SunFounder davinci-kit-for-raspberry-pi

www.sunfounder.com

2024 年 04 月 03 日

目次

第 1 章 Da Vinci キットについて 3

1.1 部品一覧 . . . 3
1.2 前書き . . . 7
1.3 準備 . . . 8
1.4 ライブラリ . . . 38
1.5 WiringPi をインストールして確認する . . . 39
1.6 GPIO 拡張ボード . . . 40
1.7 コードをダウンロードする . . . 43
1.8 1 出力 (Pi 5 用ではない) . . . 44
1.9 2 入力 (Pi 5 用ではない) . . . 195
1.10 3 拡張 (Pi 5 用ではない) . . . 343
1.11 Python ビデオコース (Pi 5 用ではない) . . . 503
1.12 Python で遊ぼう (すべての Pi モデル用) . . . 528
1.13 Processing で遊ぶ (Pi 5 用ではない) . . . 793
1.14 Scratch で遊ぶ (Pi 5 用ではない) . . . 827
1.15 他言語（Pi 5 用） . . . 903
1.16 付録 . . . 906
1.17 FAQ . . . 913
1.18 ありがとうございました . . . 914

第 2 章 Copyright Notice 915

SunFounder Vinci Kit をお選びいただき、ありがとうございます。

注釈: このドキュメントは以下の言語で利用可能です。

-
-
-

ご希望の言語でドキュメントにアクセスするために、それぞれのリンクをクリックしてください。

第 1 章

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 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 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！

もし、私たちが持っていない別のプロジェクトを学びたい場合は、お気軽にメールをお送りください。

電子メール：cs@sunfounder.com。

1.1 部品一覧

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。

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

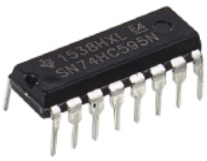
Photoresistor
1 pcs

Thermistor
1 pcs

L293D
1 pcs

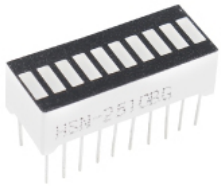
ADC0834
1 pcs

74HC595
2 pcs

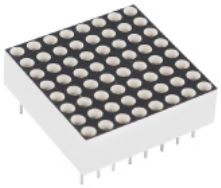


7-segment Display
1 pcs

LED Bar Graph
1 pcs

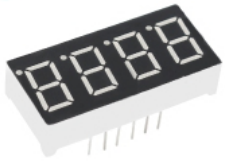


LED Matrix
1 pcs



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

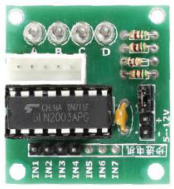
9G Servo
1 pcs

Breadboard Power Module
1 pcs

MFRC522 RFID Module
1 pcs

Relay
1 pcs

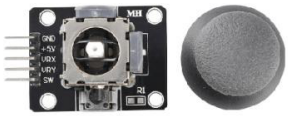
Stepping Motor Driver
1 pcs



Stepping Motor
1 pcs

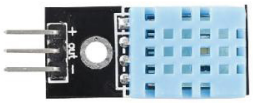
Ultrasonic Ranging Module
1 pcs

Joystick
1 pcs

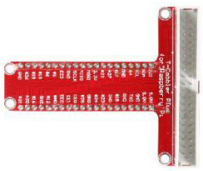


Infrare Motion Sensor
1 pcs

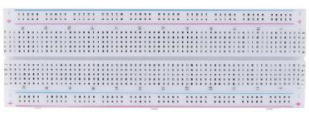
DHT-11
1 pcs



T-shape Extension Board
1 pcs



Breadboard
1 pcs



MPU6050 Module
1 pcs

1.2 前書き

DaVinci 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。

1.3 準備

この章では、まずラズベリーパイの起動方法を学びます。内容には、オペレーティングシステム、ラズベリーパイネットワークをインストールする方法とターミナルの開き方が含まれます。

注釈: 完全なチュートリアルは、ラズベリーパイの公式サイト: raspberry-pi-setting-up.

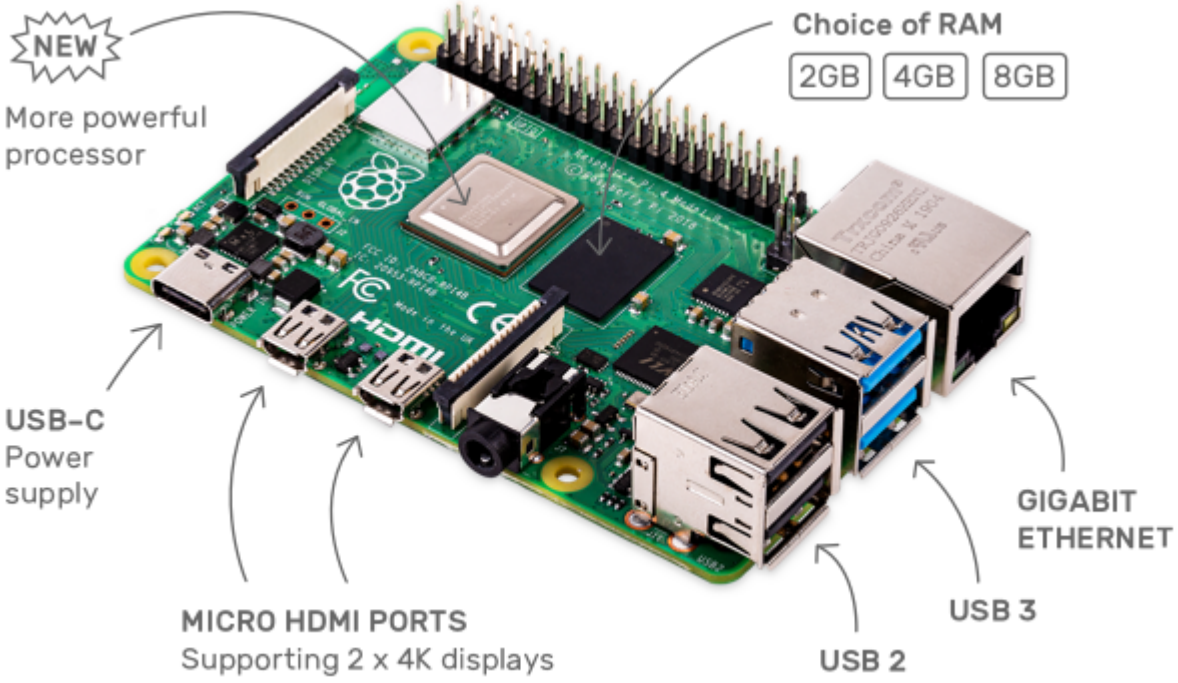
ラズベリーパイがセットアップされている場合は、この部分をスキップして次の章に進むことができます。

1.3.1 何が必要？

必要なものなど

Raspberry Pi

Raspberry Pi は低価格のクレジットカード　サイズのコンピューターで、コンピューターの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します。Raspberry Pi はあらゆる年齢の人々がコンピュータを利用し、Scratch や Python などの言語でプログラミングする方法を学ぶことを可能にする、有能な小さなデバイスです。



電源アダプター

電源コンセントに接続するために Raspberry Pi には (スマホなどと同じ) マイクロ USB ポートがあります。 電源容量は少なくとも 2.5A/h 以上の製品が必要です。

マイクロ SD カード

Raspberry Pi では Raspberry Pi OS とユーザーのデータを保存するためのマイクロ SD カードが必要です。 通常のパソコンの SSD に当たります。最低でも 8GB 以上の容量のマイクロ SD カードが必要です。

その他

ディスプレイ

Raspberry Pi のデスクトップを表示するには HDMI 端子付きのテレビ画面かパソコン用のディスプレイが必要になります。 画面側にスピーカーが内蔵されている場合は音もそこから再生する事ができます。

マウスとキーボード

Raspberry Pi 本体の操作のためには USB キーボードと USB マウスも必要です。なお USB アダプタ付きの無線マウスとキーボードも利用可能です。

HDMI ケーブル

Raspberry Pi には最新のテレビやディスプレイ用の HDMI 出力ポートがあります。モデルによって使えるケーブルが異なるので確認してください。 またモニターが古く DVI または VGA ポートしかない場合は適切な変換器を使用する必要があります。

ケース

Raspberry Pi は専用ケースに入れることができます。 これによりデバイスを保護できますが、利用するロボットキットなどにはケースが付属していたりしますので必要かどうか確認してください。

イヤフォン出力

Raspberry Pi には 3.5mm のイヤフォン。ジャックが装備されておりモニターにスピーカーが内蔵されていない場合にイヤフォンや PC 用外付けスピーカーなどが利用可能です。こちらもロボットハットなどにはスピーカーを内蔵しておりますのでキットの内容をご確認ください。

1.3.2 Raspberry pi OS のインストール

必要なもの

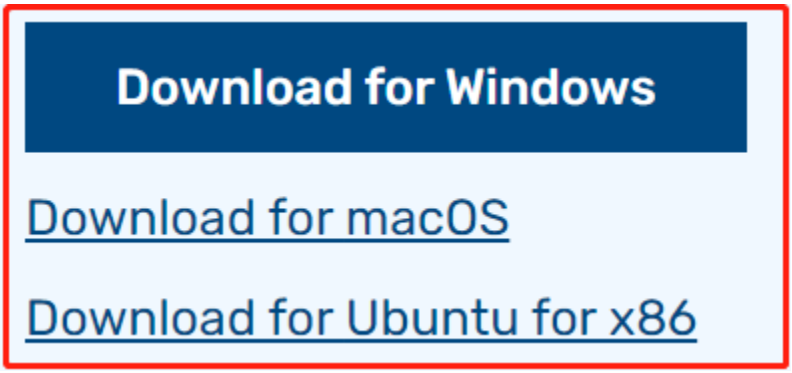
- Raspberry Pi 本体
- PC（Windows Mac Linux/UNIX）
- マイクロ SD カード 8GB 以上のもの

ステップ 1

Raspberry Pi 専用に SD カード書き込みツールが用意されました。Mac、Windows ならびに Ubuntu 18.04 で動作しますので、ほとんどの方にとって最も簡単な Raspberry pi 用の SD カードの作成方法です。

ダウンロードページにアクセスしてください: https://www.raspberrypi.org/software/

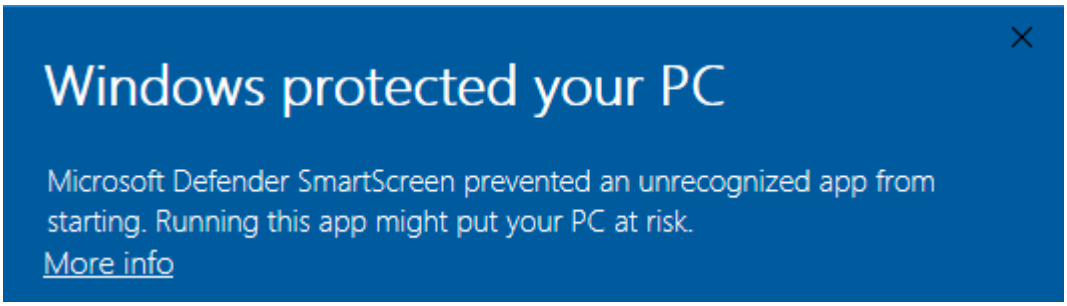
お使いのパソコン用の Raspberry Pi Imager のリンクをクリックしダウンロードし、完了したらそれをクリックしてインストーラーを起動します。



ステップ 2

インストーラーを起動するとＯＳがインストーラーの実行を禁止しようとする場合があります。 たとえば、Windows では次のメッセージが表示されます。

このポップアップが表示された場合は、[**More info**] をクリックしてから [**Run anyway**] をクリックし指示に従って Raspberry Pi Imager をインストールします。



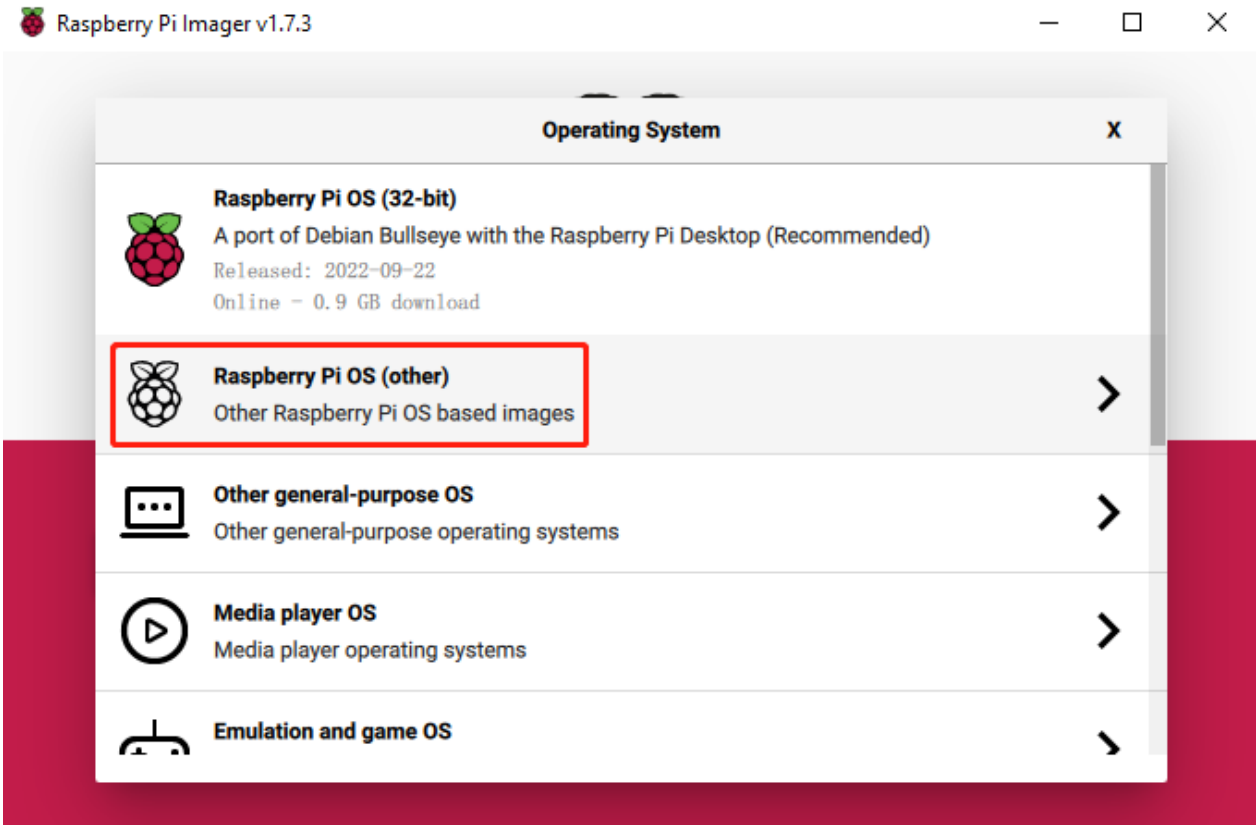
ステップ 3

SD カードをパソコンの SD カード スロットに挿入します。（SD カードスロットが無いパソコンの場合 USB マイクロ SD カードアダプターを使用してください）SD カードのフォーマットなどのメニュー画面が表示された場合、それらは無視して閉じてください。

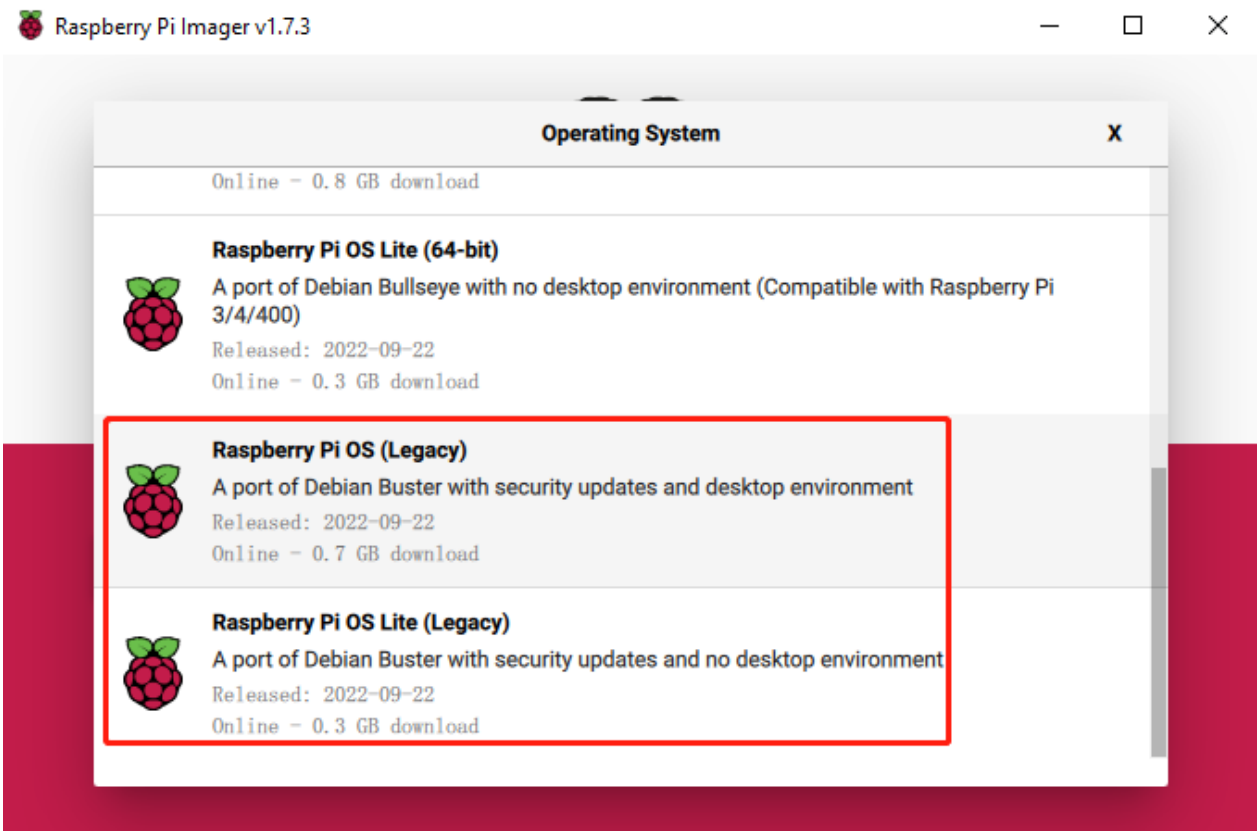
ステップ 4

> **警告:** Raspberry Pi OS を **Debian Bullseye** にアップグレードすると、一部の機能が動作しなくなるため、**Debian Buster** バージョンを引き続き使用することをお勧めします。

Raspberry Pi Imager で、 **CHOOSE OS** から **Raspberry Pi OS(other)** を選択してクリックします。

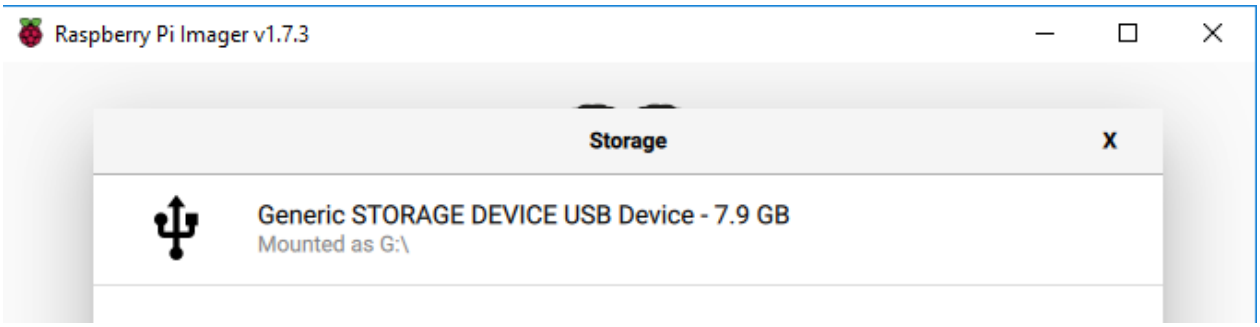


新しく開いたページの最後までスクロールすると **Raspberry Pi OS(Legacy)** と **Raspberry Pi OS Lite(Legacy)** が表示されます。これらの違いはデスクトップ（GUI 環境：普段から馴染みのあるマウスで操作できる画面）の有る／無しの違いです。**Raspberry Pi OS(Legacy)** をインストールすることをお勧めします。これにより Raspberry Pi デスクトップにアクセスしてカメラで捉えている映像をリアルタイムで見ることができるようになります。



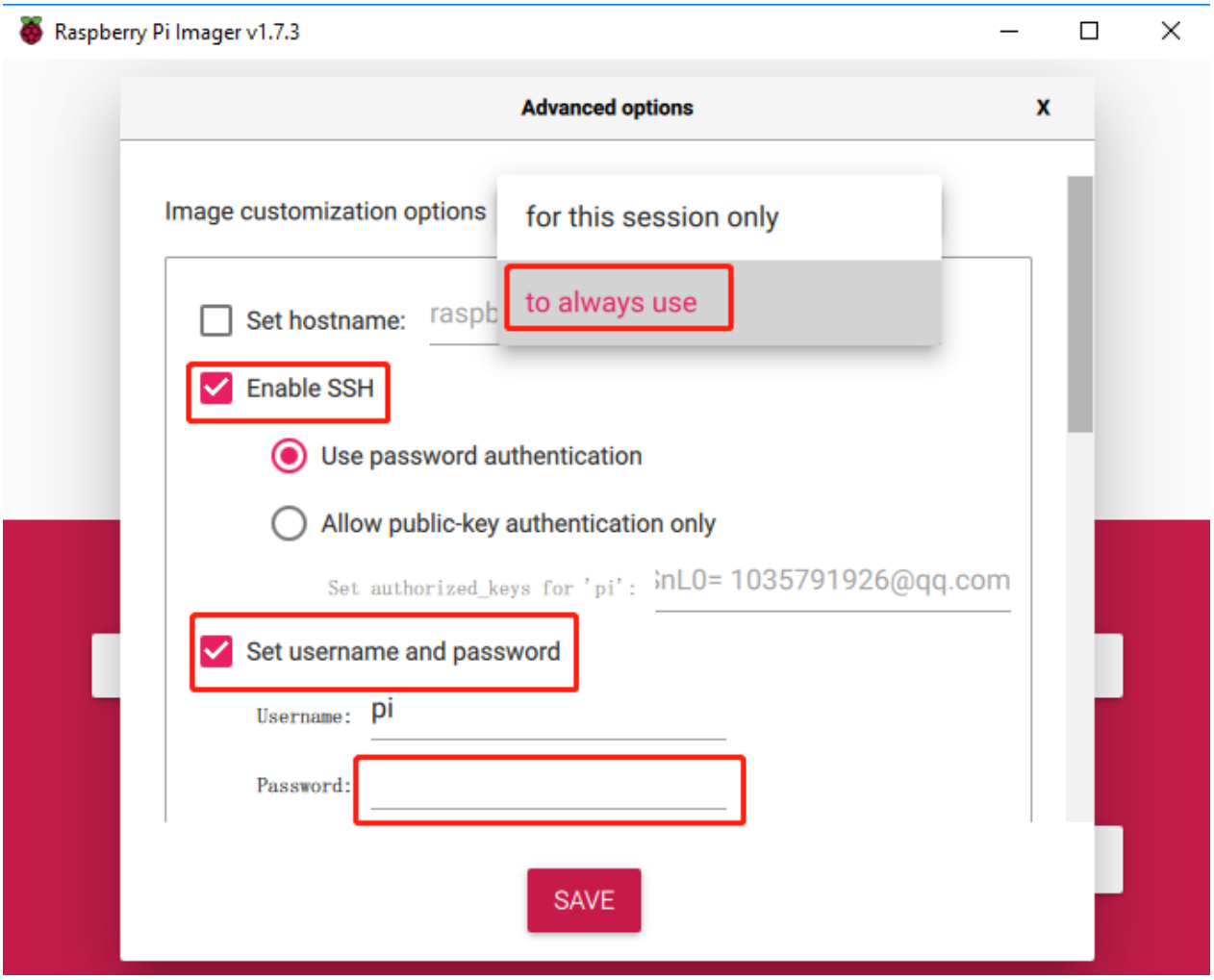
ステップ 5

使用する SD カードを選択します。



ステップ 6

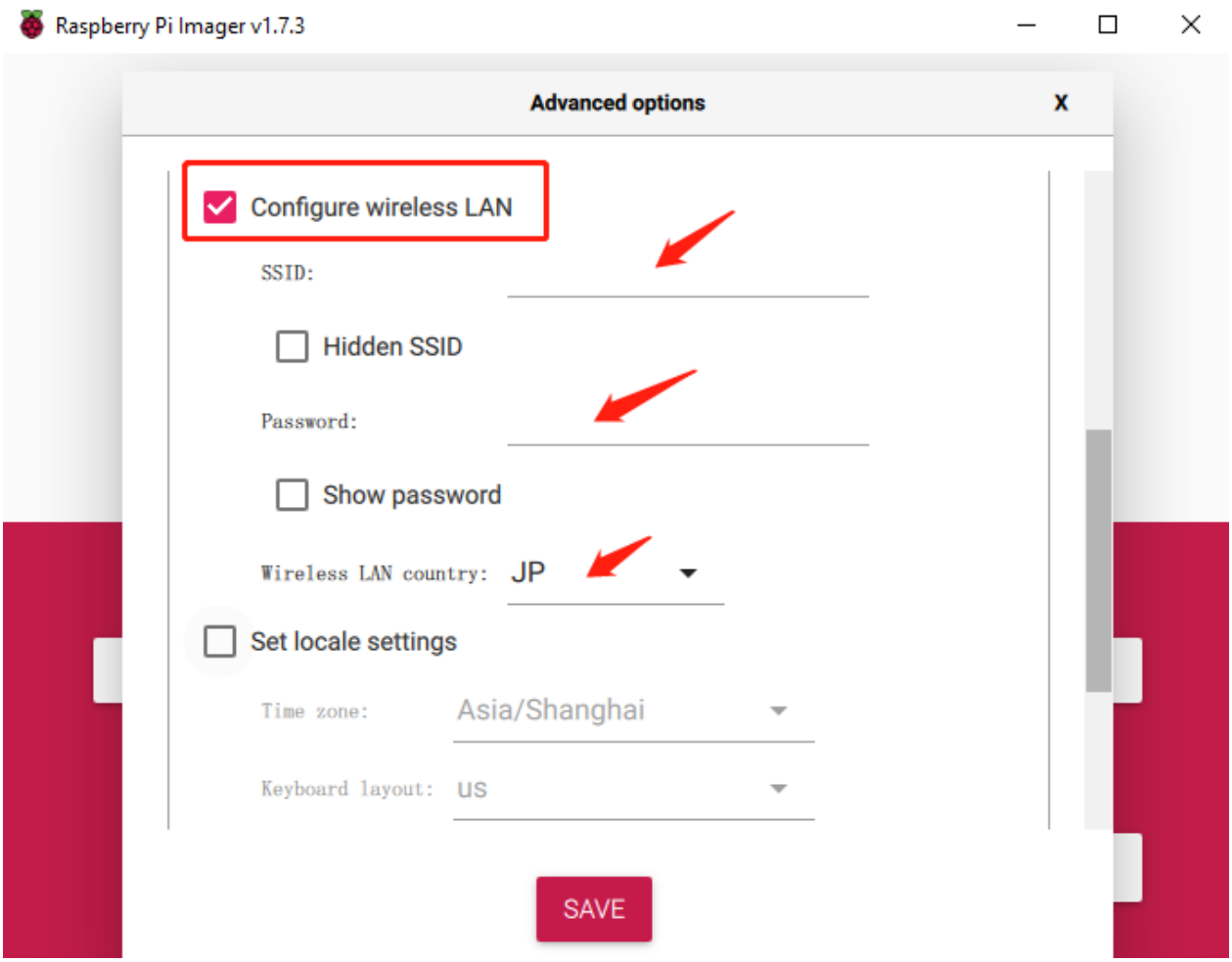
設定 ボタン（画面右下のギアマーク：OS を選択すると表示される）をクリックして **詳細オプション** ページを開き、SSH を有効にして Wi-Fi を設定します。最低これら 2 つの項目を設定する必要があり他の項目は選択に応じて異なりますがパスワードはここで設定しておくと楽です。このカスタマイズ オプションを常に使用するように選択できます（次回利用時のデフォルトになる）。



下にスクロールして Wi-Fi の設定を完了し **SAVE** をクリックします。

注釈: **wifi country** には、2 文字の ISO/IEC alpha2 コード を設定する必要があります。

日本国内で使用しているならば「JP」と設定してください。

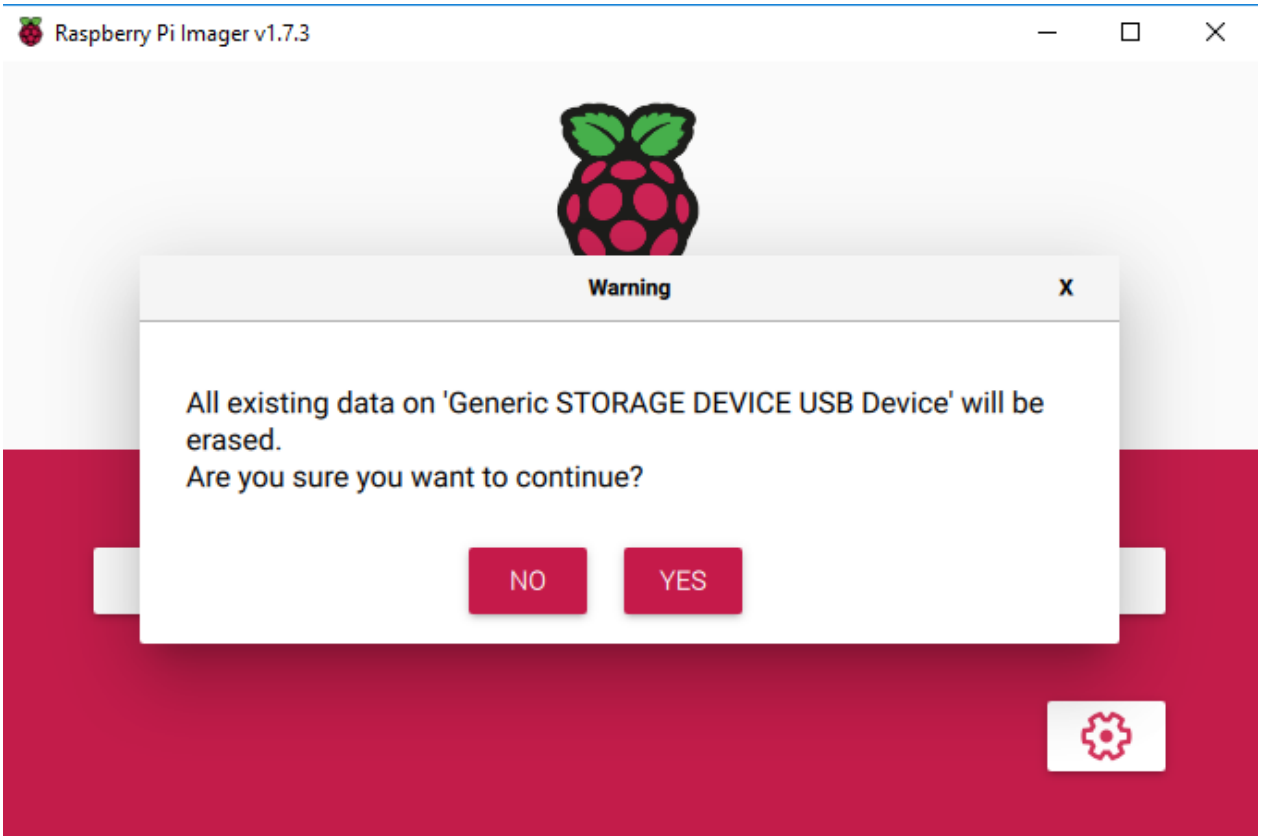


ステップ 7

書き込む ボタンをクリックしてください。

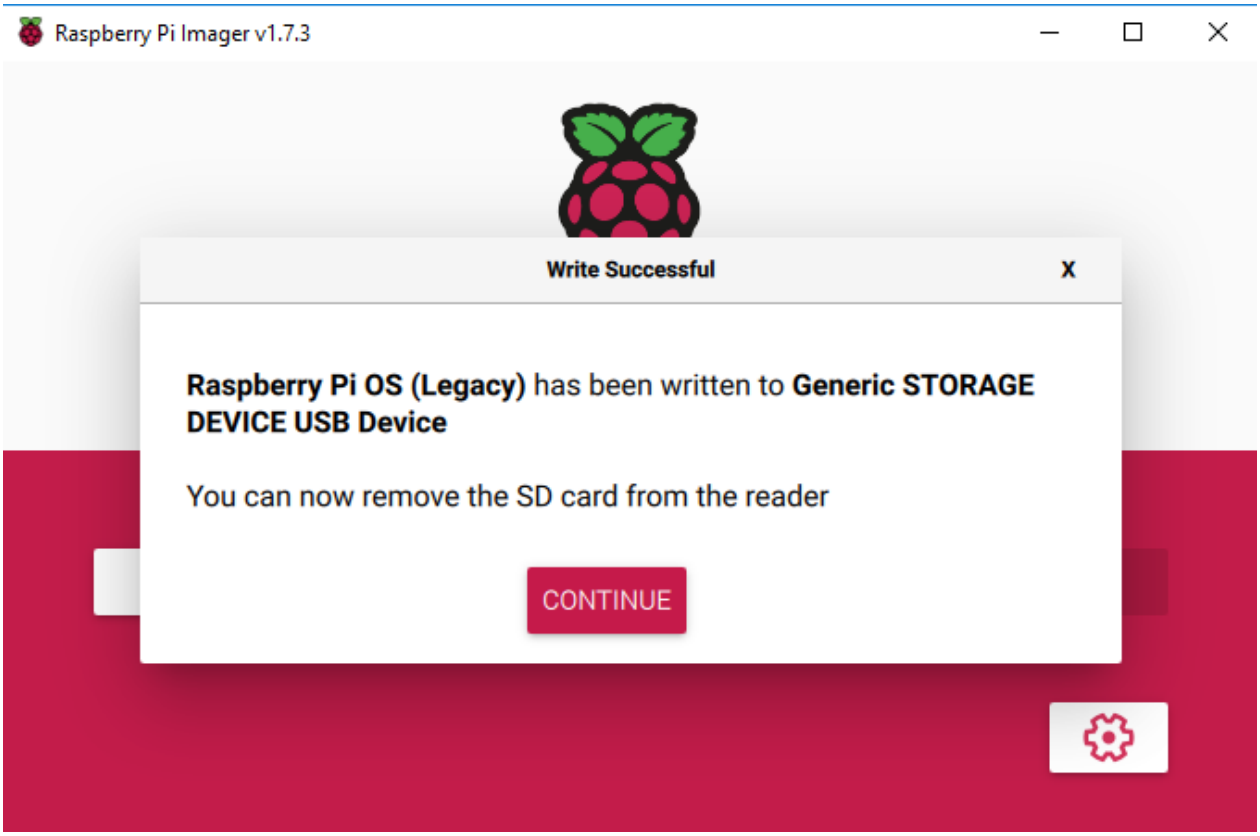
ステップ 8

現在 SD カードにファイルが保存されている場合はそれらのファイルを削除してしまわないように、事前にファイルをバックアップすることをお勧めします。バックアップするファイルがない場合は、[**はい**] をクリックします。



ステップ 9

しばらく待つと、次のウィンドウが表示され、書き込みが完了したことを示します。そのまま SD カードを取り出して Raspberry Pi Imager を終了させます。



1.3.3 Raspberry Pi の設定

モニターに接続できる場合

PC 用のモニター画面（あるいは HDMI 入力のあるテレビ）があれば Raspberry Pi の操作は簡単です。

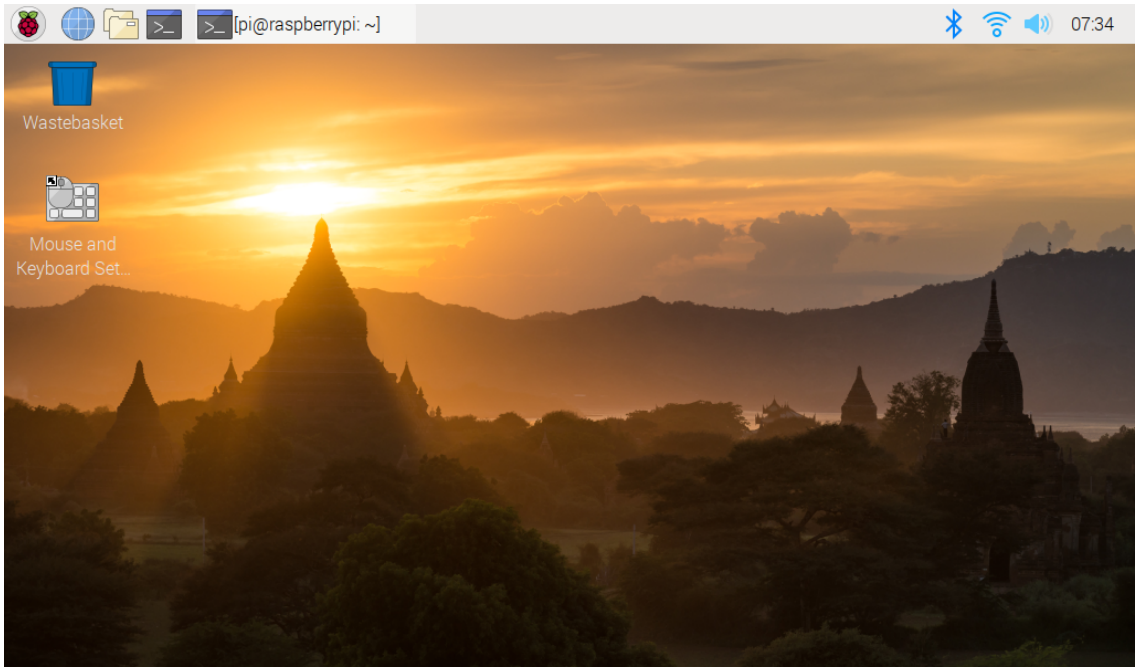
必要なもの

- Raspberry Pi 3 または 4
- 1 * 電源アダプター
- 1 * マイクロ SD カード
- 1 * モニター用電源アダプター（必要ならば）
- 1 * HDMI ケーブル
- 1 * モニター（もしくは TV）
- 1 * マウス
- 1 * キーボード

1. Raspberry Pi OS をインストールした SD カードを、Raspberry Pi の底面にあるマイクロ SD カード・スロットに挿入します。
2. マウスとキーボードを接続します。
3. モニターを Raspberry Pi の HDMI ポートに接続し、モニターの電源が壁のコンセントに差し込まれ電源が入っていることを確認します。

> **注釈:** Raspberry Pi 4 を使用する場合はモニターを HDMI0（電源ポートに最も近い方）に接続する必要があります。

4. 電源アダプターを使用して Raspberry Pi に電源を供給します。 少し待つとに Raspberry Pi OS デスクトップが表示されます。なお Raspberry pi には電源スイッチがありませんので電源アダプターを接続するとそのまま起動します。



モニターがない場合

モニターがない場合は Raspberry Pi にリモートでログインできます。

SSH コマンドを適用して Raspberry Pi の Bash Shell を開くことができます。Bash は Linux の標準のデフォルト・シェルです。シェル自体はユーザーが Unix/Linux を利用する際のコマンドラインインタプリタ（CLI）です。シェルを通して必要な作業のほとんどが行えます。

コマンド・ウィンドウを使用して Raspberry Pi を利用するすることに満足できない場合はリモートデスクトップ機能を使用して GUI を利用して Raspberry Pi 内のファイルを簡単に管理することもできます。

各システムの詳しいチュートリアルは下記をご覧ください。

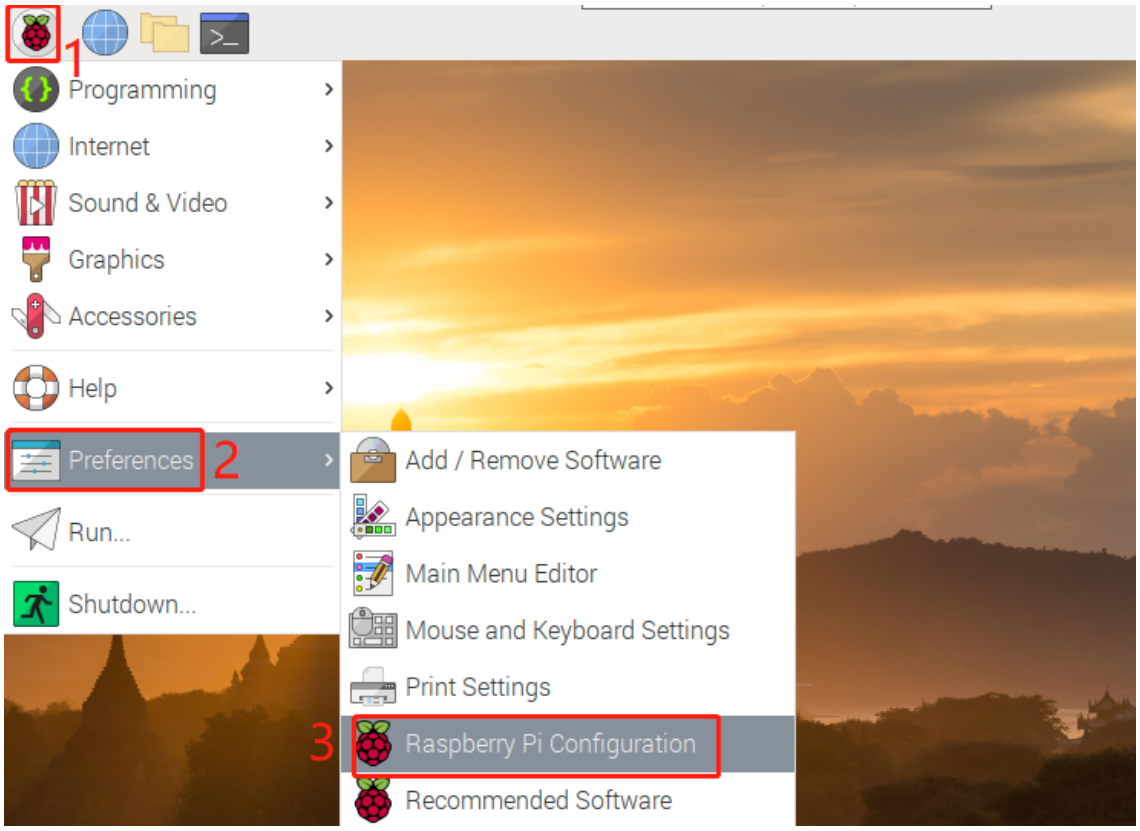
Mac OS X ユーザ

しかし Mac ユーザーの場合は可能であれば一時的にでもモニター、キーボード、マウスをセットして Rspberry pi 側の VNC の設定を行った方が楽です。まずはその方法から説明いたします。

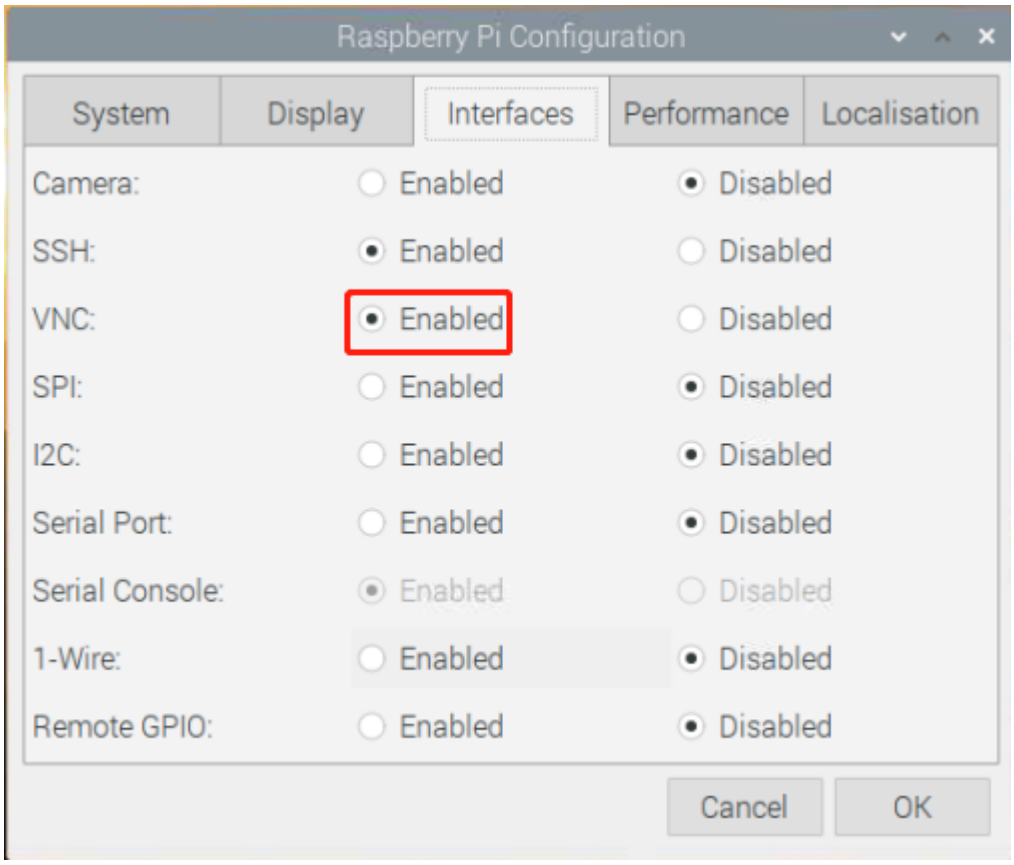
なお予め断っておきますが、この方法の場合 Mac と Raspberry pi の間の通信は暗号化による保護がなされません。実際の利用に際しては個人宅もしくは社内のネットワーク内での通信となるので保護がなくても問題にはならないでしょう。しかし気になる方は別途記述する VNC アプリを導入してご利用ください。

Mac OS X ユーザーで一時的にモニター（あるいは TV）を利用できる方

1. Raspberry Pi にディスプレイ、マウス、キーボードを接続し、電源を入れる。図の番号に従ってメニューを選択します。

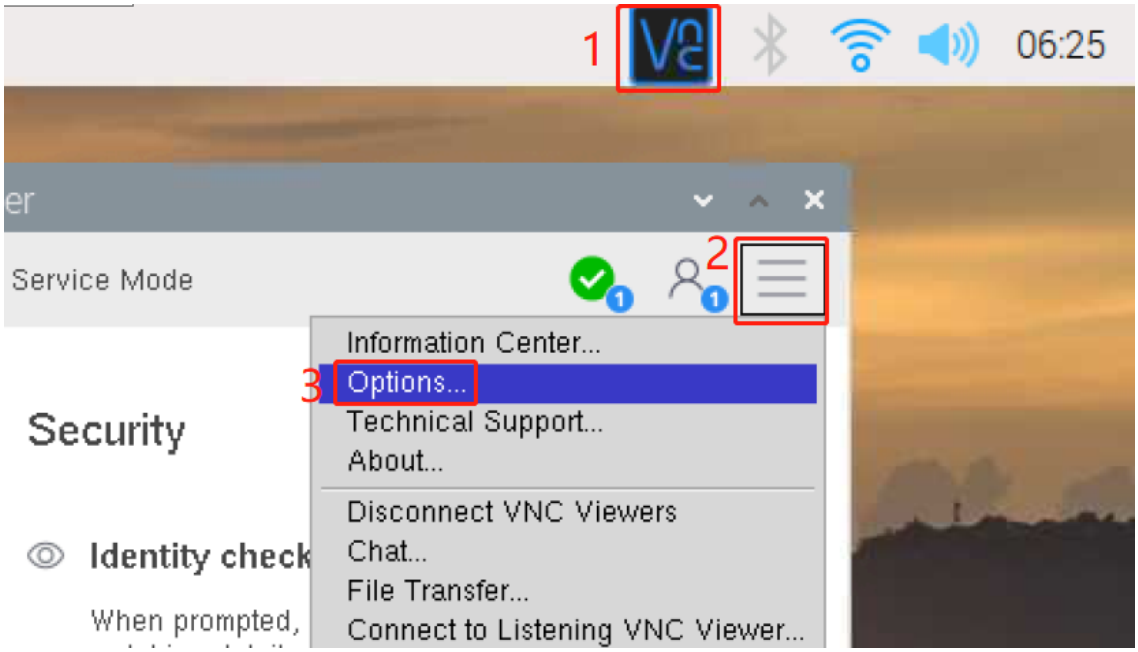


2. 以下の画面が表示されるので **Interfaces** タブをクリックし、さらに **VNC** を **Enabled** に設定してから画面右下の **OK** ボタンをクリックします。

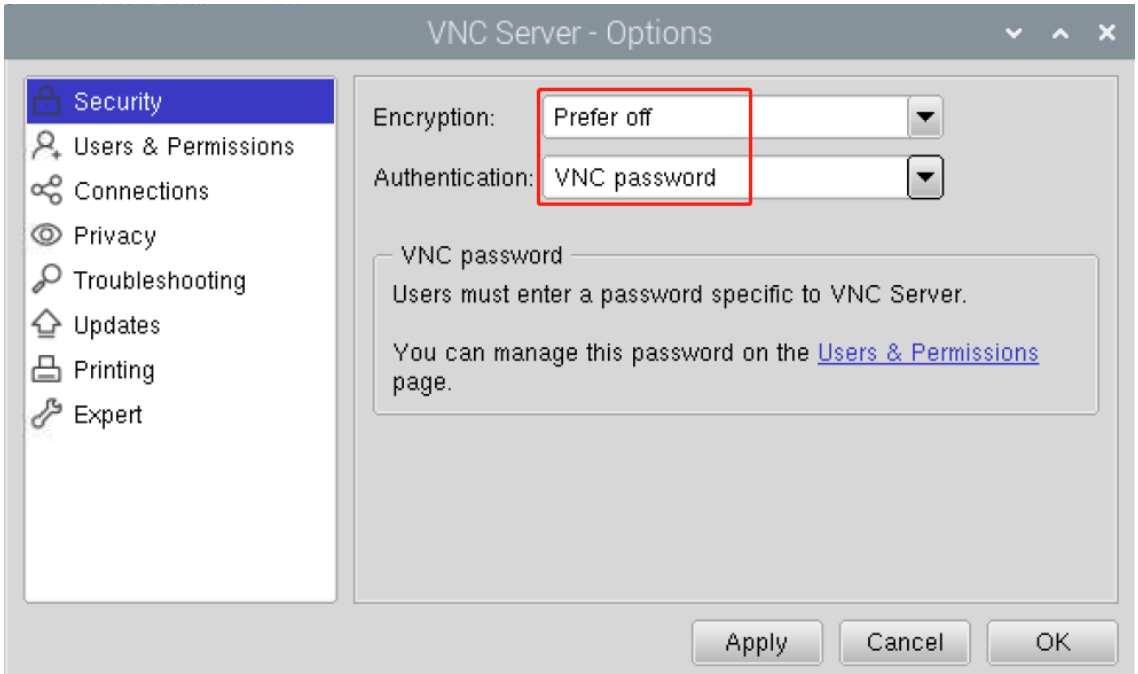


3. 画面右上に **VNC** のアイコンが表示され VNC サーバーが起動します。

4. 表示された **VNC** アイコンをクリックして VNC Server ウィンドウを表示し、さらに右上のメニューボタンをクリック、続いて **Options** をクリックします



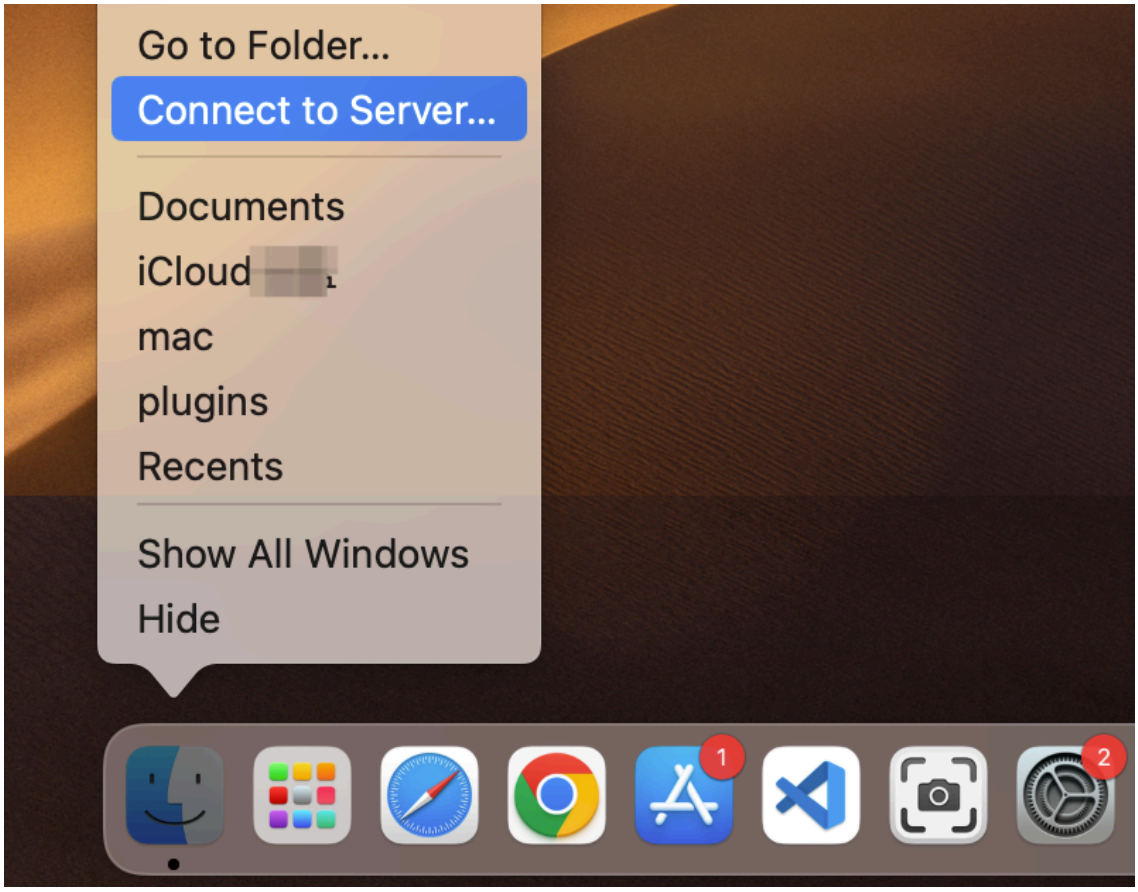
5. 以下の画面が表示されるのでオプションを変更します。



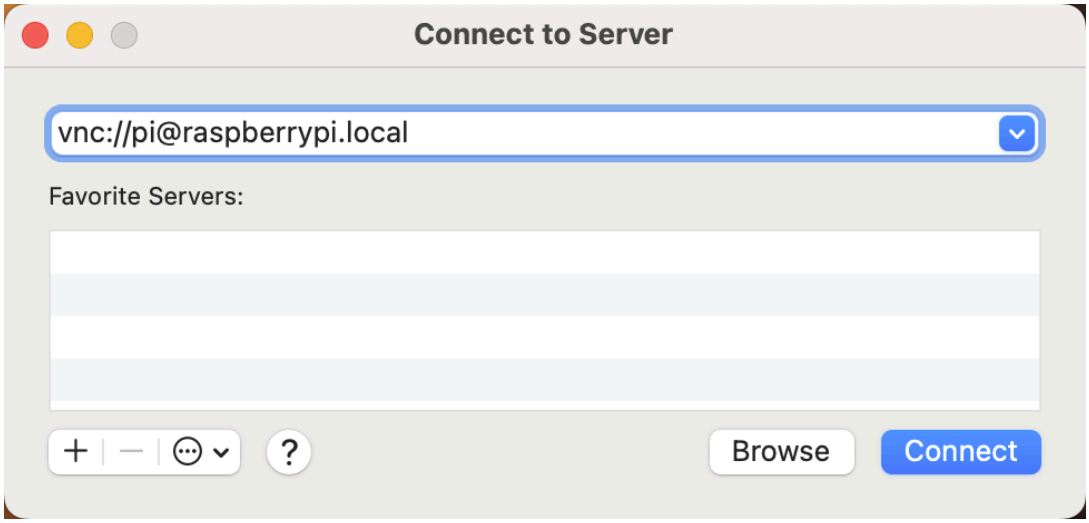
Encryption を **Prefer offon** に、Authentication を **VNC password** に設定してください。**OK** ボタンをクリックするとパスワードの入力画面が表示されるので Raspberry pi のパスワードと同じでも、別のパスワードでも構わないので入力して **OK** をクリックしてください。これで Mac から接続できる準備が整いました。もうモニターなどは接続を外しても大丈夫です。

ここからは Mac 側での操作になります。

1. ファインダーの移動メニューからサーバーへ接続を選択します。

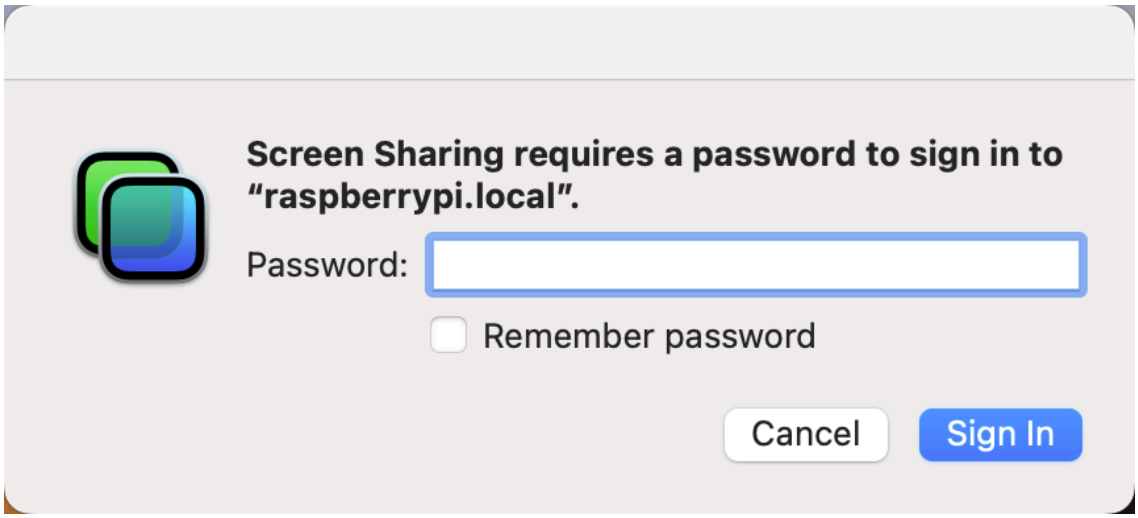


2. Raspberry pi のアドレスを入力します。以下のように **Raspberri pi の名前** ＋ **.local** でアドレスを指定します。

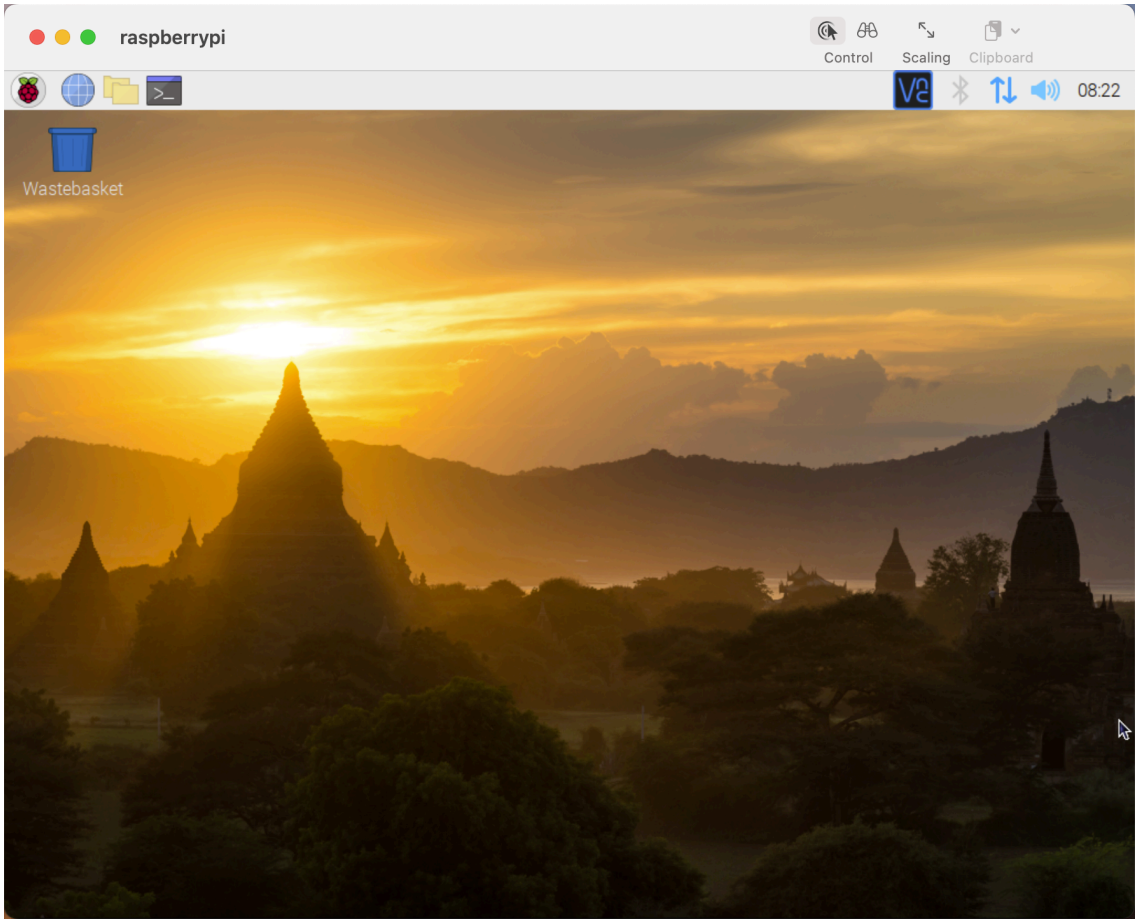


接続 ボタンをクリックします。

3. パスワードを聞いてくるので入力してください。



4. Raspberry pi のデスクトップが表示され、Mac 上からそのまま操作ができるようになります。

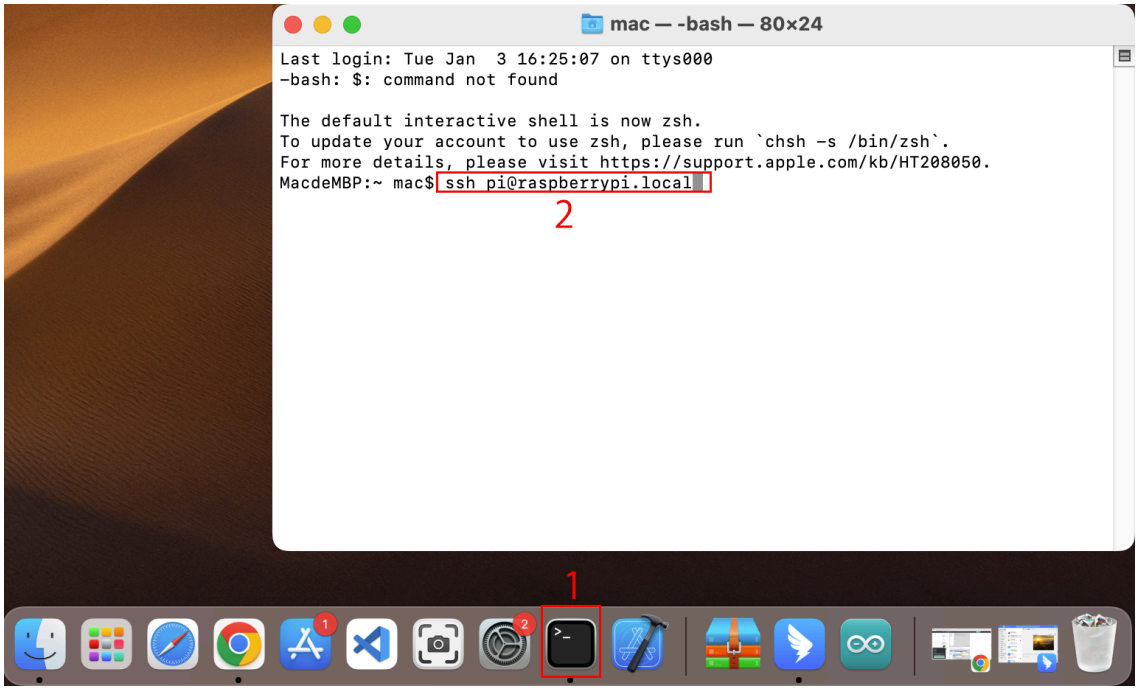


Mac OS X ユーザーで一時的であってもモニター（あるいは TV）を利用できない方

- SSH コマンドを適用して Raspberry Pi の Bash Shell を開くことができます。
- Bash は Linux の標準のデフォルト・シェルです。
- シェル自体はユーザーが Unix/Linux を利用する際のコマンド（命令）です。
- シェルを通して必要な作業のほとんどが行えます。
- Raspberry pi 側の設定が終われば前述の **Mac** からファインダーを使って Raspberry pi のデスクトップにアクセスできます。

1. Raspberry Pi への **SSH** 接続 ssh <username>@<hostname>.local と入力して **リターンキー** を押します。

```
ssh pi@raspberrypi.local
```



2. 最初にログインするときだけ以下のメッセージが表示されるので **yes** と入力してください。

```
The authenticity of host 'raspberrypi.local␣
↪(2400:2410:2101:5800:635b:f0b6:2662:8cba)' can't be established.
ED25519 key fingerprint is SHA256:oo7x3ZSgAo032wD1tE8eW0fFM/kmewIvRwkBys6XRwg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

3. Raspberry pi の **パスワード** を入力します。パスワードは入力しても表示されませんので入力を間違えないように注意してください。

```
pi@raspberrypi.local's password:
Linux raspberrypi 5.15.61-v8+ #1579 SMP PREEMPT Fri Aug 26 11:16:44 BST 2022␣
↪aarch64

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: Thu Sep 22 12:18:22 2022
pi@raspberrypi:~ $
```

4. これで Raspberry pi に正常にログインできましたので、Mac から VNC でログイン出来るように設定します。まずは OS のアップデートを行います。以下のコマンドを入力します。

   ```
   sudo apt update
   sudo apt upgrade
   ```

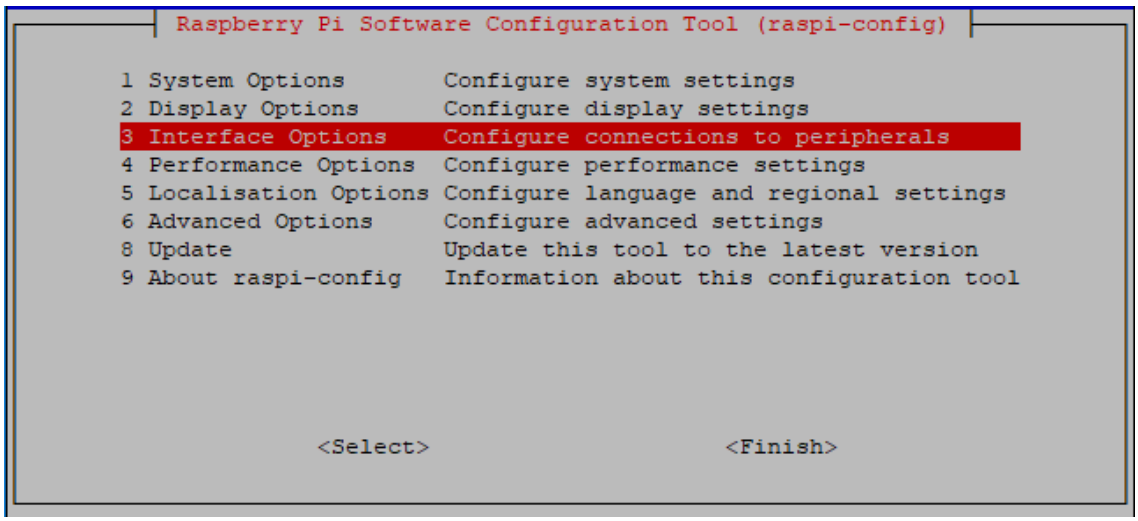
   `Do you want to continue? [Y/n]` が表示されたら **Y** を入力します。

   アップデートが終わるまでしばし時間がかかりることがあります。（その時のアップデート量によります。）

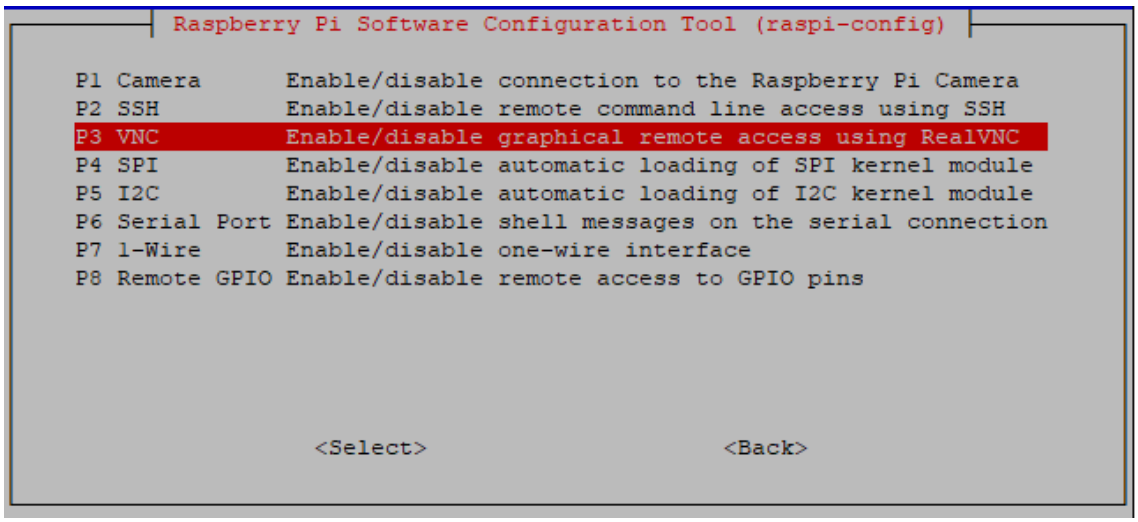
5. VNC を有効化して VNC サーバーを起動します。以下のコマンドを入力します。

   ```
   sudo raspi-config
   ```

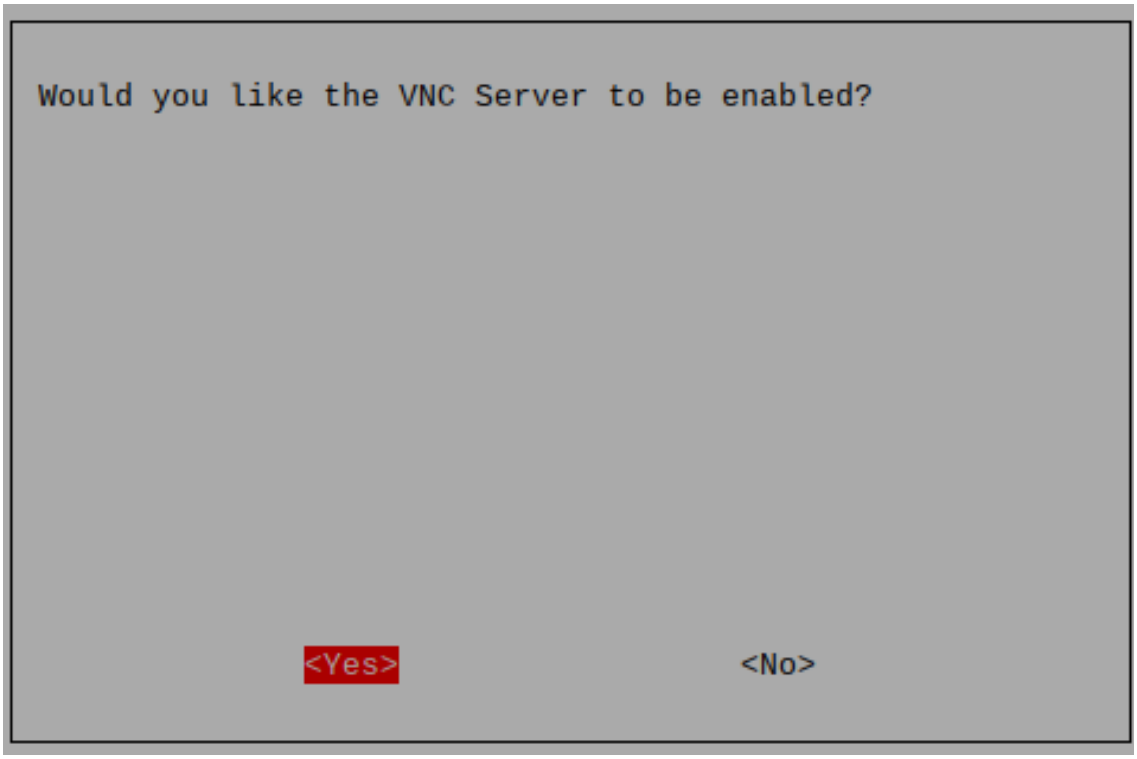
6. 以下の画面が表示されますので、キーボードの矢印キーで **3 Interface Options** を選択してリターンキーを押します。



7. 以下の画面が表示されますので、キーボードの矢印キーで **P3 VNC** を選択してリターンキーを押します。

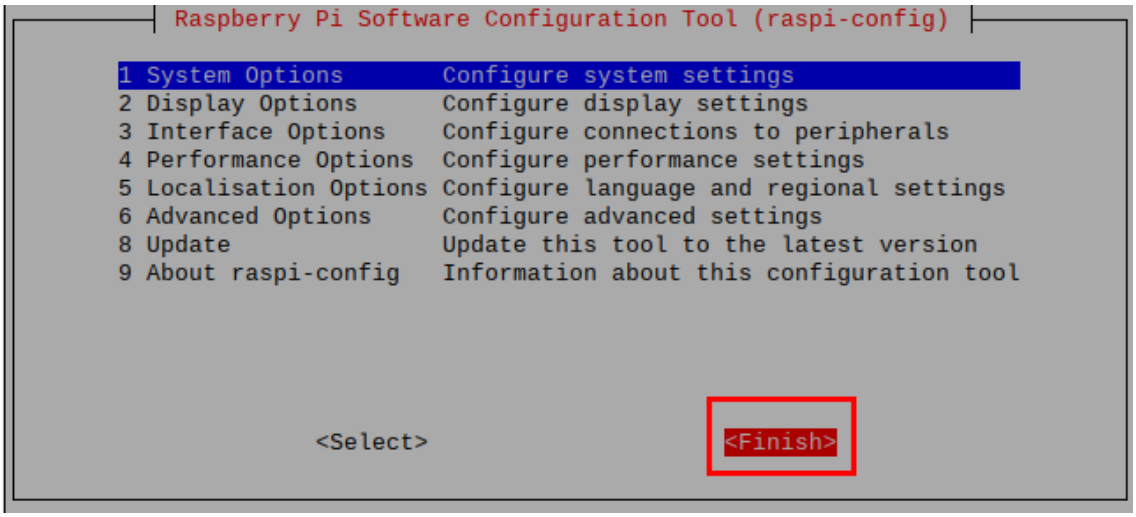


8. 以下の画面が表示されますので、キーボードの矢印キーで **<Yes>** を選択してリターンキーを押します。



9. 以下の画面が表示されますので、キーボードの矢印キーで **<Ok>** を選択してリターンキーを押します。

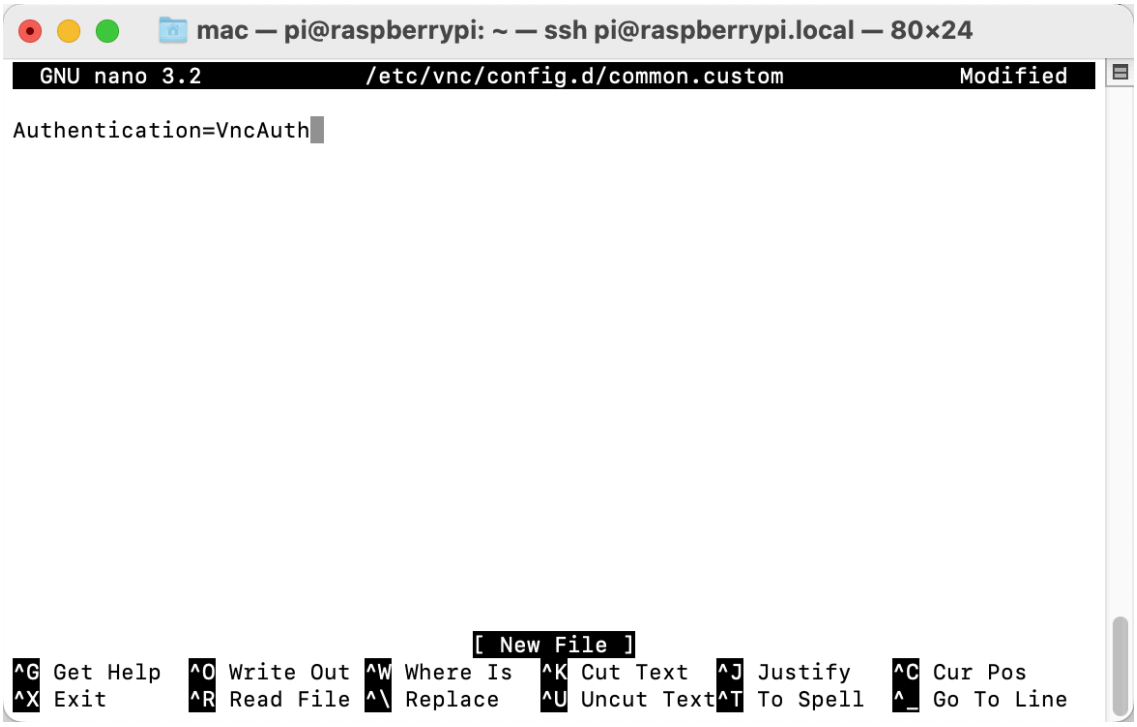
10. 最後にキーボードの矢印キーで **<Finish>** を選択してリターンキーを押します。



11. VNC サーバーが起動するようになったので、今度は Mac からの接続様の設定を変更します。

```
sudo nano /etc/vnc/config.d/common.custom
```

以下のような画面になるので Authentication=VncAuth と入力し、 Ctl + o を押し、リターンキーを押し、さらに Ctl + x を押します。



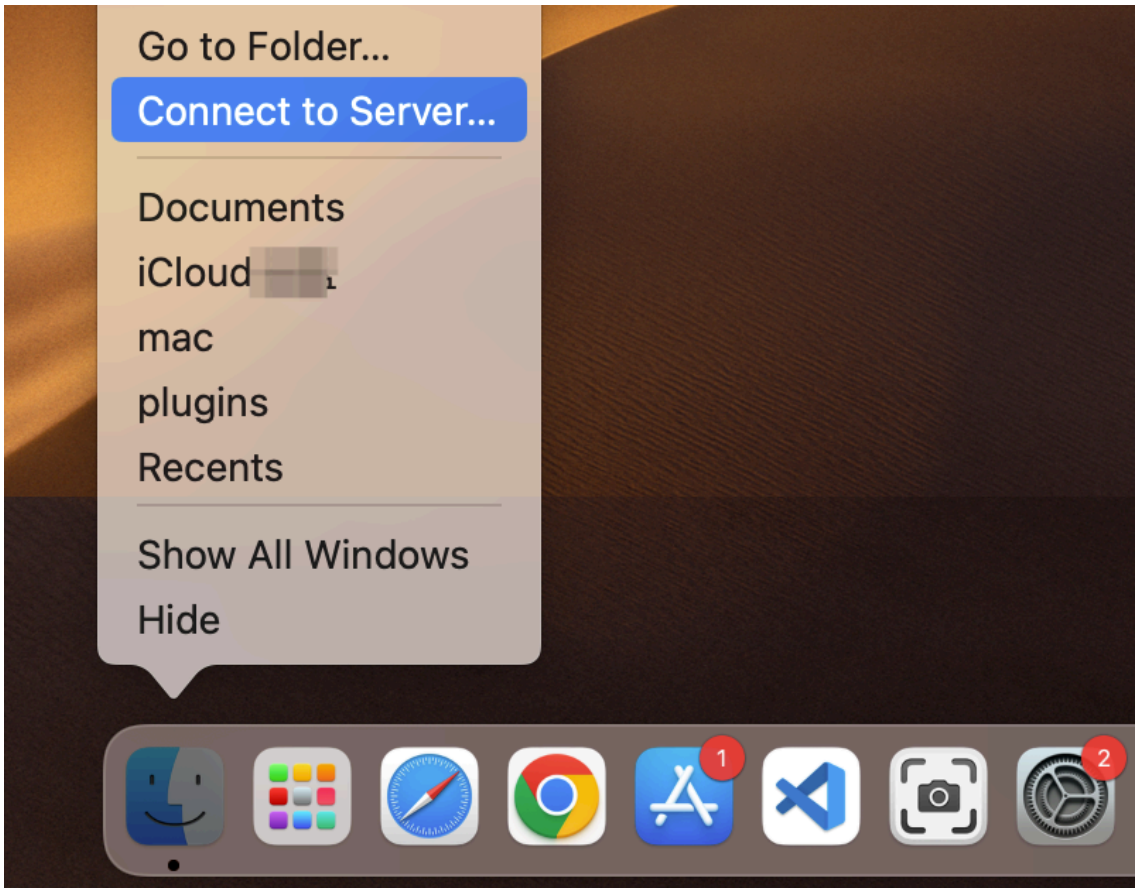
12. さらに Mac から VNC でログインするときのパスワードを設定します。Raspberry pi のパスワードと同じで問題はないはずです。

```
sudo vncpasswd -service
password: (パスワードを入力)
```

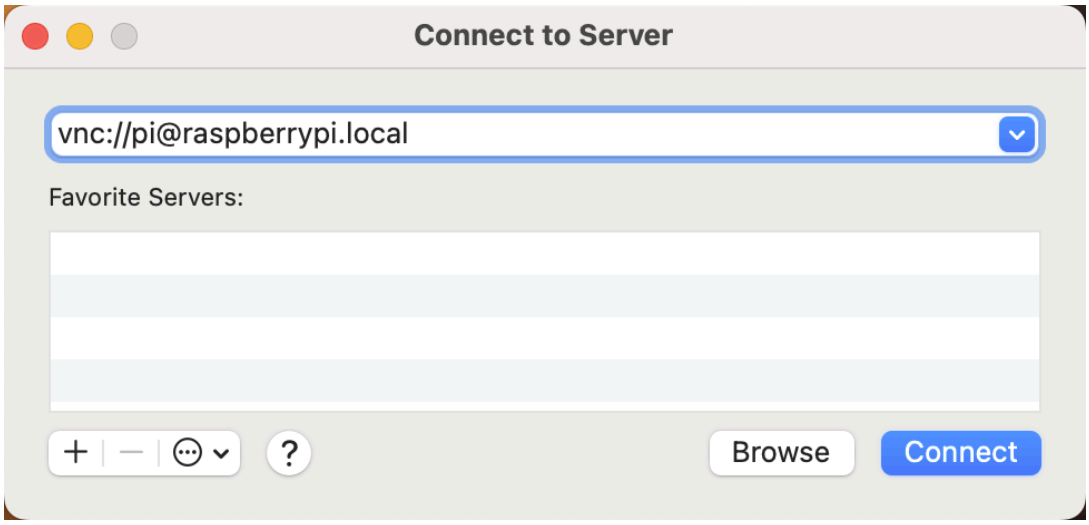
13. これで設定が出来ましたので設定を反映させるために Raspberry pi を再起動します。

```
sudo sudo reboot
```

14. ファインダーの移動メニューからサーバーへ接続を選択します。

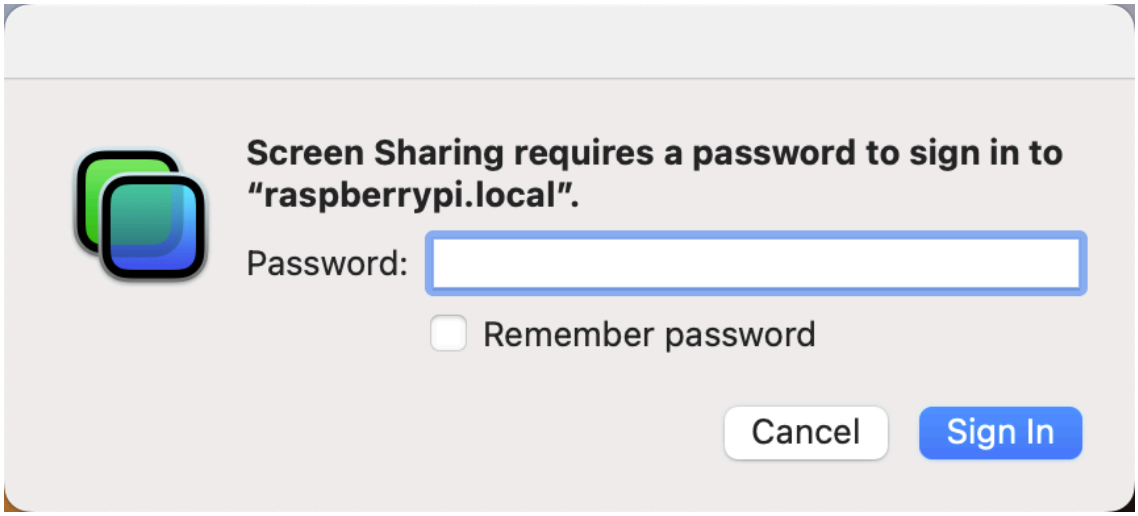


15. Raspberry pi のアドレスを入力します。以下のように **Raspberri pi の名前** ＋ **.local** でアドレスを指定します。

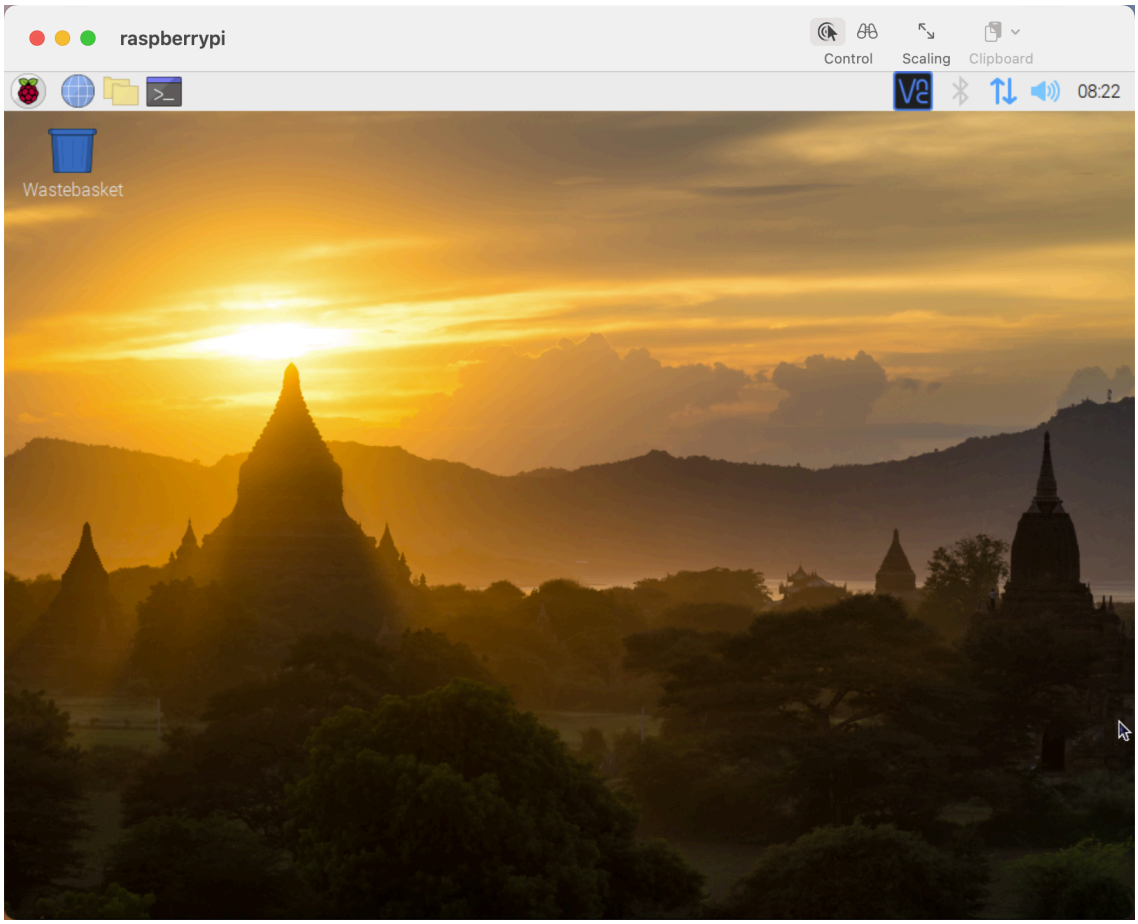


Connect ボタンをクリックします。

16. パスワードを聞いてくるので入力してください。



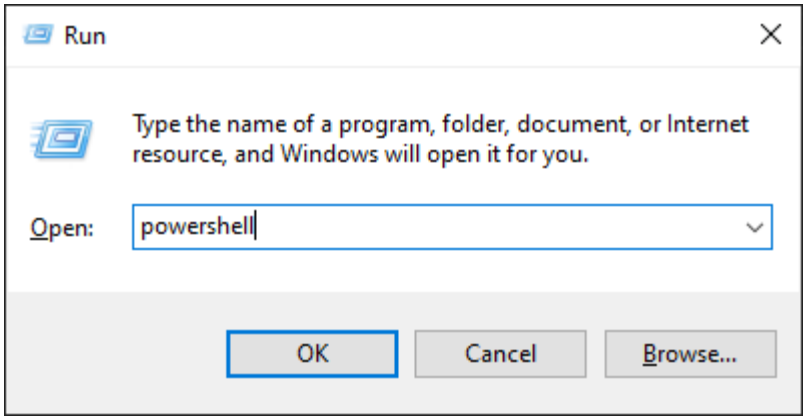
17. Raspberry pi のデスクトップが表示され、Mac 上からそのまま操作ができるようになります。



Windows ユーザ

Windows ユーザーの場合は **Windows PowerShell** を利用できます。

1. `windows + R` キーを押し入力ウィンドウに **powershell** と入力して PowerShell を起動してください。



2. 後は Mac や Linux のターミナル操作と同じで開かれた PowerShell の画面で `ssh <username>@<hostname>.local` と入力します。

```
ssh pi@raspberrypi.local
```

3. 以下のメッセージが表示されます.

```
The authenticity of host 'raspberrypi.local (192.168.6.116)' can't be␣
↪established.
ECDSA key fingerprint is SHA256:7ggckKZ2EEgS76a557cddfxFNDOBBuzcJsgaqA/igz4.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

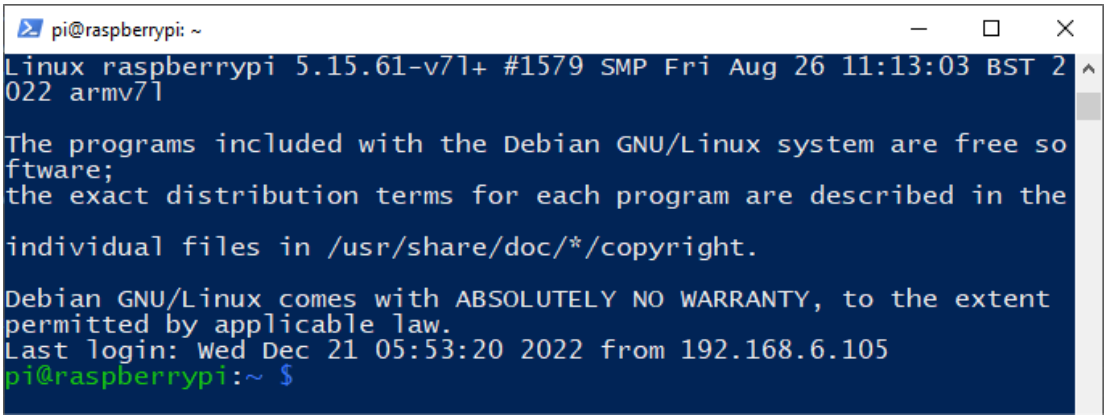
"yes" と入力してください。

4. パスワードを入力してください。 (デフォルトの場合は raspberry です。)

注釈: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

---

5. これで Raspberry pi に正常にログインできましたので、OS のアップデートを行います。



リモート・デスクトップ

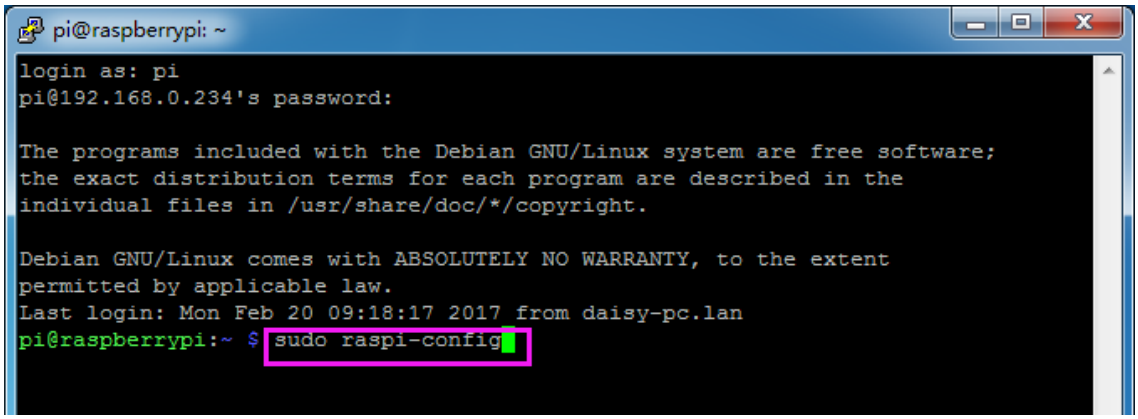
VNC でリモート・デスクトップを利用できます。

VNC サービスの有効化

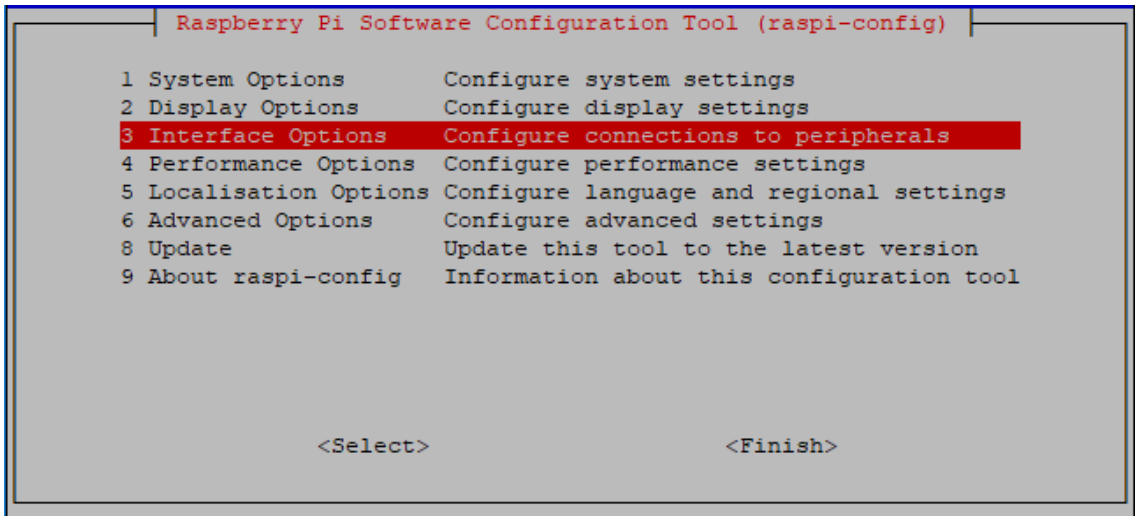
VNC は標準でインストールされておりますが、デフォルトでは無効化されていますので、有効化する必要があります。

1. 以下のコマンドを入力します:

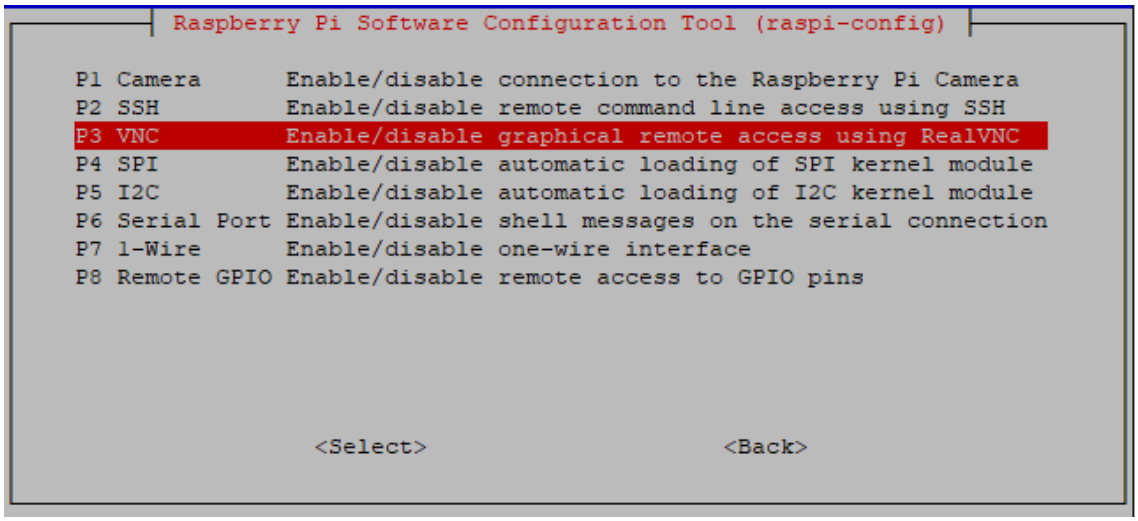
```
$ sudo raspi-config
```



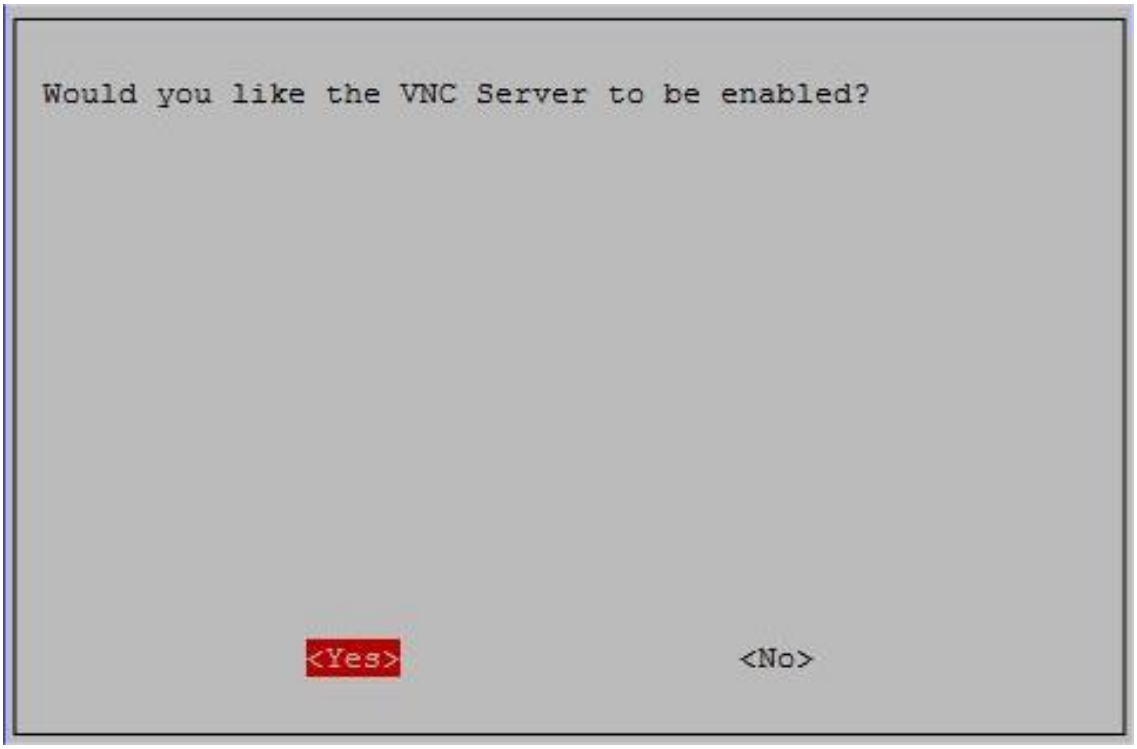
2. **3** の **Interfacing Options** を下矢印で選択してリターン機を押します。



3. **P3 VNC** を矢印キーで選択してリターンキーを押します。



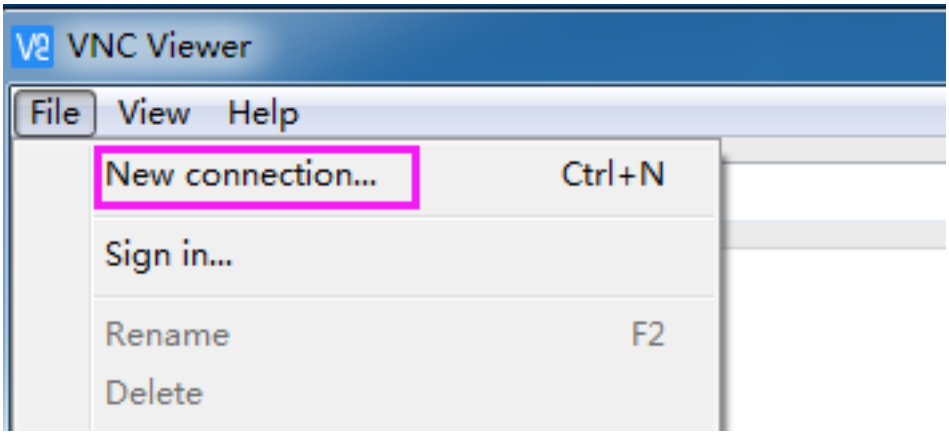
4. **Yes** → **OK -> Finish** と順番に選択して設定を終了させます。



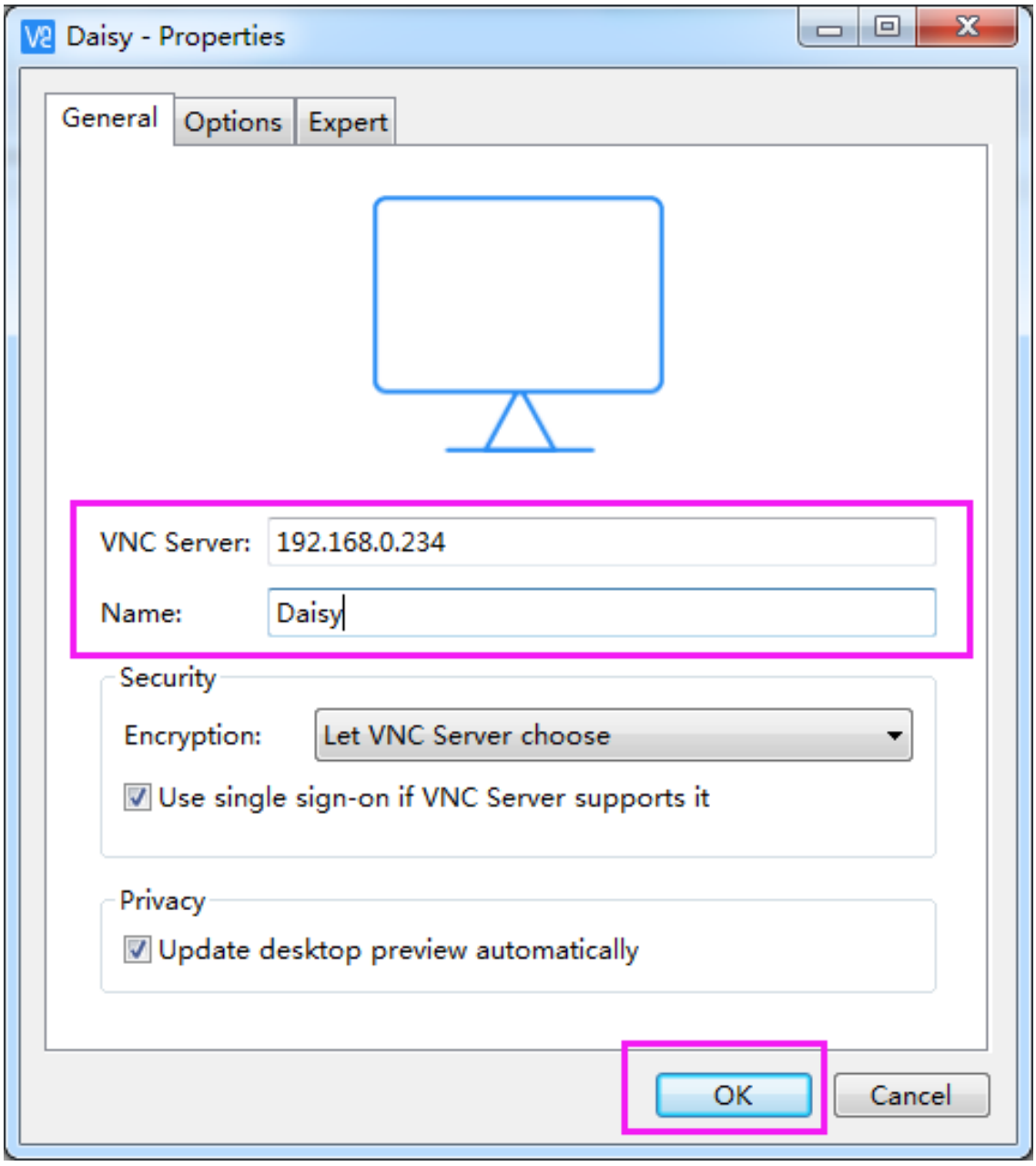
VNC によるログイン

1. 先ず VNC Viewer をダウンロードしてファイルを開き、指示に従いインストールします。インストールしたら起動します。

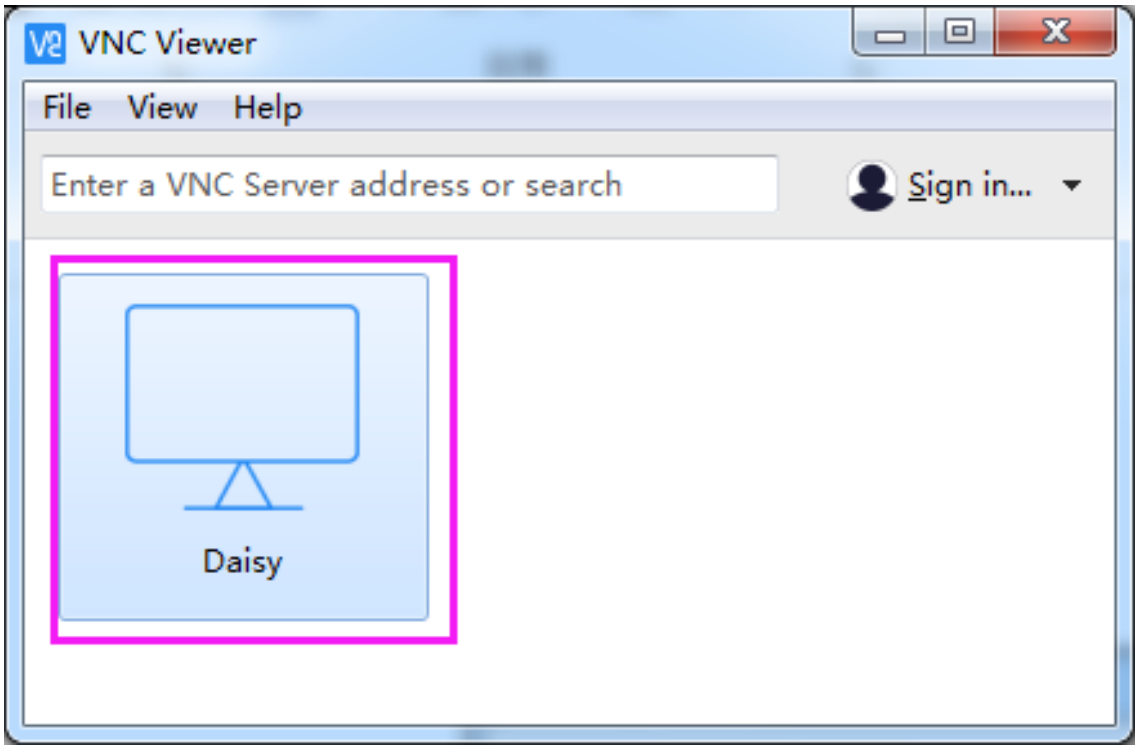
2. メニューのファイルから **New connection** を選択します。



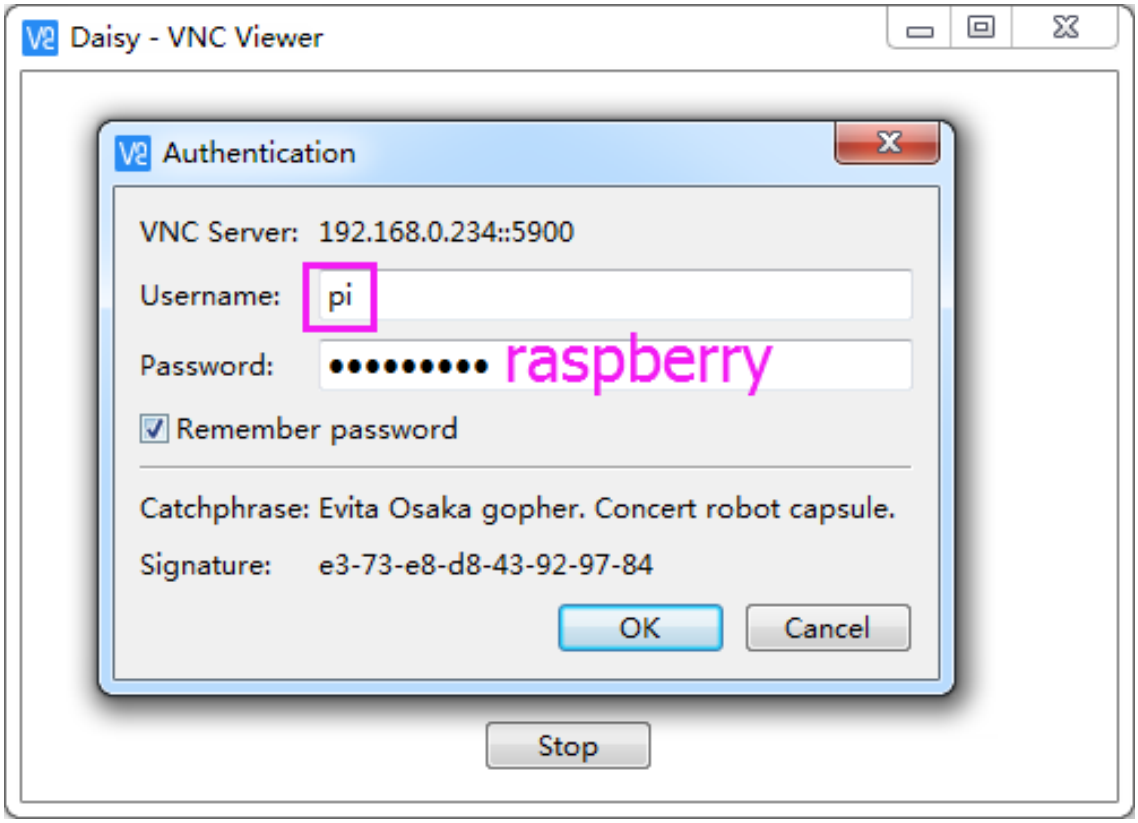
3. Raspberry pi の IP アドレスを入力します。 **Name** には好きな名前を付けます。(Raspberry pi の名前と同じが良いかもしれません) 「OK」をクリックします。



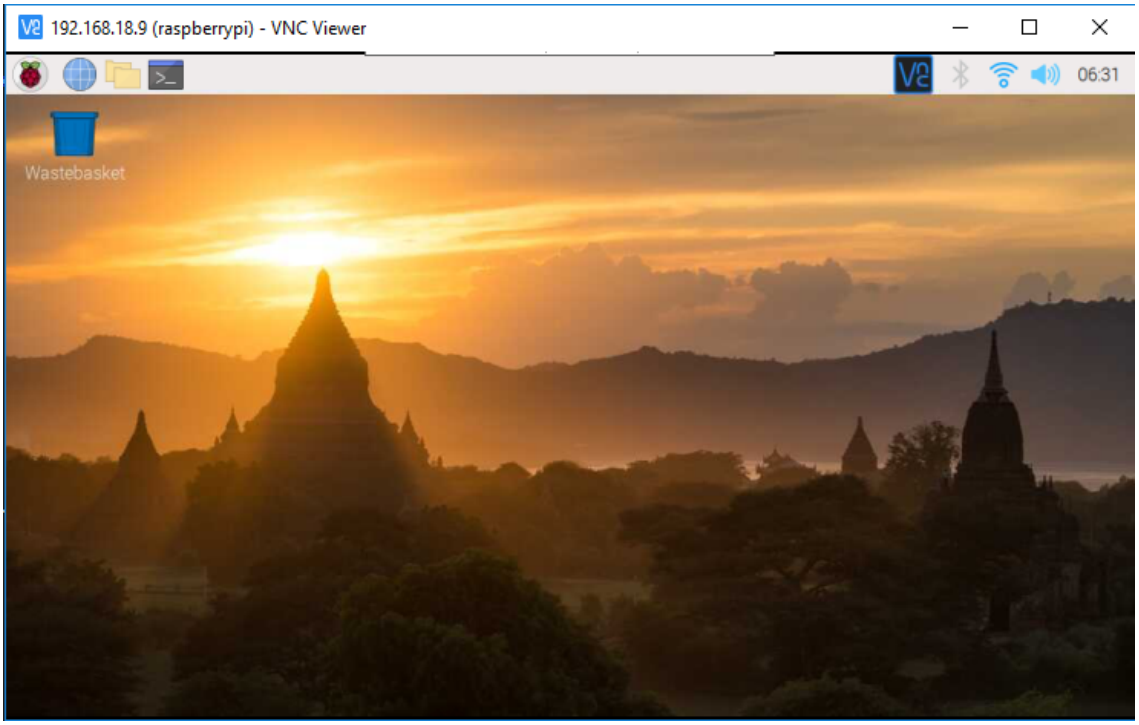
4. 作成した **接続先** をダブルクリックします。:



5. ログイン・ユーザー名 (**pi**) と SD カードを作成した際に指定したパスワード (デフォルトでは **raspberry**) を入力します。



6. Raspberry Pi のデスクトップ画面が表示されるはずです。:



Linux / UNIX ユーザー

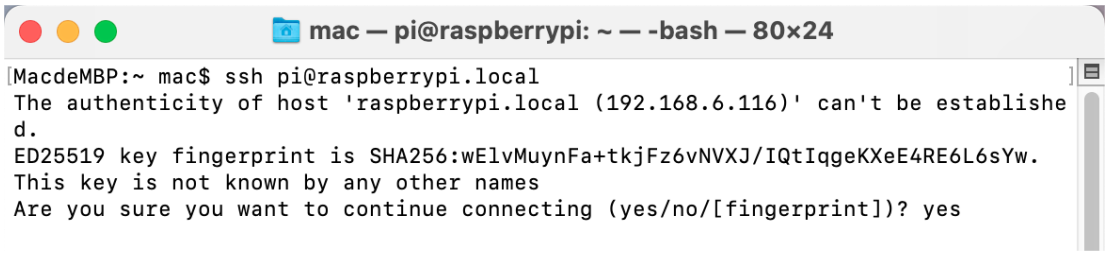
1. ターミナルを開いて ssh <username>@<hostname>.local と入力してください。

```
ssh pi@raspberrypi.local
```

2. 以下のメッセージが表示されます。

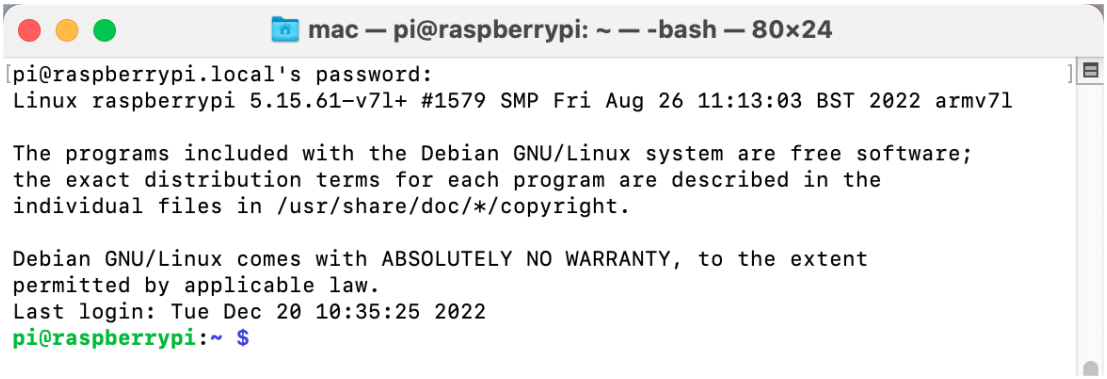
```
The authenticity of host 'raspberrypi.local (192.168.6.116)' can't be␣
↪established.
ECDSA key fingerprint is SHA256:7ggckKZ2EEgS76a557cddfxFNDOBBuzcJsgaqA/igz4.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

3. "yes" と入力してください。



4. パスワードを入力してください。 (デフォルトの場合は raspberry です。)

5. これで Raspberry pi に正常にログインできましたので、OS のアップデートを行います。



1.4 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 の Raspbian 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。

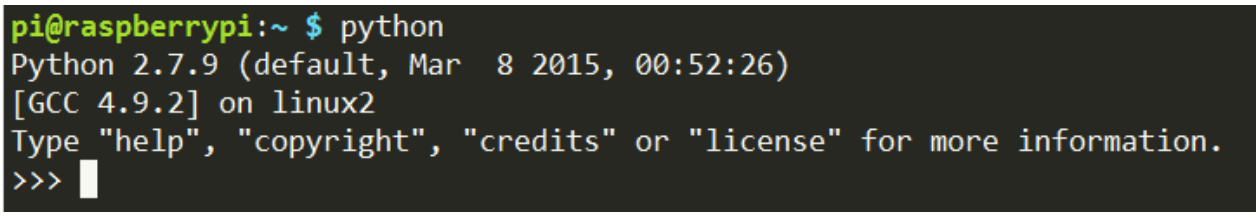
1.4.1 RPi.GPIO

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：http://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
```



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:~ $
```

1.5 WiringPi をインストールして確認する

wiringPi は Raspberry Pi 用の C 言語 GPIO ライブラリです。 GUN Lv3 に準拠。 wiringPi の機能は、Arduino の配線システムに似ています。 これにより、Arduino に精通しているユーザーは、wiringPi をより簡単に使用できるようになります。

wiringPi には、Raspberry Pi のさまざまなインターフェイスを制御できるようにする多数の GPIO コマンドが含まれています。

次のコマンドを実行して wiringPi ライブラリをインストールしてください。

```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```

次の手順で、wiringPi ライブラリが正常にインストールされたかどうかをテストできます。

```
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.
```

次のコマンドで GPIO を確認します。

```
gpio readall
```

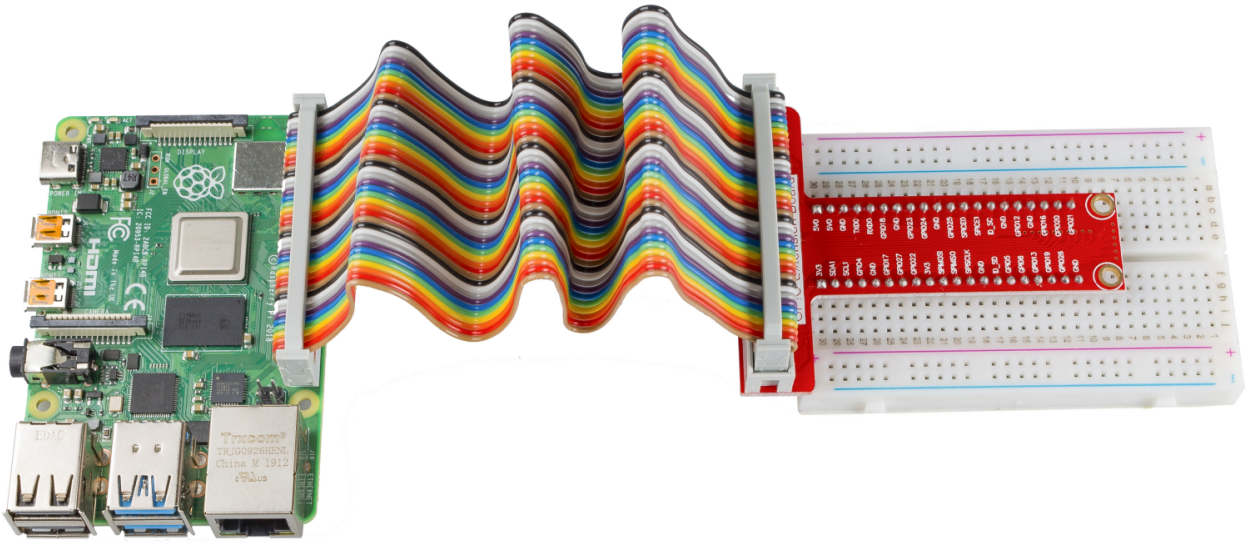
```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  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  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

配線 Pi の詳細については、 WiringPi を参照してください。

1.6 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。

注釈: 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、適用される命名方法は Board と BCM であり、関数 GPIO.setmode() を使用してそれらを設定します。

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	CE0	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

1.7 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/
```

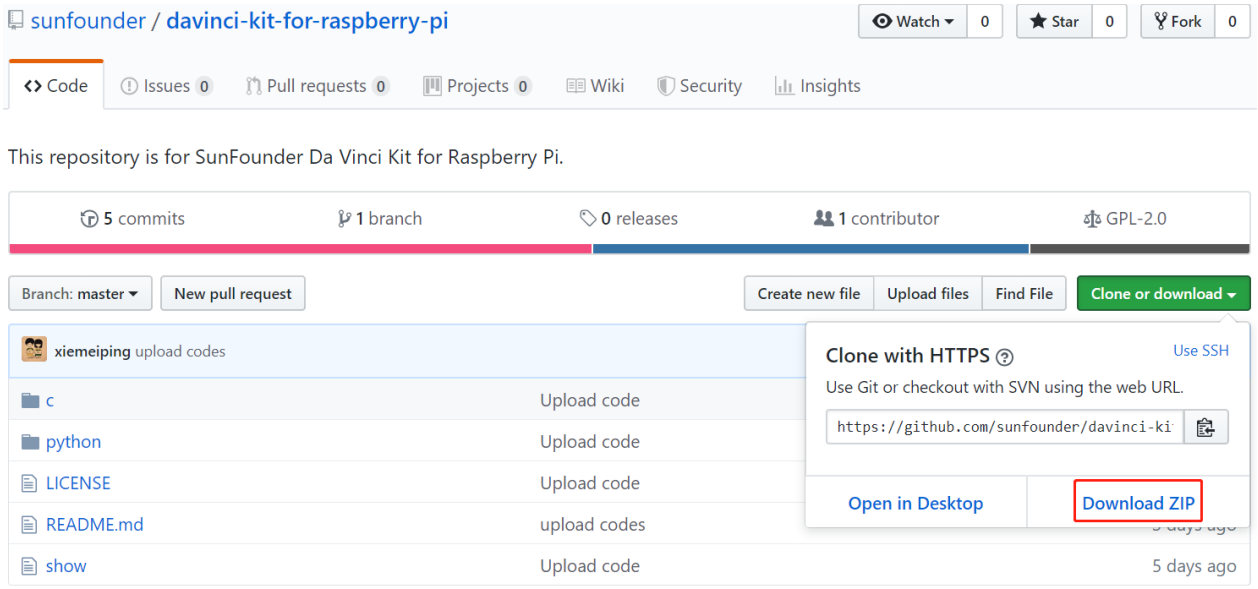
注釈: 現在のパスから目的のディレクトリに移動する cd。非公式には、パス /home/pi/ に移動する。

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.



1.8 1 出力 (Pi 5 用ではない)

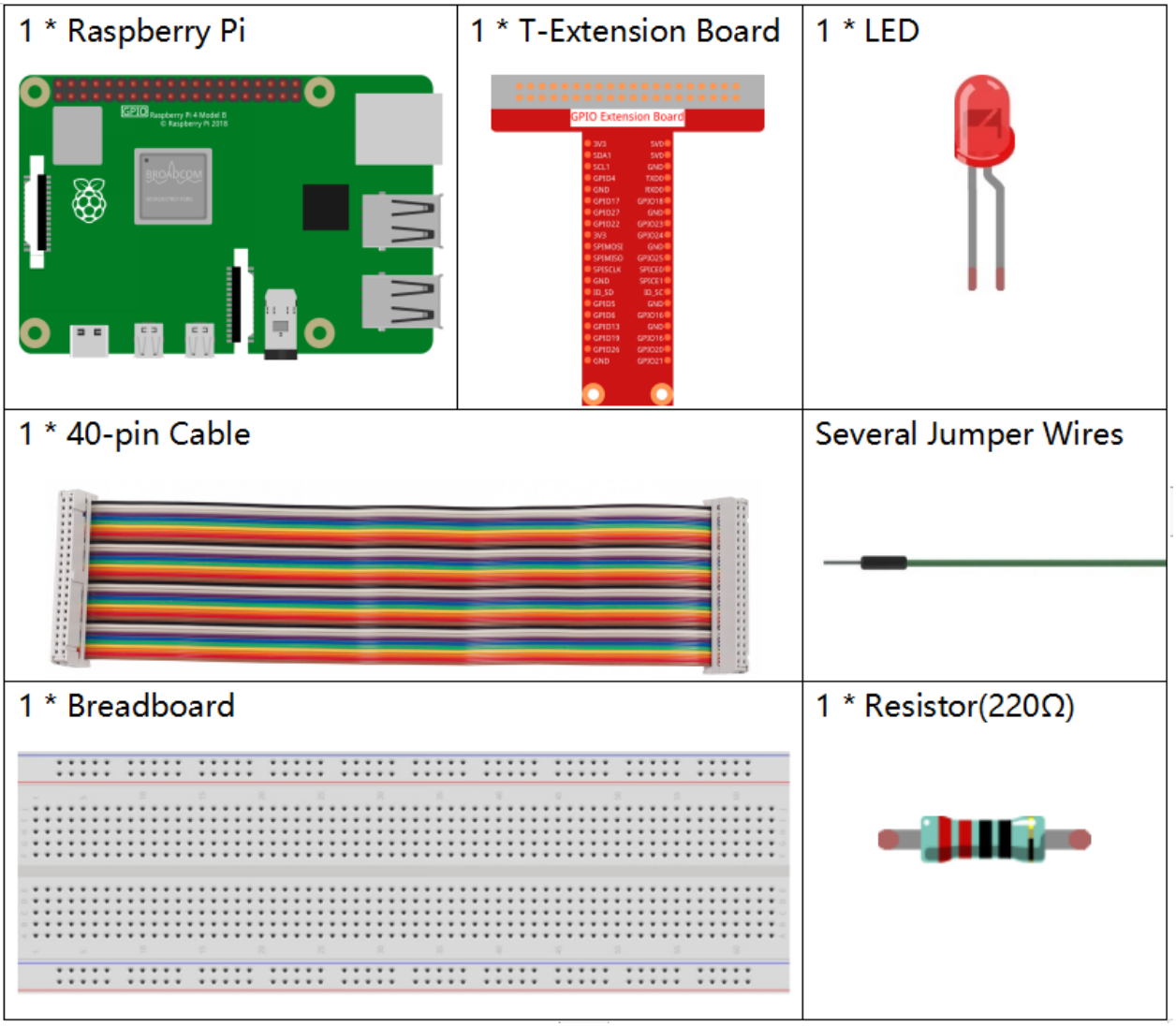
1.1 ディスプレイ

1.8.1 1.1.1 点滅 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



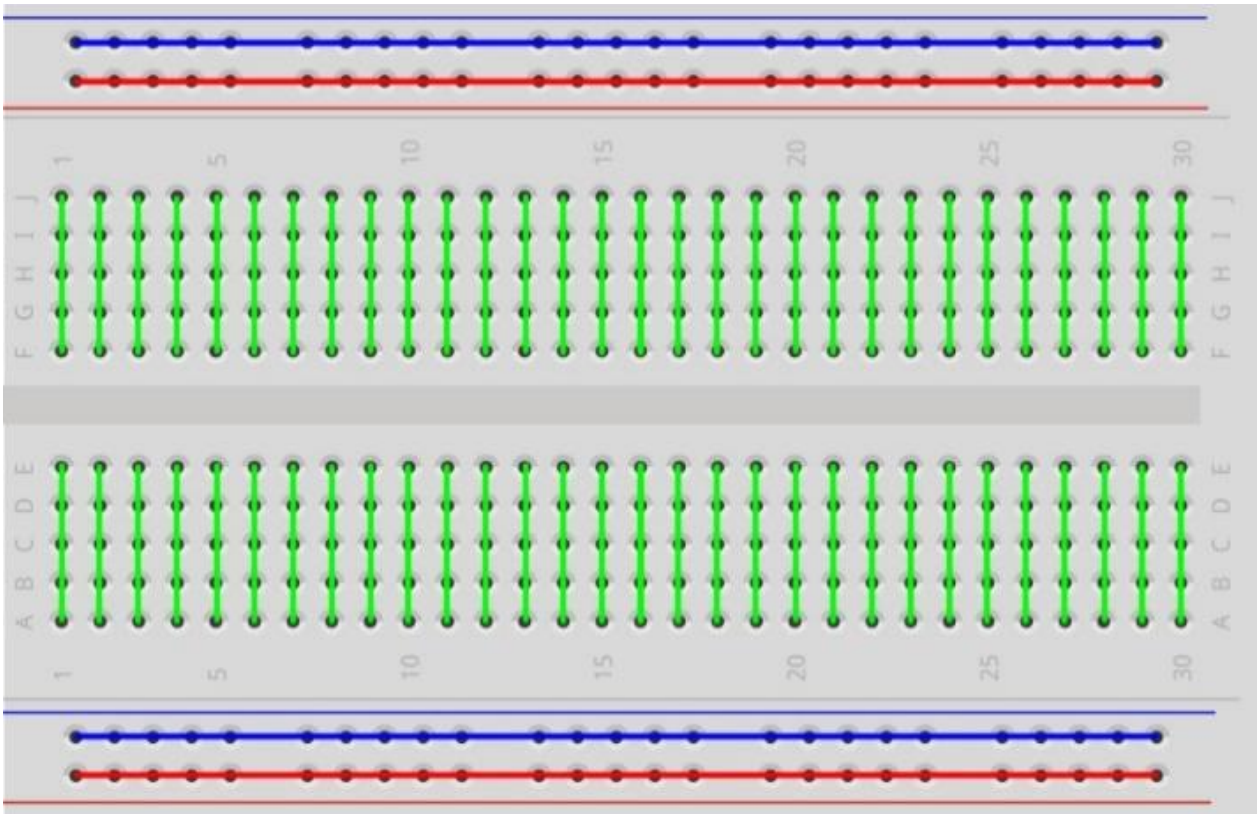
注釈: スムーズに進めるには、独自の Raspberry Pi、TF カード、および Raspberry Pi の電源を用意する必要がある。

原理

ブレッドボード

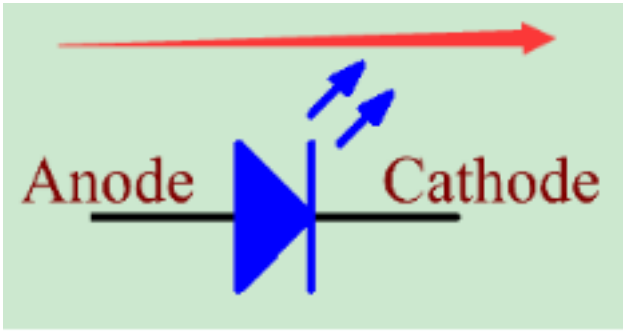
ブレッドボードは、電子装置のプロトタイピングの構築基盤である。それは回路設計を完了する前に、回路を迅速に作って、テストするために使用される。また、上記の部品を IC や抵抗器、ジャンパー線などのように挿入できる多くの穴が搭載されている。ブレッドボードを使用すると、部品を簡単に差し込んだり、取り外したりすることができる。

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。160 Ω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。

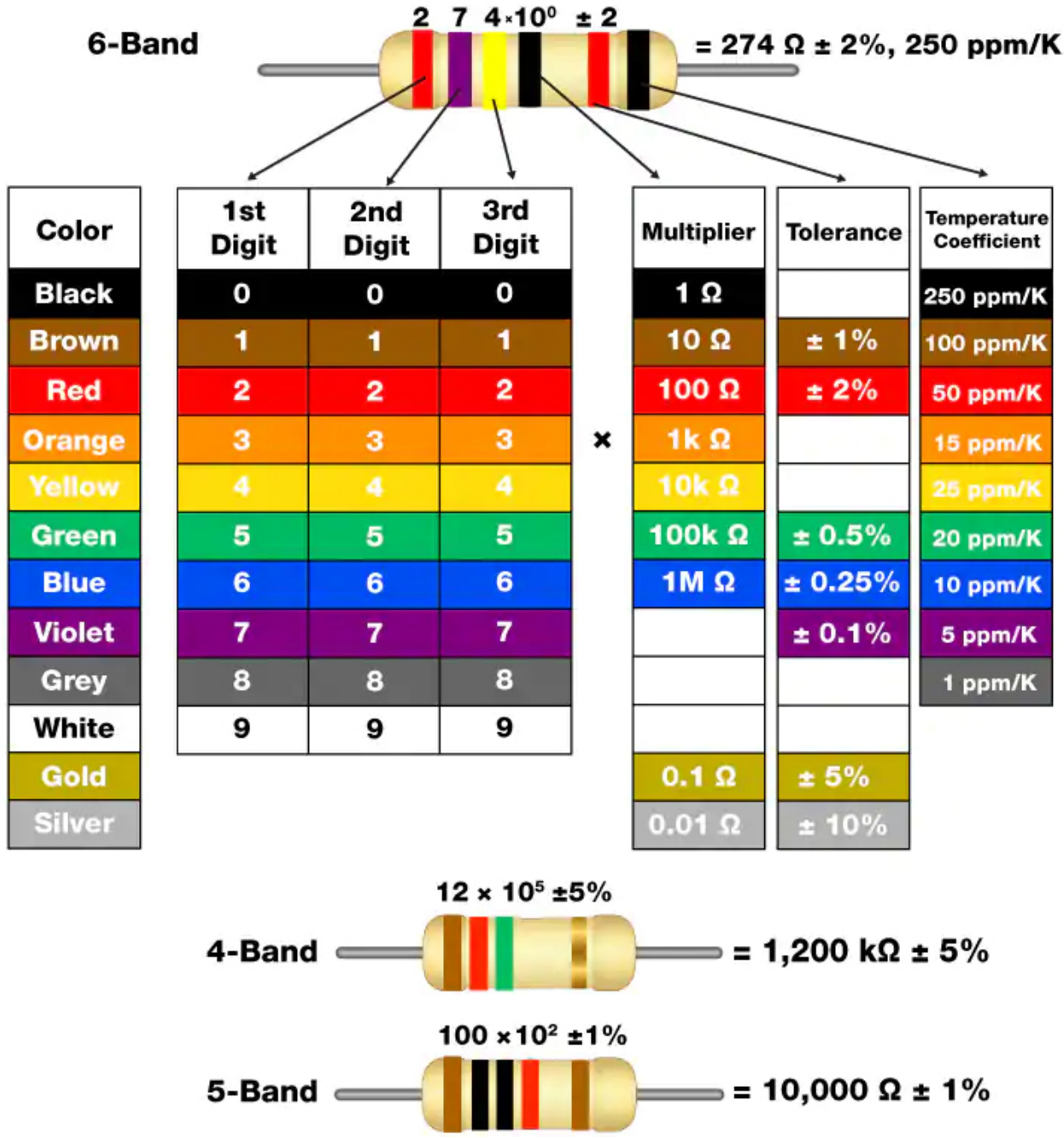
抵抗器

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、220 Ω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 K Ω、M Ωなどである。それらの関係は次のように示している：1M= 1000K Ω、1K Ω=1000 Ω。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

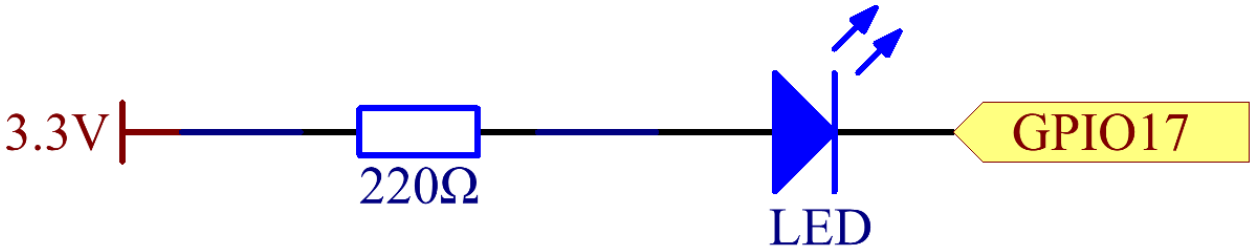
この実験では、まず 220 Ωの抵抗器を陽極（LED の長いピン）に接続し、それから抵抗器を 3.3 V の電源に接続し、LED の陰極（短いピン）を Raspberry Pi の GPIO17 に接続する。したがって、LED をオンにするには、GPIO17 を低（0V）レベルにする必要がある。プログラミングを通じてこの現象を取得できる

注釈: **Pin11** は Raspberry Pi の左から 11 番目のピンを指し、それに対応する **wiringPi** と **BCM** のピン番号は以

下の表の通りです。

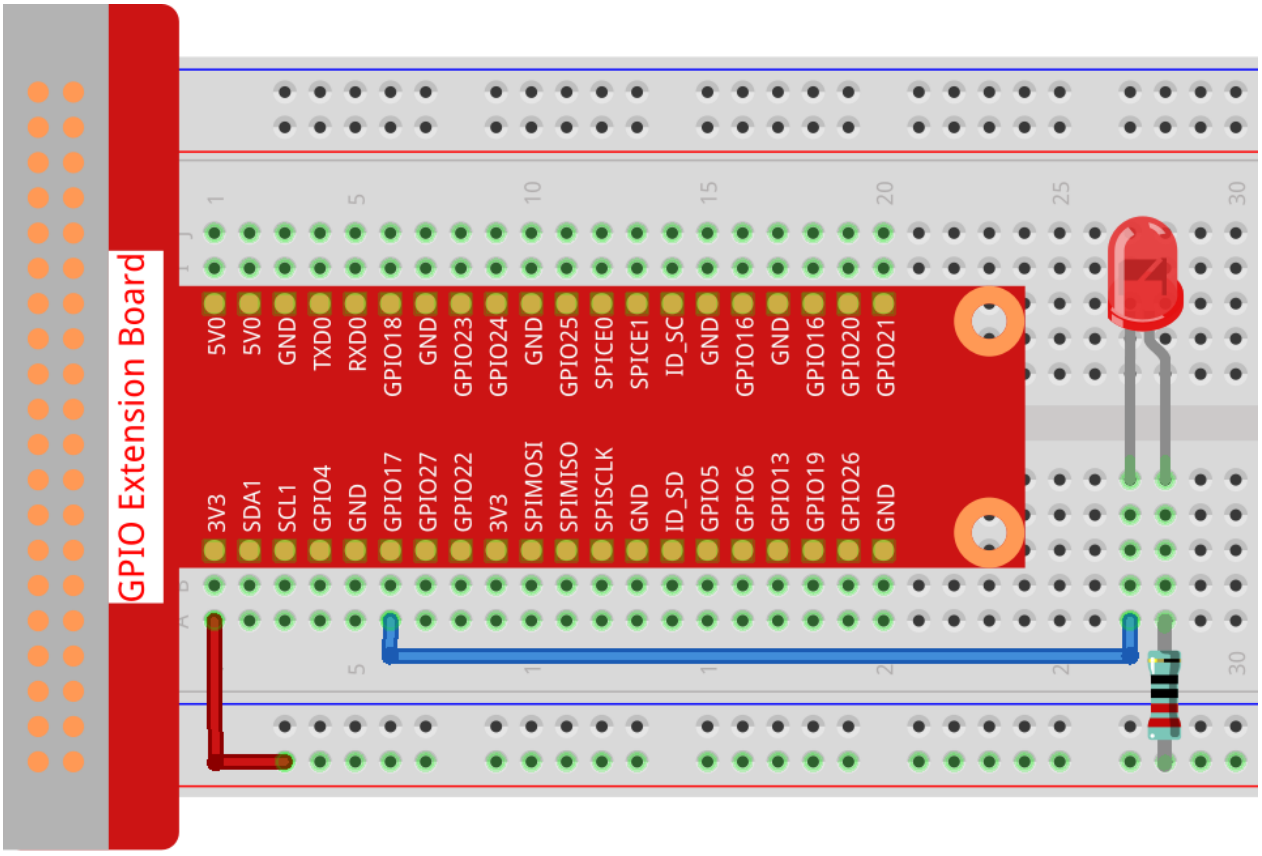
C 言語関連のコンテンツでは、wiringPi で GPIO0 を 0 と同等にする。Python 言語関連のコンテンツの中で、BCM 17 は次の表の BCM 列の 17 である。同時に、それらは Raspberry Pi の 11 番目のピン-ピン 11 と同じである。

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ 1: 回路を作る。



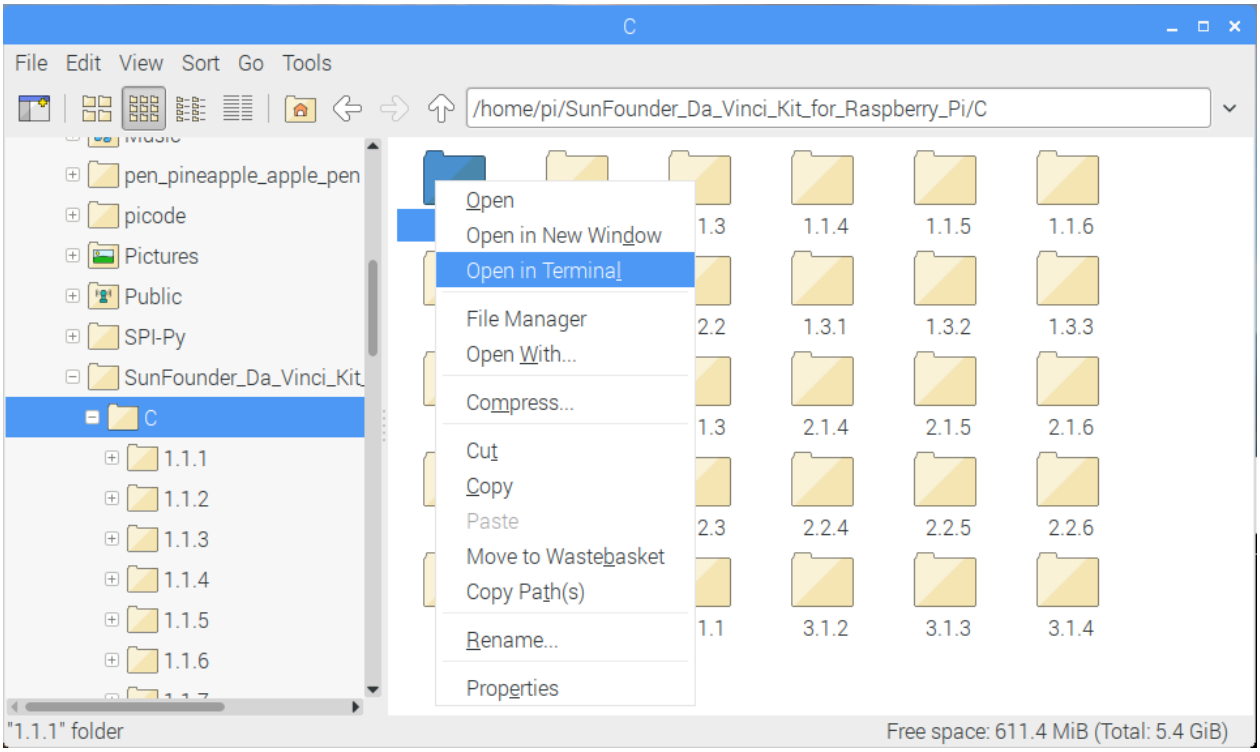
C 言語ユーザー向け

ステップ 2: コードのフォルダーに入る。

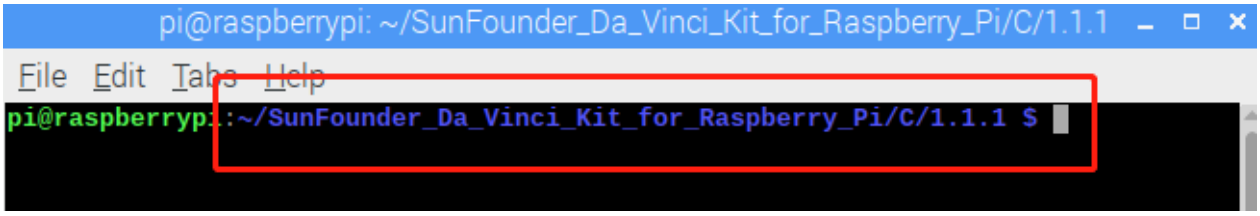
1) モニターを使用する場合は、次の手順を実行することをお勧めする。

/home/pi/ に入り、 davinci-kit-for-raspberry-pi フォルダーを見つけてください。

フォルダー内で **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。

2) 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/
```

注釈: この実験では、cd を使用してディレクトリをコードのパスに変更します。

どちらにしても、現在は C フォルダにいることになります。この 2 つの方法に基づくその後の手順は同じです。次に進みましょう。

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
```

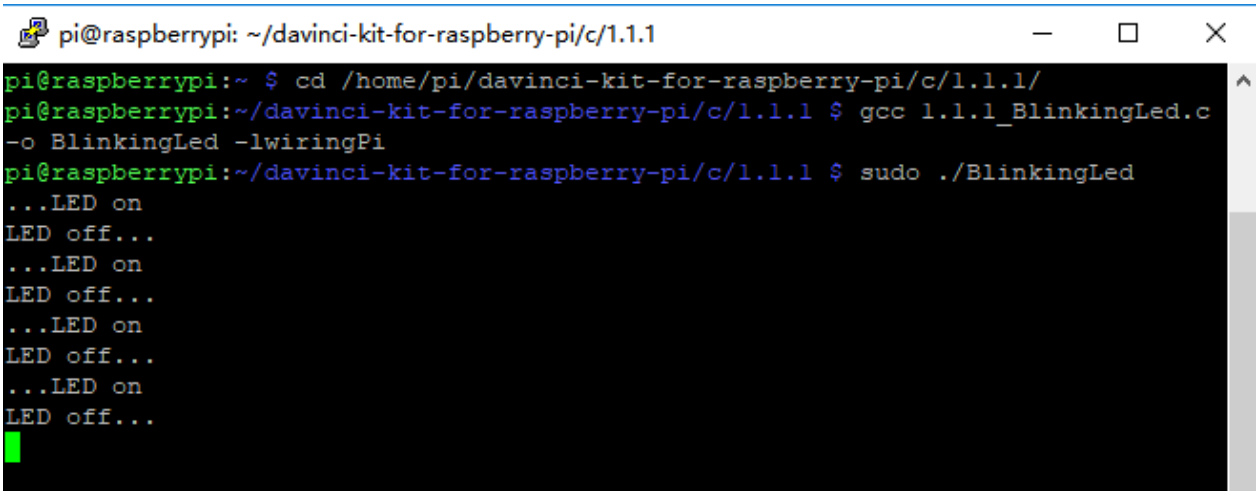
注釈: gcc は GNU Compiler Collection です。ここでは、C 言語ファイル*1_BlinkingLed.c*をコンパイルし、実行ファイルを出力するような機能を持っています。

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
```

注釈: 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。



コードの実行後、LED が点滅する。

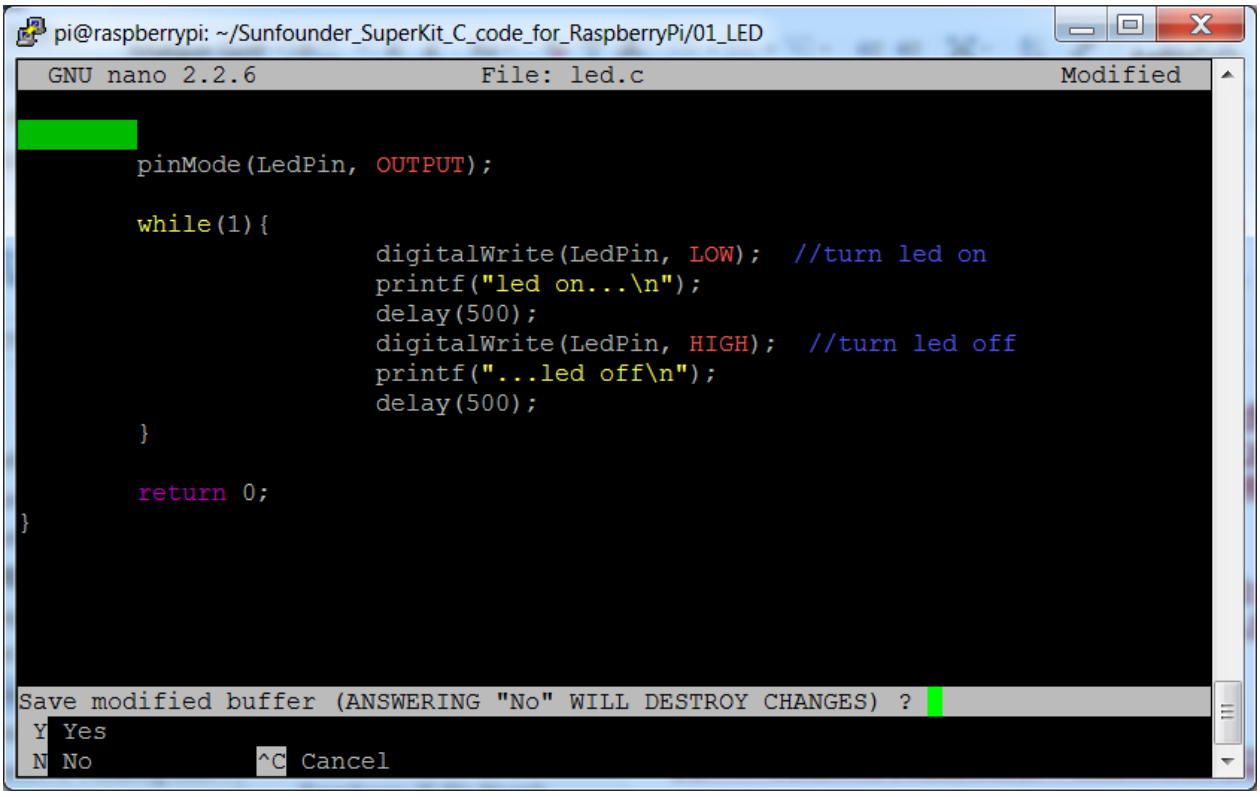
注釈: 実行しても動作しない場合、または「wiringPi.h: No such file or directory」というエラー プロンプトが表示される場合は、*C* **コードが機能していませんか？** を参照してください。

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
```

注釈: 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 pintf 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 SURU。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^^c2^^a0off\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」にあります。呼び出しの一般的な形式は次のとおりです。printf（"フォーマット制御文字列"、出力テーブルの列）。フォーマット制御文字列は、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出力フォーマットを指定するために使用されます。フォーマット文字列は「%」で始まり、10 進整数出力の場合は「% d」などのフォーマット文字が続きます。フォーマットされていない文字列はプロトタイプとして印刷されます。ここで使用されているのは、フォーマットされていない文字列であり、その後に改行文字である「\n」が続きます。これは、文字列を印刷した後の自動行折り返しを表します。

```
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、メイン関数の後ろに配置され、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。

1. 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。

2. 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
```

注釈: 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
```

注釈: 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
```

注釈: 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 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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。

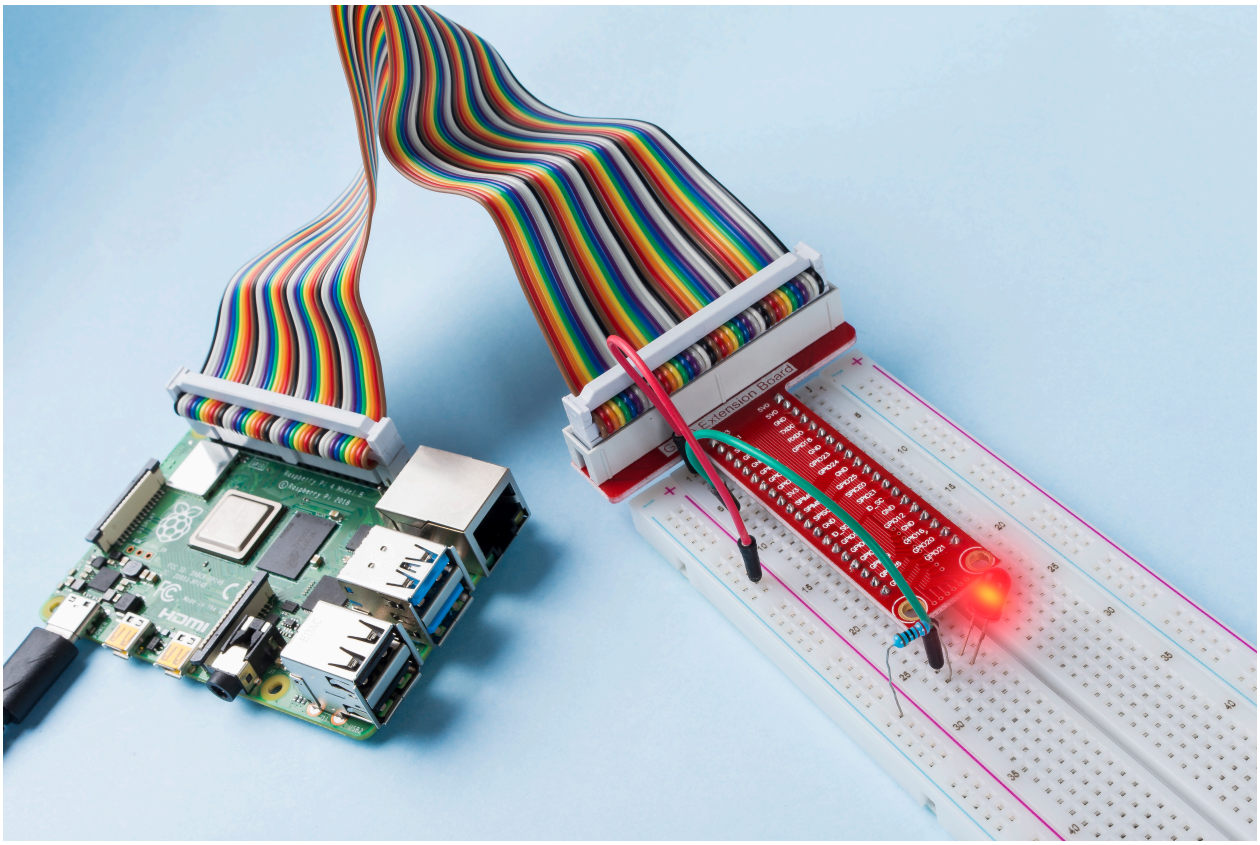
```
def destroy():
    GPIO.cleanup()
```

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

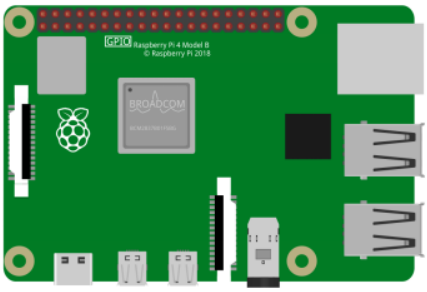
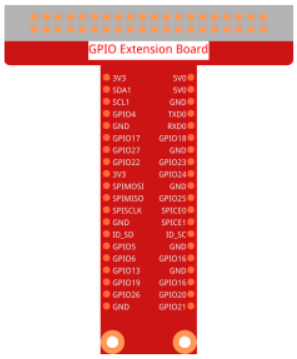
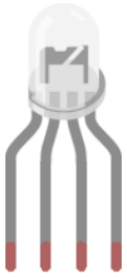
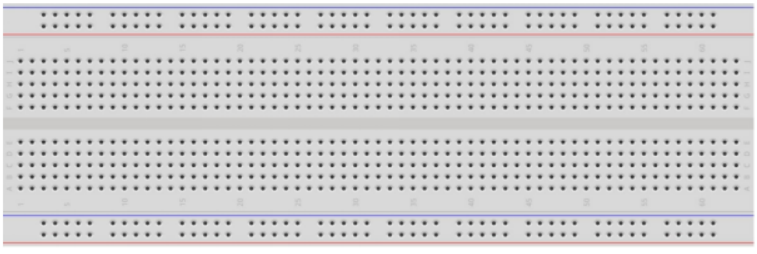


1.8.2 1.1.2 RGB LED

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

1 * Raspberry Pi 	1 * T-Extension Board 	1 * RGB LED 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Resistor(220Ω)

原理

PWM

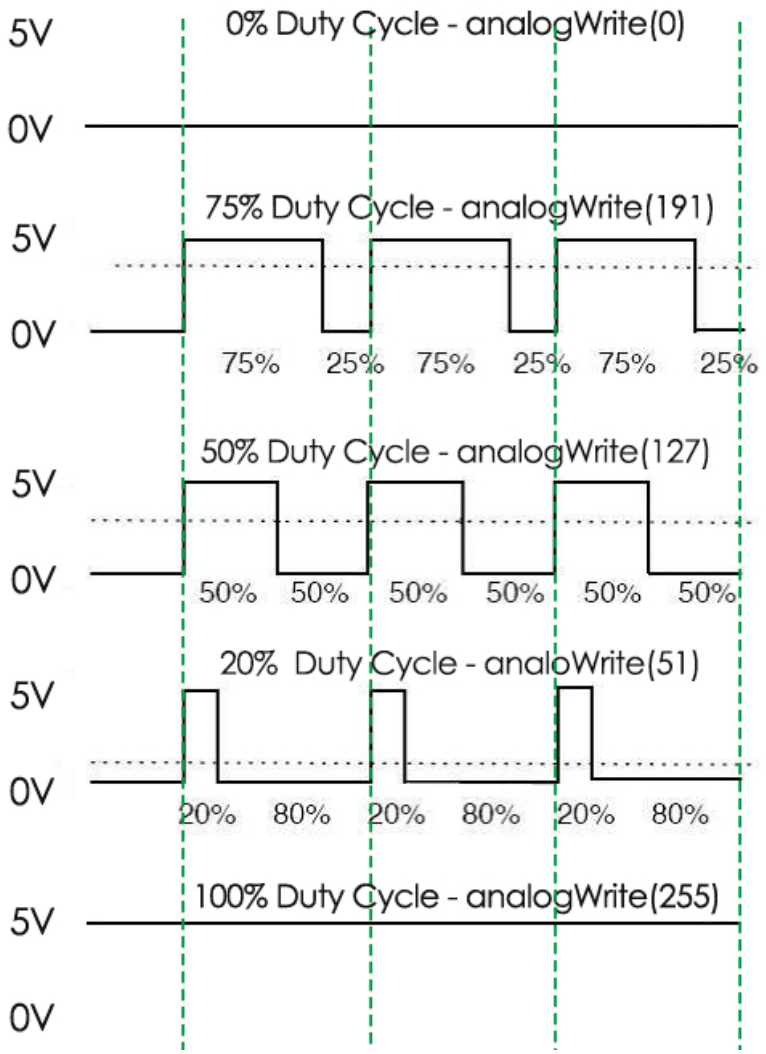
パルス幅変調（PWM）は、デジタル手段でアナログ結果を取得するための技術である。デジタル制御は、オンとオフを切り替える信号である方形波を作成するために使用される。このオン/オフパターンは、信号がオンになる時間と信号がオフになる時間の部分を変更することにより、完全にオン（5 ボルト）である時と完全にオフ（0 ボルト）である時の間の電圧をシミュレートできる。「オンタイム」の期間はパルス幅と呼ばれる。さまざまなアナログ値を取得するには、その幅を変更または変調できる。このオン/オフパターンを何らかのデバイス、たとえば LED で十分に速く繰り返すと、結果は次のようになる：信号は LED の輝度を制御する 0〜5v の安定した電圧である。

デューティサイクル

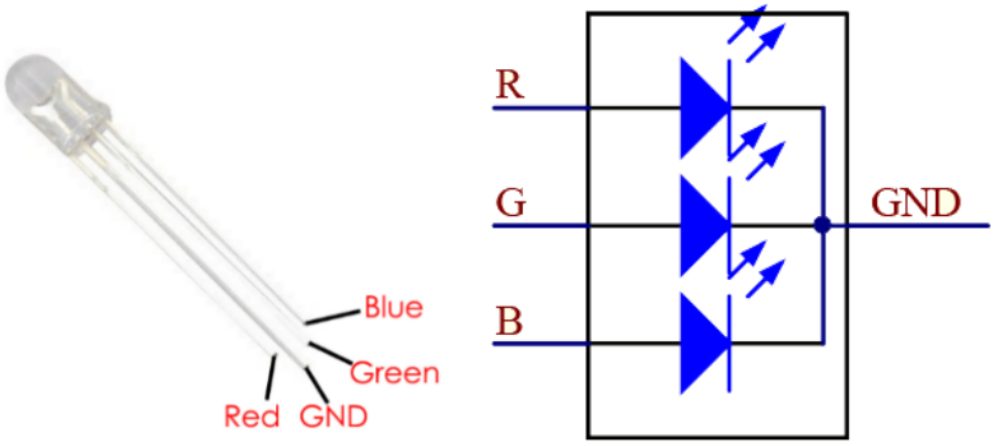
デューティサイクルは、信号が有効である 1 周期の割合である。期間とは、信号がオンとオフのサイクルを完了するのにかかる時間である。式として、デューティサイクルは次のように表示される：

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

ここで、 **D** はデューティサイクル、 **T** は信号がアクティブな時間、 **P** は信号の合計周期です。したがって、60 % のデューティサイクルは、信号が 60 %の時間オンであるが、40 %の時間オフであることを意味します。60 %のデューティサイクルの「定刻」は、期間の長さに応じて、数分の 1 秒、1 日、または 1 週間になることもあります。



RGB LED



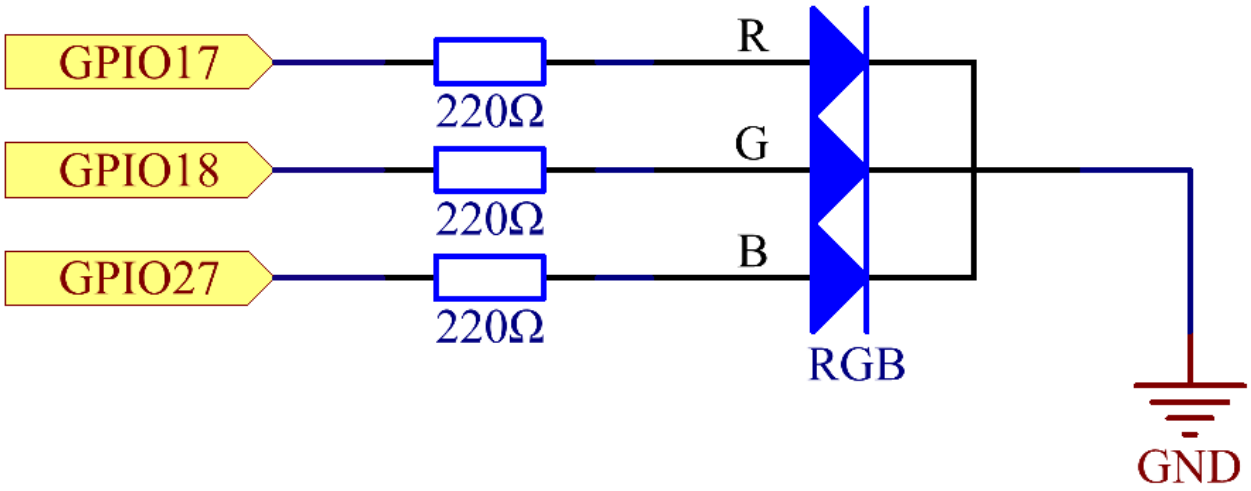
RGB LED の 3 つの原色は、輝度によってさまざまな色に混ぜることができる。LED の輝度は PWM で調整可能である。Raspberry Pi にはハードウェア PWM 出力用のチャネルが 1 つしかないが、RGB LED を制御するには 3

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

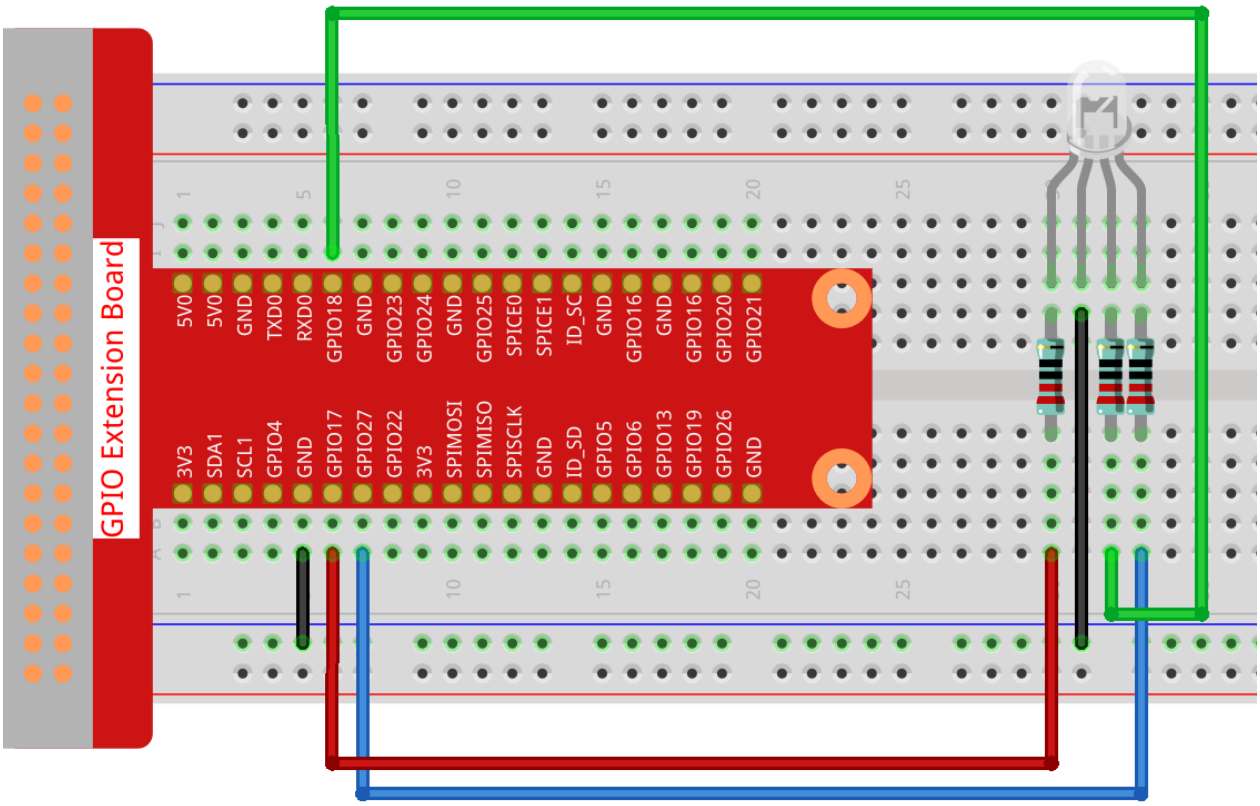
R、G、および B のピンを電流制限抵抗器に接続した後、それぞれ GPIO17、GPIO18、および GPIO27 に接続してください。LED の最も長いピン（GND）は、Raspberry Pi の接地に接続する。3 つのピンに異なる PWM 値が与えられると、RGB LED は異なる色を表示する。

T-Board Name	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
```

注釈: 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 messageto 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");
```

(次のページに続く)

(前のページからの続き)

```
        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);
```

- **pin:** ラズベリーパイの任意の GPIO ピンは、PWM ピンとして設定することができます。
- **initialValue:** 初期パルス幅は initialValue に 100us を掛けたものである。
- **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) ;
```

- **pin:** ラズベリーパイの任意の GPIO ピンは、PWM ピンとして設定することができます。
- **Value:** PWM のパルス幅は、値の 100us の倍数です。値は、以前に定義された pwmRange より小さく、pwmRange より大きい場合、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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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
```

(次のページに続く)

(前のページからの続き)

```
pins = {'Red':17, 'Green':18, 'Blue':27}


def setup():
    global p_R, p_G, p_B
    # 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)
    for i in pins:
        GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.HIGH)

    # Set all led as pwm channel and frequece to 2KHz
    p_R = GPIO.PWM(pins['Red'], 2000)
    p_G = GPIO.PWM(pins['Green'], 2000)
    p_B = GPIO.PWM(pins['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
# input color should be Hexadecimal with
# red value, blue value, green value.
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
# these three lines are used for analyzing the col variables
# assign the first two values of the hexadecimal to R, the middle two assigned to G
# assign the last two values to B, please refer to the shift operation of the␣
↪hexadecimal for details.
```

(次のページに続く)

(前のページからの続き)

```
    # 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)
    # Assign the mapped duty cycle value to the corresponding PWM channel to change the␣
↪luminance.
    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:# Assign every item in the COLOR list to the color␣
↪respectively and change the color of the RGB LED via the setColor() function.
            setColor(color)# change the color of the RGB LED
            time.sleep(0.5)# set delay for 0.5s after each color changing. Modify this␣
↪parameter will changed the LED's color changing rate.

def destroy():
    # Stop all pwm channel
    p_R.stop()
    p_G.stop()
    p_B.stop()
    # 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

```
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。

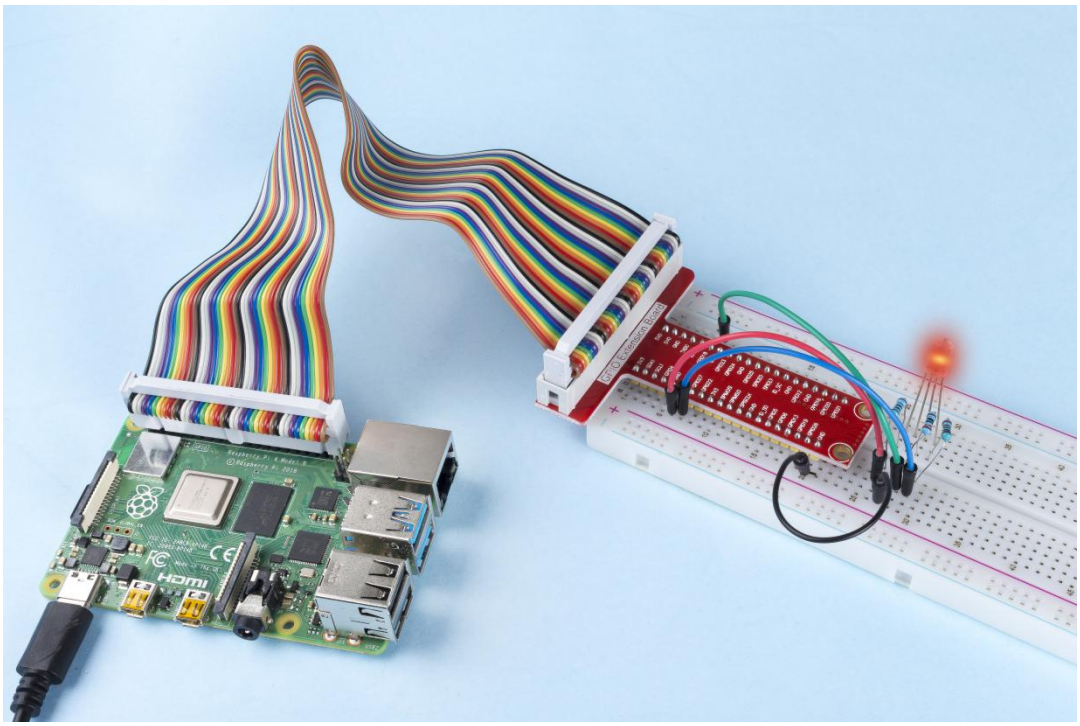
```
p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

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

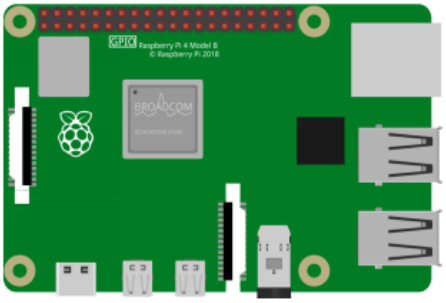
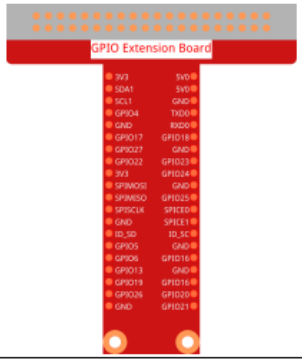
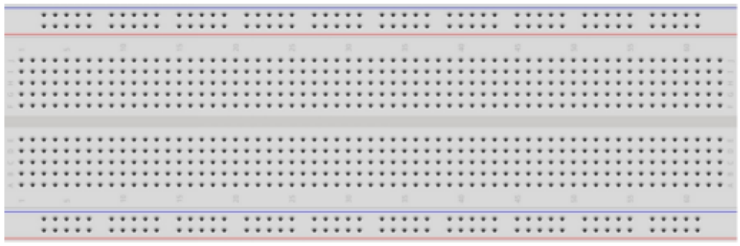


1.8.3 1.1.3 LED 棒グラフ

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。

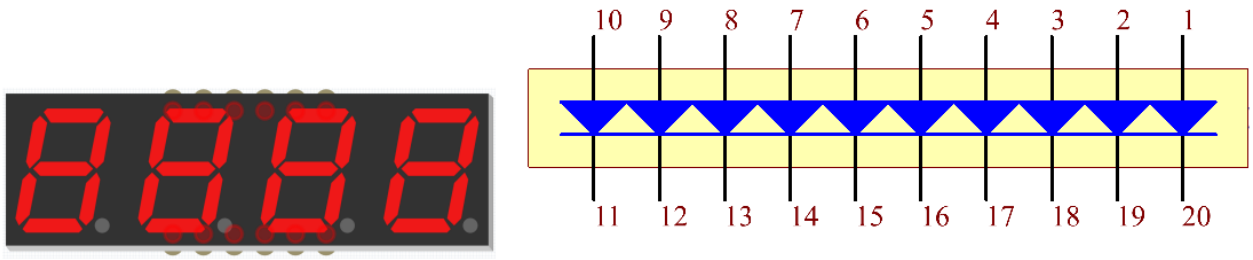
部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Bargraph
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		10 * Resistor(220Ω)

原理

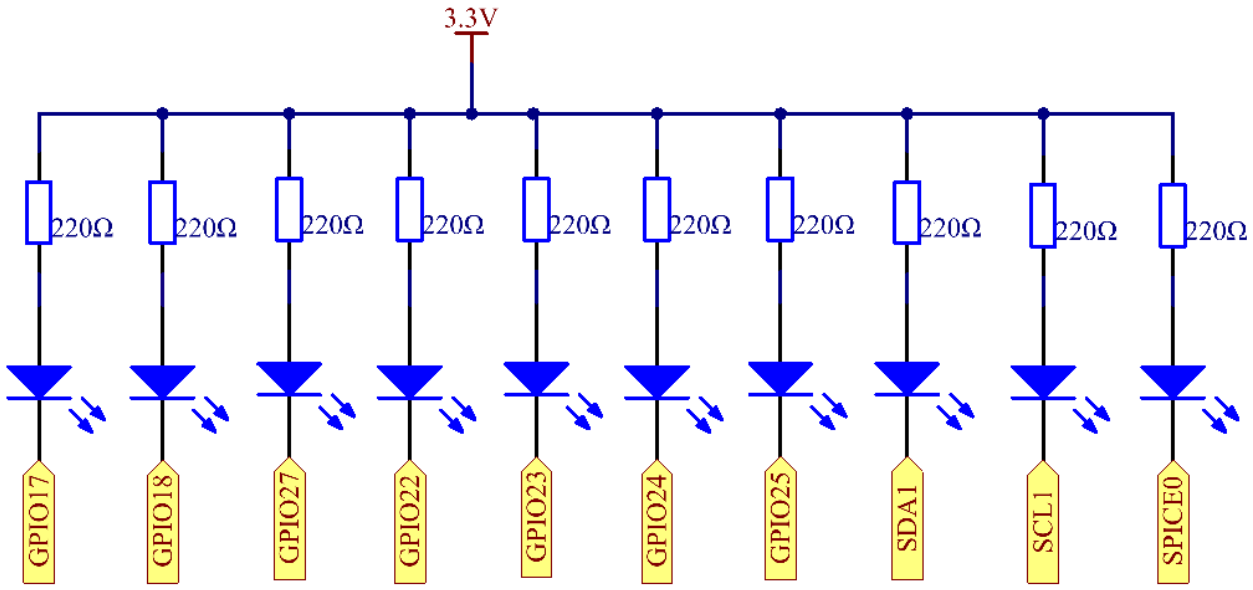
LED 棒グラフ

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。



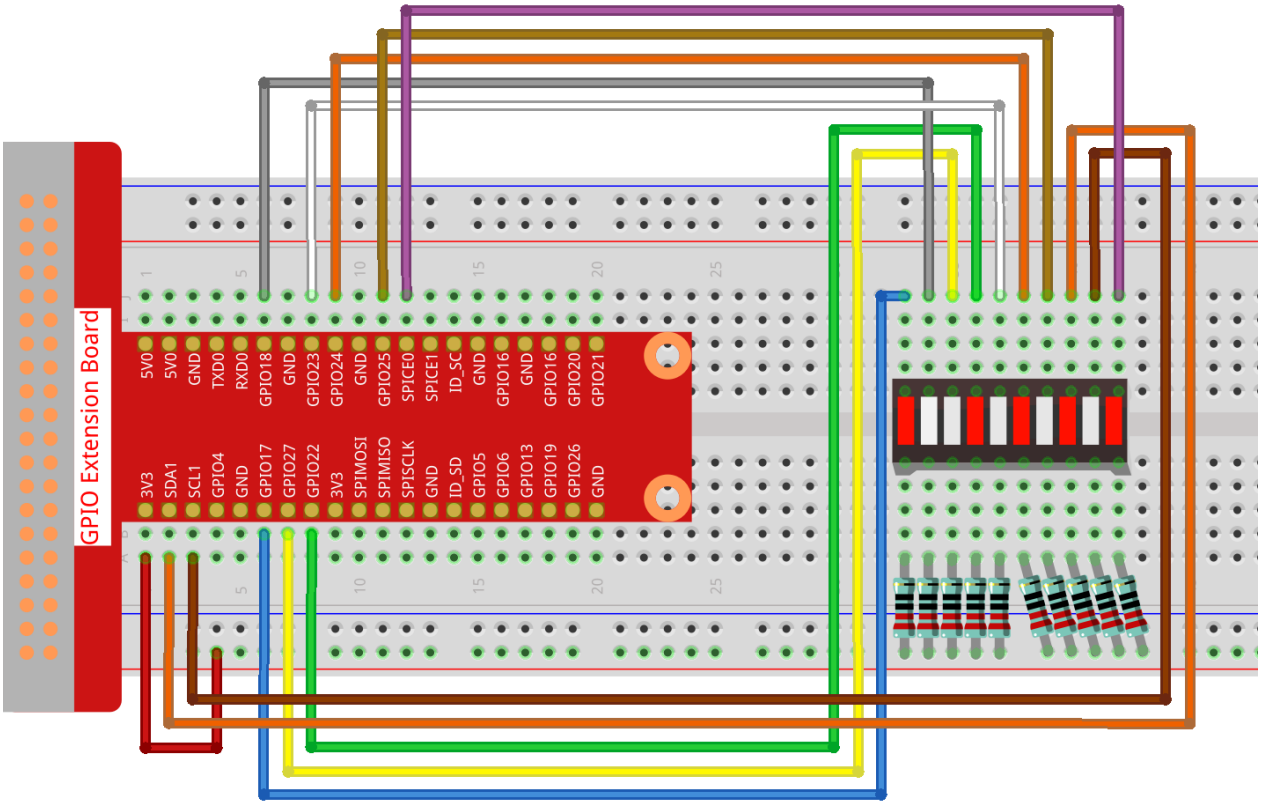
回路図

T-Board Name	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();
```

(次のページに続く)

(前のページからの続き)

```
        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++){
```

(次のページに続く)

(前のページからの続き)

```
        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: EXE ファイルを実行する。

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

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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):
```

(次のページに続く)

(前のページからの続き)

```
        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.BOARD)        # 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。

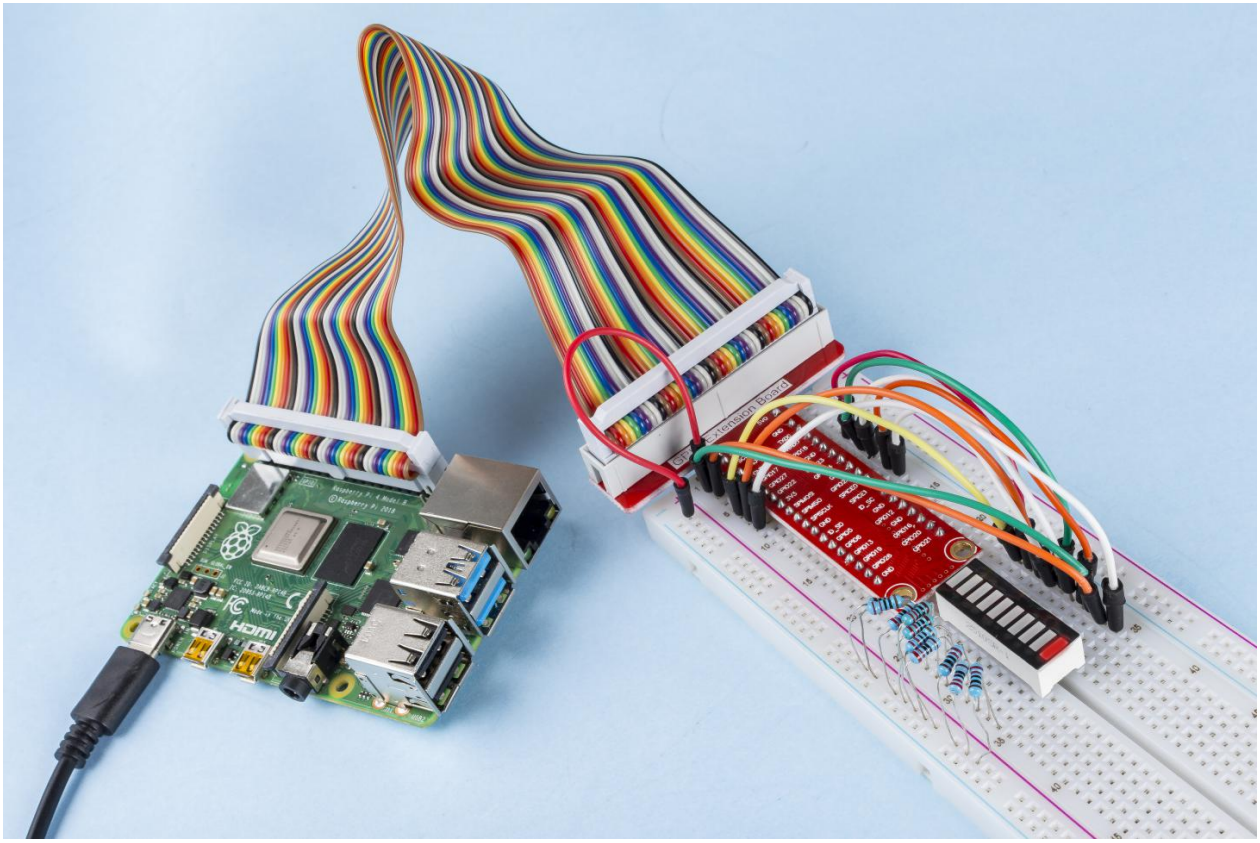
```
def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)
```

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

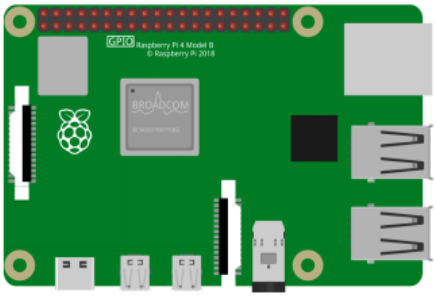
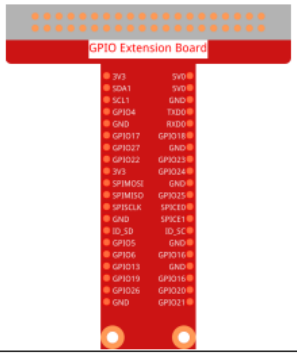
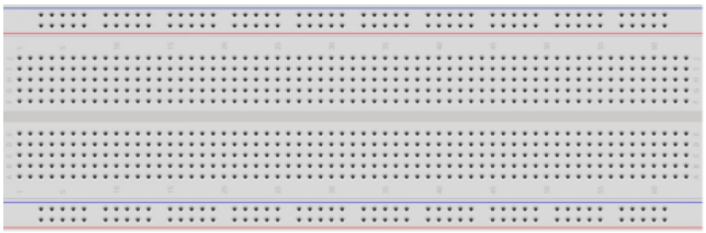


1.8.4 1.1.4 7 セグメントディスプレイ

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

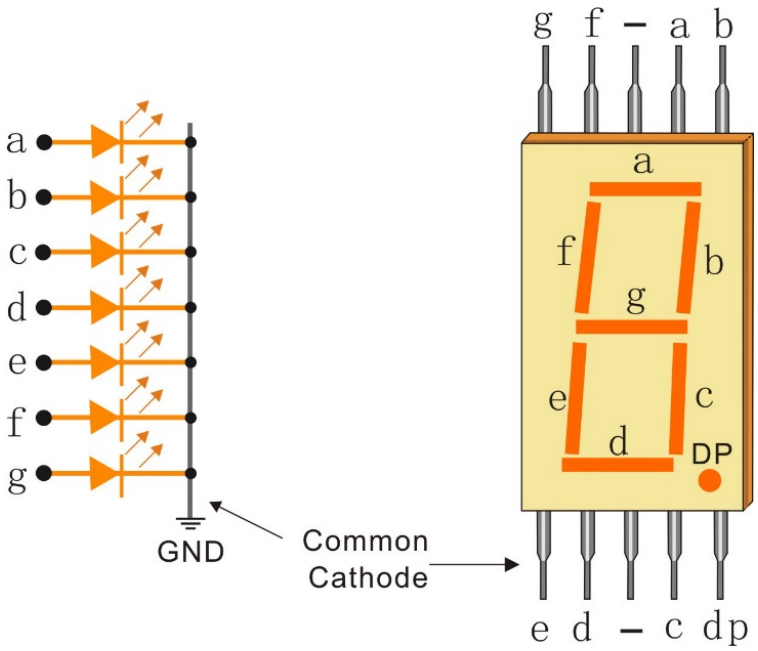
1 * Raspberry Pi 	1 * T-Extension Board 	1 * 7-segment display
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * 74HC595

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

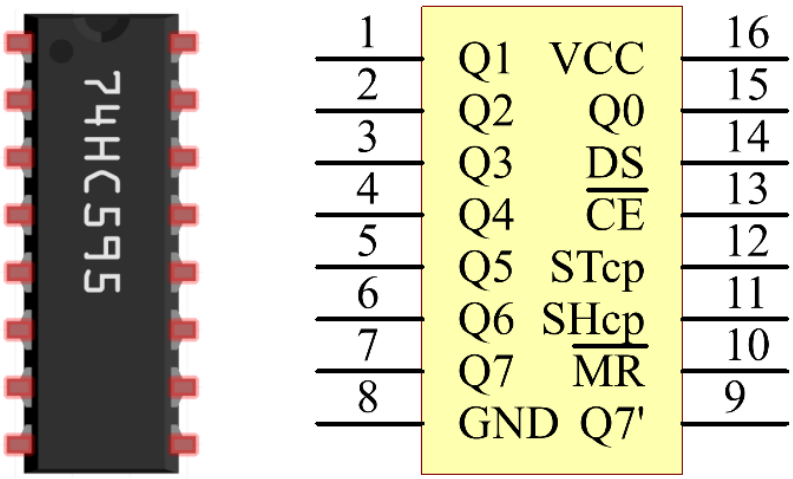
7 セグメントディスプレイ（カソードコモン）がどのように番号を表示するかを知るために、次の表を作成した。数字は、7 セグメントディスプレイに表示される数字 0～F である。（DP）GFEDCBA は、0 または 1 に設定された対応する LED を指す。たとえば、00111111 は、DP と G が 0 に設定され、他が 1 に設定されることを意味する。したがって、7 セグメントディスプレイには 0 が表示され、HEX コードは 16 進数に対応する。

Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
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。



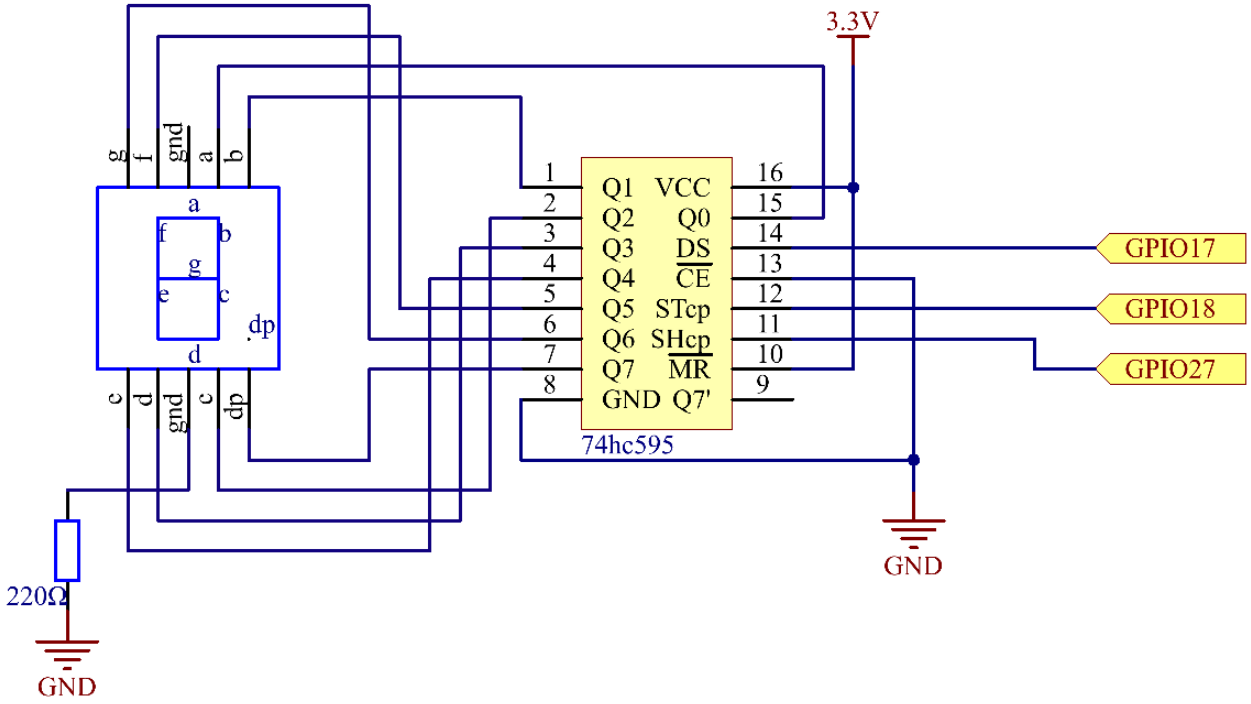
74HC595 のピンとその機能

- **Q0-Q7** : 8 ビットパラレルデータ出力端子。8 個の LED または 8 個の 7 セグメントディスプレイを直接制御することができます。
- **Q7'** : 複数の 74HC595 を直列に接続するために，別の 74HC595 の DS に接続される直列出力端子。
- **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は、シフトレジスタのデータが 1 ビット連続して移動します。つまり、Q1 のデータが Q2 に移動します。 立ち下がりエッジにある間、シフトレジスタのデータは変更されません。
- **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します。
- **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

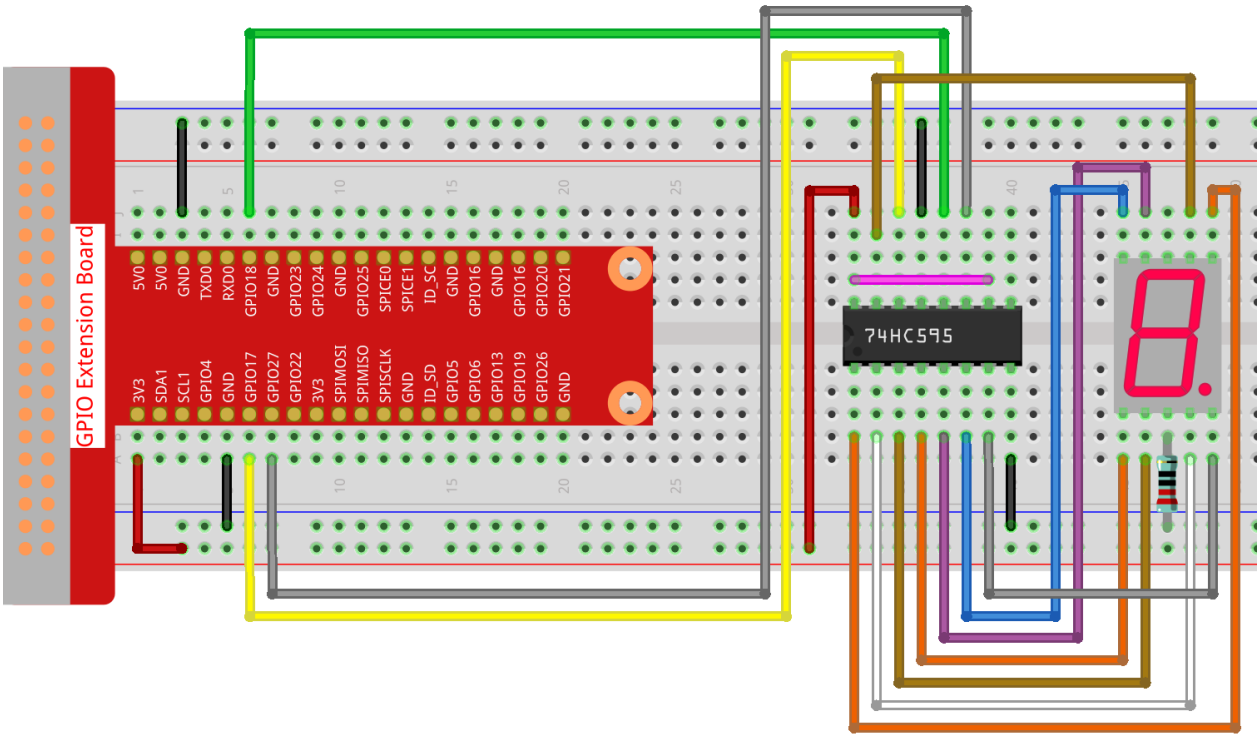
74HC595 のピン ST_CP を Raspberry Pi GPIO18 に、SH_CP を GPIO27 に、DS を GPIO17 に、並列出力ポートを LED セグメントディスプレイの 8 つのセグメントに接続する。SH_CP（シフトレジスタのクロック入力）が立ち上がりエッジにある場合は DS ピンのデータをシフトレジスタに入力し、ST_CP（メモリのクロック入力）が立ち上がりエッジにある場合はそのデータをメモリレジスタに入力する。次に、Raspberry Pi GPIO を介して SH_CP および ST_CP の状態を制御し、直列データ入力を並列データ出力に変換して、Raspberry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

(次のページに続く)

(前のページからの続き)

```
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 messageto 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);
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
    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);
}
```

Set 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);
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
```

この for ループでは、i を 16 進数で出力するために、"%1X" を使用しています。i を適用して SegCode[] 配列の対応するセグメントコードを求め、 hc595_shift() で 74HC595 のシフトレジスターに SegCode を渡します。

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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
# Common cathode
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)
```

(次のページに続く)

(前のページからの続き)

```
    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)
```

Set 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)
```

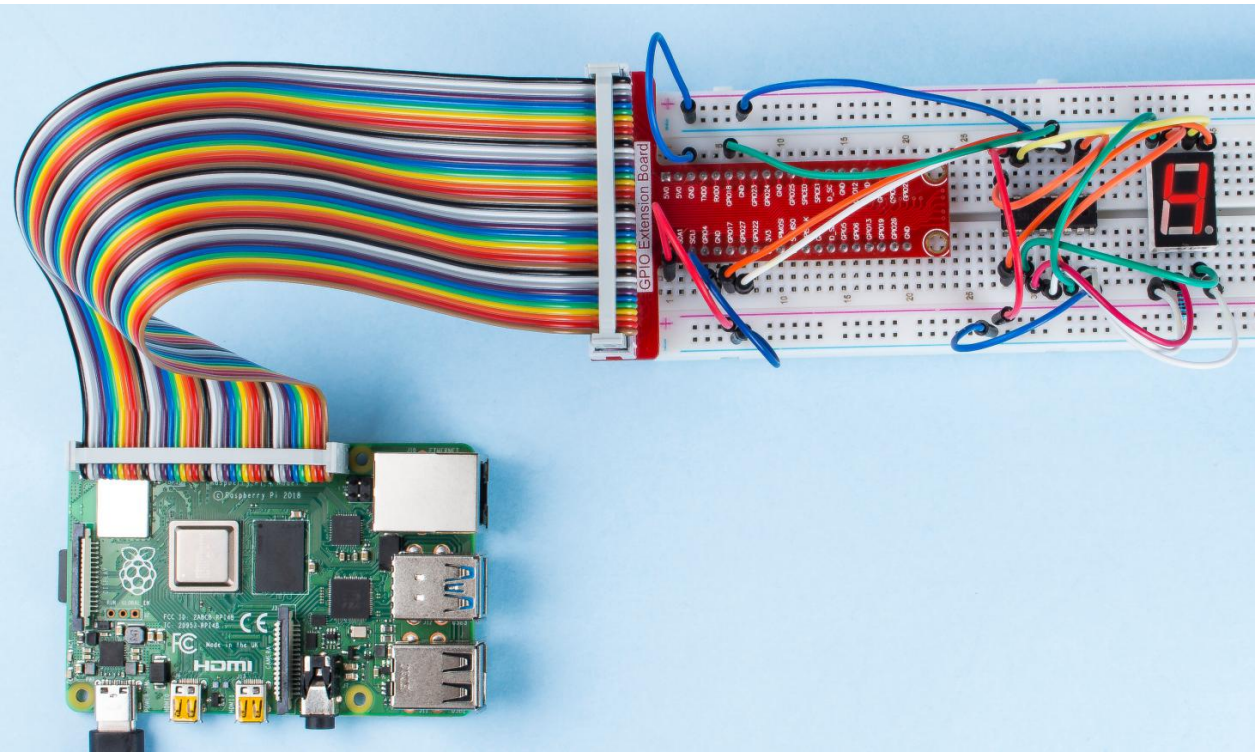
SRCLK の初期値は元々 LOW に設定されていたが、ここでは立ち上がりエッジを生成し、DS データをシフトレジスタに切り替えるために HIGH に設定されている。

```
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。

注釈: 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

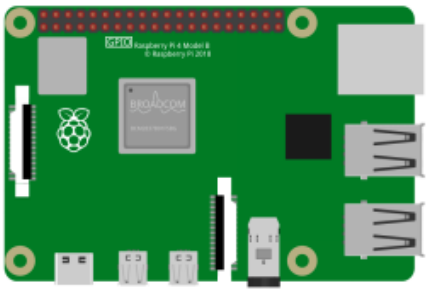
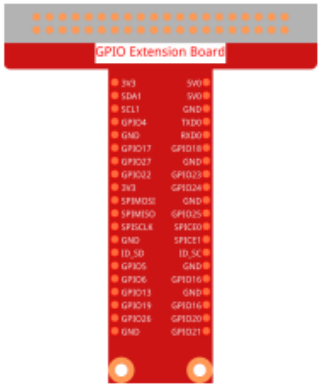
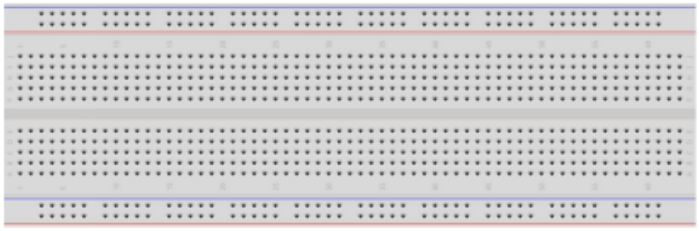


1.8.5 1.1.5 4 桁 7 セグメントディスプレイ

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。

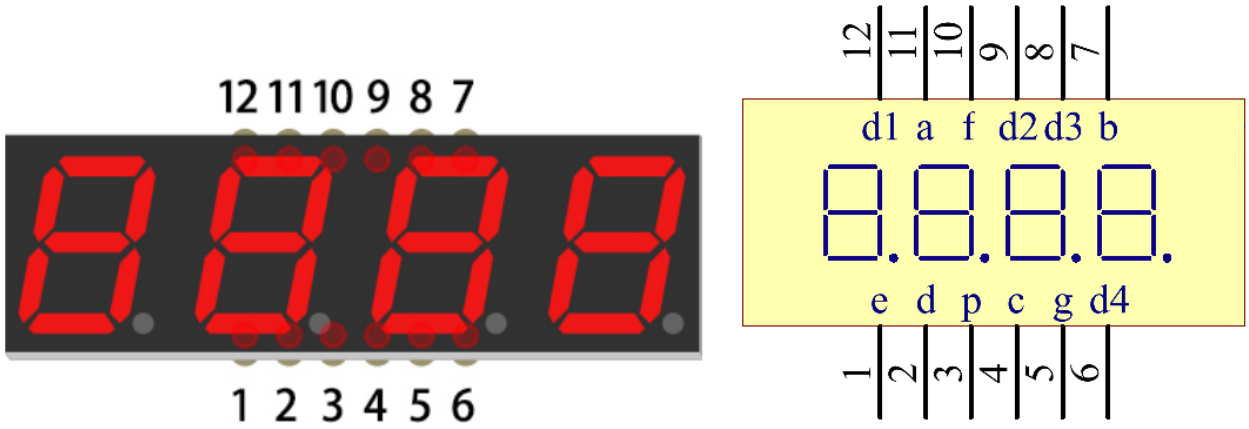
部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display
		1 * 74HC595
1 * 40-pin Cable		4 * Resistor(220Ω)
1 * Breadboard 		Several Jumper Wires

原理

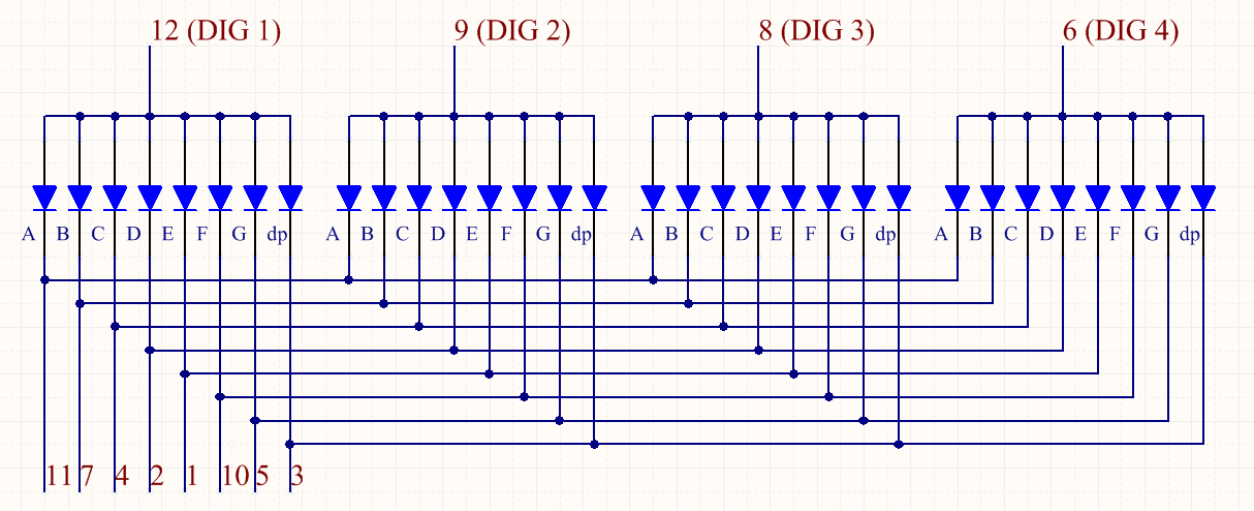
4 桁 7 セグメントディスプレイ

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。



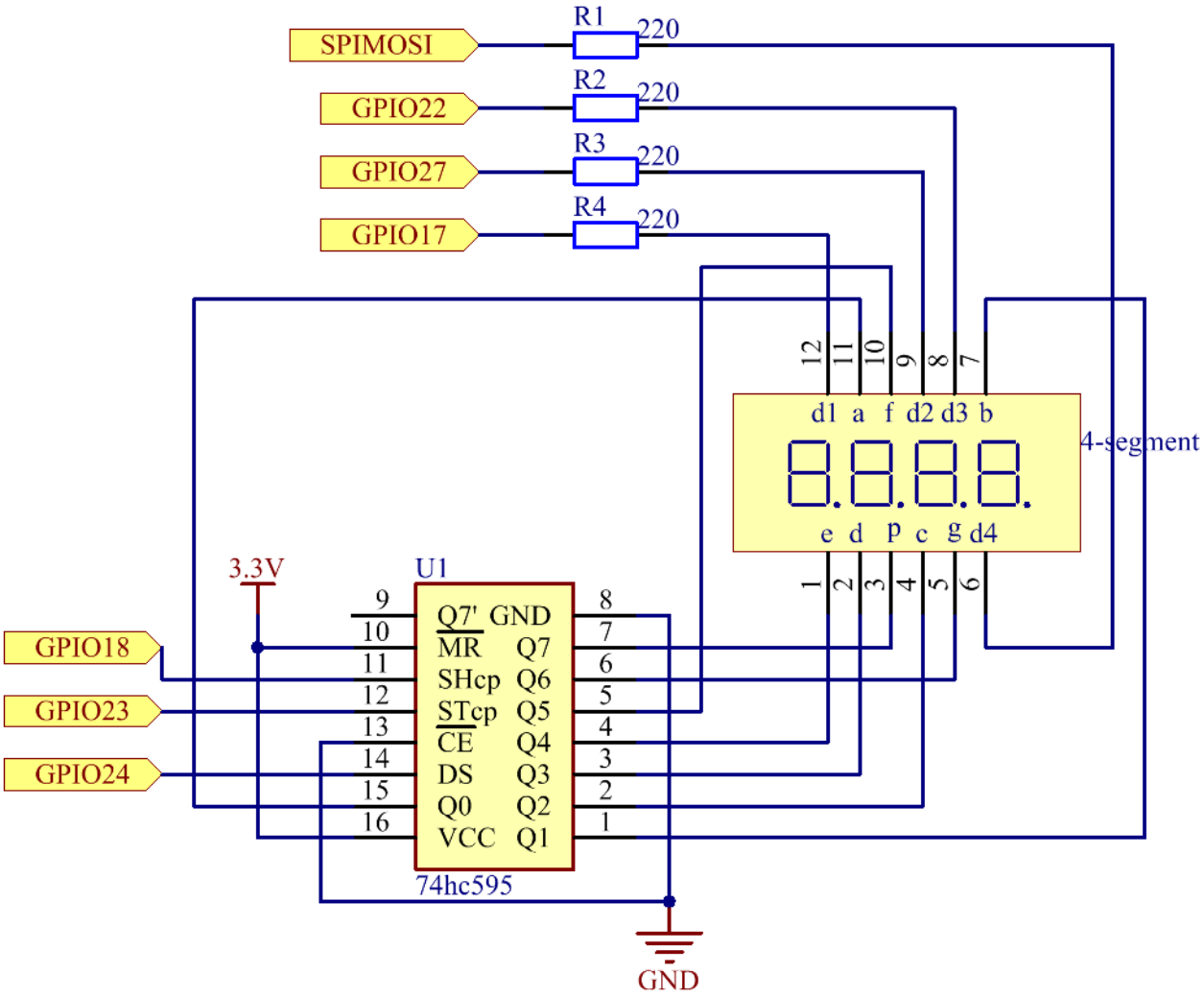
表示コード

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)GFEDCBA	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			

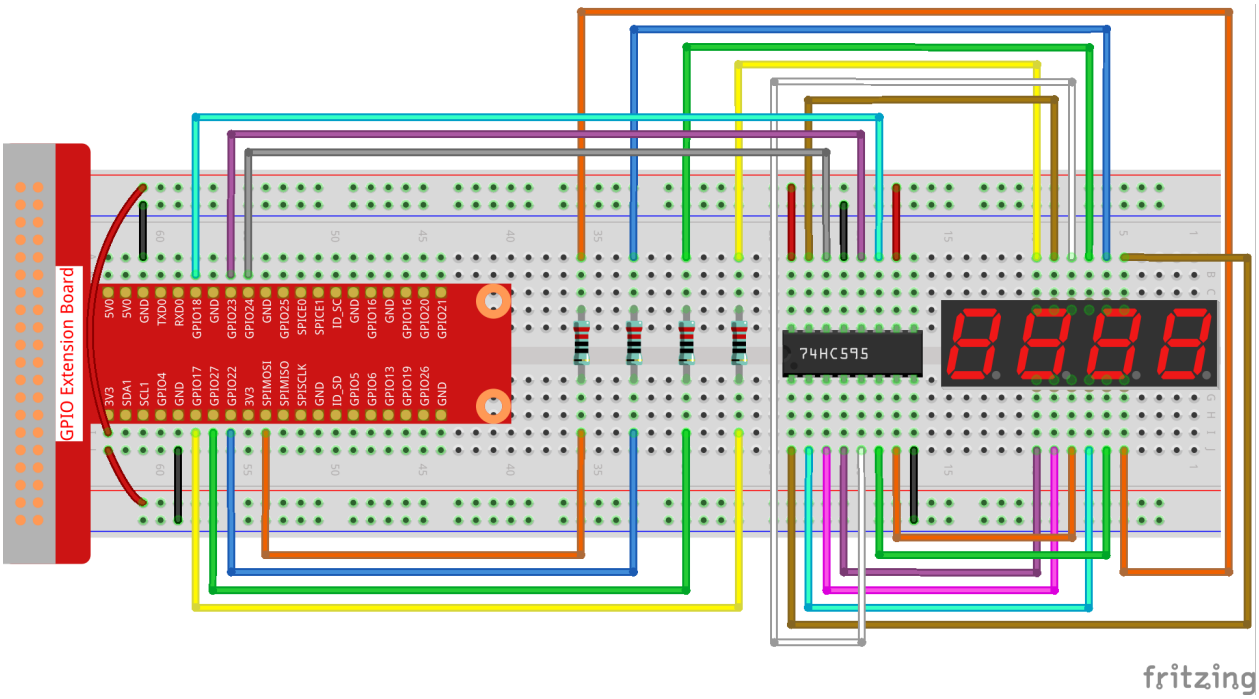
回路図

T-Board Name	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



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>
```

