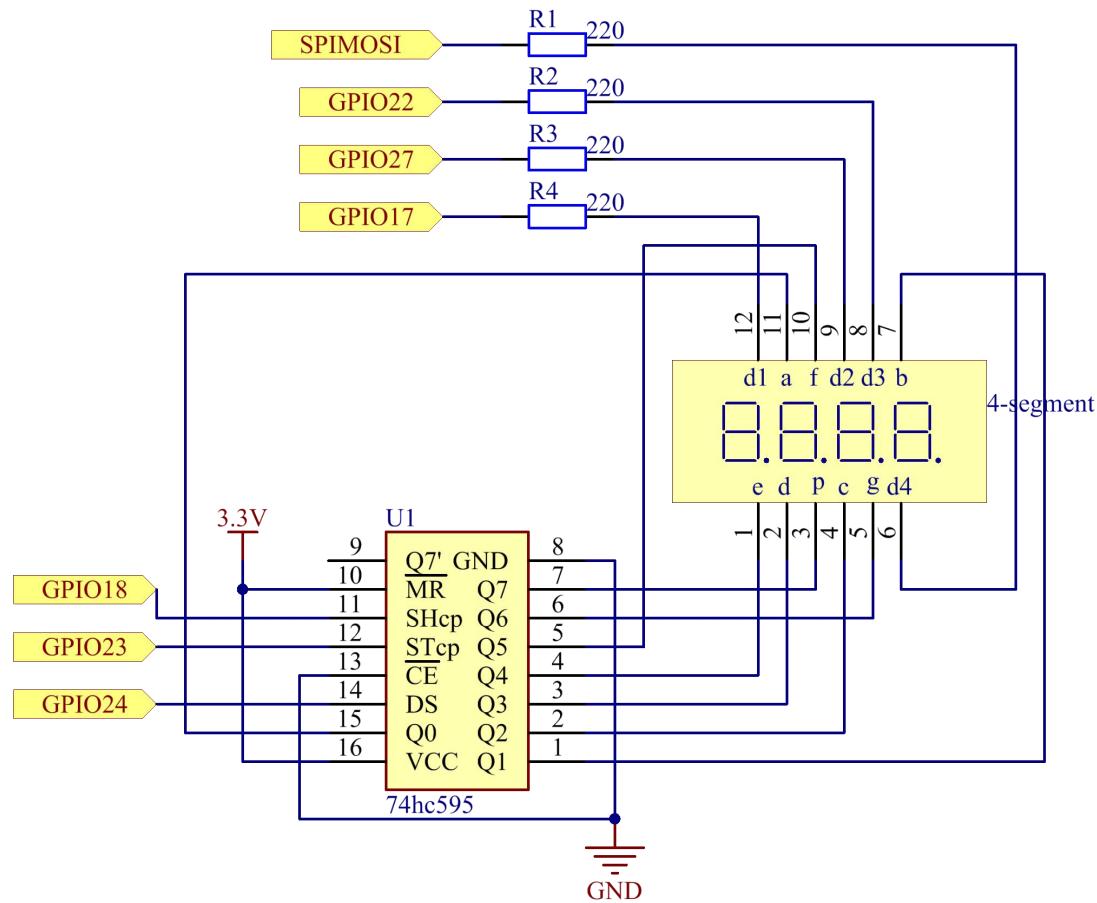
表示コード

7セグメントディスプレイ（アノードコモン）がどのように番号を表示するかを知るために、次の表をご参照ください。数字は7セグメントディスプレイに表示される0～Fの数字である。（DP）GFEDCBAは、0または1に設定された対応のLEDを指す。たとえば、11000000はDPおよびGが1に設定され、他のLEDが0に設定されることを意味する。したがって、7セグメントディスプレイには0が表示され、HEXコードは16進数に対応する。

Numbers	Common Anode		Numbers	Common Anode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCB A	Hex Code
0	11000000	0xc0	A	10001000	0x88
1	11111001	0xf9	B	10000011	0x83
2	10100100	0xa4	C	11000110	0xc6
3	10110000	0xb0	D	10100001	0xa1
4	10011001	0x99	E	10000110	0x86
5	10010010	0x92	F	10001110	0x8e
6	10000010	0x82	.	01111111	0x7f
7	11111000	0xf8			
8	10000000	0x80			
9	10010000	0x90			

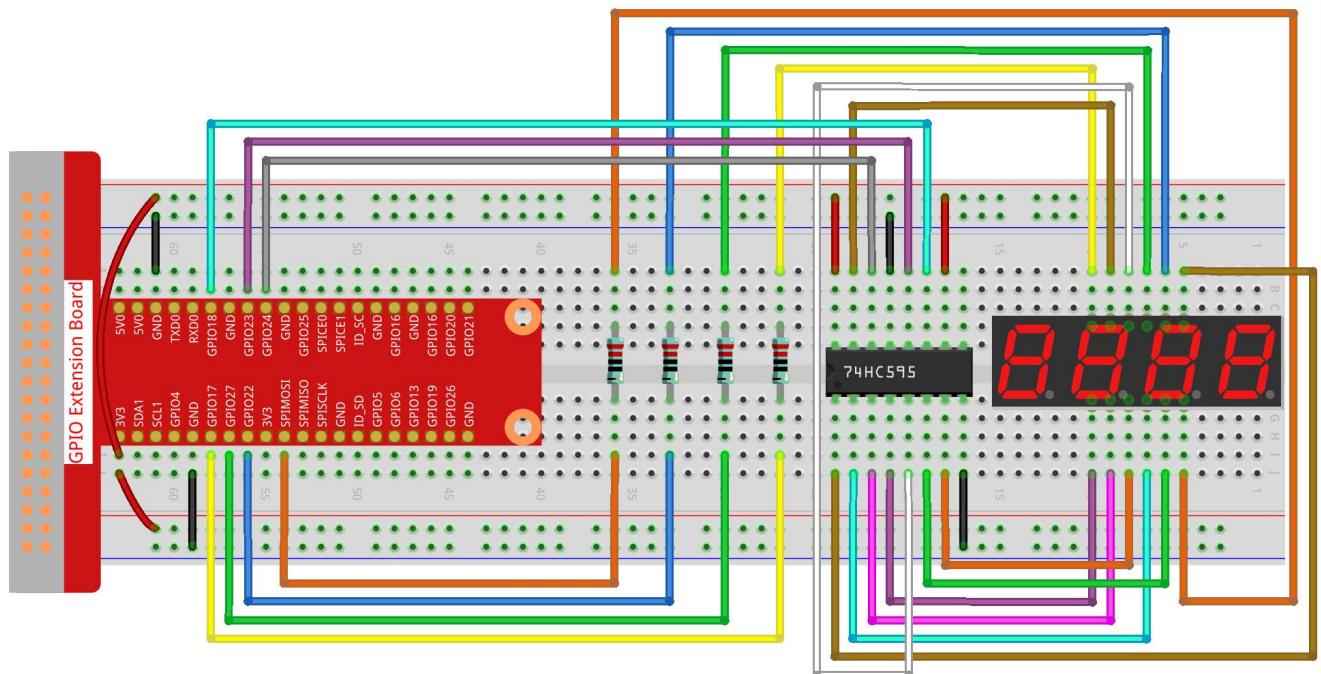
回路図

Tボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPI MOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



実験手順

ステップ 1: 回路を作る。



➤ C言語ユーザー向け

ステップ2：コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.5/
```

ステップ3：コードをコンパイルする。

```
gcc 1.1.5_4-Digit.c -lwiringPi
```

ステップ4：EXEファイルを実行する。

```
sudo ./a.out
```

コードの実行後、プログラムは1秒間に1つずつ増やすカウントを行い、4桁の7セグメントディスプレイにそのカウントが表示される。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>

#define SDI 5
#define RCLK 4
#define SRCLK 1

const int placePin[] = {12, 3, 2, 0};
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

int counter = 0;

void pickDigit(int digit)
{
    for (int i = 0; i < 4; i++)
    {
        digitalWrite(placePin[i], 0);
    }
    digitalWrite(placePin[digit], 1);
}

void hc595_shift(int8_t data)
{
    int i;
```

```

for (i = 0; i < 8; i++)
{
    digitalWrite(SDI, 0x80 & (data << i));
    digitalWrite(SRCLK, 1);
    delayMicroseconds(1);
    digitalWrite(SRCLK, 0);
}
digitalWrite(RCLK, 1);
delayMicroseconds(1);
digitalWrite(RCLK, 0);
}

void clearDisplay()
{
int i;
for (i = 0; i < 8; i++)
{
    digitalWrite(SDI, 1);
    digitalWrite(SRCLK, 1);
    delayMicroseconds(1);
    digitalWrite(SRCLK, 0);
}
digitalWrite(RCLK, 1);
delayMicroseconds(1);
digitalWrite(RCLK, 0);
}

void loop()
{
    while(1){
        clearDisplay();
        pickDigit(0);
        hc595_shift(number[counter % 10]);

        clearDisplay();
        pickDigit(1);
        hc595_shift(number[counter % 100 / 10]);

        clearDisplay();
        pickDigit(2);
        hc595_shift(number[counter % 1000 / 100]);
}

```

```
clearDisplay();
pickDigit(3);
hc595_shift(number[counter % 10000 / 1000]);
}

}

void timer(int timer1)
{
    if (timer1 == SIGALRM)
    {
        counter++;
        alarm(1);
        printf("%d\n", counter);
    }
}

void main(void)
{
    if (wiringPiSetup() == -1)
    {
        printf("setup wiringPi failed !");
        return;
    }
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);

    for (int i = 0; i < 4; i++)
    {
        pinMode(placePin[i], OUTPUT);
        digitalWrite(placePin[i], HIGH);
    }
    signal(SIGALRM, timer);
    alarm(1);
    loop();
}
```

コードの説明

```
const int placePin[] = {12, 3, 2, 0};
```

これらの4つのピンは、4桁の7セグメントディスプレイのアノードコモンピンを制御する。

```
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
```

16進数（アノードコモン）の0～9のセグメントコード配列。

```
void pickDigit(int digit)
{
    for (int i = 0; i < 4; i++)
    {
        digitalWrite(placePin[i], 0);
    }
    digitalWrite(placePin[digit], 1);
}
```

値の場所を選択する。毎回有効な場所は1つだけである。有効な場所はhighに書き込まれる。

```
void loop()
{
    while(1){
        clearDisplay();
        pickDigit(0);
        hc595_shift(number[counter % 10]);
        clearDisplay();
        pickDigit(1);
        hc595_shift(number[counter % 100 / 10]);
        clearDisplay();
        pickDigit(2);
        hc595_shift(number[counter % 1000 / 100]);
        clearDisplay();
        pickDigit(3);
        hc595_shift(number[counter % 10000 / 1000]);
    }
}
```

この機能を使用して、4桁の7セグメントディスプレイに表示される番号を設定する。

clearDisplay(): 11111111に書き込み、7セグメントディスプレイ上のこれら8つのLEDをオフにして、表示されたコンテンツを消去する。

pickDigit(0): 4番目の7セグメントディスプレイを選択する。

hc595_shift(number[counter%10]): カウンターの1桁の数字が4番目のセグメントに表示される。

```
signal(SIGALRM, timer);
```

これもシステム組み込み関数である。コードのプロトタイプは:

```
sig_t signal(int signum, sig_t handler);
```

signal()を実行した後、プロセスは対応する signum (ここで SIGALRM) を受信すると、すぐに既存のタスクを一時停止し、設定関数 (ここで timer (sig))を処理する。

```
alarm(1);
```

これもシステム組み込み関数である。コードのプロトタイプは

```
unsigned int alarm (unsigned int seconds);
```

指定した秒数後に SIGALRM シグナルを生成する。

```
void timer(int timer1)
{
    if (timer1 == SIGALRM)
    {
        counter++;
        alarm(1);
        printf("%d\n", counter);
    }
}
```

上記の関数を使用して、タイマー関数を実装する。

alarm () が SIGALRM シグナルを生成した後、タイマー関数が呼び出される。カウンターに 1 を追加すると、1秒後に関数 alarm (1) が繰り返し呼び出される。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 1.1.5_4-Digit.py
```

コードの実行後、プログラムは1秒ずつ増加するカウントをして、4桁のディスプレイにカウントが表示される。

コード

```
import RPi.GPIO as GPIO
import time
import threading

SDI = 24
RCLK = 23
SRCLK = 18

placePin = (10, 22, 27, 17)
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

counter = 0
timer1 = 0

def clearDisplay():
    for i in range(8):
        GPIO.output(SDI, 1)
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)

def hc595_shift(data):
    for i in range(8):
        GPIO.output(SDI, 0x80 & (data << i))
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)
```

```

def pickDigit(digit):
    for i in placePin:
        GPIO.output(i,GPIO.LOW)
    GPIO.output(placePin[digit], GPIO.HIGH)

def timer():
    global counter
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter += 1
    print("%d" % counter)

def loop():
    global counter
    while True:
        clearDisplay()
        pickDigit(0)
        hc595_shift(number[counter % 10])

        clearDisplay()
        pickDigit(1)
        hc595_shift(number[counter % 100//10])

        clearDisplay()
        pickDigit(2)
        hc595_shift(number[counter % 1000//100])

        clearDisplay()
        pickDigit(3)
        hc595_shift(number[counter % 10000//1000])

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    for i in placePin:
        GPIO.setup(i, GPIO.OUT)
    global timer1
    timer1 = threading.Timer(1.0, timer)

```

```

timer1.start()

def destroy():  # When "Ctrl+C" is pressed, the function is executed.
    global timer1
    GPIO.cleanup()
    timer1.cancel() # cancel the timer

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

コードの説明

placePin = (10, 22, 27, 17)

これらの 4 つのピンは、4 桁の 7 セグメントディスプレイのアノードコモンピンを制御する。

number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

16 進数の 0~9 のセグメントコード配列（アノードコモン）。

```

def clearDisplay():
    for i in range(8):
        GPIO.output(SDI, 1)
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)

```

SDI に「1」を 8 回書き込むと、7 セグメントディスプレイの 8 つの LED が消灯し、表示されたコンテンツが消去される。

```

def pickDigit(digit):
    for i in placePin:
        GPIO.output(i,GPIO.LOW)
        GPIO.output(placePin[digit], GPIO.HIGH)

```

値の場所を選択する。毎回有効な場所は 1 つだけである。有効な場所は high に書き込まれる。

```
def loop():
    global counter
    while True:
        clearDisplay()
        pickDigit(0)
        hc595_shift(number[counter % 10])

        clearDisplay()
        pickDigit(1)
        hc595_shift(number[counter % 100//10])

        clearDisplay()
        pickDigit(2)
        hc595_shift(number[counter % 1000//100])

        clearDisplay()
        pickDigit(3)
        hc595_shift(number[counter % 10000//1000])
```

この機能は、4 行の 7 セグメントディスプレイに表示される番号を設定するために使用される。

まず、4 番目のセグメントディスプレイを開始し、1 行の数字を書き込む。次に、3 番目のセグメントディスプレイを開始し、10 行の数字を入力する。その後、2 番目と 1 番目のセグメントディスプレイをそれぞれ開始し、それぞれ数百行と数千行を書き込む。リフレッシュ速度が非常に速いため、完全な 4 行のディスプレイが表示される。

```
timer1 = threading.Timer(1.0, timer)
timer1.start()
```

モジュール、スレッドは Python の一般的なスレッドモジュールであり、タイマーはそのサブクラスである。

コードのプロトタイプは次のとおりです：

```
class threading.Timer(interval, function, args=[], kwargs={})
```

間隔の後、関数が実行される。ここでは、間隔は 1.0、関数は timer () である。

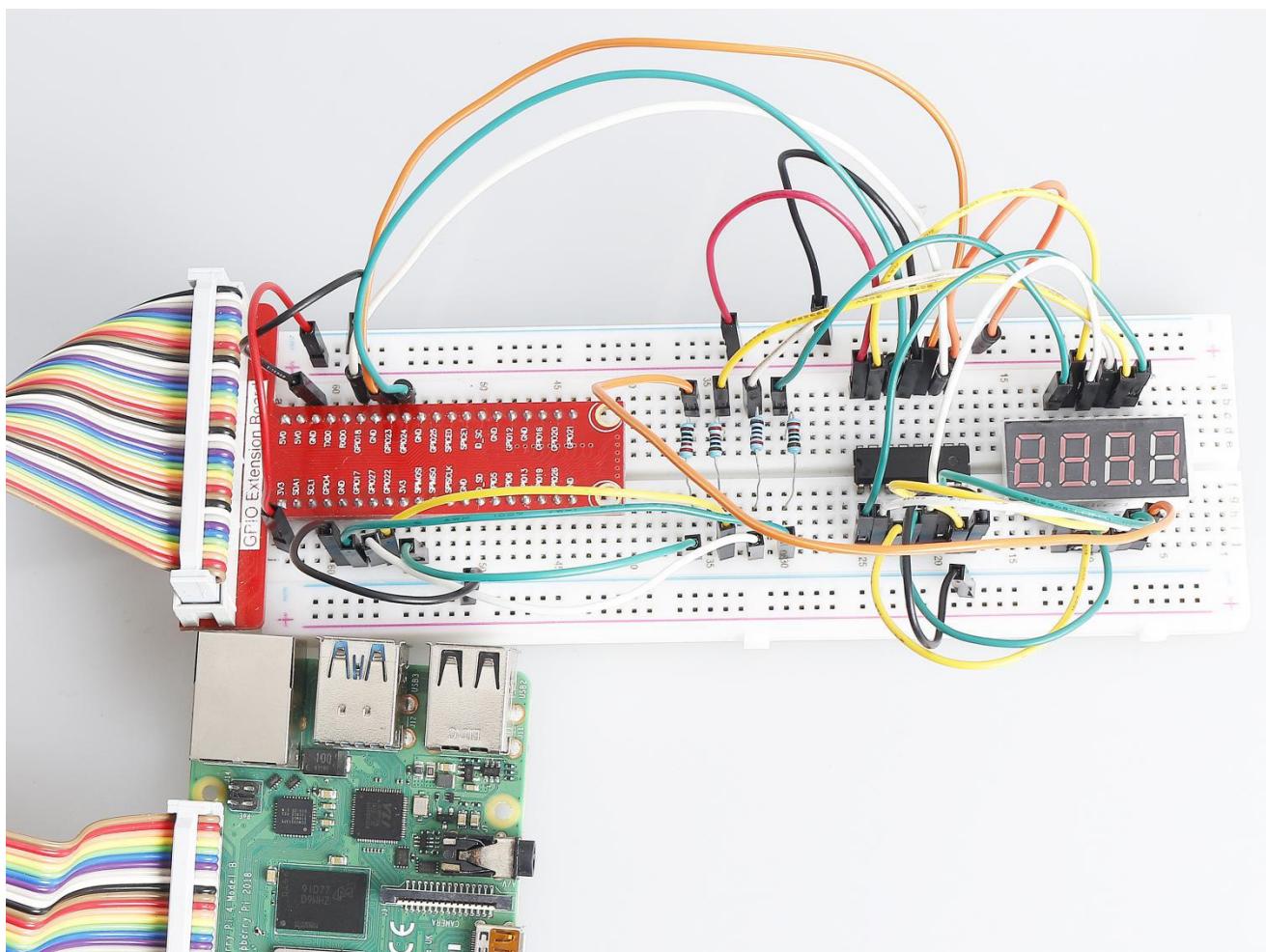
start () は、タイマーがこの時点で開始することを意味する。

```
def timer():
```

```
global counter
global timer1
timer1 = threading.Timer(1.0, timer)
timer1.start()
counter += 1
print("%d" % counter)
```

Timer が 1.0 秒に達すると、Timer 関数が呼び出される。カウンターに 1 を追加すると、タイマーが再び使用されて、1 秒ごとに繰り返し実行される。

現象画像

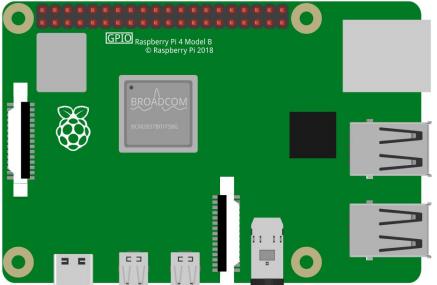
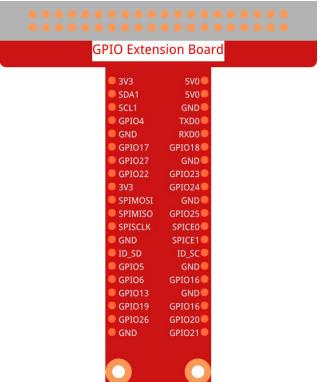
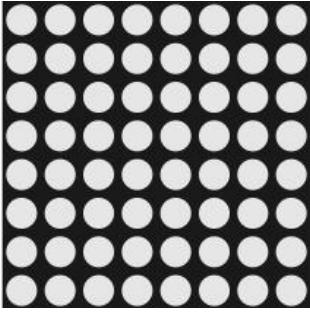
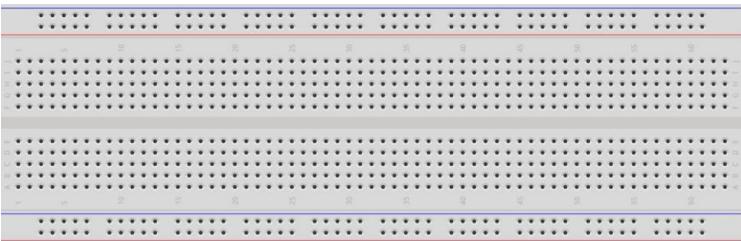


1.1.6 LED Dot Matrix

前書き

名前が示すように、LED ドットマトリックスは LED で構成されるマトリックスである。LED の点灯と滅光は、さまざまな文字とパターンを形成する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	LED ドットマトリックス * 1																																								
	 <p>GPIO Extension Board</p> <table border="1"> <tbody> <tr><td>3V3</td><td>5V0</td></tr> <tr><td>SDA1</td><td>5V0</td></tr> <tr><td>SCL1</td><td>GND</td></tr> <tr><td>GPIO4</td><td>TXD0</td></tr> <tr><td>GND</td><td>RXD0</td></tr> <tr><td>GPIO17</td><td>GPIO18</td></tr> <tr><td>GPIO27</td><td>GND</td></tr> <tr><td>GPIO22</td><td>GPIO23</td></tr> <tr><td>3V3</td><td>GPIO24</td></tr> <tr><td>SPIMISO</td><td>GND</td></tr> <tr><td>SPIMOSI</td><td>GPIO25</td></tr> <tr><td>SPISCL</td><td>SPICE0</td></tr> <tr><td>GND</td><td>SPICE1</td></tr> <tr><td>ID_SD</td><td>ID_SC</td></tr> <tr><td>GPIO9</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td></tr> <tr><td>GPIO13</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO15</td></tr> <tr><td>GPIO26</td><td>GPIO10</td></tr> <tr><td>GND</td><td>GPIO20</td></tr> </tbody> </table>	3V3	5V0	SDA1	5V0	SCL1	GND	GPIO4	TXD0	GND	RXD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPIMISO	GND	SPIMOSI	GPIO25	SPISCL	SPICE0	GND	SPICE1	ID_SD	ID_SC	GPIO9	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO15	GPIO26	GPIO10	GND	GPIO20	
3V3	5V0																																									
SDA1	5V0																																									
SCL1	GND																																									
GPIO4	TXD0																																									
GND	RXD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPIMISO	GND																																									
SPIMOSI	GPIO25																																									
SPISCL	SPICE0																																									
GND	SPICE1																																									
ID_SD	ID_SC																																									
GPIO9	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO15																																									
GPIO26	GPIO10																																									
GND	GPIO20																																									
40 ピンケーブル*1		何本のジャンパー線																																								
																																										
ブレッドボード*1		74HC595* 2																																								
																																										

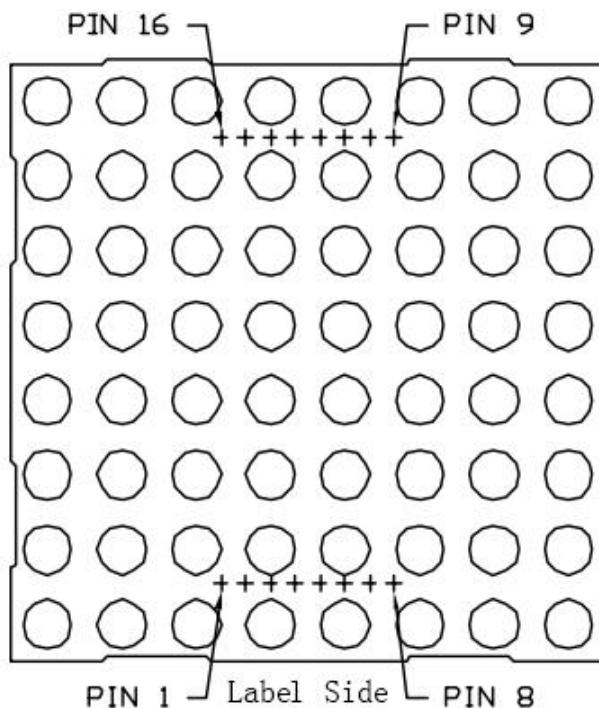
原理

LED ドットマトリックス

一般に、LED ドットマトリックスは、カソードコモン (CC) とアノードコモン (CA) の 2 つのタイプに分類できる。見た目は似ているが、内部的には違いがある。テストを行うとすぐに分かる。このキットでは CA が使用される。側面に 788BS というラベルが付いている。

下の図を参照してください。ピンは背面の両端に配置されている。ラベル側を例にする：この端のピンはピン 1~8、もう一端はピン 9~16 である。

外部ビュー：

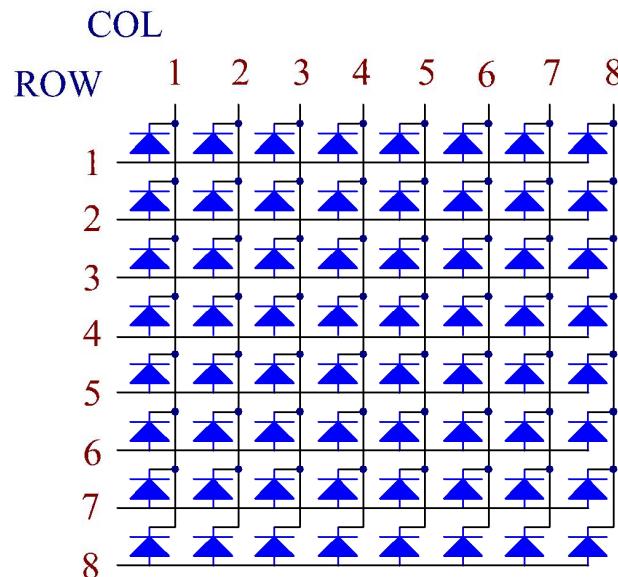


図の下に内部構造を示している。CA LED ドットマトリックスが見える。ROW は LED のアノードを表し、COL はカソードを表す。CC の場合は逆である。共通点が 1 つある：両方のタイプで、ピン 13、3、4、10、6、11、15、および 16 はすべて COL である。

ピン 9、14、8、12、1、7、2、および 5 がすべて ROW である。左上隅の最初の LED をオンにする場合、CA LED ドットマトリックスに対して、ピン 9 を High、ピン 13 を Low に設定し、CC 1 に対して、ピン 13 を High に、ピン 9 を Low に設定する。CA の最初の列全体を点灯させる場合は、ピン 13 を Low に、行 9、14、8、12、1、7、2、および 5 を High に設定する

CC の場合、ピン 13 を High に、行 9、14、8、12、1、7、2、および 5 を Low に設定する。理解を深めるために、次の図を検討してください。

内部ビュー:



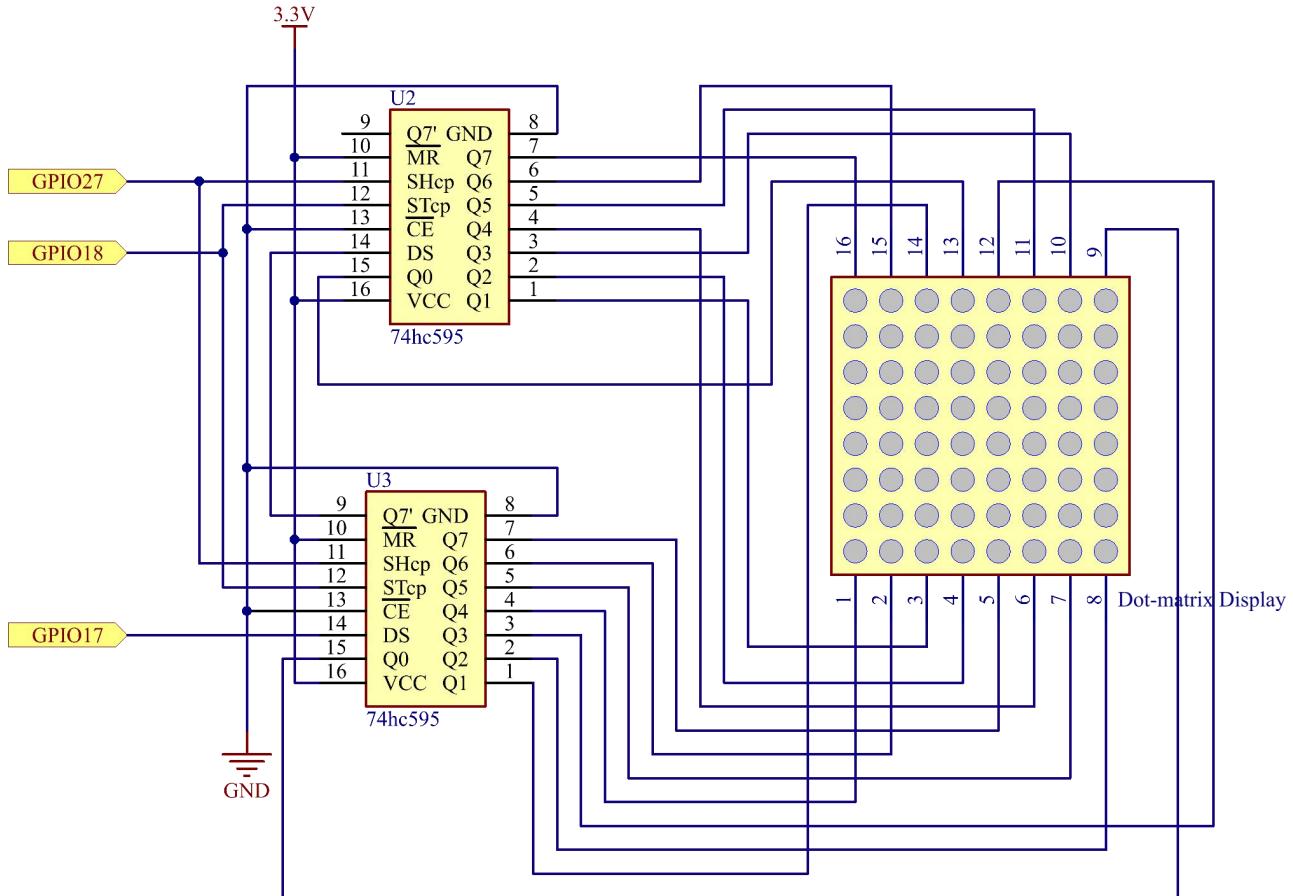
上記の行と列に対応するピン番号:

COL	1	2	3	4	5	6	7	8
ピン番号	13	3	4	10	6	11	15	16
ROW	1	2	3	4	5	6	7	8
ピン番号	9	14	8	12	1	7	2	5

さらに、2つの74HC595チップがここで使用されている。1つはLEDドットマトリックスの行を制御し、もう1つは列を制御する。

回路図

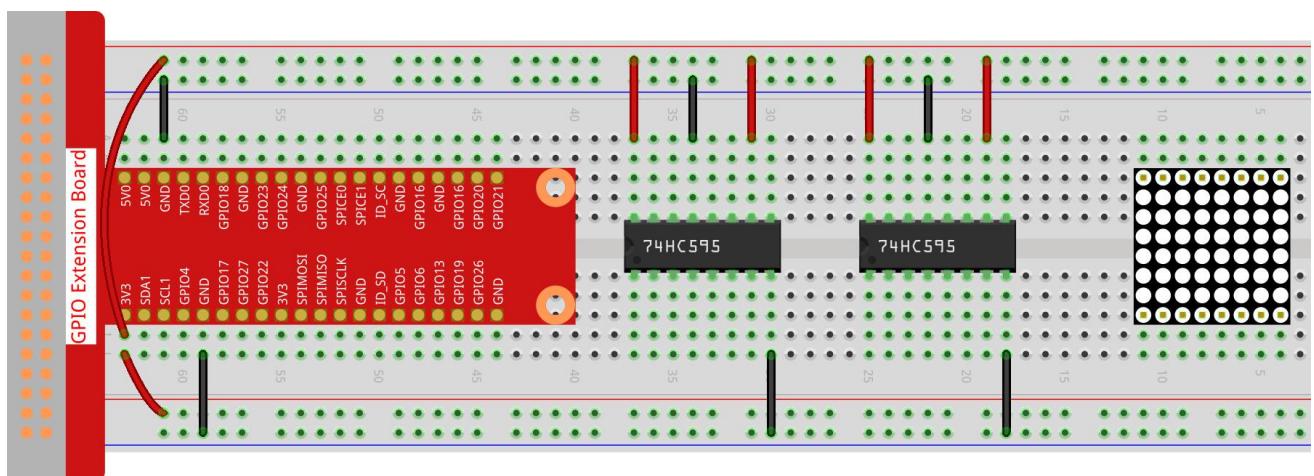
Tボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



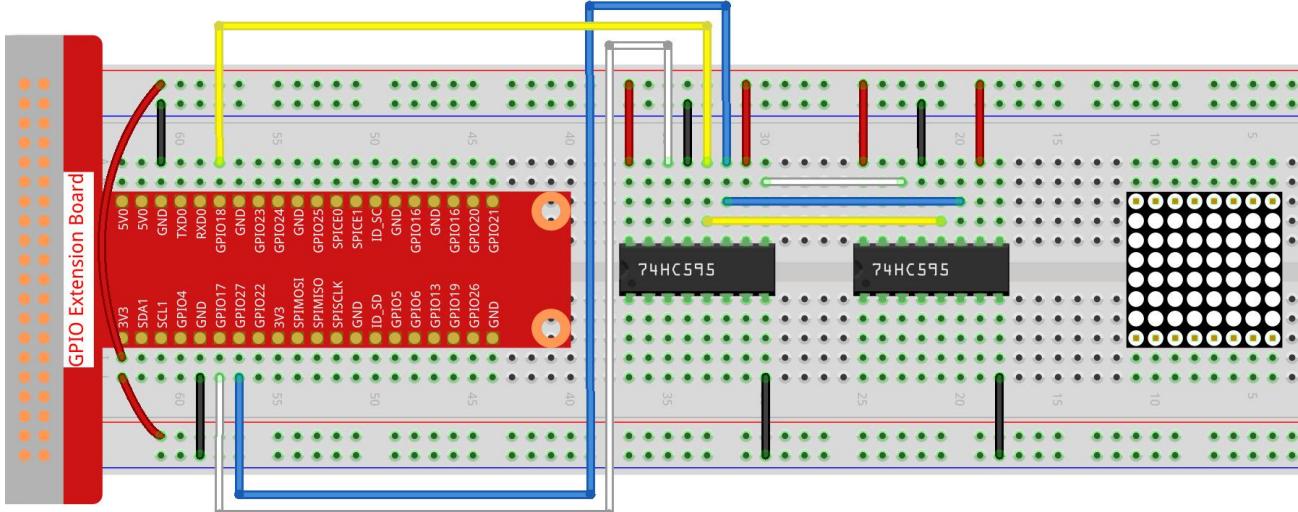
実験手順

ステップ1: 回路を作る。配線が複雑なので、段階的に作りましょう。まず、T-Cobbler、LED ドットマトリックス、および 2 つの 74HC595 チップをブレッドボードに挿入する。T-Cobbler の 3.3V と接地をボードの両側の穴に接続し、2 つの 74HC595 チップのピン 16 と 10 を VCC に、ピン 13 とピン 8 を接地に接続する。

ご注意: 上の Fritzing 画像では、ラベルのある側が下にある。

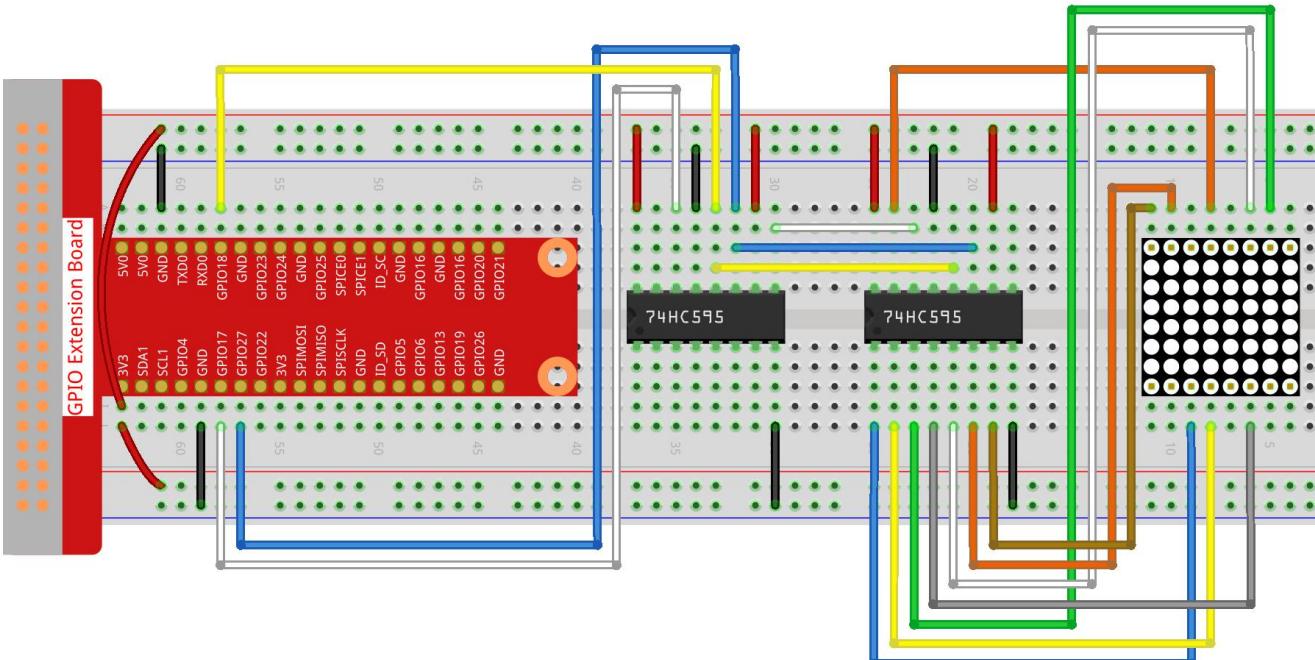


ステップ2: 2 つの 74HC595 のピン 11 を一緒に接続し、GPIO27 に接続する。次に、2 つのチップのピン 12 を一緒に接続し、GPIO18 に接続する。それから左側の 74HC595 のピン 14 を GPIO17 に、ピン 9 を 2 番目の 74HC595 のピン 14 に接続する。



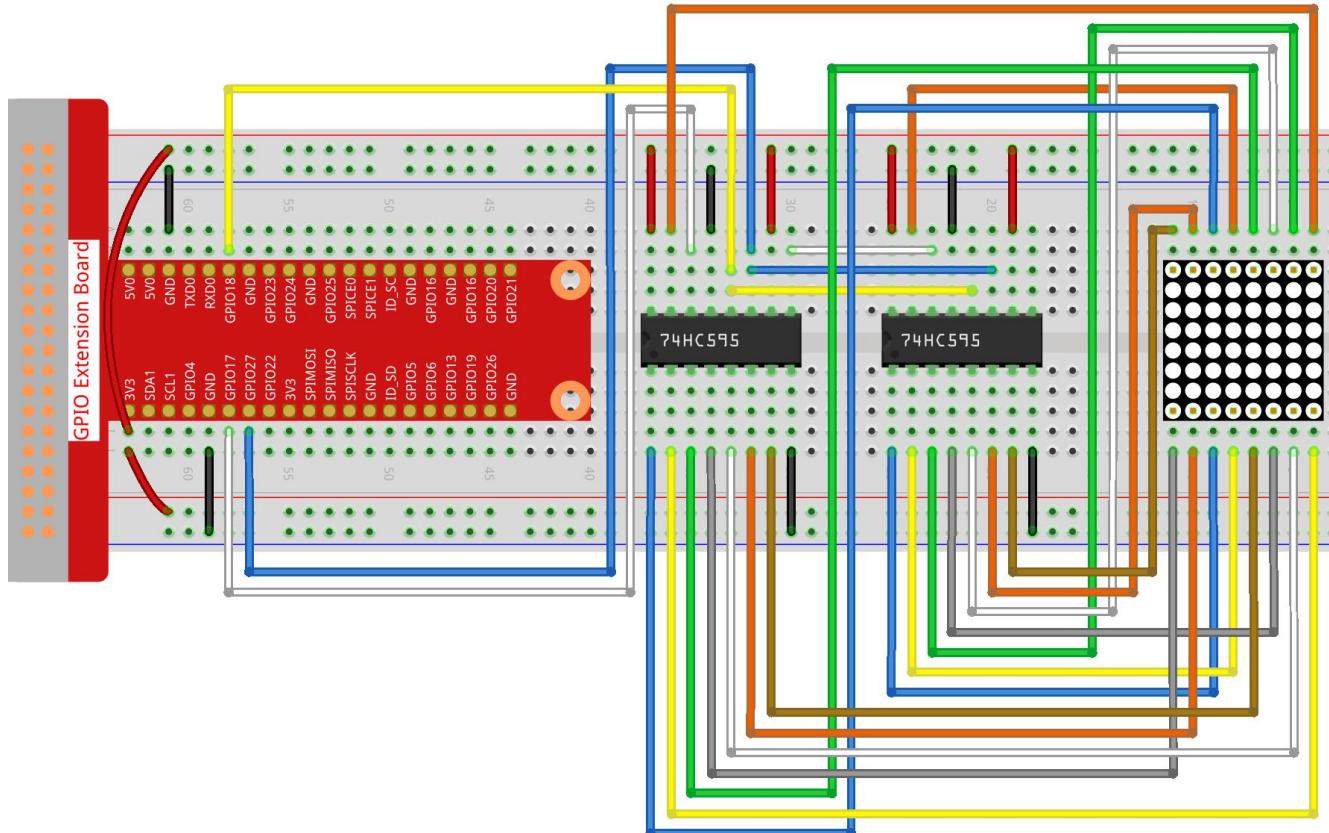
ステップ3: 右側の74HC595は、LEDドットマトリックスの列を制御する。マッピングについては、以下の表を参照してください。したがって、74HC595のQ0～Q7ピンは、それぞれピン13、3、4、10、6、11、15、および16にマップされる。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



ステップ4: 次に、LED ドットマトリックスの行を接続する。左側の 74HC595 は LED ドットマトリックスの行を制御する。マッピングについては、以下の表を参照してください。左側の 74HC595 の Q0～Q7 は、それぞれピン 9、14、8、12、1、7、2、および 5 にマッピングされていることは分かった。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



➤ C言語ユーザー向け

ステップ5: コードのフォルダーに移動する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.6/
```

ステップ6: コンパイルする。

```
gcc 1.1.6_LedMatrix.c -lwiringPi
```

ステップ7: 実行する。

```
sudo ./a.out
```

コードの実行後、LED ドットマトリックスが行ごとに、列ごとに点灯したり消灯したりする。

コード

```
#include <wiringPi.h>
#include <stdio.h>

#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)

unsigned char code_H[20] =
{0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0x
ff};

unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
fe,0xfd,0xfb,0xf7,0xef,0xdf,0xb,0x7f};

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);

    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

void hc595_in(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
    }
}

void hc595_out(){
    digitalWrite(RCLK, 1);
    delay(1);
    digitalWrite(RCLK, 0);
}
```

```

int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<sizeof(code_H);i++){
            hc595_in(code_L[i]);
            hc595_in(code_H[i]);
            hc595_out();
            delay(100);
        }

        for(i[sizeof(code_H)];i>=0;i--){
            hc595_in(code_L[i]);
            hc595_in(code_H[i]);
            hc595_out();
            delay(100);
        }
    }

    return 0;
}

```

コードの説明

```

unsigned char code_H[20] =
{0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0x
ff};

unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
bf,0x7f};

```

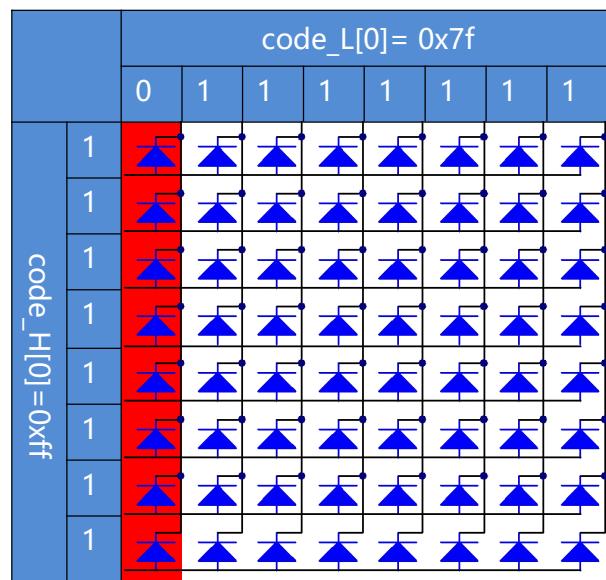
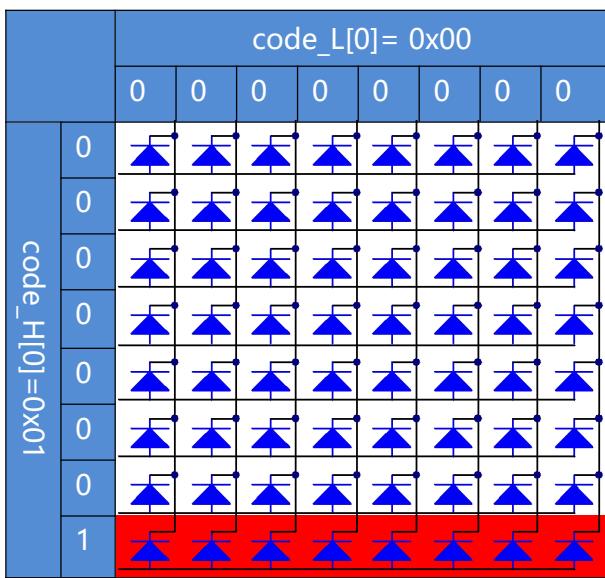
配列 code_H は LED ドットマトリックス行の要素を表し、配列 code_L は列の要素を表す。文字が表示されると、行の要素と列の要素が取得され、それぞれ 2 つの HC595 チップに割り当てられる。したがって、パターンは LED ドットマトリックスに表示される。

例として、code_H の最初の数字 0x01 と code_L の最初の数字 0x00 を取り上げる。

二進法に変換された 0x01 は 00000001 になり、二進法に変換された 0x00 は 0000 0000 になる。

このキットでは、アノードコモン LED ドットマトリックスディスプレイが適用されるため、8列目の8つのLEDのみが点灯する。

コードHが0xffで、code_Lが0x7fであるという条件が同時に満たされると、最初の列のこれら8つのLEDが点灯する。



```
void hc595_in(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
    }
}
```

datの値をHC595のSDIピンにビット単位で書き込む。SRCLKの初期値は元々に0に設定されていたが、ここでは1に設定されている。これは、立ち上がりエッジパルスを生成し、ピンSDI(DS)の日付をシフトレジスタにシフトする。

```
void hc595_out(){
    digitalWrite(RCLK, 1);
    delay(1);
    digitalWrite(RCLK, 0);
}
```

RCLKの初期値は元々に0に設定されていたが、ここでは1に設定されている。これは、立ち上がりエッジパルスを生成し、データーをシフトレジスタからストレージレジスターにシフトする。

```
while(1){
    for(i=0;i<sizeof(code_H);i++){
        hc595_in(code_L[i]);
    }
}
```

```
    hc595_in(code_H[i]);  
    hc595_out();  
    delay(100);  
}
```

このループでは、2つの配列 `code_L` および `code_H` のこれらの 20 個の要素が 2 つの 74HC595 チップに 1 つずつアップロードされる。次に、関数 `hc595_out()` を呼び出して、データをシフトレジスタからストレージレジスタにシフトする。

➤ Python 言語ユーザー向け

ステップ 5: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 6: 実行する。

sudo python3 1.1.6 LedMatrix.py

コードの実行後、LED ドットマトリックスが行ごとに、列ごとに点灯したり消灯したりする。

コード

```
import RPi.GPIO as GPIO
import time

SDI    = 17
RCLK   = 18
SRCLK = 27

# we use BX matrix, ROW for anode, and COL for cathode
# ROW +++
code_H =
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0x
ff]
# COL ---
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
bf,0x7f]

def setup():
    GPIO.setmode(GPIO.BCM)      # Number GPIOs by its BCM location
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
```

```

GPIO.setup(SRCLK, GPIO.OUT)
GPIO.output(SDI, GPIO.LOW)
GPIO.output(RCLK, GPIO.LOW)
GPIO.output(SRCLK, GPIO.LOW)

# Shift the data to 74HC595
def hc595_shift(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(RCLK, GPIO.LOW)

def main():
    while True:
        for i in range(0, len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            time.sleep(0.1)

        for i in range(len(code_H)-1, -1, -1):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            time.sleep(0.1)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()

```

コードの説明

```
code_H =  
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0x  
ff]  
code_L =  
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x  
fe,0xfd,0xfb,0xf7,0xef,0xdf,0xb  
f,0x7f]
```

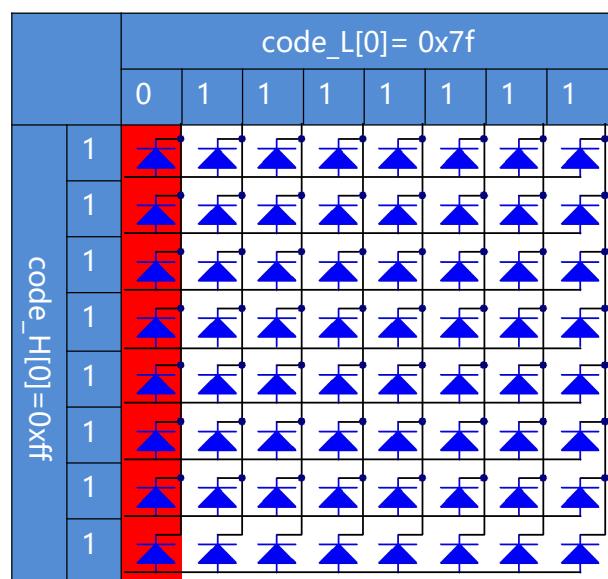
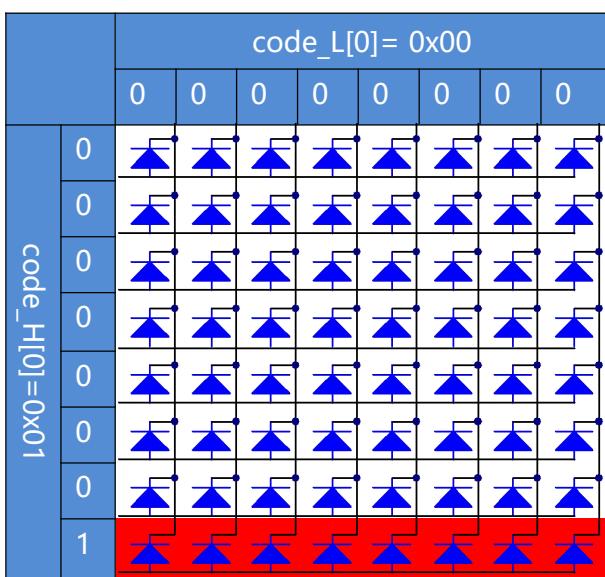
配列 code_H は matix 行の要素を表し、配列 code_L は列の要素を表す。文字が表示されると、行の要素と列の要素が取得され、それぞれ 2 つの HC595 チップに割り当てられる。したがって、パターンは LED ドットマトリックスに表示される。

例として、code H の最初の数字 0x01 と code L の最初の数字 0x00 を取り上げる。

二進法に変換された 0x01 は 00000001 になり、二進法に変換された 0x00 は 0000 0000 になる。

このキットでは、アノードコモン LED ドットマトリックスが適用されるため、8 行目の 8 つの LED のみが点灯する。

コード H が 0xff で、code_L が 0x7f であるという条件が同時に満たされると、最初の列のこれら 8 つの LED が点灯する。

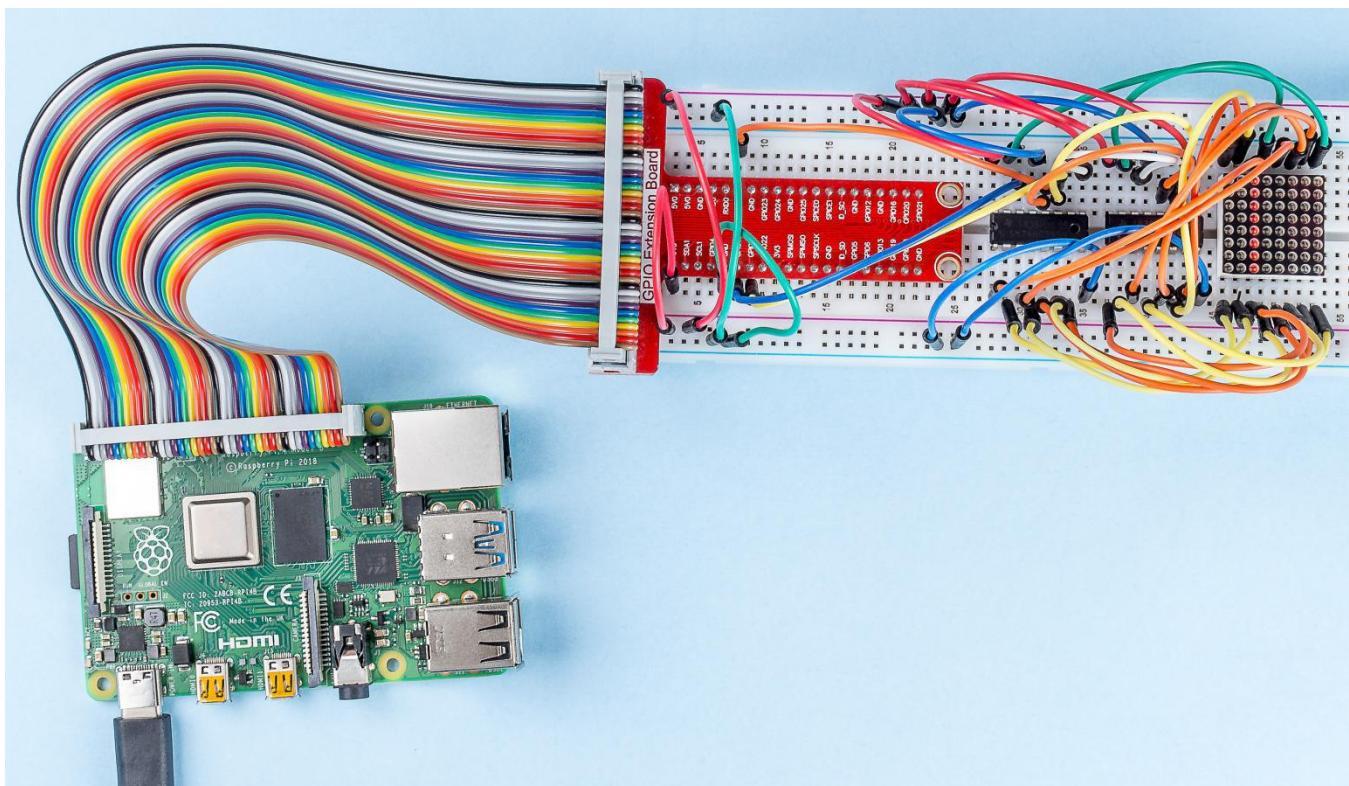


```
for i in range(0, len(code_H)):  
    hc595_shift(code_L[i])  
    hc595_shift(code_H[i])
```

このループでは、2つの配列 code_L および code_H のこれらの 20 個の要素が 2 つの HC595 チップに 1 つずつアップロードされる。

ご注意: LED ドットマトリックスに文字を表示する場合は、Python コードを参照してください: https://github.com/sunfounder/SunFounder_Dot_Matrix

現象画像



1.1.7 I2C LCD1602

前書き

LCD1602 は文字型液晶ディスプレイで、32 (16 * 2) 文字を同時に表示できる。

部品

Raspberry Pi 本体*1

T 拡張ボード*1

I2C LCD1602*1

40 ピンケーブル*1

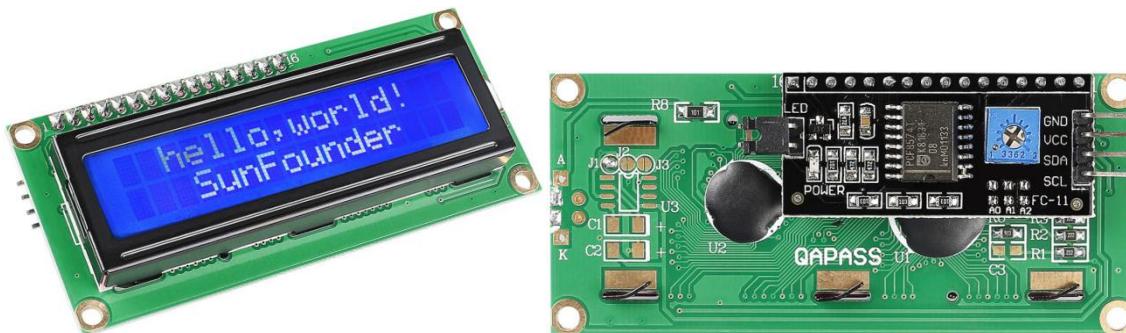
何本のジャンパー線

ブレッドボード*1

原理

I2C LCD1602

ご存知のように、LCD やその他のディスプレイはマンマシンの相互作用を大幅に強化するが、共通の弱点を共有している。それらがコントローラーに接続されると、外部ポートがあまりないコントローラーの複数の IO が占用される。また、コントローラーの他の機能も制限される。したがって、この問題を解決するために、I2C バスを備えた LCD1602 が開発された。



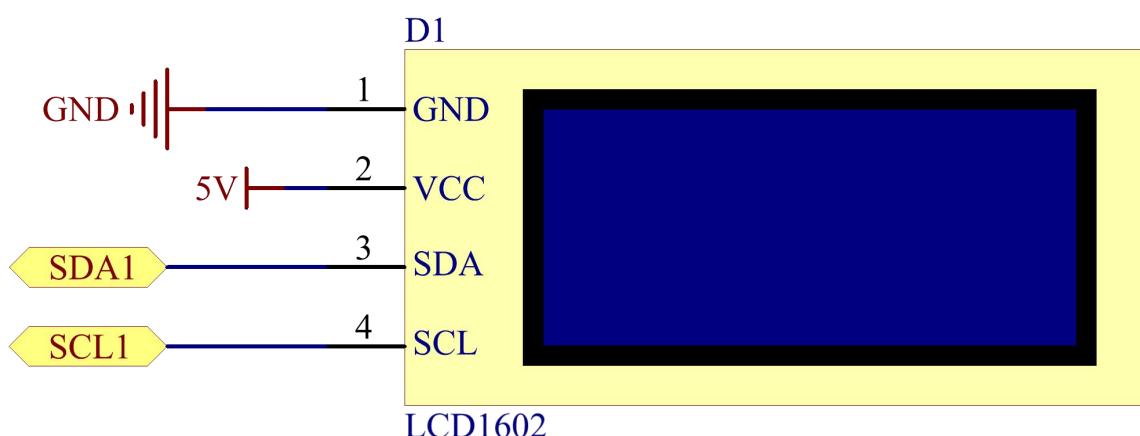
I2C 通信

I2C (アイ・スクエアド・シー) バスは、一つマスターデバイス（または複数のマスターデバイス）と单一または複数のスレーブデバイス間の通信用の非常に強力なバスである。

I2C メインコントローラーを使用して、IO エクスパンダー、各種センサー、EEPROM、ADC/DACなどを制御できる。これらはすべて、ホストの 2 つのピン、シリアルデータ (SDA1) ラインとシリアルクロックライン (SCL1) によってのみ制御される。

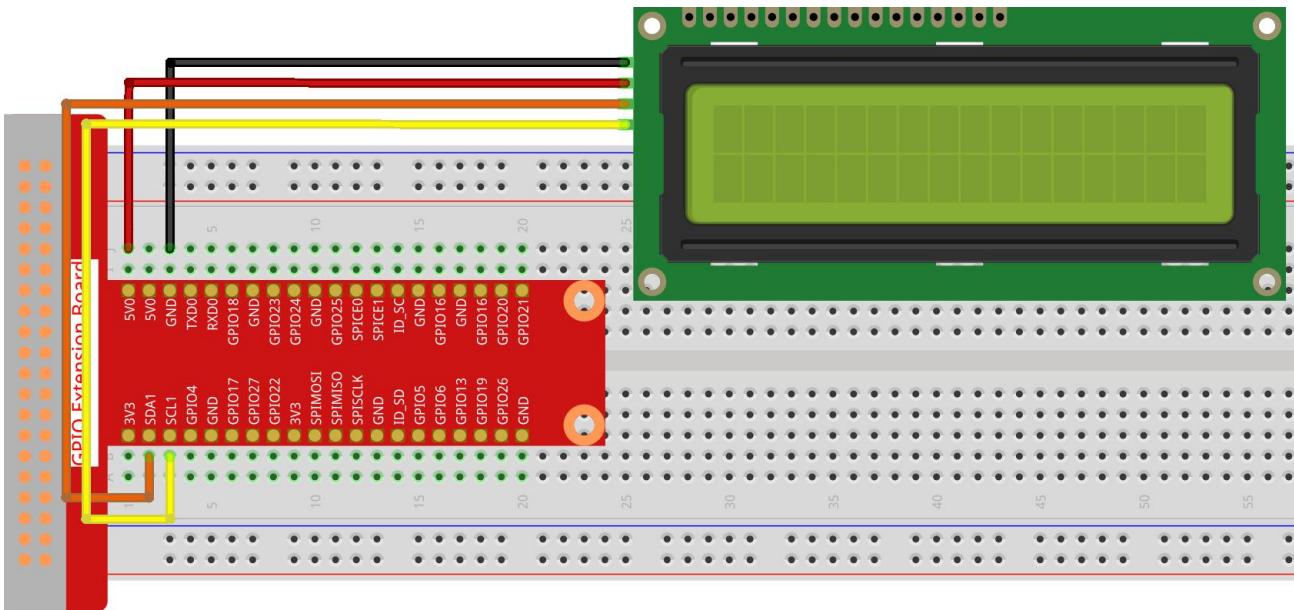
回路図

T ボード名	physical
SDA1	Pin 3
SCL1	Pin 5



実験手順

ステップ 1: 回路を作る。



ステップ 2: I2C 設定 (付録を参照してください。I2C を設定している場合は、この手順をスキップしてください。)

➤ C 言語ユーザー向け

ステップ 3: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.7/
```

ステップ 4: コンパイルする。

```
gcc 1.1.7_Lcd1602.c -lwiringPi
```

ステップ 5: 実行する。

```
sudo ./a.out
```

コードの実行後、LCD に「Greetings」、「SunFounder から」が表示される。

コード

ご注意: 次の省略される関数は完全ではない。Bash インターフェースでコマンド nano 1.1.7_lcd1602.c を使用すると、完全なコードを表示できる。

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPi2C.h>
#include <string.h>

int LCDAddr = 0x27;
```

```

int BLEN = 1;
int fd;

void write_word(int data){.....}
void send_command(int comm){.....}
void send_data(int data){.....}
void init(){.....}
void clear(){.....}
void write(int x, int y, char data[]){.....}

void main(){
    fd = wiringPil2CSetup(LCDAddr);
    init();
    write(0, 0, "Greetings!");
    write(1, 1, "From SunFounder");
}

```

コードの説明

```

void write_word(int data){.....}
void send_command(int comm){.....}
void send_data(int data){.....}
void init(){.....}
void clear(){.....}
void write(int x, int y, char data[]){.....}

```

これらの関数は、I2C LCD1602 オープンソースコードを制御するために使用される。これにより、I2C LCD1602 を簡単に使用できる。

これらの関数の中で、init () は初期化に使用され、clear () は画面の消去に使用され、write () は表示内容の書き込みに使用され、他の関数は上記の関数をサポートする。

```
fd = wiringPil2CSetup(LCDAddr);
```

この関数は指定されたデバイスシンボルで I2C システムを初期化する。関数のプロトタイプ：

```
int wiringPil2CSetup(int devId);
```

パラメーター devId は I2C デバイスのアドレスであり、i2cdetect コマンド（付録を参照）で見つけることができ、I2C LCD1602 の devId は通常 0x27 である。

```
void write(int x, int y, char data[]){}
```

この関数では、data []は LCD にプリントされる文字であり、パラメーター x と y はプリントの位置を決定する（行 y + 1、列 x + 1 はプリントされる文字の開始位置である）。

➤ Python 言語ユーザー向け

ステップ3: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ4: 実行。

```
sudo python3 1.1.7_Lcd1602.py
```

コードの実行後、LCD に「Greetings」、「SunFounder から」が表示される。

コード

```
import LCD1602
import time

def setup():
    LCD1602.init(0x27, 1)      # init(slave address, background light)
    LCD1602.write(0, 0, 'Greetings!!')
    LCD1602.write(1, 1, 'from SunFounder')
    time.sleep(2)

def destroy():
    LCD1602.clear()

if __name__ == "__main__":
    try:
        setup()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
import LCD1602
```

このファイルは I2C LCD1602 を制御するためのオープンソースファイルである。I2C LCD1602 を簡単に使用できる。

```
LCD1602.init(0x27, 1)
```

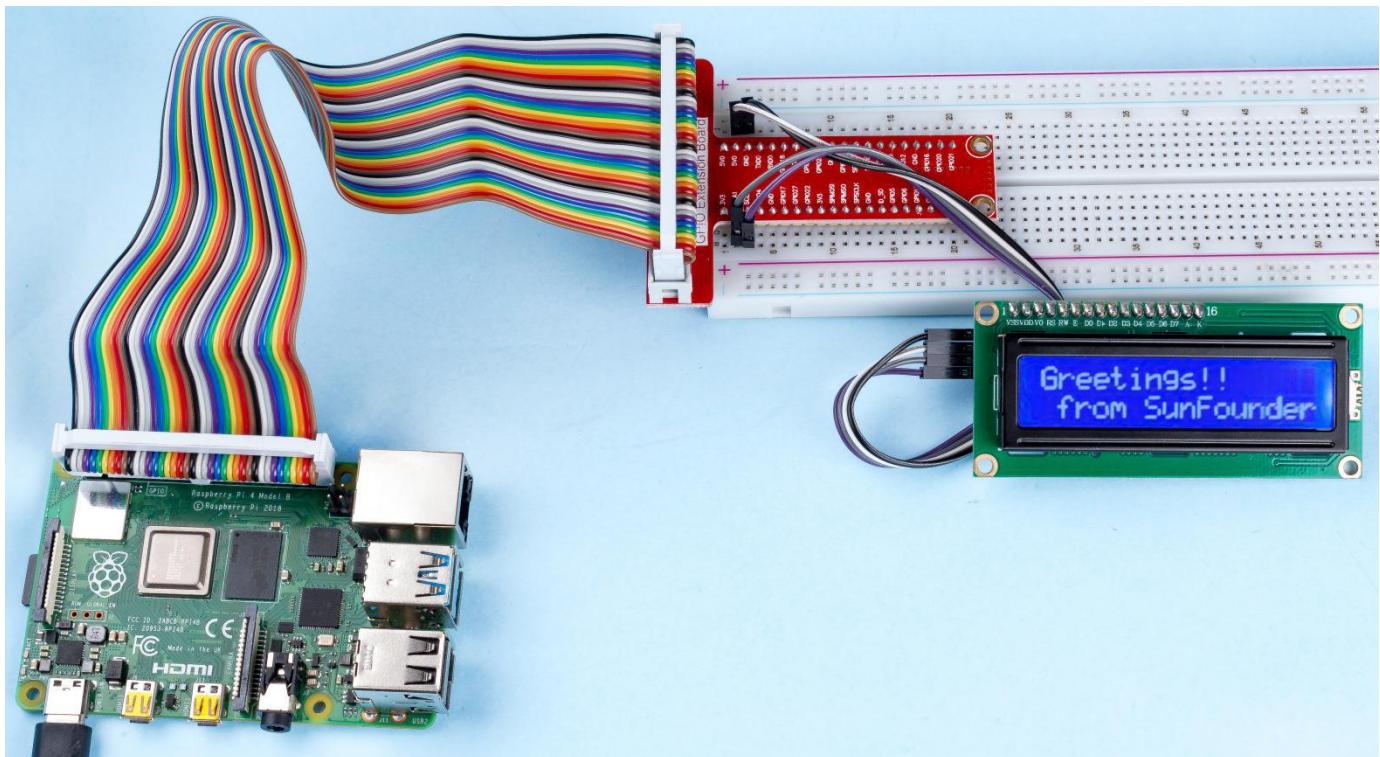
この関数は指定されたデバイスシンボルで I2C システムを初期化する。最初のパラメーターは I2C デバイスのアドレスで、i2cdetect コマンドで検出できる（詳細については付録を参照してください）。I2C LCD1602 のアドレスは通常 0x27 である。

```
LCD1602.write(0, 0, 'Greetings!!')
```

この関数内で、「Greetings !!」は、LCD の行 0+1、列 0+1 にプリントされる文字である。

これで、「Greetings!が見える SunFounder」が LCD に表示される。

現象画像



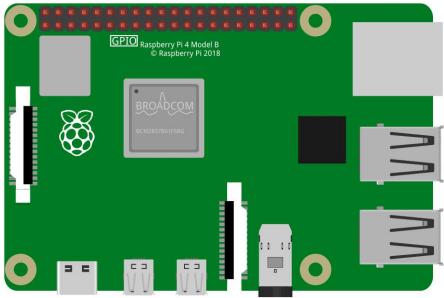
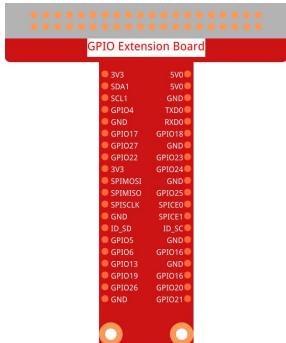
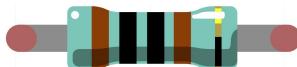
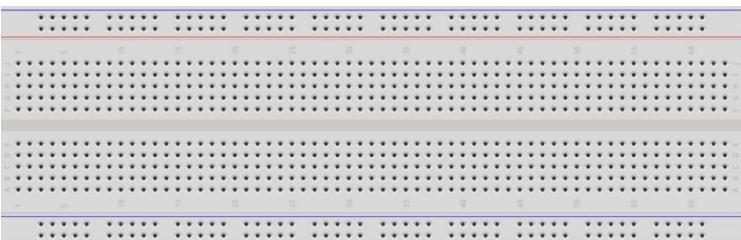
1.2 音声

1.2.1 Active Buzzer

前書き

このレッスンでは、PNPトランジスタでアクティブブザーを鳴らす方法を学習する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	アクティブブザー*1
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">3V3 SVIDSDA1 GNDSCL1 TXD0GND RXD0GPIO4 GPIO18GPIO17 GPIO15GPIO27 GPIO23GPIO22 GPIO233V3 GPIO23SPIMOSI GNDSPIMISO GPIO25SPISCLK SPICE09ID_SD ID_SDGPIOS GNDGPIO6 GPIO16GPIO13 GNDGPIO19 GPIO16GPIO26 GPIO23GND GND	
40 ピンケーブル*1		抵抗器 (1kΩ) *1
		
ブレッドボード*1		何本のジャンパー線
		
		S8550 PNP トランジスター*1
		

原理

ブザー

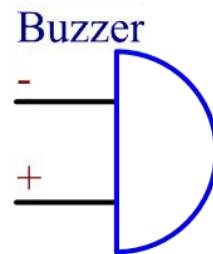
統合された構造を持つ電子ブザーの一種として、DC電源によって電圧供給されるブザーは、コンピューター、プリンター、コピー機、警報器、電子玩具、自動車用電子装置、電話、タイマー、その他の電子製品または音声装置で広く使用されている。ブザーは、アクティブとパッシブに分類できる（次の図を参照）。ピンが上を向くようにブザーを回し、緑色の回路基板を備えたブザーはパッシブブザーで、黒いテープで囲まれたブザーはアクティブである。

アクティブブザーとパッシブブザーの違い：



アクティブブザーとパッシブブザーの違いは次の通りである：アクティブブザーには振動源が内蔵されているため、通電すると音が鳴る。ただし、パッシブブザーにはそのような振動源がないため、DC信号が使用されてもビープ音は鳴らない。代わりに、周波数が2K~5Kの方波を使用して駆動する必要がある。アクティブブザーは、多くの場合、複数の発振回路が内蔵されているため、パッシブブザーよりも高価である。

以下はブザーの電気記号である。両極の2つのピンが搭載されている。表面の+は陽極を表し、もう1つは陰極を表す。

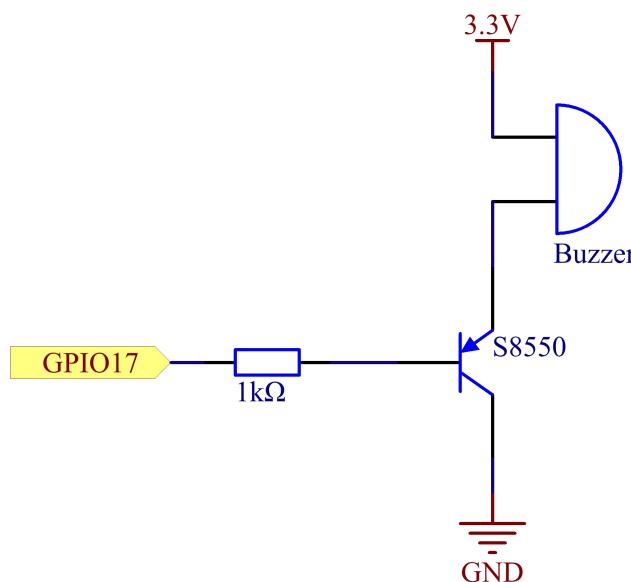


ブザーのピンをチェックすることができ、長い方が陽極で、短い方が陰極である。接続時にそれらを混同しないでください。混同すると、ブザーが鳴らない。

回路図

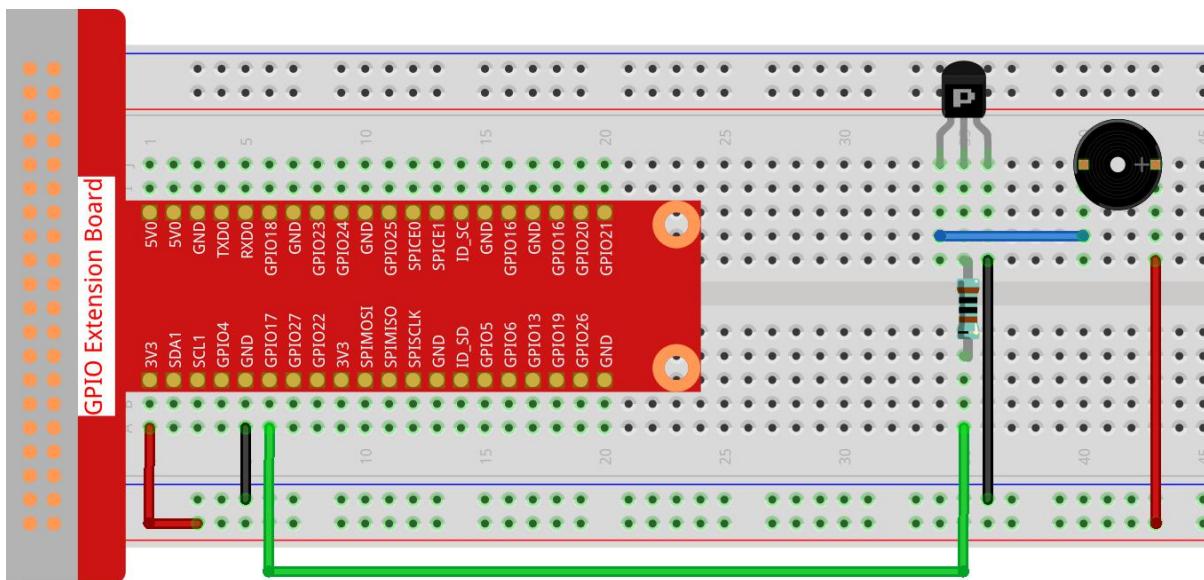
この実験では、アクティブブザー、PNPトランジスタ、および1k抵抗器をトランジスタのベースとGPIOの間に使用して、トランジスタを保護する。Raspberry Pi出力のGPIO17にプログラミングによって低レベル(0V)が供給されると、電流飽和のためトランジスタが導通し、ブザーが音を出す。しかし、Raspberry PiのIOに高レベルが供給されると、トランジスターが切断され、ブザーは音を出さない。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ1: 回路を作る。(ブザーの両極に注意してください: +ラベルが付いている方が正極で、もう一方が負極である。)



➤ C 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.2.1/
```

ステップ 3: コードをコンパイルする。

```
gcc 1.2.1_ActiveBuzzer.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードが実行されると、ブザーが鳴く。

コード

```
#include <wiringPi.h>
#include <stdio.h>

#define BeepPin 0
int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(BeepPin, OUTPUT); //set GPIO0 output
    while(1){
        //beep on
        printf("Buzzer on\n");
        digitalWrite(BeepPin, LOW);
        delay(100);
        printf("Buzzer off\n");
        //beep off
        digitalWrite(BeepPin, HIGH);
        delay(100);
    }
    return 0;
}
```

コードの説明

```
digitalWrite(BeepPin, LOW);
```

この実験ではアクティブブザーを使用しているため、直流に接続すると自動的に音が鳴く。このスケッチは、I/O ポートを低レベル (0V) に設定して、トランジスタを管理し、ブザーを鳴らすためのものである。

```
digitalWrite(BeepPin, HIGH);
```

I/O ポートを高レベル (3.3V) に設定するため、トランジスターは通電されず、ブザーは鳴らない。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: 実行する。

```
sudo python3 1.2.1_ActiveBuzzer.py
```

コードが実行されると、ブザーが鳴く。

コード

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as buzzer pin
BeepPin = 17

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.HIGH)

def main():
    while True:
        # Buzzer on (Beep)
        print ('Buzzer On')
        GPIO.output(BeepPin, GPIO.LOW)
        time.sleep(0.1)
        # Buzzer off
        print ('Buzzer Off')
        GPIO.output(BeepPin, GPIO.HIGH)
```

```
time.sleep(0.1)

def destroy():
    # Turn off buzzer
    GPIO.output(BeepPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
GPIO.output(BeepPin, GPIO.LOW)
```

ビープ音を鳴らすには、ブザーピンを低レベルに設定してください。

```
time.sleep(0.1)
```

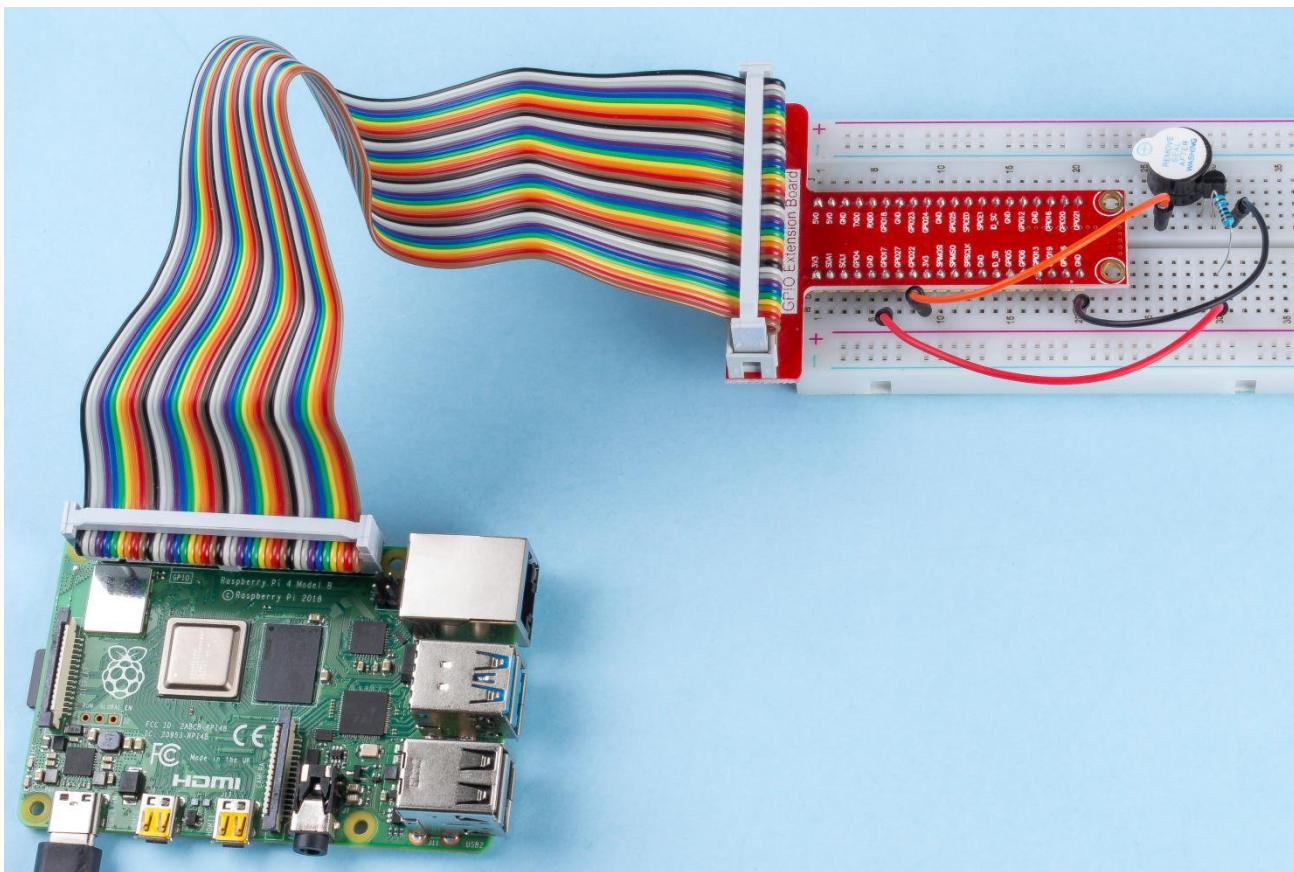
0.1秒待つ。このパラメーターを変更して、スイッチング周波数を変更する。

ご注意: これは音の周波数ではない。アクティブブザーは音の周波数を変更できない。

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

ブザーを閉じる。

現象画像

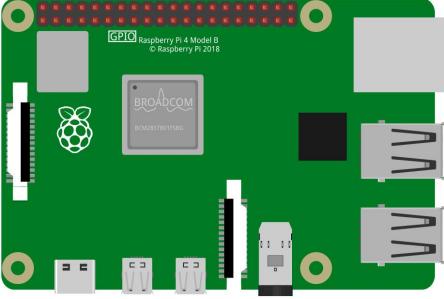
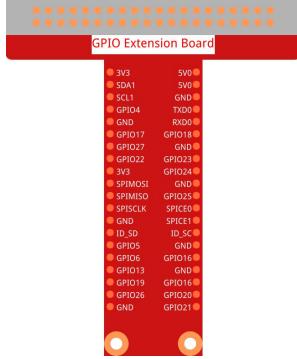
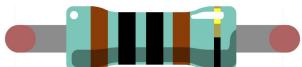
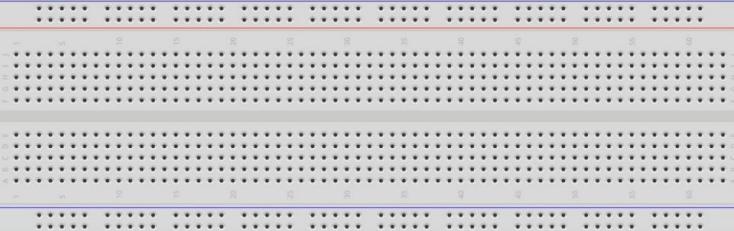


1.2.2 Passive Buzzer

前書き

このレッスンでは、パッシブブザーで音楽を再生する方法を学習する。

部品

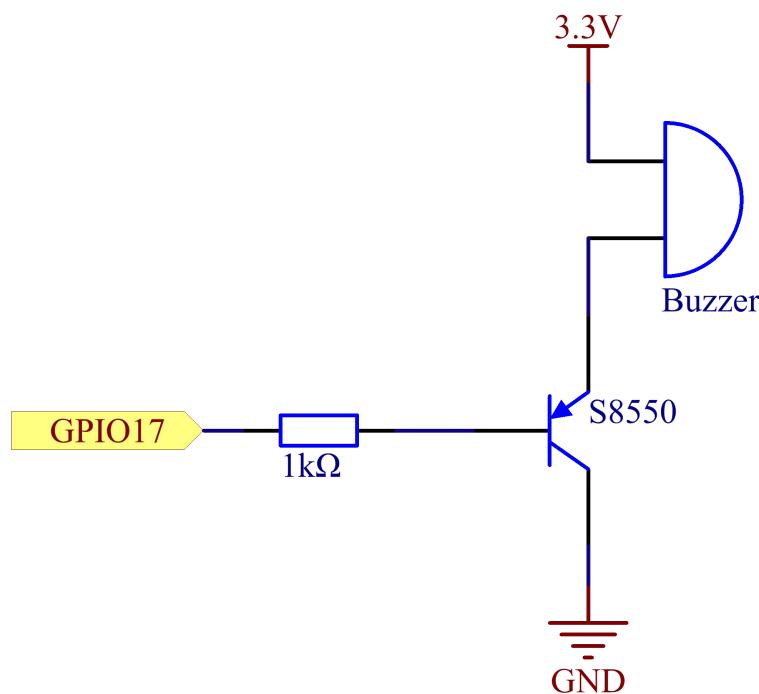
Raspberry Pi 本体*1	T 拡張ボード*1	パッシブブザー*1
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">● VIO ● SVIO ●● GND1 ● GND2 ●● SCL1 ● GND3 ●● GPIO4 ● TXD0 ●● GND ● RXD0 ●● GPIO17 ● GPIO18 ●● GPIO27 ● GND4 ●● GPIO22 ● GPIO23 ●● 3V ● GPIO24 ●● SPI_MOSI ● GND5 ●● SPI_MISO ● GND6 ●● SPI_SCLK ● SPI_CE0 ●● GND ● SPI_CE1 ●● ID_SD ● ID_SC ●● GPIO5 ● GND7 ●● GPIO6 ● GPIO16 ●● GPIO13 ● GND8 ●● GPIO19 ● GPIO16 ●● GPIO26 ● GPIO20 ●● GND ● GPIO21 ●	
40 ピンケーブル*1		抵抗器 (1kΩ) *1
		
ブレッドボード*1		何本のジャンパー線
		
		S8550 PNP トランジスタ -*1
		

回路図

この実験では、トランジスタを保護するために、トランジスタのベースと GPIO の間にパッシブブザー、PNPトランジスタ、および1k 抵抗器を使用する。

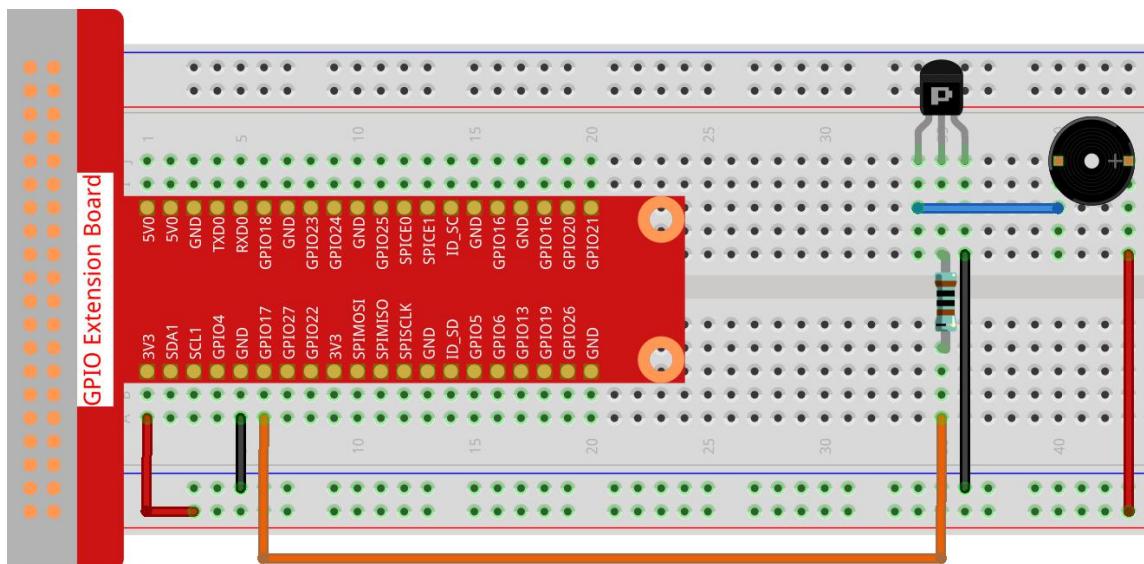
GPIO17に異なる周波数が与えられると、パッシブブザーは異なる音を出す。このようにして、ブザーは音楽を再生する。

Tボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.2.2/
```

ステップ3: コンパイルする。

```
gcc 1.2.2_PassiveBuzzer.c -lwiringPi
```

ステップ4: 実行。

```
sudo ./a.out
```

コードが実行されると、ブザーが音楽を再生する。

コード

```
#include <wiringPi.h>
#include <softTone.h>
#include <stdio.h>

#define BuzPin    0

#define CL1    131
#define CL2    147
#define CL3    165
#define CL4    175
#define CL5    196
#define CL6    221
#define CL7    248

#define CM1    262
#define CM2    294
#define CM3    330
#define CM4    350
#define CM5    393
#define CM6    441
#define CM7    495

#define CH1    525
#define CH2    589
#define CH3    661
#define CH4    700
#define CH5    786
#define CH6    882
```

```
#define CH7 990

int song_1[] = {CM3,CM5,CM6,CM3,CM2,CM3,CM5,CM6,CH1,CM6,CM5,CM1,CM3,CM2,
                CM2,CM3,CM5,CM2,CM3,CM3,CL6,CL6,CL6,CM1,CM2,CM3,CM2,CL7,
                CL6,CM1,CL5};

int beat_1[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,1,2,1,1,
                1,1,1,1,1,3};

int song_2[] = {CM1,CM1,CM1,CL5,CM3,CM3,CM1,CM1,CM3,CM5,CM5,CM4,CM3,CM2,
                CM2,CM3,CM4,CM4,CM3,CM2,CM3,CM1,CM1,CM3,CM2,CL5,CL7,CM2,CM1
                };

int beat_2[] = {1,1,1,3,1,1,1,3,1,1,1,1,1,1,3,1,1,1,2,1,1,1,3,1,1,1,3,3,2,3};

int main(void)
{
    int i, j;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    if(softToneCreate(BuzPin) == -1){
        printf("setup softTone failed !");
        return 1;
    }

    while(1){
        printf("music is being played...\n");

        for(i=0;i<sizeof(song_1)/4;i++){
            softToneWrite(BuzPin, song_1[i]);
            delay(beat_1[i] * 500);
        }

        for(i=0;i<sizeof(song_2)/4;i++){
            softToneWrite(BuzPin, song_2[i]);
            delay(beat_2[i] * 500);
        }
    }
}
```

```

    }
    return 0;
}
}
```

コードの説明

```
#define CL1 131
#define CL2 147
#define CL3 165
#define CL4 175
#define CL5 196
#define CL6 221
#define CL7 248

#define CM1 262
#define CM2 294
...
```

各音の周波数は以下のように示している。CL - 低音、CM - 中音、CH - 高音、1 ~ 7 は音 C、D、E、F、G、A、B に対応する。

```
int song_1[] = {CM3,CM5,CM6,CM3,CM2,CM3,CM5,CM6,CH1,CM6,CM5,CM1,CM3,CM2,
                CM2,CM3,CM5,CM2,CM3,CM3,CL6,CL6,CL6,CM1,CM2,CM3,CM2,CL7,
                CL6,CM1,CL5};
int beat_1[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,1,1,2,1,1,
                1,1,1,1,1,1,3};
```

配列 song_1 [] は曲の楽譜を保存する。beat_1 [] は曲の各音符の拍を表す（1 拍ごとに 0.5 秒）。

