SunFounder davinci-kit-for-raspberry-pi

www.sunfounder.com

03.04.2024

Inhaltsverzeichnis

1 Über das Da Vinci Kit **3**
1.1 Komponentenliste . . . 3
1.2 Einführung . . . 7
1.3 Vorbereitung . . . 8
1.4 Bibliotheken . . . 22
1.5 GPIO-Erweiterungskarte . . . 24
1.6 Laden Sie die Kode herunter . . . 27
1.7 1 Ausgabe (nicht für Pi 5) . . . 27
1.8 2 Eingabe (nicht für Pi 5) . . . 158
1.9 3 Erweiterung (nicht für Pi 5) . . . 286
1.10 Python Videokurs (nicht für Pi 5) . . . 413
1.11 Spielen Sie mit Python (für alle Pi-Modelle) . . . 433
1.12 Spielen Sie mit der Processing (nicht für Pi 5) . . . 667
1.13 Spielen mit Scratch (nicht für Pi 5) . . . 695
1.14 Andere Sprache (für Pi 5) . . . 770
1.15 Anhang . . . 773
1.16 FAQ . . . 788

2 Urheberrechtshinweis **789**

Danke, dass Sie sich für unser Vinci Kit entschieden haben.

Bemerkung: Dieses Dokument ist in den folgenden Sprachen verfügbar.

-
-
-

Bitte klicken Sie auf die jeweiligen Links, um das Dokument in Ihrer bevorzugten Sprache aufzurufen.

KAPITEL 1

Über das Da Vinci Kit

Dieses Da Vinci-Kit gilt für den Raspberry Pi 5/4 Modell B, 3 Modell A +, 3 Modell B +, 3 Modell B, 2 Modell B, 1 Modell B +, 1 Modell A +, zero W and zero. Es enthält verschiedene Komponenten und Chips, die dazu beitragen können, verschiedene interessante Phänomene zu erzeugen, die Sie durch eiige Operationen unter Anleitung von Versuchsanweisungen erhalten können. In diesem Prozess können Sie einige Grundkenntnisse über Programmieren erlernen. Sie können auch mehrere Anwendungen selbst erkunden. Jetzt mach es!

Wenn Sie andere Projekte kennenlernen möchten, die wir nicht haben, senden Sie bitte eine E-Mail und wir werden so schnell wie möglich auf unsere Online-Tutorials aktualisieren. Vorschläge sind willkommen.

Hier ist die E-Mail: cs@sunfounder.com.

1.1 Komponentenliste

Überprüfen Sie bitte nach dem Öffnen der Verpackung, ob Komponentsanzahl der Produktbeschreibung entspricht und ob alle Komponenten in gutem Zustand sind.

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

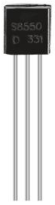
Yellow LED
4 pcs

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

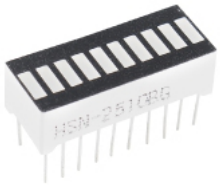
L293D
1 pcs

ADC0834
1 pcs

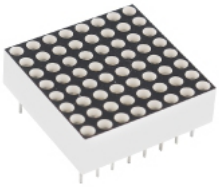
74HC595
2 pcs

7-segment Display
1 pcs

LED Bar Graph
1 pcs

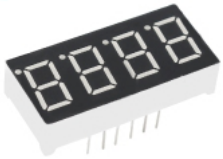


LED Matrix
1 pcs



Active Buzzer
2 pcs

4-Digit 7-segment Display
1 pcs



Tilt Switch
1 pcs

Potentiometer
3 pcs

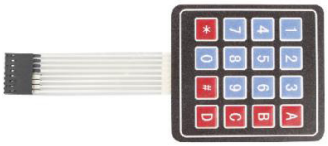
Passive Buzzer
1 pcs

Button
4 pcs

Motor
1 pcs

Slide Switch
2 pcs

Keypad
1 pcs



I2C LCD 1602
1 pcs

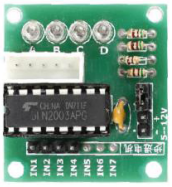
9G Servo
1 pcs

Breadboard Power Module
1 pcs

MFRC522 RFID Module
1 pcs

Relay
1 pcs

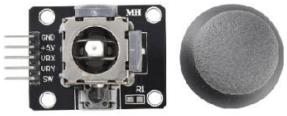
Stepping Motor Driver
1 pcs



Stepping Motor
1 pcs

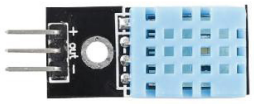
Ultrasonic Ranging Module
1 pcs

Joystick
1 pcs

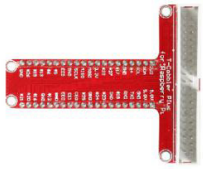


Infrare Motion Sensor
1 pcs

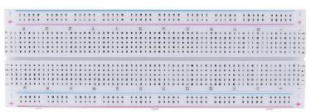
DHT-11
1 pcs



T-shape Extension Board
1 pcs



Breadboard
1 pcs



MPU6050 Module
1 pcs

1.2 Einführung

Da Vinci Kit ist ein Basis-Kit für intelligente Anfänger mit Projektszeitplan. Es enthält 26 häufig verwendete Eingangs- und Ausgangskomponenten und -module sowie eine Reihe grundlegender elektronischer Geräte (wie Widerstände, Kondensatoren), die Sie beim Programmierenlernen unterstützen können.

Nach dem Kit können Sie einige Grundkenntnisse über Raspberry Pi erlernen, einschließlich Installationsmethode von Raspberry Pi, Kenntnisse über Bash Shell und GPIO. Nachdem Sie dieses Wissen verstanden haben, können Sie mit der Programmierung beginnen.

Wenn Sie keinen Hardware Hintergrund Kenntnisse haben, bietet Ihnen dieses Kit Dokument 30 Lektionen zum Nachführen und Lernen, einschließlich 26 grundlegende I/o-Lektionen und 4 einfache praktische Beispiele. Beachten Sie bitte darauf, die Anordnung dieser Kurse basiert nicht auf Schwierigkeitsgrad, sondern Funktionen in der Praxis. Entsprechende Kurse finden Sie nach Ihren Bedürfnissen. Mit anderen Worten, selbst wenn Sie noch keinen vollständigen Kurs absolviert oder die Verwendung dieser Komponenten gemeistert haben,wird dieses Dokument eine wichtige Rolle spielen, um Sie bei der Durchführung praktischer Projekte in der Zukunft zu unterstützen.

Wir freuen uns auf Ihre Projekte und hoffen, dass Sie Ihre Erfolge oder Kreationen in unserem Forum teilen können, während Sie dieses Dokument lesen.

1.3 Vorbereitung

Vorbereitung In diesem Kapitel lernen wir zunächst den Raspberry Pi Der Inhalt umfasst die Installation des Betriebssystems und Raspberry Pi-Netzwerks und das Öffnen des Terminals.

Bemerkung: Sie können das vollständige Tutorial auf der offiziellen Website des Raspberry Pi nachlesen: raspberry-pi-setting-up.

Wenn Ihr Raspberry Pi eingerichtet ist, können Sie den Teil überspringen und mit dem nächsten Kapitel fortfahren.

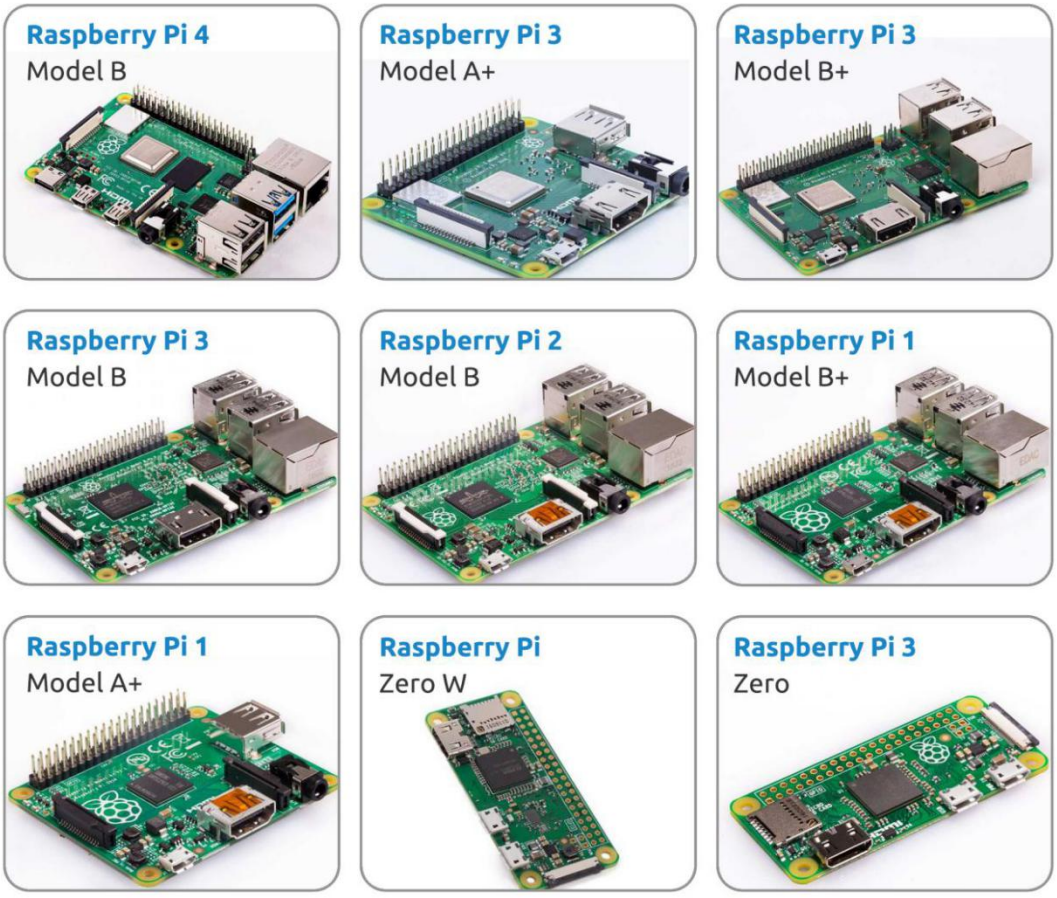
1.3.1 Was brauchen wir?

Erforderliche Komponenten

Raspberry -Pi

Der Raspberry Pi ist ein kostengünstiger Computer in Kreditkartengröße, der an einen Computermonitor oder Fernseher angeschlossen wird und eine Standard Tastatur und Maus verwendet. Es ist ein leistungsfähiges kleines Gerät, das Menschen mit allen Altern ermöglicht, Rechnung zu erforschen wie man in Sprachen wie Scratch und Python programmiert.

Unser Kit gilt für die folgenden Versionen des Produkts von Raspberry Pi:



Netzteil

Für den Anschluss an eine Steckdose verfügt der Raspberry Pi über einen Micro-USB-Anschluss (gleich wie bei vielen Mobiltelefonen). Sie benötigen eine Stromversorgung mit mindestens 2.5 Ampere.

Micro SD Karte

Ihr Raspberry Pi benötigt eine Micro-SD-Karte, um alle Dateien und das Raspberry Pi-Betriebssystem zu speichern. Sie benötigen eine Micro-SD-Karte mit einer Kapazität von mindestens 8 GB

Optionale Komponenten

Bildschirm

Um die Bidlshirm Umgebung von Raspberry Pi anzuzeigen, brauchen Sie einen Bildschirm, es kann ein Fernsehbildschirm oder ein Computermonitor sein. Wenn der Bildschirm hat eingebaute Lautsprecher, spielt der Pi über den Ton damit ab.

Maus & Tastatur

Wenn Sie einen Bildschirm verwenden, werden auch eine USB-Tastatur und eine USB-Maus benötigt.

HDMI

Der Raspberry Pi verfügt über einen HDMI-Ausgangsanschluss, der mit HDMI-Anschlüssen von meisten modernen Fernseh- und Computermonitore kompatibel ist. Wenn Ihr Bildschirm nur über DVI- oder VGA-Anschlüsse verfügt, müssen Sie die entsprechende Konvertierungsleitung verwenden.

Tasche

Sie können den Raspberry Pi in eine Tasche stecken; Auf diese Weise können Sie Ihr Gerät schützen.

Ton oder Kopfhörer

Der Raspberry Pi ist mit einem Audioanschluss von ca. 3,5 mm ausgestattet, Sie können den verwenden, wenn Ihr Bildschirm keine eingebauten Lautsprecher hat oder wenn es keinen Bildschirmsoperation gibt.

1.3.2 Installieren des Betriebssystems

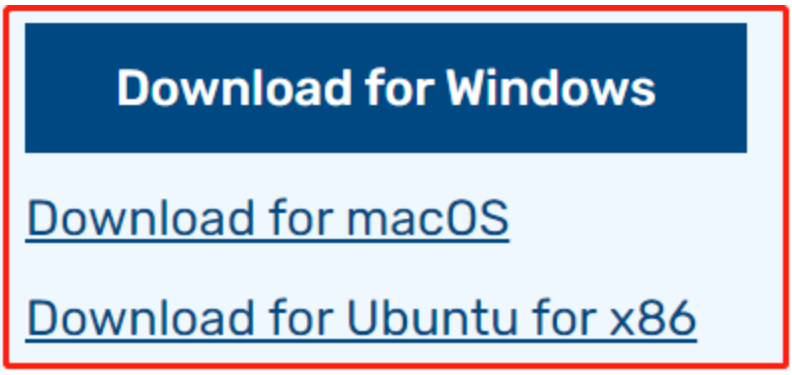
Erforderliche Komponenten

Raspberry Pi	1 * Personal Computer
1 * Micro SD Karte	

Schritt 1

Raspberry Pi haben ein grafisches SD-Karten-Schreibtool entwickelt, das auf Mac OS, Ubuntu 18.04 und Windows funktioniert und ist die einfachste Option für die meisten Benutzer, da es das Bild herunterladen und automatisch auf der SD-Karte installieren wird.

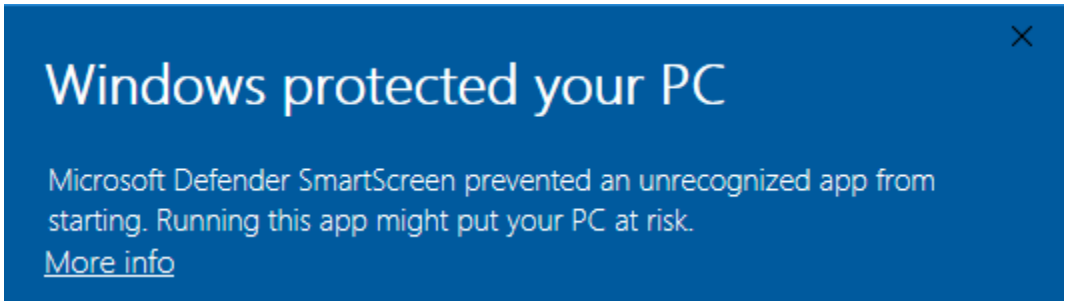
Besuchen Sie die Download-Seite: https://www.raspberrypi.org/software/. Klicken Sie auf den Link für den Raspberry Pi Imager, der Ihrem Betriebssystem entspricht. Wenn der Download abgeschlossen ist, klicken Sie darauf und das Installationsprogramm startet.



Schritt 2

Wenn Sie das Installationsprogramm starten, wird Ihr Betriebssystem möglicherweise versuchen, die Ausführung zu verhindern. Unter Windows erhalte ich beispielsweise die folgende Meldung:

Wenn dies angezeigt wird, klicken Sie auf **More info** und **Run anyway** , befolgen Sie dann die Anweisungen, um den Raspberry Pi Imager zu installieren.

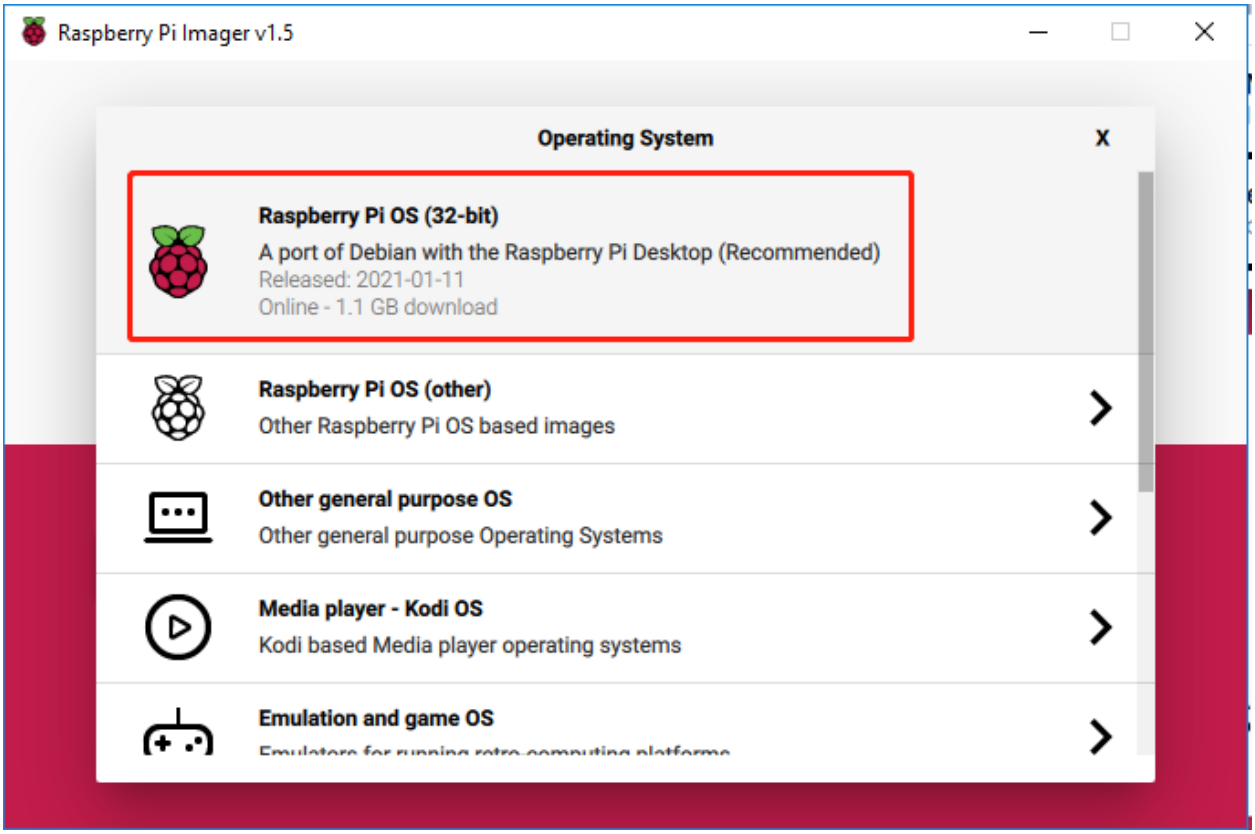


Schritt 3

Legen Sie Ihre SD-Karte in den SD-Kartensteckplatz des Computers oder Laptops ein.

Schritt 4

Wählen Sie im Raspberry Pi Imager das Betriebssystem aus, das Sie installieren möchten, und die SD-Karte, auf der Sie es installieren möchten.

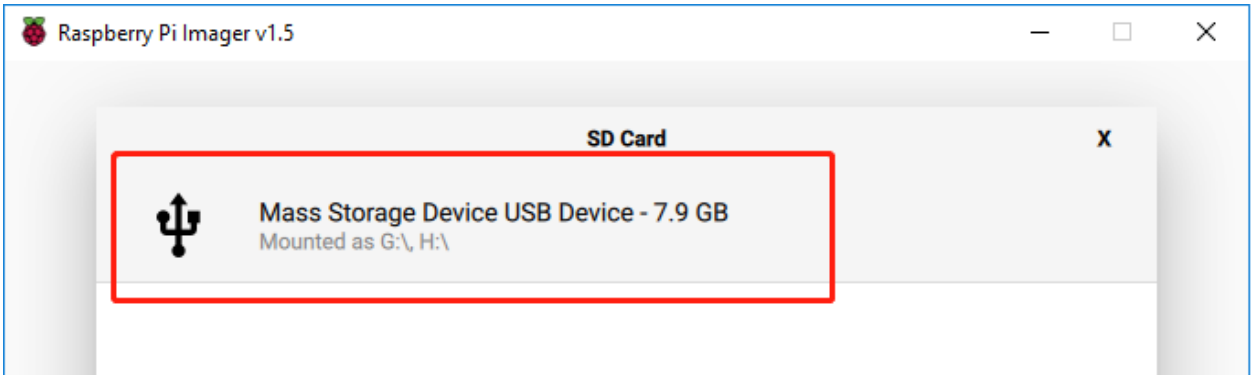


Bemerkung:

1. Sie müssen zum ersten Mal mit dem Internet verbunden sein.
2. Dieses Betriebssystem wird dann für die zukünftige Offline-Nutzung gespeichert (lastdownload.cache, C:/Users/yourname/AppData/Local/Raspberry Pi/Imager/cache).Wenn Sie die Software das nächste Mal öffnen, wird die Anzeige „Freigegeben: Datum, auf Ihrem Computer zwischengespeichert“ angezeigt.

Schritt 5

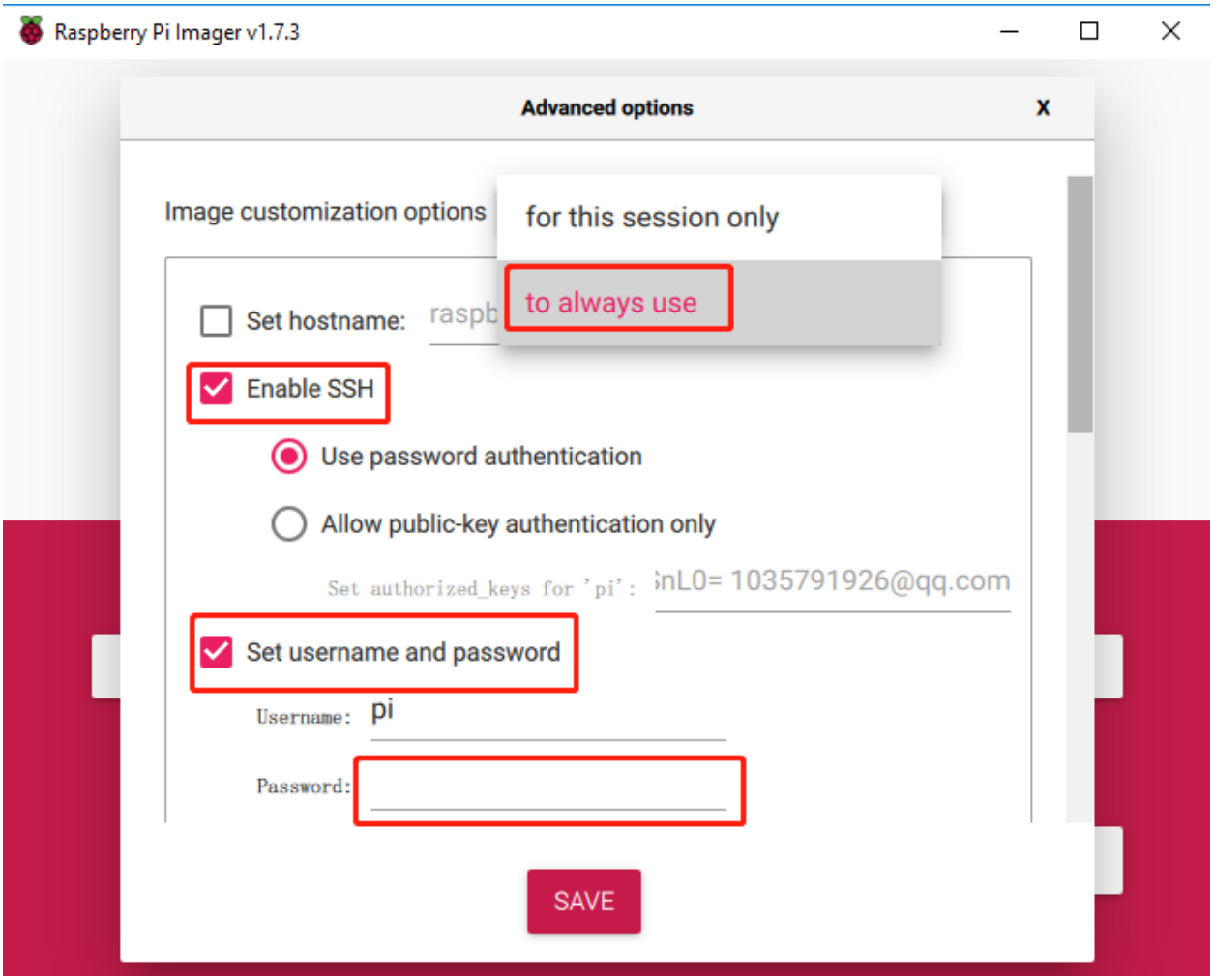
Wählen Sie die momentan verwendete SD-Karte aus.



Schritt 6

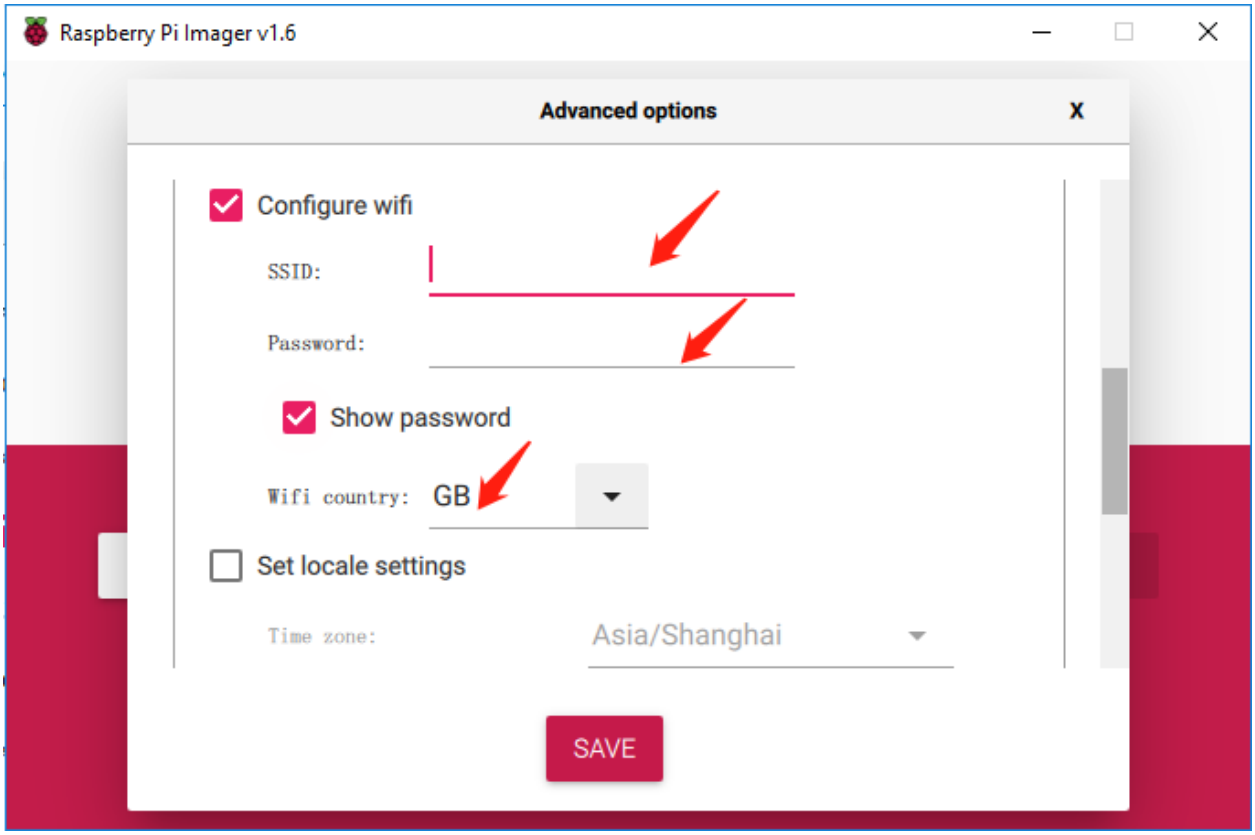
Drücken Sie **Ctrl+Shift+X**, um die Seite **Erweiterte Optionen** zu öffnen, um SSH zu aktivieren und WLAN zu konfigurieren. Diese beiden Elemente müssen festgelegt werden und die anderen hängen von Ihrer Wahl ab. Sie können

diese Bildanpassungsoptionen immer verwenden.



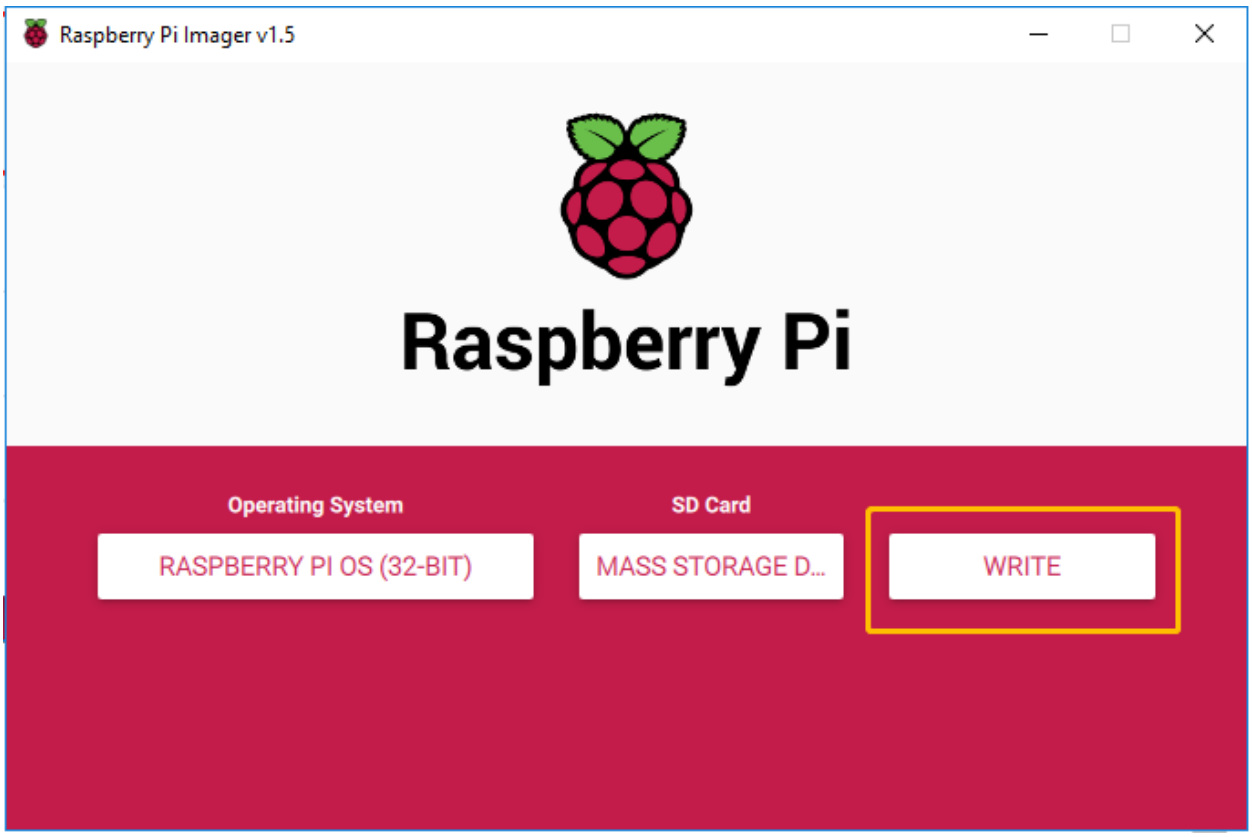
Scrollen Sie dann nach unten, um die WLAN-Konfiguration abzuschließen, und klicken Sie auf **SPEICHERN**.

Bemerkung: **Wlan-Land** sollte der zweistellige ISO/IEC alpha2 kode eingestellt werden.



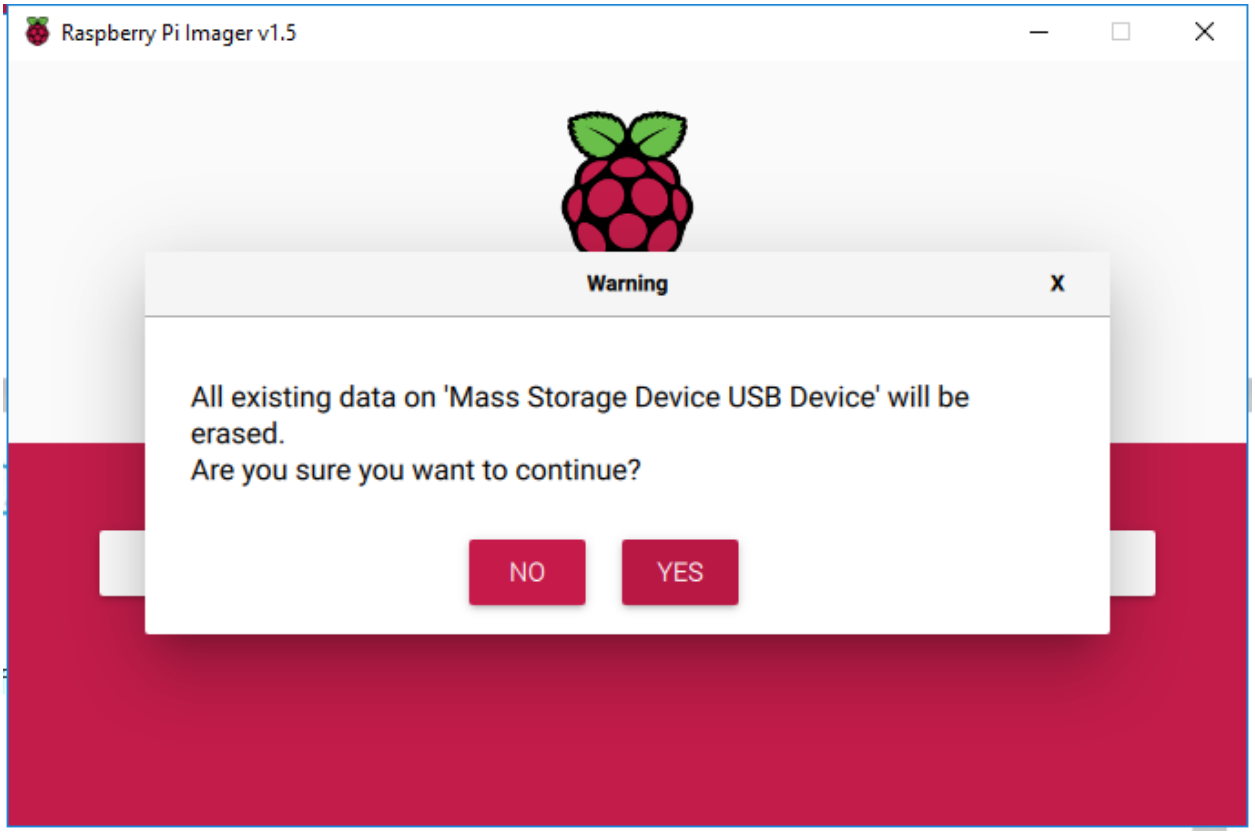
Schritt 7

Klicken Sie auf die Schaltfläche **SCHREIBEN**.



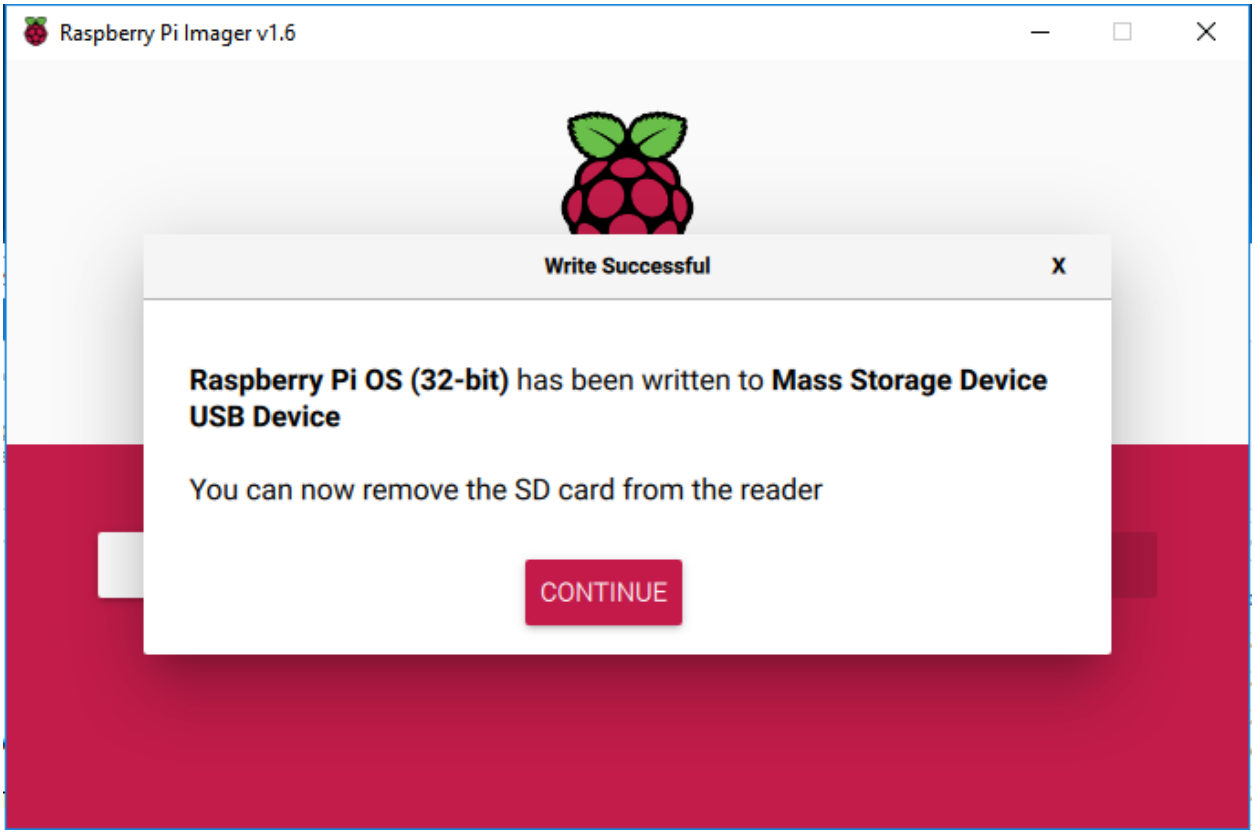
Schritt 8

Wenn auf Ihrer SD-Karte derzeit Dateien gespeichert sind, bitte speichern Sie diese möglicherweise zuerst, um zu verhindern, dass Sie sie dauerhaft verlieren. Wenn keine zu sichernde Datei vorhanden ist, klicken Sie auf **Ja**.



Schritt 9

Nach einer gewissen Wartezeit wird das folgende Fenster angezeigt und stellt den Abschluss des Schreibens dar.



1.3.3 Richten Sie Ihren Raspberry Pi ein

Wenn Sie einen Bildschirm haben

Wenn Sie einen Bildschirm haben, können Sie den Raspberry Pi problemlos bedienen.

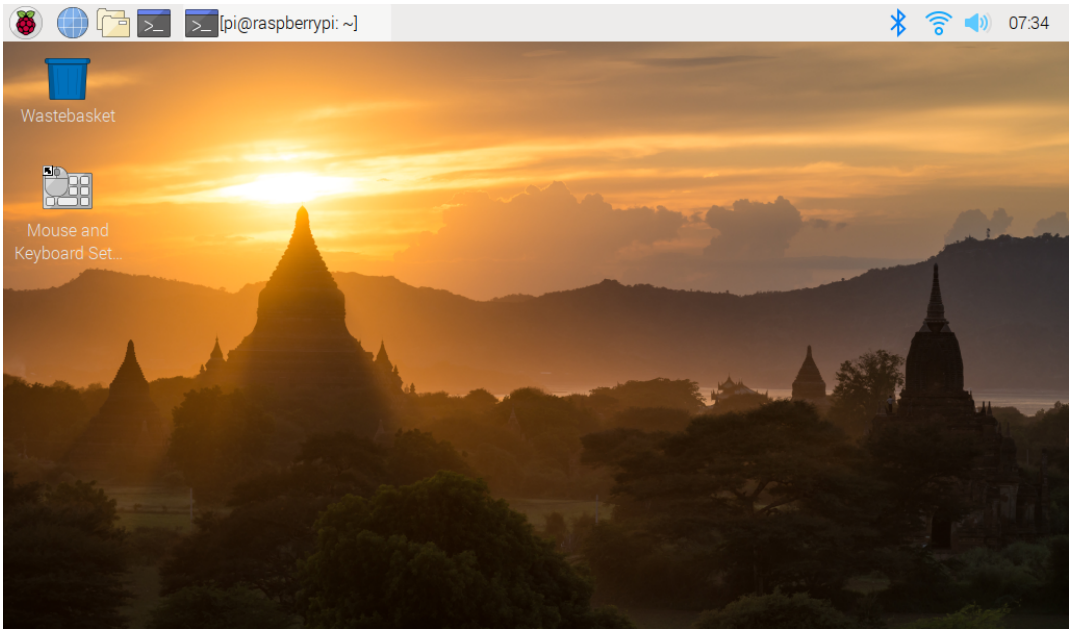
Erforderliche Komponenten

Raspberry Pi	1 * Netzteil
1 * Micro SD Karte	1 * Bildschirm Netzteil
1 * HDMI Kabel	1 * Bildschirm
1 * Maus	1 * Tastatur

1. Legen Sie die mit Raspberry Pi OS eingerichtete SD-Karte in den Micro-SD-Kartensteckplatz an der Unterseite Ihres Raspberry Pi ein.
2. Stecken Sie die Maus und Tastatur ein.
3. Schließen Sie den Bildschirm an den HDMI-Anschluss von Raspberry Pi an und stellen Sie sicher, dass Ihr Bildschirm an eine Wand Steckdose angeschlossen und eingeschaltet ist.

   Bemerkung: note: Wenn Sie einen Raspberry Pi 4 verwenden, müssen Sie den Bildschirm an den HDMI0 anschließen (der dem Stromanschluss am nächsten liegt).

4. Verwenden Sie das Netzteil und versorgen den Raspberry Pi mit Strom. Nach einigen Sekunden wird der Raspberry Pi OS-Bildschirm angezeigt.



Wenn Sie keinen Bildschirm haben

Wenn Sie keine Anzeige haben, können Sie sich aus der Ferne beim Raspberry Pi anmelden. Zuvor müssen Sie jedoch die IP-Addresse des Raspberry Pi abrufen.

Die IP-Adresse bekommen

Nachdem der Raspberry Pi mit WIFI verbunden ist, müssen wir die IP-Adresse davon erhalten. Es gibt viele Möglichkeiten, die IP-Adresse zu ermitteln, und zwei davon sind wie folgt aufgeführt.

1. Überprüfung über den Router

Wenn Sie berechtigt sind, sich beim Router anzumelden (z. B. in einem Heimnetzwerk), können Sie die Raspberry Pi zugewiesenen Adressen auf der Administrationsoberfläche des Routers überprüfen.

Der Default Hostname des Raspberry Pi-Betriebssystems ist raspberrypi, und bitte Sie ihn finden. (Wenn Sie das ArchLinuxARM-System verwenden, suchen Sie bitte alarmpi.)

2. Scannen von Netzwerksegmenten

Sie können auch Netzwerk Scannen verwenden, um die IP-Adresse von Raspberry Pi zu ermitteln. Sie können die Software anwenden, den **Advanced IP scanner** usw.

Scannen Sie den eingestellten IP-Bereich, und alle angeschlossenen Gerätsnamen wird angezeigt. Ebenso, der Default Hostname des Raspberry Pi-Betriebssystems ist *raspberrypi* , wenn Sie ihn nicht geändert haben.

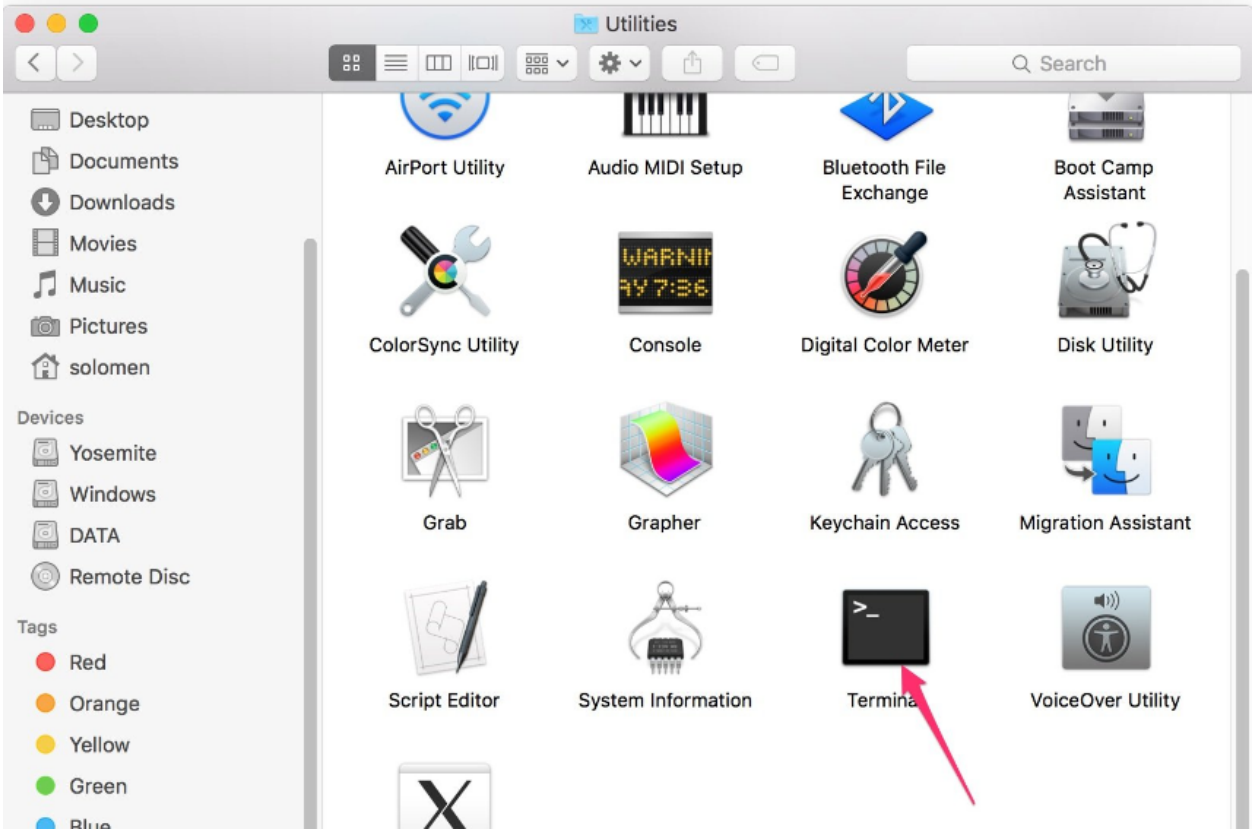
Verwenden Sie die SSH-Fernbedienung

Wir können die Bash Shell von Raspberry Pi öffnen, mit Anwendung von SSH. Bash ist die Default Standard Shell von Linux. Die Shell selbst ist ein in C geschriebenes Programm und stellt die Brücke zwischen den Kunden und Unix / Linux dar. Darüber hinaus kann es helfen, den größten Teil der erforderlichen Arbeit zu erledigen.

Für Linux- oder Mac OS X-Benutzer

Schritt 1

Gehen Sie zu **Applications->Utilities**, suchen Sie das **Terminal**, und öffnen Sie es.



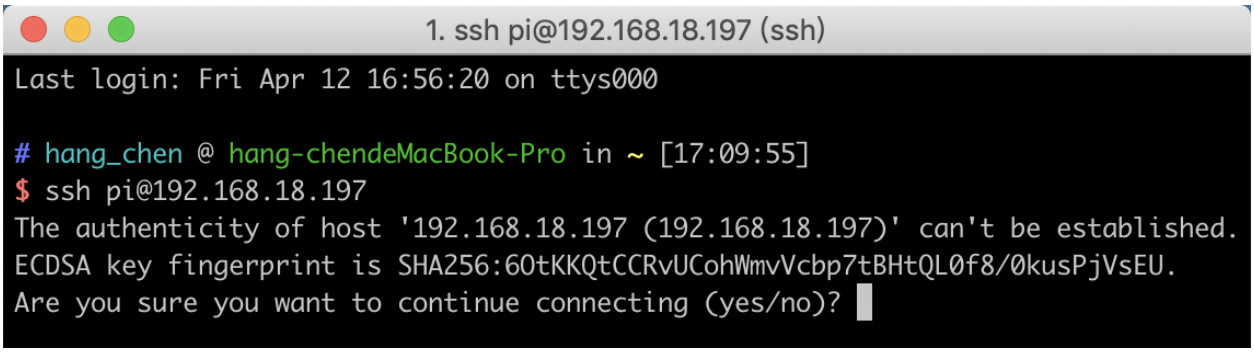
Schritt 2

Geben Sie `ssh pi@ip_address` ein. `Pi` ist Ihr Benutzername und `ip_address` ist Ihre IP-Adresse. Z. B. :

```
ssh pi@192.168.18.197
```

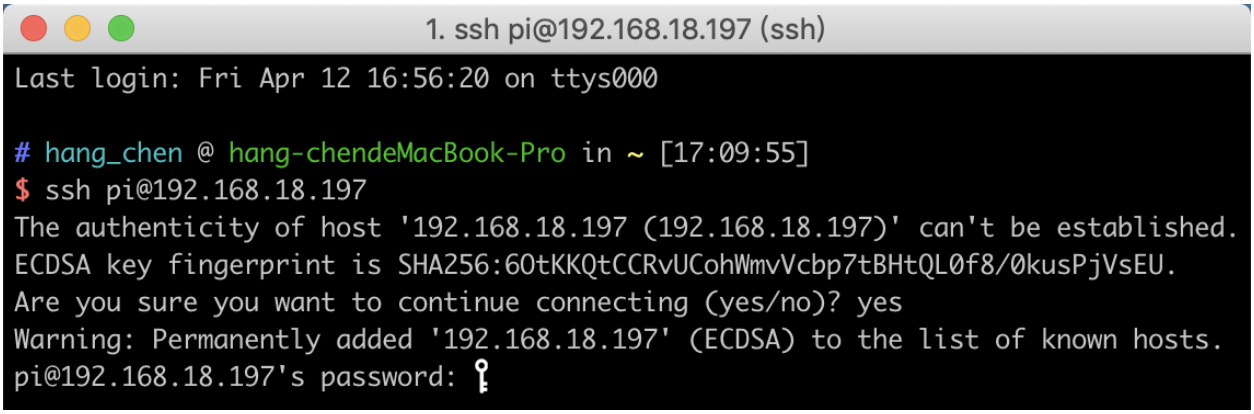
Schritt 3

Geben Sie `yes` ein.



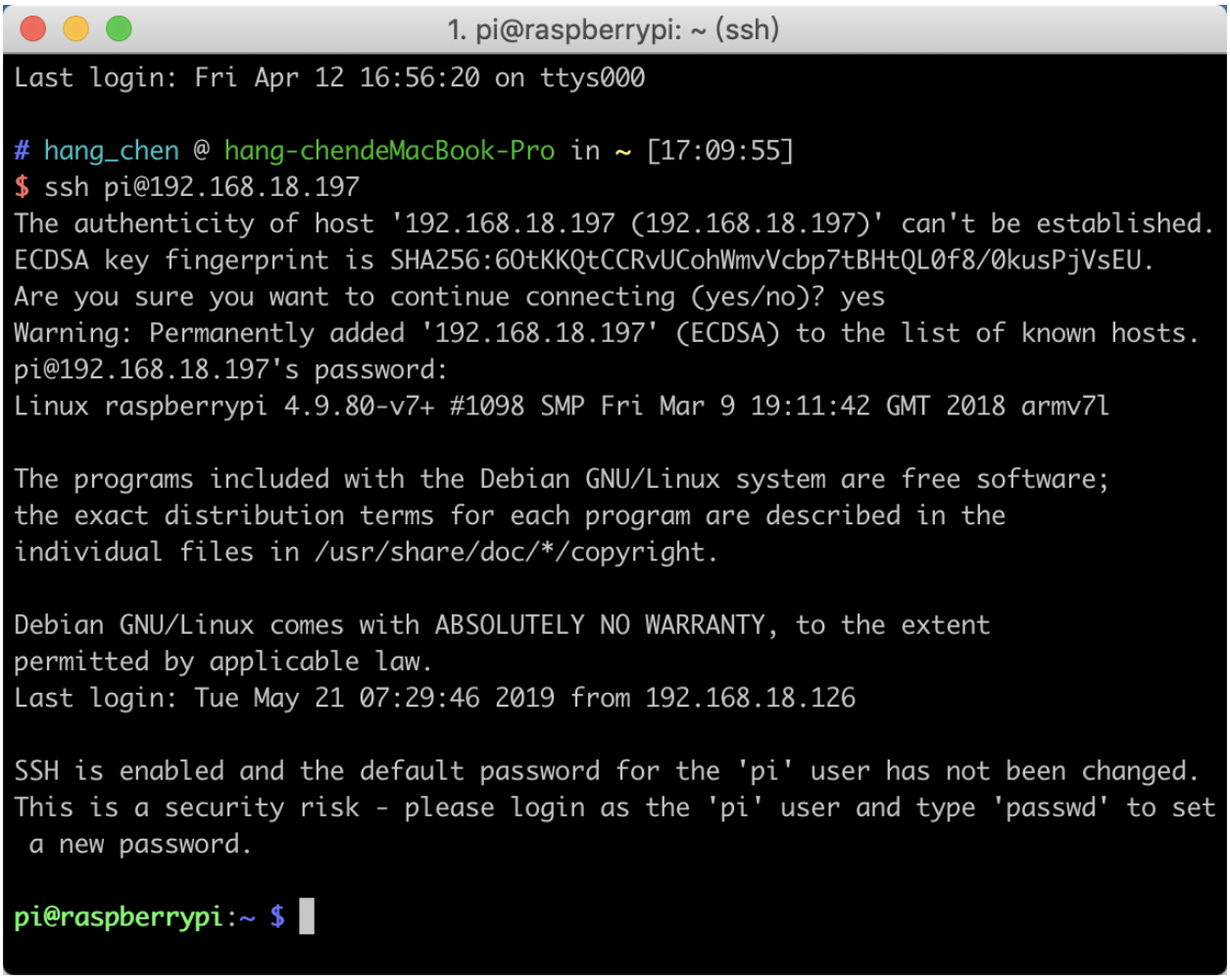
Schritt 4

Geben Sie den Password ein und das Default Passwort lautet **raspberry**.



Schritt 5

Wir verbinden jetzt den Raspberry Pi und sind bereit, mit dem nächsten Schritt fortzufahren.



Bemerkung: Wenn Sie das Kennwort eingeben, werden die Zeichen im Fenster nicht entsprechend angezeigt, was normal ist. Sie brauchen lediglich das richtige Passwort einzugeben.

Für Windows-Benutzer

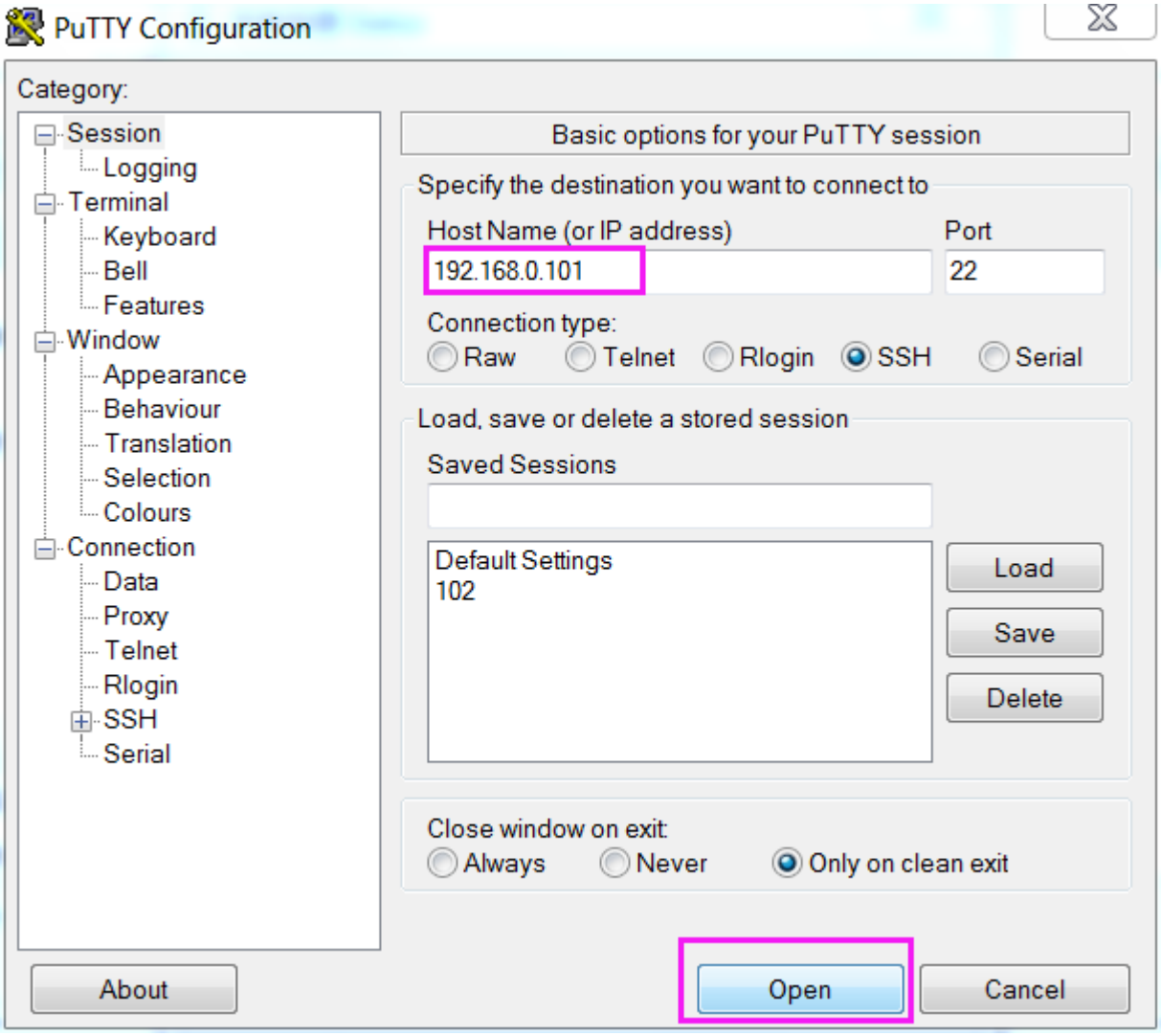
Wenn Sie ein Windows-Benutzer sind, können Sie SSH mit der Anwendung einiger Software verwenden. Hier empfehlen wir **PuTTY** .

Schritt** 1

Laden** Sie PuTTY herunter.

Schritt 2

Öffnen Sie PuTTY und klicken Sie auf **Session** in der linken baumähnlichen Struktur. Geben Sie die IP-Adresse des RPi in das Textfeld unter **Host Name (or IP address)** und **22** unter **Port** ein (Default ist 22).



Schritt 3

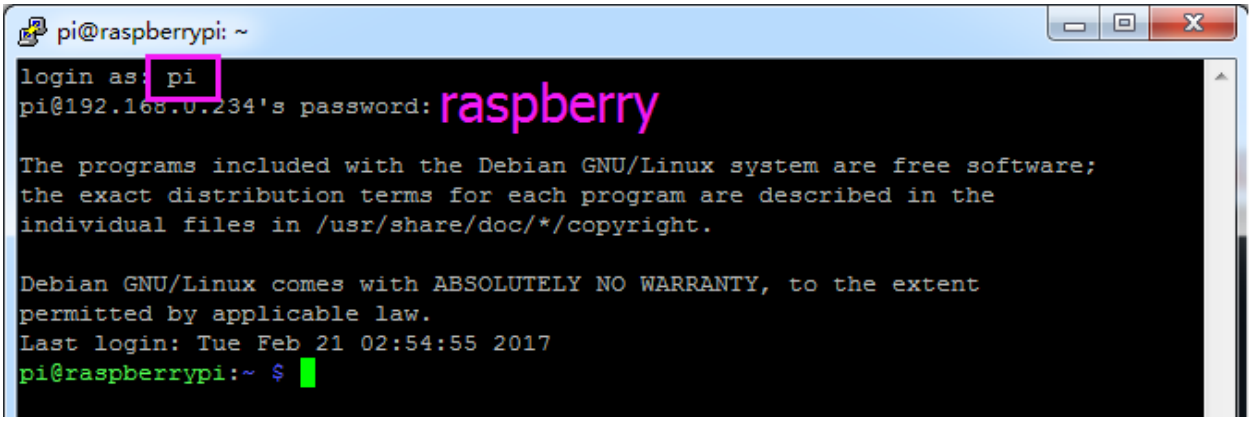
Klicken Sie auf **Open**. Beachten Sie bitte, beim ersten Anmelden am Raspberry Pi mit der IP-Adresse eine Sicherheitserinnerung wird anzeigen. Klicken Sie einfach auf **Yes**.

Schritt 4

Wenn das PuTTY-Fenster **login as:** auffordert, geben Sie `pi` (den Benutzernamen des RPi) und das **password** ein: `raspberry` (die Standardeinstellung, wenn Sie sie nicht geändert haben).

Bemerkung: Wenn Sie das Kennwort eingeben, werden die Zeichen im Fenster nicht entsprechend angezeigt, was normal ist. Sie brauchen lediglich das richtige Passwort einzugeben.

Wenn neben PuTTY inaktiv angezeigt wird, bedeutet dies, dass die Verbindung unterbrochen wurde und erneut verbunden werden muss.



Schritt 5

Hier wird der Raspberry Pi angeschlossen und es ist Zeit, die nächsten Schritte durchzuführen.

Bemerkung: Wenn Sie mit der Verwendung des Befehlsfensters zur Steuerung des Raspberry Pi nicht zufrieden sind, können Sie auch die Remote-Desktop-Funktion verwenden, die uns helfen kann, die Dateien im Raspberry Pi einfach zu verwalten.

Einzelheiten dazu finden Sie unter *Remotedesktop*.

1.4 Bibliotheken

Bei der Programmierung mit Raspberry Pi werden zwei wichtige Bibliotheken verwendet: WiringPi und RPi.GPIO. Das Raspberry Pi OS installiert sie standardmäßig, Sie können sie direkt verwenden.

1.4.1 RPi.GPIO

Wenn Sie ein Python-Benutzer sind, können Sie GPIOs mit der von RPi.GPIO bereitgestellten API programmieren.

RPi.GPIO ist ein Modul zur Steuerung von Raspberry Pi GPIO-Kanälen. Dieses Paket enthält eine Klasse zur Steuerung des GPIO einer Raspberry Pi. Beispiele und Dokumente finden Sie unter http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

Beim Testen ob RPi.GPIO installiert ist oder nicht, geben Sie bitte Python ein:

```
python
```

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

Geben Sie in Python CLI `import RPi.GPIO` ein. Wenn kein Fehler stellt, bedeutet, RPi.GPIO ist installiert.

```
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
>>>
```

Wenn Sie die Python-CLI beenden möchten, geben Sie ein:

```
exit()
```

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

1.4.2 Installieren und überprüfen Sie das WiringPi

`wiringPi` ist eine GPIO-Bibliothek in C, die auf den Raspberry Pi angewendet wird. Es entspricht GUN Lv3. Die Funktionen in wiringPi ähneln denen im Verdrahtungssystem von Arduino. Sie ermöglichen den mit Arduino vertrauten Benutzern, wiringPi einfacher zu verwenden.

`wiringPi` enthält viele GPIO-Befehle, mit denen Sie alle Arten von Schnittstellen auf dem Raspberry Pi steuern können.

Bitte führen Sie den folgenden Befehl aus, um die `wiringPi`-Bibliothek zu installieren.

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

Sie können anhand der folgenden Anleitung testen, ob die wiringPi-Bibliothek erfolgreich installiert wurde oder nicht.

```
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.
```

Überprüfen Sie den GPIO mit dem folgenden Befehl:

```
gpio readall
```

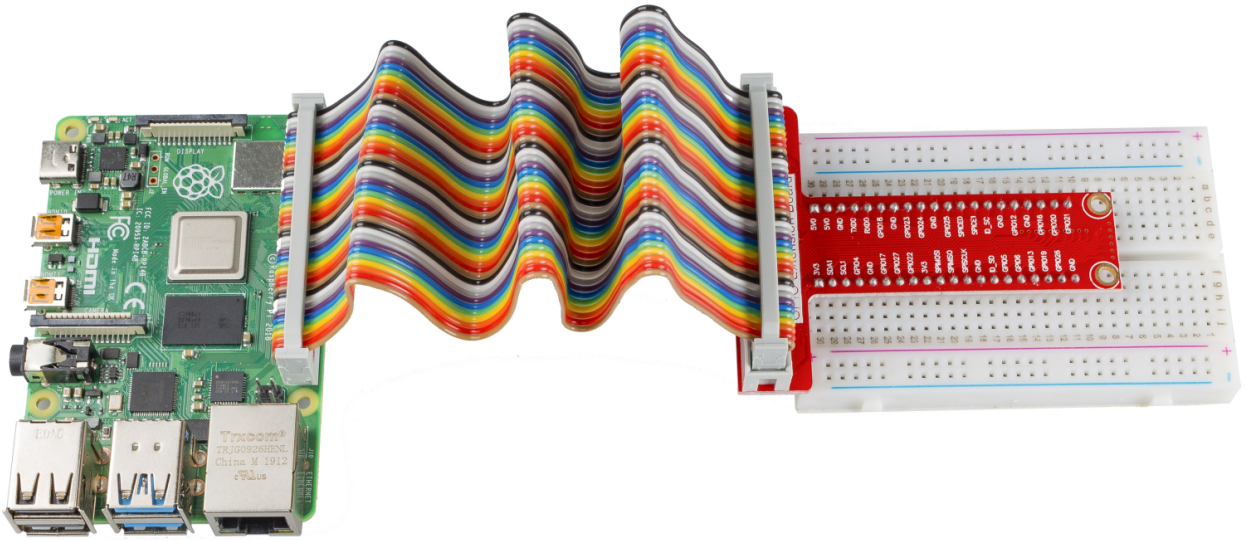
```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |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

Weitere Einzelheiten zu WiringPi finden Sie unter WiringPi.

1.5 GPIO-Erweiterungskarte

Vor dem Erlernen der Befehle, müssen Sie zunächst mehr über die Pins von Raspberry Pi wissen, der für die nachfolgende Studie wichtige Rolle spielt.

Wir können die Pins Raspberry Pi leicht über das GPIO Erweiterungskarte zum Steckbrett führen, um GPIO-Schäden aus häufiges Ein- oder Ausstecken zu vermeiden. Dies ist unser 40-pin GPIO-Erweiterungskarte und GPIO-Kabel für Raspberry Pi Modell B +, 2 Modell B und 3, 4 Modell B.



Pin Nummer

Die Pins von Raspberry Pi können auf drei Arten benannt werden: WiringPi, BCM und Board.

Unter diesen Benennungsmethoden verwendet die 40-pin GPIO-Erweiterungskarte die Benennungsmethode BCM. Für einige spezielle Pins wie den I2C-Port und den SPI-Port verwenden sie jedoch den Namen, der mit ihnen geliefert wird.

Die folgende Tabelle zeigt die Benennungsmethoden von WiringPi, Karte und den systeminternen Name jedes Pins auf der GPIO-Erweiterungskarte. Beispielsweise für GPIO17 die Karte ist Benennungsmethode 11, die wiringPi-Benennungsmethode 0 und die intrinsische Benennungsmethode GPIO0.

Bemerkung:

1. In der Sprache C wird die Benennungsmethode WiringPi verwendet.
2. In Python Language werden Karte und BCM als Benennungsmethoden verwendet, und die Funktion `GPIO.setmode()` wird verwendet, um sie einzustellen.

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

1.6 Laden Sie die Kode herunter

Beachten Sie vor dem Herunterladen des Codes, dass der Beispielcode NUR unter Raspberry Pi OS getestet wird. Wir bieten zwei Methoden zum Download an:

Methode 1: git clon verwenden (empfohlen)

Melden Sie sich bei Raspberry Pi an und wechseln Sie dann das Verzeichnis in `/home/pi`.

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

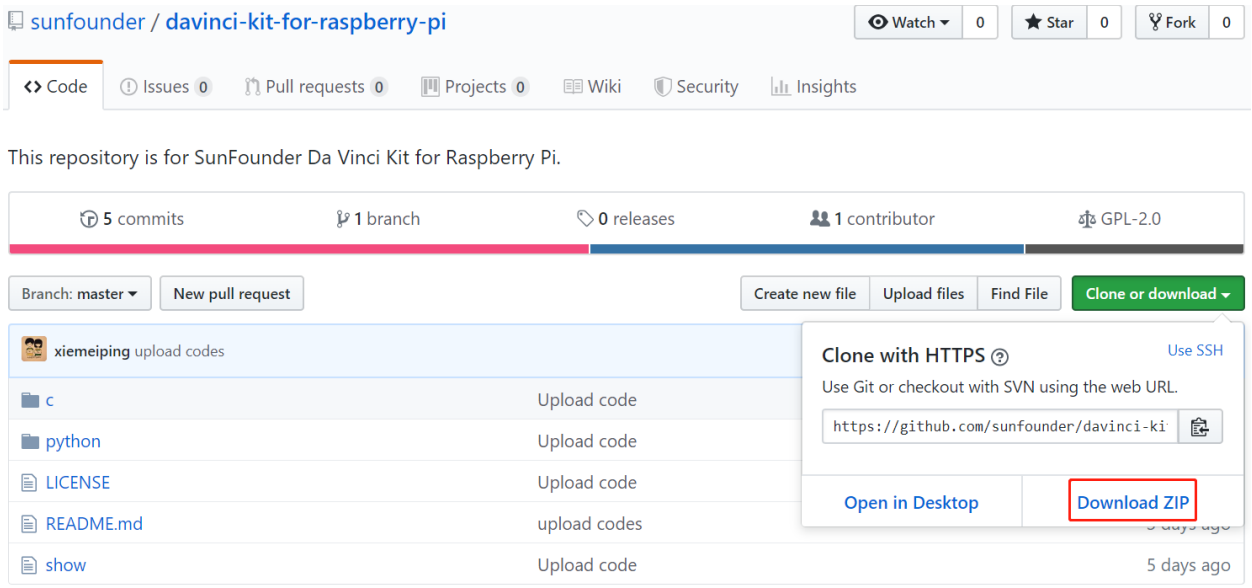
Bemerkung: cd, um vom aktuellen Pfad in das beabsichtigte Verzeichnis zu wechseln. Informell geht es hier zum Pfad `/home/pi/`.

Klonen Sie das Repository von GitHub.

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

Methode 2: Laden Sie den Kode herunter.

Laden Sie den Quellcode von github herunter: https://github.com/sunfounder/davinci-kit-for-raspberry-pi.



1.7 1 Ausgabe (nicht für Pi 5)

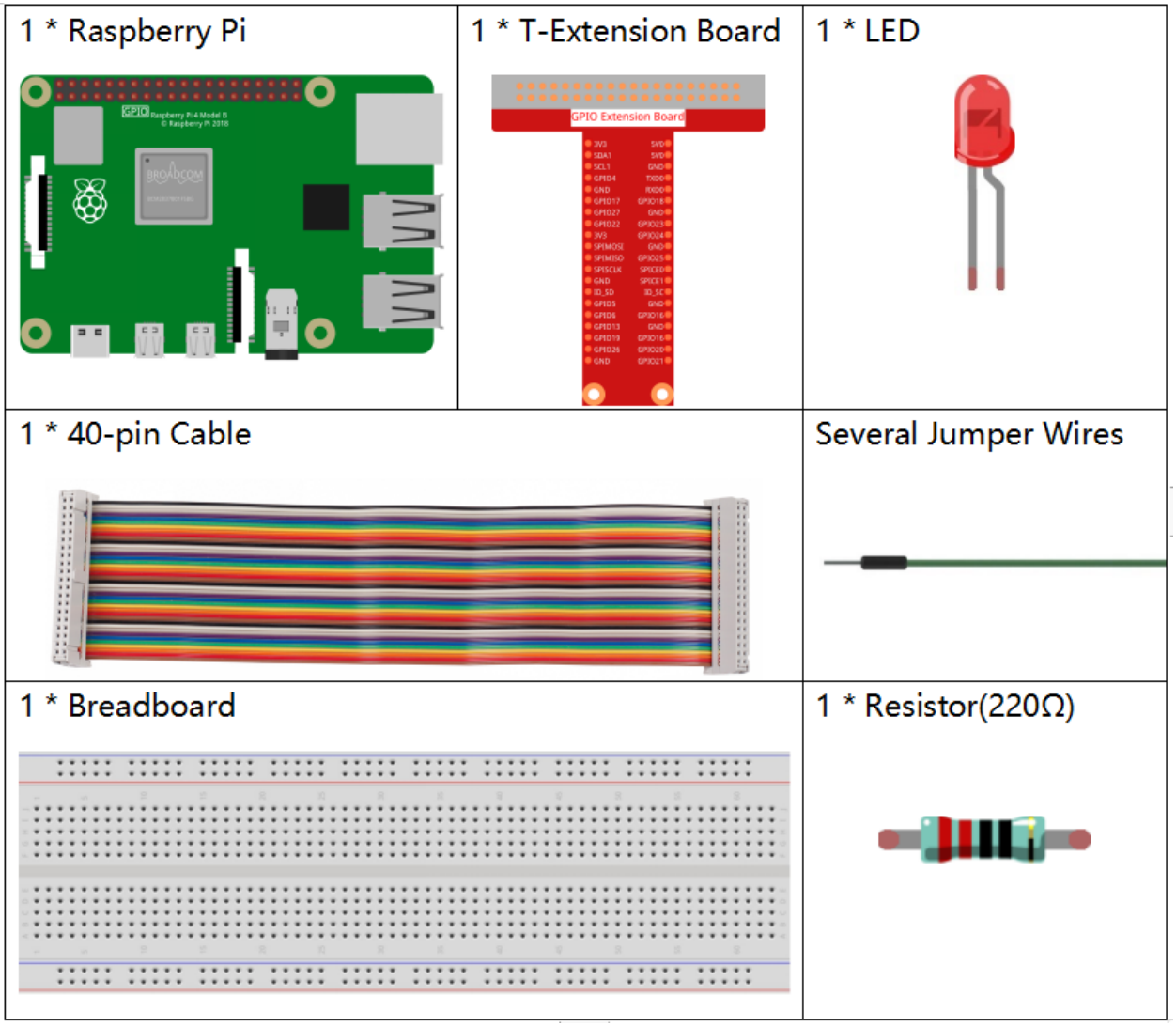
1.1 Anzeigen

1.7.1 1.1.1 Blinkende LED

Einführung

In dieser Lektion lernen wir, wie man durch Programmierung eine blinkende LED erzeugt. Durch Ihre Einstellungen kann Ihre LED eine Reihe interessanter Phänomene erzeugen. Jetzt mach es.

Komponenten



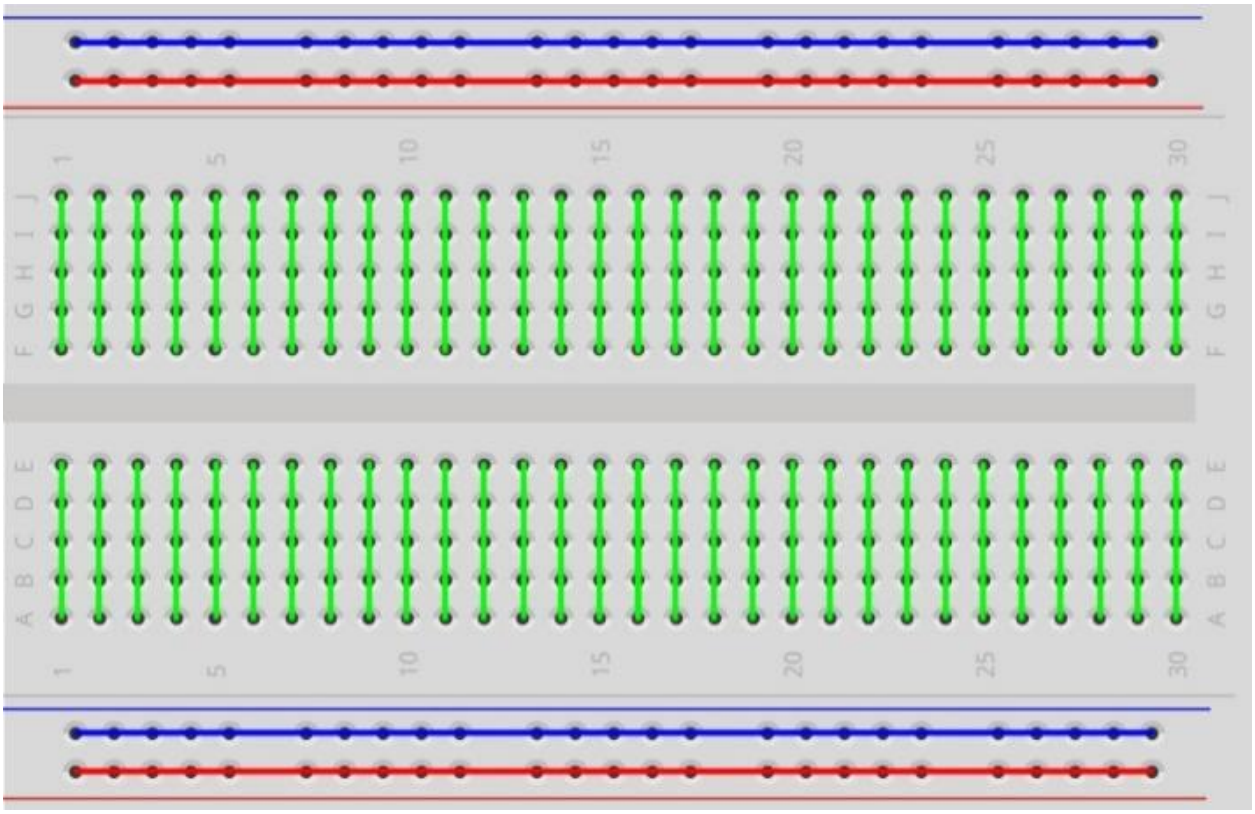
Bemerkung: Um reibungslos fortzufahren, müssen Sie Ihren eigenen Raspberry Pi, TF-Karte und Raspberry Pi-Strom mitbringen.

Prinzip

Steckbrett

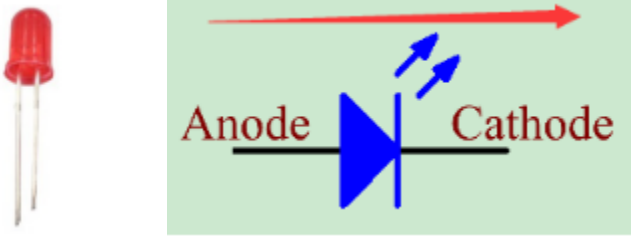
Ein Steckbrett ist eine Konstruktionsbasis für das Prototyping von Elektronik. Es wird verwendet, um Schaltungen schnell aufzubauen und zu testen, bevor ein Schaltungsentwurf abgeschlossen wird. Und es hat viele Löcher, in die oben erwähnte Komponenten wie ICs und Widerstände sowie Jumperdrähte eingesetzt werden können. Mit dem Steckbrett können Sie Komponenten einfach anschließen und entfernen.

Das Bild zeigt die interne Struktur eines Voll+ Steckbretts. Obwohl diese Löcher auf dem Steckbrett unabhängig voneinander zu sein scheinen, sind sie tatsächlich intern über Metallstreifen miteinander verbunden.



LED

LED ist eine Art Diode. Die LED leuchtet nur, wenn der lange Stift der LED mit der positiven Elektrode und der kurze Stift mit der negativen Elektrode verbunden ist.



Die LED kann nicht direkt an die Stromversorgung angeschlossen werden, die Komponente kann daruas beschädigt werden. Ein Widerstand mit 160 oder mehr (Arbeit in 5V) muss in der LED-Schaltung in Reihe geschaltet werden.

Widerstand

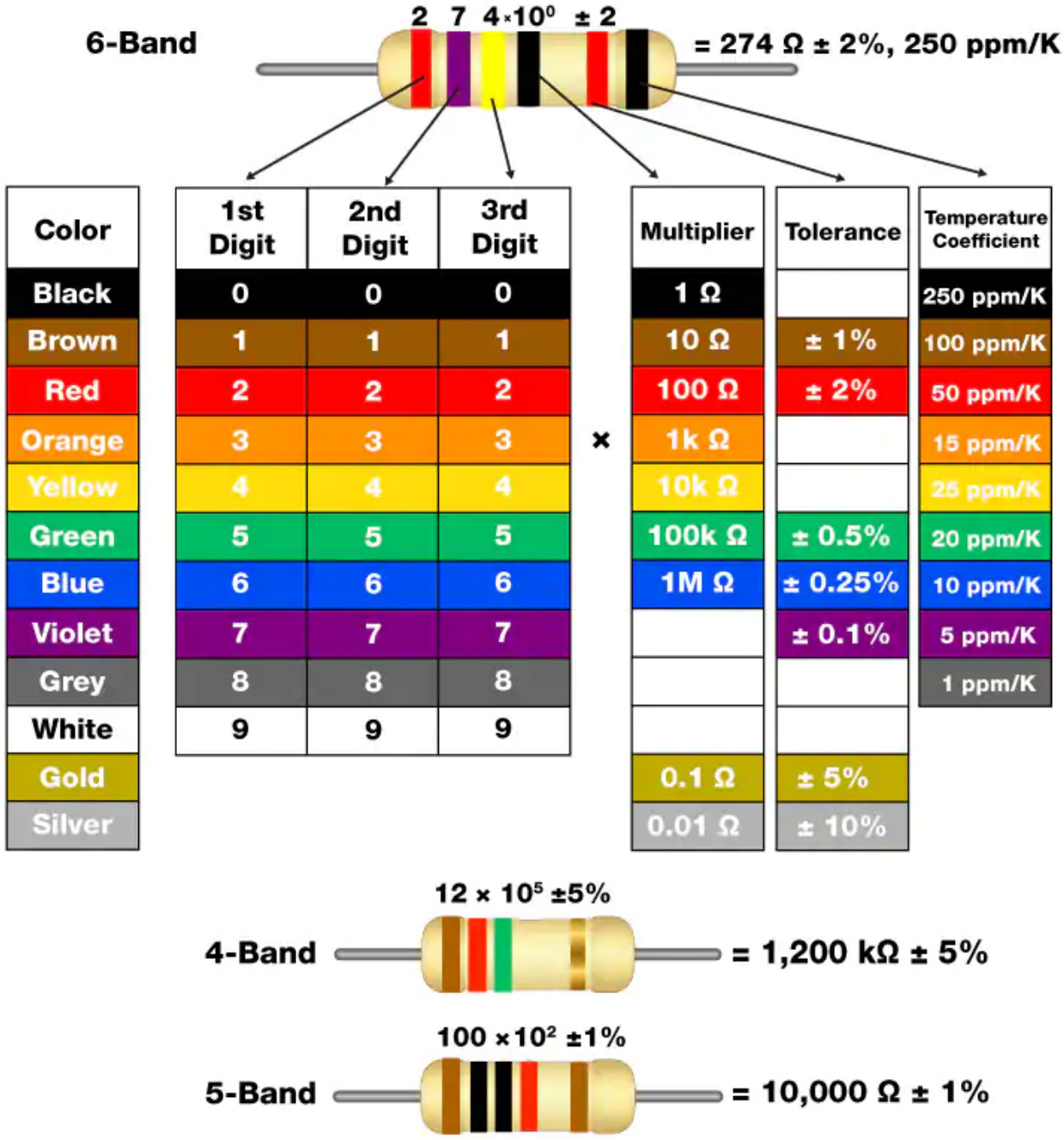
Der Widerstand ist ein elektronisches Element, das den Zweigstrom begrenzen kann. Ein fester Widerstand ist ein Tpy

von Widerstand, dessen Widerstand nicht geändert werden kann, während der eines Potentiometers oder eines variablen Widerstands eingestellt werden kann.

In diesem Kit wird ein fester Widerstand angewendet. In der Schaltung ist es wichtig, die angeschlossenen Komponenten zu schützen. Die folgenden Bilder zeigen ein reales Objekt, einen 220-Widerstand und zwei allgemein verwendete Schaltungssymbole des Widerstands. ist die Widerstandseinheit und die größeren Einheiten umfassen K, M usw. Ihre Beziehung kann wie folgt gezeigt werden: 1 M = 1000 K, 1 K = 1000 . Normalerweise ist der Widerstandswert darauf markiert. Wenn Sie diese Symbole in einer Schaltung sehen, bedeutet, es gibt ein Widerstand.

Wenn wir einen Widerstand verwenden, müssen wir zuerst seinen Widerstand kennen. Hier sind zwei Methoden: Sie können die Bänder am Widerstand beobachten oder den Widerstand mit einem Multimeter messen. Es wird empfohlen, die erste Methode zu verwenden, da diese bequemer und schneller ist. Verwenden Sie ein Multimeter, um den Wert zu messen.

Wie auf der Karte gezeigt, steht jede Farbe für eine Nummer.



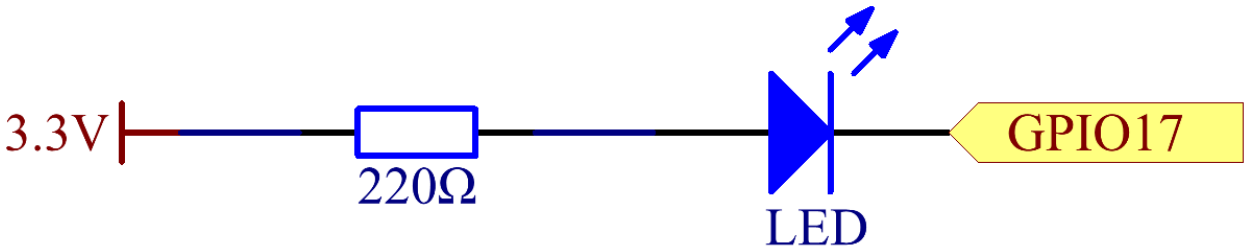
Schematische Darstellung

In diesem Experiment schließen Sie einen 220 -Widerstand an die Anode (den langen Pin der LED) und dann den Widerstand an 3,3 V an und verbinden Sie die Kathode (den kurzen Pin) der LED mit GPIO17 von Raspberry Pi. Da um eine LED einzuschalten, muss der GPIO17-Niveau niedrig (0V) sein. Wir können dieses Phänomen durch Programmierung erhalten.

Bemerkung: **Pin11** bezieht sich von links nach rechts auf den 11. Pin des Raspberry Pi. Die entsprechenden **wiringPi** und BCM-Pin-Nummern sind in der folgenden Tabelle aufgeführt.

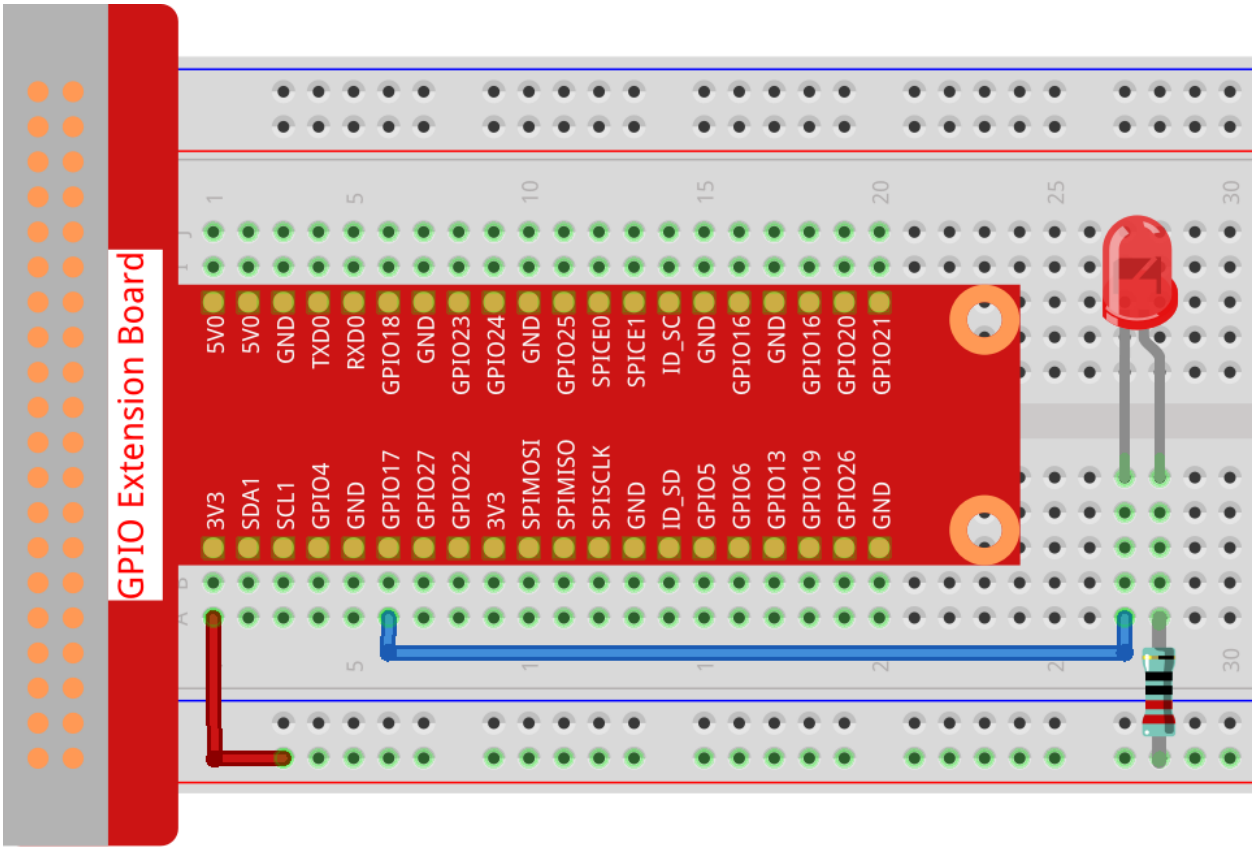
In den C-Sprachinhalten machen wir GPIO0 sz bis 0 in der wiringPi. U Unter den Python-Sprachinhalten, BCM 17 ist 17 in der BCM-Spalte der folgenden Tabelle. Gleichzeitig sind sie gleich mit dem 11. Pin des Raspberry Pi, Pin 11.

T-Karte Name	physisch	wiringPi	BCM
GPIO17	Pin 11	0	17



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



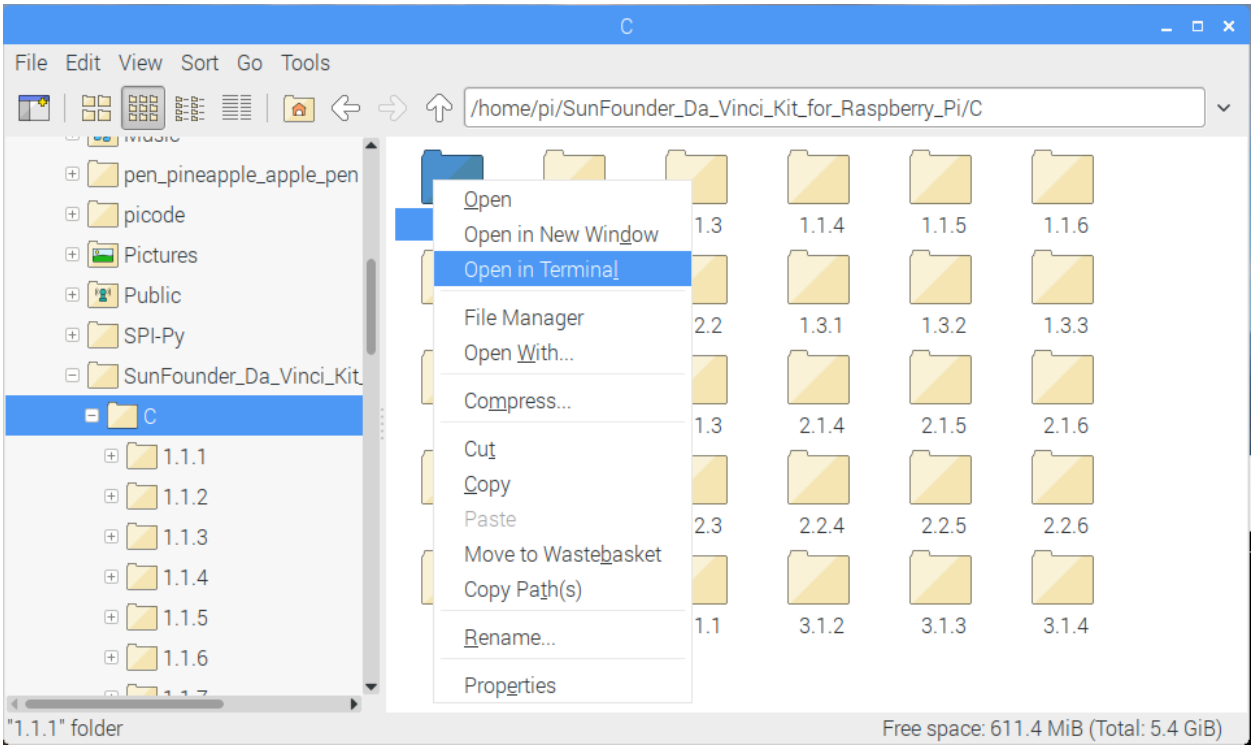
Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner des Codes.

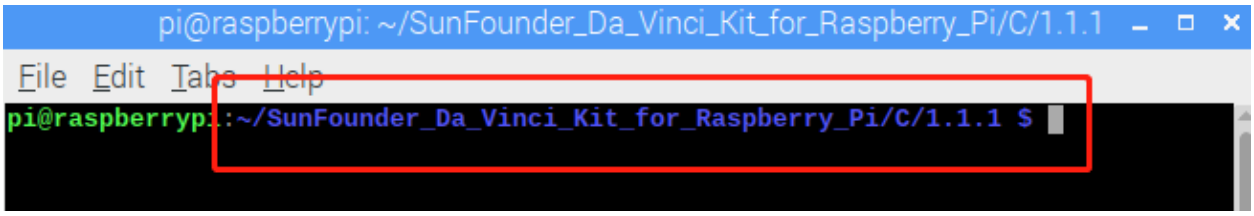
1. Wenn Sie einen Bildschirm verwenden, sind die folgenden Schritte empfohlen.

Gehen Sie zu `/home/pi/` und suchen Sie den Ordner `davinci-kit-for-raspberry-pi` .

Suchen Sie C im Ordner, klicken Sie mit der rechten Maustaste darauf und wählen **Open in Terminal**.



Dann öffnet sich ein Fenster wie unten gezeigt. Nun haben Sie den Pfad des Codes **1.1.1_BlinkingLed.c** eingegeben.



In den folgenden Lektionen verwenden wir den Befehl, um die Codedatei einzugeben, anstatt mit der rechten Maustaste zu klicken. Aber Sie können die Methode wählen, die Sie bevorzugen.

2. Wenn Sie sich beim Raspberry Pi aus der Ferne anmelden, wechseln Sie mit `cd` das Verzeichnis:

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

Bemerkung: Ändern Sie das Verzeichnis in den Codepfad in diesem Experiment durch `cd`.

In jedem Fall befinden Sie sich jetzt im Ordner C. Die weiteren Verfahren, die auf diesen beiden Methoden basieren, sind gleich. Lass uns weitermachen.

Schritt 3: Kompilieren Sie die Kode

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

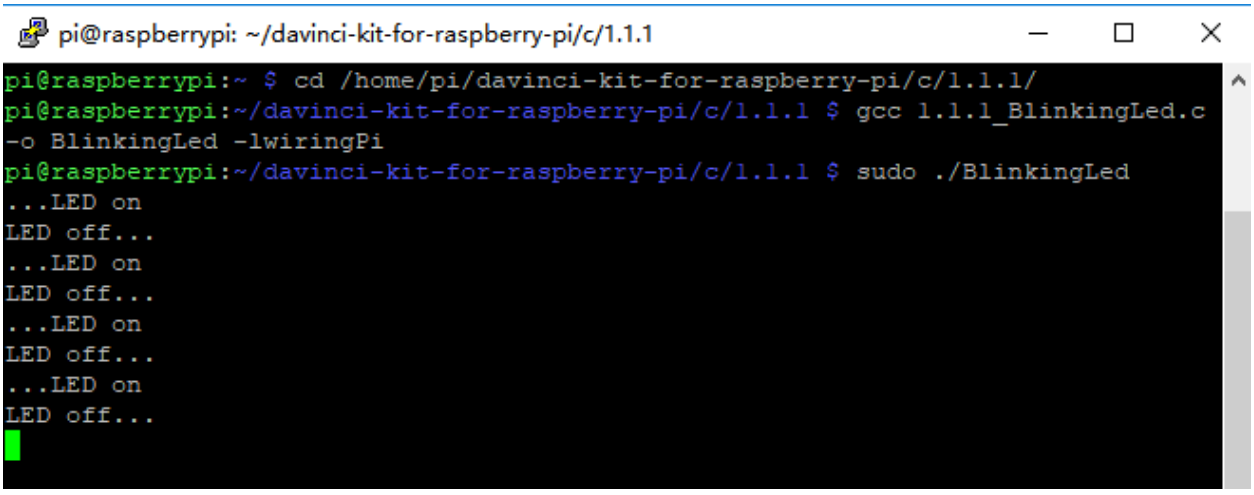
Bemerkung: note: gcc ist die gcc ist die GNU Compilersammlung.. Hier funktioniert es wie das Kompilieren der C-Sprachdatei `1_BlinkingLed.c` und das Ausgeben einer ausführbaren Datei.

Im Befehl bedeutet `-o` die Ausgabe (das Zeichen unmittelbar nach `-o` ist die Dateinamenausgabe nach der Kompilierung, und eine ausführbare Datei namens `BlinkingLed` wird hier generiert) , `-lwiringPi` ist das Laden der Bibliothek wiringPi (`l` ist die Abkürzung der Bibliothek).

Schritt 4: Führen Sie die Ausgabe der ausführbaren Datei im vorherigen Schritt aus.

```
sudo ./BlinkingLed
```

Bemerkung: Um das GPIO zu steuern, müssen Sie das Programm mit dem Befehl sudo (Superuser do) ausführen. Der Befehl `./` gibt das aktuelle Verzeichnis an. Der gesamte Befehl ist für die Ausführung der `BlinkingLed` im aktuellen Verzeichnis.



Nachdem der Kode ausgeführt wurde, blinkt die LED.

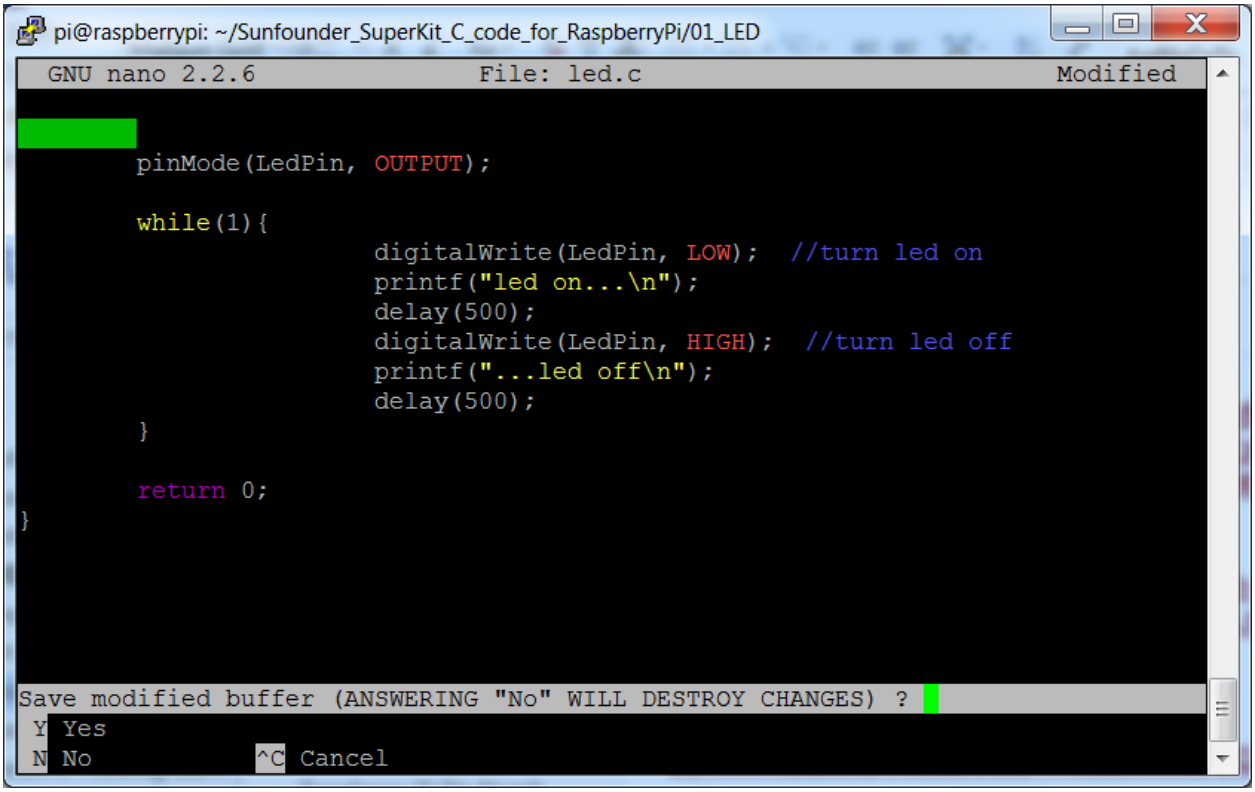
Bemerkung: Wenn es nach dem Ausführen nicht funktioniert oder eine Fehlermeldung angezeigt wird `wiringPi.h: No such file or directory`, please refer to *C-Code funktioniert nicht?* .

Wenn Sie die Kodedatei `1.1.1_BlinkingLed.c` bearbeiten möchten, drücken Sie `Ctrl + C` , um die Ausführung des Kodes zu beenden. Geben Sie dann den folgenden Befehl ein und öffnen es:

```
nano 1.1.1_BlinkingLed.c
```

Bemerkung: nano ist ein Texteditor. Der Befehl wird verwendet, um die Kodedatei `1.1.1_BlinkingLed.c` mit diesem Tool zu öffnen.

Drücken Sie `Ctrl+X` für Ausfahrt Wenn Sie die Kode geändert haben, werden Sie gefragt, ob Sie die Änderungen speichern möchten oder nicht. Geben Sie `Y` (speichern) oder `N` (nicht speichern) ein. Drücken Sie dann die `Enter` , um den Vorgang zu beenden. Wiederholen Sie `Schritt 3` und `Schritt 4` , um den Effekt nach dem Ändern zu sehen.



Code

Der Programmkode wird wie folgt angezeigt:

```
#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;
}
```

Code Erklärung

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

Die Hardware-Laufwerksbibliothek wurde für die C-Sprache von Raspberry Pi entwickelt. Das Hinzufügen dieser Bibliothek fördert die Initialisierung der Hardware und die Ausgabe von I/O ports, PWM usw.

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

Standard I/O Bibliothek. Die Pintf-Funktion zum Drucken der auf dem Bildschirm angezeigten Daten wird von dieser Bibliothek realisiert. Es gibt viele andere Leistungsfunktionen, die Sie erkunden können.

```
#define LedPin 0
```

Pin GPIO17 der T_Erweiterungskarte entspricht dem GPIO0 in WiringPi. Weisen Sie Ledpin GPIO0 zu, was GPIO0 im zukünftigen Kode darstellt.

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

Dadurch wird wiringPi initialisiert und geht davon aus, dass das aufrufende Programm das wiringPi Nummerierungsschema verwendet.

Diese Funktion muss mit Root-Rechten aufgerufen werden. Wenn die Initialisierung der Verkabelung fehlgeschlagen ist, drucken Sie die Nachricht auf dem Bildschirm. Die Funktion `return` wird verwendet, um aus der aktuellen Funktion herauszuspringen. Wenn Sie die Funktion Rückgabe in `main()` verwenden, wird das Programm beendet.

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

Stellen Sie LedPin als Ausgabe ein, um einen Wert darauf zu schreiben.

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

Stellen Sie GPIO0 auf 0V (niedriger Niveau) ein. Da die Kathode der LED mit GPIO0 verbunden ist, leuchtet die LED auf, wenn GPIO0 niedrig eingestellt ist. Im Gegenteil, stellen Sie GPIO0 als High-Level ein. `digitalWrite(LedPin, HIGH)`: Die LED erlischt.

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

Die printf-Funktion ist eine Standardbibliotheksfunktion und ihr Funktionsprototyp befindet sich in der Header-Datei `stdio.h`. Die allgemeine Form des Aufrufs lautet: `printf("Format Control String", Spalten der Ausgabetabelle)` . Die Formatsteuerzeichenfolge wird verwendet, um das Ausgabeformat anzugeben, das in Formatzeichenfolge und Nichtformatzeichenfolge unterteilt ist. Die Formatzeichenfolge beginnt mit `%`, gefolgt von Formatzeichen, z. B. `%d` für die Ausgabe von Dezimalzahlen. Unformatierte Zeichenfolgen werden als Prototypen gedruckt. Hier wird eine nicht formatierte Zeichenfolge verwendet, gefolgt von `n`, einem Zeilenumbruchzeichen, das den automatischen Zeilenumbruch nach dem Drucken einer Zeichenfolge darstellt.

```
delay(500);
```

Die `delay(500)` hält den aktuellen HIGH- oder LOW-Status für 500 ms.

Dies ist eine Funktion, die das Programm für einen bestimmten Zeitraum anhält. Und die Geschwindigkeit des Programms wird von unserer Hardware bestimmt. Hier schalten wir die LED ein oder aus. Wenn es keine Verzögerungsfunktion gibt, führt das Programm das gesamte Programm sehr schnell und kontinuierlich aus. Wir brauchen also die Verzögerungsfunktion, um das Programm schreiben und debuggen zu können.

```
return 0;
```

Normalerweise wird es hinter der Hauptfunktion platziert, was anzeigt, dass die Funktion bei erfolgreicher Ausführung 0 zurückgibt.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner des Codes und führen Sie ihn aus.

1. Wenn Sie einen Bildschirm verwenden, sind die folgenden Schritte empfohlen.

Suchen Sie `1.1.1_BlinkingLed.py` und doppelklicken Sie darauf, um es zu öffnen. Jetzt bist du in der Datei.

Klicken Sie im Fenster auf **Run** ->**Run Module** . Der folgende Inhalt wird angezeigt.

Um die Ausführung zu stoppen, klicken Sie einfach auf die X-Schaltfläche oben rechts, um sie zu schließen, und kehren Sie dann zum Code zurück. Wenn Sie die Kode ändern, müssen Sie ihn zuerst speichern, bevor Sie auf **Run Module (F5)** klicken. Dann können Sie die Ergebnisse sehen.

2. Wenn Sie sich beim Raspberry Pi aus der Ferne anmelden, geben Sie den folgenden Befehl ein:

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

Bemerkung: Wechseln Sie in diesem Experiment per `cd` in den Pfad der Kode.

Schritt 3: Führen Sie die Kode aus

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

Bemerkung: Hier tun sudo - superuser do und python bedeutet, die Datei von Python auszuführen.

Nachdem der Kode ausgeführt wurde, blinkt die LED.

Schritt 4: Wenn Sie die Dodedatei `1.1.1_BlinkingLed.py` , bearbeiten möchten, drücken Sie `Ctrl + C`, um die Ausführung der Kode zu beenden. Geben Sie dann den folgenden Befehl ein, um `1.1.1_BlinkingLed.py`: zu öffnen:

```
nano 1.1.1_BlinkingLed.py
```

Bemerkung: nano ist ein Texteditor. Mit dem Befehl wird die Dodedatei `1.1.1_BlinkingLed.py` by this tool.von diesem Tool geöffnet.

Drücken Sie `Ctrl+X` für Ausfahrt Wenn Sie die Kode geändert haben, werden Sie gefragt, ob Sie die Änderungen speichern möchten oder nicht. Geben Sie `Y` (speichern) oder `N` (nicht speichern) ein.

Drücken Sie dann `Enter`, um den Vorgang zu beenden. Geben Sie nano `1.1.1_BlinkingLed.py` erneut ein, um den Effekt nach der Änderung zu sehen.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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()
```

Code Erklärung

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

Wenn das System dies erkennt, durchsucht es den Installationspfad von Python in der Umgebung env und ruft dann den entsprechenden Interpreter auf, um den Vorgang abzuschließen. Dies soll verhindern, dass der Benutzer die Python nicht auf dem Defaultpfad `/usr/bin` installiert.

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

Importieren Sie auf diese Weise die RPi.GPIO-Bibliothek und definieren Sie dann eine Variable, GPIO, um RPI.GPIO in der folgenden Kode zu ersetzen.

```
import time
```

Zeitpaket importieren, für Zeitverzögerungsfunktion im folgenden Programm.

```
LedPin = 17
```

Die LED wird mit dem GPIO17 der T-förmigen Erweiterungskarte, nämlich BCM 17, verbunden.

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

Stellen Sie den LedPin-Modus auf Ausgabe und den AnfangsNiveau auf Hoch (3.3v).

Es gibt zwei Wisen, die IO -Pins eines Raspberry Pi in RPi.GPIO zu nummerieren: BOARD-Nummern und BCM-Nummern. In unseren Lektionen verwenden wir BCM-Nummern. Sie müssen jeden Kanal einrichten, den Sie als Eingang oder Ausgang verwenden.

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

Stellen Sie GPIO17 (BCM17) auf 0V (niedriger Niveau) ein. Da die Kathode der LED mit GPIO17 verbunden ist, leuchtet die LED auf.

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

Verzögerung um 0,5 Sekunden. Hier ähnelt die Anweisung der Verzögerungsfunktion in der Sprache C, die Einheit ist die zweite.

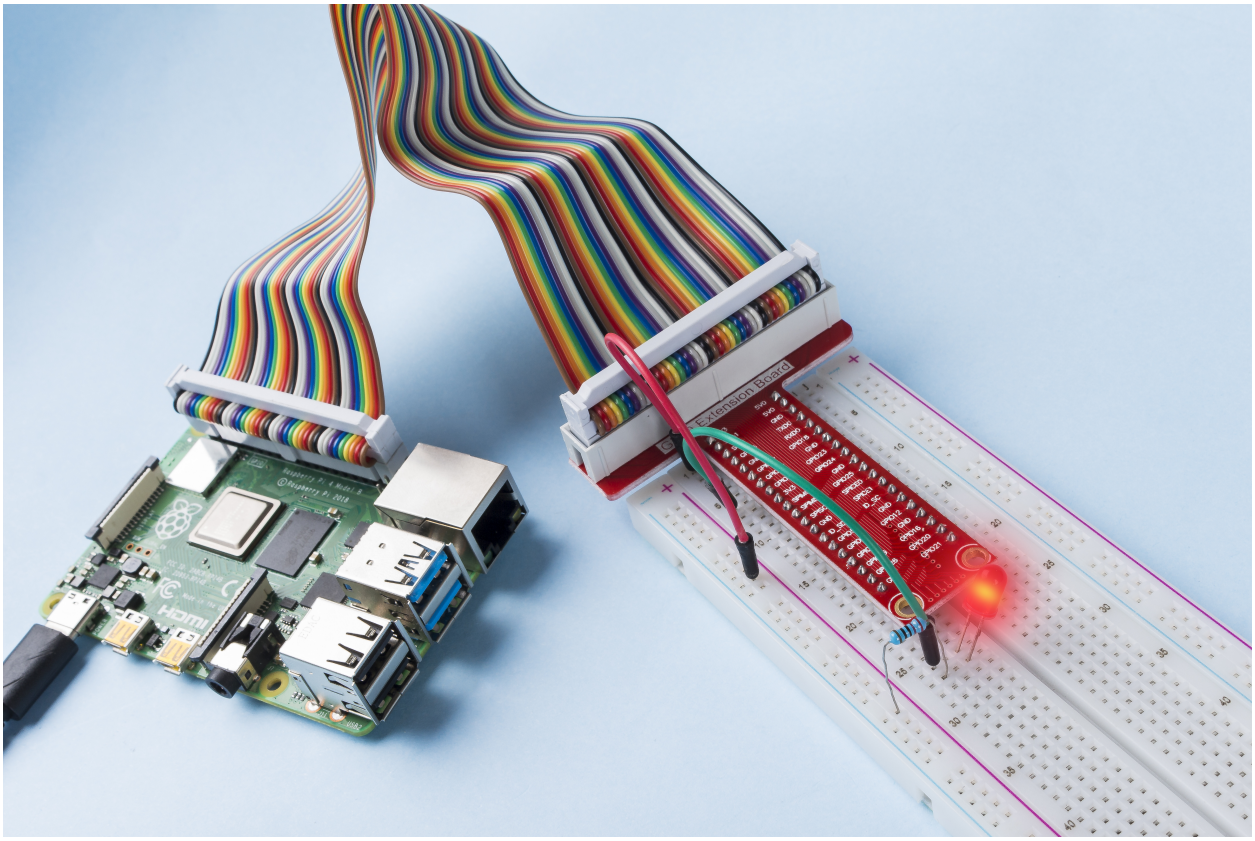
```
def destroy():
    GPIO.cleanup()
```

Definieren Sie eine Zerstörungsfunktion, um alles nach Abschluss des Skripts zu bereinigen.

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

Dies ist die allgemeine Ausführungsstruktur der Kode Wenn das Programm gestartet wird, wird der Pin durch Ausführen von `setup()` initialisiert und anschließend der Kode in der Funktion `main()` ausgeführt, um den Pin auf hohe und niedrige Niveau zu setzen. Wenn `Ctrl+C` gedrückt wird, wird das Programm `destroy()` ausgeführt.

Phänomen Bild

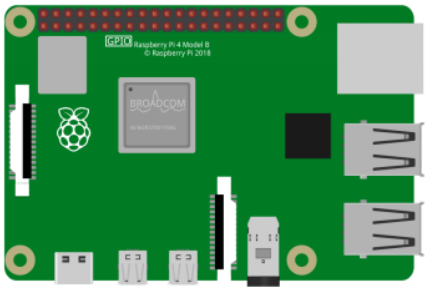
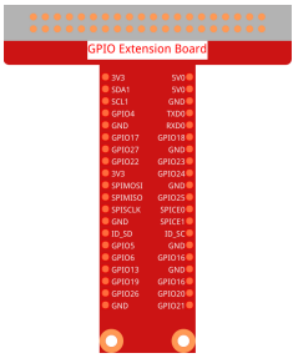
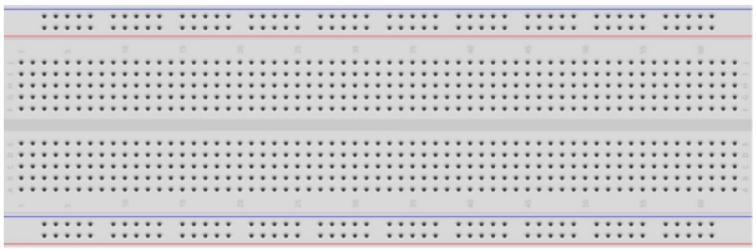


1.7.2 1.1.2 RGB-LED

Einführung

In dieser Lektion steuern wir mit eine RGB-LED, die verschiedene Arten von Farben zu blinken.

Komponenten

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

Prinzip

PWM

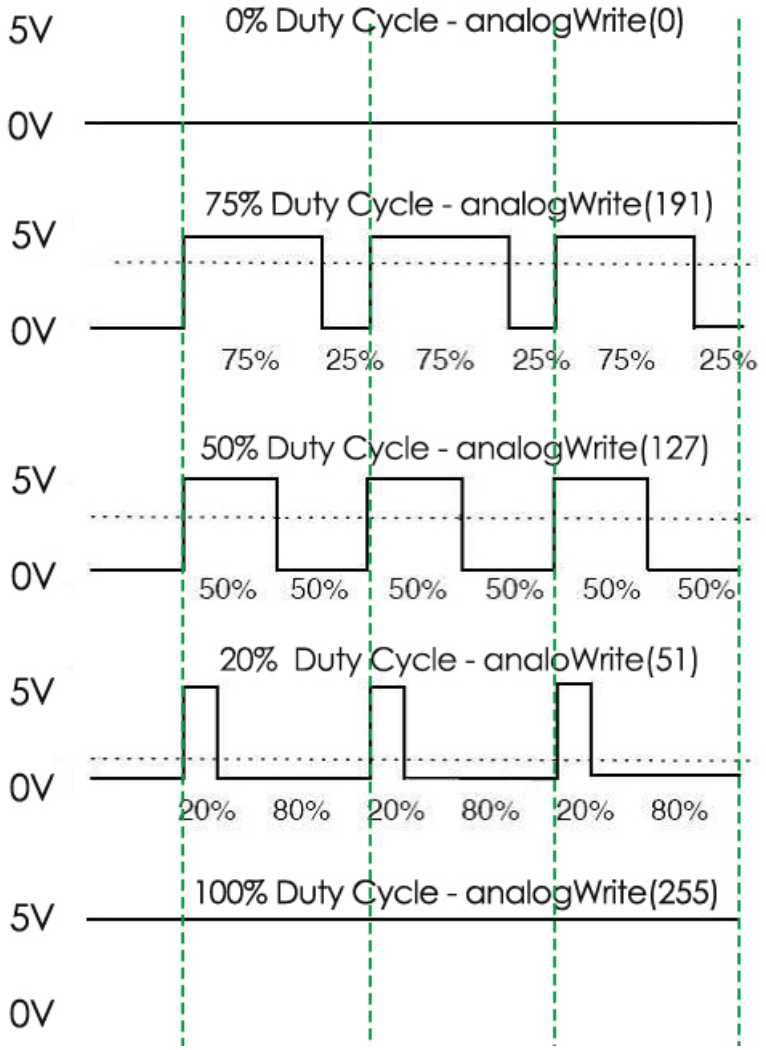
Die Pulsweitenmodulation oder PWM ist eine Technik, mit der analoge Ergebnisse mit digitalen Mitteln erzielt werden können. Die digitale Steuerung wird verwendet, um eine Rechteckwelle zu erzeugen, um ein Signal zwischen Ein und Aus zu schalten. Dieses Ein-Aus-Muster kann Spannungen zwischen Voll-Ein (5 Volt) und Aus (0 Volt) simulieren, mit der Änderung von Zeit, in dem das Signal an ist, gegenüber der Zeit, in der das Signal abläuft. Die Dauer der „Einschaltzeit“ wird als Impulsbreite bezeichnet. Um unterschiedliche analoge Werte zu erhalten, ändern oder modulieren Sie diese Breite. Wenn Sie dieses Ein-Aus-Muster mit einem Gerät, beispielsweise einer LED, schnell genug wiederholen, sieht das Ergebnis folgendermaßen aus: Das Signal ist eine konstante Spannung zwischen 0 und 5 V, die die Helligkeit der LED steuert.

Arbeitszyklus

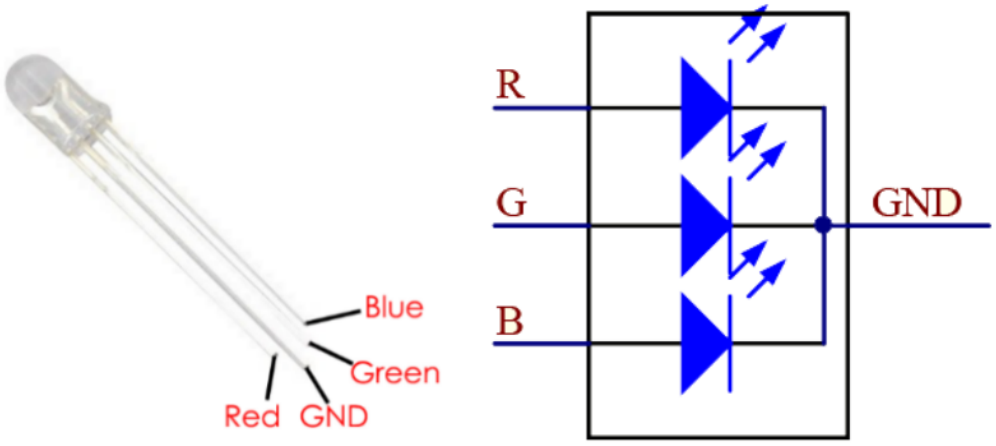
Ein Arbeitszyklus ist der Prozentsatz einer Periode, in der ein Signal aktiv ist. Eine Periode ist die Zeit, die ein Signal benötigt, um einen Ein- und Ausschaltzyklus abzuschließen. Als Formel kann ein Arbeitszyklus ausgedrückt werden als:

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

Wobei **D** das Tastverhältnis ist, **T** die Zeit ist, zu der das Signal aktiv ist, und **P** die Gesamtperiode des Signals ist. Ein Tastverhältnis von 60% bedeutet also, dass das Signal in 60% der Fälle eingeschaltet ist, in 40% der Fälle jedoch ausgeschaltet ist. Die „Pünktlichkeit“ für einen Arbeitszyklus von 60% kann je nach Dauer des Zeitraums einen Bruchteil einer Sekunde, eines Tages oder sogar einer Woche betragen.



RGB LED

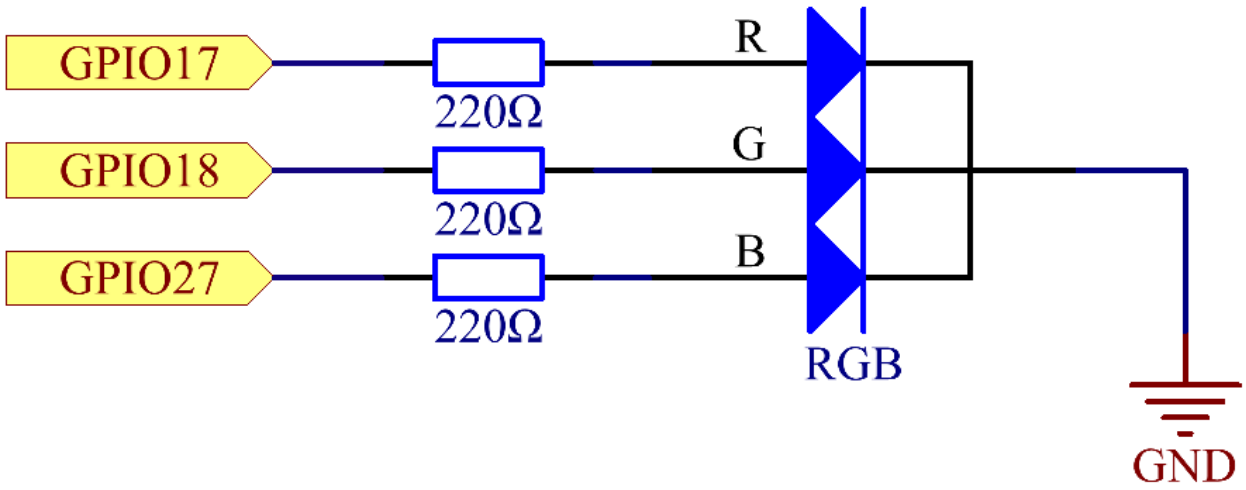


Die drei Primärfarben der RGB-LED können durch Helligkeit in verschiedene Farben gemischt werden. Die Helligkeit der LED kann mit PWM eingestellt werden. Raspberry Pi hat nur einen Kanal für die Hardware-PWM-Ausgabe, benötigt jedoch drei Kanäle zur Steuerung der RGB-LED, was bedeutet, dass es schwierig ist, die RGB-LED mit der Hardware-PWM von Raspberry Pi zu steuern. Glücklicherweise simuliert die softPwm-Bibliothek PWM (softPwm) durch Programmierung. Alles, was Sie tun müssen, ist die Titeldatei SoftPwm.h (C-Sprachbenutzer) und rufen Sie die API, die es bietet für die einfache Steuerung der RGB-LED durch die Mehrkanal-PWM-Ausgabe, um eine Vielzahl von Farben anzuzeigen.

Schematische Darstellung

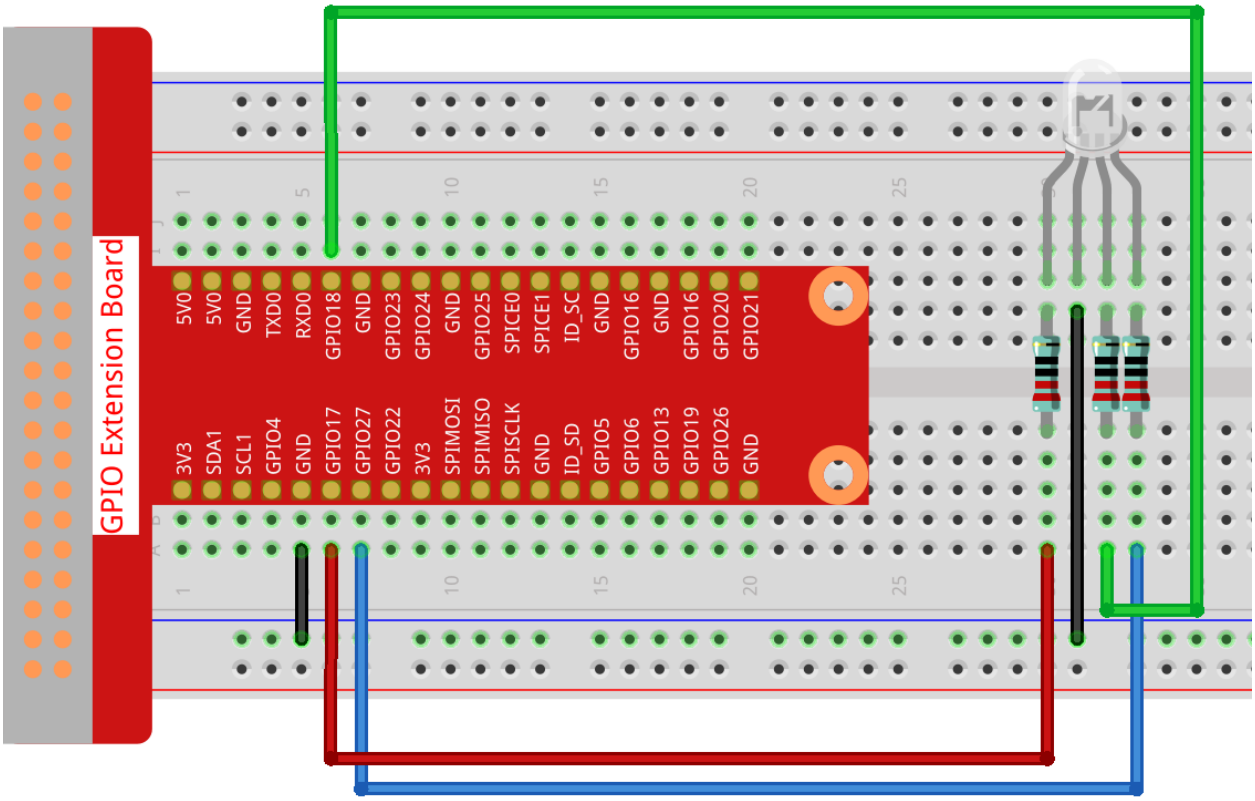
Nachdem Sie die Pins von R, G und B mit einem Strombegrenzungswiderstand verbunden haben, verbinden Sie sie mit dem GPIO17, GPIO18 bzw. GPIO27. Der längste Pin (GND) der LED ist mit dem GND des Raspberry Pi verbunden. Wenn die drei Pins unterschiedliche PWM-Werte erhalten, zeigt die RGB-LED unterschiedliche Farben an.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Kompilieren Sie die Kode.

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

Bemerkung: Wenn die Anweisung `gcc` ausgeführt wird und `-o` nicht aufgerufen wird, wird ausführbare Datei `a.out` benannt.

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, sehen Sie, dass RGB Rot, Grün, Blau, Gelb, Pink und Cyan anzeigt.

Code

```c
#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");
        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;
}
```

Code Erklärung

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

Bibliothek zur Realisierung der PWM-Funktion der Software.

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

Die Funktion besteht darin, mit Software einen PWM-Pin zu erstellen und dessen Periode zwischen 0x100us und 100x100us einzustellen.

Der Prototyp der Funktion `softPwmCreate(LedPinRed, 0, 100)` lautet wie folgt:

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

- **Parameter pin:** Jeder GPIO-Pin von Raspberry Pi kann als PWM-Pin gesetzt werden.
- **Parameter initialValue:** Die anfängliche Impulsbreite ist der initialValue times100us.
- **Parameter pwmRange:** Die Periode von PWM ist die pwmRange times100us.

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

Diese Funktion dient zum Einstellen der Farben der LED. Bei Verwendung von RGB repräsentiert der formale Parameter `r_val` die Luminanz des roten, `g_val` des grünen, `b_val` des blauen.

Der Prototyp der Funktion `softPwmWrite(LedPinBlue, b_val)` lautet wie folgt:

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

- **Parameter pin:** Jeder GPIO-Pin von Raspberry Pi kann als PWM-Pin gesetzt werden.
- **Parameter Value:** Die Pulsbreite von PWM ist Wert mal 100us. Beachten Sie, dass der Wert nur kleiner als der zuvor definierte pwmRange sein kann. Wenn er größer als pwmRange ist, erhält der Wert einen festen Wert, pwmRange.

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

Rufen Sie die zuvor definierte Funktion auf. Schreiben Sie 0xff in LedPinRed und 0x00 in LedPinGreen und LedPinBlue. Nach dem Ausführen der Kode leuchtet nur die rote LED auf. Wenn Sie LEDs in anderen Farben aufleuchten möchten, ändern Sie einfach die Parameter.

Für Python-Sprachbenutzer

Schritt 2: Öffnen Sie die Kodedatei.

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

Schritt 3: Ausführen.

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

Nachdem die Kode ausgeführt wurde, sehen Sie, dass RGB Rot, Grün, Blau, Gelb, Pink und Cyan anzeigt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

# Set up a color table in Hexadecimal
COLOR = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]
# Set pins' channels with dictionary
pins = {'Red':17, 'Green':18, 'Blue':27}


def setup():
    global p_R, p_G, p_B
    # 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):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# 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.

    # 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()
```

Code Erklärung

```
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)
```

Rufen Sie die Funktion `GPIO.PWM()` auf, um Rot, Grün und Blau als PWM-Pins zu definieren und die Frequenz der PWM-Pins auf 2000 Hz einzustellen. Verwenden Sie dann die Funktion `Start()` , um den anfänglichen Arbeitszyklus auf Null zu setzen.

```
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
```

Definieren Sie eine MAP-Funktion zum Zuordnen von Werten. Zum Beispiel ist x = 50, in_min = 0, in_max = 255, out_min = 0, out_max = 100. Nach der Zuordnung der Kartenfunktion wird (50-0) * (100-0)/(255-0) +0=19.6, zurückgegeben, was bedeutet, dass 50 in 0-255 19,6 in 0-100 entspricht.

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

Konfiguriert die Luminanz der drei LEDs mit dem eingegebenen Farbwert. Weisen Sie R_val die ersten beiden Hexadezimalwerte zu, G_val die beiden mittleren und B_val die letzten beiden Werte. Wenn beispielsweise color = 0xFF00FF ist, ist R_val = 0xFF00FF & 0xFF0000 >> 16 = 0xFF, G_val = 0x00, B_val = 0xFF.

```
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)
```

Verwenden Sie die Zuordnungsfunktion, um den R-, G-, B-Wert zwischen 0 und 255 in den PWM-Arbeitszyklusbereich von 0 bis 100 abzubilden.

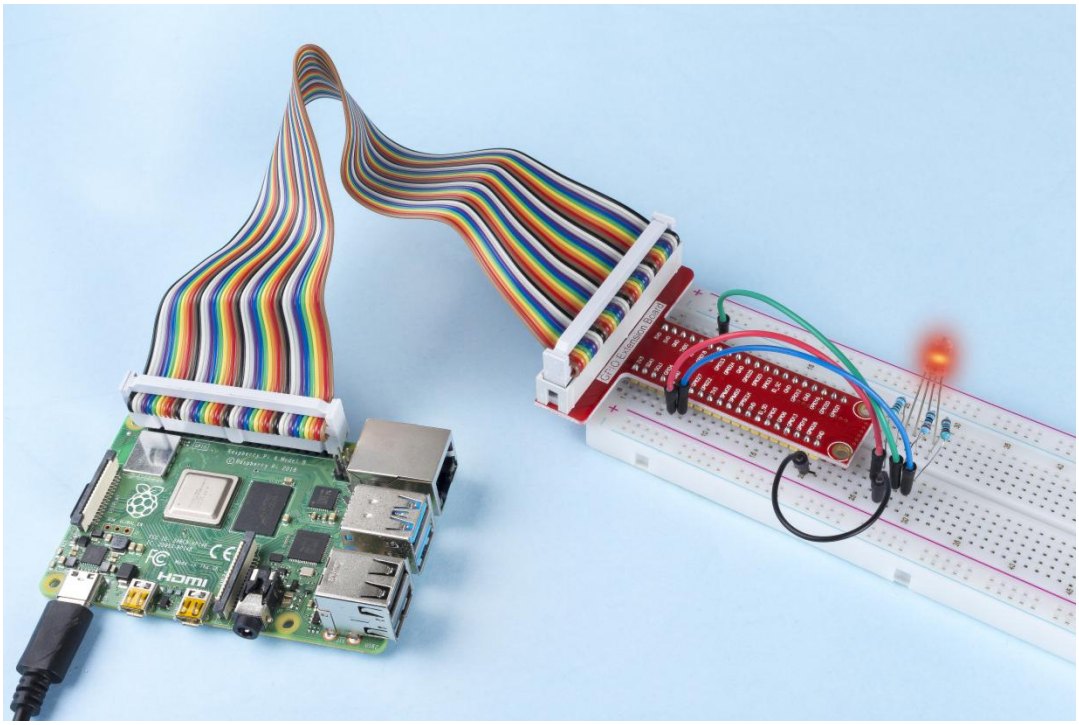
```
p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

Weisen Sie den zugeordneten Tastverhältniswert dem entsprechenden PWM-Kanal zu, um die Luminanz zu ändern.

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

Ordnen Sie jedes Element in der COLOR-Liste der jeweiligen Farbe zu und ändern Sie die Farbe der RGB-LED über die Funktion `setColor()` .

Phänomen Bild

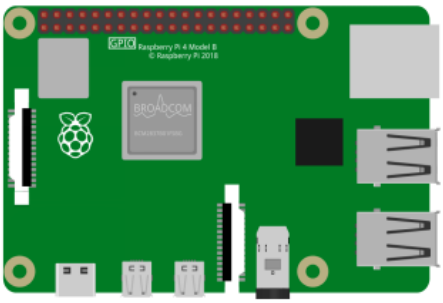
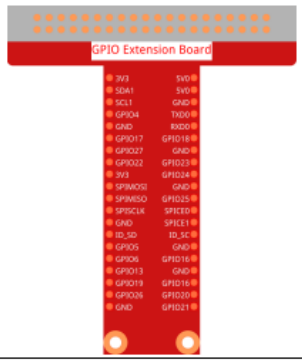
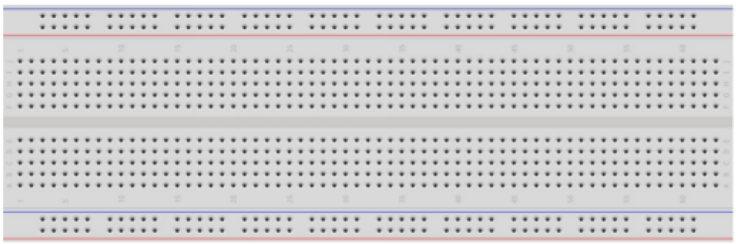


1.7.3 1.1.3 LED-Balkendiagramm

Einführung

In diesem Projekt beleuchten wir nacheinander die Lichter auf dem LED-Balkendiagramm.

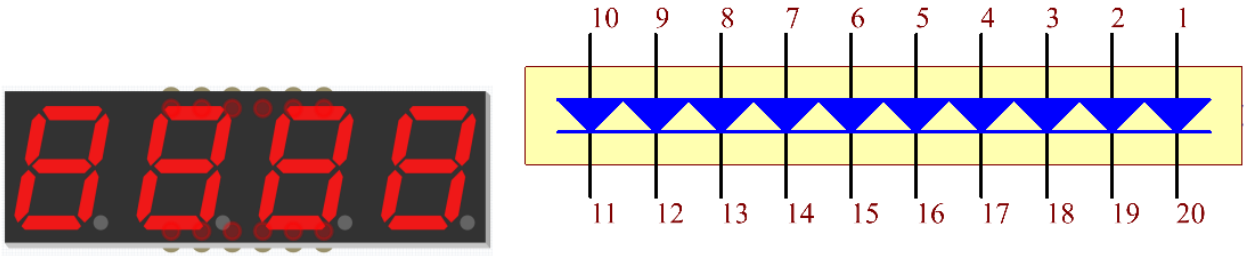
Komponenten

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

Prinzip

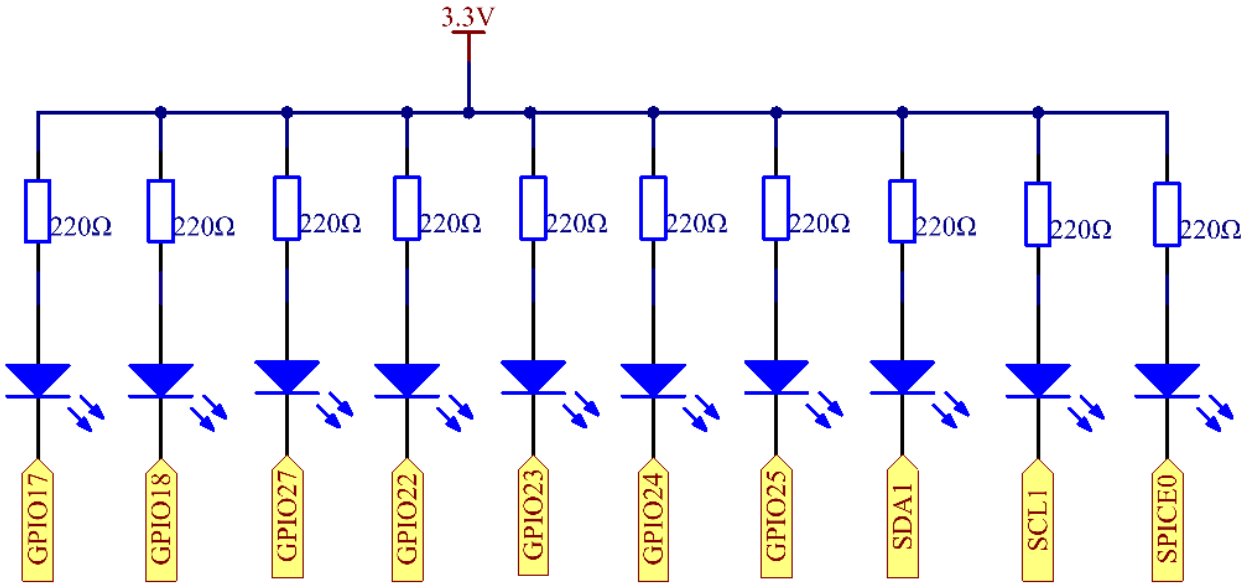
LED-Balkendiagramm

LED-Balkendiagramm ist ein LED-Array, das zur Verbindung mit einer elektronischen Schaltung oder einem Mikrocontroller verwendet wird. Es ist einfach, ein LED-Balkendiagramm mit der Schaltung zu verbinden, es ist wie 10 einzelne LEDs mit 10 Ausgangspins. Im Allgemeinen können wir das LED-Balkendiagramm als Batteriestandsanzeige, Audiogeräte und industrielle Bedienfelder verwenden. Es gibt viele andere Anwendungen von LED-Balkendiagrammen.



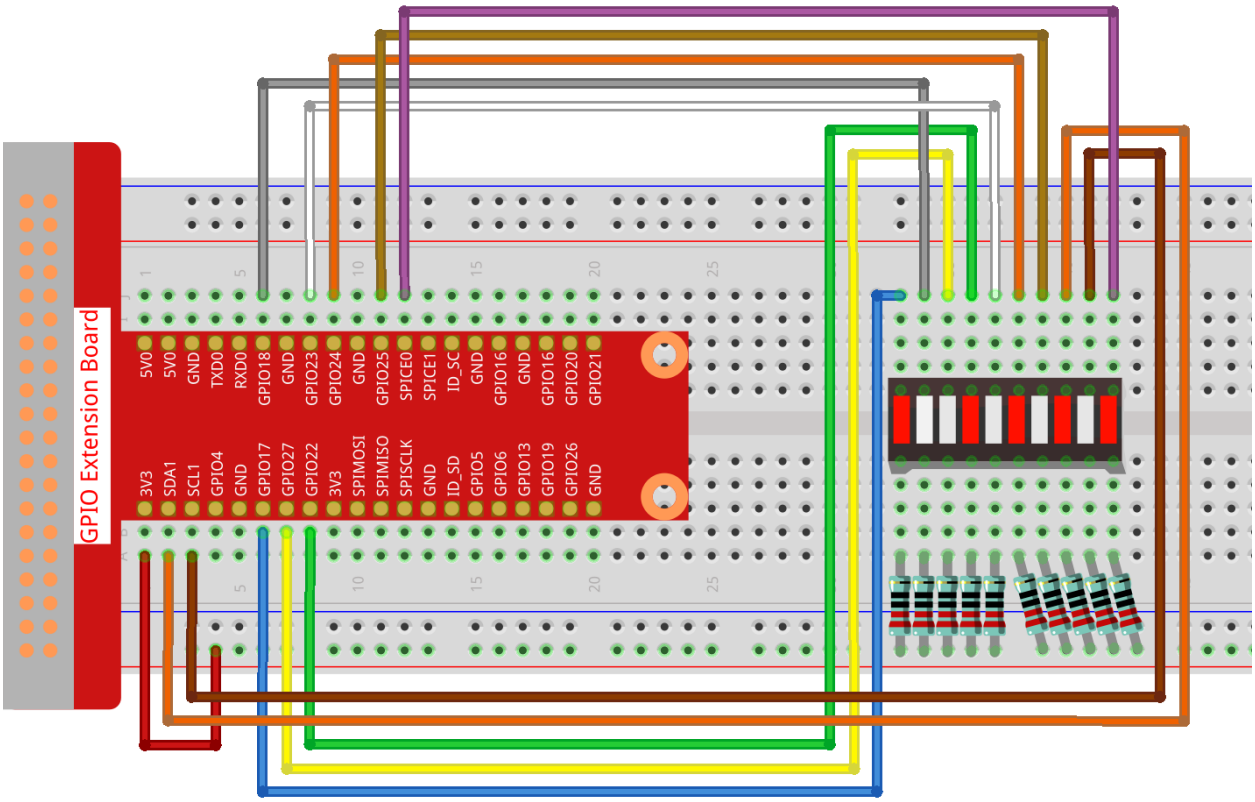
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, werden die LEDs in der LED-Leiste regelmäßig ein- und ausgeschaltet.

Code

```
#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++){
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        int j=i*2;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
void evenLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2+1;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
void allLedBarGraph(void){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
        delay(300);
        digitalWrite(pins[i],LOW);
    }
}
int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(int i=0;i<10;i++){       //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],LOW);
    }
    while(1){
        oddLedBarGraph();
        delay(300);
        evenLedBarGraph();
        delay(300);
        allLedBarGraph();
        delay(300);
    }
    return 0;
}
```

Code Erklärung

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

Erstellen Sie ein Array und weisen Sie es der Pin-Nummer zu, die dem LED-Balkendiagramm (0,1,2,3,4,5,6,8,9,10) entspricht. Das Array wird zur Steuerung der LED verwendet.

```
void oddLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Lassen Sie die LED an der ungeraden Stelle des LED-Balkendiagramms der Reihe nach leuchten.

```
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);
    }
}
```

Schalten Sie die LED auf der geraden Ziffer des LED-Balkendiagramms der Reihe nach ein.

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

Lassen Sie die LED auf dem LED-Balkendiagramm nacheinander leuchten.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, werden die LEDs in der LED-Leiste regelmäßig ein- und ausgeschaltet.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

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

def setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.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()
```

Code Erklärung

`ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]` Erstellen Sie ein Array und weisen Sie es der Pin-Nummer zu, die dem LED-Balkendiagramm (11, 12, 13, 15, 16, 18, 22, 3, 5, 24) entspricht. Das Array wird zur

Steuerung der LED verwendet.

```
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)
```

Lassen Sie die LED an der ungeraden Stelle des LED-Balkendiagramms der Reihe nach leuchten.

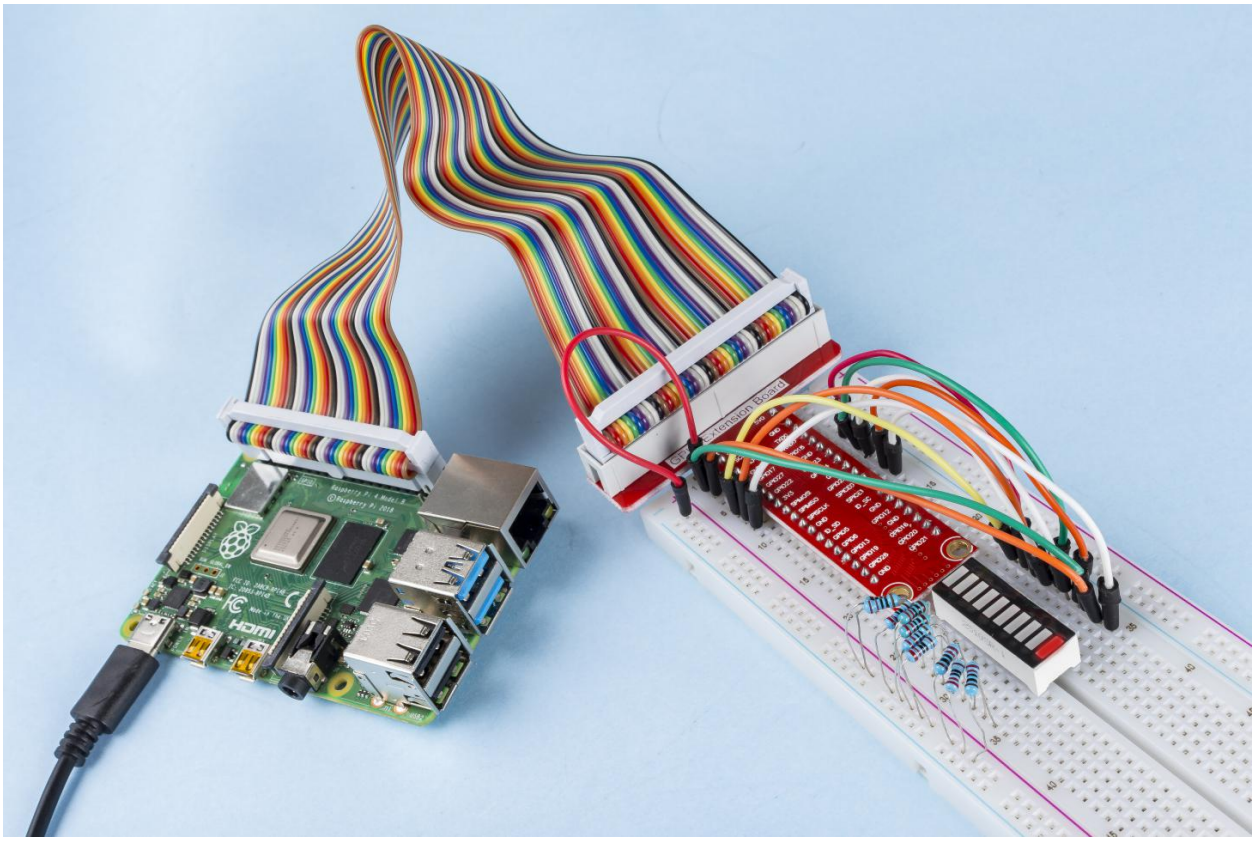
```
def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)
```

Schalten Sie die LED auf der geraden Ziffer des LED-Balkendiagramms der Reihe nach ein.

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

Lassen Sie die LED auf dem LED-Balkendiagramm nacheinander leuchten.

Phänomen Bild

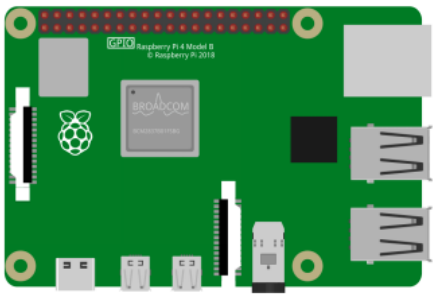
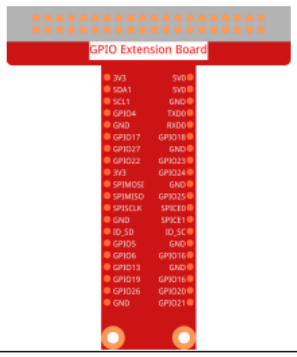
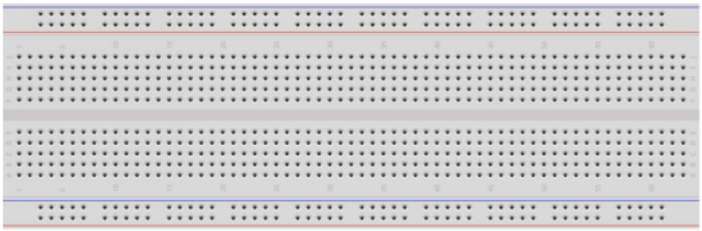


1.7.4 1.1.4 7-Segment-Anzeige

Einführung

Versuchen wir, eine 7-Segment-Anzeige anzutreiben, um eine Nummer von 0 bis 9 und von A bis F anzuzeigen.

Komponenten

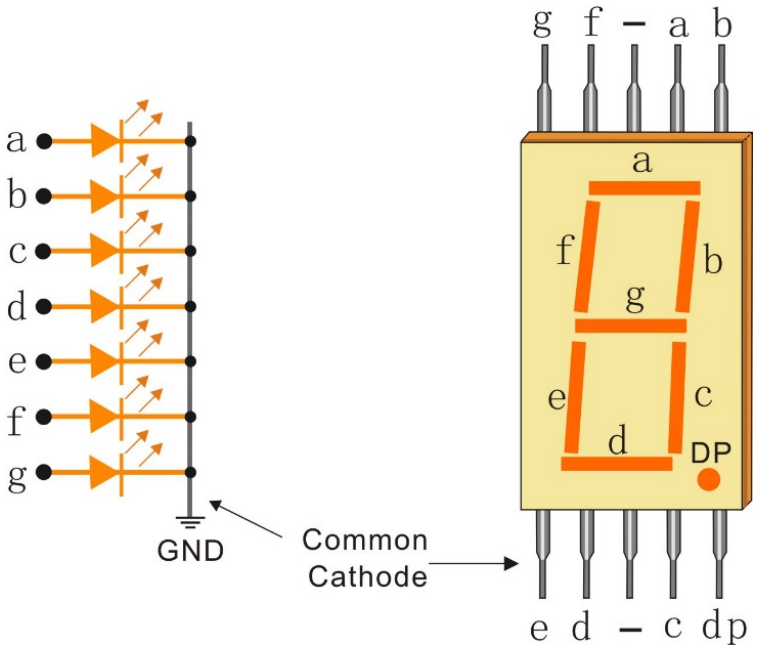
1 * Raspberry Pi 	1 * T-Extension Board 	1 * 7-segment display
1 * 40-pin Cable		1 * Resistor(220Ω)
		Several Jumper Wires
1 * Breadboard 		1 * 74HC595

Prinzip

7-Segment-Anzeige

Ein 7-Segment-Display ist eine 8-förmige Komponente, die 7 LEDs enthält. Jede LED wird als Segment bezeichnet. Bei Erregung ist ein Segment Teil einer anzuzeigenden Ziffer.

Es gibt zwei Typen von Pin-Verbindungen: Common Cathode (CC) und Common Anode (CA). Wie der Name schon sagt, sind an einer CC-Anzeige alle Kathoden der 7 LEDs angeschlossen, wenn an einer CA-Anzeige alle Anoden der 7 Segmente angeschlossen sind. In diesem Kit verwenden wir der erstere.



Jede der LEDs im Display erhält ein Positionssegment, wobei einer der Verbindungsstifte aus dem rechteckigen Kunststoffgehäuse herausgeführt wird. Diese LED-Pins sind von *a* bis *g* bezeichnet und repräsentieren jede einzelne LED. Die anderen LED-Pins sind miteinander verbunden und bilden einen gemeinsamen Pin. Wenn Sie also die entsprechenden Pins der LED-Segmente in einer bestimmten Reihenfolge nach vorne vorspannen, werden einige Segmente heller und andere dunkel bleiben, wodurch das entsprechende Zeichen auf dem Display angezeigt wird.

Koden anzeigen

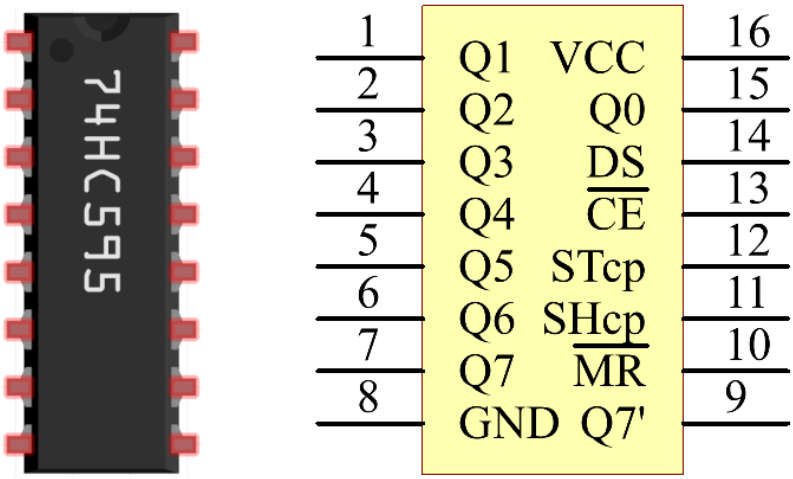
Damit Sie wissen können, wie 7-Segment-Anzeigen (Common Cathode) Numbers anzeigen, haben wir die folgende Tabelle gezeichnet. Nummer sind die Nummer 0-F, die auf der 7-Segment-Anzeige angezeigt werden. (DP) GFEDCBA bezieht sich auf die entsprechende LED, die auf 0 oder 1 gesetzt ist. Beispielsweise bedeutet 00111111, dass DP und G auf 0 gesetzt sind, während andere auf 1 gesetzt sind. Daher wird die Nummer 0 auf dem 7-Segment-Display angezeigt, während der HEX-Kode der Hexadezimalzahl entspricht.

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

Der 74HC595 besteht aus einem 8-Bit-Schieberegister und einem Speicherregister mit parallelen Ausgängen mit drei Zuständen. Es wandelt den seriellen Eingang in einen parallelen Ausgang um, sodass Sie E / A-Ports einer MCU speichern können.

Wenn MR (Pin 10) einen hohen Niveaul und OE (Pin 13) einen niedrigen Niveaul aufweist, werden Daten in die ansteigende Flanke von SHcp eingegeben und gehen über die ansteigende Flanke von SHcp in das Speicherregister. Wenn die beiden Takte miteinander verbunden sind, ist das Schieberegister immer einen Impuls früher als das Speicherregister. Es gibt einen seriellen Verschiebungseingangspin (Ds), einen seriellen Ausgangspin (Q) und eine asynchrone Rücksetztaste (niedriger Niveaul) im Speicherregister. Das Speicherregister gibt einen Bus mit einem parallelen 8-Bit und in drei Zuständen aus. Wenn OE aktiviert ist (niedriger Niveaul), werden die Daten im Speicherregister an den Bus ausgegeben.



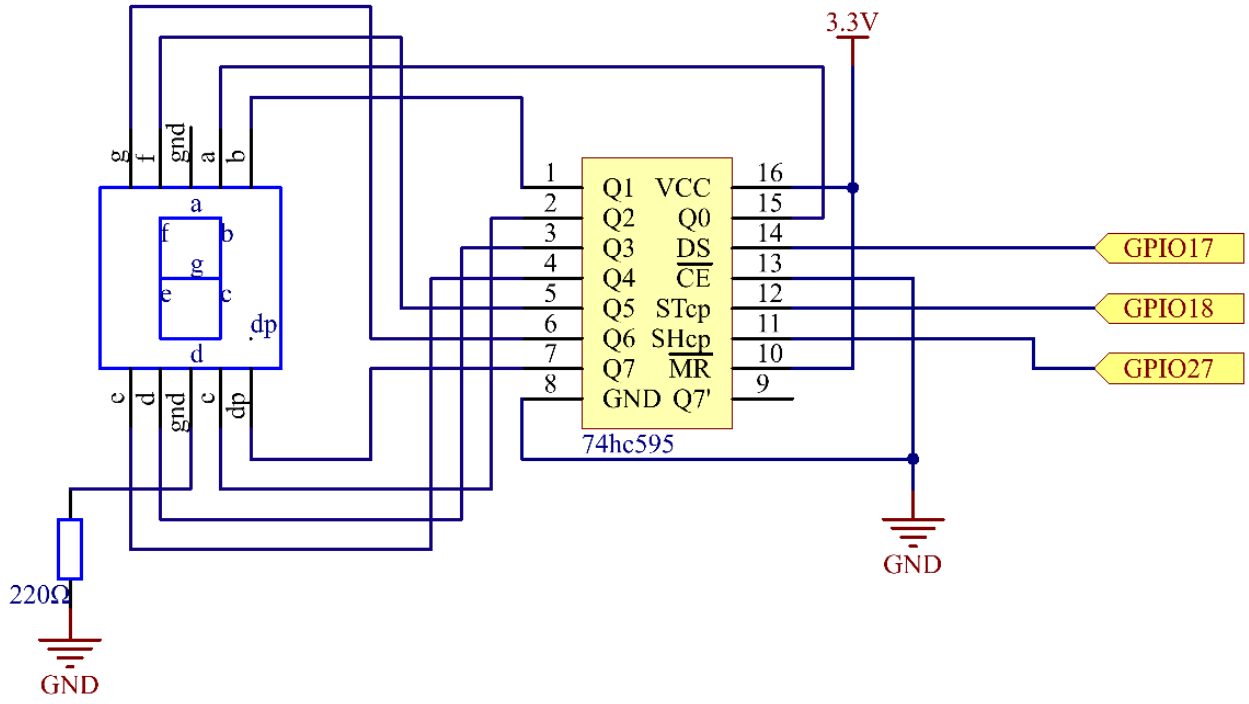
Pins von 74HC595 und ihre Funktionen::

- **Q0-Q7** : 8-Bit-Parallel-Datenausgangspins, die 8 LEDs oder 8 Pins der 7-Segment-Anzeige direkt steuern können.
- **Q7'** : Serienausgangspin, verbunden mit DS eines anderen 74HC595, um mehrere 74HC595 in Reihe zu schalten.
- **MR** : Reset-Pin, aktiv bei niedrigem Niveau;
- **SHcp** : Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.
- **STcp** : Zeitfolgeeingabe des Speicherregisters. Bei der steigenden Flanke werden Daten im Schieberegister in das Speicherregister verschoben.
- **CE** : Ausgangsfreigabepin, aktiv auf niedrigem Niveau.
- **DS** : Serieller Dateneingangspin
- **VCC** : Positive Versorgungsspannung
- **GND** : Boden

Schematische Darstellung

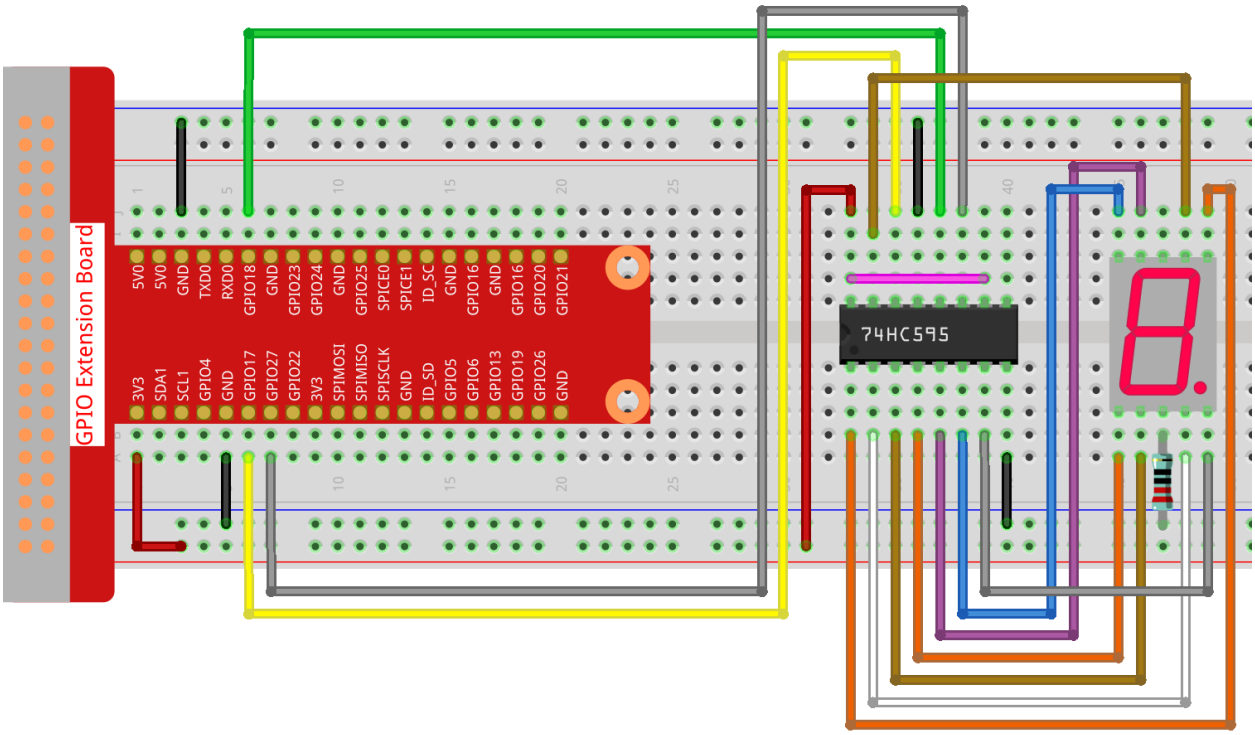
Verbinden Sie Pin ST_CP von 74HC595 mit Raspberry Pi GPIO18, SH_CP mit GPIO27, DS mit GPIO17 und parallele Ausgangsanschlüsse mit 8 Segmenten der LED-Segmentanzeige. Geben Sie Daten in den DS-Pin in das Schieberegister ein, wenn sich SH_CP (der Takteingang des Schieberegisters) an der ansteigenden Flanke befindet, und in das Speicherregister, wenn sich ST_CP (der Takteingang des Speichers) an der ansteigenden Flanke befindet. Anschließend können Sie die Zustände von SH_CP und ST_CP über die Raspberry Pi-GPIOs steuern, um die serielle Dateneingabe in eine parallele Datenausgabe umzuwandeln, um Raspberry Pi-GPIOs zu speichern und die Anzeige zu steuern.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Kompilieren.

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

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Nachdem der Code ausgeführt wurde, wird die 7-Segment-Anzeige 0-9, A-F angezeigt.

Code

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

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

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

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

Ein Segmentcode-Array von 0 bis F in hexadezimaler Darstellung (gemeinsame Kathode).

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

Setzen Sie `ds` , `st_cp` , `sh_cp` drei Pins auf OUTPUT und den Anfangszustand auf 0.

```
void hc595_shift(unsigned char dat){}
```

Zuweisen eines 8-Bit-Werts zum Schieberegister des 74HC595.

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

Ordnen Sie die Datendaten SDI (DS) in Bits zu. Hier nehmen wir an, dass `dat = 0x3f` (0011 1111, wenn i = 2, 0x3f 2 Bits nach links (<<) verschiebt. `1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000` ist wahr.

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

Der Anfangswert von SRCLK wurde auf 0 gesetzt, und hier wird er auf 1 gesetzt, um einen Anstiegsflankenimpuls zu erzeugen und dann das DS-Datum in das Schieberegister zu verschieben.

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

Der Anfangswert von RCLK wurde auf 0 gesetzt, und hier wird er auf 1 gesetzt, um eine ansteigende Flanke zu erzeugen und dann Daten vom Schieberegister zum Speicherregister zu verschieben.

```
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);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        }
    }
```

In dieser for-Schleife verwenden wir `%1X` , um i als Hexadezimalzahl auszugeben. Wenden Sie i an, um den entsprechenden Segmentcode im `SegCode[]` -Array zu finden, und verwenden Sie `hc595_shift()`, um den SegCode in das Schieberegister des 74HC595 zu übergeben.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Ausführen.