(次のページに続く)

(前のページからの続き)

```
#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++)
    {
```

(次のページに続く)

(前のページからの続き)

```
        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);
    }
```

(次のページに続く)

(前のページからの続き)

```
}

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);
```

(次のページに続く)

(前のページからの続き)

```
}
```

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);
```

これはシステム組み込みの関数であり、コードのプロトタイプは次のとおりである：

```
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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

(次のページに続く)

(前のページからの続き)

```
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()
```

(次のページに続く)

(前のページからの続き)

```
    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
```

(次のページに続く)

(前のページからの続き)

```
if __name__ == '__main__':  # Program starting from here
    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()
```

(次のページに続く)

(前のページからの続き)

```
        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 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 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。

```
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={})
```

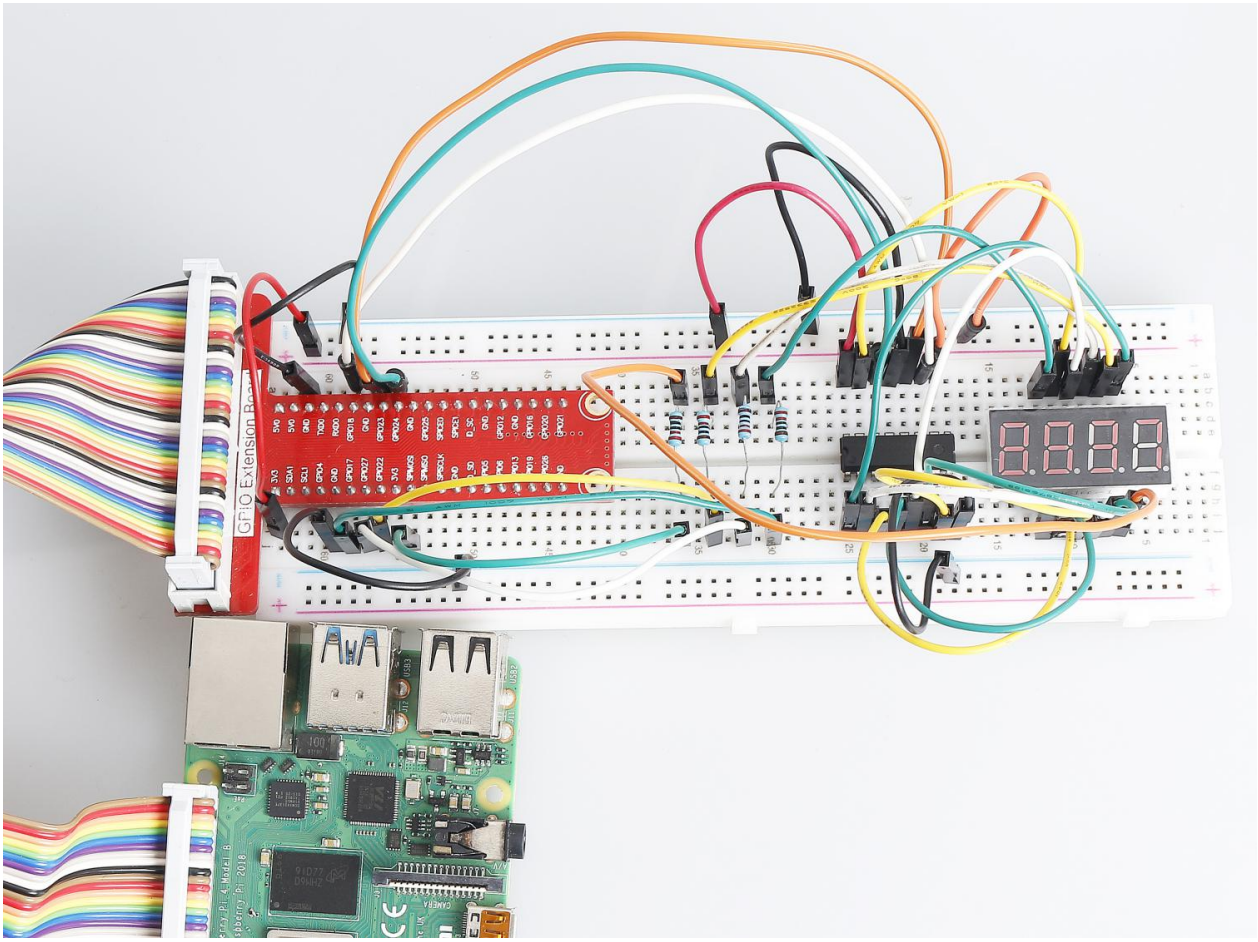
間隔の後、関数が実行される。ここでは、間隔は 1.0、関数は `timer()` である。 `start()` は、タイマーがこの時点で開始することを意味する。

```
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

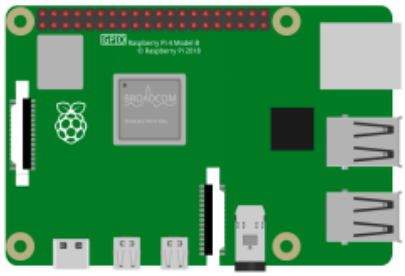
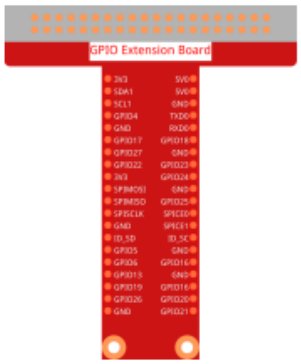
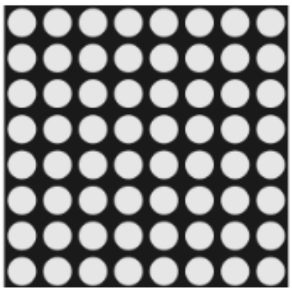
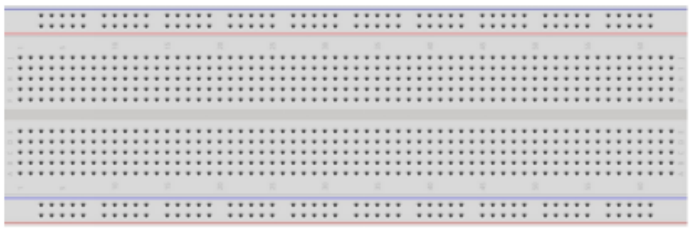


1.8.6 1.1.6 LED ドットマトリックス

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 の点灯と調光は、さまざまな文字とパターンを形成する。

部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Dot Matrix 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		2 * 74HC595

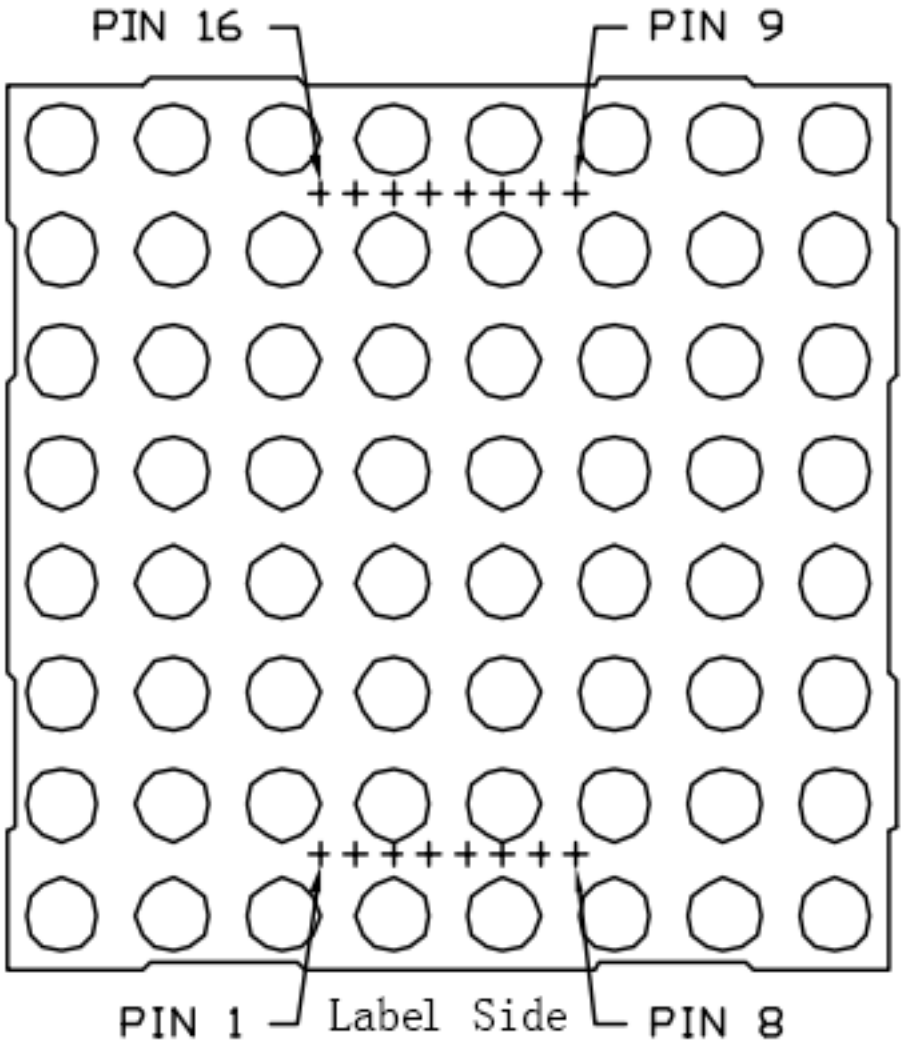
原理

LED ドットマトリックス

一般に、LED ドットマトリックスは、cathode (CC) と common anode (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。

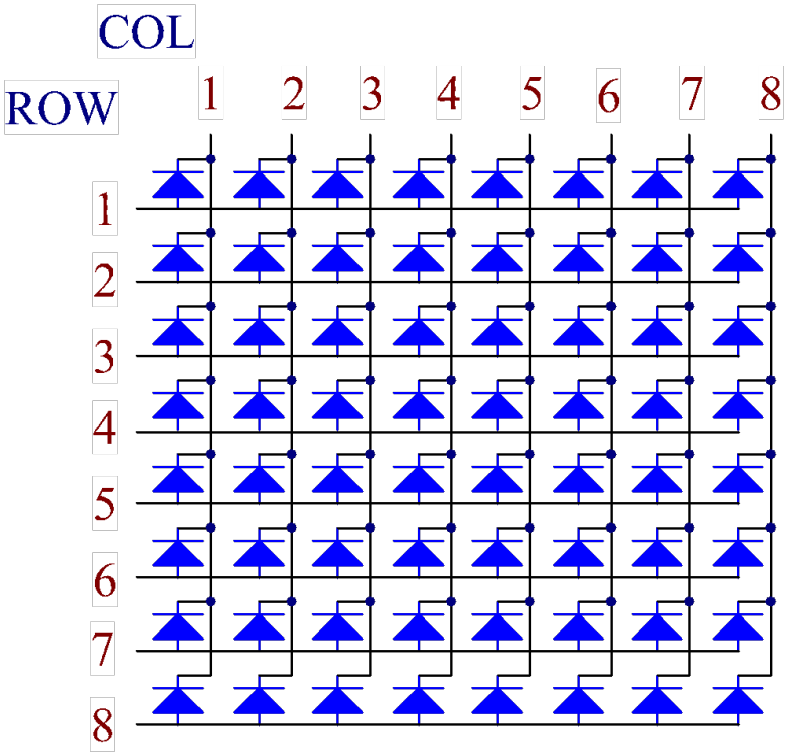
下の図を参照してください。ピンは背面の両端に配置されている。ラベル側を参考にしてください。この端のピンはピン 1〜8 で、もう一方のピンはピン 9〜16 である。

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：



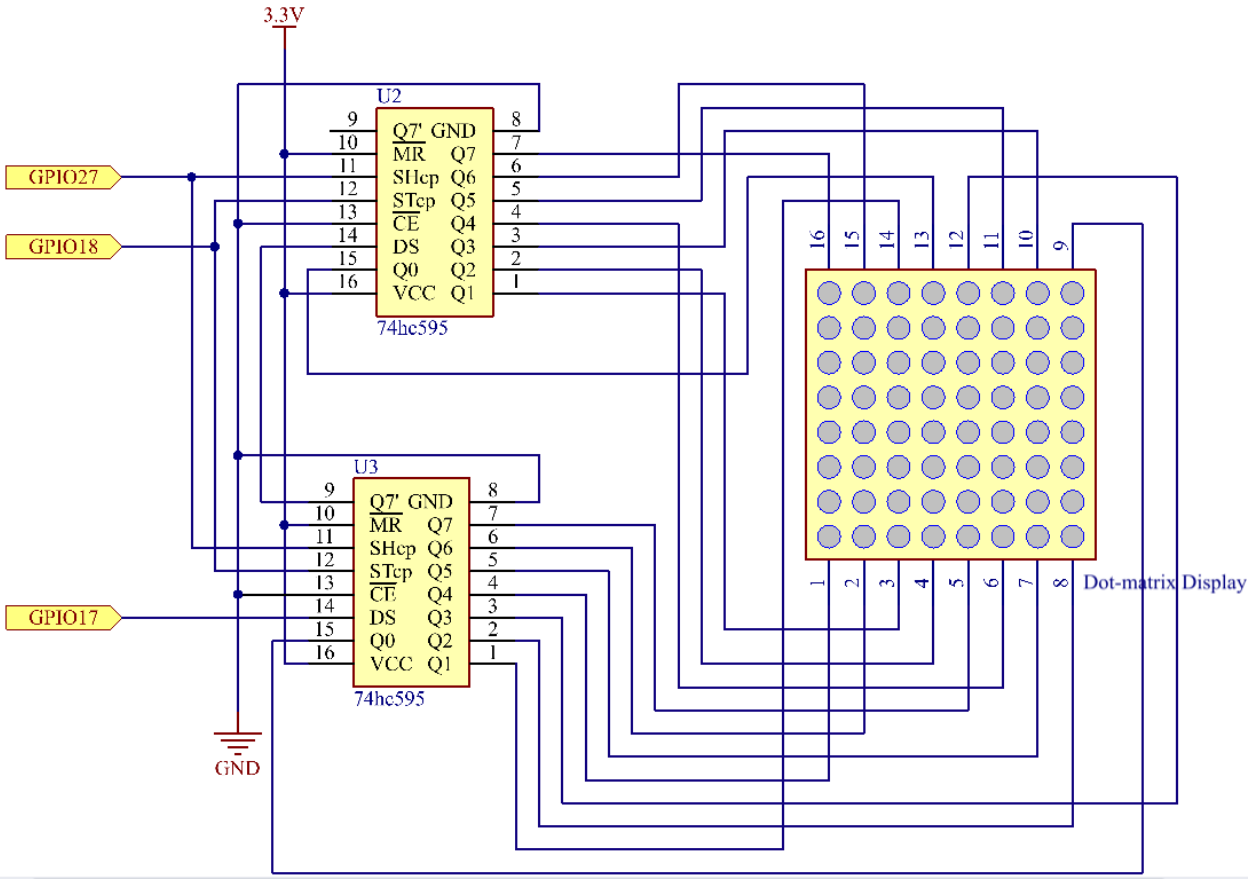
上記の行と列に対応するピン番号：

COL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**13**	**3**	**4**	**10**	**6**	**11**	**15**	**16**
ROW	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**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

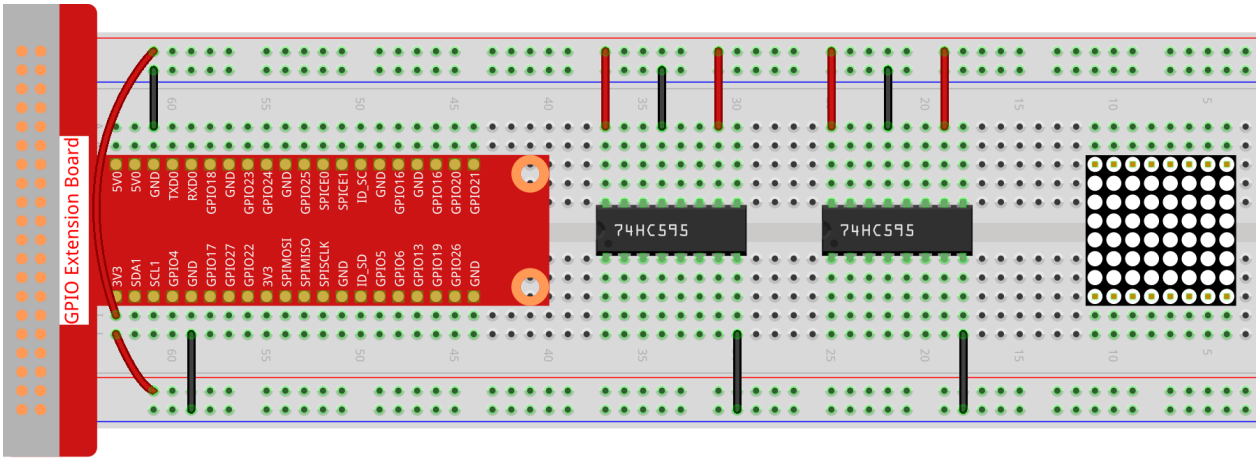
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



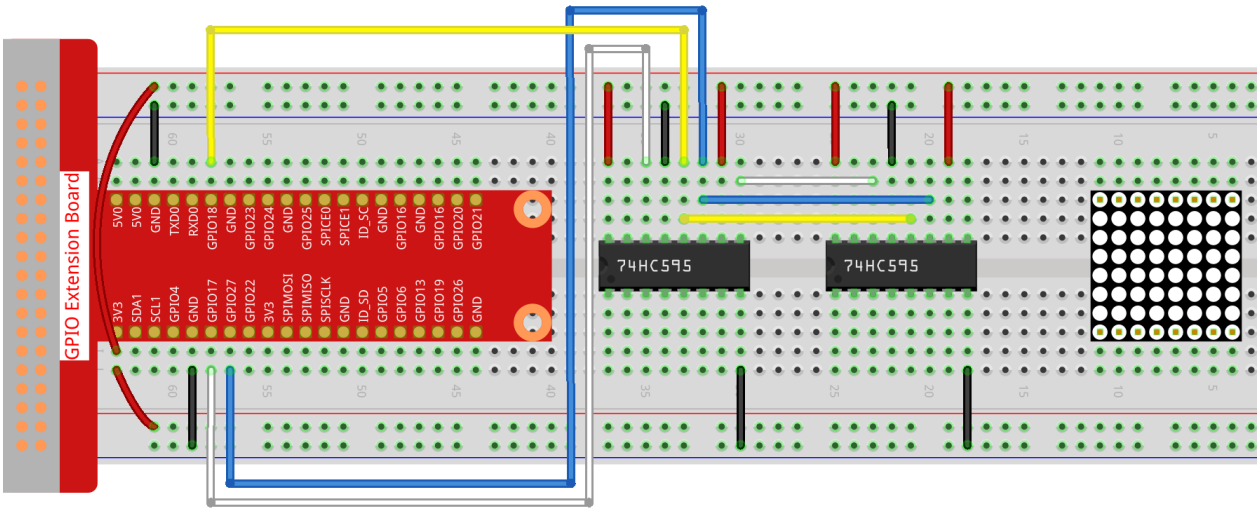
実験手順

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。

注釈: 上の Fritzing 画像では、ラベルのある側が下にある。

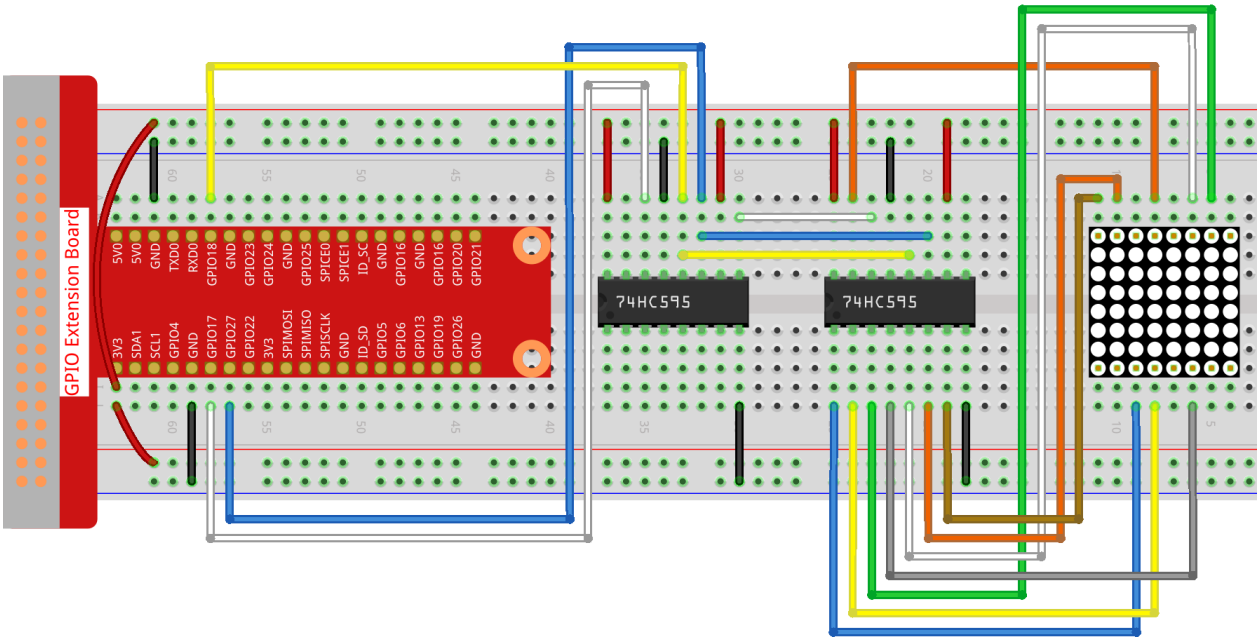


ステップ 2: 2 つの 74HC595 のピン 11 を一緒に接続し、GPIO27 に接続する。次に、2 つのチップのピン 12 を一緒に接続し、GPIO18 に接続する。それから左側の 74HC595 のピン 14 を GPIO17 に、ピン 9 を 2 番目の 74HC595 のピン 14 に接続する。



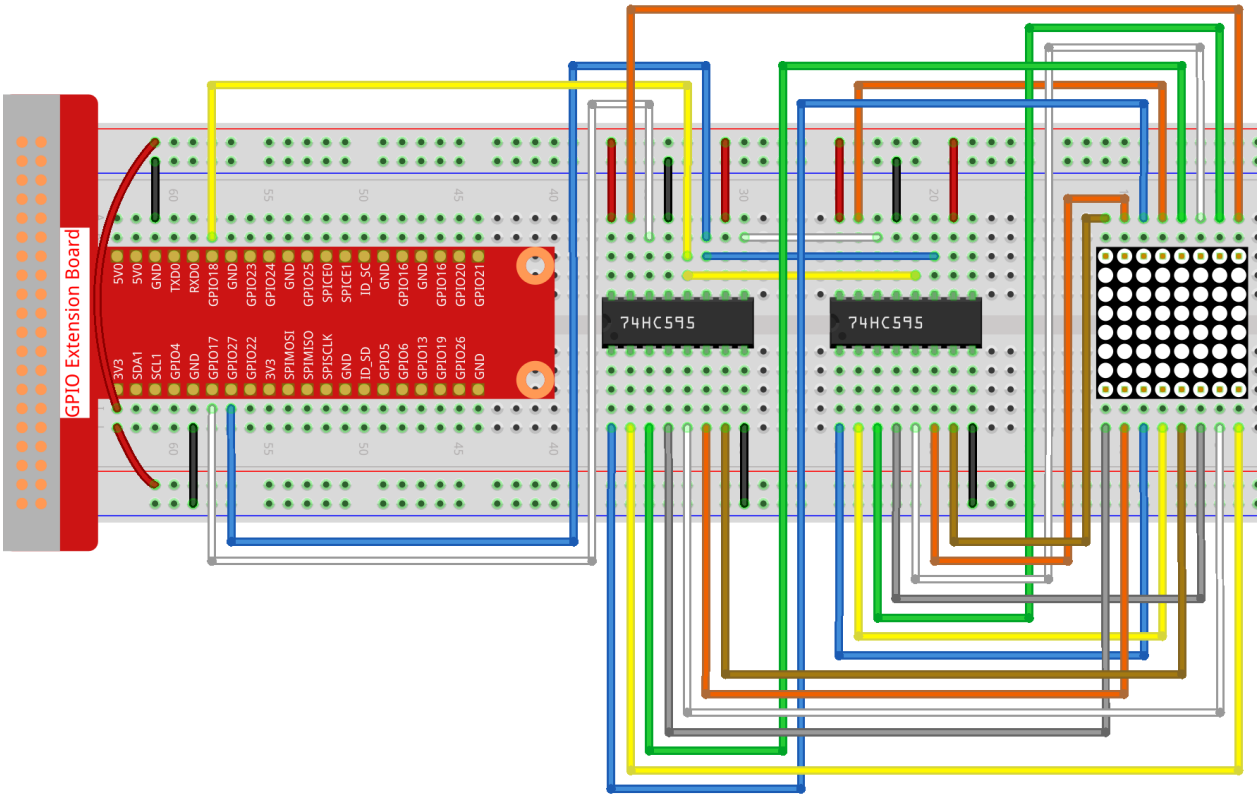
ステップ 3: 右側の 74HC595 は、LED ドットマトリックスの列を制御する。マッピングについては、以下の表を参照してください。したがって、74HC595 の Q0〜Q7 ピンは、それぞれピン 13、3、4、10、6、11、15、および 16 にマップされる。

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
```

(次のページに続く)

(前のページからの続き)

```
#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,0xff,0xff,0xff};
unsigned char code_L[20] = {0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
↪0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,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;
```

(次のページに続く)

(前のページからの続き)

```
    }
    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;
}
```

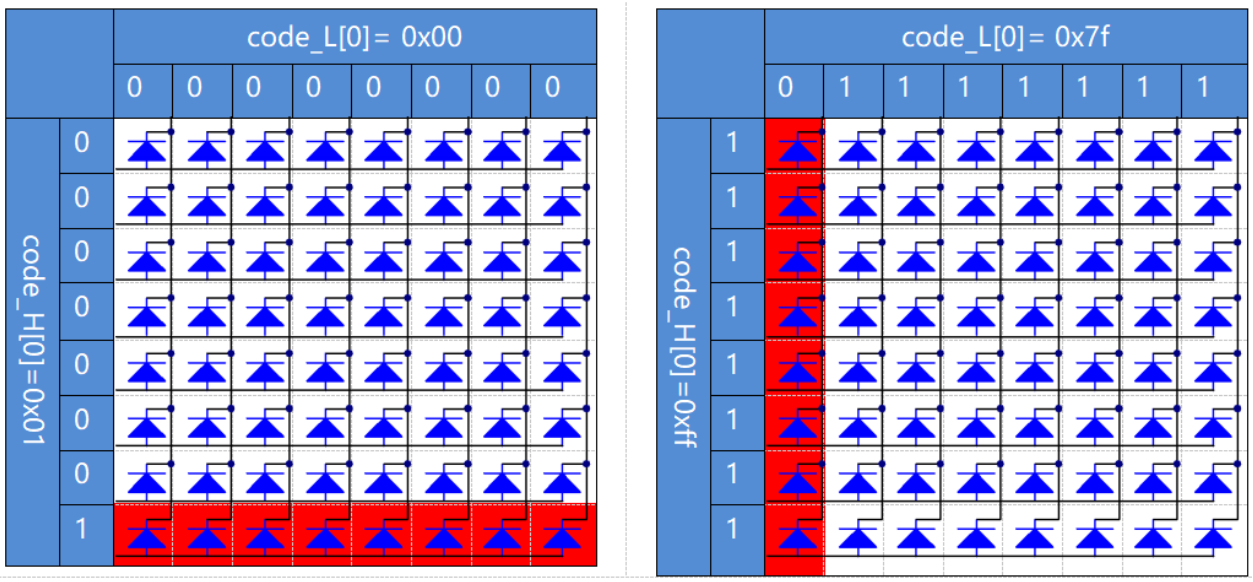
Code Explanation

```
unsigned char code_H[20] = {0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
↪0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};
unsigned char code_L[20] = {0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
↪0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,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);
```

HC 595 ビットのピン SDI にビット単位で DAT の値を書き込みます。SRCLK の初期値は 0 に設定され、ここで 1 がセットされ、立ち上がりエッジパルスを生成し、次に PHSDI（DS）の日付をシフトレジスタにシフトさせる。

```
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);
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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,0xff,0xff,0xff]
# COL  ----
code_L = [0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,
↪0xf7,0xef,0xdf,0xbf,0x7f]

def setup():
```

(次のページに続く)

(前のページからの続き)

```
    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:
```

(次のページに続く)

(前のページからの続き)

```
    destroy()
```

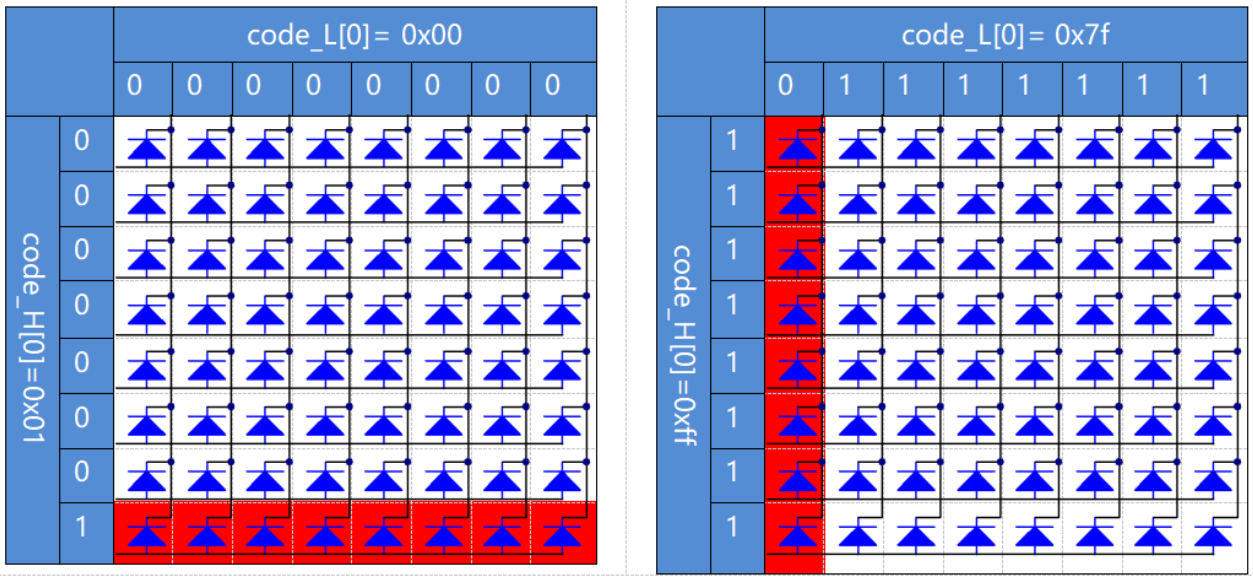
コードの説明

```
code_H = [0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,
↪0xff,0xff,0xff,0xff,0xff]
code_L = [0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,
↪0xf7,0xef,0xdf,0xbf,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。

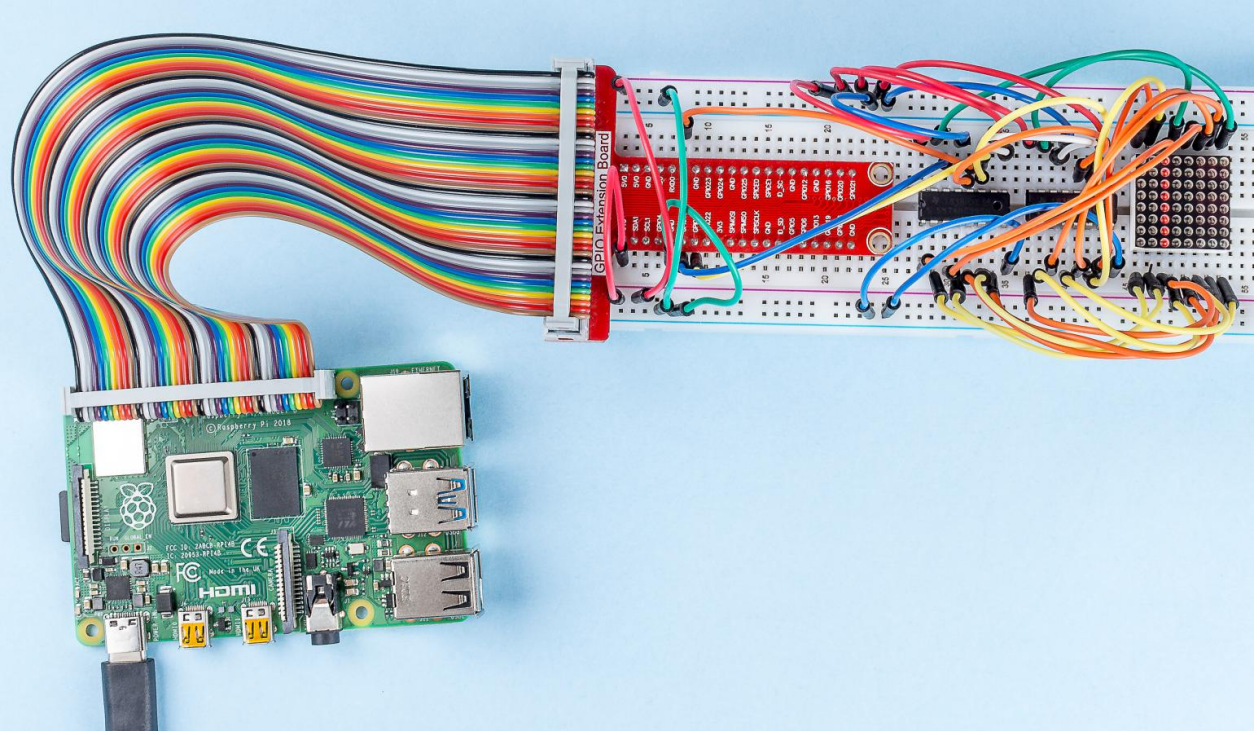



```
for i in range(0, len(code_H)):
    hc595_shift(code_L[i])
    hc595_shift(code_H[i])
```

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。

注釈: 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

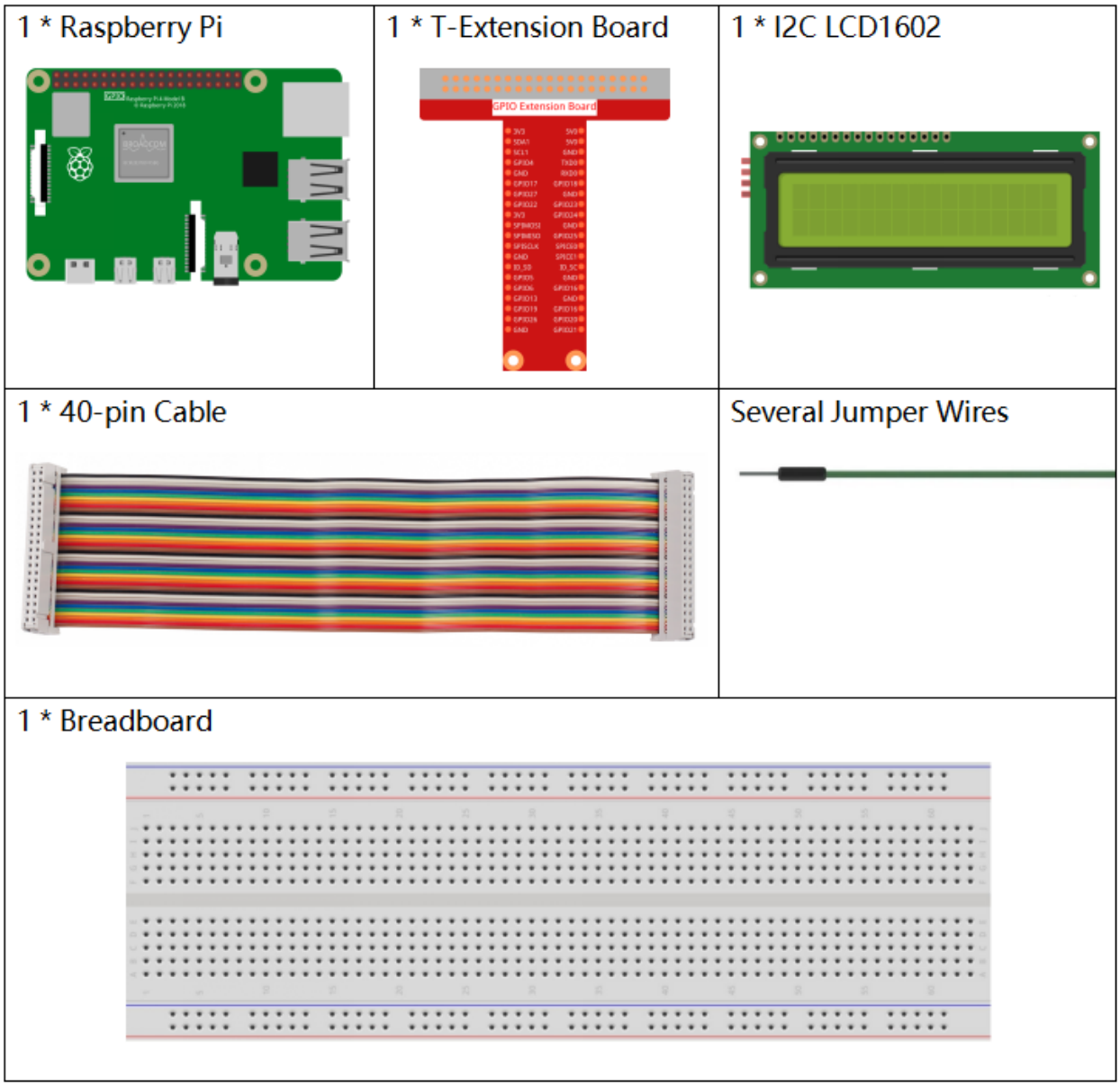


1.8.7 1.1.7 I2C LCD1602

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。

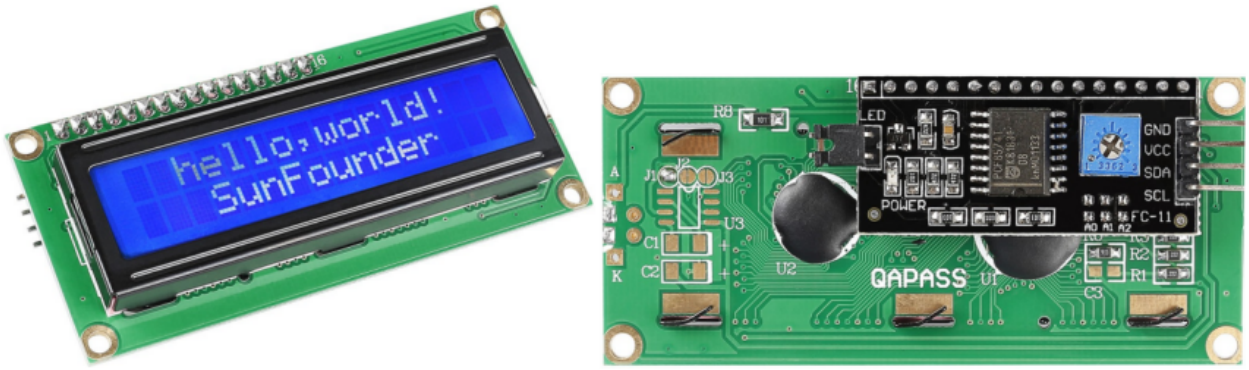
部品



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。



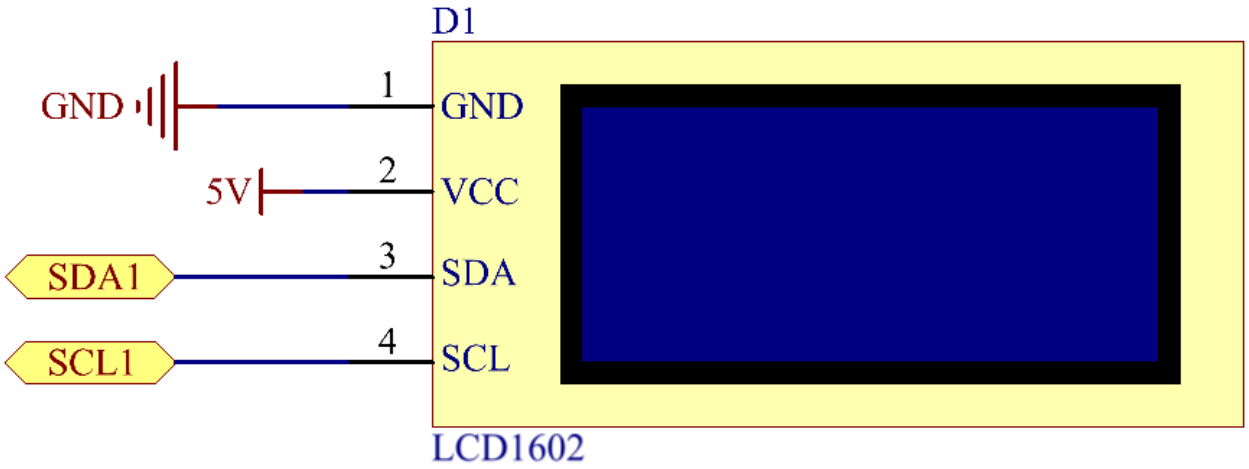
I2C 通信

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）と単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。

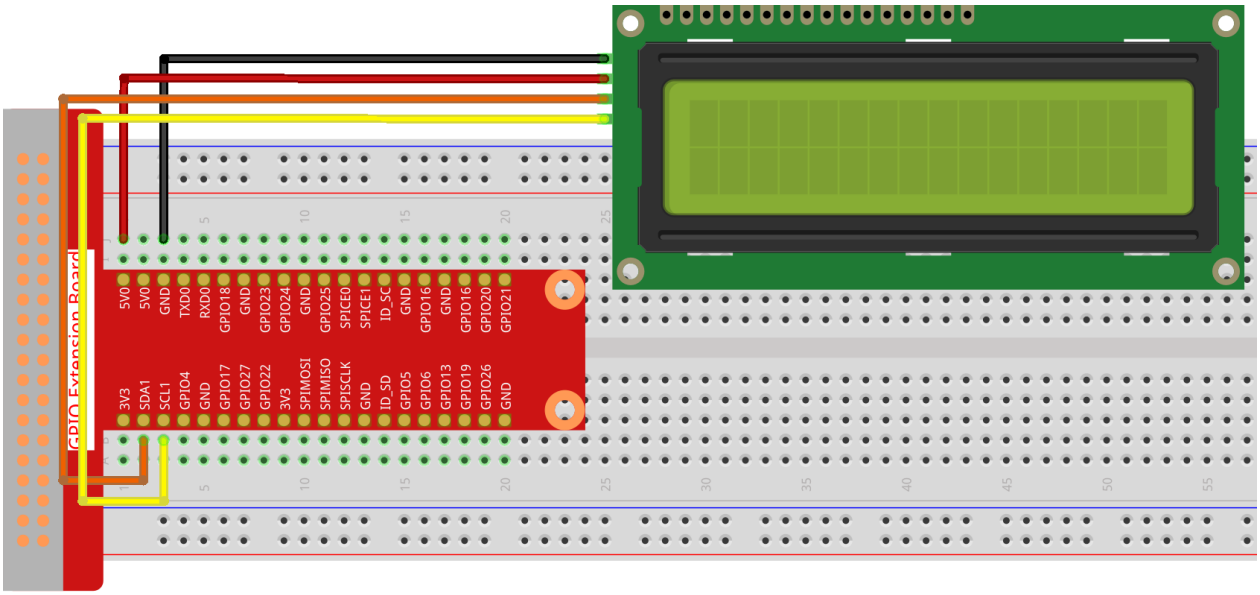
回路図

T-Board Name	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 をセットアップします（*I2C* **構成** 。I2C を設定している場合は、このステップ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。

注釈: コードと配線に問題がないのに LCD に内容が表示されない場合は、背面のポテンショメータをひねってみてください。

コード

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <string.h>

int LCDAddr = 0x27;
int BLEN = 1;
int fd;

void write_word(int data){
    int temp = data;
    if ( BLEN == 1 )
        temp |= 0x08;
    else
        temp &= 0xF7;
    wiringPiI2CWrite(fd, temp);
}

void send_command(int comm){
    int buf;
    // Send bit7-4 firstly
    buf = comm & 0xF0;
    buf |= 0x04;                    // RS = 0, RW = 0, EN = 1
    write_word(buf);
    delay(2);
    buf &= 0xFB;                    // Make EN = 0
    write_word(buf);

    // Send bit3-0 secondly
    buf = (comm & 0x0F) << 4;
    buf |= 0x04;                    // RS = 0, RW = 0, EN = 1
    write_word(buf);
    delay(2);
    buf &= 0xFB;                    // Make EN = 0
    write_word(buf);
}

void send_data(int data){
    int buf;
```

(次のページに続く)

(前のページからの続き)

```
    // Send bit7-4 firstly
    buf = data & 0xF0;
    buf |= 0x05;                    // RS = 1, RW = 0, EN = 1
    write_word(buf);
    delay(2);
    buf &= 0xFB;                    // Make EN = 0
    write_word(buf);

    // Send bit3-0 secondly
    buf = (data & 0x0F) << 4;
    buf |= 0x05;                    // RS = 1, RW = 0, EN = 1
    write_word(buf);
    delay(2);
    buf &= 0xFB;                    // Make EN = 0
    write_word(buf);
}

void init(){
    send_command(0x33);     // Must initialize to 8-line mode at first
    delay(5);
    send_command(0x32);     // Then initialize to 4-line mode
    delay(5);
    send_command(0x28);     // 2 Lines & 5*7 dots
    delay(5);
    send_command(0x0C);     // Enable display without cursor
    delay(5);
    send_command(0x01);     // Clear Screen
    wiringPiI2CWrite(fd, 0x08);
}

void clear(){
    send_command(0x01);     //clear Screen
}

void write(int x, int y, char data[]){
    int addr, i;
    int tmp;
    if (x < 0)  x = 0;
    if (x > 15) x = 15;
```

(次のページに続く)

(前のページからの続き)

```
    if (y < 0)  y = 0;
    if (y > 1)  y = 1;

    // Move cursor
    addr = 0x80 + 0x40 * y + x;
    send_command(addr);

    tmp = strlen(data);
    for (i = 0; i < tmp; i++){
        send_data(data[i]);
    }
}


void main(){
    fd = wiringPiI2CSetup(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 = wiringPiI2CSetup(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 wiringPiI2CSetup(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。

注釈:

- FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1' というエラーが発生した場合は、 *I2C* **構成** を参照して I2C を有効にする必要があります。
- ModuleNotFoundError: No module named 'smbus2' エラーが発生した場合は、sudo pip3 install smbus2 を実行してください。
- OSError: [Errno 121] Remote I/O というエラーが発生した場合は、モジュールの配線が間違っているか、モジュールが破損しています。
- コードと配線に問題がないのに LCD に内容が表示されない場合は、背面のポテンショメータをひねってみてください。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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。

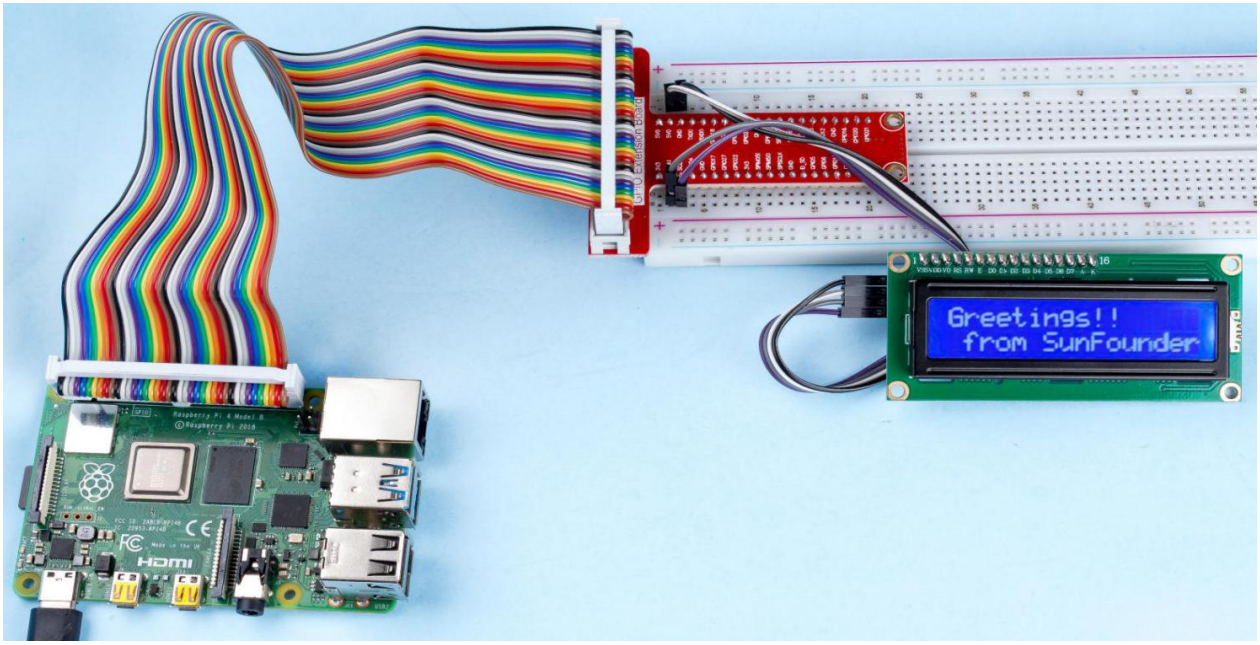
```
LCD1602.init(0x27, 1)
```

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!!')
```

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

現象画像



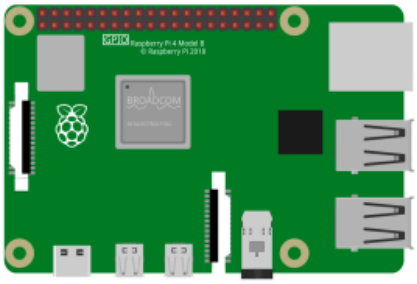
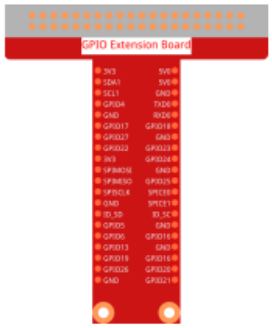
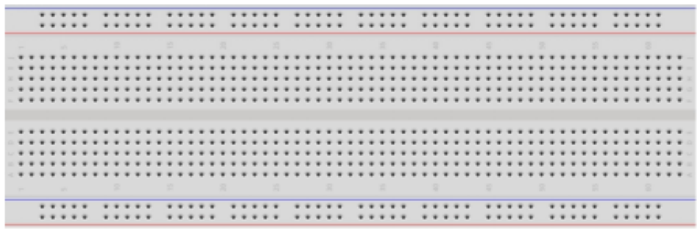
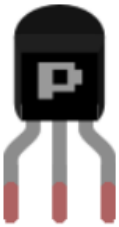
1.2 音声

1.8.8 1.2.1 アクティブブザー

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

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Active Buzzer
		1 * Resistor(1kΩ)
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * S8550 PNP Transistor 

原理

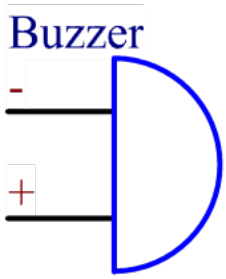
ブザー

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 の方形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。

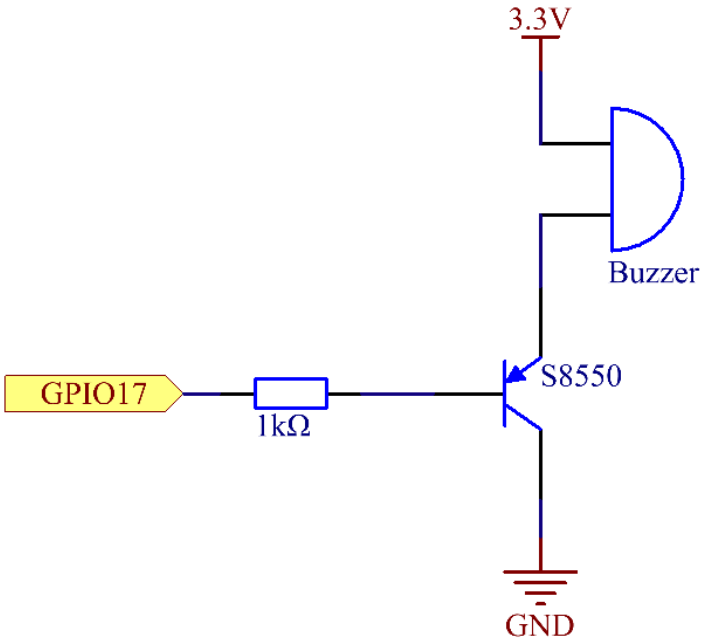


ブザーのピンをチェックすることができ、長い方が陽極で、短い方が陰極である。接続時にそれらを混同しないでください。混同すると、ブザーが鳴らない。

回路図

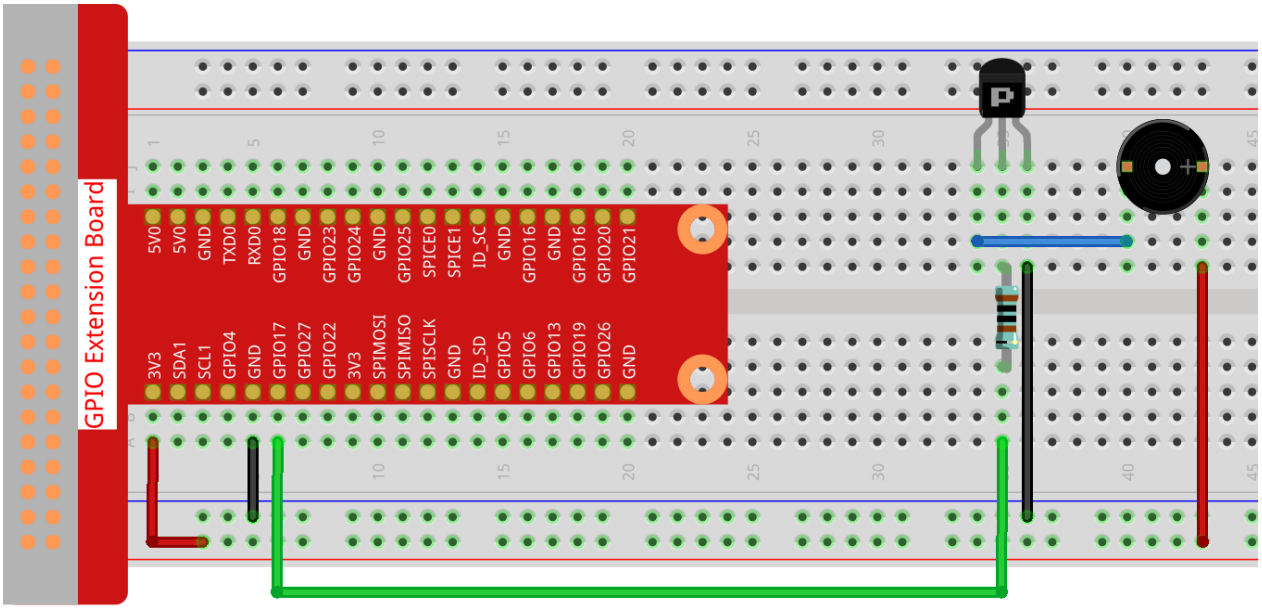
この実験では、アクティブブザー、PNP トランジスタ、および 1k 抵抗器をトランジスタのベースと GPIO の間に使用して、トランジスタを保護する。Raspberry Pi 出力の GPIO17 にプログラミングによって低レベル（0V）が供給されると、電流飽和のためトランジスタが導通し、ブザーが音を出す。しかし、Raspberry Pi の IO に高レベルが供給されると、トランジスターが切断され、ブザーは音を出さない。

T-Board Name	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 messageto 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

# Set #17 as buzzer pin
BeepPin = 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(BeepPin, GPIO.OUT, initial=GPIO.HIGH)
```

(次のページに続く)

(前のページからの続き)

```
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)
```

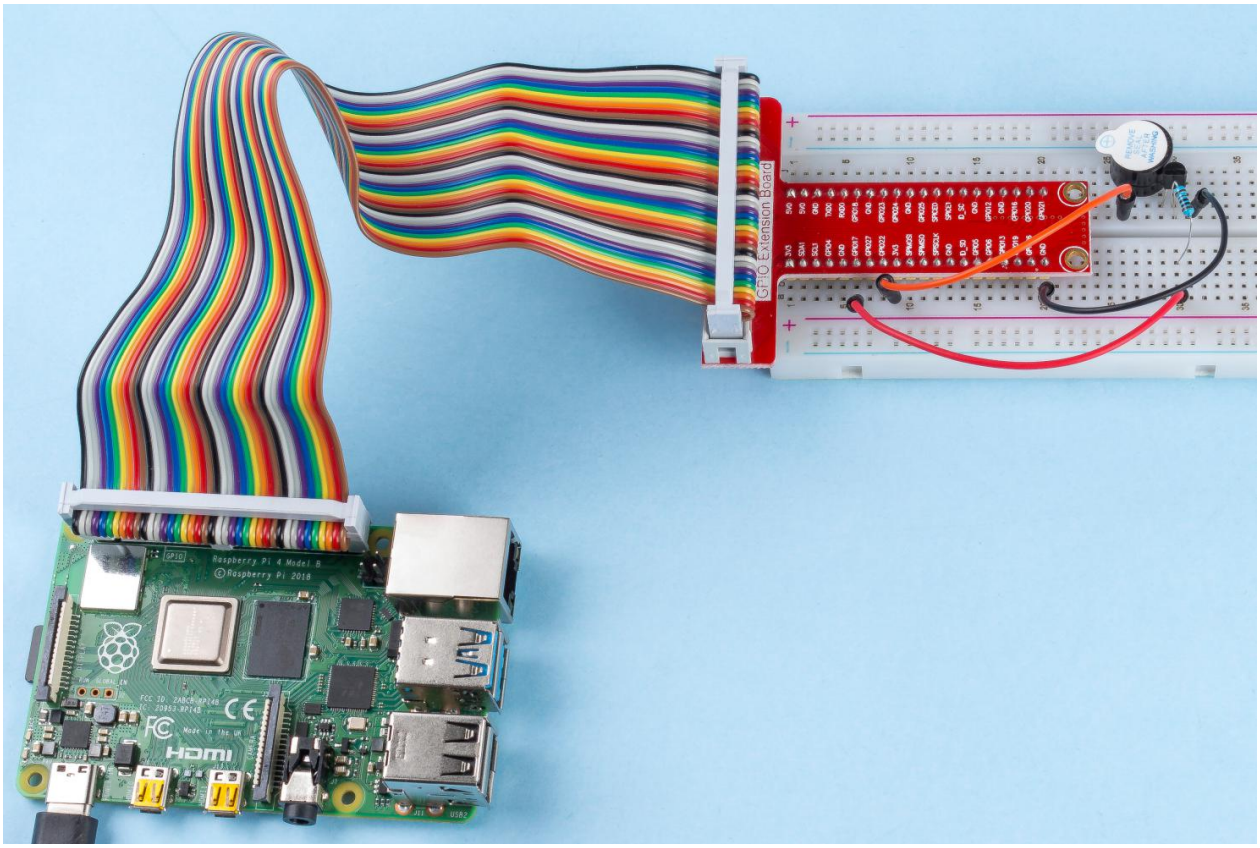
0.1 秒待つ。このパラメーターを変更して、スイッチング周波数を変更する。

注釈: 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

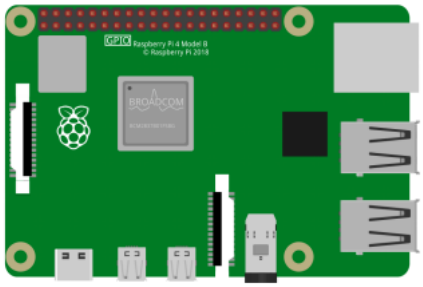
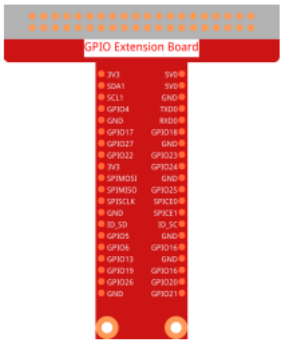
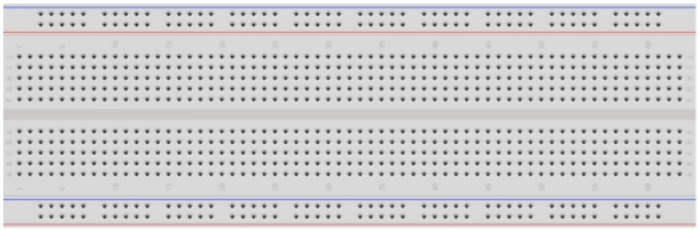


1.8.9 1.2.2 パッシブブザー

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

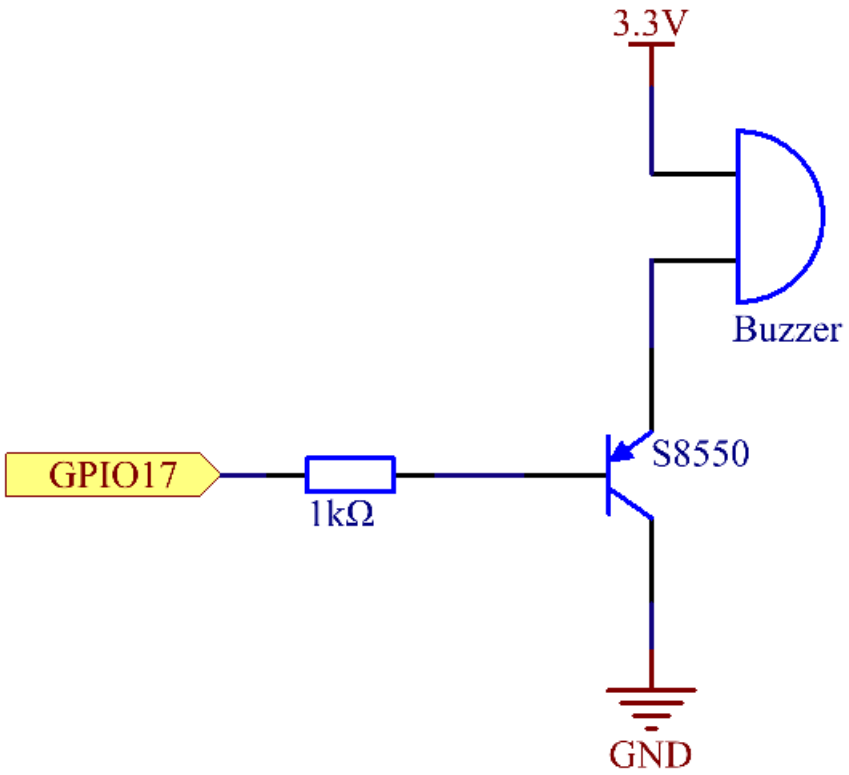
1 * Raspberry Pi 	1 * T-Extension Board 	1 * Passive Buzzer
1 * 40-pin Cable		1 * Resistor(1kΩ)
		Several Jumper Wires
1 * Breadboard 		1 * S8550 PNP Transistor

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。

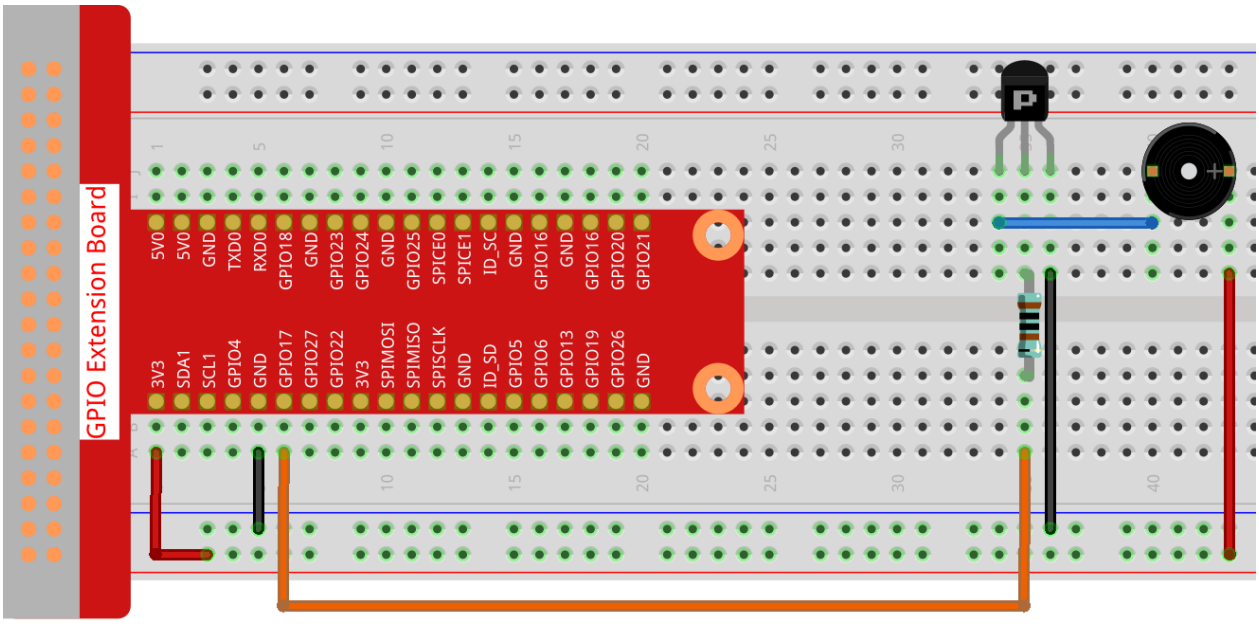
GPIO17 に異なる周波数が与えられると、パッシブブザーは異なる音を出す。このようにして、ブザーは音楽を再生する。

T-Board Name	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： コードをコンパイルする。

```
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
```

(次のページに続く)

(前のページからの続き)

```
#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[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,2,1,1,
                1,1,1,1,1,1,3};


int song_2[] = {CM1,CM1,CM1,CL5,CM3,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;
```

(次のページに続く)

(前のページからの続き)

```
    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");
        delay(100);
        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[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,1,2,1,1,
                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 ビートを取るため、4 で除算します。

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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
            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] ]

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

(次のページに続く)

(前のページからの続き)

```
song_2 = [  CM[1], CM[1], CM[1], CL[5], CM[3], 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
```

(次のページに続く)

(前のページからの続き)

```
    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.
        destory()
```

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

すべての音符の拍（各番号）は、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])
```

(次のページに続く)

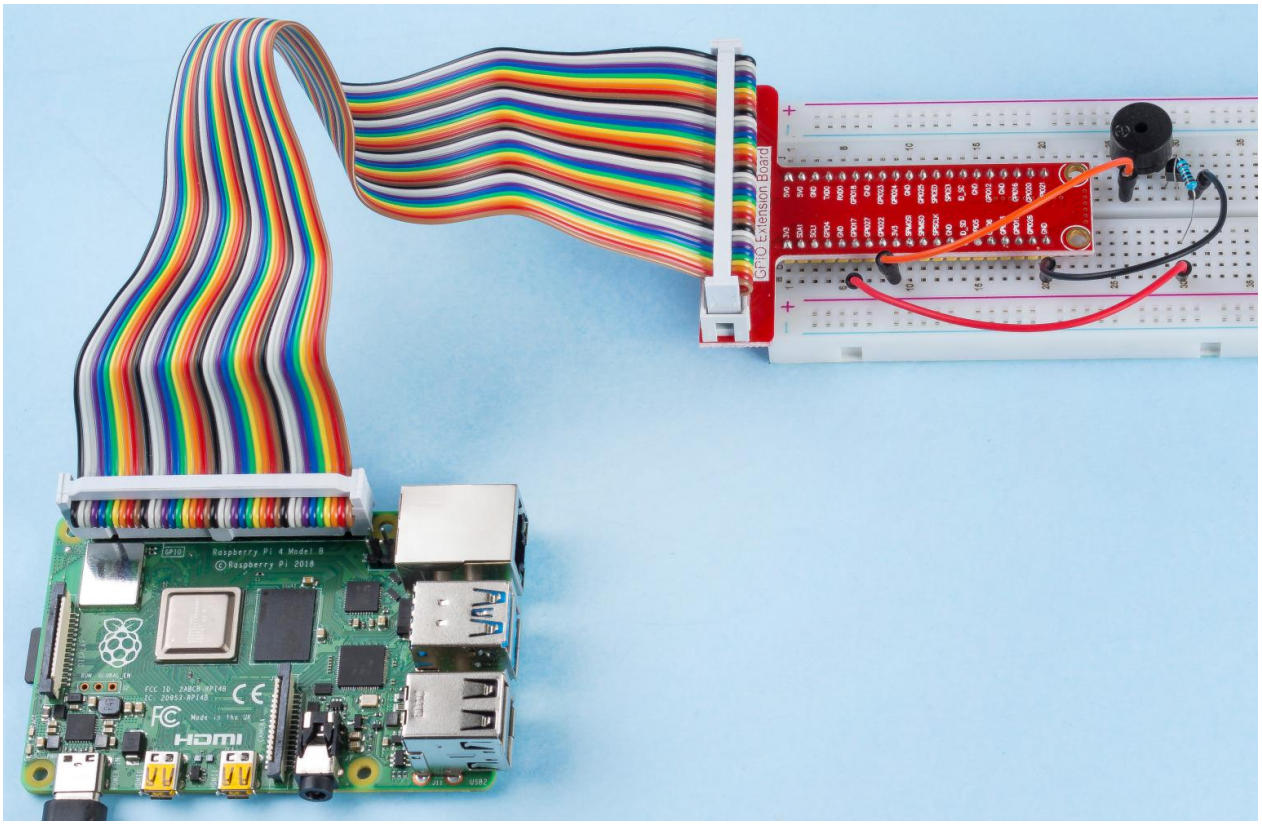
(前のページからの続き)

```
            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



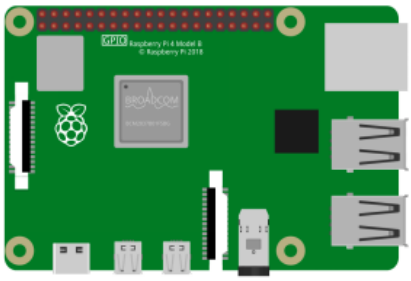
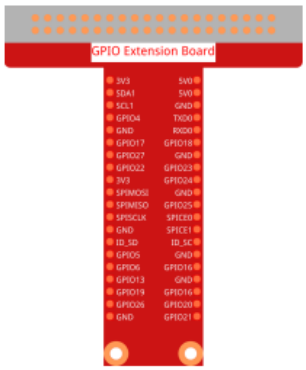
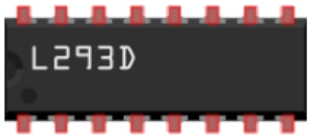
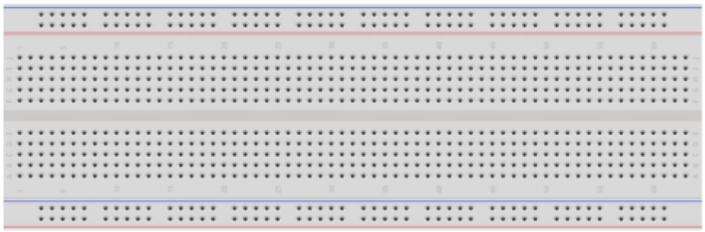
1.3 ドライバー

1.8.10 1.3.1 モーター

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

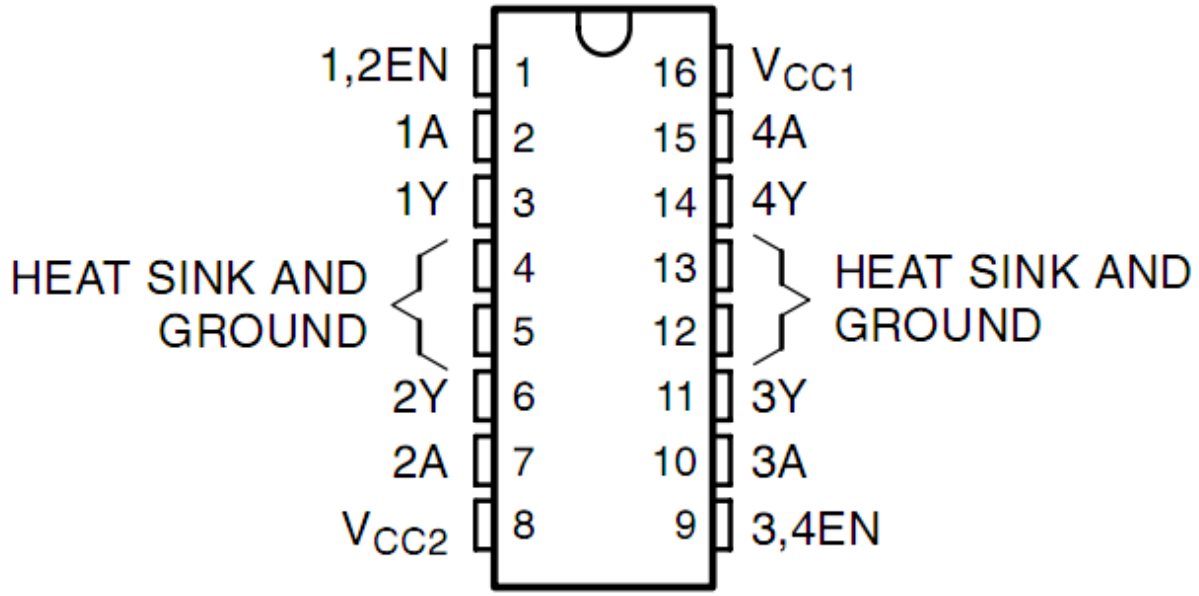
1 * Raspberry Pi 	1 * T-Extension Board 	1 * Power Module (with 9V battery and buckle)
1 * 40-pin Cable		1 * L293D 
		Several Jumper Wires
1 * Breadboard 		1 * DC Motor

原理

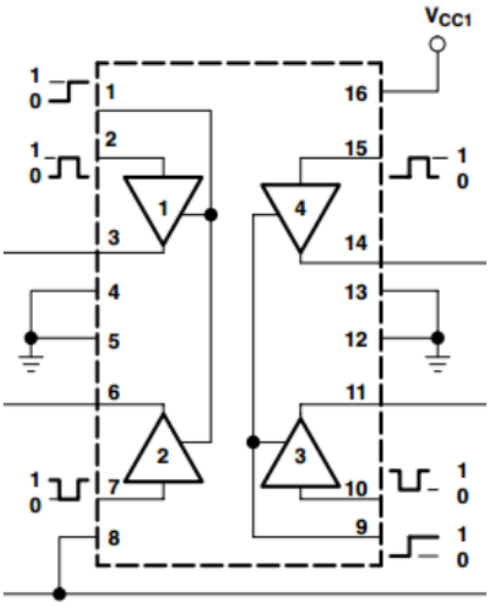
L293D

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（Vcc1 と Vcc2）がある。Vcc2 はモーターに電力を供給し、Vcc1 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。



INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,
Z = high impedance (off)

DC モーター

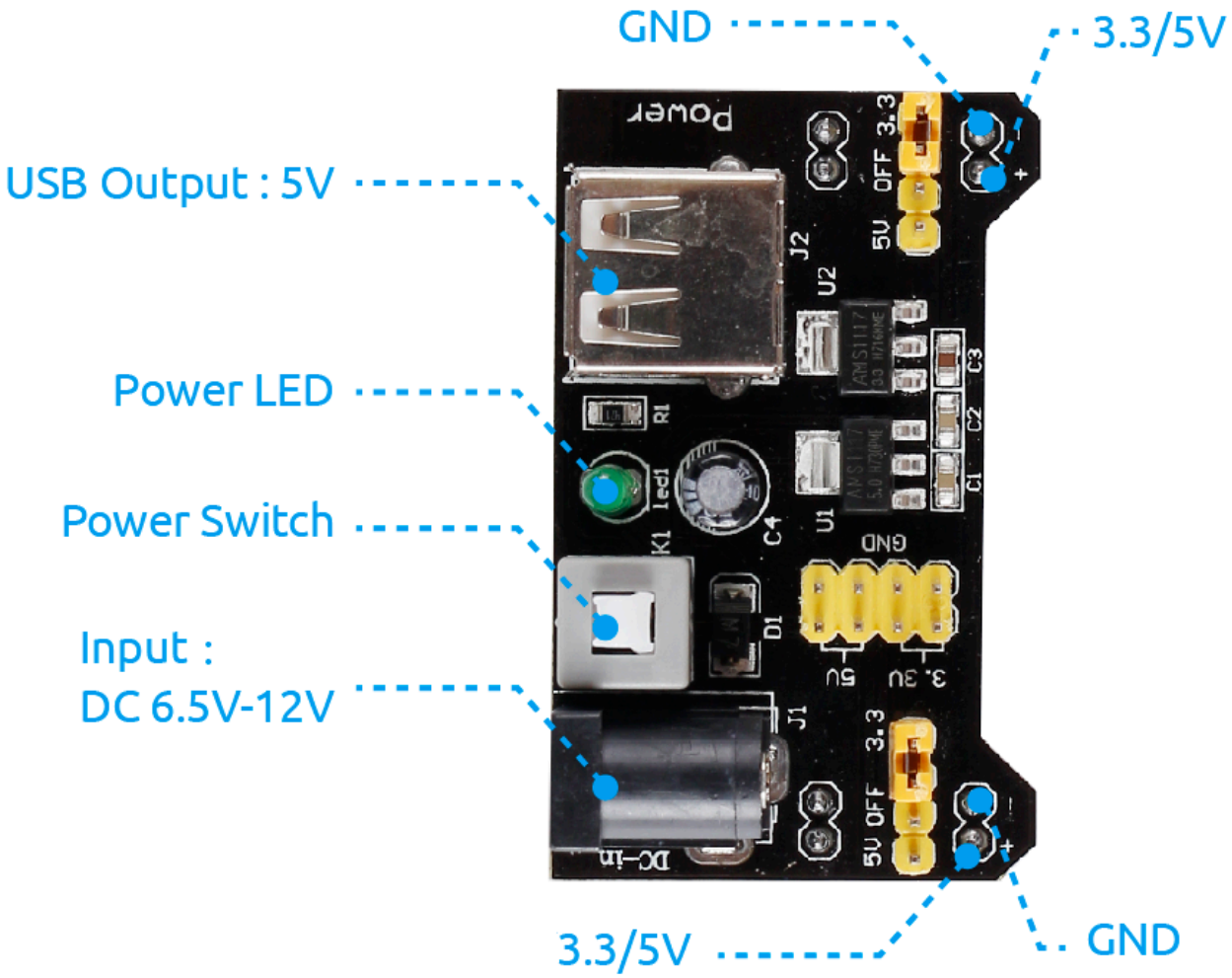
これは 5V DC モーターである。銅板の 2 つの端子に 1 つの高レベルと 1 つの低レベルを与えると回転する。便宜上、ピンを溶接することができる。

Size: 25*20*15MM	Operation Voltage: 1-6V
Free-run current (3V):70m	A Free-run speed (3V): 13000RPM
Stall current (3V): 800mA	Shaft diameter: 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。

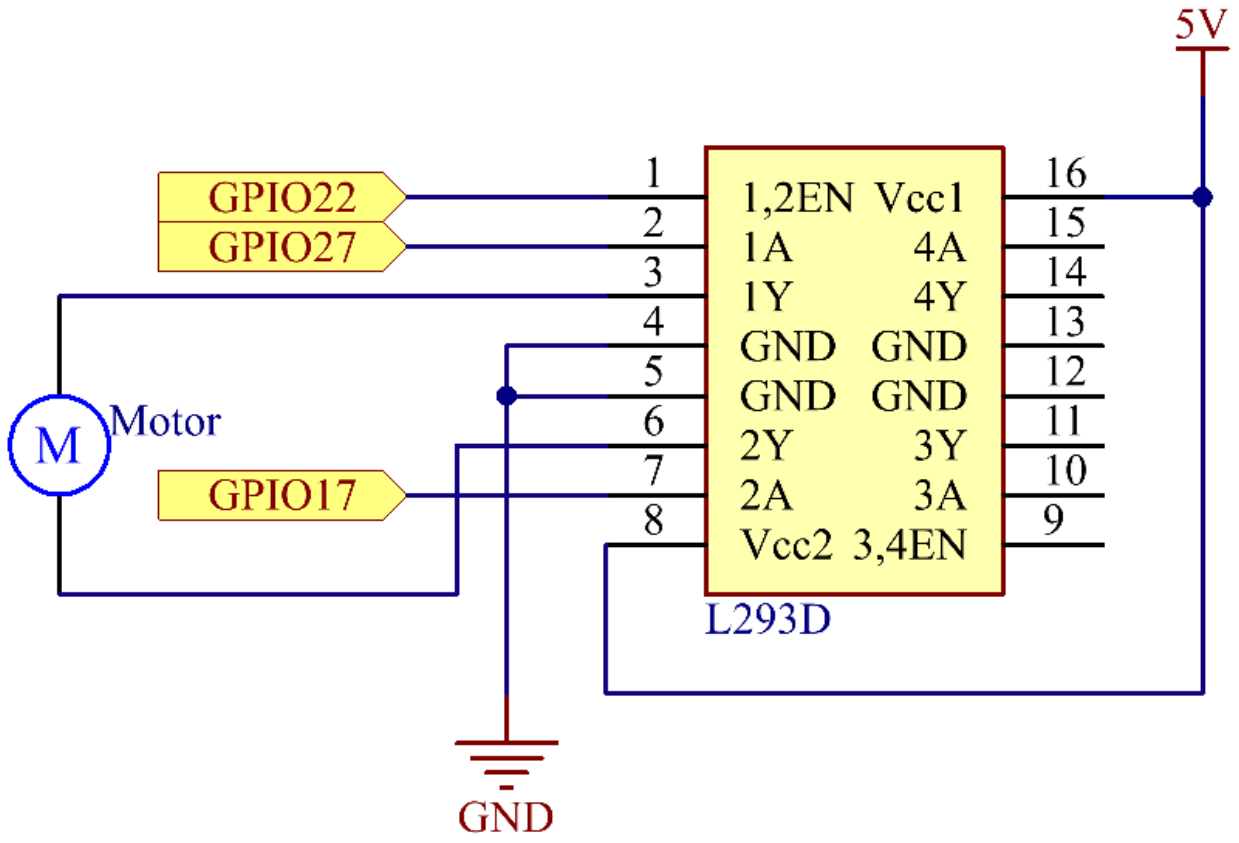
ブレッドボードに差し込むだけで電力を供給できる。3.3V と 5V の電圧を提供し、付属のジャンパーキャップを介してどちらでも接続できる。



回路図

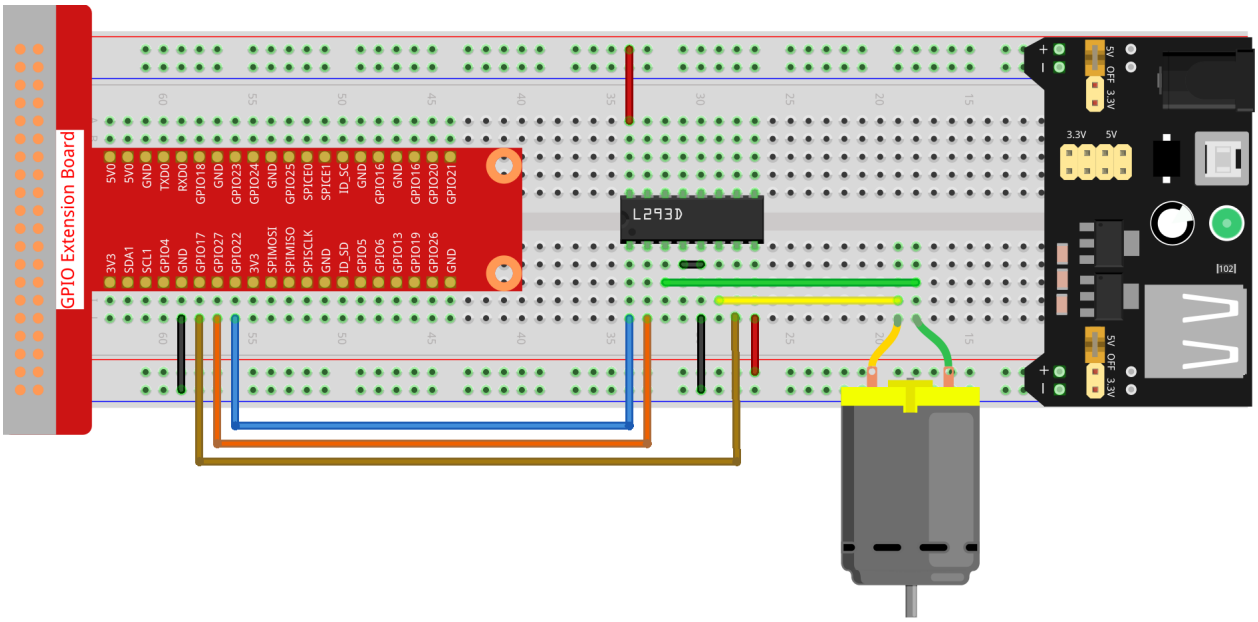
電源モジュールをブレッドボードに差し込み、ジャンパーキャップを 5V のピンに挿入すると、5V の電圧が出力される。L293D のピン 1 を GPIO22 に接続し、それを高レベルに設定する。ピン 2 を GPIO27 に、ピン 7 を GPIO17 に接続し、一方のピンを high に、もう一方のピンを high に設定する。したがって、モーターの回転方向を変更できる。

T-Board Name	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。



注釈: 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 messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(MotorPin1, OUTPUT);
    pinMode(MotorPin2, OUTPUT);
    pinMode(MotorEnable, OUTPUT);
    while(1){
        printf("Clockwise\n");
        delay(100);
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, HIGH);
        digitalWrite(MotorPin2, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Stop\n");
        delay(100);
        digitalWrite(MotorEnable, LOW);
        for(i=0;i<3;i++){
            delay(1000);
```

(次のページに続く)

(前のページからの続き)

```
        }

        printf("Anti-clockwise\n");
        delay(100);
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, LOW);
        digitalWrite(MotorPin2, HIGH);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Stop\n");
        delay(100);
        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 * 1000ms 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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
```

(次のページに続く)

(前のページからの続き)

```
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:
```

(次のページに続く)

(前のページからの続き)

```
            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)):
```

(次のページに続く)

(前のページからの続き)

```
        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 motor(direction):
    # Clockwise
    if direction == 1:
```

(次のページに続く)

(前のページからの続き)

```
        # 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。

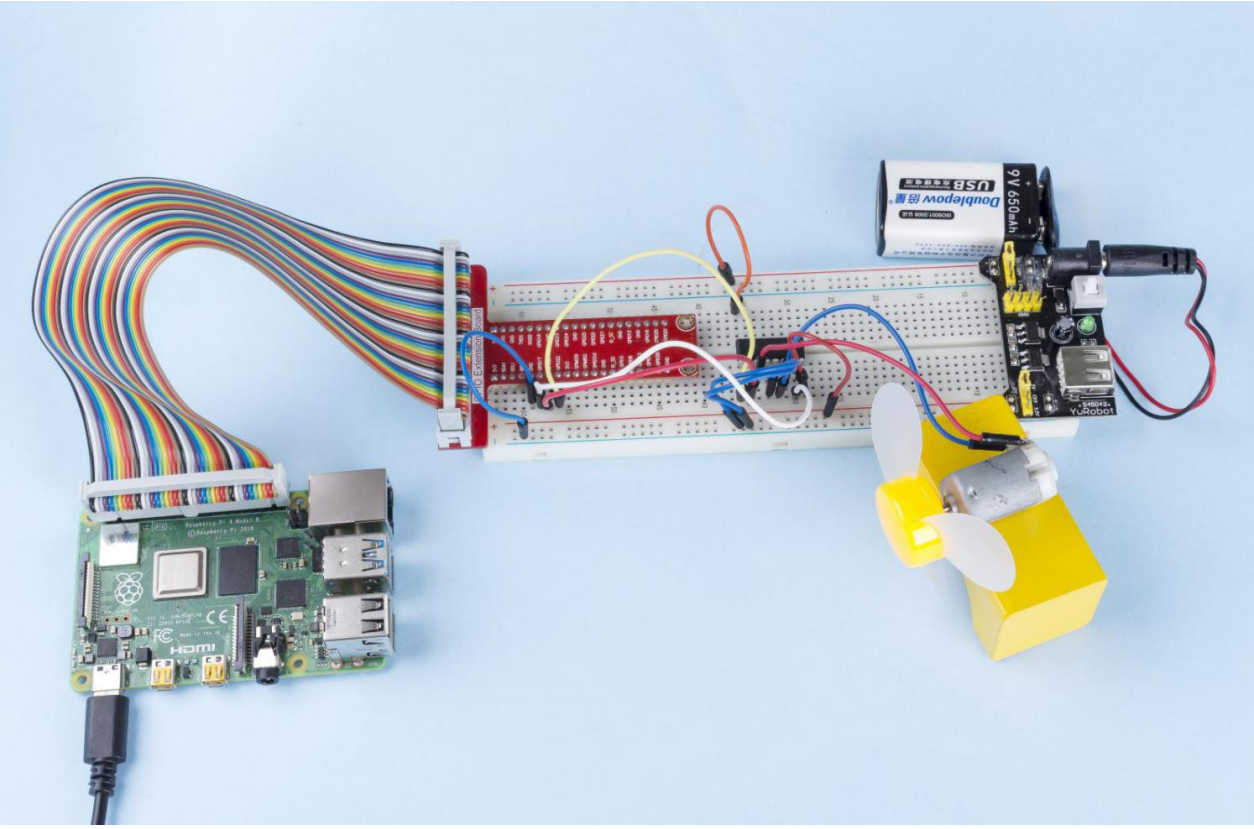
```
def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}
    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])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

main() 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

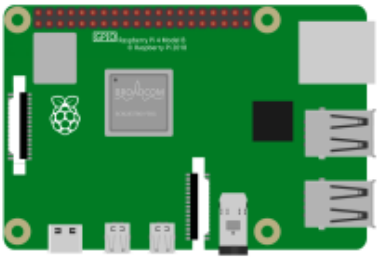
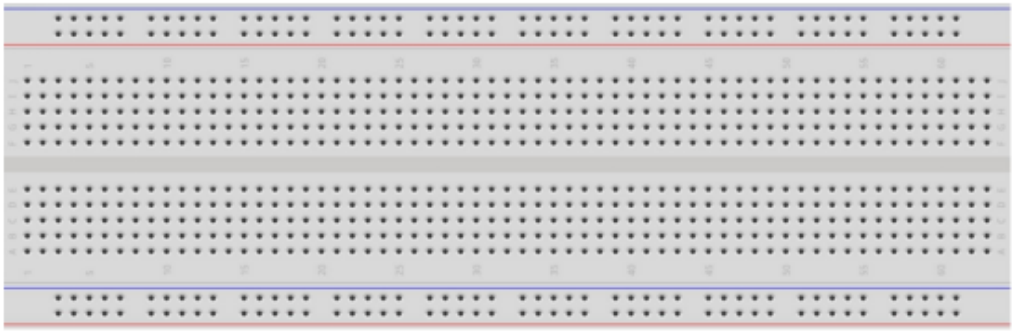


1.8.11 1.3.2 サーボ

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

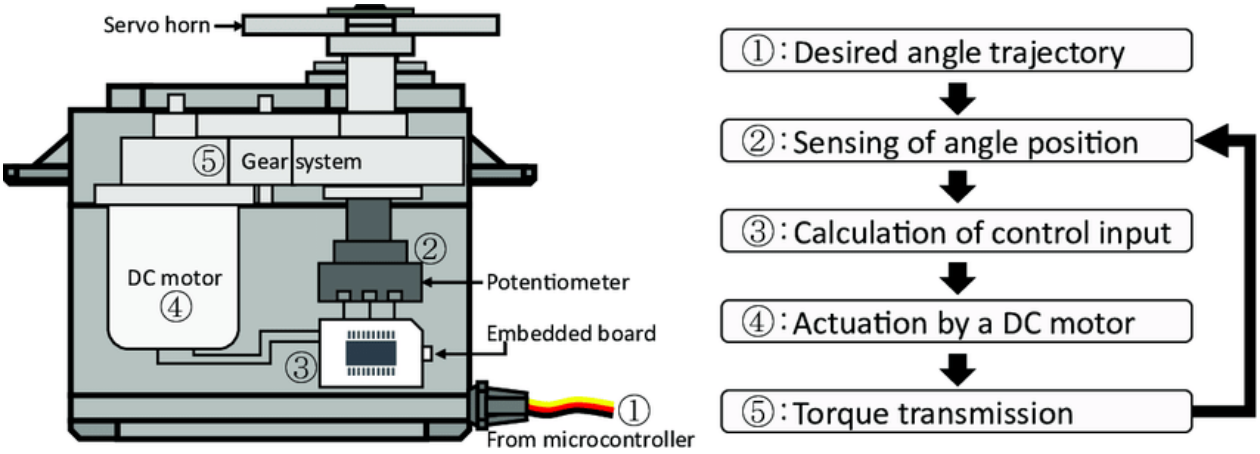
1 * Raspberry Pi 	1 * T-Extension Board	1 * Servo
		Several Jumper Wires
1 * 40-pin Cable		
1 * Breadboard 		

原理

サーボ

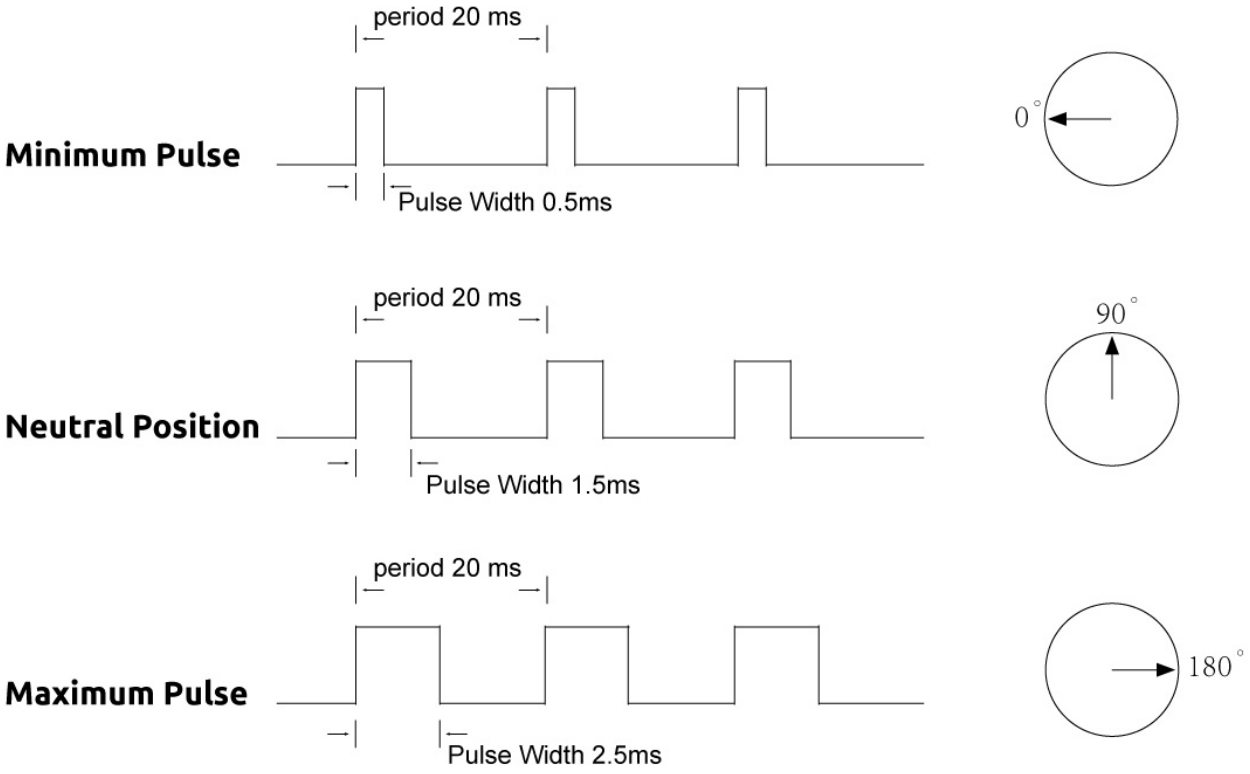
サーボは一般に、ケース、シャフト、ギアシステム、ポテンショメーター、DC モーター、および内蔵式ボードで構成されている。

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。



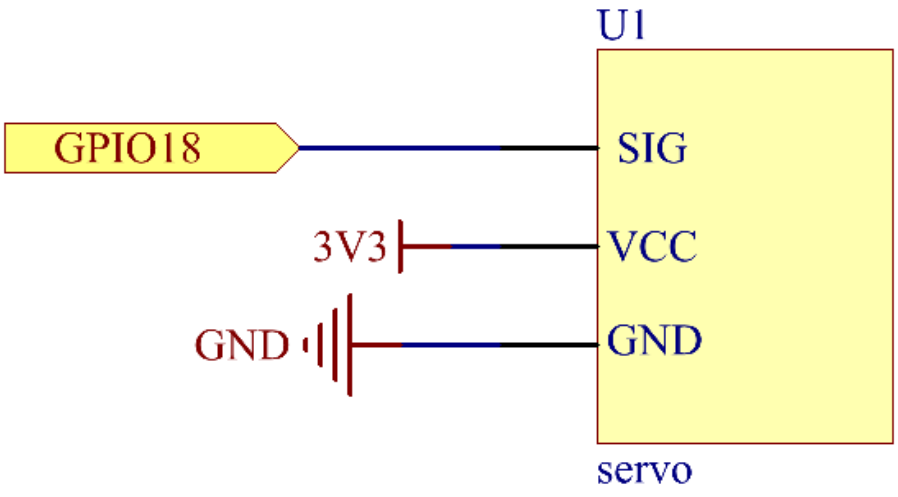
角度は制御ワイヤに適用されるパルスの持続時間によって決まる。これはパルス幅変調と呼ばれる。サーボは 20 ミリ秒ごとに 1 パルスを期待している。パルスの長さにより、モーターの回転距離が決まる。たとえば、1.5ms パルスは、モーターを 90 度の位置（ニュートラル位置）に回転させる。

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。



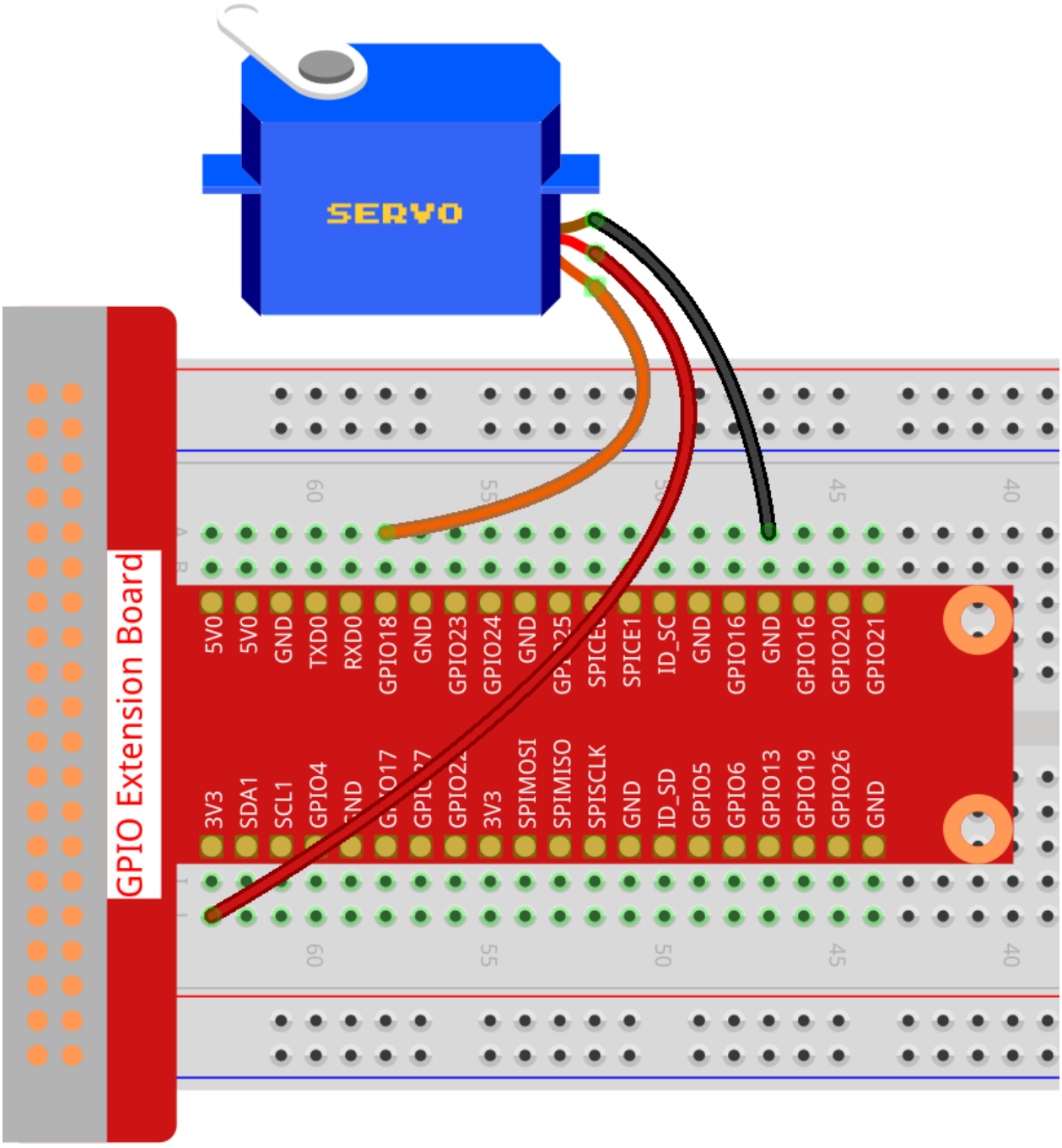
回路図

T-Board Name	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 funtion 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 !");
```

(次のページに続く)

(前のページからの続き)

```
        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 funtion 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 200x100us = 20ms に設定されているため、0〜180 を 5x100us〜25x100us 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) ;
```

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

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

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

ServoPin = 18

def map(value, inMin, inMax, outMin, outMax):
```

(次のページに続く)

(前のページからの続き)

```
    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 Frequecy 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 Frequecy 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。

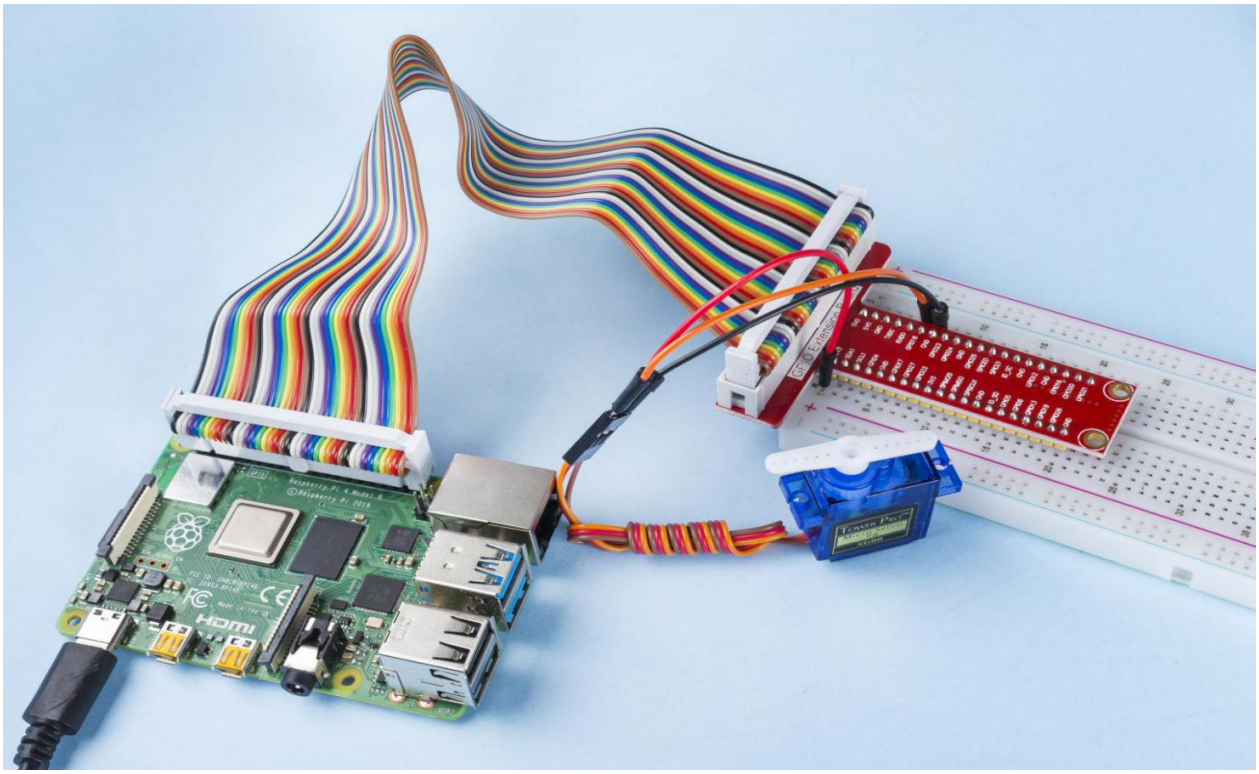
max() メソッドは、反復可能または最大の 2 つ以上のパラメーターで最大要素を返す。角度が 0 以上の場合は 0 を返し、そうでない場合は角度を返す。

```
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

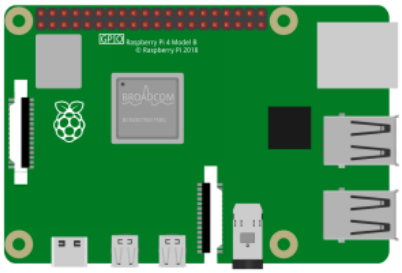
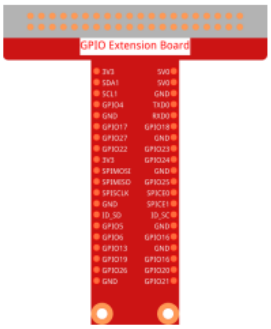
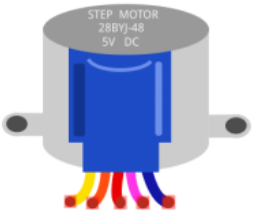
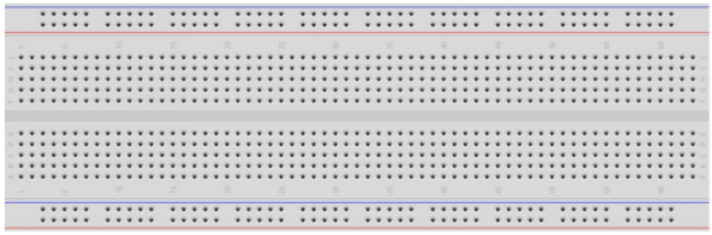
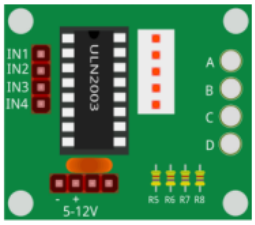


1.8.12 1.3.3 ステッピングモーター

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

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Stepper Motor 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * ULN2003 

原理

ステッピングモーター

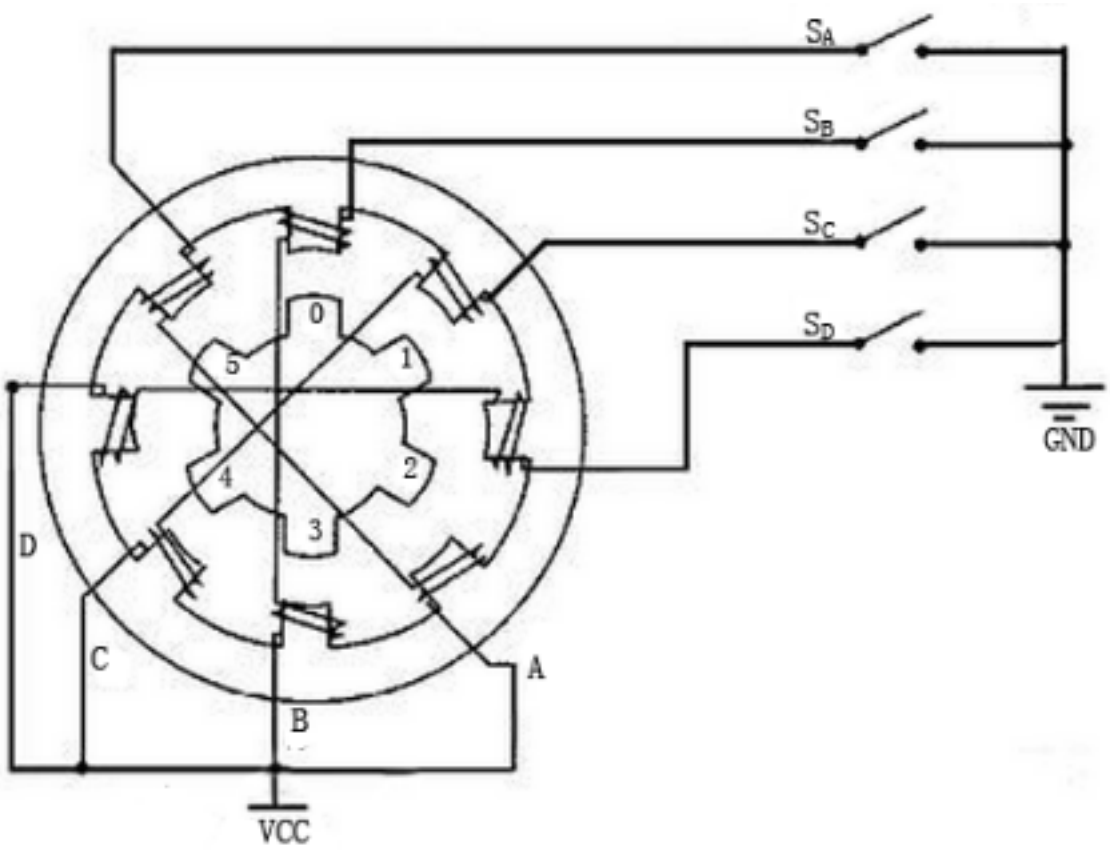
ステッパーには、ユニポーラーとバイポーラーの 2 つのタイプがあり、使用しているタイプを知ることが非常に重要である。この実験では、ユニポーラステッパーを使用する。

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、一歩一歩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く。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。

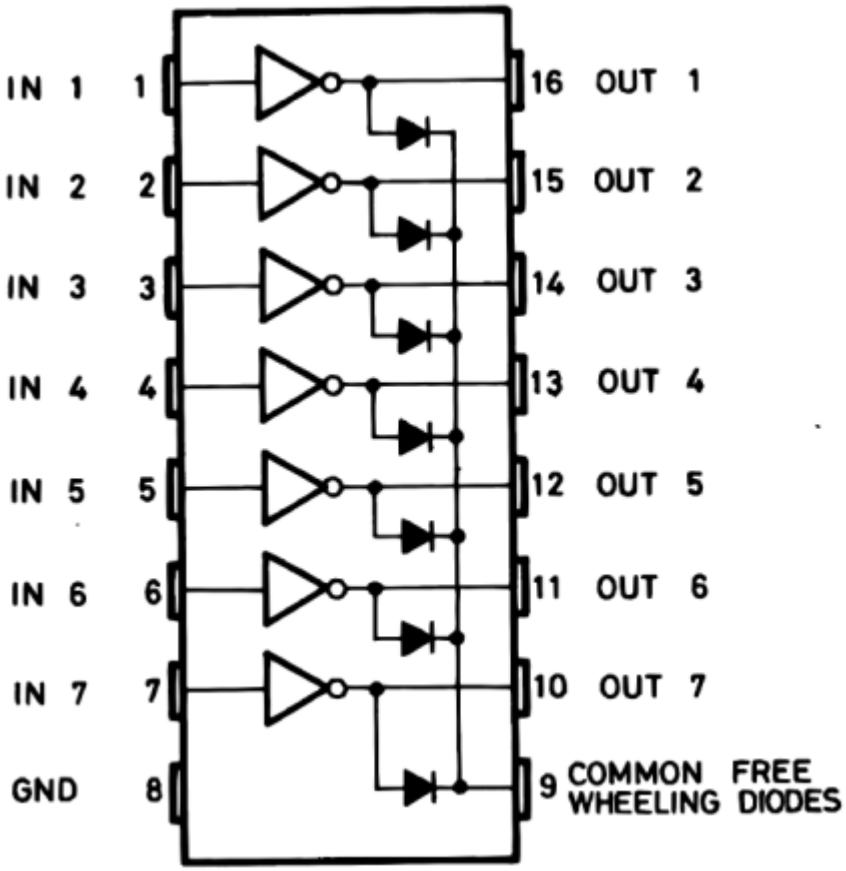


4 相ステッピングモーターには、3 つの動作モードがある：シングル 4 ステップ、ダブル 4 ステップ、および 8 ステップ。シングル 4 ステップとダブル 4 ステップのステップ角は同じであるが、シングル 4 ステップの駆動トルクは小さくなる。8 ステップのステップ角は、シングル 4 ステップおよびダブル 4 ステップの半分である。したがって、8 ステップの動作モードは、高い駆動トルクを維持し、制御精度を向上させることができる。

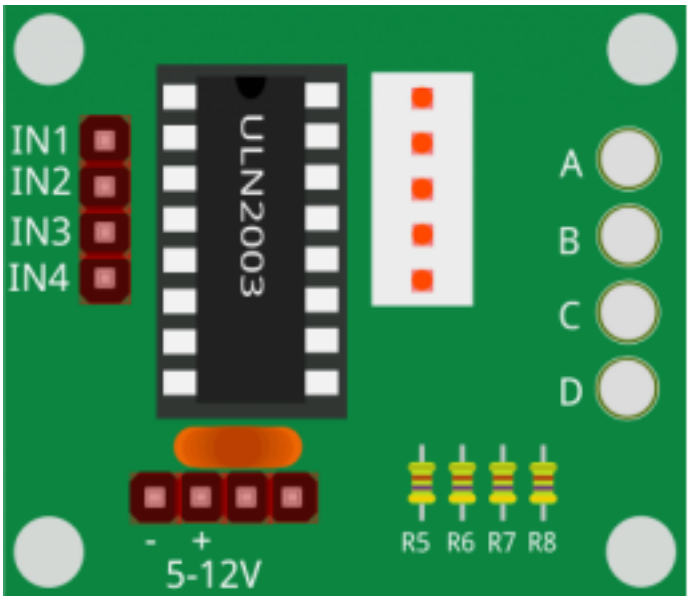
使用するステッピングモーターのステータには 32 個の磁極があるため、円には 32 ステップが必要である。ステッピングモーターの出力軸は減速装置セットに接続され、減速比は 1/64 である。ですから、最終出力シャフトは 32 * 64 = 2048 ステップを必要とする円を回転させる。

ULN2003

モーターを円に適用するには、ドライバーボードを使用する必要があります。ステッパーモータードライバ-ULN2003 は 7 チャネルインバータサークルです。つまり、入力ピンが高レベルの場合、ULN2003 の出力ピンは低レベルであり、その逆も同様です。IN1 に高レベルを供給し、IN2、IN3、IN4 に低レベルを供給すると、出力端 OUT1 は低レベルであり、他のすべての出力端は高レベルになります。チップの内部構造を以下に示します。

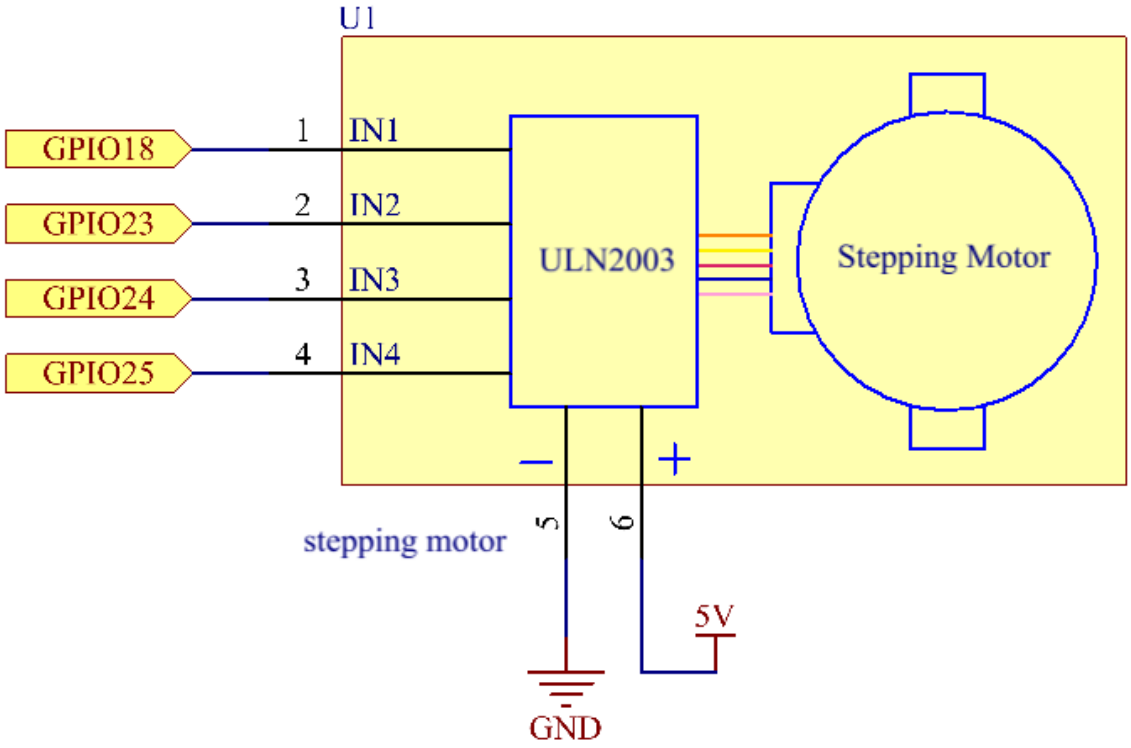


ULN2003 チップと 4 つの LED で構成されるステッピングモータードライバーを以下に示す。ボードでは、IN1、IN2、IN3、IN4 が入力として機能し、4 つの LED、A、B、C、D は入力ピンの指示器である。さらに、OUT1、OUT2、OUT3、および OUT4 はそれぞれ、ステッピングモータードライバーの SA、SB、SC、および SD に接続されている。IN1 の値が高レベルに設定されると、A が点灯し、スイッチ SA の電源がオンになり、ステッピングモーターが 1 ステップで回転する。同様のケースが何度も繰り返される。したがって、ステッピングモーターに特定の時系列を与える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


IN1
IN2
IN3
IN4
ULN2003
A
B
C
D
R5 R6 R7 R8
- +
5-12V

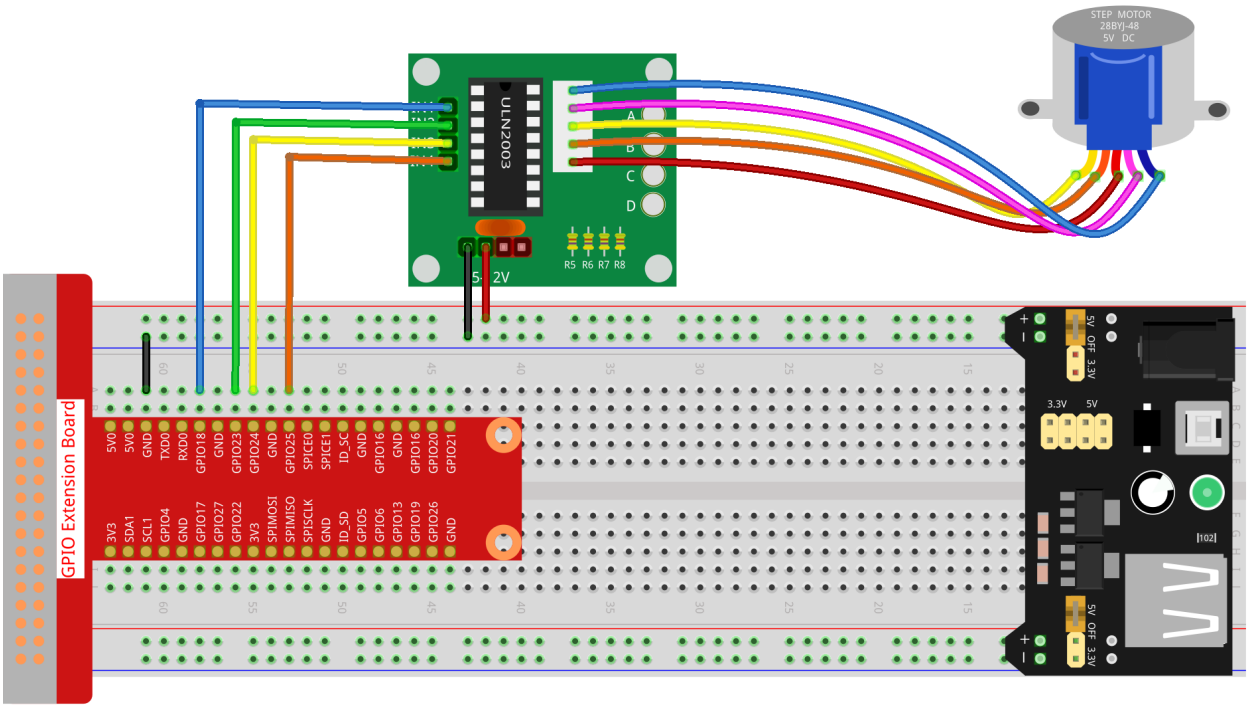
回路図

T-Board Name	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;
```

(次のページに続く)

(前のページからの続き)

```
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: ");
        delay(100);
        direction=getchar();
        if (direction == 'c')
        {
            printf("motor running clockwise\n");
            delay(100);
            break;
        }
        else if (direction == 'a')
        {
            printf("motor running anti-clockwise\n");
            delay(100);
            break;
```

(次のページに続く)

(前のページからの続き)

```
        }
        else
        {
            printf("input error, please try again!\n");
            delay(100);
        }
    }
    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。 `(60000000/rolePerMinute)/stepsPerRevolution` （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 rotary(direction) 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);
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
    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

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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'):
```

(次のページに続く)

(前のページからの続き)

```
            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
```

(次のページに続く)

(前のページからの続き)

```
        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 rotary(direction) を呼び出して、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

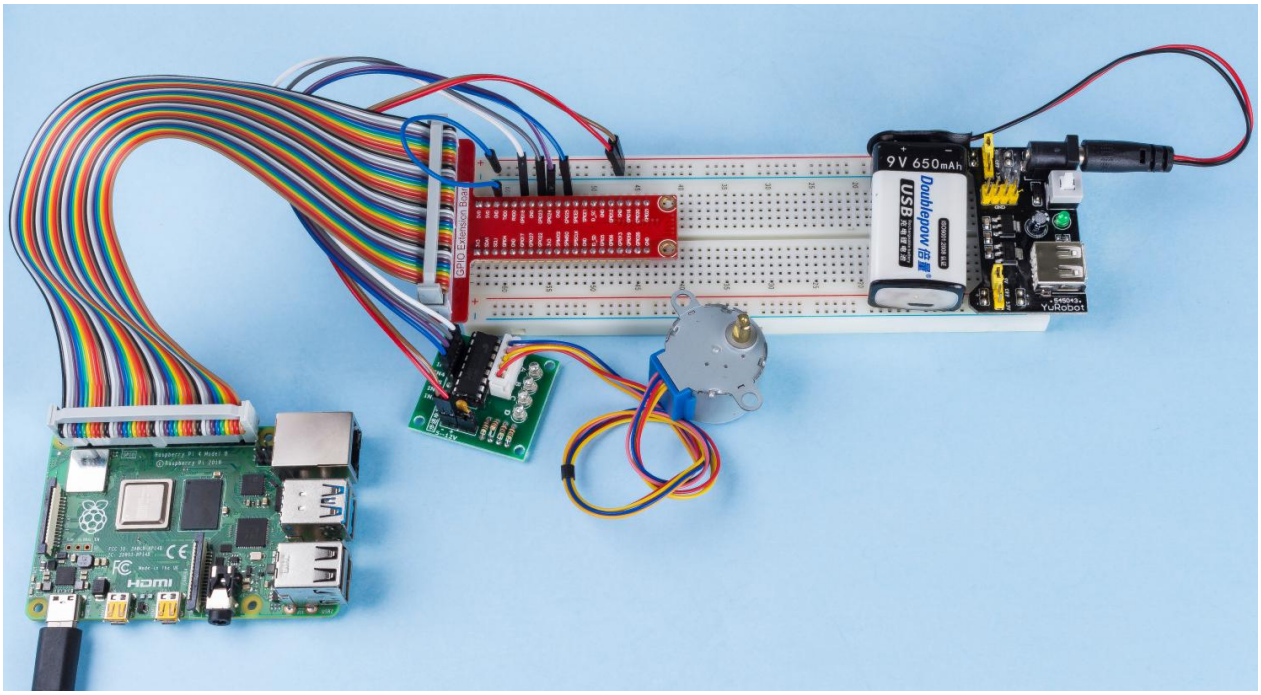
ステップ 2 では、j=1、i=0～4。

`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

まれる。

現象画像

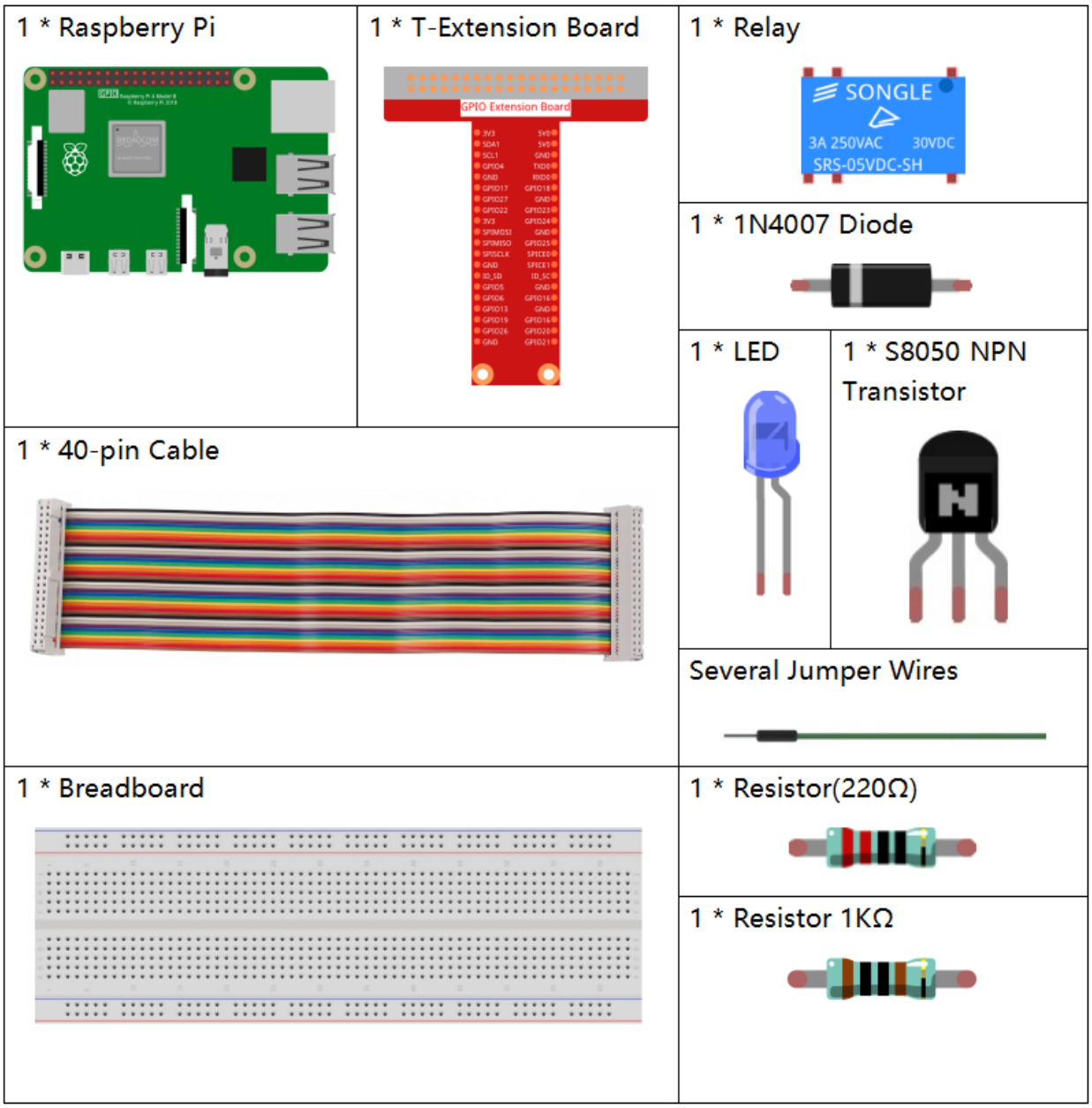


1.8.13 1.3.4 リレー

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。

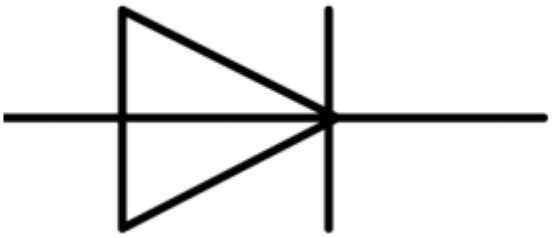
部品



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る。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 つのパーツがある：

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。

電機子 - 可動磁気ス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。

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。

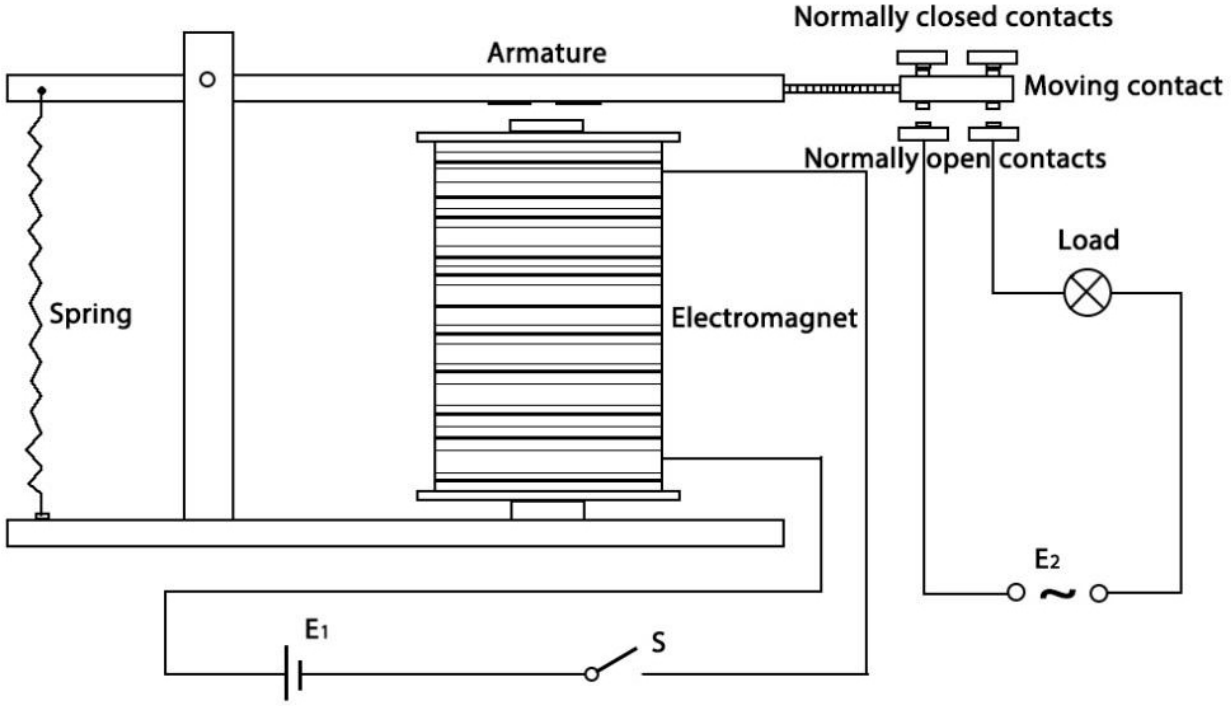
電気 **接点** のセット-2 つの接点がある：

- 通常開-リレーがアクティブなときに接続され、非アクティブなときに切断されます。
- 常閉-リレーが作動している時に接続しなく、リ

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。

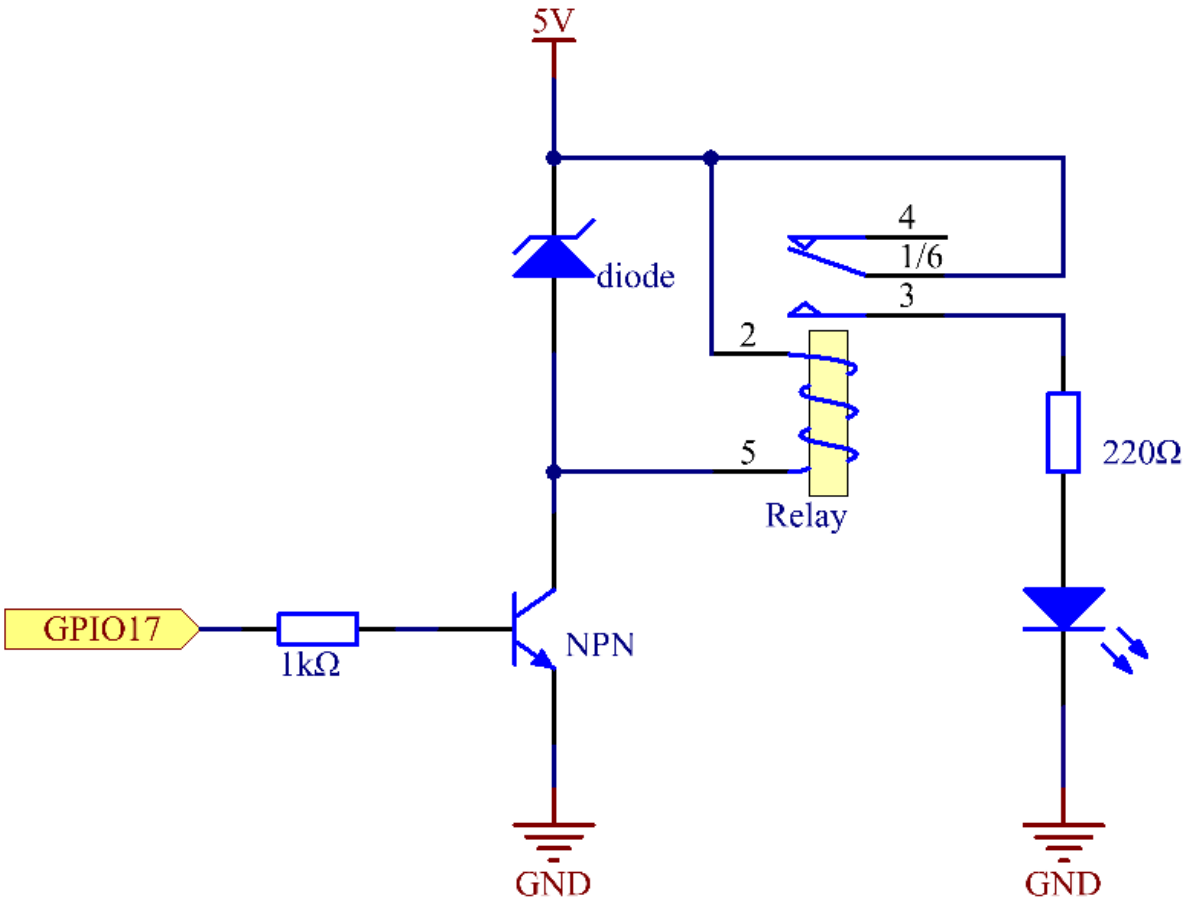
リレーの動作

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、可動接点が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。



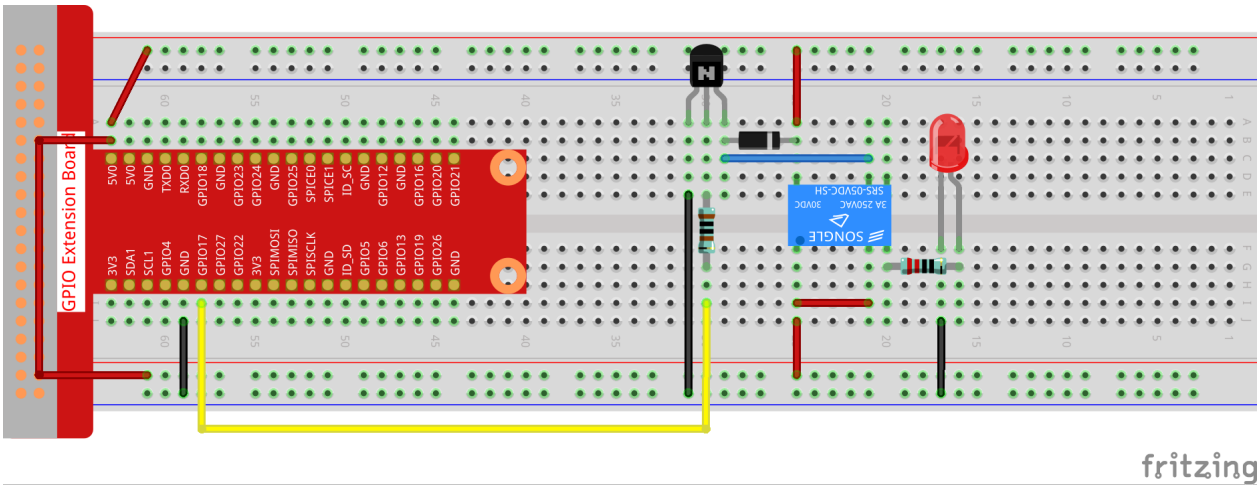
回路図

T-Board Name	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

```
#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
```

(次のページに続く)

(前のページからの続き)

```
    while(1){
        // Tick
        printf("Relay Open......\n");
        delay(100);
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
        printf("......Relay Close\n");
        delay(100);
        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 ユーザーの場合

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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()
```

(次のページに続く)

(前のページからの続き)

```
# 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。

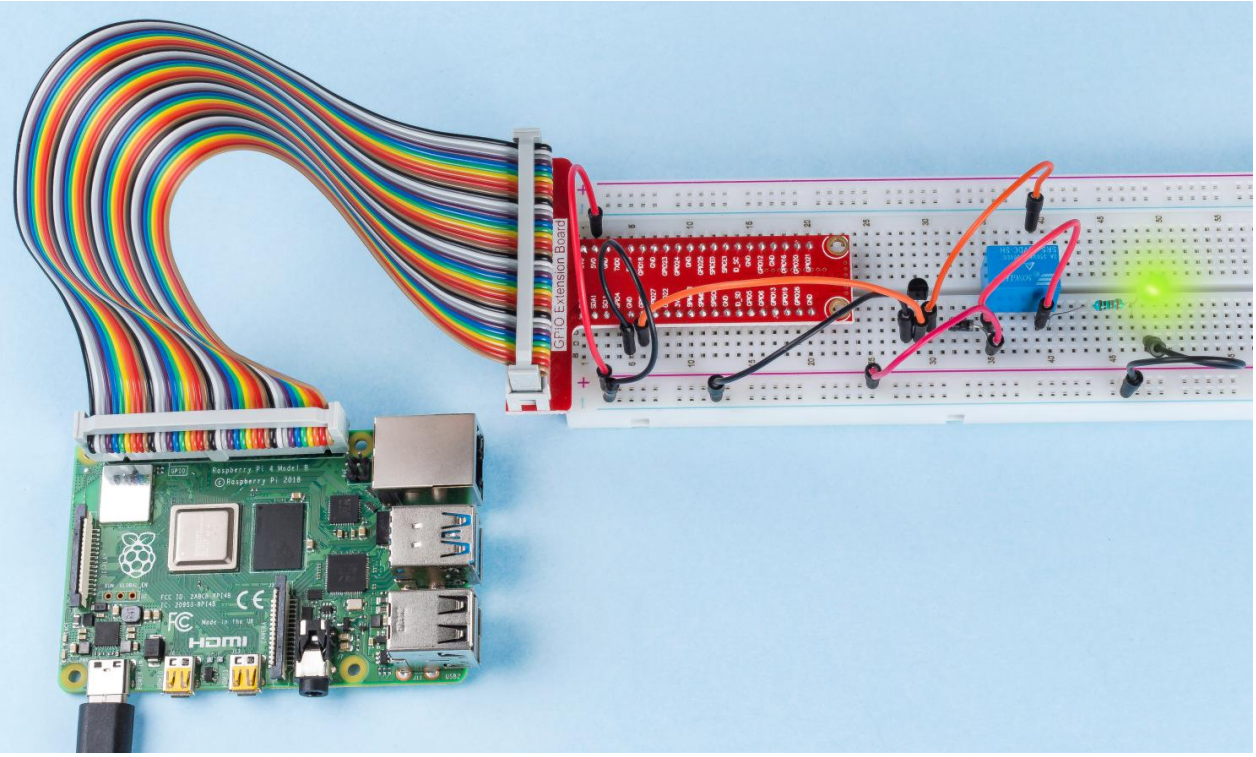
```
time.sleep(1)
```

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



1.9 2 入力 (Pi 5 用ではない)

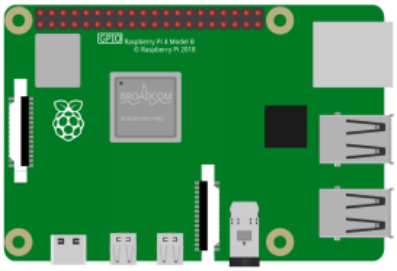
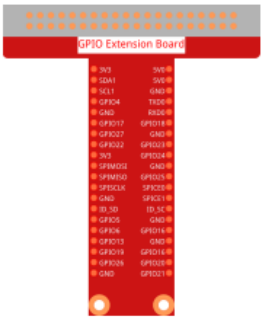
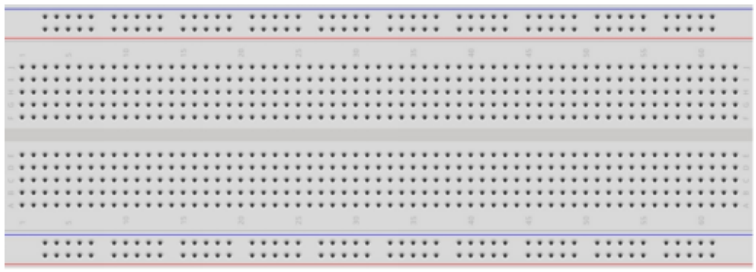
2.1 コントローラー

1.9.1 2.1.1 ボタン

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

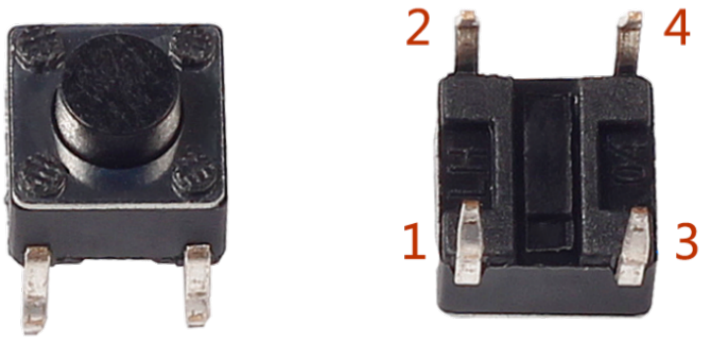
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED
		1 * Resistor 10KΩ
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * Button

原理

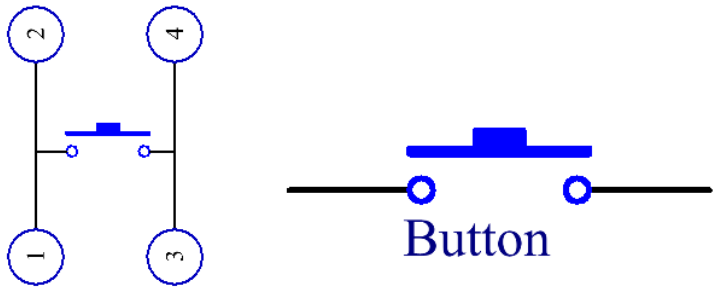
ボタン

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。

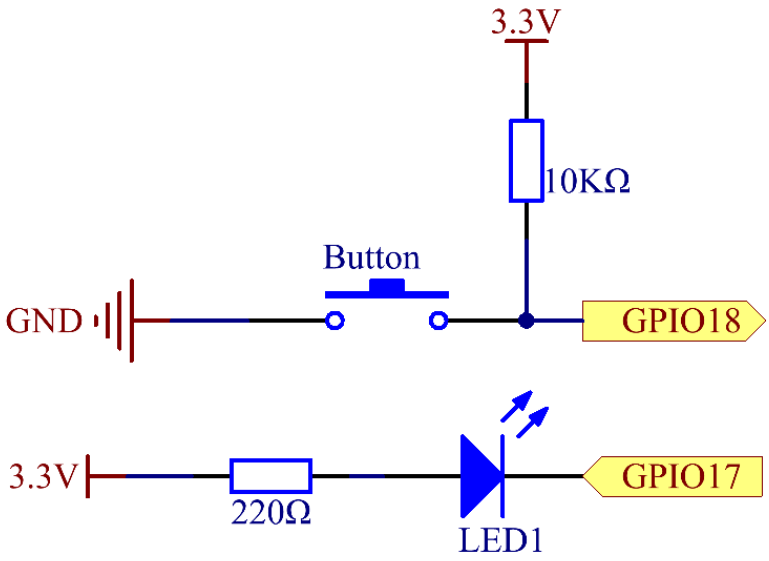


ボタンを押すと、4 つのピンが接続され、回路が閉じる。

回路図

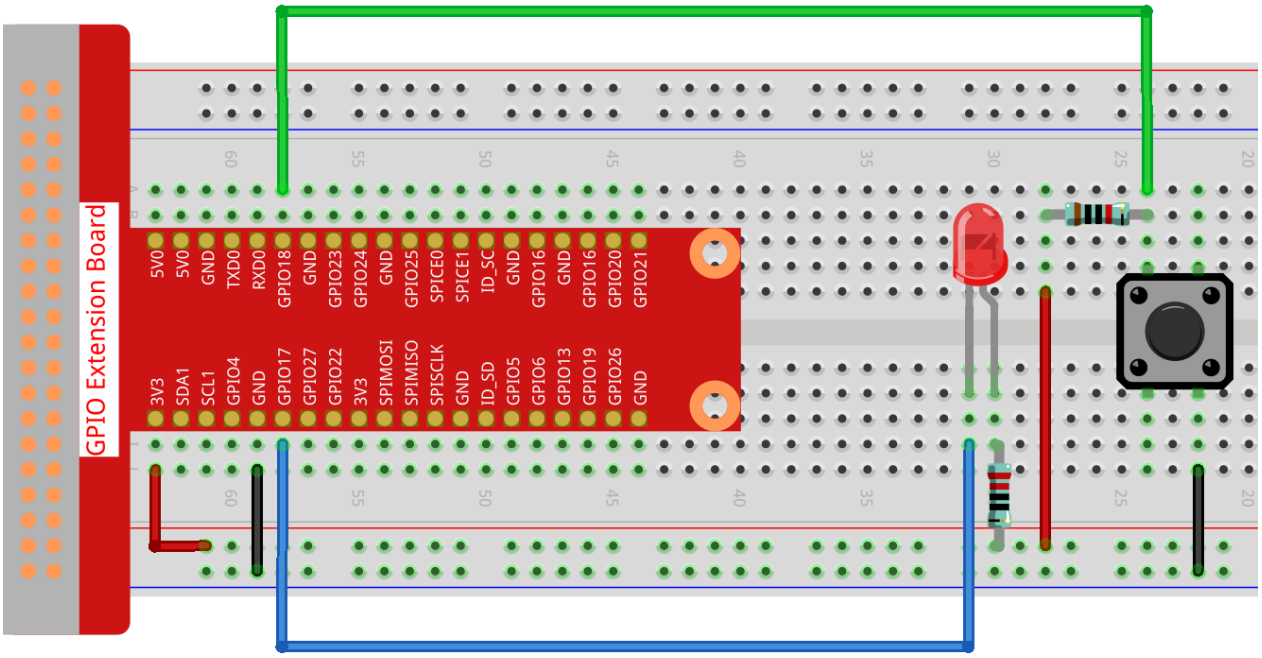
Raspberry Pi の入力として常開ボタンを使用し、接続は下の概略図に示されている。ボタンを押すと、GPIO18 は低レベル（0V）に変わる。プログラミングによって GPIO18 の状態を検出できる。つまり、GPIO18 が低レベルになった場合、ボタンが押されたことを意味する。ボタンが押されたときに対応するコードを実行すると、LED が点灯する。

T-Board Name	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/
```

注釈: 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){
```

(次のページに続く)

(前のページからの続き)

```
            // 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
```

(次のページに続く)

(前のページからの続き)

```
            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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

LedPin = 17 # Set GPIO17 as LED pin
BtnPin = 18 # Set GPIO18 as button pin


Led_status = True # Set Led status to True(OFF)

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
```

(次のページに続く)

(前のページからの続き)

```
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)  # Set LedPin's mode to output, and
↪initial level to high (3.3v)
    GPIO.setup(BtnPin, GPIO.IN) # Set BtnPin's mode to input.

# 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.
```

(次のページに続く)

(前のページからの続き)

```
    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。

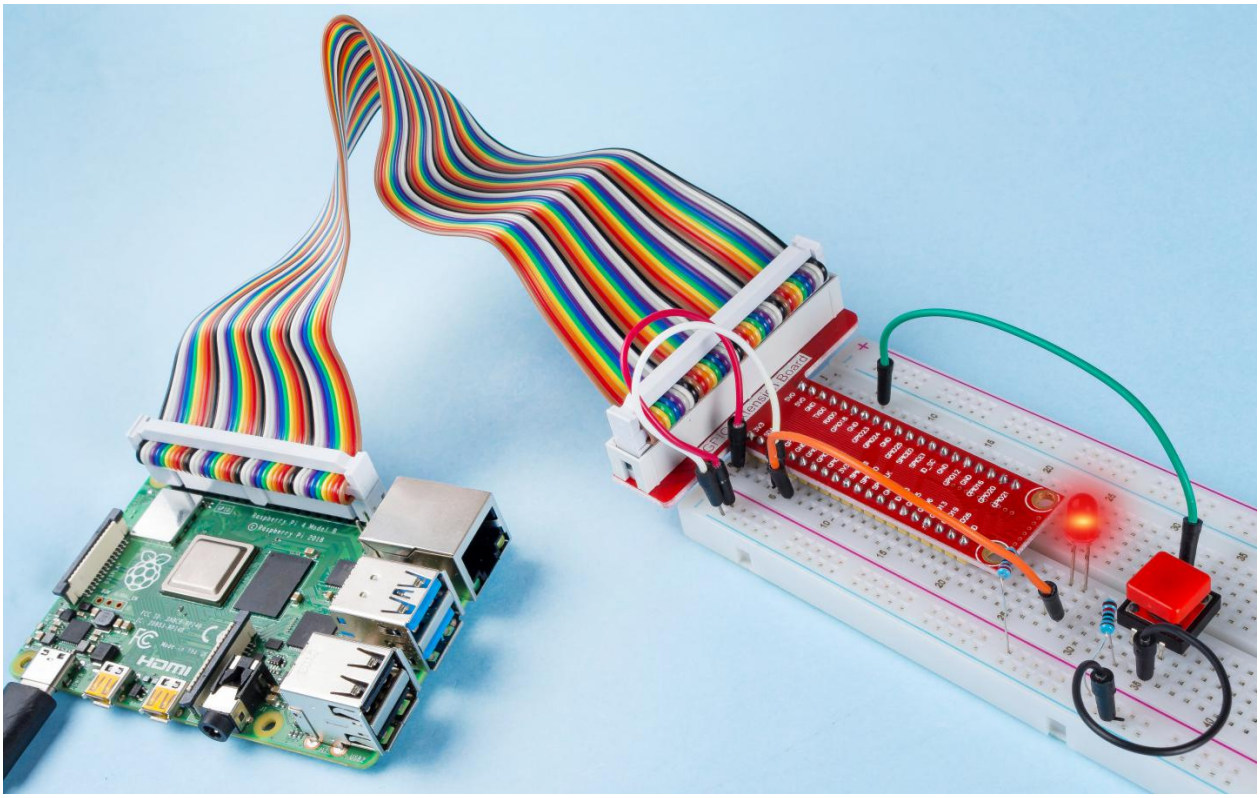
```
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

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

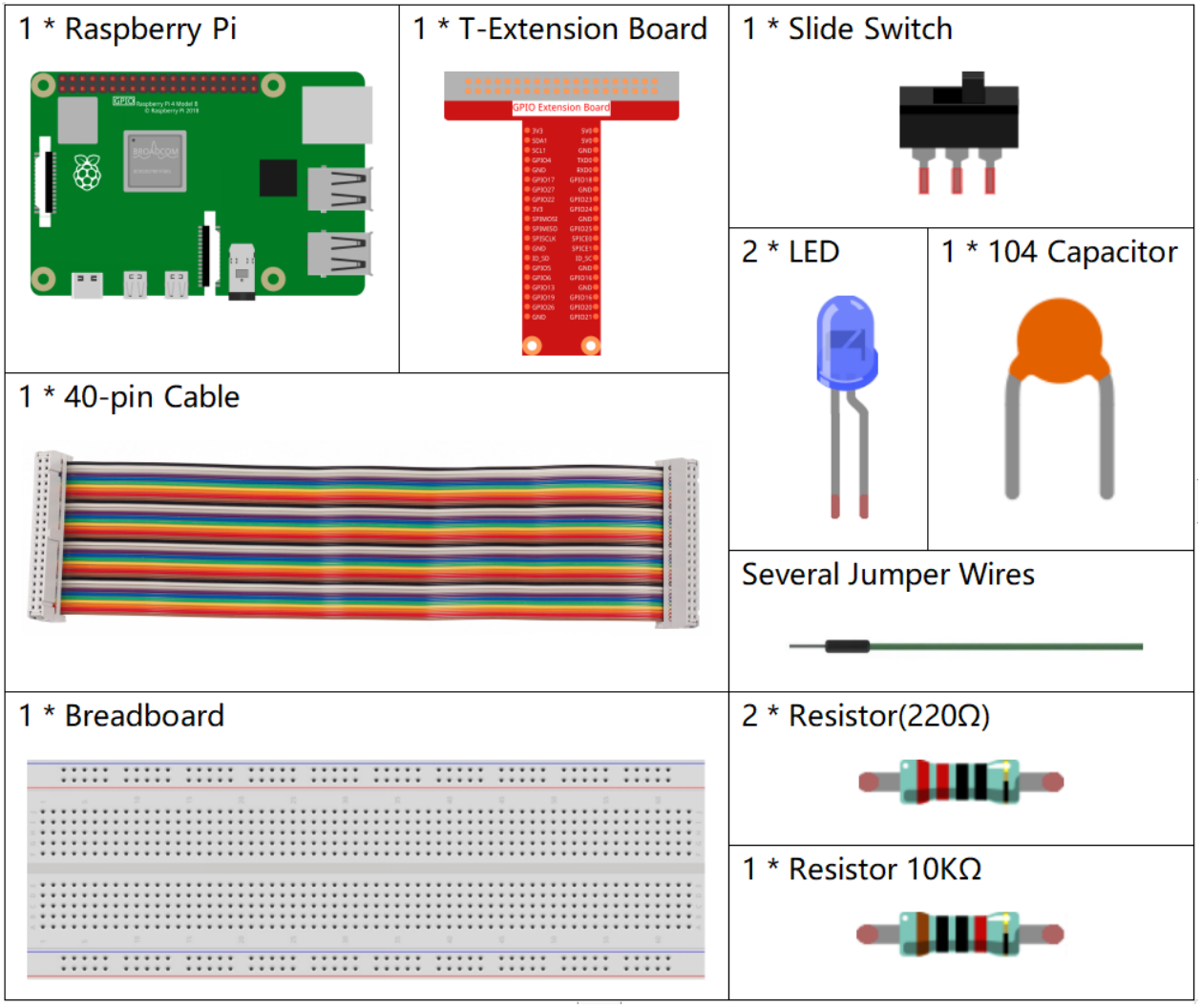


1.9.2 2.1.2 スライドスイッチ

前書き

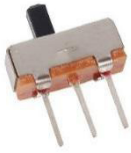
このレッスンでは、スライドスイッチの使用方法を学習する。通常、スライドスイッチは電源スイッチとして PCB にはんだ付けされるが、ここではブレッドボードに挿入する必要があるため、締め付けられない場合がある。そして、その機能を示すためにブレッドボードで使用する。

部品



原理

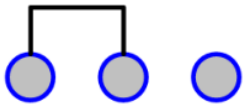
スライドスイッチ

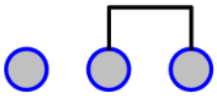


スライドスイッチは名前が示すように、スイッチバーをスライドさせて回路を接続または遮断し、さらに回路を切り替えるものである。汎用タイプは、SPDT、SPTT、DPDT、DPTT などである。スライドスイッチは低電圧回路で一般的に使用されている。融通性と安定性の特徴を備えており、電気機器や電気玩具に広く適用されている。

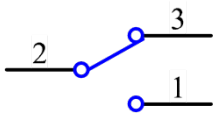
仕組み：中央のピンを固定ピンとして設定する。スライドを左に引くと、左の 2 つのピンが接続され、右に引くと、右側の 2 つのピンが接続される。したがって、回路を接続または遮断するスイッチとして機能する。以下の図

を参照してください：





スライドスイッチの回路記号を以下に示す。図のピン 2 は中央のピンを指す。



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 (μ F) 1 μ F = 1/1,000,000 = 0.000001 = 10^{-6} F

Nanofarad (nF) 1nF = 1/1,000,000,000 = 0.000000001 = 10^{-9}F

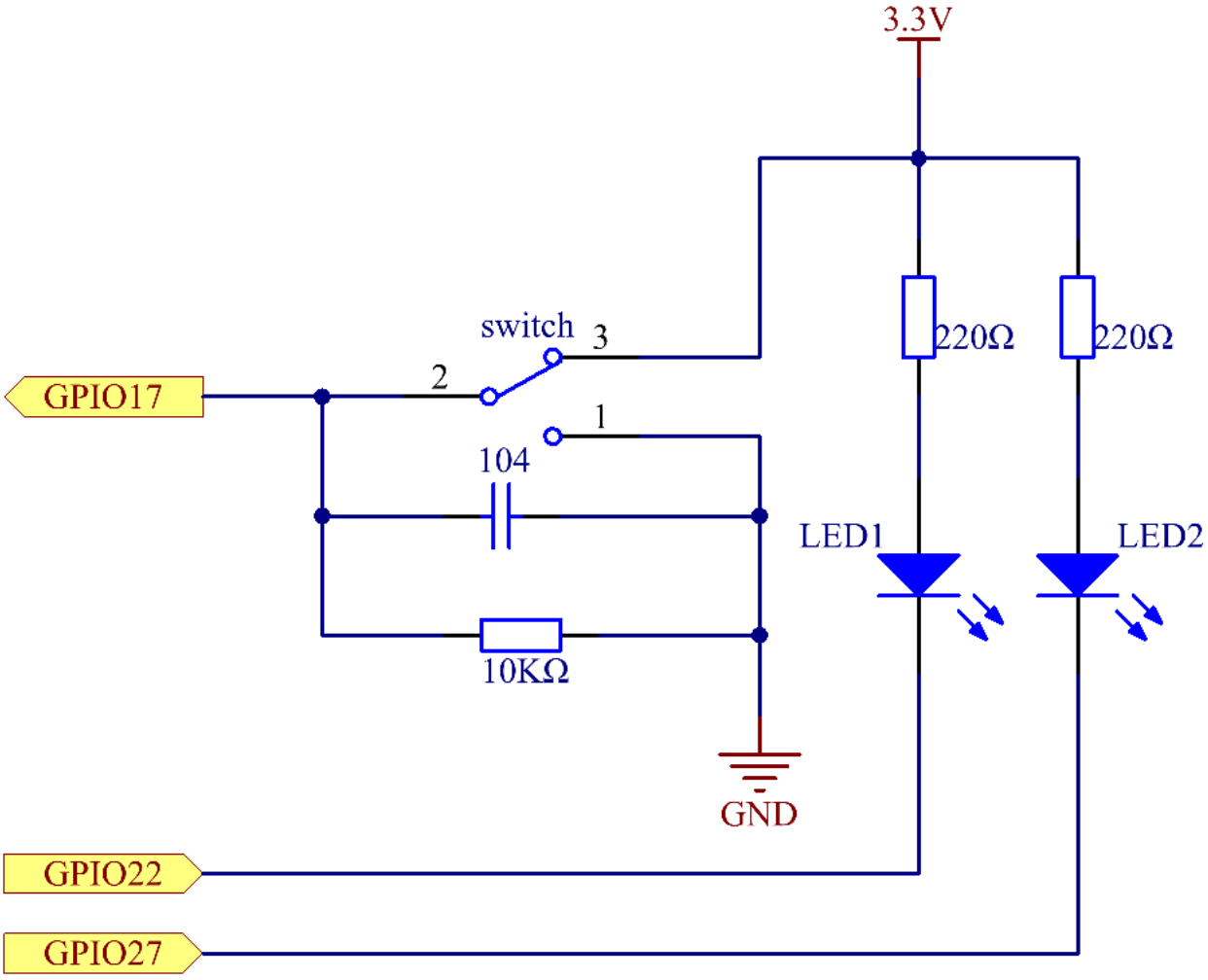
Picofarad (pF) 1pF = 1/1,000,000,000,000 = 0.000000000001 = 10^{-12}F

注釈: ここでは、**104 コンデンサ** (10 x 10^{4}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（pF 単位）。

回路図

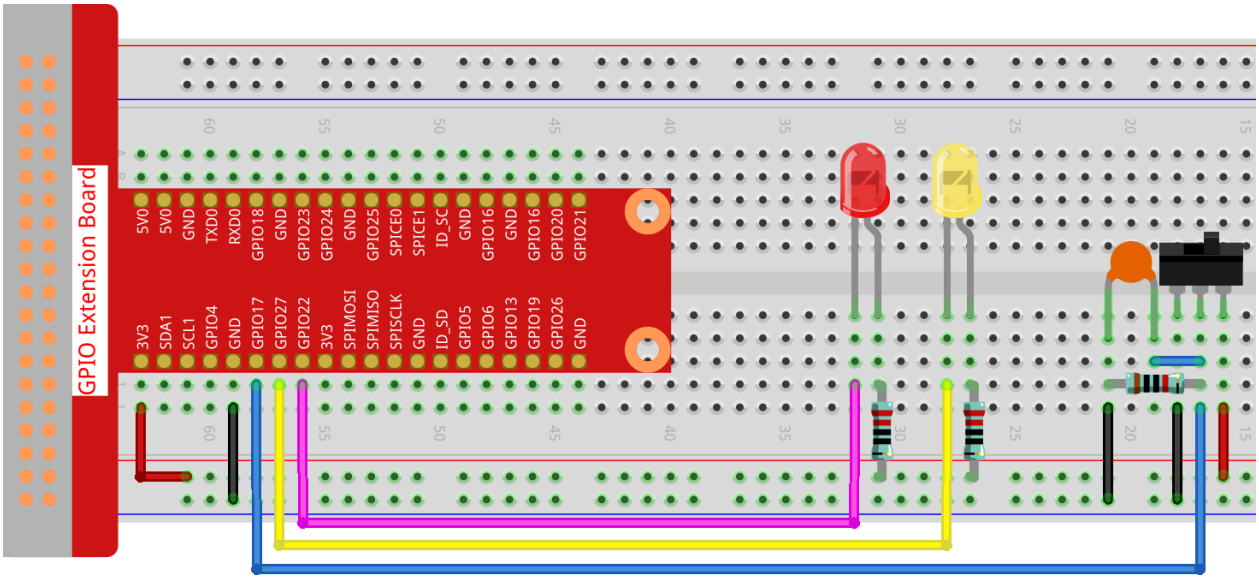
スライドスイッチの中央のピンを GPIO17 に接続し、2 つの LED をそれぞれ GPIO22 と GPIO27 に接続する。次に、スライドを引くと、2 つの LED が交互に点灯する。

T-Board Name	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： コードのフォルダーに入る。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.2
```

ステップ 3： コンパイルする。

```
gcc 2.1.2_Slider.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、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

```
#include <wiringPi.h>
#include <stdio.h>
#define slidePin        0
#define led1            3
#define led2            2

int main(void)
```

(次のページに続く)

(前のページからの続き)

```
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    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");
            delay(100);
        }
        // slide switch low, led2 on
        if(digitalRead(slidePin) == 0){
            digitalWrite(led2, LOW);
            digitalWrite(led1, HIGH);
            printf(".....LED2 on\n");
            delay(100);
        }
    }
    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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

# Set #17 as slide switch pin, #22 as led1 pin, #27 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)
```

(次のページに続く)

(前のページからの続き)

```
    # 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.
```

(次のページに続く)

(前のページからの続き)

```
    except KeyboardInterrupt:
        destroy()
```

コードの説明

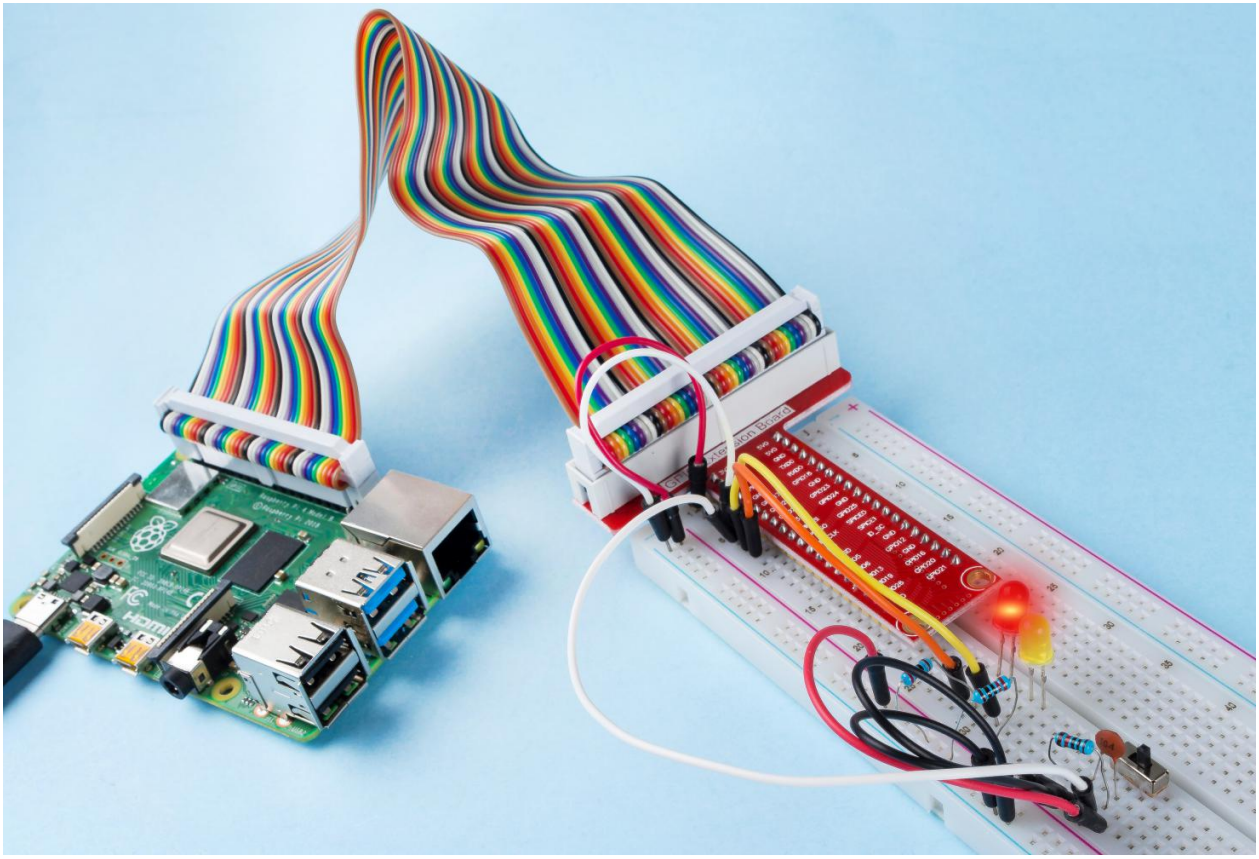
```
if GPIO.input(slidePin) == 1:
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.HIGH)
```

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

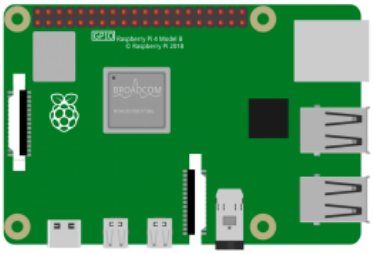
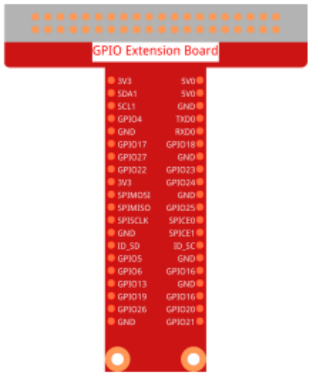
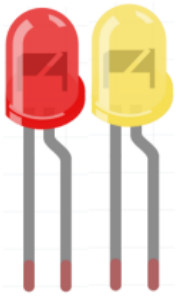
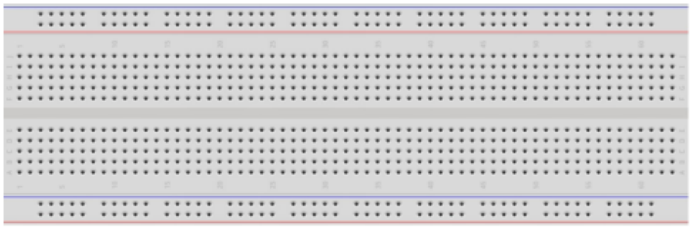


1.9.3 2.1.3 チルトスイッチ

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。

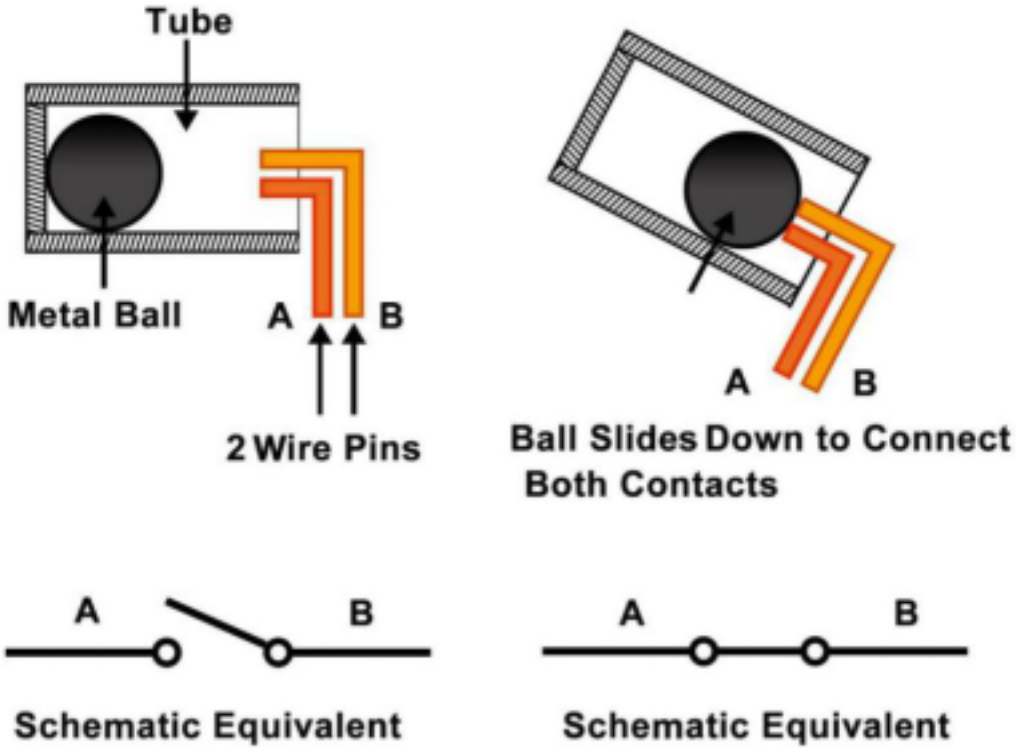
部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Tilt Switch	2 * LED 
1 * 40-pin Cable		Several Jumper Wires	
1 * Breadboard 		2 * Resistor(220Ω)	
		1 * Resistor 10KΩ	

原理

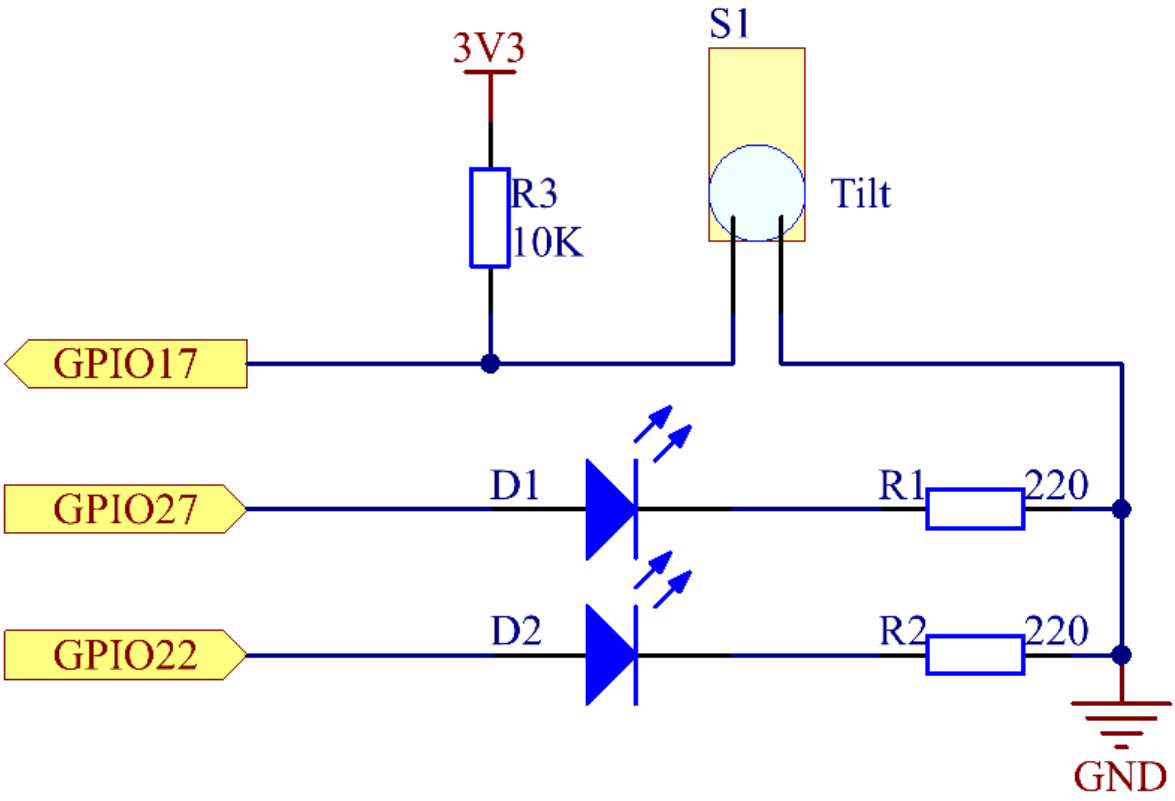
チルト

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。



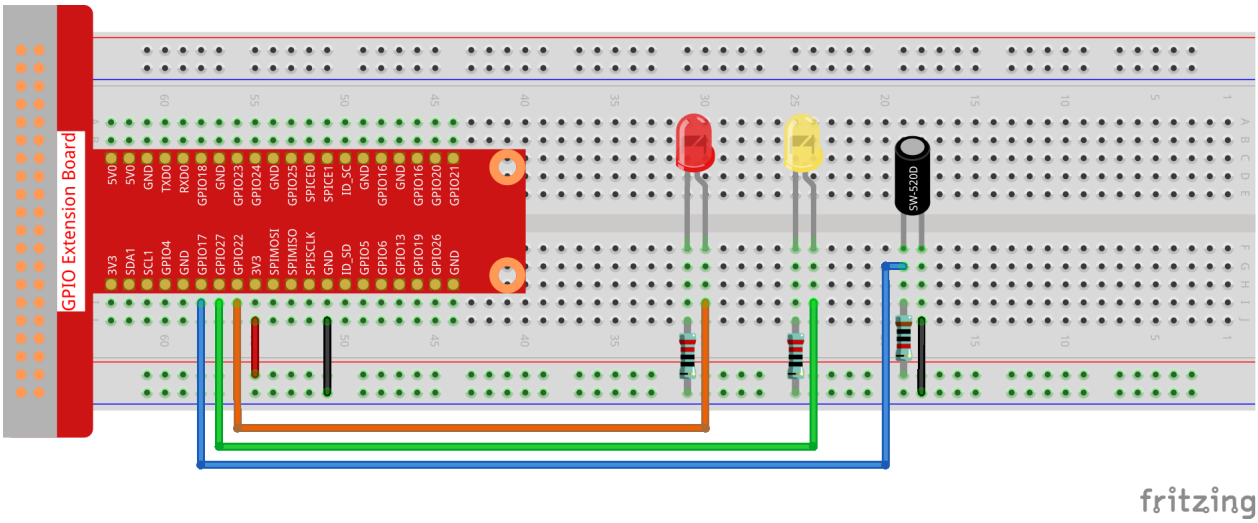
回路図

T-Board Name	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
```

(次のページに続く)

(前のページからの続き)

```
        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");
                delay(100);
            }
        }
        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")
```

(次のページに続く)

(前のページからの続き)

```
    {
        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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
    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:
```

(次のページに続く)

(前のページからの続き)

```
            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
```

(次のページに続く)

(前のページからの続き)

```
    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

```
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。

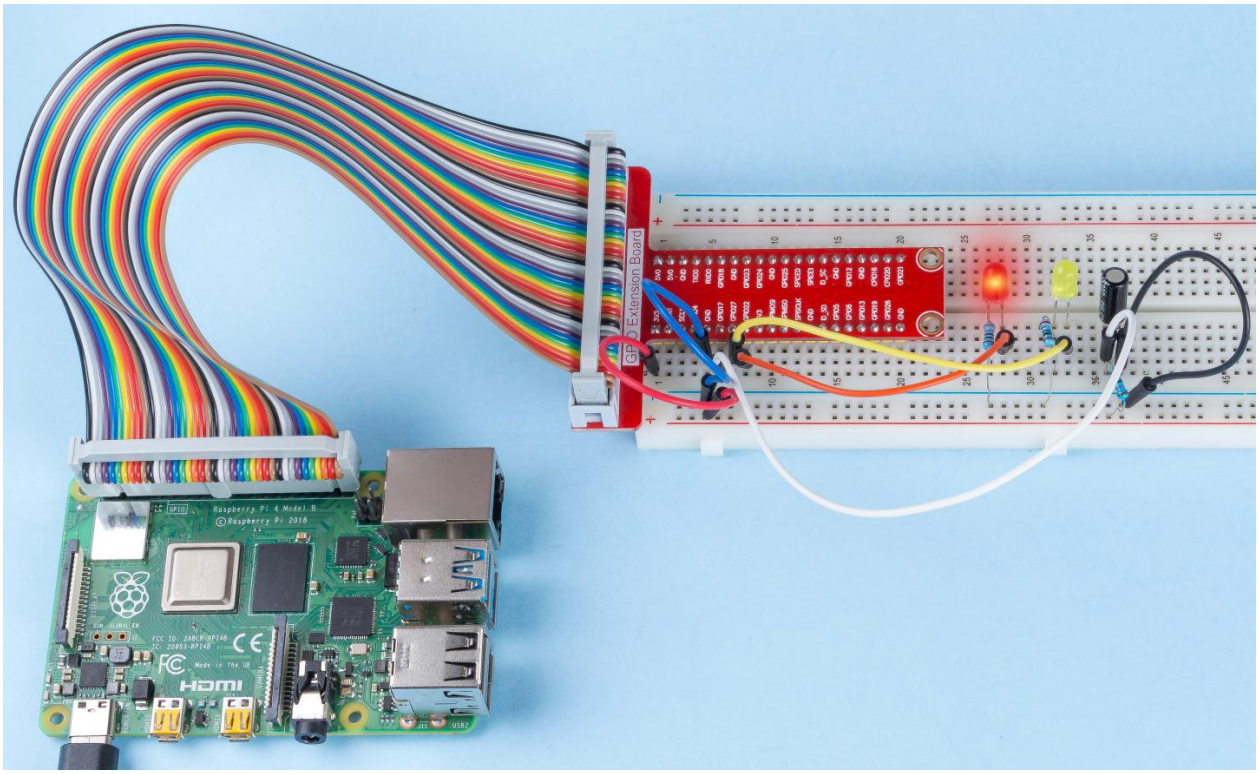
```
def Print(x):
    if x == 0:
        print ('    **************')
        print ('    *   Tilt!    *')
        print ('    **************')
```

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

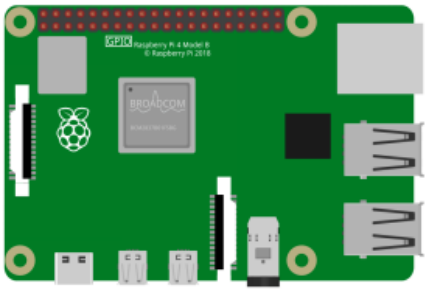
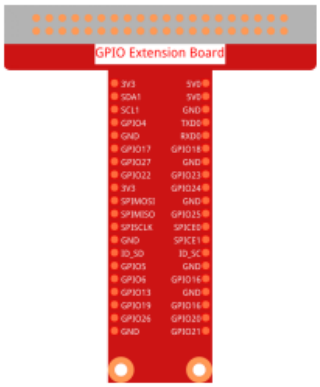
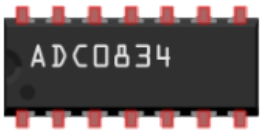
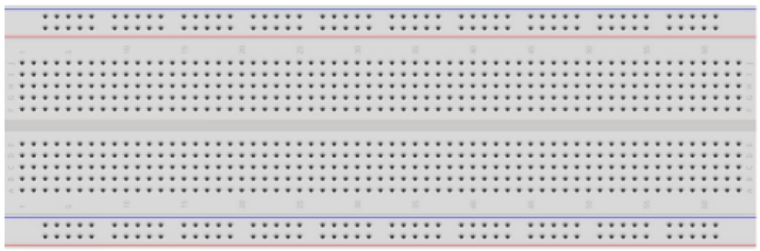


1.9.4 2.1.4 ポテンショメータ

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。ポテンショメーターは ADC 機能によって変換される物理量-電圧を変更する。

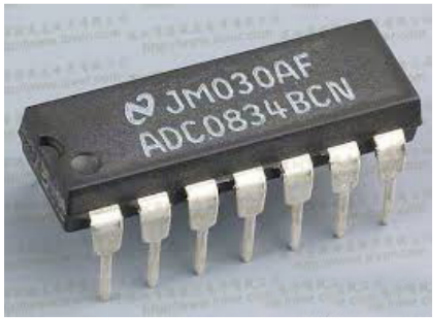
部品

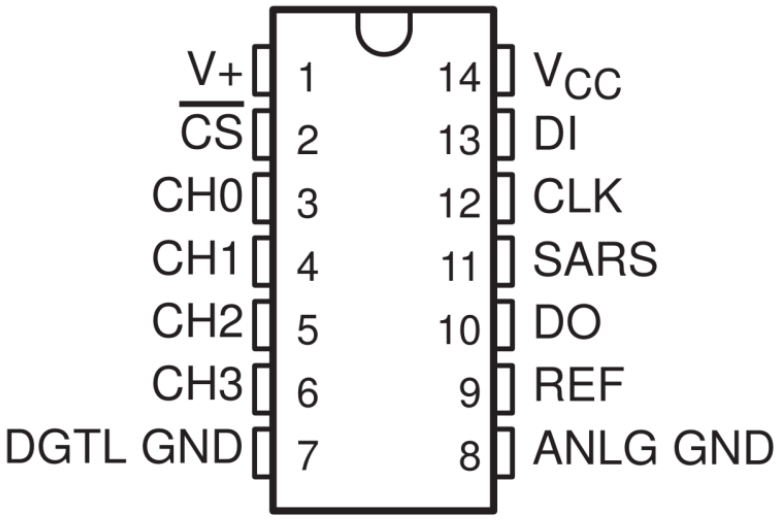
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Potentiometer
 </td></tr>
<tr><td rowspan="2">1 * 40-pin Cable
 </td><td>1 * ADC0834
 
</td></tr>
<tr><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td rowspan="2">1 * Breadboard
  </td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * LED
 </td></tr>
</table>

原理

ADC0834

ADC0834 は、入力構成可能なマルチチャンネル マルチプレクサとシリアル入出力を備えた 8 ビットの逐次比較型 A/D コンバータです。シリアル入出力は、標準的なシフトレジスターやマイクロプロセッサーとのインターフェースとして構成されています。



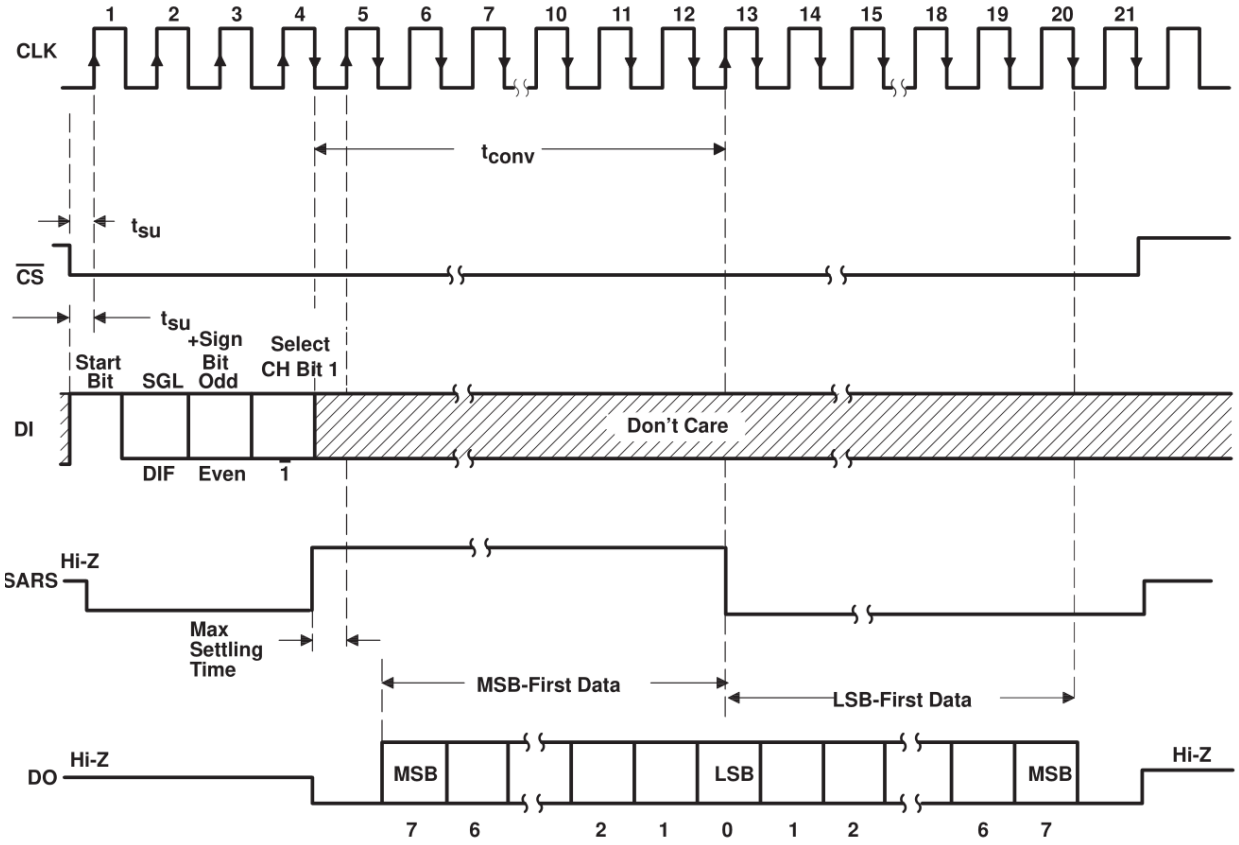


操作の順序

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アドレス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 Statu 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出力され、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。



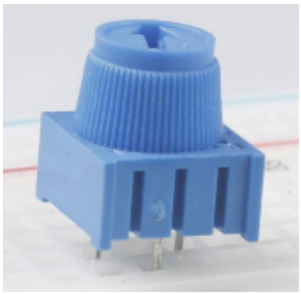
ADC0834 MUX アドレス制御論理表

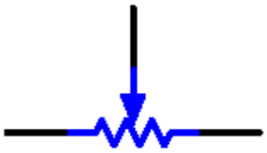
MUX ADDRESS			CHANNEL NUMBER			
SGL/$\overline{\text{DIF}}$	ODD/$\overline{\text{EVEN}}$	SELECT BIT 1	0	1	2	3
L	L	L	+	−		
L	L	H			+	−
L	H	L	−	+		
L	H	H			−	+
H	L	L	+			
H	L	H			+	
H	H	L		+		
H	H	H				+

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。





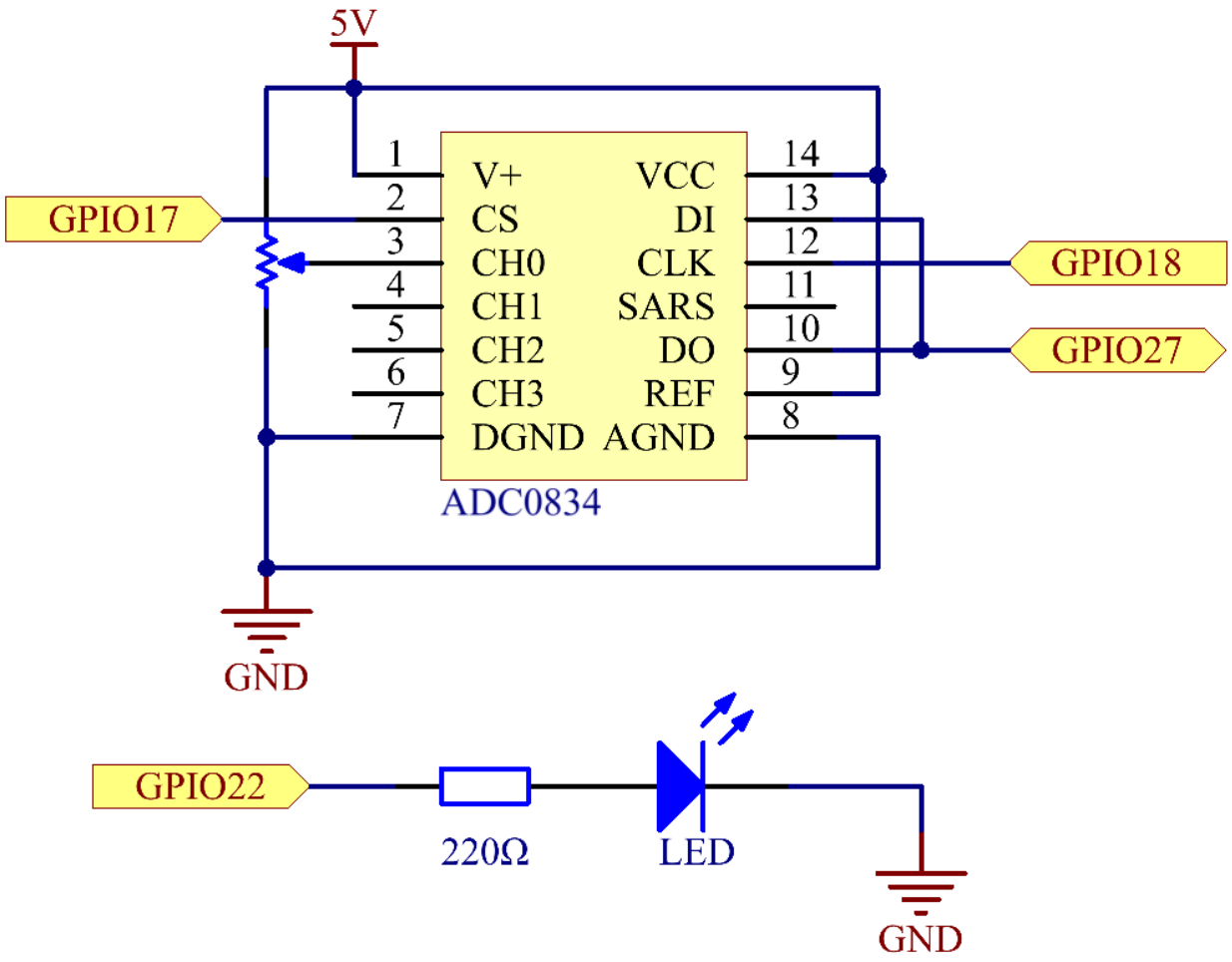
回路内のポテンショメータの機能は次のとおりである：

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。

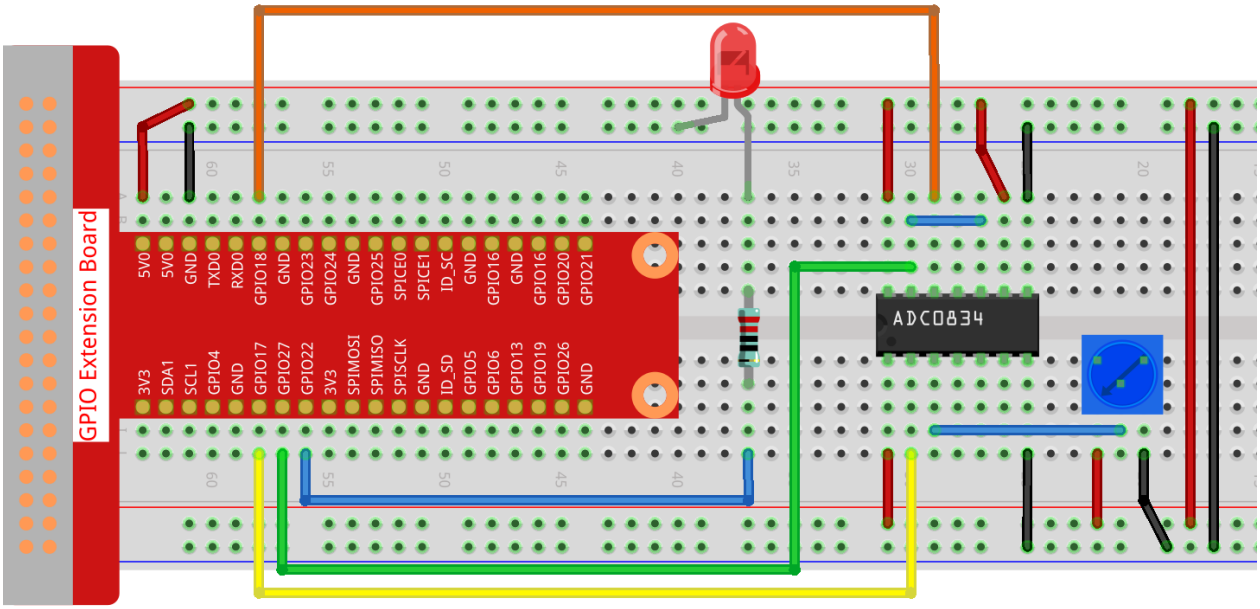
回路図

T-Board Name	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。



注釈: 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>
```

(次のページに続く)

(前のページからの続き)

```
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);
```

(次のページに続く)

(前のページからの続き)

```
    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 messageto 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);
        delay(100);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
```

(次のページに続く)

(前のページからの続き)

```
    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);
```

(次のページに続く)

(前のページからの続き)

```
    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);
```

(次のページに続く)

(前のページからの続き)

```
igitalWrite(ADC_DIO,1);    delayMicroseconds(2);
gitalWrite(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/$\overline{\text{DIF}}$	ODD/$\overline{\text{EVEN}}$	SELECT BIT 1	0	1	2	3
L	L	L	+	−		
L	L	H			+	−
L	H	L	−	+		
L	H	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 x 100us 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 プログラムで、 ポテンショメーターに最初接続されているチャンネル 0 0 の値 を 読み取る 。そして、最初値を最初変数 analogVal に保存してから LedPin に書き込む。これで、ポテンショメータの値によって LED の輝度が変化することを見ることができる。

Python 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequece 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

```
import ADC0834
```

ADC0834 ライブラリをインポートする。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)
```

(次のページに続く)

(前のページからの続き)

```
    ADC0834.setup()
    # Set led as pwm channel and frequece 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。

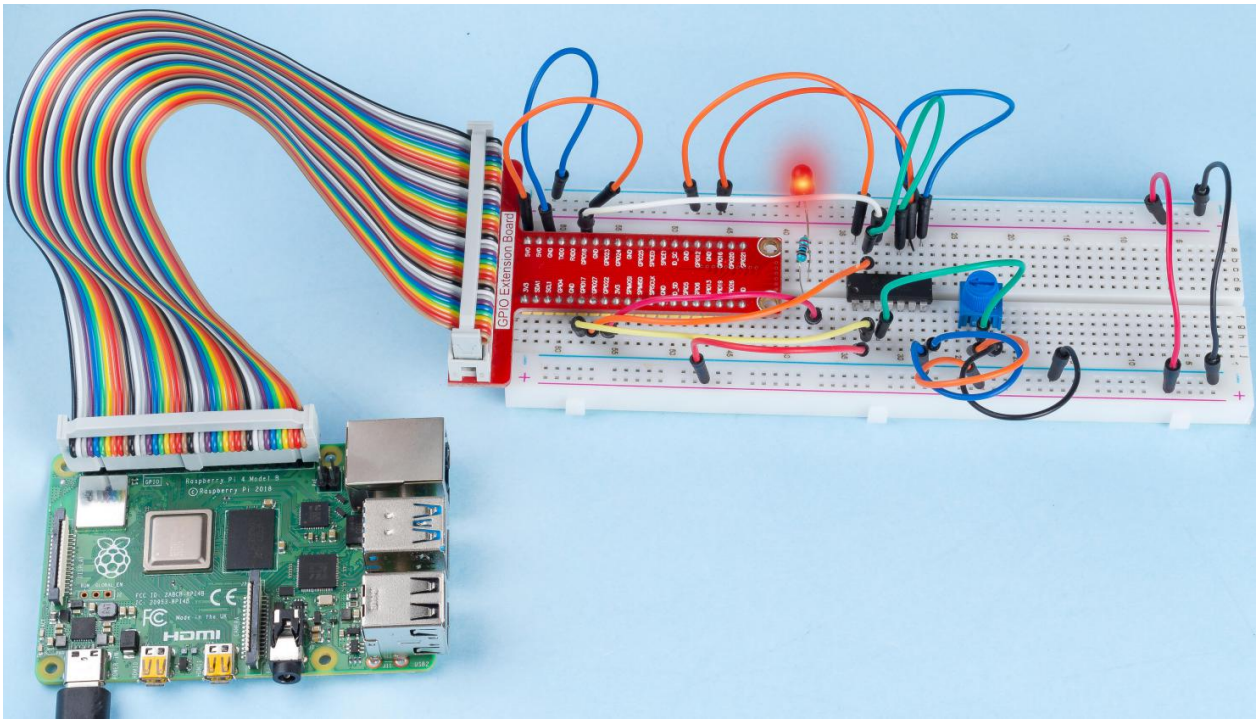
ADC0834.setup(): ADC0834 を初期化し、ADC0834 の定義された CS、CLK、DIO をそれぞれ GPIO17、GPIO18、GPIO27 に接続する。

```
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

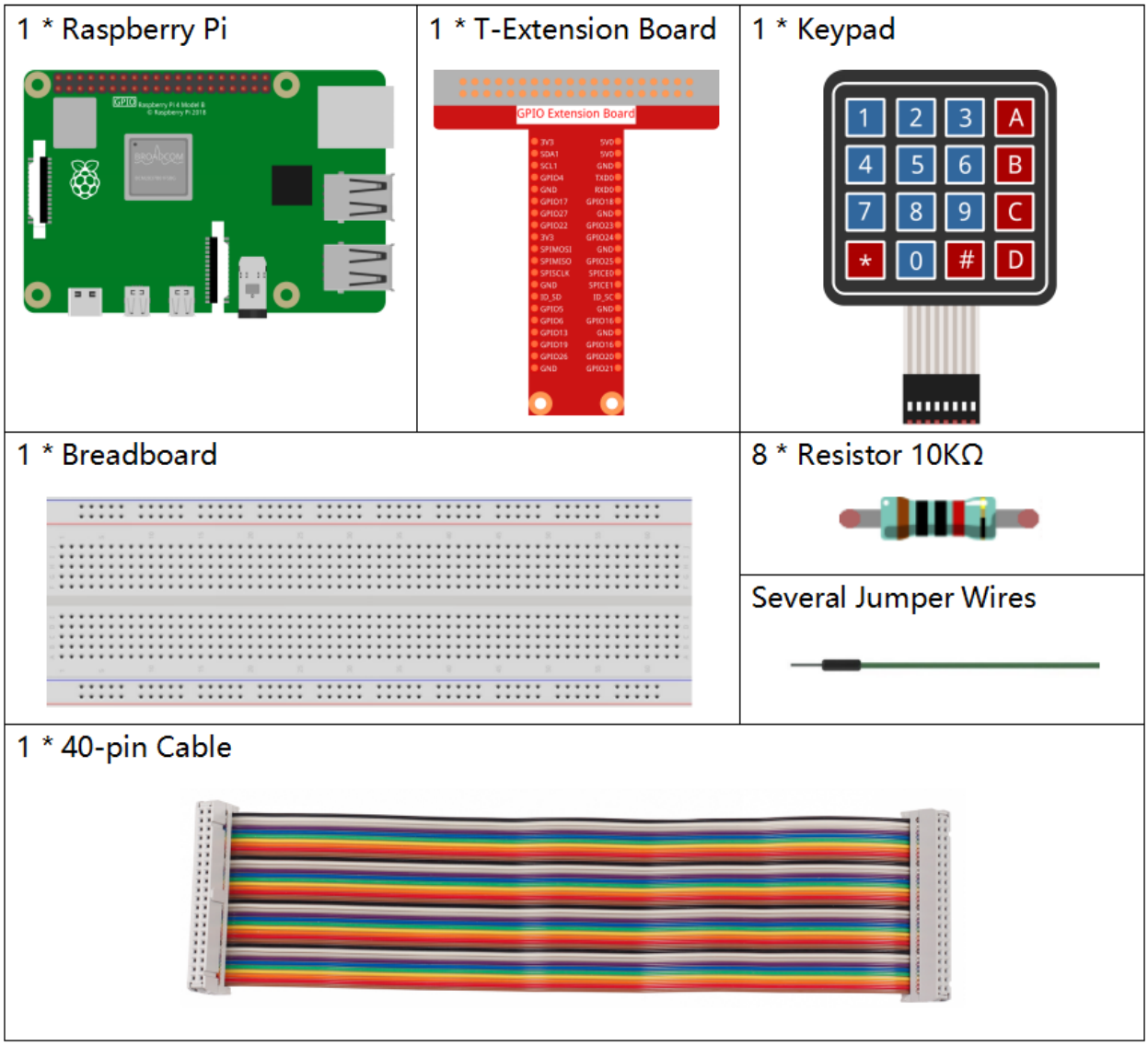


1.9.5 2.1.5 キーパッド

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。

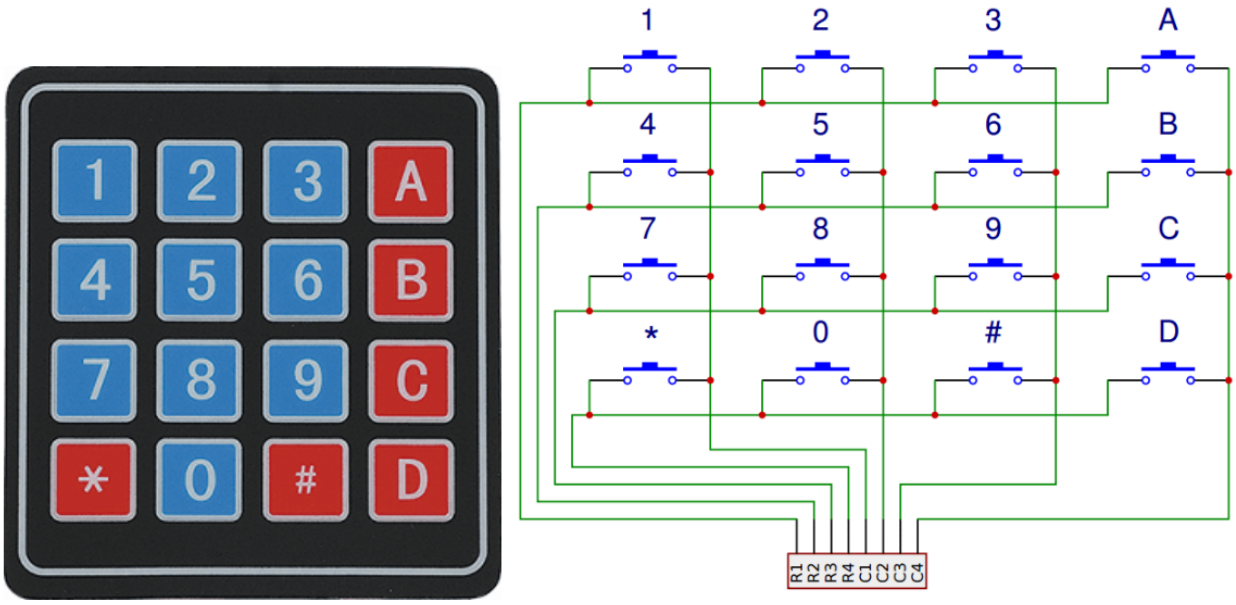
部品



原理

キーパッド

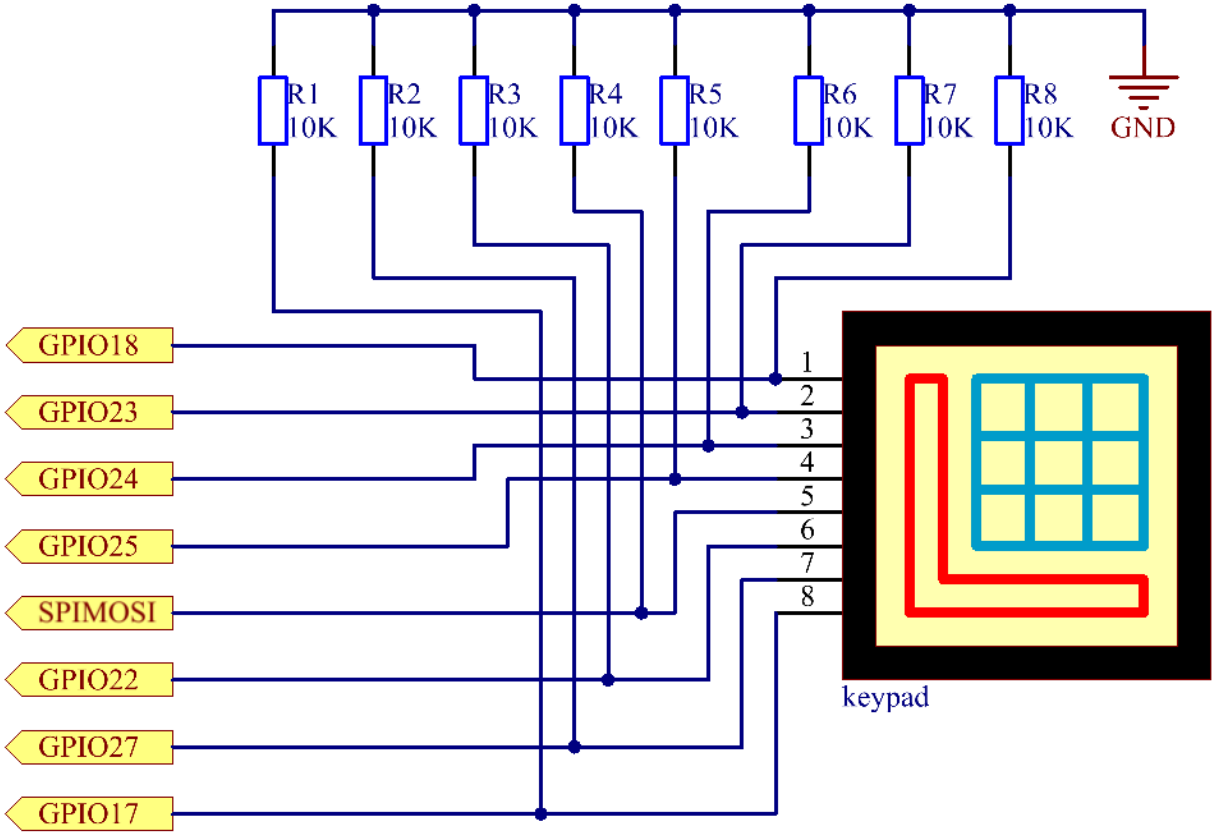
キーパッドは 12 個または 16 個の OFF-（ON）ボタンの長方形配列である。リボンケーブルとの接続またはプリント基板への挿入に適したヘッダーを介してそれらの接点にアクセスする。一部のキーパッドでは、各ボタンはヘッダーの個別の連絡先に接続されるが、すべてのボタンは共通の接地を共有する。



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ー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。

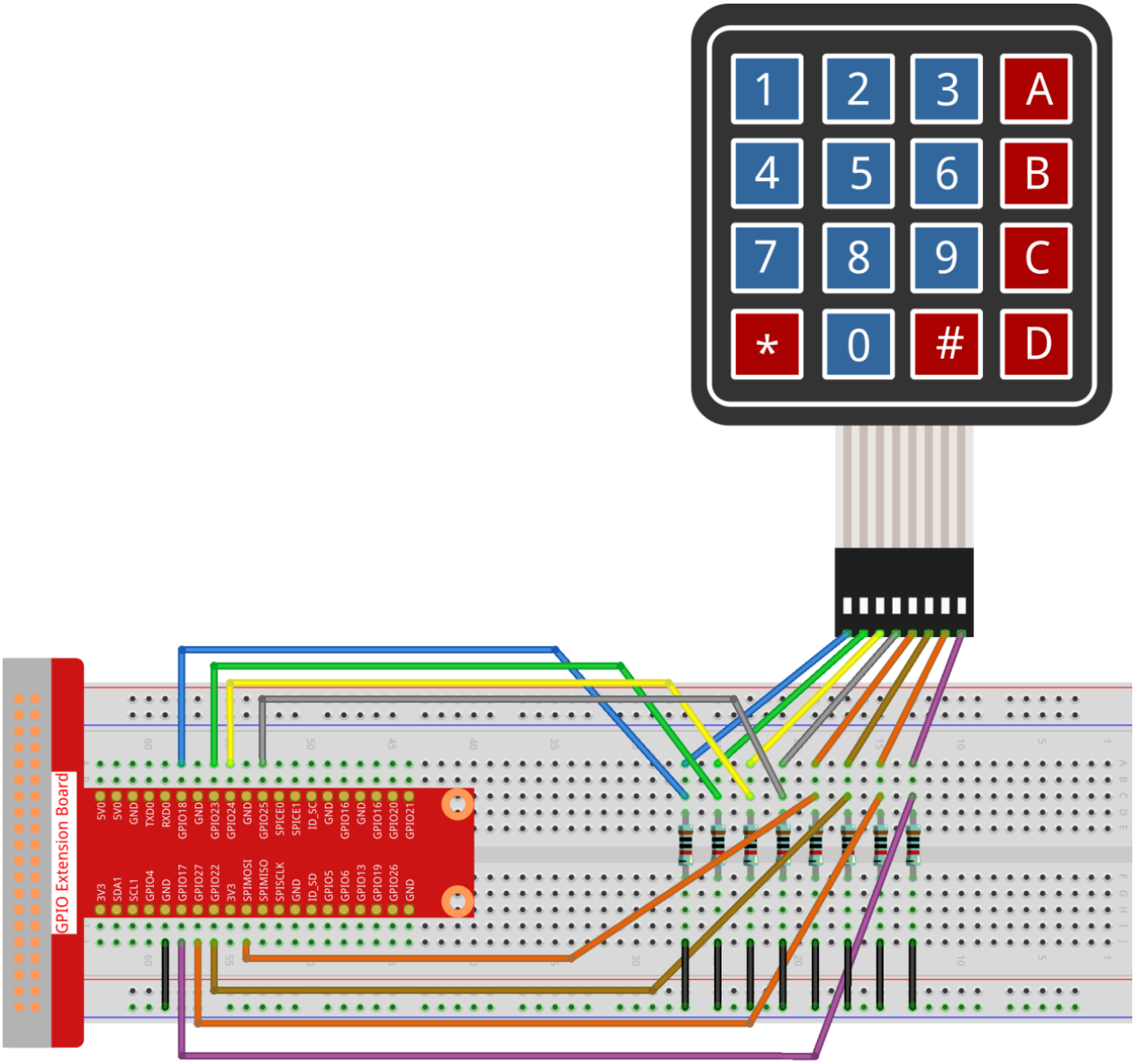
回路図

T-Board Name	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



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];
```