```
if(softToneCreate(BuzPin) == -1){
    printf("setup softTone failed !");
    return 1;
```

これにより、ソフトウェア制御のトーンピンが作成される。任意の GPIO ピンを使用でき、ピンの番号は使用した wiringPiSetup () 関数の番号になる。成功した場合の戻り値は 0 である。それ以外の場合は、グローバル変数 errno をチェックして、何が問題なのかを確認する必要がある。

```
for(i=0;i<sizeof(song_1)/4;i++){
    softToneWrite(BuzPin, song_1[i]);
    delay(beat_1[i] * 500);
}
```

song_1 を再生するために for statement を使用する。

判断条件 `i < sizeof(song_1) / 4` では、配列 song_1 [] は整数のデータ型の配列であり、各要素は 4 拍を占有するため、「devide by 4」が使用されている。

song_1 の要素の数（音符の数）は、`sizeof (song_4)` を 4 で割ることによって得られる。

各音符を拍* 500ms で再生できるようにするには、関数 `delay (beat_1 [i] * 500)` が呼び出される。

`softToneWrite (BuzPin, song_1 [i])` のプロトタイプ：

```
void softToneWrite (int pin, int freq);
```

これにより、指定されたピンのトーン周波数値が更新される。周波数を 0 に設定するまで、トーンの再生は停止しない。

➤ Python 言語ユーザー向け

ステップ 2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 1.2.2_PassiveBuzzer.py
```

コードが実行されると、ブザーが音楽を再生する。

コード

```
import RPi.GPIO as GPIO
import time

Buzzer = 11

CL = [0, 131, 147, 165, 175, 196, 211, 248]      # Frequency of Bass tone in C major
CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Midrange tone in C major
CH = [0, 525, 589, 661, 700, 786, 882, 990]      # Frequency of Treble tone in C major

song_1 = [ CM[3], CM[5], CM[6], CM[3], CM[2], CM[3], CM[5], CM[6], # Notes of song1
           CM[1], CM[6], CM[5], CM[1], CM[3], CM[2], CM[2], CM[3],
           CM[5], CM[2], CM[3], CM[3], CL[6], CL[6], CM[1],
           CM[2], CM[3], CM[2], CL[7], CL[6], CM[1], CL[5] ]

beat_1 = [ 1, 1, 3, 1, 1, 3, 1, 1,          # Beats of song 1, 1 means 1/8 beat
            1, 1, 1, 1, 1, 3, 1,
            1, 3, 1, 1, 1, 1, 1,
```

```

1, 2, 1, 1, 1, 1, 1, 1,
1, 1, 3 ]

song_2 = [ CM[1], CM[1], CM[1], CL[5], CM[3], CM[3], CM[1], # Notes of song2
          CM[1], CM[3], CM[5], CM[5], CM[4], CM[3], CM[2], CM[2],
          CM[3], CM[4], CM[4], CM[3], CM[2], CM[3], CM[1], CM[1],
          CM[3], CM[2], CL[5], CL[7], CM[2], CM[1]      ]

beat_2 = [ 1, 1, 2, 2, 1, 1, 2, 2,                      # Beats of song 2, 1 means 1/8 beat
           1, 1, 2, 2, 1, 1, 3, 1,
           1, 2, 2, 1, 1, 2, 2, 1,
           1, 2, 2, 1, 1, 3 ]


def setup():
    GPIO.setmode(GPIO.BOARD)          # Numbers GPIOs by physical location
    GPIO.setup(Buzzer, GPIO.OUT)       # Set pins' mode is output
    global Buzz                      # Assign a global variable to replace GPIO.PWM
    Buzz = GPIO.PWM(Buzzer, 440)      # 440 is initial frequency.
    Buzz.start(50)                  # Start Buzzer pin with 50% duty cycle


def loop():
    while True:
        print ('\n      Playing song 1...')
        for i in range(1, len(song_1)):      # Play song 1
            Buzz.ChangeFrequency(song_1[i]) # Change the frequency along the song note
            time.sleep(beat_1[i] * 0.5)     # delay a note for beat * 0.5s
        time.sleep(1)                      # Wait a second for next song.

        print ('\n\n      Playing song 2...')
        for i in range(1, len(song_2)):      # Play song 1
            Buzz.ChangeFrequency(song_2[i]) # Change the frequency along the song note
            time.sleep(beat_2[i] * 0.5)     # delay a note for beat * 0.5s


def destory():
    Buzz.stop()                      # Stop the buzzer
    GPIO.output(Buzzer, 1)            # Set Buzzer pin to High
    GPIO.cleanup()                   # Release resource


if __name__ == '__main__':          # Program start from here
    setup()
    try:

```

```

loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
destroy()

```

コードの説明

```

CL = [0, 131, 147, 165, 175, 196, 211, 248]      # Frequency of Bass tone in C major
CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Midrange tone in C major
CH = [0, 525, 589, 661, 700, 786, 882, 990]      # Frequency of Treble tone in C major

```

これらは各音符の周波数である。番号 1～7 が音色の CDEFGAB に対応するように、最初の 0 は CL[0] をスキップする。

```

song_1 = [ CM[3], CM[5], CM[6], CM[3], CM[2], CM[3], CM[5], CM[6],
           CH[1], CM[6], CM[5], CM[1], CM[3], CM[2], CM[2], CM[3],
           CM[5], CM[2], CM[3], CM[3], CL[6], CL[6], CL[6], CM[1],
           CM[2], CM[3], CM[2], CL[7], CL[6], CM[1], CL[5] ]

```

これらの配列は歌の音符である。

```

beat_1 = [ 1, 1, 3, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 3, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 3 ]

```

すべての音符の拍（各番号）は、 $\frac{1}{8}$ 拍で、つまり 0.5 秒を表す。

```

Buzz = GPIO.PWM(Buzzer, 440)
Buzz.start(50)

```

ピンブザーを PWM ピンとして定義し、その周波数を 440 に設定し、Buzz.start (50) を使用して PWM を実行する。さらに、デューティサイクルを 50% に設定する。

```

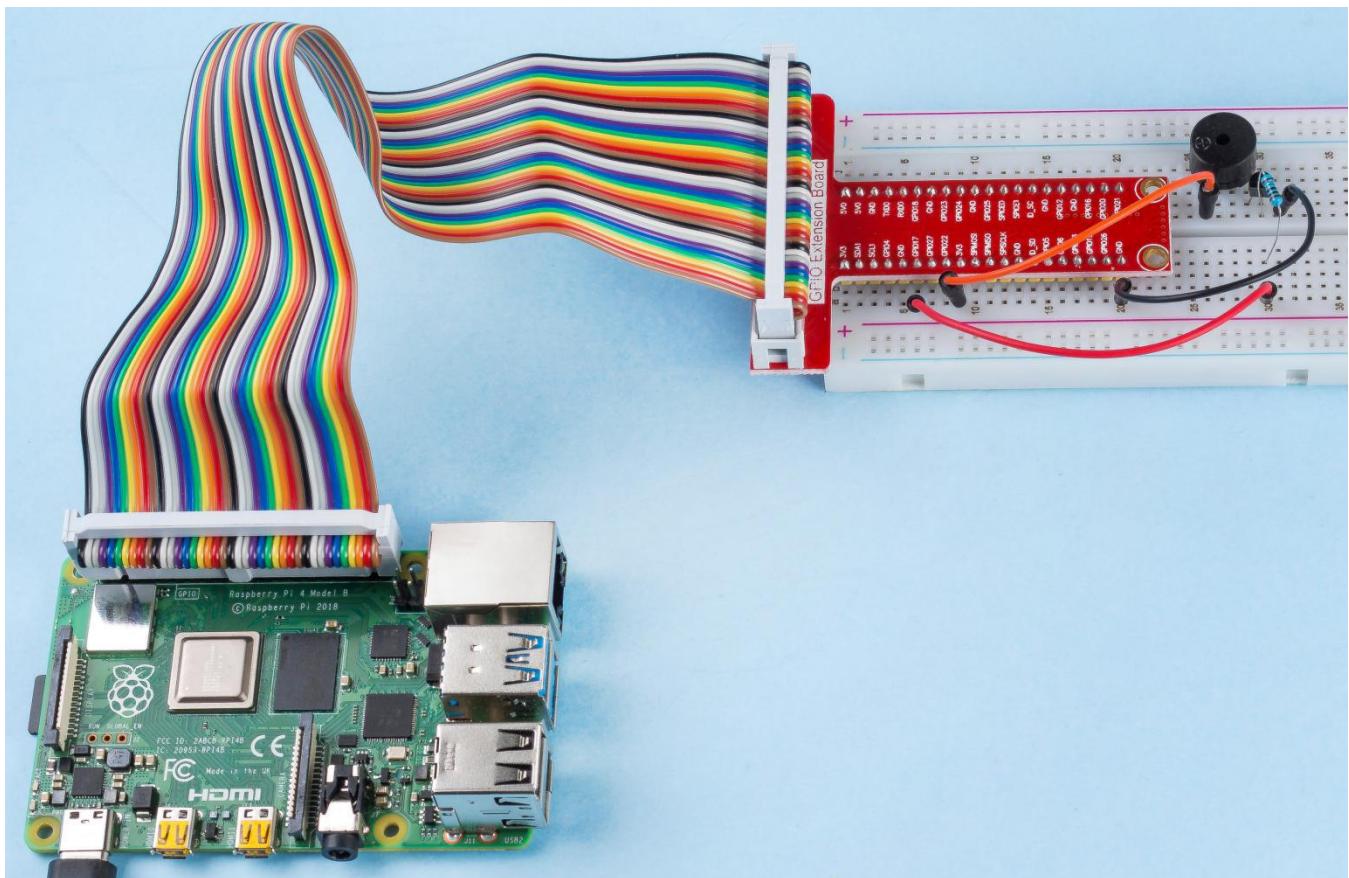
for i in range(1, len(song_1)):
    Buzz.ChangeFrequency(song_1[i])
    time.sleep(beat_1[i] * 0.5)

```

for 文を実行すると、ブザーは配列 song_1 [] の音符を beat_1 [] 配列の拍で再生する。

これで、パッシブブザーが音楽を再生していることが聞こえる。

現象画像



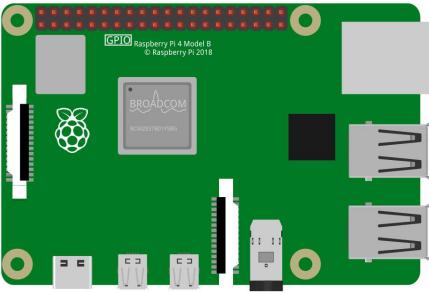
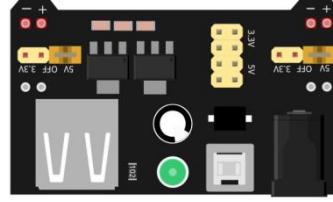
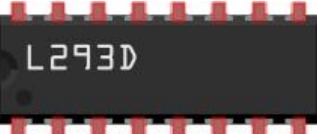
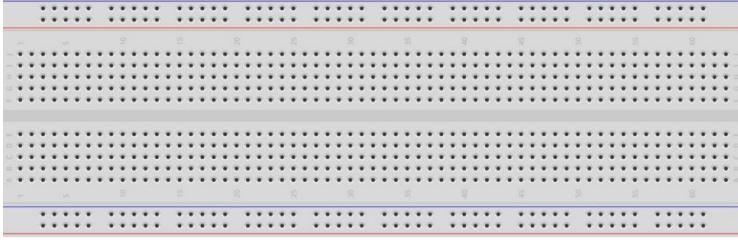
1.3 ドライバー

1.3.1 Motor

前書き

このレッスンでは、L293D を使用して DC モーターを駆動し、時計回りと反時計回りに回転させる方法を学習する。安全上の理由で、DC モーターは大電流を必要とするため、ここでは電源モジュールを使用してモーターに電力を供給する。

部品

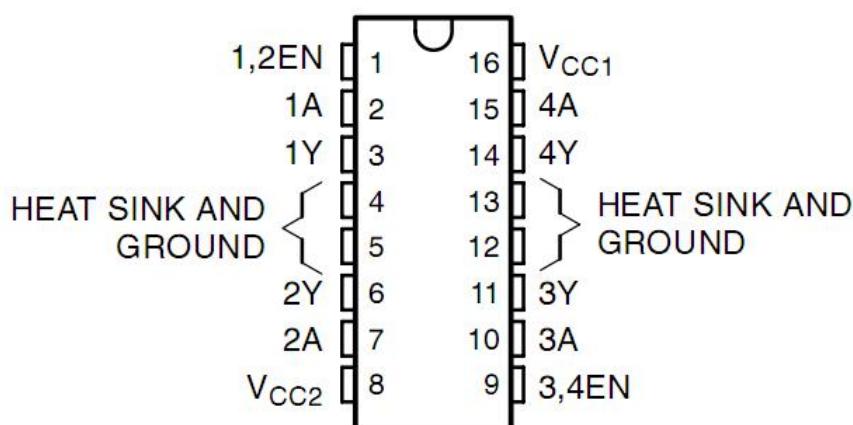
Raspberry Pi 本体*1	T 拡張ボード*1	電源モジュール*1 (9V バッテリーとバックル付き)
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">3V3 SVOSDA1 GNDSCL1 TXD0GPIO4 RXD0GND GNDGPIO17 GPIO18GPIO27 GPIO26GPIO22 GPIO233V0 GNDSPI MOSI GNDSPI MISO GPIO25SPI SCLK SPI CE0GND SPI CE1ID_SD ID_SCGPIO5 GNDGPIO6 GPIO16GPIO13 GNDGPIO19 GPIO16GPIO26 GPIO20GND GNDGPIO21 GPIO21	
40 ピンケーブル*1	L293D*1	何本のジャンパー線
		
ブレッドボード*1	DC モーター*1	
		

原理

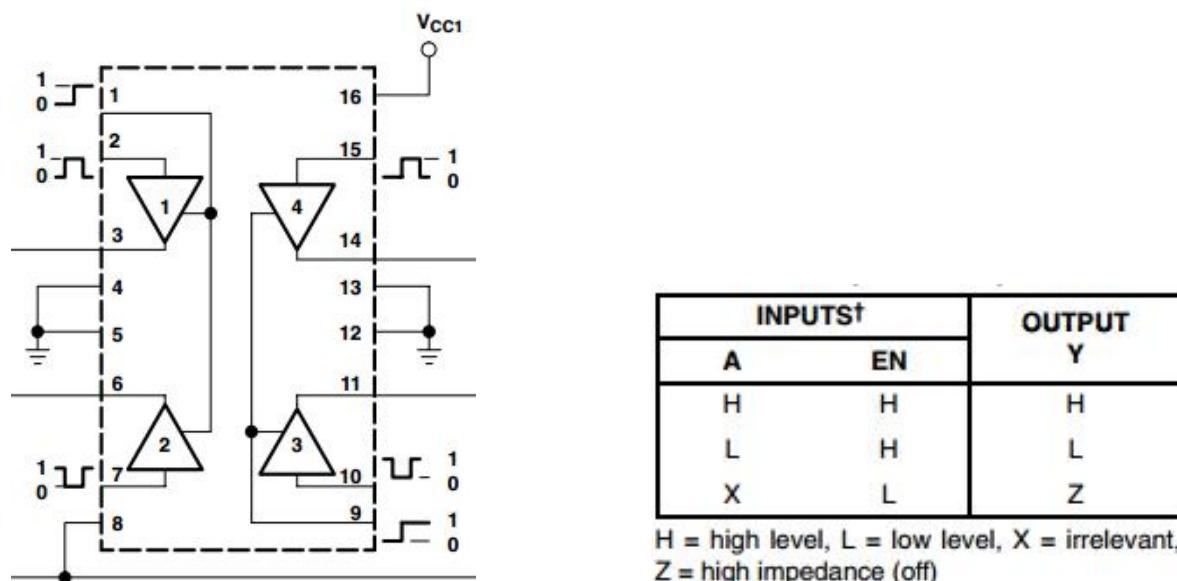
L293D

L293D は、高電圧と高電流のチップで統合された 4 チャネルモータードライバーである。標準の DTL、TTL ロジックレベルに接続し、誘導負荷（リレーコイル、DC、ステッピングモーターなど）およびパワースイッチングトランジスタなどを駆動するように設計される。DC モーターは、DC 電気エネルギーを機械エネルギーに変換するデバイスである。それらは、優れた速度調整性能の利点により、電気駆動装置で広く使用されている。

ピンの図については、以下の図を参照してください。L293D には、電源用の 2 つのピン (V_{CC1} と V_{CC2}) がある。 V_{CC2} はモーターに電力を供給し、 V_{CC1} はチップに電力を供給するために使用される。ここでは小型の DC モーターが使用されているため、両方のピンを +5V に接続してください。



以下は L293D の内部構造である。ピン EN はイネーブルピンであり、高レベルでのみ機能する。A は入力を表し、Y は出力を表す。それらの間の関係は右下に見ることができる。ピン EN が High レベルのとき、A が High の場合、Y は High レベルを出力する。A が Low の場合、Y は Low レベルを出力する。ピン EN が Low レベルの場合、L293D は機能しない。



DC モーター



これは 5V DC モーターである。銅板の 2 つの端子に 1 つの高レベルと 1 つの低レベルを与えると回転する。便宜上、ピンを溶接することができる。

サイズ: 25 * 20 * 15MM

動作電圧: 1-6V

フリーラン電流 (3V) : 70mA

フリーラン速度 (3V) : 13000RPM

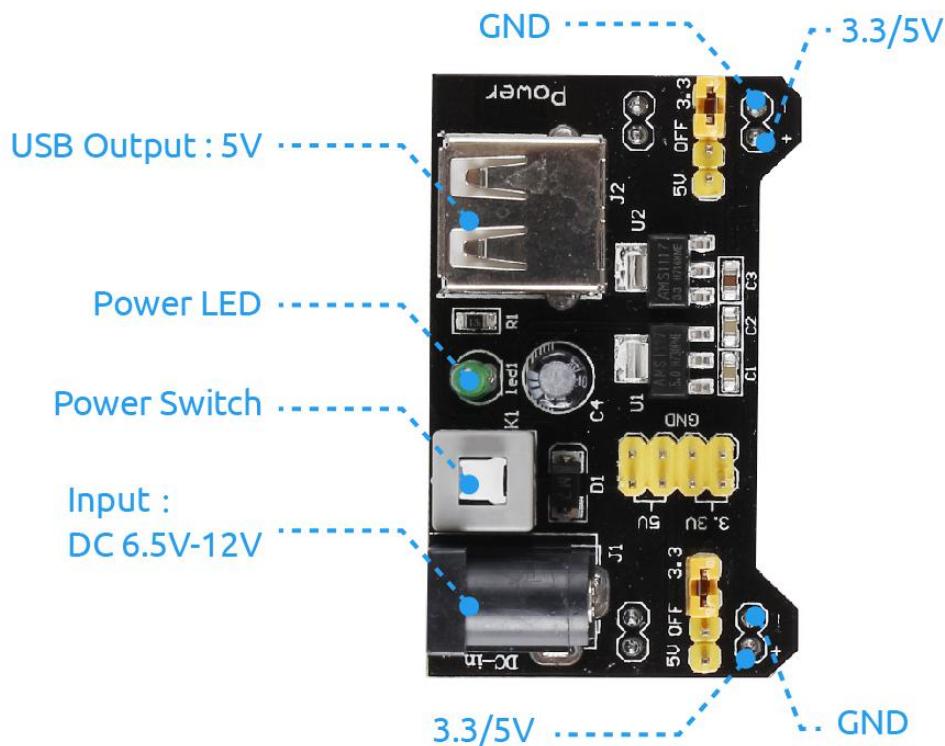
ストール電流 (3V) : 800mA

軸径: 2mm

電源モジュール

この実験では、特に起動時と停止時にモーターを駆動するために大きな電流が必要である。これは、Raspberry Pi の通常の動作を大幅に妨害する可能性がある。そのため、このモジュールによってモーターに個別に電力を供給し、安全かつ着実に動作させる。

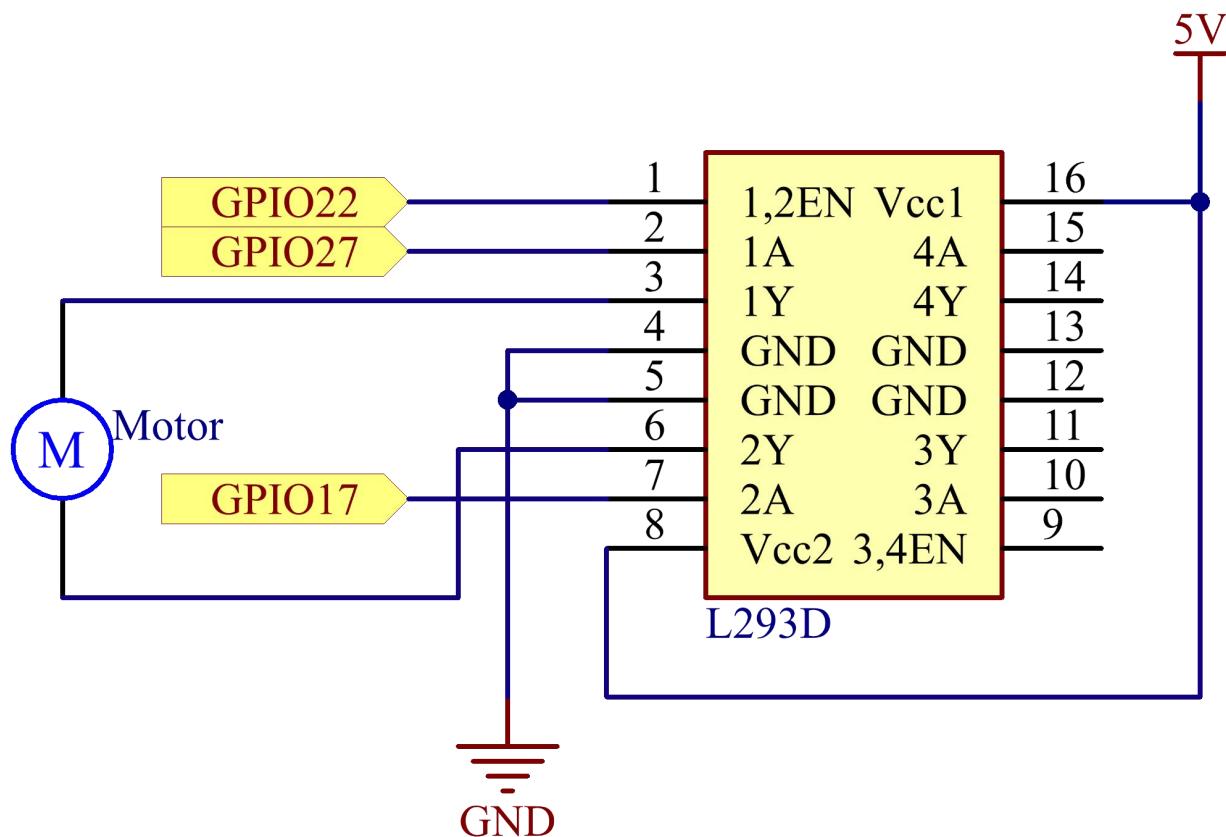
ブレッドボードに差し込むだけで電力を供給できる。3.3V と 5V の電圧を提供し、付属のジャンパークリップを介してどちらでも接続できる。



回路図

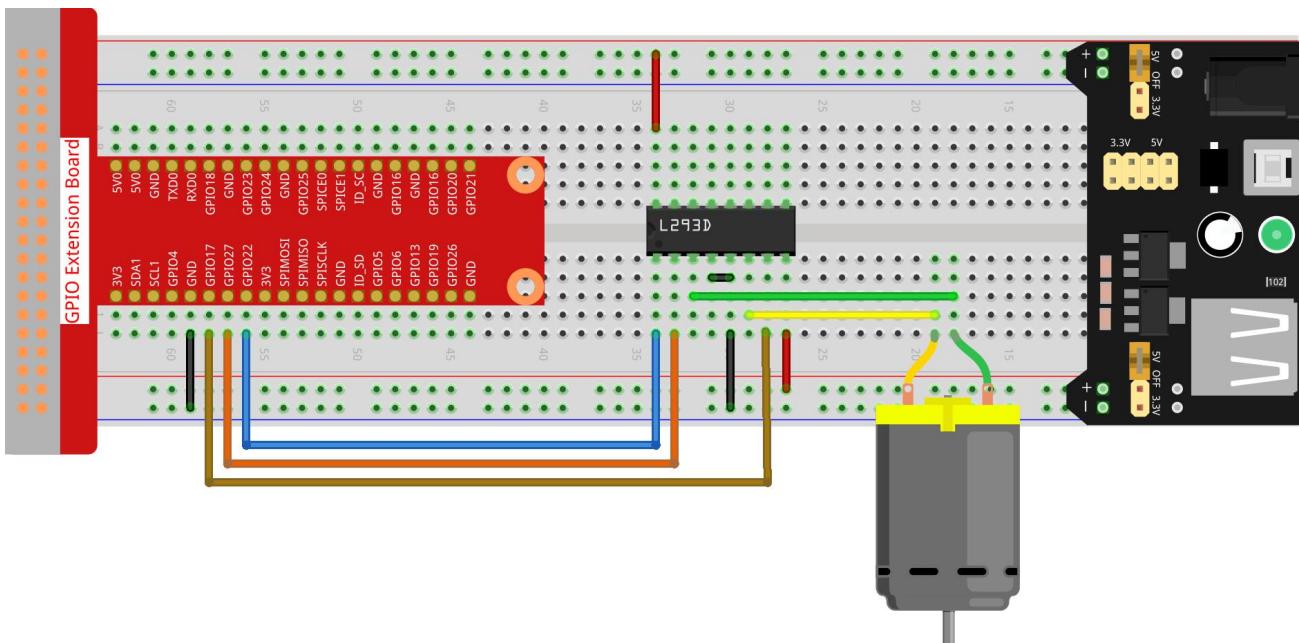
電源モジュールをブレッドボードに差し込み、ジャンパーを 5V のピンに挿入すると、5V の電圧が出力される。L293D のピン 1 を GPIO22 に接続し、それを高レベルに設定する。ピン 2 を GPIO27 に、ピン 7 を GPIO17 に接続し、一方のピンを high に、もう一方のピンを high に設定する。したがって、モーターの回転方向を変更できる。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



実験手順

ステップ1: 回路を作る。



ご注意: 電源モジュールはキットの9Vバッテリーバックルで9Vバッテリーを適用できる。電源モジュールのジャンパキャップをブレッドボードの5Vバスストリップに挿入する



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.1/
```

ステップ3: .コンパイルする。

```
gcc 1.3.1_Motor.c -lwiringPi
```

ステップ4: EXEファイルを実行する。

```
sudo ./a.out
```

コードが実行されると、モーターは最初に5秒間時計回りに回転し、それから5秒間停止し、その後5秒間反時計回りに回転してから5秒間停止する。この一連の動作は繰り返し実行される。

コード

```
#include <wiringPi.h>
#include <stdio.h>

#define MotorPin1      0
#define MotorPin2      2
#define MotorEnable    3

int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(MotorPin1, OUTPUT);
    pinMode(MotorPin2, OUTPUT);
    pinMode(MotorEnable, OUTPUT);
    while(1){
        printf("Clockwise\n");
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, HIGH);
        digitalWrite(MotorPin2, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Stop\n");
        digitalWrite(MotorEnable, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Anti-clockwise\n");
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, LOW);
        digitalWrite(MotorPin2, HIGH);
        for(i=0;i<3;i++){
            delay(1000);
        }
    }
}
```

```
printf("Stop\n");
digitalWrite(MotorEnable, LOW);
for(i=0;i<3;i++){
    delay(1000);
}
return 0;
}
```

コードの説明

```
digitalWrite(MotorEnable, HIGH);
```

L239D を有効にする。

```
digitalWrite(MotorPin1, HIGH);
digitalWrite(MotorPin2, LOW);
```

2A (ピン 7) に高レベルを設定する。1,2EN (ピン 1) は高レベルなので、2Y は高レベルを出力する。

1A に低レベルを設定すると、1Y が低レベルを出力し、モーターが回転する。

```
for(i=0;i<3;i++){
    delay(1000);
}
```

このループは $3 * 1000\text{ms}$ 遅延する。

```
digitalWrite(MotorEnable, LOW)
```

1,2EN (ピン 1) が低レベルの場合、L293D は機能しない。モーターが回転を停止する。

```
digitalWrite(MotorPin1, LOW)
digitalWrite(MotorPin2, HIGH)
```

モーターの電流を逆にすると、モーターが逆回転する。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ3: 実行する。

```
sudo python3 1.3.1_Motor.py
```

コードが実行されると、モーターは最初に5秒間時計回りに回転し、それから5秒間停止し、その後5秒間反時計回りに回転してから5秒間停止する。この一連の動作は繰り返し実行される。

コード

```
import RPi.GPIO as GPIO
import time

# Set up pins
MotorPin1    = 17
MotorPin2    = 27
MotorEnable = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set pins to output
    GPIO.setup(MotorPin1, GPIO.OUT)
    GPIO.setup(MotorPin2, GPIO.OUT)
    GPIO.setup(MotorEnable, GPIO.OUT, initial=GPIO.LOW)

# Define a motor function to spin the motor
# direction should be
# 1(clockwise), 0(stop), -1(counterclockwise)
def motor(direction):
    # Clockwise
    if direction == 1:
        # Set direction
        GPIO.output(MotorPin1, GPIO.HIGH)
        GPIO.output(MotorPin2, GPIO.LOW)
        # Enable the motor
        GPIO.output(MotorEnable, GPIO.HIGH)
        print ("Clockwise")
    # Counterclockwise
```

```

if direction == -1:
    # Set direction
    GPIO.output(MotorPin1, GPIO.LOW)
    GPIO.output(MotorPin2, GPIO.HIGH)
    # Enable the motor
    GPIO.output(MotorEnable, GPIO.HIGH)
    print ("Counterclockwise")

# Stop
if direction == 0:
    # Disable the motor
    GPIO.output(MotorEnable, GPIO.LOW)
    print ("Stop")

def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}
    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)
        # Anticlockwise
        motor(directions['CCW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)

def destroy():
    # Stop the motor
    GPIO.output(MotorEnable, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:

```

```
main()
# When 'Ctrl+C' is pressed, the program
# destroy() will be executed.
except KeyboardInterrupt:
    destroy()
```

コードの説明

```
def motor(direction):
    # Clockwise
    if direction == 1:
        # Set direction
        GPIO.output(MotorPin1, GPIO.HIGH)
        GPIO.output(MotorPin2, GPIO.LOW)
        # Enable the motor
        GPIO.output(MotorEnable, GPIO.HIGH)
        print ("Clockwise")
    ...
    ...
```

変数が direction である関数 motor () を作成する。direction = 1 の条件が満たされると、モーターは時計回りに回転する。direction = -1 の場合、モーターは反時計回りに回転する。そして、direction = 0 の条件下では、回転を停止する。

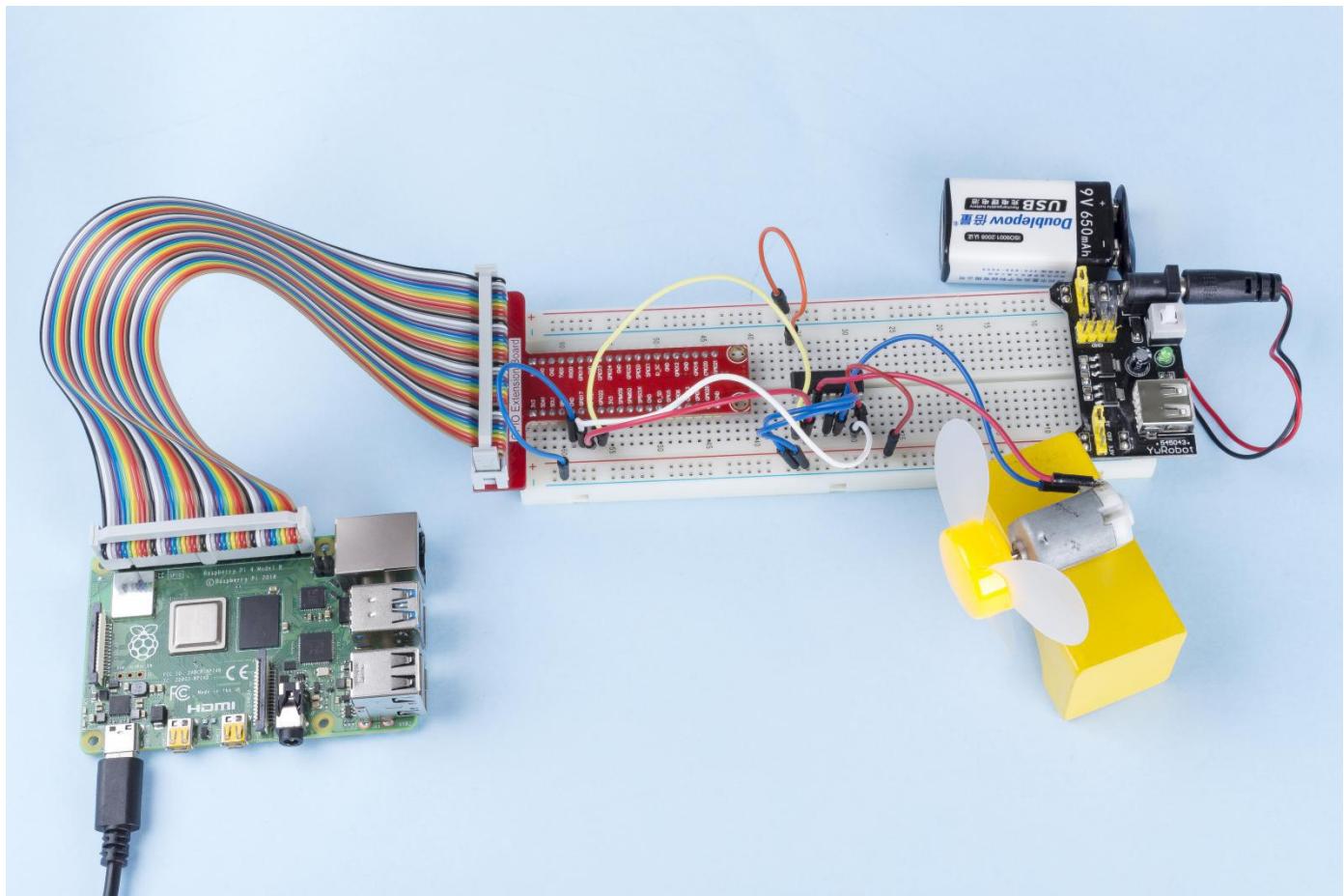
```
def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}
    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)
        # Anticlockwise
        motor(directions['CCW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)
```

メイン（）関数で、CW が 1、CCW の値が -1、0 が Stop を指す配列 directions[]を作成する。

コードが実行されると、モーターは最初に 5 秒間時計回りに回転し、それから 5 秒間停止し、その後 5 秒間反時計回りに回転してから 5 秒間停止する。この一連の動作は繰り返し実行される。

これで、モーターブレードが回転していることが分かる。

現象画像

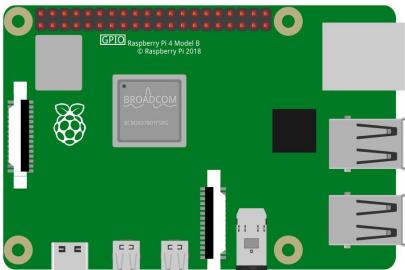
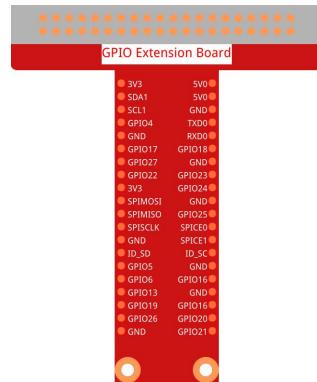
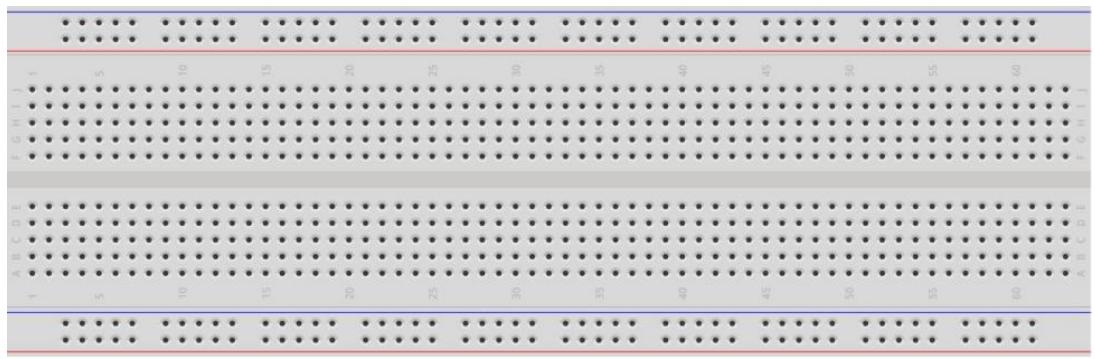


1.3.2 Servo

前書き

このレッスンでは、サーボを回転させる方法を学ぶ。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	サーボ *1 何本のジャンパー線
  		
40 ピンケーブル*1		
		
ブレッドボード*1		
		

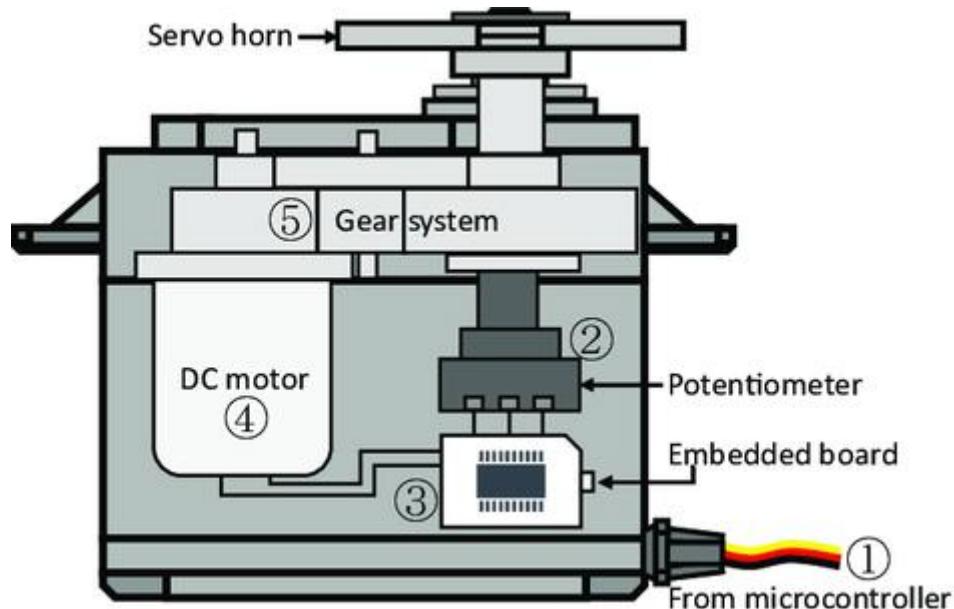
原理

サーボ

サーボは一般に、ケース、シャフト、ギアシステム、ポテンショメーター、DCモーター、および内蔵式ボードで構成されている。



これは次のように動作する：マイクロコントローラーは PWM 信号をサーボに送信し、サーボの内蔵式ボードは信号ピンを介して信号を受信し、内部のモーターを制御して回転させる。その結果、モーターはギアシステムを駆動し、減速後にシャフトを駆動する。サーボのシャフトとポテンショメーターは接続されている。シャフトが回転する時、ポテンショメーターが駆動されるため、ポテンショメーターは電圧信号を内蔵式ボードに出力する。その後、ボードは現在の位置に基づいて回転の方向と速度を決めるため、定義された位置で正確に停止してそのまま保持する。

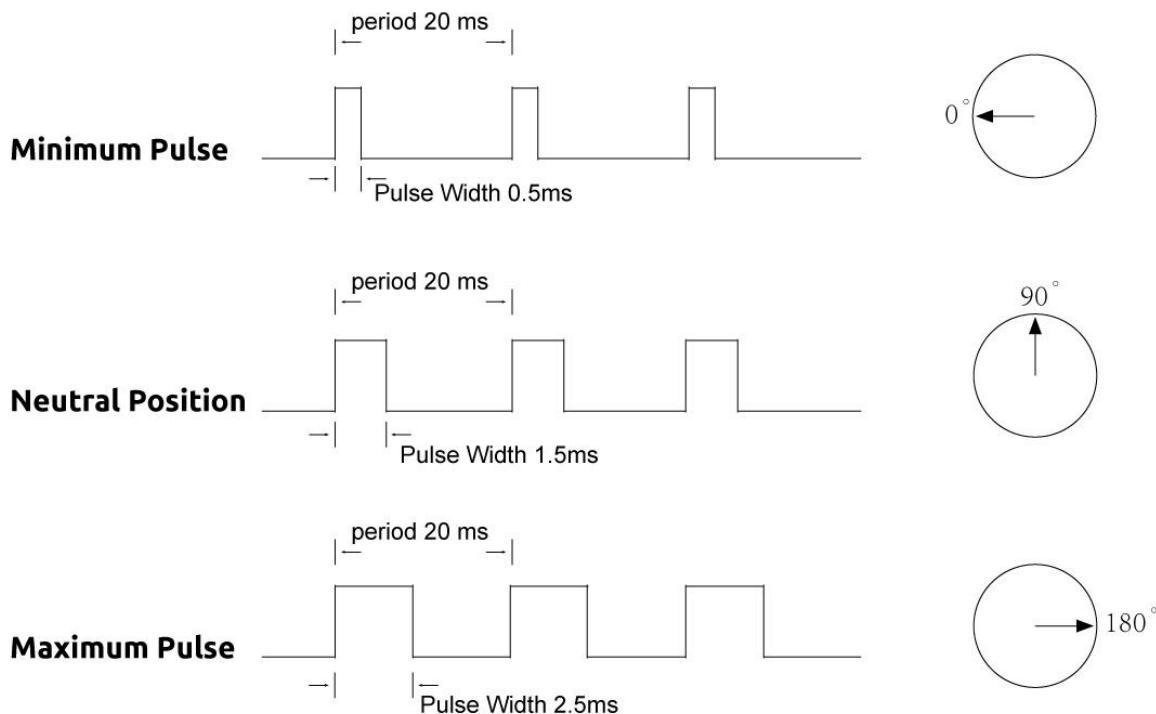


角度は制御ワイヤに適用されるパルスの持続時間によって決まる。これはパルス幅変調と呼ばれる。サーボは 20 ミリ秒ごとに 1 パルスを期待している。

パルスの長さにより、モーターの回転距離が決まる。たとえば、1.5ms

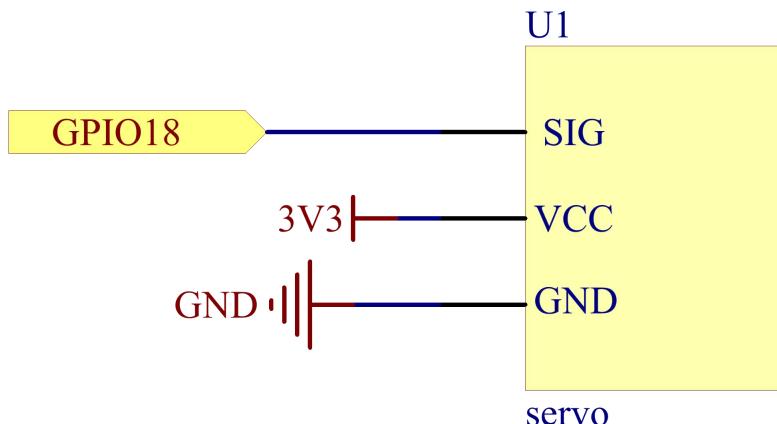
パルスは、モーターを 90 度の位置（ニュートラル位置）に回転させる。

1.5 ms 未満のパルスがサーボに送信されると、サーボはある位置まで回転し、出力軸をニュートラル位置から反時計回りにある程度保持する。パルスが 1.5 ミリ秒を上回る場合、逆のことが起こる。有効な位置にサーボを回転させるように命令するパルスの最小幅と最大幅は、各サーボの機能である。通常、パルスの最小幅は約 0.5 ms で、最大幅は 2.5 ms である。



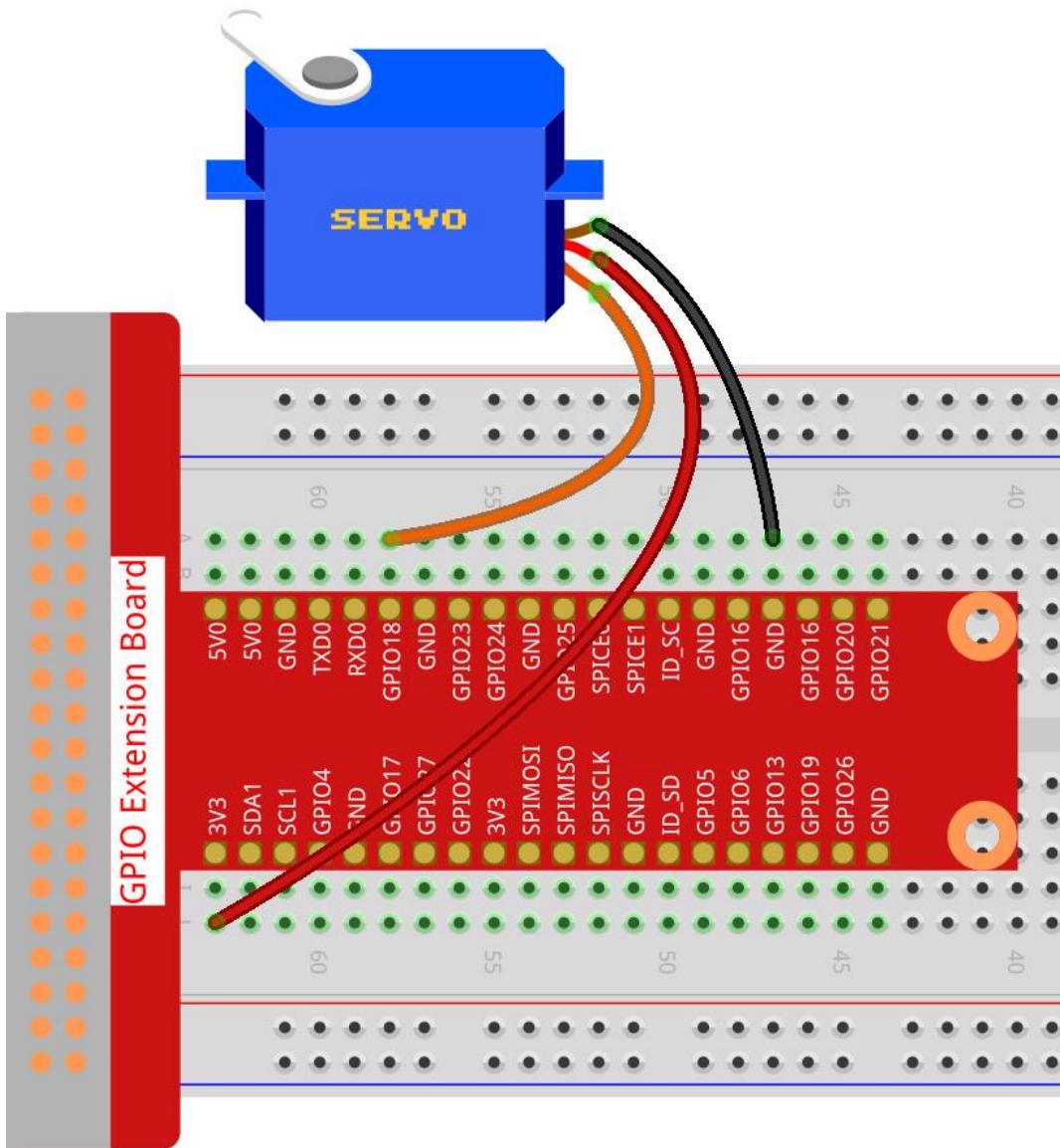
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.2
```

ステップ3: コードをコンパイルする。

```
gcc 1.3.2_Servo.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

プログラムが実行されると、サーボは 0 度から 180 度まで回転し、それから 180 度から 0 度まで循環的に回転する。

コード

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>

#define ServoPin    1      //define the servo to GPIO1
long Map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void setAngle(int pin, int angle){ //Create a function to control the angle of the servo.
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,Map(angle, 0, 180, 5, 25));
}

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ServoPin, 0, 200); //initialize PMW pin of servo
    while(1){
        for(i=0;i<181;i++){ // Let servo rotate from 0 to 180.
            setAngle(ServoPin,i);
            delay(2);
        }
        delay(1000);
        for(i=181;i>-1;i--){ // Let servo rotate from 180 to 0.
            setAngle(ServoPin,i);
            delay(2);
        }
        delay(1000);
    }
    return 0;
}
```

コードの説明

```
long Map(long value, long fromLow, long fromHigh, long toLow, long toHigh){  
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;  
}
```

次のコードで値をマップする Map () 関数を作成する

```
void setAngle(int pin, int angle){ //Create a function to control the angle of the servo.  
    if(angle < 0)  
        angle = 0;  
    if(angle > 180)  
        angle = 180;  
    softPwmWrite(pin,Map(angle, 0, 180, 5, 25));  
}
```

角度をサーボに書き込むために、関数 setAngle () を作成する

```
softPwmWrite(pin,Map(angle,0,180,5,25));
```

この関数は PWM のデューティサイクルを変更できる。

サーボを 0~180°に回転させるために、周期が 20ms のときにパルス幅を 0.5ms~2.5ms の範囲内で変更してください。関数 softPwmCreate () では、周期が $200 \times 100\text{us} = 20\text{ms}$ に設定されているため、0~180 を $5 \times 100\text{us} \sim 25 \times 100\text{us}$ にマッピングする必要がある。

この関数のプロトタイプを以下に示す。

```
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
```

Parameter pin: Raspberry Pi の GPIO ピンは PWM ピンとして設定できる。

Parameter initialValue: 初期パルス幅は、initialValue に 100us を掛けたものである。

Parameter pwmRange: PWM の周期は、pwmRange に 100us を掛けたものである。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 1.3.2_Servo.py
```

プログラムが実行されると、サーボは 0 度から 180 度まで回転し、それから 180 度から 0 度まで循環的に回転する。

コード

```
import RPi.GPIO as GPIO
import time

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500
ServoPin = 18

def map(value, inMin, inMax, outMin, outMax):
    return (outMax - outMin) * (value - inMin) / (inMax - inMin) + outMin

def setup():
    global p
    GPIO.setmode(GPIO.BCM)      # Numbers GPIOs by BCM
    GPIO.setup(ServoPin, GPIO.OUT) # Set ServoPin's mode is output
    GPIO.output(ServoPin, GPIO.LOW) # Set ServoPin to low
    p = GPIO.PWM(ServoPin, 50)   # set Freqeucy to 50Hz
    p.start(0)                  # Duty Cycle = 0

def setAngle(angle):      # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it

def loop():
    while True:
        for i in range(0, 181, 5):  #make servo rotate from 0 to 180 deg
            setAngle(i)      # Write to servo
            time.sleep(0.002)
        time.sleep(1)
        for i in range(180, -1, -5): #make servo rotate from 180 to 0 deg
            setAngle(i)
            time.sleep(0.001)
        time.sleep(1)

def destroy():
    p.stop()
    GPIO.cleanup()

if __name__ == '__main__':      #Program start from here
    setup()
    try:
```

```
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.

destroy()
```

コードの説明

```
p = GPIO.PWM(ServoPin, 50)      # set Frequency to 50Hz
p.start(0)                      # Duty Cycle = 0
```

servoPin を PWM ピンに設定し、次に周波数を 50hz に、周期を 20ms に設定する。

p.start (0) : PWM 関数を実行し、初期値を 0 に設定する。

```
def setAngle(angle):          # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it
```

関数 setAngle () を作成して、0~180 の範囲の角度をサーボに書き込む。

```
angle = max(0, min(180, angle))
```

このコードは角度を 0~180° の範囲に制限するために使用される。

min () 関数は入力値の最小値を返す。角度が 180 以下の場合、180 を返す。そうではない場合、角度を返す。

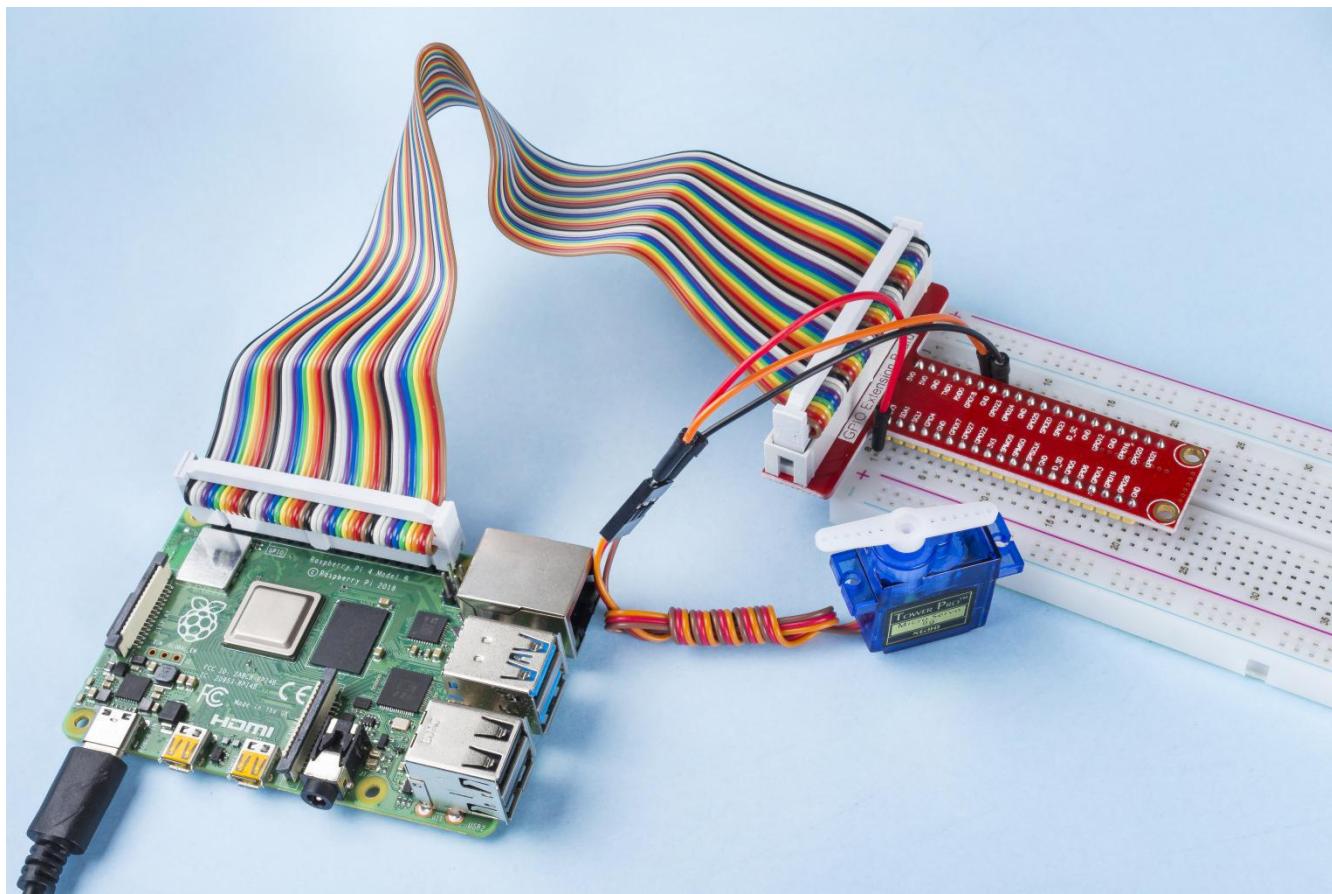
max () メソッドは、反復可能または最大の 2 つ以上のパラメーターで最大要素を返す。角度が 0 以上の場合は 0 を返し、そうでない場合は角度を返す。

```
pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
pwm = map(pulse_width, 0, 20000, 0, 100)
p.ChangeDutyCycle(pwm)
```

0~180° の範囲を サーボにレンダリングすると、サーボのパルス幅は 0.5ms (500us) ~ 2.5ms (2500us) に設定される。

PWM の周期は 20ms (20000us) であるため、PWM のデューティサイクルは $(500/20000) \%- (2500/20000) \%$ であり、0~180 の範囲は 2.5~12.5 にマッピングされる。

現象画像

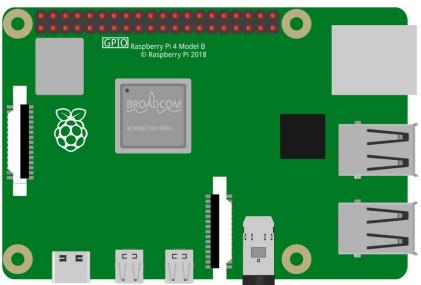
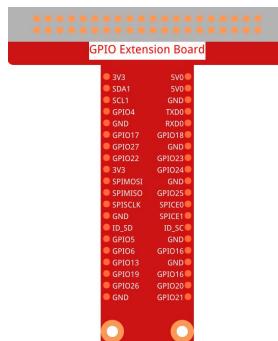
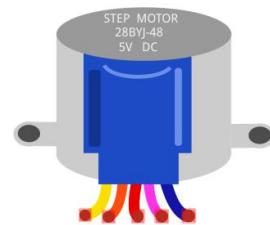
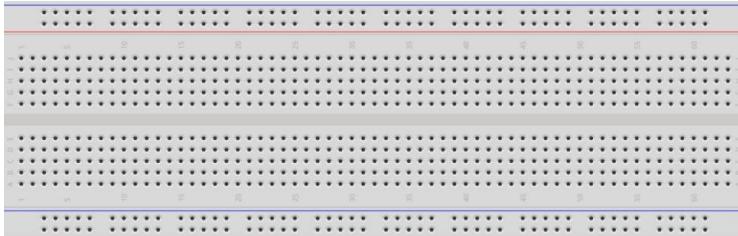
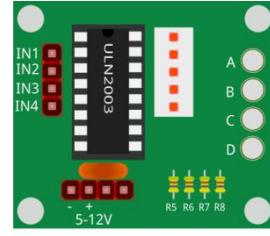


1.3.3 Stepper Motor

前書き

独自の設計により、ステッピングモーターはフィードバックメカニズムなしで高度な精度で制御できる。一連の磁石が取り付けられたステッパーのシャフトは、特定のシーケンスで正と負に帯電する一連の電磁コイルによって制御され、小さな「ステップ」で前後に正確に移動する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	ステッピングモーター*1
	 GPIO Extension Board Pinout: 3V3 SDA1 SDO SCL1 SDA2 GND GPI04 TXD0 GND RXD0 GPI017 GPIO18 GPI027 GPIO27 GPI022 GPIO22 3V3 GPIO24 SPI MOSI GND SPI MISO GPIO25 SPI SCLK GPIO26 GND SPI CS ID_SD ID SC GPI05 GND GPI06 GPIO16 GPI013 GND GPI019 GPIO10 GPI028 GPIO23 GND GPIO22	 STEP MOTOR 28BYJ-48 5V DC
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		ULN2003*1
		 ULN2003 IN1 IN2 IN3 IN4 A B C D + 5-12V R5 R6 R7 R8

原理

ステッピングモーター

ステッパーには、ユニポーラーとバイポーラーの2つのタイプがあり、使用しているタイプを知ることが非常に重要である。この実験では、ユニポーラステッパーを使用する。

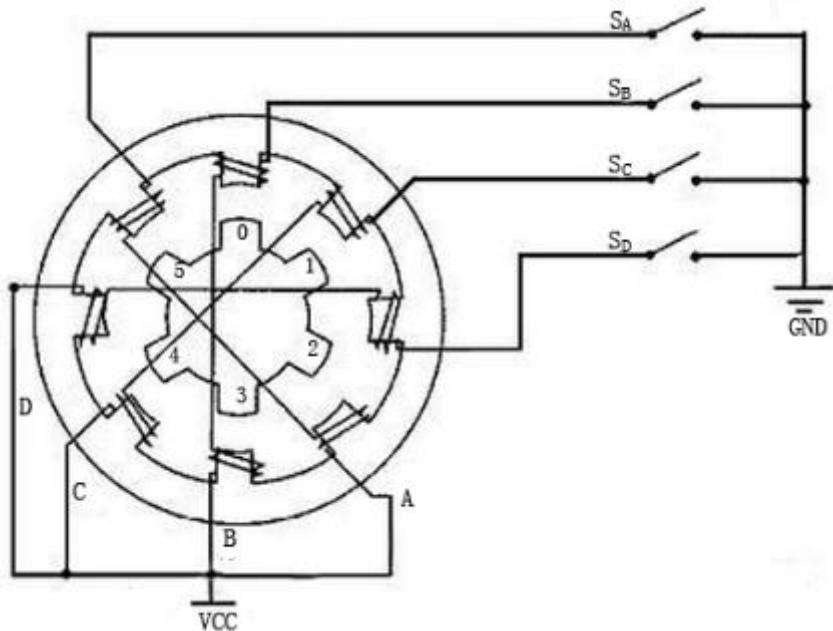
ステッピングモーターは単相DC電源を使用する4相モーターである。適切な時系列でモーターのすべての相巻線に通電する限り、一步一步に回転させることができる。4相リアクティブステッピングモーターの概略図：



図では、モーターの中央にローター-歯車状の永久磁石がある。ローターの周りの0~5は歯である。さらに外側には、8つの磁極があり、それぞれ反対側の2つの磁極がコイル巻線で接続されている。そのため、AからDまでの4つのペアを形成し、相と呼ばれる。スイッチSA、SB、SC、SDに接続するリード線が4本ある。したがって、回路では4つの相が並列になっており、1つの相の2つの磁極は直列になっている。

4相ステッピングモーターの動作原理は次の通りである：

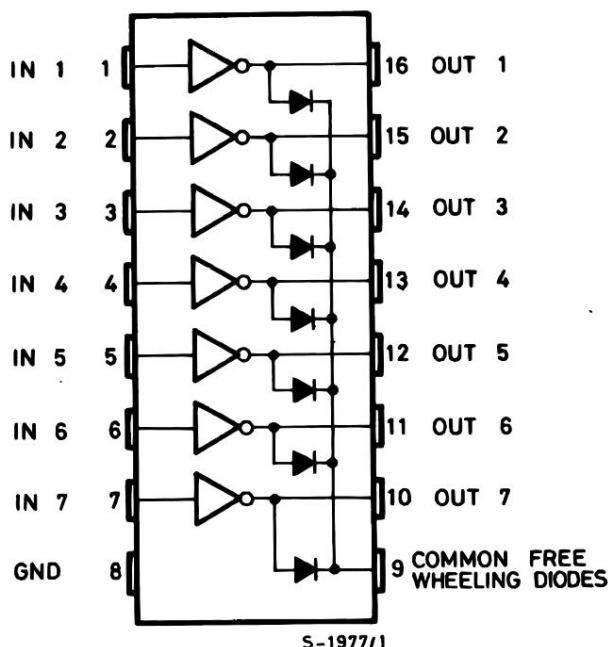
スイッチSBの電源がオンになると、スイッチSA、SC、およびSDの電源がオフになり、B相の磁極がローターの歯0と3と整列する。同時に、歯1と4は、C相とD相の極を持つ互い違いに配列した歯を生成する。歯2と5は、D相とA相の極を持つ互い違いに配列した歯を生成する。スイッチSCの電源がオン、スイッチSB、SA、およびSDの電源がオフの場合、ローターはC相巻線の磁場と歯1と4の間の磁場の下で回転する。次に、歯1と4がC相巻線の磁極と整列する。一方、歯0と3はA相とB相の極を持つ互い違いに配列した歯を生成し、歯2と5はA相とD相の極を持つ互い違いに配列した歯を生成する。同様の状況が続いている。A、B、C、D相に順番に通電すると、ローターはA、B、C、Dの順に回転する。



4相ステッピングモーターには、3つの動作モードがある：シングル4ステップ、ダブル4ステップ、および8ステップ。シングル4ステップとダブル4ステップのステップ角は同じであるが、シングル4ステップの駆動トルクは小さくなる。8ステップのステップ角は、シングル4ステップおよびダブル4ステップの半分である。したがって、8ステップの動作モードは、高い駆動トルクを維持し、制御精度を向上させることができる。

使用するステッピングモーターのステータには32個の磁極があるため、円には32ステップが必要である。ステッピングモーターの出力軸は減速装置セットに接続され、減速比は1/64である。ですから、最終出力シャフトは $32 * 64 = 2048$ ステップを必要とする円を回転させる。

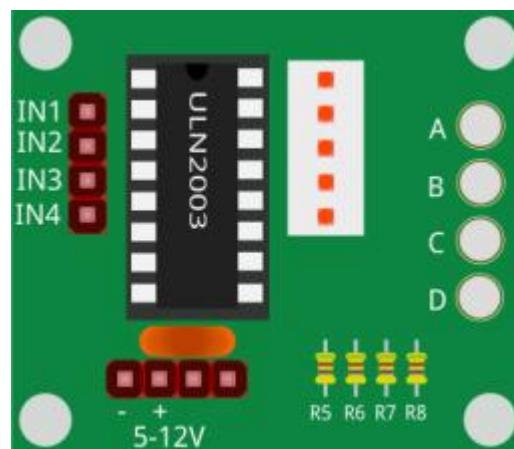
ULN2003



T回路にモーターを適用するには、ドライバーボードを使用しなければならない。ステッピングモータードライバー ULN2003 は7チャネルインバーター回路である。つまり、入力ピンが高レベルのとき、ULN2003 の出力ピンは低レベルになり、逆の場合も同様である。IN1 に高レベル、IN2、IN3、IN4 に低レベルを供給すると、出力端 OUT1 は低レベルになり、他のすべての出力端は高レベルになる。

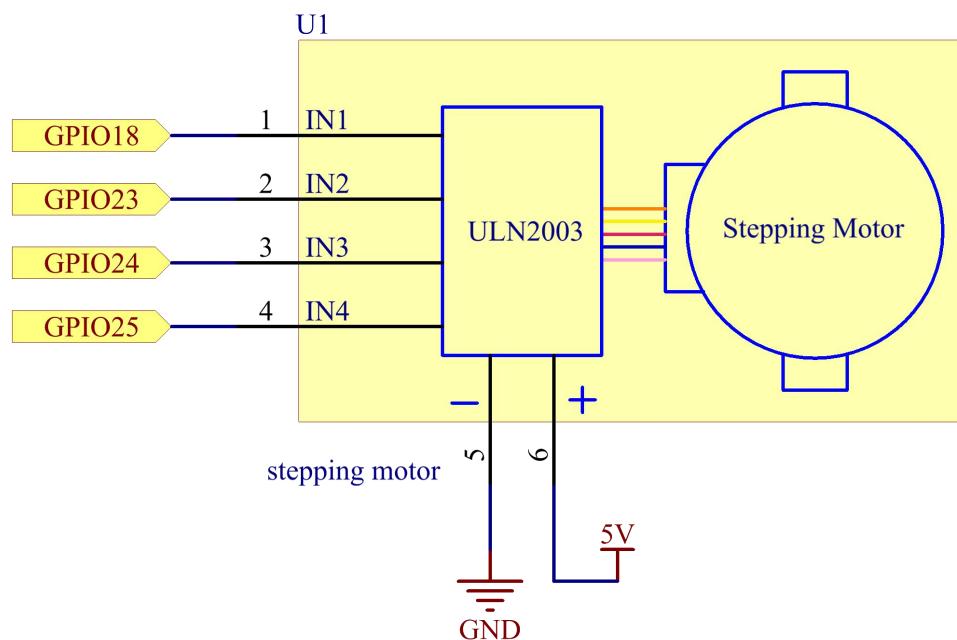
チップの内部構造を以下に示す。

ULN2003 チップと 4 つの LED で構成されるステッピングモータードライバーを以下に示す。ボードでは、IN1、IN2、IN3、IN4 が入力として機能し、4 つの LED、A、B、C、D は入力ピンの指示器である。さらに、OUT1、OUT2、OUT3、および OUT4 はそれぞれ、ステッピングモータードライバーの SA、SB、SC、および SD に接続されている。IN1 の値が高レベルに設定されると、A が点灯し、スイッチ SA の電源がオンになり、ステッピングモーターが 1 ステップで回転する。同様のケースが何度も繰り返される。したがって、ステッピングモーターに特定の時系列を与えるだけで、ステップごとに回転する。ステッピングモーターに特定の時系列を提供するために、ここで ULN2003 を使用する。



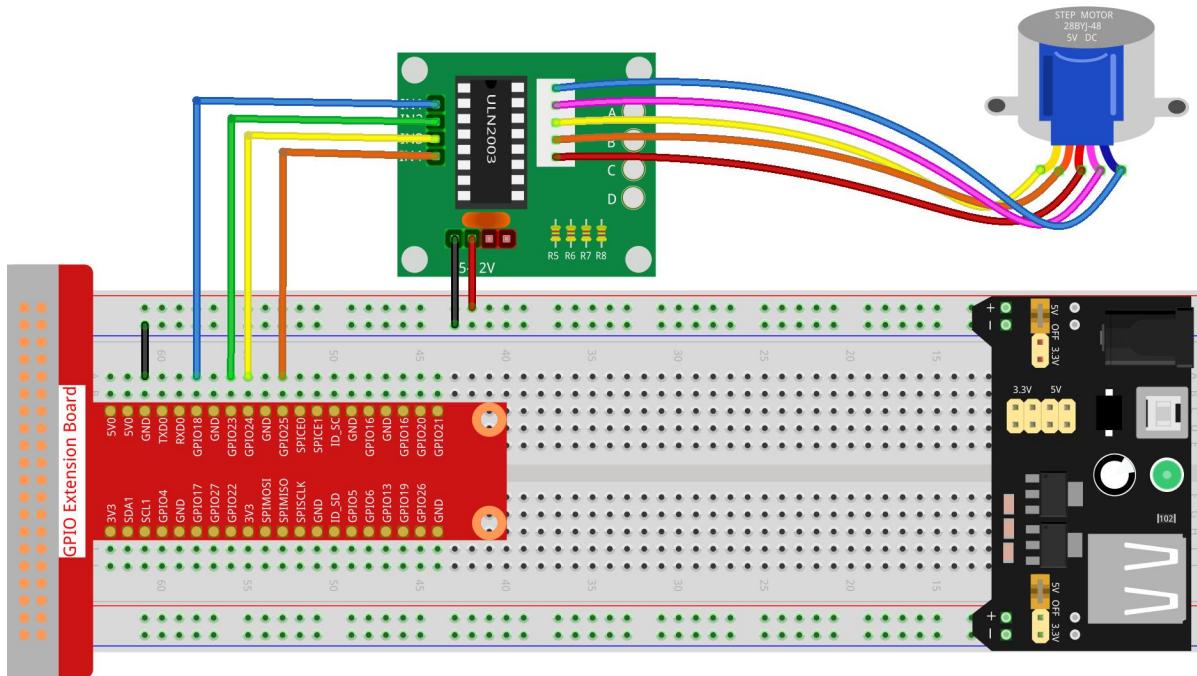
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.3/
```

ステップ3: コードをコンパイルする。

```
gcc 1.3.3_StepperMotor.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードが実行されると、入力「a」または「c」に応じてステッピングモーターが時計回りまたは反時計回りに回転する。

コード

```
#include <stdio.h>
#include <wiringPi.h>

const int motorPin[] = {1, 4, 5, 6};
int rolePerMinute = 15;
int stepsPerRevolution = 2048;
int stepSpeed = 0;

void rotary(char direction){
```

```

if(direction == 'c'){
    for(int j=0;j<4;j++){
        for(int i=0;i<4;i++)
            {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}
        delayMicroseconds(stepSpeed);
    }
}

else if(direction =='a'){
    for(int j=0;j<4;j++){
        for(int i=0;i<4;i++)
            {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}
        delayMicroseconds(stepSpeed);
    }
}

void loop()
{
    char direction = '0';
    while (1)
    {
        printf("select motor direction a=anticlockwise, c=clockwise: ");
        direction=getchar();
        if (direction == 'c')
        {
            printf("motor running clockwise\n");
            break;
        }
        else if (direction == 'a')
        {
            printf("motor running anti-clockwise\n");
            break;
        }
        else
        {
            printf("input error, please try again!\n");
        }
    }
    while(1)
    {
        rotary(direction);
    }
}

```

```

    }
}

void main(void)
{
    if (wiringPiSetup() == -1)
    {
        printf("setup wiringPi failed !");
        return;
    }
    for (int i = 0; i < 4; i++)
    {
        pinMode(motorPin[i], OUTPUT);
    }
    stepSpeed = (60000000 / rolePerMinute) / stepsPerRevolution;
    loop();
}

```

コードの説明

```

int rolePerMinute = 15;
int stepsPerRevolution = 2048;
int stepSpeed = 0;

```

rolePerMinute: 1分あたりの回転数。このキットで使用されるステッピングモーターの RPM は 0~17 である。

stepPerRevolution: 1 ターンのステップ数、およびこのキットで使用されるステッピングモーターには、1 回転あたり 2048 ステップが必要である。

stepSpeed: 各ステップに使用される時間、main () では、それらに値を割り当てる。

$\lceil (60000000/\text{rolePerMinute}) / \text{stepsPerRevolution} \rceil$ (60,000,000 us = 1 分)

```

void loop()
{
    char direction = '0';
    while (1)
    {
        printf("select motor direction a=anticlockwise, c=clockwise: ");
        direction=getchar();
        if (direction == 'c')

```

```
{  
    printf("motor running clockwise\n");  
    break;  
}  
else if (direction == 'a')  
{  
    printf("motor running anti-clockwise\n");  
    break;  
}  
else  
{  
    printf("input error, please try again!\n");  
}  
}  
while(1)  
{  
    rotary(direction);  
}  
}
```

loop () 関数は、2つの部分（2つの while (1) の間にある）に大まかに分けられている：最初の部分の目的は、key value を取得することである。「a」または「c」が取得されたら、ループを終了して入力を停止してください。

2番目の部分はロータリー（方向）を呼び出して、ステッピングモーターを動作させる。

```
void rotary(char direction){  
    if(direction == 'c'){  
        for(int j=0;j<4;j++){  
            for(int i=0;i<4;i++)  
                {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}  
            delayMicroseconds(stepSpeed);  
        }  
    }  
    else if(direction =='a'){  
        for(int j=0;j<4;j++){  
            for(int i=0;i<4;i++)  
                {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}  
            delayMicroseconds(stepSpeed);  
        }  
    }  
}
```

ステッピングモーターを時計回りに回転させるために、motorPin のレベルス状態要求は以下の通りである：

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH
Step2	HIGH	HIGH	LOW	LOW
Step3	LOW	HIGH	HIGH	LOW
Step4	LOW	LOW	HIGH	HIGH
Step5(Step1)	HIGH	LOW	LOW	HIGH

したがって、2層の for 文を使用して MotorPin の潜在的な書き込みを実装する。

Step1 では、j=0、i=0~4。

motorPin [0]は高レベル ($10011001 \& 00001000 = 1$) で書き込まれる

motorPin [1]は低レベル ($10011001 \& 00000100 = 0$) で書き込まれる

motorPin [2]は低レベル ($10011001 \& 00000010 = 0$) で書き込まれる

motorPin [3]は高レベル ($10011001 \& 00000001 = 1$) で書き込まれる。

Step2 では、j=1、i= 0~4。

motorPin [0]は高レベル ($01001100 \& 00001000 = 1$) で書き込まれる

motorPin [1]は低レベル ($01001100 \& 00000100 = 1$) などで書き込まれる。

また、ステッピングモーターを反時計回りに回転させるために、motorPin のレベルステータスを次の表に示す。

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH
Step2	LOW	LOW	HIGH	HIGH
Step3	LOW	HIGH	HIGH	LOW
Step4	HIGH	HIGH	LOW	LOW
Step5(1)	HIGH	LOW	LOW	HIGH

Step1 では、j=0、i=0~4。

motorPin [0]は高レベル ($10011001 \& 10000000 = 1$) で書き込まれる motorPin [1]は低レベル ($10011001 \& 01000000 = 0$) で書き込まれる。

ステップ2 では、j=1、i=0~4。

motorPin [0]は高レベル ($00110010 \& 10000000 = 0$) で書き込まれる motorPin [1]は低レベル ($00110010 \& 01000000 = 0$) で書き込まれる。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 1.3.3_StepperMotor.py
```

コードが実行されると、入力「a」または「c」に応じて、ステッピングモーターが時計回りまたは反時計回りに回転する。

コード

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

motorPin = (18,23,24,25)
rolePerMinute = 15
stepsPerRevolution = 2048
stepSpeed = (60/rolePerMinute)/stepsPerRevolution

def setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    for i in motorPin:
        GPIO.setup(i, GPIO.OUT)

def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i], 0x99 >> j & (0x08 >> i))
                sleep(stepSpeed)

    elif(direction == 'a'):
```

```

for j in range(4):
    for i in range(4):
        GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
        sleep(stepSpeed)

def loop():
    while True:
        direction = input('select motor direction a=anticlockwise, c=clockwise: ')
        if(direction == 'c'):
            print('motor running clockwise\n')
            break
        elif(direction == 'a'):
            print('motor running anti-clockwise\n')
            break
        else:
            print('input error, please try again!')
    while True:
        rotary(direction)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

コードの説明

```

rolePerMinute =15
stepsPerRevolution = 2048
stepSpeed = (60/rolePerMinute)/stepsPerRevolution

```

rolePerMinute: 1分あたりの回転数。このキットで使用されるステッピングモーターのRPMは0~17である。

stepPerRevolution: 1ターンのステップ数、およびこのキットで使用されるステッピングモーターには、1回転あたり2048ステップが必要である。

stepSpeed: 各ステップに使用される時間。それらに値を割り当てる: 「(60 / rolePerMinute) / stepsPerRevolution」 (60s = 1分)。

```
def loop():
    while True:
        direction = input('select motor direction a=anticlockwise, c=clockwise: ')
        if(direction == 'c'):
            print('motor running clockwise\n')
            break
        elif(direction == 'a'):
            print('motor running anti-clockwise\n')
            break
        else:
            print('input error, please try again!')
    while True:
        rotary(direction)
```

loop () 関数は、大きく分けて2つの部分に分かれている (2つのwhile (1) にある) :

最初の部分の目的は、key value を取得することである。「a」または「c」が取得されたら、ループを終了して入力を停止してください。

2番目の部分は、ロータリー (方向) を呼び出し、ステッピングモーターを動作させる。

```
def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99>>j & (0x08>>i))
                sleep(stepSpeed)

    elif(direction == 'a'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
                sleep(stepSpeed)
```

ステッピングモーターを時計回りに回転させるために、motorPin のレベルステータスを次の表に示す。

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH

Step2	HIGH	HIGH	LOW	LOW
Step3	LOW	HIGH	HIGH	LOW
Step4	LOW	LOW	HIGH	HIGH
Step5(1)	HIGH	LOW	LOW	HIGH

したがって、2層のfor文を使用して MotorPin の潜在的な書き込みを実装する。

Step1 では、j=0、i=0~4。

motorPin [0]は高レベル (10011001&00001000 = 1) で書き込まれる motorPin [1]は低レベル (10011001&00000100=0) で書き込まれる motorPin [2]は低レベル (10011001 & 00000010 = 0) で書き込まれる motorPin [3] は高レベル (10011001 & 00000001=1) で書き込まれる。

Step2 では、j=1、i= 0~4。

motorPin [0]は高レベル (01001100&00001000 =1) で書き込まれ、motorPin [1]は低レベル (01001100&00000100 =1) で書き込まれる。

また、ステッピングモーターを反時計回りに回転させるために、motorPin のレベルステータスを次の表に示す。

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH
Step2	LOW	LOW	HIGH	HIGH
Step3	LOW	HIGH	HIGH	LOW
Step4	HIGH	HIGH	LOW	LOW
Step5(1)	HIGH	LOW	LOW	HIGH

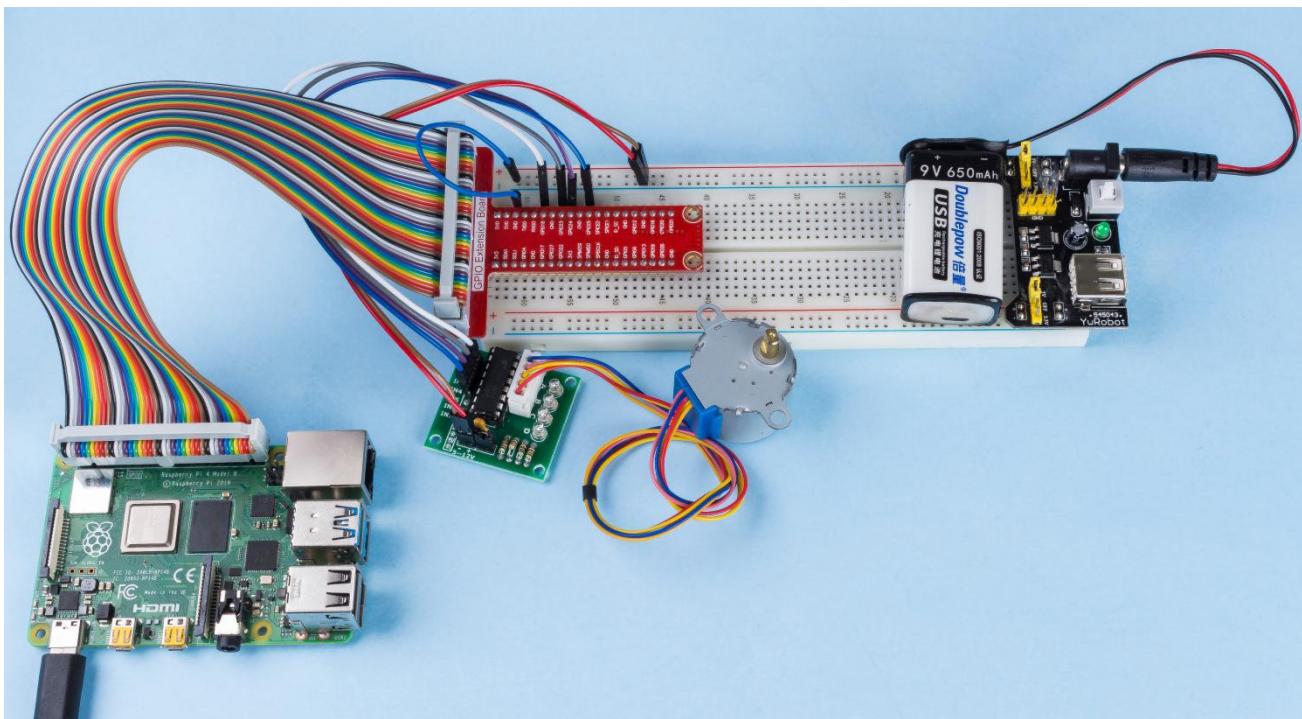
Step1 では、j=0、i=0~4。

motorPin [0]は高レベル (10011001&10000000 = 1) で書き込まれる motorPin [1]は低レベル (10011001&01000000 = 0) で書き込まれる。

ステップ2 では、j=1、i=0~4。

motorPin [0]は高レベル (00110010&10000000 = 0) で書き込まれる motorPin [1]は低レベル (00110010&01000000 = 0) で書き込まれる。

現象画像

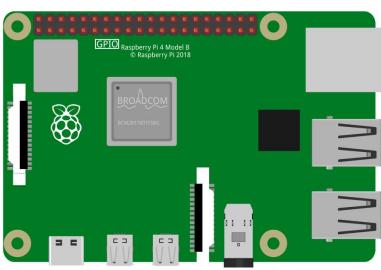
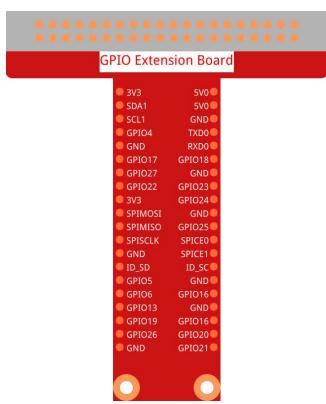
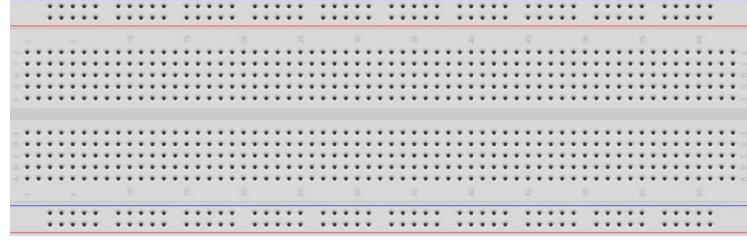
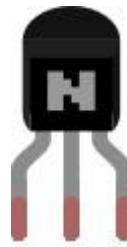
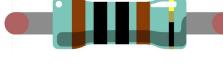


1.3.4 Relay

前書き

このレッスンでは、リレーの使用方法を学習する。これは、自動制御システムで一般的に使用される部品の1つである。電圧、電流、温度、圧力などが所定の値に到達、超過、または低下すると、リレーは回路を接続または中断して、機器を制御したり保護したりする。

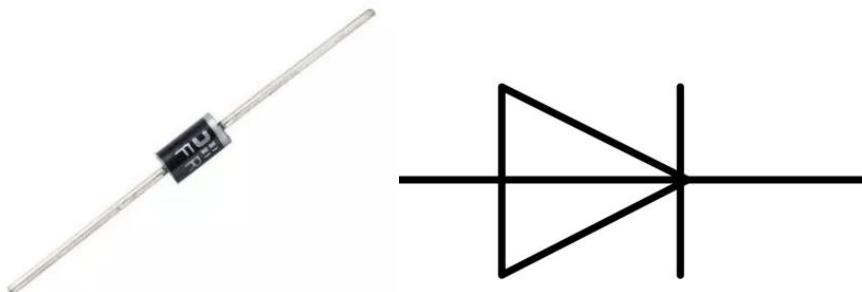
部品

Raspberry Pi 本体*1	T 拡張ボード*1	リレー*1																																								
	 GPIO Extension Board <table border="1"><tr><td>● 3V3</td><td>5V0</td></tr><tr><td>● SDA1</td><td>5V0</td></tr><tr><td>● SCL1</td><td>GND</td></tr><tr><td>● GPIO4</td><td>TXD0</td></tr><tr><td>● GND</td><td>RXD0</td></tr><tr><td>● GPIO17</td><td>GPIO18</td></tr><tr><td>● GPIO27</td><td>GND</td></tr><tr><td>● GPIO22</td><td>GPIO23</td></tr><tr><td>● 3V3</td><td>GPIO24</td></tr><tr><td>● SPIMOSI</td><td>GND</td></tr><tr><td>● SPIMISO</td><td>GPIO25</td></tr><tr><td>● SPICLK</td><td>SPI00</td></tr><tr><td>● GND</td><td>SPIE1</td></tr><tr><td>● ID_SD</td><td>ID_SC</td></tr><tr><td>● GPIO5</td><td>GND</td></tr><tr><td>● GPIO6</td><td>GPIO16</td></tr><tr><td>● GPIO13</td><td>GND</td></tr><tr><td>● GPIO19</td><td>GPIO16</td></tr><tr><td>● GPIO26</td><td>GPIO20</td></tr><tr><td>● GND</td><td>GPIO21</td></tr></table>	● 3V3	5V0	● SDA1	5V0	● SCL1	GND	● GPIO4	TXD0	● GND	RXD0	● GPIO17	GPIO18	● GPIO27	GND	● GPIO22	GPIO23	● 3V3	GPIO24	● SPIMOSI	GND	● SPIMISO	GPIO25	● SPICLK	SPI00	● GND	SPIE1	● ID_SD	ID_SC	● GPIO5	GND	● GPIO6	GPIO16	● GPIO13	GND	● GPIO19	GPIO16	● GPIO26	GPIO20	● GND	GPIO21	 SONGLE 3A 250VAC 30VDC-SH
● 3V3	5V0																																									
● SDA1	5V0																																									
● SCL1	GND																																									
● GPIO4	TXD0																																									
● GND	RXD0																																									
● GPIO17	GPIO18																																									
● GPIO27	GND																																									
● GPIO22	GPIO23																																									
● 3V3	GPIO24																																									
● SPIMOSI	GND																																									
● SPIMISO	GPIO25																																									
● SPICLK	SPI00																																									
● GND	SPIE1																																									
● ID_SD	ID_SC																																									
● GPIO5	GND																																									
● GPIO6	GPIO16																																									
● GPIO13	GND																																									
● GPIO19	GPIO16																																									
● GPIO26	GPIO20																																									
● GND	GPIO21																																									
40 ピンケーブル*1		1N4007 ダイオード*1																																								
																																										
ブレッドボード*1	1 * LED	S8050 NPN トランジスタ*1																																								
																																										
	何本のジャンパー線																																									
	抵抗器 (220Ω) *1																																									
	抵抗器 (1kΩ) *1																																									

原理

ダイオード

ダイオードは一方向に電流が流れる電子機器の2端子部品である。電流の流れる方向に低い抵抗を提供し、反対方向に高い抵抗を提供する。ダイオードは主に、通常は分極している回路の起電力による部品の損傷を防ぐために使用される。



ダイオードの2つの端子は極性があり、陽極と呼ばれる正の端と陰極と呼ばれる負の端がある。陰極は通常、銀でできているか、またあカラーバンドを持っている。電流の方向を制御することは、ダイオードの重要な機能の1つである。ダイオードの電流は陽極から陰極に流れ。ダイオードの動作は、逆止弁の動作に似ている。ダイオードの最も重要な特性の1つは、非線形電流電圧である。より高い電圧が陽極に接続されている場合、電流は陽極から陰極に流れ、プロセスは順方向バイアスと呼ばれる。ただし、より高い電圧が陰極に接続されている場合、ダイオードは電気を通さず、プロセスは逆方向バイアスと呼ばれる。

リレー

ご存知のように、リレーは、入力信号に応じて2つ以上のポイントまたはデバイス間の接続を提供するために使用されるデバイスである。つまり、デバイスはACとDCの両方で動作する可能性があるため、リレーはコントローラーとデバイスの間を分離する。しかしながら、DC上で動作するマイクロコントローラーから信号を受信するため、ギャップを埋めるためのリレーが必要である。小さな電気信号で大量の電流または電圧を制御する必要がある場合には、リレーは非常に有用である。

すべてのリレーには5つのパートがある：

1. **電磁石**-ワイヤーのコイルで巻かれた鉄心で構成されている。通電させると、磁気になります。したがって、電磁石と呼ばれる。.
2. **電機子**-可動磁気トリップは電機子と呼ばれる。それらに電流が流れると、コイルが通電されて磁場が生成され、常開 (N/O) または常閉 (N/C) ポイントを作成したり切断したりするために使用される。また、電機子は直流 (DC) と交流 (AC) で移動できる。
3. **スプリング**-電磁石のコイルに電流が流れない場合、スプリングは電機子を引き離し、回路を完成できない。
4. **電気接点のセット**-2つの接点がある：