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

Nachdem der Code ausgeführt wurde, wird die 7-Segment-Anzeige 0-9, A-F angezeigt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
    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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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()
```

Code Erklärung

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

Ein Segmentcode-Array von 0 bis F in hexadezimaler Darstellung (gemeinsame Kathode).

```
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)
```

Setzen Sie `ds` , `st_cp` , `sh_cp` auf drei Pins und den Ausgangszustand auf niedrigen Niveau.

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

Ordnen Sie die Datendaten SDI (DS) in Bits zu. Hier nehmen wir an, dass `dat = 0x3f` (0011 1111, wenn Bit = 2, 0x3f 2 Bits nach rechts (<<) verschiebt). `1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000` ist wahr.

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

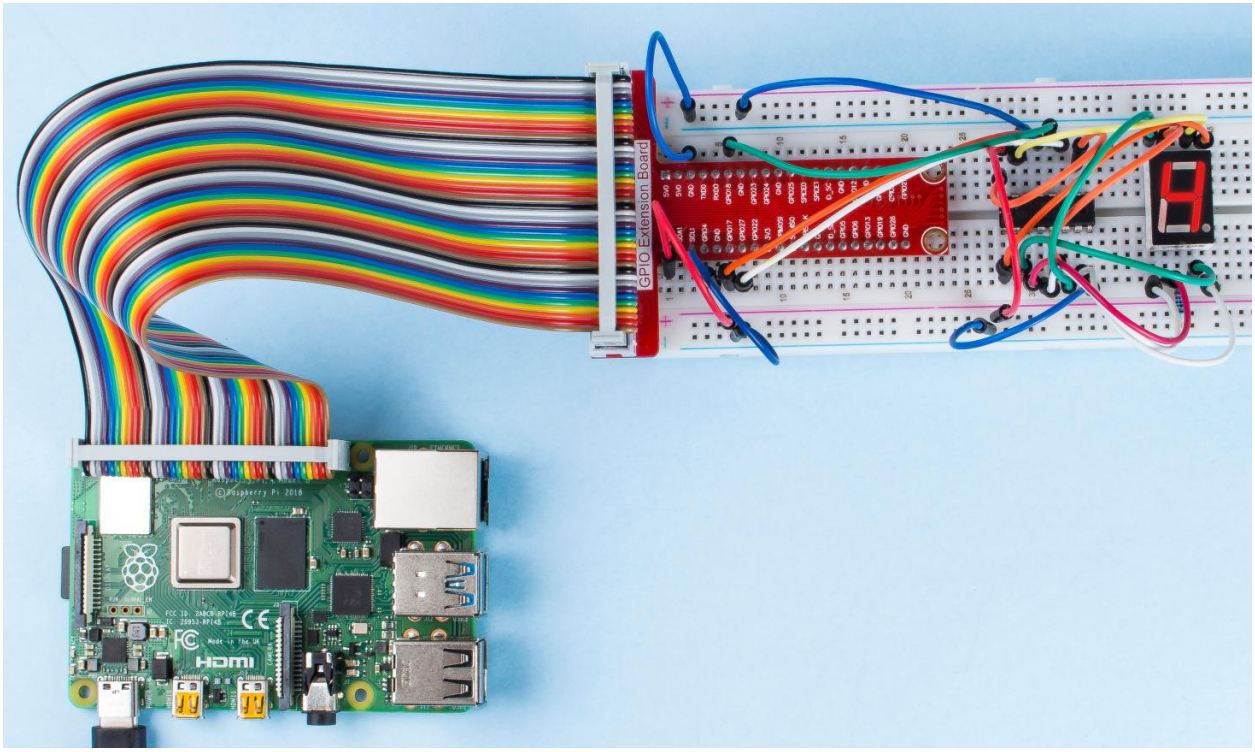
Der Anfangswert von SRCLK war auf NIEDRIG gesetzt, und hier wird er auf HIGH gesetzt, um einen ansteigenden Flankenimpuls zu erzeugen und dann das DS-Datum in das Schieberegister zu verschieben.

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

Der Anfangswert von RCLK wurde auf NIEDRIG gesetzt, und hier wird er auf HOCH gesetzt, um eine ansteigende Flanke zu erzeugen und dann Daten vom Schieberegister zum Speicherregister zu verschieben.

Bemerkung: Das hexadezimale Format der Nummer 0 bis 15 ist (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Phänomen Bild

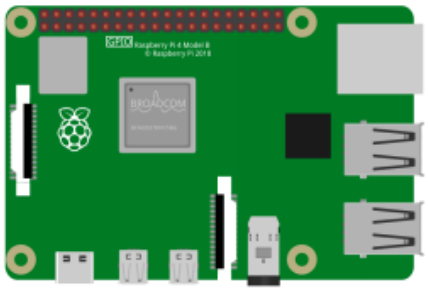
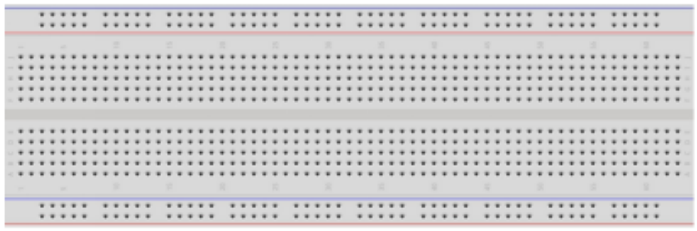


1.7.5 1.1.5 4-stellige 7-Segment-Anzeige

Einführung

Folgen Sie mir als Nächstes, um zu versuchen, die 4-stellige 7-Segment-Anzeige zu steuern.

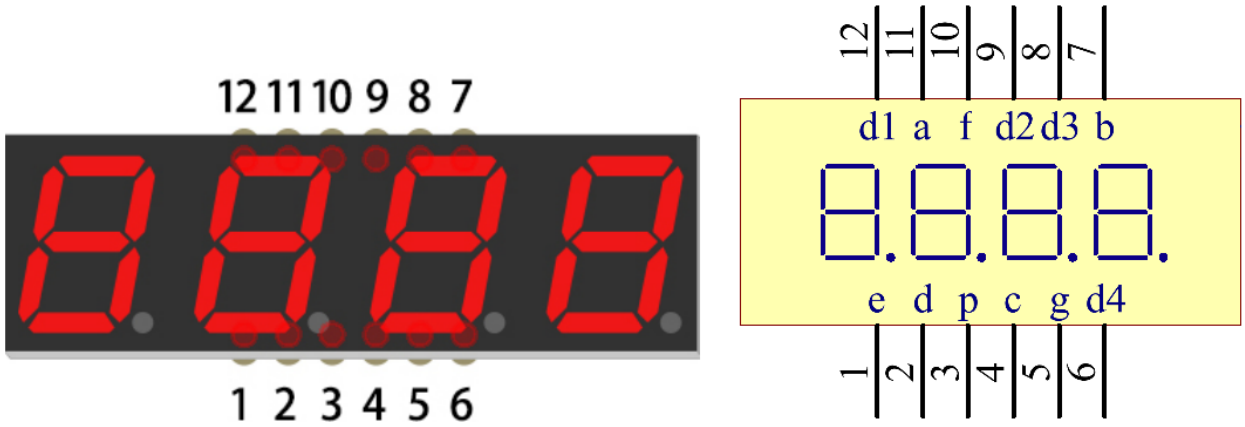
Komponenten

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

Prinzip

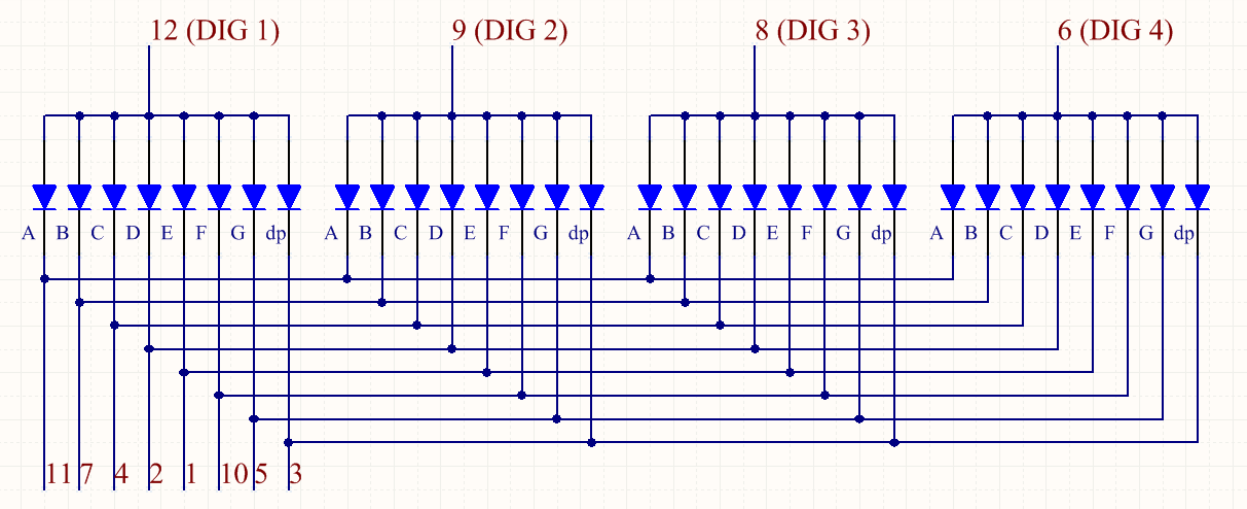
4-stellige 7-Segment-Anzeige

Die 4-stellige 7-Segment-Anzeige besteht aus vier 7-Segment-Anzeigen, die zusammenarbeiten.



Das 4-Digtal-7-Segment-Display arbeitet unabhängig. Es verwendet das Prinzip der menschlichen visuellen Persistenz, um die Zeichen jedes 7-Segments schnell in einer Schleife anzuzeigen und fortlaufende Zeichenfolgen zu bilden.

Wenn beispielsweise *1234* auf dem Display angezeigt wird, wird *1* auf dem ersten 7-Segment angezeigt und *234* wird nicht angezeigt. Nach einer gewissen Zeit zeigt das zweite 7-Segment *2* , das 1. 3. 4. des 7-Segments zeigt nicht usw. Die vier Digitalanzeigen werden nacheinander angezeigt. Dieser Vorgang ist sehr kurz (normalerweise 5 ms), und aufgrund des optischen Nachleuchteffekts und des Prinzips der visuellen Rückstände können wir vier Zeichen gleichzeitig sehen.



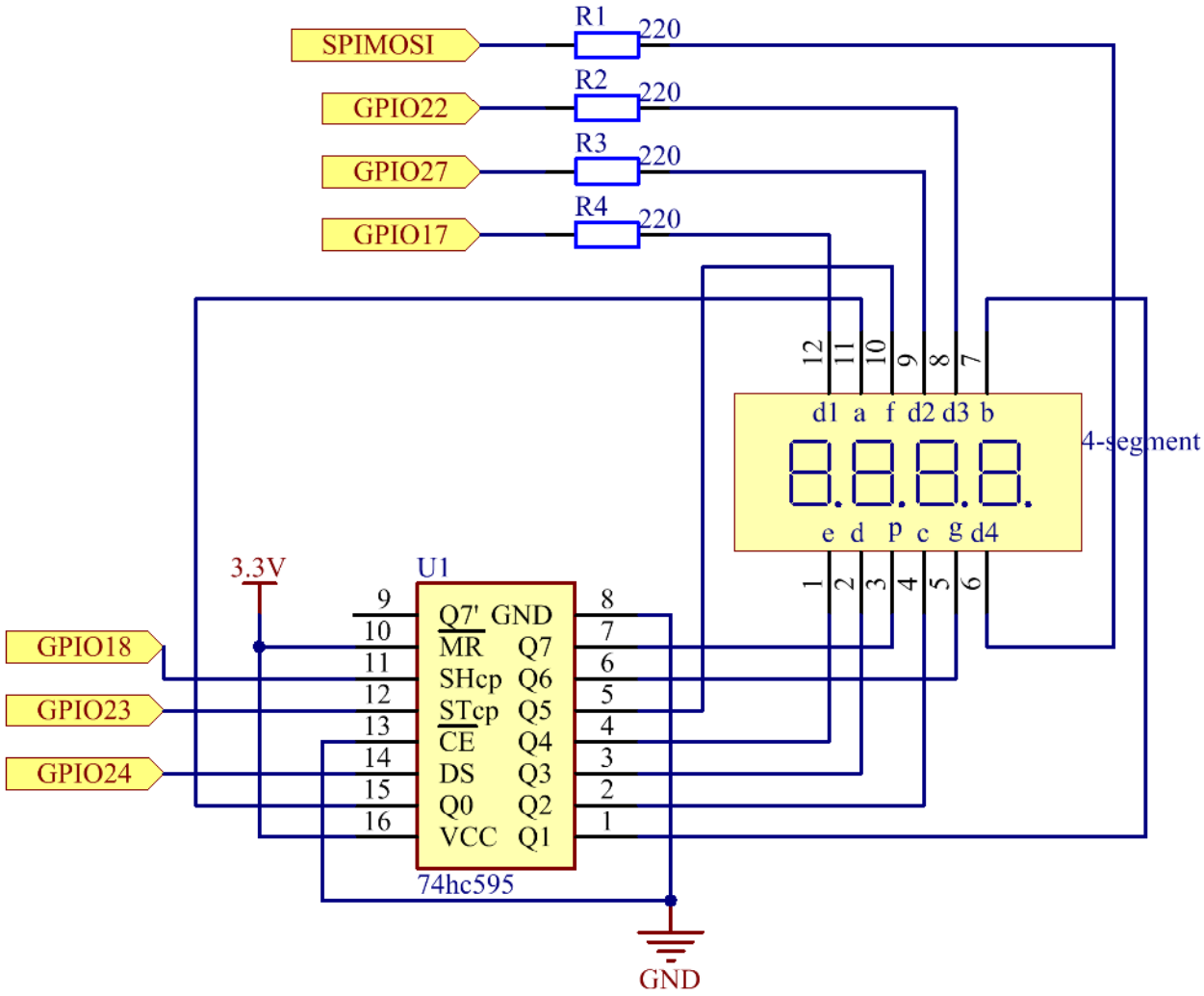
Koden anzeigen

Um Ihnen zu zeigen, wie 7-Segment-Anzeigen (Gemeisame Anode) Nummer anzeigen, haben wir die folgende Tabelle gezeichnet. Nummer sind die Nummer 0-F, die auf der 7-Segment-Anzeige angezeigt werden. (DP) GFEDCBA bezieht sich auf die entsprechende LED, die auf 0 oder 1 gesetzt ist. Beispielsweise bedeutet 11000000, dass DP und G auf 1 gesetzt sind, während andere auf 0 gesetzt sind. Daher wird die Nummer 0 auf dem 7-Segment-Display angezeigt, während der HEX-Kode der Hexadezimalzahl entspricht.

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			

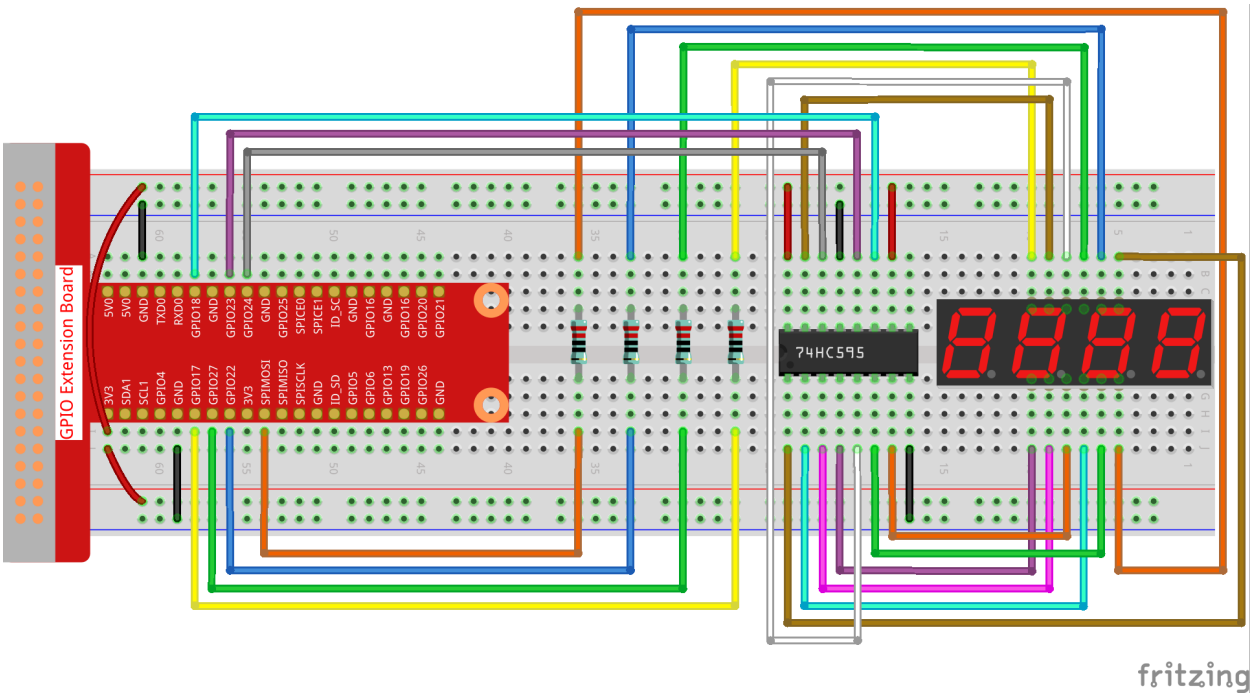
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem der Code ausgeführt wurde, nimmt das Programm eine Zählung vor, die um 1 pro Sekunde erhöht wird, und die 4-stellige 7-Segment-Anzeige zeigt die Zählung an.

Code

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

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
int counter = 0;

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

void hc595_shift(int8_t data)
{
    int i;
    for (i = 0; i < 8; i++)
    {
        digitalWrite(SDI, 0x80 & (data << i));
        digitalWrite(SRCLK, 1);
        delayMicroseconds(1);
        digitalWrite(SRCLK, 0);
    }
    digitalWrite(RCLK, 1);
    delayMicroseconds(1);
    digitalWrite(RCLK, 0);
}

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

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

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

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

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

Code Erklärung

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

Diese vier Pins steuern die gemeinsamen Anodenpins der vierstelligen 7-Segment-Anzeigen.

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

Ein Segmentkode-Array von 0 bis 9 in hexadezimaler Darstellung (gemeinsame Anode).

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

Wählen Sie den Ort des Wertes. Es gibt nur einen Ort, der jedes Mal aktiviert werden sollte. Der aktivierte Ort wird hoch geschrieben.

```
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]);
    }
}
```

Mit dieser Funktion wird die auf der 4-stelligen 7-Segment-Anzeige angezeigte Nummer eingestellt.

- `clearDisplay()` Schreiben Sie in 11111111, um diese acht LEDs auf der 7-Segment-Anzeige auszuschalten und den angezeigten Inhalt zu löschen.
- `pickDigit(0)` Wählen Sie die vierte 7-Segment-Anzeige.
- `hc595_shift(number[counter%10])` Die Nummer in der einzelnen Ziffer des Zählers wird im vierten Segment angezeigt.

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

Dies ist eine vom System bereitgestellte Funktion. Der Prototyp der Kode lautet:

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

Nach dem Ausführen des `signal()` hält der Prozess, sobald er das entsprechende Signal (in diesem Fall SIGALRM) erhalten hat, die vorhandene Aufgabe sofort an und verarbeitet die eingestellte Funktion (in diesem Fall `timer(sig)`).

```
alarm(1);
```

Dies ist auch eine vom System bereitgestellte Funktion. Der Kode-Prototyp ist:

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

Es erzeugt nach einer bestimmten Anzahl von Sekunden ein SIGALRM-Signal.

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

Wir verwenden die obigen Funktionen, um die Timer-Funktion zu implementieren. Nachdem der `alarm()` das SIGALRM-Signal erzeugt hat, wird die Timer-Funktion aufgerufen. Addiere 1 zum Zähler und die Funktion `alarm(1)` wird nach 1 Sekunde wiederholt aufgerufen.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, nimmt das Programm eine Zählung vor, die um 1 pro Sekunde erhöht wird, und die 4-stellige Anzeige zeigt die Zählung an.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

SDI = 24
RCLK = 23
SRCLK = 18

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

counter = 0
timer1 = 0

def clearDisplay():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

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


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


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

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

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

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


def setup():
    GPIO.setmode(GPIO.BCM)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

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

Code Erklärung

```
placePin = (10, 22, 27, 17)
```

Diese vier Pins steuern die gemeinsamen Anodenpins der vierstelligen 7-Segment-Anzeigen.

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

Ein Segmentkode-Array von 0 bis 9 in hexadezimaler Darstellung (gemeinsame Anode).

```
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)
```

Schreiben Sie achtmal 1 in SDI., Damit die acht LEDs auf dem 7-Segment-Anzeige erlöschen, um den angezeigten Inhalt zu löschen.

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

Wählen Sie den Ort des Wertes. Es gibt nur einen Ort, der jedes Mal aktiviert werden sollte. Der aktivierte Ort wird hoch geschrieben.

```
def loop():
    global counter
    while True:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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])
```

Mit dieser Funktion wird die auf der 4-stelligen 7-Segment-Anzeige angezeigte Nummer eingestellt.

Starten Sie zuerst die vierte Segmentanzeige und schreiben Sie die einstellige Nummer. Dann starten Sie die Anzeige des dritten Segments und geben Sie die Zehnerstelle ein. Starten Sie danach die zweite bzw. die erste Segmentanzeige und schreiben Sie die Hunderter- bzw. Tausenderstellen. Da die Aktualisierungsgeschwindigkeit sehr hoch ist, sehen wir eine vollständige vierstellige Anzeige.

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

Das Modul Threading ist das übliche Threading-Modul in Python Tim und Timer ist die Unterklasse davon. Der Prototyp der Kode ist:

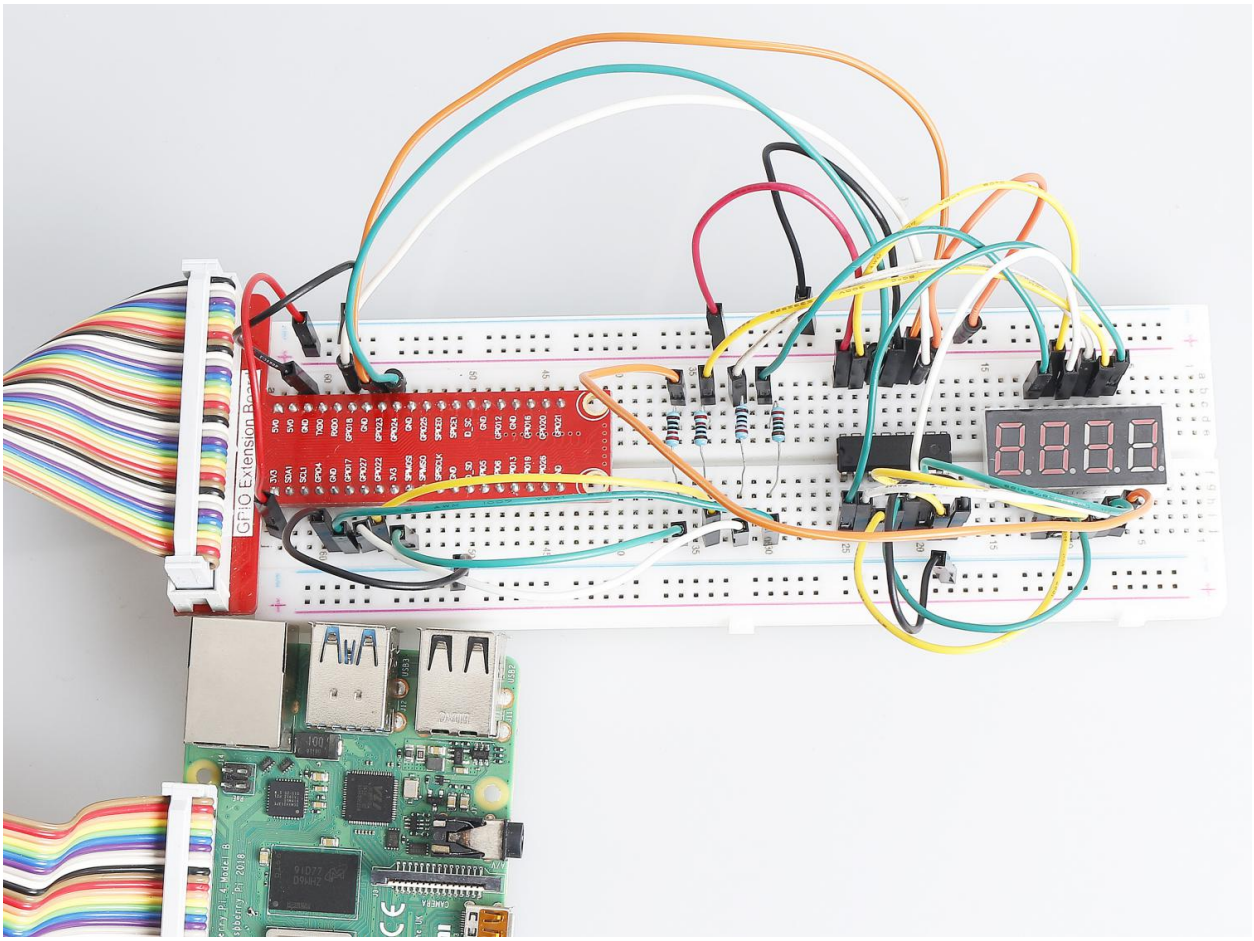
```
class threading.Timer(interval, function, args=[], kwargs={})
```

Nach dem Intervall wird die Funktion ausgeführt. Hier beträgt das Intervall 1.0 und die Funktion ist `timer()` . `start()` bedeutet, dass der Timer an diesem Punkt startet.

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

Nachdem der Timer 1,0 Sekunden erreicht hat, wird die Timer-Funktion aufgerufen. Addiere 1 zum Zähler und der Timer wird erneut verwendet, um sich jede Sekunde wiederholt auszuführen.

Phänomen Bild

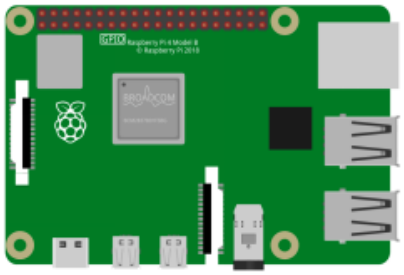
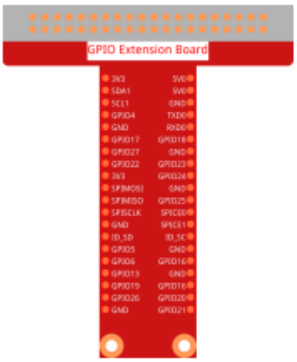
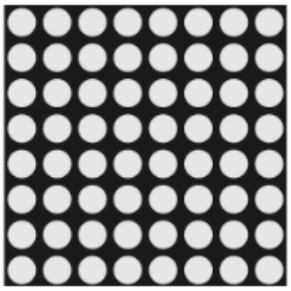
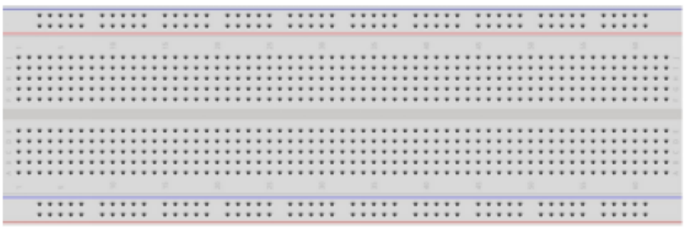


1.7.6 1.1.6 LED-Punktmatrix

Einführung

Wie der Name schon sagt, eine LED-Punktmatrix ist eine Matrix aus LEDs. Das Aufleuchten und Dimmen der LEDs formuliert unterschiedliche Zeichen und Muster.

Komponenten

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

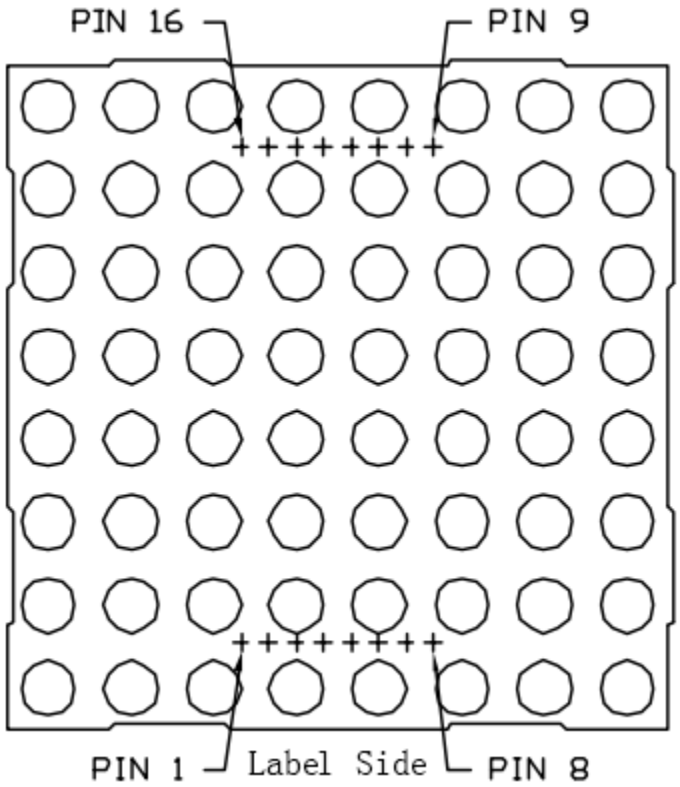
Prinzip

LED-Punktmatrix

Im Allgemeinen kann die LED-Punktmatrix in zwei Typen eingeteilt werden: gemeinsame Kathode (CC) und gemeinsame Anode (CA). Sie sehen sich sehr ähnlich, aber innerlich liegt der Unterschied. Sie können durch Test erkennen. In diesem Kit wird eine CA verwendet. Sie können 788BS an der Seite beschriftet sehen.

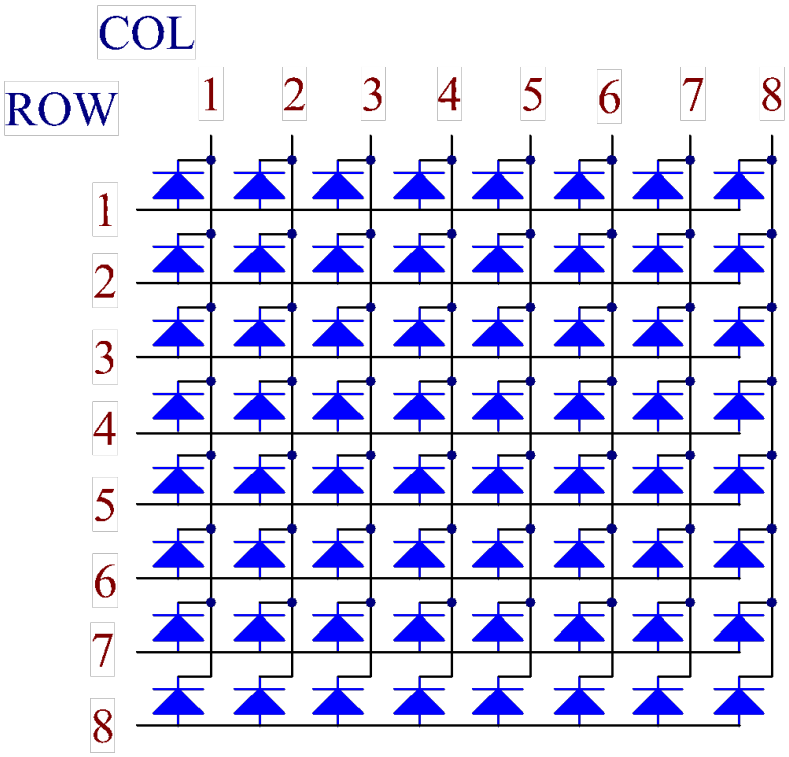
Siehe die Abbildung unten. Die Pins sind an den beiden Enden hinten angeordnet. Nehmen Sie die Etikettenseite als Referenz: Die Pins an diesem Ende sind Pin 1-8 und die anderen sind Pin 9-16.

Die Außenansicht:



Unten zeigen die Abbildungen ihre interne Struktur. Sie können in einer CA-LED-Punktmatrix sehen, dass ROW die Anode der LED darstellt und COL die Kathode ist. es ist das Gegenteil für einen CC. Eines ist gemeinsam: Für beide Typen sind Pin 13, 3, 4, 10, 6, 11, 15 und 16 alle COL, wenn Pin 9, 14, 8, 12, 1, 7, 2 und 5 alle COL sind REIHE. Wenn Sie die erste LED in der oberen linken Ecke einschalten möchten, setzen Sie für eine CA-LED-Punktmatrix einfach Pin 9 als High und Pin 13 als Low und für eine CC-Pin Pin 13 als High und Pin 9 als NIEDRIG . Wenn Sie die gesamte erste Spalte für CA aufleuchten möchten, setzen Sie Pin 13 auf Niedrig und REIHE 9, 14, 8, 12, 1, 7, 2 und 5 auf Hoch. Wenn Sie für CC Pin 13 auf Hoch und setzen REIHE 9, 14, 8, 12, 1, 7, 2 und 5 als niedrig. Betrachten Sie die folgenden Abbildungen zum besseren Verständnis.

Die interne Ansicht:



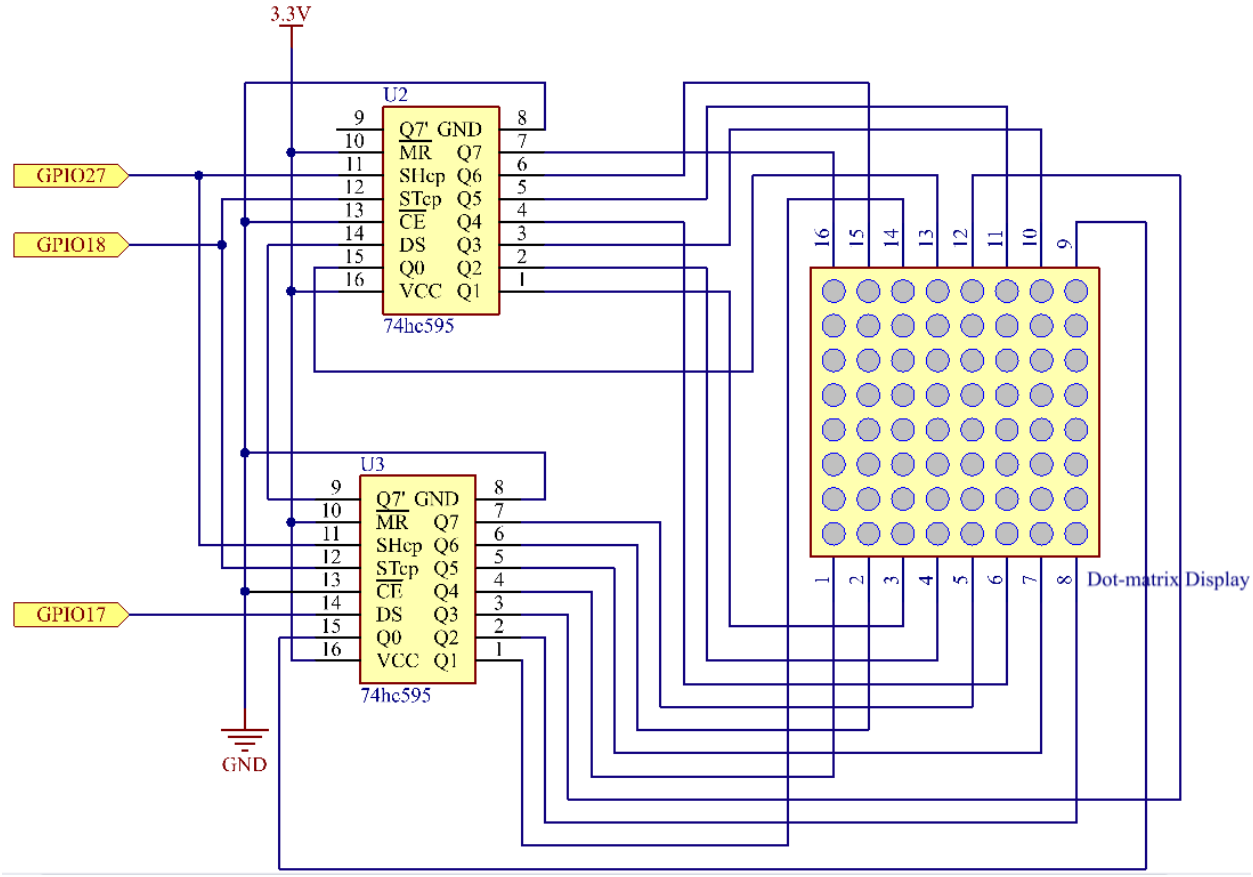
Pin-Nummerierung entsprechend den obigen Zeilen und Spalten:

MIT DEM	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin Nr.	**13**	**3**	**4**	**10**	**6**	**11**	**15**	**16**
REIHE	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin Nr.	**9**	**14**	**8**	**12**	**1**	**7**	**2**	**5**

Zusätzlich werden hier zwei 74HC595-Chips verwendet. Eine besteht darin, die Zeilen der LED-Punktmatrix zu steuern, während die andere die Spalten steuert.

Schematische Darstellung

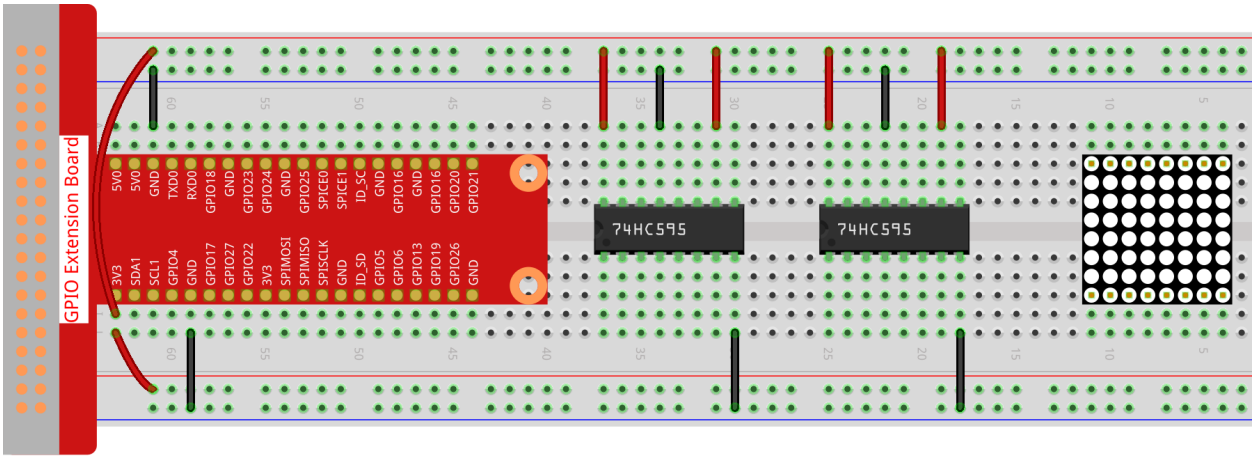
T-Karte Name	physisch	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



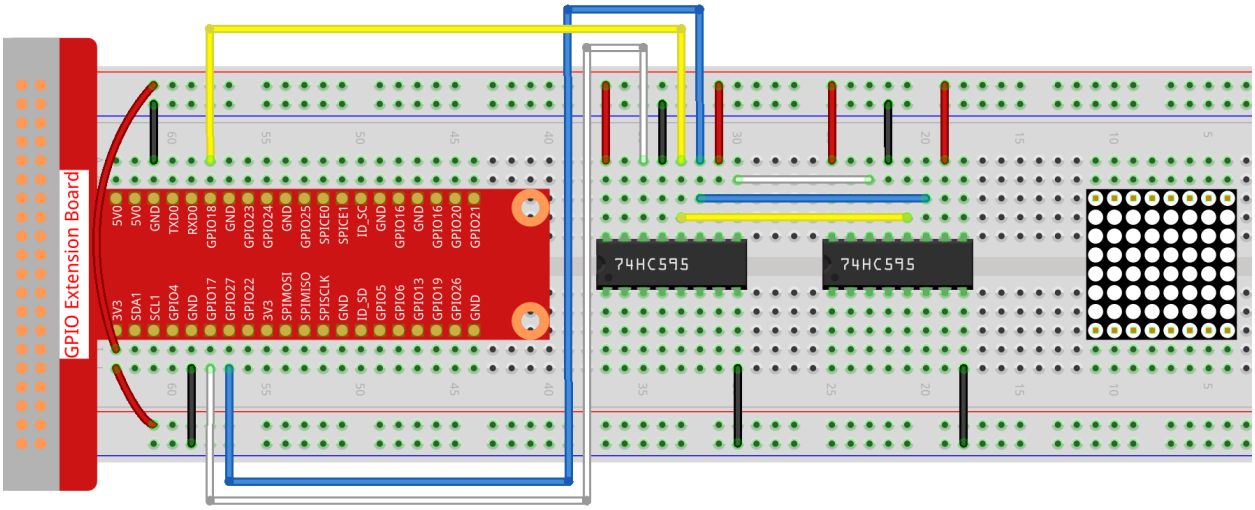
Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf. Da die Verkabelung kompliziert ist, machen wir es Schritt für Schritt. Setzen Sie zuerst den T-Cobbler, die LED-Punktmatrix und zwei 74HC595-Chips in das Steckbrett ein. Verbinden Sie die 3, 3 V und GND des T-Cobbler mit den Löchern auf den beiden Seiten der Platine und schließen Sie dann Pin 16 und 10 der beiden 74HC595-Chips an VCC, Pin 13 und Pin 8 an GND an.

Bemerkung: Im Fritzing-Bild oben befindet sich die Seite mit der Beschriftung unten.

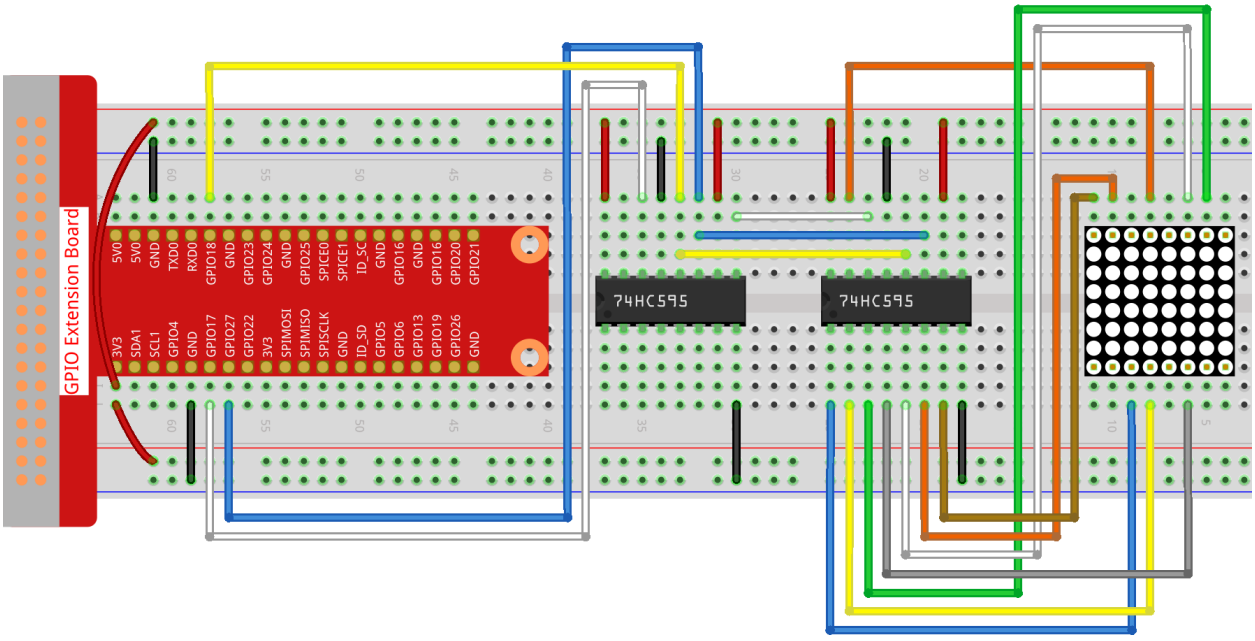


Schritt 2: Verbinden Sie Pin 11 der beiden 74HC595 miteinander und dann mit GPIO27. dann Pin 12 der beiden Chips und GPIO18; Als nächstes Pin 14 des 74HC595 auf der linken Seite an GPIO17 und Pin 9 an Pin 14 des zweiten 74HC595.



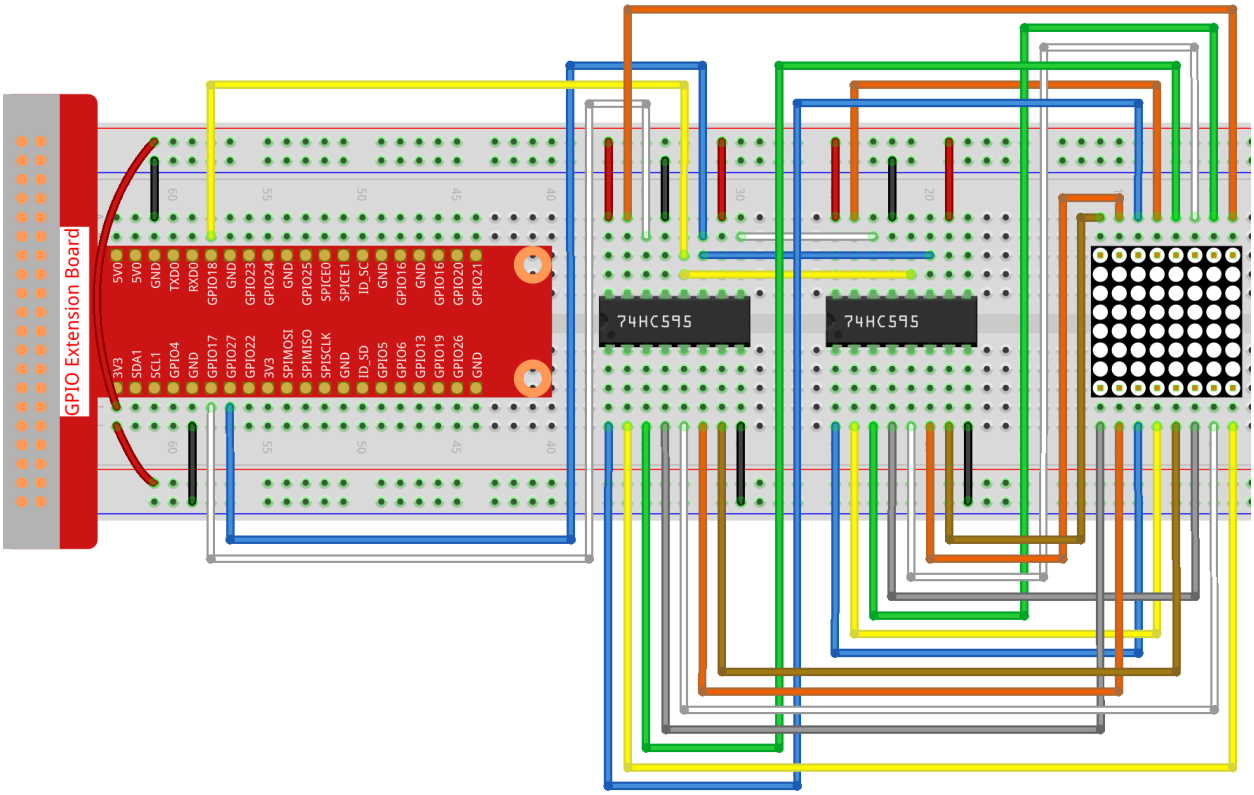
Schritt 3: Der 74HC595 auf der rechten Seite steuert die Spalten der LED-Punktmatrix. Die Zuordnung finden Sie in der folgenden Tabelle. Daher werden die Q0-Q7-Pins des 74HC595 mit Pin 13, 3, 4, 10, 6, 11, 15 bzw. 16 abgebildet.

74HC595	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
LED-Punktmatrix	**13**	**3**	**4**	**10**	**6**	**11**	**15**	**16**



Schritt 4: Verbinden Sie nun die REIHEN der LED-Punktmatrix. Der 74HC595 auf der linken Seite steuert die REIHE der LED-Punktmatrix. Die Zuordnung finden Sie in der folgenden Tabelle. Wir können sehen, dass Q0-Q7 des 74HC595 auf der linken Seite mit Pin 9, 14, 8, 12, 1, 7, 2 bzw. 5 abgebildet sind.

74HC595	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
LED-Punktmatrix	**9**	**14**	**8**	**12**	**1**	**7**	**2**	**5**



Für Benutzer in C-Sprache

Schritt 5: Wechseln Sie in den Codeordner.

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

Schritt 6: Kompilieren.

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

Schritt 7: Ausführen.

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

Nachdem die Kode ausgeführt wurde, leuchtet die LED-Punktmatrix Zeile für Zeile und Spalte für Spalte auf und aus.

Code

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

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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;
   }
   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);
      }
   }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    return 0;
}
```

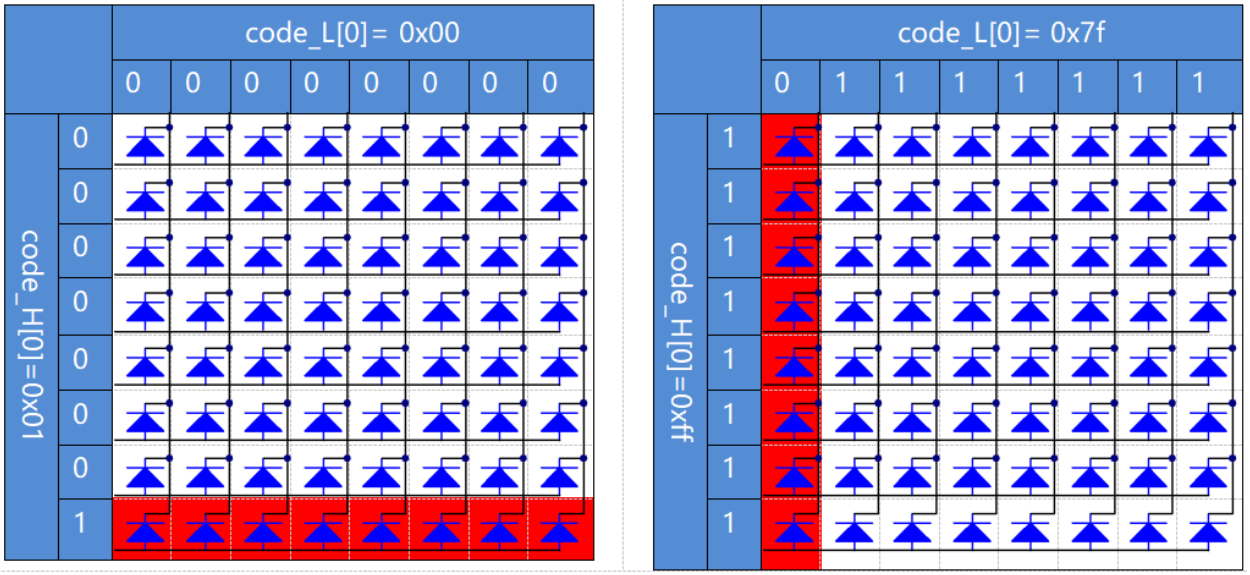
Code Erklärung

```
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};
```

Der Array-Kode_H repräsentiert die Elemente der LED-Punktmatrixzeile, und der Array-Code_L bezieht sich auf die Elemente der Spalte. Wenn Zeichen angezeigt werden, werden ein Element in Zeile und eines in Spalte erfasst und den beiden HC595-Chips zugewiesen. Somit ist ein Muster auf der LED-Punktmatrix gezeigt. Nehmen Sie als Beispiele die erste Nummer von code_H, 0x01 und die erste Nummer von code_L, 0x00.

0x01 konvertiert in Binär wird 00000001; 0x00 konvertiert in Binär wird 0000 0000.

In diesem Kit wird eine LED-Punktmatrixanzeige mit gemeinsamer Anode verwendet, sodass nur die acht LEDs in der achten Reihe aufleuchten. Wenn die Bedingungen, dass Code H 0xff und Code_L 0x7f ist, gleichzeitig erfüllt sind, leuchten diese 8 LEDs in der ersten Spalte.




```
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);
```

Schreiben Sie den Wert von dat bitweise auf den Pin SDI des HC595. Der Anfangswert von SRCLK wurde auf 0 gesetzt, und hier ist er auf 1 gesetzt, was einen ansteigenden Flankenimpuls erzeugen und dann das pinSDI(DS)-Datum in das Schieberegister verschieben soll.

```
void hc595_out(){
    digitalWrite(RCLK, 1);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    delay(1);
    digitalWrite(RCLK, 0);
```

Der Anfangswert von RCLK wurde auf 0 gesetzt, und hier wird er auf 1 gesetzt, um eine ansteigende Flanke zu erzeugen und dann Daten vom Schieberegister zum Speicherregister zu verschieben.

```
while(1){
    for(i=0;i<sizeof(code_H);i++){
        hc595_in(code_L[i]);
        hc595_in(code_H[i]);
        hc595_out();
        delay(100);
    }
}
```

In dieser Schleife werden diese 20 Elemente in den beiden Arrays Kode_L und code_H nacheinander auf die beiden 74HC595-Chips hochgeladen. Rufen Sie dann die Funktion hc595_out () auf, um Daten vom Schieberegister zum Speicherregister zu verschieben.

Für Python-Sprachbenutzer

Schritt 5: Gehen Sie in den Kode-Ordner.

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

Schritt 6: Ausführen.

```
sudo python3 1.1.6_LedMatrix.py
```

Nachdem die Kode ausgeführt wurde, leuchtet die LED-Punktmatrix Zeile für Zeile und Spalte für Spalte auf und aus.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

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

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

def destroy():
    GPIO.cleanup()

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

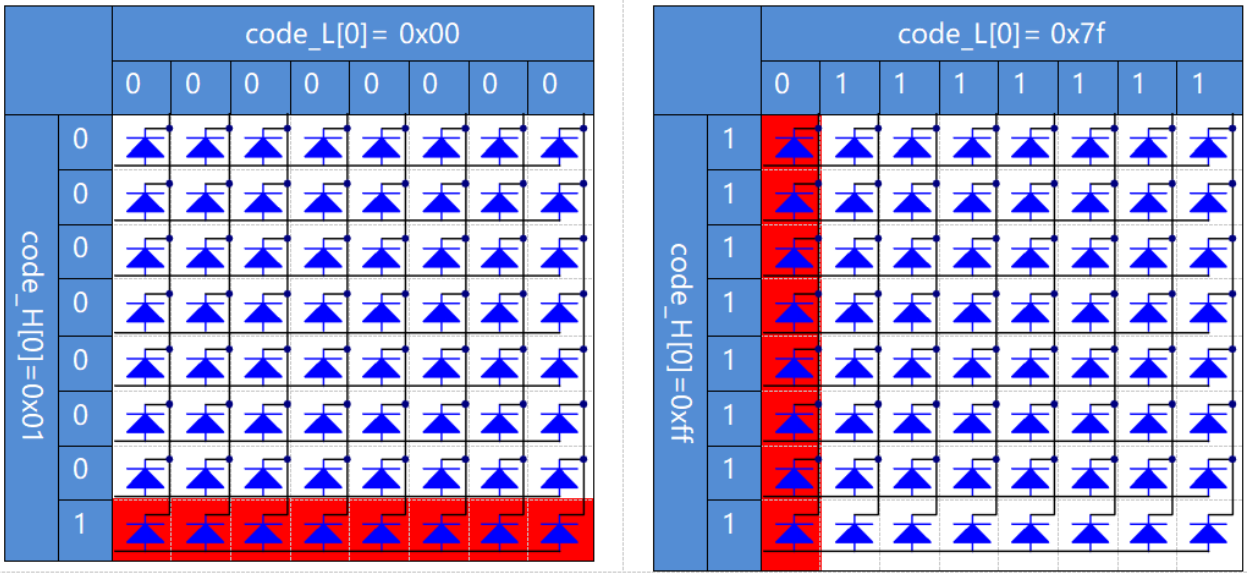
Code Erklärung

```
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]
```

Das Array code_H repräsentiert die Elemente der Matix-Zeile, und das Array code_L bezieht sich auf die Elemente der Spalte. Wenn Zeichen angezeigt werden, werden ein Element in Zeile und eines in Spalte erfasst und den beiden HC595-Chips zugewiesen. Somit ist ein Muster auf der LED-Punktmatrix gezeigt. Nehmen Sie als Beispiele die erste Nummer von code_H, 0x01 und die erste Nummer von code_L, 0x00.

0x01 konvertiert in Binär wird 00000001; 0x00 konvertiert in Binär wird 0000 0000.

In diesem Kit wird eine gemeinsame Anoden-LED-Punktmatrix angewendet, sodass nur die acht LEDs in der achten Reihe aufleuchten. Wenn die Bedingungen, dass Code H 0xff und Code_L 0x7f ist, gleichzeitig erfüllt sind, leuchten diese 8 LEDs in der ersten Spalte.

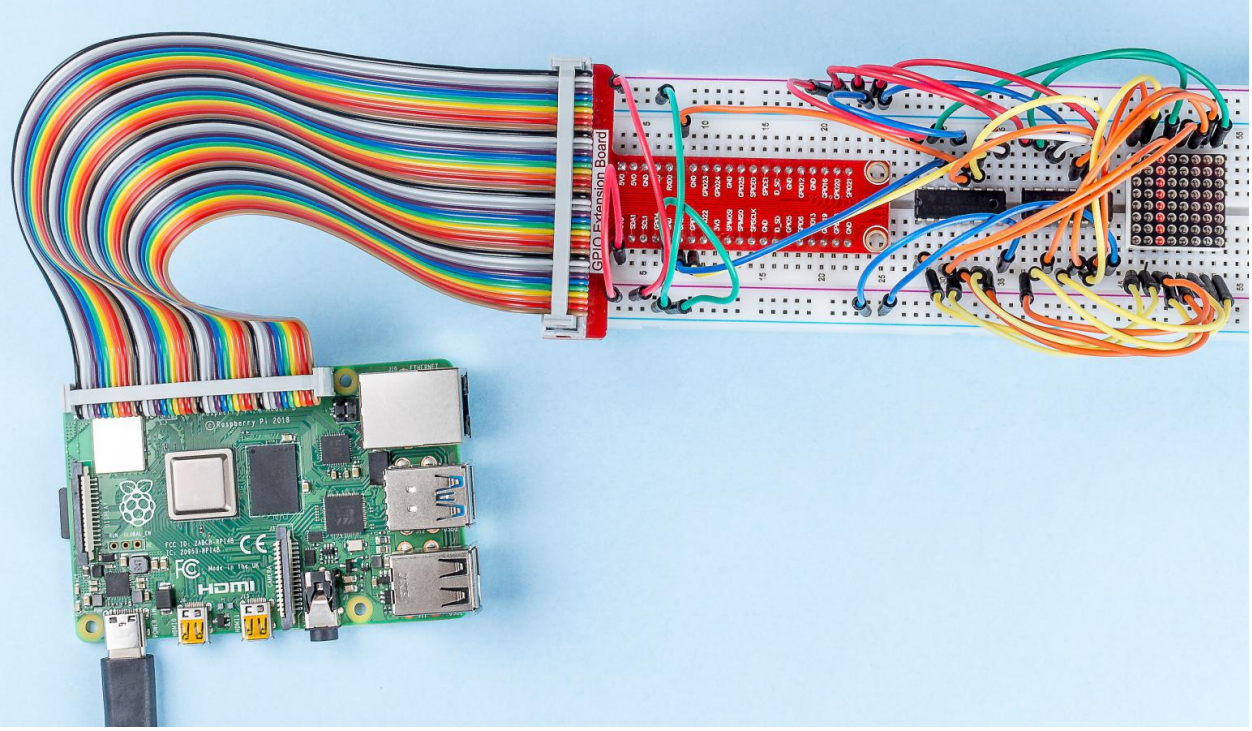



```
for i in range(0, len(code_H)):
   hc595_shift(code_L[i])
   hc595_shift(code_H[i])
```

In dieser Schleife werden diese 20 Elemente in den beiden Arrays Kode_L und Kode_H nacheinander auf den HC595-Chip hochgeladen.

Bemerkung: Wenn Sie Zeichen in der LED-Punktmatrix anzeigen möchten, lesen Sie bitte einen Python-Code: https://github.com/sunfounder/SunFounder_Dot_Matrix.

Phänomen Bild

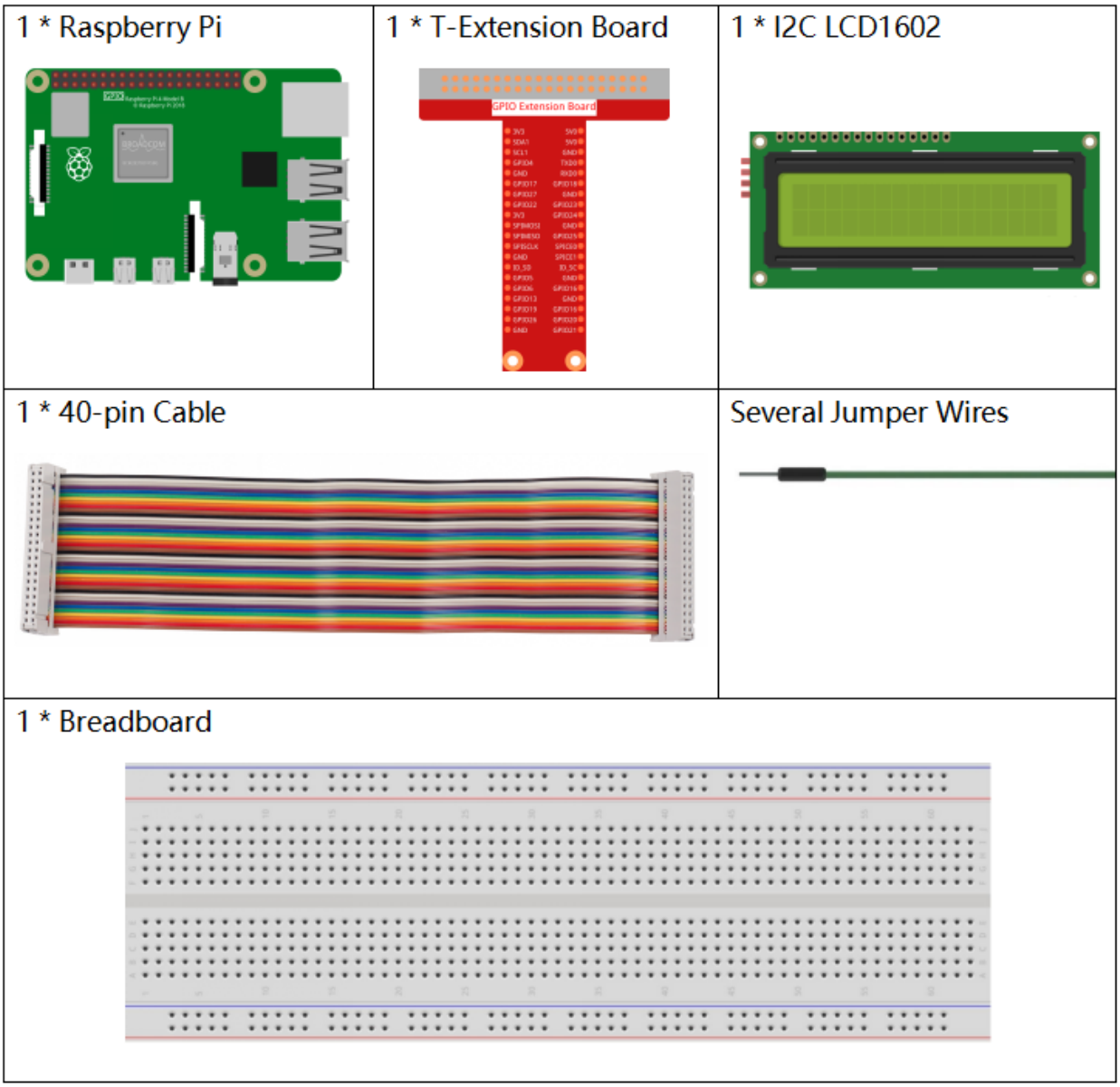


1.7.7 1.1.7 I2C LCD1602

Einführung

LCD1602 ist eine Flüssigkristallanzeige vom Zeichentyp, die gleichzeitig 32 (16 * 2) Zeichen anzeigen kann.

Komponenten



Prinzip

I2C LCD1602

Wie wir alle wissen, bereichern LCD und einige andere Anzeiger die Mensch-Maschine-Interaktion erheblich, weisen jedoch eine gemeinsame Schwäche auf. Wenn sie mit einem Controller verbunden sind, werden mehrere E / A des Controllers belegt, der nicht so viele äußere Ports hat. Es schränkt auch andere Funktionen der Steuerung ein. Daher wurde LCD1602 mit einem I2C-Bus entwickelt, um das Problem zu lösen.

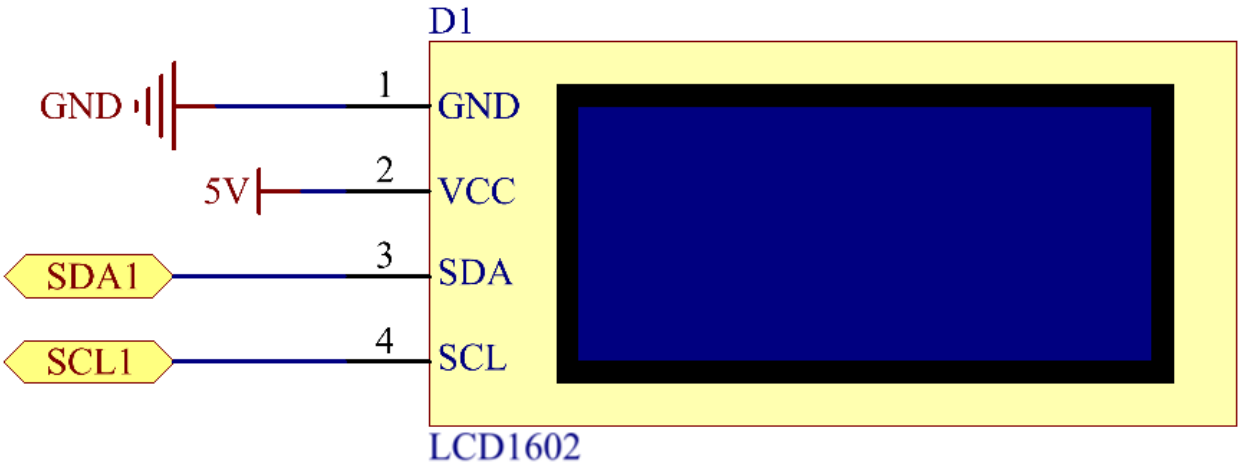
I2C-Kommunikation

Der I2C-Bus (Inter-Integrated Circuit) ist ein sehr beliebter und leistungsstarker Bus für die Kommunikation zwischen einem Master-Gerät (oder Master-Geräten) und einem oder mehreren Slave-Geräten.

Der I2C-Hauptcontroller kann zur Steuerung des E / A-Expanders, verschiedener Sensoren, des EEPROM, des ADC / DAC usw. verwendet werden. Alle diese werden nur von den beiden Pins des Hosts gesteuert, der seriellen Datenleitung (SDA1) und der seriellen Taktleitung (SCL1).

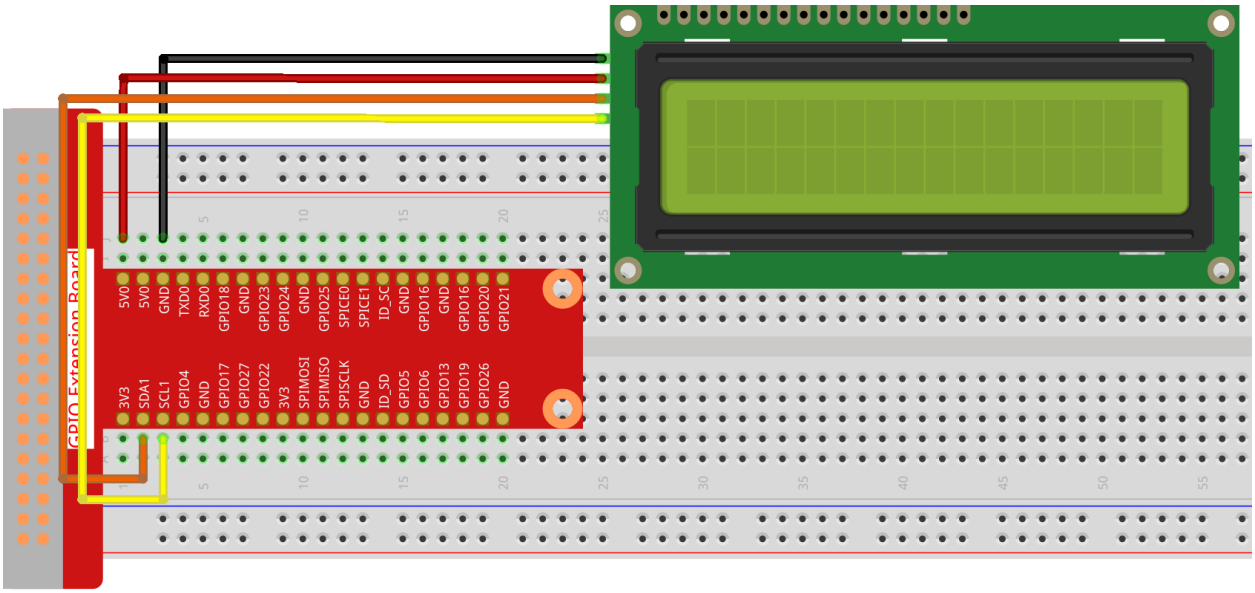
Schematische Darstellung

T-Karte Name	physisch
SDA1	Pin 3
SCL1	Pin 5



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: I2C einrichten (siehe *I2C-Konfiguration*. Wenn Sie I2C eingestellt haben, überspringen Sie diesen Schritt.)

Für Benutzer in C-Sprache

Schritt 3: Verzeichnis wechseln.

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

Schritt 4: Kompilieren.

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

Schritt 5: Ausführen.

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

Nachdem die Kode ausgeführt wurde, werden auf dem LCD "Greetings" und "From SunFounder" angezeigt.

Bemerkung: Wenn es kein Problem mit dem Code und der Verkabelung gibt, aber das LCD immer noch keinen Inhalt anzeigt, können Sie versuchen, das Potentiometer auf der Rückseite zu drehen.

Code

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

int LCDAddr = 0x27;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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;
    // 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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
    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");
}
```

Code Erklärung

```
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[]){......}
```

Diese Funktionen werden zur Steuerung des Open Source-Codes I2C LCD1602 verwendet. Sie ermöglichen die einfache Verwendung von I2C LCD1602. Unter diesen Funktionen wird `init()` zur Initialisierung verwendet, `clear()` wird zum Löschen des Bildschirms verwendet, `write()` wirdzum Schreiben der angezeigten Elemente verwendet und

andere Funktionen unterstützen die obigen Funktionen.

```
fd = wiringPiI2CSetup(LCDAddr);
```

Diese Funktion initialisiert das I2C-System mit dem angegebenen Gerätesymbol. Der Prototyp der Funktion:

```
int wiringPiI2CSetup(int devId);
```

Parameter devId ist die Adresse des I2C-Geräts. Sie kann über den Befehl i2cdetect (siehe Anhang) ermittelt werden. Die devId des I2C LCD1602 ist im Allgemeinen 0x27.

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

In dieser Funktion ist `data[]` das Zeichen, das auf dem LCD gedruckt werden soll, und die Parameter x und y bestimmen die Druckposition (Zeile y + 1, Spalte x + 1 ist die Startposition des zu druckenden Zeichens).

Für Python-Sprachbenutzer

Schritt 3: Verzeichnis wechseln.

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

Schritt 4: Ausführen.

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

Nachdem die Kode ausgeführt wurde, werden auf dem LCD "Greetings" und "From SunFounder" angezeigt.

- **Wenn Sie den Fehler `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` erhalten, müssen Sie sich auf *I2C-Konfiguration* beziehen, um I2C zu aktivieren.**
  - Wenn Sie den Fehler `ModuleNotFoundError: No module named 'smbus2'` erhalten, führen Sie bitte `sudo pip3 install smbus2` aus.
  - Wenn der Fehler `OSError: [Errno 121] Remote I/O` erscheint, bedeutet dies, dass das Modul falsch verdrahtet oder das Modul defekt ist.
  - Wenn es kein Problem mit dem Code und der Verkabelung gibt, aber das LCD immer noch keinen Inhalt anzeigt, können Sie versuchen, das Potentiometer auf der Rückseite zu drehen.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
def destroy():
    LCD1602.clear()

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

Code Erklärung

```
import LCD1602
```

Diese Datei ist eine Open Source-Datei zur Steuerung von I2C LCD1602. Es ermöglicht uns die einfache Verwendung von I2C LCD1602.

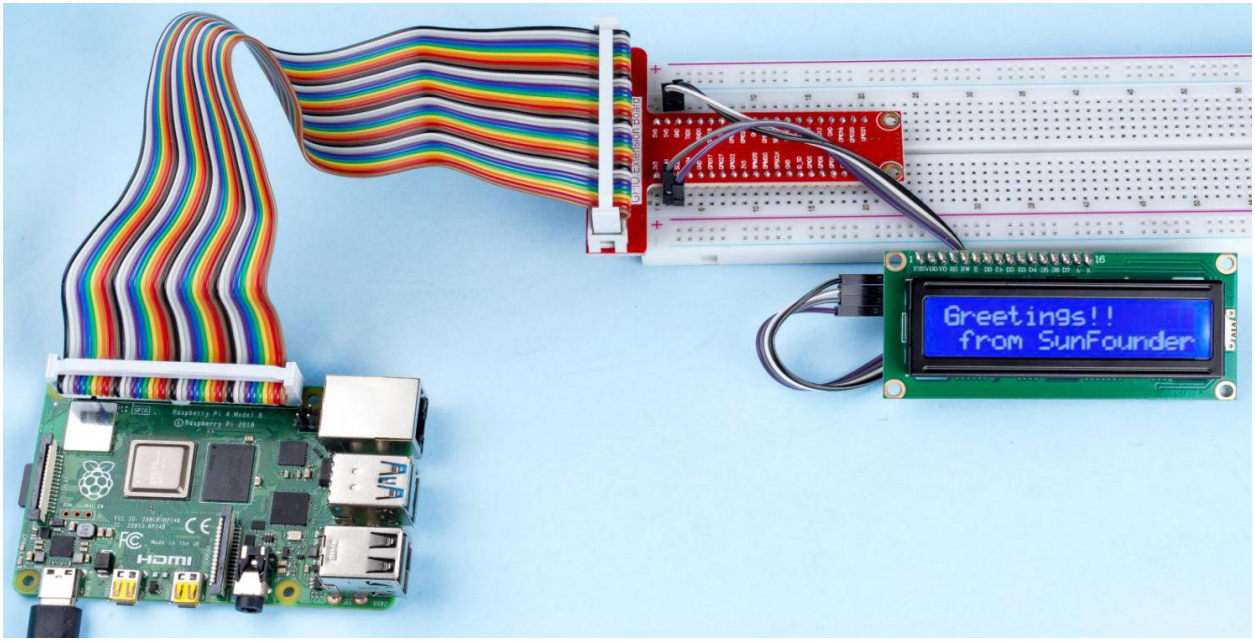
```
LCD1602.init(0x27, 1)
```

Die Funktion initialisiert das I2C-System mit dem angegebenen Gerätesymbol. Der erste Parameter ist die Adresse des I2C-Geräts, die mit dem Befehl i2cdetect erkannt werden kann (Einzelheiten siehe Anhang). Die Adresse des I2C LCD1602 lautet im Allgemeinen 0x27.

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

Innerhalb dieser Funktion 'Greetings!!' ist das Zeichen, das in der Zeile 0 + 1, Spalte 0 + 1 auf dem LCD gedruckt werden soll. Jetzt können Sie "Greetings! From SunFounder" auf dem LCD angezeigt.

Phänomen Bild



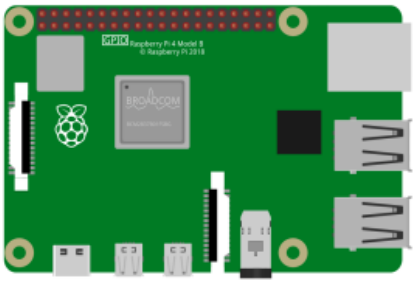
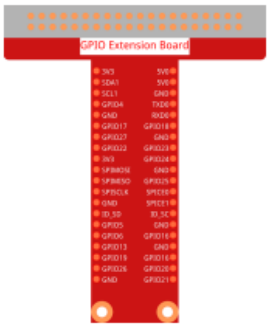
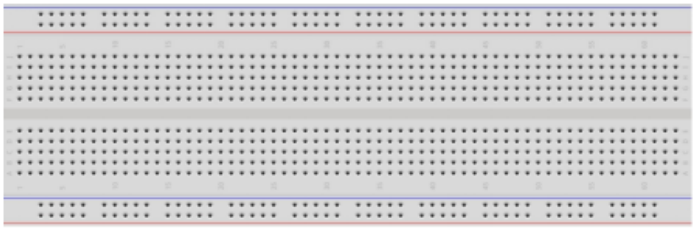
1.2 Ton

1.7.8 1.2.1 Aktiver Summer

Einführung

In dieser Lektion lernen wir, wie man einen aktiven Summer ansteuert, um mit einem PNP-Transistor zu piepen.

Komponenten

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Active Buzzer
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(1kΩ)
 </td></tr>
<tr><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * S8550 PNP Transistor
 </td></tr>
</table>

Prinzip

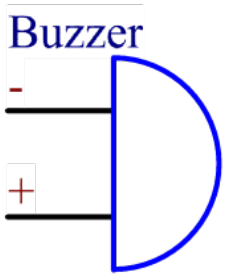
Summer

Als eine Art elektronischer Summer mit integrierter Struktur werden Summer, die mit Gleichstrom versorgt werden, häufig in Computern, Druckern, Fotokopierern, Alarmen, elektronischem Spielzeug, elektronischen Kraftfahrzeugen, Telefonen, Zeitschaltuhren und anderen elektronischen Produkten oder Sprachgeräten verwendet. Summer können in aktive und passive unterteilt werden (siehe folgendes Bild). Drehen Sie den Summer so, dass seine Stifte nach oben zeigen, und der Summer mit einer grünen Leiterplatte ist ein passiver Summer, während der mit einem schwarzen Band umschlossene ein aktiver ist.

Der Unterschied zwischen einem aktiven und einem passiven Summer:

Der Unterschied zwischen einem aktiven und einem passiven Summer besteht darin, dass ein aktiver Summer über eine integrierte Oszillationsquelle verfügt, sodass er bei Elektrifizierung Geräusche erzeugt. Ein passiver Summer verfügt jedoch nicht über eine solche Quelle, sodass bei Verwendung von Gleichstromsignalen kein Piepton ertönt. Stattdessen müssen Sie Rechteckwellen verwenden, deren Frequenz zwischen 2K und 5K liegt, um sie anzutreiben. Der aktive Summer ist aufgrund mehrerer eingebauter Schwingkreise oft teurer als der passive.

Das Folgende ist das elektrische Symbol eines Summers. Es hat zwei Stifte mit positiven und negativen Polen. Mit einem + in der Oberfläche steht die Anode und das andere ist die Kathode.

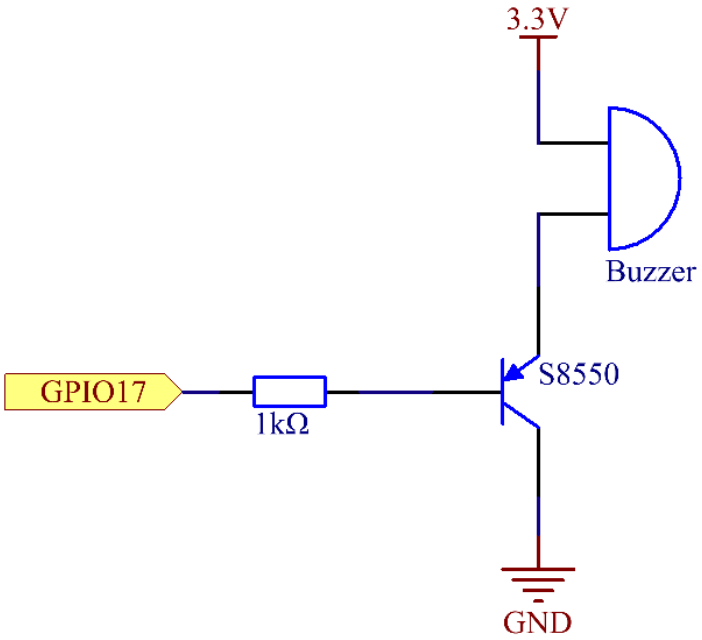


Sie können die Pins des Summers überprüfen, je länger die Anode und je kürzer die Kathode ist. Bitte verwechseln Sie sie beim Anschließen nicht, da sonst kein Summer ertönt.

Schematische Darstellung

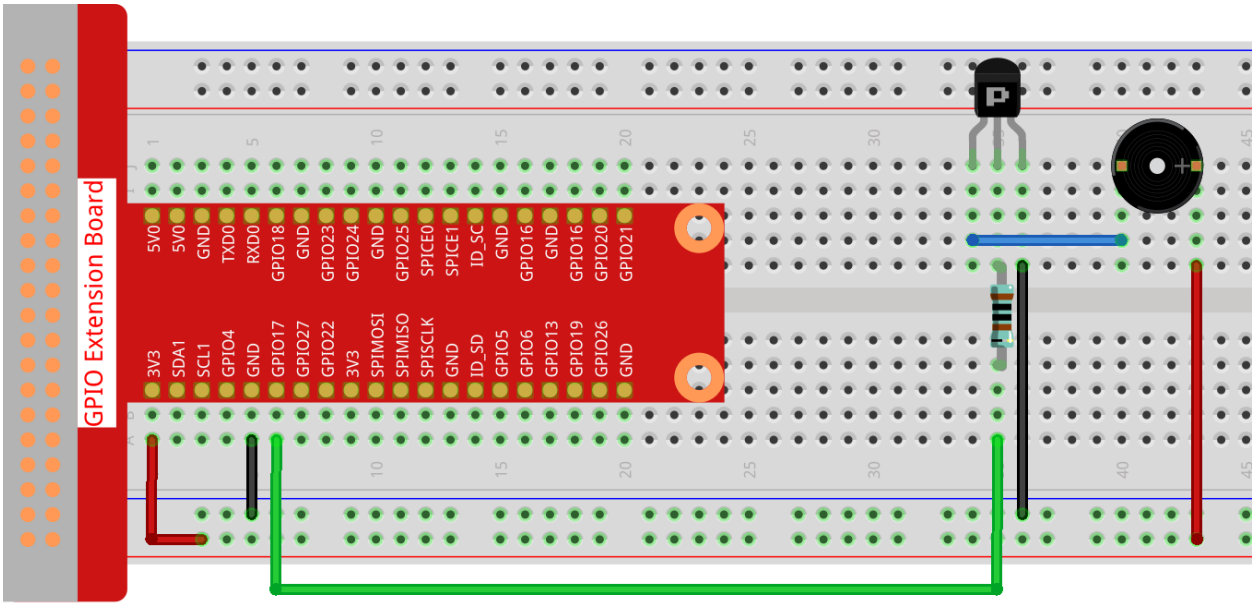
In diesem Experiment werden ein aktiver Summer, ein PNP-Transistor und ein 1k-Widerstand zwischen der Basis des Transistors und GPIO verwendet, um den Transistor zu schützen. Wenn der GPIO17 des Raspberry Pi-Ausgangs durch Programmierung mit einem niedrigen Niveau (0V) versorgt wird, leitet der Transistor aufgrund der Stromsättigung und der Summer gibt Geräusche aus. Wenn jedoch die IO des Raspberry Pi mit einem hohen Niveau versorgt wird, wird der Transistor abgeschaltet und der Summer gibt keine Geräusche von sich.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf. (Achten Sie auf die Pole des Summers: Der mit dem + Etikett ist der positive Pol und der andere der negative.)



Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Codedatei.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Die Kode läuft, der Summer piept.

Code

```
#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;
}
```

Code Erklärung

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

In diesem Experiment verwenden wir einen aktiven Summer, der beim Anschließen an den Gleichstrom automatisch einen Ton erzeugt. Diese Skizze dient dazu, den I/O -Port auf einen niedrigen Niveau (0V) einzustellen, um so den Transistor zu verwalten und den Summer piepen zu lassen.

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

Um den I/O -Anschluss auf einen hohen Niveau (3,3V) einzustellen, wird der Transistor nicht erregt und der Summer piept nicht.

Für Python-Sprachbenutzer

Schritt 2: Öffnen Sie die Codedatei.

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

Schritt 3: Ausführen.

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

Die Kode läuft, der Summer piept.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)

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()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

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

Stellen Sie den Summer Pin auf einen niedrigen Niveau ein, damit der Summer piept.

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

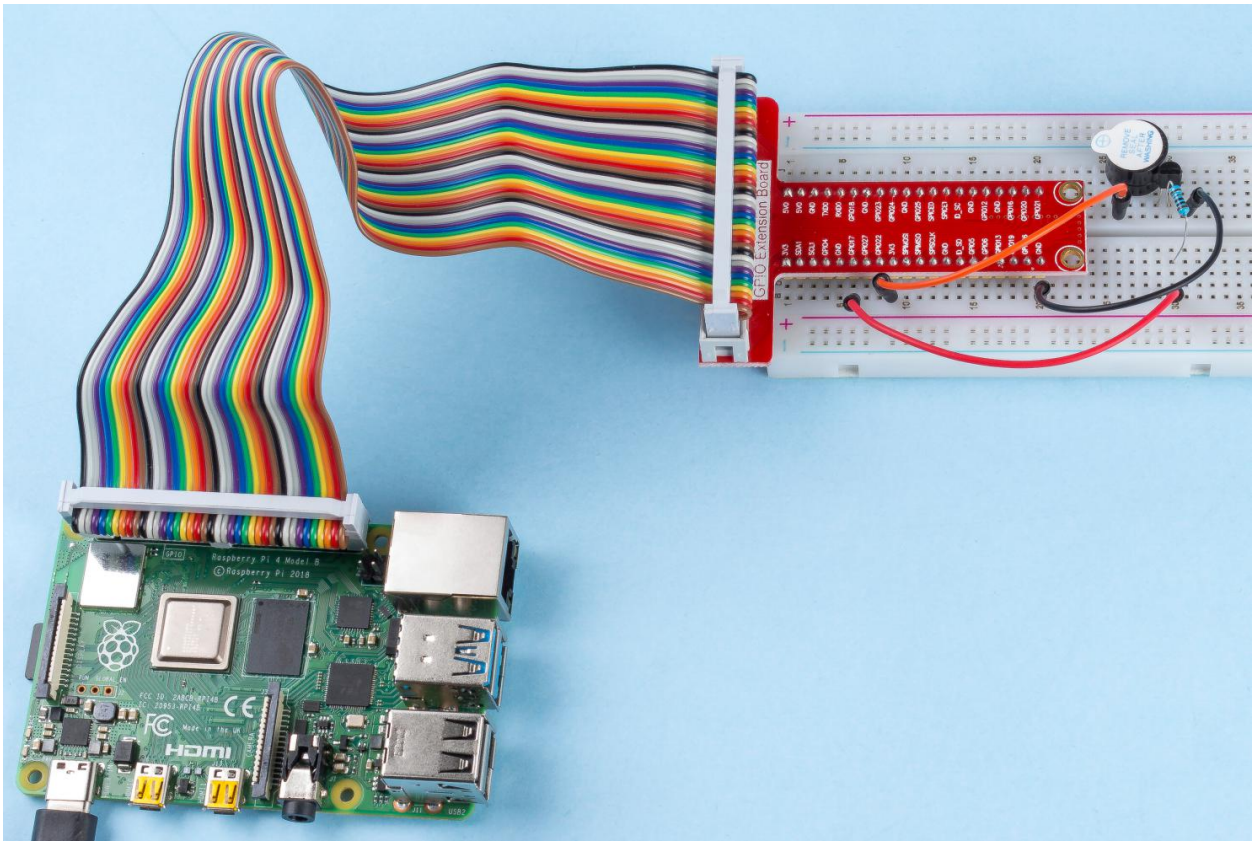
Warten Sie 0,1 Sekunden. Ändern Sie die Schaltfrequenz, indem Sie diesen Parameter ändern.

Bemerkung: Nicht die Schallfrequenz. Der aktive Summer kann die Schallfrequenz nicht ändern.

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

Schließen Sie den Summer.

Phänomen Bild

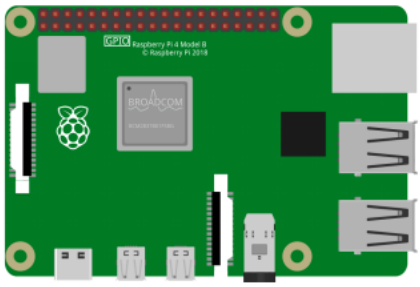
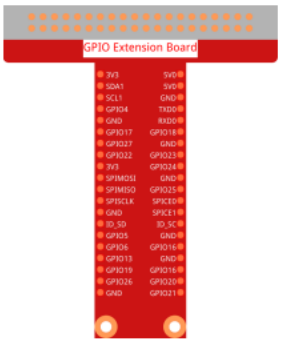
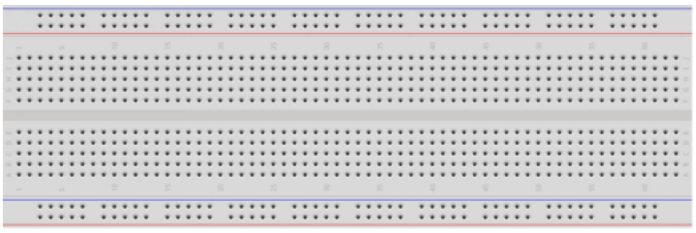


1.7.9 1.2.2 Passiver Summer

Einführung

In dieser Lektion lernen wir, wie man einen passiven Summer dazu bringt, Musik zu spielen.

Komponenten

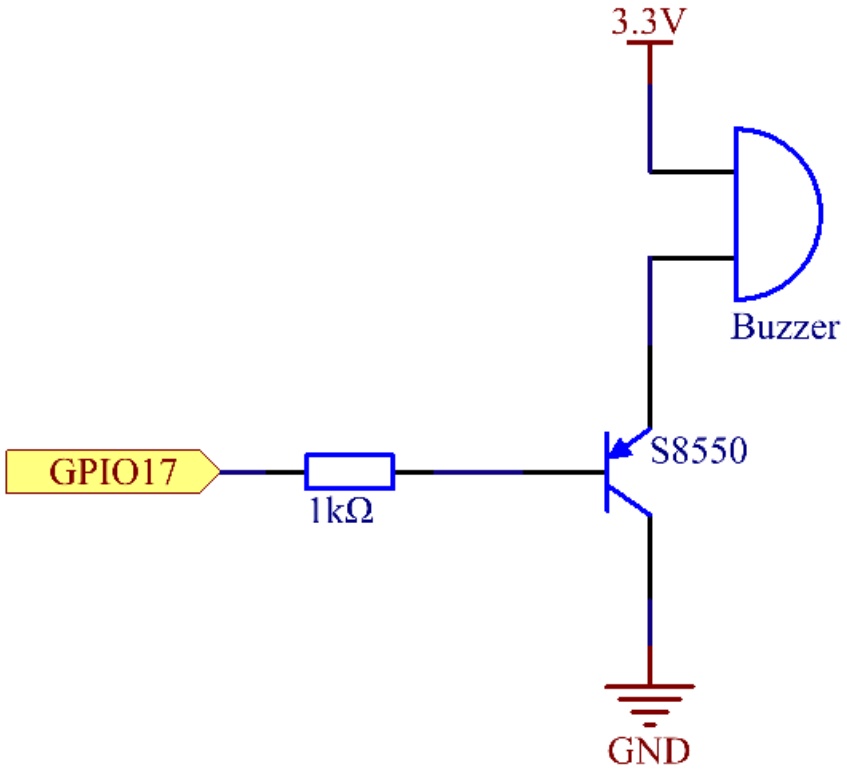
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Passive Buzzer
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(1kΩ)
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * S8550 PNP Transistor
 </td></tr>
</table>

Schematische Darstellung

In diesem Experiment werden ein passiver Summer, ein PNP-Transistor und ein 1k-Widerstand zwischen der Basis des Transistors und GPIO verwendet, um den Transistor zu schützen.

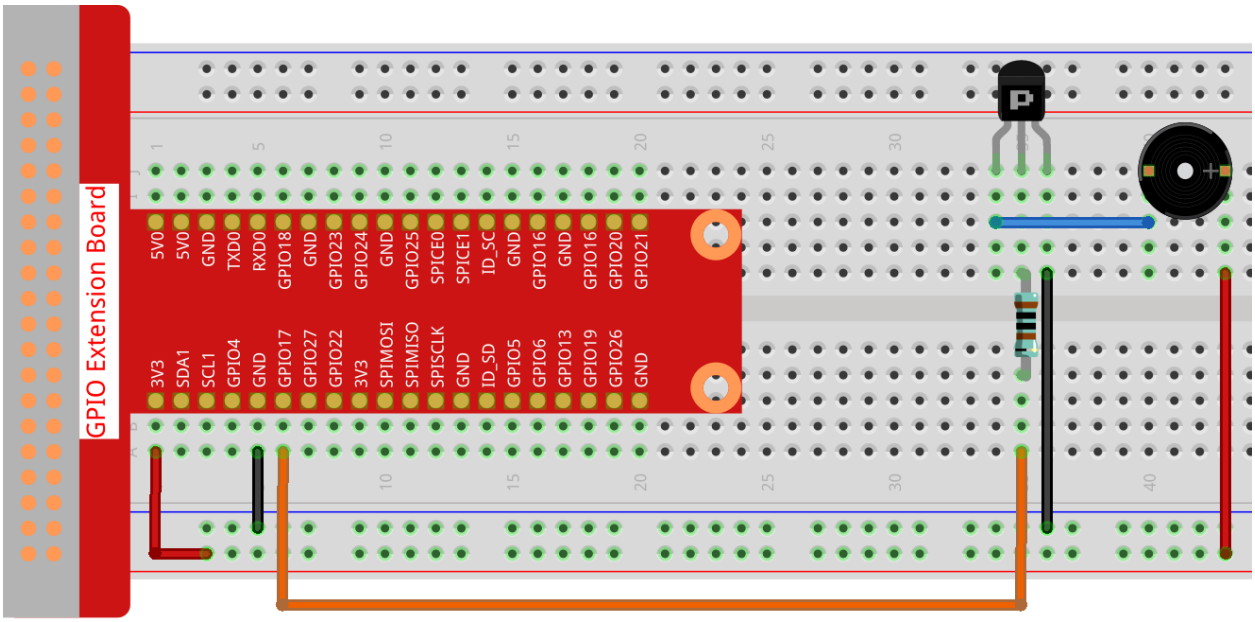
Wenn GPIO17 unterschiedliche Frequenzen erhält, gibt der passive Summer unterschiedliche Töne aus. Auf diese Weise spielt der Summer Musik.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Kompilieren.

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

Schritt 4: Ausführen.

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

Die Kode läuft, der Summer spielt ein Musikstück.

Code

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

#define BuzPin    0

#define  CL1  131
#define  CL2  147
#define  CL3  165
#define  CL4  175
#define  CL5  196
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
#define  CL6  221
#define  CL7  248

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

#define  CH1  525
#define  CH2  589
#define  CH3  661
#define  CH4  700
#define  CH5  786
#define  CH6  882
#define  CH7  990

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

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


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;
    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);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        }

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

    return 0;
}
```

Code Erklärung

```
#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
```

Diese Frequenzen jeder Note sind wie gezeigt. CL bezieht sich auf tiefe Note, CM mittlere Note, CH hohe Note, 1-7 entsprechen den Noten C, D, E, F, G, A, B.

```
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};
```

Das Array `song_1[]` speichert eine Musikpartitur eines Gelieds, in der sich `beat_1[]` auf der Schlage jeder Note im Gelied bezieht (0,5 S für jeder Schlage).

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

Dadurch wird ein softwaregesteuerter Ton Pin erstellt. Sie können einen beliebigen GPIO-Pin verwenden. Die Pin-Nummerierung entspricht der von Ihnen verwendeten Funktion `wiringPiSetup()` . Der Rückgabewert ist 0 für Erfolg. Alles andere und Sie sollten die globale Fehlerbehebung überprüfen, um festzustellen, was schief gelaufen ist.

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

Verwenden Sie eine for-Anweisung, um `song_1` abzuspielen.

In der Urteilsbedingung wird `i<sizeof(song_1)/4` , „devide by 4“ verwendet, da das Array `song_1[]` ein Array des Datentyps ganzer Nummer ist und jedes Element vier Bytes in Anspruch nimmt.

Die Anzahl der Elemente in `song_1` (die Anzahl der Noten) wird erhalten, indem `sizeof(song_4)` um 4 geteilt wird.

Damit jede Note für `beat * 500ms` gespielt werden kann, wird die `delay(beat_1[i]*500)` aufgerufen.

Der Prototyp von `softToneWrite(BuzPin, song_1[i])` :

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

Dadurch wird der Tonfrequenzwert am angegebenen Pin aktualisiert. Der Ton hört erst auf zu spielen, wenn Sie die Frequenz auf 0 eingestellen.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

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

Die Kode läuft, der Summer spielt ein Musikstück.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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 ]

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],
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            CM[3], CM[4], CM[4], CM[3], CM[2], CM[3], CM[1], CM[1],
            CM[3], CM[2], CL[5], CL[7], CM[2], CM[1]        ]

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

def setup():

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

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

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

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

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

Code Erklärung

```
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
```

Dies sind die Frequenzen jeder Note. Die erste 0 ist das Überspringen von `CL[0]` , so dass die Nummer 1-7 dem

CDEFGAB des Tons entspricht.

```
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] ]
```

Diese Arrays sind die Noten eines Gelieds.

```
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 ]
```

Jeder Klangschlage (jede Nummer) repräsentiert den -Schlage oder 0.5s

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

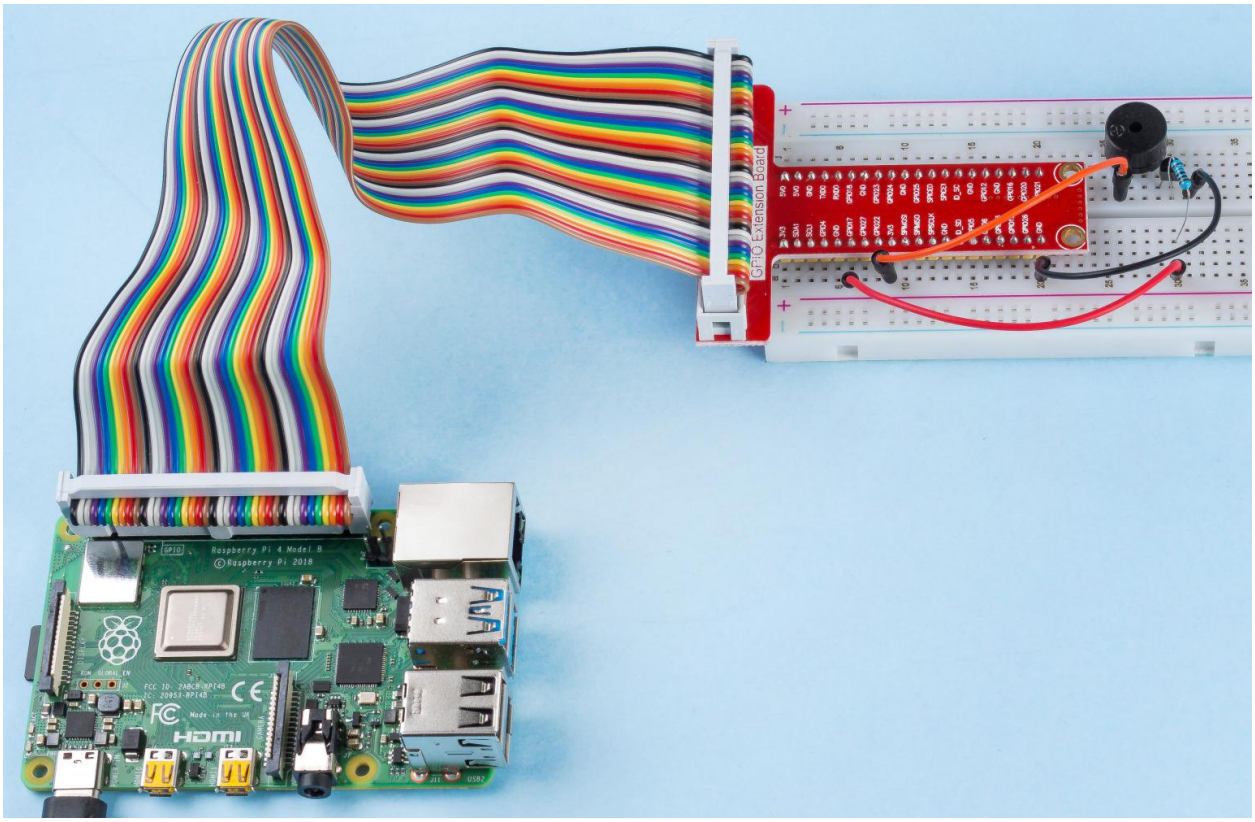
Definieren Sie den Pin-Summer als PWM-Pin, stellen Sie dann seine Frequenz auf 440 ein und `Buzz.start(50)` wird zum Ausführen von PWM verwendet. Und stellen Sie auch den Arbeitszyklus auf 50% ein.

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

Führen Sie eine for-Schleife aus, dann spielt der Summer die Noten im Array `song_1[]` mit den Schlagen im Array `beat_1[]`.

Jetzt können Sie den passiven Summer Musik spielen hören.

Phänomen Bild



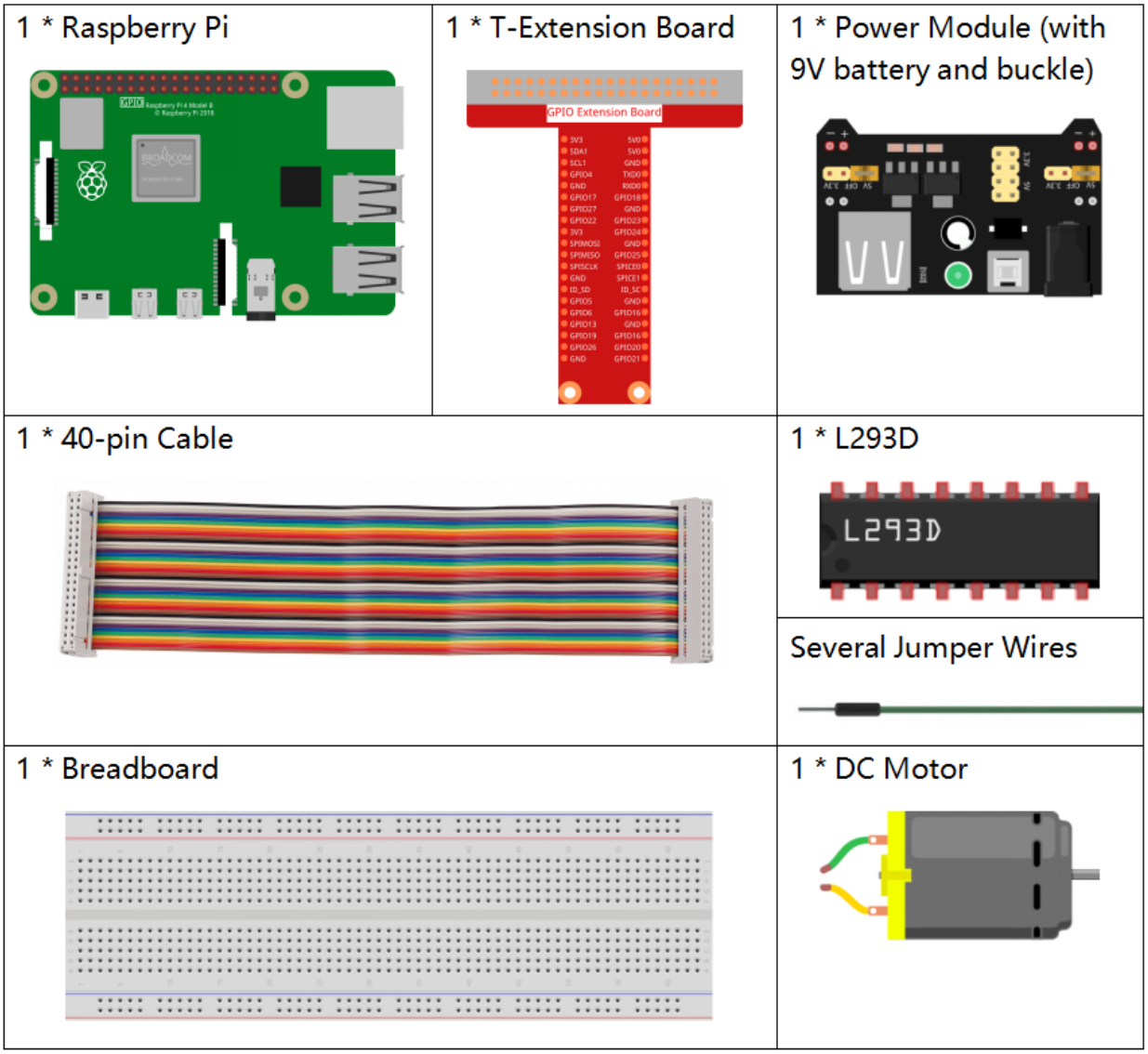
1.3 Treiber

1.7.10 1.3.1 Motor

Einführung

In dieser Lektion lernen wir, mit L293D einen Gleichstrommotor antreiben und ihn im und gegen den Uhrzeigersinn drehen. Da der Gleichstrommotor aus Sicherheitsgründen einen größeren Strom benötigt, verwenden wir hier das Stromversorgungsmodul zur Versorgung der Motoren.

Komponenten

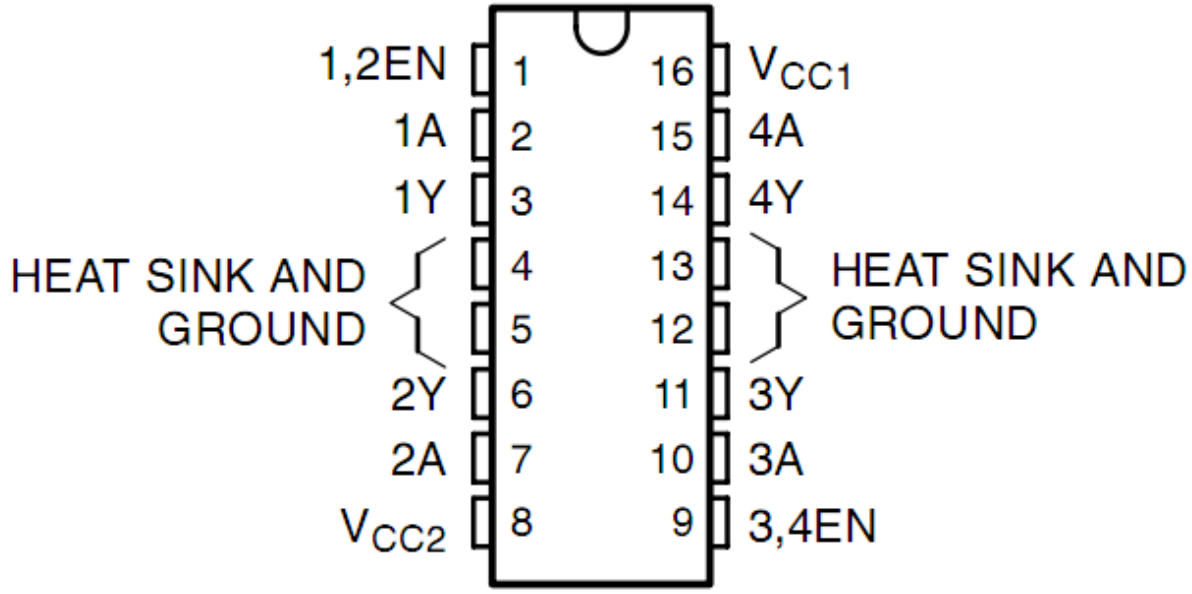


Prinzip

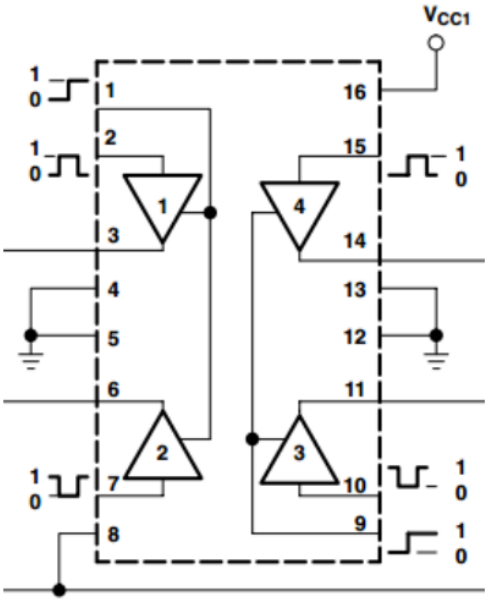
L293D

L293D ist ein 4-Kanal-Motortreiber, der durch einen Chip mit hoher Spannung und hohem Strom integriert ist. Es ist für den Anschluss an Standard-DTL-, TTL-Logikniveau und ansteuerungsinduktive Lasten (wie Relaisspulen, Gleichstrom-, Schrittmotoren) sowie Leistungsschalttransistoren usw. ausgelegt. Gleichstrommotoren sind Geräte, die elektrische Gleichstromenergie in mechanische Energie umwandeln. Sie werden im elektrischen Antrieb wegen ihrer überlegenen Geschwindigkeitsregelungsleistung häufig verwendet.

Siehe die Abbildung der Pins unten. Der L293D verfügt über zwei Pins (Vcc1 und Vcc2) für die Stromversorgung. Vcc2 wird verwendet, um den Motor mit Strom zu versorgen, während Vcc1 dient, um den Chip zu versorgen. Da hier ein kleiner Gleichstrommotor verwendet wird, verbinden Sie beide Pins mit + 5V.



Das Folgende ist die interne Struktur von L293D. Pin EN ist ein Freigabepin und funktioniert nur mit hohem Niveau. A steht für Eingabe und Y für Ausgabe. Sie können die Beziehung zwischen ihnen unten rechts sehen. Wenn Pin EN auf Hohe Niveau steht und A auf High steht, gibt Y Hohe Niveau aus. Wenn A niedrig ist, gibt Y einen niedrigen Niveau aus. Wenn Pin EN auf niedgrigen Niveau steht, funktioniert der L293D nicht.



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)

Gleichspannungs Motor

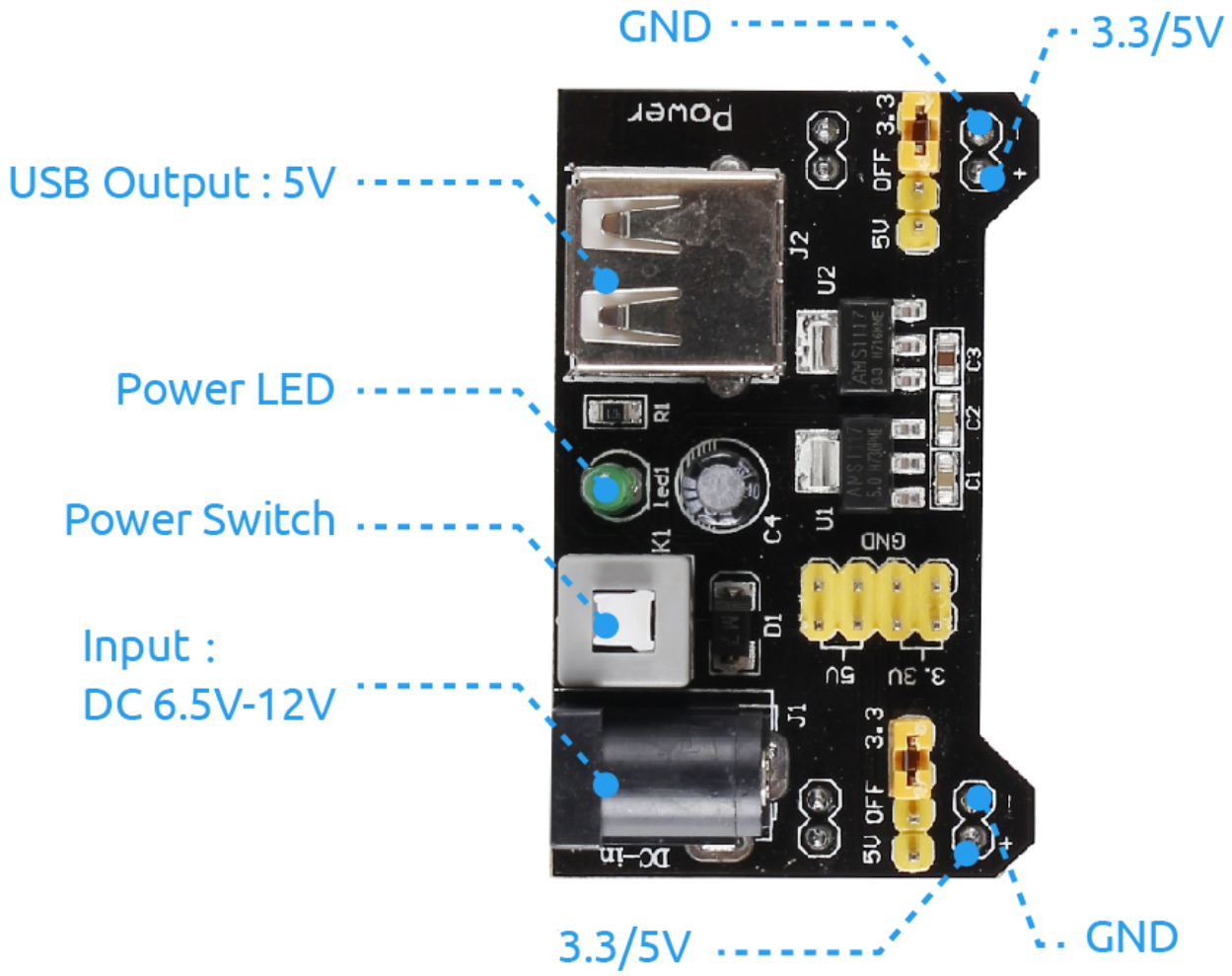
Dies ist ein 5V Gleichstrommotor. Es dreht sich, wenn Sie den beiden Anschlüssen des Kupferblechs einen hohen und einen niedrigen Niveau geben. Der Einfachheit halber können Sie die Pin daran schweißen.

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

Energieversorgung Modul

In diesem Experiment werden große Ströme benötigt, um den Motor anzutreiben, insbesondere wenn er startet und stoppt, was die normale Arbeit von Raspberry Pi stark beeinträchtigt. Da versorgen wir diesen Motor separat mit Strom, damit er sicher und gleichmäßig läuft.

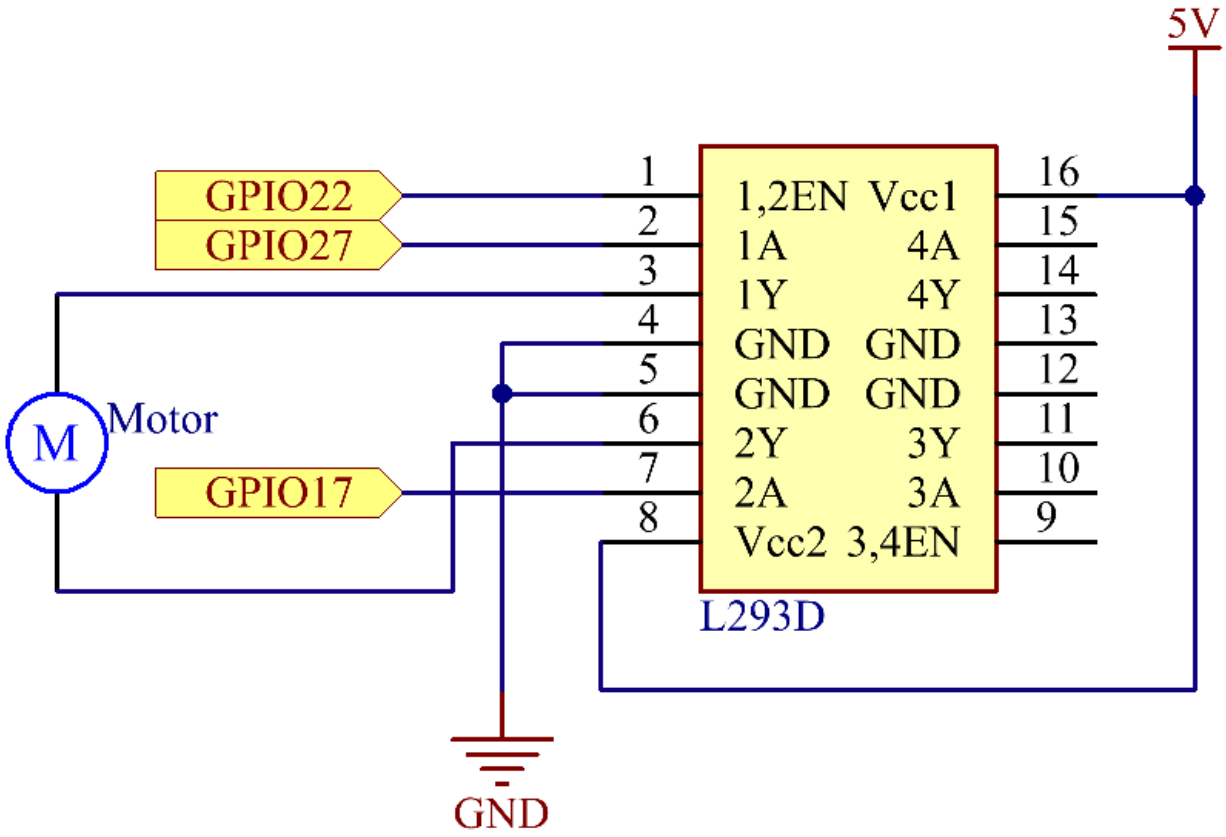
Sie können es einfach in das Steckbrett einstecken, um Strom zu liefern. Es liefert eine Spannung von 3,3V und 5V, und Sie können entweder über eine mitgelieferte Überbrückungskappe anschließen.



Schematische Darstellung

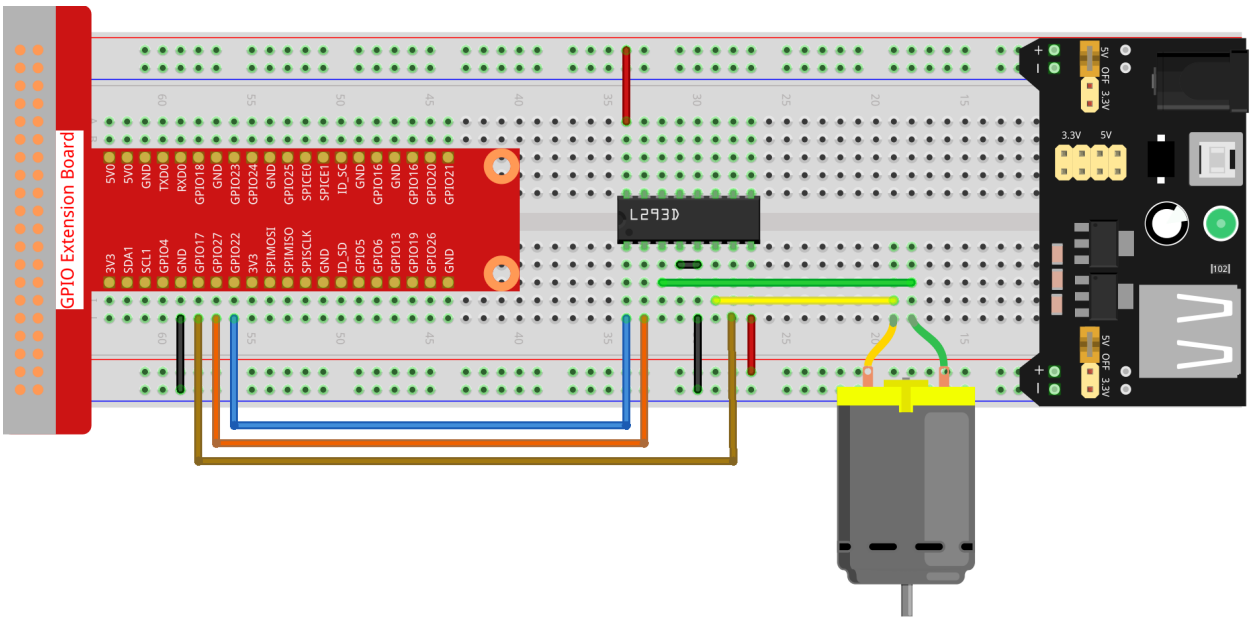
Stecken Sie das Netzteilmodul in das Steckbrett und setzen Sie die Überbrückungskappe auf 5V, dann wird eine Spannung von 5V ausgegeben. Verbinden Sie Pin 1 des L293D mit GPIO22 und stellen Sie ihn auf Hohe Niveau ein. Verbinden Sie Pin2 mit GPIO27 und Pin7 mit GPIO17 und setzen Sie dann einen Pin hoch, während der andere niedrig ist. So können Sie die Drehrichtung des Motors ändern.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Bemerkung: Das Leistungsmodul kann eine 9-V-Batterie mit der im Kit enthaltenen 9V-Batterieschnalle anlegen. Setzen Sie die Überbrückungskappe des Leistungsmoduls in die 5V-Busleisten des Steckbretts ein.

Für Benutzer in C-Sprache

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Kompilieren.

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

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Während die Kode läuft, dreht sich der Motor zuerst 5 Sekunden lang im Uhrzeigersinn und stoppt dann 5 Sekunden lang. Danach dreht er sich 5 Sekunden lang gegen den Uhrzeigersinn. Anschließend stoppt der Motor für 5S. Diese Reihe von Aktionen wird wiederholt ausgeführt.

Code

```
#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);
        }

        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;
}
```

Code Erklärung

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

Aktivieren Sie den L239D.

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

Stellen Sie einen hohen Niveau für 2A ein (Pin 7); Da sich 1,2EN (Pin 1) auf einem hohen Niveau befindet, gibt 2Y einen hohen Niveau aus.

Stellen Sie einen niedrigen Niveau für 1A ein, dann gibt 1Y einen niedrigen Niveau aus und der Motor dreht sich.

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

Diese Schleife soll 3*1000ms verzögern.

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

Wenn sich 1,2EN (Pin1) auf einem niedrigen Niveau befindet, funktioniert L293D nicht. Motor stoppt sich zu drehen.

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

Den Stromfluss des Motors umkehren, dann dreht sich der Motor umgekehrt.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Ausführen.

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

Während die Kode läuft, dreht sich der Motor zuerst 5 Sekunden lang im Uhrzeigersinn und stoppt dann 5 Sekunden lang. Danach dreht er sich 5 Sekunden lang gegen den Uhrzeigersinn. Anschließend stoppt der Motor für 5S. Diese Reihe von Aktionen wird wiederholt ausgeführt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

dhtPin = 17
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
GPIO.setmode(GPIO.BCM)

MAX_UNCHANGE_COUNT = 100

STATE_INIT_PULL_DOWN = 1
STATE_INIT_PULL_UP = 2
STATE_DATA_FIRST_PULL_DOWN = 3
STATE_DATA_PULL_UP = 4
STATE_DATA_PULL_DOWN = 5

def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)

    unchanged_count = 0
    last = -1
    data = []
    while True:
        current = GPIO.input(dhtPin)
        data.append(current)
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > MAX_UNCHANGE_COUNT:
                break

    state = STATE_INIT_PULL_DOWN

    lengths = []
    current_length = 0

    for current in data:
        current_length += 1

        if state == STATE_INIT_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_INIT_PULL_UP
            else:
                continue
        if state == STATE_INIT_PULL_UP:
            if current == GPIO.HIGH:
                state = STATE_DATA_FIRST_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_FIRST_PULL_DOWN:
            if current == GPIO.LOW:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                state = STATE_DATA_PULL_UP
            else:
                continue
        if state == STATE_DATA_PULL_UP:
            if current == GPIO.HIGH:
                current_length = 0
                state = STATE_DATA_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_PULL_DOWN:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_PULL_UP
            else:
                continue
    if len(lengths) != 40:
        #print ("Data not good, skip")
        return False

    shortest_pull_up = min(lengths)
    longest_pull_up = max(lengths)
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []
    the_bytes = []
    byte = 0

    for length in lengths:
        bit = 0
        if length > halfway:
            bit = 1
        bits.append(bit)
    #print ("bits: %s, length: %d" % (bits, len(bits)))
    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0
    #print (the_bytes)
    checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False

    return the_bytes[0], the_bytes[2]

def main():

    while True:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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()
```

Code Erklärung

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

Erstellen Sie eine Funktion, `motor()` , deren Variable die Richtung ist. Wenn die Bedingung erfüllt ist, dass `direction = 1` erfüllt ist, dreht sich der Motor im Uhrzeigersinn. Wenn die `direction = -1` ist, dreht sich der Motor gegen den Uhrzeigersinn. und unter der Bedingung, dass `direction = 0` ist, hört es auf, sich zu drehen.

```
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)
```

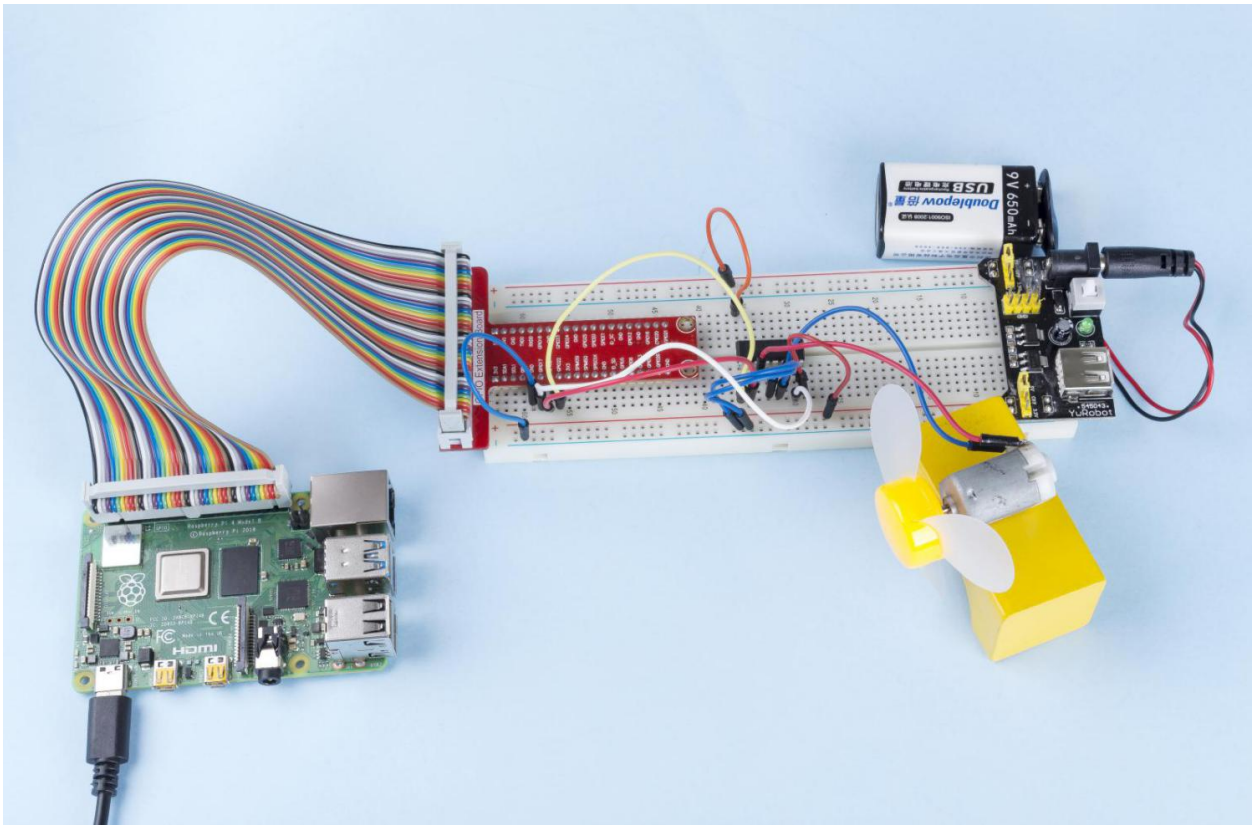
In der `main()` -erstellen Sie ein Array, `directions[]` , in dem CW gleich 1 ist, der Wert von CCW -1 ist und die Nummer 0 auf Stop verweist.

Während die Kode läuft, dreht sich der Motor zuerst 5 Sekunden lang im Uhrzeigersinn und stoppt dann 5 Sekunden

lang. Danach dreht er sich 5 Sekunden lang gegen den Uhrzeigersinn. Anschließend stoppt der Motor für 5s. Diese Reihe von Aktionen wird wiederholt ausgeführt.

Jetzt sollte sich das Motorblatt drehen.

Phänomen Bild

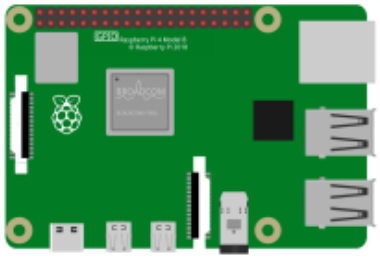
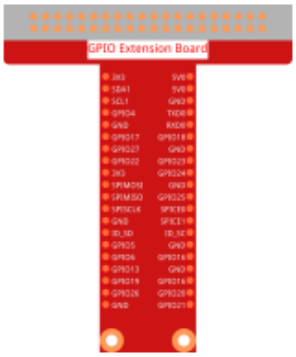


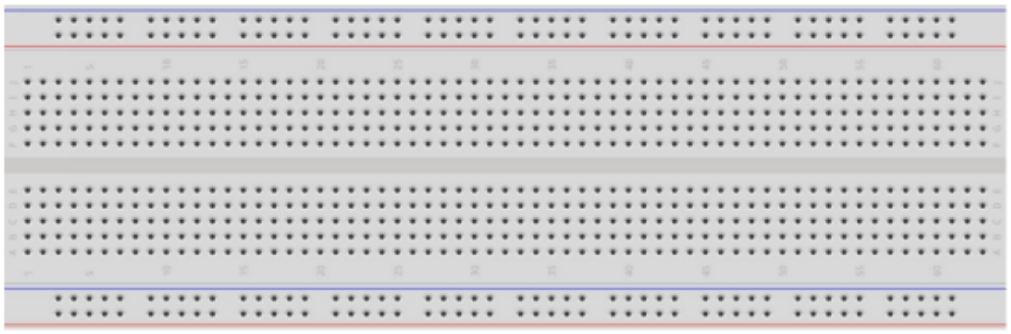
1.7.11 1.3.2 Servo

Einführung

In dieser Lektion lernen wir, wie man das Servo dreht.

Komponenten

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Servo

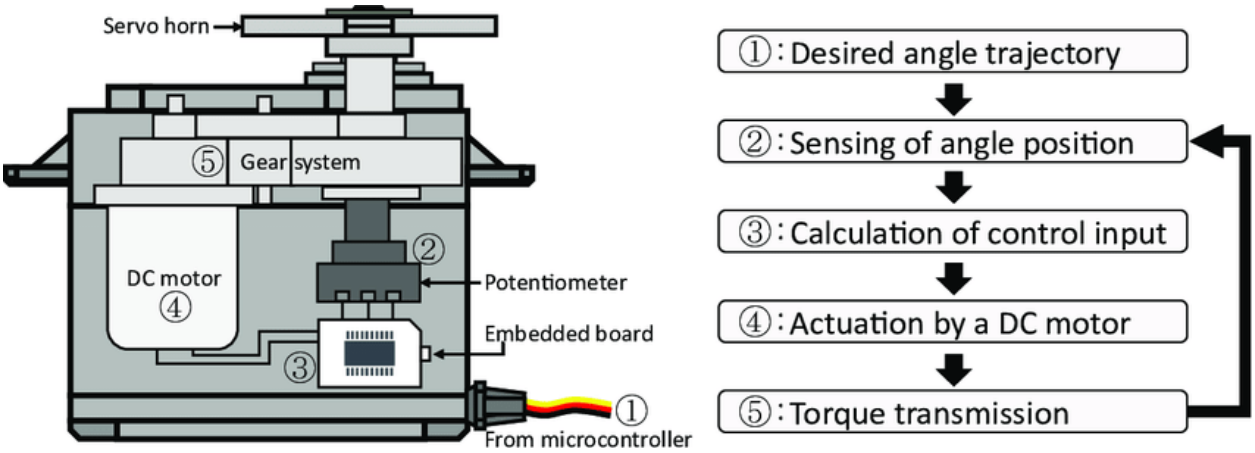
Several Jumper Wires
 </td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

Prinzip

Servo

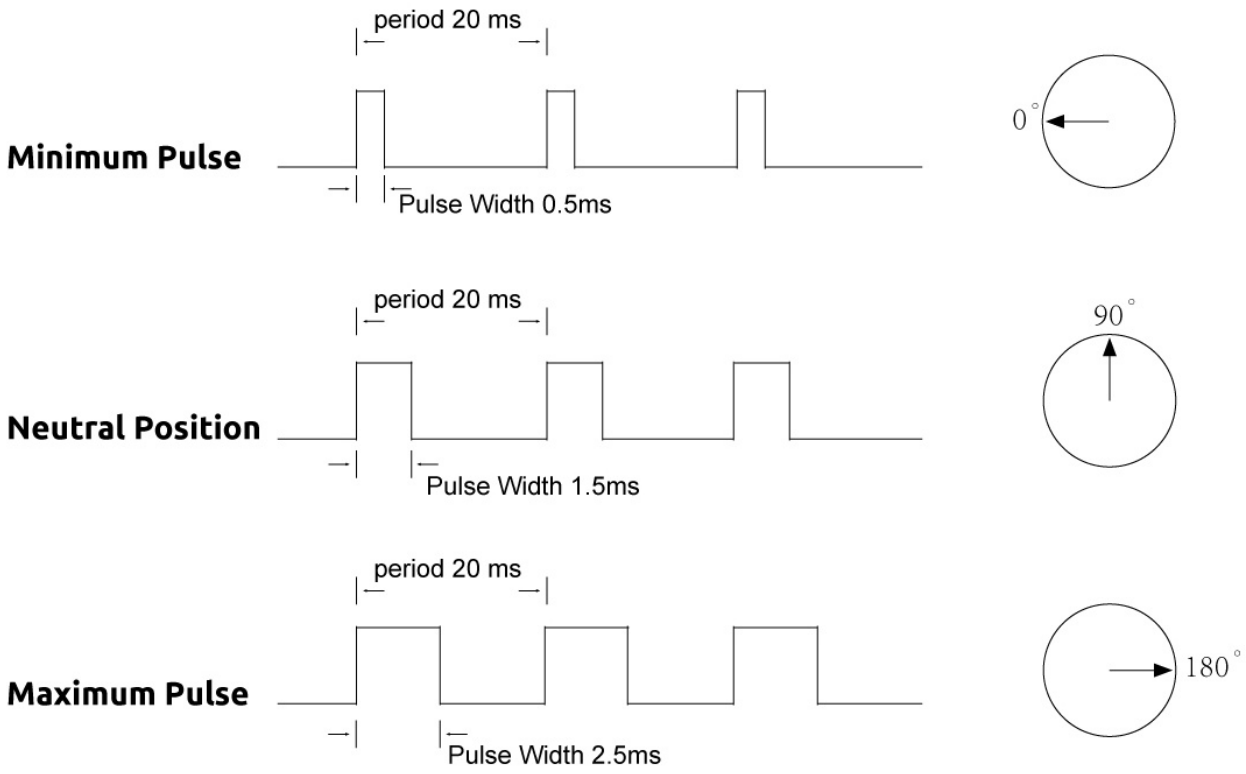
Ein Servo besteht im Allgemeinen aus folgenden Teilen: Gehäuse, Welle, Getriebe, Potentiometer, Gleichstrommotor und eingebettete Platine.

Das funktioniert so: Der Mikrocontroller sendet PWM-Signale an das Servo, und dann empfängt die im Servo eingebettete Karte die Signale über den Signal Pin und steuert den Motor im Inneren, um sich zu drehen. Infolgedessen treibt der Motor das Zahnradsystem an und motiviert dann die Welle nach dem Abbremsen. Die Welle und das Potentiometer des Servos sind miteinander verbunden. Wenn sich die Welle dreht, treibt sie das Potentiometer an, sodass das Potentiometer ein Spannungssignal an die eingebettete Platine ausgibt. Dann bestimmt das Board die Richtung und Geschwindigkeit der Drehung basierend auf der aktuellen Position, so dass es genau an der richtigen Position wie definiert anhalten und dort halten kann.



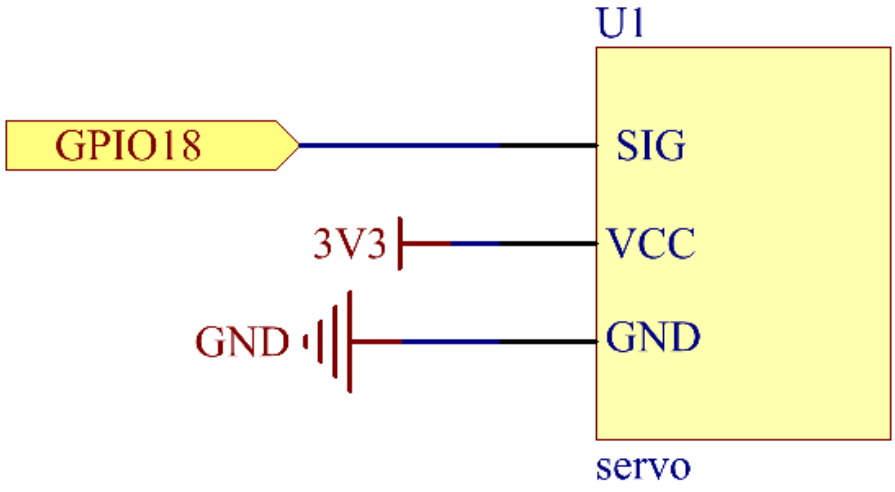
Der Winkel wird durch die Dauer eines Impulses bestimmt, der an den Steuerdraht angelegt wird. Dies wird als Pulsweitenmodulation bezeichnet. Das Servo erwartet alle 20 ms einen Impuls. Die Länge des Impulses bestimmt, wie weit sich der Motor dreht. Zum Beispiel bringt ein Impuls von 1,5 ms den Motor in die 90-Grad-Position (neutrale Position).

Wenn ein Impuls an ein Servo gesendet wird, das weniger als 1,5 ms beträgt, dreht sich das Servo in eine Position und hält seine Ausgangswelle einige Grad gegen den Uhrzeigersinn vom Neutralpunkt entfernt. Wenn der Impuls breiter als 1,5 ms ist, tritt das Gegenteil auf. Die minimale Breite und die maximale Impulsbreite, die das Servo anweisen, sich in eine gültige Position zu drehen, sind Funktionen jedes Servos. Im Allgemeinen ist der minimale Impuls ungefähr 0,5 ms breit und der maximale Impuls ist 2,5 ms breit.



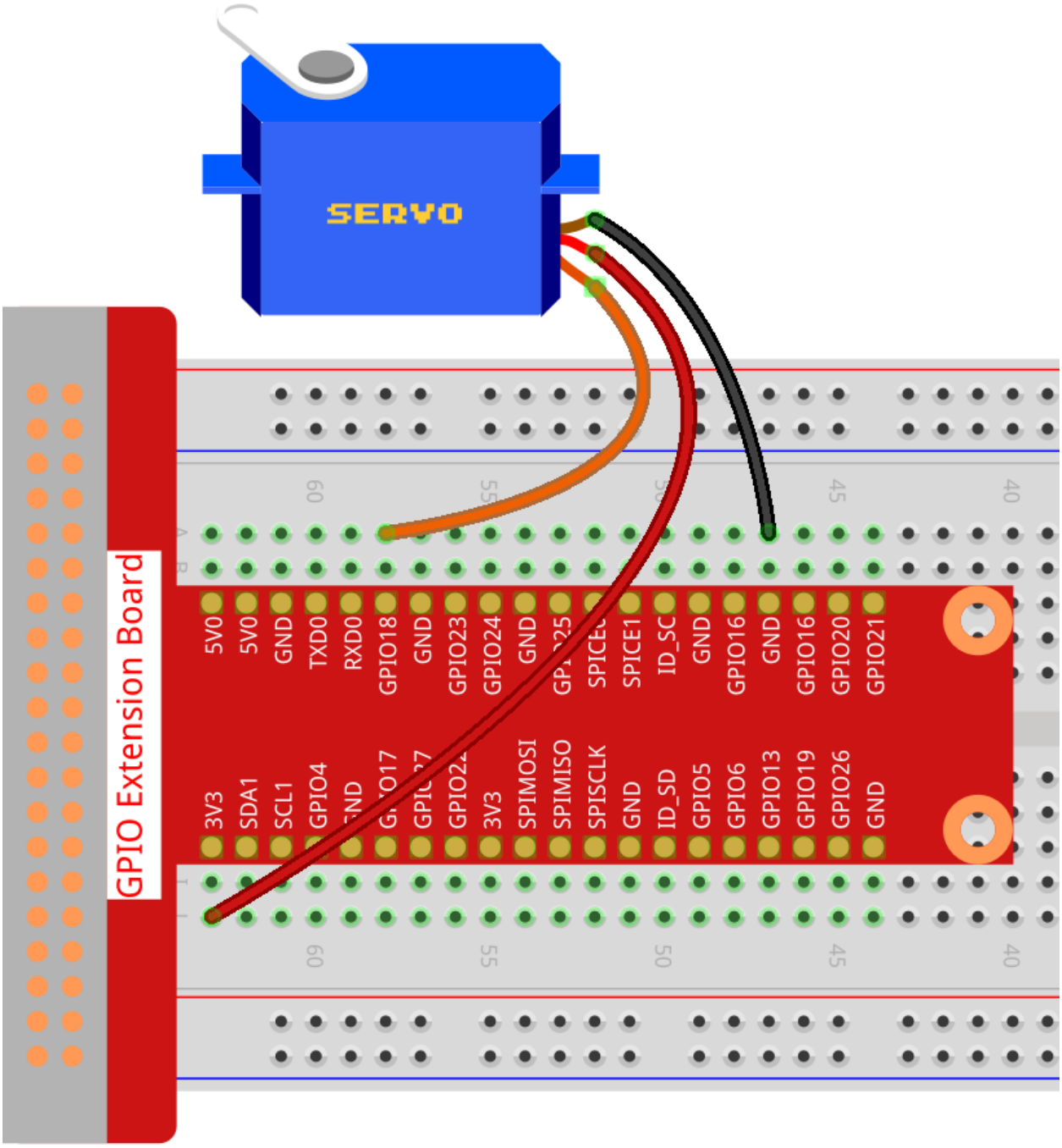
Schematische Darstellung

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem das Programm ausgeführt wurde, dreht sich das Servo kreisförmig von 0 Grad auf 180 Grad und dann von 180 Grad auf 0 Grad.

Code

```
#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 !");
        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);
        }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        delay(1000);
    }
    return 0;
}
```

Code Erklärung

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

Erstellen Sie eine `Map()` - Funktion, um den Wert im folgenden Kode zuzuordnen.

```
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));
}
```

Erstellen Sie eine Funktion, `setAngle()` , um den Winkel zum Servo zu schreiben.

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

Diese Funktion kann das Einschaltdauer der PWM ändern.

Damit sich das Servo auf 0 bis 180° dreht, sollte sich die Impulsbreite im Bereich von 0,5 ms bis 2,5 ms ändern, wenn die Periode 20 ms beträgt. In der Funktion `softPwmCreate()` haben wir festgelegt, dass der Zeitraum 200x100us = 20ms beträgt. Daher müssen wir 0 ~ 180 bis 5x100us ~ 25x100us zuordnen.

Der Prototyp dieser Funktion ist unten dargestellt.

```
int softPwmCreateint pinint initialValueint pwmRange;
```

pin: Jeder GPIO-Pin von Raspberry Pi kann als PWM-Pin gesetzt werden.

initialValue: Die anfängliche Impulsbreite ist der initialValue mal 100us.

pwmRange: Die Periode von PWM ist die pwmRange mal 100us.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

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

Nachdem das Programm ausgeführt wurde, dreht sich das Servo kreisförmig von 0 Grad auf 180 Grad und dann von 180 Grad auf 0 Grad.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

ServoPin = 18

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

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

Code Erklärung

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

Stellen Sie den ServoPin auf den PWM-Pin, dann die Frequenz auf 50 Hz und die Periode auf 20 ms.

`p.start(0)` : Führen Sie die PWM-Funktion, und setzen Sie den Anfangswert auf 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
```

Erstellen Sie eine Funktion, `setAngle()` , um einen Winkel von 0 bis 180 in das Servo zu schreiben.

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

Diese Kode wird verwendet, um den Winkel im Bereich von 0 bis 180° zu begrenzen.

Die Funktion `min()` gibt das Minimum der Eingabewerte zurück. Wenn 180 <Winkel, then 180 zurückgeben, wenn nicht, Winkel zurückgeben .

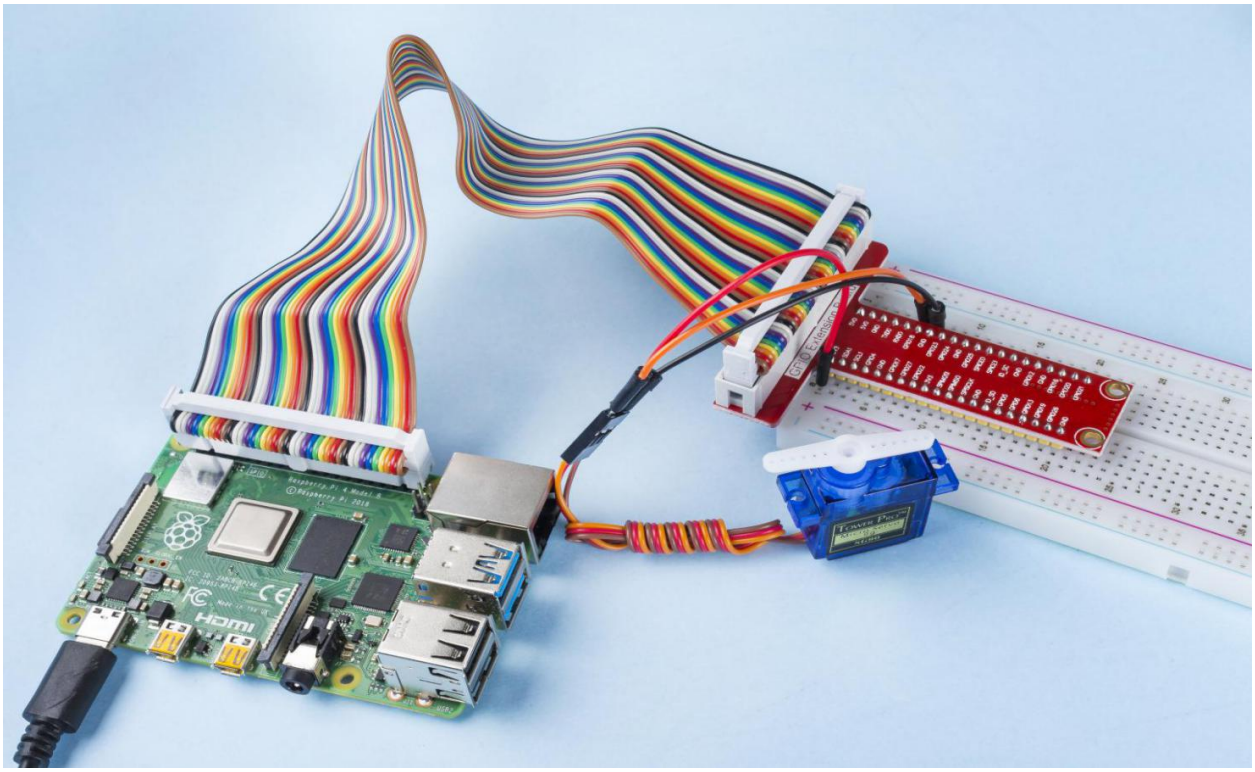
Die `max()` -Methode gibt das maximale Element in einem iterierbaren oder größten von zwei oder mehr Parametern zurück. Wenn 0> Winkel, dann 0 zurückgeben, wenn nicht, Winkel zurückgeben.

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

Um einen Bereich von 0 bis 180° zum Servo zu rendern, wird die Impulsbreite des Servos auf 0,5 ms (500 us) bis 2,5 ms (2500 us) eingestellt.

Die Periode der PWM beträgt 20 ms (20000us), daher beträgt das Tastverhältnis der PWM (500/20000)% - (2500/20000)%, und der Bereich 0 bis 180 wird auf 2.5 bis 12.5 abgebildet.

Phänomen Bild

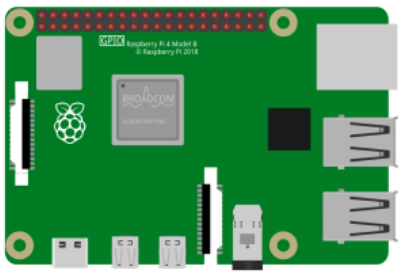
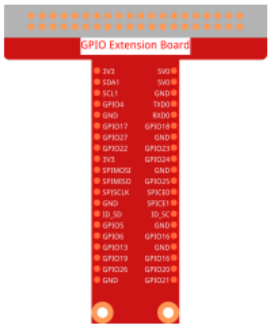
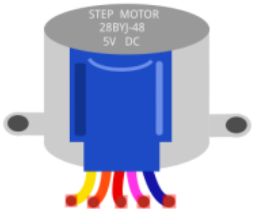
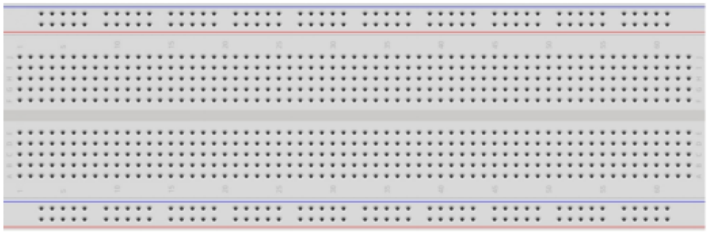
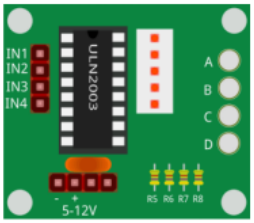


1.7.12 1.3.3 Schrittmotor

Einführung

Schrittmotoren können aufgrund ihres einzigartigen Designs ohne Rückkopplungsmechanismen mit hoher Genauigkeit gesteuert werden. Die Welle eines Schrittmachers, der mit einer Reihe von Magneten montiert ist, wird von einer Reihe elektromagnetischer Spulen gesteuert, die in einer bestimmten Reihenfolge positiv und negativ geladen werden und diese in kleinen „Schritten“ präzise vorwärts oder rückwärts bewegen.

Komponenten

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

Prinzip

Schrittmotor

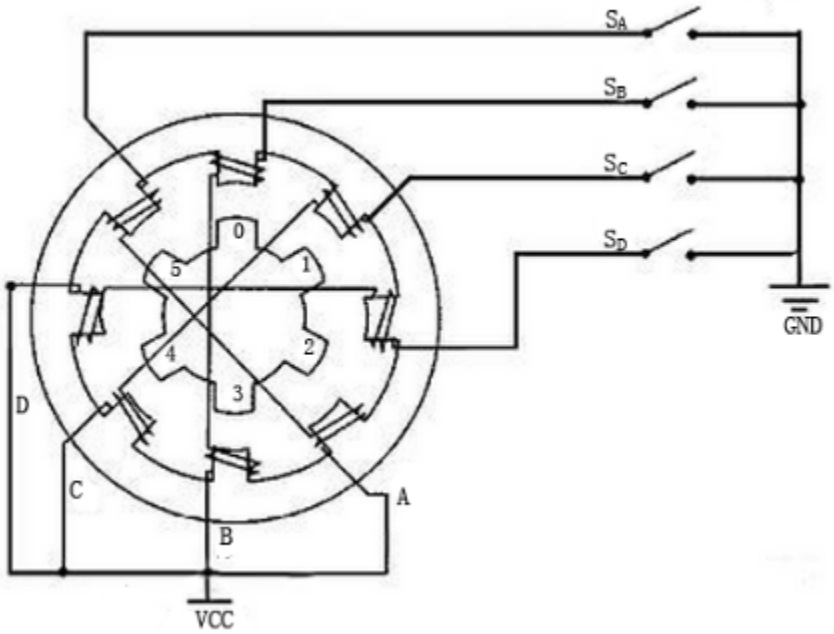
Es gibt zwei Typen von Steppern, Unipolare und Bipolare, und es ist sehr wichtig zu wissen, mit welchem Typ Sie arbeiten. In diesem Experiment verwenden wir einen unipolaren Stepper.

Der Schrittmotor ist ein Vierphasenmotor, der eine Gleichstromversorgung mit Unipolarität verwendet. Solange Sie alle Phasenwicklungen des Motors durch eine geeignete Zeitfolge elektrifizieren, können Sie ihn Schritt für Schritt drehen lassen. Das schematische Diagramm eines vierphasigen reaktiven Schrittmotors:

In der Abbildung befindet sich in der Mitte des Motors ein Rotor - ein zahnradförmiger Permanentmagnet. Um den Rotor herum sind 0 bis 5 Zähne. Dann weiter draußen gibt es 8 Magnetpole, wobei jeweils zwei gegenüberliegende durch Spulenwicklung verbunden sind. Sie bilden also vier Paare von A nach D, was als Phase bezeichnet wird. Es verfügt über vier Anschlusskabel, die mit den Schaltern SA, SB, SC und SD verbunden werden können. Daher sind die vier Phasen in der Schaltung parallel und die zwei Magnetpole in einer Phase sind in Reihe geschaltet.

So funktioniert ein 4-Phasen-Schrittmotor:

Wenn der Schalter SB eingeschaltet ist, sind die Schalter SA, SC und SD ausgeschaltet, und die B-Phasen-Magnetpole sind auf die Zähne 0 und 3 des Rotors ausgerichtet. Gleichzeitig erzeugen Zahn 1 und 4 versetzte Zähne mit C- und D-Phasenpolen. Zahn 2 und 5 erzeugen versetzte Zähne mit D- und A-Phasenpolen. Wenn der Schalter SC eingeschaltet ist, die Schalter SB, SA und SD ausgeschaltet sind, dreht sich der Rotor unter dem Magnetfeld der C-Phasenwicklung und dem zwischen Zahn 1 und 4. Dann richten sich Zahn 1 und 4 an den Magnetpolen der C-Phasenwicklung aus. Während Zahn 0 und 3 versetzte Zähne mit A- und B-Phasenpolen erzeugen, erzeugen Zahn 2 und 5 versetzte Zähne mit den Magnetpolen von A- und D-Phasenpolen. Die ähnliche Situation geht weiter und weiter. Schalten Sie die Phasen A, B, C und D nacheinander ein, und der Rotor dreht sich in der Reihenfolge A, B, C und D.



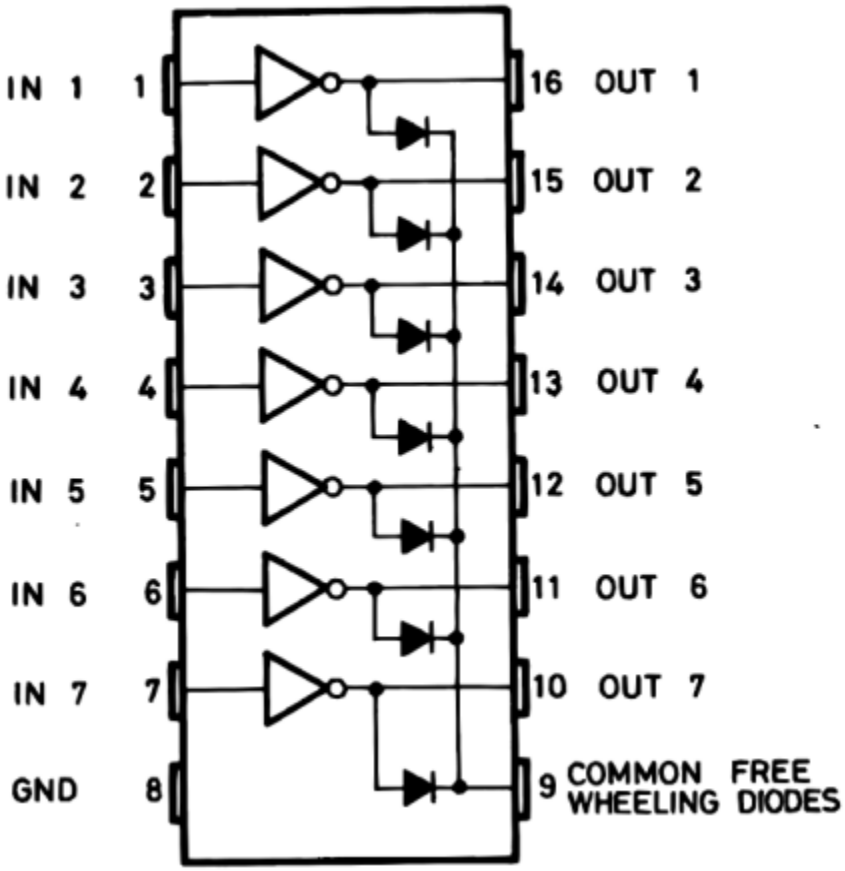
Der Vierphasen-Schrittmotor verfügt über drei Betriebsarten: einfach vierstufig, doppelt vierstufig und achtstufig. Der Schrittwinkel für den einzelnen vierstufigen und den doppelten vierstufigen ist gleich, aber das Antriebsmoment für den einzelnen vierstufigen ist kleiner. Der Schrittwinkel des Achtstufens ist halb so groß wie der des Einzel-Vier-Stufen- und des Doppel-Vier-Stufen-Winkels. Somit kann der achtstufige Betriebsmodus ein hohes Antriebsmoment beibehalten

und die Steuergenauigkeit verbessern.

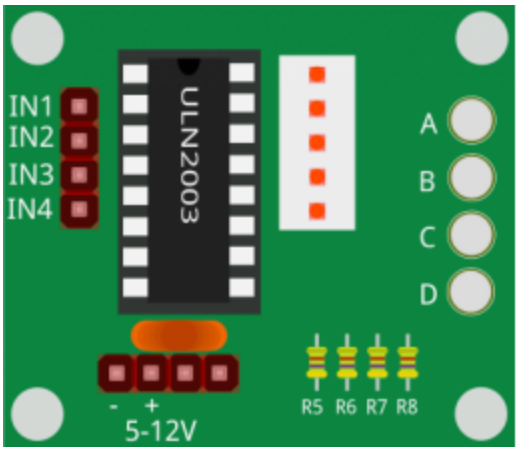
Der von uns verwendete Stator des Schrittmotors hat 32 Magnetpole, sodass ein Kreis 32 Schritte benötigt. Die Abtriebswelle des Schrittmotors ist mit einem Untersetzungsgetriebesatz verbunden, und das Untersetzungsverhältnis beträgt 1/64. Die endgültige Abtriebswelle dreht also einen Kreis, der einen Schritt von 32 * 64 = 2048 erfordert.

ULN2003

Um den Motor in die Schaltung einzubringen, muss eine Treiberplatine verwendet werden. Stepper Motor Driver-ULN2003 ist eine 7-Kanal-Inverterschaltung. Das heißt, wenn der Eingangspin auf High-Pegel ist, ist der Ausgangspin von ULN2003 auf Low-Pegel und umgekehrt. Wenn wir IN1 High-Pegel und IN2, IN3 und IN4 Low-Pegel liefern, dann ist das Ausgangsende OUT1 auf Low-Pegel und alle anderen Ausgangsenden auf High-Pegel. Die interne Struktur des Chips ist wie folgt dargestellt.

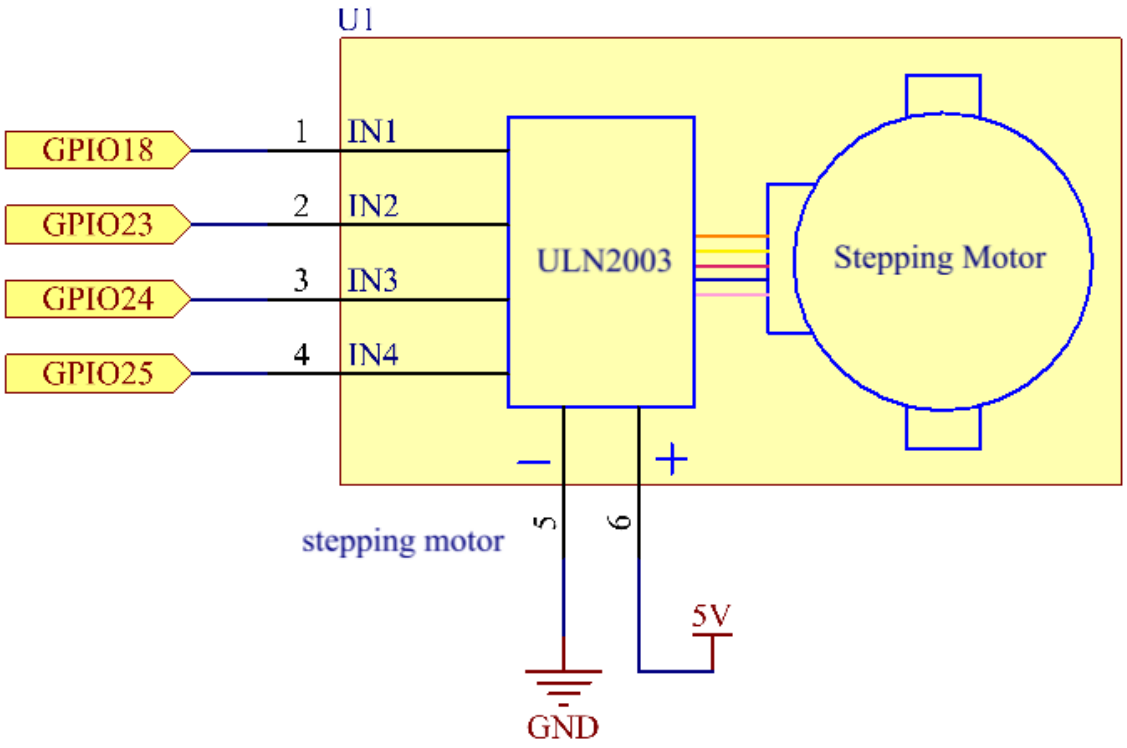


Der Schrittmotortreiber, der aus dem ULN2003-Chip und 4 LEDs besteht, ist wie folgt dargestellt. Auf der Platine fungieren IN1, IN2, IN3 und IN4 als Eingang und die vier LEDs A, B, C, D sind die Anzeigen des Eingangspins. Zusätzlich sind OUT1, OUT2, OUT3 und OUT4 mit SA, SB, SC und SD am Schrittmotortreiber verbunden. Wenn der Wert von IN1 auf einen hohen Niveau eingestellt ist, leuchtet A auf; Schalter SA ist eingeschaltet und der Schrittmotor dreht sich einen Schritt. Der ähnliche Fall wiederholt sich immer weiter. Geben Sie dem Schrittmotor daher einfach eine bestimmte Zeitfolge, er dreht sich Schritt für Schritt. Der ULN2003 wird hier verwendet, um bestimmte Zeitabläufe für den Schrittmotor bereitzustellen.