(次のページに続く)

(前のページからの続き)

```
    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;
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
    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;
```

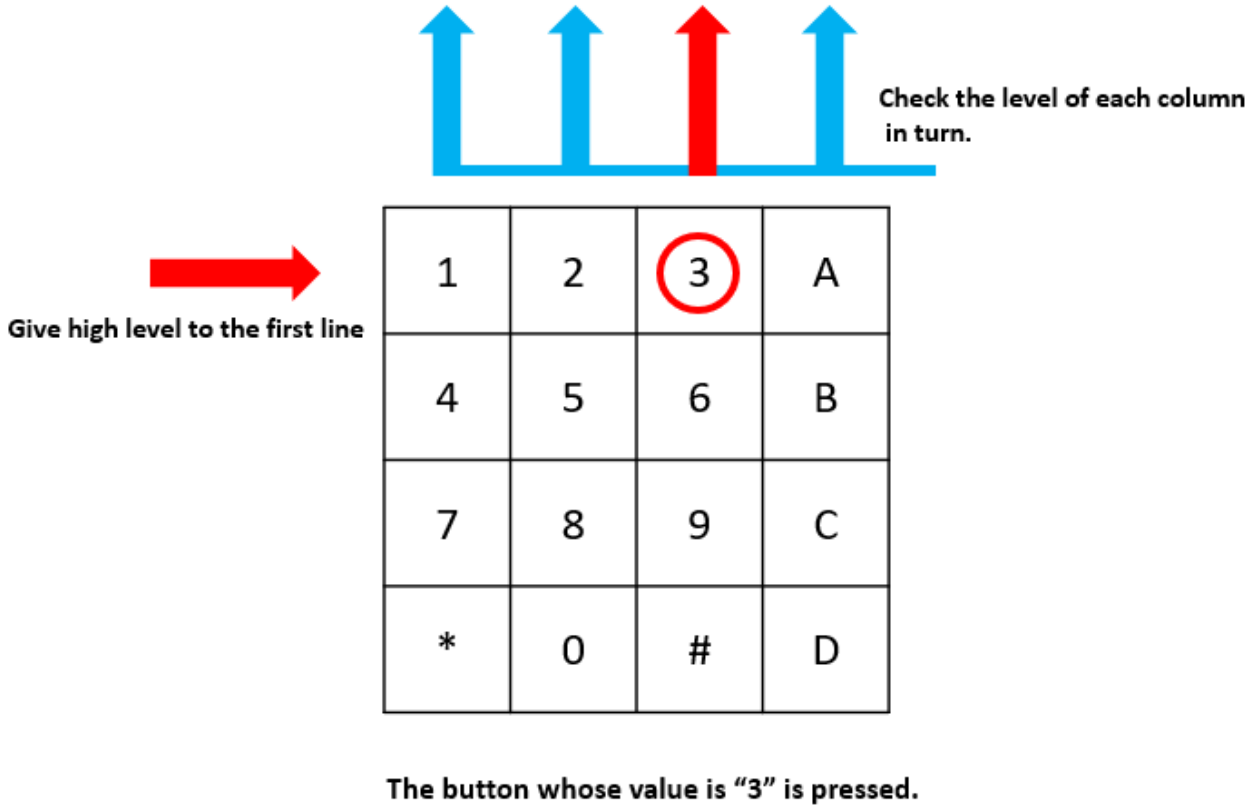
(次のページに続く)

(前のページからの続き)

```
            }
        }
        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（ reasult[] ）が生成される。

ボタン 3 を押すとき：



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] {
```

(次のページに続く)

(前のページからの続き)

```
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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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):
```

(次のページに続く)

(前のページからの続き)

```
                    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):
```

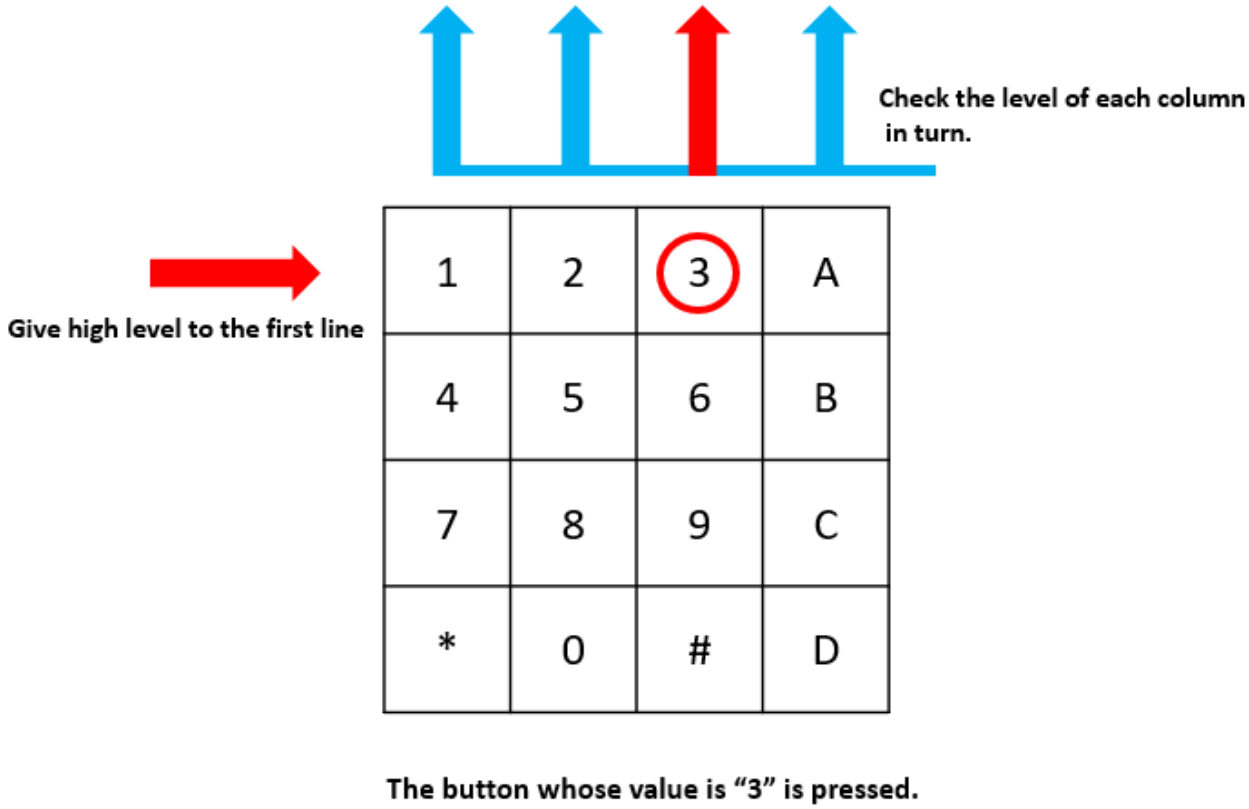
(次のページに続く)

(前のページからの続き)

```
                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。

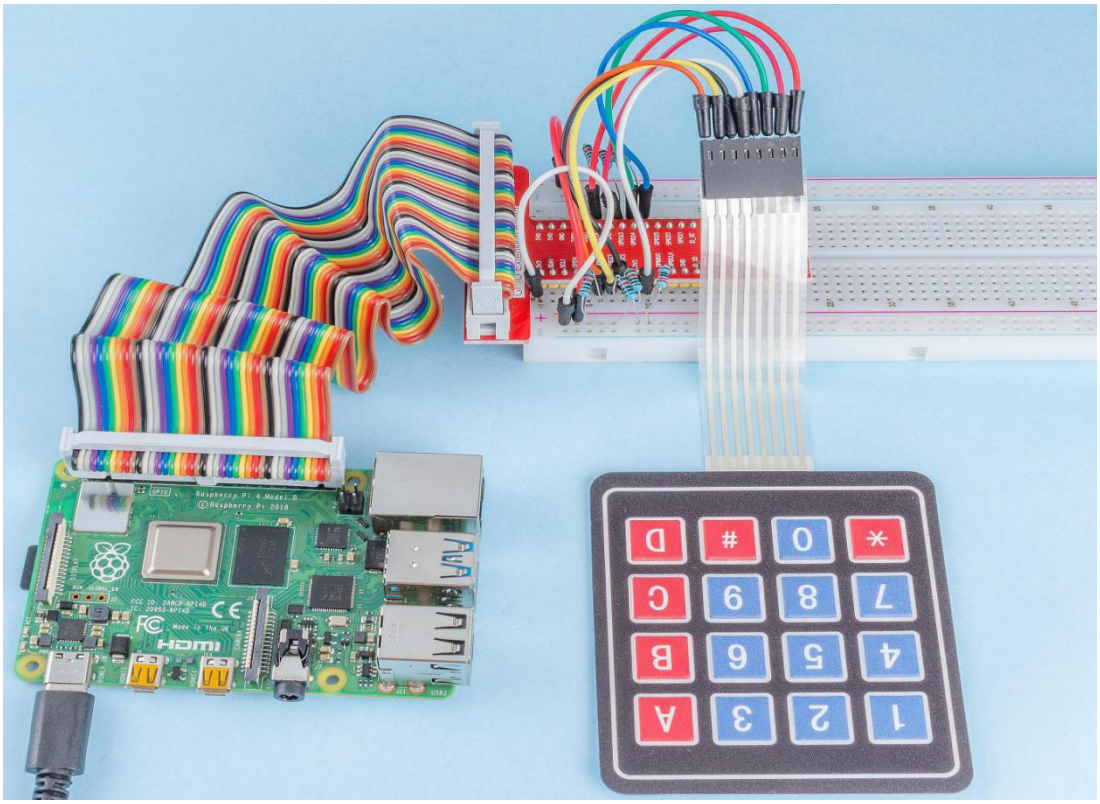
4 つの状態がある： 0、0、1、0 。そして、 pressed_keys に「3」を書き込む。

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

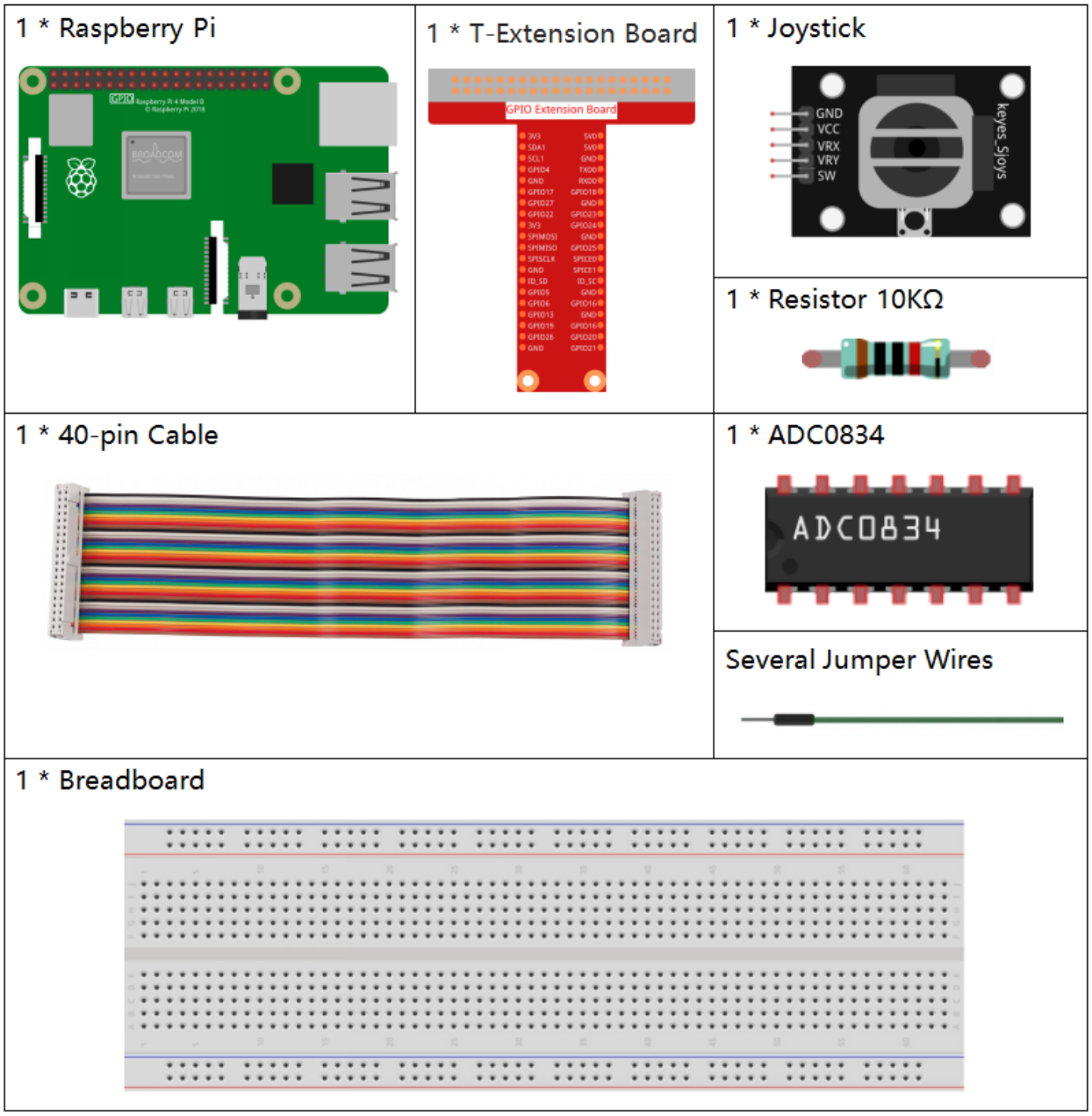


1.9.6 2.1.6 ジョイスティック

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



原理

ジョイスティック

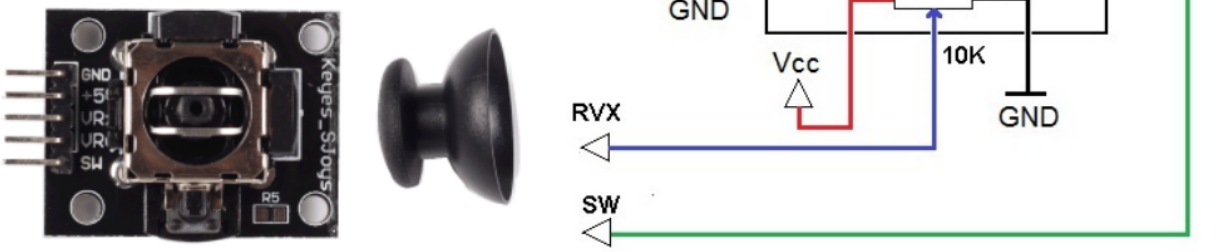
ジョイスティックの基本的な原理は、スティックの動きをコンピューターが処理できる電子情報に変換することである。

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。

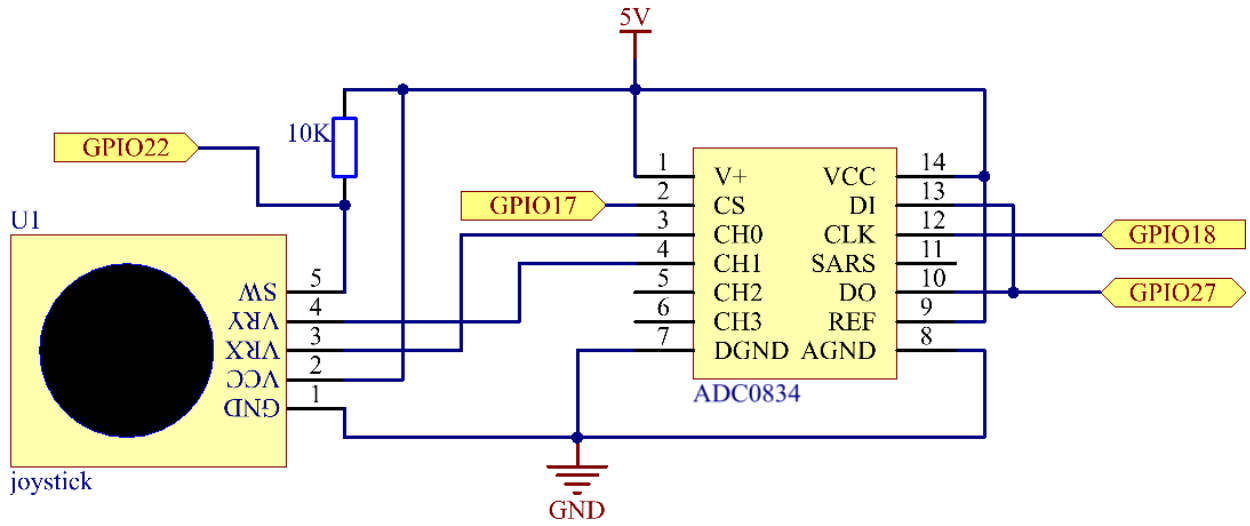
ジョイスティックには、ジョイスティックを押し下げたときに作動するデジタル入力もある。



回路図

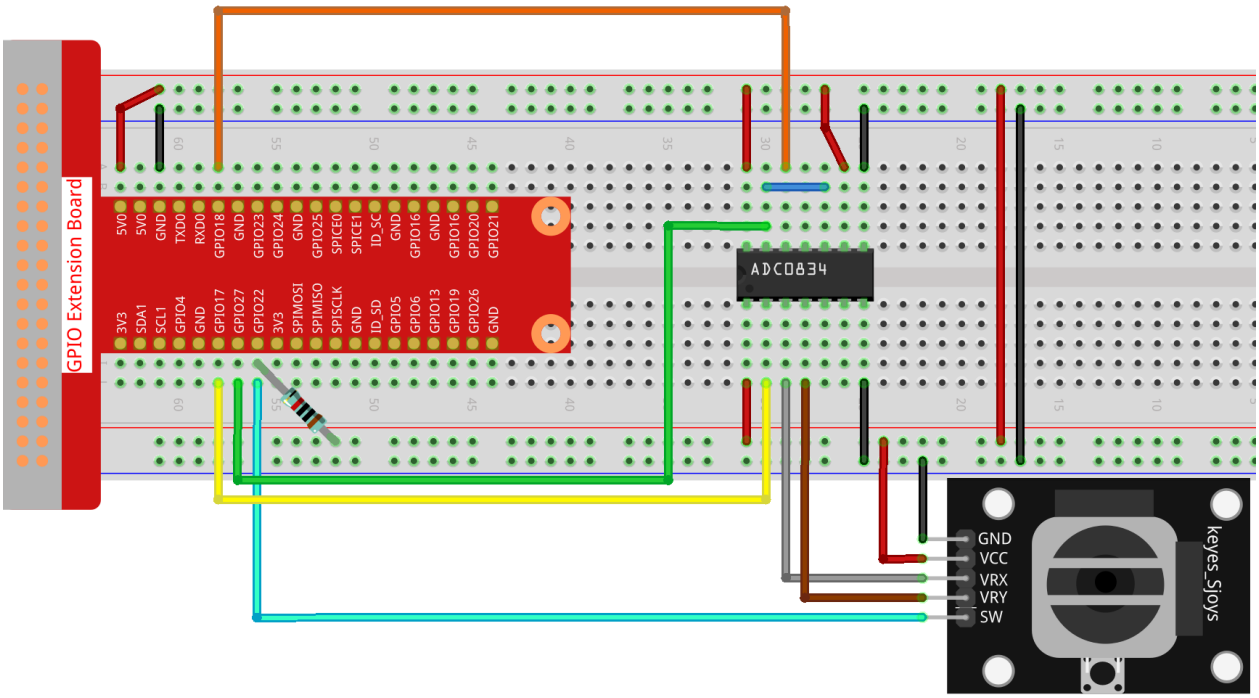
ジョイスティックのデータを読み取ると、軸間でいくつかの違いがある：X 軸と Y 軸のデータはアナログであり、ADC0834 を使用してアナログ値をデジタル値に変換する必要がある。Z 軸のデータはデジタルであるため、GPIO を直接使用して読み取るか、または ADC を使用して読み取ることができる。

T-Board Name	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);
```

(次のページに続く)

(前のページからの続き)

```
    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 messageto screen
```

(次のページに続く)

(前のページからの続き)

```
        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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import ADC0834
import time

BtnPin = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
```

(次のページに続く)

(前のページからの続き)

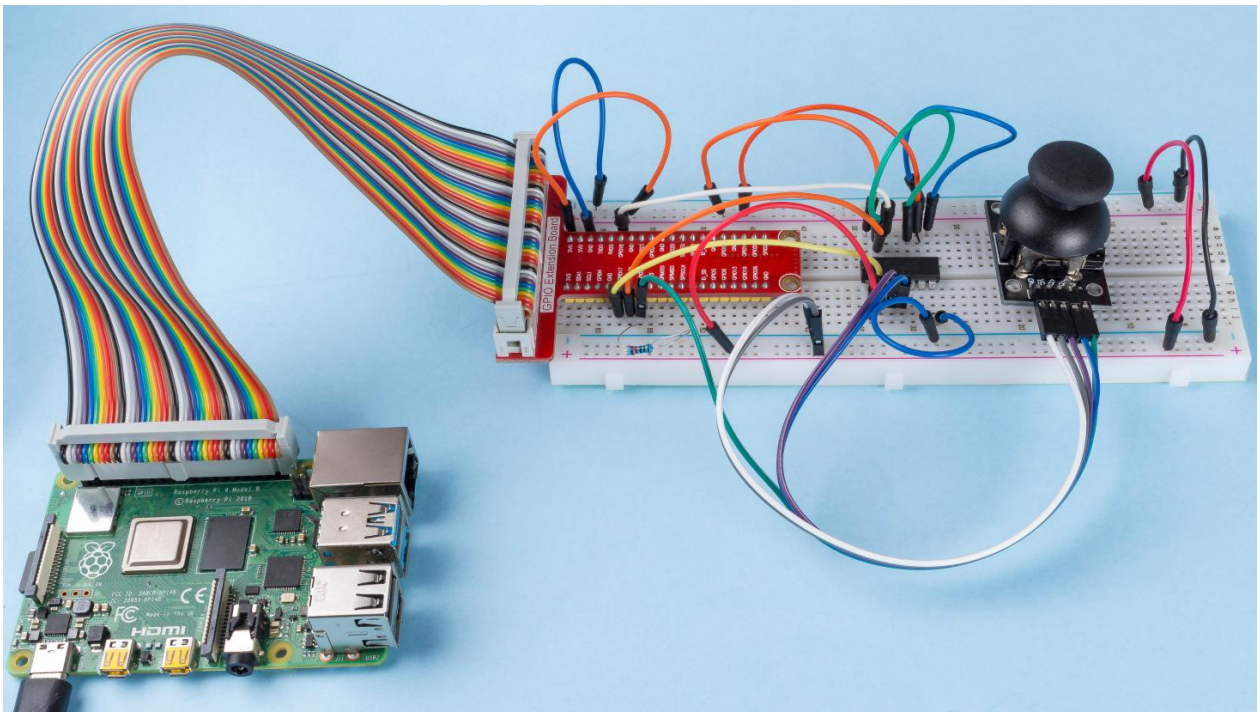
```
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC0834.setup()

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



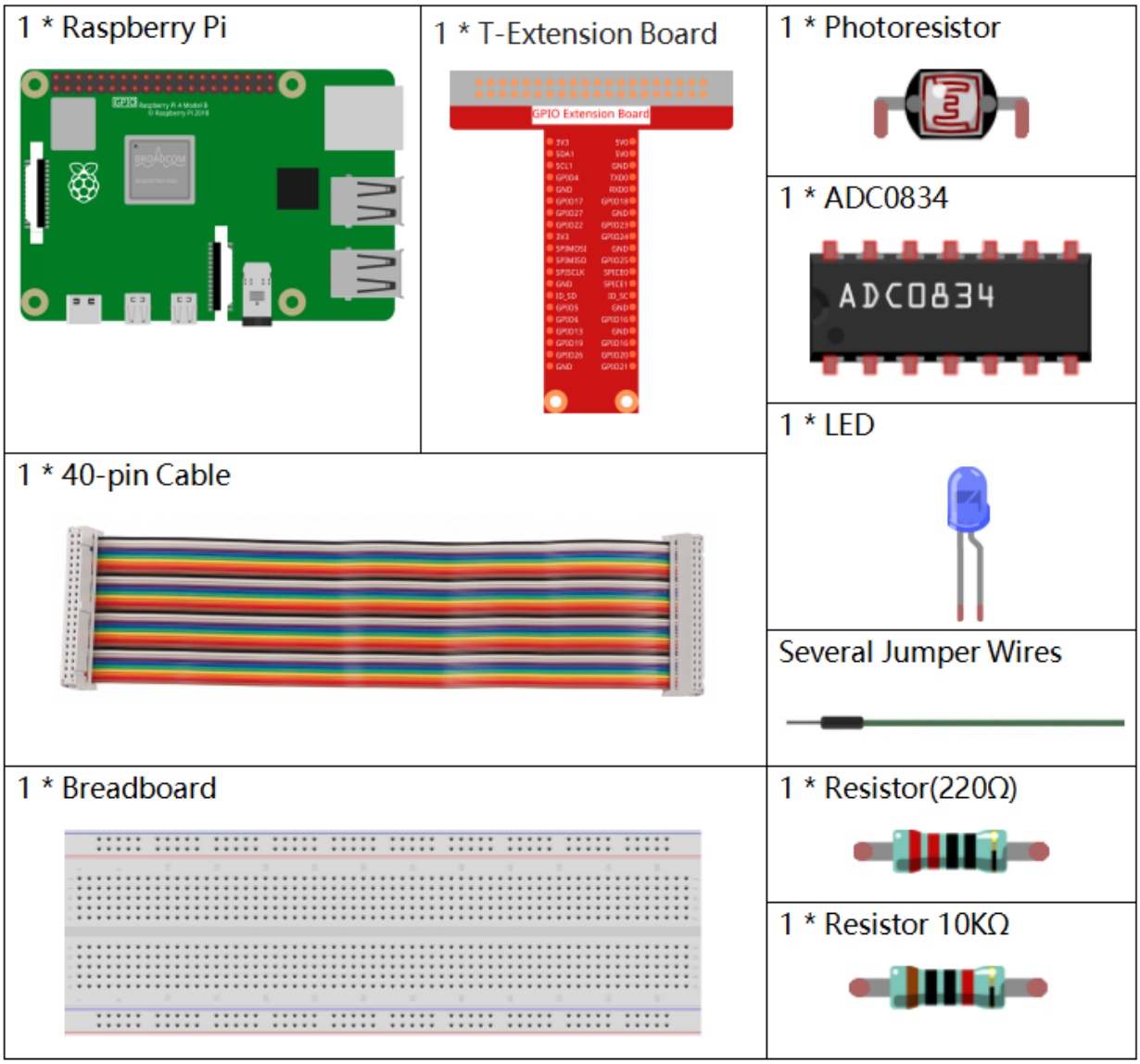
2.2 センサー

1.9.7 2.2.1 フォトレジスタ

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、光を感知するための電圧を変えると思うかもしれない。

部品

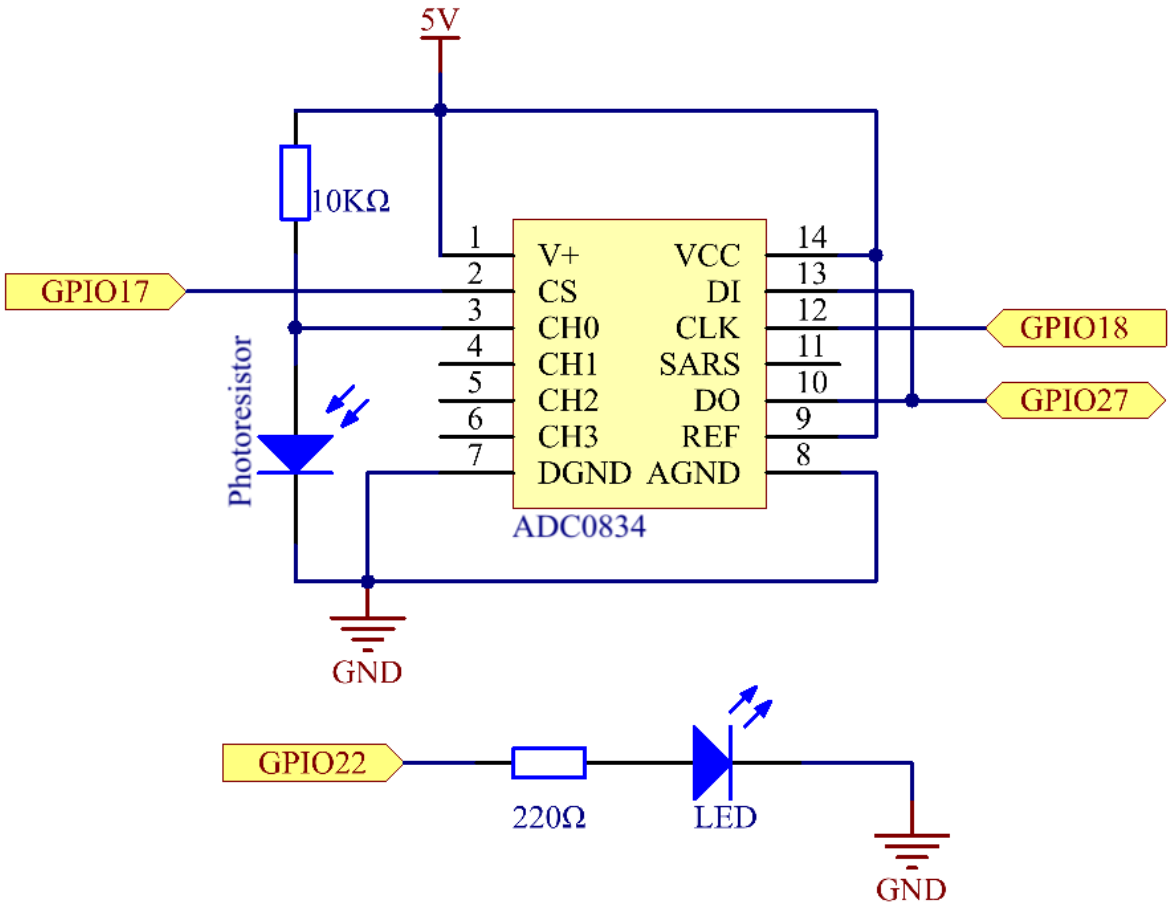


原理

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。

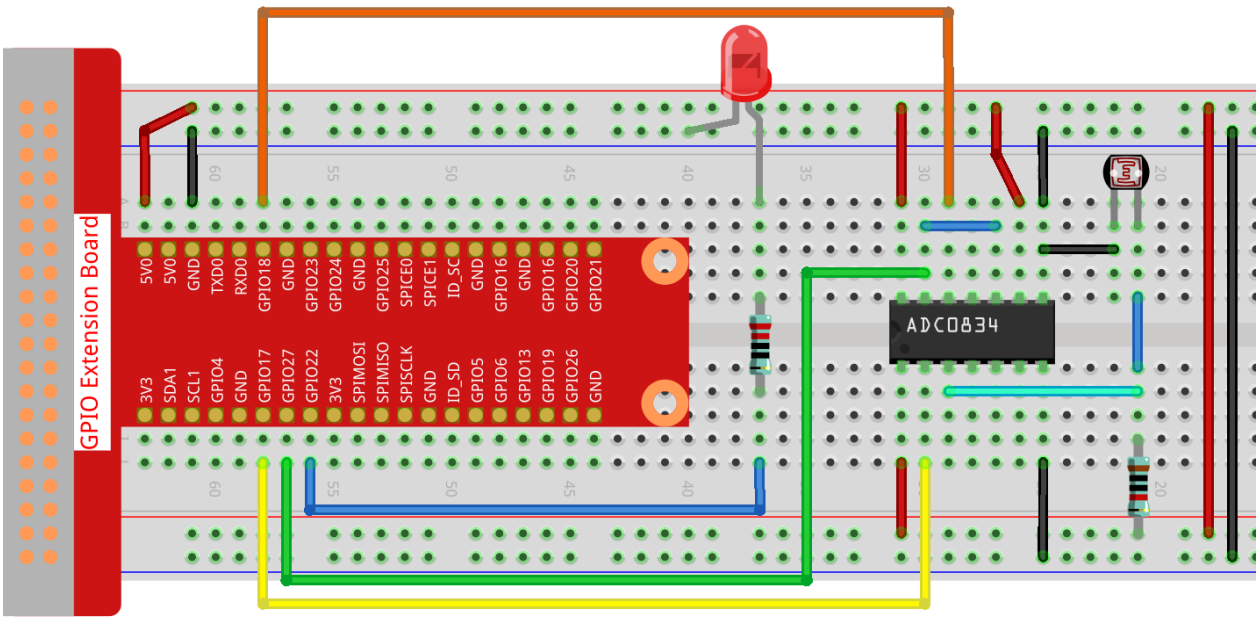
回路図

T-Board Name	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;
```

(次のページに続く)

(前のページからの続き)

```
#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);
```

(次のページに続く)

(前のページからの続き)

```
        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 messageto 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);
        delay(100);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

コードの説明

ここのコードは、2.1.4 ポテンショメーターのコードと同じである。他に質問がある場合は、2.1.4Potentiometer.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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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 frequece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)

    # Set all begin with value 0
    led_val.start(0)

def destroy():
```

(次のページに続く)

(前のページからの続き)

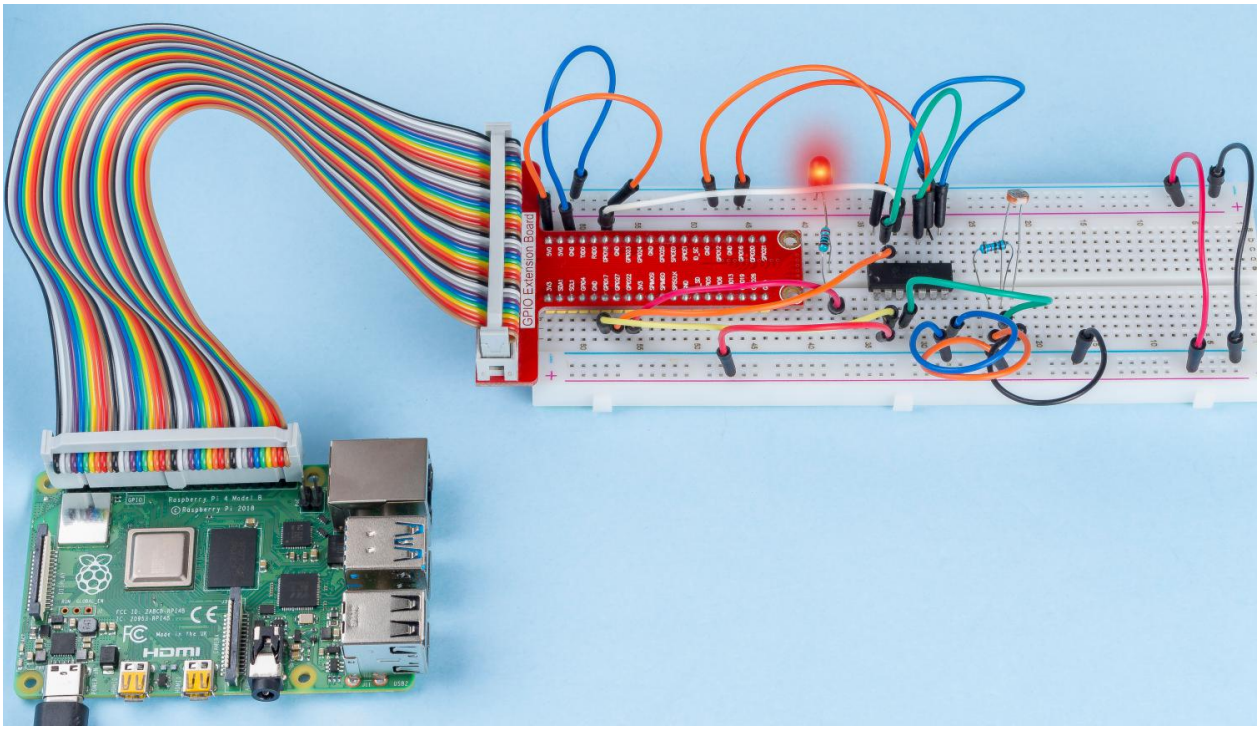
```
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()

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

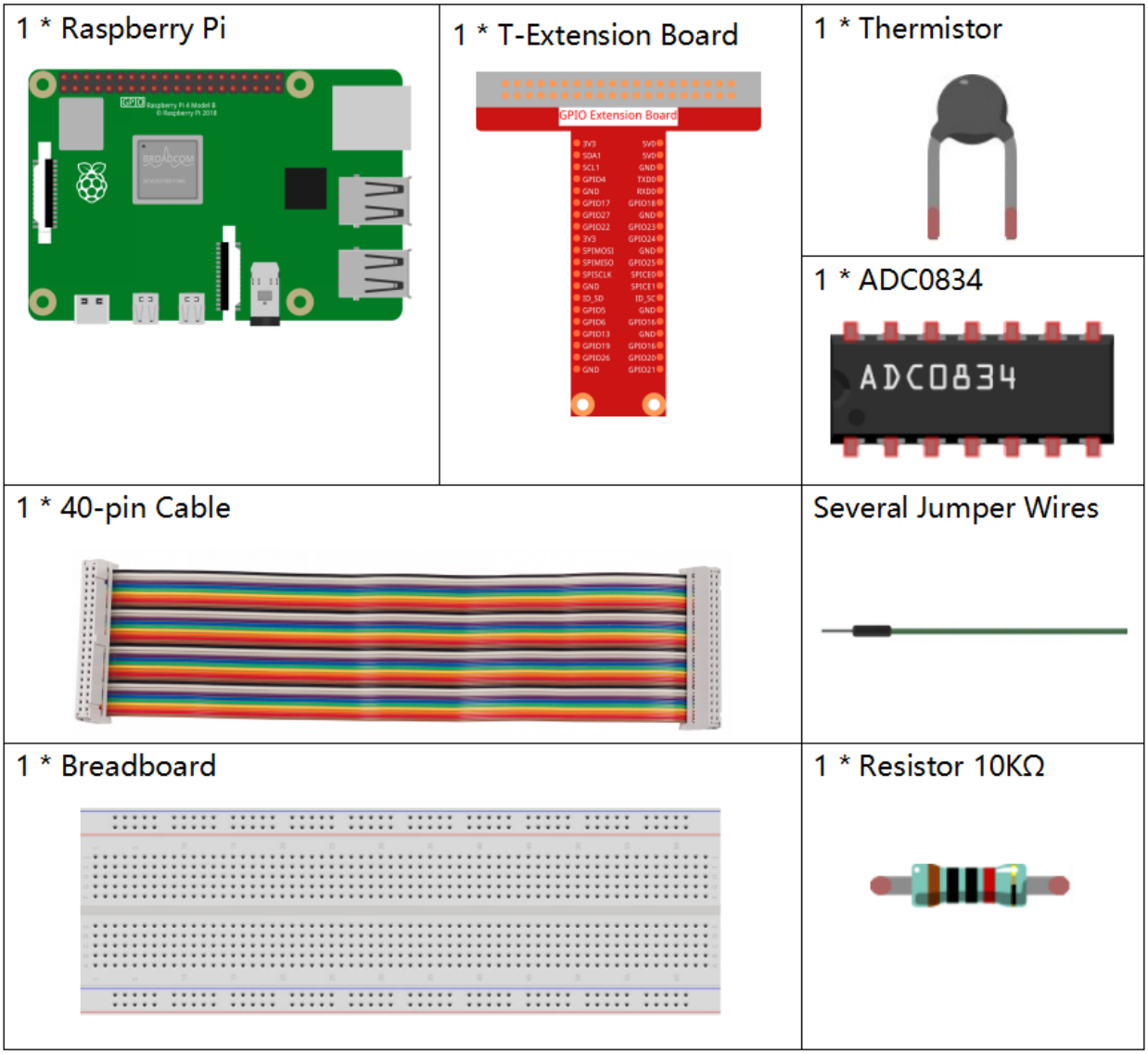


1.9.8 2.2.2 サーミスタ

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



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 ℃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 のときの NTC サーミスタの抵抗である。

RN は、定格温度 TN での NTC サーミスタの抵抗である。ここで、 RN の数値は 10k である。

TK はケルビン温度で、単位は K である。ここで、 TK の数値は 273.15 である。+ 摂氏度である。

TN は定格ケルビン温度で、単位も K である。ここで、 TN の数値は 273.15 + 25 である。

また、NTC サーミスタの構成要素定数である B（ベータ）は、数値 3950 の熱感受性指数とも呼ばれる。

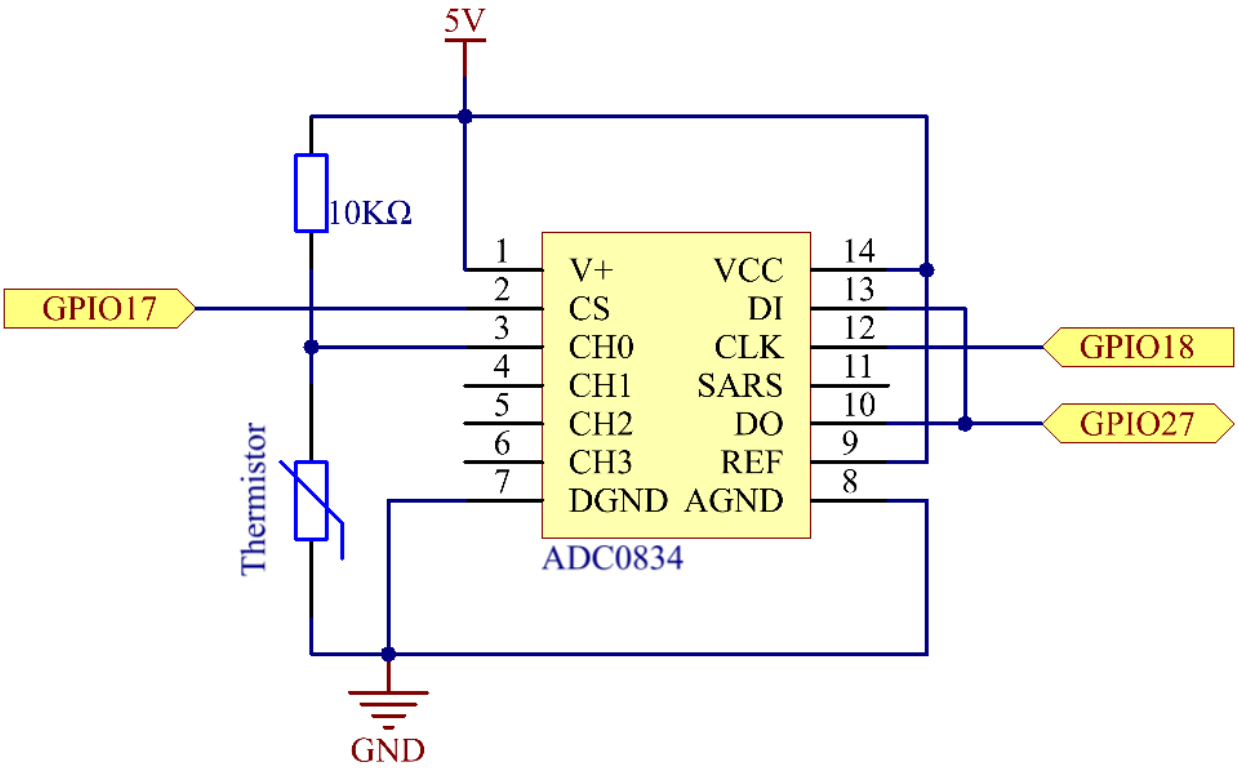
exp は指数の略語で、基数 e は自然数であり、約 2.7 に等しくなる。

この式 $T_K=1/(\ln(R_T/R_N)/B+1/T_N)$ を変換して、マイナス 273.15 が摂氏温度に相当するケルビン温度を取得する。

この関係は実験式である。温度と抵抗が有効範囲内にある場合にのみそれは正確である。

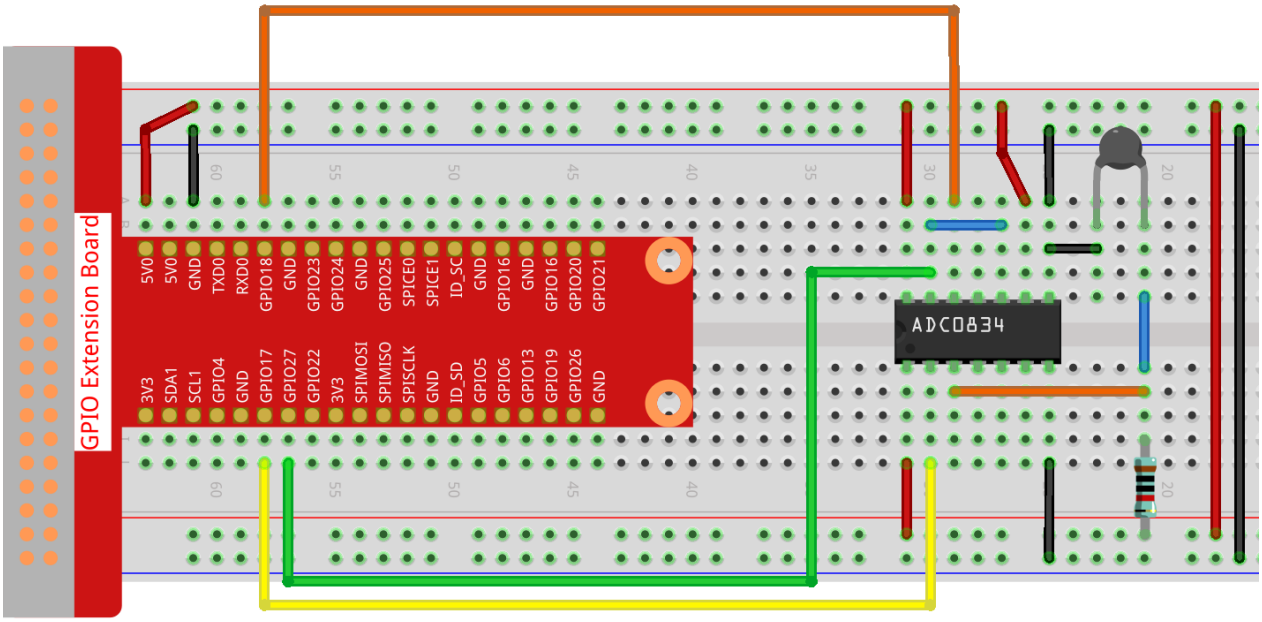
回路図

T-Board Name	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
```

注釈: -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>
```

(次のページに続く)

(前のページからの続き)

```
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++)
```

(次のページに続く)

(前のページからの続き)

```
    {
        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 messageto 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);
```

(次のページに続く)

(前のページからの続き)

```
        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_Thermistor.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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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))
```

(次のページに続く)

(前のページからの続き)

```
        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 ^^e2^^84^^89' % (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。

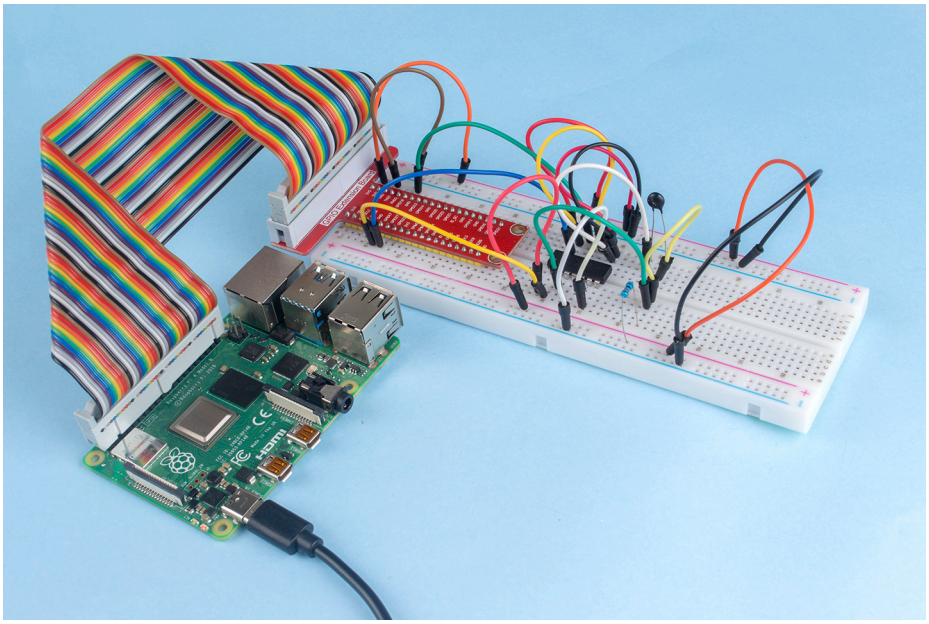
```
Fah = Cel * 1.8 + 32
```

摂氏を華氏に変換する。

```
print ('Celsius: %.2f ° C  Fahrenheit: %.2f ^^e2^^84^^89' % (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



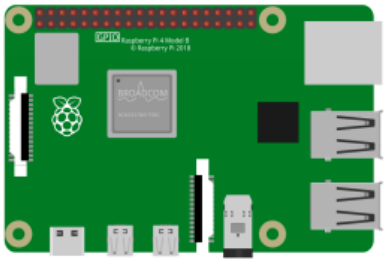
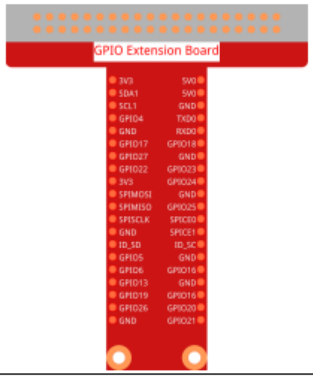
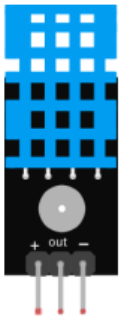
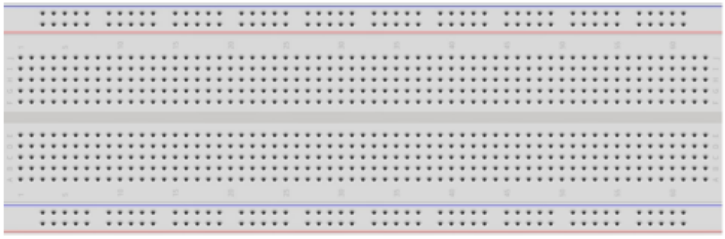
1.9.9 2.2.3 DHT-11

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。

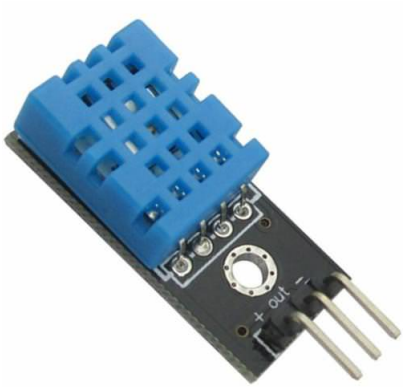
センサーには、湿式素子抵抗センサーと NTC 温度センサーが含まれており、高性能の 8 ビットマイクロコントローラーに接続されている。

部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * DHT-11 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor 10KΩ

原理

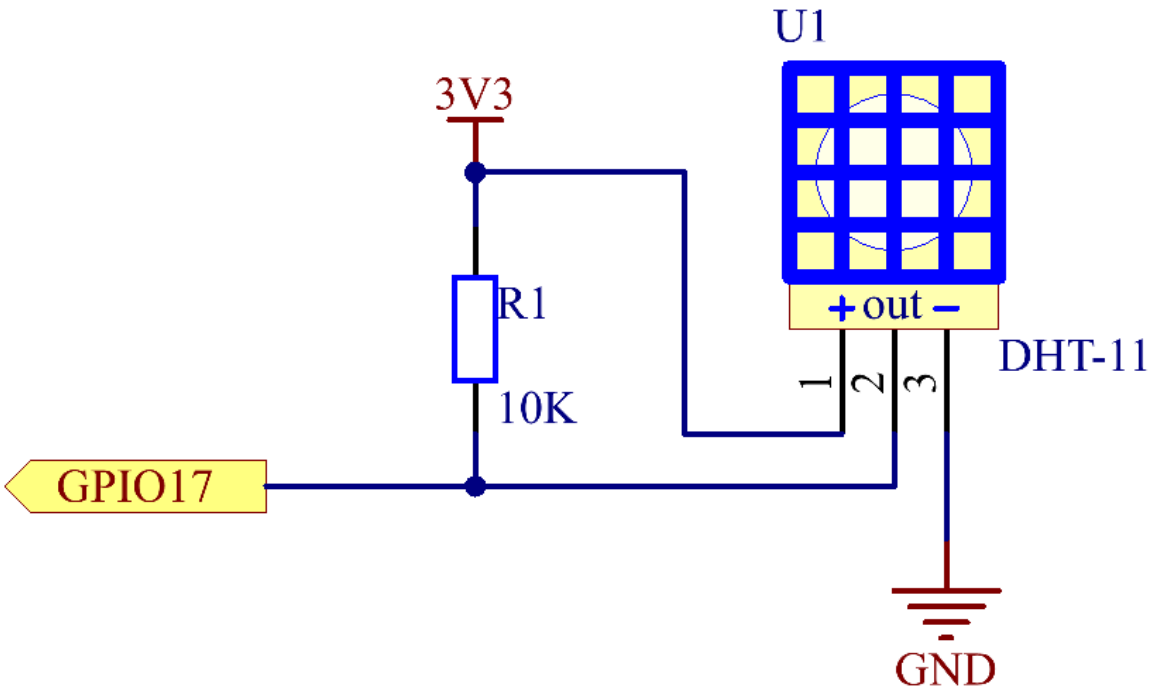
DHT11 は基本的な超低コストのデジタル温湿度センサーである。容量性湿度センサーとサーミスタを使用して周囲の空気を測定し、データピンにデジタル信号を出力する（アナログ入力ピンは不要）。



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 ビット温度整数 + 8 ビット温度 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。

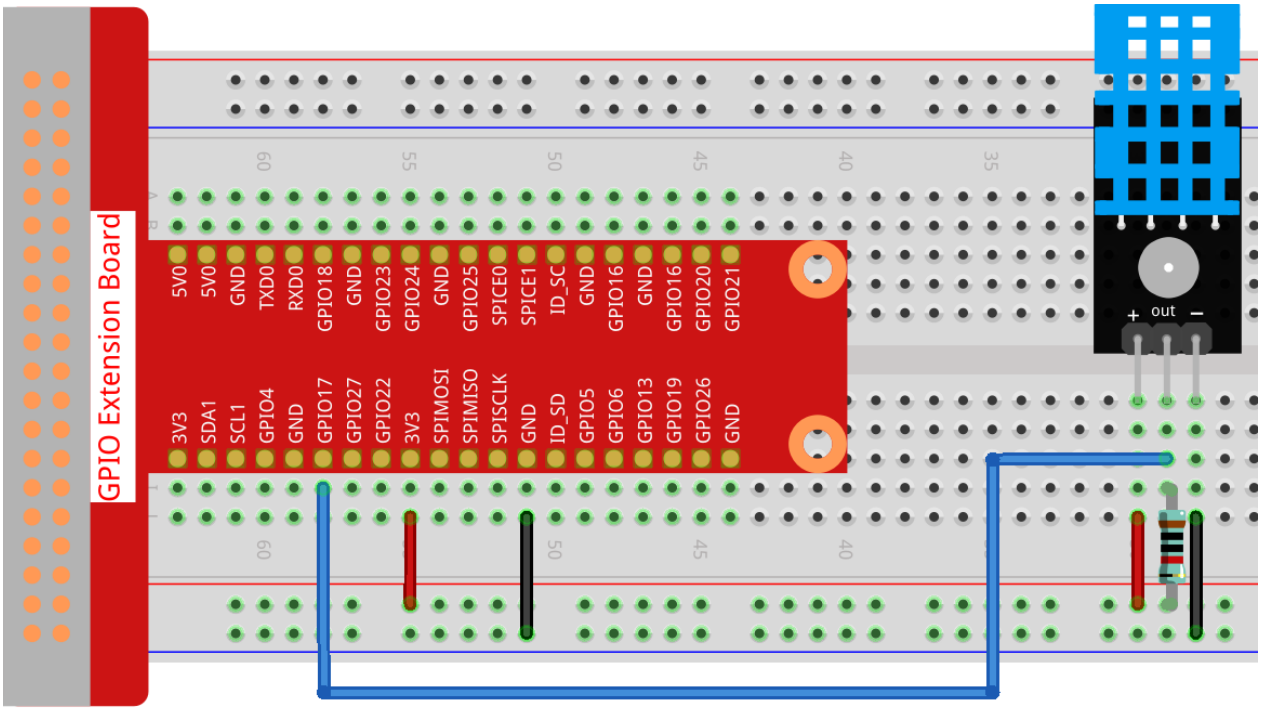
回路図

T-Board Name	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>
```

(次のページに続く)

(前のページからの続き)

```
#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)
```

(次のページに続く)

(前のページからの続き)

```
                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 messageto 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;
    // ...
}
```

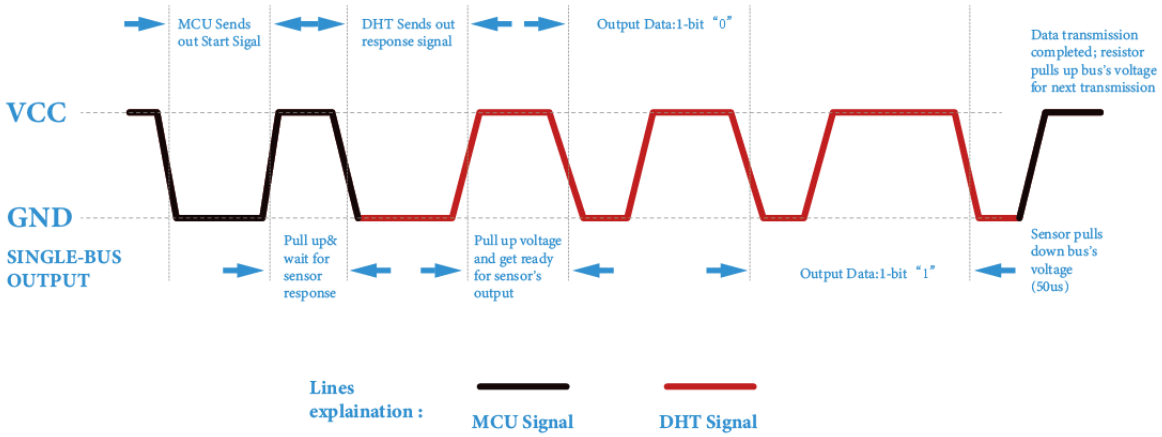
この機能は DHT11 の機能を実現するために使用される。

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;
            }
```

(次のページに続く)

(前のページからの続き)

```
        }
        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：

8 ビット湿度整数データ + 8 ビット湿度 10 進データ + 8 ビット温度整数データ + 8 ビット温度 10 進データ + 8 ビットチェックビット。

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、 チェックビット (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）の場合、

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 ℃。

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
    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
```

(次のページに続く)

(前のページからの続き)

```
            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)
```

(次のページに続く)

(前のページからの続き)

```
            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 = []
```

(次のページに続く)

(前のページからの続き)

```
    #...
```

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：

8 ビット湿度整数データ +8 ビット湿度 10 進データ +8 ビット温度整数データ + 8 ビット温度 10 進データ + 8 ビットチェックビット。

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）の場合

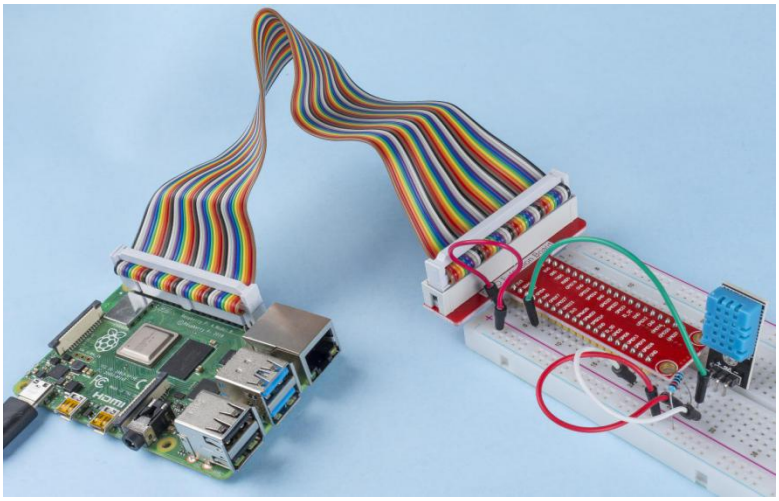
計算:

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」が出力される。

現象画像

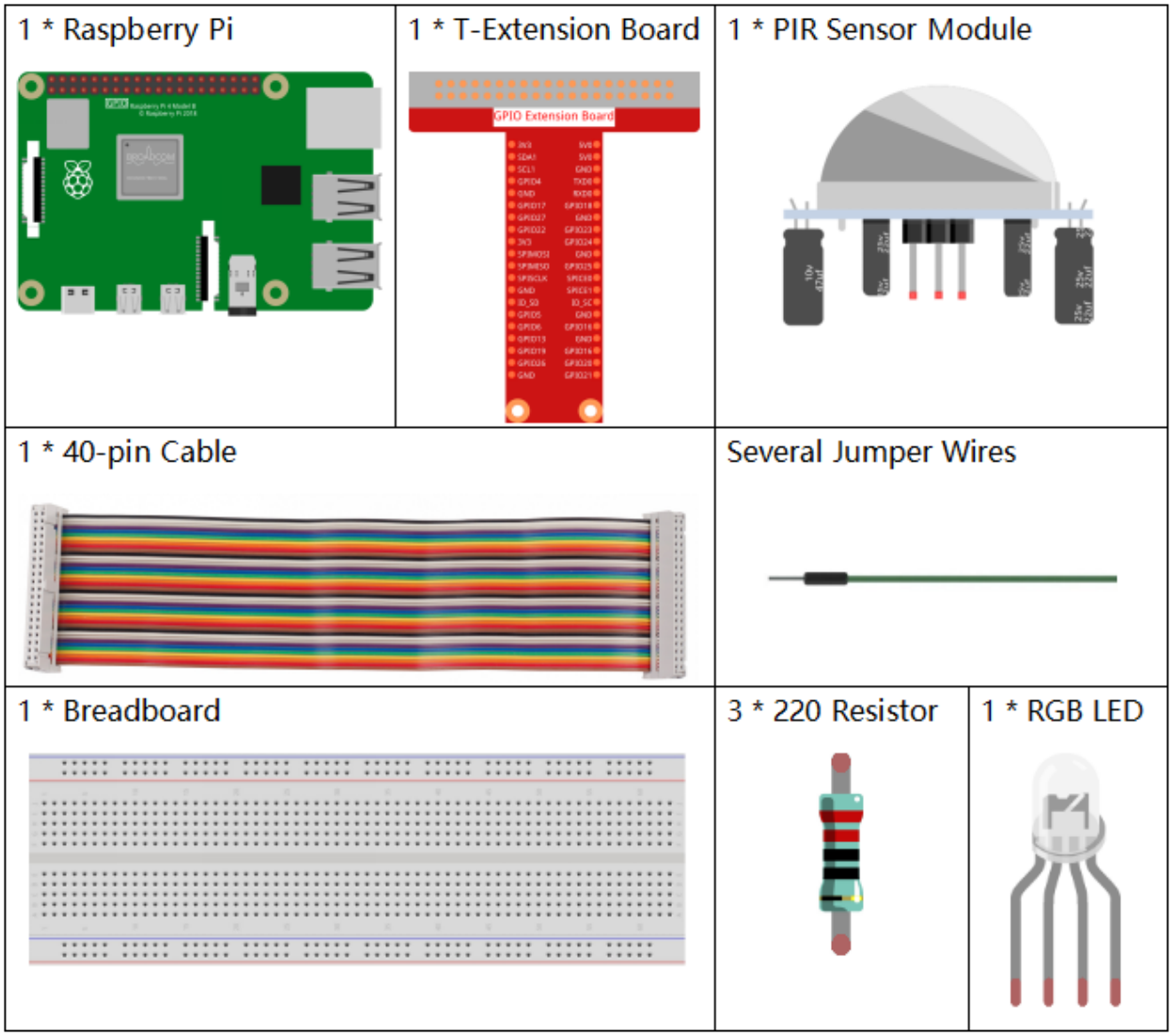


1.9.10 2.2.4 PIR

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

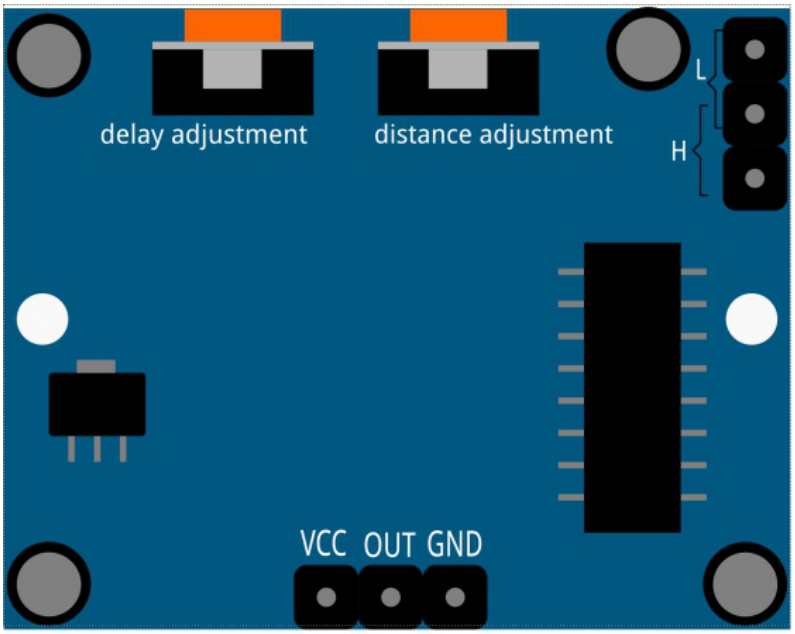


原理

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。

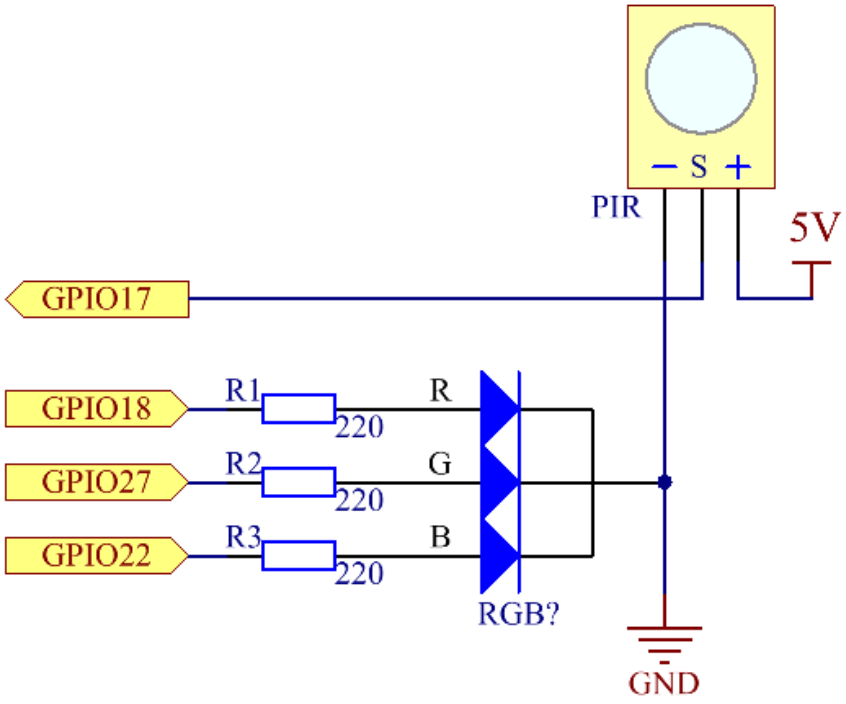
二つのトリガーモード：(ジャンパーキャップを使用して異なるモードを選択する)。

- **H： 繰り返し可能なトリガーモード**、人体を感知した後、モジュールは高レベルを出力します。 その後の遅延期間中に、誰かが検知範囲に入ると、出力は高レベルのままになります。

- **L： 繰り返し不可のトリガーモード**、人体を感知すると高レベルを出力します。 遅延後、出力は自動的に高レベルから低レベルに変わります。

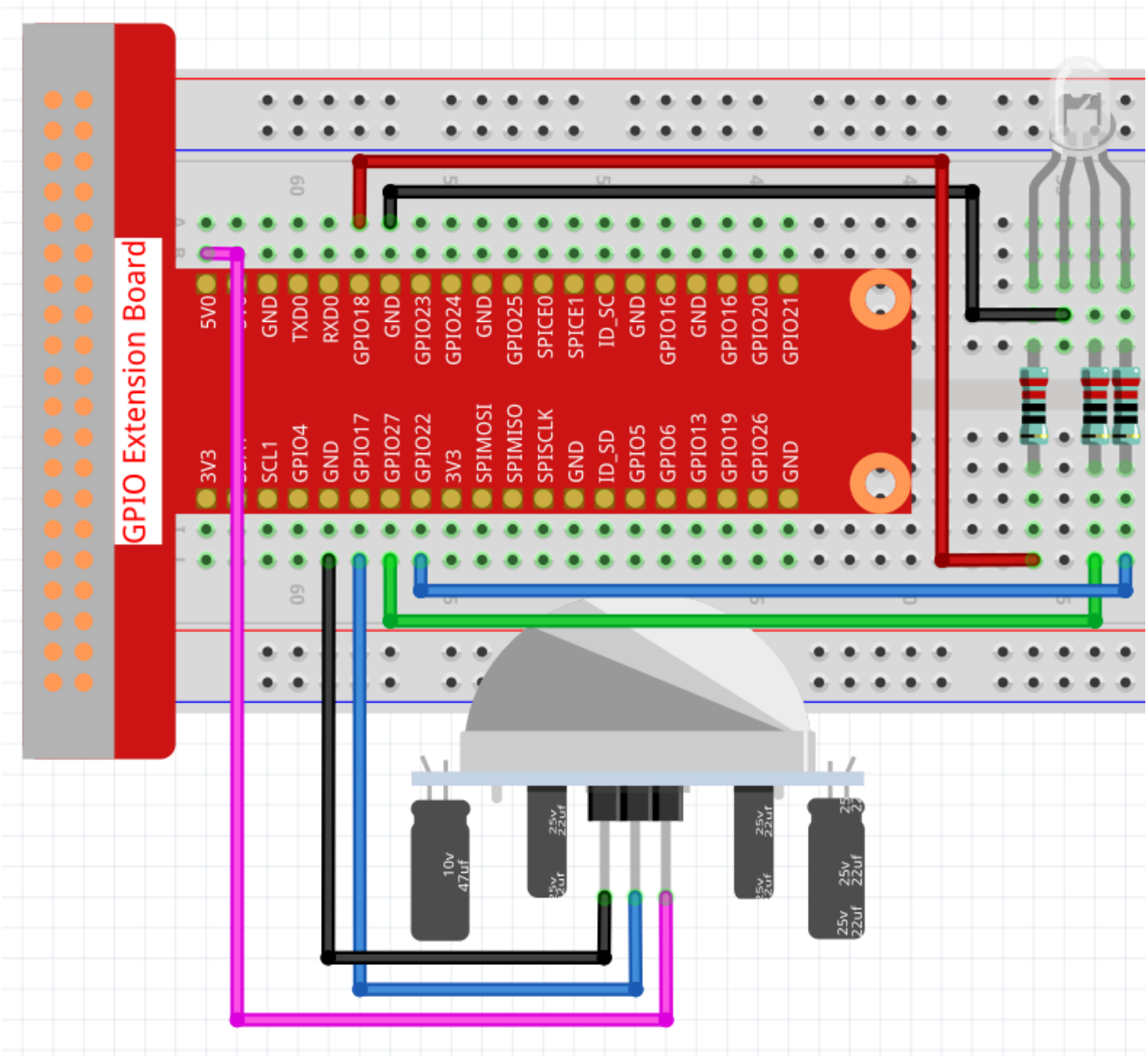
回路図

T-Board Name	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 は周囲を検出し、誰かが通り過ぎるのを感知すると、RGB LED を黄色に点灯させます。 PIR モジュールには 2 つのポテンショメータがあり、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す。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。

コード

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define uchar unsigned char
```

(次のページに続く)

(前のページからの続き)

```
#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);
```

(次のページに続く)

(前のページからの続き)

```
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ます。詳細については、 *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 は周囲を検出し、誰かが通り過ぎるのを感知すると、RGB LED を黄色に点灯させます。 PIR モジュールには 2 つのポテンショメータがあり、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す。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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
```

(次のページに続く)

(前のページからの続き)

```
    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():
```

(次のページに続く)

(前のページからの続き)

```
    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):
```

(次のページに続く)

(前のページからの続き)

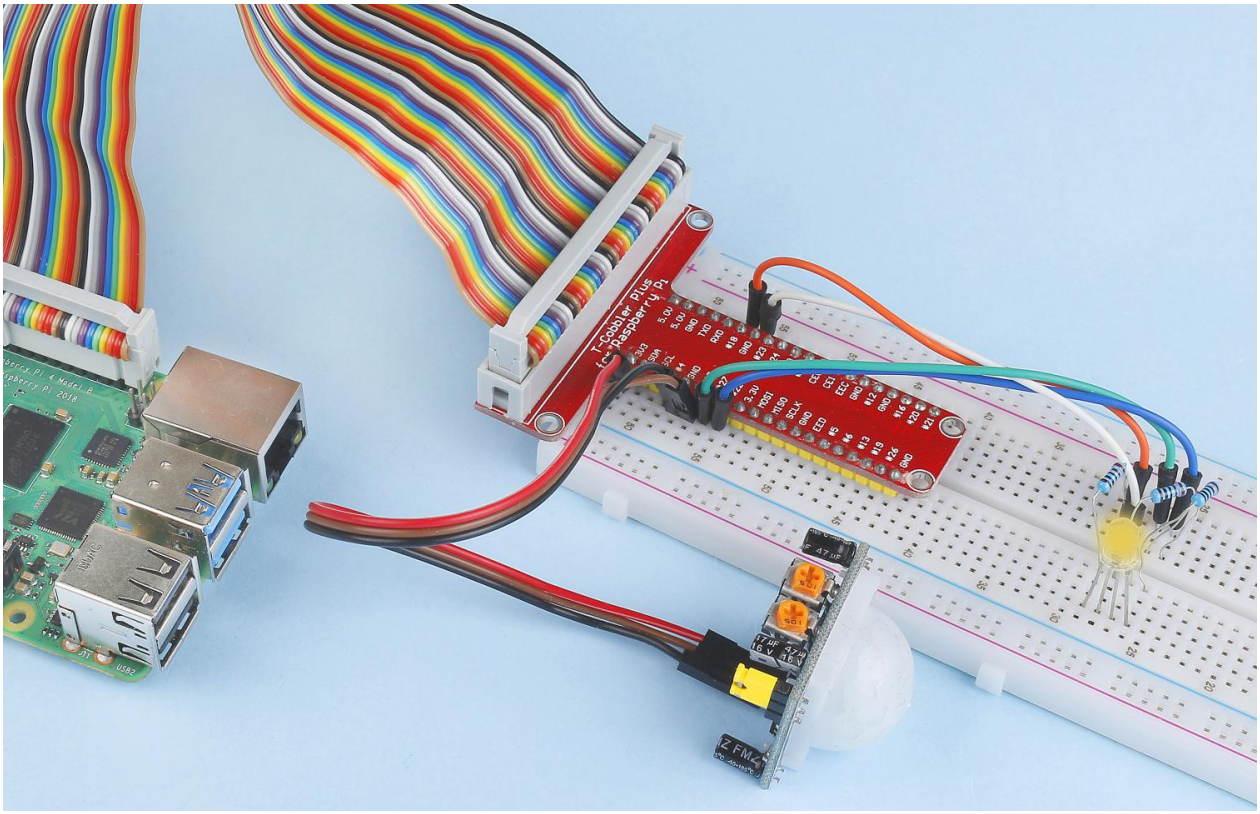
```
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

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ます。詳細については、 *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

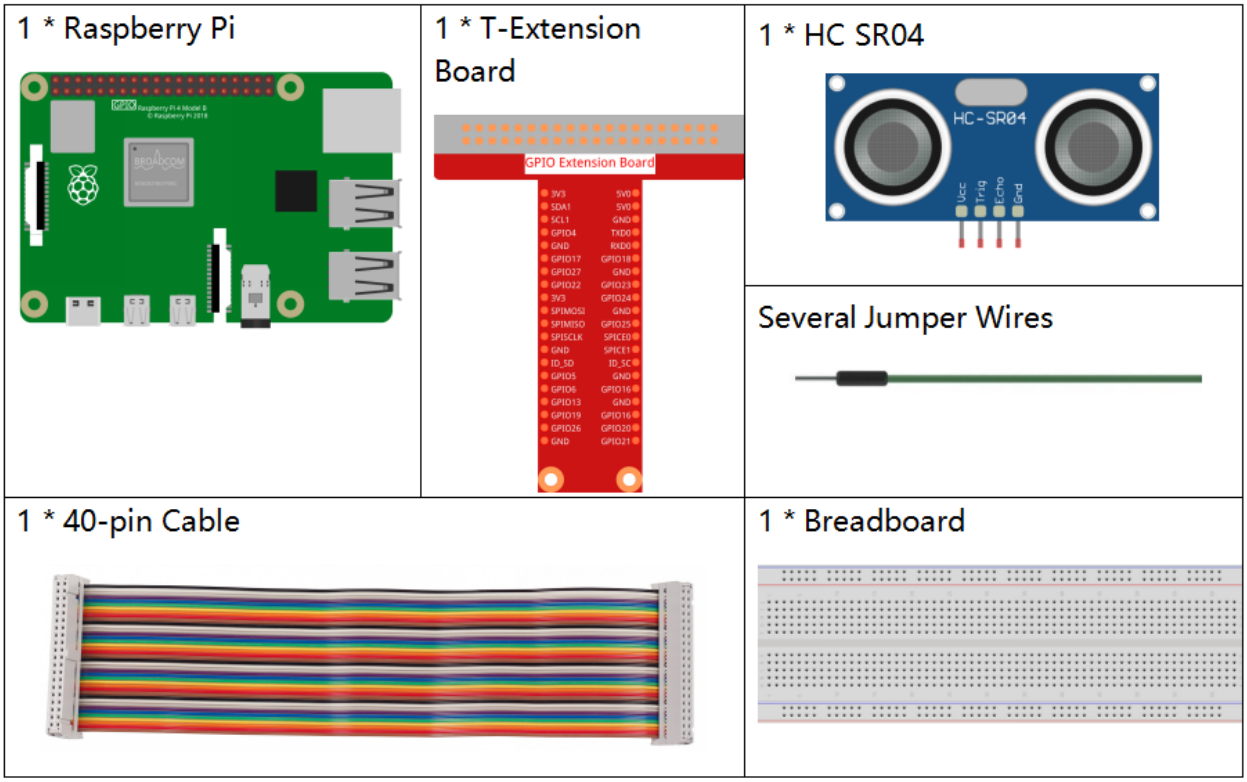


1.9.11 2.2.5 超音波センサーモジュール

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。

部品



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。

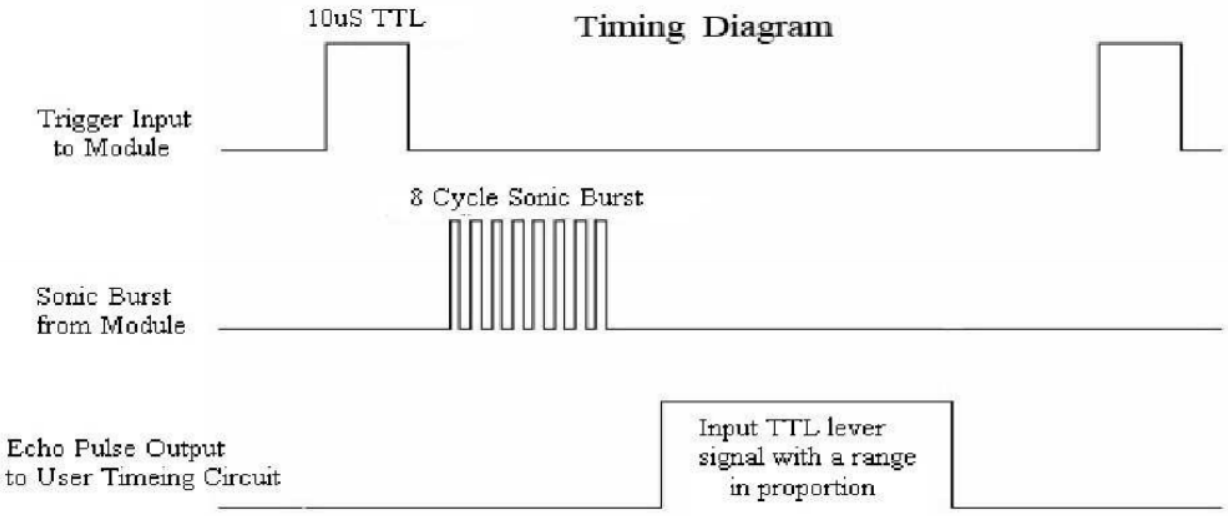
(2) モジュールは 8 つの 40khz を自動的に送信し、パルス信号が戻すかどうかを検出する。

(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=（高時間 x 音速（340 m/s）/ 2。

TRIG	Trigger Pulse Input
ECHO	Echo Pulse Output
GND	Ground
VCC	Supply

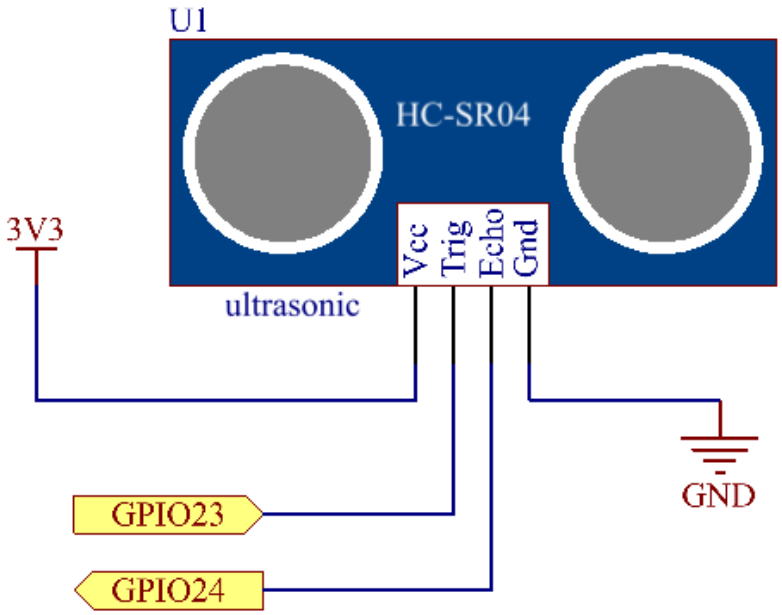
タイミング図を以下に示す。トリガー入力に 10us の短いパルスを供給してレンジングを開始するだけで、モジュールは 40 kHz で 8 サイクルの超音波バーストを送信し、エコーを上げる。トリガー信号を送信してからエコー信号を受信するまでの時間間隔で範囲を計算できる。

式：us/58 =センチメートルまたは us/148 =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。



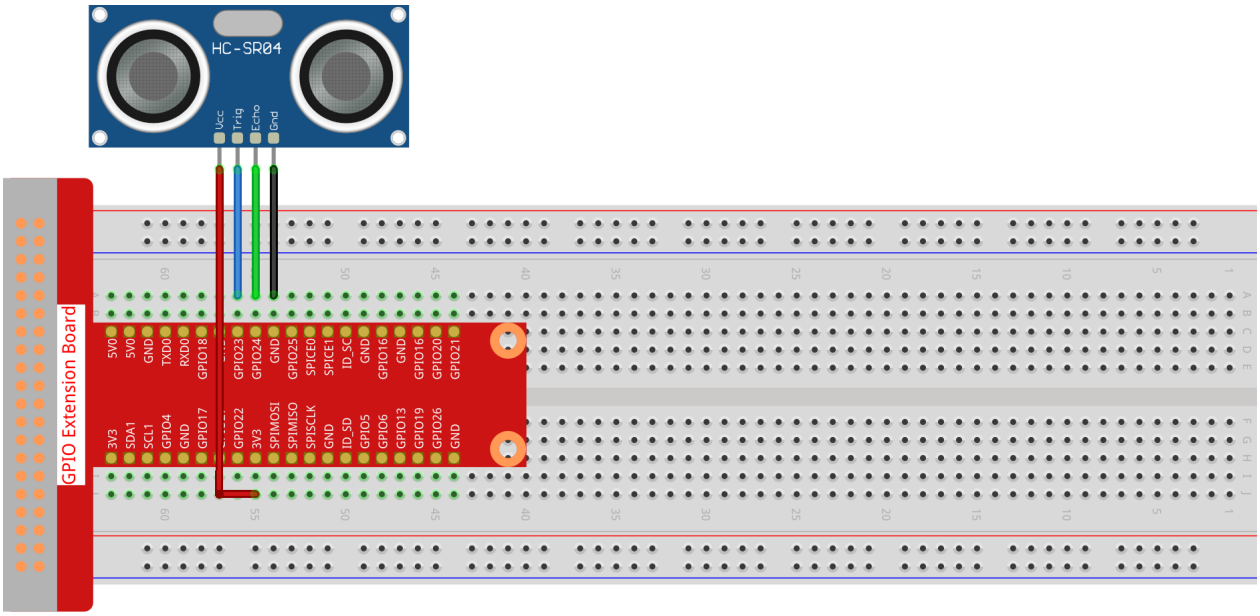
回路図

T-Board Name	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

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define Trig    4
#define Echo    5
```

(次のページに続く)

(前のページからの続き)

```
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
```

(次のページに続く)

(前のページからの続き)

```
        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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
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。

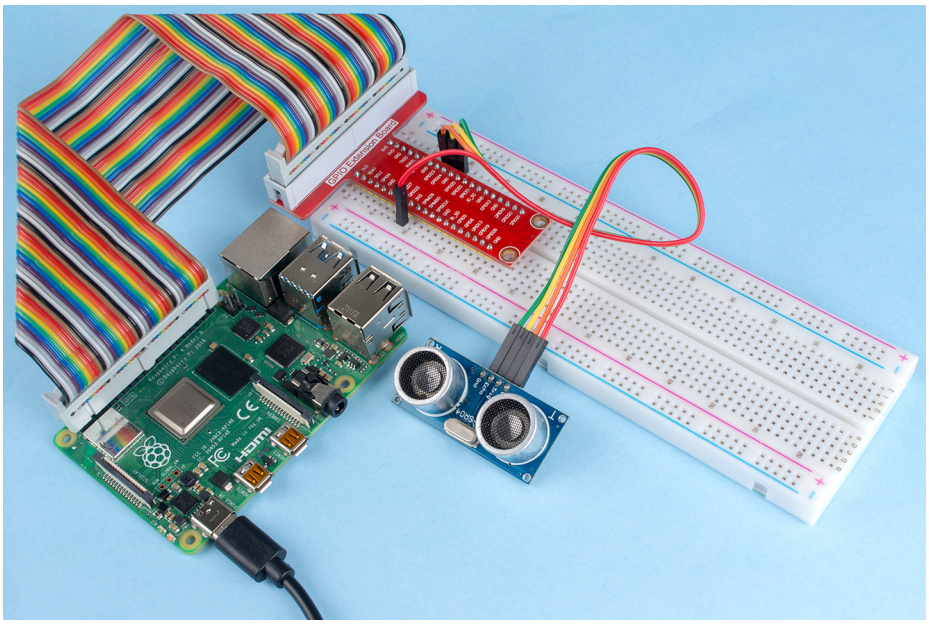
```
during = time2 - time1
```

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



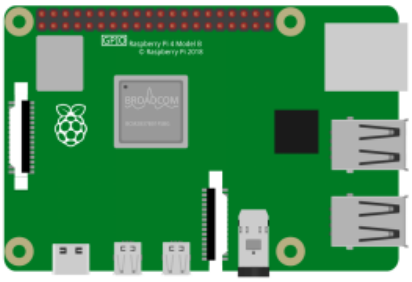
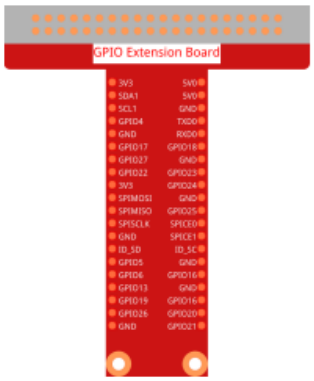
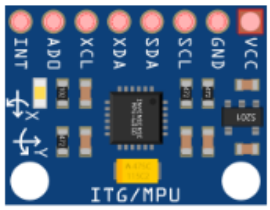
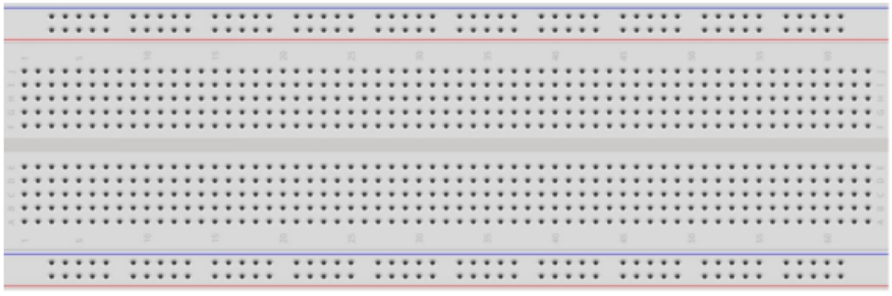
1.9.12 2.2.6 MPU6050 モジュール

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

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * MPU6050
 
Several Jumper Wires
</td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

原理

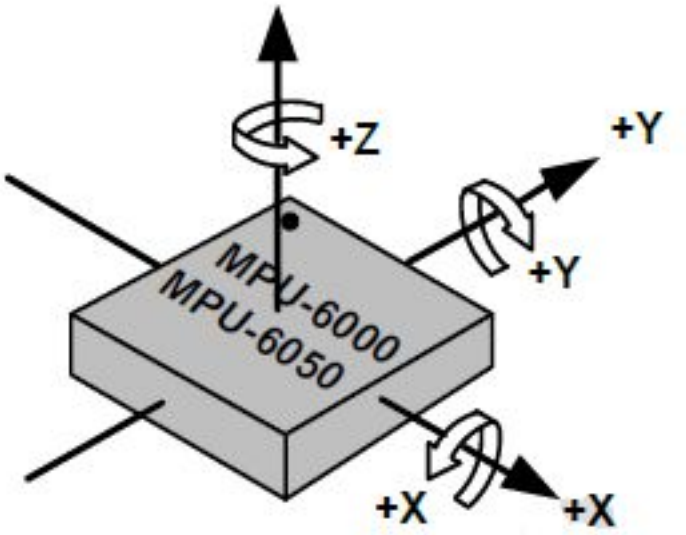
MPU6050

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

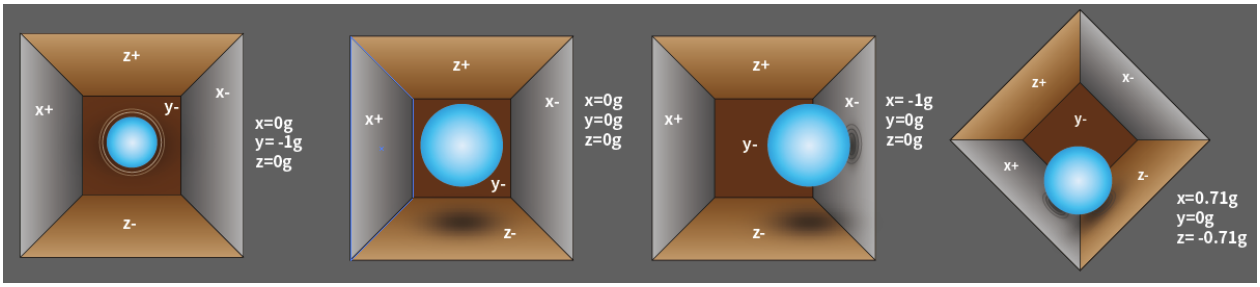
認してください。次に、上向きの直立方向がチップの z 軸である。左から右への方向は X 軸と見なされる。したがって、後ろから前への方向は Y 軸として定義される。



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-32768～32767 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。

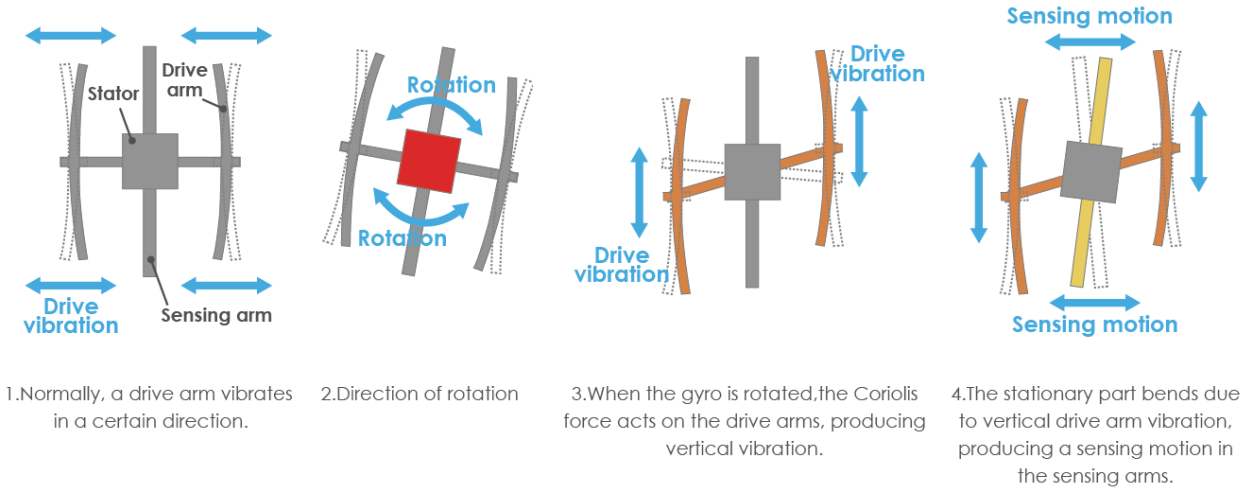
加速度=（加速度計軸の生データ / 65536 * フルスケールの加速度範囲）g

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 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：

角速度=（ジャイロスコープの軸生データ/ 65536 *フルスケールジャイロスコープの範囲）° / s

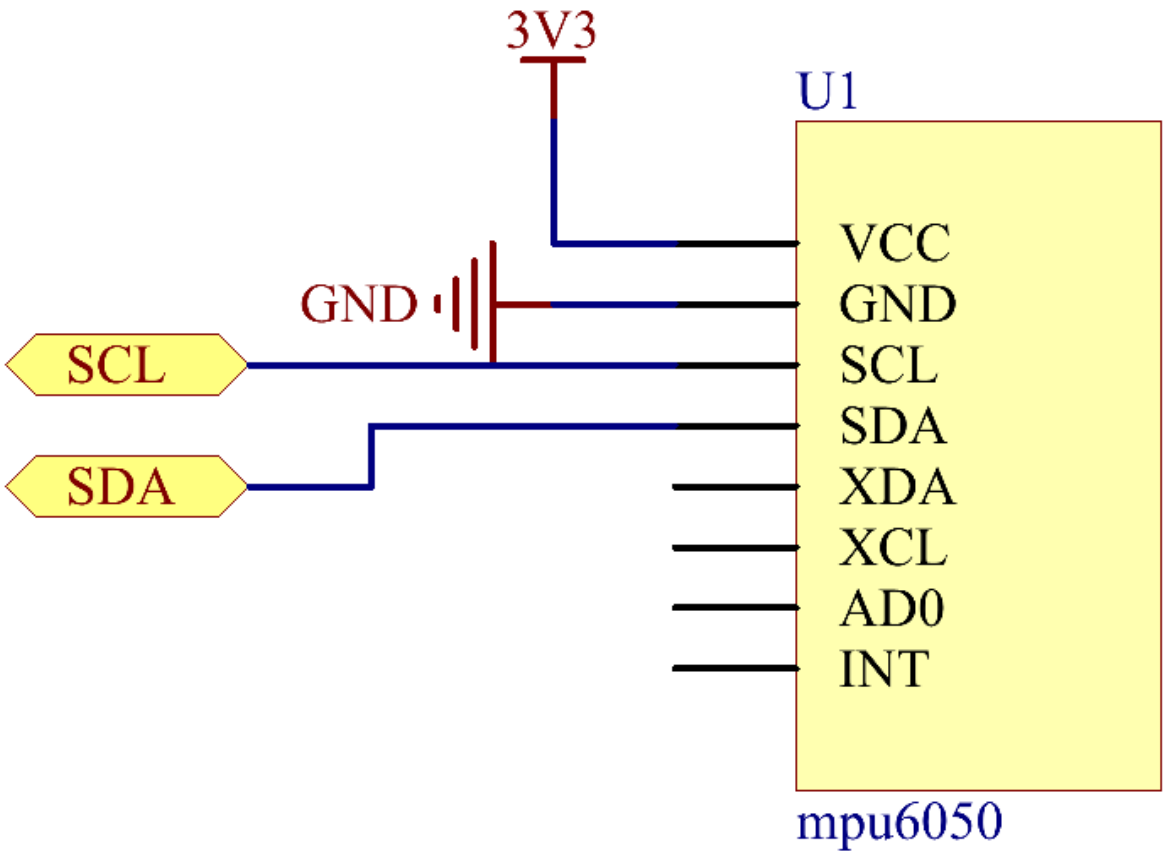
X 軸、たとえば、加速度計の X 軸の生データは 16384 で、範囲は + /-250 ° / s である：

X 軸による角速度 = (16384 / 65536 * 500) ° /s =125 ° /s

回路図

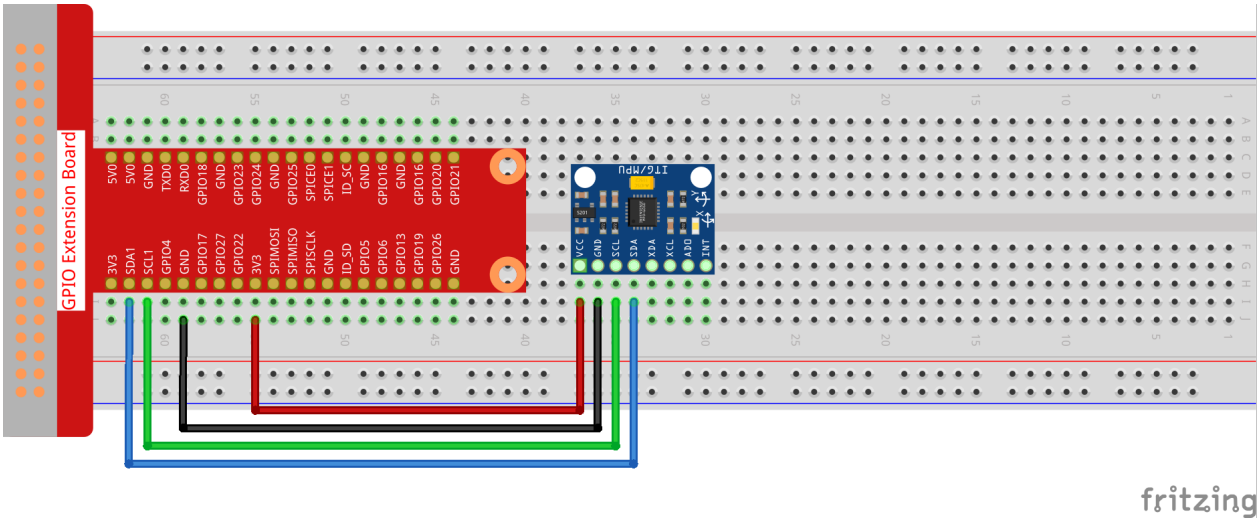
MPU6050 は I2C バスインターフェイスを介してマイクロコントローラーと通信する。SDA1 と SCL1 を対応するピンに接続する必要がある。

T-Board Name	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* **構成** （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  <wiringPiI2C.h>
#include <wiringPi.h>
#include  <stdio.h>
#include  <math.h>
int fd;
int acclX, acclY, acclZ;
int gyroX, gyroY, gyroZ;
```

(次のページに続く)

(前のページからの続き)

```
double acclX_scaled, acclY_scaled, acclZ_scaled;
double gyroX_scaled, gyroY_scaled, gyroZ_scaled;

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;
}

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 = wiringPiI2CSetup (0x68);
wiringPiI2CWriteReg8 (fd,0x6B,0x00);//disable sleep mode
printf("set 0x6B=%X\n",wiringPiI2CReadReg8 (fd,0x6B));
```

(次のページに続く)

(前のページからの続き)

```
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", gyroY X_scaled);
    delay(100);
    printf("My gyroY_scaled: %f\n", gyroY Y_scaled);
    delay(100);
    printf("My gyroZ_scaled: %f\n", gyroY Z_scaled);
    delay(100);

    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);
    delay(100);
    printf("My acclY_scaled: %f\n", acclY_scaled);
    delay(100);
    printf("My acclZ_scaled: %f\n", acclZ_scaled);
    delay(100);

    printf("My X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_
↪scaled));
    delay(100);
    printf("My Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_
↪scaled));
```

(次のページに続く)

(前のページからの続き)

```
    delay(100);

    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", gyroY 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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:
```

(次のページに続く)

(前のページからの続き)

```
        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)
```

(次のページに続く)

(前のページからの続き)

```
    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。

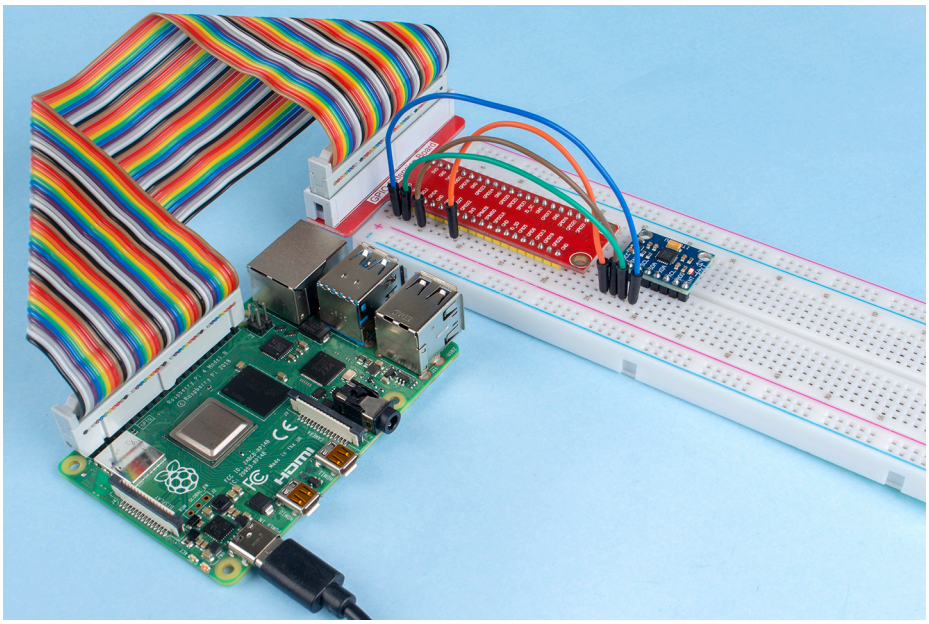
```
accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
accel_zout = read_word_2c(0x3f)

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



1.9.13 2.2.7 MFRC522 RFID モジュール

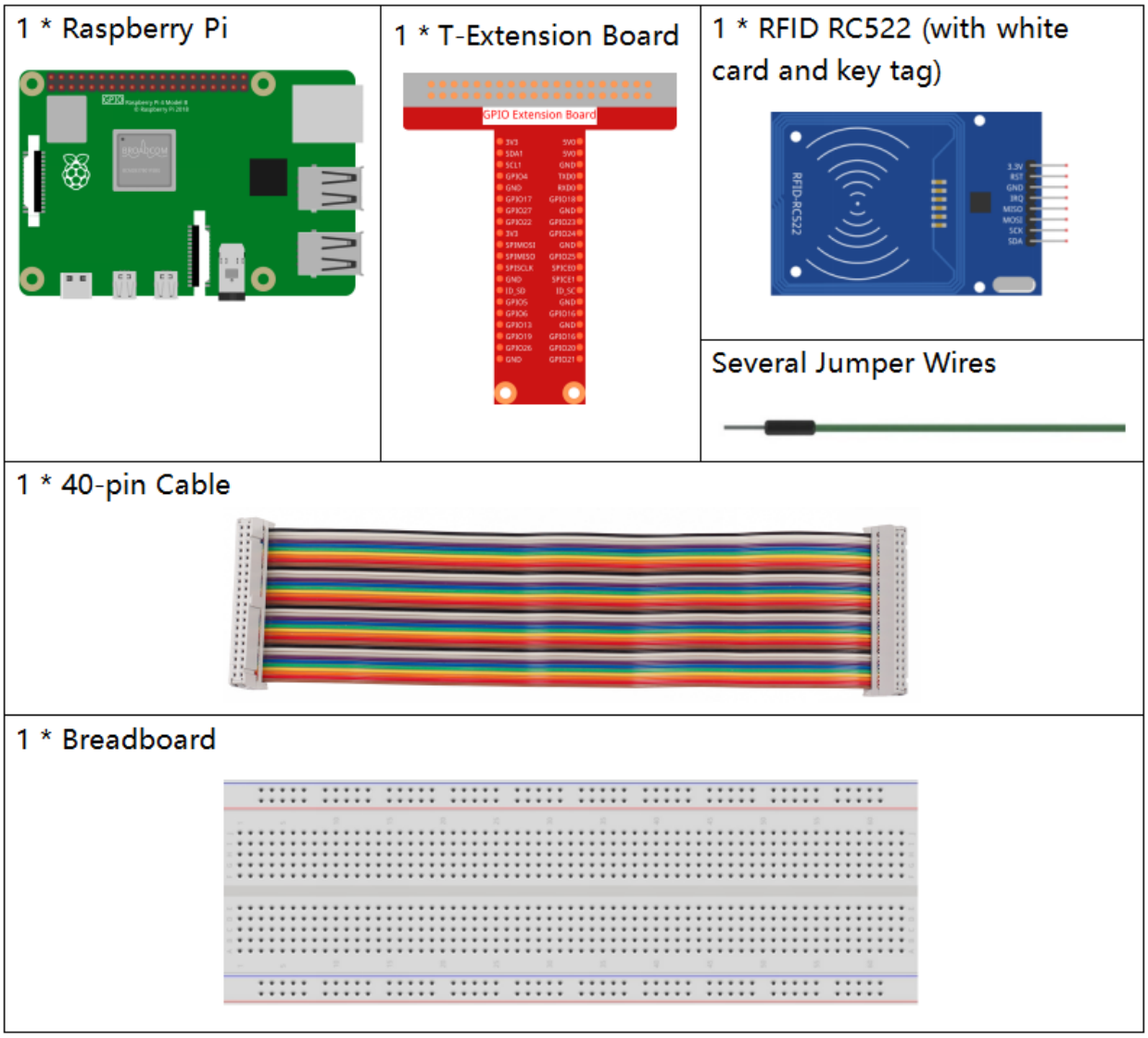
前書き

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



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。

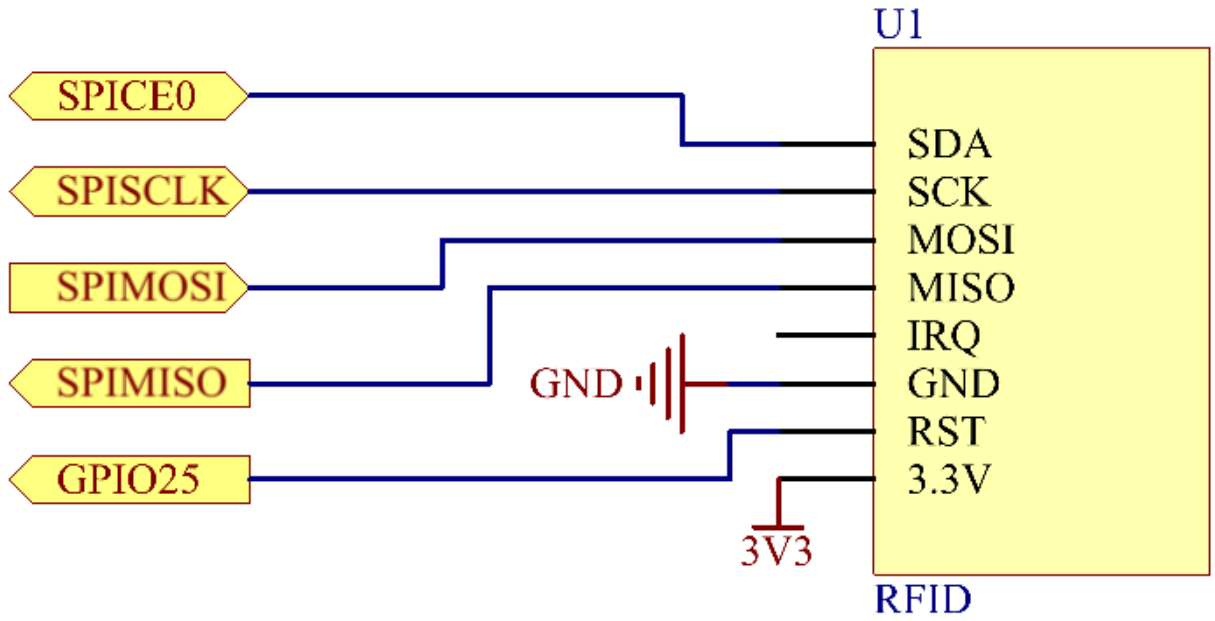
MFRC522

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転びやすい手持ち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。

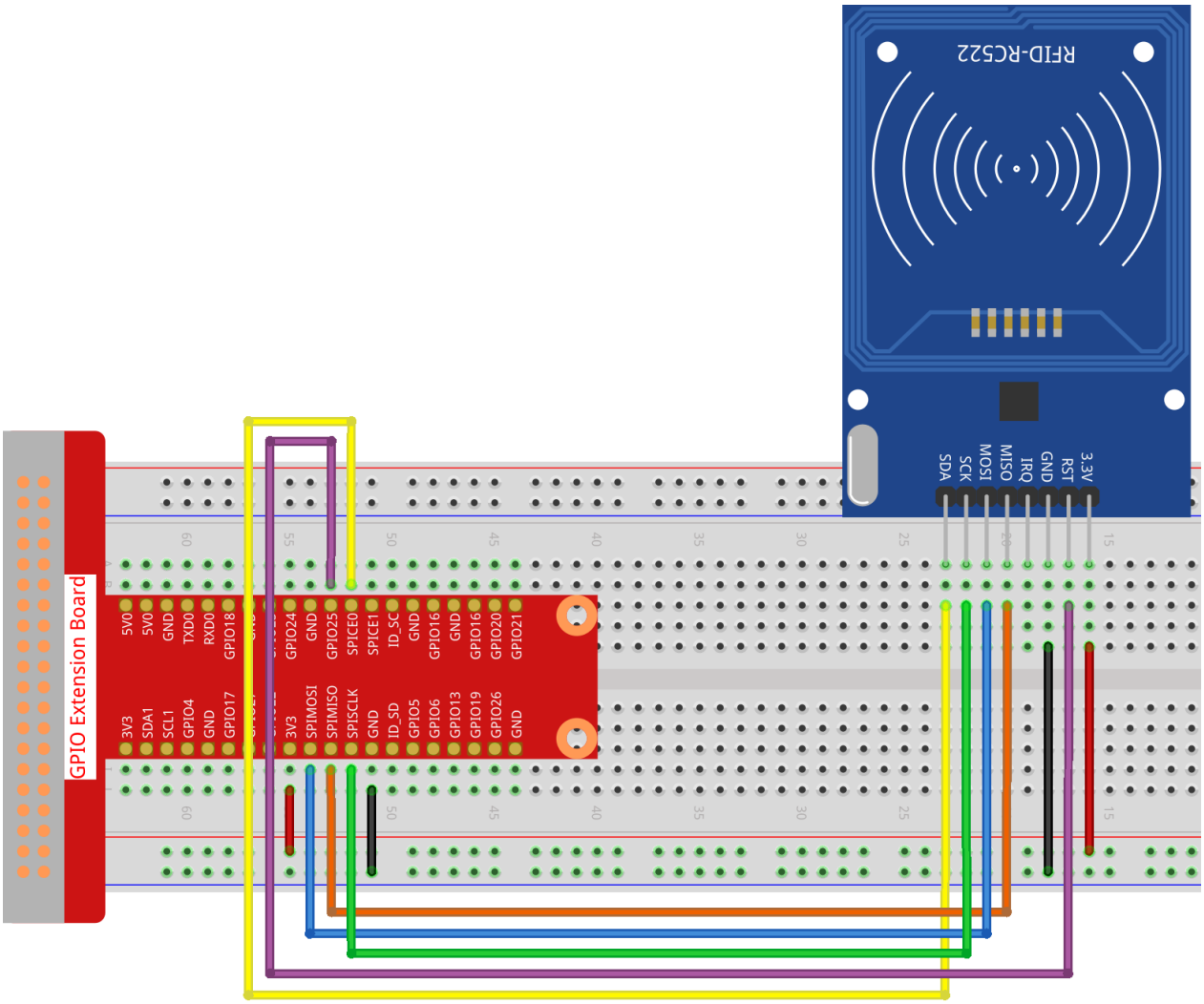
回路図

T-Board Name	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* **構成** (SPI を設定している場合は、このステップ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
```

注釈: 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、読み取りが成功すると「1」が返されます。

```
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」を返します。 *data は、カードに書き込まれる情報です。

Python 言語ユーザー向け

ステップ 2: 仮想環境の有効化。

注意:

- 有効化する前に、仮想環境を作成していることを確認してください。詳細はこちらを参照してください: **仮想環境の作成**.
- Raspberry Pi を再起動するたびや、新しいターミナルを開くたびに、仮想環境を有効化するために次のコマンドを再度実行する必要があります。

```
source myenv/bin/activate
```

仮想環境が有効化されると、コマンドラインのプロンプトの前に環境名が表示され、仮想環境内で作業していることが示されます。

ステップ 3: ライブラリのインストール。

spidev ライブラリは SPI とのやり取りを処理し、このチュートリアルの重要なコンポーネントであり、Raspberry Pi が RFID RC522 とのやり取りに必要です。

以下のコマンドを実行して、 pip を介して spidev を Raspberry Pi にインストールします。

```
sudo pip3 install spidev
```

MFRC522 ライブラリのインストールを続行します。 MFRC522 ライブラリには 2 つのファイルが含まれています: MFRC522.py と SimpleMFRC522.py。

MFRC522.py

は RFID RC522 インターフェースの実装であり、このライブラリは Pi の SPI インターフェースを介して RFID との通信のすべての重い作業を処理します。

SimpleMFRC522.py は MFRC522.py ファイルを取り、わずかな関数の代わりにわずかな関数で処理できるように大幅に簡素化します。

```
sudo pip3 install mfrc522
```

ステップ 4: 仮想環境の終了。

作業を完了し、仮想環境から退出したい場合は、単純に次のコマンドを実行します:

```
deactivate
```

これにより、システムのグローバル Python 環境に戻ります。

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/2.2.7
```

ステップ 6： EXE ファイルを実行する。

```
sudo python3 2.2.7_read.py
```

```
sudo python3 2.2.7_write.py
```

注釈: 2.2.7_read.py を実行すると、カードの ID とテキストがシェルに出力されます。2.2.7_write.py を実行した後、最初にメッセージを書き込み、Enter を押して確認し、最後にカードを MFRC522 RFID モジュールに置いて書き込みを完了する必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
# read
import time
from mfrc522 import SimpleMFRC522
import RPi.GPIO as GPIO

reader = SimpleMFRC522()

def main():
    while True:
        print("Reading...Please place the card...")
        id, text = reader.read()
        print("ID: %s\nText: %s" % (id,text))
        time.sleep(3)

def destroy():
    GPIO.cleanup()

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

```
# write
from mfrc522 import SimpleMFRC522
import RPi.GPIO as GPIO

reader = SimpleMFRC522()

def main():
    while True:
        text = input('Please write new data:')
```

(次のページに続く)

(前のページからの続き)

```
        print("Please place the card to complete writing")
        reader.write(text)
        print("Data writing is complete")

def destroy():
    GPIO.cleanup()

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

コードの説明

```
reader = SimpleMFRC522()
```

SimpleMFRC522() クラスをインスタンス化します。

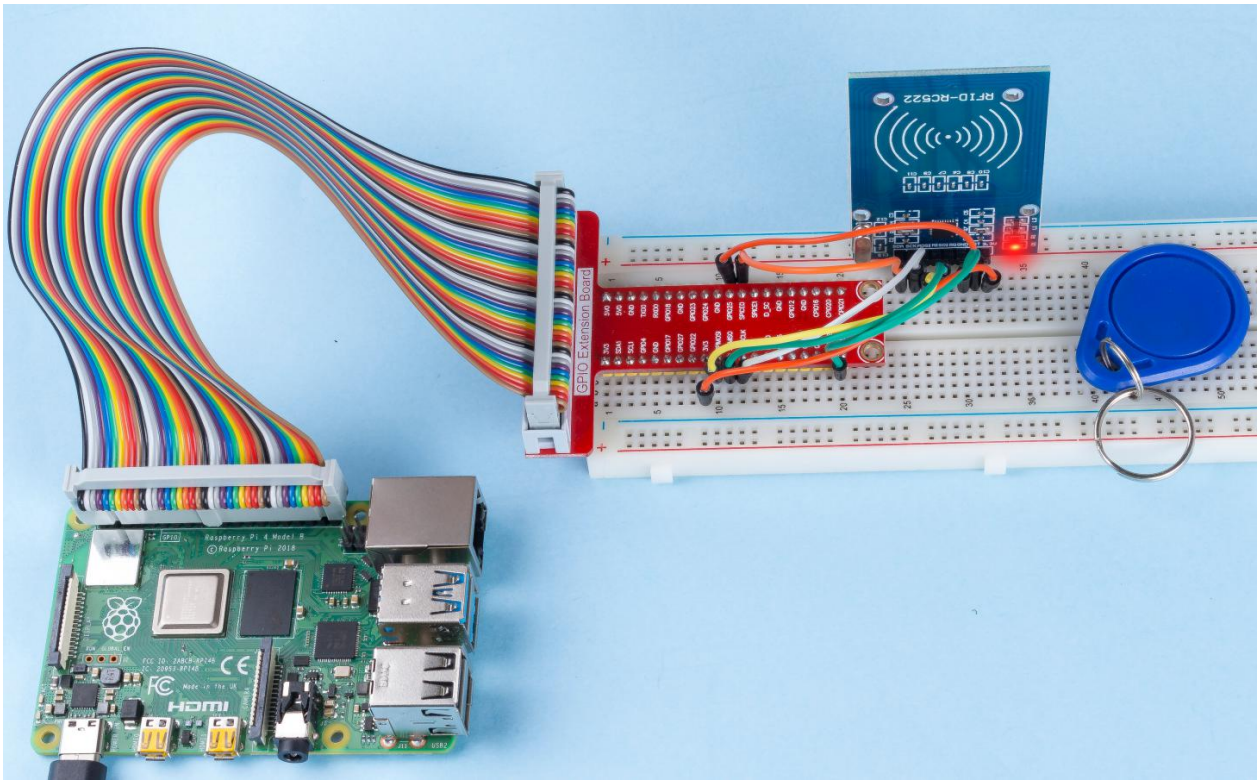
```
reader.read()
```

この機能は、カードデータを読み取るために使用されます。 読み取りが成功すると、ID とテキストが返されます。

```
reader.write(text)
```

この機能は、カードに情報を書き込むために使用されます。「Enter」キーを押して書き込みを終了します。 `text` は、カードに書き込まれる情報です。

現象画像



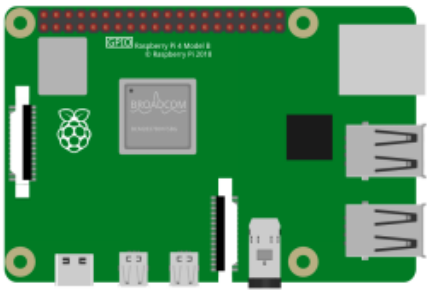
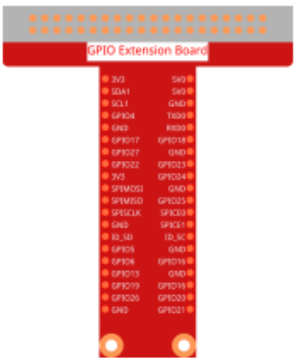
1.10 3 拡張 (Pi 5 用ではない)

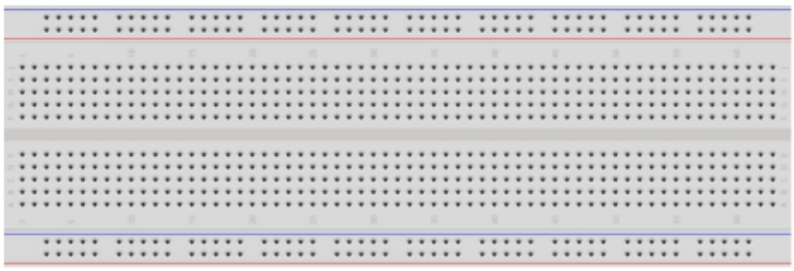
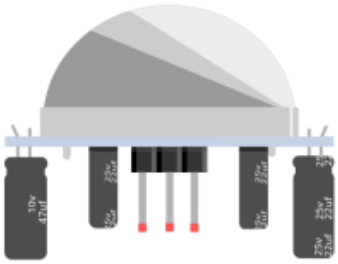
1.10.1 3.1.1 計数装置

前書き

ここでは、PIR センサーと 4 桁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 桁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。

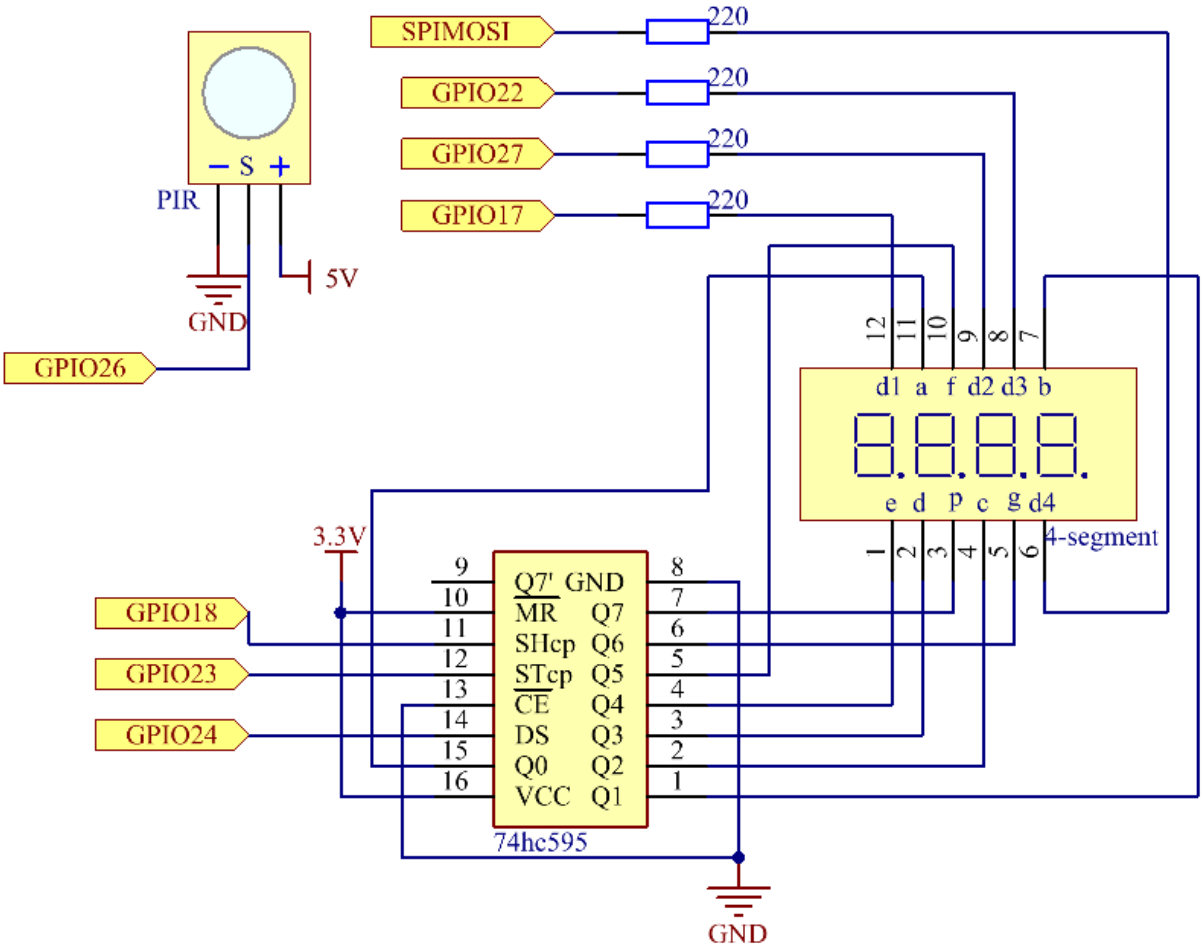
部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display
		4 * Resistor(220Ω)
1 * 40-pin Cable		1 * 74HC595

	Several Jumper Wires
1 * Breadboard 	1 * PIR Sensor Module 

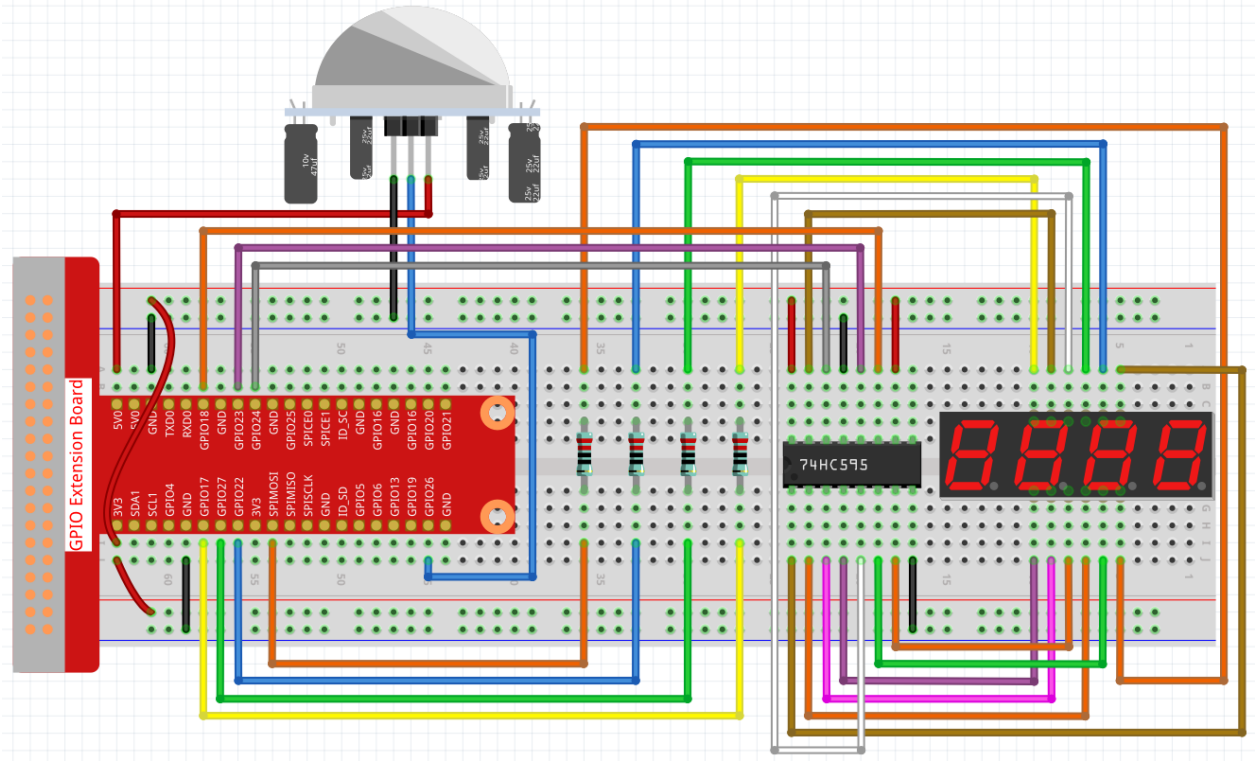
回路図

T-Board Name	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 桁のセグメントディスプレイの数字に 1 が加算される。

PIR モジュールには 2 つのポテンショメータがあり、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す。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。

コードの説明

```
void^^c2^^a0display()
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a010]);

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(1);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a0100^^c2^^a0/
↪^^c2^^a010]);

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(2);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a01000^^c2^^a0/
↪^^c2^^a0100]);

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(3);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a010000^^c2^^a0/
↪^^c2^^a01000]);
}
```

まず、4 番目のセグメントディスプレイを開始し、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。

```
void^^c2^^a0loop(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentState^^c2^^a0=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0lastState=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=digitalRead(sensorPin);
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if((currentState==0)&&
↪(lastState==1)){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0counter
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

sensorPin = 26

SDI = 24
RCLK = 23
SRCLK = 18

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

counter = 0
```

(次のページに続く)

(前のページからの続き)

```
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 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])
```

(次のページに続く)

(前のページからの続き)

```
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

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)
    GPIO.setup(sensorPin, GPIO.IN)

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

if __name__ == '__main__':  # Program starting from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

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^^c2^^a0display():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0counter
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(0)
```

(次のページに続く)

(前のページからの続き)

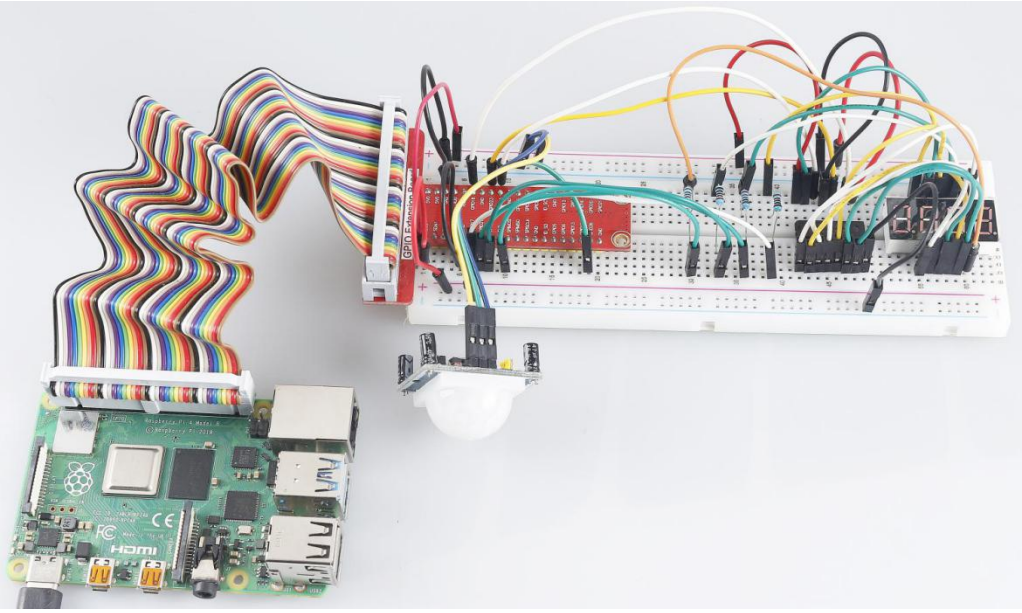
```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a010])

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a0100//10])

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(2)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a01000//100])

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0clearDisplay()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pickDigit(3)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(number[counter^^c2^^a0%^^c2^^a010000//1000])
```

まず、4 番目のセグメントディスプレイを開始し、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^^c2^^a0loop():
global^^c2^^a0counter
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=GPIO.
↪input(sensorPin)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(currentState^^c2^^a0==^^c2^^a
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0counter^
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=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

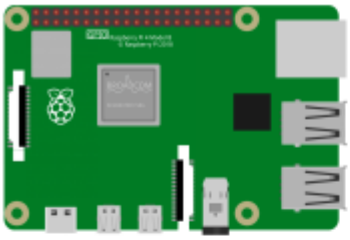
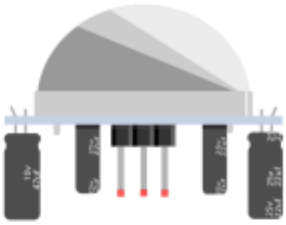
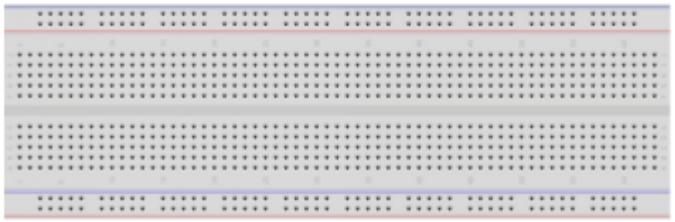


1.10.2 3.1.2 いらっしゃいませ

前書き

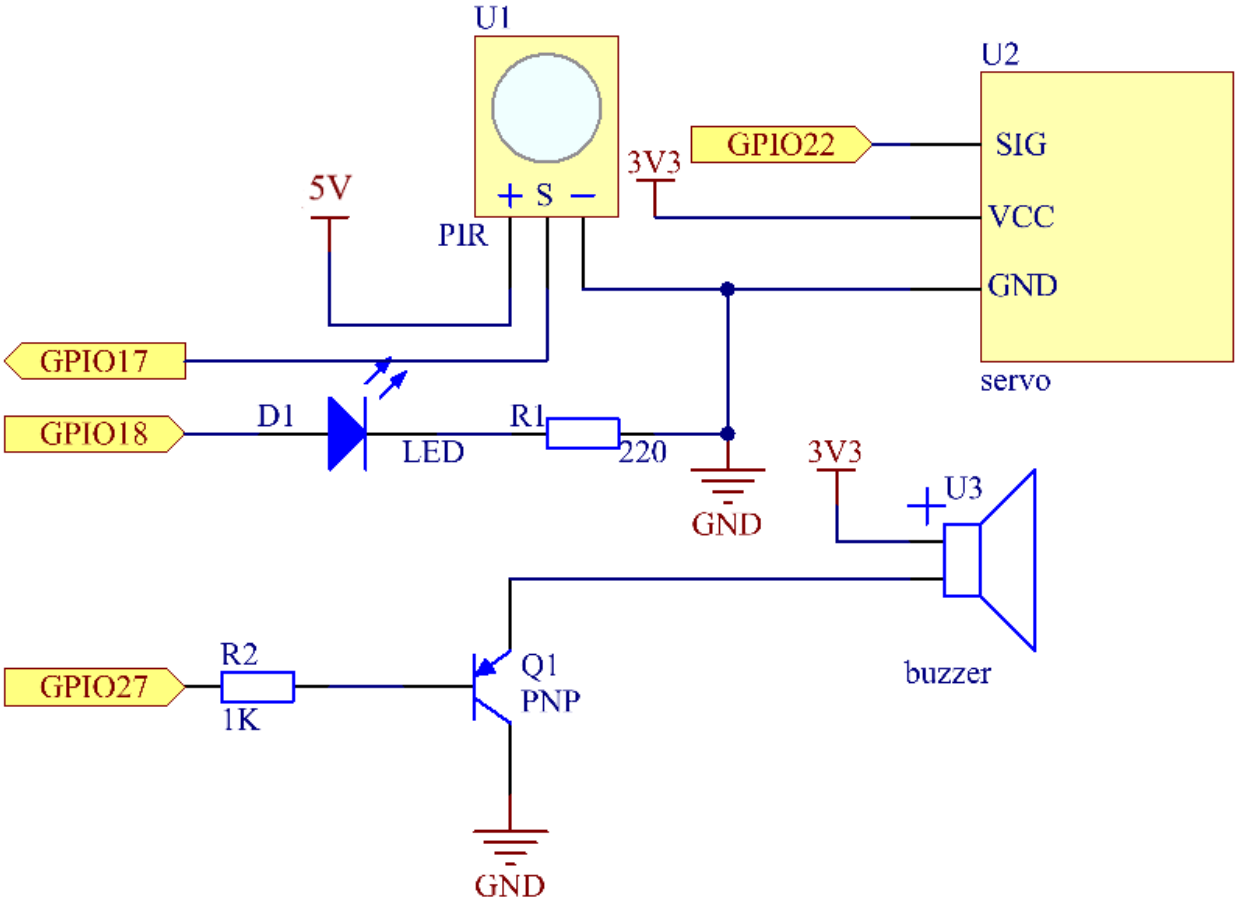
このプロジェクトでは、PIR を使用して歩行者の動きを検知し、サーボ、LED、ブザーを使用してコンビニのセンサードアの動作をシミュレートする。歩行者が PIR の検知範囲内に現れると、インジケータライトが点灯し、ドアが開き、ブザーがオープニングベルを鳴らす。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * PIR 	
1 * Servo	1 * Passive Buzzer	1 * S8550 PNP Transistor	1 * LED
1 * Resistor 1kΩ	1 * Resistor 220Ω	Several Jumper Wires	
1 * 40-pin Cable			
1 * Breadboard 			

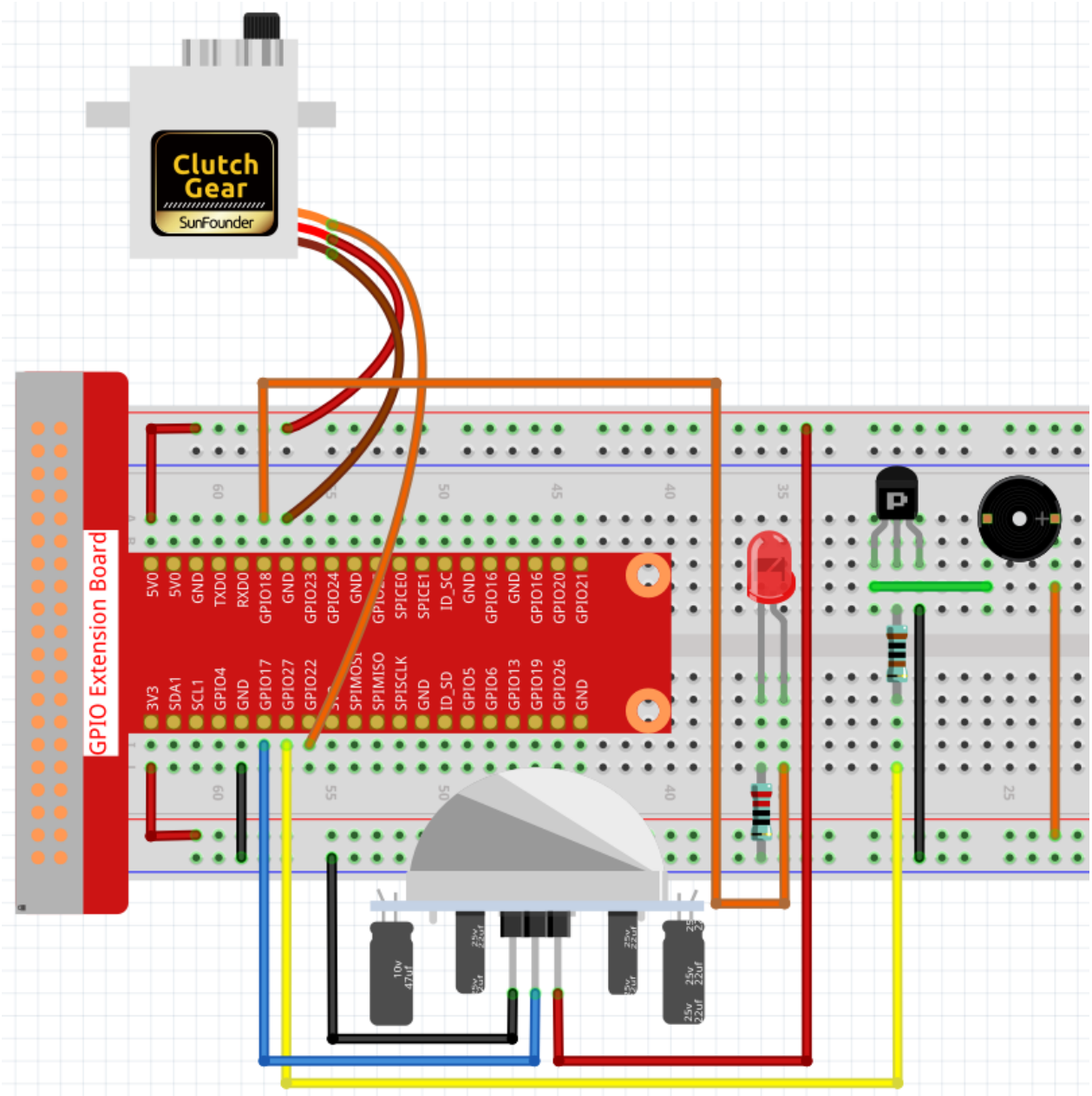
回路図

T-Board Name	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。

PIR モジュールには 2 つのポテンショメータがあり、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す。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。

コードの説明

```
void setAngle(int pin, int angle){    //Create a funtion 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。

PIR モジュールには 2 つのポテンショメータがあり、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す。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

ledPin = 18    # define the ledPin
pirPin = 17    # define the sensorPin
servoPin = 22  # define the servoPin
buzPin = 27    # define the buzzerpin


CL = [0, 131, 147, 165, 175, 196, 211, 248]        # Frequency of Low C notes

CM = [0, 262, 294, 330, 350, 393, 441, 495]        # Frequency of Middle C notes

CH = [0, 525, 589, 661, 700, 786, 882, 990]        # Frequency of High C notes

song = [    CH[5],CH[2],CM[6],CH[2],CH[3],CH[6],CH[3],CH[5],CH[3],CM[6],CH[2]    ]

beat = [    1,1,1,1,1,2,1,1,1,1,1,]


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
```

(次のページに続く)

(前のページからの続き)

```
    GPIO.setup(servoPin, GPIO.OUT)   # Set servoPin's mode is output
    GPIO.output(servoPin, GPIO.LOW)  # Set servoPin to low
    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

    p = GPIO.PWM(servoPin, 50)     # set Frequece to 50Hz
    p.start(0)                     # Duty Cycle = 0

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


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 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
    time.sleep(1)                        # Wait a second for next song.

def closedoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
        setAngle(i)
        time.sleep(0.001)
    time.sleep(1)
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)
    time.sleep(1)
    doorbell()
```

(次のページに続く)

(前のページからの続き)

```
    closedoor()

def loop():
    while True:
        if GPIO.input(pirPin)==GPIO.HIGH:
            opendoor()


def destroy():
    GPIO.cleanup()                     # Release resource
    p.stop()
    Buzz.stop()

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

```
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 Frequece 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 song1
        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。

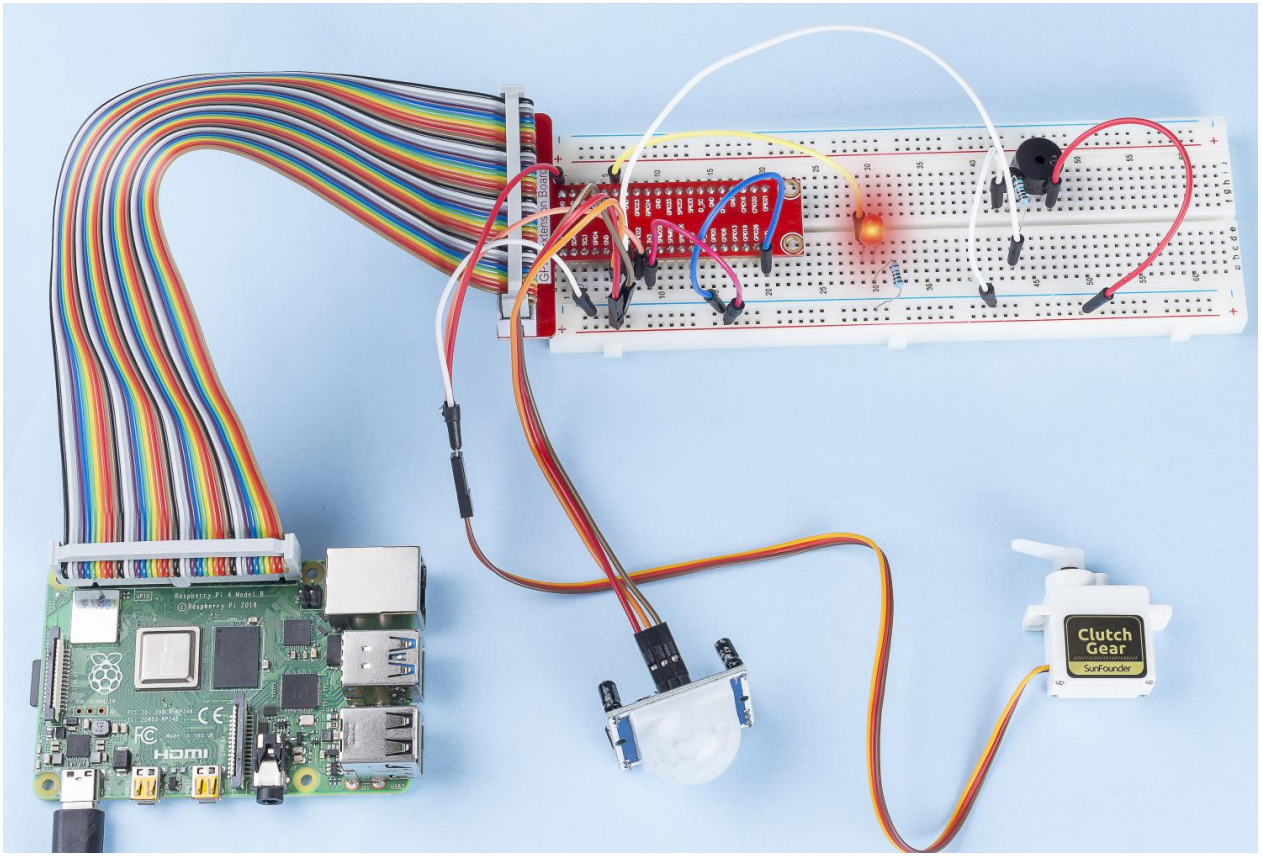
```
def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1):   #make servo rotate from 0 to 180 deg
        setAngle(i)     # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()
```

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()
```

PIR 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

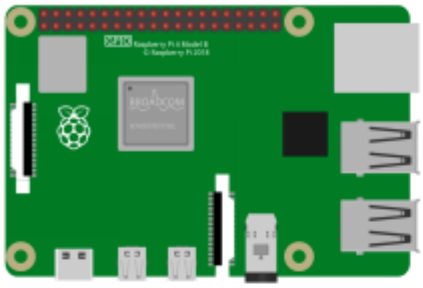
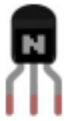
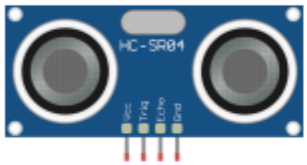
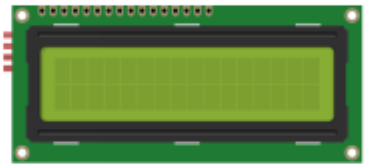
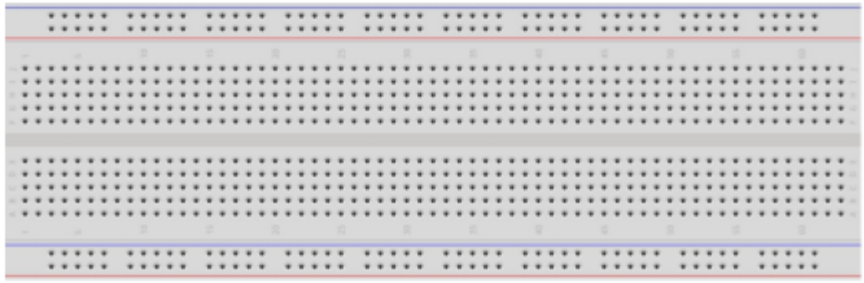


1.10.3 3.1.3 後退警報装置

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。

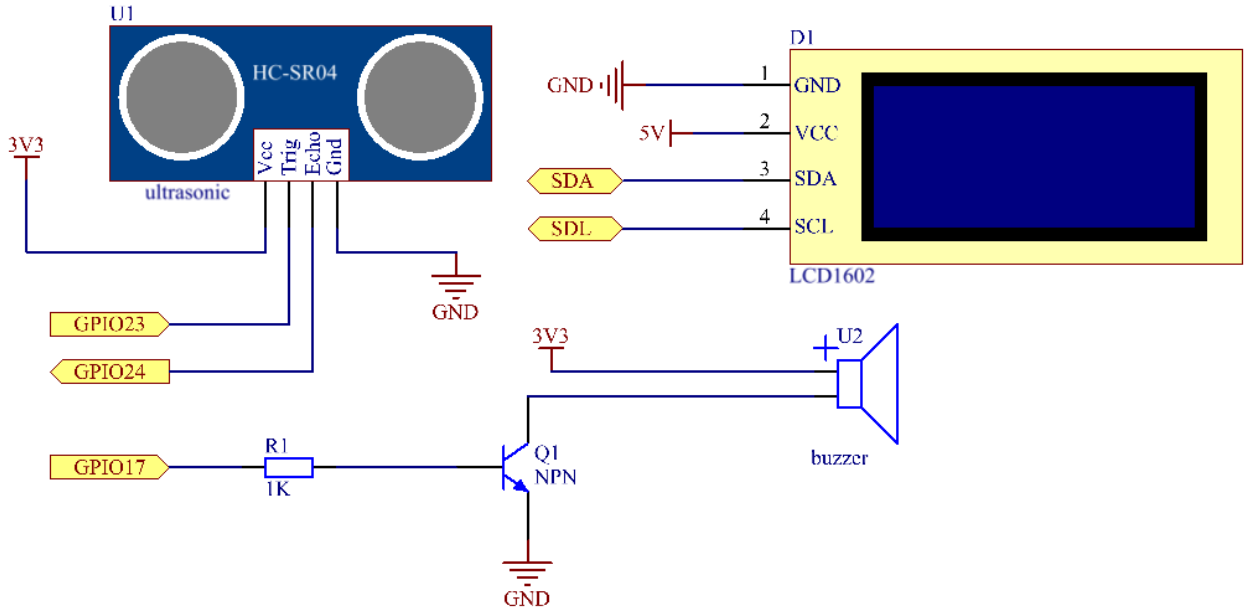
部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * Active Buzzer 1 * S8050 NPN Transistor 
1 * HC SR04 	1 * I2C LCD1602 	Several Jumper Wires 1 * Resistor(1kΩ)
1 * 40-pin Cable		
1 * Breadboard 		

回路図

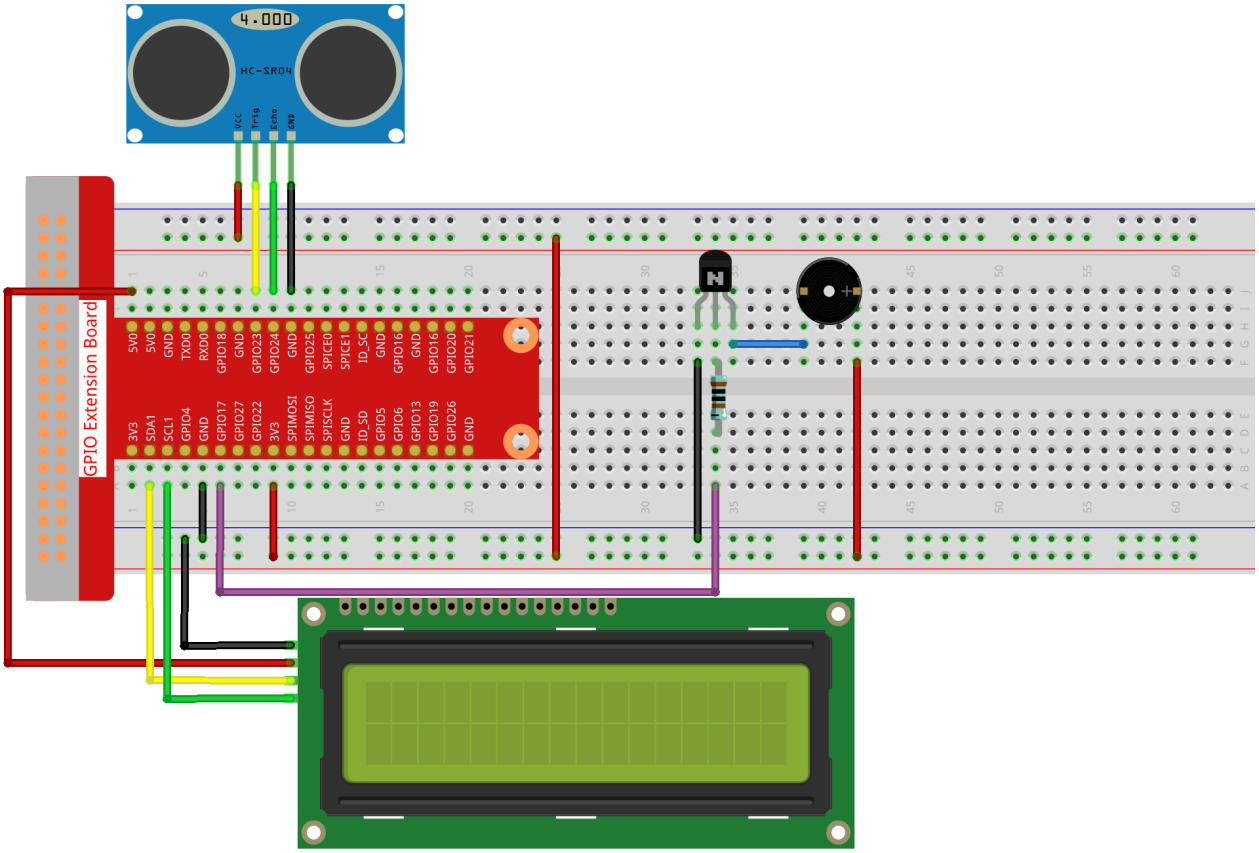
超音波センサーは障害物との間の距離を検出し、コードの形式で LCD に表示する。同時に、超音波センサーにより、距離値に応じてブザーが異なる周波数の警告音を発する。

T-Board Name	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

注釈: 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 ultraInit(void){...}

float disMeasure(void){...}

//here is the main function
int main(void)
```

(次のページに続く)

(前のページからの続き)

```
{
    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);
        delay(100);
        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++){
```

(次のページに続く)

(前のページからの続き)

```
                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
{
```

(次のページに続く)

(前のページからの続き)

```
    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 桁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);
    }
}
```

この判定条件でブザーの音をコントロールします。距離の違いにより、音の周波数が異なる 3 つのケースに分けられます。遅延の合計値は 500 なので、どのケースも超音波センサに 500ms の間隔を与えることができる。

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
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()
```

(次のページに続く)

(前のページからの続き)

```
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):
```

(次のページに続く)

(前のページからの続き)

```
            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')
```

(次のページに続く)

(前のページからの続き)

```
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')
```

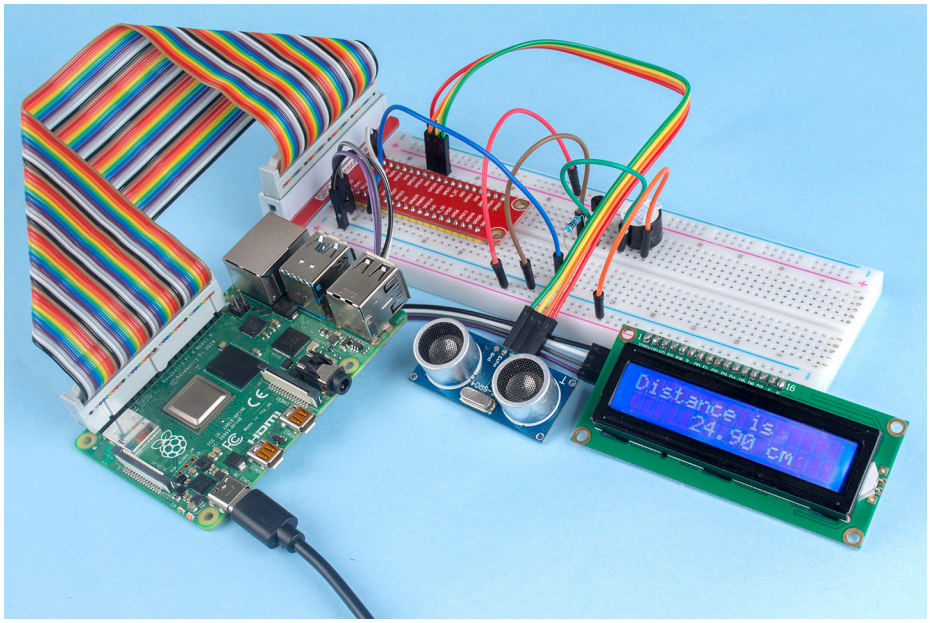
LCD 出力は文字タイプのみをサポートするため、 str() を使用して数値を文字に変換する必要がある。小数点以下 2 桁に丸める

```
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

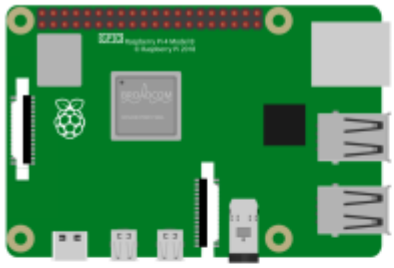
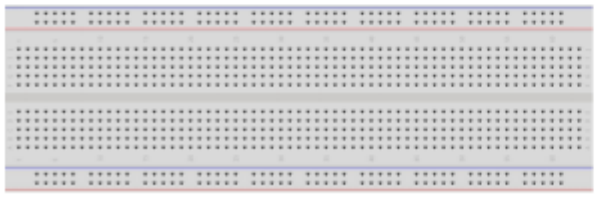


1.10.4 3.1.4 スマートファン

前書き

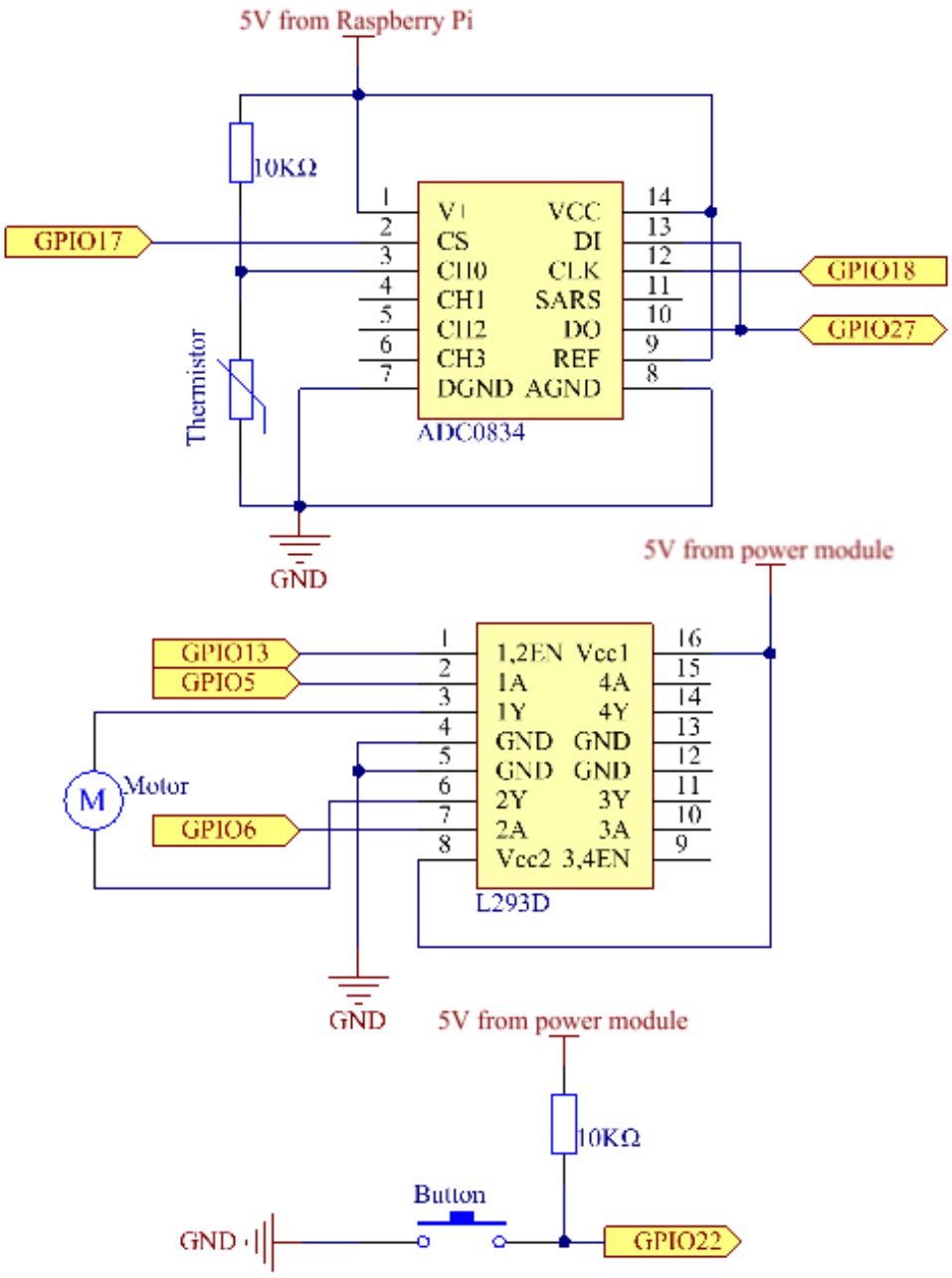
このコースでは、モーター、ボタン、サーミスターを使用して、風速が調整可能な手動 + 自動のスマートファンを作成する。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * Power Module
		2 * Resistor 10KΩ
1 * 40-pin Cable	1 * Thermistor	1 * L293D
		1 * ADC0834
1 * Breadboard 	1 * Button	1 * DC Motor
		Several Jumper Wires

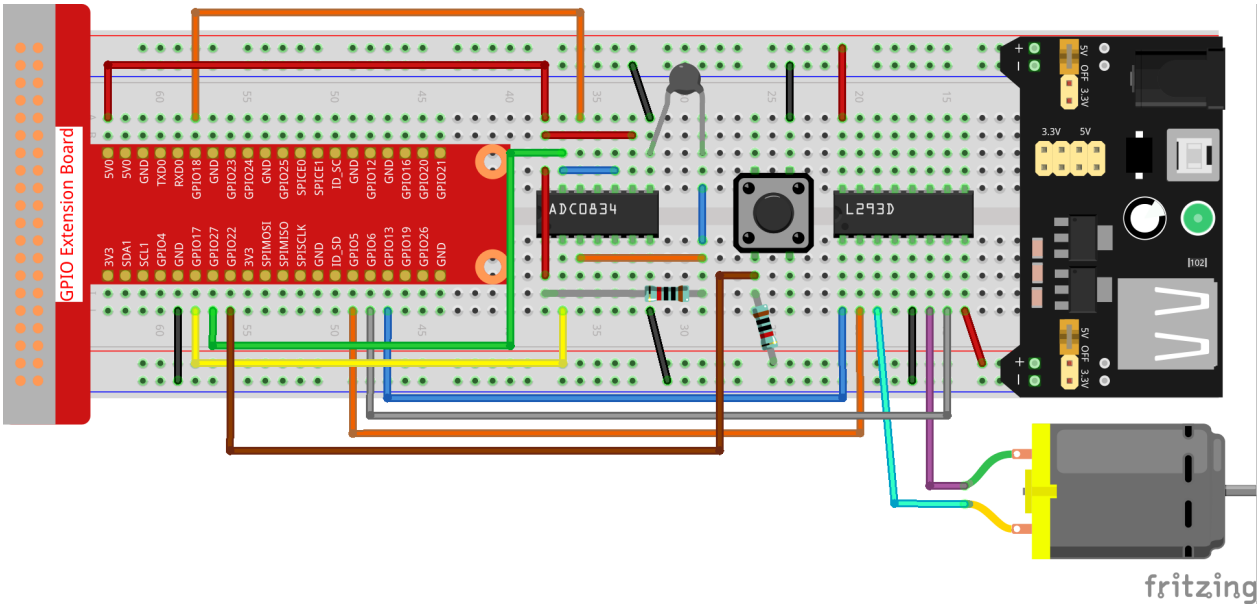
回路図

T-Board Name	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。



注釈: 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 ℃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^^c2^^a0temperture(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0analogVal;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0double^^c2^^a0Vr,^^c2^^a0Rt,^^c2^^a0temp,^^c2^^a0cel,
↪^^c2^^a0Fah;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0analogVal^^c2^^a0=^^c2^^a0get_ADC_Result(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Vr^^c2^^a0=^^c2^^a05^^c2^^a0*^^c2^^a0(double)(analogVal)^^c2^^a0/
↪^^c2^^a0255;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Rt^^c2^^a0=^^c2^^a010000^^c2^^a0*^^c2^^a0(double)(Vr)^^c2^^a0/
↪^^c2^^a0(5^^c2^^a0-^^c2^^a0(double)(Vr));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0temp^^c2^^a0=^^c2^^a01^^c2^^a0/^^c2^^a0(((log(Rt/
↪10000))^^c2^^a0/^^c2^^a03950)+(1^^c2^^a0/^^c2^^a0(273.15^^c2^^a0+^^c2^^a025)));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0cel^^c2^^a0=^^c2^^a0temp^^c2^^a0-^^c2^^a0273.15;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Fah^^c2^^a0=^^c2^^a0cel^^c2^^a0*^^c2^^a01.8^^c2^^a0+32;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0t=cel;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0t;
}
```

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って機能します。詳細については、 *2.2.2* **サーミスタ** を参照してください。

```
int^^c2^^a0motor(int^^c2^^a0level){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(level==0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(MotorEnable,
↪LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(level>=4){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level^^c2^^a0=4;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(MotorEnable,HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0softPwmWrite(MotorPin1,^^c2^^a0level*25);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0level;
}
```

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^^c2^^a0main(void)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentState,lastState=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0level^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentTemp,markTemp=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=digitalRead(BtnPin);
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentTemp=temperture();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(currentTemp
↪<=0){continue;}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(currentState==1&
↪&lastState==0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level=(
↪%5;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0markTemp
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(5
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(level!=0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^
↪markTemp<=-2){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a
↪1;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^
↪markTemp>=2){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
↪
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level=motor(level);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

関数 main() 上 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 れベル 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 ℃ + 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 ℃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

注 釈: 以 下 の コ ー ド を **変 更/リ セ ッ ト/コ ピ ー/実 行/停 止** で き ま す 。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time
import ADC0834
import math
```

(次のページに続く)

(前のページからの続き)

```
# Set up pins
MotorPin1   = 5
MotorPin2   = 6
MotorEnable = 13
BtnPin  = 22


def setup():
    global p_M1,p_M2
    ADC0834.setup()
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(MotorPin1, GPIO.OUT)
    GPIO.setup(MotorPin2, GPIO.OUT)
    p_M1=GPIO.PWM(MotorPin1,2000)
    p_M2=GPIO.PWM(MotorPin2,2000)
    p_M1.start(0)
    p_M2.start(0)
    GPIO.setup(MotorEnable, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(BtnPin, GPIO.IN)

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

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
```

(次のページに続く)

(前のページからの続き)

```
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)


def destroy():
    GPIO.output(MotorEnable, GPIO.LOW)
    p_M1.stop()
    p_M2.stop()
    GPIO.cleanup()

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

コードの説明

```
def^^c2^^a0temperature():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0analogVal^^c2^^a0=^^c2^^a0ADC0834.getResult()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Vr^^c2^^a0=^^c2^^a05^^c2^^a0*^^c2^^a0float(analogVal)^^c2^^a0/
↪^^c2^^a0255
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Rt^^c2^^a0=^^c2^^a010000^^c2^^a0*^^c2^^a0Vr^^c2^^a0/
↪^^c2^^a0(5^^c2^^a0-^^c2^^a0Vr)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0temp^^c2^^a0=^^c2^^a01/(((math.log(Rt^^c2^^a0/
↪^^c2^^a010000))^^c2^^a0/^^c2^^a03950)^^c2^^a0+^^c2^^a0(1^^c2^^a0/^^c2^^a0(273.15+25)))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Cel^^c2^^a0=^^c2^^a0temp^^c2^^a0-^^c2^^a0273.15
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Fah^^c2^^a0=^^c2^^a0Cel^^c2^^a0*^^c2^^a01.
↪8^^c2^^a0+^^c2^^a032
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0Cel
```

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って機能します。詳細については、 *2.2.2* **サーミスタ** を参照してください。

```
def^^c2^^a0motor(level):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0level^^c2^^a0==^^c2^^a00:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(MotorEnable,
↪^^c2^^a0GPIO.LOW)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0level>=4:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level^^c2^^a0=^^c2^^a04
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(MotorEnable,^^c2^^a0GPIO.HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0p_M1.ChangeDutyCycle(level*25)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0level
```

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^^c2^^a0main():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level=0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0markTemp^^c2^^a0=^^c2^^a0temperature()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState^^c2^^a0=GPIO.
↪input(BtnPin)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentTemp=temperature()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0currentState^^c2^^a0==^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level=(
```

(次のページに続く)

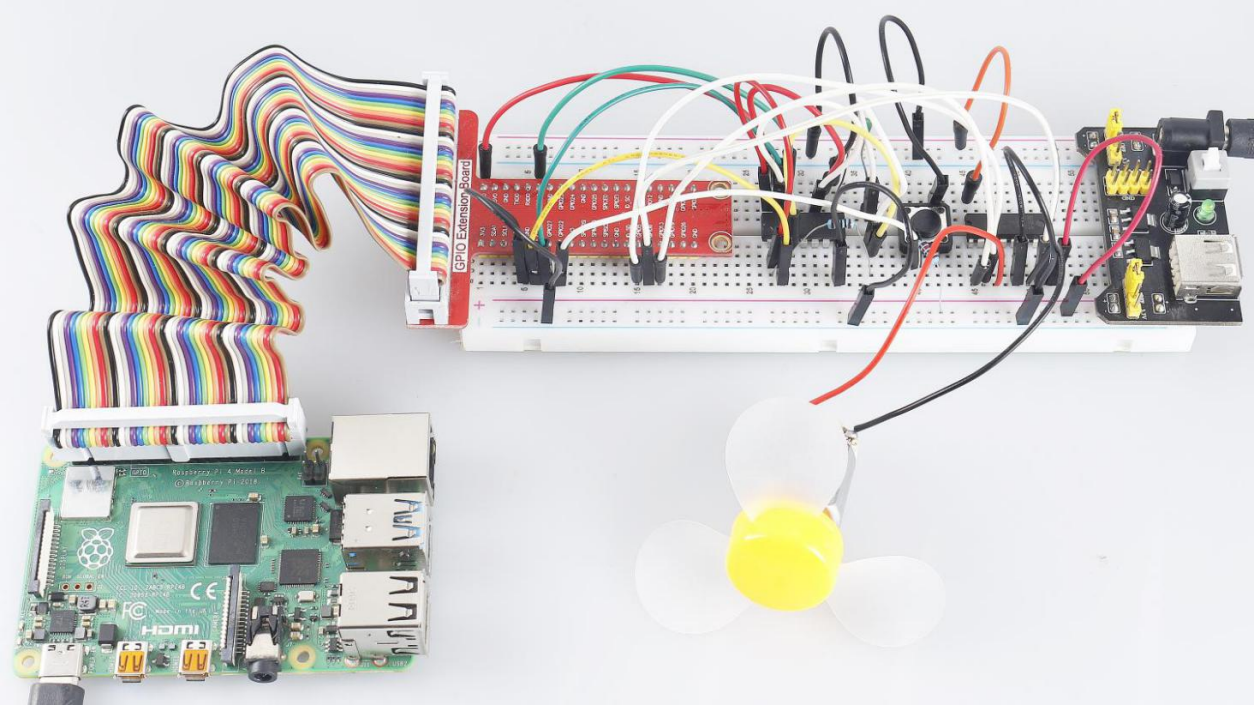
(前のページからの続き)

```
↪%5
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0markTemp
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.
↪sleep(0.5)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0level!=0:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^
↪markTemp^^c2^^a0<=^^c2^^a0-2:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪1
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^
↪markTemp^^c2^^a0>=^^c2^^a02:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0level^^c2^^a0=^^c2^^a0motor(level)
```

関数 main() 上 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 れベル 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 ℃ + 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

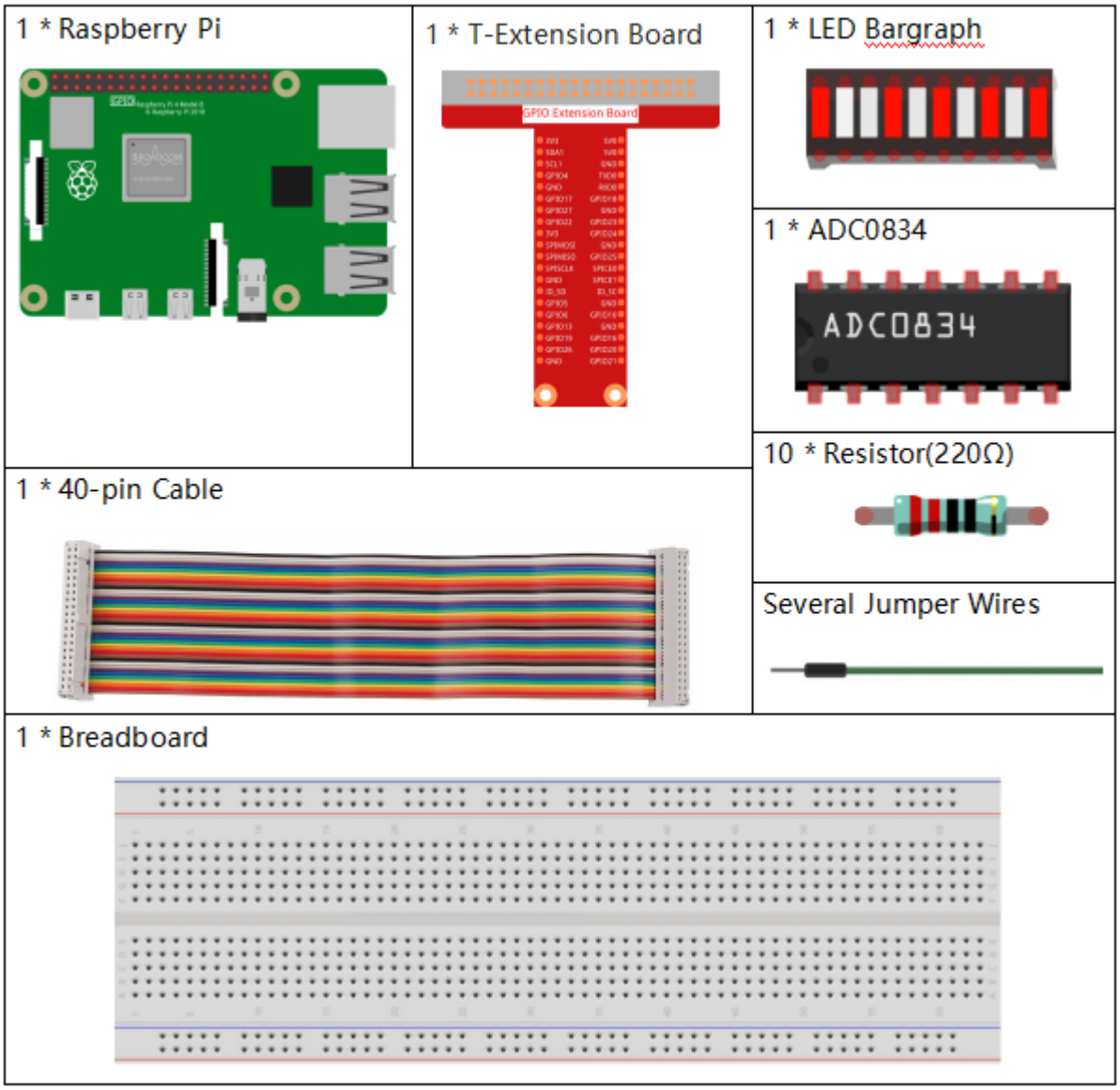


1.10.5 3.1.5 バッテリーインジケーター

前書き

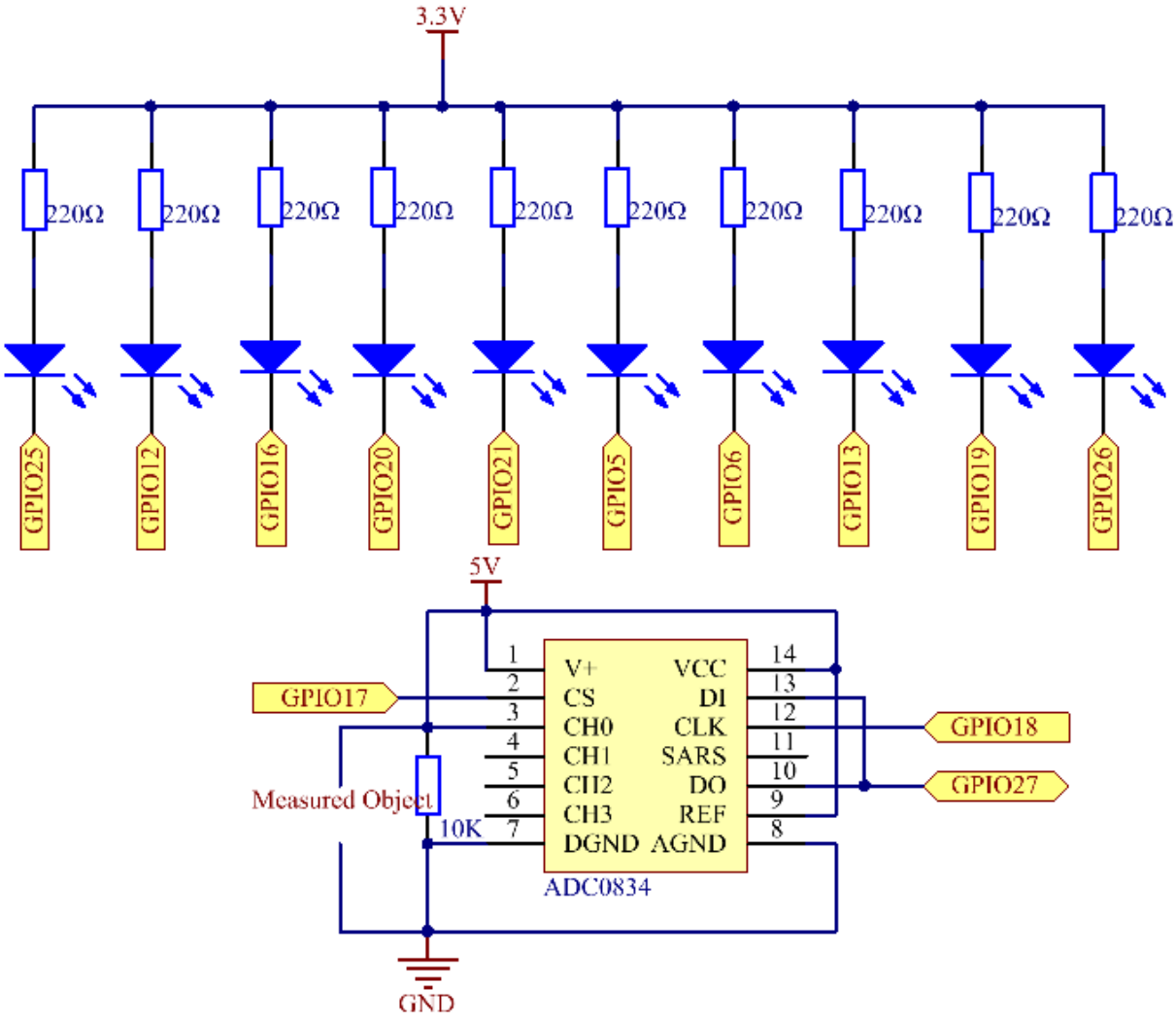
このコースでは、LED バーグラフにバッテリーレベルを視覚的に表示できるバッテリー指示装置を作成する。

部品


1 * Raspberry Pi
1 * T-Extension Board
GPIO Extension Board
1 * LED Bargraph
1 * ADC0834
ADC0834
10 * Resistor(220Ω)
1 * 40-pin Cable
Several Jumper Wires
1 * Breadboard

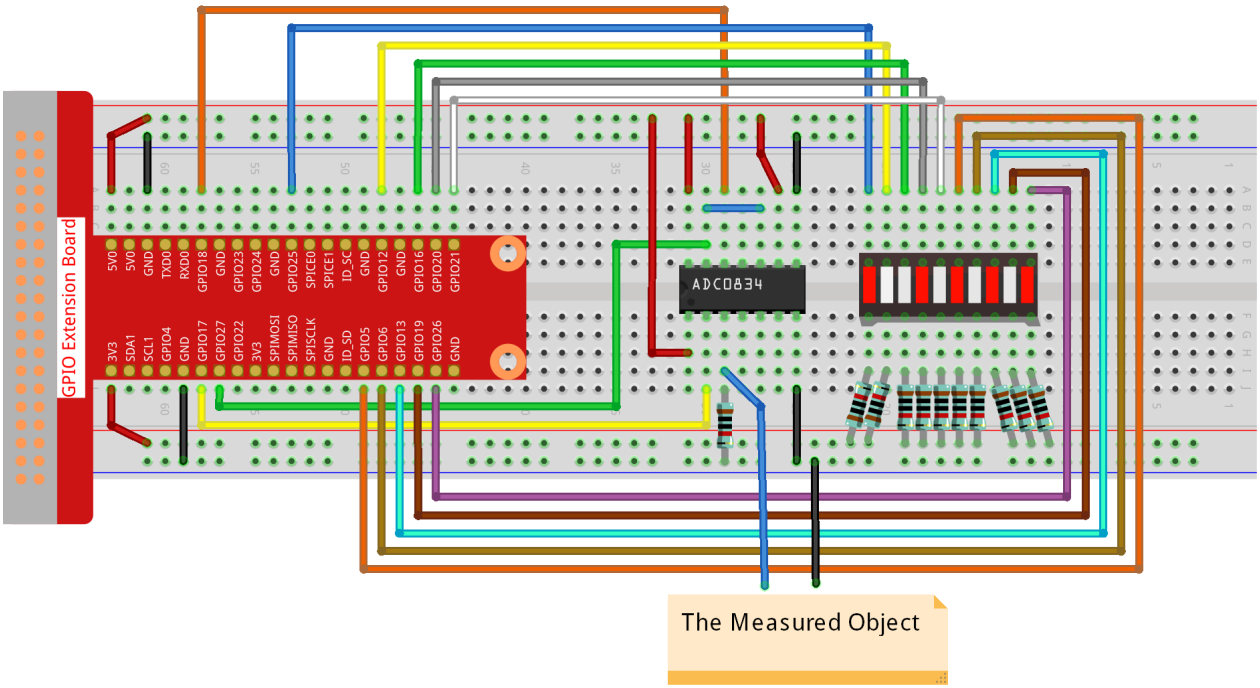
回路図

T-Board Name	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^^c2^^a0LedBarGraph(int^^c2^^a0value){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0i=0;i<10;i++){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(pins[i],
↪HIGH);
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0i=0;i<value;i++){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(pins[i],
↪LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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^^c2^^a0main(void)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0uchar^^c2^^a0analogVal;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(wiringPiSetup()^^c2^^a0==^^c2^^a0-1){^^c2^^a0//
↪when^^c2^^a0initialize^^c2^^a0wiring^^c2^^a0failed,print^^c2^^a0messageto^^c2^^a0screen
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf(
↪"setup^^c2^^a0wiringPi^^c2^^a0failed^^c2^^a0!");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pinMode(ADC_CS,^^c2^^a0^^c2^^a0OUTPUT);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pinMode(ADC_CLK,^^c2^^a0OUTPUT);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0i=0;i<10;i++)
↪{^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0//make^^c2^^a0led^^c2^^a0pins
↪'^^c2^^a0mode^^c2^^a0is^^c2^^a0output
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pinMode(pins[i],
↪^^c2^^a0OUTPUT);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(pins[i],
↪HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0analogVal^^c2^^a0=^^c2^^a0get_
↪ADC_Result(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LedBarGraph(analogVal/
↪25);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(100);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import ADC0834
import time

ledPins = [25, 12, 16, 20, 21, 5, 6, 13, 19, 26]

def setup():
    GPIO.setmode(GPIO.BCM)
    ADC0834.setup()
    for i in ledPins:
        GPIO.setup(i, GPIO.OUT)
        GPIO.output(i, GPIO.HIGH)

def LedBarGraph(value):
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
```

(次のページに続く)

(前のページからの続き)

```
    for i in range(value):
        GPIO.output(ledPins[i],GPIO.LOW)

def destroy():
    GPIO.cleanup()

def loop():
    while True:
        analogVal = ADC0834.getResult()
        LedBarGraph(int(analogVal/25))

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^^c2^^a0LedBarGraph(value):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0i^^c2^^a0in^^c2^^a0ledPins:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(i,GPIO.HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0i^^c2^^a0in^^c2^^a0range(value):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.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。

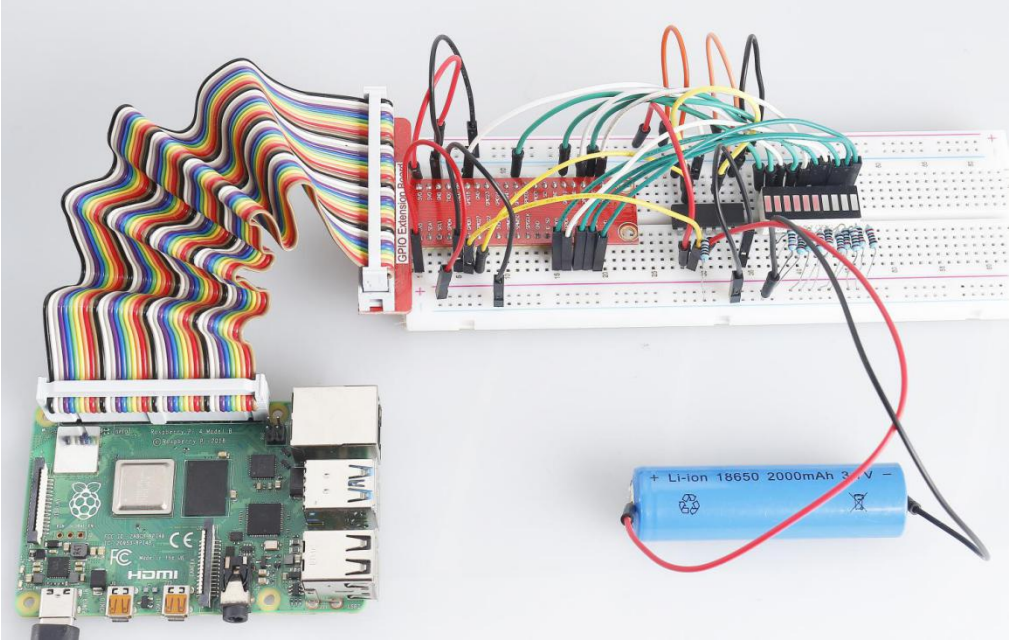
```
def^^c2^^a0loop():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0analogVal^^c2^^a0=^^c2^^a0ADC0834.
↪getResult()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LedBarGraph(int(analogVal/
↪25))
```

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。

現象画像

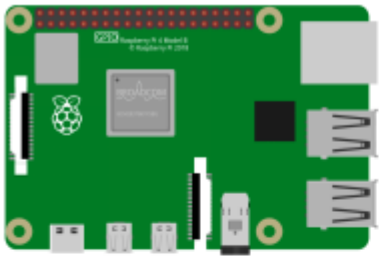
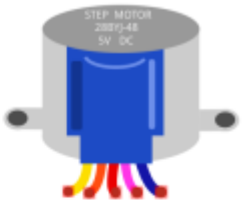
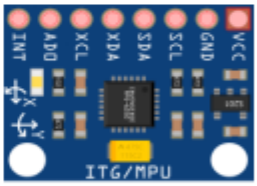
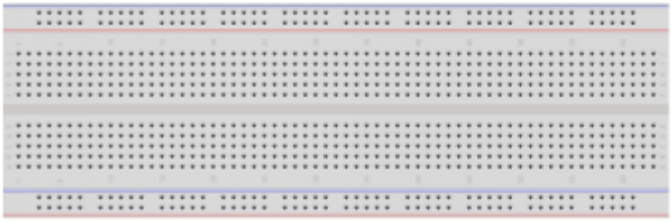
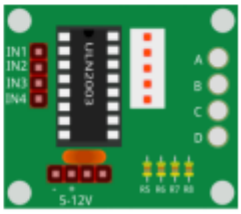


1.10.6 3.1.6 モーション制御

前書き

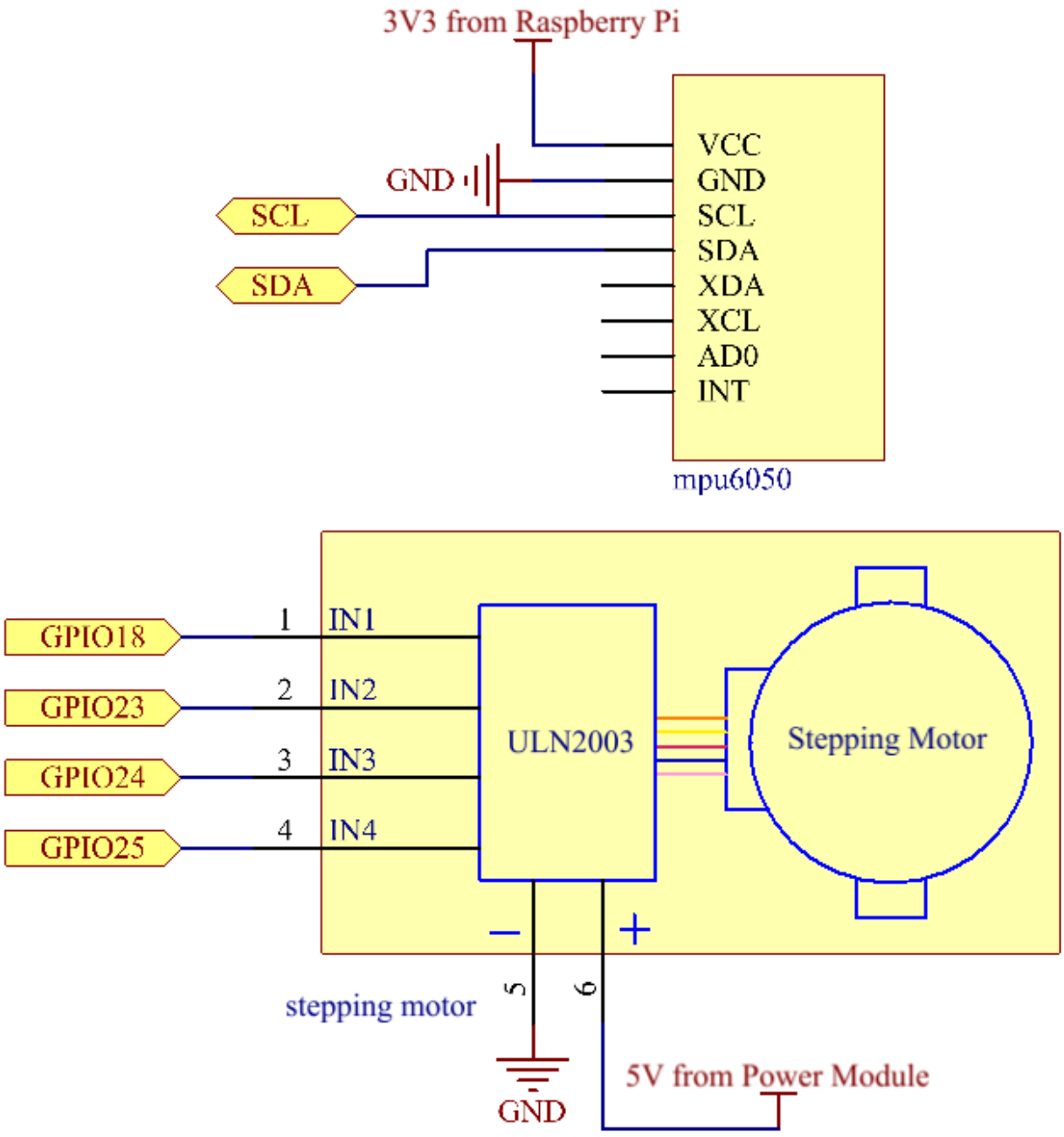
このレッスンでは、簡単なモーション検知と制御装置を作成する。MPU6050 はセンサーとして使用され、ステッピングモーターは制御装置として使用される。MPU6050 をグローブに取り付けた状態で、手首を回すことでステッピングモーターを制御できる。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * Stepper Motor 
1 * 40-pin Cable		Several Jumper Wires
		1 * MPU6050 Module 
1 * Breadboard 		1 * ULN2003 Module 

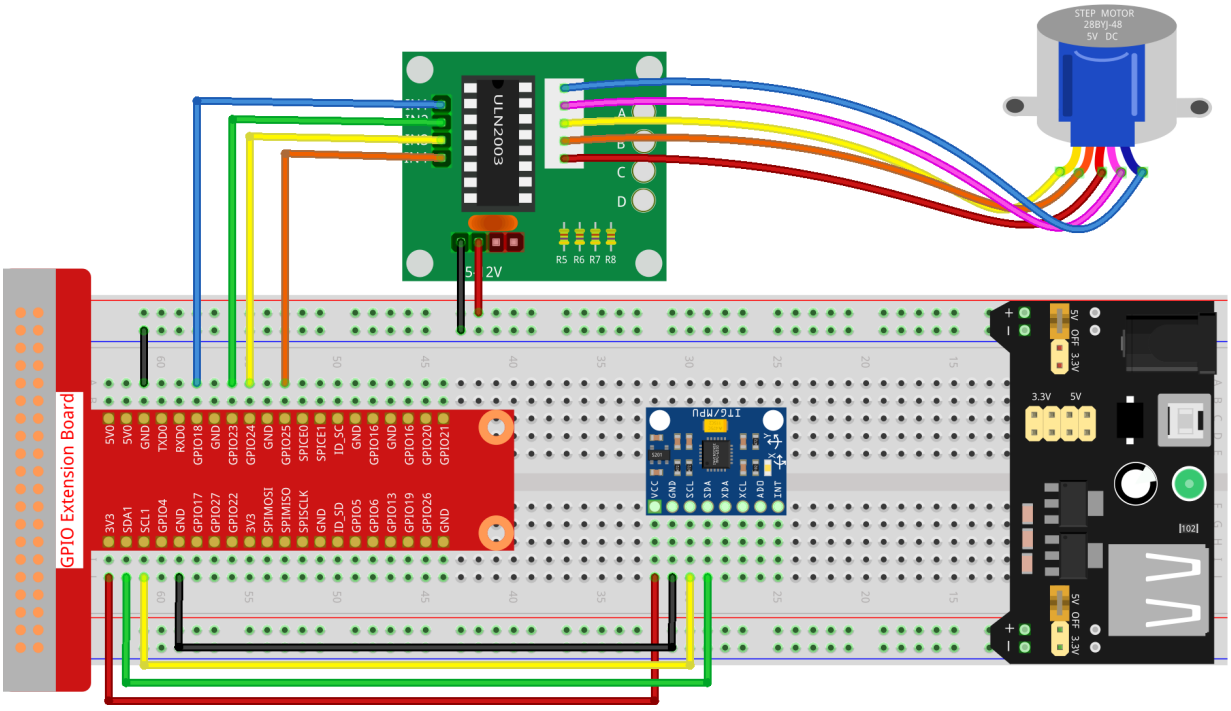
回路図

T-Board Name	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： 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/3.1.6/
```

ステップ 3： コードをコンパイルする。

```
gcc 3.1.6_MotionControl.c -lwiringPi -lm
```

ステップ 4： EXE ファイルを実行する。

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

コードが実行されている間、もし mpu6050 のチルト角度が Y-axis 45 ℃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します。-45 ℃未満の場合、ステッピングモーターは時計回りに回転します。

コードの説明

```
double^^c2^^a0mpu6050(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclX^^c2^^a0=^^c2^^a0read_word_2c(0x3B);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclY^^c2^^a0=^^c2^^a0read_word_2c(0x3D);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclZ^^c2^^a0=^^c2^^a0read_word_2c(0x3F);
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclX_scaled^^c2^^a0=^^c2^^a0acclX^^c2^^a0/^^c2^^a016384.
↪0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclY_scaled^^c2^^a0=^^c2^^a0acclY^^c2^^a0/^^c2^^a016384.
↪0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0acclZ_scaled^^c2^^a0=^^c2^^a0acclZ^^c2^^a0/^^c2^^a016384.
↪0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0double^^c2^^a0angle=get_y_rotation(acclX_scaled,
↪^^c2^^a0acclY_scaled,^^c2^^a0acclZ_scaled);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0angle;
}
```

mpu6050 は Y 軸の方向の傾斜角を取得する。

```
void^^c2^^a0rotary(char^^c2^^a0direction){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(direction^^c2^^a0==^^c2^^a0'c'){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0j=0;j<4;
↪j++){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int
↪i<4;i++)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
```

を参照してください。

```
int^^c2^^a0main()
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0double^^c2^^a0angle;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1)^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0angle^^c2^^a0=^^c2^^a0mpu6050();
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(angle^^c2^^a0>
↪=45){rotary('a');}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if^^c2^^a0(angle
↪<=-45){rotary('c');}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

Y 軸方向の傾斜角は mpu6050 から読み取られ、45 ℃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 ℃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 ℃ 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 ℃ 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

(次のページに続く)

(前のページからの続き)

```
import math
import time



# Power management registers
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

bus = smbus.SMBus(1)
address = 0x68
bus.write_byte_data(address, power_mgmt_1, 0)

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

#mpu6050
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))
```

(次のページに続く)

(前のページからの続き)

```
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)

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

#Stepper Motor
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)


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

(次のページに続く)

(前のページからの続き)

```
def loop():
    while True:
        angle=mpu6050()
        if angle >=45 :
            rotary('a')
        elif angle <=-45:
            rotary('c')

def destroy():
    GPIO.cleanup()

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

コードの説明

```
def^^c2^^a0mpu6050():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_xout^^c2^^a0=^^c2^^a0read_word_2c(0x3b)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_yout^^c2^^a0=^^c2^^a0read_word_2c(0x3d)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_zout^^c2^^a0=^^c2^^a0read_word_2c(0x3f)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_xout_scaled^^c2^^a0=^^c2^^a0accel_xout^^c2^^a0/
↪^^c2^^a016384.0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_yout_scaled^^c2^^a0=^^c2^^a0accel_yout^^c2^^a0/
↪^^c2^^a016384.0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0accel_zout_scaled^^c2^^a0=^^c2^^a0accel_zout^^c2^^a0/
↪^^c2^^a016384.0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0angle=get_y_rotation(accel_xout_scaled,^^c2^^a0accel_
↪yout_scaled,^^c2^^a0accel_zout_scaled)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0angle
```

mpu6050 は Y 軸の方向の傾斜角を取得する。

```
def^^c2^^a0rotary(direction):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(direction^^c2^^a0==^^c2^^a0'c'):
```

(次のページに続く)

(前のページからの続き)

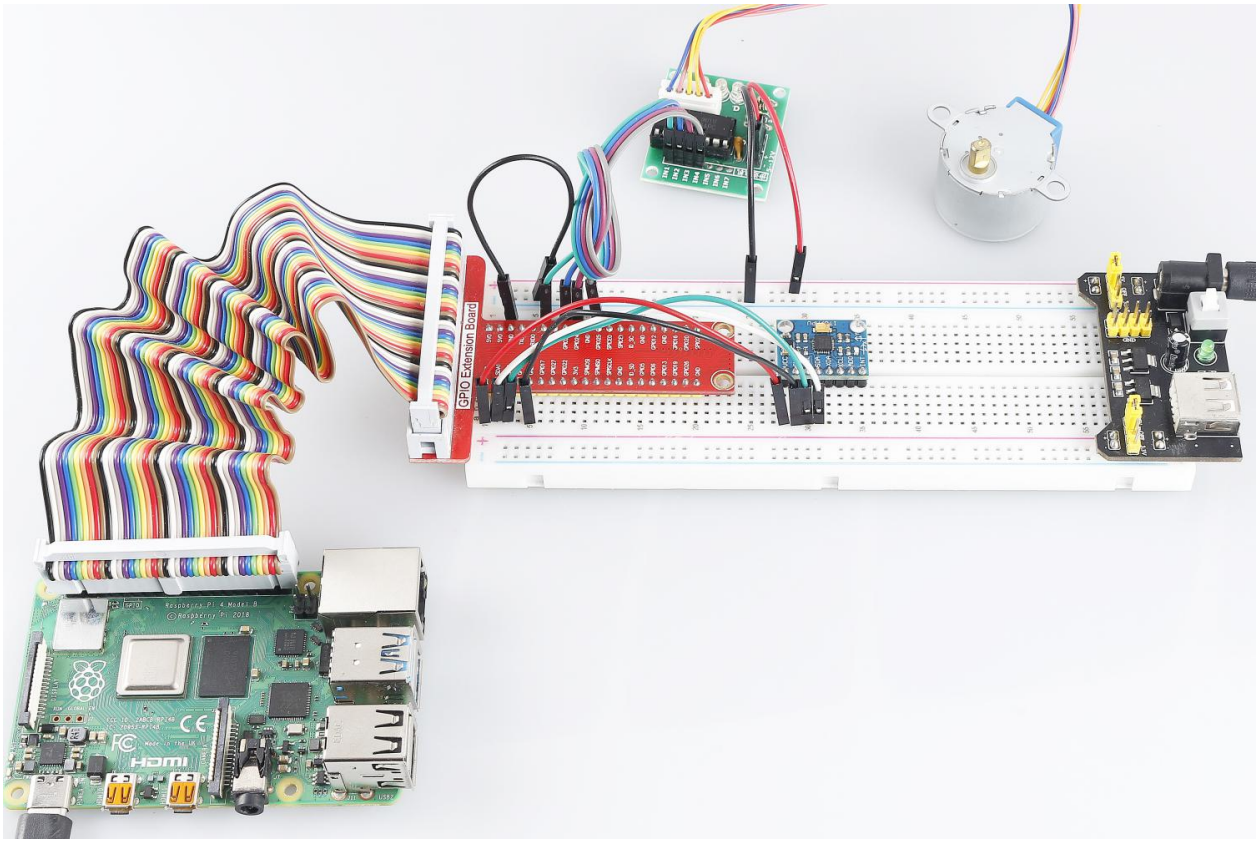
```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0j^^c2^^a0in^^c2^^a0range(4):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪output(motorPin[i],0x99>>j^^c2^^a0&^^c2^^a0(0x08>>i))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.
↪sleep(stepSpeed)

^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif(direction^^c2^^a0==^^c2^^a0'a'):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0j^^c2^^a0in^^c2^^a0range(4):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪output(motorPin[i],0x99<<j^^c2^^a0&^^c2^^a0(0x80>>i))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.
↪sleep(stepSpeed)
```

受信方向 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します。キー が「a 」の場合、モーターは反時計回りに回転します。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* **ステッピングモーター** を参照してください。

```
def^^c2^^a0loop():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0angle=mpu6050()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0angle^^c2^^a0>
↪=45^^c2^^a0:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0rotary(
↪'a')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0angle^^c2^^a0
↪<=-45:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0rotary(
↪'c')
```

mpu6050 から Y 軸方向の傾斜角を読み取り、45 ° C より大きい場合は、 rotary() を呼び出して、ステッピングモーターを反時計回りに回転させます。-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します。

現象画像

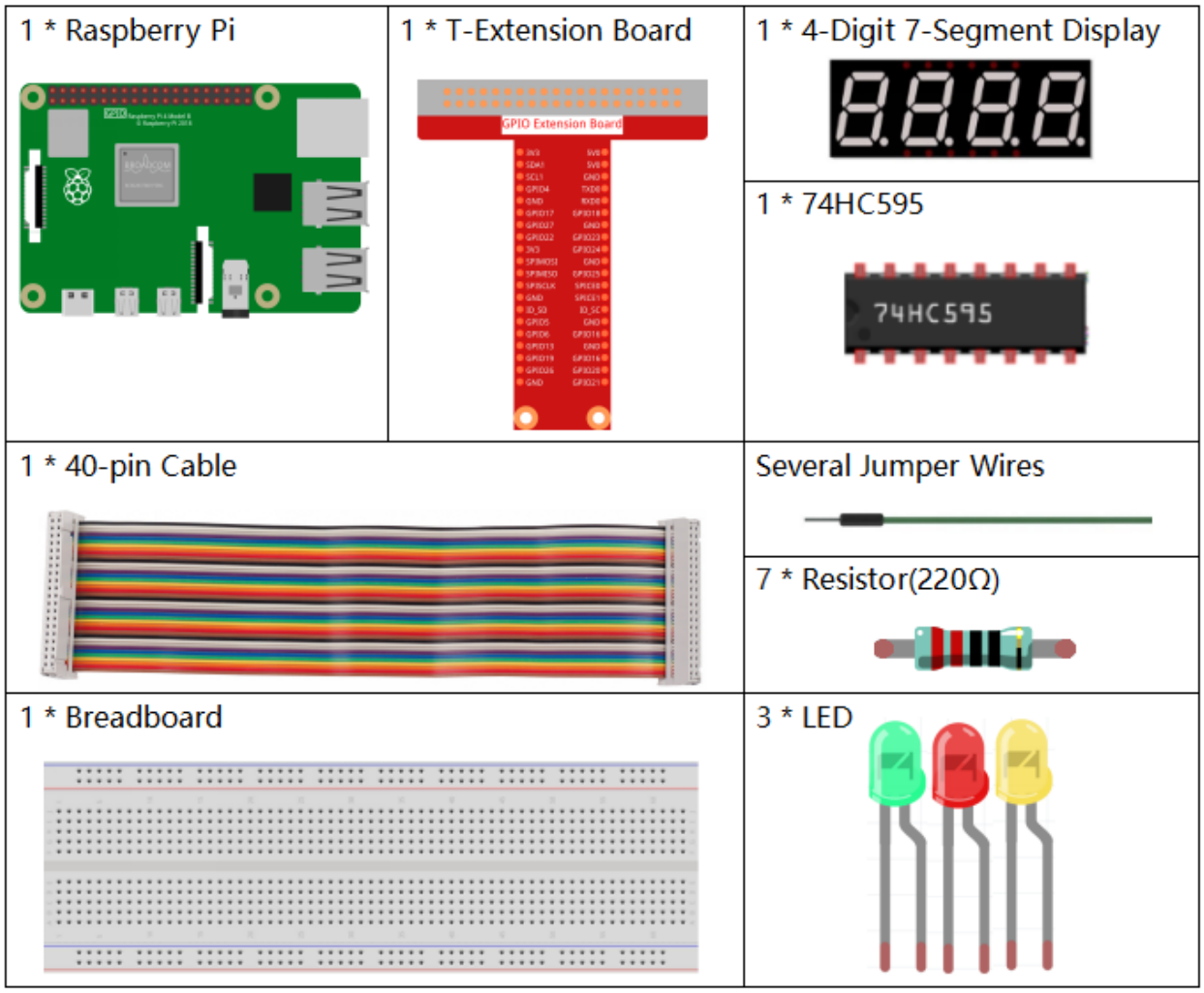


1.10.7 3.1.7 信号機

前書き

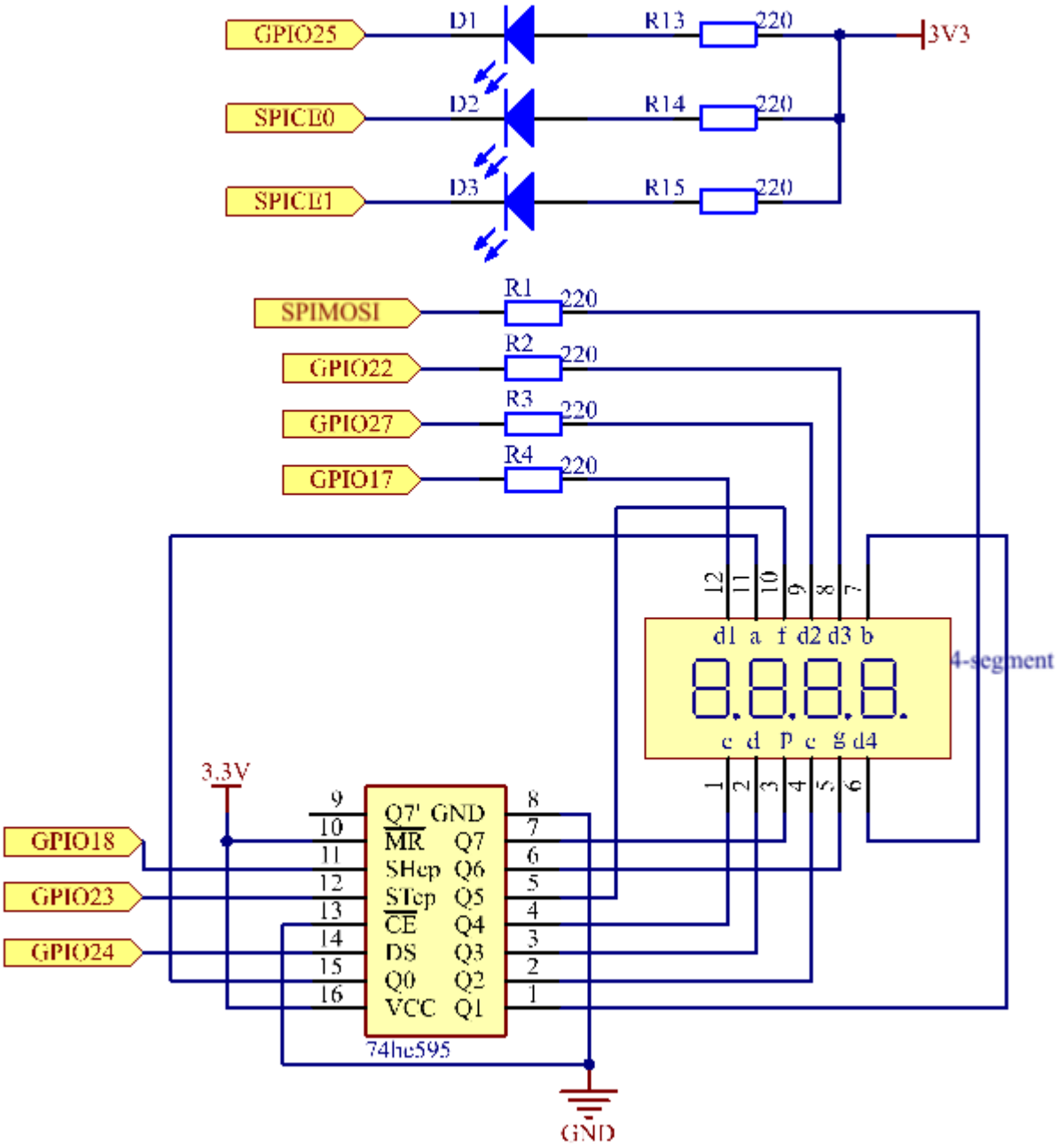
このプロジェクトでは、3 色の LED を使用して交通信号の変化を実現し、4 桁の 7 セグメントディスプレイを使用して各交通状態のタイミングを表示する。