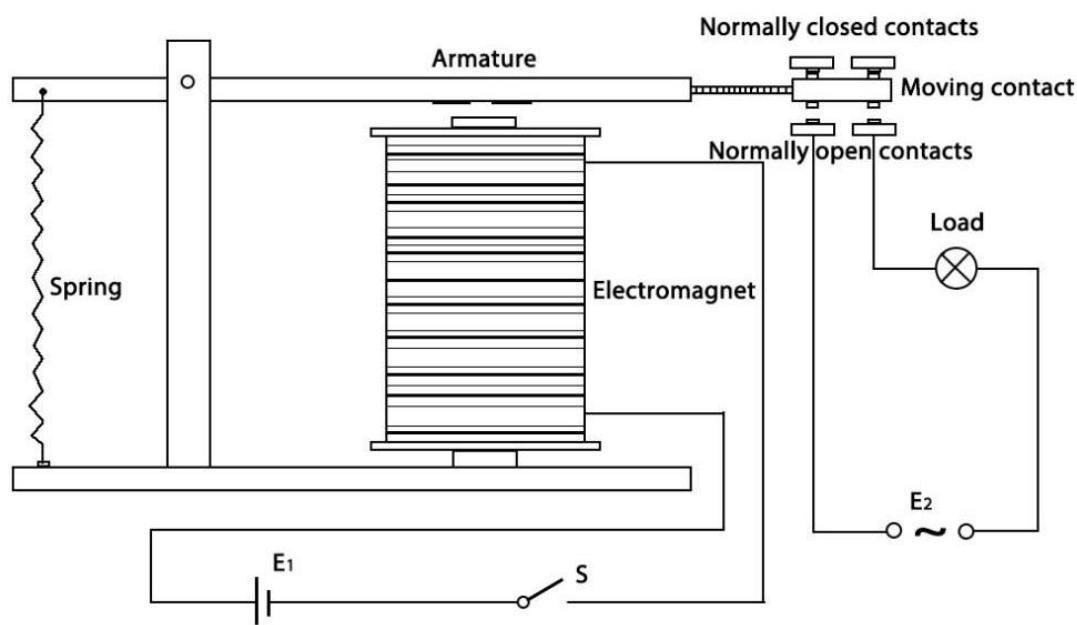
・常開-リレーが作動している時に接続し、リレーが作動していないときに遮断する。

・常閉-リレーが作動している時に接続しなく、リレーが作動していないときに接続する。

5.成形フレーム-保護のため、リレーはプラスチックで覆われている。

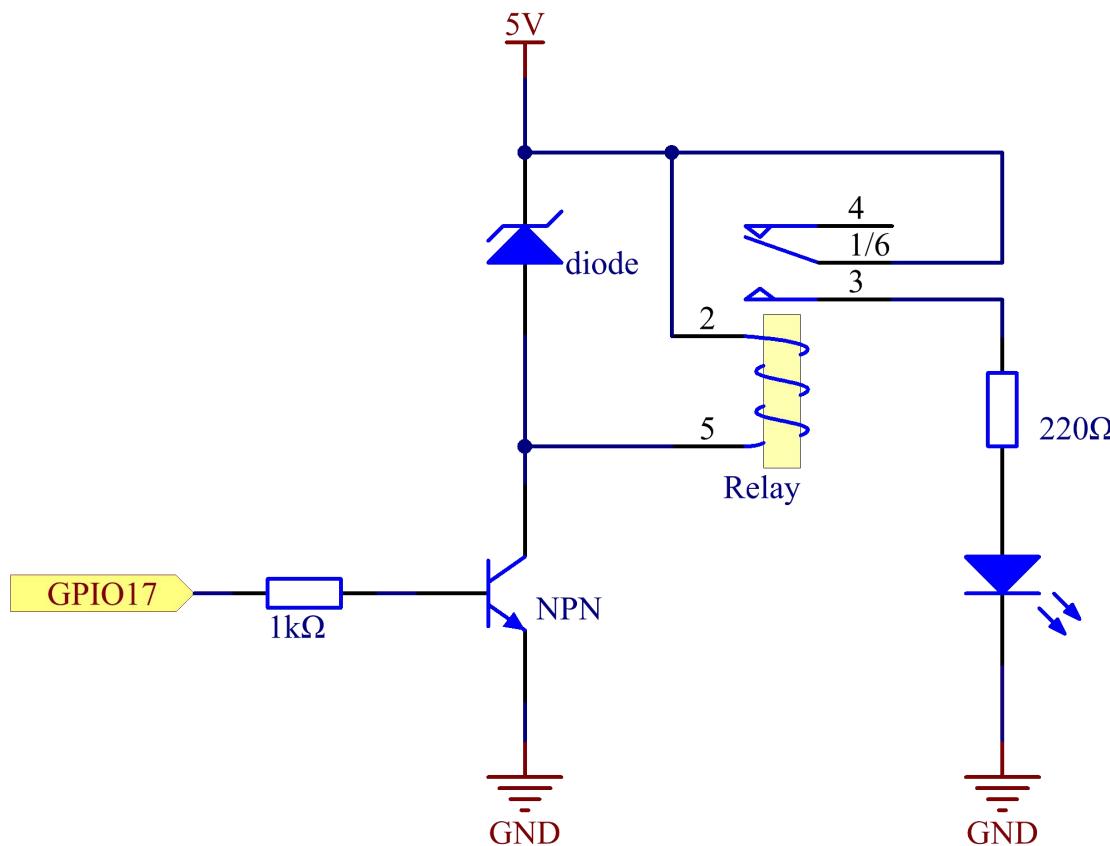
リレーの動作

リレーの動作原理は非常に簡単である。リレーに電力が供給されると、制御コイルに電流が流れ始める。その結果、電磁石が通電を開始する。次に、電機子がコイルに引き付けられ、可動接点と一緒に引き下げられ、常開接点に接続される。したがって、負荷がかけられた回路が通電される。次に、可動接点がスプリングの力で常閉接点に引き上げられるため、回路を遮断することも同様のケースになる。このようにして、リレーのオンとオフの切り替えにより、負荷回路の状態を制御できる。



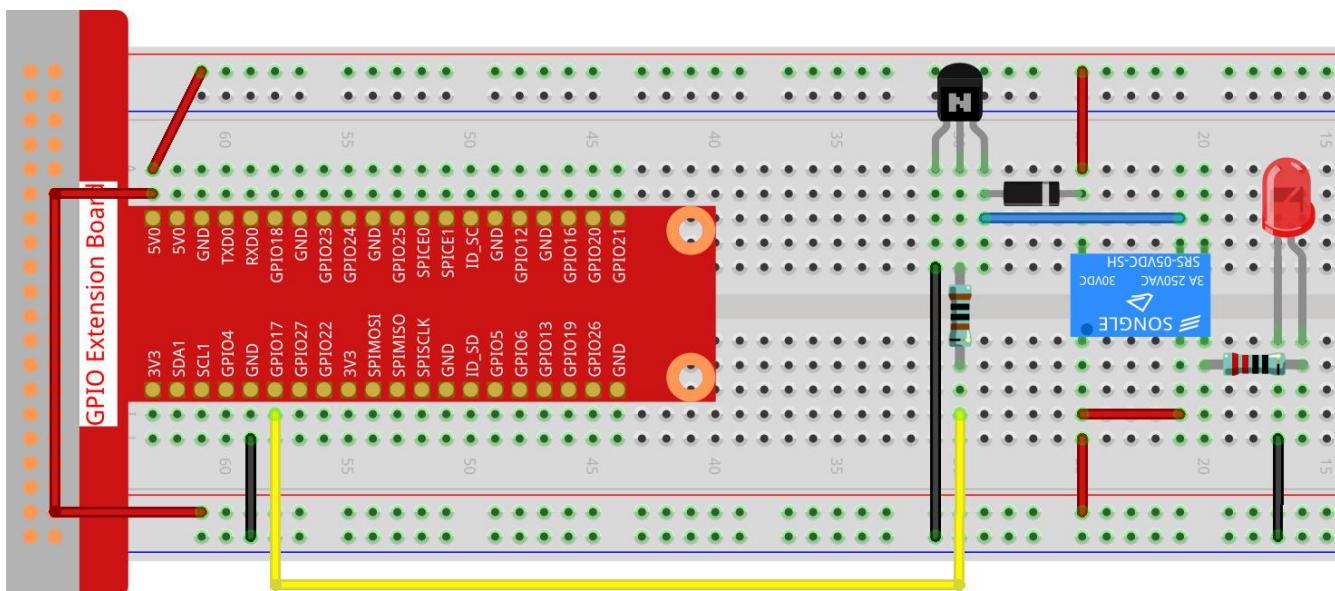
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ 1：回路を作る。



➤ C言語ユーザー向け

ステップ2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.4
```

ステップ3: コードをコンパイルする。

```
gcc 1.3.4_Relay.c -lwiringPi
```

ステップ4: EXEファイルを実行する。

```
sudo ./a.out
```

コードの実行後、LEDが点灯する。さらに、通常は閉じている接点を切斷し、通常は開いている接点を閉じることによって発生するカチカチという声が聞こえる。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#define RelayPin 0

int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(RelayPin, OUTPUT); //set GPIO17(GPIO0) output
    while(1){
        // Tick
        printf("Relay Open.....\n");
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
        printf(".....Relay Close\n");
        digitalWrite(RelayPin, HIGH);
        delay(1000);
    }

    return 0;
}
```

コードの説明

```
digitalWrite(RelayPin, LOW);
```

I/O ポートを低レベル (0V) に設定すると、トランジスタに通電されず、コイルに電力が供給されない。電磁力がないため、リレーが開き、LED は点灯しない。

```
digitalWrite(RelayPin, HIGH);
```

I/O ポートを高レベル (5V) に設定して、トランジスタに通電する。リレーのコイルに電力を供給し、電磁力が発生すると、リレーが閉じて LED が点灯する。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: 実行する。

```
sudo python3 1.3.4_Relay.py
```

コードの実行中、LED が点灯する。さらに、通常は閉じている接点を切断し、通常は開いている接点を閉じることによって発生するカチカチという声が聞こえる。

コード

```
#!/usr/bin/env python3

import RPi.GPIO as GPIO
import time

# Set GPIO17 as control pin
relayPin = 17

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set relayPin's mode to output,
    # and initial level to High(3.3v)
    GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():
    while True:
```

```
print ('Relay open...')

# Tick
GPIO.output(relayPin, GPIO.LOW)
time.sleep(1)
print ('...Relay close')
# Tock
GPIO.output(relayPin, GPIO.HIGH)
time.sleep(1)

# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(relayPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the child program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
GPIO.output(relayPin, GPIO.LOW)
```

トランジスタのピンを低レベルに設定して、リレーを開く。LED は点灯しない。

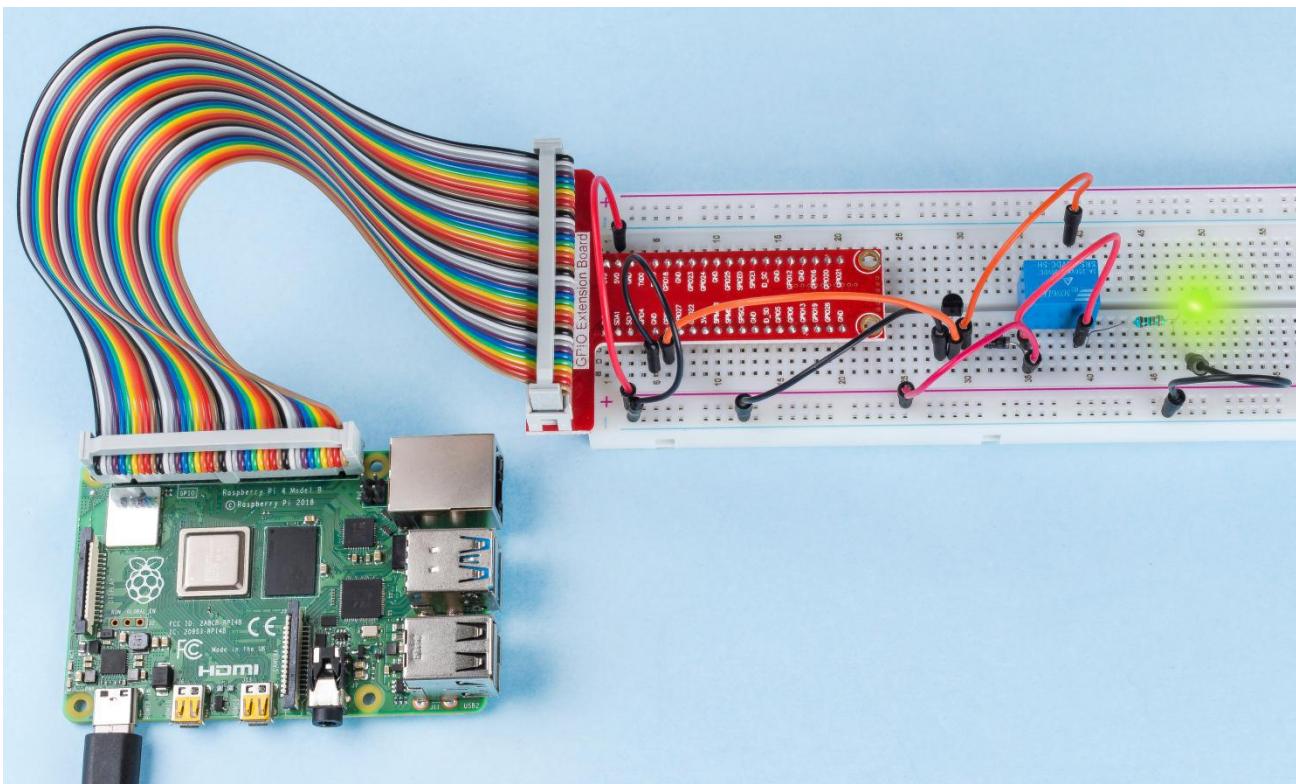
```
time.sleep(1)
```

1秒待つ。

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

トランジスタのピンを低レベルに設定してリレーを作動させ、LED が点灯する。

現象画像



2 入力

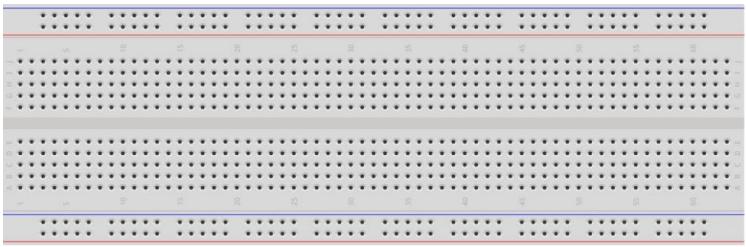
2.1 コントローラー

2.1.1 Button

前書き

このレッスンでは、LED をボタンでオンまたはオフにする方法を学習する。

部品

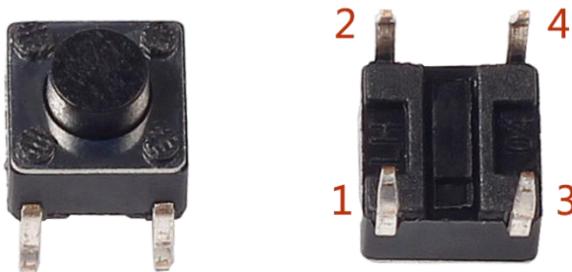
Raspberry Pi 本体*1	T 拡張ボード*1	LED*1  抵抗器 (10KΩ) *1 
40 ピンケーブル*1 		抵抗器 (220Ω) *1  何本のジャンパー線
ブレッドボード*1 		ボタン*1 

原理

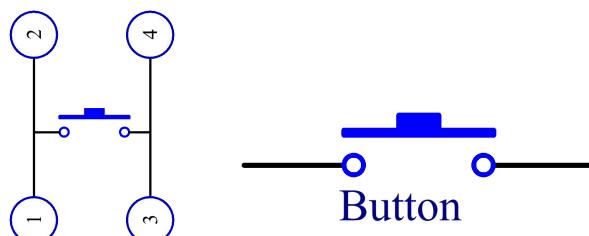
ボタン

ボタンは電子デバイスを制御するために使用される一般的な部品である。通常、回路を接続または遮断するためのスイッチとして使用される。ボタンにはさまざまなサイズと形状があるが、ここで使用するものは、次の図に示すように 6mm のミニボタンである。

左側の 2 つのピンが接続されており、右側の方は左側と同じである。以下を参照してください：



以下に示す記号は、通常、回路内のボタンを表すために使用される。



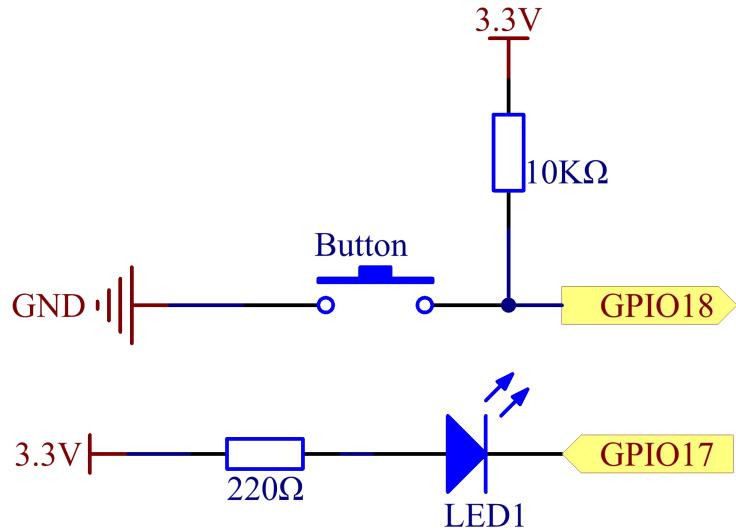
ボタンを押すと、4 つのピンが接続され、回路が閉じる。

回路図

Raspberry Pi の入力として常開ボタンを使用し、接続は下の概略図に示されている。ボタンを押すと、GPIO18 は低レベル (0V) に変わる。プログラミングによって GPIO18 の状態を検出できる。つまり、GPIO18 が低レベルになった場合、ボタンが押されたことを意味する。ボタンが押されたときに応答するコードを実行すると、LED が点灯する。

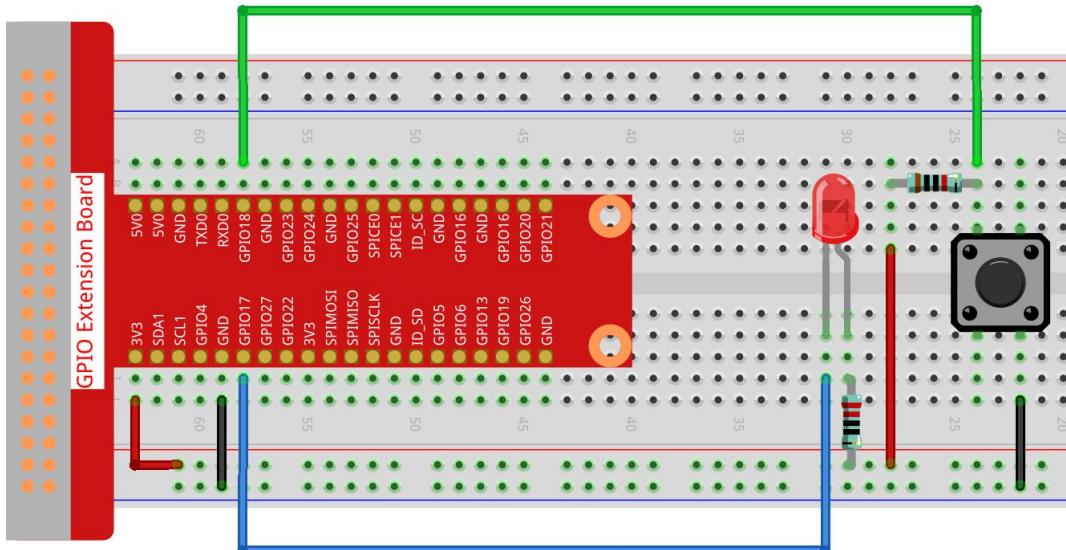
ご注意： LED の長いピンは陽極で、短い方は陰極である。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.1/
```

ご注意: cd を使用して、この実験のコードのパスにディレクトリを変更する。

ステップ3: コードをコンパイルする。

```
gcc 2.1.1_Button.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、ボタンを押すと、LED が点灯する。それ以外の場合は消灯する。

コード

```
#include <wiringPi.h>
#include <stdio.h>

#define LedPin      0
#define ButtonPin   1

int main(void){
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(LedPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    digitalWrite(LedPin, HIGH);

    while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 0){
            // Led on
            digitalWrite(LedPin, LOW);
            // printf("...LED on\n");
        }
        else{
            // Led off
            digitalWrite(LedPin, HIGH);
            // printf("LED off...\n");
        }
    }
    return 0;
}
```

コードの説明

```
#define LedPin      0
```

T_Extension ボードの GPIO17 ピンは、wiringPi の GPIO0 と同じである。

```
#define ButtonPin   1
```

ButtonPin は GPIO1 に接続されている。

```
pinMode(LedPin, OUTPUT);
```

LedPin を出力として設定し、値を割り当てる。

```
pinMode(ButtonPin, INPUT);
```

ButtonPin を入力として設定し、ButtonPin の値を読み取る。

```
while(1){  
    // Indicate that button has pressed down  
    if(digitalRead(ButtonPin) == 0){  
        // Led on  
        digitalWrite(LedPin, LOW);  
        // printf("...LED on\n");  
    }  
    else{  
        // Led off  
        digitalWrite(LedPin, HIGH);  
        // printf("LED off...\n");  
    }  
}
```

if (digitalRead (ButtonPin) == 0): ボタンが押されたかどうかを確認してください。ボタンを押して LED を点灯させると、digitalWrite (LedPin, LOW) を実行してください。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: コードを実行する。

```
sudo python3 2.1.1_Button.py
```

それから、ボタンを押すと、LED が点灯する。もう一度ボタンを押すと、LED が消灯する。同時に、LED の状態が画面にプリントされる。

コード

```
import RPi.GPIO as GPIO
import time

LedPin = 17 # Set GPIO17 as LED pin
BtnPin = 18 # Set GPIO18 as button pin

# Set Led status to True(OFF)
Led_status = True

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,
    # and initial level to high (3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    # Set BtnPin's mode to input,
    # and pull up to high (3.3V)
    GPIO.setup(BtnPin, GPIO.IN)

# Define a callback function for button callback
def swLed(ev=None):
    global Led_status
    # Switch led status(on-->off; off-->on)
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
    if Led_status:
        print ('LED OFF...')
    else:
        print ('...LED ON')

# Define a main function for main process
def main():
    # Set up a falling detect on BtnPin,
    # and callback function to swLed
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
    while True:
        # Don't do anything.
        time.sleep(1)

# Define a destroy function for clean up everything after
```

```
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

コードの説明

LedPin = 17

GPIO17 を LED ピンとして設定する

BtnPin = 18

GPIO18 をボタンピンとして設定する

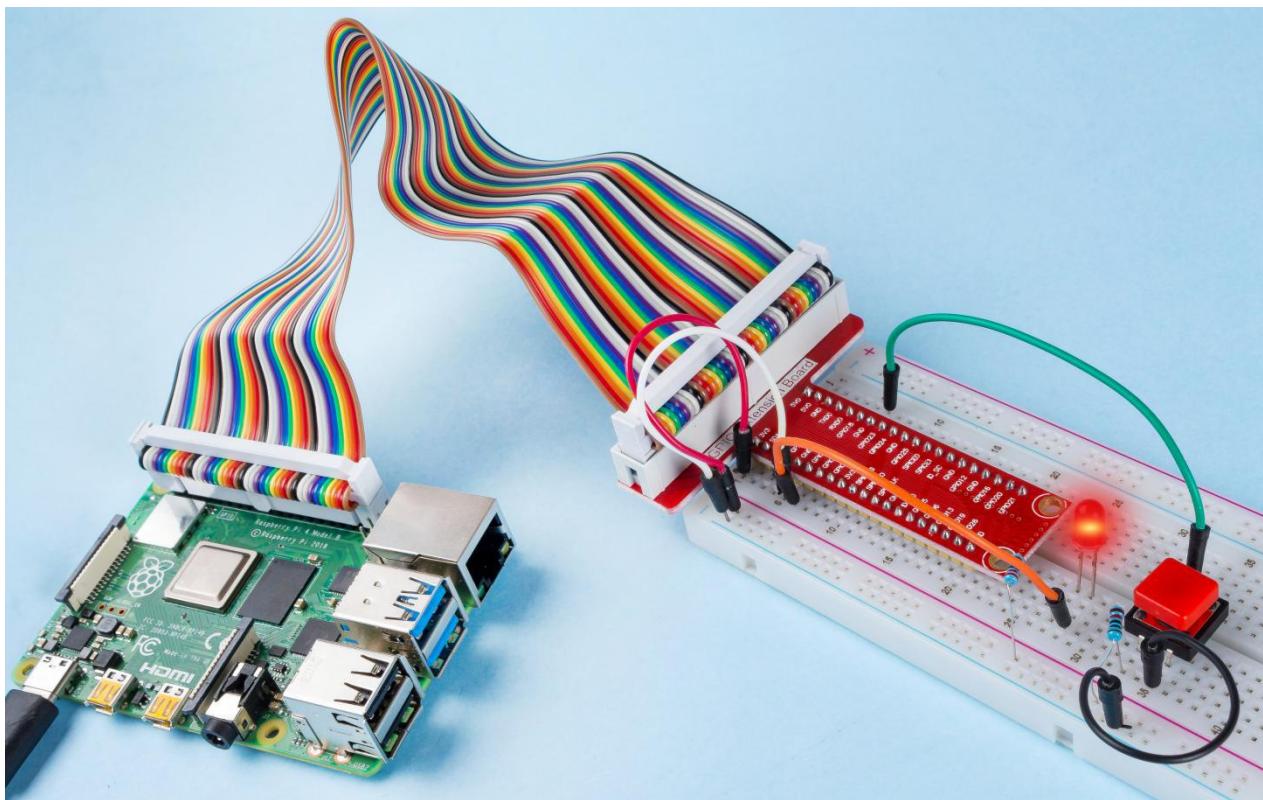
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)

BtnPin で立ち下がり検出を設定してから、BtnPin の値が高レベルから低レベルに変わると、ボタンが押されたことを意味する。次のステップは、関数 swled を呼び出す。

```
def swLed(ev=None):
    global Led_status
    # Switch led status(on-->off; off-->on)
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
```

ボタンコールバックとしてコールバック関数を定義する。ボタンが初めて押され、not Led_status 条件が false の場合、GPIO.output () 関数を呼び出して LED を点灯させる。ボタンをもう一度押すと、LED の状態が false から true に変換され、LED が消灯する。

現象画像

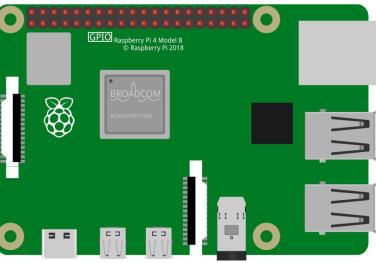
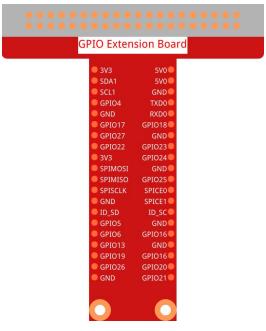
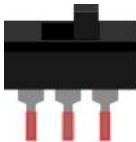
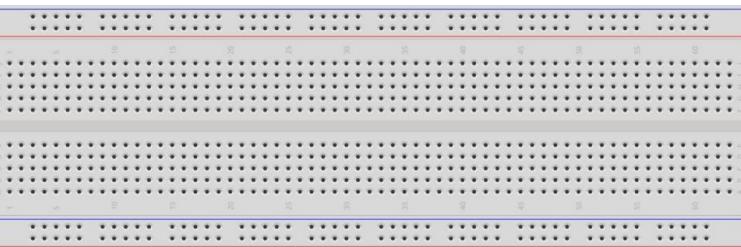


2.1.2 Slide Switch

前書き

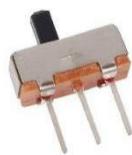
このレッスンでは、スライドスイッチの使用方法を学習する。通常、スライドスイッチは電源スイッチとしてPCBにはんだ付けされるが、ここではブレッドボードに挿入する必要があるため、締め付けられない場合がある。そして、その機能を示すためにブレッドボードで使用する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	スライドスイッチ*1
	 GPIO Extension Board ■ 3V3 S9000 ■ GND S9001 ■ SCL1 GND900 ■ GPIO1 TXD0900 ■ GND RXD0900 ■ GPIO17 GND1800 ■ GND GND900 ■ GPIO22 GPIO23900 ■ 3V3 GPIO24900 ■ SPI MOSI GND900 ■ SPI CS GND900 ■ GND SPI CS1900 ■ GND ID_SD ID_SD900 ■ GPIO5 GND1600 ■ GPIO6 GPIO16900 ■ GPIO13 GND900 ■ GPIO19 GPIO20900 ■ GND GPIO21900	
40 ピンケーブル*1	LED*2	104 コンデンサ*1
		何本のジャンパー線
ブレッドボード*1		
		

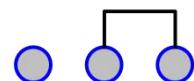
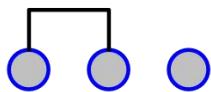
原理

スライドスイッチ

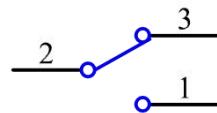


スライドスイッチは名前が示すように、スイッチバーをスライドさせて回路を接続または遮断し、さらに回路を切り替えるものである。汎用タイプは、SPDT、SPTT、DPDT、DPTTなどである。スライドスイッチは低電圧回路で一般的に使用されている。融通性と安定性の特徴を備えており、電気機器や電気玩具に広く適用されている。

仕組み：中央のピンを固定ピンとして設定する。スライドを左に引くと、左の2つのピンが接続され、右に引くと、右側の2つのピンが接続される。したがって、回路を接続または遮断するスイッチとして機能する。以下の図を参照してください：



The circuit symbol of the slide switch is shown as below. The pin2 in the figure refers to the middle pin.



コンデンサ

コンデンサーは、小さな充電式バッテリーのように、エネルギーを電荷の形で蓄えたり、プレート間に電位差（静的電圧）を生成する能力を持つ部品である。

静電容量の標準単位

Microfarad (μF) $1\mu\text{F} = 1/1,000,000 = 0.000001 = 10^{-6} \text{ F}$

Nanofarad (nF) $1\text{nF} = 1/1,000,000,000 = 0.000000001 = 10^{-9}\text{F}$

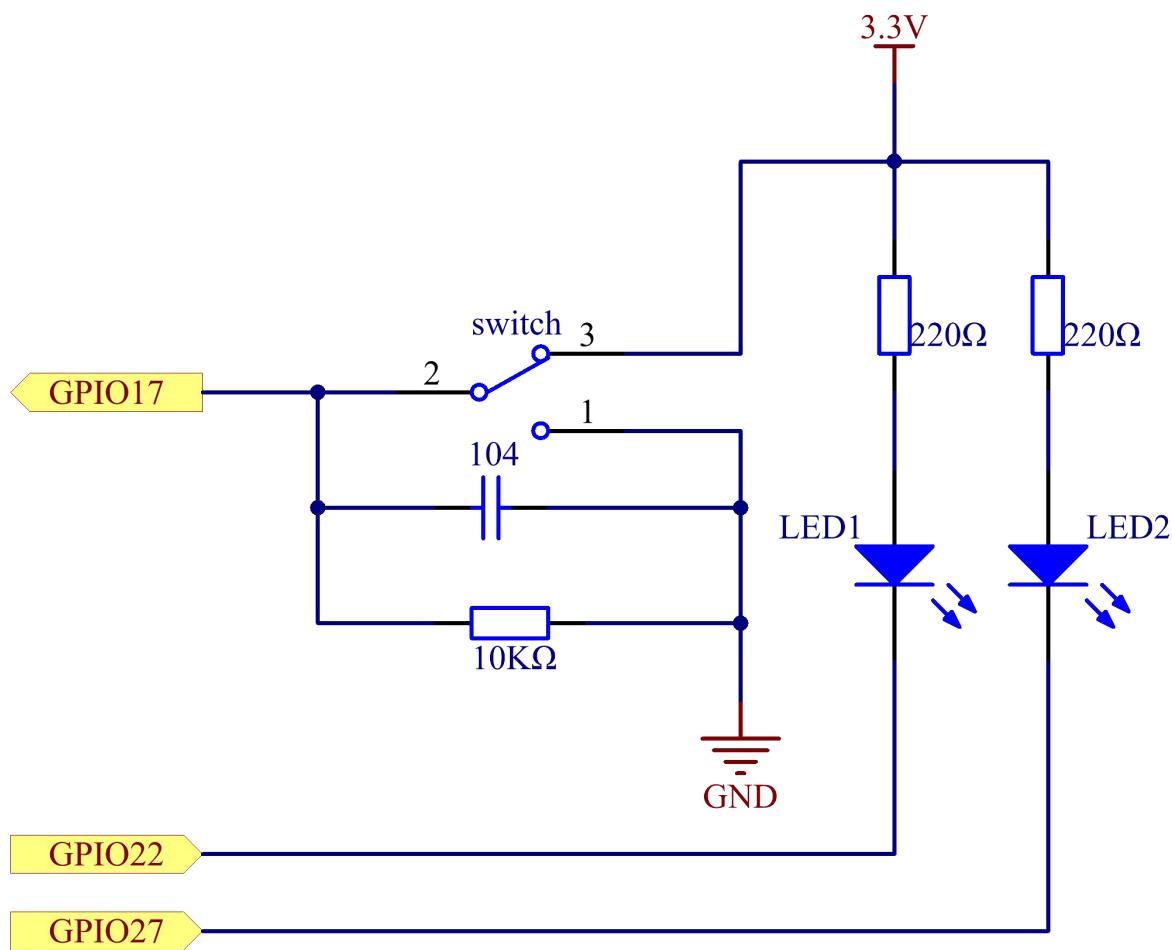
Picofarad (pF) $1\text{pF} = 1/1,000,000,000,000 = 0.000000000001 = 10^{-12}\text{F}$

ご注意： ここでは、104 コンデンサ ($10 \times 10^4 \text{ pF}$) を使用する。抵抗器のリングのように、コンデンサの数字は、ボードに組み立てられた後に値を読み取ることに役立つ。最初の2桁は値を表し、数字の最後の桁は乗数を指す。したがって、104 は 100 nF に等しい 10×10^4 (pF 単位) の累乗を表す。

回路図

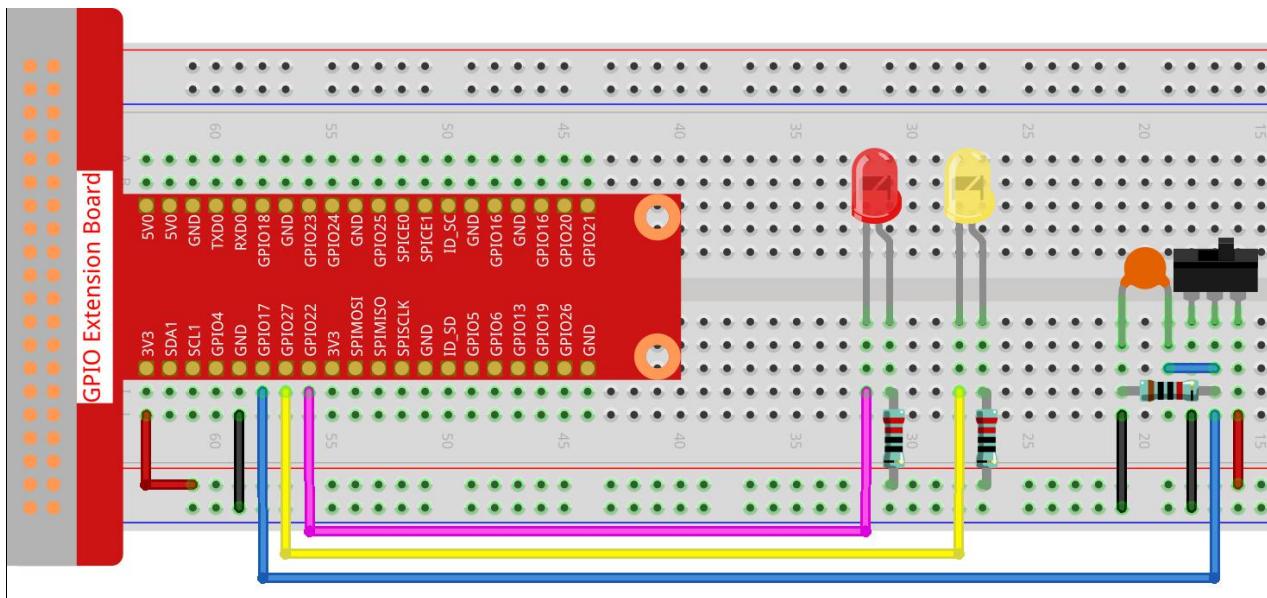
スライドスイッチの中央のピンを GPIO17 に接続し、2つの LED をそれぞれ GPIO22 と GPIO27 に接続する。次に、スライドを引くと、2つの LED が交互に点灯する。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



実験手順

ステップ1: 回路を作る。



```
}

pinMode(slidePin, INPUT);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
while(1){
    // slide switch high, led1 on
    if(digitalRead(slidePin) == 1){
        digitalWrite(led1, LOW);
        digitalWrite(led2, HIGH);
        printf("LED1 on\n");
    }
    // slide switch low, led2 on
    if(digitalRead(slidePin) == 0){
        digitalWrite(led2, LOW);
        digitalWrite(led1, HIGH);
        printf("....LED2 on\n");
    }
}
return 0;
}
```

コードの説明

```
if(digitalRead(slidePin) == 1){
    digitalWrite(led1, LOW);
    digitalWrite(led2, HIGH);
    printf("LED1 on\n");
}
```

スライドを右に引くと、中央のピンと右のピンが接続される。 Raspberry Pi は中央のピンで高レベルを読み取るため、 LED1 は点灯し、 LED2 は消灯する。

```
if(digitalRead(slidePin) == 0){
    digitalWrite(led2, LOW);
    digitalWrite(led1, HIGH);
    printf("....LED2 on\n");
}
```

スライドを左に引くと、中央のピンと左のピンが接続されます。 Raspberry Pi が低レベルを読み取るため、 LED2 が点灯し、 LED1 が消灯する。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ3: 実行する。

```
sudo python3 2.1.2_Slider.py
```

コードの実行中に、スイッチを左側に接続すると、黄色の LED が点灯する。右側に接続すると、赤いライトが点灯する。

コード

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as slide switch pin, GPIO22 as led1 pin, GPIO27 as led2 pin
slidePin = 17
led1Pin = 22
led2Pin = 27

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set slidePin input
    # Set ledPin output,
    # and initial level to High(3.3v)
    GPIO.setup(slidePin, GPIO.IN)
    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():
    while True:
        # slide switch high, led1 on
        if GPIO.input(slidePin) == 1:
            print('    LED1 ON    ')
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)

        # slide switch low, led2 on
        if GPIO.input(slidePin) == 0:
```

```
print('    LED2 ON    ')
GPIO.output(led2Pin, GPIO.LOW)
GPIO.output(led1Pin, GPIO.HIGH)

time.sleep(0.5)
# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(led1Pin, GPIO.HIGH)
    GPIO.output(led2Pin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

コードの説明

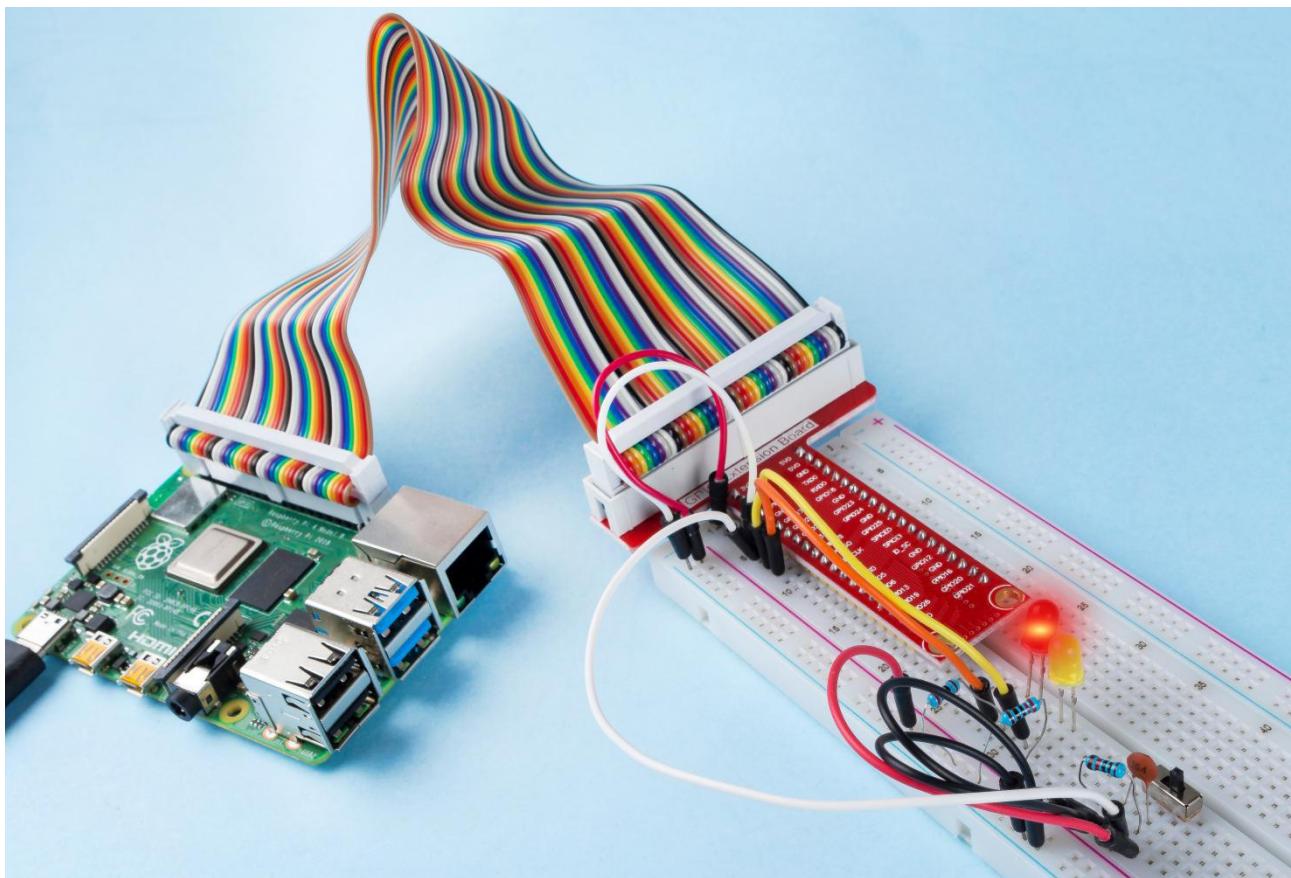
```
if GPIO.input(slidePin) == 1:
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.HIGH)
```

スライドを右に引くと、中央のピンと右のピンが接続される。Raspberry Pi は中央のピンで高レベルを読み取るため、LED1 は点灯し、LED2 は消灯する。

```
if GPIO.input(slidePin) == 0:
    GPIO.output(led2Pin, GPIO.LOW)
    GPIO.output(led1Pin, GPIO.HIGH)
```

スライドを左に引くと、中央のピンと左のピンが接続されます。Raspberry Pi が低レベルを読み取るため、LED2 が点灯し、LED1 が消灯する。

現象画像

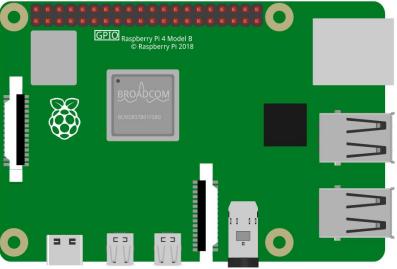
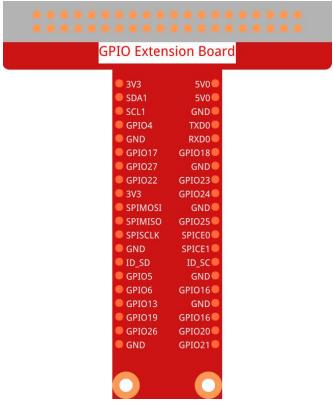
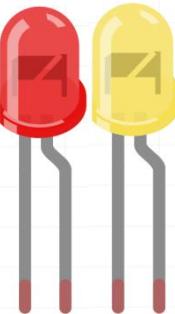
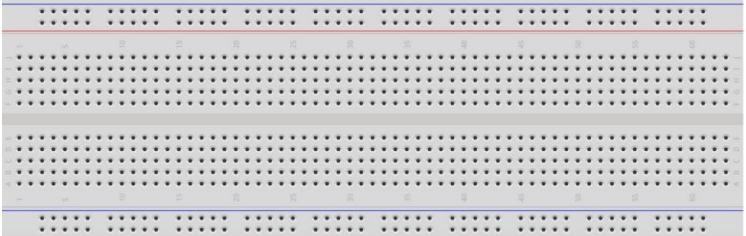


2.1.3 Tilt Switch

前書き

これは、内部に金属製のボールがあるボールチルトスイッチである。小さな角度の傾きを検出するために使用される。

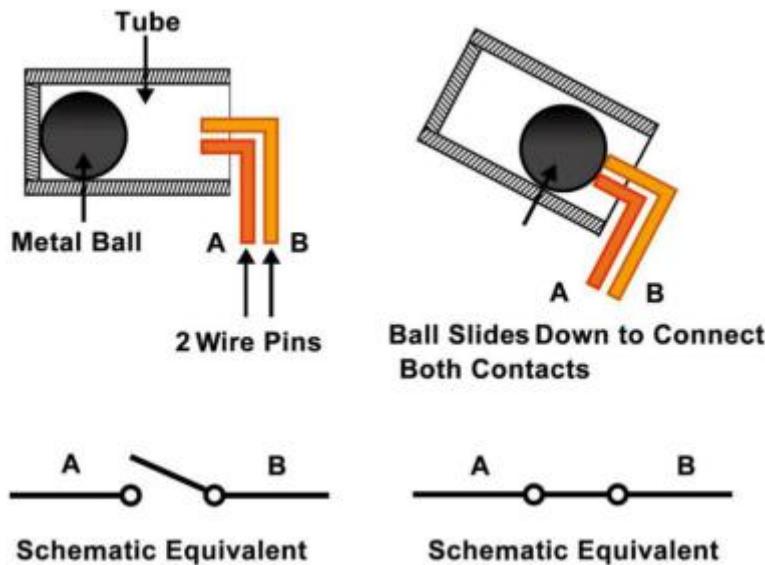
部品

Raspberry Pi 本体*1	T 拡張ボード*1	傾斜スイッチ*1	LED*2
	 GPIO Extension Board Pinout: 3V3 SDA1 GND SCL1 GND GPIO4 TXD0 GND RXD0 GPIO17 GPIO18 GPIO27 GND GPIO22 GPIO23 3V2 GPIO24 GPIMOSI GP1025 GPIMISO GP1026 SPICLK SPIE0 GND SPIE1 ID_SD ID_SC GP105 GND GP106 GPIO16 GP103 GPIO17 GP109 GPIO18 GP1026 GPIO20 GND GPIO21	 SW-520D	
40 ピンケーブル*1		何本のジャンパー線	
			
ブレッドボード*1		抵抗器 (220Ω) *2	
		抵抗器 (10KΩ) *1	

原理

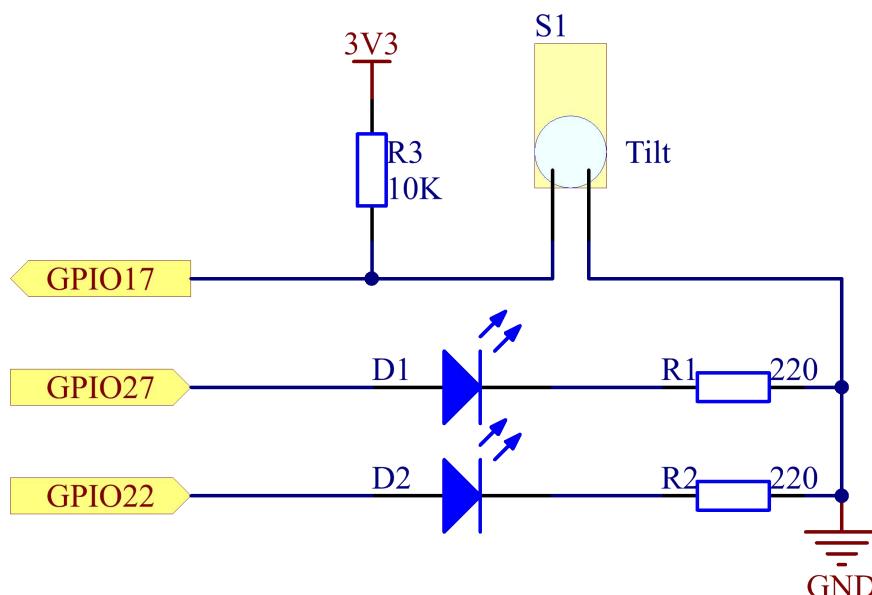
チルト

原理は非常に簡単である。スイッチが特定の角度に傾けられると、内側のボールが転がり落ち、外側のピンに接続された2つの接点に触れて、回路をトリガーする。そうしないと、ボールが接点から遠ざかり、回路が遮断される。



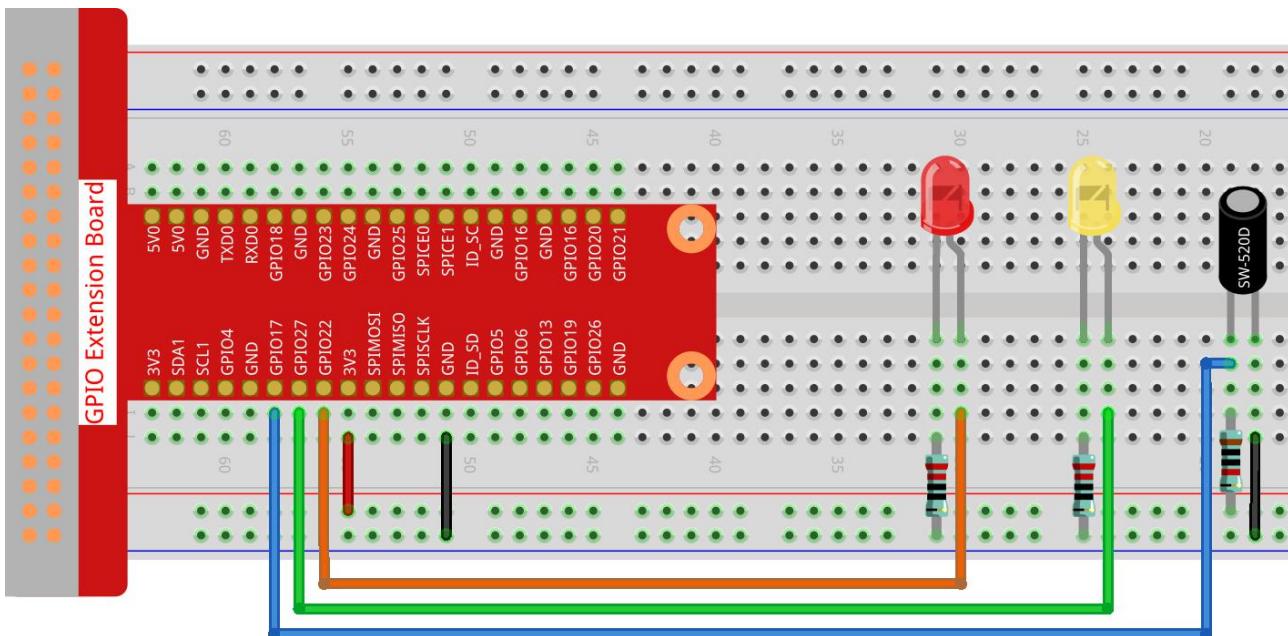
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



実験手順

ステップ1：回路を作る。



➤ C言語ユーザー向け

ステップ2：ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.3/
```

ステップ3：コンパイルする。

```
gcc 2.1.3_Tilt.c -lwiringPi
```

ステップ4：実行する。

```
sudo ./a.out
```

水平に置くと、緑色のLEDが点灯する。傾けると、「Tilt!」画面にプリントされ、赤いLEDが点灯する。再び水平に置くと、緑色のLEDが再び点灯する。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#define TiltPin      0
#define Gpin         2
#define Rpin         3

void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
```

```

if (color == "RED")
{
    digitalWrite(Rpin, HIGH);
    digitalWrite(Gpin, LOW);
}
else if (color == "GREEN")
{
    digitalWrite(Rpin, LOW);
    digitalWrite(Gpin, HIGH);
}
else
    printf("LED Error");
}

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(TiltPin, INPUT);
    LED("GREEN");

    while(1){
        if(0 == digitalRead(TiltPin)){
            delay(10);
            if(0 == digitalRead(TiltPin)){
                LED("RED");
                printf("Tilt!\n");
            }
        }
        else if(1 == digitalRead(TiltPin)){
            delay(10);
            if(1 == digitalRead(TiltPin)){
                LED("GREEN");
            }
        }
    }
    return 0;
}

```

コードの説明

```
void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == "RED")
    {
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}
```

関数 LED() を定義して、2つの LED をオン・オフにする。パラメータの色が赤の場合、赤の LED が点灯する。同様に、パラメータの色が緑の場合、緑の LED が点灯する。

```
while(1){
    if(0 == digitalRead(TiltPin)){
        delay(10);
        if(0 == digitalRead(TiltPin)){
            LED("RED");
            printf("Tilt!\n");
        }
    }
    else if(1 == digitalRead(TiltPin)){
        delay(10);
        if(1 == digitalRead(TiltPin)){
            LED("GREEN");
        }
    }
}
```

傾斜スイッチの読み取り値が 0 の場合、傾斜スイッチが傾斜していることを意味し、関数 LED にパラメーター 「RED」 を書き込んで赤色 LED を点灯させる。そうしない場合、緑色の LED が点灯する。

➤ Python 言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: 実行する。

```
sudo python3 2.1.3_Tilt.py
```

水平に置くと、緑色の LED が点灯する。傾けると、「Tilt!」画面にプリントされ、赤い LED が点灯する。再び水平に置くと、緑色の LED が再び点灯する。

コード

```
import RPi.GPIO as GPIO

TiltPin = 11
Gpin    = 13
Rpin    = 15

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(Gpin, GPIO.OUT)     # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)     # Set Red Led Pin mode to output
    GPIO.setup(TiltPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)   # Set BtnPin's mode is
input, and pull up to high level(3.3V)
    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)

def Print(x):
    if x == 0:
        print ('*****')
        print (' * Tilt! *')
        print ('*****')

def detect(chn):
    Led(GPIO.input(TiltPin))
```

```

Print(GPIO.input(TiltPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.HIGH)      # Green led off
    GPIO.output(Rpin, GPIO.HIGH)      # Red led off
    GPIO.cleanup()                   # Release resource

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
        destroy()

```

コードの説明

```
GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)
```

TiltPin で検出を設定し、検出する関数をコールバックする。

```

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)

```

関数 Led() を定義して、2つの LED をオンまたはオフにする。x=0 の場合、赤い LED が点灯する。そうしないと、緑色の LED が点灯する。

```

def Print(x):
    if x == 0:
        print ('*****')
        print (' * Tilt! *')
        print ('*****')

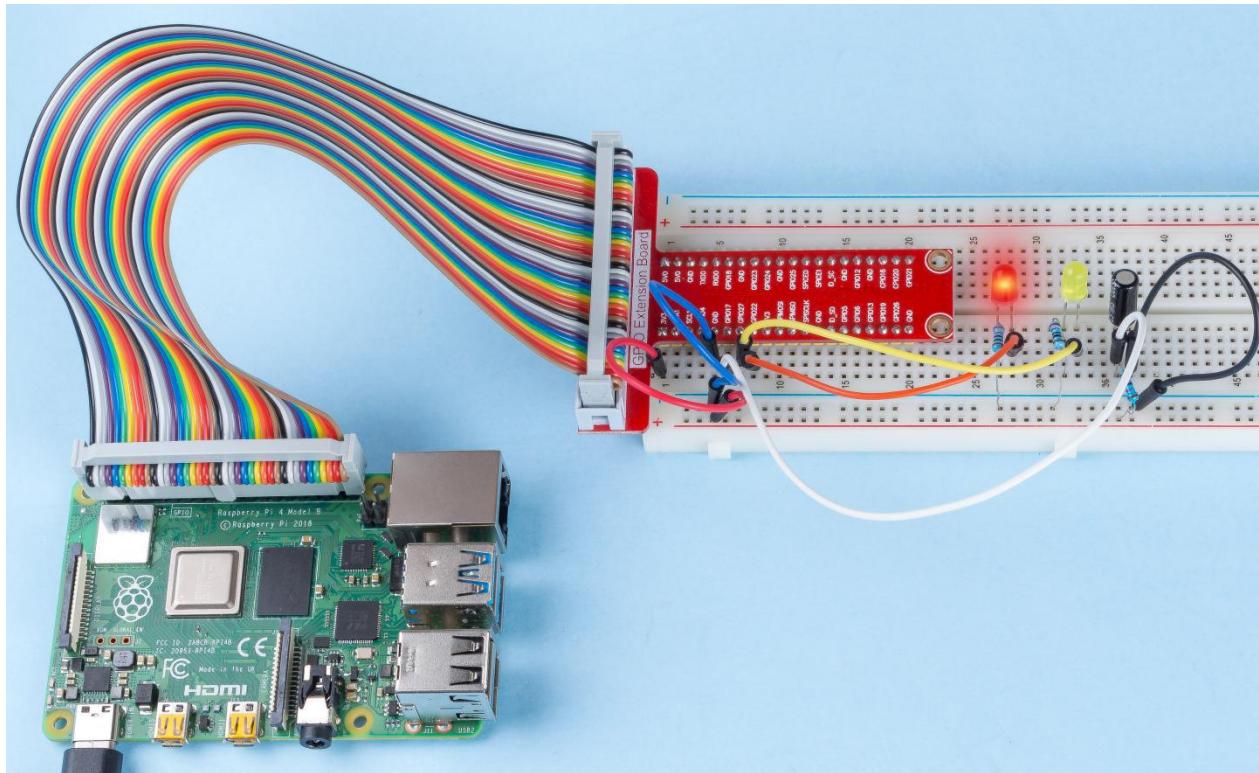
```

関数 Print() を作成して、画面上の上記の文字をプリントする。

```
def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))
```

傾斜コールバックのコールバック関数を定義する。傾斜スイッチの読み取り値を取得してから、関数 Led() が傾斜スイッチの読み取り値に依存する 2 つの LED を点灯・消灯させる。

現象画像

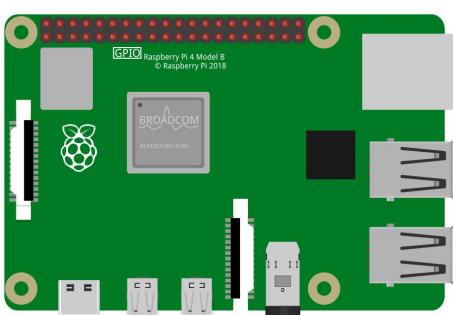
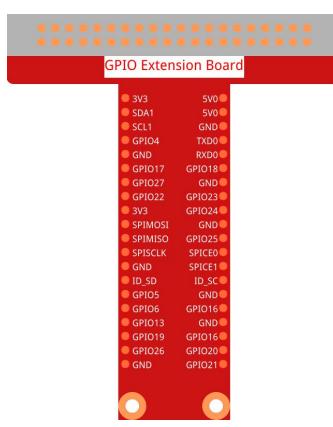
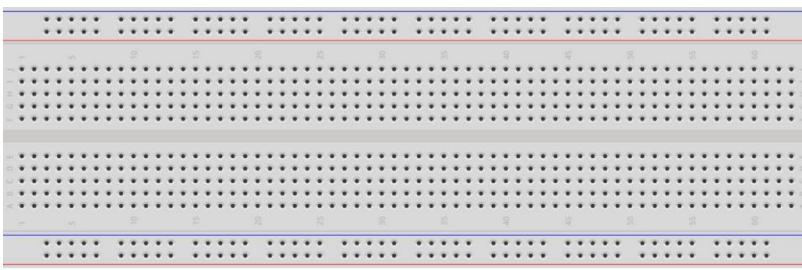


2.1.4 Potentiometer

前書き

ADC 機能を使用してアナログ信号をデジタル信号に変換でき、この実験では、ADC0834 を使用して ADC に関する関数を取得する。ここでは、ポテンショメータを使用してこのプロセスを実装する。ポテンショメータは ADC 機能によって変換される物理量-電圧を変更する。

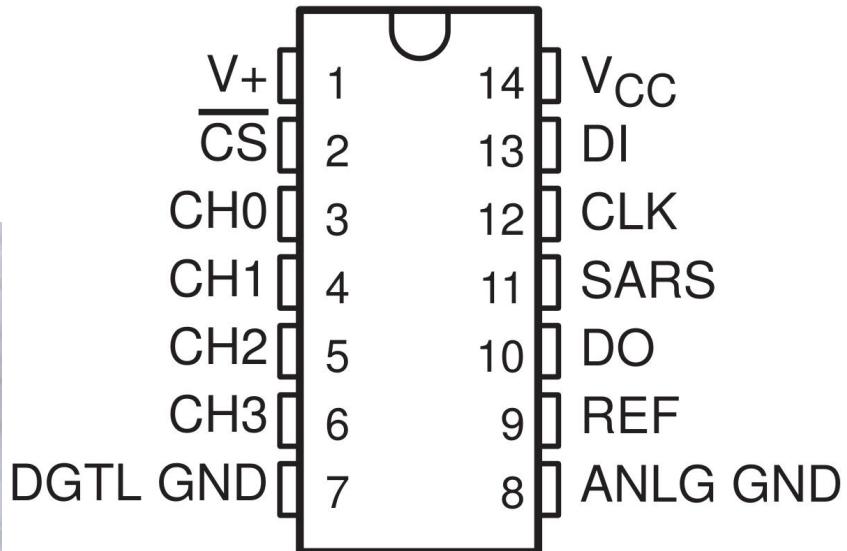
部品

Raspberry Pi 本体*1	T 拡張ボード*1	ポテンショメーター*1																																								
	 GPIO Extension Board <table border="1"><tr><td>3V3</td><td>5V0</td></tr><tr><td>SDA1</td><td>GND</td></tr><tr><td>SCL1</td><td>TxD0</td></tr><tr><td>GPIO4</td><td>RxD0</td></tr><tr><td>GND</td><td>GND</td></tr><tr><td>GPIO17</td><td>GPIO18</td></tr><tr><td>GPIO27</td><td>GPIO23</td></tr><tr><td>GPIO22</td><td>GPIO24</td></tr><tr><td>3V</td><td>GPIO25</td></tr><tr><td>SPI_MISO</td><td>GND</td></tr><tr><td>SPI_MOSI</td><td>GPIO25</td></tr><tr><td>SPI_SCLK</td><td>SPI_CE0</td></tr><tr><td>GND</td><td>SPI_CE1</td></tr><tr><td>ID_SD</td><td>ID_SC</td></tr><tr><td>GPIO5</td><td>GND</td></tr><tr><td>GPIO6</td><td>GPIO16</td></tr><tr><td>GPIO13</td><td>GND</td></tr><tr><td>GPIO19</td><td>GPIO16</td></tr><tr><td>GPIO26</td><td>GPIO20</td></tr><tr><td>GND</td><td>GPIO21</td></tr></table>	3V3	5V0	SDA1	GND	SCL1	TxD0	GPIO4	RxD0	GND	GND	GPIO17	GPIO18	GPIO27	GPIO23	GPIO22	GPIO24	3V	GPIO25	SPI_MISO	GND	SPI_MOSI	GPIO25	SPI_SCLK	SPI_CE0	GND	SPI_CE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	
3V3	5V0																																									
SDA1	GND																																									
SCL1	TxD0																																									
GPIO4	RxD0																																									
GND	GND																																									
GPIO17	GPIO18																																									
GPIO27	GPIO23																																									
GPIO22	GPIO24																																									
3V	GPIO25																																									
SPI_MISO	GND																																									
SPI_MOSI	GPIO25																																									
SPI_SCLK	SPI_CE0																																									
GND	SPI_CE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
40 ピンケーブル*1	ADC0834*1																																									
	抵抗器 (220Ω) *1																																									
ブレッドボード*1	何本のジャンパー線																																									
	LED*1																																									

原理

ADC0834

ADC0834 は、入力設定可能なマルチチャンネルマルチプレクサーとシリアル入力/出力を備えた 8 ビット逐次比較型アナログ-デジタルコンバーターである。シリアル入力/出力は、標準のシフトレジスタまたはマイクロプロセッサとのインターフェースを構成するものである。

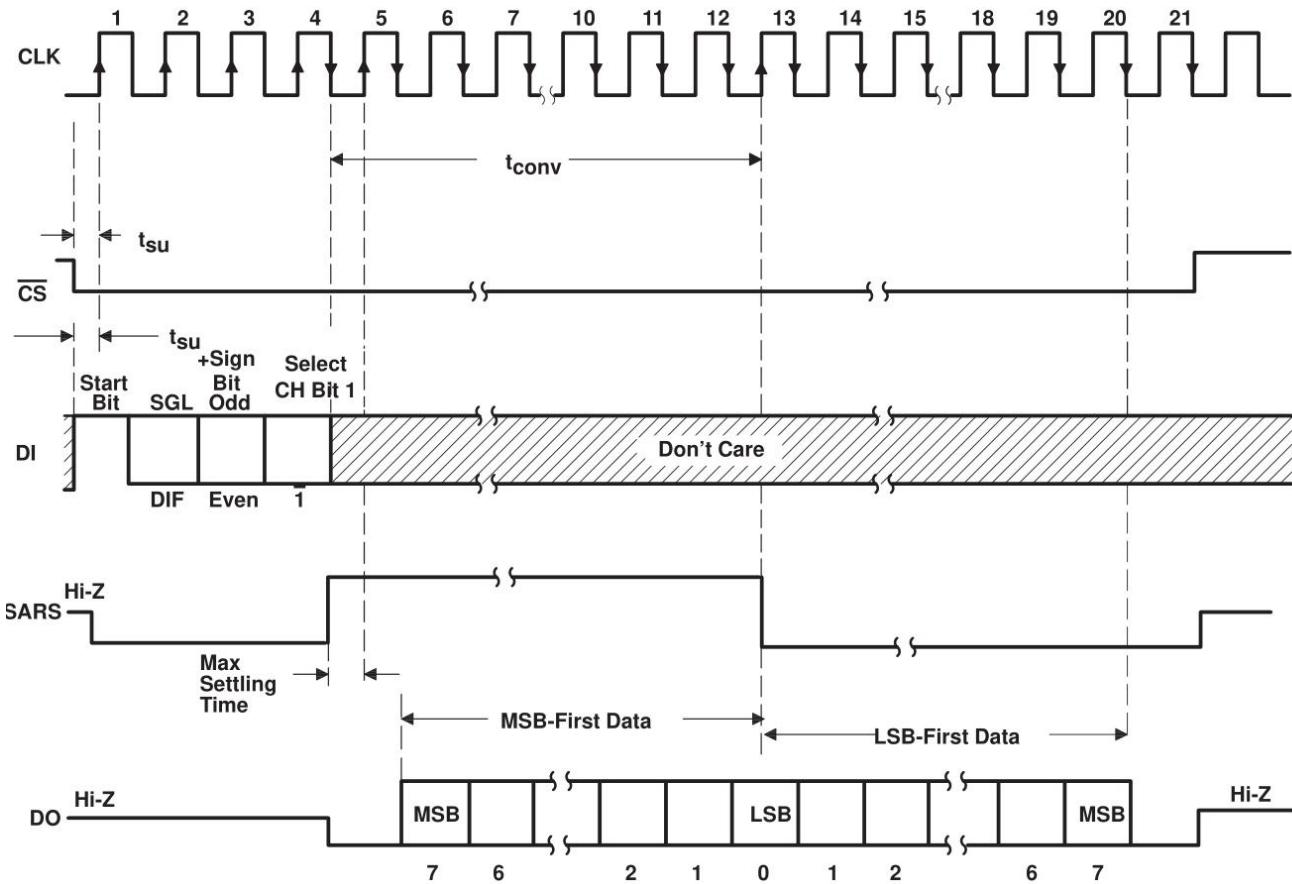


操作シーケンス

変換は CS を low に設定することで開始され、論理回路がすべて有効になる。変換プロセスを完了するには、CS を low に保持する必要がある。次に、プロセッサからクロック入力を受信する。クロック入力の Low から High への遷移ごとに、DI 上のデータがマルチプレクサーチャンネルシフトレジスタに入力される。入力の最初のロジック high はスタートビットである。スタートビットの後に、3~4 ビットの割り当てワードが続く。クロック入力の Low から High への遷移ごとに、スタートビットと割り当てワードがシフトレジスターにシフトする。スタートビットがマルチプレクサレジスタの開始位置にシフトされると、入力チャネルが選択され、変換が開始される。SAR Status 出力 (SARS) は、変換が進行中であることを示すために高レベルになり、マルチプレクサーシフトレジスタへの DI は変換中は無効になる。

1 クロック周期の間隔が自動的に挿入されて、選択されたマルチプレックスチャネルの安定化を実現する。データ出力 DO は高インピーダンス状態から出て、マルチプレクサの整定時間のこの 1 クロック期間に先行する Low を提供する。SAR コンパレータは、抵抗ラダーからの連続出力を入力アナログ信号と比較する。コンパレータ出力は、アナログ入力が抵抗ラダー出力より大きいか小さいかを示す。変換が進むと、変換データが DO 出力ピンから同時に output され、最上位ビット (MSB) が最初になる。

8 クロック周期後、変換が完了し、SARS 出力が LOW になる。最後に、MSB ファーストデータストリームの後に最下位ビットファーストデータを出力する。



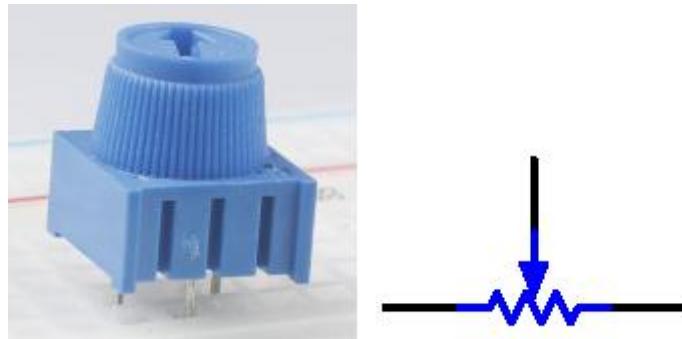
ADC0834 MUX アドレス制御論理表

MUX ADDRESS			CHANNEL NUMBER			
SGL/ \overline{DIF}	ODD/ \overline{EVEN}	SELECT BIT 1	O	1	2	3
L	L	L	+	-		
		H		+/-	+	-
	H	L		-/+	-	+
	H	H				
H	L	L	+			
		H		+		
	H	L				
	H	H				

H = high level, L = low level, – or + = polarity of selected input pin

ポテンショメータ

ポテンショメーターも3つの端子を持つ抵抗部品であり、その抵抗値は定期的な変動に応じて調整できる。ポテンショメータは通常、抵抗器と可動ブラシで構成されている。ブラシが抵抗に沿って移動しているとき、変位に応じて特定の抵抗または電圧出力が生成される。



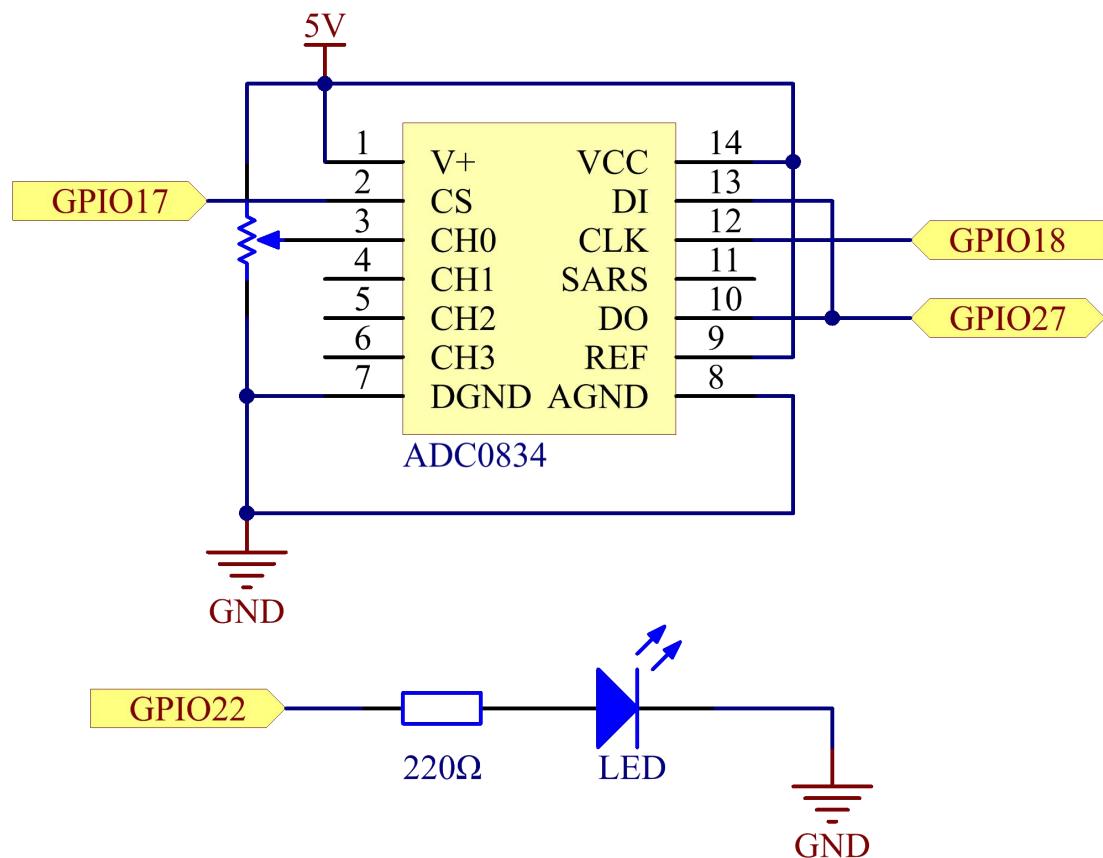
回路内のポテンショメータの機能は次のとおりである：

1. 分圧器として機能する

ポテンショメータは連続的に調整可能な抵抗器である。ポテンショメータのシャフトまたはスライドハンドルを調整すると、可動接点が抵抗器上でスライドする。この時点で、ポテンショメータに印加される電圧と、可動アームが回転した角度または移動距離に応じて、電圧を出力できる。

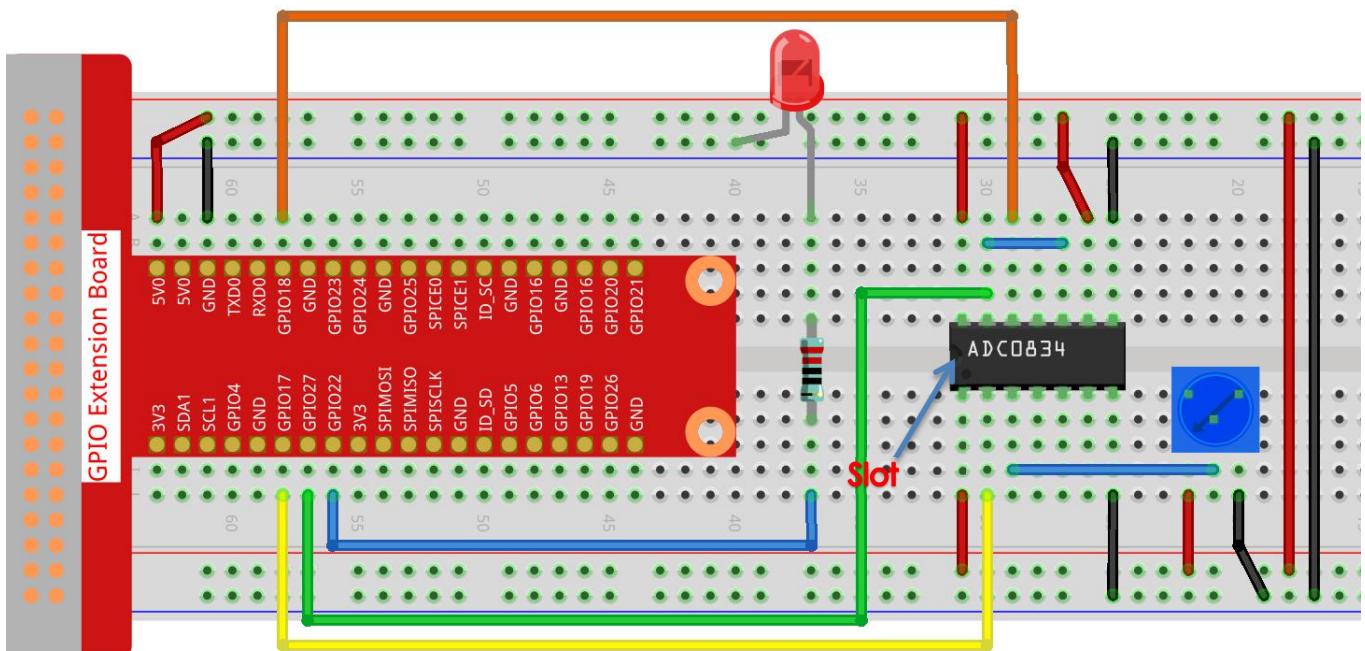
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22



実験手順

ステップ 1: 回路を作る。



ご注意: 写真に示されている対応する位置を参照して、チップを配置してください。配置するときにチップの溝は左側にあることに注意してください。

➤ C 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.4/
```

ステップ 3: コードをコンパイルする。

```
gcc 2.1.4_Potentiometer.c -lwiringPi
```

ステップ 4: 実行する。

```
sudo ./a.out
```

コードの実行後、ポテンショメーターのノブを回すと、それに応じて LED の輝度が変化する。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>

typedef unsigned char uchar;
typedef unsigned int uint;

#define ADC_CS    0
#define ADC_CLK   1
#define ADC_DIO   2
#define LedPin    3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
```

```

digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);

digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

for(i=0;i<8;i++)
{
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

    pinMode(ADC_DIO, INPUT);
    dat1=dat1<<1 | digitalRead(ADC_DIO);
}

for(i=0;i<8;i++)
{
    dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
}

digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
}

```

```
if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
    printf("setup wiringPi failed !");
    return 1;
}
softPwmCreate(LedPin, 0, 100);
pinMode(ADC_CS, OUTPUT);
pinMode(ADC_CLK, OUTPUT);

while(1){
    analogVal = get_ADC_Result(0);
    printf("Current analogVal : %d\n", analogVal);
    softPwmWrite(LedPin, analogVal);
    delay(100);
}
return 0;
}
```

コードの説明

```
#define ADC_CS 0
#define ADC_CLK 1
#define ADC_DIO 2
#define LedPin 3
```

ADC0834 の CS、CLK、DIO を定義し、それぞれ GPIO0、GPIO1、GPIO2 に接続する。それから、GPIO3 に LED を取り付ける。

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
```

```
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);

digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
for(i=0;i<8;i++)
{
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

    pinMode(ADC_DIO, INPUT);
    dat1=dat1<<1 | digitalRead(ADC_DIO);
}

for(i=0;i<8;i++)
{
    dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
}

digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
}
```

ADC0834 には、アナログからデジタルへの変換を行う機能がある。特定のワークフローは次のとおりです：

```
digitalWrite(ADC_CS, 0);
```

CS を低レベルに設定し、AD 変換の有効化を開始する。

```
// Start bit
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

クロック入力の low から high への遷移が最初に発生したとき、スタートビットとして DIO を 1 に設定する。次の三つのステップには、割り当て単語が 3 つある。

```
//Single End mode
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

クロック入力の low から high への遷移が二回発生したらすぐに、DIO を 1 に設定し、SGL モードを選択する。

```
// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

三回目に発生すると、DIO の値は変数 odd によって制御される。

```
//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);
```

CLK のパルスが 4 番目に低レベルから高レベルに変換されると、DIO の値は変数 sel によって制御される。

channel = 0、sel = 0、odd = 0 の条件下では、sel および odd に関する演算式は次のとおりである：

```
int sel = channel > 1 & 1;
int odd = channel & 1;
```

channel = 1、sel = 0、odd = 1 という条件が満たされている場合、次のアドレス制御ロジックテーブルを参照してください。ここで、CH1 が選択され、開始ビットがマルチプレクサレジスタの開始位置にシフトされ、変換が開始される。

MUX ADDRESS			CHANNEL NUMBER			
SGL/ \overline{DIF}	ODD/EVEN	SELECT BIT 1	O	1	2	3
L	L	L	+	-		
L	H	H	-	+	+	-
L	H	L		-	-	+
		H				
H	L	L	+			
H	L	H		+	+	
H	H	L				+
H	H	H				

```
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
```

ここでは、DIO を 1 に二回設定し、それを無視してください。

```
for(i=0;i<8;i++)
{
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

    pinMode(ADC_DIO, INPUT);
    dat1=dat1<<1 | digitalRead(ADC_DIO);
}
```

最初の for() statement で、CLK の五番目のパルスが High レベルから低レベルに変換したらすぐに、DIO を入力モードに設定してください。それから、変換が開始され、変換された値が変数 dat1 に保存される。8 クロック周期後、変換が完了する。

```
for(i=0;i<8;i++)
{
    dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
}
```

2 番目最初の for() statement では、他の最初の 8 つの後に DO を介して値を変換を出力し、変数 dat2 に保存する。

```
digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
```

return (dat1 == dat2) ? dat1: 0 は、変換中に得られた値と出力値を比較するために使用される。それらが互いに等しい場合、変換値 dat1 を出力する。それ以外の場合は、0 を出力する。これで、ADC0834 の処理が完了した。

```
softPwmCreate(LedPin, 0, 100);
```

この機能はソフトウェアを使用して PWM ピン LedPin を作成し、初期パルス幅を 0 に設定し、PWM の周期を $100 \times 100\text{us}$ にするために使用される。

```
while(1){  
    analogVal = get_ADC_Result(0);  
    printf("Current analogVal : %d\n", analogVal);  
    softPwmWrite(LedPin, analogVal);  
    delay(100);  
}
```

メインプログラムで、ポテンショメータに接続されているチャネル 0 の値を読み取ります。そして、値を変数 analogVal に保存し、それを LedPin に書き込みます。これで、ポテンショメーターの値によって LED の明るさが変化することがわかります。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 2.1.4_Potentiometer.py
```

コードの実行後、ポテンショメーターのノブを回すと、それに応じて LED の輝度が変化する。

コード

```
#!/usr/bin/env python3  
  
import RPi.GPIO as GPIO  
import ADC0834  
import time  
  
LedPin = 22  
  
def setup():  
    global led_val
```

```

# Set the GPIO modes to BCM Numbering
GPIO.setmode(GPIO.BCM)
# Set all LedPin's mode to output and initial level to High(3.3v)
GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
ADC0834.setup()
# Set led as pwm channel and frequency to 2KHz
led_val = GPIO.PWM(LedPin, 2000)

# Set all begin with value 0
led_val.start(0)

# Define a MAP function for mapping values. Like from 0~255 to 0~100
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def destroy():
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()

def loop():
    while True:
        res = ADC0834.getResult()
        print ('res = %d' % res)
        R_val = MAP(res, 0, 255, 0, 100)
        led_val.ChangeDutyCycle(R_val)
        time.sleep(0.2)

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
                           # executed.
        destroy()

```

コードの説明

```
import ADC0834
```

ADC0834 ライブライをインポートする。コマンド nano ADC0834.py を呼び出して、ライブラリの内容を確認できる。

```
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequency to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)

    # Set all begin with value 0
    led_val.start(0)
```

setup() で、命名方法を BCM として定義し、LedPin を PWM チャネルとして設定し、2Khz の周波数にレンダリングする。

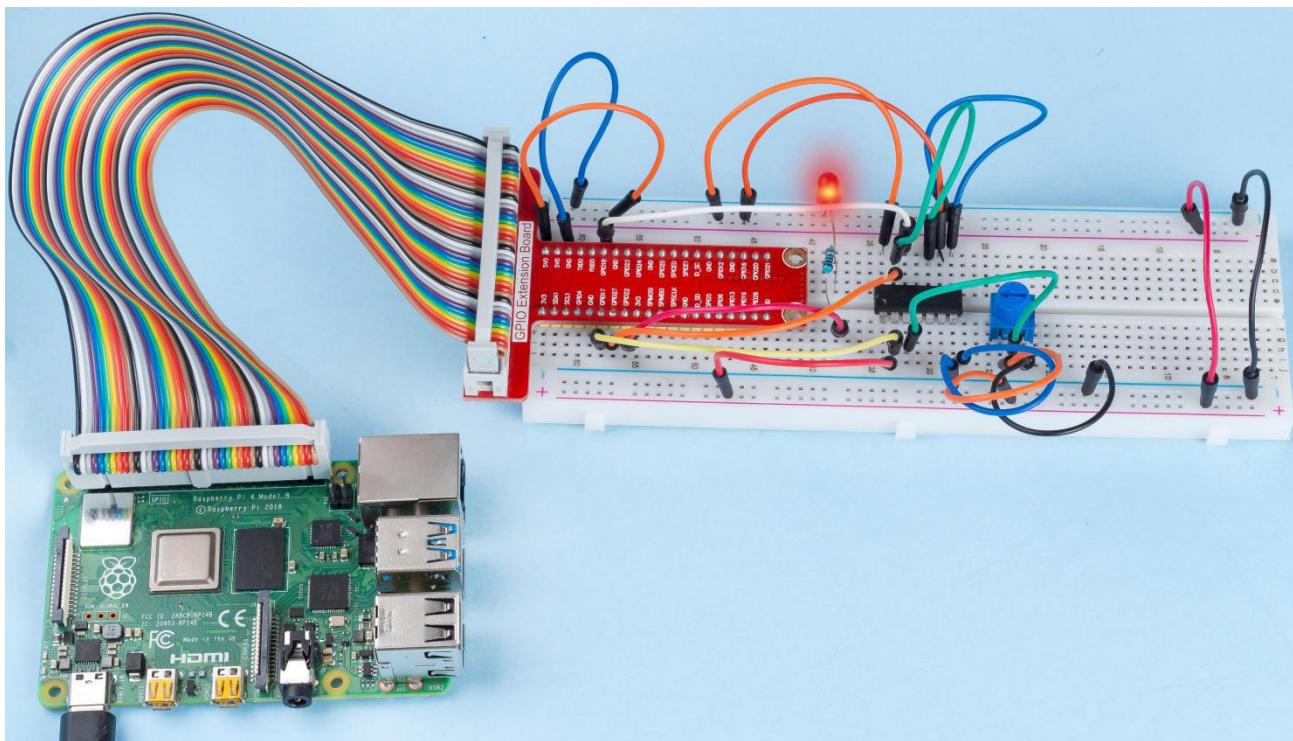
ADC0834.setup(): ADC0834 を初期化し、ADC0834 の定義された CS、CLK、DIO をそれぞれ GPIO17、GPIO18、GPIO27 に接続する。

```
def loop():
    while True:
        res = ADC0834.getResult()
        print ('res = %d' % res)
        R_val = MAP(res, 0, 255, 0, 100)
        led_val.ChangeDutyCycle(R_val)
        time.sleep(0.2)
```

関数 getResult() は ADC0834 の 4 つのチャンネルのアナログ値を読み取るために使用される。デフォルトでは、関数は CH0 の値を読み取り、他のチャネルを読み取りたい場合は、() にチャネル番号を入力してください (例えば、getResult(1))。

関数 loop() は最初に CH0 の値を読み取り、それから変数 res に値を割り当てる。その後、関数 MAP を呼び出して、ポテンショメーターの読み取り値を 0~100 にマッピングする。このステップは LedPin のデューティサイクルを制御するために使用される。これで、ポテンショメータの値によって LED の輝度が変化していることがわかる。

現象画像

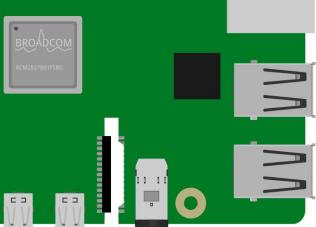
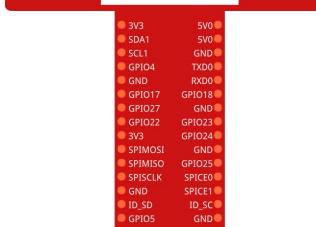
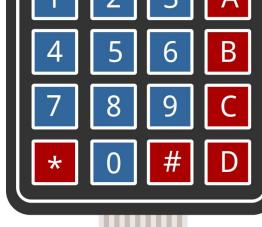
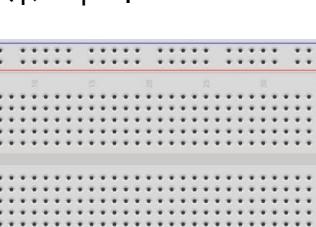


2.1.5 Keypad

前書き

キーパッドは、ボタンの長方形の配列である。このプロジェクトでは、入力文字を使用する。

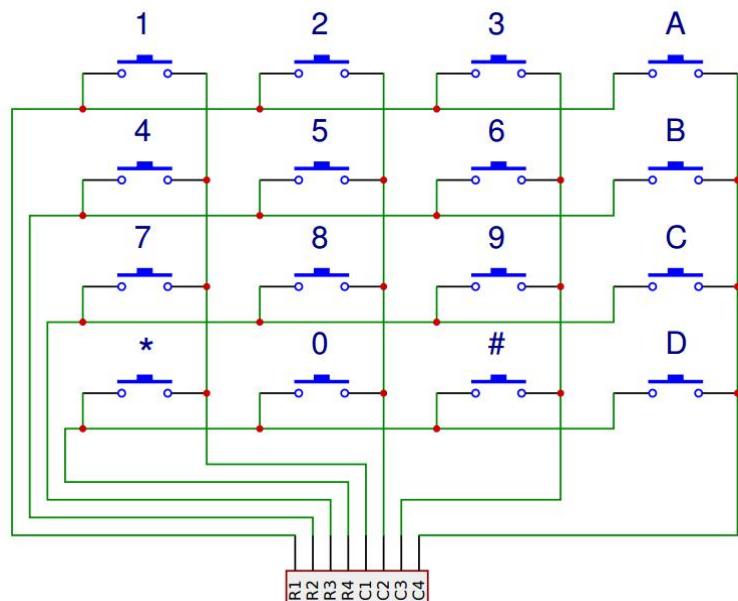
部品

Raspberry Pi 本体*1	T 拡張ボード*1	キーパッド*1																																								
	 GPIO Extension Board <table border="1"><tr><td>3V3</td><td>5V0</td></tr><tr><td>SDA1</td><td>5V0</td></tr><tr><td>SCL1</td><td>GND</td></tr><tr><td>GPIO4</td><td>TXD0</td></tr><tr><td>GND</td><td>RXD0</td></tr><tr><td>GPIO17</td><td>GPIO18</td></tr><tr><td>GPIO27</td><td>GND</td></tr><tr><td>GPIO22</td><td>GPIO23</td></tr><tr><td>3V</td><td>GPIO24</td></tr><tr><td>SPIMOSI</td><td>GND</td></tr><tr><td>SPIMISO</td><td>GPIO25</td></tr><tr><td>SPISCLK</td><td>GPIO26</td></tr><tr><td>GND</td><td>SRIC1</td></tr><tr><td>IO_SD</td><td>ID_SC</td></tr><tr><td>GPIO5</td><td>GND</td></tr><tr><td>GPIO6</td><td>GPIO16</td></tr><tr><td>GPIO13</td><td>GND</td></tr><tr><td>GPIO19</td><td>GPIO16</td></tr><tr><td>GPIO26</td><td>GPIO20</td></tr><tr><td>GND</td><td>GPIO21</td></tr></table>	3V3	5V0	SDA1	5V0	SCL1	GND	GPIO4	TXD0	GND	RXD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V	GPIO24	SPIMOSI	GND	SPIMISO	GPIO25	SPISCLK	GPIO26	GND	SRIC1	IO_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	 1 2 3 A 4 5 6 B 7 8 9 C * 0 # D
3V3	5V0																																									
SDA1	5V0																																									
SCL1	GND																																									
GPIO4	TXD0																																									
GND	RXD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V	GPIO24																																									
SPIMOSI	GND																																									
SPIMISO	GPIO25																																									
SPISCLK	GPIO26																																									
GND	SRIC1																																									
IO_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
ブレッドボード*1		抵抗器 (10KΩ) *8																																								
																																										
40 ピンケーブル*1		何本のジャンパー線																																								
																																										

原理

キーパッド

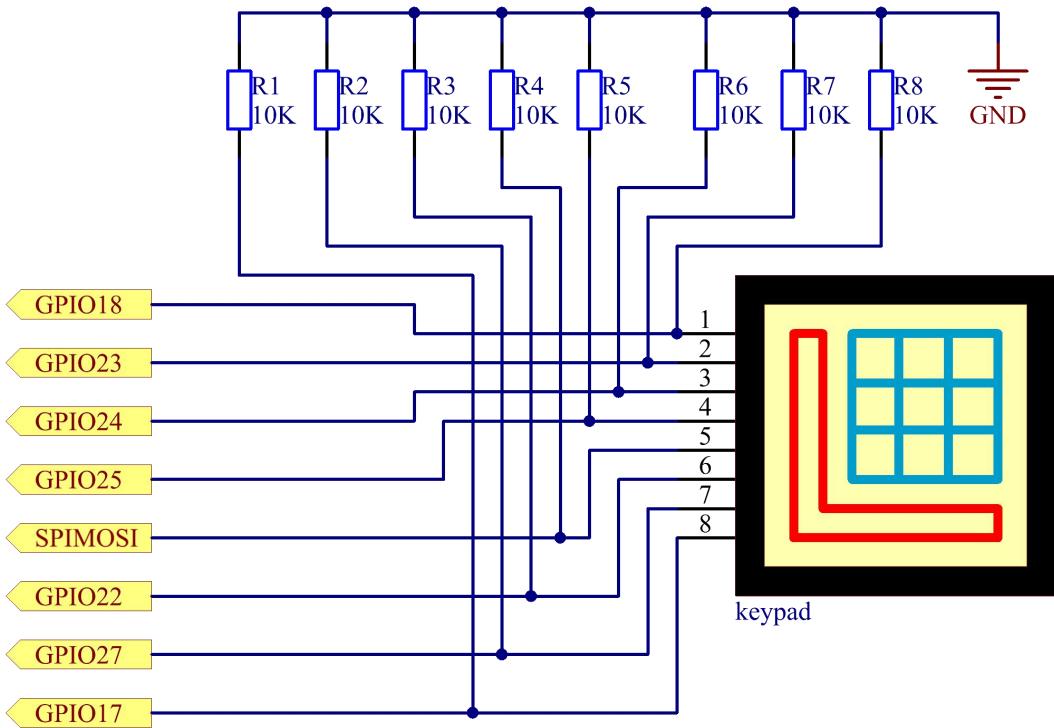
キーパッドは12個または16個のOFF- (ON) ボタンの長方形配列である。リボンケーブルとの接続またはプリント基板への挿入に適したヘッダーを介してそれらの接点にアクセスする。一部のキーパッドでは、各ボタンはヘッダーの個別の連絡先に接続されるが、すべてのボタンは共通の接地を共有する。