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

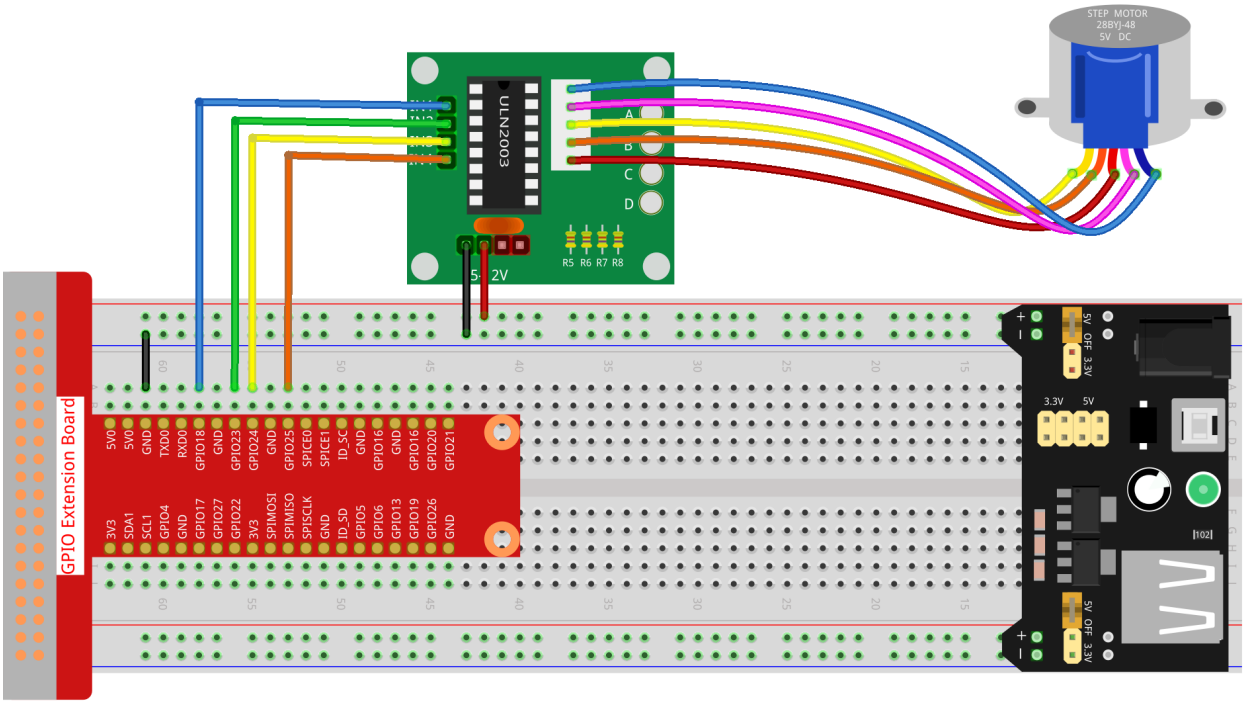
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Während der Code läuft, dreht sich der Schrittmotor entsprechend Ihrer Eingabe ‚a' oder ‚c' im oder gegen den Uhrzeigersinn.

Code

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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;
        }
        else
        {
            printf("input error, please try again!\n");
            delay(100);
        }
    }
    while(1)
    {
        rotary(direction);
    }
}

void main(void)
{
    if (wiringPiSetup() == -1)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

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

rolePerMinute: Umdrehungen pro Minute sollte die Drehzahl des in diesem Kit verwendeten Schrittmotors 0 bis 17 betragen.

stepPerRevolution: Die Anzahl der Schritte für jede Umdrehung und der in diesem Kit verwendete Schrittmotor benötigen 2048 Schritte pro Umdrehung.

stepSpeed: Die für jeden Schritt verwendete Zeit. In `main()` weisen wir ihnen die folgenden Werte zu: (60000000 / rolePerMinute) / stepPerRevolution (60.000.000 us = 1 Minute)

```
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);
    }
}
```

Die Funktion `loop()` ist grob in zwei Teile unterteilt (zwischen zwei `while(1)`):

Der erste Teil besteht darin, den Schlüsselwert zu erhalten. Wenn ‚a' oder ‚c' erhalten wird, verlassen Sie die Schleife und stoppen Sie die Eingabe.

Der zweite Teil ruft rotary(direction) auf, um den Schrittmotor laufen zu lassen.

```
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);
        }
    }
}
```

Damit sich der Schrittmotor im Uhrzeigersinn dreht, sollte der Füllstandsstatus von motorPin in der folgenden Tabelle angezeigt werden:

	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

Daher wird das potentielle Schreiben von MotorPin unter Verwendung einer zweischichtigen for-Schleife implementiert.

In Schritt 1 ist j = 0, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (10011001 & 00001000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (10011001 & 00000100 = 0)

`motorPin[2]` wird in der niedrigen Ebene geschrieben (10011001 & 00000010 = 0)

`motorPin[3]` wird in der hohen Ebene geschrieben (10011001 & 00000001 = 1)

In Schritt 2 ist j = 1, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (01001100 & 00001000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (01001100 & 00000100 = 1)

usw.

Damit sich der Schrittmotor gegen den Uhrzeigersinn dreht, wird der Füllstandsstatus von motorPin in der folgenden Tabelle angezeigt.

	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

In Schritt 1 ist j = 0, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (10011001 & 10000000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (10011001 & 01000000 = 0)

In Schritt 2 ist j = 1, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (00110010 & 10000000 = 0)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (00110010 & 01000000 = 0)

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner des Codes.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

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

Während der Code läuft, dreht sich der Schrittmotor abhängig von Ihrer Eingabe ‚a' oder ‚a' im oder gegen den Uhrzeigersinn.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

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

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

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

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

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

def destroy():
    GPIO.cleanup()

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

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

rolePerMinute: Umdrehungen pro Minute sollte die Drehzahl des in diesem Kit verwendeten Schrittmotors 0 bis 17 betragen.

stepPerRevolution: Die Anzahl der Schritte für jede Umdrehung und der in diesem Kit verwendete Schrittmotor benötigen 2048 Schritte pro Umdrehung.

stepSpeed: Die Zeit, die für jeden Schritt verwendet wird, und wir weisen ihnen die Werte zu: (60 / rolePerMinute) / stepPerRevolution (60s = 1minute).

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

Die Funktion `loop()` ist grob in zwei Teile unterteilt (in zwei Teilen, `while(1)`):

Der erste Teil besteht darin, den Schlüsselwert zu erhalten. Wenn ‚a' oder ‚c' erhalten wird, verlassen Sie die Schleife und stoppen Sie die Eingabe.

Der zweite Teil ruft `rotary(direction)` auf, um den Schrittmotor laufen zu lassen.

```
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)
```

Um den Schrittmotor im Uhrzeigersinn drehen zu lassen, wird der Niveau status von motorPin in der folgenden Tabelle angezeigt:

	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

Daher wird das potentielle Schreiben von MotorPin unter Verwendung einer zweischichtigen for-Schleife implementiert.

In Schritt 1 ist j = 0, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (10011001 & 00001000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (10011001 & 00000100 = 0)

`motorPin[2]` wird in der niedrigen Ebene geschrieben (10011001 & 00000010 = 0)

`motorPin[3]` wird in der hohen Ebene geschrieben (10011001 & 00000001 = 1)

In Schritt 2 ist j = 1, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (01001100 & 00001000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (01001100 & 00000100 = 1)

usw

Um den Schrittmotor gegen den Uhrzeigersinn drehen zu lassen, wird der Füllstandsstatus von motorPin in der folgenden Tabelle angezeigt.

	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

In Schritt 1 ist j = 0, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (10011001 & 10000000 = 1)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (10011001 & 01000000 = 0)

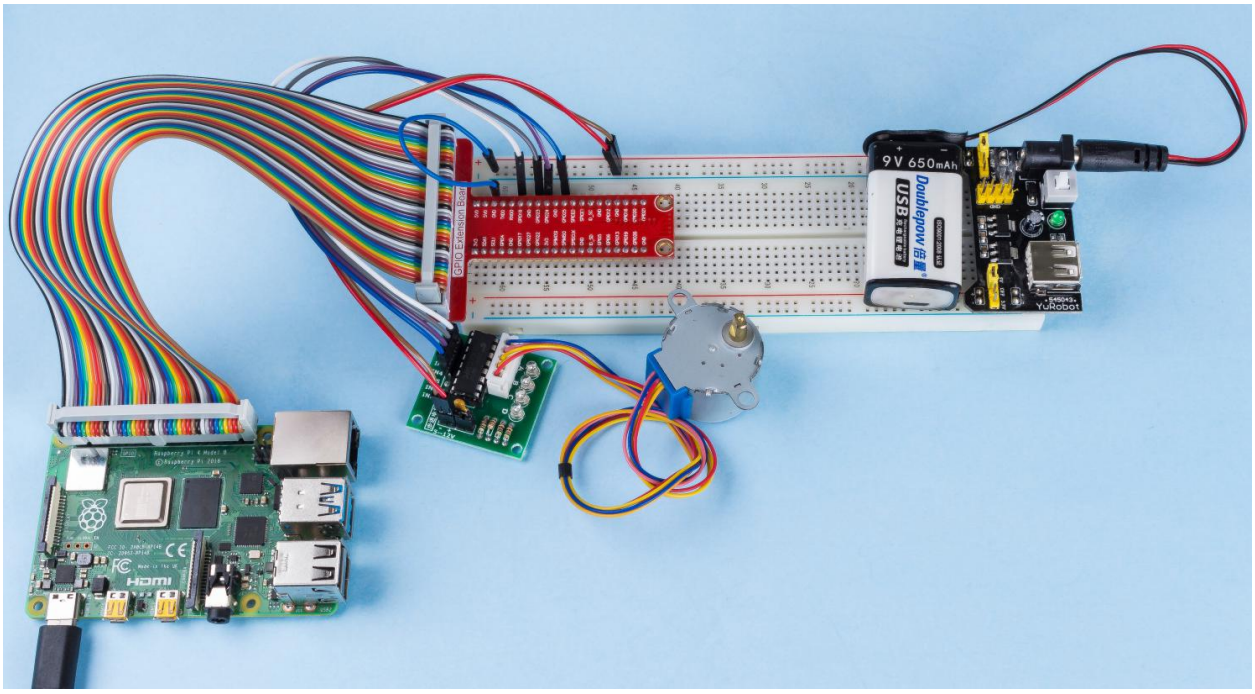
In Schritt 2 ist j = 1, i = 0 ~ 4.

`motorPin[0]` wird in der hohen Ebene geschrieben (00110010 & 10000000 = 0)

`motorPin[1]` wird auf dem niedrigen Niveau geschrieben (00110010 & 01000000 = 0)

usw.

Phänomen Bild

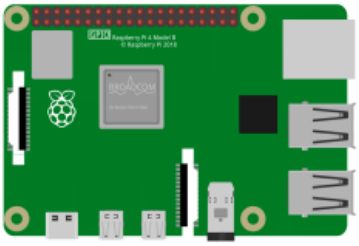
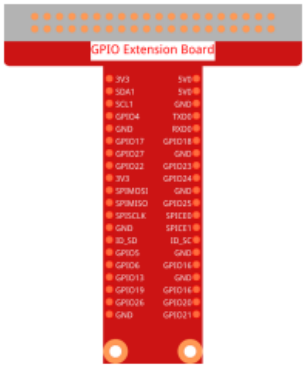


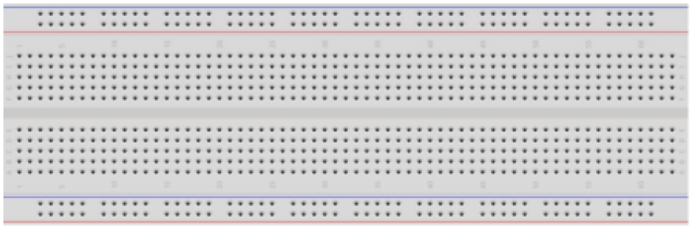
1.7.13 1.3.4 Relais

Einführung

In dieser Lektion lernen wir, ein Relais zu verwenden. Es ist eine der am häufigsten verwendeten Komponenten im automatischen Steuerungssystem. Wenn die Spannung, der Strom, die Temperatur, der Druck usw. den vorgegebenen Wert erreichen, überschreiten oder unterschreiten, wird das Relais den Stromkreis anschließen oder unterbrechen, um das Gerät zu steuern und zu schützen.

Komponenten

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Relay
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
 </td><td>1 * 1N4007 Diode
 </td></tr>
<tr><td>1 * LED

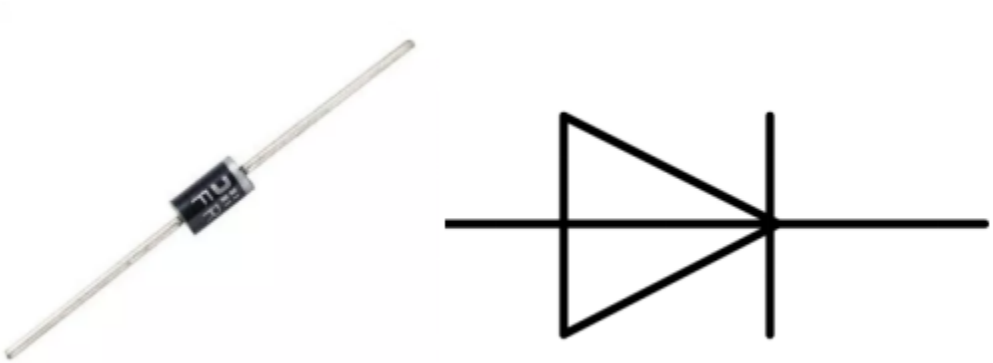
1 * S8050 NPN Transistor
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * Breadboard
  </td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Resistor(220Ω)

1 * Resistor 1KΩ
 </td></tr>
</table>

Prinzip

Diode

Eine Diode ist eine zweipolige Komponente in der Elektronik mit einem unidirektionalen Stromfluss. Es bietet einen geringen Widerstand in Richtung des Stromflusses und einen hohen Widerstand in der entgegengesetzten Richtung. Dioden werden meistens verwendet, um Schäden an Bauteilen zu vermeiden, insbesondere aufgrund elektromotorischer Kraft in Schaltkreisen, die normalerweise polarisiert sind.



Die beiden Anschlüsse einer Diode sind polarisiert, wobei das positive Ende als Anode und das negative Ende als Kathode bezeichnet wird. Die Kathode besteht üblicherweise aus Silber oder hat ein Farbband. Die Steuerung der Stromflussrichtung ist eines der Hauptmerkmale von Dioden - der Strom in einer Diode fließt von Anode zu Kathode. Das Verhalten einer Diode ähnelt dem Verhalten eines Rückschlagventils. Eine der wichtigsten Eigenschaften einer Diode ist die nichtlineare Stromspannung. Wenn eine höhere Spannung an die Anode angeschlossen ist, fließt Strom von Anode zu Kathode, und der Prozess wird als Vorwärtsvorspannung bezeichnet. Wenn jedoch die höhere Spannung an die Kathode angeschlossen ist, leitet die Diode keine Elektrizität, und der Prozess wird als Sperrvorspannung bezeichnet.

Relais

Wie wir vielleicht wissen, ist Relais ein Gerät, das verwendet wird, um eine Verbindung zwischen zwei oder mehr Punkten oder Geräten als Reaktion auf das angelegte Eingangssignal herzustellen. Mit anderen Worten, Relais stellen eine Isolation zwischen der Steuerung und dem Gerät bereit, da Geräte sowohl mit Wechselstrom als auch mit Gleichstrom arbeiten können. Sie empfangen jedoch Signale von einem Mikrocontroller, der mit Gleichstrom arbeitet, weshalb ein Relais erforderlich ist, um die Lücke zu schließen. Das Relais ist äußerst nützlich, wenn Sie eine große Menge an Strom oder Spannung mit einem kleinen elektrischen Signal steuern müssen.

Jedes Relais besteht aus 5 Teilen:

Elektromagnet - Er besteht aus einem Eisenkern, der durch eine Drahtspule gewickelt ist. Wenn Elektrizität durchgelassen wird, wird sie magnetisch. Daher wird es Elektromagnet genannt.

Anker - Der bewegliche Magnetstreifen wird als Anker bezeichnet. Wenn Strom durch sie fließt, wird die Spule erregt, wodurch ein Magnetfeld erzeugt wird, das verwendet wird, um die normalerweise offenen (N/O) oder normalerweise geschlossenen (N/C) Punkte zu erzeugen oder zu brechen. Der Anker kann sowohl mit Gleichstrom (DC) als auch mit Wechselstrom (AC) bewegt werden.

Feder - Wenn am Elektromagneten keine Ströme durch die Spule fließen, zieht die Feder den Anker weg, sodass der Stromkreis nicht abgeschlossen werden kann.

Satz elektrischer **Kontakte** - Es gibt zwei Kontaktpunkte:

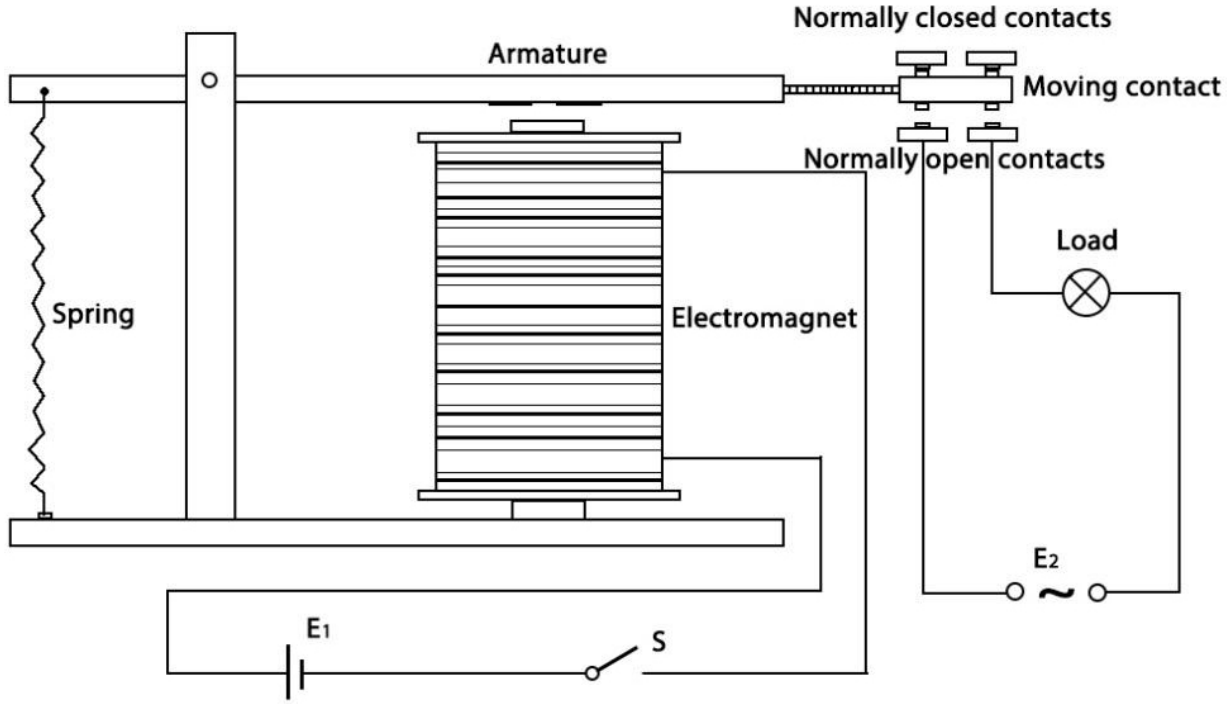
- Normalerweise offen - verbunden, wenn das Relais aktiviert ist, und getrennt, wenn es inaktiv ist.

- Normalerweise geschlossen - nicht angeschlossen, wenn das Relais aktiviert ist, und angeschlossen, wenn es inaktiv ist. aktiviert ist,

Molded frame - Die Relais sind zum Schutz mit Kunststoff bedeckt.

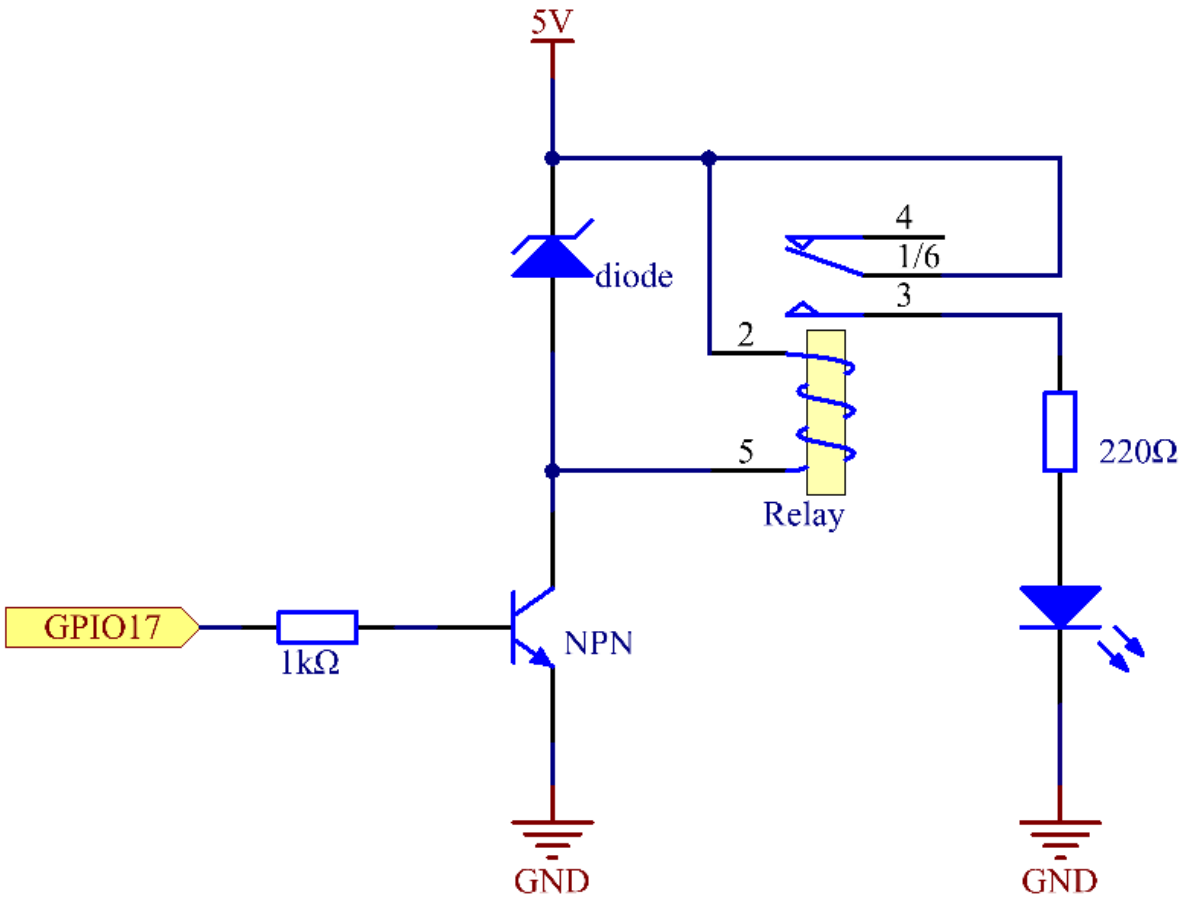
Funktionieren des Relais

Das Funktionsprinzip des Relais ist einfach. Wenn das Relais mit Strom versorgt wird, fließen Ströme durch die Steuerspule. Infolgedessen beginnt der Elektromagnet zu erregen. Dann wird der Anker von der Spule angezogen, bewegliche Kontakt zusammezihen, wodurch eine Verbindung mit den normalerweise offenen Kontakten hergestellt wird. Der Stromkreis mit der Last wird also erregt. Ein Unterbrechen des Stromkreises wäre dann ein ähnlicher Fall, da der bewegliche Kontakt unter der Kraft der Feder zu den normalerweise geschlossenen Kontakten hochgezogen wird. Auf diese Weise kann das Ein- und Ausschalten des Relais den Zustand eines Lastkreises steuern.



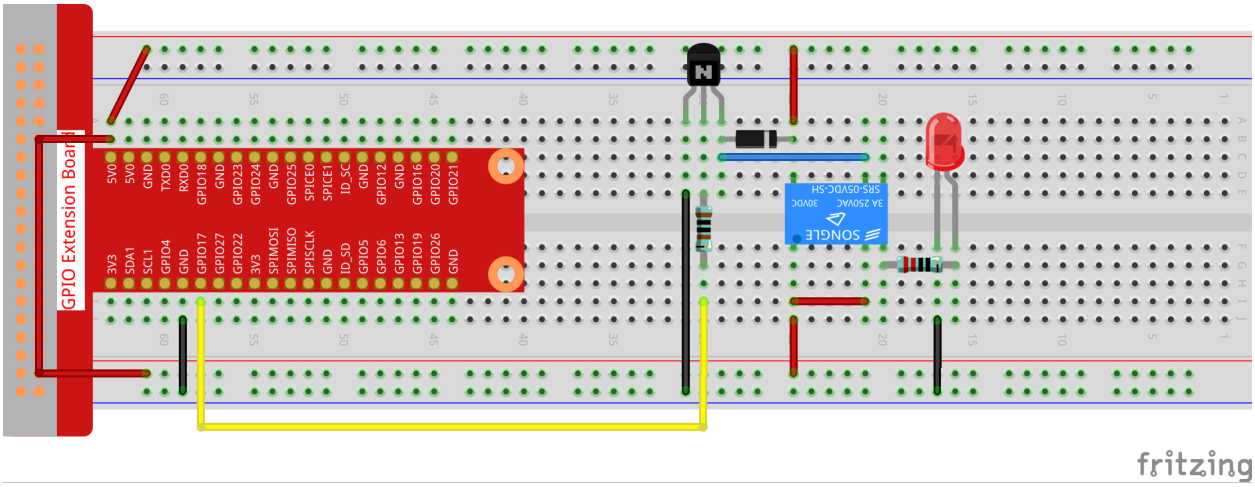
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Kodedatei.

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

Schritt 3: Kompilieren Sie den Code.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, leuchtet die LED auf. Außerdem können Sie ein Ticktock hören, das durch das Unterbrechen des normalerweise geschlossenen Kontakts und das Schließen des normalerweise offenen Kontakts verursacht wird.

Code

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

int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(RelayPin, OUTPUT);   //set GPIO17(GPIO0) output
    while(1){
        // Tick
        printf("Relay Open......\n");
        delay(100);
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        printf("......Relay Close\n");
        delay(100);
        digitalWrite(RelayPin, HIGH);
        delay(1000);
    }

    return 0;
}
```

Code Erklärung

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

Stellen Sie den I/O -Anschluss auf einen niedrigen Niveau (0V) ein, damit der Transistor nicht erregt und die Spule nicht mit Strom versorgt wird. Es gibt keine elektromagnetische Kraft, daher öffnet sich das Relais und die LED leuchtet nicht.

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

Stellen Sie den I/O -Anschluss auf einen hohen Niveau (5V) ein, um den Transistor mit Strom zu versorgen. Die Spule des Relais wird mit Strom versorgt und erzeugt elektromagnetische Kraft. Das Relais schließt, die LED leuchtet auf.

Für Python-Benutzer

Schritt 2: Öffnen Sie die Codedatei.

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

Schritt 3: Ausführen.

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

Während der Code läuft, leuchtet die LED. Außerdem können Sie ein Ticktock hören, das durch das Unterbrechen des normalerweise geschlossenen Kontakts und das Schließen des normalerweise offenen Kontakts verursacht wird.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # and initial level to High(3.3v)
    GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():
    while True:
        print ('Relay open...')
        # Tick
        GPIO.output(relayPin, GPIO.LOW)
        time.sleep(1)
        print ('...Relay close')
        # Tock
        GPIO.output(relayPin, GPIO.HIGH)
        time.sleep(1)

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

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

Code Erklärung

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

Stellen Sie die Pins des Transistors auf einen niedrigen Niveau ein, damit das Relais geöffnet wird. Die LED leuchtet nicht.

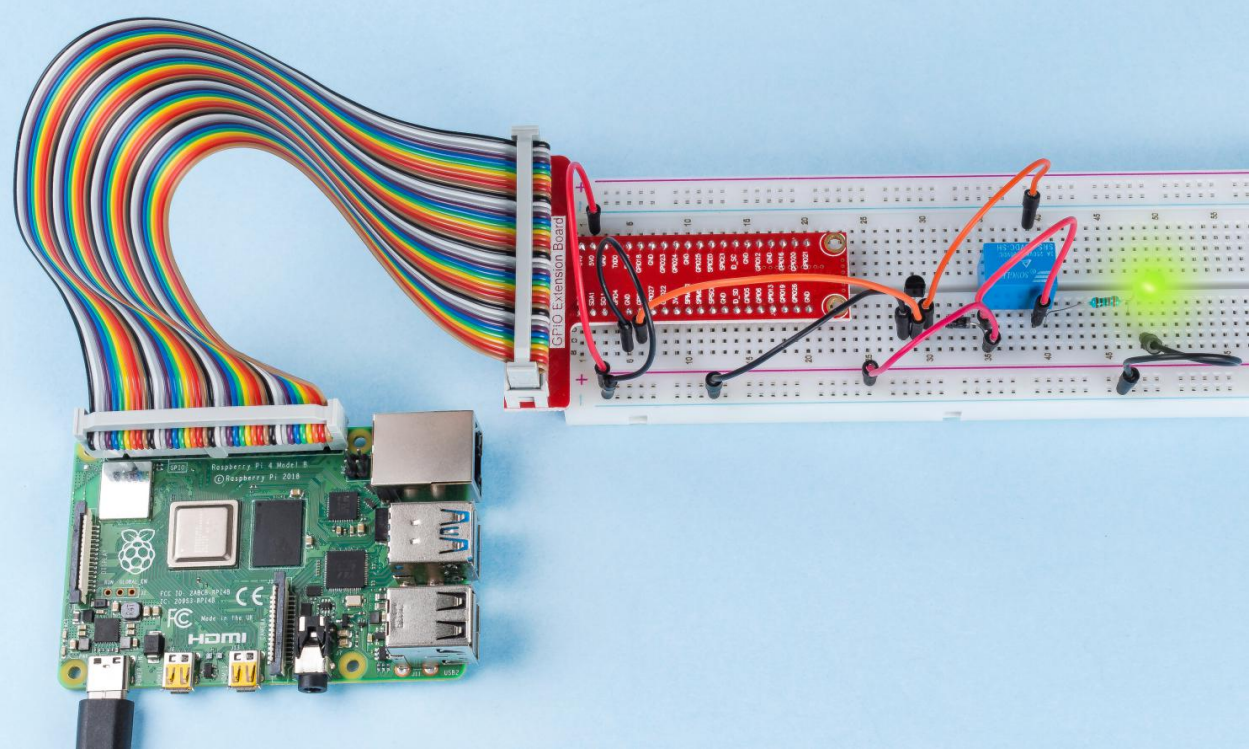
```
time.sleep(1)
```

1 Sekunde warten.

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

Stellen Sie die Pins des Transistors auf einen niedrigen Niveau ein, um das Relais zu betätigen. Die LED leuchtet auf.

Phänomen Bild



1.8 2 Eingabe (nicht für Pi 5)

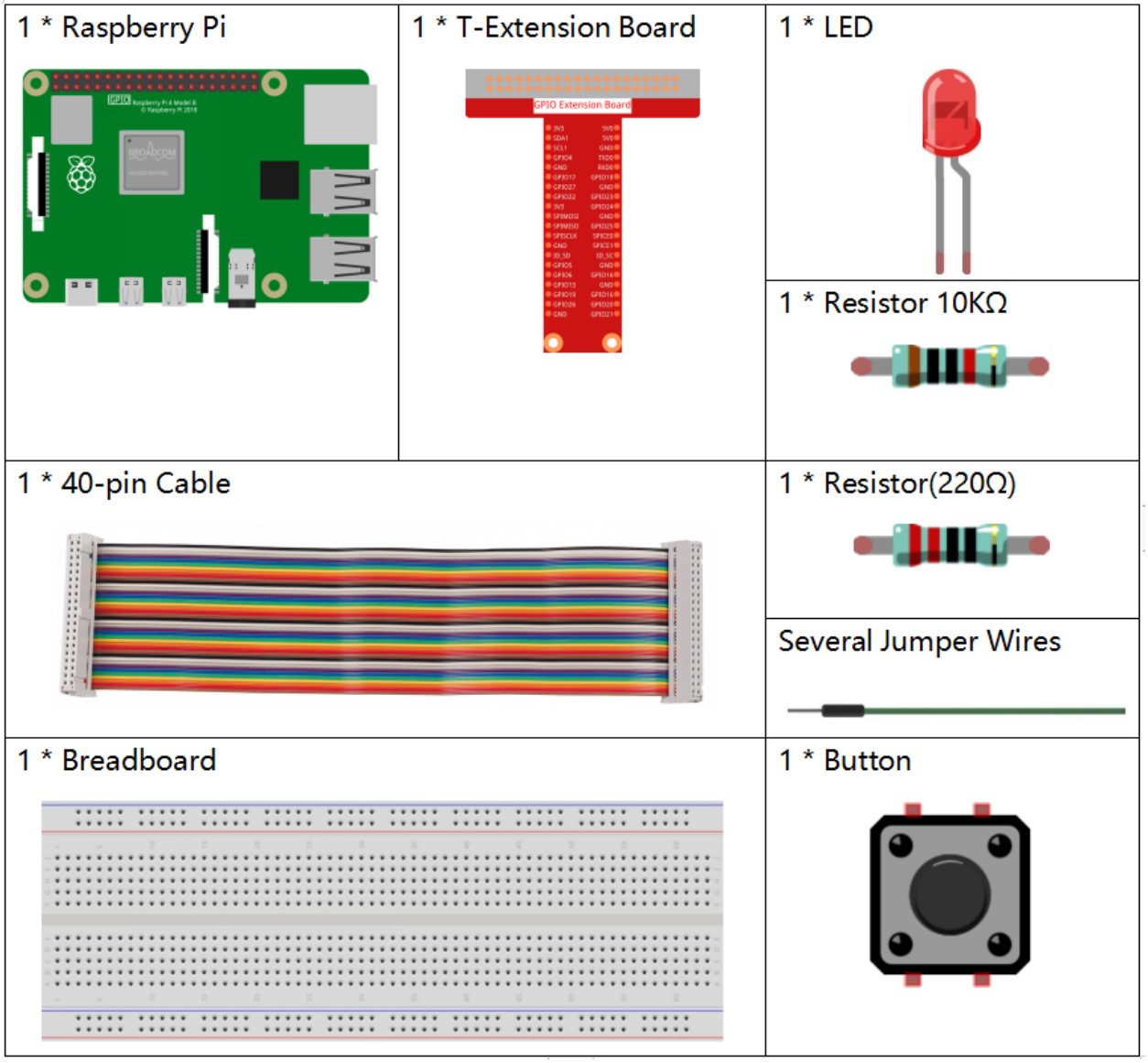
2.1 Steuerungen

1.8.1 2.1.1 Taste

Einführung

In dieser Lektion lernen wir, wie Sie die LED mit einer Taste ein- oder ausschalten.

Komponenten

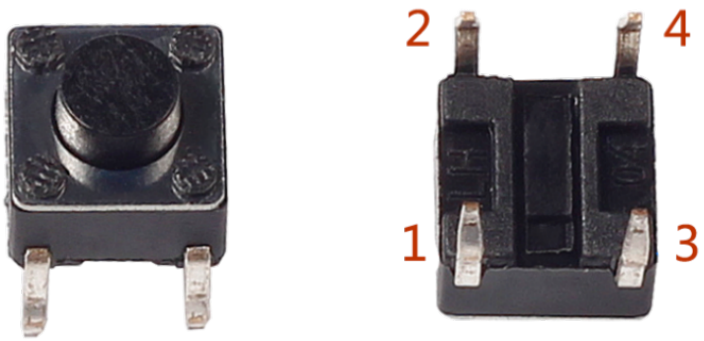


Prinzip

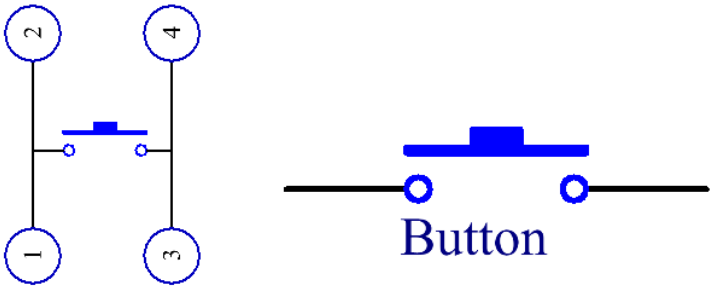
Taste

Die Taste ist eine übliche Komponente zur Steuerung elektronischer Geräte. Es wird normalerweise als Schalter zum Anschließen oder Unterbrechen von Stromkreisen verwendet. Obwohl die Tasten in verschiedenen Größen und Formen erhältlich sind, wird hier ein 6-mm-Miniknopf verwendet, wie in den folgenden Bildern gezeigt.

Zwei Pins auf der linken Seite sind verbunden, und der eine auf der rechten Seite ähnelt dem linken, der unten gezeigt wird:



Das unten gezeigte Symbol wird normalerweise verwendet, um eine Schaltfläche in Schaltkreisen darzustellen.



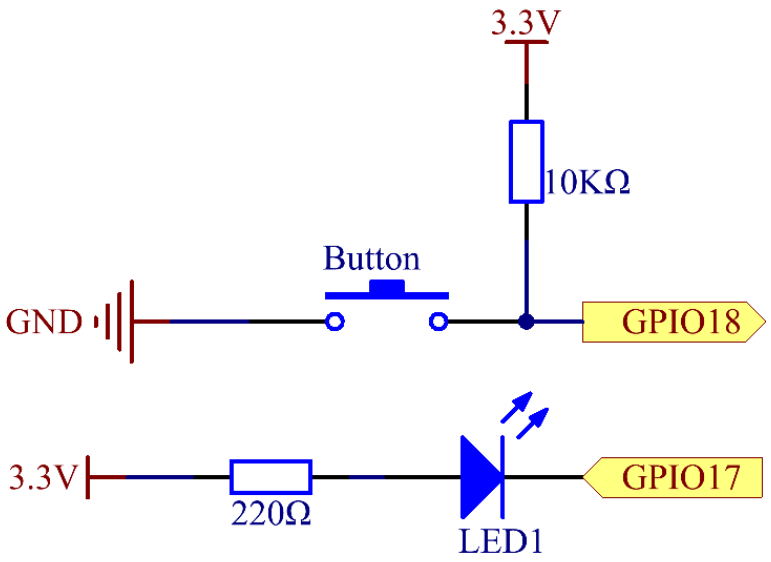
Wenn die Taste gedrückt wird, werden die 4 Pins verbunden, wodurch der Stromkreis geschlossen wird.

Schematische Darstellung

Verwenden Sie eine normalerweise geöffnete Taste als Eingang für Raspberry Pi. Die Verbindung ist in der folgenden schematischen Darstellung dargestellt. Wenn die Taste gedrückt wird, wird der GPIO18 auf einen niedrigen Niveau (0V) eingestellt. Wir können den Zustand des GPIO18 durch Programmierung erkennen. Das heißt, wenn der GPIO18 auf einen niedrigen Niveau wechselt, bedeutet dies, dass die Taste gedrückt wird. Sie können den entsprechenden Kode ausführen, wenn die Taste gedrückt wird, und dann leuchtet die LED auf.

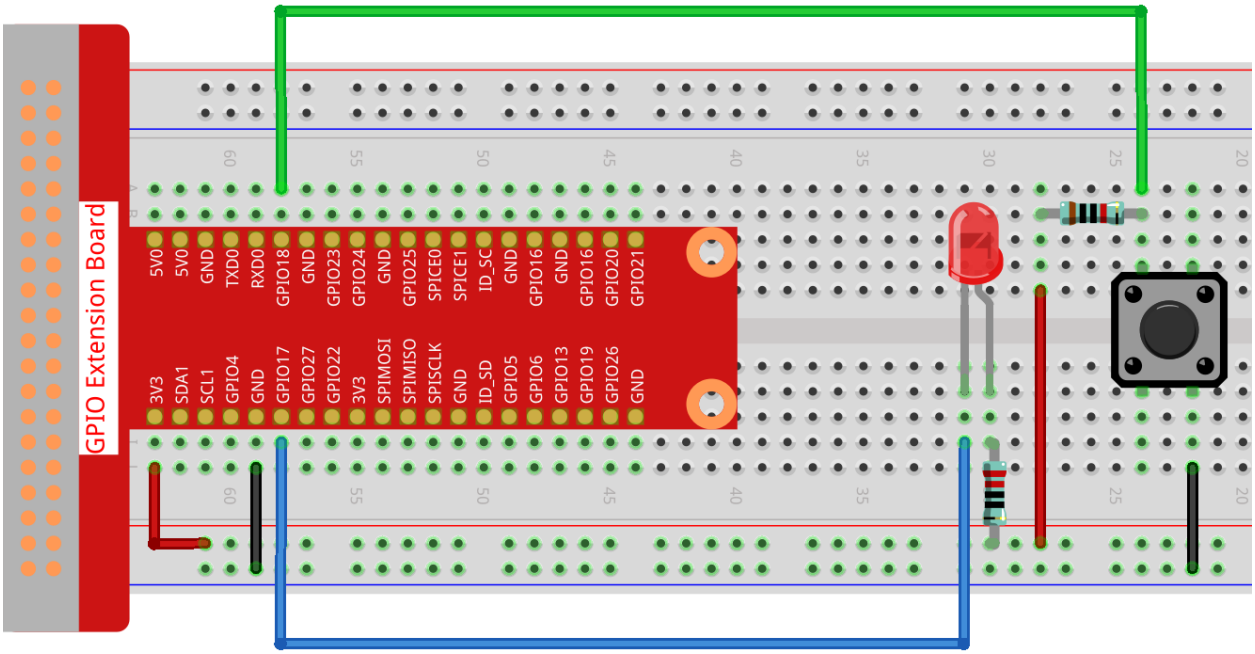
Bemerkung: Der längere Pin der LED ist die Anode und der kürzere ist die Kathode.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Kodedatei.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.1/
```

Bemerkung: Wechseln Sie in diesem Experiment in den Pfad der Kode mit `cd`.

Schritt 3: Kompilieren Sie den Code.

```
gcc 2.1.1_Button.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem der Code ausgeführt wurde, drücken Sie die Taste, die LED leuchtet auf; andernfalls erlischt.

Code

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

#define LedPin      0
#define ButtonPin   1

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

    pinMode(LedPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    digitalWrite(LedPin, HIGH);

    while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 0){
            // Led on
            digitalWrite(LedPin, LOW);
        //  printf("...LED on\n");
        }
        else{
            // Led off
            digitalWrite(LedPin, HIGH);
        //  printf("LED off...\n");
        }
    }
    return 0;
}
```

Code Erklärung

```
#define LedPin      0
```

Der Pin GPIO17 in der T_Extension-Karte entspricht dem GPIO0 im wiringPi.

```
#define ButtonPin   1
```

ButtonPin ist mit GPIO1 verbunden.

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

Stellen Sie LedPin als Ausgabe ein, um ihm einen Wert zuzuweisen.

```
pinMode(ButtonPin, INPUT);
```

Legen Sie ButtonPin als Eingabe fest, um den Wert von ButtonPin zu lesen.

```
while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 0){
            // Led on
            digitalWrite(LedPin, LOW);
        //  printf("...LED on\n");
        }
        else{
            // Led off
            digitalWrite(LedPin, HIGH);
        //  printf("LED off...\n");
        }
    }
```

`if (digitalRead (ButtonPin)) == 0` : Überprüfen Sie, ob die Taste gedrückt wurde. Führen Sie `digitalWrite(LedPin, LOW)` aus, wenn die Taste gedrückt wird, um die LED zu leuchten.

Für Python-Sprachbenutzer

Schritt 2: Öffnen Sie die Kodedatei.

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

Schritt 3: Führen Sie die Kode aus.

```
sudo python3 2.1.1_Button.py
```

Drücken Sie nun die Taste und die LED leuchtet auf. Drücken Sie die Taste erneut und die LED erlischt. Gleichzeitig wird der Status der LED auf dem Bildschirm gedruckt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
    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.
    except KeyboardInterrupt:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        destroy()
```

Code Erklärung

```
LedPin = 17
```

Stellen Sie GPIO17 als LED-Pin ein.

```
BtnPin = 18
```

Stellen Sie GPIO18 als Tasten Pin ein.

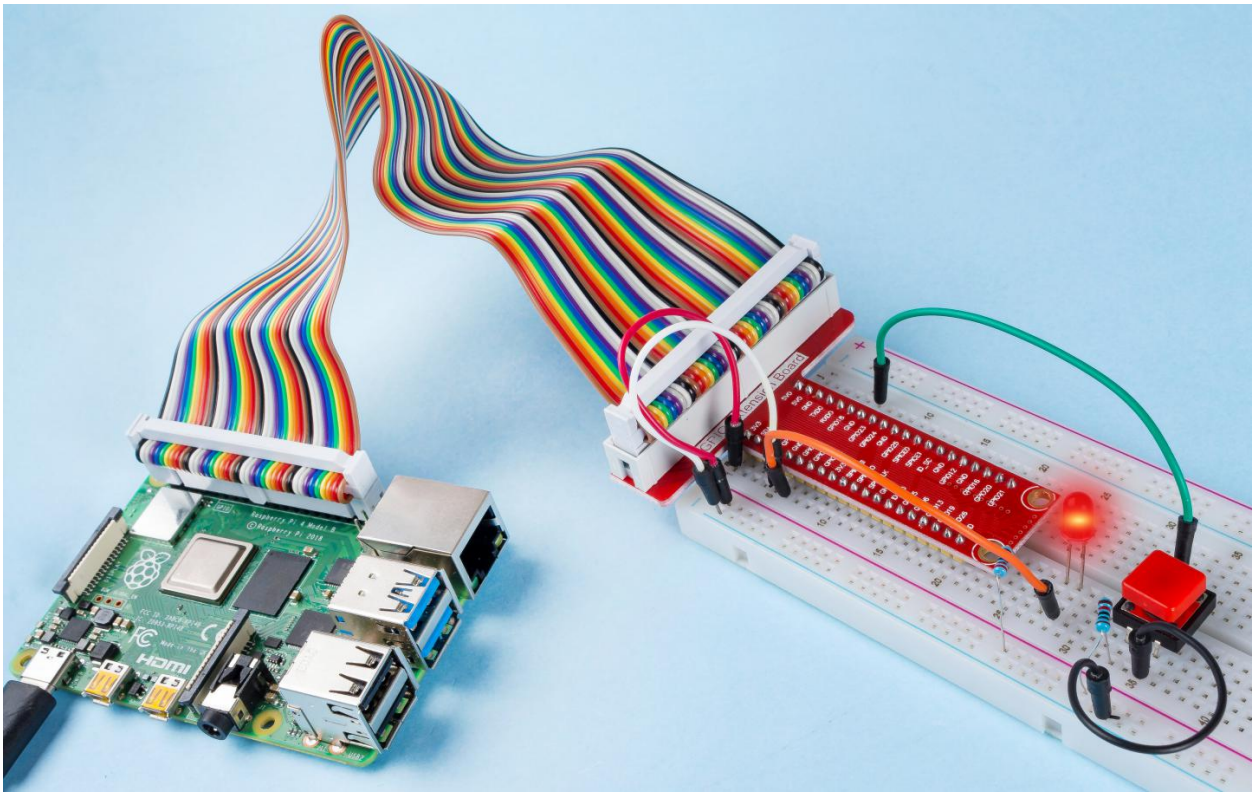
```
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

Richten Sie eine Fallerkennung für `BtnPin` ein. Wenn sich der Wert von `BtnPin` von einem hohen auf einen niedrigen Wert ändert, bedeutet dies, dass die Taste gedrückt wird. Der nächste Schritt ist das Aufrufen der Funktion `swled` .

```
def swLed(ev=None):
global Led_status
# Switch led status(on-->off; off-->on)
Led_status = not Led_status
GPIO.output(LedPin, Led_status)
```

Definieren Sie eine Rückruffunktion als Tastenrückruf. Wenn die Taste beim ersten Mal gedrückt wird und die Bedingung, nicht Led_status, falsch ist, wird die Funktion `GPIO.output()` aufgerufen, um die LED zu beleuchten. Wenn die Taste erneut gedrückt wird, wird der Status der LED von falsch in wahr umgewandelt, sodass die LED erlischt.

Phänomen Bild

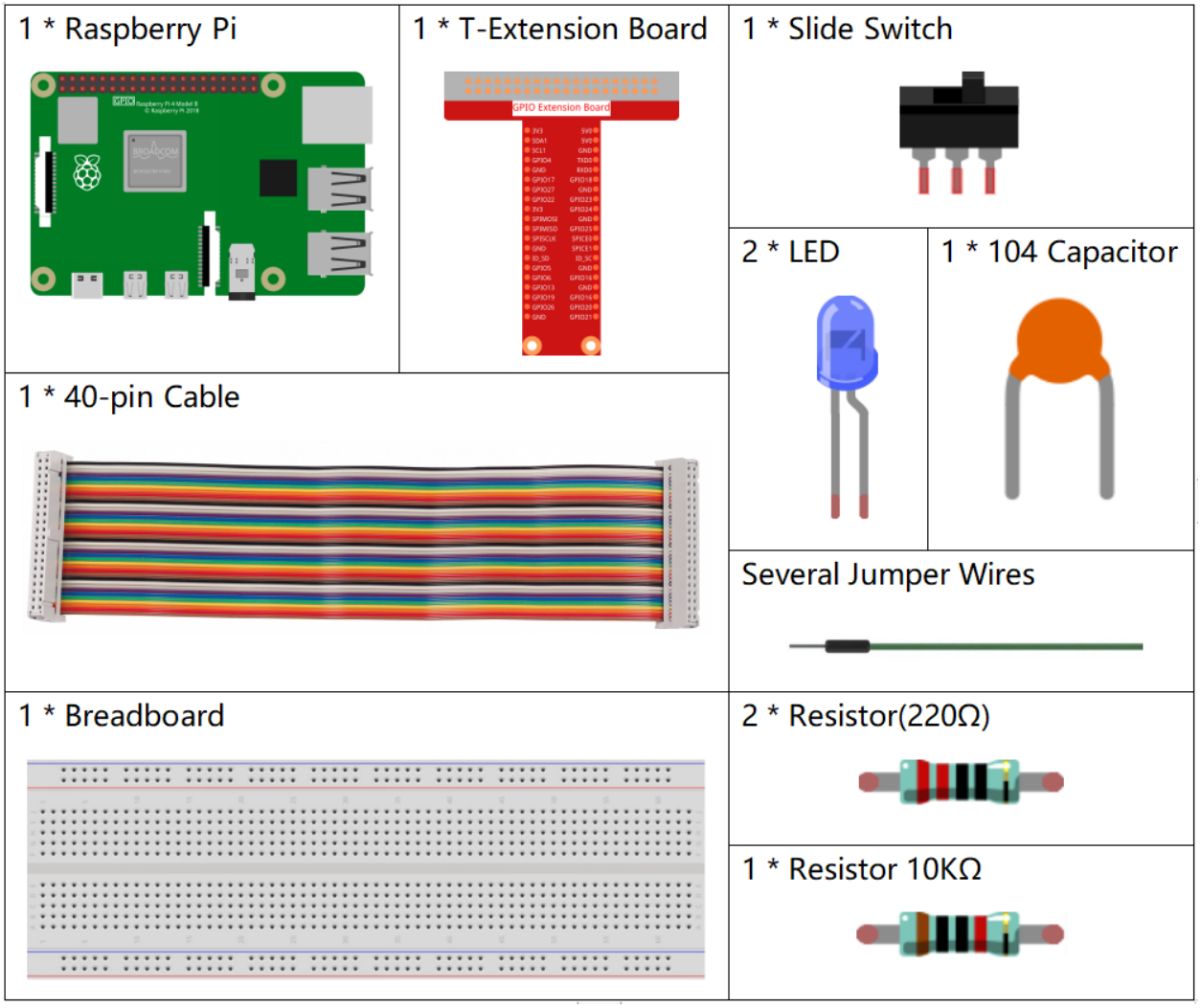


1.8.2 2.1.2 Schiebeschalter

Einführung

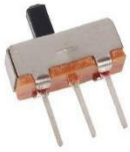
In dieser Lektion lernen wir, wie man einen Schiebeschalter benutzt. Normalerweise wird der Schiebeschalter als Netzschalter auf die Leiterplatte gelötet, aber hier müssen wir ihn in das Steckbrett einsetzen, damit er möglicherweise nicht festgezogen wird. Und wir verwenden es auf dem Steckbrett, um seine Funktion zu zeigen.

Komponenten



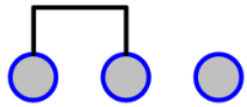
Prinzip

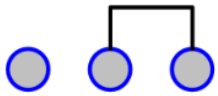
Schiebeschalter



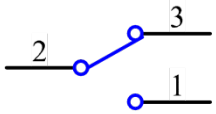
Ein Schiebeschalter dient, wie der Name schon sagt, dazu, die Schaltleiste zu schieben, um den Stromkreis anzuschließen oder zu unterbrechen, und weitere Schaltkreise. Die am häufigsten verwendeten Typen sind SPDT, SPTT, DPDT, DPTT usw. Der Schiebeschalter wird üblicherweise in Niederspannungsschaltungen verwendet. Es hat die Merkmale Flexibilität und Stabilität und ist in elektrischen Instrumenten und elektrischem Spielzeug weit verbreitet.

So funktioniert es: Stellen Sie den mittleren Pin als festen Pin ein. Wenn Sie den Schieber nach links ziehen, sind die beiden Pins links verbunden. Wenn Sie es nach rechts ziehen, sind die beiden Pins rechts verbunden. Somit funktioniert es als Schalter, der Schaltkreise verbindet oder trennt. Siehe die folgende Abbildung:





Das Schaltungssymbol des Schiebeschalters ist wie folgt dargestellt. Der Pin2 in der Abbildung bezieht sich auf den mittleren Pin.



Kondensator

Der Kondensator ist eine Komponente, die Energie in Form von elektrischer Ladung speichern oder eine Potentialdifferenz (statische Spannung) zwischen ihren Platten erzeugen kann, ähnlich wie eine kleine wiederaufladbare Batterie.

Standardkapazitätseinheiten

Microfarad (F) 1F = 1/1,000,000 = 0.000001 = F

Nanofarad (nF) 1nF = 1/1,000,000,000 = 0.000000001 = F

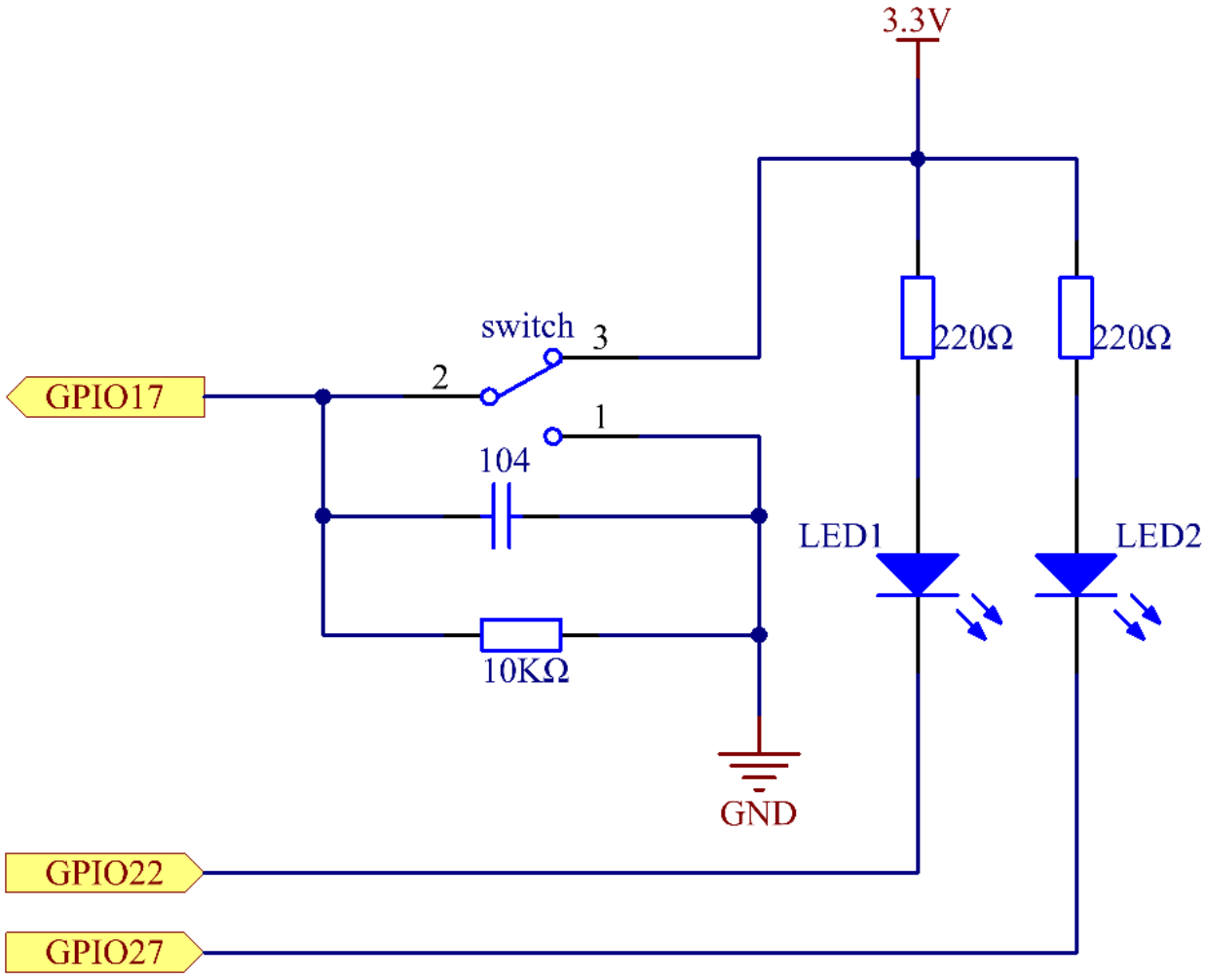
Picofarad (pF) 1pF = 1/1,000,000,000,000 = 0.000000000001 = F

Bemerkung: Hier verwenden wir **104 Kondensatoren (10 x 104PF)**. Genau wie beim Widerstandsring helfen die Nummer auf den Kondensatoren beim Ablesen der Werte, die auf der Platine montiert wurden. Die ersten beiden Ziffern stellen den Wert dar und die letzte Ziffer der Nummer bedeutet den Multiplikator. Somit repräsentiert 104 eine Potenz von 10 x 10 zu 4 (in pF) gleich wie 100 nF.

Schematische Darstellung

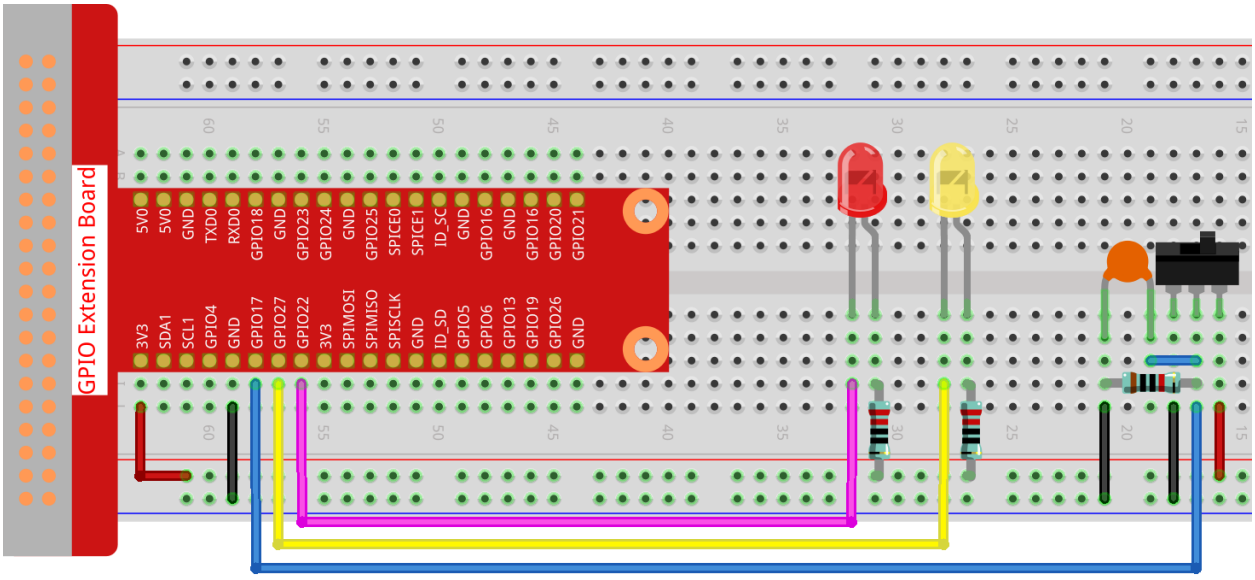
Verbinden Sie den mittleren Pin des Schiebeschalters mit GPIO17 und zwei LEDs mit Pin GPIO22 bzw. GPIO27. Wenn Sie dann an der Folie ziehen, leuchten die beiden LEDs abwechselnd auf.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Kompilieren.

```
gcc 2.1.2_Slider.c -lwiringPi
```

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Während der Code ausgeführt wird, schalten Sie den Schalter links ein, und die gelbe LED leuchtet auf. rechts leuchtet das rote Licht auf.

Code

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

int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(slidePin, INPUT);
    pinMode(led1, OUTPUT);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
}
```

Code Erklärung

```
if(digitalRead(slidePin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("LED1 on\n");
    }
```

Wenn der Schieber nach rechts gezogen wird, sind der mittlere und der rechte Stift verbunden. Der Raspberry Pi liest einen hohen Niveau am mittleren Pin, sodass die LED1 an und die LED2 aus ist.

```
if(digitalRead(slidePin) == 0){
            digitalWrite(led2, LOW);
            digitalWrite(led1, HIGH);
            printf(".....LED2 on\n");
        }
```

Wenn der Schieber nach links gezogen wird, sind der mittlere und der linke Pins verbunden. Der Raspberry Pi zeigt einen niedrigen Wert an, sodass die LED2 leuchtet und die LED1 aus ist

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Ausführen.

```
sudo python3 2.1.2_Slider.py
```

Während der Code ausgeführt wird, schalten Sie den Schalter links ein, und die gelbe LED leuchtet auf. rechts leuchtet das rote Licht auf.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
    # 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()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

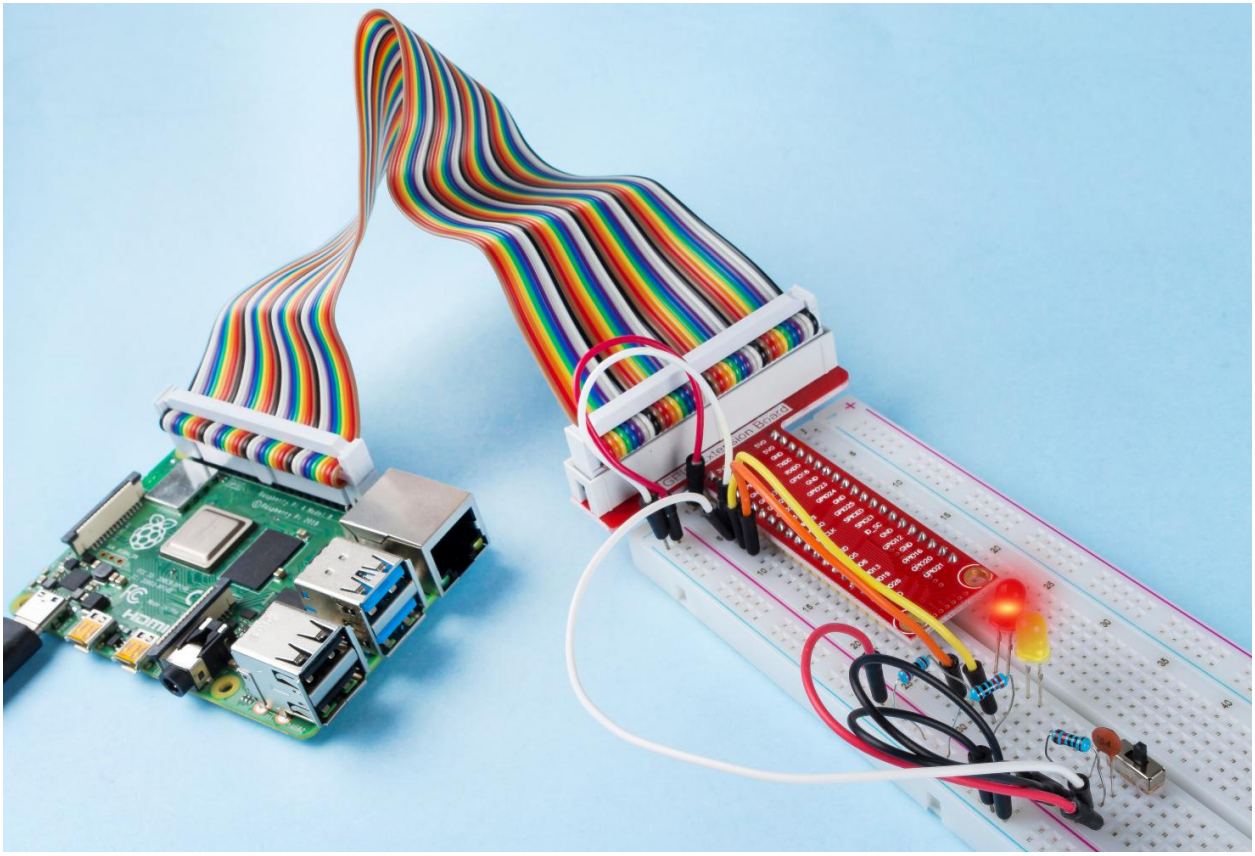
```
if GPIO.input(slidePin) == 1:
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.HIGH)
```

Wenn der Schieber nach rechts gezogen wird, sind der mittlere und der rechte Pins verbunden. Der Raspberry Pi liest einen hohen Niveau am mittleren Pin, sodass die LED1 an und die LED2 aus ist.

```
if GPIO.input(slidePin) == 0:
    GPIO.output(led2Pin, GPIO.LOW)
    GPIO.output(led1Pin, GPIO.HIGH)
```

Wenn der Schieber nach links gezogen wird, sind der mittlere und der linke Pins verbunden. Der Raspberry Pi zeigt einen niedrigen Wert an, sodass die LED2 leuchtet und die LED1 aus ist.

Phänomen Bild

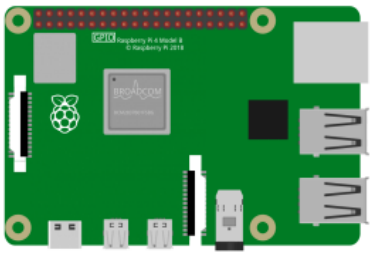
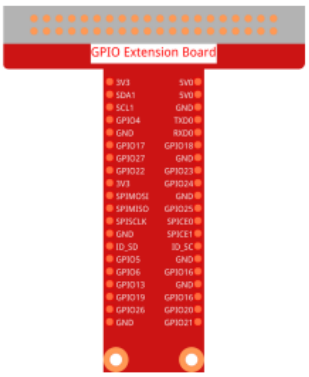
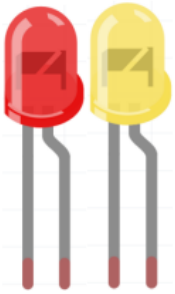
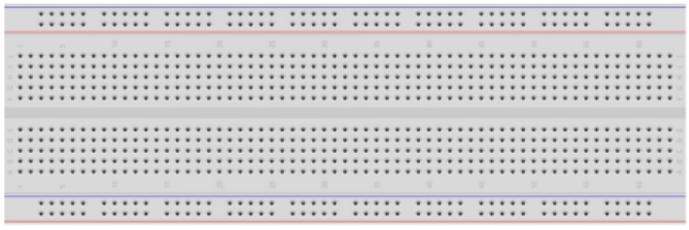


1.8.3 2.1.3 Neigungsschalter

Einführung

Dies ist ein Kugelkippschalter mit einer Metallkugel im Inneren. Es wird verwendet, um Neigungen eines kleinen Winkels zu erfassen.

Komponenten

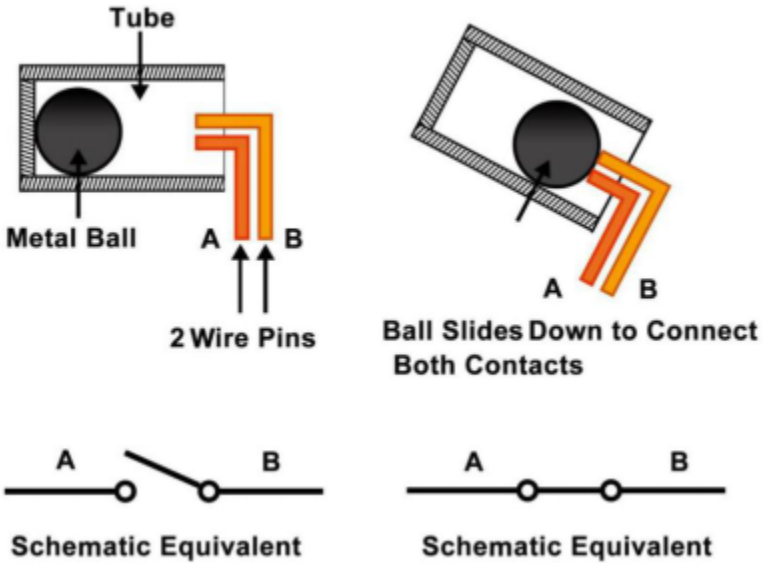
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Tilt Switch
</td><td>2 * LED
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td colspan="2">Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td colspan="2">2 * Resistor(220Ω)

1 * Resistor 10KΩ
 </td></tr>
</table>

Prinzip

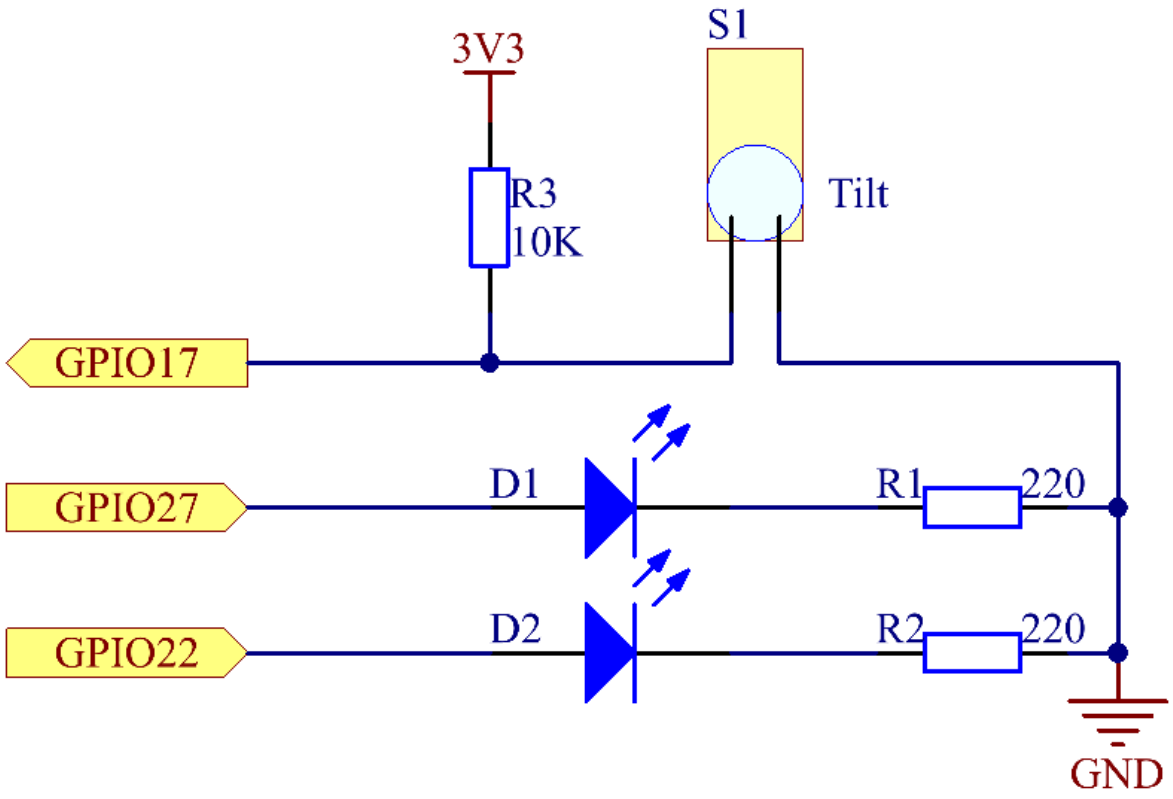
Neigung

Das Prinzip ist sehr einfach. Wenn der Schalter in einem bestimmten Winkel gekippt wird, rollt die Kugel im Inneren nach unten und berührt die beiden Kontakte, die mit den Pins außen verbunden sind, wodurch Schaltkreise ausgelöst werden. Andernfalls bleibt der Ball von den Kontakten fern und unterbricht so die Stromkreise.



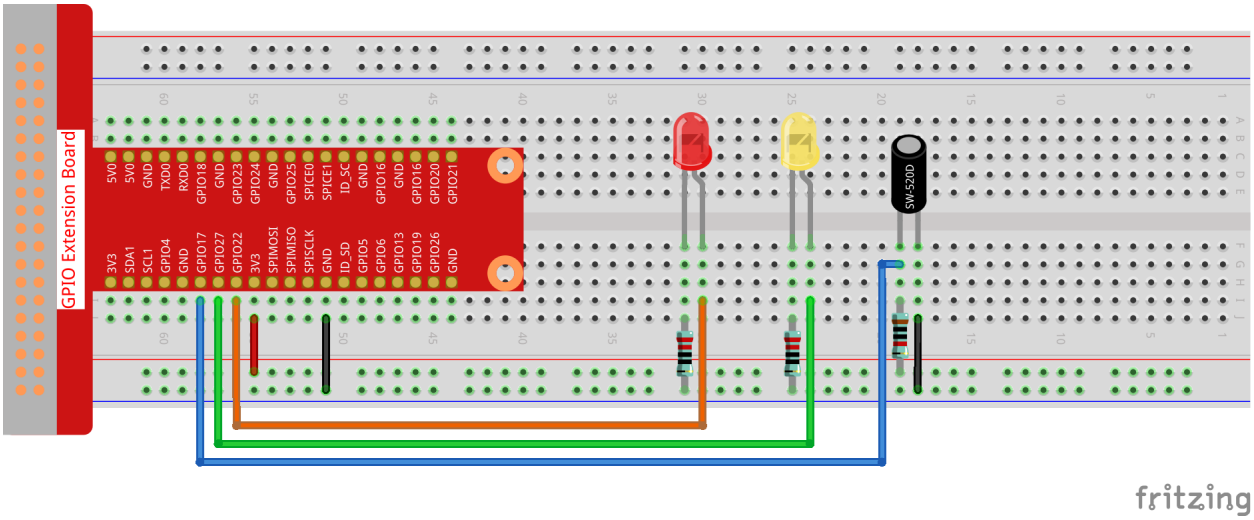
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.3/
```

Schritt 3: Kompilieren.

```
gcc 2.1.3_Tilt.c -lwiringPi
```

Schritt 4: Ausführen.

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

Wenn Sie die Neigung horizontal platzieren, leuchtet die grüne LED auf. Wenn Sie es kippen, „Neigung!“ wird auf dem Bildschirm gedruckt und die rote LED leuchtet auf. Stellen Sie es wieder horizontal auf und die grüne LED leuchtet wieder auf.

Code

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

#define TiltPin     0
#define Gpin        2
#define Rpin        3

void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == "RED")
    {
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}

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

    pinMode(TiltPin, INPUT);
    LED("GREEN");
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
}
```

Code Erklärung

```
void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == "RED")
    {
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}
```

Definieren Sie eine Funktions- `LED()` , um die beiden LEDs ein- oder auszuschalten. Wenn die Parameterfarbe ROT ist, leuchtet die rote LED auf. Wenn die Parameterfarbe GRÜN ist, leuchtet die grüne LED ebenfalls auf.

```
while(1){
        if(0 == digitalRead(TiltPin)){
            delay(10);
            if(0 == digitalRead(TiltPin)){
                LED("RED");
                printf("Tilt!\n");
            }
        }
        else if(1 == digitalRead(TiltPin)){
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            delay(10);
            if(1 == digitalRead(TiltPin)){
                LED("GREEN");
            }
        }
    }
```

Wenn der Lesewert des Neigungsschalters 0 ist, bedeutet dies, dass der Neigungsschalter gekippt ist. Dann schreiben Sie den Parameter „ROT“ in die Funktions-LED, damit die rote LED aufleuchtet. Andernfalls leuchtet die grüne LED.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 2.1.3_Tilt.py
```

Wenn Sie die Neigung horizontal platzieren, leuchtet die grüne LED auf. Wenn Sie es kippen, „Neigung!“ wird auf dem Bildschirm gedruckt und die rote LED leuchtet auf. Stellen Sie es wieder horizontal auf und die grüne LED leuchtet auf.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    unchanged_count = 0
    last = -1
    data = []
    while True:
        current = GPIO.input(dhtPin)
        data.append(current)
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > MAX_UNCHANGE_COUNT:
                break

    state = STATE_INIT_PULL_DOWN

    lengths = []
    current_length = 0

    for current in data:
        current_length += 1

        if state == STATE_INIT_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_INIT_PULL_UP
            else:
                continue
        if state == STATE_INIT_PULL_UP:
            if current == GPIO.HIGH:
                state = STATE_DATA_FIRST_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_FIRST_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_DATA_PULL_UP
            else:
                continue
        if state == STATE_DATA_PULL_UP:
            if current == GPIO.HIGH:
                current_length = 0
                state = STATE_DATA_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_PULL_DOWN:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_PULL_UP
            else:
                continue
    if len(lengths) != 40:
        #print ("Data not good, skip")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        return False

    shortest_pull_up = min(lengths)
    longest_pull_up = max(lengths)
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []
    the_bytes = []
    byte = 0

    for length in lengths:
        bit = 0
        if length > halfway:
            bit = 1
        bits.append(bit)
    #print ("bits: %s, length: %d" % (bits, len(bits)))
    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0
    #print (the_bytes)
    checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False

    return the_bytes[0], the_bytes[2]

def main():

    while True:
        result = readDht11()
        if result:
            humidity, temperature = result
            print ("humidity: %s %%,  Temperature: %s C`" % (humidity, temperature))
        time.sleep(1)

def destroy():
    GPIO.cleanup()

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

Code Erklärung

```
GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)
```

Richten Sie eine Erkennung auf TiltPin und eine Rückruffunktion zur Erkennung ein.

```
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)
```

Definieren Sie eine Funktion `Led()` , um die beiden LEDs ein- oder auszuschalten. Wenn x = 0 ist, leuchtet die rote LED auf. Andernfalls leuchtet die grüne LED.

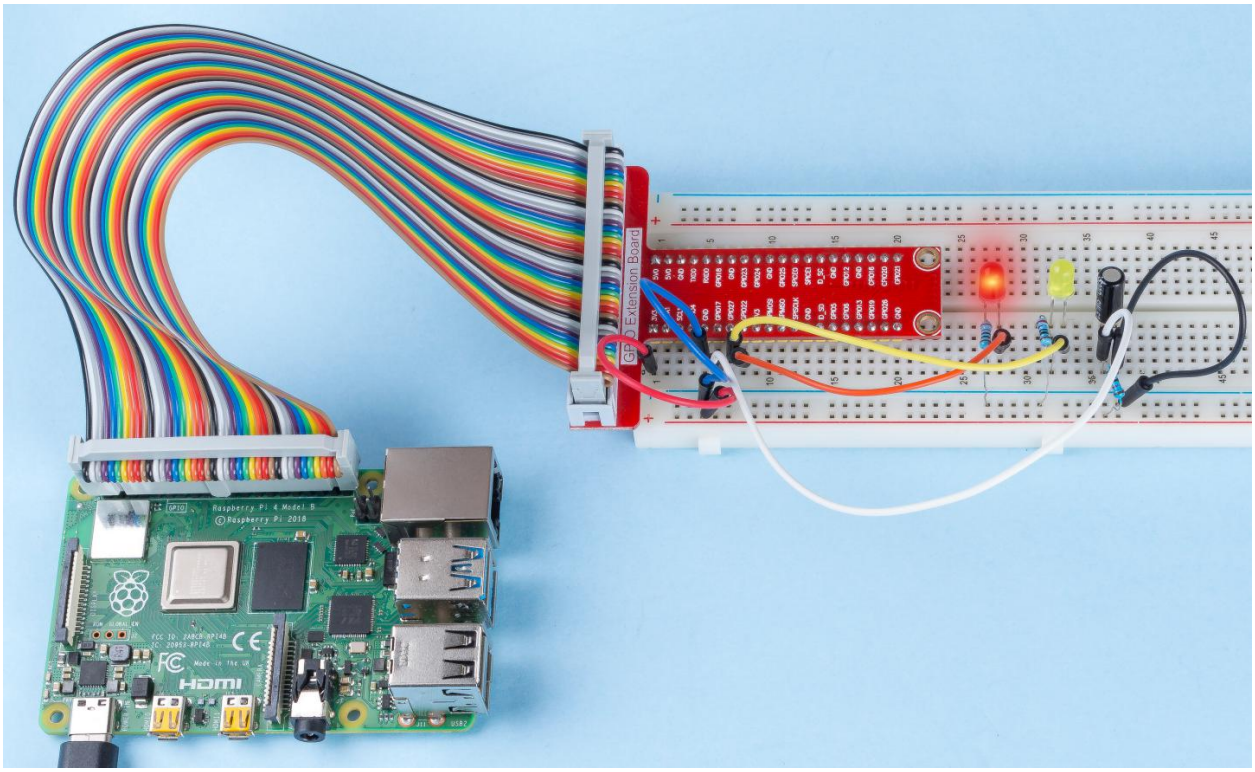
```
def Print(x):
    if x == 0:
        print ('    *************')
        print ('    *   Tilt!   *')
        print ('    *************')
```

Erstellen Sie eine Funktion, `Print()` , um die obigen Zeichen auf dem Bildschirm zu drucken.

```
def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))
```

Definieren Sie eine Rückruffunktion für den Neigungsrückruf. Holen Sie sich den Lesewert des Neigungsschalters, dann steuert die Funktion `LED()` das Ein- oder Ausschalten der beiden LEDs, abhängig vom Lesewert des Neigungsschalters.

Phänomen Bild

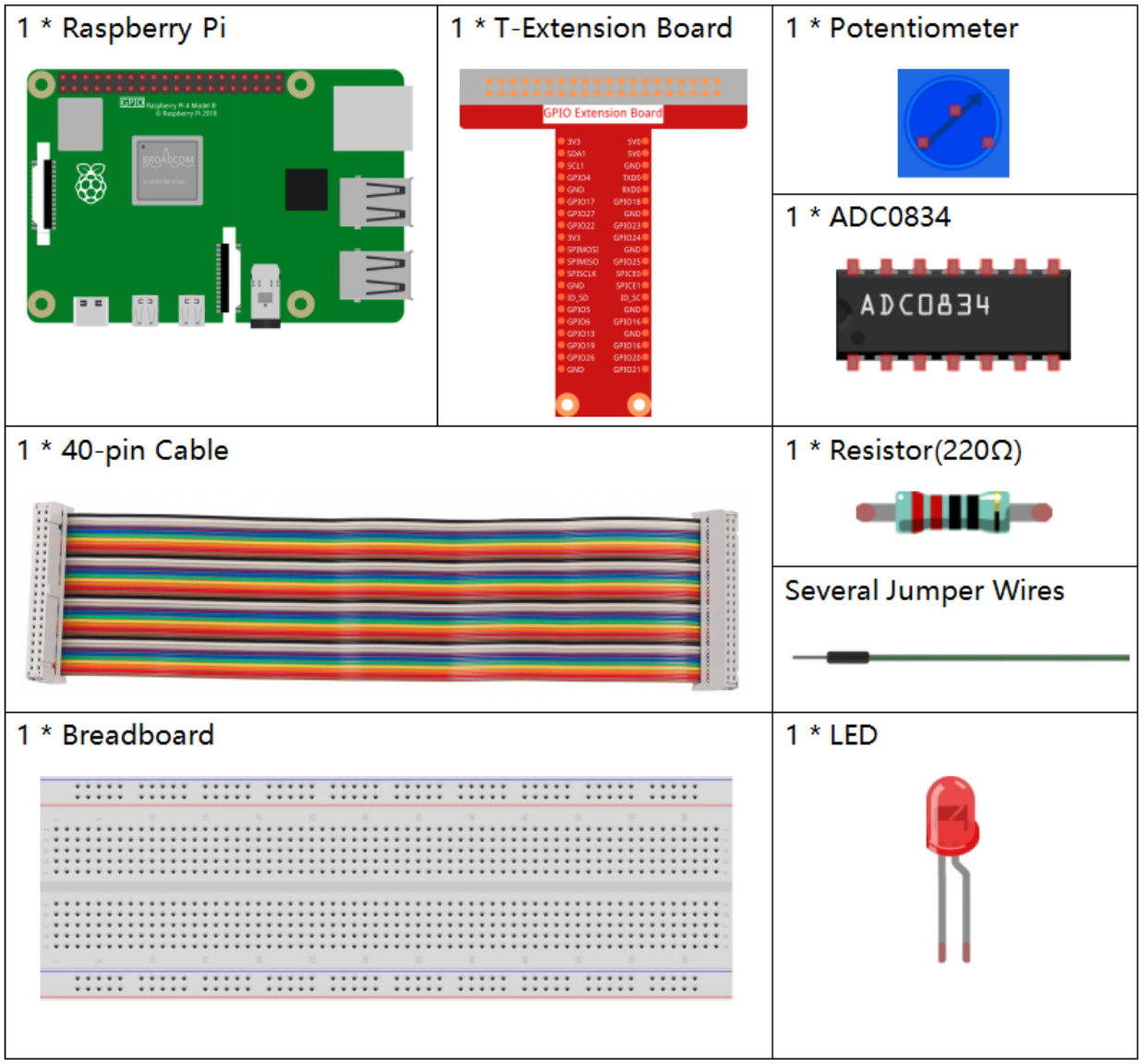


1.8.4 2.1.4 Potentiometer

Einführung

Die ADC-Funktion kann verwendet werden, um analoge Signale in digitale Signale umzuwandeln, und in diesem Experiment wird ADC0834 verwendet, um die Funktion zu erhalten, an der ADC beteiligt ist. Hier implementieren wir diesen Prozess mithilfe eines Potentiometers. Das Potentiometer ändert die physikalische Größe - Spannung, die von der ADC-Funktion umgewandelt wird.

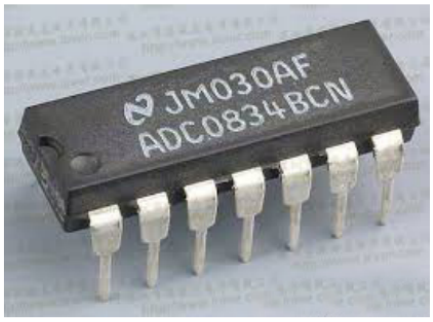
Prinzip

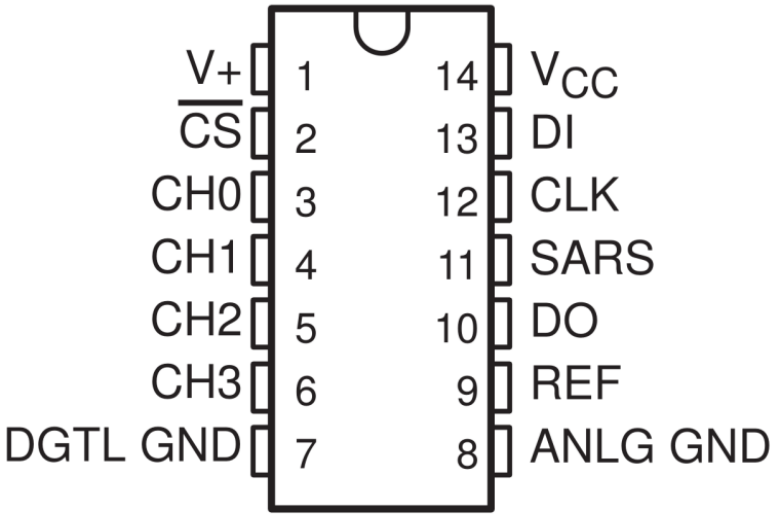


Prinzip

ADC0834

ADC0834 ist ein 8-Bit-der mit einem eingangskonfigurierbaren Mehrkanal-Multiplexer und einem seriellen Ein- / Ausgang ausgestattet ist. Der serielle Ein- / Ausgang ist für die Schnittstelle mit Standardschieberegistern oder Mikroprozessoren konfiguriert.



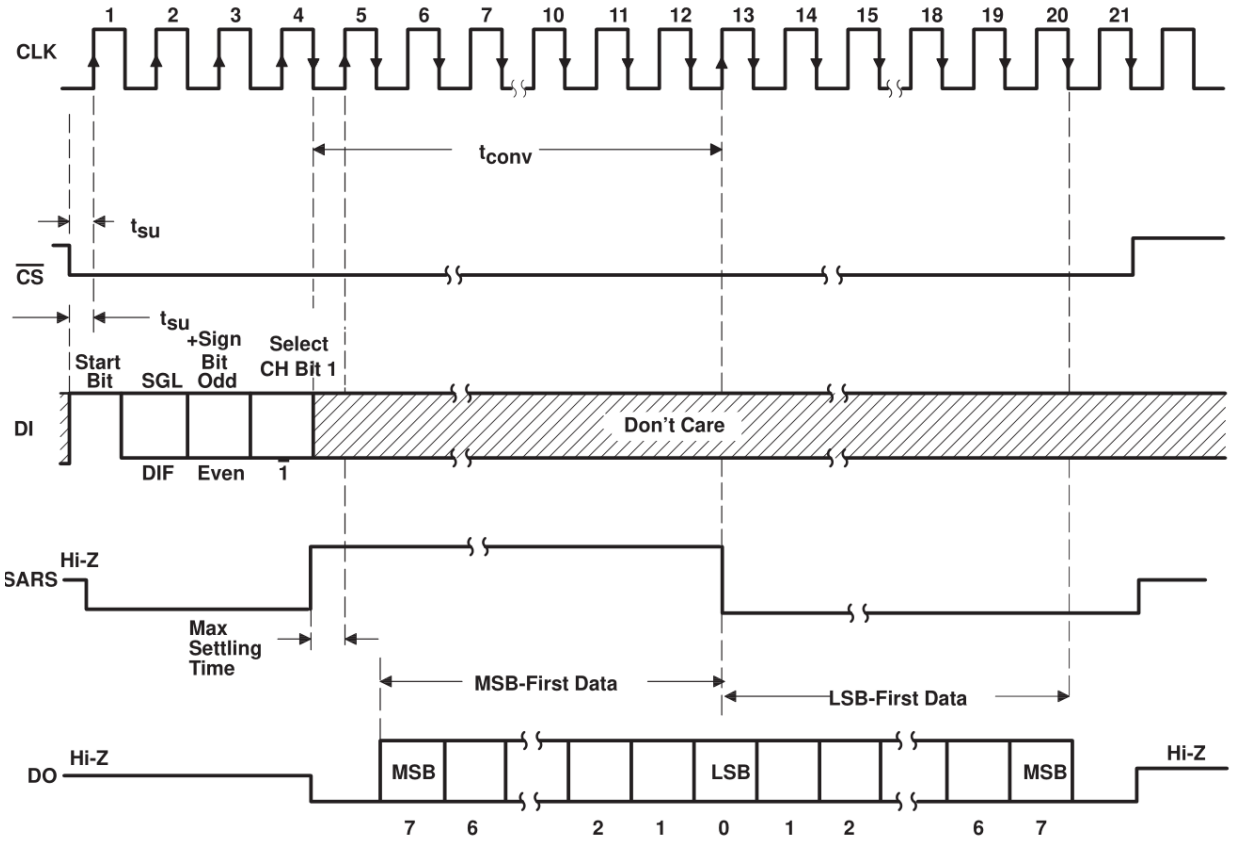


Betriebsablauf

Eine Konvertierung wird eingeleitet, indem CS auf niedrig gesetzt wird, wodurch alle Logikschaltungen aktiviert werden. CS muss für den gesamten Konvertierungsprozess niedrig gehalten werden. Ein Takteingang wird dann vom Prozessor empfangen. Bei jedem Übergang von niedrig nach hoch des Takteingangs werden die Daten auf DI in das Multiplexer-Adressschieberegister getaktet. Die erste Logik hoch am Eingang ist das Startbit. Auf das Startbit folgt ein 3- bis 4-Bit-Zuweisungswort. Bei jedem aufeinanderfolgenden Übergang von niedrig nach hoch des Takteingangs werden das Startbit und das Zuweisungswort durch das Schieberegister verschoben. Wenn das Startbit in den Startort des Multiplexerregisters verschoben wird, wird der Eingangskanal ausgewählt und die Umwandlung beginnt. Der SAR-Statu-Ausgang (SARS) geht auf High, um anzuzeigen, dass eine Konvertierung läuft, und DI in das Multiplexer-Schieberegister ist während der Konvertierungsdauer deaktiviert.

Ein Intervall von einer Taktperiode wird automatisch eingefügt, damit sich der ausgewählte Multiplexkanal einstellen kann. Der Datenausgang DO kommt aus dem hochohmigen Zustand heraus und liefert ein führendes Tief für diese Eintaktperiode der Multiplexer-Einschwingzeit. Der SAR-Komparator vergleicht aufeinanderfolgende Ausgänge von der Widerstandsleiter mit dem eingehenden analogen Signal. Der Komparatorausgang zeigt an, ob der Analogeingang größer oder kleiner als der Widerstandsleiterausgang ist. Während der Konvertierung werden die Konvertierungsdaten gleichzeitig vom DO-Ausgangspin ausgegeben, wobei das höchstwertige Bit (MSB) zuerst angezeigt wird.

Nach acht Taktperioden ist die Konvertierung abgeschlossen und der SARS-Ausgang wird niedrig. Schließlich werden die niedrigstwertigen Bit-First-Daten nach dem MSB-First-Datenstrom ausgegeben.



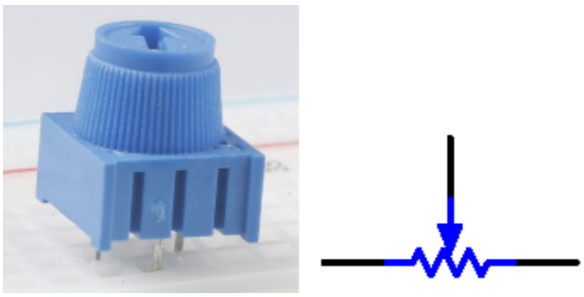
ADC0834 MUX ADDRESS CONTROL LOGIC TABLE

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

Potentiometer

Das Potentiometer ist auch eine Widerstandskomponente mit 3 Anschlüssen und sein Widerstandswert kann gemäß einigen regelmäßigen Abweichungen eingestellt werden. Das Potentiometer besteht normalerweise aus einem Widerstand und einer beweglichen Bürste. Wenn sich die Bürste entlang des Widerstands bewegt, gibt es abhängig von der Verschiebung einen bestimmten Widerstand oder eine bestimmte Spannung.



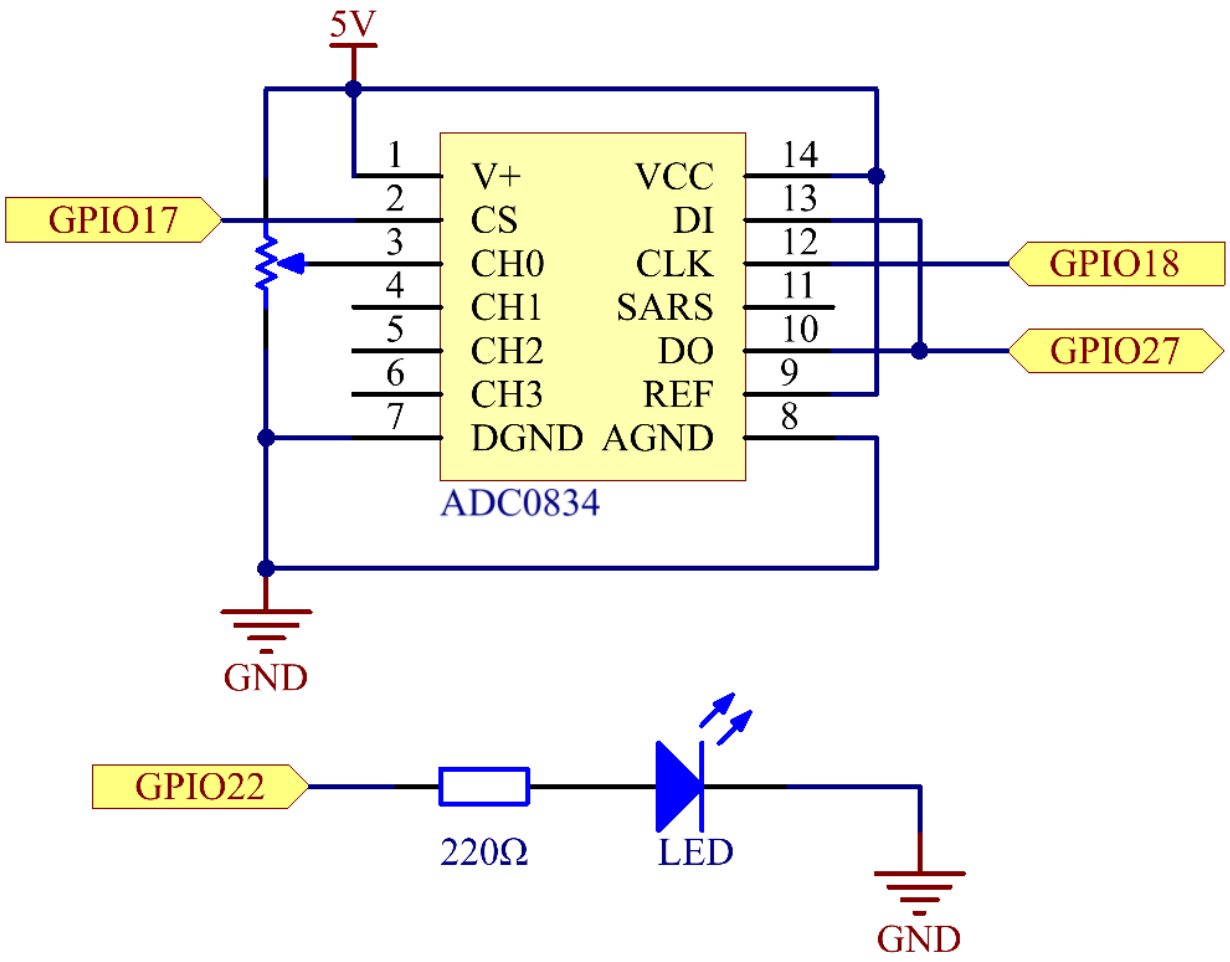
Die Funktionen des Potentiometers in der Schaltung sind wie folgt:

1. Dient als Spannungsteiler

Das Potentiometer ist ein stufenlos einstellbarer Widerstand. Wenn Sie die Welle oder den Schiebegriff des Potentiometers einstellen, gleitet der bewegliche Kontakt auf dem Widerstand. Zu diesem Zeitpunkt kann eine Spannung ausgegeben werden, die von der an das Potentiometer angelegten Spannung und dem Winkel abhängt, in den sich der bewegliche Arm gedreht hat, oder von der Entfernung, um die er sich bewegt.

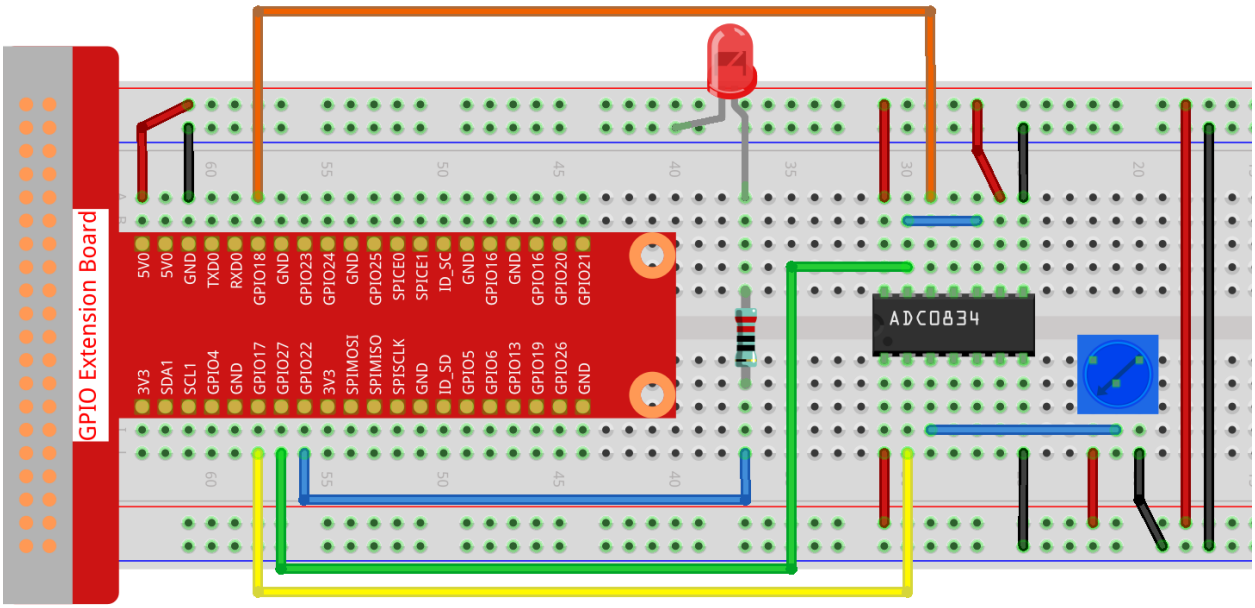
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Bemerkung: Bitte platzieren Sie den Chip unter Bezugnahme auf die entsprechende Position auf dem Bild. Beachten Sie, dass sich die Rillen auf dem Chip beim Platzieren links befinden sollten.

Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Kodedatei.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.4/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.1.4_Potentiometer.c -lwiringPi
```

Schritt 4: Ausführen.

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

Nachdem der Kode ausgeführt wurde, drehen Sie die Taste am Potentiometer. Die Intensität der LED ändert sich entsprechend.

Code

```
#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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
#define     ADC_DIO   2
#define     LedPin    3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
//Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}

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;
}
```

Code Erklärung

```
#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2
#define     LedPin    3
```

Definieren Sie CS, CLK, DIO von ADC0834 und verbinden Sie sie mit GPIO0, GPIO1 bzw. GPIO2. Schließen Sie dann die LED an GPIO3 an.

```
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);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
}
```

Es gibt eine Funktion von ADC0834, um die Analog-Digital-Wandlung zu erhalten. Der spezifische Workflow lautet wie folgt:

```
digitalWrite(ADC_CS, 0);
```

Stellen Sie CS auf einen niedrigen Wert ein und aktivieren Sie die AD-Konvertierung.

```
// Start bit
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

Wenn der Übergang von niedrig zu hoch des Takteingangs zum ersten Mal auftritt, setzen Sie DIO als Startbit auf 1. In den folgenden drei Schritten gibt es 3 Zuweisungswörter.

```
//Single End mode
digitalWrite(ADC_CLK,0);
igitalWrite(ADC_DIO,1);    delayMicroseconds(2);
gitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

Sobald der von niedrig zu hoch Übergang des Takteingangs zum zweiten Mal erfolgt, setzen Sie DIO auf 1 und wählen Sie den SGL-Modus.

```
// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

Einmal zum dritten Mal auftritt, wird der Wert von DIO durch die Variable **odd** gesteuert.

```
//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);
```

Wenn der Impuls von CLK zum vierten Mal von einem niedrigen auf einen hohen Niveau umgewandelt wird, wird der Wert von DIO durch die Variable **sel** gesteuert.

Unter der Bedingung, dass `channel=0` , `sel=0` , `odd=0` ist, lauten die Betriebsformeln bezüglich **sel** und **odd** wie folgt:

```
int sel = channel > 1 & 1;
int odd = channel & 1;
```

Wenn die Bedingung erfüllt ist, dass `channel=1` , `sel=0` , `odd=1` ist, lesen Sie bitte die folgende Adresssteuerungslogiktabelle. Hier wird CH1 gewählt und das Startbit wird in den Startort des Multiplexerregisters verschoben und die Umwandlung beginnt.

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);
```

Hier setzen Sie DIO zweimal auf 1, bitte ignorieren Sie es.

```
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);
    }
```

Stellen Sie in der ersten `for()` - Anweisung DIO in den Eingangsmodus, sobald der fünfte Impuls von CLK von einem hohen Niveau in einen niedrigen Niveau umgewandelt wurde. Dann beginnt die Konvertierung und der konvertierte Wert wird in der Variablen `dat1` gespeichert. Nach acht Taktperioden ist die Konvertierung abgeschlossen.

```
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);
    }
```

Geben Sie in der zweiten `for()` - Anweisung die konvertierten Werte nach weiteren acht Taktperioden über DO aus und speichern Sie sie in der Variablen `dat2` .

```
digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
```

`return(dat1==dat2) ? dat1 : 0` wird verwendet, um den während der Konvertierung erhaltenen Wert mit dem Ausgabewert zu vergleichen. Wenn sie gleich sind, geben Sie den Konvertierungswert dat1 aus. Andernfalls wird 0 ausgegeben. Hier ist der Workflow von ADC0834 abgeschlossen.

```
softPwmCreate(LedPin,  0, 100);
```

Die Funktion besteht darin, mithilfe von Software einen PWM-Pin, LedPin, zu erstellen, dann die anfängliche Impulsbreite auf 0 zu setzen und die PWM-Periode 100 x 100us zu betragen.

```
while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
```

Lesen Sie im Hauptprogramm den Wert von Kanal 0 ab, der mit einem Potentiometer verbunden wurde. Speichern Sie den Wert in der Variablen `analogVal` und schreiben Sie ihn in LedPin. Jetzt können Sie sehen, wie sich die Helligkeit der LED mit dem Wert des Potentiometers ändert.

Für Python-Benutzer

Schritt 2: Öffnen Sie die Kodedatei

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

Schritt 3: Ausführen.

```
sudo python3 2.1.4_Potentiometer.py
```

Nachdem der Kode ausgeführt wurde, drehen Sie die Taste am Potentiometer. Die Intensität der LED ändert sich entsprechend.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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():
    # 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()
```

Code Erklärung

```
import ADC0834
```

Importieren `ADC0834` -Bibliothek Sie können den Inhalt der Bibliothek überprüfen, indem Sie den Befehl `nano ADC0834.py` aufrufen.

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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)
```

Definieren Sie in `setup()` die Benennungsmethode als BCM, legen Sie LedPin als PWM-Kanal fest und rendern Sie eine Frequenz von 2Khz.

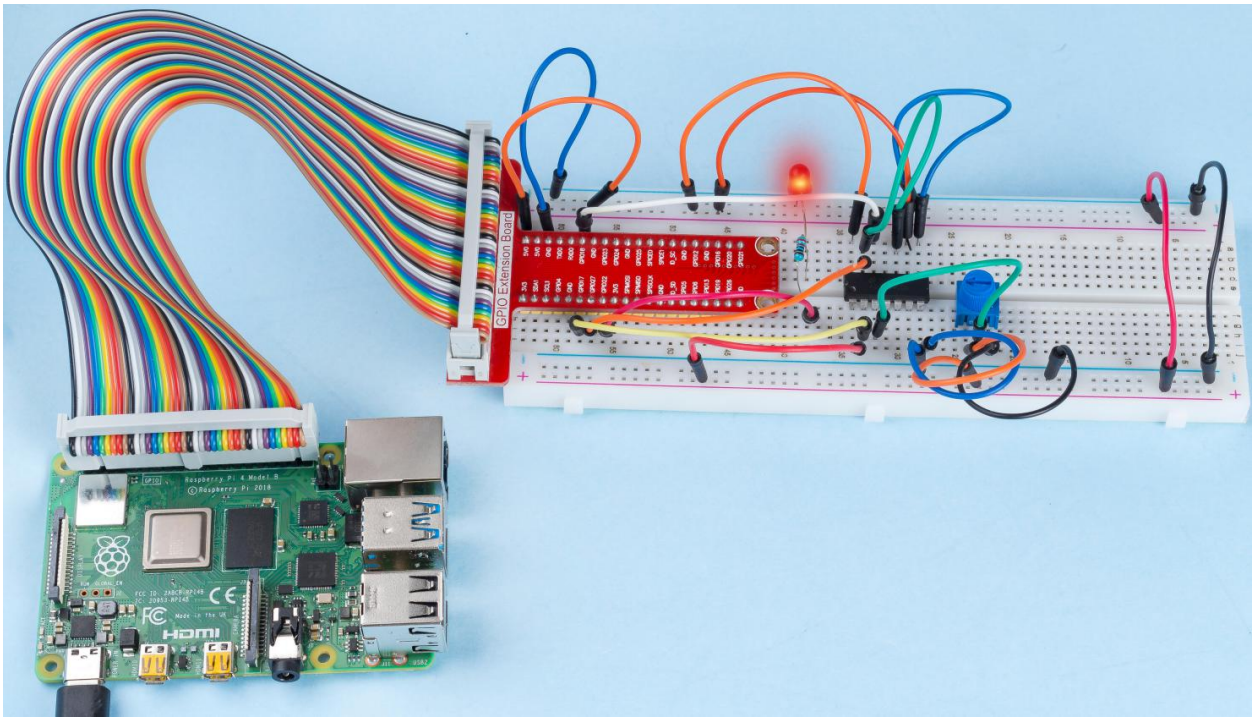
`ADC0834.setup()` : Initialize ADC0834, and connect the defined CS, CLK, DIO of ADC0834 to GPIO17, GPIO18 and GPIO27 respectively.

```
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)
```

Mit der Funktion `getResult()` werden die Analogwerte der vier Kanäle von ADC0834 gelesen. Standardmäßig liest die Funktion den Wert von CH0. Wenn Sie andere Kanäle lesen möchten, geben Sie bitte die Kanalnummer in () ein, z. `getResult(1)` .

Die Funktion `loop()` liest zuerst den Wert von CH0 und weist ihn dann der Variablen res zu. Rufen Sie danach die Funktion MAP auf, um den Lesewert des Potentiometers auf 0 ~ 100 abzubilden. Dieser Schritt wird verwendet, um den Arbeitszyklus von LedPin zu steuern. Jetzt können Sie sehen, dass sich die Helligkeit der LED mit dem Wert des Potentiometers ändert.

Phänomen Bild

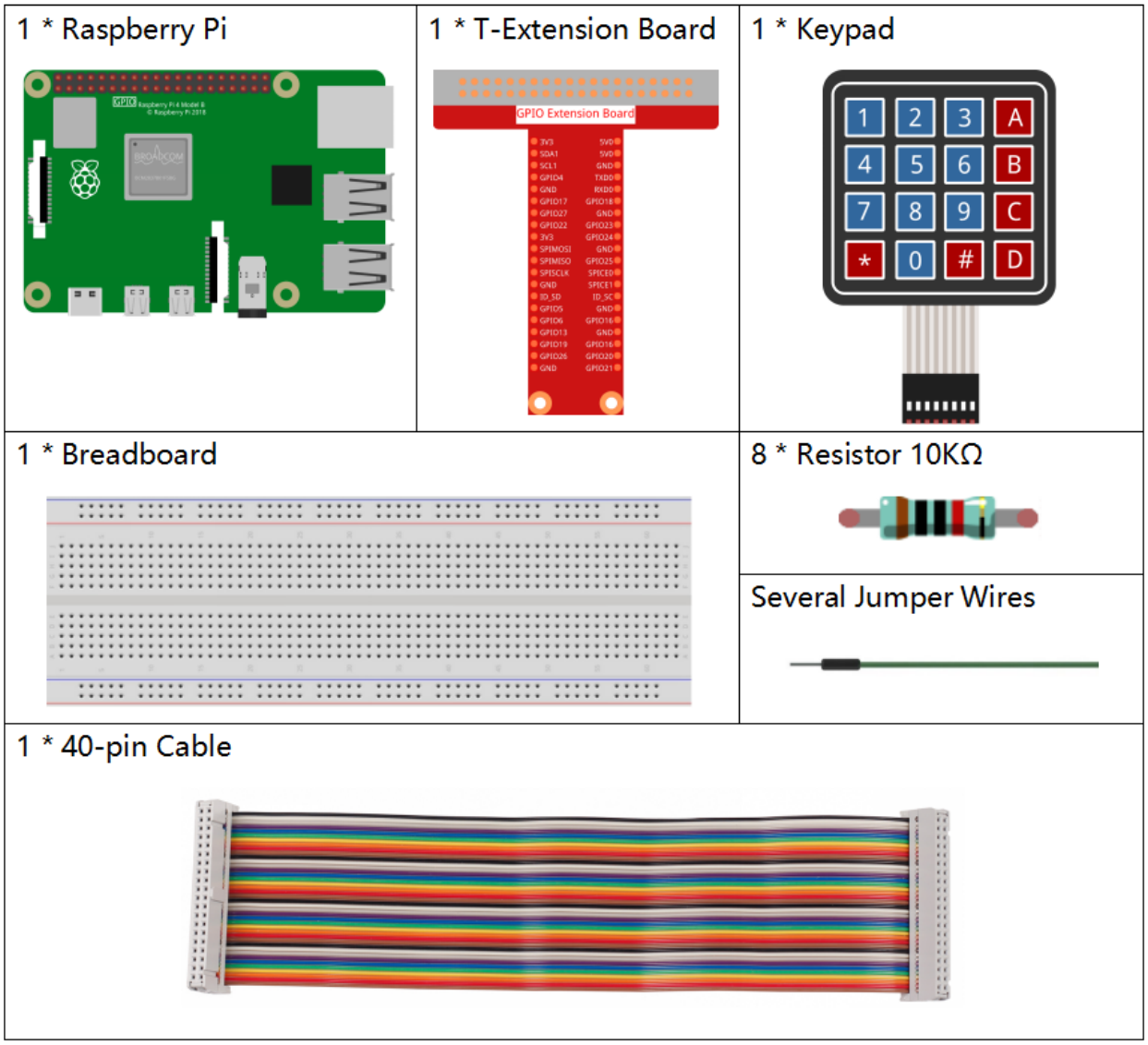


1.8.5 2.1.5 Tastatur

Einführung

Eine Tastatur ist eine rechteckige Anordnung von Schaltflächen. In diesem Projekt werden wir Eingabezeichen verwenden.

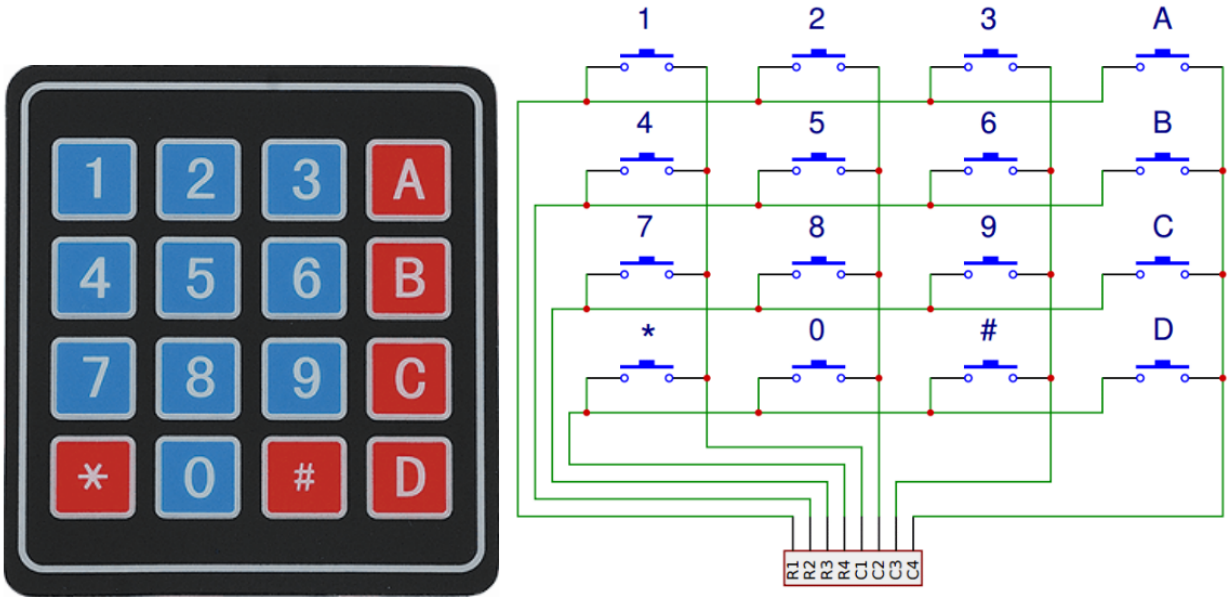
Komponenten



Prinzip

Tastenfeld

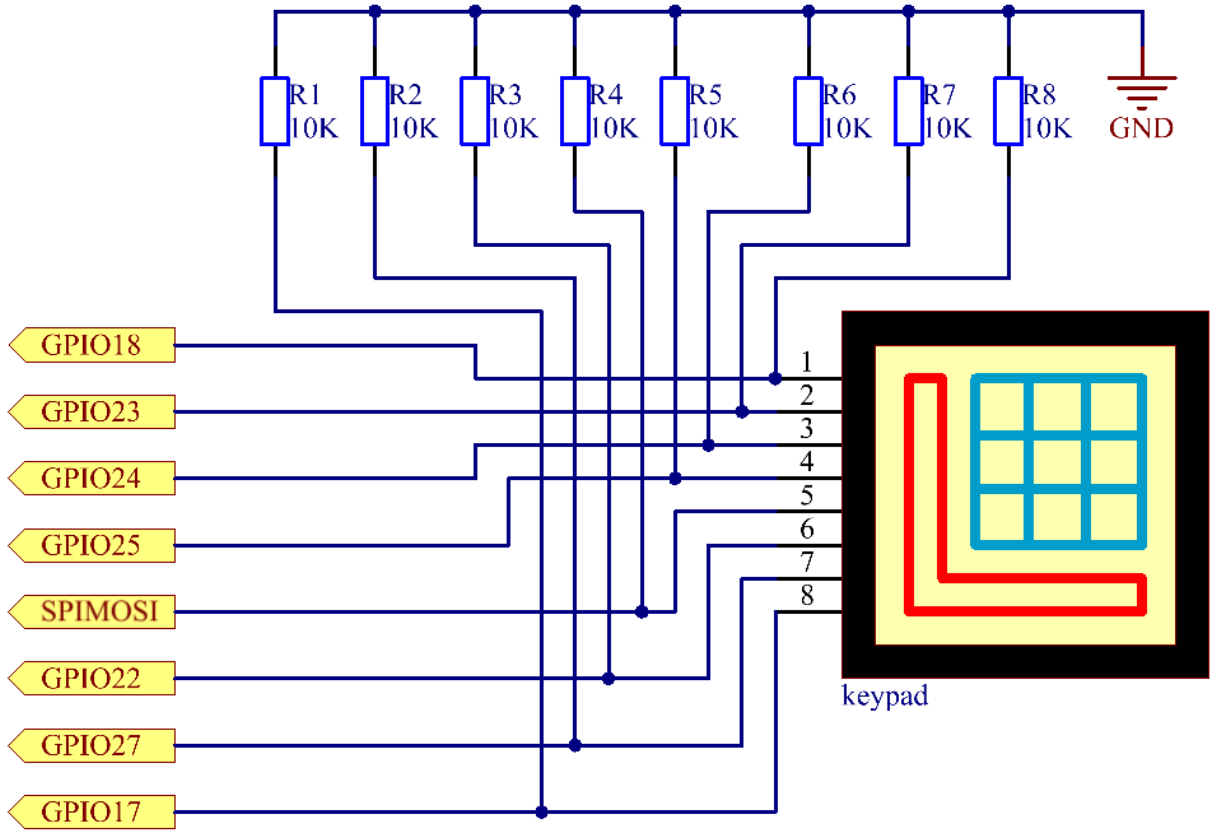
Eine Tastatur ist eine rechteckige Anordnung von 12 oder 16 AUS- (EIN) Tasten. Der Zugriff auf ihre Kontakte erfolgt über einen Header, der zum Anschluss mit einem Flachbandkabel oder zum Einsetzen in eine Leiterplatte geeignet ist. Bei einigen Tastaturen ist jede Taste mit einem separaten Kontakt in der Kopfzeile verbunden, während alle Tasten eine gemeinsame Masse haben.



Häufiger sind die Tasten Matrix kodiert, was bedeutet, dass jede von ihnen ein eindeutiges Leiterpaar in einer Matrix überbrückt. Diese Konfiguration eignet sich zum Abfragen durch einen Mikrocontroller, der so programmiert werden kann, dass er nacheinander einen Ausgangsimpuls an jeden der vier horizontalen Drähte sendet. Während jedes Impulses werden die verbleibenden vier vertikalen Drähte nacheinander überprüft, um festzustellen, welcher, falls vorhanden, ein Signal führt. Pullup- oder Pulldown-Widerstände sollten zu den Eingangsleitungen hinzugefügt werden, um zu verhindern, dass sich die Eingänge des Mikrocontrollers unvorhersehbar verhalten, wenn kein Signal vorhanden ist.

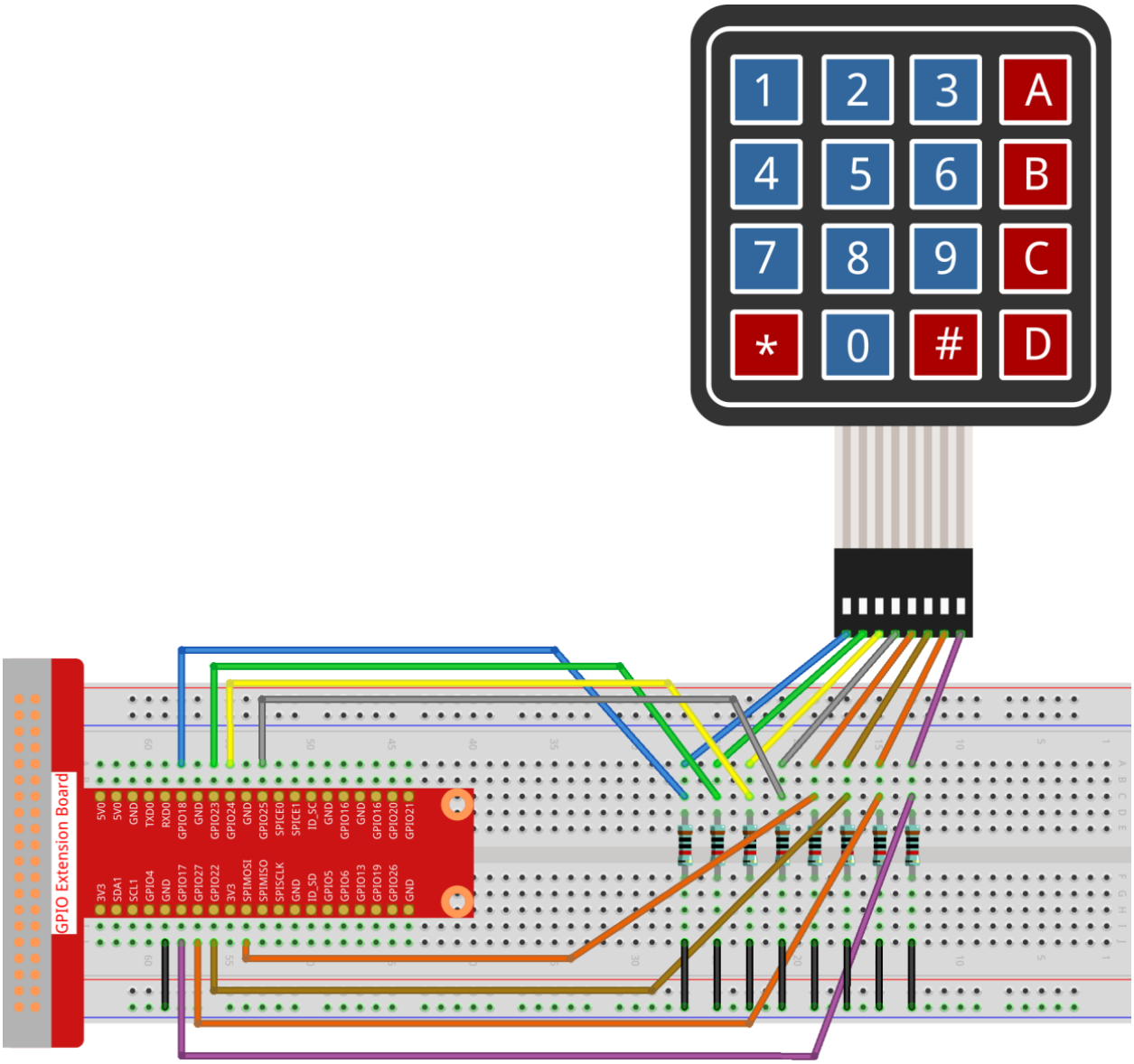
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Kodedatei.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.5/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.1.5_Keypad.cpp -lwiringPi
```

Schritt 4: Ausführen.

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

Nachdem die Kode ausgeführt wurde, werden die Werte der gedrückten Tasten auf der Tastatur (Tastenwert) auf dem Bildschirm gedruckt.

Code

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

#define ROWS  4
#define COLS  4
#define BUTTON_NUM (ROWS * COLS)

unsigned char KEYS[BUTTON_NUM] {
'1','2','3','A',
'4','5','6','B',
'7','8','9','C',
'*','0','#','D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};

void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);

void init(void) {
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
        pinMode(colPins[i], INPUT);
    }
}

int main(void){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            keyPrint(pressed_keys);
            keyCopy(last_key_pressed, pressed_keys);
        }
        delay(100);
    }
    return 0;
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
    for(int i=0 ; i<ROWS ; i++ ){
        digitalWrite(rowPins[i], HIGH);
        for(int j =0 ; j < COLS ; j++){
            index = i * ROWS + j;
            if(digitalRead(colPins[j]) == 1){
                result[count]=KEYS[index];
                count += 1;
            }
        }
        delay(1);
        digitalWrite(rowPins[i], LOW);
    }
}

bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}

void keyPrint(unsigned char* a){
    if (a[0] != 0){
        printf("%c",a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c",a[i]);
        }
    }
    printf("\n");
}

void keyClear(unsigned char* a){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = 0;
    }
}

int keyIndexOf(const char value){
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    for (int i=0; i<BUTTON_NUM; i++){
        if ((const char)KEYS[i] == value){
            return i;
        }
    }
    return -1;
}
```

Code Erklärung

```
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};
```

Deklarieren Sie jede Taste der Matrixtastatur zu den Array- `keys[]` und definieren Sie die Pins für jede Zeile und Spalte.

```
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);
    }
```

Dies ist der Teil der Hauptfunktion, der den Tastenwert liest und druckt.

Die Funktion `keys[]` liest den Status jeder Taste.

Mit `KeyCompare()` und `keyCopy()` wird beurteilt, ob sich der Status einer Schaltfläche geändert hat (dh eine Schaltfläche wurde gedrückt oder losgelassen).

`keyPrint()` druckt den Tastenwert der Taste, deren aktueller Niveau hoch ist (die Taste wird gedrückt).

```
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;
            }
        }
```

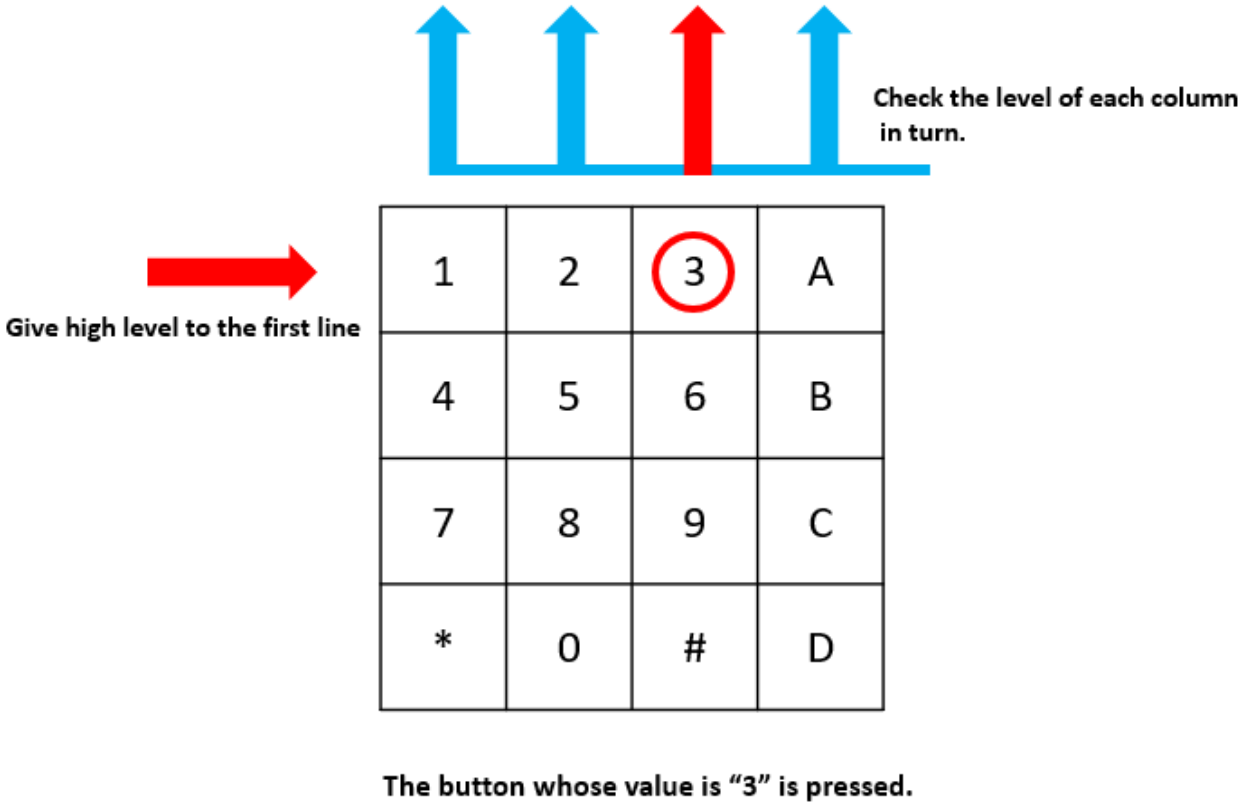
(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        delay(1);
        digitalWrite(rowPins[i], LOW);
    }
}
```

Diese Funktion weist jeder Zeile nacheinander eine hohe Ebene zu, und wenn die Taste in der Spalte gedrückt wird, erhält die Spalte, in der sich die Taste befindet, eine hohe Ebene. Nach der zweischichtigen Schleifenbeurteilung generiert die Schlüsselzustandskompilierung ein Array (`reasult[]`).

Beim Drücken von Taste 3:



`rowPin[0]` writes in the high level, and `colPin[2]` gets the high level. `colPin[0]` , `colPin[1]` , `colPin[3]` get the low level.

This gives us 0,0,1,0. When `rowPin[1]` , `rowPin[2]` and `rowPin[3]` are written in high level, `colPin[0]` ~ `colPin[4]` will get low level.

Nach Abschluss der Schleifenbeurteilung wird ein Array generiert:

`rowPin[0]` schreibt auf der hohen Ebene und `colPin[2]` erhält die hohe Ebene. `colPin[0]` , `colPin[1]` , `colPin[3]` erhalten den niedrigen Wert.

Dies gibt uns 0,0,1,0. Wenn `rowPin[1]` , `rowPin[2]` und `rowPin[3]` auf hoher Ebene geschrieben werden, wird `colPin[0]` ~ `colPin[4]` auf niedriger Ebene.

Nach Abschluss der Schleifenbeurteilung wird ein Array generiert:

```
result[BUTTON_NUM] {
0, 0, 1, 0,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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];
    }
}
```

Diese beiden Funktionen werden verwendet, um zu beurteilen, ob sich der Tastenstatus geändert hat. Wenn Sie beispielsweise Ihre Hand loslassen, wenn Sie ‚3' oder ‚2' drücken, gibt `keyCompare()` false zurück.

Mit `KeyCopy()` wird der aktuelle Schaltflächenwert für ein Array (`last_key_pressed[BUTTON_NUM]`) nach jedem Vergleich neu geschrieben. So können wir sie beim nächsten Mal vergleichen.

```
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");
}
```

Mit dieser Funktion wird der Wert der aktuell gedrückten Taste gedruckt. Wenn die Taste ‚1' gedrückt wird, wird die ‚1' gedruckt. Wenn die Taste ‚1' gedrückt wird und die Taste ‚3' gedrückt wird, wird die ‚1, 3' gedruckt.

Für Python-Sprachbenutzer

Schritt 2: Öffnen Sie die Kodedatei.

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

Schritt 3: Ausführen.

```
sudo python3 2.1.5_Keypad.py
```

Nachdem die Kode ausgeführt wurde, werden die Werte der gedrückten Tasten auf der Tastatur (Tastenwert) auf dem Bildschirm gedruckt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

class Keypad():

    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys

def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
    colsPins = [10,22,27,17]
    keys = ["1","2","3","A",
            "4","5","6","B",
            "7","8","9","C",
            "*","0","#","D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []

def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Release resource
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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()
```

Code Erklärung

```
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 = []
```

Deklarieren Sie jede Taste der Matrixtastatur zu den Array- `keys[]` und definieren Sie die Pins für jede Zeile und Spalte.

```
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)
```

Dies ist der Teil der Hauptfunktion, der den Tastenwert liest und druckt.

Die Funktion `keyRead()` liest den Status jeder Taste.

Die Anweisung if `len(pressed_keys) != 0 and last_key_pressed != pressed_keys` wird zur Beurteilung verwendet

ob eine Taste gedrückt wird und der Status der gedrückten Taste. (Wenn Sie ‚3‘ drücken, während Sie ‚1‘ drücken, ist das Urteil haltbar.)

Druckt den Wert der aktuell gedrückten Taste, wenn die Bedingung haltbar ist.

Die Anweisung `last_key_pressed = pressed_keys` weist einem Array `last_key_pressed` den Status jeder Beurteilung zu, um die nächste Runde der bedingten Beurteilung zu erleichtern.

```
def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
```

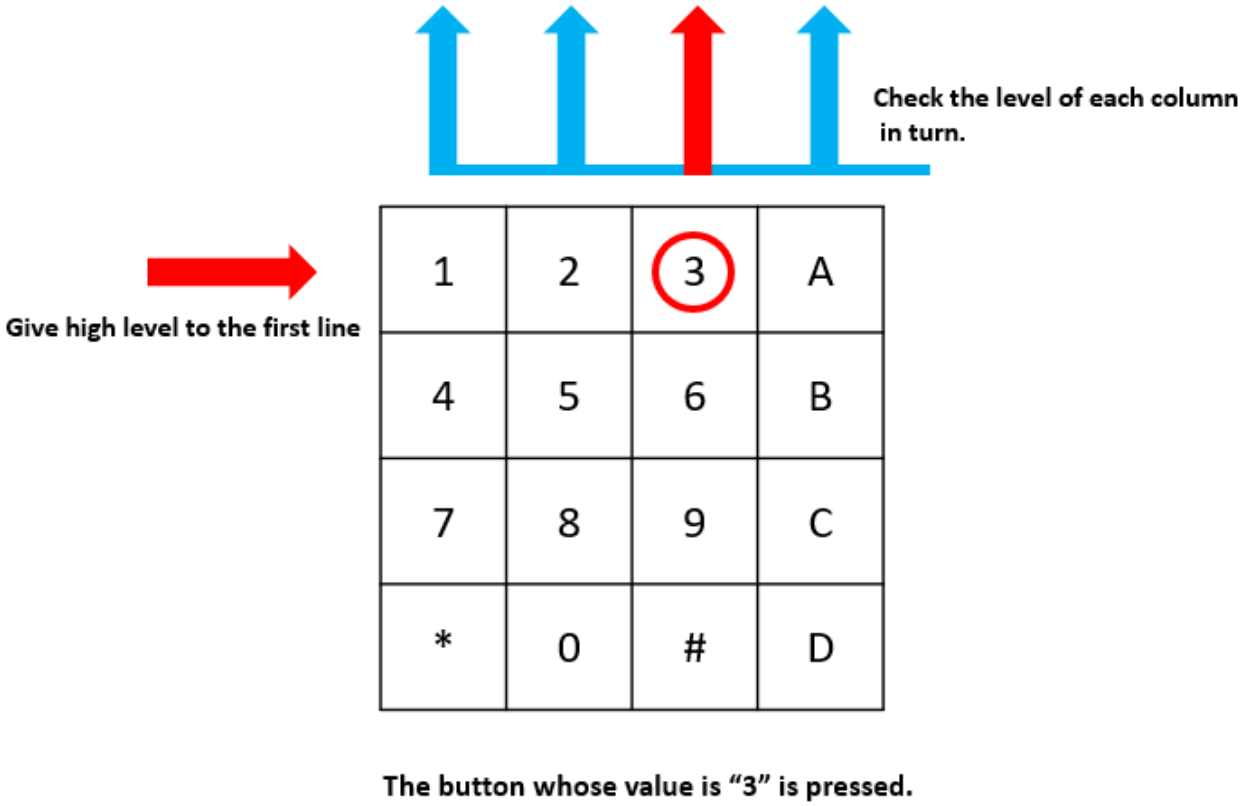
(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                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
```

Diese Funktion weist jeder Zeile nacheinander eine hohe Ebene zu, und wenn die Schaltfläche in der Spalte gedrückt wird, erhält die Spalte, in der sich die Taste befindet, eine hohe Ebene. Nachdem die Zweischichtschleife beurteilt wurde, wird der Wert der Schaltfläche, deren Status 1 ist, im Array press_keys gespeichert.

Wenn Sie die Taste ‚3' drücken:



rowPins[0] wird auf hoher Ebene geschrieben, und colPins[2] wird auf hoher Ebene geschrieben.

colPins[0] colPins[1] colPins[3] erhalten einen niedrigen Wert.

Es gibt vier Zustände: 0, 0, 1, 0; und wir schreiben '3' in press_keys.

Wenn rowPins[1], rowPins[2], rowPins[3] auf eine hohe Ebene geschrieben werden, erhalten colPins[0] ~ colPins[4] eine niedrige Ebene.

Die Schleife wurde gestoppt, dort wird press_keys = '3' zurückgegeben.

Wenn Sie die Tasten '1' und '3' drücken, wird pressed_keys = ['1','3'] zurückgegeben.

`rowPins[0]` wird auf hoher Ebene geschrieben, und `colPins[2]` wird auf hoher Ebene geschrieben.

`colPins[0] colPins[1] colPins[3]` erhalten einen niedrigen Wert.

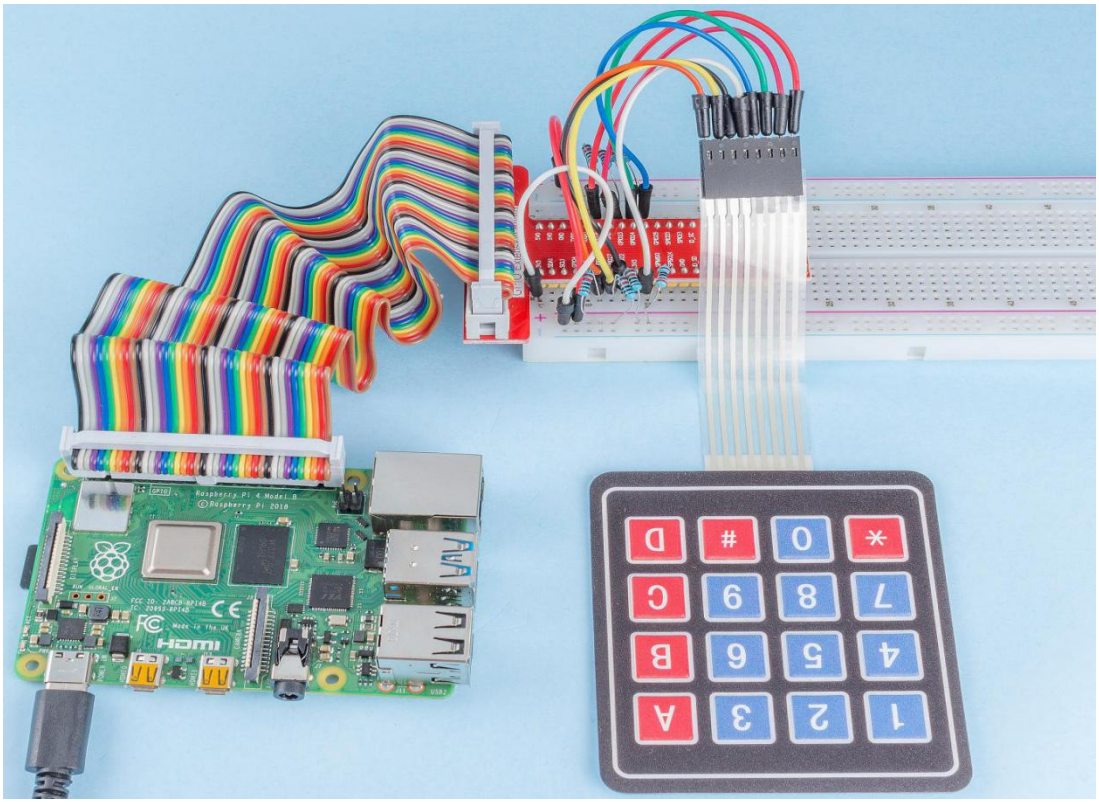
Es gibt vier Zustände: 0, 0, 1, 0; und wir schreiben ‚3' in press_keys.

Wenn `rowPins[1]` , `rowPins[2]` , `rowPins[3]` auf eine hohe Ebene geschrieben werden, erhalten `colPins[0]` ~ `colPins[4]` eine niedrige Ebene.

Die Schleife wurde gestoppt, dort wird `press_keys = '3'` zurückgegeben.

Wenn Sie die Tasten ‚1' und ‚3' drücken, wird `pressed_keys = ['1','3']` zurückgegeben.

Phänomen Bild

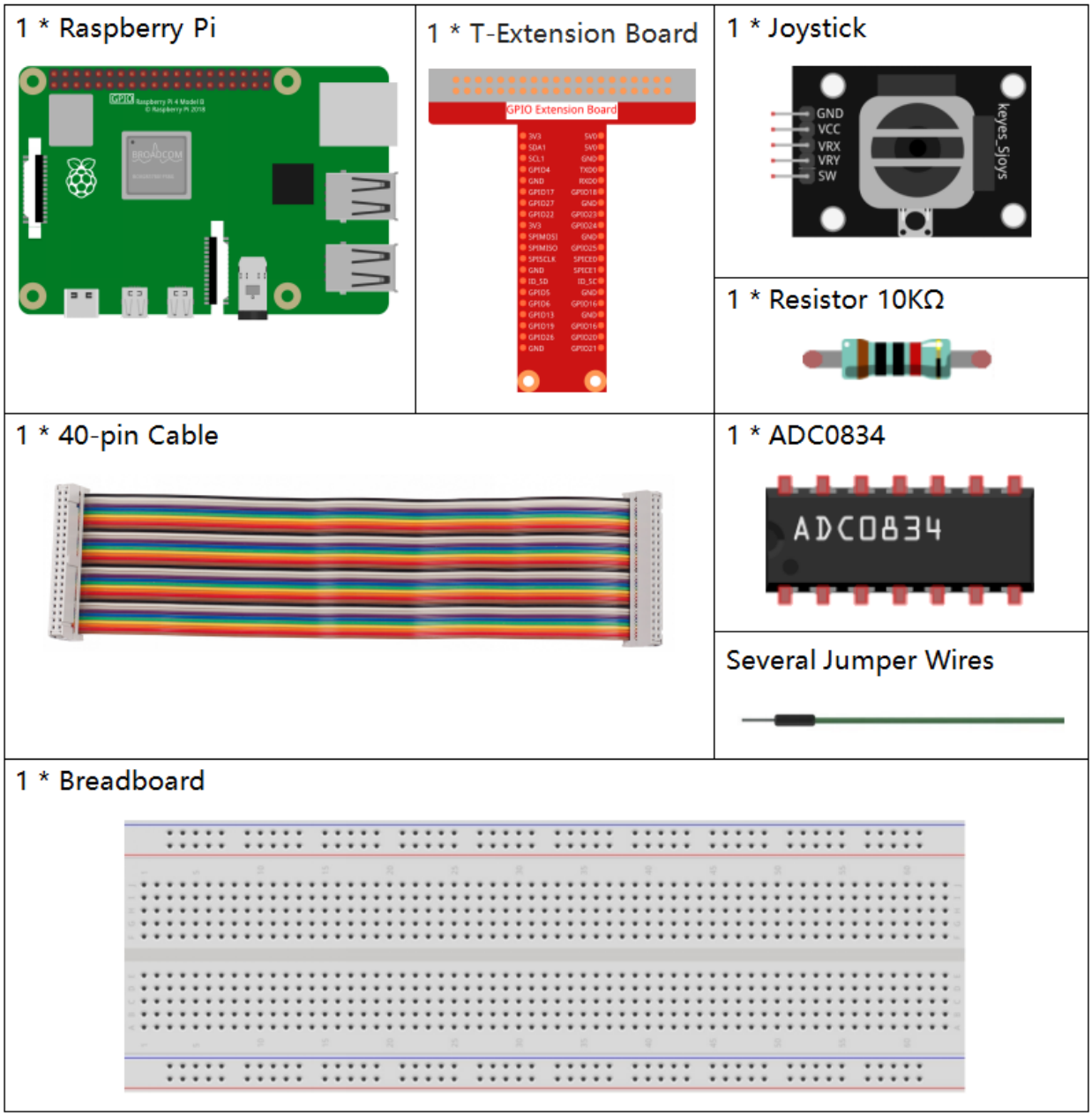


1.8.6 2.1.6 Joystick

Einführung

In diesem Projekt lernen wir, wie der Joystick funktioniert. Wir manipulieren den Joystick und zeigen die Ergebnisse auf dem Bildschirm an.

Komponenten



Prinzip

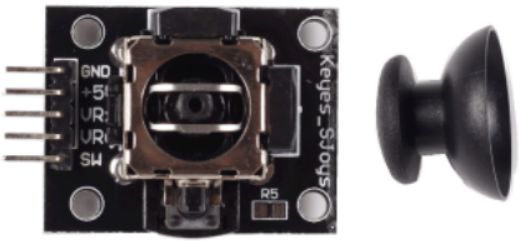
Joystick

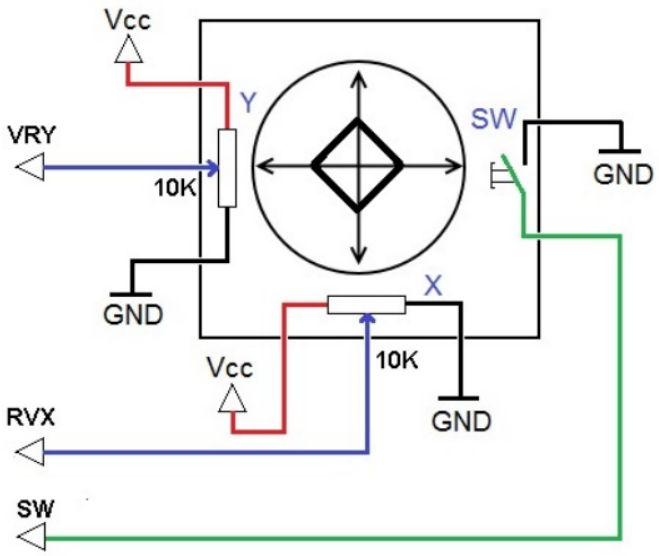
Die Grundidee eines Joysticks besteht darin, die Bewegung eines Sticks in elektronische Informationen umzuwandeln, die ein Computer verarbeiten kann.

Um dem Computer einen vollständigen Bewegungsbereich zu übermitteln, muss ein Joystick die Position des Sticks auf zwei Achsen messen - der X-Achse (von links nach rechts) und der Y-Achse (nach oben und unten). Genau wie in der Grundgeometrie bestimmen die XY-Koordinaten genau die Position des Sticks.

Um die Position des Steuerknüppels zu bestimmen, überwacht das Joystick-Steuerungssystem einfach die Position jeder Welle. Das herkömmliche analoge Joystick-Design macht dies mit zwei Potentiometern oder variablen Widerständen.

Der Joystick verfügt außerdem über einen digitalen Eingang, der beim Herunterdrücken des Joysticks betätigt wird.