部品



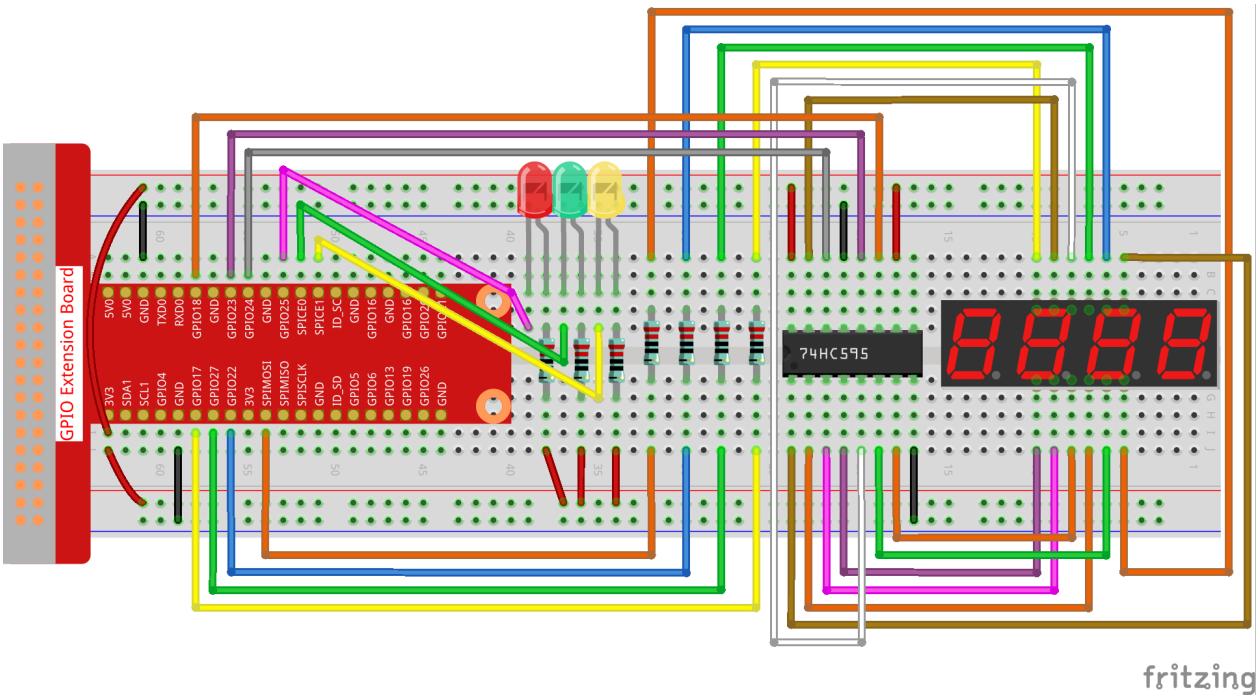
回路図

T-Board Name	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
```

(次のページに続く)

(前のページからの続き)

```
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 桁 7 セグメントディスプレイの数値表示機能を実現するために使用されます。詳細については、ドキュメントの *1.1.5 4 桁 7* **セグメントディスプレイ** を参照してください。ここでは、コードを使用して信号時間のカウントダウンを表示します。

```
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;
```

(次のページに続く)

(前のページからの続き)

```
            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ます。 詳細については、 *1.1.5 4 桁 7* **セグメントディスプレイ** を参照してください。 ここで、タイマーがゼロに戻ると、 colorState が切り替えられて LED が切り替わり、タイマーが新しい値に割り当てられます。

```
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 桁 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 桁の 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

#define the pins connect to 74HC595
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 = (10,22,27,17)
ledPin =(25,8,7)

greenLight = 30
yellowLight = 5
redLight = 60
lightColor=("Red","Green","Yellow")

colorState=0
counter = 60
timer1 = 0


def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
```

(次のページに続く)

(前のページからの続き)

```
    GPIO.setup(SRCLK, GPIO.OUT)
    for pin in placePin:
        GPIO.setup(pin,GPIO.OUT)
    for pin in ledPin:
        GPIO.setup(pin,GPIO.OUT)
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()

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():        #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):
```

(次のページに続く)

(前のページからの続き)

```
            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]))

def lightup():
    global colorState
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.HIGH)
    GPIO.output(ledPin[colorState], GPIO.LOW)

def display():
    global counter

    a = counter % 10000//1000 + counter % 1000//100
    b = counter % 10000//1000 + counter % 1000//100 + counter % 100//10
    c = counter % 10000//1000 + counter % 1000//100 + counter % 100//10 + counter % 10

    if (counter % 10000//1000 == 0):
        clearDisplay()
    else:
        clearDisplay()
        pickDigit(3)
        hc595_shift(number[counter % 10000//1000])

    if (a == 0):
        clearDisplay()
    else:
        clearDisplay()
        pickDigit(2)
        hc595_shift(number[counter % 1000//100])

    if (b == 0):
        clearDisplay()
    else:
        clearDisplay()
```

(次のページに続く)

(前のページからの続き)

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

    if(c == 0):
        clearDisplay()
    else:
        clearDisplay()
        pickDigit(0)
        hc595_shift(number[counter % 10])

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()
```

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 = (10,22,27,17)

def clearDisplay():
def hc595_shift(data):
def pickDigit(digit):
def display():
```

これらのコードは、4 桁 7 セグメントの数値表示機能を実現するために使用されます。詳細については、ドキュメントの *1.1.5 4 桁 7 セグメントディスプレイ* を参照してください。ここでは、コードを使用して信号時間のカウントダウンを表示します。

```
ledPin =(25,8,7)
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。

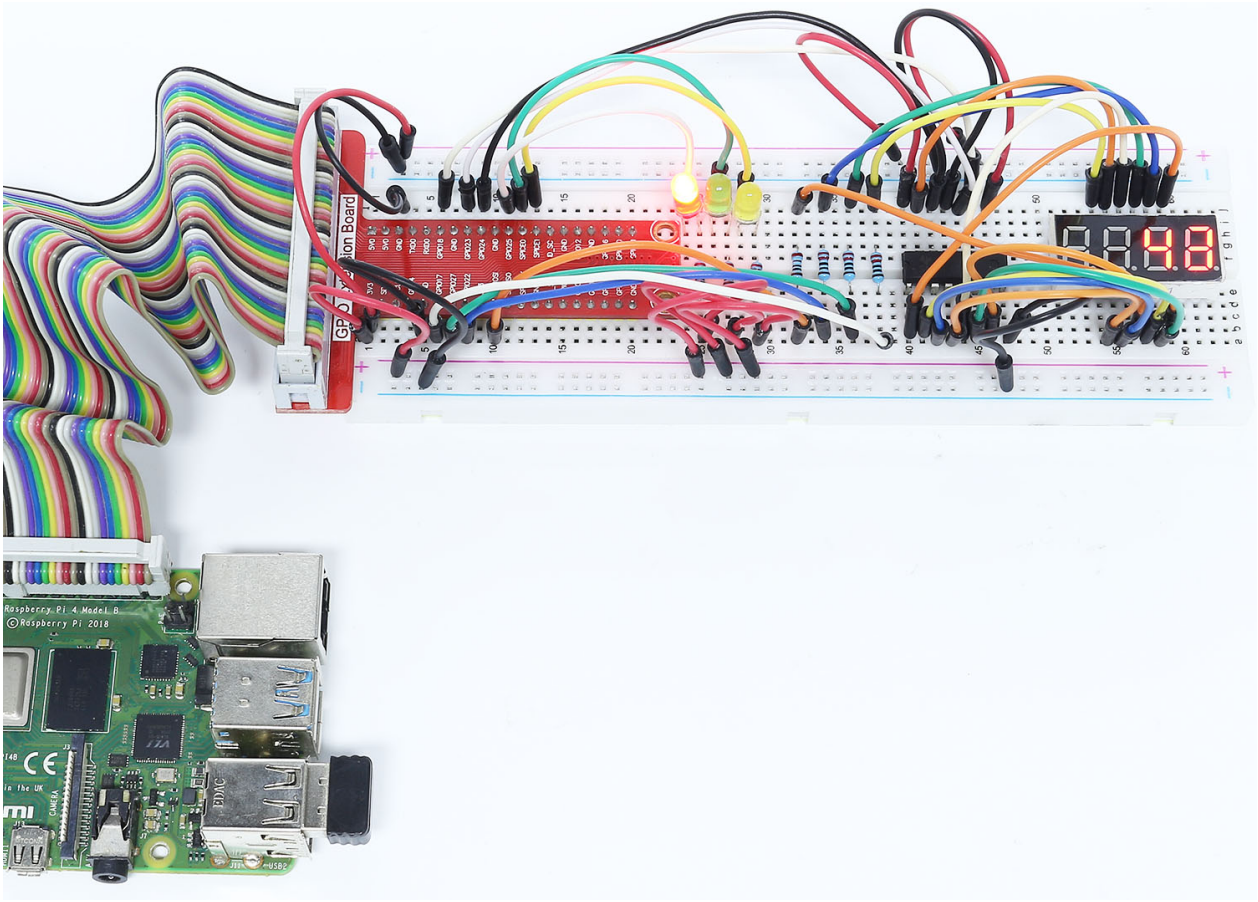
```
greenLight = 30
yellowLight = 5
redLight = 60
lightColor=("Red","Green","Yellow")

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ます。詳細については、 *1.1.5 4* **桁** *7* **セグメントディスプレイ** を参照してください。ここで、タイマーがゼロに戻ると、colorState が切り替えられて LED が切り替わり、タイマーが新しい値に割り当てられます。

```
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-桁の 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

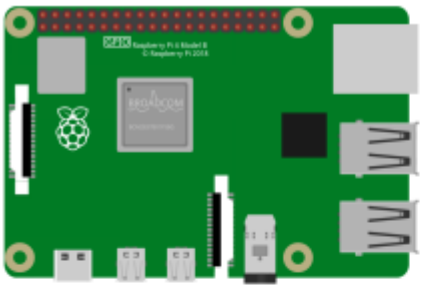
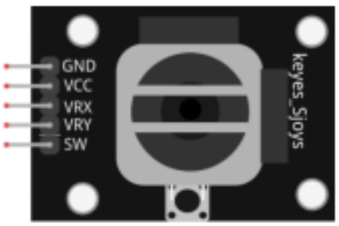
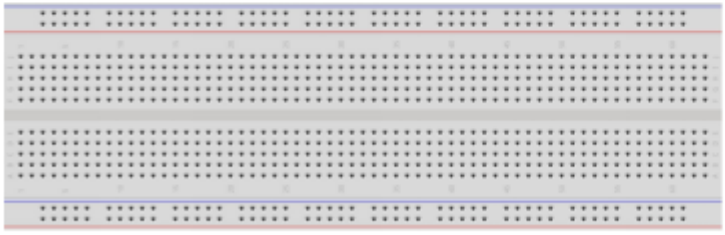
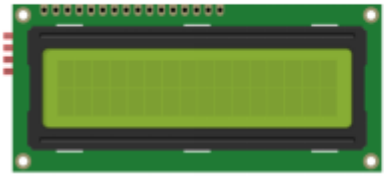


1.10.8 3.1.8 過熱モニター

前書き

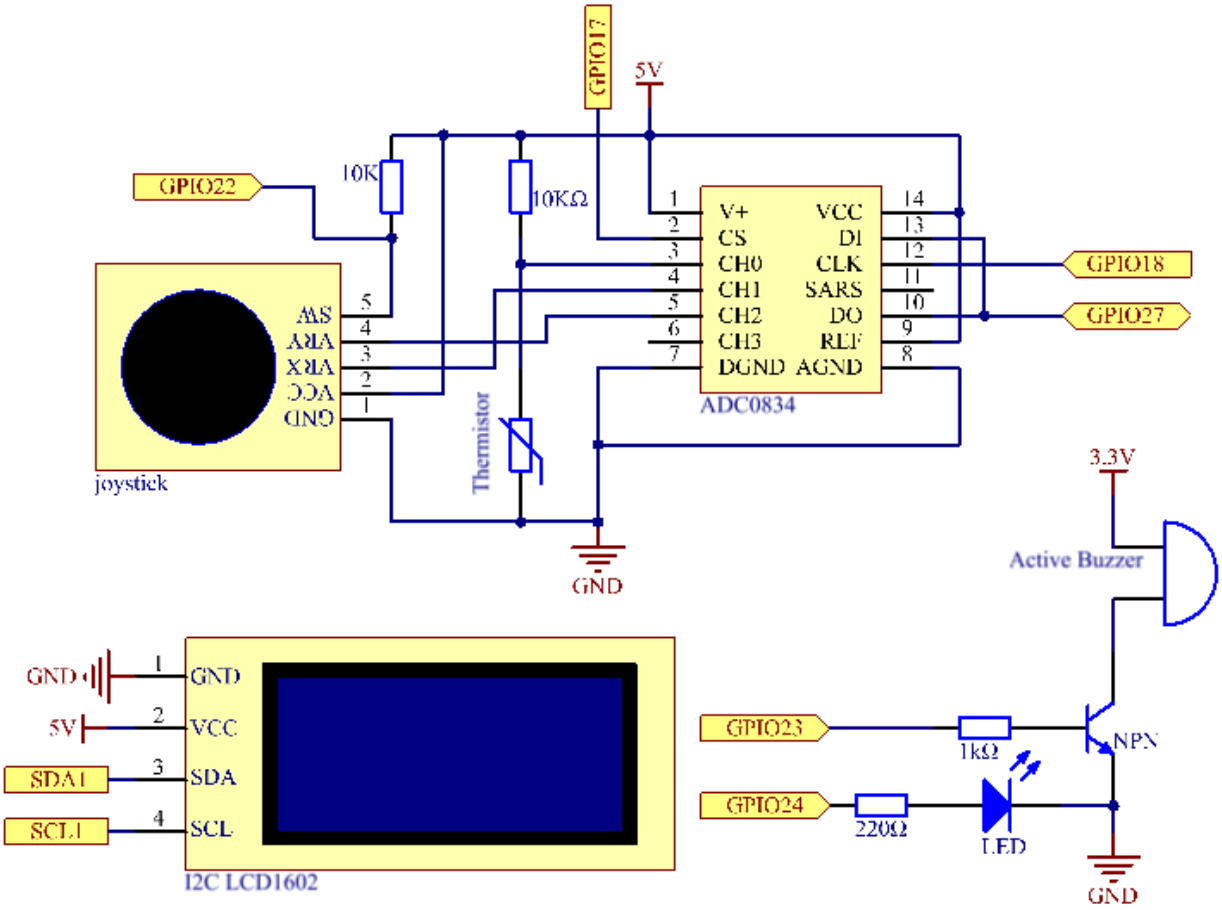
回路の過熱が発生したときに警報装置とタイムリーな自動電源切断を希望する場合は、工場などのさまざまな状況に適用される過熱監視装置を作成することができる。このレッスンでは、サーミスタ、ジョイスティック、ブザー、LED、と LCD を使用して、しきい値が調整可能なスマートな温度監視装置を作成する。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * Joystick 
1 * 40-pin Cable		1 * ADC0834
1 * Breadboard 		1 * LED
		1 * S8050 NPN Transistor
1 * Resistor(220Ω)	1 * Resistor 1KΩ	2 * Resistor 10KΩ
1 * I2C LCD1602 	1 * Active Buzzer	Several Jumper Wires

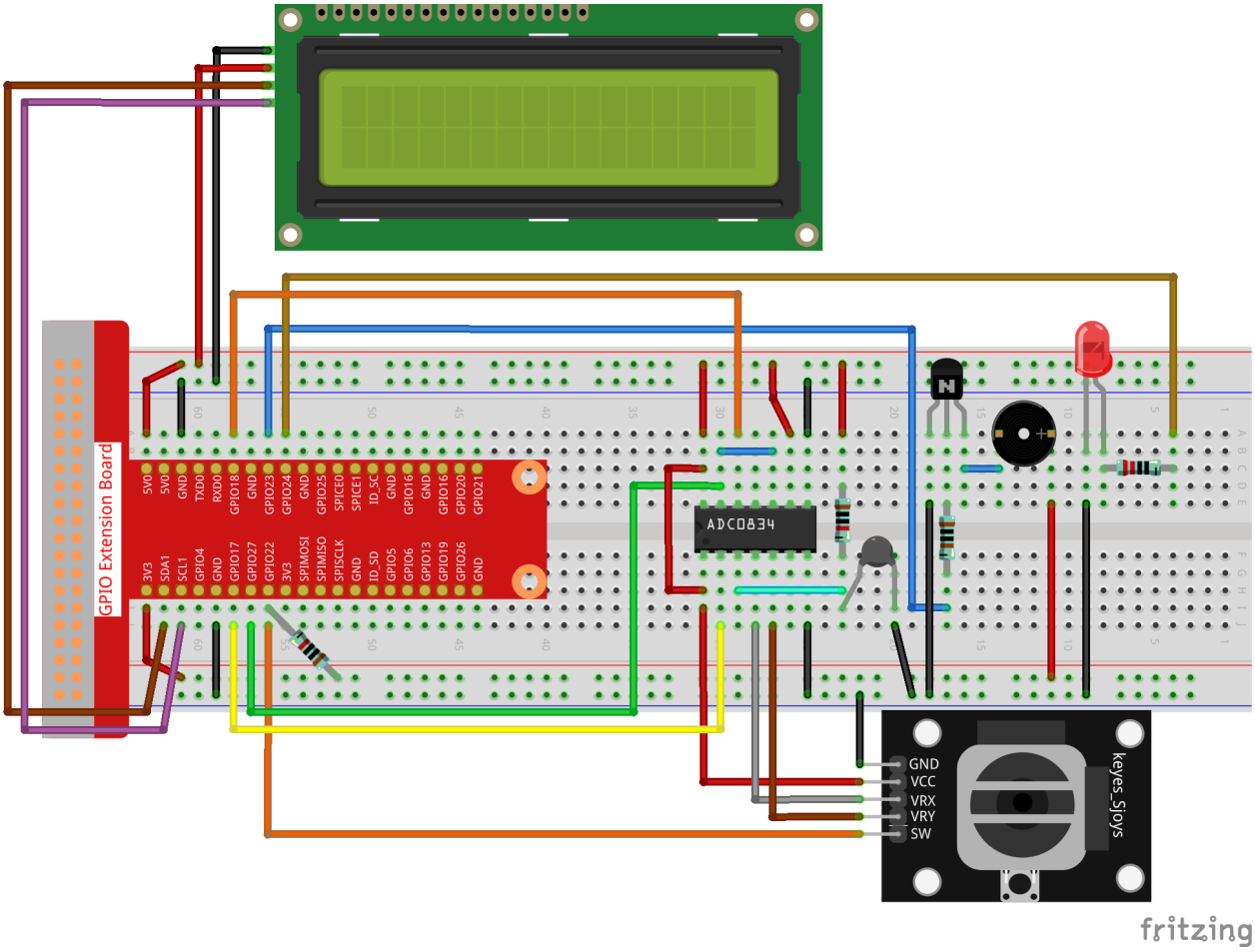
回路図

T-Board Name	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 -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^^c2^^a0get_joystick_value(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0uchar^^c2^^a0x_val;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0uchar^^c2^^a0y_val;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0x_val^^c2^^a0=^^c2^^a0get_ADC_Result(1);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0y_val^^c2^^a0=^^c2^^a0get_ADC_Result(2);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(x_val^^c2^^a0>^^c2^^a0200){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(x_val^^c2^^a0<^^c2^^a050){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0-1;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(y_val^^c2^^a0>^^c2^^a0200){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0-10;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(y_val^^c2^^a0<^^c2^^a050){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a010;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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^^c2^^a0upper_tem_setting(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,^^c2^^a00,^^c2^^a0"Upper^^c2^^a0Adjust:");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0change^^c2^^a0=^^c2^^a0get_joystick_value();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upperTem^^c2^^a0=^^c2^^a0upperTem^^c2^^a0+^^c2^^a0change;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0str[6];
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0snprintf(str,3,"%d",upperTem);
write(0,1,str);
int^^c2^^a0len;
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0len^^c2^^a0=^^c2^^a0strlen(str);
^^c2^^a0^^c2^^a0write(len,1,
↪"^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^
↪");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(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^^c2^^a0temperature(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0temp_value;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0double^^c2^^a0Vr,^^c2^^a0Rt,^^c2^^a0temp,^^c2^^a0cel,
↪^^c2^^a0Fah;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0temp_value^^c2^^a0=^^c2^^a0get_ADC_Result(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Vr^^c2^^a0=^^c2^^a05^^c2^^a0*^^c2^^a0(double)(temp_
↪value)^^c2^^a0/^^c2^^a0255;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Rt^^c2^^a0=^^c2^^a010000^^c2^^a0*^^c2^^a0(double)(Vr)^^c2^^a0/
↪^^c2^^a0(5^^c2^^a0-^^c2^^a0(double)(Vr));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0temp^^c2^^a0=^^c2^^a01^^c2^^a0/^^c2^^a0(((log(Rt/
↪10000))^^c2^^a0/^^c2^^a03950)+(1^^c2^^a0/^^c2^^a0(273.15^^c2^^a0+^^c2^^a025)));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0cel^^c2^^a0=^^c2^^a0temp^^c2^^a0-^^c2^^a0273.15;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Fah^^c2^^a0=^^c2^^a0cel^^c2^^a0*^^c2^^a01.8^^c2^^a0+32;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0cel;
}
```

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^^c2^^a0monitoring_temp(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0str[6];
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0double^^c2^^a0cel^^c2^^a0=^^c2^^a0temperature();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0snprintf(str,6,"%.2f",cel);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,^^c2^^a00,^^c2^^a0"Temp:^^c2^^a0");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(6,^^c2^^a00,^^c2^^a0str);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0snprintf(str,3,"%d",upperTem);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,^^c2^^a01,^^c2^^a0"Upper:^^c2^^a0");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(7,^^c2^^a01,^^c2^^a0str);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(100);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(cel^^c2^^a0>=^^c2^^a0upperTem){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(buzzPin,
```

(次のページに続く)

(前のページからの続き)

```
↪^^c2^^a0HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(LedPin,
↪^^c2^^a0HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(cel^^c2^^a0<^^c2^^a0upperTem){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(buzzPin,
↪^^c2^^a0LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(LedPin,
↪^^c2^^a0LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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^^c2^^a0main(void)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0lastState^^c2^^a0=1;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0stage=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentState^^c2^^a0=^^c2^^a0
↪BtnPin);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(currentState==1^^c2^^a0&
↪&^^c2^^a0lastState^^c2^^a0==^^c2^^a00){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage=(s
↪%2;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(10
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_
↪clear();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(stage==1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper_
↪tem_setting();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else{
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0monitori
↪temp();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

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：

- 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。
- 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import LCD1602
import RPi.GPIO as GPIO
```

(次のページに続く)

(前のページからの続き)

```
import ADC0834
import time
import math

Joy_BtnPin = 22
buzzPin = 23
ledPin = 24



upperTem = 40

def setup():
    ADC0834.setup()
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(ledPin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(buzzPin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Joy_BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    LCD1602.init(0x27, 1)

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

def upper_tem_setting():
    global upperTem
    LCD1602.write(0, 0, 'Upper Adjust: ')
    change = int(get_joystick_value())
    upperTem = upperTem + change
    strUpperTem = str(upperTem)
```

(次のページに続く)

(前のページからの続き)

```
    LCD1602.write(0, 1, strUpperTem)
    LCD1602.write(len(strUpperTem),1, '               ')
    time.sleep(0.1)

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)

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)

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:
```

(次のページに続く)

(前のページからの続き)

```
            upper_tem_setting()
        else:
            monitoring_temp()

def destroy():
    LCD1602.clear()
    ADC0834.destroy()
    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^^c2^^a0get_joystick_value():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0x_val^^c2^^a0=^^c2^^a0ADC0834.getResult(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0y_val^^c2^^a0=^^c2^^a0ADC0834.getResult(2)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(x_val^^c2^^a0>^^c2^^a0200):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif(x_val^^c2^^a0<^^c2^^a050):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0-1
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif(y_val^^c2^^a0>^^c2^^a0200):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0-10
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif(y_val^^c2^^a0<^^c2^^a050):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a010
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00
```

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^^c2^^a0upper_tem_setting():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0upperTem
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,^^c2^^a00,^^c2^^a0
↪'Upper^^c2^^a0Adjust:^^c2^^a0')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0change^^c2^^a0=^^c2^^a0int(get_joystick_value())
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upperTem^^c2^^a0=^^c2^^a0upperTem^^c2^^a0+^^c2^^a0change
LCD1602.write(0,^^c2^^a01,^^c2^^a0str(upperTem))
LCD1602.write(len(strUpperTem),1, '               ')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.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^^c2^^a0temperature():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0analogVal^^c2^^a0=^^c2^^a0ADC0834.getResult()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Vr^^c2^^a0=^^c2^^a05^^c2^^a0*^^c2^^a0float(analogVal)^^c2^^a0/
↪^^c2^^a0255
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Rt^^c2^^a0=^^c2^^a010000^^c2^^a0*^^c2^^a0Vr^^c2^^a0/
↪^^c2^^a0(5^^c2^^a0-^^c2^^a0Vr)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0temp^^c2^^a0=^^c2^^a01/(((math.log(Rt^^c2^^a0/
↪^^c2^^a010000))^^c2^^a0/^^c2^^a03950)^^c2^^a0+^^c2^^a0(1^^c2^^a0/^^c2^^a0(273.15+25)))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Cel^^c2^^a0=^^c2^^a0temp^^c2^^a0-^^c2^^a0273.15
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Fah^^c2^^a0=^^c2^^a0Cel^^c2^^a0*^^c2^^a01.
↪8^^c2^^a0+^^c2^^a032
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0round(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^^c2^^a0monitoring_temp():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0upperTem
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Cel=temperature()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,^^c2^^a00,^^c2^^a0'Temp:^^c2^^a0')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,^^c2^^a01,^^c2^^a0'Upper:^^c2^^a0')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(6,^^c2^^a00,^^c2^^a0str(Cel))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(7,^^c2^^a01,^^c2^^a0str(upperTem))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(0.1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0Cel^^c2^^a0>=^^c2^^a0upperTem:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(buzzPin,
↪^^c2^^a0GPIO.HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ledPin,
↪^^c2^^a0GPIO.HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else:
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(buzzPin,
↪^^c2^^a0GPIO.LOW)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ledPin,
↪^^c2^^a0GPIO.LOW)
```

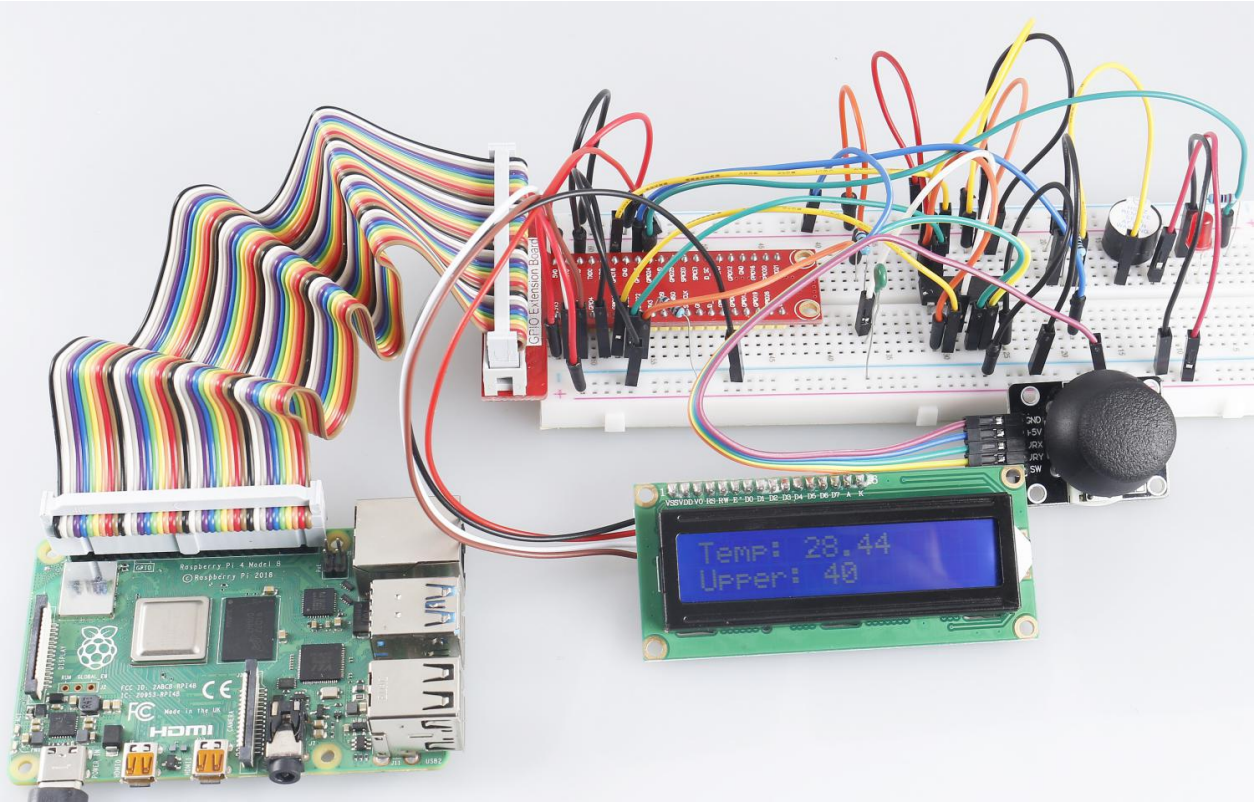
コードが実行されると、現在の温度と高温のしきい値 40 が I2C LCD1602 に表示される。現在の温度がしきい値よりも大きい場合、ブザーと LED が起動して警告を発する。

```
def^^c2^^a0loop():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=1
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage=0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=GPIO.
↪input(Joy_BtnPin)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0currentState==1^^c2^^a0and^^c2
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage=(s
↪%2
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.
↪sleep(0.1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602
↪clear()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0stage^^c2^^a0==^^c2^^a01:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper_
↪tem_setting()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0monitori
↪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る：

- 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。
- 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



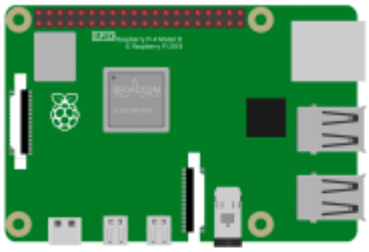
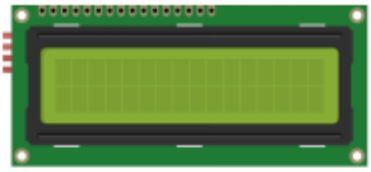
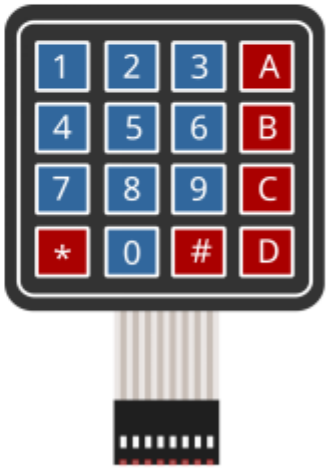
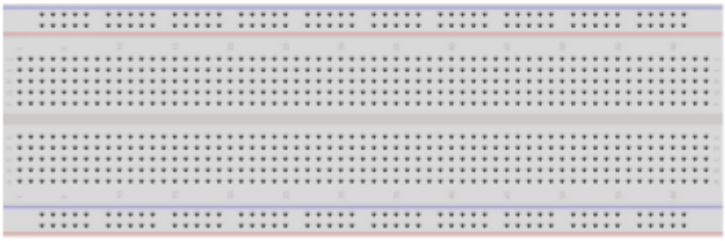
1.10.9 3.1.9 パスワードロック

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。

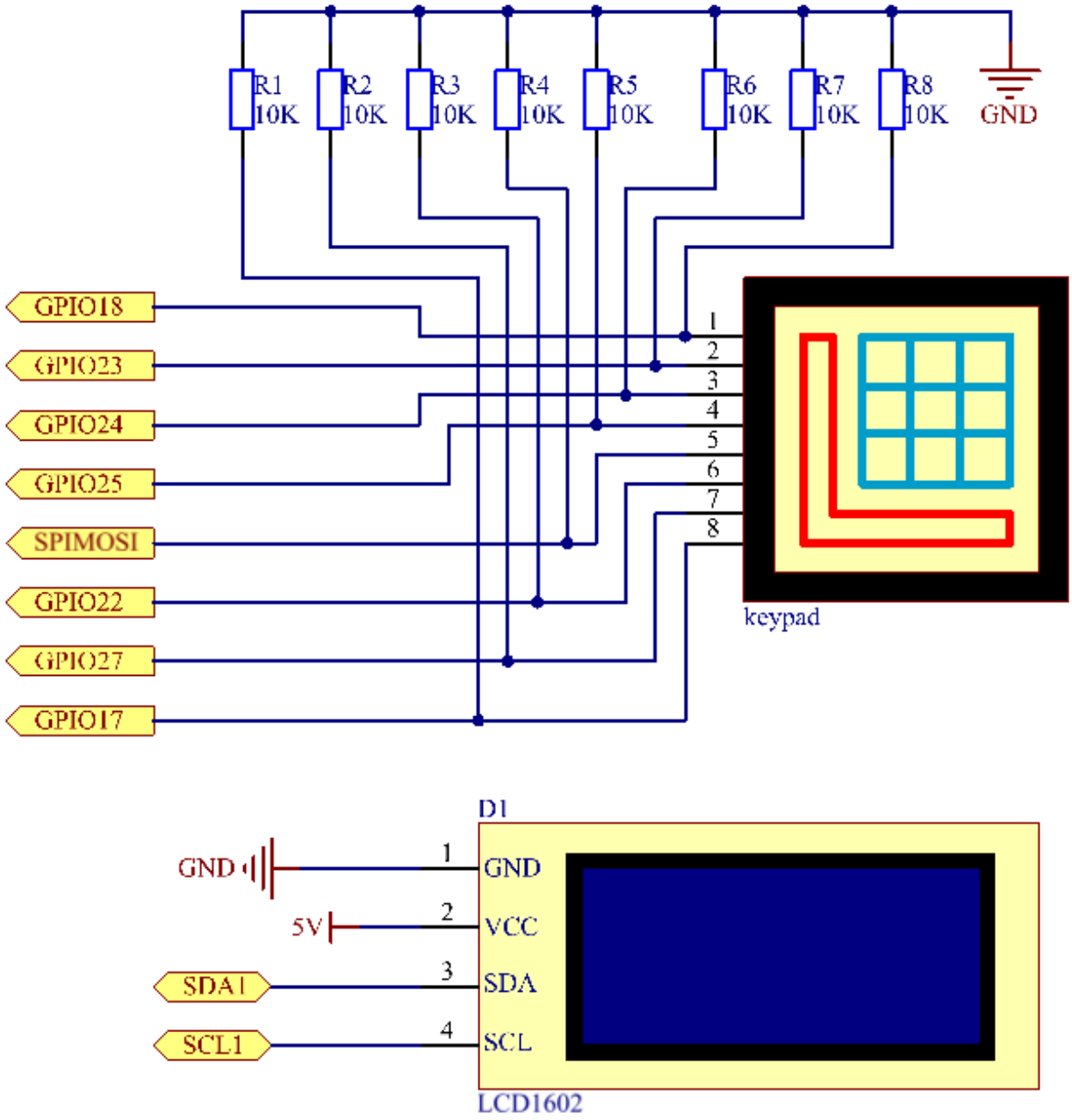
このプロジェクトに基づいて、ブザー、LED などの電子部品を追加して、パスワード入力にさまざまな実験現象を追加できる。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * I2C LCD1602 
		Several Jumper Wires
1 * 40-pin Cable		1 * Keypad 
1 * Breadboard 		
		8 * Resistor 10KΩ

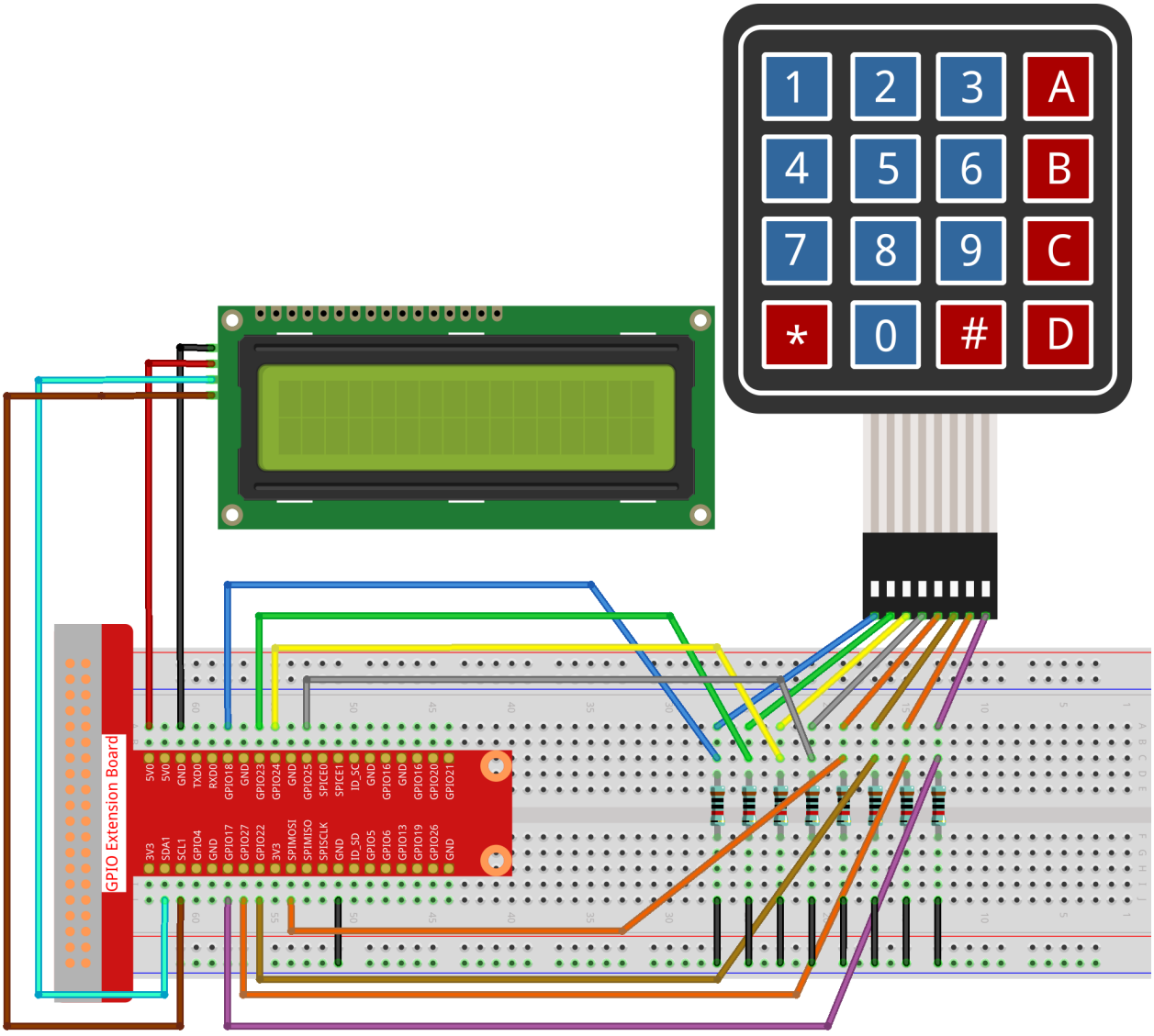
回路図

T-Board Name	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
SPIMOSI	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機能の宣言があります。詳細については、このドキュメントの *2.1.5* **キーパッド** 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 コードのサブ機能の宣言があります。参照してください。詳細については、このドキュメントの *1.1.7*

I2C LCD1602 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。

```
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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time
import LCD1602

###################### HERE IS THE KEYPAD LIBRARY TRANSPLANTED FROM Arduino ############
#class Key:Define some of the properties of Key
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

################ EXAMPLE CODE START HERE ################
LENS = 4
password=['1','9','8','4']
testword=['0','0','0','0']
keyIndex=0
```

(次のページに続く)

(前のページからの続き)

```
def check():
    for i in range(0,LENS):
        if(password[i]!=testword[i]):
            return 0
    return 1

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 = []
    LCD1602.init(0x27, 1)    # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'WELCOME!')
    LCD1602.write(2, 1, 'Enter password')
    time.sleep(2)

def destroy():
    LCD1602.clear()
    GPIO.cleanup()

def loop():
    global keyIndex
    global LENS
    global keypad, last_key_pressed
    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
            if (keyIndex is LENS):
```

(次のページに続く)

(前のページからの続き)

```
                if (check() is 0):
                    LCD1602.clear()
                    LCD1602.write(3, 0, "WRONG KEY!")
                    LCD1602.write(0, 1, "please try again")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "CORRECT!")
                    LCD1602.write(2, 1, "welcome back")
            keyIndex=keyIndex%LENS

        last_key_pressed = pressed_keys
        time.sleep(0.1)

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

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
```

(次のページに続く)

(前のページからの続き)

```
        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す。詳細については、このドキュメントの *2.1.5* **キーパッド** を参照してください。

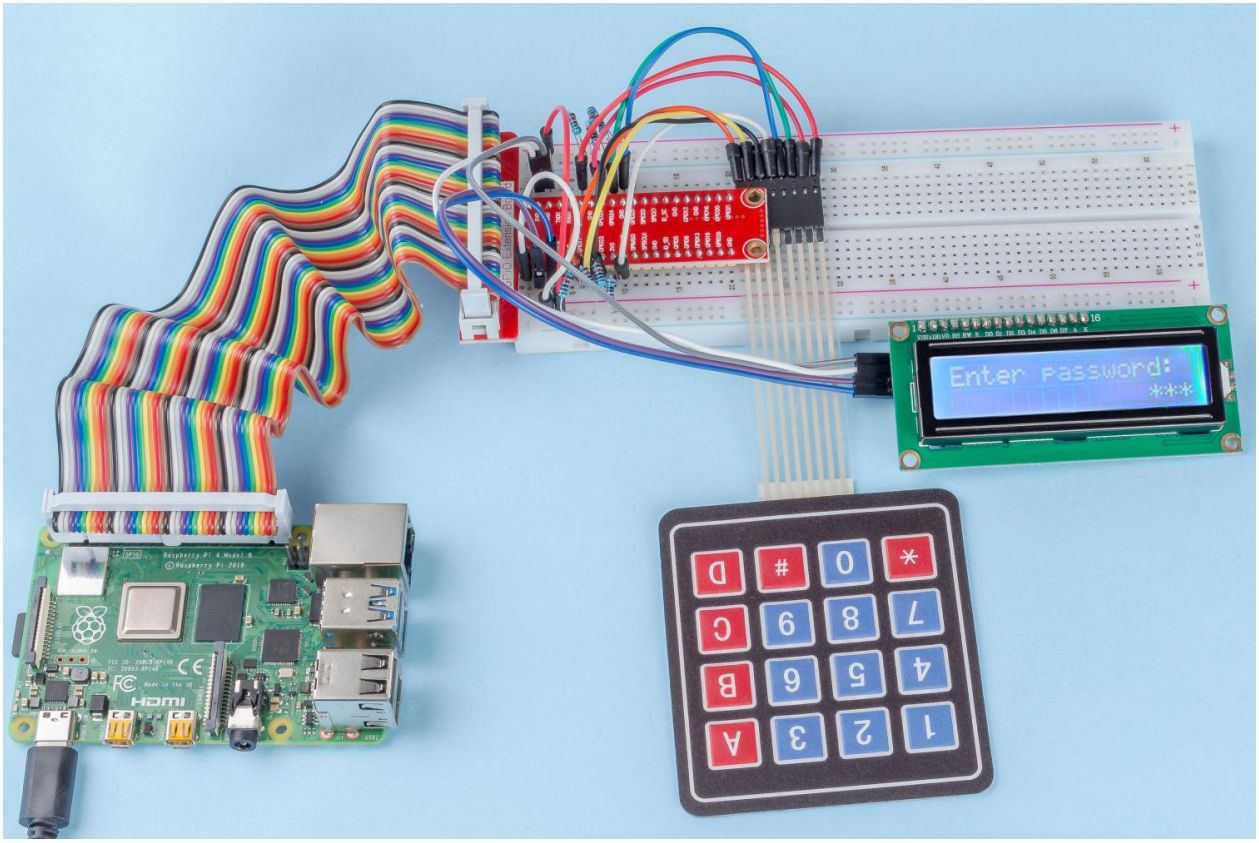
```
while(True):
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, pressed_keys)
            testword[keyIndex]=pressed_keys
            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

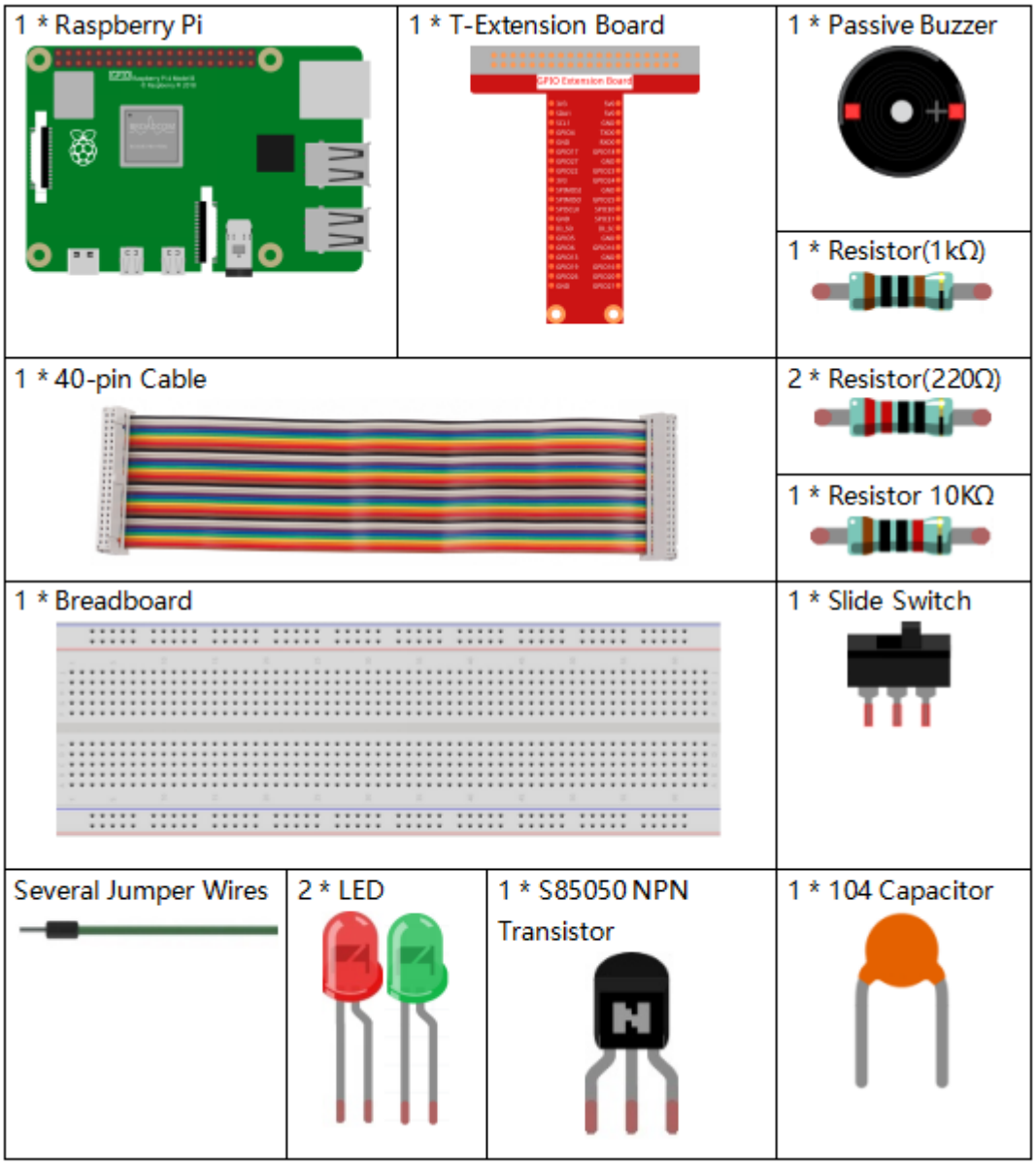


1.10.10 3.1.10 警報ベル

前書き

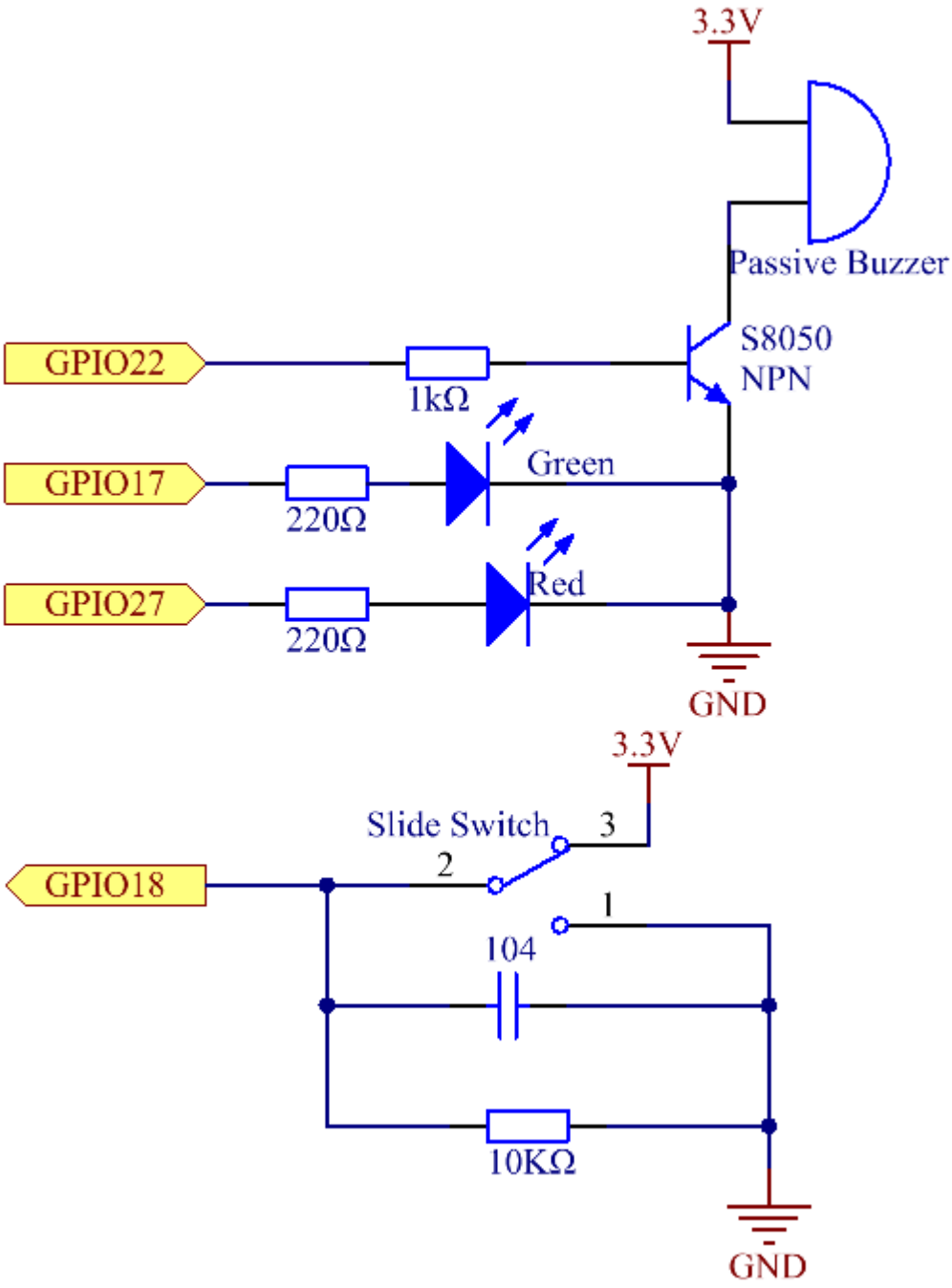
このコースでは、手動警報装置を作成する。トグルスイッチをサーミスタまたは感光センサーに交換して、温度警報または光警報を作成できる。

部品



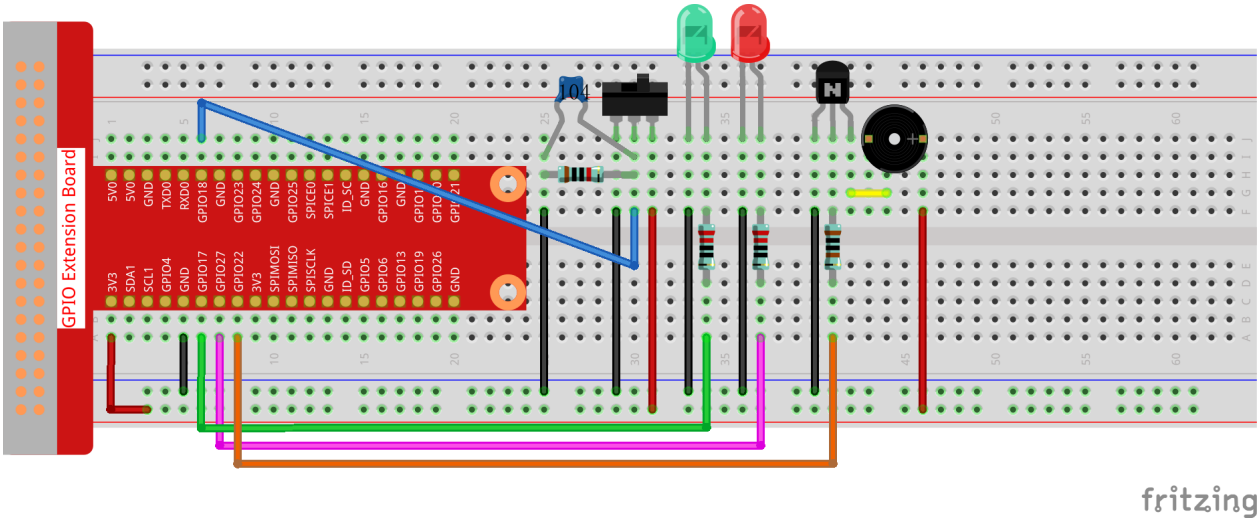
回路図

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22


3.3V
Passive Buzzer
S8050
NPN
GPIO22
1kΩ
GPIO17
Green
220Ω
GPIO27
Red
220Ω
GND
3.3V
Slide Switch
3
2
GPIO18
1
104
10KΩ
GND

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^^c2^^a0<pthread.h>
```

このコードでは、新しいライブラリ thread.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 パラメーターを追加して、LED とブザーを独立して動作させる。

```
void^^c2^^a0*ledWork(void^^c2^^a0*arg){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(flag==0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_
↪exit(NULL);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(ALedPin,
↪HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(500);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(ALedPin,
↪LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(BLedPin,
↪HIGH);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(500);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(BLedPin,
↪LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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^^c2^^a0*buzzWork(void^^c2^^a0*arg){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(flag==0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_
↪exit(NULL);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if((note>=800)||(note
↪<=130)){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pitch^^
↪pitch;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0note=note+pitch;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0softToneWrite(BeepPin,
↪note);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(10);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
```

(次のページに続く)

(前のページからの続き)

```
}
```

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^^c2^^a0on(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0flag^^c2^^a0=^^c2^^a01;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(softToneCreate(BeepPin)^^c2^^a0==^^c2^^a0-1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf(
↪"setup^^c2^^a0softTone^^c2^^a0failed^^c2^^a0!");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_t^^c2^^a0tLed;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_create(&tLed,NULL,ledWork,NULL);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_t^^c2^^a0tBuzz;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pthread_create(&tBuzz,NULL,buzzWork,NULL);
}
```

関数 on() で：

- 制御スレッドの終了を示すマーク「flag = 1」を定義する。
- ソフトウェア制御のトーンピン BeepPin を作成する。
- 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^^c2^^a0off(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0flag^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0softToneStop(BeepPin);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(ALedPin,LOW);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0digitalWrite(BLedPin,LOW);
}
```

スレッド ledWork と BuzzWork を終了するために、関数 Off() は「flag = 0」を定義しそして、ブザーと LED をオフにする。

```
int^^c2^^a0main(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0lastState^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentState^^c2^^a0=^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0((currentState^^c2^^a0==^^c2^^
↪&(lastState==0)){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0on();
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if((currentState^^c2^^a0==^^
↪&(lastState==1)){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0off();
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

BeepPin=22
ALedPin=17
BLedPin=27
switchPin=18

Buzz=0
flag =0
note=150
pitch=20

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT)
    GPIO.setup(ALedPin,GPIO.OUT,initial=GPIO.LOW)
    GPIO.setup(BLedPin,GPIO.OUT,initial=GPIO.LOW)
    GPIO.setup(switchPin,GPIO.IN)
    global Buzz
    Buzz=GPIO.PWM(BeepPin,note)
```

(次のページに続く)

(前のページからの続き)

```
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)

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)


def on():
    global flag
    flag = 1
    Buzz.start(50)
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()

def off():
    global flag
    flag = 0
    Buzz.stop()
    GPIO.output(ALedPin,GPIO.LOW)
    GPIO.output(BLedPin,GPIO.LOW)


def main():
```

(次のページに続く)

(前のページからの続き)

```
    lastState=0
    while True:
        currentState =GPIO.input(switchPin)
        if currentState == 1 and lastState == 0:
            on()
        elif currentState == 0 and lastState == 1:
            off()
        lastState=currentState


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


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

コードの説明

```
import^^c2^^a0threading
```

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^^c2^^a0ledWork():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0flag:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ALedPin,GPIO.
↪HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(0.5)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ALedPin,GPIO.
↪LOW)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(BLedPin,GPIO.
↪HIGH)
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(0.5)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(BLedPin,GPIO.
↪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。

```
def^^c2^^a0buzzerWork():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0pitch
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0note
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0flag:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0note^^c2^^a0>
↪=^^c2^^a0800^^c2^^a0or^^c2^^a0note^^c2^^a0<=130:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pitch^^c
↪pitch
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0note^^c2^^a0=^^c2^^a0note^^c2^^a0+^^c2^^
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Buzz.
↪ChangeFrequency(note)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.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^^c2^^a0on():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0flag
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0flag^^c2^^a0=^^c2^^a01
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Buzz.start(50)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0tBuzz^^c2^^a0=^^c2^^a0threading.Thread(target=buzzerWork)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0tBuzz.start()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0tLed^^c2^^a0=^^c2^^a0threading.Thread(target=ledWork)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0tLed.start()
```

関数 on() で：

- 制御スレッドの終了を示すマーク「flag = 1」を定義する。
- バズを開始し、デューティサイクルを 50 %に設定する。
- 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： class threading.Thread(group=None, target=None, name=None, args=(), kwargs={}, *,

daemon=None)

構築メソッドの中で、主要なパラメーターは ターゲット 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 ターゲット 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。

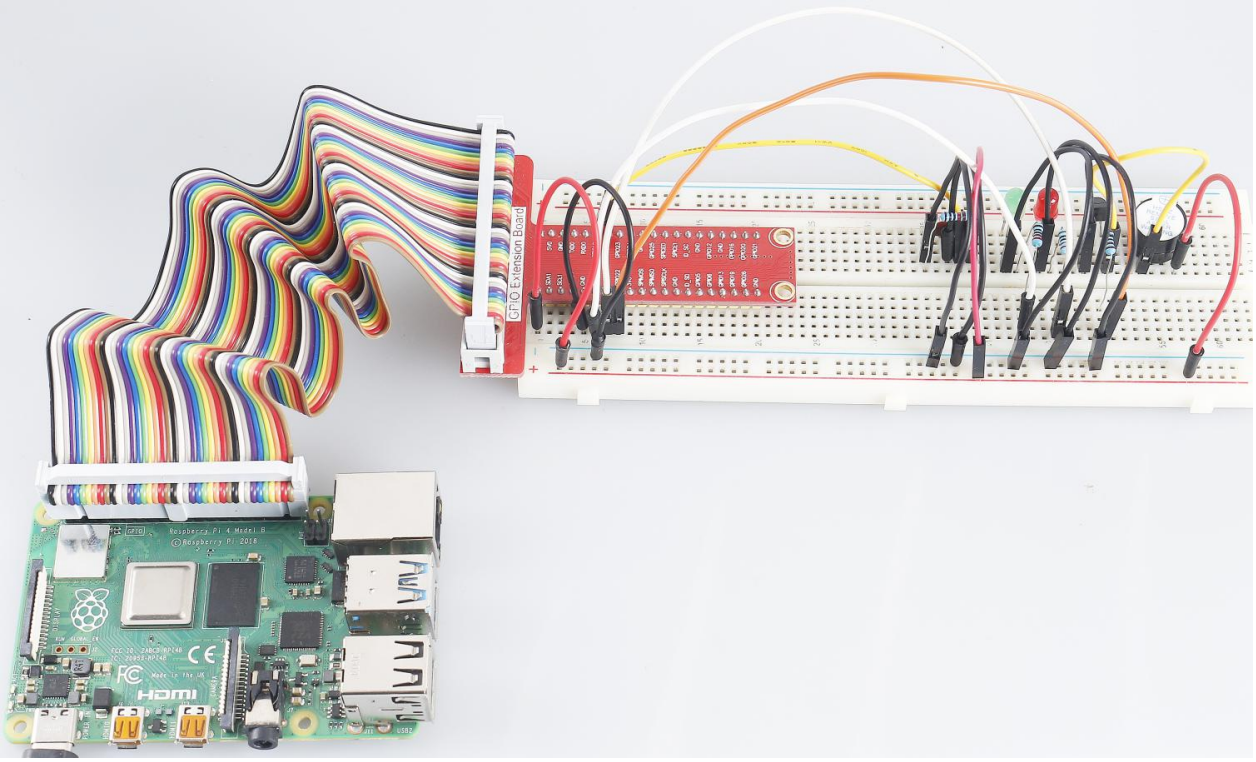
```
def^^c2^^a0off():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0flag
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0flag^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Buzz.stop()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ALedPin,GPIO.LOW)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(BLedPin,GPIO.LOW)
```

スレッド ledWork と BuzzWork を終了するために、関数 Off() は「flag = 0」を定義しそして、ブザーと LED をオフにする。

```
def^^c2^^a0main():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState^^c2^^a0=GPIO.
↪input(switchPin)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0currentState^^c2^^a0==^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0on()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0currentState^^c2^^a0==^^c2^^
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0off()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=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

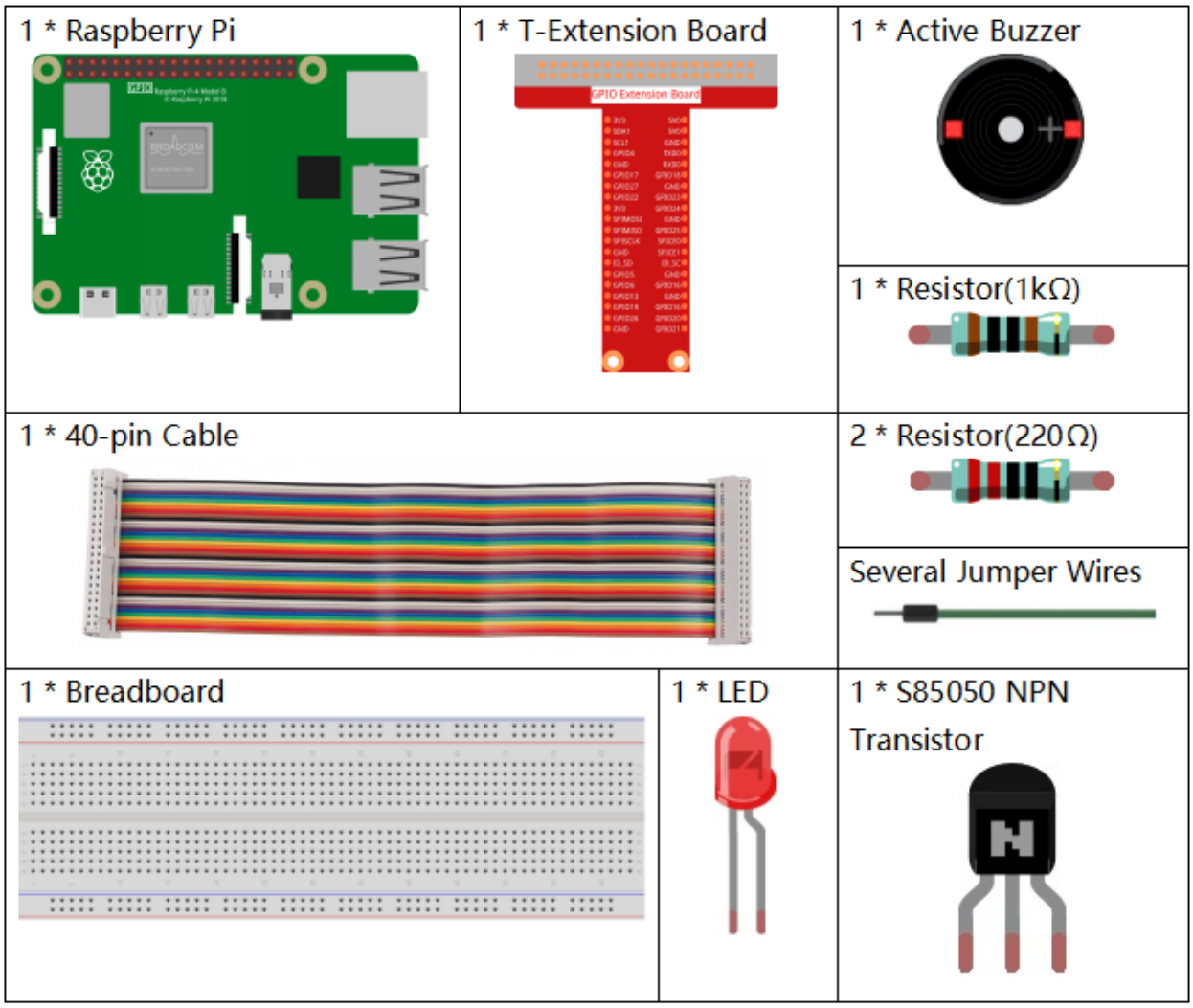


1.10.11 3.1.11 モールス信号発生器

前書き

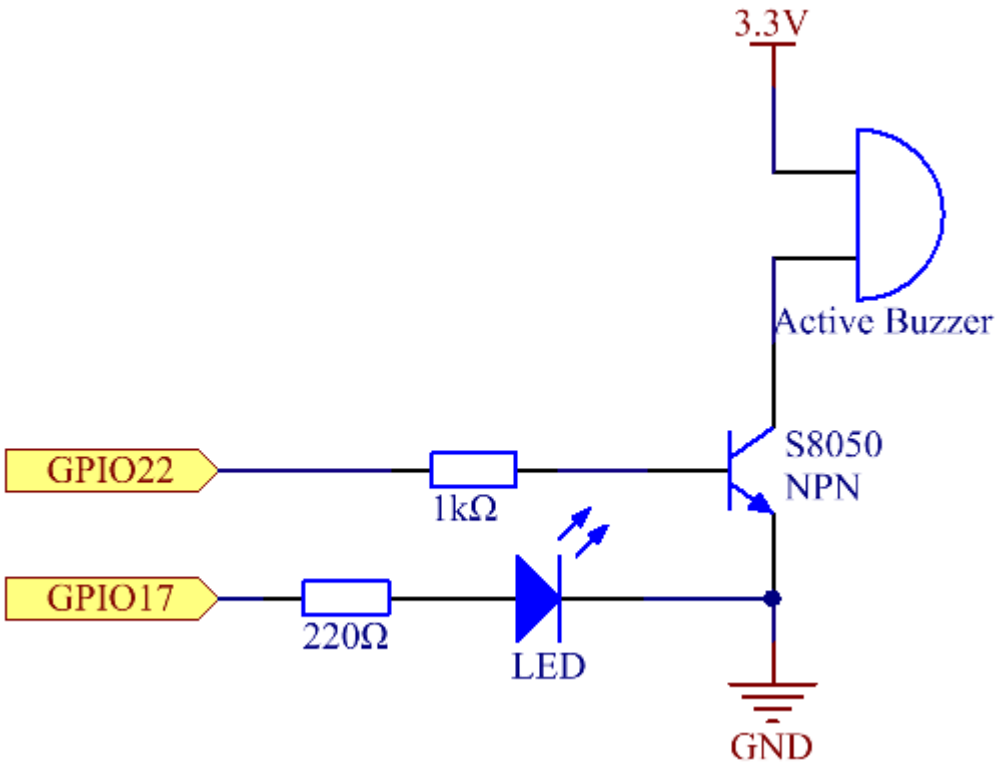
このレッスンでは、モールス符号ジェネレーターを作成する。ここでは、Raspberry Pi に一連の英語の文字を入力して、モールス符号として表示する。

部品



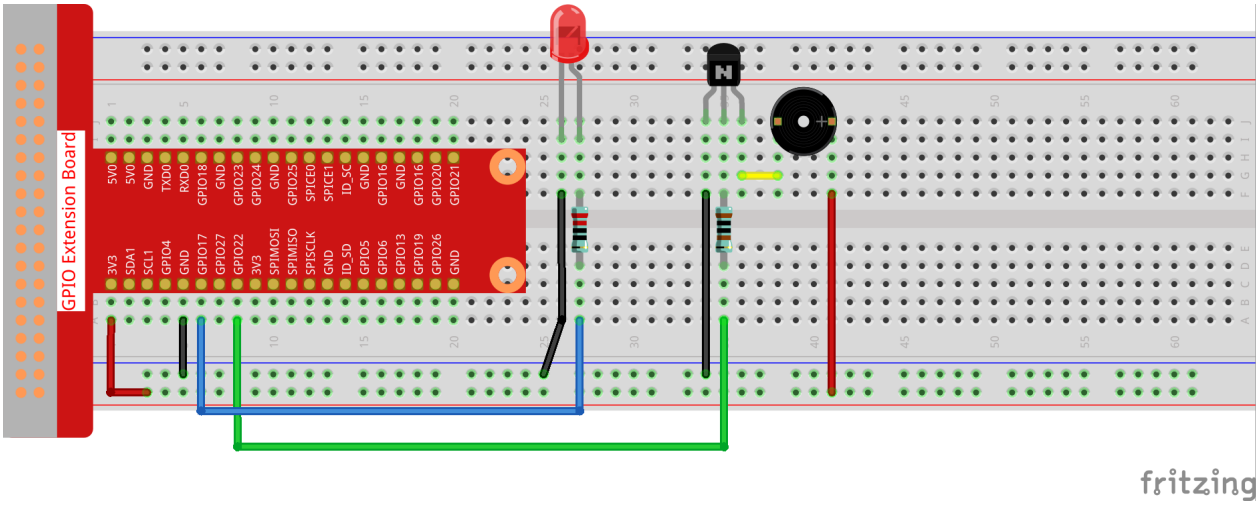
回路図

T-Board Name	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^^c2^^a0MORSE{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0word;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0*code;
};

struct^^c2^^a0MORSE^^c2^^a0morseDict[]=
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'A',"01"},^^c2^^a0{'B',"1000"},^^c2^^a0{'C',"1010"},
↪^^c2^^a0{'D',"100"},^^c2^^a0{'E',"0"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'F',"0010"},^^c2^^a0{'G',"110"},^^c2^^a0{'H',"0000"},
↪^^c2^^a0{'I',"00"},^^c2^^a0{'J',"0111"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'K',"101"},^^c2^^a0{'L',"0100"},^^c2^^a0{'M',"11"},
↪^^c2^^a0{'N',"10"},^^c2^^a0{'O',"111"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'P',"0110"},^^c2^^a0{'Q',"1101"},^^c2^^a0{'R',"010"},
↪^^c2^^a0{'S',"000"},^^c2^^a0{'T',"1"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'U',"001"},^^c2^^a0{'V',"0001"},^^c2^^a0{'W',"011"},
↪^^c2^^a0{'X',"1001"},^^c2^^a0{'Y',"1011"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'Z',"1100"},{'1',"01111"},^^c2^^a0{'2',"00111"},^^c2^^a0
↪{'3',"00011"},^^c2^^a0{'4',"00001"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'5',"00000"},{'6',"10000"},^^c2^^a0{'7',"11000"},
↪^^c2^^a0{'8',"11100"},^^c2^^a0{'9',"11110"},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{'0',"11111"},{'?',"001100"},^^c2^^a0{'/',"10010"},
↪^^c2^^a0{',',"110011"},^^c2^^a0{'.',"010101"},
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{';',"101010"},{'!',"101011"},^^c2^^a0{'@',"011010"},
↪^^c2^^a0{':',"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^^c2^^a0*lookup(char^^c2^^a0key,struct^^c2^^a0MORSE^^c2^^a0*dict,int^^c2^^a0length)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0(int^^c2^^a0i=0;i<length;i++)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(dict[i].word==key){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
```

(前のページからの続き)

```
}
```

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する。

```
void^^c2^^a0morsecode(char^^c2^^a0*code){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0pause^^c2^^a0=^^c2^^a0250;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*point^^c2^^a0=^^c2^^a0NULL;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0length^^c2^^a0=^^c2^^a0sizeof(morseDict)/
↪sizeof(morseDict[0]);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0(int^^c2^^a0i=0;i<strlen(code);i++)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0point=lookup(code[i],
↪morseDict,length);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0(int^^c2^^a0j=0;
↪j<strlen(point);j++){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
```

```
int^^c2^^a0toupper(int^^c2^^a0c)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0((c^^c2^^a0>=^^c2^^a0'a')^^c2^^a0&&
↪^^c2^^a0(c^^c2^^a0<=^^c2^^a0'z'))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0c^^c2^^a0+^^c2^^a0(
↪'A'^^c2^^a0-^^c2^^a0'a');
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a0c;
}
char^^c2^^a0*strupr(char^^c2^^a0*str)
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*orign=str;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0(;^^c2^^a0*str!='\0';^^c2^^a0str++)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0*str^^c2^^a0=^^c2^^a0toupper(*str);
↪
return^^c2^^a0orign;
}
```

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^^c2^^a0main(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*code;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0length=8;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0code^^c2^^a0=^^c2^^a0(char*)malloc(sizeof(char)*length);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf(
↪"Please^^c2^^a0input^^c2^^a0the^^c2^^a0messenger:");
        delay(100);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0scanf("%s",code);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0code=strupr(code);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf("%s\n",code);
        delay(100);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0morsecode(code);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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、 長さ 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time

BeepPin=22
ALedPin=17

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',
    }

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(ALedPin,GPIO.OUT,initial=GPIO.LOW)
```

(次のページに続く)

(前のページからの続き)

```
def on():
    GPIO.output(BeepPin, 1)
    GPIO.output(ALedPin, 1)

def off():
    GPIO.output(BeepPin, 0)
    GPIO.output(ALedPin, 0)

def beep(dt):       # dt for delay time.
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)

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)

def main():
    while True:
        code=input("Please input the messenger:")
        code = code.upper()
        print(code)
        morsecode(code)

def destroy():
    print("")
    GPIO.output(BeepPin, GPIO.LOW)
    GPIO.output(ALedPin, GPIO.LOW)
    GPIO.cleanup()

if __name__ == '__main__':
```

(次のページに続く)

(前のページからの続き)

```
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
MORSECODE^^c2^^a0=^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'A':'01',^^c2^^a0'B':'1000',^^c2^^a0'C':'1010',^^c2^^a0'D
↪':'100',^^c2^^a0'E':'0',^^c2^^a0'F':'0010',^^c2^^a0'G':'110',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'H':'0000',^^c2^^a0'I':'00',^^c2^^a0'J':'0111',^^c2^^a0'K
↪':'101',^^c2^^a0'L':'0100',^^c2^^a0'M':'11',^^c2^^a0'N':'10',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'O':'111',^^c2^^a0'P':'0110',^^c2^^a0'Q':'1101',^^c2^^a0
↪'R':'010',^^c2^^a0'S':'000',^^c2^^a0'T':'1',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'U':'001',^^c2^^a0'V':'0001',^^c2^^a0'W':'011',^^c2^^a0'X
↪':'1001',^^c2^^a0'Y':'1011',^^c2^^a0'Z':'1100',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'1':'01111',^^c2^^a0'2':'00111',^^c2^^a0'3':'00011',
↪^^c2^^a0'4':'00001',^^c2^^a0'5':'00000',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'6':'10000',^^c2^^a0'7':'11000',^^c2^^a0'8':'11100',
↪^^c2^^a0'9':'11110',^^c2^^a0'0':'11111',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'?':'001100',^^c2^^a0'/':'10010',^^c2^^a0',':'110011',
↪^^c2^^a0'.':'010101',^^c2^^a0';':'101010',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0'!':'101011',^^c2^^a0'@':'011010',^^c2^^a0':':'111000',
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
```

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^^c2^^a0on():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(BeepPin,^^c2^^a01)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ALedPin,^^c2^^a01)
```

関数 on() はブザーと LED を起動する。

```
def^^c2^^a0off():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(BeepPin,^^c2^^a00)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(ALedPin,^^c2^^a00)
```

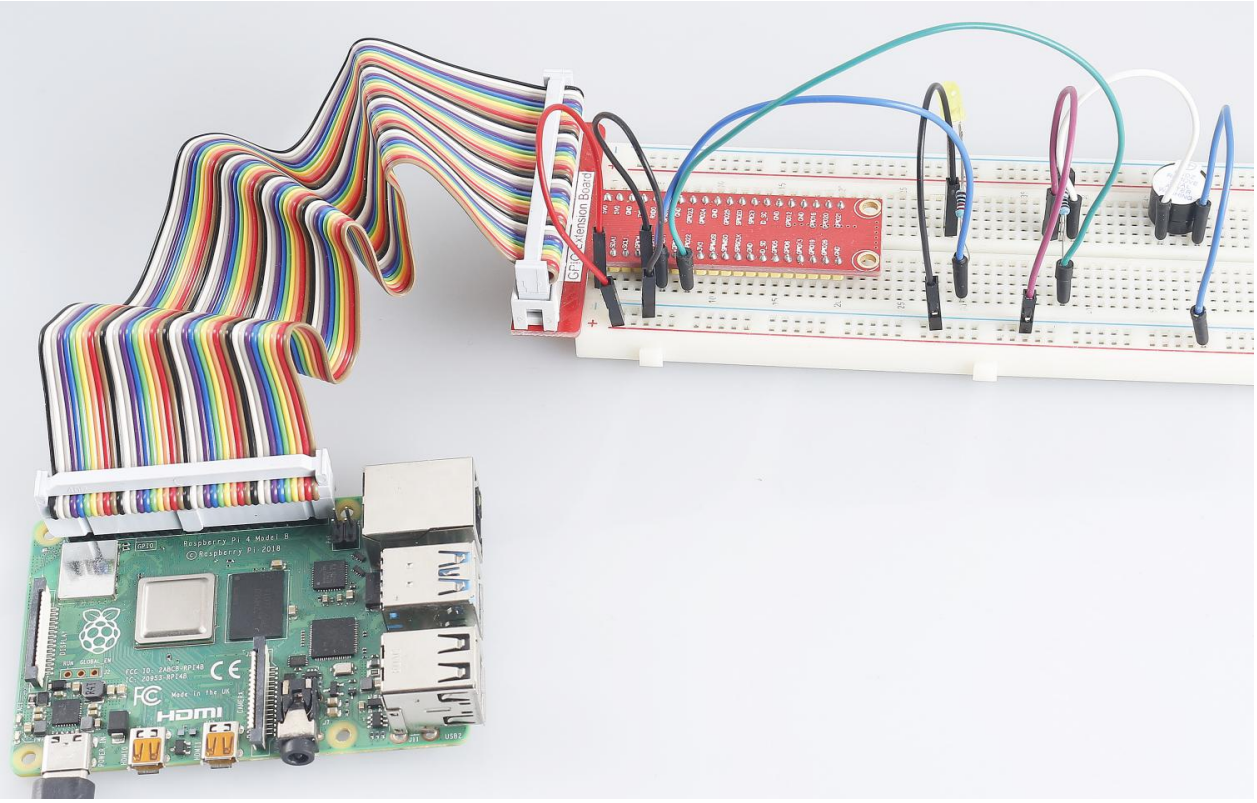
関数 off() はブザーと LED をオフにする。

```
def^^c2^^a0beep(dt):^^c2^^a0^^c2^^a0^^c2^^a0
↪#^^c2^^a0x^^c2^^a0for^^c2^^a0dalay^^c2^^a0time.
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0on()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(dt)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0off()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.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する。

```
def^^c2^^a0morsecode(code):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pause^^c2^^a0=^^c2^^a00.25
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0letter^^c2^^a0in^^c2^^a0code:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0tap^^c2^^a0in^^c2^^a0MORSECOD
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^
↪'0':
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪2)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
```

現象画像

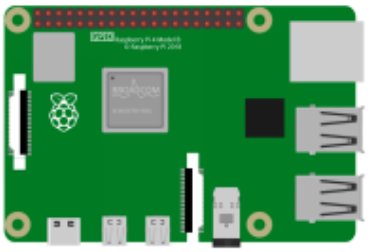
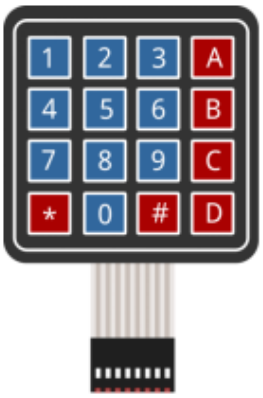
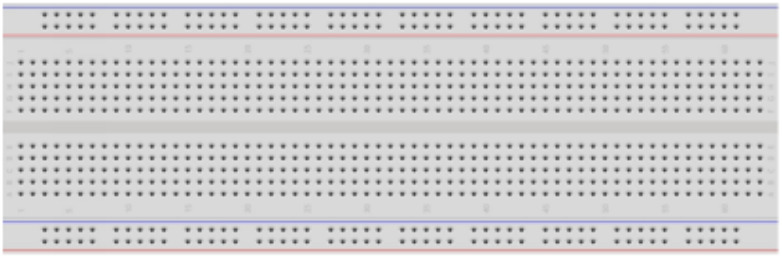
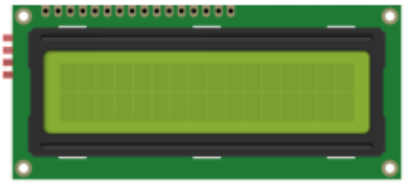


1.10.12 3.1.12 ゲーム–数字推測

前書き

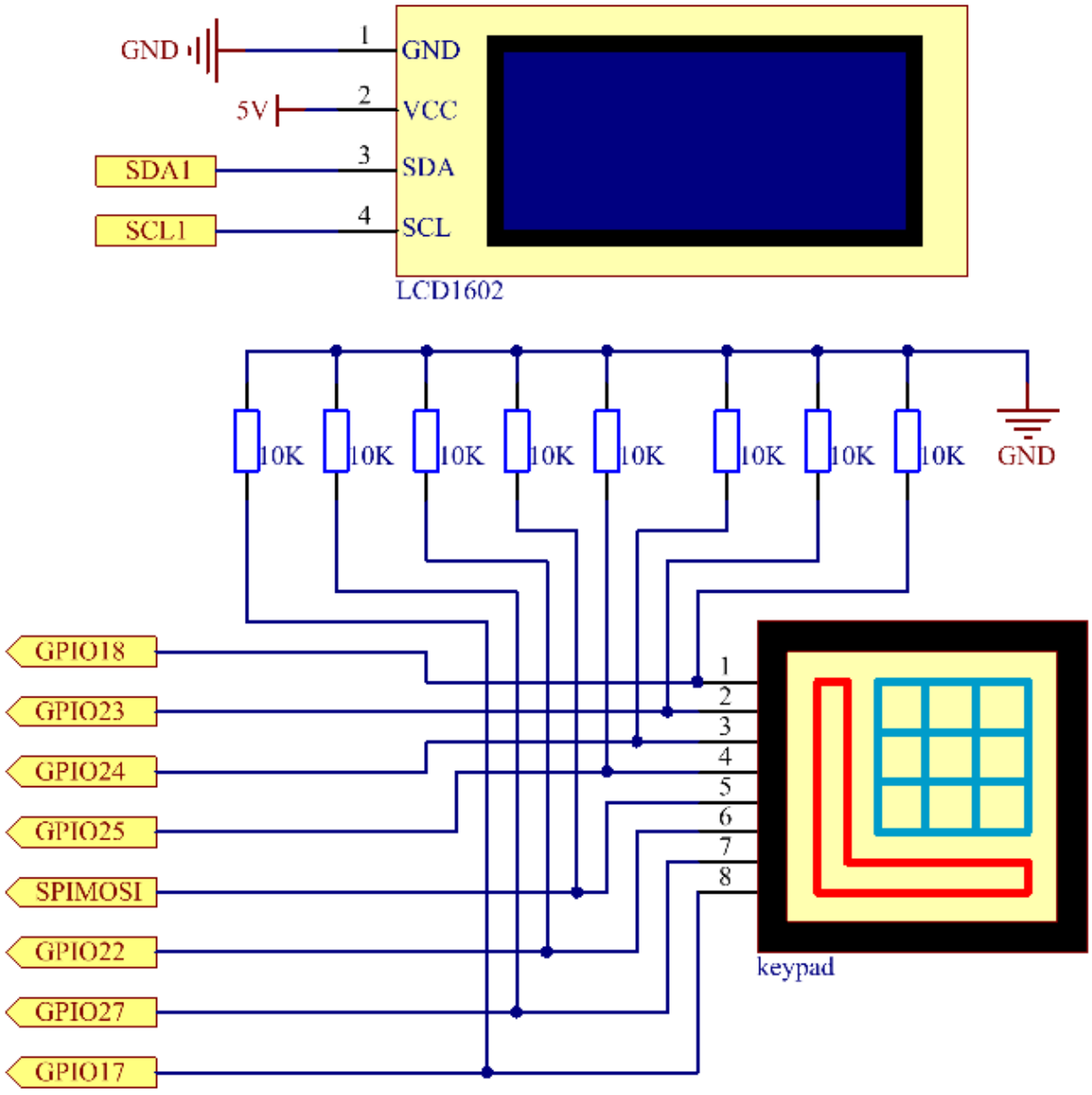
数字を推測することは、あなたとあなたの友人が交互に数字を入力する楽しいパーティーゲームである（0〜99）。プレーヤーがなぞなぞに正しく答えるまで、数字を入力すると範囲は小さくなる。その後、プレイヤーは敗北し、処罰される。たとえば、ラッキーナンバーが 51 で、プレーヤーがそれを見ることができず、プレーヤーが 50 を入力する場合、番号範囲は 50〜99 に変わる。もしプレイヤーが 70 を入力する場合、番号の範囲は 50〜70 になる。プレイヤーが 51 を入力した場合、このプレイヤーは不運なプレイヤーである。ここでは、キーパッドを使用して数字を入力し、LCD を使用して結果を表示させる。

部品

1 * Raspberry Pi 	1 * T-Extension Board	1 * Keypad 
1 * 40-pin Cable		Several Jumper Wires
		8 * Resistor 10KΩ
1 * Breadboard 		1 * I2C LCD1602 

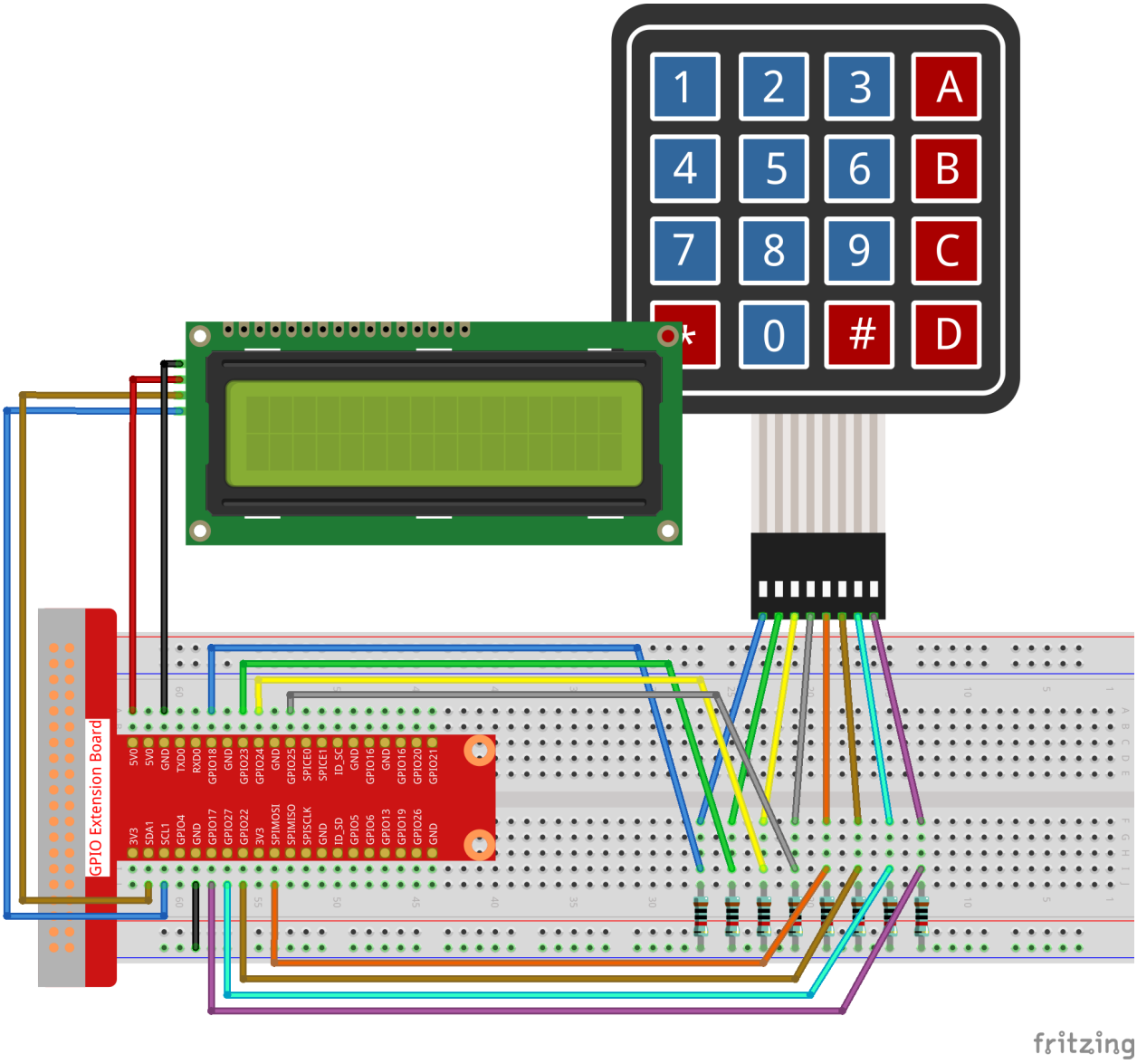
回路図

T-Board Name	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* **構成** （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^^c2^^a0A^^c2^^a0to^^c2^^a0go!
```

「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 ^^e2^^80^^b9point^^e2^^80^^b9 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* **キーパッド** 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^^c2^^a0from^^c2^^a0here
/*****************************************/
void^^c2^^a0init(void){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0fd^^c2^^a0=^^c2^^a0wiringPiI2CSetup(LCDAddr);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_init();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_clear();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0i=0^^c2^^a0;^^c2^^a0i<4^^c2^^a0;
↪^^c2^^a0i++)^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pinMode(rowPins[i],
↪^^c2^^a0OUTPUT);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pinMode(colPins[i],
↪^^c2^^a0INPUT);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_clear();
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,^^c2^^a00,^^c2^^a0"Welcome!");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,^^c2^^a01,^^c2^^a0
↪"Press^^c2^^a0A^^c2^^a0to^^c2^^a0go!");
}
```

この関数は元々 I2C LCD1602 と Keypad を定義し、「Welcome!」と「Press A to go!」を表示するために使用される。

```
void^^c2^^a0init_new_value(void){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0srand(time(0));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pointValue^^c2^^a0=^^c2^^a0rand()%100;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper^^c2^^a0=^^c2^^a099;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lower^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf("point^^c2^^a0is^^c2^^a0%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^^c2^^a0detect_point(void){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(count^^c2^^a0>^^c2^^a0pointValue){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(count^^c2^^a0
↪<^^c2^^a0upper){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper^^c
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(count^^c2^^a0<^^c2^^a0pointValue){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(count^^c2^^a0>
↪^^c2^^a0lower){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lower^^c
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(count^^c2^^a0=^^c2^^a0pointValue){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

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 上限 と 下限 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^^c2^^a0lcd_show_input(bool^^c2^^a0result){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*str=NULL;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0str^^c2^^a0=(char*)malloc(sizeof(char)*3);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_clear();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(result^^c2^^a0==^^c2^^a01){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,1,"You
↪'ve^^c2^^a0got^^c2^^a0it!");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(5000);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0init_new_value();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_show_input(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,0,"Enter^^c2^^a0number:");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Int2Str(str,count);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(13,0,str);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Int2Str(str,lower);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(0,1,str);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(3,1,"<Point<");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0Int2Str(str,upper);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0write(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^^c2^^a0main(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0pressed_keys[BUTTON_NUM];
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0last_key_pressed[BUTTON_NUM];
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(wiringPiSetup()^^c2^^a0==^^c2^^a0-1){^^c2^^a0//
↪when^^c2^^a0initialize^^c2^^a0wiring^^c2^^a0failed,print^^c2^^a0messageto^^c2^^a0screen
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0printf(
↪"setup^^c2^^a0wiringPi^^c2^^a0failed^^c2^^a0!");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0init();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0init_new_value();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
```

(次のページに続く)

(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0keyRead(pressed_keys);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0bool^^c2^^a0comp^^c2^^a0=^^c2^^a0keyComp
↪keys,^^c2^^a0last_key_pressed);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(!comp){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(press
↪keys[0]^^c2^^a0!=^^c2^^a00){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0]^^c2^^a0==^^c2^^a0'A'){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪new_value();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0]^^c2^^a0==^^c2^^a0'D'){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪point();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(result);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0]^^c2^^a0>='0'^^c2^^a0&&^^c2^^a0pressed_keys[0]^^c2^^a0<=^^c2^^a0'9'){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0]^^c2^^a0-^^c2^^a048);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪=10){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪point();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(result);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
```

(次のページに続く)

(前のページからの続き)

```
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0keyCopy(
↪key_pressed,^^c2^^a0pressed_keys);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(100);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00;
}
```

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 を押すと、coun の値が変更されます。 カウント が 10 より大きい場合、判断が開始されます。
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^^c2^^a0A^^c2^^a0to^^c2^^a0go!
```

「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 ^^e2^^80^^b9point^^e2^^80^^b9 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

```
import RPi.GPIO as GPIO
import time
import LCD1602
import random

###################### HERE IS THE KEYPAD LIBRARY TRANSPLANTED FROM Arduino ############
#class Key:Define some of the properties of Key
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
```

(次のページに続く)

(前のページからの続き)

```
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys

################ EXAMPLE CODE START HERE ################

count = 0
pointValue = 0
upper=99
lower=0

def setup():
    global keypad, last_key_pressed,keys
    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 = []
    LCD1602.init(0x27, 1)    # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'Welcome!')
    LCD1602.write(0, 1, 'Press A to Start!')

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))


def detect_point():
    global count,upper,lower
    if count > pointValue:
```

(次のページに続く)

(前のページからの続き)

```
        if count < upper:
                upper = count
    elif count < pointValue:
        if count > lower:
                lower = count
    elif count == pointValue:
        count = 0
        return 1
    count = 0
    return 0

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))

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","#","*"]):
```

(次のページに続く)

(前のページからの続き)

```
                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)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Release resource
    GPIO.cleanup()
    LCD1602.clear()

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

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* **キーパッド** 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^^c2^^a0init_new_value():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0pointValue,upper,count,lower
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pointValue^^c2^^a0=^^c2^^a0random.randint(0,99)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper^^c2^^a0=^^c2^^a099
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lower^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0print('point^^c2^^a0is^^c2^^a0%d'^^c2^^a0%(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^^c2^^a0detect_point():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0count,upper,lower
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0count^^c2^^a0>^^c2^^a0pointValue:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0count^^c2^^a0
↪<^^c2^^a0upper:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0upper^^c
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0count^^c2^^a0<^^c2^^a0pointValue:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0count^^c2^^a0>
↪^^c2^^a0lower:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lower^^c
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0count^^c2^^a0==^^c2^^a0pointValue:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a01
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0count^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^c2^^a00
```

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 上限 と 下限 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^^c2^^a0lcd_show_input(result):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.clear()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0result^^c2^^a0==^^c2^^a01:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,1,
↪'You^^c2^^a0have^^c2^^a0got^^c2^^a0it!')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(5)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0init_new_value()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lcd_show_input(0)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,0,'Enter^^c2^^a0number:')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(13,0,str(count))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(0,1,str(lower))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.write(3,1,'^^c2^^a0<^^c2^^a0Point^^c2^^a0
↪<^^c2^^a0')
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0LCD1602.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。

sstr(count)：Because 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^^c2^^a0loop():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0keypad,^^c2^^a0last_key_pressed,count
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(True):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0result^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0pressed_
↪keys^^c2^^a0=^^c2^^a0keypad.read()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0len(pressed_
↪keys)^^c2^^a0!=^^c2^^a00^^c2^^a0and^^c2^^a0last_key_pressed^^c2^^a0!=^^c2^^a0pressed_
↪keys:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^
↪keys^^c2^^a0==^^c2^^a0["A"]:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪new_value()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(0)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2
↪keys^^c2^^a0==^^c2^^a0["D"]:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪point()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(result)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2
↪keys[0]^^c2^^a0in^^c2^^a0keys:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0]^^c2^^a0in^^c2^^a0list(["A","B","C","D","#","*"]):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪keys[0])
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪=^^c2^^a010:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪point()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0
↪show_input(result)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0print(pr
↪keys)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0last_key_
↪pressed^^c2^^a0=^^c2^^a0pressed_keys
```

(次のページに続く)

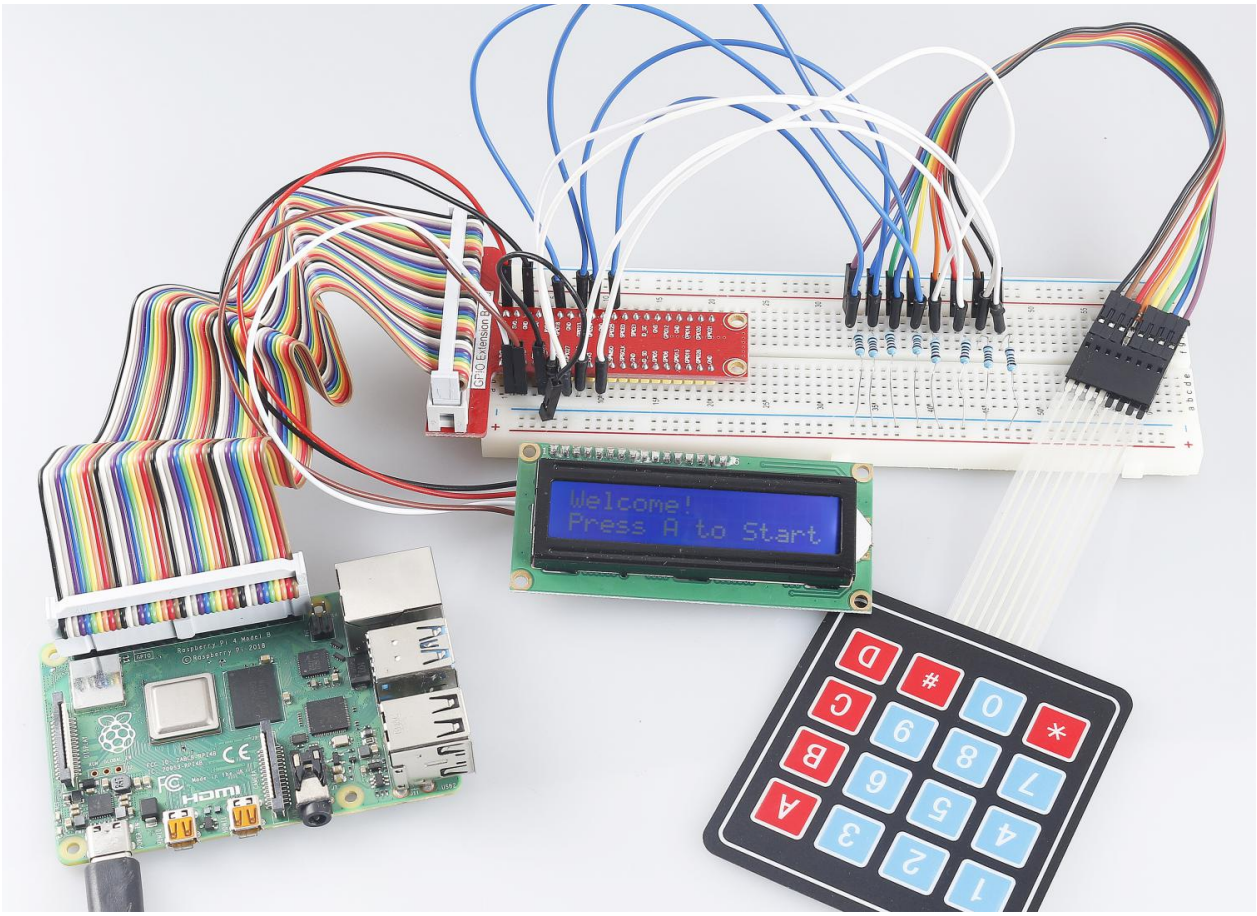
(前のページからの続き)

```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.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 を押すと、coun の値が変更されます。 カウント が 10 より大きい場合、判断が開始されます。
6. ゲームの変化とその値は LCD1602 に表示される。

現象画像

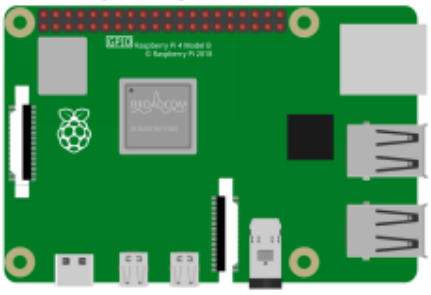
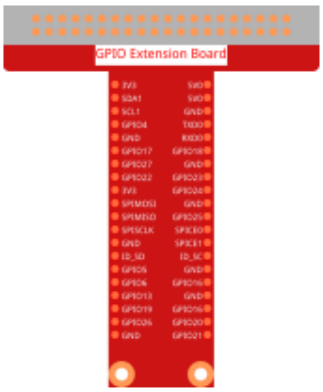
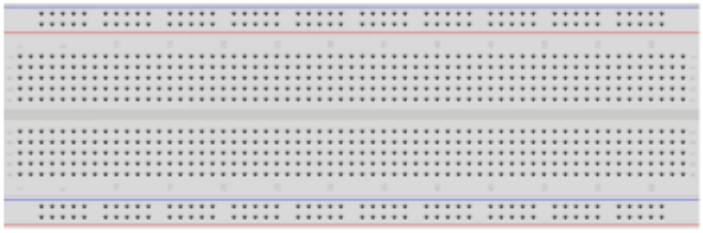


1.10.13 3.1.13 ゲーム– 10 秒

前書き

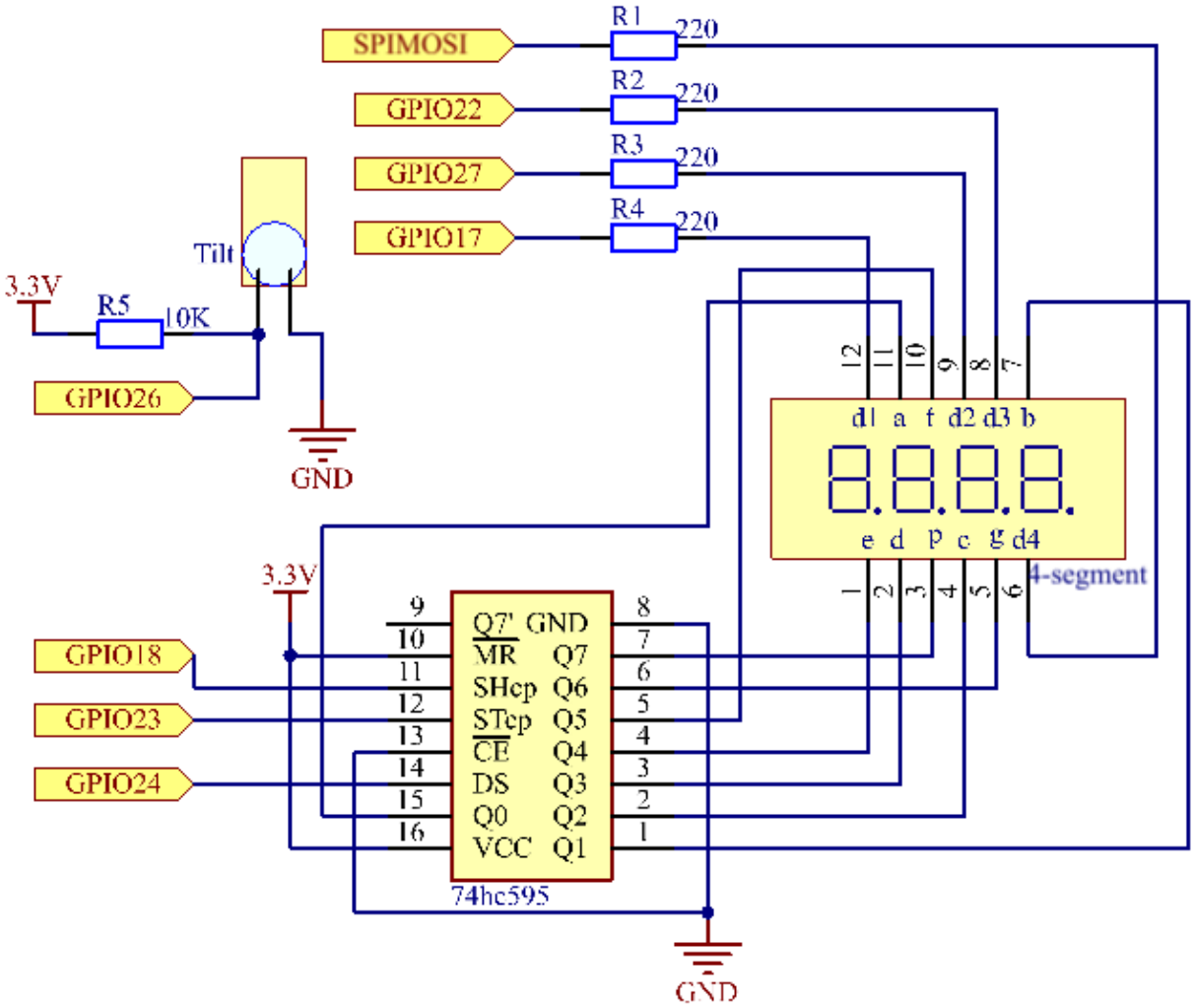
それから、あなたの集中力に挑戦できるゲームデバイスを作ろう。傾斜スイッチをスティックにつないで、魔法の棒を作る。棒を振ると、4 桁のセグメントディスプレイがカウントを開始し、もう一度振るとカウントを停止する。表示されたカウントを 10.00 に保つことに成功した場合、あなたが勝つ。友達とゲームをプレイして、タイムウィザードが誰かを確認できる。

部品

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
 
</td>
<td>1 * 4-Digit 7-Segment Display
 </td>
</tr>
<tr>
<td colspan="2" rowspan="2">1 * 40-pin Cable
 </td>
<td>1 * 74HC595
</td>
</tr>
<tr>
<td>1 * Tilt Switch
</td>
</tr>
<tr>
<td colspan="2" rowspan="2">1 * Breadboard
  </td>
<td>Several Jumper Wires
</td>
</tr>
<tr>
<td>4 * Resistor(220Ω)
 </td>
</tr>
<tr>
<td colspan="2"></td>
<td>1 * Resistor 10KΩ
 </td>
</tr>
</table>

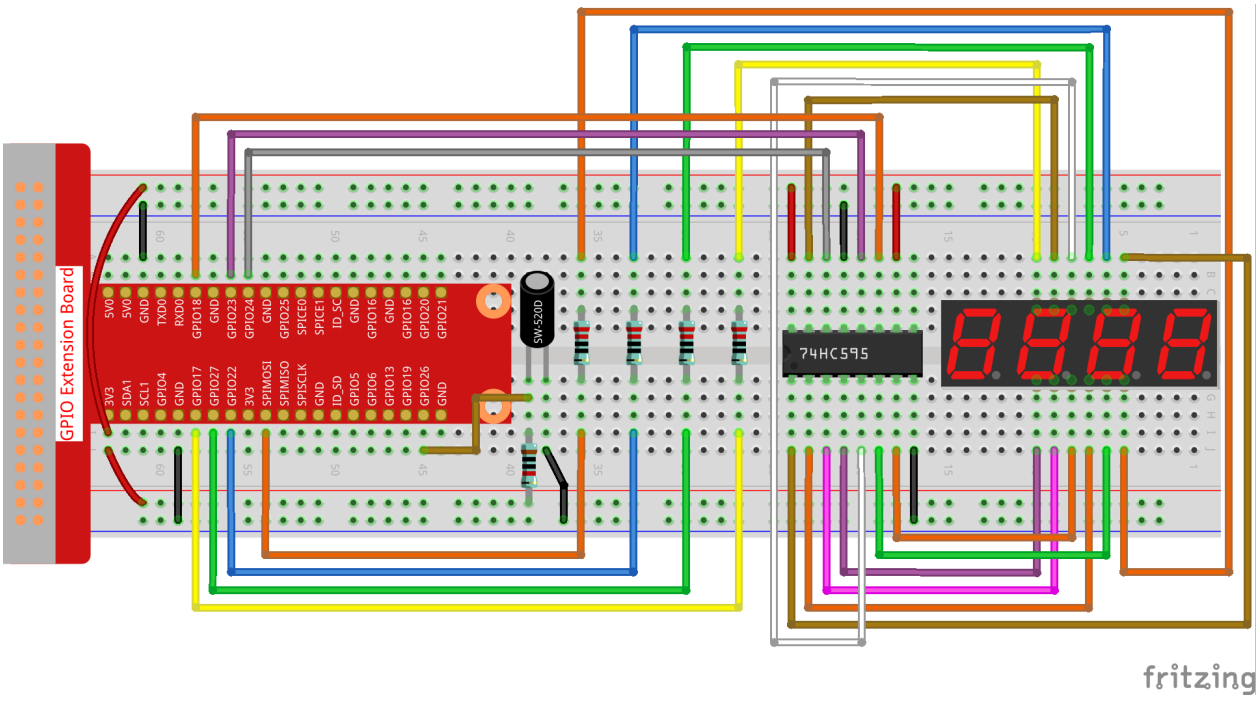
回路図

T-Board Name	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^^c2^^a0stateChange(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(gameState^^c2^^a0==^^c2^^a00){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0counter^^c2^^a0=^^c2^^a00;
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(1000);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0ualarm(10000,10000);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}else{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0alarm(0);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0delay(1000);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0gameState^^c2^^a0=^^c2^^a0(gameState^^c2^^a0+^^c2^^a01)
↪%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^^c2^^a0loop(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0currentState^^c2^^a0=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0lastState=0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=digitalRead(sensorPin);
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if((currentState==0)&&
↪(lastState==1)){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stateCha
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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 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

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。 ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

sensorPin = 26

SDI = 24
RCLK = 23
SRCLK = 18

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

counter = 0
timer =0
gameState =0

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

(次のページに続く)

(前のページからの続き)

```
        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 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]-0x80)

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

def stateChange():
    global gameState
    global counter
    global timer1
```

(次のページに続く)

(前のページからの続き)

```
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState ==1:
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState+1)%2

def loop():
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState=currentState

def timer():
    global counter
    global timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
    counter += 1

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)
    GPIO.setup(sensorPin, GPIO.IN)

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

(次のページに続く)

(前のページからの続き)

```
    timer1.cancel()

if __name__ == '__main__':  # Program starting from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

コードの説明

```
def^^c2^^a0stateChange():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0gameState
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0counter
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0timer1
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0gameState^^c2^^a0==^^c2^^a00:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0counter^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timer()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0gameState^^c2^^a0==1:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timer1.cancel()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0time.sleep(1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0gameState^^c2^^a0=^^c2^^a0(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^^c2^^a0loop():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0counter
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState^^c2^^a0=^^c2^^a00
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0currentState=GPIO.
↪input(sensorPin)
```

(次のページに続く)

(前のページからの続き)

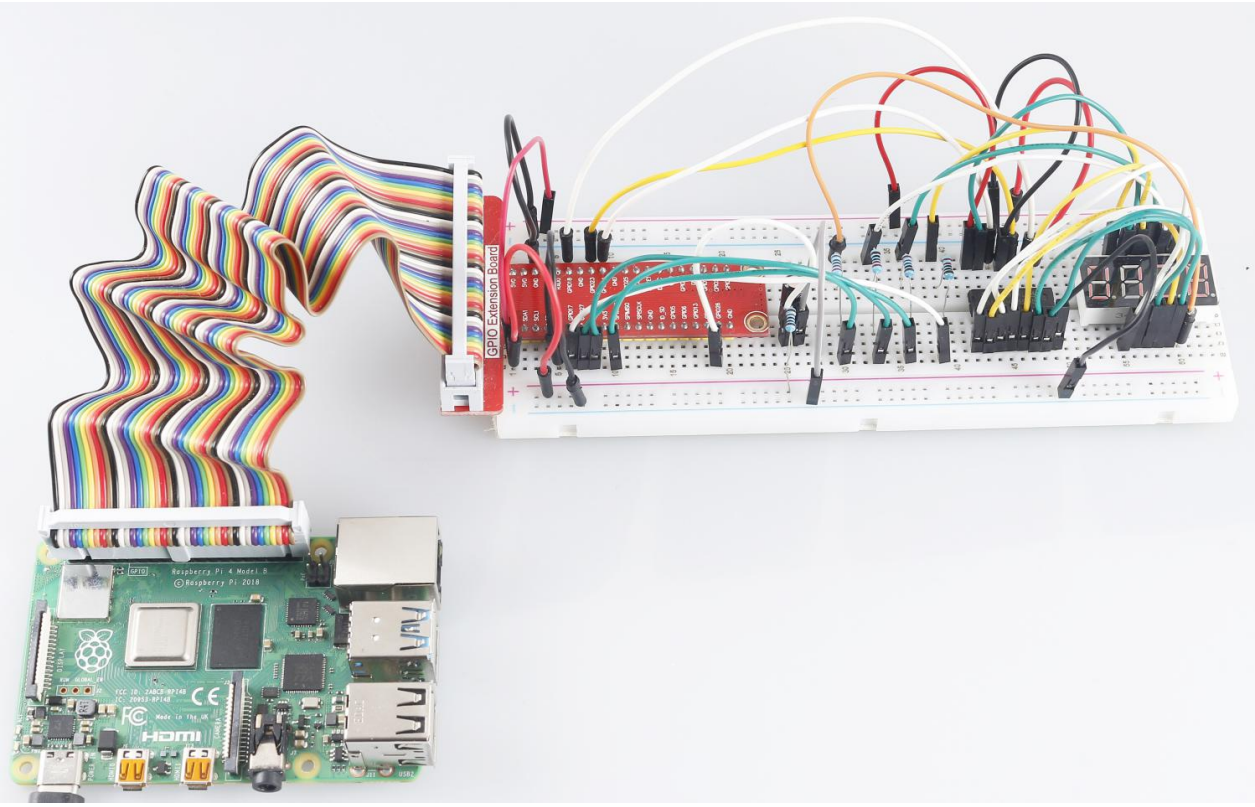
```
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(currentState^^c2^^a0==^^c2^^a
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stateCha
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0lastState=currentState
```

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



1.10.14 3.1.14 ゲーム–Not Not-頭脳耐久ゲーム

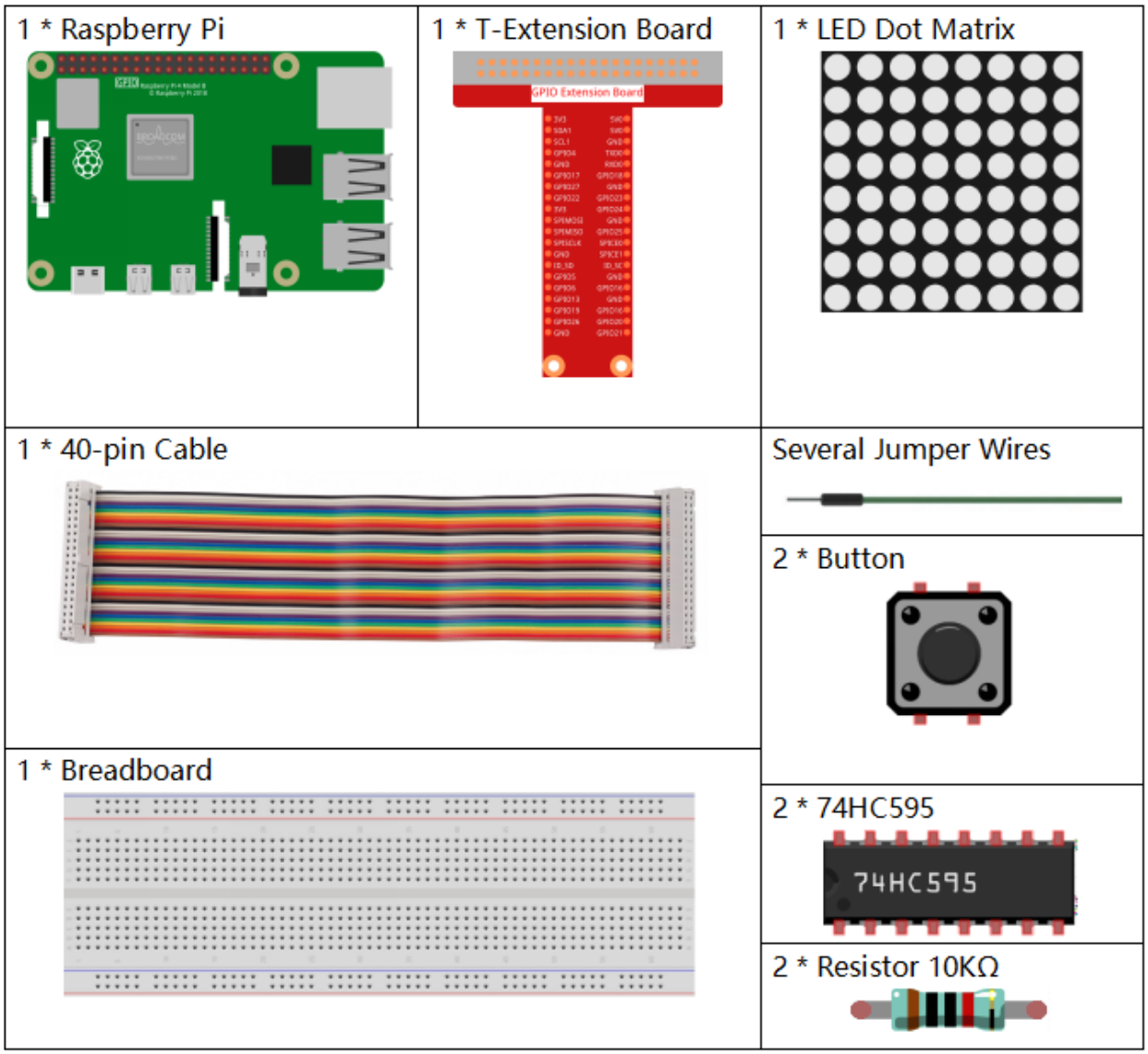
前書き

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。

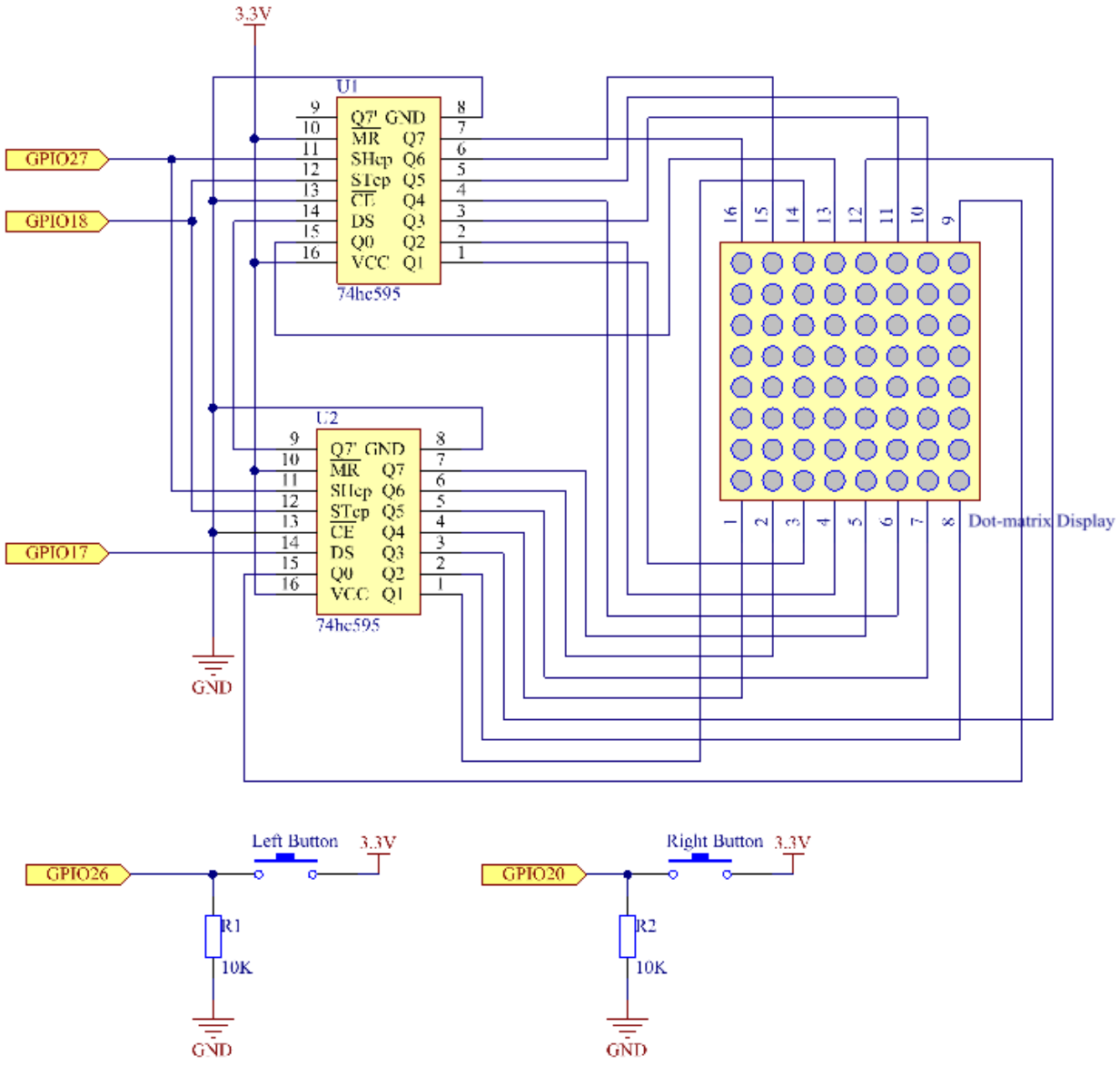
このゲームは本当にあなたの逆思考を実践することができる。

部品



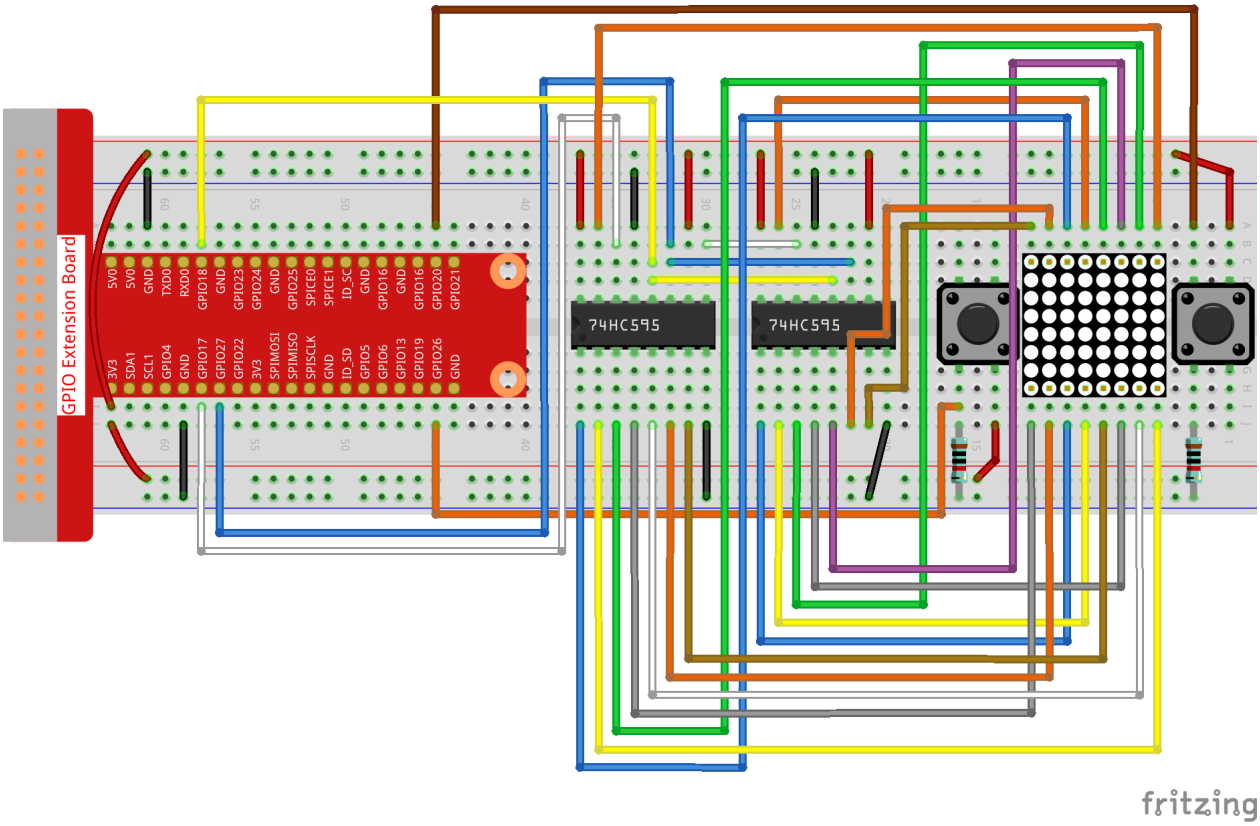
回路図

T-Board Name	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 2： コードのフォルダに移動します。

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.14/
```

ステップ 3： コンパイルします。

```
gcc 3.1.14_GAME_NotNot.c -lwiringPi
```

ステップ 4： 実行します。

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

プログラムの開始後、左矢印または右矢印がドットマトリックス上でランダムに更新されます。限られた時間内に矢印の反対方向にボタンを押す必要があります。次に、「√」がドットマトリックスに表示されます。時間切れの場合、または矢印と同じ方向のボタンが押された場合は、外に出て、ドットマトリックスに「x」が表示されます。また、2 つの新しいボタンを追加したり、上下左右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ます。

4 方向でゲームの難易度を上げることができます。

コードの説明

1.1.6 LED ドットマトリックス に基づいて、このレッスンでは 2 ボタンを追加して面白いゲームデバイスを作成します。 したがって、ドットマトリックスにあまり詳しくない場合は、 *1.1.6 LED* **ドットマトリックス** 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、 タイマー 1 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、 タイマー 1 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、 タイマー 2 を生成する。
5. タイマー 2 が時間切れを想起させると、 ステップ 1 を再実行する。

```
struct^^c2^^a0GLYPH{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*word;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0unsigned^^c2^^a0char^^c2^^a0code[8];
};

struct^^c2^^a0GLYPH^^c2^^a0arrow[2]=
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{"right",{0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF}},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0//^^c2^^a0{"down",{0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,
↪0xFF}},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0//^^c2^^a0{"up",{0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF}
↪},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{"left",{0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF}}
};

struct^^c2^^a0GLYPH^^c2^^a0check[2]=
{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{"wrong",{0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF}},
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{"right",{0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF}}
};
```

GLYPH 構造は辞書のように動作します。 word 属性は辞書の **key** に対応し、 code 属性は **value** に対応します。

ここでは、ドットマトリックスが画像を表示するための配列（8 × 8 ビットの配列）を格納するためのコードを使用しています。

ここでは、配列 arrow を用いて、上下左右の矢印パターンを LED ドットマトリクスに表示することができます。

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^^c2^^a0*lookup(char^^c2^^a0*key,struct^^c2^^a0GLYPH^^c2^^a0*glyph,int^^c2^^a0length)
↪{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0(int^^c2^^a0i=0;i<length;i++)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(strcmp(glyph[i].word,
↪key)==0){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0return^^
↪code;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

関数 lookup() は、辞書を引くことで機能します。key を定義し、構造 GLYPH *glyph の key 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。

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 を返す。

```
void^^c2^^a0display(char^^c2^^a0*glyphCode){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for(int^^c2^^a0i;i<8;i++){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_in(glyphCode[i]);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_in(0x80>>i);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_out();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

指定したパターンをドットマトリックスに表示する。

```
void^^c2^^a0createGlyph(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0srand(time(NULL));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0int^^c2^^a0i=rand()%(sizeof(arrow)/sizeof(arrow[0]));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint=arrow[i].word;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage="PLAY";
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0alarm(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^^c2^^a0checkPoint(char^^c2^^a0*inputKey){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0alarm(0)==0;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if(inputKey==waypoint||inputKey=="empty")
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint="wrong";
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint="right";
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage="CHECK";
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0alarm(1);
}
```

checkPoint() は、ボタンの入力をチェックするためのもので、ボタンが押されていなかったり、矢印と同じ方向のボタンが押されていたりすると、ウェイポイントの結果がおかしくなり、ドットマトリクスに「x」と表示されます。そうでなければ、ウェイポイントの結果は正しく、ドットマトリクスには「√」と表示されます。ここでは、ステージ が CHECK となり、1 秒単位の目覚まし時計機能を設定することができます。

alarm() は「目覚まし時計」とも呼ばれ、タイマーを設定し、設定した時間が経過すると SIGALRM 信号をプログレスに送信します。

```
void^^c2^^a0getKey(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(digitalRead(AButtonPin)==1&&
↪digitalRead(BButtonPin)==0)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{checkPoint("right");}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if^^c2^^a0(digitalRead(AButtonPin)==0&&
↪digitalRead(BButtonPin)==1)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{checkPoint("left");}
}
```

getKey() は、これら 2 つのボタンの状態を読み取ります。右ボタンが押されている場合、関数 checkPoint() のパラメータは right 、左ボタンが押されている場合、パラメータは left となります。

```
void^^c2^^a0timer(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(stage=="PLAY"){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0checkPoint("empty");
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(stage=="CHECK"){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0createGlyph();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

これまでは、 alarm() が鳴ったときに timer() が呼ばれていました。そして、「PLAY」モードでは、結果を判断するために checkPoint() を呼び出すことになっています。「CHECK」モードの場合は、 createGlyph() を呼び出してパターンを選択します。

```
void^^c2^^a0main(){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0setup();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0signal(SIGALRM,timer);
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0createGlyph();
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0char^^c2^^a0*code^^c2^^a0=^^c2^^a0NULL;
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while(1){
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0(stage^^c2^^a0==^^c2^^a0
↪"PLAY")
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0code=loc
↪arrow,sizeof(arrow)/sizeof(arrow[0]));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display(
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0getKey()
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else^^c2^^a0if(stage^^c2^^a0==^^c2^^a0
↪"CHECK")
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0{
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0code^^c2
↪check,sizeof(check)/sizeof(check[0]));
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display(
↪
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0}
}
```

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。

プログラムが起動したら、最初に 1 回だけ createGlyph() を呼び出してから、ループを開始します。

ループ内：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ます。チェックモードの場合、表示されるのは「x」または「√」です。

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ます。

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

ステップ 3：実行します。

```
sudo python3 3.1.14_GAME_NotNot.py
```

プログラムを起動すると、ドットマトリックス上に右または左を指す矢印が表示されます。あなたがする必要があるのは、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す。次に、「√」がドットマトリックスに表示されます。時間切れの場合、または矢印と同じ方向のボタンが押された場合は、外に出て、ドットマトリックスに「x」が表示されます。また、2 つの新しいボタンを追加したり、上下左右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ます。4 方向でゲームの難易度を上げることができます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、その前に、davinci-kit-for-raspberry-pi/python のようなソースコードパスに移動する必要があります。

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

SDI   = 17
RCLK  = 18
SRCLK = 27

timerPlay = 0
timerCheck = 0

```

(次のページに続く)

(前のページからの続き)

```
AButtonPin = 20
BButtonPin = 26

waypoint = "NULL"
stage = "NULL"

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]
}

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)
    GPIO.setup(AButtonPin,GPIO.IN)
    GPIO.setup(BButtonPin,GPIO.IN)

# 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)
        GPIO.output(SRCLK, GPIO.LOW)

def display(glyphCode):
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80>>i)
```

(次のページに続く)

(前のページからの続き)

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

def creatGlyph():
    global waypoint
    global stage
    global timerPlay
    waypoint=random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()

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()

def timeOut():
    checkPoint("empty")

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")

def main():
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
```

(次のページに続く)

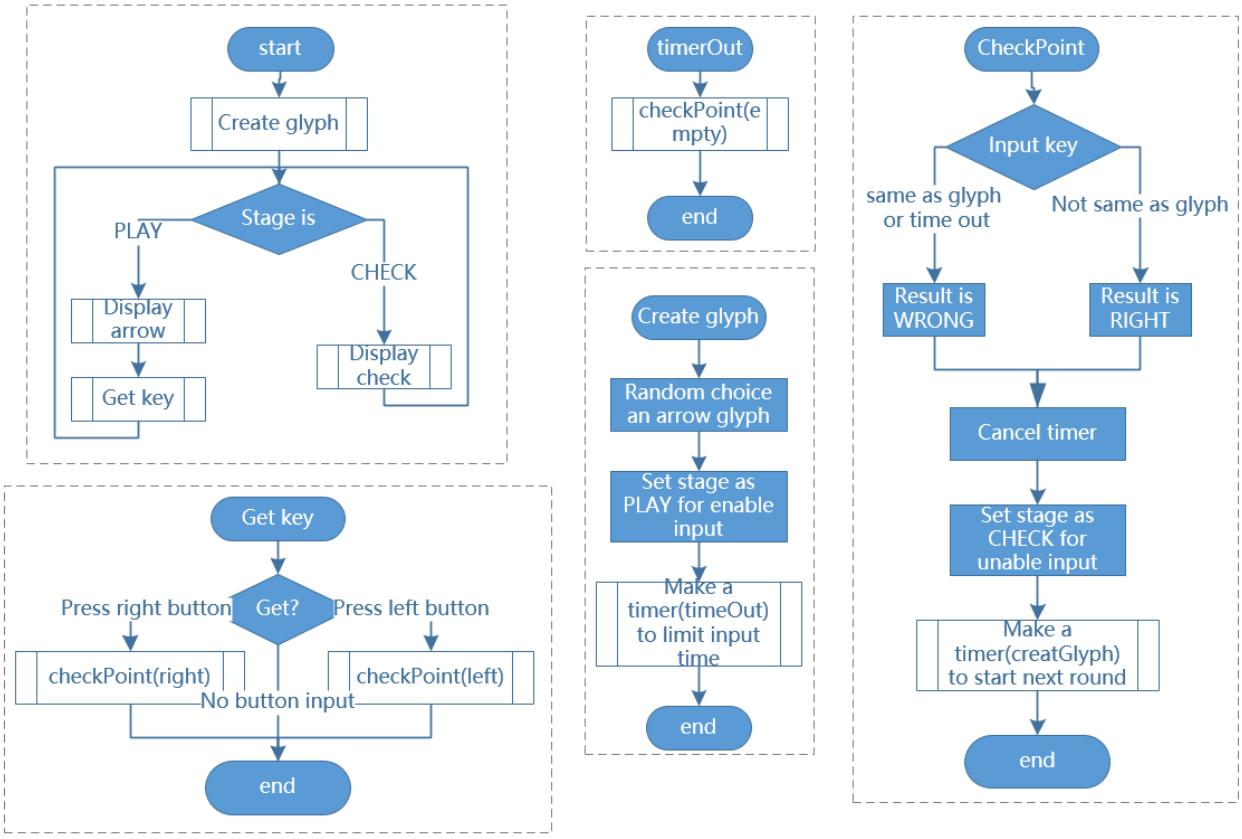
(前のページからの続き)

```
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])

def destroy():
    global timer1
    GPIO.cleanup()
    timerPlay.cancel()  # cancel the timer
    timerCheck.cancel()

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

コードの 説明

1.1.6 LED ドットマトリックス に基づいて、このレッスンでは 2 ボタンを追加して面白いゲームデバイスを作成します。したがって、ドットマトリックスにあまり詳しくない場合は、*1.1.6 LED* **ドットマトリックス** 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、 タイマー 1 を生成する。
2. ドットマトリックスに対応する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、 タイマー 1 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、 タイマー 2 を生成する。
5. タイマー 2 が時間切れを想起させると、 ステップ 1 を再実行する。

```
def^^c2^^a0main():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0creatGlyph()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0while^^c2^^a0True:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0stage^^c2^^a0==^^c2^^a0
↪"PLAY":
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display(
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0getKey()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0stage^^c2^^a0==^^c2^^a0
↪"CHECK":
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0display(
```

main() には実行中のプロセス全体が含まれている。

プログラムが起動したら、最初に 1 回だけ createGlyph() を呼び出してから、ループを開始します。

ループ内：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ます。チェックモードの場合、表示されるのは「x」または「√」です。

```
arrow={
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0#"down"^^c2^^a0:[0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,
↪0xFF],
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0#"up":[0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF],
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0"right"^^c2^^a0:^^c2^^a0[0xFF,0xEF,0xDF,0x81,0xDF,0xEF,
↪0xFF,0xFF],
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0"left":[0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF]
}
check={
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0"wrong":[0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF],
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0"right":[0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF]
}
```

ここで、辞書 矢印を使用して、LED ドットマトリックス上で上、下、左、右の方向に矢印パターンを表示できます。

上下が付注され、必要に応じて解除できる。

辞書 チェックは、「×」と「√」の 2 つの画像を表示するために使用されます。

```
def^^c2^^a0display(glyphCode):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0for^^c2^^a0i^^c2^^a0in^^c2^^a0range(0,^^c2^^a08):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(glyphCode[i])
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0hc595_shift(0x80>>i)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(RCLK,
↪^^c2^^a0GPIO.HIGH)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0GPIO.output(RCLK,
↪^^c2^^a0GPIO.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^^c2^^a0creatGlyph():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0waypoint
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0stage
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0timerPlay
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint=random.choice(list(arrow.keys()))
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage^^c2^^a0=^^c2^^a0"PLAY"
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timerPlay^^c2^^a0=^^c2^^a0threading.Timer(2.0,
```

(次のページに続く)

(前のページからの続き)

```
↪^^c2^^a0timeOut)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timerPlay.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「右」と「左」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^^c2^^a0checkPoint(inputKey):
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0waypoint
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0stage
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0global^^c2^^a0timerCheck
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0inputKey^^c2^^a0==^^c2^^a0"empty
↪"^^c2^^a0or^^c2^^a0inputKey^^c2^^a0==^^c2^^a0waypoint:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint^^c2^^a0=^^c2^^a0
↪"wrong"
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0else:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0waypoint^^c2^^a0=^^c2^^a0
↪"right"
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timerPlay.cancel()
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0stage^^c2^^a0=^^c2^^a0"CHECK"
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timerCheck^^c2^^a0=^^c2^^a0threading.Timer(1.0,
↪^^c2^^a0creatGlyph)
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0timerCheck.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か、矢印と同じ方向のボタンが押されている場合、 waypoint の割り当てられた値は wrong 、ドットマトリックスに x が表示されます。

それ以外の場合、ウェイポイントは正しく、「 √ 」が表示される。

これで、ステージは 「 CHECK 」 になり、1 秒のタイマー timerCheck を開始して、関数 creatGlyph() を 1 秒で呼び出します。

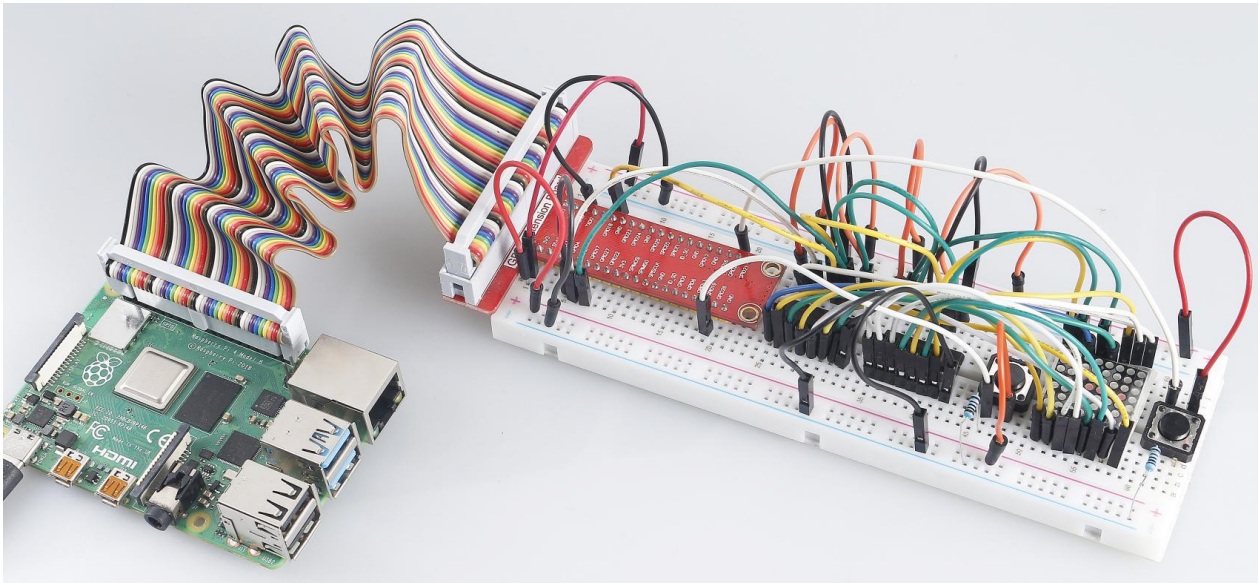
```
def^^c2^^a0timeOut():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0checkPoint("empty")
```

関数 timeout() で、 checkPoint() のパラメーターを「empty」に設定します。

```
def^^c2^^a0getKey():
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0if^^c2^^a0GPIO.
↪input(AButtonPin)==1^^c2^^a0and^^c2^^a0GPIO.input(BButtonPin)==0:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0checkPoint("right")
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0elif^^c2^^a0GPIO.
↪input(AButtonPin)==0^^c2^^a0and^^c2^^a0GPIO.input(BButtonPin)==1:
^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0^^c2^^a0checkPoint("left")
```

getKey() は、これら 2 つのボタンの状態を読み取り、右ボタンが押された場合、 checkPoint() のパラメーターは 右 です。左ボタンが押された場合、パラメータは 左 です。

現象画像



1.11 Python ビデオコース (Pi 5 用ではない)

この簡潔な Python ビデオコースで、ラズベリーパイの世界に飛び込みましょう。初心者から趣味人まで、どなたでも最適です。ラズベリーパイの機能紹介から始まり、Python プログラミングの基礎、GPIO ピン制御、センサー統合を含む一連のガイドを提供します。LED や LCD ディスプレイなどの主要部品を探索するインタラクティブなプロジェクトを通して進行します。このコースは、実践的な実行を通じて複雑な概念をアクセスしやすくすることで、プログラミングおよびハードウェアインターフェースのための必須スキルを装備します。デジタル世界でラズベリーパイの潜在能力を解き放つために、私たちに参加しましょう。

警告: これらのビデオは、Raphael Kit と呼ばれる私たちの別のキットの一部です。しかし、このスイートでもいくつかのコンテンツを使用することができますが、それによって特定のレッスン番号に中断が生じる場合があります。

プロジェクト

1.11.1 Video 1: 最初の起動

この動画では、基本的なセットアップから高度なプロジェクトまで、あらゆることを網羅する Raspberry Pi チュートリアルシリーズを紹介しています。回路、センサー、GPIO ピンに焦点を当てた徹底的な学習アプローチを強調しています。SunFounder がこのシリーズをスポンサーしており、包括的で手頃な価格の Raspberry Pi キットを提供しています。これからのレッスンには、Raspberry Pi のセットアップ、基本的な Linux の使用、Python の概要、GPIO ピンの使用が含まれます。また、必要なハードウェアについても議論しており、Raspberry Pi モデル、SD カード、電源、HDMI アダプターが含まれます。別の動画では、Raspberry Pi デスクトップインターフェースのカスタマイズに焦点を当て、パネル構成、アプレット管理、テーマ設定、トラブルシューティングのヒントをカバーしています。

1. 体系的な教育リソースへの需要に応えるための Raspberry Pi チュートリアルシリーズの紹介。
2. プロジェクトベースの学習ではなく、包括的な学習に重点を置く。
3. Arduino との差別化のため、回路、センサー、GPIO ピンに焦点を当てる。
4. SunFounder のスポンサーシップにより、包括的で手頃な価格の Raspberry Pi キットを提供。
5. 基本的なセットアップ、Linux の使用、Python、GPIO ピンを含む今後のレッスンの概要。
6. Raspberry Pi モデル、SD カード、電源、HDMI アダプターなど、必要なハードウェアの説明。
7. パネルレイアウト、アプレット、スペーサー、テーマなど、Raspberry Pi デスクトップインターフェースのカスタマイズオプション。
8. アイコンの欠如やパネル要素の誤配置など、一般的な問題に対するトラブルシューティングのヒント。

動画

1.11.2 Video 2: Raspberry Pi で Linux の基本とコマンドラインの基礎をマスターする

このチュートリアルでは、Raspberry Pi 上で Linux の基本とコマンドラインの基本を総合的に学ぶことができます。必要なコマンド、ファイルシステムのナビゲーション、ファイル管理、ソフトウェアのインストール、スーパーユーザーモード、ファイル操作、パス構造、ベストプラクティスについてカバーしています。ディレクトリのナビゲーション、ファイルの作成、名前変更、削除、ソフトウェアパッケージのインストール、特権コマンドの実行、ディレクトリの管理、再帰的操作の実行、nano テキストエディタを使用したファイルの編集、パス構造の理解、ターミナル内のファイルタイプの色分けの利用方法を学びます。

1. **ファイルシステムのナビゲーション**: cd、ls、pwd コマンドを使用したディレクトリのナビゲーションを学ぶ。
2. **ファイル管理**: touch、mv、rm コマンドを使用したファイルの作成、名前変更、削除を理解する。
3. **ソフトウェアインストール**: apt-get と sudo コマンドを使ったソフトウェアパッケージのインストールを探る。
4. **スーパーユーザーモード**: スーパーユーザーモードと、特権コマンドを実行するためのその適用についての洞察を得る。
5. **ディレクトリ管理**: mkdir と rmdir コマンドを使用してディレクトリを作成および削除する方法を学ぶ。
6. **再帰的操作**: rm -r コマンドを使用してファイルとディレクトリを再帰的に削除する方法を理解する。
7. **ファイル編集**: コマンドラインから直接ファイルを作成および編集するための nano テキストエディタを探る。
8. **パス構造**: 効果的なファイルシステムナビゲーションのための絶対および相対パス構造についての洞察を得る。
9. **Linux ファイルシステムのナビゲーション**: Linux ファイルシステム階層のナビゲーションの基本を理解する。
10. **色分け**: ターミナルでファイルタイプを簡単に識別するための色分けについて学ぶ。

ビデオ

1.11.3 Video 3: 一回のセッションで Python の基本をマスター

これらのチュートリアルは、Raspberry Pi のファン、初心者、そして重要なプログラミング概念を把握したい人向けにカスタマイズされた、Python プログラミングの包括的な紹介を提供します。ビデオは変数、配列、多次元配列、条件文、ループ、その他の Python の特徴など、さまざまなトピックをカバーしています。

1. **Python vs. Arduino**: Python のインタープリテッド言語と Arduino のコンパイル言語の違いを理解する。

2. **Python の構文と変数**: Python は変数を明示的に宣言せずに扱い、整数、浮動小数点数、文字列など様々なデータ型をサポートしています。
3. **配列**: Python の配列は柔軟で、異なるデータ型の複数の値を保持することができます。
4. **多次元配列**: データを行と列で整理するのに重要で、画像処理のようなプロジェクトに理想的です。
5. **配列要素へのアクセス**: インデックスをマスターして配列要素にアクセスする方法を学ぶ、開始はゼロからです。
6. **追加の Python 特徴**: 配列への追加や文字列内のインデクシングなど、Python プログラミングのレパートリーを拡大する洞察を得る。
7. **変数への値の割り当て**。
8. **ユーザー入力の取得**: 'input()' 関数を使用してユーザーに入力を促す。
9. **条件文**: 意思決定のための'if', 'elif', 'else' 文を実装する。
10. **算術演算**: 実行

ビデオ

1.11.4 Video 4: GPIO ピンの理解と使用

この包括的なチュートリアルは、Raspberry Pi の GPIO ピンの基礎と、Python プログラミングによるその使用法をカバーしています。視聴者は GPIO ピンのセットアップ方法、LED の制御、遅延の導入、使用後の GPIO リソースのクリーンアップ方法を学びます。さらに、実践的な応用を通じて学習を強化するための宿題が提供されます。

1. Raspberry Pi GPIO ピンとその番号付けシステムの紹介。
2. ブレッドボードの構成と使用法の説明。
3. LED 回路図の概要。
4. 回路図をブレッドボード上の物理的なコンポーネントに翻訳するデモンストレーション。
5. プロジェクト中にコンポーネントを整理して整頓するためのヒント。
6. LED の足の長さなど、コンポーネントの向きを正しく識別することの重要性。
7. `gpio.setmode()` および `gpio.setup()` 関数を使用した GPIO ピンのセットアップ。
8. `gpio.output()` 関数と状態値を格納する変数を使用した LED の制御。
9. `time.sleep()` 関数を使用した Python プログラム内での遅延の導入。
10. `gpio.cleanup()` 関数による適切なリソース管理。
11. 課題: ユーザー入力に基づいて LED を点滅させる Python プログラムを書く。

12. 視聴者へのヒント: 使用後の GPIO ピンをクリーンアップすることの重要性を強調し、学習の検証のために宿題の解答を共有する。

ビデオ

宿題の解答

1.11.5 Video 5: 二進数の理解

コンピューティングにおける二進数の重要性を探り、二進算の基本を学びます。LED と抵抗を使用して 5 桁の二進カウンターを作成する、Raspberry Pi の実践的な宿題に取り組みましょう。

1. **二進数の基礎**: コンピュータは二進数、すなわちゼロと一を使ってすべての計算を行います。
2. **データ表現**: 二進数は、数値からテキスト、画像までさまざまなデータタイプを符号化します。
3. **ハードウェアコンポーネント**: オン・オフスイッチがコンピューター内の二進表現の基礎を形成する方法を学びます。
4. **二進算**: 従来の繰り上げ方法を反映した二進加算のプロセスを理解します。
5. **Raspberry Pi 宿題**: LED と抵抗を使用して 5 桁の二進カウンターを作成する、実践的な課題に挑戦します。

ビデオ

宿題の解答

1.11.6 ビデオ 6: GPIO 入力、プルアップ抵抗とプルダウン抵抗の理解

Paul McWhorter のこの包括的なチュートリアルで、プルアップ抵抗とプルダウン抵抗の重要性と、Raspberry Pi での GPIO 入力の実践的な実装を学びます。

1. **GPIO 入力の基本**: GPIO ピンからのデジタル入力の読み取り方を学ぶ。
2. **回路セットアップ**: 押しボタン、抵抗、マイクロコントローラーでのセットアップデモ。
3. **GPIO ピンの設定**: 入力モードの設定と Python コードでの読み取り方。
4. **浮遊入力の防止**: プルアップ/プルダウン抵抗での安定化の方法。
5. **プルアップ&プルダウン抵抗**: スイッチの状態に応じた入力ピンの電圧接続方法。
6. **機能性**: スイッチの開閉による回路の振る舞いと論理レベルの影響。
7. **実装**: ブレッドボード上に回路をセットアップする手順。
8. **宿題**: LED 制御の実践と解決策の YouTube 共有を奨励。

ビデオ

宿題の解答

1.11.7 Video 7: ボタンスイッチからの GPIO 入力

Raspberry Pi の GPIO ピン内蔵のプルアップ抵抗を使用して、スイッチやプッシュボタンの GPIO 回路を簡素化する方法について学びます。これには、概念のデモンストレーション、回路セットアップ、LED をトグルする Python コード例が含まれます。

1. GPIO 回路におけるプルアップ抵抗とプルダウン抵抗を理解する。
2. プッシュボタン、プルアップ抵抗、LED を使用した回路セットアップをデモンストレーションする。
3. Raspberry Pi の GPIO ピン内蔵のプルアップ抵抗を使用する概念を紹介する。
4. GPIO ピン設定で内蔵プルアップ抵抗をアクティブにする方法を説明する。
5. プッシュボタンを使用して LED をトグルする Python コード例を提供する。内蔵プルアップ抵抗を利用します。
6. 実用的なアプリケーションでのトグルスイッチの重要性を強調し、既存の回路を使用してトグルスイッチを実装する宿題を割り当てる。

ビデオ

1.11.8 Video 8: ボタンをトグルスイッチとして使用し LED の制御を行う

Raspberry Pi と Python プログラミングを使用して、トグル LED 回路を作成する方法について学びます。

1. **回路セットアップ**: Raspberry Pi の GPIO ピンに接続された LED とプッシュボタンスイッチのセットアップ。
2. **Python プログラミング**: RPi.GPIO ライブラリを使用して GPIO ピンを制御し、プッシュボタンスイッチの状態を読み取る。
3. **トグルロジック**: プッシュボタンスイッチの状態に基づいて LED をトグルする Python 内のロジックを実装する。
4. **変数管理**: プッシュボタンスイッチと LED の現在および以前の状態を追跡するために変数を使用する。

ビデオ

1.11.9 Video 9: GPIO ピンを使用した PWM によるアナログ出力のシミュレーション

このビデオでは、Raspberry Pi の GPIO ピンを使用した PWM（パルス幅変調）の基本について案内し、LED の明るさ制御とリアルタイムな周波数調整をデモンストレーションします。

1. PWM とそのアナログ信号のシミュレーションにおける役割の紹介。
2. デューティサイクルの理解と PWM 信号におけるその重要性。
3. Raspberry Pi の GPIO ピンを使用した PWM による LED の明るさ制御の実践デモンストレーション。
4. オシロスコープを使用して PWM 信号を視覚化し、デューティサイクルと周波数を確認。
5. 周波数の説明と PWM 信号への影響。
6. Raspberry Pi のターミナルで Python コマンドを使用してデューティサイクルと周波数をリアルタイムに変更するデモンストレーション。
7. プッシュボタンを使用した正確な LED の明るさ制御を含む宿題。実践的な学習と実験を促進します。
8. LED、モーター、センサーなど様々なコンポーネントを制御する際の GPIO ピンの重要性。

ビデオ

1.11.10 Video 10: 2 つのプッシュボタンで調光可能な LED を作る

在本教程中，我将使用 Raspberry Pi 的 GPIO 引脚建一个可光 LED 路，以平滑的亮度控制。Paul McQuarter 逐演示了从路置到的程。

1. **回路セットアップ**: Raspberry Pi の GPIO ピンを使用して調光可能な LED 回路を構築し、制御のためにプッシュボタンを利用する方法の説明とデモンストレーション。
2. **Python コーディング**: PWM 制御と LED の明るさ制御のための指数スケールをそれぞれ実装する Python コードのウォークスルー。
3. **PWM の原理**: PWM デューティサイクルの理解と LED の明るさ制御への応用。
4. **指数スケール**: コーディングにおいてスムーズかつ線形な明るさ制御を実現するために指数スケールを利用する。
5. **対数概念**: 明るさ調整のための特定のボタン押下数に必要な定数を決定するために対数を適用する。
6. **非線形の明るさ調整**: スムーズな明るさの遷移のためのデューティサイクルの非線形変化を探る。
7. **同期問題**: PWM 周波数とカメラのスキャンレート間の同期問題に対処する。
8. **トラブルシューティングと洗練**: LED の調光機能を最適化するためのコードのデバッグと洗練技術。

ビデオ

1.11.11 Video 11: RGB LED の理解と使用方法

このチュートリアルでは、Raspberry Pi の GPIO ピンを使用して RGB LED を制御する方法について学びます。セットアップ、配線、そして Python コーディングの手順をカバーしています。

1. **RGB LED コンポーネント**: RGB LED の赤、緑、青のピンを理解する。
2. **GPIO セットアップ**: Raspberry Pi 上での GPIO セットアップにおける物理ピン番号システムについて学ぶ。
3. **配線ガイド**: 電流制限抵抗を含む、Raspberry Pi の GPIO ピンに RGB LED を配線するためのステップバイステップガイドに従う。
4. **Python コーディング**: GPIO ピンを使用して RGB LED の色を制御する Python コードのデモンストレーションを見る。
5. **トラブルシューティングのヒント**: コンポーネントと接続の効果的な取り扱いのための有用なヒントを得る。

ビデオ

1.11.12 Video 12: プッシュボタンによる RGB LED の制御

Raspberry Pi プラットフォームを使用して RGB LED を制御する方法を探究します。ハードウェアのセットアップ、コードの組織化、GPIO ピンの設定、カラーミキシング、調光可能な LED 制御、デバッグ、プロジェクトの拡張可能性についてカバーしています。

1. **ハードウェアセットアップ**: RGB LED、ボタン、GPIO 接続を設定します。
2. **コードの組織化**: 保守が容易になるようにコードを体系的に整理します。
3. **GPIO ピンセットアップ**: 入出力機能のための GPIO ピンを設定します。
4. **メインループの実装**: ボタンの状態を監視し、LED の状態を更新するメインループを作成します。
5. **デバッグとテスト**: デバッグ手順を通じて、ハードウェアとソフトウェアの機能を確実にします。
6. **RGB LED カラーミキシング**: 赤、緑、青を使用した色の組み合わせを実験します。
7. **調光可能な LED 制御**: 正確な色と明るさの調整のために LED を調光する方法を探究します。
8. **回路トラブルシューティング**: 接続不良などの回路問題を特定し、解決します。
9. **プログラミングロジック**: RGB LED の振る舞いを制御するためのコードロジックを開発します。
10. **プロジェクトの拡張**: 調光可能な LED 制御などの高度な機能でプロジェクトを拡張することを検討します。

ビデオ

1.11.13 Video 13: プッシュボタンで RGB LED の色を設定する

このビデオでは、Raspberry Pi を使ってボタンで RGB LED を制御する方法をデモンストレーションしています。ハードウェアのセットアップ、コーディング、トラブルシューティングを網羅しています。

1. Raspberry Pi GPIO に RGB LED とボタンを接続する方法を学びます。
2. デバッグが容易になるように、コードを体系的に整理する重要性を理解します。
3. 入力（ボタン）と出力（LED 制御）の両方の GPIO ピンをセットアップします。
4. ボタンの状態を連続的に読み取り、LED の状態を更新するメインループを実装します。
5. 適切な機能性を保証するために、デバッグとテストの手順に従います。
6. 赤、緑、青の値を調整して、色のスペクトラムを作り出すカラーミキシングを探求します。

ビデオ

1.11.14 Video 14: Raspberry Pi に Visual Python をインストールする

このビデオでは、Raspberry Pi を使用してボタンで RGB LED を制御する方法に焦点を当て、Visual Python のインストール、シンプルな 3D アニメーションの作成、および以前のレッスンからの LED 制御コードとの統合についてデモンストレーションしています。チュートリアルでは、Visual Python のインストール、既存の回路と RGB LED 制御のためのコードの統合、LED の状態を表す 3D アニメーションの作成、LED 制御に使用される PWM デューティサイクルと Visual Python 内の色表現の変換の理解、コード実行と実際の LED と仮想 LED のキャリブレーションの問題のトラブルシューティング、さらにアニメーションと LED 制御のカスタマイズと最適化の可能性の探求についてカバーしています。

1. Raspberry Pi に Visual Python をインストールし、環境を設定します。
2. 以前のレッスンからの既存の回路と RGB LED 制御のためのコードを統合します。
3. LED の状態を表すために Visual Python を使用して 3D アニメーションを作成します。
4. LED 制御に使用される PWM デューティサイクルと Visual Python での色表現の間の変換を理解します。
5. コード実行と実際の LED と仮想 LED 間のキャリブレーションに関する問題のトラブルシューティングを行います。
6. アニメーションと LED 制御のさらなるカスタマイズと最適化の可能性を探求します。

ビデオ

1.11.15 Video 15: ADC0834 を使用した Raspberry Pi でのアナログ入力

ADC0834 アナログ・デジタルコンバーターを使用して Raspberry Pi でアナログ入力を有効にする方法を学びます。ハードウェアのセットアップ、ライブラリのインストール、そして Python 開発のための Thonny IDE への移行を含みます。

1. アナログ入力と Raspberry Pi に ADC が必要な理由についての紹介。
2. ADC0834 チップのためのハードウェアセットアップと GPIO ピン接続。
3. コンポーネントのための適切な電源とグラウンド接続の確立。
4. Python 用 ADC0834 ライブラリのインストールと使用。
5. クリーンで効率的なコーディングのための現代的な IDE、Thonny への移行。
6. プログラミング前に接続をトラブルシューティングして確認するためのベストプラクティス。
7. ライブラリインストール: Raspberry Pi 用の ADC ライブラリをインストールします。
8. Python セットアップ: ライブラリコードを Python ファイルに保存し、正しいディレクトリに移動します。
9. GPIO の使用: GPIO ピンを使用してアナログ入力を読み取る Python プログラムをセットアップします。
10. アナログ読み取り: ポテンショメーターからアナログ値を連続的に読み取る Python スクリプトを書きます。
11. チャンネル切り替え: ADC チップの異なるチャンネル間での切り替えをデモンストレーションします。
12. 宿題: ポテンショメーターの位置で制御される調光可能な LED を作成します。

ビデオ

1.11.16 Video 16: ポテンショメーターで制御する調光可能な LED の作成

このビデオでは、Raspberry Pi、ADC-0834 アナログ・デジタルコンバーター、PWM（パルス幅変調）技術を使用して調光可能な LED を作成する方法をデモンストレーションします。ハードウェアのセットアップから Python でのコーディングまで、包括的なガイドを提供します。

1. **紹介**: SunFounder が提供するチュートリアルシリーズの概要。
2. **宿題の課題**: 前回のレッスンの課題、ポテンショメーターで制御される調光可能な LED を作成することのまとめ。
3. **ハードウェアセットアップ**: 必要なコンポーネント、Raspberry Pi、ADC-0834、ポテンショメーター、および電流制限抵抗付き LED の説明。
4. **GPIO ピンセットアップ**: GPIO ピンのセットアップと LED 制御のための PWM の初期化。

5. **Python コーディング**: Thonny を使用した Python でのコーディングのステップバイステップガイド。ライブラリのインポート、ピンセットアップ、デューティサイクル計算をカバーします。
6. **数学的計算**: ポテンショメーターの読み取り値を LED の明るさレベルにマッピングする際の数学的な説明。
7. **デモンストレーション**: 調光可能な LED 機能のライブデモンストレーション。
8. **次の宿題の課題**: RGB LED と複数のポテンショメーターを使用して色の混合を行う次の課題の発表。
9. **謝辞**: Patreon サポーターの認識と、将来のコンテンツをサポートするために寄付を検討するよう視聴者に促すこと。

ビデオ

1.11.17 Video 17: RGB LED で任意の色を混合する

このビデオは、Raspberry Pi と ADC0834 チップを使用して RGB LED の制御を構築し、マスターするプロセスについて説明しています。SunFounder が提供するチュートリアルシリーズの紹介から始まり、Raspberry Pi プロジェクトに焦点を当てています。前回のエピソードの課題は調光可能な LED の作成でしたが、今回はそれを RGB LED に置き換えることに焦点を当てています。

回路の概要には、RGB LED セットアップのための ADC0834 チップ、ポテンショメーター、抵抗の利用が含まれます。配線指示では、清潔な構築と GPIO ピンの割り当てのために直線ジャンパーワイヤーの使用の重要性が強調されます。RGB LED 制御のコーディングに進むと、Python と Thonny IDE が使用され、各色チャネルの PWM 制御のために LED 制御プログラムを変更します。

1. SunFounder が提供する Raspberry Pi プロジェクトのためのチュートリアルシリーズの紹介。
2. 前回のエピソードの課題は調光可能な LED の作成でしたが、今回は RGB LED に置き換えることが課題です。
3. 回路の概要には、RGB LED セットアップのための ADC0834 チップ、ポテンショメーター、抵抗が含まれます。
4. 清潔な構築と GPIO ピンの割り当てのために直線ジャンパーワイヤーを使用する重要性。
5. 各色チャネルの PWM 制御のために Python と Thonny IDE を使用して RGB LED 制御をコーディング。
6. RGB LED の制御のために Raspberry Pi 上で PWM のセットアップ。
7. アナログ入力値の読み取りのための ADC0834 の初期化。
8. LED の明るさを決定するために ADC0834 チップからアナログ値を読み取る。
9. 正確な PWM 制御のためにアナログ値をデューティサイクルに変換するデューティサイクル変換。
10. アナログ入力値に基づいて LED の明るさを動的に調整。

11. 効率的な LED 制御のためのコードの組織化とトラブルシューティング。

ビデオ

1.11.18 Video 18: Raspberry Pi でジョイスティックを使用する

SunFounder キットを使用して Raspberry Pi プロジェクトにジョイスティックを組み込む方法を学びましょう。セットアップ、配線、コーディング、テストについてカバーします。

1. **紹介**: Raspberry Pi プロジェクトでジョイスティックを使用する多様性を探ります。
2. **SunFounder キットデモンストレーション**: SunFounder キットに含まれるコンポーネントの概要。
3. **ジョイスティックの理解**: ジョイスティックのコンポーネントと機能性の説明。
4. **配線セットアップ**: Raspberry Pi の GPIO ピンにジョイスティックを配線するステップバイステップガイド。
5. **コーディングプロセス**: Python と RPi.GPIO ライブラリを使用して、ジョイスティックからのアナログ値を読み取り、ボタン押下を検出します。
6. **テストと検証**: テストを通じてジョイスティックとボタンの適切な機能性を確認します。

ビデオ

1.11.19 Video 19: Raspberry Pi からサーボモーターを簡単に制御する

Raspberry Pi を使用してサーボモーターを制御する方法を、Paul McQuarter がセットアップ、コーディング、テストプロセスを通じて案内します。

1. **ハードウェアセットアップ**: Paul は、サーボモーターを Raspberry Pi に正しく配線し、整列させる方法をデモンストレーションします。
2. **電源に関する考慮事項**: 大型のサーボを使用する際は Raspberry Pi を損傷から守るために外部電源を使用する重要性を説明します。
3. **PWM の基礎**: パルス幅変調（PWM）が PWM 信号のデューティサイクルを調整することにより、サーボモーターの位置をどのように制御するかを理解します。
4. **Python コード実装**: GPIO ピンを使用してサーボモーターをセットアップし、制御するための Python コードのウォークスルー。
5. **テストとキャリブレーション**: 異なる PWM パーセンテージを入力することにより、サーボモーターの動作範囲をテストし、キャリブレーションする方法を学びます。
6. **エラーハンドリング**: Python の try-except ブロックを使用して GPIO ピンの適切なクリーンアップを保証し、エラーを処理する方法を発見します。

ビデオ

1.11.20 Video 20: ポテンショメーターでサーボの位置を制御する

このビデオでは、ポテンショメーターと Raspberry Pi GPIO を使用してサーボ位置を制御する方法をデモンストレーションします。回路セットアップ、Python コーディング、ADC、PWM、数学的計算、エラーハンドリング、接続の重要性についての説明が含まれます。

1. **回路セットアップ**: Raspberry Pi 用 ADC0834 を使用した前回のレッスンのアナログ入力回路のレビュー。
2. **サーボセットアップ**: 回路内でのサーボの設定方法についての指示、配線と電源に関する考慮事項を含む。
3. **ADC と PWM を理解する**: アナログ・デジタルコンバーター（ADC）とパルス幅変調（PWM）の概念に慣れる。
4. **アナログ信号の読み取り**: Raspberry Pi に接続されたポテンショメーターからアナログ信号を読み取る方法を発見する。
5. **PWM パーセンテージの計算**: 取得したアナログ信号値に基づいて PWM パーセンテージを計算する式を学ぶ。
6. **サーボモーターの制御**: 計算した PWM パーセンテージを適用してサーボモーターの位置を制御するプロセスを探る。
7. **同期**: ポテンショメーターの位置とサーボモーターの位置の間の同期を確保し、正確な制御を行う。
8. **数学的計算**: ポテンショメーターの読み取りに基づいて PWM 値を決定するために使用される数学的計算の説明。
9. **エラーハンドリング**: プログラム終了と GPIO のクリーンアップのための try-except ブロックの実装。
10. **接続の重要性**: 正確な読み取りと出力のために信頼性の高い接続を確立することの重要性を理解する。

ビデオ

1.11.21 Video 21: HC-SR04 超音波センサーを使ったエコーロケーション

このチュートリアルでは、HC-SR04 センサーを使用して Raspberry Pi で超音波距離センサーを作成するプロセスを説明します。このビデオでは、エコーロケーションの原理を説明し、必要なコンポーネントを紹介し、配線セットアップをデモンストレーションし、コーディングプロセスをステップバイステップで説明します。正確な距離測定のためのコード実行における正確なタイミングの重要性を強調し、良いエンジニアリングの実践を促します。

1. **エコーロケーションの紹介**: コウモリやクジラに着想を得た、音を使って物体の位置を検出する方法。
2. **コンポーネントの概要**: HC-SR04 超音波センサーと Raspberry Pi への接続についての紹介。

3. **配線セットアップ**: 電源、グランド、トリガー、エコーのために HC-SR04 センサーを Raspberry Pi の GPIO ピンに接続する。
4. **コーディングプロセス**: 超音波パルスを生成し、センサーをトリガーし、エコーの帰還時間を測定する Python コードのウォークスルー。
5. **タイミングの考慮**: 正確な距離測定のための正確なタイミングの重要性。
6. **良いエンジニアリングの実践**: 実装前に計画とコードの理解を重視する。
7. **エコーピンの待機**: エコーピンが高くなるのを待つために while ループを使用する。
8. **開始時間の記録**: 測定開始をマークするために、エコーピンが高くなったときのシステム時間をキャプチャする。
9. **ピングの移動時間の測定**: エコーピンが高くなってから低くなるまでの時間差を決定することにより、ピングの移動時間を計算する。
10. **単位の変換**: 読みやすさのためにピングの移動時間を 1 万倍する。
11. **遅延の追加**: 複数のエコーを防ぐために各測定後に遅延を導入する。
12. **距離の計算**: 音速とピングの移動時間を使用して目標までの距離を計算する。

ビデオ

1.11.22 Video 22: HC-SR04 超音波センサーで距離を測定する

HC-SR04 超音波センサーと Raspberry Pi を使用して距離を測定する方法を学びます。配線、コーディング、単位変換を含みます。トラブルシューティングのヒントを探索し、距離測定の原理を理解し、実用的な応用を発見し、センサーを使用して音速を実験的に測定する宿題を受け取ります。

1. **紹介**: チュートリアルの概要とスポンサーシップの謝辞。
2. **前回の課題**: 前回のレッスンからの HC-SR04 センサー課題の復習。
3. **セットアップと配線**: Raspberry Pi への接続を含む配線セットアップの説明。
4. **コード説明**: ピンの移動時間と距離を測定するコードを理解する。
5. **トラブルシューティング**: コーディングエラーを特定して修正するためのヒント。
6. **距離測定の理解**: 超音波センサーの原理についての説明。
7. **補正係数**: 正確な距離測定のために必要な補正を認識する。
8. **単位変換**: 正確な測定のための単位変換についての議論。
9. **実用的な応用**: 超音波センサーの実世界での使用を探る。

10. **Fusion 360 の統合**: Raspberry Pi のスキルと 3D プリントの知識を組み合わせることを奨励する。
11. **宿題**: センサーを使用して音速を実験的に測定する課題。

ビデオ

1.11.23 Video 23: 超音波センサーを使用して音速を測定する

このステップバイステップのチュートリアルでは、Raspberry Pi と HC-SR04 超音波センサーを使用して音速を測定する方法を学びます。

1. 音速を測定するための HC-SR04 超音波センサーの使用についての紹介。
2. 距離、速度、時間の基本方程式を理解する。
3. Raspberry Pi と超音波センサーを使用したハードウェアセットアップ。
4. センサーを制御し、音速を計算する Python コードの作成。
5. 最終的な速度計算のための単位変換を行うための次元分析の実行。
6. 定規とターゲットを使用したセットアップと測定プロセスの実践的デモンストレーション。

ビデオ

1.11.24 Video 24: Raspberry Pi で PIR 動作センサーを使用する

Paul McQuarter によるこのチュートリアルでは、Raspberry Pi で PIR 動作センサーを使用する方法について学びます。彼はセンサーのコンポーネント、セットアップ、コーディング、動作検出プロジェクトのためのセンサーのテストについて説明します。

1. **PIR 動作センサーの紹介**: PIR 動作センサーが何であり、どのように機能するかを理解します。
2. **センサーコンポーネント**: 赤外線センサーやフレネルレンズを含む PIR 動作センサーの主要コンポーネントについて学びます。
3. **検出メカニズム**: センサーがどのように動作と熱を検出し、正確にトリガーするかを発見します。
4. **センサーセットアップ**: Paul が GPIO ピンを使用して PIR センサーを Raspberry Pi に接続する方法をデモンストレーションします。
5. **センサーのコーディング**: RPi.GPIO ライブラリを使用して動作を検出する簡単な Python スクリプトを書く方法を見ます。
6. **テストと微調整**: 最適なセンサー性能のためのテスト手順と微調整オプションについて学びます。

ビデオ

1.11.25 Video 25: DHT-11 センサーで温度と湿度を測定する

このチュートリアルでは、DHT11 温度湿度センサーを Raspberry Pi に接続し、Python を使用してデータを読み取る方法を学びます。

1. **紹介**: SunFounder が提供する Raspberry Pi チュートリアルシリーズにご参加ください。
2. **必要なコンポーネント**: Raspberry Pi、ジャンパーケーブル、DHT11 センサー、10k 抵抗を用意しましょう。
3. **配線指示**: センサーを損傷しないように、コンポーネントを正しく配線するための詳細な指示に従います。
4. **Python ライブラリのインストール**: DHT11 センサーとやり取りするために必要な Python ライブラリのインストール方法を学びます。
5. **コードの実装**: センサーから温度と湿度データを読み取り、ターミナルに表示する Python コードを書きます。
6. **コードの実行**: センサーの読み取り値を解釈するコードを実行してみましょう。
7. **今後のプロジェクト**: SunFounder キットからのコンポーネントを組み合わせた潜在的なプロジェクトを探求します。

ビデオ

1.11.26 Video 26: I2C を使用した LCD1602 LCD ディスプレイの使用方法

Paul McWhorter によるこの包括的なチュートリアルで、I2C 通信を使用して LCD 1602 ディスプレイを Raspberry Pi にセットアップし、接続する方法を学びます。

1. Raspberry Pi キットを提供してくれた SunFounder へのスポンサーシップの謝辞。
2. **プロジェクトの説明**: I2C を使用して Raspberry Pi に LCD 1602 ディスプレイを接続する。
3. **I2C 通信の利点**: 以前の方法と比べて 4 つの接続のみで配線が簡素化される。
4. **ハードウェアセットアップ**: ブレイクアウトボードを介して LCD ディスプレイを Raspberry Pi の GPIO ピンに接続する。
5. **Raspberry Pi での I2C の設定**: raspi-config を介して I2C インターフェイスを有効にする。

ビデオ

1.11.27 Video 27: LCD ディスプレイを使用した温度と湿度のシステムの構築

このチュートリアルでは、視聴者は Raspberry Pi、DHT11 センサー、LCD ディスプレイ、押しボタンを使用して温度と湿度を感知するデバイスを構築する方法を学びます。このビデオでは、プロジェクトに必要なハードウェアセットアップと Python コーディングについて詳細なガイダンスを提供します。

1. Raspberry Pi を使用した温度と湿度を感知するデバイスの構築についての紹介。
2. 必要なコンポーネントには、Raspberry Pi、DHT11 センサー、LCD ディスプレイ、押しボタン、メス-オスワイヤー、ジャンパーワイヤーが含まれます。
3. ハードウェアセットアップは、Raspberry Pi を SunFounder ブレイクアウトボードに接続し、その後、ジャンパーワイヤーを使用して DHT11 センサー、LCD ディスプレイ、押しボタンを接続することを含みます。
4. LCD1602、GPIO、DHT11、time などの必要なライブラリの設定。
5. GPIO モードの設定、DHT11 センサー、LCD ディスプレイ、押しボタンの初期化を含むコンポーネントの初期化。
6. GPIO 入力を通じてボタンの状態を読み取り、LCD ディスプレイ上で華氏と摂氏の読み取りを切り替える。

ビデオ

1.11.28 Video 28: アクティブ・パッシブブザーの理解と使用

この包括的なチュートリアルでは、Raspberry Pi プロジェクトにブザーを組み込み、温度センサーアラームシステムを作成する方法を学びます。ハードウェアセットアップ、プログラミング、テスト、コミュニティとのエンゲージメントについてカバーしています。

1. **アクティブブザーとパッシブブザー**: アクティブブザーとパッシブブザーの違いと特性を理解します。
2. **ハードウェアセットアップ**: ブザー、トランジスター、抵抗器、GPIO ピンを正しく接続するための回路図に従います。
3. **プログラミング**: ブザーを制御し、温度センサー機能を実装する Python コードを探索します。
4. **モードトランジション**: プログラムモードとモニターモードの間で切り替えるための押しボタンを実装します。
5. **コミュニティとのエンゲージメント**: プロジェクトのソリューションを YouTube で共有し、コミュニティと交流します。
6. **スキルの統合**: 以前に学んだコンセプトを適用し、包括的なプロジェクトを開発します。

ビデオ

1.11.29 Video 29: プログラム可能な温度感知アラームを構築する

このビデオでは、Raspberry Pi を使用してプログラム可能な温度監視システムとアラームを構築する方法をデモンストレーションします。ハードウェアのセットアップ、Python でのコーディング、デバッグ技術について説明します。このチュートリアルは、コードの組織と明瞭さを維持するための構造化されたプログラミングとトラブルシューティング方法を強調しています。

1. **ハードウェアセットアップ**: 必要なハードウェアコンポーネントを説明します。これには、Raspberry Pi、ブレイクアウトボード、ADC0834、DHT11 温度センサー、トグルボタン、アクティブブザー、トランジスターが含まれます。
2. **プロジェクトの概要**: 完成したデバイスとそのコンポーネント、機能をデモンストレーションします。
3. **回路図の概要**: プロジェクトの回路図について議論し、コンポーネント間の接続を強調します。
4. **プログラミング**: センサー値の読み取り、モード切り替え、温度閾値に基づくアラームのトリガリングを説明する Python コードを通じて説明します。
5. **デバッグ**: コードとハードウェア接続のエラーを特定し、修正するためのデバッグ技術をデモンストレーションします。
6. 温度感知のためのアナログからデジタルへの変換。
7. リアルタイムの温度表示のための LCD ディスプレイのインターフェース。
8. プログラミングモードとモニタリングモードの間で切り替えるためのボタンによるユーザーインタラクション。
9. 温度閾値に基づいてアラームをトリガーする。
10. コードの組織と明瞭さを維持するための構造化プログラミングアプローチ。

ビデオ

1.11.30 Video 30: Raspberry Pi で光を検出するフォトレジスターの使用

Raspberry Pi プロジェクトにフォトレジスターを組み込んで光の強度を測定し、明るさに基づいてアクションを制御する方法を学びましょう。

1. **フォトレジスターの紹介**: 光の強度に応じてフォトレジスターの抵抗がどのように変化するかを学びます。
2. **フォトレジスターの物理学**: フォトレジスターの動作に基づく固体物理学の原理を理解します。
3. **回路の構築**: フォトレジスターと負荷抵抗器を使用して電圧分割回路を作成する方法を発見します。
4. **Raspberry Pi のプログラミング**: ADC0834 モジュールを使用してフォトレジスターからアナログ値を読み取る Python コードを書きます。

5. **回路のテスト**: 回路が照明条件の変化にどのように反応するかを見て、抵抗と電圧の変化を示します。
6. **実用的な応用**: 自動照明システムなどのプロジェクトでフォトレジスターを使用する潜在的な用途を探索します。

ビデオ

1.11.31 Video 31: 暗闇での動きを検出するアラーム

このビデオでは、Raspberry Pi、動作センサー、光センサー、ブザーを使用して侵入者アラームシステムを構築する方法をデモンストレーションし、ハードウェアセットアップと Python コーディングの両方についてステップバイステップのガイダンスを提供します。

1. **プロジェクトへの紹介**: Raspberry Pi を使用して侵入者アラームシステムを構築する。
2. **使用されるコンポーネントの概要**: Raspberry Pi、PIR 動作センサー、光センサー（ADC-0834）、アクティブブザー、トランジスター（S8550）。
3. **ハードウェアセットアップの説明**: コンポーネントを配線し、ブザー、動作センサー、光センサーを Raspberry Pi の GPIO ピンに接続する方法。
4. **アラームシステムのコーディング**: センサー入力を読み取り、動作と暗闇を検出し、それに応じてブザーアラームをトリガーする Python コードの使用。
5. **システムのテスト**: 異なる照明条件での侵入者検出をシミュレートすることにより、アラームシステムの機能性をデモンストレーション。
6. **将来のプロジェクトの議論**: 加速度計、ジャイロスコープ、RFID、カメラなどの追加コンポーネントに関する今後のチュートリアルをプレビューし、さらなる Raspberry Pi プロジェクトについて議論。

ビデオ

1.11.32 Video 32: Raspberry Pi でキーパッドを理解し、使用する

Raspberry Pi プロジェクトにキーパッドを効率的に組み込む方法を学びましょう。入力読み取りに受動的な行列アドレッシングを使用します。

1. **紹介**: Raspberry Pi プロジェクトにキーパッドを統合する方法を学びます。
2. **受動的行列アドレッシング**: 効率的なキーパッド入力読み取りのための受動的アドレッシングの概念を理解します。
3. **配線セットアップ**: キーパッドの行と列を使用した配線セットアップを発見します。
4. **GPIO ピンの設定**: 行のための出力として、列のための入力として GPIO ピンをセットアップします。

5. **キーのラベリング**: キーパッド上のキーを簡単に参照するためにマトリックスを作成します。
6. **実装**: キーパッドをプロジェクトに組み込むためのコーディングプロセスに沿って進みます。

ビデオ

1.11.33 Video 33: Raspberry Pi プロジェクトにキーパッドを取り入れる方法

このビデオでは、Raspberry Pi を使用してキーパッド入力を読み取る方法をデモンストレーションし、回路セットアップ、Python コードの実装、そして「スティッキーボタン」問題に対処します。

1. **キーパッドスキャン方法**: 行と列を使用したキーパッドスキャンの概念を理解する。
2. **回路セットアップ**: 行と列のための GPIO ピンを設定し、内部プルダウン抵抗の使用を含む。
3. **Python コードの実装**: 行を活性化し、列を読み取ることによりキーパッド入力を読み取るための Python コードのステップバイステップのウォークスルー。
4. **「スティッキーボタン」問題**: 繰り返しボタンが押される問題の特定と解決。

ビデオ

1.11.34 Video 34: Raspberry Pi 用キーパッド読取り Python クラス作成

Raspberry Pi に接続されたキーパッドから入力を読み取る Python クラス作成について学びます。SunFounder 提供のチュートリアルシリーズの一環で、キーパッド接続のハードウェアセットアップと Python でのプログラミング方法に焦点を当てています。

1. **プロジェクト紹介**: Raspberry Pi を使用したキーパッド入力システム構築。
2. **ハードウェアセットアップ**: GPIO を利用したキーパッドの配線説明。
3. **Python プログラミング**: キーパッドからの入力を読み取るクラスのコード作成方法。
4. **入力読取り実装**: 行と列をスキャンして入力を読み取る機能の実装。
5. **問題解決**: 「粘着ボタン」問題の特定と対策。
6. **応用**: キーパッド機能を包含した Python ライブラリ作成、LCD 表示器やカメラとの統合可能性探索。

ビデオ

1.11.35 Video 35: キーパッドと LCD を使ったモバイル Raspberry Pi プロジェクトの作成

このチュートリアルでは、入力用のキーパッドと出力用の LCD ディスプレイを使用して Raspberry Pi アラームシステムを作成する方法、および効率的なプログラムフローのための Python コーディング技術について学びます。

1. **プロジェクト紹介**: Raspberry Pi アラームシステムの構築概要。
2. **コンポーネントセットアップ**: キーパッドと LCD ディスプレイコンポーネントの設定。
3. **Python コーディング**: キーパッドと LCD と対話する Python コードの記述。
4. **スレッディングコンセプト**: プログラムフローを改善するためのスレッディングの使用についての議論。
5. **アラームシステムの状態**: アラームシステムの異なる状態（アームド、ディスアームド、パスワード変更）の説明。
6. **宿題**: 異なる状態とパスワード検証のためのプログラムフローを実装する課題の割り当て。

ビデオ

1.11.36 Video 36: Raspberry Pi でキーパッド入力を監視する方法

Raspberry Pi を使ってキーパッドからの入力を読み取り、LCD で出力するアラームシステムを作る方法を学びます。配線方法、スレッディングによるマルチタスキング、コマンド処理、パスワード検証などについて説明します。実用化と展開の重要性も強調されています。

1. **携帯性と展開**: 実世界での応用に向けたプロジェクト。
2. **キーパッドと LCD セットアップ**: GPIO ピンを使った接続方法。
3. **Python コーディング**: キーパッド入力の読み取りと LCD への表示。
4. **スレッディングによる改善**: プログラムフローを向上させるスレッディングの利用。
5. **アラームシステムの状態管理**: 異なるアラームシステムの状態の説明。
6. **宿題**: 異なる状態のプログラムフローとパスワード検証の実装。

ビデオ

1.11.37 Video 37: キーパッドと LCD を使った Raspberry Pi アラームシステム

Raspberry Pi を使用して、キーパッドと LCD ディスプレイを組み込んだアラームシステムの構築方法を学びます。配線、Python コード、マルチタスキング、エラーハンドリングに重点を置き、実用化を目指します。

1. **携帯性と展開**: デスクトップ超えたアプリケーションへ。
2. **キーパッドと LCD の設定**: GPIO ピンを介した接続。
3. **スレッディング**: タスク同時実行のため。
4. **プログラムフロー**: アーム、ディスアーム、パスワード変更の実装。
5. **エラーハンドリング**: 正しいプログラム終了の保証。
6. **視聴者チャレンジ**: システムの機能強化提案。

ビデオ

1.11.38 Video 38: Raspberry Pi セキュリティシステム

Python コードと pygame ライブラリを使用して、携帯可能な Raspberry Pi アラームシステムに音響アラームを追加する方法を学びます。

1. **ポータブルプロジェクト開発**: Raspberry Pi を中心に展開可能なプロジェクトを構築する方法を教えるシリーズの一部です。
2. **音響アラームの追加**: pygame ライブラリを使用して MP3 アラーム音をインポートし再生することで、アラームシステムに音響警告を追加する方法をデモンストレーションします。
3. **実装詳細**: PIR センサーによって侵入者が検出されたときにアラーム音をロードして再生する手順をステップバイステップで提供します。
4. **カスタマイズオプション**: アラームシステムの多様性を高めるための異なるアラーム音を取得して保存する方法についてのガイダンスを提供します。
5. **宿題**: アラームシステムをさらに改善するために、複数のアラームオプションをプログラムする課題を視聴者に挑戦させます。

ビデオ

1.11.39 Video 39: Raspberry Pi セキュリティシステム

この Raspberry Pi チュートリアルでは、PIR 動作センサー、キーパッド、LCD ディスプレイを使用したホームセキュリティシステムのアラーム音をカスタマイズする方法を学びます。

1. SunFounder が提供する Raspberry Pi プロジェクトに焦点を当てたチュートリアルシリーズの紹介。
2. Raspberry Pi を使用して LCD、PIR 動作センサー、キーパッドの統合を含むホームセキュリティシステムを構築する概要。
3. アーミング、ディスアーミング、パスワード変更、アラーム音の選択のための Python コードの説明。
4. キーパッド、LCD、PIR センサーなどのハードウェアコンポーネントを接続して設定するステップバイステップガイド。
5. ユーザー入力に基づいた複数のアラーム音をプログラミングするデモンストレーションと、一般的なエラーのトラブルシューティングのヒント。

ビデオ

1.11.40 Video 40: Raspberry Pi セキュリティプロジェクトを Putty で遠隔操作

SSH と Putty を使用して Raspberry Pi に遠隔からアクセスし、制御する方法を学びます。これにより、デスクトップ環境に縛られずにプロジェクトの開発が可能になります。

1. **Raspberry Pi での SSH 有効化**: 遠隔アクセスを可能にするため、raspi-config を通じて SSH を有効にします。
2. **Raspberry Pi の IP アドレスの検出**: ifconfig を使用して IP アドレスを見つけ、一貫した遠隔アクセスのために静的 IP アドレスの割り当てを検討します。
3. **Putty のダウンロードとインストール**: PC に Putty をダウンロードしインストールして、Raspberry Pi への SSH 接続を確立します。
4. **Putty を使用した Raspberry Pi への接続**: Putty を使用して SSH 経由で遠隔から Raspberry Pi に接続し、資格情報でログインしてコマンドを遠隔実行します。
5. **Raspberry Pi の遠隔制御**: Putty を通じて Raspberry Pi 上のプログラムを遠隔で実行し、遠隔アクセスの実用性を示します。

ビデオ

1.11.41 Video 41: Raspberry Pi で VNC を使用したリモートデスクトップの設定方法

Windows PC から物理的な周辺機器を必要とせずに Raspberry Pi を制御できるように、リモートデスクトップアクセスを設定する方法を学びます。

1. SunFounder 提供の Raspberry Pi チュートリアルシリーズの紹介。
2. 物理的な周辺機器を避けて Raspberry Pi にリモートデスクトップアクセスする必要性の説明。
3. リモート端末アクセス用の PuTTY 経由の SSH の使用概要。
4. フルデスクトップインターフェース用の VNC（Virtual Network Computing）の紹介。
5. Raspberry Pi で VNC を有効にし、表示オプションを設定する手順。
6. リモート接続のために Raspberry Pi の IP アドレスを知ることの重要性。

ビデオ

1.11.42 Video 42: Raspberry Pi と PC 間の UDP を使ったデータ送信方法

Python と UDP（User Datagram Protocol）を使用して Raspberry Pi と PC 間のクライアントサーバー通信を設定する方法をデモします。サーバーとクライアントのプログラム作成、ソケットの初期化とバインド、データ交換、基本機能の実装について説明します。

1. **クライアントサーバー通信の紹介**: リモート接続のクリーンアップの必要性を理解する。
2. **サーバープログラムの作成**: Raspberry Pi で Python のソケットライブラリを使用してサーバープログラムを作成する手順。
3. **データ交換**: PC と Raspberry Pi 間でのデータ送受信のプロセスを探る。
4. **クライアントプログラムの作成**: Raspberry Pi サーバーと通信する PC 側のクライアントプログラムの作成。
5. **基本機能の実装**: カウンター操作の簡単な例で双方向通信をデモする。

ビデオ

1.11.43 Video 43: Raspberry Pi を使ったシンプルなクライアントサーバーシステムの構築

Raspberry Pi をサーバーとして、PC クライアントからのコマンド受信時に温度と湿度の測定を行うクライアントサーバー通信システムの確立方法を学びます。

1. **Raspberry Pi サーバーの設定**: GPIO ピンの設定、UDP サーバーソケットの作成。
2. **クライアント要求の処理**: コマンドを継続的にリスニングし、データのデコードとパース。

3. **センサー読み取り**: DHT11 センサーからの温度と湿度の読み取り。
4. **データ交換**: センサーデータをクライアントに送り返し、表示させる。
5. **クライアント側の実装**: クライアントソケットの確立、コマンドの送信、データの受信と処理。

ビデオ

1.11.44 Video 44: Raspberry Pi で高度なクライアント・サーバー接続を構築

Raspberry Pi を使い、PC からのコマンドに応じて温湿度データを送受信するクライアント・サーバーシステムの設定方法を解説します。エラーハンドリングやデータ伝送も扱います。

1. **Raspberry Pi サーバー設定**: UDP サーバーソケットの作成。
2. **クライアントリクエスト処理**: コマンドのリスニングとデータ解析。
3. **センサー読取り**: DHT11 センサーからの温湿度データ取得。
4. **データ交換**: クライアントへのセンサーデータ送信。
5. **クライアント側実装**: コマンド送信とデータ受信処理。

ビデオ

1.11.45 ビデオ 69: ラズベリーパイで MFRC522 RFID モジュールとタグを使う方法

このチュートリアルでは、ラズベリーパイプロジェクトでのアクセス制御や出席追跡などに RFID モジュールを統合する方法を学びます。Python でのコーディング実践を含むステップバイステップの指示に従ってください。

1. **RFID モジュール入門**: RFID 技術がラズベリーパイプロジェクトをどう強化するか理解します。
2. **ハードウェアセットアップ**: MFRC522 RFID モジュールをラズベリーパイに接続する方法を学びます。
3. **ラズベリーパイで SPI セットアップ**: SPI を有効化し、ラズベリーパイと RFID モジュールの通信に必要なライブラリをインストールします。
4. **RFID モジュール操作**: Python コードで RFID リーダーオブジェクトを初期化し、カード ID を読み、ID にテキストデータを関連付ける方法を学びます。
5. **RFID タグの読み書き**: Python で RFID タグにデータを読み書きする実践的なデモを見ます。
6. **宿題**: 人の名前と好きな色を記録するシステムを RFID タグで作成し、プロジェクトへの RFID 統合に関する理解を深めます。

ビデオ

1.11.46 ビデオ 70: ラズベリーパイで MFRC522 RFID モジュールとタグを使う方法

このビデオでは、ラズベリーパイを使用して RFID/NFC カードをプログラミングするための上級技術について説明します。データバンドル、ユーザー入力の処理、ハードウェアとのインタラクション、エラーチェックなどのトピックをカバーしています。このチュートリアルは、RFID/NFC カードプログラミングプロジェクトにおける管理機能の実装に向けた、効率的なデータ管理、ユーザープロンプト、デバッグ技術に関する洞察を提供します。

1. RC522 RFID モジュールと RGB LED のセットアップ。
2. 効率的なデータ管理のための辞書の活用。
3. ユーザープロンプトと応答の実装。
4. 新しい RFID/NFC カードをプログラミングするための管理機能。
5. データバンドル: コロン区切りを使用して効率的にデータをバンドルする。
6. ユーザー入力の適切なフォーマットとアポストロフィの処理。
7. NFC カードにパーソナライズされたデータを書き込む。
8. NFC カードから読み取ったデータに基づいて LED を制御する。
9. エラーチェックの重要性とエラーハンドリングの技術。
10. 宿題: プログラムの管理セクションで色入力のエラーチェックを実装し、辞書を動的に更新する。

ビデオ

1.12 Python で遊ぼう (すべての Pi モデル用)

1.12.1 「GPIO Zero」の確認

もし、Raspberry Pi 5 を使用して Python ユーザーであれば、「GPIO Zero」が提供する API を使用して GPIO をプログラムできます。

「GPIO Zero」は、Raspberry Pi の GPIO ピンを制御するためのモジュールです。このパッケージは、Raspberry Pi 上の GPIO を制御するための使いやすいクラスや関数の範囲を提供します。詳細や例については、こちらを参照してください：https://gpiozero.readthedocs.io/en/latest/。

GPIO Zero がインストールされているかどうかをテストするには、Python で次のコマンドを入力してください：

```
python
```

```
pi@raspberrypi:~ $ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python CLI で、 `import gpiozero` と入力します。エラーメッセージが表示されなければ、GPIO Zero がインストールされていることを意味します。

```
import gpiozero
```

```
pi@raspberrypi:~ $ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import gpiozero
>>>
```

Python CLI を終了する場合は、次のコマンドを入力してください：

```
exit()
```

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

1.12.2 1 出力

1.1 ディスプレイ

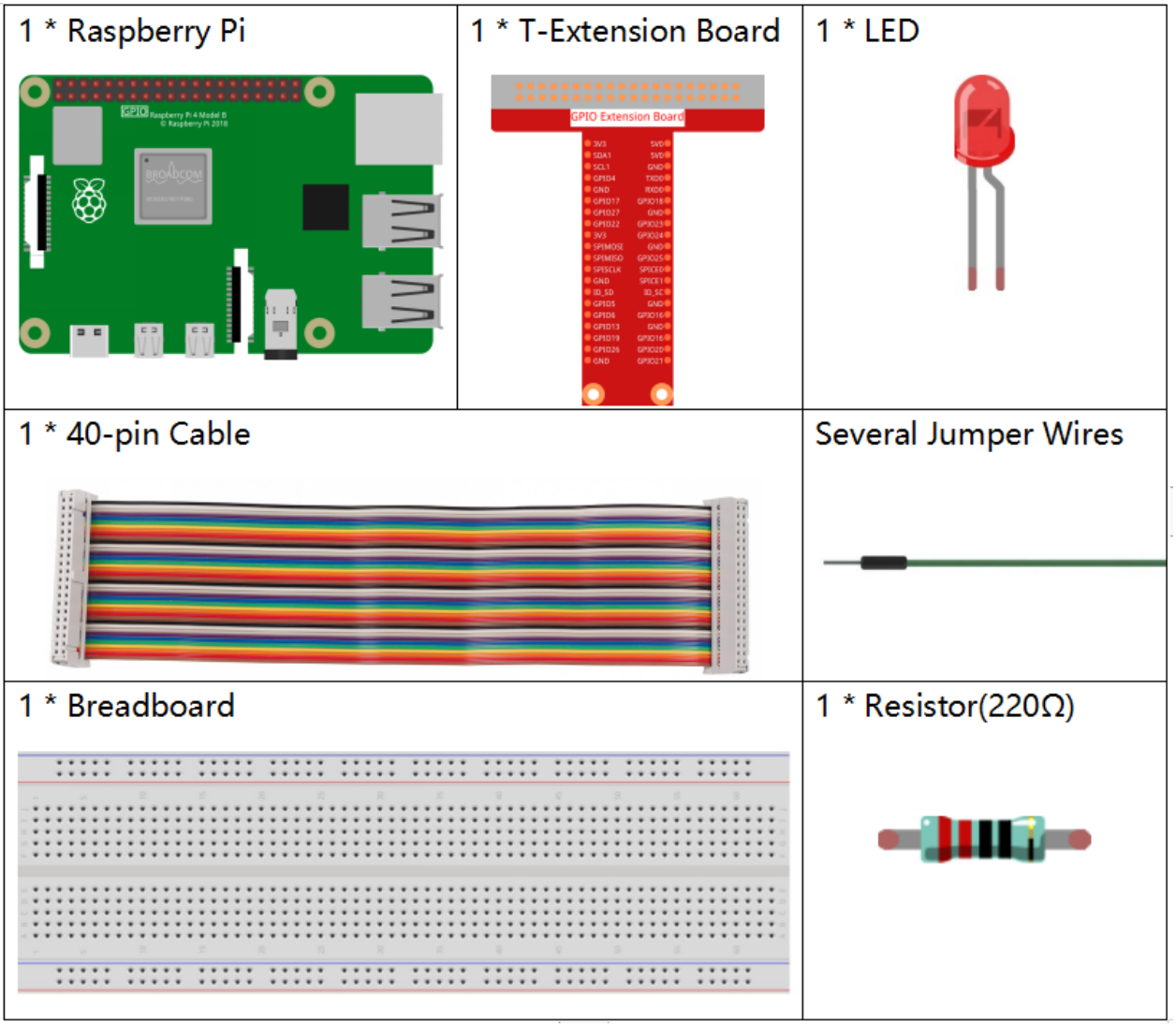
1.1.1 点滅する LED

概要

このプロジェクトでは、プログラムによって点滅する LED を作成する方法を学びます。設定により、LED はさまざまな興味深い現象を生成できます。さあ、始めましょう。

必要なコンポーネント

このプロジェクトには、次のコンポーネントが必要です。



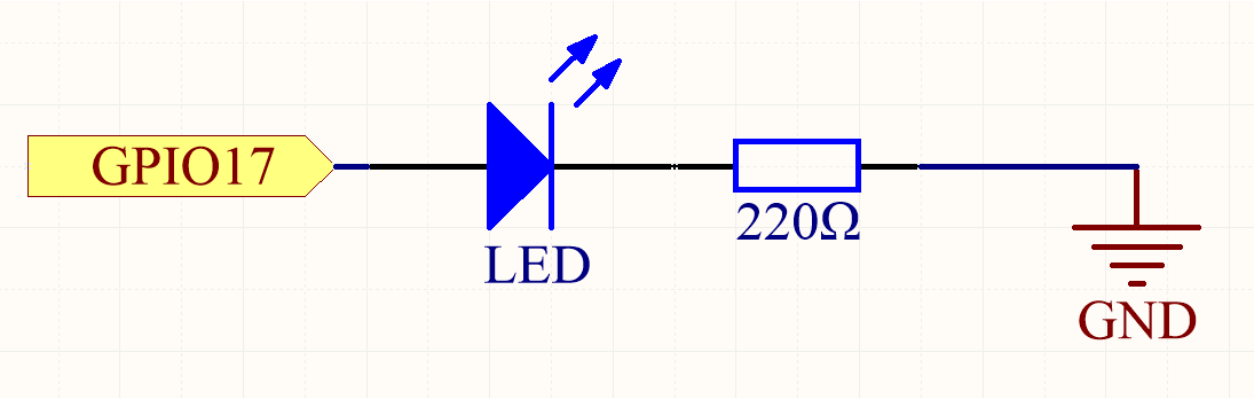
回路図

この実験では、Raspberry Pi の GPIO ピン 17 を LED の陽極（長いピン）に接続し、LED の陰極（短いピン）を抵抗に接続し、抵抗のもう一方の端を Raspberry Pi の GND ピンに接続します。LED を点灯させるには、GPIO17 を高電圧（3.3V）に設定する必要があります。これはプログラムによって実現できます。

注釈: **Pin11** は、Raspberry Pi の左から右への 11 番目のピンを指します。それに対応する **BCM** ピン番号は以下の表に示されています。

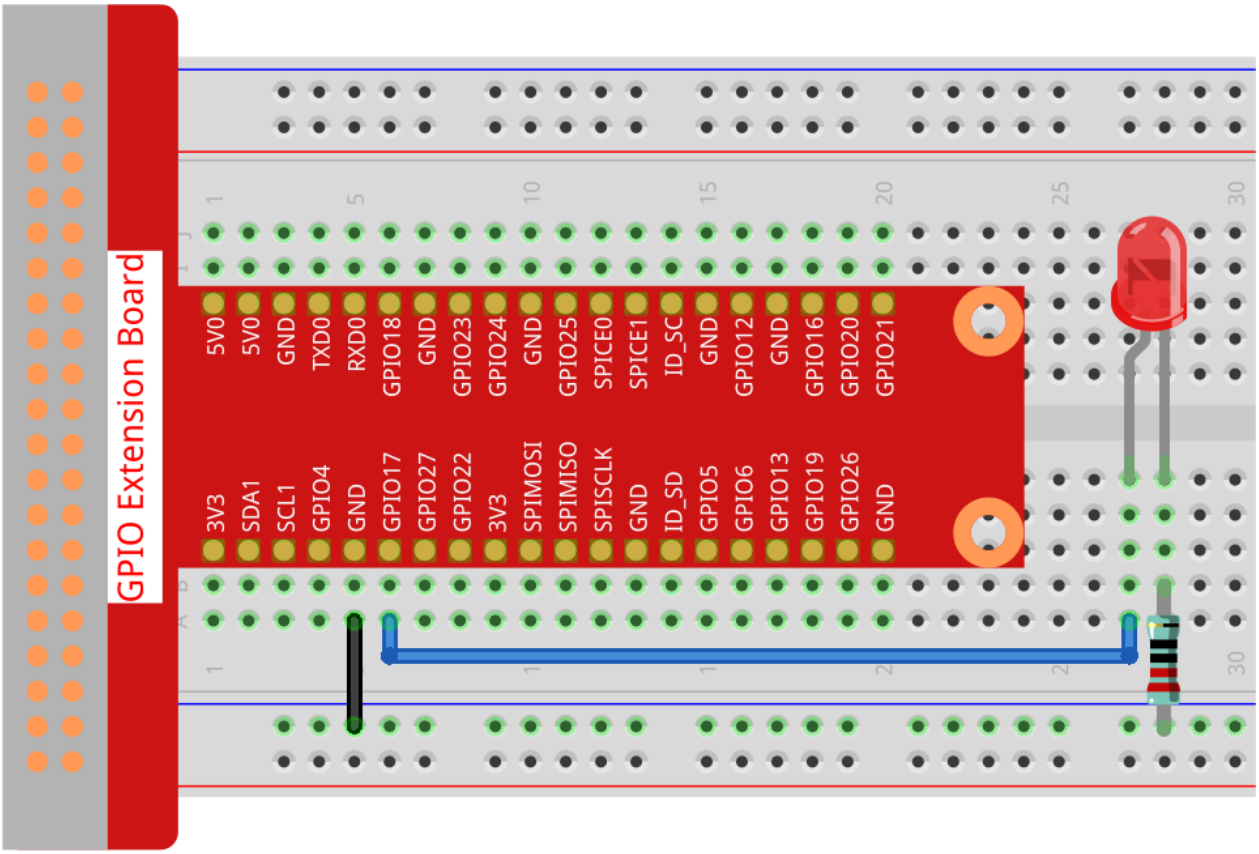
Python 言語関連の内容の中で、BCM 17 は以下の表の BCM 列で 17 になります。同時に、それは Raspberry Pi の 11 番目のピンであり、Pin 11 と同じです。

T-Board Name	physical	BCM
GPIO17	Pin 11	17



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動し、実行します。

1. 画面を使用する場合、以下の手順をお勧めします。

「1.1.1_BlinkingLed_zero.py」を見つけてダブルクリックして開きます。これでファイルが開きます。

ウィンドウで **Run -> Run Module** をクリックすると、次のコンテンツが表示されます。

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、コードに戻ります。コードを変更した場合は、**Run Module (F5)** をクリックする前に保存する必要があります。その後、結果を確認できます。

2. 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します：

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

注釈: この実験のコードのパスに cd を使用してディレクトリを変更します。

ステップ 3: コードを実行します。

```
sudo python3 1.1.1_BlinkingLed_zero.py
```

注釈: ここで、sudo - superuser do、python は Python でファイルを実行することを意味します。

コードを実行した後、LED が点滅するのを見ることができます。

ステップ 4: コードファイル「1.1.1_BlinkingLed_zero.py」を編集したい場合、「Ctrl + C」を押してコードの実行を停止します。その後、次のコマンドを入力して「1.1.1_BlinkingLed_zero.py」を開きます：

```
nano 1.1.1_BlinkingLed_zero.py
```

注釈: nano はテキストエディタツールです。このツールを使用してコードファイル「1.1.1_BlinkingLed_zero.py」を開くためにこのコマンドが使用されます。

「Ctrl+X」を押して終了します。コードを変更した場合、変更を保存するかどうかを確認するプロンプトが表示されます。保存する場合は「Y」（保存）、保存しない場合は「N」（保存しない）を入力します。

その後、効果を確認するために再度「nano 1.1.1_BlinkingLed_zero.py」を入力します。

コード

以下はプログラムコードです：

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import LED
from time import sleep

# GPIO Zero ライブラリを使用して GPIO ピン 17 に接続された LED を初期化します。
led = LED(17)

try:
    # LED の状態を切り替えるための無限ループを開始します。
    while True:
        # LED をオンにしてコンソールにメッセージを表示します
        led.on()
        print('...LED ON')

        # LED を 0.5 秒間オンにしたまま待機します。
        sleep(0.5)

        # LED をオフにしてコンソールにメッセージを表示します
        led.off()
        print('LED OFF...')

        # LED を 0.5 秒間オフにしたまま待機します。
        sleep(0.5)

except KeyboardInterrupt:
    # キーボードインタラプト（Ctrl+C を押すなど）を優雅に処理してループを終了します。
    # GPIO Zero は終了時に GPIO 設定のクリーンアップを自動的に処理します。
    pass
```

コードの説明

1. システムがこれを検出すると、env 設定で python のインストールパスを検索し、対応するインタープリタを呼び出して操作を完了します。これは、ユーザーが python を /usr/bin のデフォルトパスにインストールしていない場合でも、操作を行うためのものです。

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

2. これらの行は必要なクラスと関数をインポートします。 LED は LED 制御用の gpiozero から、 sleep は遅延用の time からです。

```
from gpiozero import LED
from time import sleep
```

3. この行は GPIO ピン 17 に接続された LED オブジェクトを作成します。

```
# GPIO Zero ライブラリを使用して GPIO ピン 17 に接続された LED を初期化します。
led = LED(17)
```