多くの場合、ボタンはマトリックスエンコードされている。つまり、各ボタンはマトリックス内の一意のコンダクターペアをブリッジしている。この構成は、マイクロコントローラによるポーリングに適し、4本の水平線のそれぞれに順番に出力パルスを送信するようにプログラムできる。各パルス中に、残りの4本の垂直ワイヤを順番にチェックして、信号を伝送しているのがどれかを判断する。信号が存在しない場合にマイクロコントローラの入力が予期しない動作をすることを防ぐため、入力ワイヤにプルアップまたはプルダウン抵抗を追加してください。

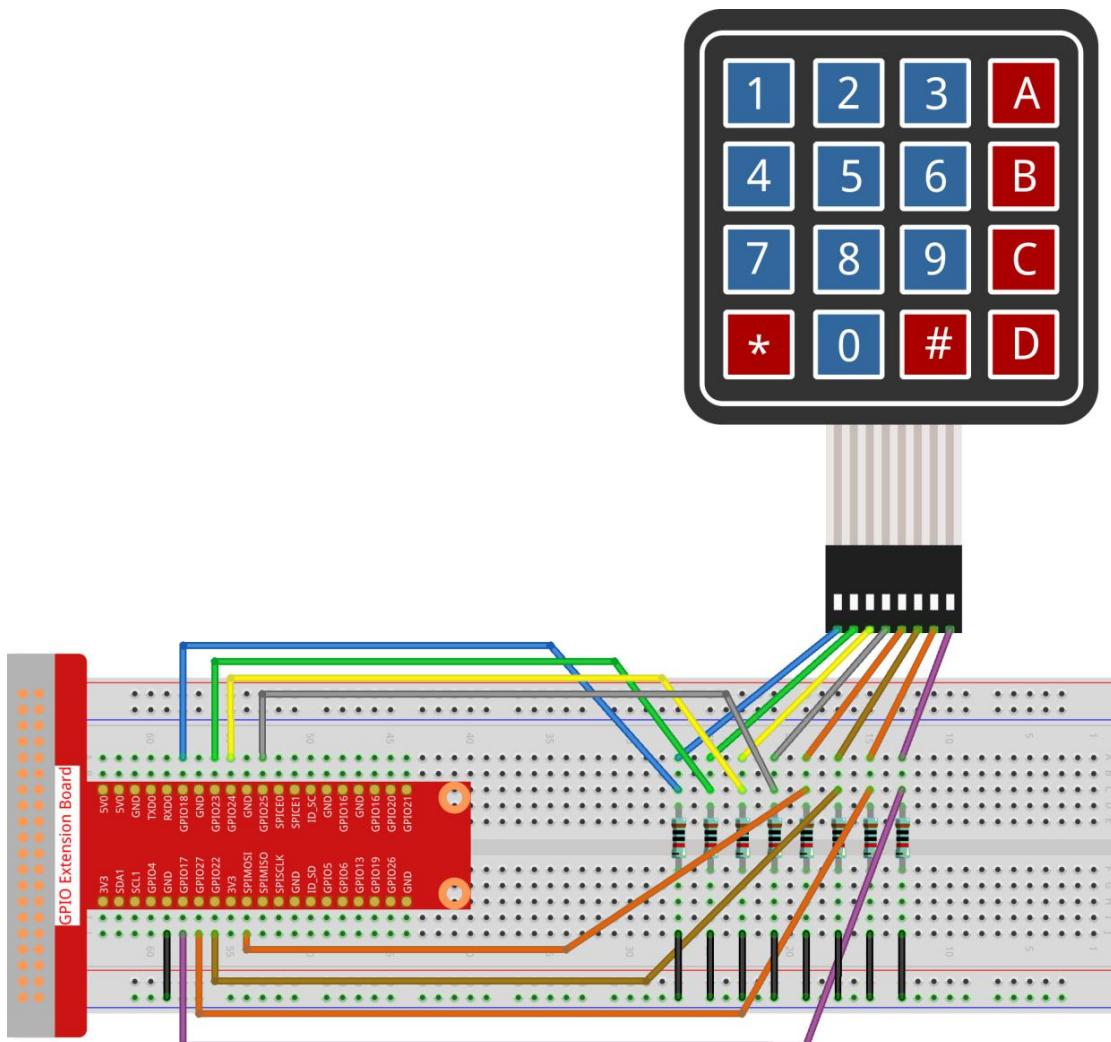
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPI MOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.5/
```

ステップ 3: コードをコンパイルする。

```
gcc 2.1.5_Keypad.cpp -lwiringPi
```

ステップ 4: 実行する。

```
sudo ./a.out
```

コードの実行後、キーパッドで押されたボタンの値（ボタン値）が画面にプリントされる。

コード

```
#include <wiringPi.h>
#include <stdio.h>

#define ROWS 4
#define COLS 4
#define BUTTON_NUM (ROWS * COLS)

unsigned char KEYS[BUTTON_NUM] {
    '1','2','3','A',
    '4','5','6','B',
    '7','8','9','C',
    '*', '0', '#', 'D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};

void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);

void init(void) {
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
        pinMode(colPins[i], INPUT);
```

```

    }

}

int main(void){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            keyPrint(pressed_keys);
            keyCopy(last_key_pressed, pressed_keys);
        }
        delay(100);
    }
    return 0;
}

void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
    for(int i=0 ; i<ROWS ; i++){
        digitalWrite(rowPins[i], HIGH);
        for(int j =0 ; j < COLS ; j++){
            index = i * ROWS + j;
            if(digitalRead(colPins[j]) == 1){
                result[count]=KEYS[index];
                count += 1;
            }
        }
        delay(1);
        digitalWrite(rowPins[i], LOW);
    }
}

```

}

```
bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}
```

```
void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}
```

```
void keyPrint(unsigned char* a){
    if (a[0] != 0){
        printf("%c",a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c",a[i]);
        }
    }
    printf("\n");
}
```

```
void keyClear(unsigned char* a){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = 0;
    }
}
```

```
int keyIndexOf(const char value){
    for (int i=0; i<BUTTON_NUM; i++){
        if ((const char)KEYS[i] == value){
            return i;
        }
    }
```

```

    }
    return -1;
}

```

コードの説明

```

unsigned char KEYS[BUTTON_NUM] {
    '1','2','3','A',
    '4','5','6','B',
    '7','8','9','C',
    '*','0','#','D';
}

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};

```

マトリックスキーボードの各キーを配列 keys []に表示し、各行と列にピンを定義する。

```

while(1){
    keyRead(pressed_keys);
    bool comp = keyCompare(pressed_keys, last_key_pressed);
    if (!comp){
        keyPrint(pressed_keys);
        keyCopy(last_key_pressed, pressed_keys);
    }
    delay(100);
}

```

これは、ボタン値を読み取り、プリントするメイン関数の一部である。

関数 keyRead() は、すべてのボタンの状態を読み取る。

KeyCompare() と keyCopy () は、ボタンの状態が変化したかどうか（つまり、ボタンが押されたか離されたか）を判断するために使用される。

KeyPrint() は現在のレベルが高レベル（ボタンが押されている）のボタンのボタン値をプリントする。

```

void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
    for(int i=0 ; i<ROWS ; i++ ){
        digitalWrite(rowPins[i], HIGH);

```

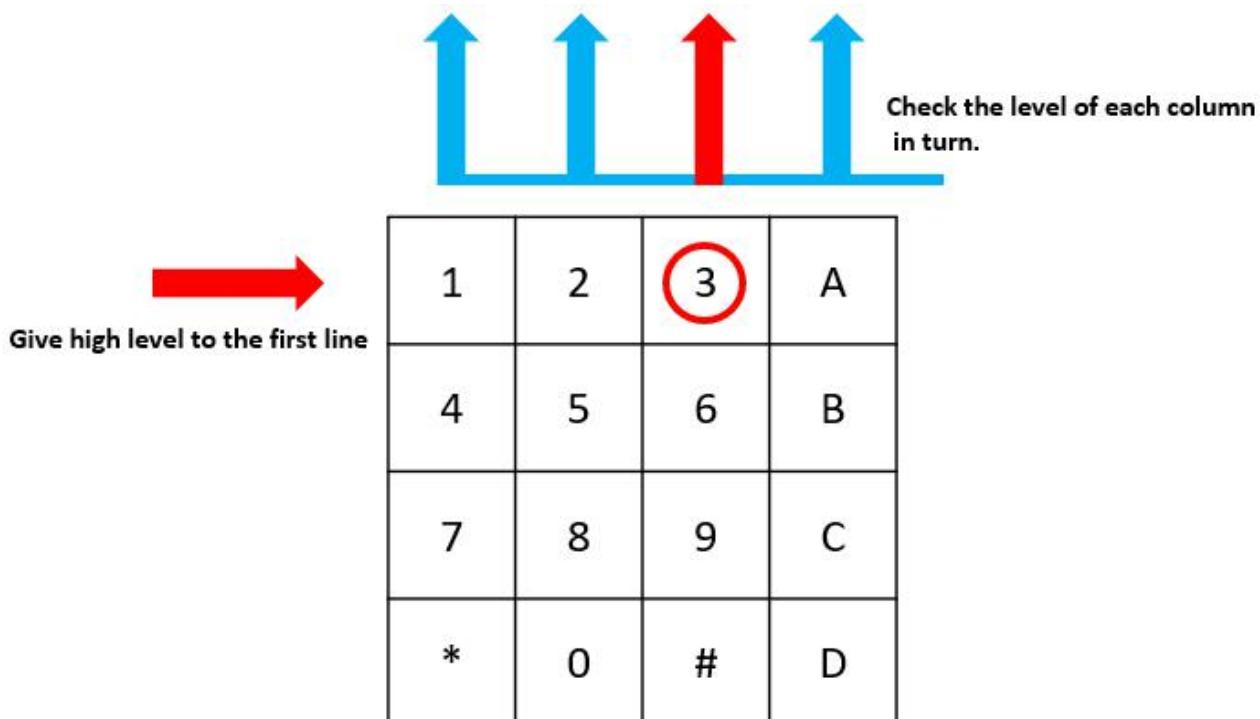
```

for(int j =0 ;j < COLS ;j++){
    index = i * ROWS + j;
    if(digitalRead(colPins[j]) == 1){
        result[count]=KEYS[index];
        count += 1;
    }
}
delay(1);
digitalWrite(rowPins[i], LOW);
}
}

```

この関数は各行に順番に高レベルを割り当て、列のキーが押されると、キーが配置されている列が高レベルになる。two-layer loop の判定後、キー状態のコンパイルにより配列 (result[]) が生成される。

ボタン 3 を押すとき：



The button whose value is “3” is pressed.

RowPin [0]は高レベルで書き込み、colPin [2]は高レベルになる。ColPin [0]、colPin [1]、colPin [3]は低レベルになる。

これにより、0,0,1,0 が得られる。rowPin [1]、rowPin [2]、rowPin [3]が高レベルで書き込まれると、colPin [0]～colPin [4]は低レベルになる。

ループ判定が完了すると、配列が生成される：

```
result[BUTTON_NUM] {
    0, 0, 1, 0,
    0, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 0, 0};
```

```
bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}
```

これら二つの関数は、キーの状態が変化したかどうかを判断するために使用され、たとえば、「3」または「2」を押したときに手を離すと、keyCompare () は false を返す。

KeyCopy () はそれぞれの比較後に配列 (last_key_pressed [BUTTON_NUM]) の現在のボタン値を書き換えるために使用される。ですから次回にそれらを比較できる。

```
void keyPrint(unsigned char* a){
    //printf("{");
    if (a[0] != 0){
        printf("%c", a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c", a[i]);
        }
    }
    printf("\n");
}
```

この関数は現在押されているボタンの値をプリントするために使用される。「1」ボタンを押すと、「1」がプリントされる。ボタン「1」と「3」が押されると、「1, 3」がプリントされる。

➤ Python 言語ユーザー向け

ステップ2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: 実行する。

```
sudo python3 2.1.5_Keypad.py
```

コードの実行後、キーパッドで押されたボタンの値（ボタン値）が画面にプリントされる。

コード

```
import RPi.GPIO as GPIO
import time

class Keypad():

    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys

    def setup():
```

```
global keypad, last_key_pressed
rowsPins = [18,23,24,25]
colsPins = [10,22,27,17]
keys = ["1","2","3","A",
        "4","5","6","B",
        "7","8","9","C",
        "*","0","#","D"]
keypad = Keypad(rowsPins, colsPins, keys)
last_key_pressed = []

def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Release resource
    GPIO.cleanup()

if __name__ == '__main__':      # Program start from here
    try:
        setup()
        while True:
            loop()
    except KeyboardInterrupt:    # When 'Ctrl+C' is pressed, the program destroy() will be
                               # executed.
        destroy()
```

コードの説明

```
def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
    colsPins = [10,22,27,17]
    keys = ["1","2","3","A",
            "4","5","6","B",
            "7","8","9","C",
            "*","0","#","D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []
```

マトリックスキーボードの各キーを配列 keys []に表示し、各行と列にピンを定義する。

```
def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)
```

これは、ボタン値を読み取り、プリントするメイン関数の一部である。

関数 keyRead () は、すべてのボタンの状態を読み取る。

if len (pressed_keys) ! = 0 と last_key_pressed! = Pressed_keys のステートメントは、キーが押されたかどうか、押されたボタンの状態を判断するために使用される。
(「1」を押したときに「3」を押した場合、判断は受け入れられる。)

条件が主張できる場合、現在押されているキーの値をプリントする。

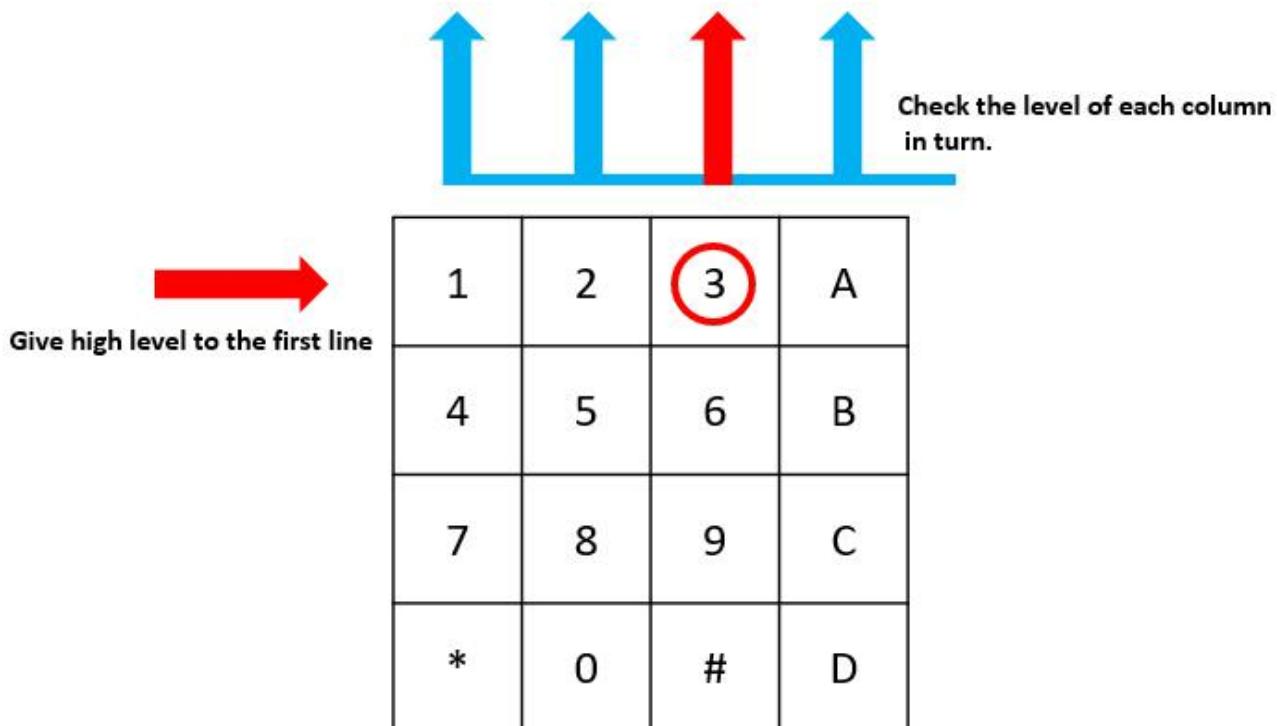
ステートメント last_key_pressed = pressed_keys は、それぞれの判断の状態を配列 last_key_pressed に割り当て、次の条件判断を容易にする。

```
def read(self):
    pressed_keys = []
    for i, row in enumerate(self.rowsPins):
        GPIO.output(row, GPIO.HIGH)
        for j, col in enumerate(self.colsPins):
            index = i * len(self.colsPins) + j
            if (GPIO.input(col) == 1):
```

```
pressed_keys.append(self.keys[index])
GPIO.output(row, GPIO.LOW)
return pressed_keys
```

この関数は各行に順番に高レベルを割り当て、列のボタンが押されると、キーが配置されている列が高レベルになる。2層ループが判定された後、状態が 1 のボタンの値は、pressed_keys 配列に保存される。

キー「3」を押すと：



The button whose value is "3" is pressed.

rowPins [0]は高レベルで書き込まれ、colPins [2]は高レベルになり、colPins [0]、colPins [1]、colPins [3]は低レベルになる。

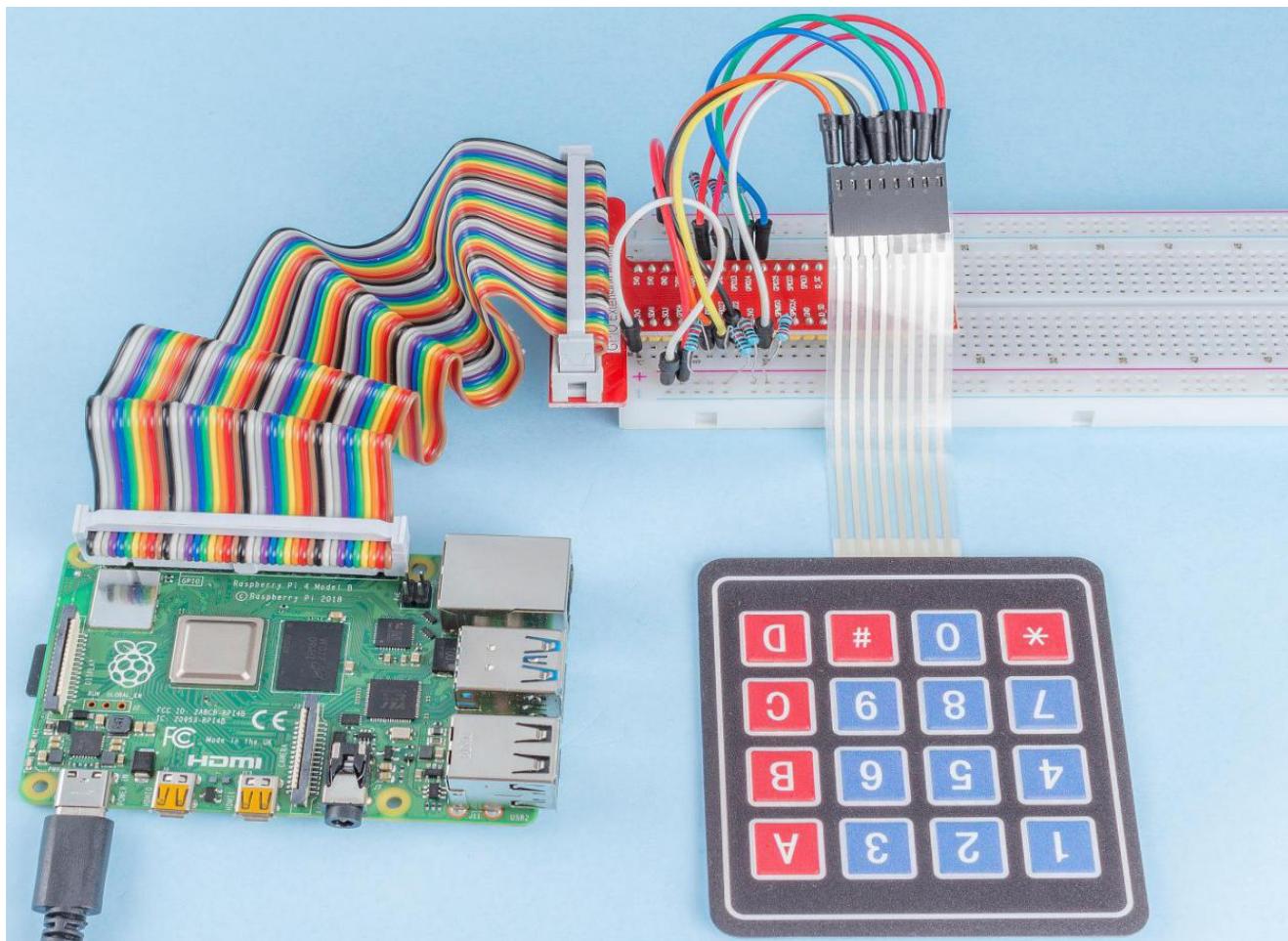
4つの状態がある：0、0、1、0。そして、pressed_keys に「3」を書き込む。

rowPins [1]、rowPins [2]、rowPins [3]が高レベルに書き込まれると、colPins [0] ~ colPins [4]は低レベルになる。

ループが停止し、pressed_keys = '3'が返される。

ボタン「1」と「3」を押すと、pressed_keys = ['1', '3']が返される。.

現象画像

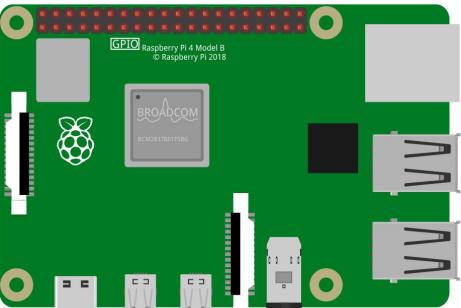
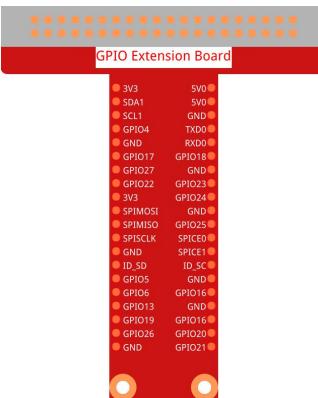
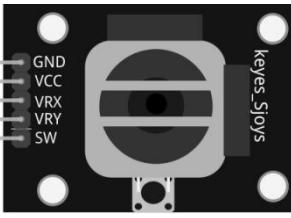
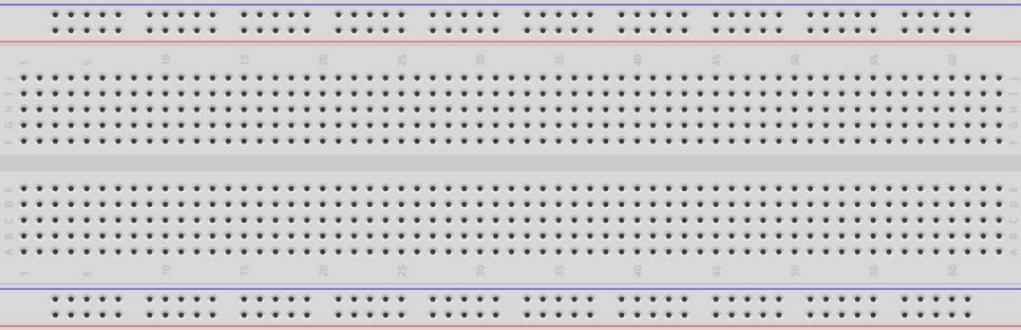


2.1.6 Joystick

前書き

このプロジェクトでは、ジョイスティックの仕組みを学習する。ジョイスティックを操作して、結果を画面に表示する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	ジョイスティック*1																																								
	 GPIO Extension Board <table border="1"><tr><td>3V3</td><td>5V0</td></tr><tr><td>SDA1</td><td>GND</td></tr><tr><td>SCL1</td><td>GND</td></tr><tr><td>GPIO4</td><td>TXD0</td></tr><tr><td>GND</td><td>RXD0</td></tr><tr><td>GPIO17</td><td>GPIO18</td></tr><tr><td>GPIO27</td><td>GND</td></tr><tr><td>GPIO22</td><td>GPIO23</td></tr><tr><td>3V3</td><td>GPIO24</td></tr><tr><td>SPI MOSI</td><td>GND</td></tr><tr><td>SPI MISO</td><td>GPIO25</td></tr><tr><td>SPI CLK</td><td>SPICE0</td></tr><tr><td>GND</td><td>SPICE1</td></tr><tr><td>ID_SD</td><td>ID_SC</td></tr><tr><td>GPIO5</td><td>GND</td></tr><tr><td>GPIO6</td><td>GPIO16</td></tr><tr><td>GPIO13</td><td>GND</td></tr><tr><td>GPIO19</td><td>GPIO10</td></tr><tr><td>GPIO26</td><td>GPIO20</td></tr><tr><td>GND</td><td>GPIO21</td></tr></table>	3V3	5V0	SDA1	GND	SCL1	GND	GPIO4	TXD0	GND	RXD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPI MOSI	GND	SPI MISO	GPIO25	SPI CLK	SPICE0	GND	SPICE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO10	GPIO26	GPIO20	GND	GPIO21	
3V3	5V0																																									
SDA1	GND																																									
SCL1	GND																																									
GPIO4	TXD0																																									
GND	RXD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPI MOSI	GND																																									
SPI MISO	GPIO25																																									
SPI CLK	SPICE0																																									
GND	SPICE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO10																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
40 ピンケーブル*1		抵抗器 (10KΩ) *1																																								
																																										
ブレッドボード*1		何本のジャンパー線																																								
																																										

原理

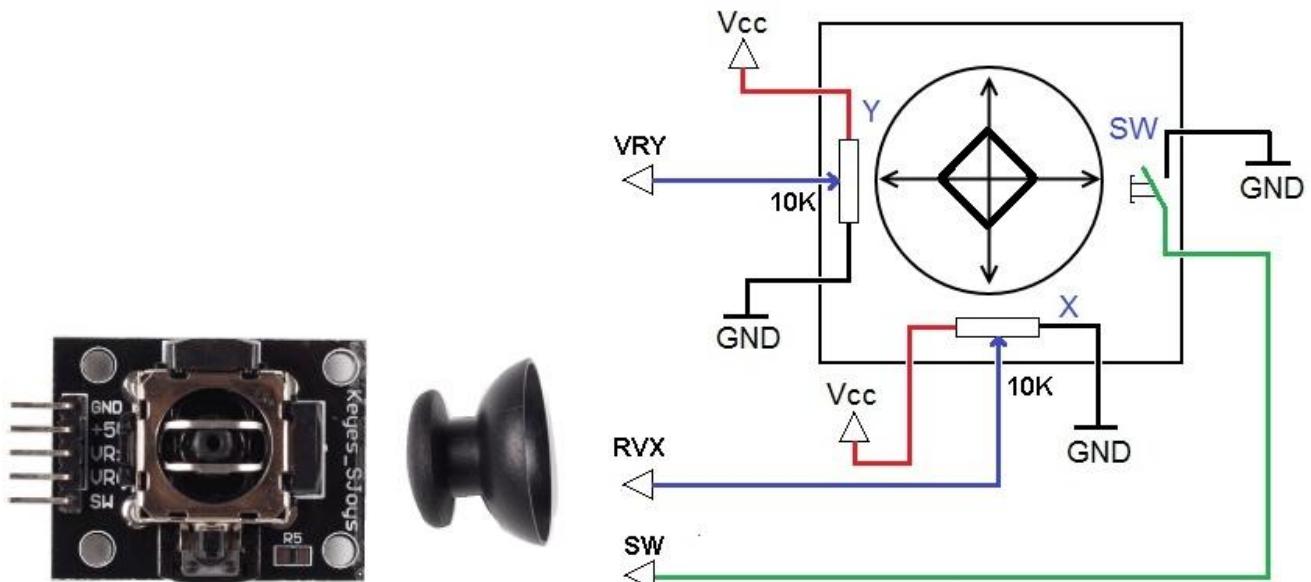
ジョイスティック

ジョイスティックの基本的な原理は、スティックの動きをコンピューターが処理できる電子情報に変換することである。

モーションの全範囲をコンピューターに通信するために、ジョイスティックは X 軸（左から右）と Y 軸（上下）の二つの軸でスティックの位置を測定する必要がある。基本的なジオメトリと同様に、X-Y 座標はスティックの位置を正確に特定する。

スティックの位置を決定するために、ジョイスティック制御システムは各シャフトの位置を監視する。従来のアナログジョイスティックの設計では、これを二つのポテンショメーターまたは可変抵抗器で行う。

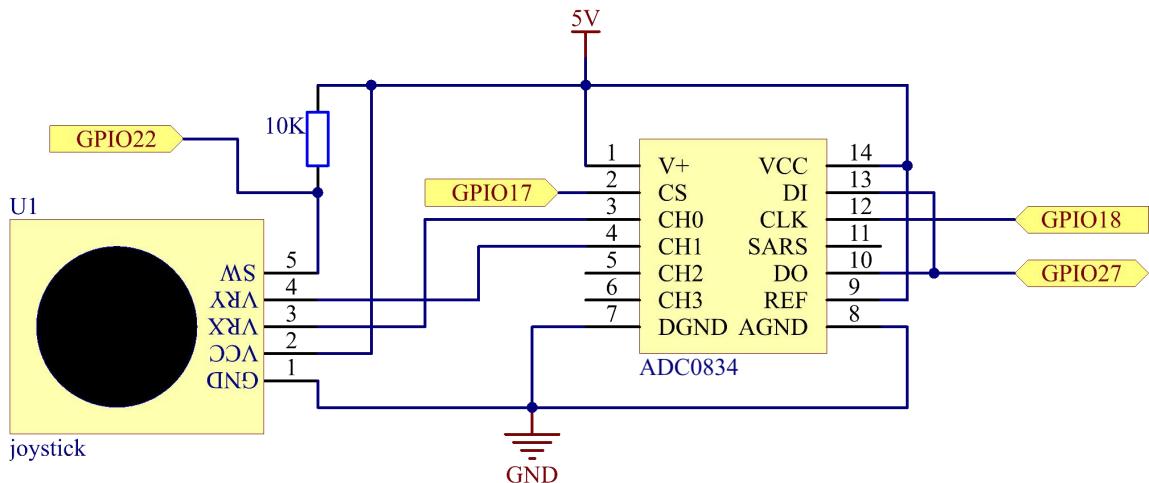
ジョイスティックには、ジョイスティックを押し下げたときに作動するデジタル入力もある。.



回路図

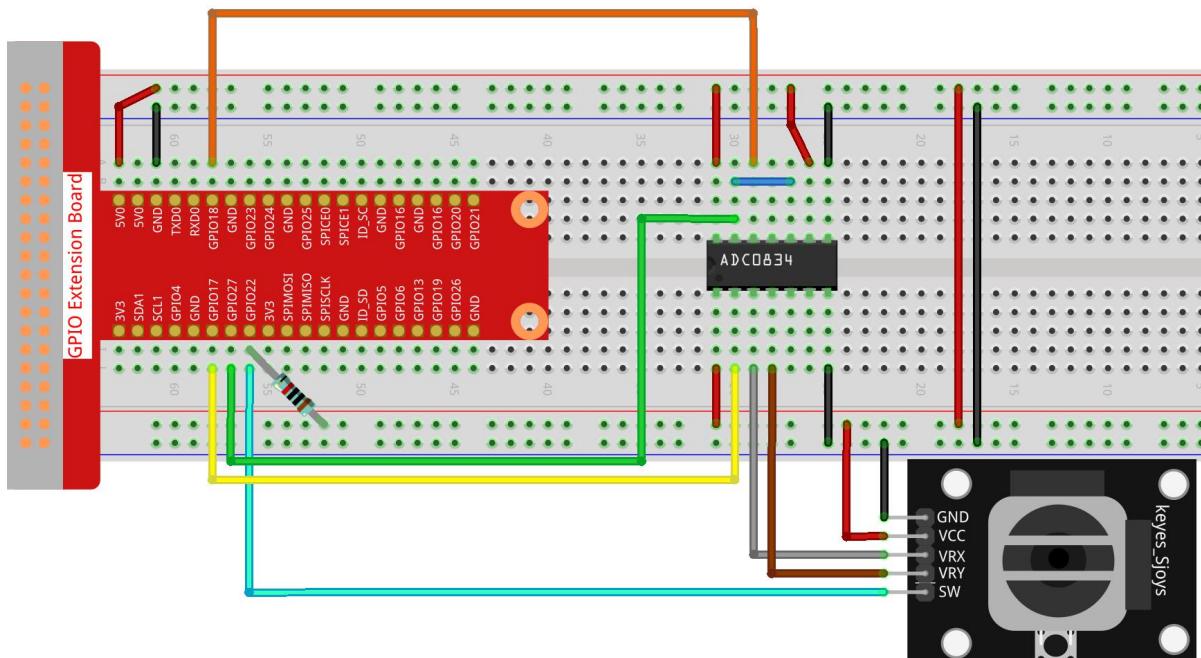
ジョイスティックのデータを読み取ると、軸間でいくつかの違いがある：X 軸と Y 軸のデータはアナログであり、ADC0834 を使用してアナログ値をデジタル値に変換する必要がある。Z 軸のデータはデジタルであるため、GPIO を直接使用して読み取るか、または ADC を使用して読み取ることができる。

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22



実験手順

ステップ1：回路を作る。



➤ C言語ユーザー向け

ステップ2：コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.6/
```

ステップ3：コードをコンパイルする。

```
gcc 2.1.6_Joystick.c -lwiringPi
```

ステップ4：EXEファイルを実行する。

```
sudo ./a.out
```

コードの実行後、ジョイスティックを回すと、対応する x、y、Btn の値が画面に表示される。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>

typedef unsigned char uchar;
typedef unsigned int uint;

#define ADC_CS    0
#define ADC_CLK   1
#define ADC_DIO   2
#define BtnPin   3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
```

```

for(i=0;i<8;i++)
{
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    pinMode(ADC_DIO, INPUT);
    dat1=dat1<<1 | digitalRead(ADC_DIO);
}
for(i=0;i<8;i++)
{
    dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
}
digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar x_val;
    uchar y_val;
    uchar btn_val;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(BtnPin, INPUT);
    pullUpDnControl(BtnPin, PUD_UP);
    pinMode(ADC_CS, OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        x_val = get_ADC_Result(0);
        y_val = get_ADC_Result(1);
        btn_val = digitalRead(BtnPin);
        printf("x = %d, y = %d, btn = %d\n", x_val, y_val, btn_val);
        delay(100);
    }
    return 0;
}

```

コードの説明

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    ....
```

機能の動作プロセスについては、2.1.4 ポテンショメーターで詳しく説明している。

```
while(1){
    x_val = get_ADC_Result(0);
    y_val = get_ADC_Result(1);
    btn_val = digitalRead(BtnPin);
    printf("x = %d, y = %d, btn = %d\n", x_val, y_val, btn_val);
    delay(100);
}
```

ジョイスティックの VRX と VRY は、それぞれ ADC0834 の CH0、CH1 に接続されている。したがって、関数 getResult() が呼び出されて、CH0 と CH1 の値が読み取られる。それから、読み取った値を変数 x_val と y_val に保存してください。さらに、ジョイスティックの SW の値を読み取り、変数 Btn_val に保存する。最後に、x_val、y_val、と Btn_val の値は print() 関数で出力される。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: 実行する。

```
sudo python3 2.1.6_Joystick.py
```

コードの実行後、ジョイスティックを回すと、対応する x、y、Btn の値が画面に表示される。

コード

```
#!/usr/bin/env python3

import RPi.GPIO as GPIO
import ADC0834
import time

BtnPin = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC0834.setup()

def destroy():
    # Release resource
    GPIO.cleanup()

def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)

if __name__ == '__main__':
    setup()
    try:
```

```

loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.

destroy()

```

コードの説明

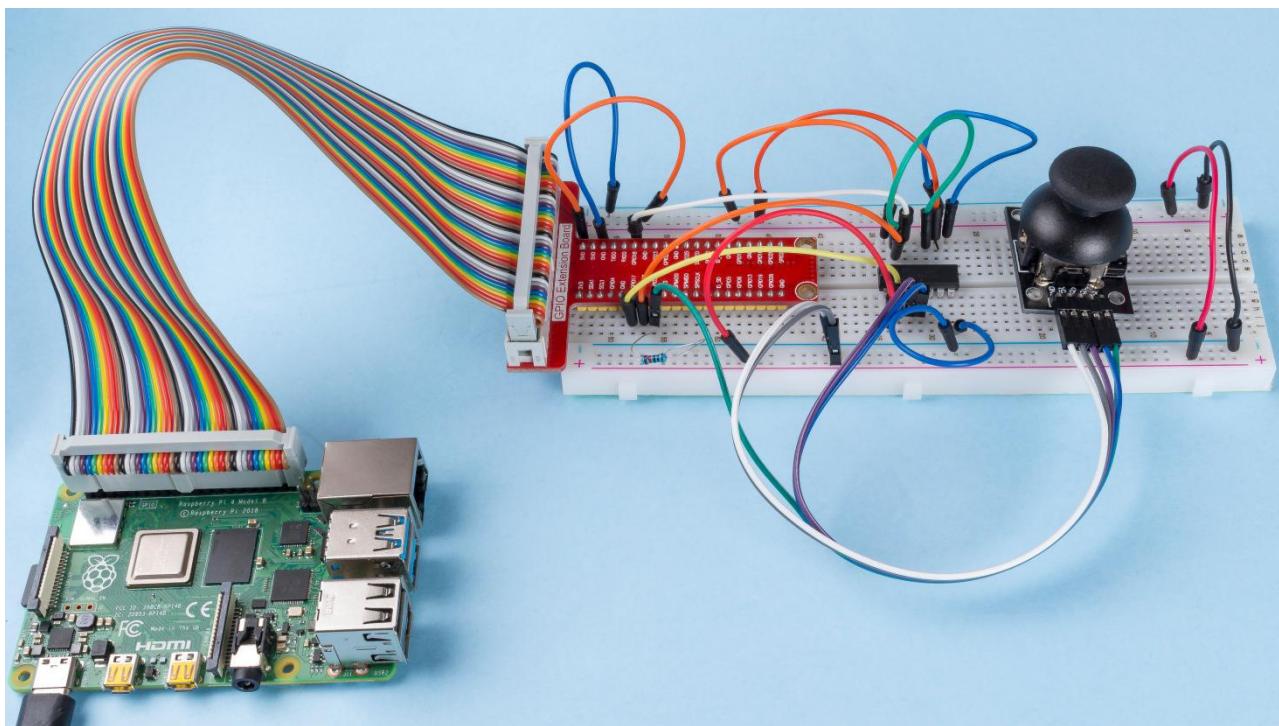
```

def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)

```

ジョイスティックの VRX と VRY は、それぞれ ADC0834 の CH0、CH1 に接続されている。したがって、関数 getResult () が呼び出されて、CH0 と CH1 の値が読み取られる。それから、読み取った値を変数 x_val と y_val に保存してください。さらに、ジョイスティックの SW の値を読み取り、変数 Btn_val に保存する。最後に、x_val、y_val、と Btn_val の値は print () 関数で出力される。

現象画像



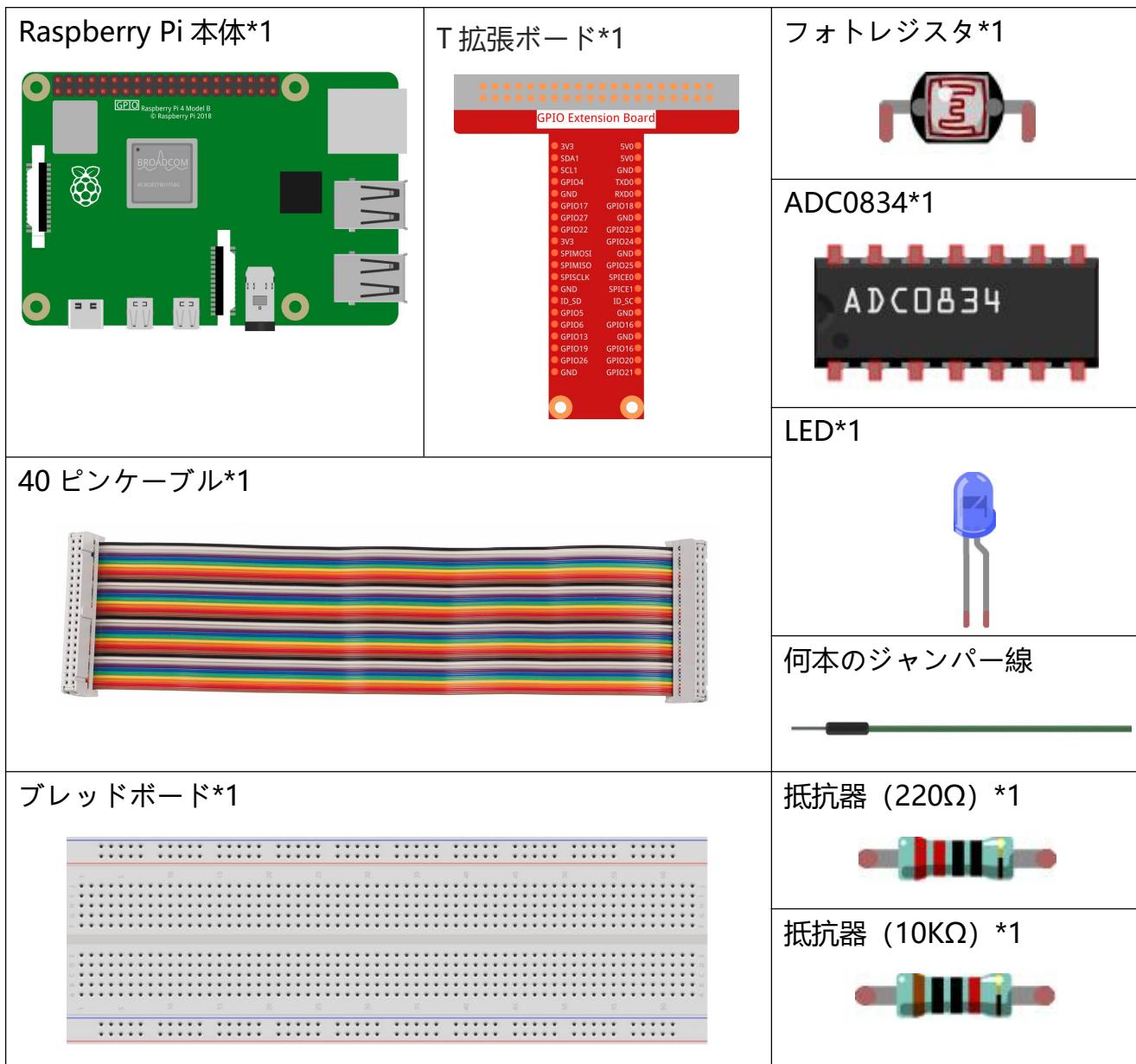
2.2 センサー

2.2.1 Photoresistor

前書き

フォトレジスタは生活の中で環境光の強度によく使用される部品である。コントローラーが昼と夜を認識し、夜間ランプなどの調光機能を実現することに役立つ。このプロジェクトはポテンショメータによく似ており、光を感じるための電圧を変えると思うかもしれない。

晶部



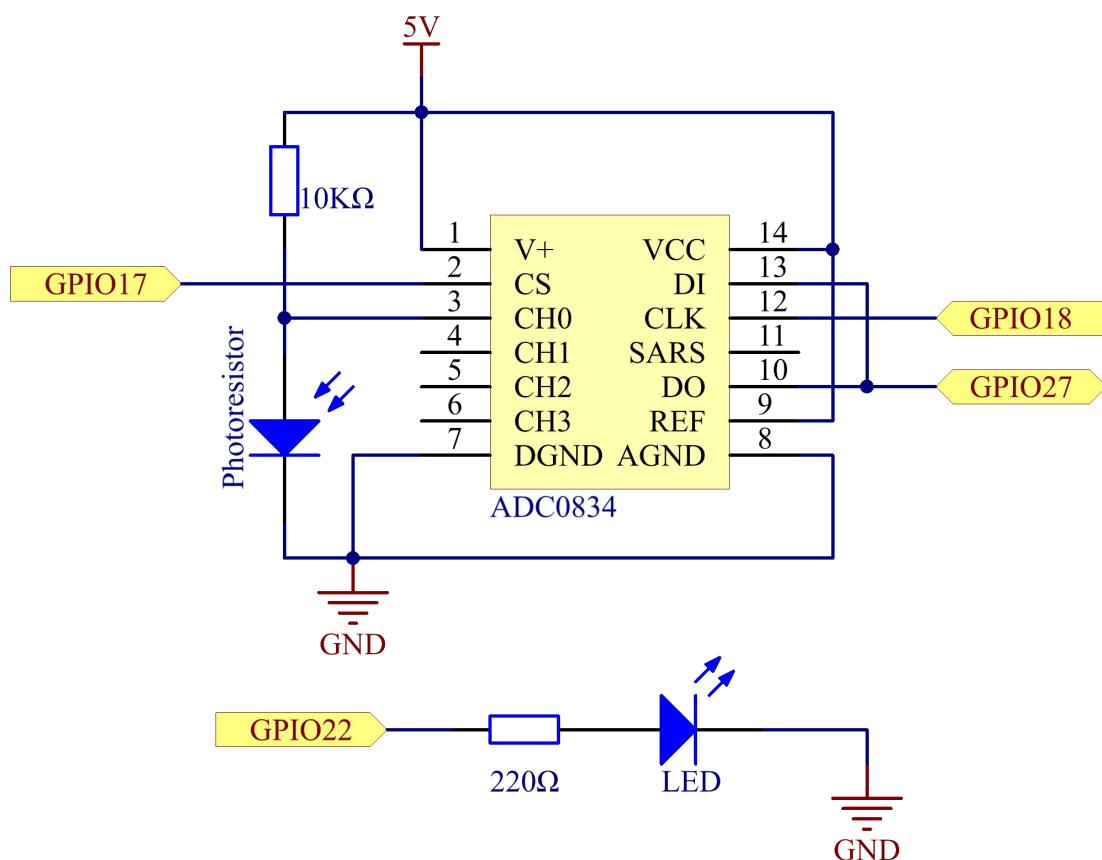
原理

フォトレジスタまたはフォトセルは光制御可変抵抗器である。フォトレジストの抵抗は入射光強度の増加とともに減少する。つまり、光伝導性を示す。フォトレジスタは、光に敏感な検出回路、および光・暗闇で作動する切換回路に適用できる。



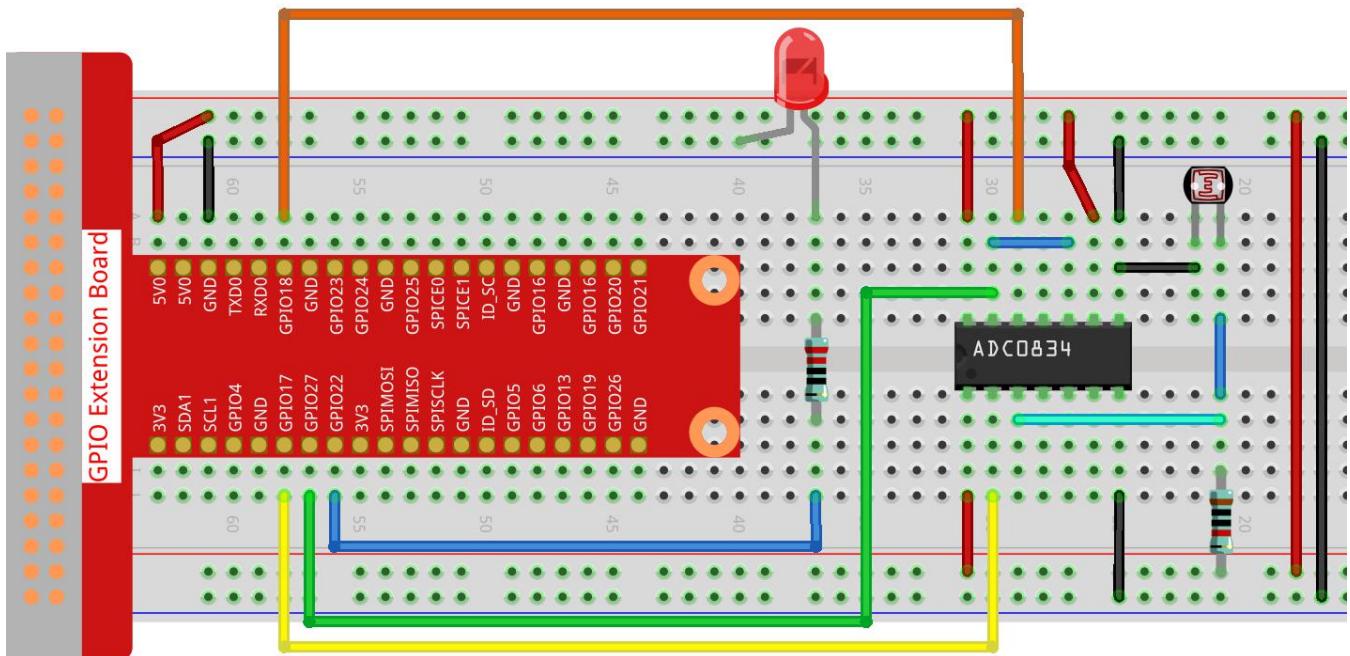
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin14	3	22



実験手順

ステップ 1: 回路を作る。



C言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.1/
```

ステップ 3: コードをコンパイルする。

```
gcc 2.2.1_Photoresistor.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードを実行すると、LED の輝度はフォトトレジスターが感知する光の強度に応じて変化する。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>

typedef unsigned char uchar;
typedef unsigned int uint;

#define ADC_CS 0
#define ADC_CLK 1
```

```
#define      ADC_DIO    2
#define      LedPin     3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }
}
```

```
}

for(i=0;i<8;i++)
{
    dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
}

digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(LedPin, 0, 100);
    pinMode(ADC_CS, OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

コードの説明

このコードは、2.1.4_Potentiometer.c のコードと同じである。他に質問がある場合は、2.1.4_Potentiometer.c のコード説明を参照してください。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 2.2.1_Photoresistor.py
```

コードを実行すると、LED の輝度はフォトトレジスターが感知する光の強度に応じて変化する。

コード

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import ADC0834
import time
LedPin = 22
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and freuece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)
    # Set all begin with value 0
    led_val.start(0)
def destroy():
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
if __name__ == '__main__':
    setup()
```

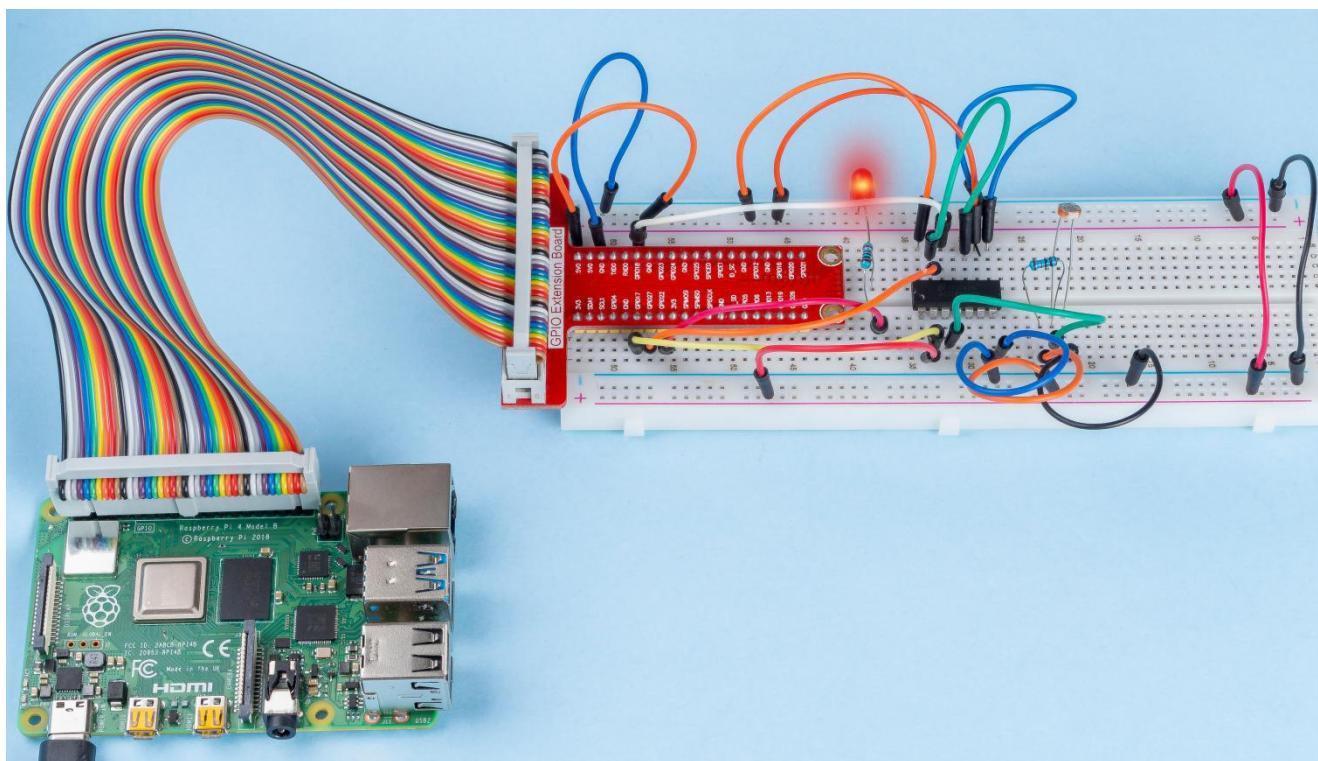
```
try:  
    loop()  
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be  
executed.  
    destroy()
```

コードの説明

```
def loop():  
    while True:  
        analogVal = ADC0834.getResult()  
        print ('analog value = %d' % analogVal)  
        led_val.ChangeDutyCycle(analogVal*100/255)  
        time.sleep(0.2)
```

ADC0834 の CH0 のアナログ値を読み取る。デフォルトでは、関数 getResult () を使用して CH0 の値を読み取る。したがって、他のチャネルを読み取る場合は、関数 getResult () の () に 1、2、または 3 を入力してください。次に、プリント機能を使用して値をプリントする必要がある。変化する要素は計算式である LedPin のデューティサイクルであるため、analogVal をパーセンテージに変換するには $analogVal * 100/255$ が必要である。最後に、ChangeDutyCycle () が呼び出されて、パーセンテージが LedPin に書き込まれる。

現象画像

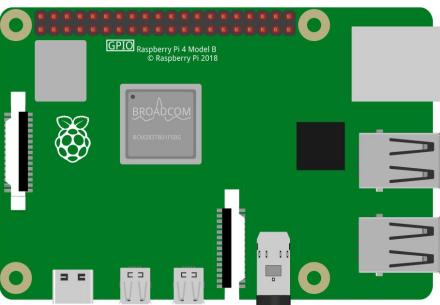
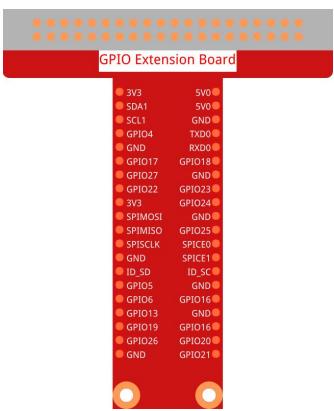
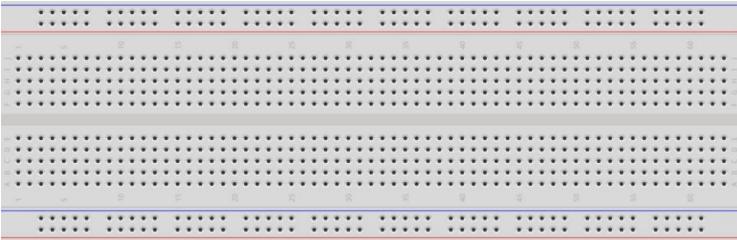


2.2.2 Thermistor

前書き

フォトレジスターが光を感知できるように、サーミスターは温度に敏感な電子デバイスであり、過熱警報装置の作成など、温度制御の機能を実現するために使用できる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	サーミスタ*1
	 GPIO Extension Board Pinout: ● 3V3 5V ● SDA1 GND ● SCL1 GND ● GPIO4 TXD0 ● GND RXD0 ● GPIO17 GPIO18 ● GND GND ● GPIO27 GPIO23 ● GND GPIO24 ● 3V3 GND ● SPIMISO GPIO25 ● SPIMSCLK SPICE0 ● GND SPICE1 ● ID_SD ID_SD ● GPIO5 GND ● GPIO6 GPIO16 ● GPIO13 GND ● GPIO19 GPIO16 ● GPIO26 GPIO20 ● GND GPIO21	
40 ピンケーブル*1		ADC0834*1
		
ブレッドボード*1		何本のジャンパー線
		
抵抗器 (10KΩ) *1		

原理

サーミスタは、温度のわずかな変化に比例して抵抗の予測可能な変化を正確に示す熱に敏感な抵抗器である。その抵抗がどの程度変化するかは、その独自の構成に依存する。サーミスタは、受動部品の大きなグループの一部である。また、対応するアクティブコンポーネントとは異なり、パッシブデバイスは回路に電力上昇または増幅を提供できない。

サーミスタは敏感な要素であり、負の温度係数 (NTC) と正の温度係数 (PTC) の 2 つのタイプがあり、NTC と PTC とも呼ばれる。その抵抗は温度によって大きく異なる。PTC サーミスタの抵抗は温度とともに増加するが、NTC の条件は前者とは逆である。この実験では、NTC を使用する。



その動作原理は、NTC サーミスタの抵抗が外部環境の温度とともに変化することである。環境のリアルタイム温度を検出できる。温度が高くなると、サーミスタの抵抗が減少する。次に、電圧データは A/D アダプターによってデジタル量に変換される。摂氏または華氏の温度はプログラミングにより出力される。

この実験では、サーミスタと 10k プルアップ抵抗が使用される。各サーミスタには通常の抵抗がある。ここでは、25°Cで測定されると、10k ohm である。

抵抗と温度の関係は次の通りである：

$$R_T = R_N \exp^{B(1/TK - 1/TN)}$$

RT は、温度が TK の場合の NTC サーミスタの抵抗です。

RN は、定格温度 TN での NTC サーミスタの抵抗です。ここで、RN の数値は 10k です。

TK はケルビン温度で、単位は K です。ここで、TK の数値は 273.15+摂氏です。

TN は、ケルビン温度の定格です。ここで、TN の数値は 273.15+25 です。

また、NTC サーミスタの材料定数である B (ベータ) は、数値 3950 の感熱指数とも呼ばれます。

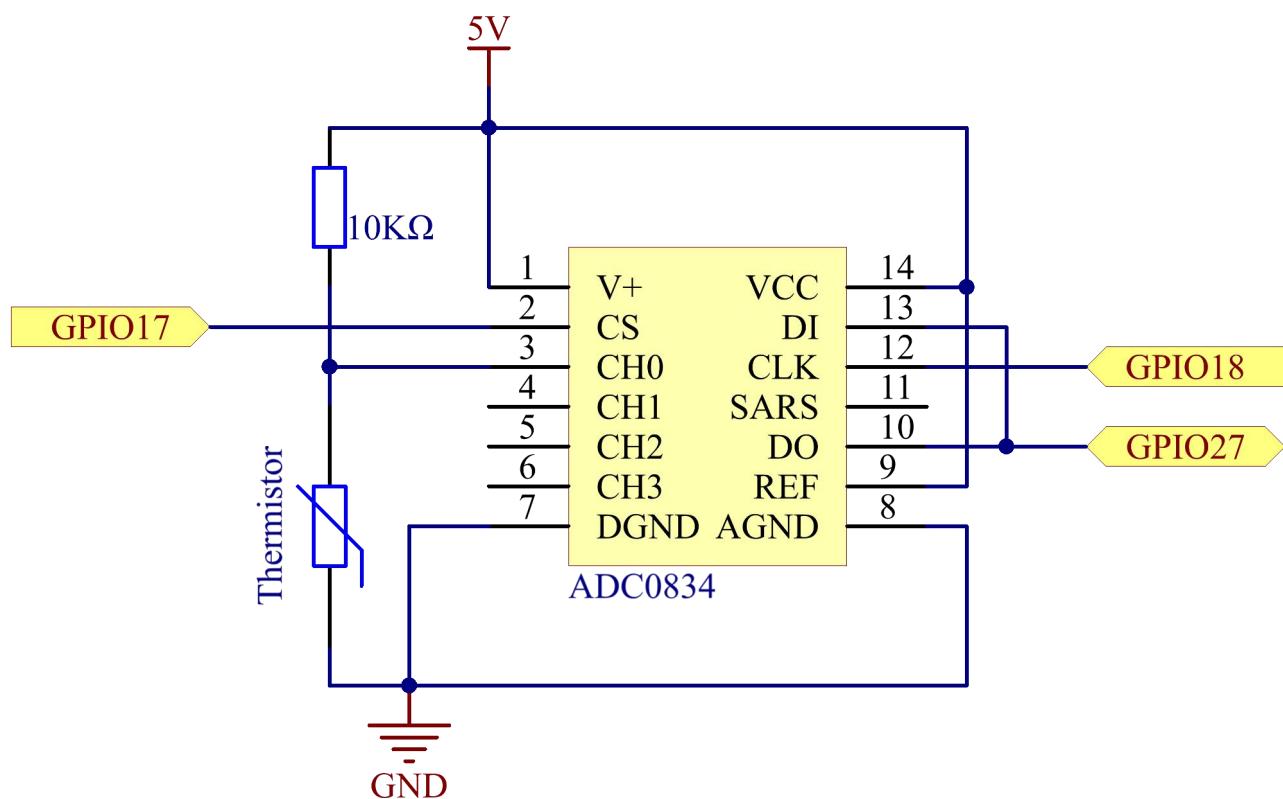
exp は指数の省略形であり、基底 e は自然数であり、およそ 2.7 です。

この式 $TK = 1 / (\ln(RT / RN) / B + 1 / TN)$ を変換して、ケルビン温度を取得します。ケルビンで 273.15 の減少は、摂氏温度です。

この関係は経験式です。温度と抵抗が有効範囲内にある場合のみ正確です。

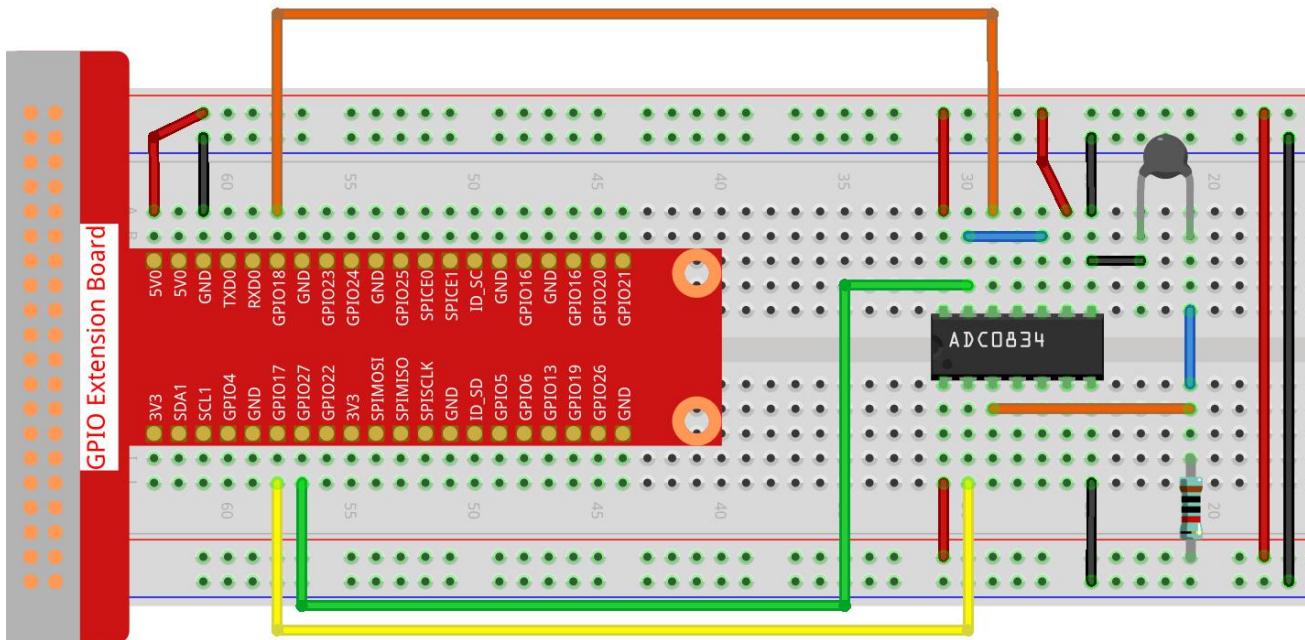
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.2/
```

ステップ3: コードをコンパイルする。

```
gcc 2.2.2_Thermistor.c -lwiringPi -lm
```

ご注意: -lm はライブラリの数学をロードする。省略すると、エラーが発生する。

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードを実行すると、サーミスタは周囲温度を検出する。周囲温度は、プログラムの計算が終了すると画面に出力される。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <math.h>

typedef unsigned char uchar;
typedef unsigned int uint;
```

```

#define      ADC_CS     0
#define      ADC_CLK    1
#define      ADC_DIO    2

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);

    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
    }
}

```

```

        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    unsigned char analogVal;
    double Vr, Rt, temp, cel, Fah;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        Vr = 5 * (double)(analogVal) / 255;
        Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
        temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
        cel = temp - 273.15;
        Fah = cel * 1.8 +32;
        printf("Celsius: %.2f C  Fahrenheit: %.2f F\n", cel, Fah);
        delay(100);
    }
    return 0;
}

```

コードの説明

```
#include <math.h>
```

一般的な数学的操作と変換を計算する一連の関数を宣言する C 数値ライブラリがある。

```
analogVal = get_ADC_Result(0);
```

この関数はサーミスタの値を読み取るために使用される。

```
Vr = 5 * (double)(analogVal) / 255;  
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));  
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));  
cel = temp - 273.15;  
Fah = cel * 1.8 +32;  
printf("Celsius: %.2f C  Fahrenheit: %.2f F\n", cel, Fah);
```

これらの計算により、サーミスタ値が摂氏値に変換される。

```
Vr = 5 * (double)(analogVal) / 255;  
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
```

Rt (サーミスタの抵抗) を取得するために、これらの 2 行のコードは読み取り値アナログを使って電圧分布を計算している。)

```
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
```

このコードは、Rt を式 $TK=1/(\ln(RT/RN)/B+1/TN)$ に挿入してケルビン温度を取得することを意味する。

```
temp = temp - 273.15;
```

ケルビン温度を摂氏に変換する。

```
Fah = cel * 1.8 +32;
```

摂氏を華氏に変換する。

```
printf("Celsius: %.2f C  Fahrenheit: %.2f F\n", cel, Fah);
```

ディスプレイに摂氏度、華氏度とそれらの単位を表示する。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 2.2.2_Termistor.py
```

コードを実行すると、サーミスタは周囲温度を検出する。周囲温度は、プログラムの計算が終了すると画面に出力される。

コード

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

import RPi.GPIO as GPIO
import ADC0834
import time
import math

def init():
    ADC0834.setup()

def loop():
    while True:
        analogVal = ADC0834.getResult()
        Vr = 5 * float(analogVal) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
        Cel = temp - 273.15
        Fah = Cel * 1.8 + 32
        print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
        time.sleep(0.2)

if __name__ == '__main__':
    init()
    try:
        loop()
    except KeyboardInterrupt:
        ADC0834.destroy()
```

コードの説明

```
import math
```

一般的な数学的操作と変換を計算する一連の関数を宣言する C 数値ライブラリがある。

```
analogVal = ADC0834.getResult()
```

この関数はサーミスタの値を読み取るために使用される。

```
Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
Cel = temp - 273.15
Fah = Cel * 1.8 + 32
print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
```

これらの計算はサーミスタの値を摂氏度と華氏度に変換する。

```
Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)
```

Rt (サーミスタの抵抗) を取得するために、これらの 2 行のコードは読み取り値アナログを使って電圧分布を計算している。

```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

このコードは、Rt を式 $TK=1/(\ln(RT/RN)/B+1/TN)$ に差し込んでケルビン温度を取得することを意味する。

```
temp = temp - 273.15
```

ケルビン温度を摂氏に変換する。

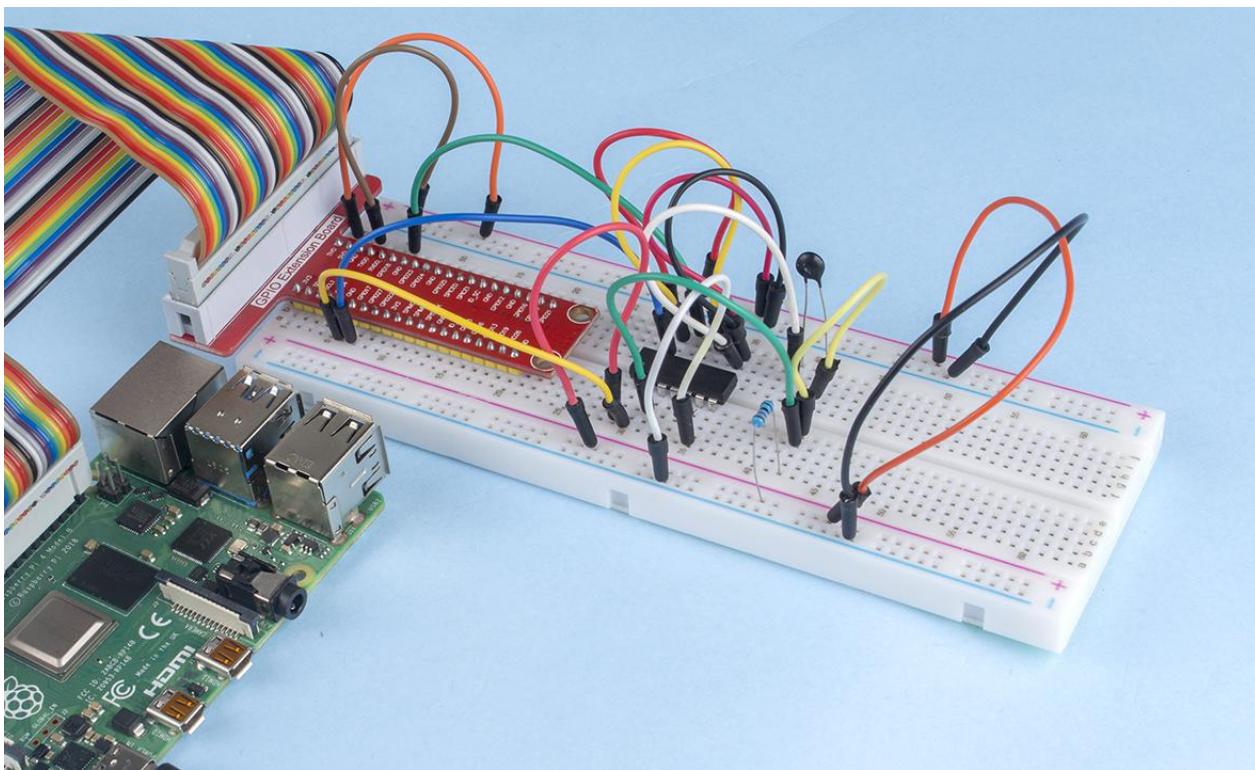
```
Fah = Cel * 1.8 + 32
```

摂氏を華氏に変換する。

```
print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
```

ディスプレイに摂氏度、華氏度とそれらの単位を表示する。

現象画像



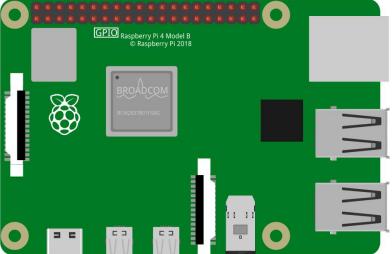
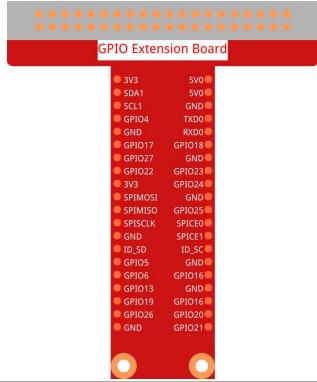
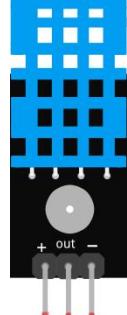
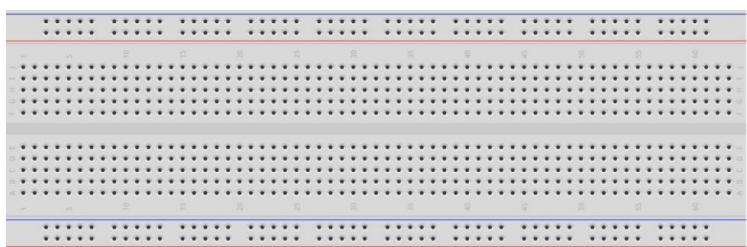
2.2.3 DHT-11

前書き

デジタル温湿度センサー DHT11 は、温度と湿度の校正済みデジタル信号出力を含む複合センサーである。専用のデジタルモジュールコレクションの技術と温湿度検知の技術を適用して、製品の高い信頼性と優れた安定性を確保している。

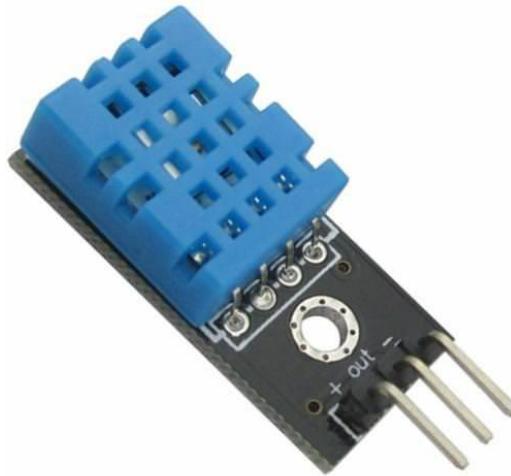
センサーには、湿式素子抵抗センサーと NTC 温度センサーが含まれており、高性能の 8 ビットマイクロコントローラーに接続されている。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	DHT-11*1																																										
	 <table border="1"> <thead> <tr> <th colspan="2">GPIO Extension Board</th> </tr> </thead> <tbody> <tr><td>● 3V3</td><td>5V0</td></tr> <tr><td>● SDA1</td><td>5V0</td></tr> <tr><td>● SCL1</td><td>GND</td></tr> <tr><td>● GPIO4</td><td>TXD0</td></tr> <tr><td>● GND</td><td>RXD0</td></tr> <tr><td>● GPIO17</td><td>GPIO18</td></tr> <tr><td>● GPIO27</td><td>GND</td></tr> <tr><td>● GPIO22</td><td>GPIO23</td></tr> <tr><td>● 3V3</td><td>GPIO24</td></tr> <tr><td>● SPIMOSI</td><td>GND</td></tr> <tr><td>● SPIMISO</td><td>GPIO25</td></tr> <tr><td>● SPICLK</td><td>SPICE1</td></tr> <tr><td>● GND</td><td>SPICE1</td></tr> <tr><td>● ID_SD</td><td>ID_SD</td></tr> <tr><td>● GPIO5</td><td>GND</td></tr> <tr><td>● GPIO6</td><td>GPIO16</td></tr> <tr><td>● GPIO13</td><td>GND</td></tr> <tr><td>● GPIO19</td><td>GPIO16</td></tr> <tr><td>● GPIO26</td><td>GPIO20</td></tr> <tr><td>● GND</td><td>GPIO21</td></tr> </tbody> </table>	GPIO Extension Board		● 3V3	5V0	● SDA1	5V0	● SCL1	GND	● GPIO4	TXD0	● GND	RXD0	● GPIO17	GPIO18	● GPIO27	GND	● GPIO22	GPIO23	● 3V3	GPIO24	● SPIMOSI	GND	● SPIMISO	GPIO25	● SPICLK	SPICE1	● GND	SPICE1	● ID_SD	ID_SD	● GPIO5	GND	● GPIO6	GPIO16	● GPIO13	GND	● GPIO19	GPIO16	● GPIO26	GPIO20	● GND	GPIO21	
GPIO Extension Board																																												
● 3V3	5V0																																											
● SDA1	5V0																																											
● SCL1	GND																																											
● GPIO4	TXD0																																											
● GND	RXD0																																											
● GPIO17	GPIO18																																											
● GPIO27	GND																																											
● GPIO22	GPIO23																																											
● 3V3	GPIO24																																											
● SPIMOSI	GND																																											
● SPIMISO	GPIO25																																											
● SPICLK	SPICE1																																											
● GND	SPICE1																																											
● ID_SD	ID_SD																																											
● GPIO5	GND																																											
● GPIO6	GPIO16																																											
● GPIO13	GND																																											
● GPIO19	GPIO16																																											
● GPIO26	GPIO20																																											
● GND	GPIO21																																											
40 ピンケーブル*1		何本のジャンパー線																																										
																																												
ブレッドボード*1		抵抗器 (10KΩ) *1																																										
																																												

原理

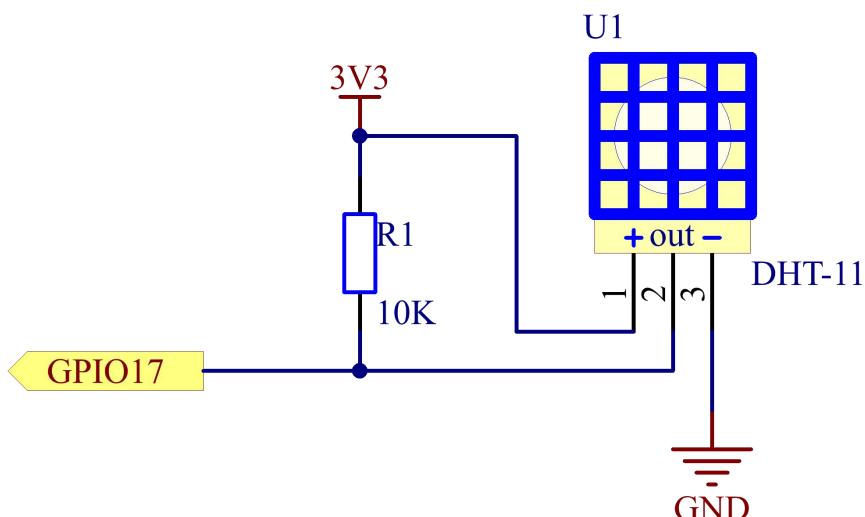
DHT11 は基本的な超低コストのデジタル温湿度センサーである。容量性湿度センサーとサーミスタを使用して周囲の空気を測定し、データピンにデジタル信号を出力する（アナログ入力ピンは不要）。



VCC、GND、と DATA の三つのピンのみが利用できる。通信プロセスは開始信号を DHT11 に送信する DATA ラインから始まり、DHT11 は信号を受信して応答信号を返す。次に、ホストは応答信号を受信し、40 ビットの湿度データ（8 ビット湿度整数 + 8 ビット湿度 10 進数 + 8 ビット湿度整数 + 8 ビット湿度 10 進数 + 8 ビットチェックサム）の受信を開始する。詳細については、DHT11 データシートを参照してください。

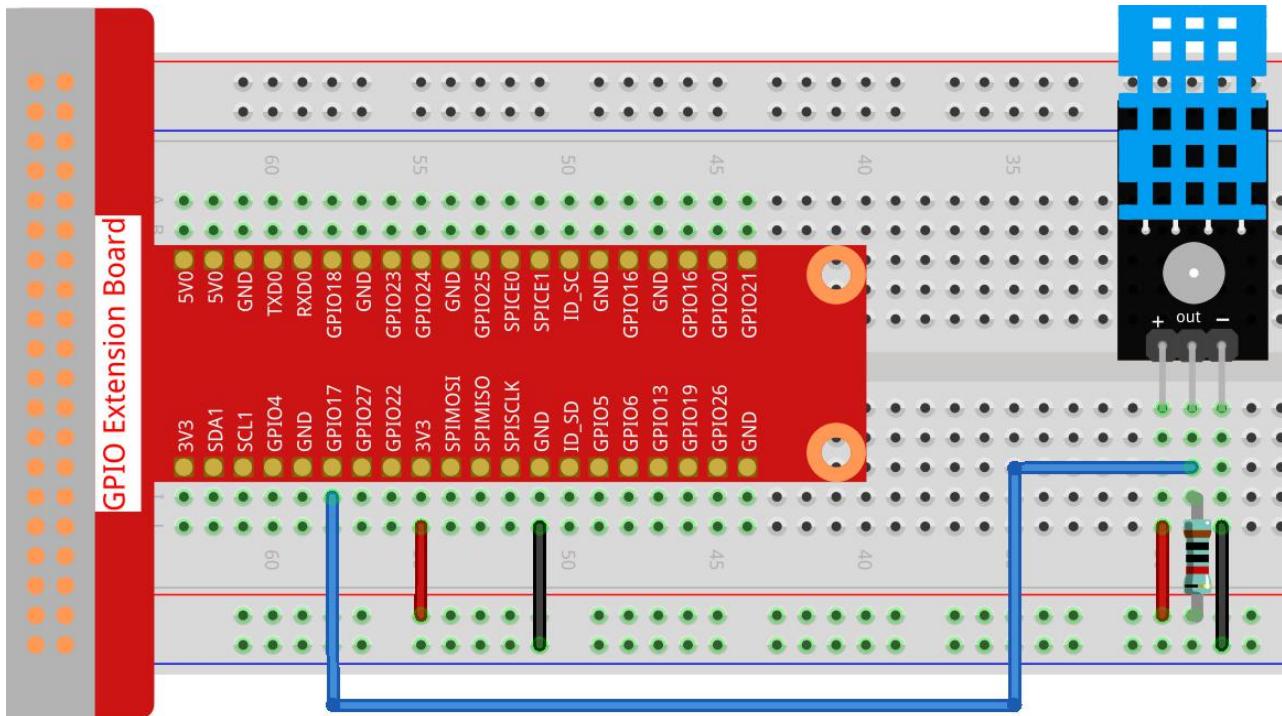
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.3/
```

ステップ3: コードをコンパイルする。

```
gcc 2.2.3_DHT.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、プログラムは DHT11 によって検出された温度と湿度をコンピューター画面にプリントする。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define maxTim 85
#define dhtPin 0
```

```

int dht11_dat[5] = {0,0,0,0,0};

void readDht11() {
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit

    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // pull pin down for 18 milliseconds
    pinMode(dhtPin, OUTPUT);
    digitalWrite(dhtPin, LOW);
    delay(18);
    // then pull it up for 40 microseconds
    digitalWrite(dhtPin, HIGH);
    delayMicroseconds(40);
    // prepare to read the pin
    pinMode(dhtPin, INPUT);

    // detect change and read data
    for (i=0; i< maxTim; i++) {
        counter = 0;
        while (digitalRead(dhtPin) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 255) {
                break;
            }
        }
        laststate = digitalRead(dhtPin);

        if (counter == 255) break;
        // ignore first 3 transitions
        if ((i >= 4) && (i%2 == 0)) {
            // shove each bit into the storage bytes
            dht11_dat[j/8] <<= 1;
            if (counter > 50)
                dht11_dat[j/8] |= 1;
            j++;
        }
    }
}

```

```

}

// check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
// print it out if data is good
if ((j >= 40) &&
    (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] + dht11_dat[3])  

& 0xFF)) ) {
    Fah = dht11_dat[2] * 9. / 5. + 32;
    printf("Humidity = %d.%d %% Temperature = %d.%d *C (%.1f *F)\n",
           dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], Fah);
}
}

int main (void) {
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    while (1) {
        readDht11();
        delay(500); // wait 1sec to refresh
    }
    return 0 ;
}

```

コードの説明

```

void readDht11() {
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit
    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // ...
}

```

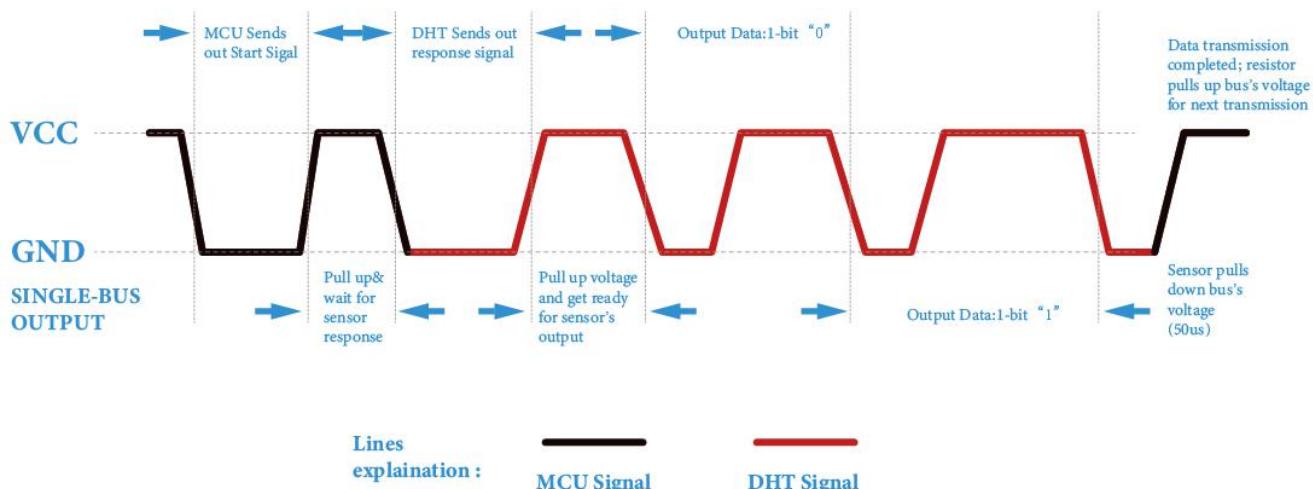
この機能は DHT11 の機能を実現するために使用される。

通常、次の 3 つの部分に分けることができる：

1. ピンを読む準備ができた：

```
// pull pin down for 18 milliseconds
pinMode(dhtPin, OUTPUT);
digitalWrite(dhtPin, LOW);
delay(18);
// then pull it up for 40 microseconds
digitalWrite(dhtPin, HIGH);
delayMicroseconds(40);
// prepare to read the pin
pinMode(dhtPin, INPUT);
```

その通信フローは、作業のタイミングによって決まる。



DHT11 が起動すると、MCU は低レベルの信号を送信し、40us の間信号を高レベルに保つ。その後、外部環境の状態の検出が開始される。

2. データの読み取り：

```
// detect change and read data
for ( i=0; i< maxTim; i++) {
    counter = 0;
    while (digitalRead(dhtPin) == laststate) {
        counter++;
        delayMicroseconds(1);
        if (counter == 255) {
            break;
        }
    }
    laststate = digitalRead(dhtPin);
    if (counter == 255) break;
    // ignore first 3 transitions
```

```

if ((i >= 4) && (i%2 == 0)) {
    // shove each bit into the storage bytes
    dht11_dat[j/8] <<= 1;
    if (counter > 50)
        dht11_dat[j/8] |= 1;
    j++;
}
}

```

ループは検出されたデータを dht11_dat[] 配列に保存する。DHT11 は一度に 40 ビットのデータを転送する。最初の 16 ビットは湿度に関連し、中央の 16 ビットは温度に関連し、最後の 8 ビットは検証に使用される。データ形式は次のとおりである：

8bit humidity integer data + 8bit humidity decimal data + 8bit temperature integer data + 8bit temperature decimal data + 8bit check bit.