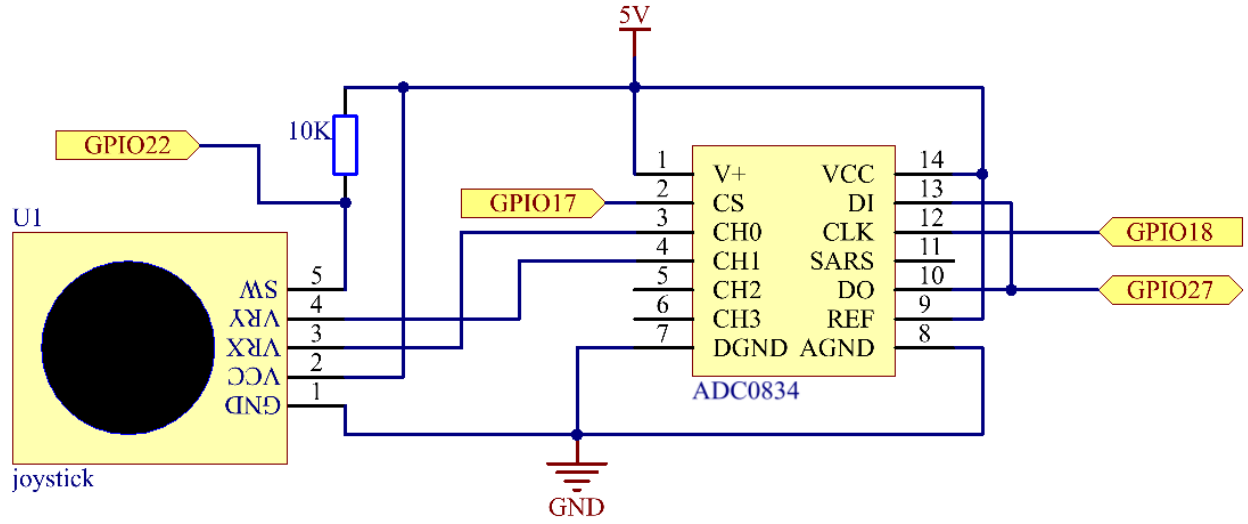




Schematische Darstellung

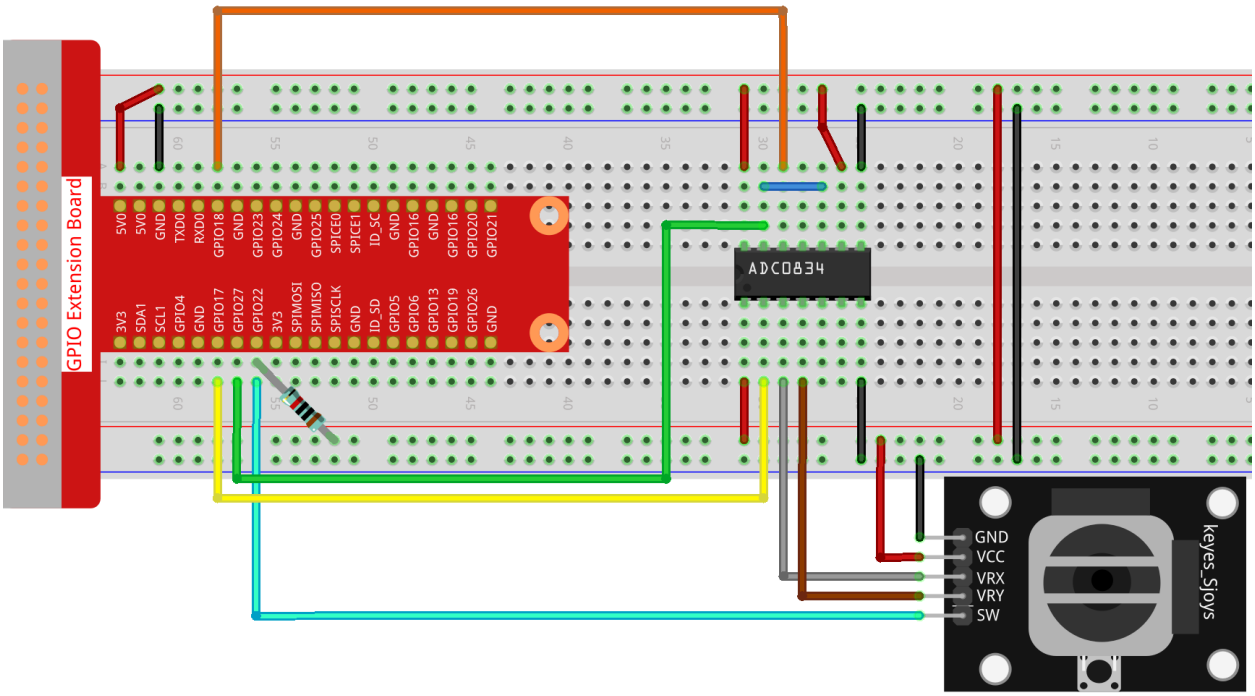
Wenn die Daten des Joysticks gelesen werden, gibt es einige Unterschiede zwischen den Achsen: Die Daten der X- und Y-Achse sind analog. Daher muss ADC0834 verwendet werden, um den Analogwert in einen Digitalwert umzuwandeln. Die Daten der Z-Achse sind digital, sodass Sie den GPIO direkt zum Lesen oder den ADC zum Lesen verwenden können.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.6/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.1.6_Joystick.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, drehen Sie den Joystick, und die entsprechenden Werte von x, y, Btn werden auf dem Bildschirm angezeigt.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;

#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2
#define     BtnPin    3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
//Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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
        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;
}
```

Code Erklärung

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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);
    ......
```

Der Arbeitsprozess der Funktion ist in in *2.1.4 Potentiometer* beschrieben.

```
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);
    }
```

VRX und VRY des Joysticks sind mit CH0 bzw. CH1 des ADC0834 verbunden. Daher wird die Funktion `getResult()` aufgerufen, um die Werte von CH0 und CH1 zu lesen. Dann sollten die gelesenen Werte in den Variablen `x_val` und `y_val` gespeichert werden. Lesen Sie außerdem den Wert von SW des Joysticks und speichern Sie ihn in der Variablen `btn_val`. Schließlich sollen die Werte von `x_val`, `y_val` und `btn_val` mit der Funktion `print()` gedruckt werden.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Ausführen.

```
sudo python3 2.1.6_Joystick.py
```

Nachdem die Kode ausgeführt wurde, drehen Sie den Joystick, und die entsprechenden Werte von x, y, btn werden auf dem Bildschirm angezeigt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

BtnPin = 22
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

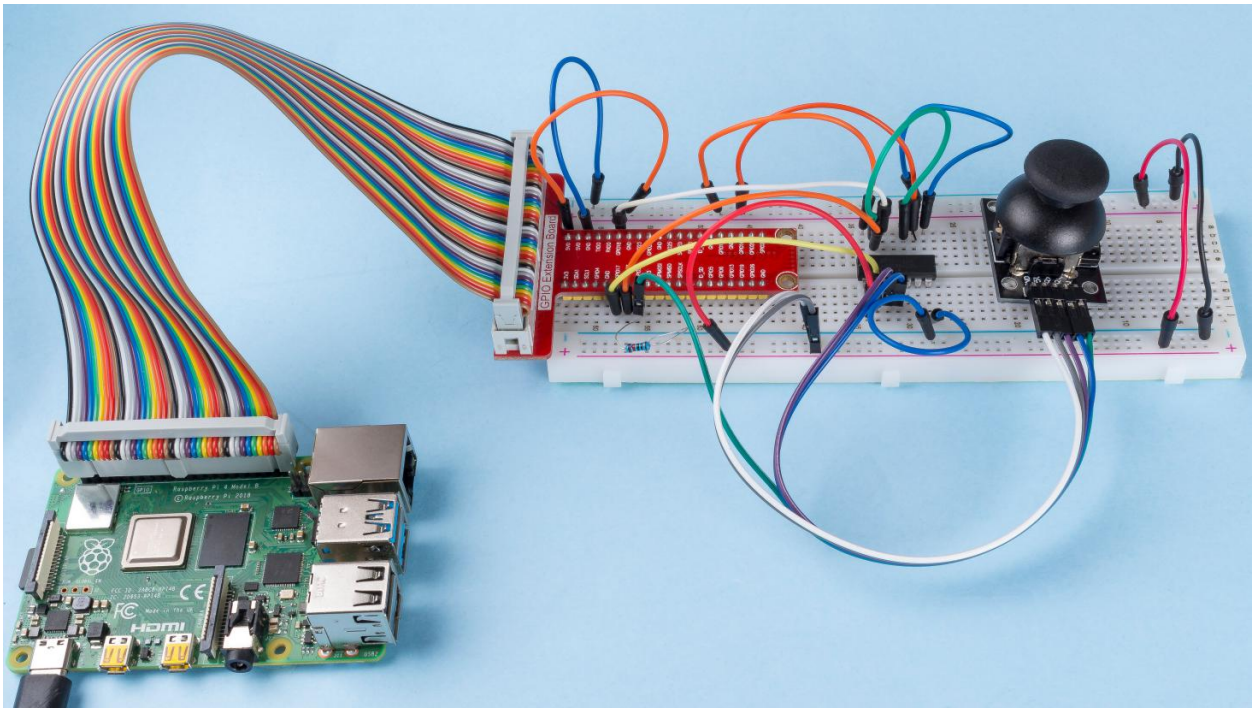
```
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC0834.setup()

def destroy():
    # Release resource
    GPIO.cleanup()

def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)

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

Code Erklärung

```
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)
```

VRX und VRY des Joysticks sind mit CH0 bzw. CH1 des ADC0834 verbunden. Daher wird die Funktion `getResult()` aufgerufen, um die Werte von CH0 und CH1 zu lesen. Dann sollten die gelesenen Werte in den Variablen `x_val` und `y_val` gespeichert werden. Lesen Sie außerdem den Wert von SW des Joysticks und speichern Sie ihn in der Variablen `btn_val`. Schließlich sollen die Werte von `x_val`, `y_val` und `btn_val` mit der Funktion `print()` gedruckt werden.

Phänomen Bild



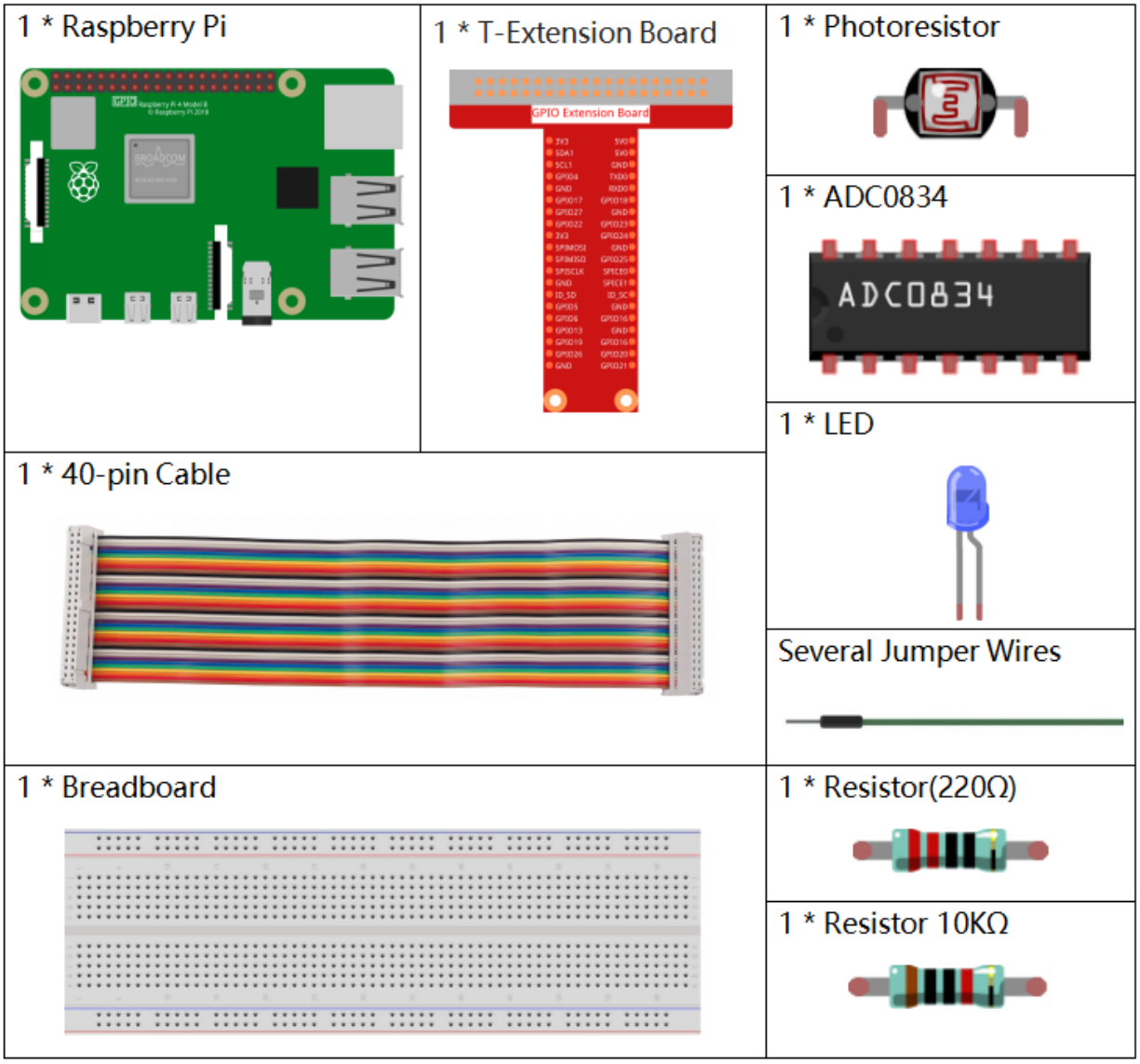
2.2 Sensoren

1.8.7 2.2.1 Fotowiderstand

Einführung

Der Fotowiderstand ist eine häufig verwendete Komponente der Umgebungslichtintensität im Leben. Es hilft dem Controller, Tag und Nacht zu erkennen und Lichtsteuerungsfunktionen wie Nachtlampen zu realisieren. Dieses Projekt ist dem Potentiometer sehr ähnlich, und Sie könnten denken, es ändert die Spannung, um Licht zu erfassen.

Komponenten

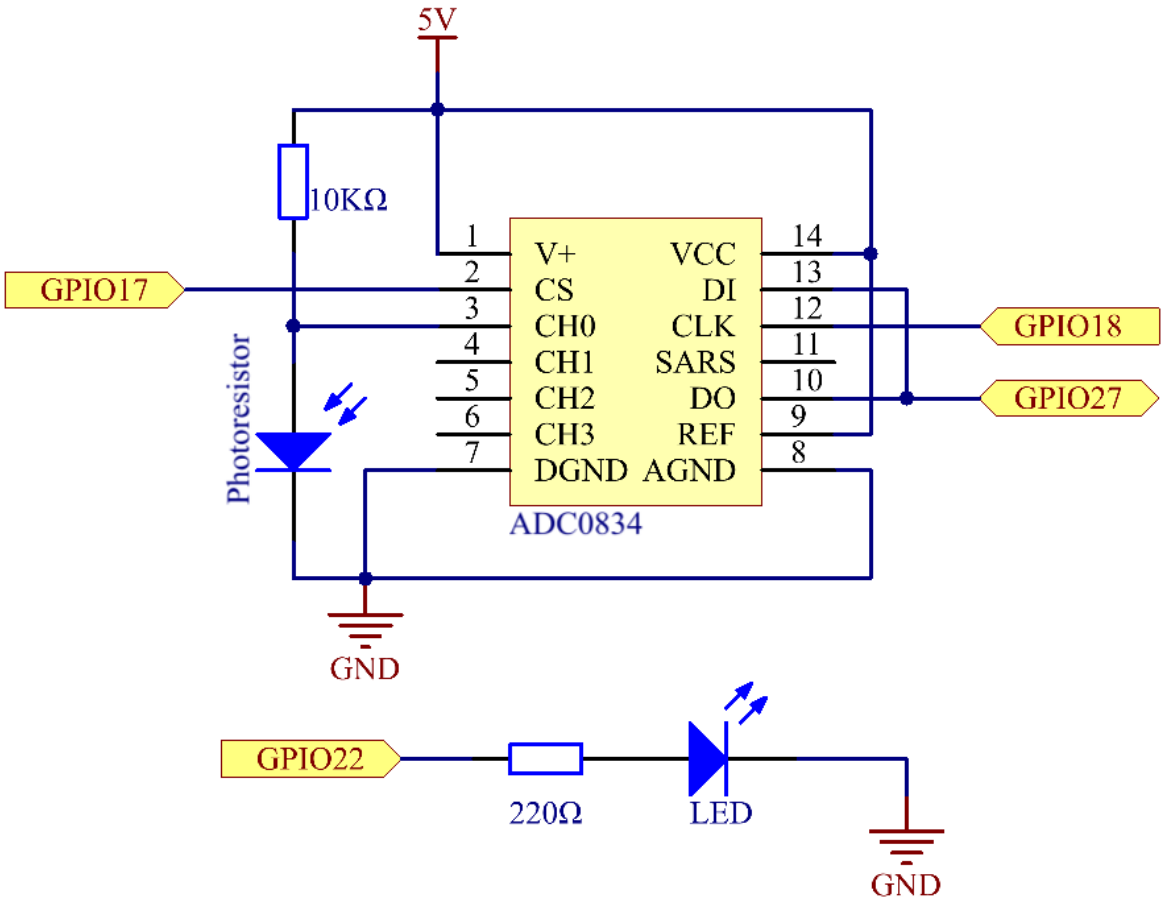


Prinzip

Ein Fotowiderstand oder eine Fotozelle ist ein lichtgesteuerter variabler Widerstand. Der Widerstand eines Fotowiderstands nimmt mit zunehmender Intensität des einfallenden Lichts ab; mit anderen Worten, es zeigt Fotoleitfähigkeit. Ein Fotowiderstand kann in lichtempfindlichen Detektorschaltungen sowie in licht- und dunkelheitsaktivierten Schaltkreisen eingesetzt werden.

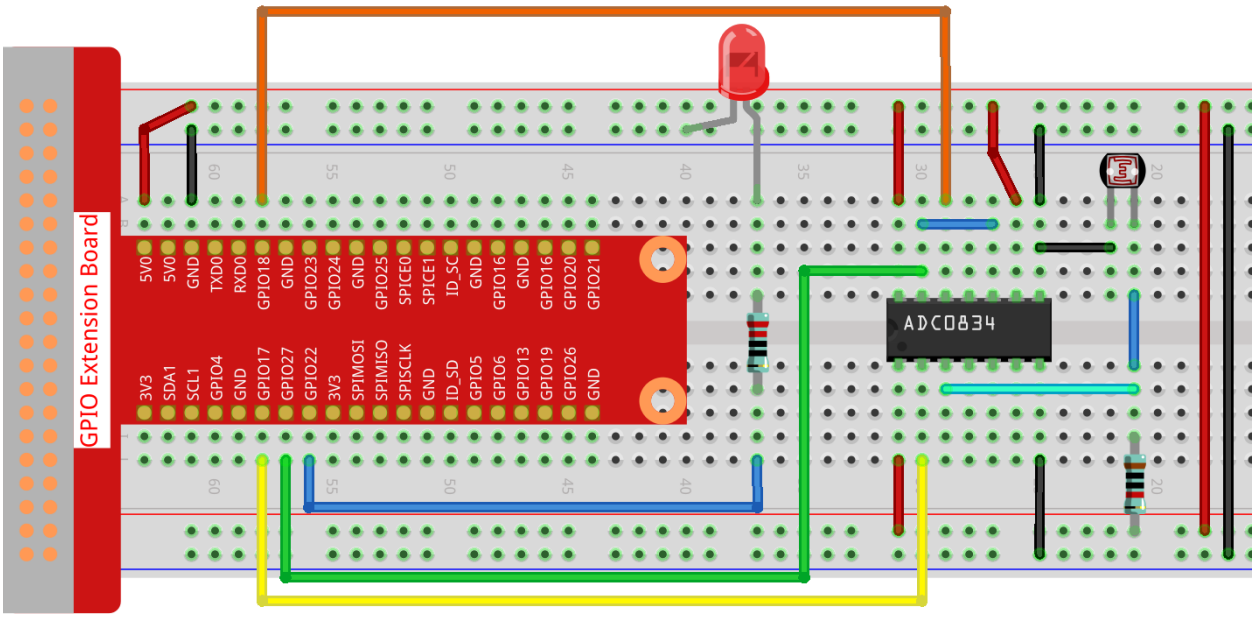
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.1/
```

Schritt 3: Kompilieren Sie der Kode.

```
gcc 2.2.1_Photoresistor.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Wenn die Kode ausgeführt wird, variiert die Helligkeit der LED in Abhängigkeit von der Lichtintensität, die der Fotowiderstand erfasst.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;

#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2
#define     LedPin    3

uchar get_ADC_Result(uint channel)
{
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
}
```

Code Erklärung

Die Koden hier sind die gleichen wie in 2.1.4 Potentiometer. Wenn Sie weitere Fragen haben, lesen Sie bitte die Kode-Erklärung von *2.1.4 Potentiometer* für Details.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.1_Photoresistor.py
```

Wenn die Kode ausgeführt wird, variiert die Helligkeit der LED in Abhängigkeit von der Lichtintensität, die der Fotowiderstand erfasst.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

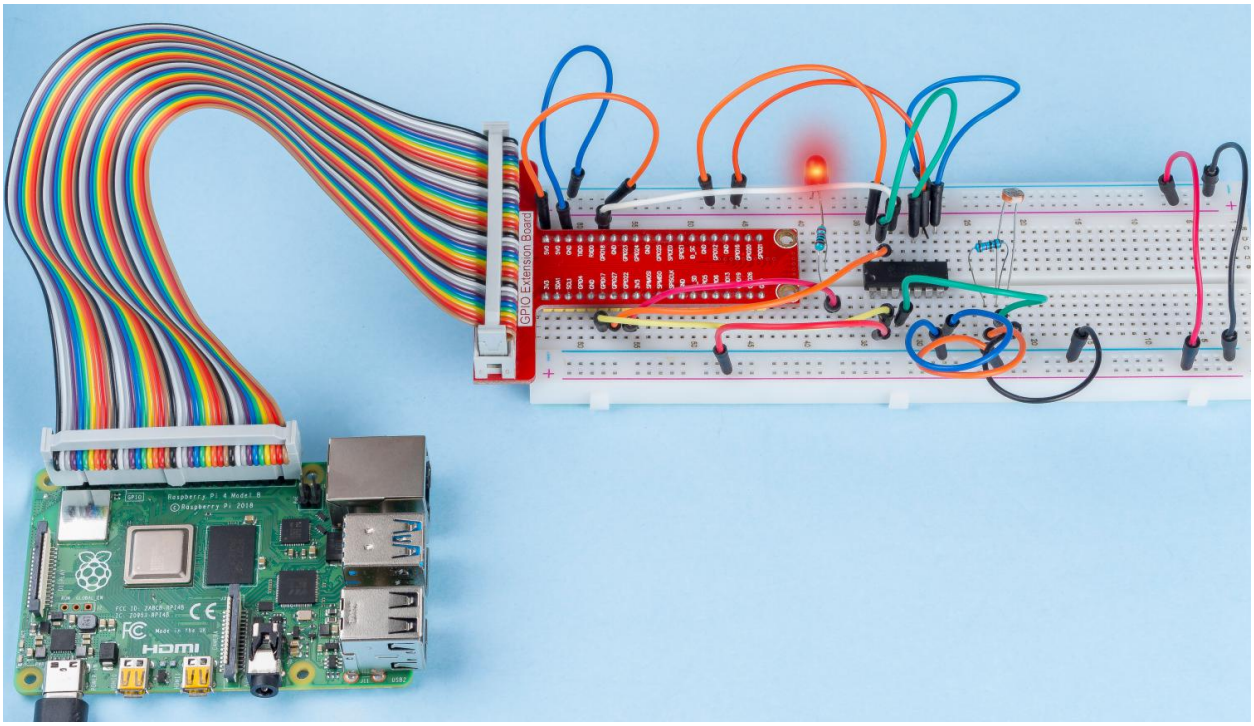
```
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)

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

def destroy():
    # 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()
```

Code Erklärung

```
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
```

Lesen Sie den Analogwert von CH0 von ADC0834 ab. Standardmäßig wird mit der Funktion `getResult()` der Wert von CH0 gelesen. Wenn Sie also andere Kanäle lesen möchten, geben Sie bitte 1, 2 oder 3 in () der Funktion `getResult()` ein. Als nächstes müssen Sie den Wert über die Druckfunktion drucken. Da das sich ändernde Element das Tastverhältnis von LedPin ist, wird die Berechnungsformel `analogVal * 100/255` benötigt, um analogVal in Prozent umzuwandeln. Schließlich wird `ChangeDutyCycle()` aufgerufen, um den Prozentsatz in LedPin zu schreiben.

Phänomen Bild

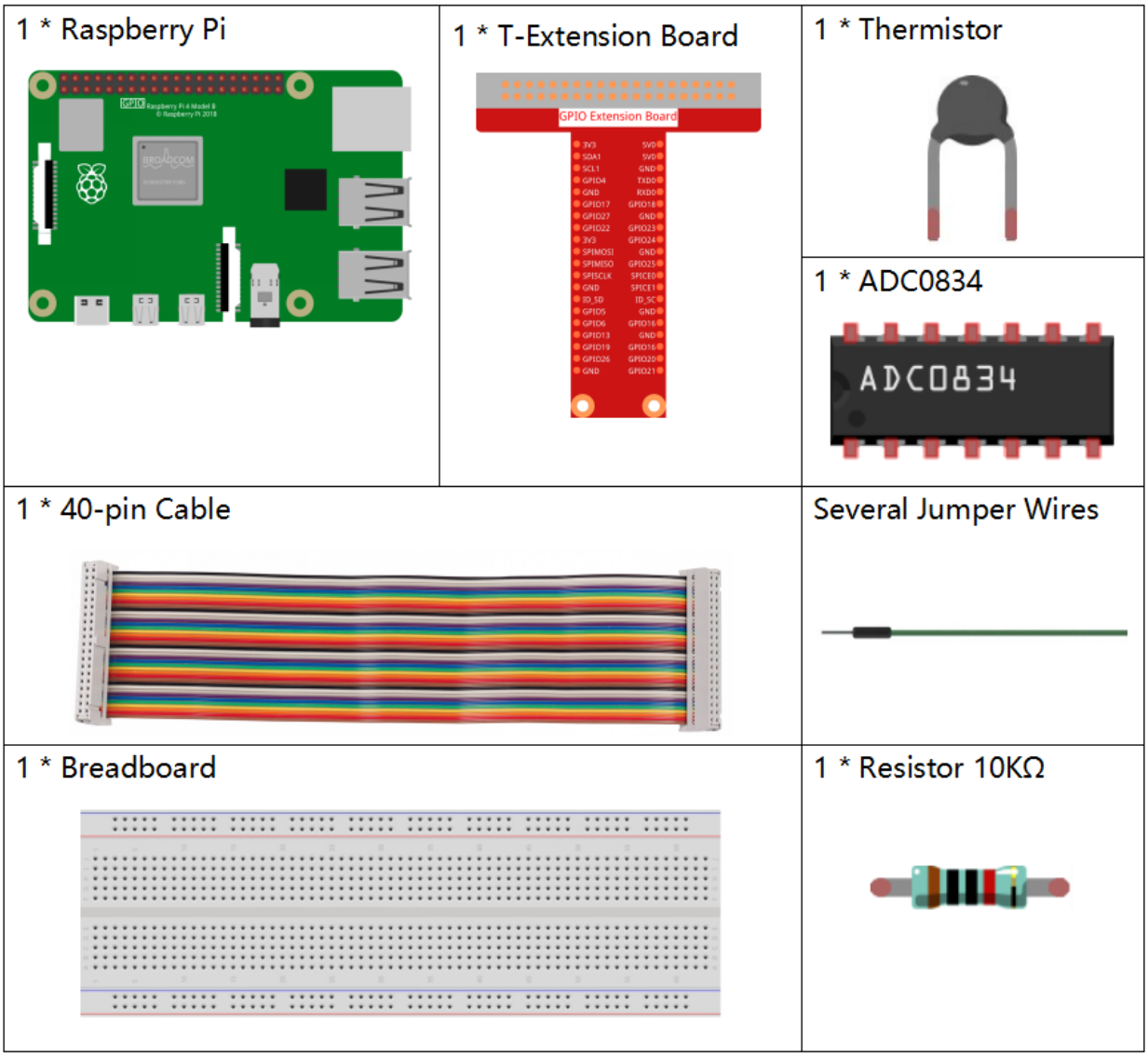


1.8.8 2.2.2 Thermistor

Einführung

Genau wie der Fotowiderstand Licht erfassen kann, ist der Thermistor ein temperaturempfindliches elektronisches Gerät, mit dem Funktionen der Temperaturregelung wie z. B. ein Wärmealarm realisiert werden können.

Komponenten



Prinzip

Ein Thermistor ist ein wärmeempfindlicher Widerstand, der eine präzise und vorhersagbare Widerstandsänderung proportional zu kleinen Temperaturänderungen aufweist. Wie sehr sich sein Widerstand ändert, hängt von seiner einzigartigen Zusammensetzung ab. Thermistoren sind Teile einer größeren Gruppe passiver Komponenten. Und im Gegensatz zu ihren Gegenstücken mit aktiven Komponenten können passive Geräte keine Leistungsverstärkung oder Verstärkung für eine Schaltung bereitstellen.

Der Thermistor ist ein empfindliches Element und es gibt zwei Arten: den negativen Temperaturkoeffizienten (NTC) und den positiven Temperaturkoeffizienten (PTC), auch bekannt als NTC und PTC. Sein Widerstand variiert erheblich mit der Temperatur. Der Widerstand des PTC-Thermistors steigt mit der Temperatur, während der Zustand des NTC dem ersteren entgegengesetzt ist. In diesem Experiment verwenden wir NTC.

Das Prinzip ist, dass sich der Widerstand des NTC-Thermistors mit der Temperatur der äußeren Umgebung ändert. Es erfasst die Echtzeit-Temperatur der Umgebung. Wenn die Temperatur höher wird, nimmt der Widerstand des Thermistors ab. Anschließend werden die Spannungsdaten vom A / D-Adapter in digitale Größen umgewandelt. Die Temperatur in Celsius oder Fahrenheit wird über die Programmierung ausgegeben.

In diesem Experiment werden ein Thermistor und ein 10k-Pull-up-Widerstand verwendet. Jeder Thermistor hat einen normalen Widerstand. Hier sind es 10 kOhm, die unter 25 Grad Celsius gemessen werden.

Hier ist die Beziehung zwischen dem Widerstand und der Temperatur:

$R_T = R_N \exp^{B(1/TK - 1/TN)}$

RT ist der Widerstand des NTC-Thermistors bei einer Temperatur von **TK**.

RN ist der Widerstand des NTC-Thermistors unter der Nenntemperatur **TN**. Hier beträgt der numerische Wert von **RN** 10k.

TK ist eine Kelvin-Temperatur und die Einheit ist K. Hier beträgt der numerische Wert von **TK** 273.15 + Grad Celsius.

TN ist eine Kelvin-Nenntemperatur; Das Gerät ist auch K. Hier beträgt der numerische Wert von **TN** 273.15+25.

Und **B**(beta), die Materialkonstante des NTC-Thermistors, wird auch als Wärmeempfindlichkeitsindex mit einem numerischen Wert von 3950 bezeichnet.

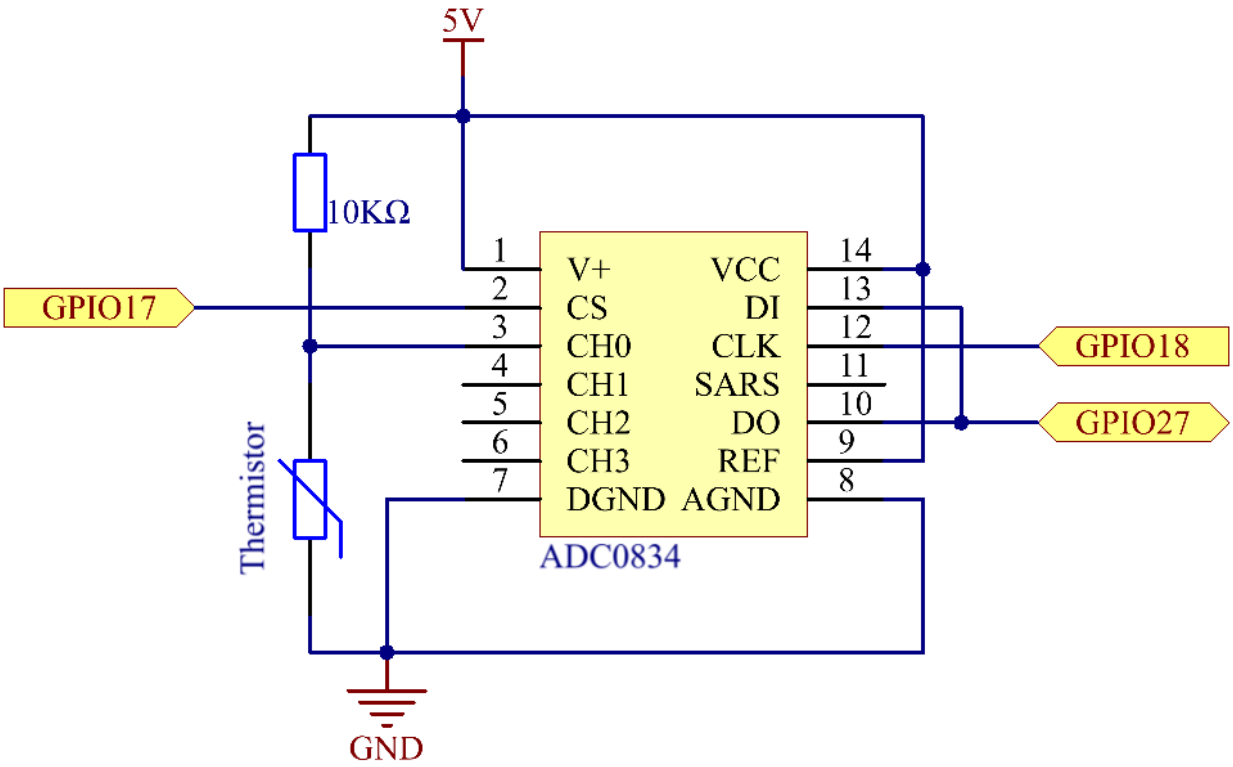
exp ist die Abkürzung für exponentiell, und die Basisnummer ist eine natürliche Nummer und entspricht ungefähr 2.7.

Konvertieren Sie diese Formel $T_K=1/(\ln(R_T/R_N)/B+1/T_N)$ um eine Kelvin-Temperatur zu erhalten, die minus 273.15 Grad Celsius entspricht.

Diese Beziehung ist eine empirische Formel. Sie ist nur dann genau, wenn Temperatur und Widerstand im effektiven Bereich liegen.

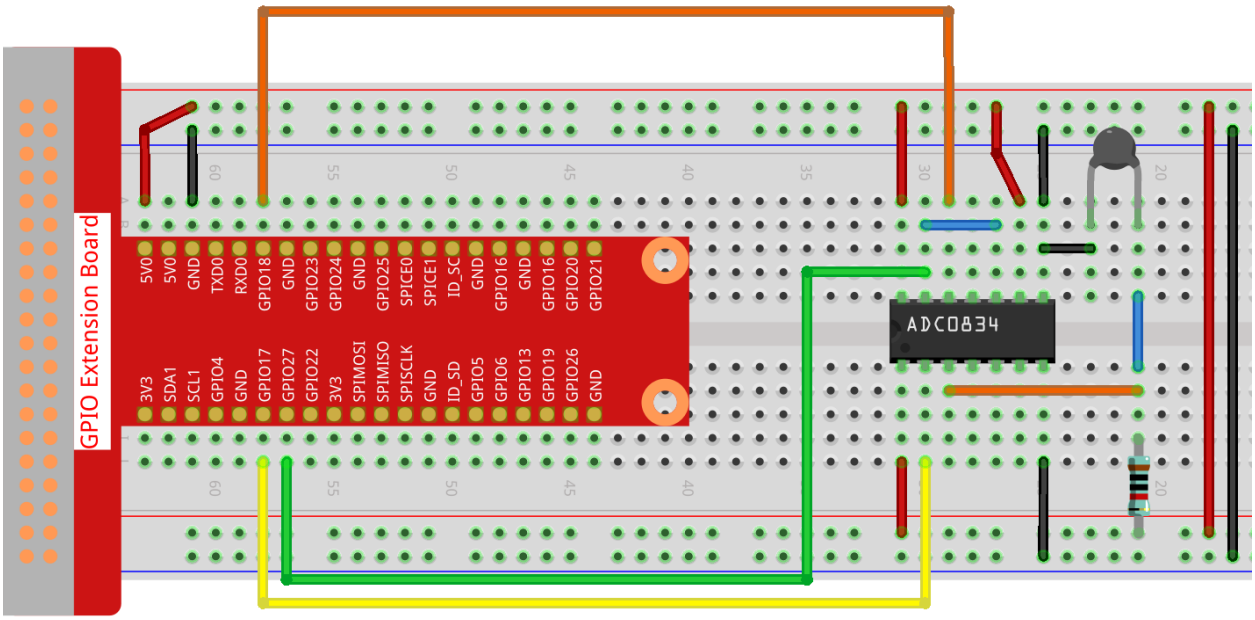
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.2/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.2.2_Thermistor.c -lwiringPi -lm
```

Bemerkung: -lm dient zum Laden der Bibliotheksmathematik. Nicht weglassen, sonst wird ein Fehler gemacht.

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Während die Kode ausgeführt wird, erkennt der Thermistor die Umgebungstemperatur, die nach Abschluss der Programmberechnung auf dem Bildschirm angezeigt wird.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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);
        delay(100);
    }
    return 0;
}
```

Code Erklärung

```
#include <math.h>
```

Es gibt eine C-Numerik-Bibliothek, die eine Reihe von Funktionen deklariert, um allgemeine mathematische Operationen und Transformationen zu berechnen.

```
analogVal = get_ADC_Result(0);
```

Mit dieser Funktion wird der Wert des Thermistors abgelesen.

```
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);
```

Diese Berechnungen wandeln die Thermistorwerte in Celsiuswerte um.

```
Vr = 5 * (double)(analogVal) / 255;
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
```

Diese beiden Kodezeilen berechnen die Spannungsverteilung mit dem analogen Lesewert, um Rt (Widerstand des Thermistors) zu erhalten.

```
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
```

Diese Kode bezieht sich auf das Einstecken von Rt in die Formel **TK=1/(ln(RT/RN)/B+1/TN)** ,um die Kelvin-Temperatur zu erhalten.

```
temp = temp - 273.15;
```

Wandeln Sie die Kelvin-Temperatur in Grad Celsius um.

```
Fah = cel * 1.8 +32;
```

Konvertieren Sie Grad Celsius in Fahrenheit.

```
printf("Celsius: %.2f C  Fahrenheit: %.2f F\n", cel, Fah);
```

Drucken Sie Celsius, Fahrenheit und ihre Einheiten auf dem Display.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.2_Thermistor.py
```

Während die Kode ausgeführt wird, erkennt der Thermistor die Umgebungstemperatur, die nach Abschluss der Programmberechnung auf dem Bildschirm angezeigt wird.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        time.sleep(0.2)

if __name__ == '__main__':
    init()
    try:
        loop()
    except KeyboardInterrupt:
        ADC0834.destroy()
```

Code Erklärung

```
import math
```

Es gibt eine numerische Bibliothek, die eine Reihe von Funktionen deklariert, um allgemeine mathematische Operationen und Transformationen zu berechnen.

```
analogVal = ADC0834.getResult()
```

Mit dieser Funktion wird der Wert des Thermistors abgelesen.

```
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 ' % (Cel, Fah))
```

Diese Berechnungen wandeln die Thermistorwerte in Grad Celsius und Fahrenheit um.

```
Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)
```

Diese beiden Kodezeilen berechnen die Spannungsverteilung mit dem analogen Lesewert, um Rt (Widerstand des Thermistors) zu erhalten.

```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

Diese Kode bezieht sich auf das Einstecken von Rt in die Formel **TK=1/(ln(RT/RN)/B+1/TN)** ,um die Kelvin-Temperatur zu erhalten.

```
temp = temp - 273.15
```

Wandeln Sie die Kelvin-Temperatur in Grad Celsius um.

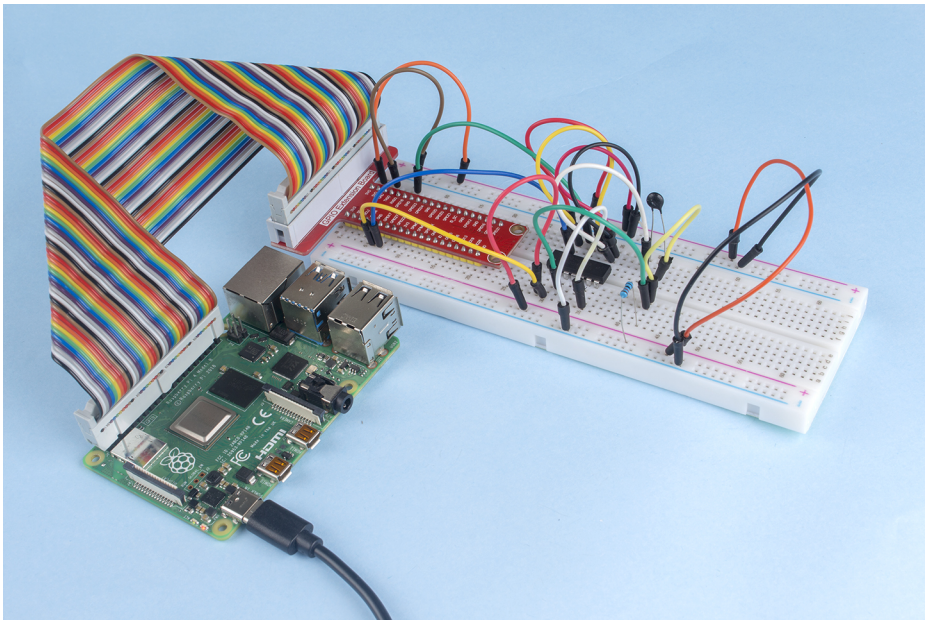
```
Fah = Cel * 1.8 + 32
```

Konvertieren Sie den Celsius-Grad in Fahrenheit-Grad.

```
print ('Celsius: %.2f °C  Fahrenheit: %.2f ' % (Cel, Fah))
```

Drucken Sie Celsius, Fahrenheit und ihre Einheiten auf dem Display.

Phänomen Bild



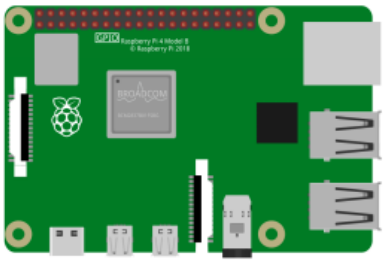
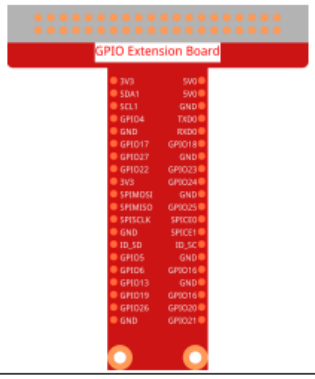
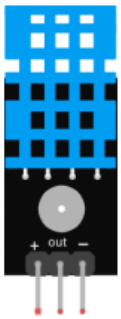
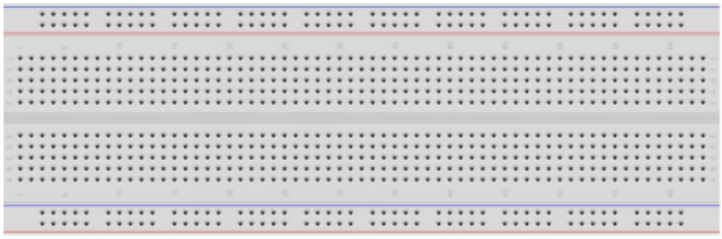
1.8.9 2.2.3 DHT-11

Einführung

Der digitale Temperatur- und Feuchtigkeitssensor DHT11 ist ein Verbundsensor, der einen kalibrierten digitalen Signalausgang für Temperatur und Luftfeuchtigkeit enthält. Die Technologie einer speziellen Sammlung digitaler Module und die Technologie der Temperatur- und Feuchtigkeitsmessung werden angewendet, um sicherzustellen, dass das Produkt eine hohe Zuverlässigkeit und ausgezeichnete Stabilität aufweist.

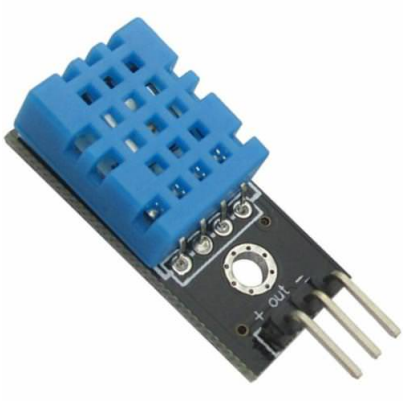
Die Sensoren umfassen einen Nasselement-Widerstandssensor und einen NTC-Temperatursensor und sind mit einem Hochleistungs-8-Bit-Mikrocontroller verbunden.

Komponenten

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

Prinzip

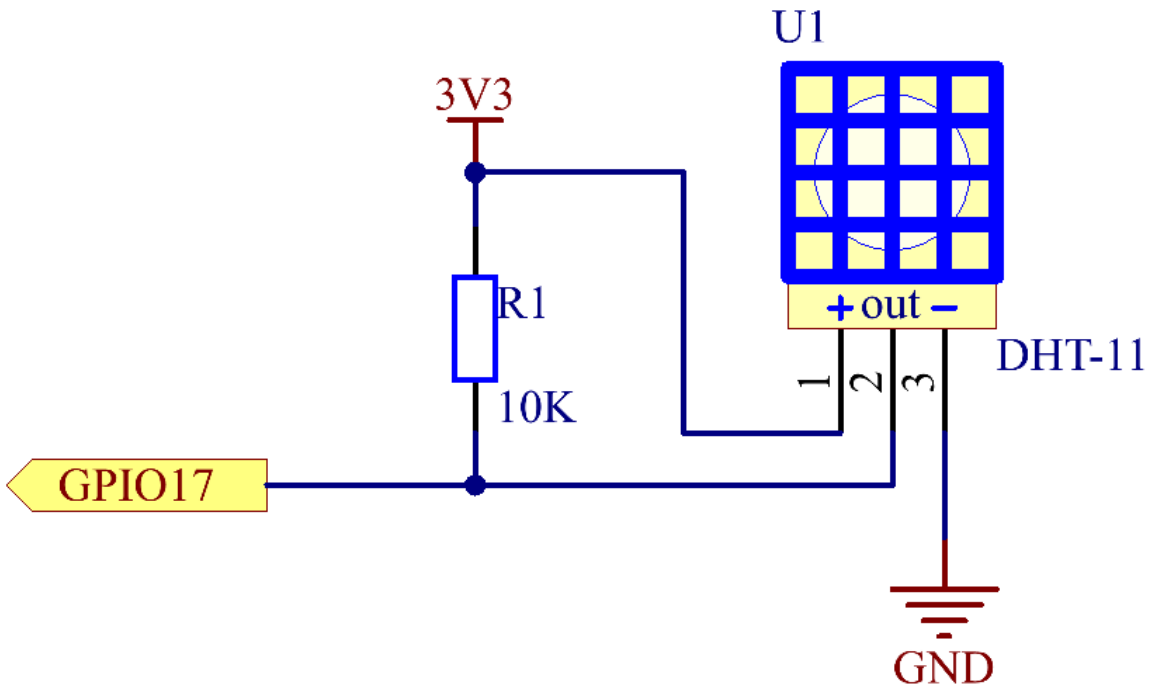
Der DHT11 ist ein grundlegender, äußerst kostengünstiger digitaler Temperatur- und Feuchtigkeitssensor. Es verwendet einen kapazitiven Feuchtigkeitssensor und einen Thermistor, um die Umgebungsluft zu messen, und spuckt ein digitales Signal auf den Datenstift aus (es werden keine analogen Eingangsstifte benötigt).



Es sind nur drei Pins verfügbar: VCC, GND und DATA. Der Kommunikationsprozess beginnt damit, dass die DATA-Leitung Startsignale an DHT11 sendet, und DHT11 empfängt die Signale und gibt ein Antwortsignal zurück. Dann empfängt der Host das Antwortsignal und beginnt mit dem Empfang von 40-Bit-Feuchtigkeitsdaten (8-Bit-Feuchtigkeits-Ganzzahl + 8-Bit-Feuchtigkeits-Dezimalzahl + 8-Bit-Temperatur-Ganzzahl + 8-Bit-Temperatur-Dezimalzahl + 8-Bit-Prüfsumme). Weitere Informationen finden Sie im DHT11-Datenblatt.

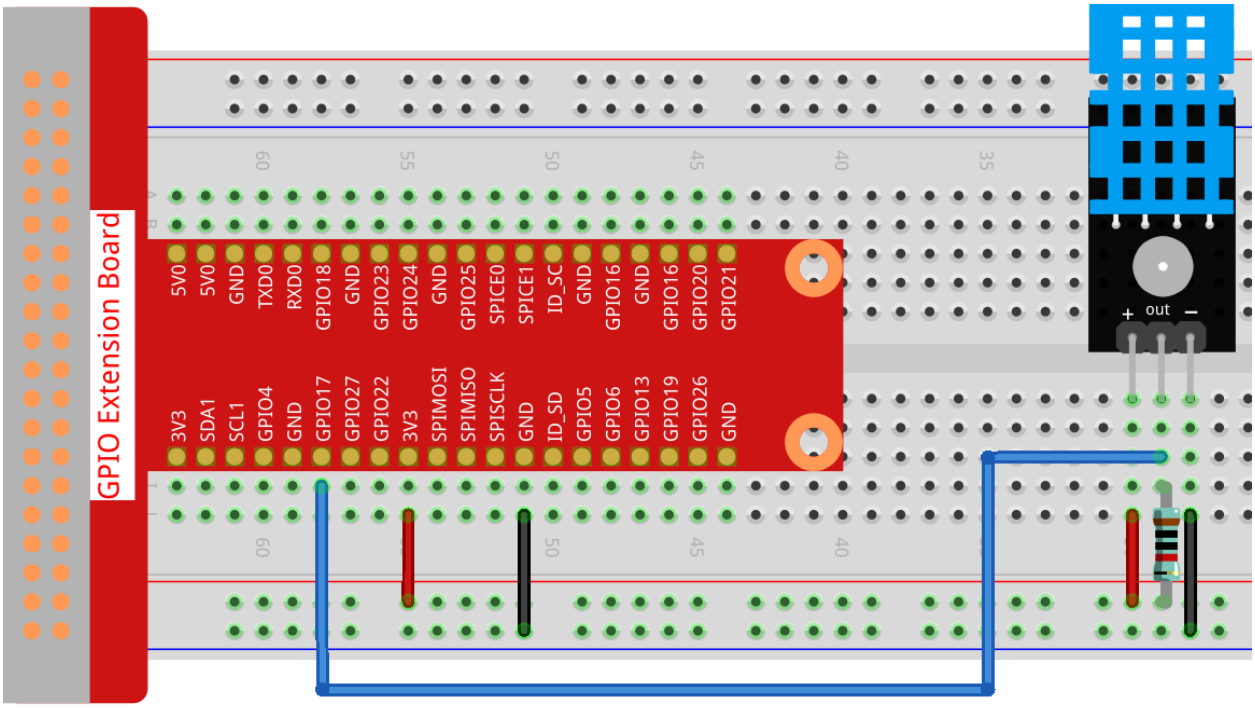
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.3/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.2.3_DHT.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem die Kode ausgeführt wurde, druckt das Programm die von DHT11 erfasste Temperatur und Luftfeuchtigkeit auf dem Computerbildschirm.

Code

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

#define maxTim 85
#define dhtPin 0

int dht11_dat[5] = {0,0,0,0,0};

void readDht11() {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit
    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // pull pin down for 18 milliseconds
    pinMode(dhtPin, OUTPUT);
    digitalWrite(dhtPin, LOW);
    delay(18);
    // then pull it up for 40 microseconds
    digitalWrite(dhtPin, HIGH);
    delayMicroseconds(40);
    // prepare to read the pin
    pinMode(dhtPin, INPUT);

    // detect change and read data
    for ( i=0; i< maxTim; i++) {
        counter = 0;
        while (digitalRead(dhtPin) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 255) {
                break;
            }
        }
        laststate = digitalRead(dhtPin);

        if (counter == 255) break;
        // ignore first 3 transitions
        if ((i >= 4) && (i%2 == 0)) {
            // shove each bit into the storage bytes
            dht11_dat[j/8] <<= 1;
            if (counter > 50)
                dht11_dat[j/8] |= 1;
            j++;
        }
    }
    // check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
    // print it out if data is good
    if ((j >= 40) &&
            (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] + dht11_
↪dat[3]) & 0xFF)) ) {
        Fah = dht11_dat[2] * 9. / 5. + 32;
        printf("Humidity = %d.%d %% Temperature = %d.%d *C (%.1f *F)\n",
                dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], Fah);
    }
}

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
    while (1) {
        readDht11();
        delay(500); // wait 1sec to refresh
    }
    return 0 ;
}
```

Code Erklärung

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

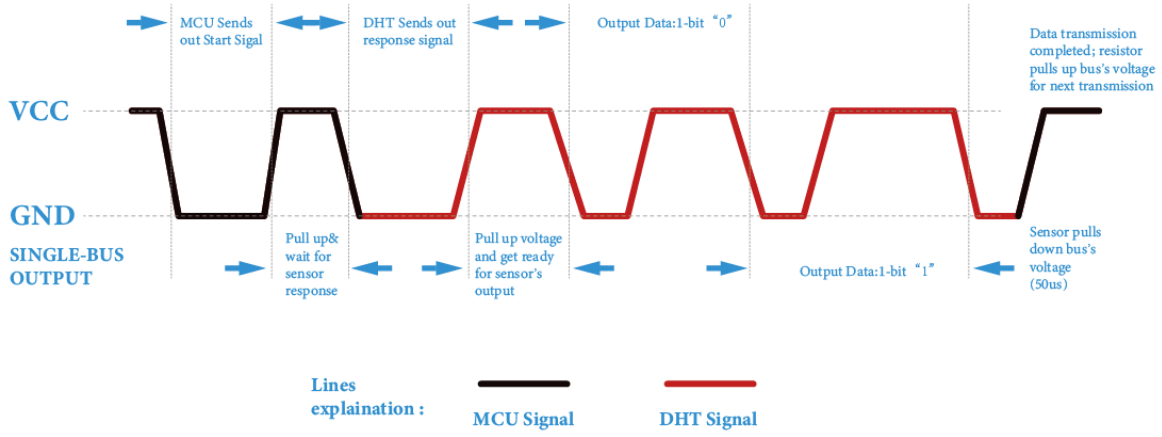
Diese Funktion wird verwendet, um die Funktion von DHT11 zu realisieren.

Es kann im Allgemeinen in 3 Teile unterteilt werden:

1. Bereiten Sie sich darauf vor, den Stift zu lesen:

```
// 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);
```

Sein Kommunikationsfluss wird durch den Arbeitszeitpunkt bestimmt.



Wenn DHT11 startet, sendet die MCU ein Signal mit niedrigem Niveau und hält das Signal dann für 40us auf hohem

Niveau. Danach beginnt die Erkennung des Zustands der externen Umgebung.

2. Daten lesen:

```
// detect change and read data
for ( i=0; i< maxTim; i++) {
        counter = 0;
        while (digitalRead(dhtPin) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 255) {
                break;
            }
        }
        laststate = digitalRead(dhtPin);
        if (counter == 255) break;
        // ignore first 3 transitions
        if ((i >= 4) && (i%2 == 0)) {
            // shove each bit into the storage bytes
            dht11_dat[j/8] <<= 1;
            if (counter > 50)
                dht11_dat[j/8] |= 1;
            j++;
        }
    }
```

Die Schleife speichert die erkannten Daten im Array `dht11_dat[]` . DHT11 überträgt Daten von jeweils 40 Bit. Die ersten 16 Bits beziehen sich auf die Luftfeuchtigkeit, die mittleren 16 Bits auf die Temperatur und die letzten acht Bits werden zur Überprüfung verwendet. Das Datenformat ist:

`8bit Feuchtigkeits-Integer-Daten + 8bit Feuchtigkeits-Dezimaldaten `` + ``8bit Temperatur-Integer-Daten + 8bit Temperatur-Dezimaldaten + 8bit Prüfbit` .

3. Luftfeuchtigkeit und Temperatur drucken.

```
// 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);
}
```

Wenn der Datenspeicher bis zu 40 Bit beträgt, überprüfen Sie die Gültigkeit der Daten über das Prüfbit (`dht11_dat[4]`), und drucken Sie dann Temperatur und Luftfeuchtigkeit aus.

Wenn die empfangenen Daten beispielsweise `00101011``(8-Bit-Wert der Feuchtigkeits-Ganzzahl) ``00000000` (8-Bit-Wert der Feuchtigkeits-Dezimalzahl) `00111100` (8-Bit-Wert der Temperatur-Ganzzahl) `00000000` (8-Bit-Wert der Temperatur-Dezimalzahl) `01100111` (Bit prüfen)

Berechnung:

00101011+00000000+00111100+00000000=01100111.

Das Endergebnis ist gleich mit den Prüfbitdaten, dann sind die empfangenen Daten korrekt:

Luftfeuchtigkeit =43%, Temperatur =60*C.

Wenn es nicht gleich mit den Prüfbitdaten ist, ist die Datenübertragung nicht normal und die Daten werden erneut empfangen.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.3_DHT.py
```

Nachdem die Kode ausgeführt wurde, druckt das Programm die von DHT11 erfasste Temperatur und Luftfeuchtigkeit auf dem Computerbildschirm.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

dhtPin = 17

GPIO.setmode(GPIO.BCM)

MAX_UNCHANGE_COUNT = 100

STATE_INIT_PULL_DOWN = 1
STATE_INIT_PULL_UP = 2
STATE_DATA_FIRST_PULL_DOWN = 3
STATE_DATA_PULL_UP = 4
STATE_DATA_PULL_DOWN = 5

def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)

    unchanged_count = 0
    last = -1
    data = []
    while True:
        current = GPIO.input(dhtPin)
        data.append(current)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > MAX_UNCHANGE_COUNT:
                break

    state = STATE_INIT_PULL_DOWN

    lengths = []
    current_length = 0

    for current in data:
        current_length += 1

        if state == STATE_INIT_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_INIT_PULL_UP
            else:
                continue
        if state == STATE_INIT_PULL_UP:
            if current == GPIO.HIGH:
                state = STATE_DATA_FIRST_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_FIRST_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_DATA_PULL_UP
            else:
                continue
        if state == STATE_DATA_PULL_UP:
            if current == GPIO.HIGH:
                current_length = 0
                state = STATE_DATA_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_PULL_DOWN:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_PULL_UP
            else:
                continue
    if len(lengths) != 40:
        #print ("Data not good, skip")
        return False

    shortest_pull_up = min(lengths)
    longest_pull_up = max(lengths)
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []
    the_bytes = []
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    byte = 0

    for length in lengths:
        bit = 0
        if length > halfway:
            bit = 1
        bits.append(bit)
    #print ("bits: %s, length: %d" % (bits, len(bits)))
    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0
    #print (the_bytes)
    checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False

    return the_bytes[0], the_bytes[2]

def main():

    while True:
        result = readDht11()
        if result:
            humidity, temperature = result
            print ("humidity: %s %%,  Temperature: %s C`" % (humidity, temperature))
        time.sleep(1)

def destroy():
    GPIO.cleanup()

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

Code Erklärung

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    unchanged_count = 0
    last = -1
    data = []
    #...
```

Diese Funktion wird verwendet, um die Funktionen von DHT11 zu implementieren. Es speichert die erkannten Daten im Array `the_bytes[]` . DHT11 überträgt Daten von jeweils 40 Bit. Die ersten 16 Bits beziehen sich auf die Luftfeuchtigkeit, die mittleren 16 Bits auf die Temperatur und die letzten acht Bits werden zur Überprüfung verwendet. Das Datenformat ist:

`8bit Feuchtigkeits-Integer-Daten + 8bit Feuchtigkeits-Dezimaldaten `` + ``8bit Temperatur-Integer-Daten` + `8bit Temperatur-Dezimaldaten` + `8bit Prüfbit` .

Wenn die Gültigkeit über das Prüfbit erkannt wird, gibt die Funktion zwei Ergebnisse zurück: 1. Fehler; 2. Luftfeuchtigkeit und Temperatur.

```
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]
```

Wenn die empfangenen Daten beispielsweise `00101011``(8-Bit-Wert der Feuchtigkeits-Ganzzahl) ``00000000` (8-Bit-Wert der Feuchtigkeits-Dezimalzahl) `00111100` (8-Bit-Wert der Temperatur-Ganzzahl) `00000000` (8-Bit-Wert der Temperatur-Dezimalzahl) `01100111` (Bit prüfen)

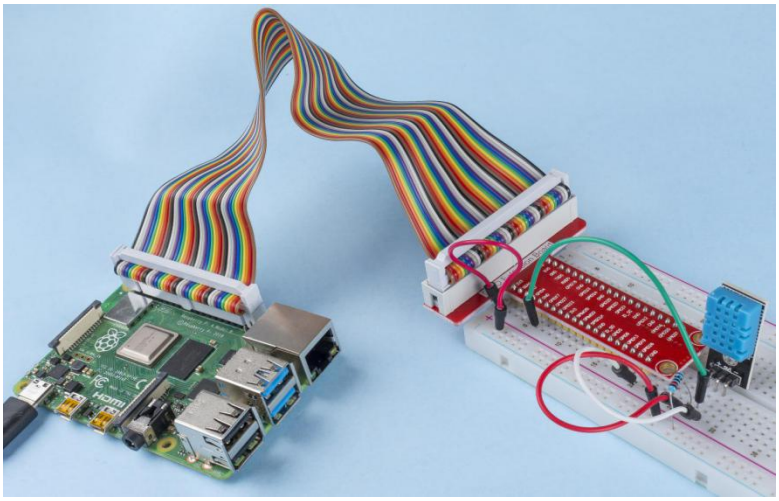
Berechnung:

00101011+00000000+00111100+00000000=01100111.

Wenn das Endergebnis den Prüfbitdaten entspricht, ist die Datenübertragung abnormal: return False.

Wenn das Endergebnis den Prüfbitdaten entspricht, sind die empfangenen Daten korrekt, dann werden `the_bytes[0]` und `the_bytes[2]` zurückgegeben und **Humidity = 43% , Temperature = 60 °C** ausgegeben.

Phänomen Bild

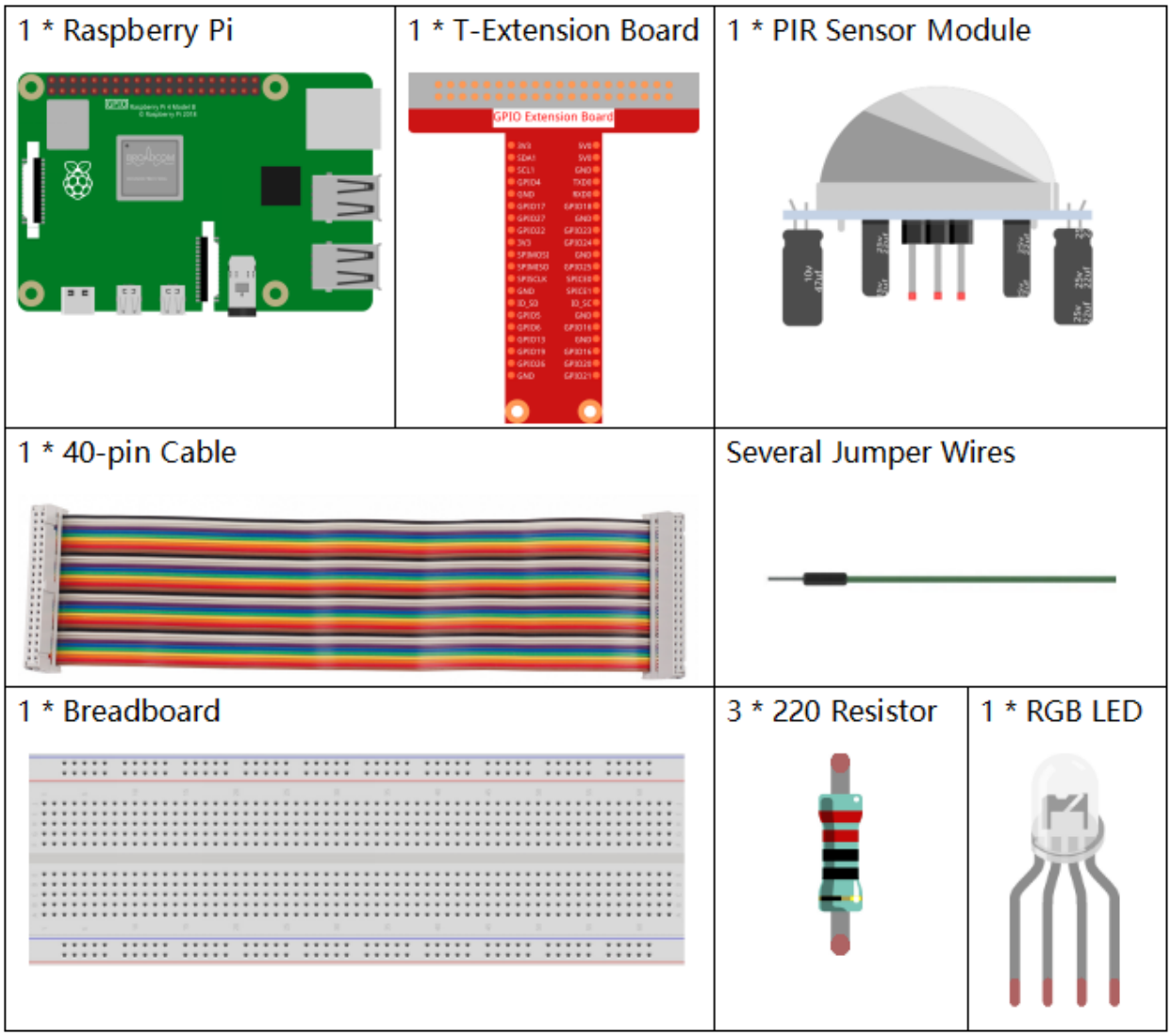


1.8.10 2.2.4 PIR

Einführung

In diesem Projekt werden wir ein Gerät unter Verwendung der pyroelektrischen Infrarotsensoren des menschlichen Körpers herstellen. Wenn sich jemand der LED nähert, leuchtet die LED automatisch auf. Wenn nicht, geht das Licht aus. Dieser Infrarot-Bewegungssensor ist ein Typ Sensor, der das von Mensch und Tier emittierte Infrarot erfassen kann.

Komponenten

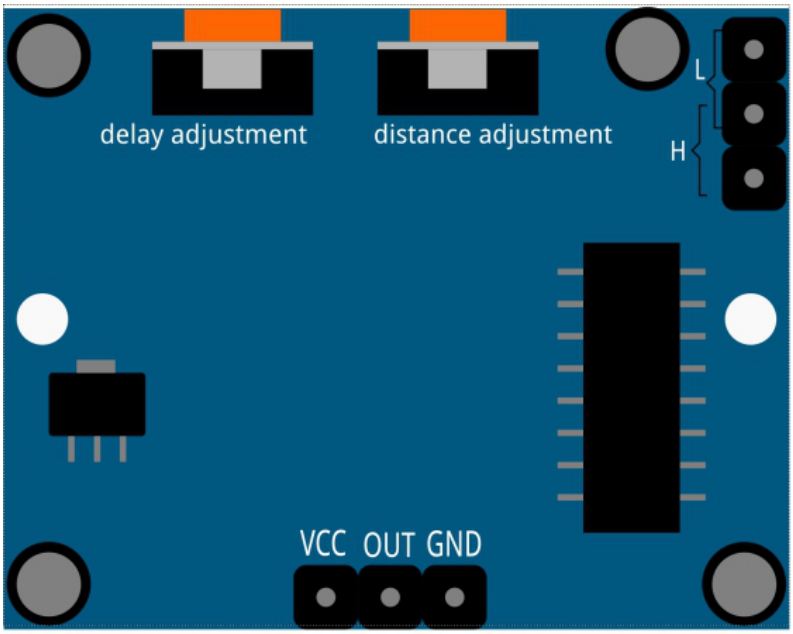


Prinzip

Der PIR-Sensor erfasst Infrarot-Wärmestrahlung, mit der das Vorhandensein von Organismen erfasst werden kann, die Infrarot-Wärmestrahlung emittieren.

Der PIR-Sensor ist in zwei Steckplätze unterteilt, die an einen Differenzverstärker angeschlossen sind. Wenn sich ein stationäres Objekt vor dem Sensor befindet, empfangen die beiden Schlitze die gleiche Strahlungsmenge und der Ausgang ist Null. Wenn sich ein sich bewegendes Objekt vor dem Sensor befindet, empfängt einer der Schlitze mehr Strahlung als der andere, wodurch der Ausgang stark oder niedrig schwankt. Diese Änderung der Ausgangsspannung ist ein Ergebnis der Bewegungserkennung.

Nach dem Verdrahten des Sensormoduls erfolgt eine einminütige Initialisierung. Während der Initialisierung wird das Modul in Intervallen 0 bis 3 Mal ausgegeben. Dann befindet sich das Modul im Standby-Modus. Bitte halten Sie die Interferenz von Lichtquellen und anderen Quellen von der Oberfläche des Moduls fern, um Fehlfunktionen durch das Störsignal zu vermeiden. Und Sie sollten das Modul besser ohne zu viel Wind verwenden, da der Wind auch den Sensor stören kann.



Abstandseinstellung

Durch Drehen des Knopfes des Potentiometers zur Entfernungseinstellung im Uhrzeigersinn vergrößert sich der Bereich der Erfassungsentfernung und der maximale Erfassungsentfernungsbereich beträgt etwa 0 bis 7 Meter. Wenn Sie ihn gegen den Uhrzeigersinn drehen, verringert sich die Reichweite der Erfassungsentfernung, und die minimale Reichweite der Erfassungsentfernung beträgt etwa 0 bis 3 Meter.

Verzögerungseinstellung

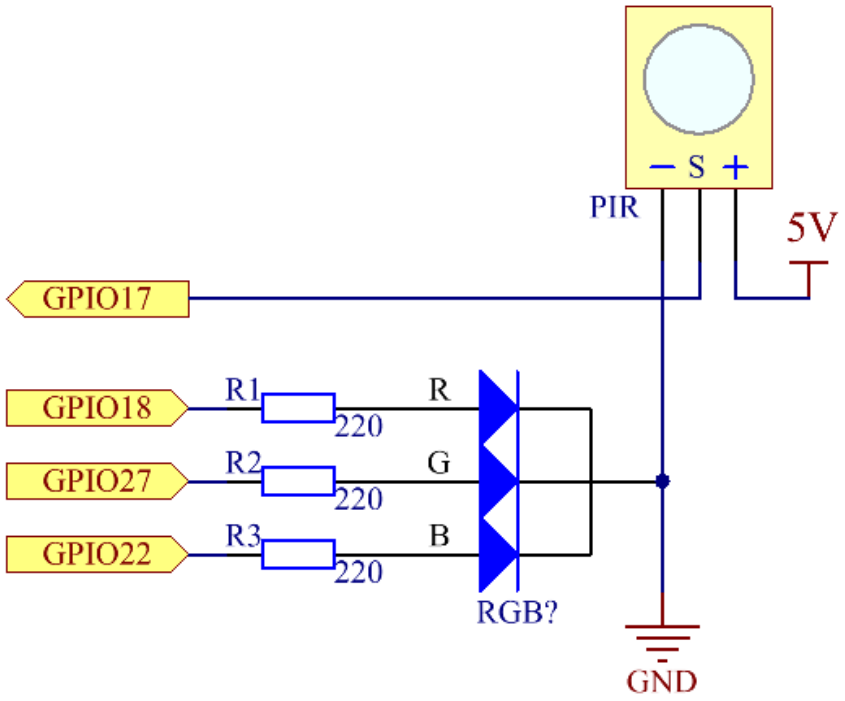
Drehen Sie die Taste des Potentiometers für die Verzögerungseinstellung im Uhrzeigersinn. Sie können auch sehen, wie die Erfassungsverzögerung zunimmt. Das Maximum der Erfassungsverzögerung kann bis zu 300 s erreichen. Im Gegenteil, wenn Sie es gegen den Uhrzeigersinn drehen, können Sie die Verzögerung um mindestens 5 Sekunden verkürzen.

Zwei Triggermodus: (Auswahl verschiedener Modus mit der Überbrückungskappe).

- **H:** Wiederholbarer Triggermodus, nachdem der menschliche Körper erfasst wurde, gibt das Modul einen hohen Niveau aus. Wenn während der nachfolgenden Verzögerungszeit jemand den Erfassungsbereich betritt, bleibt der Ausgang auf dem hohen Niveau.
- **L:** Nicht wiederholbarer Triggermodus, gibt einen hohen Pegel aus, wenn der menschliche Körper erfasst wird. Nach der Verzögerung wechselt der Ausgang automatisch von High- auf Low-Pegel.

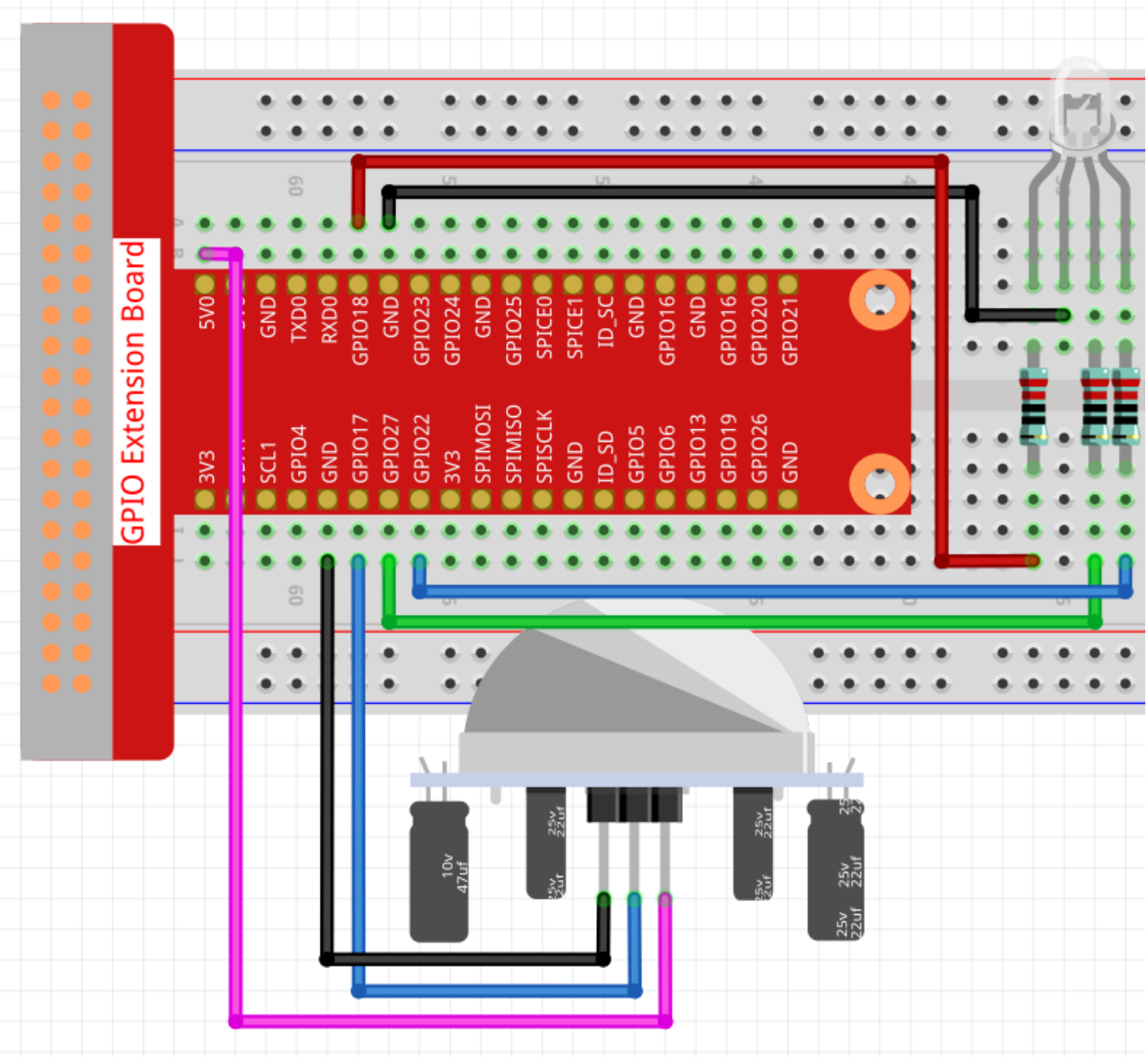
Schematische Darstellung

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.4/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.2.4_PIR.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

After the code runs, PIR detects surroundings and let RGB LED glow yellow if it senses someone walking by. There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.

Code

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

#define pirPin    0     //the pir connect to GPIO0
#define redPin    1
#define greenPin  2
#define bluePin   3

void ledInit(void){
    softPwmCreate(redPin,  0, 100);
    softPwmCreate(greenPin,0, 100);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
}
```

Code Erklärung

```
void ledInit(void);
void ledColorSet(uchar r_val, uchar g_val, uchar b_val);
```

Mit dieser Kode wird die Farbe der RGB-LED eingestellt. Weitere Informationen finden Sie unter *1.1.2 RGB-LED* for more details.

```
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;
}
```

Wenn PIR das menschliche Infrarotspektrum erkennt, sendet die RGB-LED das gelbe Licht aus. Wenn nicht, wird das blaue Licht ausgesendet.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.4_PIR.py
```

After the code runs, PIR detects surroundings and let RGB LED glow yellow if it senses someone walking by. There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen

Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import time

rgbPins = {'Red':18, 'Green':27, 'Blue':22}
pirPin = 17    # the pir connect to pin17

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

    # Set all led as pwm channel and frequece to 2KHz
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)

    # Set all begin with value 0
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

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

# Define a function to set up colors
def setColor(color):
# configures the three LEDs' luminance with the inputted color value .
    # Devide colors from 'color' veriable
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0
    # Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

    #Assign the mapped duty cycle value to the corresponding PWM channel to change the␣
↪luminance.
    p_R.ChangeDutyCycle(R_val)
    p_G.ChangeDutyCycle(G_val)
    p_B.ChangeDutyCycle(B_val)
    #print ("color_msg: R_val = %s, G_val = %s,     B_val = %s"%(R_val, G_val, B_val))

def loop():
    while True:
        pir_val = GPIO.input(pirPin)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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()
```

Code Erklärung

```
rgbPins = {'Red':18, 'Green':27, 'Blue':22}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    # ......
    for i in rgbPins:
        GPIO.setup(rgbPins[i], GPIO.OUT, initial=GPIO.HIGH)
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

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

def setColor(color):
...
```

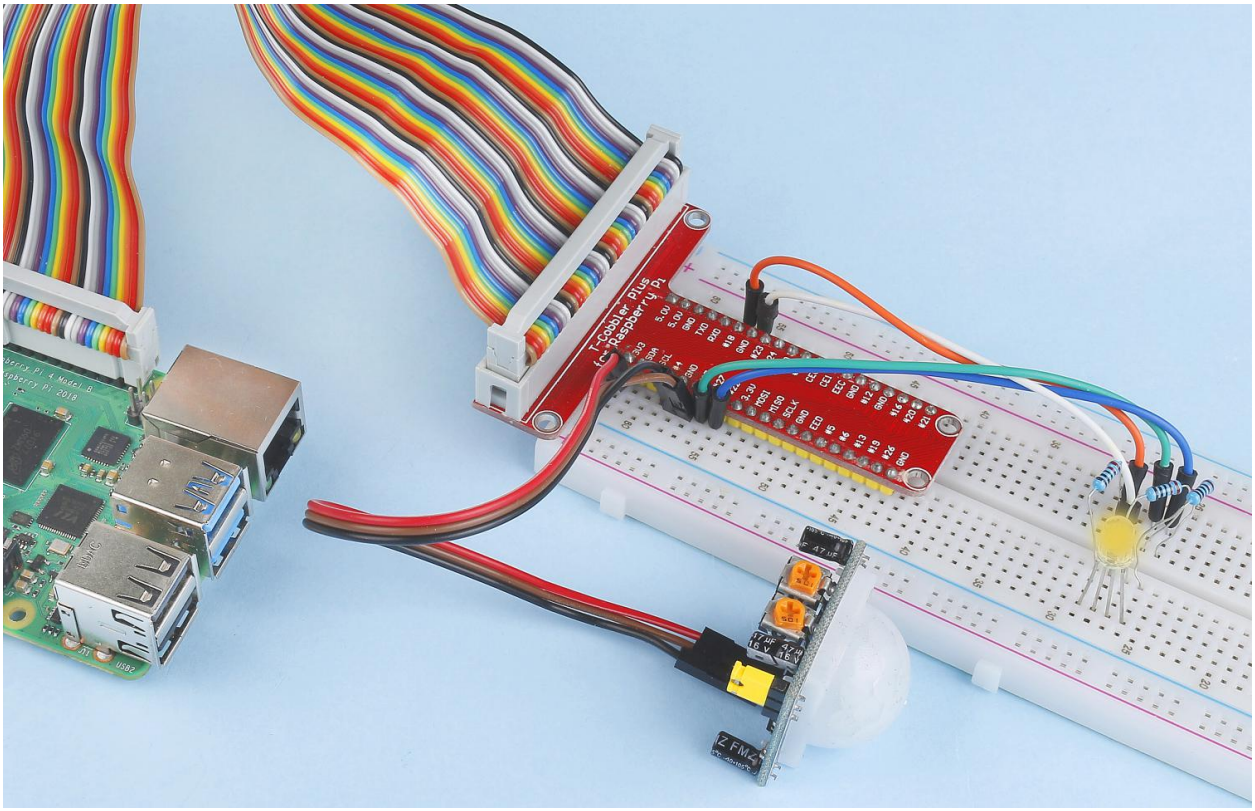
Mit dieser Kode wird die Farbe der RGB-LED eingestellt. Weitere Informationen finden Sie unter *1.1.2 RGB-LED* .

```
def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)
```

Wenn PIR das menschliche Infrarotspektrum erkennt, sendet die RGB-LED das gelbe Licht aus. Wenn nicht, wird das

blaue Licht ausgesendet.

Phänomen Bild

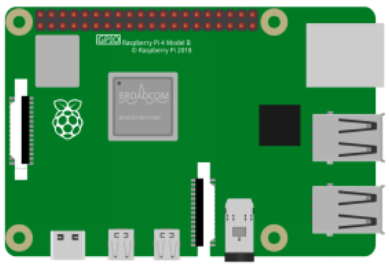
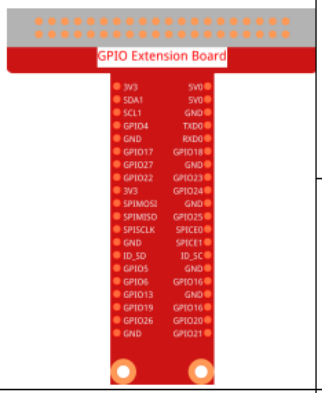
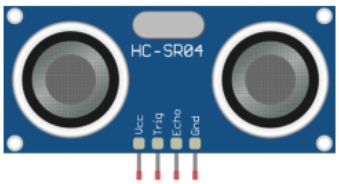
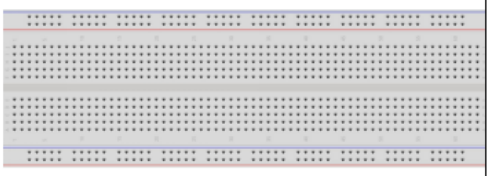


1.8.11 2.2.5 Ultraschallsensormodul

Einführung

Der Ultraschallsensor verwendet Ultraschall, um Objekte genau zu erfassen und die Entfernungen zu messen. Es sendet Ultraschallwellen aus und wandelt sie in elektronische Signale um.

Komponenten

1 * Raspberry Pi 	1 * T-Extension Board 	1 * HC SR04 
		Several Jumper Wires
1 * 40-pin Cable		1 * Breadboard 

Prinzip

Ultraschall

Das Ultraschall-Entfernungsmessmodul bietet eine berührungslose Messfunktion von 2cm bis 400 cm, und die Entfernungsgenauigkeit kann bis zu 3mm betragen. Es kann sicherstellen, dass das Signal innerhalb von 5m stabil ist und das Signal nach 5m allmählich geschwächt wird, bis die 7m-Position verschwindet.

Das Modul enthält Ultraschallsender, Empfänger und Steuerschaltung. Die Grundprinzipien sind wie folgt:

(1)Use an IO flip-flop to process a high level signal of at least 10us.

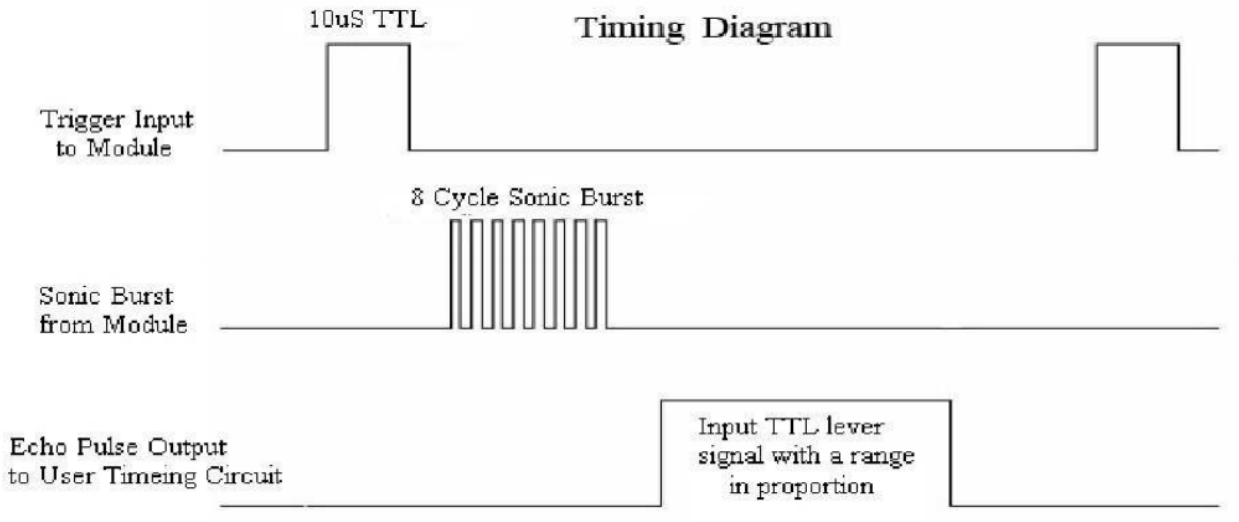
(2)The module automatically sends eight 40khz and detects if there is a pulse signal return.

(3)If the signal returns, passing the high level, the high output IO duration is the time from the transmission of the ultrasonic wave to the return of it. Here, test distance = (high time x sound speed (340 m / s) / 2.

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

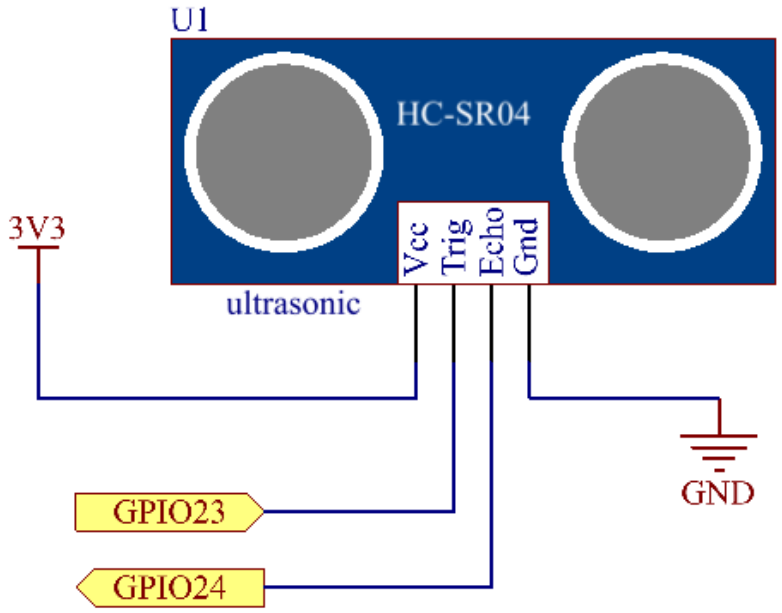
Das Zeitdiagramm ist unten dargestellt. Sie müssen nur einen kurzen 10us-Impuls für den Triggereingang liefern, um die Entfernungsmessung zu starten. Anschließend sendet das Modul einen Ultraschallstoß von 8 Zyklen bei 40 kHz und erhöht sein Echo. Sie können den Bereich über das Zeitintervall zwischen dem Senden des Triggersignals und dem Empfangen des Echosignals berechnen.

Formel: us / 58 = Zentimeter oder us / 148 = Zoll; oder: der Bereich = hohe Niveauzeit * Geschwindigkeit (340M/S) / 2; Es wird empfohlen, einen Messzyklus über 60ms zu verwenden, um Signalkollisionen des Triggersignals und des Echosignals zu vermeiden.



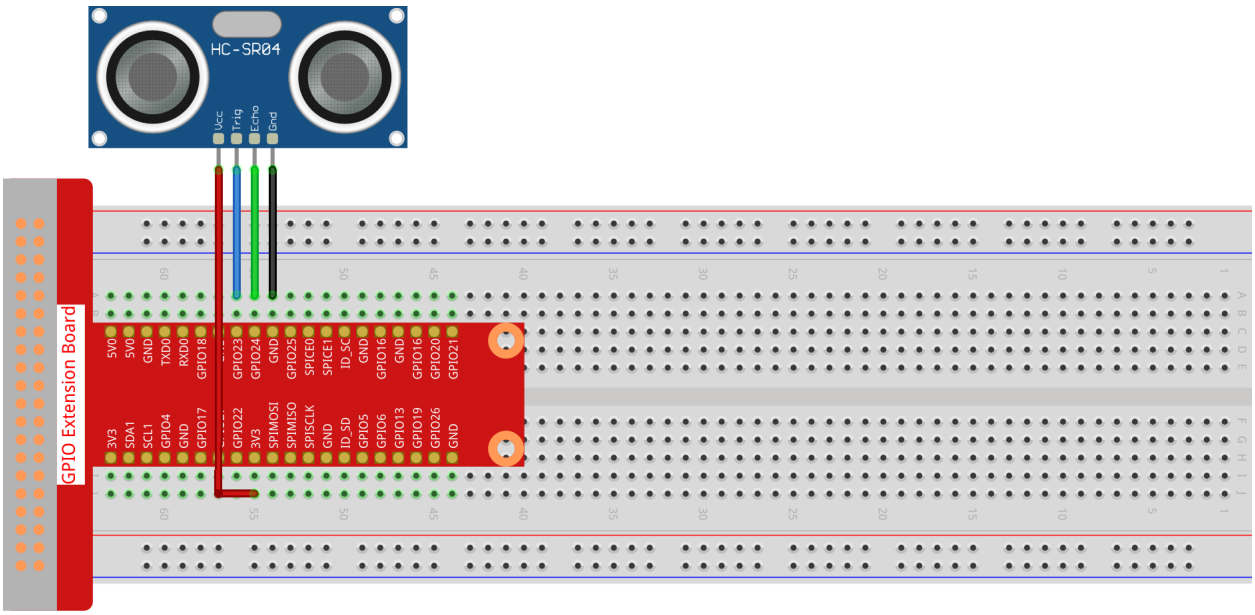
Schematische Darstellung

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.5/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 2.2.5_Ultrasonic.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Wenn die Kode ausgeführt wird, erkennt das Ultraschallsensormodul den Abstand zwischen dem vorausfahrenden Hindernis und dem Modul selbst. Anschließend wird der Abstandswert auf dem Bildschirm gedruckt.

Code

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

#define Trig    4
#define Echo    5

void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
float disMeasure(void)
{
    struct timeval tv1;
    struct timeval tv2;
    long time1, time2;
float dis;

    digitalWrite(Trig, LOW);
    delayMicroseconds(2);

    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);

    while(!(digitalRead(Echo) == 1));
    gettimeofday(&tv1, NULL);

    while(!(digitalRead(Echo) == 0));
    gettimeofday(&tv2, NULL);

    time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
    time2  = tv2.tv_sec * 1000000 + tv2.tv_usec;

    dis = (float)(time2 - time1) / 1000000 * 34000 / 2;

    return dis;
}

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

    ultraInit();

    while(1){
        dis = disMeasure();
        printf("%0.2f cm\n\n",dis);
        delay(300);
    }

    return 0;
}
```

Code Erklärung

```
void ultraInit(void)
{
    pinMode(Echo, INPUT);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    pinMode(Trig, OUTPUT);
}
```

Initialisieren Sie den Ultraschall Pin. Stellen Sie währenddessen Echo auf Eingabe und Trig auf Ausgabe.

```
float disMeasure(void){};
```

Diese Funktion wird verwendet, um die Funktion des Ultraschallsensors durch Berechnung der Rückerkennungsentfernung zu realisieren.

```
struct timeval tv1;
struct timeval tv2;
```

Strukturzeitwert ist eine Struktur, die zum Speichern der aktuellen Zeit verwendet wird. Die vollständige Struktur ist wie folgt:

```
struct timeval
{
__time_t tv_sec;         /* Seconds. */
__suseconds_t tv_usec;  /* Microseconds. */
};
```

Hier repräsentiert `tv_sec` die Sekunden, die Epoch beim Erstellen des Strukturzeitwerts verbracht hat. `tv_usec` steht für Mikrosekunden oder einen Bruchteil von Sekunden.

```
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
```

Ein 10us Ultraschallimpuls wird gesendet.

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);
```

Diese leere Schleife wird verwendet, um sicherzustellen, dass beim Senden des Triggersignals kein störendes Echosignal vorhanden ist, und um dann die aktuelle Zeit zu erhalten.

```
while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);
```

Diese leere Schleife wird verwendet, um sicherzustellen, dass der nächste Schritt erst ausgeführt wird, wenn das Echosignal empfangen wird, und um dann die aktuelle Zeit abzurufen.

```
time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
time2  = tv2.tv_sec * 1000000 + tv2.tv_usec;
```

Konvertieren Sie die von struct timeval gespeicherte Zeit in eine volle Mikrosekundenzeit.

```
dis = (float)(time2 - time1) / 1000000 * 34000 / 2;
```

Die Entfernung wird durch das Zeitintervall und die Geschwindigkeit der Schallausbreitung berechnet. Die Schallgeschwindigkeit in der Luft: 34000cm/s.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: 8 Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.5_Ultrasonic.py
```

Wenn die Kode ausgeführt wird, erkennt das Ultraschallsensormodul den Abstand zwischen dem vorausfahrenden Hindernis und dem Modul selbst. Anschließend wird der Abstandswert auf dem Bildschirm gedruckt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        time.sleep(0.3)

def destroy():
    GPIO.cleanup()

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

Code Erklärung

```
def distance():
```

Diese Funktion wird verwendet, um die Funktion des Ultraschallsensors durch Berechnung der Rückerkennungsentfernung zu realisieren.

```
GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)
```

Dies sendet einen 10us Ultraschallimpuls aus.

```
while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
```

Diese leere Schleife wird verwendet, um sicherzustellen, dass beim Senden des Triggersignals kein störendes Echosignal vorhanden ist, und um dann die aktuelle Zeit zu erhalten.

```
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()
```

Diese leere Schleife wird verwendet, um sicherzustellen, dass der nächste Schritt erst ausgeführt wird, wenn das Echosignal empfangen wird, und um dann die aktuelle Zeit abzurufen.

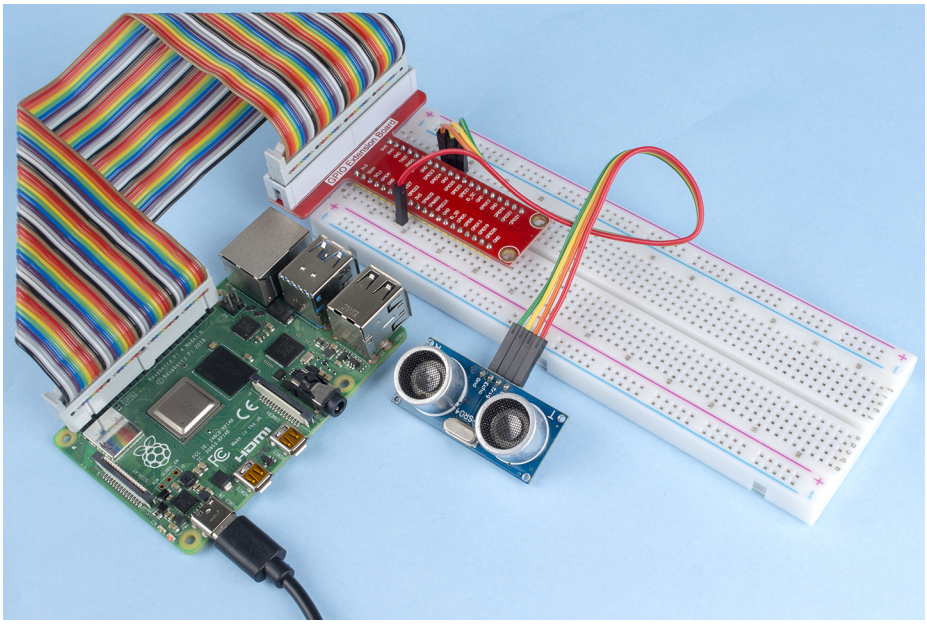
```
during = time2 - time1
```

Führen Sie die Intervallberechnung durch.

```
return during * 340 / 2 * 100
```

Die Entfernung wird unter Berücksichtigung des Zeitintervalls und der Schallausbreitungsgeschwindigkeit berechnet. Die Schallgeschwindigkeit in der Luft: 340 m/s.

Phänomen Bild



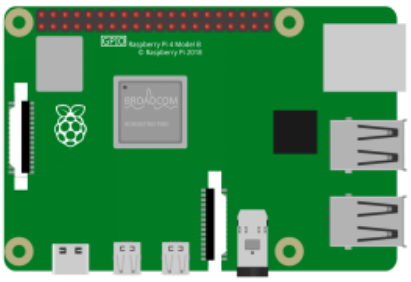
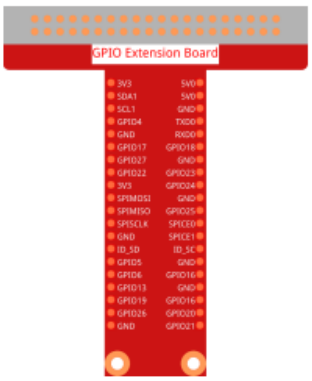
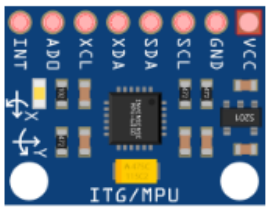
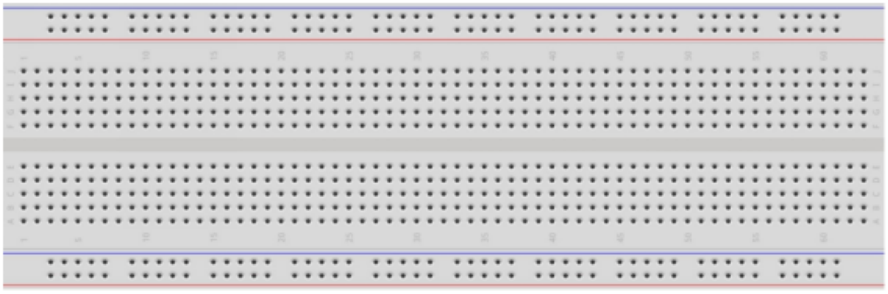
1.8.12 2.2.6 MPU6050-Modul

Einführung

Die MPU-6050 ist das weltweit erste und einzige 6-Achsen-Bewegungsverfolgungsgerät (3-Achsen-Gyroskop und 3-Achsen-Beschleunigungsmesser), das für Smartphones, Tablets und tragbare Sensoren entwickelt wurde, die diese Funktionen aufweisen, einschließlich geringer Leistung, geringer Kosten und hoher Leistung Leistungsanforderungen.

Verwenden Sie in diesem Experiment I2C, um die Werte des dreiachsigen Beschleunigungssensors und des dreiachsigen Gyroskops für MPU6050 zu erhalten und auf dem Bildschirm anzuzeigen.

Komponenten

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

Prinzip

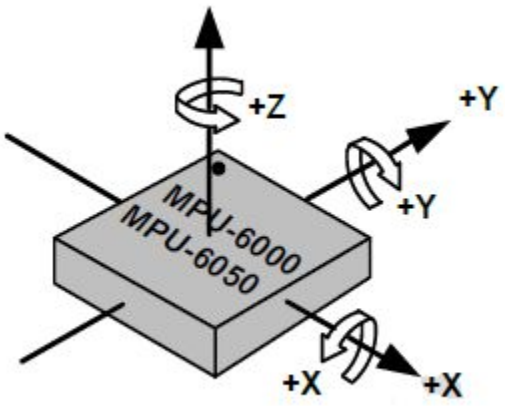
MPU6050

Die MPU-6050 ist ein 6-Achsen-Bewegungsverfolgungsgerät (kombiniert 3-Achsen-Gyroskop, 3-Achsen-Beschleunigungsmesser).

Seine drei Koordinatensysteme sind wie folgt definiert:

Legen Sie die MPU6050 flach auf den Tisch und stellen Sie sicher, dass das Gesicht mit dem Etikett nach oben zeigt und sich ein Punkt auf dieser Oberfläche in der oberen linken Ecke befindet. Dann ist die aufrechte Richtung nach

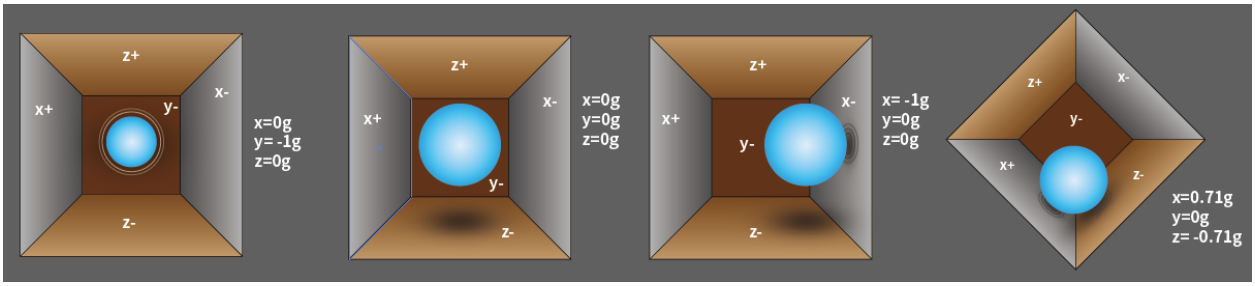
oben die z-Achse des Chips. Die Richtung von links nach rechts wird als X-Achse angesehen. Dementsprechend ist die Richtung von hinten nach vorne als Y-Achse definiert.



3-Achsbeschleunigungsmesser

Der Beschleunigungsmesser arbeitet nach dem Prinzip des piezoelektrischen Effekts, der Fähigkeit bestimmter Materialien, als Reaktion auf angelegte mechanische Beanspruchung eine elektrische Ladung zu erzeugen.

Stellen Sie sich hier eine quaderförmige Kisten vor, in der sich eine kleine Kugel befindet, wie im obigen Bild. Die Wände dieser Kisten bestehen aus piezoelektrischen Kristallen. Immer wenn Sie die Kisten kippen, muss sich der Ball aufgrund der Schwerkraft in Richtung der Neigung bewegen. Die Wand, mit der die Kugel kollidiert, erzeugt winzige piezoelektrische Ströme. Es gibt insgesamt drei Paare gegenüberliegender Wände in einem Quader. Jedes Paar entspricht einer Achse im 3D-Raum: X-, Y- und Z-Achse. Abhängig vom Strom, der von den piezoelektrischen Wänden erzeugt wird, können wir die Neigungsrichtung und ihre Größe bestimmen.



Wir können die MPU6050 verwenden, um ihre Beschleunigung auf jeder Koordinatenachse zu erfassen (im stationären Desktop-Zustand beträgt die Beschleunigung der Z-Achse 1 Schwerkrafteinheit und die X- und Y-Achse 0). Wenn es gekippt ist oder sich in einem schwerelosen / übergewichtigen Zustand befindet, ändert sich der entsprechende Wert.

Es gibt vier Arten von Messbereichen, die programmgesteuert ausgewählt werden können: +/-2g, +/-4g, +/-8g und +/-16g (standardmäßig 2 g), die jeder Genauigkeit entsprechen. Die Werte reichen von -32768 bis 32767.

Der Messwert des Beschleunigungsmessers wird in einen Beschleunigungswert umgewandelt, indem der Messwert vom Messbereich auf den Messbereich abgebildet wird.

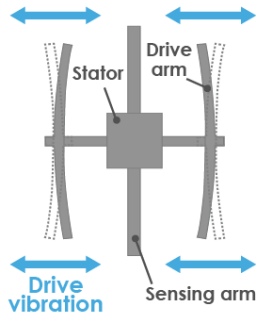
Beschleunigung = (Rohdaten der Beschleunigungsmesserachse / 65536 * voller Beschleunigungsbereich) g

Nehmen Sie als Beispiel die X-Achse, wenn die Rohdaten der X-Achse des Beschleunigungsmessers 16384 sind und der Bereich als +/- 2 g ausgewählt ist:

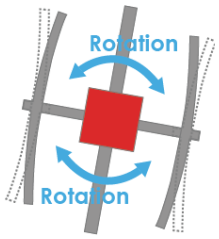
Beschleunigung entlang der X-Achse = (16384/65536 * 4) g = 1g

3-Achsen-Gyroskop

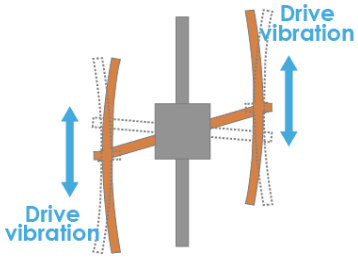
Gyroskope arbeiten nach dem Prinzip der Coriolis-Beschleunigung.. Stellen Sie sich vor, es gibt eine gabelartige Struktur, die sich ständig hin und her bewegt. Es wird mit piezoelektrischen Kristallen an Ort und Stelle gehalten. Immer wenn Sie versuchen, diese Anordnung zu kippen, erfahren die Kristalle eine Kraft in Neigungsrichtung. Dies wird durch die Trägheit der beweglichen Gabel verursacht. Die Kristalle erzeugen somit einen Strom, der mit dem piezoelektrischen Effekt übereinstimmt, und dieser Strom wird verstärkt.



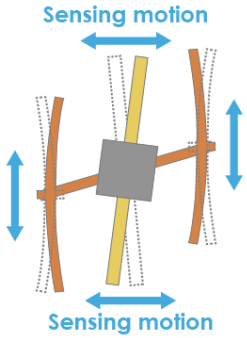
1.Normally, a drive arm vibrates in a certain direction.



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.

Das Gyroskop verfügt außerdem über vier Arten von Messbereichen: +/- 250, +/- 500, +/- 1000, +/- 2000. Die Berechnungsmethode und die Beschleunigung sind grundsätzlich konsistent.

Die Formel zum Umwandeln des Messwerts in die Winkelgeschwindigkeit lautet wie folgt:

Winkelgeschwindigkeit = (Rohdaten der Gyroskopachse / 65536 * Gyroskopbereich im vollen Maßstab) °/s

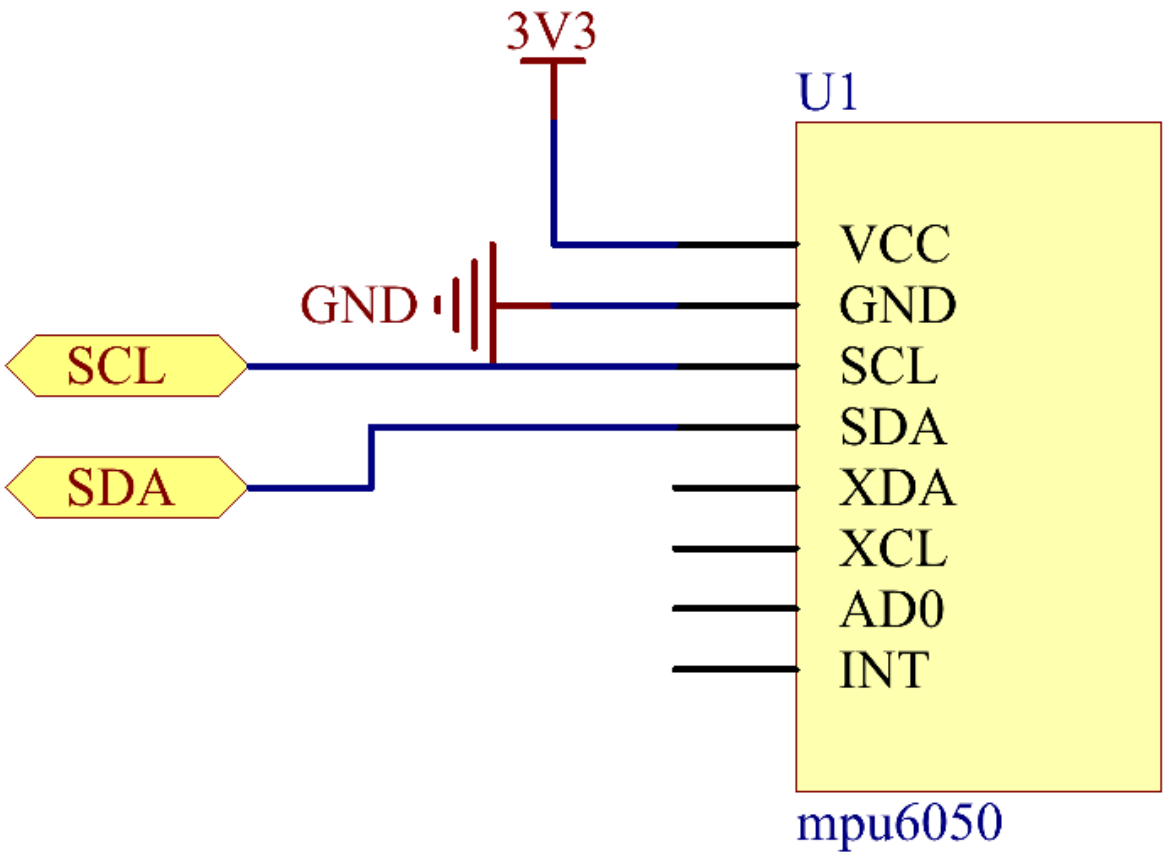
Die X-Achse, zum Beispiel die Rohdaten der X-Achse des Beschleunigungsmessers, ist 16384 und reicht von +/- 250 °/s:

Winkelgeschwindigkeit entlang der X-Achse = (16384/65536 * 500) °/s = 125 °/s

Schematische Darstellung

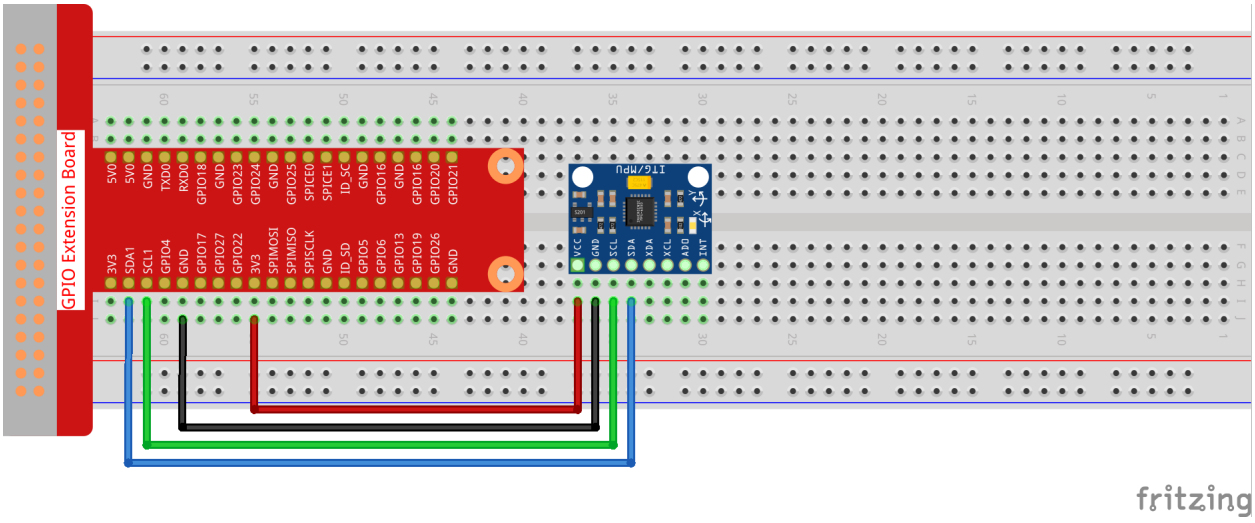
Die MPU6050 kommuniziert mit dem Mikrocontroller über die I2C-Busschnittstelle. Der SDA1 und der SCL1 müssen mit dem entsprechenden Pin verbunden werden.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: I2C einrichten (siehe Anhang. Wenn Sie *I2C-Konfiguration* haben, überspringen Sie diesen Schritt.)

Für Benutzer in C-Sprache

Schritt 3: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.6/
```

Schritt 4: Kompilieren Sie der Kode.

```
gcc 2.2.6_mpu6050.c -lwiringPi -lm
```

Schritt 5: Führen Sie die ausführbare Datei aus.

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

Wenn die Kode ausgeführt wird, werden der Ablenkwinkel der x-Achse, der y-Achse und die Beschleunigung sowie die Winkelgeschwindigkeit auf jeder von der MPU6050 gelesenen Achse nach der Berechnung auf dem Bildschirm gedruckt.

Code

```
#include  <wiringPiI2C.h>
#include <wiringPi.h>
#include  <stdio.h>
#include  <math.h>
int fd;
int acclX, acclY, acclZ;
int gyroX, gyroY, gyroZ;
double acclX_scaled, acclY_scaled, acclZ_scaled;
double gyroX_scaled, gyroY_scaled, gyroZ_scaled;

int read_word_2c(int addr)
{
    int val;
    val = wiringPiI2CReadReg8(fd, addr);
    val = val << 8;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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));

    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);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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));
        delay(100);

        delay(100);
    }
    return 0;
}
```

Code Erklärung

```
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;
}
```

Lesen Sie die von der MPU6050 gesendeten Sensordaten.

```
double get_y_rotation(double x, double y, double z)
{
double radians;
radians = atan2(x, dist(y, z));
return -(radians * (180.0 / M_PI));
}
```

Wir erhalten den Ablenkwinkel auf der Y-Achse.

```
double get_x_rotation(double x, double y, double z)
{
double radians;
radians = atan2(y, dist(x, z));
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
return (radians * (180.0 / M_PI));
}
```

Berechnen Sie den Ablenkwinkel der x-Achse.

```
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);
```

Lesen Sie die Werte der x-Achse, der y-Achse und der z-Achse auf dem Gyroskopsensor, konvertieren Sie die Metadaten in Winkelgeschwindigkeitswerte und drucken Sie sie dann aus.

```
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);
```

Lesen Sie die Werte der x-, y- und z-Achse auf dem Beschleunigungssensor ab, konvertieren Sie die Metadaten in beschleunigte Geschwindigkeitswerte (Schwerkrafteinheit) und drucken Sie sie dann aus.

```
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));
```

Drucken Sie die Ablenkwinkel der x- und y-Achse.

Für Python-Sprachbenutzer

Schritt 3: Gehen Sie zum Ordner der Kode.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.6_mpu6050.py
```

Wenn die Kode ausgeführt wird, werden der Ablenkwinkel der x- und y-Achse sowie die Beschleunigung und Winkelgeschwindigkeit auf jeder von MPU6050 gelesenen Achse nach der Berechnung auf dem Bildschirm gedruckt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import smbus
import math
import time

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

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)


bus = smbus.SMBus(1) # or bus = smbus.SMBus(1) for Revision 2 boards
address = 0x68       # This is the address value read via the i2cdetect command

# Now wake the 6050 up as it starts in sleep mode
bus.write_byte_data(address, power_mgmt_1, 0)

while True:
    time.sleep(0.1)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))

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

    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
    print ("accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled)
    print ("accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled)

    print ("x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled, accel_
↪zout_scaled))
    print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_
↪zout_scaled))

    time.sleep(0.5)
```

Code Erklärung

```
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
```

Lesen Sie die von der MPU6050 gesendeten Sensordaten.

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

Berechnen Sie den Ablenkwinkel der y-Achse.

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

Berechnen Sie den Ablenkwinkel der x-Achse.

```
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))
```

Lesen Sie die Werte der x-Achse, der y-Achse und der z-Achse auf dem Gyroskopsensor, konvertieren Sie die Metadaten in Winkelgeschwindigkeitswerte und drucken Sie sie dann aus.

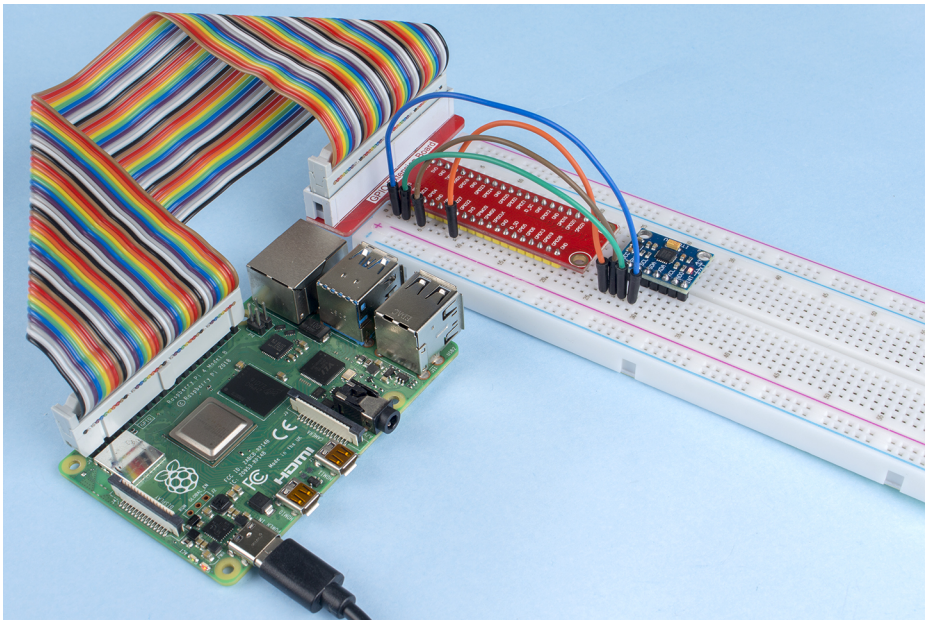
```
accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
accel_zout = read_word_2c(0x3f)

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)
```

Lesen Sie die Werte der x-Achse, der y-Achse und der z-Achse auf dem Gyroskopsensor, konvertieren Sie die Metadaten in Winkelgeschwindigkeitswerte und drucken Sie sie dann aus.

```
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))
```

Drucken Sie die Ablenkwinkel der x- und y-Achse.

Phänomen Bild



1.8.13 2.2.7 MFRC522 RFID-Modul

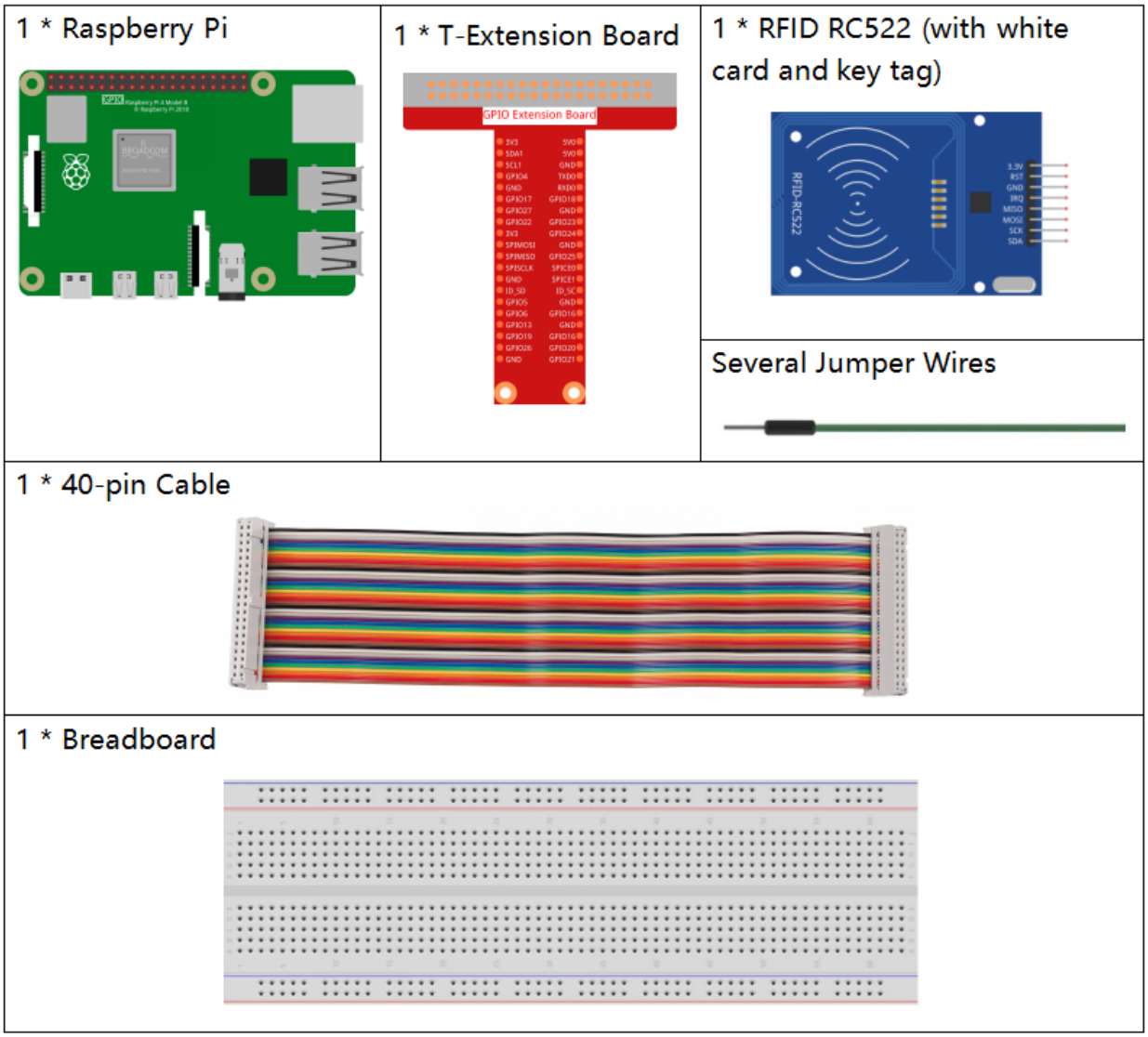
Einführung

Radio Frequency Identification (RFID) bezieht sich auf Technologien, die die drahtlose Kommunikation zwischen einem Objekt (oder Tag) und einem Abfragegerät (oder Lesegerät) verwenden, um solche Objekte automatisch zu verfolgen und zu identifizieren.

Einige der häufigsten Anwendungen für diese Technologie sind Lieferketten für den Einzelhandel, militärische Lieferketten, automatisierte Zahlungsmethoden, Gepäckverfolgung und -verwaltung, Dokumentenverfolgung und pharmazeutische Verwaltung, um nur einige zu nennen.

In diesem Projekt werden wir RFID zum Lesen und Schreiben verwenden.

Komponenten



Prinzip

RFID

Radio Frequency Identification (RFID) bezieht sich auf Technologien, bei denen eine drahtlose Kommunikation zwischen einem Objekt (oder Tag) und einem Abfragegerät (oder Lesegerät) verwendet wird, um solche Objekte automatisch zu verfolgen und zu identifizieren. Die Tag-Übertragungsreichweite ist auf mehrere Meter vom Lesegerät begrenzt. Eine klare Sichtlinie zwischen Lesegerät und Etikett ist nicht unbedingt erforderlich.

Die meisten Etikette enthalten mindestens eine integrierte Schaltung (IC) und eine Antenne. Der Mikrochip speichert Informationen und ist für die Verwaltung der Hochfrequenzkommunikation mit dem Lesegerät verantwortlich. Passive Etiketts haben keine unabhängige Energiequelle und sind auf ein externes elektromagnetisches Signal angewiesen, das vom Lesegerät bereitgestellt wird, um ihren Betrieb zu betreiben. Aktive Etiketts enthalten eine unabhängige Energiequelle, z. B. eine Batterie. Dann können sie eine erhöhte Verarbeitung, Übertragungsfähigkeit und Reichweite aufweisen.

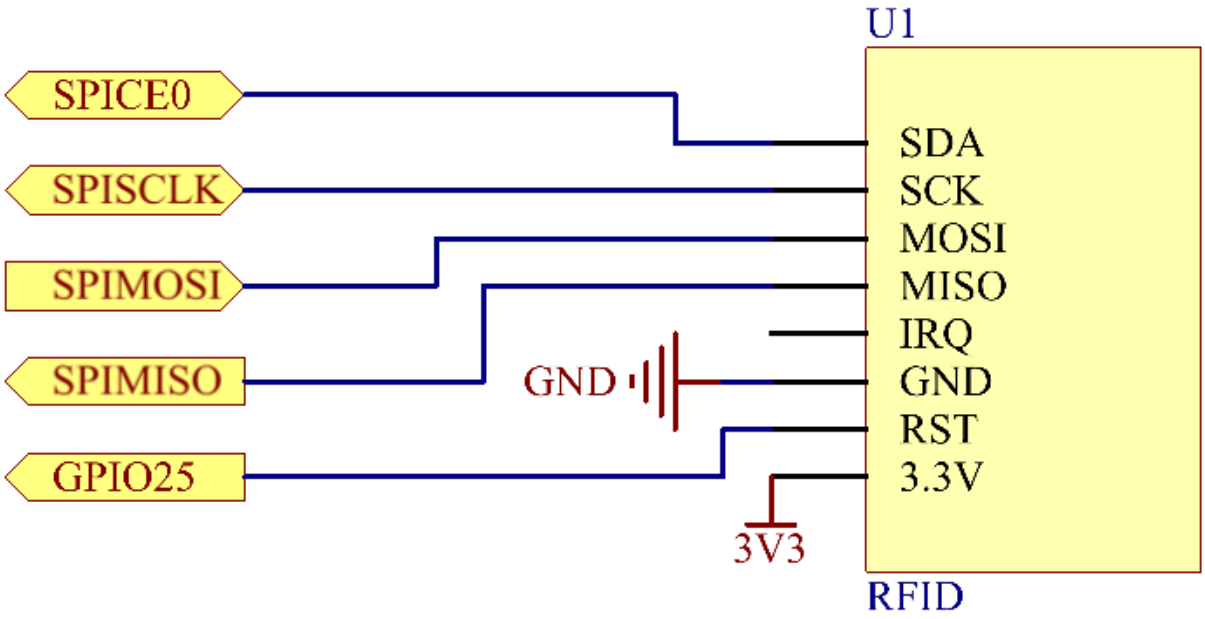
MFRC522

MFRC522 ist eine Art integrierter Lese- und Schreibkartenchip. Es wird üblicherweise im Radio bei 13,56 MHz verwendet. Es wurde von der NXP Company eingeführt und ist ein berührungsloser, kostengünstiger und kleiner Niederspannungs-Kartenchip, der die beste Wahl für intelligente Instrumente und tragbare Handheld-Geräte darstellt.

Der MF RC522 verwendet ein fortschrittliches Modulations- und Demodulationskonzept, das in allen Arten von passiven kontaktlosen Kommunikationsmethoden und -protokollen mit 13,56 MHz vollständig dargestellt wird. Darüber hinaus unterstützt es den schnellen CRYPTO1-Verschlüsselungsalgorithmus zur Überprüfung von MIFARE-Produkten. Der MFRC522 unterstützt auch die berührungslose Hochgeschwindigkeitskommunikation der MIFARE-Serie mit einer bidirektionalen Datenübertragungsrate von bis zu 424 kbit/s. Als neues Mitglied der hochintegrierten 13,56-MHz-Lesekartenserie ist der MF RC522 dem vorhandenen MF RC500 und MF RC530 sehr ähnlich, es gibt jedoch auch große Unterschiede. Es kommuniziert mit dem Host-Computer über die serielle Art und Weise, die weniger Verkabelung erfordert. Sie können zwischen SPI-, I2C- und seriellem UART-Modus (ähnlich wie RS232) wählen, um die Verbindung zu reduzieren, Platz auf der Leiterplatte zu sparen (kleinere Größe) und Kosten zu senken.

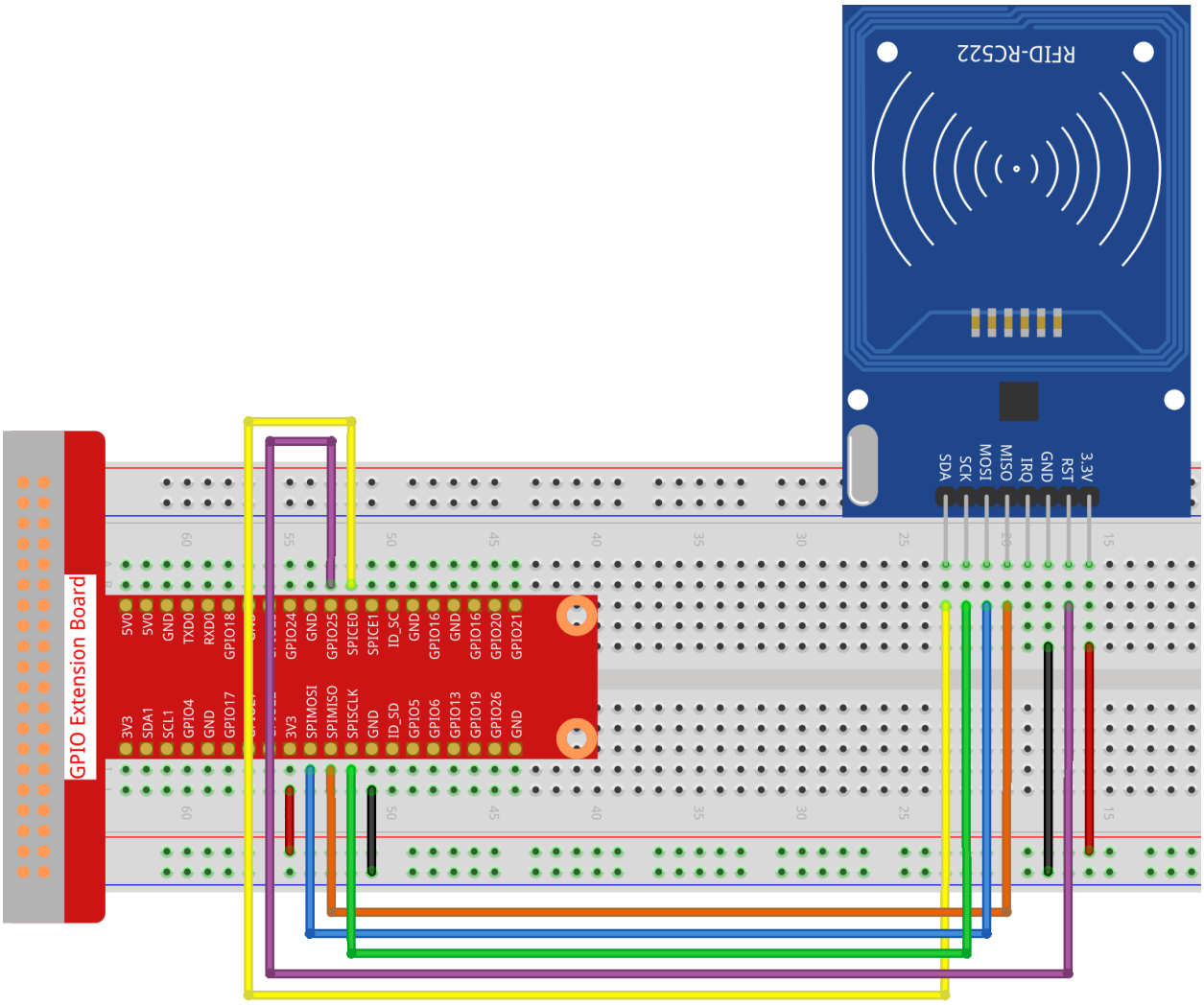
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: SPI einrichten (weitere Informationen finden Sie in Anhang. Wenn Sie *SPI-Konfiguration* haben, überspringen Sie diesen Schritt.)

Für Benutzer in C-Sprache

Schritt 3: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.7/
```

Schritt 4: Kompilieren Sie die Kode.

```
make read
make write
```

Bemerkung: Es gibt zwei Beispiele, mit denen Sie die Karten-ID lesen oder schreiben können. Sie können je nach

Bedarf eines davon auswählen.

Schritt 5: Führen Sie die ausführbare Datei aus.

```
sudo ./read
```

```
sudo ./write
```

Code Erklärung

```
InitRc522();
```

Mit dieser Funktion wird das RFID RC522-Modul initialisiert.

```
uint8_t read_card_data();
```

Diese Funktion wird zum Lesen der Daten der Karte verwendet. Wenn der Lesevorgang erfolgreich ist, wird „1“ zurückgegeben.

```
uint8_t write_card_data(uint8_t *data);
```

Diese Funktion wird verwendet, um die Daten der Karte zu schreiben und gibt „1“ zurück, wenn das Schreiben erfolgreich war. `*data` sind die Informationen, die auf die Karte geschrieben werden.

Für Python-Sprachbenutzer

Schritt 2: Aktivierung der virtuellen Umgebung.

Bemerkung:

- Bevor Sie die Aktivierung durchführen, müssen Sie sicherstellen, dass Sie eine virtuelle Umgebung erstellt haben. Bitte beachten Sie: *Erstellen einer virtuellen Umgebung*.
- Jedes Mal, wenn Sie den Raspberry Pi neu starten oder ein neues Terminal öffnen, müssen Sie erneut den folgenden Befehl ausführen, um die virtuelle Umgebung zu aktivieren.

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

Nachdem die virtuelle Umgebung aktiviert ist, sehen Sie den Umgebungsnamen vor dem Befehlszeilen-Prompt, was darauf hinweist, dass Sie innerhalb der virtuellen Umgebung arbeiten.

Schritt 3: Installation der Bibliotheken.

Die `spidev`-Bibliothek hilft bei der Handhabung von Interaktionen mit SPI und ist ein Schlüsselelement dieses Tutorials, da wir sie für den Raspberry Pi benötigen, um mit dem RFID RC522 zu interagieren.

Führen Sie den folgenden Befehl aus, um `spidev` über `pip` auf Ihrem Raspberry Pi zu installieren.

```
sudo pip3 install spidev
```

Fahren Sie mit der Installation der MFRC522-Bibliothek fort. Die MFRC522-Bibliothek enthält zwei Dateien: `MFRC522.py` und `SimpleMFRC522.py`.

Dabei ist `MFRC522.py` die Realisierung der RFID-RC522-Schnittstelle, diese Bibliothek übernimmt alle schweren Arbeiten bei der Kommunikation mit RFID über die SPI-Schnittstelle des Pi.

SimpleMFRC522.py nimmt die Datei MFRC522.py und vereinfacht sie erheblich, indem sie es Ihnen ermöglicht, nur mit einigen Funktionen anstatt mit einigen Funktionen umzugehen.

```
sudo pip3 install mfrc522
```

Schritt 4: Beenden der virtuellen Umgebung.

Wenn Sie Ihre Arbeit abgeschlossen haben und die virtuelle Umgebung verlassen möchten, führen Sie einfach folgenden Befehl aus:

```
deactivate
```

Damit kehren Sie zur globalen Python-Umgebung des Systems zurück.

Schritt 5: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/2.2.7
```

Schritt 6: Führen Sie die ausführbare Datei aus.

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

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

Bemerkung:

- Nach dem Ausführen von 2.2.7_read.py werden die id und der text Ihrer Karte in der Shell ausgegeben.
- Nachdem Sie 2.2.7_write.py ausgeführt haben, müssen Sie zuerst eine Nachricht schreiben, zur Bestätigung Enter drücken und schließlich Ihre Karte auf das MFRC522 RFID-Modul legen, um das Schreiben abzuschließen.

---

code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie davinci-kit-for-raspberry-pi/python gehen.

```
# 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()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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:')
        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()
```

Code Erklärung

```
reader = SimpleMFRC522()
```

Instanziieren Sie die Klasse `SimpleMFRC522()` .

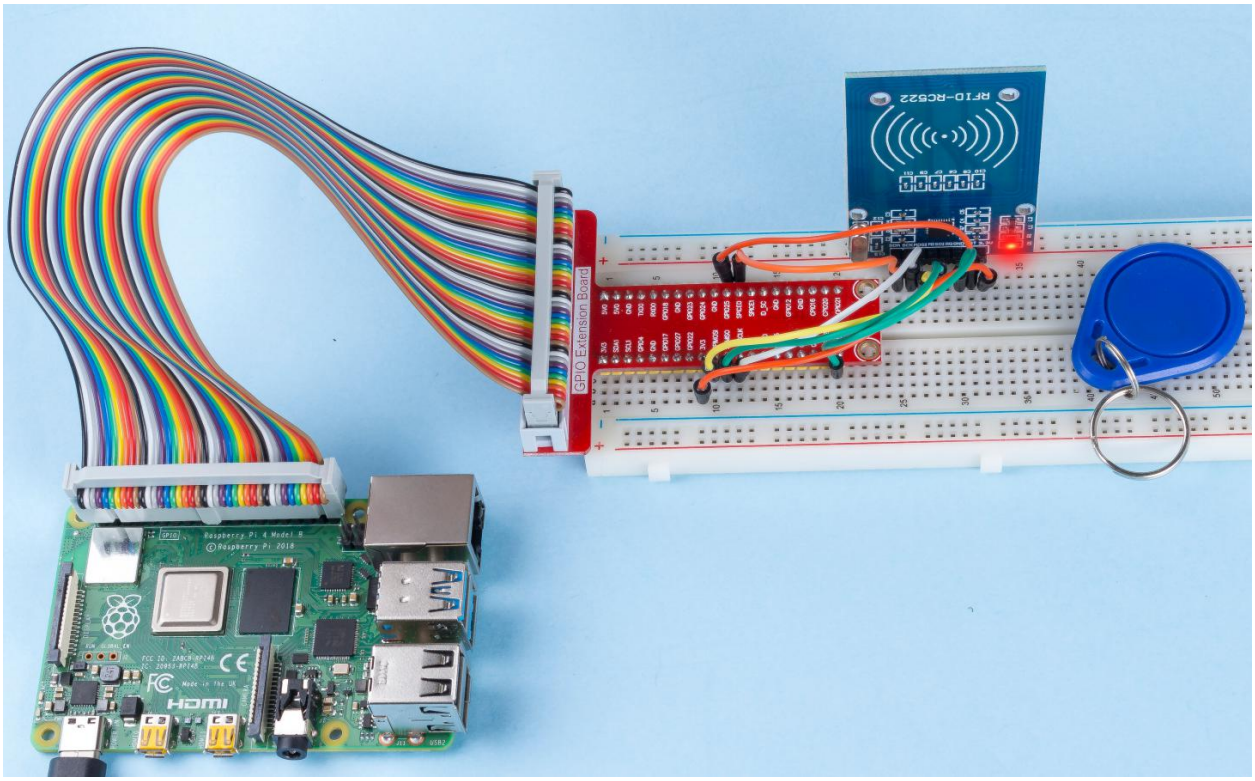
```
reader.read()
```

Diese Funktion dient zum Auslesen von Kartendaten. Wenn das Lesen erfolgreich ist, werden ID und Text zurückgegeben.

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

Diese Funktion wird verwendet, um Informationen auf die Karte zu schreiben. Drücken Sie die Taste `Enter` , um das Schreiben zu beenden. `Text` ist die Information, die auf die Karte geschrieben werden soll.

Phänomen Bild



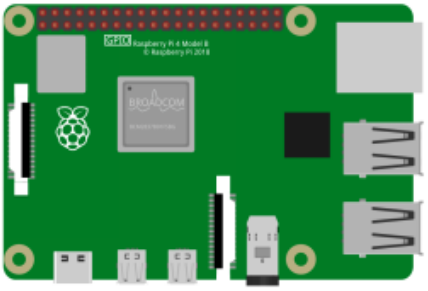
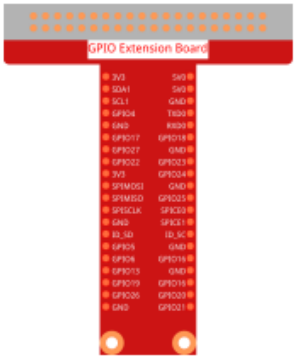
1.9 3 Erweiterung (nicht für Pi 5)

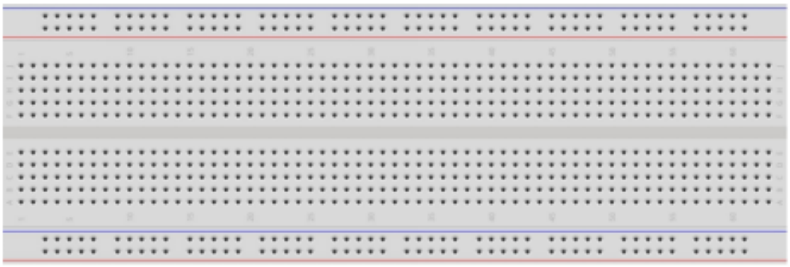
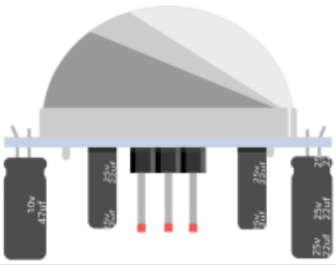
1.9.1 3.1.1 Zählgerät

Einführung

Hier werden wir ein Zählersystem mit Nummeranzeige herstellen, das aus einem PIR-Sensor und einer 4-stelligen Segmentanzeige besteht. Wenn der PIR feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1. Mit diesem Zähler können Sie die Anzahl der Personen zählen, die durch den Durchgang gehen.

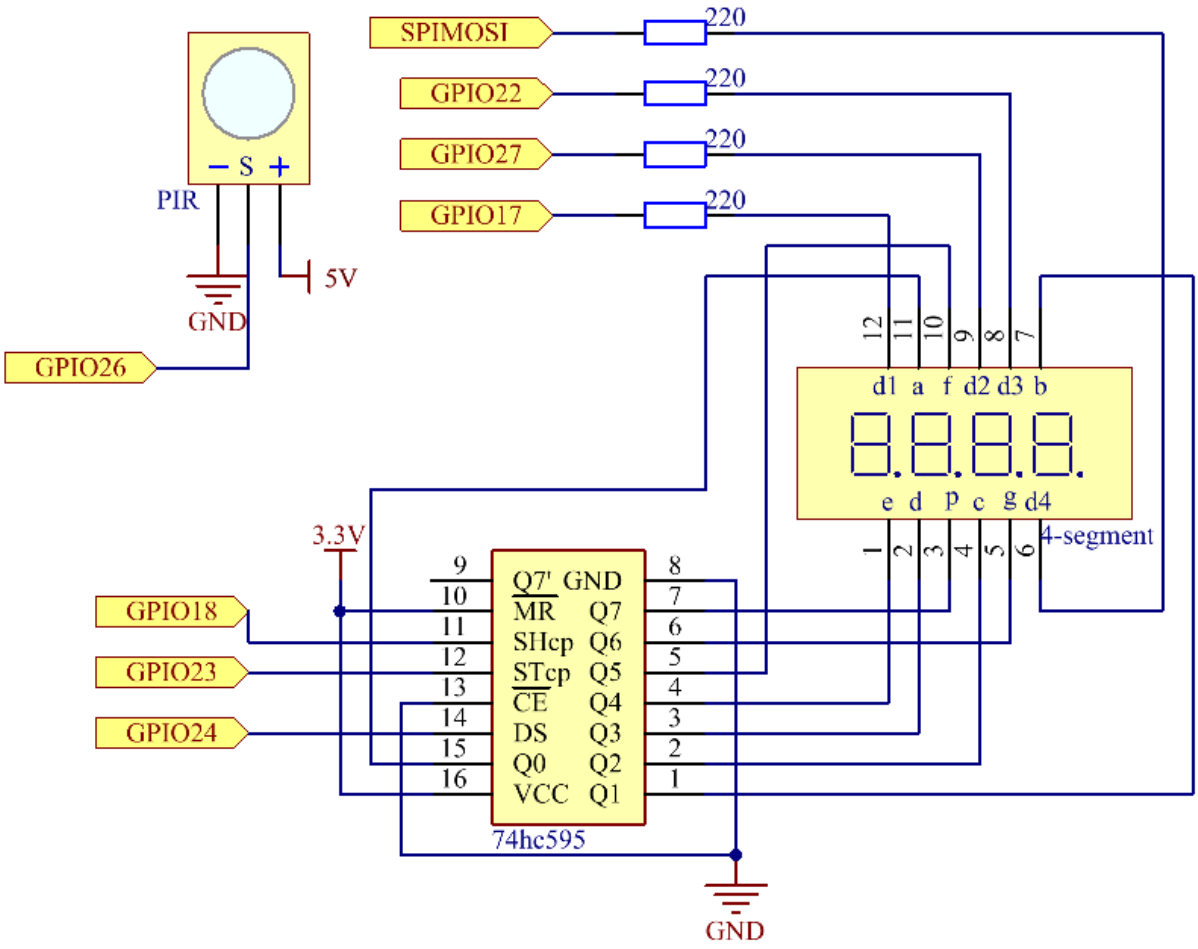
Komponenten

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 

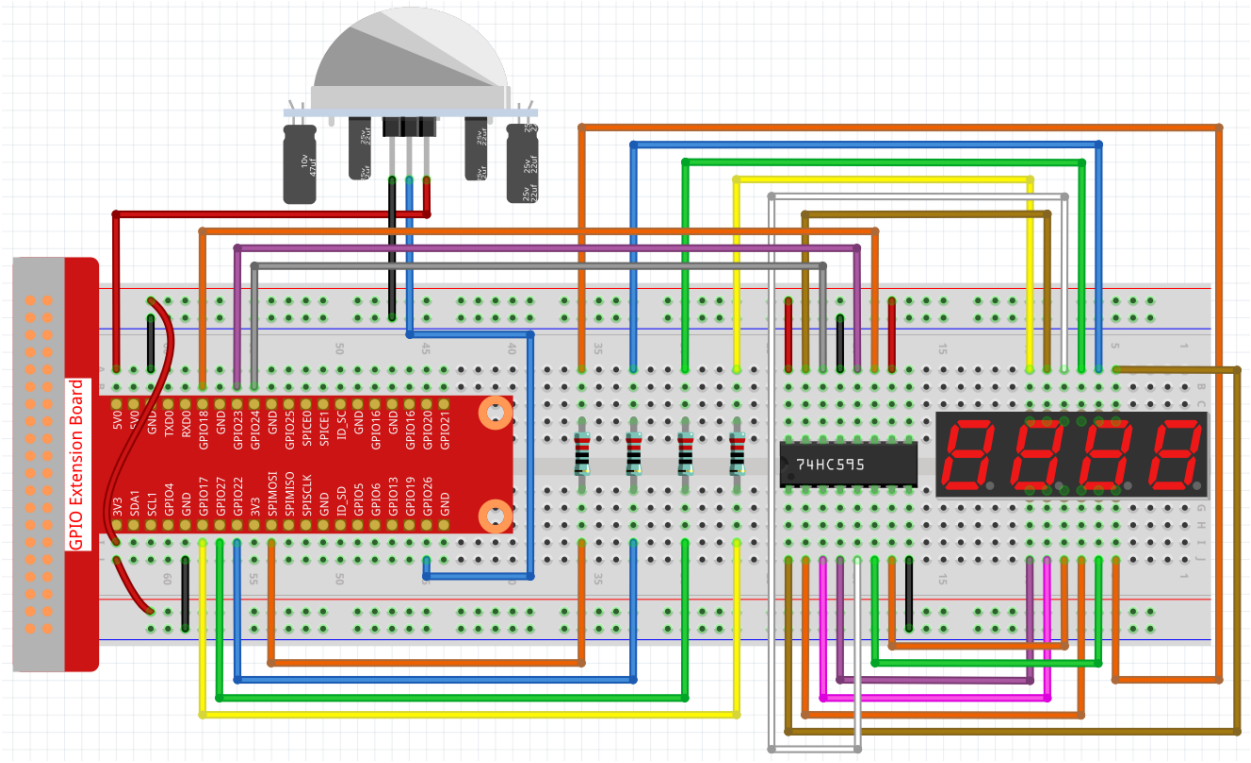
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.1/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 3.1.1_CountingDevice.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Wenn der PIR nach dem Ausführen der Kode feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1.

There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.

Code Erklärung

```
void display()
{
    clearDisplay();
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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]);
}
```

Starten Sie zuerst die vierte Segmentanzeige und schreiben Sie die einstellige Nummer. Dann starten Sie die Anzeige des dritten Segments und geben Sie die Zehnerstelle ein. Starten Sie danach die zweite bzw. die erste Segmentanzeige und schreiben Sie die Hunderter- bzw. Tausenderstellen. Da die Aktualisierungsgeschwindigkeit sehr hoch ist, sehen wir eine vollständige vierstellige Anzeige.

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            counter +=1;
        }
        lastState=currentState;
    }
}
```

Dies ist die Hauptfunktion: Zeigen Sie die Nummer auf der 4-stelligen Segmentanzeige an und lesen Sie den PIR-Wert. Wenn der PIR feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.1_CountingDevice.py
```

Wenn der PIR nach dem Ausführen der Kode feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1.

code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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

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])
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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()
```

Code Erklärung

Basierend auf der *1.1.5 4-stellige 7-Segment-Anzeige* wird in dieser Lektion ein PIR-Modul hinzugefügt, um die automatische Zählung von Lektion 1.1.5 in Zählerkennung zu ändern. Wenn der PIR feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1.

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

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

    clearDisplay()
    pickDigit(2)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

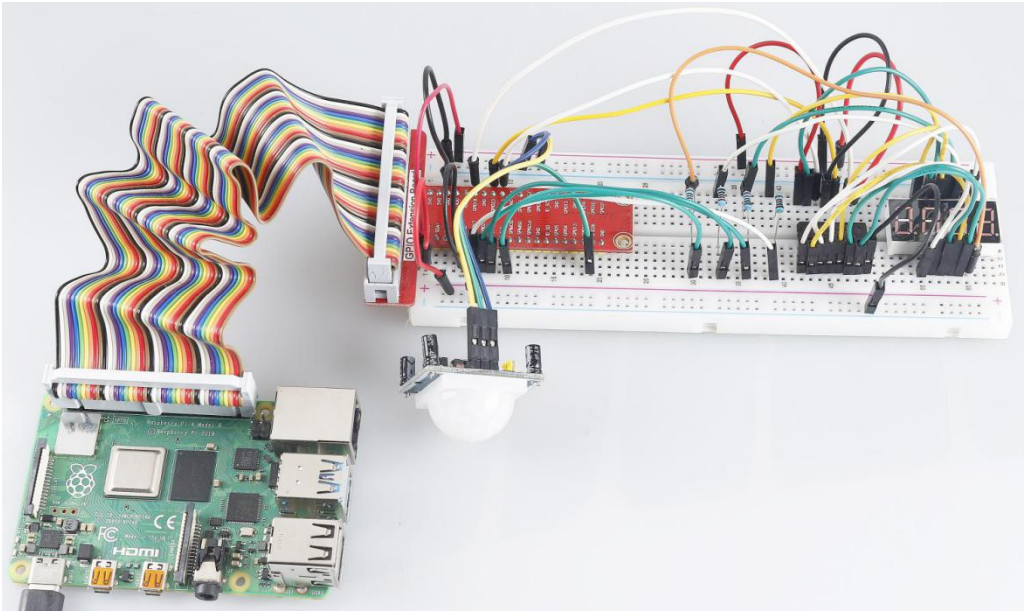
```
    hc595_shift(number[counter % 1000//100])

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

Starten Sie zuerst die vierte Segmentanzeige und schreiben Sie die einstellige Nummer. Dann starten Sie die Anzeige des dritten Segments und geben Sie die Zehnerstelle ein. Starten Sie danach die zweite bzw. die erste Segmentanzeige und schreiben Sie die Hunderter- bzw. Tausenderstellen. Da die Aktualisierungsgeschwindigkeit sehr hoch ist, sehen wir eine vollständige vierstellige Anzeige.

```
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
```

Dies ist die Hauptfunktion: Zeigen Sie die Nummer auf der 4-stelligen Segmentanzeige an und lesen Sie den PIR-Wert. Wenn der PIR feststellt, dass jemand vorbeikommt, addiert die Nummer auf der 4-stelligen Segmentanzeige 1.

Phänomen Bild

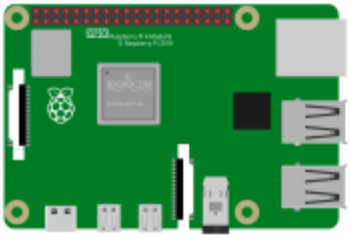
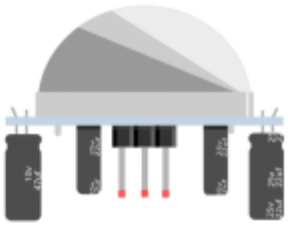
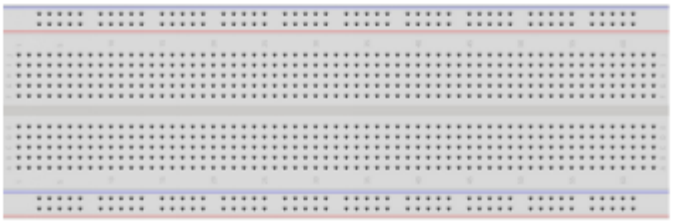


1.9.2 3.1.2 Willkommen

Einführung

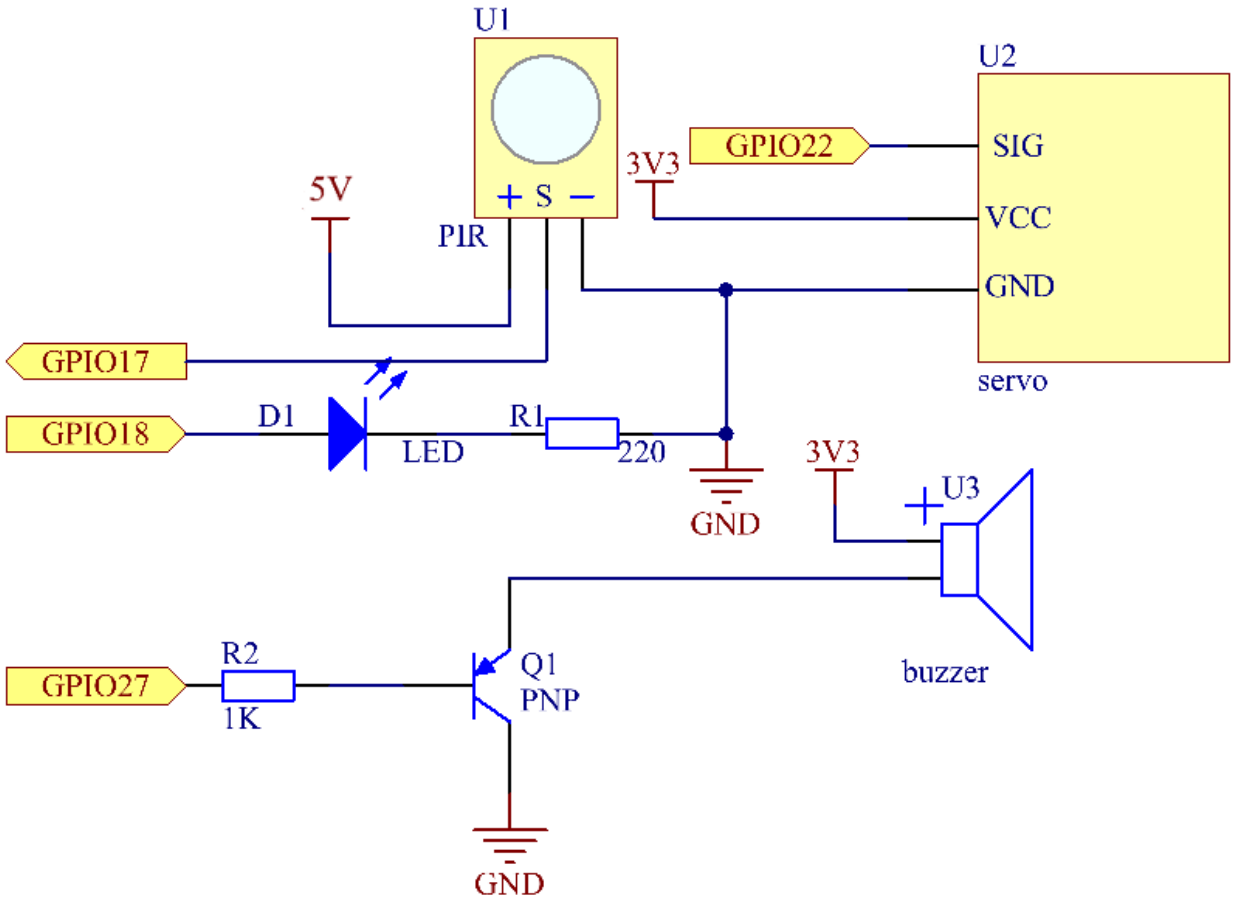
In diesem Projekt werden wir PIR verwenden, um die Bewegung von Fußgängern zu erfassen, und Servos, LED und Summer verwenden, um die Arbeit der Sensortür des Supermarkts zu simulieren. Wenn der Fußgänger innerhalb des Erfassungsbereichs des PIR erscheint, leuchtet die Anzeigelampe, die Tür wird geöffnet und der Summer ertönt die Öffnungsglocke.

Komponenten

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td colspan="2">1 * PIR
  </td></tr>
<tr><td>1 * Servo
 </td><td>1 * Passive Buzzer
 </td><td>1 * S8550 PNP Transistor
 </td><td>1 * LED
 </td></tr>
<tr><td>1 * Resistor 1kΩ
 </td><td>1 * Resistor 220Ω
 </td><td colspan="2">Several Jumper Wires
 </td></tr>
<tr><td colspan="4">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="4">1 * Breadboard
  </td></tr>
</table>

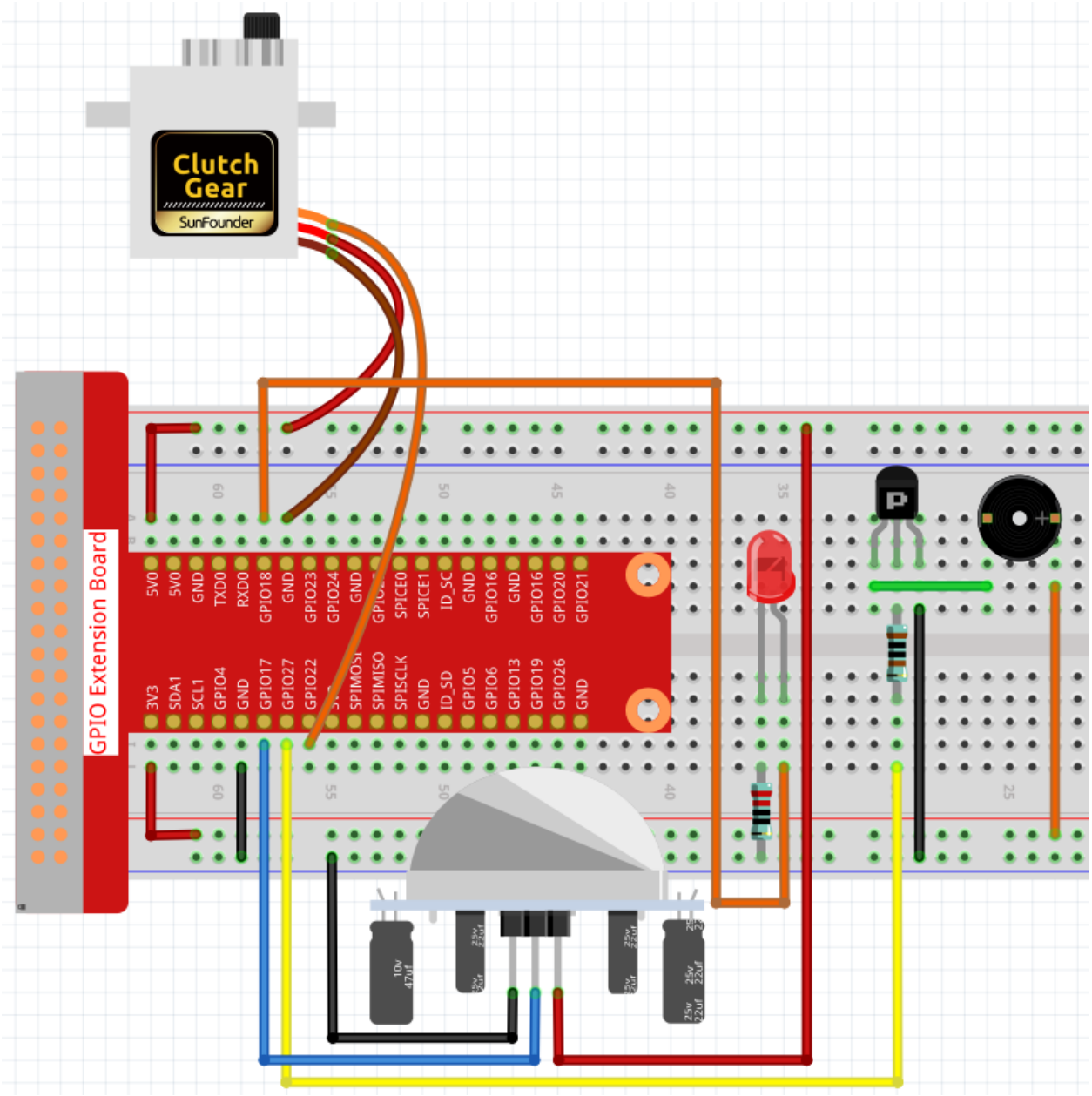
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.2/
```

Schritt 3: Kompilieren.

```
gcc 3.1.2_Welcome.c -lwiringPi
```

Schritt 4: Ausführen.

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

Wenn der PIR-Sensor nach dem Ausführen der Kode jemanden erkennt, der vorbeikommt, öffnet sich die Tür automatisch (vom Servo simuliert), schaltet die Anzeige ein und spielt die Türklingelmusik ab. Nachdem die Türklingelmusik abgespielt wurde, schließt das System automatisch die Tür und schaltet die Anzeigelampe aus, um auf das nächste Mal zu warten, wenn jemand vorbeikommt.

There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.

Code Erklärung

```
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));
}
```

Erstellen Sie eine Funktion, setAngle, um den Winkel im Servo von 0-180 zu schreiben.

```
void doorbell(){
for(int i=0;i<sizeof(song)/4;i++){
        softToneWrite(BuzPin, song[i]);
        delay(beat[i] * 250);
    }
```

Erstellen Sie eine Funktion, Türklingel, damit der Summer Musik abspielen kann.

```
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);
    }
}
```

Erstellen Sie eine Closedoor-Funktion, um das Schließen der Tür zu simulieren, schalten Sie die LED aus und lassen Sie das Servo von 180 Grad auf 0 Grad drehen.

```
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();
}
```

Die Funktion `opendoor()` besteht aus mehreren Teilen: Schalten Sie die Anzeigelampe ein, schalten Sie das Servo ein (simulieren Sie das Öffnen der Tür), spielen Sie die Klingelmusik des Supermarkts und rufen Sie nach dem Abspielen der Musik die Funktion geschlossene `closedoor()` auf.

```
    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;
......
```

Initialisieren Sie in der Funktion `main()` die Bibliothek wiringPi und richten Sie softTone ein. Setzen Sie dann ledPin in den Ausgabestatus und pirPin in den Eingabestatus. Wenn der PIR-Sensor jemanden erkennt, der vorbeikommt, wird die Funktion opendoor aufgerufen, um das Öffnen der Tür zu simulieren.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.2_Welcome.py
```

Wenn der PIR-Sensor nach dem Ausführen der Kode jemanden erkennt, der vorbeikommt, öffnet sich die Tür automatisch (vom Servo simuliert), schaltet die Anzeige ein und spielt die Türklingelmusik ab. Nachdem die Türklingelmusik abgespielt wurde, schließt das System automatisch die Tür und schaltet die Anzeigelampe aus, um auf das nächste Mal zu warten, wenn jemand vorbeikommt.

There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.

code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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
    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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Erklärung

```
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
```

Diese Anweisungen werden verwendet, um die Pins jeder Komponente zu initialisieren.

```
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
```

Erstellen Sie eine Funktion von servowrite, um den Winkel in das Servo zu schreiben, der 0-180 ist.

```
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
```

Erstellen Sie eine Funktion, Türklingel, damit der Summer Musik abspielen kann.

```
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)
```

Schließen Sie die Tür und schalten Sie die Kontrollleuchte aus.

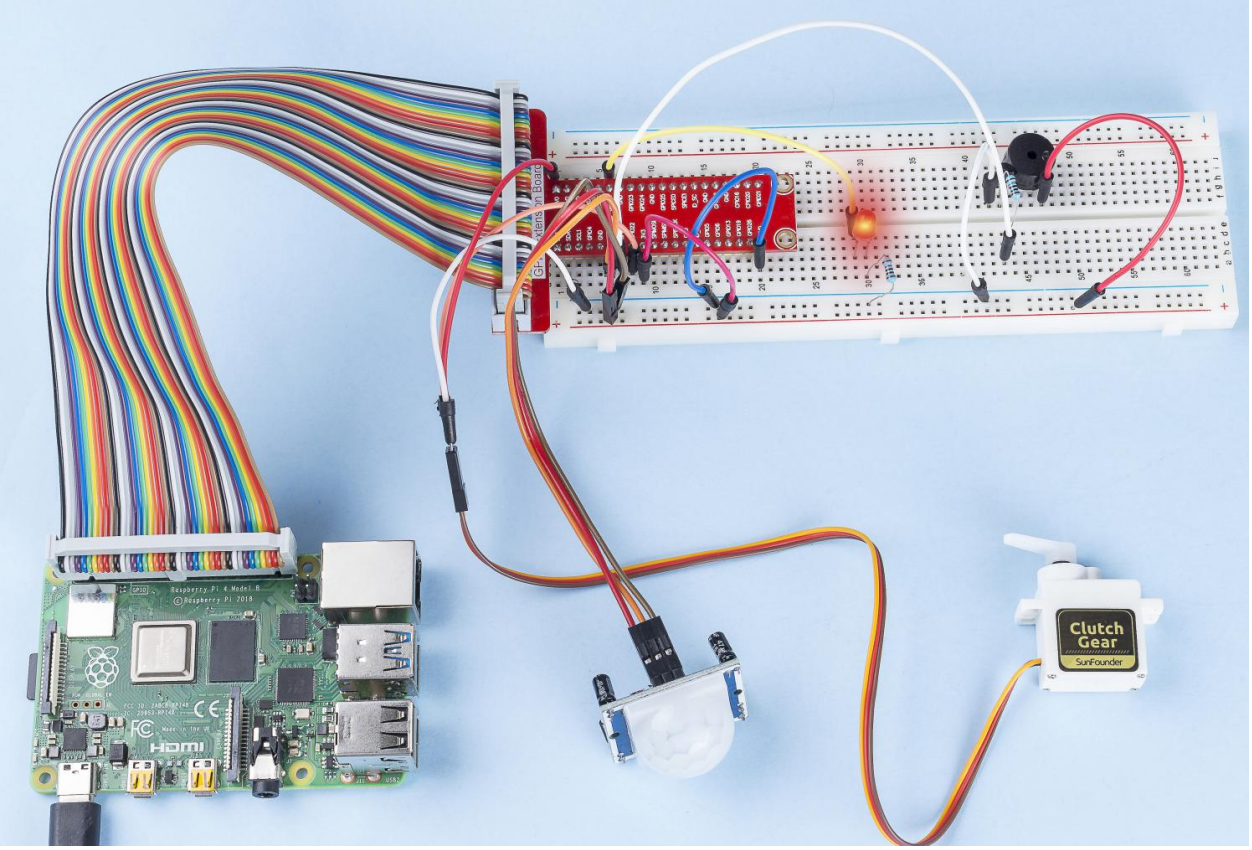
```
def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1):   #make servo rotate from 0 to 180 deg
        setAngle(i)     # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()
```

Die Funktion `opendoor()` besteht aus mehreren Teilen: Schalten Sie die Anzeigelampe ein, schalten Sie das Servo ein (um das Öffnen der Tür zu simulieren), spielen Sie die Klingelmusik des Supermarkts und rufen Sie nach dem Spielen die Funktion `closedoor()` auf Musik.

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

Wenn RIP erkennt, dass jemand vorbeikommt, ruft es die Funktion `opendoor()` auf.

Phänomen Bild

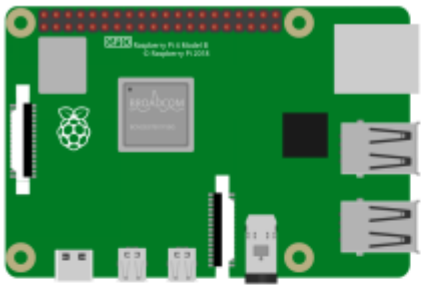
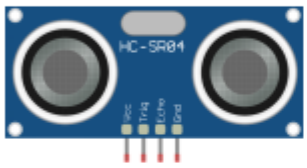
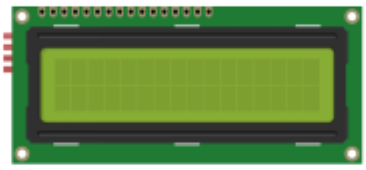
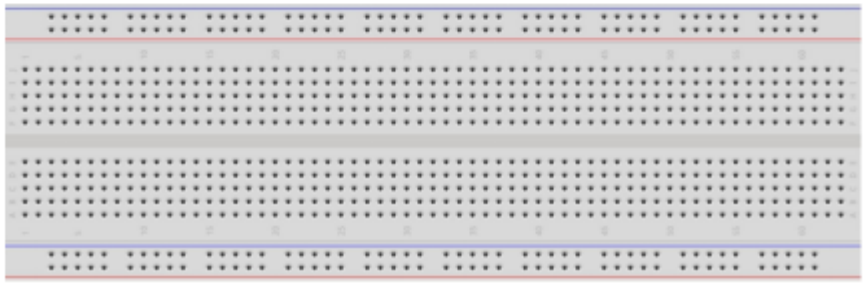


1.9.3 3.1.3 Alarm umkehren

Einführung

In diesem Projekt werden wir LCD-, Summer- und Ultraschallsensoren verwenden, um ein Rückwärtshilfesystem herzustellen. Wir können es auf das ferngesteuerte Fahrzeug setzen, um den tatsächlichen Vorgang des Rückwärtsfahrens des Autos in die Garage zu simulieren.

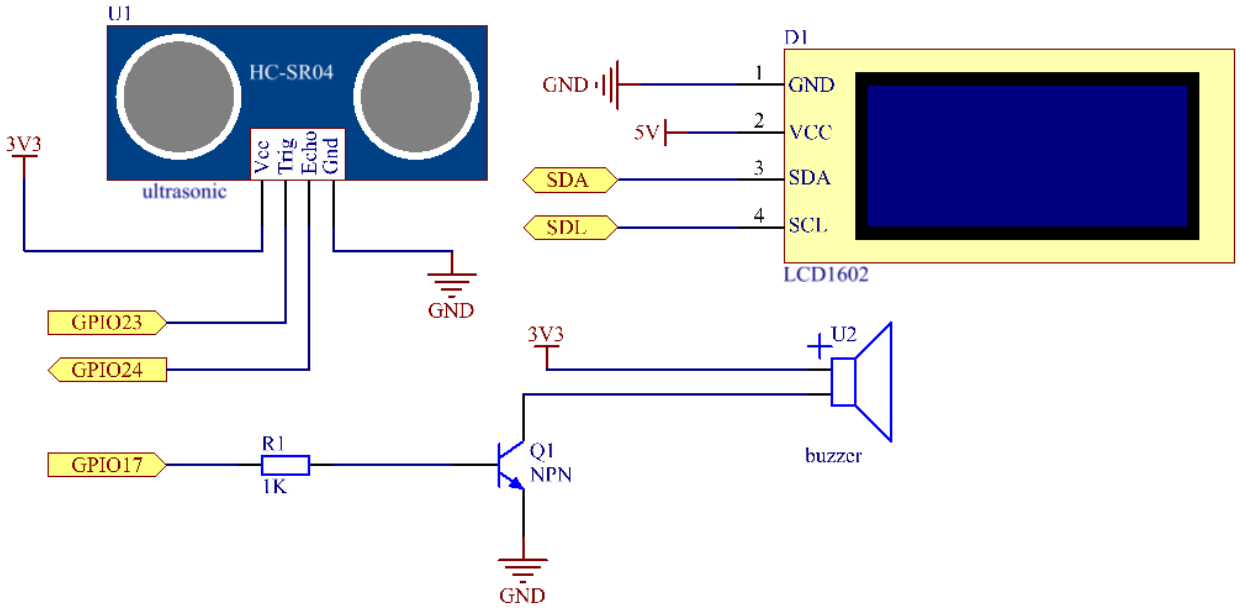
Komponenten

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 		

Schematische Darstellung

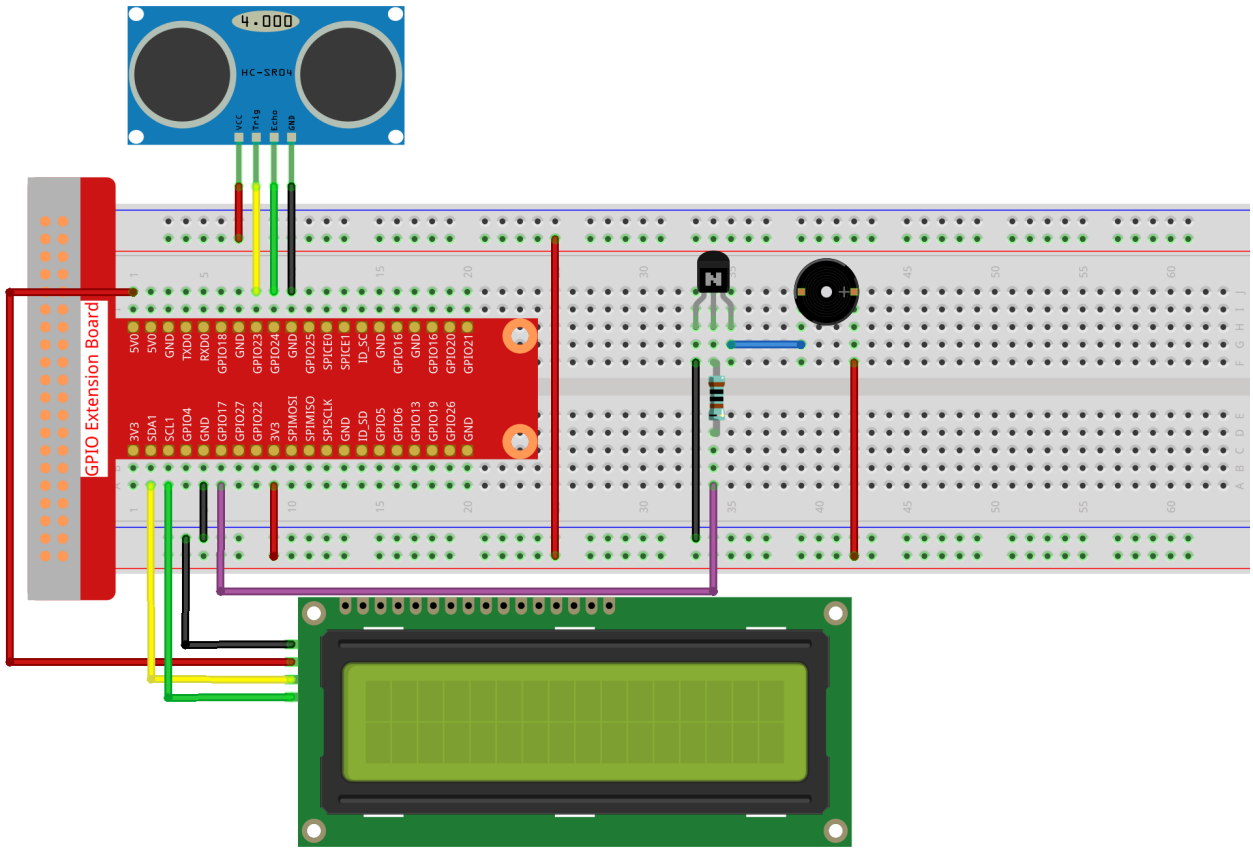
Der Ultraschallsensor erkennt den Abstand zwischen sich und dem Hindernis, der in Form einer Kode auf dem LCD angezeigt wird. Gleichzeitig ließ der Ultraschallsensor den Summer einen sofortigen Ton unterschiedlicher Frequenz je nach Entfernungswert ausgeben.

T-Karte Name	physisch	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.3/
```

Schritt 3: Kompilieren.

```
gcc 3.1.3_ReversingAlarm.c -lwiringPi
```

Schritt 4: Ausführen.

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

Während die Kode ausgeführt wird, erkennt das Ultraschallsensormodul die Entfernung zum Hindernis und zeigt dann die Informationen zur Entfernung auf dem LCD1602 an. Außerdem gibt der Summer einen Warnton aus, dessen Frequenz sich mit der Entfernung ändert.

Code

Bemerkung: Die folgenden der Kode sind unvollständig. Wenn Sie die vollständigen Kode überprüfen möchten, wird empfohlen, den Befehl `nano 3.1.1_ReversingAlarm.c` zu verwenden.

```
#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)
{
    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){
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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++){
                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;
}
```

Code Erklärung

```
pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();
```

In diesem Programm wenden wir frühere Komponenten synthetisch an. Hier verwenden wir Summer, LCD und Ultraschall. Wir können sie auf die gleiche Weise wie zuvor initialisieren.

```
dis = disMeasure();
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 printf("%.2f cm \n",dis);
digitalWrite(Buzzer,LOW);
if (dis > 400){
     write(0, 0, "Error");
     write(3, 1, "Out of range");
}
else
{
    write(0, 0, "Distance is");
    sprintf(result,"%.2f cm",dis);
    write(5, 1, result);
    }
```

Hier erhalten wir den Wert des Ultraschallsensors und die Entfernung durch Berechnung.

Wenn der Entfernungswert größer als der zu erkennende Bereichswert ist, wird eine Fehlermeldung auf dem LCD gedruckt. Wenn der Abstandswert innerhalb des Bereichs liegt, werden die entsprechenden Ergebnisse ausgegeben.

```
sprintf(result,"%.2f cm",dis);
```

Da der Ausgabemodus des LCD nur den Zeichentyp unterstützt und die Variable den Wert des Float-Typs nicht speichert, müssen wir `sprintf()` verwenden. Die Funktion konvertiert den Float-Typ-Wert in ein Zeichen und speichert ihn in der String-Variablen `result[]` . `%.2f` bedeutet, zwei Dezimalstellen beizubehalten.

```
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);
    }
}
```

Diese Beurteilungsbedingung wird verwendet, um das Geräusch des Summers zu steuern. Je nach Entfernungsunterschied kann es in drei Fälle unterteilt werden, in denen unterschiedliche Schallfrequenzen auftreten. Da der Gesamtwert der Verzögerung 500 beträgt, können alle Fälle ein Intervall von 500 ms für den Ultraschallsensor bereitstellen.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.3_ReversingAlarm.py
```

Während die Kode ausgeführt wird, erkennt das Ultraschallsensormodul die Entfernung zum Hindernis und zeigt dann die Informationen zur Entfernung auf dem LCD1602 an. Außerdem gibt der Summer einen Warnton aus, dessen Frequenz sich mit der Entfernung ändert.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

TRIG = 16
ECHO = 18
BUZZER = 11

def lcdsetup():
LCD1602.init(0x27, 1)   # init(slave address, background light)
LCD1602.clear()
LCD1602.write(0, 0, 'Ultrasonic Starting')
LCD1602.write(1, 1, 'By SunFounder')
time.sleep(2)

def setup():
GPIO.setmode(GPIO.BOARD)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
lcdsetup()

def distance():
GPIO.output(TRIG, 0)
time.sleep(0.000002)

GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def destroy():
GPIO.output(BUZZER, GPIO.LOW)
GPIO.cleanup()
LCD1602.clear()

def loop():
while True:
    dis = distance()
    print (dis, 'cm')
    print ('')
    GPIO.output(BUZZER, GPIO.LOW)
    if (dis > 400):
        LCD1602.clear()
        LCD1602.write(0, 0, 'Error')
        LCD1602.write(3, 1, 'Out of range')
        time.sleep(0.5)
    else:
        LCD1602.clear()
        LCD1602.write(0, 0, 'Distance is')
        LCD1602.write(5, 1, str(round(dis,2)) +' cm')
        if(dis>=50):
            time.sleep(0.5)
        elif(dis<50 and dis>20):
            for i in range(0,2,1):
                GPIO.output(BUZZER, GPIO.HIGH)
                time.sleep(0.05)
                GPIO.output(BUZZER, GPIO.LOW)
                time.sleep(0.2)
        elif(dis<=20):
            for i in range(0,5,1):
                GPIO.output(BUZZER, GPIO.HIGH)
                time.sleep(0.05)
                GPIO.output(BUZZER, GPIO.LOW)
                time.sleep(0.05)


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

Code Erklärung

```
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()
```

In diesem Programm wenden wir die zuvor verwendeten Komponenten synthetisch an. Hier verwenden wir Summer, LCD und Ultraschall. Wir können sie auf die gleiche Weise wie zuvor initialisieren.

```
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')
```

Hier erhalten wir die Werte des Ultraschallsensors und die Entfernung durch Berechnung. Wenn der Entfernungswert größer als der zu erkennende Wertebereich ist, wird eine Fehlermeldung auf dem LCD gedruckt. Und wenn der Abstand innerhalb des Arbeitsbereichs liegt, werden die entsprechenden Ergebnisse ausgegeben.