4. while True: を使用して無限ループを開始します。ループ内では、LED がオンになり（ led.on() ）、メッセージが表示されます。プログラムは 0.5 秒間一時停止します（ sleep(0.5) ）。その後、LED がオフになり（ led.off() ）、別のメッセージが表示され、プログラムは再び 0.5 秒間一時停止します。

```
try:
    # LED の状態を切り替える無限ループを開始します。
    while True:
        # LED をオンにしてコンソールにメッセージを表示します。
        led.on()
        print('...LED ON')

        # LED がオンの状態で 0.5 秒間待ちます。
        sleep(0.5)

        # LED をオフにしてコンソールにメッセージを表示します。
        led.off()
        print('LED OFF...')

        # LED がオフの状態で 0.5 秒間待ちます。
        sleep(0.5)
```

5. except ブロックは KeyboardInterrupt （Ctrl+C を押すなど）をキャッチし、ループを安全に終了します。ここで pass ステートメントは、割り込み時に特定のアクションを実行しないことを示すプレースホルダーとして使用されます。

```
except KeyboardInterrupt:
    # キーボードの割り込み（Ctrl+C）を安全に処理し、ループを中断します。
```

(次のページに続く)

(前のページからの続き)

```
    # 終了時には GPIO Zero が自動的に GPIO の設定をクリーンアップします。
    pass
```

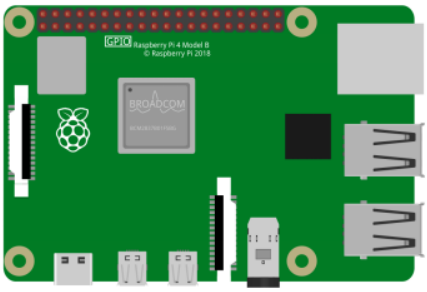
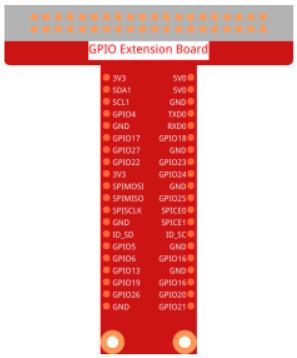
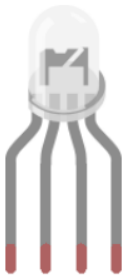
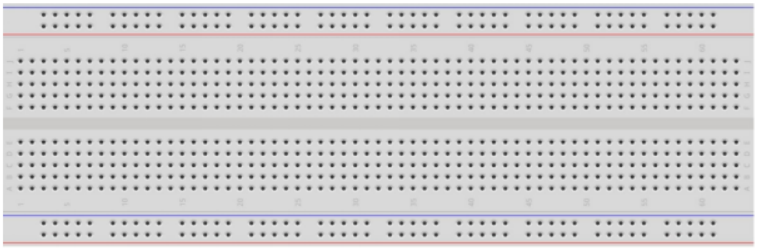
1.1.2 RGB LED

概要

このプロジェクトでは、RGB LED を制御してさまざまな色を点滅させます。

必要なコンポーネント

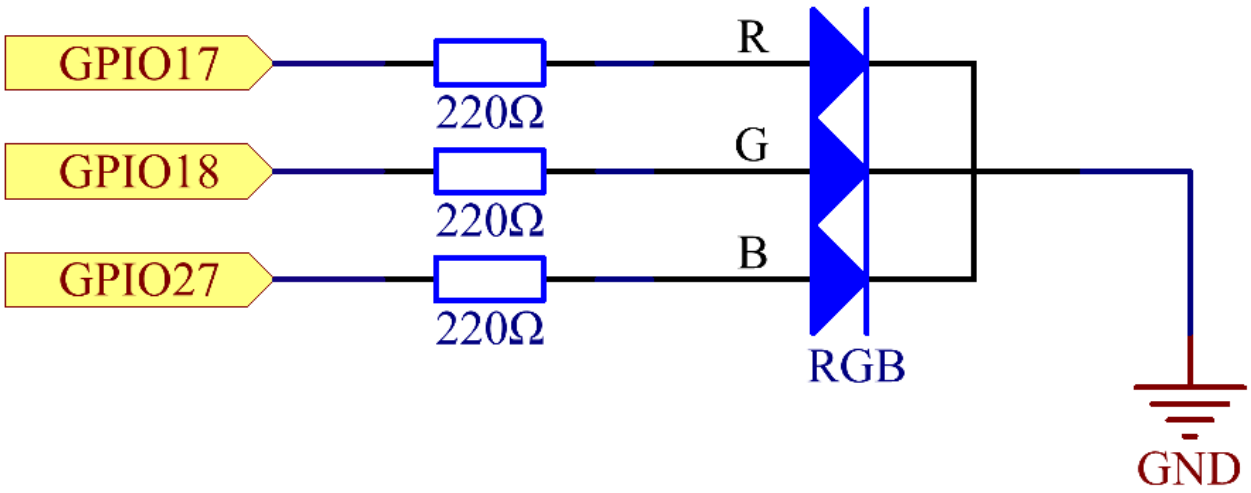
このプロジェクトには、次のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * RGB LED 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Resistor(220Ω)

回路図

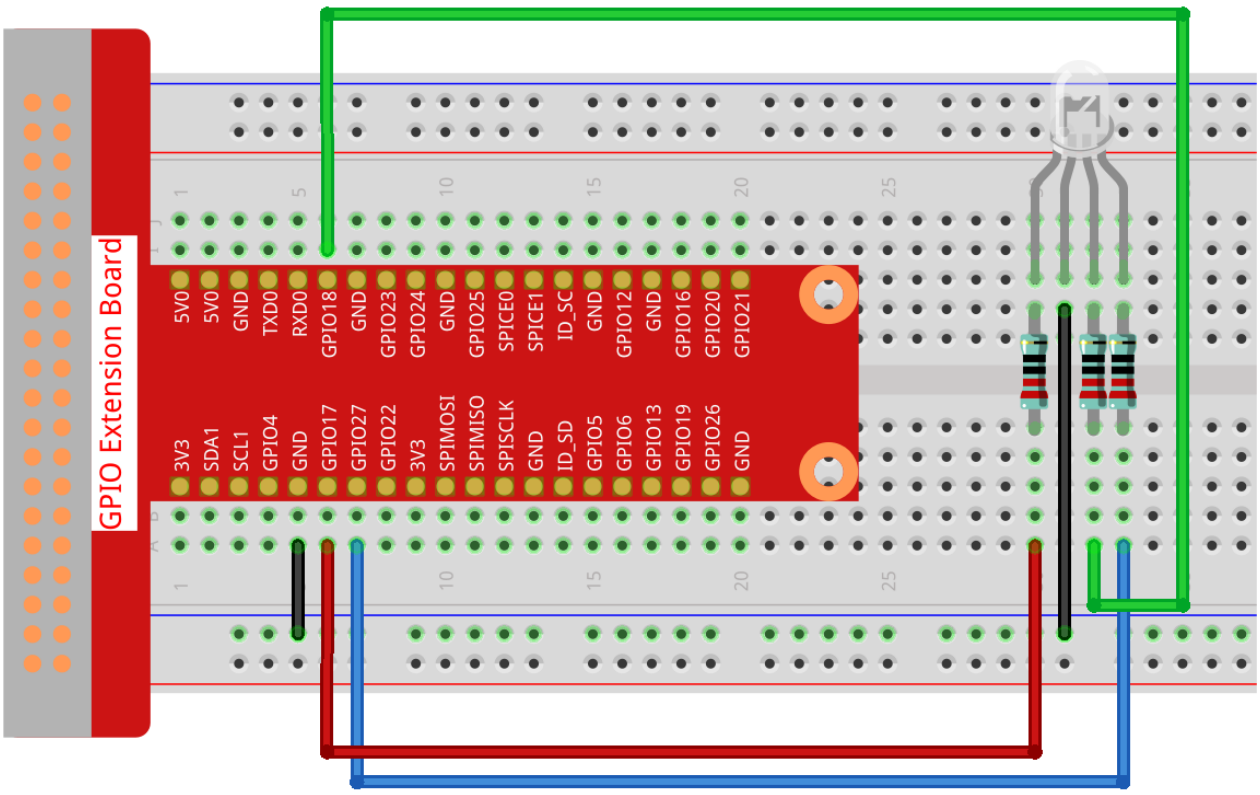
R、G、および B のピンを電流制限抵抗に接続した後、それぞれを GPIO17、GPIO18、GPIO27 に接続します。LED の最長のピン（GND）を Raspberry Pi の GND に接続します。これらの 3 つのピンに異なる PWM 値を与えると、RGB LED は異なる色を表示します。

T-Board Name	physical	BCM
GPIO17	Pin 11	17
GPIO18	Pin 12	18
GPIO27	Pin 13	27



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 1.1.2_rgbLed_zero.py
```

コードを実行した後、RGB LED が赤、緑、青、黄、ピンク、シアンを表示するのを確認できます。

コード

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import RGBLED
from time import sleep
```

(次のページに続く)

(前のページからの続き)

```
# RGB 形式（赤、緑、青）で RGB LED の色のリストを定義します。
# 各色成分は 0（オフ）から 1（フル強度）までの範囲です。
COLORS = [(1, 0, 0), (0, 1, 0), (0.2, 0.1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1)]

# RGB LED を初期化します。赤の成分を GPIO 17 に、緑を GPIO 18 に、青を GPIO 27 に接続します。
rgb_led = RGBLED(red=17, green=18, blue=27)

try:
    # 定義した色を連続してサイクルします。
    while True:
        for color in COLORS:
            # RGB LED を現在の色に設定します。
            rgb_led.color = color
            # 現在の色をコンソールに出力します。
            print(f"Color set to: {color}")
            # 次の色に切り替える前に 1 秒待ちます。
            sleep(1)

except KeyboardInterrupt:
    # キーボードインタラプト（Ctrl+C を押すなど）を優雅に処理してループを終了します。
    # GPIO のクリーンアップは、スクリプトの終了時に GPIO Zero によって自動的に管理されます。
    pass
```

コードの説明

1. これは RGB LED を制御するための gpiozero ライブラリから RGBLED クラスをインポートし、コード内で遅延を実装するための time ライブラリをインポートしています。

   ```
   #!/usr/bin/env python3
   from gpiozero import RGBLED
   from time import sleep

   # RGB LED の RGB 形式で異なる色を表すタプルを含む COLORS リストを定義します。
   # 各色成分は 0 から 1 までの範囲で、0 はオフ、1 は各色成分のフル強度を表します。
   ```

2. COLORS リストには RGB 形式で異なる色を表すタプルが含まれています。R、G、B ピンに異なるパルス幅変調（PWM）値を rgb_led.color 属性を介して割り当てることにより、LED はさまざまな色を生成できます。PWM 値は 0 から 1 までの範囲で、0 は無強度（オフ）を表し、各色成分のフル強度を表すのは 1 です。

3. 例えば、 rgb_led.color = (1, 0, 0) を設定すると、LED は赤に点灯します。これは赤成分にフル強度

を設定し、緑と青をオフにしています。同様に、これらの値の異なる組み合わせによって異なる色が表示されます。PWM を介した色の混合は、RGB LED で幅広い色の範囲を作成するための技術です。

```
COLORS = [(1, 0, 0), (0, 1, 0), (0.2, 0.1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1)]
```

4. RGB LED は、その赤、緑、青の成分をそれぞれ GPIO ピン 17、18、27 に接続して初期化されます。

```
# RGB LED を初期化します。赤の成分を GPIO 17 に、緑を GPIO 18 に、青を GPIO 27 に接続します。
rgb_led = RGBLED(red=17, green=18, blue=27)
```

5. while True: ループは COLORS で定義された色を連続してサイクルします。各色に対して、rgb_led.color = color は LED をその色に設定し、 sleep(1) は 1 秒間の一時停止を行います。

```
try:
    # 定義した色を連続してサイクルします。
    while True:
        for color in COLORS:
            # RGB LED を現在の色に設定します。
            rgb_led.color = color
            # 現在の色をコンソールに出力します。
            print(f"Color set to: {color}")
            # 次の色に切り替える前に 1 秒待ちます。
            sleep(1)
```

6. このセクションは KeyboardInterrupt （Ctrl+C を押すなど）を優雅に処理します。 pass ステートメントは、中断時に特定のアクションを指定しないことを示すプレースホルダとして使用され、GPIO Zero は GPIO のクリーンアップを自動的に処理します。

```
except KeyboardInterrupt:
    # キーボードインタラプト（Ctrl+C を押すなど）を優雅に処理してループを終了します。
    # GPIO のクリーンアップは、スクリプトの終了時に GPIO Zero によって自動的に管理されます。
    pass
```

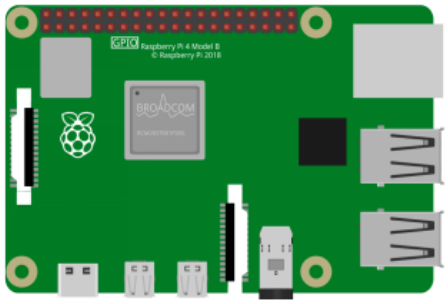
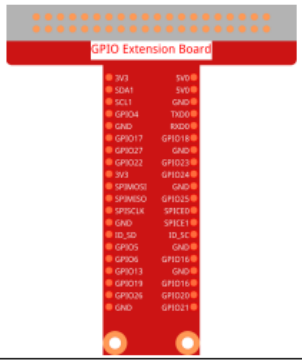
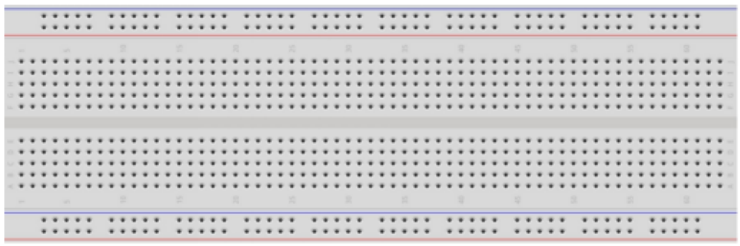
1.1.3 LED バーグラフ

概要

このプロジェクトでは、LED バーグラフ上のライトを順番に点灯させます。

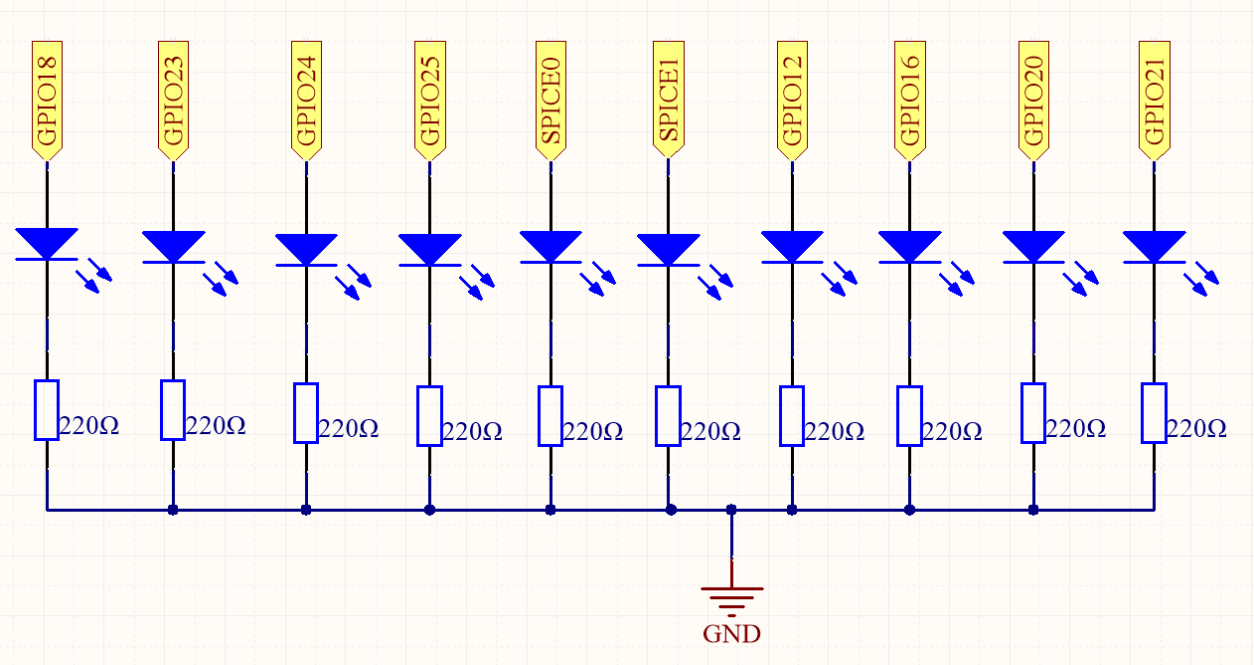
必要なコンポーネント

このプロジェクトには、次のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Bargraph
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		10 * Resistor(220Ω)

回路図

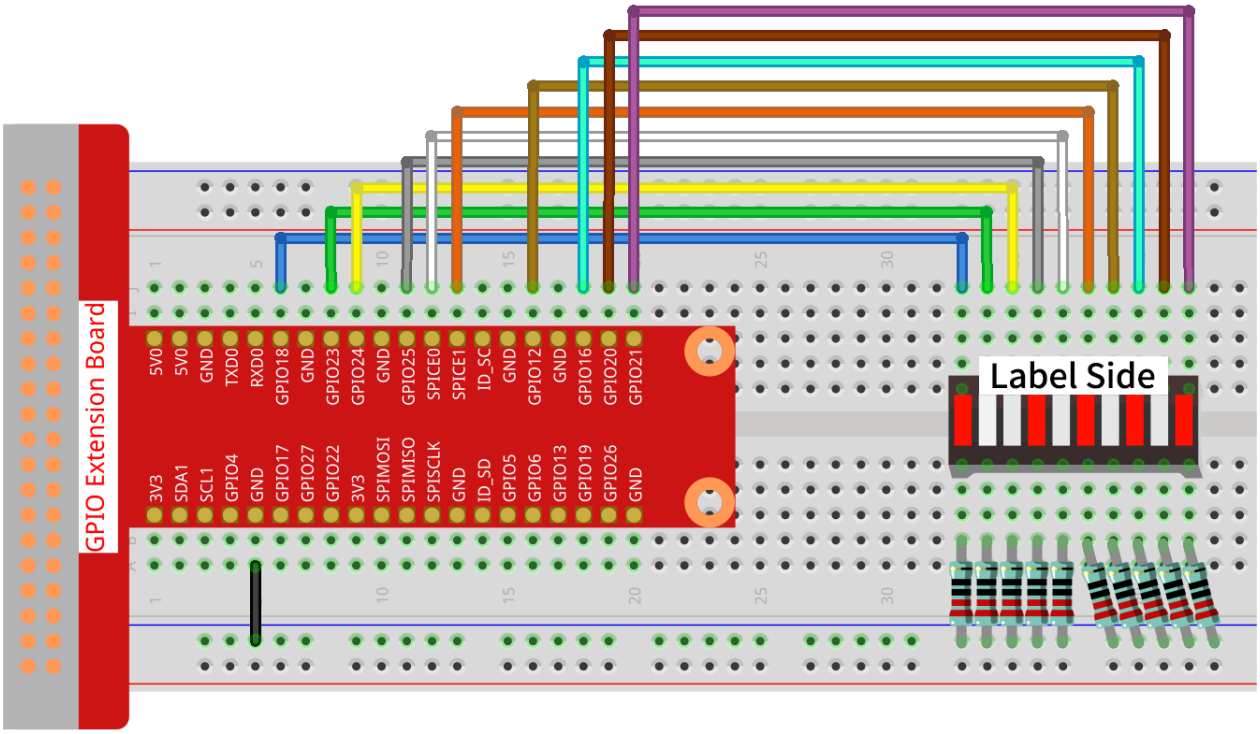
T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	Pin 26	11	7
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	22
GPIO21	Pin 40	29	21



実験手順

ステップ 1: 回路を組み立てます。

注釈: 接続する際に方向に注意してください。逆に接続すると点灯しません。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 1.1.3_LedBarGraph_zero.py
```

コードを実行した後、LED バーグラフ上の LED が定期的に点灯および消灯するのを確認できます。

コード

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import LED
from time import sleep

# LED が接続されている GPIO ピンを定義します
```

(次のページに続く)

(前のページからの続き)

```
led_pins = [18, 23, 24, 25, 8, 7, 12, 16, 20, 21]

# 各ピンに対応する LED オブジェクトを作成します
leds = [LED(pin) for pin in led_pins]

def odd_led_bar_graph():
    # 奇数番目の LED（インデックス 0, 2, 4 など）を順番に点灯させます。
    for i in range(5):
        j = i * 2  # 奇数のインデックスを計算
        leds[j].on()  # 奇数番目の LED を点灯
        sleep(0.3)    # 視覚効果のための遅延
        leds[j].off() # LED を消灯

def even_led_bar_graph():
    # 偶数番目の LED（インデックス 1, 3, 5 など）を順番に点灯させます。
    for i in range(5):
        j = i * 2 + 1  # 偶数のインデックスを計算
        leds[j].on()   # 偶数番目の LED を点灯
        sleep(0.3)     # 視覚効果のための遅延
        leds[j].off()  # LED を消灯

def all_led_bar_graph():
    # すべての LED を順番に点灯させます
    for led in leds:
        led.on()       # LED を点灯
        sleep(0.3)     # 視覚効果のための遅延
        led.off()      # LED を消灯

def turn_off_all_leds():
    # すべての LED を一斉に消灯させます
    for led in leds:
        led.off()

try:
    # LED パターンをサイクルするためのメインループ
    while True:
        odd_led_bar_graph()   # 奇数番目の LED をアクティブに
        sleep(0.3)            # パターン間の一時停止
        even_led_bar_graph()  # 偶数番目の LED をアクティブに
```

(次のページに続く)

(前のページからの続き)

```
        sleep(0.3)              # パターン間の一時停止
        all_led_bar_graph()     # すべての LED をアクティブに
        sleep(0.3)              # 再起動前の一時停止

except KeyboardInterrupt:
    # 中断（Ctrl+C）を処理する
    turn_off_all_leds()       # 終了時にすべての LED が消灯するようにします
    pass
```

コードの説明

1. これらの行は必要なクラスと関数をインポートします。LED 制御のための gpiozero の LED および遅延のための time の sleep です。

```
#!/usr/bin/env python3
from gpiozero import LED
from time import sleep
```

2. led_pins リストには GPIO ピン番号が含まれており、 leds は led_pins の各ピンに対応する LED オブジェクトのリストです。

```
# LED が接続されている GPIO ピンを定義します
led_pins = [18, 23, 24, 25, 8, 7, 12, 16, 20, 21]

# 各ピンに対応する LED オブジェクトを作成します
leds = [LED(pin) for pin in led_pins]
```

3. LED バーグラフの奇数番号の LED を順番に点灯させます。

```
def odd_led_bar_graph():
    # 奇数番号の LED（インデックス 0, 2, 4 など）を順番に点灯させます
    for i in range(5):
        j = i * 2  # 奇数のインデックスを計算します
        leds[j].on()  # 奇数番号の LED を点灯させます
        sleep(0.3)    # 視覚的な効果のための遅延
        leds[j].off() # LED を消灯させます
```

4. LED バーグラフの偶数番号の LED を順番に点灯させます。

```
def even_led_bar_graph():
    # 偶数番号の LED（インデックス 1, 3, 5 など）を順番に点灯させます
```

(次のページに続く)

(前のページからの続き)

```
    for i in range(5):
        j = i * 2 + 1  # 偶数のインデックスを計算します
        leds[j].on()   # 偶数番号の LED を点灯させます
        sleep(0.3)     # 視覚的な効果のための遅延
        leds[j].off()  # LED を消灯させます
```

5. LED バーグラフの LED を 1 つずつ順番に点灯させます。

```
def all_led_bar_graph():
    # すべての LED を順番に点灯させます
    for led in leds:
        led.on()       # LED を点灯させます
        sleep(0.3)     # 視覚的な効果のための遅延
        led.off()      # LED を消灯させます
```

6. while True ループは LED パターンを連続してサイクルします。 except ブロックは KeyboardInterrupt（Ctrl+C）を処理し、終了時にすべての LED が消灯することを確認します。

```
try:
    # LED パターンをサイクルするメインループ
    while True:
        odd_led_bar_graph()    # 奇数番号の LED をアクティブにする
        sleep(0.3)             # パターン間の一時停止
        even_led_bar_graph()   # 偶数番号の LED をアクティブにする
        sleep(0.3)             # パターン間の一時停止
        all_led_bar_graph()    # すべての LED をアクティブにする
        sleep(0.3)             # 再開前の一時停止

except KeyboardInterrupt:
    # インタラプト（Ctrl+C）を処理します
    turn_off_all_leds()      # 終了時にすべての LED が消灯していることを確認します
    pass
```

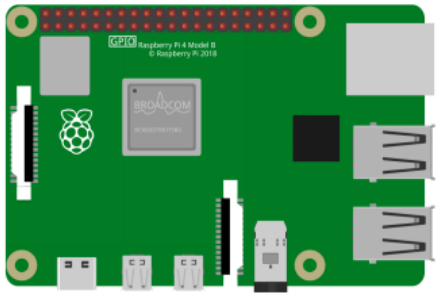
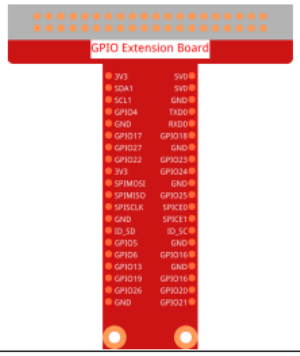
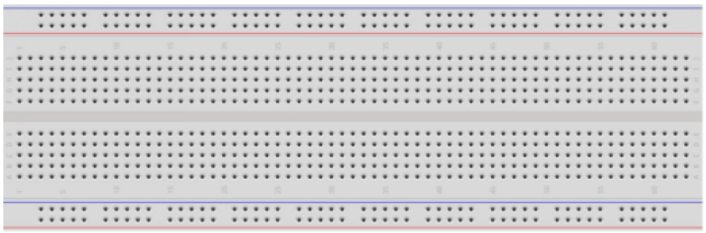
1.1.4 7 セグメントディスプレイ

概要

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

必要なコンポーネント

このプロジェクトには、次のコンポーネントが必要です。

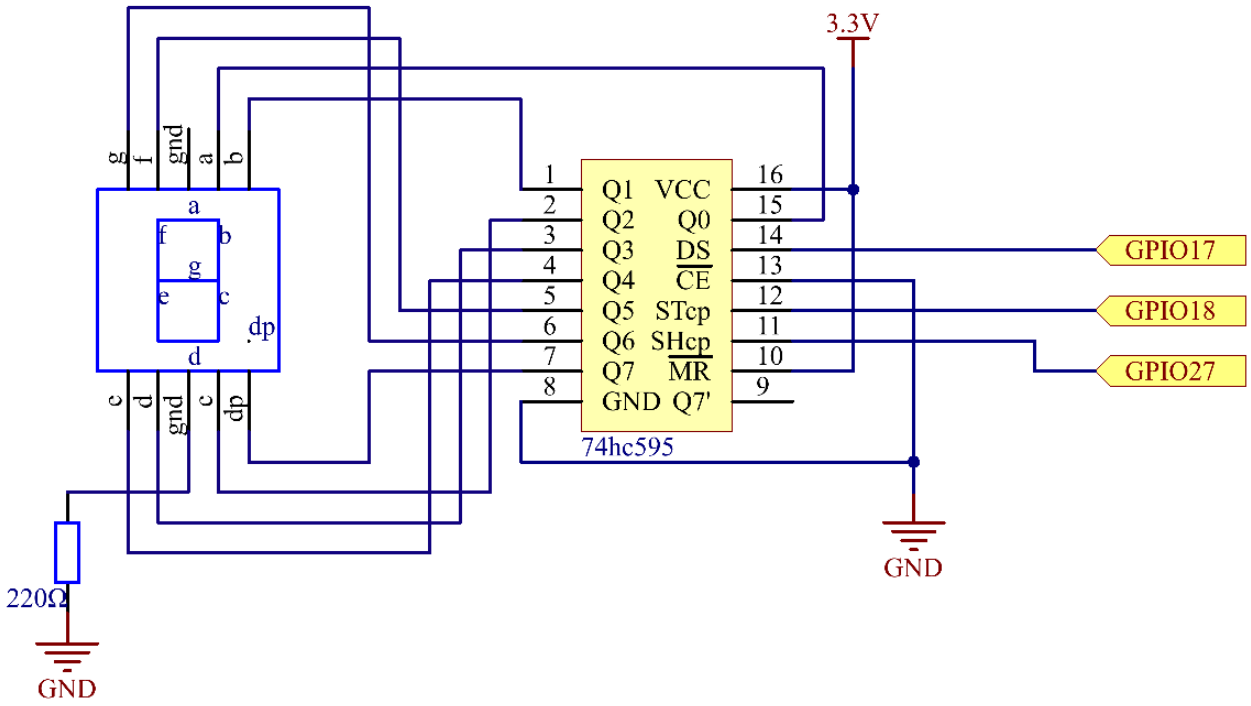
1 * Raspberry Pi 	1 * T-Extension Board 	1 * 7-segment display
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * 74HC595

回路図

74HC595 の ST_CP ピンを Raspberry Pi の GPIO18 に、SH_CP を GPIO27 に、DS を GPIO17 に接続し、LED セグメントディスプレイの 8 つのセグメントに並列出力ポートを接続します。

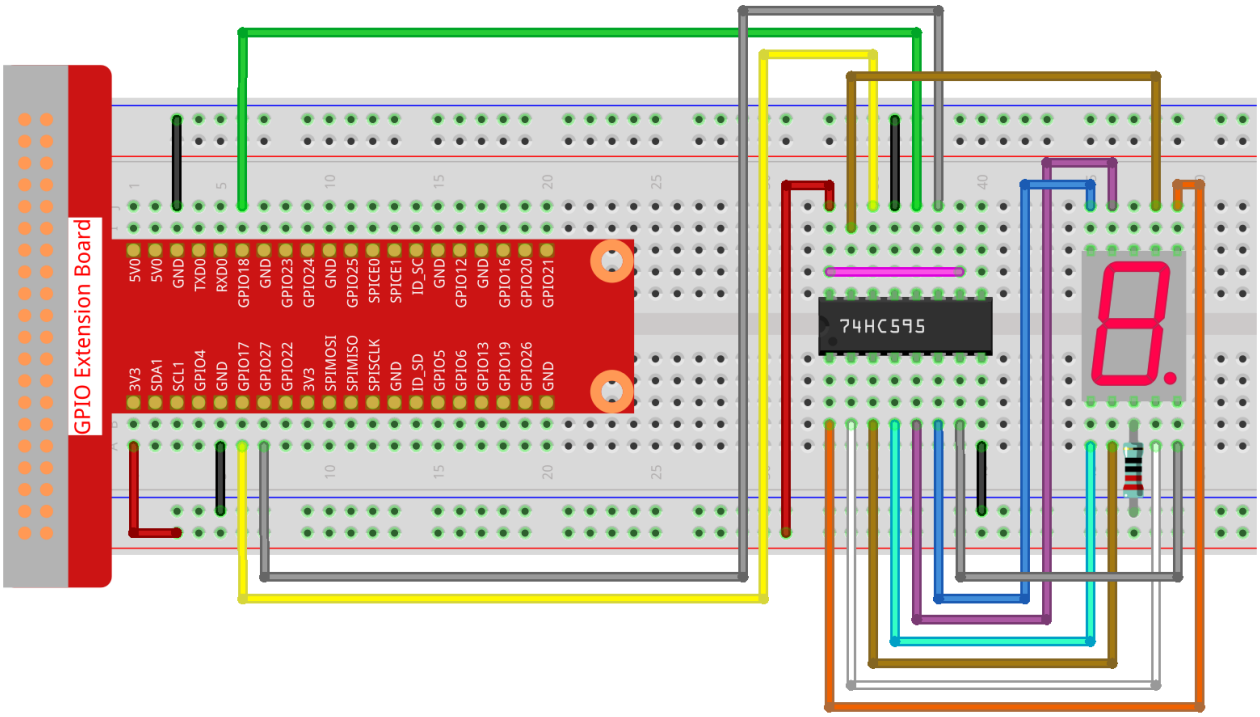
SH_CP（シフトレジスタのクロック入力）が立ち上がりエッジのときに DS ピンへデータを入力し、ST_CP（メモリのクロック入力）が立ち上がりエッジのときにメモリレジスタにデータを入力します。その後、Raspberry Pi の GPIO を介して SH_CP と ST_CP の状態を制御して、シリアルデータ入力を並列データ出力に変換し、Raspberry Pi の GPIO を節約してディスプレイを駆動できます。

T-Board Name	physical	BCM
GPIO17	Pin 11	17
GPIO18	Pin 12	18
GPIO27	Pin 13	27



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

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

コードを実行した後、7 セグメントディスプレイが 0 から 9、A から F まで表示されるのを確認できます。

コード

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。問題がないことを確認したら、修正されたコードをコピーするための Copy ボタンを使用し、ターミナルで「nano」コマンドを使用してソースコードを開き、コードを貼り付けることができます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep

# 74HC595 シフトレジスタに接続された GPIO ピン
SDI = OutputDevice(17)   # シリアルデータ入力
RCLK = OutputDevice(18)  # メモリクロック入力（レジスタクロック）
SRCLK = OutputDevice(27) # シフトレジスタクロック

# 7 セグメントディスプレイの共通陰極の 16 進数デジットコード
segCode = [
    0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
    0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71
]

def hc595_shift(data):
    # 74HC595 に 8 ビットのデータをシフト
    for bit in range(8):
        # データビットに基づいて SDI を高または低に設定
        SDI.value = 0x80 & (data << bit)
        # シフトレジスタクロックをトリガ
        SRCLK.on()
        sleep(0.001)
        SRCLK.off()
    # メモリクロックをトリガしてデータを出力にラッチ
    RCLK.on()
    sleep(0.001)
    RCLK.off()

def display_all_on():
    # すべてのセグメントをオンにする関数（共通陰極の 7 セグメントディスプレイ用）
    all_on_code = 0x3f
    hc595_shift(all_on_code)
    print("Displaying all segments on")

try:
    while True:
        # 7 セグメントディスプレイに各 16 進数デジットを表示
        for code in segCode:
```

(次のページに続く)

(前のページからの続き)

```
            hc595_shift(code)  # コードを 74HC595 にシフト
            # 表示されたセグメントコードを表示
            print(f"segCode[{segCode.index(code)}]: 0x{code:02X}")
            sleep(0.5)  # 各デジットを表示する間の一時停止

except KeyboardInterrupt:
    # スクリプトの中断（例：Ctrl+C）を優雅に処理
    pass
```

コードの説明

1. このスニペットはプロジェクトに必要なクラスをインポートします。 gpiozero から OutputDevice は、GPIO ピンに接続されたハードウェアコンポーネントを制御するために、 time から sleep は遅延を追加するために使用されます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep
```

2. SDI、RCLK、SRCLK は、74HC595 のシリアルデータ入力、メモリクロック入力（レジスタクロック）、シフトレジスタクロックピンに対応しています。

```
# 74HC595 シフトレジスタに接続された GPIO ピン
SDI = OutputDevice(17)   # シリアルデータ入力
RCLK = OutputDevice(18)  # メモリクロック入力（レジスタクロック）
SRCLK = OutputDevice(27) # シフトレジスタクロック
```

3. segCode は、7 セグメントディスプレイに表示する各デジットの 16 進数コードを含む配列です。

```
# 一般陰極型 7 セグメントディスプレイ用の 16 進数のデジットコード
segCode = [
    0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
    0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71
]
```

4. この関数は、データを 74HC595 に 8 ビットシフトします。それは各ビットを SDI に連続的に入力し、ビットをシフトするために SRCLK をトグルし、データを出力にラッチするために RCLK を使用します。

```
def hc595_shift(data):
    # 74HC595 に 8 ビットのデータをシフトします
```

(次のページに続く)

(前のページからの続き)

```
    for bit in range(8):
        # データビットに基づいて SDI を高または低に設定します
        SDI.value = 0x80 & (data << bit)
        # シフトレジスタクロックをトリガします
        SRCLK.on()
        sleep(0.001)
        SRCLK.off()
    # データをアウトプットにラッチするためにメモリクロックをトリガします
    RCLK.on()
    sleep(0.001)
    RCLK.off()
```

5. この関数は、ディスプレイのすべてのセグメントをオンにします。これは、共通カソードの 7 セグメントディスプレイ用です。

```
def display_all_on():
    # すべてのセグメントをオンにするための関数（一般陰極型 7 セグメントディスプレイ用）
    all_on_code = 0x3f
    hc595_shift(all_on_code)
    print("Displaying all segments on")
```

6. メインループでは、 segCode の各コードが順番にディスプレイに送られ、各コードの間に遅延があります。

```
try:
    while True:
        # 7 セグメントディスプレイに各 16 進数のデジットを表示します
        for code in segCode:
            hc595_shift(code)  # コードを 74HC595 にシフト
            # 表示されているセグメントコードを表示
            print(f"segCode[{segCode.index(code)}]: 0x{code:02X}")
            sleep(0.5)  # 各デジットの表示間に一時停止
```

7. この部分のコードは、スクリプトの中断（Ctrl+C など）を優雅に処理します。

```
except KeyboardInterrupt:
    # スクリプトの中断を優雅に処理します（例：Ctrl+C）
    pass
```

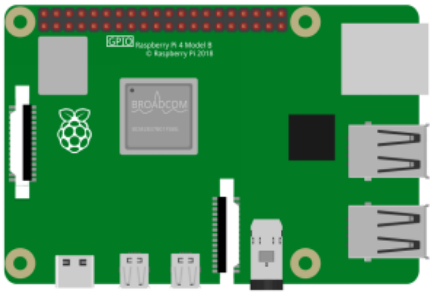
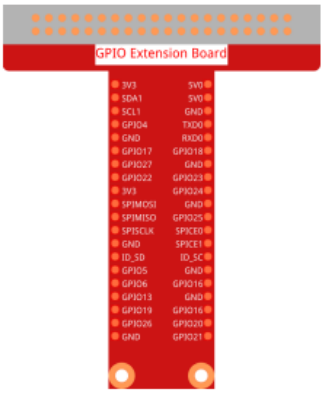
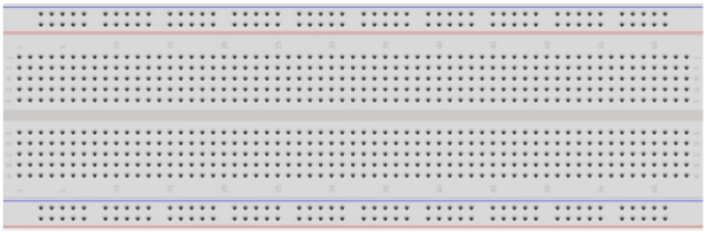
1.1.5 4 桁 7 セグメントディスプレイ

概要

次に、4 桁の 7 セグメントディスプレイを制御してみましょう。

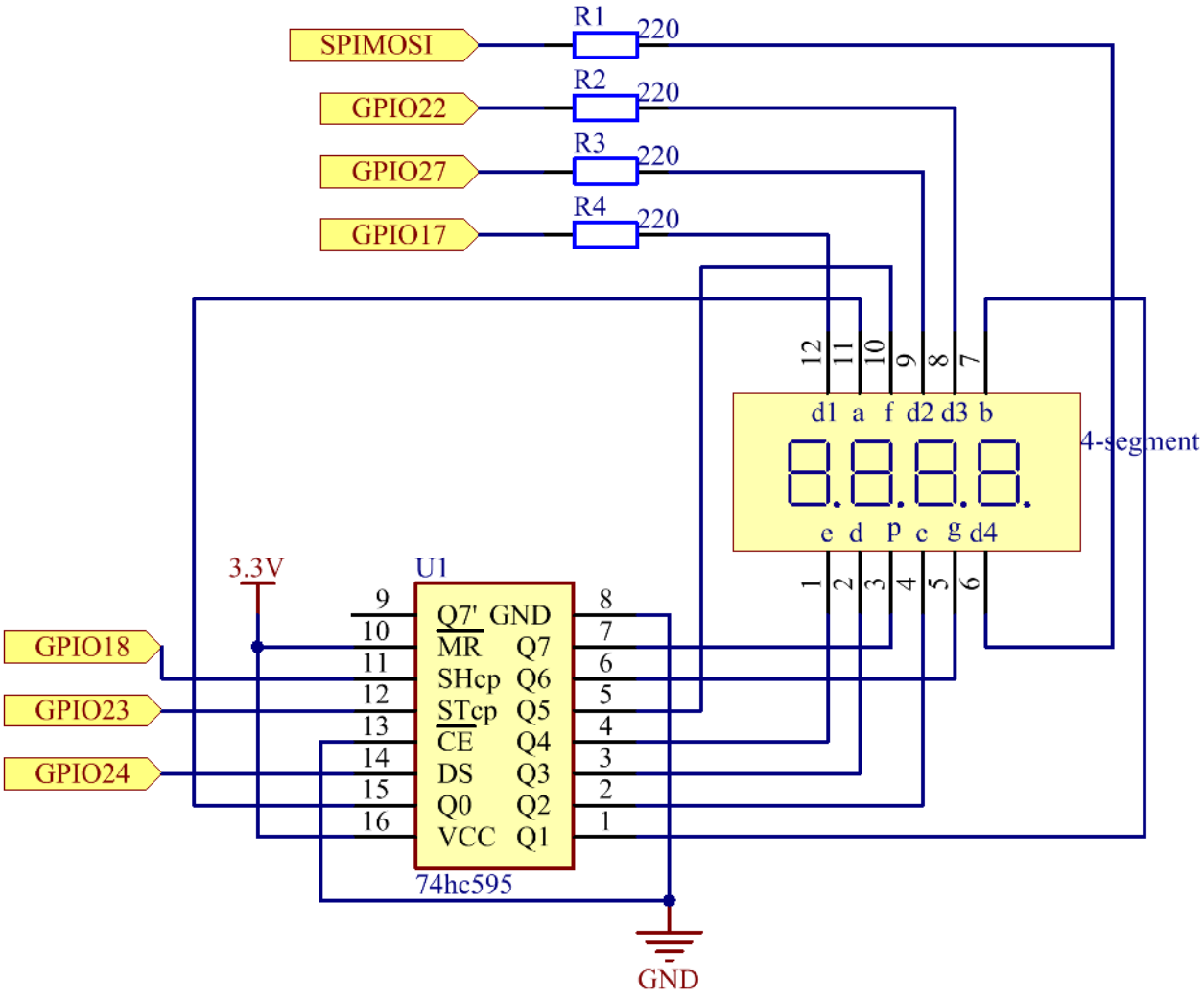
必要なコンポーネント

このプロジェクトには、次のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display
		1 * 74HC595
1 * 40-pin Cable		4 * Resistor(220Ω)
1 * Breadboard 		Several Jumper Wires

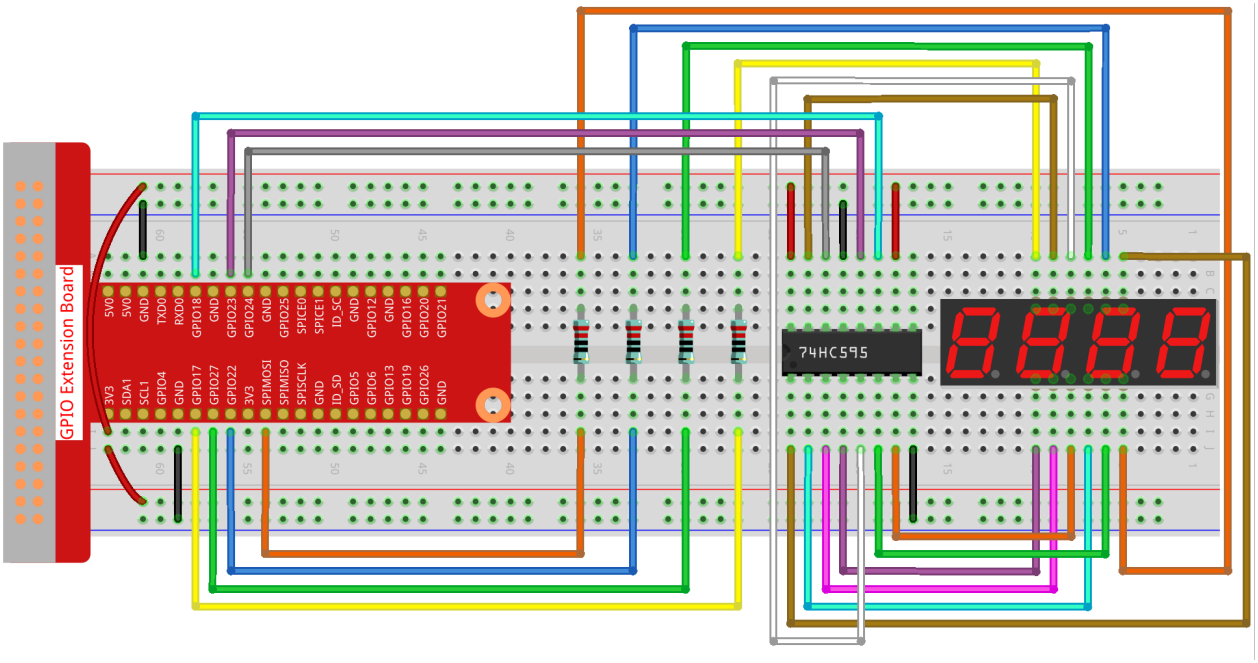
回路図

T-Board Name	physical	BCM
GPIO17	Pin 11	17
GPIO27	Pin 13	27
GPIO22	Pin 15	22
SPIMOSI	Pin 19	10
GPIO18	Pin 12	18
GPIO23	Pin 16	23
GPIO24	Pin 18	24



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

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

コードを実行した後、プログラムはカウントを 1 秒ごとに増やし、4 桁のディスプレイにカウントを表示します。

コード

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。問題がないことを確認したら、修正されたコードをコピーするための Copy ボタンを使用し、ターミナルで「nano」コマンドを使用してソースコードを開き、コードを貼り付けることができます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
```

(次のページに続く)

(前のページからの続き)

```
import time
import threading

# 74HC595 シフトレジスタ用の GPIO ピンを定義
SDI = OutputDevice(24)   # シリアルデータ入力
RCLK = OutputDevice(23)  # レジスタクロック
SRCLK = OutputDevice(18) # シフトレジスタクロック

# 7 セグメントディスプレイの数字を選択するための GPIO ピンを定義
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメントディスプレイに表示する 0 から 9 までの数字のセグメントコードを定義
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

counter = 0  # 表示用のカウンターを初期化
timer1 = 0   # カウンターの増分用のタイマーを初期化

def clearDisplay():
    """ 7 セグメントディスプレイをクリアします。 """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ 1 バイトのデータを 74HC595 シフトレジスタにシフトします。 """
    for i in range(8):
        SDI.value = 0x80 & (data << i)  # データビットに基づいて SDI を高/低に設定
        SRCLK.on()  # シフトレジスタクロックをパルス
        SRCLK.off()
    RCLK.on()  # レジスタクロックをパルスしてデータを出力にラッチ
    RCLK.off()

def pickDigit(digit):
    """ 7 セグメントディスプレイに表示する数字を選択します。 """
    for pin in placePin:
        pin.off()  # すべての数字選択ピンをオフにする
```

(次のページに続く)

(前のページからの続き)

```
    placePin[digit].on()  # 選択された数字をオンにする

def timer():
    """ カウンターを 1 秒ごとに増分するタイマー関数です。 """
    global counter, timer1
    timer1 = threading.Timer(1.0, timer)  # 次の増分のためにタイマーをリセット
    timer1.start()
    counter += 1  # カウンターを増分
    print("%d" % counter)  # 現在のカウンターの値を表示

def setup():
    """ 初期状態を設定し、タイマーを開始します。 """
    global timer1
    timer1 = threading.Timer(1.0, timer)  # タイマーの初期化と開始
    timer1.start()

def loop():
    """ カウンターの値で 7 セグメントディスプレイを更新するメインループです。 """
    global counter
    while True:
        for i in range(4):  # 各数字をループ
            clearDisplay()  # 新しい数字を設定する前にディスプレイをクリア
            pickDigit(i)    # 表示する数字を選択
            digit = (counter // (10 ** i)) % 10
            hc595_shift(number[digit])  # 数字の値を 74HC595 にシフト
            time.sleep(0.001)  # ディスプレイの安定性のための短い遅延

def destroy():
    """ 終了時に GPIO リソースをクリーンアップし、タイマーを停止します。 """
    global timer1
    timer1.cancel()  # タイマーを停止
    for device in [SDI, RCLK, SRCLK] + placePin:
        device.close()  # GPIO デバイスをクローズ

try:
    setup()  # 初期セットアップを実行
    while True:
        loop()  # メインループを開始
```

(次のページに続く)

(前のページからの続き)

```
except KeyboardInterrupt:
    # スクリプトの中断（例：Ctrl+C）
    destroy()  # 終了時にリソースをクリーンアップ
```

コードの説明

1. placePin リストは、4 桁の 7 セグメントディスプレイの各桁を制御するための共通アノードピンを制御します。
2. 16 進数で 0 から 9 までのセグメントコード（共通アノード）を含む number 配列を定義します。
3. setup 関数は、カウンタ増加用のタイマーを 1 秒ごとにトリガするためのセットアップを行います。
4. タイマーが 1.0 秒に達すると、タイマー関数が呼び出され、カウンタに 1 を追加し、次にタイマー自体が 1 秒ごとに繰り返し実行されます。
5. hc595_shift 関数は、74HC595 シフトレジスタに 1 バイトのデータをシフトし、ディスプレイセグメントを制御します。
6. メインループ loop では、現在のカウンタ値を表示し、各桁を順番に表示します。各桁を表示する前にディスプレイをクリアします。
7. clearDisplay 関数は、次の桁を表示する前にすべてのセグメントをオフに設定してディスプレイをクリアします。
8. pickDigit 関数は、アクティブ化する 7 セグメントディスプレイの桁を選択します。各桁は個別の GPIO ピンで制御されます。
9. プログラムが中断された場合、GPIO リソースを適切に解放し、タイマーを停止します。

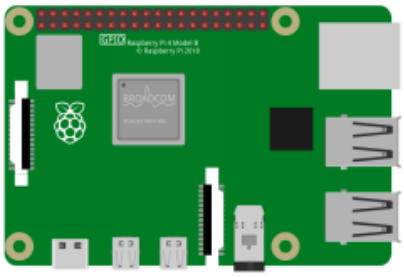
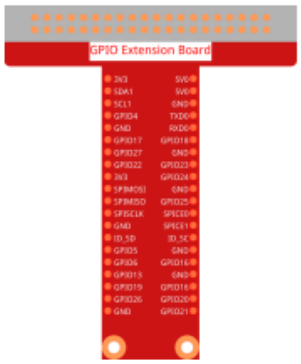
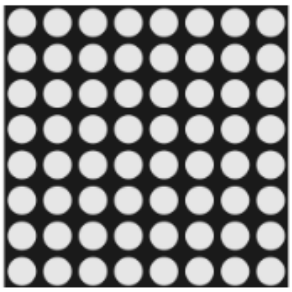
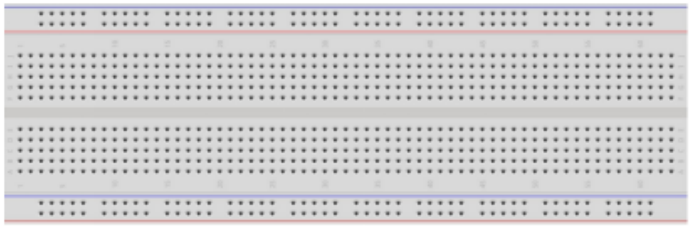
1.1.6 LED ドットマトリックス

はじめに

その名前からも分かるように、LED ドットマトリックスは LED で構成されたマトリックスです。LED の点灯と消灯によって、異なる文字やパターンを形成します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Dot Matrix 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		2 * 74HC595

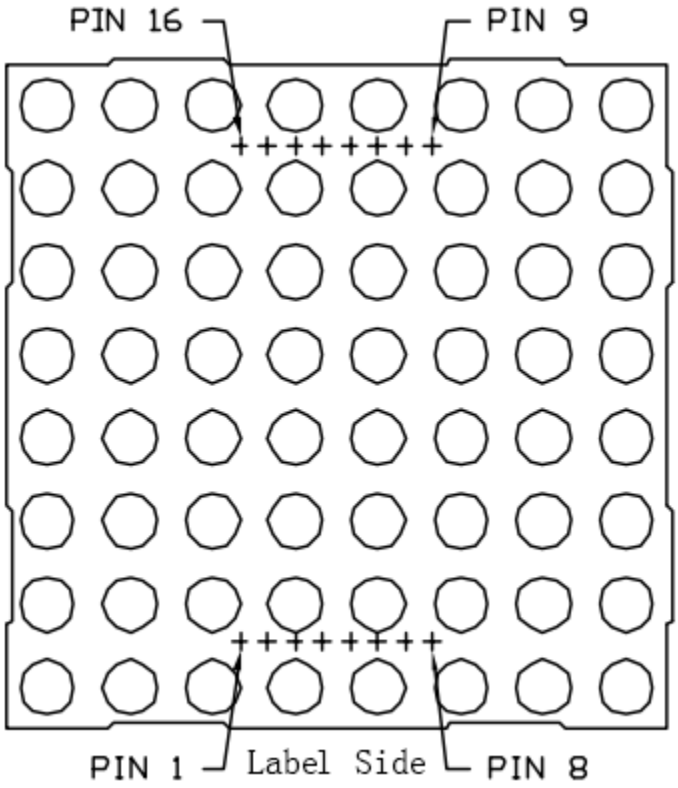
原理

LED ドットマトリックス

一般的に、LED ドットマトリックスは 2 つのタイプに分類されます：共通陰極（CC）と共通陽極（CA）。外観は似ていますが、内部に違いがあります。テストで区別できます。このキットでは CA タイプが使用されています。サイドにラベル「788BS」と表示されています。

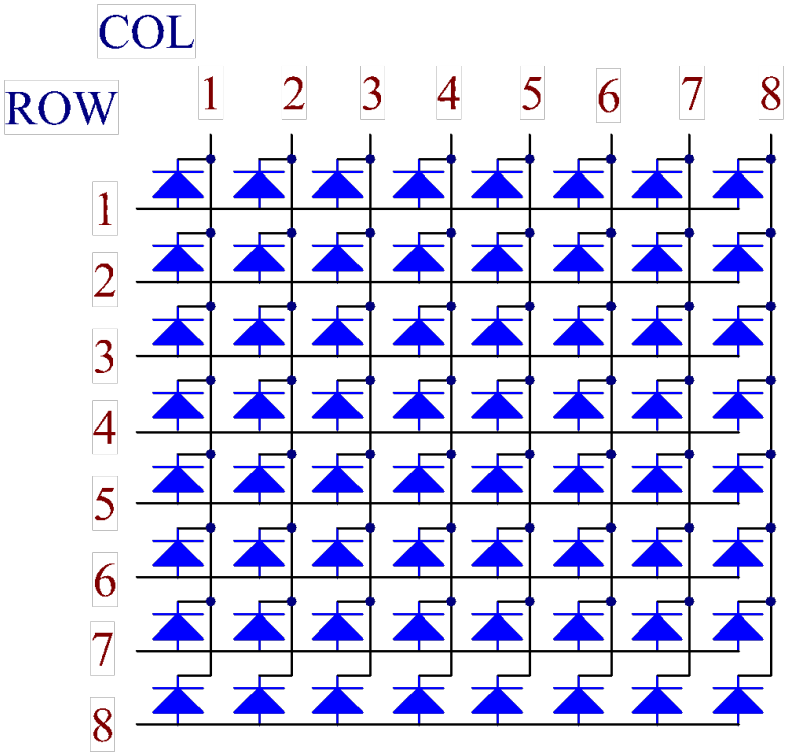
以下の図をご覧ください。ピンは背面の 2 つの端に配置されています。ラベル側を参照してください：この端のピンはピン 1 から 8 で、もう一方はピン 9 から 16 です。

外部ビュー：



以下の図は内部構造を示しています。CA LED ドットマトリックスでは、ROW は LED の陽極を表し、COL は陰極です。CC の場合は逆です。共通点は、両方のタイプともに、ピン 13、3、4、10、6、11、15、16 がすべて COL であり、ピン 9、14、8、12、1、7、2、5 がすべて ROW であることです。左上隅の最初の LED を点灯させたい場合、CA LED ドットマトリックスの場合、ピン 9 を High に設定してピン 13 を Low に、CC の場合はピン 13 を High に設定してピン 9 を Low にします。最初の列全体を点灯させたい場合、CA の場合、ピン 13 を Low にし、ROW 9、14、8、12、1、7、2、5 を High に設定し、CC の場合、ピン 13 を High にし、ROW 9、14、8、12、1、7、2、5 を Low に設定します。理解を深めるために、以下の図を参考にしてください。

内部ビュー：



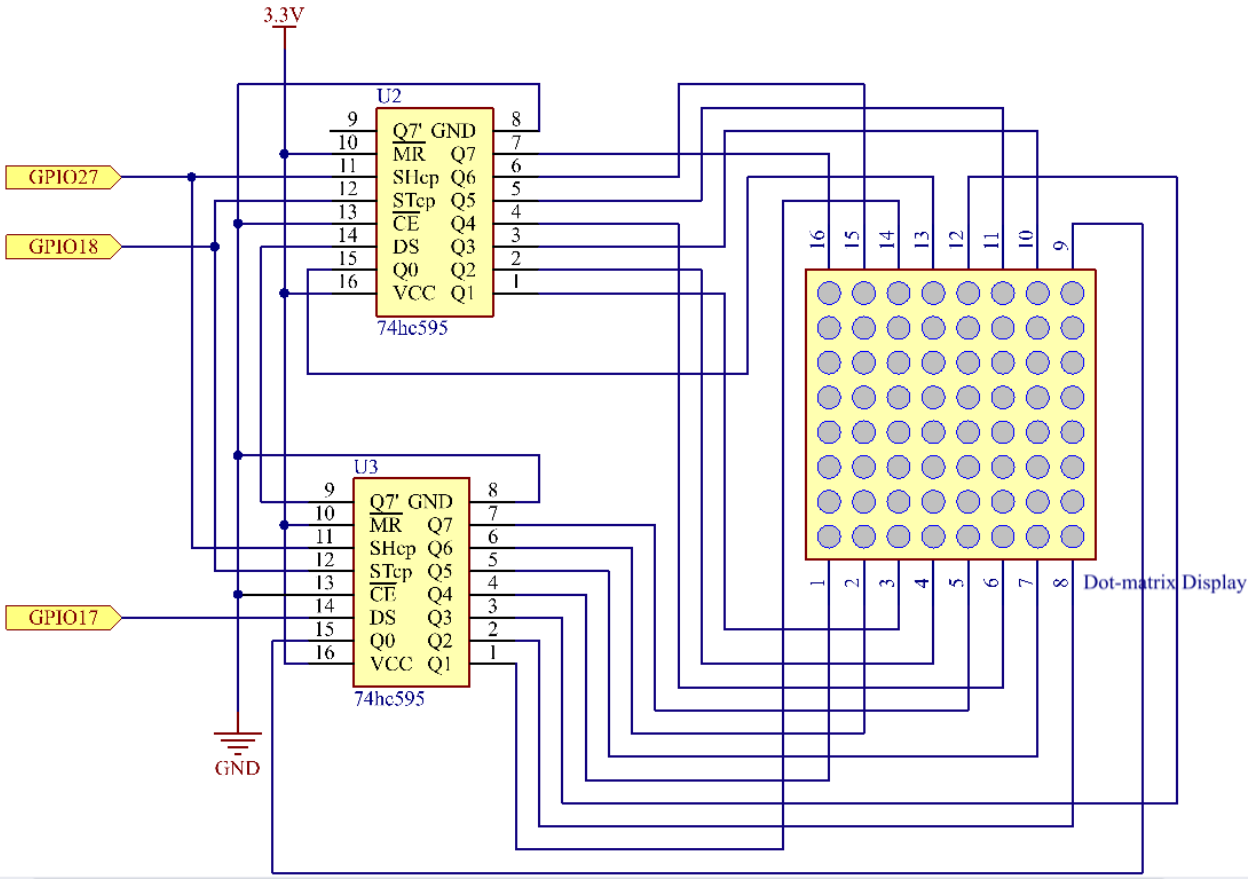
上記の行と列に対応するピン番号：

COL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**13**	**3**	**4**	**10**	**6**	**11**	**15**	**16**
ROW	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**9**	**14**	**8**	**12**	**1**	**7**	**2**	**5**

さらに、ここでは 2 つの 74HC595 チップを使用しています。1 つは LED ドットマトリックスの行を制御し、もう 1 つは列を制御します。

回路図

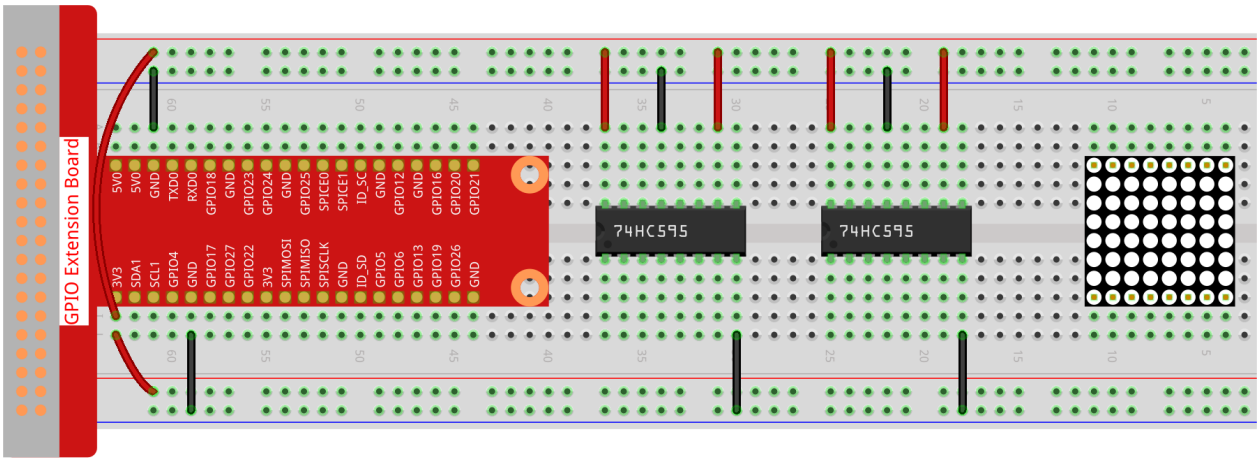
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



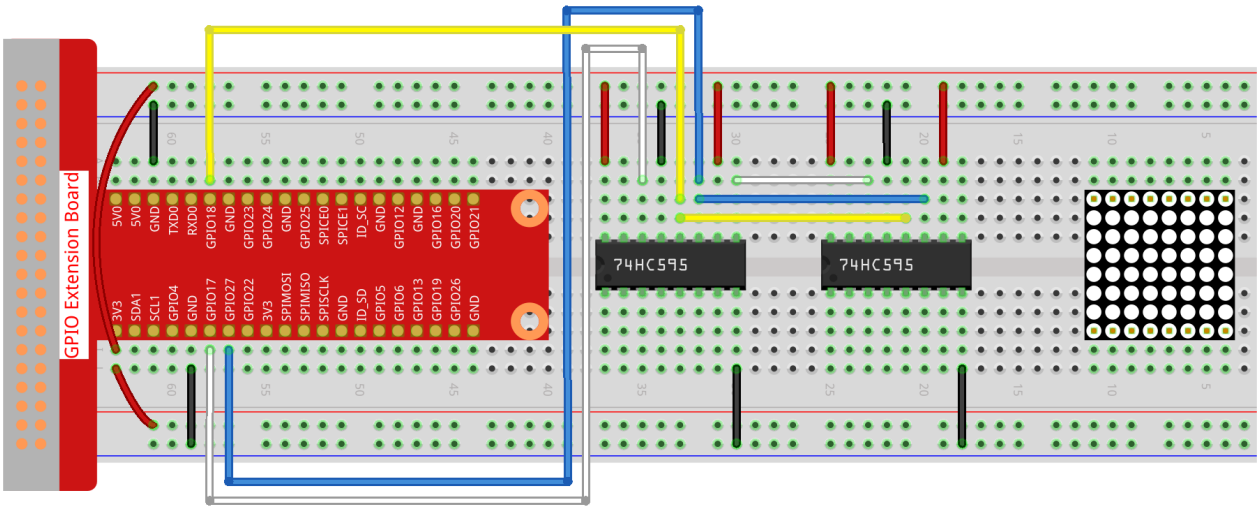
実験手順

ステップ 1: 回路を組み立てます。配線が複雑なので、ステップバイステップで進めましょう。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します。T-Cobbler の 3.3V と GND をボードの両側の穴に接続し、次に 2 つの 74HC595 チップのピン 16 と 10 を VCC に、ピン 13 とピン 8 を GND に接続します。

注釈: 上記の Fritzing イメージでは、ラベルが下にある側です。

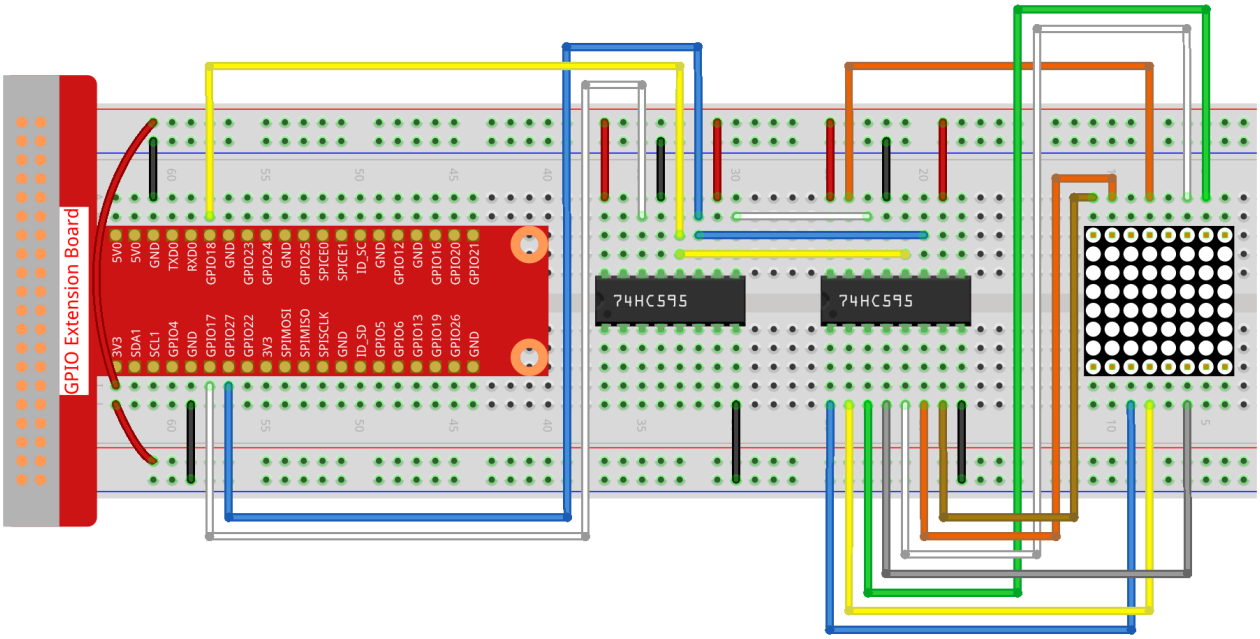


ステップ 2: 2 つの 74HC595 のピン 11 を一緒に接続し、次に GPIO27 に接続します。次に、2 つのチップのピン 12 を接続し、GPIO18 に接続します。その後、左側の 74HC595 のピン 14 を GPIO17 に、ピン 9 を 2 番目の 74HC595 のピン 14 に接続します。



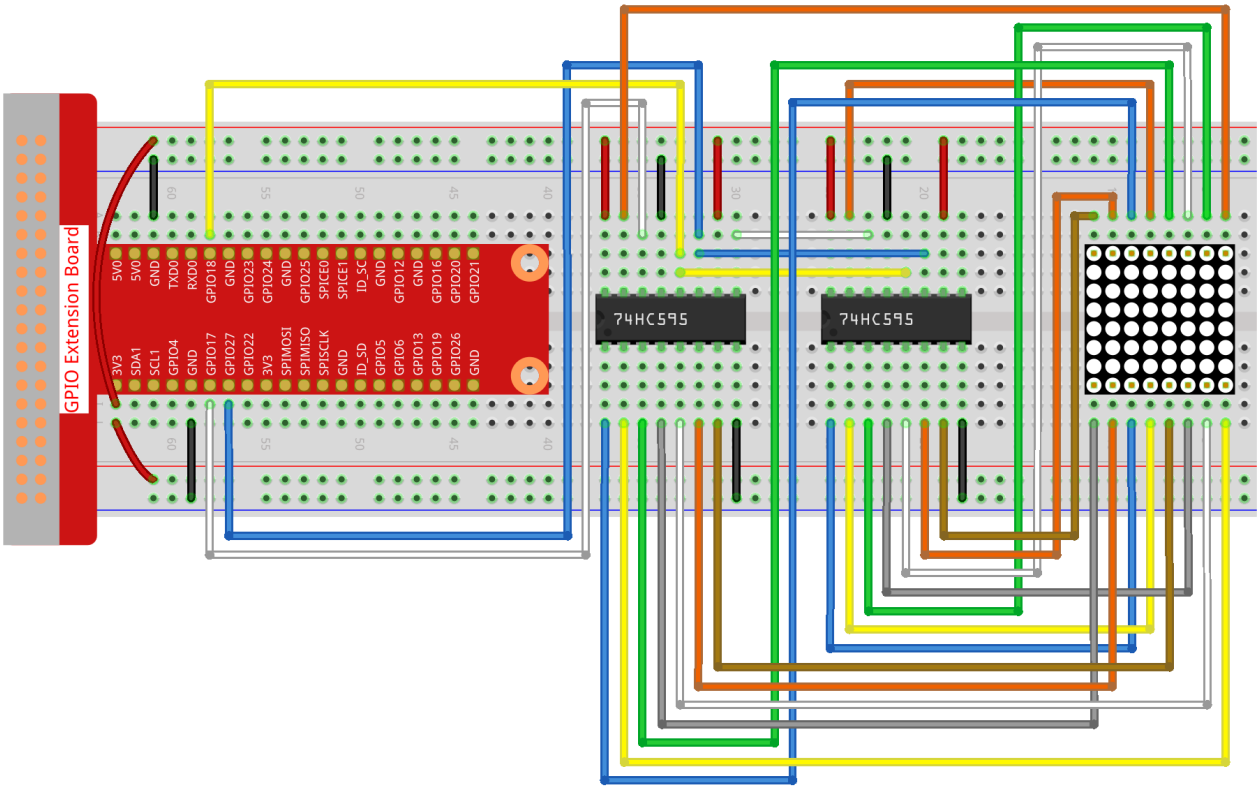
ステップ 3: 右側の 74HC595 は LED ドットマトリックスの列を制御します。以下のテーブルを参照してマッピングを確認してください。したがって、74HC595 の Q0-Q7 ピンは、ピン 13、3、4、10、6、11、15、16 とそれぞれマッピングされています。

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: さて、LED ドットマトリックスの ROW を接続します。左側の 74HC595 が LED ドットマトリックスの ROW を制御します。以下のテーブルを参照してマッピングを確認してください。左側の 74HC595 の Q0-Q7 は、ピン 9、14、8、12、1、7、2、5 とそれぞれマッピングされています。

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



ステップ 5: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 6: 実行します。

```
sudo python3 1.1.6_led_dot_matrix_zero.py
```

コードを実行すると、LED ドットマトリックスが一列ずつ点灯し、列ごとに消灯します。

コード

注釈: 以下のコードは **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に「davinci-kit-for-raspberry-pi/python-pi5」というソースコードのパスに移動する必要があります。コードを変更した後、そのまま実行して効果を確認できます。問題がないことを確認したら、修正されたコードをコピーするための Copy ボタンを使用し、ターミナルで「nano」コマンドを使用してソースコードを開き、コードを貼り付けることができます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
```

(次のページに続く)

(前のページからの続き)

```
from time import sleep

# Define GPIO pins connected to the 74HC595 shift register
SDI = OutputDevice(17)   # Serial Data Input
RCLK = OutputDevice(18)  # Register Clock
SRCLK = OutputDevice(27) # Shift Register Clock

# Define patterns for matrix display; ROWs are anodes (+), COLs are cathodes (-)
# Pattern for ROWs (anode signals)
code_H = [0x01, 0xff, 0x80, 0xff, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0xff,␣
↪0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
# Pattern for COLs (cathode signals)
code_L = [0x00, 0x7f, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe,␣
↪0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f]

# Shift data to 74HC595
def hc595_shift(dat):
    """ Shift data to the 74HC595 shift register for displaying on the matrix. """
    for i in range(8):
        # Set SDI value and trigger shift register clock
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
    # Trigger register clock to update display
    RCLK.on()
    sleep(0.001)
    RCLK.off()

def main():
    """ Main loop for cycling through display patterns. """
    while True:
        # Cycle through patterns in ascending order
        for i in range(len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

        # Cycle through patterns in descending order
        for i in range(len(code_H)-1, -1, -1):
```

(次のページに続く)

(前のページからの続き)

```
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

# Run main loop, handle keyboard interrupt gracefully
try:
    main()
except KeyboardInterrupt:
    pass
```

コードの説明

1. このスニペットは、プロジェクトのために必要なクラスをインポートします。 gpiozero からの OutputDevice は、GPIO ピンに接続されたハードウェアコンポーネントを制御するために使用され、time からの sleep は遅延を追加するために使用されます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep
```

2. このセクションでは、74HC595 シフトレジスタに接続された GPIO ピンを初期化します。 SDI はシリアルデータ入力、 RCLK はレジスタクロック、 SRCLK はシフトレジスタクロックです。これらのピンはデータをレジスタにシフトし、LED マトリクスの表示を制御するために使用されます。

```
# 74HC595 シフトレジスタに接続された GPIO ピンを定義
SDI = OutputDevice(17)   # シリアルデータ入力
RCLK = OutputDevice(18)  # レジスタクロック
SRCLK = OutputDevice(27) # シフトレジスタクロック
```

3. code_H と code_L は、LED マトリクスの行（陽極）と列（陰極）を制御するための 2 進パターンを定義します。これらの配列の各要素は、マトリクス内のどの LED がオンまたはオフになるかを制御する 2 進パターンを表します。

```
# マトリクス表示のためのパターンを定義します。行は陽極（+）、列は陰極（-）です。
# 行のパターン（陽極信号）
code_H = [0x01, 0xff, 0x80, 0xff, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,␣
↪0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
# 列のパターン（陰極信号）
code_L = [0x00, 0x7f, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,␣
↪0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f]
```

4. この関数は、バイトデータ（ dat ）を 74HC595 シフトレジスタにシフトします。バイト内の各ビットに対して、ビット値に基づいて SDI ピンを高または低に設定し、ビットをレジスタにシフトするために SRCLK ピンをトグルさせます。すべてのビットがシフトされた後、LED マトリクスの表示を更新するために RCLK ピンをトグルさせます。

```
# データを 74HC595 にシフト
def hc595_shift(dat):
    """ マトリクスに表示するためのデータを 74HC595 シフトレジスタにシフトします。 """
    for i in range(8):
        # SDI の値を設定してシフトレジスタクロックをトリガー
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
    # レジスタクロックをトリガーして表示を更新
    RCLK.on()
    sleep(0.001)
    RCLK.off()
```

5. メイン関数には、LED マトリクスの事前定義されたパターンを無限ループで循環させるコードが含まれています。 hc595_shift 関数を使用してシフトレジスタに行と列のパターン（ code_H と code_L ）を送信し、昇順で最初に、次に降順で表示を作成します。

```
def main():
    """ 表示パターンを循環させるためのメインループ。 """
    while True:
        # 昇順でパターンを循環させます
        for i in range(len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

        # 降順でパターンを循環させます
        for i in range(len(code_H)-1, -1, -1):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)
```

6. このセグメントは、プログラムがキーボード割り込み（Ctrl+C）を使用して中断できるようにします。メインループを中断する際に急激な停止やリソースの漏洩が発生しないようにします。

```
# メインループを実行し、キーボード割り込みを丁寧に処理します
try:
    main()
except KeyboardInterrupt:
    pass
```

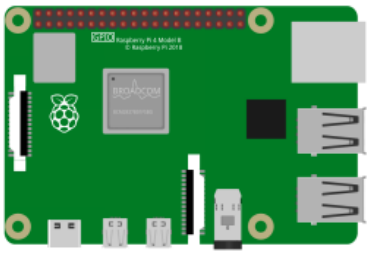
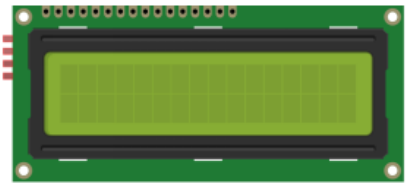
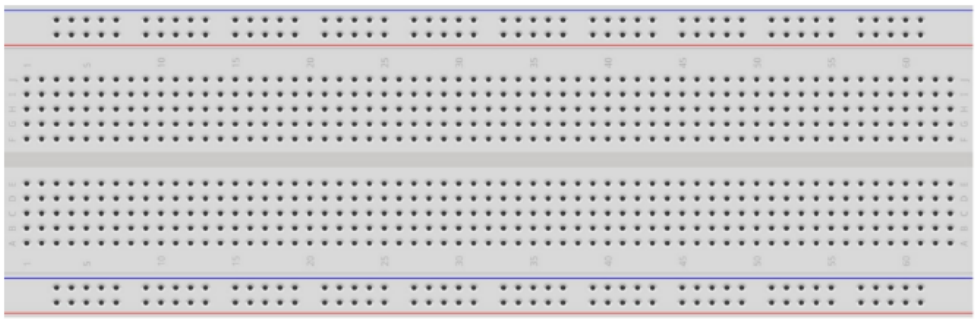
1.1.7 I2C LCD1602

はじめに

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

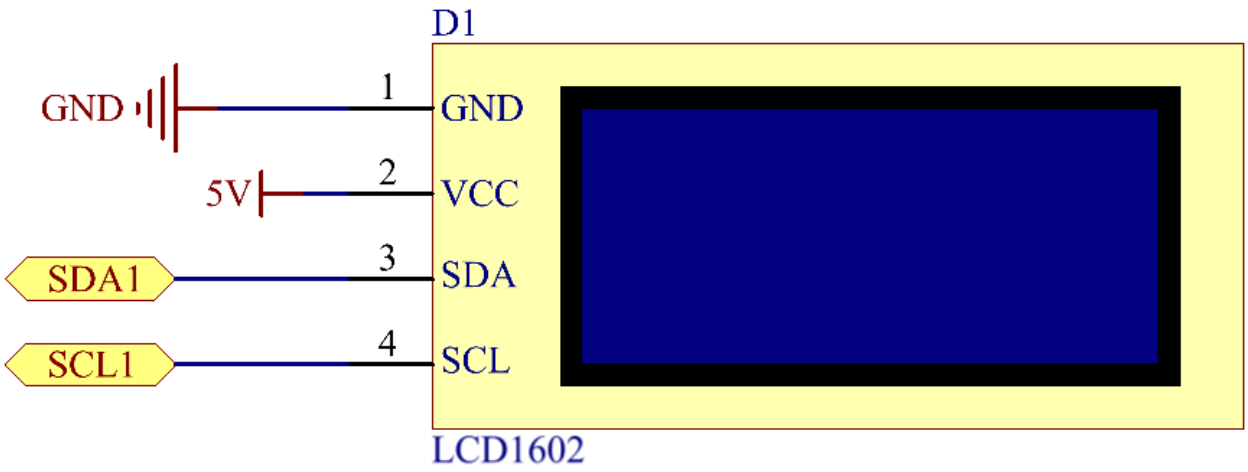
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board	1 * I2C LCD1602 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		

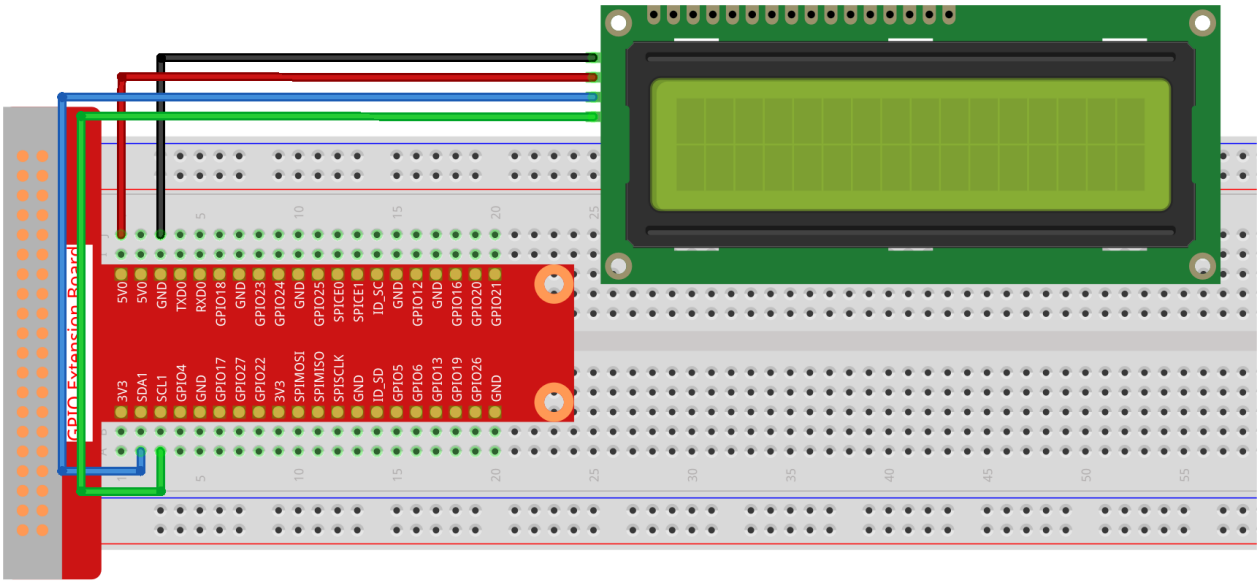
回路図

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: I2C をセットアップします（*I2C* **構成** を参照してください。I2C を設定済みの場合は、このステップをスキップしてください。）

ステップ 3: ディレクトリを変更します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 4: 実行します。

```
sudo python3 1.1.7_Lcd1602_zero.py
```

コードの実行後、LCD に「Greetings!、From SunFounder」と表示されます。

注釈:

- 「FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'」というエラーが表示された場合、I2C を有効にするには *I2C* **構成** を参照してください。
- 「ModuleNotFoundError: No module named 'smbus2'」エラーが表示された場合、 `sudo pip3 install smbus2` を実行してください。
- エラー「OSError: [Errno 121] Remote I/O error」が表示された場合、モジュールの配線が誤っているか、モジュールが壊れていることを意味します。
- コードと配線が正常であるが、LCD にコンテンツが表示されない場合、背面のポテンショメータを回してコントラストを上げることができます。

コード

注釈: 以下のコードは変更/リセット/コピー/実行/停止ができます。ただし、コードを変更する前に `davinci-kit-for-raspberry-pi/python-pi5` のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認できます。

```
#!/usr/bin/env python3
import LCD1602  # LCD1602 とのインターフェースを行うモジュールをインポート
import time     # タイミング関数のモジュールをインポート

def setup():
    # I2C アドレス 0x27 およびバックライトを有効にして LCD を初期化
    LCD1602.init(0x27, 1)
    # メッセージ 'Greetings!' を左上隅（行 0、列 0）に表示
    LCD1602.write(0, 0, 'Greetings!')
    # メッセージ 'From SunFounder' を 2 行目（行 1、列 1）に表示
    LCD1602.write(1, 1, 'From SunFounder')
    time.sleep(2)  # メッセージを 2 秒間表示

try:
    setup()  # LCD の初期化およびメッセージの表示のためのセットアップ関数を実行します

except KeyboardInterrupt:
    # キーボードの割り込み（例：Ctrl+C）が発生した場合、LCD ディスプレイをクリアします
```

(次のページに続く)

(前のページからの続き)

```
    LCD1602.clear()
    pass   # それ以上のアクションは不要です
```

コードの説明

1. 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す。これを使用すると、I2C LCD1602 を簡単に使用できます。

   ```
   import LCD1602   # LCD1602 とのインターフェースを行うモジュールをインポート
   ```

2. この関数は指定されたデバイスシンボルを使用して I2C システムを初期化します。最初のパラメータは I2C デバイスのアドレスで、i2cdetect コマンドを使用して検出できます（詳細は付録を参照）。I2C LCD1602 のアドレスは通常 0x27 です。

   ```
   # I2C アドレス 0x27 およびバックライトを有効にして LCD を初期化
   LCD1602.init(0x27, 1)
   ```

3. LCD1602.write は LCD にメッセージを表示するために使用されます。最初の 2 つのパラメータは行と列の位置で、3 番目はメッセージです。これで LCD に "Greetings!! From SunFounder" と表示されるようになります。

   ```
   # メッセージ 'Greetings!' を左上隅（行 0、列 0）に表示
   LCD1602.write(0, 0, 'Greetings!')
   # メッセージ 'From SunFounder' を 2 行目（行 1、列 1）に表示
   LCD1602.write(1, 1, 'From SunFounder')
   ```

1.2 音声

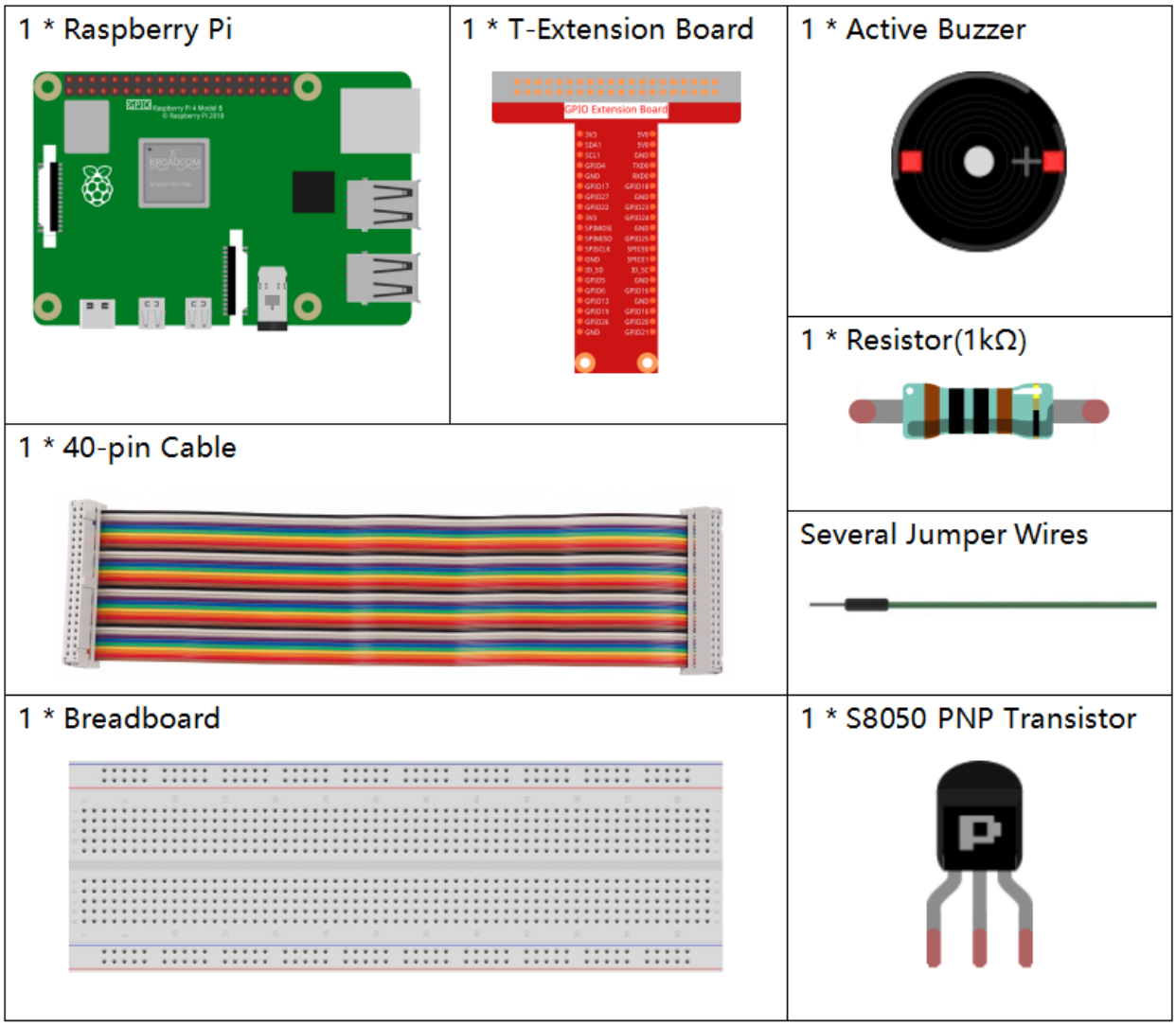
1.2.1 アクティブブザー

はじめに

このプロジェクトでは、PNP トランジスタを使用してアクティブブザーを鳴らす方法を学びます。

必要なコンポーネント

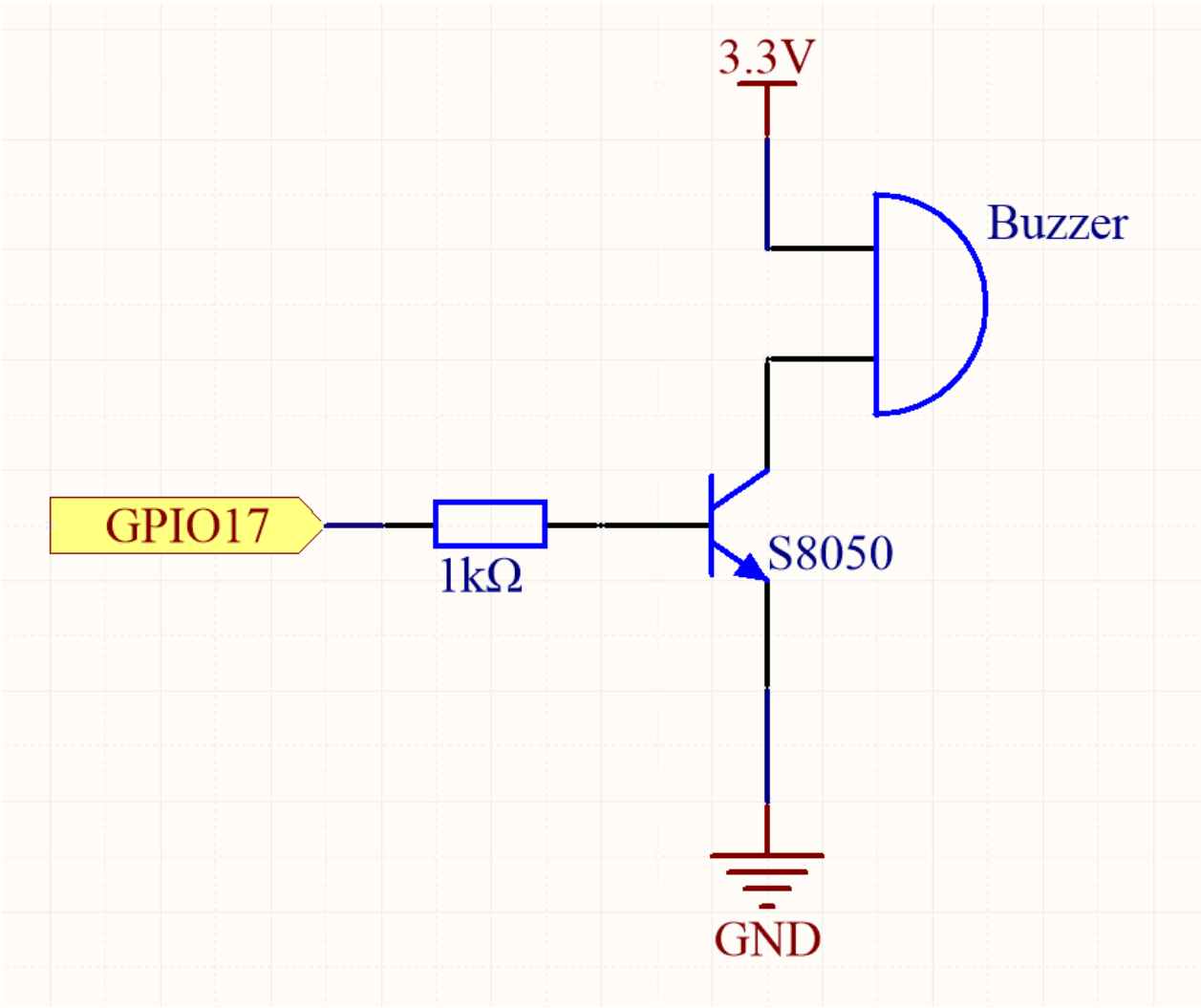
このプロジェクトでは、以下のコンポーネントが必要です。



回路図

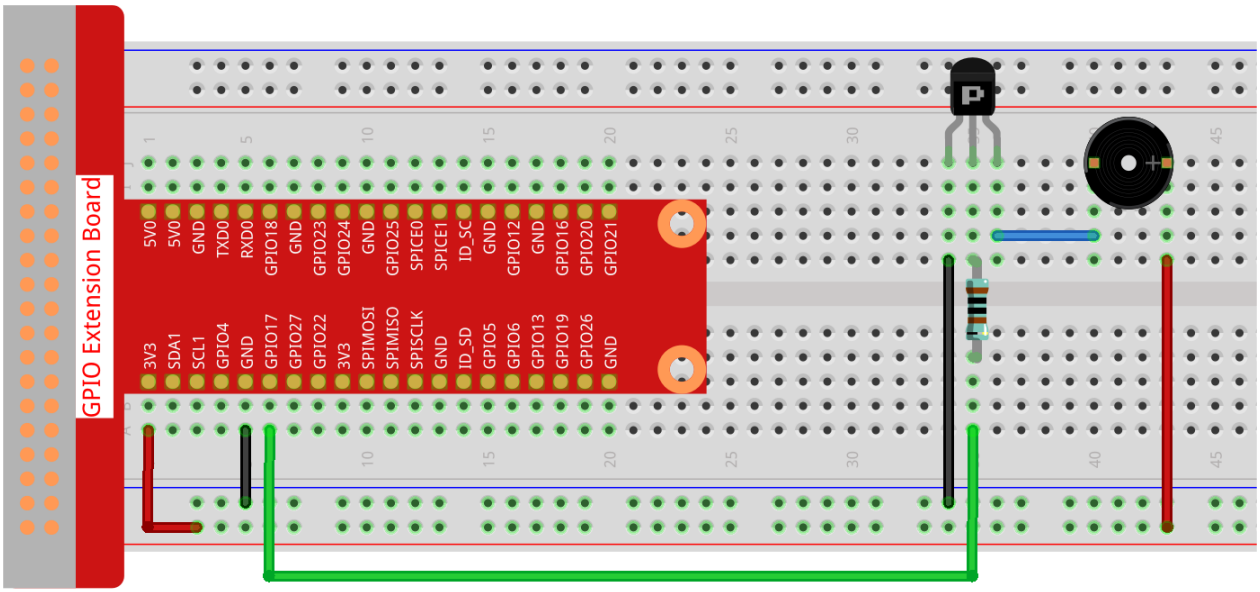
この実験では、アクティブブザー、PNP トランジスタ、および 1k Ω の抵抗器が、トランジスタのベースと GPIO の間に使用されてトランジスタを保護します。 Raspberry Pi の GPIO17 の出力に高い電圧（3.3V）が供給されると、トランジスタは電流飽和のために導通し、ブザーは音を鳴らします。しかし、Raspberry Pi の IO に低い電圧が供給されると、トランジスタは遮断され、ブザーは音を鳴らしません。

T-Board Name	physical	BCM
GPIO17	Pin 11	17



実験手順

ステップ 1: 回路を組み立てます（アクティブブザーには表面に白いテープルステッカーがあり、裏面は黒です）。



ステップ 2: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 1.2.1_ActiveBuzzer_zero.py
```

コードを実行すると、ブザーが鳴ります。

コード

注釈: 以下のコードは変更/リセット/コピー/実行/停止ができます。ただし、コードを変更する前に davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import Buzzer
from time import sleep

# GPIO ピン 17 に接続された Buzzer オブジェクトを初期化します
buzzer = Buzzer(17)

try:
    while True:
```

(次のページに続く)

(前のページからの続き)

```
        # ブザーをオンにする
        print(' ブザーオン')
        buzzer.on()
        sleep(0.1)  # ブザーを 0.1 秒間オンにします

        # ブザーをオフにする
        print(' ブザーオフ')
        buzzer.off()
        sleep(0.1)  # ブザーを 0.1 秒間オフにします

except KeyboardInterrupt:
    # キーボードの割り込み（Ctrl+C）を処理してスクリプトをきれいに終了します
    pass
```

コードの説明

1. これらのステートメントは gpiozero ライブラリから Buzzer クラスと time モジュールから sleep 関数をインポートします。

```
#!/usr/bin/env python3
from gpiozero import Buzzer
from time import sleep
```

2. この行は、Raspberry Pi の GPIO ピン 17 に接続された Buzzer オブジェクトを作成します。

```
# GPIO ピン 17 に接続された Buzzer オブジェクトを初期化します
buzzer = Buzzer(17)
```

3. 無限ループ（ while True ）内で、ブザーは 0.1 秒ごとにオンとオフになります。 print ステートメントは各アクションに対するコンソール出力を提供します。

```
try:
    while True:
        # ブザーをオンにする
        print(' ブザーオン')
        buzzer.on()
        sleep(0.1)  # ブザーを 0.1 秒間オンにします

        # ブザーをオフにする
        print(' ブザーオフ')
```

(次のページに続く)

(前のページからの続き)

```
        buzzer.off()
        sleep(0.1)  # ブザーを 0.1 秒間オフにします
```

4. このセグメントは、キーボード割り込み（Ctrl+C）を使用してプログラムを安全に終了できるようにします。

```
except KeyboardInterrupt:
# キーボードの割り込み（Ctrl+C）を処理してスクリプトをきれいに終了します
pass
```

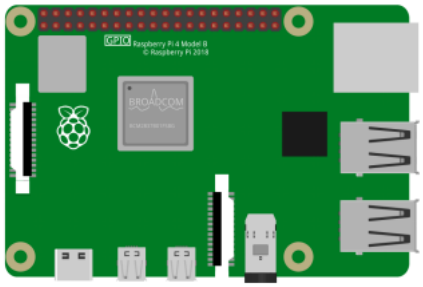
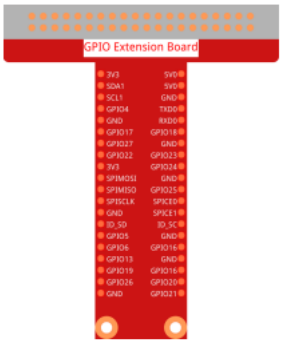
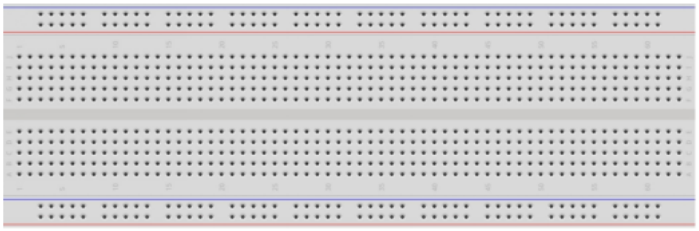
1.2.2 パッシブブザー

はじめに

このプロジェクトでは、パッシブブザーを使用して音楽を演奏する方法を学びます。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

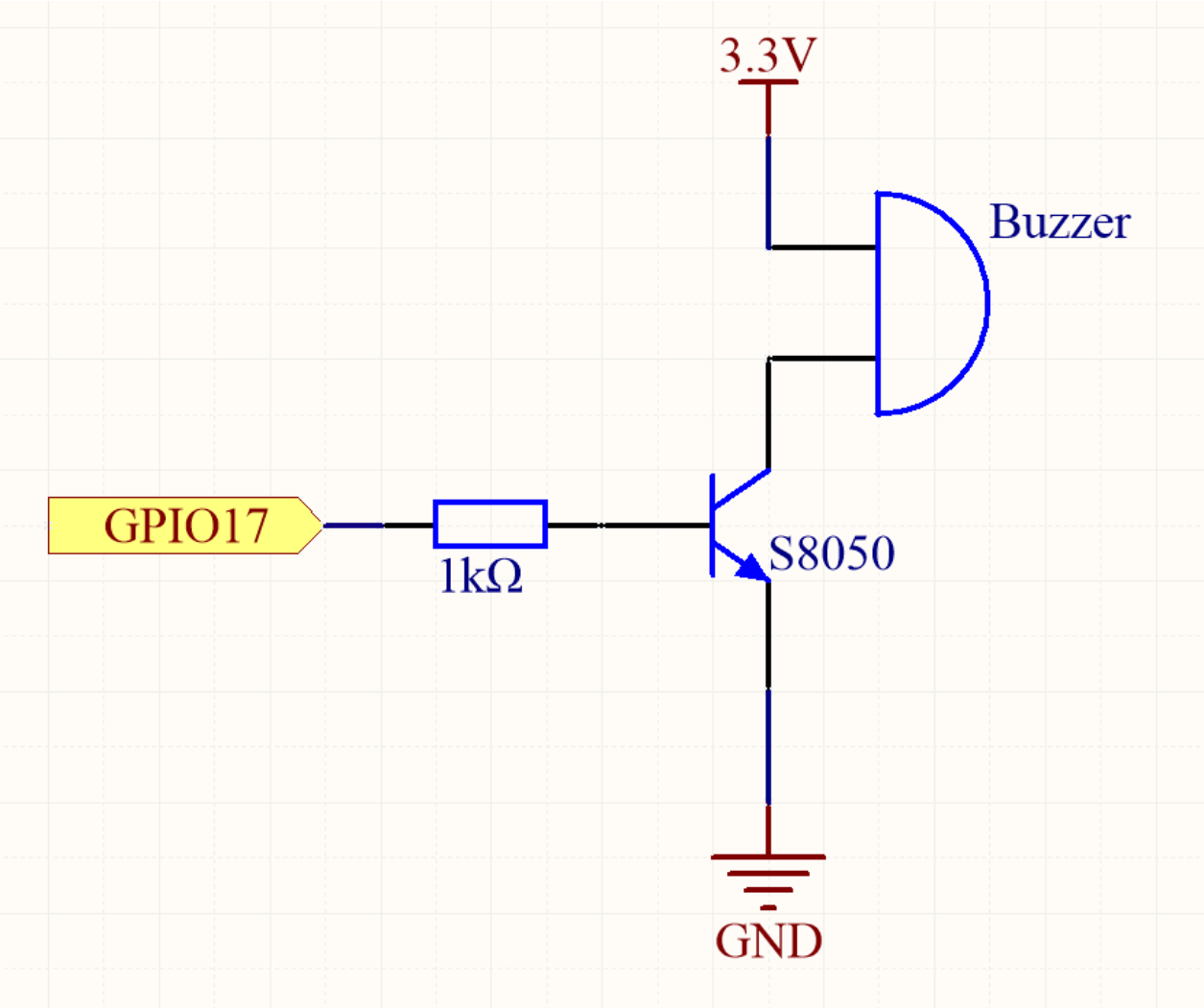
1 * Raspberry Pi 	1 * T-Extension Board 	1 * Passive Buzzer
1 * 40-pin Cable		1 * Resistor(1kΩ)
		Several Jumper Wires
1 * Breadboard 		1 * S8050 PNP Transistor

回路図

この実験では、パッシブブザー、PNP トランジスタ、および 1k Ωの抵抗器が、トランジスタのベースと GPIO の間に使用されてトランジスタを保護します。

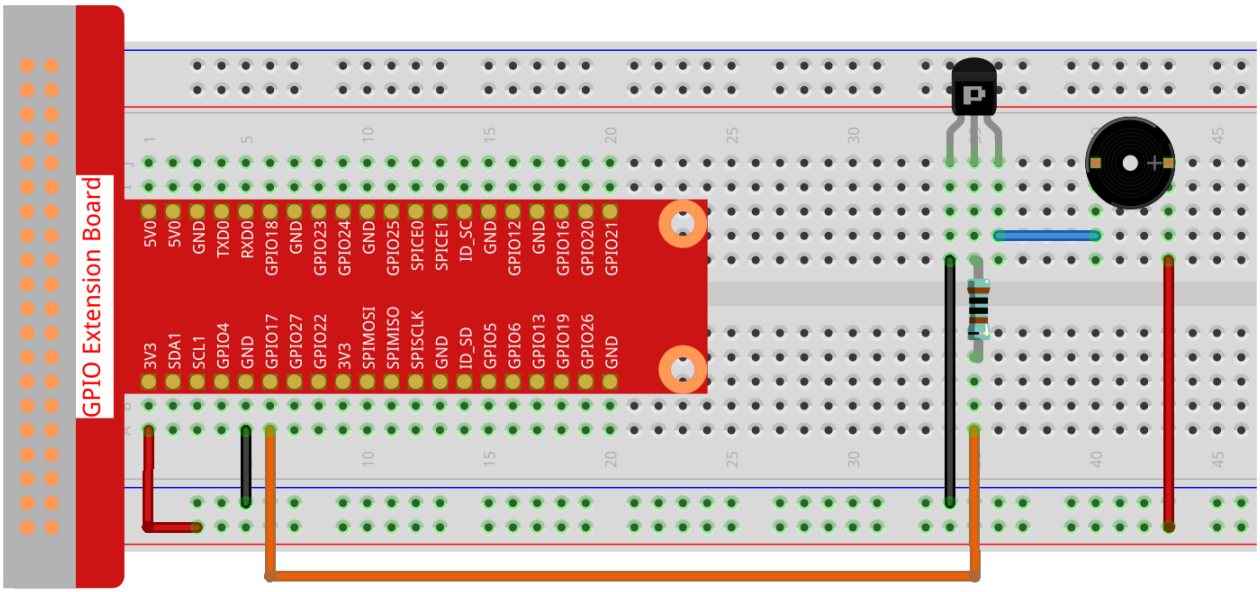
GPIO17 に異なる周波数が供給されると、パッシブブザーは異なる音を鳴らします。これにより、ブザーは音楽を演奏します。

T-Board Name	physical	BCM
GPIO17	Pin 11	17



実験手順

ステップ 1: 回路を組み立てます（パッシブブザーは裏面に緑の回路基板があります）。



ステップ 2: ディレクトリを変更します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 1.2.2_PassiveBuzzer_zero.py
```

コードを実行すると、ブザーが音楽を演奏します。

コード

注釈: 以下のコードは変更/リセット/コピー/実行/停止ができます。ただし、コードを変更する前に davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import TonalBuzzer
from time import sleep

# GPIO ピン 17 に接続された TonalBuzzer オブジェクトを初期化します
tb = TonalBuzzer(17)  # ご使用のセットアップに基づいてこのピン番号を更新します

def play(tune):
    """
```

(次のページに続く)

(前のページからの続き)

```
    ブザーを使用して音楽を演奏します。
    :param tune: ノートとその持続時間を表すタプル（ノート、持続時間）のリスト
    """
    for note, duration in tune:
        print(note)  # 現在再生中の音符を出力
        tb.play(note)  # ブザーで音符を演奏
        sleep(float(duration))  # 音符の持続時間待ち
    tb.stop()  # 曲が終わった後に演奏を停止します

# 音楽チューンをノートと持続時間のシーケンスとして定義します
tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2),
    ('Eb4', 0.2), ('E4', 0.2), (None, 0.6),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.6),
    ('Eb4', 0.2), ('E4', 0.2), (None, 0.2),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
    ('C4', 0.2), ('B4', 0.2), (None, 0.2),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
    ('B4', 0.2), ('Bb4', 0.5), (None, 0.6),
    ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
    ('D4', 0.2), ('E4', 0.2)]

try:
    play(tune)  # 演奏を開始するために play 関数を実行します

except KeyboardInterrupt:
    # キーボード割り込みをスムーズに処理
    pass
```

コードの説明

1. これらの行はブザーの制御のために gpiozero ライブラリから TonalBuzzer クラス、待機を作成するために time モジュールから sleep 関数をインポートします。

   ```
   #!/usr/bin/env python3
   from gpiozero import TonalBuzzer
   from time import sleep
   ```

2. この行は GPIO ピン 17 に接続された TonalBuzzer オブジェクトを初期化します。

```
# GPIO ピン 17 に接続された TonalBuzzer オブジェクトを初期化します
tb = TonalBuzzer(17)  # お使いのセットアップに基づいてこのピン番号を更新してください
```

3. play 関数は、音楽のノートとその持続時間を表すタプルのリストを繰り返し処理します。各ノートは指定された持続時間だけ演奏され、曲が終了するとブザーが停止します。

```
def play(tune):
    """
    ブザーを使って音楽を演奏します。
    :param tune: ノートとその持続時間を表すタプル（ノート、持続時間）のリスト。
    """
    for note, duration in tune:
        print(note)  # 現在演奏されているノートを出力
        tb.play(note)  # ブザーでノートを演奏
        sleep(float(duration))  # ノートの持続時間分遅延
    tb.stop()  # 曲が完了した後に演奏を停止
```

4. メロディは音符（周波数）と持続時間（秒）のシーケンスとして定義されています。

```
# ノートと持続時間（秒）のシーケンスとして音楽のメロディを定義します
tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2),
    ('Eb4', 0.2), ('E4', 0.2), (None, 0.6),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.6),
    ('Eb4', 0.2), ('E4', 0.2), (None, 0.2),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
    ('C4', 0.2), ('B4', 0.2), (None, 0.2),
    ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
    ('B4', 0.2), ('Bb4', 0.5), (None, 0.6),
    ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
    ('D4', 0.2), ('E4', 0.2)]
```

5. play(tune) 関数は try ブロック内で呼び出されます。 KeyboardInterrupt （Ctrl+C のような）はプログラムを正常に停止します。

```
try:
    play(tune)  # 曲を演奏を開始するために play 関数を実行します

except KeyboardInterrupt:
    # Ctrl+C のようなキーボード割り込みを処理してプログラムをきれいに終了します
    pass
```

1.3 ドライバー

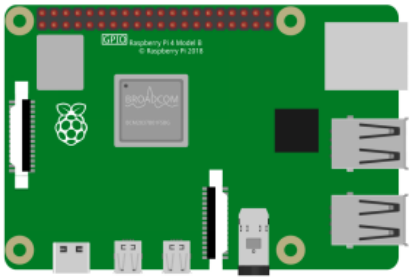
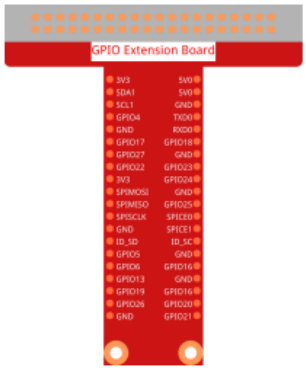
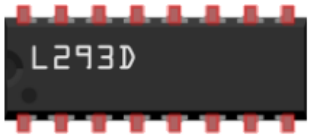
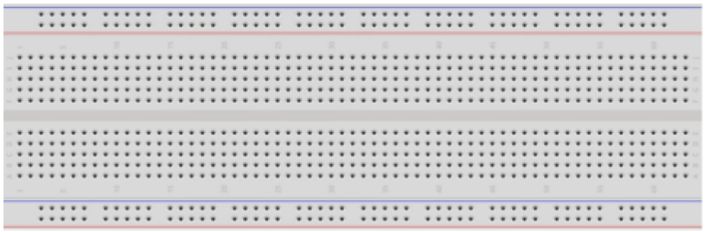
1.3.1 モーター

はじめに

このプロジェクトでは、L293D を使用して DC モーターを制御し、時計回りと反時計回りに回転させる方法を学びます。 DC モーターはより大きな電流が必要ですので、安全のためにここでは電源供給モジュールを使用してモーターに電力を供給します。

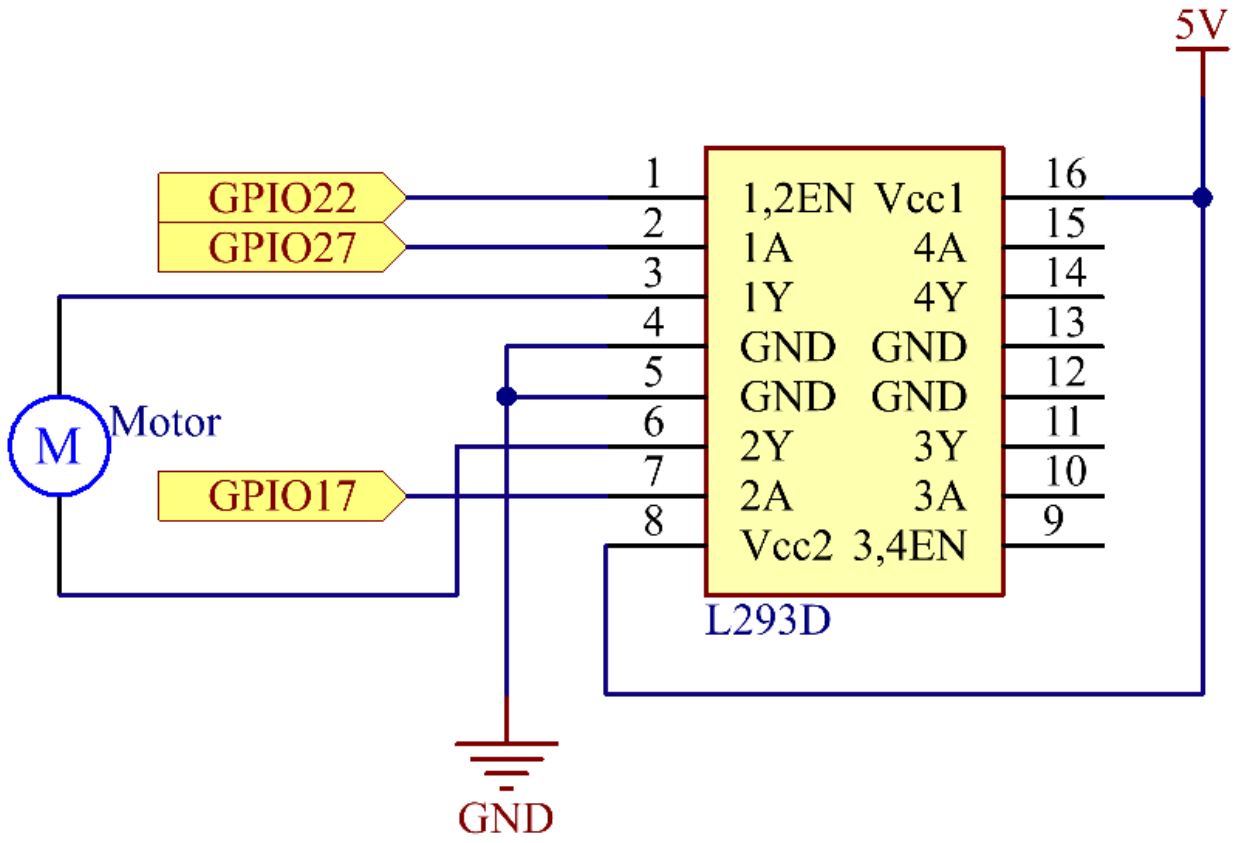
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Power Module (with 9V battery and buckle)
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * L293D
  </td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * DC Motor
 </td></tr>
</table>

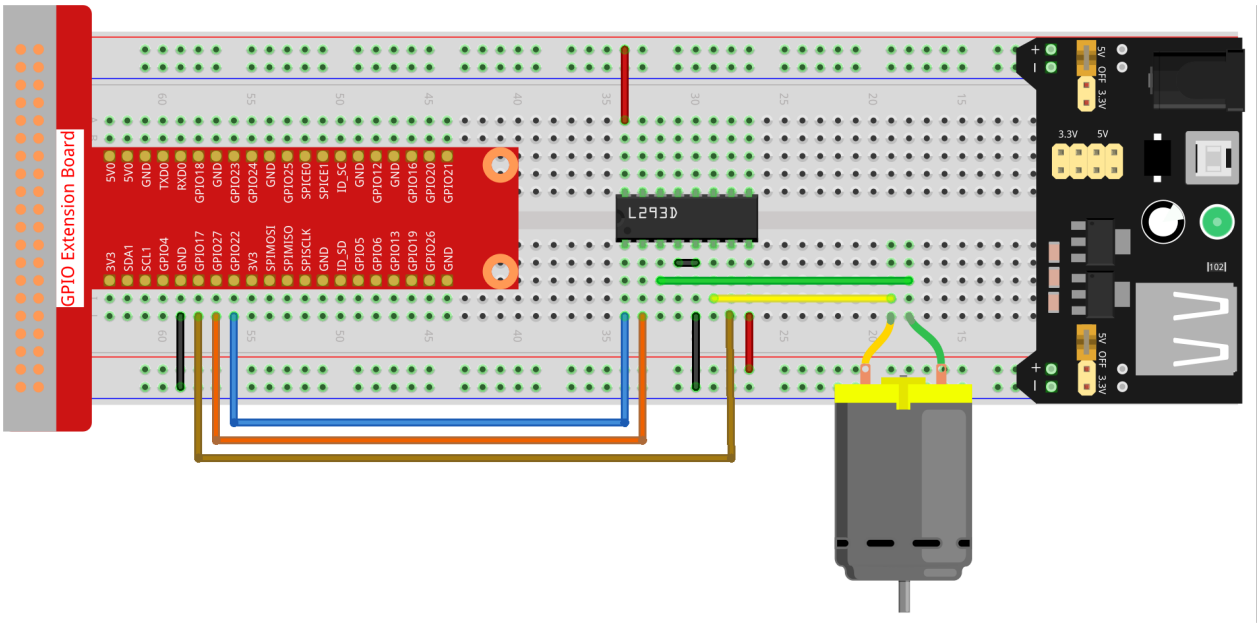
回路図

T-Board Name	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: 回路を組み立てます。



注釈: 電源モジュールには、キットに含まれている 9V バッテリーと 9V バッテリーバックルを使用できます。 電源モジュールのジャンパーキャップをブレッドボードの 5V バスストリップに挿入してください。

ステップ 2: コードのフォルダに入ります。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 1.3.1_Motor_zero.py
```

コードが実行されると、モーターは最初に時計回りに 5 秒間回転し、5 秒間停止し、その後、反時計回りに 5 秒間回転し、その後、モーターは 5 秒間停止します。この一連の動作は繰り返し実行されます。

コード

注釈: 以下のコードは変更/リセット/コピー/実行/停止ができます。ただし、コードを変更する前に davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import Motor
from time import sleep

# GPIO Zero を使用してモーターを初期化し、前進（17）、後退（27）、および有効（22）制御のための GPIO ピンを指定します。
motor = Motor(forward=17, backward=27, enable=22)

try:
    # モーターの方向と動きを制御するためのメイン関数です。
    # モーターの回転を時計回りと反時計回りに交互に切り替えます。
    actions = {'CW': motor.forward, 'CCW': motor.backward, 'STOP': motor.stop}  # 読みやすさのためにモーターアクションを定義

    while True:
        # モーターの方向を制御するために定義されたアクションをループします
        for action in ['CW', 'STOP', 'CCW', 'STOP']:
            actions[action]()  # 現在のアクション（前進、停止、後退、停止）を実行
            print(f"{action}")  # コンソールに現在のアクションを表示
            sleep(5)  # 次のアクションに進む前に 5 秒間停止

except KeyboardInterrupt:
    # プログラムを停止するためのキーボード割り込み（Ctrl+C など）を正常に処理します
    pass
```

コードの説明

1. これらのインポートステートメントは、 gpiozero ライブラリから Motor クラスと、 time モジュールから sleep 関数を取り込みます。

```
#!/usr/bin/env python3
from gpiozero import Motor
from time import sleep
```

2. この行は、前進（17）、後退（27）、および有効（22）制御のための GPIO ピンを指定して Motor オブジェクトを初期化します。

```
# Initialize the Motor with GPIO Zero, specifying GPIO pins for forward (17),␣
↪backward (27), and enable (22) control
motor = Motor(forward=17, backward=27, enable=22)
```

3. モーター制御のアクションは、可読性のために辞書で定義されています。無限ループ（ *while True* ）がこれらのアクションを繰り返し、各アクションを 5 秒間実行します。

```
try:
    # モーターの方向と動きを制御するメイン関数。
    # 時計回りと反時計回りのモーターの回転を交互に行い、間に停止します。
    actions = {'CW': motor.forward, 'CCW': motor.backward, 'STOP': motor.stop}  # 可
読性のためにモーターアクションを定義

    while True:
        # ループを使用して定義されたアクションを制御してモーターの方向を切り替えます
        for action in ['CW', 'STOP', 'CCW', 'STOP']:
            actions[action]()  # 現在のアクション（前進、停止、後退、停止）を実行します
            print(f"{action}")  # 現在のアクションをコンソールに表示します
            sleep(5)  # 次のアクションに進む前に 5 秒間一時停止します
```

4. このセグメントは、キーボード割り込み（Ctrl+C など）を使用してプログラムを安全に終了できるようにします。

```
except KeyboardInterrupt:
# キーボード割り込み（e.g., Ctrl+C）をスムーズに処理してプログラムを停止します
pass
```

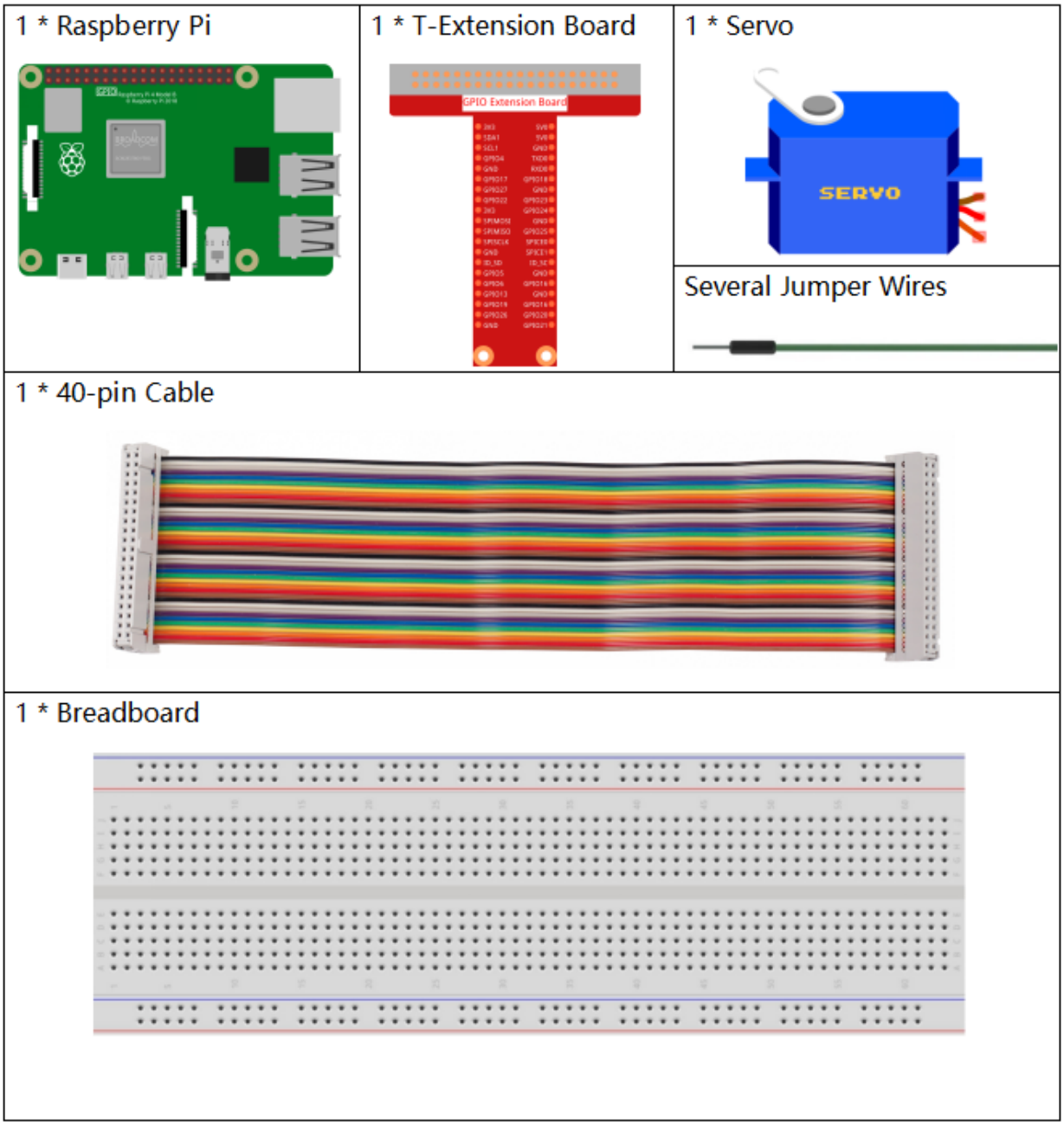
1.3.2 サーボ

はじめに

このプロジェクトでは、サーボを回転させる方法を学びます。

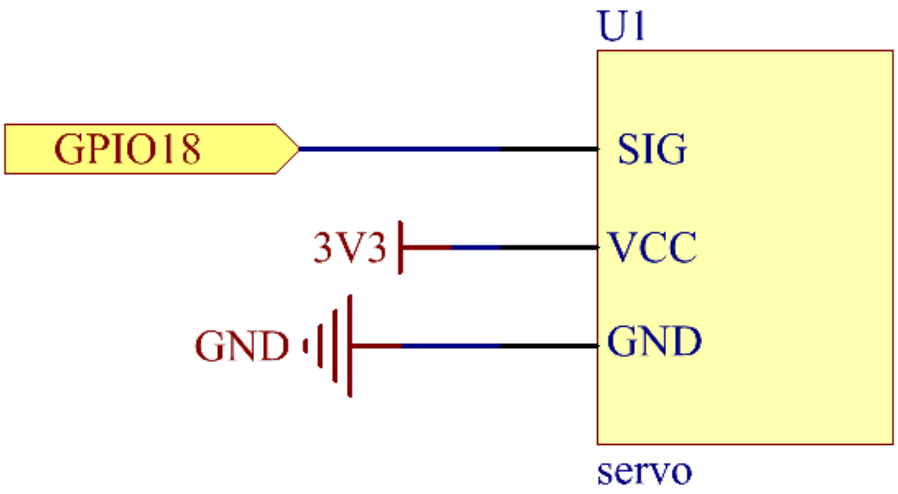
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。



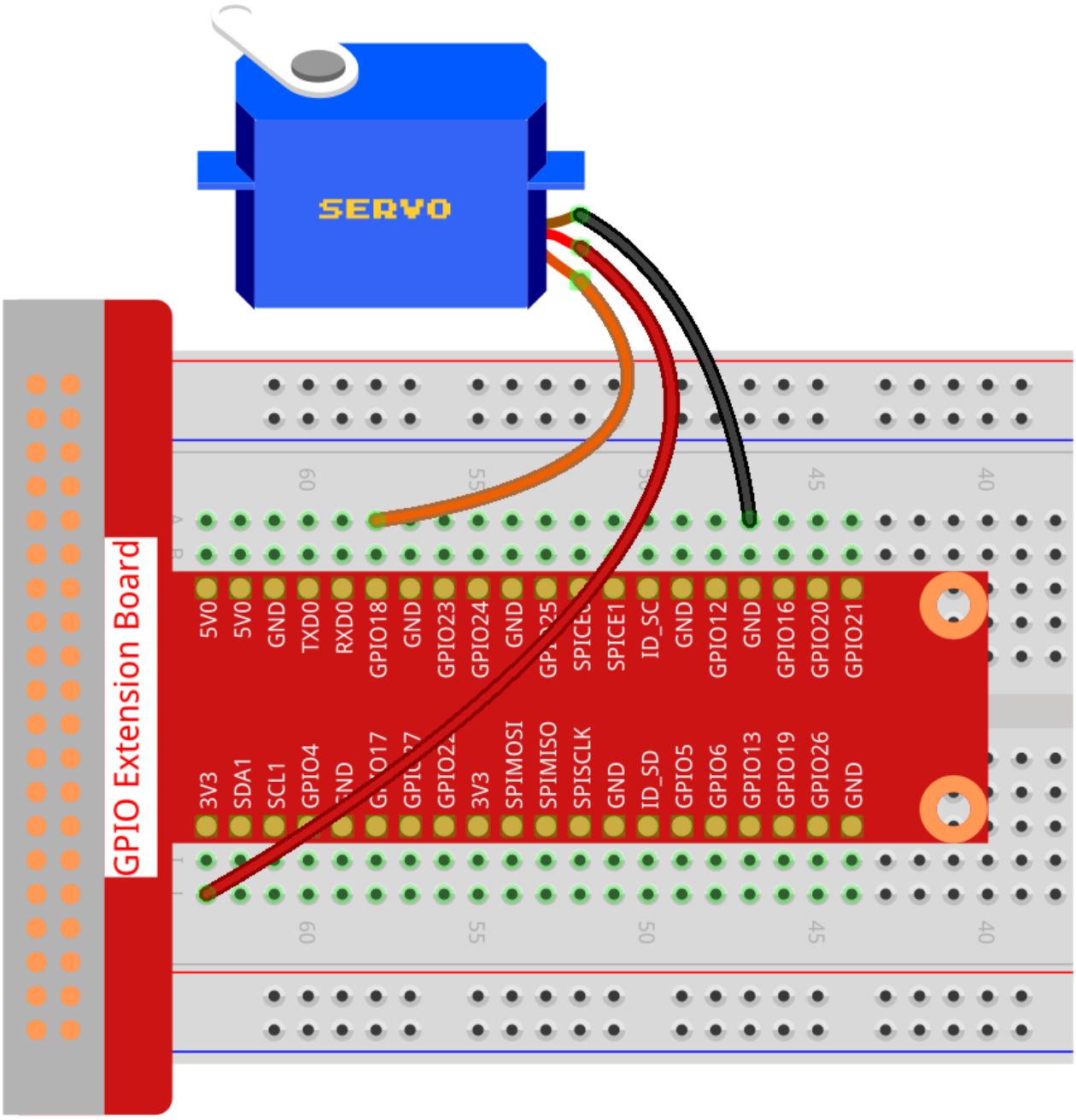
回路図

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 1.3.2_Servo_zero.py
```

プログラムが実行されると、サーボは 0 度から 90 度、180 度まで回転し、次に 180 度から 90 度、0 度まで回転し

ます。これが繰り返されます。

コード

注釈: 以下のコードは変更/リセット/コピー/実行/停止ができます。ただし、コードを変更する前に davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認できます。

```
#!/usr/bin/env python3
from gpiozero import Servo
from time import sleep

# サーボモーターを接続する GPIO ピン番号を設定
myGPIO = 18

# サーボのパルス幅を微調整するための補正係数を定義し、それを使用して最大および最小パルス幅を計算します
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # 最大パルス幅を計算
minPW = (1.0 - myCorrection) / 1000  # 最小パルス幅を計算

# カスタムパルス幅でサーボオブジェクトを初期化
servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)

try:
    while True:
        # サーボを中央に位置させて待機
        servo.mid()
        print("mid")  # 現在の位置を表示
        sleep(0.5)    # 0.5 秒の一時停止

        # サーボを最小位置に移動させて待機
        servo.min()
        print("min")  # 現在の位置を表示
        sleep(1)      # 1 秒間位置を保持

        # サーボを中央の位置に戻して待機
        servo.mid()
        print("mid")  # 現在の位置を表示
        sleep(0.5)    # 0.5 秒の一時停止
```

(次のページに続く)

(前のページからの続き)

```
        # サーボを最大位置に移動させて待機
        servo.max()
        print("max")  # 現在の位置を表示
        sleep(1)      # 1 秒間位置を保持

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C）でスクリプトを正常に終了する
    pass
```

コードの説明

1. これらのインポート文は、サーボ制御のための Servo クラスとタイミングのための sleep 関数を取り込みます。

```
#!/usr/bin/env python3
from gpiozero import Servo
from time import sleep
```

2. サーボモーターを接続するための GPIO ピン番号を 18 に設定します。

```
# サーボモーターが接続されている GPIO ピン番号を設定します
myGPIO = 18
```

3. これらの行は、補正係数を定義し、それを使用してサーボの最大および最小パルス幅を計算します。これにより、サーボの動きの範囲を微調整できます。

```
# サーボのパルス幅を微調整するための補正係数を定義し、最大パルス幅と最小パルス幅を計算します
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # 最大パルス幅を計算
minPW = (1.0 - myCorrection) / 1000  # 最小パルス幅を計算
```

4. カスタムパルス幅でサーボオブジェクトを初期化します。

```
# カスタムパルス幅でサーボオブジェクトを初期化します
servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)
```

5. try ブロックには、サーボを連続的に動かすための while True ループが含まれています。サーボは中間位置、最小位置、最大位置に配置され、各位置が表示され、指定された期間保持されます。

```
try:
    while True:
        # サーボを中間位置に配置し、待機
        servo.mid()
        print("mid")  # 現在の位置を表示
        sleep(0.5)    # 0.5 秒の短い休憩

        # サーボを最小位置に移動し、待機
        servo.min()
        print("min")  # 現在の位置を表示
        sleep(1)      # 1 秒間位置を保持

        # サーボを中間位置に戻し、待機
        servo.mid()
        print("mid")  # 現在の位置を表示
        sleep(0.5)    # 0.5 秒の短い休憩

        # サーボを最大位置に移動し、待機
        servo.max()
        print("max")  # 現在の位置を表示
        sleep(1)      # 1 秒間位置を保持

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）でスクリプトを正常に終了します
    pass
```

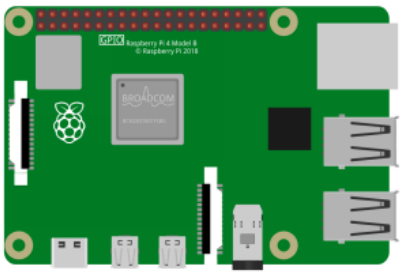
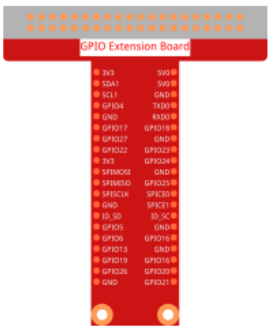
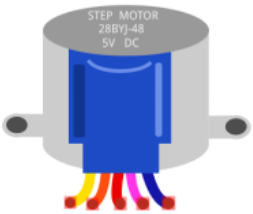
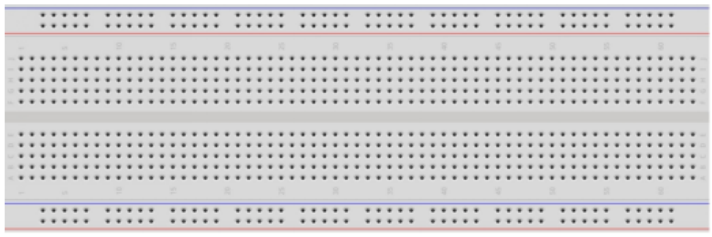
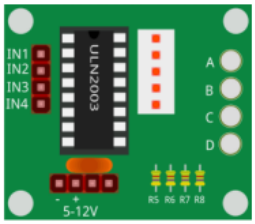
1.3.3 ステッパーモーター

はじめに

ステッパーモーターは、その独自の設計により、フィードバックメカニズムなしで高い精度で制御できます。ステッパーモーターのシャフトは、一連の磁石で取り付けられ、正および負に充電される一連の電磁コイルによって制御され、特定の順序で正確に前方または後方に小さな「ステップ」で移動します。

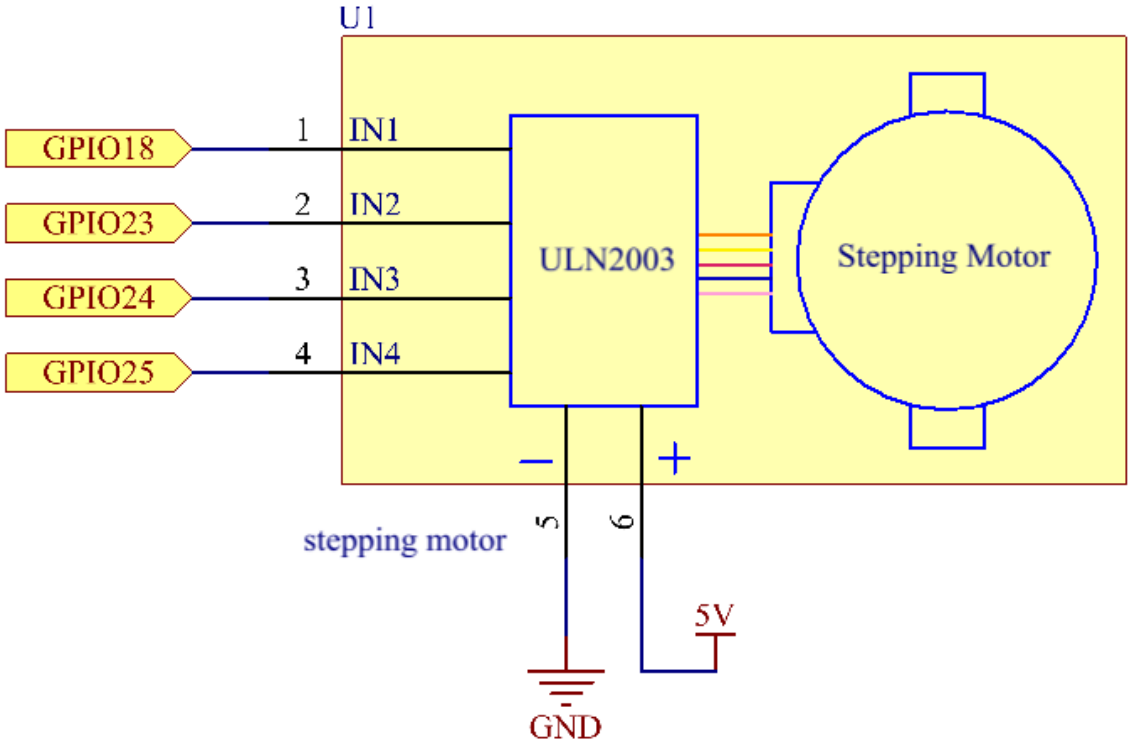
必要な部品

このプロジェクトでは、次の部品が必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Stepper Motor 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * ULN2003 

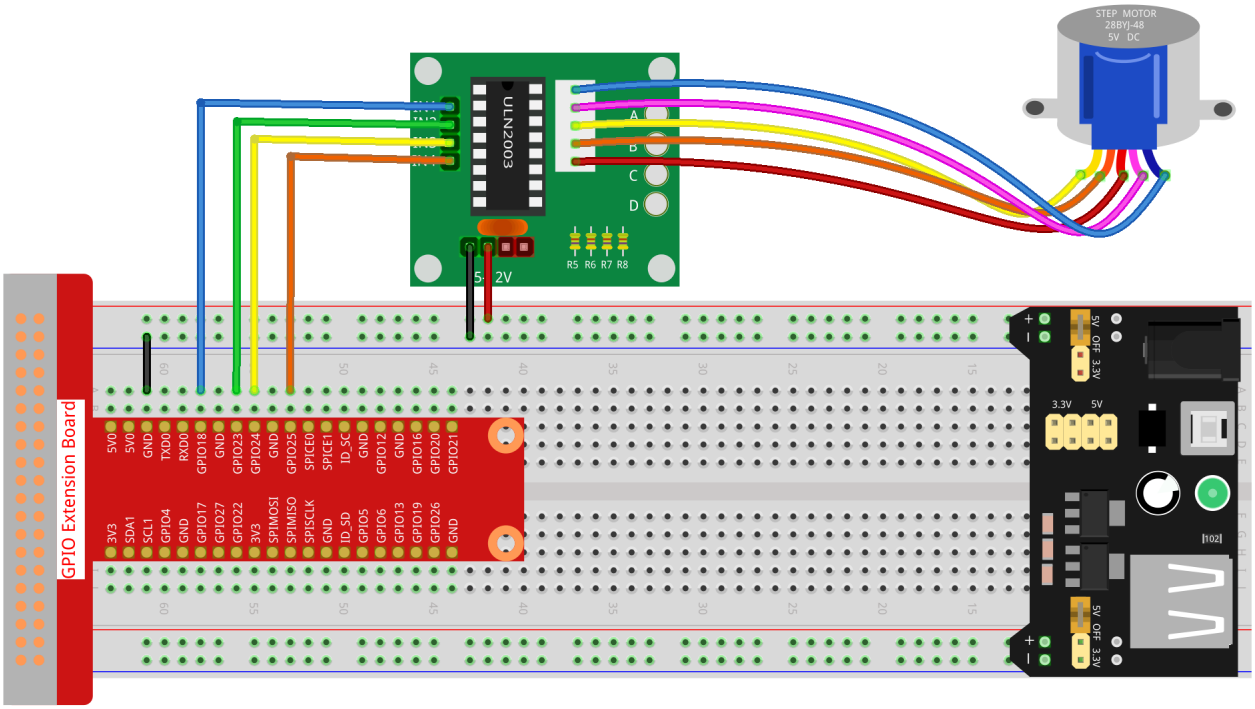
回路図

T-Board Name	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： 回路を組み立てます。



ステップ 2： コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3： 実行します。

```
sudo python3 1.3.3_StepperMotor_zero.py
```

コードが実行されると、ステッパーモーターは、 'a' または 'c' の入力に応じて時計回りまたは反時計回りに回転します。

コード

注釈: 以下のコードを変更、リセット、コピー、実行、停止することができます。ただし、それを行う前に、ソースコードのパスである davinci-kit-for-raspberry-pi/python-pi5 に移動する必要があります。コードを変更した後、その効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep

# ステッパーモーターを制御するために GPIO ピン 18、23、24、25 を初期化します
```

(次のページに続く)

(前のページからの続き)

```
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]

# モーターの回転速度パラメーターを設定し、スムーズなモーター動作のための各ステップ間の待ち時間を計算します
rotationPerMinute = 15
stepsPerRevolution = 2048
stepSpeed = (60 / rotationPerMinute) / stepsPerRevolution

def rotary(direction):
    """
    指定された方向に基づいてモーターの回転を制御します。

    :param direction: 時計回りの場合は 'c'、反時計回りの場合は 'a'
    """
    if direction == 'c':
        # 時計回りの回転のステップシーケンスを実行します
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(stepSpeed)
    elif direction == 'a':
        # 反時計回りの回転のステップシーケンスを実行します
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(stepSpeed)

def loop():
    """
    ユーザーにモーターの回転方向を選択させ続け、この入力に基づいてモーターを制御します。
    """
    while True:
        direction = input(' モーターの回転方向を選択してください a=反時計回り、c=時計回り: ')
```

(次のページに続く)

(前のページからの続き)

```
        if direction == 'c':
            print(' モーターは時計回りに動作します\n')
            break
        elif direction == 'a':
            print(' モーターは反時計回りに動作します\n')
            break
        else:
            print(' 入力エラー、もう一度やり直してください！')

    # 選択した方向でモーターを連続的に回転させます
    while True:
        rotary(direction)

def destroy():
    """
    安全にすべてのモーターピンをオフにし、クリーンなシャットダウンに使用します。
    """
    for pin in motorPin:
        pin.off()

# メインプログラムの実行
try:
    loop()
except KeyboardInterrupt:
    destroy()  # キーボード割り込みを処理してモーターを安全にシャットダウン
```

コードの説明

1. このセクションでは、必要なライブラリをインポートしています。GPIO ピンの制御には「gpiozero」、タイミング制御には「time」を使用しています。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep
```

2. ステッパーモータを制御するために GPIO ピン 18、23、24、25 を初期化します。

```
# ステッパーモータのモーターピンを初期化します
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]
```

3. モーターの回転速度を設定し、スムーズな動作のために各ステップ間の時間間隔を計算します。

```
# モーターの回転速度パラメータを設定します
rotationPerMinute = 15
stepsPerRevolution = 2048
# スムーズなモーター動作のための各ステップの間隔を計算します
stepSpeed = (60 / rotationPerMinute) / stepsPerRevolution
```

4. 「rotary」関数は、モーターの回転を制御します。ビット操作とステッピングシーケンスを使用して、時計回りまたは反時計回りの回転に適した順序でモーターピンをアクティブにします。

```
def rotary(direction):
    """
    指定された方向に基づいてモーターの回転を制御します。

    :param direction: 'c' は時計回り、'a' は反時計回り
    """
    if direction == 'c':
        # 時計回りの回転用のステッピングシーケンスを実行します
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(stepSpeed)
    elif direction == 'a':
        # 反時計回りの回転用のステッピングシーケンスを実行します
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(stepSpeed)
```

5. 「loop」関数はユーザーにモーターの回転方向を選択するように継続的にプロンプトを表示し、入力に基づいてモーターを制御します。

```
def loop():
    """
```

(次のページに続く)

(前のページからの続き)

```
    ユーザーにモーターの回転方向を選択するように継続的にプロンプトを表示し、この入力に基づいて
モーターを制御します。
    """
    while True:
        direction = input(' モーターの方向を選択してください。a=反時計回り、c=時計回り：')
        if direction == 'c':
            print(' モーターが時計回りに動作中\n')
            break
        elif direction == 'a':
            print(' モーターが反時計回りに動作中\n')
            break
        else:
            print(' 入力エラー、もう一度やり直してください！')

    # 選択した方向にモーターを継続的に回転させます
    while True:
        rotary(direction)
```

6. 「destroy」関数はすべてのモーターピンをオフにします。これはクリーンなシャットダウンのために使用され、プログラムが終了したときにモーターが安全に停止することを保証します。

```
def destroy():
    """
    クリーンなシャットダウンに使用される、すべてのモーターピンを安全にオフにします。
    """
    for pin in motorPin:
        pin.off()
```

7. メインプログラムは「loop」を呼び出し、キーボード割り込み（Ctrl+C など）を処理して「destroy」を使用してモーターを安全にシャットダウンします。

```
# メインプログラムの実行
try:
    loop()
except KeyboardInterrupt:
    destroy()  # キーボード割り込みを処理してモーターを安全にシャットダウンします
```

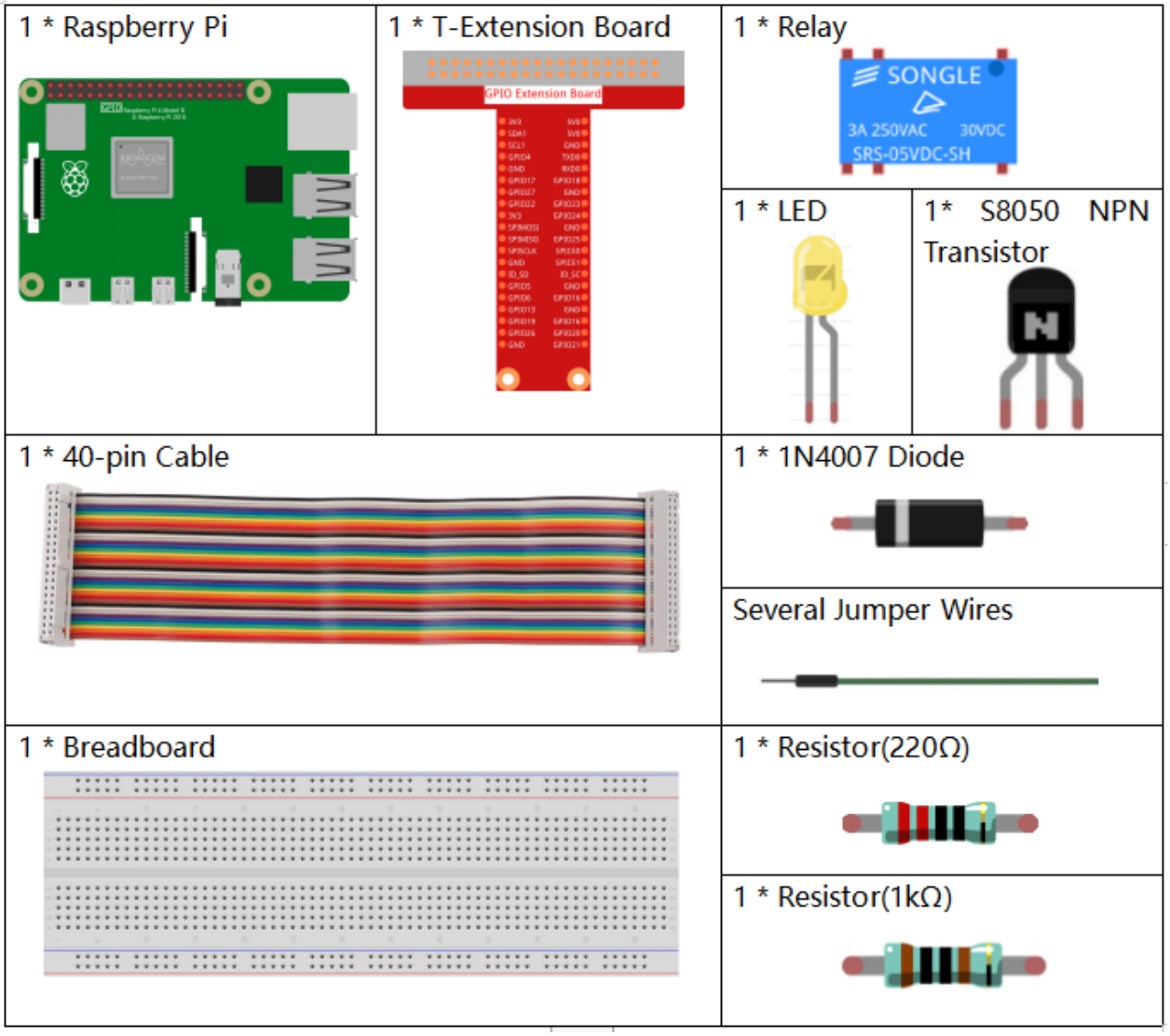
1.3.4 リレー

はじめに

このプロジェクトでは、リレーの使用方法を学びます。リレーは、自動制御システムで一般的に使用されるコンポーネントの 1 つです。電圧、電流、温度、圧力などが予め設定された値に達したり、超えたり、低下したりした場合、リレーは回路を接続または切断して、機器を制御および保護します。

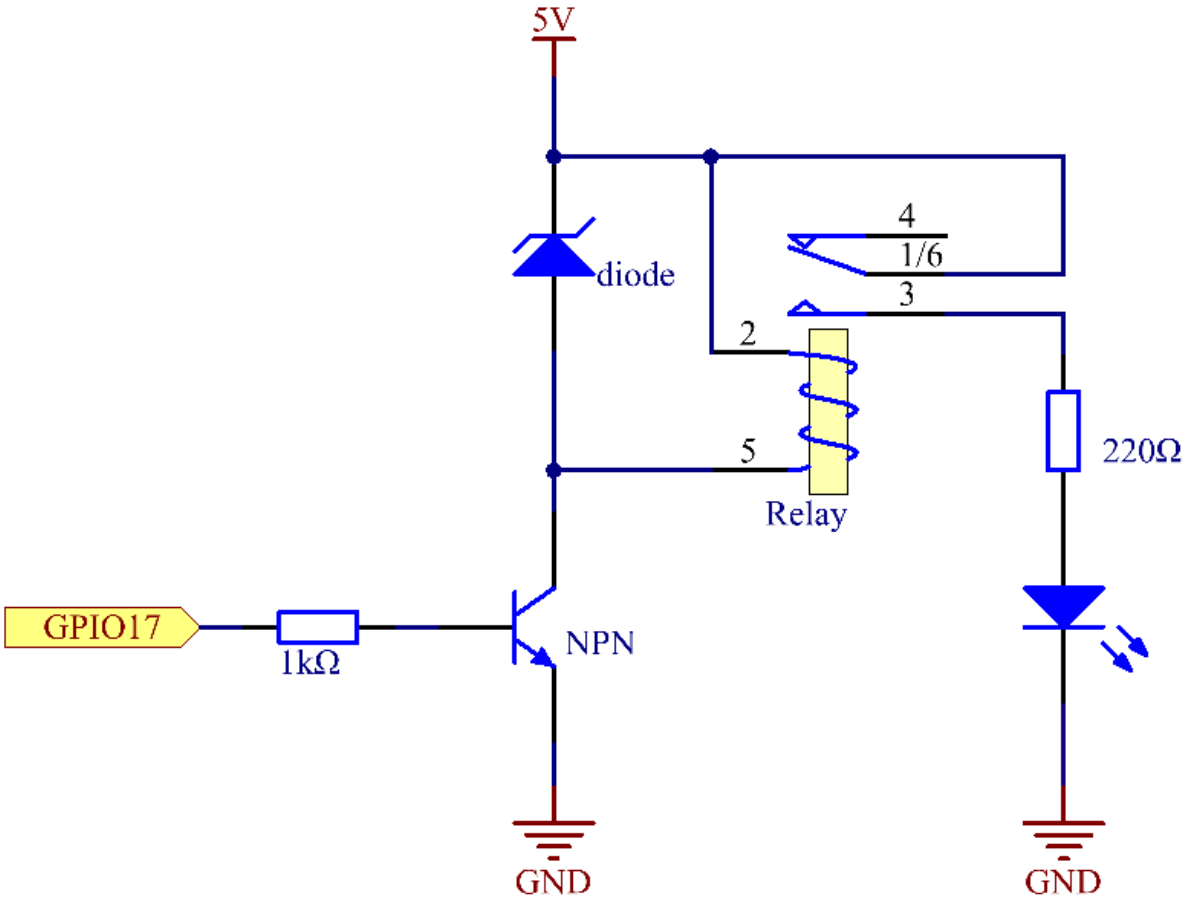
必要な部品

このプロジェクトでは、次の部品が必要です。



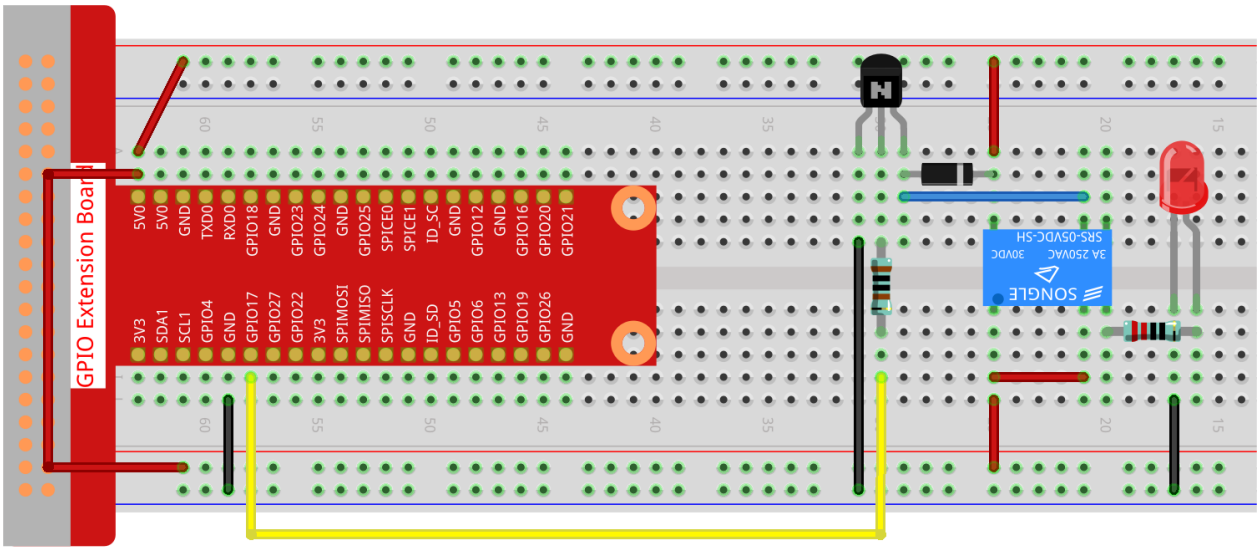
回路図

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



実験手順

ステップ 1： 回路を組み立てます。



ステップ 2： コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3： 実行します。

```
sudo python3 1.3.4_Relay_zero.py
```

コードが実行されている間、LED が点灯します。さらに、通常閉じている接点が切断され、通常開いている接点が閉じることによるチクタク音が聞こえます。

コード

注釈: 以下のコードを変更、リセット、コピー、実行、停止することができます。ただし、それを行う前に、ソースコードのパスである davinci-kit-for-raspberry-pi/python-pi5 に移動する必要があります。コードを変更した後、その効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice  # GPIO ピンを制御するためのクラスをインポート
from time import sleep  # 遅延用に sleep 関数をインポート

# GPIO ピン 17 に接続されたリレーを初期化します
relay = OutputDevice(17)

try:
    # リレーの状態を 1 秒ごとに切り替え続けるループ
```

(次のページに続く)

(前のページからの続き)

```
    while True:
        print(' リレーをオープンにします...')  # リレーがアクティブ化されていることを通知
        relay.on()  # リレーをオンにします（アクティブロー構成を想定）
        sleep(1)    # リレーを 1 秒間オンの状態に保持

        print('... リレーをクローズにします')  # リレーが非アクティブ化されていることを通知
        relay.off()  # リレーをオフにします
        sleep(1)    # リレーを 1 秒間オフの状態に保持

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）を処理してループを終了
    relay.off()  # スクリプトを終了する前にリレーをオフにする
    pass
```

コードの説明

1. このセクションでは、必要なライブラリをインポートしています。GPIO ピンを制御するために gpiozero から OutputDevice を、時間制御のために time から sleep をインポートしています。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice  # GPIO ピンを制御するためのクラスをインポート
from time import sleep  # 遅延用に sleep 関数をインポート
```

2. GPIO ピン 17 に接続されたリレーを制御するための OutputDevice オブジェクトを初期化しています。

```
# GPIO ピン 17 に接続されたリレーを初期化します
relay = OutputDevice(17)
```

3. try ブロック内で、無限ループ while True がリレーの状態を繰り返し切り替えます。リレーは、各状態の間に 1 秒の遅延があり、コンソールにプリントされたステートメントが表示されます。

```
try:
    # リレーの状態を 1 秒ごとに切り替え続けるループ
    while True:
        print(' リレーをオープンにします...')  # リレーがアクティブ化されていることを通知
        relay.on()  # リレーをオンにします（アクティブロー構成を想定）
        sleep(1)    # リレーを 1 秒間オンの状態に保持

        print('... リレーをクローズにします')  # リレーが非アクティブ化されていることを通知
        relay.off()  # リレーをオフにします
```

(次のページに続く)

(前のページからの続き)

```
        sleep(1)    # リレーを 1 秒間オフの状態に保持
```

4. KeyboardInterrupt（Ctrl+C など）をキャッチして、スクリプトを正常に終了する前にリレーをオフにします。

```
except KeyboardInterrupt:
  # キーボード割り込み（Ctrl+C など）を処理してループを終了
  relay.off()  # スクリプトを終了する前にリレーをオフにする
  pass
```

1.12.3 2 入力

2.1 コントローラー

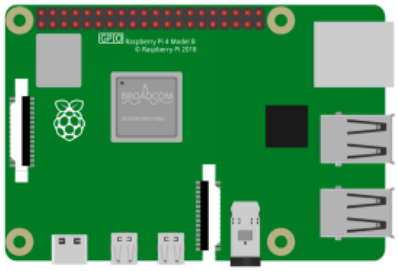
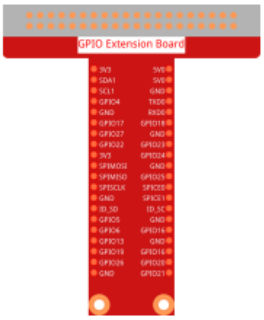
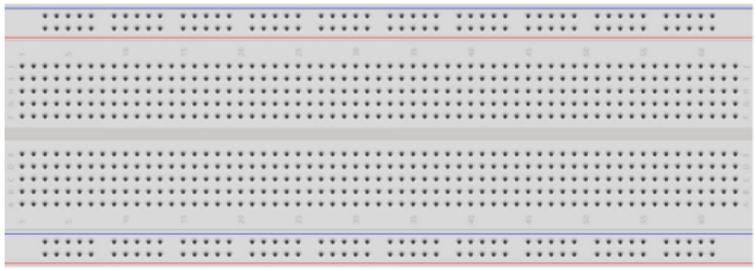
2.1.1 ボタン

はじめに

このプロジェクトでは、ボタンを使用して LED をオンまたはオフにする方法を学びます。

必要な部品

このプロジェクトには、以下の部品が必要です。

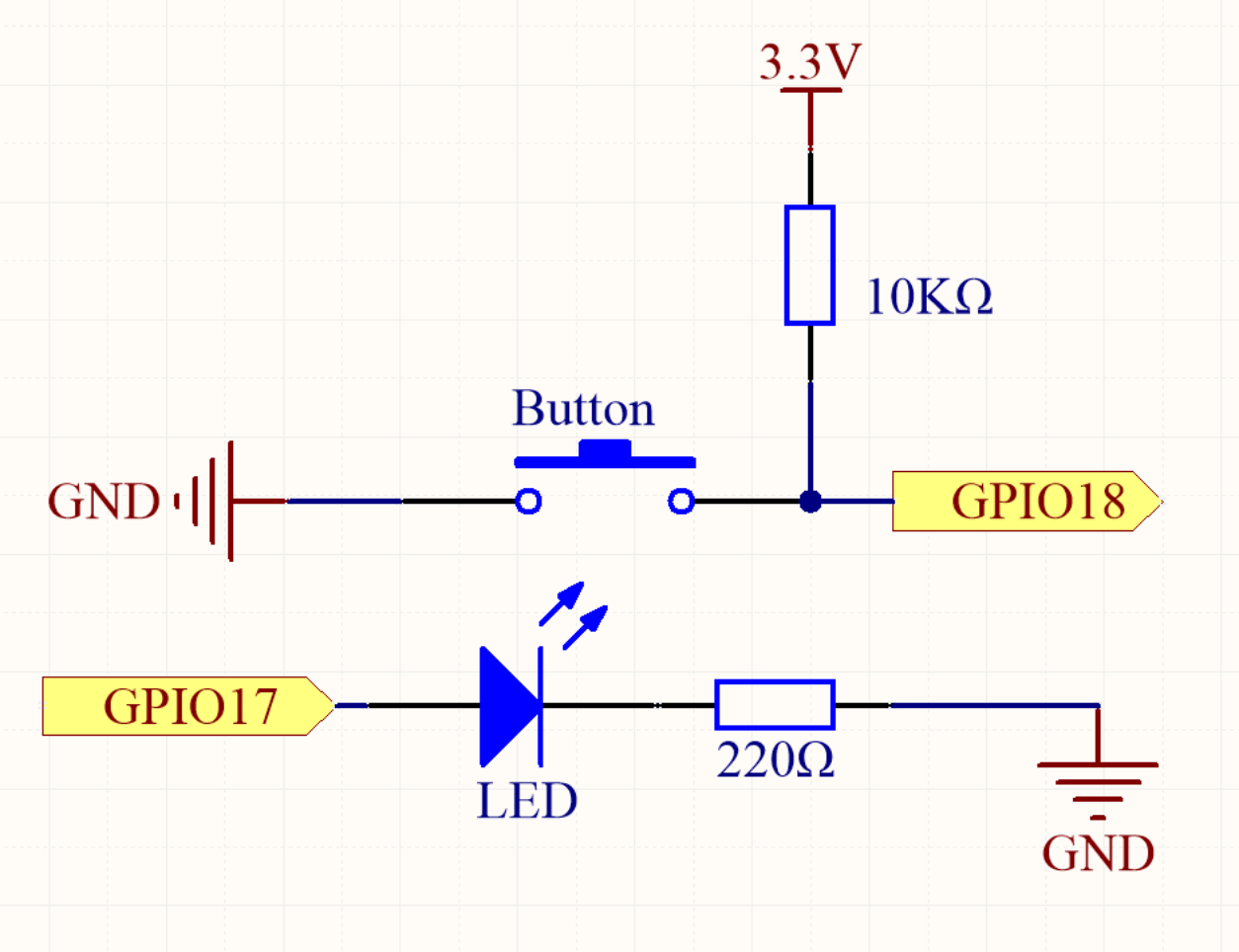
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED
		1 * Resistor 10KΩ
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * Button

回路図

Raspberry Pi の入力として通常開放ボタンを使用し、接続は以下の回路図に示す通りです。ボタンが押されると、GPIO18 は高レベル（3.3V）になります。プログラムを介して GPIO18 の状態を検出できます。つまり、GPIO18 が高レベルになると、ボタンが押されたことを意味します。ボタンが押されたときに対応するコードを実行し、LED が点灯します。

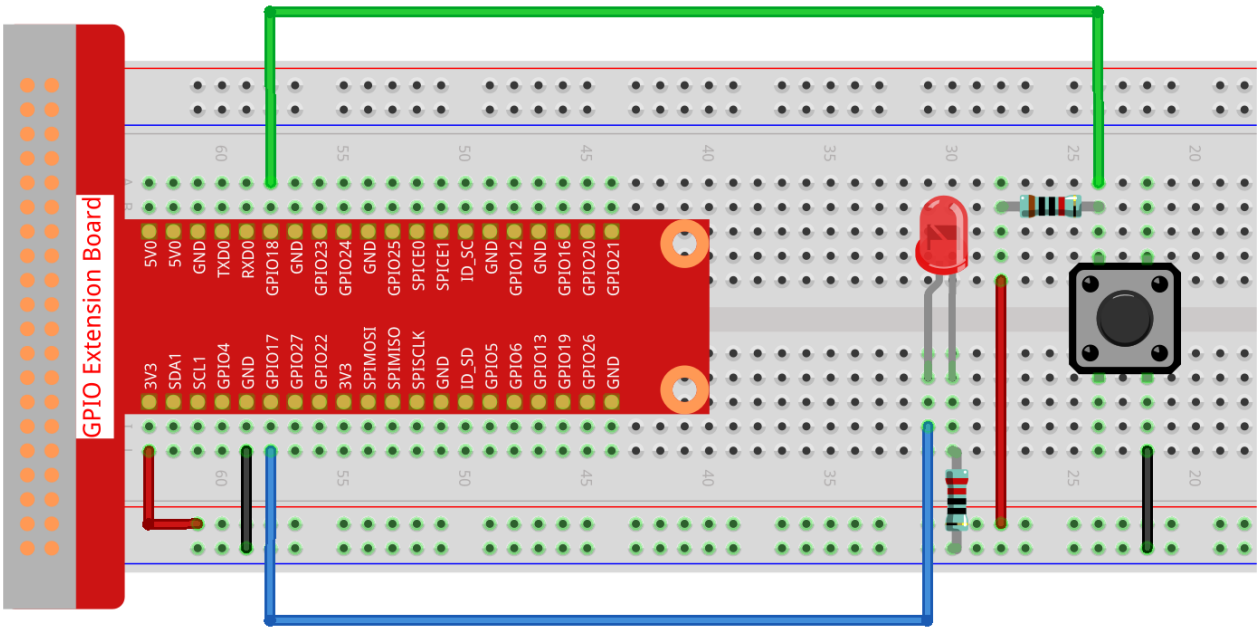
注釈: LED の長いピンが陽極であり、短いピンが陰極です。

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: コードを実行します。

```
sudo python3 2.1.1_Button_zero.py
```

ボタンを押すと LED が点灯し、ボタンを離すと LED が消灯します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、それに先立ち、davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を直接確認するために実行できます。

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # gpiozero から LED と Button クラスをインポート
from signal import pause  # signal モジュールから pause 関数をインポート

# GPIO ピン 17 に接続された LED オブジェクトを初期化
led = LED(17)
# GPIO ピン 18 に接続された Button オブジェクトを初期化
button = Button(18)
```

(次のページに続く)

(前のページからの続き)

```
# ボタンの「when_pressed」イベントを LED の「on()」メソッドにリンク
button.when_pressed = led.on
# ボタンの「when_released」イベントを LED の「off()」メソッドにリンク
button.when_released = led.off

# ボタンのイベントを待ち続け、スクリプトを実行し続けるイベントループを実行
pause()
```

コードの説明

1. このスクリプトは Python3 で書かれており、gpiozero ライブラリから LED および Button クラス、signal モジュールから pause をインポートしています。

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # gpiozero から LED と Button クラスをインポート
from signal import pause  # signal モジュールから pause 関数をインポート
```

2. GPIO ピン 17 に接続された LED オブジェクトと GPIO ピン 18 に接続されたボタンオブジェクトを初期化します。

```
# GPIO ピン 17 に接続された LED オブジェクトを初期化
led = LED(17)
# GPIO ピン 18 に接続された Button オブジェクトを初期化
button = Button(18)
```

3. ボタンが押されたときに LED が点灯し、ボタンが離されたときに LED が消灯するようにイベントハンドラを設定します。

```
# ボタンの「when_pressed」イベントを LED の「on()」メソッドにリンク
button.when_pressed = led.on
# ボタンの「when_released」イベントを LED の「off()」メソッドにリンク
button.when_released = led.off
```

4. ボタンの押下およびリリースイベントを待ち続けるために、スクリプトをイベントループ内で実行し続けます。

```
# ボタンのイベントを待ち続け、スクリプトを実行し続けるイベントループを実行
pause()
```

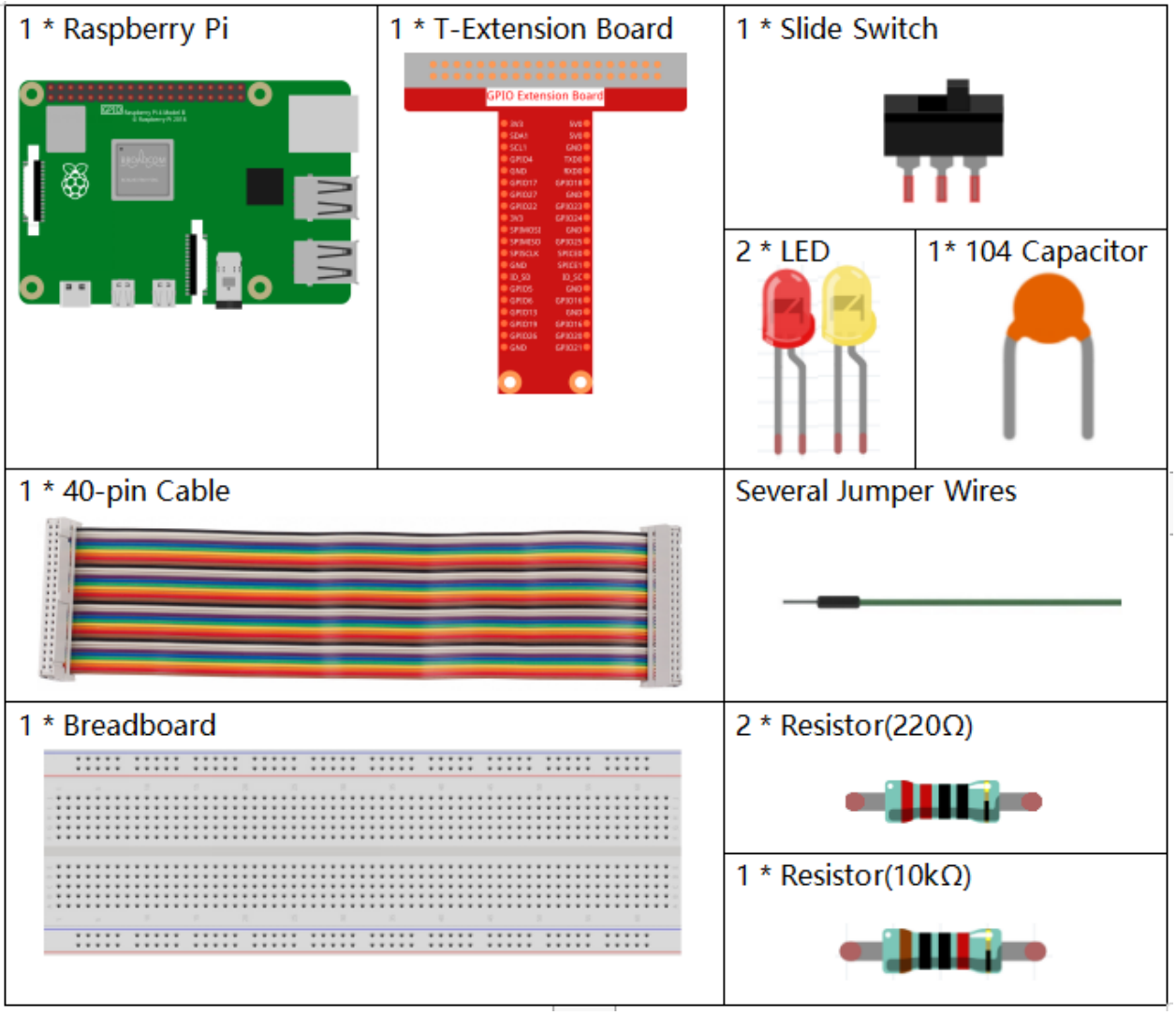
2.1.2 スライドスイッチ

はじめに

このプロジェクトでは、スライドスイッチの使用方法を学びます。通常、スライドスイッチは基板上にはんだ付けされて電源スイッチとして使用されますが、ここではそれをブレッドボードに挿入して使用し、しっかり固定されない場合があります。また、スライドスイッチをブレッドボードに取り付けてその機能を示します。

必要な部品

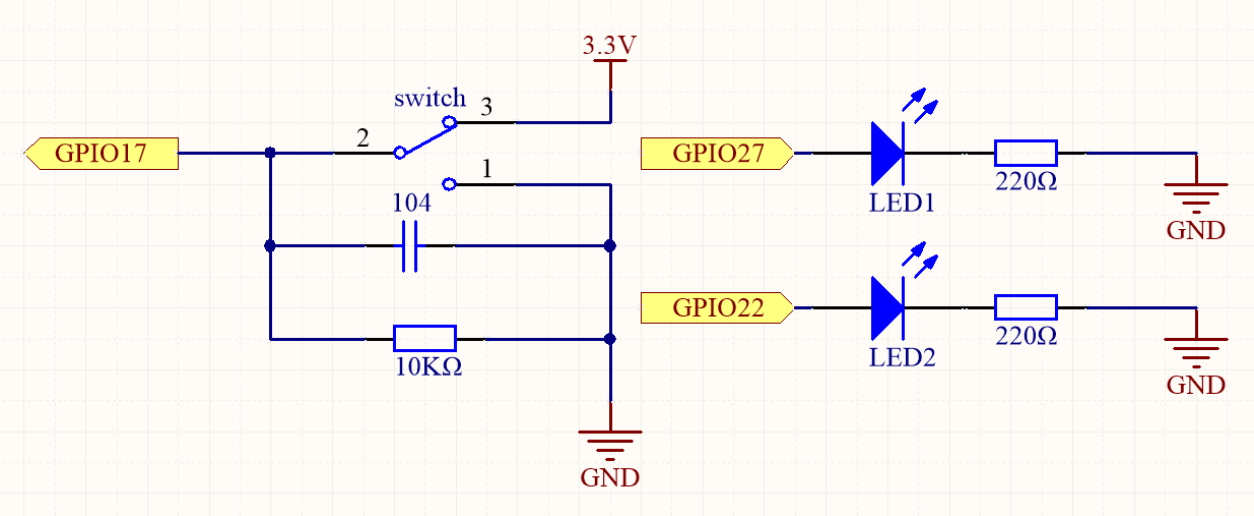
このプロジェクトには、以下の部品が必要です。



回路図

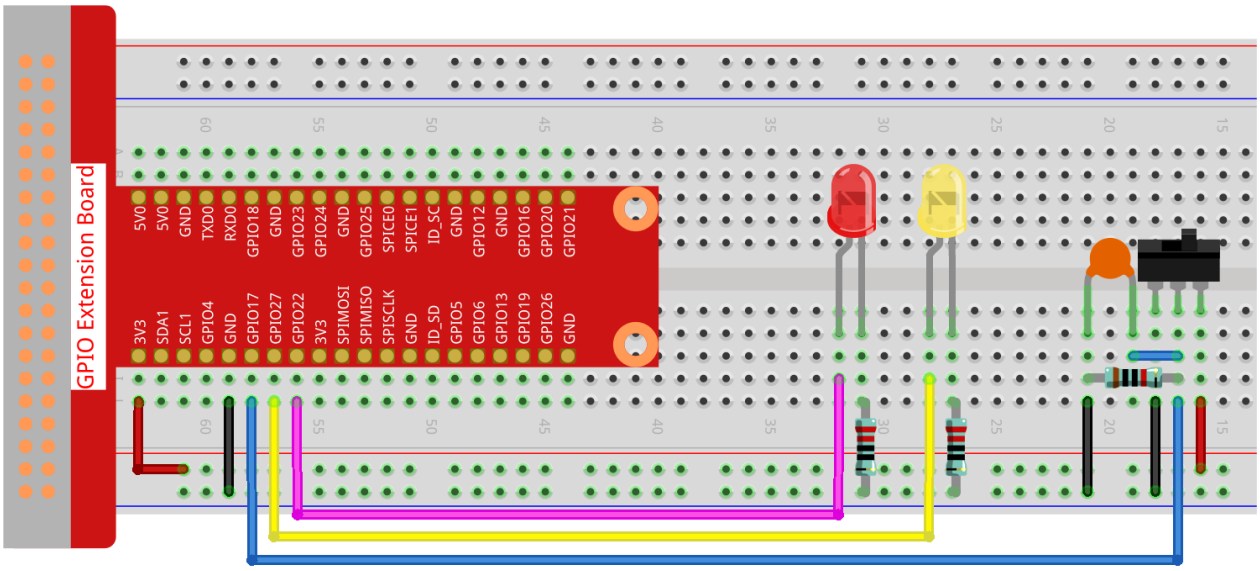
スライドスイッチの中央のピンを GPIO17 に接続し、2 つの LED をそれぞれ GPIO22 と GPIO27 のピンに接続します。そして、スライドを引くと 2 つの LED が交互に点灯するのを確認できます。

T-Board Name	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: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 2.1.2_Slider_zero.py
```

コードが実行されている間、スイッチを左に接続すると黄色の LED が点灯し、右に接続すると赤い LED が点灯します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、それに先立ち、davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を直接確認するために実行できます。

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # GPIO デバイスを制御するために LED と Button クラスをインポート
from time import sleep  # 遅延を追加するために sleep 関数をインポート

# プルアップ抵抗を無効にして GPIO ピン 17 に接続されたマイクロスイッチを初期化
micro_switch = Button(17, pull_up=False)
# GPIO ピン 22 に接続された LED1 を初期化
led1 = LED(22)
```

(次のページに続く)

(前のページからの続き)

```
# GPIO ピン 27 に接続された LED2 を初期化
led2 = LED(27)

try:
    # マイクロスイッチの状態に基づいて LED の状態を制御するメインループ
    while True:
        if micro_switch.is_pressed:  # マイクロスイッチが押されているかどうかを確認
            print('    LED1 ON    ')  # ステータスメッセージを表示
            led1.on()   # LED1 を点灯
            led2.off()  # LED2 を消灯
        else:  # マイクロスイッチが押されていない場合
            print('    LED2 ON    ')  # ステータスメッセージを表示
            led1.off()  # LED1 を消灯
            led2.on()   # LED2 を点灯

        sleep(0.5)  # スイッチの状態を再確認する前に 0.5 秒待機

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）を処理してループからきれいに終了する
    pass
```

コードの説明

1. この行はスクリプトを Python 3 で実行するように設定します。 gpiozero から GPIO デバイスを制御するために LED および Button をインポートし、遅延のために time から sleep をインポートします。

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # GPIO デバイスを制御するために LED と Button クラスをインポート
from time import sleep  # 遅延を追加するために sleep 関数をインポート
```

2. プルアップ抵抗を無効にして GPIO ピン 17 に接続されたマイクロスイッチ、GPIO ピン 22 と 27 に接続された 2 つの LED を初期化します。

```
# プルアップ抵抗を無効にして GPIO ピン 17 に接続されたマイクロスイッチを初期化
micro_switch = Button(17, pull_up=False)
# GPIO ピン 22 に接続された LED1 を初期化
led1 = LED(22)
# GPIO ピン 27 に接続された LED2 を初期化
led2 = LED(27)
```

3. メインループはマイクロスイッチの状態を確認します。押されている場合、LED1 が点灯し、LED2 が消灯します。押されていない場合、LED1 が消灯し、LED2 が点灯します。ループは 0.5 秒ごとに繰り返されます。キーボード割り込み（Ctrl+C など）をキャッチしてスクリプトのきれいな終了を許可します。

```
try:
# マイクロスイッチの状態に基づいて LED の状態を制御するメインループ
while True:
    if micro_switch.is_pressed:  # マイクロスイッチが押されているかどうかを確認
        print('    LED1 ON    ')  # ステータスメッセージを表示
        led1.on()   # LED1 を点灯
        led2.off()  # LED2 を消灯
    else:  # マイクロスイッチが押されていない場合
        print('    LED2 ON    ')  # ステータスメッセージを表示
        led1.off()  # LED1 を消灯
        led2.on()   # LED2 を点灯

    sleep(0.5)  # スイッチの状態を再確認する前に 0.5 秒待機

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）を処理してループからきれいに終了する
    pass
```

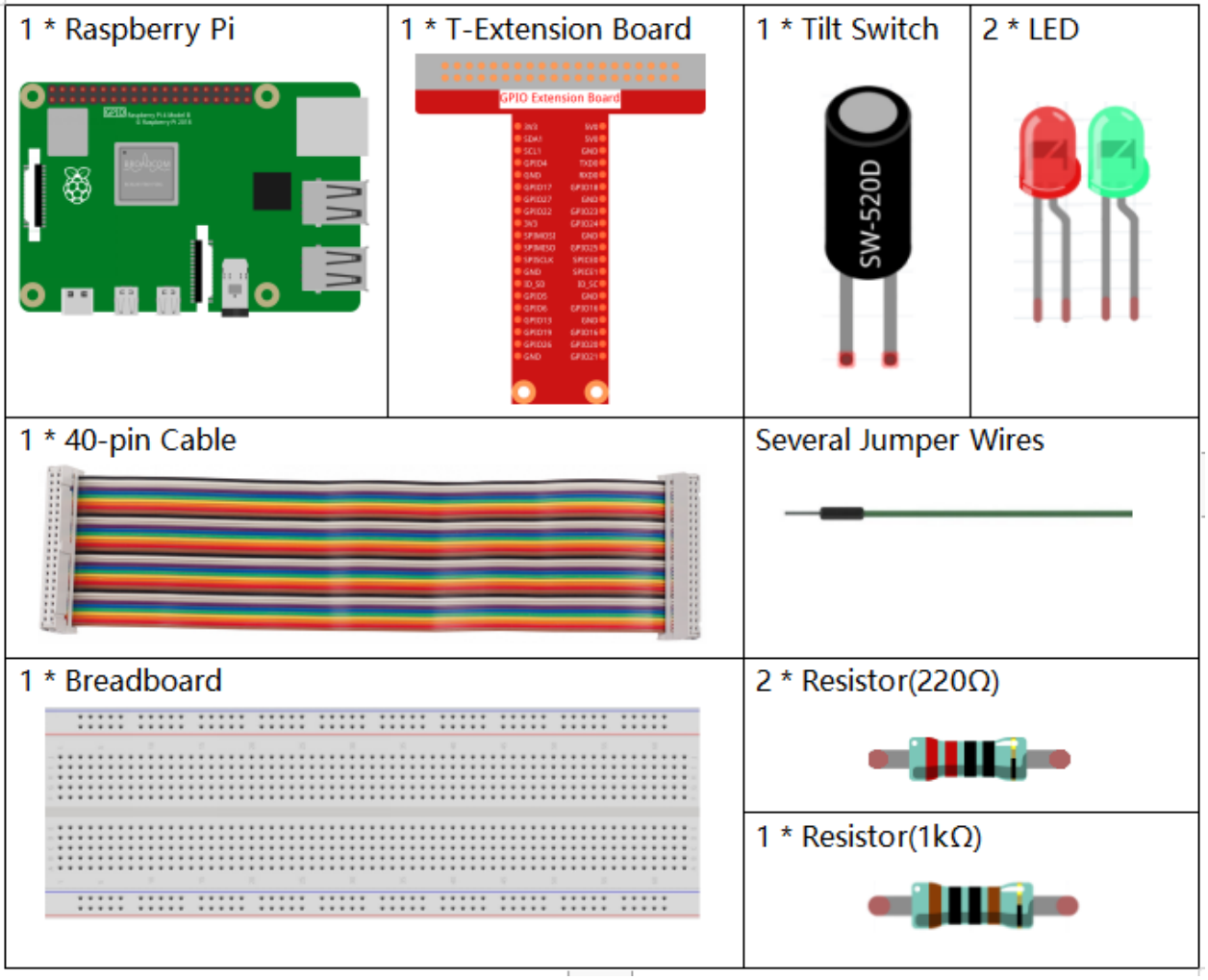
2.1.3 傾斜スイッチ

はじめに

これは内部に金属のボールが入ったボール傾斜スイッチです。小さな角度の傾斜を検出するために使用されます。

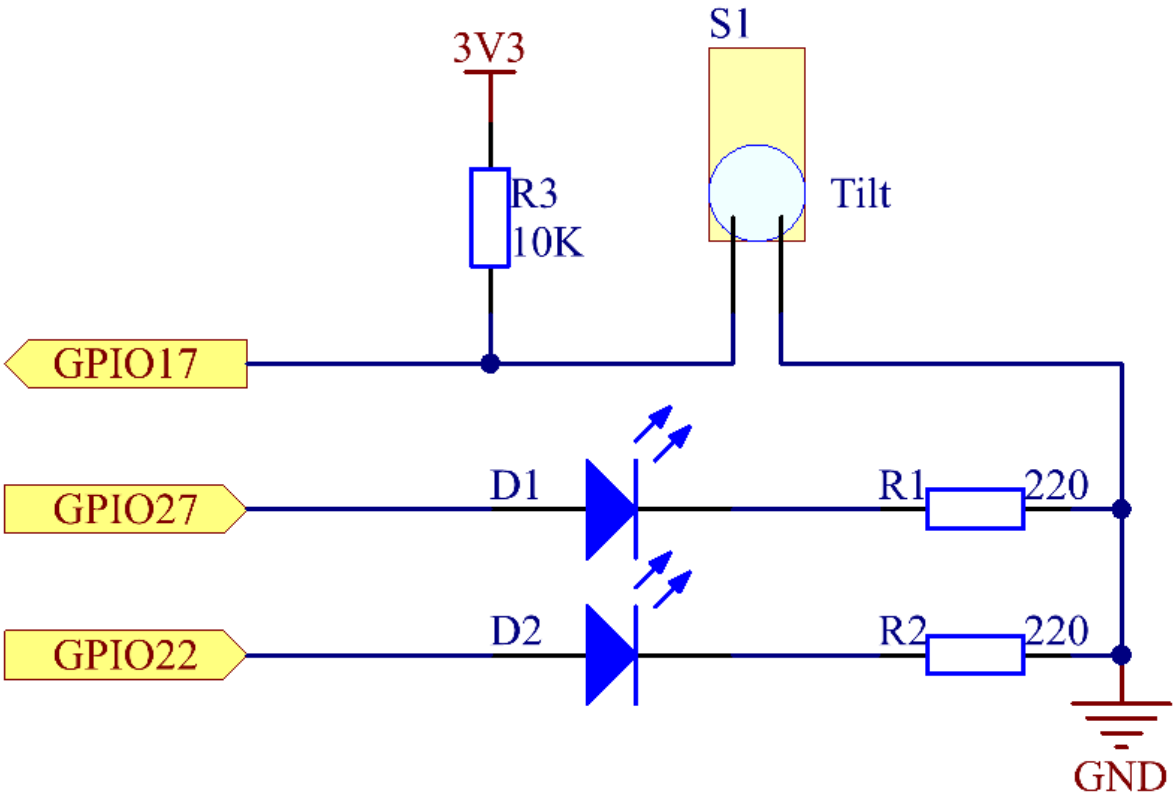
必要な部品

このプロジェクトには、以下の部品が必要です。



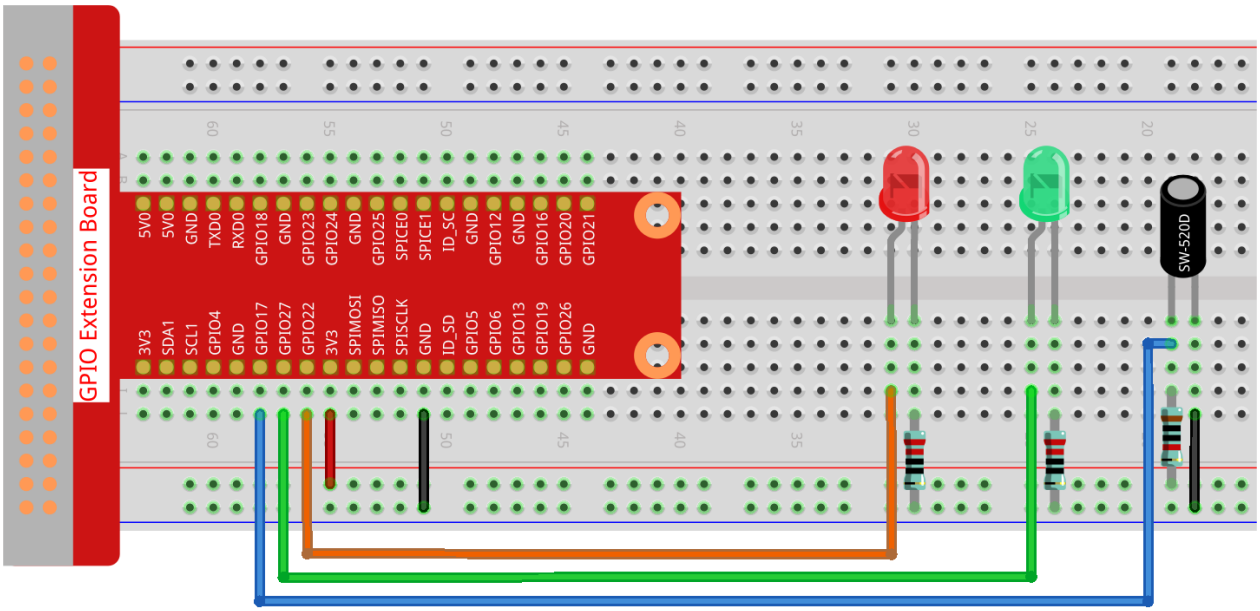
回路図

T-Board Name	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: 回路を組み立てます。



ステップ 2: ディレクトリを変更します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 2.1.3_Tilt_zero.py
```

傾けて垂直に置くと、緑の LED が点灯します。傾けると、画面に「傾けました！」と表示され、赤い LED が点灯します。再び垂直に置くと、緑の LED が点灯します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、それに先立ち、davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を直接確認するために実行できます。

```
#!/usr/bin/env python3
from gpiozero import LED, Button

# GPIO Zero を使用して傾斜センサーと LED を初期化します
TiltPin = Button(17, pull_up=False)  # プルアップ抵抗が無効化された GPIO ピン 17 に接続された傾斜センサーを初期化
green_led = LED(27)  # GPIO ピン 27 に接続された緑色の LED を初期化
red_led = LED(22)    # GPIO ピン 22 に接続された赤色の LED を初期化

def detect():
    """
    傾斜センサーの状態を検出し、LED を制御します。
    傾けた場合、赤い LED を点灯し、緑の LED を消灯します。
    傾けていない場合、その逆を行います。
    """
    if TiltPin.is_pressed:  # センサーが傾いているかどうかを確認
        print('    *************')
        print('    *    傾けました！    *')
        print('    *************')
        red_led.on()   # 赤い LED を点灯
        green_led.off()  # 緑の LED を消灯
    else:  # センサーが傾いていない場合
        red_led.off()  # 赤い LED を消灯
        green_led.on()  # 緑の LED を点灯
```

(次のページに続く)

(前のページからの続き)

```
try:
    while True:
        # 傾斜センサーの状態を継続的に確認し、LED を更新します
        TiltPin.when_pressed = detect
        TiltPin.when_released = detect

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C）を処理してループをきれいに終了します
    pass
```

コードの説明

1. この行はスクリプトを Python 3 で実行するように設定し、 gpiozero から GPIO デバイスを制御するために LED と Button をインポートします。

   ```
   #!/usr/bin/env python3
   from gpiozero import LED, Button
   ```

2. プルアップ抵抗が無効化された GPIO ピン 17 に接続された傾斜センサーと、GPIO ピン 27 および 22 に接続された 2 つの LED を初期化します。

   ```
   # GPIO Zero を使用して傾斜センサーと LED を初期化します
   TiltPin = Button(17, pull_up=False)  # プルアップ抵抗が無効化された GPIO ピン 17 に接続された傾斜センサーを初期化
   green_led = LED(27)  # GPIO ピン 27 に接続された緑色の LED を初期化
   red_led = LED(22)    # GPIO ピン 22 に接続された赤色の LED を初期化
   ```

3. detect 関数を定義し、傾斜センサーの状態をチェックします。傾けた場合、赤い LED を点灯し、緑の LED を消灯します。傾けていない場合、その逆を行います。

   ```
   def detect():
       """
       傾斜センサーの状態を検出し、LED を制御します。
       傾けた場合、赤い LED を点灯し、緑の LED を消灯します。
       傾けていない場合、その逆を行います。
       """
       if TiltPin.is_pressed:  # センサーが傾いているかどうかを確認
           print('    **************')
           print('    *    傾けました！    *')
   ```

(次のページに続く)

(前のページからの続き)

```
        print('    *************')
        red_led.on()    # 赤い LED を点灯
        green_led.off()  # 緑の LED を消灯
    else:  # センサーが傾いていない場合
        red_led.off()  # 赤い LED を消灯
        green_led.on()  # 緑の LED を点灯
```

4. メインループは傾斜センサーの when_pressed と when_released イベントに detect 関数を割り当てます。 try-except ブロックは優雅な終了のためにキーボード割り込みを処理します。

```
try:
    while True:
        # 傾斜センサーの状態を継続的に確認し、LED を更新します
        TiltPin.when_pressed = detect
        TiltPin.when_released = detect

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C）を処理してループをきれいに終了します
    pass
```

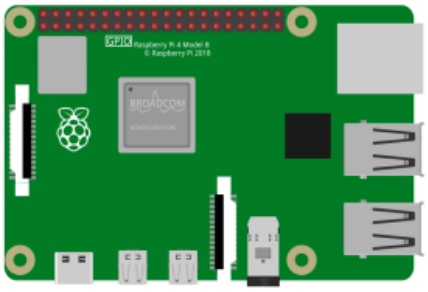
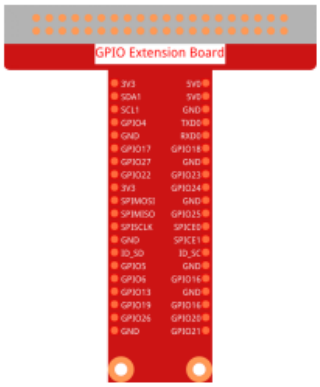
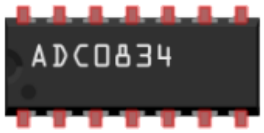
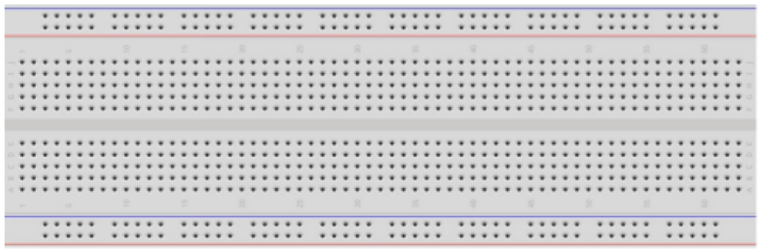
2.1.4 ポテンショメータ

はじめに

ADC（アナログデジタルコンバータ）機能は、アナログ信号をデジタル信号に変換するために使用できます。この実験では、ADC0834 を使用して ADC に関連する機能を取得します。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します。ポテンショメータは物理的な量である電圧を変化させ、ADC 機能によって変換されます。

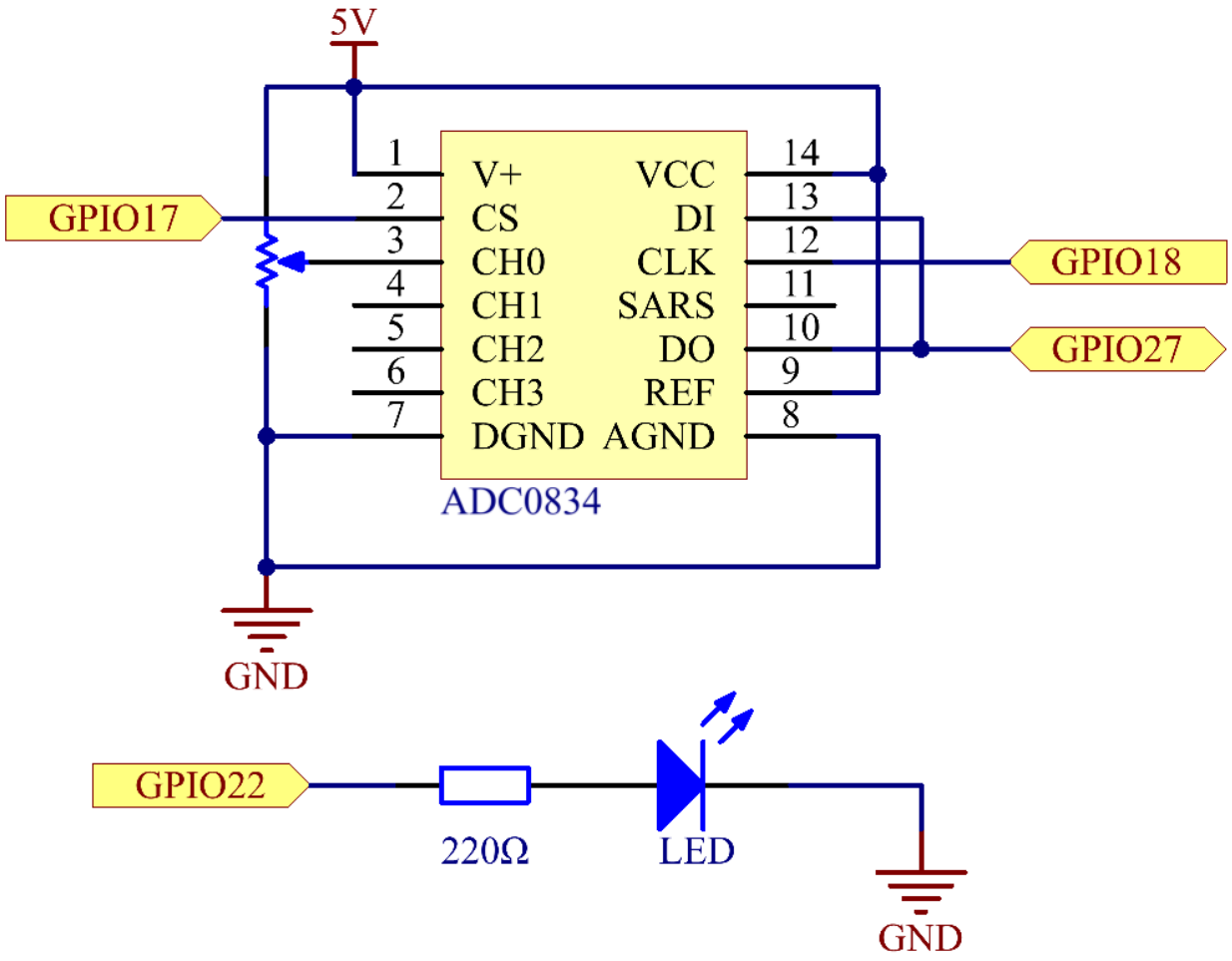
必要な部品

このプロジェクトには、以下の部品が必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Potentiometer
		1 * ADC0834 
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * LED

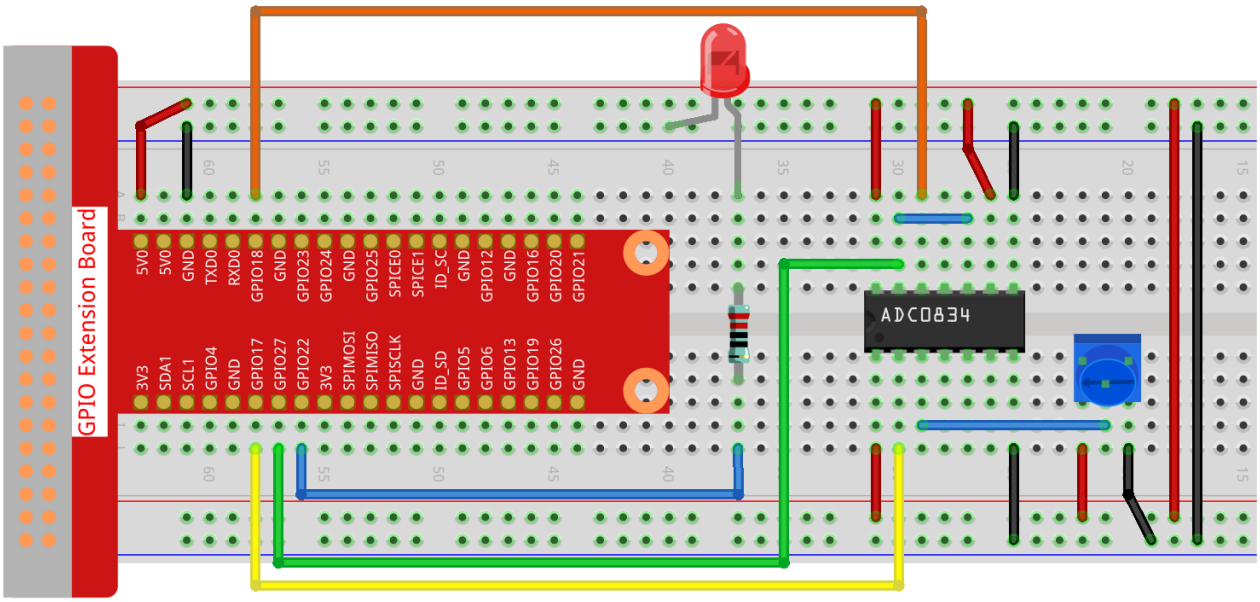
回路図

T-Board Name	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: 回路を組み立てます。



注釈: 画像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。チップの溝は左側にある必要があります。

ステップ 2: コードファイルを開きます。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 2.1.4_Potentiometer_zero.py
```

コードが実行されたら、ポテンショメータのつまみを回すと、LED の輝度がそれに応じて変化します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** できます。ただし、それに先立ち、 raphael-kit/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を直接確認するために実行できます。

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

from gpiozero import PWMLED
import ADC0834
import time

# Initialize a PWM LED on GPIO pin 22
led = PWMLED(22)

# Set up the ADC0834 module
ADC0834.setup()

def MAP(x, in_min, in_max, out_min, out_max):
    """
    Map a value from one range to another.
    :param x: The value to be mapped.
    :param in_min: The lower bound of the value's current range.
    :param in_max: The upper bound of the value's current range.
    :param out_min: The lower bound of the value's target range.
    :param out_max: The upper bound of the value's target range.
    :return: The mapped value.
    """
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

try:
    while True:
        # Get the current reading from the ADC0834 module
        res = ADC0834.getResult()
        print('res = %d' % res)

        # Map the ADC value to a range suitable for setting LED brightness
        R_val = MAP(res, 0, 255, 0, 100)

        # Set the LED brightness
        led.value = float(R_val / 100)

        # Wait for 0.2 seconds before reading again
        time.sleep(0.2)
```

(次のページに続く)

(前のページからの続き)

```
# Graceful exit when 'Ctrl+C' is pressed
except KeyboardInterrupt:
    led.value = 0  # Turn off the LED
```

コードの説明

1. PWM LED の制御には「gpiozero」、アナログからデジタルへの変換には「ADC0834」、遅延の実装には「time」が使用されています。

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

from gpiozero import PWMLED
import ADC0834
import time
```

2. GPIO ピン 22 に接続された PWMLED オブジェクトを初期化し、ADC0834 コンバータをセットアップします。

```
# GPIO ピン 22 に PWM LED を初期化します
led = PWMLED(22)

# ADC0834 モジュールをセットアップします
ADC0834.setup()
```

3. 1 つの範囲の値を別の範囲に変換するための「MAP」という名前の関数を定義し、ADC の値を適切な LED の明るさにマッピングするのに役立ちます。

```
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
```

4. ループ内で ADC 値を連続して読み取り、ADC の読み取り値（0-255）を LED の明るさ（0-100）にマッピングします。このマップされた値に基づいて LED の明るさを調整します。視認性と安定性を向上させるために 0.2 秒の遅延を実装します。

```
try:
    while True:
        # ADC0834 モジュールから現在の読み取り値を取得します
        res = ADC0834.getResult()
        print('res = %d' % res)
```

(次のページに続く)

(前のページからの続き)

```
        # ADC の値を LED の明るさを設定するための適切な範囲にマップします
        R_val = MAP(res, 0, 255, 0, 100)

        # LED の明るさを設定します
        led.value = float(R_val / 100)

        # 再度読み取る前に 0.2 秒待機します
        time.sleep(0.2)

# 'Ctrl+C' が押されたときの優雅な終了
except KeyboardInterrupt:
    led.value = 0  # LED をオフにする
```

2.1.5 キーパッド

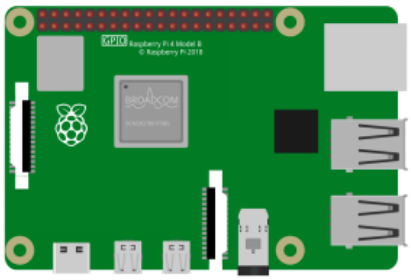
はじめに

キーパッドはボタンの長方形の配列です。このプロジェクトでは、キーパッドを文字の入力に使用します。

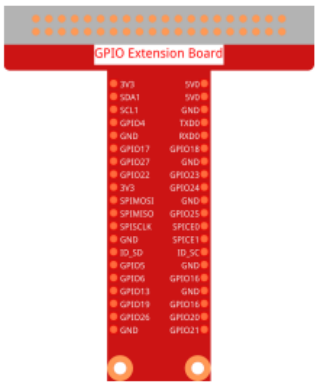
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

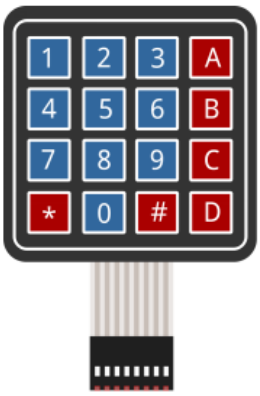
1 * Raspberry Pi



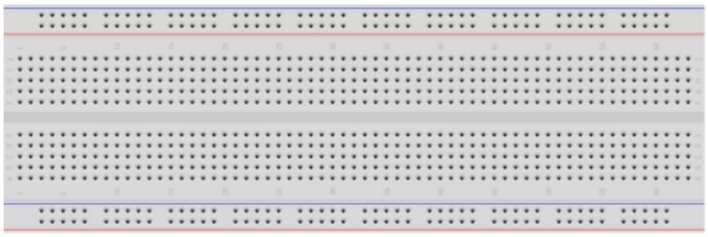
1 * T-Extension Board



1 * Keypad



1 * Breadboard



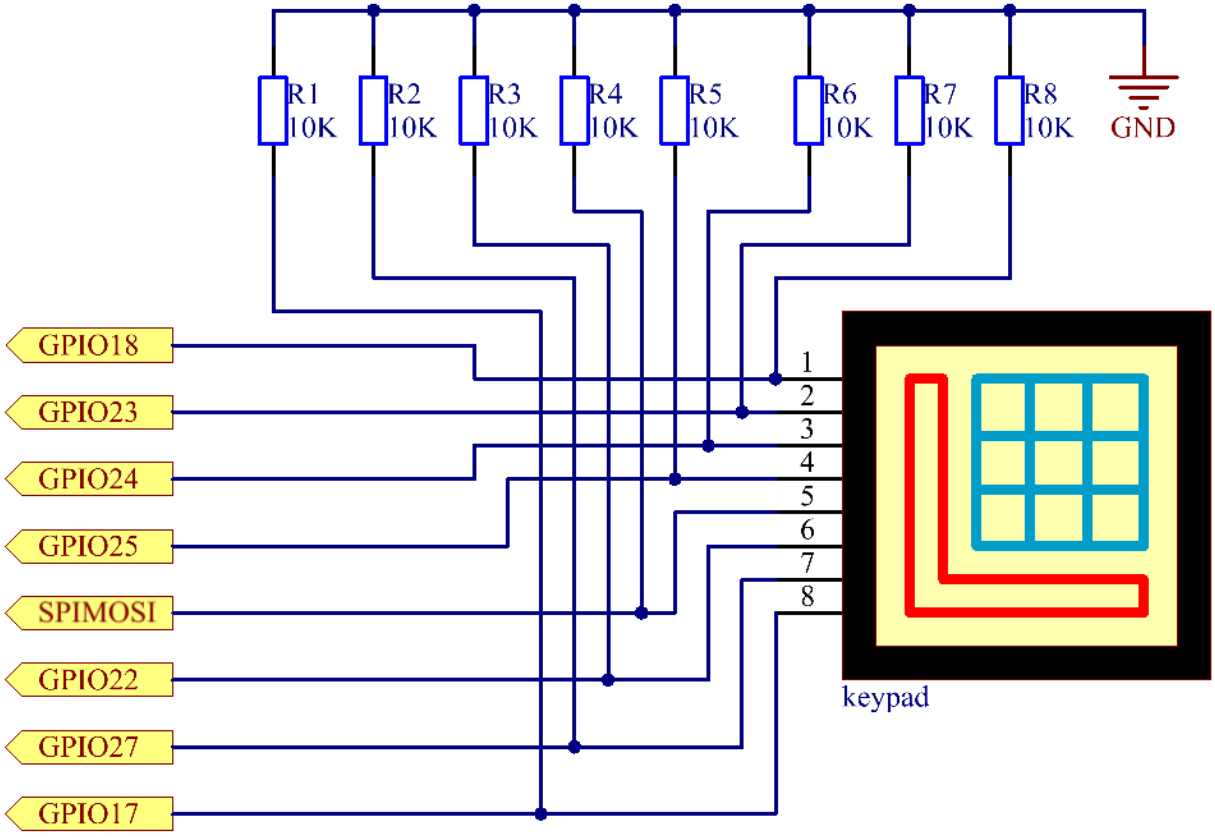
8 * Resistor 10KΩ

Several Jumper Wires

1 * 40-pin Cable

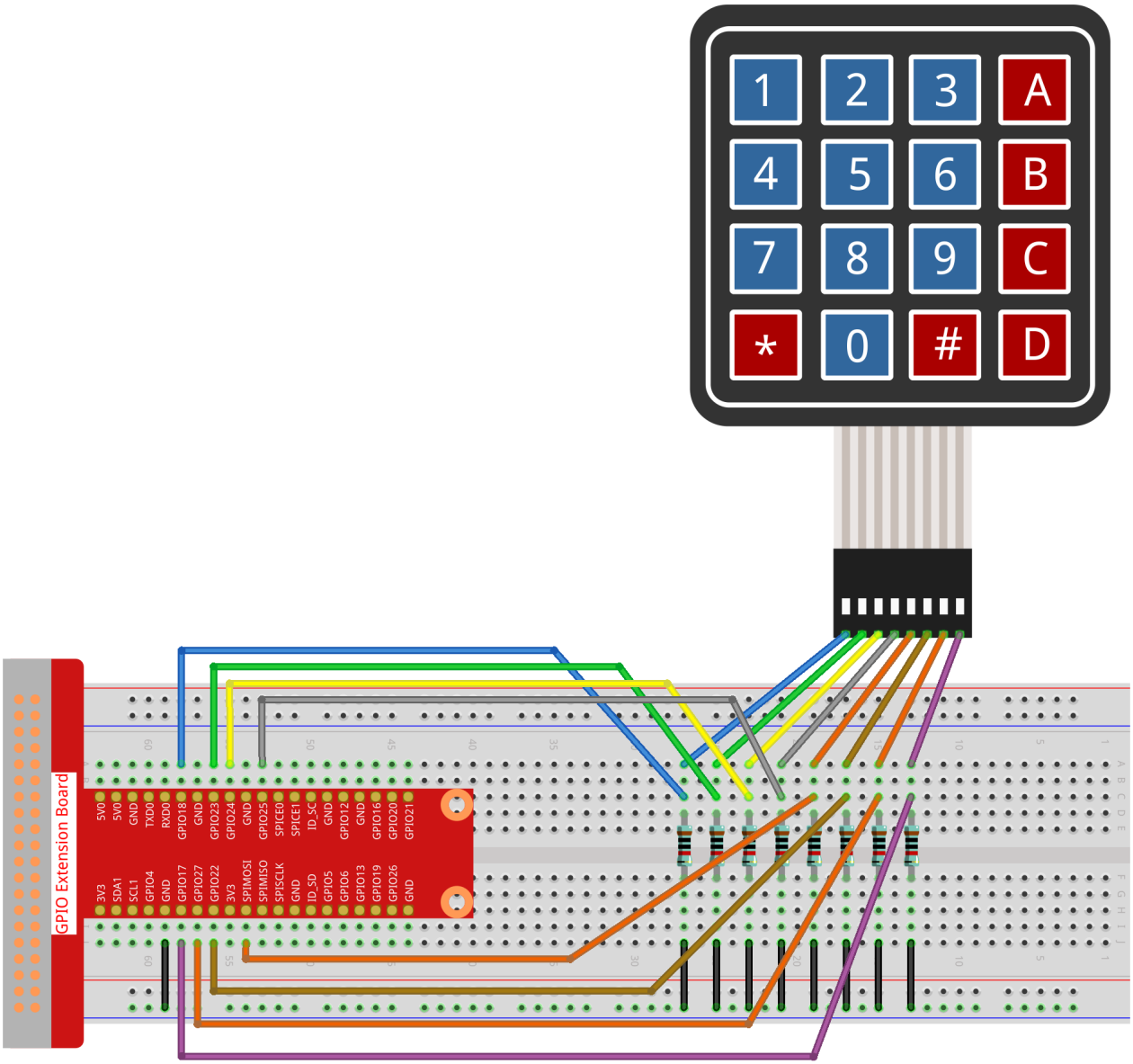
回路図

T-Board Name	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



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 2.1.5_Keypad_zero.py
```

コードを実行すると、キーパッドのボタンの押下値（ボタンの値）が画面に表示されます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に

davinci-kit-for-raspberry-pi/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import DigitalOutputDevice, Button
from time import sleep

class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        指定された行と列のピン、およびキーパッドのレイアウトでキーパッドを初期化します。
        :param rows_pins: 行の GPIO ピンのリスト。
        :param cols_pins: 列の GPIO ピンのリスト。
        :param keys: キーパッドのレイアウトのキーリスト。
        """
        # 行のピンを DigitalOutputDevice として初期化
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]
        # 列のピンを Button として初期化
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]
        self.keys = keys  # キーパッドのレイアウトを設定

    def read(self):
        """
        キーパッドで現在押されているキーを読み取ります。
        :return: 押されたキーのリスト。
        """
        pressed_keys = []
        # 押されたキーを識別するために各行と列をスキャン
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行を有効にする
            for j, col in enumerate(self.cols):
                if col.is_pressed:  # 列のボタンが押されているか確認
                    # 行と列に基づいてキーのインデックスを計算
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # 現在の行を無効にする
        return pressed_keys

try:
```

(次のページに続く)

(前のページからの続き)

```
    # 行、列、およびキーパッドのレイアウトを設定
    rows_pins = [18, 23, 24, 25]
    cols_pins = [10, 22, 27, 17]
    keys = ["1", "2", "3", "A",
            "4", "5", "6", "B",
            "7", "8", "9", "C",
            "*", "0", "#", "D"]

    # Keypad クラスのインスタンスを作成
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []

    # キーパッドを連続的に読み取り、新たに押されたキーを表示
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            print(pressed_keys)  # 押されたキーのリストを表示
            last_key_pressed = pressed_keys
        sleep(0.1)  # CPU 負荷を軽減するための短い遅延

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）を処理してスクリプトを正常に終了
    pass
```

コードの説明

1. gpiozero ライブラリから DigitalOutputDevice および Button クラス、遅延のための sleep 関数をインポートします。

```
#!/usr/bin/env python3
from gpiozero import DigitalOutputDevice, Button
from time import sleep
```

2. Keypad クラスを定義します。 __init__ メソッドは、指定された行と列のピンとキーパッドのキーレイアウトでキーパッドを初期化します。 read メソッドはキーパッドをスキャンし、押されたキーのリストを返します。

```
class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
```

(次のページに続く)

(前のページからの続き)

```
        指定された行と列のピン、およびキーパッドのレイアウトでキーパッドを初期化します。
        :param rows_pins: 行の GPIO ピンのリスト。
        :param cols_pins: 列の GPIO ピンのリスト。
        :param keys: キーパッドのレイアウトのキーリスト。
        """
        # 行のピンを DigitalOutputDevice として初期化
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]
        # 列のピンを Button として初期化
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]
        self.keys = keys  # キーパッドのレイアウトを設定

    def read(self):
        """
        キーパッドで現在押されているキーを読み取ります。
        :return: 押されたキーのリスト。
        """
        pressed_keys = []
        # 押されたキーを識別するために各行と列をスキャン
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行を有効にする
            for j, col in enumerate(self.cols):
                if col.is_pressed:  # 列のボタンが押されているか確認
                    # 行と列に基づいてキーのインデックスを計算
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # 現在の行を無効にする
        return pressed_keys
```

3. GPIO ピンの行と列を設定し、キーパッドのレイアウトを定義します。

```
try:
    # 行、列、およびキーパッドのレイアウトを設定
    rows_pins = [18, 23, 24, 25]
    cols_pins = [10, 22, 27, 17]
    keys = ["1", "2", "3", "A",
            "4", "5", "6", "B",
            "7", "8", "9", "C",
            "*", "0", "#", "D"]
```

4. 指定された設定で Keypad クラスのインスタンスを作成します。

```
try:
    ...