3. 湿度と温度をプリントする。

```

// check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
// print it out if data is good
if ((j >= 40) &&
    (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] + dht11_dat[3])
& 0xFF))) {
    Fah = dht11_dat[2] * 9. / 5. + 32;
    printf("Humidity = %d.%d %% Temperature = %d.%d *C (%.1f *F)\n",
           dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], Fah);
}
}

```

データストレージが最大 40 ビットの場合、check bit (dht11_dat[4])を通じてデータの有効性をチェックし、温度と湿度をプリントする。

たとえば、受信データが 00101011 (湿度整数の 8 ビット値) 00000000 (湿度 10 進数の 8 ビット値) 00111100 (温度整数の 8 ビット値) 00000000 (温度 10 進数の 8 ビット値) 01100111 (チェックビット)。

計算：

$$00101011 + 00000000 + 00111100 + 00000000 = 01100111.$$

最終結果はチェックビットデータに等しく、受信データは正しいである：

湿度 = 43%、温度 = 60 * C。

チェックビットデータと等しくない場合、データ送信は正常ではなく、データが再度受信される。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 2.2.3_DHT.py
```

コードの実行後、プログラムは DHT11 によって検出された温度と湿度をコンピューター画面にプリントする。

コード

```
import RPi.GPIO as GPIO
import time

dhtPin = 17

GPIO.setmode(GPIO.BCM)

MAX_UNCHANGE_COUNT = 100

STATE_INIT_PULL_DOWN = 1
STATE_INIT_PULL_UP = 2
STATE_DATA_FIRST_PULL_DOWN = 3
STATE_DATA_PULL_UP = 4
STATE_DATA_PULL_DOWN = 5

def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)

    unchanged_count = 0
    last = -1
    data = []
```

```
while True:  
    current = GPIO.input(dhtPin)  
    data.append(current)  
    if last != current:  
        unchanged_count = 0  
        last = current  
    else:  
        unchanged_count += 1  
        if unchanged_count > MAX_UNCHANGE_COUNT:  
            break  
  
state = STATE_INIT_PULL_DOWN  
  
lengths = []  
current_length = 0  
  
for current in data:  
    current_length += 1  
  
    if state == STATE_INIT_PULL_DOWN:  
        if current == GPIO.LOW:  
            state = STATE_INIT_PULL_UP  
        else:  
            continue  
    if state == STATE_INIT_PULL_UP:  
        if current == GPIO.HIGH:  
            state = STATE_DATA_FIRST_PULL_DOWN  
        else:  
            continue  
    if state == STATE_DATA_FIRST_PULL_DOWN:  
        if current == GPIO.LOW:  
            state = STATE_DATA_PULL_UP  
        else:  
            continue  
    if state == STATE_DATA_PULL_UP:  
        if current == GPIO.HIGH:  
            current_length = 0  
            state = STATE_DATA_PULL_DOWN  
        else:  
            continue
```

```

if state == STATE_DATA_PULL_DOWN:
    if current == GPIO.LOW:
        lengths.append(current_length)
        state = STATE_DATA_PULL_UP
    else:
        continue
if len(lengths) != 40:
    #print ("Data not good, skip")
    return False

shortest_pull_up = min(lengths)
longest_pull_up = max(lengths)
halfway = (longest_pull_up + shortest_pull_up) / 2
bits = []
the_bytes = []
byte = 0

for length in lengths:
    bit = 0
    if length > halfway:
        bit = 1
    bits.append(bit)
#print ("bits: %s, length: %d" % (bits, len(bits)))
for i in range(0, len(bits)):
    byte = byte << 1
    if (bits[i]):
        byte = byte | 1
    else:
        byte = byte | 0
    if ((i + 1) % 8 == 0):
        the_bytes.append(byte)
        byte = 0
#print (the_bytes)
checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
if the_bytes[4] != checksum:
    #print ("Data not good, skip")
    return False

return the_bytes[0], the_bytes[2]

```

```
def main():

    while True:
        result = readDht11()
        if result:
            humidity, temperature = result
            print ("humidity: %s %%," "Temperature: %s C`" % (humidity, temperature))
            time.sleep(1)

    def destroy():
        GPIO.cleanup()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)
    unchanged_count = 0
    last = -1
    data = []
    #...
```

この関数は DHT11 の関数を実装するために使用される。それは検出された

データを the_bytes []配列に保存する。DHT11 は一度に 40 ビットのデータを点灯する。最初の 16 ビットは湿度に関連し、中央の 16 ビットは温度に関連し、最後の 8 ビットは検証に使用される。データ形式は次のとおりである：

8bit humidity integer data +8bit humidity decimal data +8bit temperature integer data + 8bit temperature decimal data + 8bit check bit.

チェックビットを介して有効性が検出されると、関数は 2 つの結果を返す：1. エラー；2. 湿度と温度。

```
checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
if the_bytes[4] != checksum:
    #print ("Data not good, skip")
    return False

return the_bytes[0], the_bytes[2]
```

たとえば、受信した日付が 00101011（湿度整数の 8 ビット値）

00000000（湿度 10 進数の 8 ビット値） 00111100（温度整数の 8 ビット値） 00000000
(温度 10 進数の 8 ビット値) 01100111（チェックビット）。

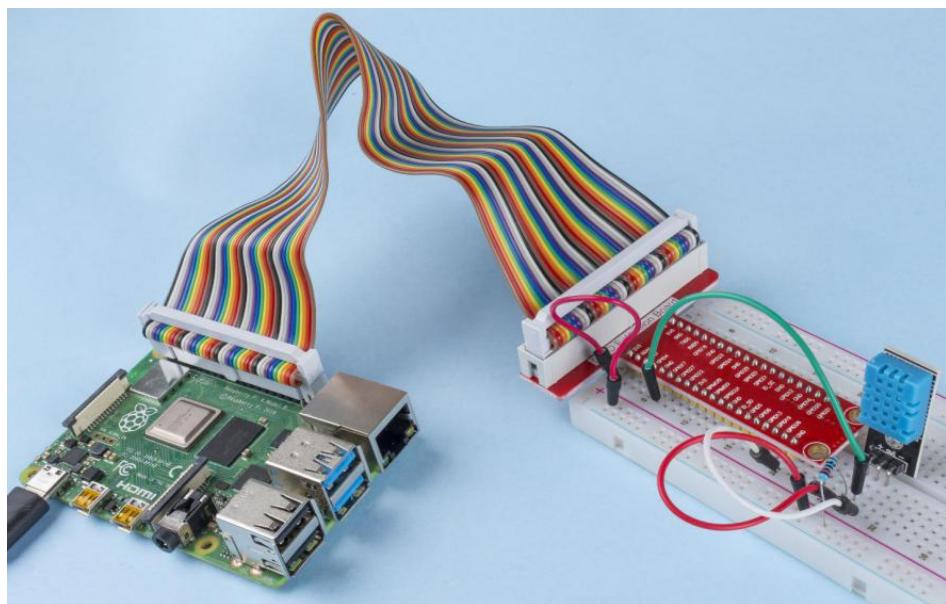
計算：

$00101011 + 00000000 + 00111100 + 00000000 = 01100111.$

最終結果がチェックビットデータと等しい場合、データ送信は異常である：False を返す。

最終結果がチェックビットデータと等しく、受信データは正しい場合、the_bytes[0] と the_bytes[2] が返され、「Humidity = 43%、Temperature= 60C」が outputされる。

現象画像

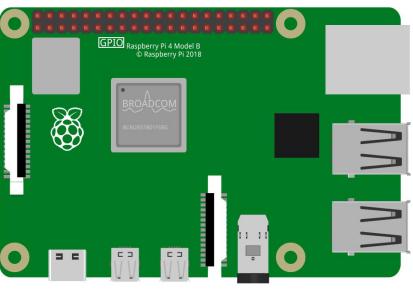
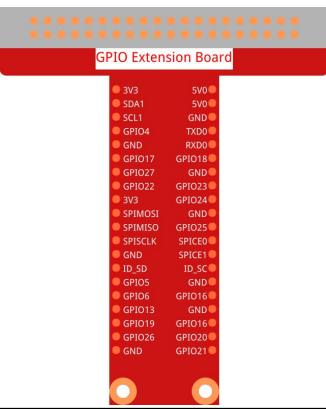
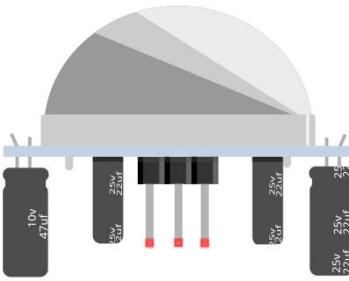
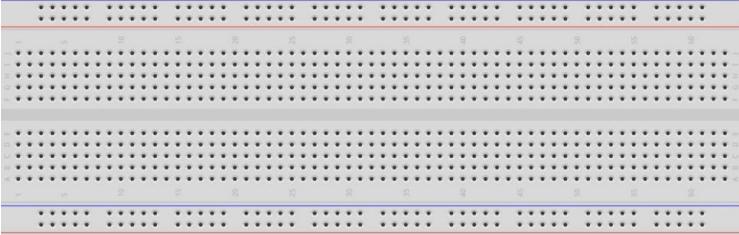


2.2.4 PIR

前書き

このプロジェクトでは、人体の赤外線焦電センサーを使用してデバイスを作成する。誰かが LED に近づくと、LED は自動的に点灯する。そうでない場合、ライトは消灯する。この赤外線モーションセンサーは、人間や動物が発する赤外線を検出できるセンサーの一種である。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	PIR センサーモジュール*1
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">3V3SDA1SCL1GPIO4GNDGPIO17GPIO27GPIO223V3SPI MOSISPI MISOSPI SCLKGNDID_SDGPIO5GPIO6GPIO13GPIO19GPIO26GND5V0GNDTXD0RXD0GPIO18GPIO23GPIO24GNDGPIO25GPIO26GPIO27GPIO16GPIO16GPIO20GPIO21	
40 ピンケーブル*1	何本のジャンパー線	
		
ブレッドボード*1	抵抗器 (220Ω) *3	RGB LED*1
		

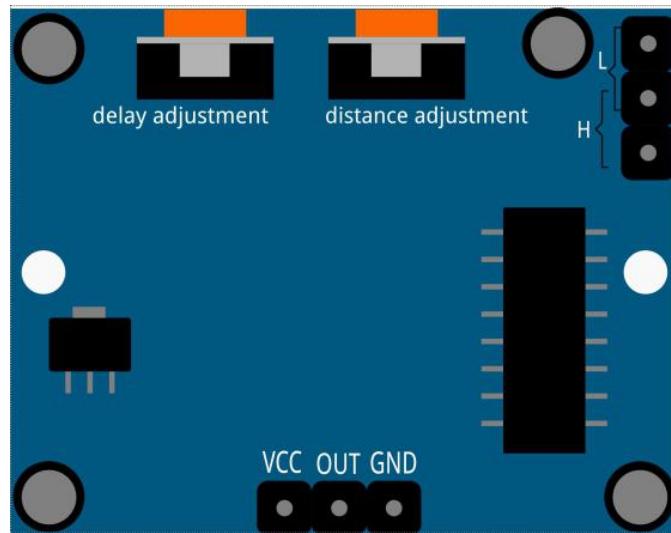
原理

PIR センサーは、赤外線熱放射を検出し、赤外線熱放射を放出する生物の存在を検出するために使用できる。

PIR センサーは差動増幅器に接続される 2 つのスロットに分割される。静止物体がセンサーの前にあるときはいつでも、二つのスロットは同じ量の放射線を受け取り、出力はゼロである。動いている物体がセンサーの前にあるときはいつでも、スロットの一つが他のスロットよりも多くの放射線を受け取り、出力を上下に変動させる。この出力電圧の変化は、動きの検出の結果である。



検知モジュールの配線後、1 分間の初期化が行われる。初期化中に、モジュールは間隔を置いて 0~3 回出力する。その後、モジュールはスタンバイモードになる。干渉信号によって引き起こされる誤動作を避けるために、光源と他の源の干渉をモジュールの表面から遠ざけてください。風はセンサーにも干渉する可能性があるため、風があまり無くてモジュールを使用することをお勧めする。



距離調整

距離調整ポテンショメータのノブを時計回りに回すと、検知距離の範囲が広がり、最大検知距離範囲は約 0~7 メートルである。反時計回りに回すと、検知距離の範囲が狭くなり、最小検知距離の範囲は約 0~3 メートルである。

遅延調整

遅延調整ポテンショメーターのノブを時計回りに回すと、検知遅延が増加することも分か

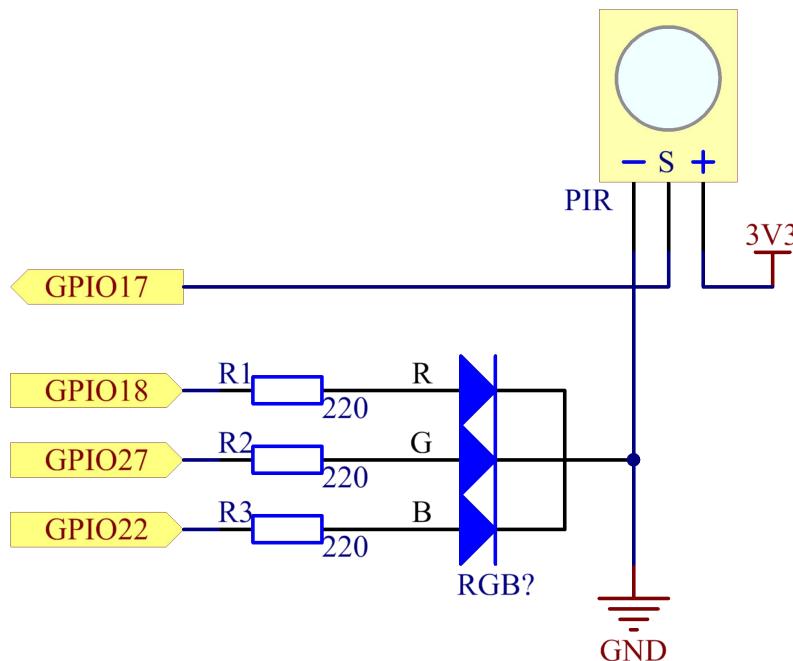
る。検知遅延の最大値は、最大 300 秒に達する可能性がある。逆に、反時計回りに回転させると、最小 5 秒で遅延を短縮できる。

二つのトリガーモード：（ジャンパークリップを使用して異なるモードを選択する）。

- ◆ **H**: 反復可能なトリガーモード。人体を検知した後、モジュールは高レベルを出力する。後続の遅延期間中に、誰かが検知範囲に入ると、出力は高レベルのままになる。
- ◆ **L**: 反復不可能なトリガーモード。人体を感知すると高レベルを出力する。遅延後、出力は自動的に高レベルから低レベルに変わる。

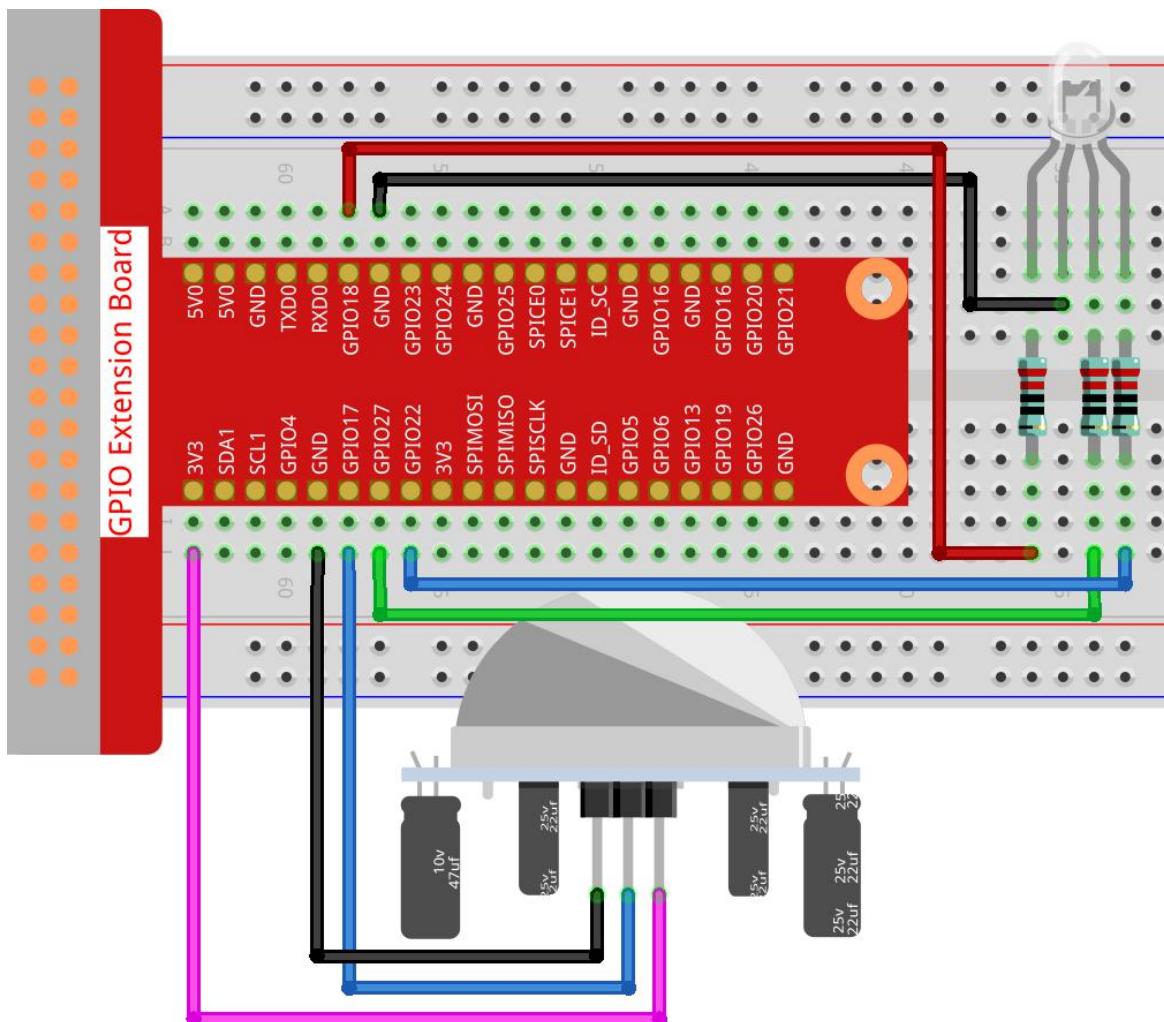
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



実験手順

ステップ 1: 回路を作る。



➤ C言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.4/
```

ステップ 3: コードをコンパイルする。

```
gcc 2.2.4_PIR.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、PIR は周囲の状況を検出し、誰かが歩いていることを感知すると、RGB LED が黄色く光るようにする。PIR モジュールには 2 つのポテンショメーターがある：1 つは感度を調整するためのもので、もう 1 つは検出距離を調整するためのものである。PIR モジュールの動作を改善するために、これら 2 つのポテンショメーターを調整する必要がある。

コード

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define uchar unsigned char

#define pirPin    0      //the pir connect to GPIO0
#define redPin    1
#define greenPin  2
#define bluePin   3

void ledInit(void){
    softPwmCreate(redPin,  0, 100);
    softPwmCreate(greenPin,0, 100);
    softPwmCreate(bluePin, 0, 100);
}

void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(redPin,  r_val);
    softPwmWrite(greenPin, g_val);
    softPwmWrite(bluePin, b_val);
}

int main(void)
{
    int pir_val;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    ledInit();
    pinMode(pirPin, INPUT);
    while(1){
        pir_val = digitalRead(pirPin);
        if(pir_val== 1){ //if read pir is HIGH level
            ledColorSet(0xff,0xff,0x00);
        }
        else {
            ledColorSet(0x00,0x00,0xff);
        }
    }
    return 0;
}
```

コードの説明

```
void ledInit(void);
void ledColorSet(uchar r_val, uchar g_val, uchar b_val);
```

これらのコードは RGB LED の色を設定するために使用され、詳しくは 1.1.2-RGB LED を参照してください。

```
int main(void)
{
    int pir_val;
    //.....
    pinMode(pirPin, INPUT);
    while(1){
        pir_val = digitalRead(pirPin);
        if(pir_val== 1){ //if read pir is HIGH level
            ledColorSet(0xff,0xff,0x00);
        }
        else {
            ledColorSet(0x00,0x00,0xff);
        }
    }
    return 0;
}
```

PIR が人間の赤外線スペクトルを検出すると、RGB LED が黄色の光を発する。そうでない場合は、青色の光を発する。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 2.2.4_PIR.py
```

コードの実行後、PIR は周囲の状況を検出し、誰かが歩いていることを感知すると、RGB LED が黄色く光るようにする。PIR モジュールには 2 つのポテンショメーターがある：1 つは感度を調整するためのもので、もう 1 つは検出距離を調整するためのものである。PIR モジュールの動作を改善するために、これら 2 つのポテンショメーターを調整する必要がある。

コード

```
import RPi.GPIO as GPIO
import time

rgbPins = {'Red':18, 'Green':27, 'Blue':22}
pirPin = 17      # the pir connect to pin17

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)      # Set the GPIO modes to BCM Numbering
    GPIO.setup(pirPin, GPIO.IN)  # Set pirPin to input
    # Set all LedPin's mode to output and initial level to High(3.3v)
    for i in rgbPins:
        GPIO.setup(rgbPins[i], GPIO.OUT, initial=GPIO.HIGH)

    # Set all led as pwm channel and frequece to 2KHz
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)

    # Set all begin with value 0
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

# Define a MAP function for mapping values. Like from 0~255 to 0~100
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

# Define a function to set up colors
def setColor(color):
    # configures the three LEDs' luminance with the inputted color value .
    # Devide colors from 'color' veriable
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0
    # Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

    #Assign the mapped duty cycle value to the corresponding PWM channel to change
    #the luminance.
```

```

p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
#print ("color_msg: R_val = %s, G_val = %s, B_val = %s"%(R_val, G_val, B_val))

def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)

def destroy():
    p_R.stop()
    p_G.stop()
    p_B.stop()
    GPIO.cleanup()           # Release resource

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy()
will be executed.
        destroy()

```

コードの説明

```

rgbPins = {'Red':18, 'Green':27, 'Blue':22}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    # .....
    for i in rgbPins:
        GPIO.setup(rgbPins[i], GPIO.OUT, initial=GPIO.HIGH)
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

```

```
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def setColor(color):
    ...

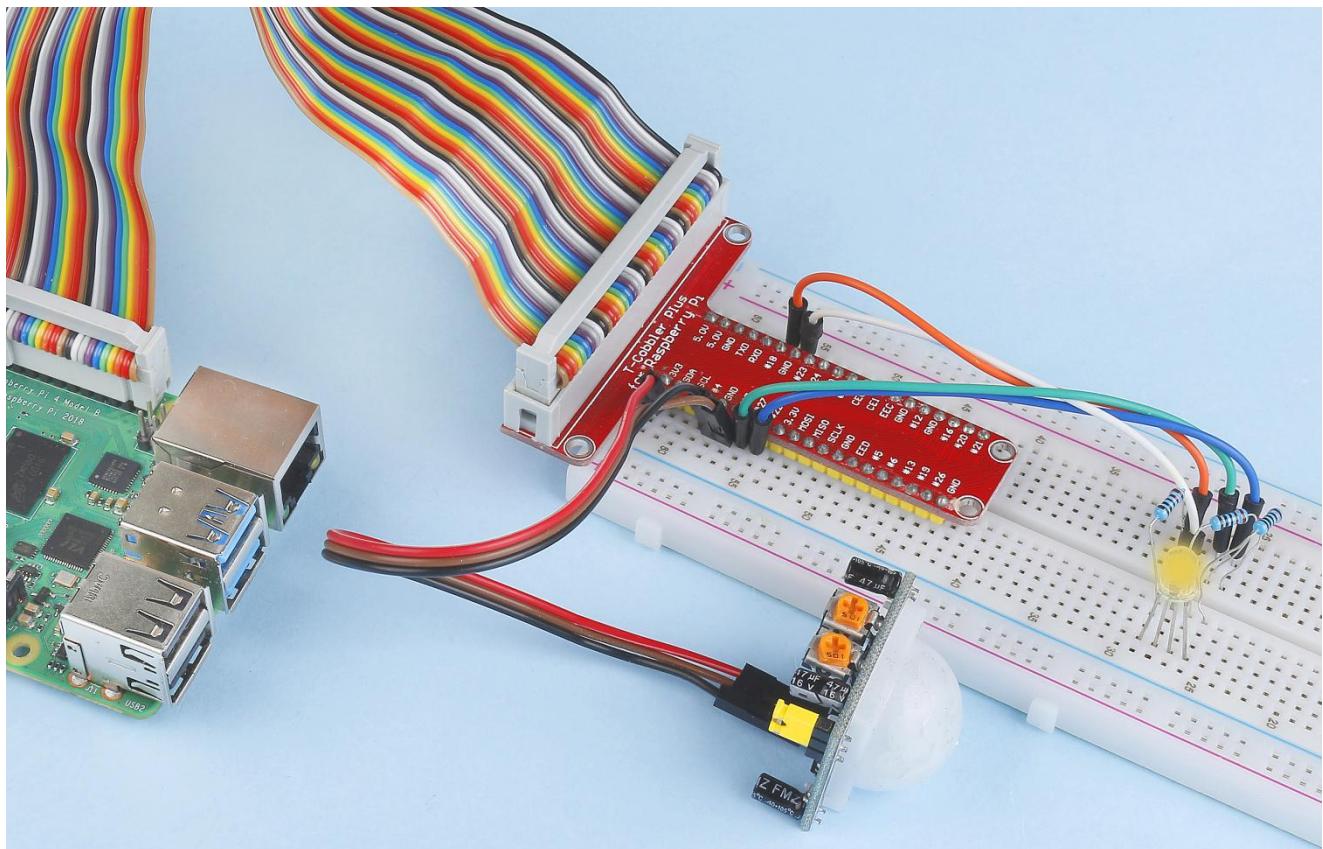

```

これらのコードは RGB LED の色を設定するために使用され、詳しくは 1.1.2-RGB LED を参照してください。

```
def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)
```

PIR が人間の赤外線スペクトルを検出すると、RGB LED が黄色の光を発する。そうでない場合は、青色の光を発する。

現象画像

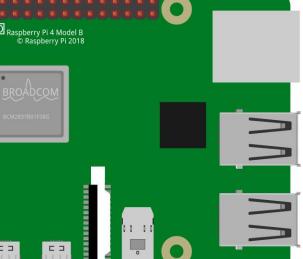
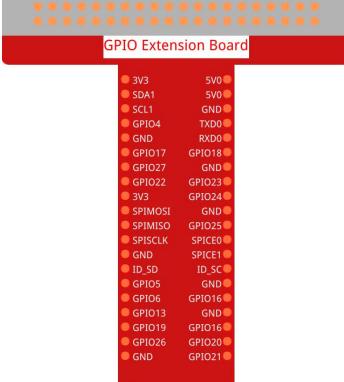
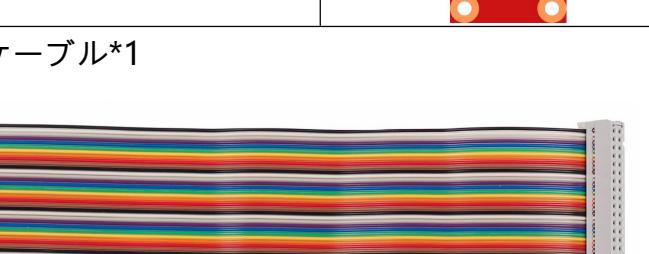
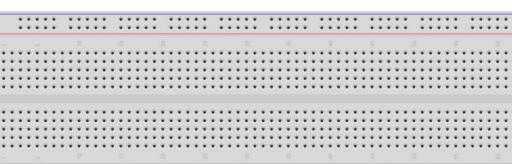


2.2.5 Ultrasonic Sensor Module

前書き

超音波センサーは超音波を使用して物体を正確に検出し、距離を測定する。超音波を送り出し、電子信号に変換する。

晶部

Raspberry Pi 本体*1	T 拡張ボード*1	HC SR04*1																																								
	 <p>GPIO Extension Board</p> <table border="1"> <tbody> <tr><td>3V3</td><td>5V0</td></tr> <tr><td>SDA1</td><td>5V0</td></tr> <tr><td>SCL1</td><td>GND</td></tr> <tr><td>GPIO4</td><td>TxD0</td></tr> <tr><td>GND</td><td>RxD0</td></tr> <tr><td>GPIO17</td><td>GPIO18</td></tr> <tr><td>GPIO27</td><td>GND</td></tr> <tr><td>GPIO22</td><td>GPIO23</td></tr> <tr><td>3V3</td><td>GPIO24</td></tr> <tr><td>SPI_MOSI</td><td>GND</td></tr> <tr><td>SPI_MISO</td><td>GPIO25</td></tr> <tr><td>SPI_SCLK</td><td>SPI_CE0</td></tr> <tr><td>GND</td><td>SPI_CE1</td></tr> <tr><td>ID_SD</td><td>ID_SC</td></tr> <tr><td>GPIO5</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td></tr> <tr><td>GPIO13</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO16</td></tr> <tr><td>GPIO26</td><td>GPIO20</td></tr> <tr><td>GND</td><td>GPIO21</td></tr> </tbody> </table>	3V3	5V0	SDA1	5V0	SCL1	GND	GPIO4	TxD0	GND	RxD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPI_MOSI	GND	SPI_MISO	GPIO25	SPI_SCLK	SPI_CE0	GND	SPI_CE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	 <p>HC-SR04</p> <p>Ucc Trig Echo Gnd</p>
3V3	5V0																																									
SDA1	5V0																																									
SCL1	GND																																									
GPIO4	TxD0																																									
GND	RxD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPI_MOSI	GND																																									
SPI_MISO	GPIO25																																									
SPI_SCLK	SPI_CE0																																									
GND	SPI_CE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
40 ピンケーブル*1		何本のジャンパー線																																								
																																										
		ブレッドボード*1																																								
																																										

原理

超音波

超音波測距モジュールは 2cm-400cm の非接触測定機能を提供し、測距精度は 3mm に達することができる。信号が 5m 以内で安定し、5m 後に信号が徐々に弱まり、7m の位置が消えることを確認できる。

モジュールには、超音波送信機、受信機、と制御回路が含まれている。基本的な原理は次のとおりである：

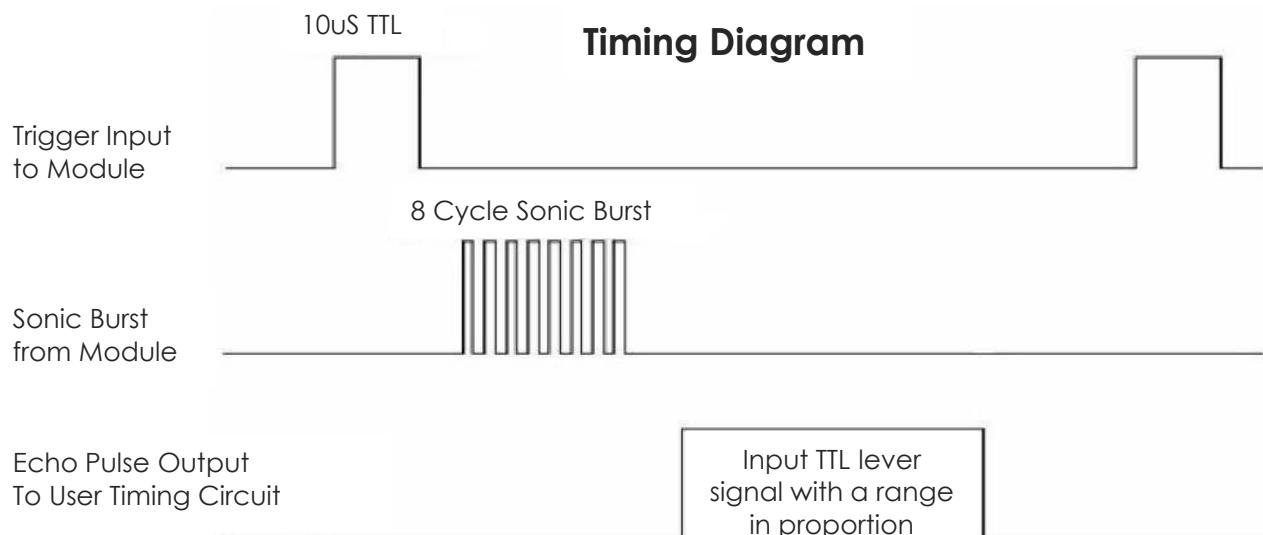
- (1) IO フリップフロップを使用して、少なくとも 10us の高レベル信号を処理する。
 - (2) モジュールは 8 つの 40khz を自動的に送信し、パルス信号が戻すかどうかを検出する。
 - (3) 信号が戻し、高レベルを通過する場合、高出力 IO 持続時間は、超音波の送信から信号の戻りまでの時間である。ここでは、テスト距離 = $(\text{高時間} \times \text{音速 } (340 \text{ m/s}) / 2)$ 。



TRIG	トリガーパルス入力
ECHO	エコーパルス出力
GND	接地
VCC	電源

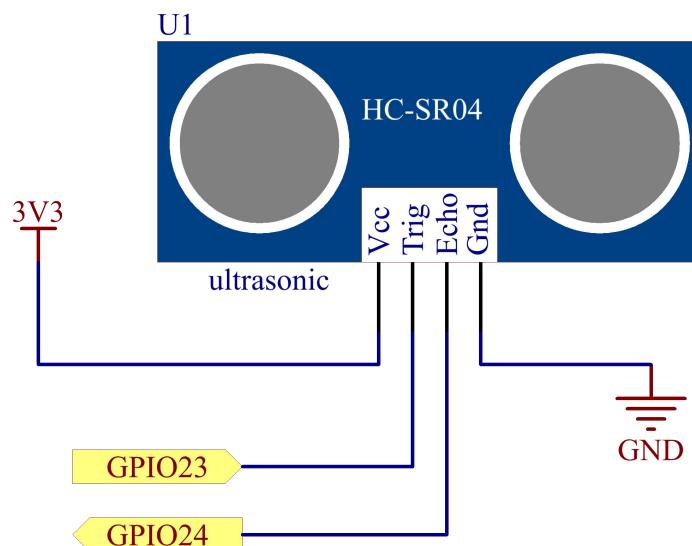
タイミング図を以下に示す。トリガー入力に 10us の短いパルスを供給してレンジングを開始するだけで、モジュールは 40 kHz で 8 サイクルの超音波バーストを送信し、エコーを上げる。トリガー信号を送信してからエコー信号を受信するまでの時間間隔で範囲を計算できる。

式: $\text{us}/58 = \text{センチメートル}$ または $\text{us}/148 = \text{インチ}$; または: 範囲 = 高レベル時間 * 速度 (340M/S) / 2; トリガー信号とエコー信号の信号衝突を防ぐために、60ms 以上の測定サイクルを使用することをお勧めする。



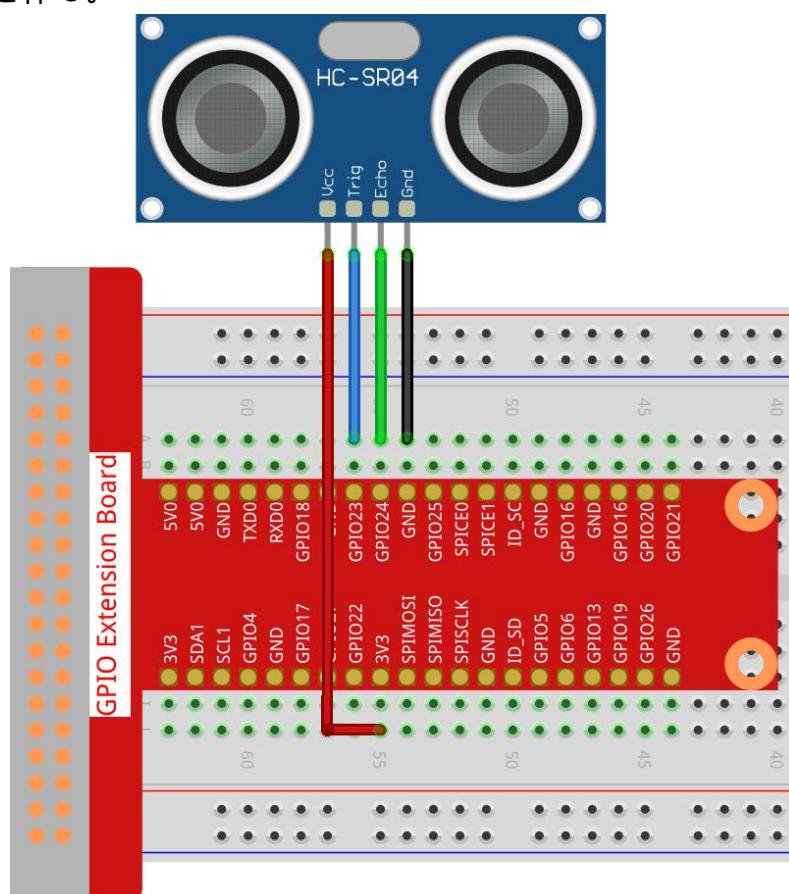
回路図

T ボード名	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.5/
```

ステップ 3: コードをコンパイルする。

```
gcc 2.2.5_Ultrasonic.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードを実行すると、超音波センサモジュールが前方の障害物とモジュール自体の間の距離を検出し、距離値が画面に出力される。

コード

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define Trig    4
#define Echo   5

void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}

float disMeasure(void)
{
    struct timeval tv1;
    struct timeval tv2;
    long time1, time2;
    float dis;

    digitalWrite(Trig, LOW);
    delayMicroseconds(2);

    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);
```

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);

while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);

time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
time2 = tv2.tv_sec * 1000000 + tv2.tv_usec;

dis = (float)(time2 - time1) / 1000000 * 34000 / 2;

return dis;
}

int main(void)
{
    float dis;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    ultraInit();

    while(1){
        dis = disMeasure();
        printf("%0.2f cm\n\n",dis);
        delay(300);
    }

    return 0;
}
```

コードの説明

```
void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
```

超音波ピンを初期化する。一方、Echo を入力、Trig を出力に設定する。

```
float disMeasure(void){};
```

戻り検出距離を計算することにより、この機能は超音波センサーの機能を実現するために使用される。

```
struct timeval tv1;
struct timeval tv2;
```

構造体 timeval は、現在の時刻を保存するために使用される構造体である。完全な構造は次の通りである：

```
struct timeval
{
    __time_t tv_sec;           /* Seconds. */
    __suseconds_t tv_usec;    /* Microseconds. */
};
```

ここで、tv_sec は、エポックが struct timeval を作成するときに費やした秒を表す。Tv_usec はマイクロ秒または秒の一部を表す。

```
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
```

10us の超音波パルスが送信されている。

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);
```

この empty loop は、トリガー信号が送信されたときに、干渉エコー信号がないことを確認してから現在の時刻を取得するために使用される。

```
while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);
```

この empty loop は、エコー信号が受信されて現在の時刻が取得されるまで次のステップが実行されないようにするために使用される。

```
time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;  
time2 = tv2.tv_sec * 1000000 + tv2.tv_usec;
```

struct timeval によって保存された時間を完全なマイクロ秒時間に変換する。

```
dis = (float)(time2 - time1) / 1000000 * 34000 / 2;
```

距離は時間間隔と音の伝播速度によって計算される。空気中の音速: 34000cm/s。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 2.2.5_Ultrasonic.py
```

コードを実行すると、超音波センサーモジュールが前方の障害物とモジュール自体の間の距離を検出し、距離値が画面に出力される。

コード

```
import RPi.GPIO as GPIO  
import time  
  
TRIG = 16  
ECHO = 18  
  
def setup():  
    GPIO.setmode(GPIO.BOARD)  
    GPIO.setup(TRIG, GPIO.OUT)  
    GPIO.setup(ECHO, GPIO.IN)  
  
def distance():  
    GPIO.output(TRIG, 0)  
    time.sleep(0.000002)  
  
    GPIO.output(TRIG, 1)  
    time.sleep(0.00001)  
    GPIO.output(TRIG, 0)
```

```

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def loop():
    while True:
        dis = distance()
        print ('Distance: %.2f' % dis )
        time.sleep(0.3)

def destroy():
    GPIO.cleanup()

if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

コードの説明

```
def distance():
```

戻り検出距離を計算することにより、この機能は超音波センサーの機能を実現するために使用される。

```

GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)

```

これは 10us の超音波パルスを送信している。

```
while GPIO.input(ECHO) == 0:  
    a = 0  
    time1 = time.time()
```

この empty loop は、トリガー信号が送信されたときに、干渉エコー信号がないことを確認してから現在の時刻を取得するために使用される。

```
while GPIO.input(ECHO) == 1:  
    a = 1  
    time2 = time.time()
```

この empty loop は、エコー信号が受信されて現在の時刻が取得されるまで次のステップが実行されないようにするために使用される。

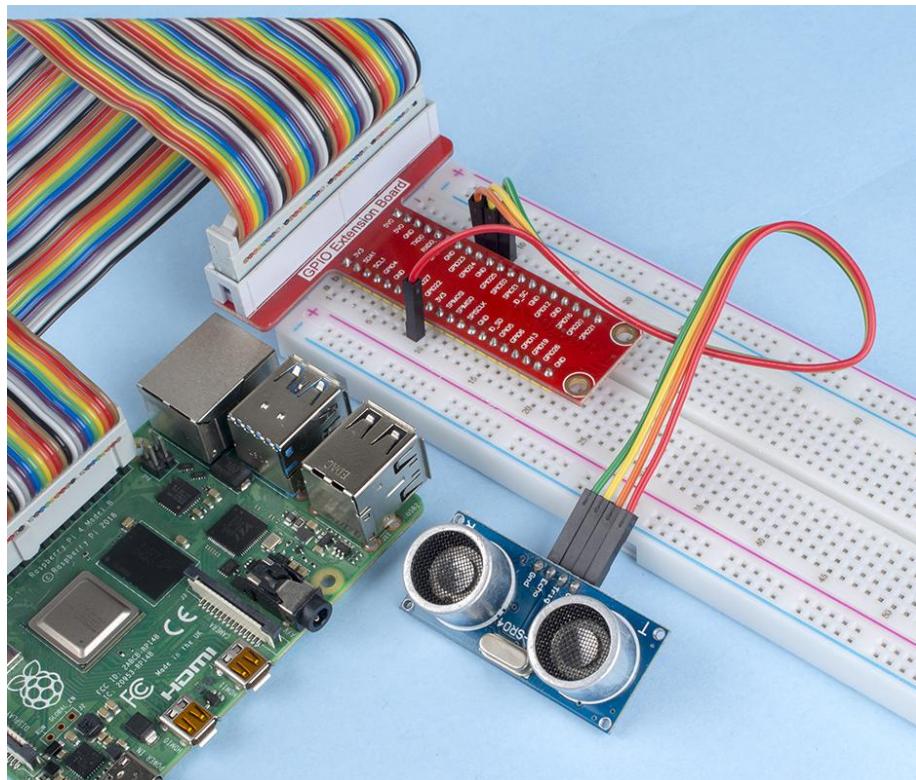
```
during = time2 - time1
```

間隔計算を実行する。

```
return during * 340 / 2 * 100
```

距離は時間間隔の光と音の伝播速度によって計算される。空気中の音速：340m/s。

現象画像



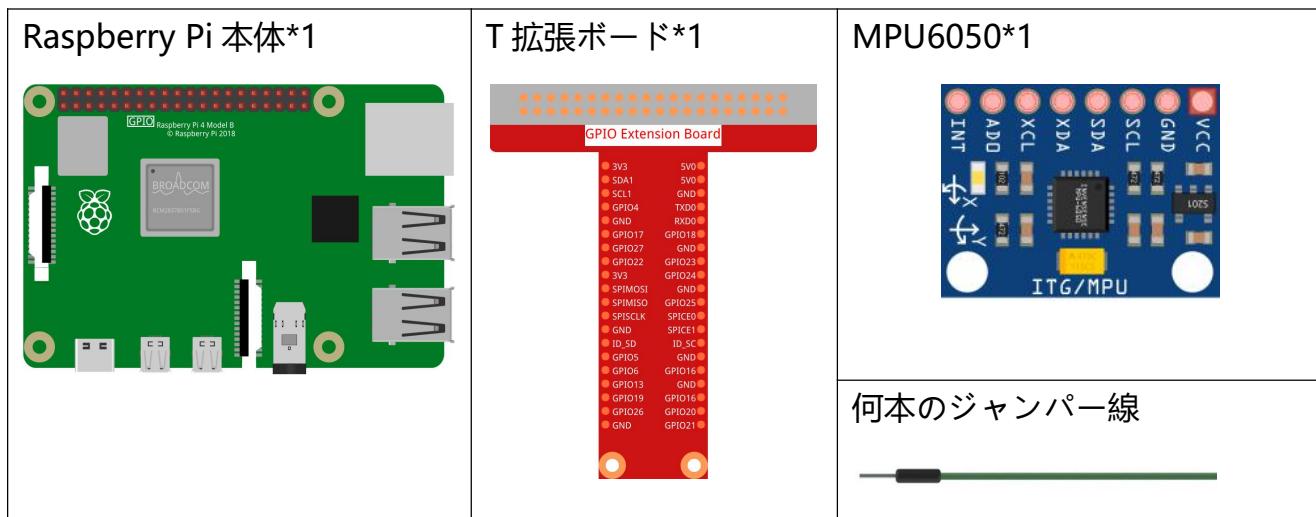
2.2.6 MPU6050 Module

前書き

MPU-6050 は、低消費電力、低コスト、高性能などの機能を備えたスマートフォン、タブレット、ウェアラブルセンサー向けに設計された世界初で唯一の 6 軸物標追跡装置（3 軸ジャイロスコープと 3 軸加速度センサー）である。

この実験では、I2C を使用して、MPU6050 の 3 軸加速度センサーと 3 軸ジャイロスコープの値を取得し、画面に表示する。

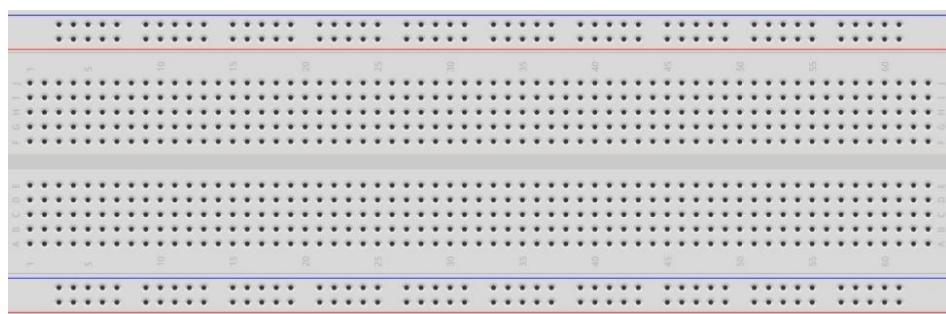
部品



40 ピンケーブル*1



ブレッドボード*1



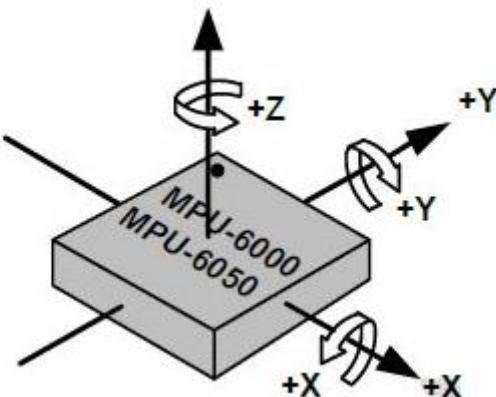
原理

MPU6050

MPU-6050 は、6 軸（3 軸ジャイロスコープ、3 軸加速度計を組み合わせた）物標追跡装置である。

その三つの座標系は次のように定義される：

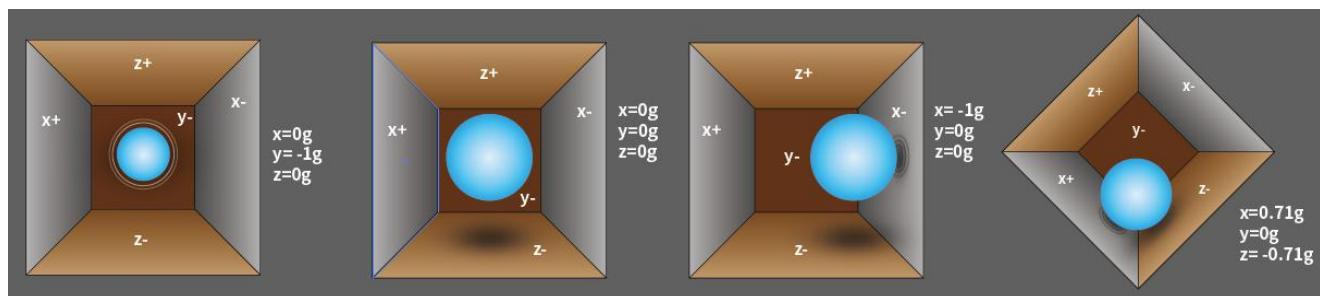
MPU6050 をテーブルの上に平らに置き、ラベルのある面が上向きで、この表面のドットが左上隅にあることを確認してください。次に、上向きの直立方向がチップの z 軸である。左から右への方向は X 軸と見なされる。したがって、後ろから前への方向は Y 軸として定義される。



3 軸加速度計

加速度計は加えられた機械応力に応答して電荷を生成する特定の材料の能力である圧電効果の原理で動作する。

ここで、上記の写真のように、小さなボールの中に直方体の箱があることを想像してください。この箱の壁は圧電結晶で作られている。箱を傾けると、重力によりボールが傾斜の方向に移動する。ボールが衝突する壁は、小さな圧電電流を生成する。合計で、立方体には 3 組の向かい合った壁がある。各ペアは、3D 空間の軸：X、Y、Z 軸に対応する。圧電壁から生成される電流に応じて、傾斜の方向とその大きさを決定できる。



MPU6050 を使用して、各座標軸の加速度を検出できる（静止デスクトップ状態では、Z 軸の加速度は 1 重力単位で、X 軸と Y 軸は

0 である）。傾斜または無重量/重量超過の状態にある場合、対応する測定値が変化する。

プログラムで選択できる測定範囲には、 $+/-2g$ 、 $+/-4g$ 、 $+/-8g$ 、と各精度に対応する $+/-16g$ （デフォルトでは $2g$ ）の4種類がある。値の範囲は $-32768 \sim 32767$ である。

読み取り値を測定範囲にマッピングすることにより、加速度計の読み取り値は加速度値に変換される。

$$\text{加速度} = (\text{加速度計軸の生データ} / 65536 * \text{フルスケールの加速度範囲}) g$$

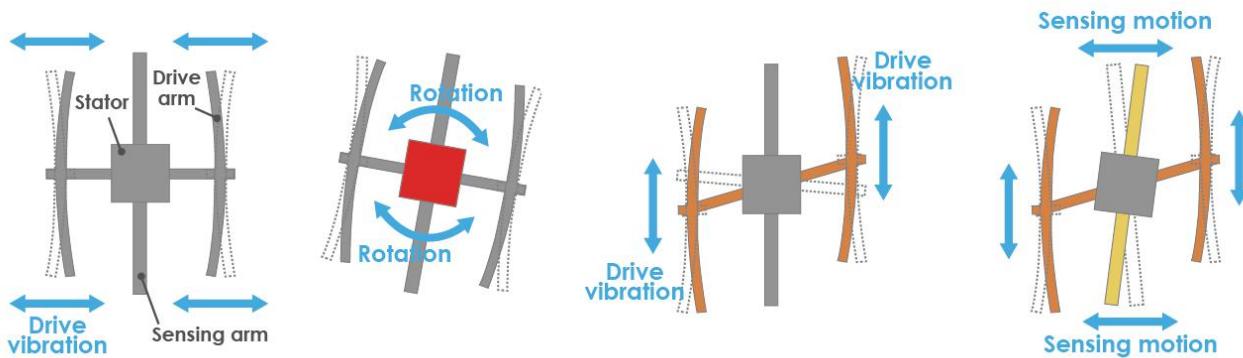
X 軸を例として、加速度計 X 軸の生データは 16384 の場合、範囲を $+/- 2g$ に選択する：

$$X \text{ 軸による加速度} = (16384 / 65536 * 4) g$$

$$= 1g$$

3 軸ジャイロスコープ

ジャイロスコープはコリオリ加速の原理で動作する。フォークのような構造があり、常に前後に動いていると想像してください。圧電結晶を使用して所定の位置に保持される。この配置を傾けようとするたびに、結晶は傾斜の方向に力を受ける。これは、可動フォークの慣性の結果によって引き起こされる。したがって、結晶は圧電効果と一致して電流を生成し、この電流は増幅される。



1. Normally, a drive arm vibrates in a certain direction.

2. Direction of rotation

3. When the gyro is rotated, the Coriolis force acts on the drive arms, producing vertical vibration.

4. The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

また、ジャイロスコープには、 $+/-250$ 、 $+/-500$ 、 $+/-1000$ 、 $+/-2000$ 。計算方法と加速は基本的に一貫している。

読み取り値を角速度に変換する式は次の通りである：

$$\text{角速度} = (\text{ジャイロスコープの軸生データ} / 65536 * \text{フルスケールジャイロスコープの範囲}) / s$$

X 軸、たとえば、加速度計の X 軸の生データは 16384 で、範囲は $+/-250 / s$ である：

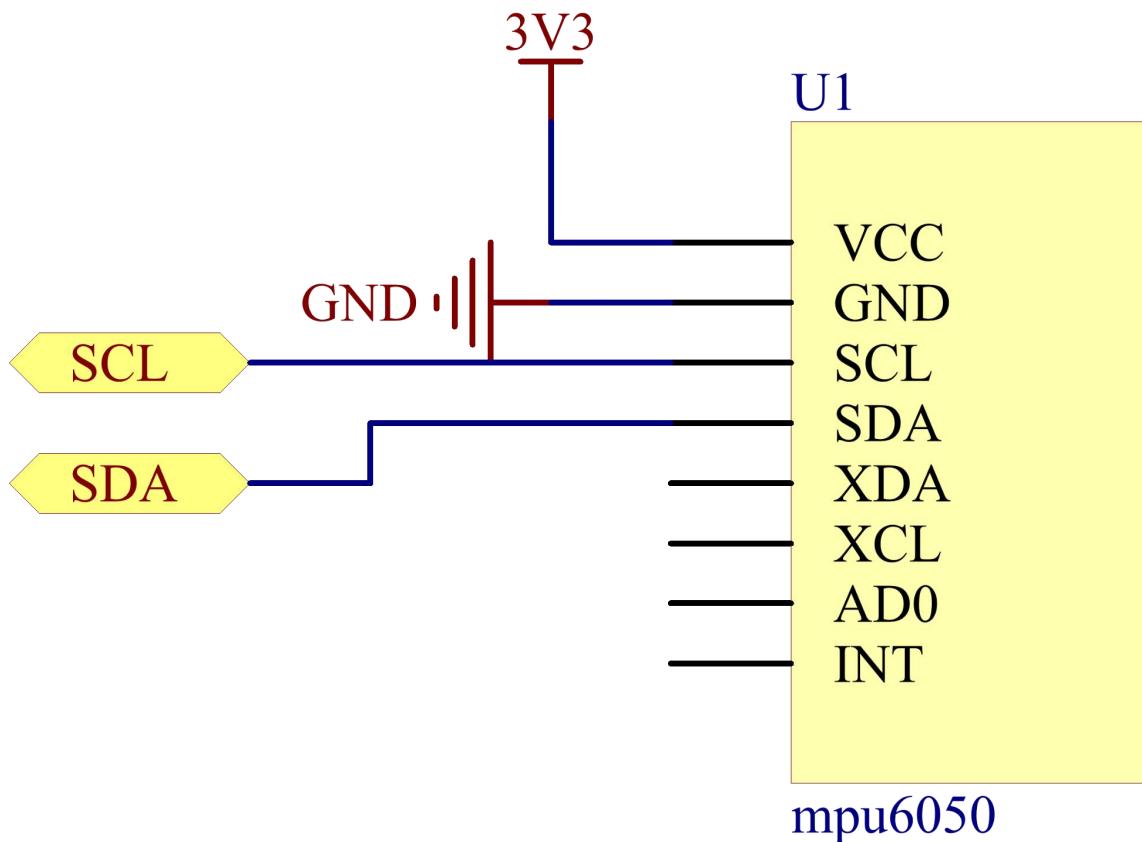
$$X \text{ 軸による角速度} = (16384 / 65536 * 500) / s$$

$$= 125 / s$$

回路図

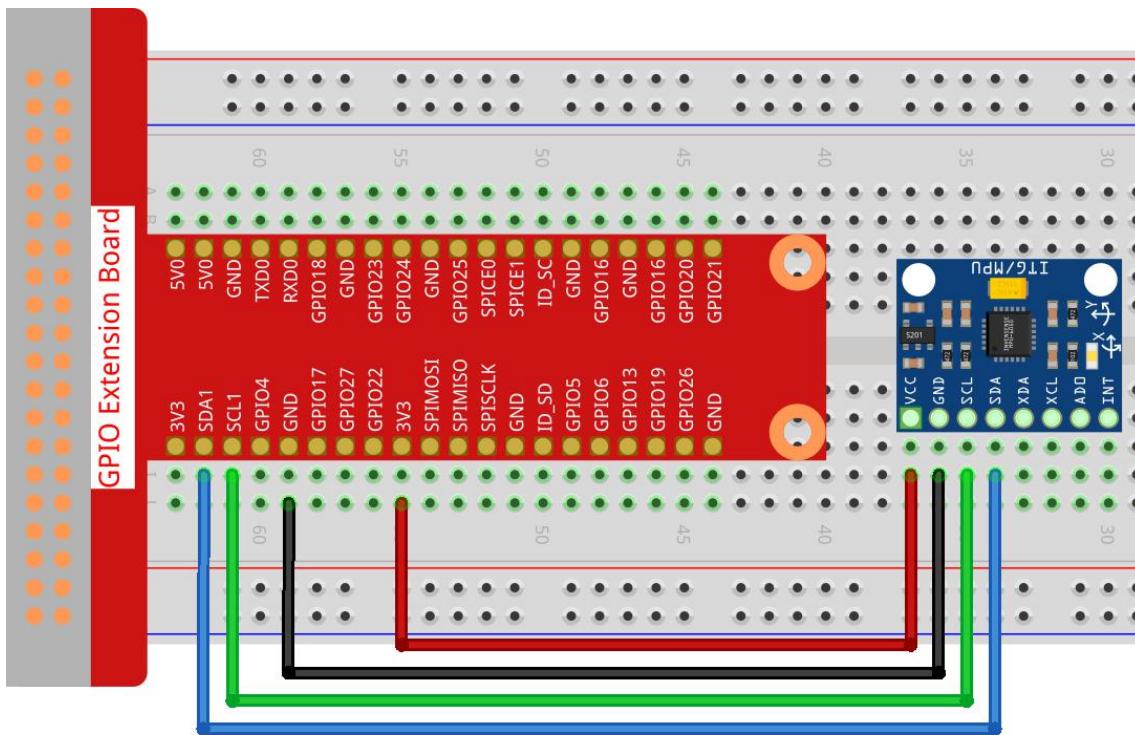
MPU6050 は I2C バスインターフェイスを介してマイクロコントローラーと通信する。SDA1 と SCL1 を対応するピンに接続する必要がある。

T ボード名	physical
SDA1	Pin 3
SCL1	Pin 5



実験手順

ステップ1：回路を作る。



ステップ2：I2C設定（付録を参照してください。I2Cを設定している場合は、この手順をスキップしてください。）

➤ C言語ユーザー向け

ステップ3：コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.6/
```

ステップ4：コードをコンパイルする。

```
gcc 2.2.6_mpu6050.c -lwiringPi -lm
```

ステップ5：EXEファイルを実行する。

```
sudo ./a.out
```

コードを実行すると、MPU6050によって読み取られたx軸、y軸の転向角、加速度、各軸の角速度が計算後に画面に入力される。

コード

```
#include <wiringPi2C.h>
#include <wiringPi.h>
#include <stdio.h>
#include <math.h>
int fd;
```

```

int acclX, acclY, acclZ;
int gyroX, gyroY, gyroZ;
double acclX_scaled, acclY_scaled, acclZ_scaled;
double gyroX_scaled, gyroY_scaled, gyroZ_scaled;

int read_word_2c(int addr)
{
    int val;
    val = wiringPi2CReadReg8(fd, addr);
    val = val << 8;
    val += wiringPi2CReadReg8(fd, addr+1);
    if (val >= 0x8000)
        val = -(65536 - val);
    return val;
}

double dist(double a, double b)
{
    return sqrt((a*a) + (b*b));
}

double get_y_rotation(double x, double y, double z)
{
    double radians;
    radians = atan2(x, dist(y, z));
    return -(radians * (180.0 / M_PI));
}

double get_x_rotation(double x, double y, double z)
{
    double radians;
    radians = atan2(y, dist(x, z));
    return (radians * (180.0 / M_PI));
}

int main()
{
    fd = wiringPi2CSetup (0x68);
    wiringPi2CWriteReg8 (fd, 0x6B, 0x00); // disable sleep mode
    printf("set 0x6B=%X\n", wiringPi2CReadReg8 (fd, 0x6B));
}

```

```
while(1) {  
  
    gyroX = read_word_2c(0x43);  
    gyroY = read_word_2c(0x45);  
    gyroZ = read_word_2c(0x47);  
  
    gyroX_scaled = gyroX / 131.0;  
    gyroY_scaled = gyroY / 131.0;  
    gyroZ_scaled = gyroZ / 131.0;  
  
    //Print values for the X, Y, and Z axes of the gyroscope sensor.  
    printf("My gyroX_scaled: %f\n", gyroX_scaled);  
    printf("My gyroY_scaled: %f\n", gyroY_scaled);  
    printf("My gyroZ_scaled: %f\n", gyroZ_scaled);  
  
    acclX = read_word_2c(0x3B);  
    acclY = read_word_2c(0x3D);  
    acclZ = read_word_2c(0x3F);  
  
    acclX_scaled = acclX / 16384.0;  
    acclY_scaled = acclY / 16384.0;  
    acclZ_scaled = acclZ / 16384.0;  
  
    //Print the X, Y, and Z values of the acceleration sensor.  
    printf("My acclX_scaled: %f\n", acclX_scaled);  
    printf("My acclY_scaled: %f\n", acclY_scaled);  
    printf("My acclZ_scaled: %f\n", acclZ_scaled);  
  
    printf("My X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));  
    printf("My Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));  
  
    delay(100);  
}  
return 0;  
}
```

コードの説明

```
int read_word_2c(int addr)
{
    int val;
    val = wiringPiI2CReadReg8(fd, addr);
    val = val << 8;
    val += wiringPiI2CReadReg8(fd, addr+1);
    if (val >= 0x8000)
        val = -(65536 - val);
    return val;
}
```

MPU6050 から送信されたセンサーデータを読み取る。

```
double get_y_rotation(double x, double y, double z)
{
    double radians;
    radians = atan2(x, dist(y, z));
    return -(radians * (180.0 / M_PI));
}
```

Y 軸の転向角を取得する。

```
double get_x_rotation(double x, double y, double z)
{
    double radians;
    radians = atan2(y, dist(x, z));
    return (radians * (180.0 / M_PI));
}
```

X 軸の転向角を計算する。

```
gyroX = read_word_2c(0x43);
gyroY = read_word_2c(0x45);
gyroZ = read_word_2c(0x47);

gyroX_scaled = gyroX / 131.0;
gyroY_scaled = gyroY / 131.0;
gyroZ_scaled = gyroZ / 131.0;

//Print values for the X, Y, and Z axes of the gyroscope sensor.
```

```
printf("My gyroX_scaled: %f\n", gyroX_X_scaled);
printf("My gyroY_scaled: %f\n", gyroY_Y_scaled);
printf("My gyroZ_scaled: %f\n", gyroY_Z_scaled);
```

ジャイロセンサーの x 軸、y 軸、z 軸の値を読み取り、メタデータを角速度値に変換してから出力する。

```
acclX = read_word_2c(0x3B);
acclY = read_word_2c(0x3D);
acclZ = read_word_2c(0x3F);

acclX_scaled = acclX / 16384.0;
acclY_scaled = acclY / 16384.0;
acclZ_scaled = acclZ / 16384.0;

//Print the X, Y, and Z values of the acceleration sensor.
printf("My acclX_scaled: %f\n", acclX_scaled);
printf("My acclY_scaled: %f\n", acclY_scaled);
printf("My acclZ_scaled: %f\n", acclZ_scaled);
```

加速度センサーの x 軸、y 軸、z 軸の値を読み取り、メタデータを加速速度値（重力単位）に変換してから出力する。

```
printf("My X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
printf("My Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
```

X 軸と Y 軸の転向角をプリントする。

➤ Python 言語ユーザー向け

ステップ 3: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 4: EXE ファイルを実行する。

```
sudo python3 2.2.6_mpu6050.py
```

コードを実行し、X 軸と Y 軸の転向角、加速度と MPU6050 によって読み取られた各軸の角速度は、計算後に画面に入力される。

コード

```
import smbus
import math
import time

# Power management registers
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

bus = smbus.SMBus(1) # or bus = smbus.SMBus(1) for Revision 2 boards
address = 0x68      # This is the address value read via the i2cdetect command
```

```
# Now wake the 6050 up as it starts in sleep mode
bus.write_byte_data(address, power_mgmt_1, 0)

while True:
    time.sleep(0.1)
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))

    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
    accel_zout = read_word_2c(0x3f)

    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
    print ("accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled)
    print ("accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled)

    print ("x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
    print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))

    time.sleep(0.5)
```

コードの説明

```
def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val
```

MPU6050 から送信されたセンサーデータを読み取る。

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

Y 軸の転向角を計算する。

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

X 軸の転向角を計算する。

```
gyro_xout = read_word_2c(0x43)
gyro_yout = read_word_2c(0x45)
gyro_zout = read_word_2c(0x47)

print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))
```

ジャイロセンサーの X 軸、Y 軸、Z 軸の値を読み取り、メタデータを角速度値に変換してから出力する。

```
accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
```

```
accel_zout = read_word_2c(0x3f)

accel_xout_scaled = accel_xout / 16384.0
accel_yout_scaled = accel_yout / 16384.0
accel_zout_scaled = accel_zout / 16384.0

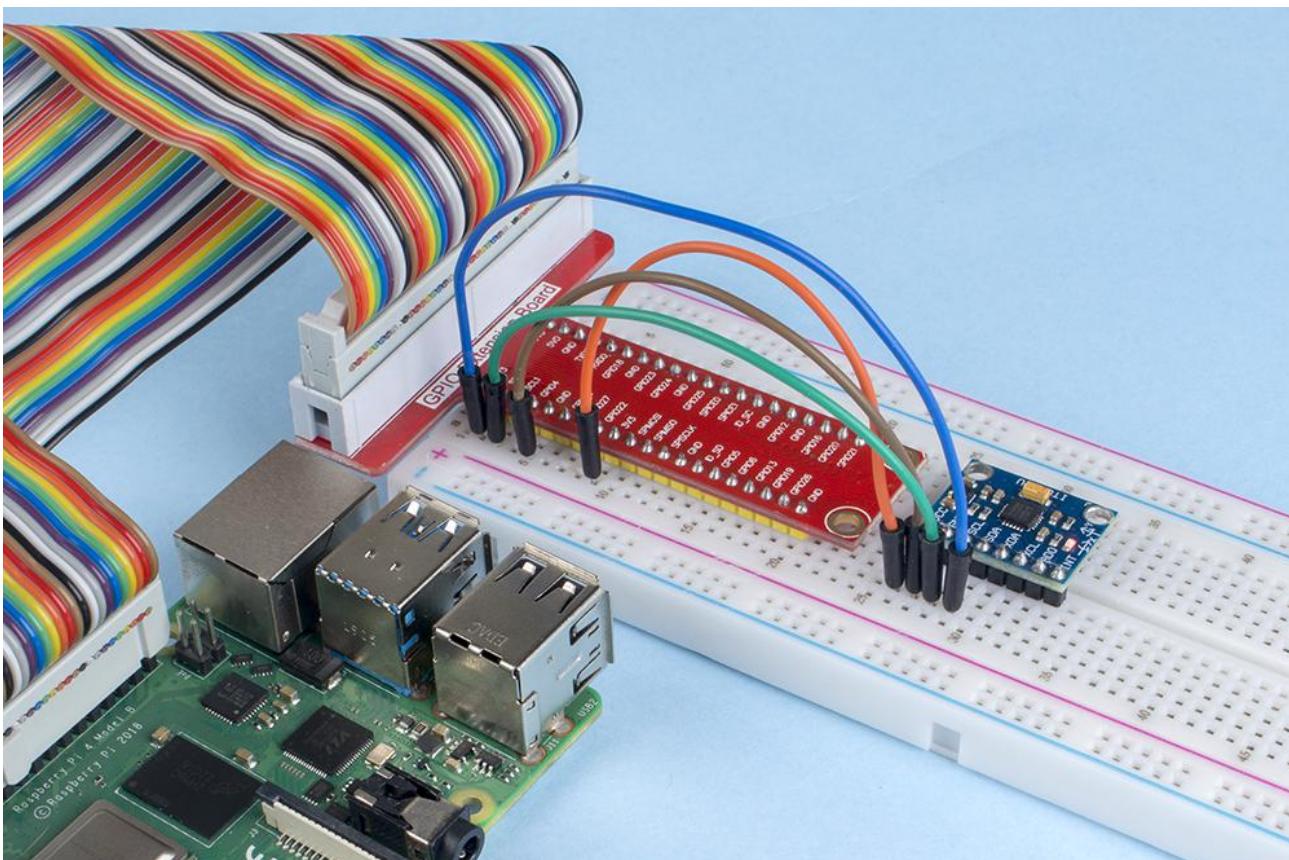
print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
print ("accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled)
print ("accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled)
```

加速度センサーの X 軸、Y 軸、Z 軸の値を読み取り、メタデータを加速度値（重力単位）に変換してから出力する。

```
print ("x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
```

X 軸と Y 軸の転向角をプリントする。

現象画像



2.2.7 MFRC522 RFID Module

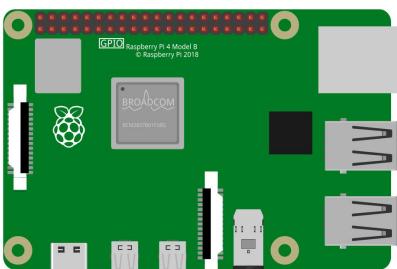
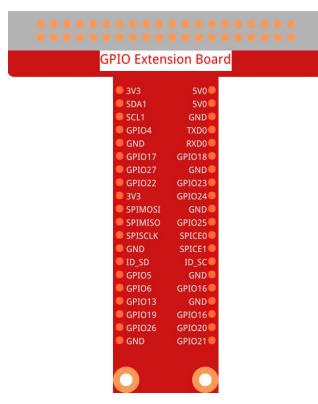
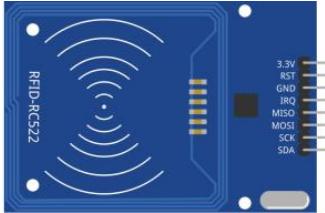
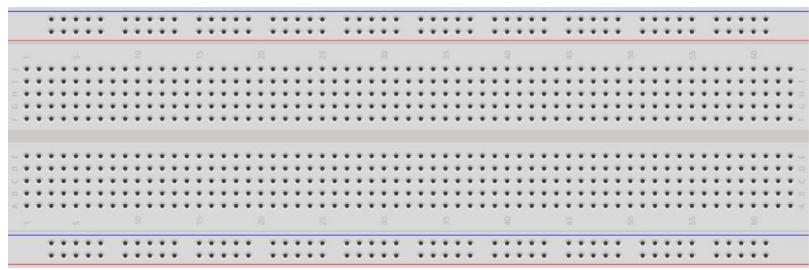
前書き

無線周波数識別 (RFID) は、オブジェクト（またはタグ）と質問デバイス（またはリーダー）の間の無線通信を使用して、そのようなオブジェクトを自動的に追跡したり識別したりする技術を指す。

この技術の最も一般的なアプリケーションには、小売サプライチェーン、軍事サプライチェーン、自動決済方法、荷物の追跡と管理、ドキュメントの追跡と医薬品管理などが含まれておる。

このプロジェクトでは、読み取りと書き込みに RFID を使用する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	RFID RC522 (ホワイトカードとキータグ付き) *1																																								
	 GPIO Extension Board <table border="1"><tr><td>● 3V3</td><td>SVO</td></tr><tr><td>● SDA1</td><td>SVO</td></tr><tr><td>● SCL1</td><td>GND</td></tr><tr><td>● GPIO4</td><td>TXD0</td></tr><tr><td>● GND</td><td>RXD0</td></tr><tr><td>● GPIO17</td><td>GPIO18</td></tr><tr><td>● GPIO27</td><td>GND</td></tr><tr><td>● GPIO22</td><td>GPIO23</td></tr><tr><td>● 3V</td><td>GPIO24</td></tr><tr><td>● SPIMOSI</td><td>GND</td></tr><tr><td>● SPIMISO</td><td>GPIO25</td></tr><tr><td>● SPICLK</td><td>SPICE0</td></tr><tr><td>● ID_SD</td><td>SPICE1</td></tr><tr><td>● GPIO5</td><td>ID_SC</td></tr><tr><td>● GPIO6</td><td>GND</td></tr><tr><td>● GPIO13</td><td>GPIO16</td></tr><tr><td>● GND</td><td>GND</td></tr><tr><td>● GPIO19</td><td>GPIO16</td></tr><tr><td>● GPIO26</td><td>GPIO20</td></tr><tr><td>● GND</td><td>GPIO21</td></tr></table>	● 3V3	SVO	● SDA1	SVO	● SCL1	GND	● GPIO4	TXD0	● GND	RXD0	● GPIO17	GPIO18	● GPIO27	GND	● GPIO22	GPIO23	● 3V	GPIO24	● SPIMOSI	GND	● SPIMISO	GPIO25	● SPICLK	SPICE0	● ID_SD	SPICE1	● GPIO5	ID_SC	● GPIO6	GND	● GPIO13	GPIO16	● GND	GND	● GPIO19	GPIO16	● GPIO26	GPIO20	● GND	GPIO21	 RFID-RC522
● 3V3	SVO																																									
● SDA1	SVO																																									
● SCL1	GND																																									
● GPIO4	TXD0																																									
● GND	RXD0																																									
● GPIO17	GPIO18																																									
● GPIO27	GND																																									
● GPIO22	GPIO23																																									
● 3V	GPIO24																																									
● SPIMOSI	GND																																									
● SPIMISO	GPIO25																																									
● SPICLK	SPICE0																																									
● ID_SD	SPICE1																																									
● GPIO5	ID_SC																																									
● GPIO6	GND																																									
● GPIO13	GPIO16																																									
● GND	GND																																									
● GPIO19	GPIO16																																									
● GPIO26	GPIO20																																									
● GND	GPIO21																																									
40 ピンケーブル*1		何本のジャンパー線																																								
																																										
ブレッドボード*1																																										

原理

RFID

無線周波数識別 (RFID) は、オブジェクト（またはタグ）と質問デバイス（またはリーダー）の間の無線通信を使用して、そのようなオブジェクトを自動的に追跡したり識別したりする技術を指す。タグの送信範囲はリーダーから数メートルに制限されている。リーダーとタグの間の明確な見通し線は必ずしも必要ではない。

ほとんどのタグには、少なくとも 1 つの集積回路 (IC) とアンテナが含まれている。マイクロチップは情報を保存し、リーダーとの無線周波数 (RF) 通信を管理する。パッシブタグは独立したエネルギー源を持たず、リーダーによって提供される外部電磁信号に依存して動作する。しかしアクティブタグバッテリーなどの独立したエネルギー源が含まれている。したがって、処理、送信機能と範囲が拡大している可能性がある。



MFRC522

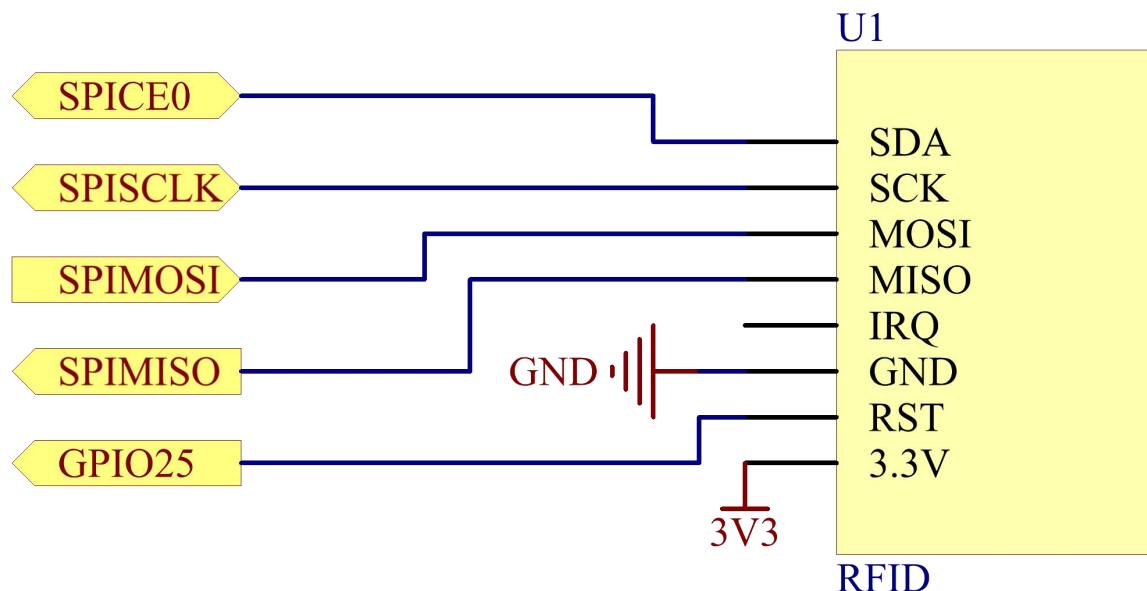
MFRC522 は、読み取りと書き込みカードチップの一種である。通常、13.56MHz の無線で使用される。NXP Company によって発売された低電圧、低コスト、小型の非接触カードチップであり、インテリジェント機器と持ち軽い手元デバイスの最良の選択である。

MF RC522 はすべてのタイプの 13.56MHz パッシブ非接触通信方法とプロトコルで完全に開示された高度な変調と復調の概念を使用している。さらに、MIFARE 製品を検証するための高速 CRYPTO1 暗号化アルゴリズムをサポートしている。MFRC522 は最大 424kbit/s の双方向データ伝送速度で、MIFARE シリーズの高速非接触通信もサポートしている。13.56MHz 高集積リーダーカードシリーズの新しいメンバーとして、MF RC522 は既存の MF RC500 と MF RC530 と非常に似ているが、

多くの違いがある。配線が少ないシリアル方式でホストマシンと通信する。SPI、I2C、とシリアル UART モード (RS232 に類似) から選択できる。これにより、接続の削減、PCB ボードスペースの節約 (サイズの縮小)、およびコストの削減に役立つ。

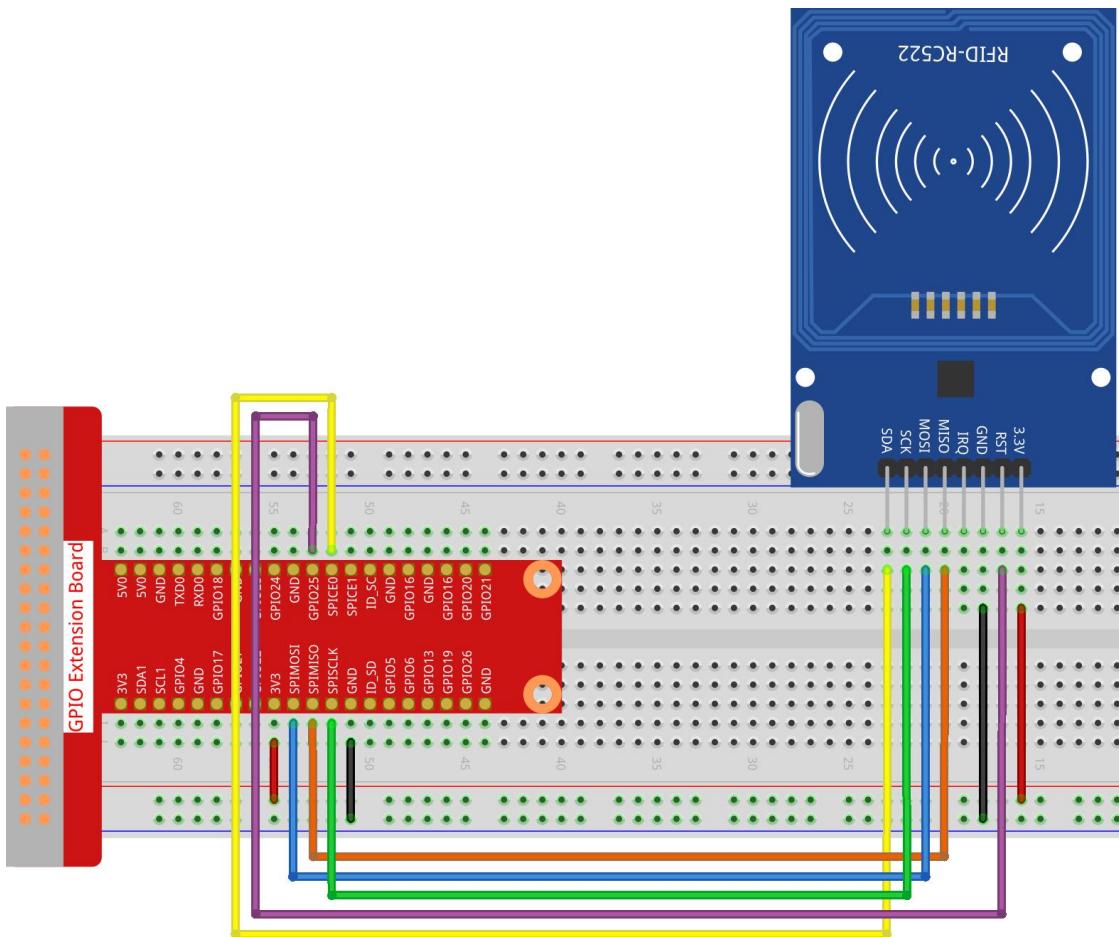
回路図

T ボード名	physical	wiringPi	BCM
SPICE0	Pin 24	10	8
SPISCLK	Pin 23	14	11
SPIMOSI	Pin 19	12	10
SPIMISO	Pin 21	13	9
GPIO25	Pin 22	6	25



実験手順

ステップ1：回路を作る。



ステップ2：SPIを設定する（詳細については、付録を参照してください。SPIを設定している場合は、この手順をスキップしてください。）

➤ C言語ユーザー向け

ステップ3：コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.7/
```

ステップ4：コードをコンパイルする。

```
make read  
make write
```

ご注意：カードIDの読み取りまたは書き込みには二つの例があり、必要に応じていずれかを選択できる。

ステップ5：EXEファイルを実行する。

```
sudo ./read  
sudo ./write
```

コードの説明

```
InitRc522();
```

この関数は RFID RC522 モジュールを初期化するために使用される。

```
uint8_t read_card_data();
```

この関数はカードのデータを読み取るために使用され、読み取りが成功した場合、「1」を返す。

```
uint8_t write_card_data(uint8_t *data);
```

この関数は、カードのデータを書き込むために使用され、書き込みが成功すると「1」を返す。

* data はカードに書き込まれる情報である。

➤ Python 言語ユーザー向け

ステップ3: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/2.2.7
```

ステップ4: EXE ファイルを実行する。

```
sudo python3 2.2.7_read.py  
sudo python3 2.2.7_write.py
```

ご注意: カード ID の読み取りまたは書き込みには二つの例があり、必要に応じていずれかを選択できる。

コードの説明

```
RC522()
```

rc522 クラスをインスタンス化する。

```
RC522.Pcd_start()
```

RFID を初期化する

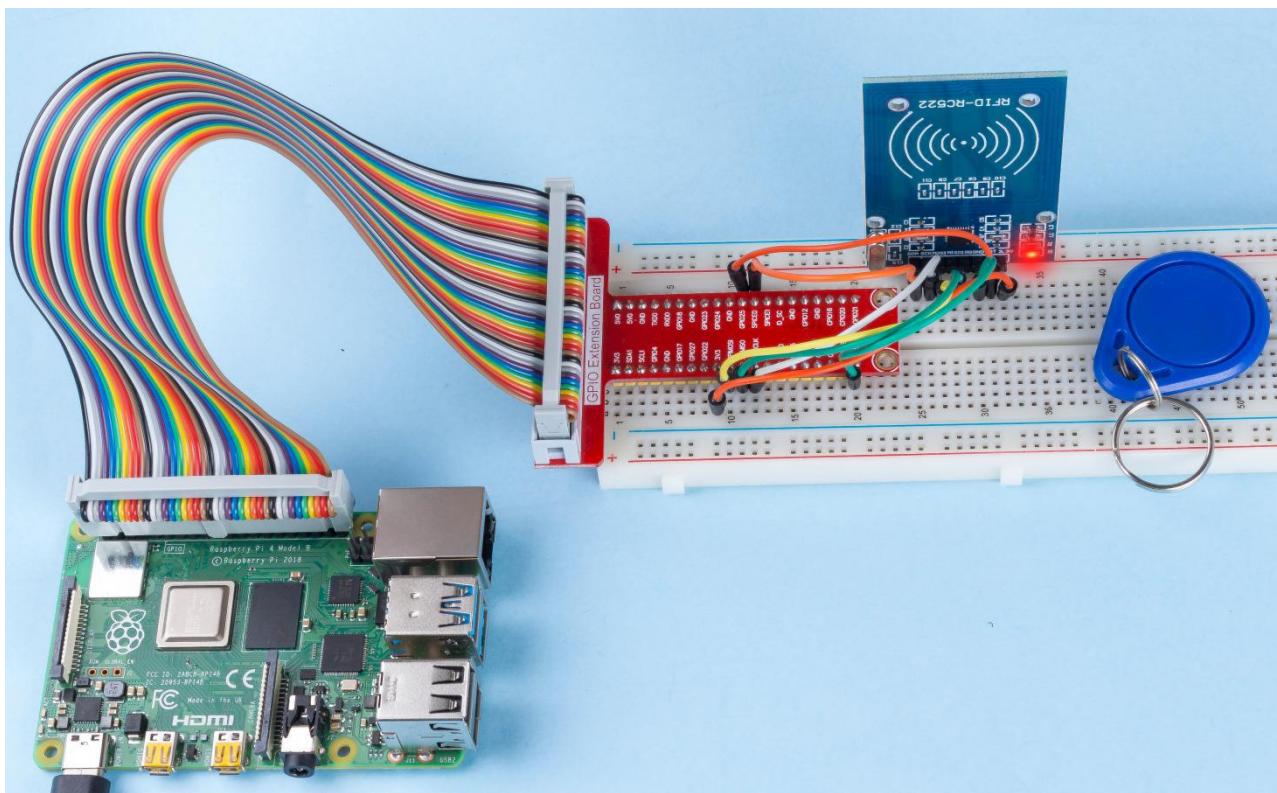
```
RC522.read_card_data(addr)
```

この関数はカードデータの読み取りに使用される。読み取りが成功すると、「1」が返される。addr はカードのアドレスである。

RC522.write_card_data(addr, data)

この関数は、カードデータの書き込みに使われ、書き込みが成功した場合、「1」が戻される。addr はカードのアドレスであり、data はカードに書き込まれる情報である。

現象画像



3 拡張

この章では、非常に面白い拡張実験をいくつか示す。次の点に注意してください:

- 1)コードと配線ははるかに複雑になり、これらの実験を完了するまでにしばらくお待ちください。
- 2)完全なコードはドキュメントに記載されていないため、コードフォルダーに入り、完全なコードをチェックすることはできる。

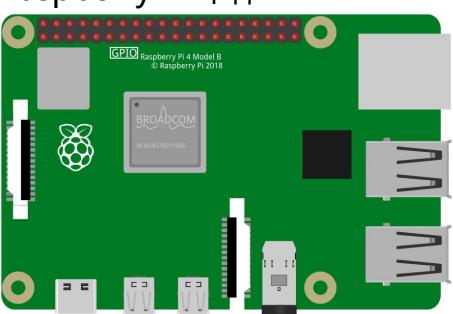
3.1 アプリケーション

3.1.1 Counting Device

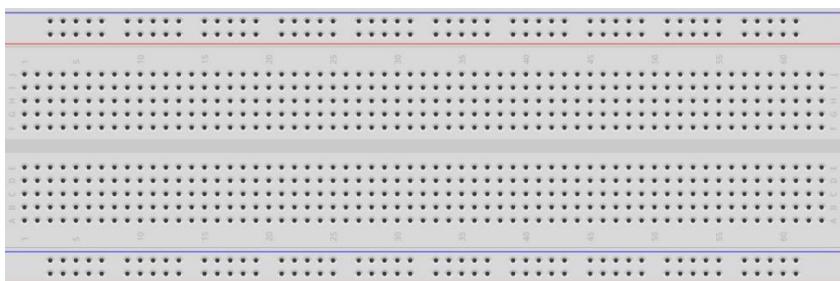
前書き

ここでは、PIR センサーと 4 枚のセグメントディスプレイで構成される数字表示カウンターシステムを作成する。PIR が誰かが通り過ぎていることを検出すると、4 枚のセグメントディスプレイの数字に 1 が加算される。このカウンターを使用して、通路を歩いている人の数をカウントできる。

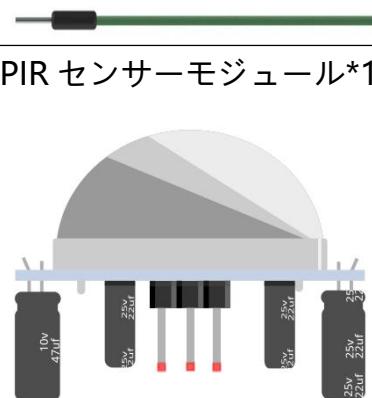
部品

Raspberry Pi 本体*1	T 拡張ボード*1	4 枚 7 セグメントディスプレイ*1
	 GPIO Extension Board Pinout: 3V3, SDA1, GND, GPIO4, TXD0, GND, GPIO17, GPIO18, GPIO27, GND, GPIO22, GPIO25, GND, SPI_MOSI, GND, SPI_MISO, GPIO25, SPI_SCLK, SPI_CE1, GND, ID_SD, ID_SC, GPIO5, GND, GPIO6, GPIO16, GPIO13, GND, GPIO19, GPIO16, GPIO26, GPIO20, GND, GPIO21, GND	
40 ピンケーブル*1		抵抗器 (220Ω) * 4 
		74HC595*1 
		何本のジャンパー線

ブレッドボード*1

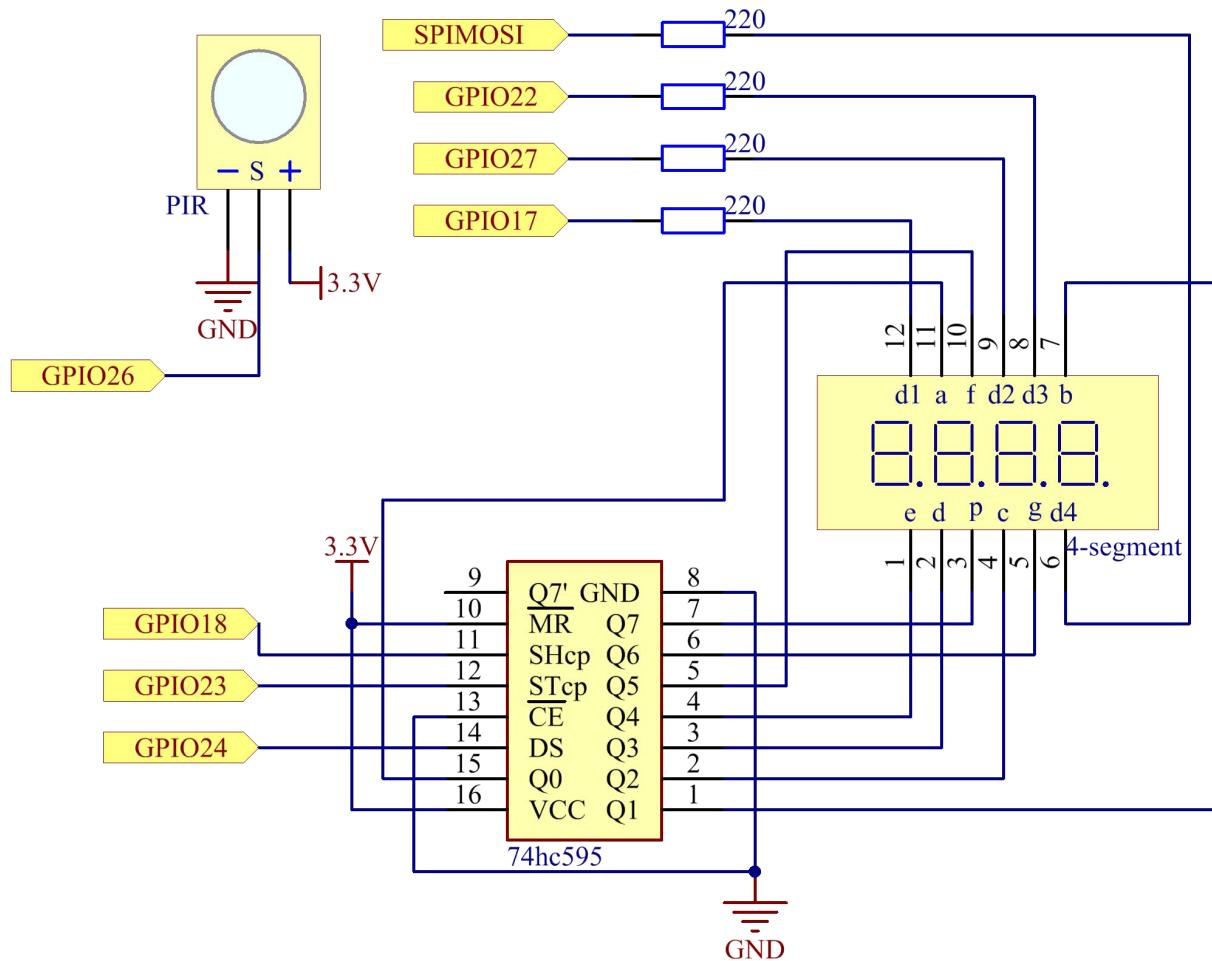


PIR センサー モジュール*1



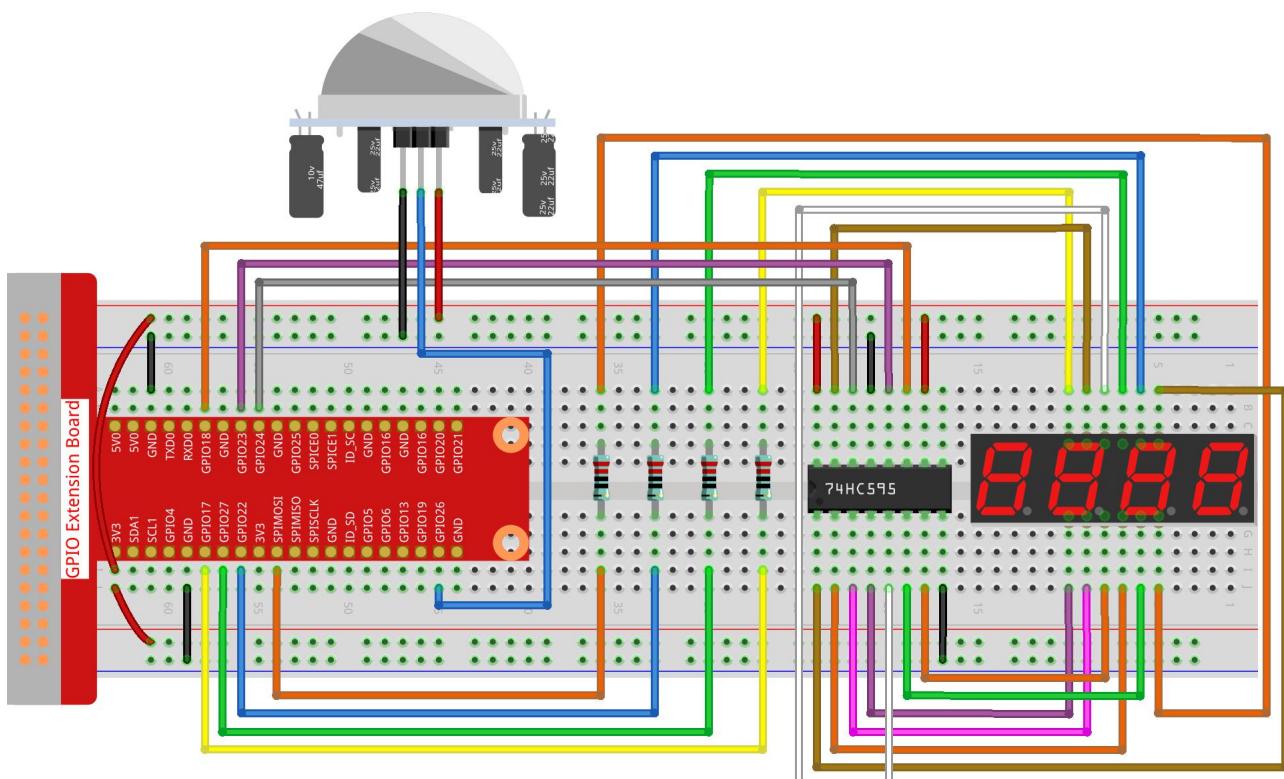
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO26	Pin 37	25	26



実験手順

ステップ1：回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.1/
```