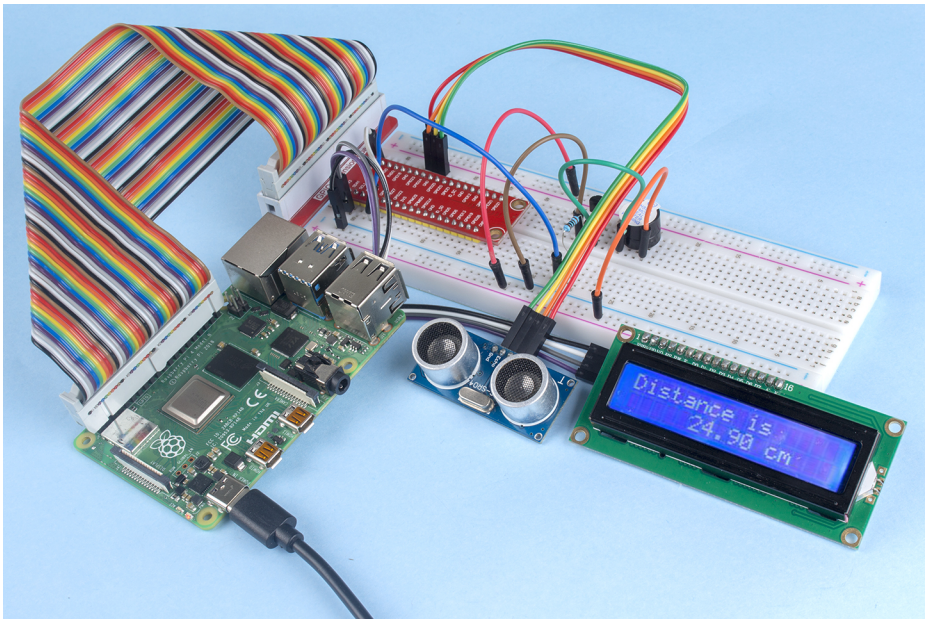
```
LCD1602.write(5, 1, str(round(dis,2)) +' cm')
```

Da der LCD-Ausgang nur Zeichentypen unterstützt, müssen wir `str()` verwenden, um numerische Werte in Zeichen umzuwandeln. Wir werden es auf zwei Dezimalstellen runden.

```
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);
        }
    }
```

Diese Beurteilungsbedingung wird verwendet, um das Geräusch des Summers zu steuern. Je nach Entfernungsunterschied kann es in drei Fälle unterteilt werden, in denen unterschiedliche Schallfrequenzen auftreten. Da der Gesamtwert der Verzögerung 500 beträgt, können alle ein Intervall von 500 ms bereitstellen, damit der Ultraschallsensor funktioniert.

Phänomen Bild

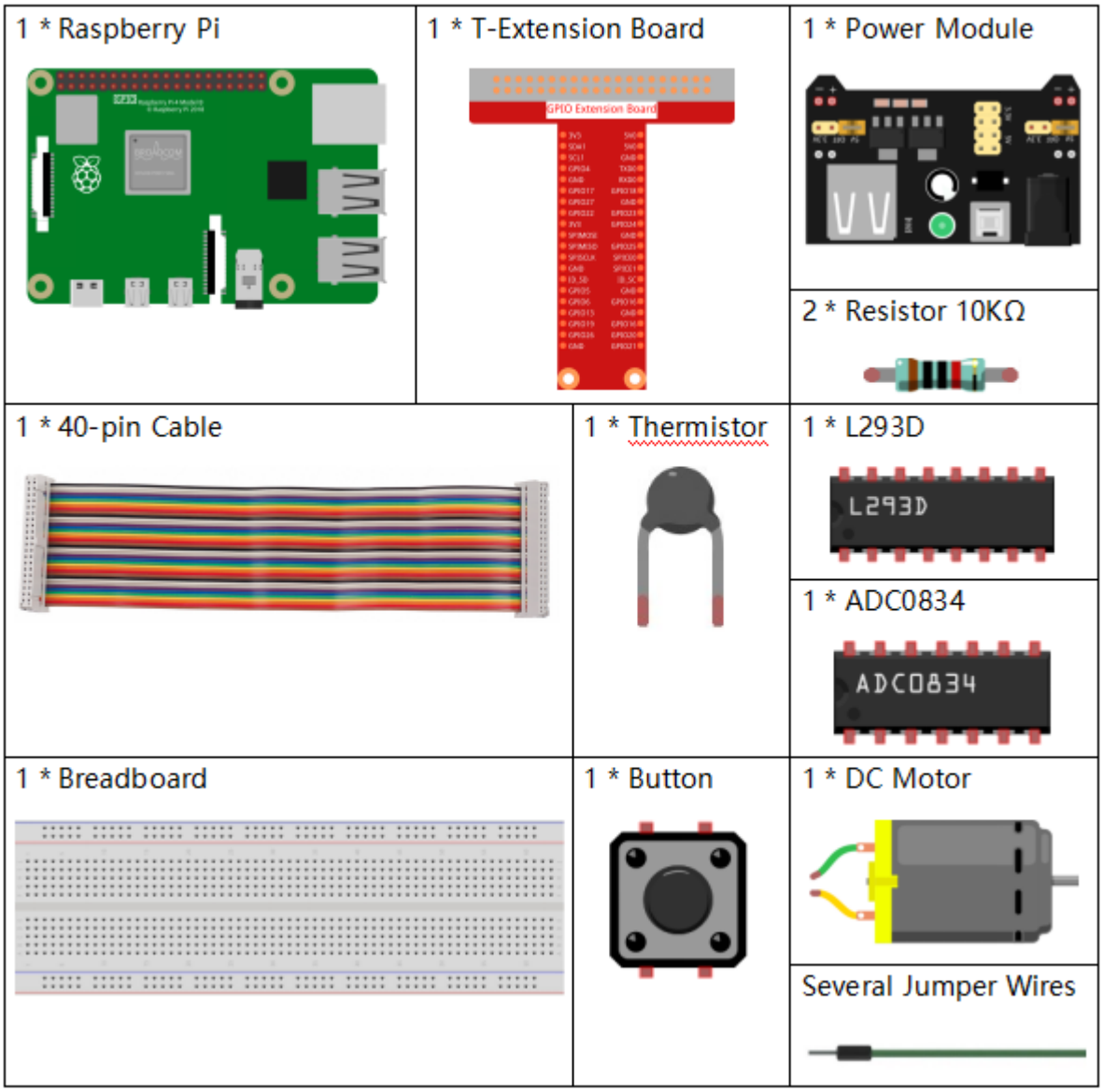


1.9.4 3.1.4 Smart Fan

Einführung

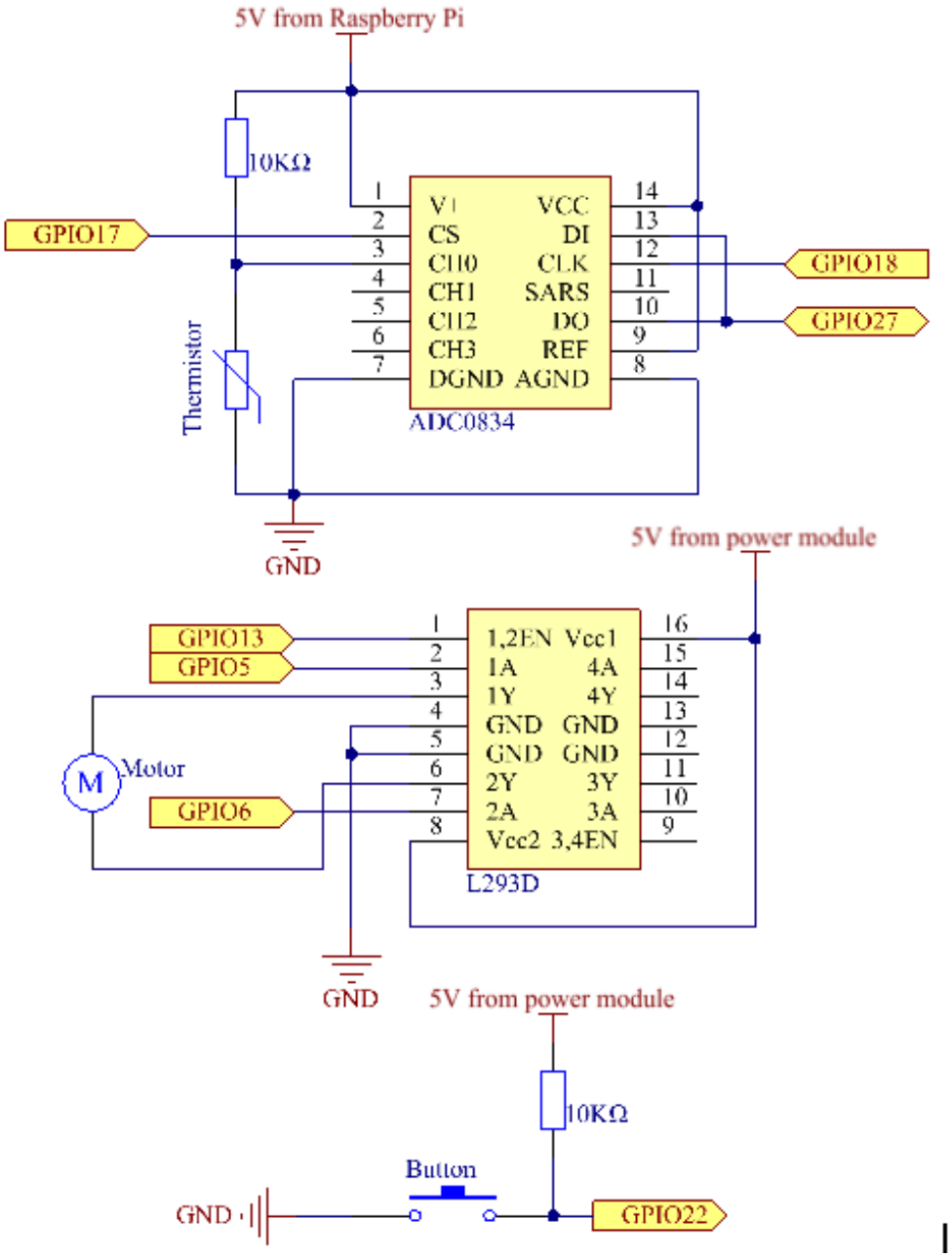
In diesem Kurs werden wir Motoren, Tasten und Thermistoren verwenden, um einen manuellen + automatischen intelligenten Lüfter herzustellen, dessen Windgeschwindigkeit einstellbar ist.

Komponenten



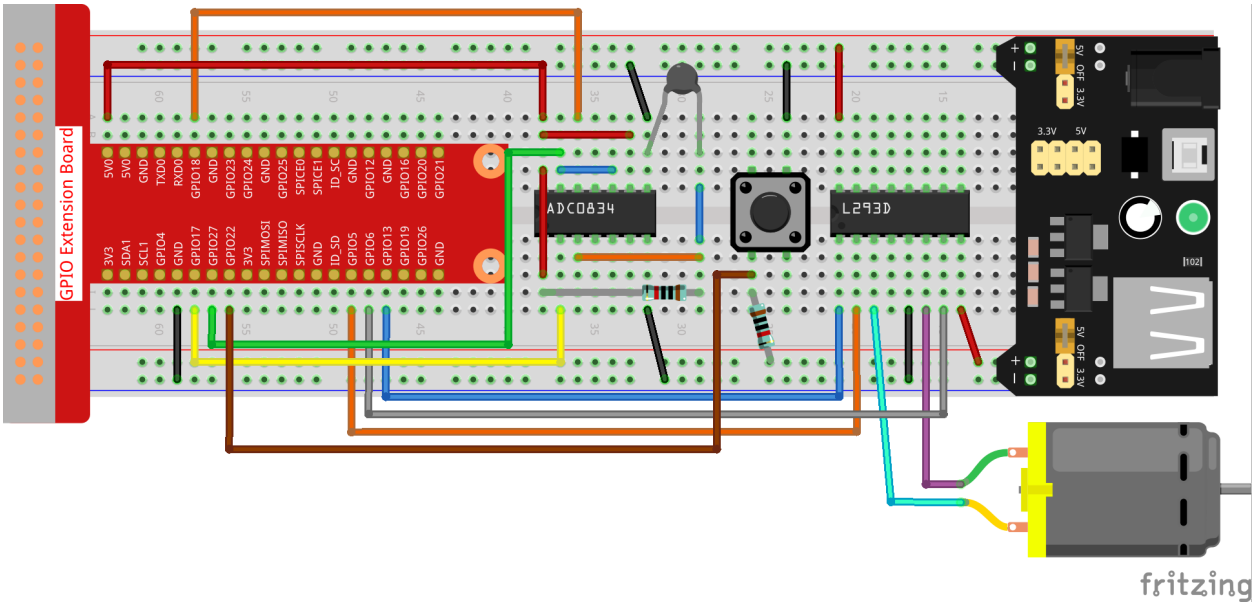
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Bemerkung: The power module can apply a 9V battery with the 9V Battery Buckle in the kit. Insert the jumper cap of the power module into the 5V bus strips of the breadboard.

Für Benutzer in C-Sprache

Schritt 2: Gehen Sie in den Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.4/
```

Schritt 3: Kompilieren.

```
gcc 3.1.4_SmartFan.c -lwiringPi -lm
```

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Starten Sie den Lüfter, während der Kode ausgeführt wird, indem Sie die Taste drücken. Jedes Mal, wenn Sie drücken, wird 1 Geschwindigkeitsstufe nach oben oder unten eingestellt. Es gibt 5 Arten von Geschwindigkeitsstufen: 0~4. Wenn Sie die vierte Geschwindigkeitsstufe einstellen und die Taste drücken, arbeitet der Lüfter nicht mehr mit einer Windgeschwindigkeit von 0.

Sobald die Temperatur länger als 2°C, steigt oder fällt, wird die Geschwindigkeit automatisch um 1 Grad schneller oder langsamer.

Code Erklärung

```
int temperture(){
    unsigned char analogVal;
    double Vr, Rt, temp, cel, Fah;
    analogVal = get_ADC_Result(0);
    Vr = 5 * (double)(analogVal) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    int t=cel;
    return t;
}
```

`temperture()` wandelt die von ADC0834 gelesenen Thermistorwerte in Temperaturwerte um. Siehe *2.2.2 Thermistor* für weitere Details.

```
int motor(int level){
    if(level==0){
        digitalWrite(MotorEnable,LOW);
        return 0;
    }
    if (level>=4){
        level =4;
    }
    digitalWrite(MotorEnable,HIGH);
    softPwmWrite(MotorPin1, level*25);
    return level;
}
```

Diese Funktion steuert die Drehzahl des Motors. Der Bereich der Stufe: 0-4 (Stufe 0 stoppt den Arbeitsmotor). Eine Stufeneinstellung steht für eine 25% ige Änderung der Windgeschwindigkeit.

```
int main(void)
{
    setup();
    int currentState,lastState=0;
    int level = 0;
    int currentTemp,markTemp=0;
    while(1){
        currentState=digitalRead(BtnPin);
        currentTemp=temperture();
        if (currentTemp<=0){continue;}
        if (currentState==1&&lastState==0){
            level=(level+1)%5;
            markTemp=currentTemp;
            delay(500);
        }
        lastState=currentState;
        if (level!=0){
            if (currentTemp-markTemp<=-2){
                level=level-1;
                markTemp=currentTemp;
            }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            if (currentTemp-markTemp>=2){
                level=level+1;
                markTemp=currentTemp;
            }
        }
        level=motor(level);
    }
    return 0;
}
```

Die Funktion `main()` enthält den gesamten Programmablauf wie folgt:

1. Lesen Sie ständig den Tastenstatus und die aktuelle Temperatur ab.
2. Jede Presse erreicht Stufe+1 und gleichzeitig wird die Temperatur aktualisiert. Die Stufe reicht von 1~4..
3. Während der Lüfter arbeitet (der Niveau ist nicht 0), wird die Temperatur erfasst. Eine Änderung von 2℃+ bewirkt das Auf und Ab des Levels.
4. Der Motor ändert die Drehzahl mit der Stufe.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie in den Ordner der Kode.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.4_SmartFan.py
```

Starten Sie den Lüfter, während der Kode ausgeführt wird, indem Sie die Taste drücken. Jedes Mal, wenn Sie drücken, wird 1 Geschwindigkeitsstufe nach oben oder unten eingestellt. Es gibt 5 Arten von Geschwindigkeitsstufen: 0~4. Wenn Sie die vierte Geschwindigkeitsstufe einstellen und die Taste drücken, arbeitet der Lüfter nicht mehr mit einer Windgeschwindigkeit von 0.

Sobald die Temperatur länger als 2℃, steigt oder fällt, wird die Geschwindigkeit automatisch um 1 Grad schneller oder langsamer.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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


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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            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()
```

Code Erklärung

```
def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return Cel
```

`temperature()` wandelt von ADC0834 gelesene Thermistorwerte in Temperaturwerte um. Siehe *2.2.2 Thermistor* für weitere Details.

```
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
```

Diese Funktion steuert die Drehzahl des Motors. Der Bereich des Hebels: 0-4 (Stufe 0 stoppt den Arbeitsmotor). Eine Stufeneinstellung steht für eine 25% ige Änderung der Windgeschwindigkeit.

```
def main():
    lastState=0
    level=0
    markTemp = temperature()
    while True:
        currentState =GPIO.input(BtnPin)
        currentTemp=temperature()
```

(Fortsetzung auf der nächsten Seite)

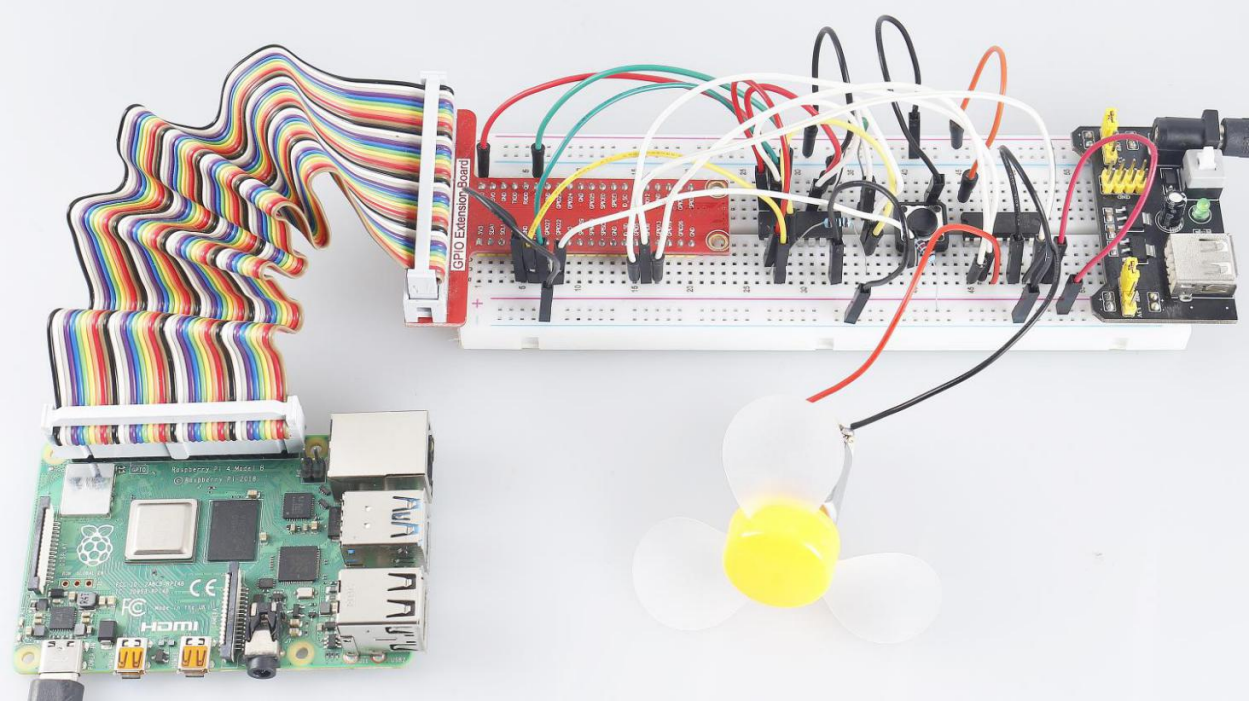
(Fortsetzung der vorherigen Seite)

```
        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)
```

Die Funktion `main()` enthält den gesamten Programmablauf wie folgt:

1. Lesen Sie ständig den Tastenstatus und die aktuelle Temperatur ab.
2. Jede Presse erreicht Stufe+1 und gleichzeitig wird die Temperatur aktualisiert. Die Stufe reicht von 1~4..
3. Während der Lüfter arbeitet (der Niveau ist nicht 0), wird die Temperatur erfasst. Eine Änderung von 2℃+ bewirkt das Auf und Ab des Levels.
4. Der Motor ändert die Drehzahl mit der Stufe.

Phänomen Bild

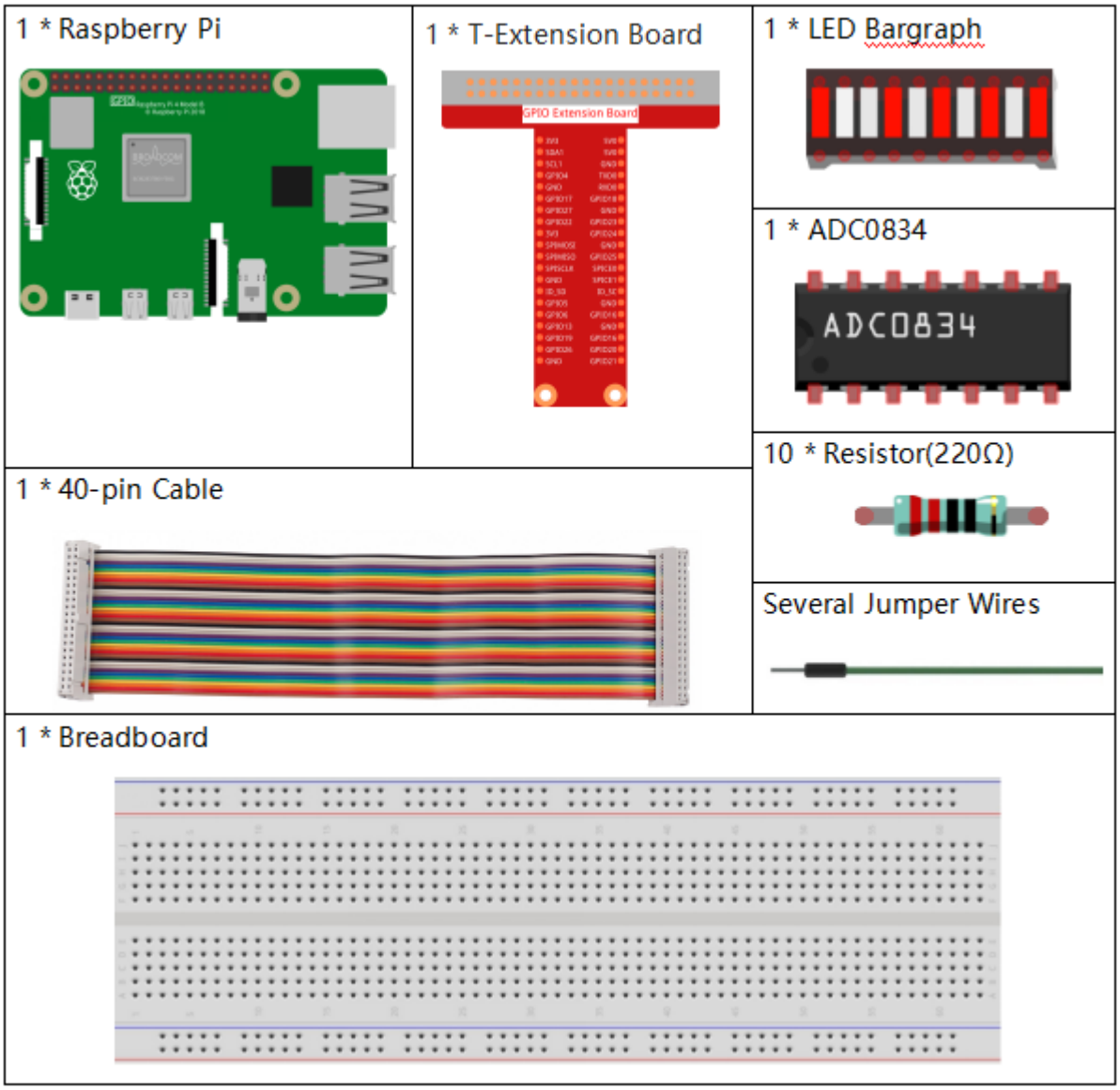


1.9.5 3.1.5 Batterieanzeige

Einführung

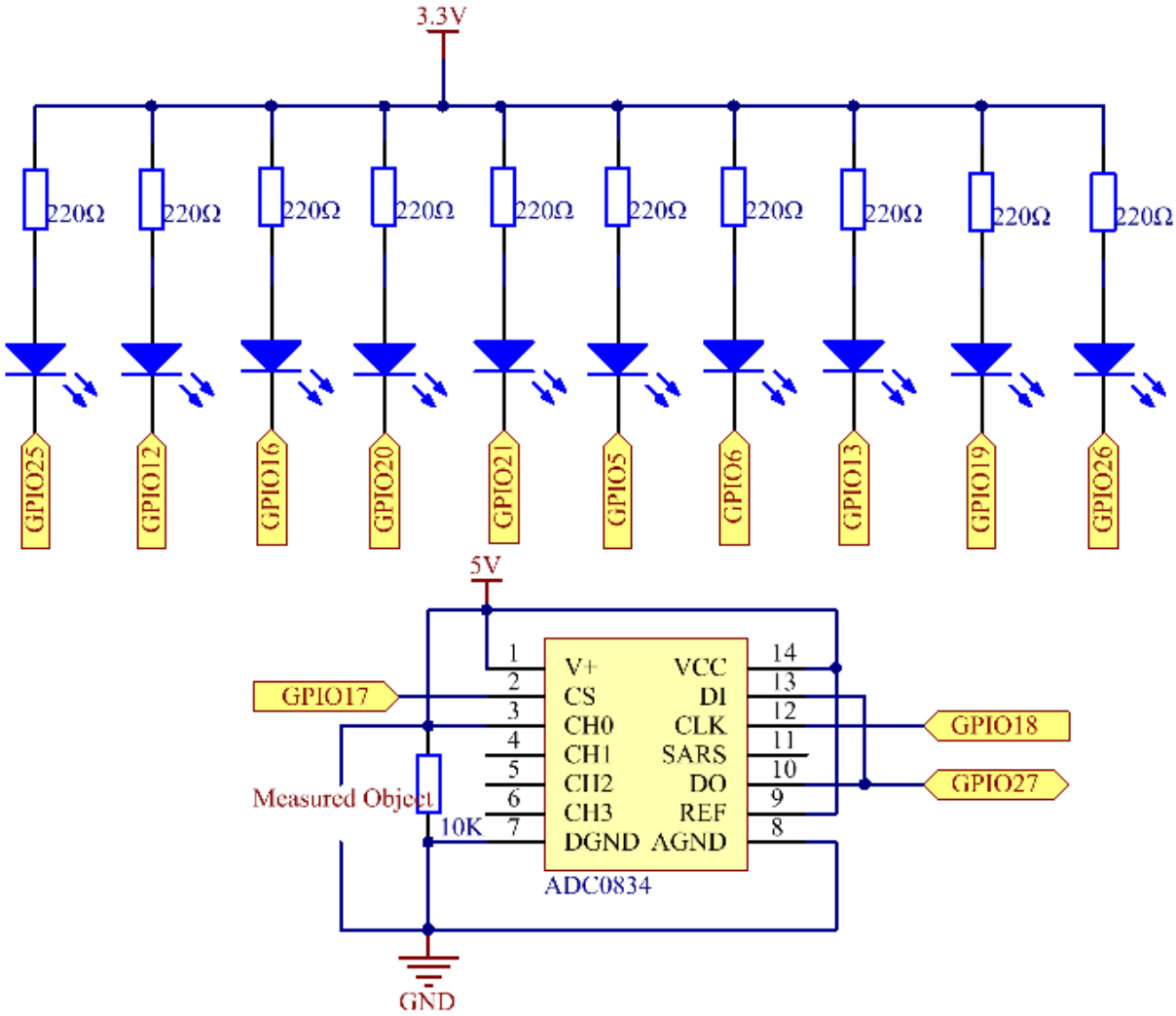
In diesem Kurs erstellen wir ein Batterieanzeigegerät, mit dem der Batteriestand auf dem LED-Balkendiagramm visuell angezeigt werden kann.

Komponenten



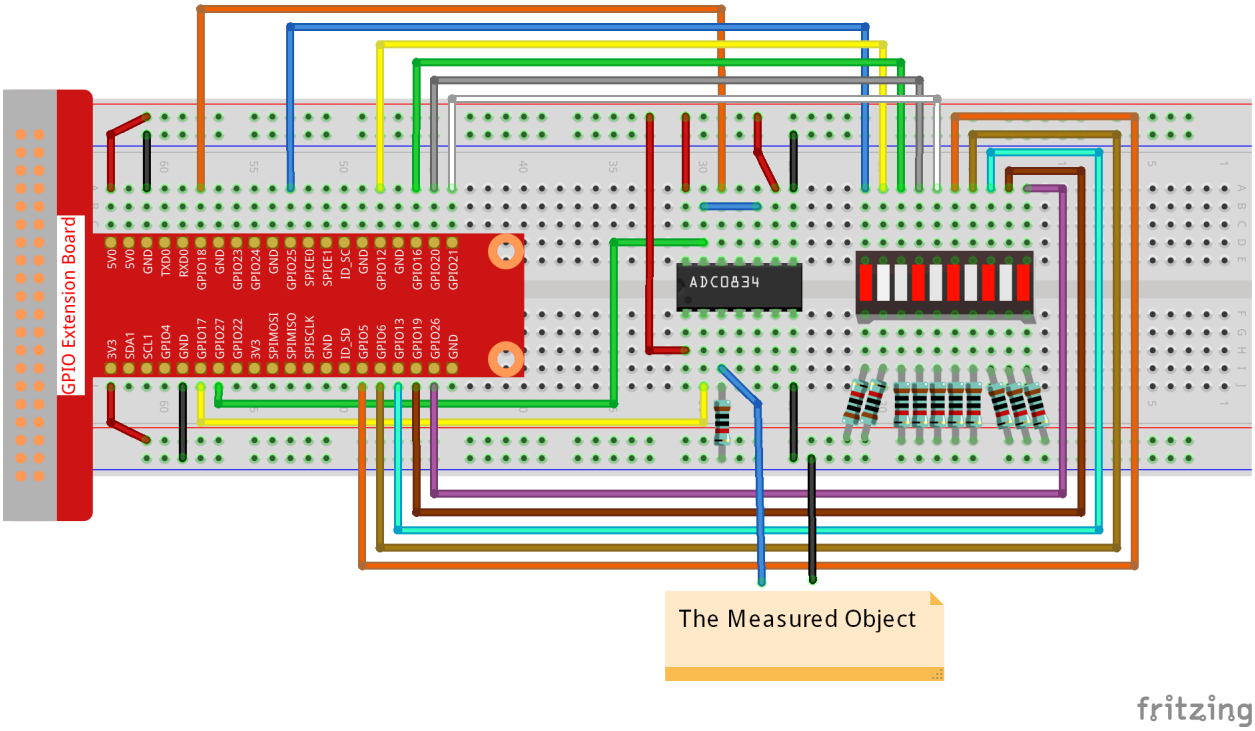
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.5/
```

Schritt 3: Kompilieren Sie der Kode.

```
gcc 3.1.5_BatteryIndicator.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Nachdem das Programm ausgeführt wurde, geben Sie dem 3. Pin von ADC0834 und dem GND separat einen Anschlussdraht und führen Sie sie dann separat zu den beiden Polen einer Batterie. Sie können sehen, dass die entsprechende LED auf dem LED-Balkendiagramm leuchtet, um den LeistungsNiveau anzuzeigen (Messbereich: 0-5V).

Code Erklärung

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
}
```

Diese Funktion dienst für die Steuerung von Ein- und Ausschalten der 10 LEDs am LED-Bargraphen. Wir geben diesen 10 LEDs hohe Niveau, damit sie zuerst ausgeschaltet sind, und entscheiden dann, wie viele LEDs aufleuchten, indem wir den empfangenen Analogwert ändern.

```
int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);
    for(int i=0;i<10;i++){       //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],HIGH);
    }
    while(1){
        analogVal = get_ADC_Result(0);
        LedBarGraph(analogVal/25);
        delay(100);
    }
    return 0;
}
```

`analogVal` erzeugt Werte (0-255) mit variierenden Spannungswerten (0-5V). Wenn beispielsweise 3V an einer Batterie erkannt werden, wird der entsprechende Wert 152 auf dem Voltmeter angezeigt.

Die 10 LEDs auf dem LED-Balkendiagramm dienen zur Anzeige der `analogVal` Messwerte. 255/10 = 25, also alle 25 erhöht sich der Analogwert, eine weitere LED leuchtet auf, z. B. wenn “analogVal = 150 (ca. 3 V), leuchten 6 LEDs”.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.5_BatteryIndicator.py
```

Nachdem das Programm ausgeführt wurde, geben Sie dem 3. Pin von ADC0834 und dem GND separat einen Anschlussdraht und führen Sie sie dann separat zu den beiden Polen einer Batterie. Sie können sehen, dass die entsprechende LED auf dem LED-Balkendiagramm leuchtet, um den LeistungsNiveau anzuzeigen (Messbereich: 0-5V).

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)
    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()
```

Code Erklärung

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

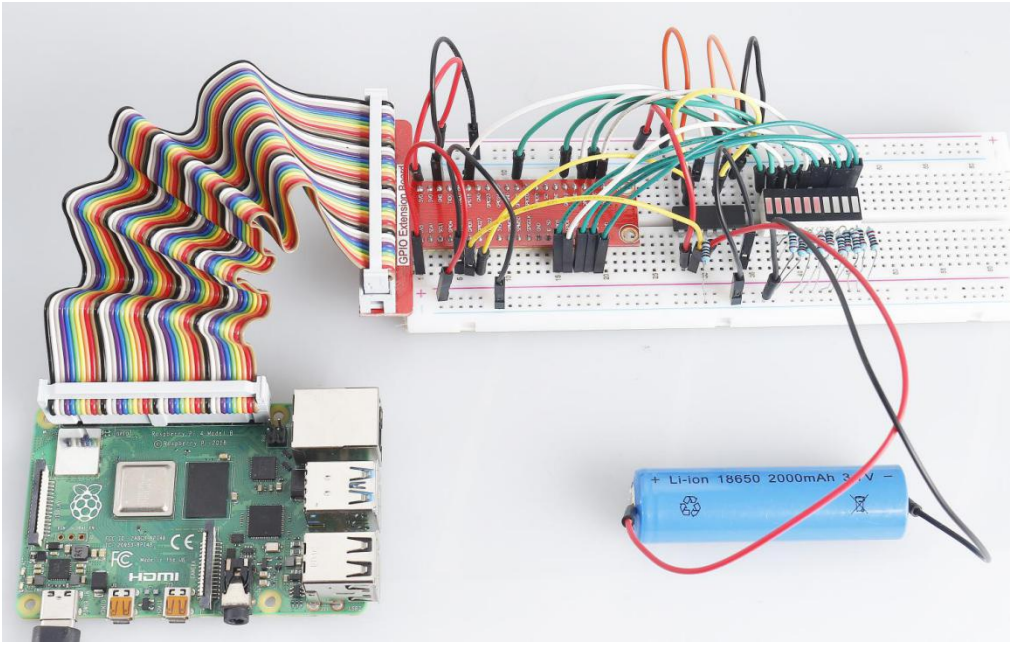
Diese Funktion dient für die Steuerung von Ein- und Ausschalten der 10 LEDs am LED-Bargraphen. Wir geben diesen 10 LEDs hohe Niveau, damit sie zuerst ausgeschaltet sind, und entscheiden dann, wie viele LEDs aufleuchten, indem wir den empfangenen Analogwert ändern.

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

`analogVal` erzeugt Werte (0-255) mit variierenden Spannungswerten (0-5V). Wenn beispielsweise 3V an einer Batterie erkannt werden, wird der entsprechende Wert 152 auf dem Voltmeter angezeigt.

Die 10 LEDs auf dem LED-Balkendiagramm dienen zur Anzeige der `analogVal` Messwerte. 255/10 = 25, also alle 25 erhöht sich der Analogwert, eine weitere LED leuchtet auf, z. B. wenn “analogVal = 150 (ca. 3 V), leuchten 6 LEDs”.

Phänomen Bild

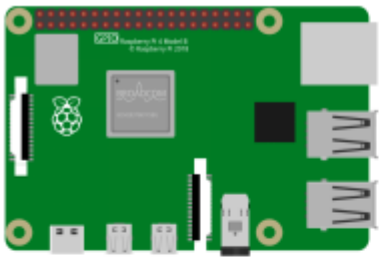
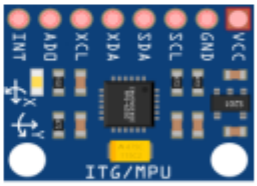
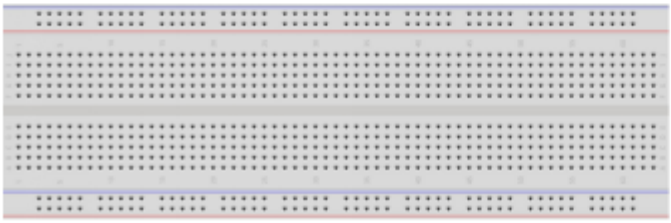
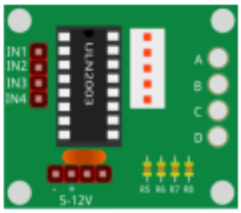


1.9.6 3.1.6 Bewegungssteuerung

Einführung

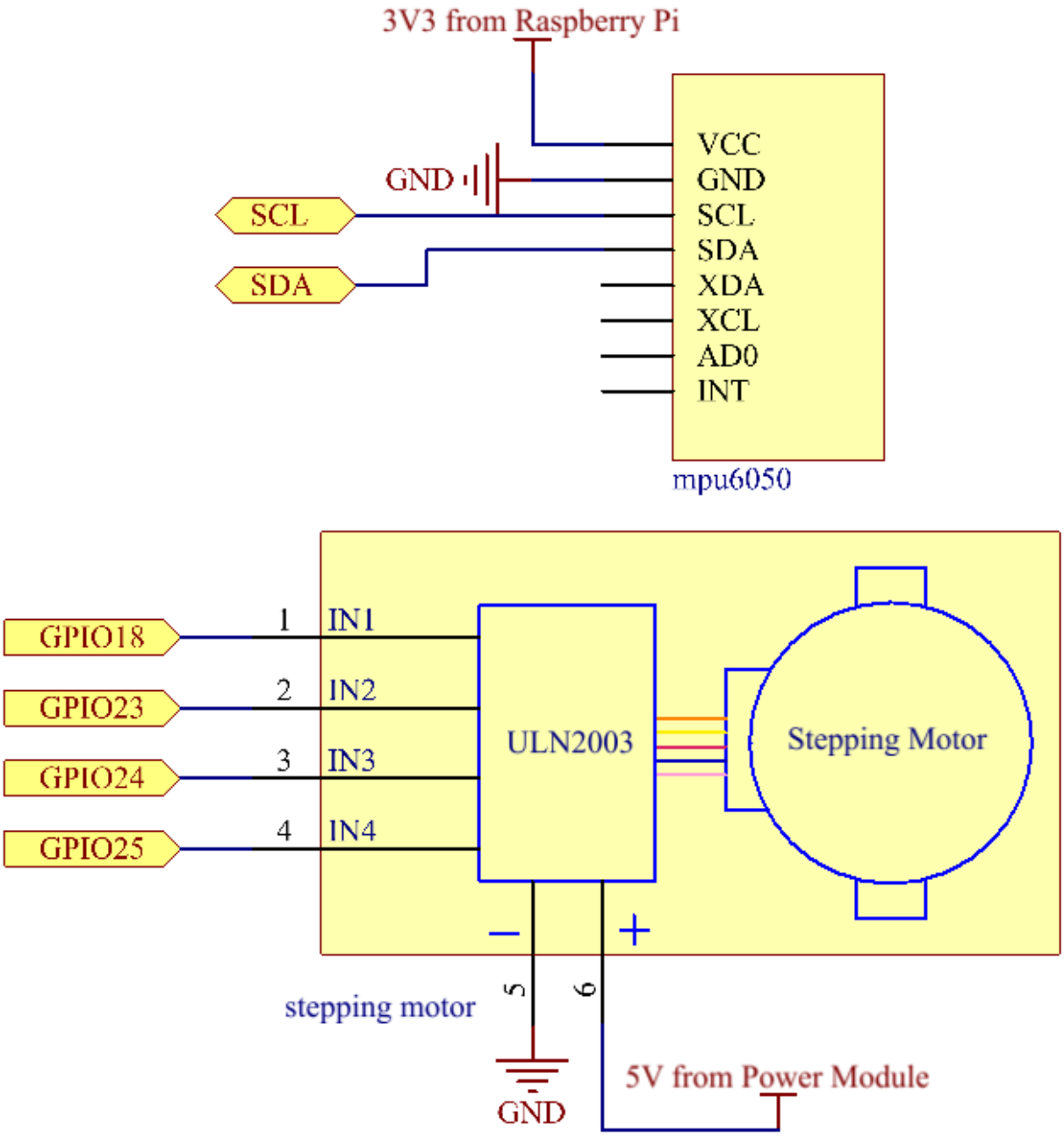
In dieser Lektion werden wir ein einfaches Bewegungserfassungs- und Steuergerät herstellen. Die MPU6050 wird als Sensor und der Schrittmotor als gesteuertes Gerät verwendet. Mit der am Handschuh montierten MPU6050 können Sie den Schrittmotor durch Drehen Ihres Handgelenks steuern.

Komponenten

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 

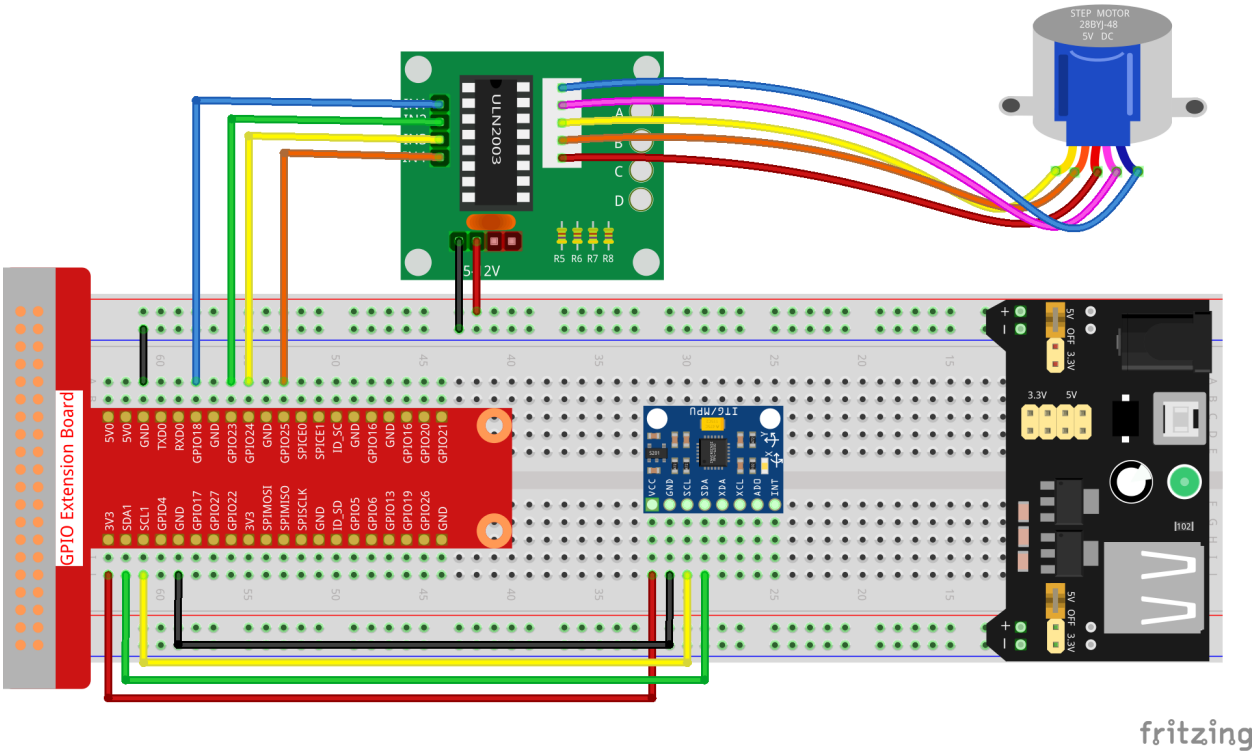
Schematische Darstellung

T-Karte Name	physisch	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		


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

Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.6/
```

Schritt 3: Kompilieren Sie die Kode.

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

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Wenn der Neigungswinkel von mpu6050 auf der Y-Achse größer als 45 °C, ist, dreht sich der Schrittmotor während der Kode gegen den Uhrzeigersinn. Bei weniger als -45 °C, dreht sich der Schrittmotor im Uhrzeigersinn.

Code Erklärung

```
double mpu6050(){
    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;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    acclZ_scaled = acclZ / 16384.0;
    double angle=get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled);
    return angle;
}
```

mpu6050 erhält den Neigungswinkel in Richtung der Y-Achse.

```
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);
        }
    }
}
```

Wenn die empfangene Richtung Taste ‚c' ist, dreht sich der Schrittmotor im Uhrzeigersinn; wenn die Taste ‚a' ist, dreht der Motor gegen den Uhrzeigersinn. Siehe *1.3.3 Schrittmotor* für weitere Details zur Berechnung der Drehrichtung des Schrittmotors.

```
int main()
{
    setup();
    double angle;
    while(1) {
        angle = mpu6050();
        if (angle >=45){rotary('a');}
        else if (angle<=-45){rotary('c');}
    }
    return 0;
}
```

Der Neigungswinkel in Richtung der Y-Achse wird von mpu6050 abgelesen. Wenn er größer als 45 °C, ist, dreht sich der Schrittmotor gegen den Uhrzeigersinn. Bei weniger als -45 °C, dreht sich der Schrittmotor im Uhrzeigersinn.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.6_MotionControl.py
```

Wenn der Neigungswinkel von mpu6050 auf der Y-Achse größer als 45 ℃, ist, dreht sich der Schrittmotor während der Kode gegen den Uhrzeigersinn. Bei weniger als -45 ℃, dreht sich der Schrittmotor im Uhrzeigersinn.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
import RPi.GPIO as GPIO
import smbus
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):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

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)


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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        elif angle <=-45:
            rotary('c')

def destroy():
    GPIO.cleanup()

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

Code Erklärung

```
def mpu6050():
    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
    accel_zout = read_word_2c(0x3f)
    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0
    angle=get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_zout_scaled)
    return angle
```

mpu6050 erhält den Neigungswinkel in Richtung der Y-Achse.

```
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)
```

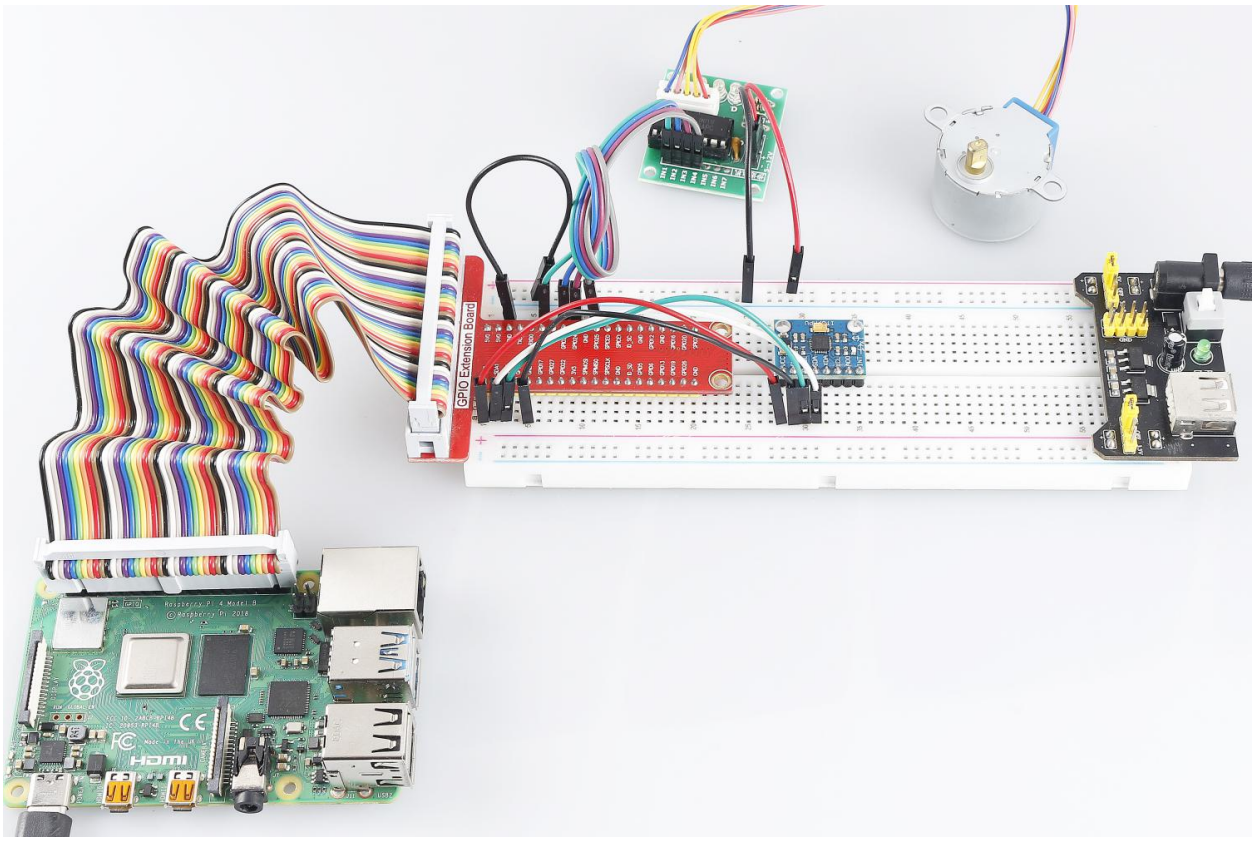
Wenn die empfangene Richtung Taste ‚c' ist, dreht sich der Schrittmotor im Uhrzeigersinn; wenn die Taste ‚a' ist, dreht der Motor gegen den Uhrzeigersinn. Siehe *1.3.3 Schrittmotor* für weitere Details zur Berechnung der Drehrichtung des Schrittmotors.

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

Der Neigungswinkel in Richtung der Y-Achse wird aus mpu6050 abgelesen. Wenn er größer als 45 °C, ist, wird

`rotary()` aufgerufen, damit sich der Schrittmotor gegen den Uhrzeigersinn dreht. Bei weniger als -45 ℃, dreht sich der Schrittmotor im Uhrzeigersinn.

Phänomen Bild

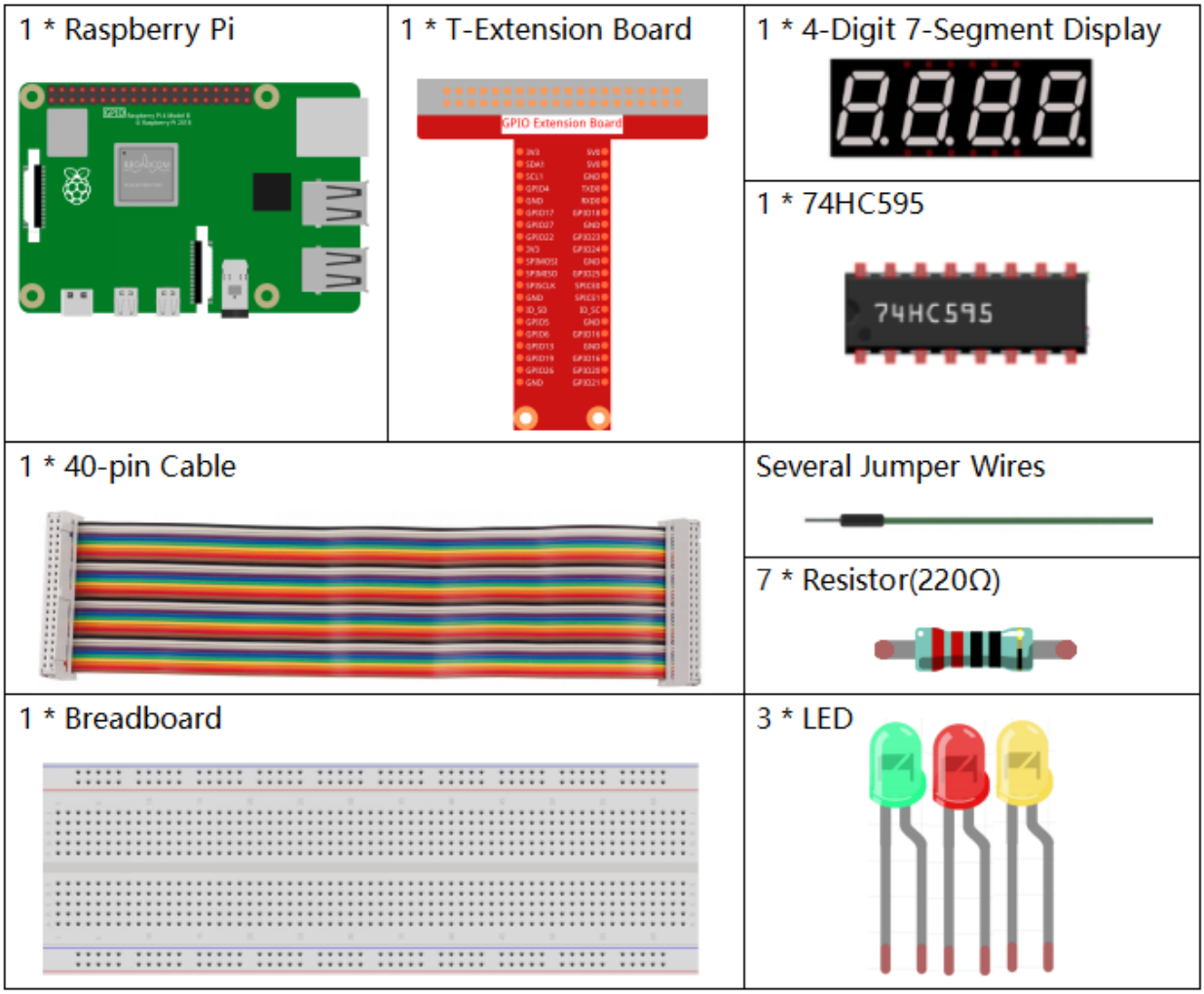


1.9.7 3.1.7 Ampel

Einführung

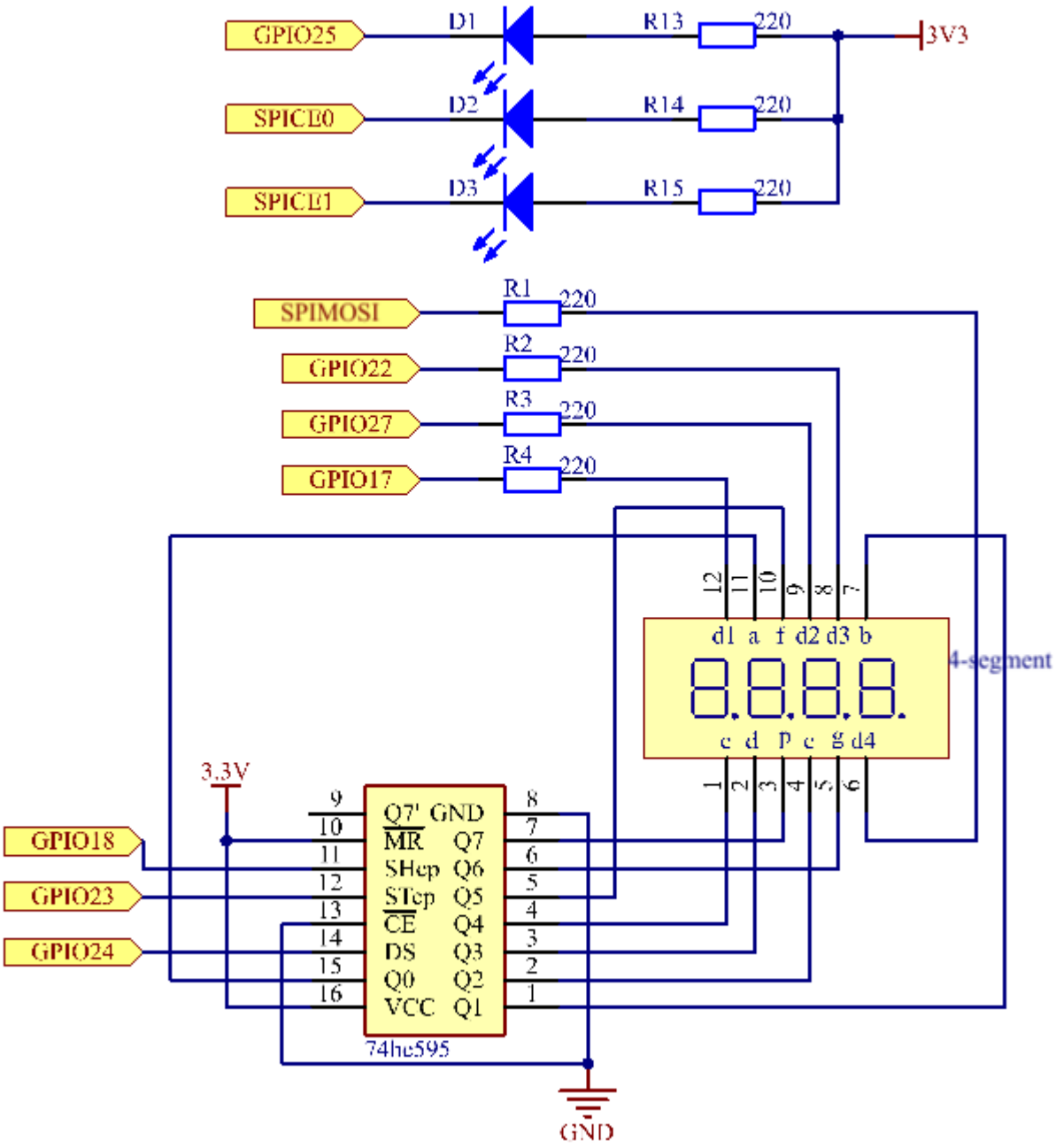
In diesem Projekt werden wir dreifarbige LED-Lichter verwenden, um den Wechsel der Ampeln zu realisieren, und eine vierstellige 7-Segment-Anzeige wird verwendet, um das Timing jedes Verkehrszustands anzuzeigen.

Komponenten



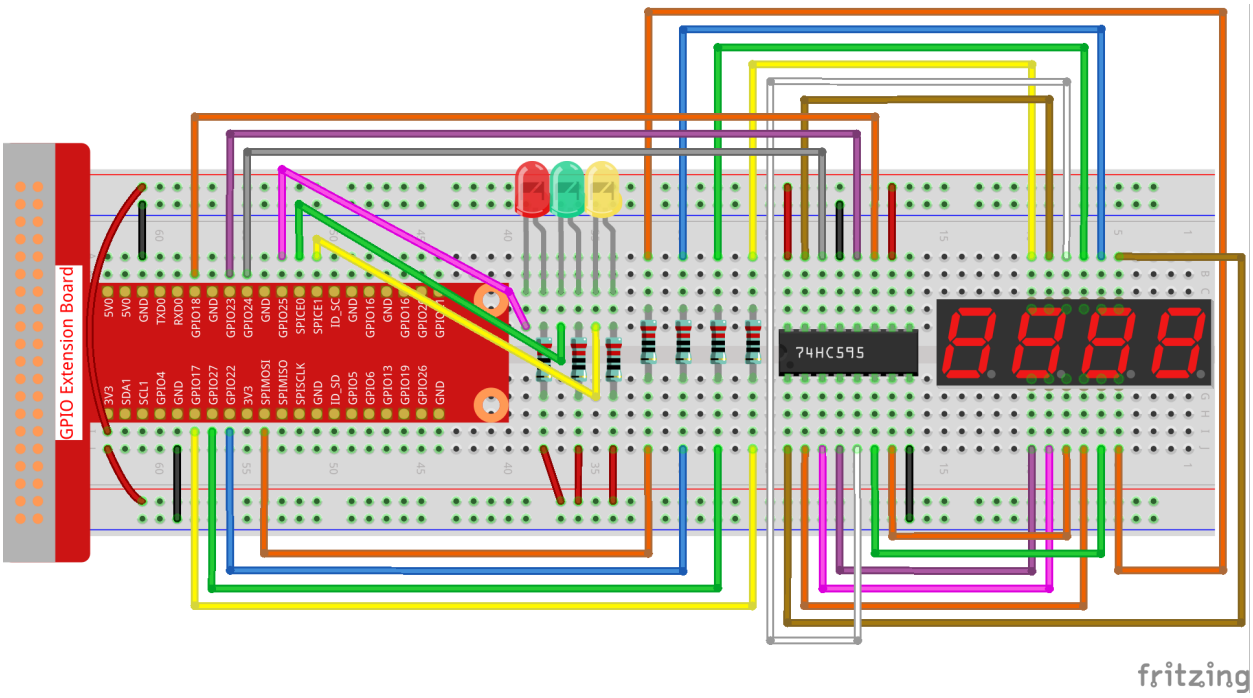
Schematische Darstellung

T-Karte Name	physisch	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


GPIO25
D1
R13
220
3V3
SPICE0
D2
R14
220
SPICE1
D3
R15
220
SPIMOSI
R1
220
GPIO22
R2
220
GPIO27
R3
220
GPIO17
R4
220
12 11 10 9 8 7
d1 a f d2 d3 b
8.8.8.8.
c d p c g d4
4-segment
1 2 3 4 5 6
3.3V
GPIO18
GPIO23
GPIO24
9 10 11 12 13 14 15 16
Q7' GND
MR Q7
SHcp Q6
STcp Q5
CE Q4
DS Q3
Q0 Q2
VCC Q1
8 7 6 5 4 3 2 1
74hc595
GND

Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.7/
```

Schritt 3: Kompilieren.

```
gcc 3.1.7_TrafficLight.c -lwiringPi
```

Schritt 4: Ausführen.

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

Während die Kode ausgeführt wird, simulieren LEDs den Farbwechsel von Ampeln. Zuerst leuchtet die rote LED 60 Sekunden lang, dann leuchtet die grüne LED 30 Sekunden lang. Als nächstes leuchtet die gelbe LED 5 Sekunden lang auf. Danach leuchtet die rote LED erneut für 60s. Auf diese Weise wird diese Reihe von Aktionen wiederholt ausgeführt.

Code Erklärung

```
#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};
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
void pickDigit(int digit);
void hc595_shift(int8_t data);
void clearDisplay();
void display();
```

Diese Codes werden verwendet, um die Funktion der Zahlenanzeige von 4-stelligen 7-Segment-Anzeigen zu realisieren. Siehe *1.1.5 4-stellige 7-Segment-Anzeige* des Dokuments für weitere Details. Hier verwenden wir die Codes, um den Countdown der Ampelzeit anzuzeigen.

```
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);
}
```

Die Kode dienen zum Ein- und Ausschalten der LED.

```
int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 60;

void timer(int  timer1){       //Timer function
    if(timer1 == SIGALRM){
        counter --;
        alarm(1);
        if(counter == 0){
            if(colorState == 0) counter = greenLight;
            if(colorState == 1) counter = yellowLight;
            if(colorState == 2) counter = redLight;
            colorState = (colorState+1)%3;
        }
        printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);
    }
}
```

Die Codes dienen zum Ein- und Ausschalten des Timers. Weitere Informationen finden Sie unter *1.1.5 4-stellige 7-Segment-Anzeige* . Wenn der Timer hier auf Null zurückkehrt, wird colorState auf LED umgeschaltet und dem Timer wird ein neuer Wert zugewiesen.

```
void loop()
{
    while(1){
    display();
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    lightup();
    }
}

int main(void)
{
    //...
    signal(SIGALRM,timer);
    alarm(1);
    loop();
    return 0;
}
```

Der Timer wird in der Funktion `main()` gestartet. Verwenden Sie in der Funktion `loop()` die Schleife `while(1)` und rufen Sie die Funktionen von 4-stelligem 7-Segment und LED auf.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.7_TrafficLight.py
```

Während die Kode ausgeführt wird, simulieren LEDs den Farbwechsel von Ampeln. Zuerst leuchtet die rote LED 60 Sekunden lang, dann leuchtet die grüne LED 30 Sekunden lang. Als nächstes leuchtet die gelbe LED 5 Sekunden lang auf. Danach leuchtet die rote LED erneut für 60s. Auf diese Weise wird diese Reihe von Aktionen wiederholt ausgeführt. Währenddessen zeigt die 4-stellige 7-Segment-Anzeige kontinuierlich die Countdown-Zeit an.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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)
    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):
            counter= greenLight
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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()
        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()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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()
```

Code Erklärung

```
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():
```

Diese Codes werden verwendet, um die Funktion der Zahlenanzeige des 4-stelligen 7-Segments zu realisieren. Siehe *1.1.5 4-stellige 7-Segment-Anzeige* des Dokuments für weitere Details. Hier verwenden wir die Codes, um den Countdown der Ampelzeit anzuzeigen.

```
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)
```

Die Kode dienen zum Ein- und Ausschalten der LED.

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

colorState=0
counter = 60
timer1 = 0

def timer():        #timer function
    global counter
    global colorState
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

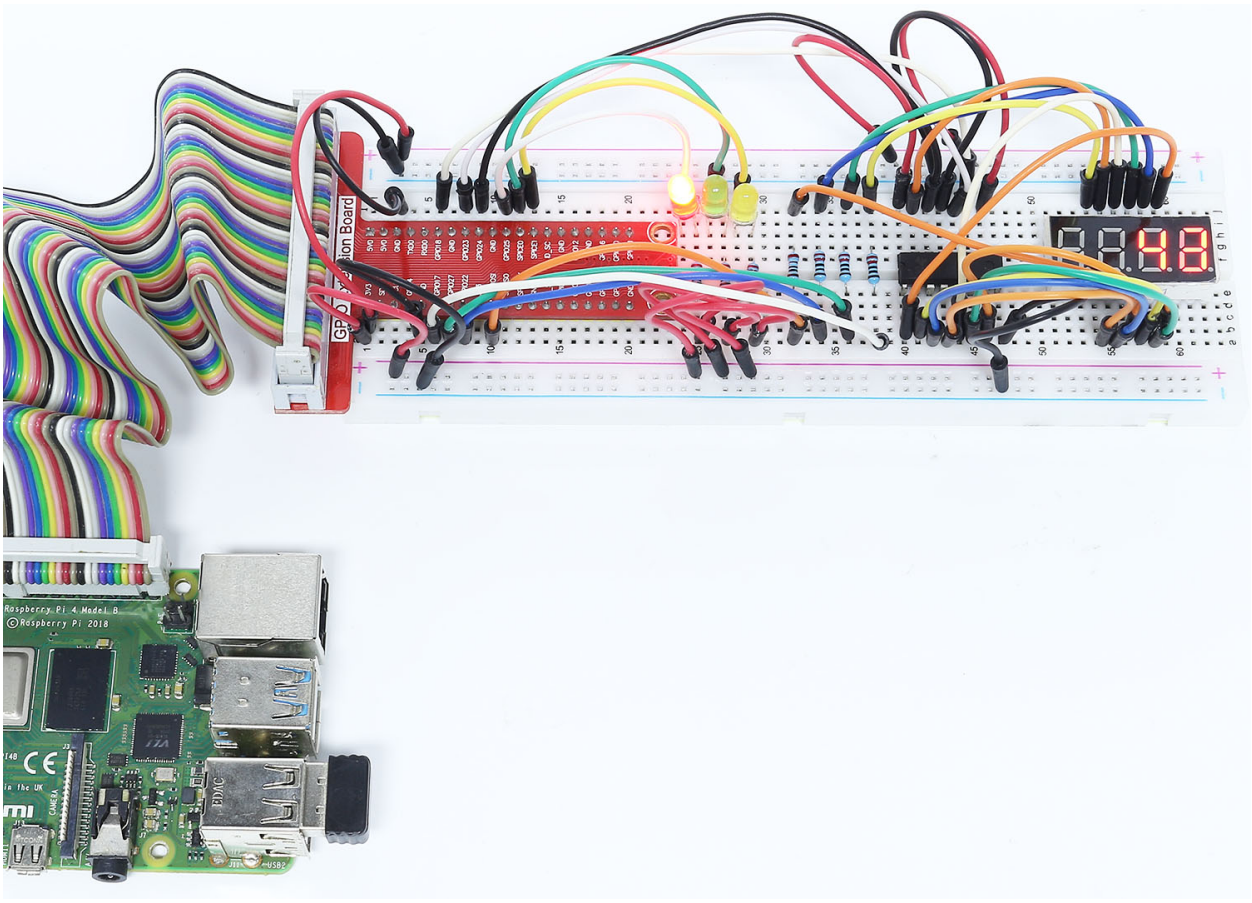
```
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
        colorState=(colorState+1)%3
    print ("counter : %d    color: %s "%(counter,lightColor[colorState]))
```

Die Codes dienen zum Ein- und Ausschalten des Timers. Weitere Informationen finden Sie unter *1.1.5 4-stellige 7-Segment-Anzeige* . Wenn der Timer hier auf Null zurückkehrt, wird colorState auf LED umgeschaltet und dem Timer wird ein neuer Wert zugewiesen.

```
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()
```

Starten Sie in der Funktion `setup()` den Timer. In der Funktion `loop()` wird eine `While True` verwendet: Rufen Sie die relativen Funktionen von 4-stelligem 7-Segment und LED kreisförmig auf.

Phänomen Bild

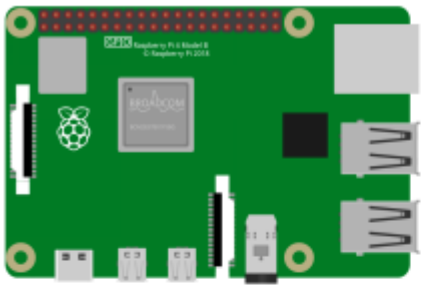
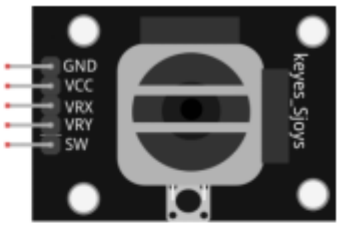
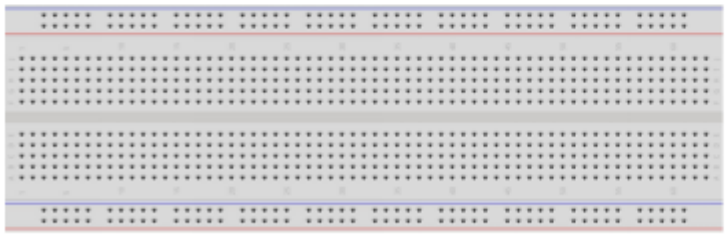
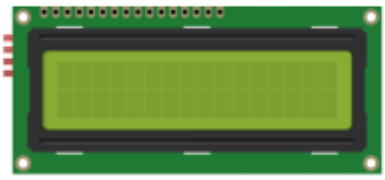


1.9.8 3.1.8 Überhitzungsmonitor

Einführung

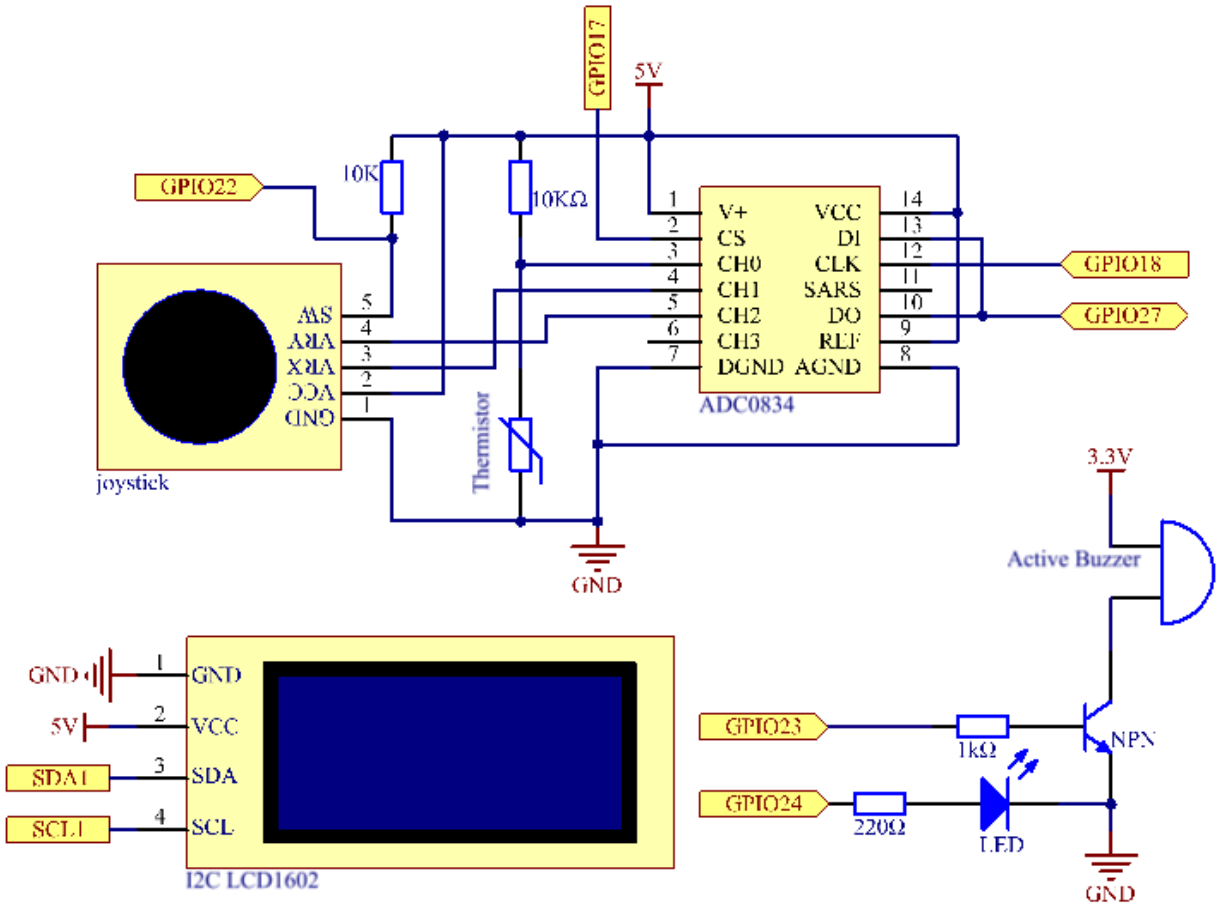
Möglicherweise möchten Sie ein Überhitzungsüberwachungsgerät herstellen, das für verschiedene Situationen gilt, z. B. im Werk, wenn wir einen Alarm und das rechtzeitige automatische Ausschalten der Maschine bei Überhitzung des Stromkreises wünschen. In dieser Lektion verwenden wir Thermistor, Joystick, Summer, LED und LCD, um ein intelligentes Temperaturüberwachungsgerät zu erstellen, dessen Schwelle einstellbar ist.

Komponenten

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

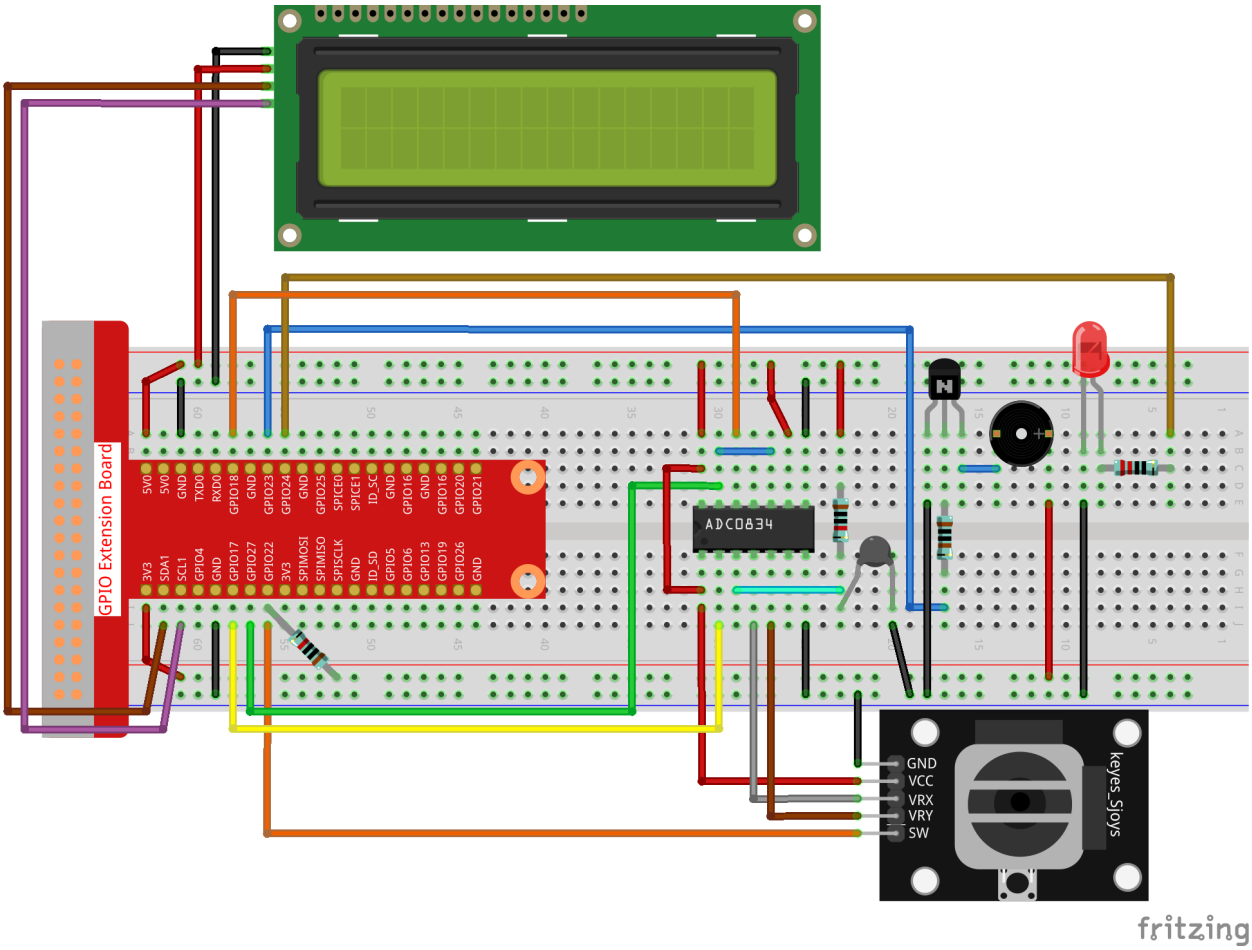
Schematische Darstellung

T-Karte Name	physisch	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		



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.8/
```

Schritt 3: Kompilieren Sie der Kode.

```
gcc 3.1.8_OverheatMonitor.c -lwiringPi -lm
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Während die Kode ausgeführt wird, werden die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.

Der Joystick dient zum Drücken, um die Hochtemperaturschwelle anzupassen. Durch Umschalten des Joystick in Richtung X-Achse und Y-Achse kann der aktuelle Hochtemperaturschwellenwert angepasst (nach oben oder unten gedreht)

werden. Drücken Sie den Joystick erneut, um den Schwellenwert auf den Anfangswert zurückzusetzen.

Code Erklärung

```
int get_joystick_value(){
    uchar x_val;
    uchar y_val;
    x_val = get_ADC_Result(1);
    y_val = get_ADC_Result(2);
    if (x_val > 200){
        return 1;
    }
    else if(x_val < 50){
        return -1;
    }
    else if(y_val > 200){
        return -10;
    }
    else if(y_val < 50){
        return 10;
    }
    else{
        return 0;
    }
}
```

Diese Funktion liest die Werte von X und Y. Wenn X> 200 ist, wird „1“ zurückgegeben. X<50, return „-1“; y> 200, geben Sie „-10“ zurück, und y<50, geben Sie „10“ zurück.

```
void upper_tem_setting(){
    write(0, 0, "Upper Adjust:");
    int change = get_joystick_value();
    upperTem = upperTem + change;
    char str[6];
    snprintf(str,3,"%d",upperTem);
write(0,1,str);
int len;
  len = strlen(str);
  write(len,1,"             ");
    delay(100);
}
```

This function is for adjusting the threshold and displaying it on the I2C LCD1602.

```
double temperature(){
    unsigned char temp_value;
    double Vr, Rt, temp, cel, Fah;
    temp_value = get_ADC_Result(0);
    Vr = 5 * (double)(temp_value) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    return cel;
}
```

Lesen Sie den Analogwert des CH0 (Thermistor) von ADC0834 ab und wandeln Sie ihn dann in einen Temperaturwert um.

```
void monitoring_temp(){
    char str[6];
    double cel = temperature();
    snprintf(str,6,"%.2f",cel);
    write(0, 0, "Temp: ");
    write(6, 0, str);
    snprintf(str,3,"%d",upperTem);
    write(0, 1, "Upper: ");
    write(7, 1, str);
    delay(100);
    if(cel >= upperTem){
        digitalWrite(buzzPin, HIGH);
        digitalWrite(LedPin, HIGH);
    }
    else if(cel < upperTem){
        digitalWrite(buzzPin, LOW);
        digitalWrite(LedPin, LOW);
    }
}
```

Während die Kode ausgeführt wird, werden die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.

```
int main(void)
{
    setup();
    int lastState =1;
    int stage=0;
    while (1)
    {
        int currentState = digitalRead(Joy_BtnPin);
        if(currentState==1 && lastState == 0){
            stage=(stage+1)%2;
            delay(100);
            lcd_clear();
        }
        lastState=currentState;
        if (stage==1){
            upper_tem_setting();
        }
        else{
            monitoring_temp();
        }
    }
    return 0;
}
```

Die Funktion `main()` enthält den gesamten Programmablauf wie folgt:

1. Wenn das Programm startet, ist der Anfangswert der Stufe 0, und die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 werden auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der

Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.

2. Drücken Sie den Joystick, und die Stufe ist 1, und Sie können die Hochtemperaturschwelle einstellen. Durch Umschalten des Joysticks in Richtung X-Achse und Y-Achse kann der aktuelle Schwellenwert angepasst (nach oben oder unten gedreht) werden. Drücken Sie den Joystick erneut, um den Schwellenwert auf den Anfangswert zurückzusetzen.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.8_OverheatMonitor.py
```

Während die Kode ausgeführt wird, werden die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.

Der Joystick dient zum Drücken, um die Hochtemperaturschwelle anzupassen. Durch Umschalten des Joystick in Richtung X-Achse und Y-Achse kann der aktuelle Hochtemperaturschwellenwert angepasst (nach oben oder unten gedreht) werden. Drücken Sie den Joystick erneut, um den Schwellenwert auf den Anfangswert zurückzusetzen.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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)
    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:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            stage=(stage+1)%2
            time.sleep(0.1)
            LCD1602.clear()
        lastState=currentState
        if stage == 1:
            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()
```

Code Erklärung

```
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
```

Diese Funktion liest die Werte von X und Y. Wenn X> 200 ist, wird „1“ zurückgegeben. X<50, return „-1“; y> 200, geben Sie „-10“ zurück, und y<50, geben Sie „10“ zurück.

```
def upper_tem_setting():
    global upperTem
    LCD1602.write(0, 0, 'Upper Adjust: ')
    change = int(get_joystick_value())
    upperTem = upperTem + change
LCD1602.write(0, 1, str(upperTem))
LCD1602.write(len(strUpperTem),1, '               ')
    time.sleep(0.1)
```

Diese Funktion dient zum Anpassen des Schwellenwerts und zum Anzeigen auf dem I2C LCD1602.

```
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)
```

Lesen Sie den Analogwert des CH0 (Thermistor) von ADC0834 ab und wandeln Sie ihn dann in einen Temperaturwert um.

```
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)
```

Während die Kode ausgeführt wird, werden die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.

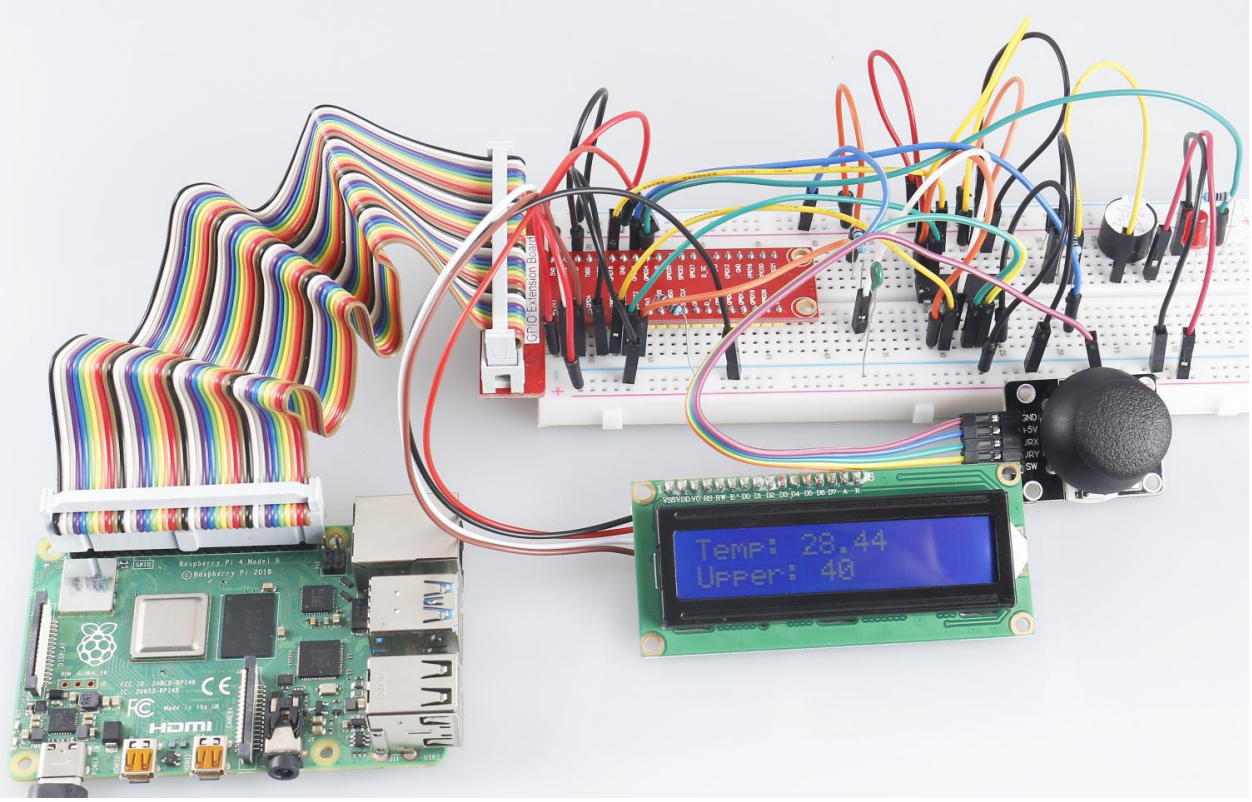
```
def loop():
    lastState=1
    stage=0
    while True:
        currentState=GPIO.input(Joy_BtnPin)
        if currentState==1 and lastState ==0:
            stage=(stage+1)%2
            time.sleep(0.1)
            LCD1602.clear()
        lastState=currentState
        if stage == 1:
            upper_tem_setting()
        else:
            monitoring_temp()
```

Die Funktion `main()` enthält den gesamten Programmablauf wie folgt:

1. Wenn das Programm startet, ist der Anfangswert der Stufe 0, und die aktuelle Temperatur und der Hochtemperaturschwellenwert 40 werden auf dem I2C LCD1602 angezeigt. Wenn die aktuelle Temperatur größer als der Schwellenwert ist, werden der Summer und die LED gestartet, um Sie zu alarmieren.
2. Drücken Sie den Joystick, und die Stufe ist 1, und Sie können die Hochtemperaturschwelle einstellen. Durch Umschalten des Joysticks in Richtung X-Achse und Y-Achse kann der aktuelle Hochtemperaturschwellenwert angepasst (nach oben oder unten gedreht) werden. Drücken Sie den Joystick erneut, um den Schwellenwert auf

den Anfangswert zurückzusetzen.

Phänomen Bild



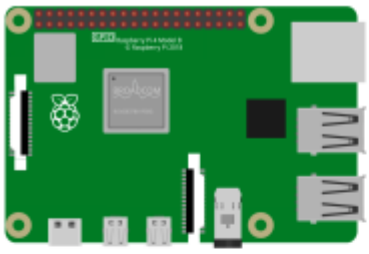
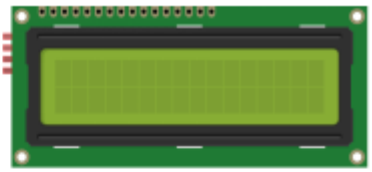
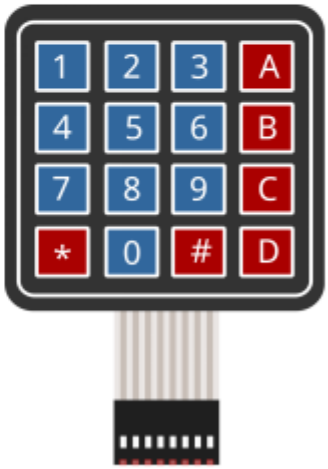
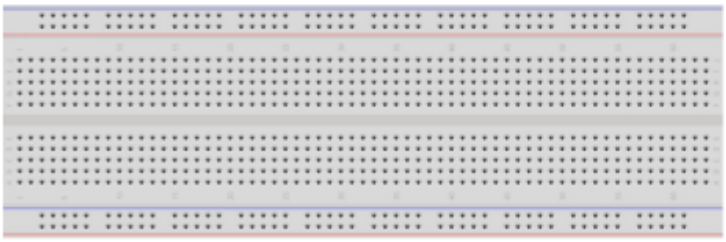
1.9.9 3.1.9 Passwortsperre

Einführung

In diesem Projekt werden wir eine Tastatur und ein LCD verwenden, um ein Nummerschloss herzustellen. Auf dem LCD wird eine entsprechende Aufforderung angezeigt, Ihr Passwort auf der Tastatur einzugeben. Wenn das Passwort korrekt eingegeben wurde, wird „Richtig“ angezeigt.

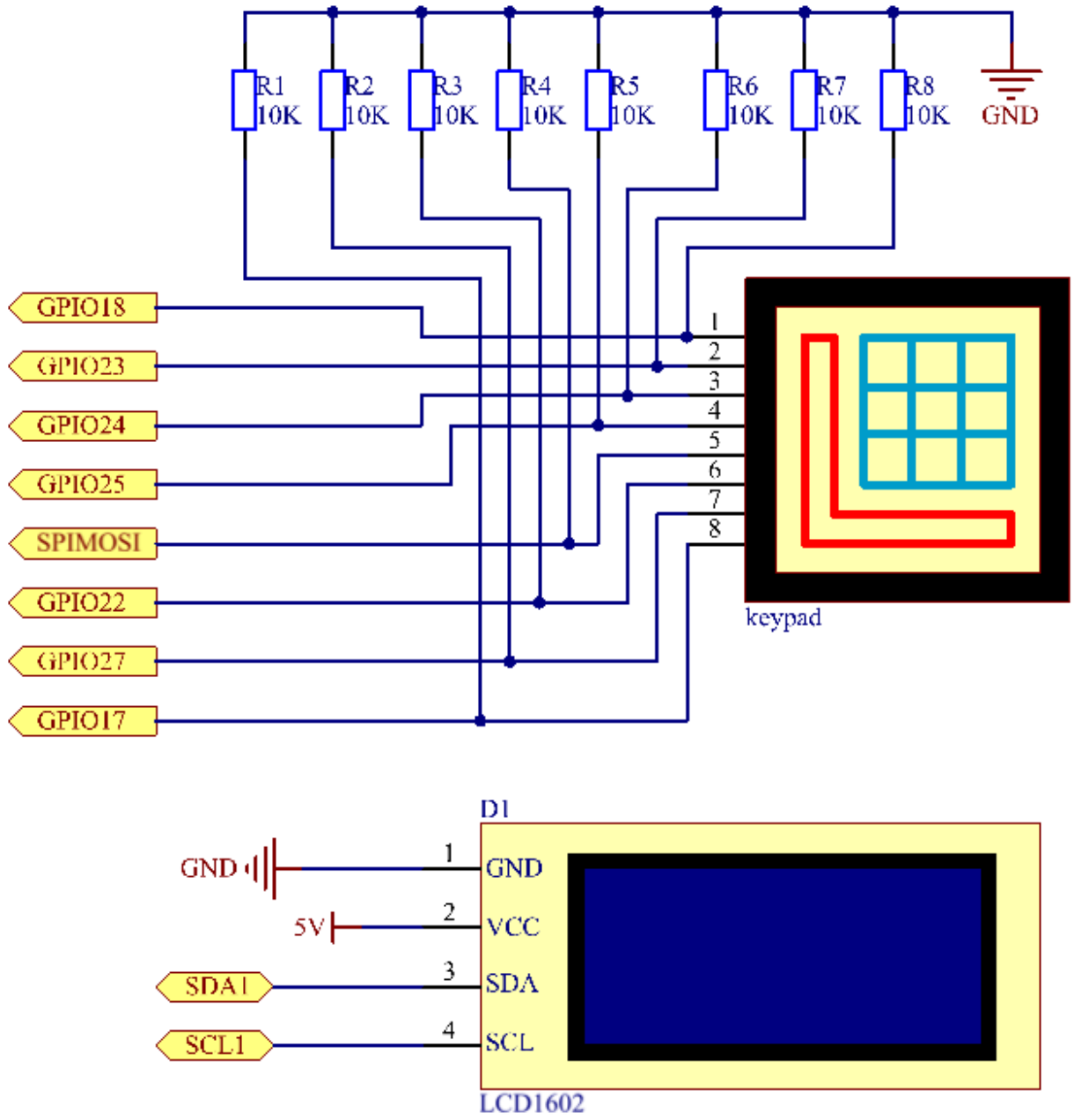
Auf der Grundlage dieses Projekts können wir zusätzliche elektronische Komponenten wie Summer, LED usw. hinzufügen, um verschiedene experimentelle Phänomene für die Passworteingabe hinzuzufügen.

Komponenten

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>1 * I2C LCD1602
  </td></tr>
<tr><td></td><td></td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td rowspan="2">1 * Keypad
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td></tr>
<tr><td colspan="2"></td><td>8 * Resistor 10KΩ
 </td></tr>
</table>

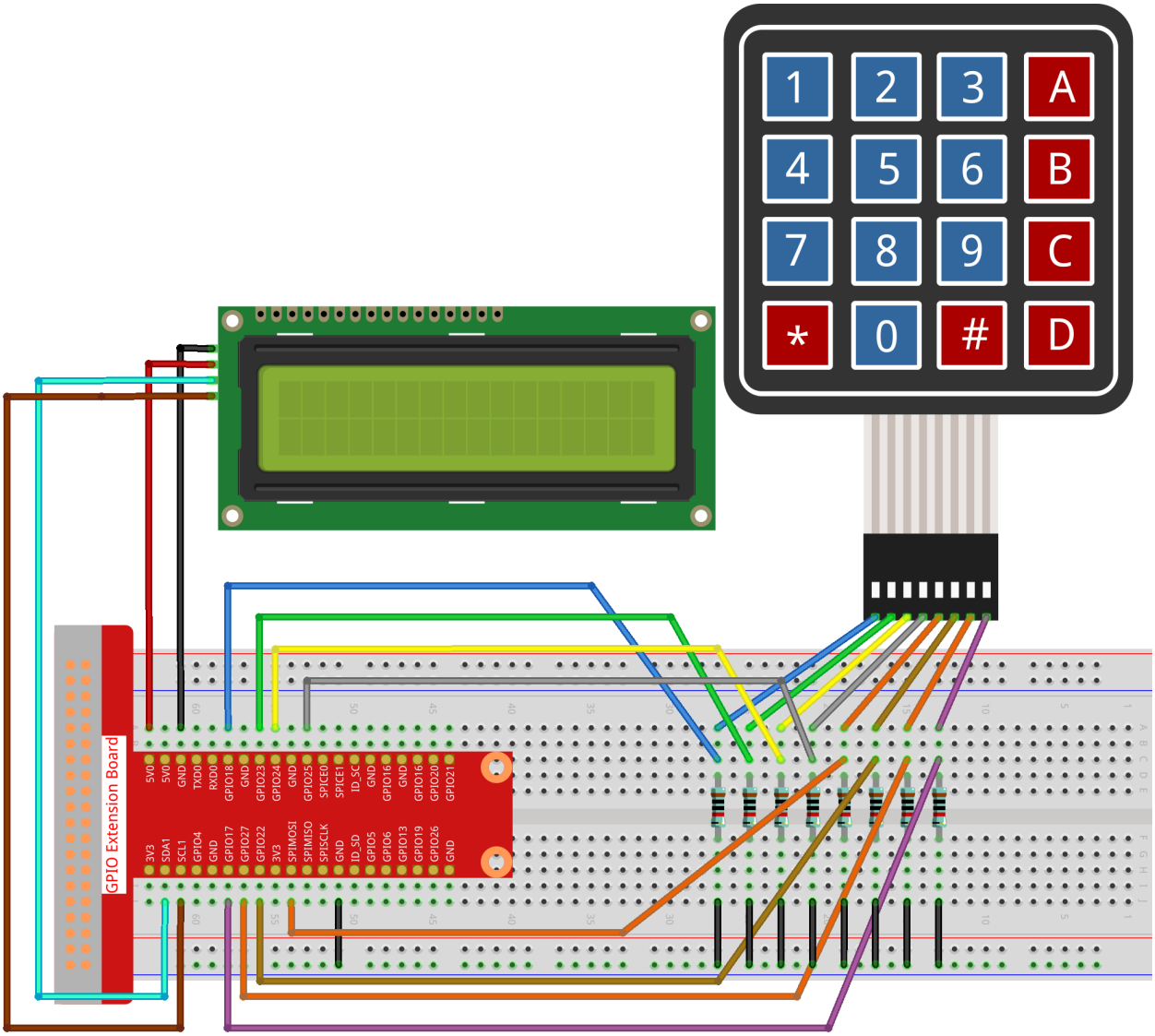
Schematische Darstellung

T-Karte Name	physisch	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		



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.9/
```

Schritt 3: Kompilieren.

```
gcc 3.1.9_PasswordLock.cpp -lwiringPi
```

Schritt 4: Ausführen.

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

Nachdem der Kode ausgeführt wurde, wird die Tastatur über die Tastatur eingegeben. Wenn auf dem LCD1602 die Meldung „RICHTIG“ angezeigt wird, ist das Kennwort nicht falsch. Andernfalls wird „FALSCHER SCHLÜSSEL“ angezeigt.

Code Erklärung

```
#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'};
```

Hier definieren wir die Länge des Kennworts LENS, des Schlüssel-Array-Tastenschlüssel-Arrays KEYS und des Arrays, in dem das richtige Kennwort gespeichert ist.

```
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);
```

Es gibt eine Deklaration der Unterfunktionen des Matrix-Tastaturcodes, siehe *2.1.5 Tastatur* dieses Dokuments für weitere Details.

```
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[]);
```

Es gibt eine Deklaration der Unterfunktionen des LCD1062-Codes, siehe *1.1.7 I2C LCD1602* dieses Dokuments für weitere Details.

```
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");
                        }
                ...
```

Lesen Sie den Schlüsselwert und speichern Sie ihn im Testarray-Testwort. Wenn die Anzahl der gespeicherten Schlüsselwerte mehr als 4 beträgt, wird die Richtigkeit des Kennworts automatisch überprüft und die Überprüfungsergebnisse werden auf der LCD-Oberfläche angezeigt.

```
int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
        {return 0;}
    }
    return 1;
}
```

Überprüfen Sie die Richtigkeit des Passworts. Geben Sie 1 zurück, wenn das Passwort korrekt eingegeben wurde, und 0, wenn nicht.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.9_PasswordLock.py
```

Nachdem der Kode ausgeführt wurde, wird über die Tastatur das Kennwort eingegeben: 1984. Wenn auf dem LCD1602 die Meldung „RICHTIG“ angezeigt wird, ist das Kennwort nicht falsch. Andernfalls wird „FALSCHER SCHLÜSSEL“ angezeigt.

Nachdem der Kode ausgeführt wurde, wird über die Tastatur das Kennwort eingegeben: 1984. Wenn auf dem LCD1602 die Meldung **CORRECT** angezeigt wird, ist das Kennwort nicht falsch. Andernfalls wird **WRONG KEY** angezeigt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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 = []
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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

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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            keyIndex+=1
            if (keyIndex is LENS):
                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()
```

Code Erklärung

```
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"]
```

Hier definieren wir die Länge des Passworts LENS, die Array-Tasten, in denen die Matrix-Tastaturtasten gespeichert sind, und das Array-Passwort, in dem das richtige Passwort gespeichert ist.

```
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)
...
```

Diese Klasse ist der Code, der die Werte der gedrückten Tasten liest. Siehe *2.1.5 Tastatur* dieses Dokuments für weitere Details.

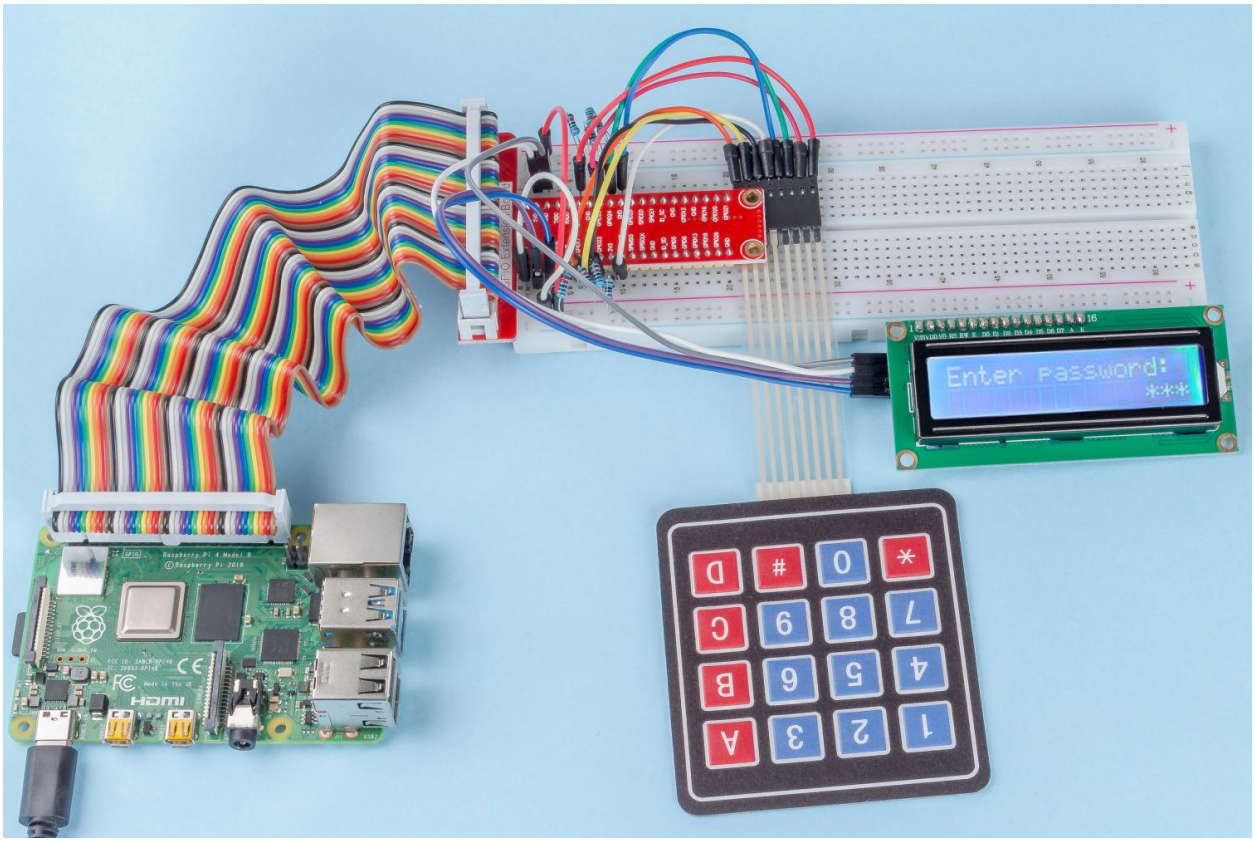
```
while(True):
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, pressed_keys)
            testword[keyIndex]=pressed_keys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

Lesen Sie den Schlüsselwert und speichern Sie ihn im Testarray-Testwort. Wenn die Anzahl der gespeicherten Schlüsselwerte mehr als 4 beträgt, wird die Richtigkeit des Kennworts automatisch überprüft und die Überprüfungsergebnisse werden auf der LCD-Oberfläche angezeigt.

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

Überprüfen Sie die Richtigkeit des Passworts. Geben Sie 1 zurück, wenn das Passwort korrekt eingegeben wurde, und 0, wenn nicht.

Phänomen Bild

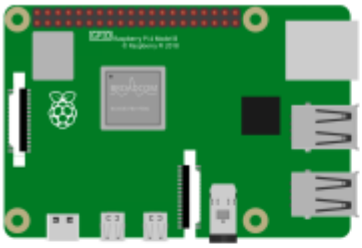


1.9.10 3.1.10 Alarmglocke

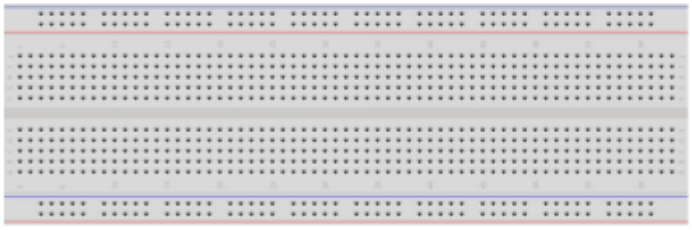
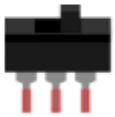
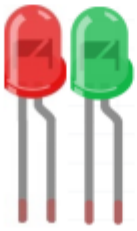
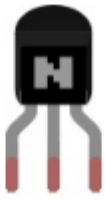
Einführung

In diesem Kurs erstellen wir ein manuelles Alarmgerät. Sie können den Kippschalter durch einen Thermistor oder einen lichtempfindlichen Sensor ersetzen, um einen Temperaturalarm oder einen Lichtalarm auszulösen.

Komponenten

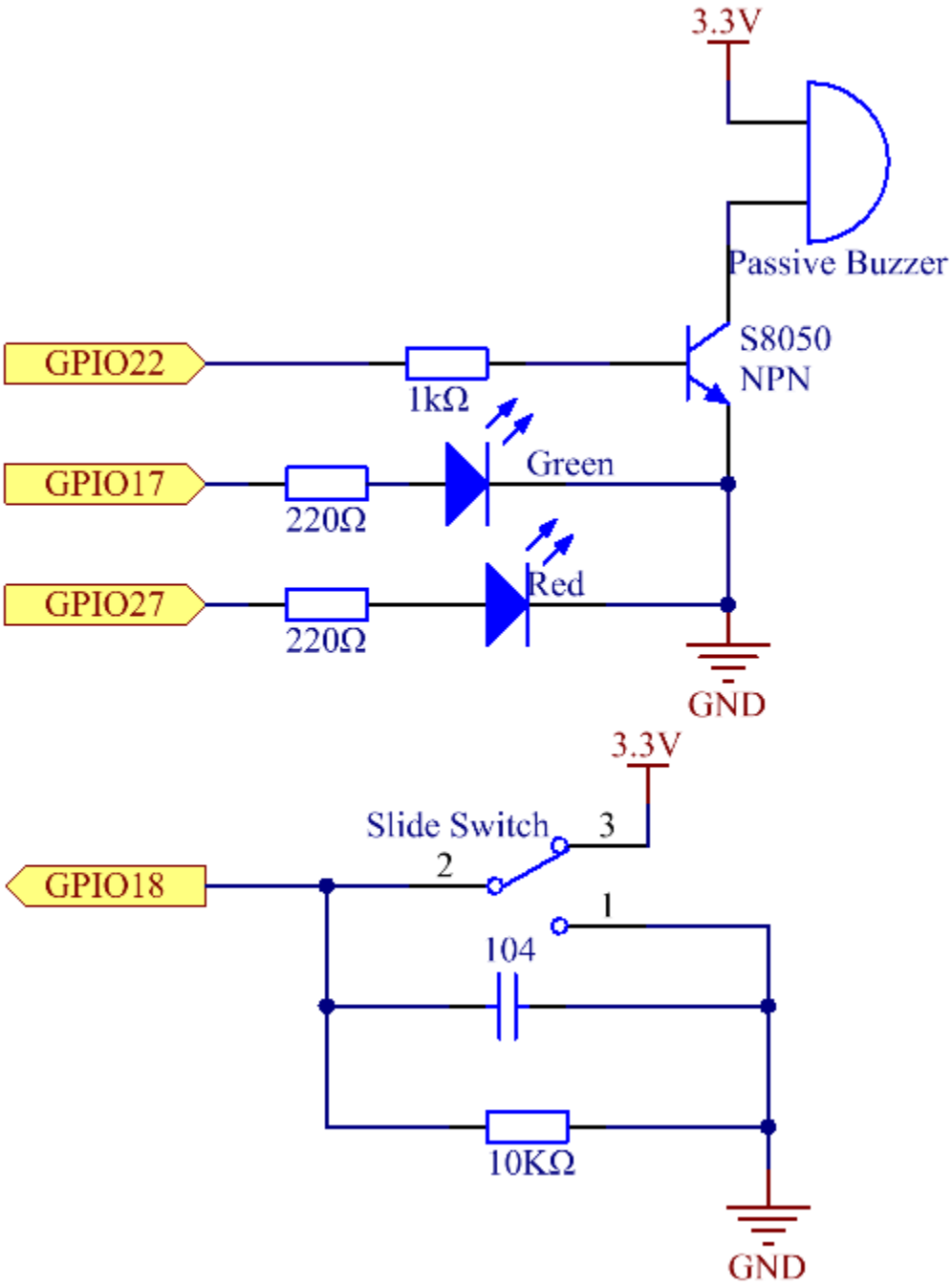
<table>
<tr><td>1 * Raspberry Pi
  </td><td colspan="2">1 * T-Extension Board
 </td><td>1 * Passive Buzzer

1 * Resistor(1kΩ)
 </td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td><td>2 * Resistor(220Ω)

1 * Resistor 10KΩ
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td><td>1 * Slide Switch
  </td></tr>
<tr><td>Several Jumper Wires
 </td><td>2 * LED
  </td><td>1 * S85050 NPN Transistor
  </td><td>1 * 104 Capacitor
 </td></tr>
</table>

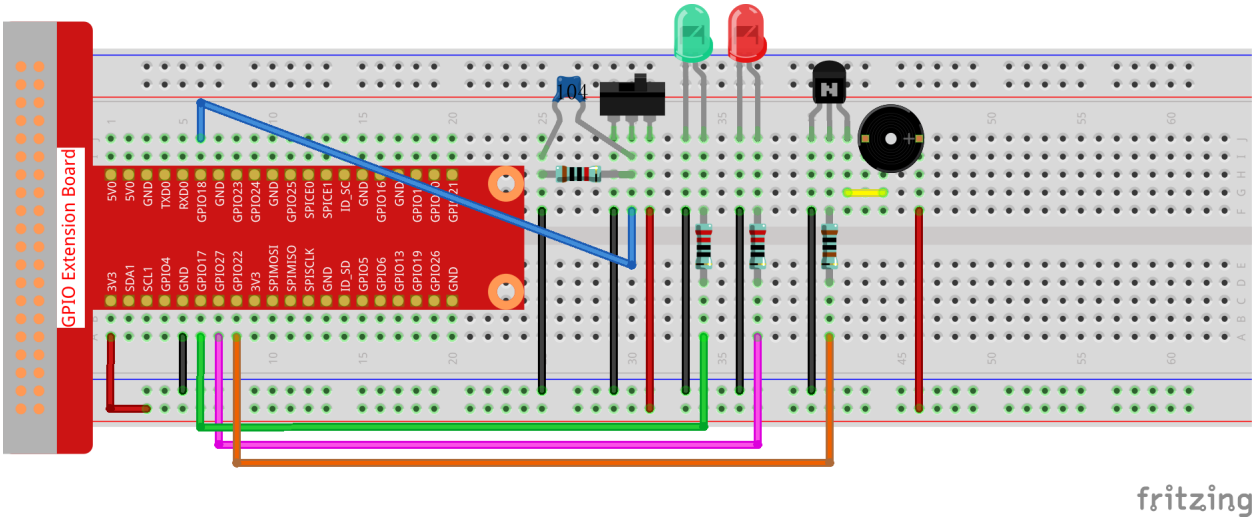
Schematische Darstellung

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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.10/
```

Schritt 3: Kompilieren.

```
gcc 3.1.10_AlarmBell.c -lwiringPi -lpthread
```

Schritt 4: Ausführen.

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

Nach dem Start des Programms wird der Kippschalter nach rechts umgeschaltet und der Summer gibt Alarmtöne aus. Gleichzeitig blinken die roten und grünen LEDs mit einer bestimmten Frequenz.

Code Erklärung

```
#include <pthread.h>
```

In dieser Kode verwenden Sie eine neue Bibliothek, `pthread.h` , die aus einer Reihe allgemeiner Thread-Bibliotheken besteht und Multithreading realisieren kann. Wir fügen den Parameter `-lpthread` zur Kompilierungszeit hinzu, damit die LED und der Summer unabhängig voneinander arbeiten können.

```
void *ledWork(void *arg){
    while(1)
    {
        if(flag==0){
            pthread_exit(NULL);
        }
        digitalWrite(ALedPin,HIGH);
        delay(500);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        digitalWrite(ALedPin,LOW);
        digitalWrite(BLedPin,HIGH);
        delay(500);
        digitalWrite(BLedPin,LOW);
    }
}
```

Die Funktion `ledWork()` hilft beim Einstellen des Arbeitszustands dieser beiden LEDs: Sie leuchtet die grüne LED 0,5 Sekunden lang auf und erlischt dann. In ähnlicher Weise leuchtet die rote LED 0,5 Sekunden lang auf und erlischt dann.

```
void *buzzWork(void *arg){
    while(1)
    {
        if(flag==0){
            pthread_exit(NULL);
        }
        if((note>=800)||(note<=130)){
            pitch = -pitch;
        }
        note=note+pitch;
        softToneWrite(BeepPin,note);
        delay(10);
    }
}
```

Mit der Funktion `buzzWork()` wird der Arbeitszustand des Summers eingestellt. Hier stellen wir die Frequenz zwischen 130 und 800 ein, um sie in einem Intervall von 20 zu akkumulieren oder abzunehmen.

```
void on(){
    flag = 1;
    if(softToneCreate(BeepPin) == -1){
        printf("setup softTone failed !");
        return;
    }
    pthread_t tLed;
    pthread_create(&tLed,NULL,ledWork,NULL);
    pthread_t tBuzz;
    pthread_create(&tBuzz,NULL,buzzWork,NULL);
}
```

In der Funktion `on()`:

1. Definieren Sie die Markierung „flag = 1“, die das Ende des Kontrollthreads angibt.
2. Erstellen Sie einen softwaregesteuerten Ton-Pin `BeepPin`.
3. Erstellen Sie zwei separate Threads, damit die LED und der Summer gleichzeitig arbeiten können.

`pthread_t tLed` : Deklariert einen Thread tLed.

`pthread_create(&tLed,NULL,ledWork,NULL)` : Erstellen Sie den Thread und sein Prototyp lautet wie folgt:

```
int pthread_create(pthread_t *restrict tidp,const pthread_attr_t *restrict_
↪attr,void*(*start_rtn)(void*),void *restrict arg);
```

Geben Sie den Wert zurück

Wenn dies erfolgreich ist, geben Sie „0“ zurück. Andernfalls geben Sie die Fallzahl „-1“ zurück.

Parameter

Der erste Parameter ist ein Zeiger auf die Thread-ID.
Der zweite wird verwendet, um das Thread-Attribut festzulegen.
Die dritte ist die Startadresse der Thread-Running-Funktion.
Der letzte ist derjenige, der die Funktion ausführt.

```
void off(){
    flag = 0;
    softToneStop(BeepPin);
    digitalWrite(ALedPin,LOW);
    digitalWrite(BLedPin,LOW);
}
```

Die Funktion `off()` definiert „flag=0“, um die Threads ledWork und BuzzWork zu verlassen und dann den Summer und die LED auszuschalten.

```
int main(){
    setup();
    int lastState = 0;
    while(1){
        int currentState = digitalRead(switchPin);
        if ((currentState == 1)&&(lastState==0)){
            on();
        }
        else if((currentState == 0)&&(lastState==1)){
            off();
        }
        lastState=currentState;
    }
    return 0;
}
```

`main()` enthält den gesamten Prozess des Programms: Lesen Sie zuerst den Wert des Schiebeschalters; Wenn der Kippschalter nach rechts umgeschaltet ist (der Messwert ist 1), wird die Funktion `on()` aufgerufen, der Summer wird zur Ausgabe von Tönen angesteuert und die rote und die grüne LED blinken. Andernfalls funktionieren der Summer und die LED nicht.

Für Python-Sprachbenutzer

Schritt 2: Verzeichnis wechseln.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.10_AlarmBell.py
```

Nach dem Start des Programms wird der Kippschalter nach rechts umgeschaltet und der Summer gibt Alarmtöne aus. Gleichzeitig blinken die roten und grünen LEDs mit einer bestimmten Frequenz.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)

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():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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():
    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()
```

Code Erklärung

```
import threading
```

Hier importieren wir das Threading-Modul und es ermöglicht Ihnen, mehrere Dinge gleichzeitig zu tun, während normale Programme Kode nur von oben nach unten ausführen können. Bei Threading-Modulen können die LED und der Summer separat arbeiten.

```
def ledWork():
    while flag:
        GPIO.output(ALedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(ALedPin,GPIO.LOW)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        GPIO.output(BLedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(BLedPin,GPIO.LOW)
```

Die Funktion `ledWork()` hilft beim Einstellen des Arbeitszustands dieser beiden LEDs: Sie leuchtet die grüne LED 0,5 Sekunden lang auf und erlischt dann. In ähnlicher Weise leuchtet die rote LED 0,5 Sekunden lang auf und erlischt dann.

```
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)
```

Mit der Funktion `buzzWork()` wird der Arbeitszustand des Summers eingestellt. Hier stellen wir die Frequenz zwischen 130 und 800 ein, um sie in einem Intervall von 20 zu akkumulieren oder abzunehmen.

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

In der Funktion `on()`:

1. Definieren Sie die Markierung „flag = 1“, die das Ende des Kontrollthreads angibt.
2. Starten Sie den Buzz und stellen Sie den Arbeitszyklus auf 50% ein.
3. Erstellen Sie 2 separate Threads, damit die LED und der Summer gleichzeitig arbeiten können.

`tBuzz = threading.Thread(target=buzzerWork)` : Create the thread and its prototype is as follows:

```
class threading.Thread(group=None, target=None, name=None, args=(), kwargs={}, *,
↪ daemon=None)
```

Unter den Konstruktionsmethoden ist der Hauptparameter target. Wir müssen dem Ziel ein aufrufbares Objekt zuweisen (hier sind die Funktionen ledWork und BuzzWork).

Next `start()` wird aufgerufen, um das Thread-Objekt zu starten. Beispiel: `tBuzz.start()` wird verwendet, um den neu installierten tBuzz-Thread zu starten.

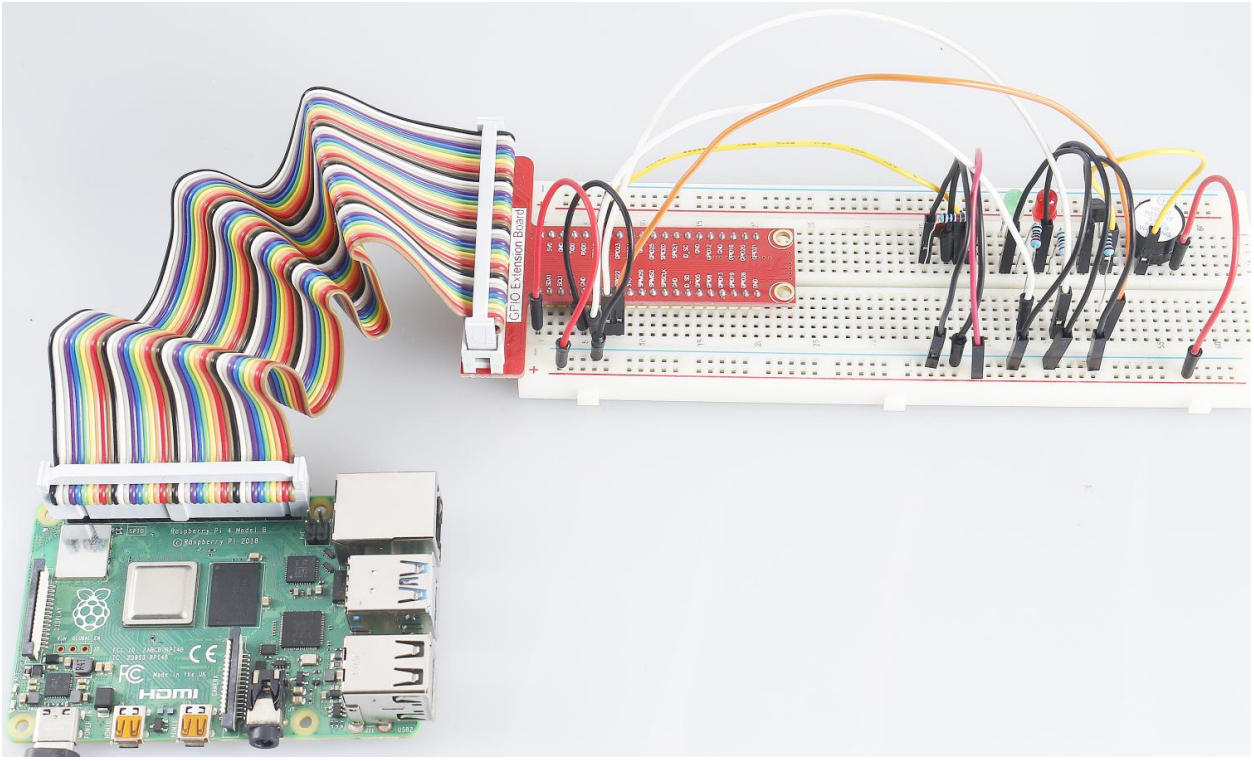
```
def off():
    global flag
    flag = 0
    Buzz.stop()
    GPIO.output(ALedPin,GPIO.LOW)
    GPIO.output(BLedPin,GPIO.LOW)
```

Die Funktion `Off()` definiert „flag=0“, um die Threads ledWork und BuzzWork zu verlassen und dann den Summer und die LED auszuschalten.

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

`main()` enthält den gesamten Prozess des Programms: Lesen Sie zuerst den Wert des Schiebeschalters; Wenn der Kippschalter nach rechts umgeschaltet ist (der Messwert ist 1), wird die Funktion `on()` aufgerufen, der Summer wird zur Ausgabe von Tönen angesteuert und die rote und die grüne LED blinken. Andernfalls funktionieren der Summer und die LED nicht.

Phänomen Bild

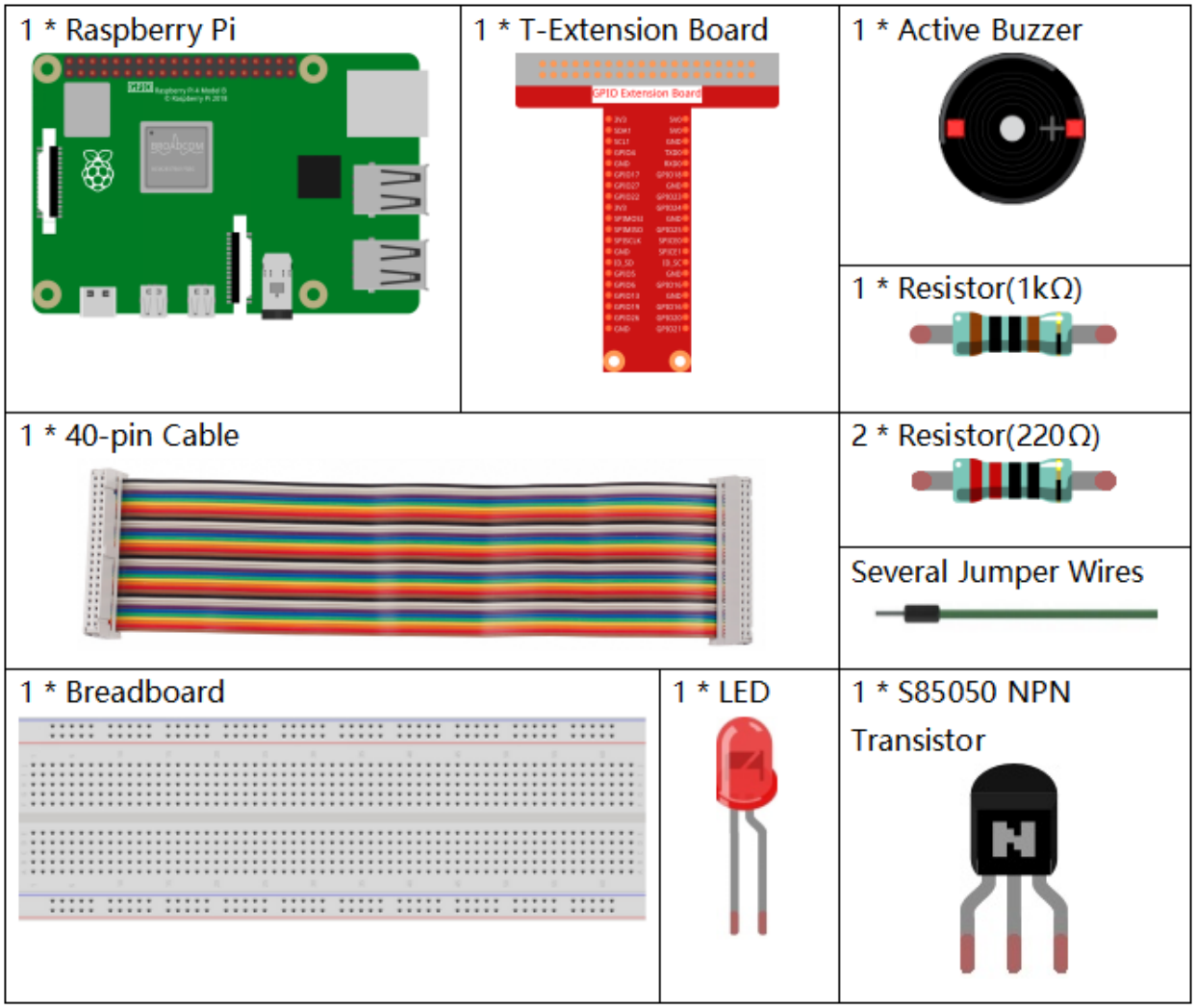


1.9.11 3.1.11 Morsekode-Generator

Einführung

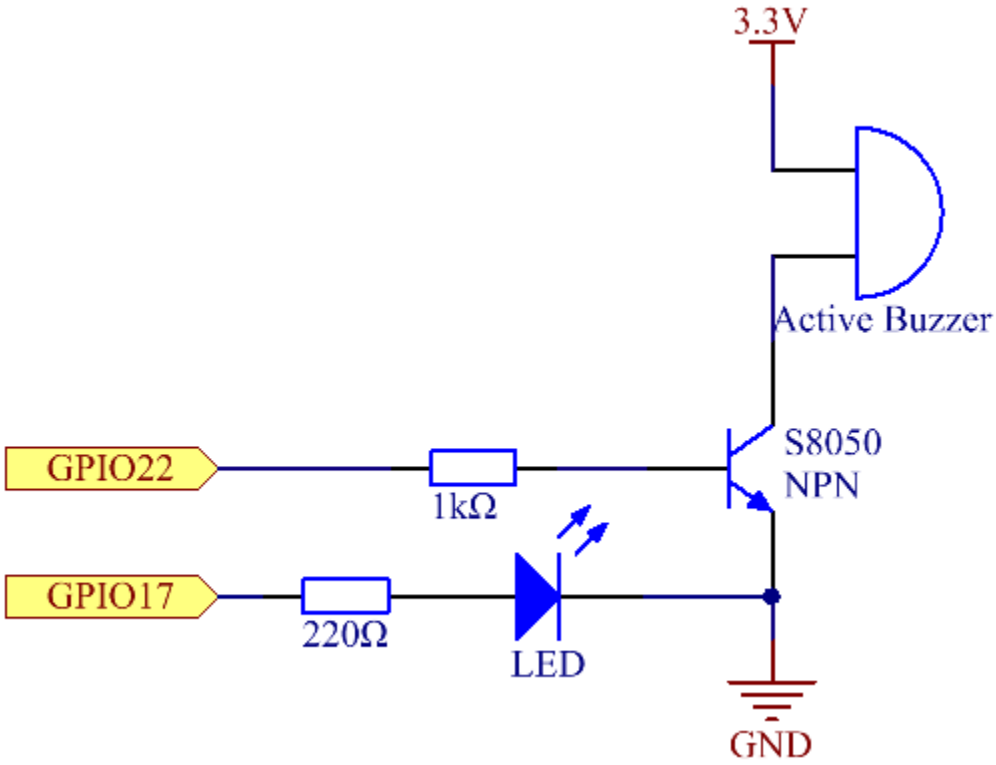
In dieser Lektion erstellen wir einen Morsekode-Generator, in den Sie eine Reihe englischer Buchstaben in den Raspberry Pi eingeben, damit er als Morsekode angezeigt wird.

Komponenten



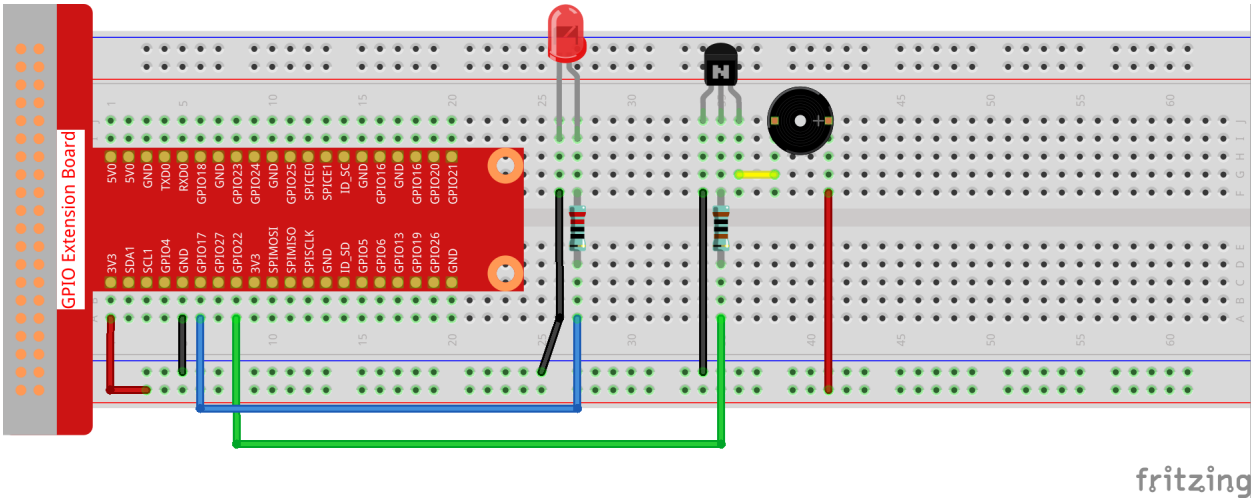
Schematische Darstellung

T-Karte Name	physisch	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO22	Pin 15	3	22



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf. (Achten Sie auf die Pole des Summers: Der mit dem + Etikett ist der positive Pol und der andere der negative.)



Für Benutzer in C-Sprache

Schritt 2: Öffnen Sie die Kodedatei.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.11/
```

Schritt 3: Kompilieren Sie den Kode.

```
gcc 3.1.11_MorseCodeGenerator.c -lwiringPi
```

Schritt 4: Führen Sie die obige ausführbare Datei aus.

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

Geben Sie nach dem Ausführen des Programms eine Reihe von Zeichen ein, und der Summer und die LED senden die entsprechenden Morsekodesignale.

Code Erklärung

```
struct MORSE{
    char word;
    unsigned char *code;
};

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

Diese Struktur MORSE ist das Wörterbuch des Morsecodes und enthält die Zeichen A-Z, die Zahlen 0-9 und die Zeichen „?“ „/“ „:“ “ “ „,“ „.“ „;“ „!“ „@“.

```
char *lookup(char key,struct MORSE *dict,int length)
{
    for (int i=0;i<length;i++)
    {
        if(dict[i].word==key){
            return dict[i].code;
        }
    }
}
```

Die Funktion `lookup()` funktioniert durch „Überprüfen des Wörterbuchs“. Definieren Sie einen Schlüssel, suchen Sie die gleichen Wörter wie den Schlüssel in der Struktur morseDict und geben Sie die entsprechenden Informationen zurück - „kode“ des bestimmten Wortes.

```
void on(){
    digitalWrite(ALedPin,HIGH);
    digitalWrite(BeepPin,HIGH);
}
```

Erstellen Sie eine Funktion `on()` , um den Summer und die LED zu starten.

```
void off(){
    digitalWrite(ALedPin,LOW);
    digitalWrite(BeepPin,LOW);
}
```

Die Funktion `off()` schaltet den Summer und die LED aus.

```
void beep(int dt){
    on();
    delay(dt);
    off();
    delay(dt);
}
```

Definieren Sie einen Funktionston `beep()` , damit der Summer und die LED in einem bestimmten Intervall von `dt` ertönen und blinken.

```
void morsecode(char *code){
    int pause = 250;
    char *point = NULL;
    int length = sizeof(morseDict)/sizeof(morseDict[0]);
    for (int i=0;i<strlen(code);i++)
    {
        point=lookup(code[i],morseDict,length);
        for (int j=0;j<strlen(point);j++){
            if (point[j]=='0')
            {
                beep(pause/2);
            }else if(point[j]=='1')
            {
                beep(pause);
            }
            delay(pause);
        }
    }
}
```

Die Funktion `morsecode()` wird verwendet, um den Morsecode von Eingabezeichen zu verarbeiten, indem die „1“ der Kode weiterhin Töne oder Lichter aussendet und die „0“ in Kürze Töne oder Lichter aussendet, z. B. „SOS“ eingibt und dort wird ein Signal sein, das drei kurze, drei lange und dann drei kurze Segmente · · · — — — · · · enthält.

```
int toupper(int c)
{
    if ((c >= 'a') && (c <= 'z'))
        return c + ('A' - 'a');
    return c;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}
char *strupr(char *str)
{
    char *orign=str;
    for (; *str!='\0'; str++)
        *str = toupper(*str);
return orign;
}
```

Vor dem Codieren müssen Sie die Buchstaben in Großbuchstaben vereinheitlichen.

```
void main(){
    setup();
    char *code;
    int length=8;
    code = (char*)malloc(sizeof(char)*length);
    while (1){
        printf("Please input the messenger:");
        delay(100);
        scanf("%s",code);
        code=strupr(code);
        printf("%s\n",code);
        delay(100);
        morsecode(code);
    }
}
```

Wenn Sie die relevanten Zeichen mit der Tastatur eingeben, konvertiert `code=strupr(code)` die Eingabebuchstaben in ihre Großbuchstaben.

`printf()` druckt dann den Klartext auf dem Computerbildschirm, und die Funktion `morsecod()` bewirkt, dass der Summer und die LED Morsecode ausgeben.

Beachten Sie, die Länge des Eingabezeichens darf die `length` nicht überschreiten(kann überarbeitet werden).

Für Python-Sprachbenutzer

Schritt 2: Öffnen Sie die Kodedatei.

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

Schritt 3: Ausführen.

```
sudo python3 3.1.11_MorseCodeGenerator.py
```

Geben Sie nach dem Ausführen des Programms eine Reihe von Zeichen ein, und der Summer und die LED senden die entsprechenden Morsekodesignale.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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)

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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

Code Erklärung

```
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',
    }
```

Diese Struktur MORSE ist das Wörterbuch des Morsecodes und enthält die Zeichen A-Z, die Zahlen 0-9 und die Zeichen „?“ „/“ „:“ “ “ „,“ „.“ „;“ „!“ „@“.

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

Die Funktion `on()` startet den Summer und die LED.

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

Mit der Funktion `off()` werden der Summer und die LED ausgeschaltet.

```
def beep(dt):   # x for dalay time.
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

Definieren Sie einen Funktionston `beep()` , damit der Summer und die LED in einem bestimmten Intervall von dt ertönen und blinken.

```
def morsecode(code):
    pause = 0.25
    for letter in code:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause/2)
            if tap == '1':
                beep(pause)
        time.sleep(pause)
```

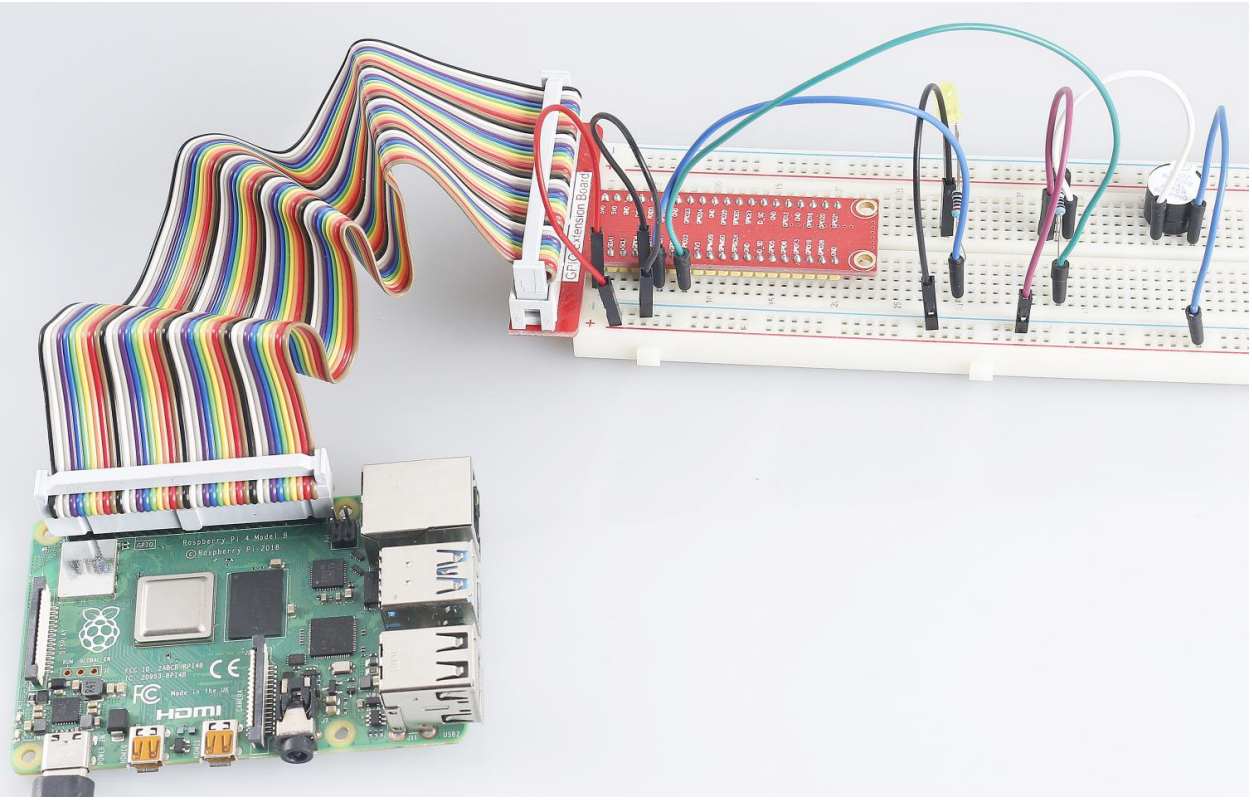
Die Funktion `morsecode()` wird verwendet, um den Morsecode von Eingabezeichen zu verarbeiten, indem die „1“ der Kode weiterhin Töne oder Lichter aussendet und die „0“ in Kürze Töne oder Lichter aussendet, z. B. **SOS** eingibt und dort wird ein Signal sein, das drei kurze, drei lange und dann drei kurze Segmente · · · — — — · · · enthält.

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

Wenn Sie die relevanten Zeichen mit der Tastatur eingeben, konvertiert `upper()` die Eingabebuchstaben in ihre Großbuchstaben.

`print()` druckt dann den Klartext auf dem Computerbildschirm, und die Funktion `morsecode()` bewirkt, dass der Summer und die LED Morsecode ausgeben.

Phänomen Bild

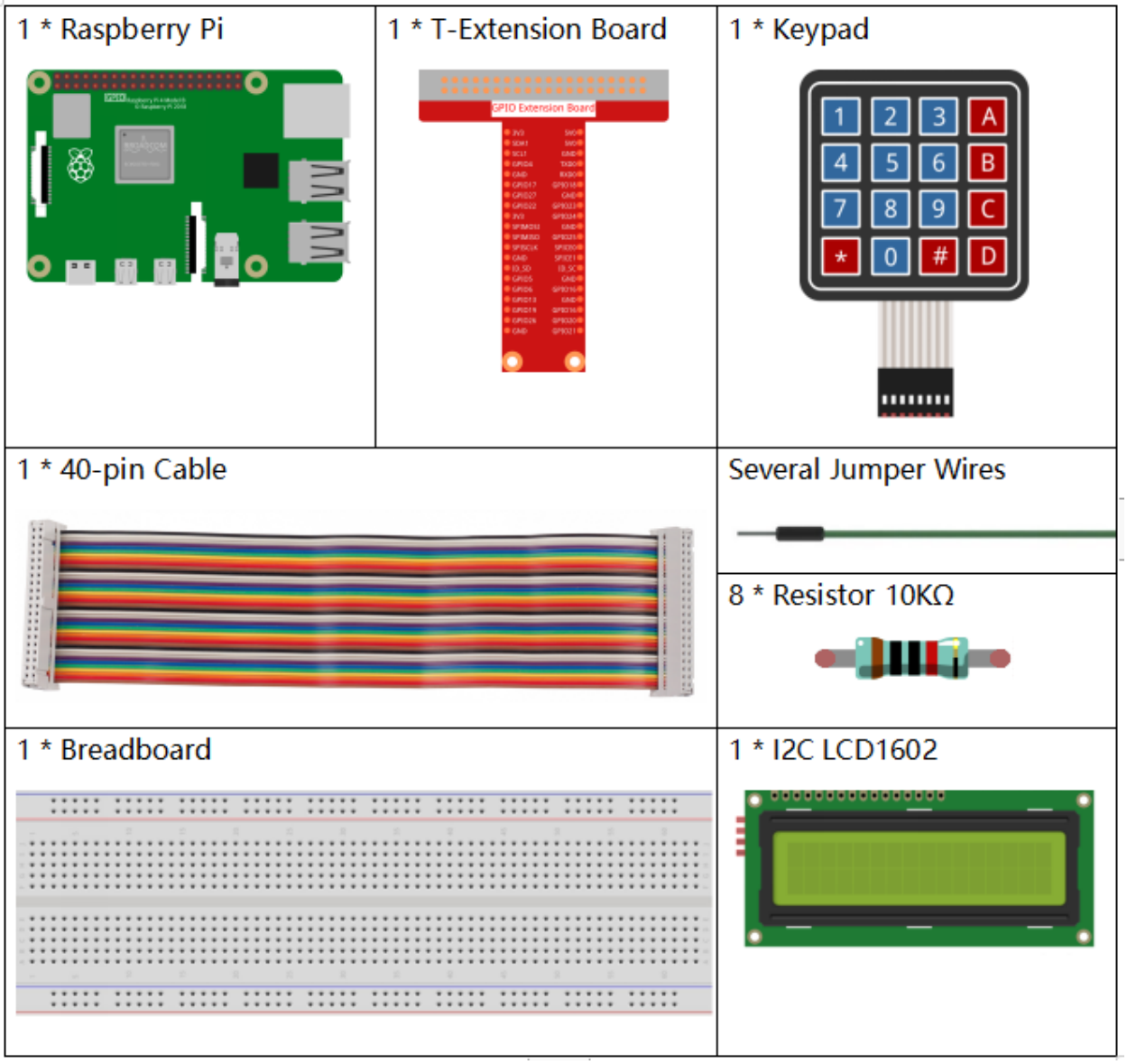


1.9.12 3.1.12 SPIEL – Nummer Vermutung

Einführung

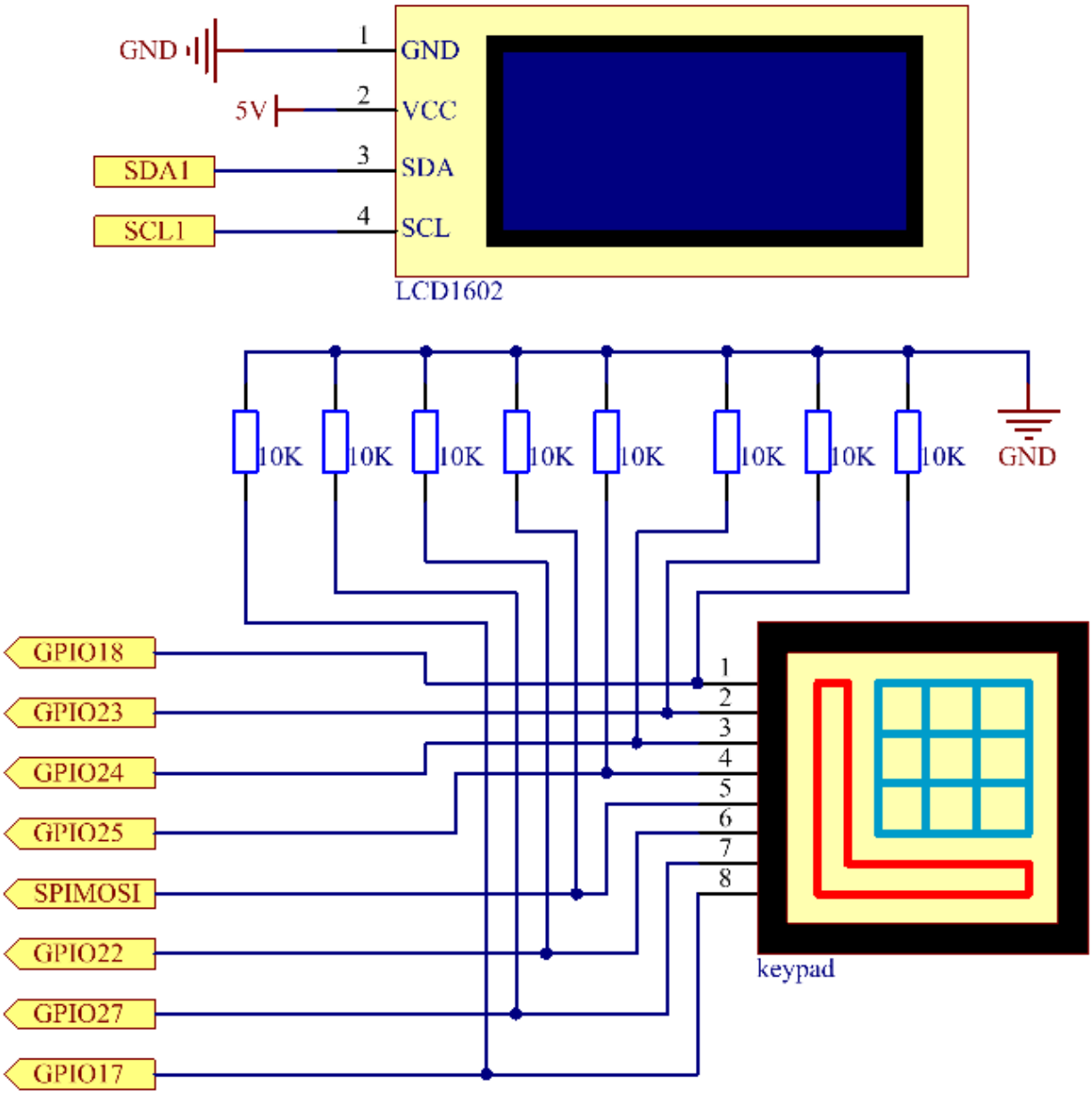
Nummer Vermutung ist ein lustiges Partyspiel, bei dem Sie und Ihre Freunde abwechselnd eine Nummer eingeben (0~99). Die Reichweite wird mit der Eingabe der Nummer kleiner, bis ein Spieler das Rätsel richtig beantwortet. Dann wird der Spieler besiegt und bestraft. Wenn zum Beispiel die Glücksnummer 51 ist, die die Spieler nicht sehen können, und der Spieler 50 eingibt, ändert sich die Eingabeaufforderung des Nummernbereichs auf 50~99; Wenn der Spieler 70 eingibt, kann der Nummerbereich zwischen 50 und 70 liegen. Wenn der Spieler 51 eingibt, ist dieser Spieler der Unglückliche. Hier verwenden wir die Tastatur zur Eingabe von Nummer und das LCD zur Ausgabe der Ergebnisse.

Komponenten



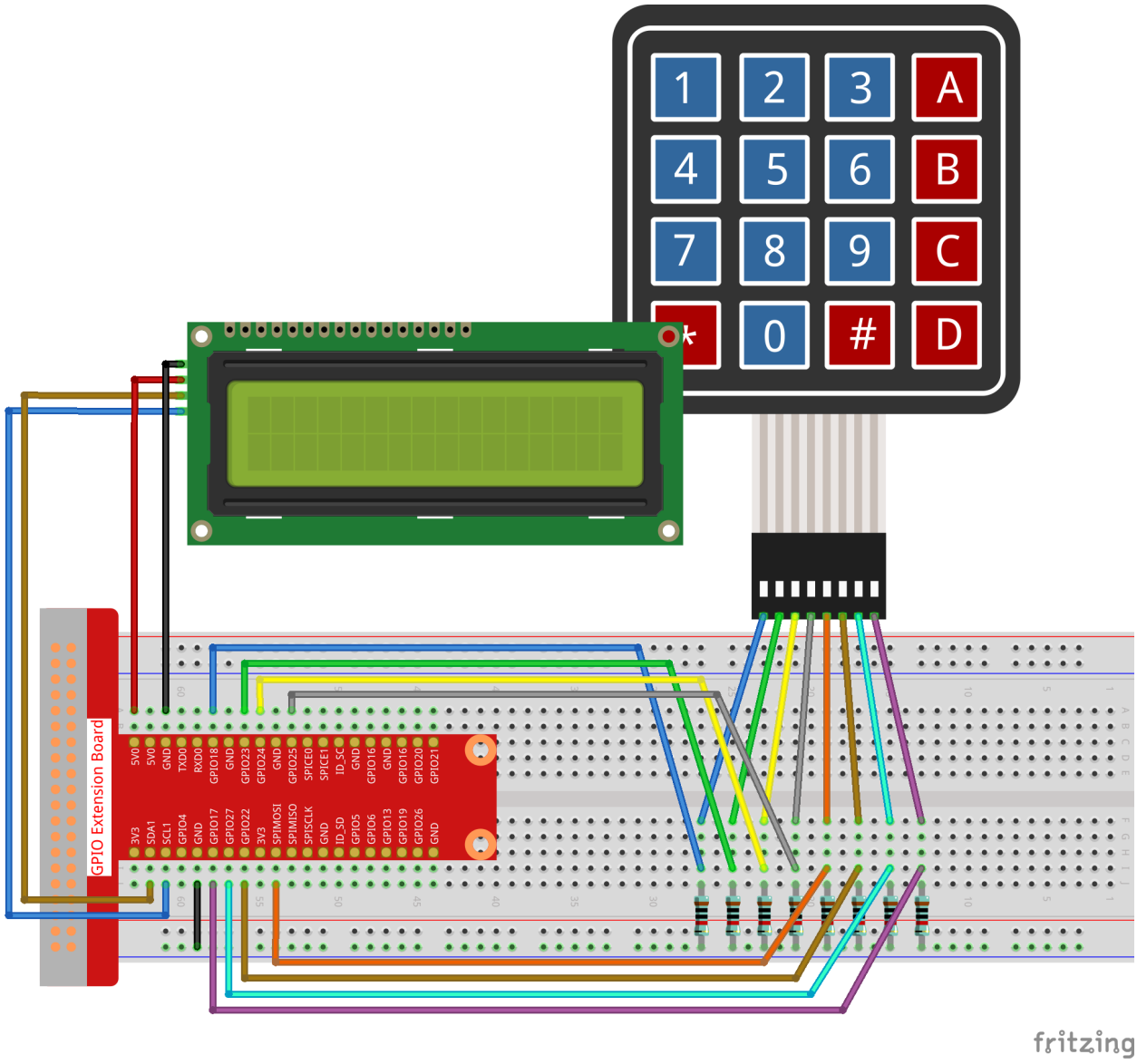
Schematische Darstellung

T-Karte Name	physisch	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)



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: I2C einrichten (siehe Anhang. Wenn Sie *I2C-Konfiguration* eingestellt haben, überspringen Sie diesen Schritt.)

Für Benutzer in C-Sprache

Schritt 3: Verzeichnis wechseln.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.12/
```

Schritt 4: Kompilieren.

```
gcc 3.1.12_GAME_GuessNumber.c -lwiringPi
```

Schritt 5: Ausführen.

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

Nachdem das Programm ausgeführt wurde, wird die erste Seite auf dem LCD angezeigt:

```
Welcome!
Press A to go!
```

Drücken Sie ‚A' und das Spiel startet und die Spieleseite erscheint auf dem LCD.

```
Enter number:
0 ‹point‹ 99
```

Zu Beginn des Spiels wird eine Zufallsnummer ‚Punkt' erzeugt, die jedoch nicht auf dem LCD angezeigt wird. Sie müssen sie nur erraten. Die eingegebene Nummer wird am Ende der ersten Zeile angezeigt, bis die endgültige Berechnung abgeschlossen ist. (Drücken Sie ‚D', um den Vergleich zu starten. Wenn die Eingangsnummer größer als 10 ist, wird der automatische Vergleich gestartet.)

Der Nummernkreis von ‚Punkt' wird in der zweiten Zeile angezeigt. Und Sie müssen die Nummer innerhalb des Bereichs eingeben. Wenn Sie eine Nummer eingeben, wird der Bereich enger. Wenn Sie die Glücksnummer glücklicherweise oder unglücklicherweise erhalten haben, wird „You've got it!" angezeigt.

Code Erklärung

Am Anfang des Codes stehen die Funktionsfunktionen der Tastatur und des I2C LCD1602. Weitere Informationen hierzu finden Sie in den Tasten *1.1.7 I2C LCD1602* und *2.1.5 Tastatur* .

Hier müssen wir Folgendes wissen:

```
/**********************************************/
//Start from here
/**********************************************/
void init(void){
    fd = wiringPiI2CSetup(LCDAddr);
    lcd_init();
    lcd_clear();
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
        pinMode(colPins[i], INPUT);
    }
    lcd_clear();
    write(0, 0, "Welcome!");
    write(0, 1, "Press A to go!");
}
```

Diese Funktion wird verwendet, um zunächst I2C LCD1602 und Tastatur zu definieren und „Welcome!“ und „Press A to go!“ anzuzeigen.

```
void init_new_value(void){
    srand(time(0));
    pointValue = rand()%100;
    upper = 99;
    lower = 0;
    count = 0;
    printf("point is %d\n",pointValue);
}
```

Die Funktion erzeugt die Zufallsnummer `pointValue` und setzt den Bereichshinweis des Punktes zurück.

```
bool detect_point(void){
    if(count > pointValue){
        if(count < upper){
            upper = count;
        }
    }
    else if(count < pointValue){
        if(count > lower){
            lower = count;
        }
    }
    else if(count = pointValue){
        count = 0;
        return 1;
    }
    count = 0;
    return 0;
}
```

`detect_point()` vergleicht die Eingabenummer mit dem erzeugten „point“. Wenn das Ergebnis des Vergleichs ist, dass sie nicht gleich sind, weist `count` `upper` und `lower` Werte zu und gibt „0“ zurück; andernfalls, wenn das Ergebnis anzeigt, dass sie gleich sind, wird „1“ zurückgegeben.

```
void lcd_show_input(bool result){
    char *str=NULL;
    str =(char*)malloc(sizeof(char)*3);
    lcd_clear();
    if (result == 1){
        write(0,1,"You've got it!");
        delay(5000);
        init_new_value();
        lcd_show_input(0);
        return;
    }
    write(0,0,"Enter number:");
    Int2Str(str,count);
    write(13,0,str);
    Int2Str(str,lower);
    write(0,1,str);
    write(3,1,"<Point<");
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    Int2Str(str,upper);
    write(12,1,str);
}
```

Diese Funktion dient zum Anzeigen der Spieleseite. Beachten Sie die Funktion `Int2Str(str,count)` , die diese Variablen `count` , `lower` und `upper` von Integer in Character String konvertiert, um die korrekte Anzeige von lcd zu gewährleisten.

```
int main(){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    init_new_value();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            if(pressed_keys[0] != 0){
                bool result = 0;
                if(pressed_keys[0] == 'A'){
                    init_new_value();
                    lcd_show_input(0);
                }
                else if(pressed_keys[0] == 'D'){
                    result = detect_point();
                    lcd_show_input(result);
                }
                else if(pressed_keys[0] >='0' && pressed_keys[0] <= '9'){
                    count = count * 10;
                    count = count + (pressed_keys[0] - 48);
                    if (count>=10){
                        result = detect_point();
                    }
                    lcd_show_input(result);
                }
            }
            keyCopy(last_key_pressed, pressed_keys);
        }
        delay(100);
    }
    return 0;
}
```

`main()` enthält den gesamten Prozess des Programms, wie unten gezeigt:

1. Initialisieren Sie I2C LCD1602 und Tastatur.
2. Verwenden Sie `init_new_value()` , um eine Zufallsnummer 0-99 zu erstellen.
3. Beurteilen Sie, ob die Taste gedrückt wurde, und lassen Sie die Taste ablesen.

4. Wenn die Taste „A“ gedrückt wird, erscheint eine Zufallsnummer 0-99 und das Spiel beginnt.
5. Wenn festgestellt wird, dass die Taste „D“ gedrückt wurde, geht das Programm in die Ergebnisbeurteilung ein und zeigt das Ergebnis auf dem LCD an. Dieser Schritt hilft Ihnen, das Ergebnis auch zu beurteilen, wenn Sie nur eine Nummer und dann die Taste „D“ drücken.
6. Wenn die Taste 0-9 gedrückt wird, wird der Zählwert geändert. Wenn die Anzahl größer als 10 ist, beginnt das Urteil.
7. Die Änderungen des Spiels und seiner Werte werden auf dem LCD1602 angezeigt.

Für Python-Sprachbenutzer

Schritt 3: Verzeichnis wechseln.

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

Schritt 4: Ausführen.

```
sudo python3 3.1.12_GAME_GuessNumber.py
```

Nachdem das Programm ausgeführt wurde, wird die erste Seite auf dem LCD angezeigt:

```
Welcome!
Press A to go!
```

Drücken Sie ‚A‘ und das Spiel startet und die Spieleseite erscheint auf dem LCD.

```
Enter number:
0 ‹point‹ 99
```

Zu Beginn des Spiels wird eine Zufallsnummer ‚Punkt‘ erzeugt, die jedoch nicht auf dem LCD angezeigt wird. Sie müssen sie nur erraten. Die eingegebene Nummer wird am Ende der ersten Zeile angezeigt, bis die endgültige Berechnung abgeschlossen ist. (Drücken Sie ‚D‘, um den Vergleich zu starten. Wenn die Eingangsnummer größer als 10 ist, wird der automatische Vergleich gestartet.)

Der Nummernkreis von ‚Punkt‘ wird in der zweiten Zeile angezeigt. Und Sie müssen die Nummer innerhalb des Bereichs eingeben. Wenn Sie eine Nummer eingeben, wird der Bereich enger. Wenn Sie die Glücksnummer glücklicherweise oder unglücklicherweise erhalten haben, wird „You’ve got it!“ angezeigt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

```
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():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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 ################

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():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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()
```

Code Erklärung

Am Anfang des Codes stehen die Funktionsfunktionen der Tastatur und des I2C LCD1602. Weitere Informationen hierzu finden Sie in den Tasten *1.1.7 I2C LCD1602* LCD1602 und *2.1.5 Tastatur* .

Hier müssen wir Folgendes wissen:

```
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))
```

Die Funktion erzeugt die Zufallsnummer `pointValue` und setzt den Bereichshinweis des Punktes zurück.

```
def detect_point():
    global count,upper,lower
    if count > pointValue:
        if count < upper:
            upper = count
    elif count < pointValue:
        if count > lower:
            lower = count
    elif count == pointValue:
        count = 0
        return 1
    count = 0
    return 0
```

`detect_point()` vergleicht die eingegebene Zahl (count) mit dem erzeugten „point“. Wenn das Ergebnis des Vergleichs ist, dass sie nicht gleich sind, weist `count` `upper` und `lower` Werte zu und gibt ‚0‘ zurück; andernfalls, wenn das Ergebnis anzeigt, dass sie gleich sind, wird ‚1‘ zurückgegeben.

```
def lcd_show_input(result):
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0,1,'You have got it!')
        time.sleep(5)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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))
```

Diese Funktion dient zum Anzeigen der Spieleseite.