    # Keypad クラスのインスタンスを作成
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []
```

5. キーパッドを連続的に読み取り、キーの状態に変更があればそれを表示し、CPU 負荷を軽減するために短い遅延を導入します。キーボード割り込み（Ctrl+C など）が発生した場合、スクリプトをきれいに終了するために処理します。

```
try:
    ...

    # キーパッドを連続的に読み取り、新たに押されたキーを表示
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            print(pressed_keys)  # 押されたキーのリストを表示
            last_key_pressed = pressed_keys
        sleep(0.1)  # CPU 負荷を軽減するための短い遅延

except KeyboardInterrupt:
    # キーボード割り込み（Ctrl+C など）を処理してスクリプトを正常に終了
    pass
```

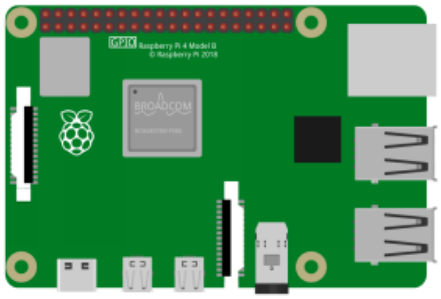
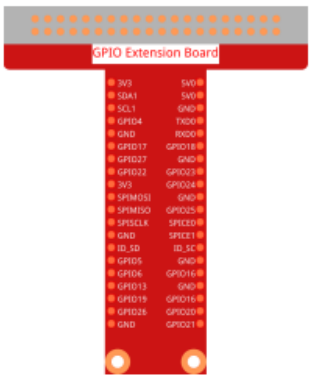
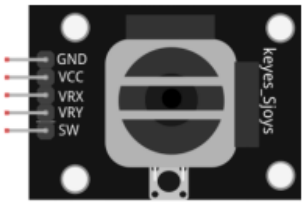
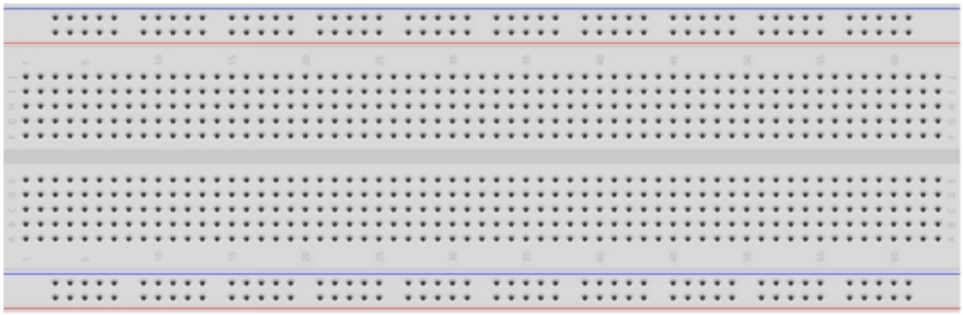
2.1.6 ジョイスティック

はじめに

このプロジェクトでは、ジョイスティックの動作を学びます。ジョイスティックを操作し、その結果を画面に表示します。

必要なコンポーネント

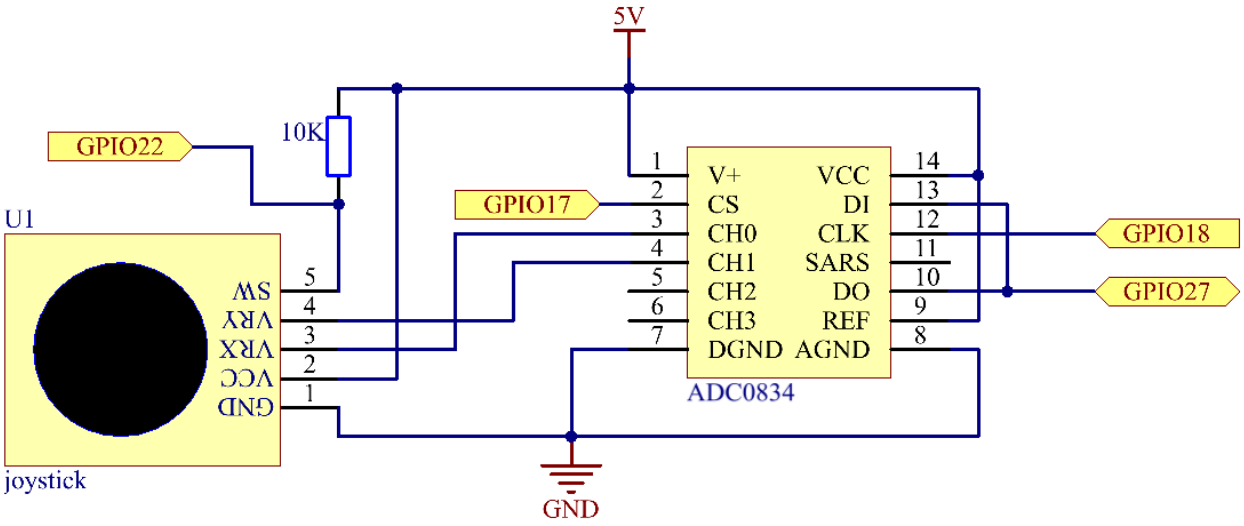
このプロジェクトでは、以下のコンポーネントが必要です。

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Joystick
  </td></tr>
<tr><td colspan="2"></td><td>1 * Resistor 10KΩ
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * ADC0834
 </td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

回路図

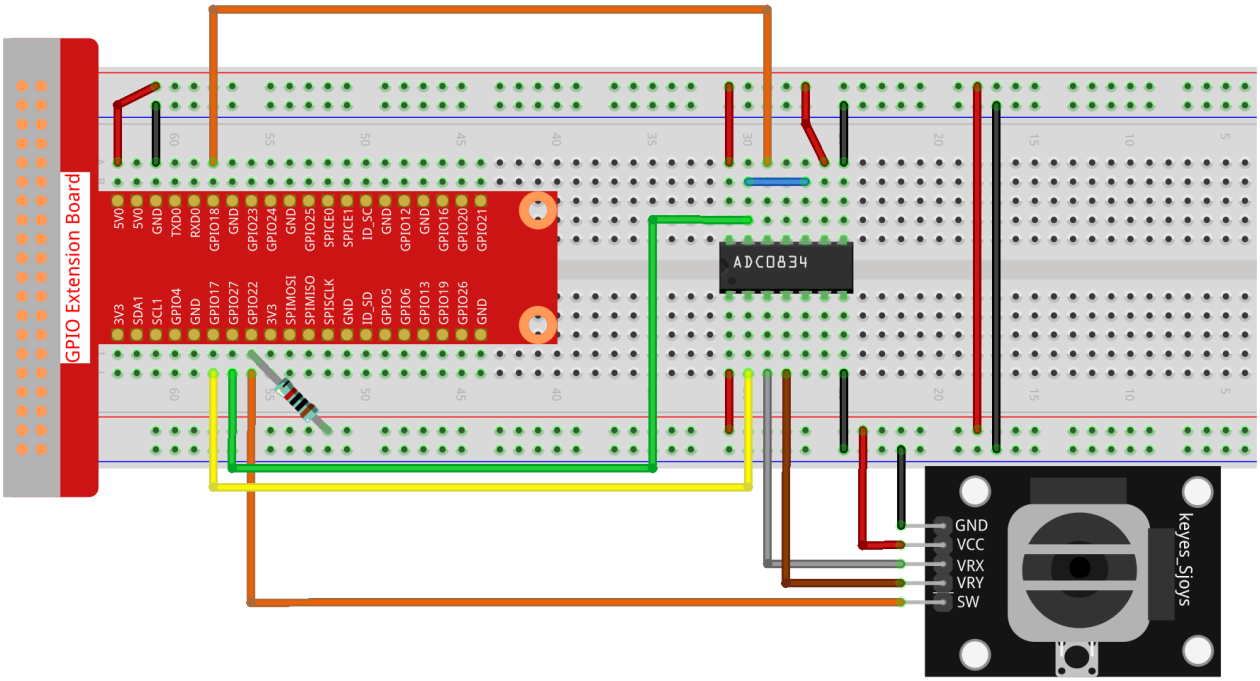
ジョイスティックのデータを読み取る際、軸ごとにいくつかの違いがあります。X 軸および Y 軸のデータはアナログで、アナログ値をデジタル値に変換するには ADC0834 を使用する必要があります。Z 軸のデータはデジタルですので、GPIO を直接読み取ることもできますし、ADC を使用しても読み取ることができます。

T-Board Name	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: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 2.1.6_Joystick_zero.py
```

コードを実行した後、ジョイスティックを動かし、対応する x、y、Btn の値が画面に表示されます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に raphael-kit/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import Button
import ADC0834
import time

# GPIO ピン 22 に接続されたボタンを初期化
BtnPin = Button(22)
```

(次のページに続く)

(前のページからの続き)

```
# ADC0834 ADC のセットアップ
ADC0834.setup()

try:
    # ADC 値とボタンの状態を読み取り、表示するメインループ
    while True:
        # ADC チャンネル 0 と 1 から X と Y の値を読み取ります
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)

        # ボタンの状態（押されているかどうか）を読み取ります
        Btn_val = BtnPin.value

        # X、Y、およびボタンの値を表示します
        print('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))

        # 次の読み取り前に 0.2 秒の遅延
        time.sleep(0.2)

# スクリプトの正常な終了を確実に処理します（たとえば、キーボード割り込みを介して）
except KeyboardInterrupt:
    pass
```

コードの説明

1. このセクションでは、 gpiozero ライブラリから Button クラスをインポートして、GPIO ピンに接続されたボタンを管理します。また、ADC0834 ADC（アナログ-デジタルコンバータ）モジュールとのインタフェースのために ADC0834 ライブラリもインポートします。

```
#!/usr/bin/env python3
from gpiozero import Button
import ADC0834
import time
```

2. GPIO ピン 22 に接続されたボタンを初期化し、ADC0834 モジュールを使用するためのセットアップを行います。

```
# GPIO ピン 22 に接続されたボタンを初期化
BtnPin = Button(22)
```

(次のページに続く)

(前のページからの続き)

```
# ADC0834 ADC のセットアップ
ADC0834.setup()
```

3. ジョイスティックの VRX および VRY 接続はそれぞれ ADC0834 の CH0 および CH1 に接続されています。これにより、CH0 および CH1 から値を読み取ることが容易になり、それらの値は x_val および y_val 変数に保存されます。さらに、ジョイスティックの SW 値が読み取られ、 Btn_val 変数に割り当てられます。取得した x_val、 y_val、および Btn_val の値は、 print() 関数を使用して表示されます。

```
try:
    # ADC 値とボタンの状態を読み取り、表示するメインループ
    while True:
        # ADC チャンネル 0 と 1 から X と Y の値を読み取ります
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)

        # ボタンの状態（押されているかどうか）を読み取ります
        Btn_val = BtnPin.value

        # X、Y、およびボタンの値を表示します
        print('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))

        # 次の読み取り前に 0.2 秒の遅延
        time.sleep(0.2)

# スクリプトの正常な終了を確実に処理します（たとえば、キーボード割り込みを介して）
except KeyboardInterrupt:
    pass
```

2.2 センサー

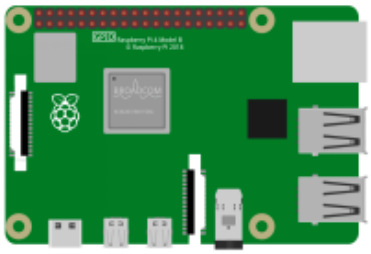
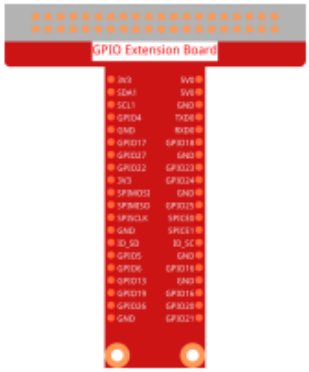
2.2.1 フォトレジスタ

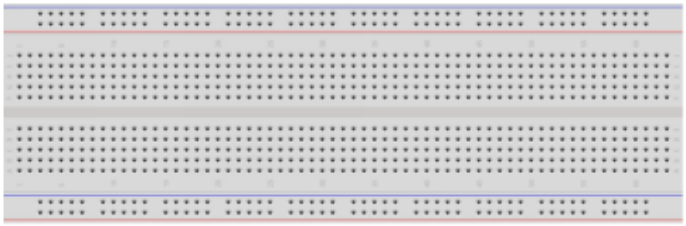
はじめに

フォトレジスタは、日常生活で環境光の強度を測定するためによく使用されるコンポーネントです。これにより、コントローラは昼と夜を認識し、ナイトランプなどの光制御機能を実現します。このプロジェクトはポテンショメーターに非常に似ており、電圧を変化させて光を検出します。

必要なコンポーネント

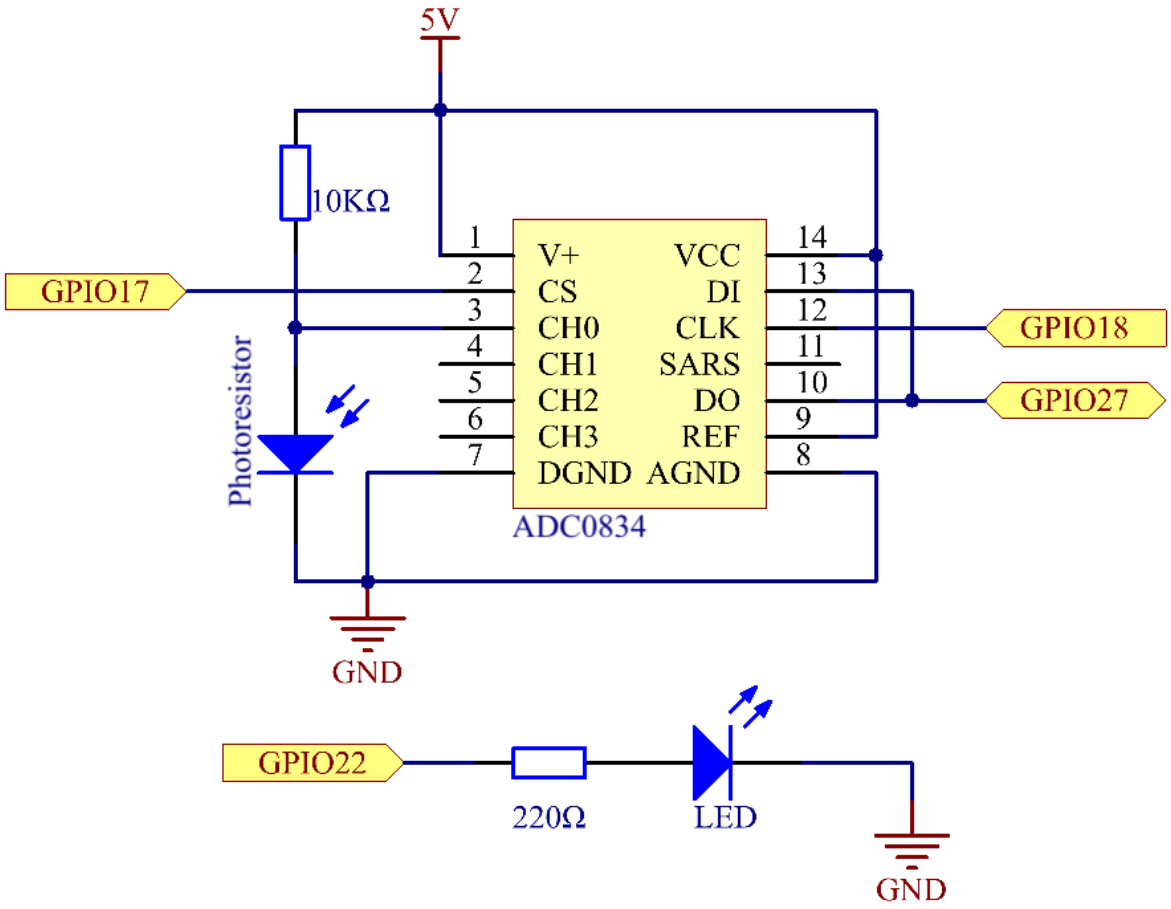
このプロジェクトでは、以下のコンポーネントが必要です。

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Photoresistor
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
 </td><td>1 * LED

1* ADC0834
 </td></tr>
<tr><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2"></td><td>1 * Resistor(10kΩ)
 </td></tr>
</table>

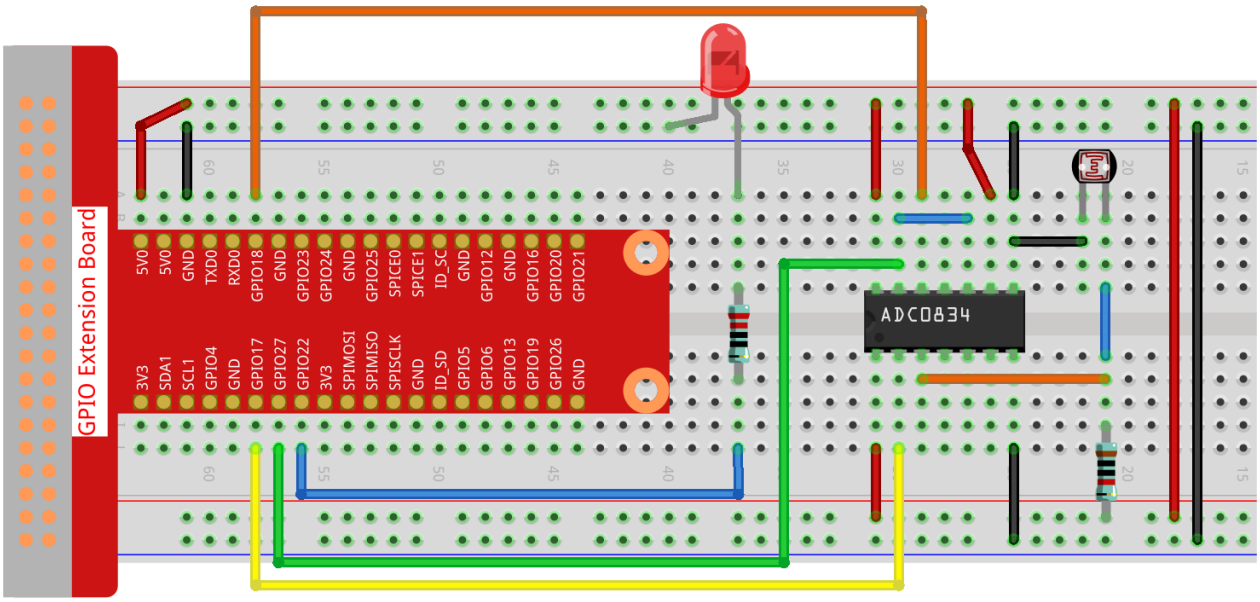
回路図

T-Board Name	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: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 2.2.1_Photoresistor_zero.py
```

コードが実行されていると、フォトレジスタで検出された光の強度に応じて LED の明るさが変化します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に raphael-kit/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import PWMLED
import ADC0834
import time
```

(次のページに続く)

(前のページからの続き)

```
# GPIO ピン 22 に接続された PWM LED を初期化
led = PWMLED(22)

# ADC0834 モジュールをセットアップ
ADC0834.setup()

# 値を別の範囲にマップするための関数を定義
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

# ADC 値を読み取り、LED の明るさを制御するメインループ
def loop():
    while True:
        # ADC からアナログ値を読み取る
        analogVal = ADC0834.getResult()
        print('value = %d' % analogVal)

        # ADC の値を PWM 値にマップして LED の明るさを設定
        led.value = float(analogVal/255)

        # 0.2 秒待つ
        time.sleep(0.2)

# メインループを実行し、KeyboardInterrupt に対応して正常なシャットダウンを処理
try:
    loop()
except KeyboardInterrupt:
    # プログラムが停止したときに LED をオフにする
    led.value = 0
```

コードの説明

1. このセクションでは、 gpiozero ライブラリから PWMLED クラスをインポートして、PWM LED の制御に必要です。また、アナログ-デジタルコンバータとのインタフェースのために ADC0834 モジュールを含み、sleep などの時間ベースの関数を実行するために time モジュールもインポートしています。

```
#!/usr/bin/env python3
from gpiozero import PWMLED
import ADC0834
import time
```

2. GPIO ピン 22 に接続された PWM LED を初期化し、ADC0834 モジュールをセットアップしてプロジェクトで使用する準備を行います。

```
# GPIO ピン 22 に接続された PWM LED を初期化
led = PWMLED(22)

# ADC0834 モジュールをセットアップ
ADC0834.setup()
```

3. 1 つの範囲から別の範囲への値をマップするための関数を定義します。この関数は、ADC の読み取りを PWM 制御に適した範囲に変換するために重要です。

```
# 値を別の範囲にマップするための関数を定義
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
```

4. このセクションには、アナログ値を ADC0834 から連続して読み取り、対応する PWM 値にマップし、LED の明るさを調整するループが含まれています。変更を可視化し、CPU への負荷を軽減するために短い遅延（*time.sleep(0.2)*）が含まれています。

```
# ADC 値を読み取り、LED の明るさを制御するメインループ
def loop():
    while True:
        # ADC からアナログ値を読み取る
        analogVal = ADC0834.getResult()
        print('value = %d' % analogVal)

        # ADC の値を PWM 値にマップして LED の明るさを設定
        led.value = float(analogVal/255)

        # 0.2 秒待つ
        time.sleep(0.2)
```

5. loop 関数を実行し、KeyboardInterrupt に対応して正常なシャットダウンを行います。プログラムが停止すると、LED がオフになるようにします。

```
# メインループを実行し、KeyboardInterrupt に対応して正常なシャットダウンを処理
try:
    loop()
except KeyboardInterrupt:
    # プログラムが停止したときに LED をオフにする
```

(次のページに続く)

(前のページからの続き)

```
    led.value = 0
```

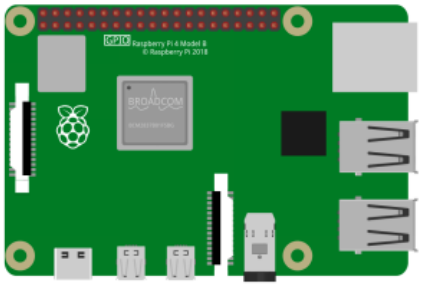
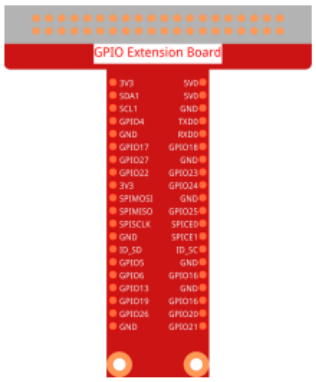
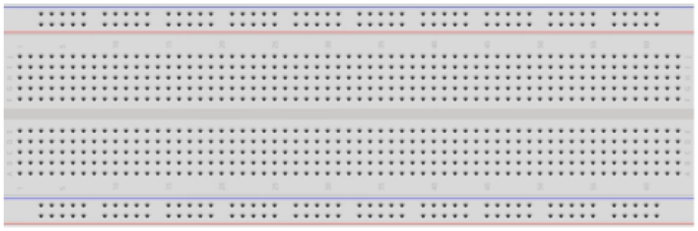
2.2.2 サーミスタ

はじめに

フォトレジスタが光を感知するように、サーミスタは温度に感応する電子デバイスで、温度制御などの機能を実現するために使用できます。たとえば、ヒートアラームを作成することができます。

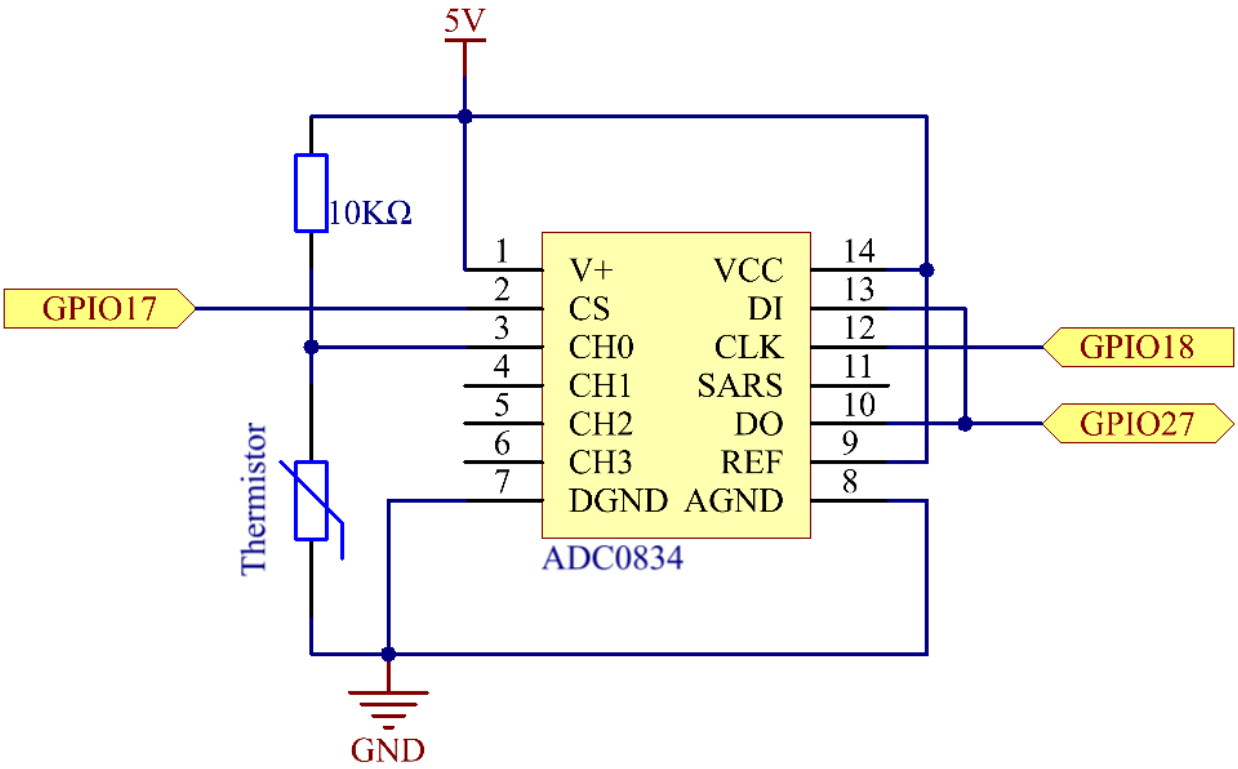
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Thermistor
		1 * ADC0834
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor 10KΩ

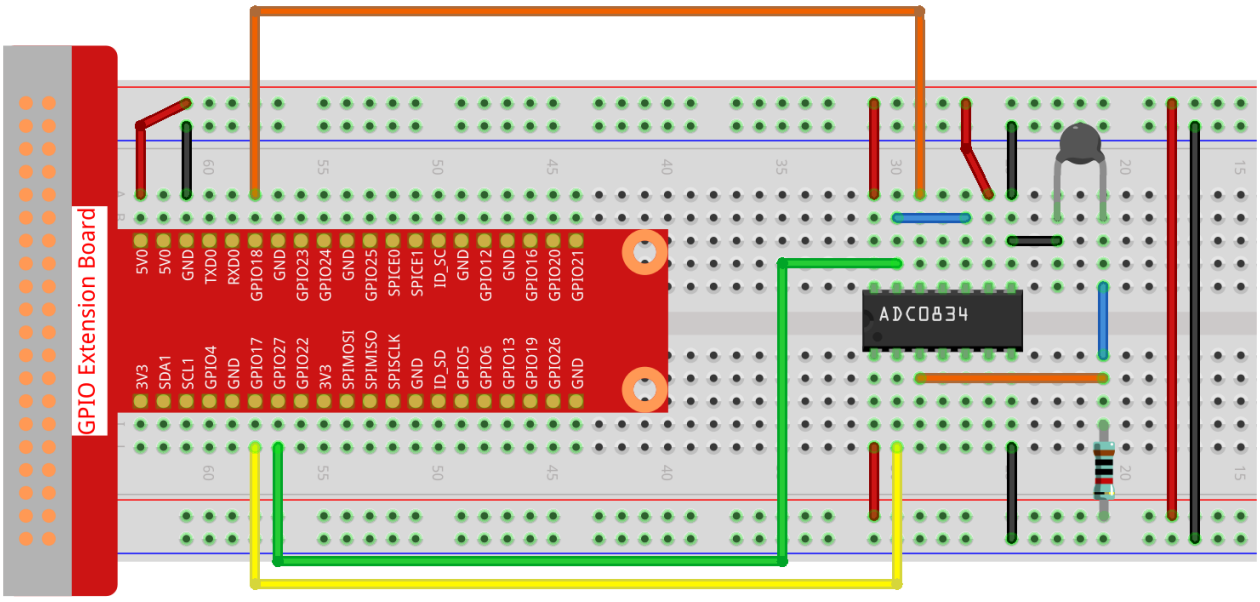
回路図

T-Board Name	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: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 2.2.2_Thermistor_zero.py
```

コードを実行すると、サーミスタが周囲の温度を検出し、プログラムの計算が終了すると温度が画面に表示されます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に raphael-kit/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

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

import ADC0834
import time
import math

# ADC0834 モジュールを初期化
ADC0834.setup()

# try-except ブロック内でプロセスを実行
try:
    while True:
        # センサーからアナログ値を読み取る
        analogVal = ADC0834.getResult()

        # アナログ値を電圧に変換
        Vr = 5 * float(analogVal) / 255

        # サーミスタの抵抗を計算
        Rt = 10000 * Vr / (5 - Vr)

        # ケルビンで温度を計算
        temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))
```

(次のページに続く)

(前のページからの続き)

```
        # ケルビンを摂氏に変換
        Cel = temp - 273.15

        # 摂氏を華氏に変換
        Fah = Cel * 1.8 + 32

        # 摂氏と華氏の両方で温度を表示
        print('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))

        # 次の読み取り前に 0.2 秒待つ
        time.sleep(0.2)

# graceful termination のために KeyboardInterrupt を処理
except KeyboardInterrupt:
    # ADC0834 のリソースをクリーンアップ
    ADC0834.destroy()
```

コードの説明

1. このセクションでは、アナログ-デジタル変換用の ADC0834 ライブラリ、遅延を実装するための time ライブラリ、数学的操作を行うための math ライブラリをインポートしています。

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

import ADC0834
import time
import math
```

2. アナログ値の読み取りを可能にするために ADC0834 モジュールを初期化します。

```
# ADC0834 モジュールを初期化
ADC0834.setup()
```

3. 無限ループを実装してデータの連続読み取りを行います。ループはサーミスタからアナログ値を読み取り、この値を電圧に変換し、サーミスタの抵抗を計算し、その抵抗をケルビン、摂氏、華氏の温度測定に変換します。温度の読み取りを摂氏と華氏の両方で出力し、各読み取りの間に 0.2 秒の待機を挿入します。

```
# try-except ブロック内でプロセスを実行
```

(次のページに続く)

(前のページからの続き)

```
try:
    while True:
        # センサーからアナログ値を読み取る
        analogVal = ADC0834.getResult()

        # アナログ値を電圧に変換
        Vr = 5 * float(analogVal) / 255

        # サーミスタの抵抗を計算
        Rt = 10000 * Vr / (5 - Vr)

        # ケルビンで温度を計算
        temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))

        # ケルビンを摂氏に変換
        Cel = temp - 273.15

        # 摂氏を華氏に変換
        Fah = Cel * 1.8 + 32

        # 摂氏と華氏の両方で温度を表示
        print('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))

        # 次の読み取り前に 0.2 秒待つ
        time.sleep(0.2)
```

4. プログラムを正常に終了するための KeyboardInterrupt 例外をキャッチし、終了時に ADC0834 のリソースをクリーンアップする命令を含めています。

```
# graceful termination のために KeyboardInterrupt を処理
except KeyboardInterrupt:
    # ADC0834 のリソースをクリーンアップ
    ADC0834.destroy()
```

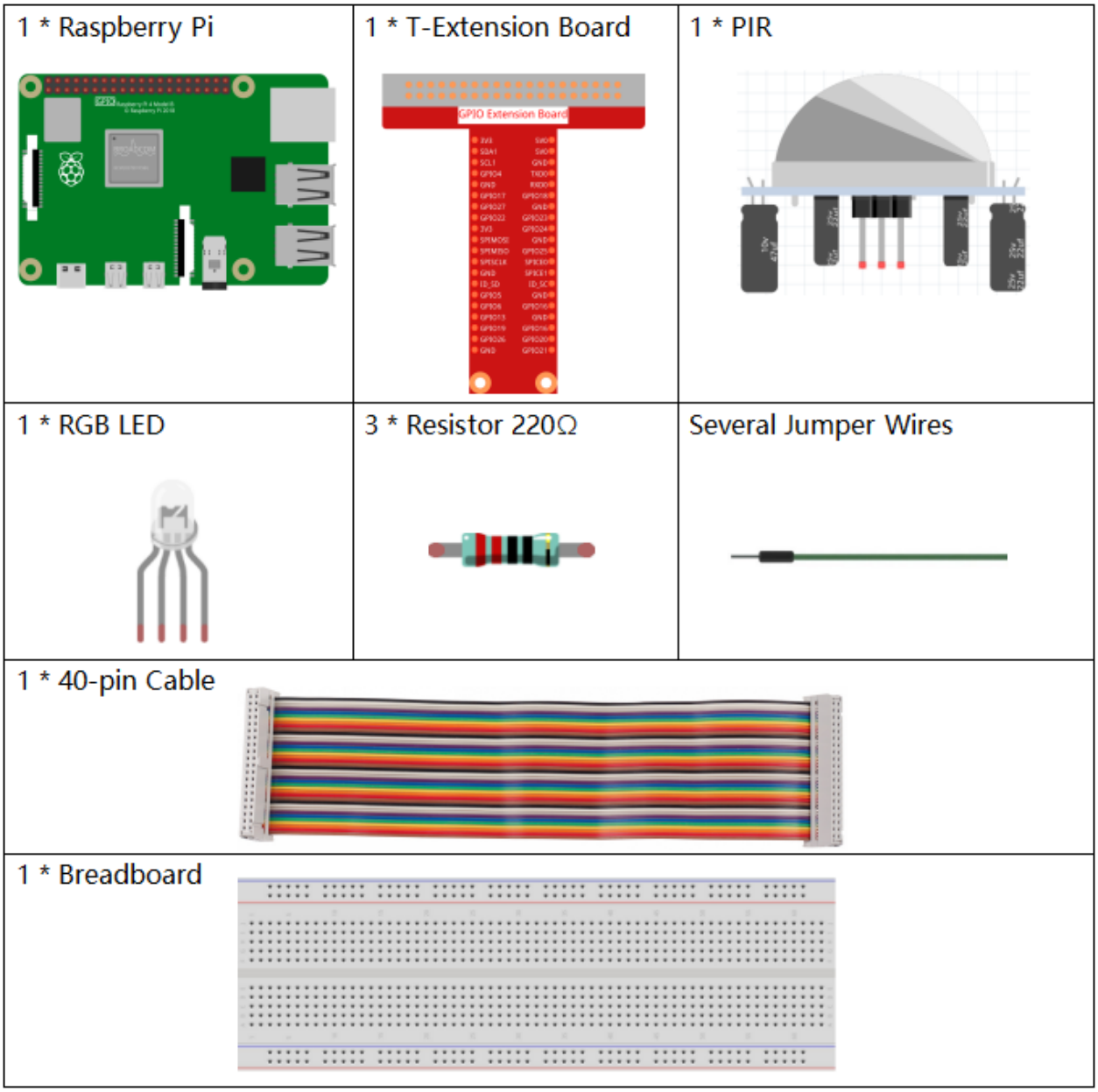
2.2.3 PIR（人体赤外線センサー）

はじめに

このプロジェクトでは、人体赤外線センサーを使用してデバイスを作成します。誰かが LED に近づくと、LED が自動で点灯します。そうでない場合、光は消えます。この赤外線モーションセンサーは、人間や動物から放射される赤外線を検出できるセンサーの一種です。

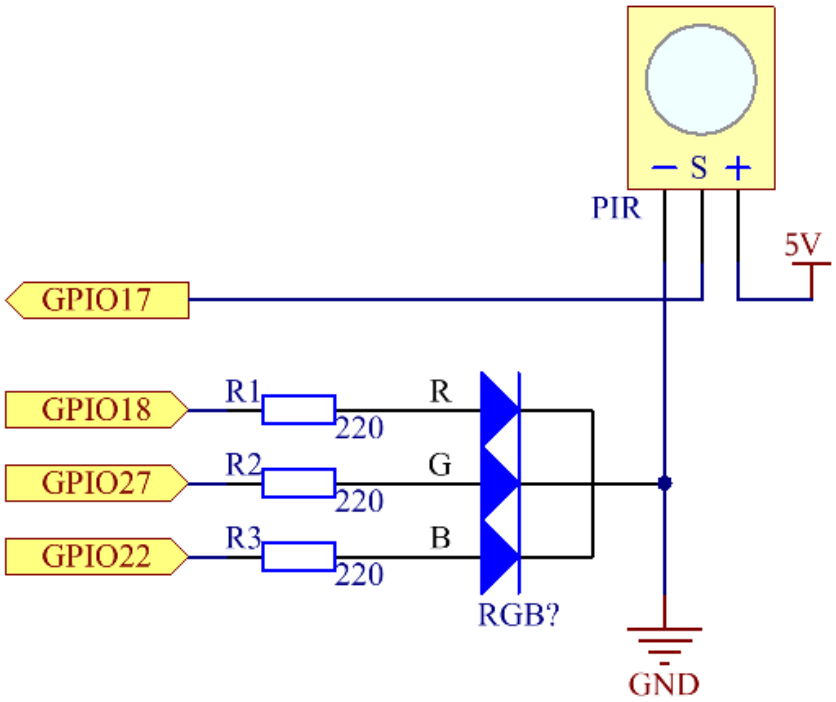
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。



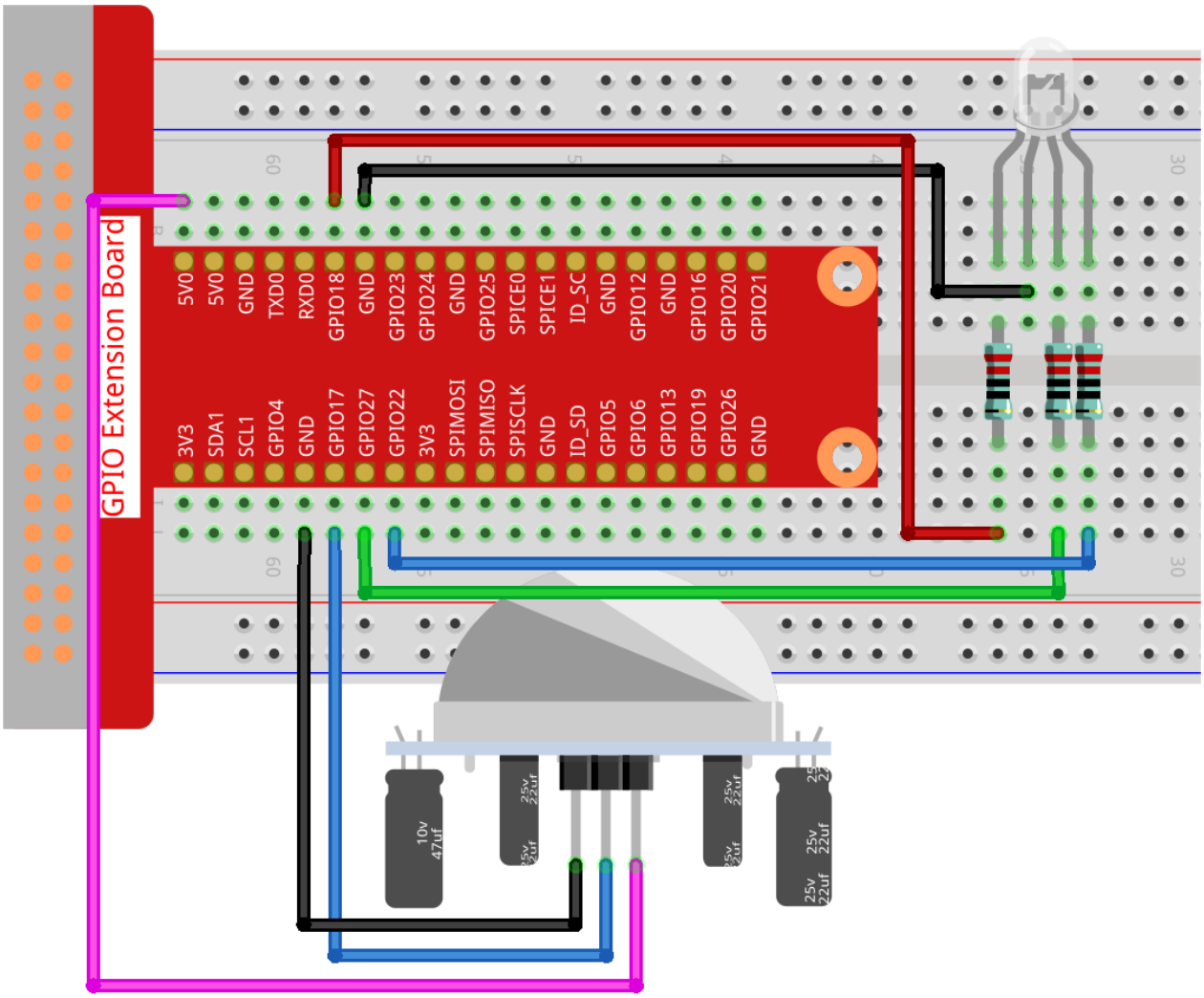
回路図

T-Board Name	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: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 2.2.4_PIR_zero.py
```

コードを実行すると、PIR が周囲を検出し、誰かが通り過ぎるのを感知すると RGB LED が黄色に光ります。

PIR モジュールには 2 つのポテンショメーターがあります：感度を調整するためのものと検出距離を調整するためのものです。 PIR モジュールをより良く動作させるには、両方を反時計回りに回して最後まで回す必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に davinci-kit-for-raspberry-pi/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import RGBLED, MotionSensor
from time import sleep

# GPIO Zero ライブラリを使用して RGB LED と PIR モーションセンサーを初期化
led = RGBLED(red=18, green=27, blue=22)  # RGB LED は GPIO ピン 18（赤）、27（緑）、22（青）に接続されています
pir = MotionSensor(17)  # PIR センサーは GPIO ピン 17 に接続されています

try:
    # モーションを監視し、LED の色を更新し続けます
    while True:
        if pir.motion_detected:  # PIR センサーによるモーションの検出を確認
            led.color = (1, 1, 0)  # LED の色を黄色に設定します（赤 + 緑）
        else:
            led.color = (0, 0, 1)  # LED の色を青に設定します（青のみ）
        sleep(0.1)  # CPU 負荷を軽減するための短い遅延
```

(次のページに続く)

(前のページからの続き)

```
except KeyboardInterrupt:
    # KeyboardInterrupt（Ctrl+C）を処理してループを gracefully に終了します
    pass
```

コードの説明

1. GPIO Zero ライブラリから RGB LED を制御するための RGBLED クラスと、モーション検出用の MotionSensor クラスをインポートします。また、遅延を作成するために sleep 関数をインポートします。

```
#!/usr/bin/env python3
from gpiozero import RGBLED, MotionSensor
from time import sleep
```

2. 赤、緑、青のコンポーネントを持つ RGB LED を初期化し、それぞれのコンポーネントを GPIO ピン 18、27、22 に接続します。また、GPIO ピン 17 に PIR センサーを初期化します。

```
# GPIO Zero ライブラリを使用して RGB LED と PIR モーションセンサーを初期化
led = RGBLED(red=18, green=27, blue=22)  # RGB LED は GPIO ピン 18（赤）、27（緑）、22（青）に接続されています
pir = MotionSensor(17)  # PIR センサーは GPIO ピン 17 に接続されています
```

3. プログラムは無限ループに入り、PIR センサーを使用してモーションを連続的に確認します。モーションが検出された場合、LED は黄色に設定されます（赤と緑の組み合わせ）。モーションが検出されない場合、LED は青に変わります。短い 0.1 秒の遅延が CPU 負荷を軽減します。

```
try:
    # モーションを監視し、LED の色を更新し続けます
    while True:
        if pir.motion_detected:  # PIR センサーによるモーションの検出を確認
            led.color = (1, 1, 0)  # LED の色を黄色に設定します（赤 + 緑）
        else:
            led.color = (0, 0, 1)  # LED の色を青に設定します（青のみ）
        sleep(0.1)  # CPU 負荷を軽減するための短い遅延

except KeyboardInterrupt:
    # KeyboardInterrupt（Ctrl+C）を処理してループを gracefully に終了します
    pass
```

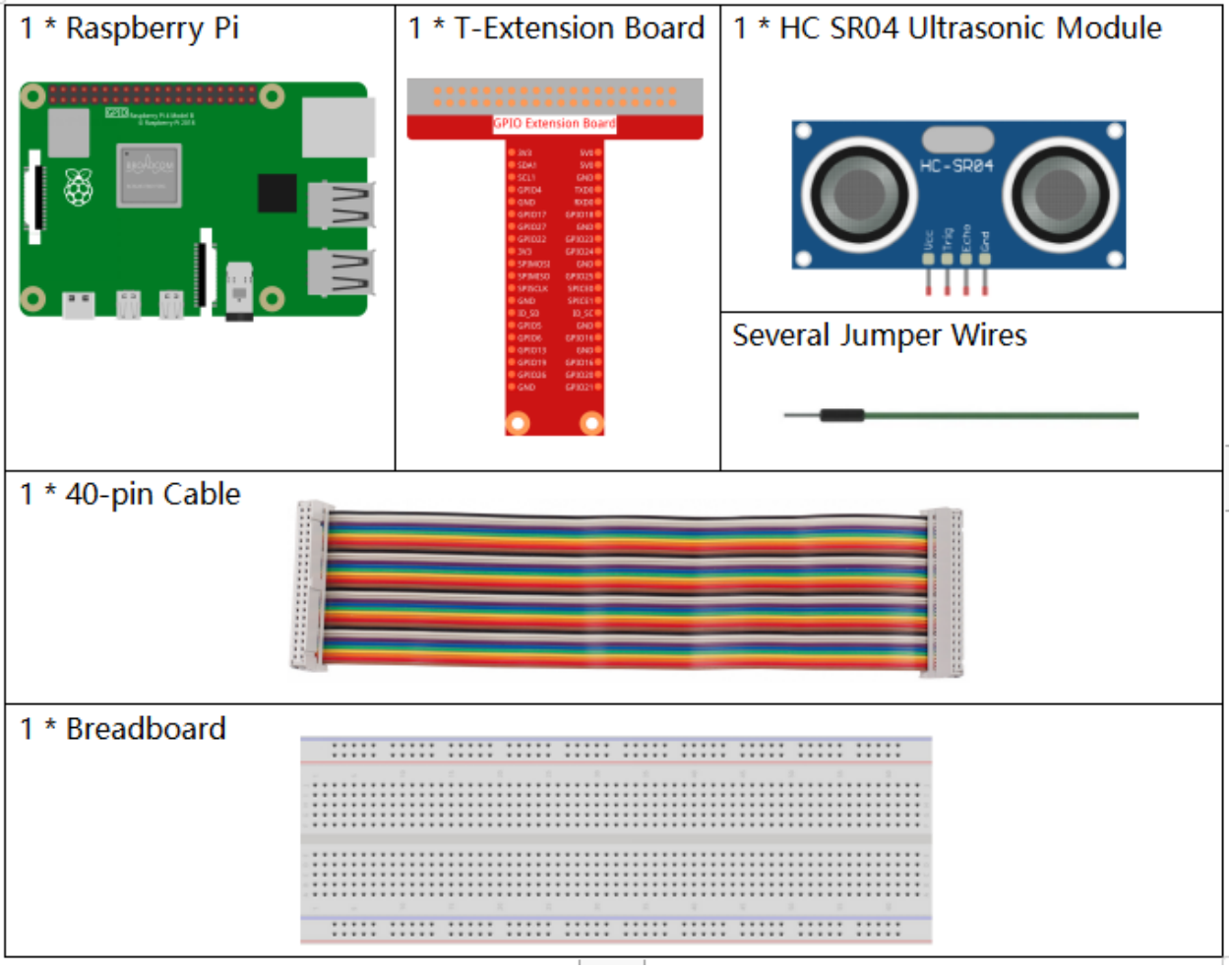
2.2.4 超音波センサーモジュール

はじめに

超音波センサーは、物体を正確に検出し、距離を測定するために超音波を使用します。それは超音波を発信し、それらを電子信号に変換します。

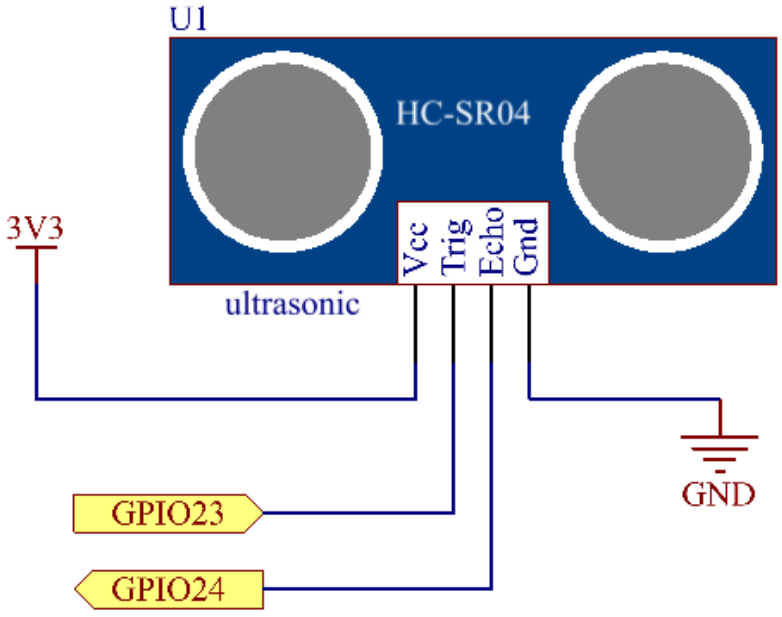
必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。



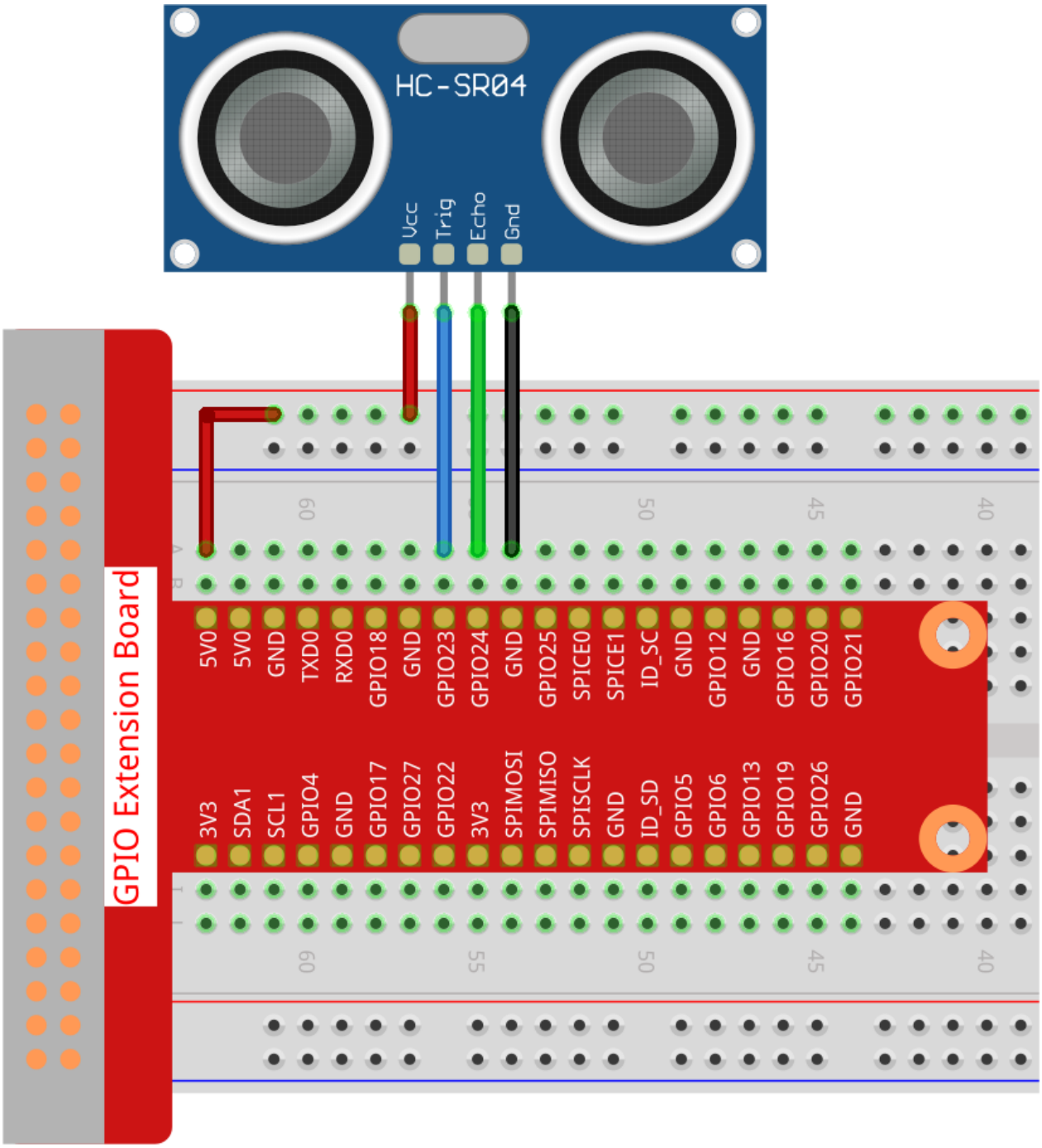
回路図

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 2.2.5_Ultrasonic_zero.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距離を検出し、その距離値が画面に表示されます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に davinci-kit-for-raspberry-pi/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import DistanceSensor
from time import sleep

# GPIO Zero ライブラリを使用して DistanceSensor を初期化
# トリガーピンは GPIO 23 に接続、エコーピンは GPIO 24 に接続
sensor = DistanceSensor(echo=24, trigger=23)

try:
    # 距離を連続的に測定および報告するためのメインループ
    while True:
        dis = sensor.distance * 100  # 距離を測定し、メートルからセンチメートルに変換
        print('Distance: {:.2f} cm'.format(dis)) # 距離を小数点以下 2 桁の精度で表示
        sleep(0.3)  # 次の測定前に 0.3 秒待機

except KeyboardInterrupt:
    # KeyboardInterrupt (Ctrl+C) を処理してループを gracefully に終了します
    pass
```

コードの説明

1. 遅延用に time モジュールから sleep 関数を含む、距離測定用の gpiozero ライブラリから DistanceSensor クラスをインポートします。

   ```
   #!/usr/bin/env python3
   from gpiozero import DistanceSensor
   from time import sleep
   ```

2. エコーピンを GPIO 24 に、トリガーピンを GPIO 23 に接続した超音波距離センサーを初期化します。

```
# GPIO Zero ライブラリを使用して DistanceSensor を初期化
# トリガーピンは GPIO 23 に接続、エコーピンは GPIO 24 に接続
sensor = DistanceSensor(echo=24, trigger=23)
```

3. メインループは距離を連続して測定し、メートルからセンチメートルに変換し、小数点以下 2 桁の精度で表示します。その後、0.3 秒待ってから距離を再測定します。 KeyboardInterrupt （Ctrl+C のような）をキャッチしてスクリプトから gracefully に終了できるようにします。

```
try:
    # 距離を連続的に測定および報告するためのメインループ
    while True:
        dis = sensor.distance * 100  # 距離を測定し、メートルからセンチメートルに変換
        print('Distance: {:.2f} cm'.format(dis))   # 距離を小数点以下 2 桁の精度で表示
        sleep(0.3)  # 次の測定前に 0.3 秒待機

except KeyboardInterrupt:
    # KeyboardInterrupt (Ctrl+C) を処理してループを gracefully に終了します
    pass
```

2.2.5 MPU6050 モジュール

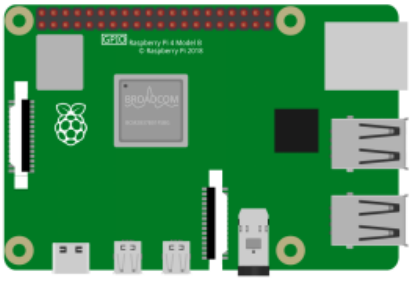
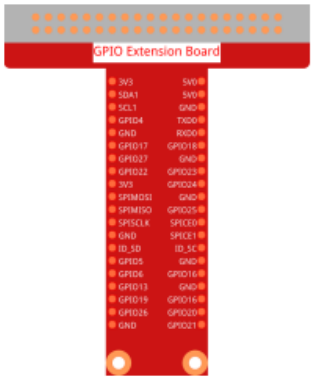
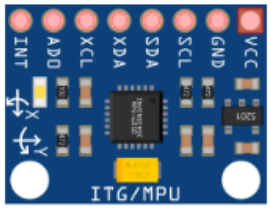
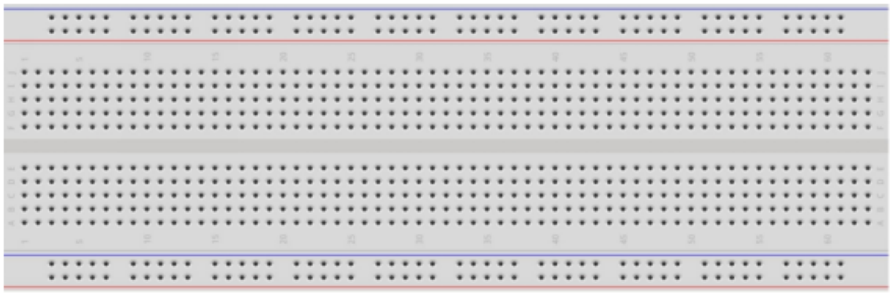
はじめに

MPU-6050 は、スマートフォン、タブレット、およびウェアラブルセンサー向けに設計された、世界初かつ唯一の 6 軸モーショントラッキングデバイス（3 軸ジャイロスコープと 3 軸アクセラメーター）です。低消費電力、低コスト、高性能の要件を持つこれらの機能を備えています。

この実験では、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 I2C を使用して取得し、それらを画面に表示します。

必要なコンポーネント

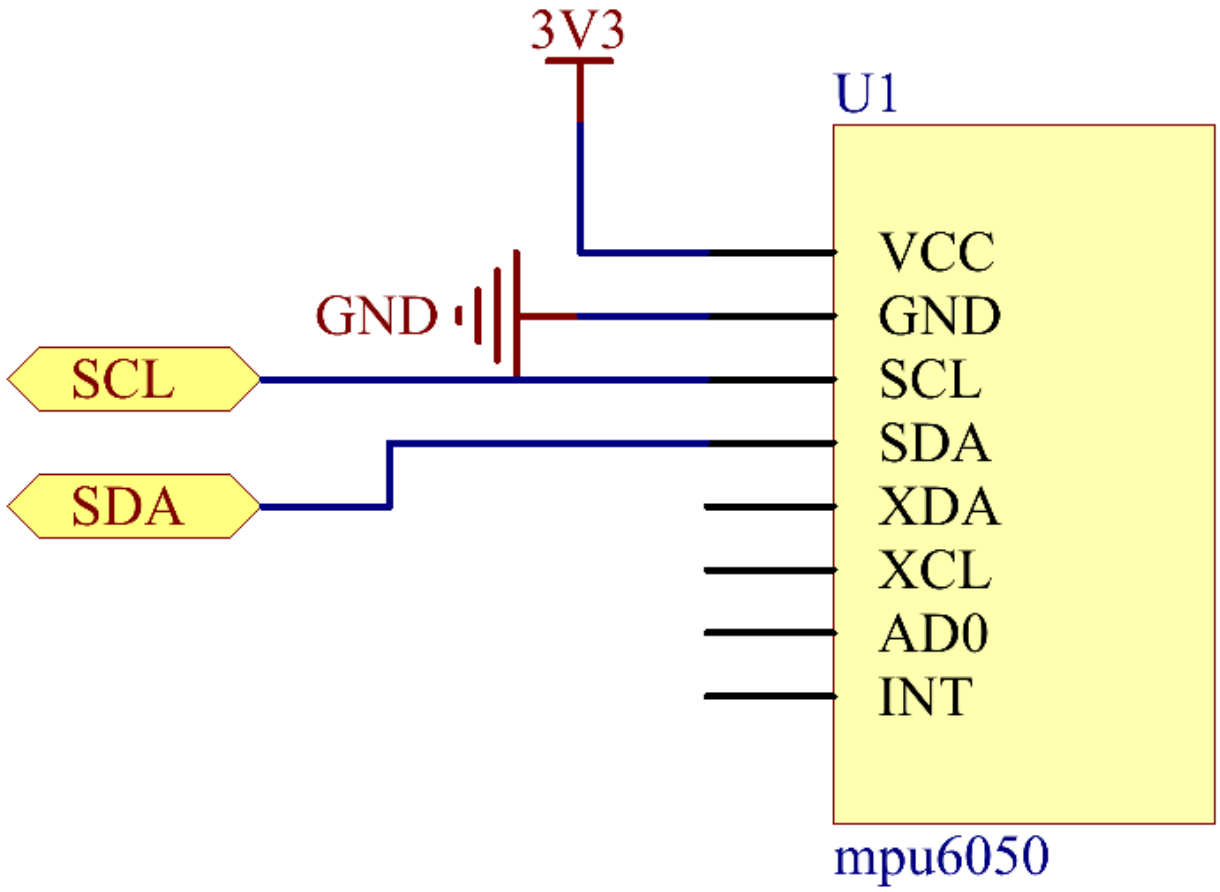
このプロジェクトでは、以下のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * MPU6050 
		Several Jumper Wires
1 * 40-pin Cable		
1 * Breadboard 		

回路図

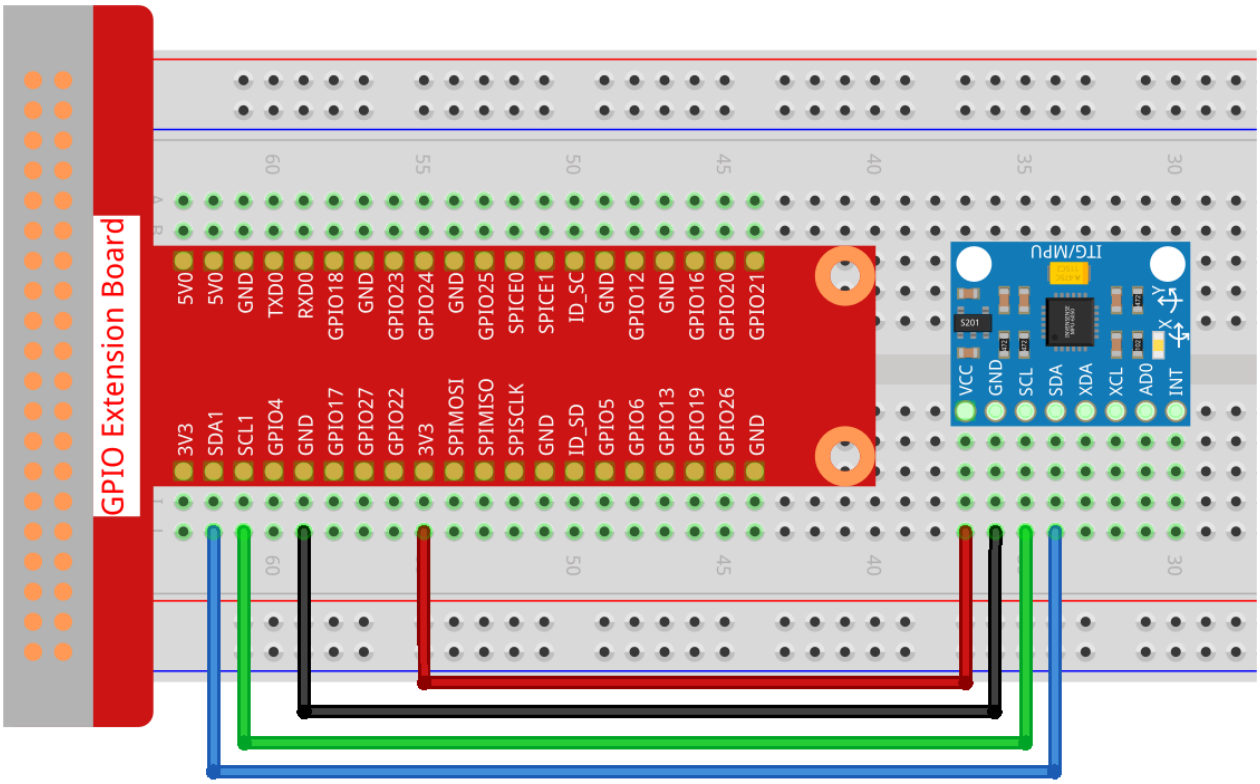
MPU6050 は I2C バスインターフェースを介してマイクロコントローラと通信します。SDA1 と SCL1 は対応するピンに接続する必要があります。

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: I2C のセットアップを行います（Appendix *I2C* **構成** を参照してください。I2C を設定済みの場合は、このステップをスキップします。）

ステップ 3: コードのフォルダに移動します。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 4: 実行可能ファイルを実行します。

```
sudo python3 2.2.6_mpu6050_zero.py
```

コードを実行すると、MPU6050 によって読み取られた x 軸および y 軸の偏角、および加速度、各軸の角速度が計算された後、画面に表示されます。

注釈:

- エラーが発生した場合、 `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` と表示される場合、I2C を有効にするために *I2C* **構成** を参照してください。

- ModuleNotFoundError: No module named 'smbus2' エラーが表示された場合、sudo pip3 install smbus2 を実行してください。
- エラー OSError: [Errno 121] Remote I/O error が表示される場合、モジュールの接続が誤っているか、モジュールが壊れていることを示します。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に davinci-kit-for-raspberry-pi/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
import smbus
import math
import time

# 電源管理レジスタ
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))
```

(次のページに続く)

(前のページからの続き)

```
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       # これは i2cdetect コマンドで読み取られるアドレス値です

# 6050 をスリープモードから起こします
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))
```

(次のページに続く)

(前のページからの続き)

```
    print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_
↪zout_scaled))

    time.sleep(1)
```

コードの説明

1. MPU6050 から送信されたセンサーデータを読み取ります。

```
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
```

2. y 軸の偏角を計算します。

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

3. x 軸の偏角を計算します。

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

4. ジャイロスコープセンサーの x 軸、y 軸、z 軸の値を読み取り、メタデータを角速度の値に変換し、それらを表示します。

```
gyro_xout = read_word_2c(0x43)
gyro_yout = read_word_2c(0x45)
gyro_zout = read_word_2c(0x47)
```

(次のページに続く)

(前のページからの続き)

```
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))
```

5. 加速度センサーの x 軸、y 軸、z 軸の値を読み取り、その値を加速度（重力単位）に変換し、それらを表示します。

```
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)
```

6. x 軸および y 軸の偏角を表示します。

```
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))
```

1.12.4 3 拡張

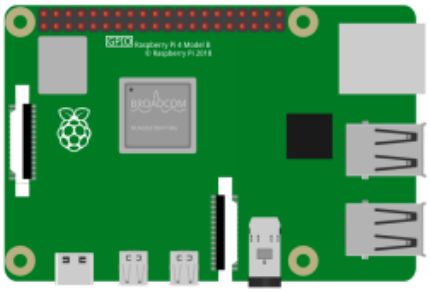
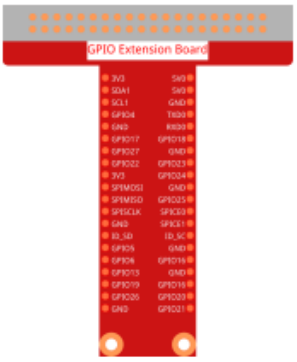
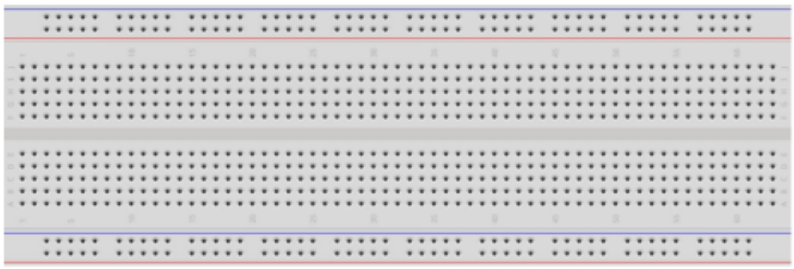
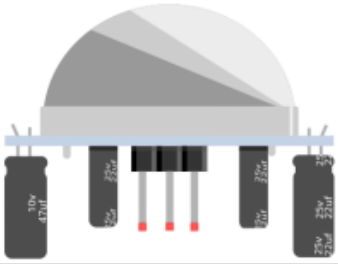
3.1.1 カウンティングデバイス

はじめに

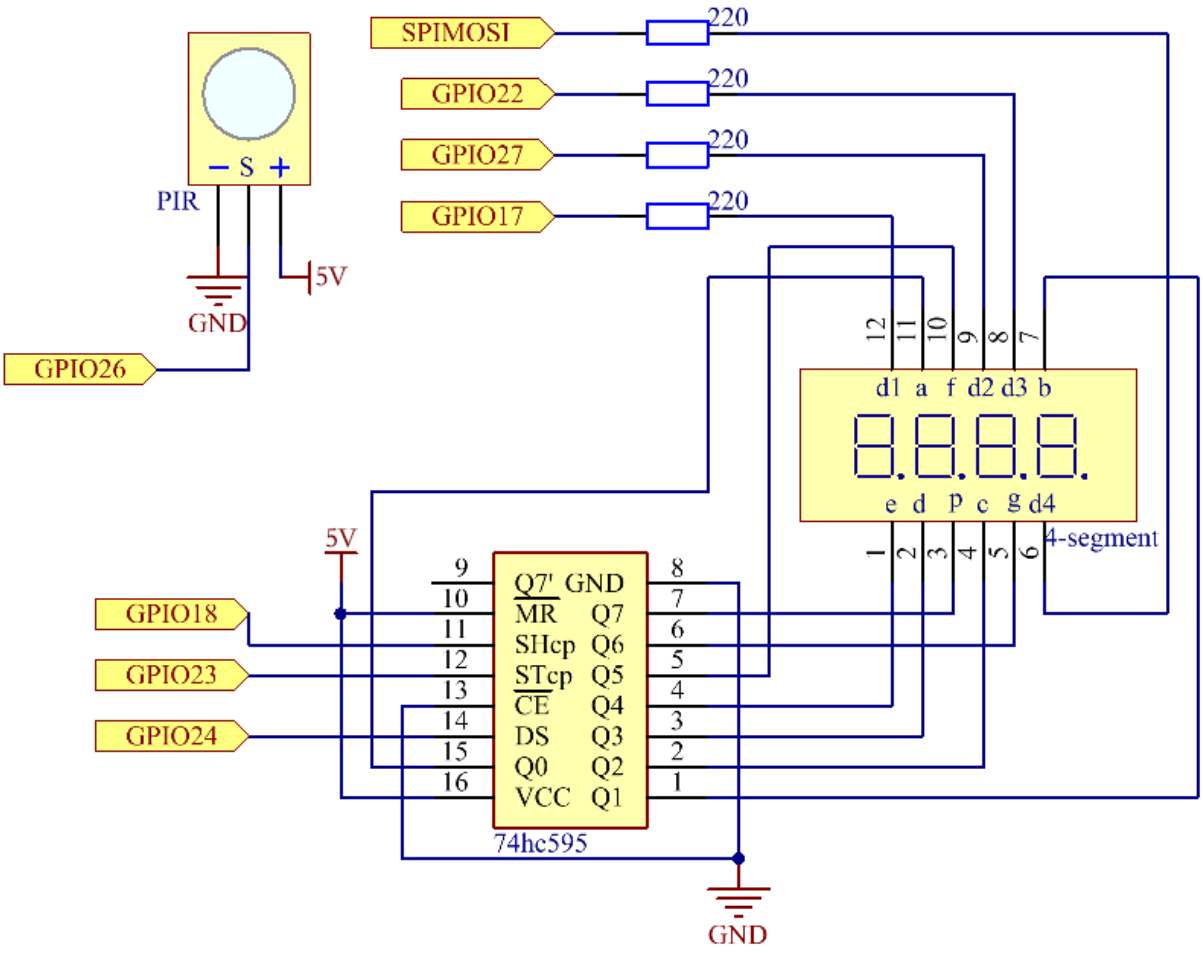
ここでは、PIR センサーと 4 桁のセグメントディスプレイからなる、数値表示カウンターシステムを作成します。PIR が通行人を検出すると、4 桁のセグメントディスプレイに表示される数値が 1 つ増えます。このカウンターを使用して、通路を通過する人数をカウントできます。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

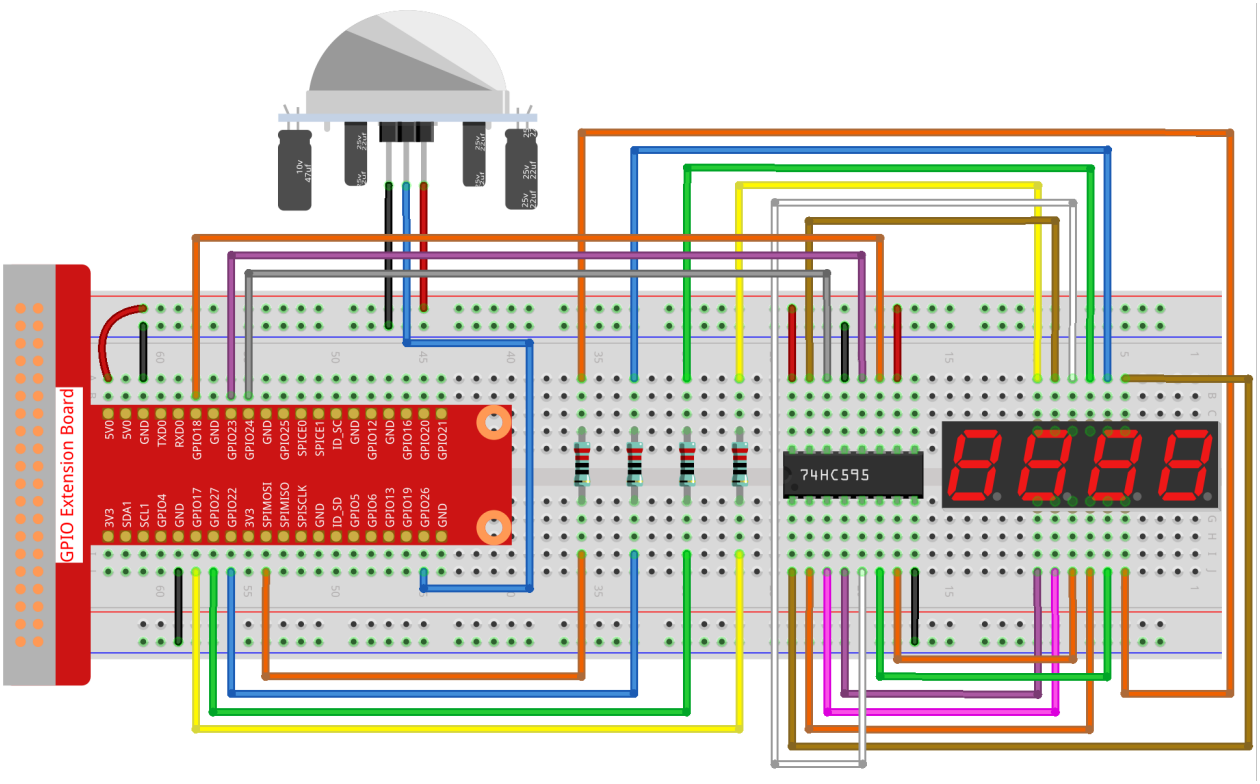
1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display
		4 * Resistor(220Ω)
1 * 40-pin Cable		1 * 74HC595
		Several Jumper Wires
1 * Breadboard 		1 * PIR Sensor Module 

回路図



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 3.1.1_CountingDevice_zero.py
```

コードを実行すると、PIR が通行人を検出すると、4 桁のセグメントディスプレイに表示される数値が 1 つ増えます。

PIR モジュールには 2 つのポテンショメータがあります。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す。PIR モジュールをより良く動作させるには、これらのポテンショメータを両方とも反時計回りにまわす必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に raphael-kit/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, MotionSensor

# GPIO 26 に接続された PIR モーションセンサーを初期化
pir = MotionSensor(26)

# シフトレジスタのピンを初期化
SDI = OutputDevice(24)    # シリアルデータ入力
RCLK = OutputDevice(23)   # レジスタクロック入力
SRCLK = OutputDevice(18)  # シフトレジスタクロック入力

# 7 セグメントディスプレイのピンを初期化
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメントディスプレイに表示する数字のコードを定義
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# 表示される数値のカウンター
```

(次のページに続く)

(前のページからの続き)

```
counter = 0

def clearDisplay():
    # すべてのセグメントをオフにしてディスプレイをクリア
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    # データを 74HC595 シフトレジスタにシフト
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    # 7 セグメントディスプレイの特定の桁をアクティブ化
    for pin in placePin:
        pin.off()
    placePin[digit].on()

def display():
    # 現在のカウンター値でディスプレイを更新
    global counter
    clearDisplay()
    pickDigit(0)
    hc595_shift(number[counter % 10])

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

    clearDisplay()
    pickDigit(2)
```

(次のページに続く)

(前のページからの続き)

```
    hc595_shift(number[counter % 1000//100])

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

def loop():
    # ディスプレイを連続的に更新し、モーションを確認するメインループ
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = 1 if pir.motion_detected else 0
        if currentState == 1 and lastState == 0:
            counter += 1
        lastState = currentState

try:
    loop()
except KeyboardInterrupt:
    # スクリプトが中断されたときにすべてのピンをオフにする
    SDI.off()
    SRCLK.off()
    RCLK.off()
    pass
```

コードの説明

1. この行では、 gpiozero ライブラリから OutputDevice と MotionSensor クラスをインポートします。 OutputDevice は LED、モーター、または出力として制御したい任意のデバイスを指します。 MotionSensor は通常、動きを検出するために使用される PIR（受動赤外線）センサーです。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, MotionSensor
```

2. GPIO ピン 26 に接続された PIR モーションセンサーを初期化します。

```
# GPIO 26 に PIR モーションセンサーを初期化
pir = MotionSensor(26)
```

3. シフトレジスタのシリアルデータ入力（SDI）、レジスタクロック入力（RCLK）、シフトレジスタクロック入力（SRCLK）に接続された GPIO ピンを初期化します。

```
# シフトレジスタピンを初期化
SDI = OutputDevice(24)    # シリアルデータ入力
RCLK = OutputDevice(23)   # レジスタクロック入力
SRCLK = OutputDevice(18)  # シフトレジスタクロック入力
```

4. 7 セグメント表示の各桁用のピンを初期化し、0-9 の数字を表示するためのバイナリコードを定義します。

```
# 7 セグメント表示用ピンを初期化
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメント表示の数字コードを定義
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

5. 次の数字を表示する前にすべてのセグメントをオフに設定して、7 セグメント表示をクリアします。

```
def clearDisplay():
    # すべてのセグメントをオフに設定して表示をクリアする
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

6. 74HC595 シフトレジスタに 1 バイトのデータをシフトし、表示セグメントを制御します。

```
def hc595_shift(data):
    # 74HC595 シフトレジスタにデータをシフトする
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

7. 7 セグメント表示のどの桁をアクティブにするかを選択します。各桁は別々の GPIO ピンによって制御されます。

```
def pickDigit(digit):
    # 7セグメント表示の特定の桁をアクティブにする
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

8. 最初に単位の桁の表示を開始し、次に十の桁の表示をアクティブにします。その後、百と千の桁の表示を順に行います。この高速なアクティベーションの連続は、連続した 4 桁の表示の錯覚を作り出します。

```
def display():
    # 現在のカウンター値で表示を更新する
    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])
```

9. メインループを定義し、表示を継続的に更新し、PIR センサーの状態をチェックします。動きが検出されると、カウンターが増加します。

```
def loop():
    # 表示を更新し、動きをチェックするメインループ
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = 1 if pir.motion_detected else 0
        if currentState == 1 and lastState == 0:
```

(次のページに続く)

(前のページからの続き)

```
            counter += 1
        lastState = currentState
```

10. メインループを実行し、キーボードコマンド（Ctrl+C）で中断できるようにします。クリーンな終了のためにすべてのピンをオフにします。

```
try:
    loop()
except KeyboardInterrupt:
    # スクリプトが中断されたときにすべてのピンをオフにする
    SDI.off()
    SRCLK.off()
    RCLK.off()
    pass
```

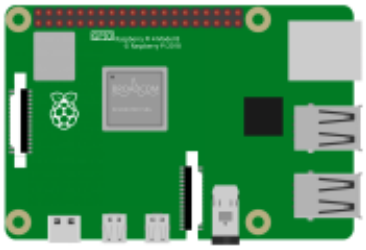
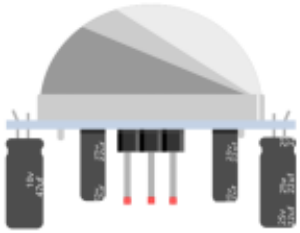
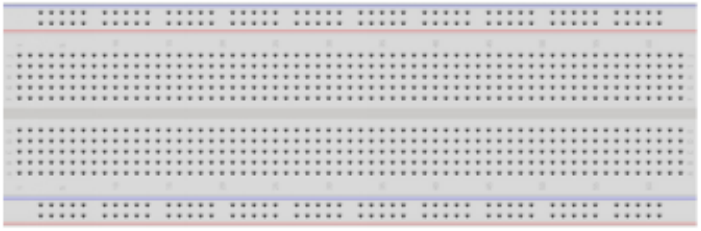
3.1.2 ようこそ

はじめに

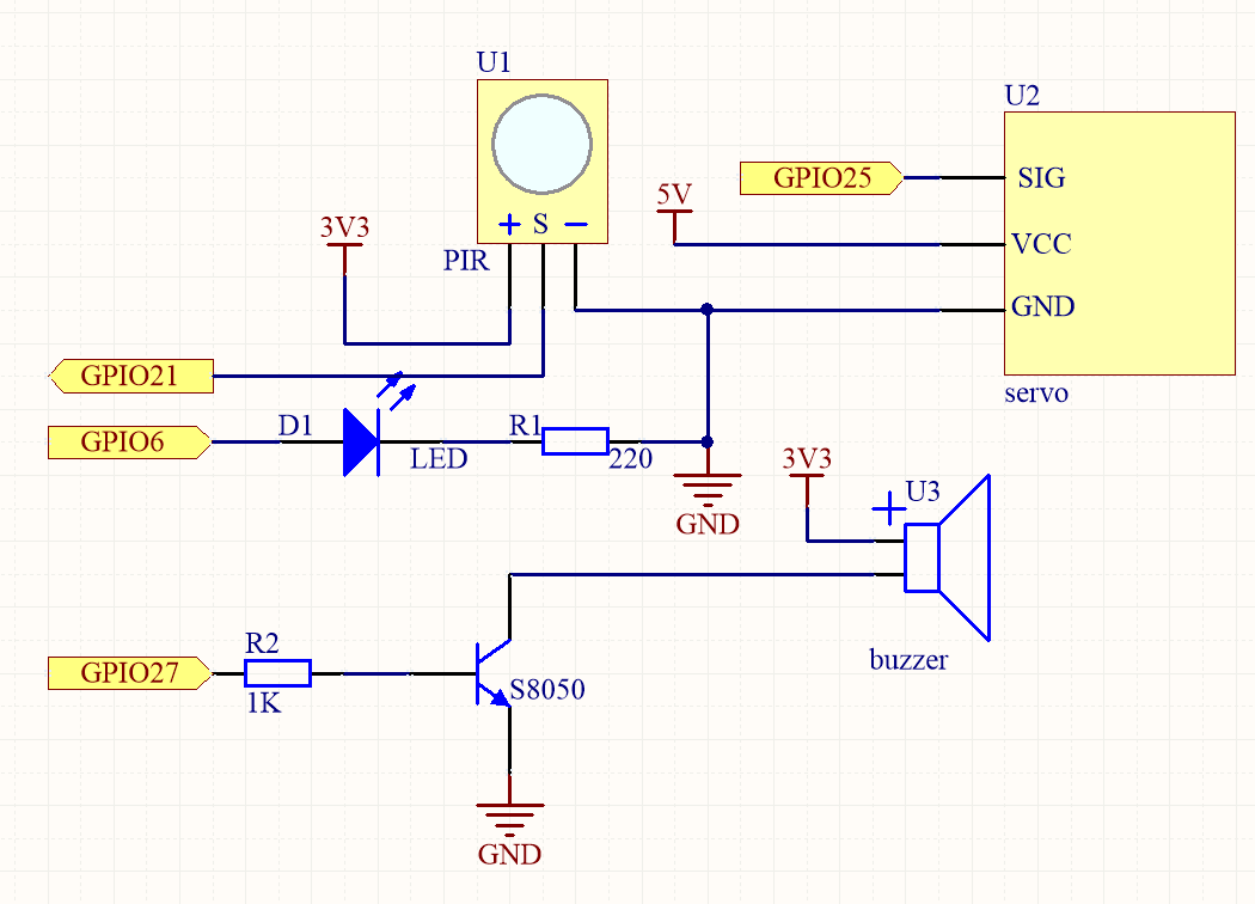
このプロジェクトでは、歩行者の動きを検出するために PIR を使用し、センサー付きコンビニエンスストアの扉の動作を模倣するためにサーボ、LED、ブザーを使用します。歩行者が PIR の感知範囲内に現れると、インジケーターライトが点灯し、扉が開き、ブザーが開店ベルを演奏します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

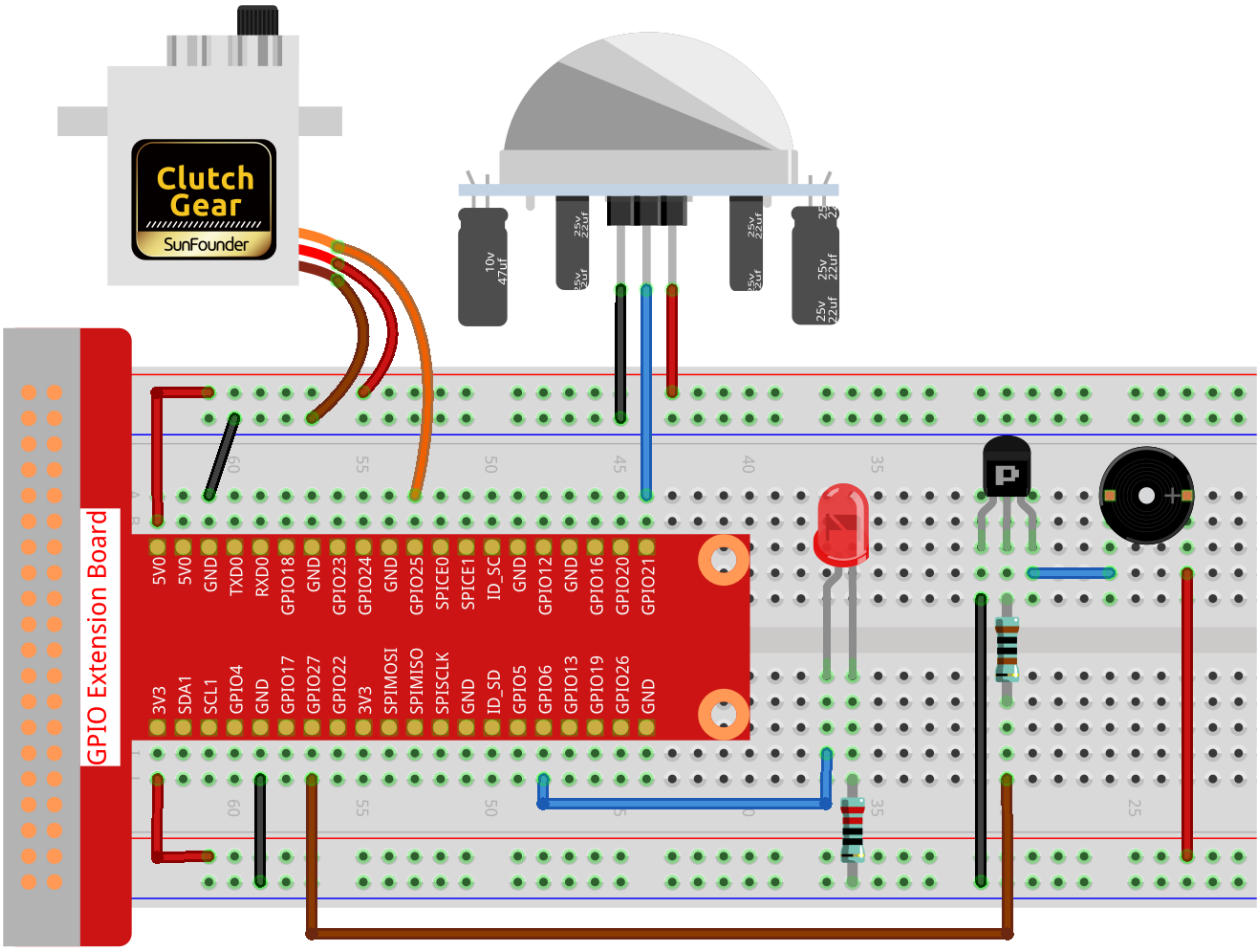
1 * Raspberry Pi 	1 * T-Extension Board	1 * PIR 	
1 * Servo	1 * Passive Buzzer	1 * S8550 PNP Transistor	1 * LED
1 * Resistor 1kΩ	1 * Resistor 220Ω	Several Jumper Wires	
1 * 40-pin Cable			
1 * Breadboard 			

回路図



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.2_Welcome_zero.py
```

コードが実行されると、PIR センサーが通行人を検出した場合、扉は自動的に開き（サーボで模倣）、インジケーターライトが点灯し、開店ベルの音楽が再生されます。ドアベルの音楽が再生された後、システムは自動的に扉を閉じ、インジケーターライトを消灯し、次回の通行人を待機します。

PIR モジュールには 2 つのポテンショメータがあります。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す。PIR モジュールをより良く動作させるには、これらのポテンショメータを両方とも反時計回りにまわす必要があります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、その前に raphael-kit/python-pi5 のソースコードパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

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

from gpiozero import LED, MotionSensor, Servo, TonalBuzzer
import time

# LED、モーションセンサー（PIR）、ブザーのための GPIO ピン設定
ledPin = LED(6)
pirPin = MotionSensor(21)
buzPin = TonalBuzzer(27)

# サーボモーターパルス幅補正ファクターと計算
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # 最大パルス幅
minPW = (1.0 - myCorrection) / 1000  # 最小パルス幅

# サーボをカスタムパルス幅で初期化
servoPin = Servo(25, min_pulse_width=minPW, max_pulse_width=maxPW)
```

(次のページに続く)

(前のページからの続き)

```
# ブザーの音楽のチューン、音符と持続時間が含まれています
tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2),
        ('Eb4', 0.2), ('E4', 0.2), (None, 0.6),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.6),
        ('Eb4', 0.2), ('E4', 0.2), (None, 0.2),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
        ('C4', 0.2), ('B4', 0.2), (None, 0.2),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
        ('B4', 0.2), ('Bb4', 0.5), (None, 0.6),
        ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
        ('D4', 0.2), ('E4', 0.2)]

def setAngle(angle):
    """
    サーボを指定した角度に移動します。
    :param angle: 角度（0-180）。
    """
    value = float(angle / 180)  # 角度をサーボの値に変換
    servoPin.value = value      # サーボの位置を設定
    time.sleep(0.001)           # サーボの動作のための短い遅延

def doorbell():
    """
    ブザーを使用して音楽を演奏します。
    """
    for note, duration in tune:
        buzPin.play(note)       # 音符を演奏
        time.sleep(float(duration))  # 音符の持続時間
    buzPin.stop()               # チューン再生後にブザーを停止

def closedoor():
    # LED をオフにし、サーボを使ってドアを閉めます
    ledPin.off()
    for i in range(180, -1, -1):
        setAngle(i)             # サーボを 180 度から 0 度まで移動
        time.sleep(0.001)       # スムーズな動作のための短い遅延
    time.sleep(1)               # ドアを閉めた後の待機

def opendoor():
```

(次のページに続く)

(前のページからの続き)

```
    # LED をオンにし、ドアを開ける（サーボを移動）、チューンを演奏し、ドアを閉じる
    ledPin.on()
    for i in range(0, 181):
        setAngle(i)             # サーボを 0 から 180 度まで移動
        time.sleep(0.001)       # スムーズな動作のための短い遅延
    time.sleep(1)               # チューンを演奏する前の待機
    doorbell()                  # ドアベルのチューンを演奏
    closedoor()                 # チューン演奏後にドアを閉じる

def loop():
    # モーションを確認し、ドアを操作するためのメインループ
    while True:
        if pirPin.motion_detected:
            opendoor()              # モーションが検出された場合はドアを開く
        time.sleep(0.1)             # ループ内の短い遅延

try:
    loop()
except KeyboardInterrupt:
    # ユーザーの割り込み（たとえば、Ctrl+C）で GPIO をクリーンアップ
    buzPin.stop()
    ledPin.off()
```

代解

1. 脚本は必要なモジュールをインポートすることから始まります。 gpiozero ライブラリは、LED、モーションセンサー、サーボモーター、音楽ブザーとのインターフェースを提供するために使用されます。 time モジュールはタイミング関連の機能を処理するために使用されます。

```
#!/usr/bin/env python3
from gpiozero import LED, MotionSensor, Servo, TonalBuzzer
import time
```

2. LED、PIR モーションセンサー、音楽ブザーをそれぞれの GPIO ピンに初期化します。

```
# LED、モーションセンサー（PIR）、ブザーのための GPIO ピン設定
ledPin = LED(6)
pirPin = MotionSensor(21)
buzPin = TonalBuzzer(27)
```

3. サーボモーターの最大および最小パルス幅を計算し、微調整のための補正ファクターを組み込みます。

```
# サーボモーターパルス幅補正ファクターと計算
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # 最大パルス幅
minPW = (1.0 - myCorrection) / 1000  # 最小パルス幅
```

4. サーボモーターを正確な位置に配置するためにカスタムパルス幅を使用して GPIO ピン 25 上で初期化します。

```
# サーボをカスタムパルス幅で初期化
servoPin = Servo(25, min_pulse_width=minPW, max_pulse_width=maxPW)
```

5. ブザーで演奏するための音楽チューンを、音符（周波数）と持続時間（秒）のシーケンスとして定義します。

```
# ブザーの音楽のチューン、音符と持続時間が含まれています
tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2),
        ('Eb4', 0.2), ('E4', 0.2), (None, 0.6),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.6),
        ('Eb4', 0.2), ('E4', 0.2), (None, 0.2),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
        ('C4', 0.2), ('B4', 0.2), (None, 0.2),
        ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
        ('B4', 0.2), ('Bb4', 0.5), (None, 0.6),
        ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
        ('D4', 0.2), ('E4', 0.2)]
```

6. 指定された角度にサーボを移動するための関数。角度をサーボの値に変換します。

```
def setAngle(angle):
    """
    サーボを指定した角度に移動します。
    :param angle: 角度（0-180）。
    """
    value = float(angle / 180)  # 角度をサーボの値に変換
    servoPin.value = value      # サーボの位置を設定
    time.sleep(0.001)           # サーボの動作のための短い遅延
```

7. ブザーを使用して音楽を演奏するための関数。 tune リスト内を繰り返し、各音符を指定された持続時間で演奏します。

```
def doorbell():
    """
```

(次のページに続く)

(前のページからの続き)

```
    ブザーを使用して音楽を演奏します。
    """
    for note, duration in tune:
        buzPin.play(note)       # 音符を演奏
        time.sleep(float(duration))  # 音符の持続時間
    buzPin.stop()               # チューン再生後にブザーを停止
```

8. サーボモーターを使用してドアを開閉するための関数。 opendoor 関数は LED を点灯させ、ドアを開け、音楽を演奏し、その後ドアを閉じます。

```
def closedoor():
    # LED をオフにし、サーボを使ってドアを閉めます
    ledPin.off()
    for i in range(180, -1, -1):
        setAngle(i)             # サーボを 180 度から 0 度まで移動
        time.sleep(0.001)       # スムーズな動作のための短い遅延
    time.sleep(1)               # ドアを閉めた後の待機

def opendoor():
    # LED をオンにし、ドアを開ける（サーボを移動）、チューンを演奏し、ドアを閉じる
    ledPin.on()
    for i in range(0, 181):
        setAngle(i)             # サーボを 0 から 180 度まで移動
        time.sleep(0.001)       # スムーズな動作のための短い遅延
    time.sleep(1)               # チューンを演奏する前の待機
    doorbell()                  # ドアベルのチューンを演奏
    closedoor()                 # チューン演奏後にドアを閉じる
```

9. モーション検出を常にチェックするメインループ。モーションが検出された場合、 opendoor 関数がトリガーされます。

```
def loop():
    # モーションを確認し、ドアを操作するためのメインループ
    while True:
        if pirPin.motion_detected:
            opendoor()          # モーションが検出された場合はドアを開く
        time.sleep(0.1)         # ループ内の短い遅延
```

10. メインループを実行し、スクリプトをキーボードコマンド（Ctrl+C）で停止できるようにし、クリーンな終了のためにブザーと LED をオフにします。

```
try:
    loop()
except KeyboardInterrupt:
    # ユーザーの割り込み（たとえば、Ctrl+C）で GPIO をクリーンアップ
    buzPin.stop()
    ledPin.off()
```

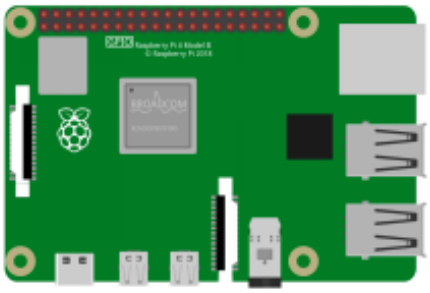
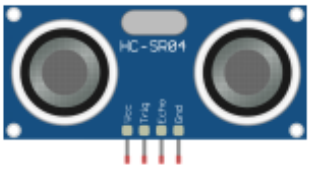
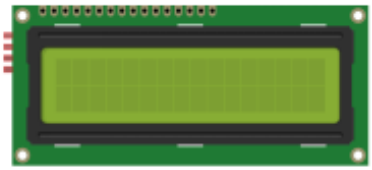
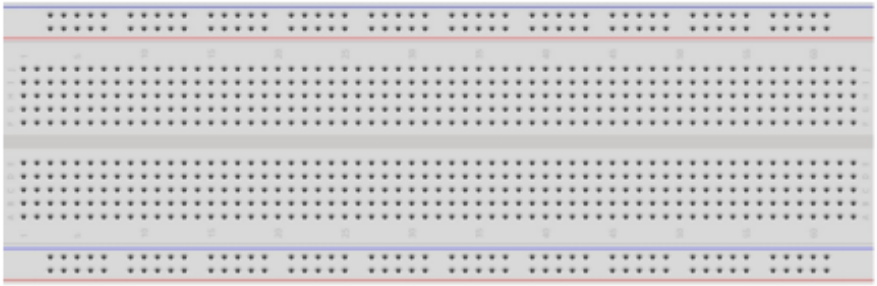
3.1.3 リバースアラーム

はじめに

このプロジェクトでは、LCD、ブザー、超音波センサーを使用して、リバースアシストシステムを作成します。これをリモートコントロール車に取り付けて、車をガレージにバックさせる実際のプロセスをシミュレートできます。

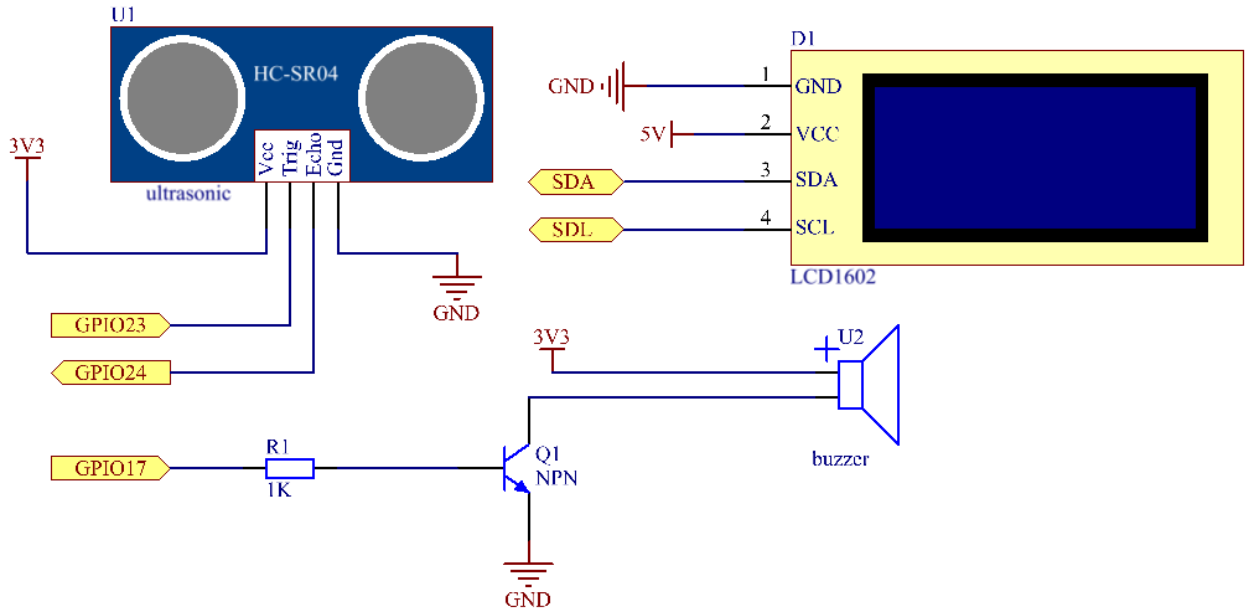
必要な部品

このプロジェクトには、以下の部品が必要です。

1 * Raspberry Pi 	1 * T-Extension Board	1 * Active Buzzer
		1 * S8050 NPN Transistor
1 * HC SR04 	1 * I2C LCD1602 	Several Jumper Wires
		1 * Resistor(1kΩ)
1 * 40-pin Cable		
1 * Breadboard 		

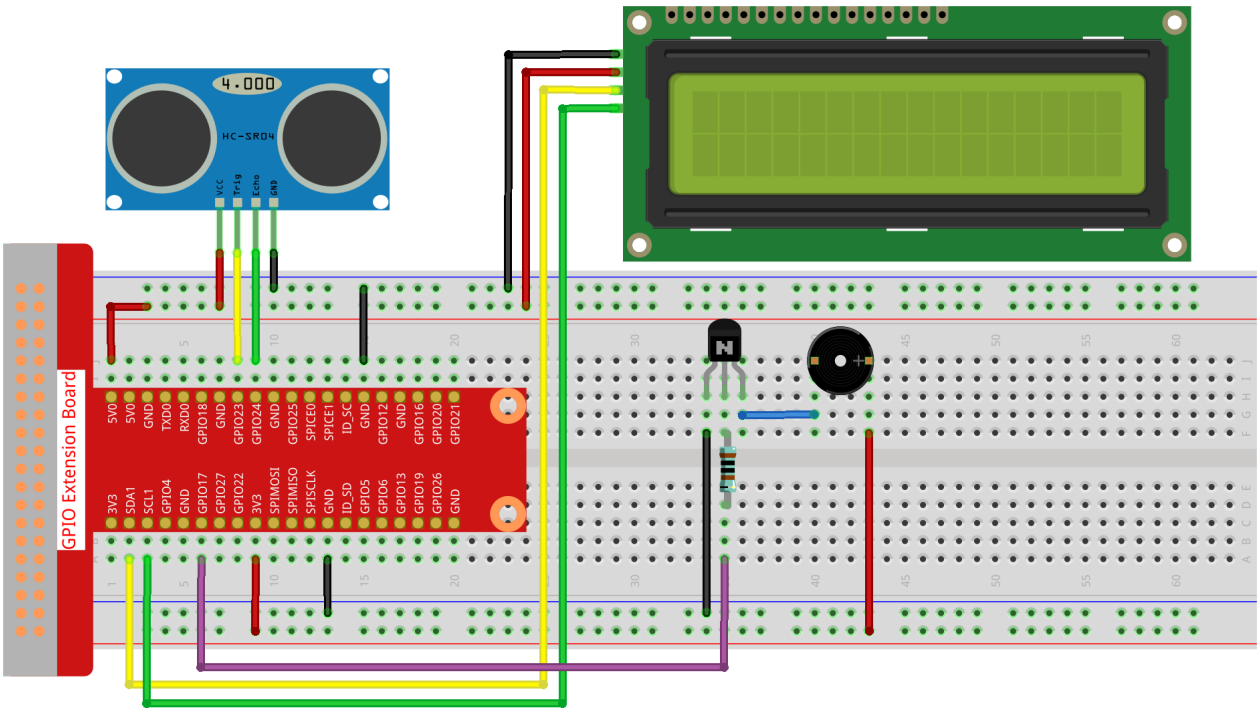
回路図

超音波センサーは、障害物との距離をコードの形で LCD に表示します。同時に、超音波センサーは異なる距離値に応じてブザーから警告音を出力します。



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.3_ReversingAlarm_zero.py
```

このコードを実行すると、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、その距離情報を LCD1602 に表示し、同時にブザーから距離に応じて周波数が変化する警告音が鳴ります。

注釈:

- FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1' エラーが発生した場合は、I2C を有効にするために *I2C* **構成** を参照してください。
- ModuleNotFoundError: No module named 'smbus2' エラーが表示された場合は、sudo pip3 install smbus2 を実行してください。
- エラー OSError: [Errno 121] Remote I/O error が表示された場合、モジュールの接続が誤っているか、モジュールが壊れていることを意味します。
- コードと配線は正常ですが、LCD にコンテンツが表示されない場合は、バックのポテンショメーターを回してコントラストを増加させることができます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、変更する前に raphael-kit/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
import LCD1602
import time
from gpiozero import DistanceSensor, Buzzer

# GPIO ピンとともに超音波センサーを初期化します
sensor = DistanceSensor(echo=24, trigger=23)  # Echo ピンは GPIO 24、Trigger ピンは GPIO 23 です

# GPIO ピン 17 に接続されたブザーを初期化します
buzzer = Buzzer(17)

def lcdsetup():
```

(次のページに続く)

(前のページからの続き)

```
    # I2C アドレスとバックライトを有効にして LCD を初期化します
    LCD1602.init(0x27, 1)  # 0x27 は LCD の I2C アドレスです
    LCD1602.clear()  # LCD ディスプレイをクリアします
    # LCD に起動メッセージを表示します
    LCD1602.write(0, 0, ' 超音波センサー起動中')
    LCD1602.write(1, 1, 'SunFounder 製')
    time.sleep(2)  # 2 秒待機します

def distance():
    # センサーによって測定された距離を計算して返します
    dis = sensor.distance * 100  # 距離をセンチメートルに変換します
    print(' 距離: {:.2f} cm'.format(dis))  # 小数点以下 2 桁の距離を表示します
    time.sleep(0.3)  # 次の測定まで 0.3 秒待機します
    return dis

def loop():
    # 距離を連続して測定し、LCD とブザーを更新します
    while True:
        dis = distance()  # 現在の距離を取得します
        # 距離に基づいて距離を表示し、アラートを処理します
        if dis > 400:  # 距離が範囲外かどうかをチェックします
            LCD1602.clear()
            LCD1602.write(0, 0, ' エラー')
            LCD1602.write(3, 1, ' 範囲外')
            time.sleep(0.5)
        else:
            # LCD に現在の距離を表示します
            LCD1602.clear()
            LCD1602.write(0, 0, ' 距離:')
            LCD1602.write(5, 1, str(round(dis, 2)) + ' cm')
            # 距離に基づいてブザーの周波数を調整します
            if dis >= 50:
                time.sleep(0.5)
            elif 20 < dis < 50:
                # 中距離：中程度のブザー周波数
                for _ in range(2):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
```

(次のページに続く)

(前のページからの続き)

```
                time.sleep(0.2)
            elif dis <= 20:
                # 近距離：高いブザー周波数
                for _ in range(5):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.05)

try:
    lcdsetup()  # LCD ディスプレイをセットアップ
    loop()      # 測定ループを開始
except KeyboardInterrupt:
    # ユーザーによる割り込み（例：Ctrl+C）でブザーをオフにして LCD をクリアします
    buzzer.off()
    LCD1602.clear()
```

コードの説明

1. このスクリプトでは、さまざまなライブラリを機能に活用しています。 LCD1602 ライブラリは LCD ディスプレイを管理し、 time は時間に関連する機能を提供します。 gpiozero ライブラリは、Raspberry Pi の GPIO ピン操作に必要であり、特に DistanceSensor と Buzzer の操作に使用されます。

   ```
   #!/usr/bin/env python3
   import LCD1602
   import time
   from gpiozero import DistanceSensor, Buzzer
   ```

2. 超音波センサーは、エコーとトリガーピンがそれぞれ GPIO ピン 24 と 23 に接続されてセットアップされています。さらに、ブザーは GPIO ピン 17 に接続されています。

   ```
   # GPIO ピンで超音波センサーを初期化
   sensor = DistanceSensor(echo=24, trigger=23)  # エコーピンは GPIO 24、トリガーピンは
   GPIO 23 です

   # GPIO ピン 17 に接続されたブザーを初期化
   buzzer = Buzzer(17)
   ```

3. LCD ディスプレイの初期化では、クリアしてから起動メッセージを表示する手順が含まれています。

```
def lcdsetup():
    # I2C アドレスとバックライトを有効にして LCD を初期化
    LCD1602.init(0x27, 1)  # 0x27 は LCD の I2C アドレスです
    LCD1602.clear()  # LCD ディスプレイをクリア
    # LCD 上に起動メッセージを表示
    LCD1602.write(0, 0, 'Ultrasonic Starting')
    LCD1602.write(1, 1, 'By SunFounder')
    time.sleep(2)  # 2 秒待機
```

4. distance 関数は超音波センサーによって測定された距離を計算し、センチメートル単位で値を返します。

```
def distance():
    # センサーによって測定された距離を計算して返す
    dis = sensor.distance * 100  # 距離をセンチメートルに変換
    print('Distance: {:.2f} cm'.format(dis))  # 距離を小数点以下 2 桁で表示
    time.sleep(0.3)  # 次の測定前に 0.3 秒待機
    return dis
```

5. メインループは距離を連続的に測定し、LCD とブザーの両方を更新します。測定された距離に基づいて異なる距離範囲を処理し、エラーメッセージを表示したり、測定された距離に基づいてブザーの周波数を変更したりします。

```
def loop():
    # 距離を連続的に測定し、LCD とブザーを更新
    while True:
        dis = distance()  # 現在の距離を取得
        # 距離に基づいてアラートを表示し、処理
        if dis > 400:  # 距離が範囲外かどうかをチェック
            LCD1602.clear()
            LCD1602.write(0, 0, 'Error')
            LCD1602.write(3, 1, 'Out of range')
            time.sleep(0.5)
        else:
            # LCD 上に現在の距離を表示
            LCD1602.clear()
            LCD1602.write(0, 0, 'Distance is')
            LCD1602.write(5, 1, str(round(dis, 2)) + ' cm')
            # 距離に基づいてブザーの周波数を調整
            if dis >= 50:
                time.sleep(0.5)
```

(次のページに続く)

(前のページからの続き)

```
            elif 20 < dis < 50:
                # 中距離：中程度のブザーの周波数
                for _ in range(2):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.2)
            elif dis <= 20:
                # 近距離：高いブザーの周波数
                for _ in range(5):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.05)
```

6. 実行時、スクリプトは LCD をセットアップし、メインループに入ります。キーボードコマンド（Ctrl+C）で割り込みが発生すると、ブザーがオフになり、LCD がクリアされます。

```
try:
    lcdsetup()  # LCD ディスプレイのセットアップ
    loop()      # 測定ループの開始
except KeyboardInterrupt:
    # ユーザー割り込み（Ctrl+C など）でブザーをオフにし、LCD をクリア
    buzzer.off()
    LCD1602.clear()
```

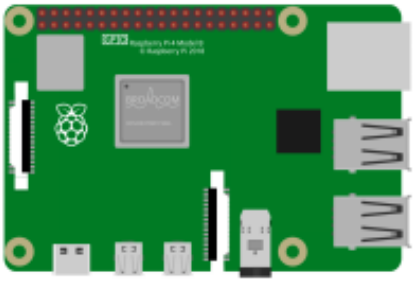
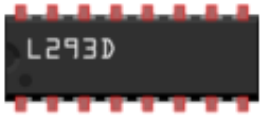
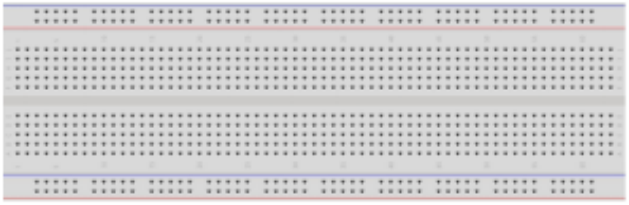
3.1.4 スマートファン

はじめに

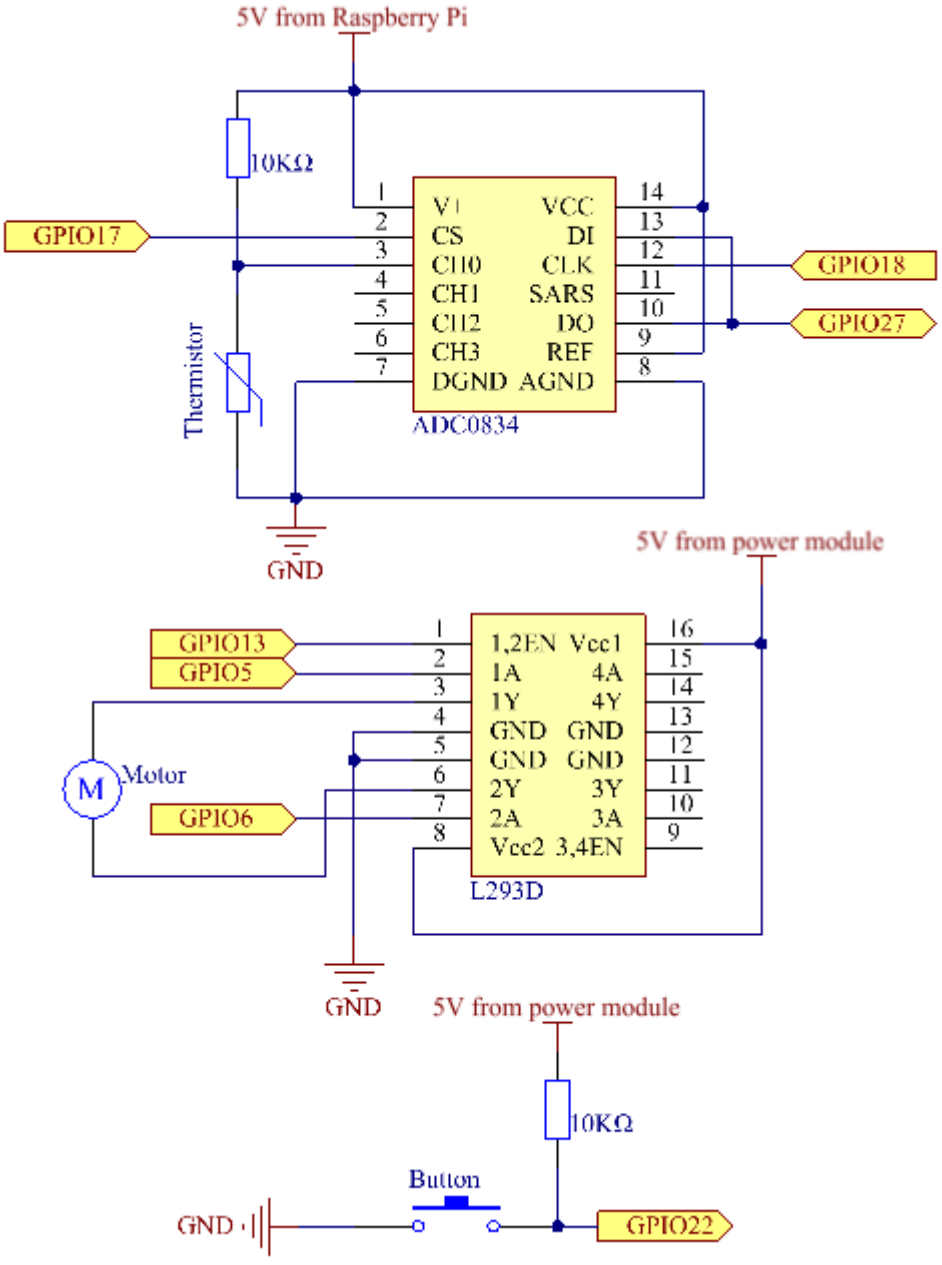
このプロジェクトでは、モーター、ボタン、サーミスタを使用して、風速調節可能なマニュアル + 自動スマートファンを作成します。

必要な部品

このプロジェクトで必要な部品は次のとおりです。

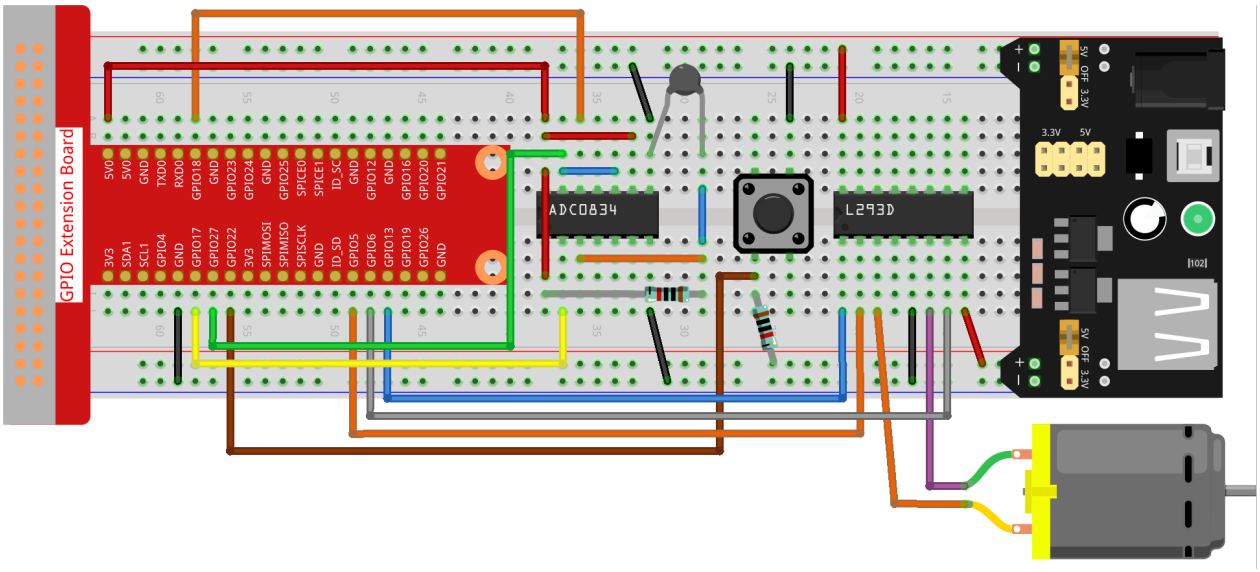
1 * Raspberry Pi 	1 * T-Extension Board	1 * Power Module 2 * Resistor 10KΩ
1 * 40-pin Cable	1 * Thermistor	1 * L293D  1 * ADC0834
1 * Breadboard 	1 * Button	1 * DC Motor Several Jumper Wires

回路図


5V from Raspberry Pi
10KΩ
GPIO17
GPIO18
GPIO27
Thermistor
V+
CS
CH0
CH1
CH2
CH3
DGND
VCC
DI
CLK
DO
REF
AGND
ADC0834
GND
5V from power module
GPIO13
GPIO5
GPIO6
Motor
M
1,2EN
1A
1Y
GND
GND
2Y
2A
Vcc2
Vcc1
4A
4Y
GND
GND
3Y
3A
3,4EN
L293D
GND
5V from power module
10KΩ
Button
GND
GPIO22

実験手順

ステップ 1: 回路を組み立てます。



注釈: 電源モジュールにはキット内の 9V バッテリーと 9V バッテリーバックルを使用できます。電源モジュールのジャンパーキャップを、ブレッドボードの 5V バスストリップに挿入します。

ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.4_SmartFan_zero.py
```

コードが実行されると、ボタンを押してファンを起動します。ボタンを押すたびに、風速が 1 段階上下に調節され

ます。風速は **0〜4** の **5 つ** の段階があります。4 番目の風速に設定されており、ボタンを押すと風速 **0** でファンが停止します。

温度が 2 ℃以上上昇または下降すると、速度は自動的に 1 段階高くまたは低くなります。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、変更する前に `raphael-kit/python-pi5` のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

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

from gpiozero import Motor, Button
from time import sleep
import ADC0834
import math

# ボタンとモーター制御のための GPIO ピンを初期化します
BtnPin = Button(22)
motor = Motor(forward=5, backward=6, enable=13)

# 温度センシング用に ADC0834 モジュールを初期化します
ADC0834.setup()

# モーター速度レベルと温度を追跡するための変数を初期化します
level = 0
currentTemp = 0
markTemp = 0

def temperature():
    """
    センサーから現在の温度を読み取り、計算します。
    戻り値:
        float: 摂氏での現在の温度。
    """
    # ADC0834 モジュールからアナログ値を読み取ります
    analogVal = ADC0834.getResult()
    # アナログ値を電圧に変換し、抵抗に変換します
```

(次のページに続く)

(前のページからの続き)

```
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    # 摂氏での温度を計算します
    temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))
    Cel = temp - 273.15
    return Cel

def motor_run(level):
    """
    指定されたレベルに基づいてモーターの速度を調整します。
    引数:
        level (int): 望ましいモーター速度レベル。
    戻り値:
        int: 調整されたモーター速度レベル。
    """
    # レベルが 0 の場合、モーターを停止します
    if level == 0:
        motor.stop()
        return 0
    # 最大速度のためにレベルを 4 で制限します
    if level >= 4:
        level = 4
    # モーターの速度を設定します
    motor.forward(speed=float(level / 4))
    return level

def changeLevel():
    """
    ボタンが押されたときにモーターの速度レベルを手動で変更し、参照温度を更新します。
    """
    global level, currentTemp, markTemp
    print("ボタンが押されました")
    # レベルを 0 から 4 までサイクルさせます
    level = (level + 1) % 5
    # 参照温度を更新します
    markTemp = currentTemp

# ボタンの押下イベントを changeLevel 関数にバインドします
BtnPin.when_pressed = changeLevel
```

(次のページに続く)

(前のページからの続き)

```
def main():
    """
    温度変化に連続的に監視し、応答するためのメイン関数。
    """
    global level, currentTemp, markTemp
    # 初期参照温度を設定します
    markTemp = temperature()
    while True:
        # 現在の温度を連続して読み取ります
        currentTemp = temperature()
        # 温度差に基づいてモーターのレベルを調整します
        if level != 0:
            if currentTemp - markTemp <= -2:
                level -= 1
                markTemp = currentTemp
            elif currentTemp - markTemp >= 2:
                if level < 4:
                    level += 1
                markTemp = currentTemp
        # 調整されたレベルでモーターを実行します
        level = motor_run(level)

# メイン関数を実行し、キーボード割り込みを処理します
try:
    main()
except KeyboardInterrupt:
    # スクリプトが中断された場合、モーターを停止します
    motor.stop()
```

コードの説明

1. モーターやボタンの管理用クラス、一時停止を導入するための sleep 関数、温度センシングのための ADC0834 ライブラリ、数学的な計算のための math ライブラリをインポートします。

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

from gpiozero import Motor, Button
from time import sleep
```

(次のページに続く)

(前のページからの続き)

```
import ADC0834
import math
```

2. ボタンを GPIO ピン 22 に設定し、モーターの制御に特定の GPIO ピンを設定します。温度測定用に ADC0834 モジュールを初期化します。また、モーター速度レベルと温度を監視するための変数を初期化します。

```
# ボタンとモーター制御のための GPIO ピンを初期化
BtnPin = Button(22)
motor = Motor(forward=5, backward=6, enable=13)

# 温度センシングのために ADC0834 モジュールを初期化
ADC0834.setup()

# モーター速度レベルと温度を追跡するための変数を初期化
level = 0
currentTemp = 0
markTemp = 0
```

3. センサーから温度を読み取り、摂氏に変換するための関数を定義します。

```
def temperature():
    """
    センサーから現在の温度を読み取り、計算します。
    戻り値:
        float: 摂氏での現在の温度。
    """
    # ADC0834 モジュールからアナログ値を取得
    analogVal = ADC0834.getResult()
    # アナログ値を電圧に変換し、抵抗に変換
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    # 摂氏での温度を計算
    temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))
    Cel = temp - 273.15
    return Cel
```

4. 指定したレベルに基づいてモーターの速度を調整する関数を実装します。

```
def motor_run(level):
    """
    指定されたレベルに基づいてモーターの速度を調整します。
    引数:
        level (int): 望ましいモーター速度レベル。
    戻り値:
        int: 調整されたモーター速度レベル。
    """
    # レベルが 0 の場合、モーターを停止
    if level == 0:
        motor.stop()
        return 0
    # レベルを最大 4 に制限
    if level >= 4:
        level = 4
    # モーター速度を設定
    motor.forward(speed=float(level / 4))
    return level
```

5. ボタンを使用してモーターの速度レベルを手動で変更し、この関数をボタンのプレスイベントにバインドします。

```
def changeLevel():
    """
    ボタンが押されたときにモーターの速度レベルを手動で変更し、参照温度を更新します。
    """
    global level, currentTemp, markTemp
    print("ボタンが押されました")
    # レベルを 0 から 4 までサイクル
    level = (level + 1) % 5
    # 参照温度を更新
    markTemp = currentTemp

# ボタンのプレスイベントを changeLevel 関数にバインド
BtnPin.when_pressed = changeLevel
```

6. 温度変化に応じてモーターの速度を連続的に調整するメイン関数を実装することが残っています。

```
def main():
    """
```

(次のページに続く)

(前のページからの続き)

```
    温度変化に連続的に監視し、応答するためのメイン関数。
    """
    global level, currentTemp, markTemp
    # 初期参照温度を設定
    markTemp = temperature()
    while True:
        # 現在の温度を連続して読み取ります
        currentTemp = temperature()
        # レベルが 0 でない場合、温度差に基づいてモーターレベルを調整します
        if level != 0:
            if currentTemp - markTemp <= -2:
                level -= 1
                markTemp = currentTemp
            elif currentTemp - markTemp >= 2:
                if level < 4:
                    level += 1
                markTemp = currentTemp
        # 調整されたレベルでモーターを実行します
        level = motor_run(level)

# メイン関数を実行し、キーボード割り込みを処理します
try:
    main()
except KeyboardInterrupt:
    # スクリプトが中断された場合、モーターを停止します
    motor.stop()
```

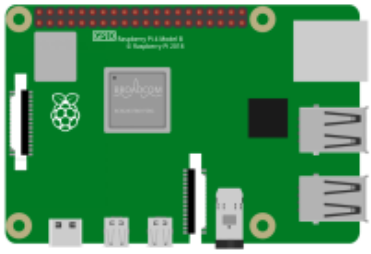
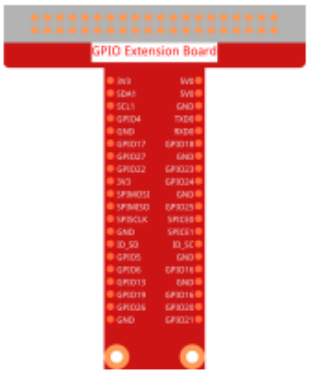
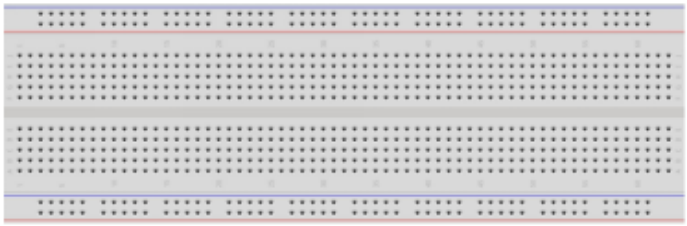
3.1.5 バッテリーインジケータ

はじめに

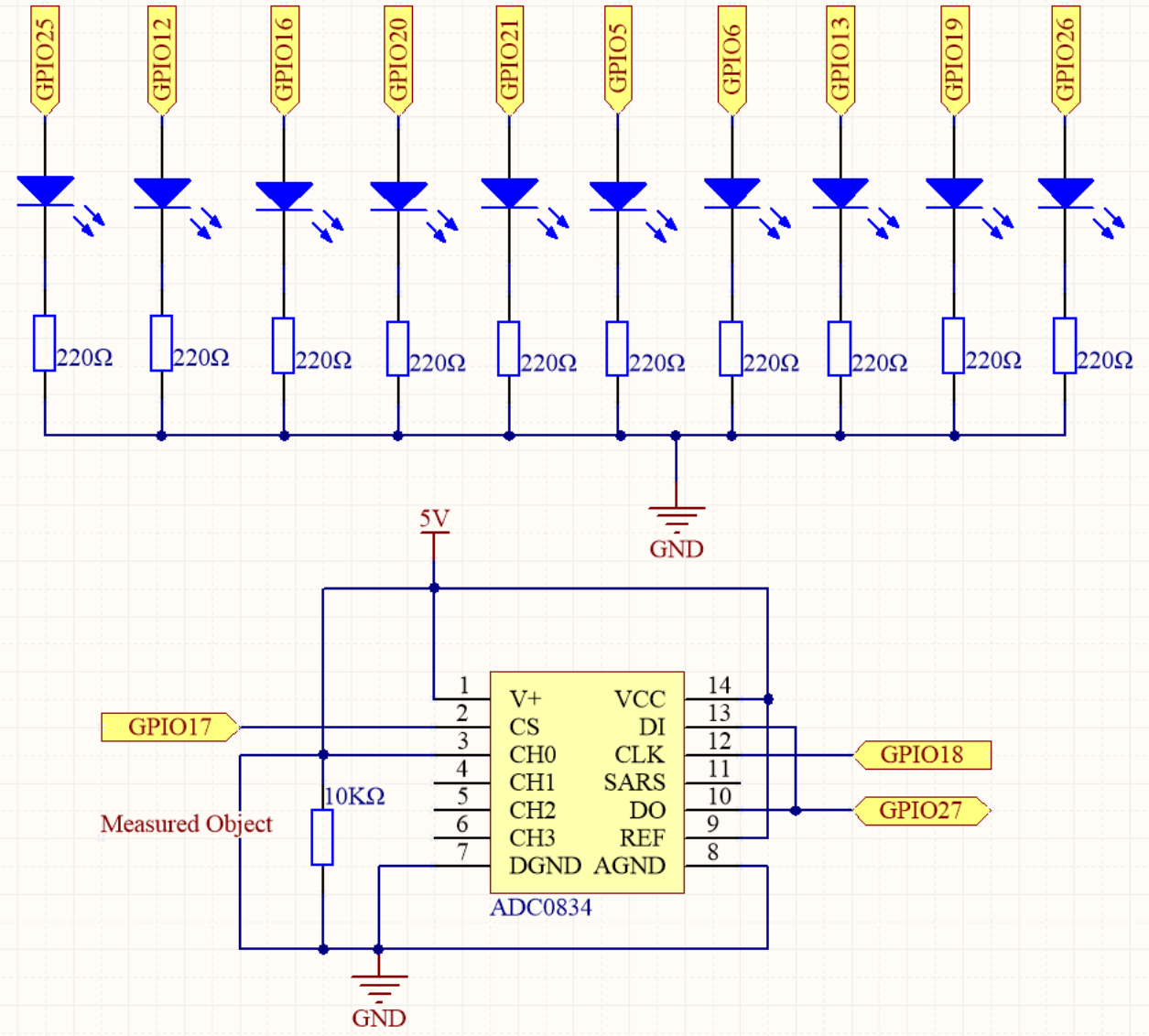
このプロジェクトでは、LED バーグラフ上にバッテリーの電力レベルを視覚的に表示できるバッテリーインジケータデバイスを作成します。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

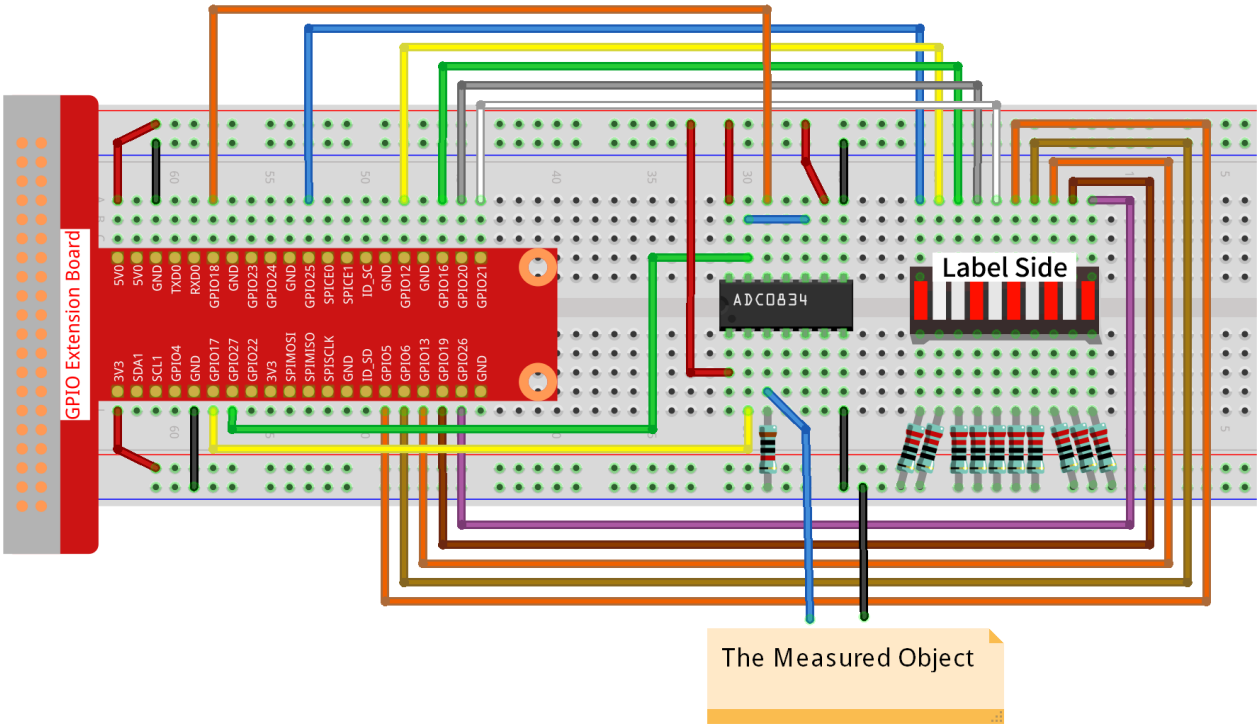
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Bargraph 1* ADC0834
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		10 * Resistor(220Ω) 1 * Resistor(10kΩ)

回路図



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 3.1.5_BatteryIndicator_zero.py
```

プログラムが実行されたら、ADC0834 の 3 番ピンと GND にそれぞれリードアウトワイヤーを接続し、それらを別々のバッテリーの 2 極にリードアウトしてください。対応する LED バーグラフの LED が点灯し、電力レベルが表示されます（測定範囲：0-5V）。

コード

注釈: 以下のコードは、 **変更/リセット/コピー/実行/停止** ができます。ただし、それに先立ち、 raphael-kit/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import LED
import ADC0834
import time
```

(次のページに続く)

(前のページからの続き)

```
# LED が接続された GPIO ピンのリスト
ledPins = [25, 12, 16, 20, 21, 5, 6, 13, 19, 26]
# リスト内の各ピンに対して LED オブジェクトを初期化
leds = [LED(pin) for pin in ledPins]

# ADC0834 モジュールのセットアップ
ADC0834.setup()

def LedBarGraph(value):
    # すべての LED をオフにする
    for i in range(10):
        leds[i].off()
    # 指定された値まで LED を点灯させる
    for i in range(value):
        leds[i].on()

try:
    # LED バーグラフを連続的に更新するためのメインループ
    while True:
        # ADC0834 からアナログ値を読み取る
        analogVal = ADC0834.getResult()
        # アナログ値を LED バーグラフのレベルに変換
        LedBarGraph(int(analogVal/25))
except KeyboardInterrupt:
    # プログラムが中断されたときにすべての LED をオフにする
    for i in range(10):
        leds[i].off()
```

コードの説明

1. このセクションは必要なライブラリをインポートしています。 gpiozero は LED を制御するため、ADC0834 は ADC モジュールとのインターフェース、 time は時間に関連する操作に使用されます。

   ```
   #!/usr/bin/env python3
   from gpiozero import LED
   import ADC0834
   import time
   ```

2. LED が接続された GPIO ピンを定義し、各ピンに対して LED オブジェクトの配列を初期化します。これに

より、配列内の各 LED を簡単に制御できます。

```
# LED が接続された GPIO ピンのリスト
ledPins = [25, 12, 16, 20, 21, 5, 6, 13, 19, 26]
# リスト内の各ピンに対して LED オブジェクトを初期化
leds = [LED(pin) for pin in ledPins]
```

3. アナログからデジタルへの変換のために ADC0834 モジュールを初期化します。

```
# ADC0834 モジュールのセットアップ
ADC0834.setup()
```

4. この関数はすべての LED をオフにし、指定された値に基づいて LED をオンにして、バーグラフの表現を作成します。

```
def LedBarGraph(value):
    # すべての LED をオフにする
    for i in range(10):
        leds[i].off()
    # 指定された値まで LED を点灯させる
    for i in range(value):
        leds[i].on()
```

5. ADC0834 からアナログ値を連続的に読み取り、この値に基づいて LED バーグラフを更新します。アナログ値は 0 から 10 の範囲にスケールダウンされます。

```
try:
    # LED バーグラフを連続的に更新するためのメインループ
    while True:
        # ADC0834 からアナログ値を読み取る
        analogVal = ADC0834.getResult()
        # アナログ値を LED バーグラフのレベルに変換
        LedBarGraph(int(analogVal/25))
```

6. プログラムが中断された場合（たとえば、Ctrl+C を押した場合）、すべての LED がオフになるようにします。

```
except KeyboardInterrupt:
    # プログラムが中断されたときにすべての LED をオフにする
    for i in range(10):
        leds[i].off()
```

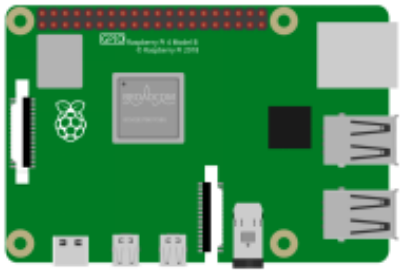
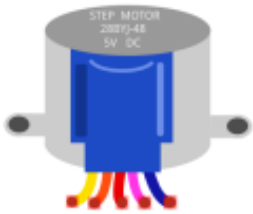
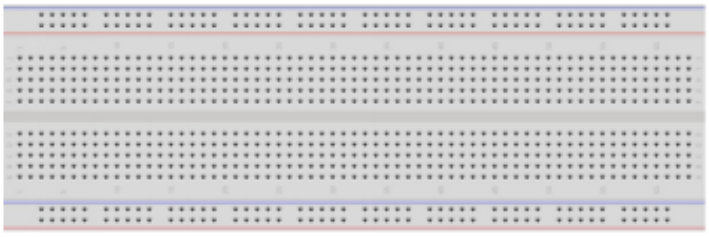
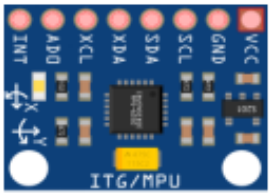
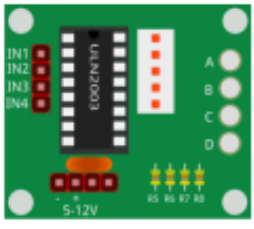
3.1.6 モーションコントロール

はじめに

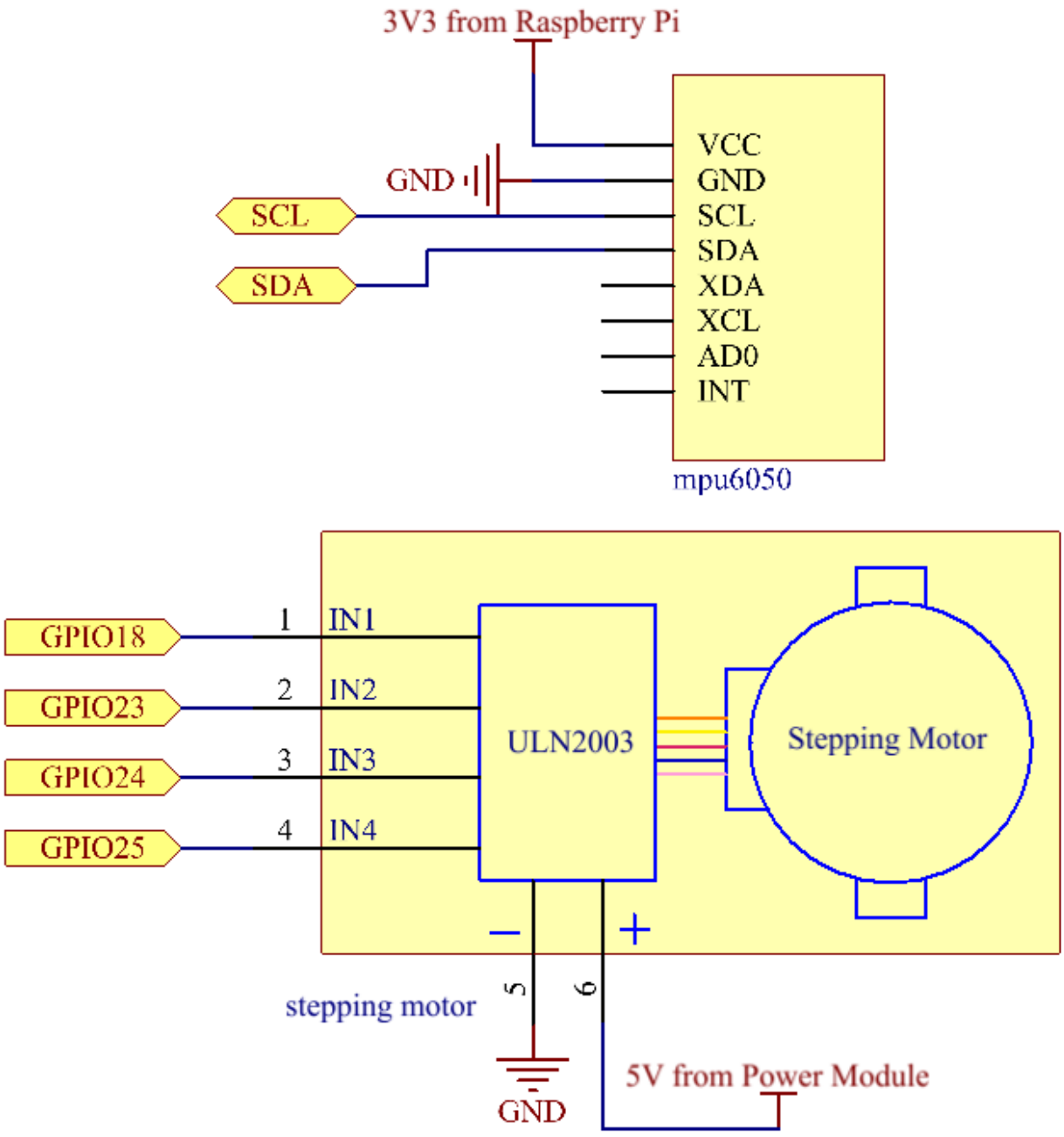
このレッスンでは、シンプルなモーションセンシングおよびコントロールデバイスを作成します。センサーとして MPU6050 を使用し、制御デバイスとしてステッパーモータを使用します。MPU6050 を手袋に取り付けることで、手首を回転させることでステッパーモータを制御できます。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

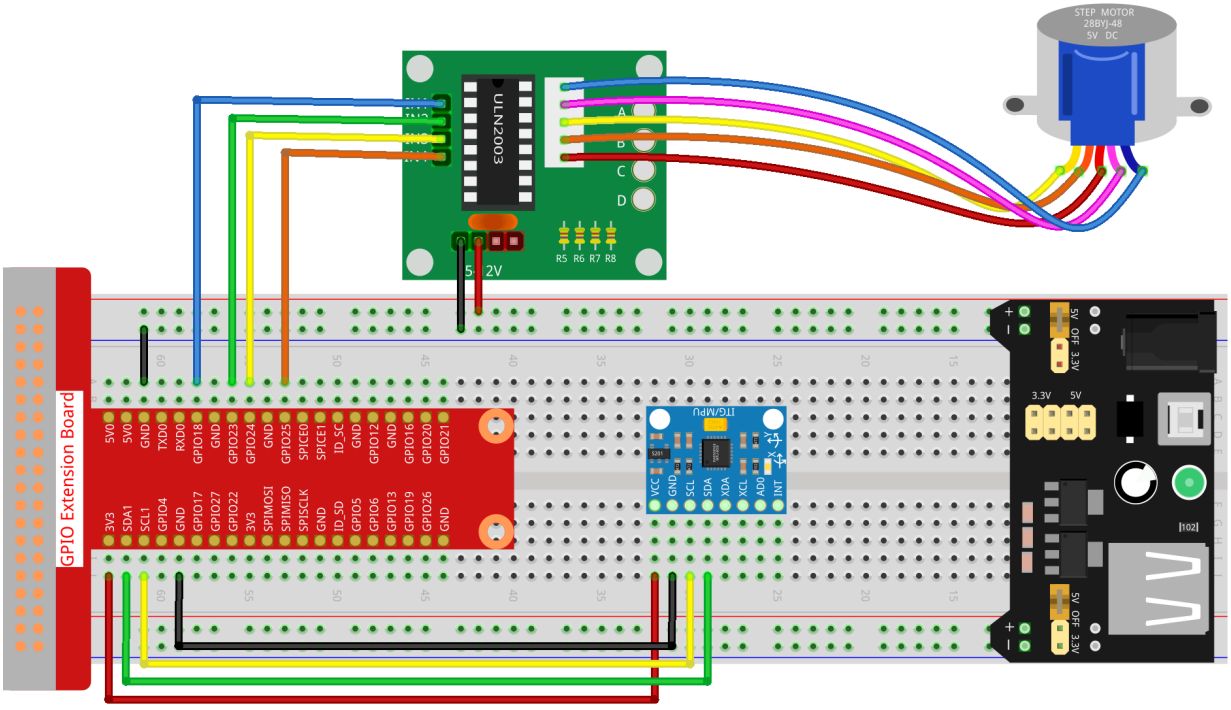
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>1 * Stepper Motor
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * Breadboard
  </td><td>1 * MPU6050 Module
  </td></tr>
<tr><td>1 * ULN2003 Module
  </td></tr>
</table>

回路図


3V3 from Raspberry Pi
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
GND
SCL
SDA
mpu6050
GPIO18
GPIO23
GPIO24
GPIO25
1
2
3
4
IN1
IN2
IN3
IN4
ULN2003
Stepping Motor
stepping motor
5
6
GND
5V from Power Module

実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードファイルを開きます。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.6_MotionControl_zero.py
```

コードが実行されると、 **mpu6050** の *Y* **軸** の傾斜角が **45** ℃ より大きい場合、ステッパーモータは反時計回りに回転し、 **-45** ℃ より小さい場合、ステッパーモータは時計回りに回転します。

コード

注釈: 以下のコードは、 **変更/リセット/コピー/実行/停止** ができます。ただし、それに先立ち、 davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
```

(次のページに続く)

(前のページからの続き)

```
import smbus
import math
import time

# MPU6050 の電源管理レジスタを初期化
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

# MPU6050 との I2C 通信の設定
bus = smbus.SMBus(1)  # SMBus の初期化
address = 0x68        # MPU6050 の I2C アドレス
bus.write_byte_data(address, power_mgmt_1, 0)  # MPU6050 を起動

# ステッパーモータのピンを GPIO ピン 18, 23, 24, 25 に初期化
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]

# モーターの回転速度パラメータを設定
rolePerMinute = 15
stepsPerRevolution = 2048
# 希望の RPM を達成するためのステップ間の遅延を計算
stepSpeed = (60 / rolePerMinute) / stepsPerRevolution

# 指定された I2C アドレスから 1 バイトを読み取る
def read_byte(adr):
    return bus.read_byte_data(address, adr)

# 指定された I2C アドレスからワード（2 バイト）を読み取る
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

# 2 の補数形式でワードを読み取る
def read_word_2c(adr):
    val = read_word(adr)
    if val >= 0x8000:
        return -((65535 - val) + 1)
    else:
```

(次のページに続く)

(前のページからの続き)

```
        return val

# 2 点間のユークリッド距離を計算
def dist(a, b):
    return math.sqrt((a * a) + (b * b))

# Y 軸回転を計算
def get_y_rotation(x, y, z):
    radians = math.atan2(x, dist(y, z))
    return -math.degrees(radians)

# X 軸回転を計算
def get_x_rotation(x, y, z):
    radians = math.atan2(y, dist(x, z))
    return math.degrees(radians)

# MPU6050 からの傾斜角を取得
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

# ステッパーモータの回転を制御
def rotary(direction):
    if direction == 'c':
        # 時計回りの回転シーケンス
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                time.sleep(stepSpeed)
    elif direction == 'a':
```

(次のページに続く)

(前のページからの続き)

```
        # 反時計回りの回転シーケンス
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                time.sleep(stepSpeed)

# 傾斜角を連続的に読み取り、モーターを制御するためのメインループ
try:
    while True:
        angle = mpu6050()
        if angle >= 45:
            rotary('a')  # 正の傾斜の場合、反時計回りに回転
        elif angle <= -45:
            rotary('c')  # 負の傾斜の場合、時計回りに回転
except KeyboardInterrupt:
    # キーボード割り込み時にすべてのモーターピンをオフにする
    for pin in motorPin:
        pin.off()
```

コードの説明

1. スクリプトは、必要なライブラリをインポートすることから始まります。GPIO ピンを制御するための「gpiozero」、I2C 通信用の「smbus」、数学的な操作に「math」、遅延を制御するための「time」をインポートします。

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice
   import smbus
   import math
   import time
   ```

2. MPU6050 センサーとの I2C 通信をセットアップします。「power_mgmt_1」と「power_mgmt_2」はセンサーの電源を管理するためのレジスタです。「power_mgmt_1」に書き込むことで、センサーは「起動」状態になります。

   ```
   # MPU6050 の電源管理レジスタを初期化
   power_mgmt_1 = 0x6b
   ```

(次のページに続く)

(前のページからの続き)

```
power_mgmt_2 = 0x6c

# MPU6050 との I2C 通信のセットアップ
bus = smbus.SMBus(1)  # SMBus の初期化
address = 0x68        # MPU6050 の I2C アドレス
bus.write_byte_data(address, power_mgmt_1, 0)  # MPU6050 を起動
```

3. Raspberry Pi 上の GPIO ピンを初期化して、ステッパーモータを制御します。各ピンはモーターのコイルに関連付けられています。

```
# ステッパーモータのピンを GPIO ピン 18, 23, 24, 25 に初期化
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]
```

4. モーターの回転数（RPM）と 1 回転あたりのステップ数を設定します。「stepSpeed」は、希望の RPM を達成するためのステップ間の遅延を計算し、スムーズなモーターの動作を確保します。

```
# モーターの回転速度パラメータを設定
rolePerMinute = 15
stepsPerRevolution = 2048
# 希望の RPM を達成するためのステップ間の遅延を計算
stepSpeed = (60 / rolePerMinute) / stepsPerRevolution
```

5. これらの関数は I2C 通信に使用されます。「read_byte」は指定されたアドレスから 1 バイトを読み取り、「read_word」は 2 バイト（ワード）を読み取り、ビット演算（ << および + ）を使用してそれらを単一の値に結合します。

```
# 指定された I2C アドレスから 1 バイトを読み取る
def read_byte(adr):
    return bus.read_byte_data(address, adr)

# 指定された I2C アドレスからワード（2 バイト）を読み取る
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
```

6. この関数は、読み取ったワードを 2 の補数形式に変換し、センサーデータから符号付きの値を解釈するために使用されます。この変換は、負のセンサー値を処理するために必要です。

```
# 2 の補数形式でワードを読み取る
def read_word_2c(adr):
    val = read_word(adr)
    if val >= 0x8000:
        return -((65535 - val) + 1)
    else:
        return val
```

7. 「dist」は 2 つのポイント間のユークリッド距離を計算し、回転計算に使用されます。「get_y_rotation」と「get_x_rotation」は「math」ライブラリの「atan2」関数を使用して Y 軸と X 軸の回転角度を計算し、結果を度に変換します。

```
# 2 点間のユークリッド距離を計算
def dist(a, b):
    return math.sqrt((a * a) + (b * b))

# Y 軸回転を計算
def get_y_rotation(x, y, z):
    radians = math.atan2(x, dist(y, z))
    return -math.degrees(radians)

# X 軸回転を計算
def get_x_rotation(x, y, z):
    radians = math.atan2(y, dist(x, z))
    return math.degrees(radians)
```

8. この関数は MPU6050 センサーから加速度計データを読み取り、読み取り値をスケーリングし、get_y_rotation 関数を使用して傾斜角を計算します。「read_word_2c」関数は 2 の補数形式でセンサーデータを読み取り、負の値を処理するために必要です。

```
# MPU6050 からの傾斜角を取得
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
```

9. 「rotary」関数はステッパーモーターの回転を制御します。指定された「direction」に基づいて時計回りまたは反時計回りの回転を実行し、特定のモーターピンをパターンに従ってオンまたはオフにします。

```
# ステッパーモーターの回転を制御
def rotary(direction):
    if direction == 'c':
        # 時計回りの回転シーケンス
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                time.sleep(stepSpeed)
    elif direction == 'a':
        # 反時計回りの回転シーケンス
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                time.sleep(stepSpeed)
```

10. メインループは MPU6050 センサーから傾斜角を連続して読み取り、角度に基づいてモーターの回転方向を制御します。プログラムが中断された場合（キーボード割り込みなど）、安全のためにすべてのモーターピンをオフにします。

```
# 傾斜角を連続的に読み取り、モーターを制御するためのメインループ
try:
    while True:
        angle = mpu6050()
        if angle >= 45:
            rotary('a')  # 正の傾斜の場合、反時計回りに回転
        elif angle <= -45:
            rotary('c')  # 負の傾斜の場合、時計回りに回転
except KeyboardInterrupt:
    # キーボード割り込み時にすべてのモーターピンをオフにする
    for pin in motorPin:
        pin.off()
```

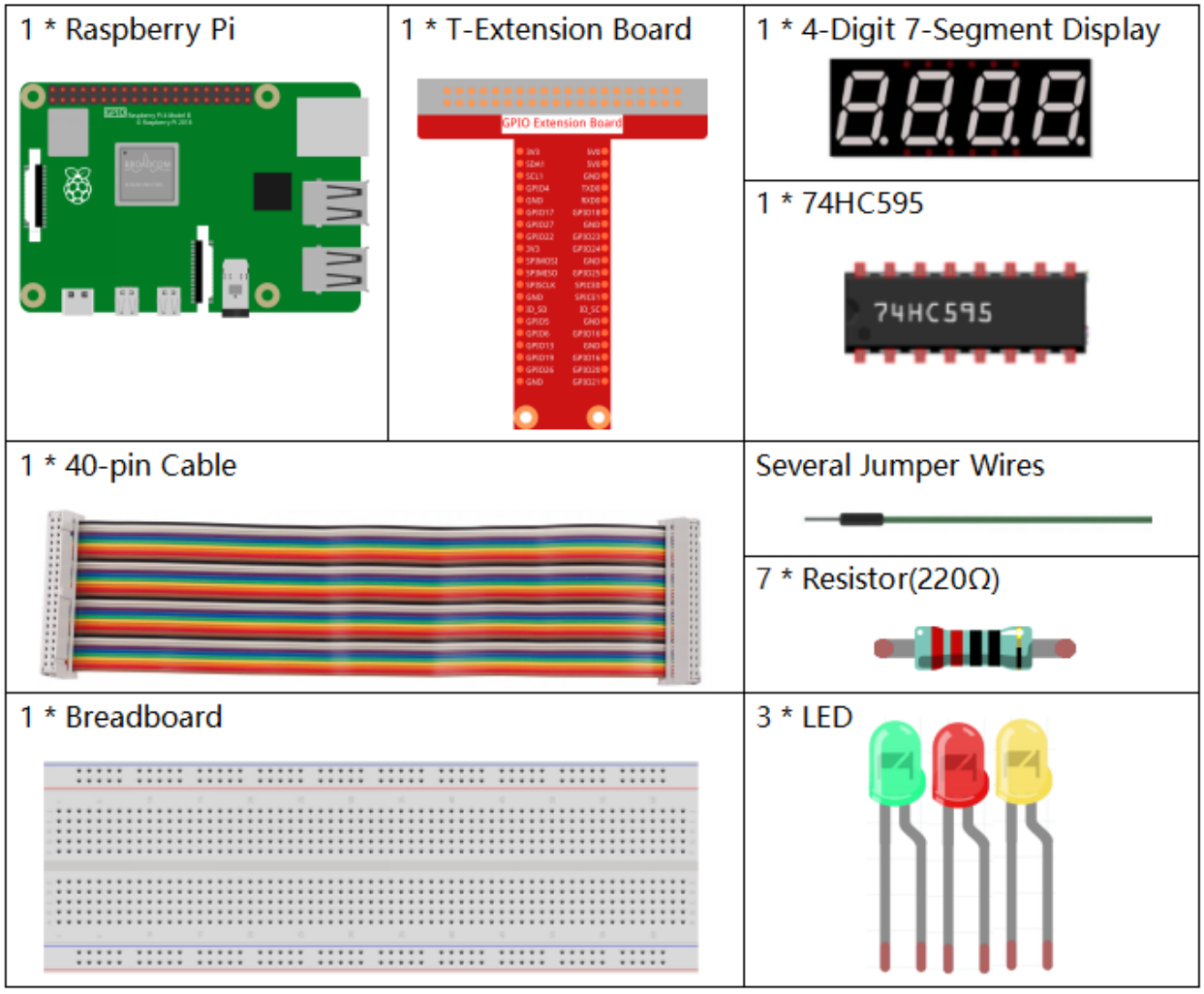
3.1.7 交通信号灯

概要

このプロジェクトでは、3 つの色の LED ライトを使用して交通信号の変更を実現し、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します。

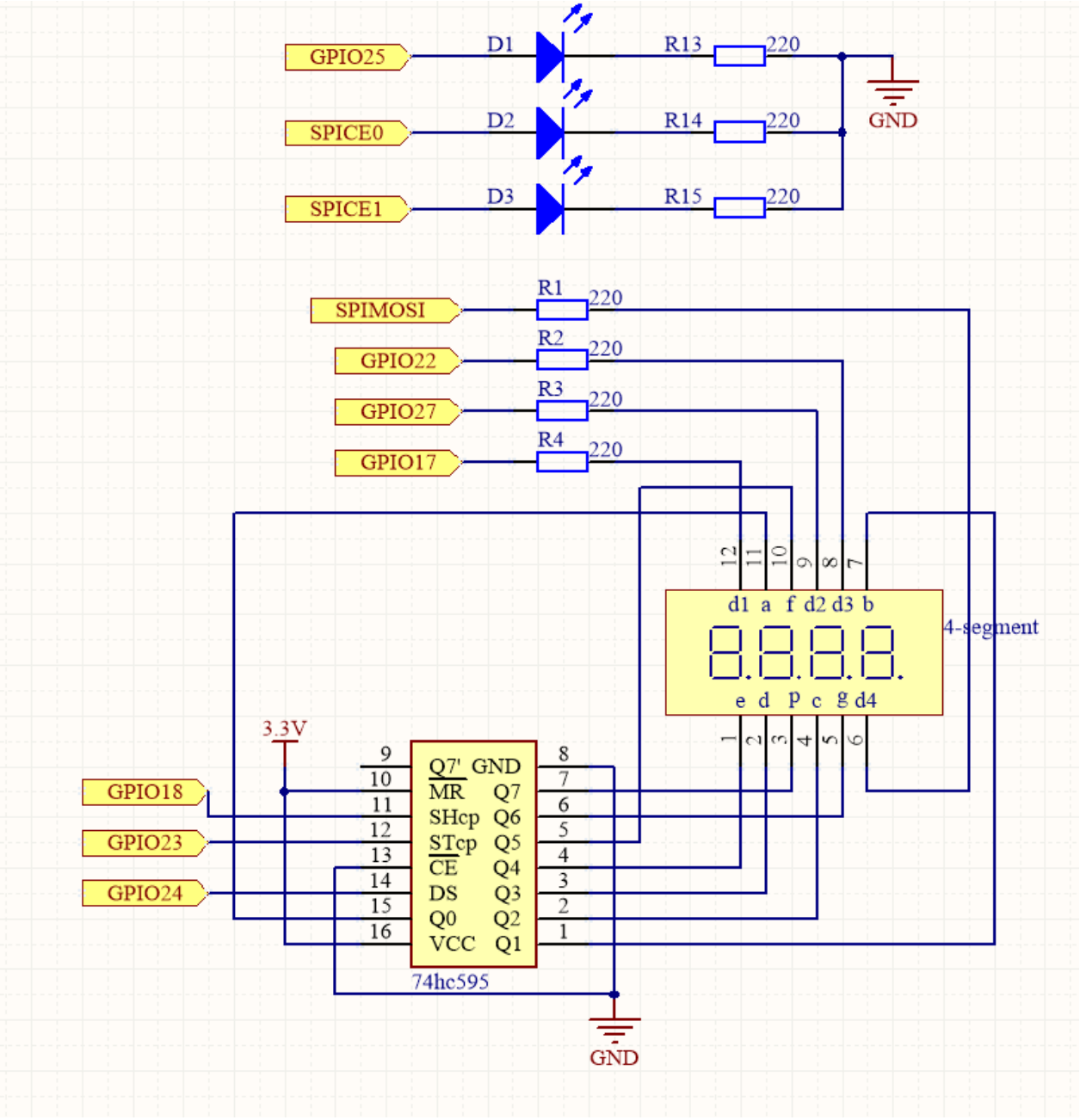
必要な部品

このプロジェクトでは、以下の部品が必要です。



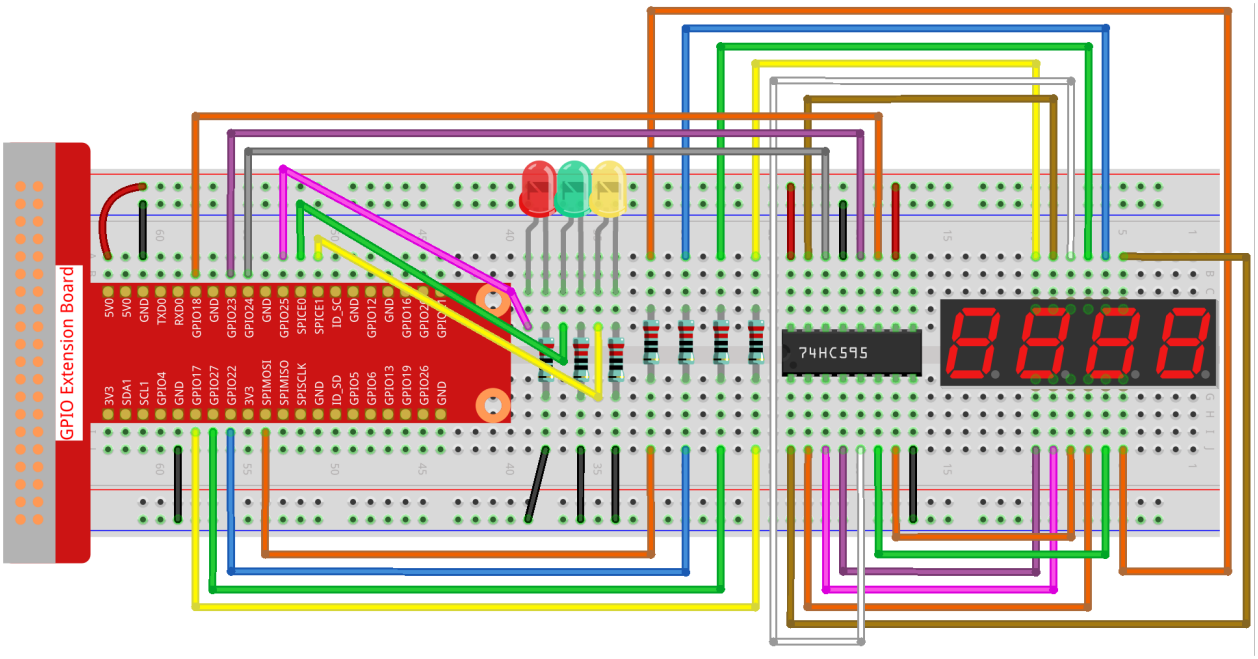
配線図

T-Board Name	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: 回路を組み立てます。



ステップ 2: ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.7_TrafficLight_zero.py
```

コードを実行すると、LED が交通信号の色変更をシミュレートします。最初に赤い LED が 60 秒点灯し、次に緑の LED が 30 秒点灯します。その後、黄色の LED が 5 秒点灯します。その後、再び赤い LED が 60 秒点灯します。このように、この一連のアクションが繰り返し実行されます。同時に、4 桁の 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ます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に、ソースコードのパス（ raphael-kit/python-pi5 のような）に移動する必要があります。コードを変更した後、それを直接実行して効果を確認できます。

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

from gpiozero import OutputDevice, LED
import threading
```

(次のページに続く)

(前のページからの続き)

```
# 74HC595 シフトレジスタ用の GPIO ピンのセットアップ
SDI = OutputDevice(24)   # シリアルデータ入力
RCLK = OutputDevice(23)  # レジスタクロック
SRCLK = OutputDevice(18) # シフトレジスタクロック

# 7 セグメントディスプレイのデジット選択用の GPIO ピンのセットアップ
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメントディスプレイに表示する 0〜9 の数字のセグメントコード
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# 交通信号の LED 用の GPIO ピンのセットアップ
ledPinR = LED(25) # 赤い LED
ledPinG = LED(8)  # 緑の LED
ledPinY = LED(7)  # 黄色の LED

# 交通信号の持続時間設定
greenLight = 30
yellowLight = 5
redLight = 60

# 交通信号の色の名前
lightColor = ("赤", "緑", "黄色")

# ステート変数の初期化
colorState = 0
counter = 60
timer1 = None

def setup():
    """ 交通信号システムを初期化し、タイマーを開始します。 """
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()

def clearDisplay():
    """ 7 セグメントディスプレイをクリアします。 """
    for _ in range(8):
```

(次のページに続く)

(前のページからの続き)

```
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ データを 74HC595 シフトレジスタにシフトしてディスプレイします。 """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    """ 7 セグメントディスプレイ上に表示する特定のディジットを選択します。 """
    for pin in placePin:
        pin.off()
    placePin[digit].on()

def timer():
    """ 交通信号の変更のタイミングを処理します。 """
    global counter, colorState, timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter -= 1
    if counter == 0:
        counter = [greenLight, yellowLight, redLight][colorState]
        colorState = (colorState + 1) % 3
    print(f"カウンタ : {counter}    色: {lightColor[colorState]}")

def lightup():
    """ 現在の状態に基づいて交通信号 LED を更新します。 """
    global colorState
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
    [ledPinR, ledPinG, ledPinY][colorState].on()
```

(次のページに続く)

(前のページからの続き)

```
def display():
    """ 7セグメントディスプレイ上に現在のカウンタ値を表示します。 """
    global counter

    for i in range(4):
        digit = counter // (10 ** (3 - i)) % 10
        if i == 0 and digit == 0:
            continue
        clearDisplay()
        pickDigit(3 - i)
        hc595_shift(number[digit])

def loop():
    """ ディスプレイとライトを連続的に更新するメインループ。 """
    while True:
        display()
        lightup()

def destroy():
    """ スクリプトが終了した際にリソースをクリーンアップします。 """
    global timer1
    timer1.cancel()
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()

try:
    setup()
    loop()
except KeyboardInterrupt:
    destroy()
```

コードの説明

1. gpiozero ライブラリから OutputDevice および LED クラスをインポートし、一般的な出力デバイスと特に LED を制御するための機能を有効にします。Python のスレッディングモジュールもインポートされており、スレッドを作成して処理するために使用されます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, LED
import threading
```

2. シフトレジスタのシリアルデータ入力（SDI）、レジスタクロック入力（RCLK）、およびシフトレジスタクロック入力（SRCLK）に接続された GPIO ピンを初期化します。

```
# 74HC595 シフトレジスタ用の GPIO ピンのセットアップ
SDI = OutputDevice(24)   # シリアルデータ入力
RCLK = OutputDevice(23)  # レジスタクロック
SRCLK = OutputDevice(18) # シフトレジスタクロック
```

3. 7 セグメントディスプレイの各ディジット用のピンを初期化し、0 から 9 までの数値を表示するためのバイナリコードを定義します。

```
# 7 セグメントディスプレイのディジット選択用の GPIO ピンのセットアップ
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメントディスプレイに 0 から 9 までの数字を表示するためのセグメントコード
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

4. 交通信号のシミュレーションに使用する赤、緑、黄色の LED の GPIO ピンを初期化します。交通信号システムの各色の状態の継続時間（秒単位）を設定します。参照のために、交通信号の色の名前も定義します。

```
# 交通信号 LED の GPIO ピンのセットアップ
ledPinR = LED(25) # 赤色の LED
ledPinG = LED(8)  # 緑色の LED
ledPinY = LED(7)  # 黄色の LED

# 交通信号の各色の継続時間設定
greenLight = 30  # 緑色
yellowLight = 5  # 黄色
redLight = 60    # 赤色

# 交通信号の色の名前
```

(次のページに続く)

(前のページからの続き)

```
lightColor = ("赤", "緑", "黄色")
```

5. 現在の色の状態、タイミングのためのカウンタ、およびタイマーオブジェクトのプレースホルダを初期化します。

```
# 状態変数の初期化
colorState = 0
counter = 60
timer1 = None
```

6. 交通信号システムを初期化し、タイマースレッドを開始します。

```
def setup():
    """ 交通信号システムを初期化し、タイマーを開始します。 """
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
```

7. 7 セグメントディスプレイの制御用の関数です。 clearDisplay はすべてのセグメントをオフにし、hc595_shift はデータをシフトレジスタにシフトし、 pickDigit はディスプレイ上の特定のディジットをアクティブにします。

```
def clearDisplay():
    """ 7セグメントディスプレイをクリアします。 """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ データを 74HC595 シフトレジスタにシフトしてディスプレイします。 """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

(次のページに続く)

(前のページからの続き)

```
def pickDigit(digit):
    """ 7セグメントディスプレイ上に表示する特定のディジットを選択します。 """
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

8. 交通信号の変更のタイミングを管理し、カウンターと色の状態を更新します。

```
def timer():
    """ 交通信号の変更のタイミングを処理します。 """
    global counter, colorState, timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter -= 1
    if counter == 0:
        counter = [greenLight, yellowLight, redLight][colorState]
        colorState = (colorState + 1) % 3
    print(f"カウンター : {counter}    色: {lightColor[colorState]}")
```

9. 現在の色の状態に基づいて交通信号の LED の状態を更新します。

```
def lightup():
    """ 現在の状態に基づいて交通信号の LED を更新します。 """
    global colorState
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
    [ledPinR, ledPinG, ledPinY][colorState].on()
```

10. 7 セグメントディスプレイの各セグメントに表示されるディジットを計算し、それに応じて更新します。

```
def display():
    """ 7セグメントディスプレイ上に現在のカウンター値を表示します。 """
    global counter

    for i in range(4):
        digit = counter // (10 ** (3 - i)) % 10
        if i == 0 and digit == 0:
            continue
        clearDisplay()
```

(次のページに続く)

(前のページからの続き)

```
        pickDigit(3 - i)
        hc595_shift(number[digit])
```

11. ディスプレイと交通信号の LED を連続的に更新するメインループです。

```
def loop():
    """ ディスプレイとライトを連続的に更新するメインループです。 """
    while True:
        display()
        lightup()
```

12. スクリプトが終了したときに LED をオフにし、タイマースレッドを停止するなど、リソースをクリーンアップします。

```
def destroy():
    """ スクリプトが終了したときにリソースをクリーンアップします。 """
    global timer1
    timer1.cancel()
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
```

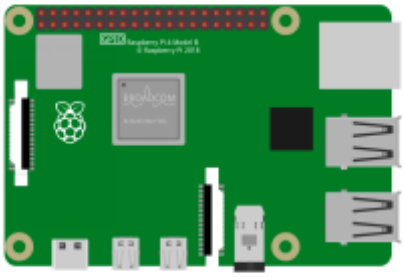
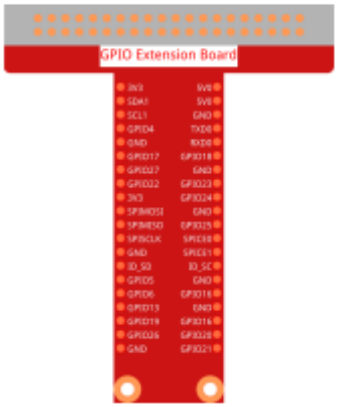
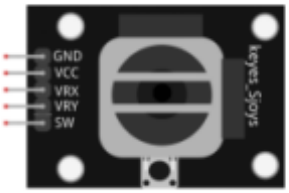
3.1.8 過熱モニター

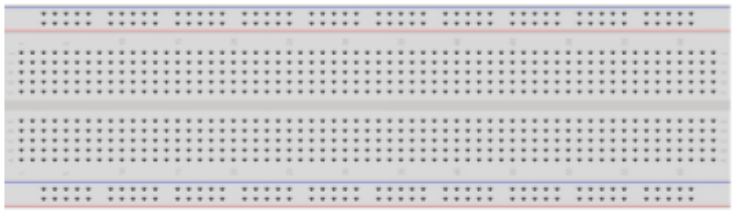
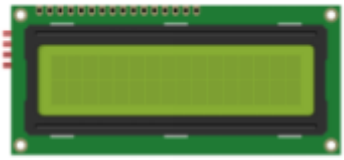
概要

あなたは過熱監視デバイスを作成したいかもしれません。例えば、工場内で回路の過熱がある場合にアラームと機械の自動シャットダウンをしたい場合です。このプロジェクトでは、サーミスタ、ジョイスティック、ブザー、LED、および 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デバイスを作成します。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

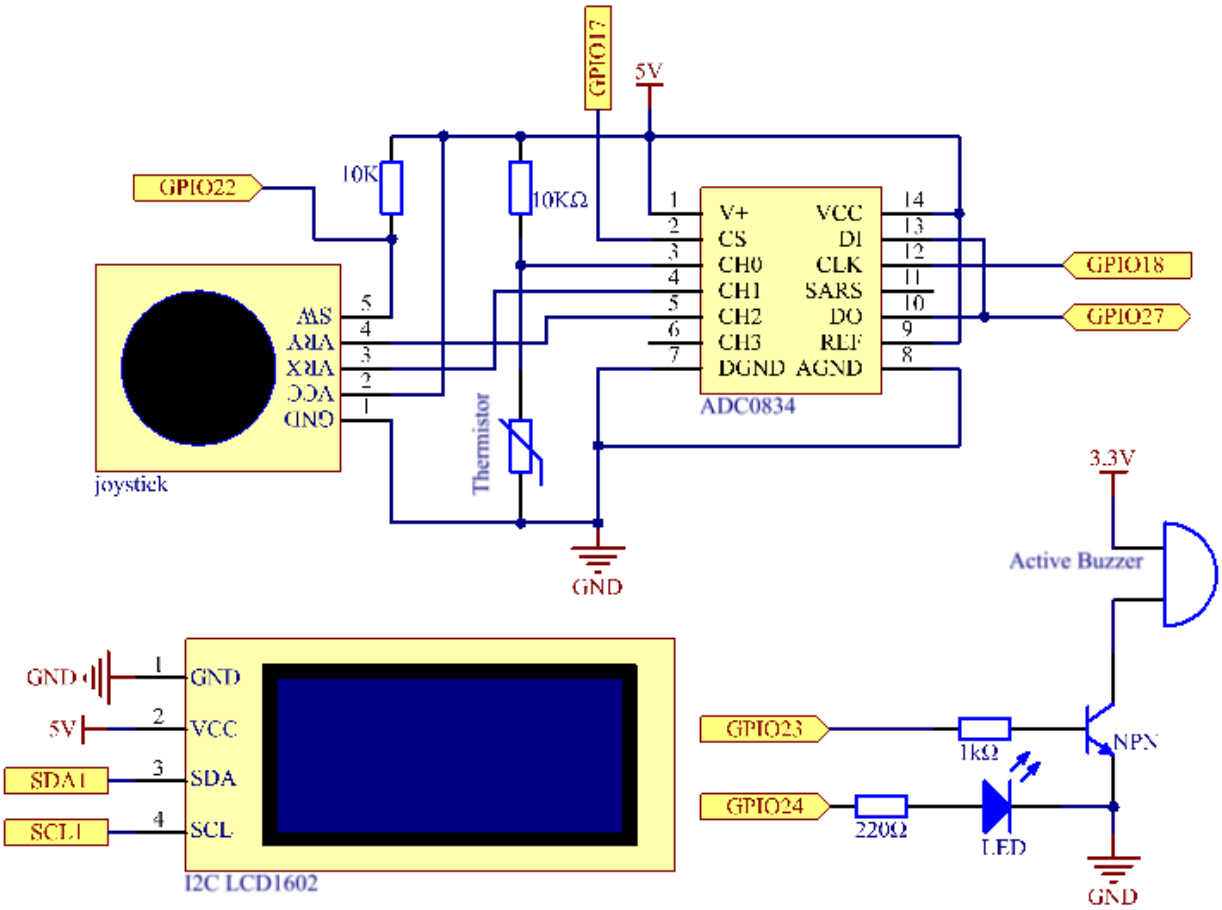
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Joystick
 
1 * ADC0834
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * LED

1* S8050 NPN Transistor
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * I2C LCD1602
  </td><td>1 * Active Buzzer
 </td><td>1*Thermistor
 </td><td>1 * Resistor 220Ω

1 * Resistor 1kΩ

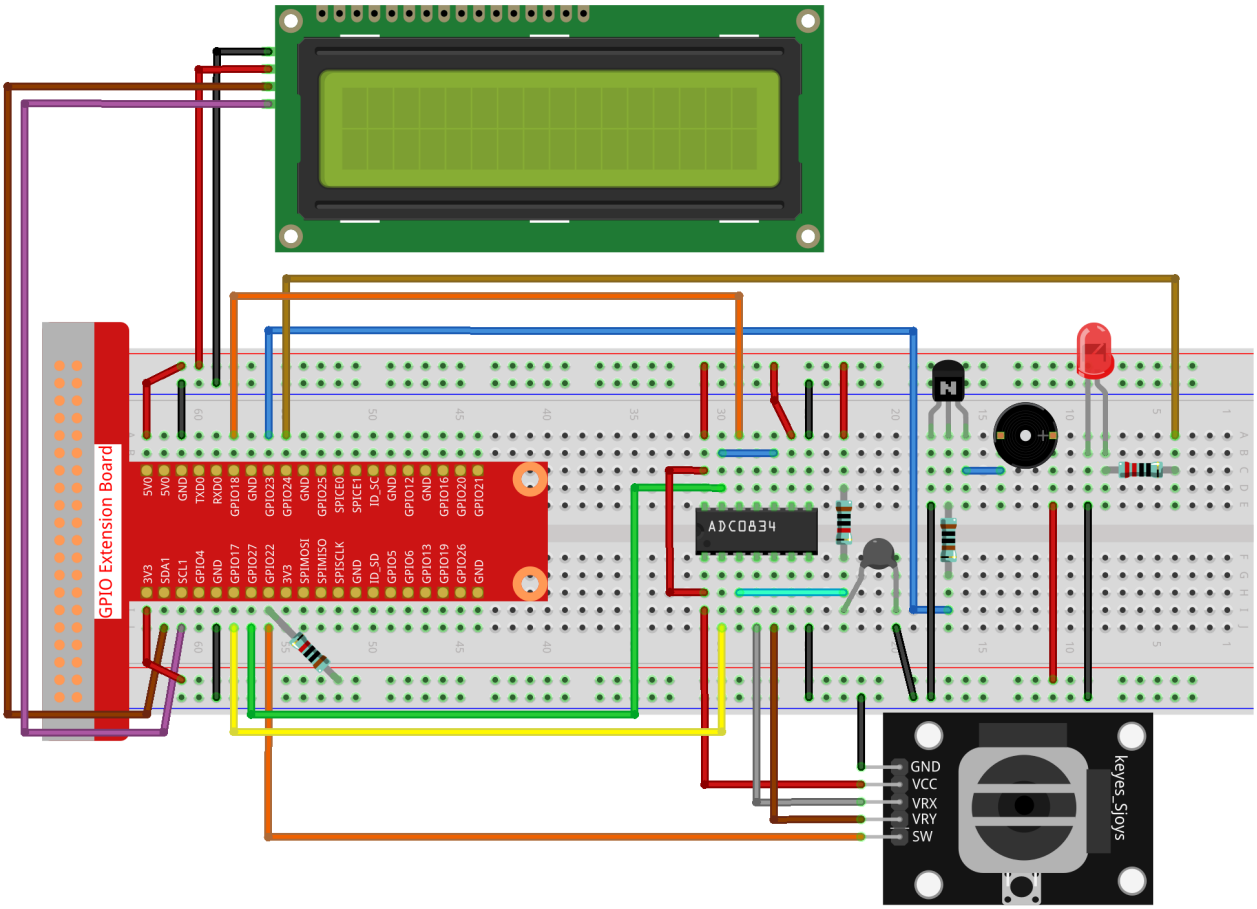
1 * Resistor 10kΩ
 </td></tr>
</table>

回路図



実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: コードのフォルダに移動します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行します。

```
sudo python3 3.1.8_OverheatMonitor_zero.py
```

コードが実行されると、現在の温度と高温閾値 **40** が **I2C LCD1602** に表示されます。現在の温度が閾値を超える場合、ブザーと LED がアラームを鳴らします。

ここでの **ジョイスティック** は、高温閾値を調整するためのボタンです。 **ジョイスティック** を X 軸および Y 軸の方向に切り替えると（上下に動かすと）現在の高温閾値を調整できます。 **ジョイスティック** をもう一度押すと、閾値が初期値にリセットされます。

注釈:

- エラーが発生した場合、 `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` 、I2C を有効にするために *I2C* **構成** を参照してください。

- ModuleNotFoundError: No module named 'smbus2' エラーが発生した場合は、sudo pip3 install smbus2 を実行してください。
- エラー OSError: [Errno 121] Remote I/O error が表示された場合、モジュールが誤って配線されているか、モジュールが壊れていることを意味します。
- コードと配線が正常であるにもかかわらず、LCD にコンテンツが表示されない場合、裏側のポテンショメーターを回してコントラストを調整できます。

コード

注釈: 以下のコードを **修正/リセット/コピー/実行/停止** できます。ただし、それより前に、 raphael-kit/python のようなソースコードのパスに移動する必要があります。コードを修正した後、効果を確認するために直接実行できます。

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

import LCD1602
from gpiozero import LED, Buzzer, Button
import ADC0834
import time
import math

# ジョイスティックボタン、ブザー、LED の初期化
Joy_BtnPin = Button(22)
buzzPin = Buzzer(23)
ledPin = LED(24)

# 初期の高温閾値を設定
upperTem = 40

# ADC と LCD モジュールのセットアップ
ADC0834.setup()
LCD1602.init(0x27, 1)

def get_joystick_value():
    """
    ジョイスティックの値を読み取り、ジョイスティックの位置に基づいて変更値を返します。
    """
```

(次のページに続く)

(前のページからの続き)

```
    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

def upper_tem_setting():
    """
    ジョイスティックの入力を使用して高温閾値を調整し、LCD に表示します。
    """
    global upperTem
    LCD1602.write(0, 0, ' 上限調整: ')
    change = int(get_joystick_value())
    upperTem += change
    strUpperTem = str(upperTem)
    LCD1602.write(0, 1, strUpperTem)
    LCD1602.write(len(strUpperTem), 1, '               ')
    time.sleep(0.1)

def temperature():
    """
    センサーから現在の温度を読み取り、摂氏で返します。
    """
    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
    return round(Cel, 2)

def monitoring_temp():
    """
```

(次のページに続く)

(前のページからの続き)

```
    現在の温度と上限温度閾値を監視および表示します。温度が上限を超える場合、ブザーと LED が作動します。

    """
    global upperTem
    Cel = temperature()
    LCD1602.write(0, 0, '温度: ')
    LCD1602.write(0, 1, '上限: ')
    LCD1602.write(6, 0, str(Cel))
    LCD1602.write(7, 1, str(upperTem))
    time.sleep(0.1)
    if Cel >= upperTem:
        buzzPin.on()
        ledPin.on()
    else:
        buzzPin.off()
        ledPin.off()

# メイン実行ループ
try:
    lastState = 1
    stage = 0
    while True:
        currentState = Joy_BtnPin.value
        # 設定モードと監視モードを切り替えます
        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()
except KeyboardInterrupt:
    # クリーンアップして終了
    LCD1602.clear()
    ADC0834.destroy()
```

代解

1. このセクションでは、プロジェクトに必要なライブラリをインポートします。 LCD1602 は LCD ディスプレイ用、 gpiozero は LED、ブザー、およびボタン用のクラスを提供します。 ADC0834 はアナログデジタル変換用で、 time および math は時間関連の関数と数学操作のための Python 標準ライブラリです。

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

import LCD1602
from gpiozero import LED, Buzzer, Button
import ADC0834
import time
import math
```

2. ここで、ジョイスティックのボタン、ブザー、LED を初期化します。 Button(22) は GPIO ピン 22 に接続されたボタンオブジェクトを作成します。 Buzzer(23) と LED(24) はそれぞれ GPIO ピン 23 と 24 にブザーと LED を初期化します。

```
# ジョイスティックボタン、ブザー、LED の初期化
Joy_BtnPin = Button(22)
buzzPin = Buzzer(23)
ledPin = LED(24)
```

3. 初期の上限温度を設定し、ADC と LCD モジュールを初期化します。LCD はアドレス（ 0x27 ）とモード（ 1 ）で初期化されます。

```
# 初期の上限温度閾値を設定
upperTem = 40

# ADC と LCD モジュールのセットアップ
ADC0834.setup()
LCD1602.init(0x27, 1)
```

4. この関数はジョイスティックの X と Y の値を ADC0834 を使用して読み取り、ジョイスティックの位置に基づいて変更値を返します。この変更値は温度の閾値を調整するために使用されます。

```
def get_joystick_value():
    """
    ジョイスティックの値を読み取り、ジョイスティックの位置に基づいて変更値を返します。
    """
    x_val = ADC0834.getResult(1)
    y_val = ADC0834.getResult(2)
    if x_val > 200:
```

(次のページに続く)

(前のページからの続き)

```
        return 1
    elif x_val < 50:
        return -1
    elif y_val > 200:
        return -10
    elif y_val < 50:
        return 10
    else:
        return 0
```

5. ジョイスティックの入力を使用して上限温度を調整し、新しい閾値を LCD に表示します。

```
def upper_tem_setting():
    """
    ジョイスティックの入力を使用して上限温度閾値を調整し、LCD に表示します。
    """
    global upperTem
    LCD1602.write(0, 0, ' 上限調整: ')
    change = int(get_joystick_value())
    upperTem += change
    strUpperTem = str(upperTem)
    LCD1602.write(0, 1, strUpperTem)
    LCD1602.write(len(strUpperTem), 1, '               ')
    time.sleep(0.1)
```

6. センサーから現在の温度を読み取り、摂氏に変換します。

```
def temperature():
    """
    センサーから現在の温度を読み取り、摂氏で返します。
    """
    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
    return round(Cel, 2)
```

7. 現在の温度と上限温度閾値を監視し、LCD に表示します。温度が上限を超える場合、ブザーと LED が作動します。

```
def monitoring_temp():
    """
    現在の温度と上限温度閾値を監視し、LCD に表示します。温度が上限を超える場合、ブザーと LED が
作動します。
    """
    global upperTem
    Cel = temperature()
    LCD1602.write(0, 0, '温度: ')
    LCD1602.write(0, 1, '上限: ')
    LCD1602.write(6, 0, str(Cel))
    LCD1602.write(7, 1, str(upperTem))
    time.sleep(0.1)
    if Cel >= upperTem:
        buzzPin.on()
        ledPin.on()
    else:
        buzzPin.off()
        ledPin.off()
```

8. メイン実行ループはジョイスティックボタンの押下に基づいて設定モードと監視モードを切り替えます。設定または温度の監視を継続的に更新します。

```
# メイン実行ループ
try:
    lastState = 1
    stage = 0
    while True:
        currentState = Joy_BtnPin.value
        # 設定モードと監視モードを切り替える
        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()
except KeyboardInterrupt:
    # クリーンアップして終了
```

(次のページに続く)

(前のページからの続き)

```
    LCD1602.clear()
    ADC0834.destroy()
```

3.1.9 パスワードロック

はじめに

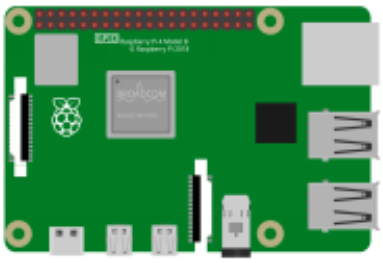
このプロジェクトでは、キーパッドと LCD を使用して組み合わせ錠を作成します。LCD には、キーパッドにパスワードを入力するための対応するプロンプトが表示されます。パスワードが正しく入力されると、「正しい」と表示されます。

このプロジェクトを基に、ブザーや LED などの追加の電子コンポーネントを追加して、パスワードの入力に異なる実験現象を追加することができます。

必要な部品

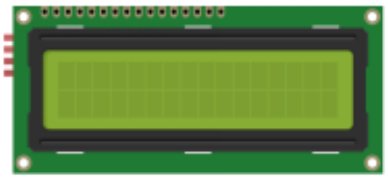
このプロジェクトでは、次の部品が必要です。

1 * Raspberry Pi



1 * T-Extension Board

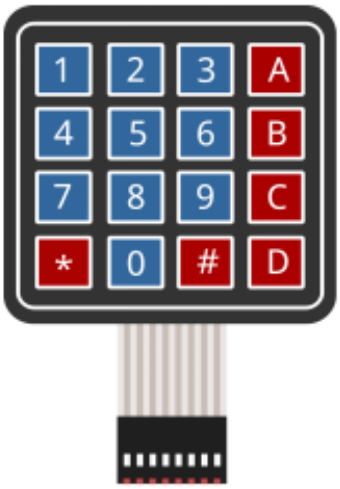
1 * I2C LCD1602



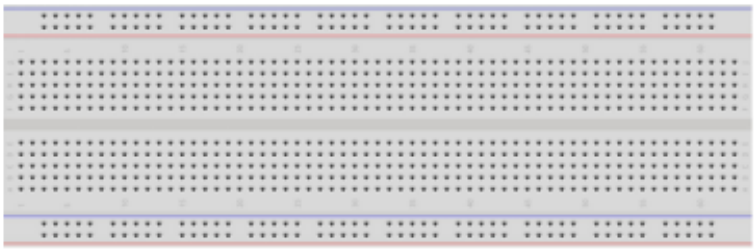
Several Jumper Wires

1 * 40-pin Cable

1 * Keypad



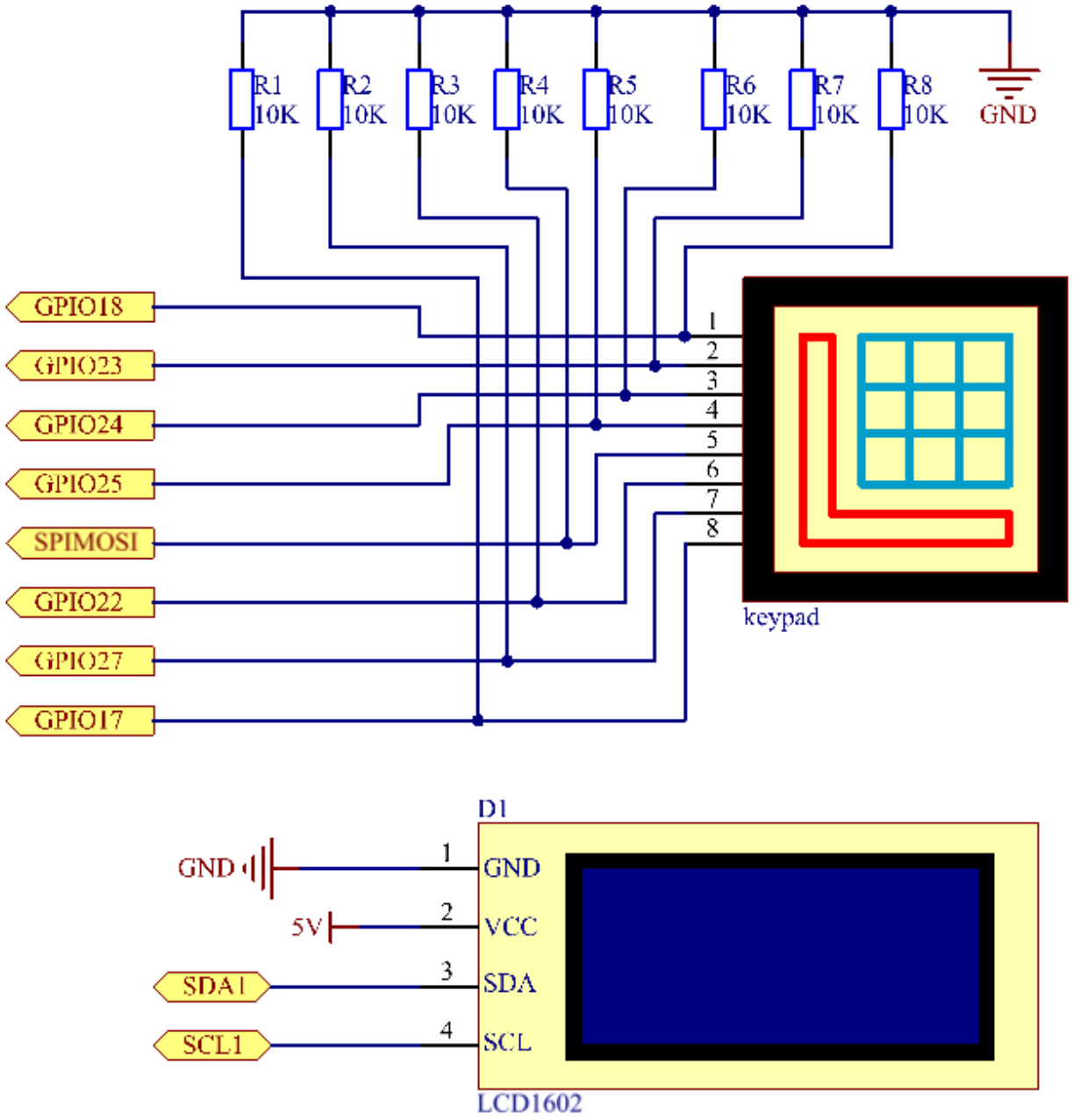
1 * Breadboard



8 * Resistor 10KΩ

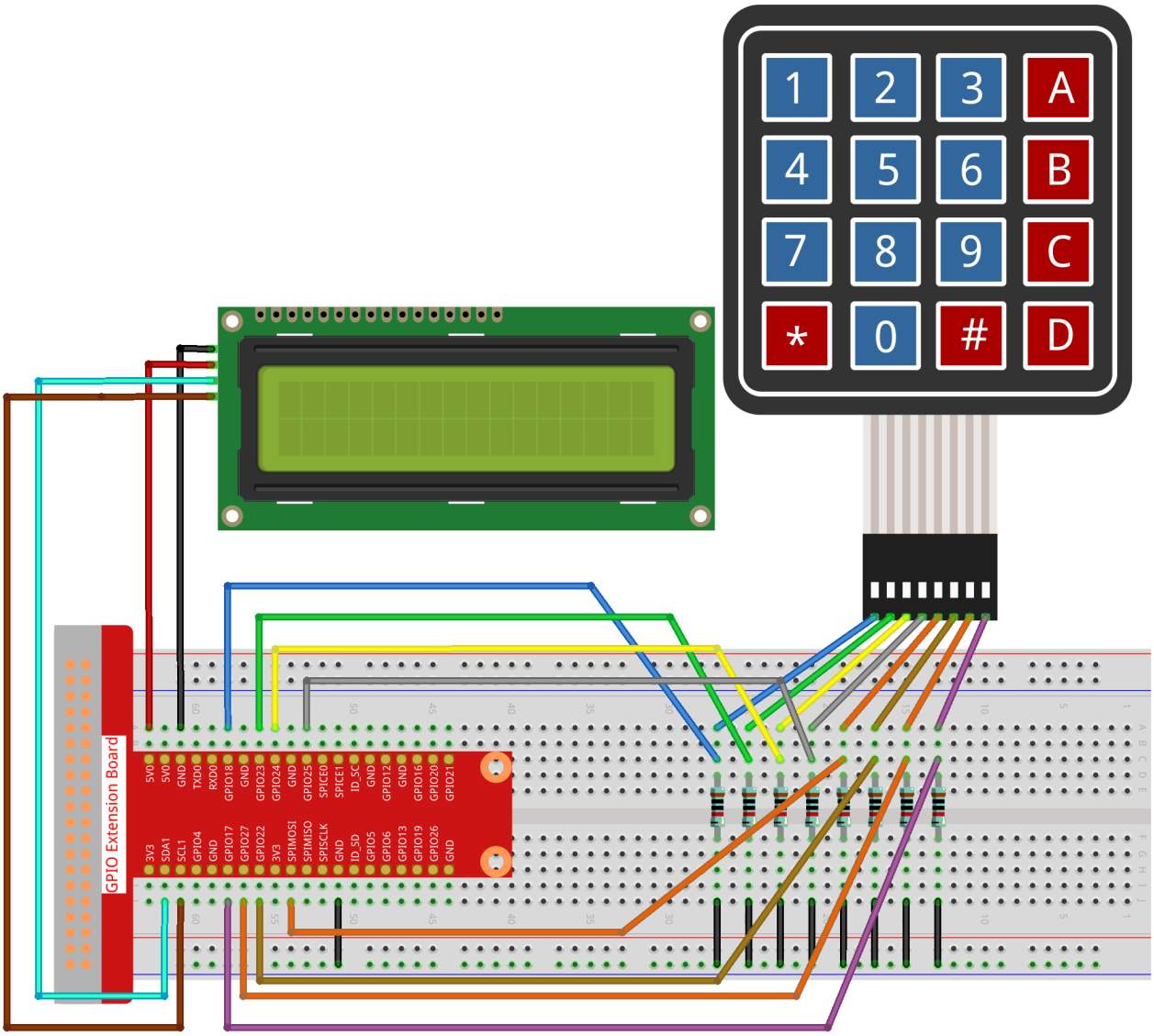
回路図

T-Board Name	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
SPIMOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		



実験手順

ステップ 1： 回路を組み立てます。



ステップ 2： ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3： 実行します。

```
sudo python3 3.1.9_PasswordLock_zero.py
```

コードが実行されると、キーパッドを使用してパスワード（1984）を入力します。LCD1602 に「CORRECT」が表示された場合、パスワードは正しいです。それ以外の場合、「WRONG KEY」が表示されます。

注釈:

- 「FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'」というエラーが表示される場合は、I2C

を有効にするために *I2C* **構成** を参照してください。

- 「ModuleNotFoundError: No module named 'smbus2'」エラーが表示される場合は、「sudo pip3 install smbus2」を実行してください。
- エラー「OSError: [Errno 121] Remote I/O error」が表示される場合、モジュールが配線されていないか、モジュールが壊れていることを意味します。
- コードと配線が正常であるが、LCD に内容が表示されない場合、背面のポテンショメータを回してコントラストを上げることができます。

コード

注釈: 以下のコードを **変更/リセット/コピー/実行/停止** することができます。ただし、それを行う前に raphael-kit/python-pi5 のようなソースコードパスに移動する必要があります。コードを変更した後、それを直接実行して効果を確認できます。

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

from gpiozero import DigitalOutputDevice, Button
from time import sleep
import LCD1602

class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        指定された行と列のピンおよびキーでキーパッドを初期化します。
        :param rows_pins: 行の GPIO ピンのリスト。
        :param cols_pins: 列の GPIO ピンのリスト。
        :param keys: キーパッドのレイアウト内のキーのリスト。
        """
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # 行ピンの設定
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # 列ピンの設定
        self.keys = keys  # キーパッドのキーレイアウト

    def read(self):
        """
        現在押されているキーのリストを読み取り、返します。
        :return: 押されているキーのリスト。
        """
```

(次のページに続く)

(前のページからの続き)

```
        pressed_keys = []
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行をアクティブにする
            for j, col in enumerate(self.cols):
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # チェック後に行を非アクティブにする
        return pressed_keys

# パスワードの検証設定
LENS = 4
password = ['1', '9', '8', '4']  # プリセットのパスワード
testword = ['0', '0', '0', '0']  # ユーザーの入力を格納
keyIndex = 0  # 入力キーのインデックス

def check():
    """
    入力されたパスワードがプリセットのパスワードと一致するか確認します。
    :return: 一致する場合は 1、それ以外の場合は 0。
    """
    for i in range(LENS):
        if password[i] != testword[i]:
            return 0
    return 1

def setup():
    """
    キーパッドと LCD ディスプレイのセットアップを行います。
    """
    global keypad, last_key_pressed
    # キーパッドのピン設定
    rows_pins = [18, 23, 24, 25]
    cols_pins = [10, 22, 27, 17]
    keys = ["1", "2", "3", "A",
            "4", "5", "6", "B",
            "7", "8", "9", "C",
            "*", "0", "#", "D"]
```

(次のページに続く)

(前のページからの続き)

```
    # キーパッドと LCD の初期化
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []
    LCD1602.init(0x27, 1)  # LCD の初期化
    LCD1602.clear()
    LCD1602.write(0, 0, 'WELCOME!')
    LCD1602.write(2, 1, ' パスワードを入力してください')
    sleep(2)

def loop():
    """
    キーパッドの入力とパスワードの検証を処理するメインループです。
    """
    global keyIndex, LENS, keypad, last_key_pressed
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            if keyIndex < LENS:
                LCD1602.clear()
                LCD1602.write(0, 0, "パスワードを入力:")
                LCD1602.write(15 - keyIndex, 1, pressed_keys[0])
                testword[keyIndex] = pressed_keys[0]
                keyIndex += 1

            if keyIndex == LENS:
                if check() == 0:
                    LCD1602.clear()
                    LCD1602.write(3, 0, "WRONG KEY!")
                    LCD1602.write(0, 1, "もう一度お試しください")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "CORRECT!")
                    LCD1602.write(2, 1, "お帰りなさい")
                keyIndex = 0  # チェック後にキーインデックスをリセット

        last_key_pressed = pressed_keys
        sleep(0.1)

try:
```

(次のページに続く)

(前のページからの続き)

```
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # インタラクト時に LCD 表示をクリア
```

コードの説明

1. スクリプトは gpiozero ライブラリからデジタル出力デバイスとボタンの管理クラスをインポートします。また、time モジュールから sleep 関数をインポートし、スクリプトの実行中に遅延を追加します。さらに、LCD1602 ライブラリもインポートして、LCD1602 ディスプレイを制御します。

```
#!/usr/bin/env python3
from gpiozero import DigitalOutputDevice, Button
from time import sleep
import LCD1602
```

2. キーパッドを管理するためのカスタムクラスを定義します。指定された行と列のピンでキーパッドを初期化し、押されたキーを検出する read メソッドを提供します。

```
class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        指定された行と列のピンおよびキーでキーパッドを初期化します。
        :param rows_pins: 行の GPIO ピンのリスト。
        :param cols_pins: 列の GPIO ピンのリスト。
        :param keys: キーパッドのレイアウト内のキーのリスト。
        """
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # 行ピンの設定
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # 列ピンの設定
        self.keys = keys  # キーパッドのキーレイアウト

    def read(self):
        """
        現在押されているキーのリストを読み取り、返します。
        :return: 押されているキーのリスト。
        """
        pressed_keys = []
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行をアクティブにする
            for j, col in enumerate(self.cols):
```

(次のページに続く)

(前のページからの続き)

```
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # チェック後に行を非アクティブにする
        return pressed_keys
```

3. パスワードの検証システムをセットアップします。 LENS はパスワードの長さを定義します。 password はプリセットの正しいパスワードで、testword はユーザーの入力を保存するために使用されます。 keyIndex はユーザーの入力の現在位置を追跡します。

```
# パスワードの検証設定
LENS = 4
password = ['1', '9', '8', '4']  # プリセットのパスワード
testword = ['0', '0', '0', '0']  # ユーザーの入力を格納
keyIndex = 0  # 入力キーのインデックス
```

4. 入力されたパスワード（ testword ）とプリセットのパスワード（ password ）を比較し、結果を返す関数。

```
def check():
    """
    入力されたパスワードがプリセットのパスワードと一致するか確認します。
    :return: 一致する場合は 1、それ以外の場合は 0。
    """
    for i in range(LENS):
        if password[i] != testword[i]:
            return 0
    return 1
```