ステップ 3: コードをコンパイルする。

```
gcc 3.1.1_CountingDevice.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行後、PIR が誰かが通り過ぎていることを検出すると、4 行のセグメントディスプレイの数字に 1 が加算される。

コードの説明

```
void display()
{
    clearDisplay();
    pickDigit(0);
    hc595_shift(number[counter % 10]);

    clearDisplay();
    pickDigit(1);
    hc595_shift(number[counter % 100 / 10]);

    clearDisplay();
    pickDigit(2);
    hc595_shift(number[counter % 1000 / 100]);

    clearDisplay();
    pickDigit(3);
    hc595_shift(number[counter % 10000 / 1000]);
}
```

まず、4 番目のセグメントディスプレイを開始し、1 行の数字を書き込む。次に、3 番目のセグメントディスプレイを開始し、10 行の数字を入力する。その後、2 番目と 1 番目のセグメントディスプレイをそれぞれ開始し、それぞれ数百桁と数千桁を書き込む。リフレッシュ速度が非常に速いため、完全な 4 行のディスプレイが表示される。

```
void loop(){
    int currentState =0;
    int lastState=0;
```

```
while(1){  
    display();  
    currentState=digitalRead(sensorPin);  
    if((currentState==0)&&(lastState==1)){  
        counter +=1;  
    }  
    lastState=currentState;  
}  
}
```

これが主な機能である：4桁のセグメントディスプレイに数字を表示し、PIR 値を読み取る。PIR が誰かが通り過ぎていることを検出すると、4桁のセグメントディスプレイの数字に1が加算される。

➤ Python 言語ユーザー向け

ステップ2：コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3：EXE ファイルを実行する。

```
sudo python3 3.1.1_CountingDevice.py
```

コードの実行後、PIR が誰かが通り過ぎていることを検出すると、4桁のセグメントディスプレイの数字に1が加算される。

コードの説明

1.1.5 4桁7セグメントディスプレイに基づいて、このレッスンでは、PIR モジュールを追加して、レッスン 1.1.5 の自動カウントをカウント検出に変更する。PIR が誰かが通り過ぎていることを検出すると、4桁のセグメントディスプレイの数字に1が加算される。

```
def display():  
    global counter  
    clearDisplay()  
    pickDigit(0)  
    hc595_shift(number[counter % 10])  
  
    clearDisplay()  
    pickDigit(1)  
    hc595_shift(number[counter % 100//10])  
  
    clearDisplay()  
    pickDigit(2)
```

```
hc595_shift(number[counter % 1000//100])
```

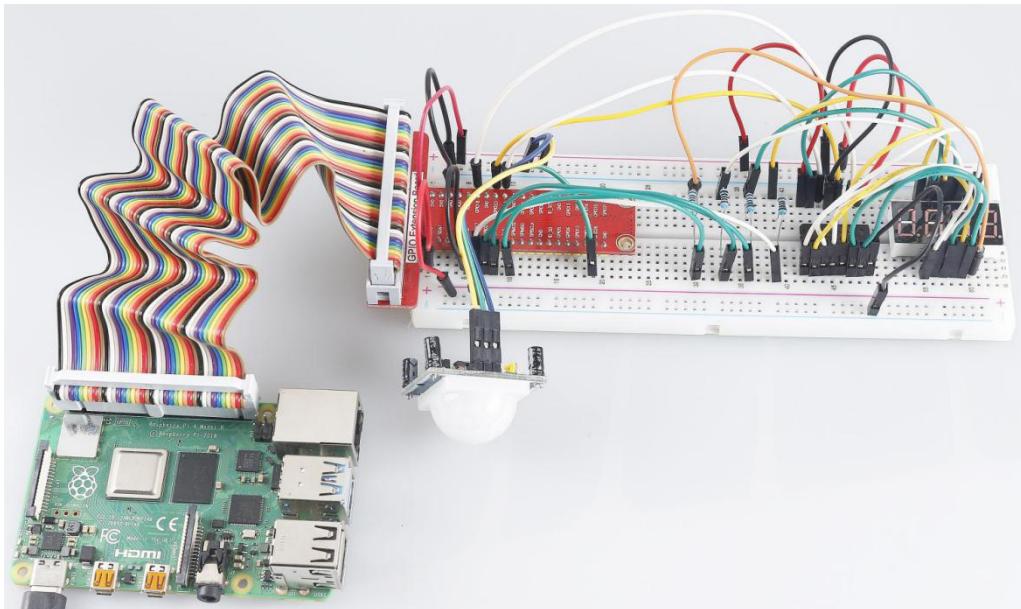
```
clearDisplay()
pickDigit(3)
hc595_shift(number[counter % 10000//1000])
```

まず、4番目のセグメント表示を開始し、1桁の数字を書き込む。次に、3番目のセグメントディスプレイを開始し、10桁の数字を入力する。その後、2番目と1番目のセグメントディスプレイをそれぞれ開始し、それぞれ数百桁と数千桁を書き込む。リフレッシュ速度が非常に速いため、完全な4桁のディスプレイが表示される。

```
def loop():
global counter
currentState = 0
lastState = 0
while True:
    display()
    currentState=GPIO.input(sensorPin)
    if (currentState == 0) and (lastState == 1):
        counter +=1
    lastState=currentState
```

これが主な機能である：4桁のセグメントディスプレイに数字を表示し、PIR値を読み取る。PIRが誰かが通り過ぎていることを検出すると、4桁のセグメントディスプレイの数字に1が加算される。

現象画像

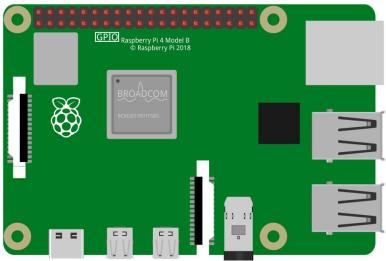
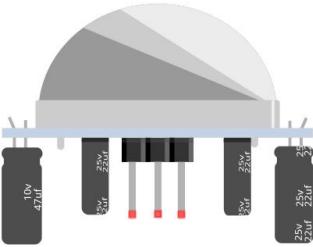
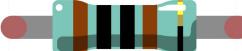
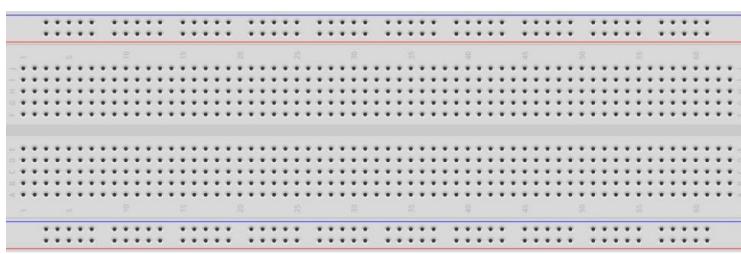


3.1.2 Welcome

前書き

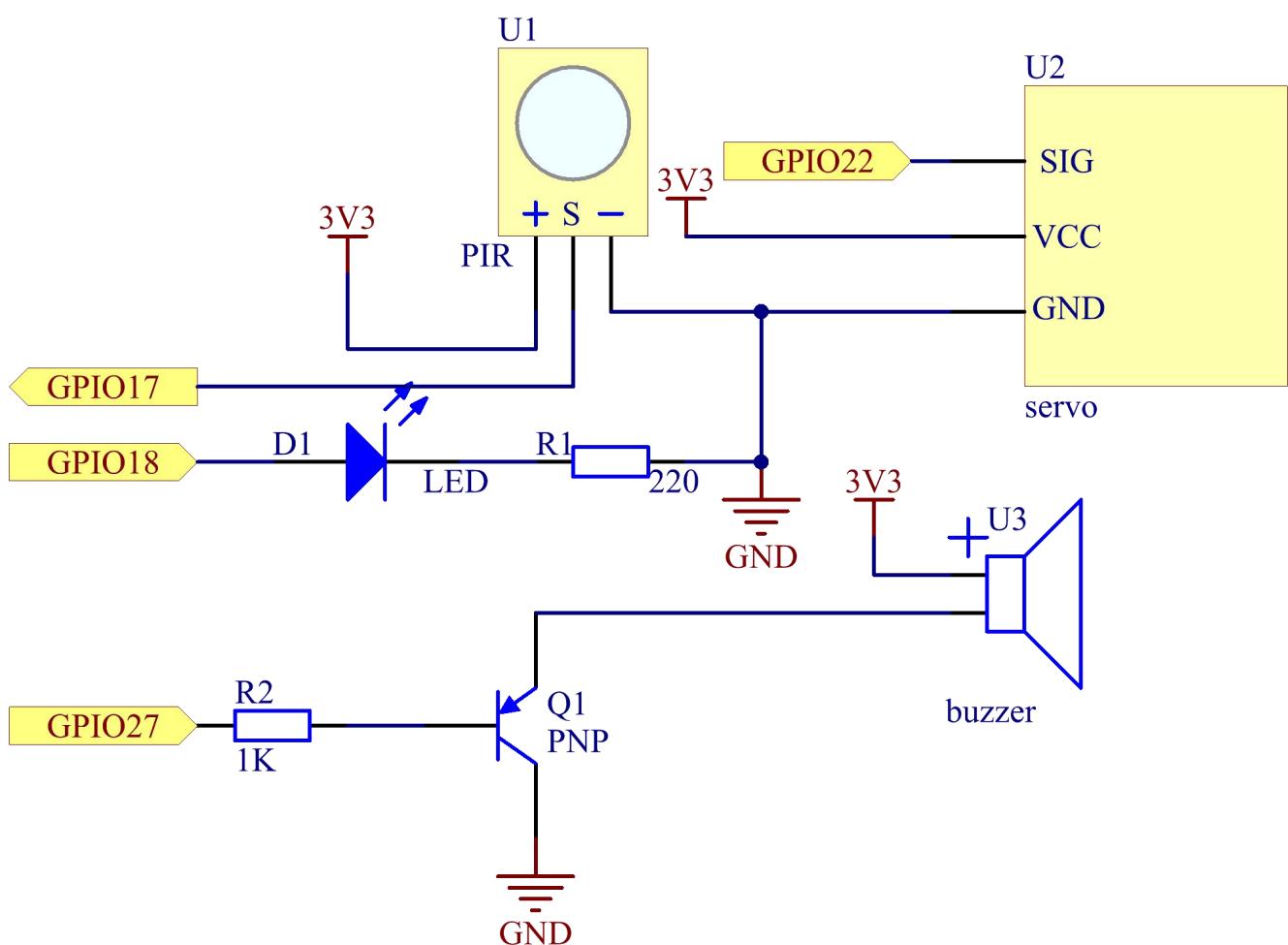
このプロジェクトでは、PIR を使用して歩行者の動きを検知し、サーボ、LED、ブザーを使用してコンビニのセンサードアの動作をシミュレートする。歩行者が PIR の検知範囲内に現れると、インジケータライトが点灯し、ドアが開き、ブザーがオープニングベルを鳴らす。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	PIR*1			
					
サーボ *1	パッシブブザー*1	S8550 PNP トランジスタ*1	LED*1		
					
抵抗器 (1KΩ) *1	抵抗器 (220Ω) *1	何本のジャンパー線			
					
40 ピンケーブル*1					
ブレッドボード*1					

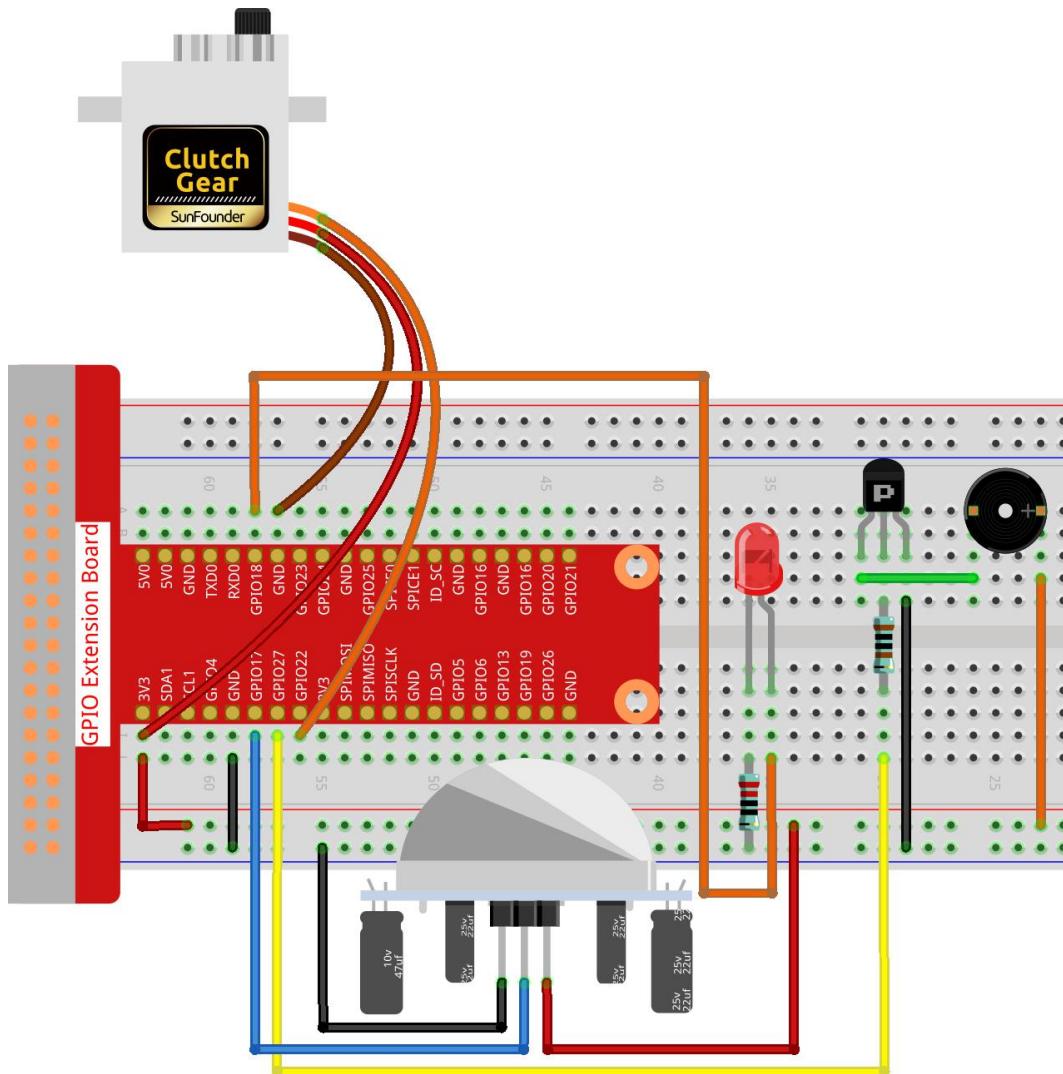
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



実験手順

ステップ1：回路を作る。



➤ C言語ユーザー向け

ステップ2：ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.2/
```

ステップ3：コンパイルする。

```
gcc 3.1.2_Welcome.c -lwiringPi
```

ステップ4：実行する。

```
sudo ./a.out
```

コードの実行後、PIRセンサーが通り過ぎる人を検出すると、ドアが自動的に開き（サーボによってシミュレートされる）、インジケーターをオンにして、ドアベルの音楽を再生する。ドアベルの音楽が再生されると、システムは自動的にドアを閉じてインジケータライトをオフにし、次に誰かが通り過ぎることを待つ。

コードの説明

```
void setAngle(int pin, int angle){ //Create a function to control the angle of the servo.
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin, Map(angle, 0, 180, 5, 25));
}
```

0～180 の角度をサーボに書き込むための関数、setAngle()を作成する。

```
void doorbell(){
    for(int i=0;i<sizeof(song)/4;i++){
        softToneWrite(BuzPin, song[i]);
        delay(beat[i] * 250);
}
```

ブザーで音楽を再生できるようにする関数、doorbell()を作成する。

```
void closedoor(){
    digitalWrite(ledPin, LOW); //led off
    for(int i=180;i>-1;i--){ //make servo rotate from maximum angle to minimum angle
        setAngle(servoPin,i);
        delay(1);
    }
}
```

ドアの閉鎖をシミュレートする関数 closedoor を作成し、LED をオフにし、サーボを 180 度から 0 度に回転させる。

```
void opendoor(){
    digitalWrite(ledPin, HIGH); //led on
    for(int i=0;i<181;i++){ //make servo rotate from minimum angle to maximum angle
        setAngle(servoPin,i);
        delay(1);
    }
    doorbell();
    closedoor();
}
```

関数 opendoor()にはいくつかの部分が含まれている：インジケータライトをオンにし、サーボを回転させ（ドアを開く動作をシミュレートする）、コンビニのドアベル音楽を再生し、音楽を再生した後に関数 closedoor()を呼び出す。

```
int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    if(softToneCreate(BuzPin) == -1){
        printf("setup softTone failed !");
        return 1;
    }
    .....
}
```

関数 main () で、ライブラリー wiringPi を初期化し、softTone をセットアップしてから、ledPin を出力状態に、pirPin を入力状態に設定する。PIR センサーが通り過ぎる人を検出すると、ドアを開くことをシミュレートするために関数 opendoor が呼び出される。

➤ Python 言語ユーザー向け

ステップ 2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 3.1.2_Welcome.py
```

コードの実行後、PIR センサーが通り過ぎる人を検出すると、ドアが自動的に開き（サーボによってシミュレートされる）、インジケーターをオンにして、ドアベルの音楽を再生する。ドアベルの音楽が再生されると、システムは自動的にドアを閉じてインジケータライトをオフにし、次に誰かが通り過ぎることを待つ。

コードの説明

```
def setup():
    global p
    global Buzz
                           # Assign a global variable to replace GPIO.PWM
    GPIO.setmode(GPIO.BCM)      # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT) # Set ledPin's mode is output
    GPIO.setup(pirPin, GPIO.IN)  # Set sensorPin's mode is input
    GPIO.setup(buzPin, GPIO.OUT) # Set pins' mode is output
    Buzz = GPIO.PWM(buzPin, 440) # 440 is initial frequency.
    Buzz.start(50)             # Start Buzzer pin with 50% duty ration
```

```
GPIO.setup(servoPin, GPIO.OUT)    # Set servoPin's mode is output
GPIO.output(servoPin, GPIO.LOW)   # Set servoPin to low
p = GPIO.PWM(servoPin, 50)       # set Freqeuce to 50Hz
p.start(0)                      # Duty Cycle = 0
```

これらのステートメントは、各部品のピンを初期化するために使用される。

```
def setAngle(angle):          # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it
```

0～180 の角度をサーボに書き込むための関数、setAngle()を作成する。

```
def doorbell():
    for i in range(1, len(song)):      # Play song 1
        Buzz.ChangeFrequency(song[i])  # Change the frequency along the song note
        time.sleep(beat[i] * 0.25)     # delay a note for beat * 0.25s
```

ブザーで音楽を再生できるようにする関数、doorbell()を作成する。

```
def closedoor():
    GPIO.output(ledPin, GPIO.LOW)
    Buzz.ChangeFrequency(1)
    for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
        setAngle(i)
        time.sleep(0.001)
```

ドアを閉じて、インジケータライトをオフにする。

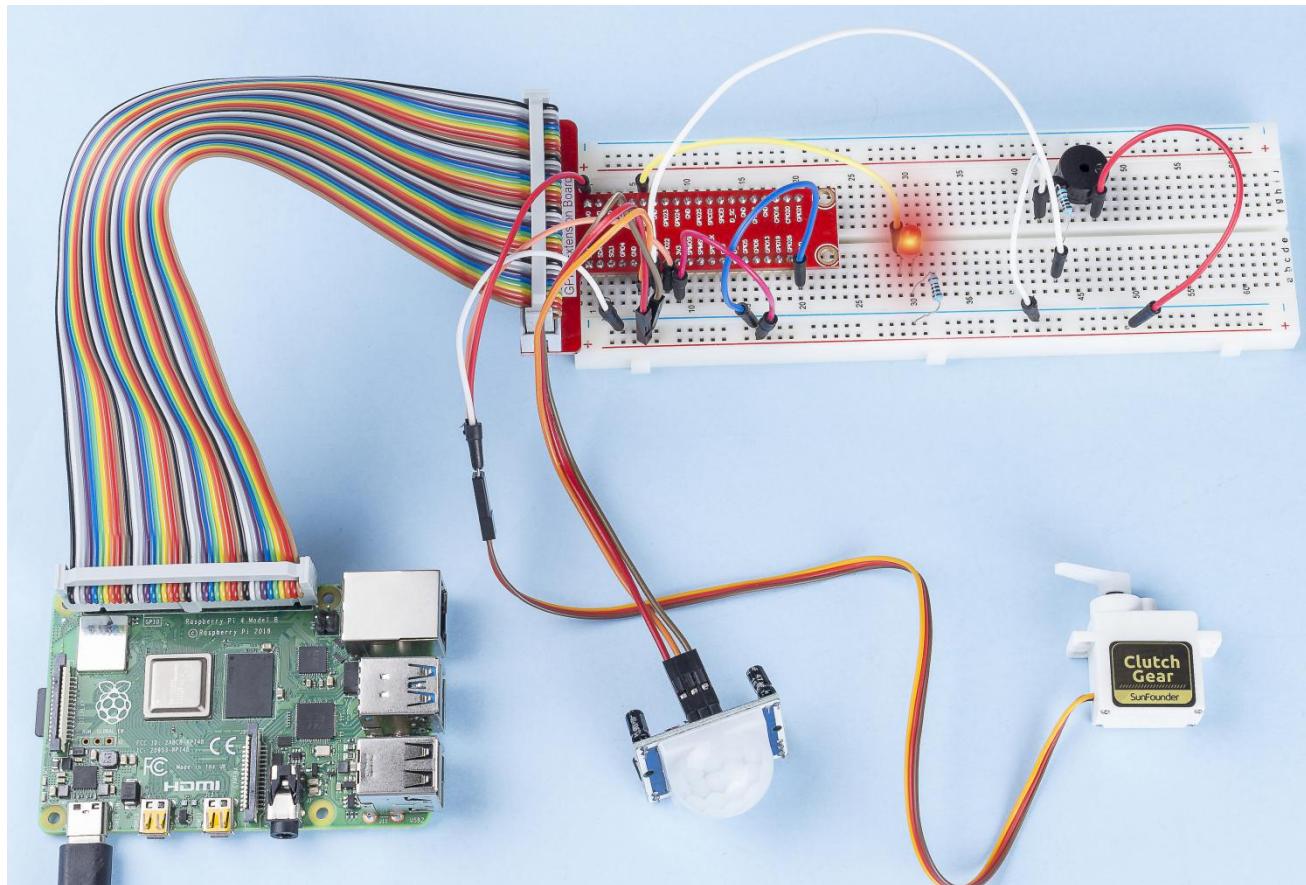
```
def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1):  #make servo rotate from 0 to 180 deg
        setAngle(i)      # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()
```

関数 opendoor() にはいくつかの部分が含まれている： インジケータライトをオンにし、サーボを回転させ（ドアを開く動作をシミュレートする）、コンビニのドアベル音楽を再生し、音楽を再生した後に関数 closedoor() を呼び出す。

```
def loop():
while True:
    if GPIO.input(pirPin)==GPIO.HIGH:
        opendoor()
```

RPI は誰かが通り過ぎることを検知すると、関数 opendoor() を呼び出す。

現象画像

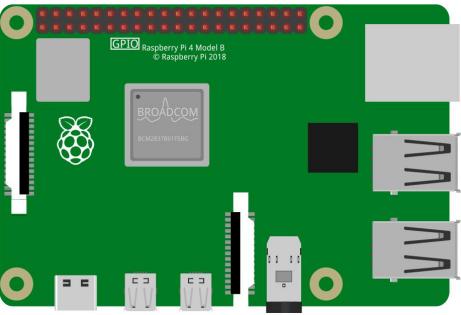
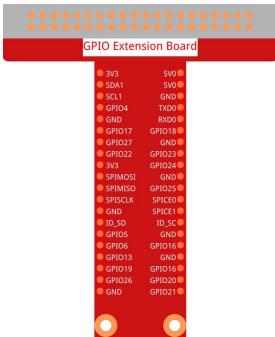
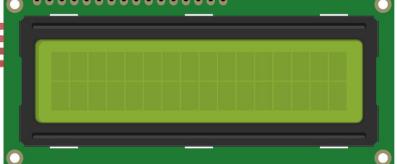


3.1.3 Reversing Alarm

前書き

このプロジェクトでは、LCD、ブザー、と超音波センサーを使用して、後退補助システムを作成する。それをリモートコントロール車両に置いて、車をガレージに後退する実際のプロセスをシミュレートできる。

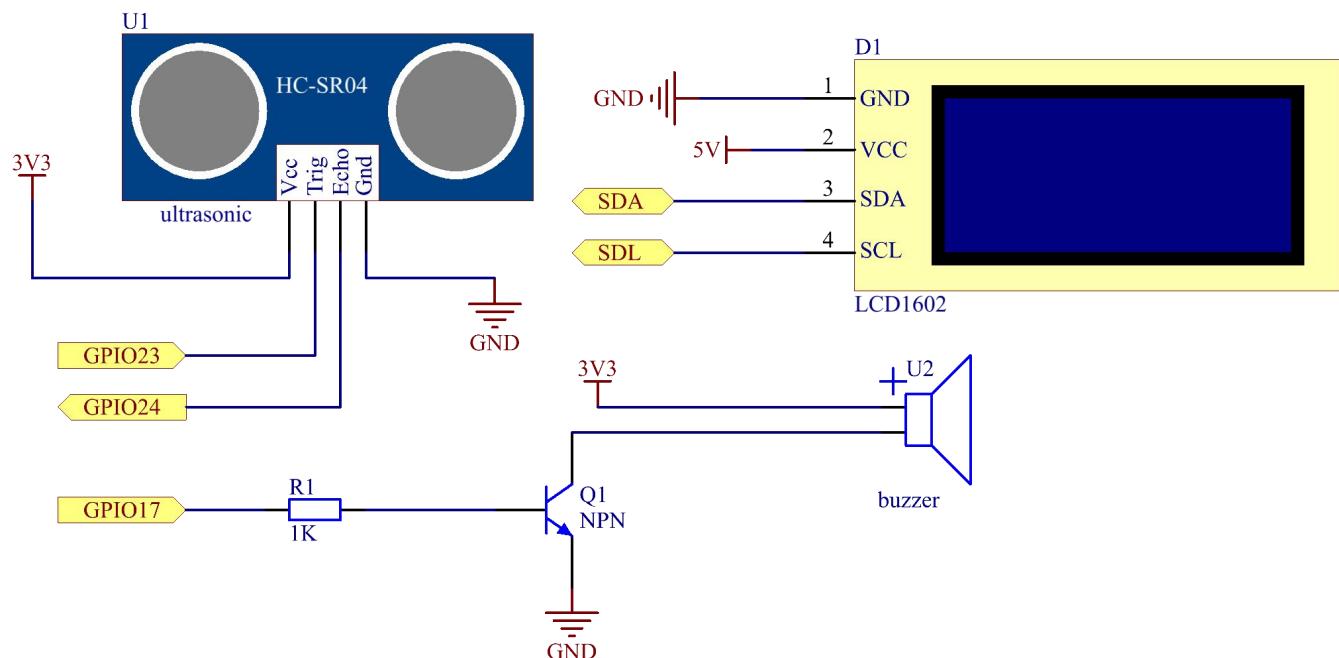
部品

Raspberry Pi 本体*1	T 拡張ボード*1	アクティブブザー*1
	 GPIO Extension Board Pinout: S9 S10 GND GND SCL1 SCL0 GPIO4 TXD0 GND RXD0 GPIO17 GPIO18 GND GPIO23 GPIO22 GPIO24 3V3 GND SPI MOSI GND SPI MISO GPIO25 GND SPI CE0 ID_SD ID_SD GND GND GPIO6 GPIO16 GPIO13 GND GPIO19 GPIO18 GND GPIO20 GND GPIO21	
HC SR04*1	I2C LCD1602*1	S8050 NPN トランジスター*1
		
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		抵抗器 (1kΩ) *1
		

回路図

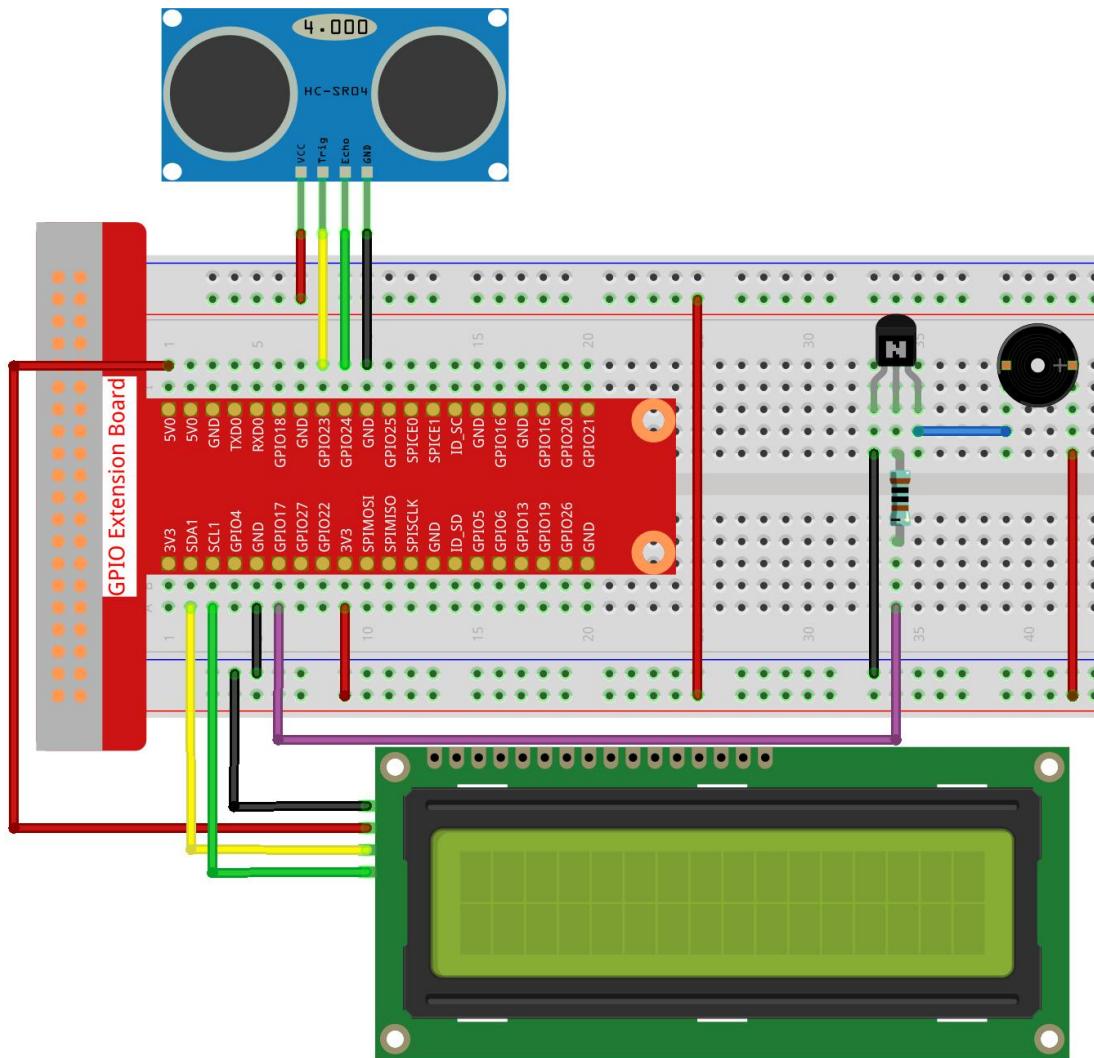
超音波センサーは障害物との間の距離を検出し、コードの形式で LCD に表示する。同時に、超音波センサーにより、距離値に応じてブザーが異なる周波数の警告音を発する。

T ボード名	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.3/
```

ステップ3: コンパイルする

```
gcc 3.1.3_ReversingAlarm.c -lwiringPi
```

ステップ4: 実行する。

```
sudo ./a.out
```

コードが実行されると、超音波センサモジュールが障害物までの距離を検出し、LCD1602に距離に関する情報を表示する。また、ブザーは距離とともに周波数が変化する警告音を発する。

コード

ご注意: 次のコードは不完全である。完全なコードを確認する場合は、コマンド nano 3.1.1_ReversingAlarm.c を使用することをお勧めする。

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <string.h>

#define Trig    4
#define Echo    5
#define Buzzer  0

int LCDAddr = 0x27;
int BLEN = 1;
int fd;

//here is the function of LCD
void write_word(int data){.....}

void send_command(int comm){.....}

void send_data(int data){.....}

void lcdInit(){.....}

void clear(){.....}

void write(int x, int y, char data[]){.....}

//here is the function of Ultrasonic
void ultralnit(void){.....}

float disMeasure(void){.....}

//here is the main function
int main(void)
{
```

```

float dis;
char result[10];
if(wiringPiSetup() == -1){
    printf("setup wiringPi failed !");
    return 1;
}

pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();

clear();
write(0, 0, "Ultrasonic Starting");
write(1, 1, "By Sunfounder");

while(1){
    dis = disMeasure();
    printf("%.2f cm \n",dis);
    digitalWrite(Buzzer,LOW);
    if (dis > 400){
        clear();
        write(0, 0, "Error");
        write(3, 1, "Out of range");
        delay(500);
    }
    else
    {
        clear();
        write(0, 0, "Distance is");
        sprintf(result,"% .2f cm",dis);
        write(5, 1, result);

        if(dis>=50)
        {delay(500);}
        else if(dis<50 & dis>20) {
            for(int i=0;i<2;i++){
                digitalWrite(Buzzer,HIGH);
                delay(50);
                digitalWrite(Buzzer,LOW);
            }
        }
    }
}

```

```
    delay(200);
}
}

else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}

return 0;
}
```

コードの説明

```
pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();
```

このプログラムでは、以前の部品を総合的に適用する。ここでは、ブザー、LCD、と超音波を使用する。以前と同じ方法で初期化できる。

```
dis = disMeasure();
printf("%.2f cm \n",dis);
digitalWrite(Buzzer,LOW);
if (dis > 400){
    write(0, 0, "Error");
    write(3, 1, "Out of range");
}
else
{
    write(0, 0, "Distance is");
    sprintf(result,"% .2f cm",dis);
    write(5, 1, result);
}
```

ここで、超音波センサーの値を取得し、計算により距離を取得する。

距離の値が検出される範囲の値より大きい場合、エラーメッセージが LCD に表示される。距離値が範囲内にある場合、対応する結果が出力される。

```
sprintf(result,"%.2f cm",dis);
```

LCD の出力モードは文字型のみをサポートし、変数 dis は float 型の値を保存するため、sprintf() を使わなければならない。この関数は float 型の値を文字に変換し、文字列変数 result[] に保存する。%.2f は小数点以下 2 衔を保持することを意味する。

```
if(dis>=50)
{delay(500);}
else if(dis<50 & dis>20) {
    for(int i=0;i<2;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(200);
    }
}
else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}
```

この判定条件はブザーの音を制御するために使用される。距離の違いに応じて、3 つのケースに分けることができる。この場合、音の周波数は異なる。遅延の合計値は 500 であるため、すべてのケースで超音波センサーに 500ms の間隔を提供できる。

➤ Python 言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: 実行する。

```
sudo python3 3.1.3_ReversingAlarm.py
```

コードが実行されると、超音波センサーモジュールが障害物までの距離を検出し、LCD1602に距離に関する情報を表示する。また、ブザーは距離とともに周波数が変化する警告音を発する

コード

```
import LCD1602
import time
import RPi.GPIO as GPIO

TRIG = 16
ECHO = 18
BUZZER = 11

def lcdsetup():
    LCD1602.init(0x27, 1)      # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'Ultrasonic Starting')
    LCD1602.write(1, 1, 'By SunFounder')
    time.sleep(2)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
    lcdsetup()

def distance():
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)

    GPIO.output(TRIG, 1)
    time.sleep(0.00001)
```

```

GPIO.output(TRIG, 0)

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def destroy():
    GPIO.output(BUZZER, GPIO.LOW)
    GPIO.cleanup()
    LCD1602.clear()

def loop():
    while True:
        dis = distance()
        print(dis, 'cm')
        print(')')
        GPIO.output(BUZZER, GPIO.LOW)
        if (dis > 400):
            LCD1602.clear()
            LCD1602.write(0, 0, 'Error')
            LCD1602.write(3, 1, 'Out of range')
            time.sleep(0.5)
        else:
            LCD1602.clear()
            LCD1602.write(0, 0, 'Distance is')
            LCD1602.write(5, 1, str(round(dis,2)) + ' cm')
            if(dis>=50):
                time.sleep(0.5)
            elif(dis<50 and dis>20):
                for i in range(0,2,1):
                    GPIO.output(BUZZER, GPIO.HIGH)
                    time.sleep(0.05)
                    GPIO.output(BUZZER, GPIO.LOW)
                    time.sleep(0.2)

```

```
elif(dis<=20):
    for i in range(0,5,1):
        GPIO.output(BUZZER, GPIO.HIGH)
        time.sleep(0.05)
        GPIO.output(BUZZER, GPIO.LOW)
        time.sleep(0.05)

if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
def lcdsetup():
    LCD1602.init(0x27, 1)    # init(slave address, background light)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
    lcdsetup()
```

このプログラムでは、以前に使用した部品を総合的に適用する。ここでは、ブザー、LCD、と超音波を使用する。以前と同じ方法で初期化できる。

```
dis = distance()
print (dis, 'cm')
print ('')
GPIO.output(BUZZER, GPIO.LOW)
if (dis > 400):
    LCD1602.clear()
    LCD1602.write(0, 0, 'Error')
    LCD1602.write(3, 1, 'Out of range')
    time.sleep(0.5)
else:
```

```
LCD1602.clear()
LCD1602.write(0, 0, 'Distance is')
LCD1602.write(5, 1, str(round(dis,2)) + ' cm')
```

ここで、超音波センサーの値を取得し、計算により距離を取得する。距離の値が検出される範囲の値より大きい場合、エラーメッセージが LCD に表示される。動作値が範囲内にある場合、対応する結果が出力される。

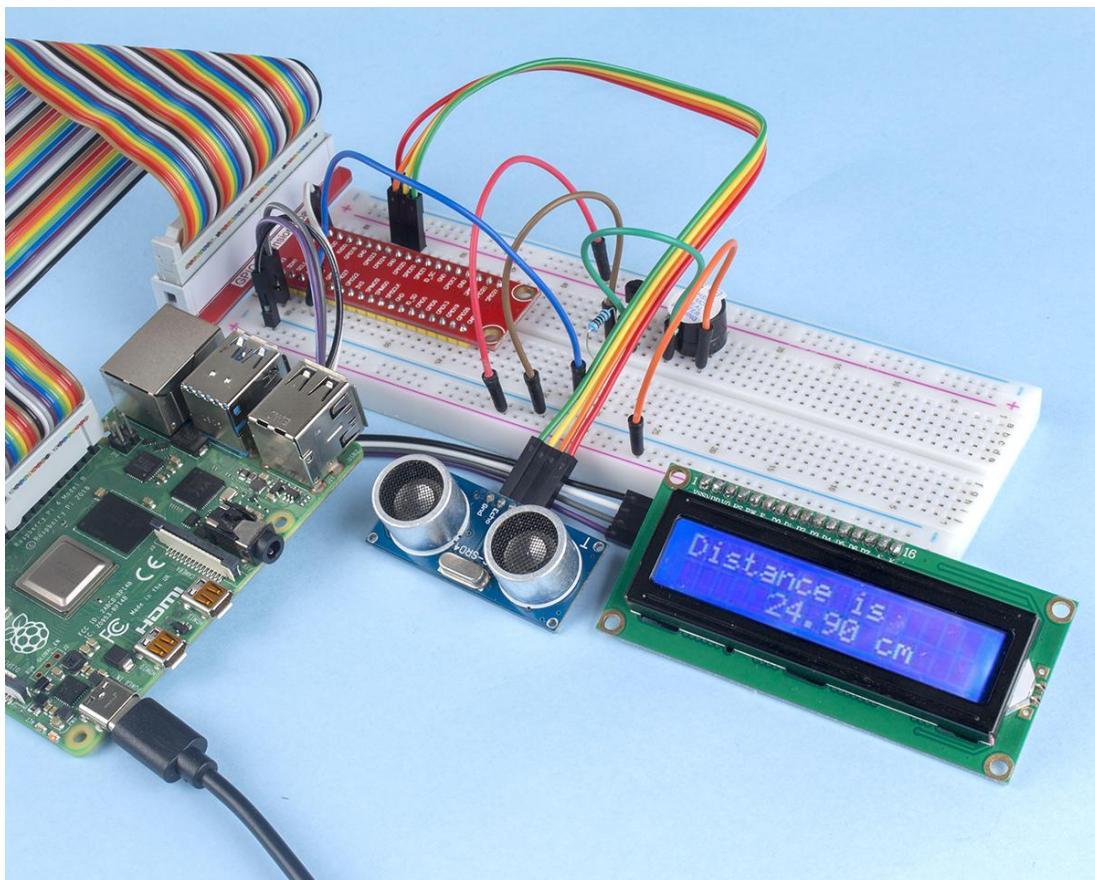
```
LCD1602.write(5, 1, str(round(dis,2)) + ' cm')
```

LCD 出力は文字タイプのみをサポートするため、str() を使用して数値を文字に変換する必要がある。小数点以下 2 衔に丸める。

```
if(dis>=50)
{delay(500);}
else if(dis<50 & dis>20) {
    for(int i=0;i<2;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(200);
    }
}
else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}
```

この判定条件はブザーの音を制御するために使用される。距離の違いに応じて、3 つのケースに分けることができる。この場合、音の周波数は異なる。遅延の合計値は 500 であるため、すべてのケースで超音波センサーに 500ms の間隔を提供できる。

現象画像

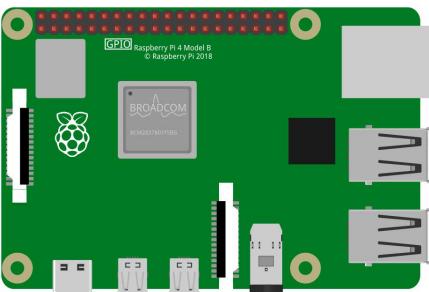
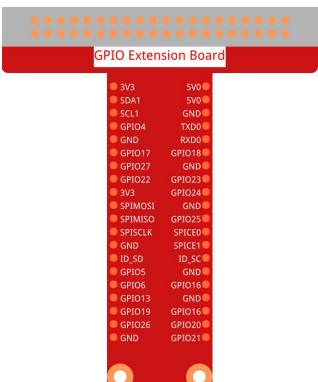
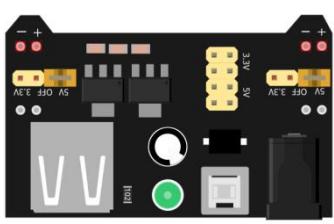
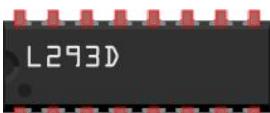
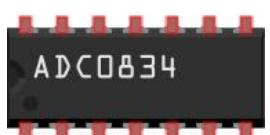
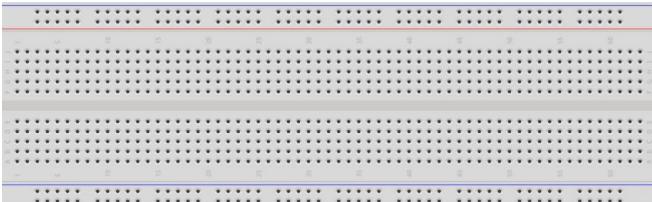
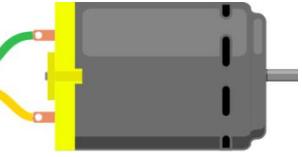


3.1.4 Smart Fan

前書き

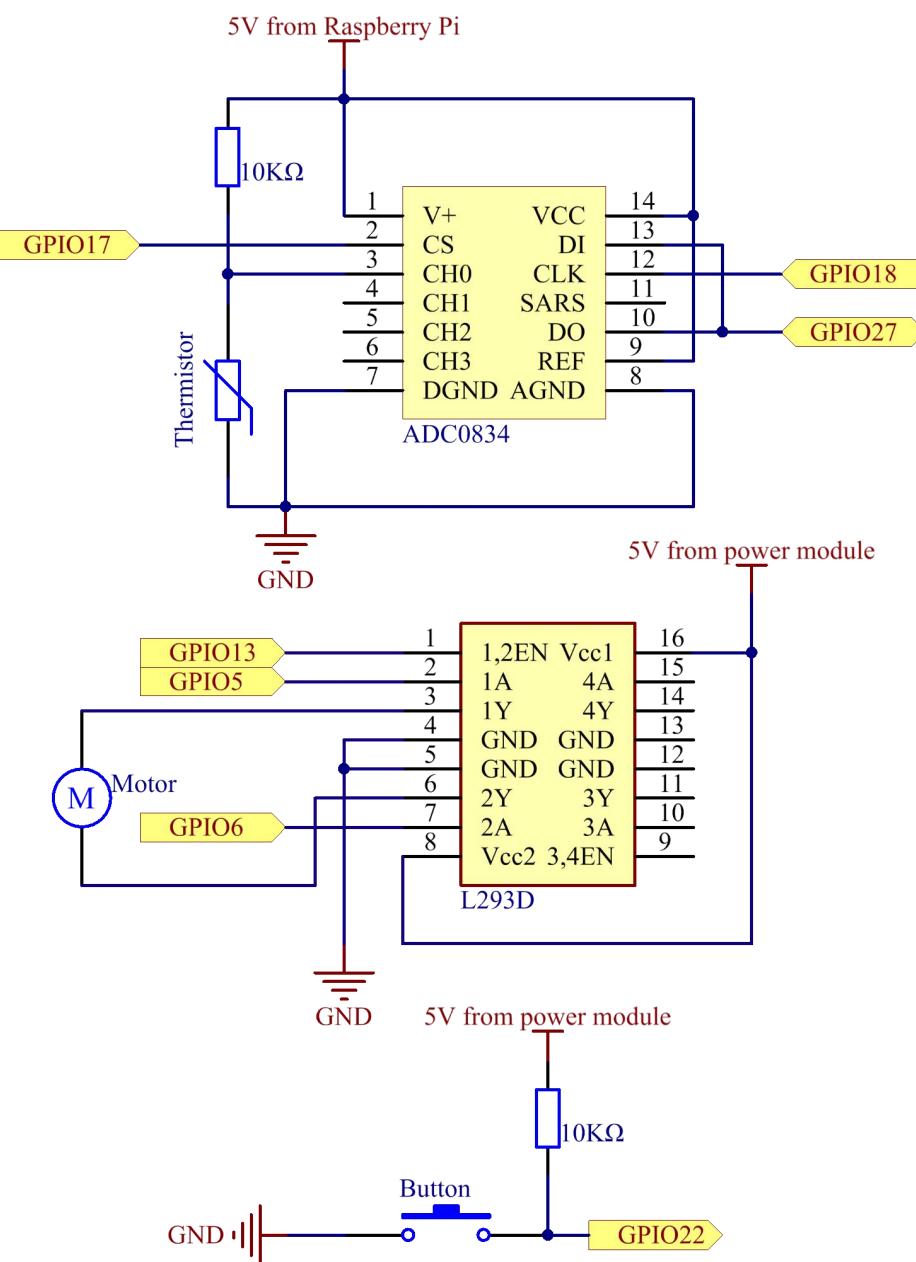
このコースでは、モーター、ボタン、サーミスターを使用して、風速が調整可能な手動+自動のスマートファンを作成する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	電源モジュール*1
	 GPIO Extension Board Pinout: 3V3 SDA1 SDA1 GND SCL1 SCL1 GPIO4 TXD0 TXD0 GND RXD0 RXD0 GPIO17 GPIO18 GPIO18 GND GND GND GPIO27 GPIO23 GPIO23 GND GPIO24 GPIO24 GND SPIMOSI SPIMOSI SPIMOSI GND SPIMISO SPIMISO SPIMISO GND SPICLK SPICLK SPICLK GND ID_SD ID_SD ID_SD GND ID_SC ID_SC ID_SC GPIO5 GND GPIO16 GND GPIO6 GND GPIO16 GND GPIO13 GND GPIO16 GND GND GPIO19 GPIO20 GND GND GPIO26 GND GND	
40 ピンケーブル*1	サーミスタ *1	抵抗器 (10KΩ) *1 
		L293D*1 
ブレッドボード*1	ボタン*1	ADC0834*1 
		DC モーター*1 
		何本のジャンパー線 

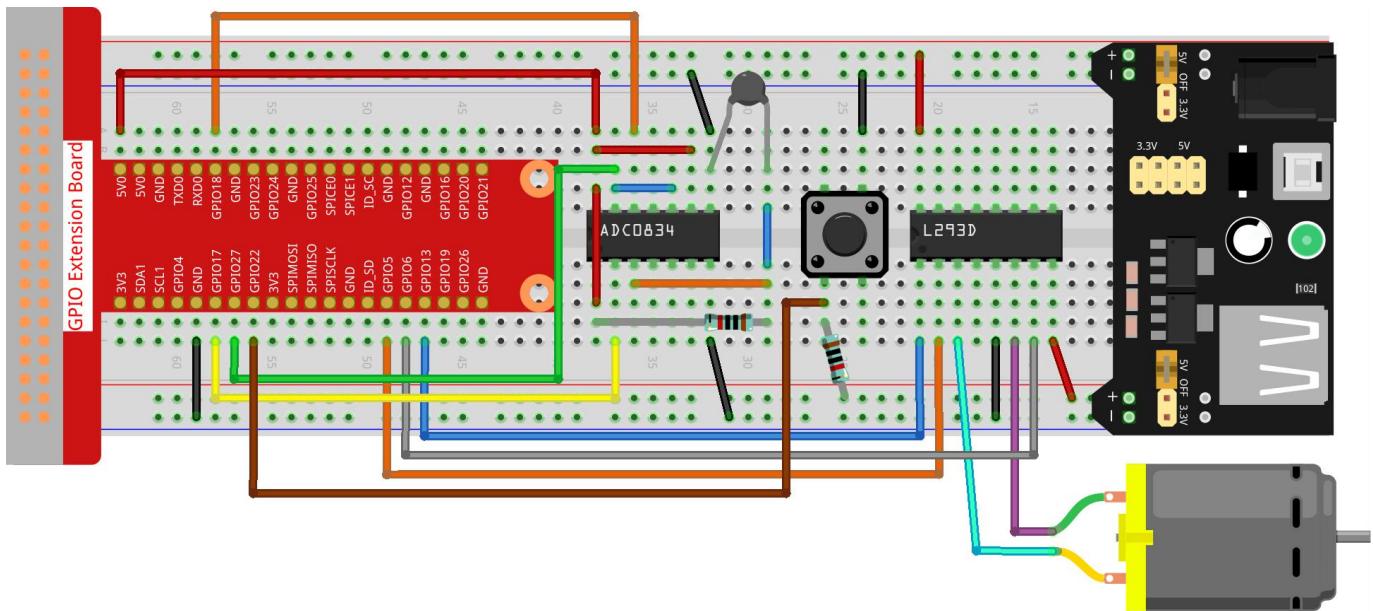
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13



実験手順

ステップ1: 回路を作る。



ご注意: 電源モジュールはキットの9Vバッテリーバックルで9Vバッテリーを適用できる。電源モジュールのジャンパキャップをブレッドボードの5Vバスストリップに挿入する。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.4/
```

ステップ3: コンパイルする。

```
gcc 3.1.4_SmartFan.c -lwiringPi -lm
```

ステップ4: EXEファイルを実行する。

```
sudo ./a.out
```

コードが実行された後、ボタンを押してファンを起動する。押すたびに、1つのレベルが上下に調整される。5種類のレベルがある：0~4。4番目のレベルに設定してボタンを押すと、ファンの風速が0で動作が停止する。

温度が2°C以上上昇または下降すると、速度は自動的に1グレード速くまたは遅くなる。

コードの説明

```
int temperture(){
    unsigned char analogVal;
    double Vr, Rt, temp, cel, Fah;
    analogVal = get_ADC_Result(0);
    Vr = 5 * (double)(analogVal) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    int t=cel;
    return t;
}
```

Temperture()は、ADC0834 が読み取ったサーミスタの値を温度値に変換することで機能する。詳細については、2.2.2 Thermistor を参照してください。

```
int motor(int level){
    if(level==0){
        digitalWrite(MotorEnable,LOW);
        return 0;
    }
    if (level>=4){
        level =4;
    }
    digitalWrite(MotorEnable,HIGH);
    softPwmWrite(MotorPin1, level*25);
    return level;
}
```

この機能は、モーターの回転速度を制御する。レベルの範囲：0～4（レベル 0 は動作中のモーターを停止する）。1 つのレベル調整は風速の 25% の変化を表す。

```
int main(void)
{
    setup();
    int currentState,lastState=0;
    int level = 0;
    int currentTemp,markTemp=0;
    while(1){
        currentState=digitalRead(BtnPin);
        currentTemp=temperture();
```

```

if (currentTemp<=0){continue;}
if (currentState==1&&lastState==0){
    level=(level+1)%5;
    markTemp=currentTemp;
    delay(500);
}
lastState=currentState;
if (level!=0){
    if (currentTemp-markTemp<=-2){
        level=level-1;
        markTemp=currentTemp;
    }
    if (currentTemp-markTemp>=2){
        level=level+1;
        markTemp=currentTemp;
    }
}
level=motor(level);
}
return 0;
}

```

関数 main () には、次のようにプログラムプロセス全体が含まれている：

- 1) ボタンの状態と現在の温度を常に読み取る。
- 2) ボタンを押すごとに、レベル+1 になり、同時に温度が更新される。レベル範囲 1～4。
- 3) ファンが作動すると（レベルは 0 ではない）、温度は検出中である。2°C+以上変更すると、レベルが上下に変化する。
- 4) モーターはレベルに応じて回転速度を変更する。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: 実行する。

```
sudo python3 3.1.4_SmartFan.py
```

コードが実行された後、ボタンを押してファンを起動する。押すたびに、1つのレベルが上下に調整される。5種類のレベルがある：0～4。4番目のレベルに設定してボタンを押すと、ファンの風速が0で動作が停止する。

温度が2°C以上上昇または下降すると、速度は自動的に1グレード速くまたは遅くなる。

コードの説明

```
def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return Cel
```

temperature()は、ADC0834によって読み取られたサーミスタ値を温度値に変換することにより機能する。詳細については、[2.2.2 Thermistor](#)を参照してください。

```
def motor(level):
    if level == 0:
        GPIO.output(MotorEnable, GPIO.LOW)
        return 0
    if level >= 4:
        level = 4
    GPIO.output(MotorEnable, GPIO.HIGH)
    p_M1.ChangeDutyCycle(level*25)
    return level
```

この機能はモーターの回転速度を制御する。レバーの範囲：0～4（レベル0は動作中のモーターを停止する）。1つのレベル調整は風速の25%の変化を表す。

```
def main():
    lastState=0
    level=0
    markTemp = temperature()
    while True:
        currentState =GPIO.input(BtnPin)
        currentTemp=temperature()
        if currentState == 1 and lastState == 0:
            level=(level+1)%5
            markTemp = currentTemp
            time.sleep(0.5)
```

```

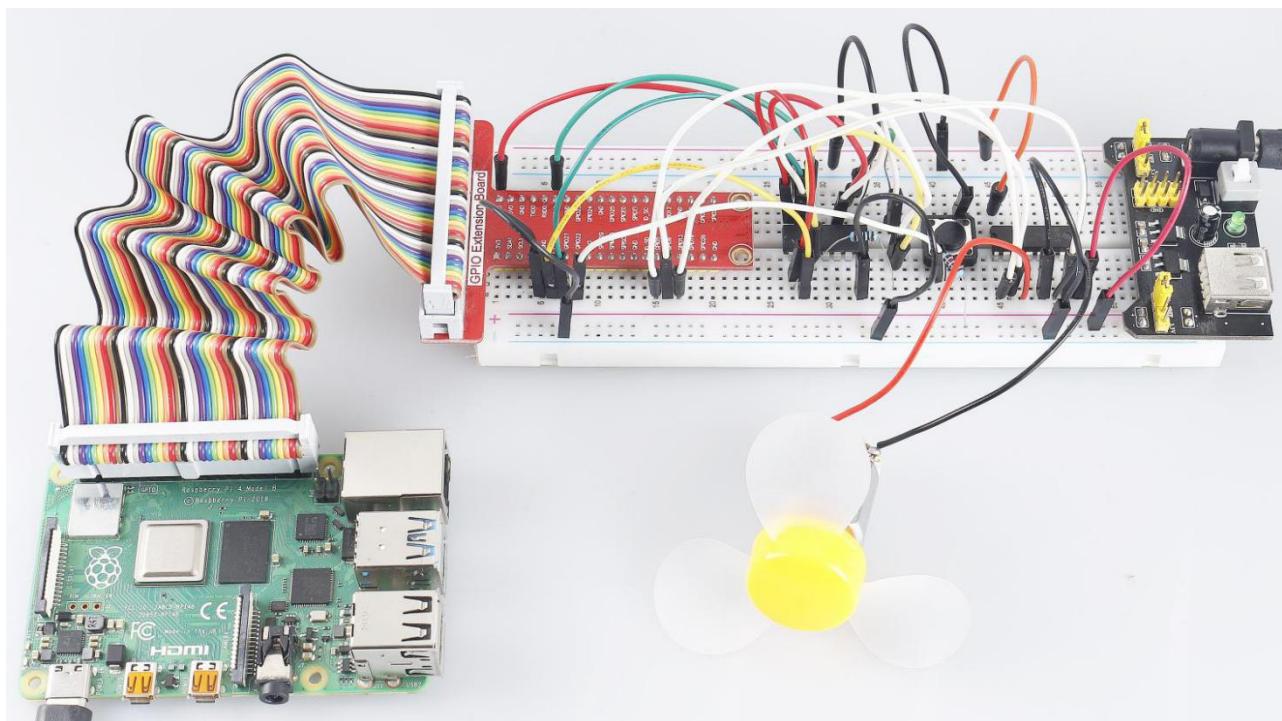
lastState=currentState
if level!=0:
    if currentTemp-markTemp <= -2:
        level = level -1
        markTemp=currentTemp
    if currentTemp-markTemp >= 2:
        level = level +1
        markTemp=currentTemp
level = motor(level)

```

関数 main () には、次のようにプログラムプロセス全体が含まれている：

- 1) ボタンの状態と現在の温度を常に読み取る。
- 2) ボタンを押すごとに、レベル+1 になり、同時に温度が更新される。レベル範囲 1~4。
- 3) ファンが作動すると（レベルは 0 ではない）、温度は検出中である。2°C+以上変更すると、レベルが上下に変化する。
- 4) モーターはレベルに応じて回転速度を変更する。

現象画像

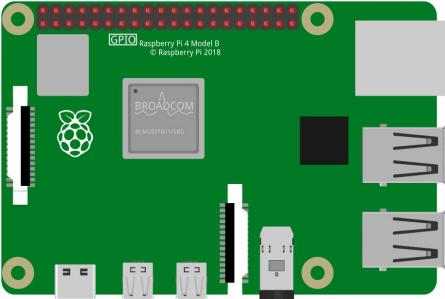
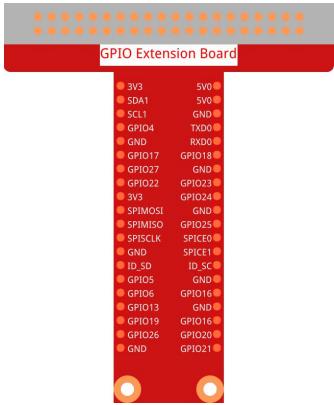
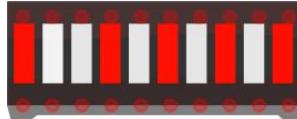
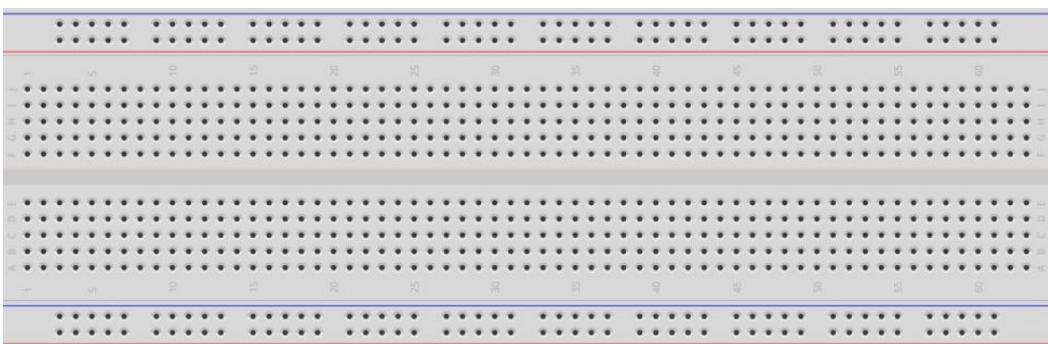


3.1.5 Battery Indicator

前書き

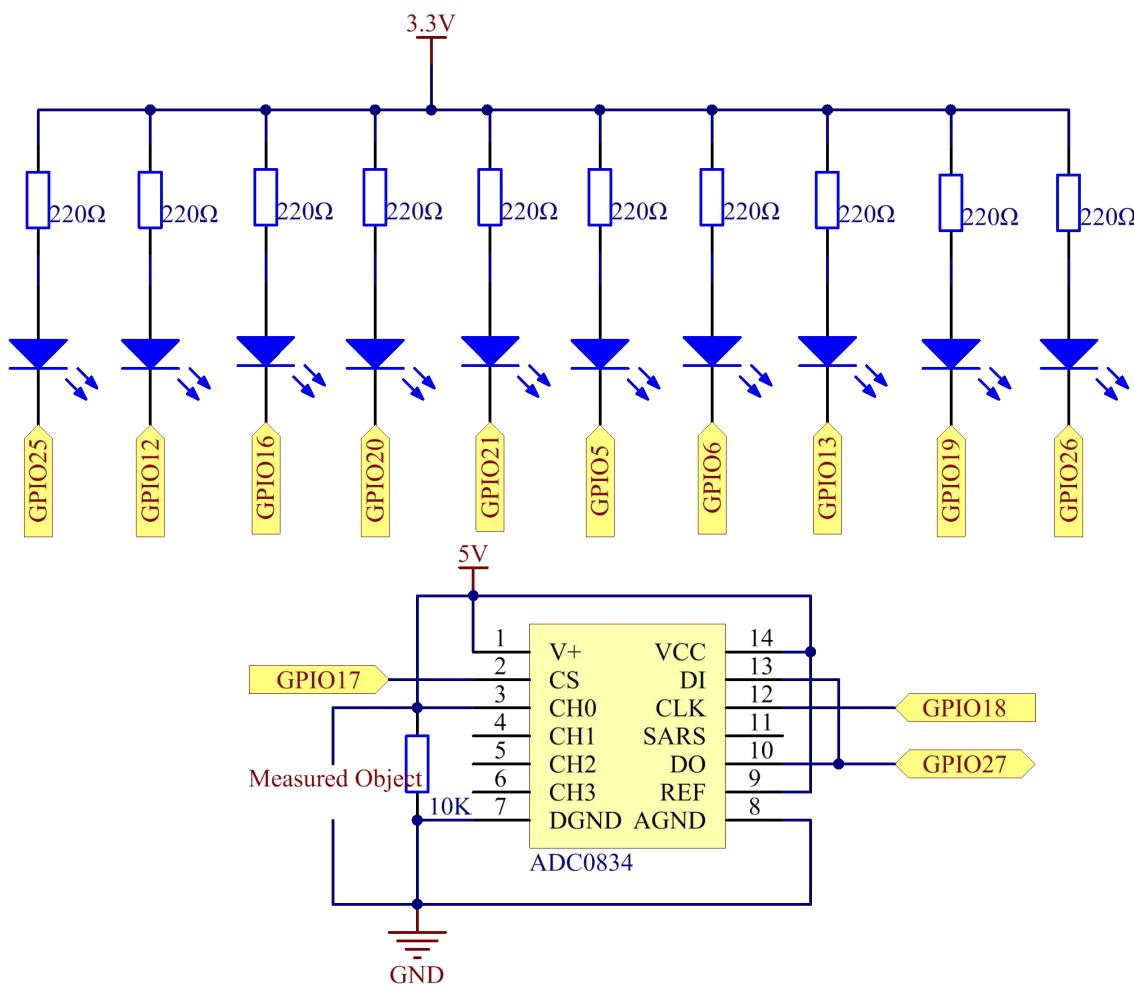
このコースでは、LED バーグラフにバッテリーレベルを視覚的に表示できるバッテリー指示装置を作成する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	LED 棒グラフ*1
	 GPIO Extension Board Pinout: ● V3.3 5V0 ● SDA1 5V0 ● SCL1 GND ● GPIO4 TXD0 ● GND RXD0 ● GPIO17 GPIO18 GND ● GPIO27 GPIO22 GND ● GPIO22 GPIO23 GND ● V3.3 GPIO24 GND ● SPI MOSI GND ● SPI MISO GPIO25 GND ● SPI SCLK SPI CE0 GND ● GND SPI CE1 GND ● ID_SD ID_SC GND ● GPIO5 GND ● GPIO6 GPIO16 GND ● GPIO13 GPIO16 GND ● GPIO19 GPIO16 GND ● GPIO26 GPIO20 GND ● GND GPIO21 GND	
40 ピンケーブル*1		ADC0834*1 
		抵抗器 (220Ω) *10 
ブレッドボード*1		何本のジャンパー線 
		

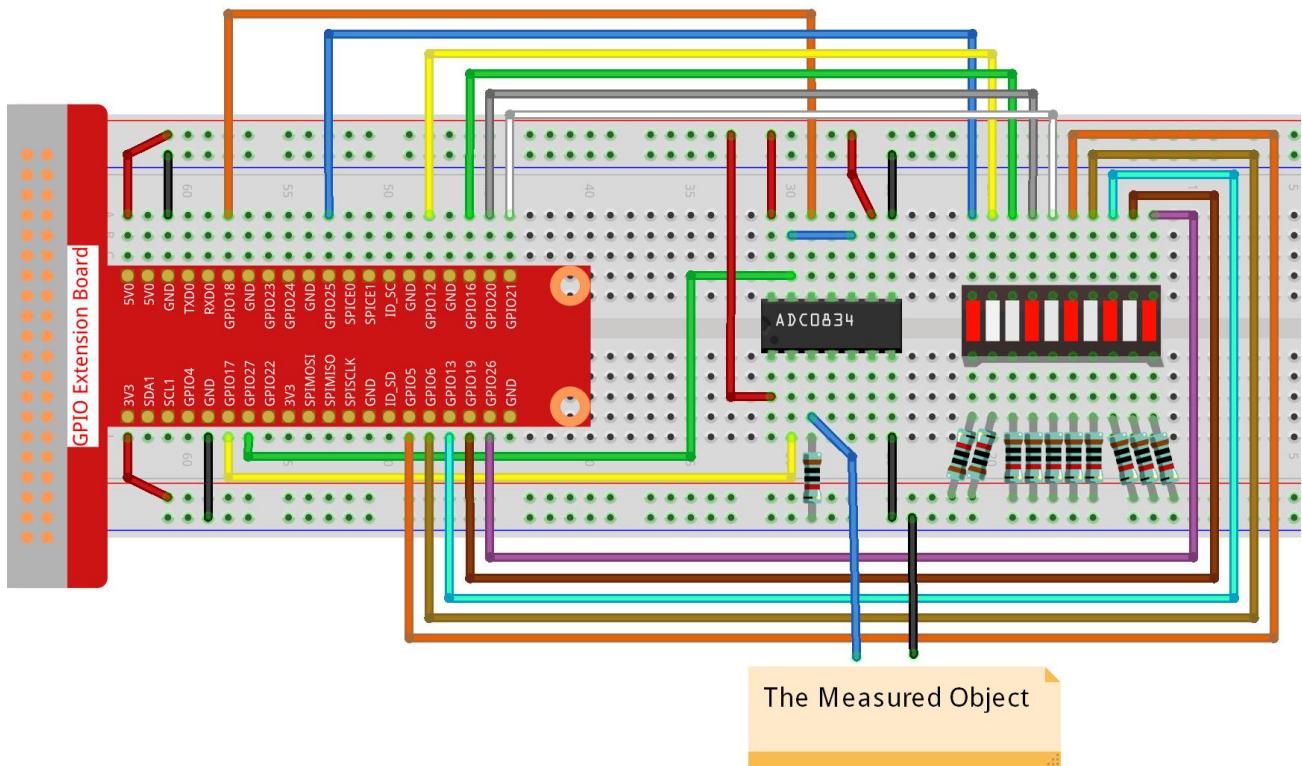
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO25	Pin 22	6	25
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	20
GPIO21	Pin 40	29	21
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13
GPIO19	Pin 35	24	19
GPIO26	Pin 37	25	26



実験手順

ステップ 1: 回路を作る。



C言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.5/
```

ステップ 3: コードをコンパイルする。

```
gcc 3.1.5_BatteryIndicator.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

プログラムの実行後、ADC0834 の 3 番目のピンと GND に個別にリード線を配線し、それらを別々にバッテリーの 2 つの極に導く。LED バーグラフの対応する LED が点灯し、電力レベルが表示される（測定範囲：0~5V）。

コードの説明

```
void LedBarGraph(int value){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
    }
    for(int i=0;i<value;i++){
        digitalWrite(pins[i],LOW);
    }
}
```

```
}
```

この機能は、LED 棒グラフの 10 個の LED の点灯/消灯を制御するために機能する。これらの 10 個の LED を最初にオフにするために高レベルを指定し、次に受信したアナログ値を変更することでいくつの LED を点灯させるかを決定する。

```
int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ADC_CS, OUTPUT);
    pinMode(ADC_CLK, OUTPUT);
    for(int i=0;i<10;i++){ //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],HIGH);
    }
    while(1){
        analogVal = get_ADC_Result(0);
        LedBarGraph(analogVal/25);
        delay(100);
    }
    return 0;
}
```

analogVal は、さまざまな電圧値 (0-5V) で値 (0-255) を生成する。たとえば、バッテリーで 3V が検出されると、対応する値 152 が電圧計に表示される。

LED 棒グラフの 10 個の LED は analogVal 測定値を表示するために使用される。 $255/10 = 25$ 。したがって、25 ごとにアナログ値が増加し、もう 1 つの LED が点灯する。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 3.1.5_BatteryIndicator.py
```

プログラムの実行後、ADC0834 の 3 番目のピンと GND に個別にリード線を配線し、それらを別々にバッテリーの 2 つの極に導く。LED バーグラフの対応する LED が点灯し、電力レベルが表示される（測定範囲：0～5V）。

コードの説明

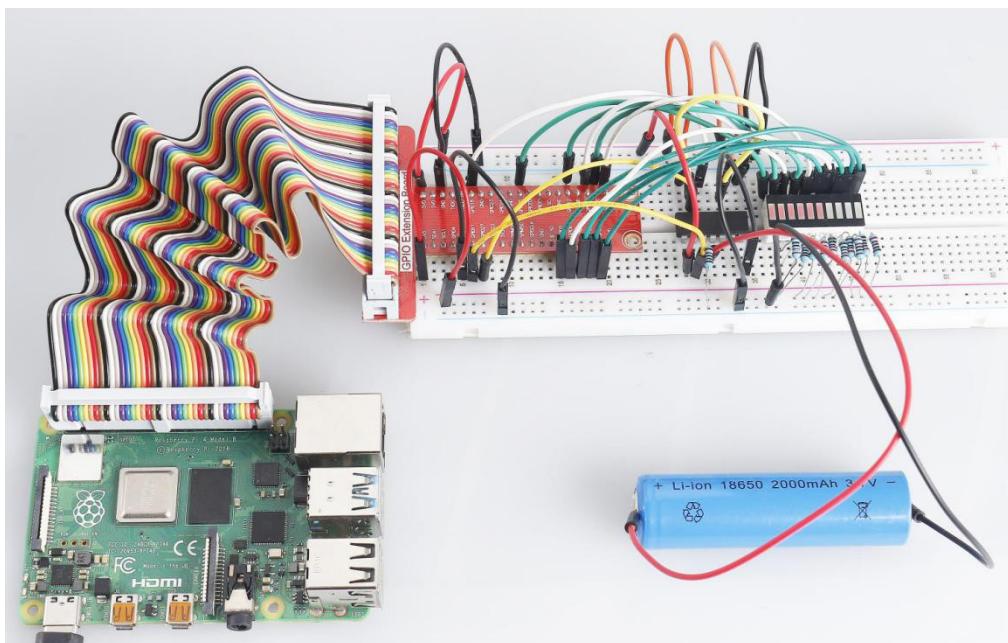
```
def LedBarGraph(value):
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
    for i in range(value):
        GPIO.output(ledPins[i],GPIO.LOW)
```

この機能は、LED 棒グラフの 10 個の LED の点灯/消灯を制御するために機能する。これらの 10 個の LED を最初にオフにするために高レベルを指定し、次に受信したアナログ値を変更することでいくつの LED を点灯させるかを決定する。

```
def loop():
    while True:
        analogVal = ADC0834.getResult()
        LedBarGraph(int(analogVal/25))
```

LED 棒グラフの 10 個の LED は analogVal 測定値を表示するために使用される。 $255/10 = 25$ 。したがって、25 ごとにアナログ値が増加し、もう 1 つの LED が点灯する。

現象画像

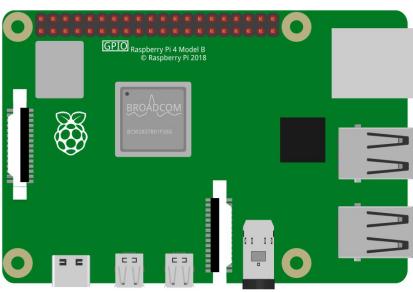
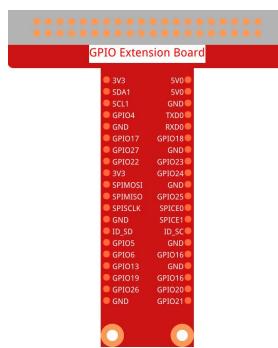
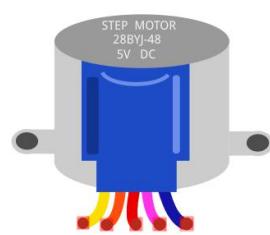
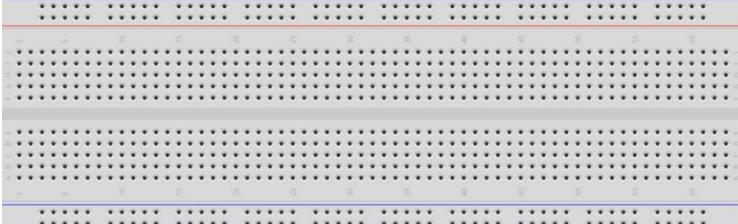
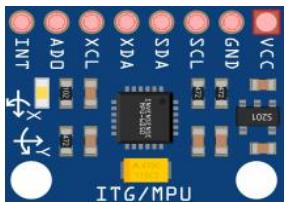
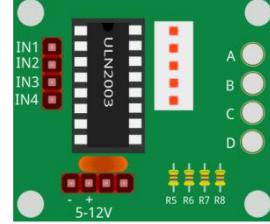


3.1.6 Motion Control

前書き

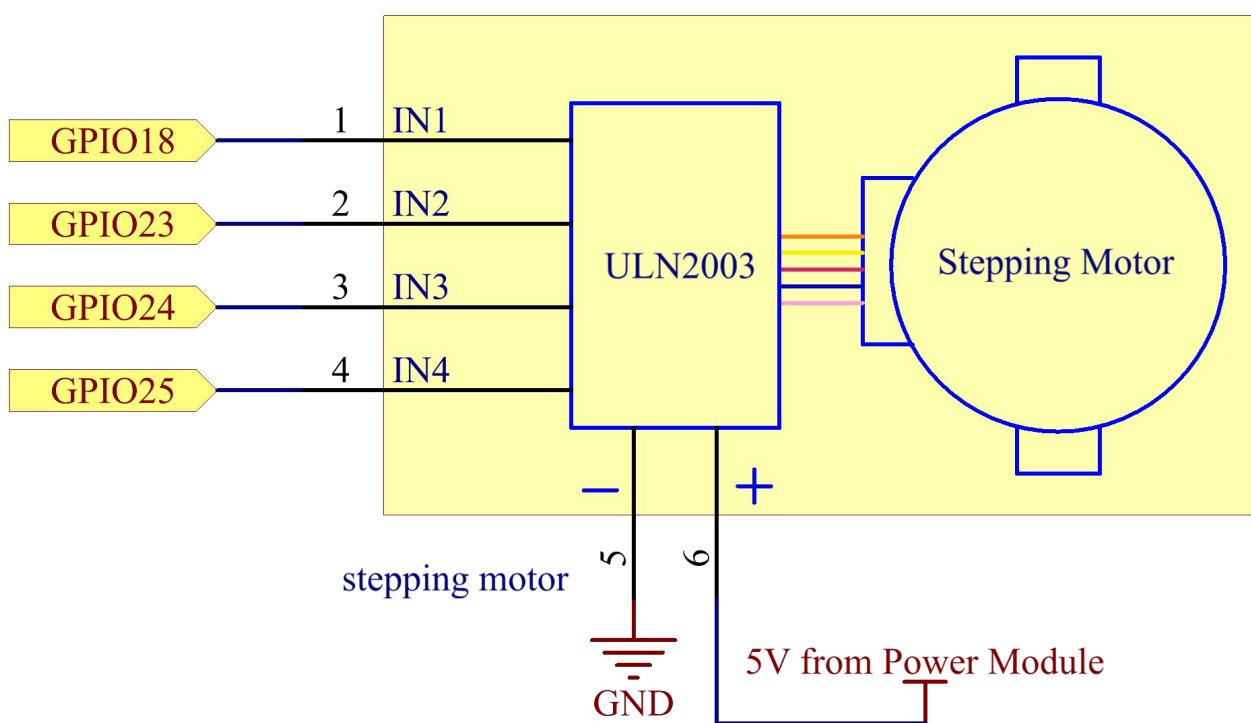
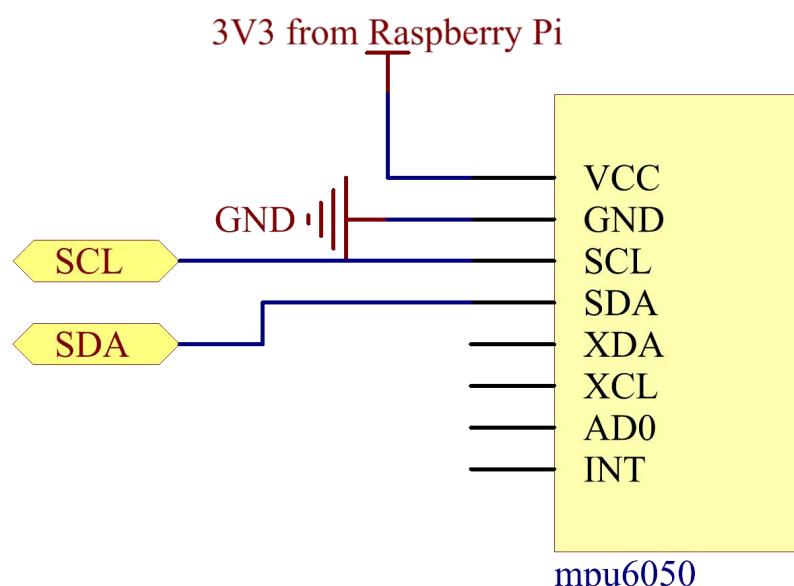
このレッスンでは、簡単なモーション検知と制御装置を作成する。MPU6050 はセンサーとして使用され、ステッピングモーターは制御装置として使用される。MPU6050 をグローブに取り付けた状態で、手首を回すことでステッピングモーターを制御できる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	ステッピングモーター*1
	 GPIO Extension Board Pinout: 3V3, SDA1, SDA2, GND1, GPIO4, TXD0, GND, RXD0, GPIO17, GPIO18, GND2, GPIO27, GPIO28, GND3, GPIO22, GPIO23, GND4, SPI_MOSI, SPI_MISO, SPI_CE0, SPI_CE1, GND5, D2, D3, GND6, GPIO5, GND7, GPIO6, GPIO16, GND8, GPIO13, GND9, GPIO19, GPIO16, GPIO26, GND10, GND11, GND12, GND13, GND14, GND15, GND16, GND17, GND18, GND19, GND20, GND21, GND22, GND23, GND24, GND25, GND26, GND27, GND28, GND29, GND30, GND31, GND32, GND33, GND34, GND35, GND36, GND37, GND38, GND39, GND40	
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		MPU6050 モジュール*1
		
		ULN2003 モジュール*1
		

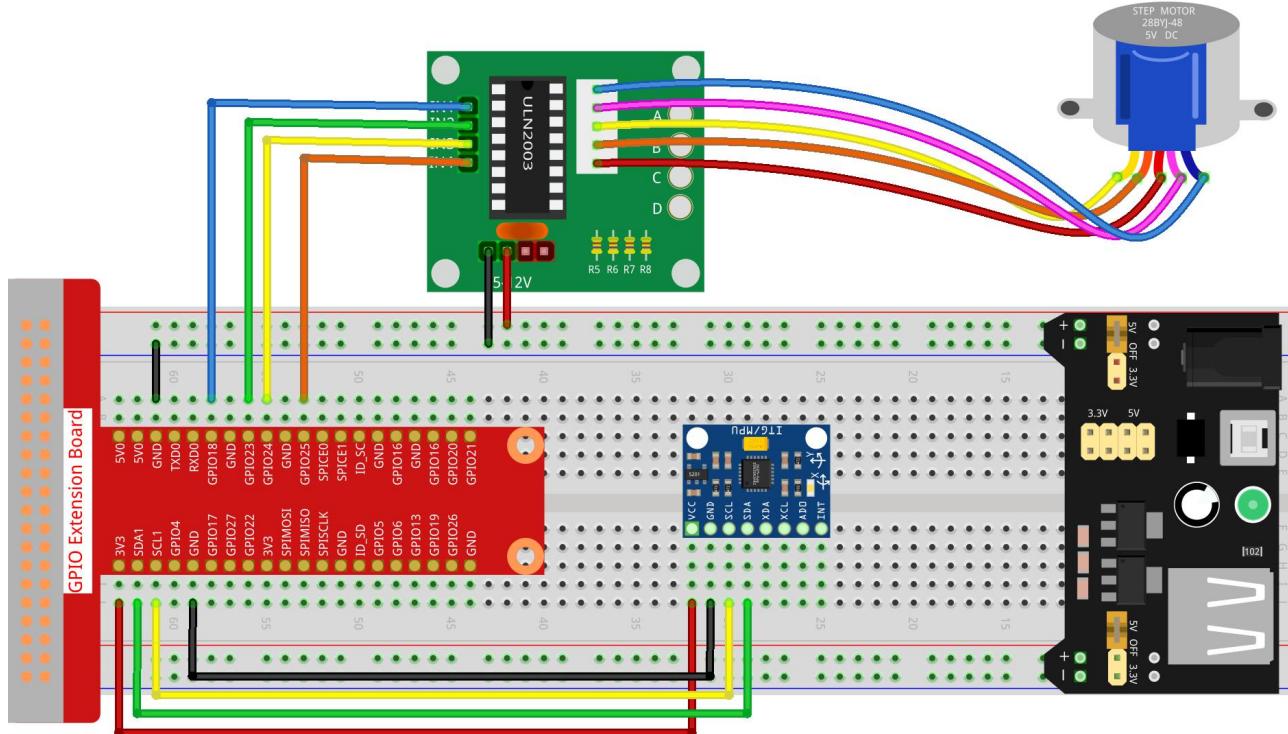
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3		
SCL1	Pin 5		



実験手順

ステップ 1: Build the circuit.



➤ C 言語ユーザー向け

ステップ 2: 回路を作る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.6/
```

ステップ 3: コードのフォルダーに入る。

```
gcc 3.1.6_MotionControl.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードの実行中に、Y 軸上の mpu6050 の傾斜角が 45°C より大きい場合、ステッピングモーターは反時計回りに回転する。-45°C 未満の場合、ステッピングモーターは時計回りに回転する。

コードの説明

```
double mpu6050(){
    accX = read_word_2c(0x3B);
    accY = read_word_2c(0x3D);
    accZ = read_word_2c(0x3F);
    accX_scaled = accX / 16384.0;
    accY_scaled = accY / 16384.0;
```

```

acclZ_scaled = acclZ / 16384.0;
double angle=get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled);
return angle;
}

```

mpu6050 は Y 軸の方向の傾斜角を取得する。

```

void rotary(char direction){
    if(direction == 'c'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++){
                {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}
                delayMicroseconds(stepSpeed);
            }
        }
    } else if(direction == 'a'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++){
                {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}
                delayMicroseconds(stepSpeed);
            }
        }
    }
}

```

受信した方向キーが「c」の場合、ステッピングモーターは時計回りに回転する。キーが「a」の場合、モーターは反時計回りに回転する。ステッピングモーターの回転方向の計算の詳細については、**1.3.3 Stepper Motor** を参照してください。

```

int main()
{
    setup();
    double angle;
    while(1) {
        angle = mpu6050();
        if (angle >=45){rotary('a');}
        else if (angle<=-45){rotary('c');}
    }
    return 0;
}

```

Y 軸方向の傾斜角は mpu6050 から読み取られ、45°Cより大きい場合、ステッピングモーターは反時計回りに回転する。 -45°C未満の場合、ステッピングモーターは時計回りに回転する。

➤ Python 言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ3: EXE ファイルを実行する。

```
sudo python3 3.1.6_MotionControl.py
```

コードの実行中に、Y 軸上の mpu6050 の傾斜角が 45°C より大きい場合、ステッピングモーターは反時計回りに回転する。-45°C 未満の場合、ステッピングモーターは時計回りに回転する。

コードの説明

```
def mpu6050():
    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
    accel_zout = read_word_2c(0x3f)
    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0
    angle=get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_zout_scaled)
    return angle
```

MPU6050 は Y 軸の方向の傾斜角を取得する。

```
def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99>>j & (0x08>>i))
                time.sleep(stepSpeed)

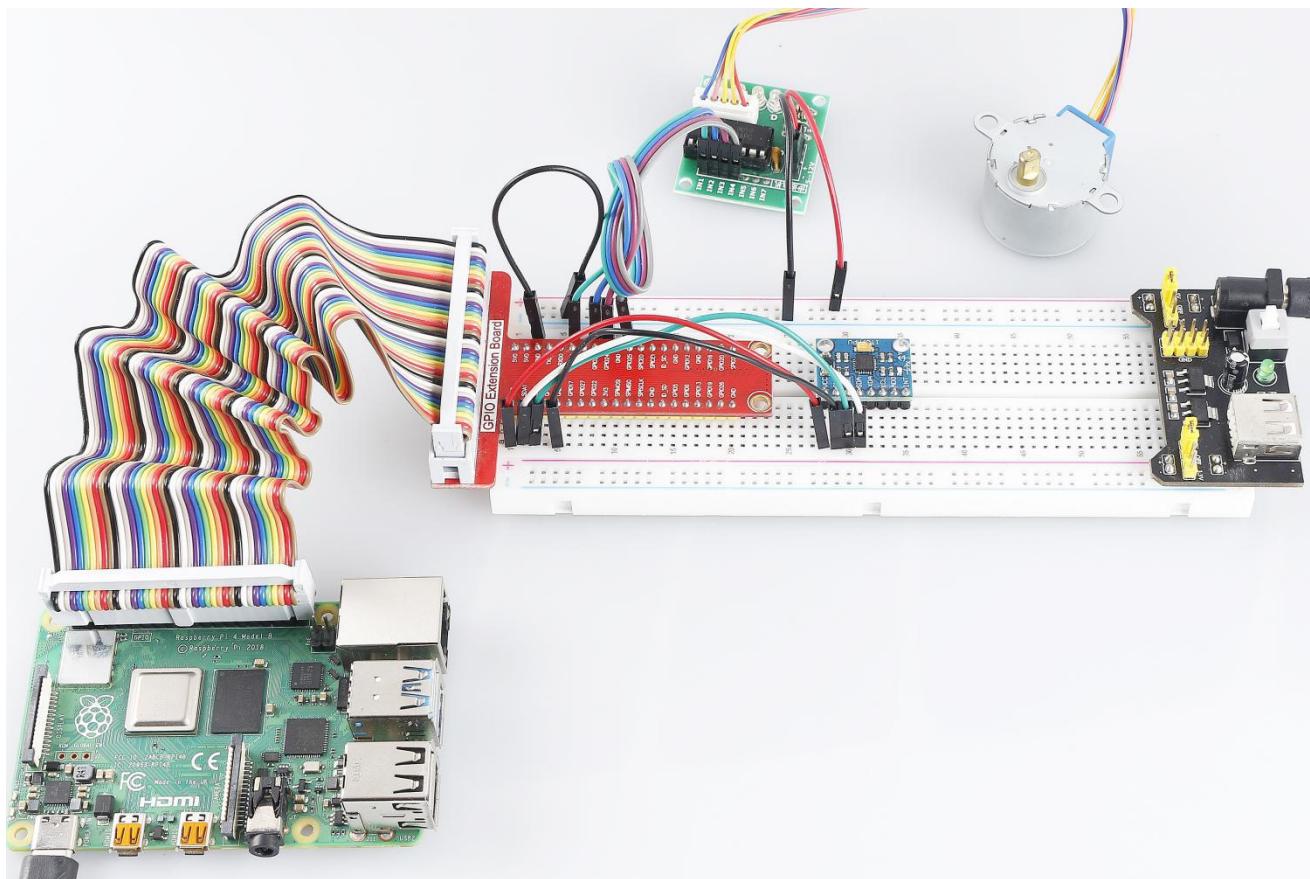
    elif(direction == 'a'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
                time.sleep(stepSpeed)
```

受信した方向キーが 'c' の場合、ステッピングモーターは時計回りに回転する。キーが「a」の場合、モーターは反時計回りに回転する。ステッピングモーターの回転方向の計算の詳細については、**1.3.3 Stepper Motor** を参照してください。