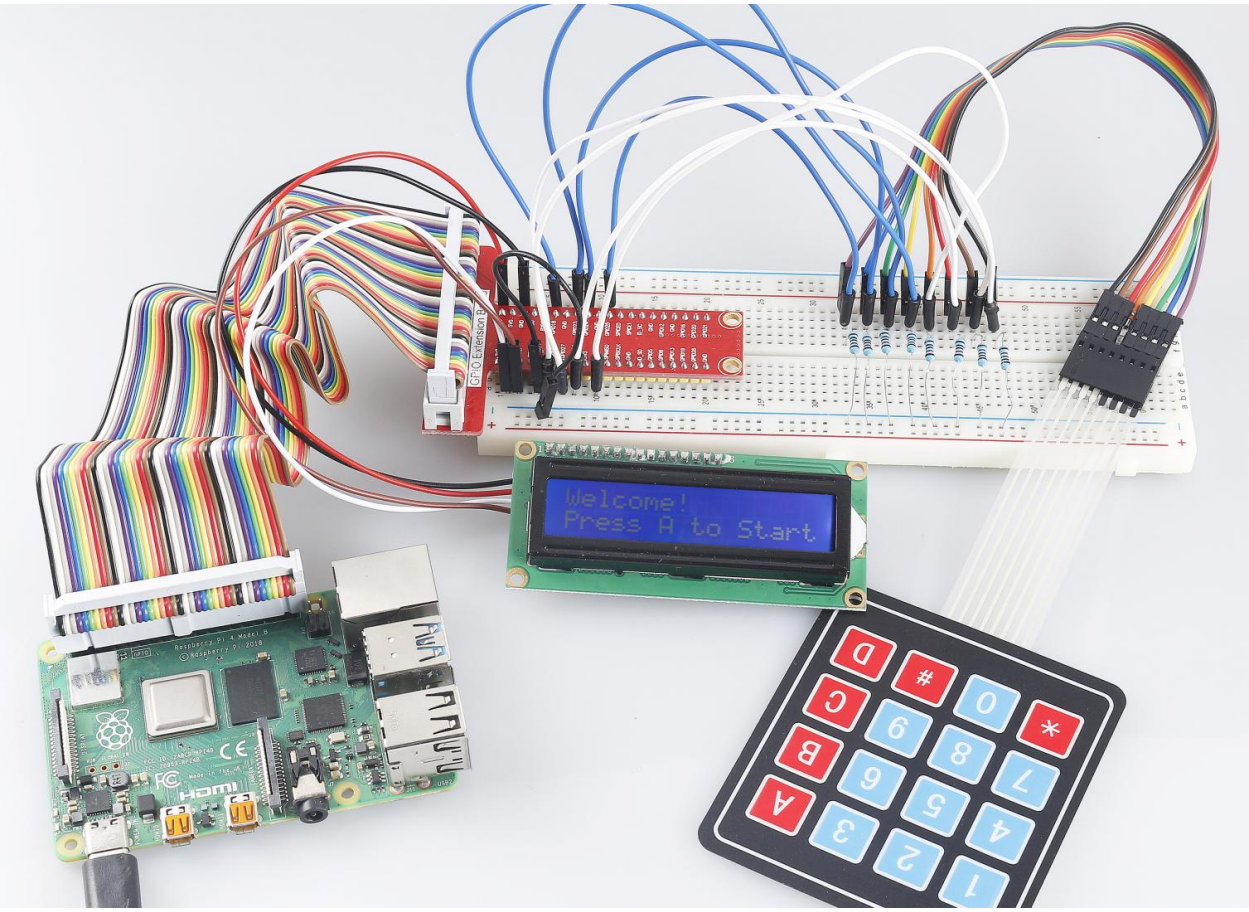
`str(count)` : Da `write()` nur den Datentyp - Zeichenfolge - unterstützen kann, wird `str()` benötigt, um die Nummer in eine Zeichenfolge umzuwandeln.

```
def loop():
    global keypad, last_key_pressed,count
    while(True):
        result = 0
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            if pressed_keys == ["A"]:
                init_new_value()
                lcd_show_input(0)
            elif pressed_keys == ["D"]:
                result = detect_point()
                lcd_show_input(result)
            elif pressed_keys[0] in keys:
                if pressed_keys[0] in list(["A","B","C","D","#","*"]):
                    continue
                count = count * 10
                count += int(pressed_keys[0])
                if count >= 10:
                    result = detect_point()
                lcd_show_input(result)
            print(pressed_keys)
        last_key_pressed = pressed_keys
        time.sleep(0.1)
```

`main()` enthält den gesamten Prozess des Programms, wie unten gezeigt:

1. Initialisieren Sie I2C LCD1602 und Tastatur.
2. Beurteilen Sie, ob die Taste gedrückt wurde, und lassen Sie die Taste ablesen.
3. Wenn die Taste „A“ gedrückt wird, erscheint eine Zufallsnummer 0-99 und das Spiel beginnt.
4. Wenn festgestellt wird, dass die Taste ‚D‘ gedrückt wurde, geht das Programm in die Ergebnisbeurteilung ein.
5. Wenn die Taste 0-9 gedrückt wird, wird der Zählwert geändert. Wenn die Anzahl größer als 10 ist, beginnt das Urteil.
6. Die Änderungen des Spiels und seiner Werte werden auf dem LCD1602 angezeigt.

Phänomen Bild

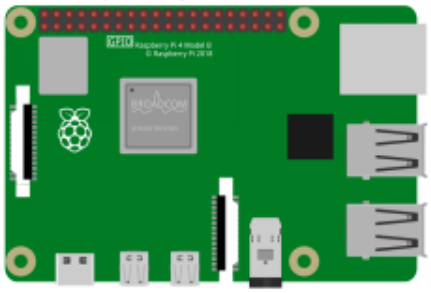
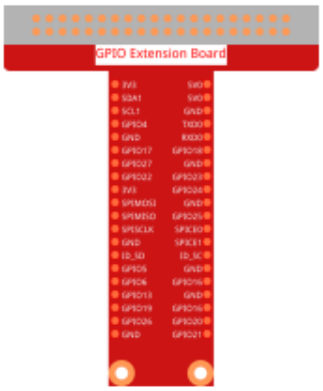
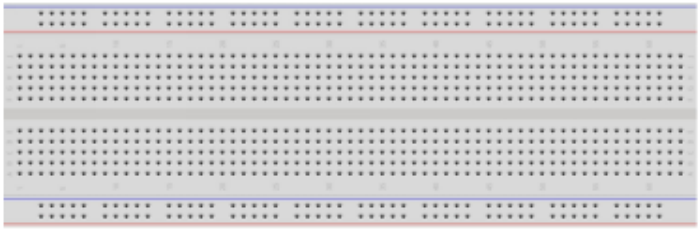


1.9.13 3.1.13 SPIEL - 10 Sekunden

Einführung

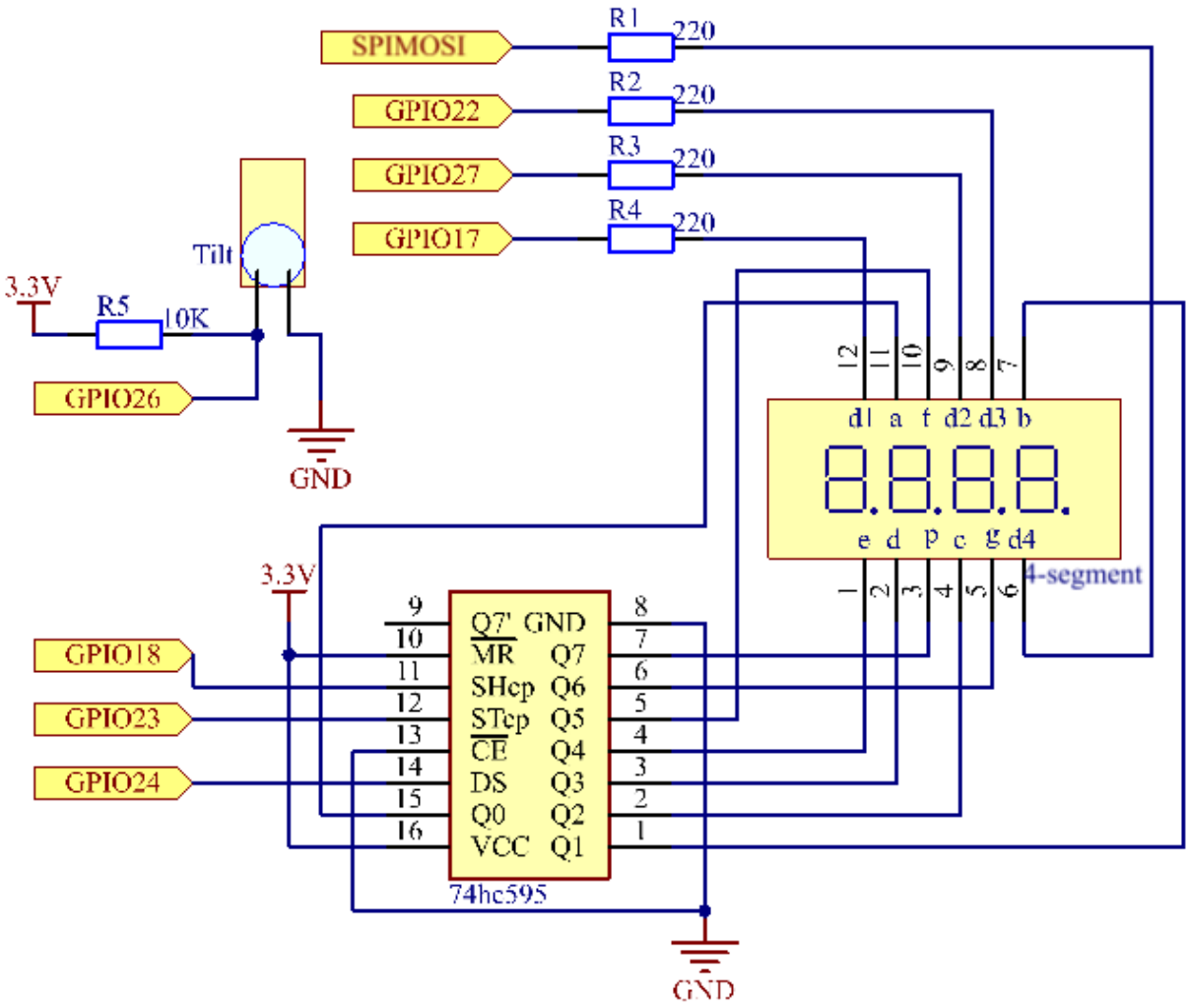
Als nächstes folge mir, um ein Spielgerät zu bauen, das deine Konzentration herausfordert. Binden Sie den Neigungsschalter an einen Stock, um einen Zauberstab herzustellen. Schütteln Sie den Stab, die 4-stellige Segmentanzeige beginnt zu zählen. Durch erneutes Schütteln wird die Zählung beendet. Wenn es Ihnen gelingt, die angezeigte Anzahl bei 10,00 zu halten, gewinnen Sie. Sie können das Spiel mit Ihren Freunden spielen, um zu sehen, wer der Zeitassistent ist.

Komponenten

<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>1 * Breadboard
  </td>
<td colspan="2">Several Jumper Wires
 </td>
</tr>
<tr>
<td></td>
<td>4 * Resistor(220Ω)
 </td>
<td>1 * Resistor 10KΩ
 </td>
</tr>
</table>

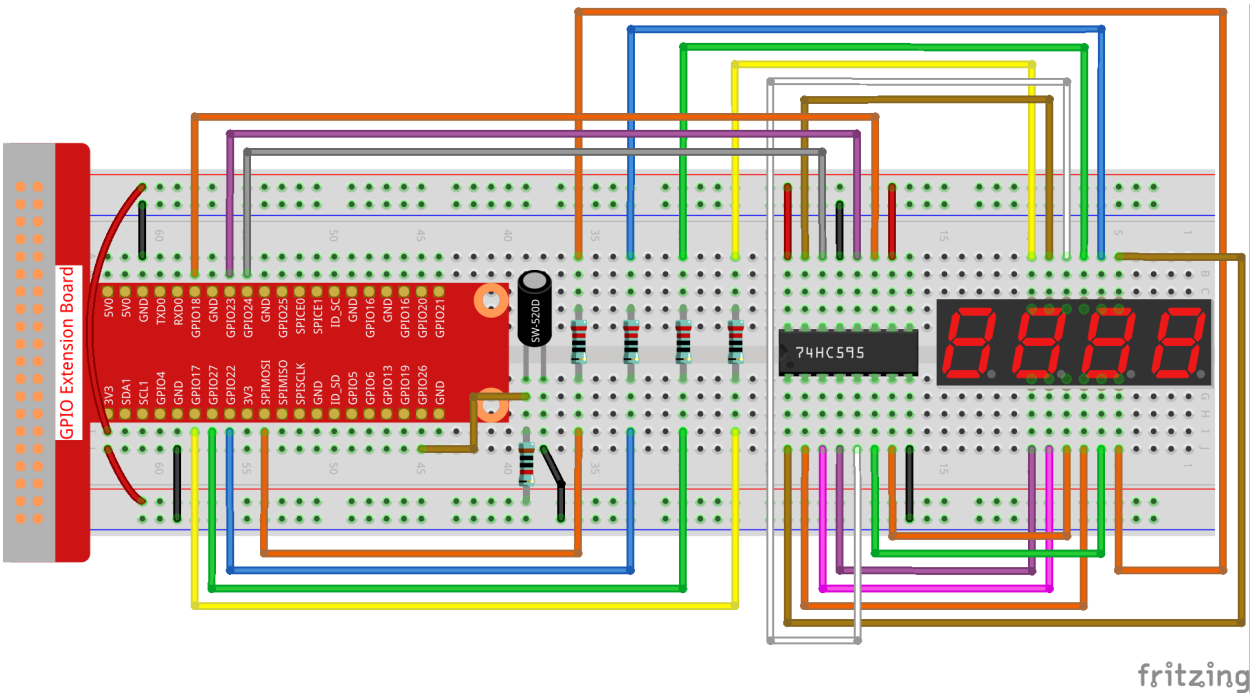
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Ordner der Kode.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.13/
```

Schritt 3: Kompilieren Sie die Kode.

```
gcc 3.1.13_GAME_10Second.c -lwiringPi
```

Schritt 4: Führen Sie die ausführbare Datei aus.

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

Schütteln Sie den Stab, die 4-stellige Segmentanzeige beginnt zu zählen. Durch erneutes Schütteln wird die Zählung beendet. Wenn es Ihnen gelingt, die angezeigte Anzahl bei 10,00 zu halten, gewinnen Sie. Schütteln Sie es noch einmal, um die nächste Runde des Spiels zu starten.

Code Erklärung

```
void stateChange(){
    if (gameState == 0){
        counter = 0;
        delay(1000);
        ualarm(10000,10000);
    }else{
        alarm(0);
        delay(1000);
    }
    gameState = (gameState + 1)%2;
}
```

Das Spiel ist in zwei Modi unterteilt:

`gameState=0` ist der „Start“ -Modus, in dem die Zeit zeitlich festgelegt und auf der Segmentanzeige angezeigt wird und der Kippschalter geschüttelt wird, um in den „Show“ -Modus zu wechseln.

`gameState=1` ist der „show“ -Modus, der das Timing stoppt und die Zeit auf der Segmentanzeige anzeigt. Durch erneutes Schütteln des Neigungsschalters wird der Timer zurückgesetzt und das Spiel neu gestartet.

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            stateChange();
        }
        lastState=currentState;
    }
}
```

`loop()` ist die Hauptfunktion. Zunächst wird die Zeit auf der 4-Bit-Segmentanzeige angezeigt und der Wert des Neigungsschalters gelesen. Wenn sich der Status des Neigungsschalters geändert hat, wird `stateChange()` aufgerufen.

Für Python-Sprachbenutzer

Schritt 2: Gehen Sie zum Ordner der Kode.

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

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.13_GAME_10Second.py
```

Schütteln Sie den Stab, die 4-stellige Segmentanzeige beginnt zu zählen. Durch erneutes Schütteln wird die Zählung beendet. Wenn es Ihnen gelingt, die angezeigte Anzahl bei 10,00 zu halten, gewinnen Sie. Schütteln Sie es noch einmal, um die nächste Runde des Spiels zu starten.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

sensorPin = 26

SDI = 24
RCLK = 23
SRCLK = 18
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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)
        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
    if gameState == 0:
        counter = 0
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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
    timer1.cancel()

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

Code Erklärung

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Das Spiel ist in zwei Modi unterteilt:

`gameState=0` ist der „start“ -Modus, in dem die Zeit zeitlich festgelegt und auf der Segmentanzeige angezeigt wird und der Kippschalter geschüttelt wird, um in den „Show“ -Modus zu wechseln.

`gameState=1` ist der „show“ -Modus, der das Timing stoppt und die Zeit auf der Segmentanzeige anzeigt. Durch erneutes Schütteln des Neigungsschalters wird der Timer zurückgesetzt und das Spiel neu gestartet.

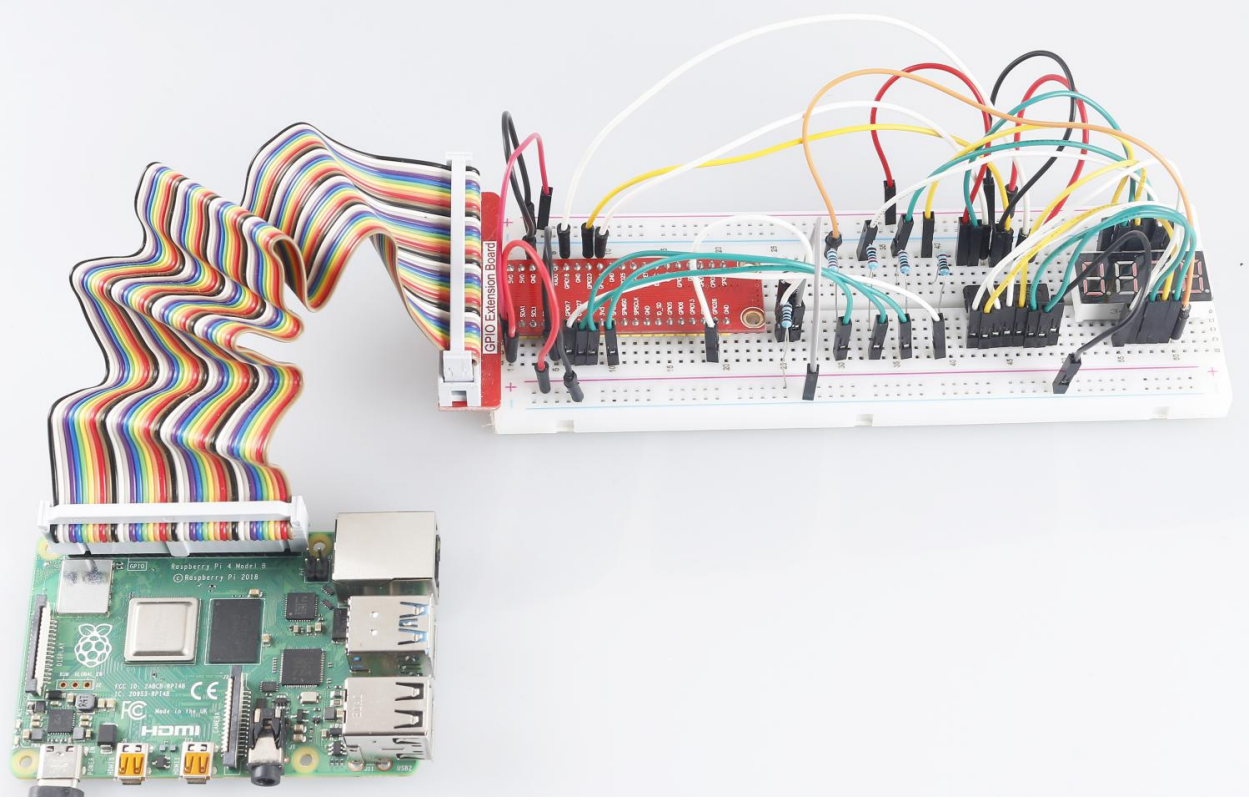
```
def loop():
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState=currentState
```

`loop()` ist die Hauptfunktion. Zunächst wird die Zeit auf der 4-Bit-Segmentanzeige angezeigt und der Wert des Neigungsschalters gelesen. Wenn sich der Status des Neigungsschalters geändert hat, wird `stateChange()` aufgerufen.

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

Nachdem das Intervall 0,01 s erreicht hat, wird die Timerfunktion aufgerufen. Addiere 1 zum Zähler und der Timer wird erneut verwendet, um sich alle 0,01 Sekunden wiederholt auszuführen.

Phänomen Bild



1.9.14 3.1.14 SPIEL - Nicht nicht

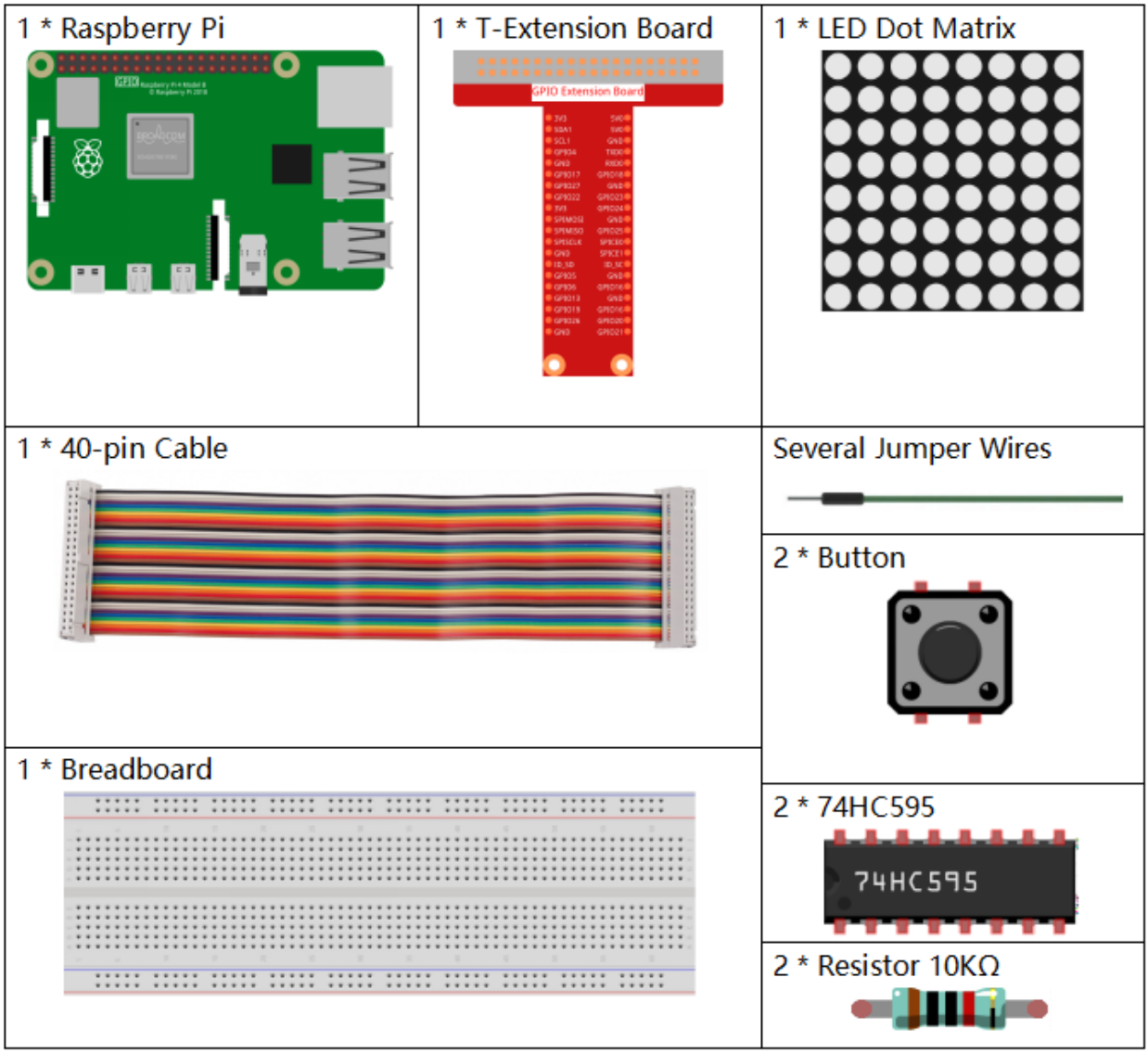
Einführung

In dieser Lektion werden wir ein interessantes Spielgerät herstellen und es „Nicht nicht“ nennen.

Während des Spiels aktualisiert die Punktmatrix einen Pfeil nach dem Zufallsprinzip. Sie müssen die Taste innerhalb einer begrenzten Zeit in die entgegengesetzte Richtung des Pfeils drücken. Wenn die Zeit abgelaufen ist oder wenn die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, sind Sie raus.

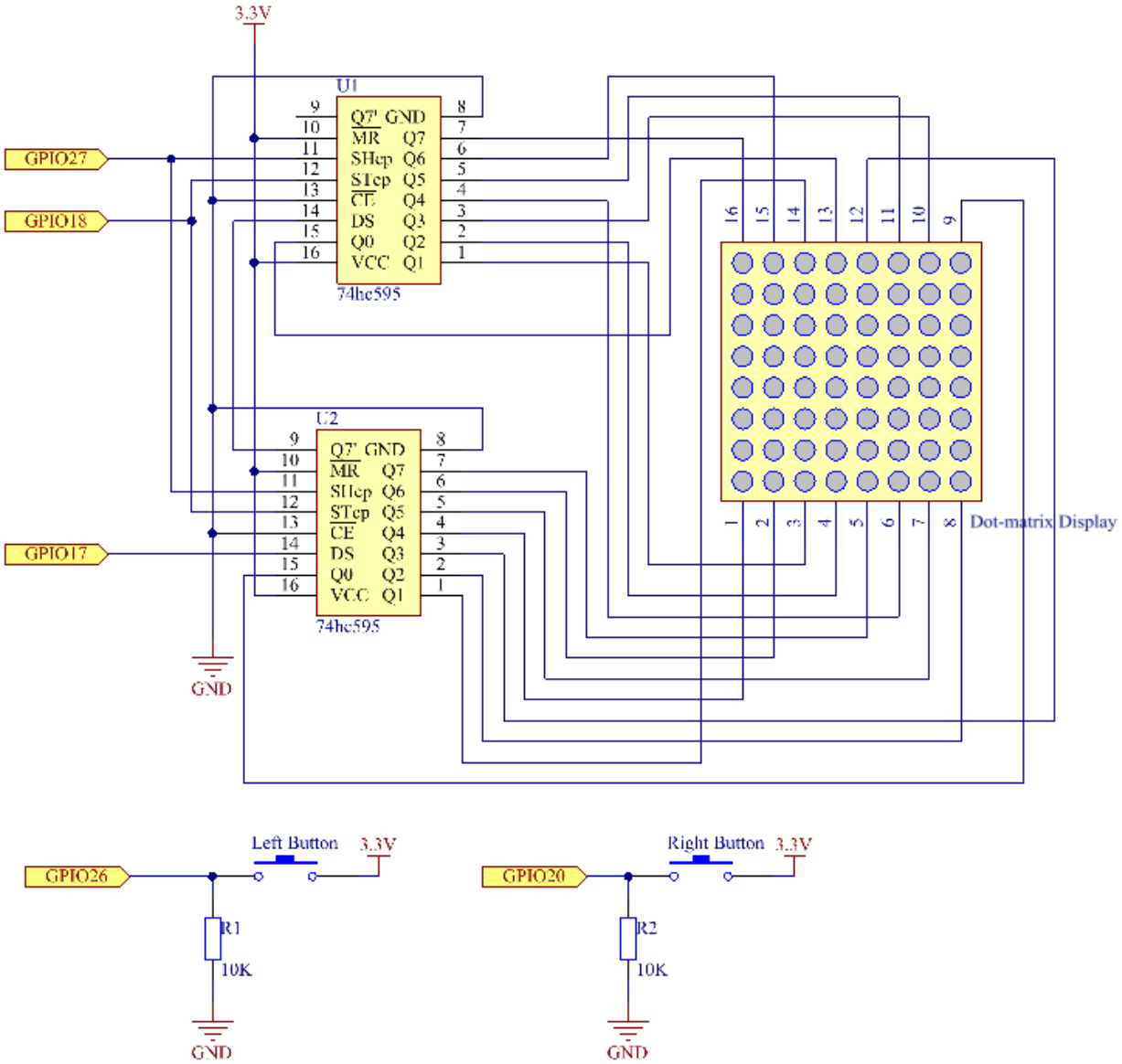
Dieses Spiel kann wirklich Ihr umgekehrtes Denken üben, und jetzt sollen wir es versuchen?

Komponenten



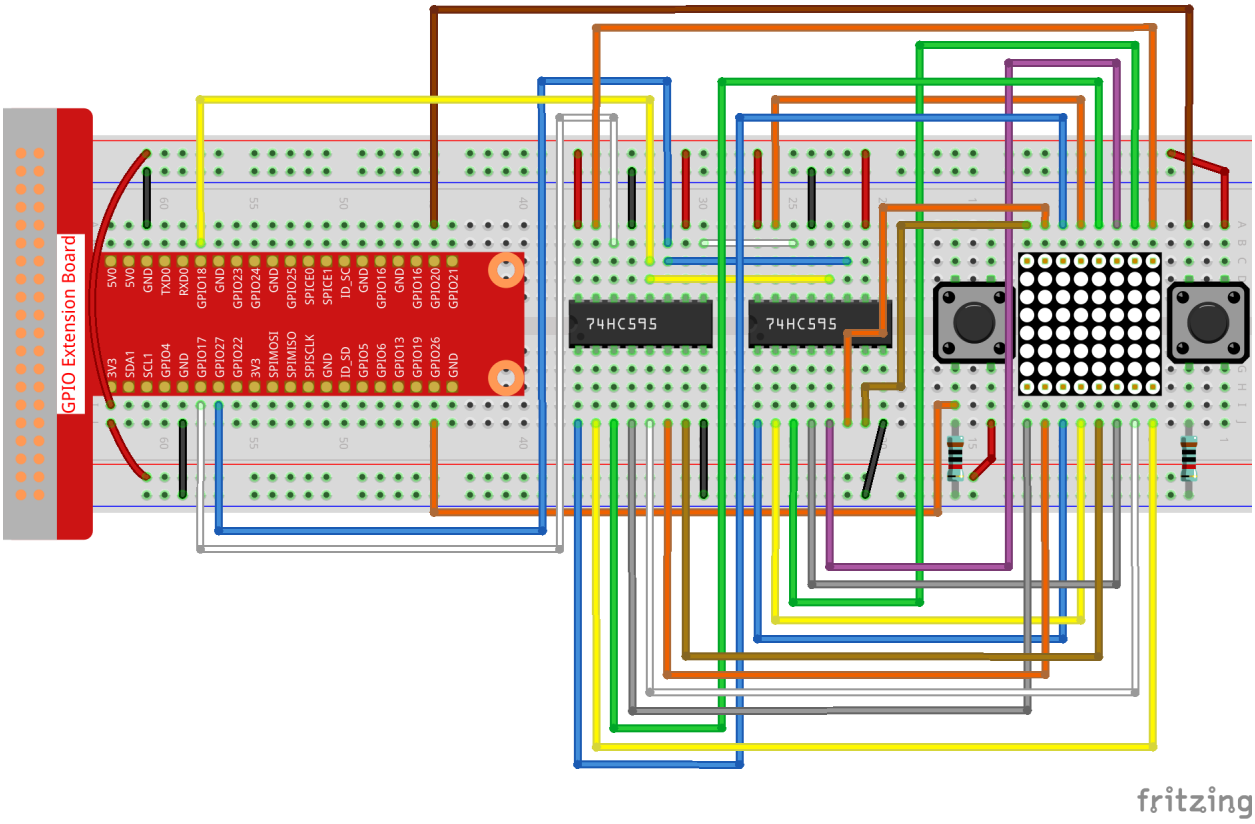
Schematische Darstellung

T-Karte Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie die Schaltung auf.



Für Benutzer in C-Sprache

Schritt 2: Gehen Sie zum Code-Ordner.

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

Schritt 3: Kompilieren.

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

Schritt 4: Ausführen.

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

Nach dem Start des Programms wird ein Pfeil nach links oder rechts in der Punktmatrix nach dem Zufallsprinzip aktualisiert. Sie müssen die Taste innerhalb einer begrenzten Zeit in die entgegengesetzte Richtung des Pfeils drücken. Dann erscheint „“ auf der Punktmatrix. Wenn die Zeit abgelaufen ist oder wenn die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, sind Sie ausgeschaltet und die Punktmatrix zeigt „x“ an. Sie können auch 2 neue Schaltflächen hinzufügen oder durch Joystick-Tasten für Auf, Ab, Links und Rechts ersetzen - 4 Richtungen, um die Schwierigkeit des Spiels zu erhöhen.

Code Erklärung

Basierend auf 1.1.6 LED Dot Matrix fügt diese Lektion 2 Tasten hinzu, um ein amüsantes Spielgerät zu erstellen. Wenn Sie mit der Punktmatrix nicht sehr vertraut sind, lesen Sie bitte *1.1.6 LED-Punktmatrix*.

Der gesamte Programmprozess ist wie folgt:

1. Wählen Sie zufällig eine Pfeilrichtung und generieren Sie Timer 1.

2. Zeigen Sie das Pfeilbild auf der Punktmatrix an.
3. Beurteilen Sie die Tasteneingabe. Wenn die Taste gedrückt wird oder Timer 1 daran erinnert, dass die Zeit abgelaufen ist, beginnt die Beurteilung.
4. Zeigen Sie das Bild anhand eines Bewertungsergebnisses an. In der Zwischenzeit Timer 2 generieren.
5. Führen Sie Schritt 1 erneut aus, wenn Timer 2 daran erinnert, dass die Zeit abgelaufen ist.

```
struct GLYPH{
    char *word;
    unsigned char code[8];
};

struct GLYPH arrow[2]=
{
    {"right",{0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF}},
    // {"down",{0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF}},
    // {"up",{0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF}},
    {"left",{0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF}}
};

struct GLYPH check[2]=
{
    {"wrong",{0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF}},
    {"right",{0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF}}
};
```

Die GLYPH-Struktur funktioniert wie ein Wörterbuch: Das Wort Attribut entspricht dem Schlüssel im Wörterbuch. Das Kode-Attribut entspricht dem Wert.

Hier wird Kode verwendet, um ein Array für die Punktmatrix zum Anzeigen von Bildern zu speichern (ein 8x8-Bit-Array).

Hier kann der Array-Pfeil verwendet werden, um das Pfeilmuster in Aufwärts-, Abwärts-, Links- und Rechtsrichtung auf der LED-Punktmatrix anzuzeigen.

Jetzt werden unten und oben kommentiert und bei Bedarf auskommentiert.

Die Array-Prüfung wird verwendet, um diese beiden Bilder anzuzeigen: „×“ und „“.

```
char *lookup(char *key,struct GLYPH *glyph,int length){
    for (int i=0;i<length;i++)
    {
        if(strcmp(glyph[i].word,key)==0){
            return glyph[i].code;
        }
    }
}
```

Die Funktion `lookup()` funktioniert durch „Überprüfen des Wörterbuchs“. Definieren Sie einen `key` , suchen Sie die gleichen Wörter wie der `key` in der Struktur `GLYPH * glyph` und geben Sie die entsprechenden Informationen zurück - „kode“ des bestimmten Wortes.

Die Funktion `strcmp()` wird verwendet, um die Identität von zwei Zeichenfolgen `glyph[i].word` und `key` zu vergleichen; Wenn die Identität beurteilt wird, geben Sie den `glyph[i].code` zurück (wie gezeigt).

```
void display(char *glyphCode){
    for(int i;i<8;i++){
        hc595_in(glyphCode[i]);
        hc595_in(0x80>>i);
        hc595_out();
    }
}
```

Zeigen Sie das angegebene Muster in der Punktmatrix an.

```
void createGlyph(){
    srand(time(NULL));
    int i=rand()%(sizeof(arrow)/sizeof(arrow[0]));
    waypoint=arrow[i].word;
    stage="PLAY";
    alarm(2);
}
```

Mit der Funktion `createGlyph()` wird zufällig eine Richtung ausgewählt (das Wortattribut eines Elements im array arrow[]: „left“, „right“…). Stellen Sie die Bühne auf „PLAY“ und starten Sie eine 2-Sekunden-Weckerfunktion.

`srand(time(NULL))` : Initialisiert zufällige Seeds, die von der Systemuhr stammen.

`(sizeof(arrow)/sizeof(arrow[0]))` : Ermittelt die Länge des Arrays, das Ergebnis ist 2.

`rand()%2` : Der Rest ist 0 oder 1, erhalten durch Teilen einer generierten Zufallsnummer durch 2.

`waypoint=arrow[i].word` : Das Ergebnis sollte `right` oder `left` sein.

```
void checkPoint(char *inputKey){
    alarm(0)==0;
    if(inputKey==waypoint||inputKey=="empty")
    {
        waypoint="wrong";
    }
    else{
        waypoint="right";
    }
    stage="CHECK";
    alarm(1);
}
```

Mit `checkPoint()` wird die Tasteneingabe überprüft. Wenn die Taste nicht gedrückt wird oder die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, ist das Ergebnis des Wegpunkts falsch und auf der Punktmatrix wird „x“ angezeigt. Andernfalls ist der Wegpunkt richtig und die Punktmatrix zeigt „“ an. Hier ist die Stufe CHECK und es kann eine 1-Sekunden-Weckerfunktion eingestellt werden.

`alarm()` wird auch als „Wecker“ bezeichnet, bei dem ein Timer eingestellt werden kann, und sendet SIGALRM-Signale an den Fortschritt, wenn die definierte Zeit abgelaufen ist.

```
void getKey(){
    if (digitalRead(AButtonPin)==1&&digitalRead(BButtonPin)==0)
    {checkPoint("right");}
    else if (digitalRead(AButtonPin)==0&&digitalRead(BButtonPin)==1)
    {checkPoint("left");}
}
```

getKey() liest die Zustände dieser beiden Schaltflächen; Wenn die rechte Taste gedrückt wird, ist der Parameter der Funktion checkPoint() rechts und wenn die linke Taste gedrückt wird, bleibt der Parameter links.

```
void timer(){
    if (stage=="PLAY"){
        checkPoint("empty");
    }
    else if(stage=="CHECK"){
        createGlyph();
    }
}
```

Zuvor wurde timer() aufgerufen, wenn die Alarmzeit abgelaufen ist. Im Modus „PLAY“ soll dann checkPoint() aufgerufen werden, um das Ergebnis zu beurteilen. Wenn das Programm auf den Modus „CHECK“ eingestellt ist, sollte die Funktion createGlyph() aufgerufen werden, um neue Muster auszuwählen.

Zuvor wurde timer() aufgerufen, wenn die Alarmzeit abgelaufen ist. Im Modus „PLAY“ soll dann checkPoint() aufgerufen werden, um das Ergebnis zu beurteilen. Wenn das Programm auf den Modus „CHECK“ eingestellt ist, sollte die Funktion createGlyph() aufgerufen werden, um neue Muster auszuwählen.

```
void main(){
    setup();
    signal(SIGALRM,timer);
    createGlyph();
    char *code = NULL;
    while(1){
        if (stage == "PLAY")
        {
            code=lookup(waypoint,arrow,sizeof(arrow)/sizeof(arrow[0]));
            display(code);
            getKey();
        }
        else if(stage == "CHECK")
        {
            code = lookup(waypoint,check,sizeof(check)/sizeof(check[0]));
            display(code);
        }
    }
}
```

Die Funktionsweise des Funktions signal(SIGALRM,timer) : Aufruf der Funktion timer(), wenn ein SIGALRM -Signal (vom Weckerfunktions alarm() erzeugt) empfangen wird.

Wenn das Programm startet, rufen Sie zunächst einmal createGlyph() auf und starten Sie dann die Schleife.

In der Schleife: Im PLAY-Modus zeigt die Punktmatrix Pfeilmuster an und überprüft den Schaltflächenstatus. Im CHECK-Modus wird „x“ oder „“ angezeigt.

Für Python-Sprachbenutzer

Schritt 2: Rufen Sie den Code-Ordner auf.

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

Schritt 3: Ausführen.

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

Nach dem Starten des Programms erscheint auf der Punktmatrix ein Pfeil nach rechts oder links. Sie müssen die Taste innerhalb einer begrenzten Zeit in die entgegengesetzte Richtung des Pfeils drücken. Dann erscheint „“ auf der Punktmatrix. Wenn die Zeit abgelaufen ist oder wenn die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, sind Sie ausgeschaltet und die Punktmatrix zeigt „x“ an. Sie können auch 2 neue Schaltflächen hinzufügen oder durch Joystick-Tasten für Auf, Ab, Links und Rechts ersetzen - 4 Richtungen, um die Schwierigkeit des Spiels zu erhöhen.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen** . Zuvor müssen Sie jedoch zu einem Quellcodepfad wie `davinci-kit-for-raspberry-pi/python` gehen.

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

SDI   = 17
RCLK  = 18
SRCLK = 27

timerPlay = 0
timerCheck = 0

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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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)
        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:
```

(Fortsetzung auf der nächsten Seite)

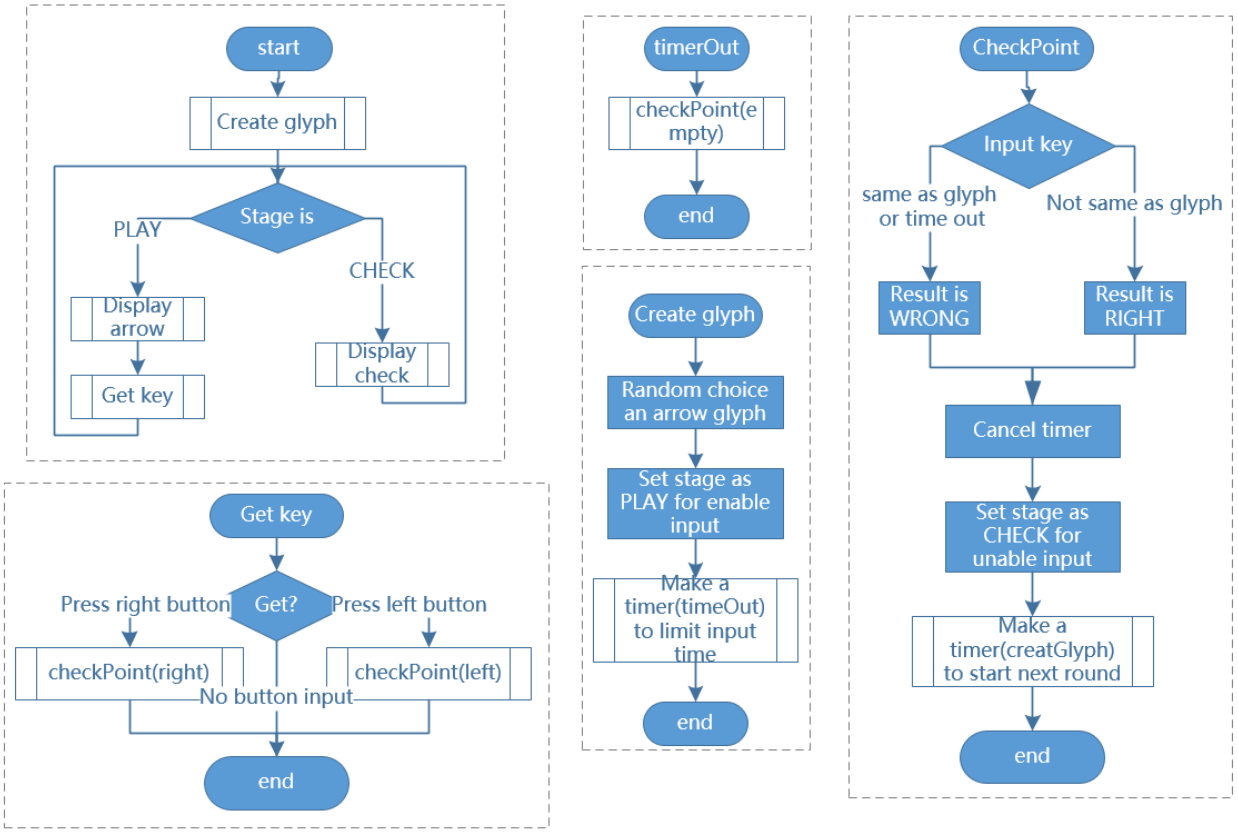
(Fortsetzung der vorherigen Seite)

```
        checkPoint("left")

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

Code Erklärung

Basierend auf 1.1.6 LED Dot Matrix fügt diese Lektion 2 Tasten hinzu, um ein amüsantes Spielgerät zu erstellen. Wenn Sie mit der Punktmatrix nicht sehr vertraut sind, lesen Sie bitte *1.1.6 LED-Punktmatrix* .

Der gesamte Programmprozess ist wie folgt:



1. Wählen Sie zufällig eine Pfeilrichtung und generieren Sie Timer 1.
2. Zeigen Sie das entsprechende Pfeilbild in der Punktmatrix an.
3. Beurteilen Sie die Tasteneingabe. Wenn die Taste gedrückt wird oder Timer 1 daran erinnert, dass die Zeit abgelaufen ist, beginnt die Beurteilung.
4. Zeigen Sie das Bild anhand eines Bewertungsergebnisses an. In der Zwischenzeit Timer 2 generieren.
5. Führen Sie Schritt 1 erneut aus, wenn Timer 2 daran erinnert, dass die Zeit abgelaufen ist.

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

`main()` contains the whole running process.

Wenn das Programm startet, rufen Sie zunächst einmal `creatGlyph()` auf und starten Sie dann die Schleife.

In der Schleife: Im PLAY-Modus zeigt die Punktmatrix Pfeilmuster an und überprüft den Schaltflächenstatus. Im CHECK-Modus wird „x“ oder „“ angezeigt.

```
arrow={
    #"down" :[0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF],
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    #"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]
}
```

Hier kann der Wörterbuch-Pfeil verwendet werden, um das Pfeilmuster nach oben, unten, links und rechts auf der LED-Punktmatrix anzuzeigen.

Jetzt werden unten und oben kommentiert und bei Bedarf auskommentiert.

Die Wörterbuch Prüfung wird verwendet, um diese beiden Bilder anzuzeigen: „×“ und „“.

```
def display(glyphCode):
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80>>i)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)
```

Zeigen Sie das angegebene Muster in der Punktmatrix an.

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

Mit der Funktion **createGlyph()** wird zufällig eine Richtung ausgewählt (das Wortattribut eines Elements im array **arrow[]:** „**left**“, „**right**“…). Stellen Sie die Bühne auf „PLAY“ und starten Sie eine 2-Sekunden-Weckerfunktion.

Mit der Funktion `creatGlyph()` wird zufällig eine Richtung ausgewählt (das Wortattribut eines Elements im array `arrow[]` : `left` , `right` …). Stellen Sie die Bühne auf „PLAY“ und starten Sie eine 2-Sekunden-Weckerfunktion.

`arrow.keys()` : Wählen Sie die Tasten `right` und `left` im Pfeilarray.

`list(arrow.keys())` : Kombinieren Sie diese Schlüssel zu einem Array.

`random.choice(list(arrow.keys()))` : Wählen Sie zufällig ein Element im Array aus.

Das Ergebnis von `waypoint=random.choice(list(arrow.keys()))` sollte also `right` oder `left` sein.

```
def checkPoint(inputKey):
    global waypoint
    global stage
    global timerCheck
    if inputKey == "empty" or inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()
```

Mit `checkPoint()` wird der aktuelle Status der Tasteneingabe ermittelt:

Wenn keine Taste gedrückt wird oder die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, ist der zugewiesene Wert des `waypoint` `wrong` und zeigt `x` auf der Punktmatrix.

Andernfalls ist der Wegpunkt richtig und „“ wird angezeigt.

Jetzt ist die Stufe `CHECK` und startet einen 1-Sekunden-Timer `timerCheck` , um die Funktion `creatGlyph()` in einer Sekunde aufzurufen.

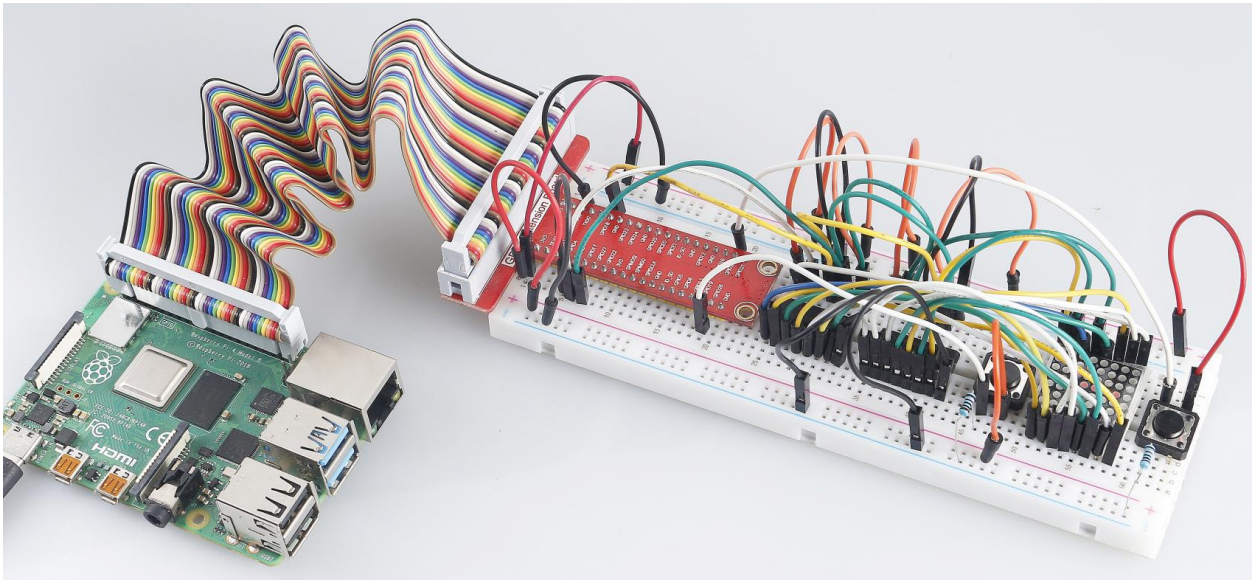
```
def timeOut():
    checkPoint("empty")
```

Setzen Sie im Funktions `timeout()` , den Parameter von `checkPoint()` auf "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")
```

`getKey()` liest den Status dieser beiden Schaltflächen, und wenn die rechte Schaltfläche gedrückt wird, ist der Parameter von `checkPoint()` **right** ; Wenn die **left** Taste gedrückt wird, bleibt der Parameter übrig.

Phänomen Bild



1.10 Python Videokurs (nicht für Pi 5)

Tauchen Sie mit unserem prägnanten Python Videokurs in das Universum des Raspberry Pi ein, ideal für Anfänger und Hobbyisten. Beginnend mit einer Einführung in die Fähigkeiten des Raspberry Pi, führt Sie diese Serie durch die Grundlagen der Python-Programmierung, einschließlich der Steuerung von GPIO-Pins und der Integration von Sensoren. Machen Sie Fortschritte durch interaktive Projekte und erkunden Sie Schlüsselkomponenten wie LEDs und LCD-Displays. Dieser Kurs vermittelt Ihnen essenzielle Fähigkeiten für Programmierung und Hardware-Schnittstellen, indem komplexe Konzepte durch praktische Ausführung zugänglich gemacht werden. Schließen Sie sich uns an, um das Potenzial des Raspberry Pi in der digitalen Welt zu erschließen.

Bemerkung: Diese Videos sind Teil eines anderen Kits von uns, das Raphael Kit genannt wird. Einige Inhalte können jedoch auch in diesem Suite verwendet werden, was zu Unterbrechungen in bestimmten Lektionsnummern führt.

Projekte

1.10.1 Video 1: Erster Start

In diesem Einführungsvideo wird eine umfassende Raspberry Pi-Tutorialreihe vorgestellt, die von der Basisinstallation bis hin zu fortgeschrittenen Projekten reicht. Es betont die Bedeutung eines tiefgehenden Lernprozesses mit Fokus auf Schaltkreisen, Sensoren und GPIO-Pins. Gesponsert von SunFounder, bietet es ein umfangreiches und preiswertes Raspberry Pi-Kit an. Zukünftige Lektionen decken die Einrichtung des Raspberry Pi, Grundlagen von Linux, Python und die Nutzung von GPIO-Pins ab. Außerdem werden die erforderliche Hardware sowie Anpassungen der Raspberry Pi-Desktopoberfläche und Tipps zur Problembehebung besprochen.

1. Einführung in die Raspberry Pi-Tutorialreihe.
2. Fokus auf tiefgehendem statt projektbasiertem Lernen.
3. Schwerpunkt auf Schaltkreisen, Sensoren und GPIO-Pins.
4. Umfangreiches und preiswertes Raspberry Pi-Kit von SunFounder.
5. Vorschau auf kommende Lektionen und erforderliche Hardware.
6. Anpassungen der Desktopoberfläche und Problembehebungstipps.

Video

1.10.2 Video 2: Linux- und Kommandozeilen-Basics mit dem Raspberry Pi

Dieses Tutorial führt kurz in die Linux-Grundlagen und Kommandozeilen-Basics auf dem Raspberry Pi ein. Es behandelt Themen wie Dateisystemnavigation, Dateiverwaltung, Softwareinstallation, Nutzung des Superuser-Modus, Dateimanipulation, Pfadstrukturen, und die Anwendung von Farbcodes für Dateitypen im Terminal. Zuschauer lernen essentielle Befehle und Praktiken für effizientes Arbeiten mit dem Raspberry Pi.

1. **Dateisystem-Navigation**: Nutzung von Befehlen wie cd, ls, pwd.
2. **Dateiverwaltung**: Erstellen, Umbenennen, Löschen mit touch, mv, rm.
3. **Softwareinstallation**: Einsatz von apt-get und sudo.
4. **Superuser-Modus**: Ausführen privilegierter Befehle.
5. **Verzeichnismanagement**: Umgang mit mkdir und rmdir.
6. **Dateibearbeitung**: Einsatz des Nano-Texteditors.
7. **Pfadstrukturen**: Verständnis von absoluten und relativen Pfaden.

8. **Farbcodierung**: Identifikation von Dateitypen im Terminal.

Video

1.10.3 Video 3: Python-Grundlagen kompakt

Diese Tutorialreihe bietet eine kompakte Einführung in Python, ideal für Raspberry Pi-Anfänger und all jene, die sich schnell essentielle Programmierkonzepte aneignen möchten. Behandelt werden Themen wie Variablen, Arrays, Bedingungsanweisungen und Schleifen, ergänzt durch spezifische Python-Features.

1. **Python vs. Arduino**: Unterschiede zwischen interpretierter und kompilierter Sprache.
2. **Syntax und Variablen**: Umgang mit verschiedenen Datentypen.
3. **Arrays und mehrdimensionale Arrays**: Flexibler Datenhaltung und Organisation.
4. **Zugriff auf Array-Elemente**: Indexierung und Datenmanipulation.
5. **Erweiterte Python-Funktionen**: Arrays erweitern und Zeichenketten indizieren.
6. **Variablenzuweisung und Benutzereingaben**: Nutzung der `input()` Funktion.
7. **Bedingungsanweisungen**: Einsatz von `if`, `elif`, `else`.
8. **Arithmetische Operationen**: Grundlegende Rechenoperationen.

Video

1.10.4 Video 4: GPIO-Pins am Raspberry Pi nutzen

Dieses Tutorial führt in die Grundlagen und die Nutzung von GPIO-Pins am Raspberry Pi mit Python ein. Zuschauer lernen, GPIO-Pins einzurichten, LEDs zu steuern, Verzögerungen zu nutzen und Ressourcen nach Gebrauch zu säubern. Eine Hausaufgabe vertieft das Gelernte praktisch.

1. GPIO-Pins des Raspberry Pi und Nummerierung.
2. Konfiguration und Nutzung des Steckbretts.
3. LED-Schaltungsschema.
4. Schaltplan auf dem Steckbrett realisieren.
5. Tipps für saubere Komponentenanordnung.
6. Bedeutung der Komponentenorientierung.
7. GPIO-Pins mit `gpio.setmode()` und `gpio.setup()` einrichten.
8. LEDs mit `gpio.output()` steuern.
9. Verzögerungen mit `time.sleep()`.
10. Ressourcen mit `gpio.cleanup()` bereinigen.
11. Aufgabe: LED-Blinkprogramm nach Benutzereingabe.
12. Tipps: Wichtigkeit der GPIO-Bereinigung und Teilen von Lösungen.

Video

Hausaufgabenlösung

1.10.5 Video 5: Verständnis binärer Zahlen

Tauchen Sie ein in die Welt der binären Zahlen, erkunden Sie ihre Bedeutung in der Informatik und lernen Sie die Grundlagen der Binärarithmetik. Machen Sie eine praktische Hausaufgabe mit dem Raspberry Pi, um einen fünfstelligen Binärzähler mit LEDs und Widerständen zu erstellen.

1. **Binäre Grundlagen**: Computer arbeiten binär, sie nutzen Nullen und Einsen für alle Berechnungen.
2. **Datenrepräsentation**: Binäre Zahlen kodieren verschiedene Datentypen, von Zahlen über Text bis hin zu Bildern.
3. **Hardware-Komponenten**: Erfahren Sie, wie Ein-Aus-Schalter die Grundlage der binären Repräsentation in Computern bilden.
4. **Binärarithmetik**: Verstehen Sie den Prozess der binären Addition, der traditionelle Übertragungsmethoden widerspiegelt.
5. **Raspberry Pi Hausaufgabe**: Werden Sie praktisch mit einer Aufgabe, einen fünfstelligen Binärzähler mit LEDs und Widerständen zu erstellen.

Video

Hausaufgabenlösung

1.10.6 Video 6: GPIO-Eingänge und Widerstände verstehen

In diesem Tutorial erklärt Paul McWhorter die Rolle von Pull-Up- und Pull-Down-Widerständen in Mikrocontroller-Schaltungen. Er zeigt, wie man GPIO-Eingänge auf Raspberry Pi-Plattformen praktisch nutzt.

1. **Einführung in GPIO-Eingänge**: Lesen digitaler Eingänge von Mikrocontroller-Pins.
2. **Schaltungsaufbau**: Aufbau mit Druckknöpfen, Widerständen und Mikrocontroller-Boards.
3. **GPIO-Pins konfigurieren**: Eingabemodus einstellen und Werte in Python lesen.
4. **Floating-Inputs verhindern**: Einsatz von Pull-Up- oder Pull-Down-Widerständen.
5. **Praktische Implementierung**: Aufbau einer Schaltung auf einem Steckbrett.
6. **Hausaufgabe**: Steuerung einer LED mit einem Druckknopf und Teilen der Lösung auf YouTube.

Video

Hausaufgabenlösung

1.10.7 Video 7: GPIO-Eingänge über Tastenschalter

Lernen Sie, wie Sie GPIO-Schaltkreise für Schalter oder Tasten vereinfachen können, indem Sie den internen Pull-Up-Widerstand der GPIO-Pins des Raspberry Pi nutzen. Dies beinhaltet eine Demonstration des Konzepts, den Aufbau des Schaltkreises und ein Python-Codebeispiel zum Umschalten einer LED.

1. Verständnis von Pull-Up- und Pull-Down-Widerständen in GPIO-Schaltkreisen.
2. Demonstration eines Schaltkreisaufbaus mit einem Taster, Pull-Up-Widerstand und LED.
3. Einführung in das Konzept der Nutzung des internen Pull-Up-Widerstands in GPIO-Pins des Raspberry Pi.
4. Erklärung, wie man den internen Pull-Up-Widerstand in der GPIO-Pin-Konfiguration aktiviert.
5. Bereitstellung eines Python-Codebeispiels zum Umschalten einer LED mit einem Taster unter Nutzung des internen Pull-Up-Widerstands.

6. Hervorhebung der Bedeutung von Kippschaltern in praktischen Anwendungen und Zuweisung einer Hausaufgabe zur Implementierung eines Kippschalters mit dem vorhandenen Schaltkreis.

Video

1.10.8 Video 8: Verwendung eines Tasters als Umschalter für die LED-Steuerung

Lernen Sie, wie Sie einen LED-Umschaltkreis mit einem Raspberry Pi und Python-Programmierung erstellen.

1. **Schaltungsaufbau**: Einrichtung einer LED und eines Tastenschalters, verbunden mit GPIO-Pins eines Raspberry Pi.
2. **Python-Programmierung**: Nutzung der RPi.GPIO-Bibliothek zur Steuerung der GPIO-Pins und zum Lesen des Zustands des Tastenschalters.
3. **Umschaltlogik**: Implementierung einer Logik in Python, um die LED basierend auf dem Zustand des Tastenschalters umzuschalten.
4. **Variablenmanagement**: Verwendung von Variablen, um die aktuellen und vorherigen Zustände des Tastenschalters und der LED zu verfolgen.

Video

1.10.9 Video 9: Nutzung von GPIO-Pins für PWM zur Simulation von Analogausgaben

Dieses Video führt Sie durch die Grundlagen von PWM (Pulsweitenmodulation) mit den GPIO-Pins des Raspberry Pi und demonstriert die Steuerung der LED-Helligkeit sowie die Echtzeitanpassung der Frequenz.

1. Einführung in PWM und seine Rolle bei der Simulation analoger Signale.
2. Verständnis des Tastverhältnisses und seiner Bedeutung bei PWM-Signalen.
3. Praktische Demonstration der LED-Helligkeitssteuerung unter Verwendung von PWM mit Raspberry Pi GPIO-Pins.
4. Einsatz eines Oszilloskops zur Visualisierung von PWM-Signalen, Überprüfung von Tastverhältnis und Frequenz.
5. Erläuterung der Frequenz und ihres Einflusses auf PWM-Signale.
6. Echtzeitdemonstration der Änderung von Tastverhältnis und Frequenz mittels Python-Befehlen im Terminal des Raspberry Pi.
7. Hausaufgabe zur präzisen Steuerung der LED-Helligkeit mit Druckknöpfen, um praktisches Lernen und Experimentieren zu fördern.
8. Bedeutung der GPIO-Pins bei der Steuerung verschiedener Komponenten wie LEDs, Motoren und Sensoren.

Video

1.10.10 Video 10: Erstellung einer dimmbaren LED mit zwei Tastern

In diesem Tutorial erfahren Sie, wie Sie mithilfe von Raspberry Pi GPIO-Pins und Python-Programmierung einen dimmbaren LED-Schaltkreis erstellen. Paul McQuarter zeigt von der Schaltungsvorbereitung bis zur Code-Implementierung jeden Schritt. Besonderer Fokus liegt auf der sanften Helligkeitssteuerung durch PWM (Pulsweitenmodulation) und der Anwendung einer exponentiellen Skala für präzise Helligkeitsanpassungen. Zudem werden Tipps zur Fehlersuche und Code-Verfeinerung für optimale Ergebnisse gegeben.

1. **Schaltungsaufbau**: Erstellen dimmbarer LED-Schaltungen mit Tastern zur Steuerung.
2. **Python-Codierung**: Implementierung von PWM für Helligkeitssteuerung und Nutzung exponentieller Skalen.
3. **PWM und exponentielle Skala**: Verständnis von Tastgrad, exponentiellen Skalen und deren Einfluss auf die Helligkeitssteuerung.
4. **Fehlersuche und Verfeinerung**: Optimierung der Schaltung und des Codes für sanfte Helligkeitsübergänge.

Video

1.10.11 Video 11: Verständnis und Nutzung einer RGB-LED

Erlernen Sie in diesem Tutorial die Steuerung einer RGB-LED mit den GPIO-Pins des Raspberry Pi, einschließlich Aufbau, Verkabelung und Python-Codierungsschritten.

1. **RGB-LED-Komponenten**: Verstehen Sie die Rot-, Grün- und Blau-Pins der RGB-LED.
2. **GPIO-Setup**: Erfahren Sie mehr über das physische Pin-Nummerierungssystem für das GPIO-Setup auf dem Raspberry Pi.
3. **Verkabelungsanleitung**: Befolgen Sie eine Schritt-für-Schritt-Anleitung zur Verkabelung der RGB-LED mit den GPIO-Pins des Raspberry Pi, einschließlich strombegrenzender Widerstände.
4. **Python-Codierung**: Sehen Sie eine Demonstration von Python-Code zur Steuerung der RGB-LED-Farben mit GPIO-Pins.
5. **Tipps zur Fehlerbehebung**: Erhalten Sie nützliche Tipps für den effektiven Umgang mit Komponenten und Verbindungen.

Video

1.10.12 Video 12: Steuerung einer RGB-LED mit Tastern

Entdecken Sie, wie Sie RGB-LEDs mit Raspberry Pi-Plattformen steuern können, einschließlich Hardware-Setup, Code-Organisation, GPIO-Pin-Konfiguration, Farbmischung, dimmbare LED-Steuerung, Debugging und Möglichkeiten zur Projekterweiterung.

1. **Hardware-Setup**: Konfiguration von RGB-LEDs, Tastern und GPIO-Verbindungen.
2. **Code-Organisation**: Systematische Organisation des Codes für eine einfachere Wartung.
3. **GPIO-Pin-Setup**: Konfiguration der GPIO-Pins für Eingabe- und Ausgabefunktionalitäten.
4. **Hauptschleifenimplementierung**: Erstellung einer Hauptschleife zur Überwachung des Tasterzustands und Aktualisierung des LED-Zustands.
5. **Debugging und Testen**: Sicherstellung der ordnungsgemäßen Hardware- und Softwarefunktionalität durch Debugging-Verfahren.
6. **RGB-LED-Farbmischung**: Experimentieren mit Farbkombinationen unter Verwendung von Rot, Grün und Blau.

7. **Dimmbare LED-Steuerung**: Erkundung der Dimmung von LEDs für präzise Farb- und Helligkeitseinstellungen.
8. **Schaltungstests**: Identifizieren und Lösen von Schaltungsproblemen wie schlechten Verbindungen.
9. **Programmierlogik**: Entwicklung von Code-Logik zur Steuerung des Verhaltens von RGB-LEDs.
10. **Projekterweiterung**: Erwägung der Erweiterung des Projekts mit fortgeschrittenen Funktionen wie dimmbare LED-Steuerung.

Video

1.10.13 Video 13: Farbe einer RGB-LED mit Tastern einstellen

Dieses Video demonstriert, wie Sie eine RGB-LED mit Tastern an einem Raspberry Pi steuern, einschließlich Hardware-Setup, Programmierung und Fehlersuche.

1. Lernen Sie, wie RGB-LEDs und Taster an die GPIO-Pins eines Raspberry Pi angeschlossen werden.
2. Verstehen Sie die Bedeutung einer systematischen Code-Organisation für einfacheres Debugging.
3. Richten Sie GPIO-Pins sowohl für Eingabe (Taster) als auch für Ausgabe (LED-Steuerung) ein.
4. Implementieren Sie eine Hauptschleife, um kontinuierlich Tasterzustände zu lesen und LED-Zustände zu aktualisieren.
5. Befolgen Sie Debugging- und Testverfahren, um eine ordnungsgemäße Funktionalität zu gewährleisten.
6. Erkunden Sie die Farbmischung, indem Sie die Rot-, Grün- und Blauwerte anpassen, um ein Spektrum von Farben zu erzeugen.

Video

1.10.14 Video 14: Installation von Visual Python auf dem Raspberry Pi

Dieses Video demonstriert, wie man eine RGB-LED mit Tastern an einem Raspberry Pi steuert, mit einem Fokus auf der Installation von Visual Python, der Erstellung einer einfachen 3D-Animation und deren Integration mit dem LED-Steuerungscode aus einer vorherigen Lektion. Das Tutorial umfasst die Installation von Visual Python, die Integration bestehender Schaltkreise und Codes zur Steuerung von RGB-LEDs, die Erstellung von 3D-Animationen zur Darstellung von LED-Zuständen, das Verständnis von PWM-Tastverhältnissen und Farbdarstellung, die Fehlerbehebung bei der Codeausführung und Kalibrierungsproblemen sowie die Erkundung weiterer Anpassungs- und Optimierungsmöglichkeiten.

1. Installation von Visual Python auf dem Raspberry Pi und Einrichtung der Umgebung.
2. Integration bestehender Schaltkreise und Codes aus vorherigen Lektionen zur Steuerung von RGB-LEDs.
3. Erstellung von 3D-Animationen mit Visual Python zur Darstellung von LED-Zuständen.
4. Verständnis der Umwandlung zwischen PWM-Tastverhältnissen, die für die LED-Steuerung verwendet werden, und der Farbdarstellung in Visual Python.
5. Fehlerbehebung bei Problemen mit der Codeausführung und der Kalibrierung zwischen realen und virtuellen LEDs.
6. Erkundung weiterer Anpassungs- und Optimierungsmöglichkeiten für die Animationen und die LED-Steuerung.

Video

1.10.15 Video 15: Analogeingabe am Raspberry Pi mit dem ADC0834

Lernen Sie, wie Sie den analogen Eingang an Ihrem Raspberry Pi mit dem Analog-Digital-Wandler ADC0834 aktivieren können, einschließlich Hardware-Setup, Bibliotheksinstallation und dem Wechsel zur Thonny IDE für die Python-Entwicklung.

1. Einführung in den analogen Eingang und die Notwendigkeit eines ADC am Raspberry Pi.
2. Hardware-Setup und GPIO-Pin-Verbindungen für den ADC0834-Chip.
3. Herstellung korrekter Strom- und Masseverbindungen für Komponenten.
4. Installation und Nutzung der ADC0834-Bibliothek für Python.
5. Wechsel zu Thonny als moderne IDE für sauberes und effizientes Programmieren.
6. Best Practices für die Fehlersuche und Überprüfung der Verbindungen vor dem Programmieren.
7. Bibliotheksinstallation: Installieren Sie die ADC-Bibliothek für Raspberry Pi.
8. Python-Setup: Speichern Sie den Bibliothekscode in einer Python-Datei und verschieben Sie ihn in das richtige Verzeichnis.
9. GPIO-Nutzung: Richten Sie ein Python-Programm ein, um analogen Eingang über GPIO-Pins zu lesen.
10. Analoge Lesung: Schreiben Sie ein Python-Skript, um kontinuierlich analoge Werte vom Potentiometer zu lesen.
11. Kanalwechsel: Demonstration des Wechsels zwischen verschiedenen Kanälen des ADC-Chips.
12. Hausaufgabe: Erstellen Sie eine dimmbare LED, die durch die Position des Potentiometers gesteuert wird.

Video

1.10.16 Video 16: Dimmbare LED mit Potentiometer erstellen

In diesem Video lernen Sie, wie Sie mithilfe eines Raspberry Pi, eines ADC-0834 Analog-Digital-Wandlers und PWM-Techniken eine dimmbare LED bauen. Es umfasst alle Schritte von der Hardware-Vorbereitung bis zur Python-Programmierung.

1. **Einführung**: Überblick über die Tutorialreihe.
2. **Hardware-Setup**: Notwendige Komponenten und ihre Verbindung.
3. **GPIO und PWM-Initialisierung**: Einrichtung für die LED-Steuerung.
4. **Python-Programmierung**: Schrittweise Anleitung inklusive Bibliotheksimport und Tastverhältnisberechnung.
5. **Mathematische Grundlagen**: Umrechnung von Potentiometerwerten in LED-Helligkeit.
6. **Demonstration**: Vorführung der dimmbaren LED.
7. **Nächste Aufgaben**: Ankündigung weiterführender Projekte.
8. **Danksagung**: Wertschätzung der Unterstützer.

Video

1.10.17 Video 17: Jede Farbe mit einer RGB-LED mischen

In diesem Video wird gezeigt, wie man mit einem Raspberry Pi und einem ADC0834-Chip eine RGB-LED steuert. Die Serie, gesponsert von SunFounder, konzentriert sich auf die Schritt-für-Schritt-Anleitung vom Hardware-Setup bis zur Python-Programmierung für die Farbmischung.

1. Vorstellung der Tutorialreihe und Übergang von einer dimmbaren zu einer RGB-LED.
2. Aufbau der Schaltung mit ADC0834-Chip, Potentiometern und RGB-LED.
3. Anleitung für Verkabelung und GPIO-Pin-Zuweisungen.
4. Programmierung der Farbsteuerung mit PWM durch Python und Thonny IDE.
5. Einrichtung von PWM auf dem Raspberry Pi zur Farbkontrolle.
6. Lesen analoger Werte zur Bestimmung der LED-Helligkeit.
7. Dynamische Anpassung der Helligkeit basierend auf Potentiometer-Eingaben.
8. Tipps zur Codeorganisation und Fehlersuche.

Video

1.10.18 Video 18: Einsatz eines Joysticks mit dem Raspberry Pi

Lernen Sie, wie Sie einen Joystick in Ihre Raspberry Pi-Projekte mit dem SunFounder-Kit integrieren, einschließlich Einrichtung, Verkabelung, Programmierung und Testen.

1. **Einführung**: Entdecken Sie die Vielseitigkeit des Einsatzes eines Joysticks in Raspberry Pi-Projekten.
2. **SunFounder-Kit-Demonstration**: Überblick über die im SunFounder-Kit enthaltenen Komponenten.
3. **Verständnis des Joysticks**: Erklärung der Komponenten und Funktionalität des Joysticks.
4. **Verkabelung**: Schritt-für-Schritt-Anleitung zur Verkabelung des Joysticks mit den GPIO-Pins des Raspberry Pi.
5. **Programmierprozess**: Nutzung von Python und der RPi.GPIO-Bibliothek zum Lesen analoger Werte und Erkennen von Tastendrücken am Joystick.
6. **Testen und Validierung**: Sicherstellung der ordnungsgemäßen Funktionalität des Joysticks und der Taste durch Tests.

Video

1.10.19 Video 19: Einfache Steuerung eines Servos mit dem Raspberry Pi

Erfahren Sie, wie Sie einen Servomotor mit einem Raspberry Pi steuern können. Paul McQuarter führt Sie durch die Einrichtung, Codierung und Tests.

1. **Hardware-Einrichtung**: Paul zeigt die korrekte Verkabelung und Ausrichtung des Servomotors.
2. **Stromversorgungsüberlegungen**: Er erklärt die Wichtigkeit einer externen Stromversorgung für größere Servos, um den Raspberry Pi nicht zu beschädigen.
3. **PWM-Grundlagen**: Erfahren Sie, wie die Pulsweitenmodulation (PWM) die Position des Servomotors steuert.
4. **Python-Code**: Schritt-für-Schritt-Anleitung zur Einrichtung und Steuerung des Servomotors mit GPIO-Pins.
5. **Testen und Kalibrieren**: Lernen Sie, den Bewegungsbereich des Servomotors durch Eingabe verschiedener PWM-Prozentsätze zu testen und zu kalibrieren.

6. **Fehlerbehandlung**: Entdecken Sie, wie Fehler behandelt und GPIO-Pins in Python ordnungsgemäß bereinigt werden.

Video

1.10.20 Video 20: Servoposition mit einem Potentiometer kontrollieren

Dieses Video zeigt, wie man mithilfe eines Potentiometers und der Raspberry Pi GPIO die Servoposition steuert. Es umfasst die Schaltungseinrichtung, Python-Codierung, ADC, PWM, mathematische Berechnungen, Fehlerbehandlung und Verbindungswichtigkeit.

1. **Schaltungseinrichtung**: Überprüfung der analogen Eingangsschaltung des vorherigen Unterrichts mit ADC0834 für Raspberry Pi.
2. **Servo-Einrichtung**: Anweisungen zur Einrichtung des Servos in der Schaltung, einschließlich Verkabelung und Stromversorgungsüberlegungen.
3. **ADC und PWM verstehen**: Vertrautheit mit den Konzepten von Analog-Digital-Wandler (ADC) und Pulsweitenmodulation (PWM).
4. **Lesen analoger Signale**: Wie man analoge Signale von einem Potentiometer am Raspberry Pi liest.
5. **Berechnung des PWM-Prozentsatzes**: Erlernen der Formel zur Berechnung des PWM-Prozentsatzes basierend auf dem analogen Signalwert.
6. **Servomotor steuern**: Den Prozess der Anwendung des berechneten PWM-Prozentsatzes zur Positionskontrolle eines Servomotors erforschen.
7. **Synchronisation**: Sicherstellen, dass die Position des Potentiometers und des Servomotors synchronisiert sind für präzise Steuerung.
8. **Mathematische Berechnungen**: Erklärung der mathematischen Berechnungen zur Bestimmung von PWM-Werten basierend auf Potentiometerablesungen.
9. **Fehlerbehandlung**: Implementierung von try-except-Blöcken für Programmbeendigung und GPIO-Bereinigung.
10. **Verbindungswichtigkeit**: Verstehen, warum zuverlässige Verbindungen für genaue Messungen und Ausgaben wichtig sind.

Video

1.10.21 Video 21: Verwendung eines HC-SR04 Ultraschallsensors zur Echolokalisierung

Dieses Tutorial zeigt die Erstellung eines Ultraschall-Abstandssensors mit einem Raspberry Pi und dem HC-SR04 Sensor. Es erklärt die Prinzipien der Echolokalisierung, stellt die benötigten Komponenten vor, richtet die Verdrahtung ein und führt durch den Codierungsprozess. Besonderes Augenmerk liegt auf präzisen Zeitabläufen für genaue Distanzmessungen und guten Ingenieurpraktiken.

1. **Echolokalisierung**: Schall zur Erfassung von Objektpositionen, inspiriert von Fledermäusen und Walen.
2. **Komponenten**: Einführung des HC-SR04 Ultraschallsensors und dessen Anschluss an den Raspberry Pi.
3. **Verdrahtung**: Anschließen des HC-SR04 Sensors an die GPIO-Pins des Raspberry Pi.
4. **Codierung**: Python-Code für Ultraschallimpulse, Sensorauslösung und Messung der Echolaufzeit.
5. **Zeitliche Genauigkeit**: Wichtigkeit präziser Zeitabläufe für genaue Distanzmessungen.
6. **Ingenieurpraktiken**: Betonung der Planung und des Verständnisses des Codes vor der Implementierung.

7. **Warten auf Echo**: Nutzung einer While-Schleife, um das Echo des Sensors abzuwarten.
8. **Startzeit erfassen**: Markierung des Messstarts durch Erfassen der Systemzeit beim Echo.
9. **Ping-Laufzeit messen**: Berechnung der Ping-Laufzeit aus der Zeitdifferenz zwischen Echo-Hoch und -Tief.
10. **Einheiten umrechnen**: Multiplikation der Ping-Laufzeit für bessere Lesbarkeit.
11. **Verzögerung**: Einführung einer Pause nach jeder Messung, um Mehrfachechos zu verhindern.
12. **Entfernung berechnen**: Nutzung von Schallgeschwindigkeit und Ping-Laufzeit zur Entfernungsberechnung.

Video

1.10.22 Video 22: Distanzmessung mit dem HC-SR04 Ultraschallsensor

Erfahren Sie, wie Sie mit dem HC-SR04 Ultraschallsensor und einem Raspberry Pi Entfernungen messen können. Dieses Tutorial umfasst Verkabelung, Codierung und Einheitenkonvertierung. Erhalten Sie Einblicke in Fehlerbehebung, verstehen Sie die Prinzipien der Distanzmessung, entdecken Sie praktische Anwendungen und erhalten Sie eine Hausaufgabe zur experimentellen Messung der Schallgeschwindigkeit.

1. **Einführung**: Überblick über das Tutorial und Anerkennung des Sponsorings.
2. **Vorherige Aufgabe**: Zusammenfassung der HC-SR04 Sensoraufgabe aus der vorherigen Lektion.
3. **Einrichtung und Verkabelung**: Erklärung der Verdrahtung, einschließlich Anschluss an den Raspberry Pi.
4. **Code-Erklärung**: Verständnis des Codes zur Messung der Ping-Laufzeit und der Entfernung.
5. **Fehlerbehebung**: Tipps zur Identifizierung und Behebung von Codierungsfehlern.
6. **Verständnis der Distanzmessung**: Erklärung der Prinzipien des Ultraschallsensors.
7. **Korrekturfaktor**: Anerkennung der Korrektur, die für eine genaue Distanzmessung erforderlich ist.
8. **Einheitenkonvertierung**: Diskussion über die Umrechnung von Einheiten für genaue Messungen.
9. **Praktische Anwendungen**: Erkunden Sie realweltliche Anwendungen von Ultraschallsensoren.
10. **Fusion 360 Integration**: Ermutigung zur Kombination von Raspberry Pi Fähigkeiten mit 3D-Druck Kenntnissen.
11. **Hausaufgabe**: Herausforderung zur experimentellen Messung der Schallgeschwindigkeit mit dem Sensor.

Video

1.10.23 Video 23: Messen der Schallgeschwindigkeit mit einem Ultraschallsensor

Erfahren Sie in diesem Schritt-für-Schritt Tutorial, wie Sie die Schallgeschwindigkeit mithilfe eines Raspberry Pi und eines HC-SR04 Ultraschallsensors messen.

1. Einführung in die Verwendung des HC-SR04 Ultraschallsensors zur Messung der Schallgeschwindigkeit.
2. Verständnis der grundlegenden Gleichung für Entfernung, Geschwindigkeit und Zeit.
3. Hardwareeinrichtung mit Raspberry Pi und dem Ultraschallsensor.
4. Schreiben von Python-Code zur Steuerung des Sensors und Berechnung der Schallgeschwindigkeit.
5. Durchführung einer dimensionsanalytischen Analyse zur Umrechnung von Einheiten für die endgültige Geschwindigkeitsberechnung.
6. Praktische Demonstration des Aufbaus und des Messvorgangs unter Verwendung eines Lineals und eines Ziels.

Video

1.10.24 Video 24: Verwendung eines PIR-Bewegungssensors mit dem Raspberry Pi

Erfahren Sie in diesem Tutorial von Paul McQuarter, wie Sie einen PIR-Bewegungssensor mit einem Raspberry Pi verwenden. Paul erklärt die Komponenten, Einrichtung, Codierung und Tests des Sensors für Bewegungserkennungsprojekte.

1. **Einführung in den PIR-Bewegungssensor**: Verstehen Sie, was ein PIR-Bewegungssensor ist und wie er funktioniert.
2. **Sensor-Komponenten**: Erfahren Sie mehr über die wichtigen Komponenten des PIR-Bewegungssensors, einschließlich des Infrarotsensors und der Fresnel-Linse.
3. **Erkennungsmechanismus**: Entdecken Sie, wie der Sensor Bewegung und Wärme erkennt, um präzise auszulösen.
4. **Sensor-Einrichtung**: Folgen Sie Paul, während er zeigt, wie man den PIR-Sensor über GPIO-Pins mit dem Raspberry Pi verbindet.
5. **Codierung des Sensors**: Sehen Sie, wie man ein einfaches Python-Skript schreibt, um Bewegungen mithilfe der RPi.GPIO-Bibliothek zu erkennen.
6. **Tests und Feinabstimmung**: Erfahren Sie mehr über Testverfahren und Feinabstimmungsoptionen für optimale Leistung des Sensors.

Video

1.10.25 Video 25: Temperatur und Luftfeuchtigkeit mit dem DHT-11 Sensor messen

In diesem Tutorial lernen Sie, wie Sie einen DHT11 Temperatur- und Luftfeuchtigkeitssensor mit einem Raspberry Pi verbinden und die Daten mit Python auslesen.

1. **Einführung**: Begleiten Sie uns in dieser von SunFounder gesponserten Raspberry Pi Tutorial-Serie.
2. **Benötigte Komponenten**: Sammeln Sie Ihren Raspberry Pi, Jumper-Kabel, den DHT11 Sensor und einen 10k Widerstand.
3. **Verkabelungsanweisungen**: Befolgen Sie detaillierte Anweisungen, um die Komponenten korrekt miteinander zu verbinden und eine Beschädigung des Sensors zu vermeiden.
4. **Installation der Python-Bibliothek**: Erfahren Sie, wie Sie die erforderliche Python-Bibliothek zur Interaktion mit dem DHT11 Sensor installieren.
5. **Implementierung des Codes**: Schreiben Sie Python-Code, um Temperatur- und Luftfeuchtigkeitsdaten vom Sensor zu lesen und sie auf dem Terminal auszugeben.
6. **Ausführung des Codes**: Sehen Sie den Code in Aktion, wenn er die Sensorwerte interpretiert.
7. **Zukünftige Projekte**: Entdecken Sie potenzielle Projekte, die Komponenten aus dem SunFounder Kit kombinieren.

Video

1.10.26 Video 26: Verwendung eines LCD1602 LCD-Displays mit I2C

Erfahren Sie in diesem umfassenden Tutorial von Paul McWhorter, wie Sie ein LCD 1602 Display mit einem Raspberry Pi über die I2C-Kommunikation einrichten und anschließen.

1. Dank an SunFounder für die Bereitstellung des Raspberry Pi Kits.
2. **Erklärung des Projekts**: Anschluss eines LCD 1602 Displays an den Raspberry Pi über I2C.
3. **Vorteile der I2C-Kommunikation**: Vereinfachte Verkabelung mit nur vier Verbindungen im Vergleich zu früheren Methoden.
4. **Hardwareeinrichtung**: Anschließen des LCD-Displays an die GPIO-Pins des Raspberry Pi über eine Breakout-Platine.
5. **Konfiguration des Raspberry Pi für I2C**: Aktivieren der I2C-Schnittstelle über raspi-config.

Video

1.10.27 Video 27: Bau eines Temperatur- und Feuchtigkeitssystems mit LCD-Anzeige

In diesem Tutorial erfahren die Zuschauer, wie sie ein Temperatur- und Feuchtigkeitserfassungsgerät mit einem Raspberry Pi, einem DHT11-Sensor, einer LCD-Anzeige und einem Druckknopf konstruieren können. Das Video bietet eine ausführliche Anleitung sowohl zur Hardware-Einrichtung als auch zur erforderlichen Python-Codierung für das Projekt.

1. Einführung in den Bau eines Temperatur- und Feuchtigkeitserfassungsgeräts mit dem Raspberry Pi.
2. Benötigte Komponenten, darunter Raspberry Pi, DHT11-Sensor, LCD-Anzeige, Druckknopf, weiblich-männliche Kabel und Jumper-Kabel.
3. Die Hardware-Einrichtung umfasst das Anschließen des Raspberry Pi an die SunFounder-Breakout-Platine und anschließend das Verbinden des DHT11-Sensors, der LCD-Anzeige und des Druckknopfs mit Jumper-Kabeln.
4. Einrichten der erforderlichen Bibliotheken wie LCD1602, GPIO, DHT11 und Zeit.
5. Initialisierung der Komponenten, einschließlich des Einrichtens des GPIO-Modus, der Initialisierung des DHT11-Sensors, der LCD-Anzeige und des Druckknopfs.
6. Das Lesen des Tastenzustands über GPIO-Eingang zum Umschalten zwischen Fahrenheit- und Celsius-Anzeigen auf der LCD-Anzeige.

Video

1.10.28 Video 28: Verständnis und Verwendung von aktiven und passiven Summer

Erfahren Sie in diesem umfassenden Tutorial, wie Sie Summer in Raspberry Pi-Projekte integrieren und ein Temperatursensor-Alarmsystem erstellen. Das Tutorial umfasst Hardware-Einrichtung, Programmierung, Tests und Community-Engagement.

1. **Aktive vs. passive Summer**: Verstehen Sie die Unterschiede und Eigenschaften von aktiven und passiven Summern.
2. **Hardware-Einrichtung**: Folgen Sie einem Schaltplan, um Summer, Transistoren, Widerstände und GPIO-Pins korrekt zu verbinden.
3. **Programmierung**: Erkunden Sie Python-Code zur Steuerung von Summer und zur Implementierung von Temperatursensor-Funktionen.

4. **Modusübergang**: Implementieren Sie einen Druckknopf zum Übergang zwischen Programmmodus und Überwachungsmodus.
5. **Community-Engagement**: Teilen Sie Projektlösungen auf YouTube und interagieren Sie mit der Community.
6. **Integration von Fähigkeiten**: Wenden Sie zuvor gelernte Konzepte an, um ein umfassendes Projekt zu entwickeln.

Video

1.10.29 Video 29: Aufbau eines programmierbaren Temperatursensor-Alarmsystems

Dieses Video zeigt, wie man mit einem Raspberry Pi ein programmierbares Temperaturüberwachungssystem mit Alarm erstellt. Es behandelt die Hardwareeinrichtung, das Codieren in Python und Debugging-Techniken. Das Tutorial betont strukturiertes Programmieren und Fehlerbehebungsmethoden zur Aufrechterhaltung der Codeorganisation und Klarheit.

1. **Hardwareeinrichtung**: Erklärt die erforderlichen Hardwarekomponenten, einschließlich Raspberry Pi, Breakout-Board, ADC0834, DHT11-Temperatursensor, Umschaltknopf, aktiver Summer und Transistor.
2. **Projektübersicht**: Zeigt das fertige Gerät und seine Komponenten und Funktionalitäten.
3. **Schematische Übersicht**: Diskutiert das Schaltbild des Projekts und hebt die Verbindungen zwischen den Komponenten hervor.
4. **Programmierung**: Geht den Python-Code durch, erläutert das Lesen des Sensorwerts, den Moduswechsel und das Auslösen des Alarms basierend auf Temperaturschwellenwerten.
5. **Debugging**: Zeigt Debugging-Techniken zur Identifizierung und Behebung von Fehlern im Code und den Hardwareverbindungen.
6. Analog-Digital-Wandlung für die Temperaturmessung.
7. LCD-Display-Schnittstelle für die Anzeige der Echtzeittemperatur.
8. Benutzerinteraktion mit Tasten zum Umschalten zwischen Programmier- und Überwachungsmodi.
9. Alarmauslösung basierend auf Temperaturschwellenwerten.
10. Strukturierte Programmieransätze zur Aufrechterhaltung der Codeorganisation und -klarheit.

Video

1.10.30 Video 30: Verwendung eines Fotowiderstands zur Lichterkennung auf dem Raspberry Pi

Erfahren Sie, wie Sie einen Fotowiderstand in Ihre Raspberry Pi-Projekte integrieren, um die Lichtintensität zu messen und Aktionen basierend auf der Helligkeit zu steuern.

1. **Einführung in den Fotowiderstand**: Lernen Sie, wie sich der Widerstand eines Fotowiderstands mit der Lichtintensität ändert.
2. **Physik hinter dem Fotowiderstand**: Verstehen Sie die physikalischen Prinzipien der Festkörperphysik, die dem Betrieb des Fotowiderstands zugrunde liegen.
3. **Aufbau des Schaltkreises**: Entdecken Sie, wie Sie mit einem Fotowiderstand und einem Lastwiderstand einen Spannungsteiler-Schaltkreis erstellen.
4. **Programmierung des Raspberry Pi**: Schreiben Sie Python-Code, um analoge Werte aus dem Fotowiderstand mithilfe des ADC0834-Moduls zu lesen.

5. **Test des Schaltkreises**: Sehen Sie, wie der Schaltkreis auf Änderungen der Lichtverhältnisse reagiert und Änderungen im Widerstand und in der Spannung demonstriert.
6. **Praktische Anwendungen**: Erforschen Sie potenzielle Anwendungen von Fotowiderständen in Projekten wie automatischen Beleuchtungssystemen.

Video

1.10.31 Video 31: Alarm zur Bewegungserkennung im Dunkeln

In diesem Video wird gezeigt, wie man ein Einbruchsalarm-System mit einem Raspberry Pi, einem Bewegungssensor, einem Lichtsensor und einem Summer baut. Es bietet eine schrittweise Anleitung sowohl für den Hardwareaufbau als auch für die Python-Codierung.

1. **Einführung in das Projekt**: Aufbau eines Einbruchsalarm-Systems mit dem Raspberry Pi.
2. **Übersicht über die verwendeten Komponenten**: Raspberry Pi, PIR-Bewegungssensor, Lichtsensor (ADC-0834), aktiver Summer und Transistor (S8550).
3. **Erläuterung des Hardware-Aufbaus**: Verkabelung der Komponenten, einschließlich Anschluss des Summers, Bewegungssensors und Lichtsensors an die GPIO-Pins des Raspberry Pi.
4. **Programmierung des Alarmsystems**: Verwendung von Python-Code zum Lesen von Sensorsignalen, Erkennen von Bewegung und Dunkelheit und Auslösen des Summeralarms entsprechend.
5. **Test des Systems**: Demonstration der Funktionalität des Alarmsystems durch Simulation der Einbruchserkennung unter verschiedenen Lichtbedingungen.
6. **Diskussion zukünftiger Projekte**: Vorschau auf kommende Tutorials zu zusätzlichen Komponenten wie Beschleunigungsmessern, Gyroskopen, RFID und Kameras für weitere Raspberry Pi-Projekte.

Video

1.10.32 Video 32: Verständnis und Verwendung eines Tastenfelds mit dem Raspberry Pi

Erfahren Sie, wie Sie effizient ein Tastenfeld in Ihre Raspberry Pi-Projekte integrieren, indem Sie eine passive Zeilen-Spalten-Adressierung für die Eingabelese verwenden.

1. **Einführung**: Lernen Sie, wie Sie ein Tastenfeld in Ihre Raspberry Pi-Projekte integrieren.
2. **Passive Zeilen-Spalten-Adressierung**: Verstehen Sie das Konzept der passiven Adressierung für eine effiziente Tastenfeld-Eingabelese.
3. **Verkabelungseinrichtung**: Entdecken Sie die Verkabelung mit Zeilen und Spalten für das Tastenfeld.
4. **Konfiguration der GPIO-Pins**: Richten Sie GPIO-Pins als Ausgänge für Zeilen und Eingänge für Spalten ein.
5. **Beschriftung der Tasten**: Erstellen Sie eine Matrix zur Beschriftung der Tasten auf dem Tastenfeld für eine einfachere Referenzierung.
6. **Implementierung**: Folgen Sie dem Codierungsprozess, um das Tastenfeld in Ihre Projekte zu integrieren.

Video

1.10.33 Video 33: Wie man ein Tastenfeld in ein Raspberry Pi-Projekt integriert

Dieses Video zeigt, wie man Tasteneingaben mit einem Raspberry Pi liest. Es behandelt die Schaltungseinrichtung, die Implementierung des Python-Codes und das Ansprechen des „haftenden Tasten"-Problems.

1. **Tastenfeld-Scanning-Methode**: Verständnis des Konzepts des Tastenfeld-Scannens mit Zeilen und Spalten.
2. **Schaltungseinrichtung**: Konfiguration der GPIO-Pins für Zeilen und Spalten, einschließlich der Verwendung interner Pull-Down-Widerstände.
3. **Implementierung des Python-Codes**: Schritt-für-Schritt-Anleitung zum Lesen von Tasteneingaben mit Python-Code durch Aktivierung von Zeilen und Lesen von Spalten.
4. **„Haftende Taste"-Problem**: Identifizierung und Lösung des Problems wiederholter Tastenanschläge.

Video

1.10.34 Video 34: Creating a Python Class and Library for Reading a Keypad

In diesem Tutorial geht es um die Erstellung einer Python-Klasse zum Lesen von Eingaben von einem Tastenfeld, das mit einem Raspberry Pi verbunden ist. Es beginnt mit einer Einführung in die Raspberry Pi-Tutorialreihe von SunFounder, gefolgt von der Erklärung der Hardware-Einrichtung und den GPIO-Verbindungen des Tastenfelds. Dann wird die schrittweise Erstellung der „Tastenfeld"-Klasse zur Verwaltung der Funktionalität beschrieben, einschließlich der Initialisierungsmethoden und GPIO-Pin-Konfiguration. Die Methode zum Lesen von Eingaben durch Zeilen- und Spaltenscan wird implementiert. Außerdem werden Code-Logik, die Erkennung von Tastendrücken, die Verkettung von Eingabezeichen und die Erstellung einer wiederverwendbaren Python-Bibliothek diskutiert. Potenzielle Anwendungen außerhalb des Tastenfelds, wie die Integration mit LCD-Displays und Kameras, werden ebenfalls behandelt.

1. Einführung in SunFounders Raspberry Pi-Tutorialreihe.
2. Erläuterung der Tastenfeld-Hardwareverbindungen über GPIO-Pins.
3. Schritt-für-Schritt-Anleitung zur Erstellung einer „Tastenfeld"-Klasse.
4. Initialisierung des Tastenfeldobjekts mit Standardparametern.
5. GPIO-Pin-Konfiguration innerhalb der Klasse.
6. Methode zum Lesen von Eingaben vom Tastenfeld.
7. Verständnis der Logik beim Aufbau eines Tastenfeldeingabesystems.
8. Techniken zur Tastendruckerkennung und Vermeidung von Wiederholungen.
9. Verkettung von Eingabezeichen zur Bildung eines Strings.
10. Konfiguration des Rückgabezeichens für Tastenfeldeingaben.
11. Erstellung einer wiederverwendbaren Python-Bibliothek für Tastenfelder.
12. Erkundung zusätzlicher Anwendungen wie LCD-Display- und Kameraintegration.

Video

1.10.35 Video 35: Mobile Raspberry Pi-Projekte mit Tastenfeld und LCD erstellen

In diesem Tutorial lernen Sie, wie Sie ein Raspberry Pi-Alarmsystem mit einem Tastenfeld zur Eingabe und einem LCD-Display zur Ausgabe erstellen und dabei Python-Codierungstechniken für einen effizienten Programmablauf verwenden.

1. Projektvorstellung: Überblick über den Bau eines Raspberry Pi-Alarmsystems.
2. Komponenteneinrichtung: Einrichten der Tastenfeld- und LCD-Anzeigekomponenten.
3. Python-Codierung: Schreiben von Python-Code zur Interaktion mit dem Tastenfeld und dem LCD.
4. Thread-Konzept: Diskussion über die Verwendung von Threads für einen verbesserten Programmablauf.
5. Alarmzustände: Erklärung verschiedener Zustände (bewaffnet, entwaffnet, Passwortänderung) im Alarmsystem.
6. Hausaufgaben: Zuweisung einer Aufgabe zur Implementierung des Programmablaufs für verschiedene Zustände und zur Passwortüberprüfung.

Video

1.10.36 Video 36: Überwachung der Benutzereingabe vom Tastenfeld mit Threading

In diesem Video wird gezeigt, wie man ein Alarmsystem mit einem Raspberry Pi erstellt, das Tastenfeldeingaben und ein LCD-Display integriert. Das Video behandelt verschiedene Aspekte wie Verkabelungsanweisungen, Threading für Multitasking, Befehlsverarbeitung, Passwortvalidierung, Fehlerbehandlung und Funktionalitätserweiterung. McCarter betont die Bedeutung von Portabilität und Einsatz für Anwendungen in der realen Welt über den Desktop hinaus.

1. **Portabilität und Einsatz**: Verlagerung von Raspberry Pi-Projekten über den Desktop hinaus für Anwendungen in der realen Welt.
2. **Tastenfeld- und LCD-Einrichtung**: Verdrahtung eines Tastenfelds und eines LCD-Displays an die GPIO-Pins des Raspberry Pi für Ein- und Ausgabe.
3. **Threading für Multitasking**: Verwendung von Threading zur Handhabung von Eingaben vom Tastenfeld während der Ausführung anderer Aufgaben.
4. **Integration von Tastenfeldeingaben**: Einbindung von Tastenfeldeingaben in das Projekt mithilfe einer Bibliothek.
5. **Befehlsverarbeitung**: Definition und Verarbeitung von Befehlen zum Bewaffnen, Entwaffnen und Ändern des Passworts.
6. **Passwortvalidierung**: Implementierung der Passwortvalidierung für die Systemsicherheit.
7. **LCD-Display-Schnittstelle**: Interaktion mit einem LCD-Display zur Bereitstellung von Benutzerfeedback und -hinweisen.
8. **Fehlerbehandlung und Debugging**: Techniken zur Identifizierung und Behebung von Fehlern während der Entwicklung.
9. **Funktionalitätserweiterung**: Vorschläge zur Erweiterung des Projekts, einschließlich der Integration zusätzlicher Sensoren und akustischer Alarme.

Video

1.10.37 Video 37: Bewegungserkennungs-Alarmsystem mit LCD und Tastenfeld

Dieses Tutorial zeigt die Erstellung eines Alarmsystems mit einem Raspberry Pi, das Tastenfeldeingaben und ein LCD-Display integriert. Es bietet schrittweise Anleitungen zur Verkabelung der Komponenten, zur Konfiguration von Bibliotheken und zur Implementierung von Funktionen wie Bewaffnung, Entwaffnung und Passwortänderung. Das Video betont die Portabilität und den Einsatz in der realen Welt.

1. **Portabilität und Einsatz**: Raspberry Pi-Projekte für die reale Welt optimieren.
2. **Tastenfeld- und LCD-Einrichtung**: Verdrahtung von Tastenfeld und LCD-Display.
3. **Threading für Multitasking**: Nutzung von Threads für Tastenfeldeingaben und andere Aufgaben.
4. **Bibliotheksintegration**: Einbindung und Nutzung relevanter Bibliotheken.
5. **Programmfluss**: Implementierung von Bewaffnungs-, Entwaffnungs- und Passwortänderungsmodi.
6. **Fehlerbehandlung**: Mechanismen zur Fehlerbehebung und Programmunterbrechung.
7. **Programmstruktur**: Aufbau von Threads und Hauptprogrammschleifen.
8. **Tastenfeldeingaben**: Verarbeitung von Befehlen und Passwortvalidierung.
9. **Alarmfunktionalität**: Bewegungserkennung und Benutzerfeedback.
10. **Benutzerinteraktion**: Anzeige von Status und Warnungen auf dem LCD.
11. **Fehlerbehandlung und Aufräumen**: Sicheres Beenden des Programms.
12. **Zuschauerherausforderung**: Verbesserung des Systems durch zusätzliche Funktionen.

Video

1.10.38 Video 38: Raspberry Pi Sicherheitssystem

Erfahren Sie, wie Sie Ihr tragbares Raspberry Pi Alarmsystem durch das Hinzufügen eines akustischen Alarms mithilfe von Python-Code und der pygame-Bibliothek verbessern können.

1. **Entwicklung des tragbaren Projekts**: Das Video ist Teil einer Serie, die den Zuschauern zeigt, wie sie um den Raspberry Pi herum einsetzbare Projekte erstellen können.
2. **Hinzufügen eines akustischen Alarms**: Demonstriert das Hinzufügen eines akustischen Alarms zum Alarmsystem mithilfe der pygame-Bibliothek zum Importieren und Abspielen eines MP3-Alarmklangs.
3. **Implementierungsdetails**: Bietet schrittweise Anleitungen zum Laden und Abspielen des Alarmklangs, wenn ein Eindringling vom PIR-Sensor erkannt wird.
4. **Anpassungsoptionen**: Bietet Anleitung zum Beschaffen und Speichern verschiedener Alarmtöne für eine verbesserte Vielfalt im Alarmsystem.
5. **Hausaufgaben**: Fordert die Zuschauer heraus, mehrere Alarmoptionen zu programmieren, um das Alarmsystem weiter zu verbessern.

Video

1.10.39 Video 39: Raspberry Pi Sicherheitssystem

In diesem Raspberry Pi Tutorial erfahren Sie, wie Sie Alarmtöne für ein Heim-Sicherheitssystem mit einem PIR-Bewegungssensor, einem Tastenfeld und einem LCD-Display anpassen können.

1. Einführung in die von SunFounder gesponserte Tutorialreihe, die sich auf Raspberry Pi-Projekte konzentriert.
2. Übersicht über den Aufbau eines Heim-Sicherheitssystems mit Raspberry Pi, einschließlich der Integration eines LCD-Displays, eines PIR-Bewegungssensors und eines Tastenfelds.
3. Erläuterung des Python-Codes für die Aktivierung, Deaktivierung, Passwortänderung und Auswahl von Alarmtönen.
4. Schritt-für-Schritt-Anleitung zum Anschließen und Konfigurieren der Hardwarekomponenten wie Tastenfeld, LCD und PIR-Sensor.
5. Demonstration der Programmierung mehrerer Alarmtöne basierend auf Benutzereingaben sowie Tipps zur Fehlerbehebung bei häufig auftretenden Fehlern.

Video

1.10.40 Video 40: Führen Sie Ihr Sicherheitsprojekt mit Putty remote aus

Erfahren Sie, wie Sie auf Ihr Raspberry Pi über SSH und Putty remote zugreifen und es steuern können, um Projekte unabhängig von einer Desktop-Umgebung zu entwickeln.

1. **SSH auf dem Raspberry Pi aktivieren**: Aktivieren Sie SSH auf Ihrem Raspberry Pi über raspi-config, um den Remotezugriff zu ermöglichen.
2. **Ermitteln der IP-Adresse des Raspberry Pi**: Lernen Sie, wie Sie die IP-Adresse Ihres Raspberry Pi mithilfe von ifconfig finden und erwägen Sie die Vergabe einer statischen IP-Adresse für konsistenten Remotezugriff.
3. **Herunterladen und Installieren von Putty**: Laden Sie Putty auf Ihren PC herunter und installieren Sie es, um SSH-Verbindungen zu Ihrem Raspberry Pi herzustellen.
4. **Verbindung zum Raspberry Pi mit Putty herstellen**: Stellen Sie über Putty eine Remoteverbindung zu Ihrem Raspberry Pi über SSH her, melden Sie sich mit Ihren Anmeldeinformationen an und führen Sie Befehle remote aus.
5. **Remote-Steuerung des Raspberry Pi**: Führen Sie Programme auf Ihrem Raspberry Pi remote aus und interagieren Sie mit ihnen über Putty, um die Praktikabilität des Remotezugriffs für die Projektbereitstellung zu demonstrieren.

Video

1.10.41 Video 41: So richten Sie einen Remote-Desktop auf dem Raspberry Pi mit VNC ein

Erfahren Sie, wie Sie den Remote-Desktop-Zugriff auf Ihrem Raspberry Pi einrichten, um die Steuerung von einem Windows-PC aus ohne physische Peripheriegeräte zu ermöglichen.

1. Einführung in die Raspberry Pi-Tutorialreihe, gesponsert von SunFounder.
2. Erklärung der Notwendigkeit des Remote-Desktop-Zugriffs auf den Raspberry Pi, um physische Peripheriegeräte zu vermeiden.
3. Überblick über die Verwendung von SSH über PuTTY für den Remote-Terminalzugriff.
4. Einführung in VNC (Virtual Network Computing) für eine vollständige Desktop-Benutzeroberfläche.

5. Anleitung zur Aktivierung von VNC auf dem Raspberry Pi und zur Konfiguration der Anzeigeoptionen.
6. Bedeutung der Kenntnis der IP-Adresse des Raspberry Pi für die Remote-Verbindung.

Video

1.10.42 Video 42: Datenübertragung zum PC über WiFi oder Ethernet unter Verwendung von UDP

Das Video zeigt, wie eine Client-Server-Kommunikation zwischen einem Raspberry Pi und einem PC mithilfe von Python und UDP (User Datagram Protocol) eingerichtet wird. Es behandelt die Erstellung von Server- und Client-Programmen, die Initialisierung und Bindung von Sockets, den Datenaustausch und die Implementierung grundlegender Funktionalitäten.

1. **Einführung in die Client-Server-Kommunikation**: Verständnis für die Notwendigkeit saubererer Remote-Verbindungen.
2. **Erstellung des Server-Programms**: Schritt-für-Schritt-Anleitung zur Erstellung eines Server-Programms auf dem Raspberry Pi mit Pythons Socket-Bibliothek.
3. **Initialisierung und Bindung von Sockets**: Erlernen der Initialisierung und Bindung von Sockets für die Kommunikation.
4. **Datenaustausch**: Erforschung des Prozesses des Sendens und Empfangens von Daten zwischen dem PC und dem Raspberry Pi.
5. **Erstellung des Client-Programms**: Entdecken, wie ein Client-Programm auf der PC-Seite erstellt wird, um mit dem Raspberry Pi-Server zu kommunizieren.
6. **Setup der Serveradresse**: Angabe der Serveradresse (IP des Raspberry Pi) im Client-Code.
7. **Erstellung des UDP-Clients**: Verwendung der Socket-Bibliothek zur Erstellung eines UDP-Clients auf der Client-Seite.
8. **Initiierung der Kommunikation**: Initiierung der Kommunikation durch Senden von Anfragen vom Client an den Server.
9. **Behandlung von Anfragen auf dem Server**: Verarbeitung von Client-Anfragen und Senden von Antworten auf dem Server.
10. **Daten-Decodierung**: Decodierung empfangener Daten unter Verwendung der UTF-8-Kodierung für die Lesbarkeit.
11. **Implementierung grundlegender Funktionalitäten**: Demonstration der bidirektionalen Kommunikation anhand eines einfachen Beispiels zur Manipulation von Zählern.

Video

1.10.43 Video 43: Wie man ein einfaches Client-Server-System mit dem Raspberry Pi erstellt

Erfahren Sie, wie Sie ein Client-Server-Kommunikationssystem mit dem Raspberry Pi aufbauen, bei dem der Raspberry Pi als Server fungiert und Temperatur- und Feuchtigkeitsmessungen durchführt, nachdem er Befehle von einem PC-Client erhalten hat.

1. **Einrichten des Raspberry Pi-Servers**: Konfiguration der GPIO-Pins, Erstellung eines UDP-Server-Sockets.
2. **Behandlung von Client-Anfragen**: Kontinuierliches Lauschen auf Befehle, Decodieren und Parsen von Daten.
3. **Sensorauslesungen**: Erfassen von Temperatur- und Feuchtigkeitsauslesungen vom DHT11-Sensor.

4. **Datenaustausch**: Senden von Sensorinformationen zurück an den Client zur Anzeige.
5. **Implementierung auf Client-Seite**: Einrichten des Client-Sockets, Senden von Befehlen, Empfangen und Verarbeiten von Daten.

Video

1.10.44 Video 44: Aufbau einer verbesserten Client-Server-Verbindung zum Pi

Dieses Tutorial bietet eine umfassende Anleitung zum Aufbau einer ausgefeilteren Client-Server-Beziehung mit dem Raspberry Pi. Es ermöglicht dem PC, Befehle für Temperatur- oder Feuchtigkeitsmessungen zu senden und entsprechende Datenpakete zu empfangen. Das Tutorial behandelt Fehlerbehandlung, Datenübertragung und zukünftige Lektionspläne.

1. **Einführung**: Gesponserte Tutorial-Serie von SunFounder für Raspberry Pi-Anwendungen.
2. **Vorherige Lektionen**: Überblick über die Datenübertragung zwischen Raspberry Pi und PC mit UDP.
3. **Einfache Client-Server-Beziehung**: Erklärung des grundlegenden Aufbaus, der in vorherigen Lektionen demonstriert wurde.
4. **Ausgefeilte Client-Server-Beziehung**: Fokus auf das Abfragen von Benutzerbefehlen und die Bearbeitung von Datenantworten.
5. **Datenpräzision**: Wichtigkeit der Kennzeichnung von Antworten für genaue Datenübertragung.
6. **Einrichtung des Raspberry Pi-Servers**: Schritt-für-Schritt-Anleitung zur Serverkonfiguration.
7. **Fehlerbehandlung**: Effektive Behandlung ungültiger Befehle oder Anfragen.
8. **Datenübertragung**: Kodierung und Dekodierung von Daten für nahtlose Kommunikation.
9. **Serverkonfiguration**: Konfigurieren des Servers für Befehlsempfang und -antwort.
10. **Konfiguration auf Clientseite**: Einrichten des Clients für Befehlssendung und Datenempfang.
11. **Robustheitstest**: Stabilitätstest durch wiederholtes Senden von Befehlen.
12. **Zukünftige Lektionen**: Einblicke in bevorstehende Lektionen zur Raspberry Pi-Kamera und Buster-Betriebssystemkonfiguration.

Video

1.10.45 Video 69: Einsatz des MFRC522 RFID-Moduls und -Tags am Raspberry Pi

Lernen Sie, wie Sie ein RFID-Modul in Ihre Raspberry Pi-Projekte für Aufgaben wie Zugangskontrolle und Anwesenheitsverfolgung integrieren können, mit schrittweisen Anleitungen und Python-Coding-Demonstrationen.

1. **Einführung in das RFID-Modul**: Verstehen Sie, wie RFID-Technologie Raspberry Pi-Projekte durch Interaktion mit RFID-Karten oder Schlüsselanhängern bereichern kann.
2. **Hardware einrichten**: Folgen Sie uns bei der Verbindung des MFRC522 RFID-Moduls mit den GPIO-Pins des Raspberry Pi unter Verwendung eines bereitgestellten Breakout-Boards.
3. **SPI auf dem Raspberry Pi einrichten**: Aktivieren Sie die SPI-Schnittstelle und installieren Sie die notwendigen Bibliotheken für die Kommunikation zwischen Raspberry Pi und dem RFID-Modul.
4. **Interaktion mit dem RFID-Modul**: Lernen Sie, wie Sie Python-Code schreiben, um das RFID-Lesegerät zu initialisieren, Karten-IDs zu lesen und Textdaten mit IDs zu verknüpfen.
5. **RFID-Tags lesen und beschreiben**: Sehen Sie Demonstrationen zum Lesen und Beschreiben von Daten auf RFID-Tags mit Python-Code, einschließlich praktischer Tests.

6. **Hausaufgabe**: Erhalten Sie die Aufgabe, ein System zu erstellen, um den Namen und die Lieblingsfarbe einer Person mit RFID-Tags aufzuzeichnen, um Ihr Verständnis für die RFID-Integration in Raspberry Pi-Projekte zu erweitern.

Video

1.10.46 Video 70: Einsatz des MFRC522 RFID-Moduls und -Tags am Raspberry Pi

Das Video erörtert fortgeschrittene Techniken zur Programmierung von RFID/NFC-Karten mit einem Raspberry Pi. Es behandelt Themen wie Datenbündelung, Benutzereingaben, Hardware-Interaktion und Fehlerprüfung. Das Tutorial bietet Einblicke in effizientes Datenmanagement, Benutzeraufforderungen und Debugging-Techniken für die Implementierung von Admin-Funktionalitäten in RFID/NFC-Kartenprogrammierprojekten.

1. Einführung in das von SunFounder gesponserte Raspberry Pi-Kit.
2. Einrichtung des RC522 RFID-Moduls und einer RGB-LED.
3. Nutzung von Wörterbüchern für effizientes Datenmanagement.
4. Implementierung von Benutzeraufforderungen und -antworten.
5. Admin-Funktionalität zum Programmieren neuer RFID/NFC-Karten.
6. Datenbündelung: Verwendung eines Doppelpunkts als Trennzeichen zur effizienten Datenbündelung.
7. Korrektes Formatieren und Apostrophbehandlung bei Benutzereingaben.
8. Schreiben von personalisierten Daten auf NFC-Karten.
9. Steuern von LEDs basierend auf den von NFC-Karten gelesenen Daten.
10. Bedeutung der Fehlerprüfung und Techniken für die Fehlerbehandlung.
11. Hausaufgabe: Implementierung der Fehlerprüfung für Farbeingaben und dynamische Aktualisierung des Wörterbuchs im Admin-Bereich des Programms.

Video

1.11 Spielen Sie mit Python (für alle Pi-Modelle)

Wenn Sie ein Raspberry Pi 5 Python-Benutzer sind, können Sie die GPIOs mit der von `GPIO.Zero` bereitgestellten API programmieren.

1.11.1 Überprüfen Sie `GPIO Zero`

Wenn Sie ein Raspberry Pi 5 Python-Nutzer sind, können Sie GPIOs mit der von `GPIO Zero` bereitgestellten API programmieren.

`GPIO Zero` ist ein Modul zur Steuerung der GPIO-Pins des Raspberry Pi. Dieses Paket bietet eine Reihe benutzerfreundlicher Klassen und Funktionen zur Steuerung der GPIOs auf einem Raspberry Pi. Für Beispiele und Dokumentationen besuchen Sie: https://gpiozero.readthedocs.io/en/latest/.

Um zu testen, ob GPIO Zero installiert ist, geben Sie in Python ein:

```
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.
>>>
```

Im Python-CLI, geben Sie `import gpiozero` ein. Wenn keine Fehlermeldung erscheint, bedeutet dies, dass GPIO Zero installiert ist.

```
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
>>>
```

Um die Python-CLI zu verlassen, tippen Sie:

```
exit()
```

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

1.11.2 1 Ausgabe

1.1 Anzeigen

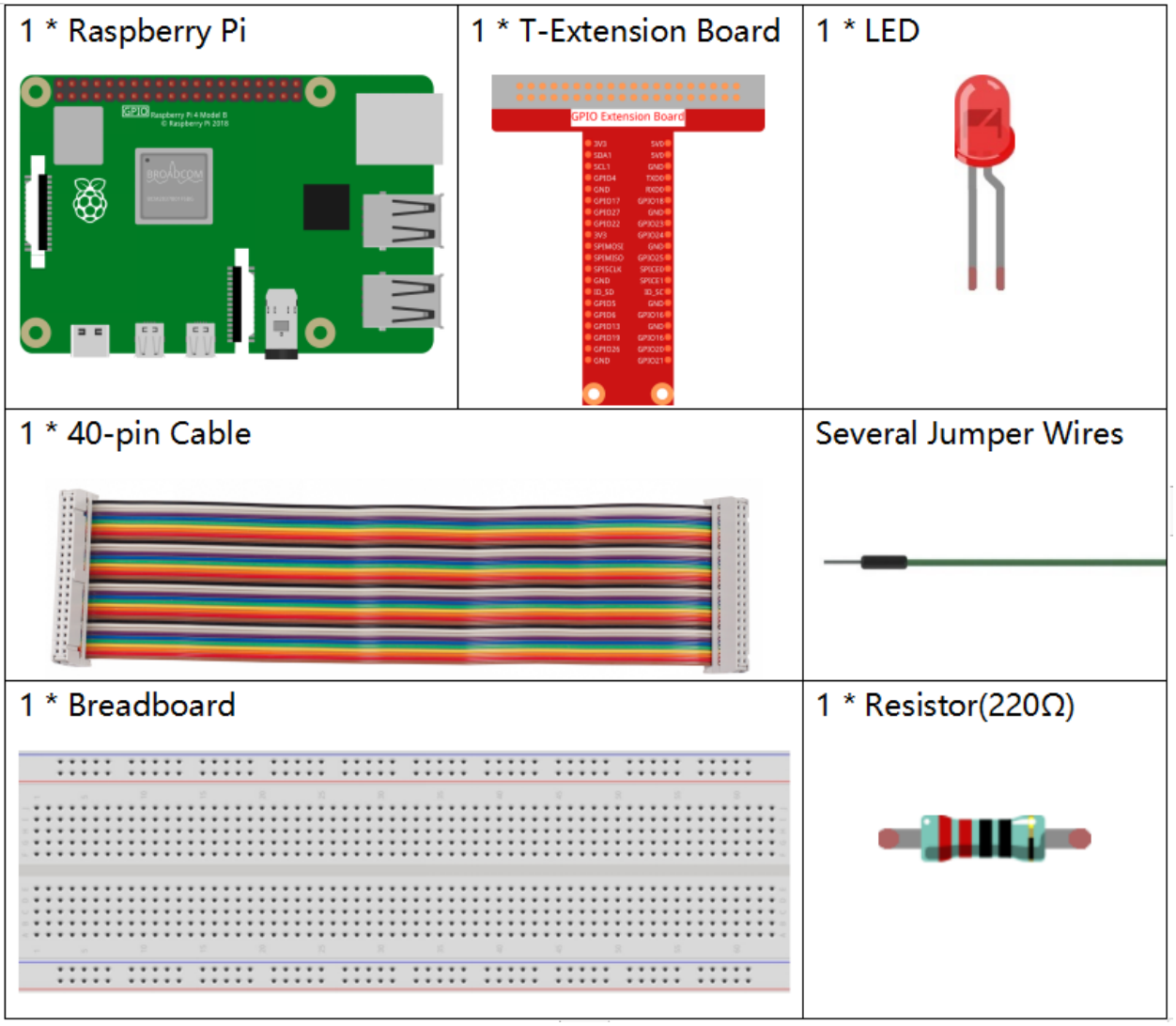
1.1.1 Blinkende LED

Einführung

In diesem Projekt lernen wir, wie man eine LED blinken lässt, indem wir sie programmieren. Durch Ihre Einstellungen kann Ihre LED eine Reihe von interessanten Phänomenen erzeugen. Jetzt legen Sie los.

Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten:



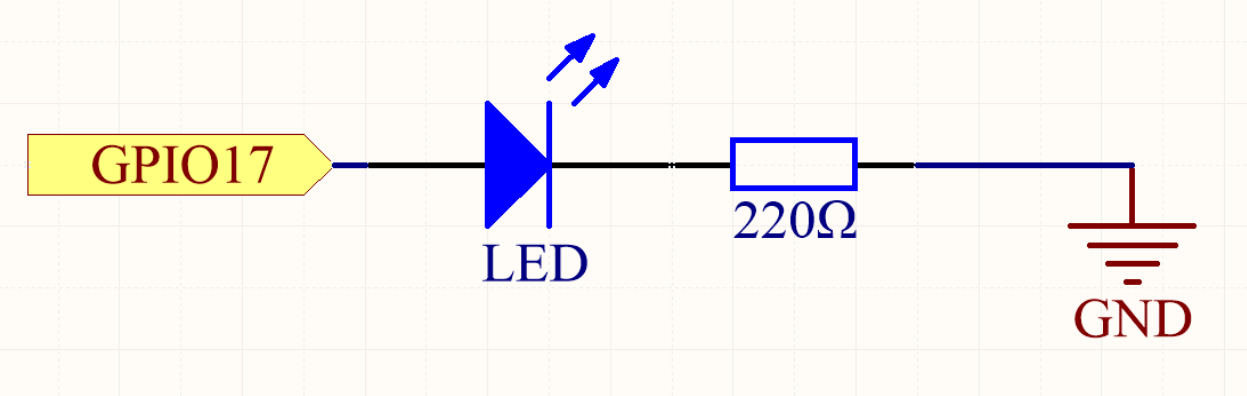
Schaltplan

In diesem Experiment verbinden wir den GPIO-Pin 17 des Raspberry Pi mit der Anode (langer Pin) der LED, dann die Kathode (kurzer Pin) der LED mit einem Widerstand und schließlich das andere Ende des Widerstands mit dem GND-Pin des Raspberry Pi. Um die LED einzuschalten, müssen wir GPIO17 auf hoch (3,3 V) setzen. Dieses Phänomen können wir durch Programmierung erreichen.

Bemerkung: **Pin11** bezieht sich auf den 11. Pin des Raspberry Pi von links nach rechts, und die entsprechenden **BCM**-Pinnummern sind in der folgenden Tabelle angegeben.

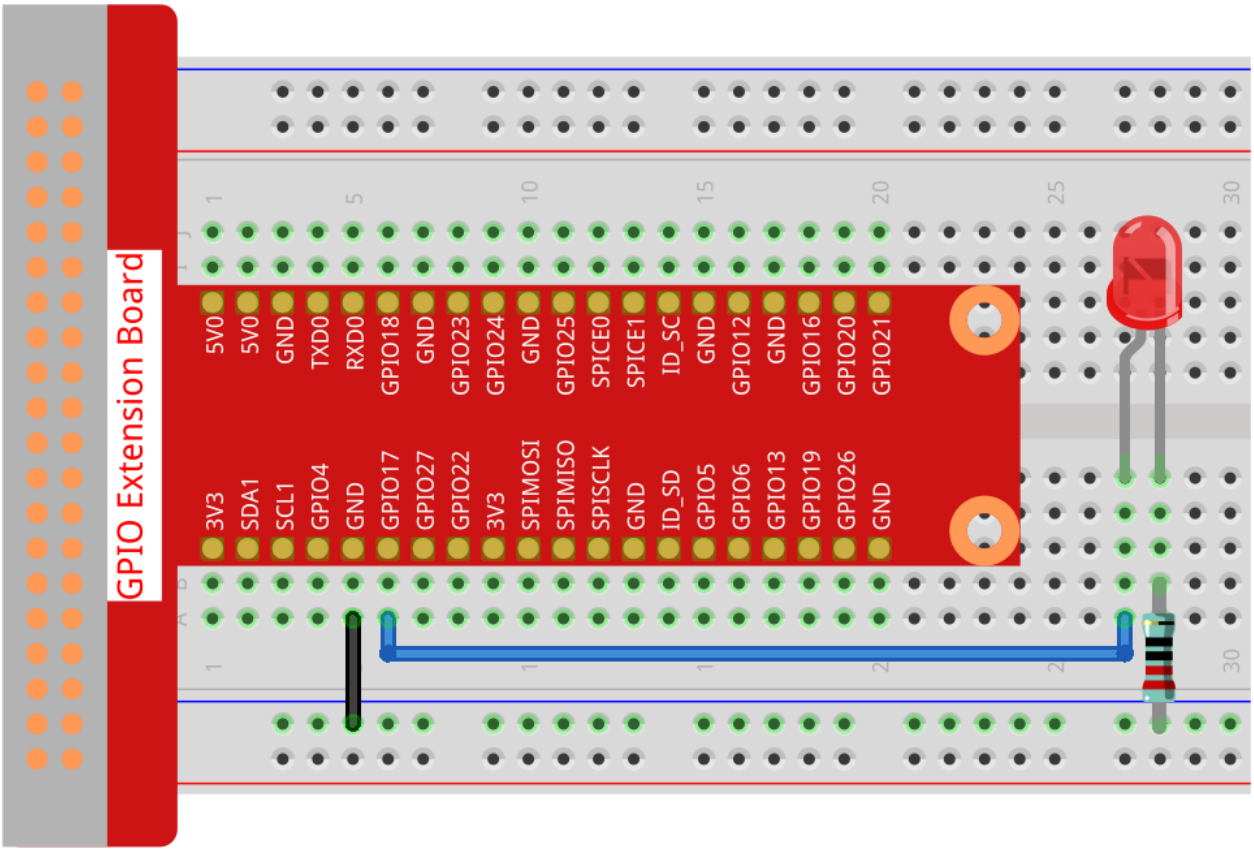
Unter den mit der Python-Sprache zusammenhängenden Inhalten ist BCM 17 die Nummer 17 in der Spalte BCM der folgenden Tabelle. Gleichzeitig entspricht dies dem 11. Pin am Raspberry Pi, Pin 11.

T-Board Name	physical	BCM
GPIO17	Pin 11	17



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Gehen Sie zum Ordner des Codes und führen Sie ihn aus.

1. Wenn Sie einen Bildschirm verwenden, wird empfohlen, die folgenden Schritte zu befolgen.

Finden Sie *1.1.1_BlinkingLed_zero.py* und doppelklicken Sie darauf, um es zu öffnen. Jetzt befinden Sie sich in der Datei.

Klicken Sie auf **Run ->Run Module** im Fenster, und die folgenden Inhalte erscheinen.

Um das Ausführen zu stoppen, klicken Sie einfach auf das X oben rechts, um es zu schließen und dann gelangen Sie zurück zum Code. Wenn Sie den Code ändern, müssen Sie ihn zuerst speichern, bevor Sie auf **Run Module (F5)** klicken. Dann können Sie die Ergebnisse sehen.

2. Wenn Sie sich remote in den Raspberry Pi einloggen, geben Sie den Befehl ein:

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Bemerkung: Wechseln Sie über `cd` in das Verzeichnis des Codes in diesem Experiment.

Schritt 3: Führen Sie den Code aus

```
sudo python3 1.1.1_BlinkingLed_zero.py
```

Bemerkung: Hier bedeutet sudo - superuser do, und python bedeutet, die Datei mit Python auszuführen.

Nachdem der Code ausgeführt wurde, sehen Sie, wie die LED blinkt.

Schritt 4: Wenn Sie die Code-Datei *1.1.1_BlinkingLed_zero.py* bearbeiten möchten, drücken Sie `Ctrl + C`, um die Ausführung des Codes zu stoppen. Geben Sie dann den folgenden Befehl ein, um *1.1.1_BlinkingLed_zero.py* zu öffnen:

```
nano 1.1.1_BlinkingLed_zero.py
```

Bemerkung: nano ist ein Texteditor-Tool. Der Befehl wird verwendet, um die Code-Datei *1.1.1_BlinkingLed_zero.py* mit diesem Tool zu öffnen.

Drücken Sie `Ctrl+X`, um zu beenden. Wenn Sie den Code geändert haben, erscheint eine Aufforderung, ob die Änderungen gespeichert werden sollen oder nicht. Geben Sie `Y` (speichern) oder `N` (nicht speichern) ein.

Drücken Sie dann `Enter`, um zu beenden. Geben Sie nano *1.1.1_BlinkingLed_zero.py* erneut ein, um die Auswirkungen nach der Änderung zu sehen.

Code

Das Folgende ist der Programmkode:

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED
from time import sleep

# Initialisieren Sie eine LED, die mit GPIO-Pin 17 verbunden ist, unter Verwendung der␣
↪GPIO Zero-Bibliothek.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
led = LED(17)

try:
    # Beginnen Sie eine unendliche Schleife, um den LED-Zustand zu wechseln.
    while True:
        # Schalten Sie die LED ein und geben Sie eine Nachricht auf der Konsole aus.
        led.on()
        print('...LED EIN')

        # Warten Sie 0,5 Sekunden mit eingeschalteter LED.
        sleep(0.5)

        # Schalten Sie die LED aus und geben Sie eine Nachricht auf der Konsole aus.
        led.off()
        print('LED AUS...')

        # Warten Sie 0,5 Sekunden mit ausgeschalteter LED.
        sleep(0.5)

except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (Ctrl+C) auf elegante Weise, indem Sie die␣
↪Schleife beenden.
    # GPIO Zero kümmert sich beim Beenden automatisch um die Bereinigung der GPIO-
↪Einstellungen.
    pass
```

Code-Erklärung

1. Wenn das System dies erkennt, sucht es den Installationspfad von python in der env-Einstellung und ruft den entsprechenden Interpreter auf, um die Operation abzuschließen. Es dient dazu, zu verhindern, dass der Benutzer python nicht auf den Standardpfad `/usr/bin` installiert hat.

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

2. Diese Zeilen importieren die notwendigen Klassen und Funktionen. `LED` von `gpiozero` zur LED-Steuerung und `sleep` von `time` für Verzögerungen.

   ```
   from gpiozero import LED
   from time import sleep
   ```

3. Diese Zeile erstellt ein LED-Objekt, das mit dem GPIO-Pin 17 verbunden ist.

   ```
   # Initialisieren Sie eine LED, die mit GPIO-Pin 17 verbunden ist, unter Verwendung␣
   ↪der GPIO Zero-Bibliothek.
   led = LED(17)
   ```

4. Eine unendliche Schleife wird mit `while True:` gestartet. Innerhalb der Schleife wird die LED eingeschaltet (`led.on()`) und eine Nachricht wird ausgegeben. Das Programm pausiert dann für 0,5 Sekunden (`sleep(0.5)`). Danach wird die LED ausgeschaltet (`led.off()`), eine weitere Nachricht wird ausgegeben, und das Programm pausiert erneut für 0,5 Sekunden.

   ```
   try:
       # Beginnen Sie eine unendliche Schleife, um den LED-Zustand zu wechseln.
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    while True:
        # Schalten Sie die LED ein und geben Sie eine Nachricht auf der Konsole aus.
        led.on()
        print('...LED EIN')

        # Warten Sie 0,5 Sekunden mit eingeschalteter LED.
        sleep(0.5)

        # Schalten Sie die LED aus und geben Sie eine Nachricht auf der Konsole aus.
        led.off()
        print('LED AUS...')

        # Warten Sie 0,5 Sekunden mit ausgeschalteter LED.
        sleep(0.5)
```

5. Der `except`-Block fängt eine `KeyboardInterrupt`-Ausnahme (wie das Drücken von Ctrl+C) ab und beendet die Schleife sicher. Das `pass`-Statement wird hier als Platzhalter verwendet, um anzuzeigen, dass bei einer Unterbrechung keine spezifische Aktion unternommen wird.

```
except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (Ctrl+C) auf elegante Weise, indem␣
↪Sie die Schleife beenden.
    # GPIO Zero kümmert sich beim Beenden automatisch um die Bereinigung der GPIO-
↪Einstellungen.
    pass
```

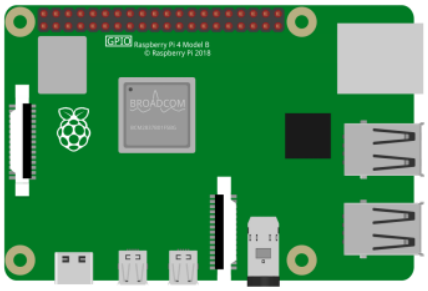
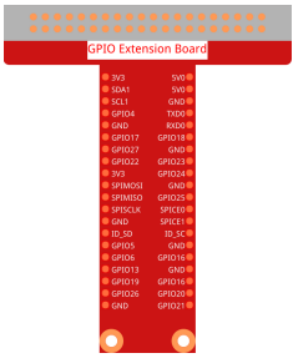
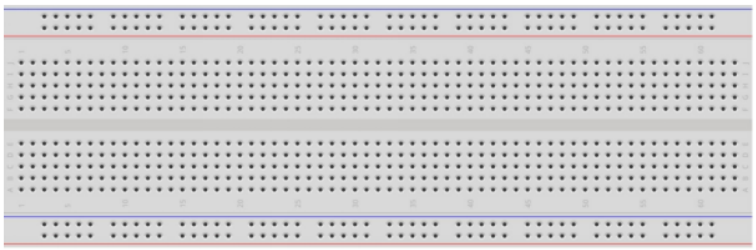
1.1.2 RGB-LED

Einführung

In diesem Projekt werden wir eine RGB-LED steuern, um verschiedene Farben blinken zu lassen.

Benötigte Komponenten

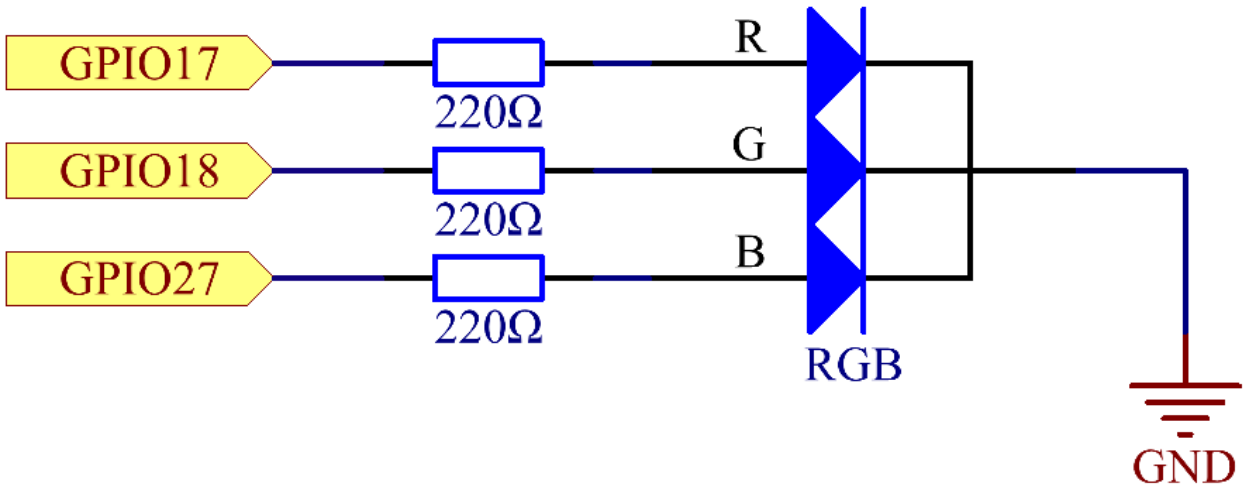
Für dieses Projekt benötigen wir die folgenden Komponenten.

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

Schaltplan

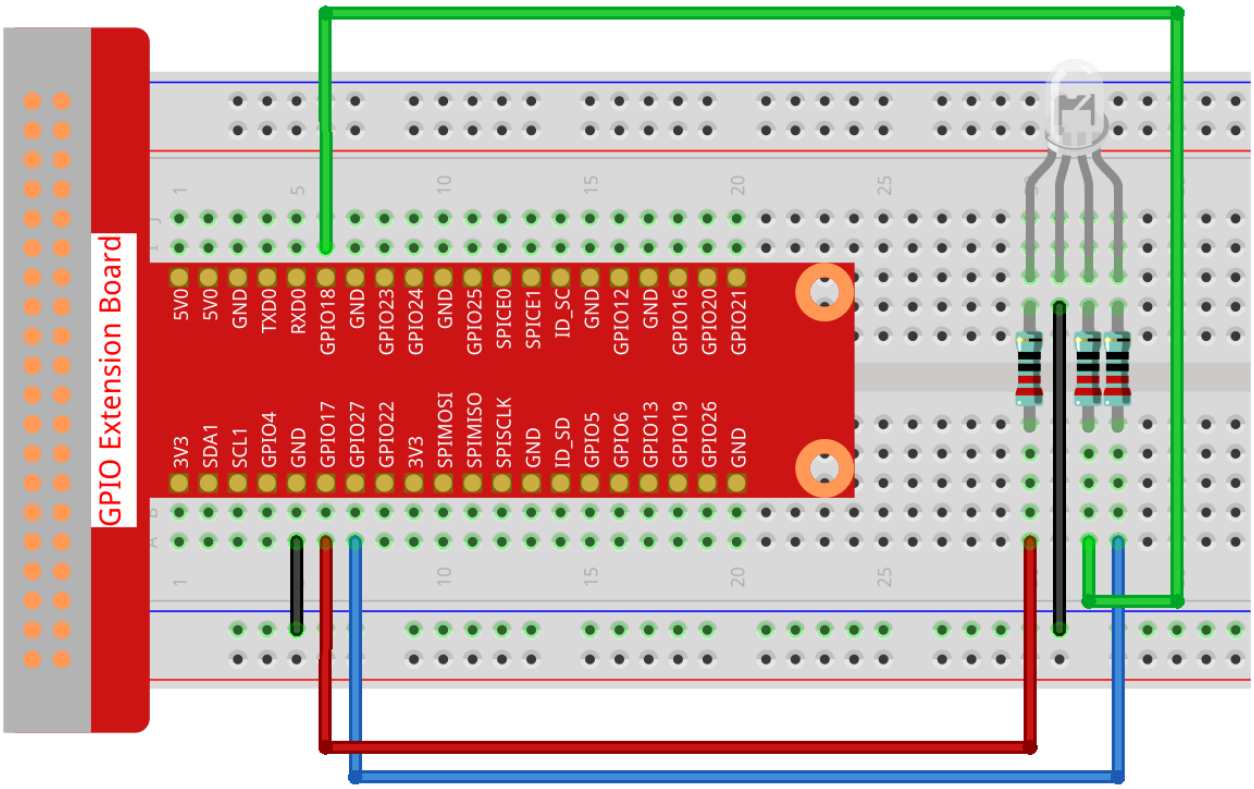
Nachdem die Pins von R, G und B mit einem Strombegrenzungswiderstand verbunden sind, verbinden Sie diese mit GPIO17, GPIO18 und GPIO27. Der längste Pin (GND) der LED verbindet sich mit dem GND des Raspberry Pi. Wenn den drei Pins unterschiedliche PWM-Werte gegeben werden, zeigt die RGB-LED verschiedene Farben an.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.1.2_rgbLed_zero.py
```

Nachdem der Code ausgeführt wurde, sehen Sie, dass RGB rot, grün, blau, gelb, rosa und cyan anzeigt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import RGBLED
from time import sleep

# Definieren Sie eine Liste von Farben für die RGB-LED im RGB-Format (Rot, Grün, Blau).
# Jede Farbkomponente reicht von 0 (aus) bis 1 (volle Intensität).
COLORS = [(1, 0, 0), (0, 1, 0), (0.2, 0.1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1)]

# Initialisieren Sie eine RGB-LED. Verbinden Sie die rote Komponente mit GPIO 17, grün␣
↪mit GPIO 18 und blau mit GPIO 27.
rgb_led = RGBLED(red=17, green=18, blue=27)

try:
    # Kontinuierlich durch die definierten Farben durchlaufen.
    while True:
        for color in COLORS:
            # Stellen Sie die RGB-LED auf die aktuelle Farbe ein.
            rgb_led.color = color
            # Geben Sie die aktuelle Farbe auf der Konsole aus.
            print(f"Farbe eingestellt auf: {color}")
            # Warten Sie 1 Sekunde, bevor Sie zur nächsten Farbe wechseln.
            sleep(1)

except KeyboardInterrupt:
    # Behandeln Sie ein KeyboardInterrupt (Ctrl+C) um die Schleife elegant zu beenden.
    # Die GPIO-Bereinigung wird automatisch von GPIO Zero bei Beendigung des Skripts␣
↪verwaltet.
    pass
```

Code-Erklärung

1. Hier werden die Klasse `RGBLED` aus der Bibliothek `gpiozero` zum Steuern einer RGB-LED und die Bibliothek `time` für Verzögerungen im Code importiert.

   ```
   #!/usr/bin/env python3
   from gpiozero import RGBLED
   from time import sleep
   ```

2. Die Liste `COLORS` enthält Tupel, die verschiedene Farben im RGB-Format darstellen. Durch Zuweisen unterschiedlicher Pulsweitenmodulations- (PWM-) Werte an jeden der R-, G- und B-Pins über das Attribut `rgb_led.color` kann die LED eine Vielzahl von Farben erzeugen. Die PWM-Werte reichen von 0 bis 1, wobei 0 keine Intensität (aus) und 1 volle Intensität für jede Farbkomponente darstellt.

3. Beispielsweise führt das Einstellen von `rgb_led.color = (1, 0, 0)` dazu, dass die LED rot leuchtet, da volle Intensität für die rote Komponente eingestellt wird, während Grün und Blau ausgeschaltet bleiben. Verschiedene Kombinationen dieser Werte führen zu unterschiedlichen Farben. Diese Technik der Farbmischung durch PWM

ermöglicht die Erstellung einer breiten Palette von Farben auf der RGB-LED.

```
COLORS = [(1, 0, 0), (0, 1, 0), (0.2, 0.1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1)]
```

4. Eine RGB-LED wird initialisiert, wobei ihre roten, grünen und blauen Komponenten jeweils an die GPIO-Pins 17, 18 und 27 angeschlossen sind.

```
# Initialisieren Sie eine RGB-LED. Verbinden Sie die rote Komponente mit GPIO 17,␣
↪grün mit GPIO 18 und blau mit GPIO 27.
rgb_led = RGBLED(red=17, green=18, blue=27)
```

5. Die Schleife `while True:` durchläuft kontinuierlich die in `COLORS` definierten Farben. Für jede Farbe setzt `rgb_led.color = color` die LED auf diese Farbe, und `sleep(1)` pausiert für 1 Sekunde.

```
try:
    # Kontinuierlich durch die definierten Farben durchlaufen.
    while True:
        for color in COLORS:
            # Stellen Sie die RGB-LED auf die aktuelle Farbe ein.
            rgb_led.color = color
            # Geben Sie die aktuelle Farbe auf der Konsole aus.
            print(f"Farbe eingestellt auf: {color}")
            # Warten Sie 1 Sekunde, bevor Sie zur nächsten Farbe wechseln.
            sleep(1)
```

6. Dieser Abschnitt behandelt ein `KeyboardInterrupt` (zum Beispiel das Drücken von Ctrl+C) auf elegante Weise. Das `pass`-Statement wird als Platzhalter verwendet, um anzuzeigen, dass bei einer Unterbrechung keine spezifische Aktion unternommen wird, da GPIO Zero die GPIO-Bereinigung automatisch handhabt.

```
except KeyboardInterrupt:
    # Behandeln Sie ein KeyboardInterrupt (Ctrl+C) um die Schleife elegant zu␣
↪beenden.
    # Die GPIO-Bereinigung wird automatisch von GPIO Zero bei Beendigung des␣
↪Skripts verwaltet.
    pass
```

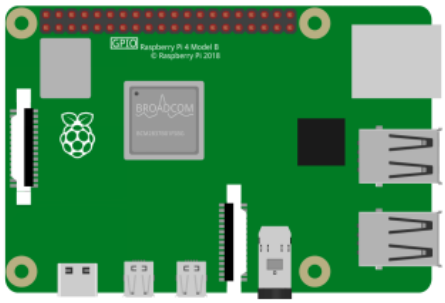
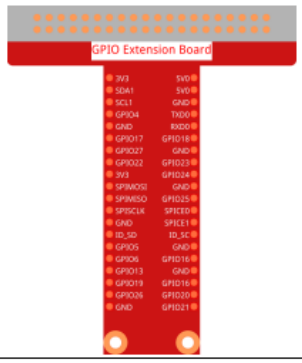
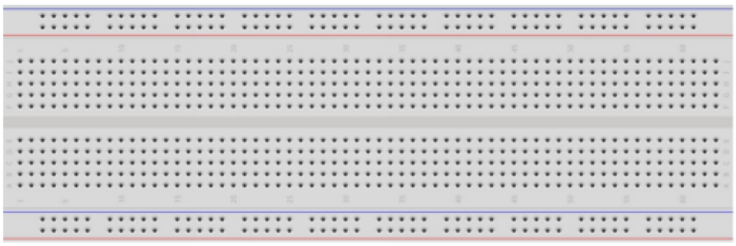
1.1.3 LED-Balkendiagramm

Einführung

In diesem Projekt beleuchten wir die Lichter des LED-Balkendiagramms nacheinander.

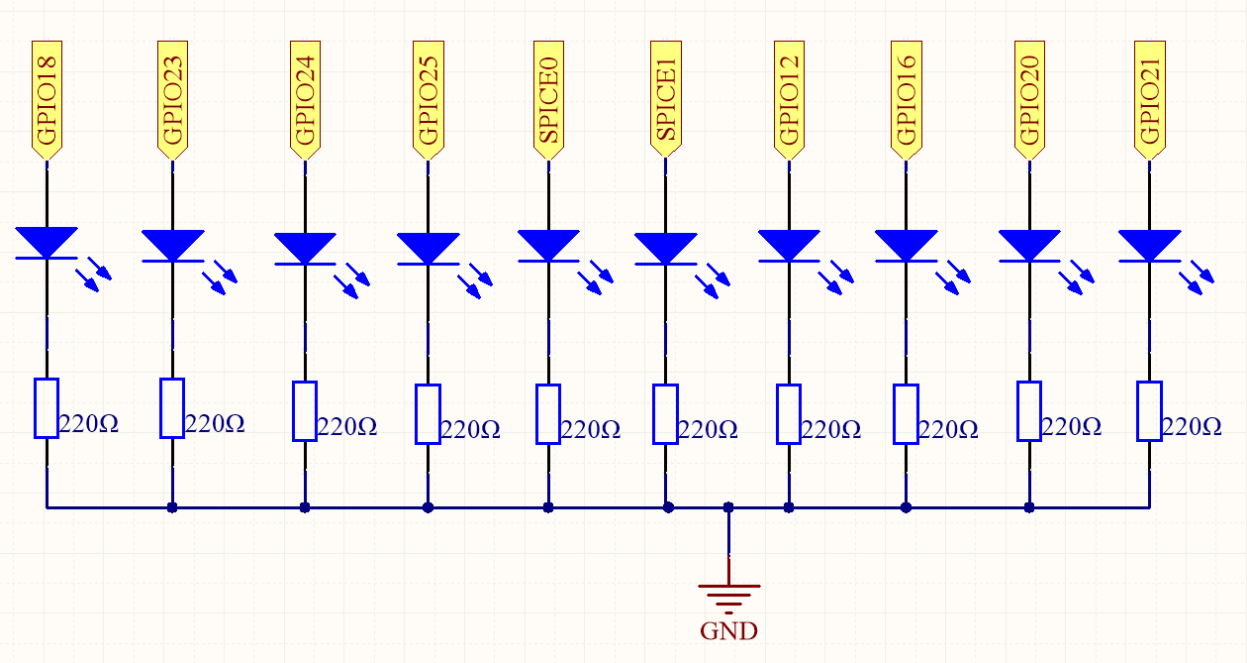
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

Schaltplan

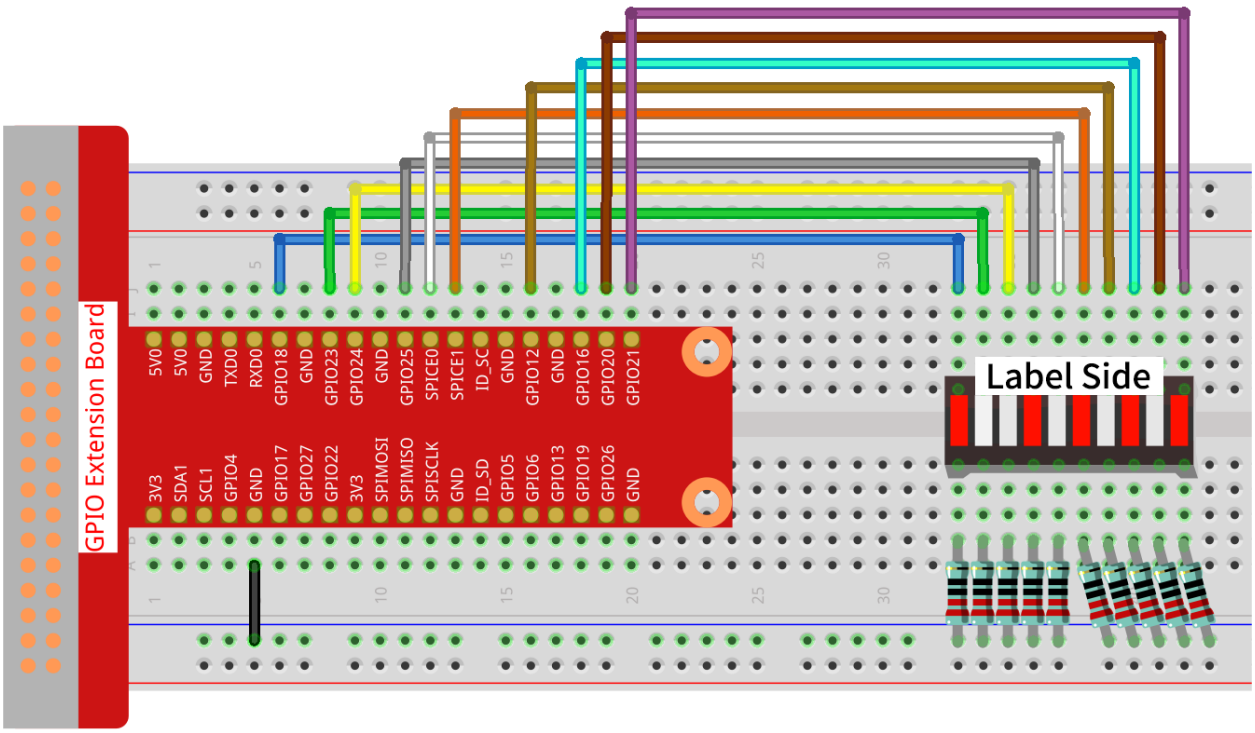
T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	Pin 26	11	7
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	22
GPIO21	Pin 40	29	21



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.

Bemerkung: Achten Sie auf die Richtung beim Anschließen. Wenn Sie es verkehrt herum anschließen, wird es nicht leuchten.



Schritt 2: Gehen Sie zum Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 1.1.3_LedBarGraph_zero.py
```

Nachdem der Code ausgeführt wurde, sehen Sie, dass die LEDs auf dem LED-Balken regelmäßig ein- und ausschalten.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED
from time import sleep

# Definieren Sie die GPIO-Pins, an denen die LEDs angeschlossen sind
led_pins = [18, 23, 24, 25, 8, 7, 12, 16, 20, 21]

# Erstellen Sie LED-Objekte für jeden Pin
leds = [LED(pin) for pin in led_pins]

def odd_led_bar_graph():
    # Nacheinander die ungerade nummerierten LEDs beleuchten (Index 0, 2, 4, usw.)
    for i in range(5):
        j = i * 2  # Ungerade Index berechnen
        leds[j].on()  # Ungerade nummerierte LED einschalten
        sleep(0.3)    # Verzögerung für visuellen Effekt
        leds[j].off() # LED ausschalten

def even_led_bar_graph():
    # Nacheinander die gerade nummerierten LEDs beleuchten (Index 1, 3, 5, usw.)
    for i in range(5):
        j = i * 2 + 1  # Gerade Index berechnen
        leds[j].on()   # Gerade nummerierte LED einschalten
        sleep(0.3)     # Verzögerung für visuellen Effekt
        leds[j].off()  # LED ausschalten

def all_led_bar_graph():
    # Nacheinander alle LEDs einzeln beleuchten
    for led in leds:
        led.on()       # LED einschalten
        sleep(0.3)     # Verzögerung für visuellen Effekt
        led.off()      # LED ausschalten

def turn_off_all_leds():
    # Alle LEDs auf einmal ausschalten
    for led in leds:
        led.off()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
try:
    # Hauptloop zum Durchlaufen der LED-Muster
    while True:
        odd_led_bar_graph()   # Ungerade nummerierte LEDs aktivieren
        sleep(0.3)            # Pause zwischen den Mustern
        even_led_bar_graph()  # Gerade nummerierte LEDs aktivieren
        sleep(0.3)            # Pause zwischen den Mustern
        all_led_bar_graph()   # Alle LEDs aktivieren
        sleep(0.3)            # Pause vor Neustart

except KeyboardInterrupt:
    # Unterbrechung (Ctrl+C) auf elegante Weise behandeln
    turn_off_all_leds()      # Sicherstellen, dass alle LEDs beim Beenden ausgeschaltet␣
↪sind
    pass
```

Code-Erklärung

1. Diese Zeilen importieren die notwendigen Klassen und Funktionen. `LED` aus `gpiozero` zur LED-Steuerung und `sleep` aus `time` für Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import LED
   from time import sleep
   ```

2. Die Liste `led_pins` enthält die GPIO-Pinnummern. `leds` ist eine Liste von `LED`-Objekten, von denen jedes einem Pin in `led_pins` entspricht.

   ```
   # Definieren Sie die GPIO-Pins, an denen die LEDs angeschlossen sind
   led_pins = [18, 23, 24, 25, 8, 7, 12, 16, 20, 21]

   # Erstellen Sie LED-Objekte für jeden Pin
   leds = [LED(pin) for pin in led_pins]
   ```

3. Lassen Sie die LED an der ungeraden Ziffer des LED-Balkendiagramms nacheinander aufleuchten.

   ```
   def odd_led_bar_graph():
       # Nacheinander die ungerade nummerierten LEDs beleuchten (Index 0, 2, 4, usw.)
       for i in range(5):
           j = i * 2  # Ungerade Index berechnen
           leds[j].on()  # Ungerade nummerierte LED einschalten
           sleep(0.3)    # Verzögerung für visuellen Effekt
           leds[j].off() # LED ausschalten
   ```

4. Lassen Sie die LED an der geraden Ziffer des LED-Balkendiagramms nacheinander aufleuchten.

   ```
   def even_led_bar_graph():
       # Nacheinander die gerade nummerierten LEDs beleuchten (Index 1, 3, 5, usw.)
       for i in range(5):
           j = i * 2 + 1  # Gerade Index berechnen
           leds[j].on()   # Gerade nummerierte LED einschalten
           sleep(0.3)     # Verzögerung für visuellen Effekt
           leds[j].off()  # LED ausschalten
   ```

5. Lassen Sie die LED am LED-Balkendiagramm nacheinander aufleuchten.

```
def all_led_bar_graph():
    # Nacheinander alle LEDs einzeln beleuchten
    for led in leds:
        led.on()       # LED einschalten
        sleep(0.3)     # Verzögerung für visuellen Effekt
        led.off()      # LED ausschalten
```

6. Die Schleife `while True` durchläuft kontinuierlich die LED-Muster. Der `except`-Block behandelt ein KeyboardInterrupt (Ctrl+C) und stellt sicher, dass alle LEDs beim Beenden ausgeschaltet sind.

```
try:
    # Hauptloop zum Durchlaufen der LED-Muster
    while True:
        odd_led_bar_graph()   # Ungerade nummerierte LEDs aktivieren
        sleep(0.3)            # Pause zwischen den Mustern
        even_led_bar_graph()  # Gerade nummerierte LEDs aktivieren
        sleep(0.3)            # Pause zwischen den Mustern
        all_led_bar_graph()   # Alle LEDs aktivieren
        sleep(0.3)            # Pause vor Neustart

except KeyboardInterrupt:
    # Unterbrechung (Ctrl+C) auf elegante Weise behandeln
    turn_off_all_leds()      # Sicherstellen, dass alle LEDs beim Beenden␣
↪ausgeschaltet sind
    pass
```

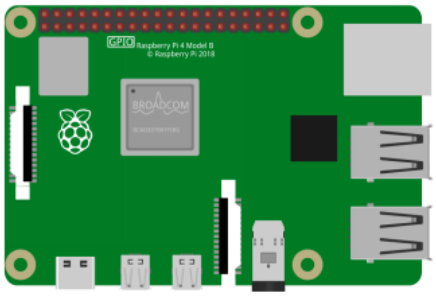
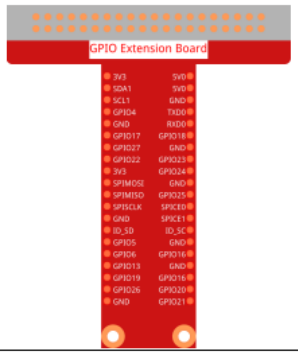
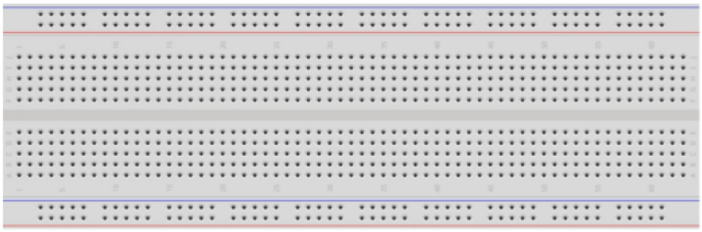
1.1.4 7-Segment-Anzeige

Einführung

Lassen Sie uns versuchen, eine 7-Segment-Anzeige anzusteuern, um Ziffern von 0 bis 9 und Buchstaben von A bis F anzuzeigen.

Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

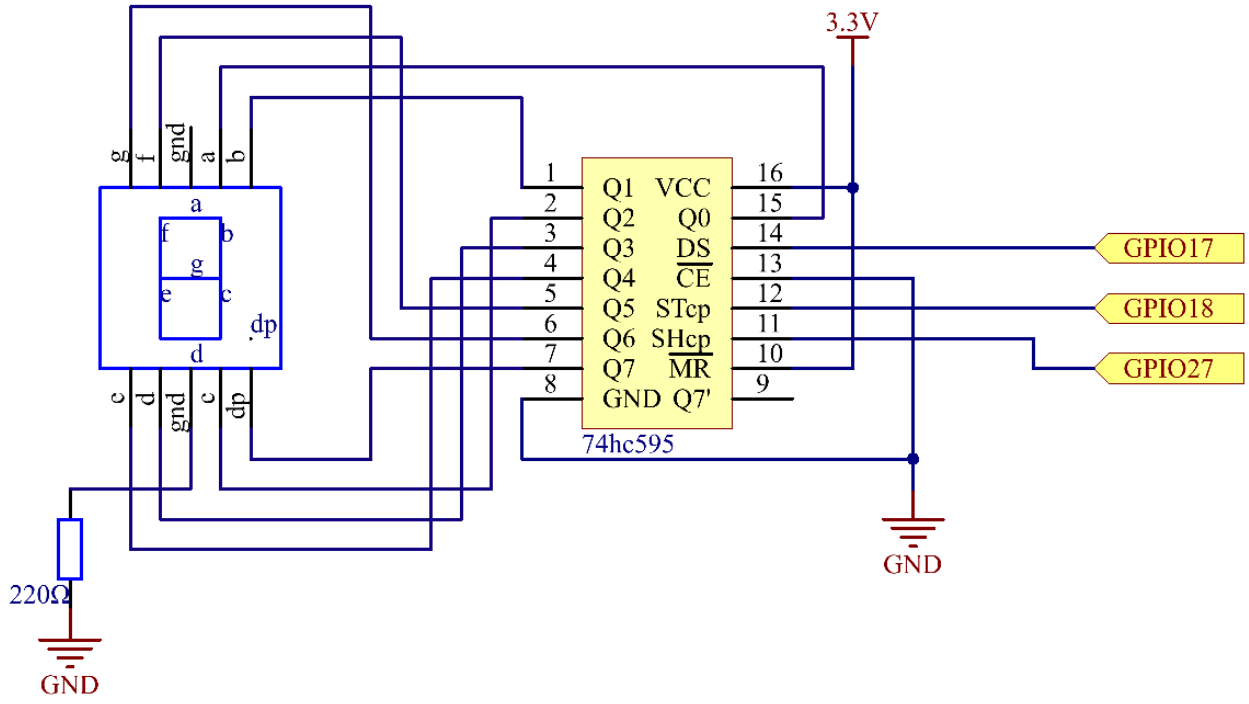
Schaltplan

Verbinden Sie den Pin ST_CP des 74HC595 mit dem GPIO18 des Raspberry Pi, SH_CP mit GPIO27, DS mit GPIO17, parallele Ausgangsports mit den 8 Segmenten der LED-Segment- Anzeige.

Daten werden in den DS-Pin eingegeben und beim Anstieg von SH_CP (dem Takt- eingang des Schieberegisters) in das Schieberegister und beim Anstieg von ST_CP (dem Takt- eingang des Speichers) in das Speicherregister übertragen.

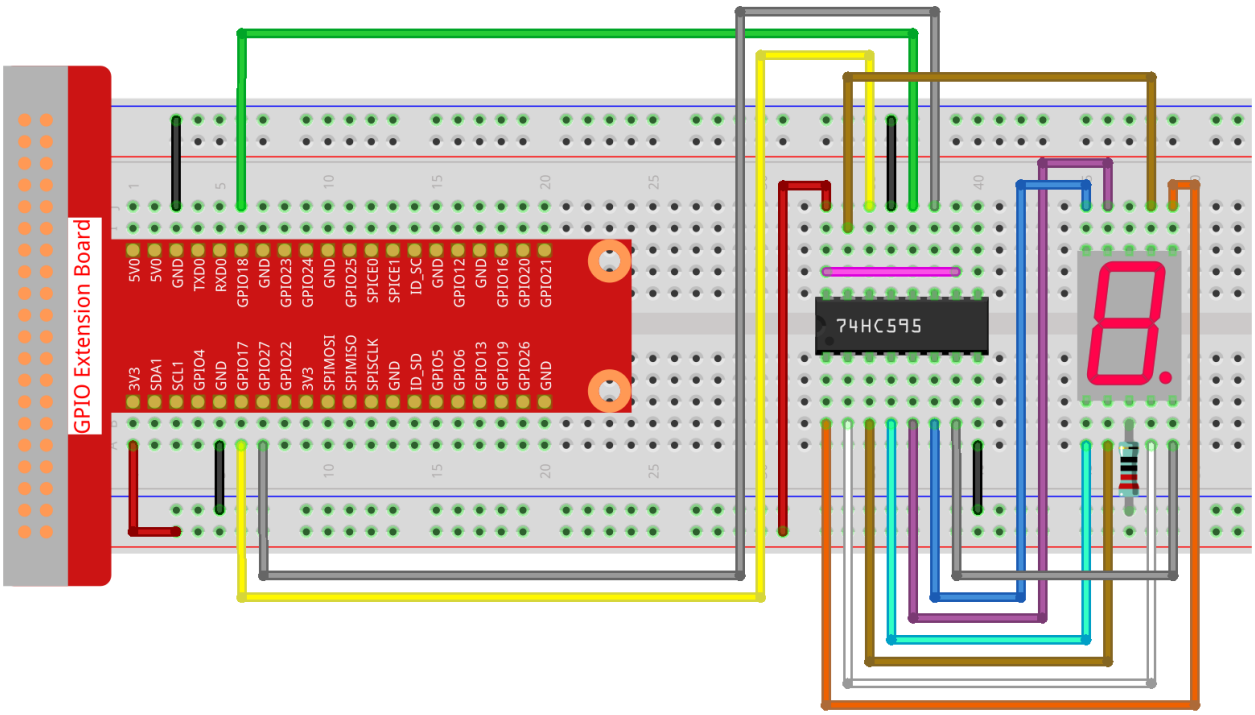
Dann können Sie die Zustände von SH_CP und ST_CP über die GPIOs des Raspberry Pi steuern, um serielle Dateneingaben in parallele Daten- ausgaben umzuwandeln, um GPIOs des Raspberry Pi zu sparen und die Anzeige anzusteuern.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie das Programm aus.

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

Nachdem der Code ausgeführt wurde, sehen Sie, dass die 7-Segment-Anzeige 0-9, A-F anzeigt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen. Nachdem Sie bestätigt haben, dass keine Probleme vorliegen, können Sie den geänderten Code mit der Kopieren-Schaltfläche kopieren, dann den Quellcode im Terminal über den Befehl `nano` öffnen und einfügen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep

# GPIO-Pins, die mit dem 74HC595 Schieberegister verbunden sind
SDI = OutputDevice(17)   # Serieller Dateneingang
RCLK = OutputDevice(18)  # Speichertakteingang (Register Clock)
SRCLK = OutputDevice(27) # Schieberegistertakt

# Hexadezimale Zifferncodes für eine gemeinsame Kathoden-7-Segment-Anzeige
segCode = [
    0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
    0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71
]

def hc595_shift(data):
    # 8 Bits Daten in den 74HC595 schieben
    for bit in range(8):
        # SDI auf hoch oder niedrig setzen, abhängig vom Datenbit
        SDI.value = 0x80 & (data << bit)
        # Schieberegistertakt auslösen
        SRCLK.on()
        sleep(0.001)
        SRCLK.off()
    # Daten durch Auslösen des Speichertakts in die Ausgabe übernehmen
    RCLK.on()
    sleep(0.001)
    RCLK.off()

def display_all_on():
    # Funktion, um alle Segmente einzuschalten (für gemeinsame Kathoden-7-Segment-
↪Anzeige)
    all_on_code = 0x3f
    hc595_shift(all_on_code)
    print("Alle Segmente eingeschaltet")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
try:
    while True:
        # Jede hexadezimale Ziffer auf der 7-Segment-Anzeige anzeigen
        for code in segCode:
            hc595_shift(code)  # Den Code in den 74HC595 schieben
            # Den angezeigten Segmentcode ausgeben
            print(f"Anzeige segCode[{segCode.index(code)}]: 0x{code:02X}")
            sleep(0.5)  # Pause zwischen der Anzeige jeder Ziffer

except KeyboardInterrupt:
    # Skriptunterbrechung (z. B. Ctrl+C) elegant behandeln
    pass
```

Code-Erklärung

1. Dieser Ausschnitt importiert die notwendigen Klassen für das Projekt. `OutputDevice` aus `gpiozero` wird verwendet, um Hardwarekomponenten, die mit GPIO-Pins verbunden sind, zu steuern, und `sleep` aus `time` für Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice
   from time import sleep
   ```

2. SDI, RCLK und SRCLK entsprechen den Pins für den seriellen Dateneingang, den Speichertakteingang (Register Clock) und den Schieberegistertakt des 74HC595.

   ```
   # GPIO-Pins, die mit dem 74HC595 Schieberegister verbunden sind
   SDI = OutputDevice(17)   # Serieller Dateneingang
   RCLK = OutputDevice(18)  # Speichertakteingang (Register Clock)
   SRCLK = OutputDevice(27) # Schieberegistertakt
   ```

3. `segCode` ist ein Array, das hexadezimale Codes für jede auf der 7-Segment-Anzeige darzustellende Ziffer enthält.