5. キーパッドと LCD ディスプレイの初期化を行います。歓迎メッセージとパスワード入力の指示を表示します。

```
def setup():
    """
    キーパッドと LCD ディスプレイのセットアップを行います。
    """
    global keypad, last_key_pressed
    # キーパッドのピン設定
    rows_pins = [18, 23, 24, 25]
    cols_pins = [10, 22, 27, 17]
    keys = ["1", "2", "3", "A",
```

(次のページに続く)

(前のページからの続き)

```
            "4", "5", "6", "B",
            "7", "8", "9", "C",
            "*", "0", "#", "D"]

    # キーパッドと LCD の初期化
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []
    LCD1602.init(0x27, 1)  # LCD の初期化
    LCD1602.clear()
    LCD1602.write(0, 0, 'WELCOME!')
    LCD1602.write(2, 1, ' パスワードを入力してください')
    sleep(2)
```

6. キーパッド入力とパスワードの検証を処理するメインループです。入力されたパスワードに基づいて LCD ディスプレイを更新し、パスワードが正しいかどうかに応じてフィードバックを提供します。

```
def loop():
    """
    キーパッドの入力とパスワードの検証を処理するメインループです。
    """
    global keyIndex, LENS, keypad, last_key_pressed
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            if keyIndex < LENS:
                LCD1602.clear()
                LCD1602.write(0, 0, "パスワードを入力:")
                LCD1602.write(15 - keyIndex, 1, pressed_keys[0])
                testword[keyIndex] = pressed_keys[0]
                keyIndex += 1

            if keyIndex == LENS:
                if check() == 0:
                    LCD1602.clear()
                    LCD1602.write(3, 0, "WRONG KEY!")
                    LCD1602.write(0, 1, "もう一度お試しください")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "CORRECT!")
```

(次のページに続く)

(前のページからの続き)

```
                LCD1602.write(2, 1, "お帰りなさい")
            keyIndex = 0  # チェック後にキーインデックスをリセット

        last_key_pressed = pressed_keys
        sleep(0.1)
```

7. セットアップを実行し、メインループに入ります。キーボード割り込み（Ctrl+C）を使用してプログラムをクリーンに終了させることができます。LCD 表示をクリアします。

```
try:
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # インタラクト時に LCD 表示をクリア
```

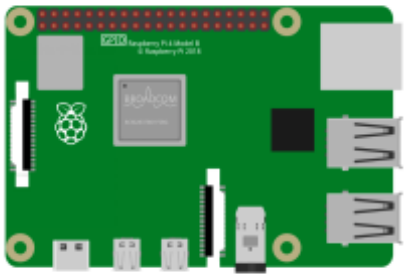
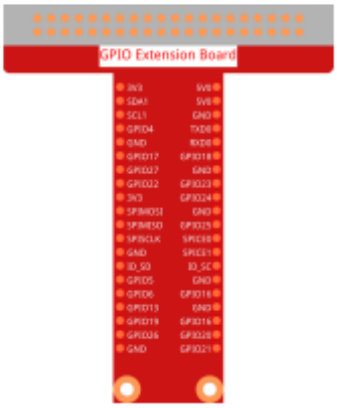
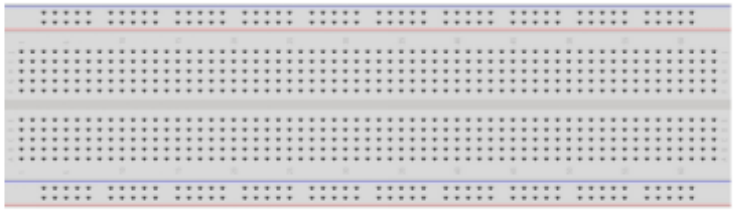
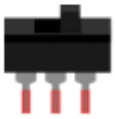
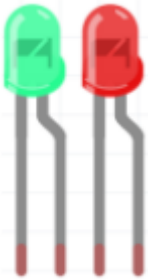
3.1.10 アラームベル

概要

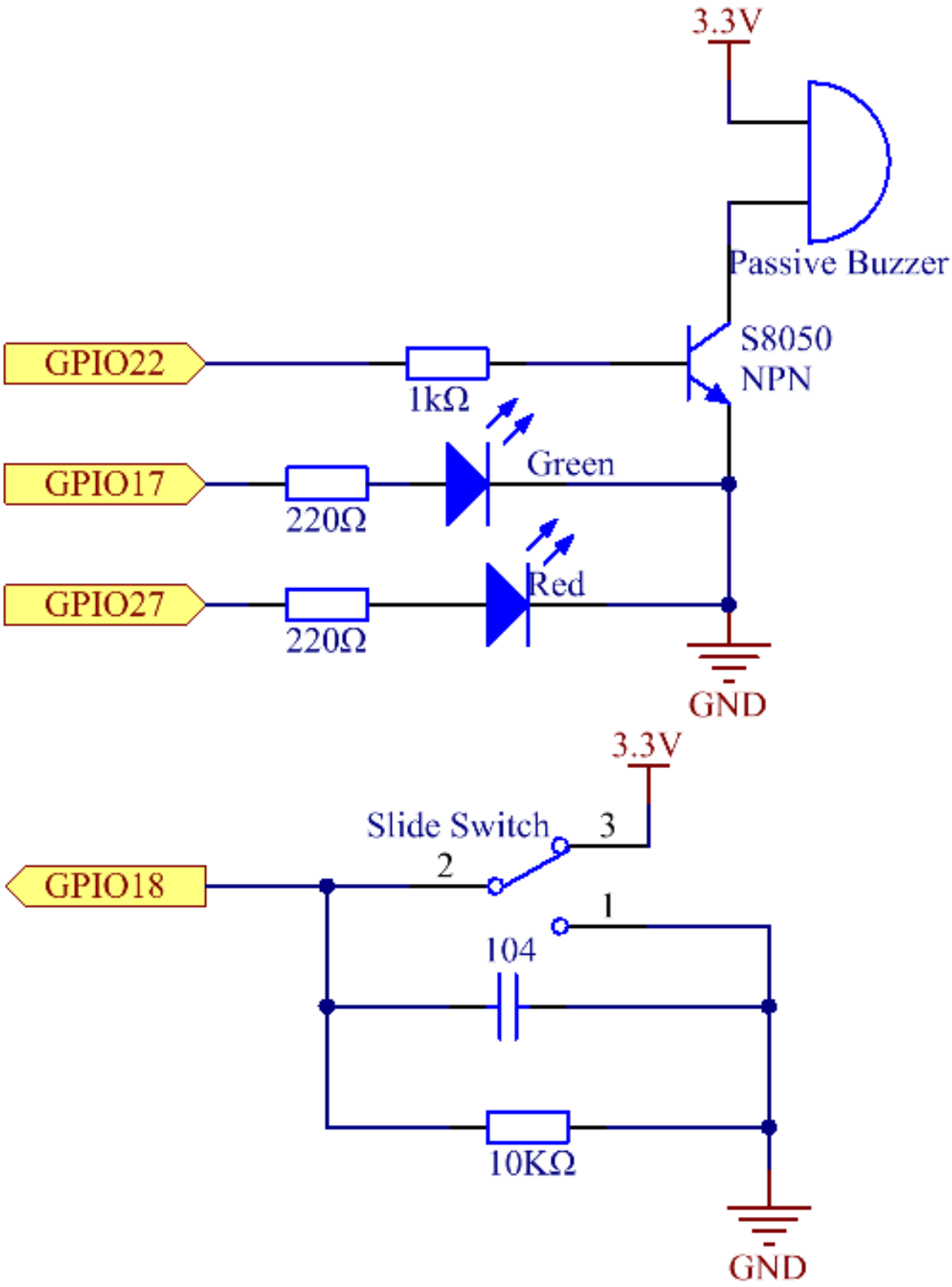
このプロジェクトでは、手動アラームデバイスを作成します。トグルスイッチをサーミスタまたは光感知センサーに置き換えて、温度アラームまたは光アラームを作成できます。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

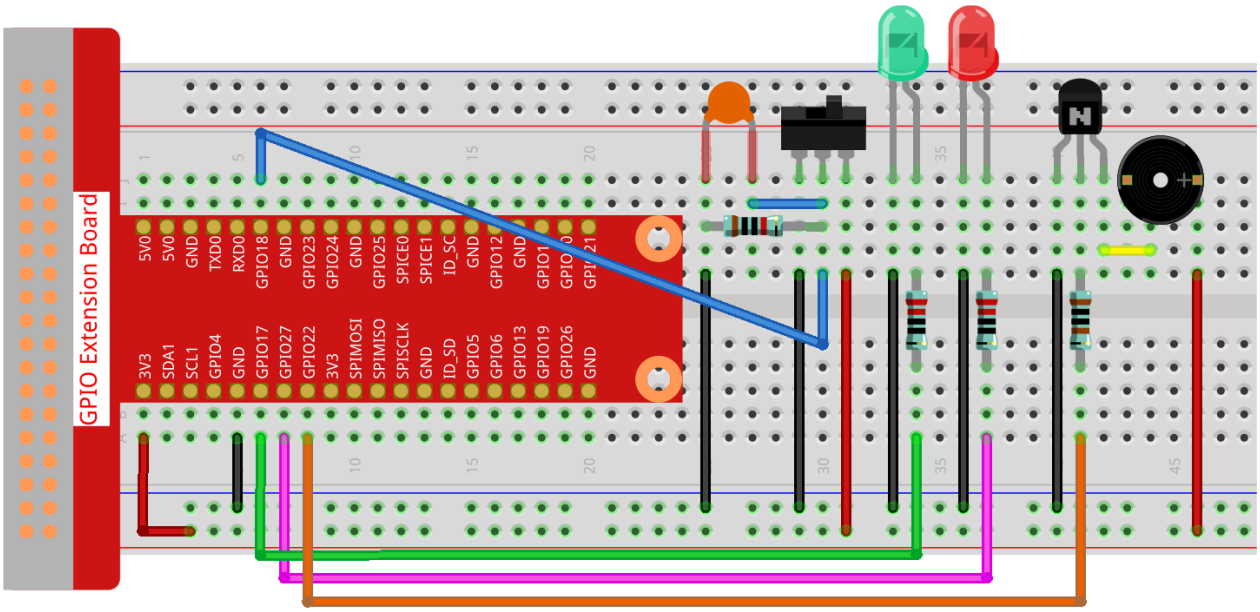
1 * Raspberry Pi 	1 * T-Extension Board 	1 * Passive Buzzer
		1 * Resistor(1kΩ)
1 * 40-pin Cable		2 * Resistor(220Ω)
		1 * Resistor(10kΩ)
1 * Breadboard 		1 * Slide Switch 
Several Jumper Wires	2 * LED 	1 * S8050 NPN Transistor
		1 * 104 Capacitor

回路図


3.3V
Passive Buzzer
S8050
NPN
GPIO22
1kΩ
GPIO17
220Ω
Green
GPIO27
220Ω
Red
GND
3.3V
Slide Switch
3
2
GPIO18
1
104
10KΩ
GND

実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行します。

```
sudo python3 3.1.10_AlarmBell_zero.py
```

プログラムを起動した後、トグルスイッチを右に切り替えると、ブザーからアラーム音が鳴ります。同時に、赤と緑の LED が一定の頻度で点滅します。

コード

注釈: 以下のコードを **修正/リセット/コピー/実行/停止** することができます。ただし、その前に `raphael-kit/python-pi5` のソースコードパスに移動する必要があります。

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

from gpiozero import LED, Button, TonalBuzzer
import time
```

(次のページに続く)

(前のページからの続き)

```
import threading

# GPIO ピン 22 に TonalBuzzer を初期化
BeepPin = TonalBuzzer(22)

# GPIO ピン 17 と 27 に LED を初期化
ALedPin = LED(17)
BLedPin = LED(27)

# GPIO ピン 18 にボタンを初期化
switchPin = Button(18)

# ブザーと LED の状態を管理するためのグローバルフラグ
flag = 0

def ledWork():
    """
    フラグの状態に基づいて LED の点滅パターンを制御します。
    フラグがセットされている場合、ALedPin と BLedPin を交互に点滅させます。
    フラグがセットされていない場合、両方の LED をオフにします。
    """
    while True:
        if flag:
            # LED を交互に点滅
            ALedPin.on()
            time.sleep(0.5)
            ALedPin.off()
            BLedPin.on()
            time.sleep(0.5)
            BLedPin.off()
        else:
            # フラグがセットされていない場合、両方の LED をオフにする
            ALedPin.off()
            BLedPin.off()

# 音楽の音符とその持続時間のリストとしてチューンを定義します
tune = [
    ('C4', 0.1), ('E4', 0.1), ('G4', 0.1),
    (None, 0.1),
```

(次のページに続く)

(前のページからの続き)

```
    ('E4', 0.1), ('G4', 0.1), ('C5', 0.1),
    (None, 0.1),
    ('C5', 0.1), ('G4', 0.1), ('E4', 0.1),
    (None, 0.1),
    ('G4', 0.1), ('E4', 0.1), ('C4', 0.1),
    (None, 0.1)
]

def buzzerWork():
    """
    フラグの状態に基づいてブザーを使って曲を演奏します。
    フラグがセットされている場合のみ曲が演奏されます。
    曲が演奏中にフラグが解除されると、曲が停止します。
    """
    while True:
        for note, duration in tune:
            if flag == 0:
                break
            print(note)  # 現在の音符をコンソールに出力
            BeepPin.play(note)  # 現在の音符を演奏
            time.sleep(duration)  # 音符の持続時間だけ一時停止
        BeepPin.stop()  # 曲の演奏後にブザーを停止

def main():
    """
    ボタンの押下を監視してフラグの状態を更新します。
    ボタンが押されるとフラグがセットされます。
    """
    global flag
    while True:
        flag = 1 if switchPin.is_pressed else 0

try:
    # ブザーと LED 制御のスレッドを初期化して開始します
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
    main()
```

(次のページに続く)

(前のページからの続き)

```
except KeyboardInterrupt:
    # プログラムが中断されたときにブザーを停止し、LED をオフにします
    BeepPin.stop()
    ALedPin.off()
    BLedPin.off()
```

コードの説明

1. このセグメントでは、遅延とスレッド処理の実装に必要なライブラリのインポートを行います。また、Raspberry Pi 上の GPIO デバイスを制御するために、gpiozero ライブラリから LED、Button、および TonalBuzzer クラスもインポートしています。

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

from gpiozero import LED, Button, TonalBuzzer
import time
import threading
```

2. GPIO ピン 22 にブザー、GPIO ピン 17 と 27 に LED、GPIO ピン 18 にボタンをセットアップします。ブザーと LED の状態を管理するために、グローバルフラグも定義されています。

```
# GPIO ピン 22 に TonalBuzzer を初期化
BeepPin = TonalBuzzer(22)

# GPIO ピン 17 と 27 に LED を初期化
ALedPin = LED(17)
BLedPin = LED(27)

# GPIO ピン 18 にボタンを初期化
switchPin = Button(18)

# ブザーと LED の状態を管理するためのグローバルフラグ
flag = 0
```

3. この関数は、フラグの状態に応じて LED の点滅を制御します。フラグが設定されている（1）場合、各 LED をオンとオフに交互に切り替えます。フラグが設定されていない（0）場合、両方の LED をオフにします。

```
def ledWork():
    """
```

(次のページに続く)

(前のページからの続き)

```
    フラグの状態に基づいて LED の点滅パターンを制御します。
    フラグがセットされている場合、ALedPin と BLedPin を交互に点滅させます。
    フラグがセットされていない場合、両方の LED をオフにします。
    """
    while True:
        if flag:
            # LED を交互に点滅
            ALedPin.on()
            time.sleep(0.5)
            ALedPin.off()
            BLedPin.on()
            time.sleep(0.5)
            BLedPin.off()
        else:
            # フラグがセットされていない場合、両方の LED をオフにする
            ALedPin.off()
            BLedPin.off()
```

4. 音楽の音符（周波数）と持続時間（秒）のシーケンスである「tune」が定義されています。

```
# 音楽の音符とその持続時間のリストとしてチューンを定義
tune = [
    ('C4', 0.1), ('E4', 0.1), ('G4', 0.1),
    (None, 0.1),
    ('E4', 0.1), ('G4', 0.1), ('C5', 0.1),
    (None, 0.1),
    ('C5', 0.1), ('G4', 0.1), ('E4', 0.1),
    (None, 0.1),
    ('G4', 0.1), ('E4', 0.1), ('C4', 0.1),
    (None, 0.1)
]
```

5. フラグが設定されている場合に予め定義されたメロディを演奏します。演奏中にフラグが解除されると、演奏が停止します。

```
def buzzerWork():
    """
    フラグの状態に基づいてブザーを使って曲を演奏します。
    フラグがセットされている場合のみ曲が演奏されます。
```

(次のページに続く)

(前のページからの続き)

```
    曲が演奏中にフラグが解除されると、曲が停止します。
    """
    while True:
        for note, duration in tune:
            if flag == 0:
                break
            print(note)  # 現在の音符をコンソールに出力
            BeepPin.play(note)  # 現在の音符を演奏
            time.sleep(duration)  # 音符の持続時間だけ一時停止
        BeepPin.stop()  # 曲の演奏後にブザーを停止
```

6. ボタンの状態を確認し、フラグを設定または解除します。

```
def main():
    """
    ボタンの押下を監視してフラグの状態を更新します。
    ボタンが押されるとフラグがセットされます。
    """
    global flag
    while True:
        flag = 1 if switchPin.is_pressed else 0
```

7. buzzerWork と ledWork のスレッドを開始し、それらをメイン関数と同時に実行できるようにします。

```
try:
    # ブザーと LED 制御のスレッドを初期化して開始します
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
    main()
```

8. プログラムが中断されたときに、きれいに終了するように、ブザーを停止し、LED をオフにします。

```
except KeyboardInterrupt:
    # プログラムが中断されたときにブザーを停止し、LED をオフにします
    BeepPin.stop()
    ALedPin.off()
    BLedPin.off()
```

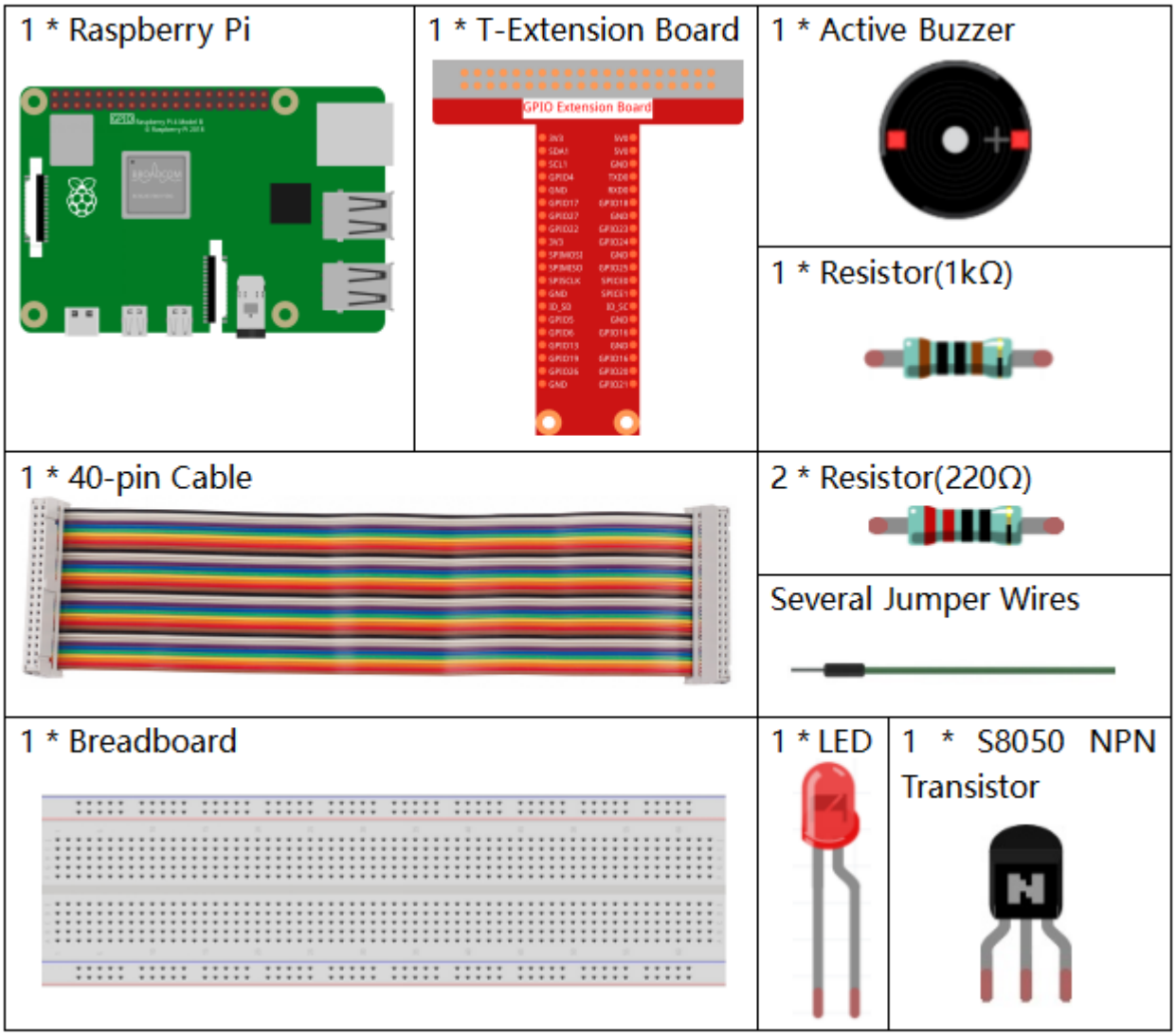
3.1.11 モールス信号ジェネレータ

はじめに

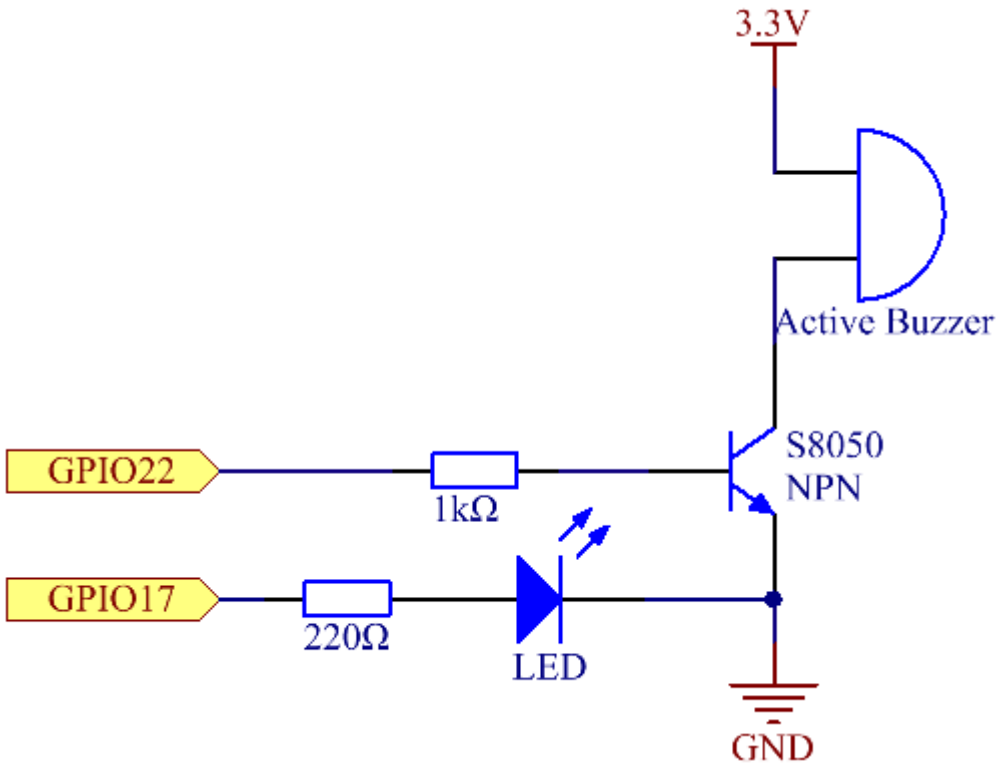
このプロジェクトでは、Raspberry Pi に英字のシリーズを入力し、それをモールス信号として表示するモールス信号ジェネレータを作成します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

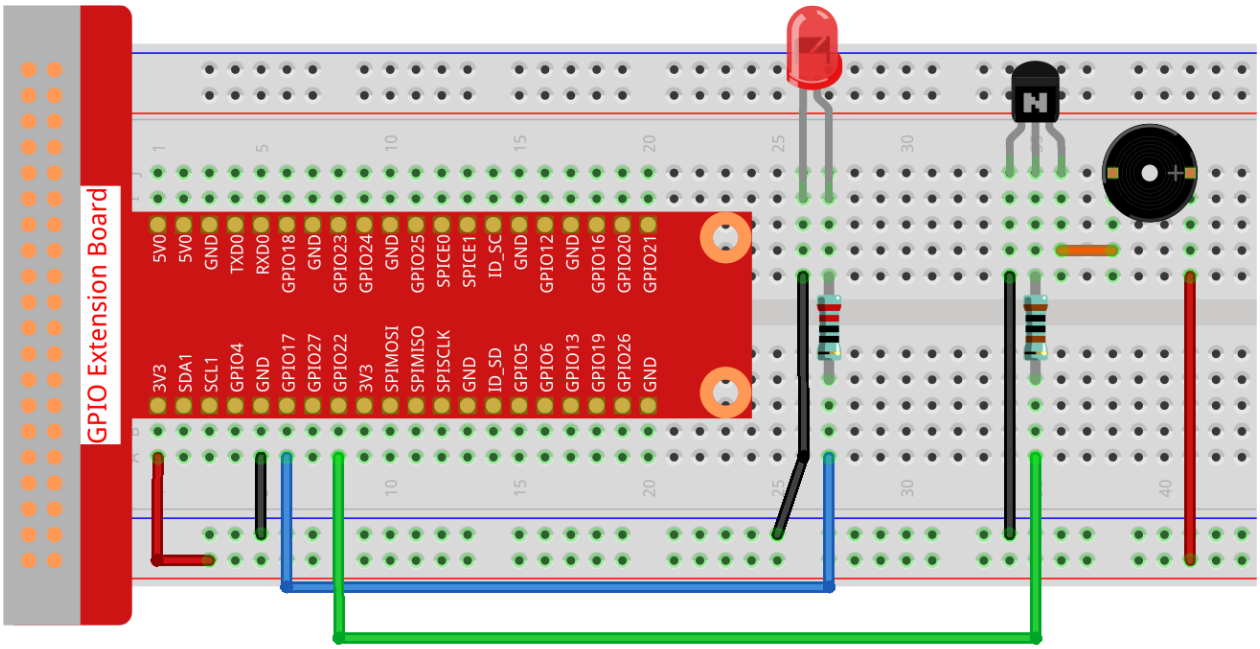


回路図



実験手順

ステップ 1： 回路を組み立てます。（ブザーの極性に注意してください：+ ラベルのある方が正極で、もう一方が負極です。）



ステップ 2： コードファイルを開きます。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3： 実行します。

```
sudo python3 3.1.11_MorseCodeGenerator_zero.py
```

プログラムが実行されたら、一連の文字列を入力し、ブザーと LED が対応するモールス信号を送信します。

コード

```
#!/usr/bin/env python3
from gpiozero import Buzzer, LED
import time

# ブザーと LED を GPIO ピンに初期化
BeepPin = Buzzer(22)
ALedPin = LED(17)

# 文字のモールス信号表現
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',
```

(次のページに続く)

(前のページからの続き)

```
    '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',
}

def on():
    """ ブザーと LED をオンにします。"""
    BeepPin.on()
    ALedPin.on()

def off():
    """ ブザーと LED をオフにします。"""
    BeepPin.off()
    ALedPin.off()

def beep(dt):  # dt は遅延時間用です。
    """
    指定された時間のブザー音と LED フラッシュを生成します。
    :param dt: ブザー音とフラッシュのための期間。
    """
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)

def morsecode(code):
    """
    入力されたコードをモールス信号に変換し、ブザーと LED を使用して信号化します。
    :param code: モールス信号に変換するテキスト。
    """
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause / 2)  # 短いビープ音
            if tap == '1':
                beep(pause)      # 長いビープ音
```

(次のページに続く)

(前のページからの続き)

```
        time.sleep(pause)  # 文字間の休憩

def destroy():
    """ スクリプトの終了時にリソースをクリーンアップします。"""
    print("")
    BeepPin.off()
    ALedPin.off()

try:
    while True:
        code = input("メッセンジャーを入力してください：")
        code = code.upper()  # モールス信号検索用に大文字に変換
        print(code)
        morsecode(code)
except KeyboardInterrupt:
    destroy()
```

コードの説明

1. このコードは、gpiozero ライブラリから Buzzer および LED クラスをインポートしています。これらのクラスは、Raspberry Pi 上の対応する GPIO デバイスを制御するために必要です。

```
#!/usr/bin/env python3
from gpiozero import Buzzer, LED
import time
```

2. GPIO ピン 22 のブザーと GPIO ピン 17 の LED を初期化して、これらのコンポーネントを制御するのに役立てます。

```
# ブザーと LED を GPIO ピンに初期化
BeepPin = Buzzer(22)
ALedPin = LED(17)
```

3. MORSECODE 構造を定義します。これは、A から Z までの文字、0 から 9 までの数字、および「?」、「/」、「:」、「,」、「.」、「;」、「!」、「@」などの記号のモールス信号表現を含む辞書です。ここで、 0 はドットを示し、 1 はダッシュを示します。

```
# 文字のモールス信号表現
MORSECODE = {
    'A': '01', 'B': '1000', 'C': '1010', 'D': '100', 'E': '0', 'F': '0010', 'G':
```

(次のページに続く)

(前のページからの続き)

```
↪'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',
}
```

4. on() 関数はブザーと LED をオンにします。 off() 関数はブザーと LED をオフにします。

```
def on():
    """ ブザーと LED をオンにします。"""
    BeepPin.on()
    ALedPin.on()

def off():
    """ ブザーと LED をオフにします。"""
    BeepPin.off()
    ALedPin.off()
```

5. beep() 関数は、ブザーと LED を指定された時間間隔で音を鳴らし、点滅させるために使用されます。

```
def beep(dt):   # dt は遅延時間用です。
    """
    指定された時間のブザー音と LED フラッシュを生成します。
    :param dt: ブザー音とフラッシュのための期間。
    """
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

6. morsecode() 関数は、入力された文字のモールス信号を処理し、ブザーと LED を使用してそれを示します。

```
def morsecode(code):
    """
    入力されたコードをモールス信号に変換し、ブザーと LED を使用して信号化します。
```

(次のページに続く)

(前のページからの続き)

```
    :param code: モールス信号に変換するテキスト。
    """
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause / 2)  # 短いビープ音
            if tap == '1':
                beep(pause)      # 長いビープ音
        time.sleep(pause)  # 文字間の休憩
```

7. destroy() 関数は、スクリプトが終了したときにブザーと LED の両方をオフにするようにします。これにより、GPIO ピンがアクティブな状態にならないようになります。

```
def destroy():
    """ スクリプトの終了時にリソースをクリーンアップします。"""
    print("")
    BeepPin.off()
    ALedPin.off()
```

8. キーボードで関連する文字を入力すると、 upper() が入力文字を大文字に変換します。 printf() はコンピュータ画面にクリアテキストを表示し、 morsecode() 関数はブザーと LED にモールス信号を出力させます。

```
try:
    while True:
        code = input("メッセンジャーを入力してください：")
        code = code.upper()  # モールス信号検索用に大文字に変換
        print(code)
        morsecode(code)
except KeyboardInterrupt:
    destroy()
```

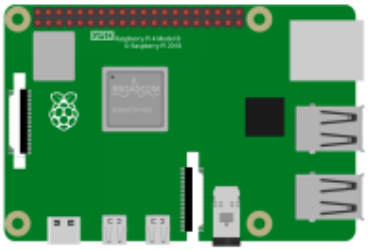
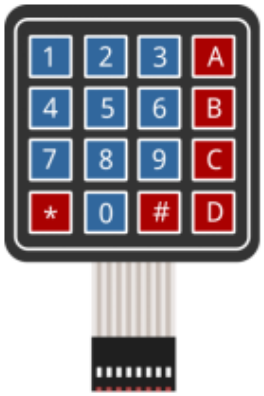
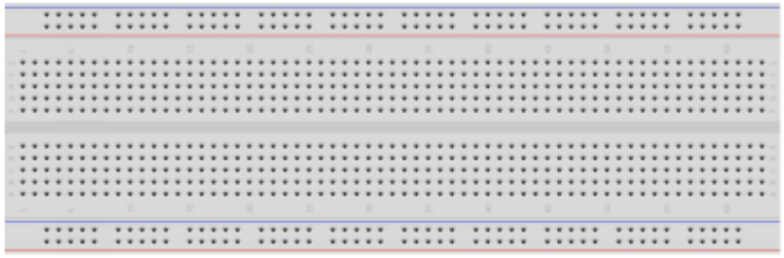
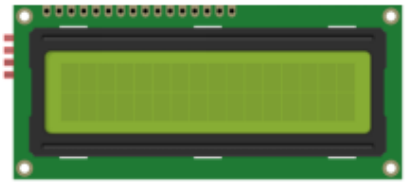
3.1.12 ゲーム – 数当て

はじめに

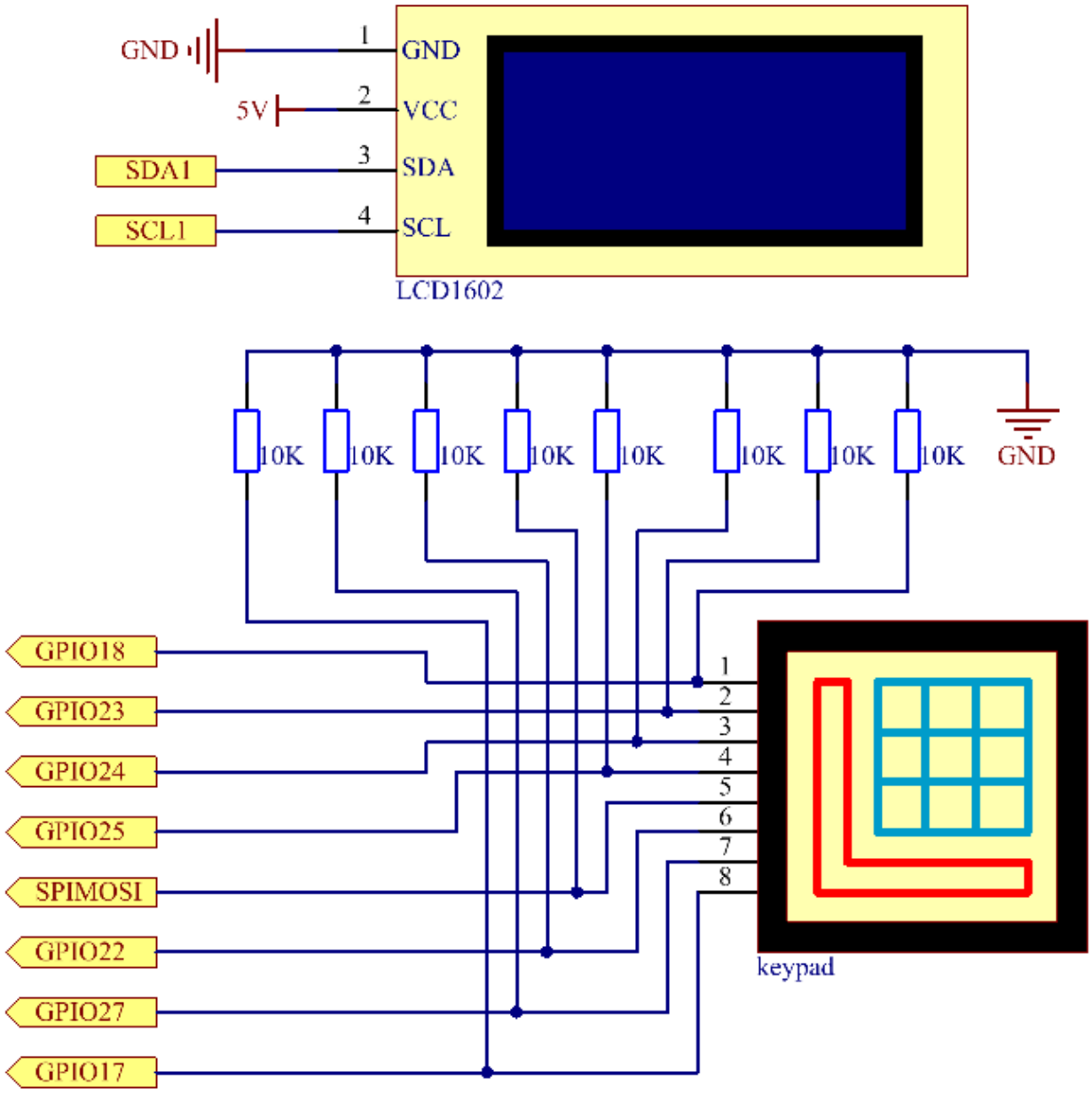
数当ては、あなたと友達が交互に数字（0〜99）を入力する楽しいパーティーゲームです。数字の入力に従って、範囲は狭まっていき、プレイヤーが謎を正しく解答するまで続きます。その後、プレイヤーは負けて罰せられます。たとえば、幸運の数字が 51 で、プレイヤー 1 が 50 を入力した場合、数字の範囲は 50〜99 に変わります。プレイヤー 2 が 70 を入力すると、数字の範囲は 50〜70 になります。プレイヤー 3 が 51 を入力した場合、そのプレイヤーは不運です。ここでは、キーパッドを使用して数字を入力し、LCD を使用して結果を表示します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

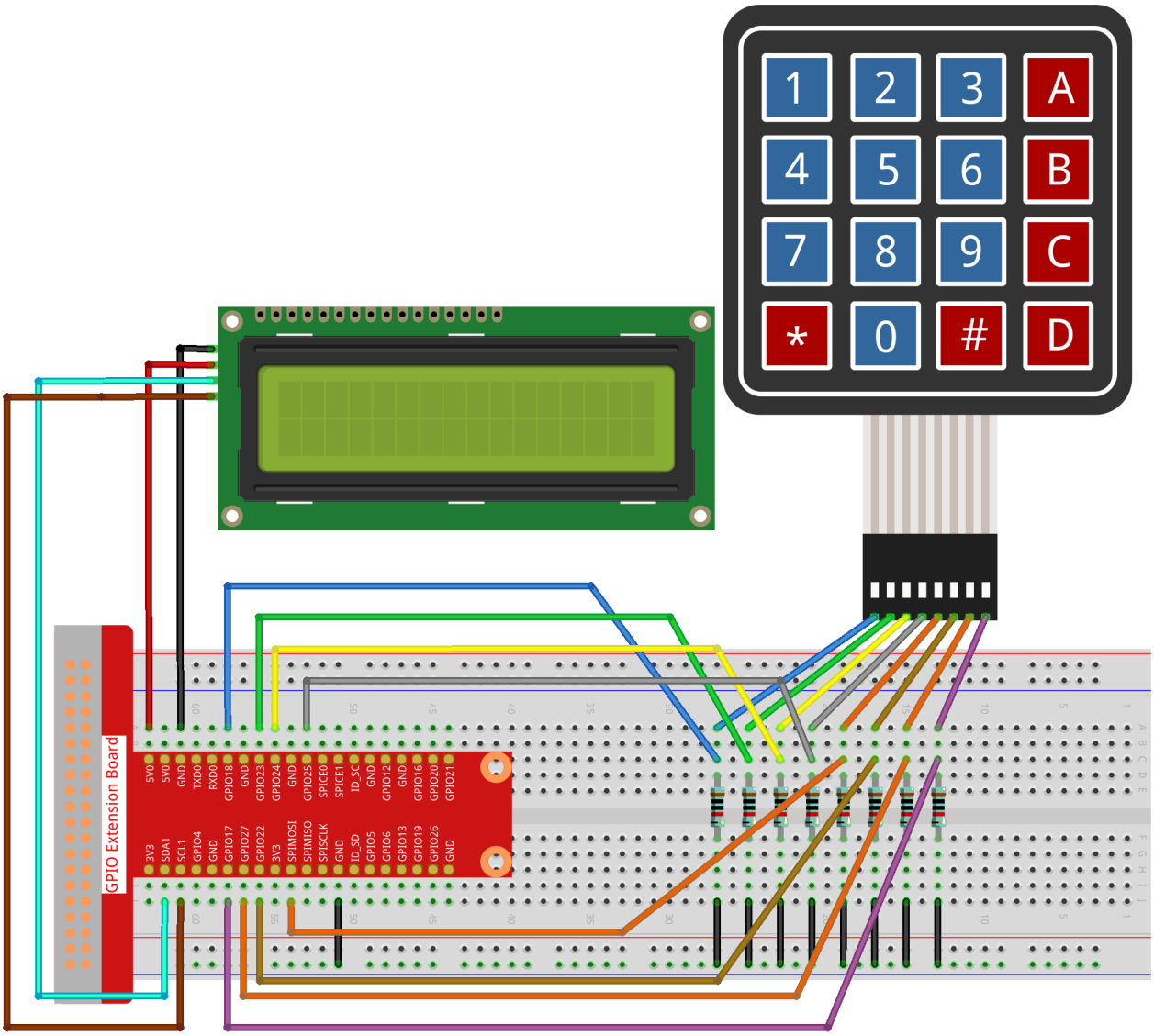
1 * Raspberry Pi 	1 * T-Extension Board	1 * Keypad 
1 * 40-pin Cable		Several Jumper Wires
		8 * Resistor 10KΩ
1 * Breadboard 		1 * I2C LCD1602 

回路図


GND
5V
SDA1
SCL1
1
2
3
4
GND
VCC
SDA
SCL
LCD1602
10K
10K
10K
10K
10K
10K
10K
10K
GND
GPIO18
GPIO23
GPIO24
GPIO25
SPIMOSI
GPIO22
GPIO27
GPIO17
1
2
3
4
5
6
7
8
keypad

実験手順

ステップ 1: 回路を組み立てます。



ステップ 2: I2C をセットアップします（詳細は *I2C* **構成** を参照）。

ステップ 3: ディレクトリを変更します。

```
cd ~/raphael-kit/python-pi5
```

ステップ 4: 実行します。

```
sudo python3 3.1.12_GAME_GuessNumber_zero.py
```

プログラムが実行されると、LCD に初期ページが表示されます：

```
ようこそ！
A ボタンを押して開始！
```

'A' ボタンを押すと、ゲームが開始し、LCD にゲームページが表示されます。

```
数字を入力してください：
0 ^^e2^^80^^b9point^^e2^^80^^b9 99
```

ランダムな数字 '‹point‹' がゲーム開始時に生成されますが、LCD には表示されません。必要なことは、それを推測することです。入力した数値は、最終的な計算が終了するまで、最初の行の末尾に表示されます（比較を開始するには 'D' を押し、入力された数値が **10** より大きい場合、自動的な比較が開始されます）。

'‹point‹' の数値範囲が 2 行目に表示されます。指定された範囲内で数値を入力する必要があります。数値を入力すると、範囲が狭まり、幸運か不運かにかかわらず、「当たりました！」と表示されます。

注釈:

- エラー「`FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'`」が表示される場合、I2C を有効にするには *I2C* **構成** を参照してください。
- エラー「`ModuleNotFoundError: No module named 'smbus2'`」が表示される場合、「`sudo pip3 install smbus2`」を実行してください。
- エラー「`OSError: [Errno 121] Remote I/O error`」が表示される場合、モジュールの接続が誤っているか、モジュールが壊れている可能性があります。
- コードと配線が正常であるにもかかわらず、LCD にコンテンツが表示されない場合は、背面のポテンショメーターを回してコントラストを上げることができます。

コード

注釈: 以下のコードを **修正/リセット/コピー/実行/停止** することができます。ただし、その前に「raphael-kit/python-pi5」のようなソースコードのパスに移動する必要があります。コードを変更した後、効果を確認するために直接実行できます。

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

from gpiozero import DigitalOutputDevice, Button
from time import sleep
```

(次のページに続く)

(前のページからの続き)

```
import LCD1602
import random

class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        キーパッドを指定した行と列のピン、キーレイアウトで初期化します。
        :param rows_pins: 行の GPIO ピンのリスト。
        :param cols_pins: 列の GPIO ピンのリスト。
        :param keys: キーパッド上のキーの配置。
        """
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # 行のピンをセットアップ
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # 列のピンをセットアッ
プ
        self.keys = keys  # キーパッドのレイアウトを定義

    def read(self):
        """
        現在押されているキーを読み取り、返します。
        :return: 押されたキーのリスト。
        """
        pressed_keys = []
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行をアクティブにする
            for j, col in enumerate(self.cols):
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])  # 押されたキーを追加
            row.off()  # 行を非アクティブにする
        return pressed_keys

# ゲームに関連する変数
count = 0
pointValue = 0
upper = 99
lower = 0

def setup():
    """
```

(次のページに続く)

(前のページからの続き)

```
    キーパッドと LCD ディスプレイを初期化するセットアップ関数。
    """
    global keypad, last_key_pressed, keys
    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 = []
    LCD1602.init(0x27, 1)  # LCD を初期化
    LCD1602.clear()
    LCD1602.write(0, 0, ' ようこそ!')
    LCD1602.write(0, 1, 'A を押して開始!')

def init_new_value():
    """
    新しい目標値を初期化し、ゲームのパラメーターをリセットします。
    """
    global pointValue, upper, lower, count
    pointValue = random.randint(0, 99)
    upper = 99
    lower = 0
    count = 0
    print(' 目標値は %d です' % pointValue)

def detect_point():
    """
    推測された数値が目標と一致するか、高すぎるか、低すぎるかをチェックします。
    :return: 1（正しい推測）または 0（それ以外）。
    """
    global count, upper, lower
    if count > pointValue and count < upper:
        upper = count
    elif count < pointValue and count > lower:
        lower = count
    elif count == pointValue:
        count = 0
```

(次のページに続く)

(前のページからの続き)

```
        return 1
    count = 0
    return 0

def lcd_show_input(result):
    """
    LCD に現在のゲームの状態と結果を表示します。
    :param result: 最後の推測の結果（0 または 1）。
    """
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0, 1, ' 当たりました！')
        sleep(5)
        init_new_value()
        lcd_show_input(0)
    else:
        LCD1602.write(0, 0, ' 数字を入力:')
        LCD1602.write(13, 0, str(count))
        LCD1602.write(0, 1, str(lower))
        LCD1602.write(3, 1, ' < 目標 < ')
        LCD1602.write(13, 1, str(upper))

def loop():
    """
    キーパッドの入力を処理し、ゲームの状態を更新し、LCD に結果を表示するメインループ。
    """
    global keypad, last_key_pressed, count
    while True:
        result = 0
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            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 ["A", "B", "C", "D", "#", "*"]:
```

(次のページに続く)

(前のページからの続き)

```
                continue
            count = count * 10 + int(pressed_keys[0])
            if count >= 10:
                result = detect_point()
            lcd_show_input(result)
        print(pressed_keys)
    last_key_pressed = pressed_keys
    sleep(0.1)

try:
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # 割り込み時に LCD をクリア
```

コード説明

1. GPIO Zero ライブラリからデジタル出力デバイスとボタンのための重要なクラスをインポートするセクションです。スクリプト内で遅延を導入するために、time モジュールの sleep 関数も含まれています。LCD1602 ライブラリは、テキストやデータ出力を表示するために役立つ LCD ディスプレイの操作に利用されます。さらに、ランダムな数字を生成するための関数を提供する random ライブラリも組み込まれています。これはプロジェクトのさまざまな面で有利になる可能性があります。

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

from gpiozero import DigitalOutputDevice, Button
from time import sleep
import LCD1602
import random
```

2. キーパッドのクラスを定義し、行と列のピンで初期化し、押されたキーを読み取るメソッドを定義します。

```
class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        指定された行と列のピン、およびキーレイアウトでキーパッドを初期化します。
        :param rows_pins: 行のための GPIO ピンのリスト。
        :param cols_pins: 列のための GPIO ピンのリスト。
        :param keys: キーパッド上のキーのレイアウト。
        """
```

(次のページに続く)

(前のページからの続き)

```
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # 行ピンの設定
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # 列ピンの設定
        self.keys = keys  # キーパッドのレイアウトを定義

    def read(self):
        """
        現在押されているキーを読み取り、返します。
        :return: 押されたキーのリスト。
        """
        pressed_keys = []
        for i, row in enumerate(self.rows):
            row.on()  # 現在の行をアクティブにする
            for j, col in enumerate(self.cols):
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])  # 押されたキーを追加
            row.off()  # 行を非アクティブにする
        return pressed_keys
```

3. ゲームに関連する変数「count」をゼロとして初期化し、キーパッドと LCD ディスプレイを設定し、ウェルカムメッセージと指示を表示します。ゲームでの特定のスコアや値を表す可能性がある「pointValue」変数をゼロに初期化します。当初は 99 に設定された「upper」は、数字当てゲームの最大値として使用される可能性があります。ゲームの最小境界として使用される可能性のあるゼロからの「lower」限界を設定します。

```
# ゲーム関連の変数
count = 0
pointValue = 0
upper = 99
lower = 0
```

4. キーパッドと LCD ディスプレイを設定し、ウェルカムメッセージと指示を表示します。

```
def setup():
    """
    キーパッドと LCD ディスプレイを初期化するためのセットアップ関数。
    """
    global keypad, last_key_pressed, keys
    rowsPins = [18, 23, 24, 25]
    colsPins = [10, 22, 27, 17]
```

(次のページに続く)

(前のページからの続き)

```
    keys = ["1", "2", "3", "A",
            "4", "5", "6", "B",
            "7", "8", "9", "C",
            "*", "0", "#", "D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []
    LCD1602.init(0x27, 1)  # LCD の初期化
    LCD1602.clear()
    LCD1602.write(0, 0, 'Welcome!')
    LCD1602.write(0, 1, 'Press A to Start!')
```

5. 新しいターゲット値を初期化し、ゲームのパラメータをリセットします。

```
def init_new_value():
    """
    新しいターゲット値を初期化し、ゲームのパラメータをリセットします。
    """
    global pointValue, upper, lower, count
    pointValue = random.randint(0, 99)
    upper = 99
    lower = 0
    count = 0
    print('point is %d' % pointValue)
```

6. 推測された数字がターゲットと一致するかどうかをチェックし、それに応じて推測範囲を更新します。

```
def detect_point():
    """
    推測された数字がターゲットか、高すぎるか、低すぎるかをチェックします。
    :return: 正しい推測の場合は 1、それ以外の場合は 0。
    """
    global count, upper, lower
    if count > pointValue and count < upper:
        upper = count
    elif count < pointValue and count > lower:
        lower = count
    elif count == pointValue:
        count = 0
        return 1
```

(次のページに続く)

(前のページからの続き)

```
    count = 0
    return 0
```

7. LCD にゲームの状態を表示し、現在の推測、範囲、結果を示します。

```
def lcd_show_input(result):
    """
    LCD に現在のゲームの状態と結果を表示します。
    :param result: 最後の推測の結果（0 または 1）。
    """
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0, 1, 'You have got it!')
        sleep(5)
        init_new_value()
        lcd_show_input(0)
    else:
        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))
```

8. キーパッド入力の処理、ゲームの状態の更新、LCD への結果の表示を行うメインループです。

```
def loop():
    """
    キーパッド入力の処理とゲームの状態の更新を行うメインゲームループ。
    """
    global keypad, last_key_pressed, count
    while True:
        result = 0
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
                if pressed_keys == ["A"]:
                    init_new_value()
                    lcd_show_input(0)
                elif pressed_keys == ["D"]:
                    result = detect_point()
```

(次のページに続く)

(前のページからの続き)

```
                lcd_show_input(result)
            elif pressed_keys[0] in keys:
                if pressed_keys[0] in ["A", "B", "C", "D", "#", "*"]:
                    continue
                count = count * 10 + int(pressed_keys[0])
                if count >= 10:
                    result = detect_point()
                lcd_show_input(result)
            print(pressed_keys)
        last_key_pressed = pressed_keys
        sleep(0.1)
```

9. セットアップを実行し、メインゲームループに入ります。キーボード割り込みを使用してクリーンな終了が可能です。

```
try:
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # 割り込み時に LCD をクリア
```

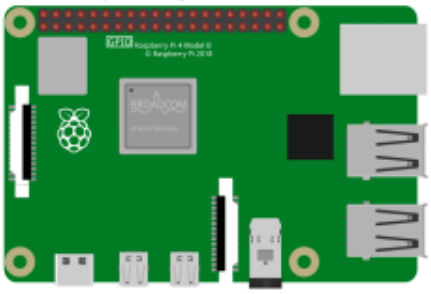
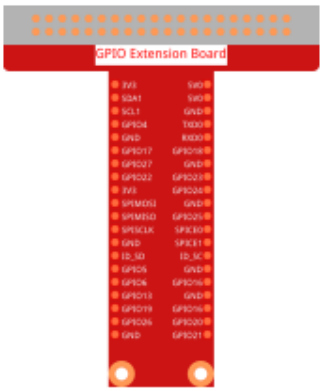
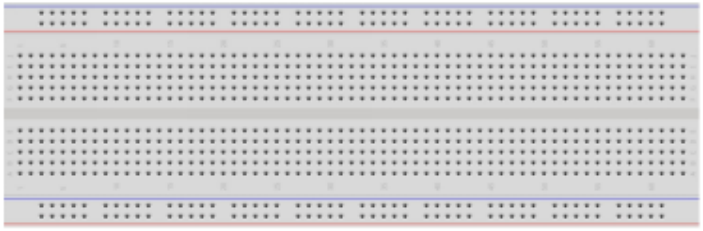
3.1.13 GAME - 10 秒間

はじめに

次に、集中力を試すゲームデバイスを作ってみましょう。傾きスイッチを棒に結び付けて、魔法の杖を作ります。杖を振ると、4 桁のセグメント表示がカウントを開始し、もう一度振るとカウントが停止します。表示されたカウントを **10.00** で停止させることができれば、あなたの勝利です。友達とこのゲームをして、誰が時間の魔法使いかを競い合うことができます。

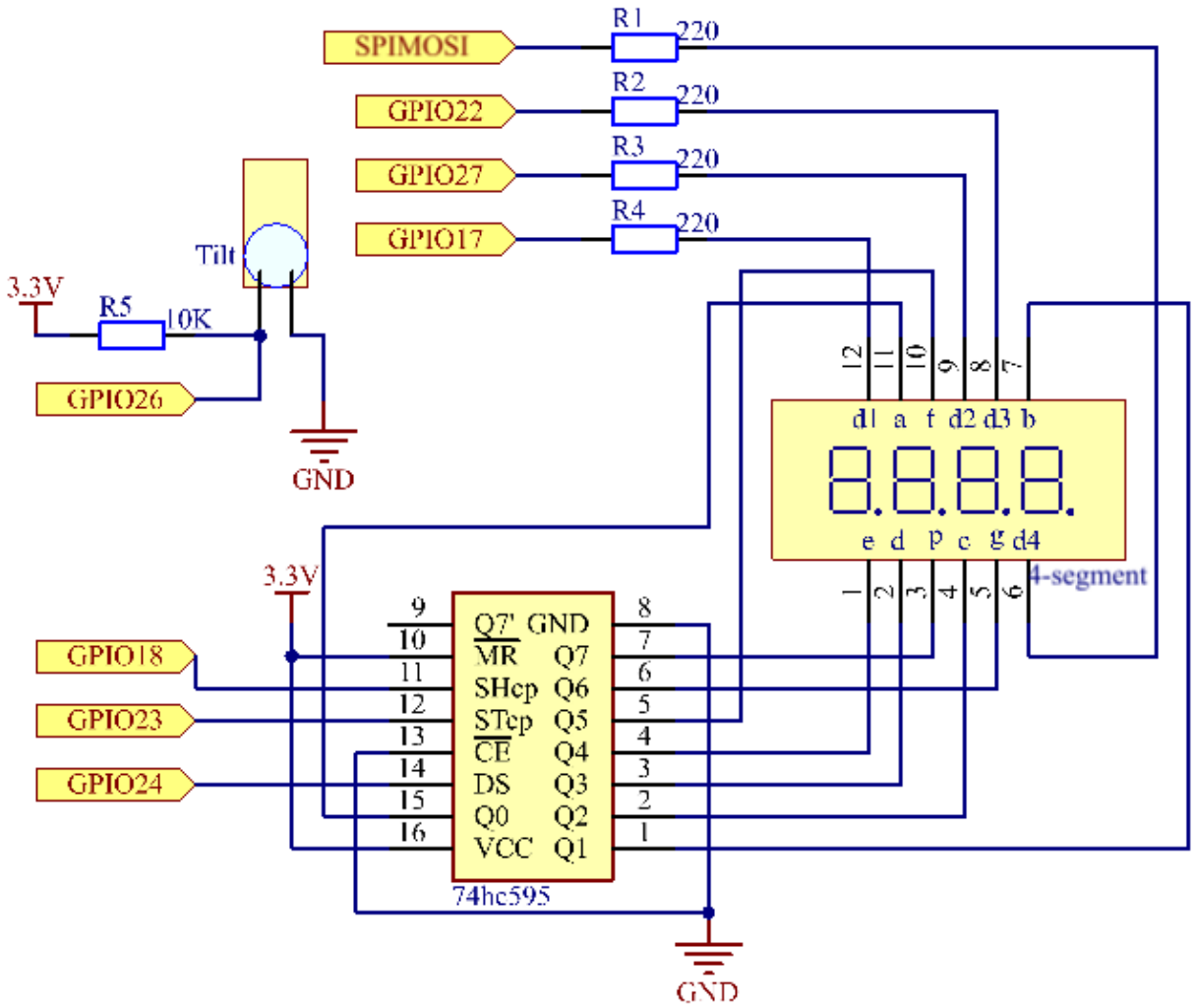
必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display
		1 * 74HC595
1 * 40-pin Cable		1 * Tilt Switch
		Several Jumper Wires
1 * Breadboard 		4 * Resistor(220Ω)
		1 * Resistor 10KΩ

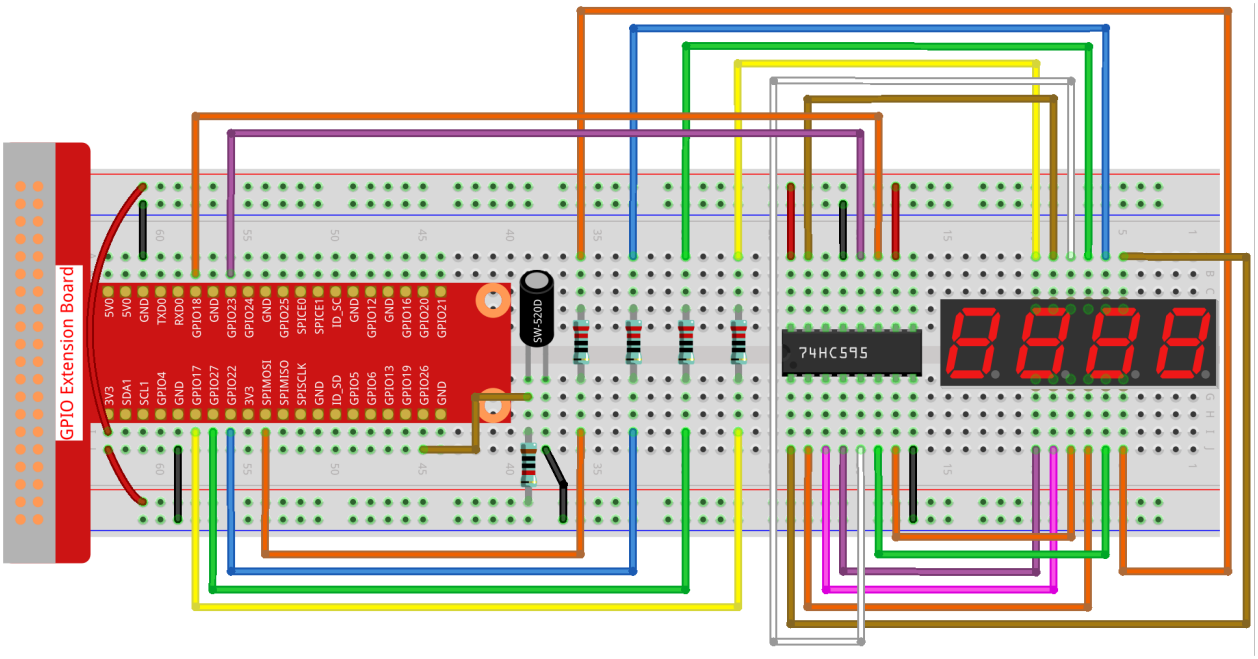
回路図

T-Board Name	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: 回路を組み立てる。



ステップ 2: コードのフォルダに移動する。

```
cd ~/raphael-kit/python-pi5
```

ステップ 3: 実行可能ファイルを実行する。

```
sudo python3 3.1.13_GAME_10Second_zero.py
```

杖を振ると、4 桁のセグメント表示がカウントを開始し、もう一度振るとカウントが停止します。表示されたカウントを **10.00** で停止させることができれば、あなたの勝利です。もう一度振ると、ゲームの次のラウンドが始まります。

コード

注釈: 下記のコードは **変更/リセット/コピー/実行/停止** が可能です。しかし、それを行う前に、 raphael-kit/python-pi5 のようなソースコードのパスに移動する必要があります。コードを変更した後、直接実行して効果を確認することができます。

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

(次のページに続く)

(前のページからの続き)

```
from gpiozero import OutputDevice, Button
import time
import threading

# GPIO 26 に接続されたボタンを初期化
sensorPin = Button(26)

# 74HC595 シフトレジスタに接続された GPIO ピンを定義
SDI = OutputDevice(24)   # シリアルデータ入力
RCLK = OutputDevice(23)  # レジスタクロック
SRCLK = OutputDevice(18) # シフトレジスタクロック

# 7 セグメント表示の各桁の選択用の GPIO ピンを定義
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメント表示で 0〜9 の数字を表示するためのセグメントコードを定義
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# カウンターとタイマー変数
counter = 0
timer1 = None
gameState = 0

def clearDisplay():
    """ 7 セグメント表示の全セグメントを消去します。 """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ デジットを表示するために 74HC595 シフトレジスタにデータをシフトします。 """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
```

(次のページに続く)

(前のページからの続き)

```
    RCLK.off()

def pickDigit(digit):
    """ 7セグメント表示でどの桁を表示するかを選択します。 """
    for pin in placePin:
        pin.off()
    placePin[digit].on()

def display():
    """ 現在のカウンター値を 7セグメント表示に表示します。 """
    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] - 0x80)

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

def stateChange():
    """ ボタン押下に基づいてカウンターの状態を変更します。 """
    global gameState, counter, timer1
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState == 1 and timer1 is not None:
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState + 1) % 2
```

(次のページに続く)

(前のページからの続き)

```
def loop():
    """ ボタンの状態をチェックし、表示を更新するメインループです。 """
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = sensorPin.value
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState = currentState

def timer():
    """ 0.01 秒ごとにカウンターを増加させるタイマー関数です。 """
    global counter, timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
    counter += 1

try:
    loop()
except KeyboardInterrupt:
    if timer1:
        timer1.cancel()
```

コードの説明

1. 必要なモジュールをインポートしてスクリプトを開始します。 gpiozero ライブラリはボタンなどの GPIO デバイスとのインターフェースに使用され、 time および threading モジュールはタイミング関連のタスクや並行操作を扱うために使用されるかもしれません。

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

   from gpiozero import OutputDevice, Button
   import time
   import threading
   ```

2. GPIO ピン 26 に接続された Button オブジェクトを初期化します。このセットアップにより、ボタン押下を検出することができます。

```
# GPIO 26 に接続されたボタンを初期化
sensorPin = Button(26)
```

3. シフトレジスタのシリアルデータ入力（SDI）、レジスタクロック入力（RCLK）、シフトレジスタクロック入力（SRCLK）に接続された GPIO ピンを初期化します。

```
# 74HC595 シフトレジスタに接続された GPIO ピンを定義
SDI = OutputDevice(24)   # シリアルデータ入力
RCLK = OutputDevice(23)  # レジスタクロック
SRCLK = OutputDevice(18) # シフトレジスタクロック
```

4. 7 セグメント表示の各桁の選択用のピンを初期化し、0〜9 の数字を表示するためのバイナリコードを定義します。

```
# 7 セグメント表示の各桁の選択用の GPIO ピンを定義
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# 7 セグメント表示で 0〜9 の数字を表示するためのセグメントコードを定義
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

5. 7 セグメント表示を制御するための関数です。 `clearDisplay` はすべてのセグメントをオフにし、`hc595_shift` はシフトレジスタにデータをシフトし、 `pickDigit` は表示上の特定の桁をアクティブにします。

```
def clearDisplay():
    """ 7 セグメント表示の全セグメントを消去します。 """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ デジットを表示するために 74HC595 シフトレジスタにデータをシフトします。 """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
```

(次のページに続く)

(前のページからの続き)

```
    RCLK.off()

def pickDigit(digit):
    """ 7セグメント表示でどの桁を表示するかを選択します。 """
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

6. 現在のカウンター値を 7 セグメント表示に表示する関数です。

```
def display():
    """ 現在のカウンター値を 7 セグメント表示に表示します。 """
    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] - 0x80)

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

7. ボタン押下に基づいてカウンターの状態を変更する関数です。

```
def stateChange():
    """ ボタン押下に基づいてカウンターの状態を変更します。 """
    global gameState, counter, timer1
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState == 1 and timer1 is not None:
```

(次のページに続く)

(前のページからの続き)

```
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState + 1) % 2
```

8. ボタンの状態を継続的にチェックし、表示を更新するメインループです。ボタンの状態が変わると、stateChange を呼び出します。

```
def loop():
    """ ボタンの状態をチェックし、表示を更新するメインループです。 """
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = sensorPin.value
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState = currentState
```

9. 0.01 秒ごとにカウンターを増加させるタイマー関数です。

```
def timer():
    """ 0.01 秒ごとにカウンターを増加させるタイマー関数です。 """
    global counter, timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
    counter += 1
```

10. メインループを実行し、キーボード割り込み（Ctrl+C）を使用してプログラムからクリーンに終了することができます。

```
try:
    loop()
except KeyboardInterrupt:
    if timer1:
        timer1.cancel()
```

3.1.14 GAME– 逆逆

はじめに

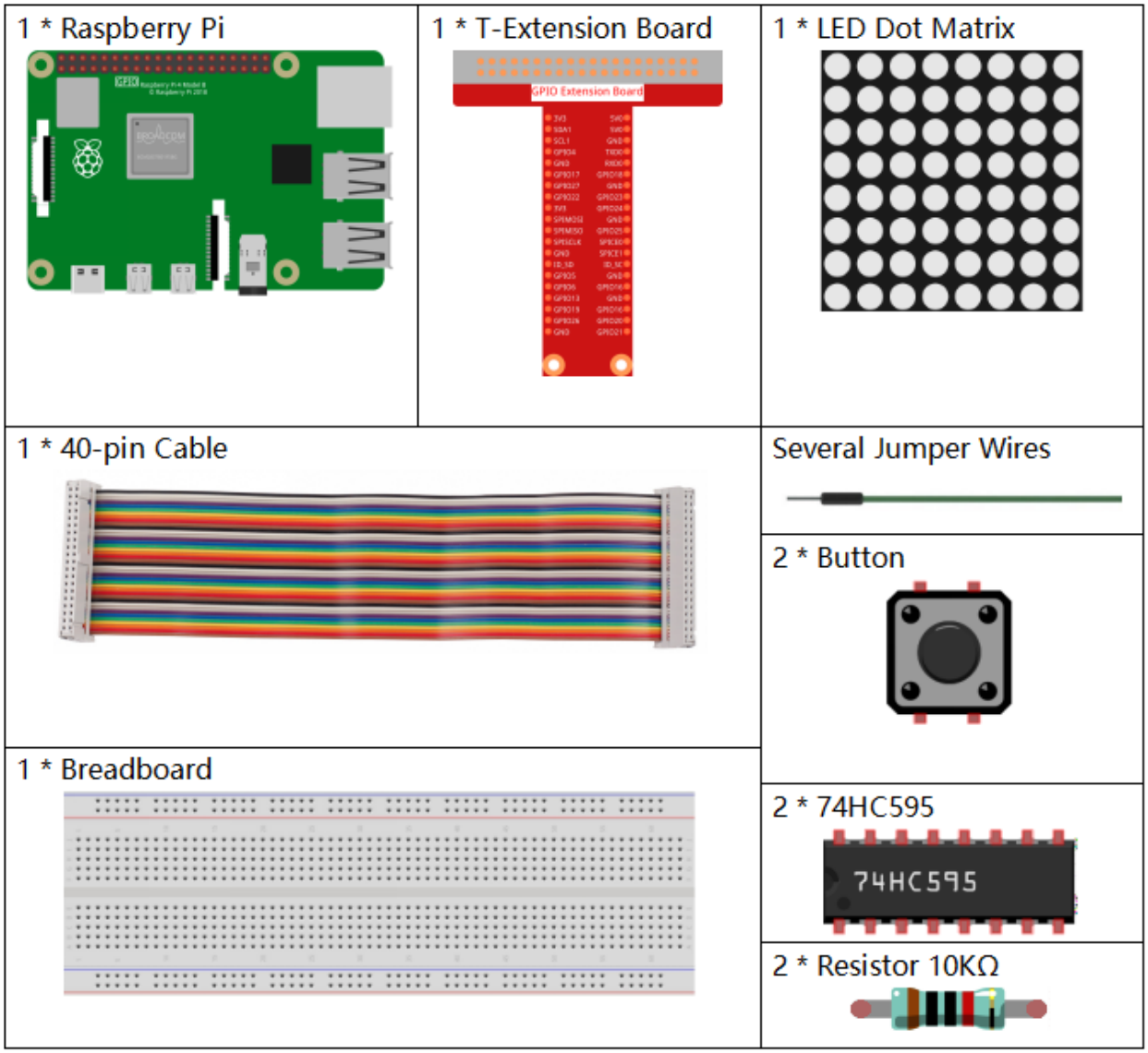
このレッスンでは、面白いゲームデバイスを作ります。その名も「逆逆」です。

ゲーム中、ドットマトリクスはランダムに矢印を表示します。あなたがすることは、限られた時間内に矢印の反対方向のボタンを押すことです。時間切れ、または矢印と同じ方向のボタンを押すと、あなたはアウトです。

このゲームはあなたの逆向き思考を本当に鍛えることができます。さあ、試してみましょうか？

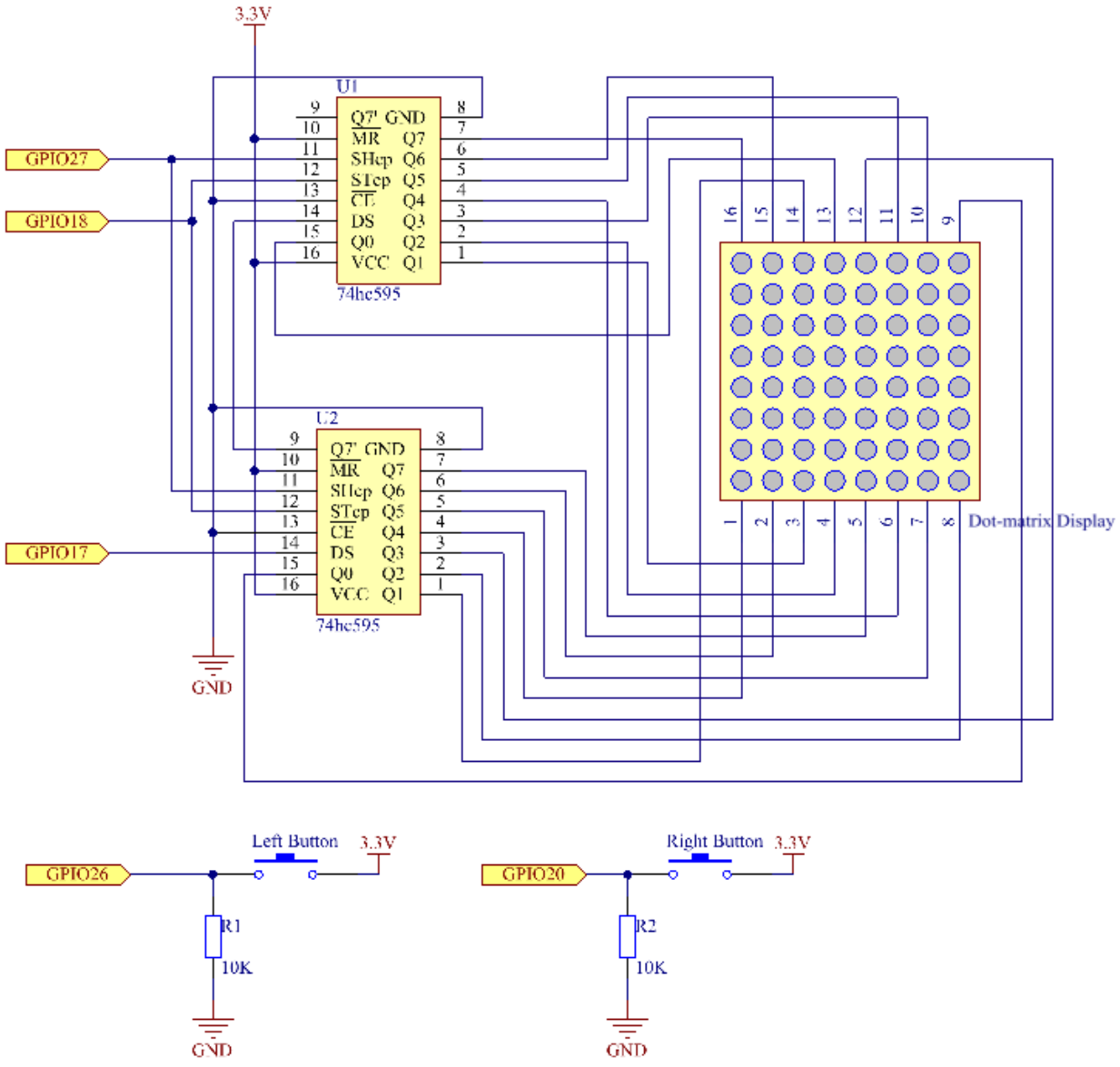
必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。



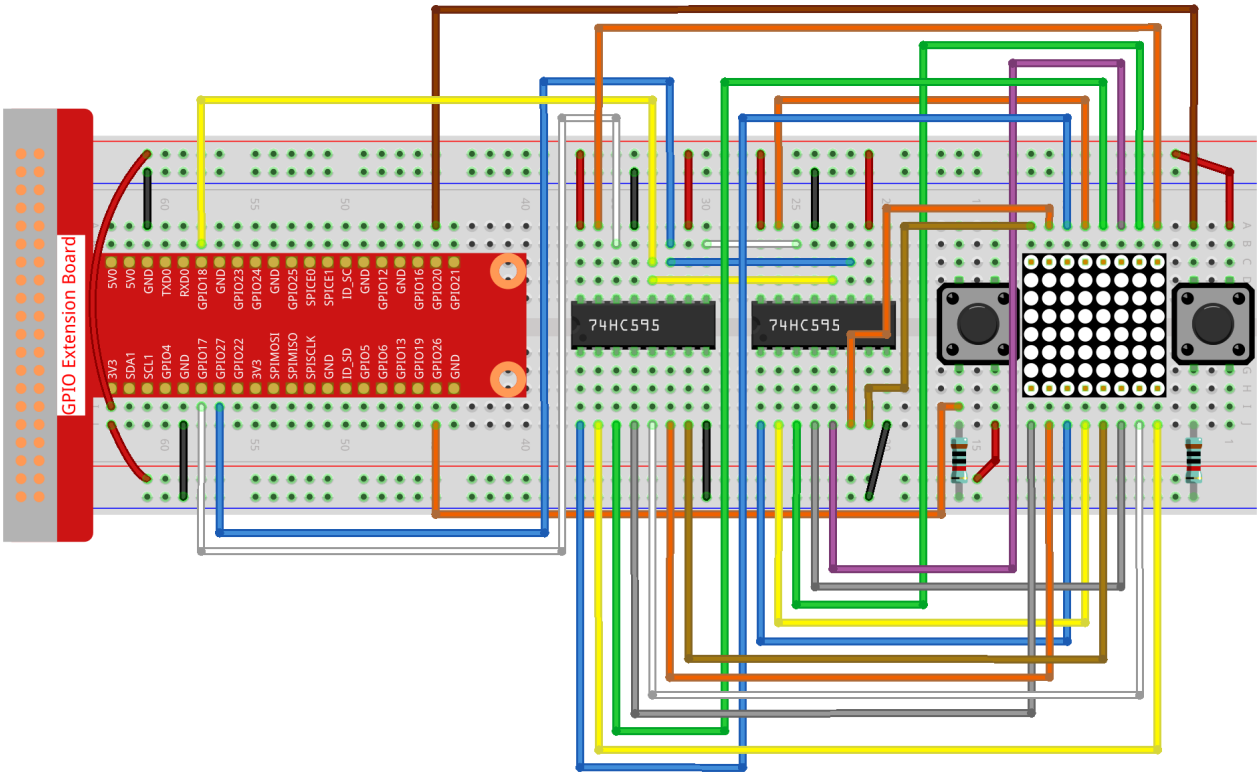
回路図

T-Board Name	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: 回路を組み立てる。



ステップ 2: コードファイルを開く。

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

ステップ 3: 実行。

```
sudo python3 3.1.14_MotionControl_zero.py
```

プログラムを開始すると、ドットマトリクスに右または左を指す矢印が表示されます。あなたがすることは、限られた時間内に矢印の反対方向のボタンを押すことです。そうするとドットマトリクスに「√」が表示されます。時間切れ、または矢印と同じ方向のボタンを押すと、アウトでドットマトリクスに「x」が表示されます。上下左右の 4 方向に対応するために、2 つの新しいボタンを追加するか、ジョイスティックキーに置き換えてゲームの難易度を上げることもできます。

コード

注釈: 下記のコードは **変更/リセット/コピー/実行/停止** が可能です。しかし、それを行う前に、davinci-kit-for-raspberry-pi/python-pi5 のようなソースコードのパスに移動する必要があります。コー

ドを変更した後、直接実行して効果を確認することができます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, Button
import time
import threading
import random

# 74HC595 シフトレジスタの GPIO ピン
SDI = OutputDevice(17)   # シリアルデータ入力
RCLK = OutputDevice(18)  # レジスタクロック
SRCLK = OutputDevice(27) # シフトレジスタクロック

# ボタンの GPIO ピン
AButtonPin = Button(20)  # ボタン A
BButtonPin = Button(26)  # ボタン B

# ゲーム変数の初期化
timerPlay = 0
timerCheck = 0
waypoint = "NULL"
stage = "NULL"

# LED マトリクス表示の矢印グリフ
arrow = {
    "right": [0xFF, 0xEF, 0xDF, 0x81, 0xDF, 0xEF, 0xFF, 0xFF],
    "left": [0xFF, 0xF7, 0xFB, 0x81, 0xFB, 0xF7, 0xFF, 0xFF]
}

# 正解/誤りのフィードバックグリフ
check = {
    "wrong": [0xFF, 0xBB, 0xD7, 0xEF, 0xD7, 0xBB, 0xFF, 0xFF],
    "right": [0xFF, 0xFF, 0xF7, 0xEB, 0xDF, 0xBF, 0xFF, 0xFF]
}

def hc595_shift(dat):
    """ 74HC595 シフトレジスタにデータをシフトします。 """
    for i in range(8):
        SDI.value = 0x80 & (dat << i)
```

(次のページに続く)

(前のページからの続き)

```
        SRCLK.on()
        SRCLK.off()

def display(glyphCode):
    """ LED マトリクスにグリフを表示します。 """
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80 >> i)
        RCLK.on()
        RCLK.off()

def creatGlyph():
    """ ゲームの新しいグリフを作成し、プレイタイマーを開始します。 """
    global waypoint, stage, timerPlay
    waypoint = random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()

def checkPoint(inputKey):
    """ プレイヤーの入力をチェックし、ゲームの状態を更新します。 """
    global waypoint, stage, timerCheck
    if inputKey == "empty" or inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()

def timeOut():
    """ ゲームのタイムアウトシナリオを処理します。 """
    checkPoint("empty")

def getKey():
    """ ボタン押下を検出し、チェックポイントをトリガーします。 """
    if AButtonPin.is_pressed and not BButtonPin.is_pressed:
        checkPoint("right")
```

(次のページに続く)

(前のページからの続き)

```
    elif not AButtonPin.is_pressed and BButtonPin.is_pressed:
        checkPoint("left")

def main():
    """ メインゲームループ。 """
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])

def destroy():
    """ プログラム終了時にリソースをクリーンアップします。 """
    global timerPlay, timerCheck
    timerPlay.cancel()  # プレイタイマーをキャンセル
    timerCheck.cancel()  # チェックポイントタイマーをキャンセル

# ゲームを実行し、キーボード割り込みでクリーンな終了を処理
try:
    main()
except KeyboardInterrupt:
    destroy()
```

コードの説明

1.1.6 LED ドットマトリクス を基にして、このレッスンでは **2** つのボタンを追加し、面白いゲームデバイスを作ります。ですので、ドットマトリクスにあまり慣れていない場合は、*1.1.6 LED* **ドットマトリックス** を参照してください。

1. 必要なライブラリをインポートしてコードを開始します。「gpiozero」はボタンや出力デバイスなどの GPIO ピンとのやり取りに使用されます。「time」は遅延を追加するため、「threading」は複数のタスクを同時に実行するため、「random」はプロジェクトでのランダム性を導入するのに役立ちます。

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, Button
import time
import threading
import random
```

2. シフトレジスタ（「SDI」、「RCLK」、「SRCLK」）およびボタン（「AButtonPin」、「BButtonPin」）用の GPIO ピンを初期化します。シフトレジスタは、少ない GPIO ピンで複数の LED を制御するために使用され、LED マトリクス表示に不可欠です。

```
# 74HC595 シフトレジスタの GPIO ピン
SDI = OutputDevice(17)   # シリアルデータ入力
RCLK = OutputDevice(18)  # レジスタクロック
SRCLK = OutputDevice(27) # シフトレジスタクロック

# ボタンの GPIO ピン
AButtonPin = Button(20)  # ボタン A
BButtonPin = Button(26)  # ボタン B
```

3. ゲームロジックに使用される変数、例えばタイマーやゲーム状態指標を初期化します。

```
# ゲーム変数の初期化
timerPlay = 0
timerCheck = 0
waypoint = "NULL"
stage = "NULL"
```

4. LED マトリクスに表示する矢印とフィードバック（正解/誤り）のバイナリパターンを定義します。各配列要素は LED マトリクスの行を表し、「1」と「0」はそれぞれ LED がオンまたはオフであることを示します。

```
# LED マトリクス表示の矢印グリフ
arrow = {
    "right": [0xFF, 0xEF, 0xDF, 0x81, 0xDF, 0xEF, 0xFF, 0xFF],
    "left": [0xFF, 0xF7, 0xFB, 0x81, 0xFB, 0xF7, 0xFF, 0xFF]
}

# 正解/誤りのフィードバックグリフ
check = {
    "wrong": [0xFF, 0xBB, 0xD7, 0xEF, 0xD7, 0xBB, 0xFF, 0xFF],
    "right": [0xFF, 0xFF, 0xF7, 0xEB, 0xDF, 0xBF, 0xFF, 0xFF]
}
```

5. この関数は 1 バイトのデータを 74HC595 シフトレジスタにシフトします。 dat バイトの各ビットに対して繰り返し処理を行い、 SDI ピンを高または低に設定し、 SRCLK ピンをトグルしてビットをレジスタにシフトします。

```
def hc595_shift(dat):
    """ 74HC595 シフトレジスタにデータをシフトします。 """
    for i in range(8):
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
```

6. この関数は LED マトリクスにグリフを表示します。 glyphCode で表されるグリフの各行とその行のアドレスを hc595_shift を使用してシフトレジスタに送信し、 RCLK ピンをトグルして表示を更新します。

```
def display(glyphCode):
    """ LED マトリクスにグリフを表示します。 """
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80 >> i)
        RCLK.on()
        RCLK.off()
```

7. この関数はランダムに arrow 辞書からグリフを選択し、プレイタイマーを開始し、ゲームのステージを「PLAY」に設定します。ゲームでのタイミング制御には threading.Timer が使用されます。

```
def creatGlyph():
    """ ゲームの新しいグリフを作成し、プレイタイマーを開始します。 """
    global waypoint, stage, timerPlay
    waypoint = random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()
```

8. この関数はプレイヤーの入力を現在のグリフと比較します。入力が正しい場合は、waypoint を「right」に設定し、そうでない場合は「wrong」に設定します。次に現在のプレイタイマーをキャンセルし、次のグリフのための新しいタイマーを開始します。

```
def checkPoint(inputKey):
    """ プレイヤーの入力をチェックし、ゲームの状態を更新します。 """
    global waypoint, stage, timerCheck
    if inputKey == "empty" or inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
```

(次のページに続く)

(前のページからの続き)

```
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()
```

9. この関数はゲームがタイムアウトしたときに呼び出されます。「empty」を引数として checkPoint を呼び出し、時間内にボタンが押されなかったことを示します。

```
def timeOut():
    """ ゲームのタイムアウトシナリオを処理します。 """
    checkPoint("empty")
```

10. この関数はボタンの状態を確認します。「AButtonPin」が押されていて「BButtonPin」が押されていない場合は、「right」として checkPoint を呼び出します。逆に「BButtonPin」が押されていて「AButtonPin」が押されていない場合は、「left」として checkPoint を呼び出します。

```
def getKey():
    """ ボタン押下を検出し、チェックポイントをトリガーします。 """
    if AButtonPin.is_pressed and not BButtonPin.is_pressed:
        checkPoint("right")
    elif not AButtonPin.is_pressed and BButtonPin.is_pressed:
        checkPoint("left")
```

11. main 関数はゲームの流れを制御します。グリフを作成して開始し、継続的にゲームのステージを確認します。"PLAY"ステージでは、現在のグリフを表示し、ボタン押下をチェックします。"CHECK"ステージでは、プレイヤーのアクションに基づいてフィードバックを表示します。

```
def main():
    """ メインゲームループ。 """
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])
```

12. この関数はプログラムが終了する際に実行中のタイマーをキャンセルし、クリーンなシャットダウンを保証します。

```
def destroy():
    """ プログラム終了時にリソースをクリーンアップします。 """
    global timerPlay, timerCheck
    timerPlay.cancel()  # プレイタイマーをキャンセル
    timerCheck.cancel()  # チェックポイントタイマーをキャンセル
```

13. ゲームは try ブロックで実行されます。 KeyboardInterrupt （Ctrl+C を押すなど）が発生した場合、例外をキャッチし、 destroy を呼び出して終了前にクリーンアップします。

```
# ゲームを実行し、キーボード割り込みでクリーンな終了を処理
try:
    main()
except KeyboardInterrupt:
    destroy()
```

1.13 Processing で遊ぶ (Pi 5 用ではない)

1.13.1 Processing とは？

Processing は、アニメーションに重点を置いた視覚指向のアプリケーションの開発を容易にし、対話を通じてユーザーに即時のフィードバックを提供するために作成された単純なプログラミング環境です。 開発者は、コードでアイデアを「スケッチ」する手段を望んでいました。 その機能が過去 10 年間で拡大するにつれて、Processing は、スケッチの役割に加えて、より高度な本番レベルの作業に使用されるようになりました。 もともとアーティストやデザイナーを対象とした Java のドメイン固有の拡張機能として構築された Processing は、大規模なインストール作業、モーショングラフィックス、複雑なデータの視覚化に使用される本格的なデザインおよびプロトタイピングツールに進化しました。

Processing は Java に基づいていますが、Processing のプログラム要素は非常に単純であるため、Java を知らなくても使用方法を学ぶことができます。 Java に精通している場合は、API がどのように機能するかを理解するまで、Processing はしばらくの間 Java と関係があることを忘れておくのが最善です。

このテキストは、チュートリアル Processing Overview からのものです。

1.13.2 Processing をインストールする

注釈: Processing を使用する前に、Raspberry Pi デスクトップにリモートでアクセスするか (*Raspberry Pi* **の設定**) 、RaspberryPi のディスプレイを接続する必要があります。

最初に https://processing.org/download にアクセスし、`Linux ARM32-bit` バージョンを選択します。

次に、ほとんどの Linux ユーザーになじみのある `.tar.gz` ファイルにダウンロードします。ファイルをホームディレクトリにダウンロードし、ターミナル ウィンドウを開いて次のように入力します。

```
tar xvfz processing-xxxx.tgz
```

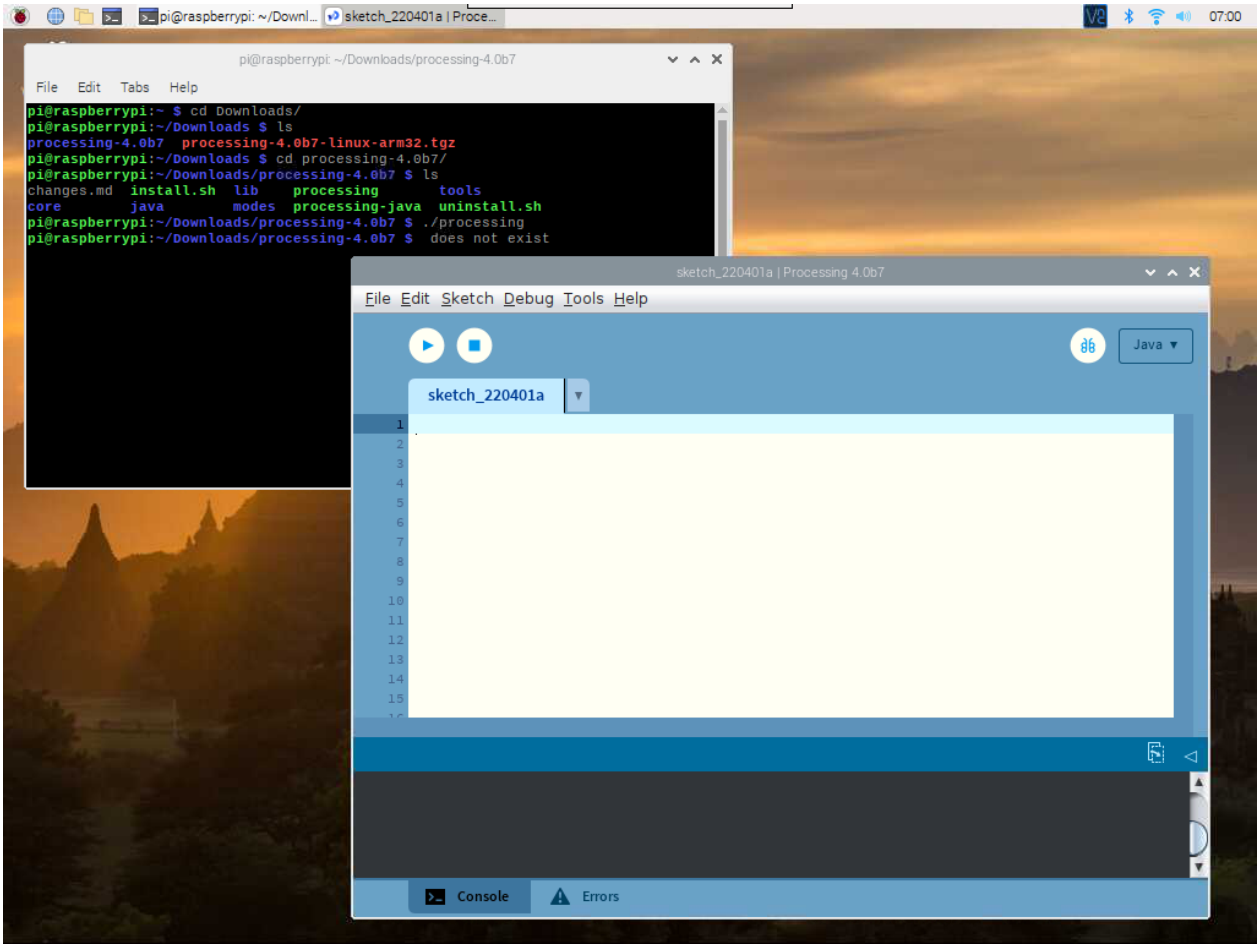
(xxxx をファイル名の残りの部分 (バージョン番号) に置き換えます。) これにより、processing-2.0 などの名前のフォルダーが作成されます。次に、そのディレクトリに移動します。

```
cd processing-xxxx
```

そしてそれを実行します：

```
./processing
```

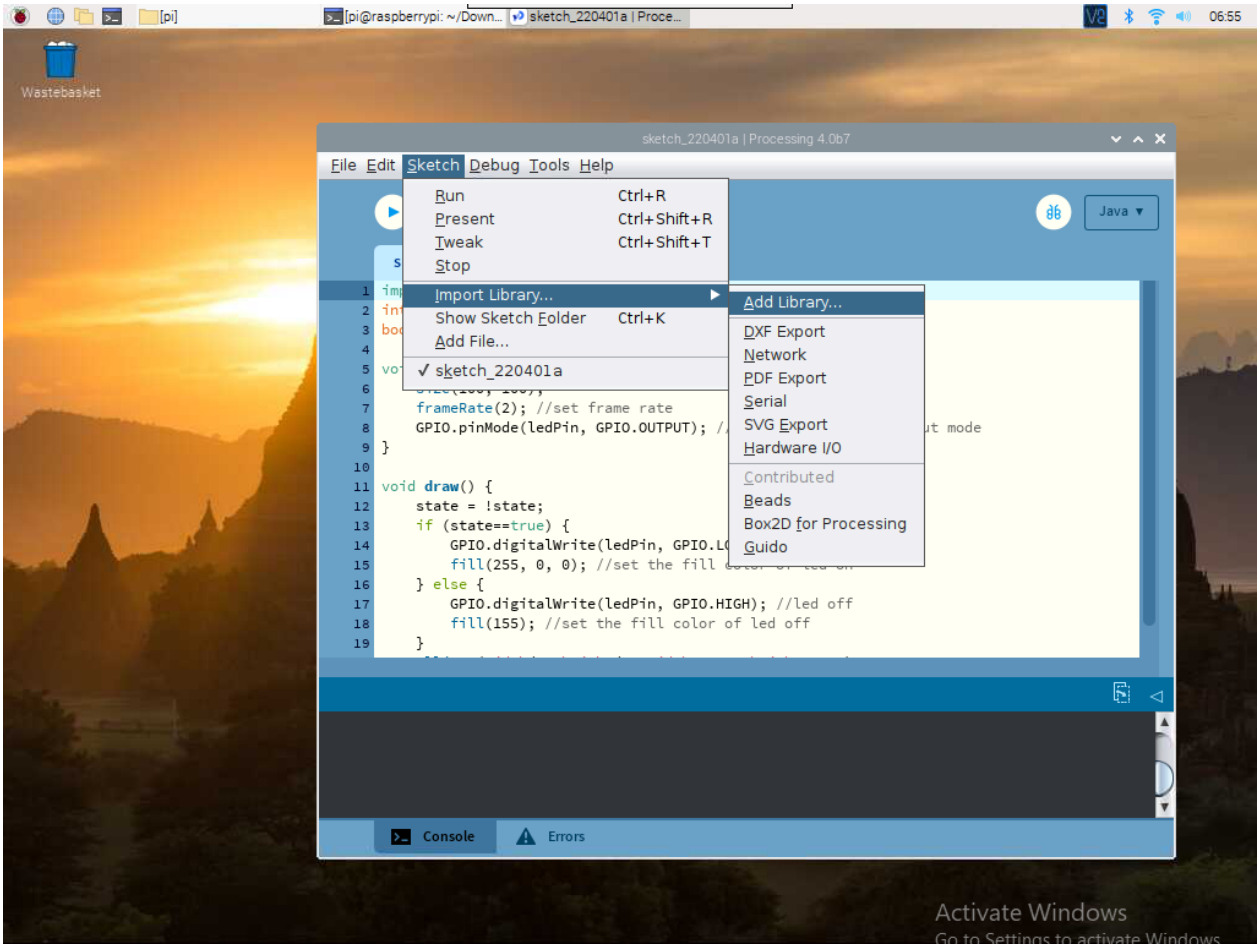
運が良ければ、メインの Processing ウィンドウが表示されます。



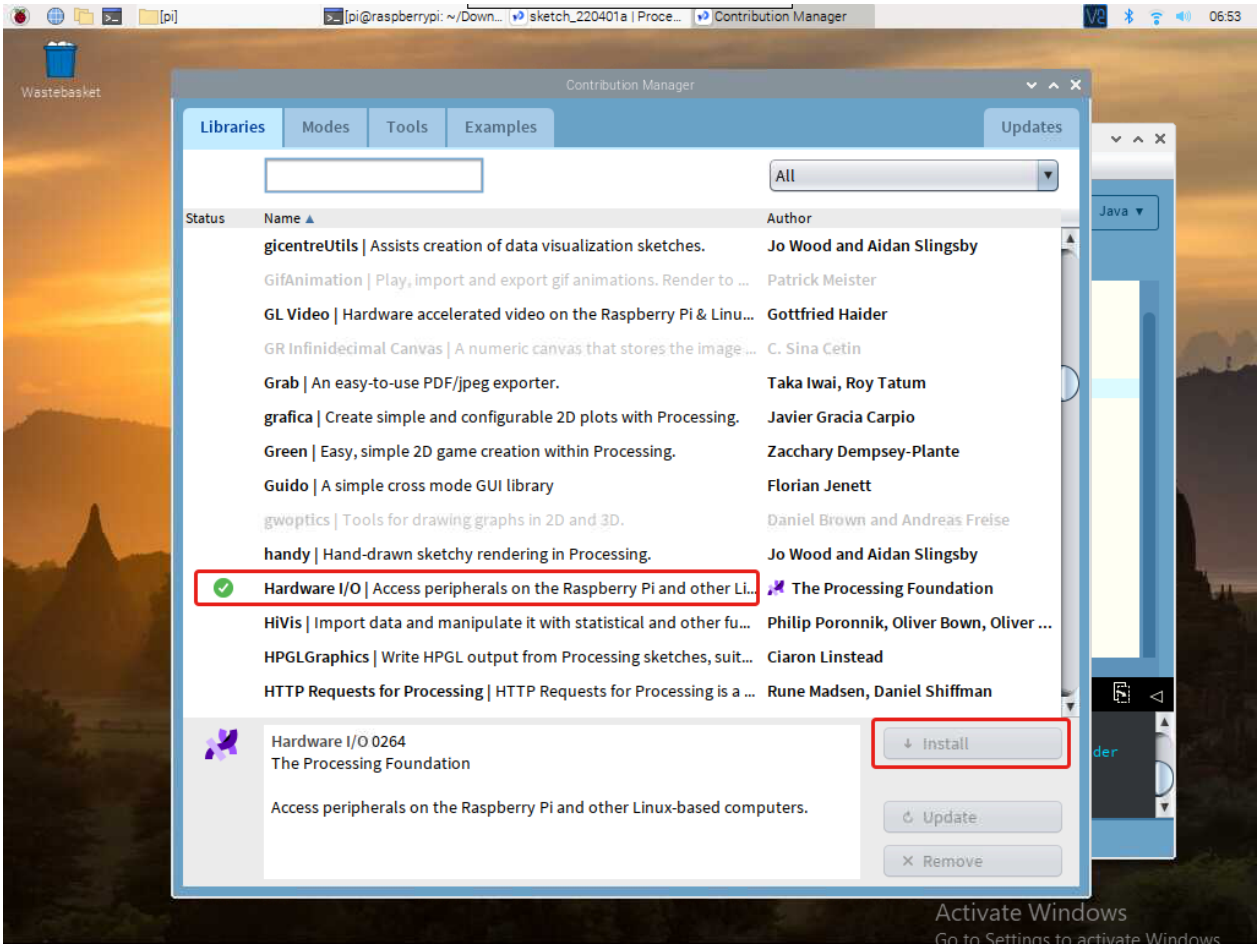
1.13.3 安装硬件 I/O

Raspberry Pi の GPIO を使用するには、 ハードウェア I/O ライブラリ を手動で追加する必要があります。

`Sketch` -> `Import Library` -> `Add Library...` をクリックします。



ハードウェア I/O を見つけて選択し、[インストール] をクリックします。完了すると、チェックマーク アイコンが表示されます。



1.13.4 プロジェクト

棒人間を描く

これで、Processing Development Environment（PDE）を実行しています。それほど多くはありません。 広い領域はテキストエディタで、上部にボタンの列があります。これがツールバーです。 エディタの下にはメッセージエリアがあり、その下にはコンソールがあります。メッセージエリアは 1 行のメッセージに使用され、コンソールはより技術的な詳細に使用されます。

Processing を使って棒線画を描くことに慣れましょう。

スケッチ

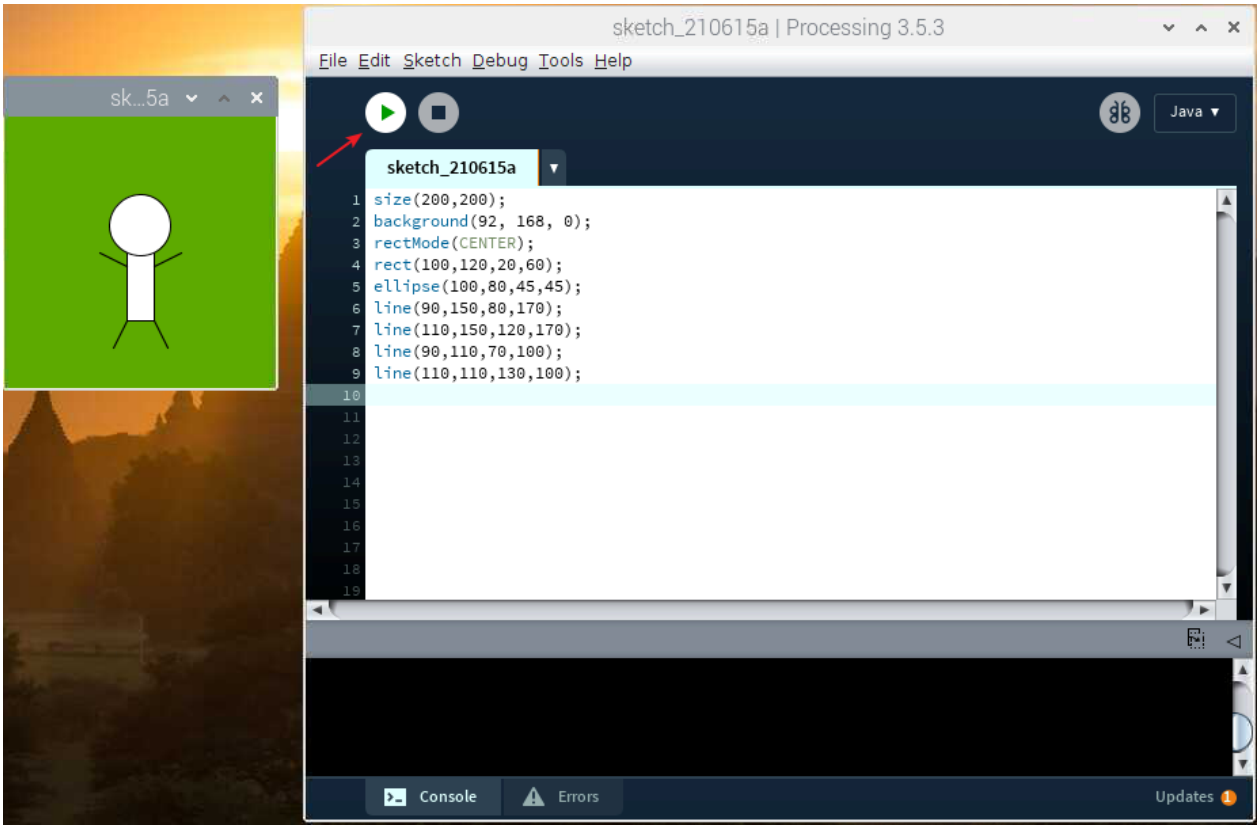
以下のスケッチを Processing にコピーして実行します。 新しい表示ウィンドウが表示され、応援の仲人が描かれます。

```
size(200,200);
background(92, 168, 0);
```

(次のページに続く)

(前のページからの続き)

```
rectMode(CENTER);
rect(100,120,20,60);
ellipse(100,80,45,45);
line(90,150,80,170);
line(110,150,120,170);
line(90,110,70,100);
line(110,110,130,100);
```



注釈: それを実行し、メッセージ領域が赤くなり、いくつかのエラーが報告された場合は、スケッチに問題があります。 サンプルスケッチを正確にコピーするようにしてください。数字は括弧で囲み、各数字の間にコンマを入れ、行はセミコロンで終了する必要があります。

原理

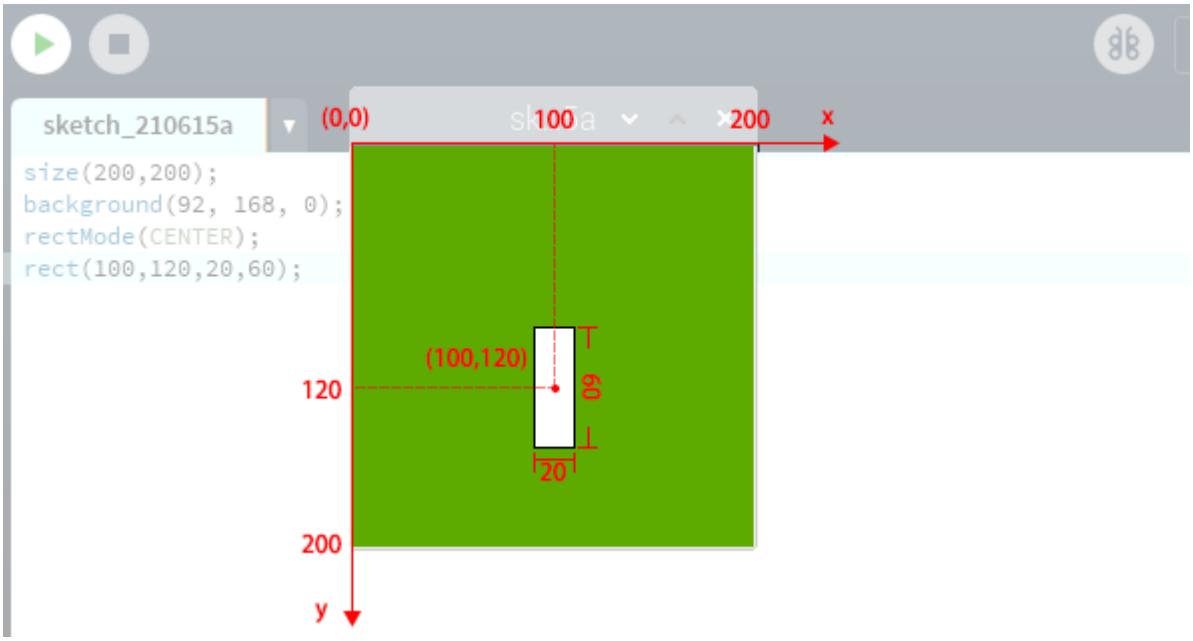
ここで重要なのは、表示ウィンドウを正方形の紙として扱うことができることを理解することです。

表示ウィンドウの各ピクセルは、空間内のポイントの位置を決定する座標（x、y）です。 座標の原点（0,0）は左上隅にあり、X 軸の正の方向は水平方向に右、Y 軸の正の方向は垂直方向に下向きです。

重要なのは、これらのピクセル座標にどのような形と色を表示するかを指定することです。

たとえば、座標（100,120）を中点として、幅 20、高さ 60 の長方形を描画します。

```
rectMode(CENTER);
rect(100,120,20,60);
```



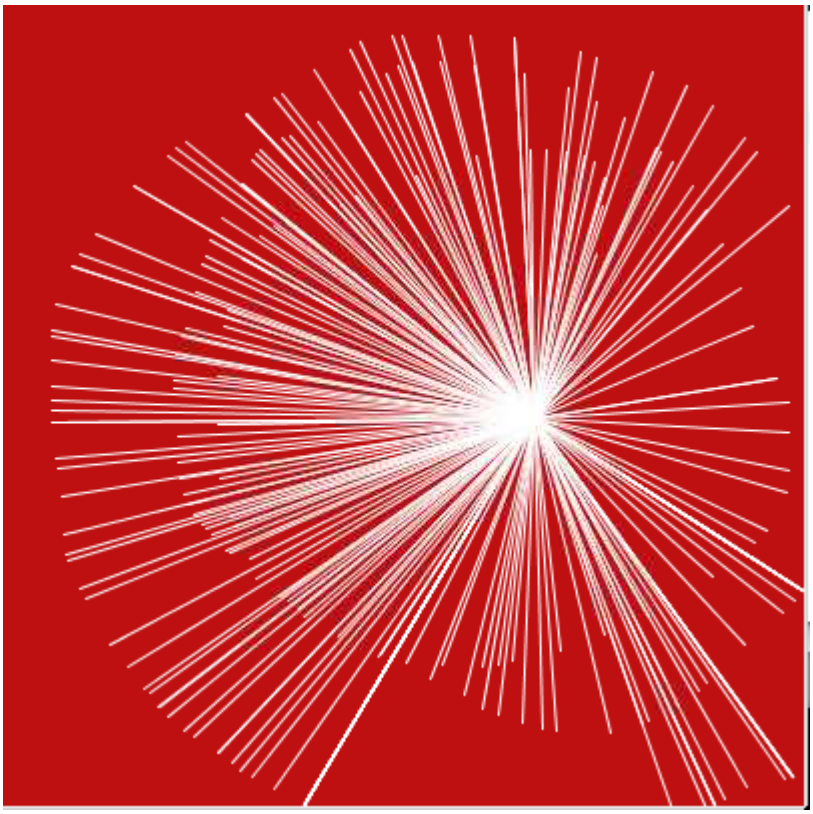
表示ウィンドウと軸の関係を理解したら、このスケッチは難しくありません。いくつかの簡単なグラフィック描画ステートメントを理解する必要があります。

- `size(width, height)`: 表示ウィンドウの幅と高さの寸法をピクセル単位で定義します。
- `background(red, green, blue)`: 表示ウィンドウの背景色を設定します。
- `rectMode(mode)`: `rect()` に指定されたパラメーターの解釈方法を変更することにより、長方形が描画される場所を変更します。
- `rect(x, y, width, height)`: 画面に長方形を描画します。
- `ellipse(x, y, width, height)`: 画面に楕円（楕円）を描画します。
- `line(x1, y1, x2, y2)`: 画面に線（2 点間の直接パス）を描画します。

詳細については、 Processing Reference を参照してください。

こんにちはマウス

このプロジェクトでは、マウスはポイントに向かって線を描き続けます。マウスを動かすと、ユニークな星の線が描画されます。マウスを押して、描画を再開します。



スケッチ

```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}

void draw() {
    line(pointX, pointY, mouseX, mouseY);
}

void mousePressed() {
    pointX=mouseX;
```

(次のページに続く)

(前のページからの続き)

```
    pointY=mouseY;
    background(192, 16, 18);
}
```

原理

前のプロジェクトは、アニメーションやインタラクションなしで単一の画像を描画していました。

インタラクティブなスケッチを作成する場合は、 setup() 関数と draw() 関数（これらは自動的に呼び出される組み込み関数）を追加してフレームを作成する必要があります。

- setup()：スケッチの開始時に 1 回だけ実行されます。
- draw() ：繰り返し実行されます。通常、アニメーションを描画するためのスケッチを追加します。

```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}

void draw() {
    line(pointX, pointY, mouseX, mouseY);
}
```

上記のこのスケッチは、インタラクティブなスケッチとしてすでにスムーズに機能します。

次に、マウスクリックイベントを追加できます。 このイベントは、 mousePressed() 関数を使用して実装できます。この関数では、ステートメントを追加して、ターゲットポイントを更新し、画面をクリアします。

```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}
```

(次のページに続く)

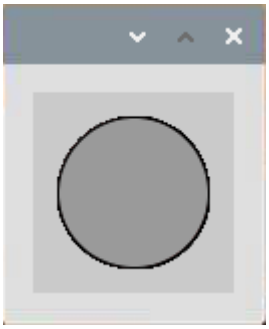
(前のページからの続き)

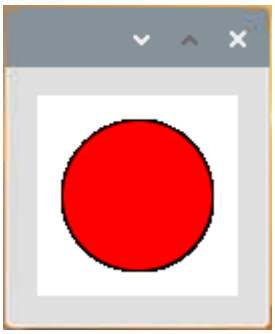
```
void draw() {
    line(pointX, pointY, mouseX, mouseY);
}

void mousePressed() {
    pointX=mouseX;
    pointY=mouseY;
    background(192, 16, 18);
}
```

詳細については Processing Reference を参照してください。

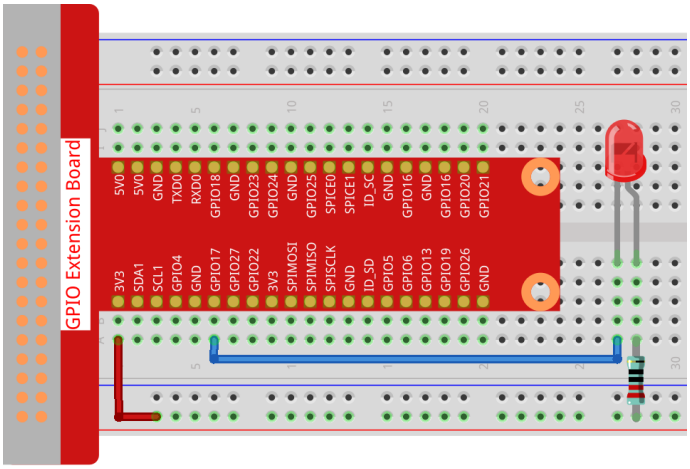
ドットの点滅

このプロジェクトでは、LED と同期して点滅する Processing にドットを描画します。図に示すように回路を構築し、スケッチを実行してください。





回路



スケッチ

```
import processing.io.*;
int ledPin = 17;
boolean state = true;

void setup() {
    size(100, 100);
    frameRate(2); //set frame rate
    GPIO.pinMode(ledPin, GPIO.OUTPUT); //set the ledPin to output mode
}

void draw() {
    state = !state;
    if (state==true) {
        GPIO.digitalWrite(ledPin, GPIO.LOW); //led on
        fill(255, 0, 0); //set the fill color of led on
    } else {
        GPIO.digitalWrite(ledPin, GPIO.HIGH); //led off
        fill(155); //set the fill color of led off
    }
    ellipse(width/2, height/2, width*0.75, height*0.75);
}
```

原理

スケッチの最初に、回路実験に不可欠な `importprocessing.io.*;` を使用して Processing の GPIO 関数ライブラリを埋め込む必要があります。

フレームレート は、ボードに表示されるビットマップの周波数であり、ヘルツ（Hz）で表されます。つまり、

draw() 関数が呼び出される頻度でもあります。 setup() で、 **フレームレート** を 2 に設定すると、0.5 秒ごとに draw() が呼び出されます。

draw() 関数の各呼び出しは、 state の逆を取り、その後それを決定します。値が true の場合、LED が点灯し、ブラシが赤で塗りつぶされます。そうでない場合は、LED がオフになり、ブラシが灰色で塗りつぶされます。

判定が終わったら、 ellipse() 関数を使って円を描きます。 width と height は、表示ウィンドウの幅と高さを格納するために使用されるシステム変数であることに注意してください。

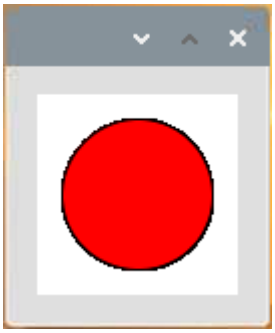
注意すべき他の 2 つのポイントがあります。 GPIO を使用する場合は、 GPIO.pinMode() 関数を使用してピンの INPUT / OUTPUT 状態を設定してから、 GPIO.digitalWrite() 関数を使用して値 (HIGH/LOW) をピンに割り当てる必要があります。

注釈: draw() で delay() を使用しないようにしてください。これは、表示ウィンドウの更新に影響します。

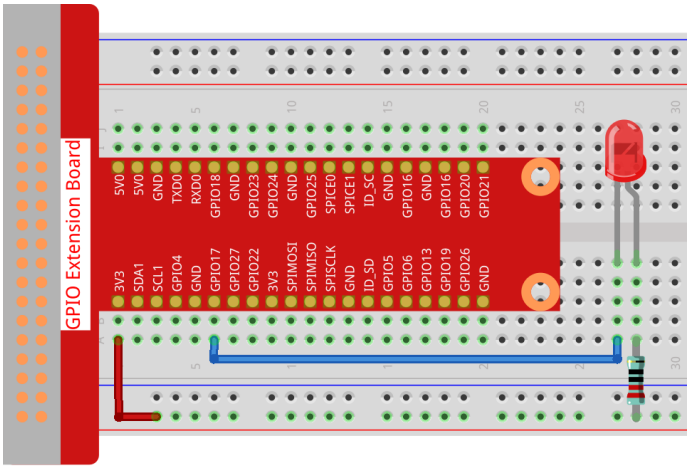
詳細については、 Processing Reference を参照してください。

クリック可能なドット

このプロジェクトでは、LED と同期して点滅する Processing にドットを描画します。図に示すように回路を構築し、スケッチを実行してください。



回路



スケッチ

```
import processing.io.*;
boolean state = false;
int ledPin = 17;

void setup() {
    GPIO.pinMode(ledPin, GPIO.OUTPUT);
    background(255);
}

void draw() {
    if (state == true) {
        GPIO.digitalWrite(ledPin, GPIO.LOW);
        fill(255, 0, 0);
    }else {
        GPIO.digitalWrite(ledPin, GPIO.HIGH);
        fill(155);
    }
    ellipse(width/2, height/2, width*0.75, height*0.75);
}

void mouseClicked() {
    //  toggles state:
    if (2*dist(mouseX,mouseY,width/2, height/2)<=width*0.75)
        {state = !state;}
}
```

原理

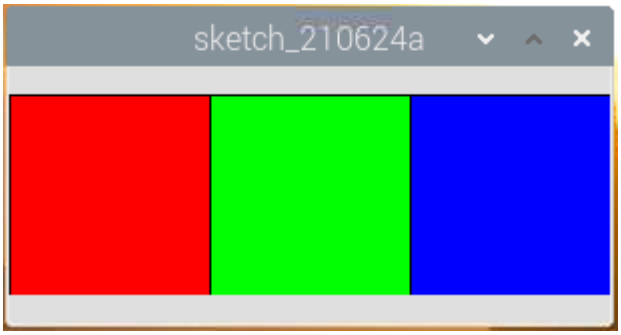
このプロジェクトには **ドットの点滅** と多くの共通点がありますが、違いは、マウスイベントにトグル状態を設定することです。 これにより、LED は自動的に点滅しませんが、マウスをクリックすると点灯および消灯します。

また、 `mouseClicked()` イベントでは、 `dist()` 関数を使用してクリック時のマウスの位置を決定します、マウスとドットの中心との距離が半径より小さい場合にのみ、ドットがクリックされたと見なされます。

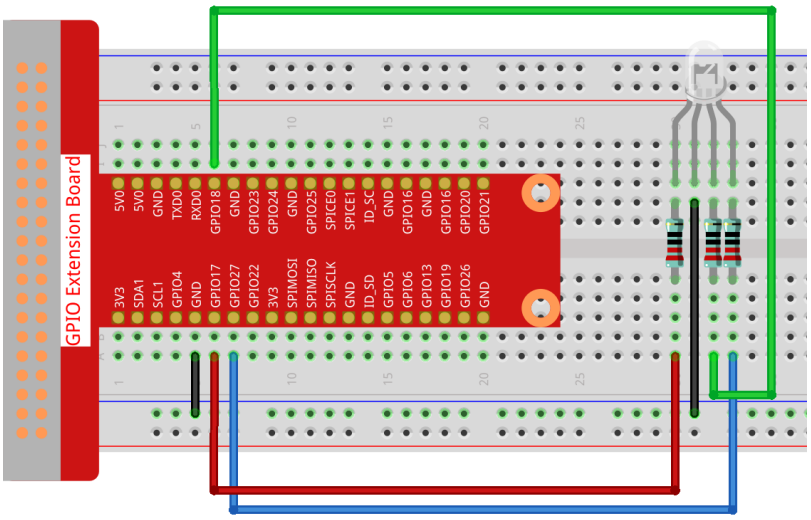
詳細については、 Processing Reference を参照してください。

クリック可能なカラーブロック

クリック可能なドットを描画して LED を制御しようとしたので、さらに一歩進んで 3 色の正方形を描画して RGB カラーを調整しましょう!



回路



スケッチ

```
import processing.io.*; // use the GPIO library

int[] pins = { 17, 18, 27 };
```

(次のページに続く)

(前のページからの続き)

```
void setup() {
    for (int i = 0; i < pins.length; i++) {
        GPIO.pinMode(pins[i], GPIO.OUTPUT);
    }
    size(300, 100);
    background(255);
}

void draw() {
    fill(255, 0, 0);
    rect(0, 0, width/3, height);

    fill(0,255,0);
    rect(width/3, 0, 2*width/3, height);

    fill(0,0,255);
    rect(2*width/3, 0, width, height);
}

void mouseClicked() {
    for (int i = 0; i < pins.length; i++) {
        GPIO.digitalWrite(pins[i],GPIO.LOW);
    }
    if (mouseX<width/3){
        GPIO.digitalWrite(pins[0],GPIO.HIGH);
    }else if (mouseX>width/3&&mouseX<2*width/3){
        GPIO.digitalWrite(pins[1],GPIO.HIGH);
    }else if (mouseX>2*width/3){
        GPIO.digitalWrite(pins[2],GPIO.HIGH);
    }
}
```

原理

このプロジェクトは、マウスクリックイベントを決定するための条件を改善することを除いて、 **クリック可能なドット** と多くの共通点があります。

最初に draw() で 3 つのカラーブロックを描画し、次に mouseX（マウスの X 軸座標）の値に基づいてクリックされたカラーブロックを取得し、最後に RGB に対応する色を点灯させます。

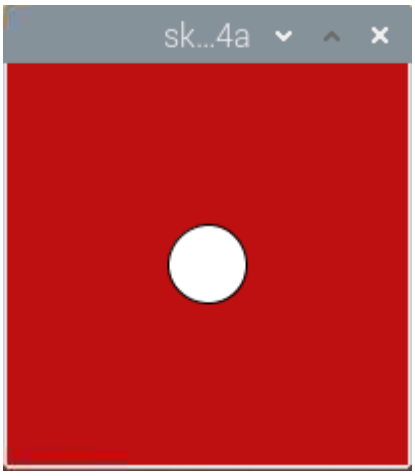
それと?

光の追加に基づいて、RGBLED ディスプレイを 7 色にすることができます。緑に赤を追加すると黄色になります。3 つの原色すべてを足し合わせると白になります。 今、それを自分で試すことができます。

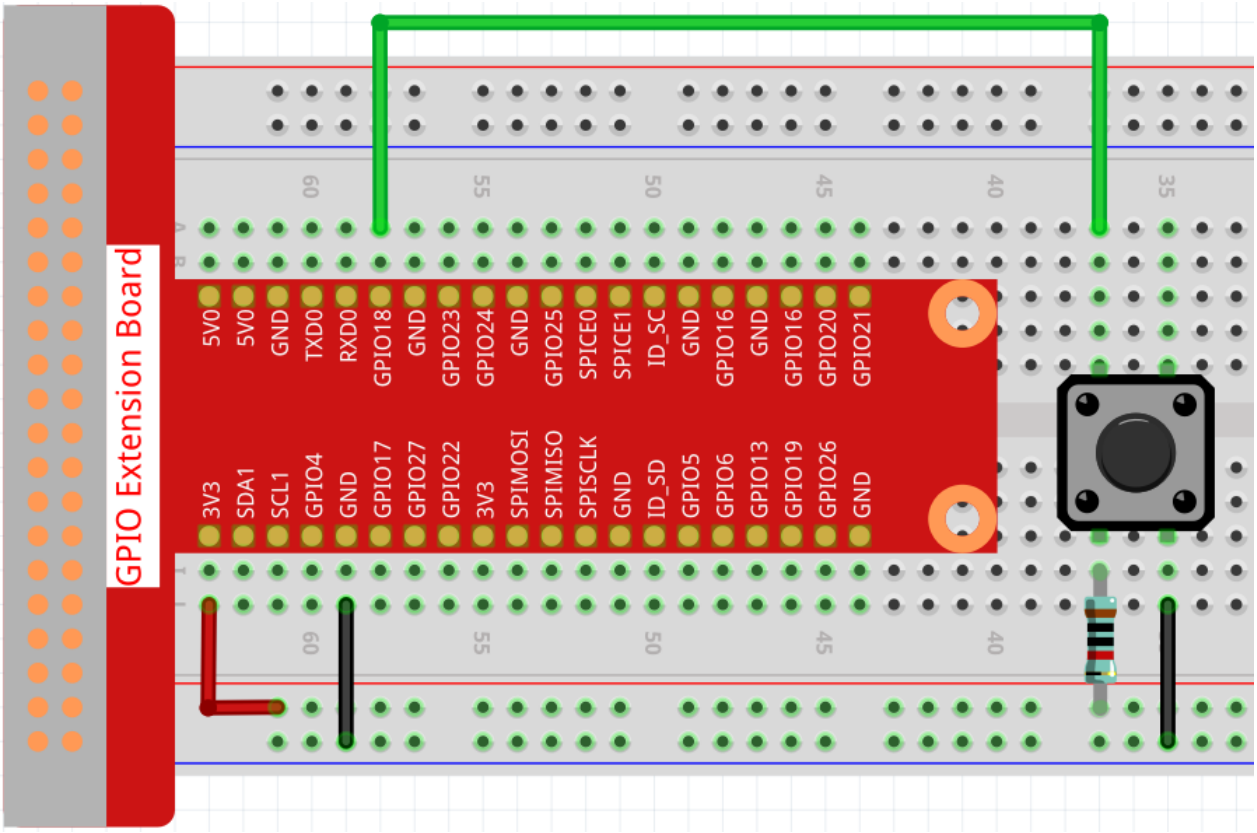
詳細については Processing Reference を参照してください。

ドットを膨らませる

次に、ボタンでドットのサイズを制御できる回路を作成しましょう。ボタンを押すと、ドットがすぐに大きくなります。 ボタンを離すと、ドットが徐々に小さくなり、風船が膨らんでいるように見えます。



回路



スケッチ

```
import processing.io.*;
int buttonPin = 18;

float diameter;

void setup() {
    size(200, 200);
    frameRate(64); //set frame rate
    GPIO.pinMode(buttonPin, GPIO.INPUT_PULLUP);
    diameter = width*0.5;
}

void draw() {
    if (GPIO.digitalRead(buttonPin)==GPIO.LOW) {
        if(diameter<width*0.8) {diameter=diameter+5;}
    } else {
        if(diameter>=width*0.2) {diameter--;}
    }
```

(次のページに続く)

(前のページからの続き)

```
    background(192, 16, 18);
    ellipse(width/2, height/2,diameter, diameter);
}
```

原理

このプロジェクトは、GPIO の出力機能を使用していた前の 2 つのプロジェクトと比較して、入力機能を使用しています。

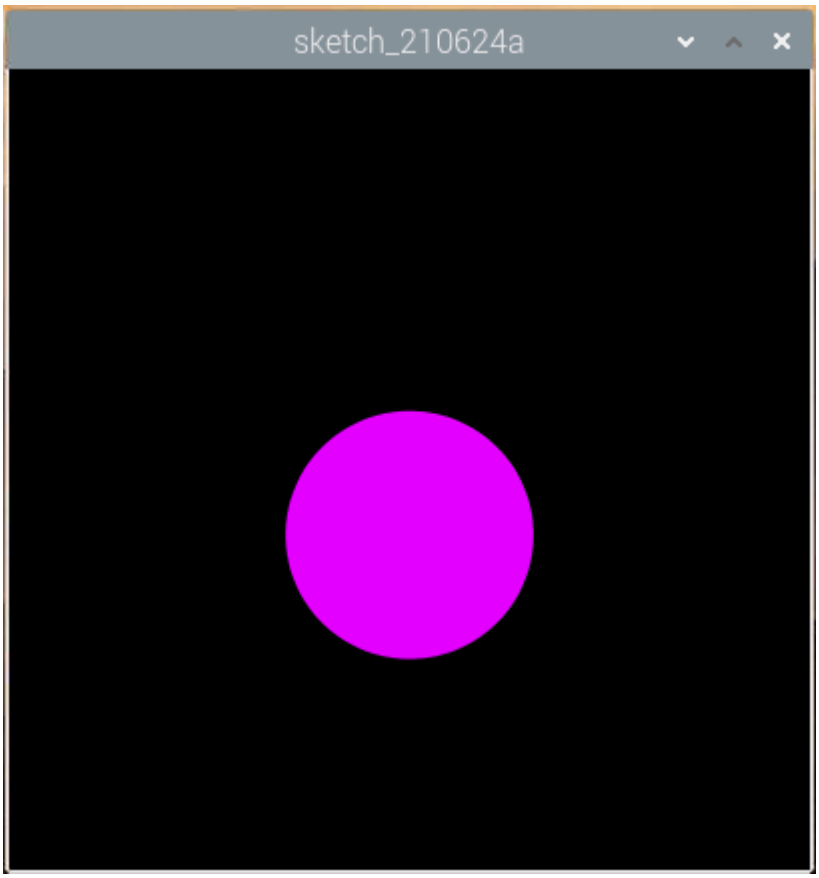
GPIO.pinMode() 関数は、 buttonPin をプルアップ入力モードに設定するために使用されます。これにより、ピンはデフォルト状態で自動的に HIGH になります。

次に、 GPIO.digitalRead() 関数を使用して buttonPin の値を読み取ります。 値が LOW の場合、ボタンが押されていることを意味します。その時点で、ドットの直径が 5 増加します。 ボタンを離すと、ドットの直径が 1 つ小さくなります。

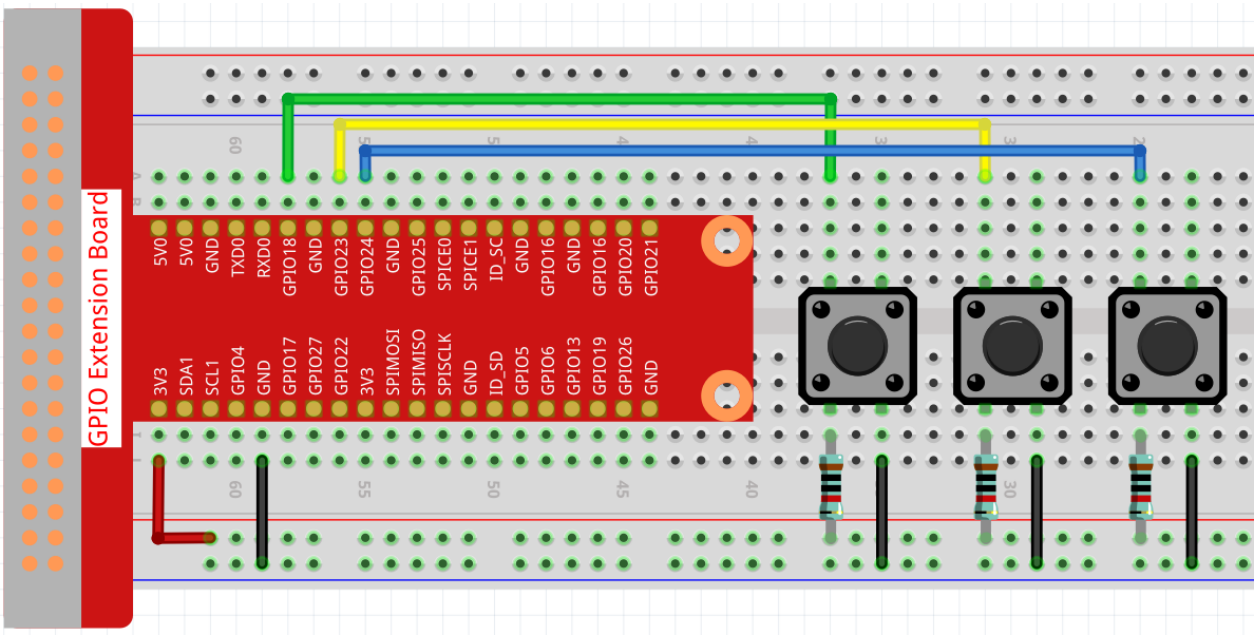
詳細については、 Processing Reference を参照してください。

スイングのドット

このプロジェクトでは、ドットのサイズを変更するためのボタン、位置を変更するためのボタン、色を変更するためのボタンの 3 つのボタンが接続されています。3 つのボタンすべてを同時に押すと、揺れるドットが表示され、色が変化します。


sketch_210624a

回路


GPIO Extension Board
5V0
5V0
GND
TXD0
RXD0
GPIO18
GND
GPIO23
GPIO24
GND
GPIO25
SPICE0
SPICE1
ID_SC
GND
GPIO16
GND
GPIO16
GPIO20
GPIO21
3V3
SDA1
SCL1
GPIO4
GND
GPIO17
GPIO27
GPIO22
3V3
SPIMOSI
SPIMISO
SPISCLK
GND
ID_SD
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
GND

スケッチ

```
import processing.io.*;

// Define an instance of the Dot object
Dot myDot;

// Define the pins that will be reading button presses
int[] pins = { 18, 23, 24 };

void setup() {
    size(400, 400);
    // Change the color mode of the sketch to HSB
    colorMode(HSB, 360, 100, 100);
    noStroke();

    for (int i = 0; i < pins.length; i++) {
        GPIO.pinMode(pins[i], GPIO.INPUT_PULLUP);
    }

    // Create a Dot in the middle of the screen
    myDot = new Dot(width / 2, height / 2, 100, 255);
}

void draw() {
    background(0);

    // Modify attributes of the Dot depending on which buttons are pressed
    if (GPIO.digitalRead(pins[0]) == GPIO.LOW) {myDot.setSize();}
    if (GPIO.digitalRead(pins[1]) == GPIO.LOW) {myDot.setPosition();}
    if (GPIO.digitalRead(pins[2]) == GPIO.LOW) {myDot.setColor();}

    // Update the Dot state
    myDot.update();
    // And draw it to the screen
    myDot.show();
}

class Dot {

    float initX;
```

(次のページに続く)

(前のページからの続き)

```
    float initY;
    float currentX;
    float currentY;
    int positionRange = 60;

    float initSize;
    float currentSize;
    int sizeRange = 50;

    int initColor;
    int currentColor;
    int ColorRange = 80;

    float timer = 0.0;
    float speed = 0.06;

    Dot(float x, float y, float s, int c) {
        initX = x;
        initY = y;
        currentX = x;
        currentY = y;

        initSize = s;
        currentSize = s;

        initColor = c;
        currentColor = c;
    }

    void setSize() {
        currentSize = initSize + sizeRange * sin( timer );
    }

    void setPosition() {
        currentY = initY + positionRange * cos( timer *2);
    }

    void setColor() {
        currentColor = int(initColor + ColorRange * sin( timer ));
```

(次のページに続く)

(前のページからの続き)

```
    }

    void update() {
        timer += speed;
    }

    void show() {
        fill(currentColor, 100, 100);
        ellipse(currentX, currentY, currentSize, currentSize);
    }
}
```

原理

ドットを直接描画する代わりに、ここで Dot クラスを作成します。 次に、オブジェクトを宣言します。（この場合は myDot ）

これは、複数の同一のプロパティを持つドットを描画する簡単な方法です。 たとえば、このプロジェクトのドットに 3 つの関数（サイズの変更、位置の変更、色の変更）を追加すると、宣言する各ドットは同じ関数になります。同じボタンを使用して同じことを実行することも、異なるボタンを使用して各ドットを個別に制御することもできます。

Class (computer programming) - Wikipedia を使用すると、スケッチが美しく、強力で柔軟になります。

次に、 Dot クラスを詳しく見てみましょう。

```
Dot(float x, float y, float s, int c)
```

宣言では、位置の X 座標値と Y 座標値、サイズ、および色（ここでは、 HSB color mode ）に設定されています）の 4 つのパラメーターを渡す必要があります。

各パラメータは、2 セットの値（初期値と現在の値）に割り当てられます。

```
float initX;
float initY;
float currentX;
float currentY;
int positionRange = 60;

float initSize;
float currentSize;
int sizeRange = 50;
```

(次のページに続く)

(前のページからの続き)

```
int initColor;
int currentColor;
int ColorRange = 80;
```

初期値と現在の値に加えて、範囲値のセットもあります。 初期値がドットの初期状態（入力パラメータによって決定される）を決定するために使用され、現在の値が範囲内で変化してドットを移動させることを意味します。

したがって、X 座標値を除いて、他の 3 つのパラメーターの現在の値は次のように計算されます:

```
void setSize() {
    currentSize = initSize + sizeRange * sin( timer );
}

void setPosition() {
    currentY = initY + positionRange * cos( timer *2);
}

void setColor() {
    currentColor = int(initColor + ColorRange * sin( timer ));
}
```

三角関数に精通している場合は、正弦と余弦 sine & cosine を理解するのは難しくありません。これにより、ドットの現在の値が滑らかに周期的に変化します（-1 から 1）。

また、周期的な変動のためにシード、 `timer` を追加する必要があります。 これは、メソッド `update()` に固定値を追加し、 `draw()` で呼び出されます。

```
void update() {
    timer += speed;
}
```

最後に、 `draw()` でも呼び出される `show()` メソッドを使用して、現在の値に従ってドットが表示されます。

```
void show() {
    fill(currentColor, 100, 100);
    ellipse(currentX, currentY, currentSize, currentSize);
}
```

それと？

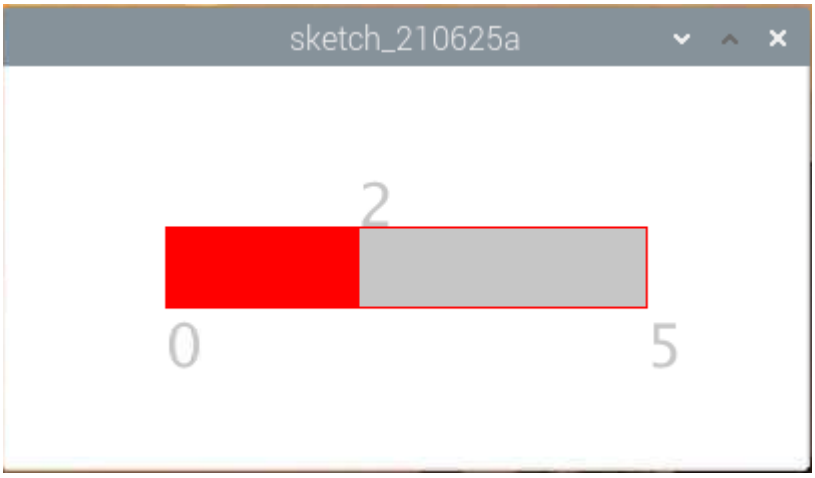
クラスの使い方をマスターすれば、同じプロパティで複数のドットを描くことができるので、もっとクールなこと

をしてみませんか。 たとえば、安定した連星系を描いたり、'DUET' ゲームを作ったりするのはどうですか？

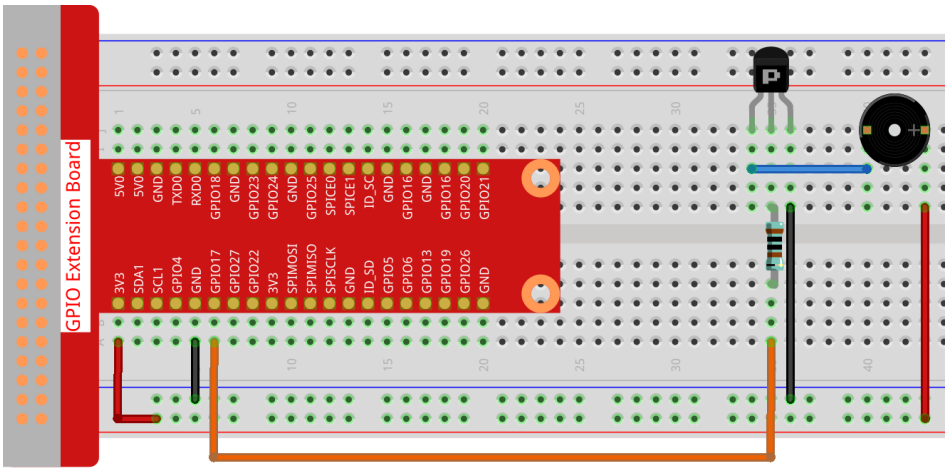
詳細については、 Processing Reference を参照してください。

メトロノーム

ここではメトロノームを作成します。メトロノームは 5 つのレベルに分けられ、レベルが高いほどブザーの呼び出しが緊急になります。



回路



注釈: ここではアクティブブザーが使用されており、白いステッカーが貼られています。

スケッチ

```
import processing.io.*;
```

(次のページに続く)

(前のページからの続き)

```
int level = 0;
int buzzerPin = 17;
int levelRange=5;
Slider mySlider;

void setup() {
    size(400, 200);
    frameRate(50);
    mySlider = new Slider(width * 0.2,height * 0.4,width * 0.8,height * 0.6,0,levelRange,
↪level);
    GPIO.pinMode(buzzerPin, GPIO.OUTPUT);
}

void draw() {

    background(255);
    mySlider.show();
    if(level==0){
        GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    }else if((frameCount/5) % (levelRange-level+1) ==0){
        GPIO.digitalWrite(buzzerPin, GPIO.LOW);
    }else{
        GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    }
}

void mouseDragged(){
    level = mySlider.dragPoint(mouseX,mouseY);
}

class Slider{
    float slotPointAX;
    float slotPointBX;
    float slotPointAY;
    float slotPointBY;
    float linePoint;
    float depth;
    int maxRange;
    int minRange;
```

(次のページに続く)

(前のページからの続き)

```
    int value;

    Slider(float ax, float ay, float bx, float by, int min, int max, int v){
        slotPointAX = ax;
        slotPointAY = ay;
        slotPointBX = bx;
        slotPointBY = by;
        maxRange = max;
        minRange = min;
        value = v;
        linePoint = slotPointAX;// + map(value, minRange, maxRange, slotPointAX,␣
↪slotPointBX);
        depth = (slotPointBY - slotPointAY)*0.75;
    }

    void show(){
        rectMode(CORNERS);
        fill(200);
        stroke(255,0,0);
        rect(slotPointAX, slotPointAY, slotPointBX, slotPointBY);
        fill(255,0,0);
        rect(slotPointAX, slotPointAY, linePoint, slotPointBY);
        fill(200);
        textSize(depth);
        text(minRange, slotPointAX, slotPointBY+depth);
        text(maxRange, slotPointBX, slotPointBY+depth);
        text(value, linePoint, slotPointAY);
    }

    int dragPoint(float mx, float my){
        if(mx>=slotPointAX && mx<=slotPointBX && my>=slotPointAY && my<=slotPointBY){
            value = int(map(mx,slotPointAX,slotPointBX,minRange,maxRange));
            linePoint = map(value,minRange,maxRange,slotPointAX,slotPointBX);
        }
        return value;
    }
}
```

原理

ここでは、 Slider クラスを作成し、それを widget として機能させました。

```
Slider(ax, ay, bx, by, min, max, v)
```

宣言では、7 つのパラメーターで渡す必要があります。

最初の 4 つのパラメーターは widget のサイズを決定し、その後に左上隅の開始点の座標（x1、y1）と右下隅の（x2、y2）が続きます。

最後の 3 つのパラメーターは、その数値範囲（最小から最大）と初期値を決定します。

これには 2 つのメソッドがあります。 dragPoint() の効果は、スライダーをドラッグ可能にして、スライダーの現在の位置の値を返すことです。

```
int dragPoint(float mx, float my){
    if(mx>=slotPointAX && mx<=slotPointBX && my>=slotPointAY && my<=slotPointBY){
        value = int(map(mx,slotPointAX,slotPointBX,minRange,maxRange));
        linePoint = map(value,minRange,maxRange,slotPointAX,slotPointBX);
    }
    return value;
}
```

別のメソッド show() は、スライダーを表示することです。 同時に、範囲値と現在値が対応する位置に表示されます。

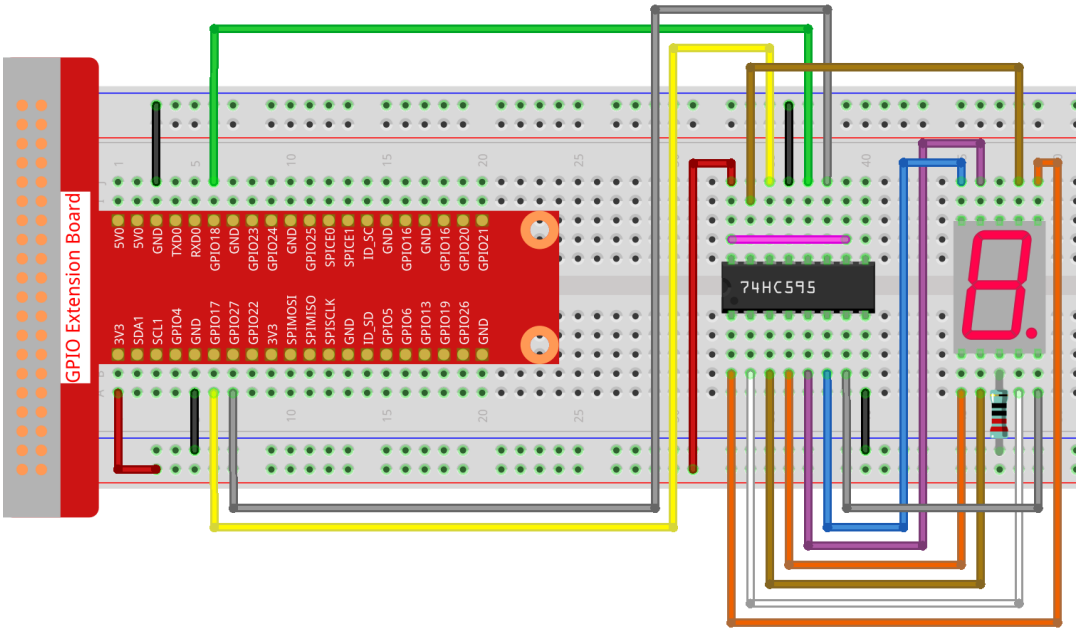
```
void show(){
    rectMode(CORNERS);
    fill(200);
    stroke(255,0,0);
    rect(slotPointAX, slotPointAY, slotPointBX, slotPointBY);
    fill(255,0,0);
    rect(slotPointAX, slotPointAY, linePoint, slotPointBY);
    fill(200);
    textSize(depth);
    text(minRange, slotPointAX, slotPointBY+depth);
    text(maxRange, slotPointBX, slotPointBY+depth);
    text(value, linePoint, slotPointAY);
}
```

詳細については、 Processing Reference を参照してください。

番号を表示

このレッスンでは、processing を使用して 7 セグメントディスプレイを駆動し、0 から 9 および A から F の数値を表示します

回路



スケッチ

```
import processing.io.*;

int SDI=17;   //serial data input
int RCLK=18;  //memory clock input(STCP)
int SRCLK =27;   //shift register clock input(SHCP)


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

void hc595_shift(int dat){
  int i;

  for(i=0;i<8;i++){
        int n=(0x80 & (dat << i));
        if ( n==0){
          GPIO.digitalWrite(SDI, 0);
```

(次のページに続く)

(前のページからの続き)

```
        } else {
          GPIO.digitalWrite(SDI, 1);
        }
        GPIO.digitalWrite(SRCLK, 1);
        delay(1);
        GPIO.digitalWrite(SRCLK, 0);
  }

        GPIO.digitalWrite(RCLK, 1);
        delay(1);
        GPIO.digitalWrite(RCLK, 0);
}

void setup() {
        size(400, 200);
        frameRate(10);

        GPIO.pinMode(SDI, GPIO.OUTPUT);
        GPIO.pinMode(RCLK, GPIO.OUTPUT);
        GPIO.pinMode(SRCLK, GPIO.OUTPUT);

        GPIO.digitalWrite(SDI, 0);
        GPIO.digitalWrite(RCLK, 0);
        GPIO.digitalWrite(SRCLK, 0);

        fill(0,25,88);
        textAlign(CENTER,CENTER);
        textSize(height*0.8);
}

void draw() {

        background(255);
        int number = (frameCount%100)/10;
        text(number, width/2, height/2);
        hc595_shift(SegCode[number]);
}
```

原理

processing.io.* をインポートし、GPIO 関数ライブラリを使用して digital tube pins を制御します。

配列を定義する SegCode = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71} 16 進数中で、0 から F までのセグメントコード配列を表す（共通カソード）

setup() 関数は、3 つのピン SDI、RCLK、SRCLK を出力として設定し、初期データを 0 として設定します。

hc595_shift(int dat) 関数は、 SegCode を 74HC595 にシフトするために使用されます。

```
void hc595_shift(int dat){
  int i;

  for(i=0;i<8;i++){
        int n=(0x80 & (dat << i));
        if ( n==0){
          GPIO.digitalWrite(SDI, 0);
        } else {
          GPIO.digitalWrite(SDI, 1);
        }
        GPIO.digitalWrite(SRCLK, 1);
        delay(1);
        GPIO.digitalWrite(SRCLK, 0);
  }

        GPIO.digitalWrite(RCLK, 1);
        delay(1);
        GPIO.digitalWrite(RCLK, 0);
}
```

n=(0x80 & (dat << i)) は、dat を i ビット左にシフトしてから、0x80 と & 操作を実行することを意味します。

& 演算のルールは、 & の両側が 1 の場合、結果は 1 であり、それ以外の場合、結果は 0 です。

たとえば、 dat=0x3f,i=2 (0011 1111 << 2 shift to 1111 1100) , then 1111 1100 & 1000 0000 (0x80) = 1000 0000 。

最後に、 dat データをビット単位で SDI（DS）に割り当てます。

digitalWrite(SRCLK, 1) SRCLK が 0 から 1 までの立ち上がりエッジパルスを生成すると、データは DS レジスタからシフトレジスタに転送されます。

digitalWrite(RCLK, 1) RCLK が 0 から 1 までの立ち上がりエッジパルスを生成すると、データはシフトレジスタからストレージレジスタに転送されます。

```
fill(0,25,88);
textAlign(CENTER,CENTER);
textSize(height*0.8);
```

- fill(): 図形の塗りつぶしに使用する色を設定します。
- textAlign(CENTER,CENTER): テキストを描画するための現在の配置を設定します。 パラメータ LEFT 、CENTER 、および RIGHT は、 text() 関数の x および y パラメータの値に関連して文字の表示特性を設定します。
- textSize(): 現在のフォントサイズを設定します。 このサイズは、以降の text() 関数のすべての呼び出しで使用されます。 フォントサイズはピクセル単位で測定されます。

これらの関数は、processing で表示されるテキストスタイルをカスタマイズできます。

```
void draw() {

        background(255);
        int number = (frameCount%100)/10;
        text(number, width/2, height/2);
        hc595_shift(SegCode[number]);
}
```

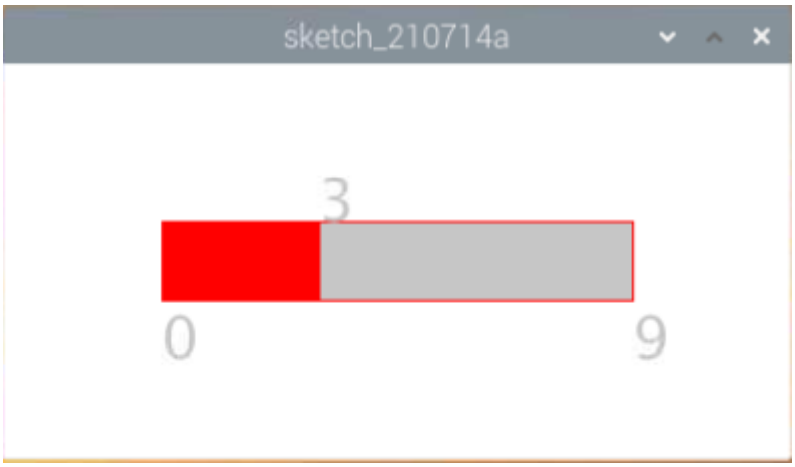
frameCount はシードであり、 frameRate に関連しています。 デフォルトでは、 frameRate は 60 です。これは、 frameCount が 1 秒間に 60 回累積することを意味します。

次に、processing と 7 セグメントディスプレイを使用して、0 から 9 までの数字と A から F までの数字を同時に表示できます。

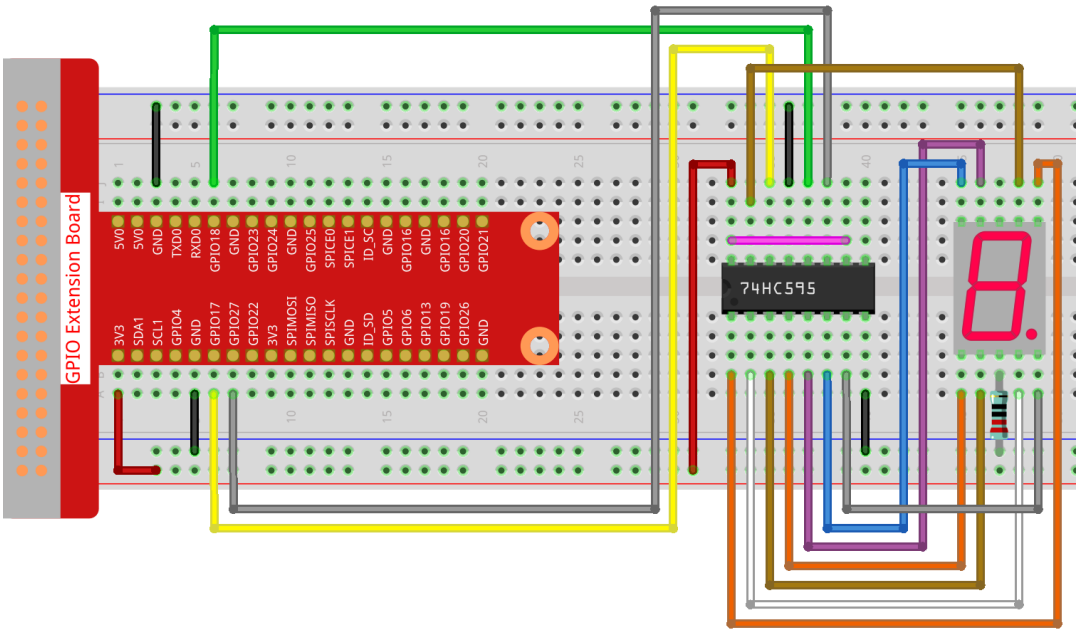
詳細については Processing Reference を参照してください。

ドラッグ番号

7 セグメントディスプレイを制御するためのスライダーバーを描いてみましょう。



回路



スケッチ

```
import processing.io.*;

int number = 0;
int levelRange=9;
Slider mySlider;

int SDI=17;   //serial data input
```

(次のページに続く)

(前のページからの続き)

```
int RCLK=18;  //memory clock input(STCP)
int SRCLK =27;   //shift register clock input(SHCP)


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

void hc595_shift(int dat){
int i;

for(i=0;i<8;i++){
    int n=(0x80 & (dat << i));
    if ( n==0){
    GPIO.digitalWrite(SDI, 0);
    } else {
    GPIO.digitalWrite(SDI, 1);
    }
    GPIO.digitalWrite(SRCLK, 1);
    delay(1);
    GPIO.digitalWrite(SRCLK, 0);
}

    GPIO.digitalWrite(RCLK, 1);
    delay(1);
    GPIO.digitalWrite(RCLK, 0);
}

void setup() {
    size(400, 200);
    frameRate(50);
    mySlider = new Slider(width * 0.2,height * 0.4,width * 0.8,height * 0.6,0,levelRange,
↪number);
    GPIO.pinMode(SDI, GPIO.OUTPUT);
    GPIO.pinMode(RCLK, GPIO.OUTPUT);
    GPIO.pinMode(SRCLK, GPIO.OUTPUT);

    GPIO.digitalWrite(SDI, 0);
    GPIO.digitalWrite(RCLK, 0);
    GPIO.digitalWrite(SRCLK, 0);
```

(次のページに続く)

(前のページからの続き)

```
}

void draw() {

    background(255);
    mySlider.show();
    hc595_shift(SegCode[number]);
}

void mouseDragged(){
    number = mySlider.dragPoint(mouseX,mouseY);
}



class Slider{
    float slotPointAX;
    float slotPointBX;
    float slotPointAY;
    float slotPointBY;
    float linePoint;
    float depth;
    int maxRange;
    int minRange;
    int value;

    Slider(float ax, float ay, float bx, float by, int min, int max, int v){
        slotPointAX = ax;
        slotPointAY = ay;
        slotPointBX = bx;
        slotPointBY = by;
        maxRange = max;
        minRange = min;
        value = v;
        linePoint = slotPointAX;// + map(value, minRange, maxRange, slotPointAX,␣
↪slotPointBX);
        depth = (slotPointBY - slotPointAY)*0.75;
    }
```

(次のページに続く)

(前のページからの続き)

```
    void show(){
        rectMode(CORNERS);
        fill(200);
        stroke(255,0,0);
        rect(slotPointAX, slotPointAY, slotPointBX, slotPointBY);
        fill(255,0,0);
        rect(slotPointAX, slotPointAY, linePoint, slotPointBY);
        fill(200);
        textSize(depth);
        text(minRange, slotPointAX, slotPointBY+depth);
        text(maxRange, slotPointBX, slotPointBY+depth);
        text(value, linePoint, slotPointAY);
    }