```
def loop():
    while True:
        angle=mpu6050()
        if angle >=45 :
            rotary('a')
        elif angle <=-45:
            rotary('c')
```

Y 軸方向の傾斜角は mpu6050 から読み取られ、45°Cより大きい場合、ステッピングモーターを反時計回りに回転させるために rotary()が呼び出される。 -45°C未満の場合、ステッピングモーターは時計回りに回転する。

現象画像

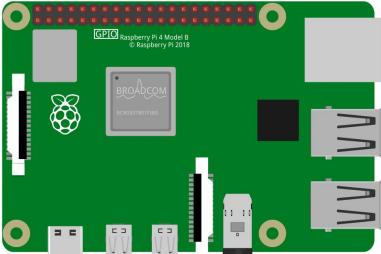
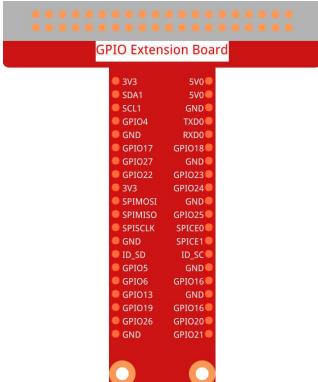
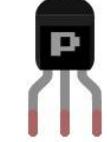
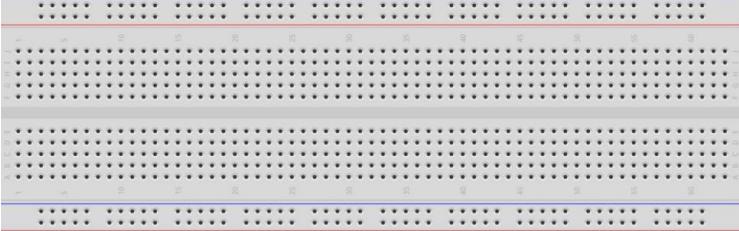


3.1.7 Traffic Light

前書き

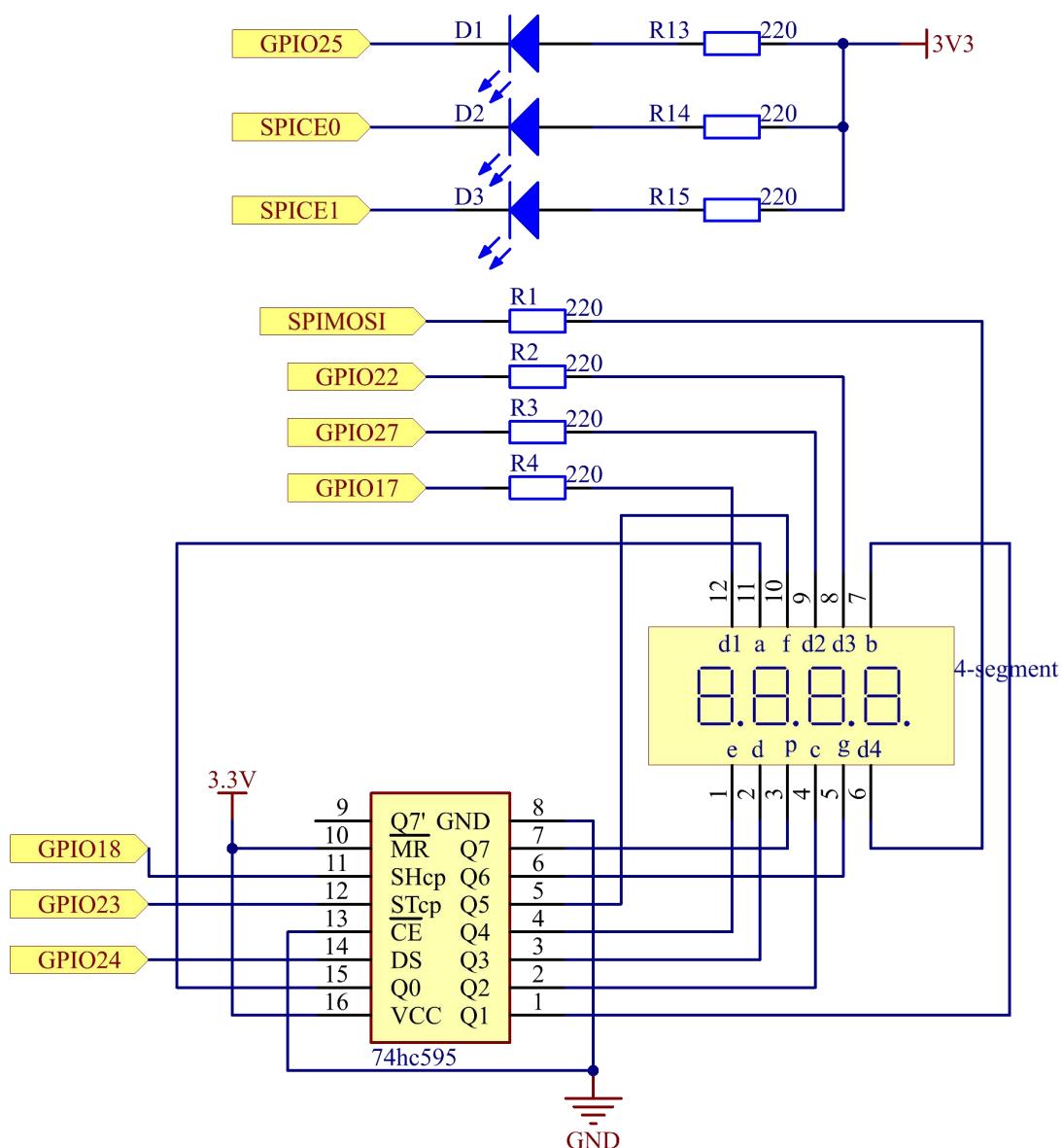
このプロジェクトでは、3色のLEDを使用して交通信号の変化を実現し、4桁の7セグメントディスプレイを使用して各交通状態のタイミングを表示する。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	4桁 7セグメントディスプレイ*1
	 GPIO Extension Board Pinout: 3V3 SDA1 5V0 GND SCL1 GND GPIO4 TXD0 GND RXD0 GPIO17 GPIO18 GPIO27 GND GPIO22 GPIO23 3V3 GPIO24 GND SPIMOSI GP9/25 GP9/26 SPIMISO GP10/25 GP10/26 GND ERCE1 ID_SD ID_SC GPIO5 GND GPIO6 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO26 GPIO20 GND GPIO21	
40ピンケーブル*1	S8550 PNPトランジスタ*1	
		
ブレッドボード*1	74HC595*1	
		
何本のジャンパー線		
抵抗器 (220Ω) *11		
抵抗器 (1kΩ) *4		

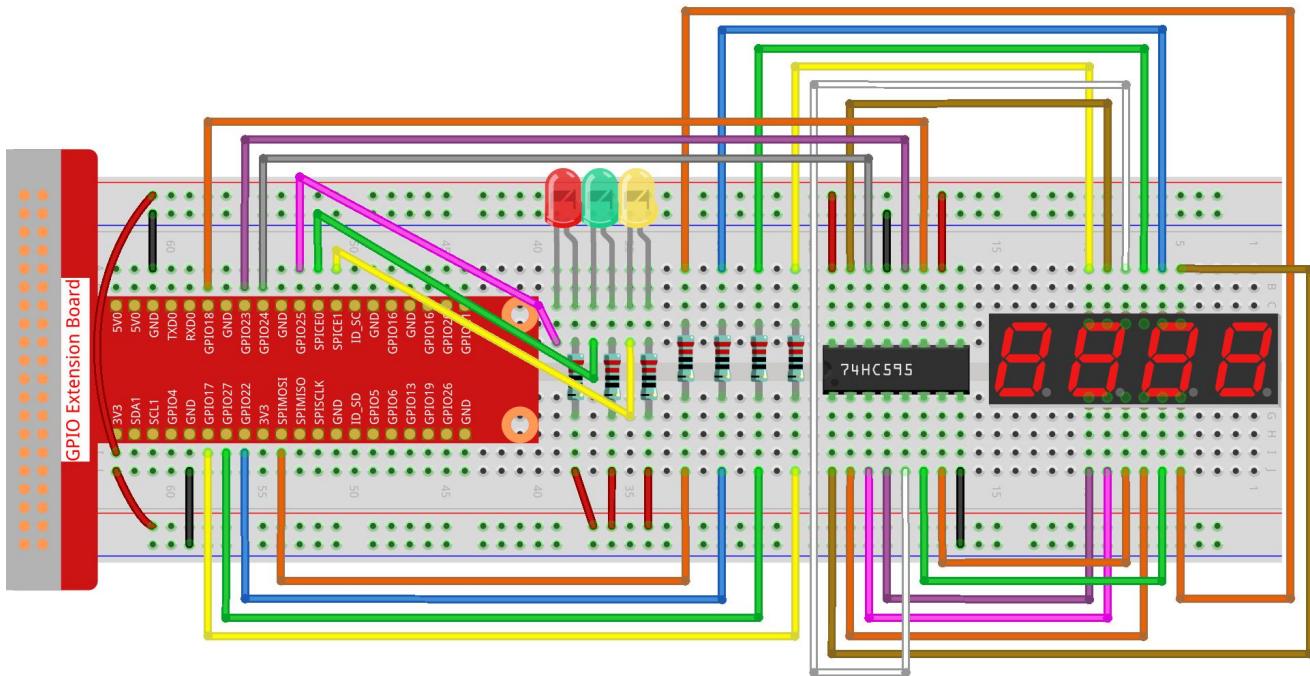
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	Pin 26	11	7



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.7/
```

ステップ3: コンパイルする。

```
gcc 3.1.7_TrafficLight.c -lwiringPi
```

ステップ4: 実行する。

```
sudo ./a.out
```

コードが実行されると、LEDは交通信号の色の変化をシミュレートする。まず、赤色のLEDが60秒間点灯し、それから緑色のLEDが30秒間点灯し、最後に、黄色のLEDが5秒間点灯する。その後、赤いLEDが60秒間再び点灯する。このようにして、この一連のアクションは繰り返し実行される。

コードの説明

```
#define SDI 5
#define RCLK 4
#define SRCLK 1

const int placePin[] = {12, 3, 2, 0};
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

void pickDigit(int digit);
void hc595_shift(int8_t data);
void clearDisplay();
void display();
```

これらのコードは 4 枝 7 セグメントディスプレイの数字表示機能を実現するために使用される。詳細については、本文の章 1.1.5 を参照してください。ここでは、コードを使用して交通信号時間のカウントダウンを表示する。

```
const int ledPin[]={6,10,11};

int colorState = 0;

void lightup()
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],HIGH);
    }
    digitalWrite(ledPin[colorState],LOW);
}
```

コードは LED のオンとオフを切り替えるために使用される。

```
int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 60;

void timer(int timer1){ //Timer function
```

```
if(timer1 == SIGALRM){  
    counter --;  
    alarm(1);  
    if(counter == 0){  
        if(colorState == 0) counter = greenLight;  
        if(colorState == 1) counter = yellowLight;  
        if(colorState == 2) counter = redLight;  
        colorState = (colorState+1)%3;  
    }  
    printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);  
}  
}
```

コードはタイマーのオンとオフを切り替えるために使用される。詳細については、1.1.5 章を参照してください。ここで、タイマーがゼロに戻ると、colorState は LED を切り替えるように切り替えられ、タイマーは新しい値に割り当てられる。

```
void loop()  
{  
    while(1){  
        display();  
        lightup();  
    }  
}  
  
int main(void)  
{  
    //...  
    signal(SIGALRM,timer);  
    alarm(1);  
    loop();  
    return 0;  
}
```

タイマーは main() 関数で始まる。Loop() 関数では、while(1) loop を使用して、4 行 7 セグメントと LED の関数を呼び出す。

➤ Python 言語ユーザー向け

ステップ 2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 3.1.7_TrafficLight.py
```

コードが実行されると、LED は交通信号の色の変化をシミュレートする。まず、赤色の LED が 60 秒間点灯し、それから緑色の LED が 30 秒間点灯し、最後に、黄色の LED が 5 秒間点灯する。その後、赤い LED が 60 秒間再び点灯する。このようにして、この一連のアクションは繰り返し実行される。一方、4 行の 7 セグメントディスプレイには、カウントダウン時間が連続して表示される。

コードの説明

```
SDI = 24      #serial data input(DS)
RCLK = 23     #memory clock input(STCP)
SRCLK = 18    #shift register clock input(SHCP)
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
placePin = (17,27,22,10)

def clearDisplay():
def hc595_shift(data):
def pickDigit(digit):
def display():
```

これらのコードは 4 行 7 セグメントの数字表示機能を実現するために使用される。詳細については、ドキュメントの 1.1.5 章を参照してください。ここでは、コードを使用して交通信号の時間のカウントダウンを表示する。

```
ledPin =(22,24,26)
colorState=0

def lightup():
    global colorState
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.HIGH)
        GPIO.output(ledPin[colorState], GPIO.LOW)
```

コードは LED のオンとオフを切り替えるために使用される。

```
greenLight = 30
```

```

yellowLight = 5
redLight = 60
lightColor=("Red","Green","Yellow")

colorState=0
counter = 60
timer1 = 0

def timer():      #timer function
    global counter
    global colorState
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
        colorState=(colorState+1)%3
    print ("counter : %d      color: %s "%(counter,lightColor[colorState]))

```

コードはタイマーのオンとオフを切り替えるために使用される。詳細については、1.1.5 章を参照してください。ここで、タイマーがゼロに戻ると、colorState は LED を切り替えるように切り替えられ、タイマーは新しい値に割り当てられる。

```

def setup():
    # ...
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()

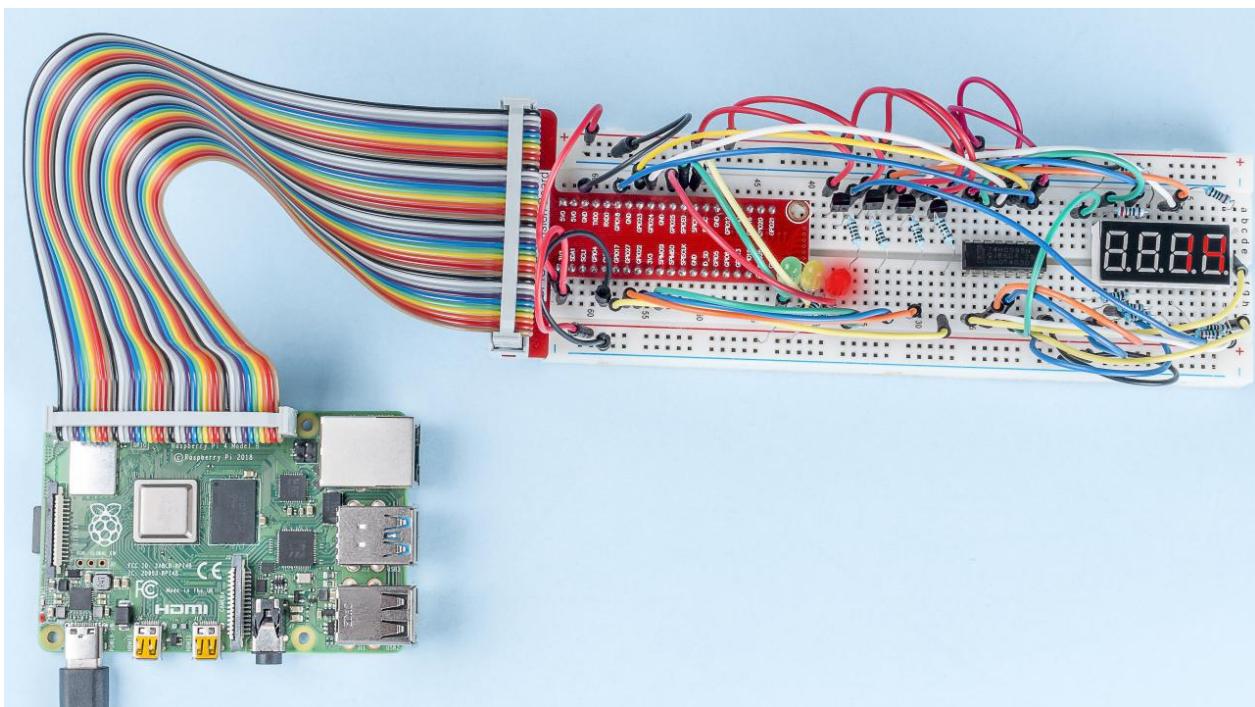
def loop():
    while True:
        display()
        lightup()

```

```
def destroy():    # When "Ctrl+C" is pressed, the function is executed.  
    global timer1  
    GPIO.cleanup()  
    timer1.cancel()      #cancel the timer  
  
if __name__ == '__main__': # Program starting from here  
    setup()  
    try:  
        loop()  
    except KeyboardInterrupt:  
        destroy()
```

setup() 関数で、タイマーを開始する。Loop() 関数では、while True が使用される：4 行の 7 セグメントと LED の相対関数を循環的に呼び出す。

現象画像

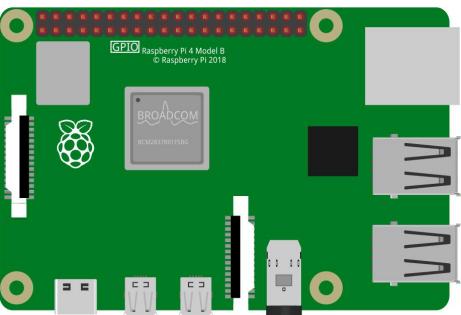
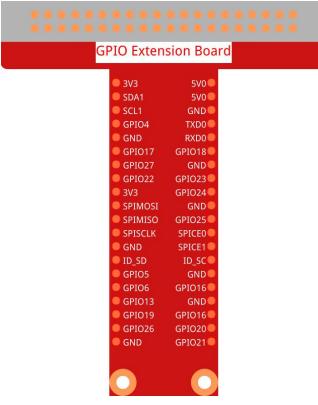
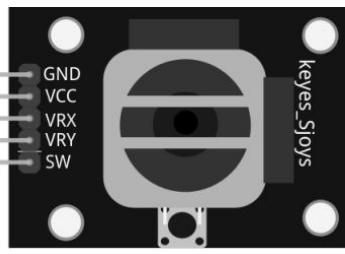
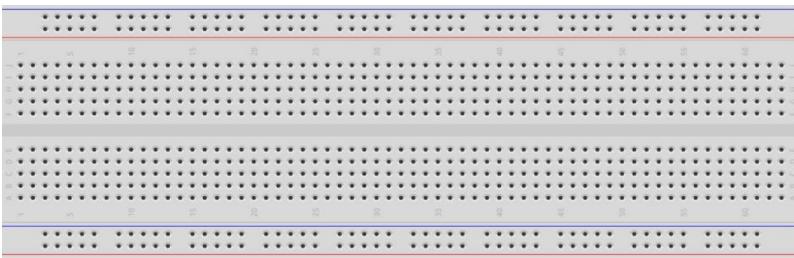
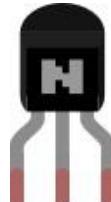


3.1.8 Overheat Monitor

前書き

回路の過熱が発生したときに警報装置とタイムリーな自動電源切断を希望する場合は、工場などのさまざまな状況に適用される過熱監視装置を作成することができる。このレッスンでは、サーミスタ、ジョイスティック、ブザー、LED、と LCD を使用して、しきい値が調整可能なスマートな温度監視装置を作成する。

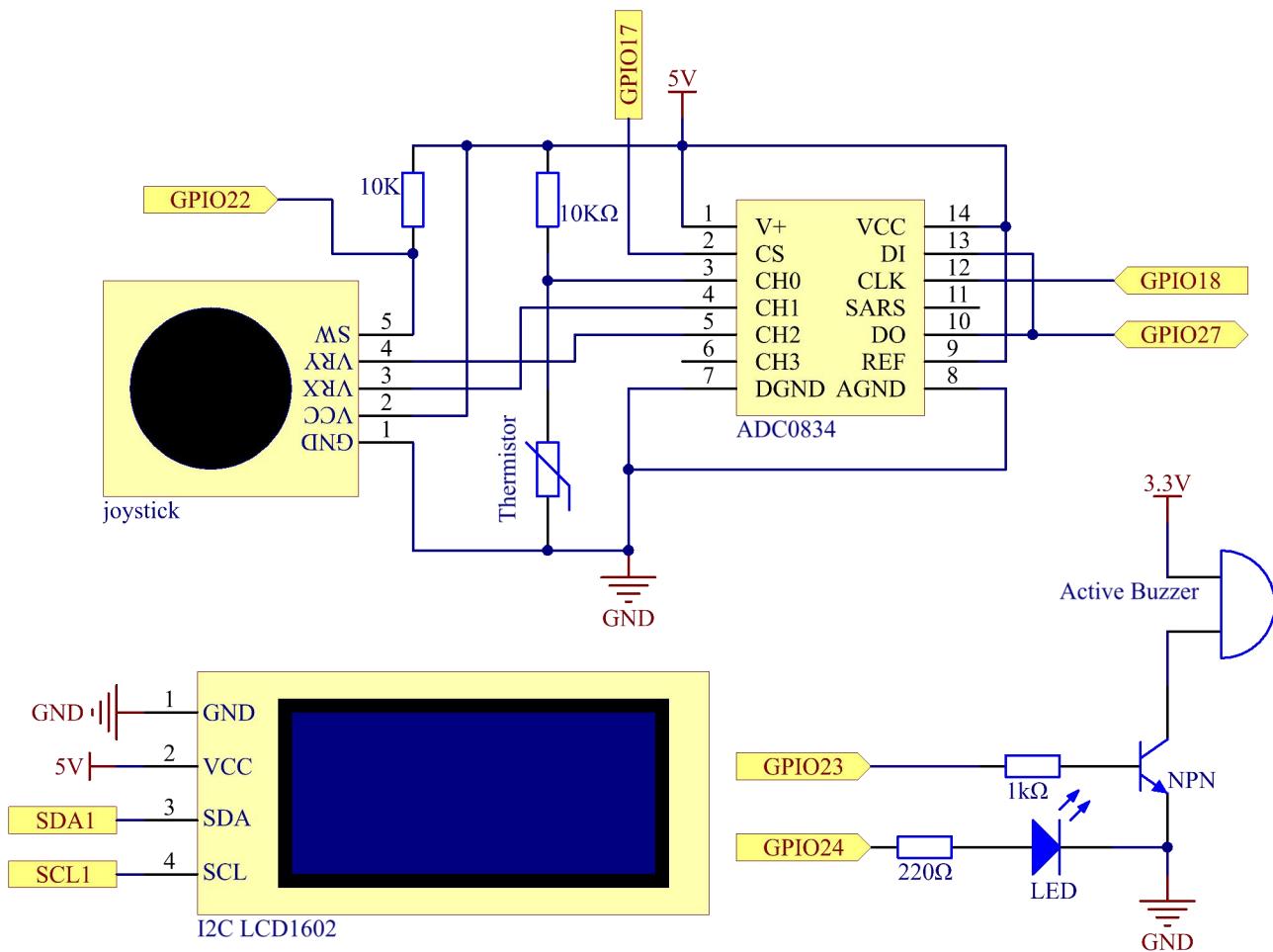
部品

Raspberry Pi 本体*1	T 拡張ボード*1	ジョイスティック*1
		
40 ピンケーブル*1	ADC0834*1	
		
ブレッドボード*1	LED*1	S8050 NPN ランジスタ*1
		
抵抗器 (220Ω) *1	抵抗器 (1kΩ) *1	抵抗器 (10KΩ) *2
		



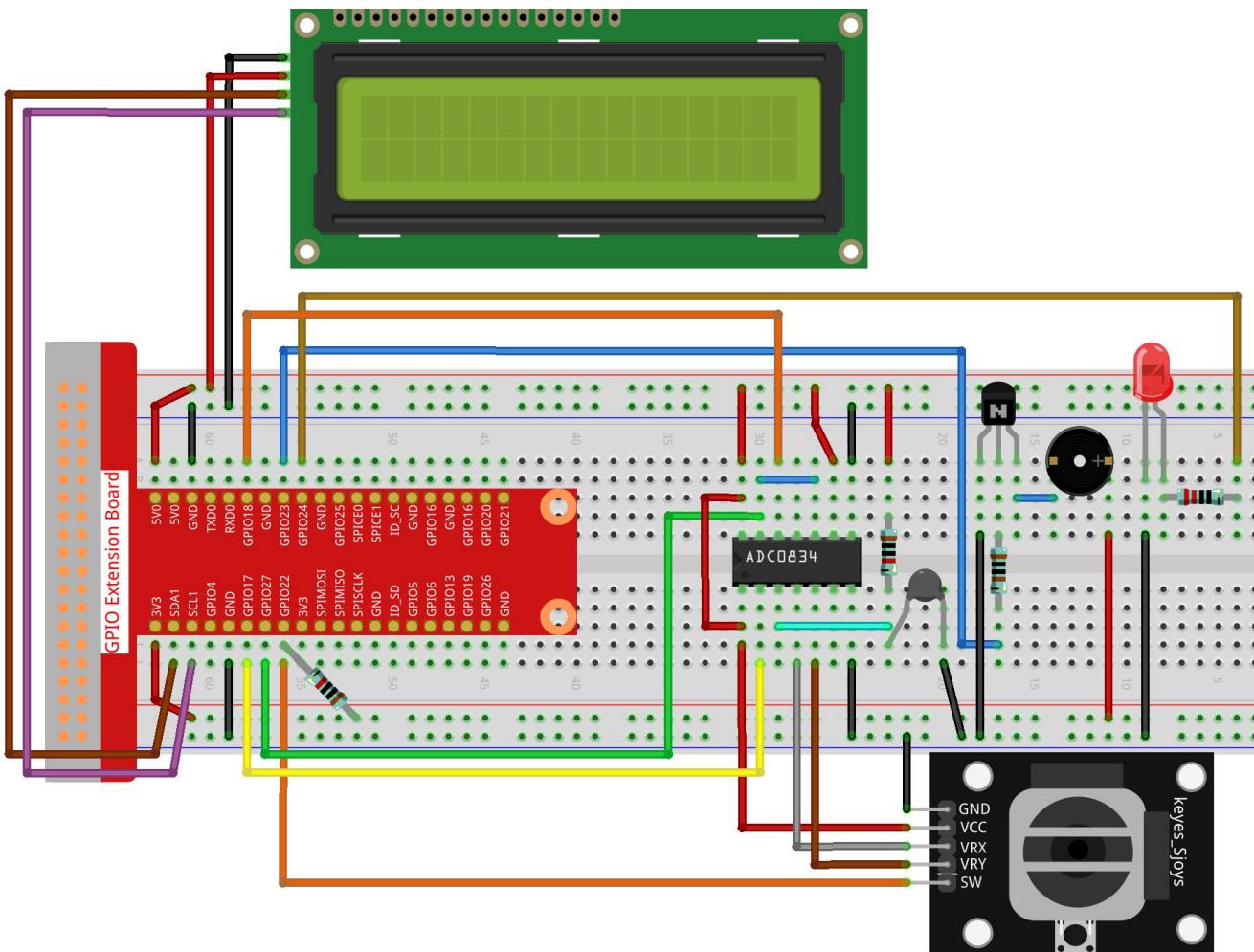
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22
GPIO23	Pin16	4	23
GPIO24	Pin18	5	24
SDA1	Pin 3		
SCL1	Pin 5		



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.8/
```

ステップ 3: コードをコンパイルする。

```
gcc 3.1.8_OverheatMonitor.c -lwiringPi
```

ステップ 4: EXE ファイルを実行する。

```
sudo ./a.out
```

コードが実行されると、現在の温度と高温のしきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を発する。

ここでのジョイスティックは高温のしきい値を調整するために使用される。ジョイスティックを X 軸と Y 軸の方向に切り替えると、現在の高温しきい値を調整できる(上/下)。ジョイスティックをもう一度押して、しきい値を初期値にリセットする。

コードの説明

```
int get_joystick_value(){
    uchar x_val;
    uchar y_val;
    x_val = get_ADC_Result(1);
    y_val = get_ADC_Result(2);
    if (x_val > 200){
        return 1;
    }
    else if(x_val < 50){
        return -1;
    }
    else if(y_val > 200){
        return -10;
    }
    else if(y_val < 50){
        return 10;
    }
    else{
        return 0;
    }
}
```

この関数は、XとYの値を読み取る。X> 200の場合、「1」が返される。 X <50 の場合、「-1」が返される。 y> 200 の場合、「-10」を返し、y <50 の場合、「10」を返す。

```
void upper_tem_setting(){
    write(0, 0, "Upper Adjust:");
    int change = get_joystick_value();
    upperTem = upperTem + change;
    char str[6];
    sprintf(str, "%d", upperTem);
    write(0, 1, str);
    int len;
    len = strlen(str);
    write(len, 1, " ");
    delay(100);
}
```

この機能は、しきい値を調整し、I2C LCD1602 に表示するために使用される。

```
double temperature(){
    unsigned char temp_value;
    double Vr, Rt, temp, cel, Fah;
    temp_value = get_ADC_Result(0);
    Vr = 5 * (double)(temp_value) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    return cel;
}
```

ADC0834 の CH0 (サーミスタ) のアナログ値を読み取り、温度値に変換する。

```
void monitoring_temp(){
    char str[6];
    double cel = temperature();
    sprintf(str,6,"% .2f",cel);
    write(0, 0, "Temp: ");
    write(6, 0, str);
    sprintf(str,3,"%d",upperTem);
    write(0, 1, "Upper: ");
    write(7, 1, str);
    delay(100);
    if(cel >= upperTem){
        digitalWrite(buzzPin, HIGH);
        digitalWrite(LedPin, HIGH);
    }
    else if(cel < upperTem){
        digitalWrite(buzzPin, LOW);
        digitalWrite(LedPin, LOW);
    }
}
```

コードが実行されると、現在の温度と高温のしきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を発する。

```
int main(void)
{
    setup();
    int lastState =1;
    int stage=0;
```

```

while (1)
{
    int currentState = digitalRead(Joy_BtnPin);
    if(currentState==1 && lastState == 0){
        stage=(stage+1)%2;
        delay(100);
        lcd_clear();
    }
    lastState=currentState;
    if (stage==1){
        upper_tem_setting();
    }
    else{
        monitoring_temp();
    }
}
return 0;
}

```

関数 main () には、次のようにプログラムプロセス全体が含まれる：

- 1) プログラムが開始すると、ステージの初期値は 0 になり、現在の温度と高温しきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を出す。
- 2) ジョイスティックを押すと、ステージが 1 になり、高温しきい値を調整できる。ジョイスティックを X 軸と Y 軸の方向に切り替えると、現在のしきい値を調整（上下）できる。ジョイスティックをもう一度押して、しきい値を初期値にリセットする。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 3.1.8_OverheatMonitor.py
```

コードが実行されると、現在の温度と高温のしきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を発する。

ここでのジョイスティックは高温のしきい値を調整するために使用される。ジョイスティックを X 軸と Y 軸の方向に切り替えると、現在の高温しきい値を調整（上下）できる。ジョイスティックをもう一度押して、しきい値を初期値にリセットする。

コードの説明

```
def get_joystick_value():
    x_val = ADC0834.getResult(1)
    y_val = ADC0834.getResult(2)
    if(x_val > 200):
        return 1
    elif(x_val < 50):
        return -1
    elif(y_val > 200):
        return -10
    elif(y_val < 50):
        return 10
    else:
        return 0
```

この関数は、XとYの値を読み取る。X> 200の場合、「1」が返される。X<50の場合、「-1」が返される。y> 200の場合、「-10」を返し、y<50の場合、「10」を返す。

```
def upper_tem_setting():
    global upperTem
    LCD1602.write(0, 0, 'Upper Adjust: ')
    change = int(get_joystick_value())
    upperTem = upperTem + change
    LCD1602.write(0, 1, str(upperTem))
    LCD1602.write(len(strUpperTem), 1, '')
    time.sleep(0.1)
```

この機能は、しきい値を調整し、I2C LCD1602に表示するために使用される。

```
def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    temp = 1/((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return round(Cel,2)
```

ADC0834のCH0（サーミスタ）のアナログ値を読み取り、温度値に変換する。

```
def monitoring_temp():
    global upperTem
    Cel=temperature()
```

```

LCD1602.write(0, 0, 'Temp: ')
LCD1602.write(0, 1, 'Upper: ')
LCD1602.write(6, 0, str(Cel))
LCD1602.write(7, 1, str(upperTem))
time.sleep(0.1)
if Cel >= upperTem:
    GPIO.output(buzzPin, GPIO.HIGH)
    GPIO.output(ledPin, GPIO.HIGH)
else:
    GPIO.output(buzzPin, GPIO.LOW)
    GPIO.output(ledPin, GPIO.LOW)

```

コードが実行されると、現在の温度と高温のしきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を出す。

```

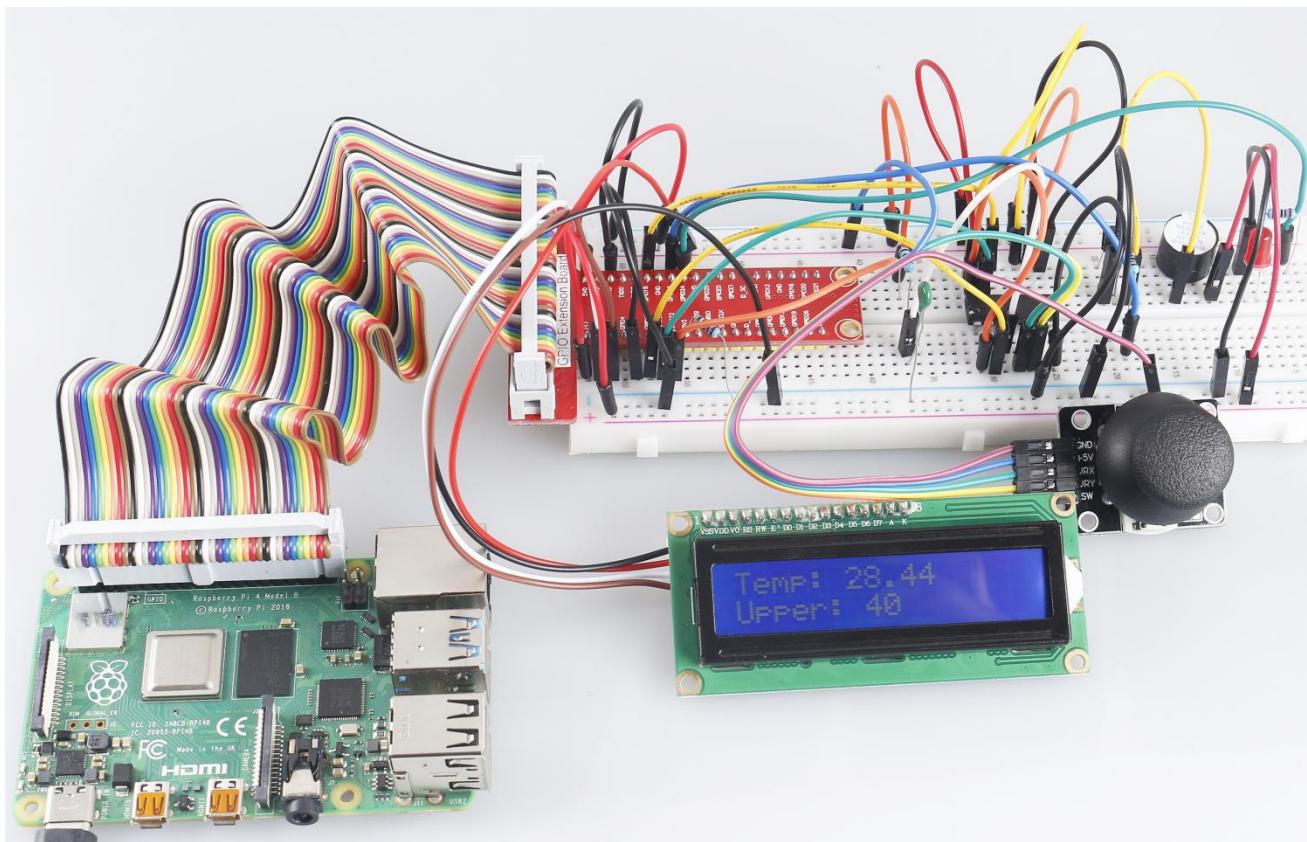
def loop():
    lastState=1
    stage=0
    while True:
        currentState=GPIO.input(Joy_BtnPin)
        if currentState==1 and lastState ==0:
            stage=(stage+1)%2
            time.sleep(0.1)
            LCD1602.clear()
        lastState=currentState
        if stage == 1:
            upper_tem_setting()
        else:
            monitoring_temp()

```

関数 main () には、次のようにプログラムプロセス全体が含まれている：

- 1) プログラムが開始すると、ステージの初期値は 0 になり、現在の温度と高温しきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を出す。
- 2) ジョイスティックを押すと、ステージが 1 になり、高温しきい値を調整できる。ジョイスティックを X 軸と Y 軸の方向に切り替えると、現在の高温しきい値を調整（上下）できる。ジョイスティックをもう一度押して、しきい値を初期値にリセットする。

現象画像



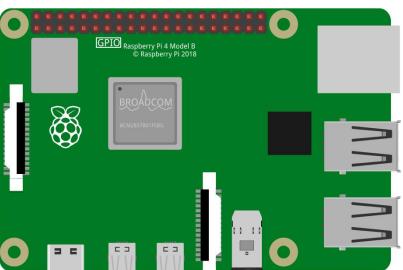
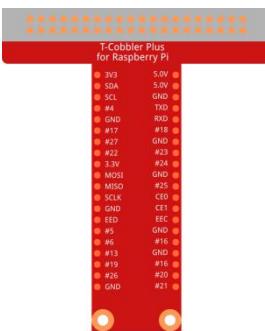
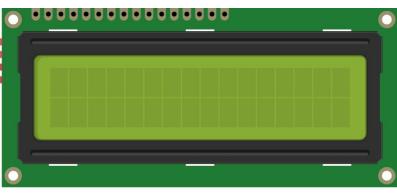
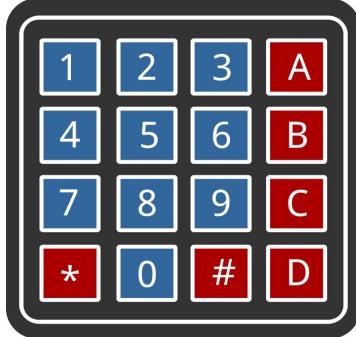
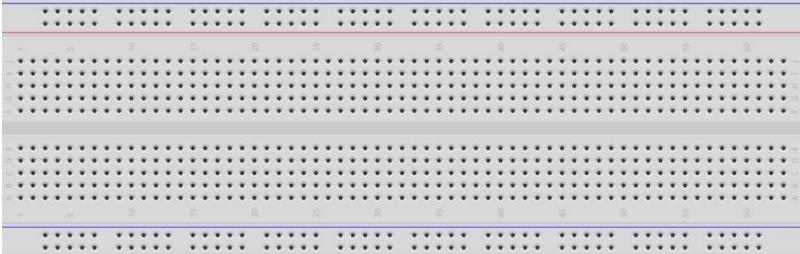
3.1.9 Password Lock

前書き

このプロジェクトでは、キーパッドと LCD を使用してコンビネーションロックを作成する。LCD はキーパッドでパスワードを入力するための対応するプロンプトを表示する。パスワードが正しく入力されると、「Correct」と表示される。

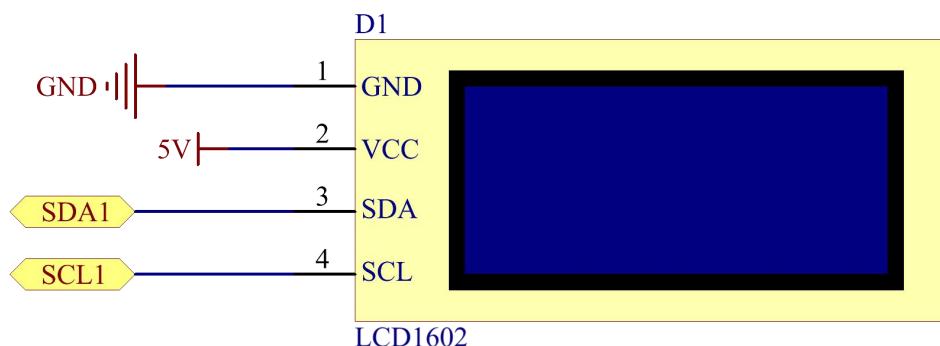
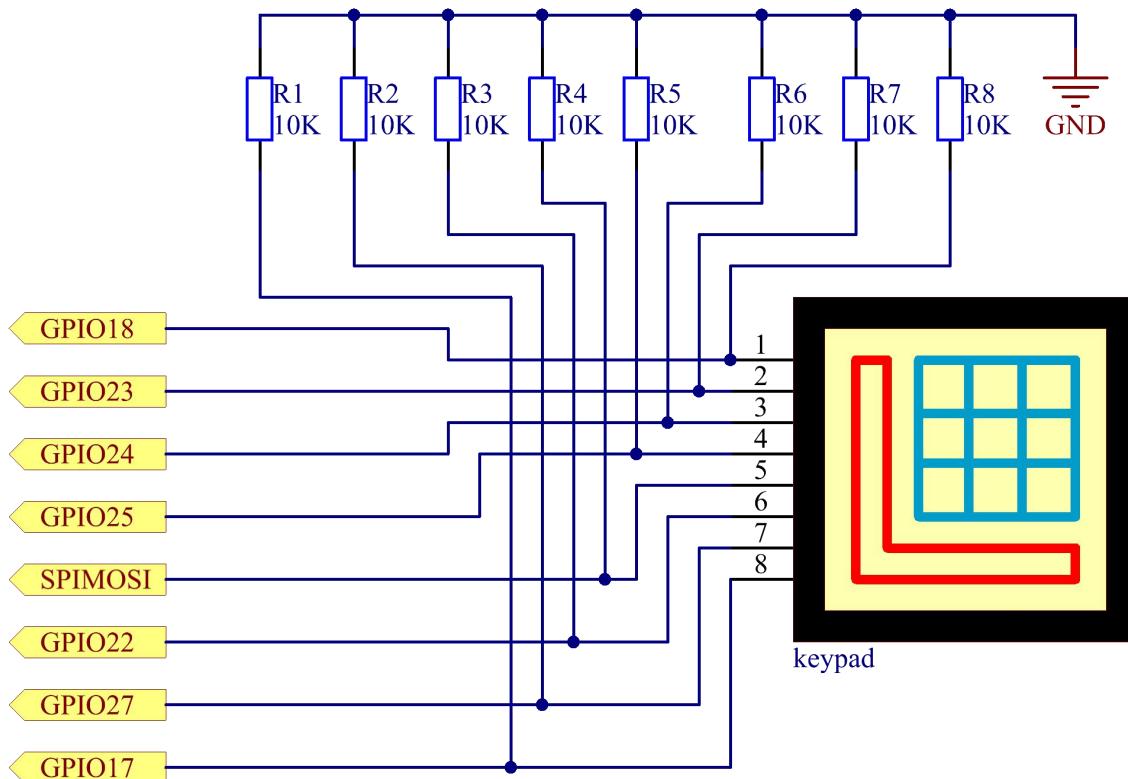
このプロジェクトに基づいて、ブザー、LED などの電子部品を追加して、パスワード入力にさまざまな実験現象を追加できる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	I2C LCD1602*1																																																						
	 T-Cobbler Plus for Raspberry Pi <table border="1"><tr><td>3V3</td><td>5V</td><td>GND</td></tr><tr><td>SDA</td><td>GND</td><td>SCL</td></tr><tr><td>SCL</td><td>I2C</td><td>TXD</td></tr><tr><td>GND</td><td>I2C</td><td>TXD</td></tr><tr><td>#17</td><td>#18</td><td>#19</td></tr><tr><td>#27</td><td>GND</td><td>#20</td></tr><tr><td>#22</td><td>#21</td><td>#23</td></tr><tr><td>#5</td><td>GND</td><td>#24</td></tr><tr><td>MOSI</td><td>GND</td><td>MISO</td></tr><tr><td>MISO</td><td>#25</td><td>SCLK</td></tr><tr><td>SCLK</td><td>CEO</td><td>GND</td></tr><tr><td>GND</td><td>CE1</td><td>ELO</td></tr><tr><td>ELO</td><td>ELO</td><td>GND</td></tr><tr><td>#6</td><td>#16</td><td>#15</td></tr><tr><td>#13</td><td>GND</td><td>#14</td></tr><tr><td>#19</td><td>#15</td><td>#26</td></tr><tr><td>#26</td><td>#20</td><td>#21</td></tr><tr><td>GND</td><td>#21</td><td>GND</td></tr></table>	3V3	5V	GND	SDA	GND	SCL	SCL	I2C	TXD	GND	I2C	TXD	#17	#18	#19	#27	GND	#20	#22	#21	#23	#5	GND	#24	MOSI	GND	MISO	MISO	#25	SCLK	SCLK	CEO	GND	GND	CE1	ELO	ELO	ELO	GND	#6	#16	#15	#13	GND	#14	#19	#15	#26	#26	#20	#21	GND	#21	GND	
3V3	5V	GND																																																						
SDA	GND	SCL																																																						
SCL	I2C	TXD																																																						
GND	I2C	TXD																																																						
#17	#18	#19																																																						
#27	GND	#20																																																						
#22	#21	#23																																																						
#5	GND	#24																																																						
MOSI	GND	MISO																																																						
MISO	#25	SCLK																																																						
SCLK	CEO	GND																																																						
GND	CE1	ELO																																																						
ELO	ELO	GND																																																						
#6	#16	#15																																																						
#13	GND	#14																																																						
#19	#15	#26																																																						
#26	#20	#21																																																						
GND	#21	GND																																																						
40 ピンケーブル*1		何本のジャンパー線																																																						
		キーパッド*1																																																						
ブレッドボード*1		 																																																						
		抵抗器 (10KΩ) *1 																																																						

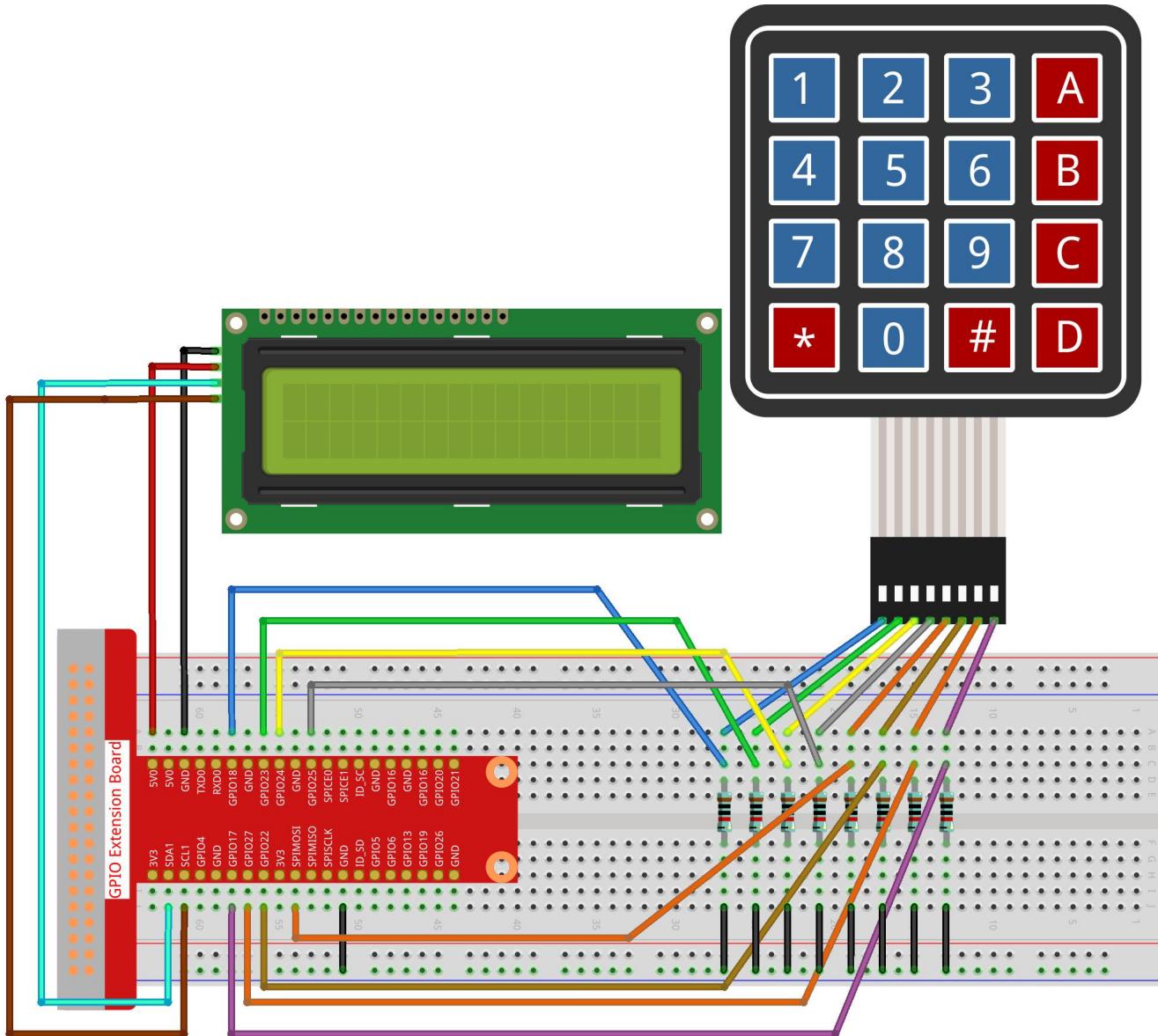
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPI MOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.9/
```

ステップ3: コンパイルする。

```
gcc 3.1.9_PasswordLock.cpp -lwiringPi
```

ステップ4: 実行する。

```
sudo ./a.out
```

コードの実行後、キーパッドを使用してパスワードを入力する。LCD1602 に「CORRECT」と表示されている場合、パスワードに問題はない。そうでない場合、「WRONG KEY」が表示される。

コードの説明

```
#define ROWS 4
#define COLS 4
#define BUTTON_NUM (ROWS * COLS)
#define LENS 4

unsigned char KEYS[BUTTON_NUM] {
    '1','2','3','A',
    '4','5','6','B',
    '7','8','9','C',
    '*', '0', '#', 'D'};

char password[LENS]={'1','9','8','4'};
```

ここでは、パスワード LENS、ストレージマトリックスキーボードキー値配列 KEYS と正しいパスワードを保存する配列の長さを定義する。

```
void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);
```

マトリックスキーボードコードのサブ関数の宣言があり、詳しくは、章 2.1.5 を参照してください。

```
void write_word(int data);
void send_command(int comm);
void send_data(int data);
void lcdInit();
void clear();
void write(int x, int y, char const data[]);
```

LCD1062 コードのサブ関数の宣言があり、詳しくは章 1.1.7 を参照してください。

```

while(1){
    keyRead(pressed_keys);
    bool comp = keyCompare(pressed_keys, last_key_pressed);
    .....
    testword[keyIndex]=pressed_keys[0];
    keyIndex++;
    if(keyIndex==LENS){
        if(check()==0){
            clear();
            write(3, 0, "WRONG KEY!");
            write(0, 1, "please try again");
        }
    .....
}

```

キー値を読み取り、テスト配列テ스트ワードに保存する。保存されているキー値の数が 4 を超える場合、パスワードの正確さが自動的に検証され、検証結果が LCD インターフェイスに表示される。

```

int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
            {return 0;}
    }
    return 1;
}

```

パスワードの正確さを確認してください。パスワードが正しく入力された場合は 1 を返し、そうでない場合は 0 を返す。

➤ Python 言語ユーザー向け

ステップ 2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 3.1.9_PasswordLock.py
```

コードの実行後、キーパッドを使用してパスワードを入力する：1984。LCD1602 に「CORRECT」と表示されている場合、パスワードに問題はない。そうでない場合、「WRONG KEY」が表示される。

コードの説明

```
LENS = 4
password=['1','9','8','4']
.....
rowsPins = [18,23,24,25]
colsPins = [10,22,27,17]
keys = ["1","2","3","A",
        "4","5","6","B",
        "7","8","9","C",
        "*","0","#","D"]
```

ここでは、パスワード LENS の長さ、マトリックスキーボードキーを保存する配列キーと正しいパスワードを保存する配列パスワードを定義する。

```
class Keypad():
    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    ...
```

このクラスは押されたキーの値を読み取るコードである。詳細については、章 2.1.5 を参照してください。

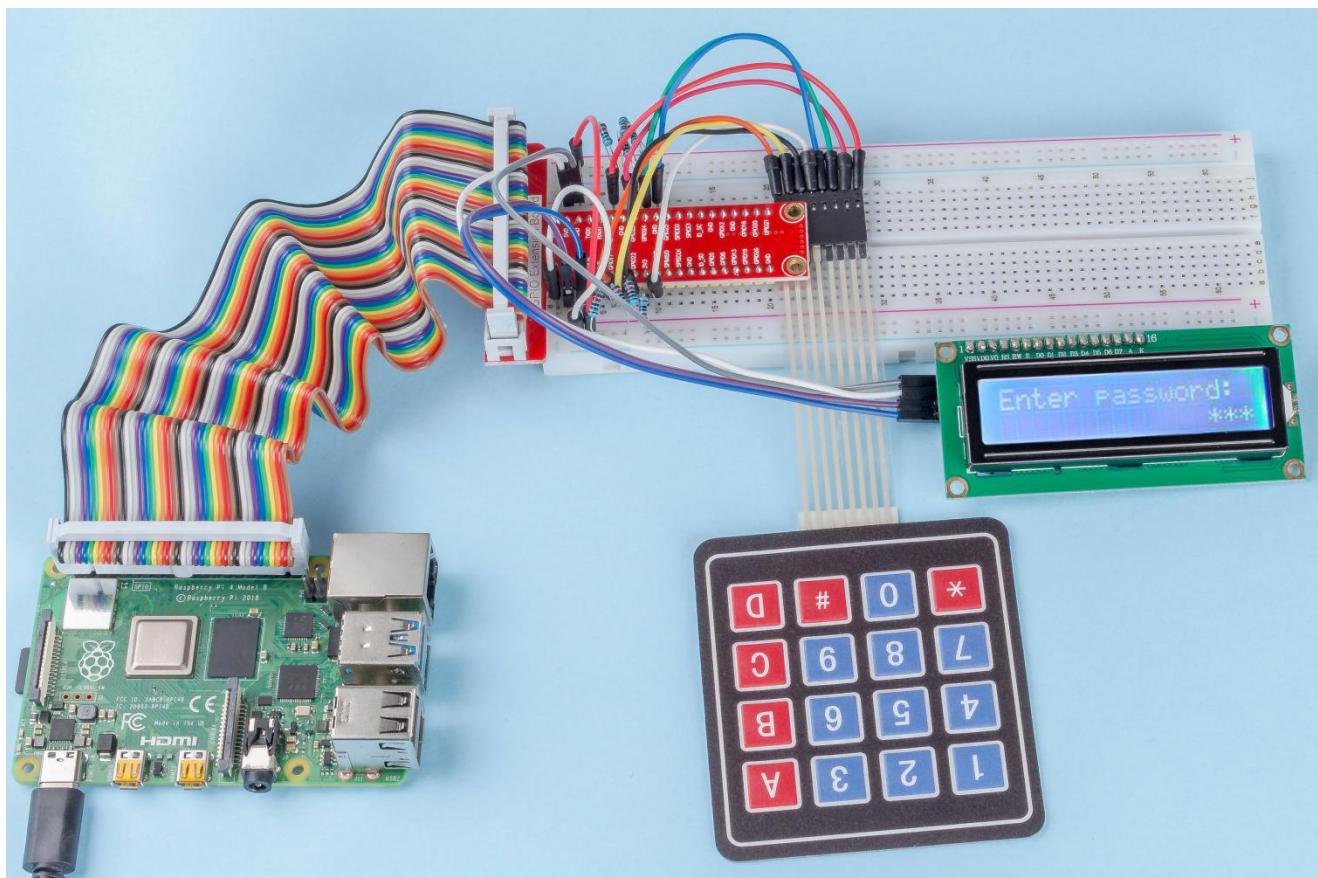
```
while(True):
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        LCD1602.clear()
        LCD1602.write(0, 0, "Enter password:")
        LCD1602.write(15-keyIndex, 1, pressed_keys)
        testword[keyIndex]=pressed_keys
        keyIndex+=1
    ...
```

キー値を読み取り、テスト配列テストワードに保存する。保存されているキー値の数が 4 を超える場合、パスワードの正確さが自動的に検証され、検証結果が LCD インターフェイスに表示される。

```
def check():
    for i in range(0,LENS):
        if(password[i]!=testword[i]):
            return 0
    return 1
```

パスワードの正確さを確認してください。パスワードが正しく入力された場合は1を返し、そうでない場合は0を返す。

現象画像

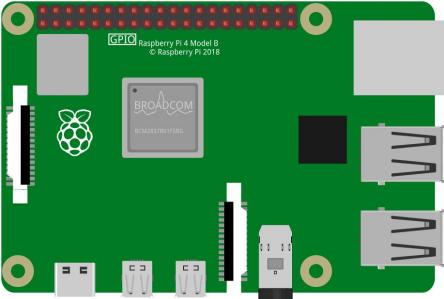
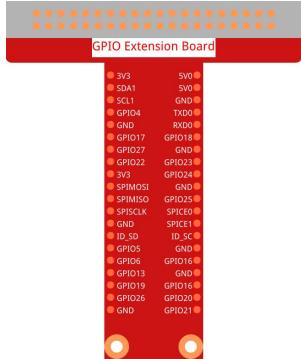
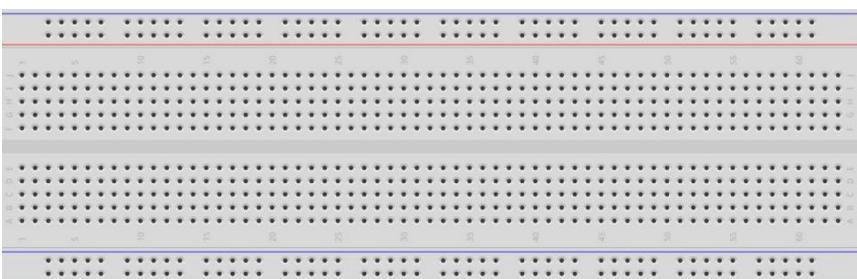
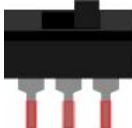
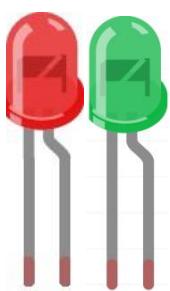
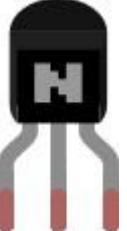


3.1.10 Alarm Bell

前書き

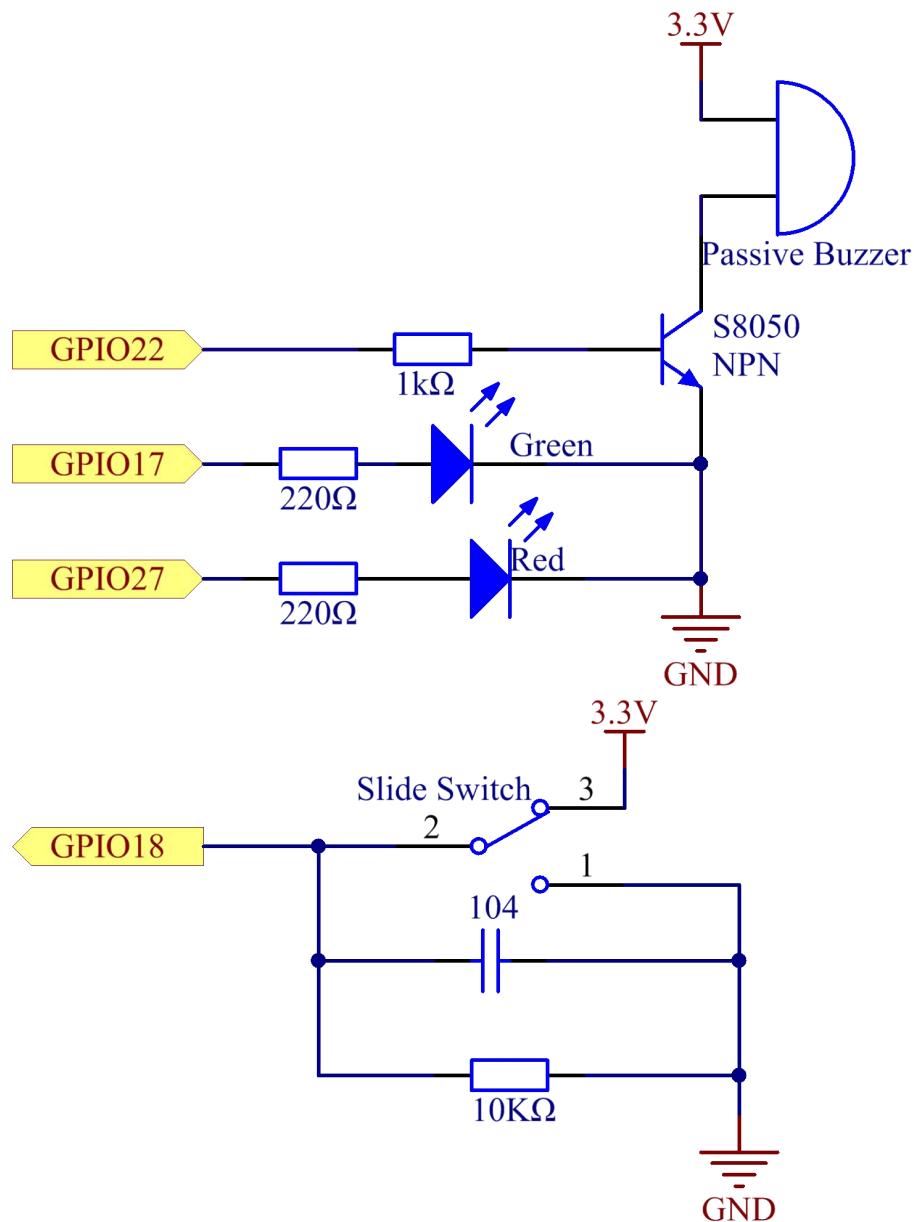
このコースでは、手動警報装置を作成する。トグルスイッチをサーミスタまたは感光センサーに交換して、温度警報または光警報を作成できる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	パッシブブザー*1
	 GPIO Extension Board Pinout: 3V3 SVO SDA1 SVO SCL1 GND GPIO4 TXD0 GND RXD0 GPIO17 GPIO8 GPIO27 GND GPIO22 GPIO23 3V3 GPIO28 SPI-MOSI GND SPI-MISO GPIO25 SPI-SCLK SPI-CE0 GND SPI-CE1 D2-D GND GPIO5 GPIO16 GPIO6 GND GPIO13 GND GPIO19 GPIO16 GPIO26 GPIO20 GND GPIO21	
40 ピンケーブル*1		抵抗器 (1KΩ) *1 
		抵抗器 (220Ω) *2 
ブレッドボード*1		抵抗器 (10KΩ) *1 
		スライドスイッチ*1 
何本のジャンパー線	LED*2 	S85050 NPN トランジスタ*1 
		104 コンデンサ*1 

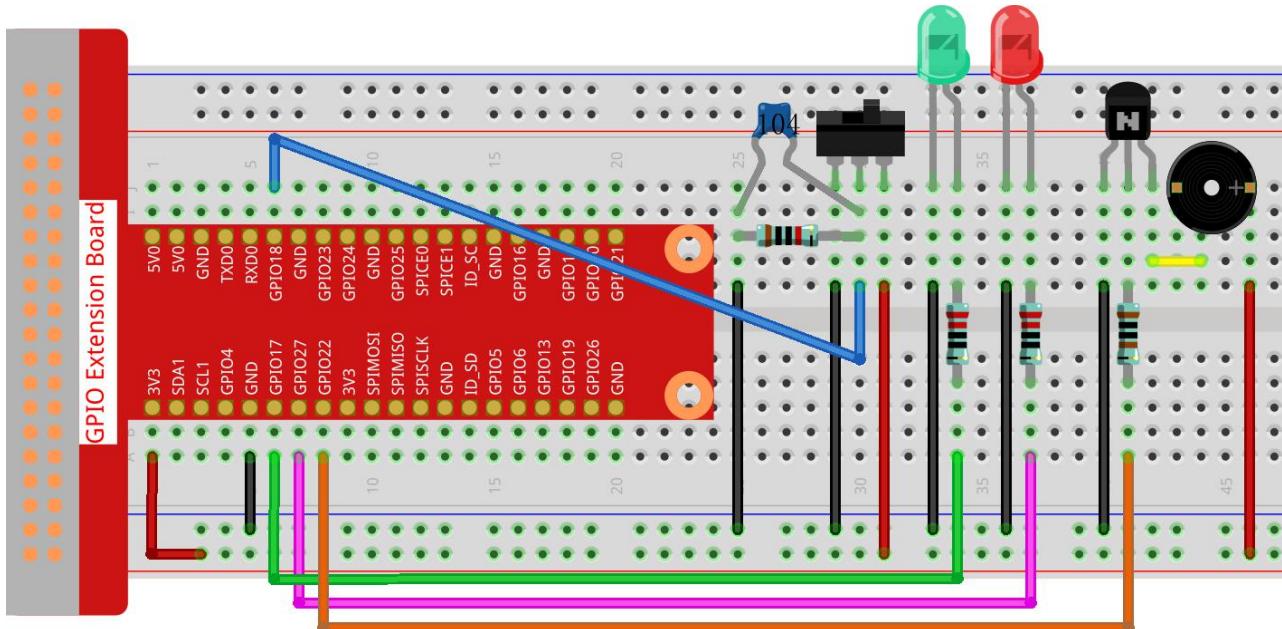
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.10/
```

ステップ3: コンパイルする。

```
gcc 3.1.10_AlarmBell.c -lwiringPi -lpthread
```

ステップ4: 実行。

```
sudo ./a.out
```

プログラムが起動すると、トグルスイッチが右に切り替わり、ブザーが警報音を出す。同時に、特定の周波数で赤と緑のLEDが点滅する。

コードの説明

```
#include <pthread.h>
```

このコードでは、新しいライブラリ pthread.h を使用する。これは、一般的なスレッドライブラリのセットであり、マルチスレッドを実現できる。コンパイル時に-lpthread パラメータを追加して、LED とブザーを独立して動作させる。

```
void *ledWork(void *arg){
    while(1)
    {
```

```

if(flag==0){
    pthread_exit(NULL);
}
digitalWrite(ALedPin,HIGH);
delay(500);
digitalWrite(ALedPin,LOW);
digitalWrite(BLedPin,HIGH);
delay(500);
digitalWrite(BLedPin,LOW);
}
}

```

関数 ledWork()は、これら 2 つの LED の動作状態を設定するために役立つ：緑色の LED を 0.5 秒間点灯させた後、消灯する。同様に、赤い LED を 0.5 秒間点灯させてから消灯する。

```

void *buzzWork(void *arg){
while(1)
{
    if(flag==0){
        pthread_exit(NULL);
    }
    if((note>=800)|| (note<=130)){
        pitch = -pitch;
    }
    note=note+pitch;
    softToneWrite(BeepPin,note);
    delay(10);
}
}

```

関数 buzzWork()は、ブザーの動作状態を設定するために使用される。ここでは、周波数を 130～800 に設定し、20 の間隔で累積・減衰する。

```

void on(){
flag = 1;
if(softToneCreate(BeepPin) == -1){
    printf("setup softTone failed !");
    return;
}
pthread_t tLed;
pthread_create(&tLed,NULL,ledWork,NULL);
pthread_t tBuzz;

```

```
pthread_create(&tBuzz,NULL,buzzWork,NULL);
}
```

関数 on()で：

- 1) 制御スレッドの終了を示すマーク 「flag = 1」 を定義する。
- 2) ソフトウェア制御のトーンピン BeepPin を作成する。
- 3) LED とブザーが同時に動作できるように、二つの個別のスレッドを作成する。

pthread_t tLed: スレッド tLed を宣言する。

pthread_create(&tLed,NULL,ledWork,NULL): スレッドを作成し、そのプロトタイプは次の通りである：

```
int pthread_create(pthread_t *restrict tidp,const pthread_attr_t *restrict_attr,void**
(*start_rtn)(void*),void *restrict arg);
```

値を返す

成功した場合は「0」を返し、それ以外の場合は落下数「-1」を返します。

パラメータ

最初のパラメーターは、スレッド ID へのポインターです。

2つ目は、スレッド属性を設定するために使用されます。

3番目は、スレッド実行関数の開始アドレスです。

最後のものは、関数を実行するものです。

```
void off(){
    flag = 0;
    softToneStop(BeepPin);
    digitalWrite(ALedPin,LOW);
    digitalWrite(BLedPin,LOW);
}
```

関数 Off () は 「flag = 0」 を定義して、スレッド ledWork と BuzzWork を終了し、ブザーと LED をオフにします。

```
int main(){
    setup();
    int lastState = 0;
    while(1){
        int currentState = digitalRead(switchPin);
        if ((currentState == 1)&&(lastState==0)){
            on();
```

```
    }
    else if((currentState == 0)&&(lastState==1)){
        off();
    }
    lastState=currentState;
}
return 0;
}
```

Main() には、プログラムのプロセス全体が含まれている：まず、スライドスイッチの値を読み取る。トグルスイッチが右に切り替えられた場合（読み取り値が 1）、関数 on () が呼び出され、ブザーが駆動されて音が鳴り、赤と緑の LED が点滅する。そうしないと、ブザーと LED が機能しない。

➤ Python 言語ユーザー向け

ステップ 2: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: 実行する。

```
sudo python3 3.1.10_AlarmBell.py
```

プログラムが起動すると、トグルスイッチが右に切り替わり、ブザーが警報音を出す。同時に、特定の周波数で赤と緑の LED が点滅する。

コードの説明

```
import threading
```

ここでは、Threading モジュールをインポートし、複数のことを一度に行えるようにするが、通常のプログラムはコードを上から下にしか実行できない。Threading モジュールを使用すると、LED とブザーを個別に動作させることができる。

```
def ledWork():
    while flag:
        GPIO.output(ALedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(ALedPin,GPIO.LOW)
        GPIO.output(BLedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(BLedPin,GPIO.LOW)
```

ledWork () 関数は、これら 2 つの LED の動作状態を設定することに役立つ：緑色の LED を 0.5 秒間点灯させた後、消灯する。同様に、赤い LED を 0.5 秒間点灯させてから消灯する。

```
def buzzerWork():
    global pitch
    global note
    while flag:
        if note >= 800 or note <=130:
            pitch = -pitch
            note = note + pitch
            Buzz.ChangeFrequency(note)
            time.sleep(0.01)
```

関数 buzzWork () は、ブザーの動作状態を設定するために使用される。ここでは、周波数を 130~800 に設定し、20 の間隔で累積・減衰する。

```
def on():
    global flag
    flag = 1
    Buzz.start(50)
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
```

関数 on () では:

- 1) 1)制御スレッドの終了を示すマーク 「flag = 1」 を定義する。
- 2) 2)バズを開始し、デューティサイクルを 50%に設定する。
- 3) 3)LED とブザーが同時に動作できるように、二つの個別のスレッドを作成する。

tBuzz = threading.Thread (target = buzzerWork) : スレッドを作成すると、そのプロトタイプは以下の通りである:

構築メソッドの中で、主要なパラメーターは target であり、呼び出し可能なオブジェクト（ここでは関数 ledWork と BuzzWork）を target に割り当てる必要がある。次に、スレッドオブジェクトを開始するために start () が呼び出される。たとえば、tBuzz.start () は、新しくインストールされた tBuzz スレッドを開始するために使用される。

```
def off():
    global flag
    flag = 0
    Buzz.stop()
    GPIO.output(ALedPin,GPIO.LOW)
```

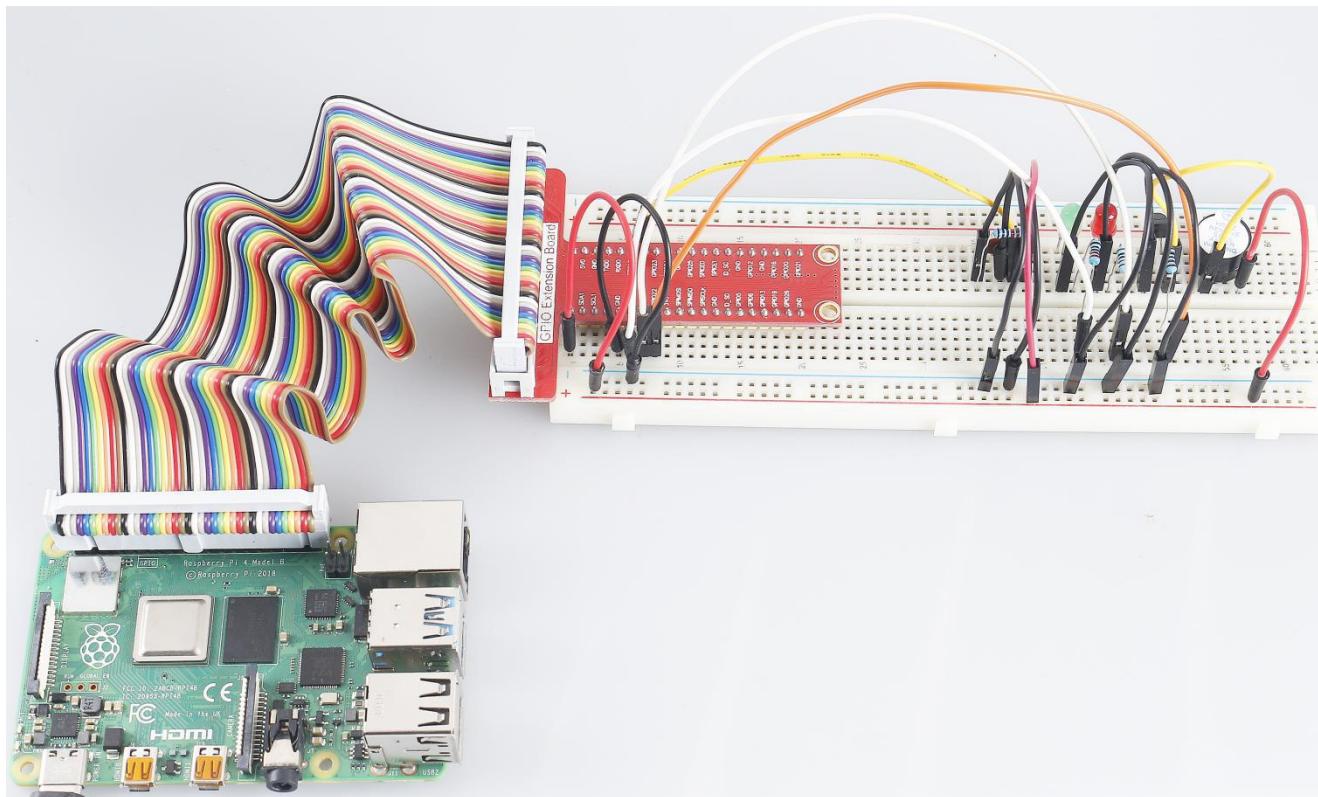
```
GPIO.output(BLedPin,GPIO.LOW)
```

スレッド ledWork と BuzzWork を終了するために関数 Off () は「flag = 0」を定義そして、ブザーと LED をオフにする。

```
def main():
    lastState=0
    while True:
        currentState =GPIO.input(switchPin)
        if currentState == 1 and lastState == 0:
            on()
        elif currentState == 0 and lastState == 1:
            off()
        lastState=currentState
```

Main () には、プログラムのプロセス全体が含まれている：まず、スライドスイッチの値を読み取る。トグルスイッチが右に切り替えられた場合（読み取り値が 1）、関数 on () が呼び出され、ブザーが駆動されて音が鳴り、赤と緑の LED が点滅する。そうしないと、ブザーと LED が機能しない。

現象画像

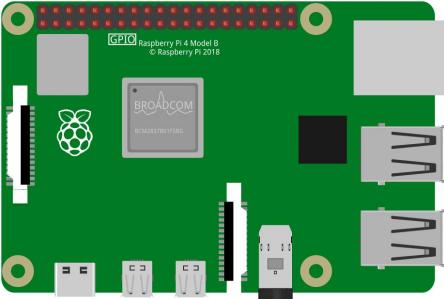
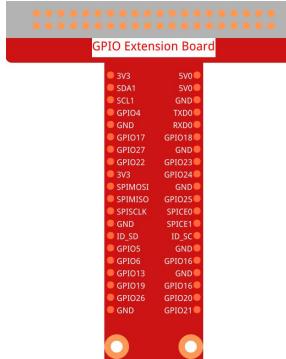
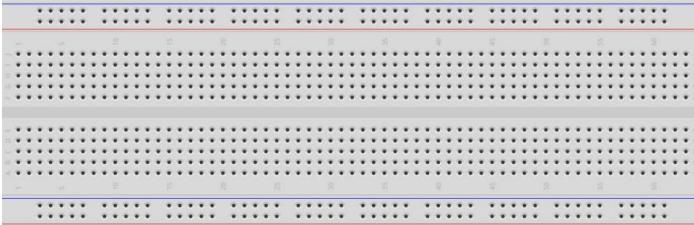


3.1.11 Morse Code Generator

前書き

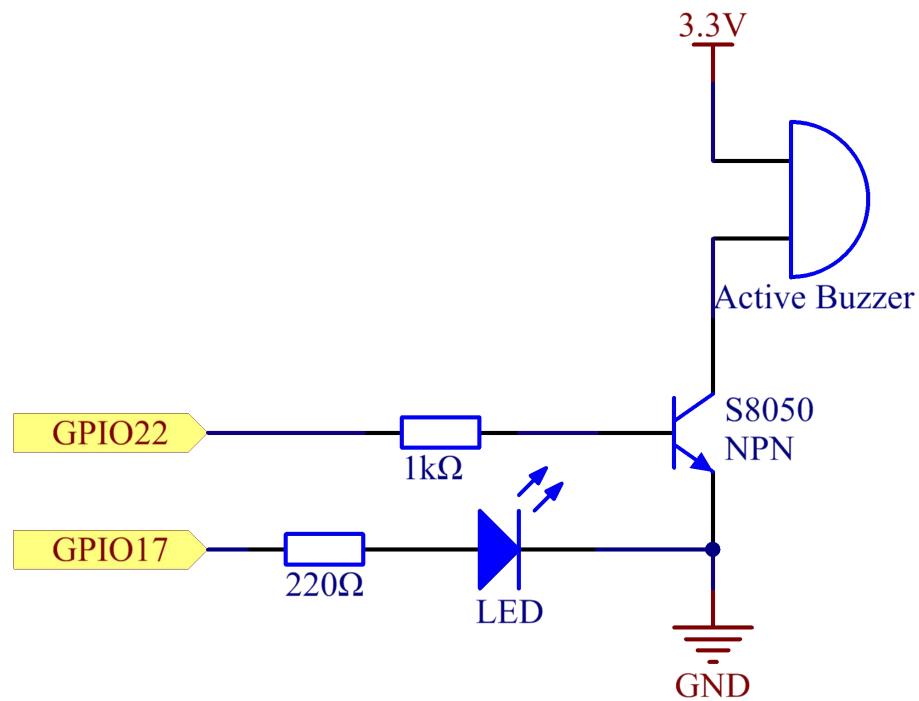
このレッスンでは、モールス符号ジェネレーターを作成する。ここでは、Raspberry Pi に一連の英語の文字を入力して、モールス符号として表示する。

部品

Raspberry Pi 本体*1 	T 拡張ボード*1 	アクティブブザー*1 
40 ピンケーブル*1 		抵抗器 (1kΩ) *1 
ブレッドボード*1 	LED*1 	抵抗器 (220Ω) *2 
		何本のジャンパー線 
		S85050 NPN トランジスタ*1 

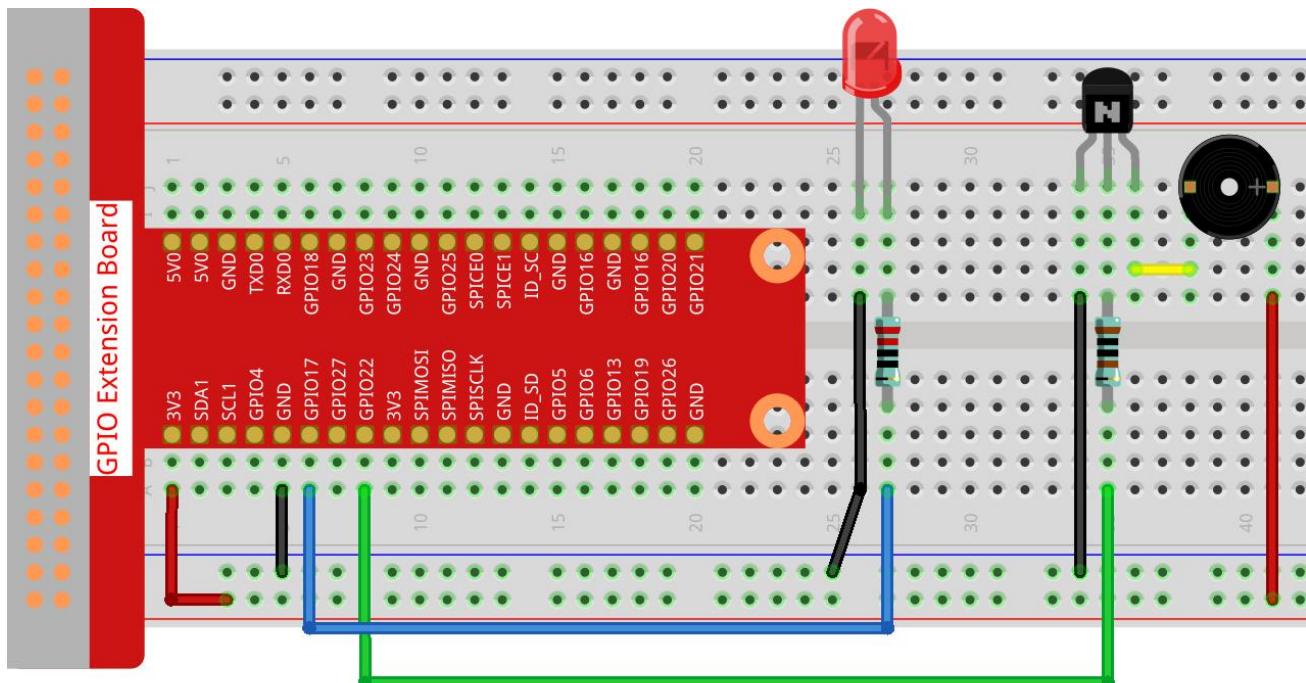
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO22	Pin 15	3	22



実験手順

ステップ 1: 回路を作る。（ブザーの両極に注意してください：+ラベルが付いている方が正極で、もう一方が負極である。）



➤ C言語ユーザー向け

ステップ2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.11/
```

ステップ3: コードをコンパイルする。

```
gcc 3.1.11_MorseCodeGenerator.c -lwiringPi
```

ステップ4: EXEファイルを実行する。

```
sudo ./a.out
```

プログラムの実行後、一連の文字を入力すると、ブザーとLEDが対応するモールス信号を送信する。

コードの説明

```
struct MORSE{
    char word;
    unsigned char *code;
};

struct MORSE morseDict[]=
{
    {'A',"01"}, {"B","1000"}, {"C","1010"}, {"D","100"}, {"E","0"},

    {"F","0010"}, {"G","110"}, {"H","0000"}, {"I","00"}, {"J","0111"},

    {"K","101"}, {"L","0100"}, {"M","11"}, {"N","10"}, {"O","111"},

    {"P","0110"}, {"Q","1101"}, {"R","010"}, {"S","000"}, {"T","1"},

    {"U","001"}, {"V","0001"}, {"W","011"}, {"X","1001"}, {"Y","1011"},

    {"Z","1100"}, {"1","01111"}, {"2","00111"}, {"3","00011"}, {"4","00001"},

    {"5","00000"}, {"6","10000"}, {"7","11000"}, {"8","11100"}, {"9","11110"},

    {"0","11111"}, {"?","001100"}, {"/","10010"}, {"'","110011"}, {"'',"010101"},

    {";","101010"}, {"!","101011"}, {"@","011010"}, {":,"111000"}
};
```

この構造MORSEは、モールス符号のディクショナリで、文字A～Z、数字0～9とマークを含む?" "/" ":" ";" "." ";" "!" "@"。

```
char *lookup(char key,struct MORSE *dict,int length)
{
    for (int i=0;i<length;i++)
    {
        if(dict[i].word==key){
            return dict[i].code;
        }
    }
}
```

```
}
```

```
}
```

```
}
```

関数 `lookup ()` は、「`checking the dictionary`」によって機能する。キーを定義し、構造 `morseDict` のキーと同じ単語を検索し、対応する情報（特定の単語の「code」）を返す。

```
void on(){
    digitalWrite(ALedPin,HIGH);
    digitalWrite(BeepPin,HIGH);
}
```

関数 `on ()` を作成して、ブザーと LED を起動する。

```
void off(){
    digitalWrite(ALedPin,LOW);
    digitalWrite(BeepPin,LOW);
}
```

関数 `off ()` はブザーと LED をオフにする。

```
void beep(int dt){
    on();
    delay(dt);
    off();
    delay(dt);
}
```

関数 `beep ()` を定義して、ブザーと LED が特定の `dt` 間隔で音を鳴らして点滅するようになる。

```
void morsecode(char *code){
    int pause = 250;
    char *point = NULL;
    int length = sizeof(morseDict)/sizeof(morseDict[0]);
    for (int i=0;i<strlen(code);i++)
    {
        point=lookup(code[i],morseDict,length);
        for (int j=0;j<strlen(point);j++){
            if (point[j]=='0')
            {
                beep(pause/2);
            }else if(point[j]=='1')
            {
```

```
    beep(pause);  
}  
delay(pause);  
}  
}  
}
```

関数 `morsecode()` はコードの「1」が音または光を放射し続け、「0」が音または光を短時間放射することにより、入力文字のモールス符号を処理するために使用される。たとえば、「SOS」を入力すると、3つの短い、3つの長いと3つの短いセグメントを含む信号になる“· - · - · - · - ·”。

```
int toupper(int c)
{
    if ((c >= 'a') && (c <= 'z'))
        return c + ('A' - 'a');
    return c;
}

char *strupr(char *str)
{
    char *origin=str;
    for (; *str!='\0'; str++)
        *str = toupper(*str);
    return origin;
}
```

コーディングする前に、文字を大文字に統一しなければならない。

```
void main(){
    setup();
    char *code;
    int length=8;
    code = (char*)malloc(sizeof(char)*length);
    while (1){
        printf("Please input the messenger:");
        scanf("%s",code);
        code=strupr(code);
        printf("%s\n",code);
        morsecode(code);
    }
}
```

キーボードで関連する文字を入力すると、code = strupr (code) は入力文字を大文字に変換する。

Printf() はコンピューター画面にクリアテキストをプリントし、morsecod () 関数はブザーと LED からモールス符号を出力させる。

入力文字の長さは、length を超えないことに注意してください（訂正可能）。

➤ Python 言語ユーザー向け

ステップ 2: コードファイルを開く。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 3: 実行する。

```
sudo python3 3.1.11_MorseCodeGenerator.py
```

プログラムの実行後、一連の文字を入力すると、ブザーと LED が対応するモールス信号を送信する。

コードの説明

```
MORSECODE = {
    'A':'01', 'B':'1000', 'C':'1010', 'D':'100', 'E':'0', 'F':'0010', 'G':'110',
    'H':'0000', 'I':'00', 'J':'0111', 'K':'101', 'L':'0100', 'M':'11', 'N':'10',
    'O':'111', 'P':'0110', 'Q':'1101', 'R':'010', 'S':'000', 'T':'1',
    'U':'001', 'V':'0001', 'W':'011', 'X':'1001', 'Y':'1011', 'Z':'1100',
    '1':'01111', '2':'00111', '3':'00011', '4':'00001', '5':'00000',
    '6':'10000', '7':'11000', '8':'11100', '9':'11110', '0':'11111',
    '?':'001100', '/':'10010', ';':'110011', ':':'010101', '@':'101010',
    '!':'101011', '@':'011010', ':':'111000',
}
```

この構造 MORSE は、モールス符号のディクショナリで、文字 A~Z、数字 0~9 とマークを含む "?" "/" ":" ";" "." ";" "!" "@" 。

```
def on():
    GPIO.output(BeepPin, 1)
    GPIO.output(ALedPin, 1)
```

関数 on () はブザーと LED を起動する。

```
def off():
    GPIO.output(BeepPin, 0)
    GPIO.output(ALedPin, 0)
```

関数 off () はブザーと LED をオフにする。

```
def beep(dt): # x for delay time.
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

関数 beep () を定義して、ブザーと LED が特定の dt 間隔で音を鳴らして点滅するようになる。

```
def morsecode(code):
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause/2)
            if tap == '1':
                beep(pause)
    time.sleep(pause)
```

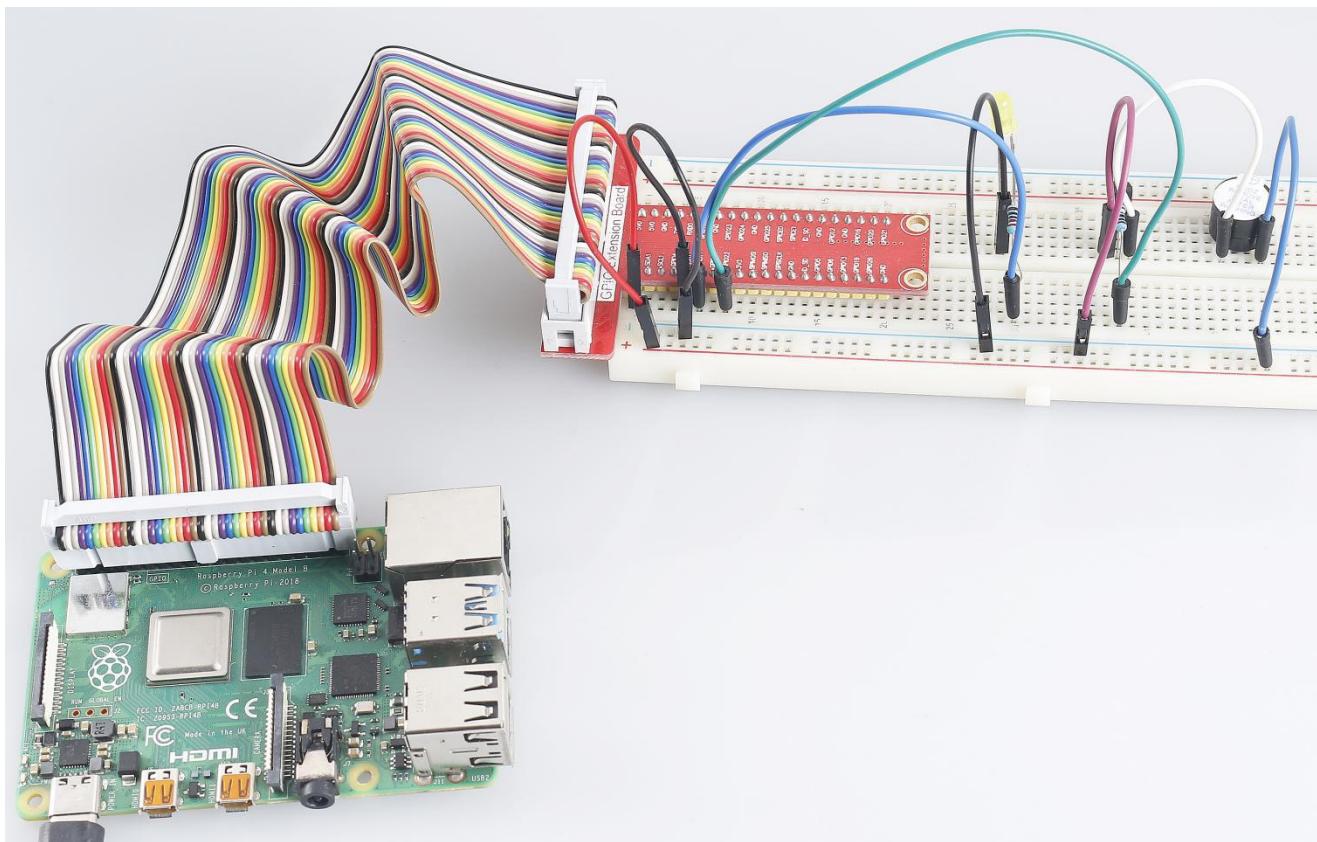
関数 morsecode () はコードの「1」が音または光を放射し続け、「0」が音または光を短時間放射することにより、入力文字のモールス符号を処理するために使用される。たとえば、「SOS」を入力すると、3つの短い、3つの長いと3つの短いセグメントを含む信号になる“ · · · - - - · · · ”。

```
def main():
    while True:
        code=input("Please input the messenger:")
        code = code.upper()
        print(code)
        morsecode(code)
```

キーボードで関連する文字を入力すると、upper()は入力文字を大文字に変換する。

printf () はコンピューター画面にクリアテキストをプリントし、morsecod () 関数はブザーと LED からモールス符号を出力させる。

現象画像



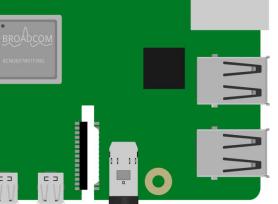
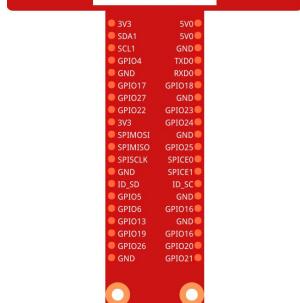
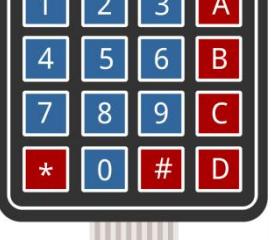
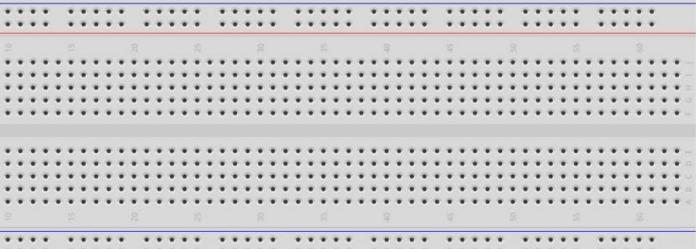
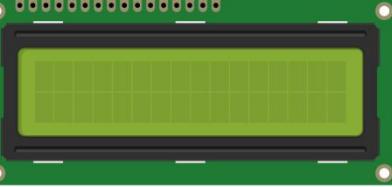
3.1.12 GAME– Guess Number