   ```
   # Hexadezimale Zifferncodes für eine gemeinsame Kathoden-7-Segment-Anzeige
   segCode = [
       0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
       0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71
   ]
   ```

4. Diese Funktion schiebt 8 Bits Daten in den 74HC595. Jedes Bit wird seriell in `SDI` eingegeben, `SRCLK` wird umgeschaltet, um das Bit zu verschieben, und `RCLK` wird verwendet, um die Daten an den Ausgang zu übertragen.

   ```
   def hc595_shift(data):
       # 8 Bits Daten in den 74HC595 schieben
       for bit in range(8):
           # SDI auf hoch oder niedrig setzen, abhängig vom Datenbit
           SDI.value = 0x80 & (data << bit)
           # Schieberegistertakt auslösen
           SRCLK.on()
           sleep(0.001)
           SRCLK.off()
       # Daten durch Auslösen des Speichertakts in die Ausgabe übernehmen
       RCLK.on()
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    sleep(0.001)
    RCLK.off()
```

5. Diese Funktion schaltet alle Segmente der Anzeige ein, indem sie einen spezifischen Code an `hc595_shift` sendet.

```
def display_all_on():
    # Funktion, um alle Segmente einzuschalten (für gemeinsame Kathoden-7-Segment-
↪Anzeige)
    all_on_code = 0x3f
    hc595_shift(all_on_code)
    print("Alle Segmente eingeschaltet")
```

6. In der Hauptschleife wird jeder Code in `segCode` nacheinander an die Anzeige gesendet, mit einer Verzögerung zwischen jedem.

```
try:
    while True:
        # Jede hexadezimale Ziffer auf der 7-Segment-Anzeige anzeigen
        for code in segCode:
            hc595_shift(code)  # Den Code in den 74HC595 schieben
            # Den angezeigten Segmentcode ausgeben
            print(f"Anzeige segCode[{segCode.index(code)}]: 0x{code:02X}")
            sleep(0.5)  # Pause zwischen der Anzeige jeder Ziffer
```

7. Dieser Teil des Codes behandelt die Unterbrechung des Skripts (wie Ctrl+C) auf elegante Weise.

```
except KeyboardInterrupt:
    # Skriptunterbrechung (z. B. Ctrl+C) elegant behandeln
    pass
```

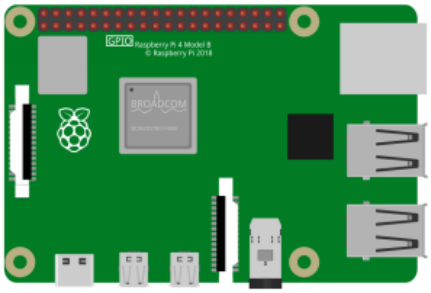
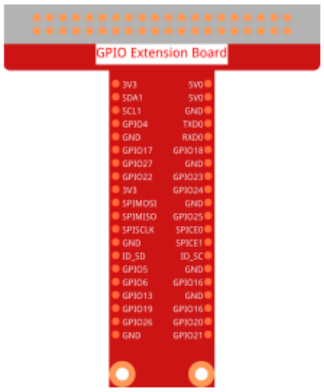
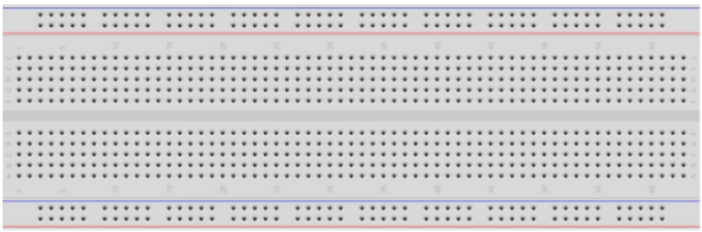
1.1.5 4-stellige 7-Segment-Anzeige

Einführung

Als Nächstes versuchen wir, die 4-stellige 7-Segment-Anzeige zu steuern.

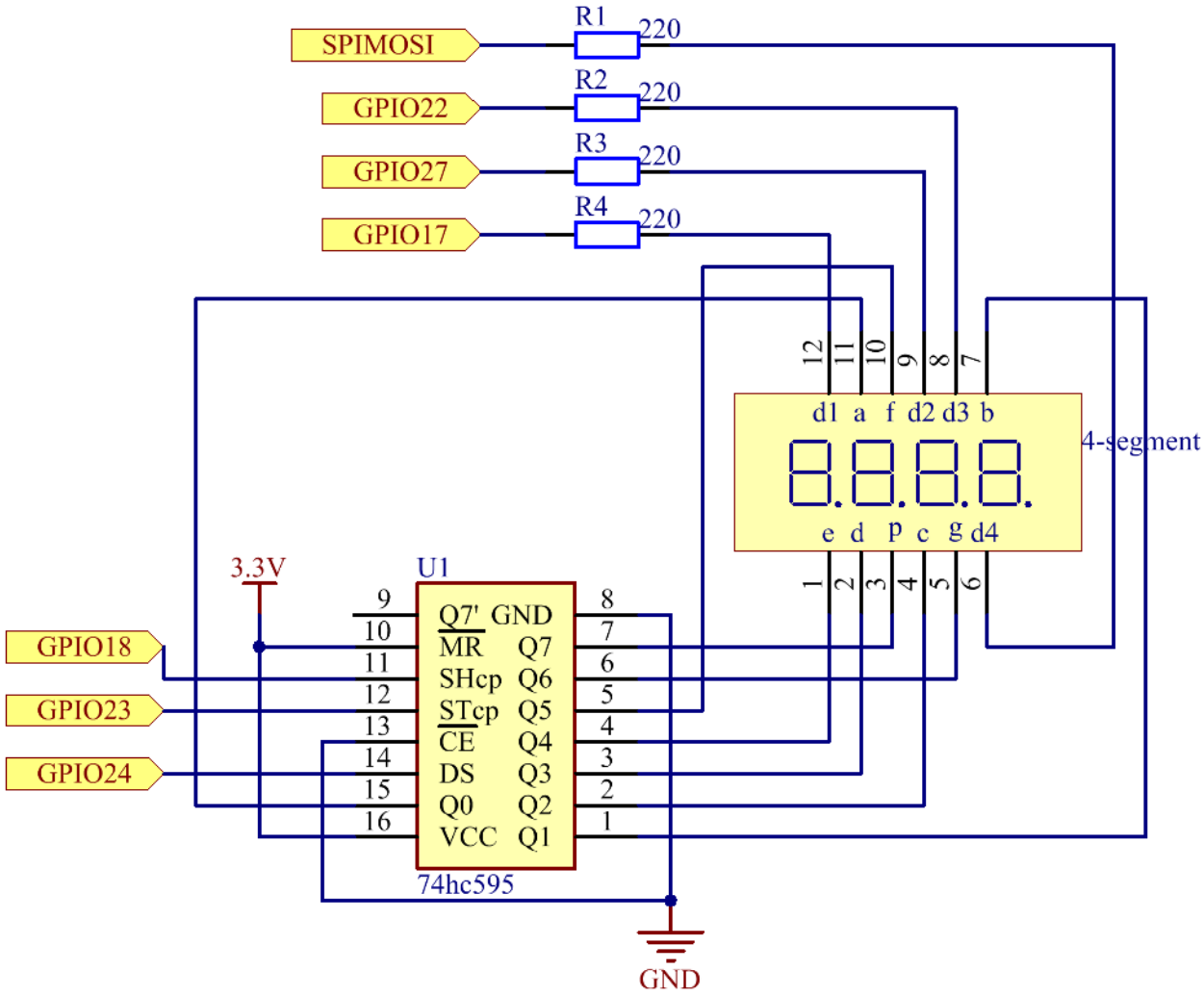
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

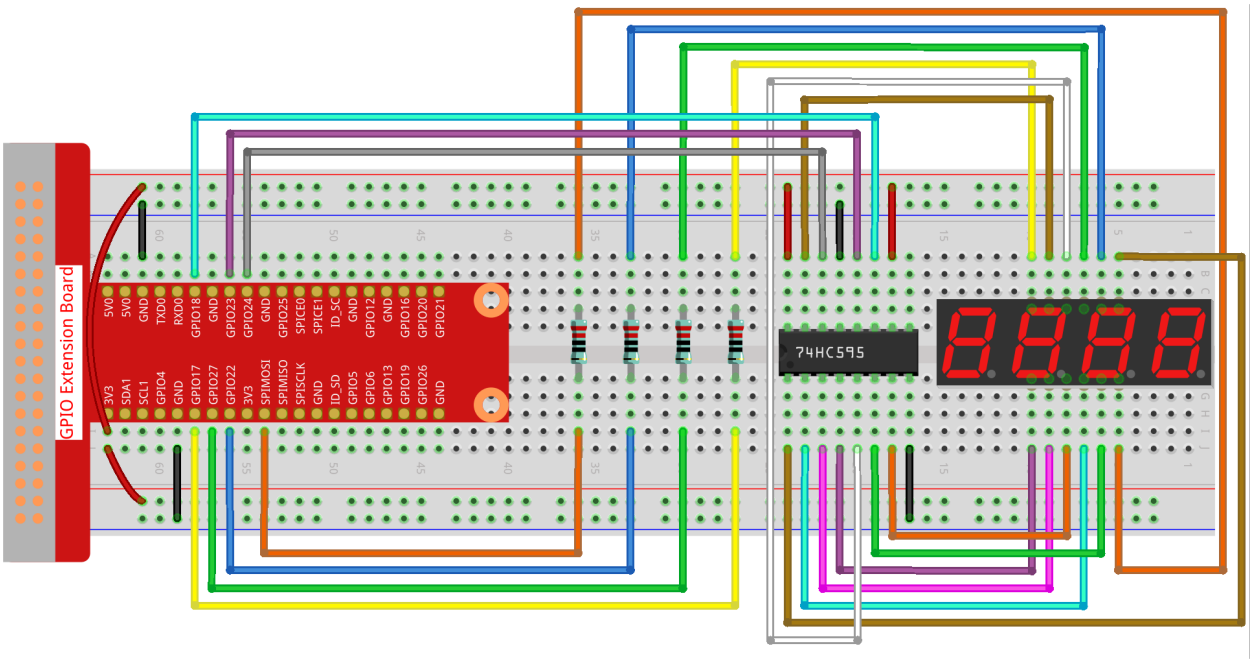
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

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

Nachdem der Code ausgeführt wurde, führt das Programm einen Zähler aus, der pro Sekunde um 1 erhöht wird, und die 4-stellige Anzeige zeigt den Zählerstand an.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
import time
import threading

# GPIO-Pins für das 74HC595 Schieberegister definieren
SDI = OutputDevice(24)   # Serieller Dateneingang
RCLK = OutputDevice(23)  # Register Clock
SRCLK = OutputDevice(18) # Schieberegistertakt

# GPIO-Pins für die Ziffernauswahl auf der 7-Segment-Anzeige definieren
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Segmentcodes für die Zahlen 0-9 auf der 7-Segment-Anzeige definieren
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
counter = 0  # Zähler für die Anzeige initialisieren
timer1 = 0   # Timer für Zählererhöhung initialisieren

def clearDisplay():
    """ Die 7-Segment-Anzeige löschen. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ Ein Byte Daten in das 74HC595 Schieberegister schieben. """
    for i in range(8):
        SDI.value = 0x80 & (data << i)  # SDI auf Hoch/Niedrig basierend auf dem␣
↪Datenbit setzen
        SRCLK.on()  # Schieberegistertakt auslösen
        SRCLK.off()
    RCLK.on()  # Daten durch Auslösen des Register Clocks in die Ausgabe übernehmen
    RCLK.off()

def pickDigit(digit):
    """ Eine Ziffer für die Anzeige auf der 7-Segment-Anzeige auswählen. """
    for pin in placePin:
        pin.off()  # Alle Ziffernauswahl-Pins ausschalten
    placePin[digit].on()  # Die ausgewählte Ziffer einschalten

def timer():
    """ Timerfunktion, um den Zähler jede Sekunde zu erhöhen. """
    global counter, timer1
    timer1 = threading.Timer(1.0, timer)  # Timer für nächste Erhöhung zurücksetzen
    timer1.start()
    counter += 1  # Zähler erhöhen
    print("%d" % counter)  # Aktuellen Zählerstand ausgeben

def setup():
    """ Anfangszustand einrichten und den Timer starten. """
    global timer1
    timer1 = threading.Timer(1.0, timer)  # Timer initialisieren und starten
    timer1.start()

def loop():
    """ Hauptloop, um die 7-Segment-Anzeige mit dem Zählerwert zu aktualisieren. """
    global counter
    while True:
        for i in range(4):  # Jede Ziffer durchlaufen
            clearDisplay()  # Anzeige löschen, bevor neue Ziffer gesetzt wird
            pickDigit(i)    # Ziffer für die Anzeige auswählen
            digit = (counter // (10 ** i)) % 10
            hc595_shift(number[digit])  # Ziffernwert in 74HC595 schieben
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            time.sleep(0.001)  # Kurze Verzögerung für Anzeigestabilität

def destroy():
    """ GPIO-Ressourcen freigeben und Timer bei Beendigung stoppen. """
    global timer1
    timer1.cancel()  # Timer stoppen
    for device in [SDI, RCLK, SRCLK] + placePin:
        device.close()  # GPIO-Geräte schließen

try:
    setup()  # Initialisierung einrichten
    while True:
        loop()  # Hauptloop starten

except KeyboardInterrupt:
    # Skriptunterbrechung (z. B. Ctrl+C) behandeln
    destroy()  # Ressourcen bei Beendigung aufräumen
```

Code-Erklärung

1. Diese vier Pins steuern die gemeinsamen Anodenpins der vierstelligen 7-Segment-Anzeigen.

   ```
   # GPIO-Pins für die Ziffernauswahl auf der 7-Segment-Anzeige definieren
   placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]
   ```

2. Ein Segmentcode-Array von 0 bis 9 in Hexadezimal (gemeinsame Anode).

   ```
   # Segmentcodes für die Zahlen 0-9 auf der 7-Segment-Anzeige definieren
   number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
   ```

3. Initialisiert einen Timer, der die *timer*-Funktion jede Sekunde auslöst. Dadurch wird die regelmäßige Zählererhöhung eingerichtet.

   ```
   def setup():
       """ Anfangszustand einrichten und den Timer starten. """
       global timer1
       timer1 = threading.Timer(1.0, timer)  # Timer initialisieren und starten
       timer1.start()
   ```

4. Nachdem der Timer 1,0 s erreicht hat, wird die Timerfunktion aufgerufen; 1 zum Zähler hinzufügen, und der Timer wird erneut verwendet, um sich selbst jede Sekunde wiederholt auszuführen.

   ```
   def timer():
       """ Timerfunktion, um den Zähler jede Sekunde zu erhöhen. """
       global counter, timer1
       timer1 = threading.Timer(1.0, timer)  # Timer für nächste Erhöhung zurücksetzen
       timer1.start()
       counter += 1  # Zähler erhöhen
       print("%d" % counter)  # Aktuellen Zählerstand ausgeben
   ```

5. Verschiebt ein Byte Daten in das 74HC595 Schieberegister und steuert damit die Anzeigesegmente.

   ```
   def hc595_shift(data):
       """ Ein Byte Daten in das 74HC595 Schieberegister schieben. """
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    for i in range(8):
        SDI.value = 0x80 & (data << i)  # SDI auf Hoch/Niedrig basierend auf dem␣
↪Datenbit setzen
        SRCLK.on()  # Schieberegistertakt auslösen
        SRCLK.off()
    RCLK.on()  # Daten durch Auslösen des Register Clocks in die Ausgabe übernehmen
    RCLK.off()
```

6. Aktualisiert kontinuierlich die Anzeige mit dem aktuellen Zählerwert, indem jede Ziffer nacheinander angezeigt wird.

```
def loop():
    """ Hauptloop, um die 7-Segment-Anzeige mit dem Zählerwert zu aktualisieren. """
    global counter
    while True:
        for i in range(4):  # Jede Ziffer durchlaufen
            clearDisplay()  # Anzeige löschen, bevor neue Ziffer gesetzt wird
            pickDigit(i)    # Ziffer für die Anzeige auswählen
            digit = (counter // (10 ** i)) % 10
            hc595_shift(number[digit])  # Ziffernwert in 74HC595 schieben
            time.sleep(0.001)  # Kurze Verzögerung für Anzeigestabilität
```

7. Löscht die 7-Segment-Anzeige, indem alle Segmente ausgeschaltet werden, bevor die nächste Ziffer angezeigt wird.

```
def clearDisplay():
    """ Die 7-Segment-Anzeige löschen. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

8. Wählt aus, welche Ziffer der 7-Segment-Anzeige aktiviert werden soll. Jede Ziffer wird durch einen separaten GPIO-Pin gesteuert.

```
def pickDigit(digit):
    """ Eine Ziffer für die Anzeige auf der 7-Segment-Anzeige auswählen. """
    for pin in placePin:
        pin.off()  # Alle Ziffernauswahl-Pins ausschalten
    placePin[digit].on()  # Die ausgewählte Ziffer einschalten
```

9. Gibt die GPIO-Ressourcen ordnungsgemäß frei und stoppt den Timer, wenn das Programm unterbrochen wird.

```
except KeyboardInterrupt:
    # Skriptunterbrechung (z. B. Ctrl+C) behandeln
    destroy()  # Ressourcen bei Beendigung aufräumen
```

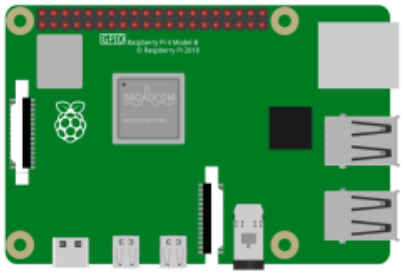
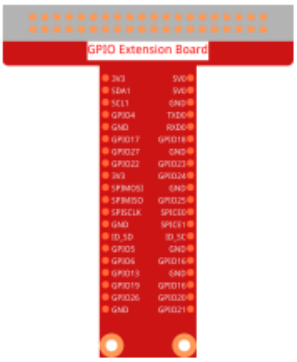
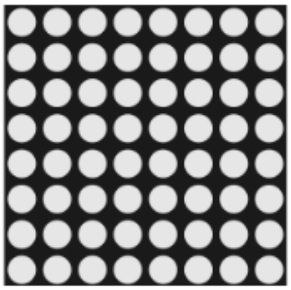
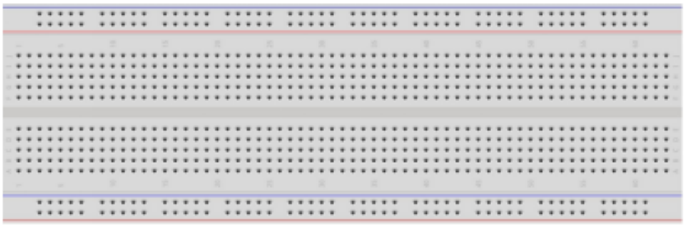
1.1.6 LED-Punktmatrix

Einführung

Wie der Name schon sagt, ist eine LED-Punktmatrix eine Matrix, die aus LEDs besteht. Das Aufleuchten und Verdunkeln der LEDs bildet verschiedene Zeichen und Muster.

Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * LED Dot Matrix
  </td></tr>
<tr><td>1 * 40-pin Cable
 </td><td colspan="2">Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td colspan="2">2 * 74HC595
 </td></tr>
</table>

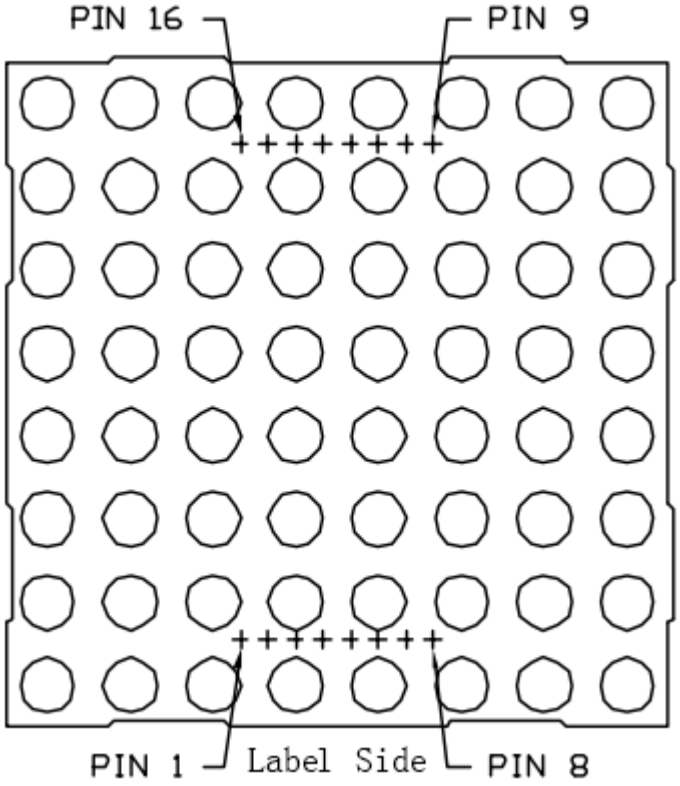
Prinzip

LED-Punktmatrix

Generell können LED-Punktmatrizen in zwei Typen eingeteilt werden: gemeinsame Kathode (CC) und gemeinsame Anode (CA). Sie sehen sich sehr ähnlich, aber der innere Aufbau ist unterschiedlich. Sie können dies durch Tests feststellen. In diesem Kit wird eine CA-Punktmatrix verwendet. Sie können 788BS an der Seite sehen.

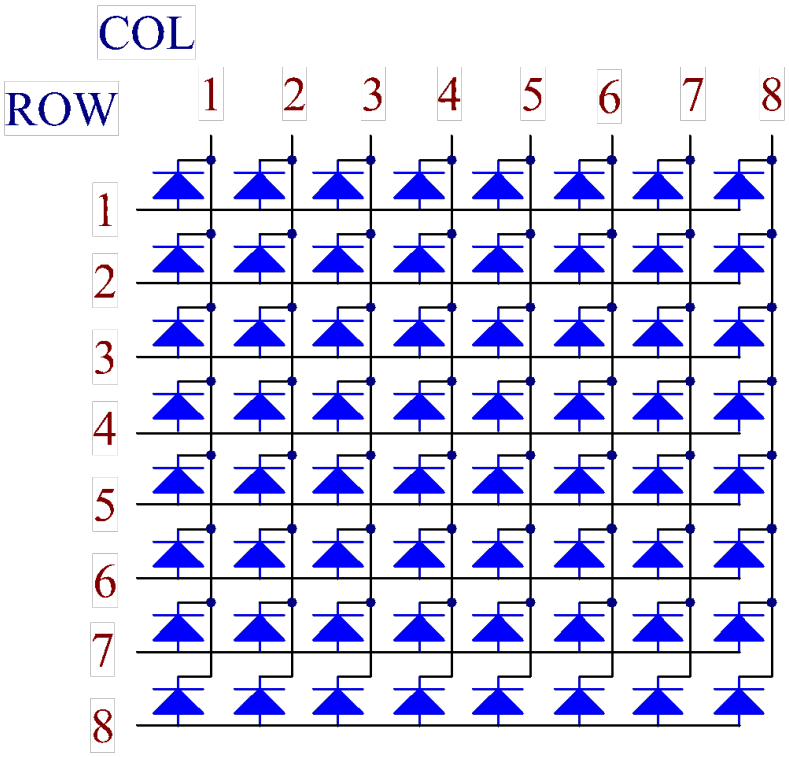
Siehe die Abbildung unten. Die Pins sind an den beiden Enden auf der Rückseite angeordnet. Nehmen Sie die Seite mit dem Etikett als Referenz: Pins an diesem Ende sind Pin 1-8, und oh die anderen sind Pin 9-16.

Die Außenansicht:



Unten die Abbildungen zeigen ihre interne Struktur. Sie können sehen, in einer CA-LED Punktmatrix repräsentiert ROW die Anode der LED und COL ist die Kathode; für eine CC-Matrix ist es umgekehrt. Eine Gemeinsamkeit: für beide Typen sind Pin 13, 3, 4, 10, 6, 11, 15 und 16 alle COL, wenn Pin 9, 14, 8, 12, 1, 7, 2, und 5 alle ROW sind. Wenn Sie die erste LED in der oberen linken Ecke einschalten möchten, für eine CA-LED-Punktmatrix, setzen Sie einfach Pin 9 auf High und Pin 13 auf Low, und für eine CC-Matrix setzen Sie Pin 13 auf High und Pin 9 auf Low. Wenn Sie die ganze erste Spalte aufleuchten lassen möchten, für CA, setzen Sie Pin 13 auf Low und ROW 9, 14, 8, 12, 1, 7, 2 und 5 auf High, wenn für CC, setzen Sie Pin 13 auf High und ROW 9, 14, 8, 12, 1, 7, 2 und 5 auf Low. Betrachten Sie die folgenden Abbildungen für ein besseres Verständnis.

Die Innenansicht:



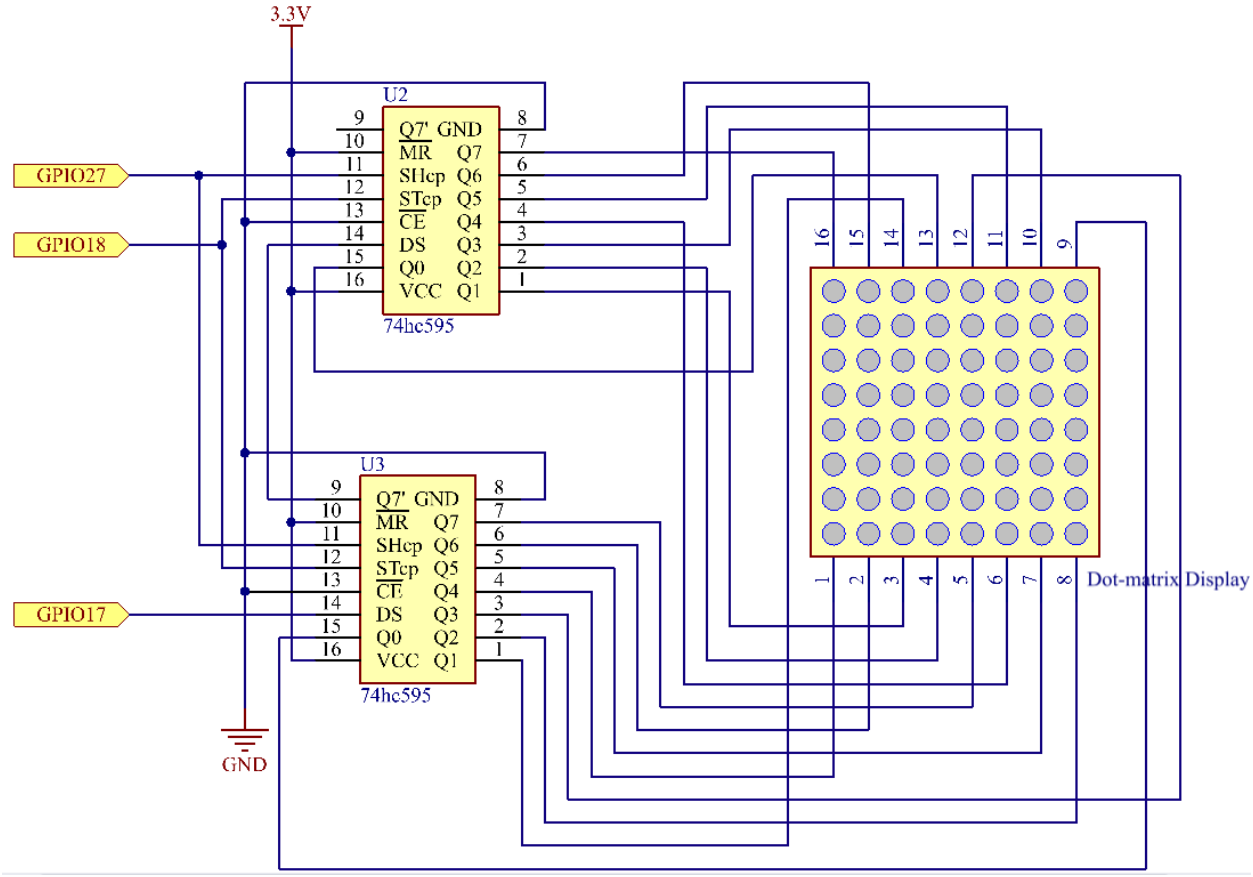
Pin-Nummerierung entsprechend den obigen Reihen und Spalten:

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**

Zusätzlich werden hier zwei 74HC595-Chips verwendet. Einer dient zur Steuerung der Reihen der LED-Punktmatrix, während der andere die Spalten steuert.

Schaltplan

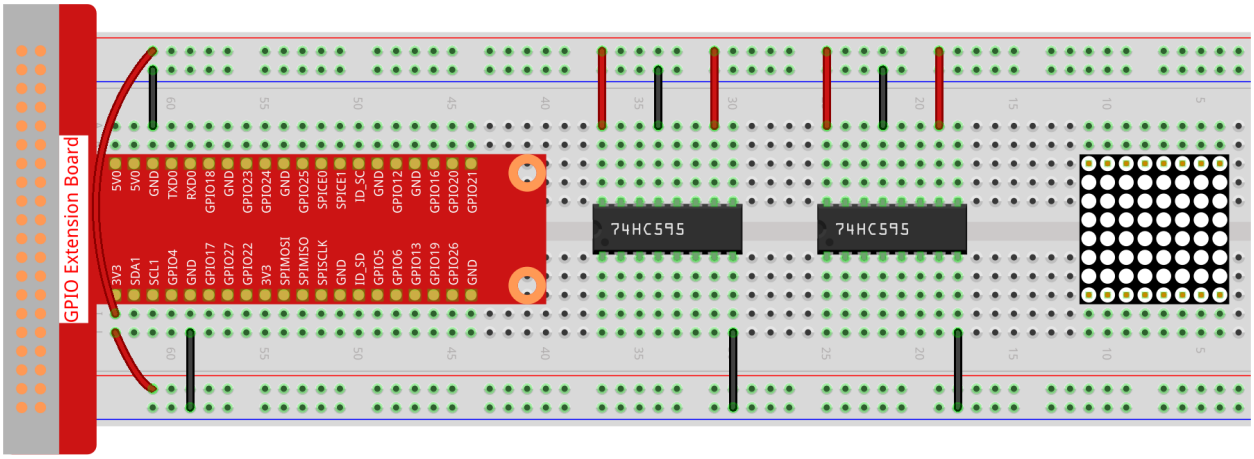
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



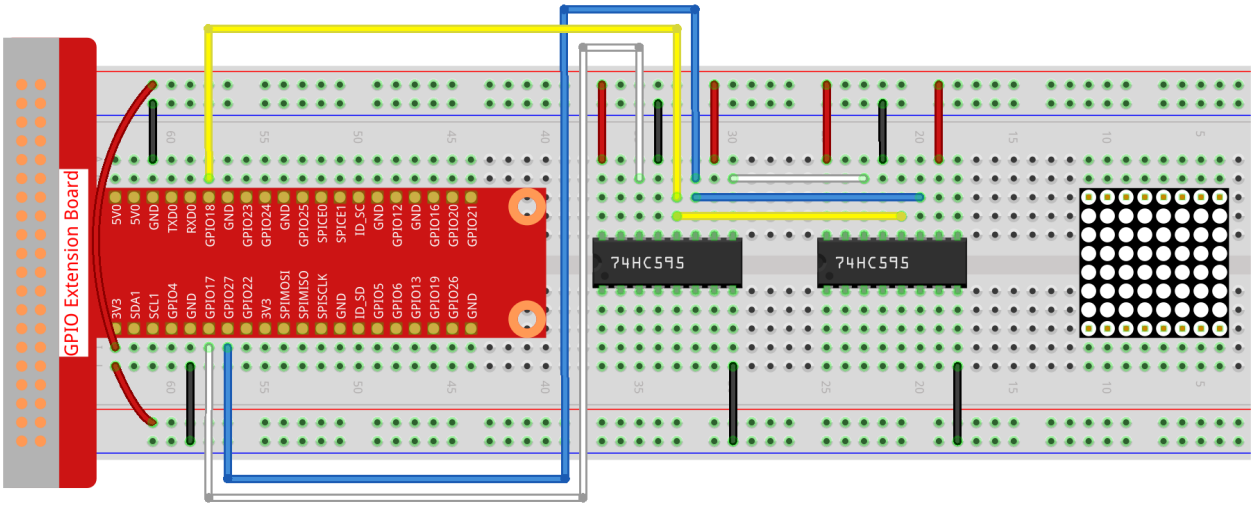
Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf. Da die Verkabelung kompliziert ist, machen wir es Schritt für Schritt. Zuerst stecken Sie den T-Cobbler, die LED-Punktmatrix und zwei 74HC595-Chips in das Steckbrett. Verbinden Sie die 3,3V und GND des T-Cobblers mit Löchern auf den beiden Seiten der Platine, dann verbinden Sie Pin 16 und 10 der beiden 74HC595-Chips mit VCC, Pin 13 und Pin 8 mit GND.

Bemerkung: Im obigen Fritzing-Bild ist die Seite mit dem Etikett unten.

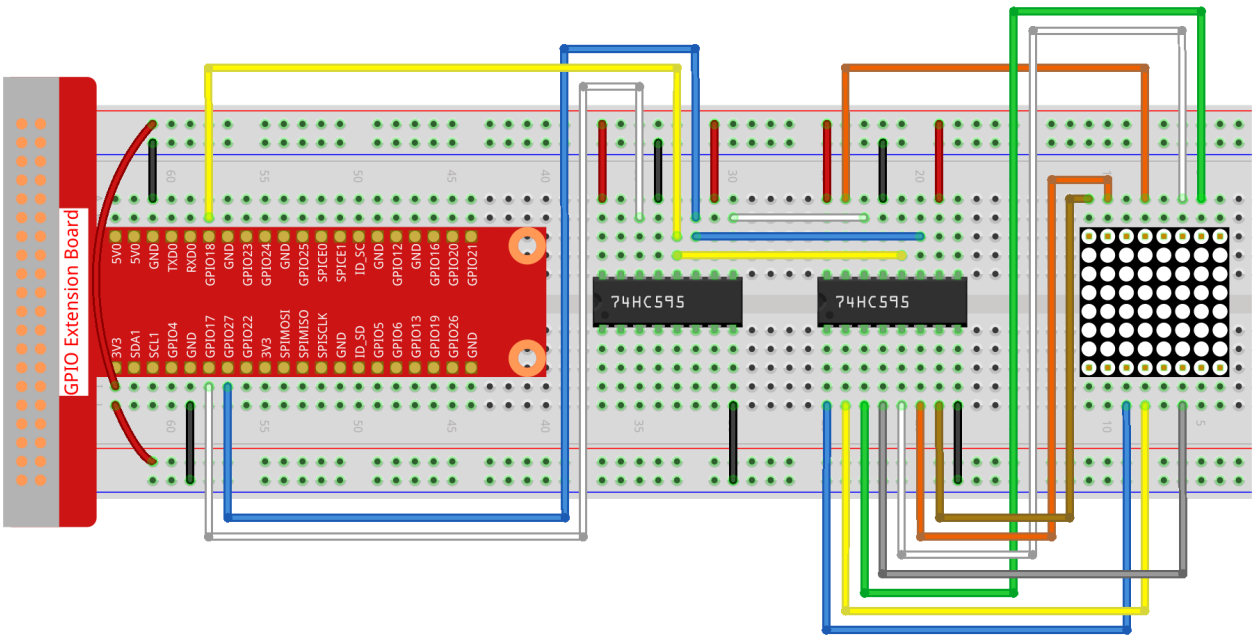


Schritt 2: Verbinden Sie Pin 11 der beiden 74HC595 miteinander und dann mit GPIO27; dann Pin 12 der beiden Chips und mit GPIO18; als Nächstes Pin 14 des 74HC595 auf der linken Seite mit GPIO17 und Pin 9 mit Pin 14 des zweiten 74HC595.



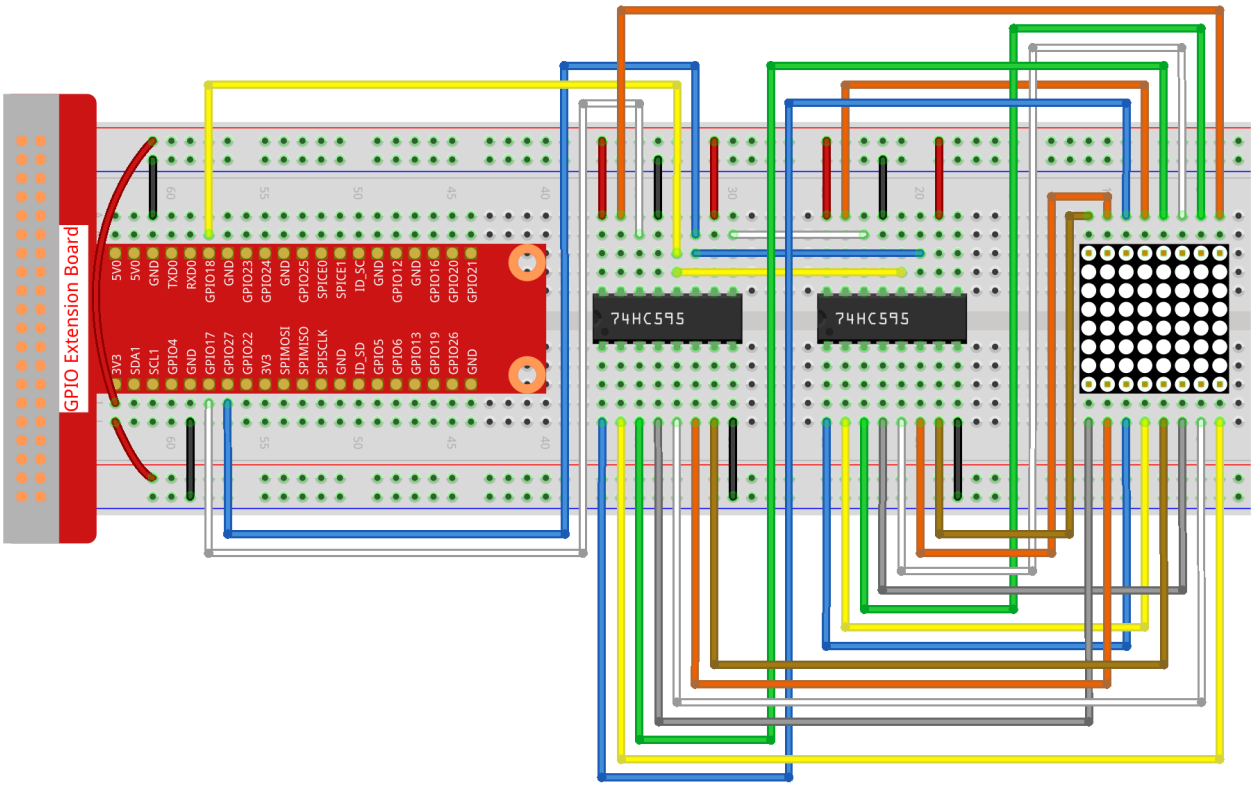
Schritt 3: Der 74HC595 auf der rechten Seite steuert die Spalten der LED-Punktmatrix. Sehen Sie die Tabelle unten für die Zuordnung. Daher sind Q0-Q7 Pins des 74HC595 mit Pin 13, 3, 4, 10, 6, 11, 15 und 16 entsprechend zugeordnet.

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



Schritt 4: Jetzt verbinden Sie die ROWs der LED-Punktmatrix. Der 74HC595 auf der linken Seite steuert die ROW der LED-Punktmatrix. Sehen Sie die Tabelle unten für die Zuordnung. Wir können sehen, Q0-Q7 des 74HC595 auf der linken Seite sind mit Pin 9, 14, 8, 12, 1, 7, 2 und 5 entsprechend zugeordnet.

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



Schritt 5: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 6: Führen Sie das Programm aus.

```
sudo python3 1.1.6_led_dot_matrix_zero.py
```

Nachdem der Code ausgeführt wurde, leuchtet die LED-Punktmatrix Reihe für Reihe und Spalte für Spalte auf und ab.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep

# GPIO-Pins, die mit dem 74HC595 Schieberegister verbunden sind, definieren
SDI = OutputDevice(17)   # Serieller Dateneingang
RCLK = OutputDevice(18)  # Register Clock
SRCLK = OutputDevice(27) # Schieberegistertakt

# Muster für die Matrixanzeige definieren; ROWs sind Anoden (+), COLs sind Kathoden (-)
# Muster für ROWs (Anodensignale)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
code_H = [0x01, 0xff, 0x80, 0xff, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0xff,␣
↪0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
# Muster für COLs (Kathodensignale)
code_L = [0x00, 0x7f, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe,␣
↪0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f]

# Daten an 74HC595 senden
def hc595_shift(dat):
    """ Daten an das 74HC595 Schieberegister senden, um auf der Matrix anzuzeigen. """
    for i in range(8):
        # SDI-Wert setzen und Schieberegistertakt auslösen
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
    # Register Clock auslösen, um die Anzeige zu aktualisieren
    RCLK.on()
    sleep(0.001)
    RCLK.off()

def main():
    """ Hauptloop, um durch die Anzeigemuster zu zyklen. """
    while True:
        # Durch die Muster in aufsteigender Reihenfolge zyklen
        for i in range(len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

        # Durch die Muster in absteigender Reihenfolge zyklen
        for i in range(len(code_H)-1, -1, -1):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

# Hauptloop ausführen, Tastaturunterbrechung elegant behandeln
try:
    main()
except KeyboardInterrupt:
    pass
```

Code-Erklärung

1. Dieser Abschnitt importiert die notwendigen Klassen für das Projekt. `OutputDevice` aus `gpiozero` wird verwendet, um Hardwarekomponenten, die mit GPIO-Pins verbunden sind, zu steuern, und `sleep` aus `time` für Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice
   from time import sleep
   ```

2. Dieser Abschnitt initialisiert GPIO-Pins, die mit dem 74HC595 Schieberegister verbunden sind. `SDI` ist der serielle Dateneingang, `RCLK` ist der Register Clock und `SRCLK` ist der Schieberegistertakt. Diese Pins werden verwendet, um Daten in das Register zu schieben und die Anzeige der LED-Matrix zu steuern.

```
# GPIO-Pins, die mit dem 74HC595 Schieberegister verbunden sind, definieren
SDI = OutputDevice(17)   # Serieller Dateneingang
RCLK = OutputDevice(18)  # Register Clock
SRCLK = OutputDevice(27) # Schieberegistertakt
```

3. code_H und code_L definieren binäre Muster für die Steuerung der Reihen (Anoden) und Spalten (Kathoden) der LED-Matrix. Jedes Element in diesen Arrays stellt ein binäres Muster dar, das steuert, welche LEDs in der Matrix ein- oder ausgeschaltet werden.

```
# Muster für die Matrixanzeige definieren; ROWs sind Anoden (+), COLs sind Kathoden␣
↪(-)
# Muster für ROWs (Anodensignale)
code_H = [0x01, 0xff, 0x80, 0xff, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,␣
↪0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
# Muster für COLs (Kathodensignale)
code_L = [0x00, 0x7f, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,␣
↪0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f]
```

4. Diese Funktion schiebt ein Byte Daten (dat) in das 74HC595 Schieberegister. Sie iteriert über jedes Bit im Byte, setzt das SDI-Pin hoch oder niedrig basierend auf dem Bitwert und schaltet das SRCLK-Pin um, um das Bit in das Register zu verschieben. Nachdem alle Bits verschoben sind, schaltet sie das RCLK-Pin um, um die LED-Matrix-Anzeige zu aktualisieren.

```
# Daten an 74HC595 senden
def hc595_shift(dat):
    """ Daten an das 74HC595 Schieberegister senden, um auf der Matrix anzuzeigen. ""
↪"
    for i in range(8):
        # SDI-Wert setzen und Schieberegistertakt auslösen
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
    # Register Clock auslösen, um die Anzeige zu aktualisieren
    RCLK.on()
    sleep(0.001)
    RCLK.off()
```

5. Die Hauptfunktion enthält eine unendliche Schleife, die durch vordefinierte Muster für die LED-Matrix zyklisch durchläuft. Sie verwendet die Funktion hc595_shift, um Reihen- und Spaltenmuster (code_H und code_L) an das Schieberegister zu senden, zuerst in aufsteigender Reihenfolge und dann in absteigender Reihenfolge, wodurch eine dynamische Anzeige entsteht.

```
def main():
    """ Hauptloop, um durch die Anzeigemuster zu zyklen. """
    while True:
        # Durch die Muster in aufsteigender Reihenfolge zyklen
        for i in range(len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)

        # Durch die Muster in absteigender Reihenfolge zyklen
        for i in range(len(code_H)-1, -1, -1):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            sleep(0.1)
```

6. Dieser Abschnitt stellt sicher, dass das Programm mit einer Tastaturunterbrechung (Ctrl+C) unterbrochen werden kann. Es beendet die Hauptschleife ohne abrupten Stopp oder Ressourcenlecks.

```
# Hauptloop ausführen, Tastaturunterbrechung elegant behandeln
try:
   main()
except KeyboardInterrupt:
   pass
```

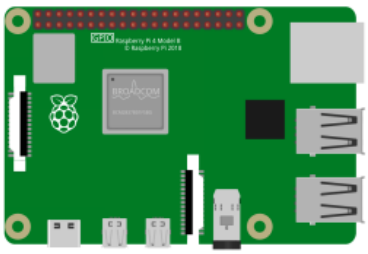
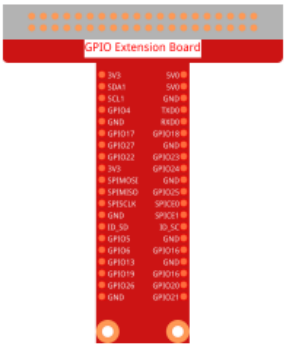
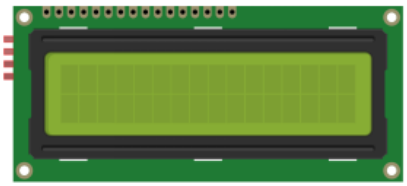
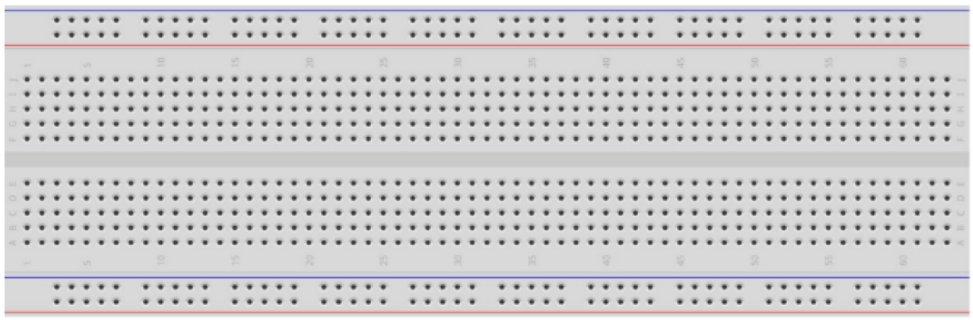
1.1.7 I2C LCD1602

Einführung

Das LCD1602 ist ein Zeichentyp-Flüssigkristalldisplay, das gleichzeitig 32 (16*2) Zeichen anzeigen kann.

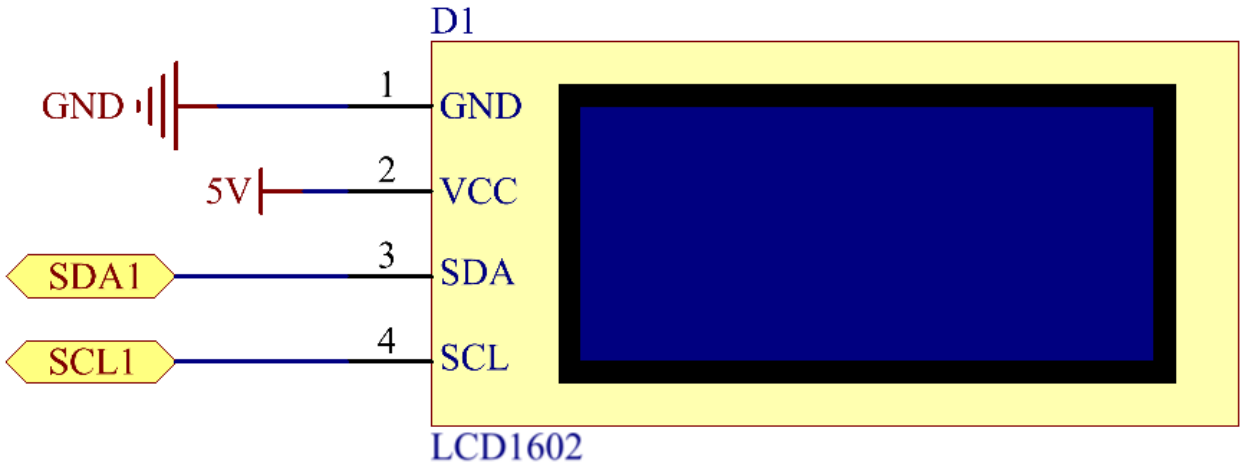
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

1 * Raspberry Pi 	1 * T-Extension Board 	1 * I2C LCD1602 
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		

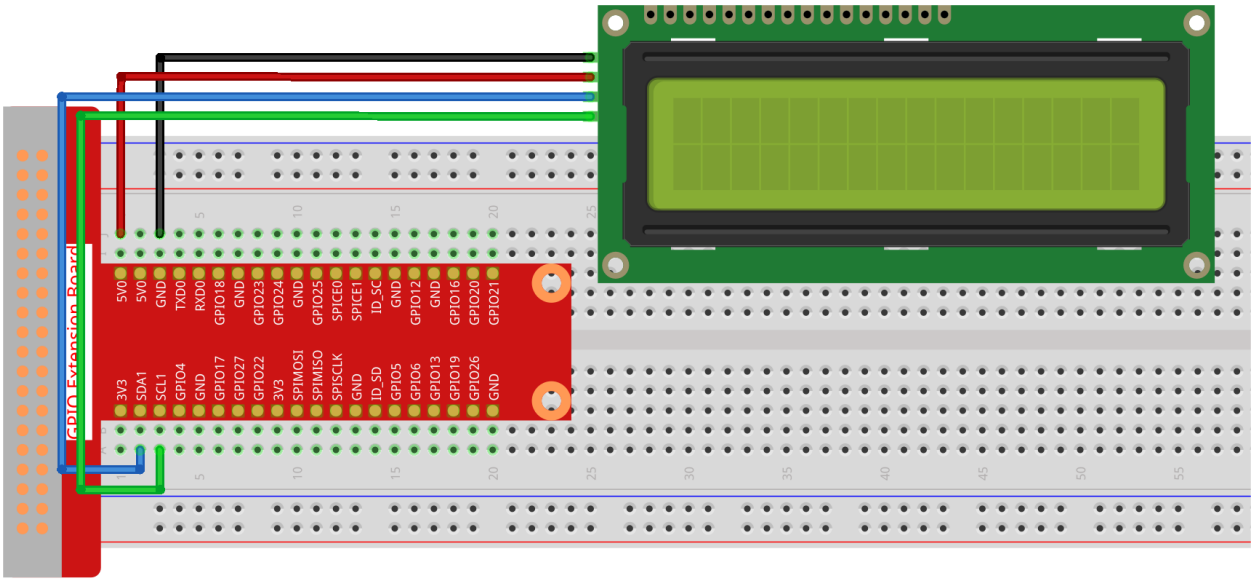
Schaltplan

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: I2C einrichten (siehe *I2C-Konfiguration*. Wenn Sie I2C bereits eingerichtet haben, überspringen Sie diesen Schritt.)

Schritt 3: Verzeichnis wechseln.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 4: Ausführen.

```
sudo python3 1.1.7_Lcd1602_zero.py
```

Nachdem der Code ausgeführt wurde, können Sie auf dem LCD sehen, dass „Greetings!, From SunFounder“ angezeigt wird.

Bemerkung:

- Wenn Sie den Fehler „FileNotFoundError: [Errno 2] No such file or directory: ‚/dev/i2c-1'“ erhalten, müssen Sie *I2C-Konfiguration* beachten, um das I2C zu aktivieren.
- Wenn Sie den Fehler „ModuleNotFoundError: No module named ‚smbus2'“ erhalten, führen Sie bitte „sudo pip3 install smbus2“ aus.
- Wenn der Fehler „OSError: [Errno 121] Remote I/O error“ auftritt, bedeutet dies, dass das Modul falsch verkabelt ist oder das Modul defekt ist.
- Wenn der Code und die Verkabelung in Ordnung sind, das LCD jedoch immer noch keinen Inhalt anzeigt, können Sie das Potentiometer auf der Rückseite drehen, um den Kontrast zu erhöhen.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
import LCD1602  # Modul für die Schnittstelle mit LCD1602 importieren
import time     # Modul für Timing-Funktionen importieren

def setup():
    # LCD mit I2C-Adresse 0x27 initialisieren und Hintergrundbeleuchtung aktivieren
    LCD1602.init(0x27, 1)
    # Die Nachricht 'Greetings!' in der oberen linken Ecke anzeigen (Zeile 0, Spalte 0)
    LCD1602.write(0, 0, 'Greetings!')
    # Die Nachricht 'From SunFounder' in der zweiten Zeile anzeigen (Zeile 1, Spalte 1)
    LCD1602.write(1, 1, 'From SunFounder')
    time.sleep(2)  # Nachrichten für 2 Sekunden anzeigen

try:
    setup()  # Setup-Funktion ausführen, um das LCD zu initialisieren und Nachrichten␣
↪anzuzeigen

except KeyboardInterrupt:
    # LCD-Display löschen, wenn eine Tastaturunterbrechung (z. B. Ctrl+C) auftritt
    LCD1602.clear()
    pass  # Ohne weitere Aktion fortfahren
```

Code-Erklärung

1. Diese Datei ist eine Open-Source-Datei zur Steuerung des I2C LCD1602. Sie ermöglicht uns, das I2C LCD1602 einfach zu verwenden.

   ```
   import LCD1602  # Modul für die Schnittstelle mit LCD1602 importieren
   ```

2. Die Funktion initialisiert das I2C-System mit dem festgelegten Gerätesymbol. Der erste Parameter ist die Adresse des I2C-Geräts, die über den Befehl i2cdetect ermittelt werden kann (siehe Anhang für Details). Die Adresse des I2C LCD1602 ist in der Regel 0x27.

   ```
   # LCD mit I2C-Adresse 0x27 initialisieren und Hintergrundbeleuchtung aktivieren
   LCD1602.init(0x27, 1)
   ```

3. `LCD1602.write` wird verwendet, um Nachrichten auf dem LCD anzuzeigen. Die ersten beiden Parameter sind die Positionen von Zeile und Spalte, der dritte ist die Nachricht. Jetzt können Sie „Greetings!! From SunFounder“ auf dem LCD angezeigt sehen.

```
# Die Nachricht 'Greetings!' in der oberen linken Ecke anzeigen (Zeile 0, Spalte 0)
LCD1602.write(0, 0, 'Greetings!')
# Die Nachricht 'From SunFounder' in der zweiten Zeile anzeigen (Zeile 1, Spalte 1)
LCD1602.write(1, 1, 'From SunFounder')
```

1.2 Ton

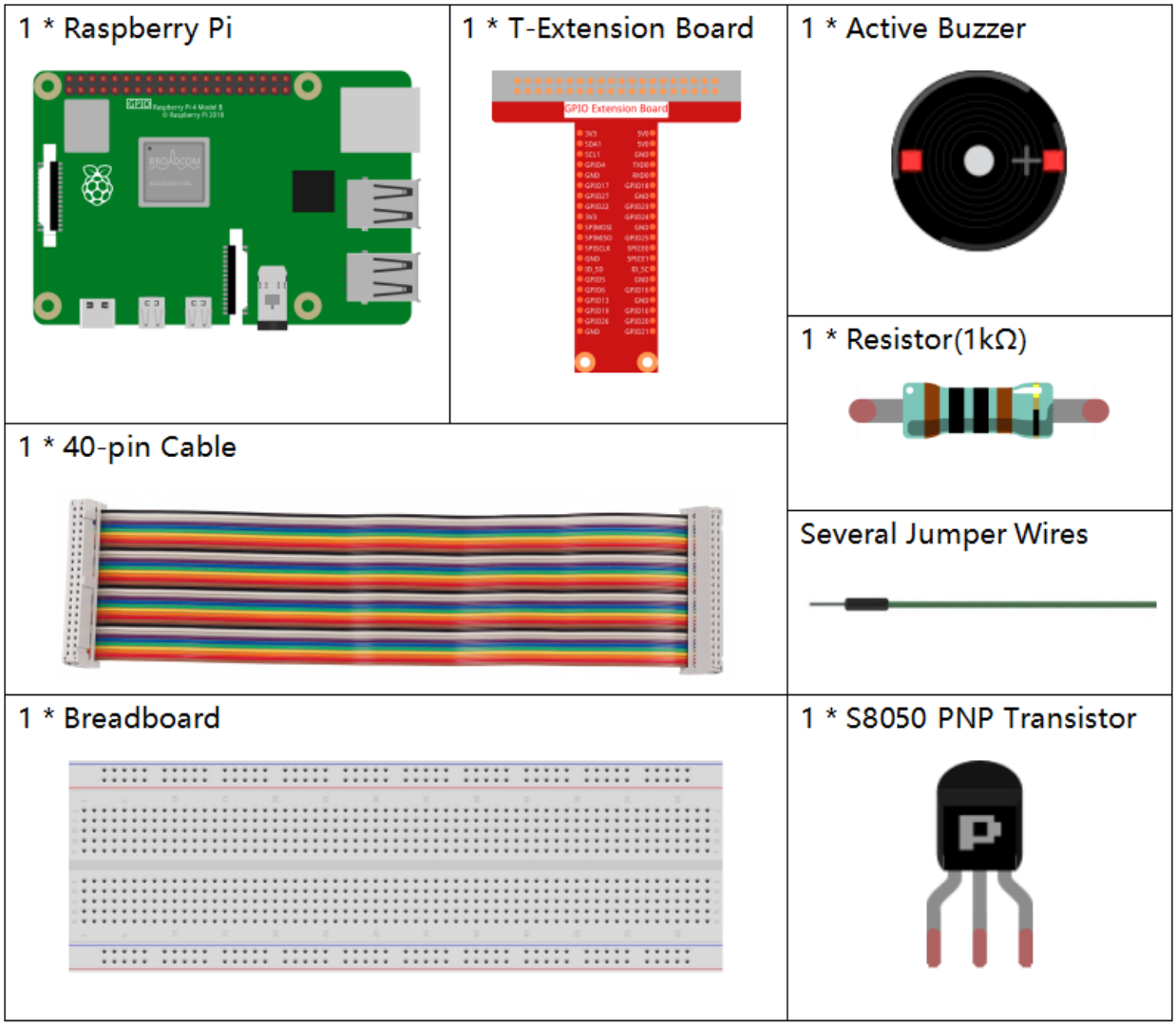
1.2.1 Aktiver Summer

Einführung

In diesem Projekt werden wir lernen, wie man einen aktiven Summer mit einem PNP-Transistor zum Piepen bringt.

Benötigte Komponenten

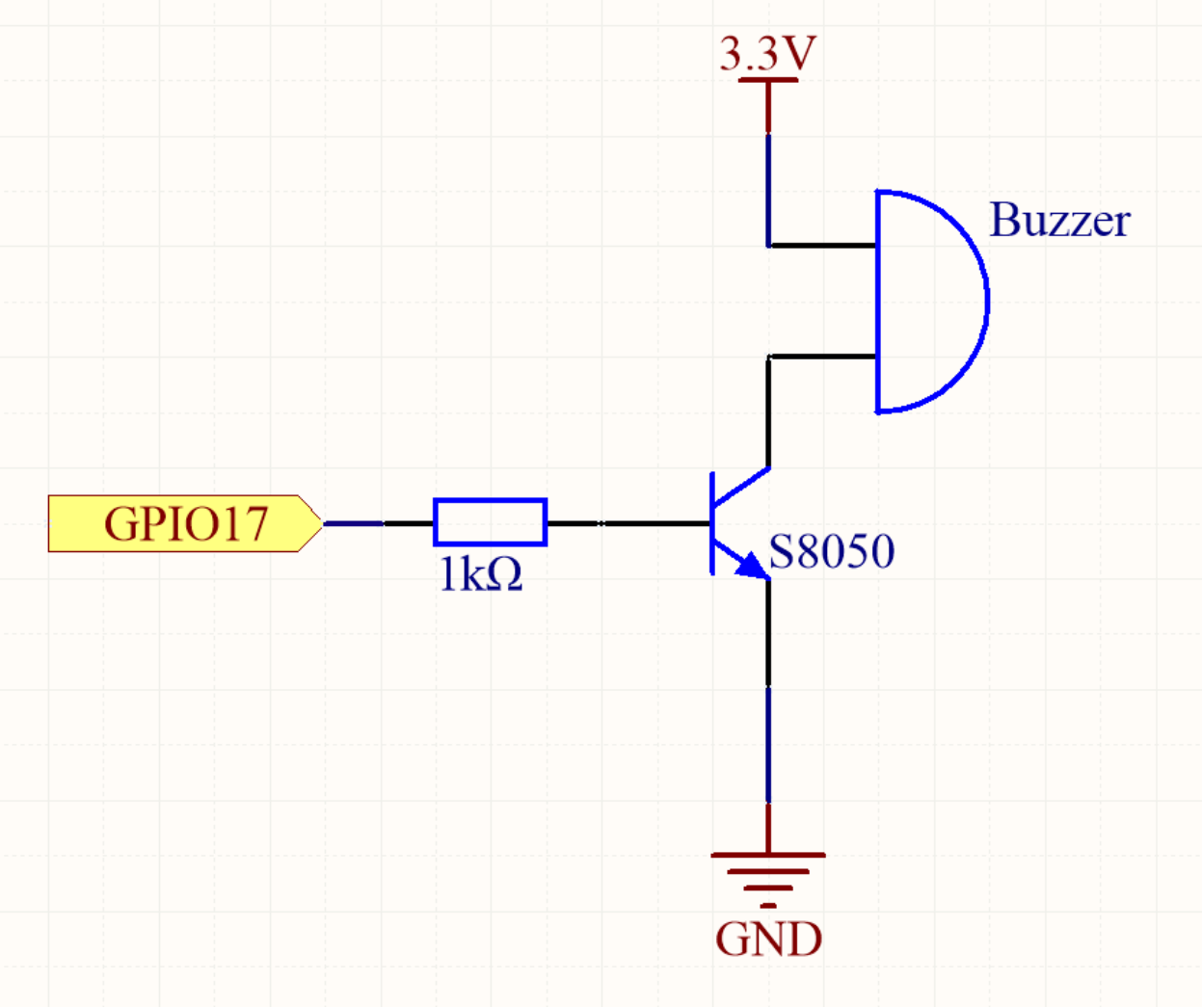
Für dieses Projekt benötigen wir die folgenden Komponenten.



Schaltplan

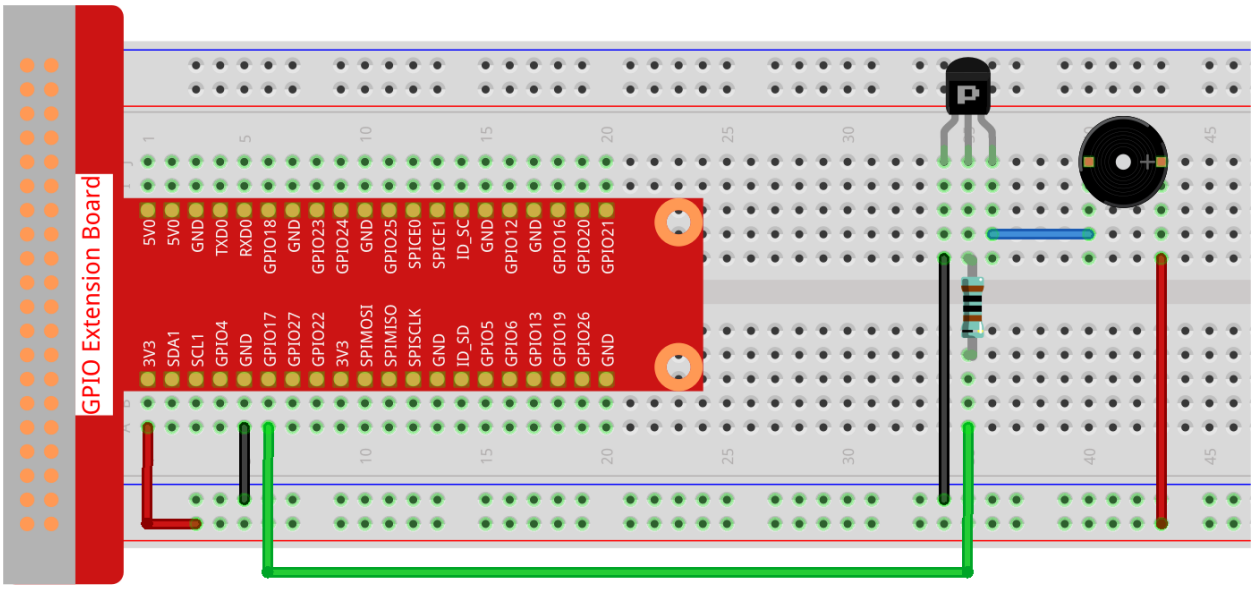
In diesem Experiment wird ein aktiver Summer, ein PNP-Transistor und ein 1k-Widerstand zwischen der Basis des Transistors und GPIO verwendet, um den Transistor zu schützen. Wenn der GPIO17 des Raspberry Pi mit High-Level (3,3V) durch Programmierung versorgt wird, leitet der Transistor wegen Stromsättigung und der Summer gibt Töne ab. Wenn jedoch ein Low-Level an den IO des Raspberry Pi geliefert wird, ist der Transistor gesperrt und der Summer gibt keine Töne von sich.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf. (Der aktive Summer hat einen weißen Tischetikettaufkleber auf der Oberfläche und eine schwarze Rückseite.)



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.2.1_ActiveBuzzer_zero.py
```

Wenn der Code ausgeführt wird, piept der Summer.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import Buzzer
from time import sleep

# Initialisieren Sie ein Buzzer-Objekt am GPIO-Pin 17
buzzer = Buzzer(17)

try:
    while True:
        # Summer einschalten
        print('Summer An')
        buzzer.on()
        sleep(0.1)  # Summer für 0,1 Sekunden eingeschaltet lassen

        # Summer ausschalten
        print('Summer Aus')
        buzzer.off()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        sleep(0.1)  # Summer für 0,1 Sekunden ausgeschaltet lassen

except KeyboardInterrupt:
    # Tastaturunterbrechung (Ctrl+C) sauber behandeln
    pass
```

Code-Erklärung

1. Diese Anweisungen importieren die Klasse `Buzzer` aus der Bibliothek `gpiozero` und die Funktion `sleep` aus dem Modul `time`.

   ```
   #!/usr/bin/env python3
   from gpiozero import Buzzer
   from time import sleep
   ```

2. Diese Zeile erstellt ein `Buzzer`-Objekt, das an GPIO-Pin 17 am Raspberry Pi angeschlossen ist.

   ```
   # Initialisieren Sie ein Buzzer-Objekt am GPIO-Pin 17
   buzzer = Buzzer(17)
   ```

3. In einer unendlichen Schleife (`while True`) wird der Summer alle 0,1 Sekunden ein- und ausgeschaltet. `print`-Anweisungen bieten eine Konsolenausgabe für jede Aktion.

   ```
   try:
       while True:
           # Summer einschalten
           print('Summer An')
           buzzer.on()
           sleep(0.1)  # Summer für 0,1 Sekunden eingeschaltet lassen

           # Summer ausschalten
           print('Summer Aus')
           buzzer.off()
           sleep(0,1)  # Summer für 0,1 Sekunden ausgeschaltet lassen
   ```

4. Dieser Abschnitt stellt sicher, dass das Programm mit einer Tastaturunterbrechung (Ctrl+C) sicher beendet werden kann, ohne einen Fehler zu verursachen.

   ```
   except KeyboardInterrupt:
   # Tastaturunterbrechung (Ctrl+C) sauber behandeln
   pass
   ```

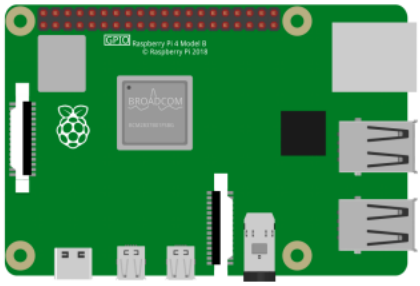
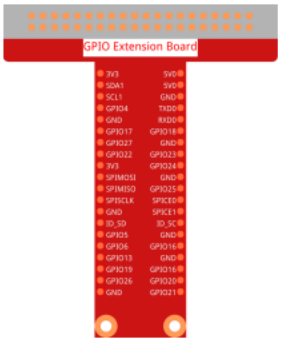
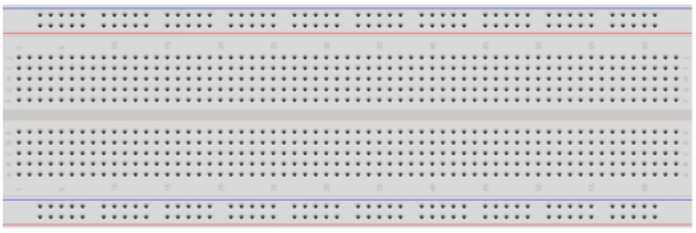
1.2.2 Passiver Summer

Einführung

In diesem Projekt werden wir lernen, wie man einen passiven Summer dazu bringt, Musik zu spielen.

Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

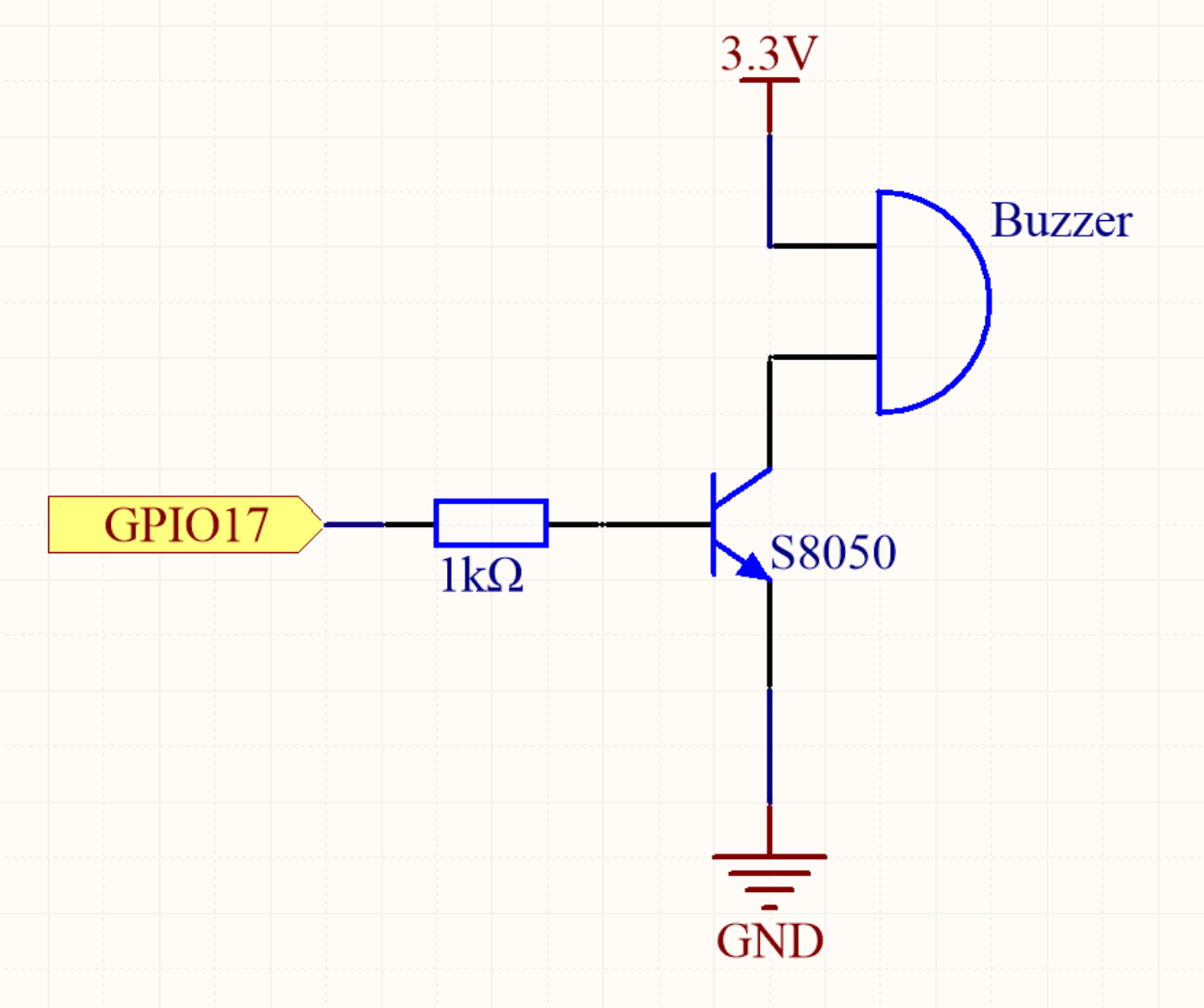
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Passive Buzzer
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(1kΩ)
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * S8050 PNP Transistor
 </td></tr>
</table>

Schaltplan

In diesem Experiment werden ein passiver Summer, ein PNP-Transistor und ein 1k-Widerstand zwischen der Basis des Transistors und GPIO verwendet, um den Transistor zu schützen.

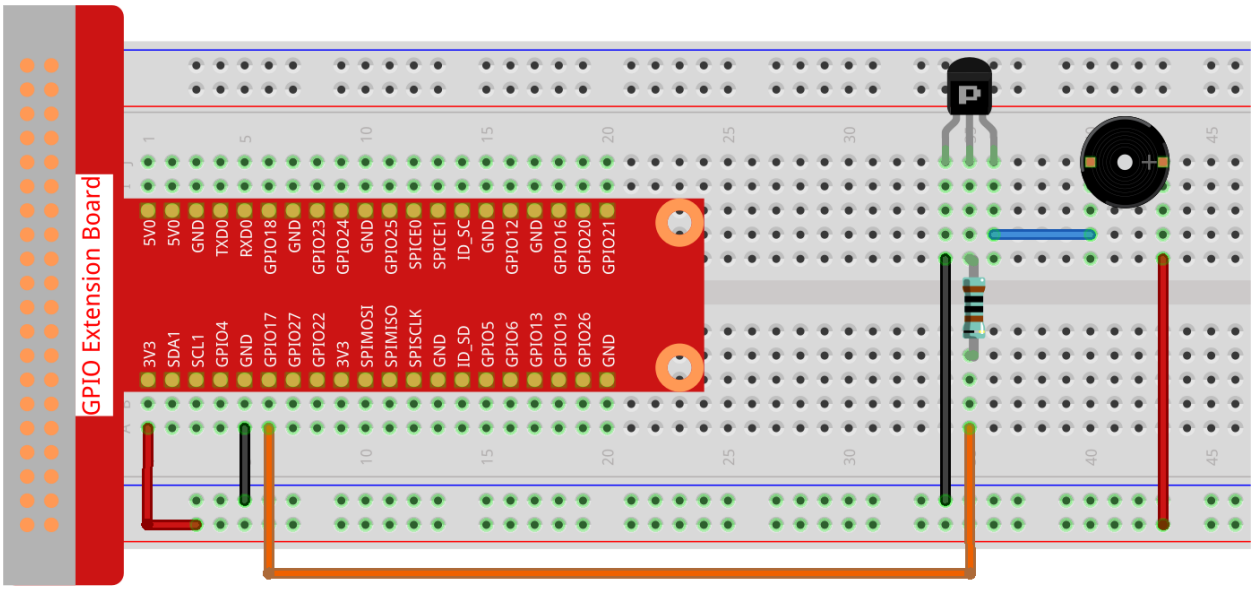
Wenn GPIO17 unterschiedliche Frequenzen erhält, gibt der passive Summer unterschiedliche Töne von sich; auf diese Weise spielt der Summer Musik.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf. (Der Passive Summer mit grüner Platine auf der Rückseite.)



Schritt 2: Verzeichnis wechseln.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.2.2_PassiveBuzzer_zero.py
```

Der Code wird ausgeführt, der Summer spielt ein Musikstück.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import TonalBuzzer
from time import sleep

# Initialisieren Sie einen TonalBuzzer am GPIO-Pin 17
tb = TonalBuzzer(17)  # Aktualisieren Sie diese Pinnummer basierend auf Ihrer Einrichtung

def play(tune):
    """
    Spielen Sie eine musikalische Melodie mit dem Summer.
    :param tune: Liste von Tupeln (Note, Dauer), wobei jedes Tupel eine Note und ihre␣
↪Dauer darstellt.
    """
    for note, duration in tune:
        print(note)  # Aktuell gespielte Note ausgeben
        tb.play(note)  # Note auf dem Summer spielen
        sleep(float(duration))  # Verzögerung für die Dauer der Note
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    tb.stop()  # Stoppen Sie das Spielen, nachdem die Melodie beendet ist

# Definieren Sie eine musikalische Melodie als eine Folge von Noten und Dauern
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)  # Führen Sie die Play-Funktion aus, um die Melodie abzuspielen

except KeyboardInterrupt:
    # Tastaturunterbrechung für eine elegante Beendigung behandeln
    pass
```

Code-Erklärung

1. Diese Zeilen importieren die Klasse `TonalBuzzer` aus der Bibliothek `gpiozero` für die Summersteuerung und die Funktion `sleep` aus dem Modul `time` für Verzögerungen.

```
#!/usr/bin/env python3
from gpiozero import TonalBuzzer
from time import sleep
```

2. Diese Zeile initialisiert ein `TonalBuzzer`-Objekt am GPIO-Pin 17.

```
# Initialisieren Sie einen TonalBuzzer am GPIO-Pin 17
tb = TonalBuzzer(17)  # Aktualisieren Sie diese Pinnummer basierend auf Ihrer␣
↪Einrichtung
```

3. Die Funktion `play` iteriert über eine Liste von Tupeln, die musikalische Noten und ihre Dauern darstellen. Jede Note wird für ihre festgelegte Dauer gespielt, und der Summer stoppt nach Abschluss der Melodie.

```
def play(tune):
    """
    Spielen Sie eine musikalische Melodie mit dem Summer.
    :param tune: Liste von Tupeln (Note, Dauer), wobei jedes Tupel eine Note und␣
↪ihre Dauer darstellt.
    """
    for note, duration in tune:
        print(note)  # Aktuell gespielte Note ausgeben
        tb.play(note)  # Note auf dem Summer spielen
        sleep(float(duration))  # Verzögerung für die Dauer der Note
    tb.stop()  # Stoppen Sie das Spielen, nachdem die Melodie beendet ist
```

4. Die Melodie ist definiert als eine Abfolge von Noten (Frequenzen) und Dauern (Sekunden).

```
# Definieren Sie eine musikalische Melodie als eine Folge von Noten und Dauern
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. Die Funktion `play(tune)` wird in einem `try`-Block aufgerufen. Eine `KeyboardInterrupt` (wie Ctrl+C) stoppt das Programm auf elegante Weise.

```
try:
    play(tune)  # Führen Sie die Play-Funktion aus, um die Melodie abzuspielen

except KeyboardInterrupt:
    # Tastaturunterbrechung für eine elegante Beendigung behandeln
    pass
```

1.3 Treiber

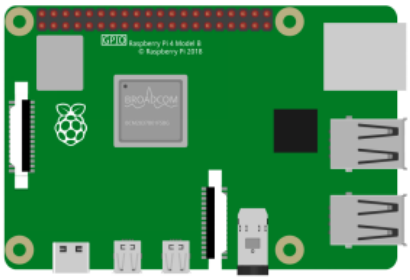
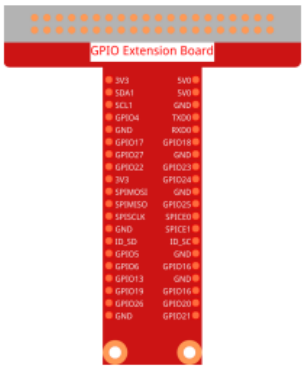
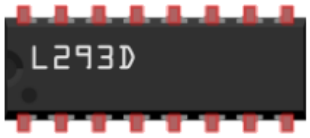
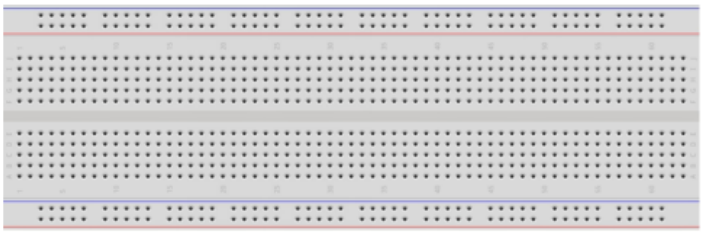
1.3.1 Motor

Einführung

In diesem Projekt werden wir lernen, wie man einen Gleichstrommotor mit einem L293D antreibt und ihn im Uhrzeigersinn und gegen den Uhrzeigersinn drehen lässt. Da der Gleichstrommotor einen größeren Strom benötigt, verwenden wir aus Sicherheitsgründen das Power Supply Modul, um die Motoren zu versorgen.

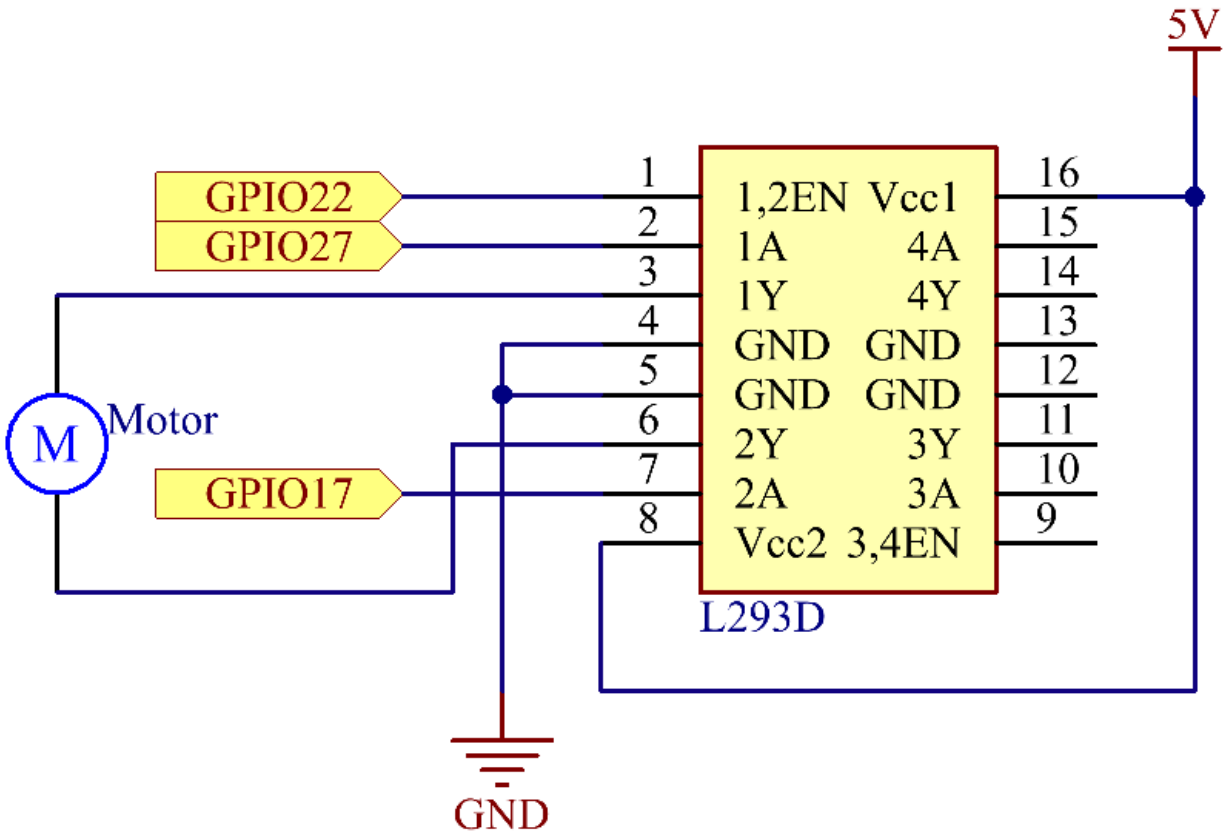
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

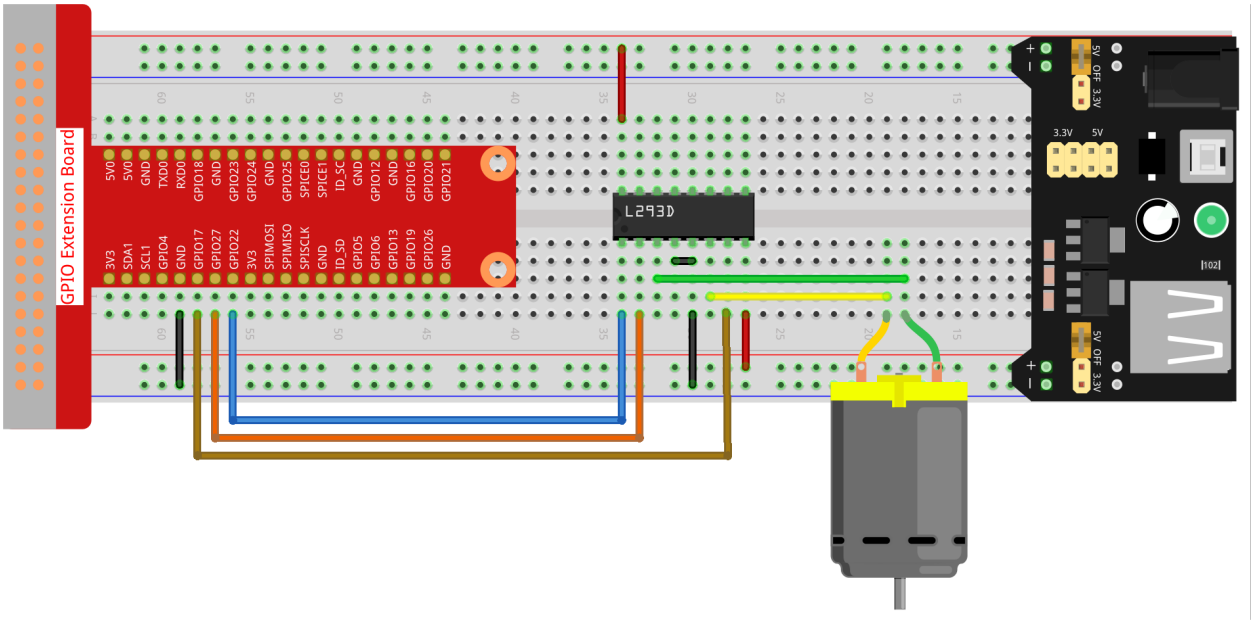
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Bemerkung: Das Power-Modul kann eine 9V-Batterie mit dem 9V-Batterie Schnalle im Kit anwenden. Stecken Sie die Jumperkappe des Power-Moduls in die 5V Busleisten des Steckbretts.

Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.3.1_Motor_zero.py
```

Wenn der Code ausgeführt wird, dreht sich der Motor zuerst 5 Sekunden lang im Uhrzeigersinn, dann stoppt er für 5 Sekunden, danach dreht er sich 5 Sekunden lang gegen den Uhrzeigersinn; anschließend stoppt der Motor für 5 Sekunden. Diese Serie von Aktionen wird wiederholt ausgeführt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**.

Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import Motor
from time import sleep

# Initialisieren Sie den Motor mit GPIO Zero, indem Sie GPIO-Pins für Vorwärts (17),␣
↪Rückwärts (27) und Enable (22) Kontrolle angeben
motor = Motor(forward=17, backward=27, enable=22)

try:
    # Hauptfunktion zur Steuerung der Richtung und Bewegung des Motors.
    # Wechselt die Drehrichtung des Motors zwischen im Uhrzeigersinn und gegen den␣
↪Uhrzeigersinn mit Stopps dazwischen.
    actions = {'CW': motor.forward, 'CCW': motor.backward, 'STOP': motor.stop}  #␣
↪Motoraktionen für Lesbarkeit definieren

    while True:
        # Durch die definierten Aktionen zyklen, um die Motordrehrichtung zu steuern
        for action in ['CW', 'STOP', 'CCW', 'STOP']:
            actions[action]()  # Führen Sie die aktuelle Aktion aus (vorwärts, stoppen,␣
↪rückwärts, stoppen)
            print(f"{action}")  # Zeigen Sie die aktuelle Aktion in der Konsole an
            sleep(5)  # Pause für 5 Sekunden, bevor Sie zur nächsten Aktion übergehen

except KeyboardInterrupt:
    # Tastaturunterbrechung (z. B. Ctrl+C) sauber behandeln
    pass
```

Code-Erklärung

1. Diese Importanweisungen holen die Klasse `Motor` aus der Bibliothek `gpiozero` und die Funktion `sleep` aus dem Modul `time`.

   ```
   #!/usr/bin/env python3
   from gpiozero import Motor
   from time import sleep
   ```

2. Diese Zeile initialisiert ein `Motor`-Objekt und gibt die GPIO-Pins für Vorwärts (17), Rückwärts (27) und Enable (22) Kontrolle an.

   ```
   # Initialisieren Sie den Motor mit GPIO Zero, indem Sie GPIO-Pins für Vorwärts (17),
   ↪ Rückwärts (27) und Enable (22) Kontrolle angeben
   motor = Motor(forward=17, backward=27, enable=22)
   ```

3. Die Aktionen zur Motorsteuerung sind in einem Wörterbuch für die Lesbarkeit definiert. Eine unendliche Schleife (*while True*) iteriert durch diese Aktionen und führt jede für 5 Sekunden aus.

   ```
   try:
       # Hauptfunktion zur Steuerung der Richtung und Bewegung des Motors.
       # Wechselt die Drehrichtung des Motors zwischen im Uhrzeigersinn und gegen den␣
   ↪Uhrzeigersinn mit Stopps dazwischen.
       actions = {'CW': motor.forward, 'CCW': motor.backward, 'STOP': motor.stop}  #␣
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪Motoraktionen für Lesbarkeit definieren

    while True:
        # Durch die definierten Aktionen zyklen, um die Motordrehrichtung zu steuern
        for action in ['CW', 'STOP', 'CCW', 'STOP']:
            actions[action]()  # Führen Sie die aktuelle Aktion aus (vorwärts,␣
↪stoppen, rückwärts, stoppen)
            print(f"{action}")  # Zeigen Sie die aktuelle Aktion in der Konsole an
            sleep(5)  # Pause für 5 Sekunden, bevor Sie zur nächsten Aktion␣
↪übergehen
```

4. Dieser Abschnitt ermöglicht es, das Programm sicher mit einer Tastaturunterbrechung (Ctrl+C) zu beenden, ohne Fehler zu verursachen.

```
except KeyboardInterrupt:
    # Tastaturunterbrechung (z. B. Ctrl+C) sauber behandeln
    pass
```

1.3.2 Servo

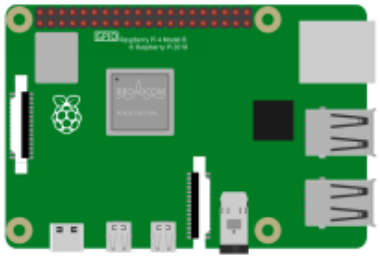
Einführung

In diesem Projekt lernen wir, wie man den Servomotor rotieren lässt.

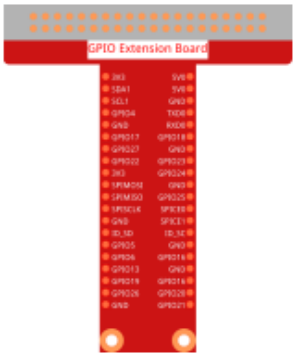
Erforderliche Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.

1 * Raspberry Pi



1 * T-Extension Board

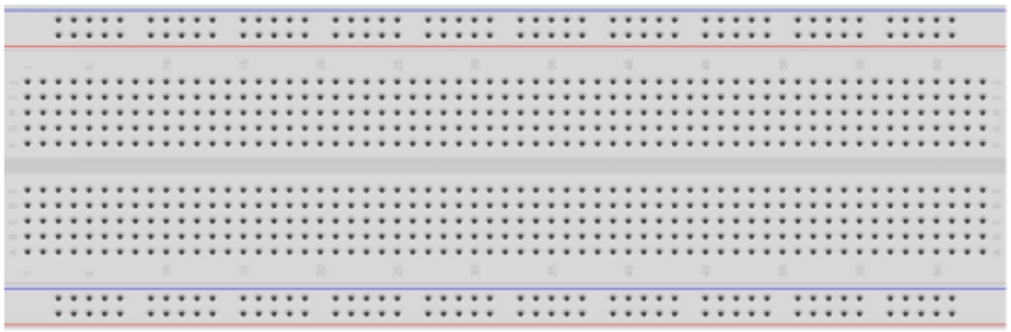


1 * Servo

Several Jumper Wires

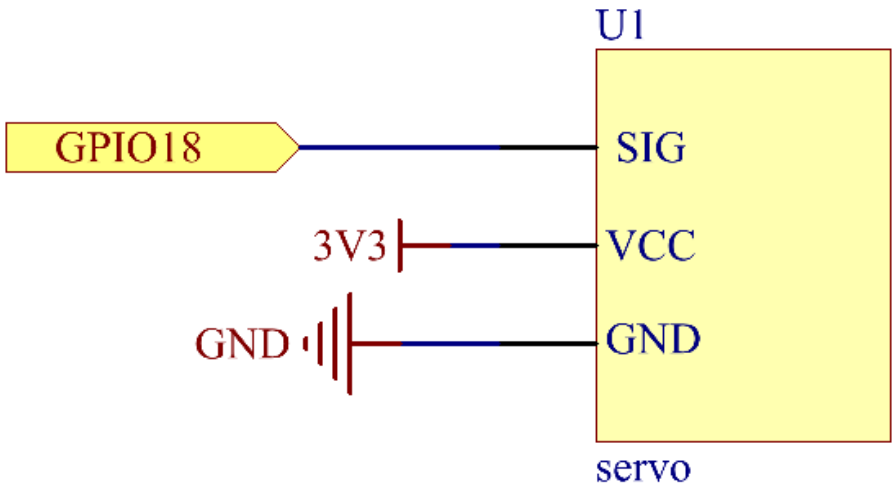
1 * 40-pin Cable

1 * Breadboard



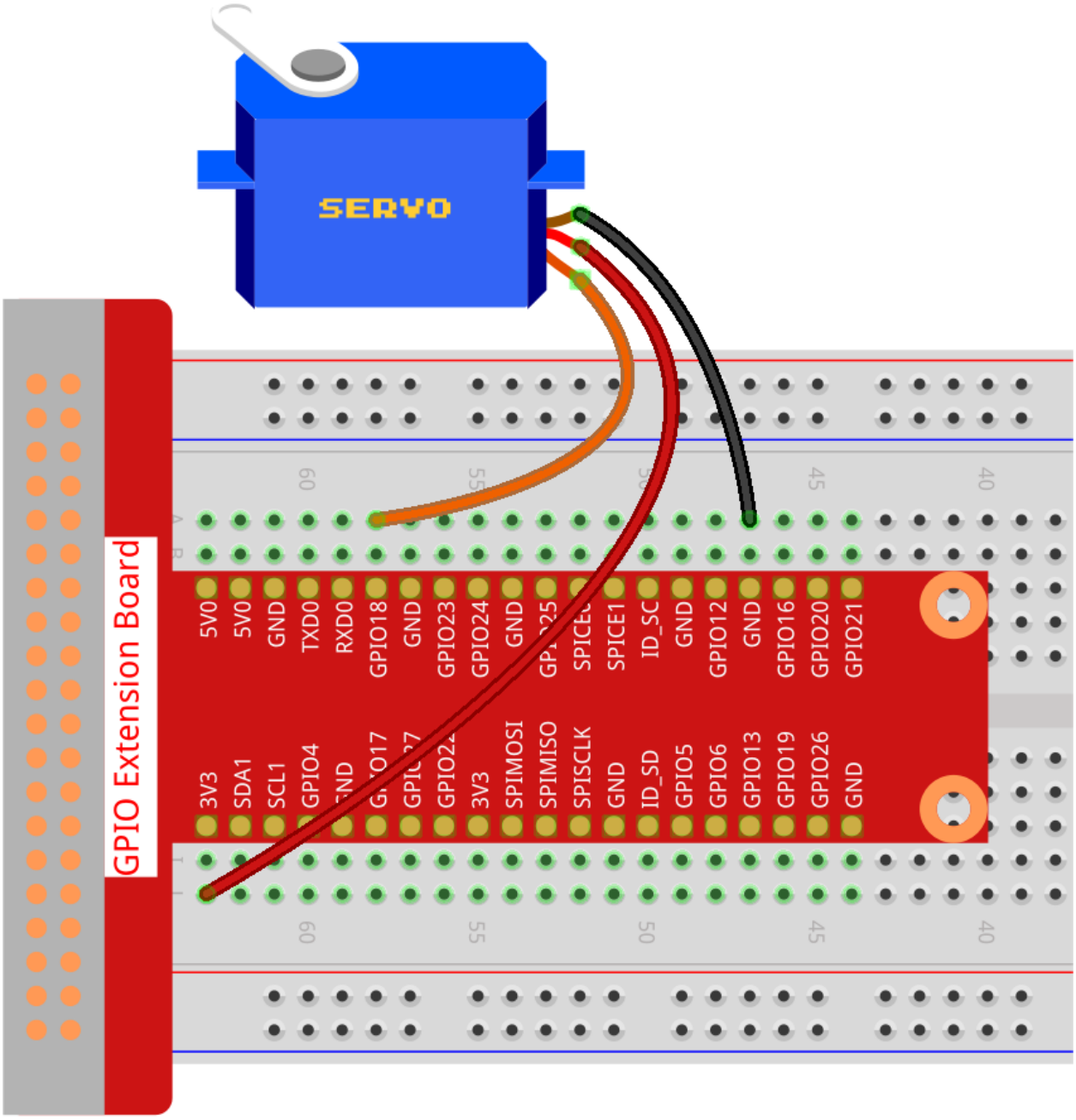
Schaltplan

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Gehen Sie in den Ordner mit dem Code.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 1.3.2_Servo_zero.py
```

Nach Ausführung des Programms wird der Servomotor sich von 0 Grad auf 90 Grad bis 180 Grad drehen und dann von 180 Grad auf 90 Grad bis 0 Grad, in einem Kreis.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nach der Änderung des Codes können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import Servo
from time import sleep

# Legen Sie die GPIO-Pin-Nummer fest, an die der Servomotor angeschlossen ist
myGPIO = 18

# Definieren Sie einen Korrekturfaktor, um die Pulsbreite des Servos feinzutunen
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # Berechnen Sie die maximale Pulsbreite
minPW = (1.0 - myCorrection) / 1000  # Berechnen Sie die minimale Pulsbreite

# Initialisieren Sie das Servo-Objekt mit benutzerdefinierten Pulsbreiten
servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)

try:
    while True:
        # Positionieren Sie den Servo in der Mitte und warten Sie
        servo.mid()
        print("Mitte")  # Aktuelle Position anzeigen
        sleep(0.5)    # Kurze Pause für 0,5 Sekunden

        # Bewegen Sie den Servo in die Minimalposition und warten Sie
        servo.min()
        print("Minimal")  # Aktuelle Position anzeigen
        sleep(1)      # Position für 1 Sekunde halten

        # Bringen Sie den Servo in die Mitte zurück und warten Sie
        servo.mid()
        print("Mitte")  # Aktuelle Position anzeigen
        sleep(0.5)    # Kurze Pause für 0,5 Sekunden

        # Bewegen Sie den Servo in die Maximalposition und warten Sie
        servo.max()
        print("Maximal")  # Aktuelle Position anzeigen
        sleep(1)      # Position für 1 Sekunde halten

except KeyboardInterrupt:
    # Beenden Sie das Skript ordnungsgemäß bei einer Tastaturunterbrechung (Strg+C)
    pass
```

Code Erklärung

1. Diese Import-Anweisungen bringen die `Servo` Klasse für die Servosteuerung und die Funktion `sleep` für die Zeitmessung.

```
#!/usr/bin/env python3
from gpiozero import Servo
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
from time import sleep
```

2. Legt die GPIO-Pin-Nummer 18 für den Anschluss des Servomotors fest.

```
# Legen Sie die GPIO-Pin-Nummer fest, an die der Servomotor angeschlossen ist
myGPIO = 18
```

3. Diese Zeilen definieren einen Korrekturfaktor und verwenden ihn, um die maximale und minimale Pulsbreite für den Servo zu berechnen, um seinen Bewegungsbereich fein abzustimmen.

```
# Definieren Sie einen Korrekturfaktor, um die Pulsbreite des Servos feinzutunen
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # Berechnen Sie die maximale Pulsbreite
minPW = (1.0 - myCorrection) / 1000  # Berechnen Sie die minimale Pulsbreite
```

4. Initialisiert das Servo-Objekt mit dem angegebenen GPIO-Pin und benutzerdefinierten Pulsbreiten.

```
# Initialisieren Sie das Servo-Objekt mit benutzerdefinierten Pulsbreiten
servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)
```

5. Der *try*-Block enthält eine *while True*-Schleife, um den Servo kontinuierlich zu bewegen. Der Servo wird in den Positionen Mitte, Minimal und Maximal positioniert, wobei jede Position gedruckt und für eine bestimmte Dauer gehalten wird.

```
try:
    while True:
        # Positionieren Sie den Servo in der Mitte und warten Sie
        servo.mid()
        print("Mitte")  # Aktuelle Position anzeigen
        sleep(0.5)    # Kurze Pause für 0,5 Sekunden

        # Bewegen Sie den Servo in die Minimalposition und warten Sie
        servo.min()
        print("Minimal")  # Aktuelle Position anzeigen
        sleep(1)      # Position für 1 Sekunde halten

        # Bringen Sie den Servo in die Mitte zurück und warten Sie
        servo.mid()
        print("Mitte")  # Aktuelle Position anzeigen
        sleep(0.5)    # Kurze Pause für 0,5 Sekunden

        # Bewegen Sie den Servo in die Maximalposition und warten Sie
        servo.max()
        print("Maximal")  # Aktuelle Position anzeigen
        sleep(1)      # Position für 1 Sekunde halten

except KeyboardInterrupt:
    # Beenden Sie das Skript ordnungsgemäß bei einer Tastaturunterbrechung (Strg+C)
    pass
```

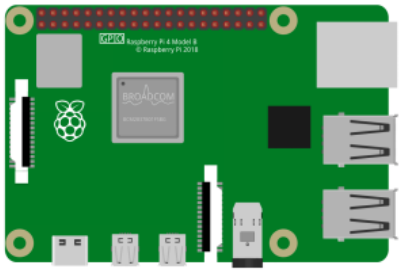
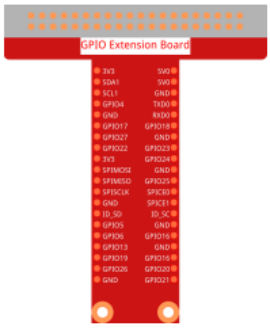
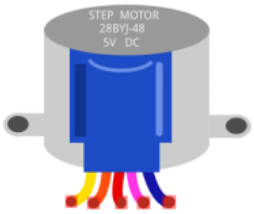
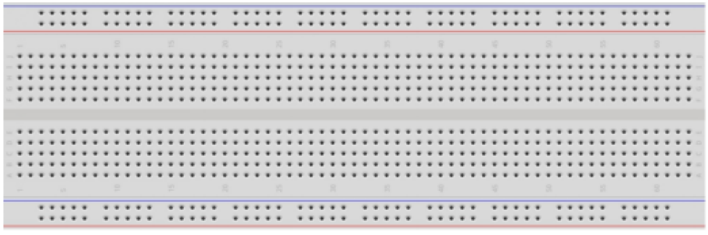
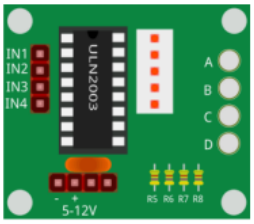
1.3.3 Schrittmotor

Einführung

Schrittmotoren können aufgrund ihrer einzigartigen Konstruktion genau gesteuert werden, ohne Rückkopplungsmechanismen zu benötigen. Die Welle eines Schrittmotors, die mit einer Reihe von Magneten versehen ist, wird von einer Reihe von elektromagnetischen Spulen gesteuert, die in einer bestimmten Sequenz positiv und negativ aufgeladen werden und sie präzise vorwärts oder rückwärts in kleinen „Schritten“ bewegen.

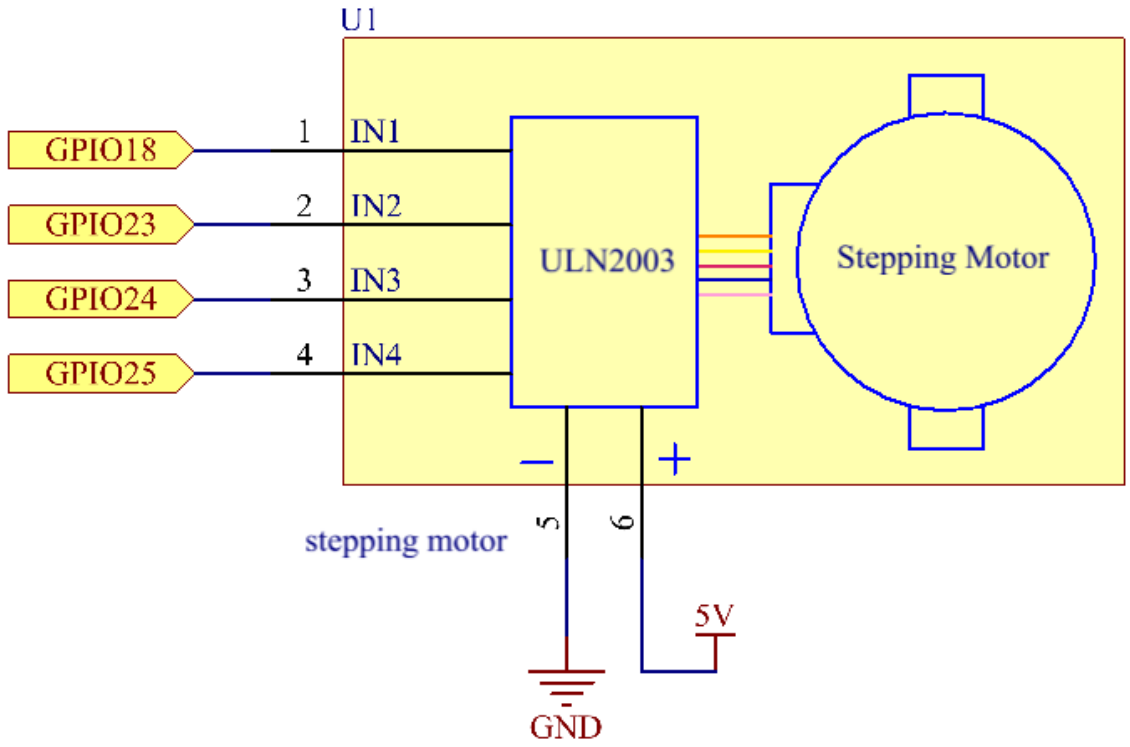
Erforderliche Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.

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

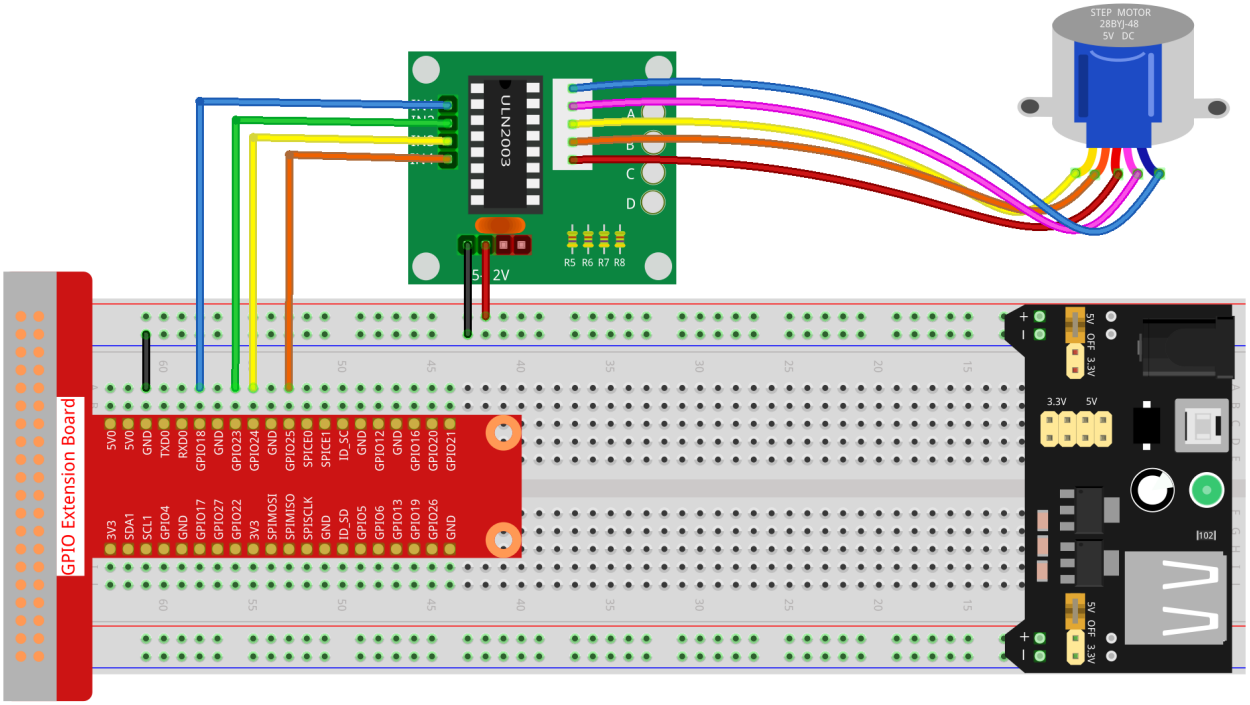
Schaltplan

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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.3.3_StepperMotor_zero.py
```

Während der Ausführung des Codes wird der Schrittmotor im Uhrzeigersinn oder gegen den Uhrzeigersinn gedreht, abhängig von Ihrer Eingabe ‚a' oder ‚c'.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nach der Änderung des Codes können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
from time import sleep

# Initialisieren Sie die Motorpins mit GPIO-Pins 18, 23, 24, 25
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]

# Setzen Sie die Motor-Drehgeschwindigkeitsparameter
UmdrehungenProMinute = 15
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
SchritteProUmdrehung = 2048
# Berechnen Sie die Wartezeit zwischen jedem Schritt für einen sanften Motorbetrieb
Schrittgeschwindigkeit = (60 / UmdrehungenProMinute) / SchritteProUmdrehung

def Schrittmotor(direction):
    """
    Steuert die Rotation des Motors basierend auf der angegebenen Richtung.

    :param direction: 'c' für im Uhrzeigersinn, 'a' für gegen den Uhrzeigersinn
    """
    if direction == 'c':
        # Führen Sie die Schrittsequenz für die Rotation im Uhrzeigersinn aus
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(Schrittgeschwindigkeit)
    elif direction == 'a':
        # Führen Sie die Schrittsequenz für die Rotation gegen den Uhrzeigersinn aus
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(Schrittgeschwindigkeit)

def Schleife():
    """
    Fordert den Benutzer kontinuierlich auf, die Motor-Richtung auszuwählen
    und steuert den Motor basierend auf dieser Eingabe.
    """
    while True:
        direction = input('Wählen Sie die Motorrichtung a=gegen den Uhrzeigersinn, c=im␣
↪Uhrzeigersinn: ')
        if direction == 'c':
            print('Motor läuft im Uhrzeigersinn\n')
            break
        elif direction == 'a':
            print('Motor läuft gegen den Uhrzeigersinn\n')
            break
        else:
            print('Eingabefehler, bitte erneut versuchen!')

    # Halten Sie den Motor in der ausgewählten Richtung in Bewegung
    while True:
        Schrittmotor(direction)

def beenden():
    """
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    Schaltet alle Motorpins sicher aus, wird für das ordnungsgemäße Herunterfahren␣
↪verwendet.
    """
    for pin in motorPin:
        pin.off()

# Hauptprogrammausführung
try:
    Schleife()
except KeyboardInterrupt:
    beenden()  # Behandeln Sie die Tastaturunterbrechung, um den Motor sicher␣
↪herunterzufahren
```

Code-Erklärung

1. Dieser Abschnitt importiert die notwendigen Bibliotheken. `gpiozero` für die Steuerung der GPIO-Pins und `time` für die in der Zeitsteuerung verwendete Funktion `sleep`.

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice
   from time import sleep
   ```

2. Initialisiert die GPIO-Pins 18, 23, 24 und 25 als Ausgabegeräte zur Steuerung des Schrittmotors.

   ```
   # Initialisieren Sie die Motorpins mit GPIO-Pins 18, 23, 24, 25
   motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]
   ```

3. Legt die Drehgeschwindigkeit des Motors fest und berechnet das Zeitintervall zwischen jedem Schritt für einen sanften Betrieb.

   ```
   # Setzen Sie die Motor-Drehgeschwindigkeitsparameter
   UmdrehungenProMinute = 15
   SchritteProUmdrehung = 2048
   # Berechnen Sie die Wartezeit zwischen jedem Schritt für einen sanften Motorbetrieb
   Schrittgeschwindigkeit = (60 / UmdrehungenProMinute) / SchritteProUmdrehung
   ```

4. Die Funktion `Schrittmotor` steuert die Rotation des Motors. Sie verwendet Bit-Manipulation und eine Schrittsequenz, um die Motorpins in der richtigen Reihenfolge für die Rotation im Uhrzeigersinn oder gegen den Uhrzeigersinn zu aktivieren.

   ```
   def Schrittmotor(direction):
       """
       Steuert die Rotation des Motors basierend auf der angegebenen Richtung.

       :param direction: 'c' für im Uhrzeigersinn, 'a' für gegen den Uhrzeigersinn
       """
       if direction == 'c':
           # Führen Sie die Schrittsequenz für die Rotation im Uhrzeigersinn aus
           for j in range(4):
               for i in range(4):
                   if 0x99 << j & (0x08 >> i):
                       motorPin[i].on()
                   else:
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                motorPin[i].off()
            sleep(Schrittgeschwindigkeit)
    elif direction == 'a':
        # Führen Sie die Schrittsequenz für die Rotation gegen den Uhrzeigersinn aus
        for j in range(4):
            for i in range(4):
                if 0x99 >> j & (0x08 >> i):
                    motorPin[i].on()
                else:
                    motorPin[i].off()
                sleep(Schrittgeschwindigkeit)
```

5. Diese Funktion fordert den Benutzer kontinuierlich auf, die Richtung der Motorrotation auszuwählen, und steuert den Motor basierend auf der Eingabe.

```
def Schleife():
    """
    Fordert den Benutzer kontinuierlich auf, die Motorrichtung auszuwählen
    und steuert den Motor basierend auf dieser Eingabe.
    """
    while True:
        direction = input('Wählen Sie die Motorrichtung a=gegen den Uhrzeigersinn,␣
↪c=im Uhrzeigersinn: ')
        if direction == 'c':
            print('Motor läuft im Uhrzeigersinn\n')
            break
        elif direction == 'a':
            print('Motor läuft gegen den Uhrzeigersinn\n')
            break
        else:
            print('Eingabefehler, bitte erneut versuchen!')

    # Halten Sie den Motor in der ausgewählten Richtung in Bewegung
    while True:
        Schrittmotor(direction)
```

6. Die beenden Funktion schaltet alle Motorpins aus. Sie wird für einen sauberen Shutdown verwendet, um sicherzustellen, dass der Motor sicher stoppt, wenn das Programm endet.

```
def beenden():
    """
    Schaltet alle Motorpins sicher aus, wird für das ordnungsgemäße Herunterfahren␣
↪verwendet.
    """
    for pin in motorPin:
        pin.off()
```

7. Das Hauptprogramm ruft Schleife auf und behandelt Tastaturunterbrechungen (wie Ctrl+C), um den Motor sicher mit beenden herunterzufahren.

```
# Hauptprogrammausführung
try:
    Schleife()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
except KeyboardInterrupt:
    beenden()  # Behandeln Sie die Tastaturunterbrechung, um den Motor sicher␣
↪herunterzufahren
```

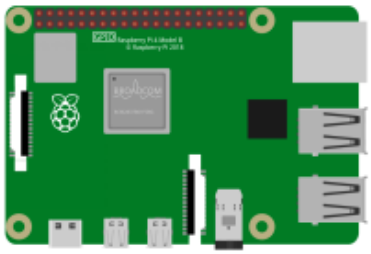
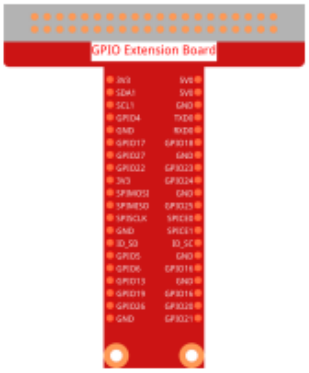
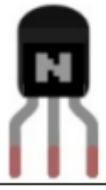
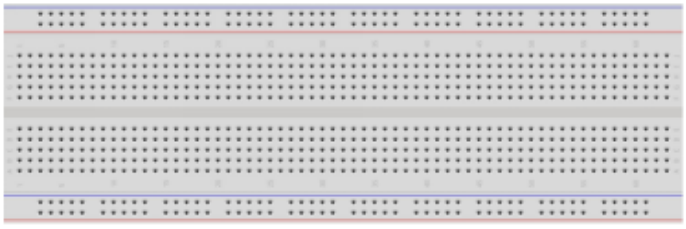
1.3.4 Relais

Einführung

In diesem Projekt lernen wir, ein Relais zu verwenden. Es ist eine der häufig verwendeten Komponenten in automatischen Steuersystemen. Wenn die Spannung, der Strom, die Temperatur, der Druck usw. den vordefinierten Wert erreicht, überschreitet oder unterschreitet, wird das Relais den Stromkreis verbinden oder unterbrechen, um die Geräte zu steuern und zu schützen.

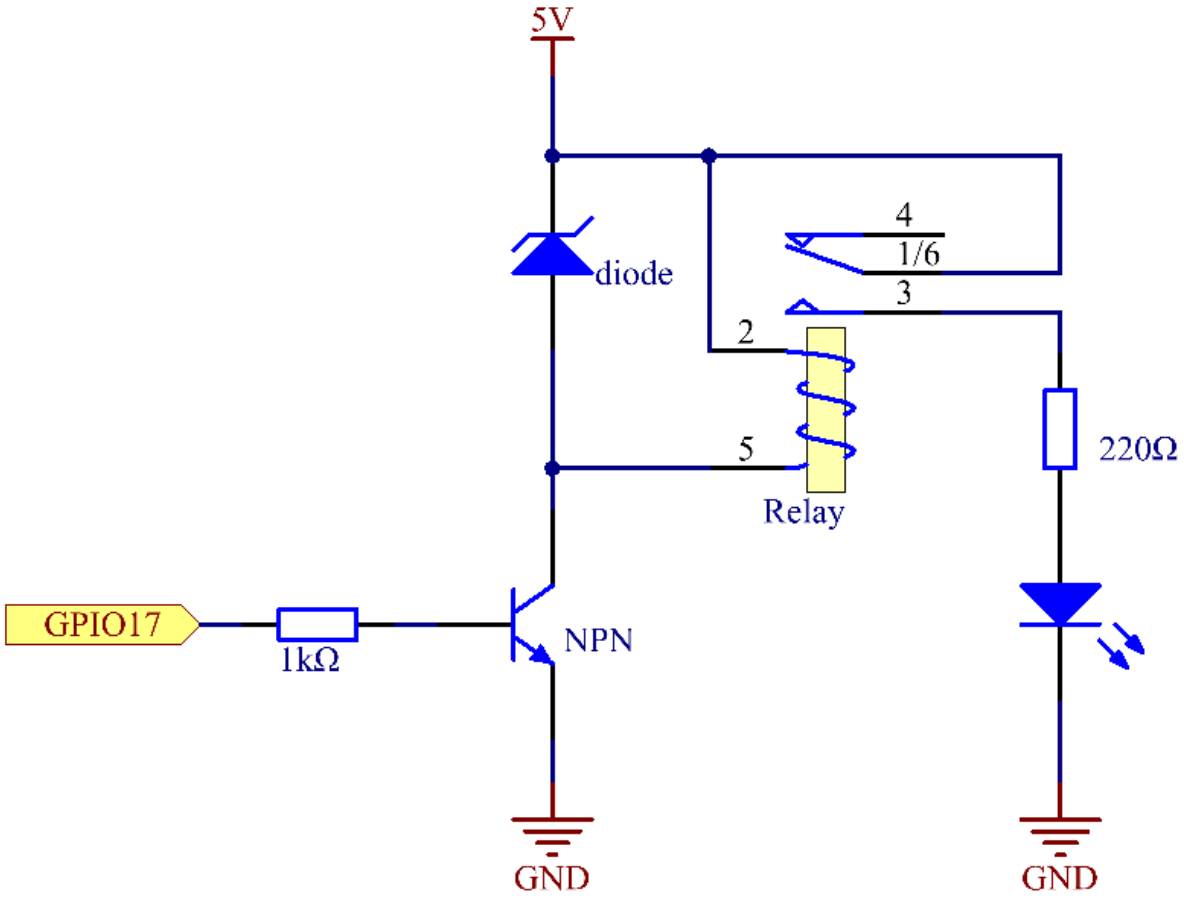
Erforderliche Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Relay
		1 * LED
		1* S8050 NPN Transistor 
1 * 40-pin Cable		1 * 1N4007 Diode
		Several Jumper Wires
1 * Breadboard 		1 * Resistor(220Ω)
		1 * Resistor(1kΩ)

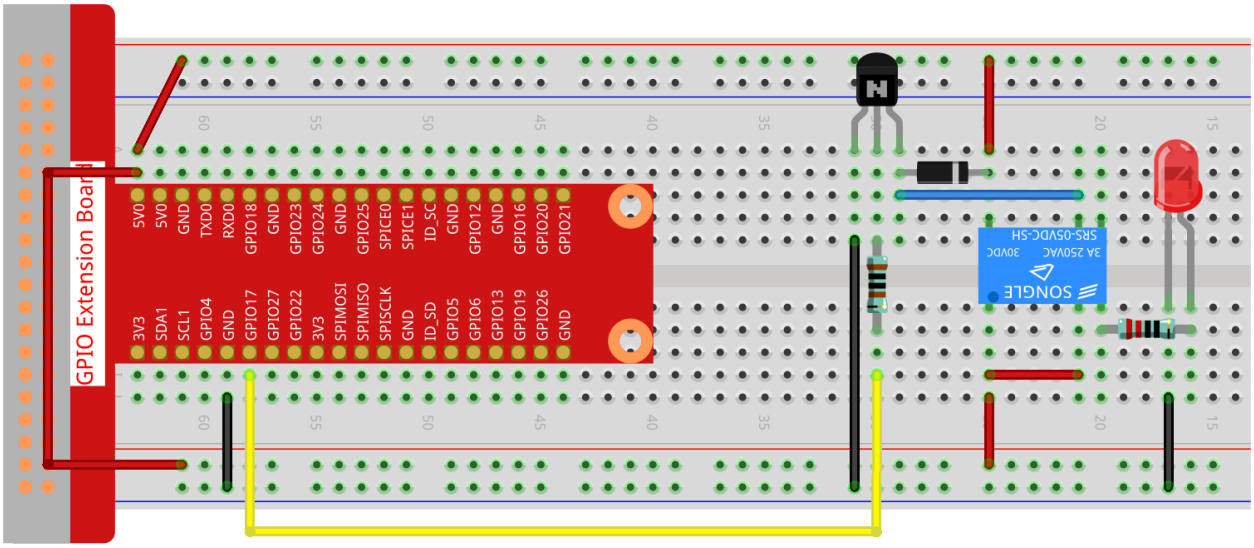
Schaltplan

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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 1.3.4_Relay_zero.py
```

Während der Ausführung des Codes leuchtet die LED auf. Darüber hinaus können Sie ein Ticken hören, verursacht durch das Unterbrechen des normalerweise geschlossenen Kontakts und das Schließen des normalerweise offenen Kontakts.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nach der Änderung des Codes können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice  # Importieren Sie die Klasse zur Steuerung der GPIO-
↪Pins
from time import sleep  # Importieren Sie die Schlaffunktion für Verzögerungen

# Initialisieren Sie das Relais, das mit dem GPIO-Pin 17 verbunden ist
relais = OutputDevice(17)

try:
    # Schleife zum kontinuierlichen Umschalten des Zustands des Relais alle Sekunde
    while True:
        print('Relais öffnet...')  # Informieren Sie, dass das Relais aktiviert wird
        relais.on()  # Schalten Sie das Relais ein (unter der Annahme einer aktiven␣
↪Niedrigkonfiguration)
        sleep(1)   # Halten Sie das Relais für 1 Sekunde im eingeschalteten Zustand

        print('...Relais schließt')  # Informieren Sie, dass das Relais deaktiviert wird
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        relais.off()  # Schalten Sie das Relais aus
        sleep(1)   # Halten Sie das Relais für 1 Sekunde im ausgeschalteten Zustand

except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (wie Ctrl+C), um aus der Schleife␣
↪auszusteigen
    relais.off()  # Stellen Sie sicher, dass das Relais ausgeschaltet ist, bevor Sie den␣
↪Vorgang beenden
    pass
```

Code-Erklärung

1. Es importiert `OutputDevice` von `gpiozero` zur Steuerung der GPIO-Pins und `sleep` von `time` zur Hinzufügung von Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice  # Importieren Sie die Klasse zur Steuerung der␣
   ↪GPIO-Pins
   from time import sleep  # Importieren Sie die Schlaffunktion für Verzögerungen
   ```

2. Initialisiert ein `OutputDevice`-Objekt für das Relais, das mit dem GPIO-Pin 17 verbunden ist.

   ```
   # Initialisieren Sie das Relais, das mit dem GPIO-Pin 17 verbunden ist
   relais = OutputDevice(17)
   ```

3. Innerhalb des `try`-Blocks schaltet eine `while True`-Schleife den Zustand des Relais kontinuierlich um. Das Relais wird ein- und ausgeschaltet, wobei zwischen jedem Zustand eine Verzögerung von 1 Sekunde besteht, begleitet von Konsolendruckanzeigen.

   ```
   try:
       # Schleife zum kontinuierlichen Umschalten des Zustands des Relais alle Sekunde
       while True:
           print('Relais öffnet...')  # Informieren Sie, dass das Relais aktiviert wird
           relais.on()  # Schalten Sie das Relais ein (unter der Annahme einer aktiven␣
   ↪Niedrigkonfiguration)
           sleep(1)   # Halten Sie das Relais für 1 Sekunde im eingeschalteten Zustand

           print('...Relais schließt')  # Informieren Sie, dass das Relais deaktiviert␣
   ↪wird
           relais.off()  # Schalten Sie das Relais aus
           sleep(1)   # Halten Sie das Relais für 1 Sekunde im ausgeschalteten Zustand
   ```

4. Fängt eine KeyboardInterrupt (wie Ctrl+C) ab, um ein ordnungsgemäßes Beenden des Skripts zu ermöglichen. Das Relais wird ausgeschaltet, bevor das Skript beendet wird.

   ```
   except KeyboardInterrupt:
   # Behandeln Sie eine Tastaturunterbrechung (wie Ctrl+C), um aus der Schleife␣
   ↪auszusteigen
   relais.off()  # Stellen Sie sicher, dass das Relais ausgeschaltet ist, bevor Sie␣
   ↪den Vorgang beenden
   pass
   ```

1.11.3 2 Eingabe

2.1 Steuerungen

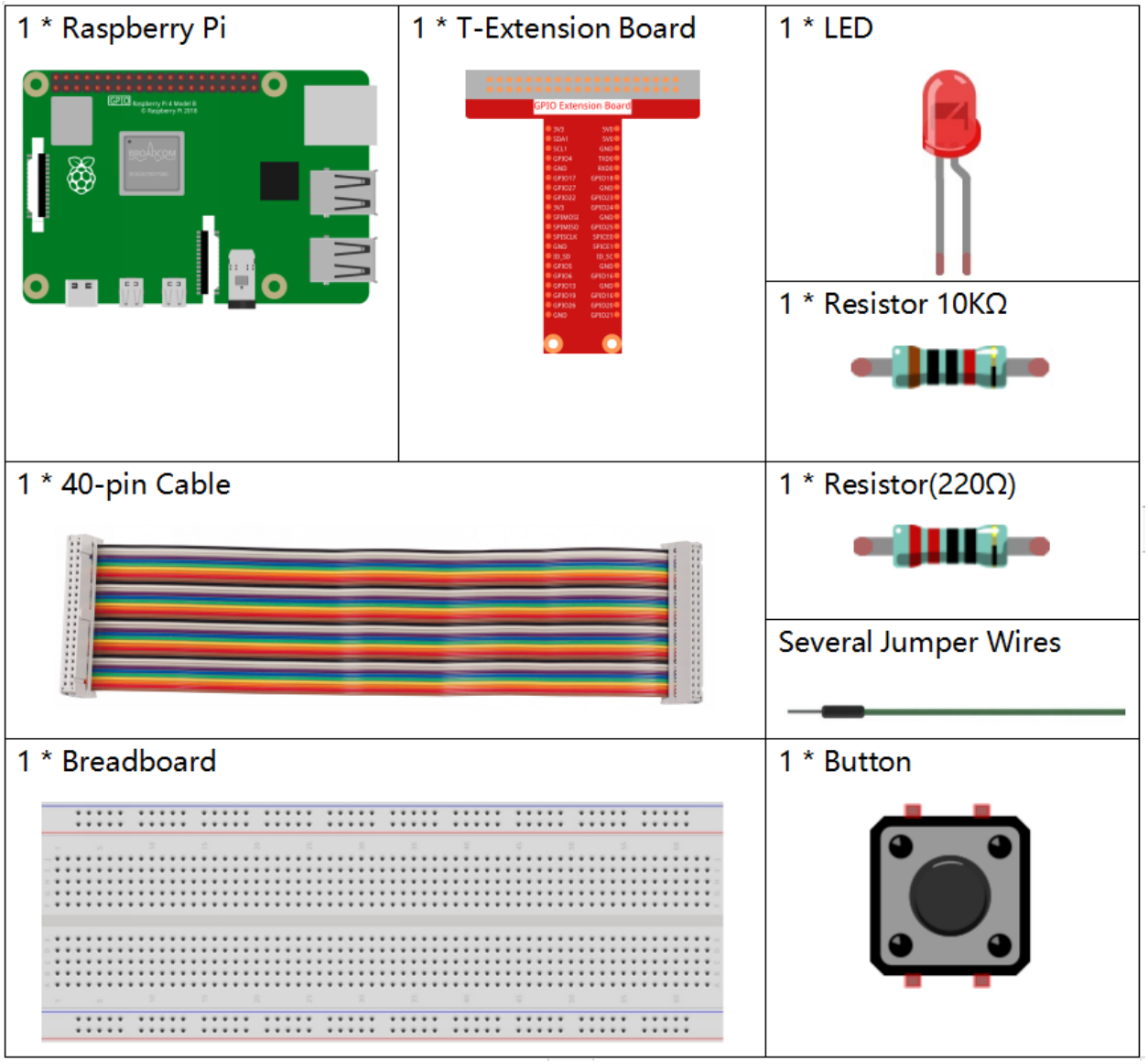
2.1.1 Taste

Einführung

In diesem Projekt lernen wir, wie man eine LED mithilfe einer Taste ein- oder ausschaltet.

Erforderliche Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.

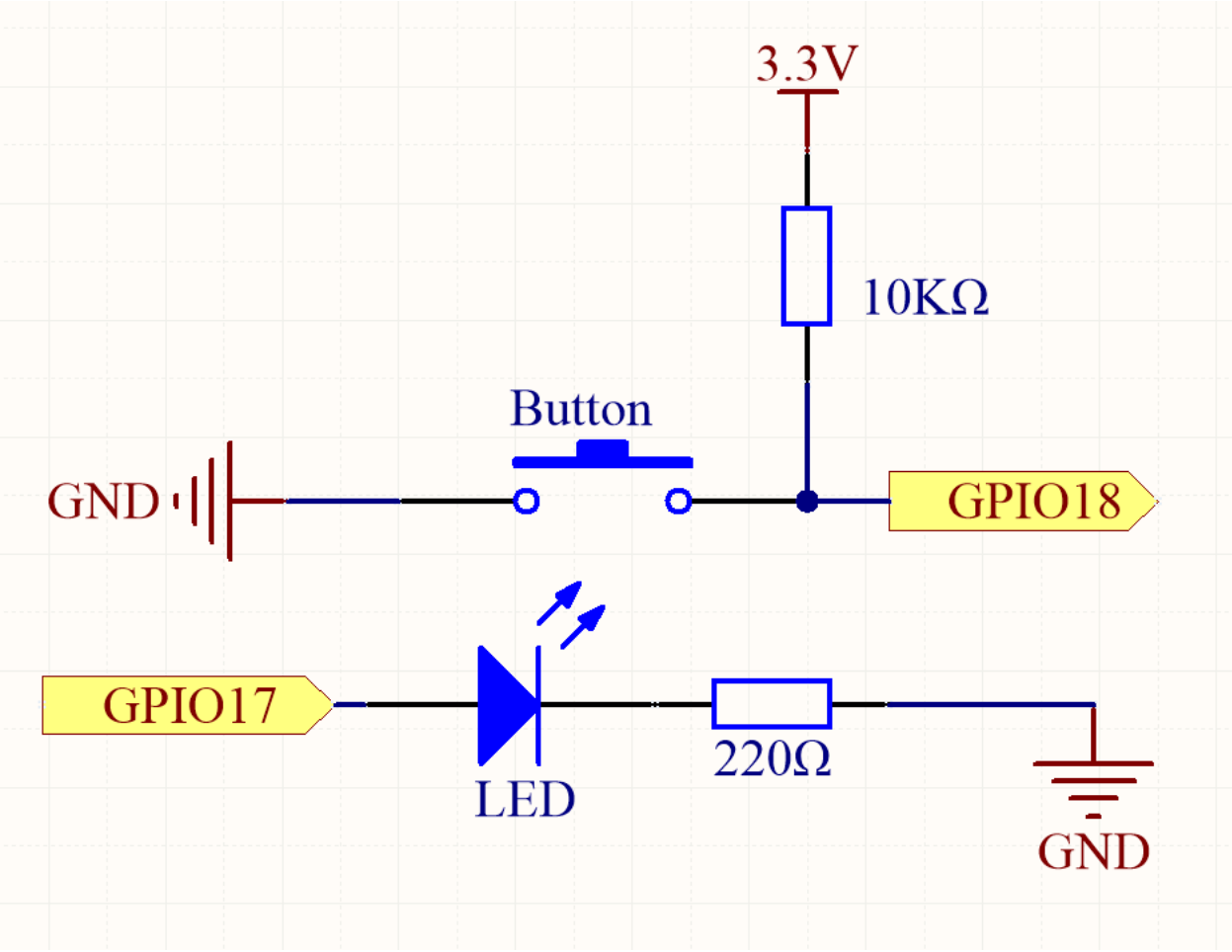


Schaltplan

Verwenden Sie eine normalerweise offene Taste als Eingang für den Raspberry Pi, die Verbindung ist im folgenden Schaltplan dargestellt. Wenn die Taste gedrückt wird, wird der GPIO18 auf ein hohes Niveau (3,3 V) geschaltet. Wir können den Zustand des GPIO18 durch Programmierung erkennen. Das bedeutet, wenn der GPIO18 auf ein hohes Niveau geschaltet wird, bedeutet dies, dass die Taste gedrückt ist. Sie können den entsprechenden Code ausführen, wenn die Taste gedrückt wird, und dann leuchtet die LED auf.

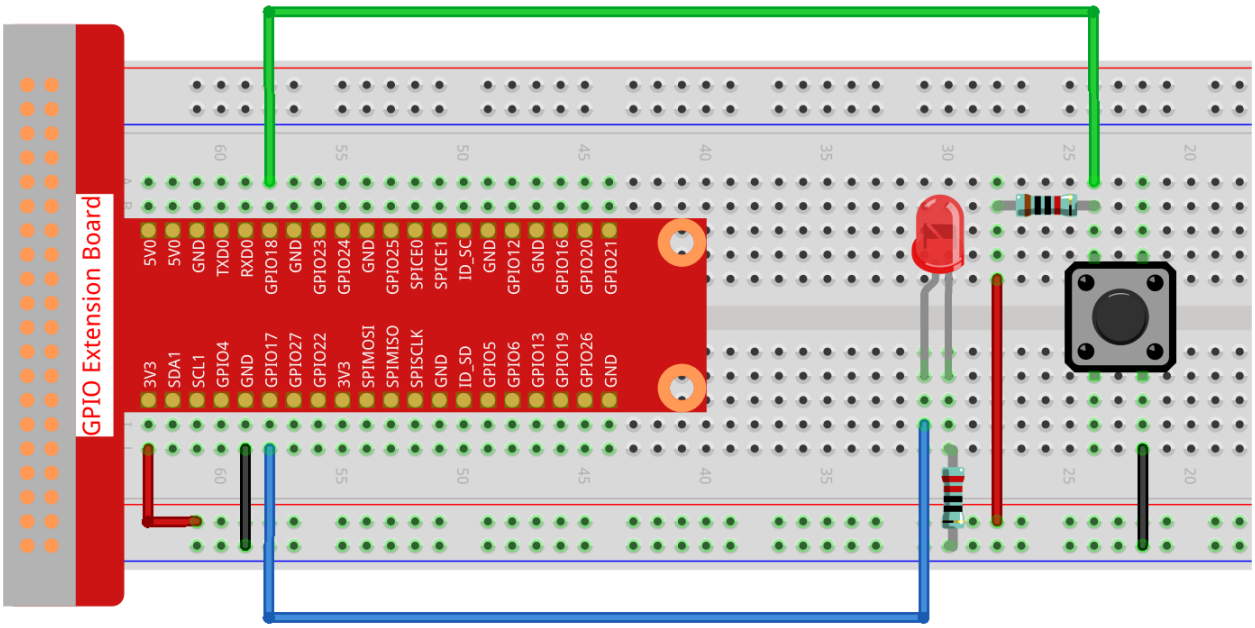
Bemerkung: Der längere Pin der LED ist die Anode und der kürzere ist die Kathode.

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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie den Code aus.

```
sudo python3 2.1.1_Button_zero.py
```

Drücken Sie jetzt die Taste und die LED leuchtet auf; lassen Sie die Taste los und die LED erlischt.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nach der Änderung des Codes können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # Importieren Sie die Klassen LED und Button aus␣
↪gpiozero
from signal import pause  # Importieren Sie die Funktion pause aus dem Modul signal

# Initialisieren Sie ein LED-Objekt am GPIO-Pin 17
led = LED(17)
# Initialisieren Sie ein Button-Objekt am GPIO-Pin 18
button = Button(18)

# Verknüpfen Sie das "when_pressed"-Ereignis der Taste mit der Methode on() der LED
button.when_pressed = led.on
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Verknüpfen Sie das "when_released"-Ereignis der Taste mit der Methode off() der LED
button.when_released = led.off

# Führen Sie eine Ereignisschleife aus, die auf Tastenereignisse wartet und das Skript␣
↪am Laufen hält
pause()
```

Code-Erklärung

1. Das Skript ist in Python3 geschrieben und importiert die Klassen `LED` und `Button` aus der Bibliothek `gpiozero` sowie `pause` aus dem Modul `signal`.

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # Importieren Sie die Klassen LED und Button aus␣
↪gpiozero
from signal import pause  # Importieren Sie die Funktion pause aus dem Modul signal
```

2. Initialisiert ein `LED`-Objekt am GPIO-Pin 17 und ein `Button`-Objekt am GPIO-Pin 18.

```
# Initialisieren Sie ein LED-Objekt am GPIO-Pin 17
led = LED(17)
# Initialisieren Sie ein Button-Objekt am GPIO-Pin 18
button = Button(18)
```

3. Richtet Ereignishandler ein, damit die LED eingeschaltet wird, wenn die Taste gedrückt wird, und ausgeschaltet wird, wenn die Taste losgelassen wird.

```
# Verknüpfen Sie das "when_pressed"-Ereignis der Taste mit der Methode on() der LED
button.when_pressed = led.on
# Verknüpfen Sie das "when_released"-Ereignis der Taste mit der Methode off() der␣
↪LED
button.when_released = led.off
```

4. Dies hält das Skript in einer Ereignisschleife, die auf Tasten-Druck- und Loslassen-Ereignisse wartet.

```
# Führen Sie eine Ereignisschleife aus, die auf Tastenereignisse wartet und das␣
↪Skript am Laufen hält
pause()
```

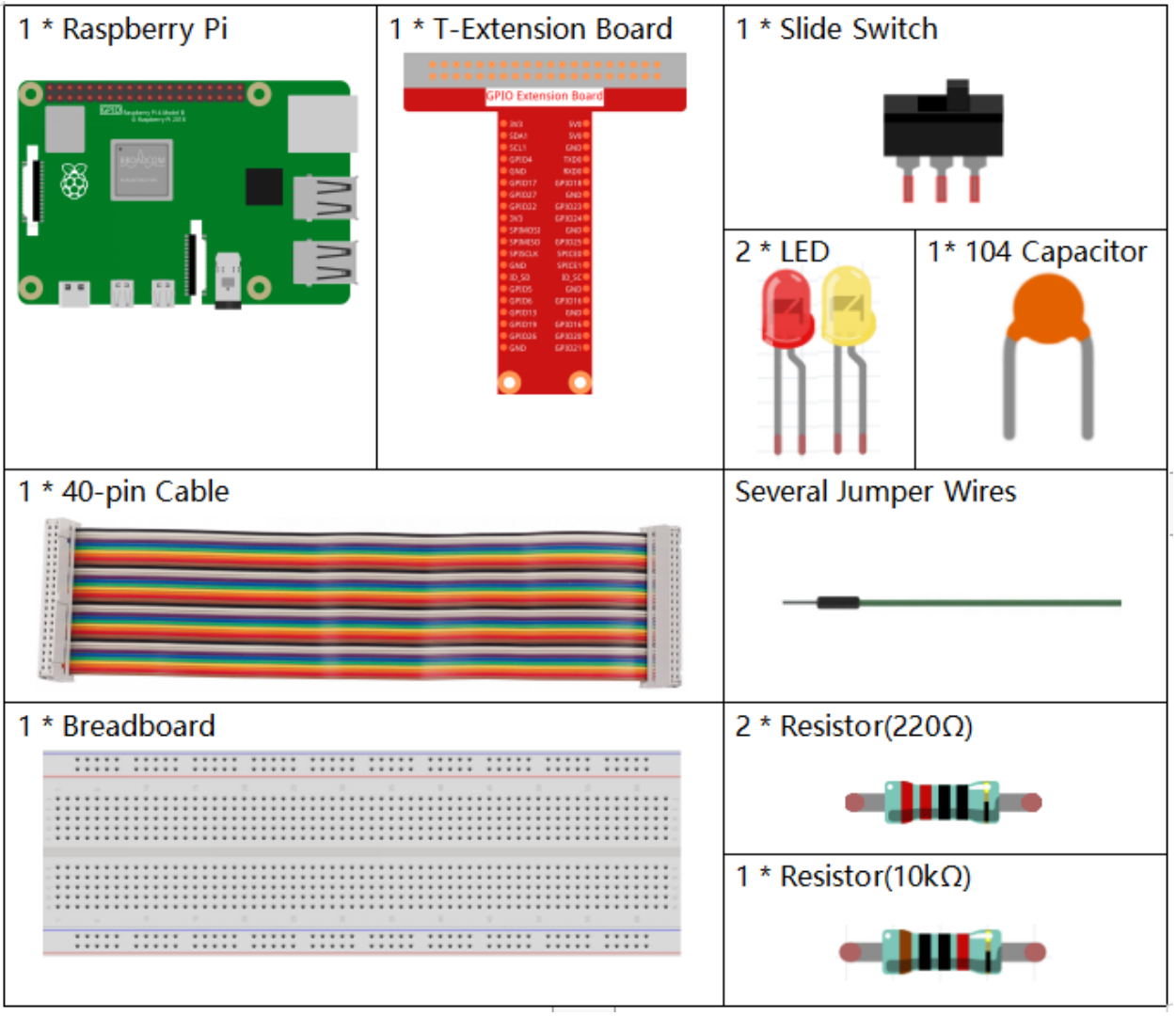
2.1.2 Schiebeschalter

Einführung

In diesem Projekt lernen wir, wie man einen Schiebeschalter verwendet. Normalerweise wird der Schiebeschalter als Netzschalter auf einer Leiterplatte (PCB) gelötet, aber hier müssen wir ihn in das Steckbrett einsetzen, daher kann er möglicherweise nicht befestigt werden. Und wir verwenden ihn auf dem Steckbrett, um seine Funktion zu zeigen.

Erforderliche Komponenten

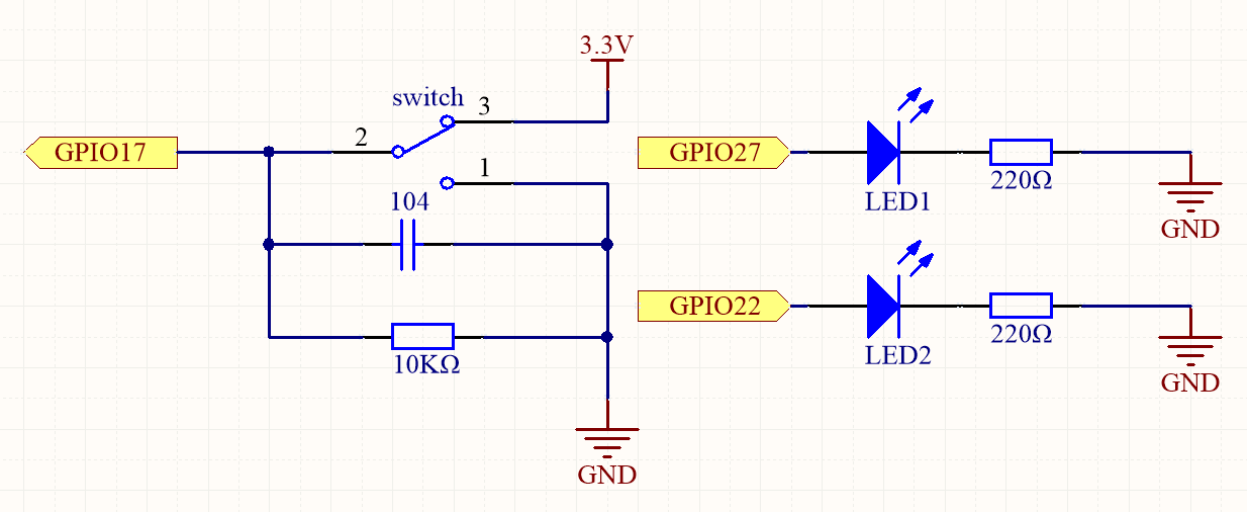
Für dieses Projekt benötigen wir folgende Komponenten.



Schaltplan

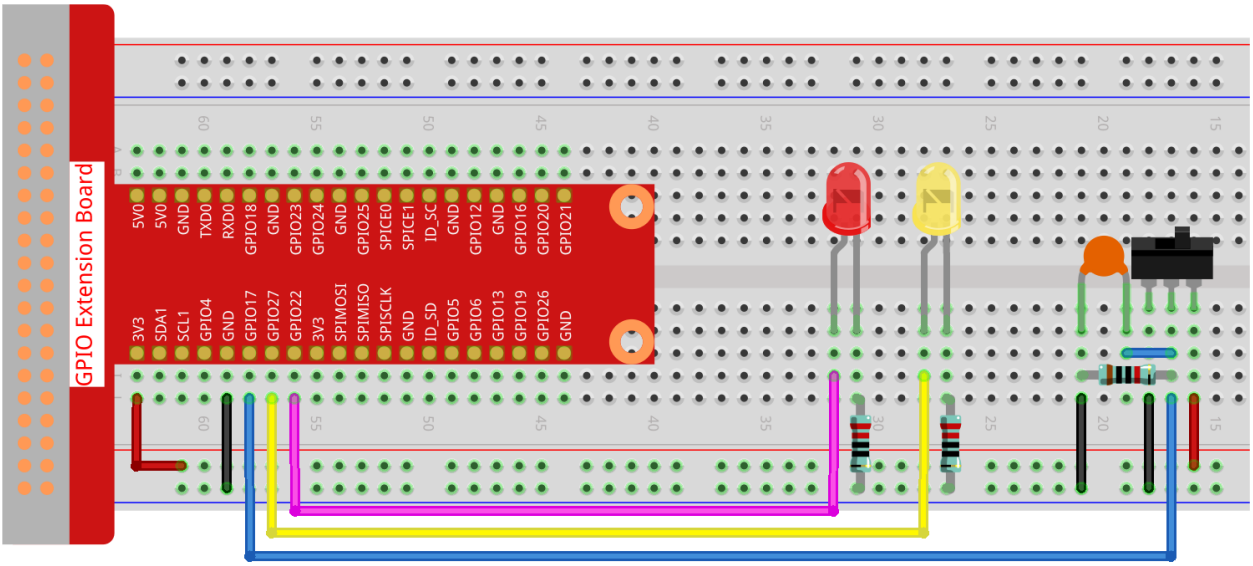
Verbinden Sie den mittleren Stift des Schiebeschalters mit GPIO17 und zwei LEDs mit den Pins GPIO22 und GPIO27. Wenn Sie den Schieber ziehen, sehen Sie, dass die beiden LEDs abwechselnd aufleuchten.

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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Wechseln Sie in den Ordner mit dem Code.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie den Code aus.

```
sudo python3 2.1.2_Slider_zero.py
```

Während der Code ausgeführt wird, schließen Sie den Schalter an die linke Seite an, dann leuchtet die gelbe LED auf; schließen Sie ihn an die rechte Seite an, dann leuchtet die rote LED auf.

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nach der Änderung des Codes können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED, Button  # Importieren Sie die Klassen LED und Button für die␣
↪GPIO-Steuerung
from time import sleep  # Importieren Sie die Funktion sleep für Verzögerungen

# Initialisieren Sie den Mikroschalter am GPIO-Pin 17 mit deaktiviertem Pull-up-
↪Widerstand
micro_switch = Button(17, pull_up=False)
# Initialisieren Sie LED1 am GPIO-Pin 22
led1 = LED(22)
# Initialisieren Sie LED2 am GPIO-Pin 27
led2 = LED(27)

try:
    # Hauptschleife zur Steuerung der LED-Zustände basierend auf dem Zustand des␣
↪Mikroschalters
    while True:
        if micro_switch.is_pressed:  # Überprüfen Sie, ob der Mikroschalter gedrückt ist
            print('    LED1 AN    ')  # Statusmeldung drucken
            led1.on()   # Schalten Sie LED1 ein
            led2.off()  # Schalten Sie LED2 aus
        else:  # Wenn der Mikroschalter nicht gedrückt ist
            print('    LED2 AN    ')  # Statusmeldung drucken
            led1.off()  # Schalten Sie LED1 aus
            led2.on()   # Schalten Sie LED2 ein

        sleep(0.5)  # Warten Sie 0,5 Sekunden, bevor Sie den Schalterzustand erneut␣
↪überprüfen

except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (Strg+C) für einen sauberen Ausstieg aus␣
↪der Schleife
    pass
```

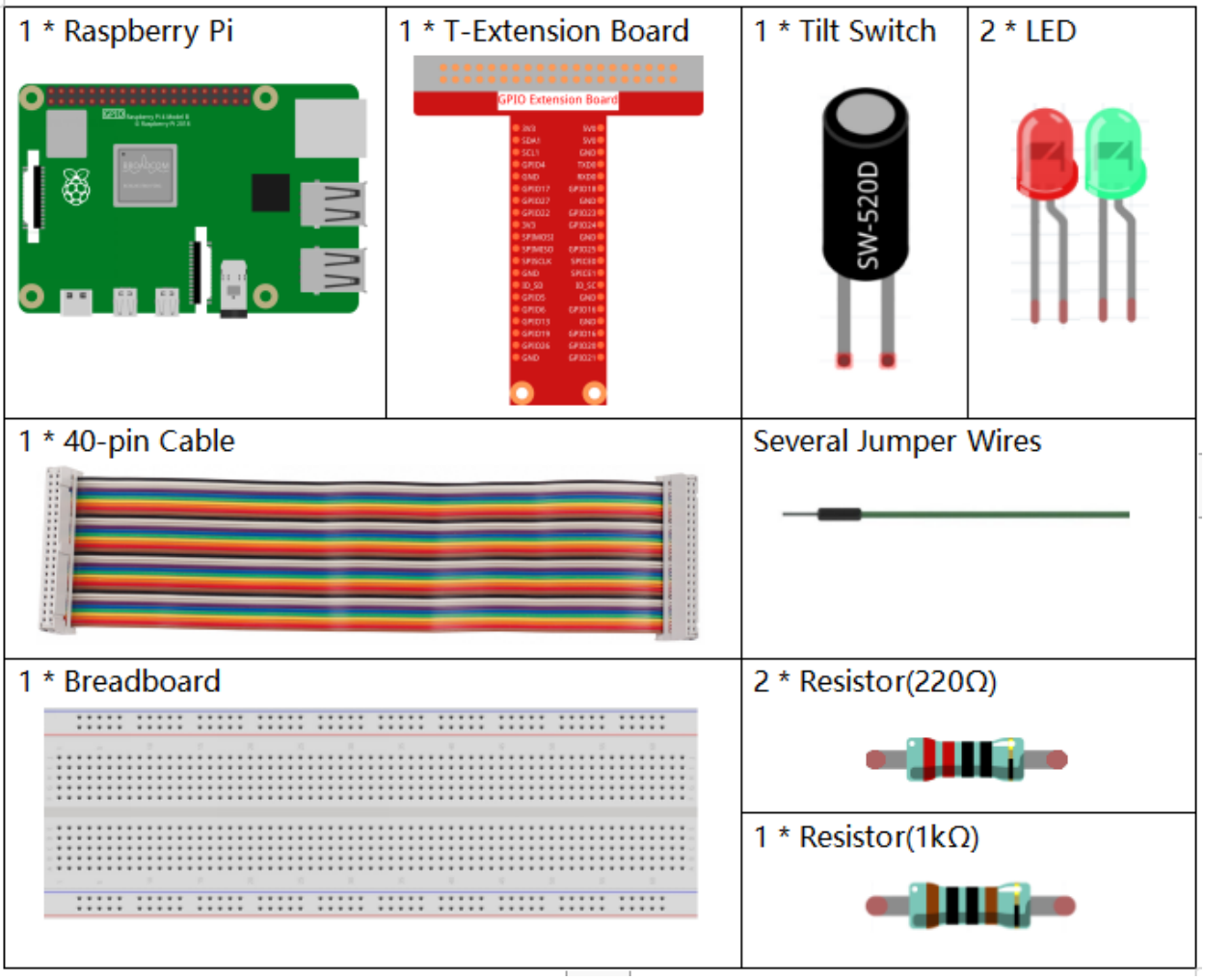
2.1.3 Neigungsschalter

Einführung

Dies ist ein Kugel-Neigungsschalter mit einer Metallkugel im Inneren. Er wird verwendet, um Neigungen eines kleinen Winkels zu erkennen.

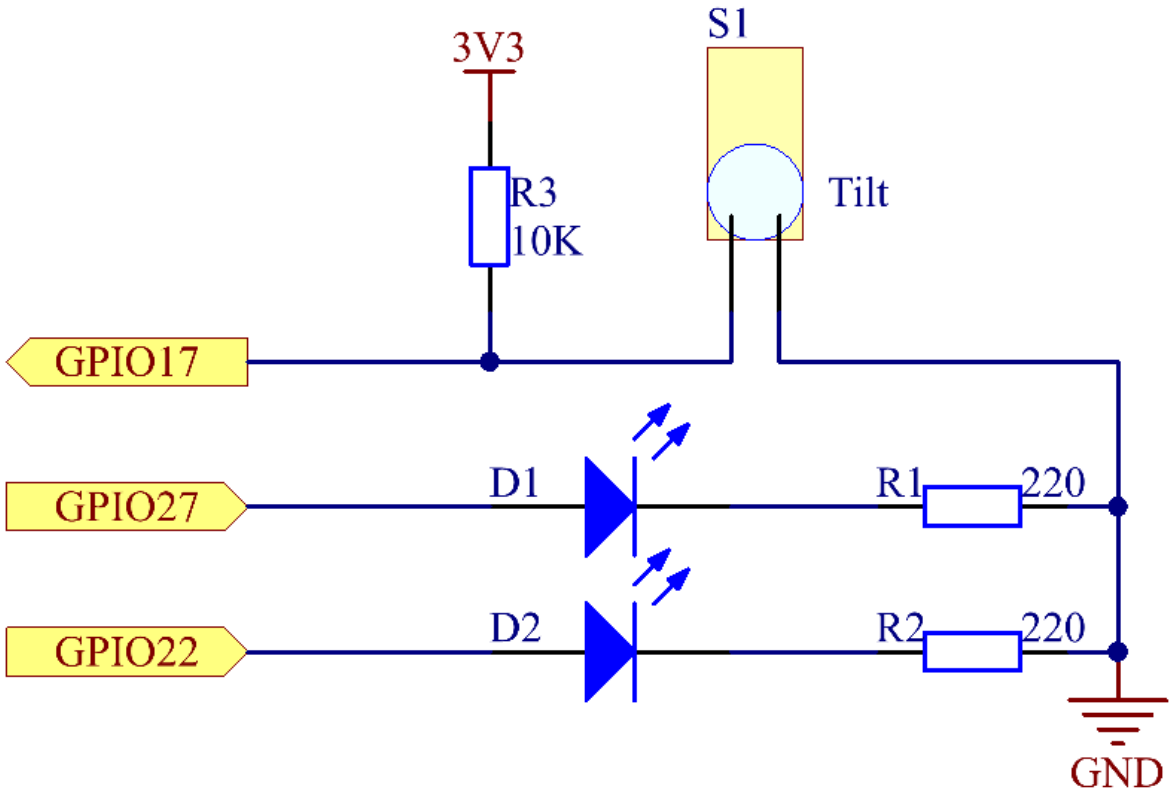
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.



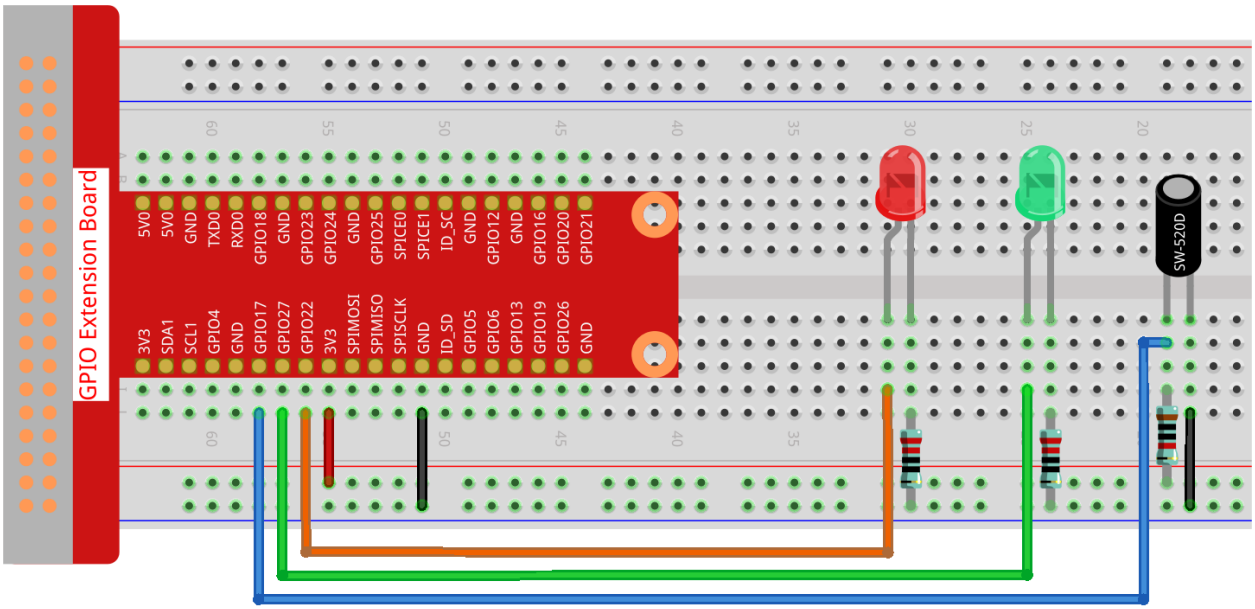
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 2.1.3_Tilt_zero.py
```

Stellen Sie den Neigungsschalter senkrecht auf, und die grüne LED leuchtet auf. Wenn Sie ihn neigen, wird „Tilt!“ auf dem Bildschirm angezeigt, und die rote LED leuchtet auf. Stellen Sie ihn wieder senkrecht auf, und die grüne LED leuchtet.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED, Button

# Initialisieren Sie den Button für den Neigungssensor und die LEDs mit GPIO Zero
TiltPin = Button(17, pull_up=False)  # Neigungssensor an GPIO-Pin 17 angeschlossen, Pull-
↪up-Widerstand deaktiviert
green_led = LED(27)  # Grüne LED an GPIO-Pin 27 angeschlossen
red_led = LED(22)   # Rote LED an GPIO-Pin 22 angeschlossen

def detect():
    """
    Erkennen Sie den Zustand des Neigungssensors und steuern Sie die LEDs.
    Schaltet die rote LED ein und die grüne LED aus, wenn geneigt.
    Schaltet die rote LED aus und die grüne LED ein, wenn nicht geneigt.
    """
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    if TiltPin.is_pressed:  # Überprüfen, ob der Sensor geneigt ist
        print('    **************')
        print('    *   Tilt!   *')
        print('    **************')
        red_led.on()   # Rote LED einschalten
        green_led.off()  # Grüne LED ausschalten
    else:  # Wenn der Sensor nicht geneigt ist
        red_led.off()  # Rote LED ausschalten
        green_led.on()  # Grüne LED einschalten

try:
    while True:
        # Kontinuierlich den Zustand des Neigungssensors überprüfen und LEDs␣
↪aktualisieren
        TiltPin.when_pressed = detect
        TiltPin.when_released = detect

except KeyboardInterrupt:
    # Tastaturunterbrechung (z. B. Ctrl+C) sauber behandeln
    pass
```

Code-Erklärung

1. Diese Zeile setzt das Skript so, dass es mit Python 3 läuft, und importiert `LED` und `Button` aus `gpiozero` zur Steuerung von GPIO-Geräten.

   ```
   #!/usr/bin/env python3
   from gpiozero import LED, Button
   ```

2. Initialisiert den Neigungssensor, der an GPIO-Pin 17 angeschlossen ist (mit deaktiviertem Pull-up-Widerstand), und zwei LEDs, die an GPIO-Pins 27 und 22 angeschlossen sind.

   ```
   # Initialisieren Sie den Button für den Neigungssensor und die LEDs mit GPIO Zero
   TiltPin = Button(17, pull_up=False)  # Neigungssensor an GPIO-Pin 17 angeschlossen,␣
   ↪Pull-up-Widerstand deaktiviert
   green_led = LED(27)  # Grüne LED an GPIO-Pin 27 angeschlossen
   red_led = LED(22)   # Rote LED an GPIO-Pin 22 angeschlossen
   ```

3. Definiert die Funktion `detect`, die den Zustand des Neigungssensors überprüft. Wenn geneigt, schaltet sie die rote LED ein und die grüne LED aus. Wenn nicht geneigt, macht sie das Gegenteil.

   ```
   def detect():
       """
       Erkennen Sie den Zustand des Neigungssensors und steuern Sie die LEDs.
       Schaltet die rote LED ein und die grüne LED aus, wenn geneigt.
       Schaltet die rote LED aus und die grüne LED ein, wenn nicht geneigt.
       """
       if TiltPin.is_pressed:  # Überprüfen, ob der Sensor geneigt ist
           print('    **************')
           print('    *   Tilt!   *')
           print('    **************')
           red_led.on()   # Rote LED einschalten
           green_led.off()  # Grüne LED ausschalten
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    else:  # Wenn der Sensor nicht geneigt ist
        red_led.off()  # Rote LED ausschalten
        green_led.on()  # Grüne LED einschalten
```

4. Die Hauptschleife weist die Funktion `detect` sowohl den `when_pressed`- als auch den `when_released`-Ereignissen des Neigungssensors zu. Der `try-except`-Block behandelt eine KeyboardInterrupt für einen sauberen Abbruch.