    int dragPoint(float mx, float my){
        if(mx>=slotPointAX && mx<=slotPointBX && my>=slotPointAY && my<=slotPointBY){
            value = int(map(mx,slotPointAX,slotPointBX,minRange,maxRange));
            linePoint = map(value,minRange,maxRange,slotPointAX,slotPointBX);
        }
        return value;
    }
}
```

原理

このプロジェクトは、前のコースの「スライダー」と「7 セグメントディスプレイ」を統合します。 具体的な知識のポイントについては、**番号を表示** および **メトロノーム** を参照してください。

1.14 Scratch で遊ぶ (Pi 5 用ではない)

Scratch は、コーディングの教育ツールとして主に 8〜16 歳の子供を対象としたブロックベースのビジュアルプログラミング言語および Web サイトです。 サイトのユーザーは、ブロックのようなインターフェイスを使用して Web 上にプロジェクトを作成できます。 このサービスは MIT メディアラボによって開発され、70 以上の言語に翻訳されており、世界のほとんどの地域で使用されています。

ここでは、RaspberryPi で Scratch3 を使用し、オンライン Scratch では不可能な Scratch で Raspberry Pi GPIO にアクセスする方法を学習します。

1.14.1 Scratch に関するクイックガイド

注釈: Scratch 3 でプログラミングする場合、より良いエクスペリエンスのために画面が必要になる場合があります。 もちろん、画面がない場合は、リモートで Raspberry Pi デスクトップにアクセスすることもできます。詳細なチュートリアルについては、 *Raspberry Pi* **の設定** を参照してください。

さらに、Scratch3 を実行するには少なくとも 1GB の RAM が必要です。少なくとも 2GB の RAM を搭載した Raspberry Pi 4 をお勧めします。Scratch3 は Raspberry Pi 2、3、3B +、または 1GBRAM の Raspberry 4 で実行できますが、これらのモデルのパフォーマンスは低になります。その同時に、実行する他のソフトウェアの影響を受けて、メモリ不足が起こり、Scratch3 が起動しない場合があります。

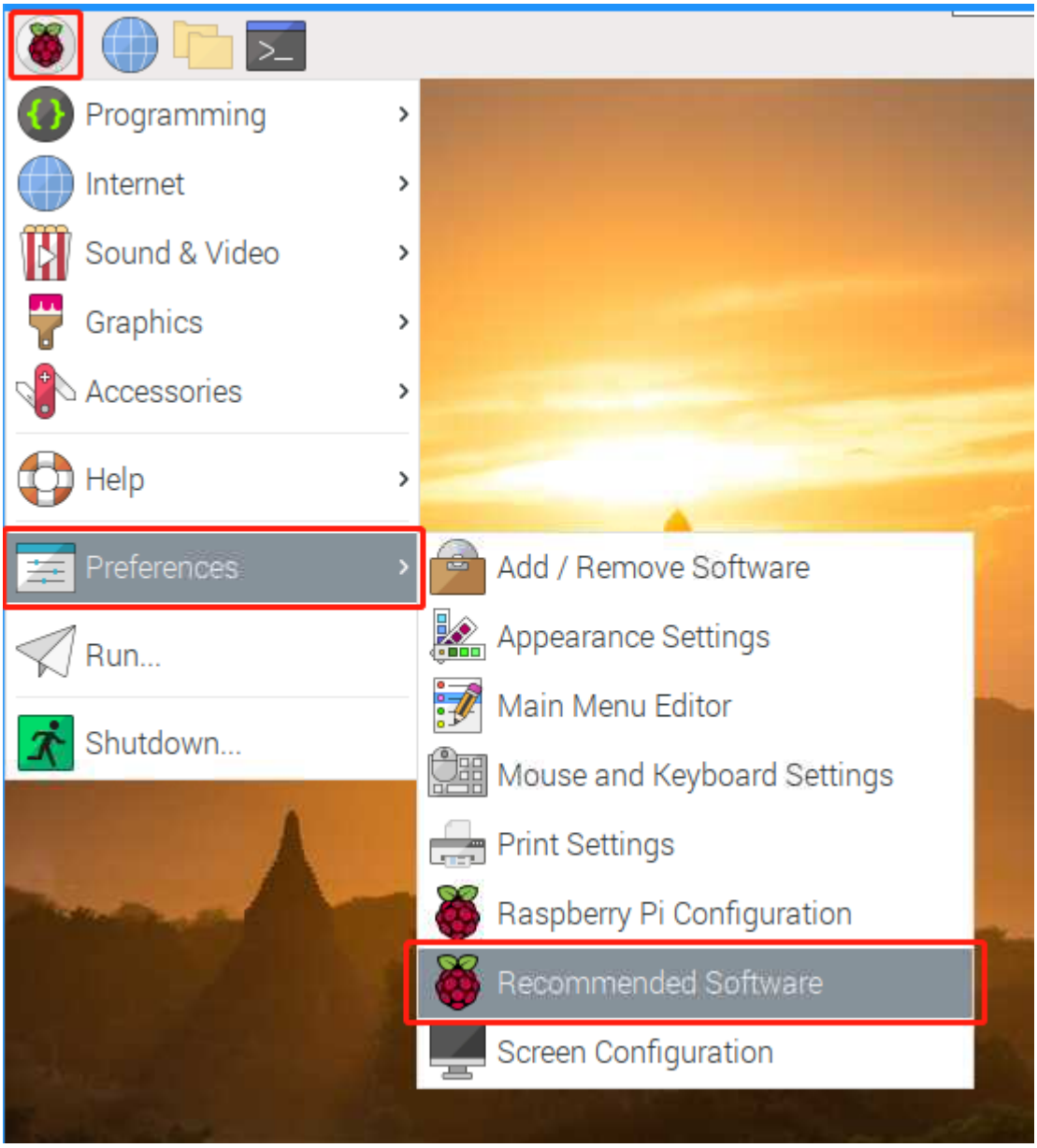
Scratch3 をインストールする

Raspberry pi OS **のインストール** ときは、デスクトップ付きのバージョンを選択する必要があります。デスクトップのみ、またはデスクトップと推奨ソフトウェア付きのも問題ありません。

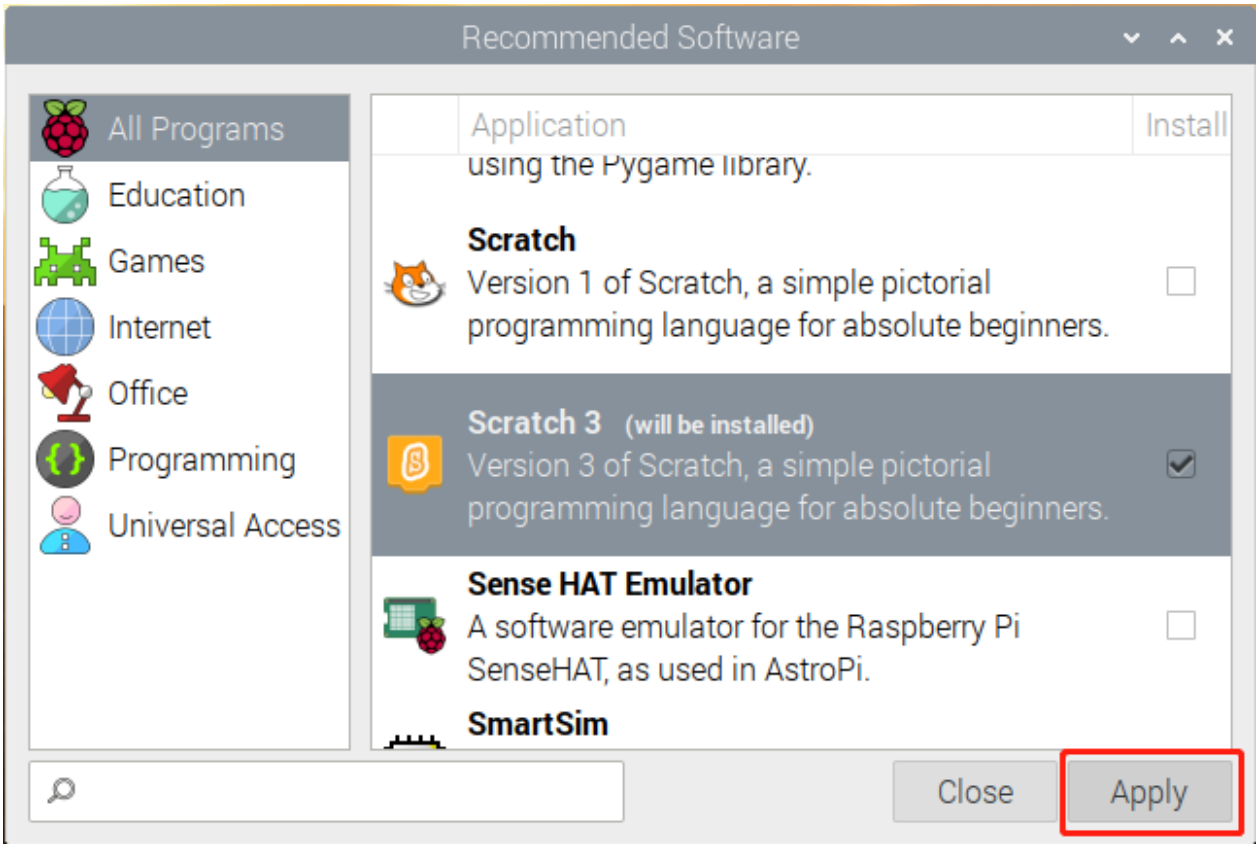
推奨ソフトウェアを使用してバージョンをインストールすると、 **Programming** のシステムメニューに Scratch 3 が表示されます。

デスクトップのみのバージョンをインストールした場合は、以下に説明するように、Scratch を手動でインストールする必要があります。

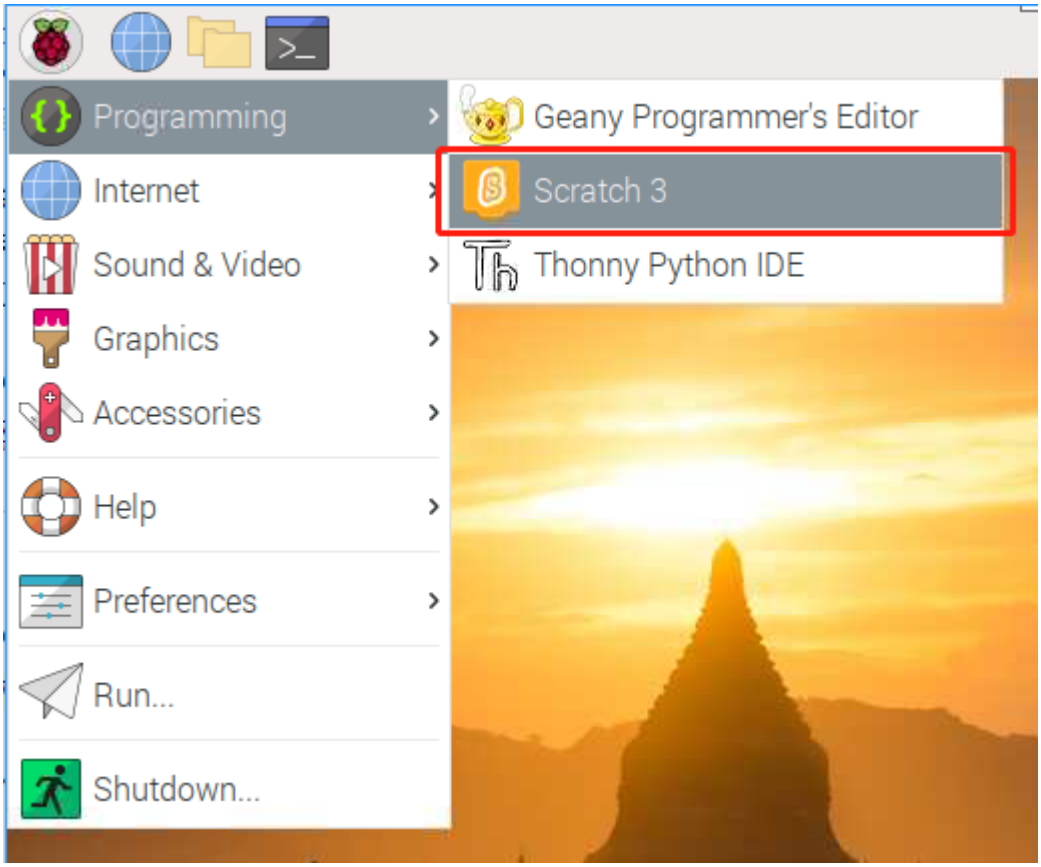
メニューを開き、 **Preferences -> Recommended Software** をクリックします。



Scratch 3 を見つけて確認し、 **Apply** をクリックして、最後にインストールが完了するのを待ちます。



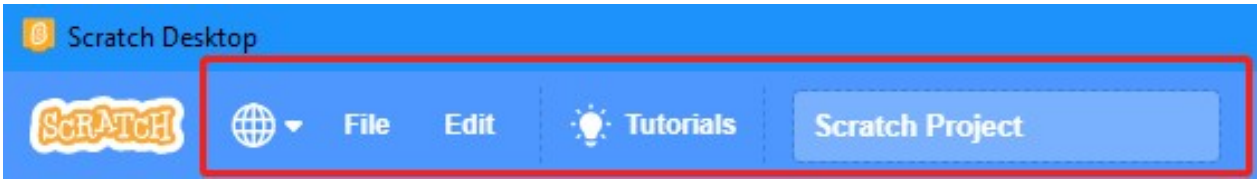
インストールが完了すると、システムメニューの **Programming** に表示されます。



Scratch3 のインターフェースについて

Scratch 3 は、楽しく、教育的で、簡単に習得できるように設計されています。ブロックベースのプログラミングを使用して、インタラクティブなストーリー、ゲーム、アート、シミュレーションなどを作成するためのツールがあります。Scratch には、独自のペイントエディタとサウンドエディタも組み込まれています。

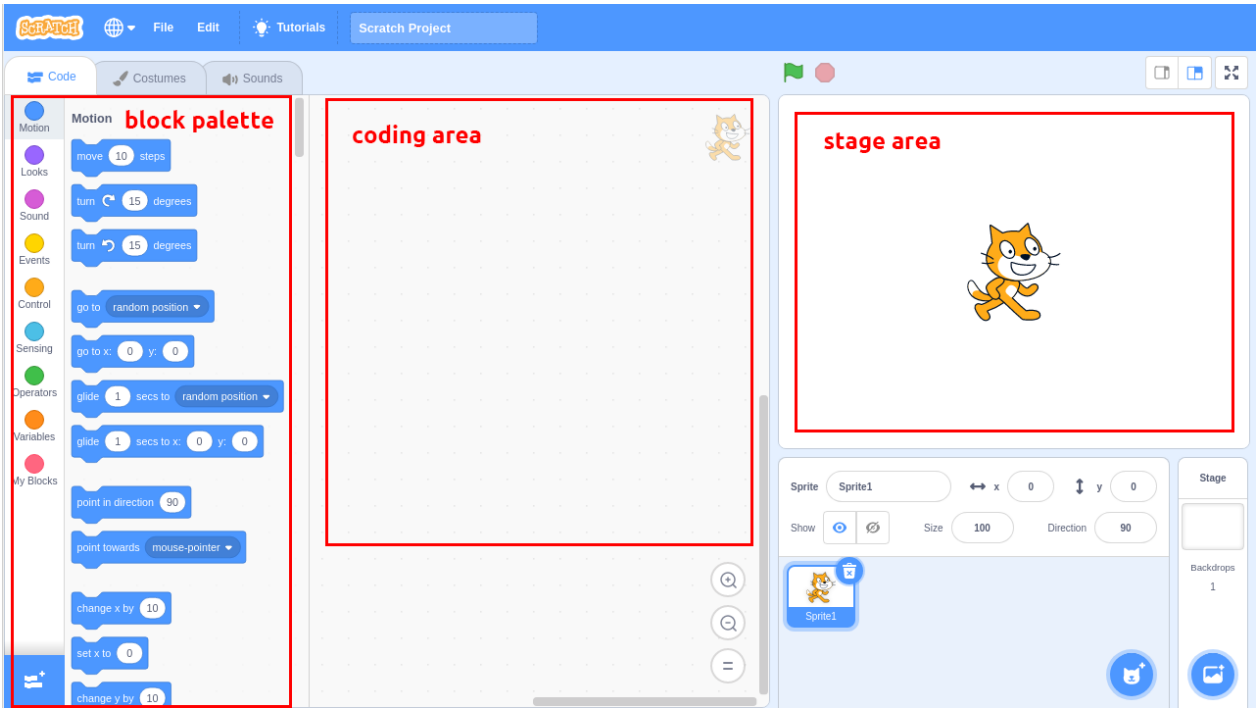
スクラッチ 3 の上部にはいくつかの基本的なオプションがあり、左から右への最初のオプションは言語オプションです。プログラミング用にさまざまな言語を選択できます。 2 つ目は File オプションです。このオプションを使用すると、新しいファイルを作成したり、ローカルファイルを読み取ったり、現在のファイルを保存したりできます。3 つ目は、Edit オプションです。これを使用すると、一部の削除操作を再開して、加速モード（sprites の動きが特に速くなる）を有効にできます。4 つ目は、Tutorials オプションです。これを使用すると、一部のプロジェクトのチュートリアルを表示できます。5 つ目は、プロジェクトの名前を変更できるファイル名付けオプションです。



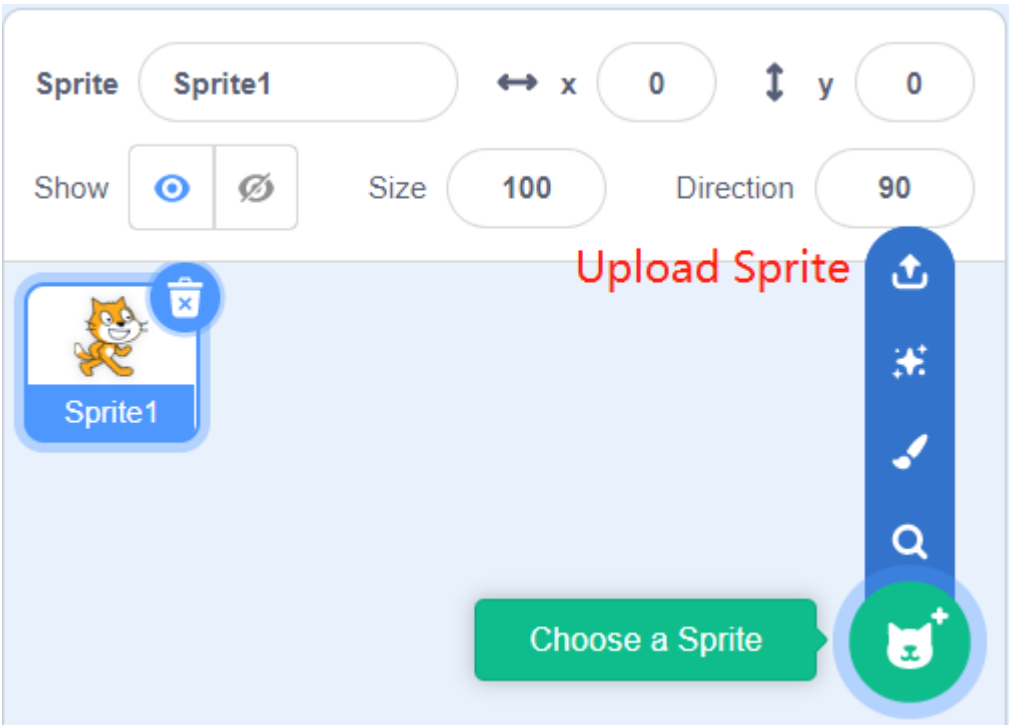
コード

ステージ領域、ブロックパレット、コーディング領域の 3 つの主要なセクションがあります。ブロックパレットの

ブロックをクリックしてコーディング領域にドラッグすることによるプログラミング。最後に、プログラミング結果がステージ領域に表示されます。

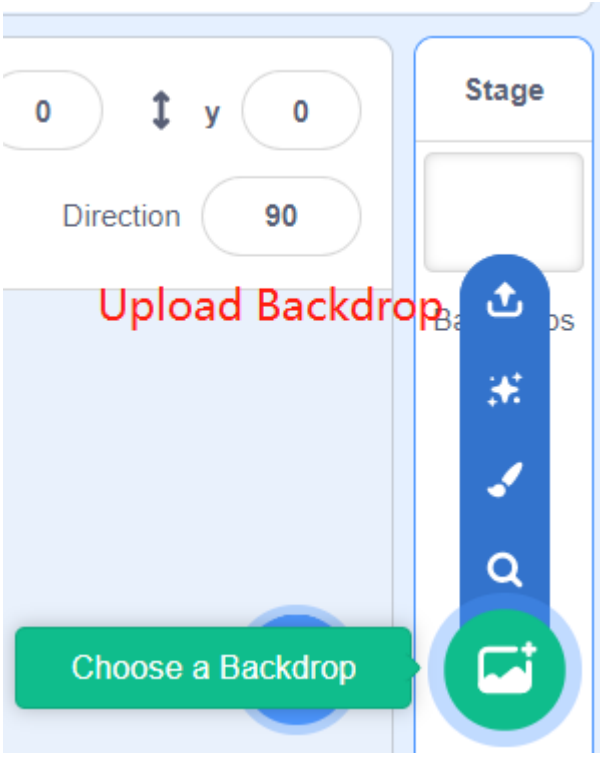


これが Scratch3 の sprites area です。この領域の上に sprites の基本パラメータがあり、Scratch 3 に付属の sprites を追加したり、ローカル sprites をアップロードしたりできます。

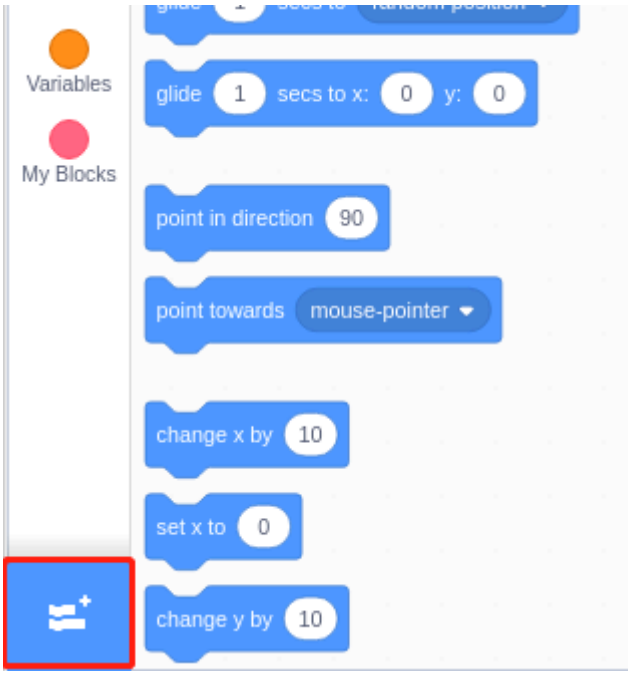


これが Scratch3 の backdrop area です。主にステージに適した backdrop を追加するために、Scratch 3 に付属の

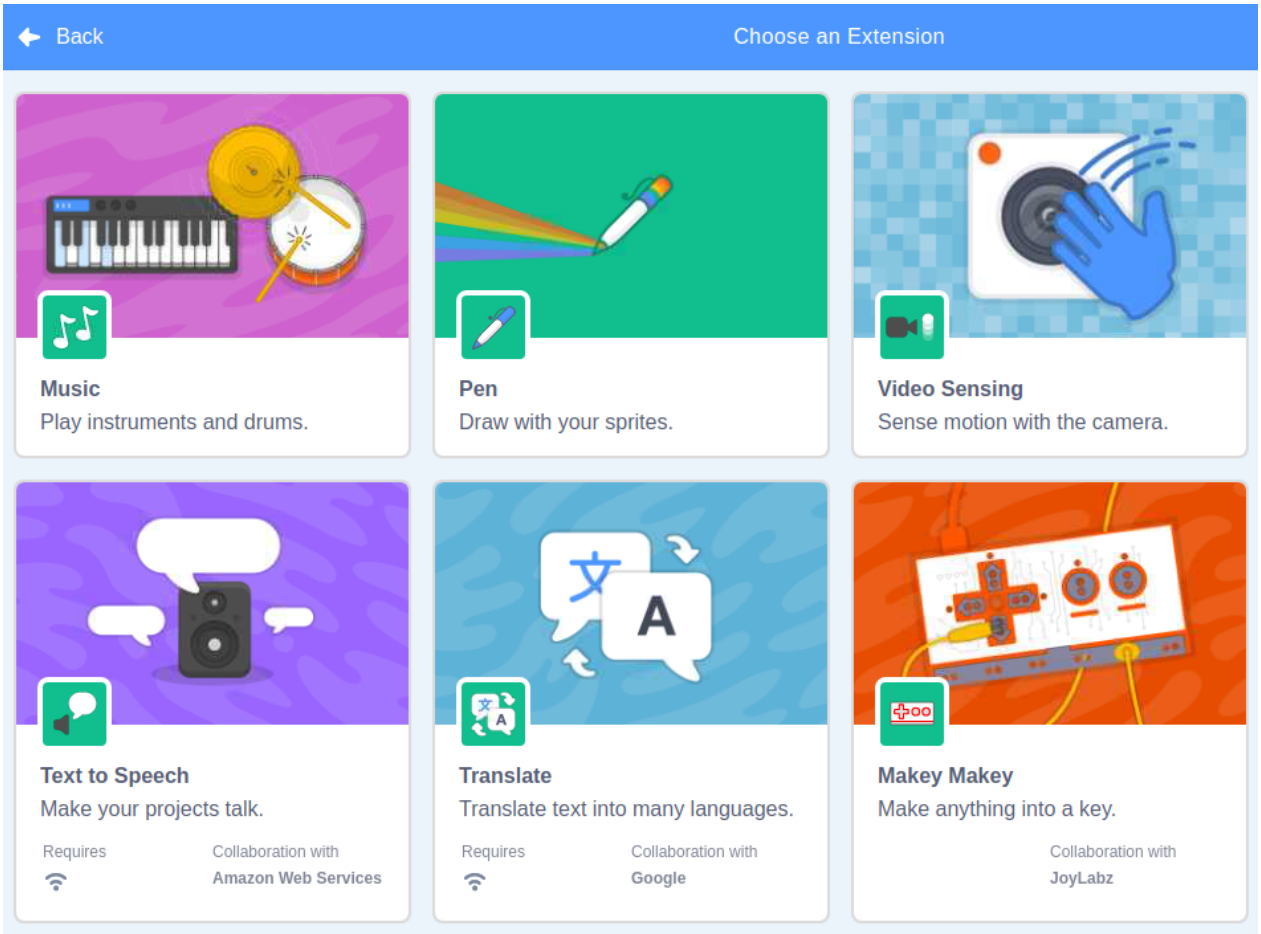
backdrop を追加するか、ローカルのものをアップロードできます。



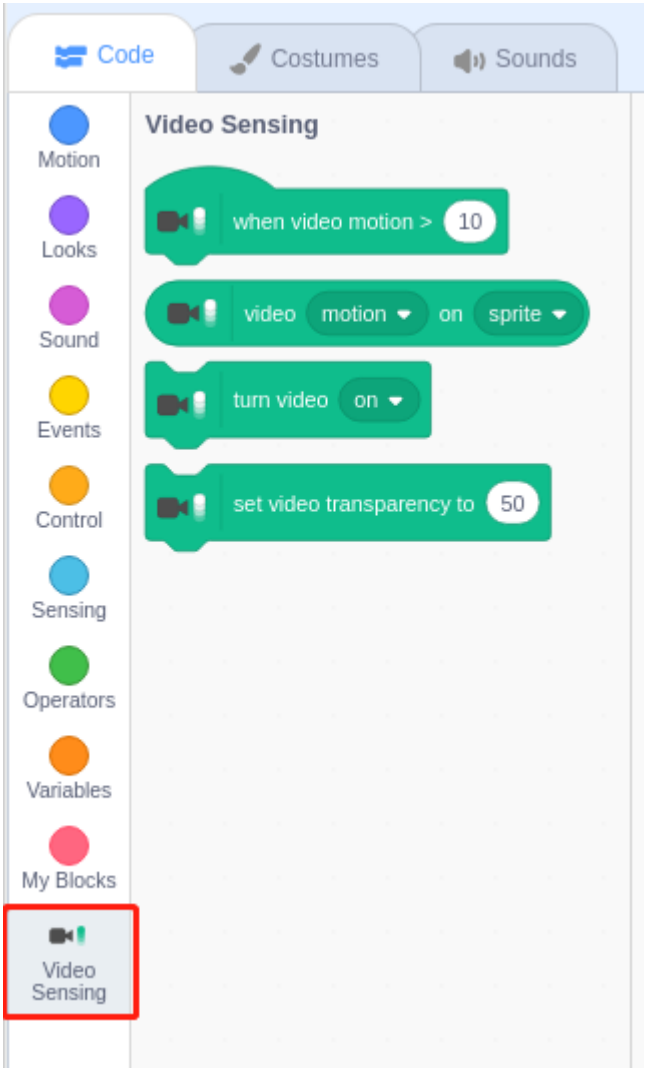
これは **Add Extension** ボタンです。



Scratch 3 では、あらゆる種類の便利な拡張機能を追加できます。ここでは、 **Video Sensing** を例として取り上げ、それをクリックします。

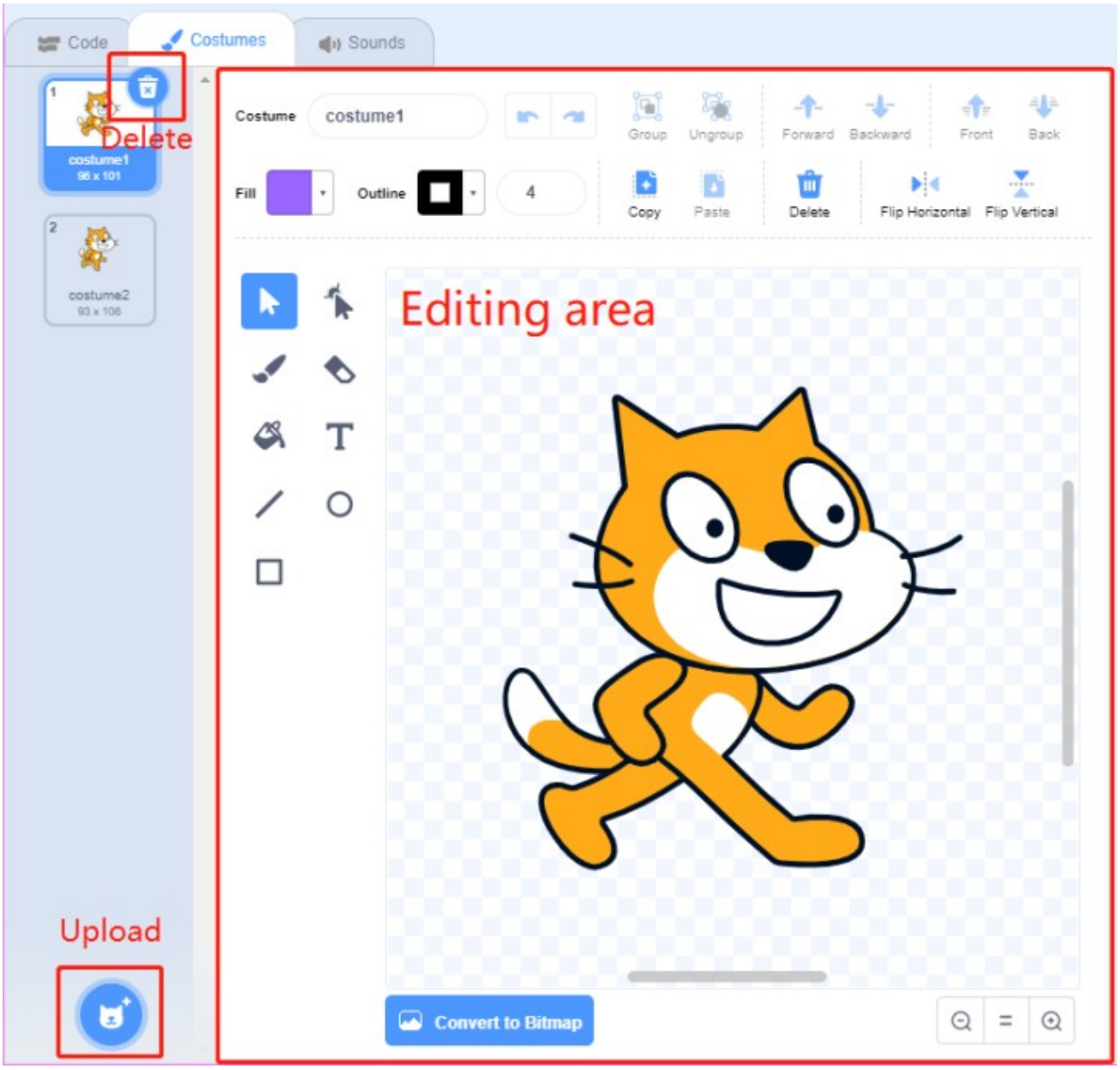


block palette に表示され、この拡張機能に関連付けられている機能を使用できます。カメラを接続している場合は、stage area にカメラ画面が表示されます。



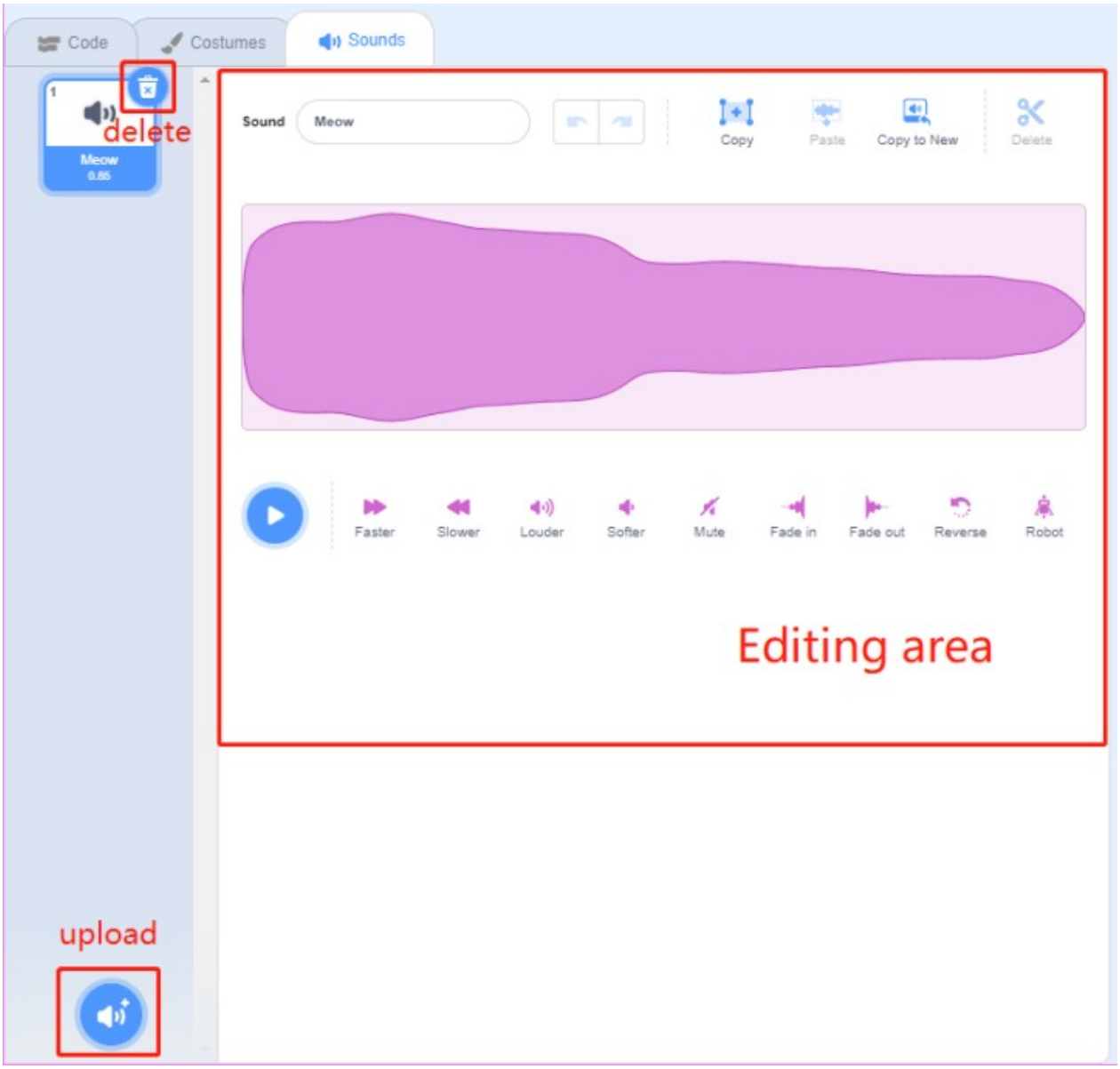
コスチューム

左上隅の **Costumes** オプションをクリックして、costumes palette に入ります。 costumes が異なれば、sprites の静的な動きも異なります。これらの静的な動きをつなぎ合わせると、一貫した動的な動きが形成されます。



音声

実験をより面白くするために、いくつかの音楽クリップを使用する必要があるかもしれません。 左上隅の Sounds オプションをクリックすると、現在のサウンドを編集したり、新しいサウンドを選択/アップロードしたりできます。



1.14.2 プロジェクト

この章では、Scratch3 から 18 のプロジェクトで遊んでいきます。

Scratch 3 を使用したばかりのユーザーの場合は、Scratch 3 をすぐに使い始めることができるように、プロジェクトを順番に試すことをお勧めします。

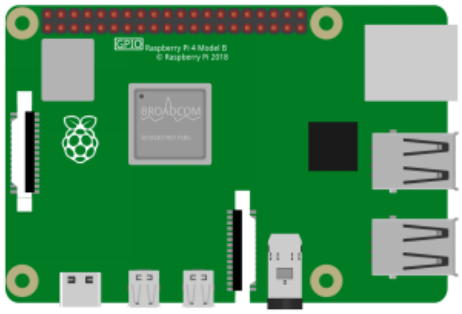
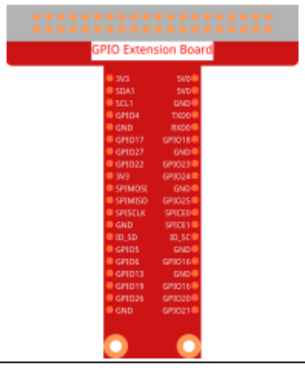
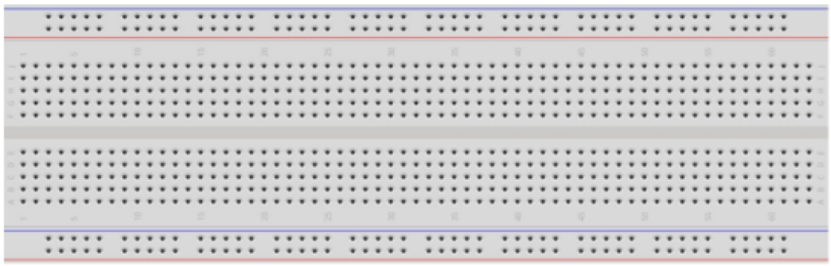
注釈: プロジェクトを試す前に、関連する資料とコードファイルをダウンロードしておく必要があります。 ターミナルを開き、次のコマンドを入力して github からダウンロードします。

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

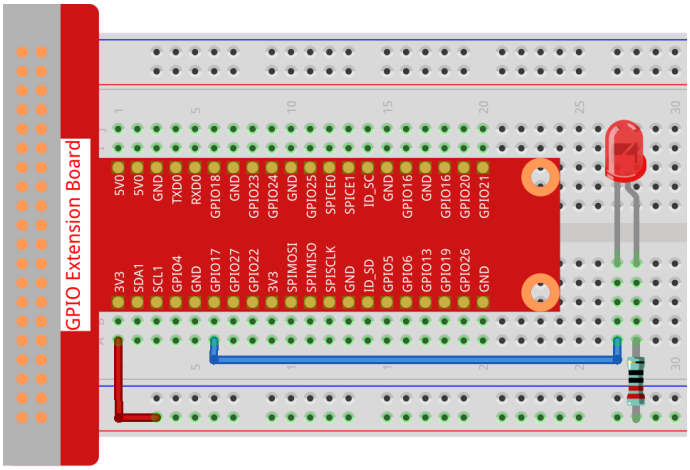
1.1 魔法の杖

今日は、LED、Raspberry Pi、Scratch を使って楽しいゲームを作ります。 魔法の杖を振ると、LED が点滅します。

必要な部品

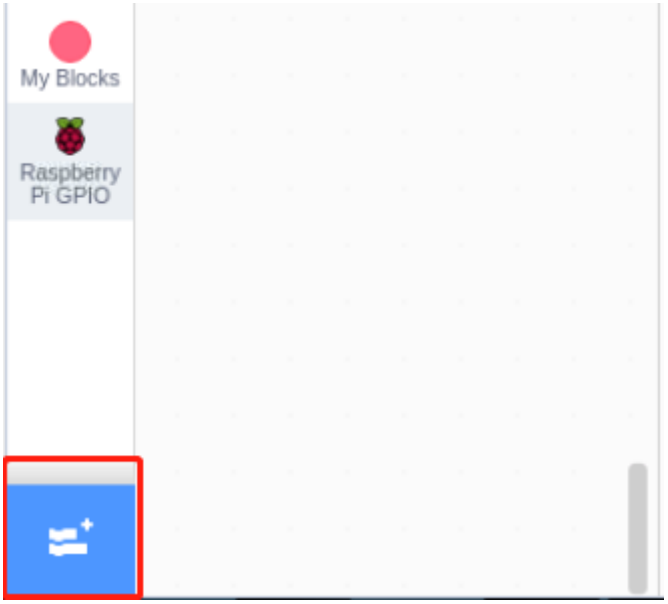
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor(220Ω)

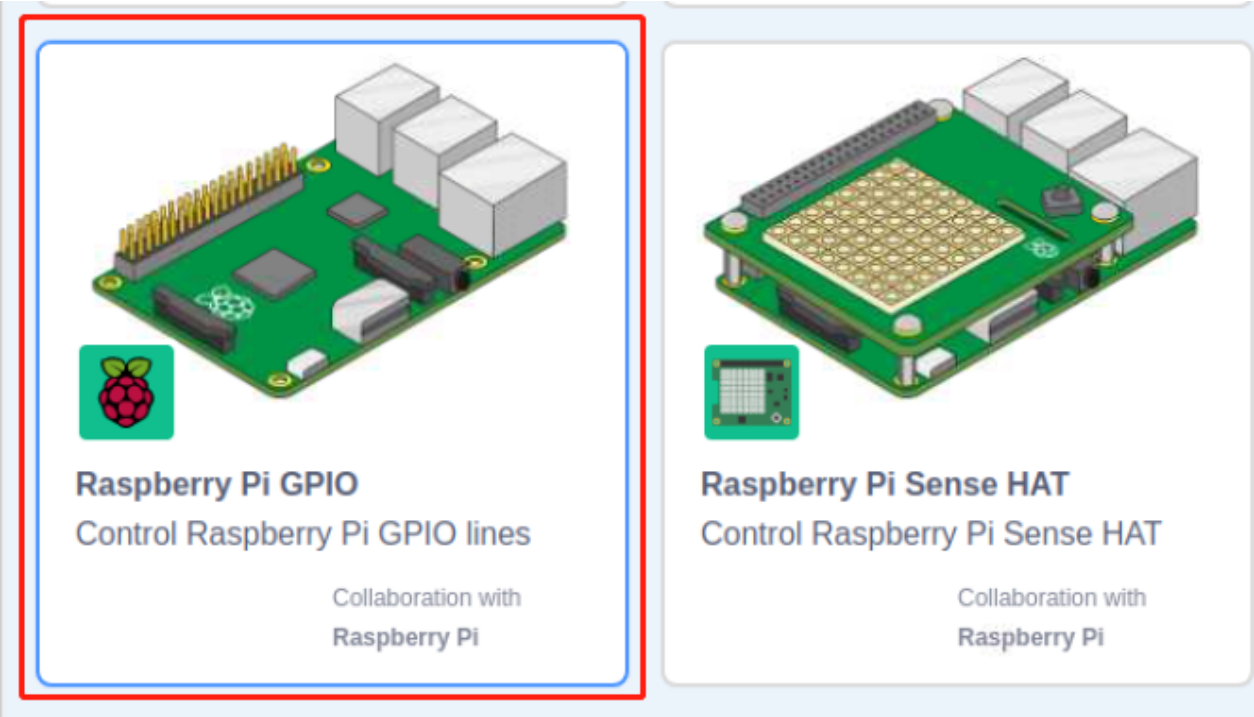
回路を構築する

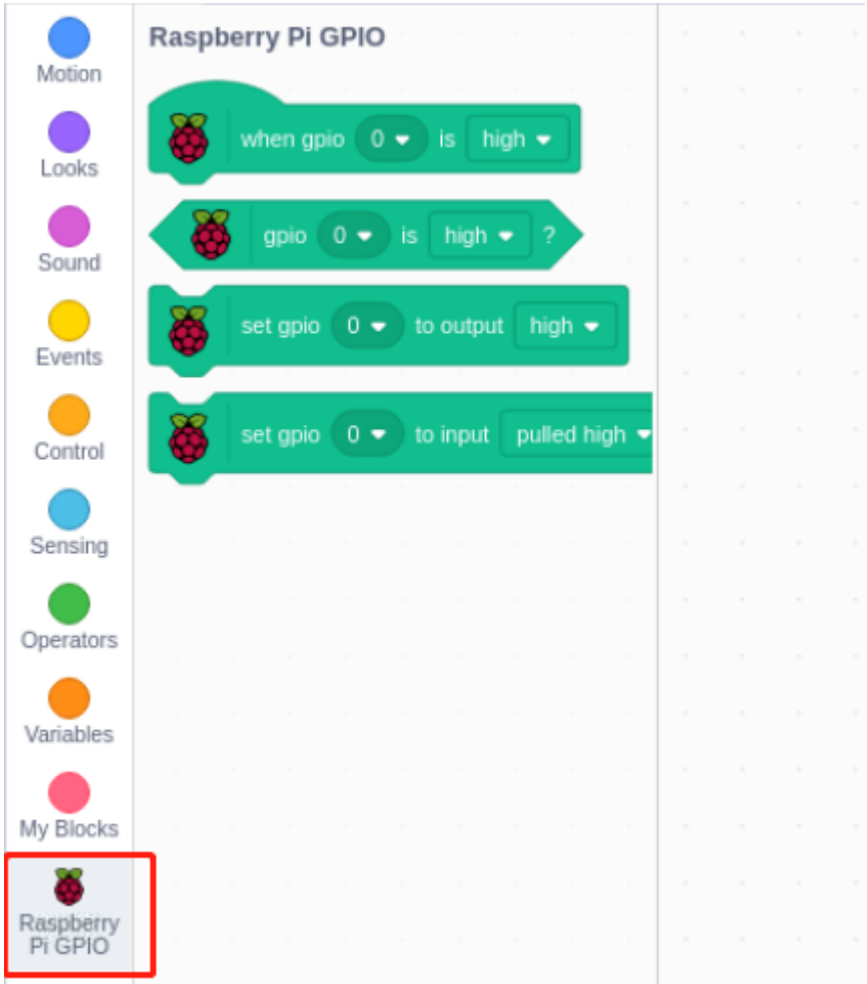


GPIO 拡張機能を追加する

左下隅にある **Add Extension** ボタンをクリックしてから、すべての Scratch プロジェクトに使用する拡張機能である **Raspberry Pi GPIO** を追加します。

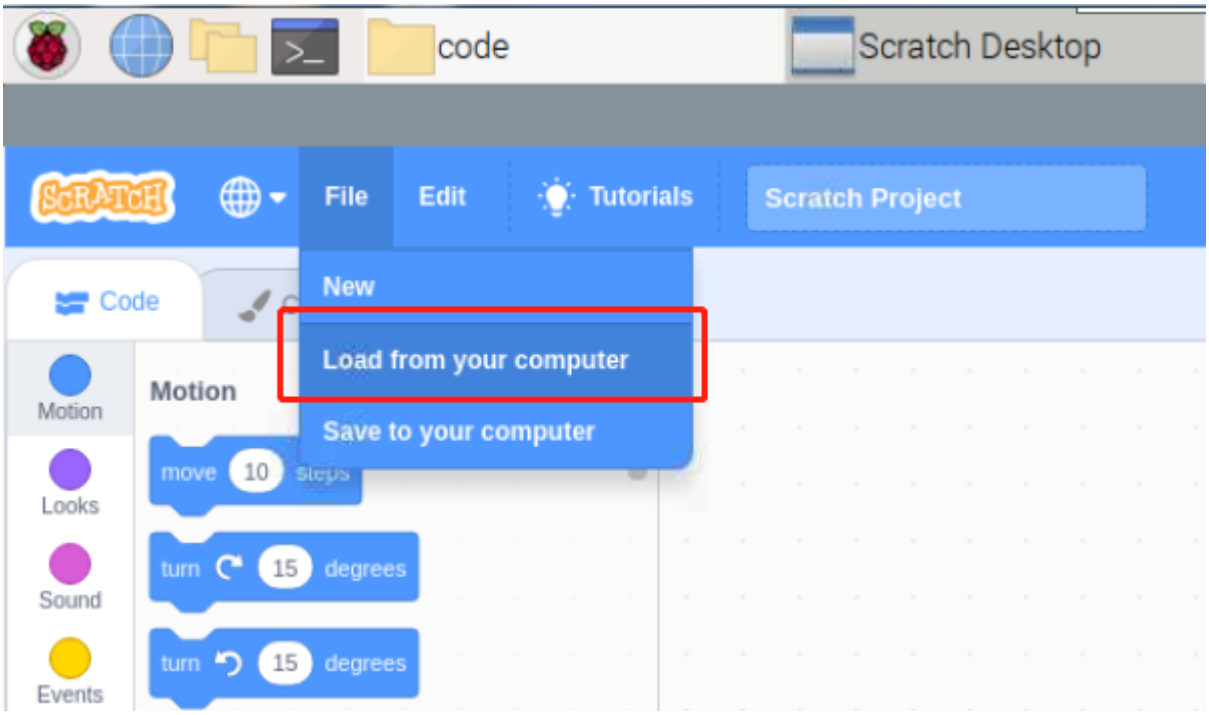


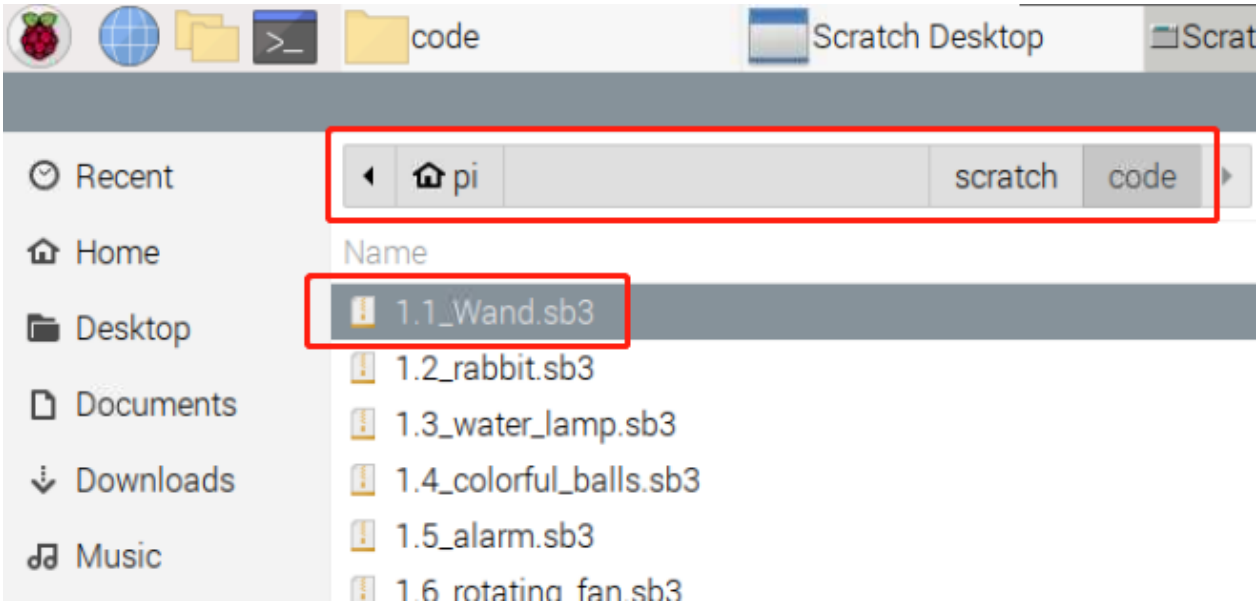

Raspberry Pi GPIO
Control Raspberry Pi GPIO lines
Collaboration with
Raspberry Pi
Raspberry Pi Sense HAT
Control Raspberry Pi Sense HAT
Collaboration with
Raspberry Pi


Motion
Looks
Sound
Events
Control
Sensing
Operators
Variables
My Blocks
Raspberry Pi GPIO
Raspberry Pi GPIO
when gpio 0 is high
gpio 0 is high ?
set gpio 0 to output high
set gpio 0 to input pulled high

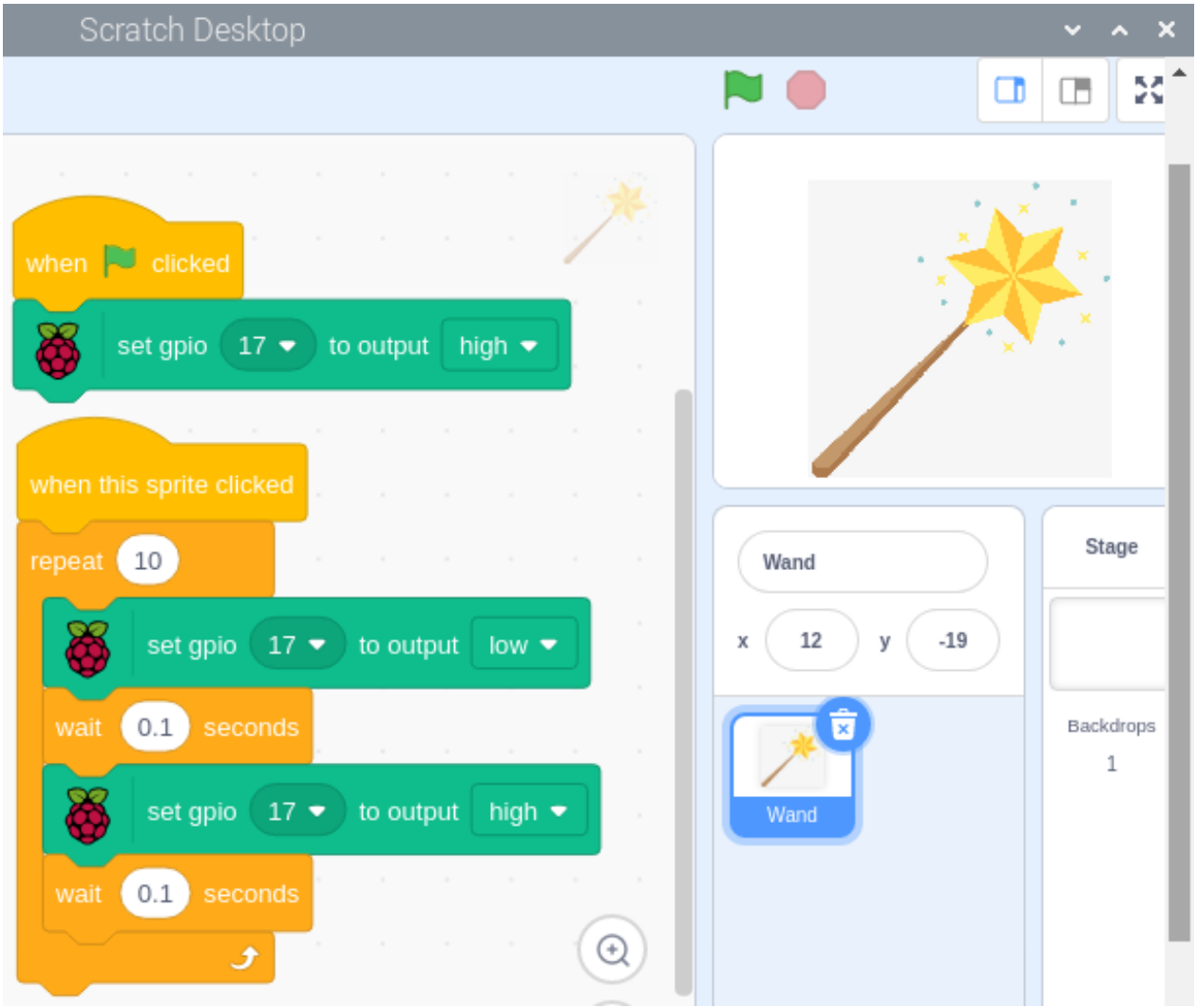
コードをロードして、何が起こるかを確認します

コンピューター (home/pi/davinci-kit-for-raspberry-pi/scratch/code) から Scratch3 にコードファイルをロードします。



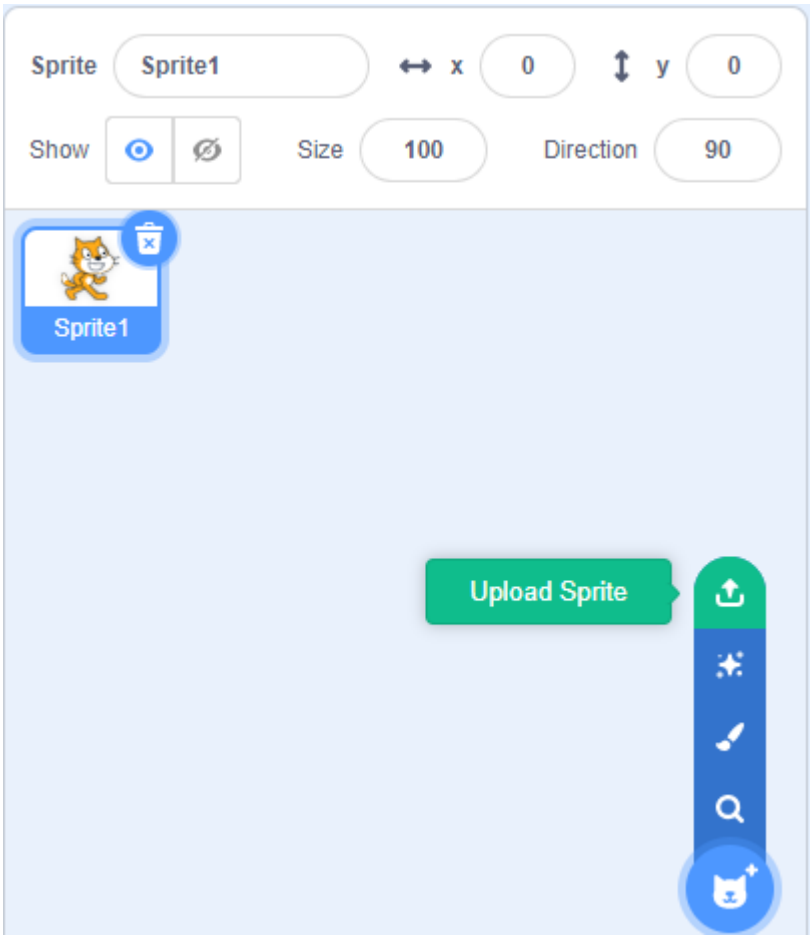


stage area の魔法の杖をクリックすると、LED が 2 秒間点滅します。

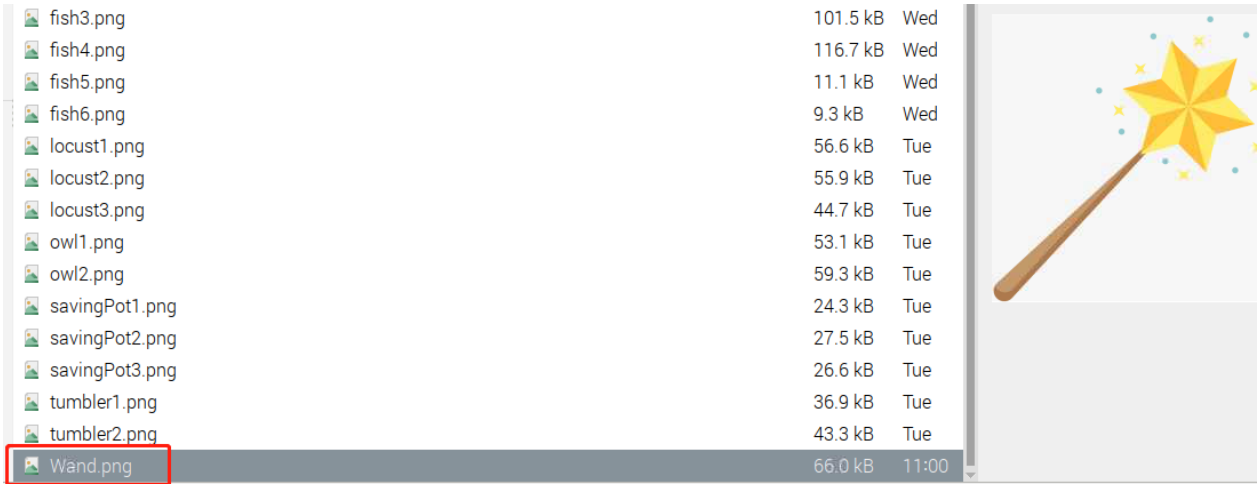


Sprite に関するヒント

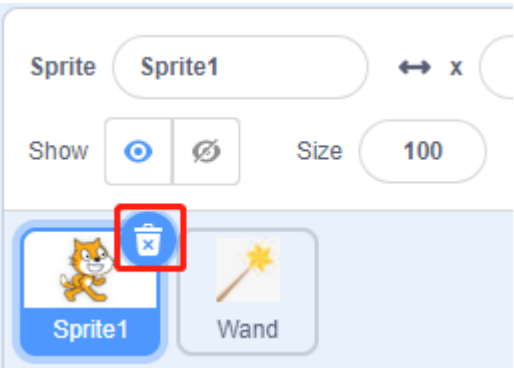
Upload Sprite をクリックします。



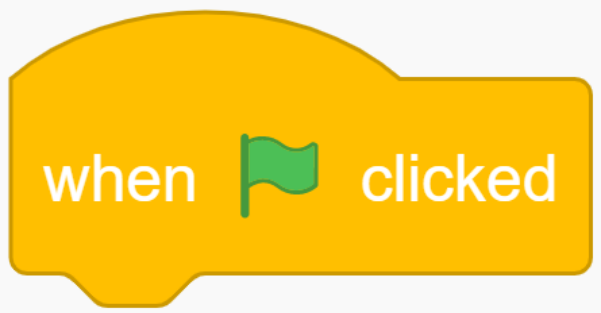
Wand.png を `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` パスから Scratch3 にアップロードします。



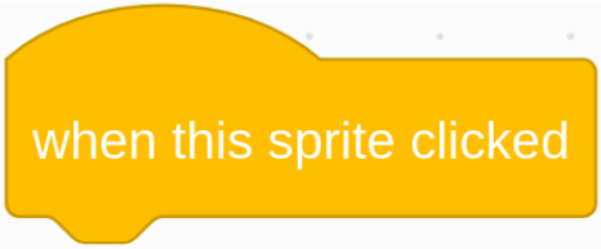
最後に、 **Sprite1** を削除します。



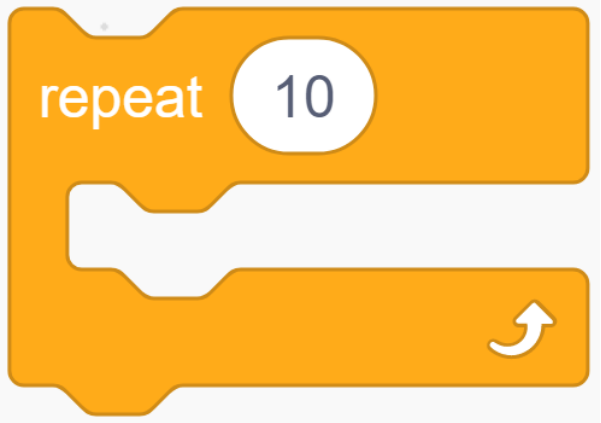
コードに関するヒント



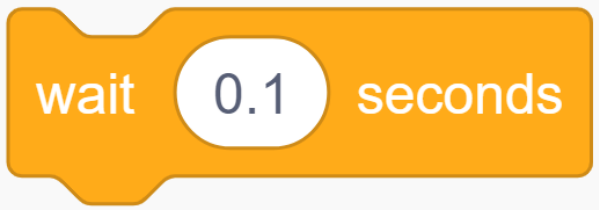
これは、ステージ上の緑色の旗をクリックすることをトリガー条件とするイベントブロックです。 すべてのコードの先頭にトリガーイベントが必要です。block palette の Events カテゴリで他のトリガーイベントを選択できます。



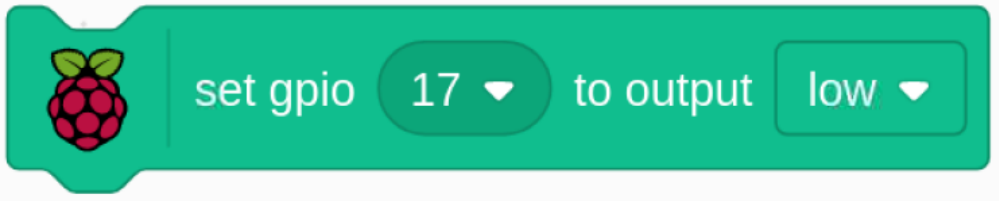
たとえば、トリガーイベントを sprite のクリックに変更できるようになりました。



これは、設定されたサイクル数のブロックです。 数字の 10 を入力すると、ブロック内のイベントが 10 回実行されます。



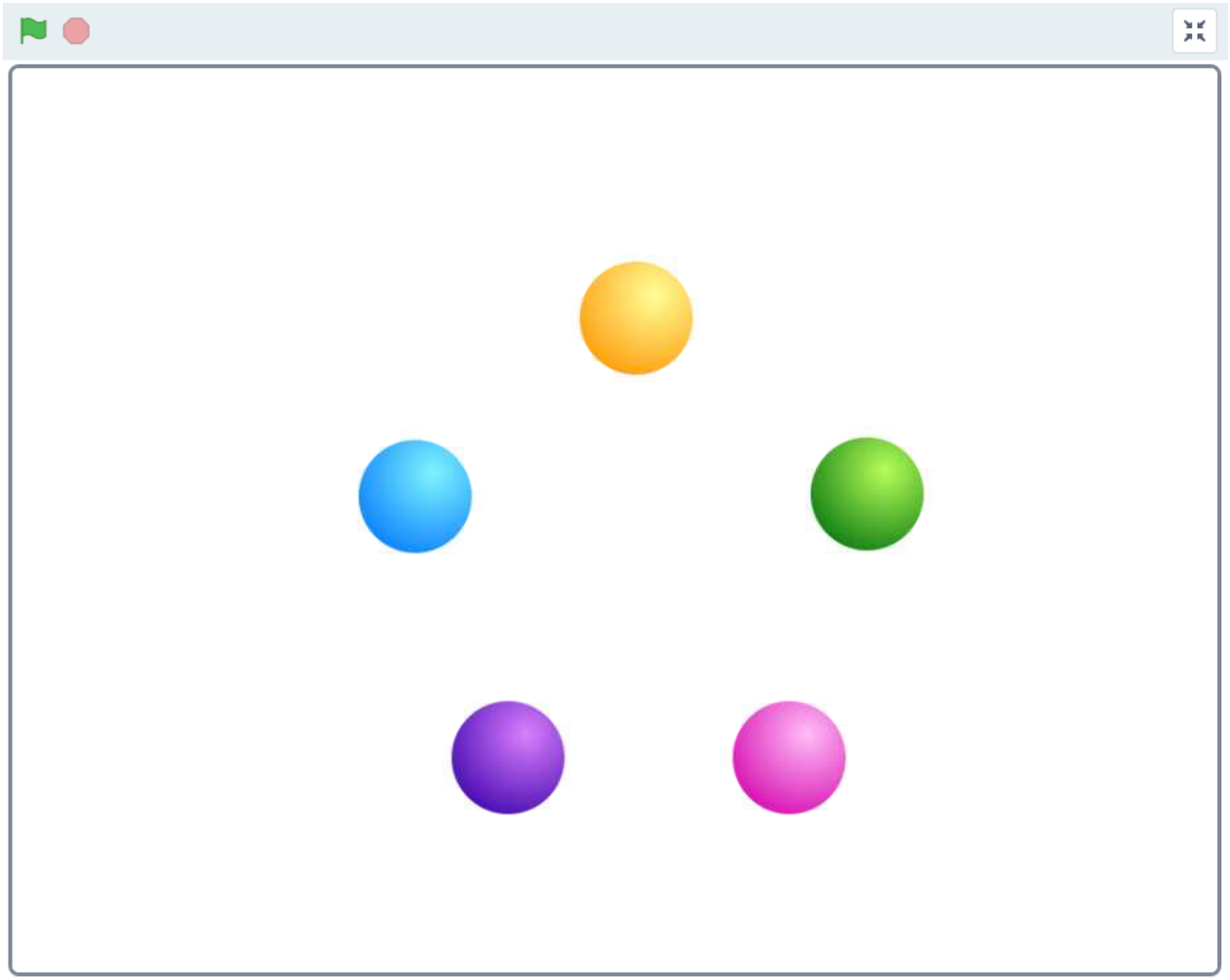
このブロックは、プログラムを秒単位で一時停止するために使用されます。



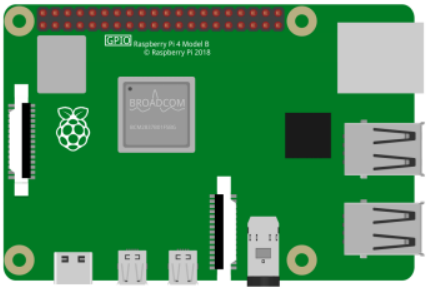
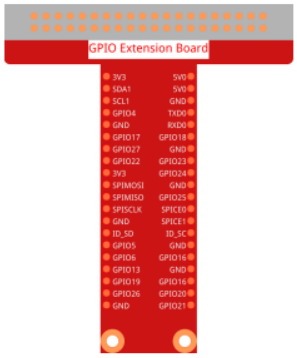
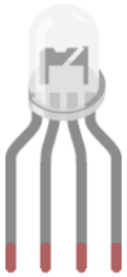
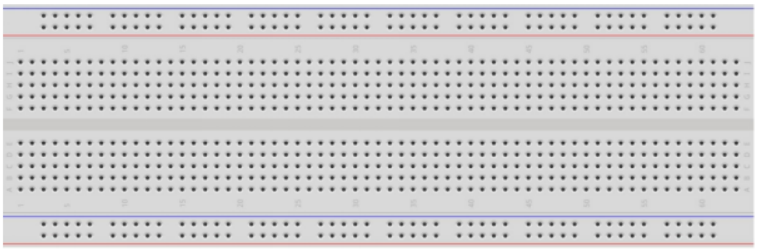
Scratch では BCM の命名方法が使用されているため、このコードは GPIO17（BCM17）を 0V（低レベル）に設定しています。LED のカソードが GPIO17 に接続されているため、LED が点灯します。逆に、GPIO（BCM17）を高く設定すると、LED が消灯します。

1.2 五色の玉

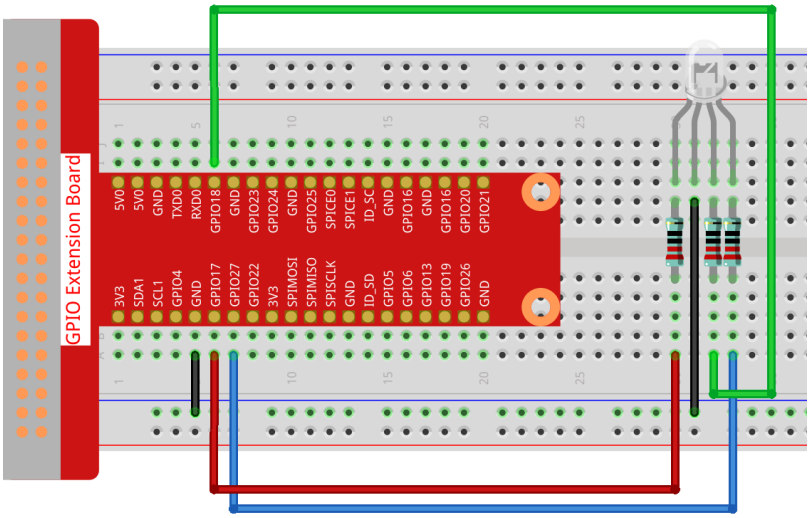
stage area の異なる色のボールをクリックすると、RGBLED が異なる色で点灯します



必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * RGB LED 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Resistor(220Ω)

回路を構築する

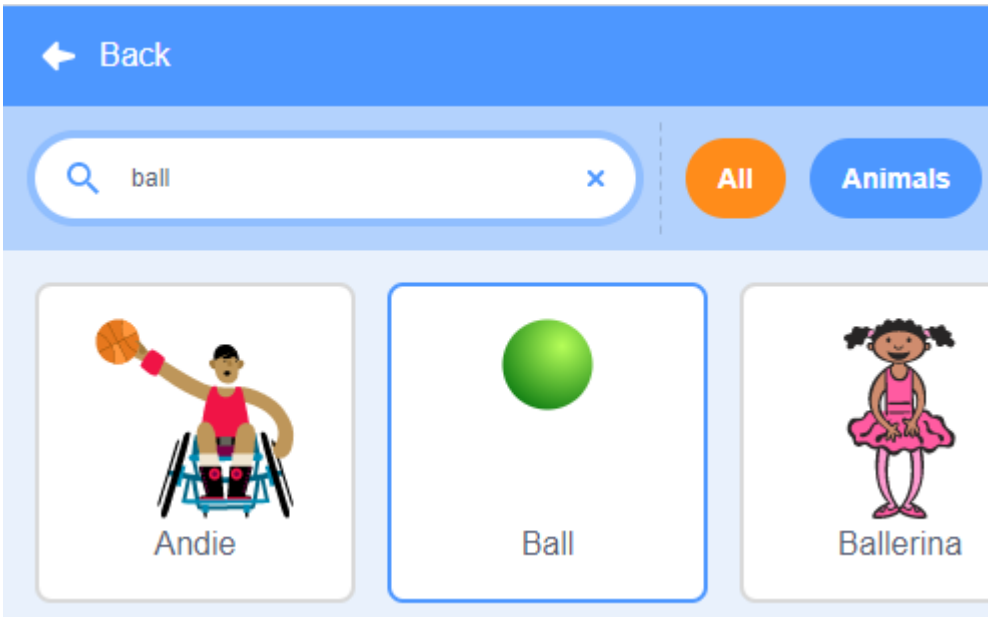


コードをロードして、何が起こるかを確認します

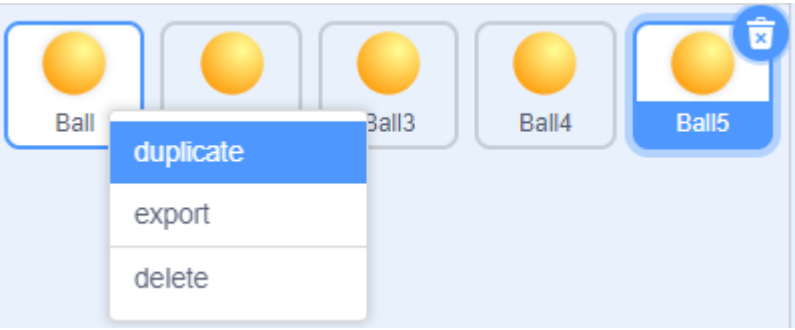
コードファイル (`1.2_colorful_balls.sb3`) を Scratch 3 にロードした後、対応するボールをクリックすると、RGB LED がそれぞれ黄色、青、赤、緑、または紫に点灯します。

Sprite に関するヒント

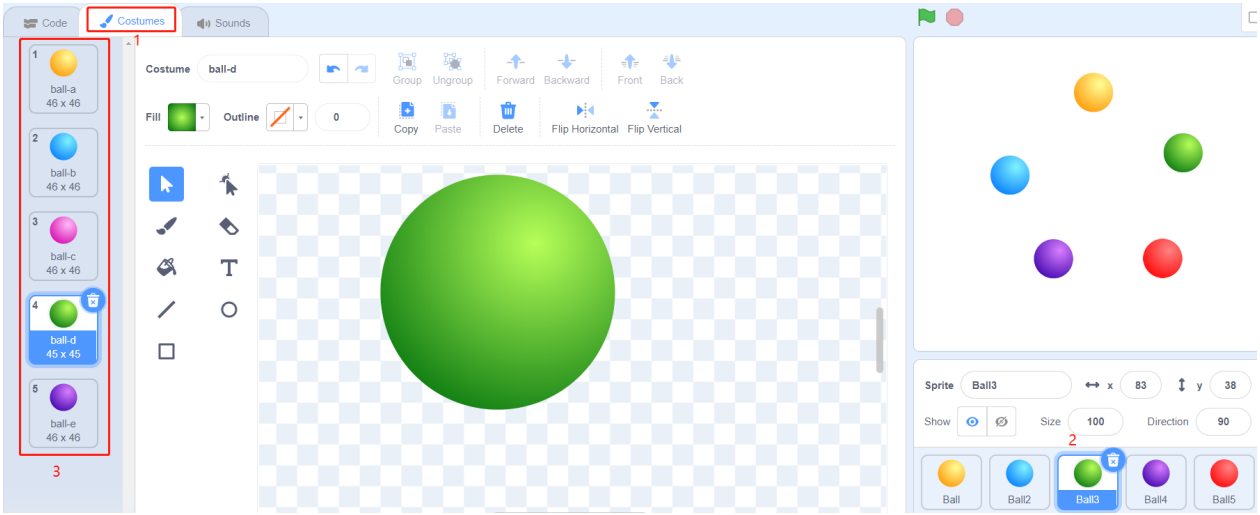
デフォルトの sprite を削除してから、 **Ball** sprite を選択します。



そしてそれを 5 回複製します。



これらの 5 つの **Ball** sprite に異なる **costumes** を選択し、対応する位置に移動します。

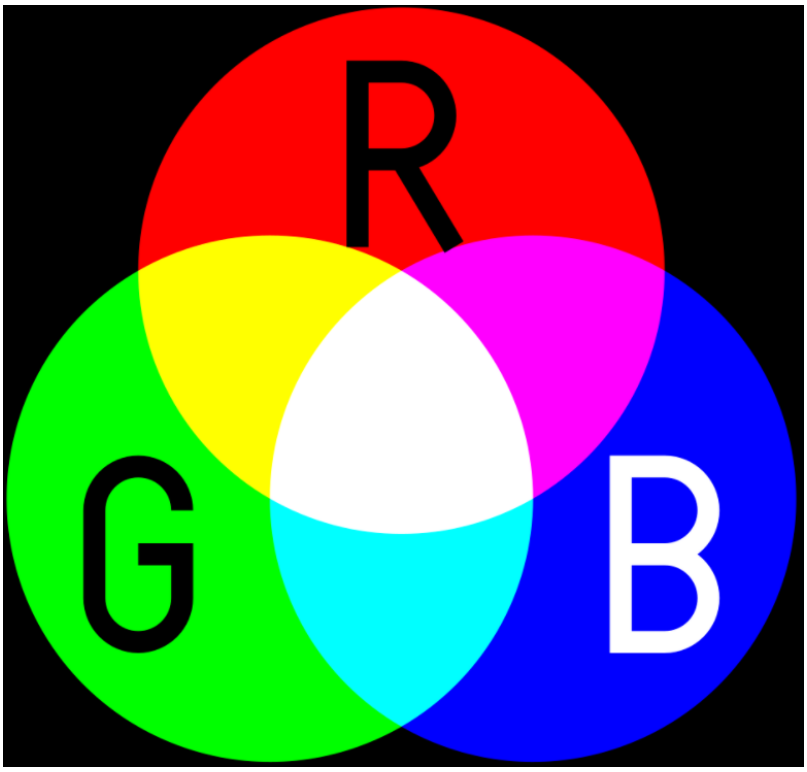


コードに関するヒント

コードを理解する前に、 RGB color model 。

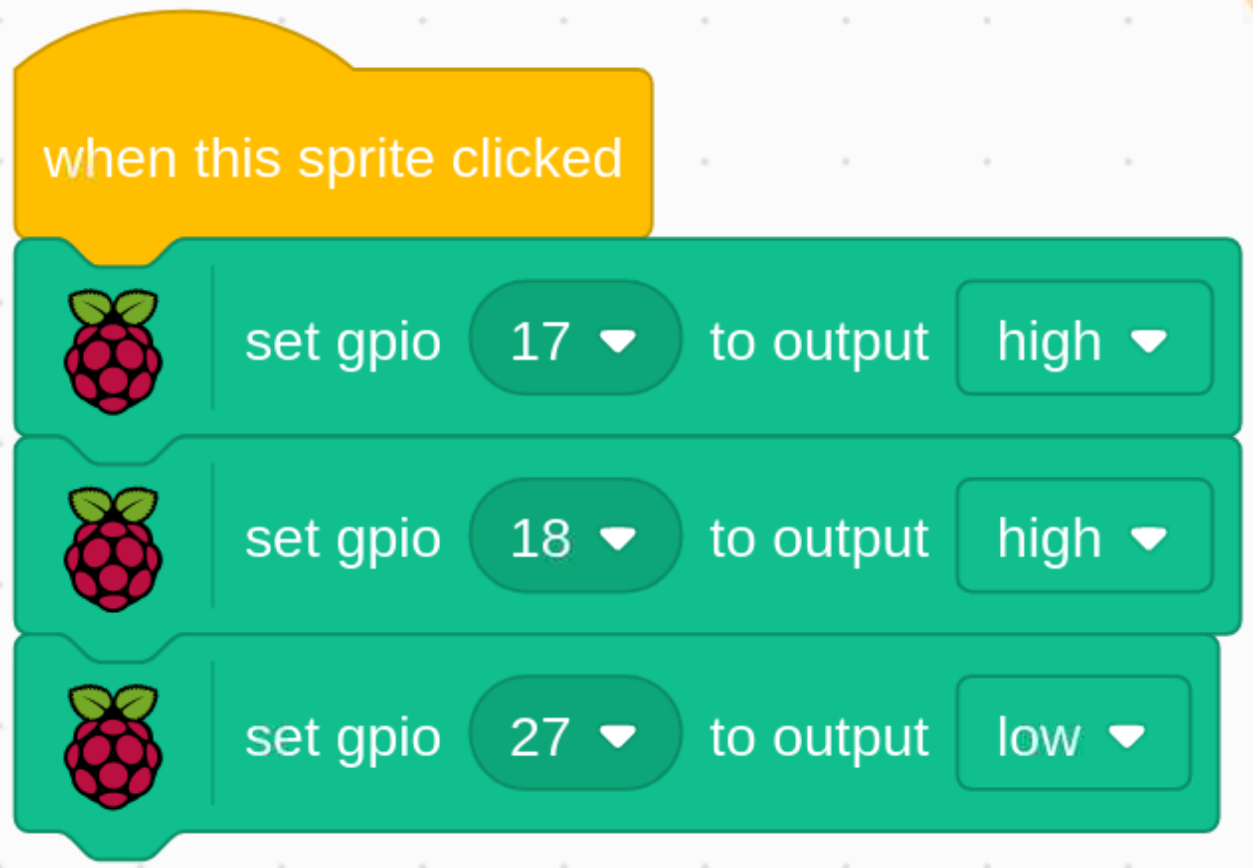
RGB カラーモデルは、赤、緑、青の光をさまざまな方法で足し合わせて、さまざまな色を再現する加法色モデルです。

加法混色：緑に赤を加えると黄色になります。 青に緑を加えるとシアンになります。 赤に青を加えるとマゼンタになります。 3 つの原色すべてを足し合わせると白になります。



RGB LED は、3 つの LED（赤色 LED、緑色 LED、青色 LED）を 1 つのパッケージに組み合わせたもので、これら 3 つの色を組み合わせることでほぼすべての色を生成できます。4 つのピンがあり、そのうちの 1 つは GND で、他の 3 つのピンはそれぞれ 3 つの LED を制御します。

したがって、RGBLED を明るい黄色にするコードは次のとおりです。

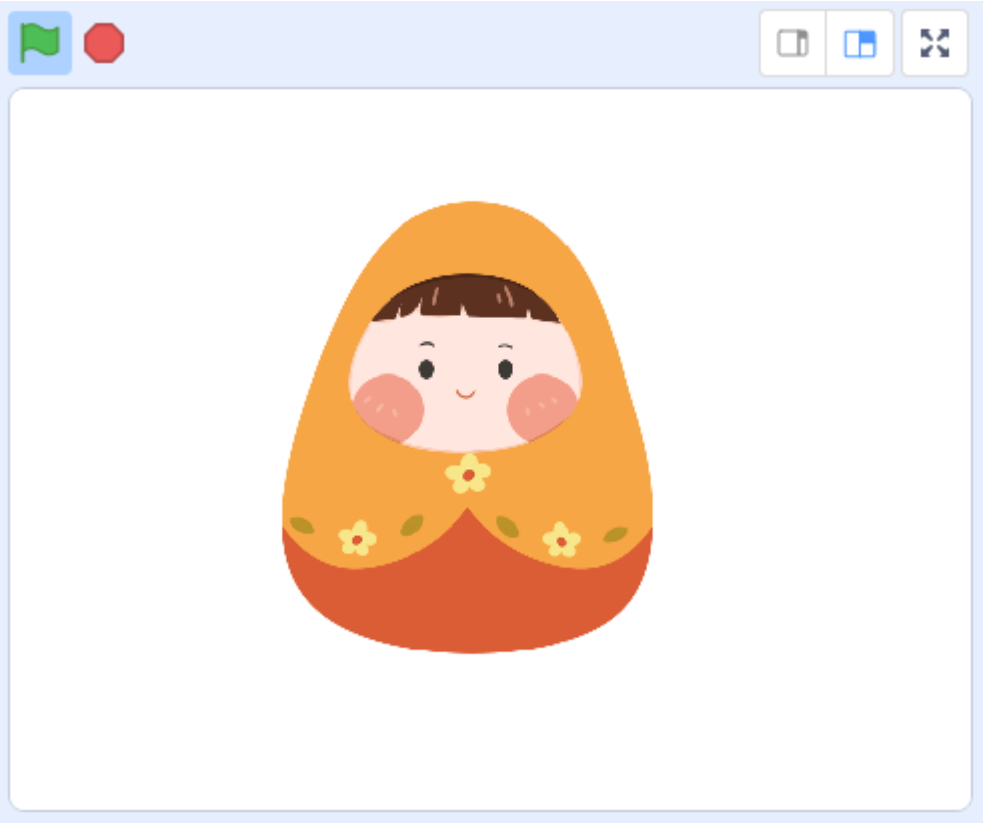


Ball sprite（黄色のボール）をクリックすると、GPIO17 が高（赤の LED がオン）、GPIO18 が高（緑の LED がオン）、GPIO27 が低（青の LED がオフ）に設定され、RGBLED が黄色に点灯するように設定します。

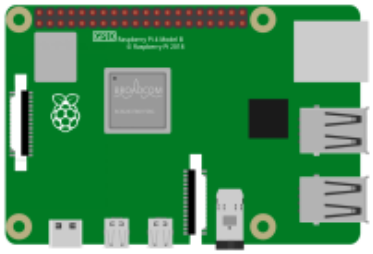
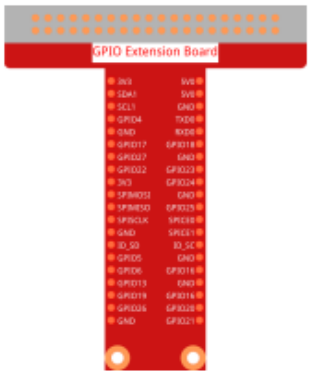
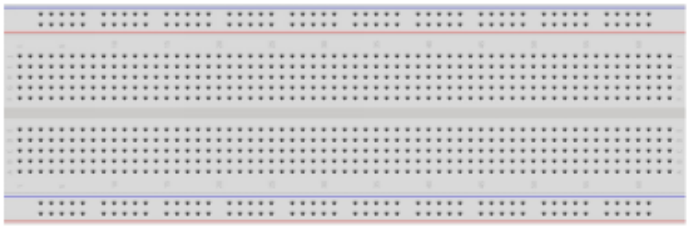
同じ方法で他の sprite にコードを書き込んで、RGB LED を対応する色で点灯させることができます。

1.3 不倒翁

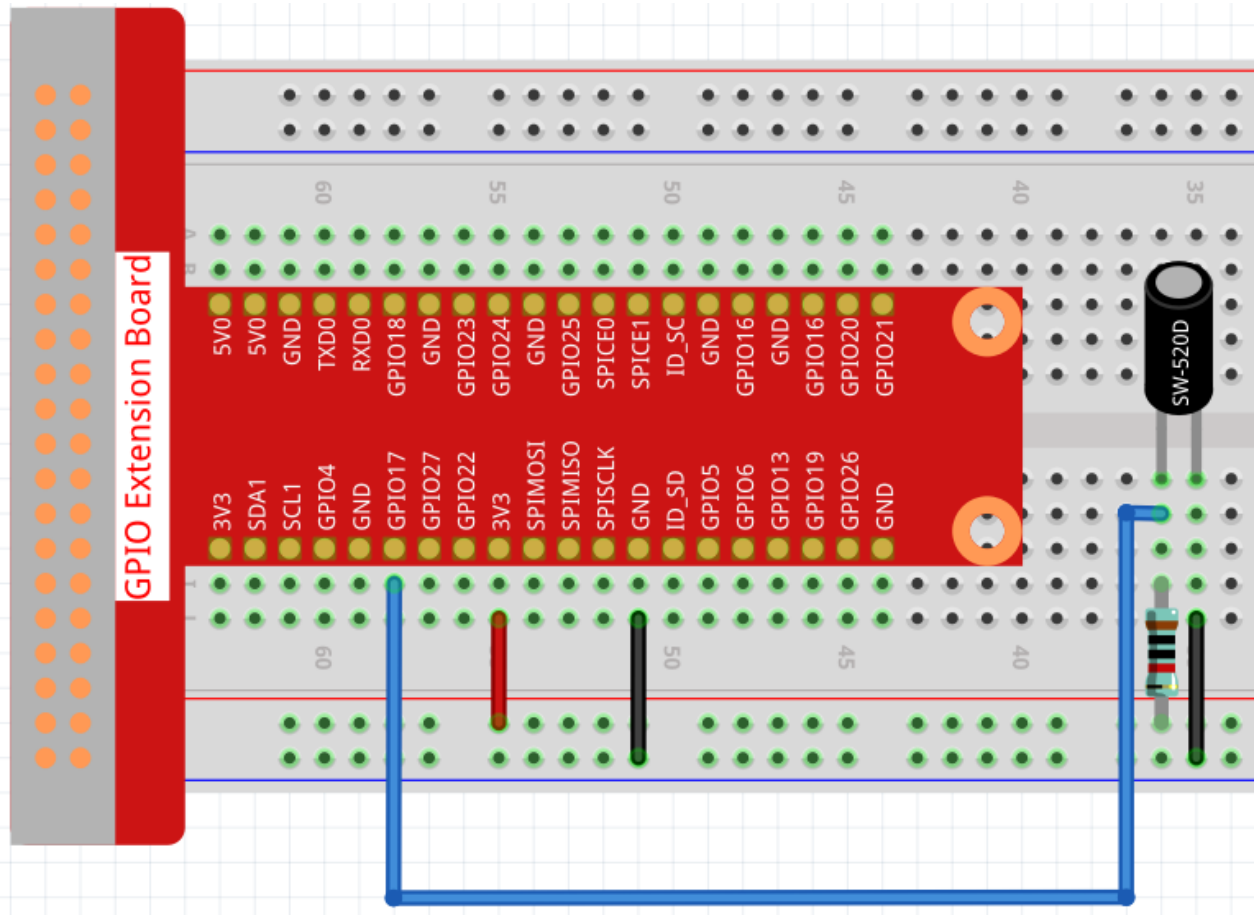
In this project, we will make a tilt switch controlled tumbler toy.



必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Tilt Switch
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor(10kΩ)

回路を構築する



コードをロードして、何が起こるかを確認します

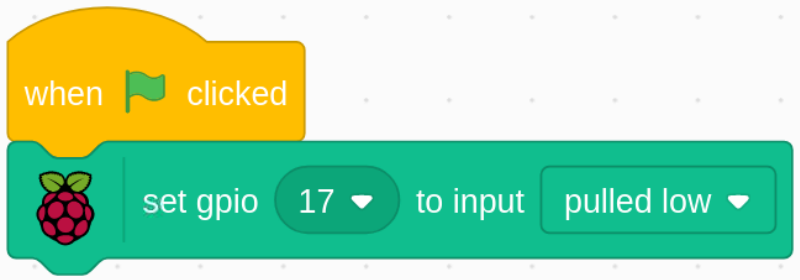
コードファイル (`1.3_tumbler.sb3`) を Scratch3 にロードします。

チルトスイッチを直立させると、タンブラーは立ったままになります。 傾けるとタンブラーも落ちます。 再び直立させると、タンブラーが再び立ち上がります。

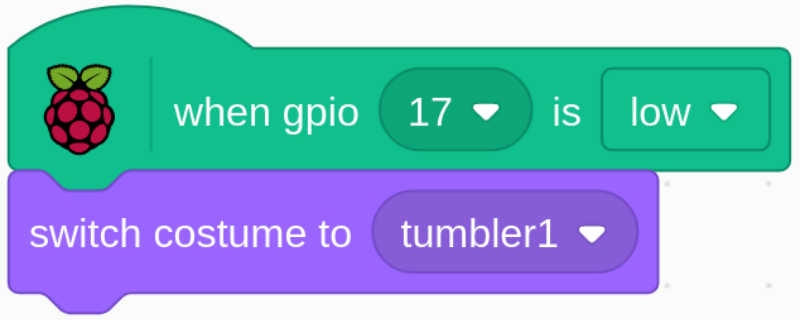
Sprite に関するヒント

Sprite1 を選択し、左上隅の **Costumes** をクリックします。**Upload Costume** ボタンを使用して、 `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` パスから **tumbler1.png** と **tumbler2.png** をアップロードします。デフォルトの 2 つの costumes を削除し、 sprite の名前を **tumbler** に変更します。

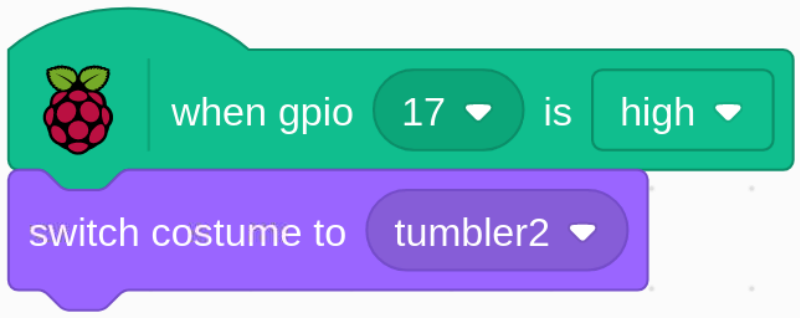
コードに関するヒント



緑の旗をクリックすると、GPIO17 の初期状態が low に設定されます。



pin17 が低い場合（チルトスイッチが直立している場合)、tumbler sprite の costume を tumbler1（直立状態）に切り替えます。

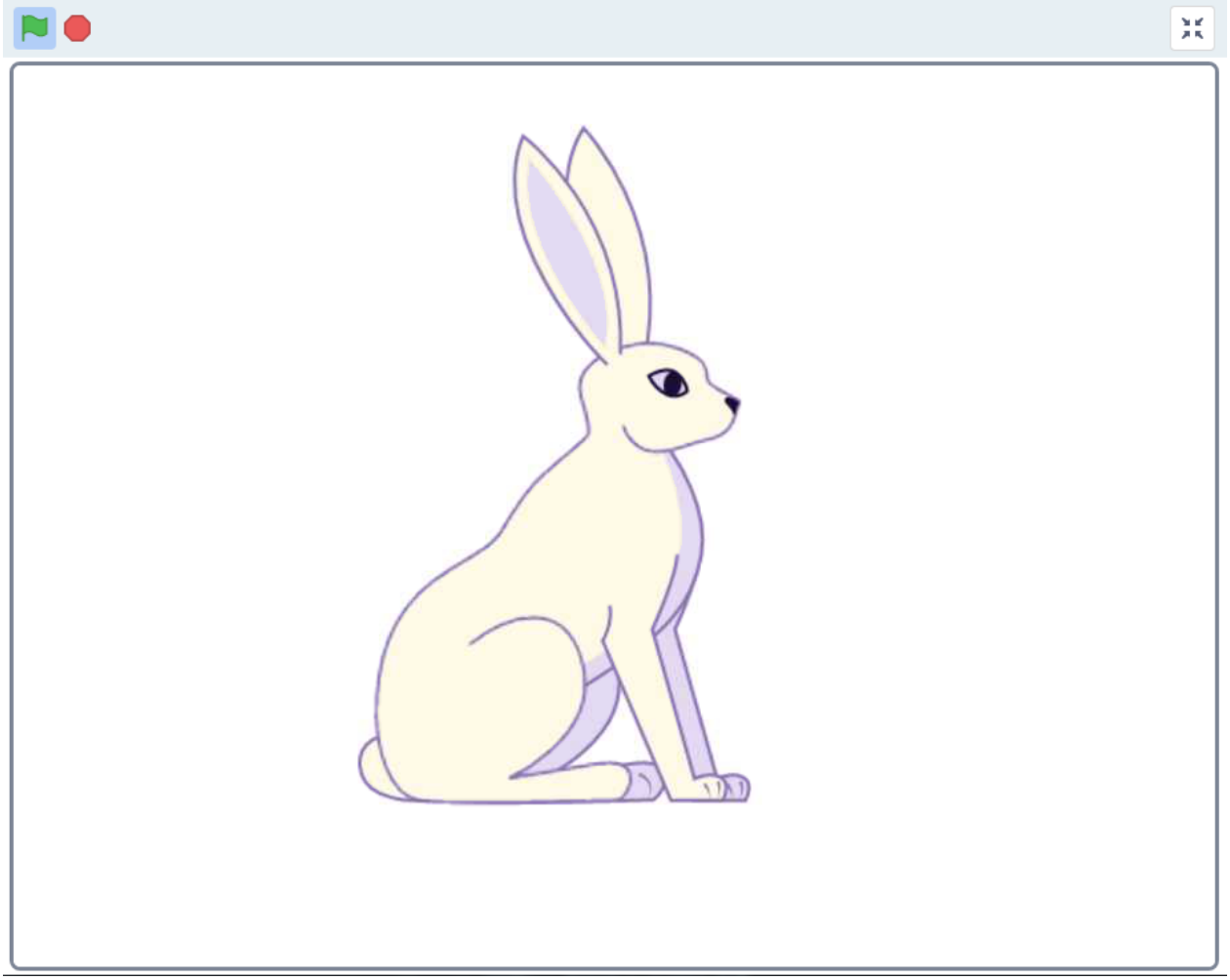


pin17 がハイ（チルトスイッチが傾いている）のとき、tumbler sprite の costume を tumbler2（チルト状態）に切り替えます。

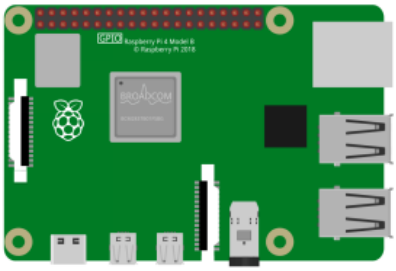
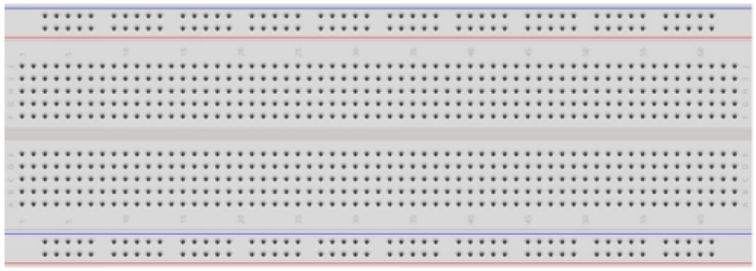
1.4 兎

今日は、Button、Raspberry Pi、Scratch を使用して、さまざまな変更を加えたウサギを作成します。

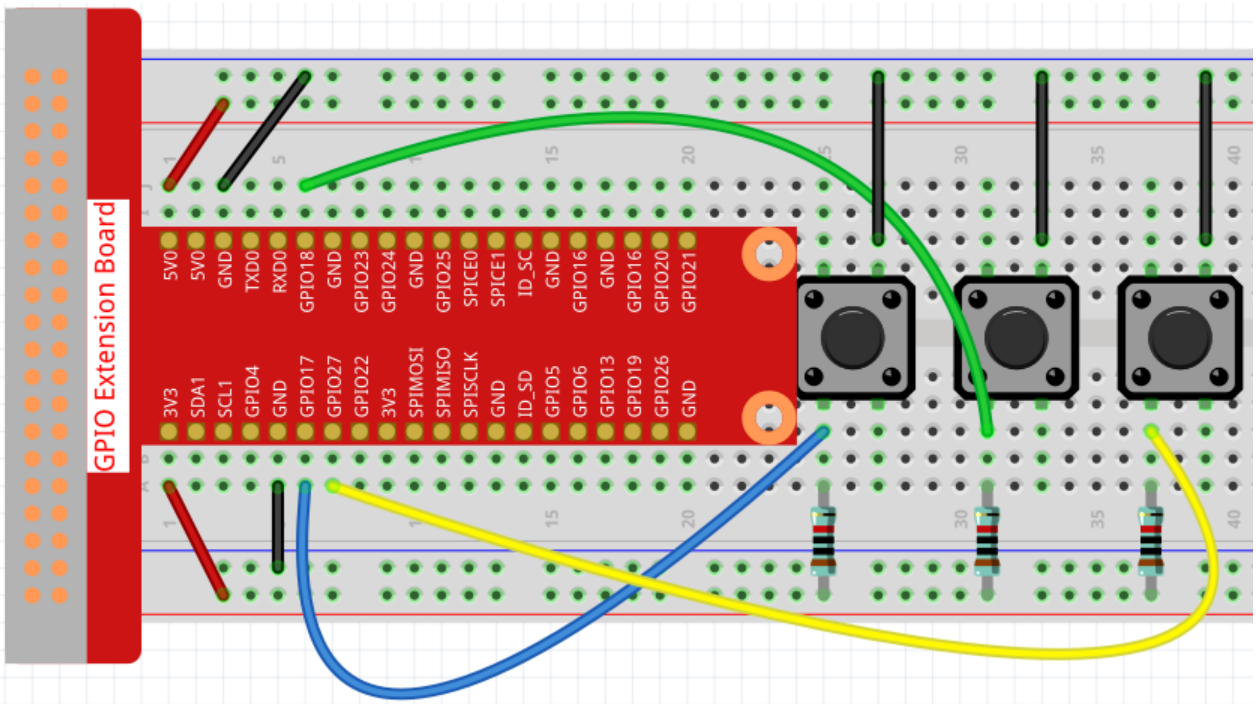
最初のボタンを押すと、stage area のうさぎの体色が変わります。 2 番目のボタンを押すと、うさぎは体のサイズを変更します。 3 番目のボタンを押すと、うさぎは一歩前進します。



必要な部品

1 * Raspberry Pi 	1 * T-Extension Board	3* Resistor 10KΩ
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Button

回路を構築する



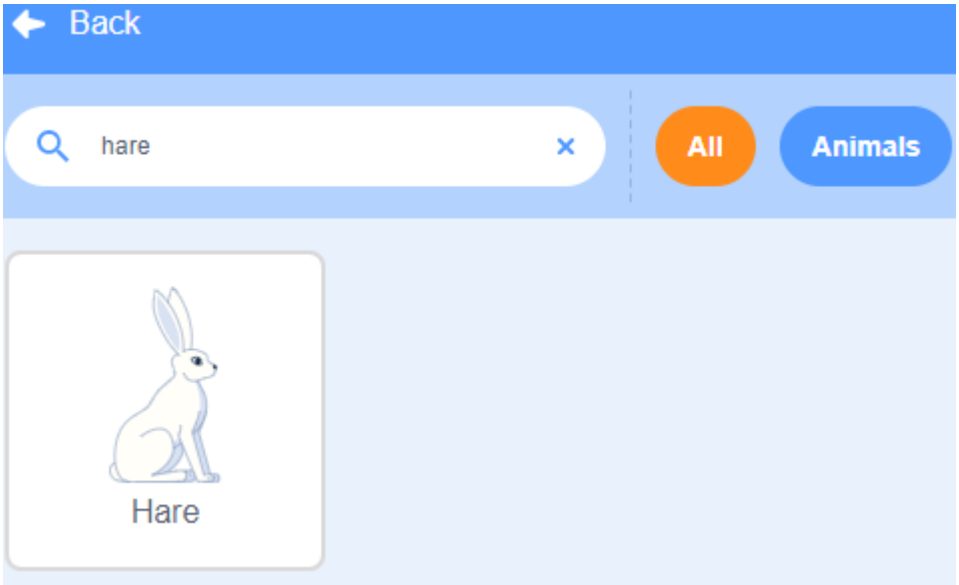
コードをロードして、何が起こるかを確認します

コードファイル (`1.4_hare.sb3`) を Scratch3 にロードします。

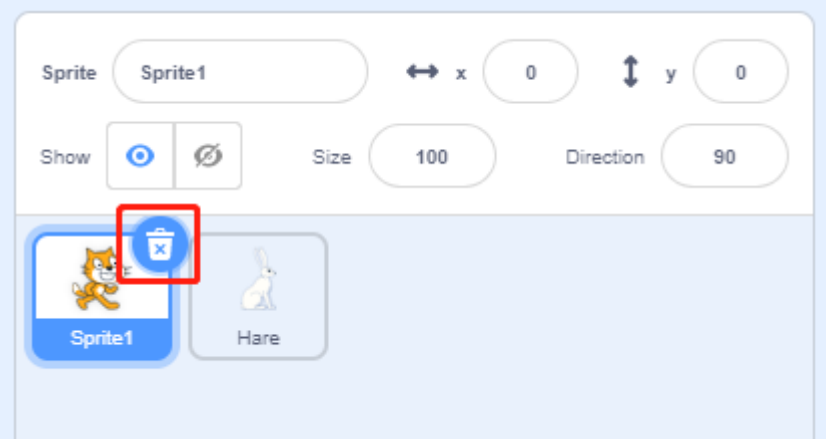
これで、3 つのボタンをそれぞれ押して、ステージ上のうさぎがどのように変化するかを確認できます。

Sprite に関するヒント

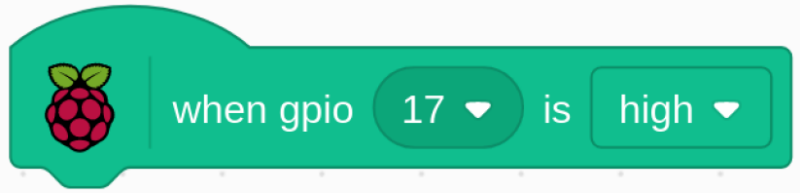
sprite area の右下隅にある **Choose a Sprite** ボタンをクリックし、検索ボックスに **Hare** と入力して、クリックして追加します。



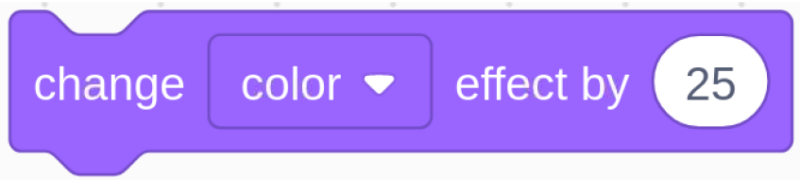
Sprite1 を削除します。



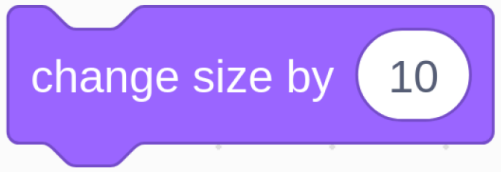
コードに関するヒント



これは、GPIO17 のレベルが高いときにトリガーされるイベントブロックです。つまり、その瞬間にボタンが押されます。

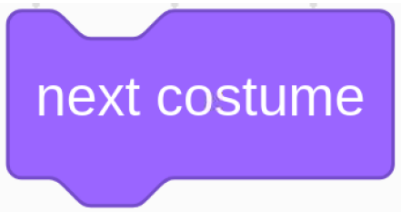


これは Hare の色を変更するためのブロックです。値の範囲は 0〜199 で、199 を超えると再び 0 から変更されます。



これは、sprite のサイズを変更するために使用されるブロックです。値が大きいほど、sprite は大きくなります。

注釈: sprite も無限に大きくはなく、その最大サイズは元の画像サイズに関連しています。



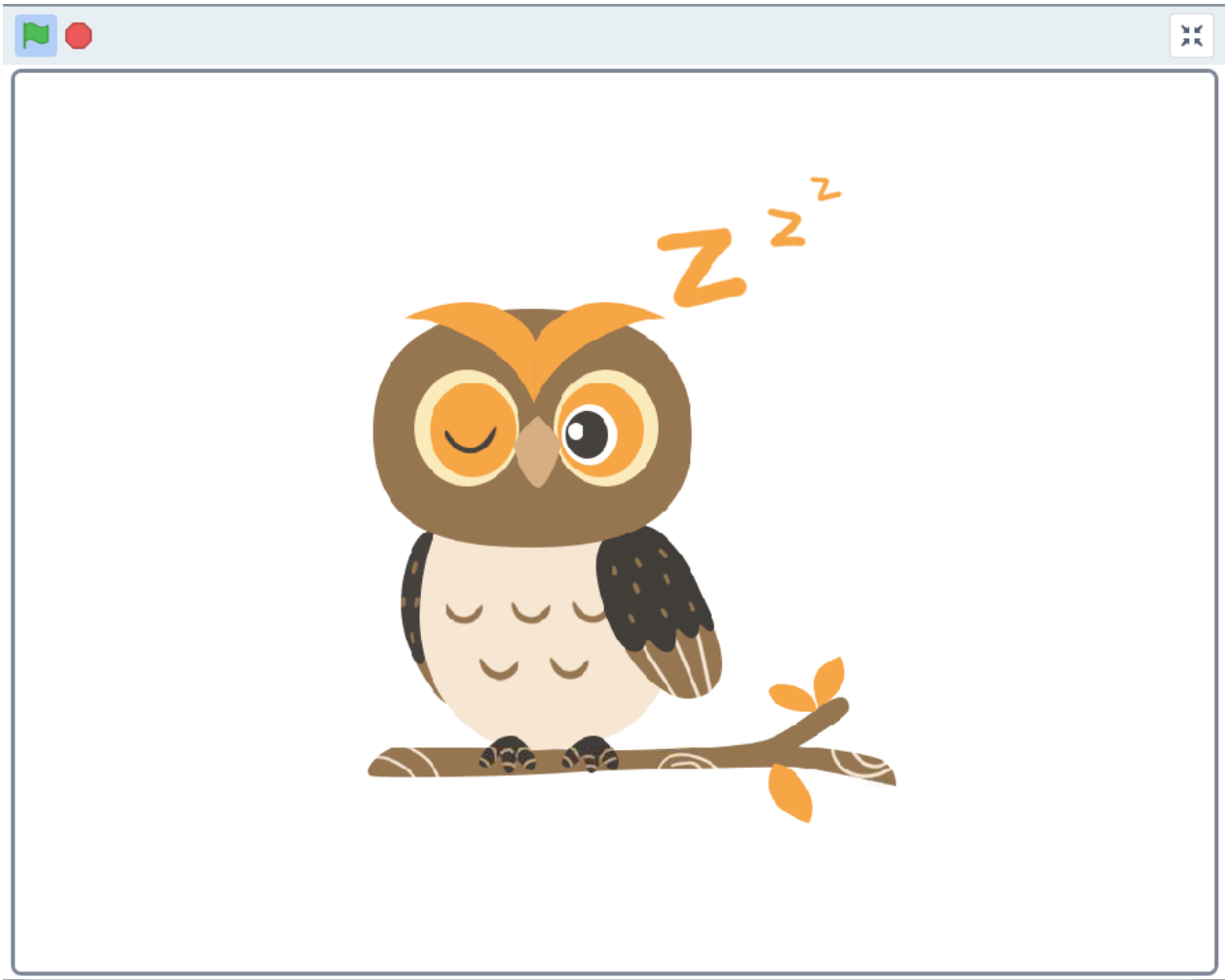
これはスプライトコスチュームを切り替えるブロックであり、Hare のコスチュームが切り替わり続けると、一連の一貫したアクションを実行します。 たとえば、このプロジェクトでは、Hare に一歩前進させます。

1.5 梟を起こす

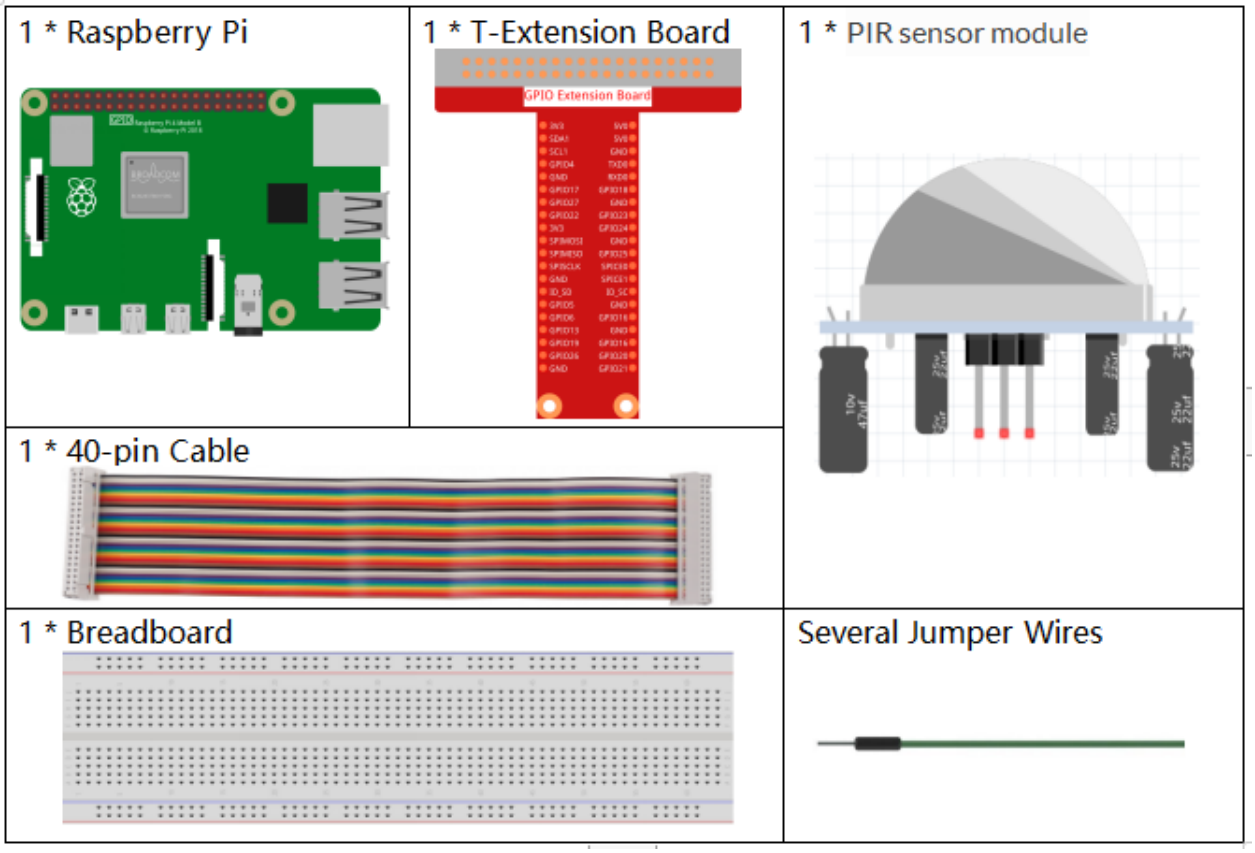
今日はフクロウを起こすゲームをします。

誰かが PIR センサーモジュールに近づくと、フクロウは眠りから目覚めます。

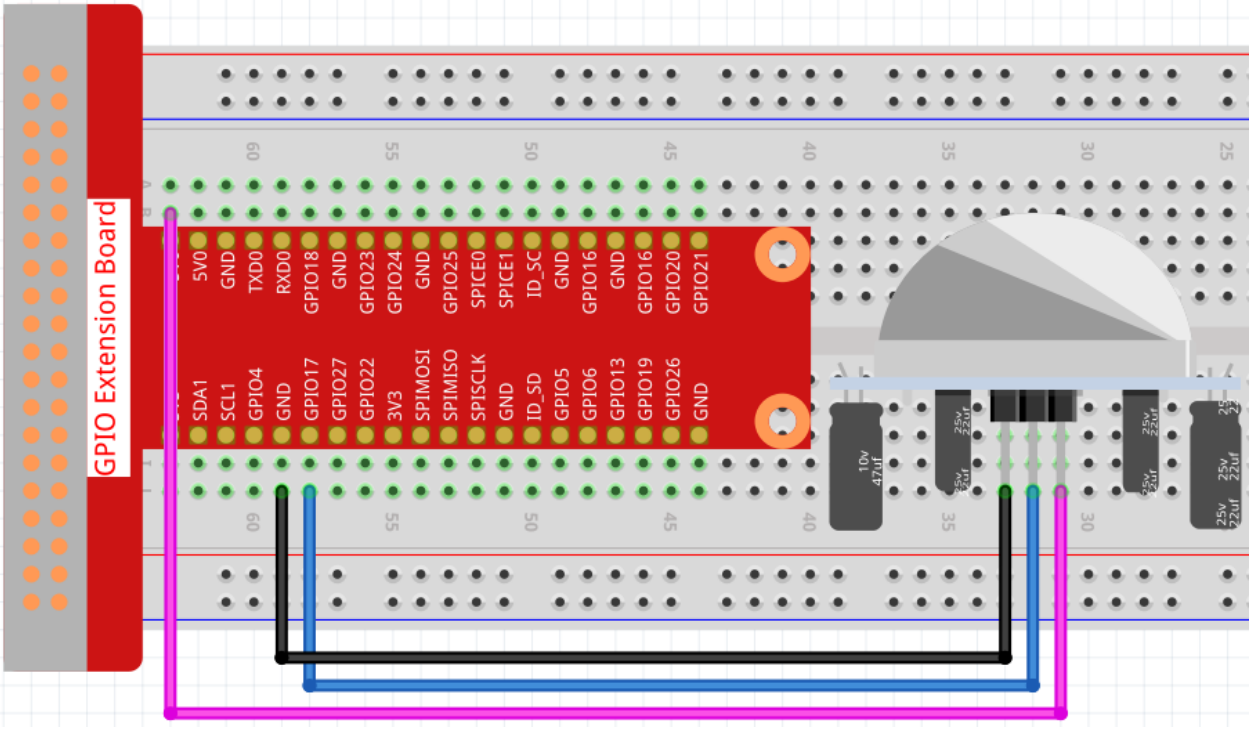
PIR モジュールには 2 つのポテンショメータがあり、1 つは感度を調整するためのもので、もう 1 つは検出距離を調整するためのものです。PIR モジュールがうまく機能するためには、両方を完全に反時計回りに回す必要があります。



必要な部品


1 * Raspberry Pi
1 * T-Extension Board
GPIO Extension Board
1 * PIR sensor module
1 * 40-pin Cable
1 * Breadboard
Several Jumper Wires

回路を構築する



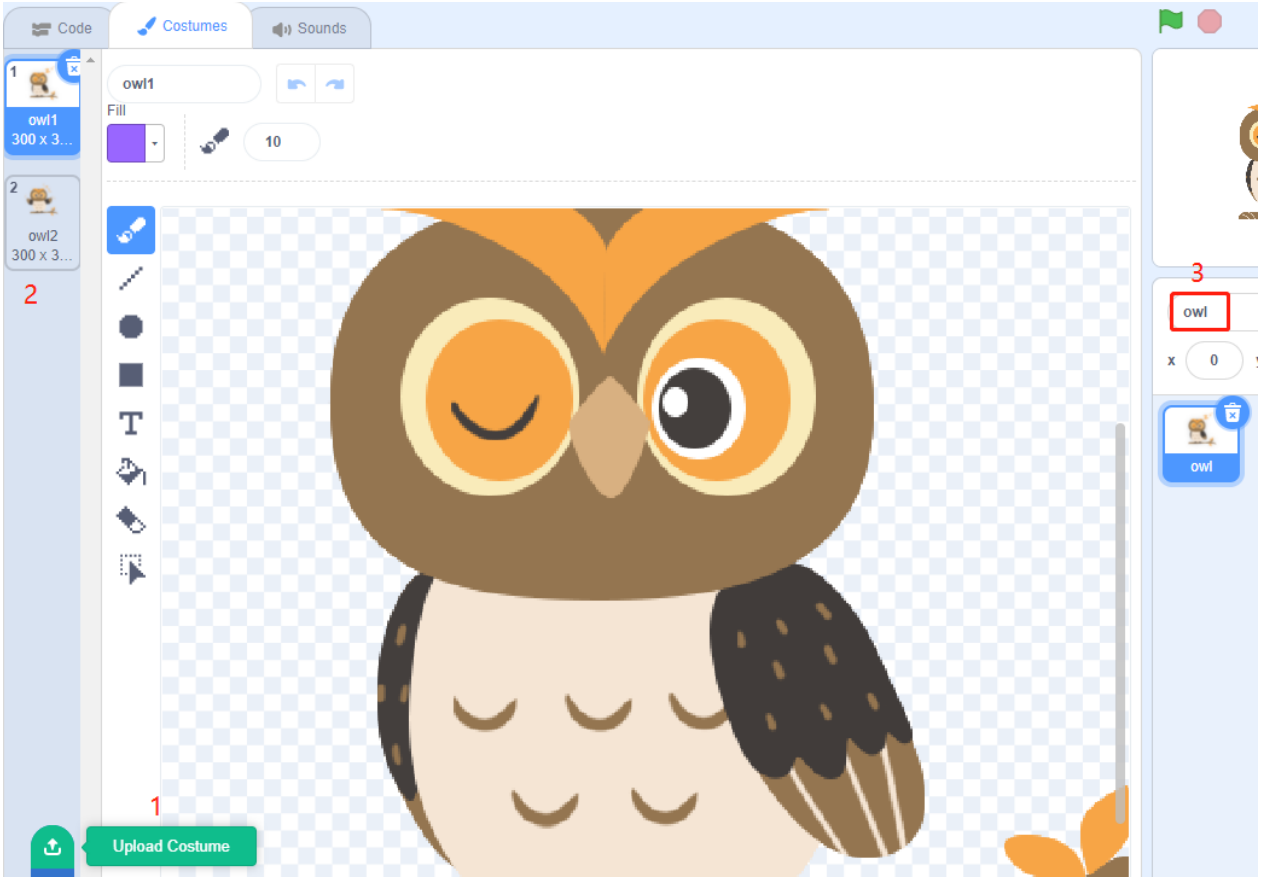
コードをロードして、何が起こるかを確認します

コードファイル (`1.5_wake_up_the_owl.sb3`) を Scratch3 にロードします。

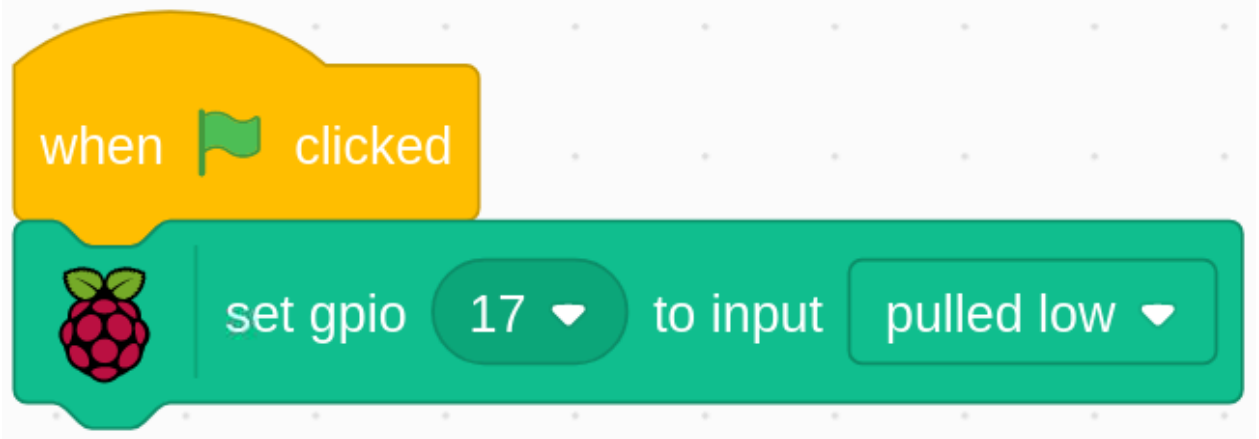
PIR センサーモジュールに近づくと、stage area のフクロウが羽を開いて目を覚ますのがわかります。離れると、フクロウは再び眠りに戻ります。

Sprite に関するヒント

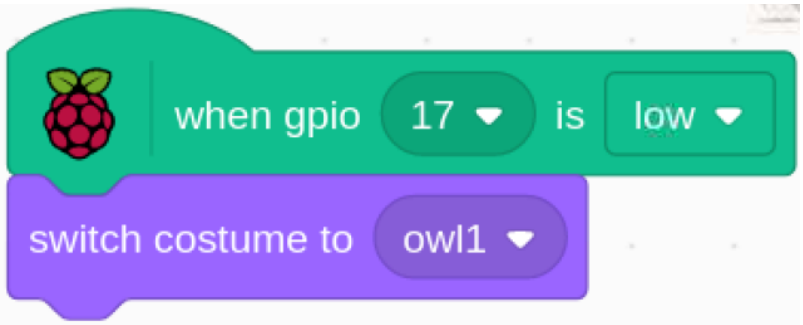
Sprite1 を選択し、左上隅にある コスチューム をクリックします。owl1.png と owl2.png を、[コスチュームのアップロード] ボタンを介して `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` パスからアップロードします。デフォルトの 2 つのコスチュームを削除し、スプライトの名前を owl に変更します。



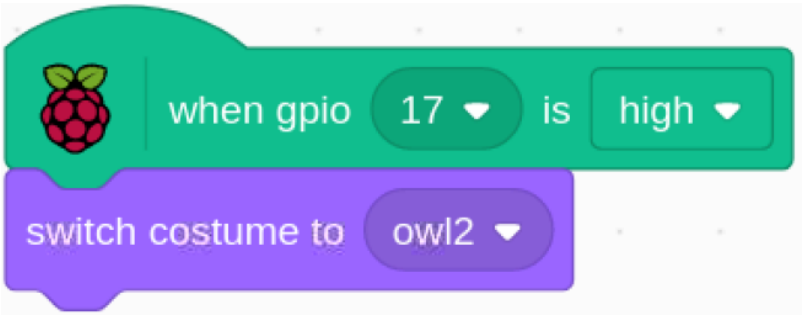
コードに関するヒント



緑の旗をクリックすると、GPIO17 の初期状態が low に設定されます。



pin17 が低い（誰も近づいていない）場合は、フクロウの Sprite の costume を owl1（sleeping 状態）に切り替えます。

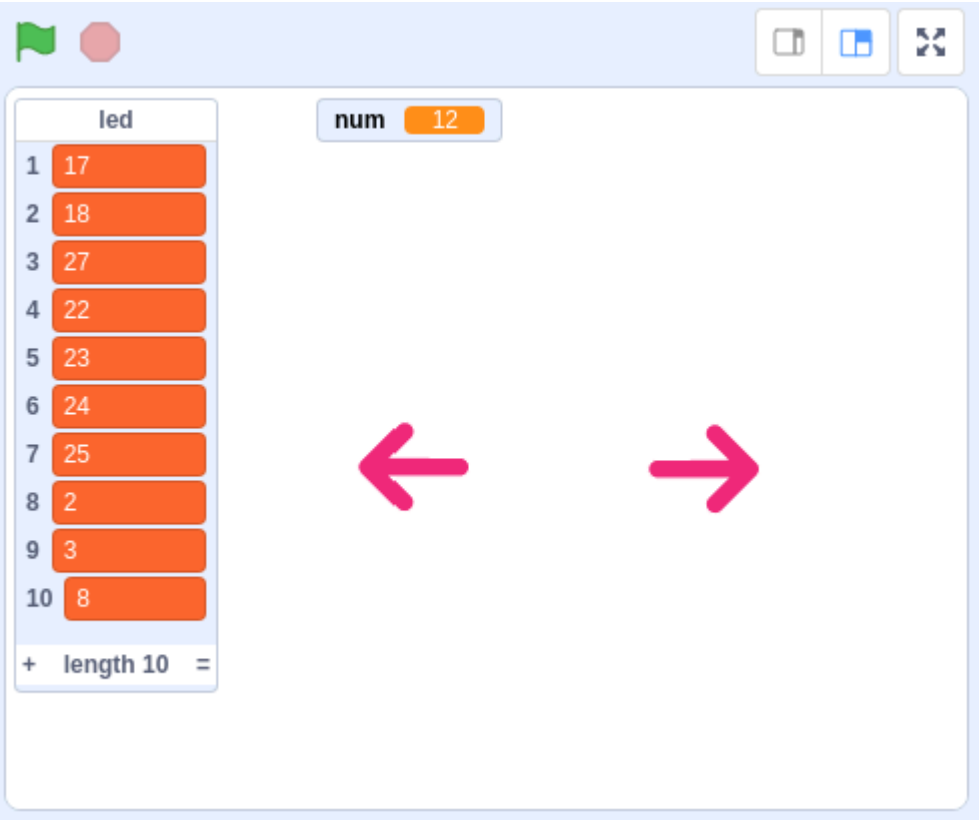


pin17 が高い（誰かが近づいている）とき、フクロウの Sprite の costume を owl2（wake up 状態）に切り替えます。

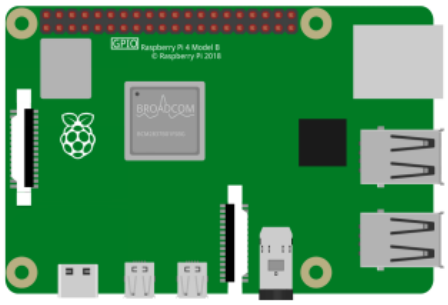
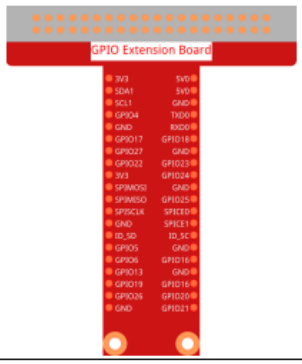
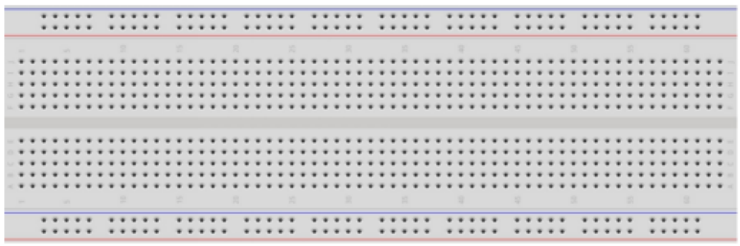
1.6 光が流れる LED

今日は LED Bargraph、Raspberry Pi、Scratch を使ってストリーマー LED を作ります。

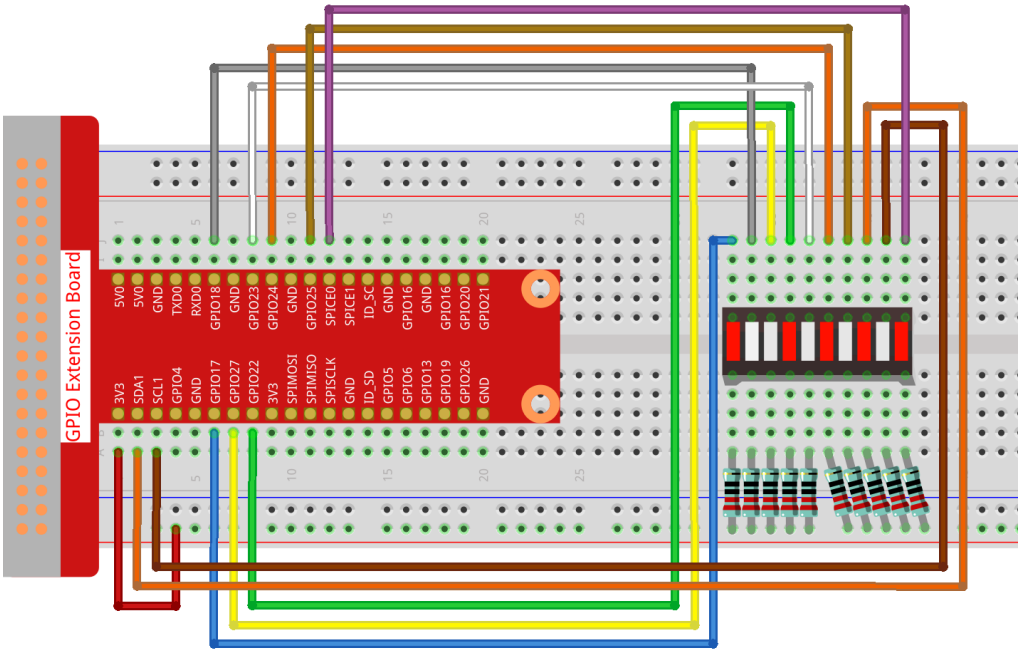
LED 棒グラフが stage 上の矢印の方向に合わせて点灯します。


led
1 17
2 18
3 27
4 22
5 23
6 24
7 25
8 2
9 3
10 8
+ length 10 =
num 12

必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Bargraph
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		10 * Resistor(220Ω)

回路を構築する



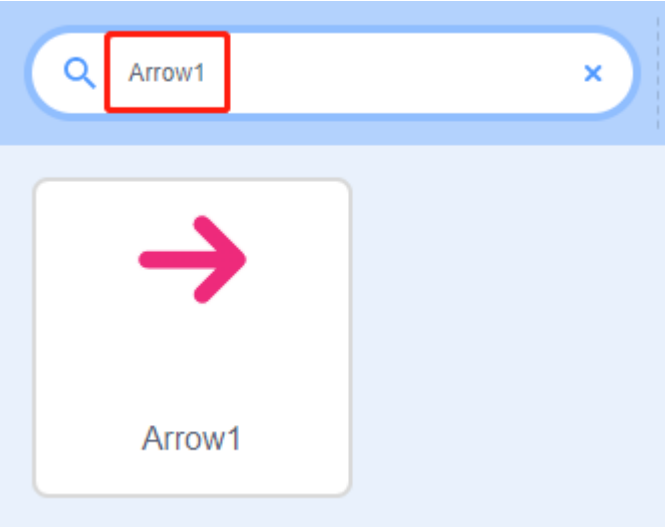
コードをロードして、何が起こるかを確認します

コードファイル (`1.6_water_lamp.sb3`) をコンピューターから Scratch3 にロードします。

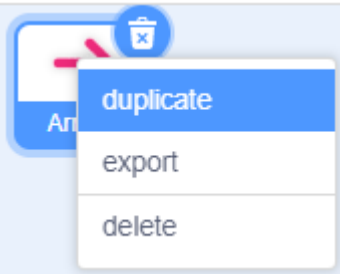
Arrow1 をクリックすると、LED バーの LED が左から右に（一度に 1 つずつ）順番に点灯し、その後消灯します。Arrow2 をクリックすると、LED が逆の順序で点灯します。

Sprite に関するヒント

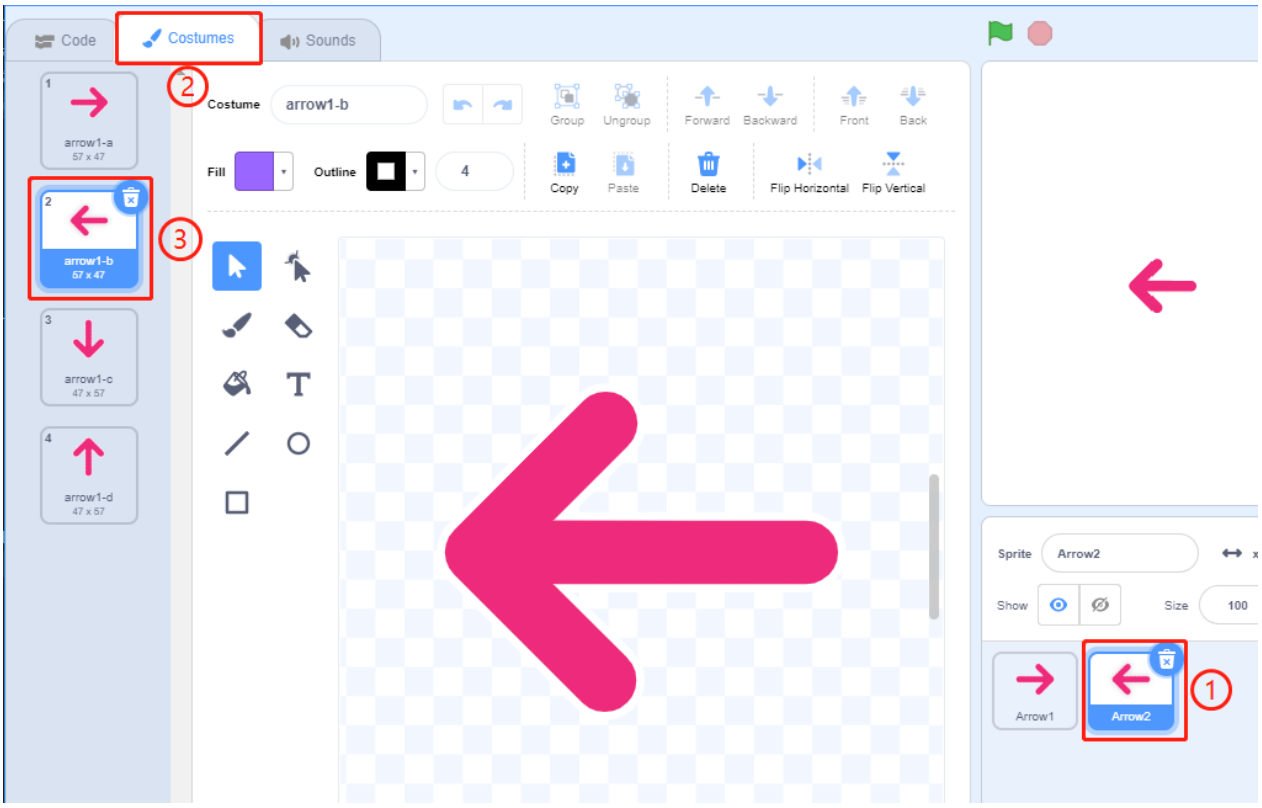
デフォルトの sprite を削除し、Arrow1 sprite を選択します。



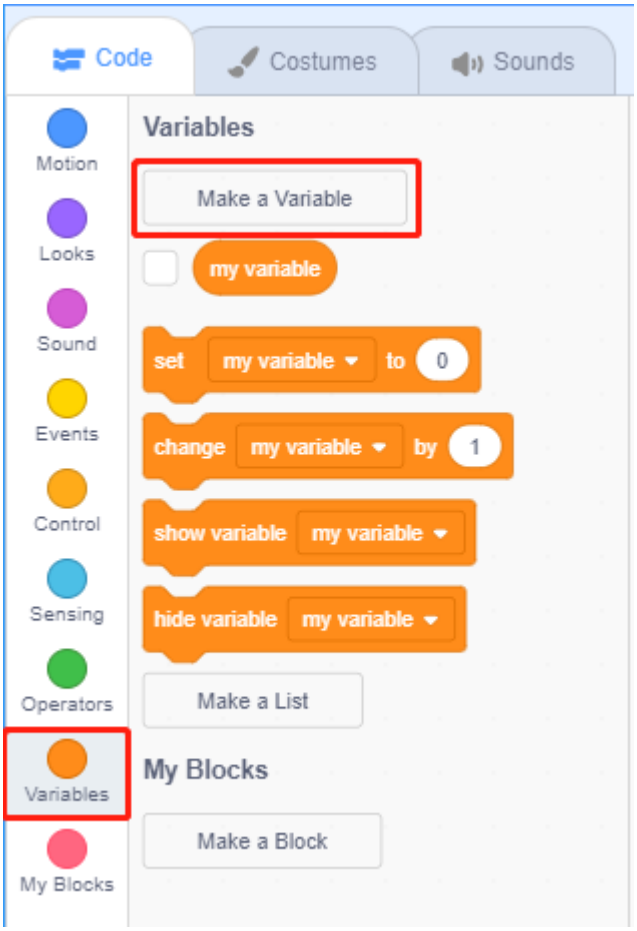
ここでは、2 つの **Arrow1** sprite が必要です。これは、複製ボタンを利用し完成できます。



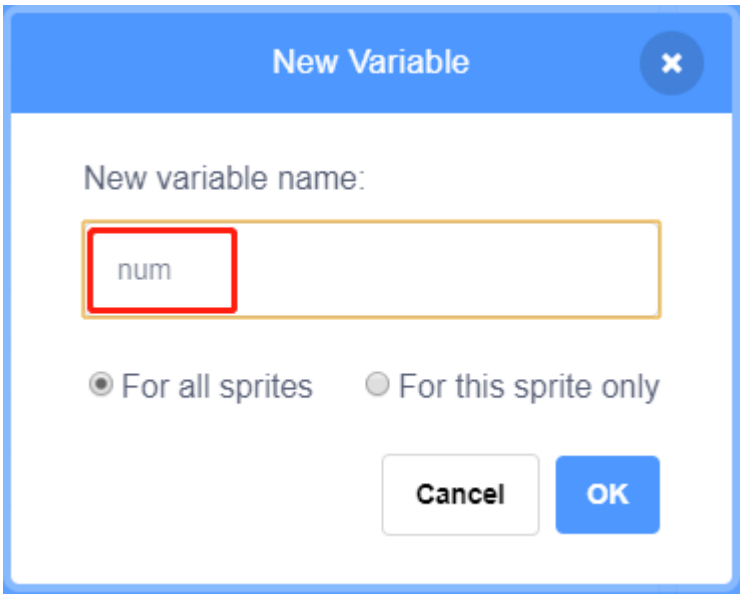
Arrow 2 の Sprite をクリックし、costume2 を選択して矢印の方向を変更します。



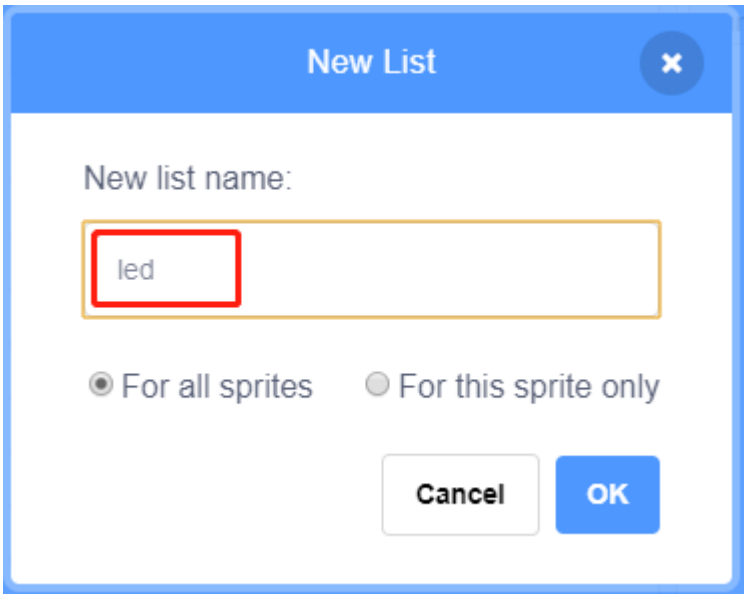
それでは、変数を作成しましょう。



num という名前を付けます。



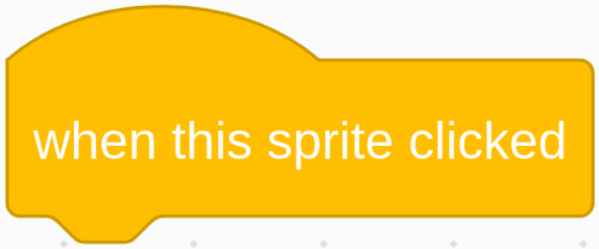
同じ方法に従って、 **led** というリストを作成します。



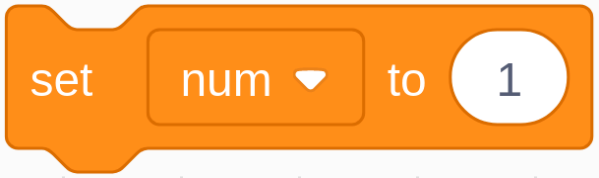
追加すると、stage area に num 変数と led リストが表示されます。

+ をクリックして 10 個のリスト項目を追加し、ピン番号を順番に入力します (17、18、27、22、23、24、25、2、3、8)。

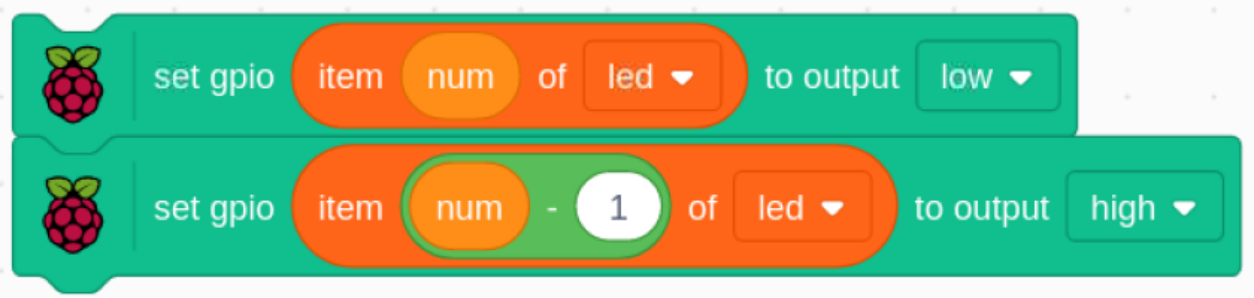
コードに関するヒント



これは、現在の sprite がクリックされたときにトリガーされるイベントブロックです。



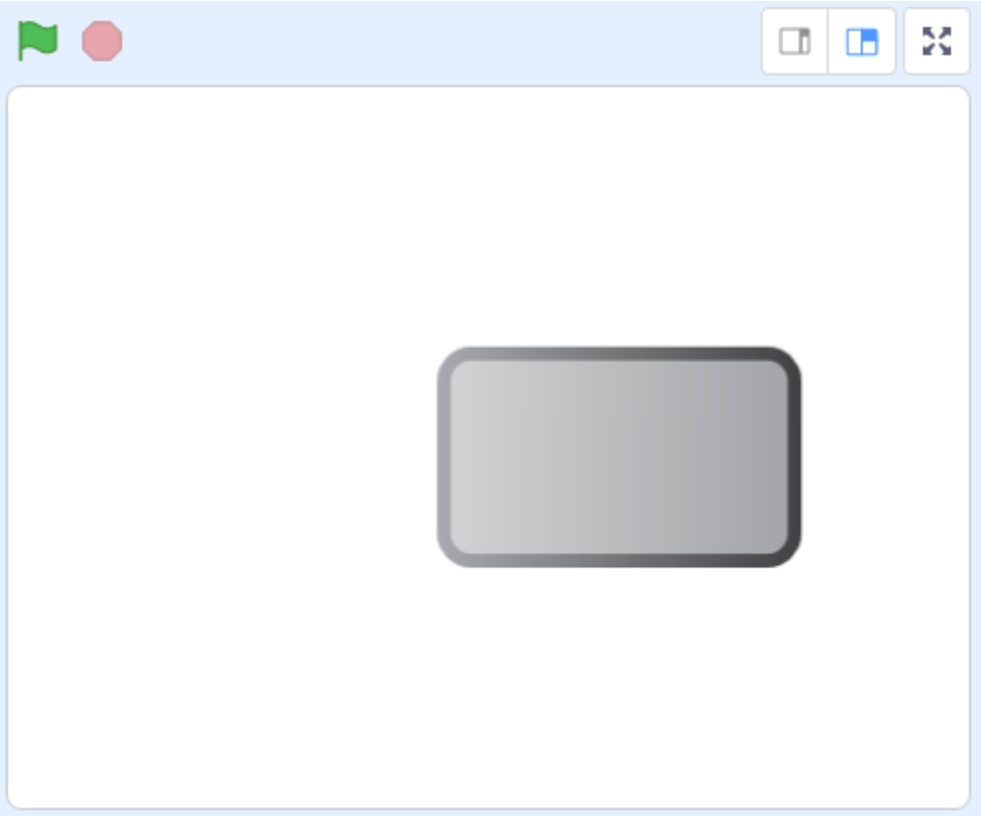
num 変数の初期値は、どの LED が最初にオンになるかを決定します。



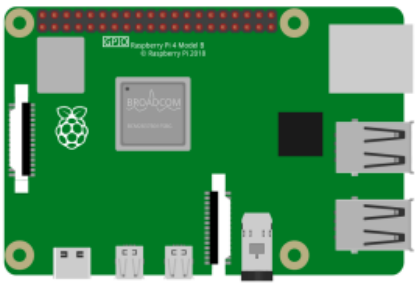
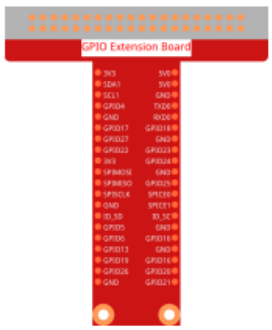
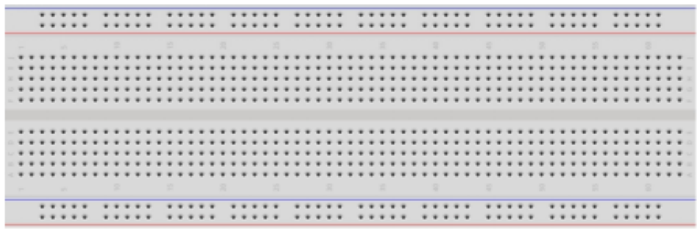
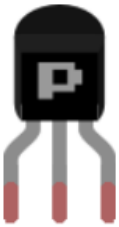
LED リストの num に対応するピンを low に設定して LED を点灯させ、次に num-1 に対応するピンを high に設定して前の LED をオフにします。

1.7 ドアベル

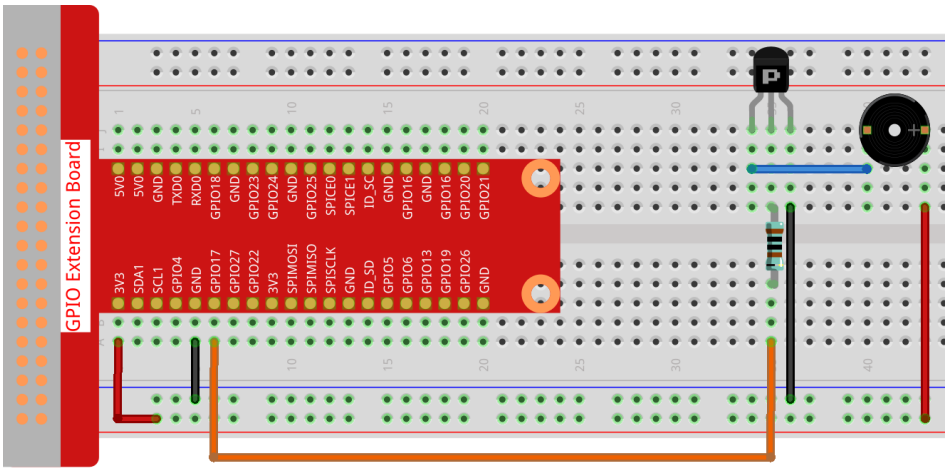
今日はドアベルを作り、ステージ上の button3　sprite をクリックすると、ブザーが鳴ります。 もう一度クリックすると、ブザーが鳴り止みます。



必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Active Buzzer
		1 * Resistor(1kΩ)
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * S8550 PNP Transistor 

回路を構築する



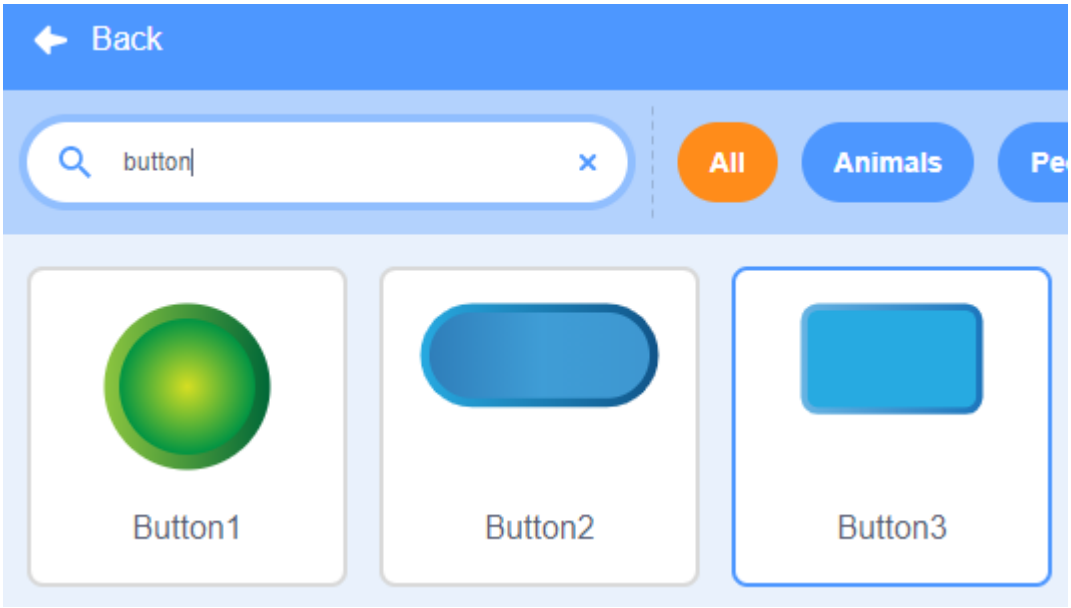
コードをロードして、何が起こるかを確認します

コードファイル (`1.7_doorbell.sb3`) を Scratch3 にロードします。

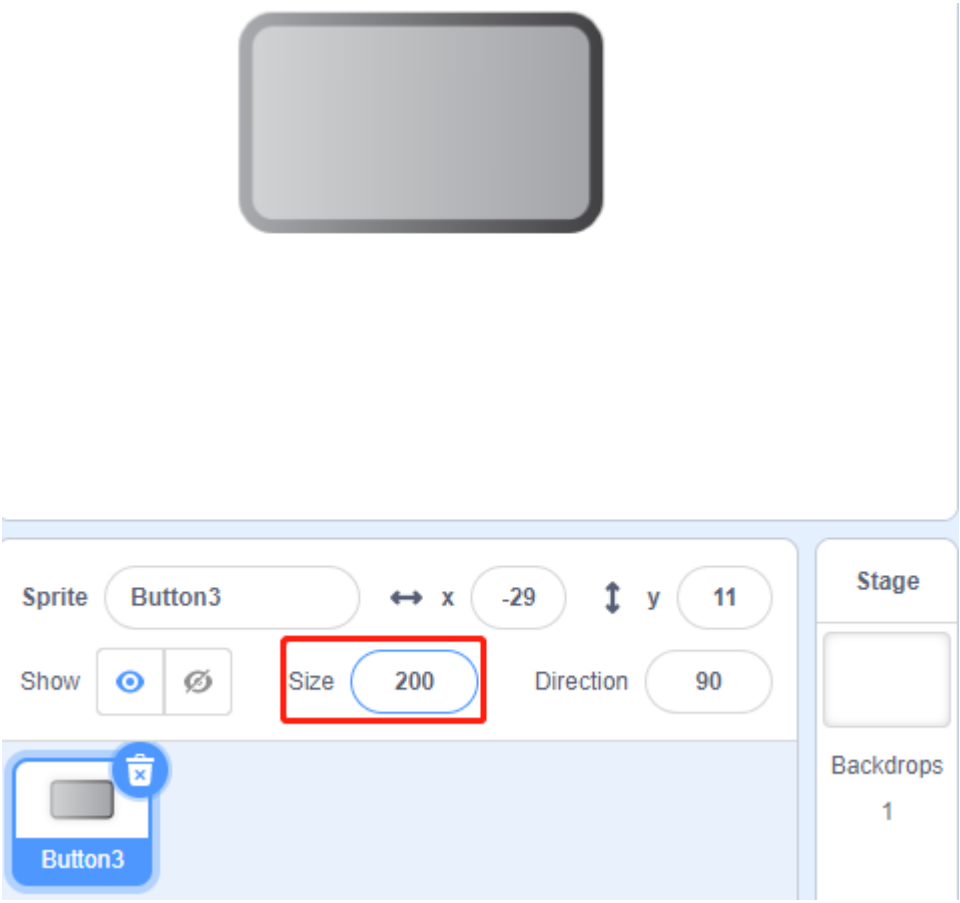
ステージ上の緑色の旗をクリックします。 ボタン 3 の sprite をクリックすると、青色に変わり、ブザーが鳴ります。 もう一度クリックすると、Button3 スプライトが灰色に戻り、ブザーが鳴り止みます。

Sprite に関するヒント

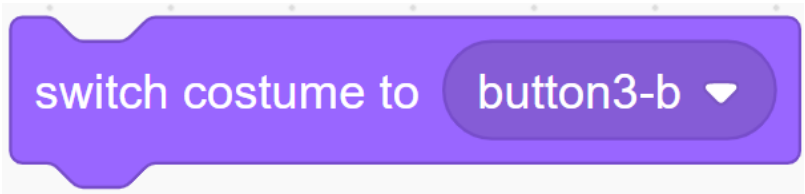
デフォルトの sprite を削除してから、Button 3 の sprite を選択します。



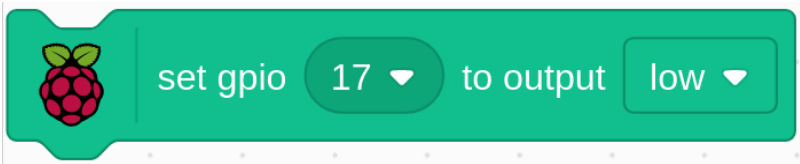
次に、サイズを 200 に設定します。



コードに関するヒント



このブロックを使用すると、sprite の costume を切り替えることができます。

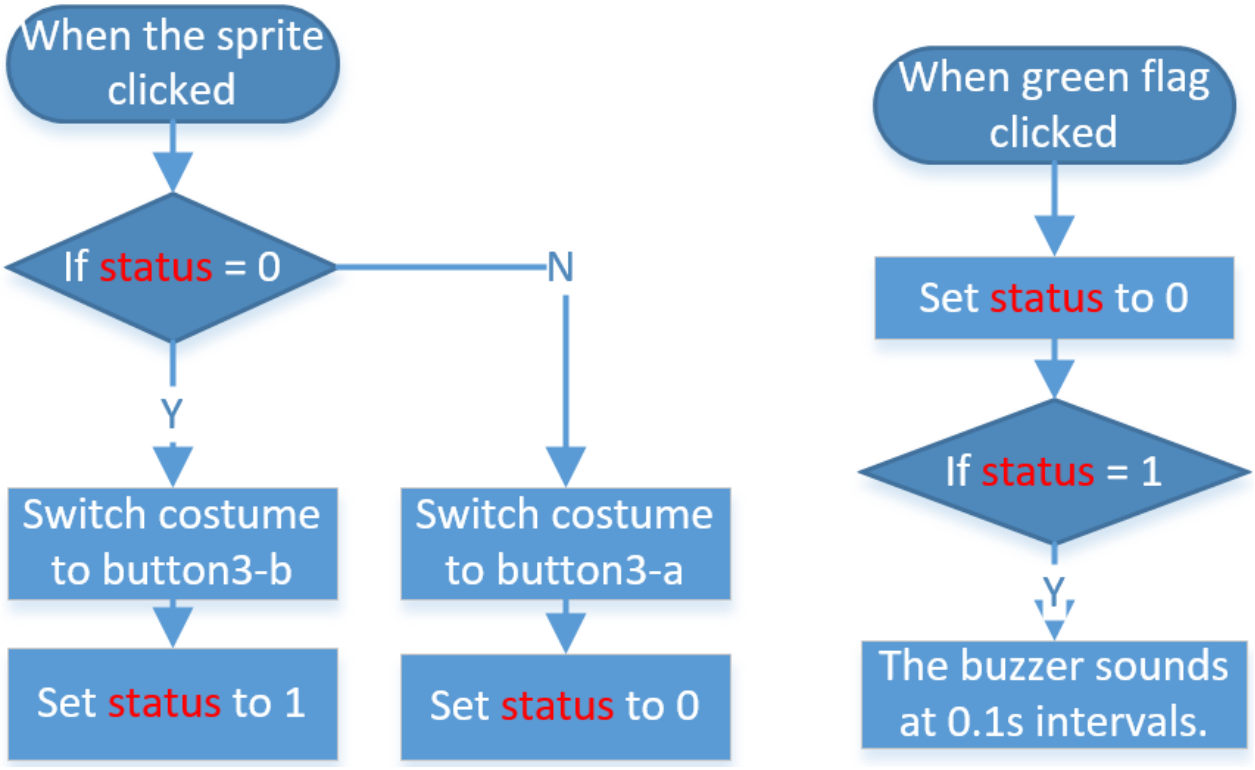


gpio17 を low に設定して、ブザーを鳴らします。 high 設定するとブザーが鳴りません。

ここでは status スイッチを使用します。ここでは、コード全体を理解するのに役立つ flowchart を使用します。

緑のフラグがクリックされると、status が最初に 0 に設定され、この時点で sprite がクリックされるのを待ちます。button3 sprite をクリックすると、button-b costume（青）として costume に切り替わり、status が 1 に設定されま

す。メインプログラムが status を受信すると 1 として、0.1 秒間隔でブザーを鳴らします。button3 をもう一度クリックすると、button-a costume（灰色）に切り替わり、status が再び 0 に設定されます。



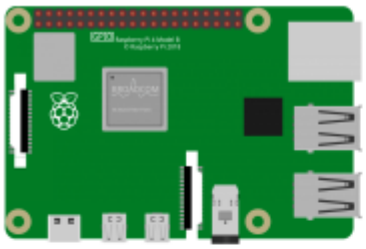
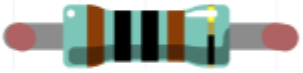
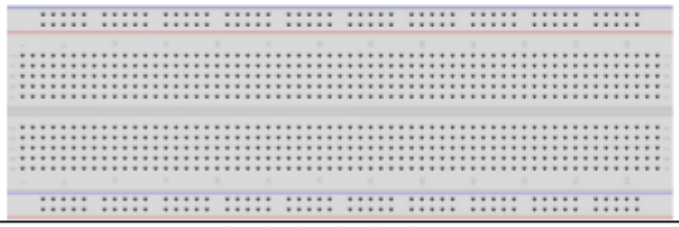
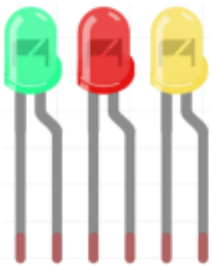
1.8 だるまさんがころんだ

今日はだるまさんがころんだのゲームをします。

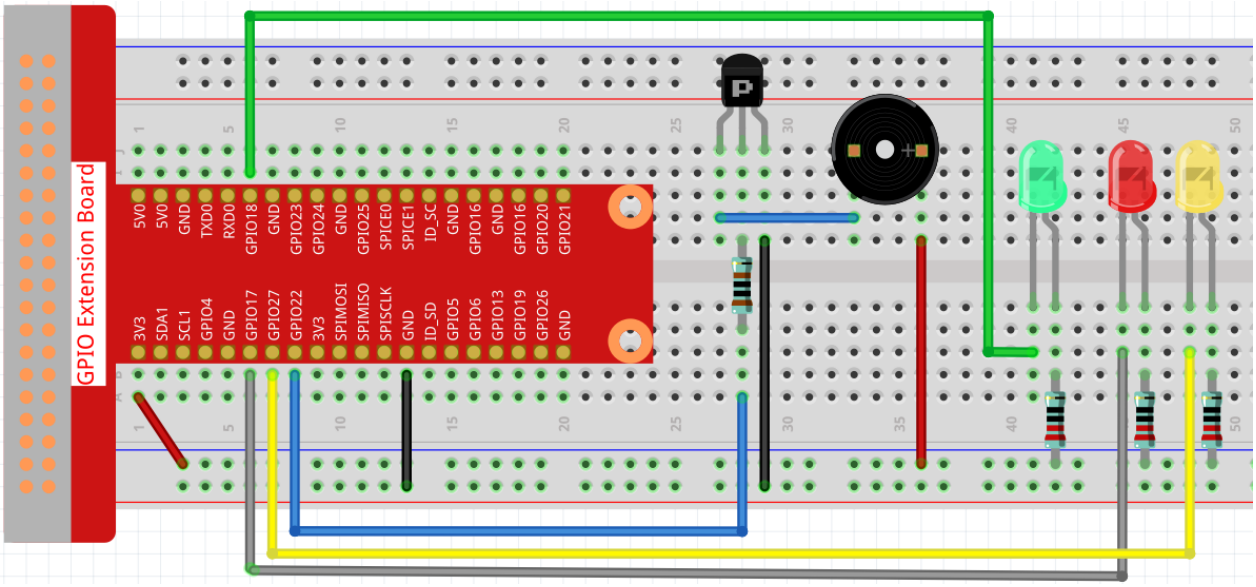
緑の旗をクリックしてゲームを開始し、キーボードの右矢印キーを押したままにして sprite を右に動かします。 緑色のライトが点灯している場合、スプライトは移動できます。 ただし、赤い LED が点灯している場合は、スプライトの移動を停止する必要があります。 そうしないと、ブザーが鳴り続けます。

status
1

必要な部品

1 * Raspberry Pi 	1 * T-Extension Board	3 * Resistor(220Ω)
		1 * Passive Buzzer
1 * 40-pin Cable		1 * Resistor(1kΩ) 
1 * Breadboard 		1 * S8550 PNP Transistor
3 * LED 		Several Jumper Wires

回路を構築する



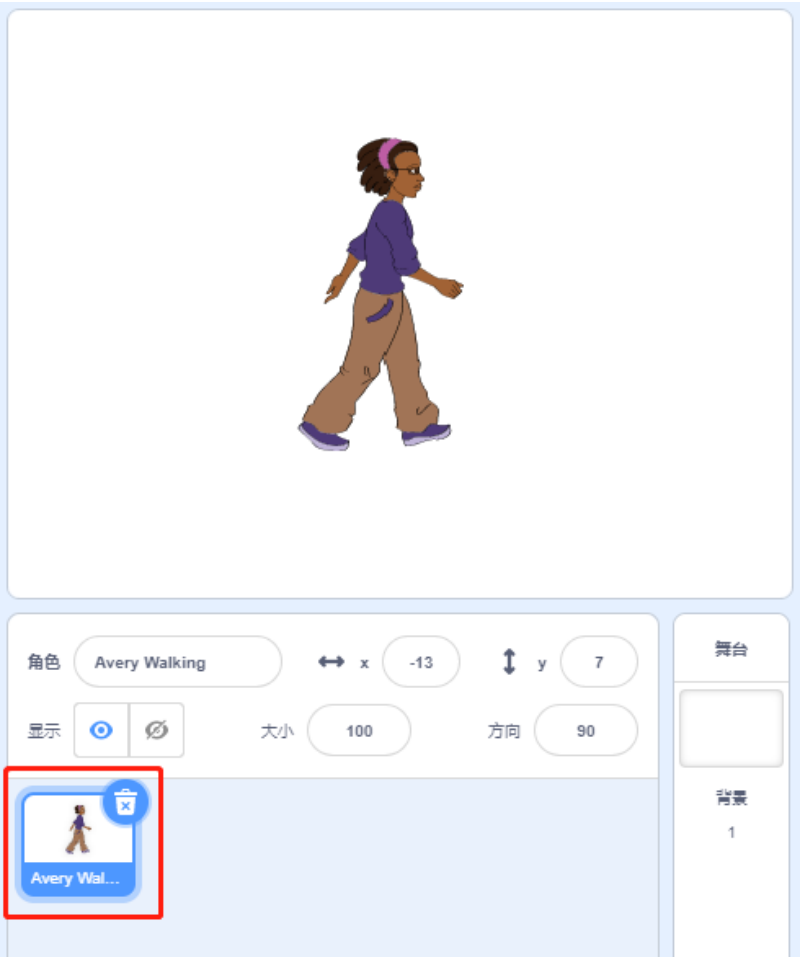
コードをロードして、何が起こるかを確認します

コードをロードして、何が起こるかを確認しますコードファイル (`1.8_123_wooden_man.sb3`) を Scratch3 にロードします。

緑色の LED が点灯している場合は、右矢印キーを使用して Avery を制御して右に歩くことができます。 赤い LED が点灯しているときに、Avery を右に動かし続けると、アラームが鳴ります。

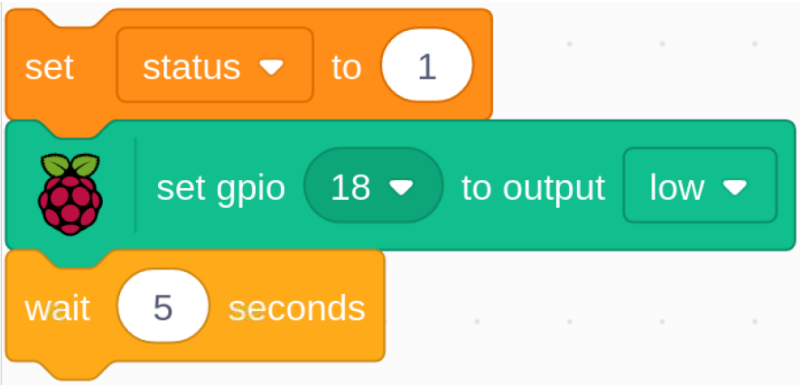
Sprite のヒント

デフォルトの Sprite を削除してから、Avery Walking sprite を選択します。

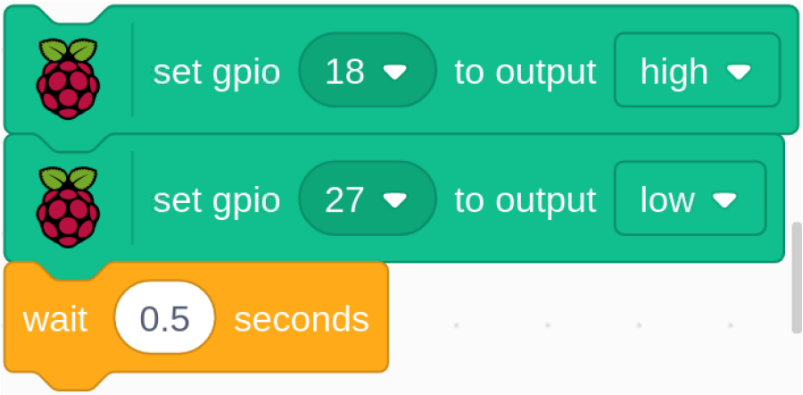


コードに関するヒント

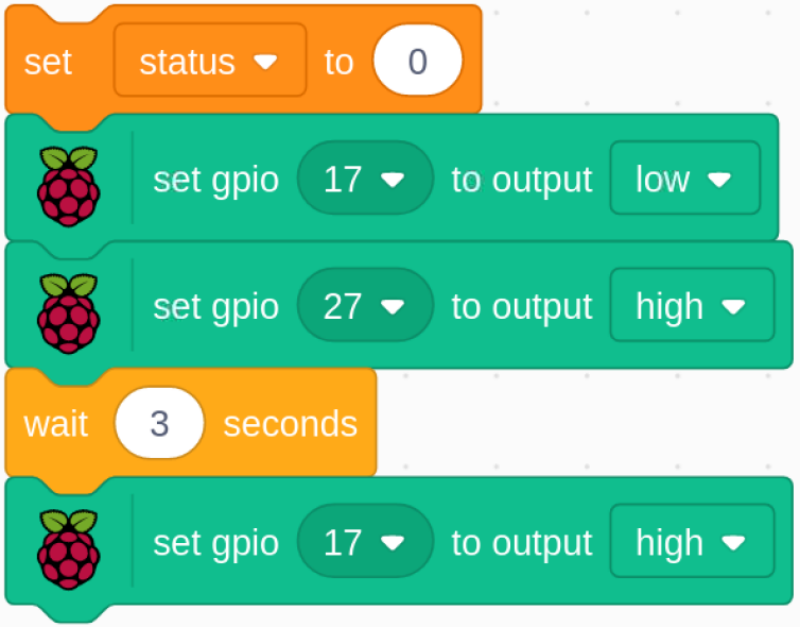
すべてのピンを High に初期化します。



ゲームが開始したら、ステータス変数を 1 に割り当てて、Avery Walking sprite が移動可能であることを示します。次に、GPIO18 を低に設定します。これにより、緑色の LED が 5 秒間点灯します。

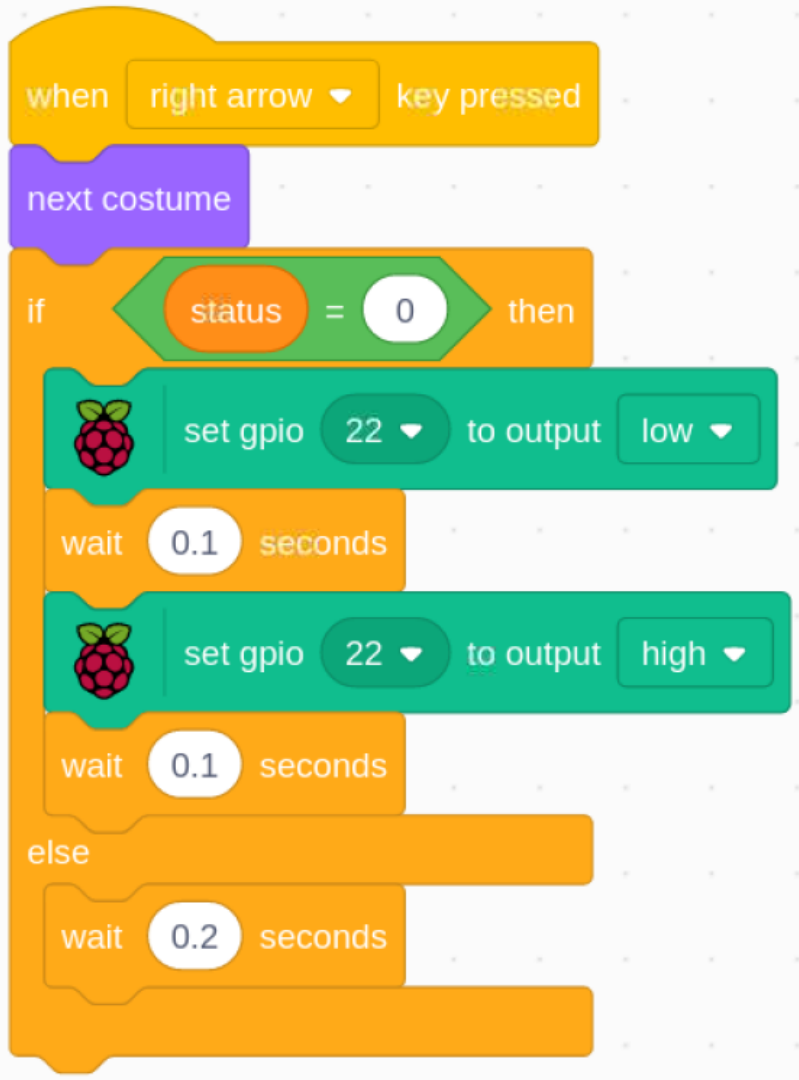


GPIO18 を高に設定してから、GPIO27 を低に設定します。これは、緑色の LED をオフにし、黄色の LED を 0.5 秒間点灯させることを意味します。



ステータス変数を 0 に割り当てます。これは、Avery Walking sprite が移動していないことを意味します。 次に、

GPIO27 を low に設定し、GPIO17 を high に設定します。これにより、黄色の LED がオフになり、次に赤色の LED が 3 秒間点灯します。 最後に、GPIO17 を high に設定して、赤い LED をオフにします。

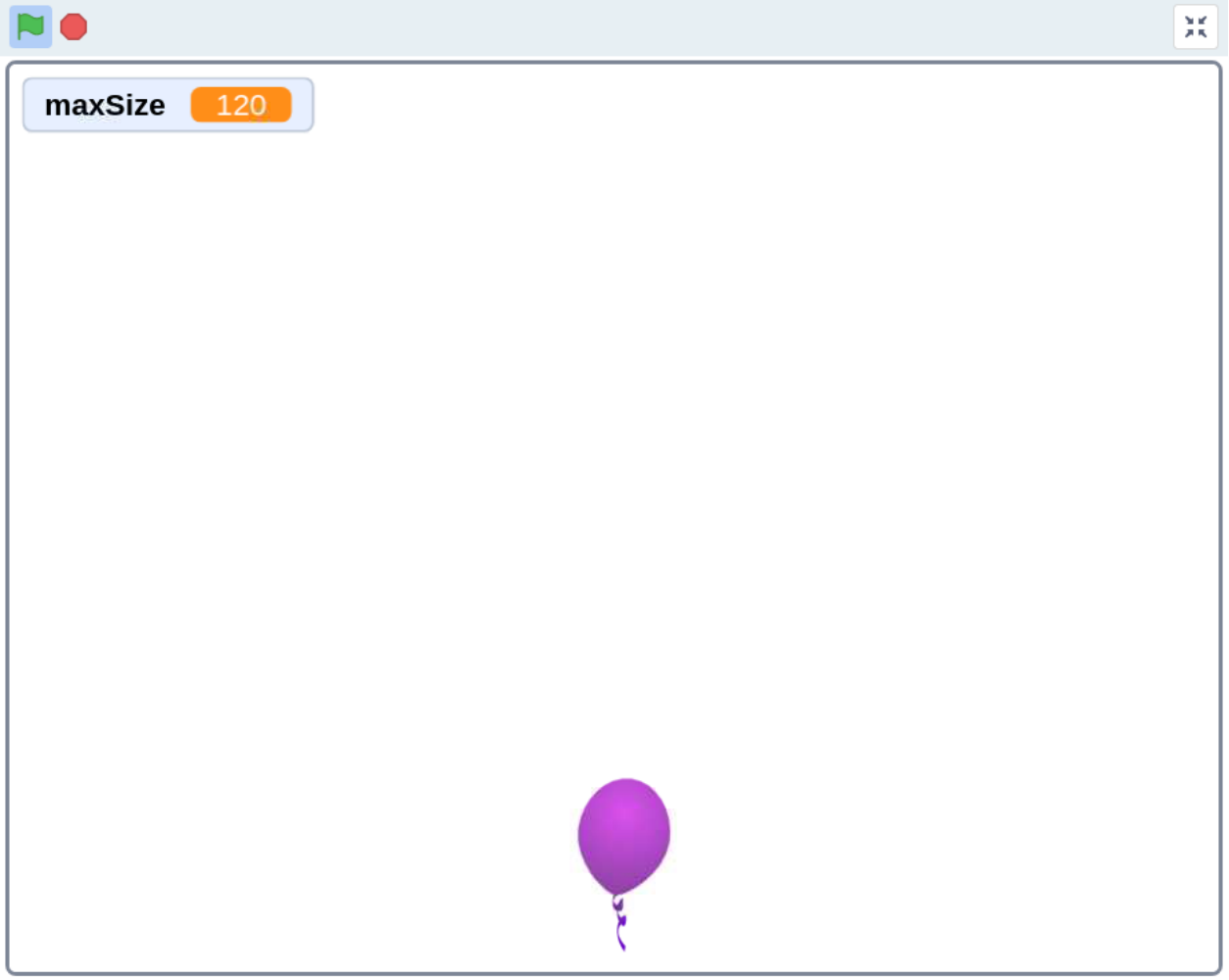


キーボードの right arrow キーを押すと、Avery が右に歩いているのが見えるように、Avery Walking sprite を次の costume に切り替える必要があります。 次に、status 変数の値を決定する必要があります。 0 の場合は、現時点で Avery Walking sprite が動いていないことを意味し、right arrow キーを再度押すことができないことを警告するブザーが鳴ります。

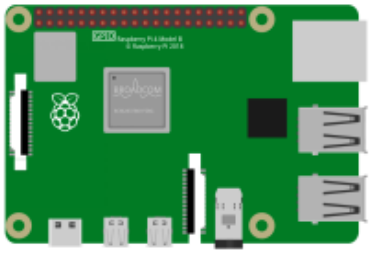
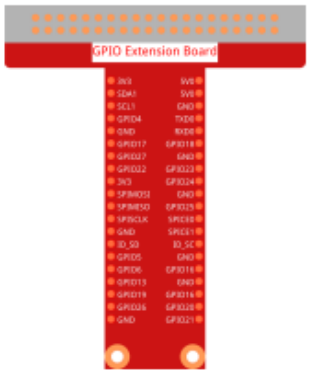
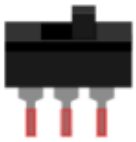
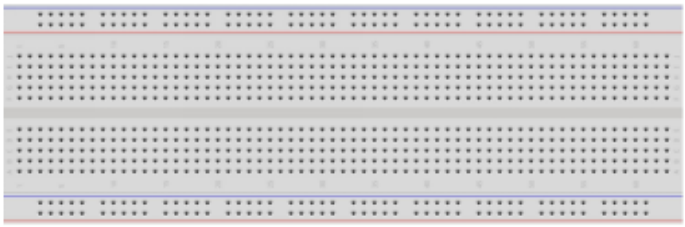
1.9 風船を膨らませる

ここでは、バルーニングのゲームをします。

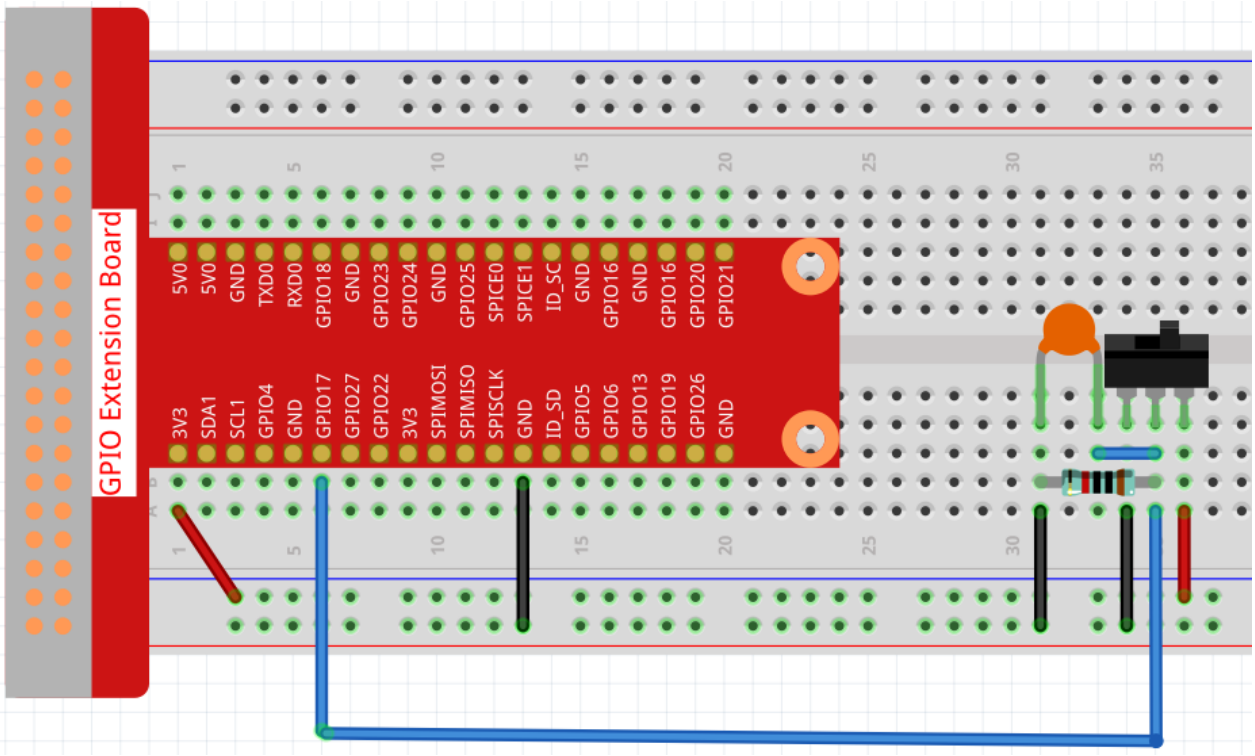
スライドを左に切り替えてバルーンを膨らませ始めると、この時点でバルーンはどんどん大きくなります。 風船が大きすぎると爆発します。 気球が小さすぎると空中に浮きません。 ポンプを停止するには、スイッチを右に切り替えるタイミングを判断する必要があります。



必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Slide Switch 
		1* 104 Capacitor
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor(10kΩ)

回路を構築する



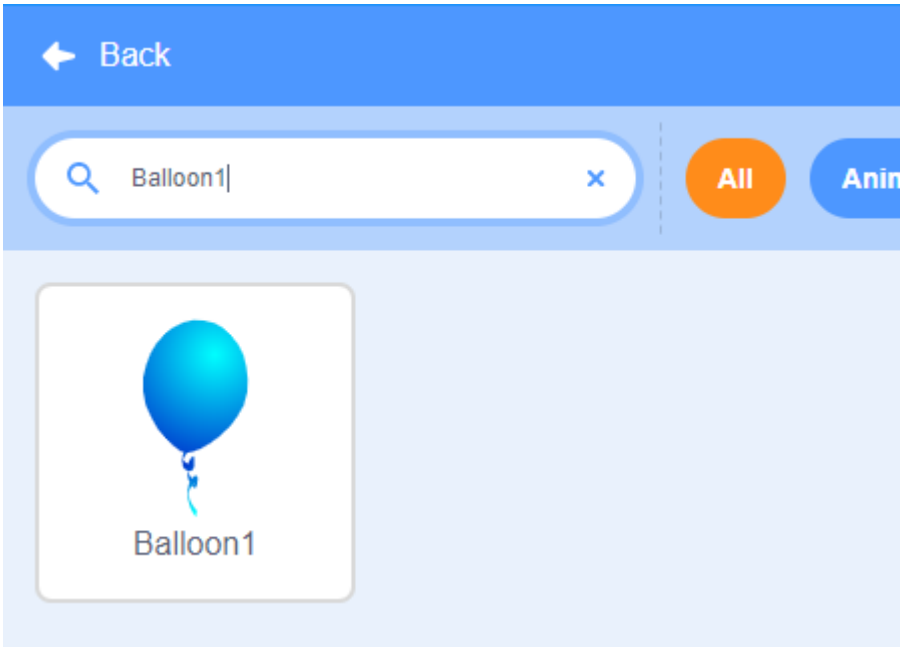
コードをロードして、何が起こるかを確認します

コードファイル (`1.9_inflating_the_balloon.sb3`) を Scratch3 にロードします。

スライダーを左に切り替えてバルーンを膨らませ始めると、この時点でバルーンはどんどん大きくなります。 風船が大きすぎると爆発します。 気球が小さすぎると空中に浮きません。 ポンプを停止するには、スイッチを右に切り替えるタイミングを判断する必要があります。

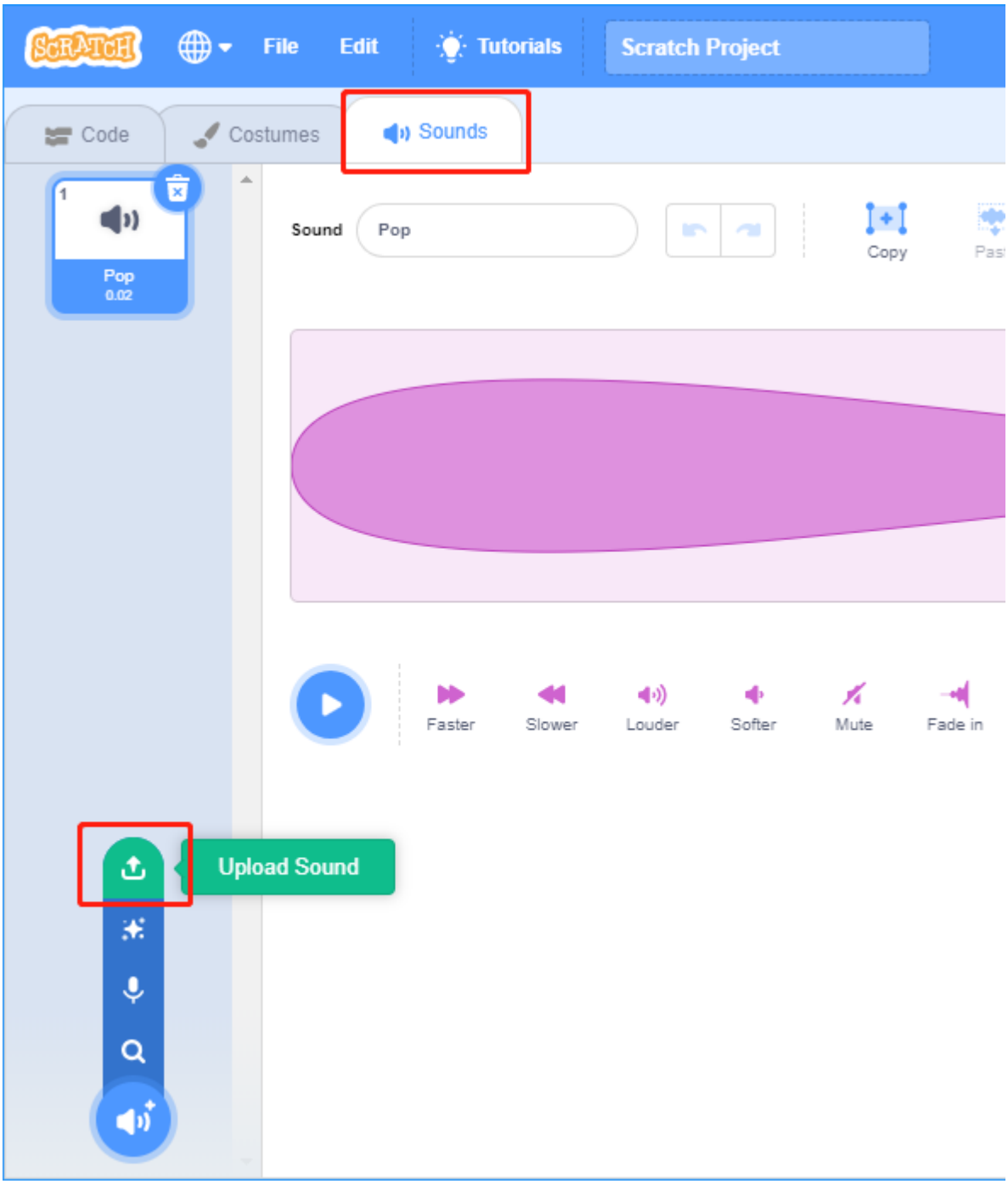
Sprite に関するヒント

前の Sprite1 sprite を削除してから、Balloon1 sprite を追加します。

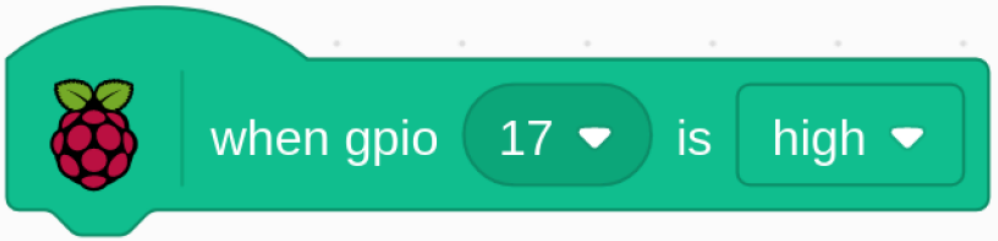


このプロジェクトでは風船爆発効果音を使用しているので、どのように追加されたか見てみましょう。

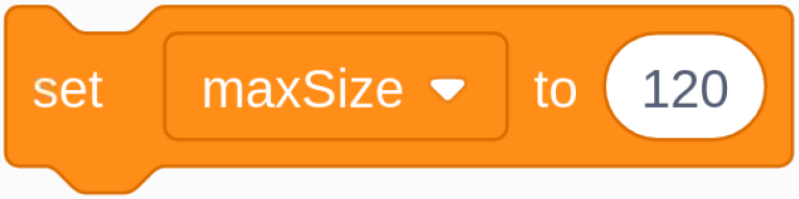
上部の Sound オプションをクリックし、Upload Sound をクリックして、 `boom.wav` を `home/pi/davinci-kit-for-raspberry-pi/scratch/sound` パスから Scratch3 にアップロードします。


File
Edit
Tutorials
Scratch Project
Code
Costumes
Sounds
Pop
0.02
Sound
Pop
Copy
Faster
Slower
Louder
Softer
Mute
Fade in
Upload Sound

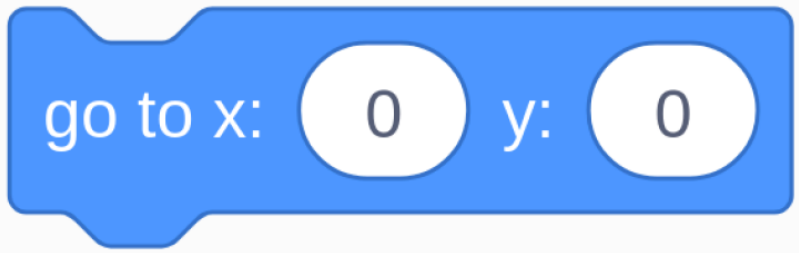
コードに関するヒント



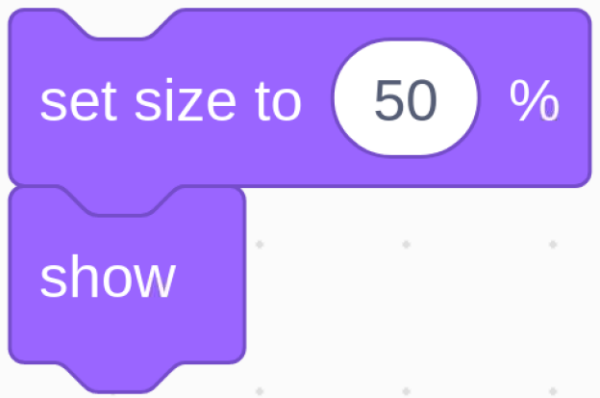
これはイベントブロックであり、トリガー条件は GPIO17 がハイである、つまりスイッチが左に切り替えられることです。



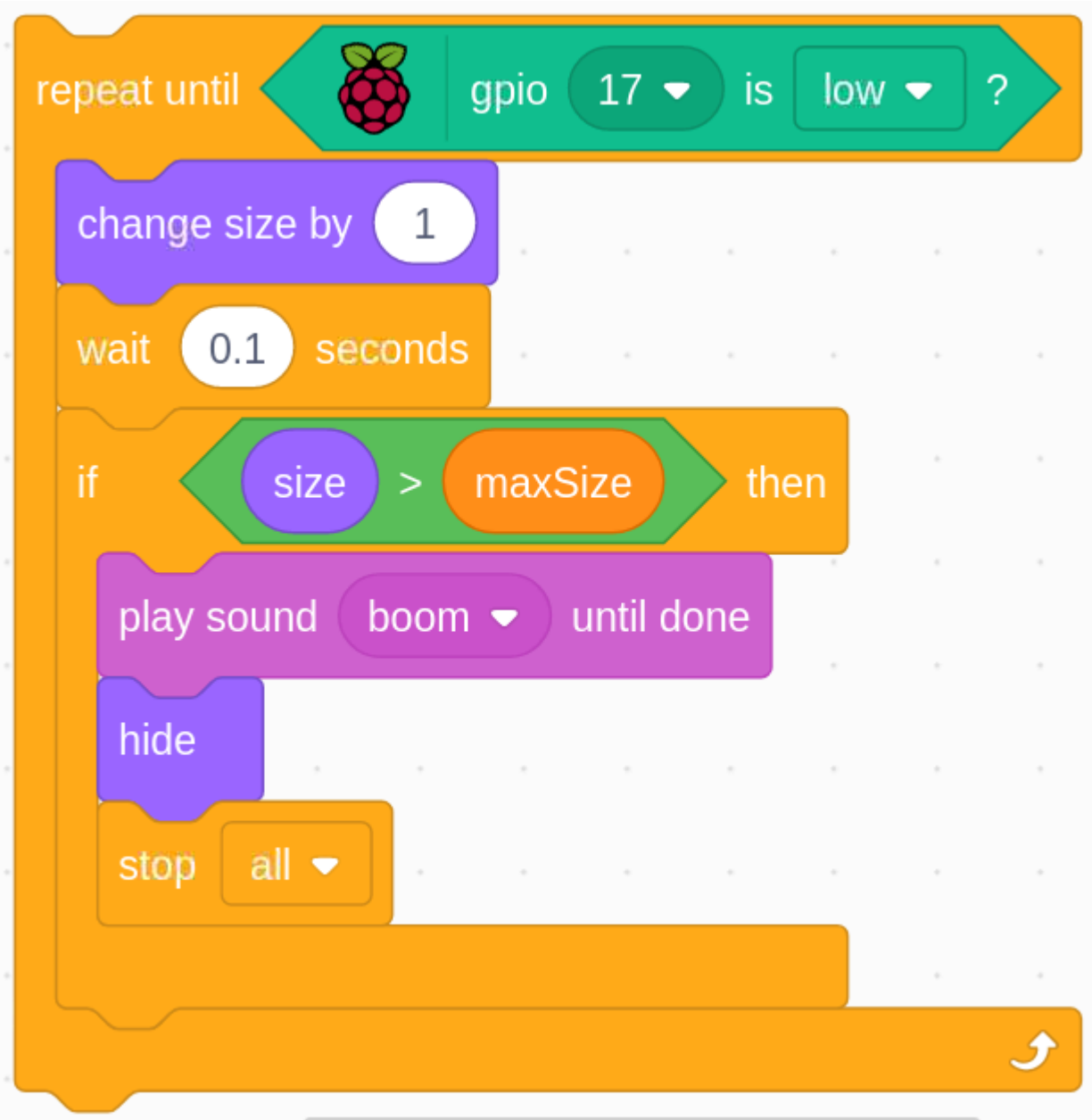
Ballon1 sprite のサイズしきい値を 120 に設定します。



Balloon1 sprite の座標を、stage area の中心である（0,0）に移動します。

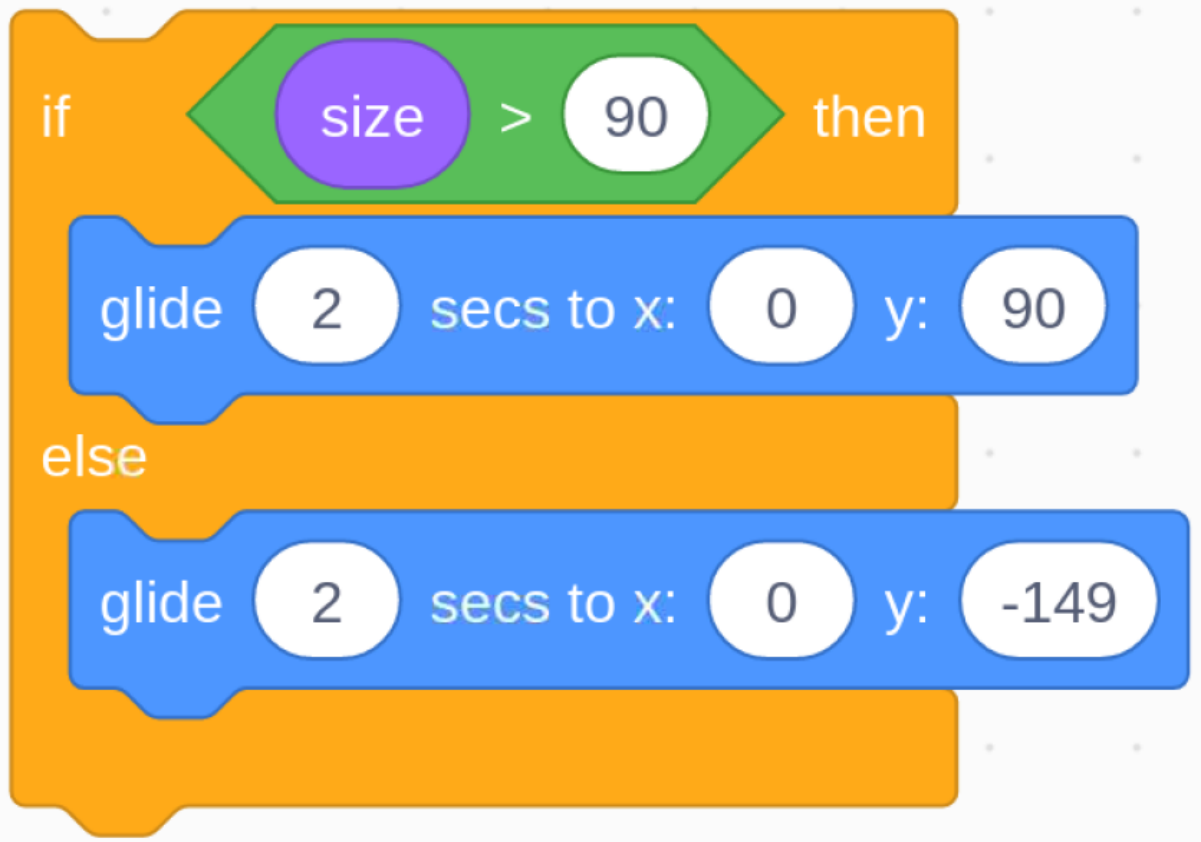


Balloon1 sprite のサイズを 50 に設定し、stage area に表示します。



バルーンを膨らませるループを設定します。このループは、スライダースイッチを右に切り替えると停止します。

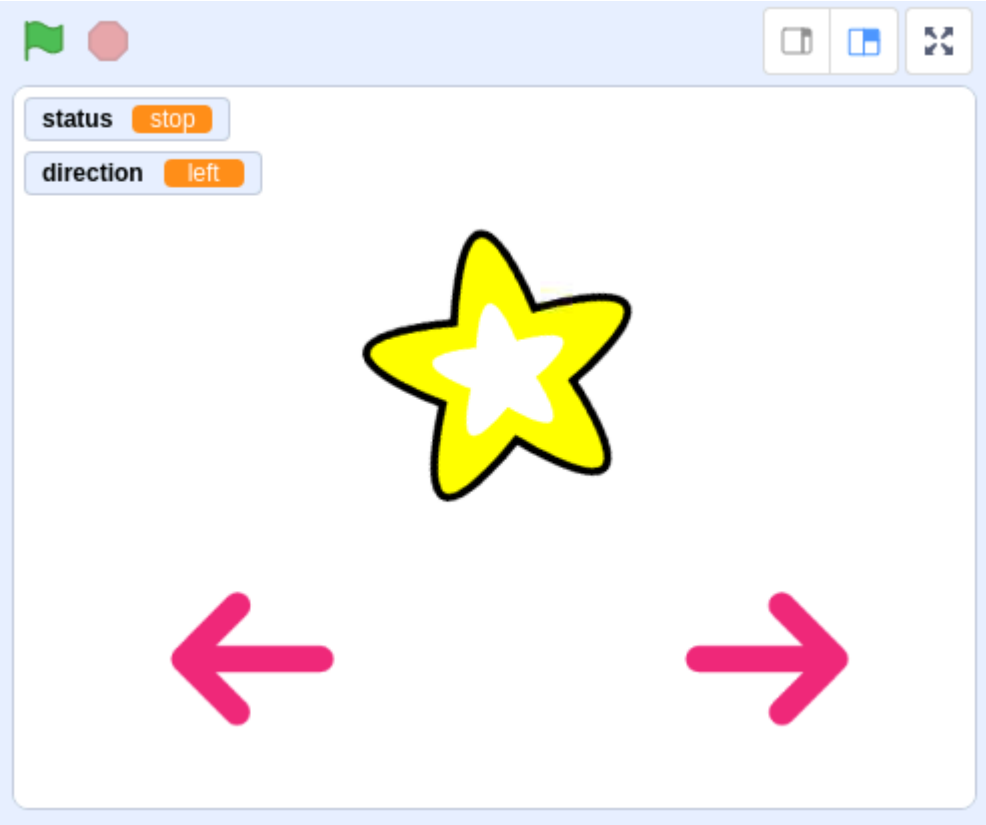
このループ内で、バルーンサイズは 0.1 秒ごとに 1 ずつ増加し、`maxSize` より大きい場合はバルーンが破裂し、その時点でブーム音が鳴り、コードが終了します。



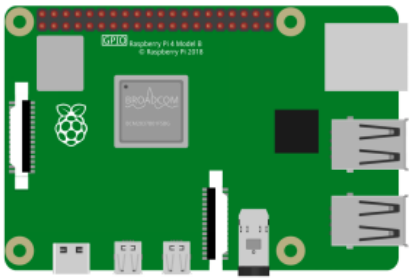
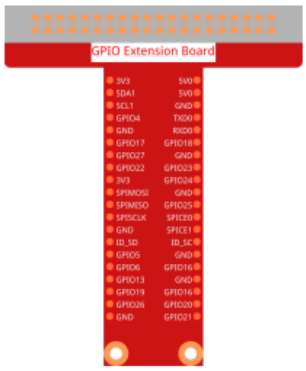
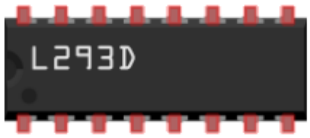
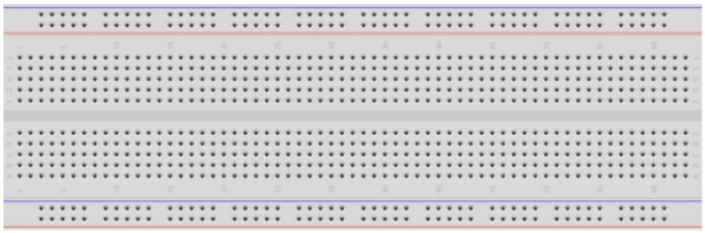
最後のループが終了した後（スライダーが右に切り替わります）、サイズに基づいて Balloon1 sprite の位置を決定します。Balloon1 sprite のサイズが 90 より大きい場合は、持ち上げて（座標を (0,90) に移動します）、そうでない場合は着陸します（座標を (0,-149) に移動します）。

1.10 回転扇

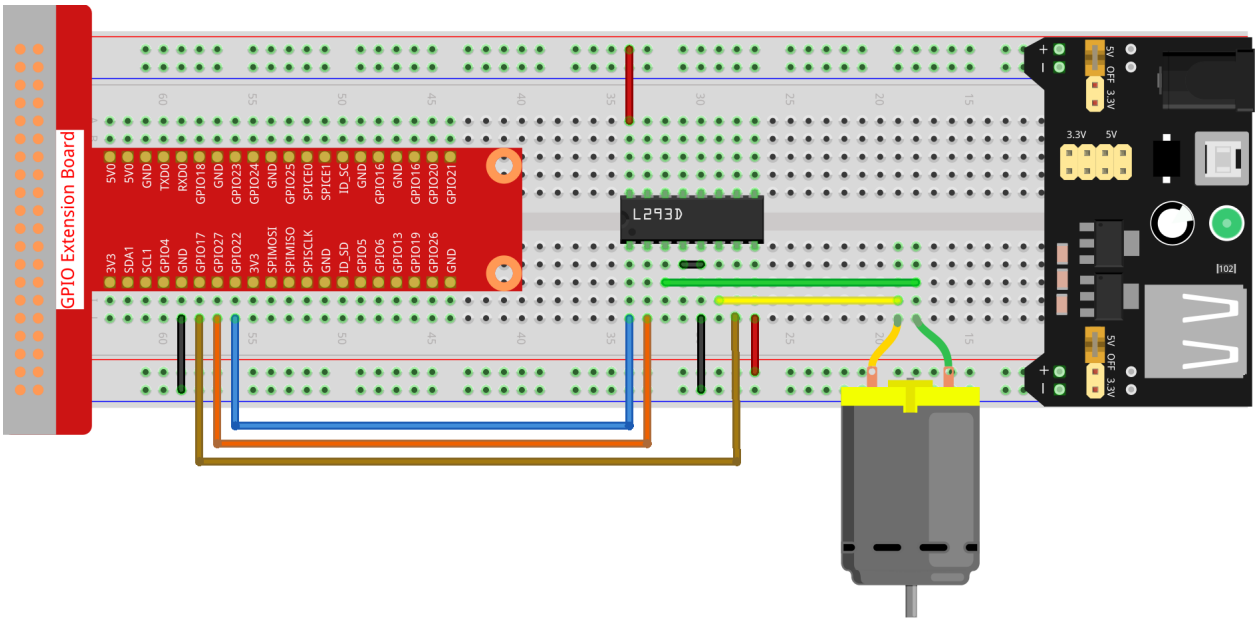
このプロジェクトでは、回転する星の sprite とファンを作成します。


status stop
direction left

必要な部品

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Power Module (with 9V battery and buckle)
1 * 40-pin Cable		1 * L293D 
		Several Jumper Wires
1 * Breadboard 		1 * DC Motor

回路を構築する



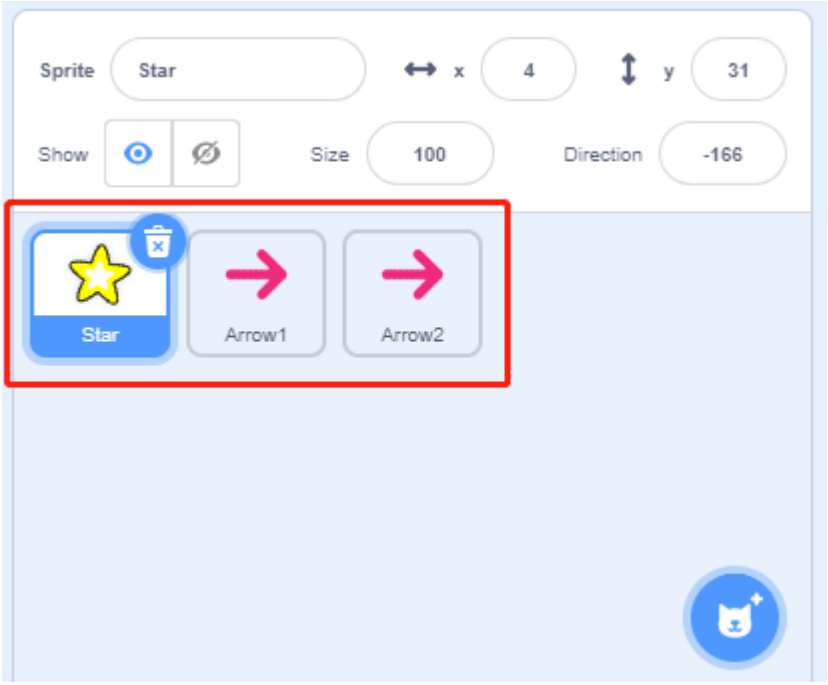
コードをロードして、何が起こるかを確認します

コードファイル (`1.10_rotating_fan.sb3`) を Scratch3 にロードします。

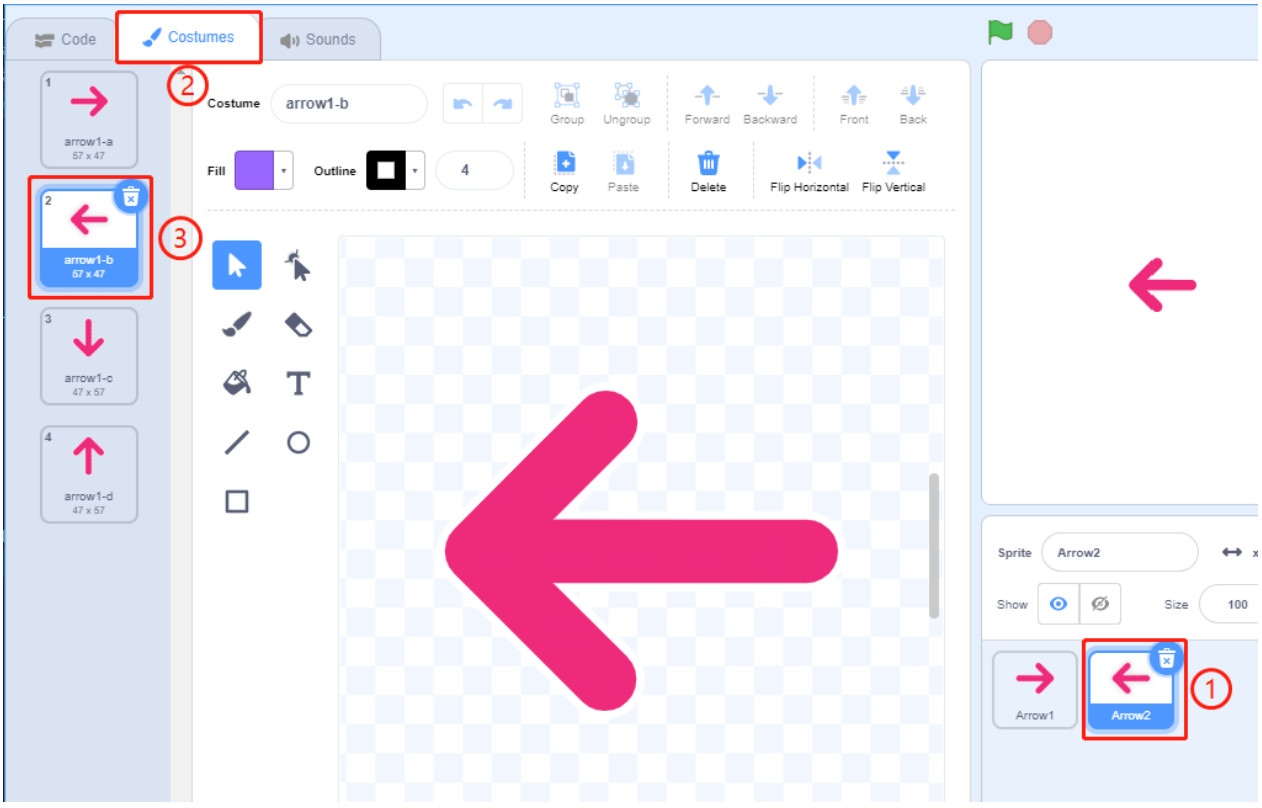
ステージ上の緑色の旗をクリックした後、star sprite をクリックすると、それとモーターが時計回りに回転します。 2 つの arrow sprites をクリックすると、回転方向を変更できます。 star sprite をもう一度クリックすると、star sprite とモーターの回転が停止します。

Sprite に関するヒント

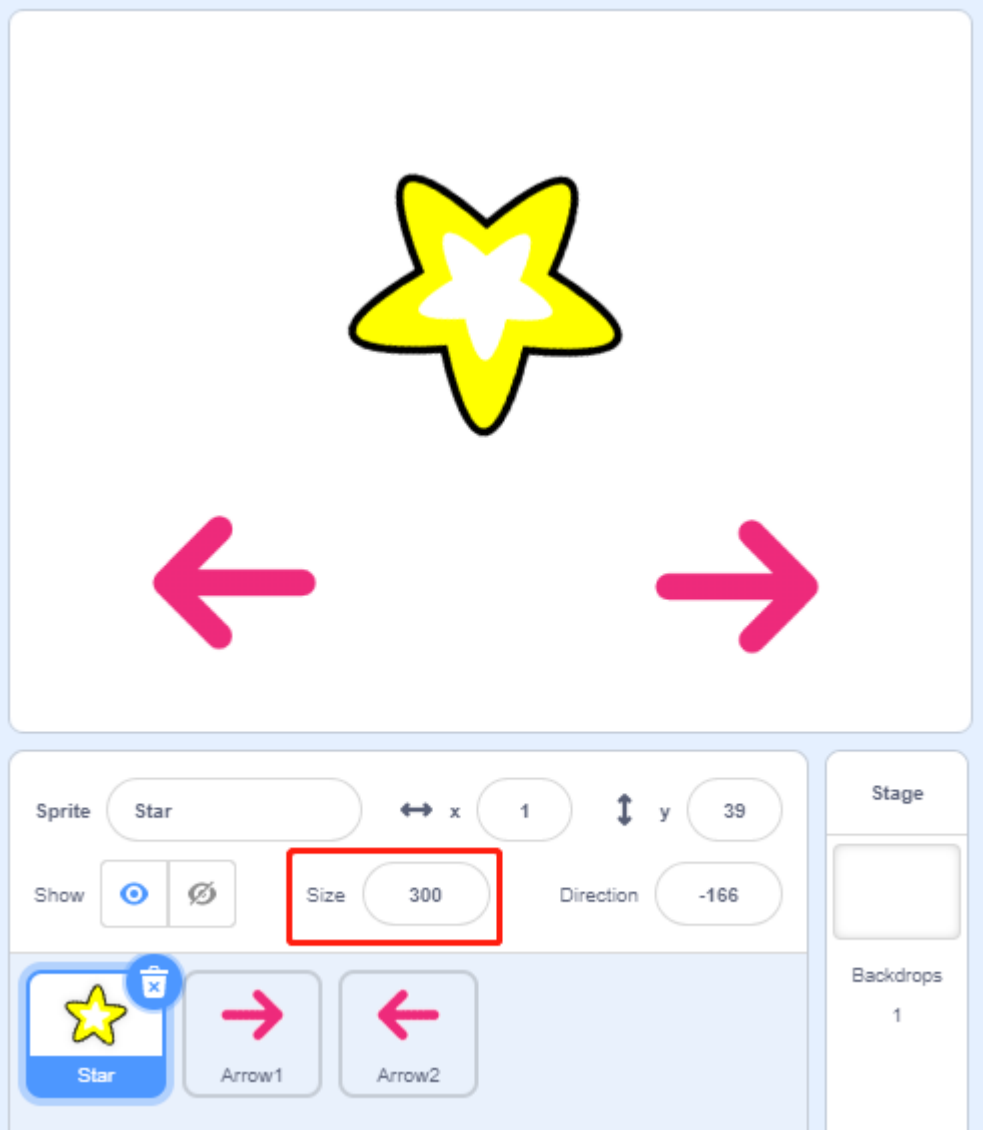
デフォルトの sprite を削除してから、Star sprite と Arrow1 sprite を選択し、Arrow1 を 1 回コピーします。



Costumes オプションで、Arrow2 sprite を別の方向の costume に変更します。

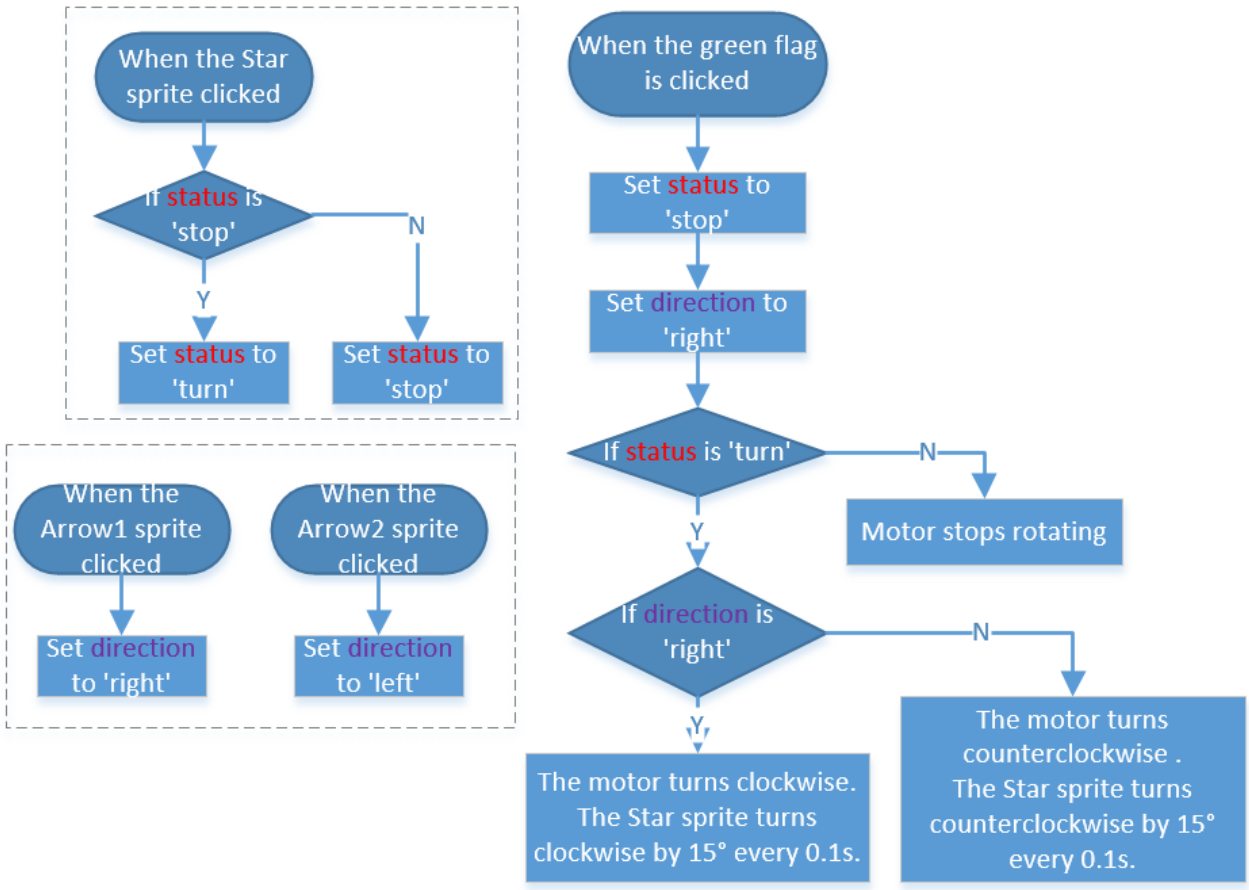


sprite のサイズと位置を適切に調整します。



コードに関するヒント

流れ図

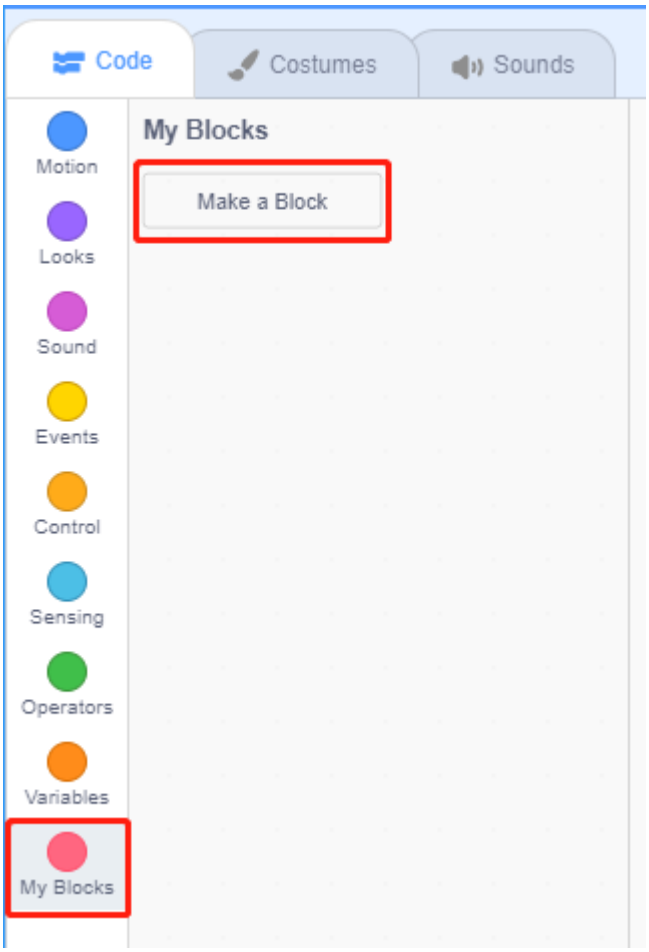


このコードでは、カスタムブロック（関数）である 2 つのピンクのブロックが左に曲がって右に曲がっています。

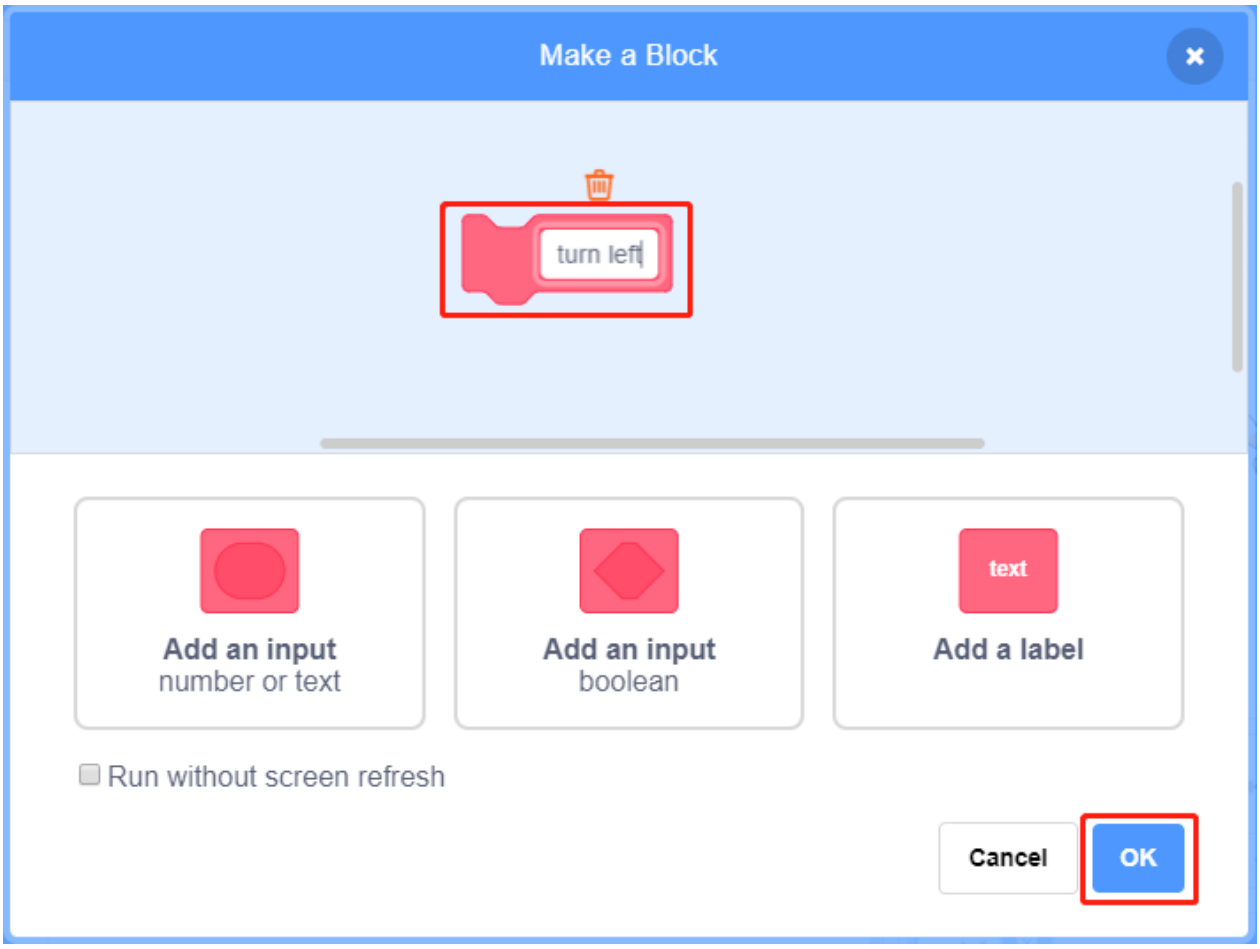
ブロックを作る方法は？

ブロック（関数）の作り方を学びましょう。 ブロック（関数）は、プログラムを単純化するために使用できます。特に同じ操作を複数回実行する場合です。 これらの操作を新しく宣言されたブロックに入れると、非常に便利です。

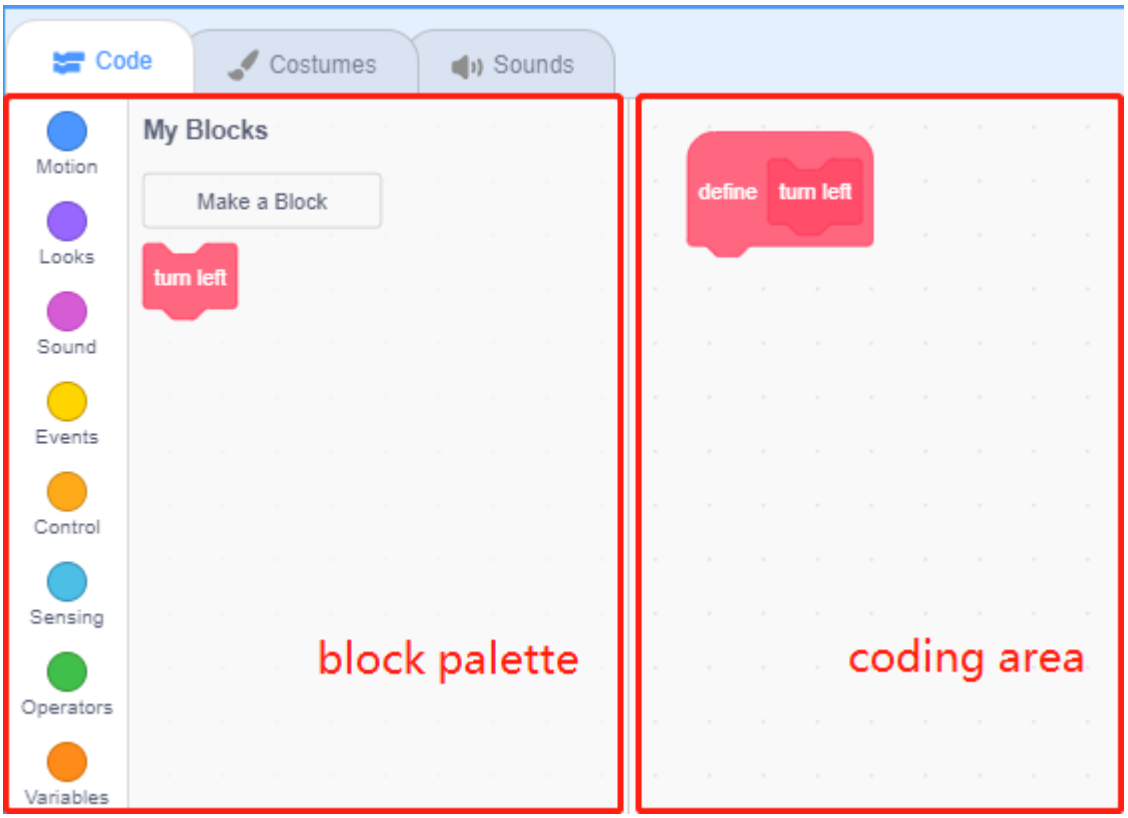
最初に block palette で My Blocks を見つけてから、Make a Block を選択します。



新しいブロックの名前を入力します。

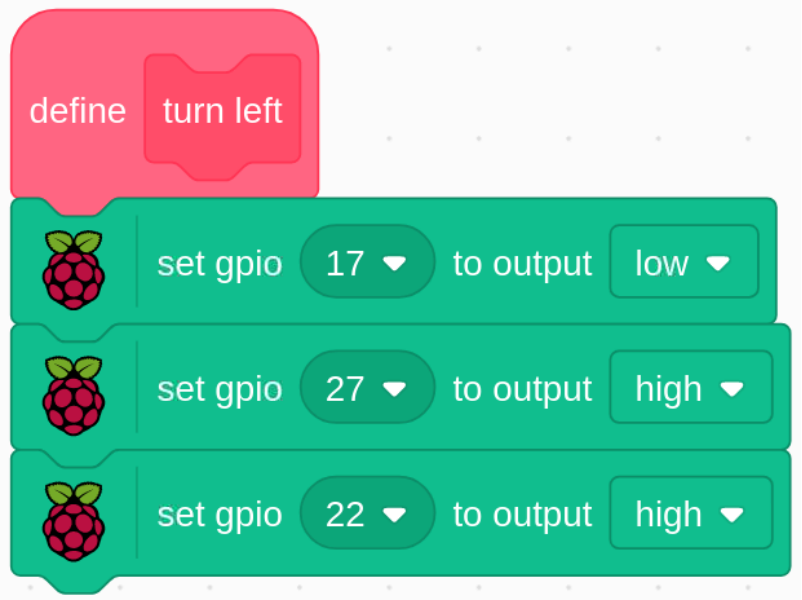


coding area に新しいブロックの機能を書き込んだ後、それを保存すると、blocks palette でブロックを見つけることができます。



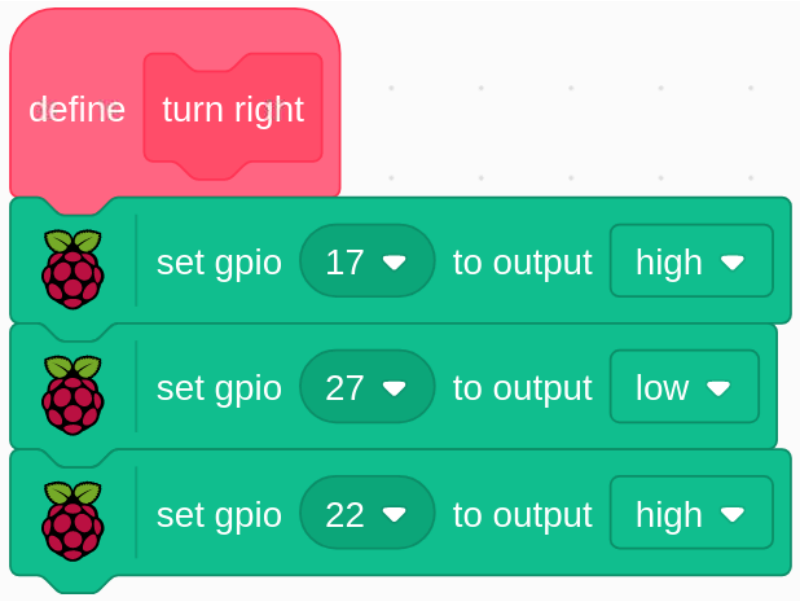
turn left

これは、モーターを反時計回りに回転させるための左折ブロック内のコードです。



turn right

これは、モーターを時計回りに回転させるための右折ブロック内のコードです。



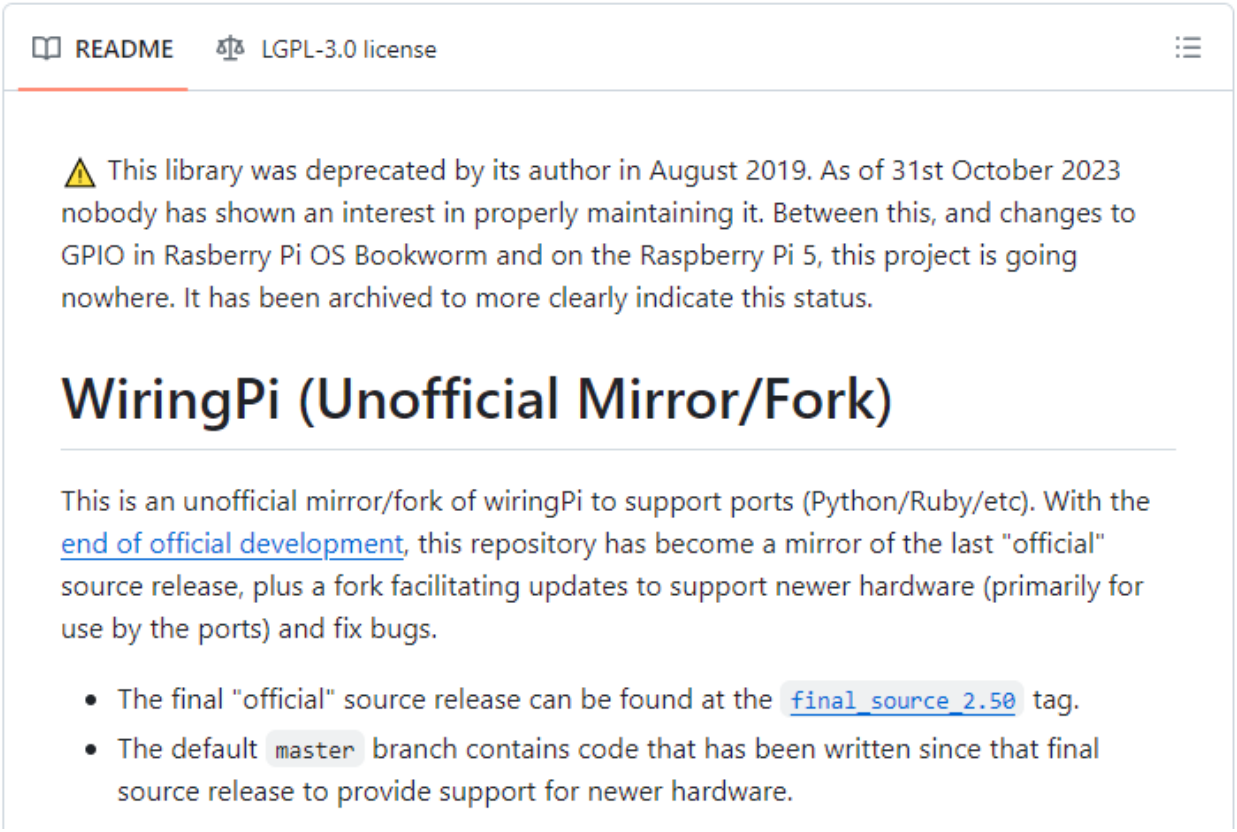
1.15 他言語（Pi 5 用）

Raspberry Pi 5 のリリースにより、より強力なモデルが私たちにもたらされましたが、GPIO に最も顕著な変更が導入されました。

標準の 40 ピンインターフェースは保持されていますが、新しく統合された RP1 サウスブリッジチップとの接続により機能が変更されました。このカスタム RP1 チップは現在、Pi 5 上の周辺機器を処理し、さまざまな互換性の懸念が生じています。

1.15.1 C 言語

C 言語の実装は、wiringPi ライブラリに依存しています。しかし、wiringPi コミュニティライブラリは現在アーカイブされており、更新が行われなくなり、Raspberry Pi 5 プロジェクトには不適切です。詳細については、以下を参照してください：https://github.com/WiringPi/WiringPi



README LGPL-3.0 license

⚠ This library was deprecated by its author in August 2019. As of 31st October 2023 nobody has shown an interest in properly maintaining it. Between this, and changes to GPIO in Rasberry Pi OS Bookworm and on the Raspberry Pi 5, this project is going nowhere. It has been archived to more clearly indicate this status.

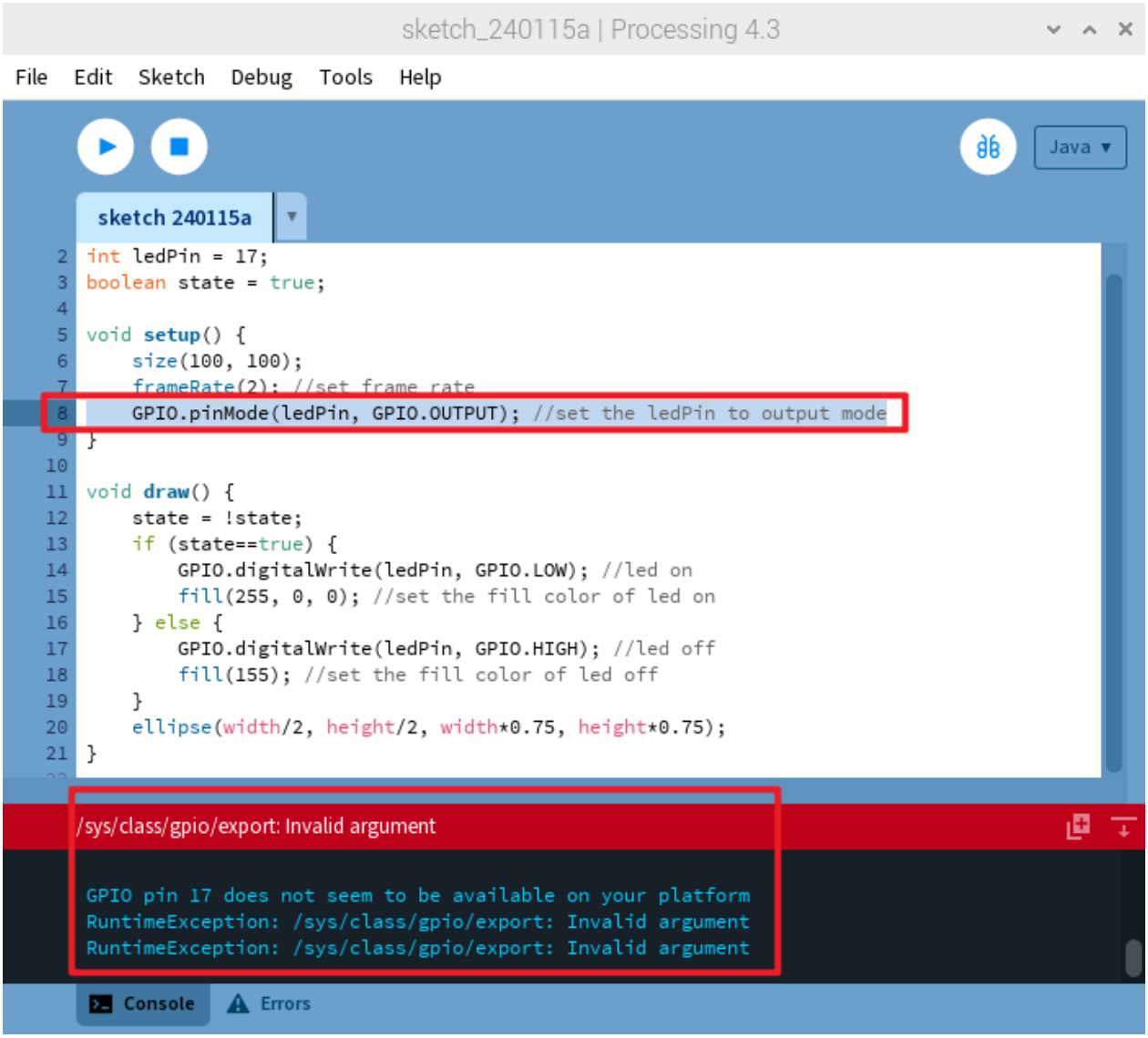
WiringPi (Unofficial Mirror/Fork)

This is an unofficial mirror/fork of wiringPi to support ports (Python/Ruby/etc). With the end of official development, this repository has become a mirror of the last "official" source release, plus a fork facilitating updates to support newer hardware (primarily for use by the ports) and fix bugs.

- The final "official" source release can be found at the `final_source_2.50` tag.
- The default `master` branch contains code that has been written since that final source release to provide support for newer hardware.

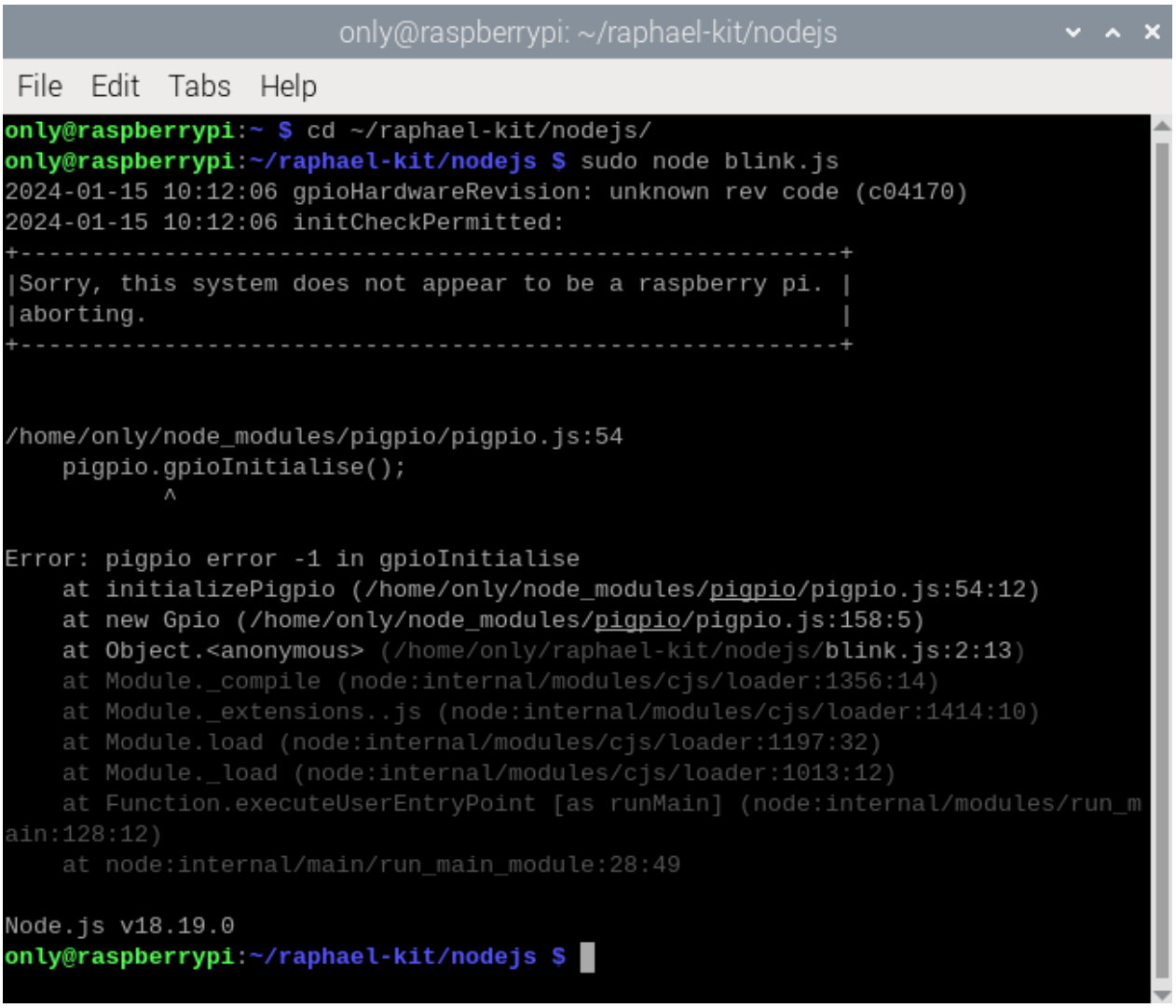
1.15.2 Processing

Raspberry Pi 5 で Processing 4 を使用する際、GPIO プログラミングには課題が伴います。"Invalid argument" や"GPIO pin 17 seems to be unavailable on your platform"などのエラーが GPIO 関連のコードの実行中に発生します（添付画像で示されているように）。詳細については、こちらをご覧ください：https://github.com/benfry/processing4/issues/807



1.15.3 Node.js

Node.js は、現時点で Raspberry Pi 5 をサポートしていない pigpio ライブラリを使用します。さらなる洞察については、こちらを訪れてください：https://github.com/joan2937/pigpio/issues/589



1.15.4 Scratch

64 ビットシステムでは、Raspberry Pi GPIO ライブラリのインポートに問題が発生し、反応しなくなります。詳細については、こちらを訪れてください：https://github.com/raspberrypi/bookworm-feedback/issues/91

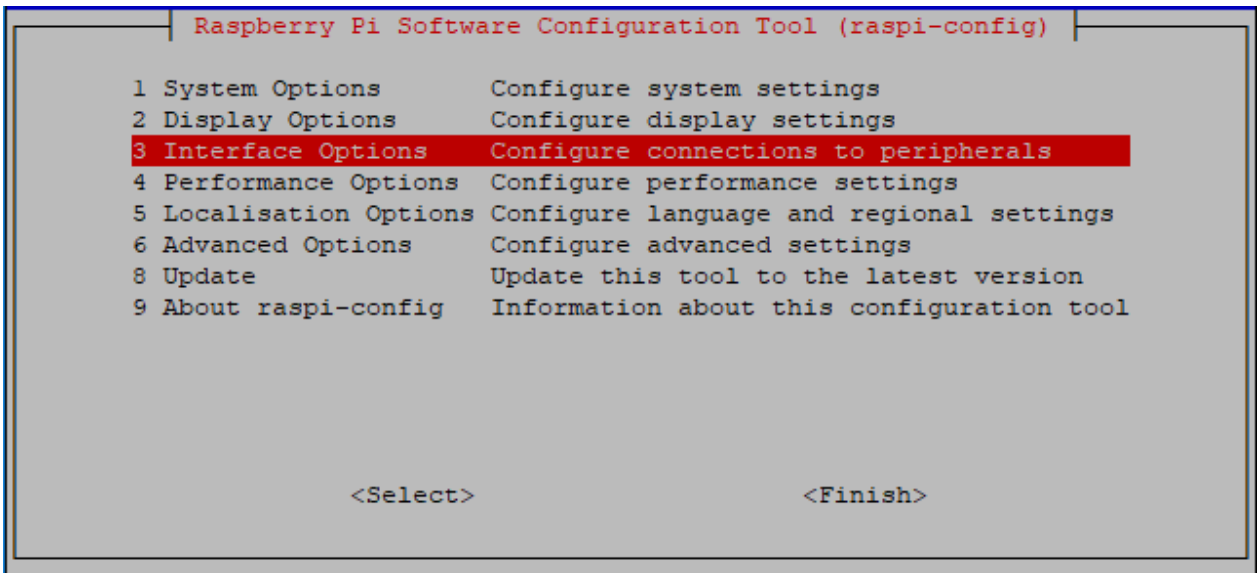
1.16 付録

1.16.1 I2C 構成

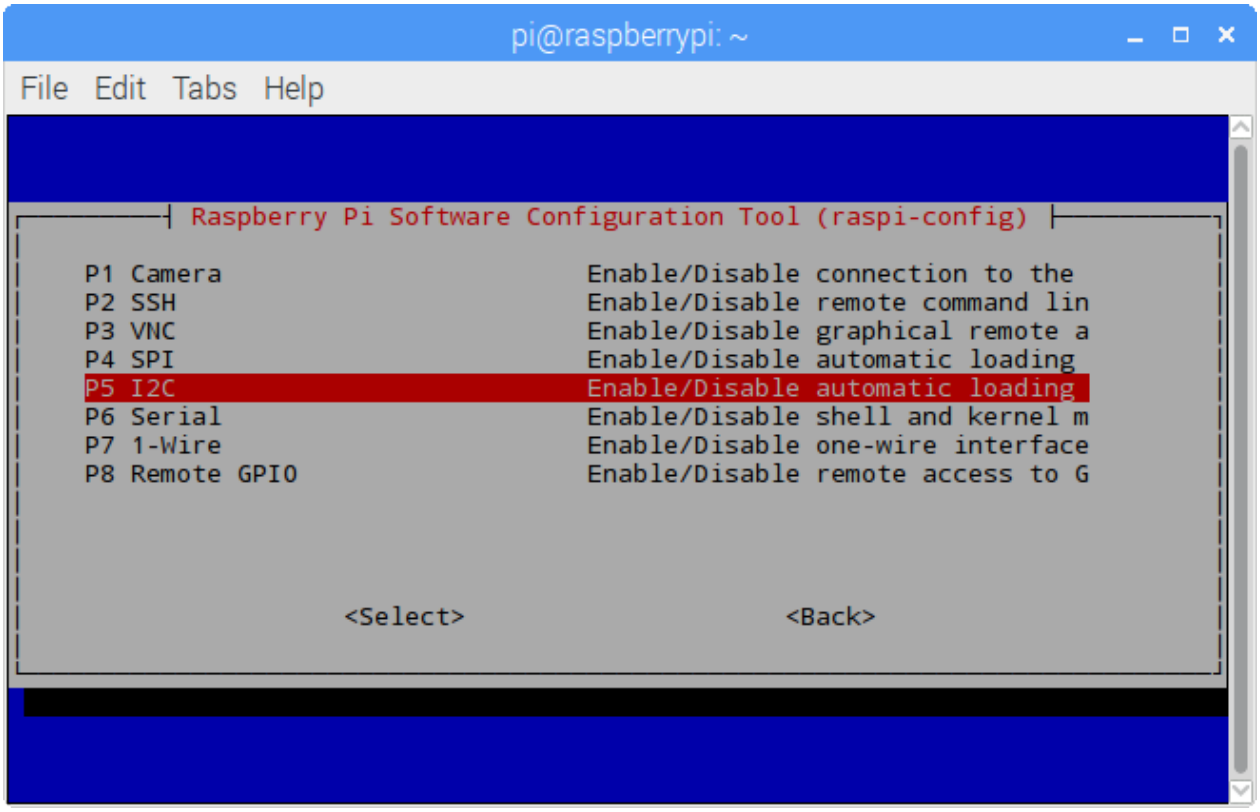
ステップ 1: Raspberry Pi の I2C ポートを有効にしてください（有効にしている場合はスキップしてください。

```
sudo raspi-config
```

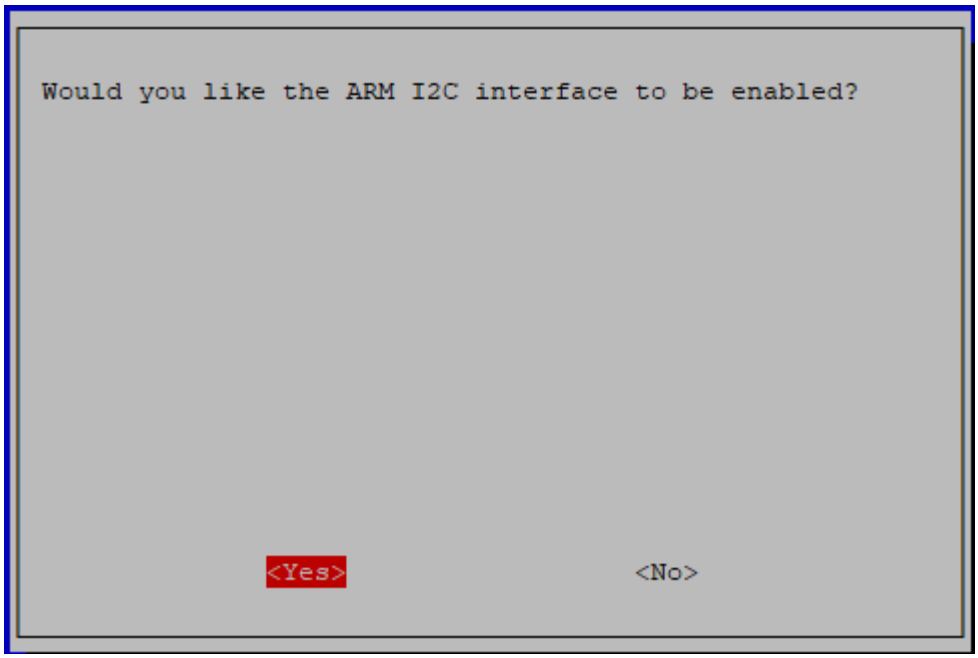
3 Interfacing options



P5 I2C



<Yes>、次に<Ok> -> <Finish>



ステップ 2: i2c モジュールがロードされ、アクティブになっているかどうかを確認します。

```
lsmod | grep i2c
```

すると、次のようなコードが表示されます（番号は異なる場合があります）。

```
i2c_dev                 6276    0
i2c_bcm2708             4121    0
```

ステップ 3： i2c-tools をインストールします。

```
sudo apt-get install i2c-tools
```

ステップ 4： I2C デバイスのアドレスを確認します。

```
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: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

(次のページに続く)

(前のページからの続き)

```
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

I2C デバイスが接続されている場合は、デバイスのアドレスが表示されます。

ステップ 5:

C 言語をお使いの方へ： libi2c-dev をインストールしてください。

```
sudo apt-get install libi2c-dev
```

Python ユーザーの場合：

For Python users:

1. 仮想環境の有効化。

注釈:

- 有効化する前に、仮想環境を作成していることを確認してください。詳細はこちらを参照してください: **仮想環境の作成**.
- Raspberry Pi を再起動するたびや、新しいターミナルを開くたびに、仮想環境を有効化するために次のコマンドを再度実行する必要があります。

```
source myenv/bin/activate
```

仮想環境が有効化されると、コマンドラインのプロンプトの前に環境名が表示され、仮想環境内で作業していることが示されます。

2. I2C のための smbus のインストール。

```
sudo pip3 install smbus2
```

3. 仮想環境からの退出。

作業を完了し、仮想環境から退出したい場合は、単純に次のコマンドを実行します:

```
deactivate
```

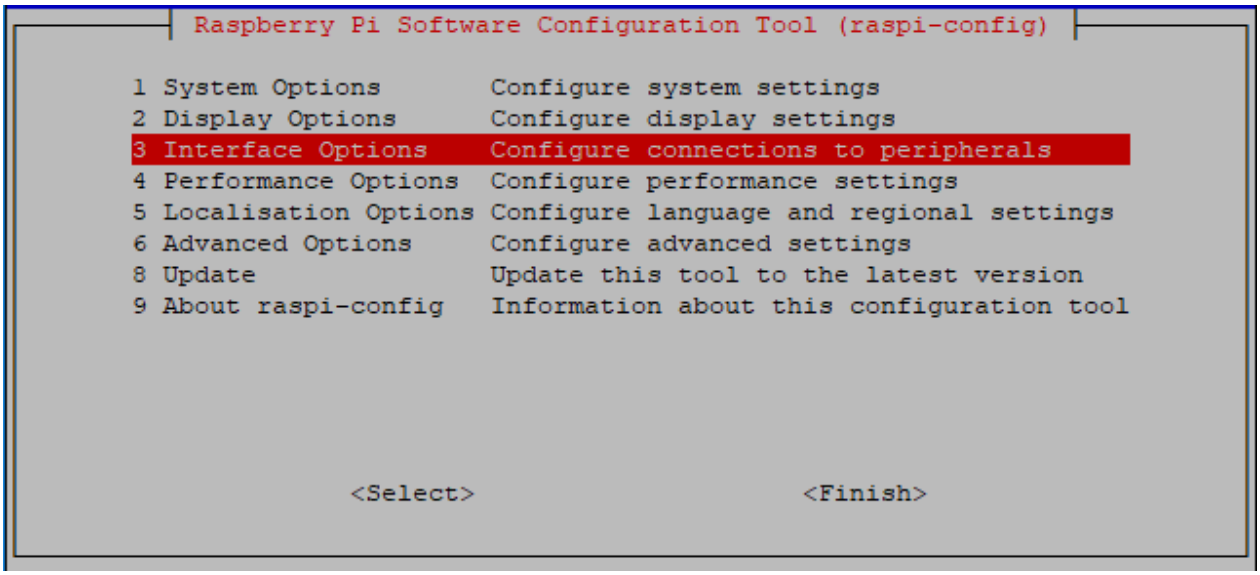
これにより、システムのグローバル Python 環境に戻ります。

1.16.2 SPI 構成

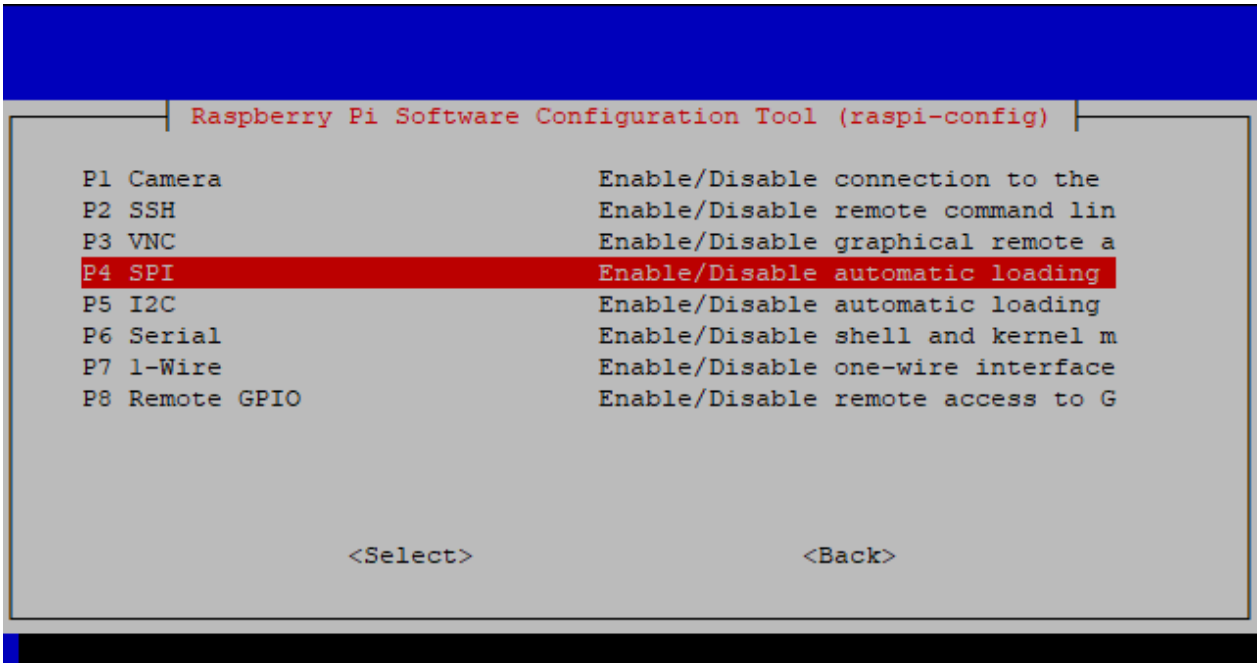
ステップ 1：Raspberry Pi の SPI ポートを有効にします（有効にしている場合はスキップします。有効にしたかどうかわからない場合は続行してください）。

```
sudo raspi-config
```

3 Interfacing options



P4 SPI



<YES> をクリックし、<OK> と <Finish> をクリックします。 これで、 sudo reboot コマンドを使用して

RaspberryPi を再起動できます。

ステップ 2： spi モジュールがロードされてアクティブになっていることを確認します。

```
ls /dev/sp*
```

次に、次のコードが表示されます（番号は異なる場合があります）。

```
/dev/spidev0.0  /dev/spidev0.1
```

ステップ 3： Python モジュール SPI-Py をインストールします。

```
git clone https://github.com/lthiery/SPI-Py.git
cd SPI-Py
sudo python3 setup.py install
```

注釈: この手順は Python ユーザー向けです。C 言語を使用している場合は、スキップしてください。

1.16.3 仮想環境の作成

Raspberry Pi や類似のデバイスを使用する場合、 `pip` を仮想環境でパッケージをインストールすることをお勧めします。これにより、依存関係の分離、システムのセキュリティの向上、システムのクリーンさの維持、プロジェクトの移行と共有の容易化が行われ、依存関係の管理が簡素化されます。これらの利点により、仮想環境は Python 開発において非常に重要で便利なツールとなっています。

以下に、仮想環境を作成する手順を示します。

1. 仮想環境の作成

まず、システムに Python がインストールされていることを確認する必要があります。Python バージョン 3.3 以降では、仮想環境を作成するための `venv` モジュールが付属しており、別途インストールする必要はありません。Python 2 または Python 3.3 以前のバージョンを使用している場合は、 `virtualenv` をインストールする必要があります。

- Python 3 の場合：

Python 3.3 以降のバージョンでは、 `venv` モジュールを直接使用できます：

```
python3 -m venv myenv
```

これにより、現在のディレクトリに名前が `myenv` の仮想環境が作成されます。

- Python 2 の場合：

まだ Python 2 を使用している場合は、まず `virtualenv` をインストールする必要があります：

```
pip install virtualenv
```

次に、仮想環境を作成します：

```
virtualenv myenv
```

これにより、現在のディレクトリに名前が `myenv` の仮想環境が作成されます。

2. 仮想環境の有効化

仮想環境を作成した後、使用するためにそれを有効化する必要があります。

注釈: Raspberry Pi を再起動するたびや、新しいターミナルを開くたびに、仮想環境を有効化するために次のコマンドを再度実行する必要があります。

```
source myenv/bin/activate
```

仮想環境が有効化されると、コマンドラインのプロンプトの前に環境名が表示され、仮想環境内で作業していることが示されます。

3. 依存関係のインストール

仮想環境を有効化した状態で、pip を使用して必要な依存関係をインストールできます。例：

```
pip install requests
```

これにより、requests ライブラリがグローバル環境ではなく、現在の仮想環境にインストールされます。このステップは 1 度だけ行えば十分です。

4. 仮想環境の終了

作業を完了し、仮想環境から退出したい場合は、単純に次のコマンドを実行します：

```
deactivate
```

これにより、システムのグローバル Python 環境に戻ります。

5. 仮想環境の削除

特定の仮想環境をもはや必要としない場合は、単純にその仮想環境を含むディレクトリを削除できます：

```
rm -rf myenv
```

1.17 FAQ

1.17.1 C コードが機能していませんか？

- 配線にがないか確認してください。
- コードがエラーを報告しているかどうかを確認します。
- 実行する前にコードを進めましたか。
- 上記の 3 つの条件がすべて問題ない場合は、wireingPi バージョン (2.50) が Raspberry Pi 4B 以降と互換性がない可能性があります。「WiringPi を確認してインストールする」を参照して、手動でバージョン 2.52 にアップグレードしてください

1.18 ありがとうございました

私たちの製品を評価してくれた評価者、チュートリアルの提案をしてくれたベテラン、そして私たちをフォローしてサポートしてくれたユーザーに感謝します。私たちへのあなたの貴重な提案は、より良い製品を提供するという私たちの動機です！

特に感謝します

- Len Davisson
- Kalen Daniel
- Juan Delacosta

さて、このアンケートに記入するために少し時間を割いていただけませんか？

注釈: アンケートを送信した後、トップに戻って結果を確認してください。

第 2 章

Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study,investigation, enjoyment, or other non-commercial or nonprofit purposes, under therelated regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.