前書き

数字を推測することは、あなたとあなたの友人が交互に数字を入力する楽しいパーティーゲームである（0～99）。プレーヤーがなぞなぞに正しく答えるまで、数字を入力すると範囲は小さくなる。その後、プレイヤーは敗北し、処罰される。

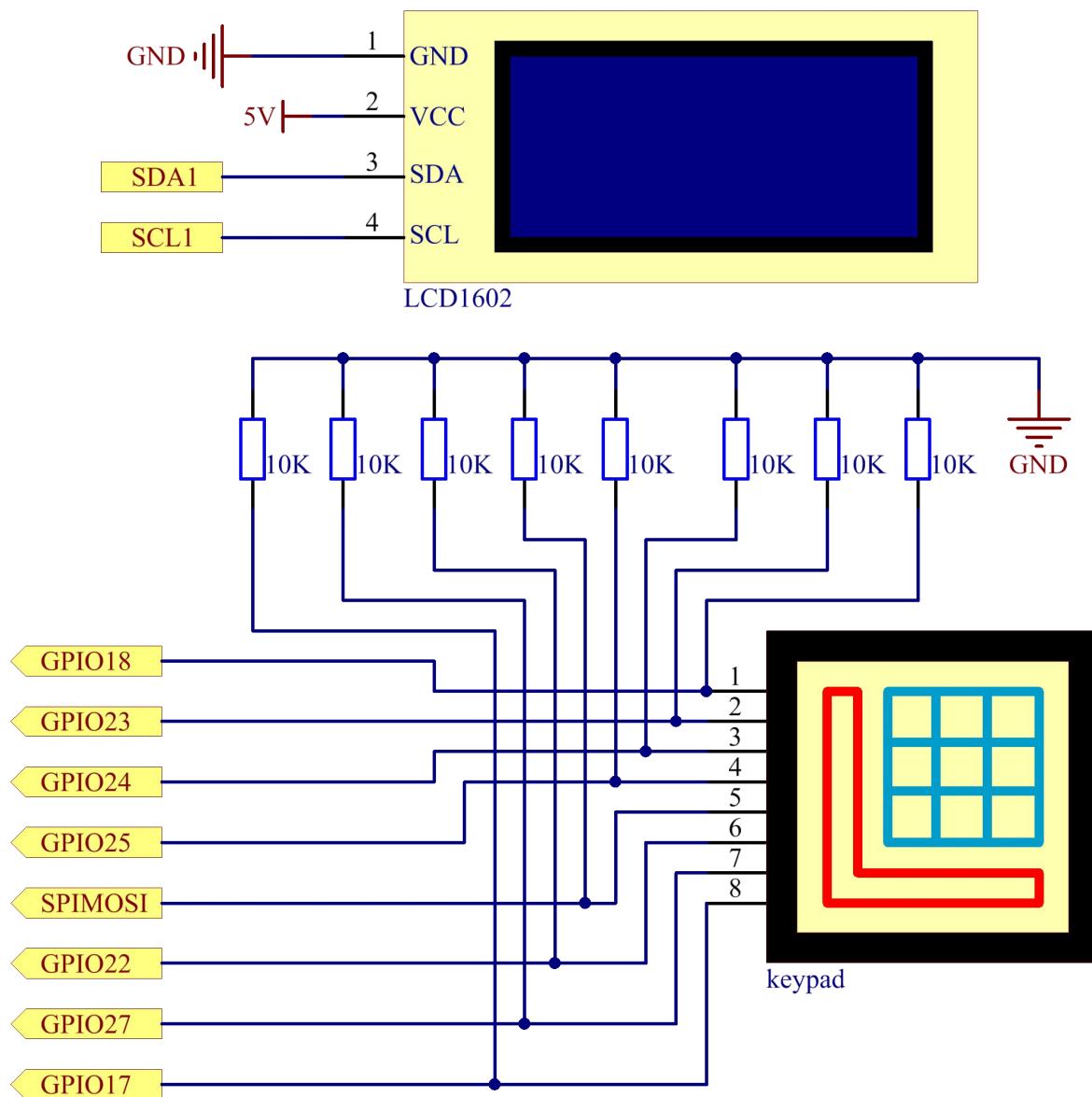
たとえば、ラッキーナンバーが 51 で、プレーヤーがそれを見ることができず、プレーヤー①が 50 を入力する場合、番号範囲は 50~99 に変わる。もしプレイヤー②が 70 を入力する場合、番号の範囲は 50~70 になる。プレイヤー③が 51 を入力した場合、このプレーヤーは不運なプレーヤーである。ここでは、キーパッドを使用して数字を入力し、LCD を使用して結果を表示させる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	キーパッド*1
		
40 ピンケーブル*1		何本のジャンパー線
		
ブレッドボード*1		抵抗器 (10KΩ) *8
		
		I2C LCD1602*1
		

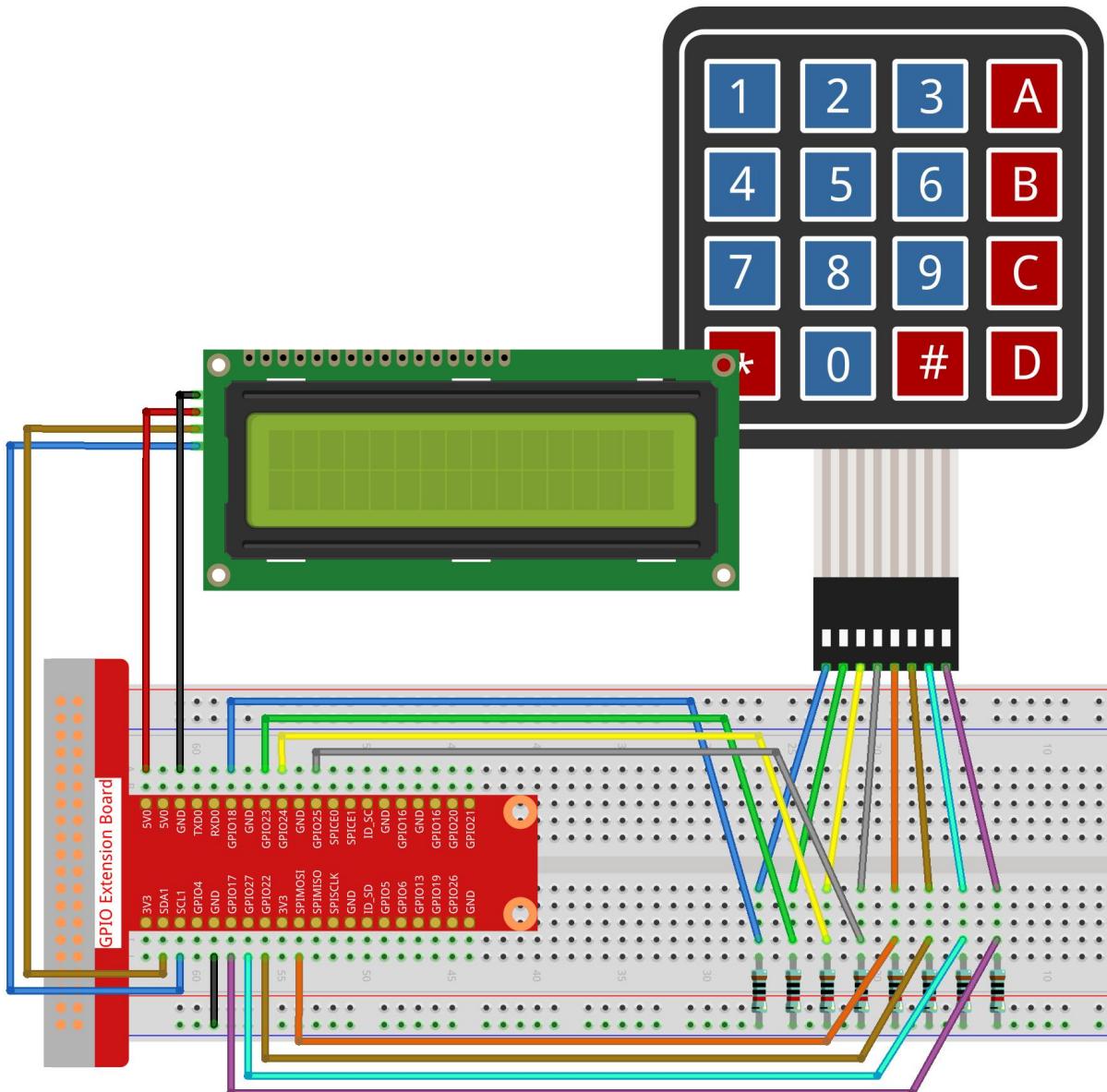
回路図

T ボード名	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17
SDA1	Pin 3	SDA1(8)	SDA1(2)
SCL1	Pin 5	SCL1(9)	SDA1(3)



実験手順

ステップ 1: 回路を作る。



ステップ 2: I2C 設定 (付録を参照してください。I2C を設定している場合は、この手順をスキップしてください。)

➤ C言語ユーザー向け

ステップ3: ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.12/
```

ステップ4: コンパイルする。

gcc 3.1.12 GAME GuessNumber.c -lwiringPi

ステップ 5: 実行する。

`sudo ./a.out`

プログラムの実行後、LCD に最初のページが表示される：

```
Welcome!  
Press A to go!
```

「A」を押すと、ゲームが開始され、ゲームページが LCD に表示される。

```
Enter number:  
0 <point> 99
```

乱数の「point」が生成されるが、ゲームの開始時に LCD に表示されない。必要なのはそれを推測することである。最終計算が完了するまで入力した数値は最初の行の最後に表示される。（「D」を押して比較を開始し、入力値が 10 より大きい場合、自動比較が開始される。）

「point」の番号範囲が 2 行目に表示される。そして、範囲内の数値を入力しなければならない。数値を入力すると、範囲が狭くなる。幸運にも不運にもラッキーナンバーを獲得した場合、「You've got it!」と表示される

コードの説明

コードの最初の部分はキーパッドと I2C LCD1602 の機能である。それらの詳細については、**1.1.7 I2C LCD1602** と **2.1.5 Keypad** を参照ください。

ここで、以下のことをわかる必要がある：

```
/******************************************/  
//Start from here  
/******************************************/  
void init(void){  
    fd = wiringPiI2CSetup(LCDAddr);  
    lcd_init();  
    lcd_clear();  
    for(int i=0 ; i<4 ; i++) {  
        pinMode(rowPins[i], OUTPUT);  
        pinMode(colPins[i], INPUT);  
    }  
    lcd_clear();  
    write(0, 0, "Welcome!");  
    write(0, 1, "Press A to go!");  
}
```

この関数は元々 I2C LCD1602 とキーパッドを定義し、「Welcome!」と「Press A to go!」を表示するために使用される。

```
void init_new_value(void){
    srand(time(0));
    pointValue = rand()%100;
    upper = 99;
    lower = 0;
    count = 0;
    printf("point is %d\n",pointValue);
}
```

この関数は乱数「point」を生成し、ポイントの範囲ヒントをリセットしする。

```
bool detect_point(void){
    if(count > pointValue){
        if(count < upper){
            upper = count;
        }
    }
    else if(count < pointValue){
        if(count > lower){
            lower = count;
        }
    }
    else if(count == pointValue){
        count = 0;
        return 1;
    }
    count = 0;
    return 0;
}
```

`detect_point()` は入力番号を生成された「point」と比較する。比較結果が同じではない場合、`count` は値を `upper` と `lower` に割り当て、「0」を返す。それ以外の場合、結果が同じであると示す場合、「1」を返す。

```
void lcd_show_input(bool result){
    char *str=NULL;
    str =(char*)malloc(sizeof(char)*3);
    lcd_clear();
    if (result == 1){
        write(0,1,"You've got it!");
        delay(5000);
        init_new_value();
```

```

lcd_show_input(0);
return;
}
write(0,0,"Enter number:");
Int2Str(str,count);
write(13,0,str);
Int2Str(str,lower);
write(0,1,str);
write(3,1,<Point<);
Int2Str(str,upper);
write(12,1,str);
}

```

この関数はゲームページを表示するために使用される。関数 Int2Str (str、count) に注意してください。lcd を正しく表示するために、これらの変数 count、lower と upper を整数から文字列に変換する。

```

int main(){
unsigned char pressed_keys[BUTTON_NUM];
unsigned char last_key_pressed[BUTTON_NUM];
if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
    printf("setup wiringPi failed !");
    return 1;
}
init();
init_new_value();
while(1){
    keyRead(pressed_keys);
    bool comp = keyCompare(pressed_keys, last_key_pressed);
    if (!comp){
        if(pressed_keys[0] != 0){
            bool result = 0;
            if(pressed_keys[0] == 'A'){
                init_new_value();
                lcd_show_input(0);
            }
            else if(pressed_keys[0] == 'D'){
                result = detect_point();
                lcd_show_input(result);
            }
        }
    }
}

```

```
else if(pressed_keys[0] >='0' && pressed_keys[0] <= '9'){
    count = count * 10;
    count = count + (pressed_keys[0] - 48);
    if (count>=10){
        result = detect_point();
    }
    lcd_show_input(result);
}
keyCopy(last_key_pressed, pressed_keys);
}
delay(100);
}
return 0;
}
```

Main() には、以下に示すように、プログラムのプロセス全体が含まれている：

- 1) I2C LCD1602 とキーパッドを初期化する。
- 2) init_new_value () を使用して、0～99 の乱数を作成する。
- 3) ボタンが押されているかどうかを判断し、ボタンの読み取り値を取得する。
- 4) ボタン「A」を押すと、0～99 の乱数が表示され、ゲームが開始される。
- 5) ボタン「D」が押されたことが検出されると、プログラムは結果判定に入り、LCD に結果を表示する。このステップは数字を 1 つだけ押してからボタン「D」を押したときの結果を判断するために役立つ。
- 6) ボタン 0-9 が押されると、カウントの値が変更される。カウントが 10 より大きい場合、判定が開始される。
- 7) ゲームの変化とその値は LCD1602 に表示される。

➤ Python 言語ユーザー向け

ステップ 3：ディレクトリを変更する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 4：実行。

```
sudo python3 3.1.12_GAME_GuessNumber.py
```

プログラムの実行後、LCD に最初のページが表示される：

```
Welcome!
Press A to go!
```

「A」を押すと、ゲームが開始され、ゲームページが LCD に表示される。

```
Enter number:
0 <point< 99
```

乱数の「point」が生成されるが、ゲームの開始時に LCD に表示されない。必要なのはそれを推測することである。最終計算が完了するまで入力した数値は最初の行の最後に表示される。（「D」を押して比較を開始し、入力値が 10 より大きい場合、自動比較が開始される。）

「ポイント」の番号範囲が 2 行目に表示される。そして、範囲内の数値を入力しなければならない。数値を入力すると、範囲が狭くなる。幸運にも不運にもラッキーナンバーを獲得した場合、「You've got it!」と表示される。

コードの説明

コードの最初の部分はキーパッドと I2C LCD1602 の機能である。それらの詳細については、1.1.7 I2C LCD1602 と 2.1.5 Keypad を参照ください。

ここで、以下のことをわかる必要がある：

```
def init_new_value():
    global pointValue,upper,count,lower
    pointValue = random.randint(0,99)
    upper = 99
    lower = 0
    count = 0
    print('point is %d' %(pointValue))
```

この関数は乱数「point」を生成し、ポイントの範囲ヒントをリセットしする。

```
def detect_point():
    global count,upper,lower
    if count > pointValue:
        if count < upper:
            upper = count
    elif count < pointValue:
        if count > lower:
            lower = count
    elif count == pointValue:
        count = 0
```

```

    return 1
count = 0
return 0

```

`detect_point()` は入力番号(count)を生成された「point」と比較する。比較結果が同じではない場合、countは値を upper と lower に割り当て、「0」を返す。それ以外の場合、結果が同じであると示す場合、「1」を返す。

```

def lcd_show_input(result):
LCD1602.clear()
if result == 1:
    LCD1602.write(0,1,'You have got it!')
    time.sleep(5)
    init_new_value()
    lcd_show_input(0)
    return
LCD1602.write(0,0,'Enter number:')
LCD1602.write(13,0,str(count))
LCD1602.write(0,1,str(lower))
LCD1602.write(3,1,' < Point < ')
LCD1602.write(13,1,str.upper())

```

この関数はゲームページを表示するために使用される。

`str(count)`: `write()`はデータ型のみをサポートできるため、文字列、`str()`は数値を文字列に変換するために必要である。

```

def loop():
global keypad, last_key_pressed, count
while(True):
    result = 0
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        if pressed_keys == ["A"]:
            init_new_value()
            lcd_show_input(0)
        elif pressed_keys == ["D"]:
            result = detect_point()
            lcd_show_input(result)
        elif pressed_keys[0] in keys:
            if pressed_keys[0] in list(["A","B","C","D","#","*"]):
                continue
            count = count * 10

```

```

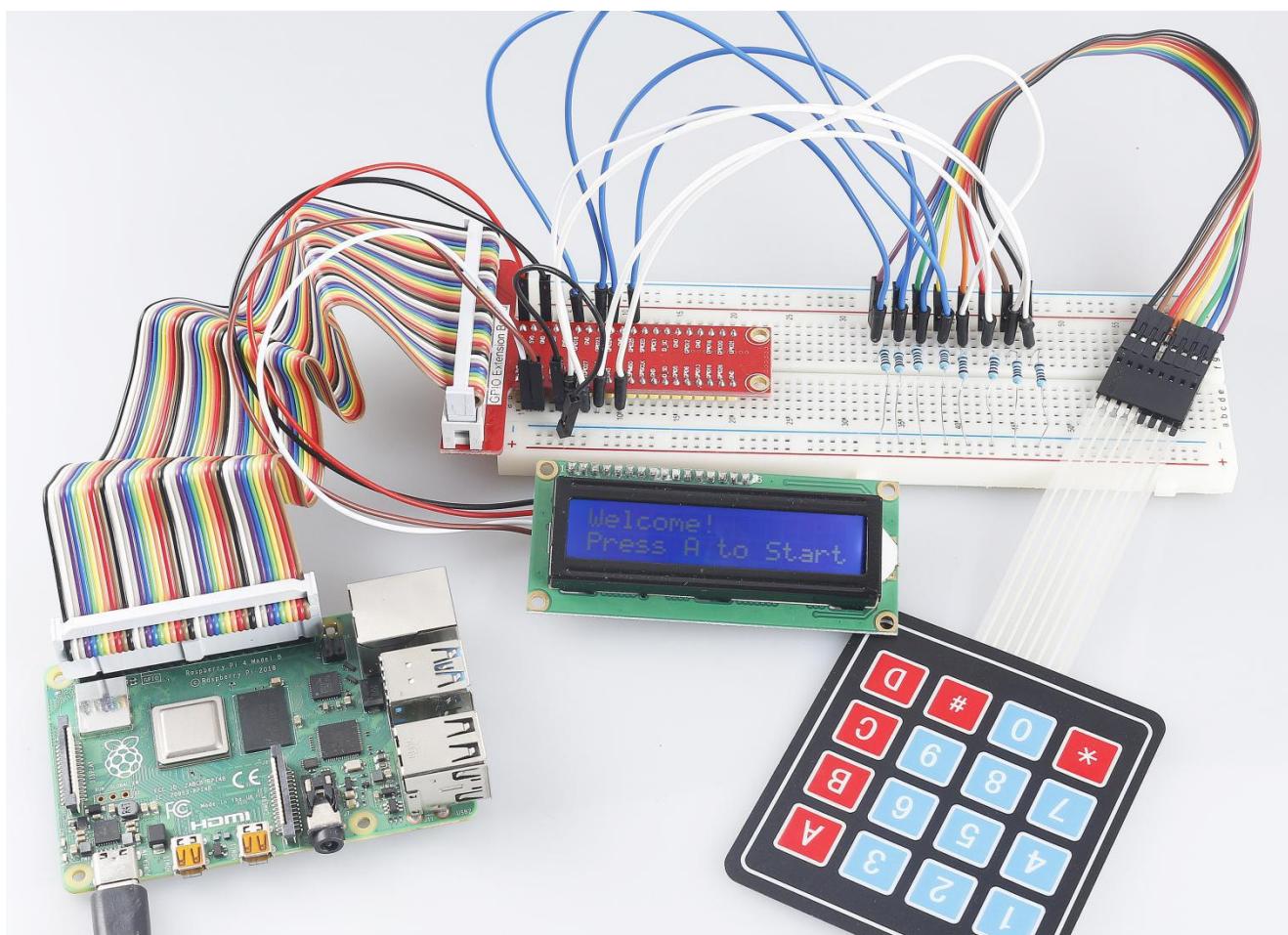
count += int(pressed_keys[0])
if count >= 10:
    result = detect_point()
    lcd_show_input(result)
    print(pressed_keys)
last_key_pressed = pressed_keys
time.sleep(0.1)

```

Main() には、以下に示すように、プログラムのプロセス全体が含まれている：

- 1) I2C LCD1602 とキーパッドを初期化する。
- 2) ボタンが押されているかどうかを判断し、ボタンの読み取り値を取得する。
- 3) ボタン「A」を押すと、0~99 の乱数が表示され、ゲームが開始される。
- 4) ボタン「D」が押されたことが検出されると、プログラムは結果判定に入る。
- 5) ボタン 0-9 が押されると、カウントの値が変更される。カウントが 10 より大きい場合、判定が開始される。
- 6) ゲームの変化とその値は LCD1602 に表示される。

現象画像

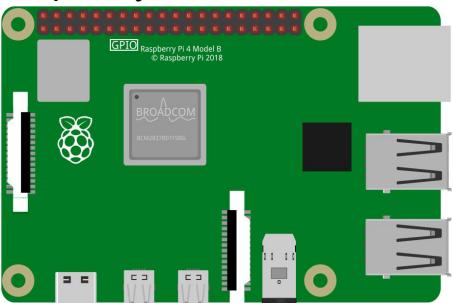
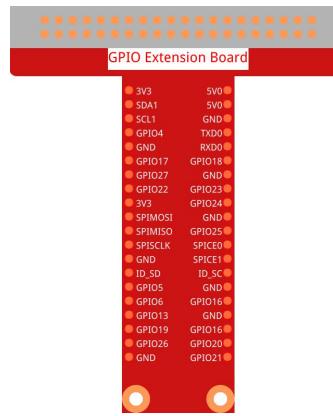
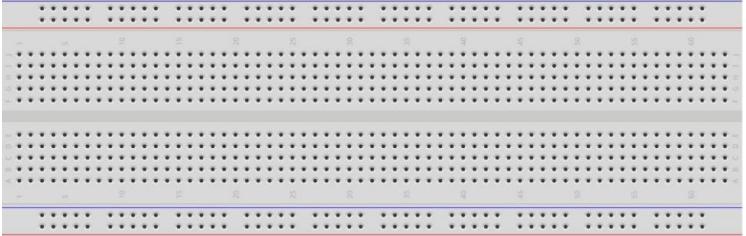


3.1.13 GAME- 10 Second

前書き

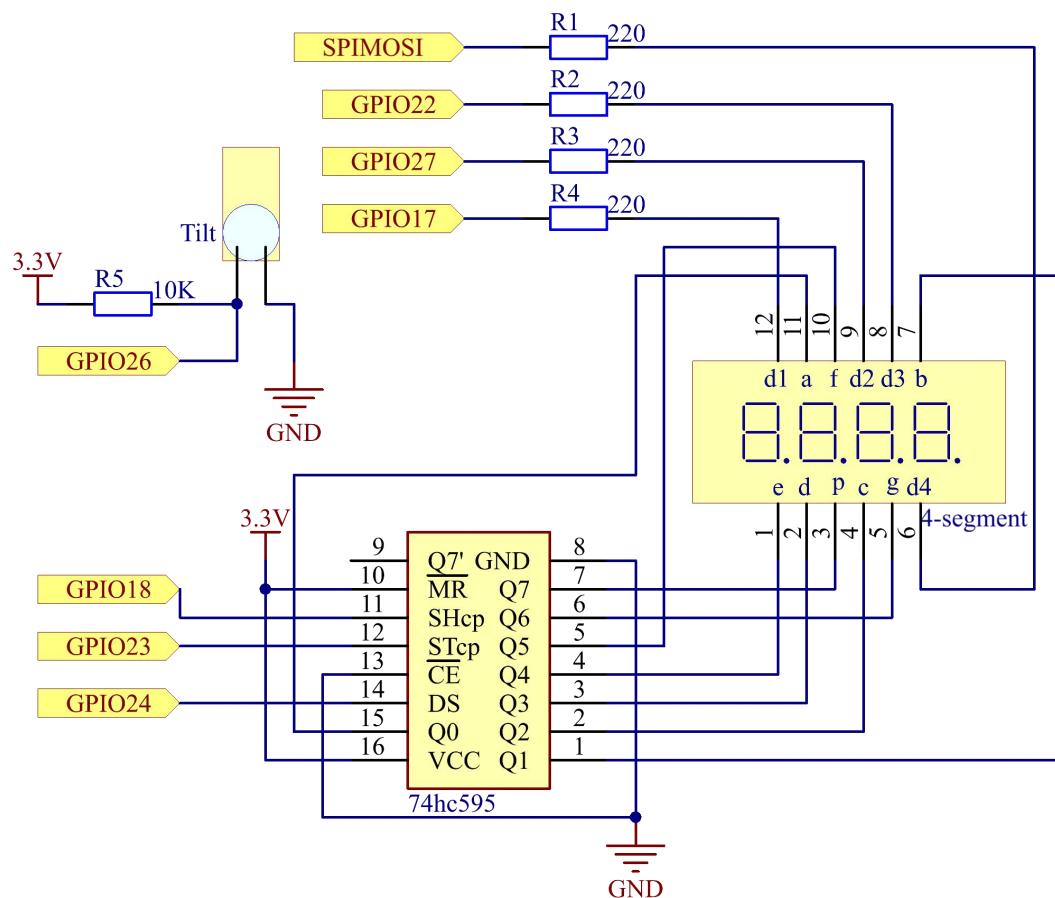
それから、あなたの集中力に挑戦できるゲームデバイスを作ろう。傾斜スイッチをステイックにつないで、魔法の棒を作る。棒を振ると、4桁のセグメントディスプレイがカウントを開始し、もう一度振るとカウントを停止する。表示されたカウントを 10.00 に保つことに成功した場合、あなたが勝つ。友達とゲームをプレイして、タイムウィザードが誰かを確認できる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	4 術 7 セグメント ディスプレイ*1																																								
	 <table border="1"><tr><td>3V3</td><td>5V0</td></tr><tr><td>SDA1</td><td>VDD</td></tr><tr><td>SCL1</td><td>GND</td></tr><tr><td>GPIO4</td><td>TXD0</td></tr><tr><td>GND</td><td>RXD0</td></tr><tr><td>GPIO17</td><td>GPIO18</td></tr><tr><td>GPIO27</td><td>GND</td></tr><tr><td>GPIO22</td><td>GPIO23</td></tr><tr><td>3V3</td><td>GPIO24</td></tr><tr><td>SPIMOSI</td><td>GND</td></tr><tr><td>SPIMISO</td><td>GPIO25</td></tr><tr><td>SPISCLK</td><td>SPICE0</td></tr><tr><td>GND</td><td>SPICE1</td></tr><tr><td>ID_SD</td><td>ID_SC</td></tr><tr><td>GPIO5</td><td>GND</td></tr><tr><td>GPIO6</td><td>GPIO16</td></tr><tr><td>GPIO13</td><td>GND</td></tr><tr><td>GPIO19</td><td>GPIO16</td></tr><tr><td>GPIO26</td><td>GPIO20</td></tr><tr><td>GND</td><td>GPIO21</td></tr></table>	3V3	5V0	SDA1	VDD	SCL1	GND	GPIO4	TXD0	GND	RXD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPIMOSI	GND	SPIMISO	GPIO25	SPISCLK	SPICE0	GND	SPICE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	
3V3	5V0																																									
SDA1	VDD																																									
SCL1	GND																																									
GPIO4	TXD0																																									
GND	RXD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPIMOSI	GND																																									
SPIMISO	GPIO25																																									
SPISCLK	SPICE0																																									
GND	SPICE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
40 ピンケーブル*1	74HC595*1	傾斜スイッチ*1																																								
																																										
ブレッドボード*1	何本のジャンパー線	抵抗器 (220Ω) * 4																																								
																																										
		抵抗器 (10KΩ) * 4																																								
																																										

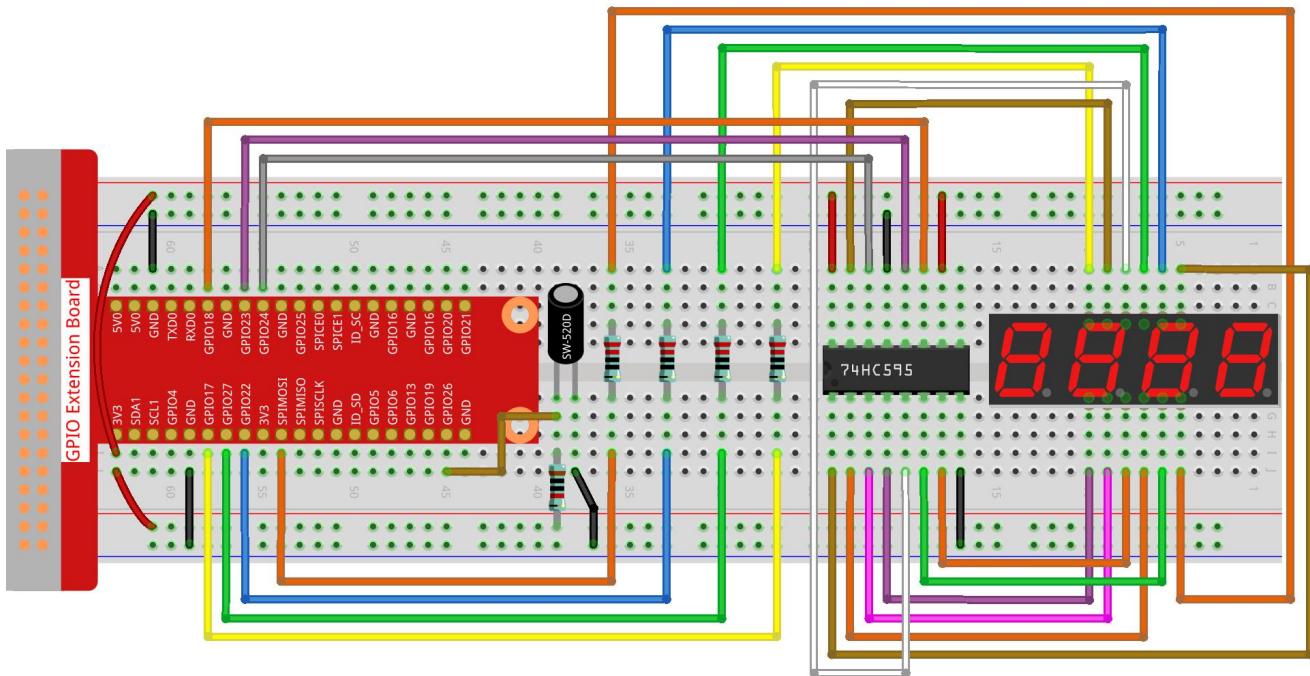
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO26	Pin 37	25	26



実験手順

ステップ1: 回路を作る。



➤ C言語ユーザー向け

ステップ2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.13/
```

ステップ3: コードをコンパイルする。

```
gcc 3.1.13_GAME_10Second.c -lwiringPi
```

ステップ4: EXE ファイルを実行する。

```
sudo ./a.out
```

棒を振ると、4桁のセグメントディスプレイがカウントを開始し、もう一度振るとカウントを停止する。表示されたカウントを 10.00 に保つことに成功した場合、あなたが勝つ。もう一度振って、ゲームの次のラウンドを開始する。

コードの説明

```
void stateChange(){
    if (gameState == 0){
        counter = 0;
        delay(1000);
        ualarm(10000,10000);
    }else{
        alarm(0);
        delay(1000);
    }
}
```

```
    }
    gameState = (gameState + 1)%2;
}
```

ゲームは二つのモードに分けられている：

gameState = 0 は「開始」モードであり、このモードでは、時間を計測してセグメントディスプレイに表示し、傾斜スイッチを振って「表示」モードに入る。

GameState = 1 は「表示」モードであり、タイミングを停止し、セグメントディスプレイに時間を表示する。傾斜スイッチを再度振ると、タイマーがリセットされ、ゲームが再起動する。

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            stateChange();
        }
        lastState=currentState;
    }
}
```

Loop() が主な関数である。最初に、4 ビットセグメントディスプレイに時間が表示され、傾斜スイッチの値が読み取られる。傾斜スイッチの状態が変更された場合、stateChange() が呼び出される。

➤ Python 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/
```

ステップ 3: EXE ファイルを実行する。

```
sudo python3 3.1.13_GAME_10Second.py
```

棒を振ると、4 行のセグメントディスプレイがカウントを開始し、もう一度振るとカウントを停止する。表示されたカウントを 10.00 に保つことに成功した場合、あなたが勝つ。もう一度振って、ゲームの次のラウンドを開始する。

コードの説明

```
def stateChange():
    global gameState
    global counter
    global timer1
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState == 1:
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState+1)%2
```

ゲームは二つのモードに分けられている：

`gameState = 0` は「開始」モードであり、このモードでは、時間を計測してセグメントディスプレイに表示し、傾斜スイッチを振って「表示」モードに入る。

`GameState = 1` は「表示」モードであり、タイミングを停止し、セグメントディスプレイに時間を表示する。傾斜スイッチを再度振ると、タイマーがリセットされ、ゲームが再起動する。

```
def loop():
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState=currentState
```

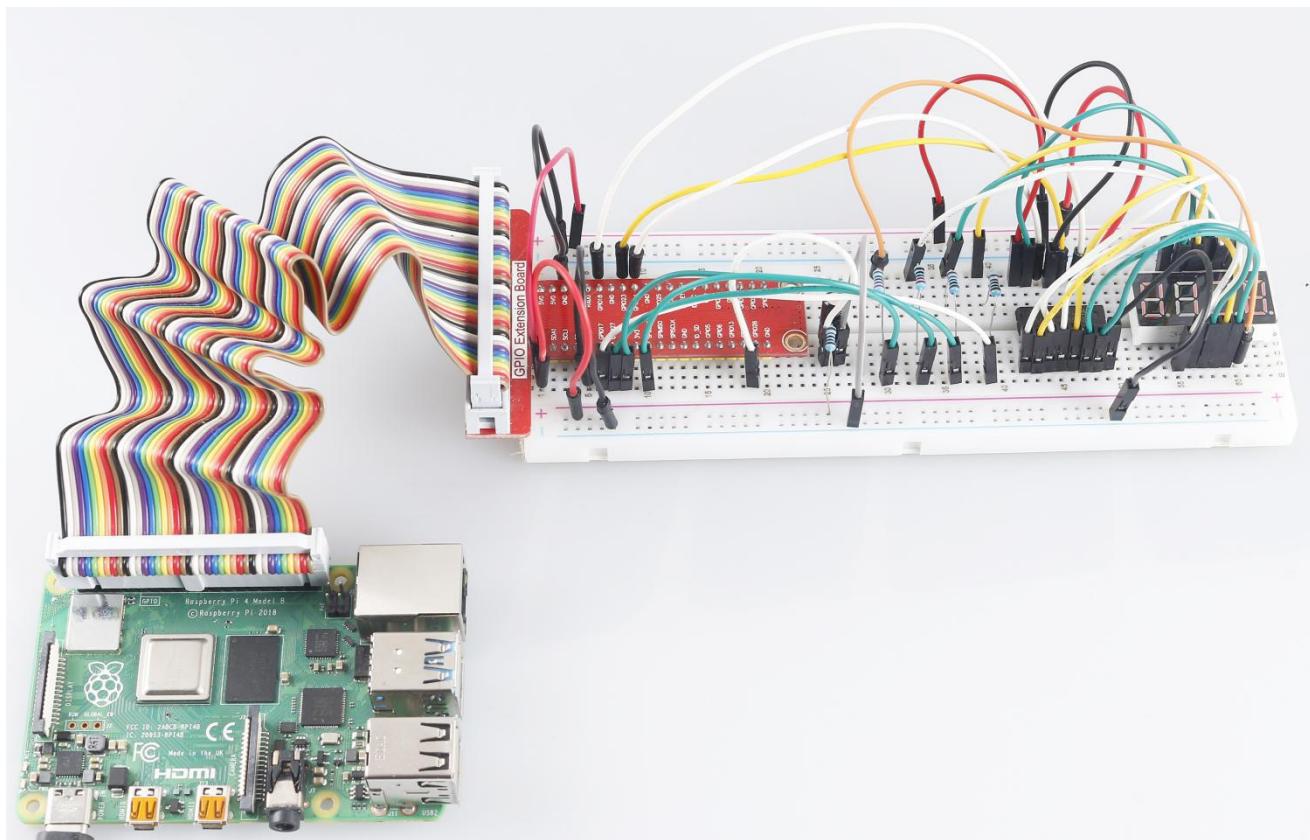
`Loop()` が主な関数である。最初に、4 ビットセグメントディスプレイに時間が表示され、傾斜スイッチの値が読み取られる。傾斜スイッチの状態が変更された場合、`stateChange()` が呼び出される。

```
def timer():
    global counter
    global timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
```

```
counter += 1
```

間隔が 0.01 秒に達すると、Timer 関数が呼び出される。カウンターに 1 を追加すると、タイマーが再び使用されて、0.01 秒ごとに繰り返し実行される。

現象画像



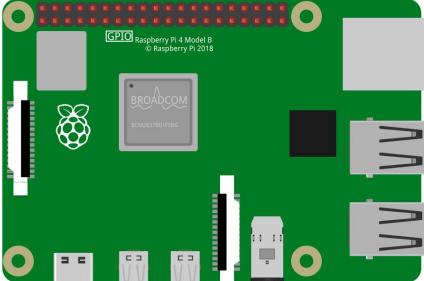
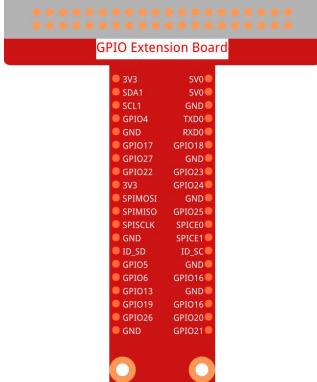
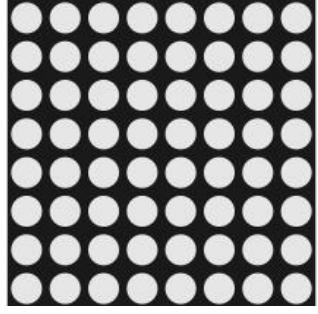
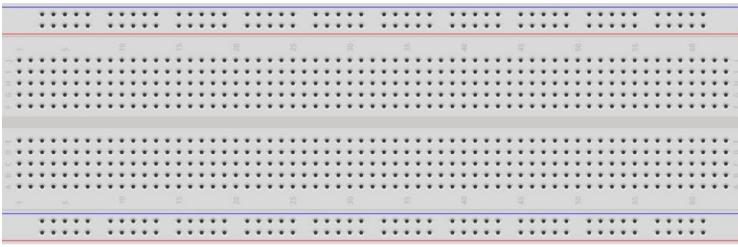
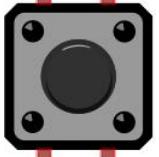
3.1.14 GAME- Not Not

前書き

このレッスンでは、面白いゲームデバイスを作成し、「Not Not」と呼ぶ。ゲーム中、ドットマトリックスは矢印をランダムに更新する。必要なのは、限られた時間内に矢印の反対方向にボタンを押すことである。時間になった場合、または矢印と同じ方向のボタンが押された場合、敗北すると意味する。

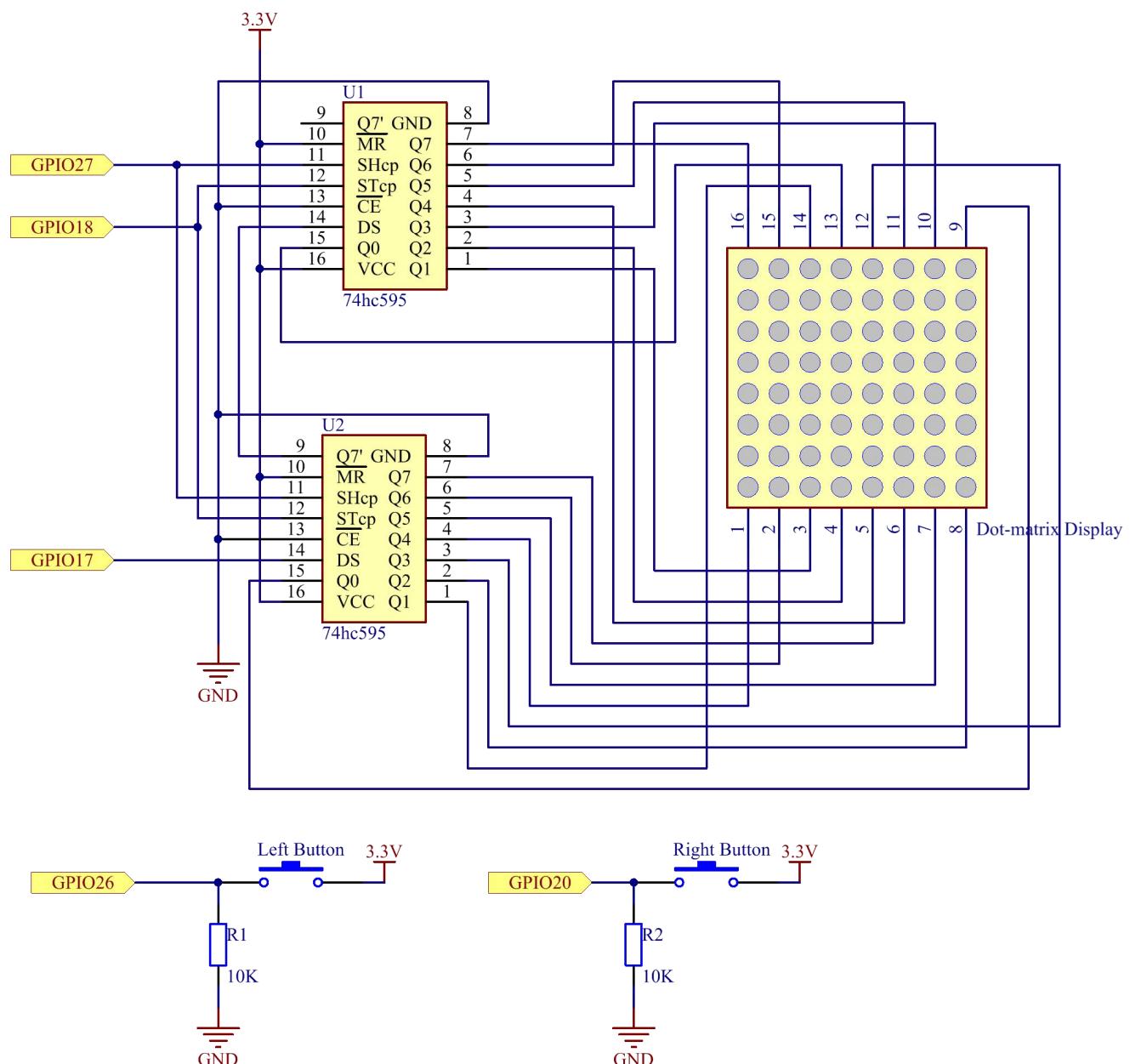
このゲームは本当にあなたの逆思考を実践することができる。

部品

Raspberry Pi 本体*1	T 拡張ボード*1	ドットマトリックス *1
	 GPIO Extension Board Pinout: <ul style="list-style-type: none">3V3SDA1SCL1GPIO04GNDGPIO17GPIO27GPIO223V3SPIMOSISPIMISOSPISCLKGNDID_SDGPIO05GPIO06GPIO13GPIO19GPIO26GND5V05V0GNDTXD0RXD0GPIO18GNDGPIO23GPIO24GNDGPIO25SPICE0SPICE1GNDGPIO16GNDGPIO16GPIO20GPIO21	
40 ピンケーブル*1	何本のジャンパー線	
		
ブレッドボード*1	ボタン*2	
		
74HC595*2		
抵抗器 (10KΩ) *2		

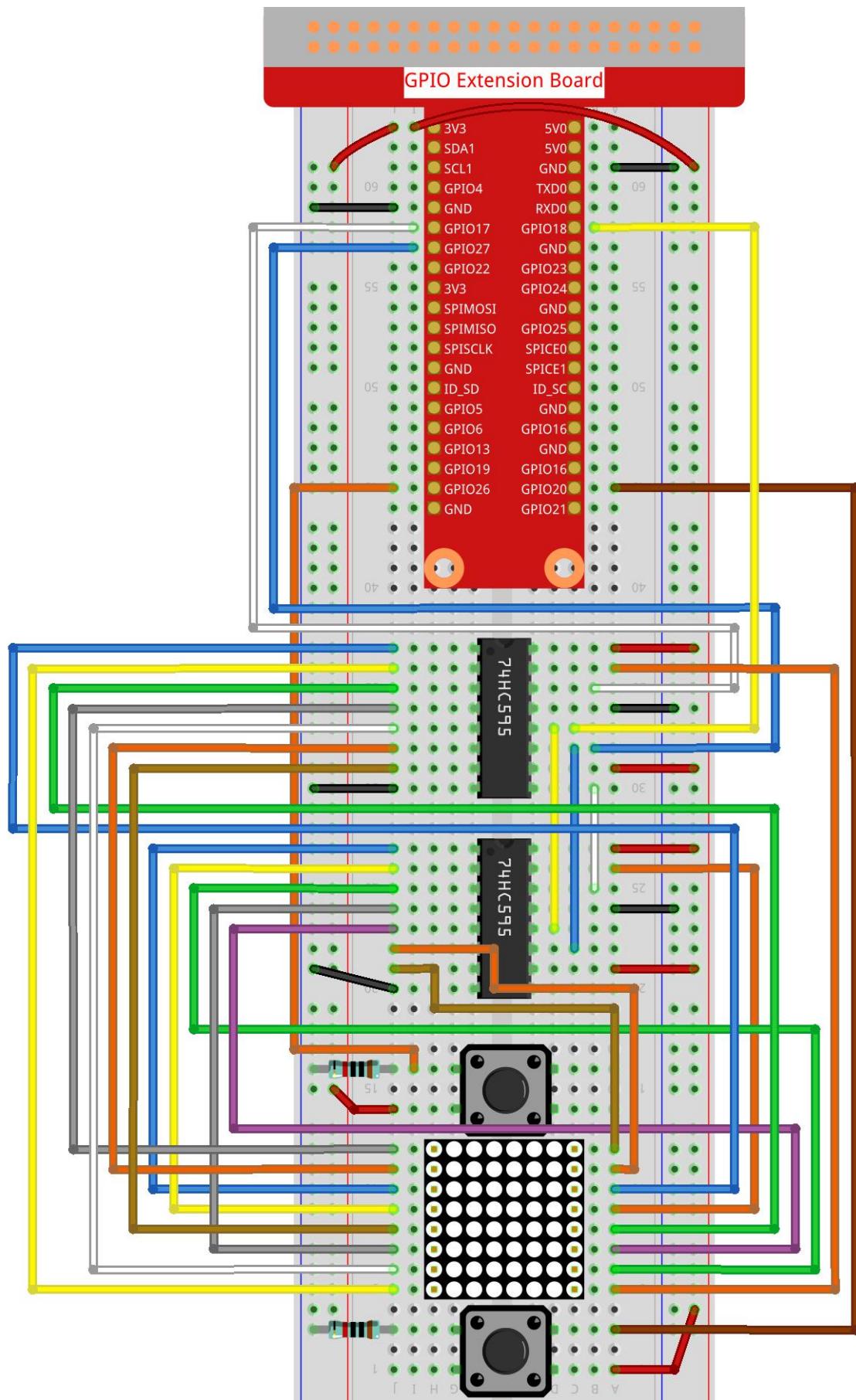
回路図

T ボード名	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO20	Pin 38	28	20
GPIO26	Pin 37	25	26



実験手順

ステップ 1: 回路を作る。



➤ C 言語ユーザー向け

ステップ 5: コードのフォルダーに移動する。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.14/
```

ステップ 6: コンパイルする。

```
gcc 3.1.14_GAME_NotNot.c -lwiringPi
```

ステップ 7: 実行する。

```
sudo ./a.out
```

プログラムの開始後、ドットマトリックス上の左矢印または右矢印がランダムに更新される。必要なのは、限られた時間内に矢印の反対方向にボタンを押すことである。次に、ドットマトリックスに「√」が表示される。時間になった場合、または矢印と同じ方向のボタンが押された場合、敗北すると意味し、ドットマトリックスに「x」が表示される。また、2つの新しいボタンを追加したり、上下左右(4つの方向でゲームの難易度を上げる)のジョイスティックキーに置き換えたりすることもできる。

コードの説明

1.1.6 LED Dot Matrix、このレッスンでは2つのボタンを追加して、面白いゲームデバイスを作成する。したがって、あまりドットマトリックスに詳しくない場合は、1.1.6 LED Dot Matrix を参照してください。

プログラム全体のプロセスは次のとおりである：

1. 矢印の方向をランダムに選択し、timer1 を生成する。
2. ドットマトリックスに矢印画像を表示する。
3. ボタン入力を判断する。ボタンが押されるか、timer1 が時間切れを想起させると、判定が開始される。
4. 判定結果に基づいて画像を表示する。その間、timer2 を生成する。
5. timer2 が時間切れを想起させると、ステップ 1 を再実行する。

```
struct GLYPH{  
    char *word;  
    unsigned char code[8];  
};  
  
struct GLYPH arrow[2]=  
{  
    {"right",{0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF}},
```

```
// {"down", {0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF}},
// {"up", {0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF}},
// {"left", {0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF}}
};

struct GLYPH check[2]=
{
    {"wrong", {0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF}},
    {"right", {0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF}}
};
```

GLYPH 構造はディクショナリのように機能する： word 属性はディクショナリのキーに対応し、 code 属性は値に対応する。

ここでは、 code を使用して、画像を表示するためのドットマトリックスの配列（8x8 ビット配列）を保存する。

ここで、配列 arrow を使用して、矢印パターンを上下左右に LED ドットマトリックスに表示することができる。上下が付注され、必要に応じて解除できる。

配列 check を使用して、これらの 2 つの画像「×」と「√」を表示する。

```
char *lookup(char *key,struct GLYPH *glyph,int length){
    for (int i=0;i<length;i++)
    {
        if(strcmp(glyph[i].word,key)==0){
            return glyph[i].code;
        }
    }
}
```

関数 lookup() は、辞書にチェックインによって機能する。 key を定義し、構造 GLYPH *glyph の key と同じ word を検索し、対応する情報（特定の word の code ）を返す。

関数 Strcmp () は、 2 つの文字列 glyph[i].word と key の同一性を比較するために使用される。同一性が判断された場合、 glyph[i].code （示すように）を返す。

```
void display(char *glyphCode){
    for(int i;i<8;i++){
        hc595_in(glyphCode[i]);
        hc595_in(0x80>>i);
        hc595_out();
    }
}
```

指定したパターンをドットマトリックスに表示する。

```
void createGlyph(){
    srand(time(NULL));
    int i=rand()%(sizeof(arrow)/sizeof(arrow[0]));
    waypoint=arrow[i].word;
    stage="PLAY";
    alarm(2);
}
```

関数 createGlyph() を使用して、方向（配列 arrow [] の要素の単語属性：“ left”、“ right” ...）をランダムに選択する。ステージを「PLAY」に設定し、2秒の目覚まし時計機能を開始する。

`srand(time(NULL))`: システムクロックからのランダムシードを初期化する。

`(sizeof(arrow)/sizeof(arrow[0]))`: 配列の長さを取得し、結果は 2 である。

`rand()%2`: 余りは 0 または 1 で、生成された乱数を 2 で割ったものである。

`waypoint=arrow[i].word`: 結果は「right」または「left」でなければならない。

```
void checkPoint(char *inputKey){
    alarm(0)==0;
    if(inputKey==waypoint||inputKey=="empty")
    {
        waypoint="wrong";
    }
    else{
        waypoint="right";
    }
    stage="CHECK";
    alarm(1);
}
```

`checkPoint()` はボタン入力をチェックするために使用される。ボタンが押されていない場合、または矢印と同じ方向のボタンが押されている場合、ウェイポイントの結果は `wrong` であり、「x」がドットマトリックスに表示される。それ以外の場合、ウェイポイントの結果は `right` であり、ドットマトリックスには「✓」が表示される。ここでは、`stage` が `CHECK` であり、1秒の目覚まし時計機能を設定することはできる。

`alarm()` は「アラームクロック」とも呼ばれ、ここでタイマーを設定でき、SIGALRM 信号を送信し、定義された時間になったら、進行状況を通知する。

```
void getKey(){
    if (digitalRead(AButtonPin)==1&&digitalRead(BButtonPin)==0)
        {checkPoint("right");}
    else if (digitalRead(AButtonPin)==0&&digitalRead(BButtonPin)==1)
        {checkPoint("left");}
}
```

getKey() はこれら 2 つのボタンの状態を読み取り、右のボタンが押された場合、checkPoint() のパラメーターは right であり、左ボタンが押された場合、パラメーターは left である。

```
void timer(){
    if (stage=="PLAY"){
        checkPoint("empty");
    }
    else if(stage=="CHECK"){
        createGlyph();
    }
}
```

以前は、alarm() タイムアップに設定するときに、timer() が呼び出された。それから 「PLAY」 モードで、checkPoint() を呼び出して結果を判断する。プログラムが 「CHECK」 モードに設定されている場合、関数 createGlyph() を呼び出して新しいパターンを選択しなければならない。

```
void main(){
    setup();
    signal(SIGALRM,timer);
    createGlyph();
    char *code = NULL;
    while(1){
        if (stage == "PLAY")
        {
            code=lookup(waypoint,arrow,sizeof(arrow)/sizeof(arrow[0]));
            display(code);
            getKey();
        }
        else if(stage == "CHECK")
        {
            code = lookup(waypoint,check,sizeof(check)/sizeof(check[0]));
            display(code);
        }
    }
}
```

```
}
```

関数 signal(SIGALRM、 timer)の動作：（目覚まし時計関数 alarm()によって生成された） SIGALRM 信号を受信したときに関数 timer()を呼び出す。

プログラムが開始したら、まず createGlyph()を 1 回呼び出してから、loop を開始する。

「PLAY」モードでは、ドットマトリックスに矢印パターンが表示され、ボタンの状態が確認される。 「CHECK」モードの場合、表示されるのは「x」または「√」である。

➤ Python 言語ユーザー向け

ステップ 5: コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/python
```

ステップ 6: 実行する。

```
sudo python3 3.1.14_GAME_NonNot.py
```

プログラムを開始すると、ドットマトリックスに右または左を指す矢印が表示される。必要なのは、限られた時間内に矢印の反対方向にボタンを押すことである。次に、ドットマトリックスに「√」が表示される。時間になった場合、または矢印と同じ方向のボタンが押された場合、敗北すると意味し、ドットマトリックスに「x」が表示される。また、2つの新しいボタンを追加したり、上下左右(4 つの方向でゲームの難易度を上げる)のジョイスティックキーに置き換えることもできる。

コードの説明

1.1.6 LED Dot Matrix、このレッスンでは 2 つのボタンを追加して、面白いゲームデバイスを作成する。したがって、あまりドットマトリックスに詳しくない場合は、1.1.6 LED Dot Matrix を参照してください。

プログラム全体のプロセスは次のとおりである：

1. 矢印の方向をランダムに選択し、timer1 を生成する。
2. ドットマトリックスに矢印画像を表示する。
3. ボタン入力を判断する。ボタンが押されるか、timer1 が時間切れを想起させると、判定が開始される。
4. 判定結果に基づいて画像を表示する。その間、timer2 を生成する。
5. timer2 が時間切れを想起させると、ステップ 1 を再実行する。

```
def main():
    creatGlyph()
```

```

while True:
    if stage == "PLAY":
        display(arrow[waypoint])
        getKey()
    elif stage == "CHECK":
        display(check[waypoint])
    
```

main()には実行中のプロセス全体が含まれている。

プログラムが開始したら、まず createGlyph()を 1 回呼び出してから、loop を開始する。

「PLAY」モードでは、ドットマトリックスに矢印パターンが表示され、ボタンの状態が確認される。「CHECK」モードの場合、表示されるのは「x」または「√」である。

```

arrow={
    "#down": [0xFF, 0xEF, 0xC7, 0xAB, 0xEF, 0xEF, 0xEF, 0xFF],
    "#up": [0xFF, 0xEF, 0xEF, 0xEF, 0xAB, 0xC7, 0xEF, 0xFF],
    "right": [0xFF, 0xEF, 0xDF, 0x81, 0xDF, 0xEF, 0xFF, 0xFF],
    "left": [0xFF, 0xF7, 0xFB, 0x81, 0xFB, 0xF7, 0xFF, 0xFF]
}
check={
    "wrong": [0xFF, 0xBB, 0xD7, 0xEF, 0xD7, 0xBB, 0xFF, 0xFF],
    "right": [0xFF, 0xFF, 0xF7, 0xEB, 0xDF, 0xBF, 0xFF, 0xFF]
}
    
```

ここでは、ディクショナリの矢印を使用して、LED ドットマトリックス上に矢印パターンを上下左右に表示できる。

上下が付注され、必要に応じて解除できる。

ディクショナリチェックを使用して、これらの 2 つの画像「×」と「√」を表示する。

```

def display(glyphCode):
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80 >> i)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)
    
```

指定したパターンをドットマトリックスに表示する。

```

def creatGlyph():
    global waypoint
    global stage
    global timerPlay
    
```

```
waypoint=random.choice(list(arrow.keys()))
stage = "PLAY"
timerPlay = threading.Timer(2.0, timeOut)
timerPlay.start()
```

関数 `createGlyph()` を使用して、方向（配列 `arrow []` の要素の単語属性：“left”、“right”...）をランダムに選択する。ステージを「PLAY」に設定し、2秒の目覚まし時計機能を開始する。

`arrow.keys()`: 矢印配列のキー「right」と「left」を選択する。

`list(arrow.keys())`: これらのキーを配列に結合する。

`random.choice(list(arrow.keys()))`: 配列内の要素をランダムに選択する。

したがって、「`waypoint=random.choice(list(arrow.keys()))`」の結果は「右」または「左」になります。

```
def checkPoint(inputKey):
    global waypoint
    global stage
    global timerCheck
    if inputKey == "empty" or inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()
```

`checkPoint ()` はボタン入力の現在の状態を検出する：

ボタンが押されていない場合、または矢印と同じ方向のボタンが押されている場合、`waypoint` に割り当てられた値は `wrong` であり、ドットマトリックスに「x」が表示される。

それ以外の場合、`waypoint` は `right` であり、「✓」が表示される。

これで `stage` は `CHECK` になり、1秒のタイマー `timerCheck` を開始して関数 `creatGlyph()` を1秒以内呼び出す。

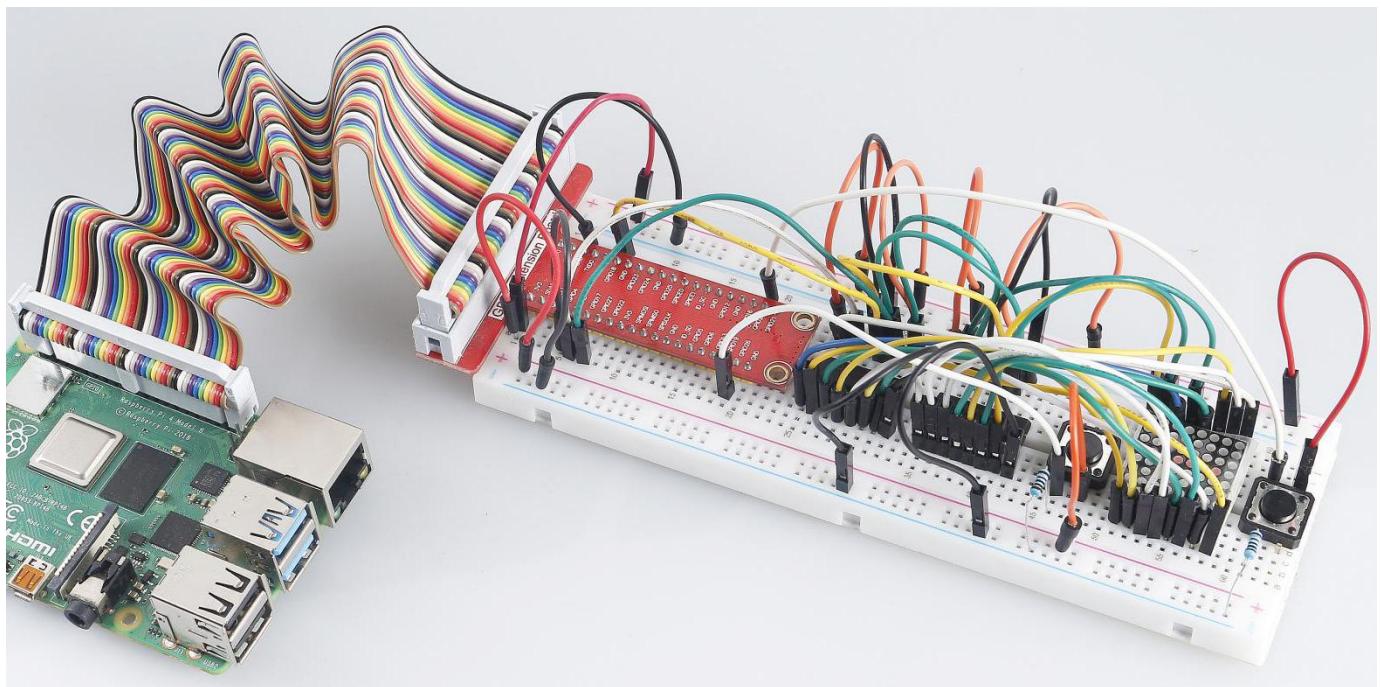
```
def timeOut():
    checkPoint("empty")
```

関数 `timeOut()` で、`checkPoint()` のパラメーターを「empty」に設定する。

```
def getKey():
    if GPIO.input(AButtonPin)==1 and GPIO.input(BButtonPin)==0:
        checkPoint("right")
    elif GPIO.input(AButtonPin)==0 and GPIO.input(BButtonPin)==1:
        checkPoint("left")
```

getKey() はこれら 2 つのボタンの状態を読み取り、右のボタンが押された場合、checkPoint() のパラメーターは「right」であり、左ボタンが押された場合、パラメーターは「left」である。

現象画像



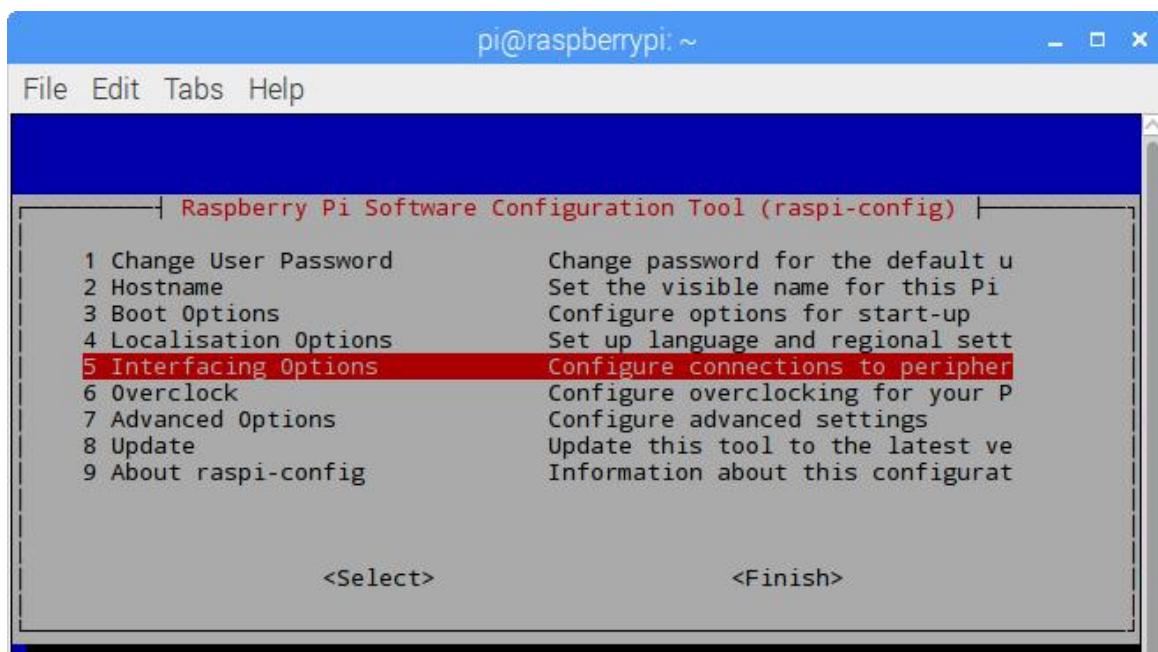
3.2 付録

3.2.1 I2C 設定

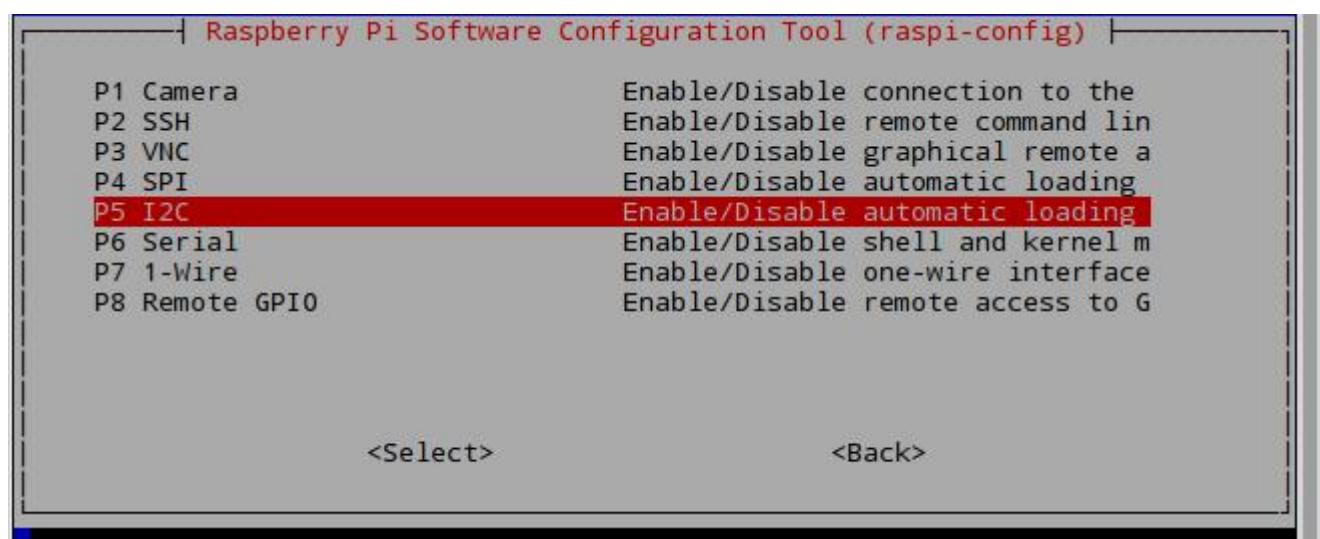
ステップ 1: raspberry Pi の I2C ポートを作動させる（作動した場合は、これをスキップしてください。これを行ったかどうかわからない場合は、続行してください）。

```
sudo raspi-config
```

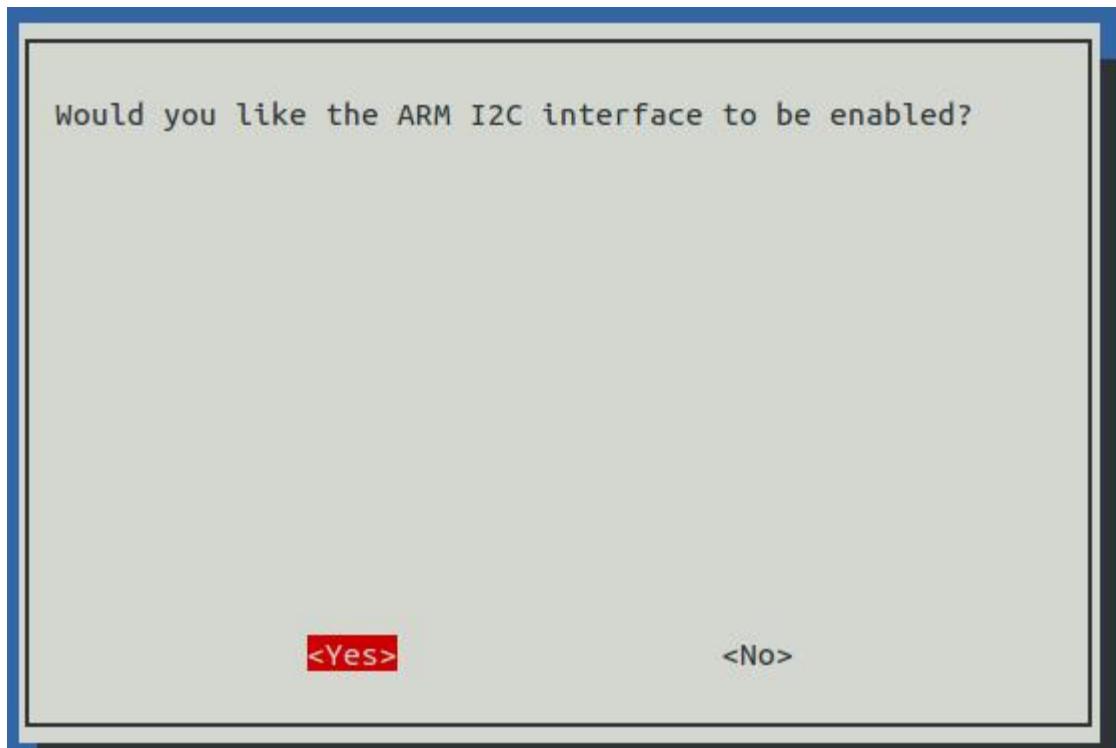
5 Interfacing options



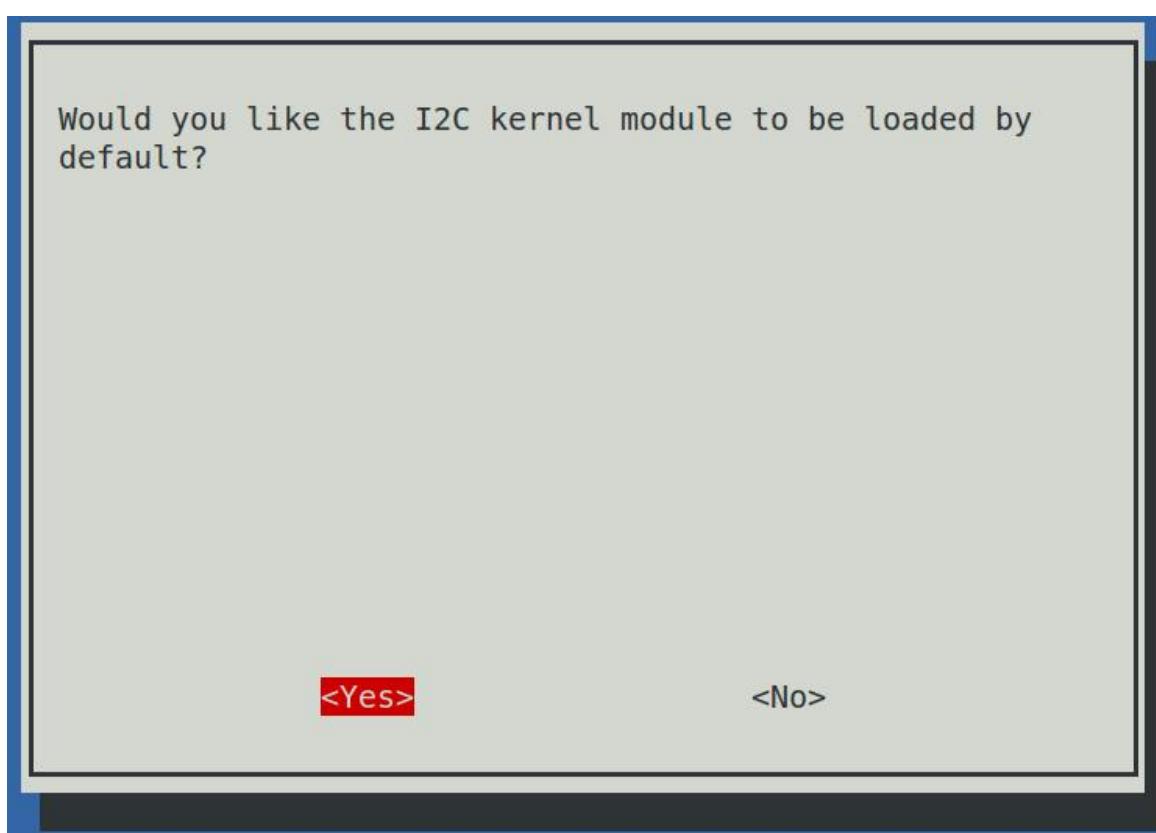
P5 I2C



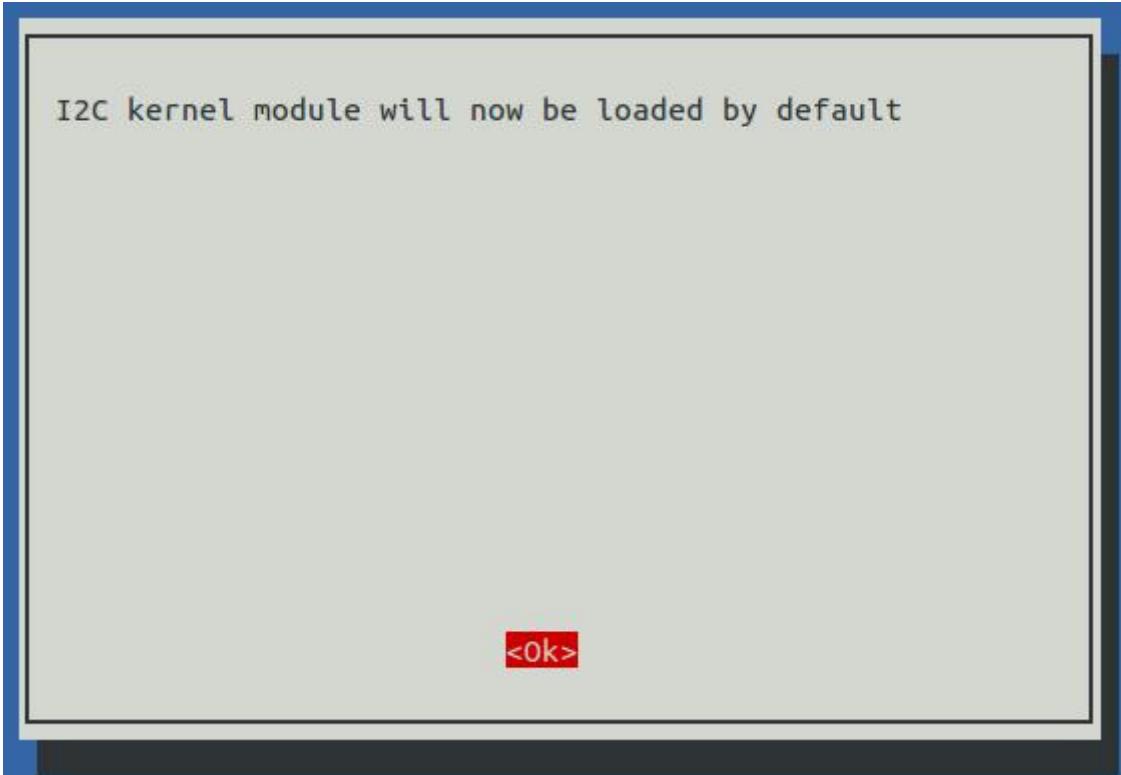
<Yes>



<Yes>



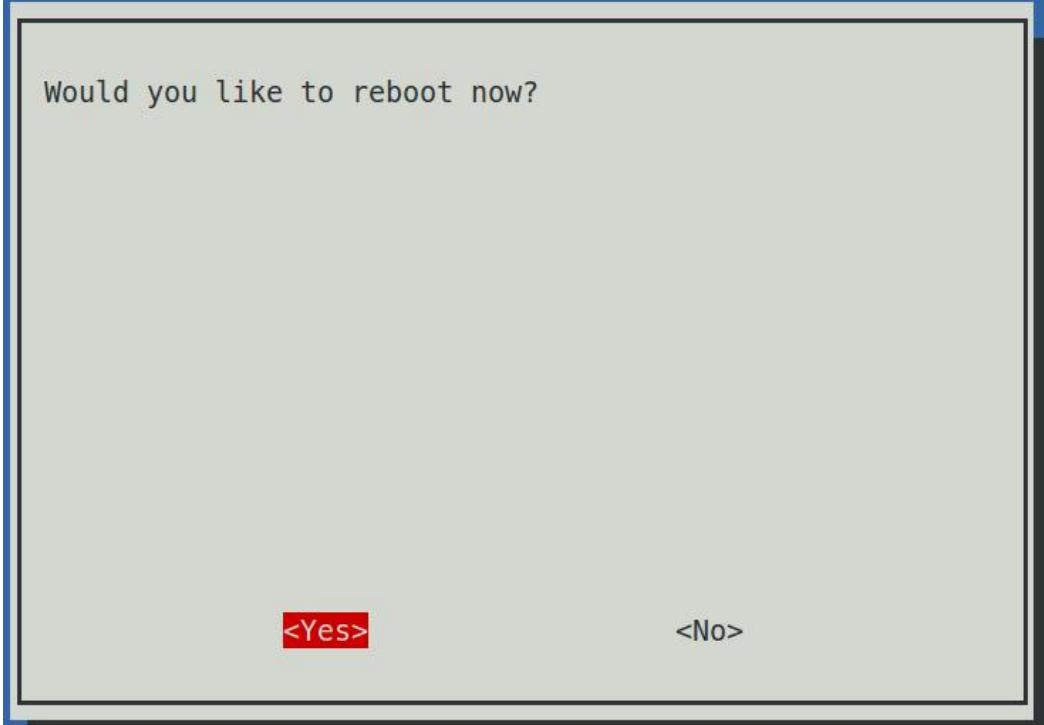
<Ok>



<Finish>



<Yes> (If you do not see this page, continue to the next step)



ステップ 2: I2C モジュールがロードされ作動しているかどうかを確認する。

```
lsmod | grep i2c
```

その後、次のコードが表示される（数は異なる場合がある）。

```
i2c_dev          6276    0
i2c_bcm2708      4121    0
```

ステップ 3: i2c-tools を実装する。

```
sudo apt-get install i2c-tools
```

ステップ 4: I2C デバイスのアドレスを確認する。

```
i2cdetect -y 1      # For Raspberry Pi 2 and higher version
i2cdetect -y 0      # For Raspberry Pi 1
pi@raspberrypi ~ $ i2cdetect -y 1
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -----
10: -----
20: -----
30: -----
40: ----- 48 -----
50: -----
60: -----
70: -----
```

I2C 装置が接続されている場合、結果は上記のようになる-デバイスのアドレスは 0x48 であるため、48 がプリントされる。

ステップ 5:

C 言語ユーザーの場合: libi2c-dev を実装する。

```
sudo apt-get install libi2c-dev
```

Python ユーザーの場合: I2C 用の smbus を実装する。

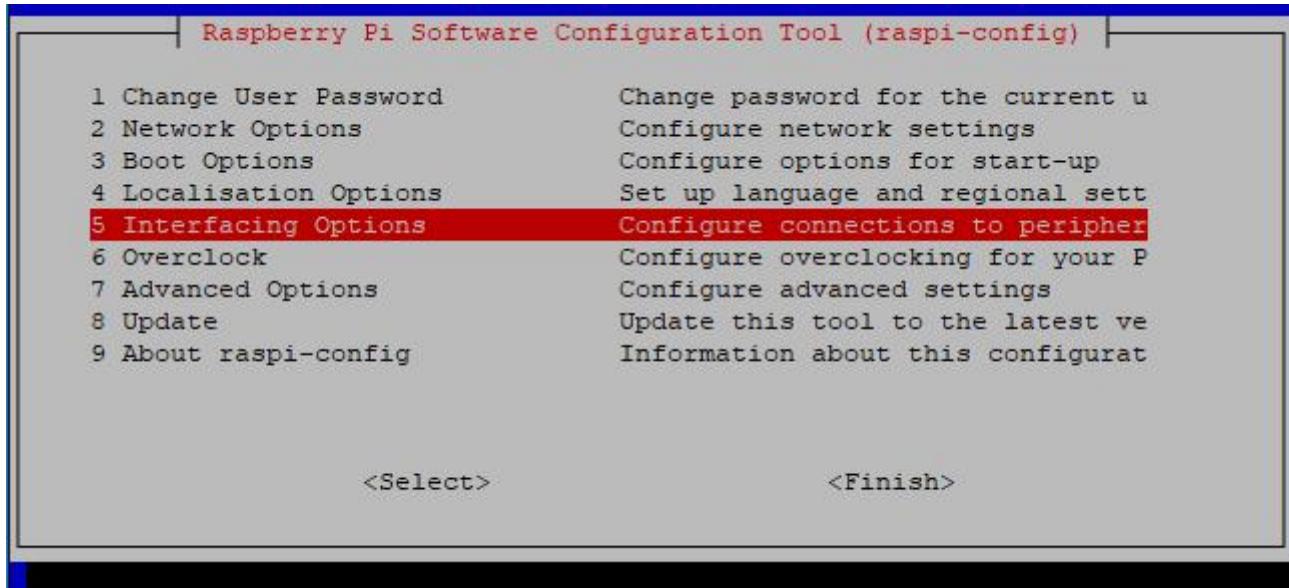
```
sudo apt-get install python-smbus
```

3.2.2 SPI 設定

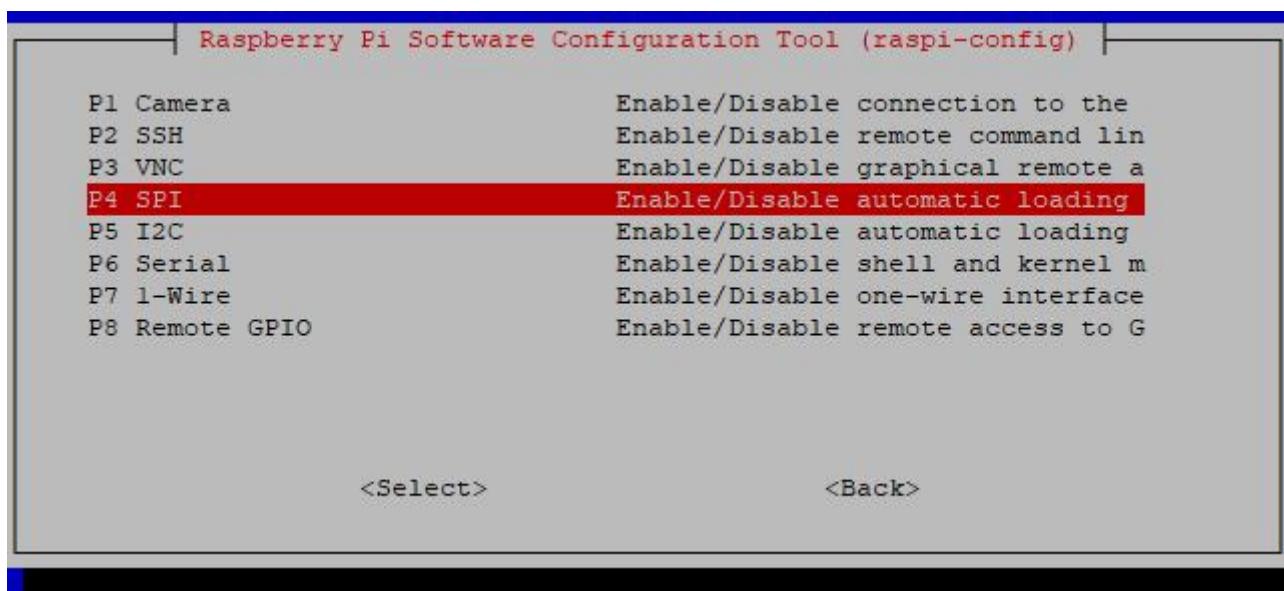
ステップ1: Raspberry Pi の SPI ポートを作動させる（作動した場合は、これをスキップしてください。これを行ったかどうかわからない場合は、続行してください）。

```
sudo raspi-config
```

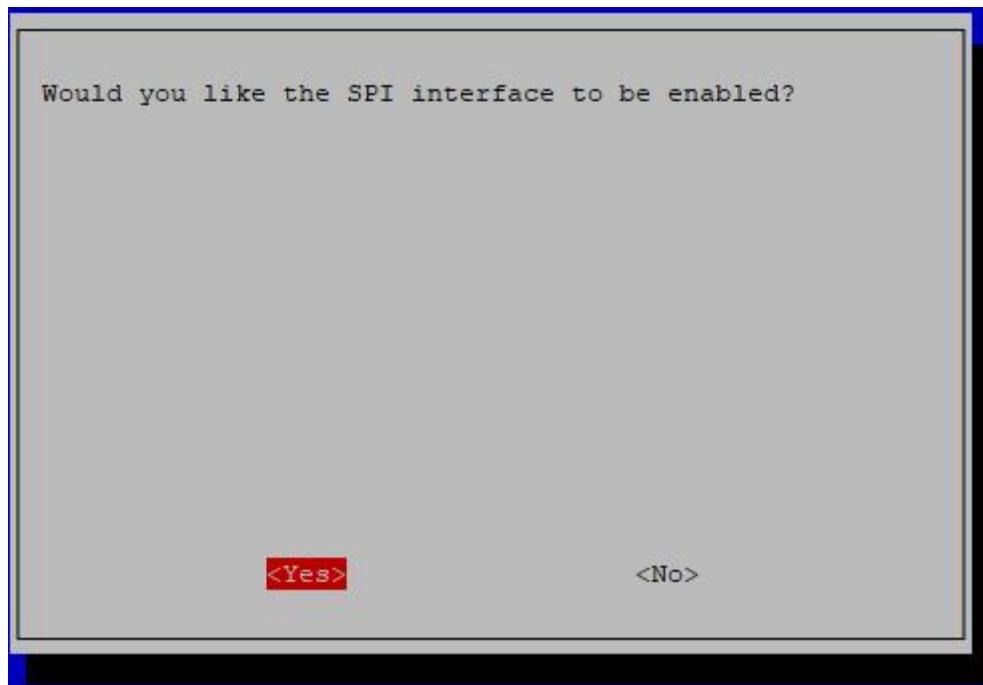
5 Interfacing options



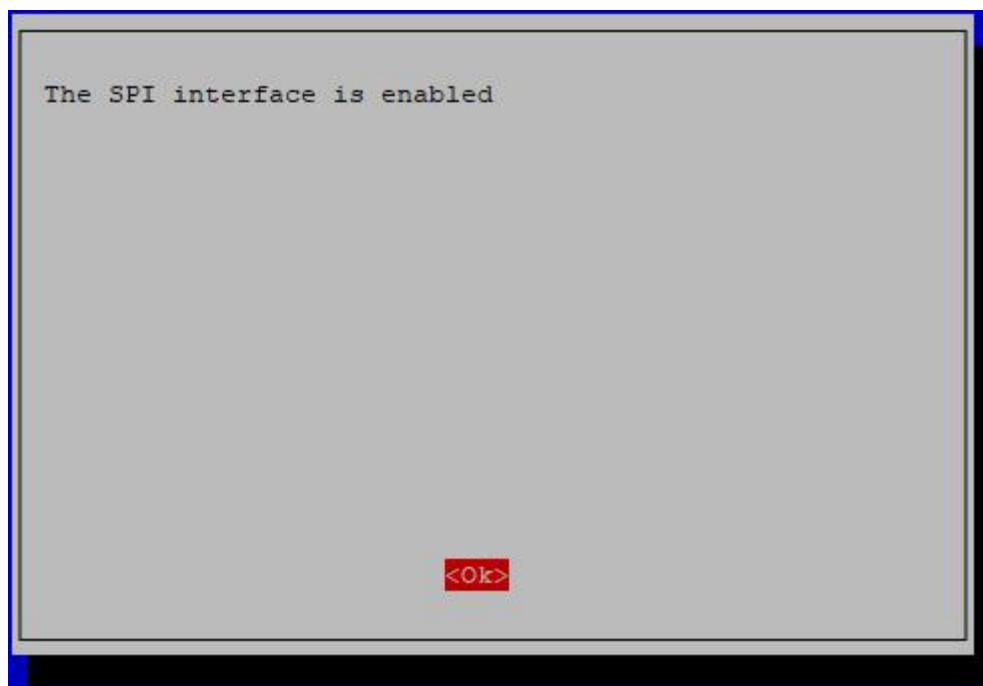
P4 SPI



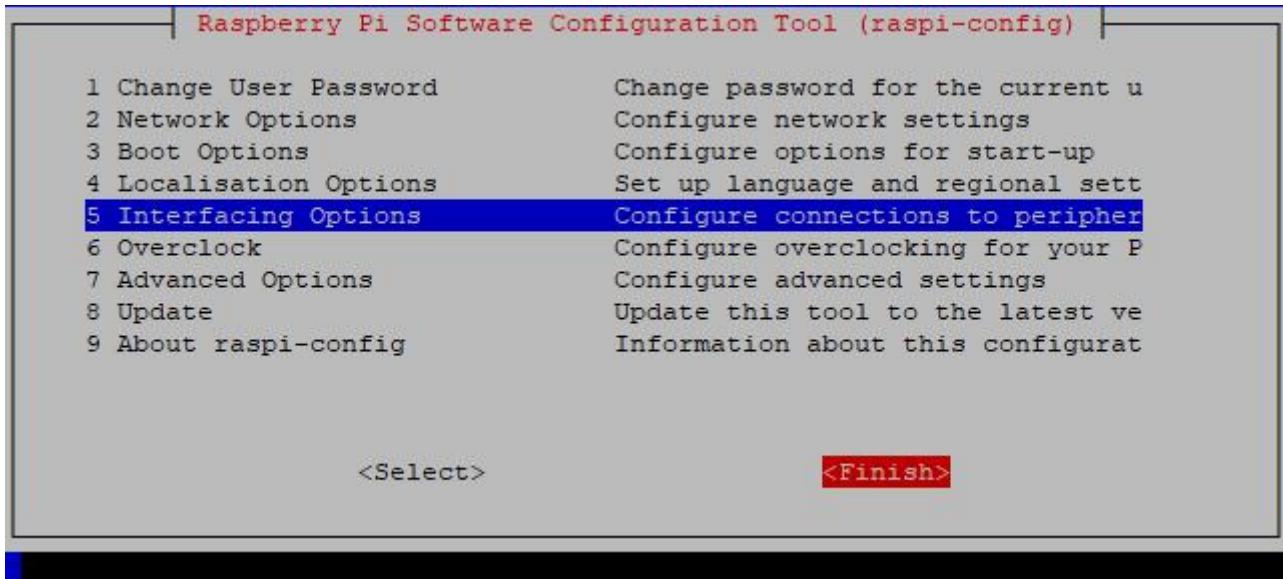
<YES>



<OK>



<Finish>



ステップ 2: i2c モジュールがロードされ作動しているかどうかを確認する。

```
ls /dev/sp*
```

その後、次のコードが表示される（数は異なる場合がある）。

```
/dev/spidev0.0  /dev/spidev0.1
```

ステップ 3: Python モジュール SPI-Py を実装する。

```
git clone https://github.com/lthiery/SPI-Py.git
cd SPI-Py
sudo python3 setup.py install
```

ご注意: この手順は Python ユーザー向けで、C 言語を使用する場合はスキップしてください。

著作権表示

このマニュアルのテキスト、画像とコードを含むがこれらに限定されないすべての内容は、SunFounder Company が所有している。関連する規制と著作権法に基づき、著者と関連する権利所有者の法的権利を侵害することなく、個人的な研究、調査、享楽、またはその他の非営利目的でのみ使用してください。許可なく営利目的でこれを使用する個人または組織については、会社は法的措置を取る権利を留保する。