```
try:
    while True:
        # Kontinuierlich den Zustand des Neigungssensors überprüfen und LEDs␣
↪aktualisieren
        TiltPin.when_pressed = detect
        TiltPin.when_released = detect

except KeyboardInterrupt:
    # Tastaturunterbrechung (z. B. Ctrl+C) sauber behandeln
    pass
```

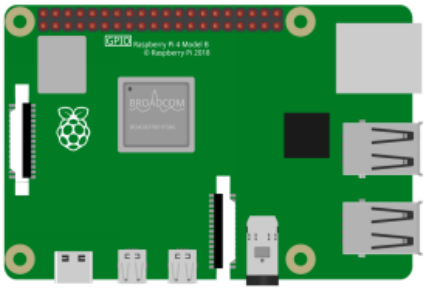
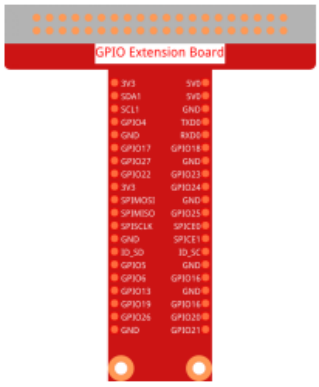
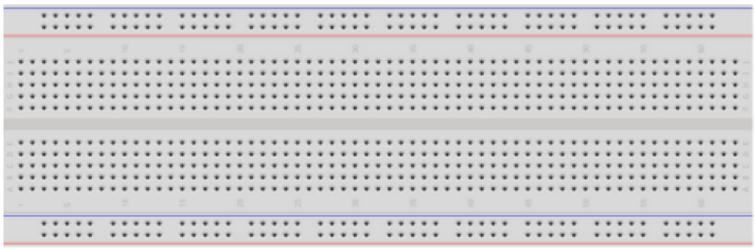
2.1.4 Potentiometer

Einführung

Die ADC-Funktion kann verwendet werden, um analoge Signale in digitale Signale umzuwandeln, und in diesem Experiment wird der ADC0834 verwendet, um die Funktion mit ADC zu bekommen. Hier implementieren wir diesen Prozess mit einem Potentiometer. Das Potentiometer ändert die physikalische Größe – die Spannung, die durch die ADC-Funktion umgewandelt wird.

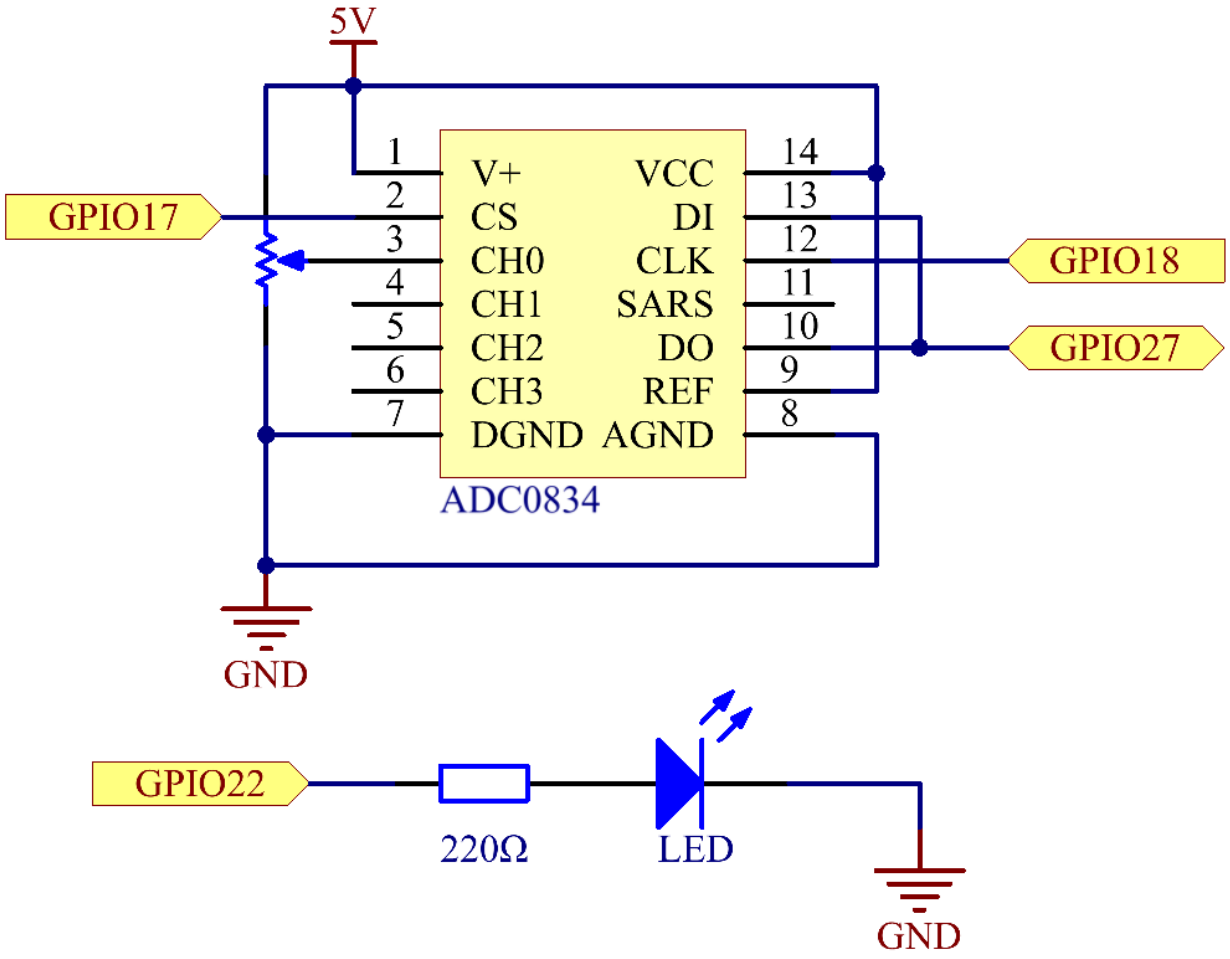
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

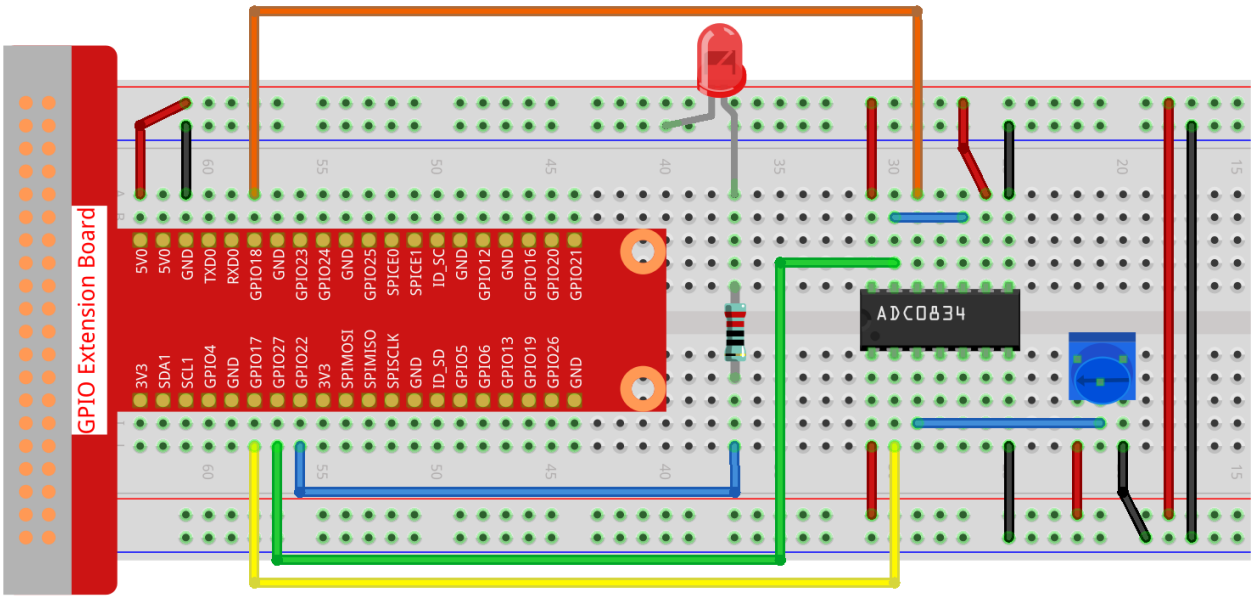
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Bemerkung: Bitte platzieren Sie den Chip, indem Sie sich auf die entsprechende Position im Bild beziehen. Beachten Sie, dass die Kerben am Chip links sein sollten, wenn er platziert wird.

Schritt 2: Öffnen Sie die Code-Datei

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 2.1.4_Potentiometer_zero.py
```

Nachdem der Code ausgeführt wurde, drehen Sie den Knopf am Potentiometer, die Intensität der LED wird entsprechend ändern.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

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

from gpiozero import PWMLED
import ADC0834
import time

# Initialisieren Sie eine PWM-LED am GPIO-Pin 22
led = PWMLED(22)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Richten Sie das ADC0834-Modul ein
ADC0834.setup()

def MAP(x, in_min, in_max, out_min, out_max):
    """
    Ordnen Sie einen Wert von einem Bereich in einen anderen zu.
    :param x: Der umzuordnende Wert.
    :param in_min: Die untere Grenze des aktuellen Bereichs des Werts.
    :param in_max: Die obere Grenze des aktuellen Bereichs des Werts.
    :param out_min: Die untere Grenze des Zielbereichs des Werts.
    :param out_max: Die obere Grenze des Zielbereichs des Werts.
    :return: Der umgeordnete Wert.
    """
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

try:
    while True:
        # Holen Sie den aktuellen Messwert vom ADC0834-Modul
        res = ADC0834.getResult()
        print('res = %d' % res)

        # Ordnen Sie den ADC-Wert einem Bereich zu, der für die Einstellung der LED-
↪Helligkeit geeignet ist
        R_val = MAP(res, 0, 255, 0, 100)

        # Stellen Sie die Helligkeit der LED ein
        led.value = float(R_val / 100)

        # Warten Sie 0,2 Sekunden, bevor Sie erneut messen
        time.sleep(0.2)

# Eleganter Ausstieg, wenn 'Ctrl+C' gedrückt wird
except KeyboardInterrupt:
    led.value = 0  # Schalten Sie die LED aus
```

Code-Erklärung

1. `gpiozero` für die PWM-LED-Steuerung, `ADC0834` für die Analog-Digital-Umwandlung und `time` für Verzögerungen.

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

   from gpiozero import PWMLED
   import ADC0834
   import time
   ```

2. Initialisieren Sie ein PWMLED-Objekt, das mit dem GPIO-Pin 22 verbunden ist, und richten Sie den ADC0834-Konverter ein.

   ```
   # Initialisieren Sie eine PWM-LED am GPIO-Pin 22
   led = PWMLED(22)

   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Richten Sie das ADC0834-Modul ein
ADC0834.setup()
```

3. Definieren Sie eine Funktion mit dem Namen `MAP`, um Werte von einem Bereich in einen anderen zu konvertieren, nützlich für die Zuordnung von ADC-Werten zu geeigneten LED-Helligkeitsstufen.

```
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. Lesen Sie kontinuierlich den ADC-Wert in einer Schleife, ordnen Sie den ADC-Wert (0-255) einer Helligkeitsstufe (0-100) für die LED zu. Passen Sie die Helligkeit der LED entsprechend diesem umgeordneten Wert an. Implementieren Sie eine Verzögerung von 0,2 Sekunden für eine bessere Lesbarkeit und Stabilität.

```
try:
    while True:
        # Holen Sie den aktuellen Messwert vom ADC0834-Modul
        res = ADC0834.getResult()
        print('res = %d' % res)

        # Ordnen Sie den ADC-Wert einem Bereich zu, der für die Einstellung der LED-
↪Helligkeit geeignet ist
        R_val = MAP(res, 0, 255, 0, 100)

        # Stellen Sie die Helligkeit der LED ein
        led.value = float(R_val / 100)

        # Warten Sie 0,2 Sekunden, bevor Sie erneut messen
        time.sleep(0.2)

# Eleganter Ausstieg, wenn 'Ctrl+C' gedrückt wird
except KeyboardInterrupt:
    led.value = 0  # Schalten Sie die LED aus
```

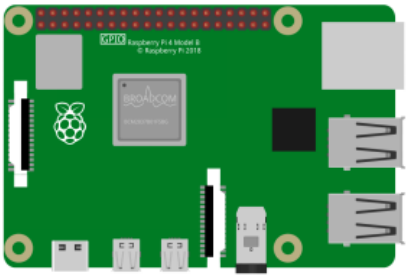
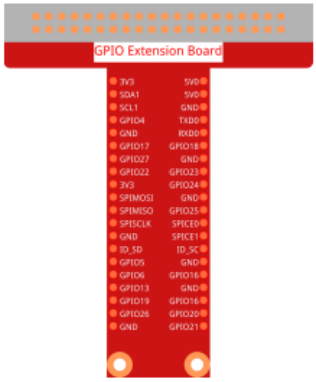
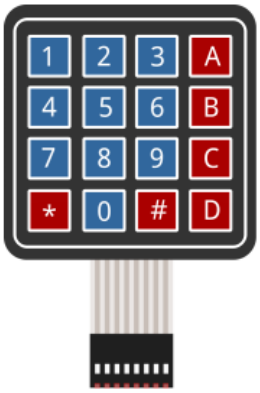
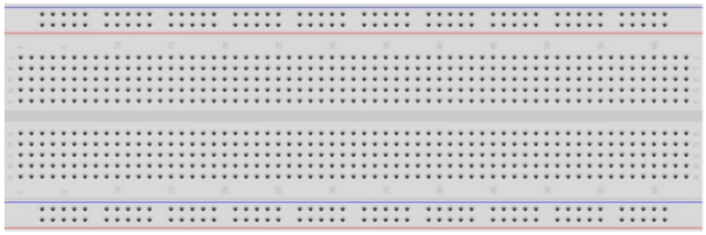
2.1.5 Tastenfeld

Einführung

Ein Tastenfeld ist ein rechteckiges Array von Tasten. In diesem Projekt werden wir es verwenden, um Zeichen einzugeben.

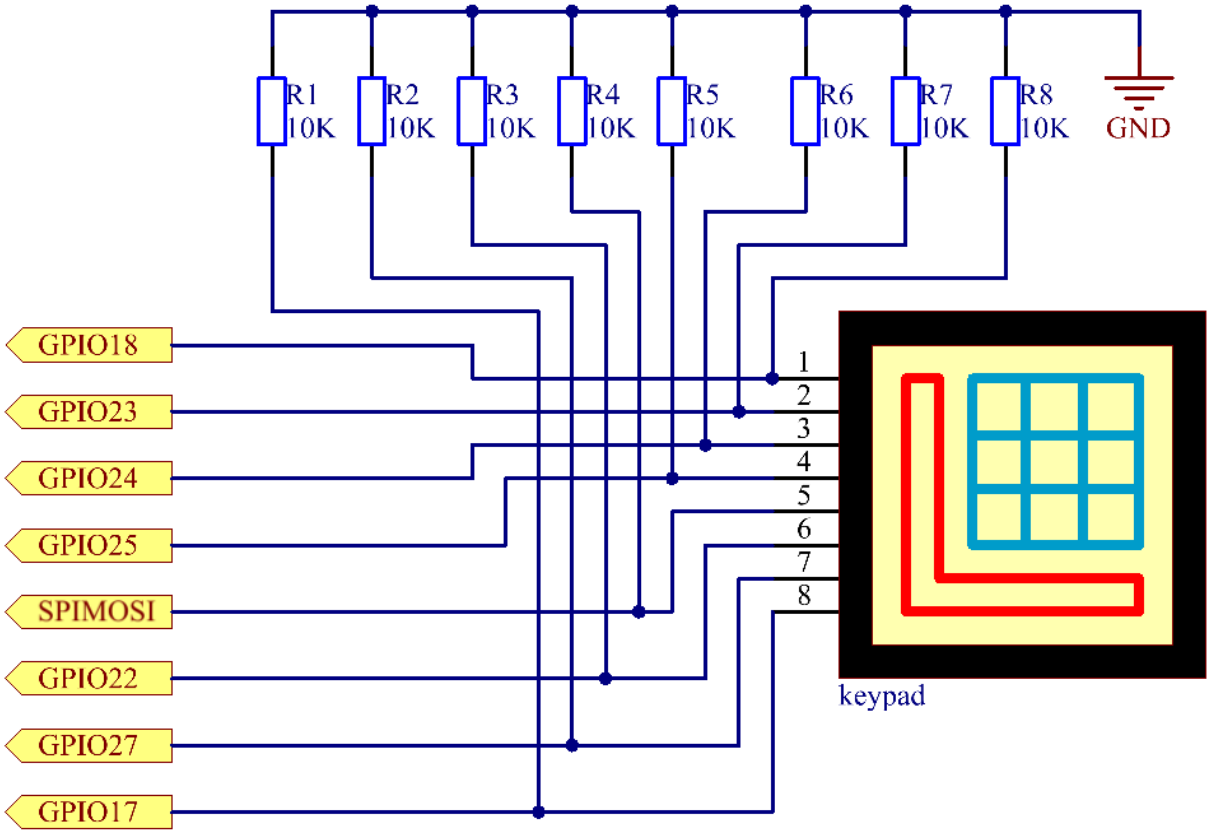
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

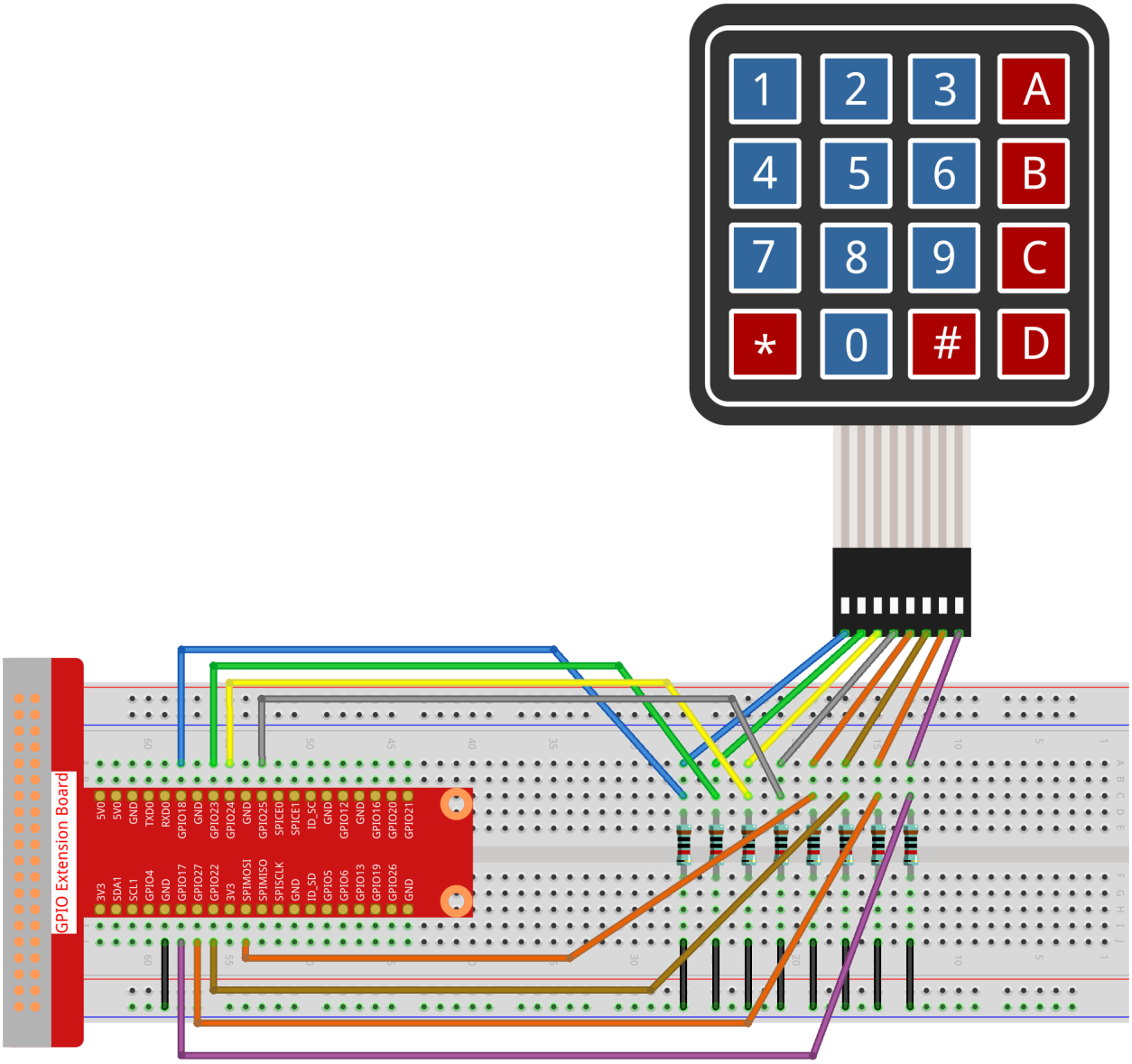
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 2.1.5_Keypad_zero.py
```

Nachdem der Code ausgeführt wurde, werden die Werte der gedrückten Tasten auf dem Tastenfeld (Tastenwert) auf dem Bildschirm angezeigt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import DigitalOutputDevice, Button
from time import sleep

class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        Initialisieren Sie das Tastenfeld mit den angegebenen Reihen- und Spaltenpins␣
↪sowie dem Tastenfeldlayout.
        :param rows_pins: Liste der GPIO-Pins für die Reihen.
        :param cols_pins: Liste der GPIO-Pins für die Spalten.
        :param keys: Liste der Tasten im Tastenfeldlayout.
        """
        # Initialisieren der Reihenpins als DigitalOutputDevice
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]
        # Initialisieren der Spaltenpins als Buttons
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]
        self.keys = keys  # Setzen des Tastenfeldlayouts

    def read(self):
        """
        Lesen Sie die derzeit gedrückten Tasten auf dem Tastenfeld.
        :return: Eine Liste der gedrückten Tasten.
        """
        pressed_keys = []
        # Scannen Sie jede Reihe und Spalte, um gedrückte Tasten zu identifizieren
        for i, row in enumerate(self.rows):
            row.on()  # Aktivieren Sie die aktuelle Reihe
            for j, col in enumerate(self.cols):
                if col.is_pressed:  # Überprüfen Sie, ob die Spaltentaste gedrückt ist
                    # Berechnen Sie den Tastenindex basierend auf Reihe und Spalte
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # Deaktivieren Sie die aktuelle Reihe
        return pressed_keys

try:
    # Konfigurieren Sie Reihen, Spalten und Tastenfeldlayout
    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"]

    # Erstellen Sie eine Instanz der Keypad-Klasse
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []

    # Lesen Sie kontinuierlich das Tastenfeld und drucken Sie neu gedrückte Tasten
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            print(pressed_keys)  # Drucken Sie die Liste der gedrückten Tasten
            last_key_pressed = pressed_keys
        sleep(0.1)  # Kurze Verzögerung zur Reduzierung der CPU-Auslastung

except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (Ctrl+C) für einen sauberen Ausstieg
    pass
```

Code-Erklärung

1. Importiert die Klassen `DigitalOutputDevice` und `Button` aus der Bibliothek `gpiozero` und die Funktion `sleep` für Verzögerungen.

```
#!/usr/bin/env python3
from gpiozero import DigitalOutputDevice, Button
from time import sleep
```

2. Definiert die Klasse `Keypad`. Die Methode `__init__` initialisiert das Tastenfeld mit den gegebenen Reihen- und Spaltenpins und Tasten. Die Methode `read` scannt das Tastenfeld und gibt eine Liste der gedrückten Tasten zurück.

```
class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        Initialisieren Sie das Tastenfeld mit den angegebenen Reihen- und␣
↪Spaltenpins sowie dem Tastenfeldlayout.
        :param rows_pins: Liste der GPIO-Pins für die Reihen.
        :param cols_pins: Liste der GPIO-Pins für die Spalten.
        :param keys: Liste der Tasten im Tastenfeldlayout.
        """
        # Initialisieren der Reihenpins als DigitalOutputDevice
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]
        # Initialisieren der Spaltenpins als Buttons
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]
        self.keys = keys  # Setzen des Tastenfeldlayouts

    def read(self):
        """
        Lesen Sie die derzeit gedrückten Tasten auf dem Tastenfeld.
        :return: Eine Liste der gedrückten Tasten.
        """
        pressed_keys = []
        # Scannen Sie jede Reihe und Spalte, um gedrückte Tasten zu identifizieren
        for i, row in enumerate(self.rows):
            row.on()  # Aktivieren Sie die aktuelle Reihe
            for j, col in enumerate(self.cols):
                if col.is_pressed:  # Überprüfen Sie, ob die Spaltentaste gedrückt␣
↪ist
                    # Berechnen Sie den Tastenindex basierend auf Reihe und Spalte
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])
            row.off()  # Deaktivieren Sie die aktuelle Reihe
        return pressed_keys
```

3. Richtet die GPIO-Pins für Reihen und Spalten ein und definiert das Tastenfeldlayout.

```
try:
    # Konfigurieren Sie Reihen, Spalten und Tastenfeldlayout
    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. Erstellt eine Instanz der `Keypad`-Klasse mit der spezifizierten Konfiguration.

```
try:
    ...

    # Erstellen Sie eine Instanz der Keypad-Klasse
    keypad = Keypad(rows_pins, cols_pins, keys)
    last_key_pressed = []
```

5. Liest kontinuierlich das Tastenfeld auf gedrückte Tasten, druckt Änderungen im Tastenzustand und führt eine kurze Verzögerung ein, um die CPU-Auslastung zu verringern. Fängt eine KeyboardInterrupt (wie Ctrl+C) ab, um einen eleganten Ausstieg aus dem Skript zu ermöglichen.

```
try:
    ...

    # Lesen Sie kontinuierlich das Tastenfeld und drucken Sie neu gedrückte Tasten
    while True:
        pressed_keys = keypad.read()
        if pressed_keys and pressed_keys != last_key_pressed:
            print(pressed_keys)  # Drucken Sie die Liste der gedrückten Tasten
            last_key_pressed = pressed_keys
        sleep(0.1)  # Kurze Verzögerung zur Reduzierung der CPU-Auslastung

except KeyboardInterrupt:
    # Behandeln Sie eine Tastaturunterbrechung (Ctrl+C) für einen sauberen Ausstieg
    pass
```

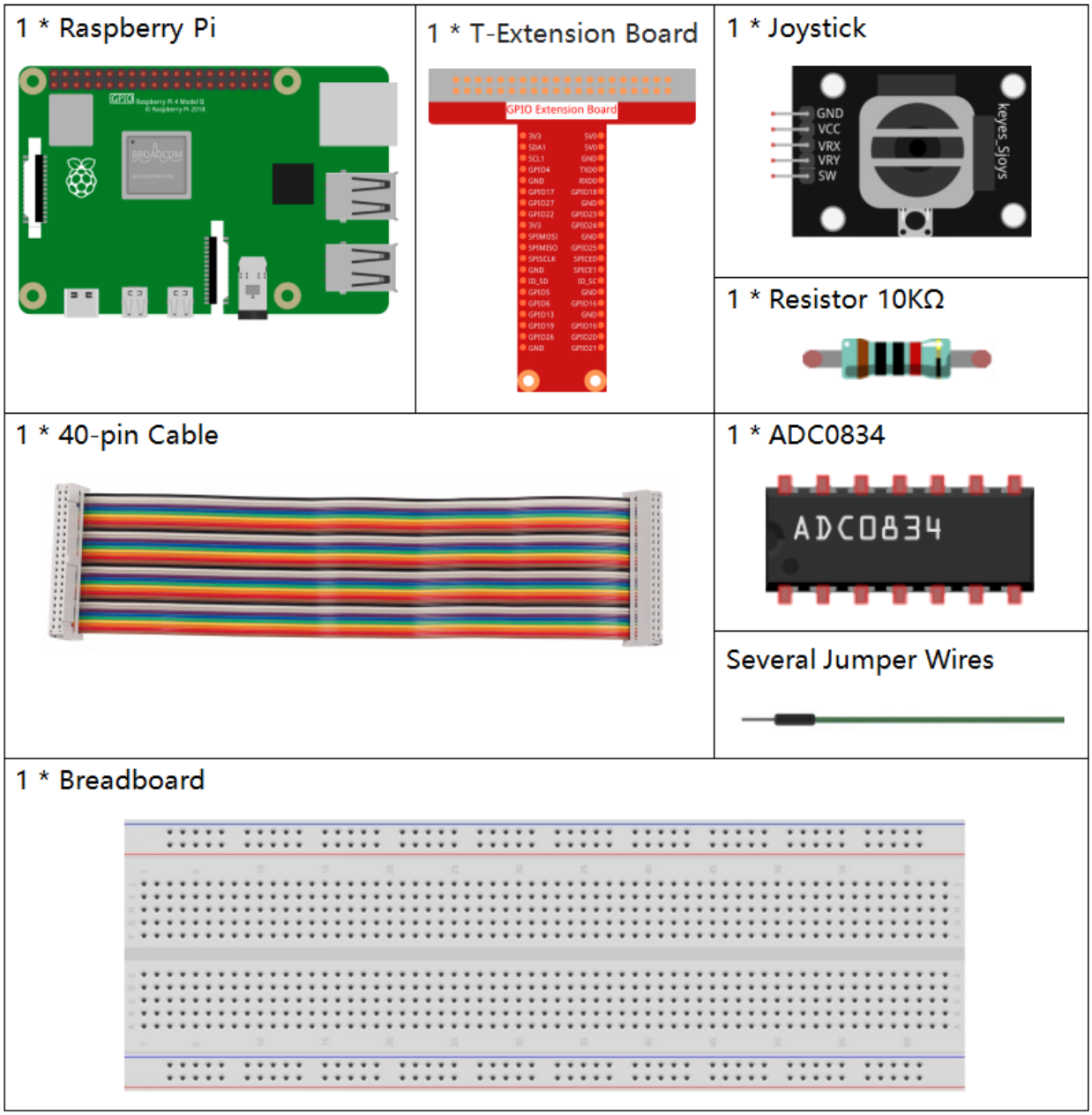
2.1.6 Joystick

Einführung

In diesem Projekt werden wir lernen, wie ein Joystick funktioniert. Wir bedienen den Joystick und zeigen die Ergebnisse auf dem Bildschirm an.

Benötigte Komponenten

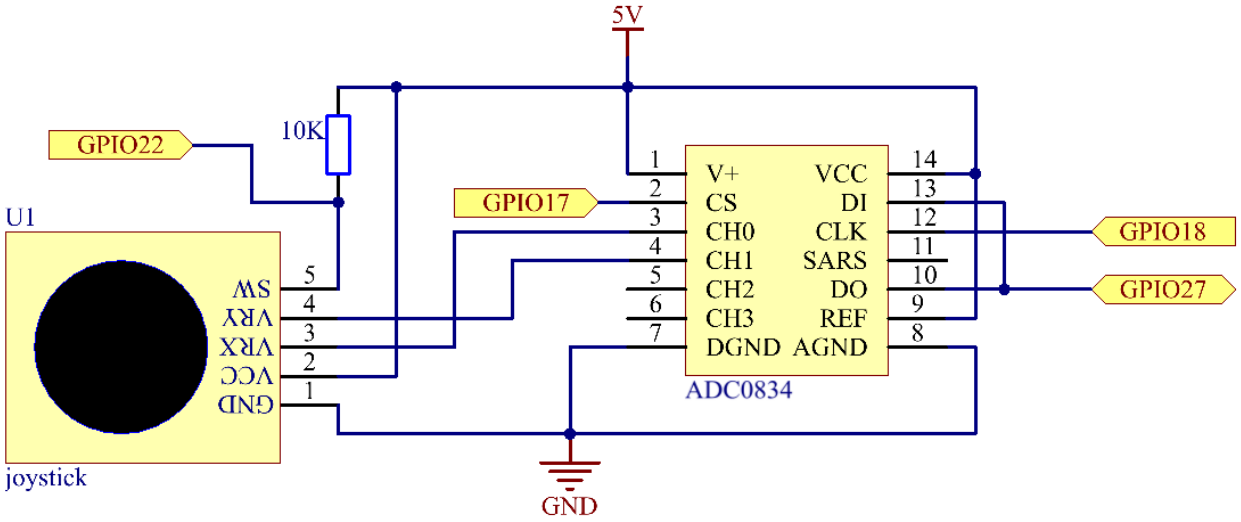
Für dieses Projekt benötigen wir die folgenden Komponenten.



Schaltplan

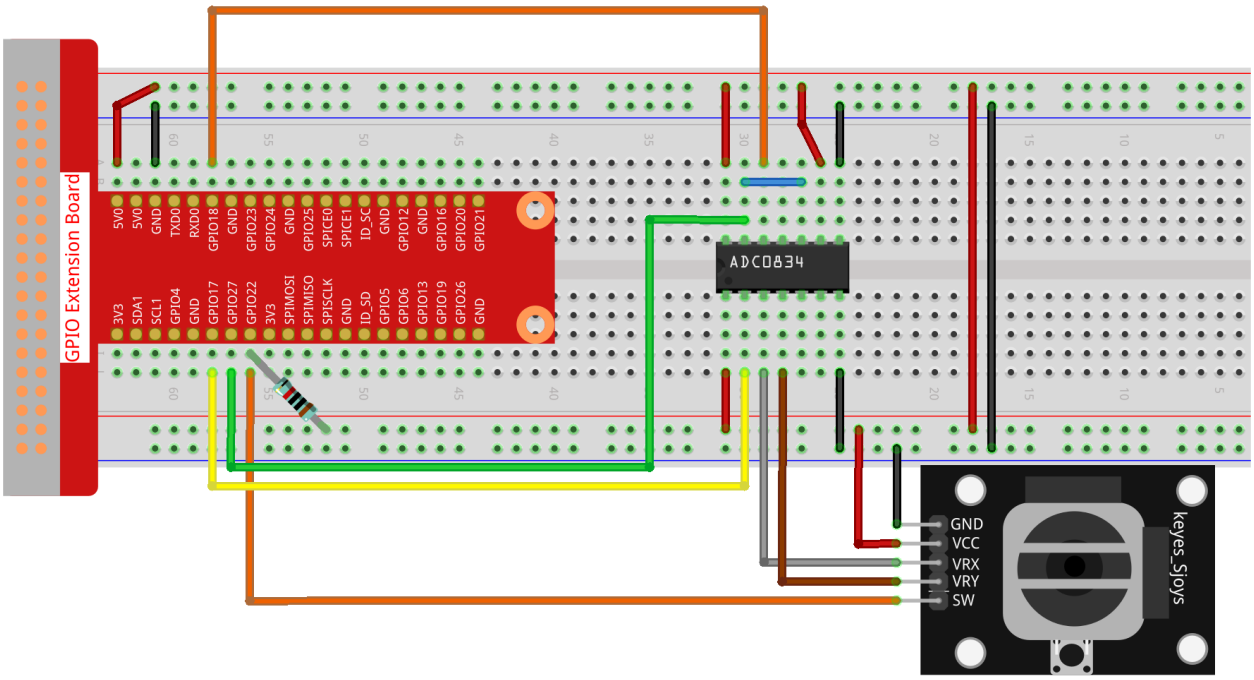
Wenn die Daten des Joysticks gelesen werden, gibt es einige Unterschiede zwischen den Achsen: Daten der X- und Y-Achse sind analog, wofür der ADC0834 benötigt wird, um den analogen Wert in einen digitalen Wert umzuwandeln. Daten der Z-Achse sind digital, also können Sie diese direkt mit dem GPIO lesen, oder Sie können auch den ADC verwenden, um zu lesen.

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 2.1.6_Joystick_zero.py
```

Nachdem der Code ausgeführt wurde, drehen Sie den Joystick, dann werden die entsprechenden Werte von x, y, Btn auf dem Bildschirm angezeigt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import Button
import ADC0834
import time

# Initialisieren Sie den Button, der mit GPIO-Pin 22 verbunden ist
BtnPin = Button(22)

# Richten Sie den ADC0834-ADC ein
ADC0834.setup()

try:
    # Hauptschleife zum Lesen und Anzeigen der ADC-Werte und des Knopfzustands
    while True:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        # Lesen Sie die X- und Y-Werte von den ADC-Kanälen 0 und 1
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)

        # Lesen Sie den Zustand des Knopfes (gedrückt oder nicht)
        Btn_val = BtnPin.value

        # Drucken Sie die X-, Y- und Knopfwerte aus
        print('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))

        # Verzögerung von 0,2 Sekunden vor dem nächsten Lesen
        time.sleep(0.2)

# Eleganter Ausstieg bei Skriptbeendigung (z. B. durch KeyboardInterrupt)
except KeyboardInterrupt:
    pass
```

Code-Erklärung

1. Dieser Abschnitt importiert die Klasse Button aus der Bibliothek `gpiozero` zur Verwaltung eines Buttons, der mit einem GPIO-Pin verbunden ist. Es importiert auch die Bibliothek `ADC0834` für die Schnittstelle mit dem ADC0834-ADC (Analog-Digital-Umwandler) Modul.

   ```
   #!/usr/bin/env python3
   from gpiozero import Button
   import ADC0834
   import time
   ```

2. Initialisiert einen Button, der mit GPIO-Pin 22 verbunden ist, und richtet das ADC0834-Modul zur Verwendung ein.

   ```
   # Initialisieren Sie den Button, der mit GPIO-Pin 22 verbunden ist
   BtnPin = Button(22)

   # Richten Sie den ADC0834-ADC ein
   ADC0834.setup()
   ```

3. Die VRX- und VRY-Verbindungen des Joysticks sind mit CH0 bzw. CH1 des ADC0834 verbunden. Diese Einrichtung ermöglicht das Lesen der Werte von CH0 und CH1, die dann in den Variablen `x_val` und `y_val` gespeichert werden. Zusätzlich wird der SW-Wert des Joysticks gelesen und der Variablen `Btn_val` zugewiesen. Die abgerufenen Werte von `x_val`, `y_val` und `Btn_val` werden anschließend mit der Funktion `print()` angezeigt.

   ```
   try:
       # Hauptschleife zum Lesen und Anzeigen der ADC-Werte und des Knopfzustands
       while True:
           # Lesen Sie die X- und Y-Werte von den ADC-Kanälen 0 und 1
           x_val = ADC0834.getResult(0)
           y_val = ADC0834.getResult(1)

           # Lesen Sie den Zustand des Knopfes (gedrückt oder nicht)
           Btn_val = BtnPin.value

           # Drucken Sie die X-, Y- und Knopfwerte aus
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        print('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))

        # Verzögerung von 0,2 Sekunden vor dem nächsten Lesen
        time.sleep(0.2)

# Eleganter Ausstieg bei Skriptbeendigung (z. B. durch KeyboardInterrupt)
except KeyboardInterrupt:
    pass
```

2.2 Sensoren

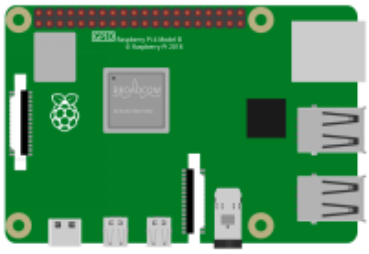
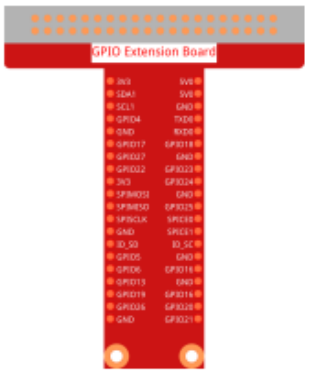
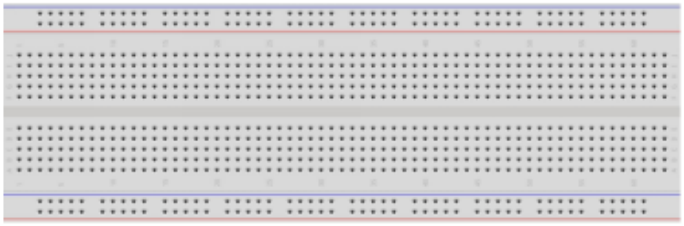
2.2.1 Fotowiderstand

Einführung

Ein Fotowiderstand ist eine häufig verwendete Komponente zur Messung der Umgebungslichtintensität im Alltag. Er hilft dem Controller, Tag und Nacht zu erkennen und Lichtsteuerungsfunktionen wie Nachtlicht zu realisieren. Dieses Projekt ist dem Potentiometer sehr ähnlich, und Sie könnten denken, dass es die Spannung ändert, um Licht zu erkennen.

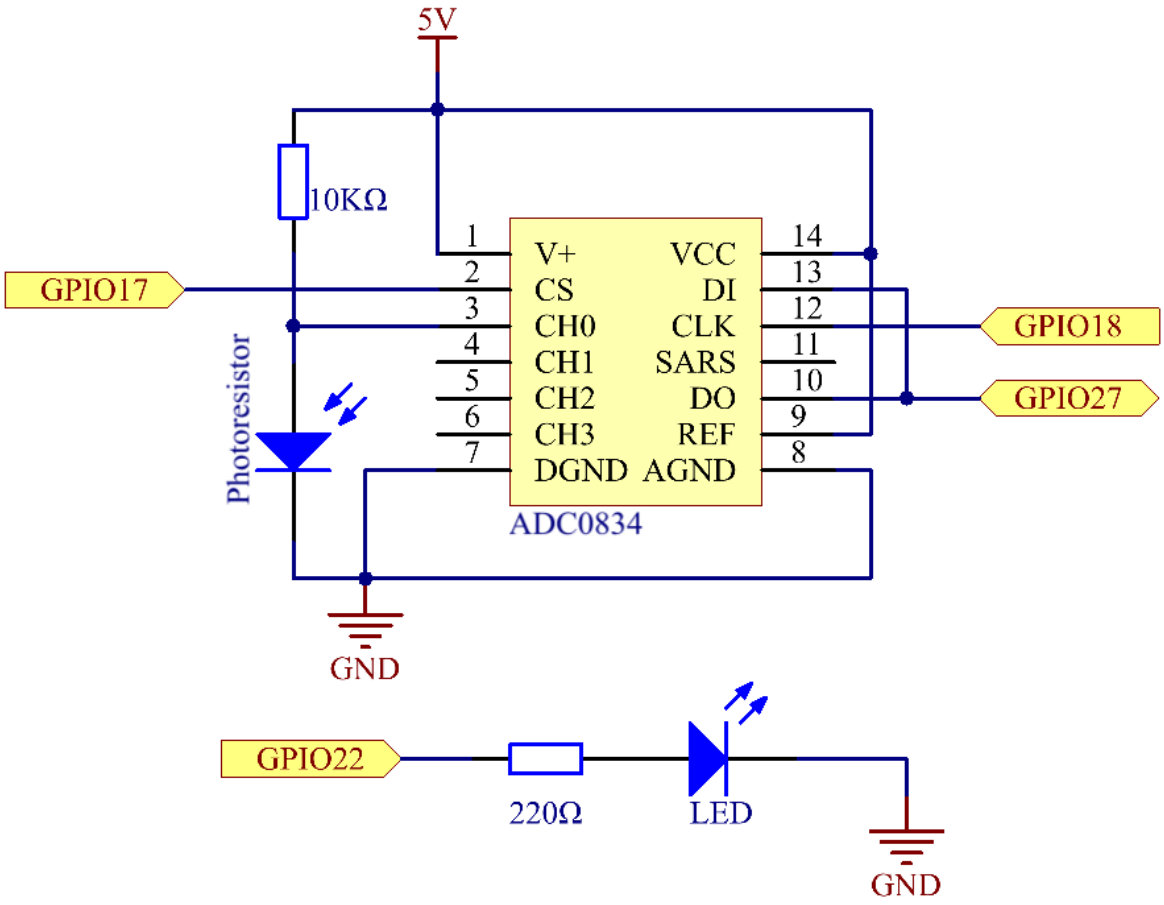
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

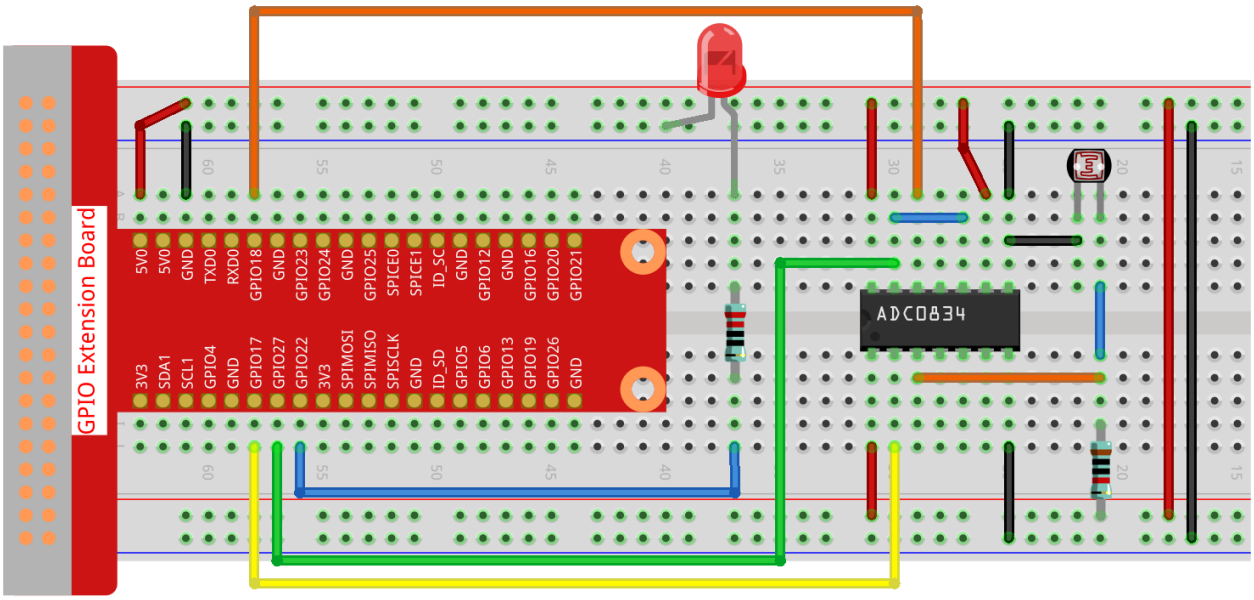
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.1_Photoresistor_zero.py
```

Wenn der Code ausgeführt wird, ändert sich die Helligkeit der LED entsprechend der Lichtintensität, die vom Fotowiderstand erfasst wird.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import PWMLED
import ADC0834
import time

# Initialisieren Sie eine PWM-LED am GPIO-Pin 22
led = PWMLED(22)

# Richten Sie das ADC0834-Modul ein
ADC0834.setup()

# Definieren Sie eine Funktion zum Abbilden von Werten von einem Bereich in einen anderen
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

# Hauptschleife zum Lesen des ADC-Werts und Steuern der LED-Helligkeit
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
def loop():
    while True:
        # Analogen Wert vom ADC lesen
        analogVal = ADC0834.getResult()
        print('value = %d' % analogVal)

        # Abbilden des ADC-Werts auf einen PWM-Wert und Einstellen der LED-Helligkeit
        led.value = float(analogVal/255)

        # Warten Sie 0,2 Sekunden
        time.sleep(0.2)

# Führen Sie die Hauptschleife aus und behandeln Sie KeyboardInterrupt für einen␣
↪eleganten Abschluss
try:
    loop()
except KeyboardInterrupt:
    # LED ausschalten, bevor Sie das Programm beenden
    led.value = 0
```

Code-Erklärung

1. Dieser Abschnitt importiert die PWMLED-Klasse aus der gpiozero-Bibliothek, die für die Steuerung von PWM-LEDs erforderlich ist. Er enthält auch das ADC0834-Modul für die Schnittstelle mit dem Analog-Digital-Wandler und das time-Modul für zeitbasierte Funktionen wie Schlaf.

   ```
   #!/usr/bin/env python3
   from gpiozero import PWMLED
   import ADC0834
   import time
   ```

2. Initialisiert eine PWM-LED, die mit GPIO-Pin 22 verbunden ist, und richtet das ADC0834-Modul ein, um es für das Projekt vorzubereiten.

   ```
   # Initialisieren Sie eine PWM-LED am GPIO-Pin 22
   led = PWMLED(22)

   # Richten Sie das ADC0834-Modul ein
   ADC0834.setup()
   ```

3. Definiert eine Funktion zur Abbildung eines Eingabewerts von einem Bereich in einen anderen. Diese Funktion ist entscheidend für die Übersetzung der ADC-Lesungen in einen geeigneten Bereich für die PWM-Steuerung.

   ```
   # Definieren Sie eine Funktion zum Abbilden von Werten von einem Bereich in einen␣
   ↪anderen
   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. Dieser Abschnitt enthält eine Schleife, die kontinuierlich den analogen Wert vom ADC0834 liest, ihn in einen entsprechenden PWM-Wert abbildet und die LED-Helligkeit anpasst. Eine kurze Verzögerung (*time.sleep(0.2)*) ist enthalten, um die Änderungen sichtbar zu machen und eine Überlastung der CPU zu vermeiden.

```
# Hauptschleife zum Lesen des ADC-Werts und Steuern der LED-Helligkeit
def loop():
    while True:
        # Analogen Wert vom ADC lesen
        analogVal = ADC0834.getResult()
        print('value = %d' % analogVal)

        # Abbilden des ADC-Werts auf einen PWM-Wert und Einstellen der LED-
↪Helligkeit
        led.value = float(analogVal/255)

        # Warten Sie 0,2 Sekunden
        time.sleep(0.2)
```

5. Führt die Loop-Funktion aus und enthält Fehlerbehandlung für einen eleganten Abschluss bei KeyboardInterrupt. Es stellt sicher, dass die LED ausgeschaltet wird, wenn das Programm gestoppt wird.

```
# Führen Sie die Hauptschleife aus und behandeln Sie KeyboardInterrupt für einen␣
↪eleganten Abschluss
try:
    loop()
except KeyboardInterrupt:
    # LED ausschalten, bevor Sie das Programm beenden
    led.value = 0
```

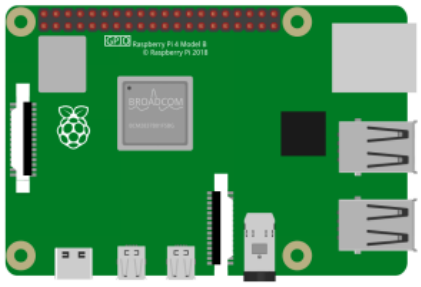
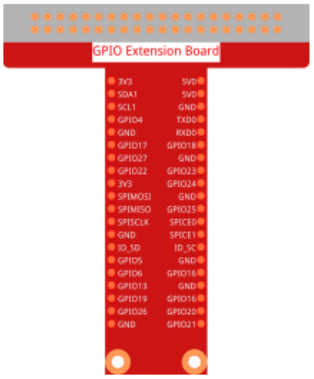
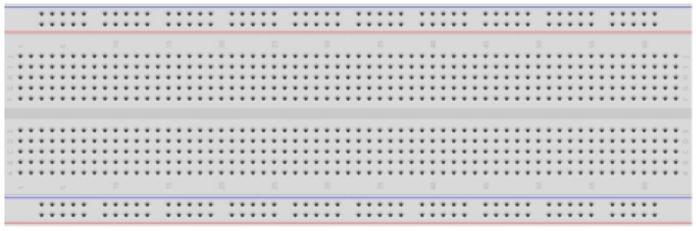
2.2.2 Thermistor

Einführung

Wie ein Fotowiderstand Licht erkennen kann, ist ein Thermistor ein temperatur- empfindliches elektronisches Bauteil, das zur Realisierung von Temperatursteuerungsfunktionen wie einem Hitzemelder verwendet werden kann.

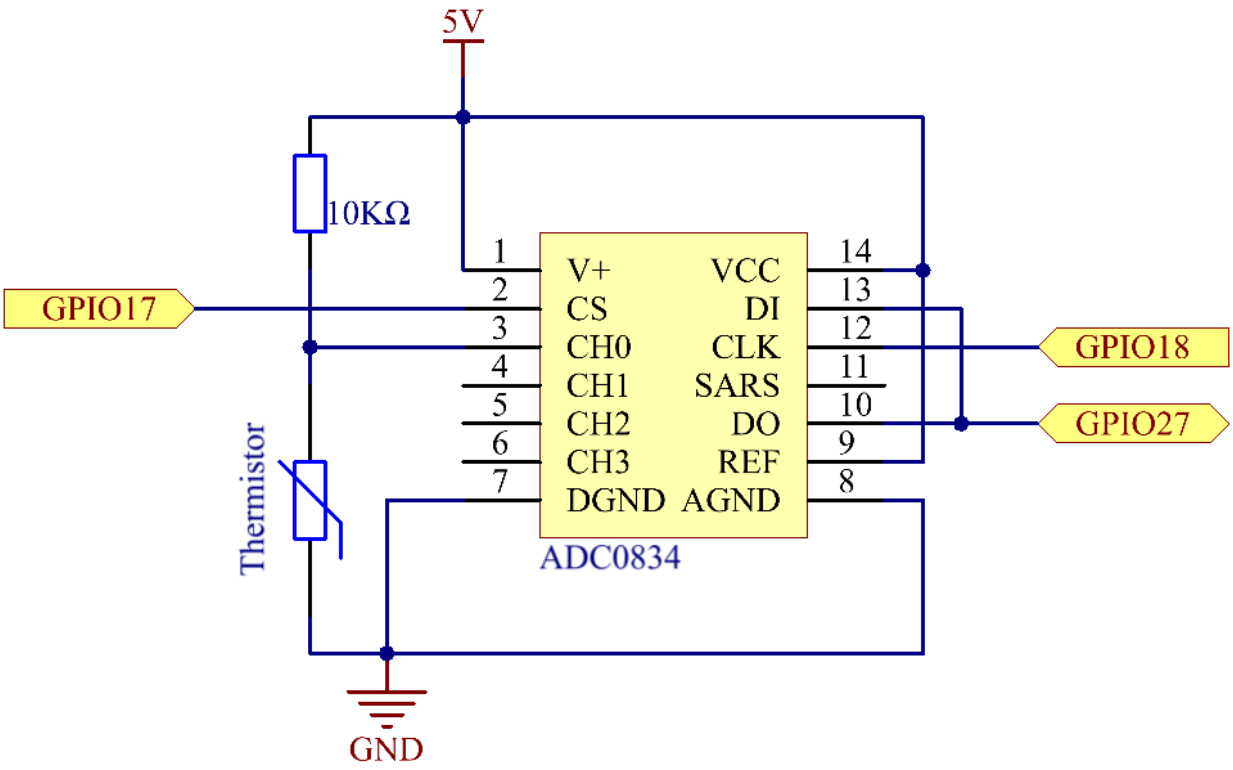
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Thermistor
		1 * ADC0834
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor 10KΩ

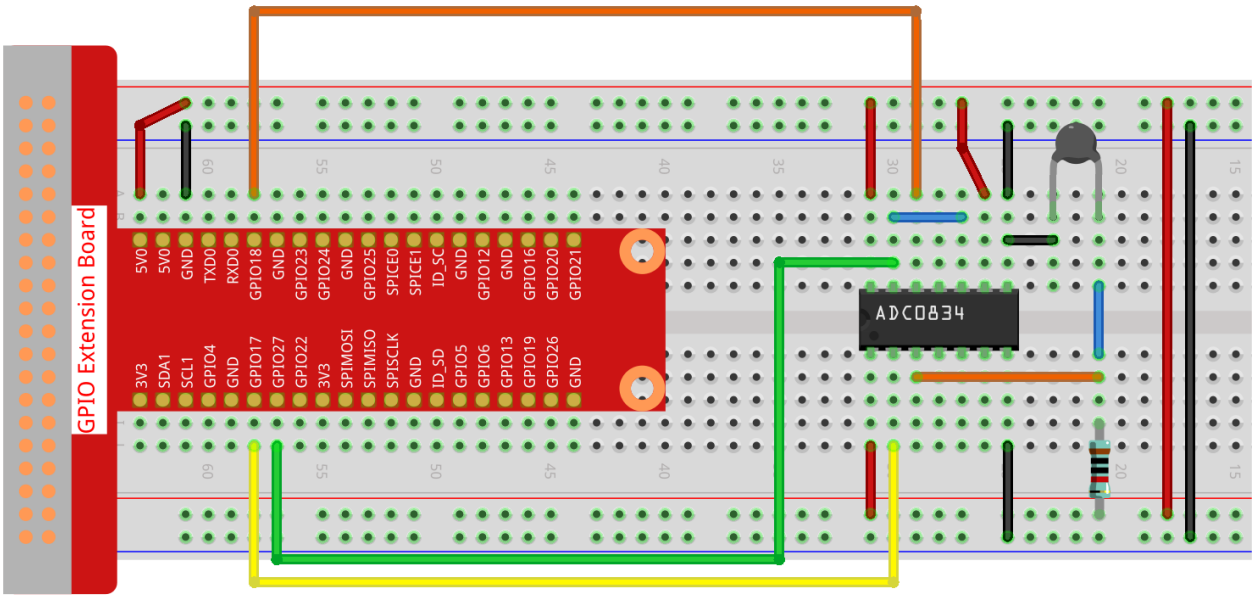
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.2_Thermistor_zero.py
```

Wenn der Code ausgeführt wird, erkennt der Thermistor die Umgebungstemperatur, die auf dem Bildschirm angezeigt wird, sobald das Programm die Berechnung abschließt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import ADC0834
import time
import math

# Richten Sie das ADC0834-Modul ein
ADC0834.setup()

# Führen Sie den Prozess in einem try-except-Block aus
try:
    while True:
        # Lesen Sie den analogen Wert vom Sensor
        analogVal = ADC0834.getResult()

        # Wandeln Sie den analogen Wert in eine Spannung um
        Vr = 5 * float(analogVal) / 255

        # Berechnen Sie den Widerstand des Thermistors
        Rt = 10000 * Vr / (5 - Vr)

        # Berechnen Sie die Temperatur in Kelvin
        temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))

        # Wandeln Sie Kelvin in Celsius um
        Cel = temp - 273.15

        # Wandeln Sie Celsius in Fahrenheit um
        Fah = Cel * 1.8 + 32

        # Drucken Sie die Temperatur in Celsius und Fahrenheit aus
        print('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))

        # Warten Sie 0,2 Sekunden vor dem nächsten Lesen
        time.sleep(0.2)

# Behandeln Sie KeyboardInterrupt für einen eleganten Abschluss
except KeyboardInterrupt:
    # Ressourcen des ADC0834 aufräumen
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ADC0834.destroy()
```

Code-Erklärung

1. Dieser Abschnitt importiert die ADC0834-Bibliothek für die Analog-Digital-Umwandlung, die time-Bibliothek für die Implementierung von Verzögerungen und die math-Bibliothek für mathematische Operationen.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import ADC0834
import time
import math
```

2. Initialisiert das ADC0834-Modul, um das Lesen von Analogwerten zu ermöglichen.

```
# Richten Sie das ADC0834-Modul ein
ADC0834.setup()
```

3. Implementiert eine Endlosschleife für kontinuierliche Datenablesung. Die Schleife liest den Analogwert von einem Thermistor, wandelt diesen Wert in eine Spannung um, berechnet den Widerstand des Thermistors und übersetzt diesen Widerstand in Temperaturmessungen in Kelvin, Celsius und Fahrenheit. Sie gibt auch die Temperaturwerte in Celsius und Fahrenheit aus und legt eine Pause von 0,2 Sekunden zwischen jeder Ablesung ein.

```
# Führen Sie den Prozess in einem try-except-Block aus
try:
    while True:
        # Lesen Sie den analogen Wert vom Sensor
        analogVal = ADC0834.getResult()

        # Wandeln Sie den analogen Wert in eine Spannung um
        Vr = 5 * float(analogVal) / 255

        # Berechnen Sie den Widerstand des Thermistors
        Rt = 10000 * Vr / (5 - Vr)

        # Berechnen Sie die Temperatur in Kelvin
        temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))

        # Wandeln Sie Kelvin in Celsius um
        Cel = temp - 273.15

        # Wandeln Sie Celsius in Fahrenheit um
        Fah = Cel * 1.8 + 32

        # Drucken Sie die Temperatur in Celsius und Fahrenheit aus
        print('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))

        # Warten Sie 0,2 Sekunden vor dem nächsten Lesen
        time.sleep(0.2)
```

4. Fängt eine KeyboardInterrupt-Ausnahme ab, um das Programm elegant zu beenden, und enthält Anweisungen zur Bereinigung der ADC0834-Ressourcen bei Beendigung.

```
# Behandeln Sie KeyboardInterrupt für einen eleganten Abschluss
except KeyboardInterrupt:
    # Ressourcen des ADC0834 aufräumen
    ADC0834.destroy()
```

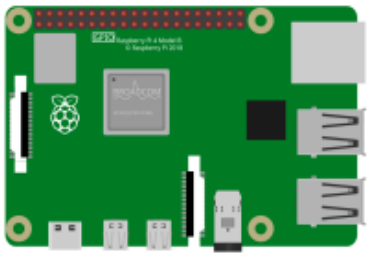
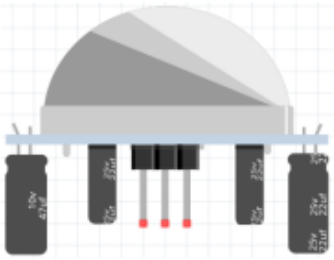
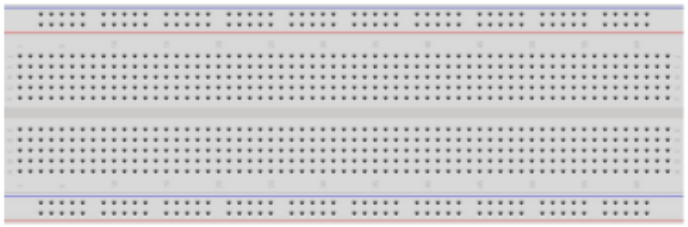
2.2.3 PIR

Einführung

In diesem Projekt werden wir ein Gerät mit Hilfe von Infrarotsensoren für den menschlichen Körper herstellen. Wenn sich jemand der LED nähert, schaltet sich die LED automatisch ein. Wenn nicht, erlischt das Licht. Dieser Infrarot-Bewegungssensor ist eine Art Sensor, der die von Menschen und Tieren abgegebene Infrarotstrahlung erkennen kann.

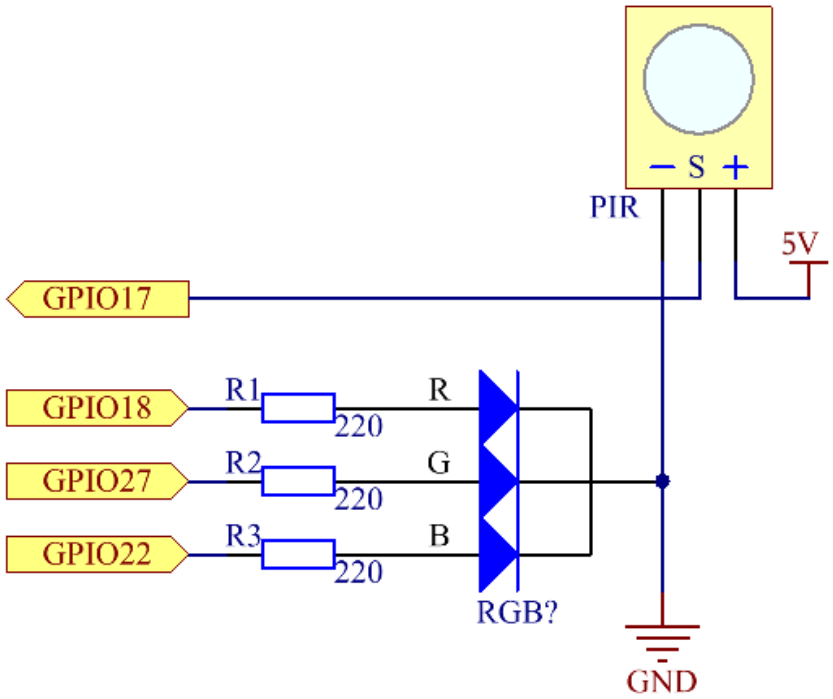
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

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

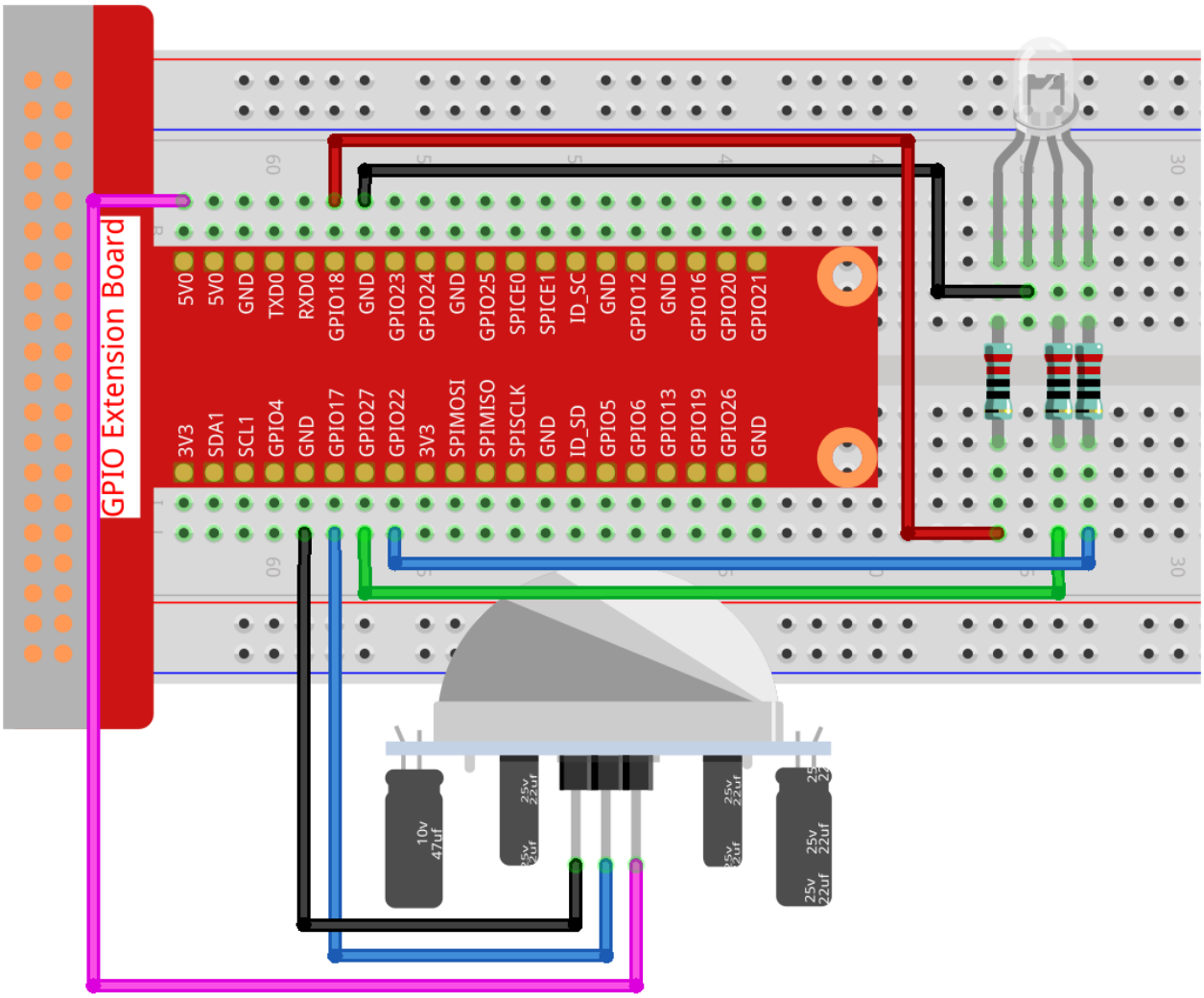
Schaltplan

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.4_PIR_zero.py
```

Nachdem der Code ausgeführt wurde, erkennt der PIR die Umgebung und lässt die RGB-LED gelb leuchten, wenn er jemanden vorbeigehen spürt.

Es gibt zwei Potentiometer auf dem PIR-Modul: eines zur Einstellung der Empfindlichkeit und das andere zur Einstellung der Erfassungsentfernung. Um das PIR-Modul optimal zu nutzen, sollten Sie beide gegen den Uhrzeigersinn bis zum Anschlag drehen.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import RGBLED, MotionSensor
from time import sleep

# Initialisieren Sie die RGB-LED und den PIR-Bewegungssensor mit der GPIO Zero-Bibliothek
led = RGBLED(red=18, green=27, blue=22)  # RGB-LED an GPIO-Pins 18 (Rot), 27 (Grün), 22␣
↪(Blau) angeschlossen
pir = MotionSensor(17)  # PIR-Sensor an GPIO-Pin 17 angeschlossen

try:
    # Kontinuierlich auf Bewegungen achten und die LED-Farbe aktualisieren
    while True:
        if pir.motion_detected:  # Überprüfen auf vom PIR-Sensor erkannte Bewegung
            led.color = (1, 1, 0)  # LED-Farbe auf Gelb einstellen (Rot + Grün)
        else:
            led.color = (0, 0, 1)  # LED-Farbe auf Blau einstellen (nur Blau)
        sleep(0.1)  # Kurze Verzögerung zur Reduzierung der CPU-Last

except KeyboardInterrupt:
    # Behandeln Sie KeyboardInterrupt (Ctrl+C) zum eleganten Beenden der Schleife
    pass
```

Code-Erklärung

1. Importiert die Klasse `RGBLED` für die Steuerung einer RGB-LED und die Klasse `MotionSensor` für die Bewegungserkennung aus der GPIO Zero-Bibliothek. Importiert auch die `sleep`-Funktion für Verzögerungen.

```
#!/usr/bin/env python3
from gpiozero import RGBLED, MotionSensor
from time import sleep
```

2. Initialisiert eine RGB-LED mit roten, grünen und blauen Komponenten, die an die GPIO-Pins 18, 27 und 22 angeschlossen sind. Außerdem wird ein PIR-Sensor am GPIO-Pin 17 initialisiert.

```
# Initialisieren Sie die RGB-LED und den PIR-Bewegungssensor mit der GPIO Zero-
↪Bibliothek
led = RGBLED(red=18, green=27, blue=22)  # RGB-LED an GPIO-Pins 18 (Rot), 27 (Grün),
↪ 22 (Blau) angeschlossen
pir = MotionSensor(17)  # PIR-Sensor an GPIO-Pin 17 angeschlossen
```

3. Das Programm betritt eine Endlosschleife, in der es kontinuierlich nach Bewegungen sucht und die PIR-Sensordaten verwendet. Wenn eine Bewegung erkannt wird, wird die LED auf Gelb (Mischung aus Rot und Grün) eingestellt. Wenn keine Bewegung erkannt wird, wird die LED auf Blau umgestellt. Eine kurze Schlafphase von 0,1 Sekunden reduziert die CPU-Last.

```
try:
    # Kontinuierlich auf Bewegungen achten und die LED-Farbe aktualisieren
    while True:
        if pir.motion_detected:  # Überprüfen auf vom PIR-Sensor erkannte Bewegung
            led.color = (1, 1, 0)  # LED-Farbe auf Gelb einstellen (Rot + Grün)
        else:
            led.color = (0, 0, 1)  # LED-Farbe auf Blau einstellen (nur Blau)
        sleep(0.1)  # Kurze Verzögerung zur Reduzierung der CPU-Last

except KeyboardInterrupt:
    # Behandeln Sie KeyboardInterrupt (Ctrl+C) zum eleganten Beenden der Schleife
    pass
```

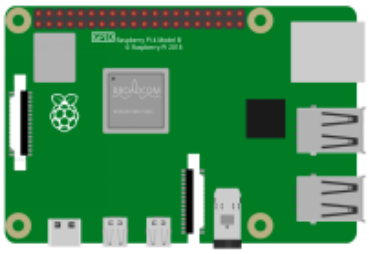
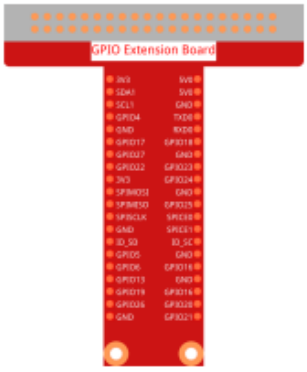
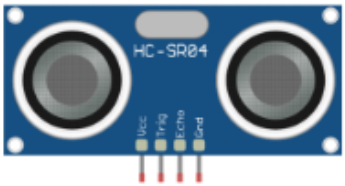
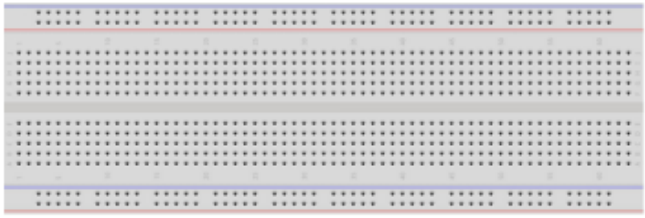
2.2.4 Ultraschall-Sensormodul

Einführung

Der Ultraschallsensor verwendet Ultraschall, um Objekte genau zu erkennen und Distanzen zu messen. Er sendet Ultraschallwellen aus und wandelt sie in elektronische Signale um.

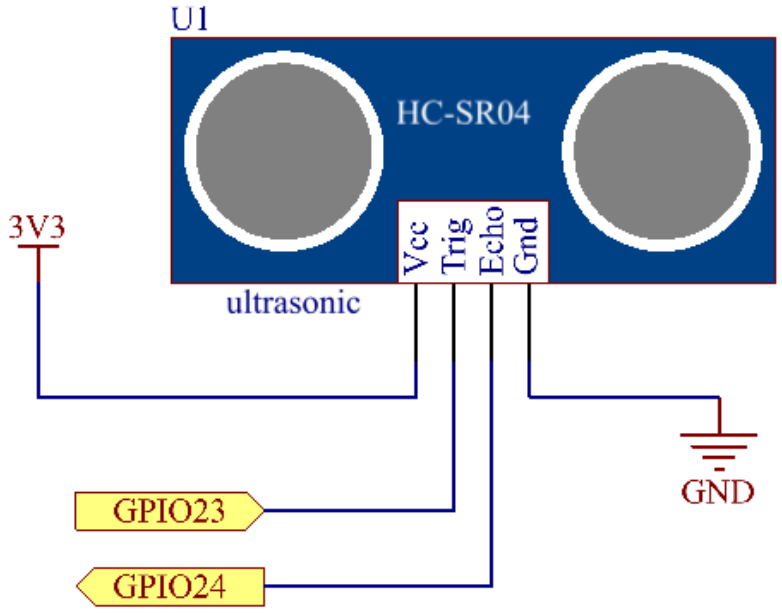
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * HC SR04 Ultrasonic Module
 
</td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

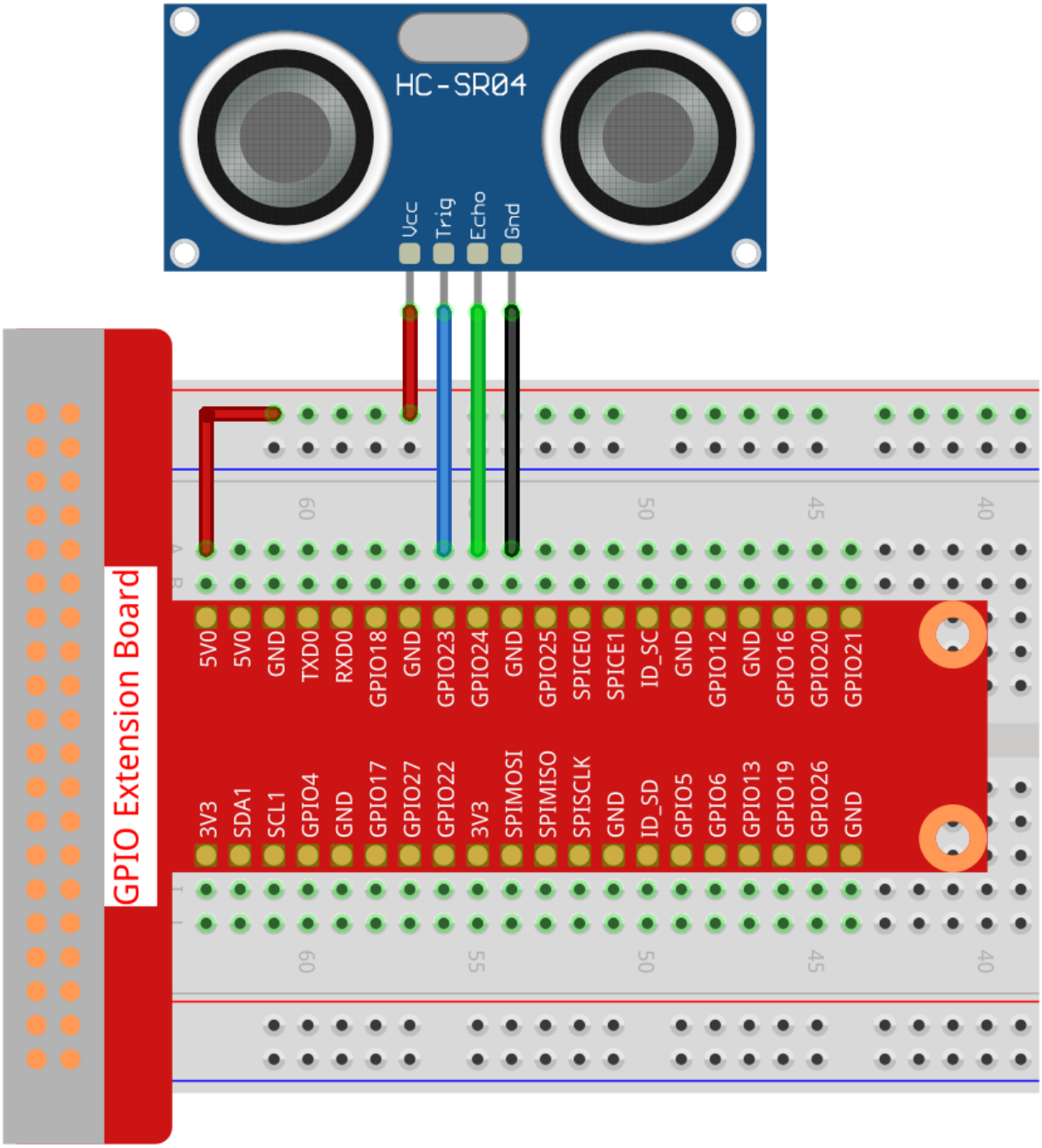
Schaltplan

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.5_Ultrasonic_zero.py
```

Wenn der Code ausgeführt wird, erfasst das Ultraschall-Sensormodul den Abstand zwischen dem Hindernis vor dem Modul und dem Modul selbst, und der Abstandswert wird auf dem Bildschirm angezeigt.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
#!/usr/bin/env python3
from gpiozero import DistanceSensor
from time import sleep

# Initialisieren Sie das Ultraschall-Distanzsensor mit der GPIO Zero-Bibliothek
# Der Trigger-Pin ist an GPIO 23 angeschlossen, der Echo-Pin an GPIO 24
sensor = DistanceSensor(echo=24, trigger=23)

try:
    # Hauptschleife zur kontinuierlichen Messung und Anzeige der Distanz
    while True:
        dis = sensor.distance * 100  # Messung der Distanz und Umrechnung von Metern in␣
↪Zentimeter
        print('Distanz: {:.2f} cm'.format(dis))  # Anzeige der Distanz mit zwei␣
↪Dezimalstellen
        sleep(0.3)  # Warten Sie 0,3 Sekunden vor der nächsten Messung

except KeyboardInterrupt:
    # Behandeln Sie KeyboardInterrupt (Ctrl+C) zum eleganten Beenden der Schleife
    pass
```

Code-Erklärung

1. Importiert die Klasse `DistanceSensor` aus der GPIO Zero-Bibliothek zur Distanzmessung und die `sleep`-Funktion aus dem `time`-Modul für Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import DistanceSensor
   from time import sleep
   ```

2. Initialisiert das Ultraschall-Distanzsensor mit dem Echo-Pin an GPIO 24 und dem Trigger-Pin an GPIO 23.

   ```
   # Initialisieren Sie das Ultraschall-Distanzsensor mit der GPIO Zero-Bibliothek
   # Der Trigger-Pin ist an GPIO 23 angeschlossen, der Echo-Pin an GPIO 24
   sensor = DistanceSensor(echo=24, trigger=23)
   ```

3. Die Hauptschleife misst kontinuierlich die Distanz, konvertiert sie von Metern in Zentimeter und gibt sie mit zwei Dezimalstellen aus. Dann wartet die Schleife 0,3 Sekunden und misst die Distanz erneut. Eine KeyboardInterrupt (wie Ctrl+C) ermöglicht ein elegantes Beenden des Skripts.

   ```
   try:
       # Hauptschleife zur kontinuierlichen Messung und Anzeige der Distanz
       while True:
           dis = sensor.distance * 100  # Messung der Distanz und Umrechnung von␣
   ↪Metern in Zentimeter
           print('Distanz: {:.2f} cm'.format(dis))  # Anzeige der Distanz mit zwei␣
   ↪Dezimalstellen
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        sleep(0.3)  # Warten Sie 0,3 Sekunden vor der nächsten Messung

except KeyboardInterrupt:
    # Behandeln Sie KeyboardInterrupt (Ctrl+C) zum eleganten Beenden der Schleife
    pass
```

2.2.5 MPU6050-Modul

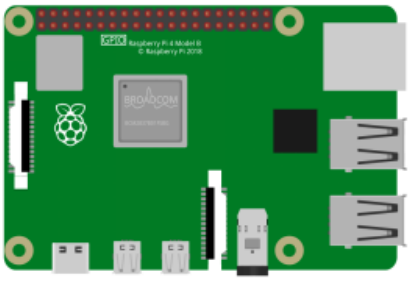
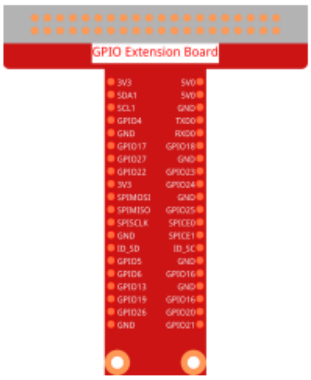
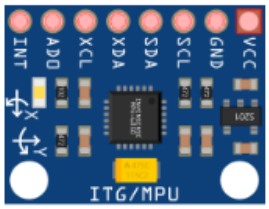
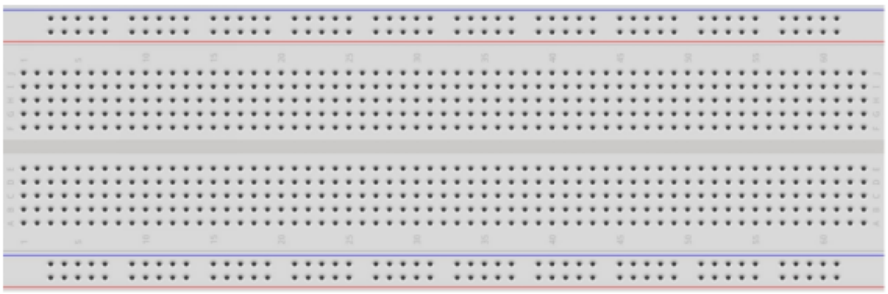
Einführung

Das MPU-6050 ist das weltweit erste und einzige 6-Achsen-Bewegungserfassungsgerät (3-Achsen-Gyroskop und 3-Achsen-Beschleunigungssensor) und wurde für Smartphones, Tablets und tragbare Sensoren entwickelt, die diese Funktionen benötigen, einschließlich geringem Energieverbrauch, geringen Kosten und hoher Leistung.

In diesem Experiment verwenden wir I2C, um die Werte des Dreiachsen- Beschleunigungssensors und des Dreiachsen-Gyroskops für das MPU6050 zu erhalten und sie auf dem Bildschirm anzuzeigen.

Benötigte Komponenten

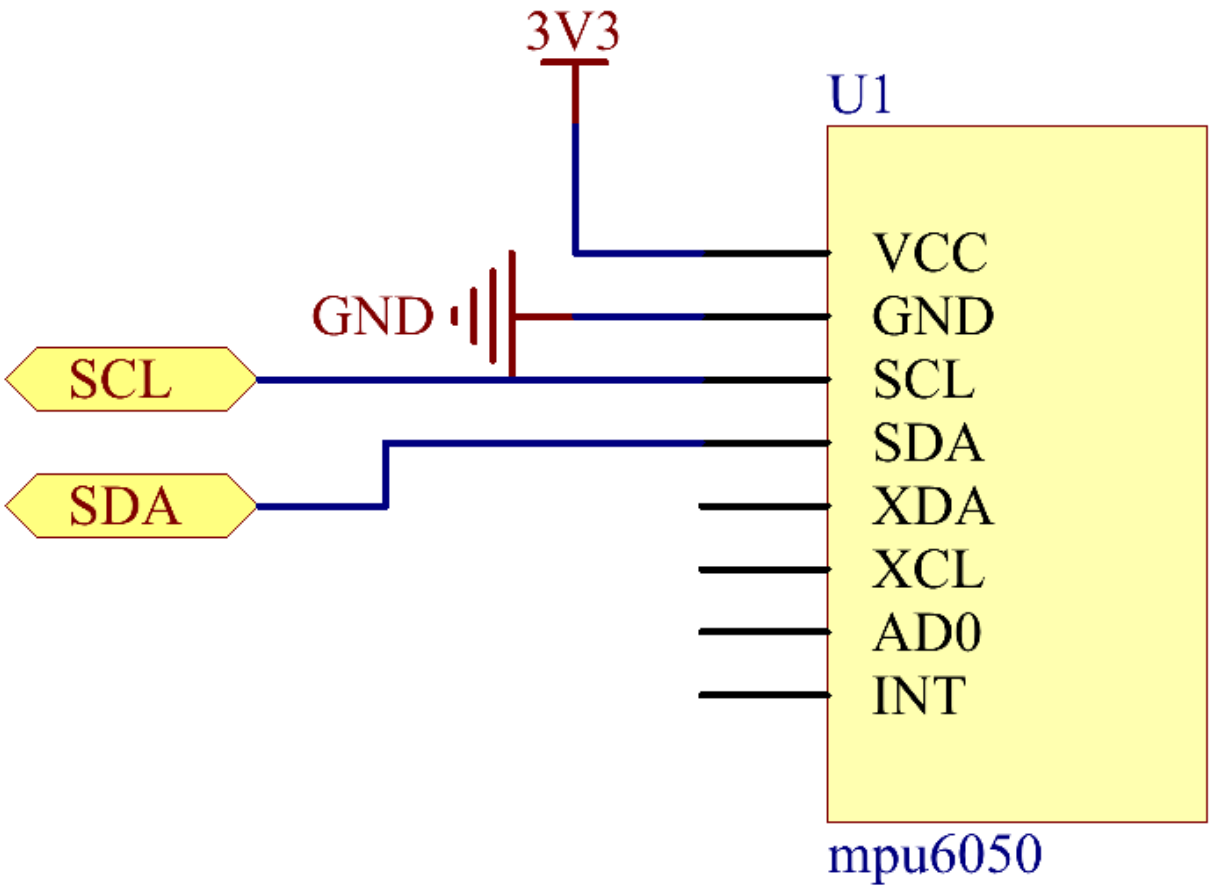
Für dieses Projekt benötigen wir die folgenden Komponenten.

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

Schaltplan

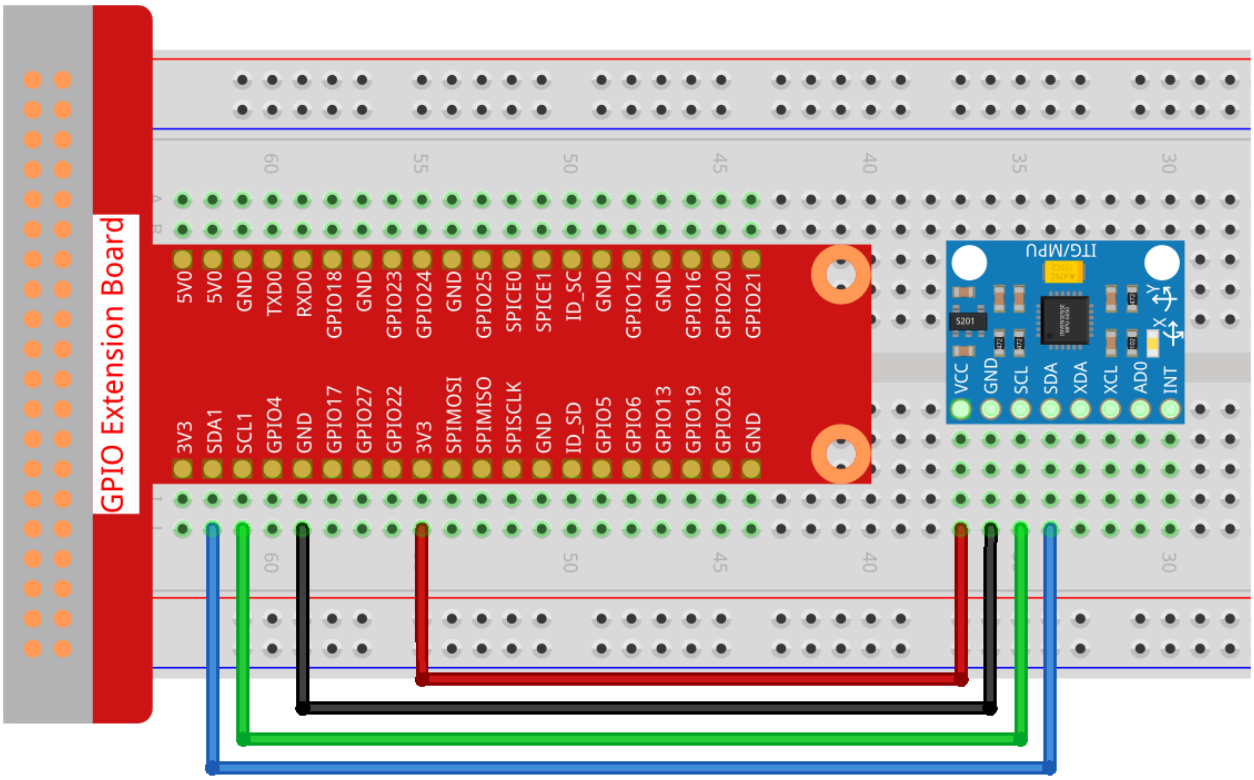
Das MPU6050 kommuniziert über die I2C-Bus-Schnittstelle mit dem Mikrocontroller. Die SDA1- und SCL1-Pins müssen mit den entsprechenden Pins verbunden werden.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: I2C einrichten (siehe Anhang *I2C-Konfiguration*. Wenn Sie I2C bereits eingerichtet haben, überspringen Sie diesen Schritt.)

Schritt 3: Wechseln Sie in den Ordner des Codes.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 4: Führen Sie die ausführbare Datei aus.

```
sudo python3 2.2.6_mpu6050_zero.py
```

Wenn der Code ausgeführt wird, erfasst das MPU6050-Modul den Ablenkwinkel der x-Achse und y-Achse sowie die Beschleunigung und Winkelgeschwindigkeit auf jeder Achse und zeigt sie auf dem Bildschirm an, nachdem sie berechnet wurden.

Bemerkung:

- Wenn Sie den Fehler `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` erhalten, beziehen Sie sich bitte auf *I2C-Konfiguration*, um das I2C zu aktivieren.
- Wenn der Fehler `ModuleNotFoundError: No module named 'smbus2'` auftritt, führen Sie bitte `sudo pip3 install smbus2` aus.
- Wenn der Fehler `OSError: [Errno 121] Remote I/O error` auftritt, bedeutet dies, dass das Modul falsch verdrahtet ist oder das Modul defekt ist.

Code

Bemerkung: Sie können den unten stehenden Code **Modifizieren/Zurücksetzen/Kopieren/Ausführen/Stoppen**.

Aber bevor Sie das tun, müssen Sie zum Quellcode-Pfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

```
import smbus
import math
import time

# Power-Management-Register
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)


bus = smbus.SMBus(1) # oder bus = smbus.SMBus(1) für Revision 2 Boards
address = 0x68       # Dies ist der Adresswert, der über den Befehl i2cdetect gelesen␣
↪wurde

# Wecken Sie jetzt das 6050 auf, da es im Schlafmodus startet
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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    print ("gyro_xout : ", gyro_xout, " skaliert: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " skaliert: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " skaliert: ", (gyro_zout / 131))

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

    accel_xout_skaliert = accel_xout / 16384.0
    accel_yout_skaliert = accel_yout / 16384.0
    accel_zout_skaliert = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " skaliert: ", accel_xout_skaliert)
    print ("accel_yout: ", accel_yout, " skaliert: ", accel_yout_skaliert)
    print ("accel_zout: ", accel_zout, " skaliert: ", accel_zout_skaliert)

    print ("x-Rotation: " , get_x_rotation(accel_xout_skaliert, accel_yout_skaliert,␣
↪accel_zout_skaliert))
    print ("y-Rotation: " , get_y_rotation(accel_xout_skaliert, accel_yout_skaliert,␣
↪accel_zout_skaliert))

    time.sleep(1)
```

Code-Erklärung

1. Sensor-Daten von MPU6050 lesen.

```
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. Ablenkwinkel der y-Achse berechnen.

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

3. Ablenkwinkel der x-Achse berechnen.

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

4. Werte der x-, y- und z-Achse des Gyroskopsensors lesen, Daten in Winkelgeschwindigkeitswerte umrechnen und dann ausgeben.

```
gyro_xout = read_word_2c(0x43)
gyro_yout = read_word_2c(0x45)
gyro_zout = read_word_2c(0x47)

print ("gyro_xout : ", gyro_xout, " skaliert: ", (gyro_xout / 131))
print ("gyro_yout : ", gyro_yout, " skaliert: ", (gyro_yout / 131))
print ("gyro_zout : ", gyro_zout, " skaliert: ", (gyro_zout / 131))
```

5. Werte der x-, y- und z-Achse des Beschleunigungssensors lesen, Elemente in beschleunigte Geschwindigkeitswerte (Gravitationseinheit) umrechnen und ausgeben.

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

accel_xout_skaliert = accel_xout / 16384.0
accel_yout_skaliert = accel_yout / 16384.0
accel_zout_skaliert = accel_zout / 16384.0

print ("accel_xout: ", accel_xout, " skaliert: ", accel_xout_skaliert)
print ("accel_yout: ", accel_yout, " skaliert: ", accel_yout_skaliert)
print ("accel_zout: ", accel_zout, " skaliert: ", accel_zout_skaliert)
```

6. Ablenkwinkel der x- und y-Achse ausgeben.

```
print ("x-Rotation: " , get_x_rotation(accel_xout_skaliert, accel_yout_skaliert,␣
↪accel_zout_skaliert))
print ("y-Rotation: " , get_y_rotation(accel_xout_skaliert, accel_yout_skaliert,␣
↪accel_zout_skaliert))
```

1.11.4 3 Erweiterung

3.1.1 Zähleinrichtung

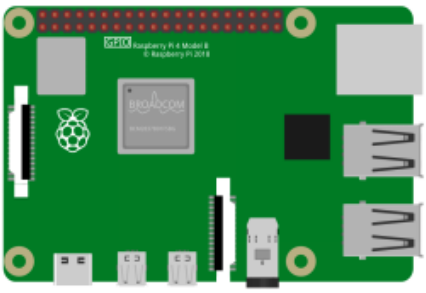
Einführung

In diesem Projekt erstellen wir ein Zählsystem mit Ziffernanzeige, bestehend aus einem PIR-Sensor und einer 4-stelligen Segmentanzeige. Sobald der PIR-Sensor erkennt, dass jemand vorbeigeht, wird die Zahl auf der 4-stelligen Segmentanzeige um 1 erhöht. Dieser Zähler kann verwendet werden, um die Anzahl der Personen zu zählen, die einen Durchgang passieren.

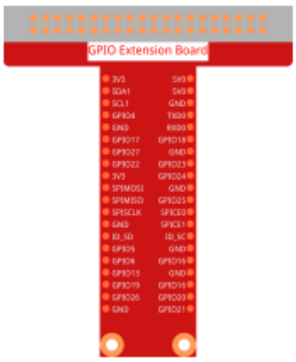
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

1 * Raspberry Pi



1 * T-Extension Board



1 * 4-Digit 7-Segment Display

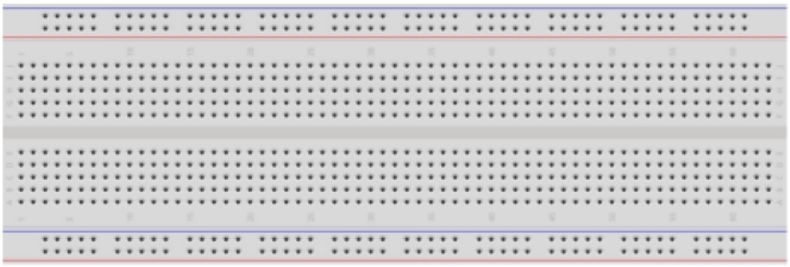
4 * Resistor(220Ω)

1 * 40-pin Cable

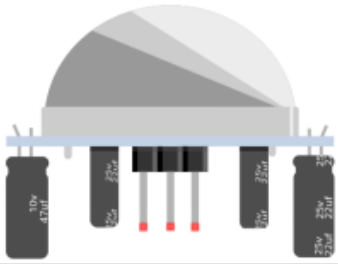
1 * 74HC595

Several Jumper Wires

1 * Breadboard

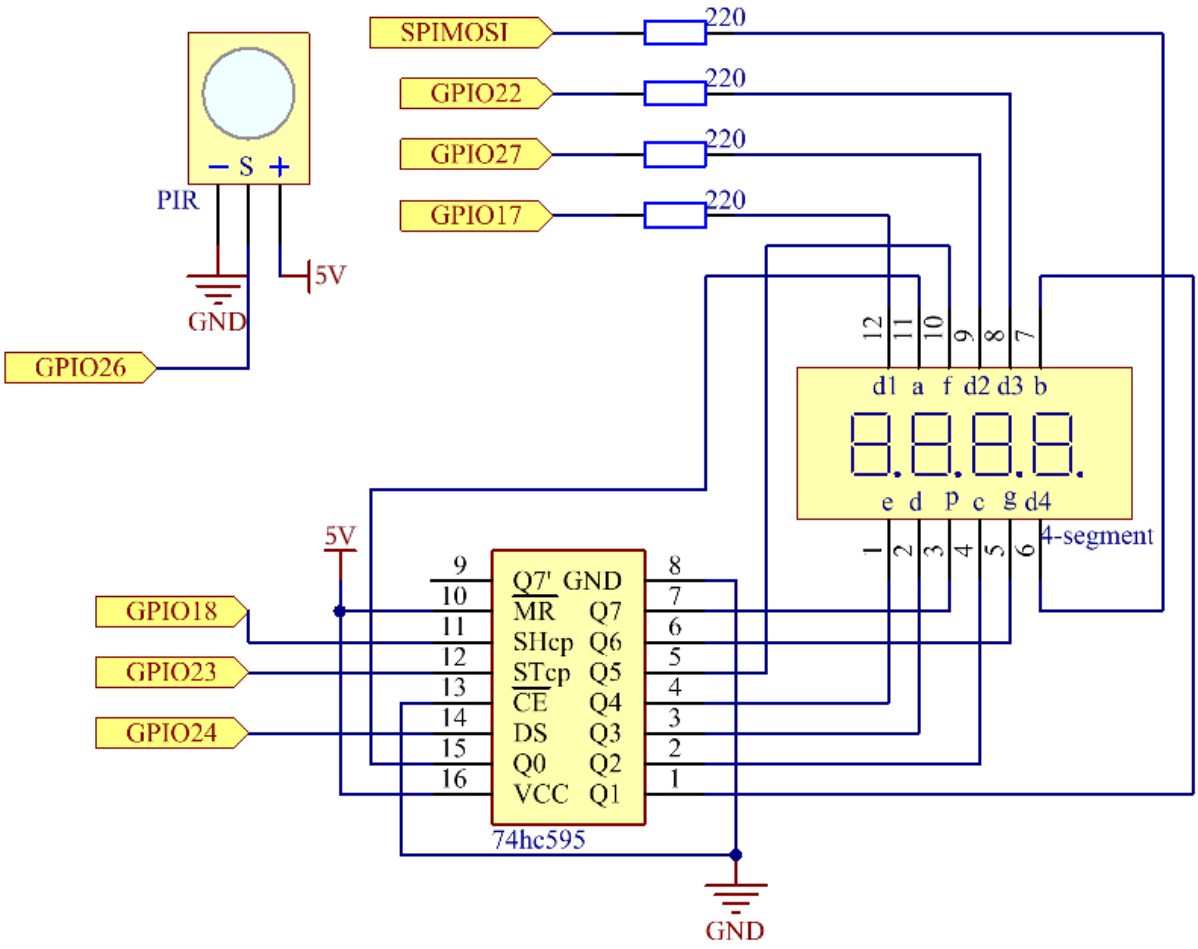


1 * PIR Sensor Module



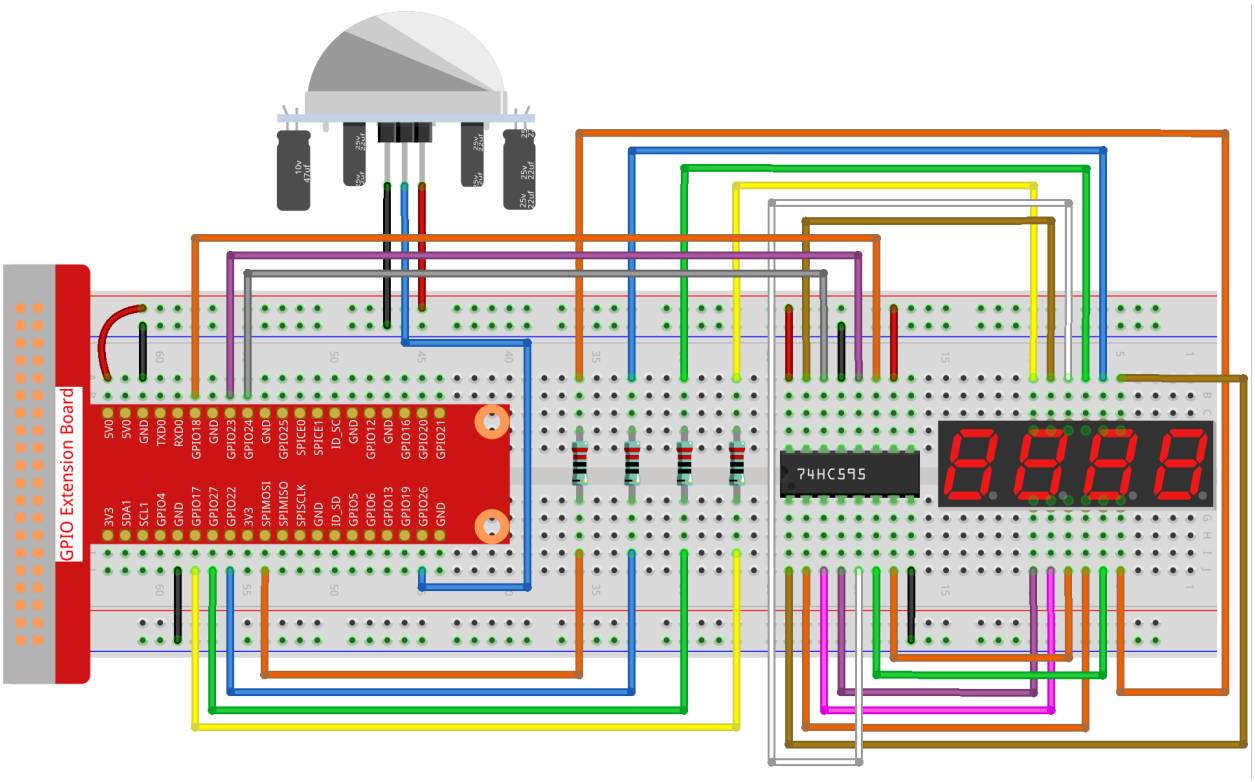
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis auf.



Schritt 2: Wechseln Sie zum Codeverzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.1_CountingDevice_zero.py
```

Nachdem der Code ausgeführt wurde, wird die Zahl auf der 4-stelligen Segmentanzeige um 1 erhöht, wenn der PIR-Sensor jemanden passieren sieht.

An dem PIR-Modul gibt es zwei Potentiometer: eines zur Einstellung der Empfindlichkeit und das andere zur Einstellung der Erkennungsdistanz. Um eine optimale Funktion des PIR-Moduls zu gewährleisten, sollten Sie beide im Uhrzeigersinn bis zum Anschlag drehen.

Code

Bemerkung: Sie können den unten stehenden Code **modifizieren, zurücksetzen, kopieren, ausführen oder stoppen**. Bevor Sie das tun, gehen Sie bitte zum Quellcodepfad wie `raphael-kit/python-pi5`. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, MotionSensor

# Initialisieren des PIR-Bewegungssensors an GPIO 26
pir = MotionSensor(26)

# Initialisieren der Schieberegisterpins
SDI = OutputDevice(24)    # Serieller Dateneingang
RCLK = OutputDevice(23)   # Register-Taktsignal
SRCLK = OutputDevice(18)  # Schieberegister-Taktsignal

# Initialisieren der 7-Segment-Anzeigepins
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Definieren der Zifferncodes für die 7-Segment-Anzeige
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# Zähler für die angezeigte Zahl
counter = 0

def clearDisplay():
    # Löscht die Anzeige, indem alle Segmente ausgeschaltet werden
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    RCLK.off()

def hc595_shift(data):
    # Verschiebt Daten in das 74HC595-Schieberegister
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    # Aktiviert eine spezifische Ziffer der 7-Segment-Anzeige
    for pin in placePin:
        pin.off()
    placePin[digit].on()

def display():
    # Aktualisiert die Anzeige mit dem aktuellen Zählerstand
    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])

def loop():
    # Hauptprogrammschleife zur Aktualisierung der Anzeige und Überprüfung auf Bewegung
    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:
    # Schaltet alle Pins aus, wenn das Skript unterbrochen wird
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    SDI.off()
    SRCLK.off()
    RCLK.off()
    pass
```

Code-Erklärung

1. Diese Zeile importiert die Klassen `OutputDevice` und `MotionSensor` aus der Bibliothek `gpiozero`. `OutputDevice` kann eine LED, ein Motor oder jedes Gerät sein, das Sie als Ausgabe steuern möchten. Der `MotionSensor` ist typischerweise ein PIR-Sensor (Passiver Infrarotsensor) zur Bewegungserkennung.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, MotionSensor
```

2. Initialisiert den PIR-Bewegungssensor, der an den GPIO-Pin 26 angeschlossen ist.

```
# Initialisieren des PIR-Bewegungssensors an GPIO 26
pir = MotionSensor(26)
```

3. Initialisiert GPIO-Pins, die mit dem seriellen Dateneingang (SDI), dem Register-Taktsignal (RCLK) und dem Schieberegister-Taktsignal (SRCLK) des Schieberegisters verbunden sind.

```
# Initialisieren der Schieberegisterpins
SDI = OutputDevice(24)    # Serieller Dateneingang
RCLK = OutputDevice(23)   # Register-Taktsignal
SRCLK = OutputDevice(18)  # Schieberegister-Taktsignal
```

4. Initialisiert die Pins für jede Ziffer der 7-Segment-Anzeige und definiert die Binärcodes für die Anzeige der Zahlen 0-9.

```
# Initialisieren der 7-Segment-Anzeigepins
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Definieren der Zifferncodes für die 7-Segment-Anzeige
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

5. Löscht die 7-Segment-Anzeige, indem alle Segmente ausgeschaltet werden, bevor die nächste Ziffer angezeigt wird.

```
def clearDisplay():
    # Löscht die Anzeige, indem alle Segmente ausgeschaltet werden
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

6. Verschiebt ein Byte Daten in das 74HC595-Schieberegister und steuert damit die Anzeigesegmente.

```
def hc595_shift(data):
    # Verschiebt Daten in das 74HC595-Schieberegister
    for i in range(8):
        SDI.value = 0x80 & (data << i)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()
```

7. Wählt aus, welche Ziffer der 7-Segment-Anzeige aktiviert werden soll. Jede Ziffer wird von einem separaten GPIO-Pin gesteuert.

```
def pickDigit(digit):
    # Aktiviert eine spezifische Ziffer der 7-Segment-Anzeige
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

8. Initiiert die Anzeige für die Einerziffer zuerst, gefolgt von der Aktivierung der Anzeige für die Zehnerziffer. Anschließend werden die Anzeigen für die Hunderter- und Tausenderziffern in dieser Reihenfolge aktiviert. Diese schnelle Abfolge von Aktivierungen erzeugt die Illusion einer kontinuierlichen vierstelligen Anzeige.

```
def display():
    # Aktualisiert die Anzeige mit dem aktuellen Zählerstand
    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. Definiert die Hauptprogrammschleife, in der die Anzeige kontinuierlich aktualisiert wird und der Zustand des PIR-Sensors überprüft wird. Wenn eine Bewegung erkannt wird, wird der Zähler erhöht.

```
def loop():
    # Hauptprogrammschleife zur Aktualisierung der Anzeige und Überprüfung auf␣
↪Bewegung
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = 1 if pir.motion_detected else 0
        if currentState == 1 und lastState == 0:
            counter += 1
        lastState = currentState
```

10. Führt die Hauptprogrammschleife aus und stellt sicher, dass das Skript mit einem Tastaturbefehl (Ctrl+C) unterbrochen werden kann, wobei alle Pins für einen sauberen Ausstieg ausgeschaltet werden.

```
try:
    loop()
except KeyboardInterrupt:
    # Schaltet alle Pins aus, wenn das Skript unterbrochen wird
    SDI.off()
    SRCLK.off()
    RCLK.off()
    pass
```

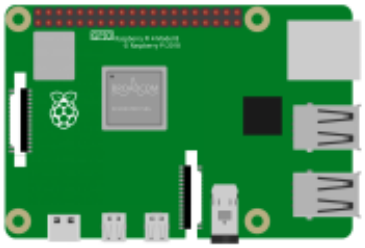
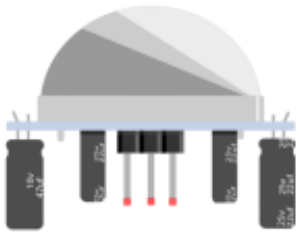
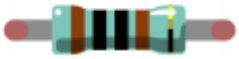
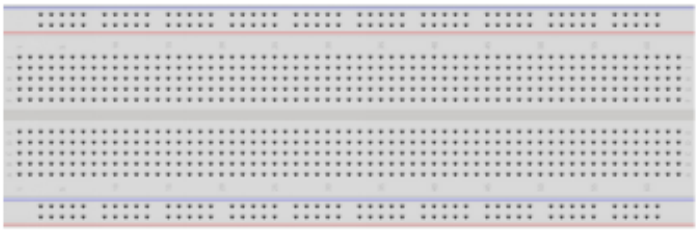
3.1.2 Willkommen

Einführung

In diesem Projekt werden wir PIR-Sensoren verwenden, um die Bewegung von Fußgängern zu erfassen. Dabei setzen wir Servomotoren, LEDs und einen Summer ein, um die Funktion der Sensortür eines Convenience-Stores zu simulieren. Erscheint ein Fußgänger im Erfassungsbereich des PIR-Sensors, wird das Anzeigelicht aktiviert, die Tür öffnet sich und der Summer spielt die Eingangsglocke.

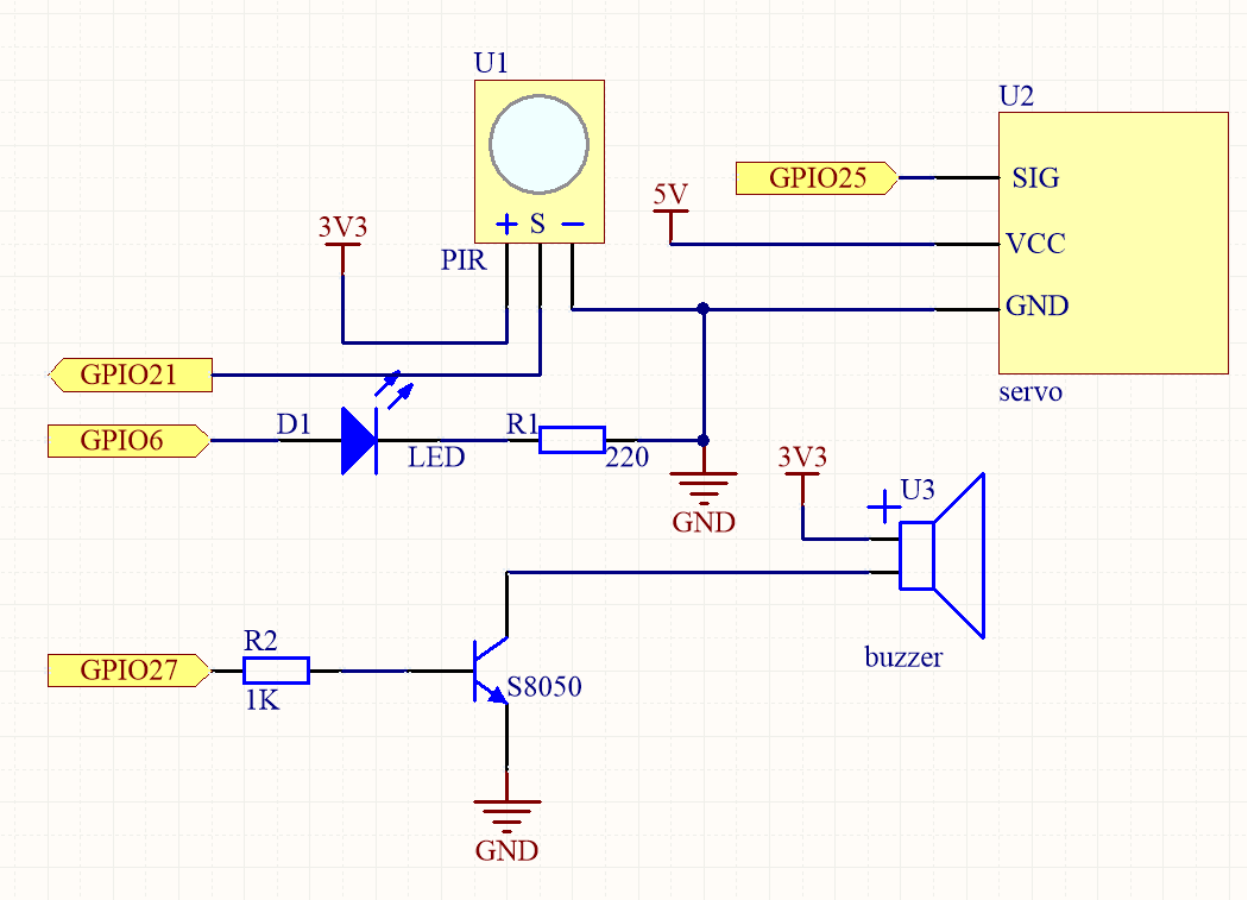
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.

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 			

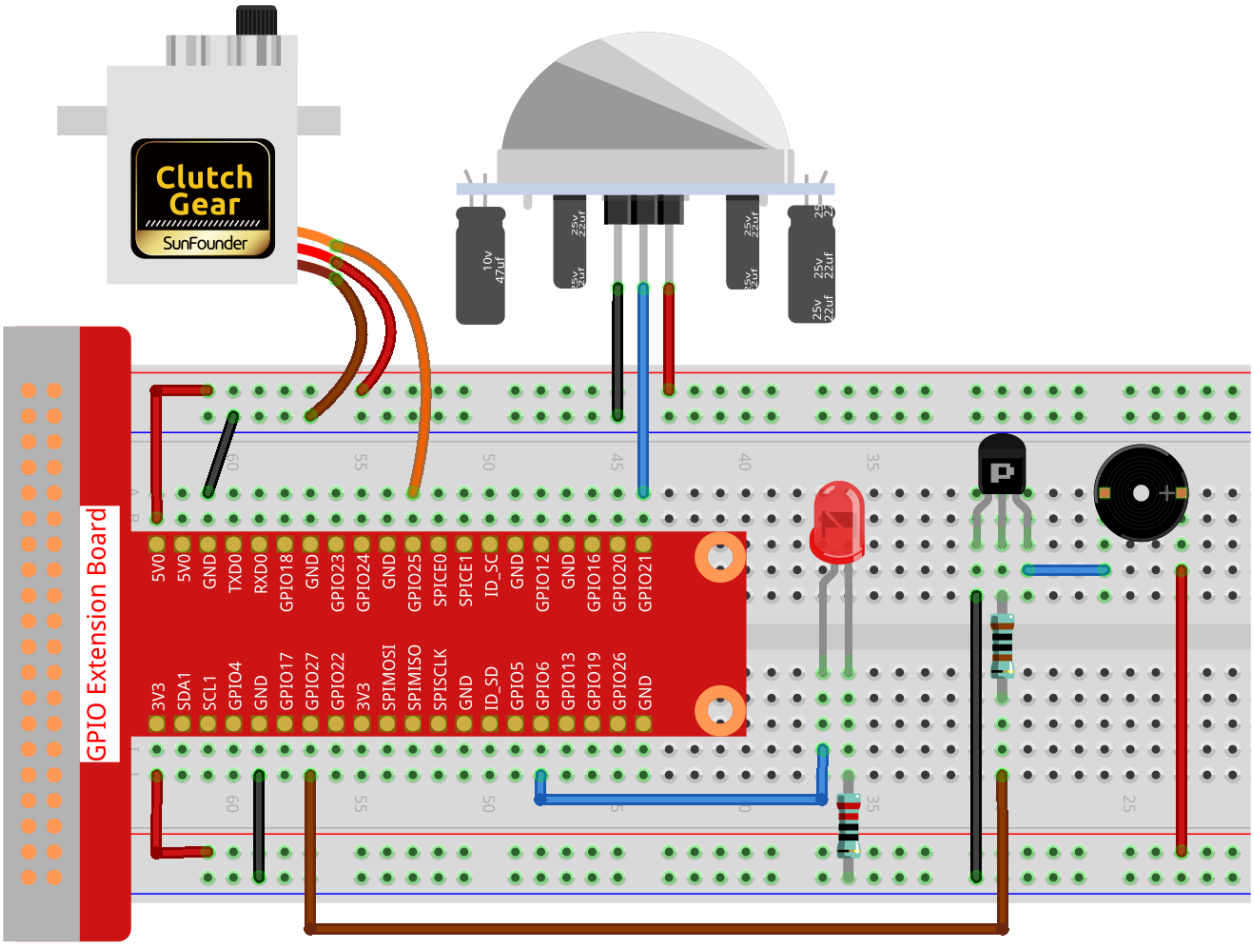
Schaltplan

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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Wechseln Sie das Verzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.2_Welcome_zero.py
```

Nachdem der Code ausgeführt wurde, öffnet sich automatisch die Tür (simuliert durch den Servo), das Anzeigelicht schaltet sich ein und die Türklingelmusik wird abgespielt, wenn der PIR-Sensor jemanden erfasst. Nachdem die Türklingelmusik abgespielt wurde, schließt das System automatisch die Tür und schaltet das Anzeigelicht aus, um auf das nächste Vorbeigehen zu warten.

Am PIR-Modul befinden sich zwei Potentiometer: eines zur Einstellung der Empfindlichkeit und das andere zur Einstellung der Erfassungsentfernung. Um das PIR-Modul optimal zu nutzen, sollten Sie beide gegen den Uhrzeigersinn bis zum Anschlag drehen.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

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

from gpiozero import LED, MotionSensor, Servo, TonalBuzzer
import time

# GPIO-Pin-Setup für LED, Bewegungssensor (PIR) und Summer
ledPin = LED(6)
pirPin = MotionSensor(21)
buzPin = TonalBuzzer(27)

# Servomotor-Pulsbreitenkorrekturfaktor und -berechnung
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # Maximale Pulsbreite
minPW = (1.0 - myCorrection) / 1000  # Minimale Pulsbreite

# Initialisierung des Servos mit angepassten Pulsbreiten
servoPin = Servo(25, min_pulse_width=minPW, max_pulse_width=maxPW)

# Musikalische Melodie für den Summer, mit Noten und Dauern
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),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
        ('D4', 0.2), ('E4', 0.2)]

def setAngle(angle):
    """
    Bewegt den Servo auf einen bestimmten Winkel.
    :param angle: Winkel in Grad (0-180).
    """
    value = float(angle / 180)  # Winkel in Servo-Wert umrechnen
    servoPin.value = value      # Servoposition setzen
    time.sleep(0.001)           # Kurze Verzögerung für Servobewegung

def doorbell():
    """
    Spielt eine musikalische Melodie mit dem Summer.
    """
    for note, duration in tune:
        buzPin.play(note)       # Note abspielen
        time.sleep(float(duration))  # Dauer der Note
    buzPin.stop()               # Summer nach dem Spielen der Melodie anhalten

def closedoor():
    # LED ausschalten und Servo bewegen, um Tür zu schließen
    ledPin.off()
    for i in range(180, -1, -1):
        setAngle(i)             # Servo von 180 auf 0 Grad bewegen
        time.sleep(0.001)       # Kurze Verzögerung für gleichmäßige Bewegung
    time.sleep(1)               # Warten nach dem Schließen der Tür

def opendoor():
    # LED einschalten, Tür öffnen (Servo bewegen), Melodie abspielen, Tür schließen
    ledPin.on()
    for i in range(0, 181):
        setAngle(i)             # Servo von 0 auf 180 Grad bewegen
        time.sleep(0.001)       # Kurze Verzögerung für gleichmäßige Bewegung
    time.sleep(1)               # Warten vor dem Spielen der Melodie
    doorbell()                  # Türklingelmelodie abspielen
    closedoor()                 # Tür nach der Melodie schließen

def loop():
    # Hauptschleife zur Bewegungserkennung und Türbedienung
    while True:
        if pirPin.motion_detected:
            opendoor()              # Tür öffnen, wenn Bewegung erkannt wird
        time.sleep(0.1)             # Kurze Verzögerung in der Schleife

try:
    loop()
except KeyboardInterrupt:
    # GPIO bei Benutzerunterbrechung aufräumen (z. B. Strg+C)
    buzPin.stop()
    ledPin.off()
```

Code-Erklärung

1. Das Skript beginnt mit dem Import der notwendigen Module. Die Bibliothek `gpiozero` wird verwendet, um mit der LED, dem Bewegungssensor, dem Servomotor und dem tonalen Summer zu interagieren. Das Modul `time` wird für zeitbezogene Funktionen eingesetzt.

```
#!/usr/bin/env python3
from gpiozero import LED, MotionSensor, Servo, TonalBuzzer
import time
```

2. Initialisiert die LED, den PIR-Bewegungssensor und den tonalen Summer an ihren jeweiligen GPIO-Pins.

```
# GPIO-Pin-Setup für LED, Bewegungssensor (PIR) und Summer
ledPin = LED(6)
pirPin = MotionSensor(21)
buzPin = TonalBuzzer(27)
```

3. Berechnet die maximalen und minimalen Pulsbreiten für den Servomotor unter Berücksichtigung eines Korrekturfaktors zur Feinabstimmung.

```
# Servomotor-Pulsbreitenkorrekturfaktor und -berechnung
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # Maximale Pulsbreite
minPW = (1.0 - myCorrection) / 1000  # Minimale Pulsbreite
```

4. Initialisiert den Servomotor am GPIO-Pin 25 mit den angepassten Pulsbreiten für eine genaue Positionierung.

```
# Initialisierung des Servos mit angepassten Pulsbreiten
servoPin = Servo(25, min_pulse_width=minPW, max_pulse_width=maxPW)
```

5. Die Melodie wird als Abfolge von Noten (Frequenz) und Dauern (Sekunden) definiert.

```
# Musikalische Melodie für den Summer, mit Noten und Dauern
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. Funktion zum Bewegen des Servos auf einen bestimmten Winkel. Wandelt den Winkel in einen Wert zwischen 0 und 1 für den Servo um.

```
def setAngle(angle):
    """
    Bewegt den Servo auf einen bestimmten Winkel.
    :param angle: Winkel in Grad (0-180).
    """
    value = float(angle / 180)  # Winkel in Servo-Wert umrechnen
    servoPin.value = value      # Servoposition setzen
    time.sleep(0.001)           # Kurze Verzögerung für Servobewegung
```

7. Funktion zum Abspielen einer musikalischen Melodie mit dem Summer. Durchläuft die Liste `tune` und spielt jede Note für ihre angegebene Dauer ab.

```
def doorbell():
    """
    Spielt eine musikalische Melodie mit dem Summer.
    """
    for note, duration in tune:
        buzPin.play(note)       # Note abspielen
        time.sleep(float(duration))  # Dauer der Note
    buzPin.stop()               # Summer nach dem Spielen der Melodie anhalten
```

8. Funktionen zum Öffnen und Schließen der Tür mit dem Servomotor. Die Funktion `opendoor` schaltet die LED ein, öffnet die Tür, spielt die Melodie und schließt dann die Tür.

```
def closedoor():
    # LED ausschalten und Servo bewegen, um Tür zu schließen
    ledPin.off()
    for i in range(180, -1, -1):
        setAngle(i)             # Servo von 180 auf 0 Grad bewegen
        time.sleep(0.001)       # Kurze Verzögerung für gleichmäßige Bewegung
    time.sleep(1)               # Warten nach dem Schließen der Tür

def opendoor():
    # LED einschalten, Tür öffnen (Servo bewegen), Melodie abspielen, Tür schließen
    ledPin.on()
    for i in range(0, 181):
        setAngle(i)             # Servo von 0 auf 180 Grad bewegen
        time.sleep(0.001)       # Kurze Verzögerung für gleichmäßige Bewegung
    time.sleep(1)               # Warten vor dem Spielen der Melodie
    doorbell()                  # Türklingelmelodie abspielen
    closedoor()                 # Tür nach der Melodie schließen
```

9. Hauptschleife, die ständig auf Bewegungserkennung prüft. Bei erkannter Bewegung wird die Funktion `opendoor` ausgelöst.

```
def loop():
    # Hauptschleife zur Bewegungserkennung und Türbedienung
    while True:
        if pirPin.motion_detected:
            opendoor()              # Tür öffnen, wenn Bewegung erkannt wird
        time.sleep(0.1)             # Kurze Verzögerung in der Schleife
```

10. Führt die Hauptschleife aus und stellt sicher, dass das Skript mit einem Tastaturbefehl (Strg+C) gestoppt werden kann, wobei der Summer und die LED für einen sauberen Ausstieg ausgeschaltet werden.

```
try:
    loop()
except KeyboardInterrupt:
    # GPIO bei Benutzerunterbrechung aufräumen (z. B. Strg+C)
    buzPin.stop()
    ledPin.off()
```

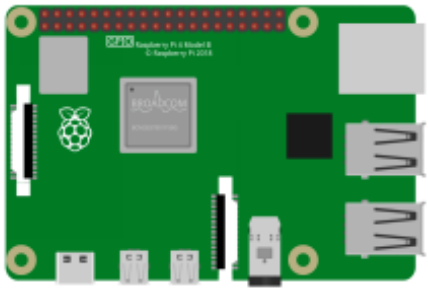
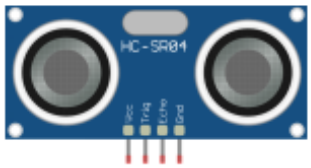
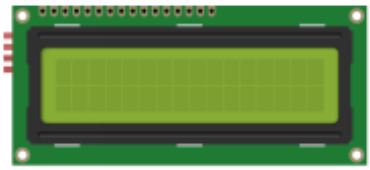
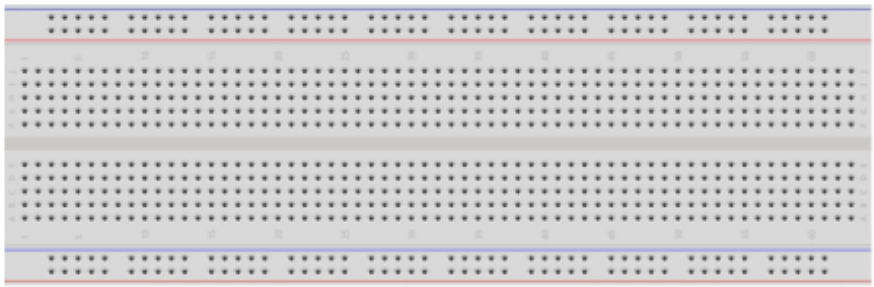
3.1.3 Rückfahrwarnung

Einführung

In diesem Projekt werden wir ein LCD, einen Summer und Ultraschallsensoren verwenden, um ein Rückfahrassistenzsystem zu erstellen. Wir können es auf ein ferngesteuertes Fahrzeug setzen, um den tatsächlichen Prozess des Rückwärtsfahrens in eine Garage zu simulieren.

Benötigte Komponenten

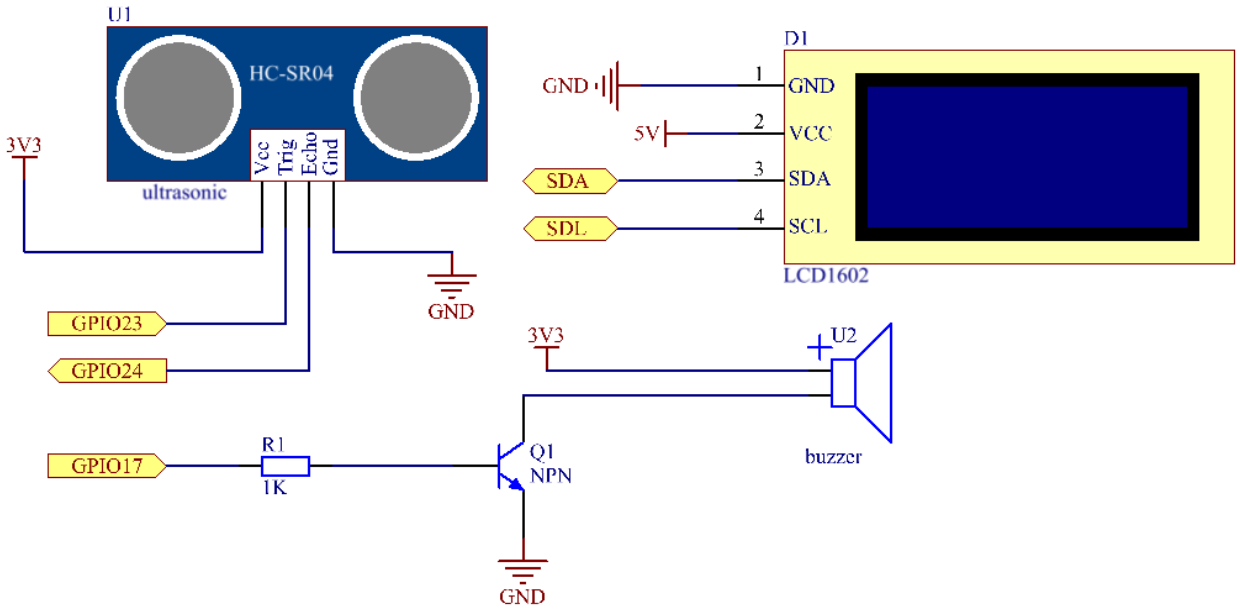
Für dieses Projekt benötigen wir folgende Komponenten.

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 		

Schaltplan

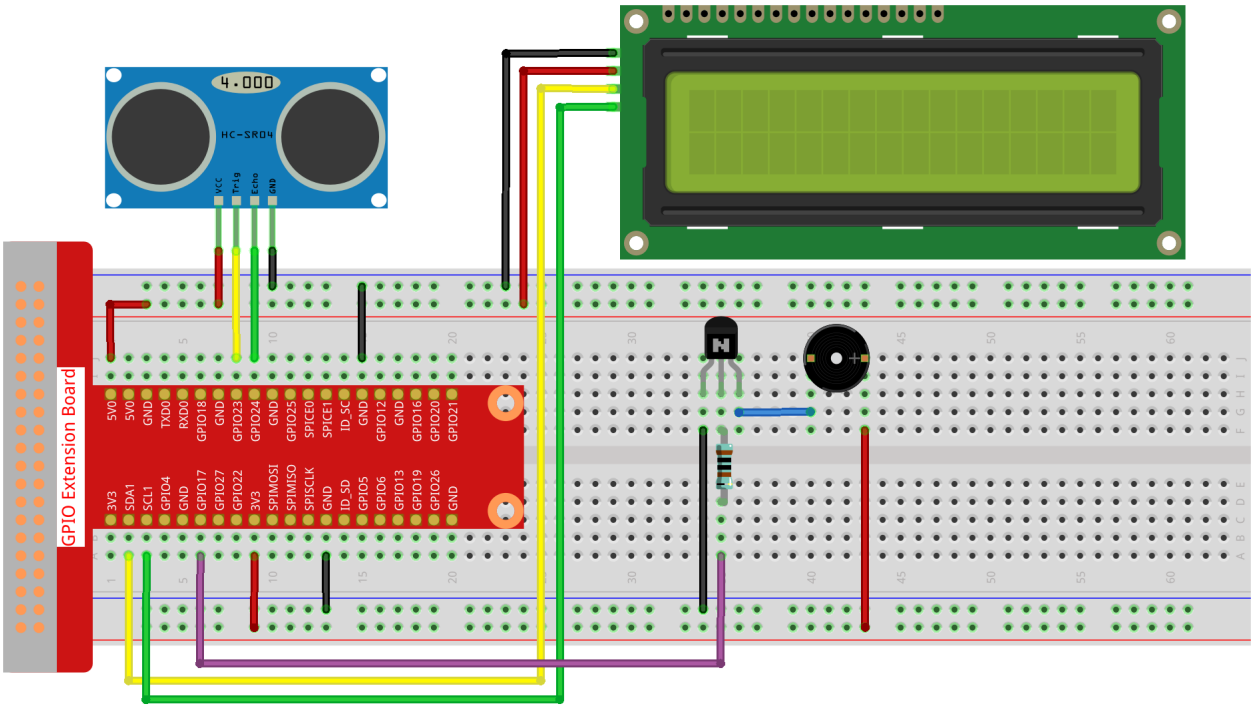
Der Ultraschallsensor erfasst die Entfernung zwischen sich und dem Hindernis, die auf dem LCD in Form von Code angezeigt wird. Gleichzeitig lässt der Ultraschallsensor den Summer Tonsignale unterschiedlicher Frequenz ausgeben, je nach Entfernungswert.

T-Board Name	physisch	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Wechseln Sie das Verzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.3_ReversingAlarm_zero.py
```

Während der Code ausgeführt wird, erkennt der Ultraschallsensormodul die Entfernung zum Hindernis und zeigt die Informationen über die Entfernung auf dem LCD1602 an; außerdem gibt der Summer ein Warnsignal aus, dessen Frequenz sich mit der Entfernung ändert.

Bemerkung:

- Wenn Sie den Fehler `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` erhalten, beziehen Sie sich bitte auf *I2C-Konfiguration*, um den I2C zu aktivieren.
- Bei der Fehlermeldung `ModuleNotFoundError: No module named 'smbus2'` führen Sie bitte `sudo pip3 install smbus2` aus.
- Wenn der Fehler `OSError: [Errno 121] Remote I/O error` auftritt, bedeutet dies, dass das Modul falsch verkabelt ist oder defekt ist.
- Wenn der Code und die Verkabelung in Ordnung sind, das LCD jedoch keinen Inhalt anzeigt, können Sie das Potentiometer auf der Rückseite drehen, um den Kontrast zu erhöhen.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

```
#!/usr/bin/env python3
import LCD1602
import time
from gpiozero import DistanceSensor, Buzzer

# Initialisierung des Ultraschallsensors mit GPIO-Pins
sensor = DistanceSensor(echo=24, trigger=23)  # Echo-Pin ist GPIO 24, Trigger-Pin ist␣
↪GPIO 23

# Initialisierung des Buzzers an GPIO-Pin 17
buzzer = Buzzer(17)

def lcdsetup():
    # Initialisierung des LCD mit I2C-Adresse und Aktivierung der Hintergrundbeleuchtung
    LCD1602.init(0x27, 1)  # 0x27 ist die I2C-Adresse des LCD
    LCD1602.clear()  # LCD-Display löschen
    # Startnachrichten auf dem LCD anzeigen
    LCD1602.write(0, 0, 'Ultraschall Start')
    LCD1602.write(1, 1, 'Von SunFounder')
    time.sleep(2)  # 2 Sekunden warten

def distance():
    # Berechnet und gibt die gemessene Entfernung des Sensors zurück
    dis = sensor.distance * 100  # Entfernung in Zentimeter umrechnen
    print('Entfernung: {:.2f} cm'.format(dis))  # Entfernung mit zwei Dezimalstellen␣
↪anzeigen
    time.sleep(0.3)  # 0.3 Sekunden warten, bevor die nächste Messung erfolgt
    return dis

def loop():
    # Kontinuierliche Entfernungsmessung und Aktualisierung von LCD und Summer
    while True:
        dis = distance()  # Aktuelle Entfernung ermitteln
        # Entfernung anzeigen und Alarme basierend auf der Entfernung handhaben
        if dis > 400:  # Prüfen, ob die Entfernung außerhalb des Bereichs liegt
            LCD1602.clear()
            LCD1602.write(0, 0, 'Fehler')
            LCD1602.write(3, 1, 'Außer Reichweite')
            time.sleep(0.5)
        else:
            # Aktuelle Entfernung auf dem LCD anzeigen
            LCD1602.clear()
            LCD1602.write(0, 0, 'Entfernung ist')
            LCD1602.write(5, 1, str(round(dis, 2)) + ' cm')
            # Summerfrequenz basierend auf der Entfernung anpassen
            if dis >= 50:
                time.sleep(0.5)
            elif 20 < dis < 50:
                # Mittlere Entfernung: mittlere Summerfrequenz
                for _ in range(2):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                time.sleep(0.2)
            elif dis <= 20:
                # Nahe Entfernung: hohe Summerfrequenz
                for _ in range(5):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.05)

try:
    lcdsetup()  # LCD-Display einrichten
    loop()      # Messschleife starten
except KeyboardInterrupt:
    # Summer ausschalten und LCD löschen bei Benutzerunterbrechung (z. B. Strg+C)
    buzzer.off()
    LCD1602.clear()
```

Code-Erklärung

1. Dieses Skript nutzt verschiedene Bibliotheken für seine Funktionalität. Die Bibliothek `LCD1602` verwaltet das LCD-Display, während `time` zeitbezogene Funktionen bereitstellt. Die Bibliothek `gpiozero` ist essentiell für die Interaktion mit den GPIO-Pins des Raspberry Pi, insbesondere für den Betrieb des DistanceSensor und des Buzzers.

   ```
   #!/usr/bin/env python3
   import LCD1602
   import time
   from gpiozero import DistanceSensor, Buzzer
   ```

2. Der Ultraschallsensor wird mit seinen Echo- und Trigger-Pins an die GPIO-Pins 24 und 23 angeschlossen. Zusätzlich ist ein Summer an GPIO-Pin 17 angeschlossen.

   ```
   # Initialisierung des Ultraschallsensors mit GPIO-Pins
   sensor = DistanceSensor(echo=24, trigger=23)  # Echo-Pin ist GPIO 24, Trigger-Pin␣
   ↪ist GPIO 23

   # Initialisierung des Buzzers an GPIO-Pin 17
   buzzer = Buzzer(17)
   ```

3. Die Initialisierung des LCD-Displays umfasst das Löschen und Anzeigen einer Startnachricht.

   ```
   def lcdsetup():
       # Initialisierung des LCD mit I2C-Adresse und Aktivierung der␣
   ↪Hintergrundbeleuchtung
       LCD1602.init(0x27, 1)  # 0x27 ist die I2C-Adresse des LCD
       LCD1602.clear()  # LCD-Display löschen
       # Startnachrichten auf dem LCD anzeigen
       LCD1602.write(0, 0, 'Ultraschall Start')
       LCD1602.write(1, 1, 'Von SunFounder')
       time.sleep(2)  # 2 Sekunden warten
   ```

4. Die Funktion `distance` berechnet die vom Ultraschallsensor gemessene Entfernung und gibt sie aus, wobei der Wert in Zentimeter zurückgegeben wird.

```
def distance():
    # Berechnet und gibt die gemessene Entfernung des Sensors zurück
    dis = sensor.distance * 100  # Entfernung in Zentimeter umrechnen
    print('Entfernung: {:.2f} cm'.format(dis))  # Entfernung mit zwei␣
↪Dezimalstellen anzeigen
    time.sleep(0.3)  # 0.3 Sekunden warten, bevor die nächste Messung erfolgt
    return dis
```

5. Die Hauptschleife misst kontinuierlich die Entfernung und aktualisiert sowohl das LCD als auch den Summer. Sie behandelt unterschiedliche Entfernungsbereiche mit spezifischen Aktionen, wie das Anzeigen von Fehlermeldungen oder das Variieren der Summerfrequenz basierend auf der gemessenen Entfernung.

```
def loop():
    # Kontinuierliche Entfernungsmessung und Aktualisierung von LCD und Summer
    while True:
        dis = distance()  # Aktuelle Entfernung ermitteln
        # Entfernung anzeigen und Alarme basierend auf der Entfernung handhaben
        if dis > 400:  # Prüfen, ob die Entfernung außerhalb des Bereichs liegt
            LCD1602.clear()
            LCD1602.write(0, 0, 'Fehler')
            LCD1602.write(3, 1, 'Außer Reichweite')
            time.sleep(0.5)
        else:
            # Aktuelle Entfernung auf dem LCD anzeigen
            LCD1602.clear()
            LCD1602.write(0, 0, 'Entfernung ist')
            LCD1602.write(5, 1, str(round(dis, 2)) + ' cm')
            # Summerfrequenz basierend auf der Entfernung anpassen
            if dis >= 50:
                time.sleep(0.5)
            elif 20 < dis < 50:
                # Mittlere Entfernung: mittlere Summerfrequenz
                for _ in range(2):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.2)
            elif dis <= 20:
                # Nahe Entfernung: hohe Summerfrequenz
                for _ in range(5):
                    buzzer.on()
                    time.sleep(0.05)
                    buzzer.off()
                    time.sleep(0.05)
```

6. Bei der Ausführung richtet das Skript das LCD ein und tritt in die Hauptschleife ein. Es kann mit einem Tastaturbefehl (Strg+C) unterbrochen werden, wodurch der Summer ausgeschaltet und das LCD gelöscht wird.

```
try:
    lcdsetup()  # LCD-Display einrichten
    loop()      # Messschleife starten
except KeyboardInterrupt:
    # Summer ausschalten und LCD löschen bei Benutzerunterbrechung (z. B. Strg+C)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    buzzer.off()
    LCD1602.clear()
```

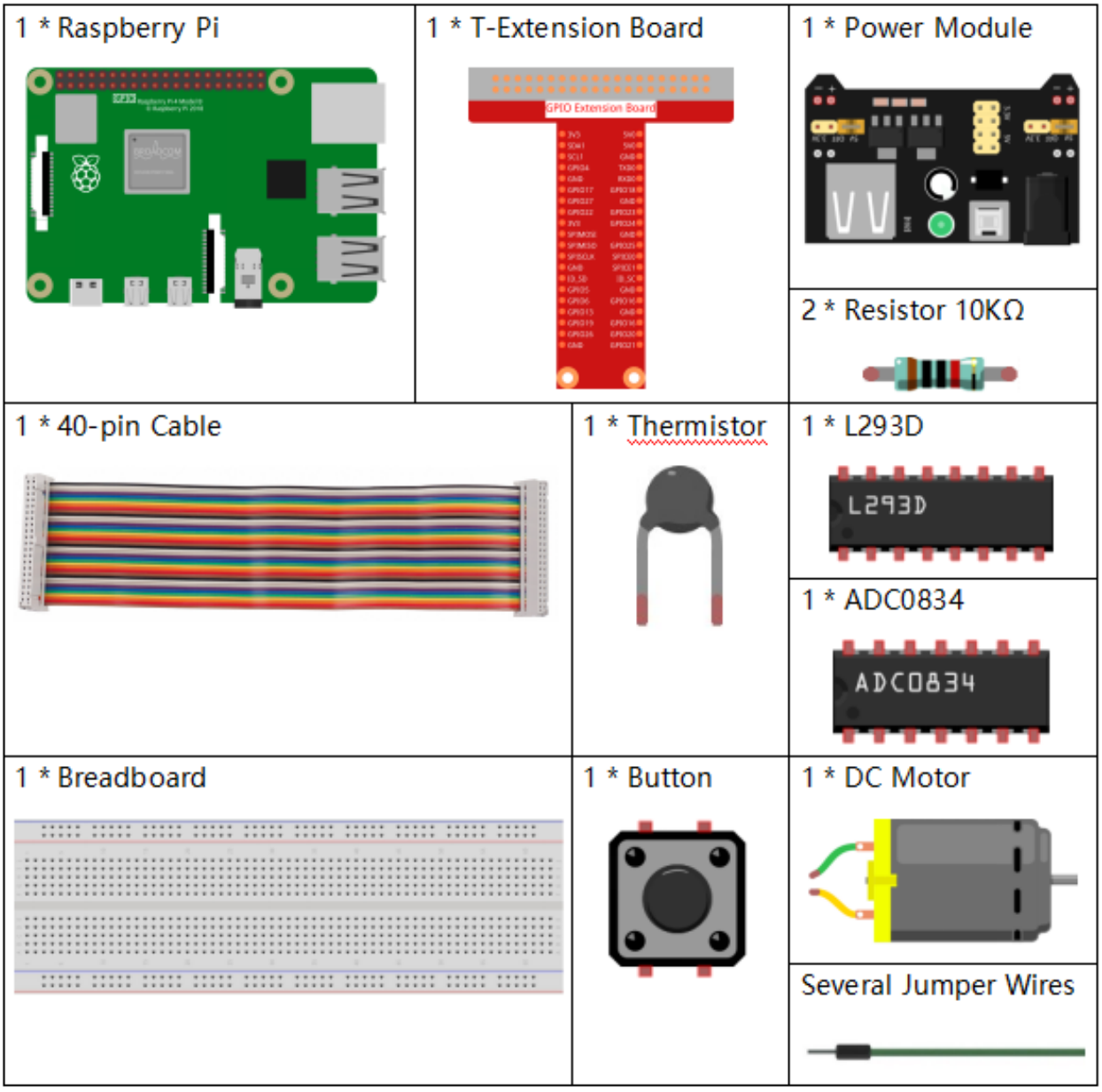
3.1.4 Intelligenter Ventilator

Einführung

In diesem Projekt werden wir Motoren, Tasten und Thermistoren verwenden, um einen manuellen + automatischen intelligenten Ventilator zu bauen, dessen Windgeschwindigkeit einstellbar ist.

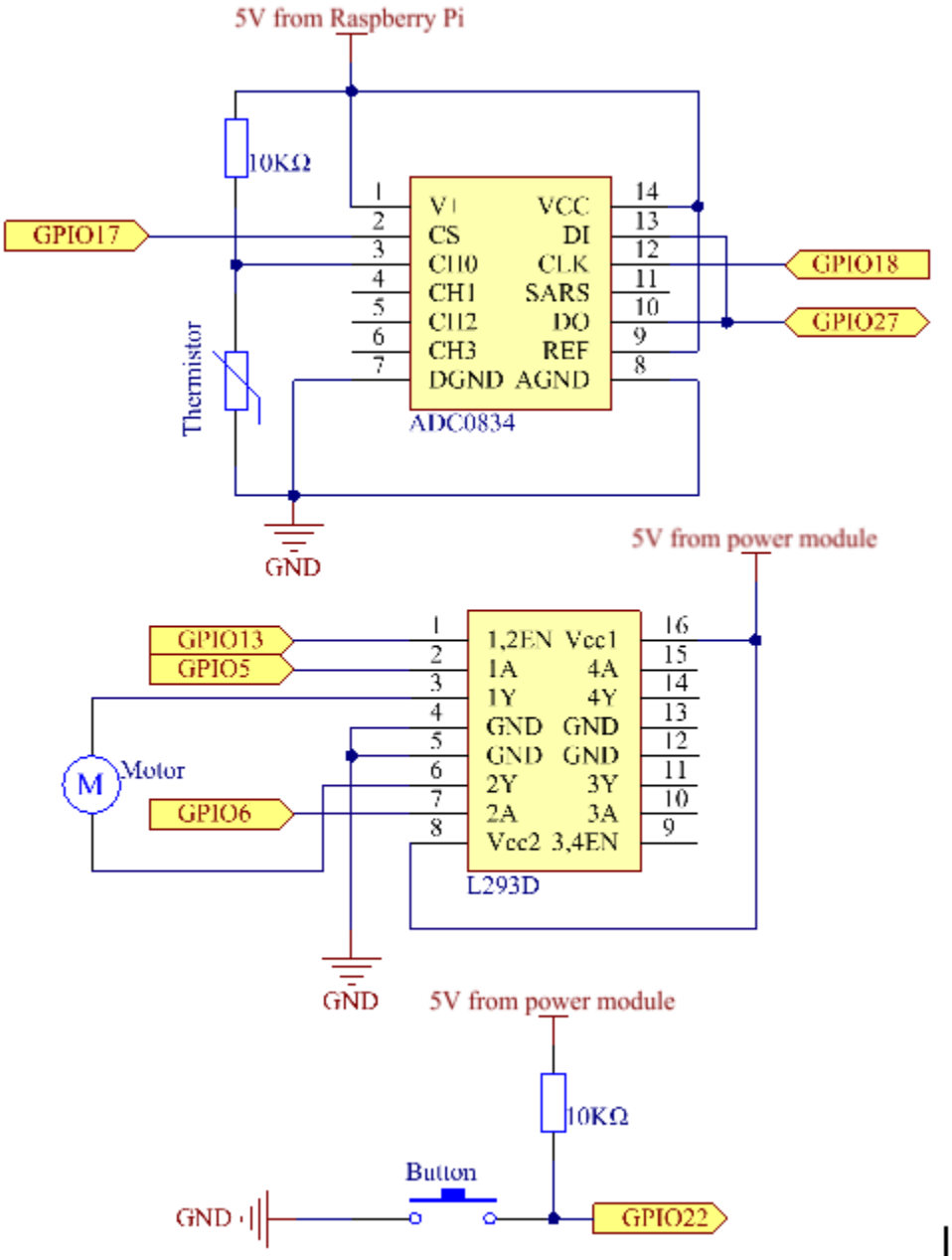
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



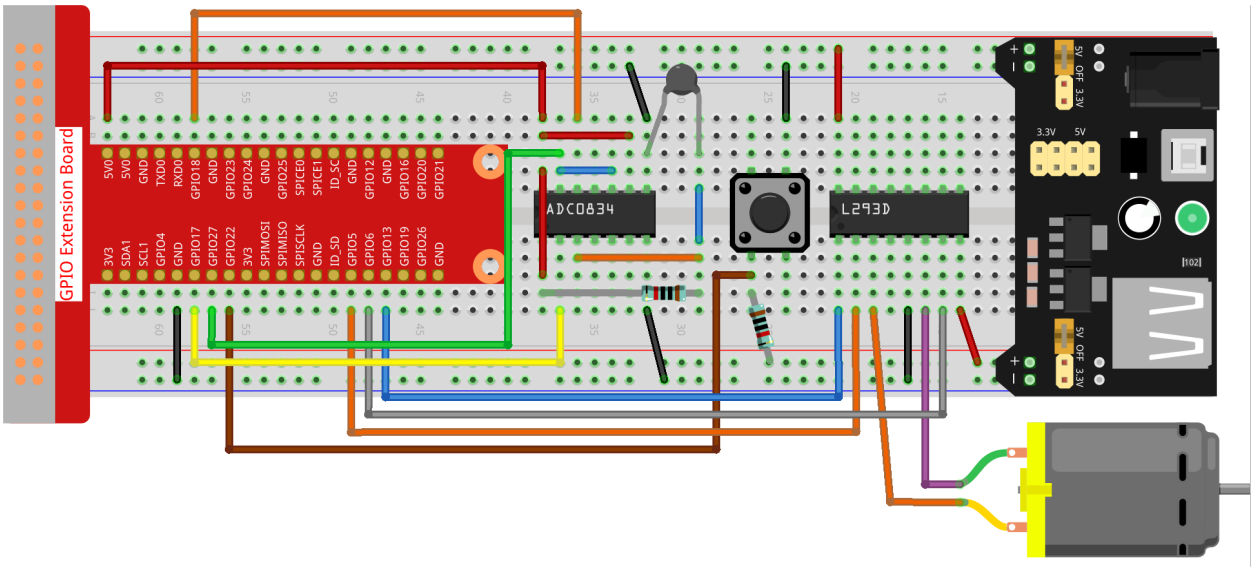
Schaltplan

T-Board Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Bemerkung: Das Powermodul kann mit dem 9V-Batterieclip im Kit eine 9V-Batterie verwenden. Stecken Sie den Jumper des Powermoduls in die 5V-Steckleisten des Steckbretts.

Schritt 2: Wechseln Sie in den Ordner des Codes.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.4_SmartFan_zero.py
```

Wenn der Code ausgeführt wird, starten Sie den Ventilator, indem Sie die Taste drücken. Jedes Mal, wenn Sie drücken, wird die Geschwindigkeitsstufe um 1 erhöht oder verringert. Es gibt **5** Geschwindigkeitsstufen: **0~4**. Wenn Sie auf die 4. Stufe eingestellt sind und die Taste drücken, stoppt der Ventilator und zeigt eine Windgeschwindigkeit von **0** an.

Steigt oder fällt die Temperatur um mehr als 2℃, wird die Geschwindigkeit automatisch um 1 Stufe schneller oder langsamer eingestellt.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

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

from gpiozero import Motor, Button
from time import sleep
import ADC0834
import math

# Initialisierung der GPIO-Pins für die Taste und die Motorsteuerung
BtnPin = Button(22)
motor = Motor(forward=5, backward=6, enable=13)

# Initialisierung des ADC0834-Moduls für die Temperaturmessung
ADC0834.setup()

# Initialisierung von Variablen zur Überwachung der Motordrehzahl und Temperaturen
level = 0
currentTemp = 0
markTemp = 0

def temperature():
    """
    Liest und berechnet die aktuelle Temperatur vom Sensor.
    Rückgabe:
        float: Die aktuelle Temperatur in Celsius.
    """
    # Analogwert vom ADC0834-Modul lesen
    analogVal = ADC0834.getResult()
    # Analogwert in Spannung und dann in Widerstand umwandeln
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    # Temperatur in Celsius berechnen
    temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))
    Cel = temp - 273.15
    return Cel

def motor_run(level):
    """
    Passt die Motorgeschwindigkeit an das angegebene Niveau an.
    Argumente:
        level (int): Gewünschtes Motorgeschwindigkeitsniveau.
    Rückgabe:
        int: Angepasstes Motorgeschwindigkeitsniveau.
    """
    # Motor stoppen, wenn das Niveau 0 ist
    if level == 0:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        motor.stop()
        return 0
    # Niveau auf 4 begrenzen für maximale Geschwindigkeit
    if level >= 4:
        level = 4
    # Motorgeschwindigkeit einstellen
    motor.forward(speed=float(level / 4))
    return level

def changeLevel():
    """
    Ändert das Motorgeschwindigkeitsniveau, wenn die Taste gedrückt wird, und␣
↪aktualisiert die Referenztemperatur.
    """
    global level, currentTemp, markTemp
    print("Taste gedrückt")
    # Durch die Niveaus 0-4 zyklisch wechseln
    level = (level + 1) % 5
    # Referenztemperatur aktualisieren
    markTemp = currentTemp

# Die Taste mit der Funktion changeLevel verbinden
BtnPin.when_pressed = changeLevel

def main():
    """
    Hauptfunktion zur kontinuierlichen Überwachung und Reaktion auf Temperaturänderungen.
    """
    global level, currentTemp, markTemp
    # Anfängliche Referenztemperatur setzen
    markTemp = temperature()
    while True:
        # Aktuelle Temperatur kontinuierlich lesen
        currentTemp = temperature()
        # Motorniveau anhand der Temperaturdifferenz anpassen
        if level != 0:
            if currentTemp - markTemp <= -2:
                level -= 1
                markTemp = currentTemp
            elif currentTemp - markTemp >= 2:
                if level < 4:
                    level += 1
                markTemp = currentTemp
        # Motor auf angepasstem Niveau betreiben
        level = motor_run(level)

# Die Hauptfunktion ausführen und auf KeyboardInterrupt reagieren
try:
    main()
except KeyboardInterrupt:
    # Motor stoppen, wenn das Skript unterbrochen wird
    motor.stop()
```

Code-Erklärung

1. Importiert Klassen zur Steuerung eines Motors und einer Taste sowie die Sleep-Funktion für Pausen. Außerdem wird die ADC0834-Bibliothek für die Temperaturmessung und die Math-Bibliothek für mathematische Berechnungen importiert.

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

from gpiozero import Motor, Button
from time import sleep
import ADC0834
import math
```

2. Richtet die Taste an GPIO-Pin 22 ein und konfiguriert den Motor mit spezifischen GPIO-Pins zur Steuerung. Initialisiert das ADC0834-Modul zur Temperaturmessung. Außerdem werden Variablen initialisiert, um die Motordrehzahl und Temperaturen zu überwachen.

```
# Initialisierung der GPIO-Pins für die Taste und die Motorsteuerung
BtnPin = Button(22)
motor = Motor(forward=5, backward=6, enable=13)

# Initialisierung des ADC0834-Moduls für die Temperaturmessung
ADC0834.setup()

# Initialisierung von Variablen zur Überwachung der Motordrehzahl und Temperaturen
level = 0
currentTemp = 0
markTemp = 0
```

3. Definiert eine Funktion zur Ablesung und Berechnung der Temperatur vom Sensor, die den Messwert in Celsius umwandelt.

```
def temperature():
    """
    Liest und berechnet die aktuelle Temperatur vom Sensor.
    Rückgabe:
        float: Die aktuelle Temperatur in Celsius.
    """
    # Analogwert vom ADC0834-Modul lesen
    analogVal = ADC0834.getResult()
    # Analogwert in Spannung und dann in Widerstand umwandeln
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    # Temperatur in Celsius berechnen
    temp = 1 / (((math.log(Rt / 10000)) / 3950) + (1 / (273.15 + 25)))
    Cel = temp - 273.15
    return Cel
```

4. Führt eine Funktion ein, um die Motorgeschwindigkeit entsprechend dem angegebenen Niveau anzupassen.

```
def motor_run(level):
    """
    Passt die Motorgeschwindigkeit an das angegebene Niveau an.
    Argumente:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        level (int): Gewünschtes Motorgeschwindigkeitsniveau.
    Rückgabe:
        int: Angepasstes Motorgeschwindigkeitsniveau.
    """
    # Motor stoppen, wenn das Niveau 0 ist
    if level == 0:
        motor.stop()
        return 0
    # Niveau auf 4 begrenzen für maximale Geschwindigkeit
    if level >= 4:
        level = 4
    # Motorgeschwindigkeit einstellen
    motor.forward(speed=float(level / 4))
    return level
```

5. Implementiert eine Funktion, um das Motorgeschwindigkeitsniveau manuell mithilfe einer Taste zu ändern und bindet diese Funktion an das Drücken der Taste.

```
def changeLevel():
    """
    Ändert das Motorgeschwindigkeitsniveau, wenn die Taste gedrückt wird, und␣
↪aktualisiert die Referenztemperatur.
    """
    global level, currentTemp, markTemp
    print("Taste gedrückt")
    # Durch die Niveaus 0-4 zyklisch wechseln
    level = (level + 1) % 5
    # Referenztemperatur aktualisieren
    markTemp = currentTemp

# Die Taste mit der Funktion changeLevel verbinden
BtnPin.when_pressed = changeLevel
```

6. Die Hauptfunktion, die entwickelt wurde, um die Motorgeschwindigkeit kontinuierlich in Reaktion auf Temperaturschwankungen anzupassen, muss noch implementiert werden.

```
def main():
    """
    Hauptfunktion zur kontinuierlichen Überwachung und Reaktion auf␣
↪Temperaturänderungen.
    """
    global level, currentTemp, markTemp
    # Anfängliche Referenztemperatur setzen
    markTemp = temperature()
    while True:
        # Aktuelle Temperatur kontinuierlich lesen
        currentTemp = temperature()
        # Motorniveau anhand der Temperaturdifferenz anpassen
        if level != 0:
            if currentTemp - markTemp <= -2:
                level -= 1
                markTemp = currentTemp
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            elif currentTemp - markTemp >= 2:
                if level < 4:
                    level += 1
                markTemp = currentTemp
        # Motor auf angepasstem Niveau betreiben
        level = motor_run(level)
```

7. Führt die Hauptfunktion aus und stellt sicher, dass der Motor stoppt, wenn das Skript unterbrochen wird.

```
# Die Hauptfunktion ausführen und auf KeyboardInterrupt reagieren
try:
    main()
except KeyboardInterrupt:
    # Motor stoppen, wenn das Skript unterbrochen wird
    motor.stop()
```

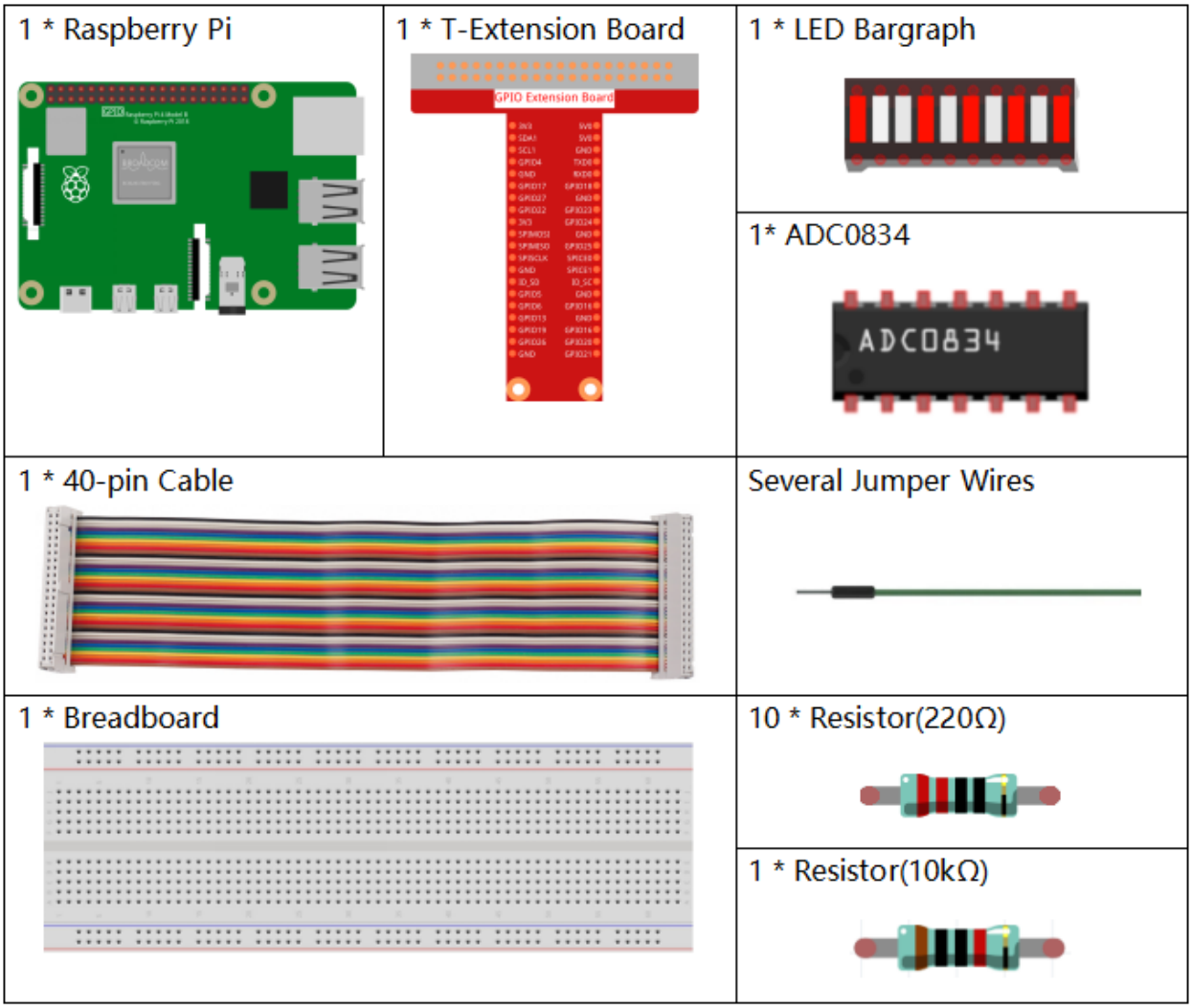
3.1.5 Batterieanzeige

Einführung

In diesem Projekt werden wir ein Batterieanzeigegerät erstellen, das den Batteriestand auf dem LED-Bargraphen visuell anzeigen kann.

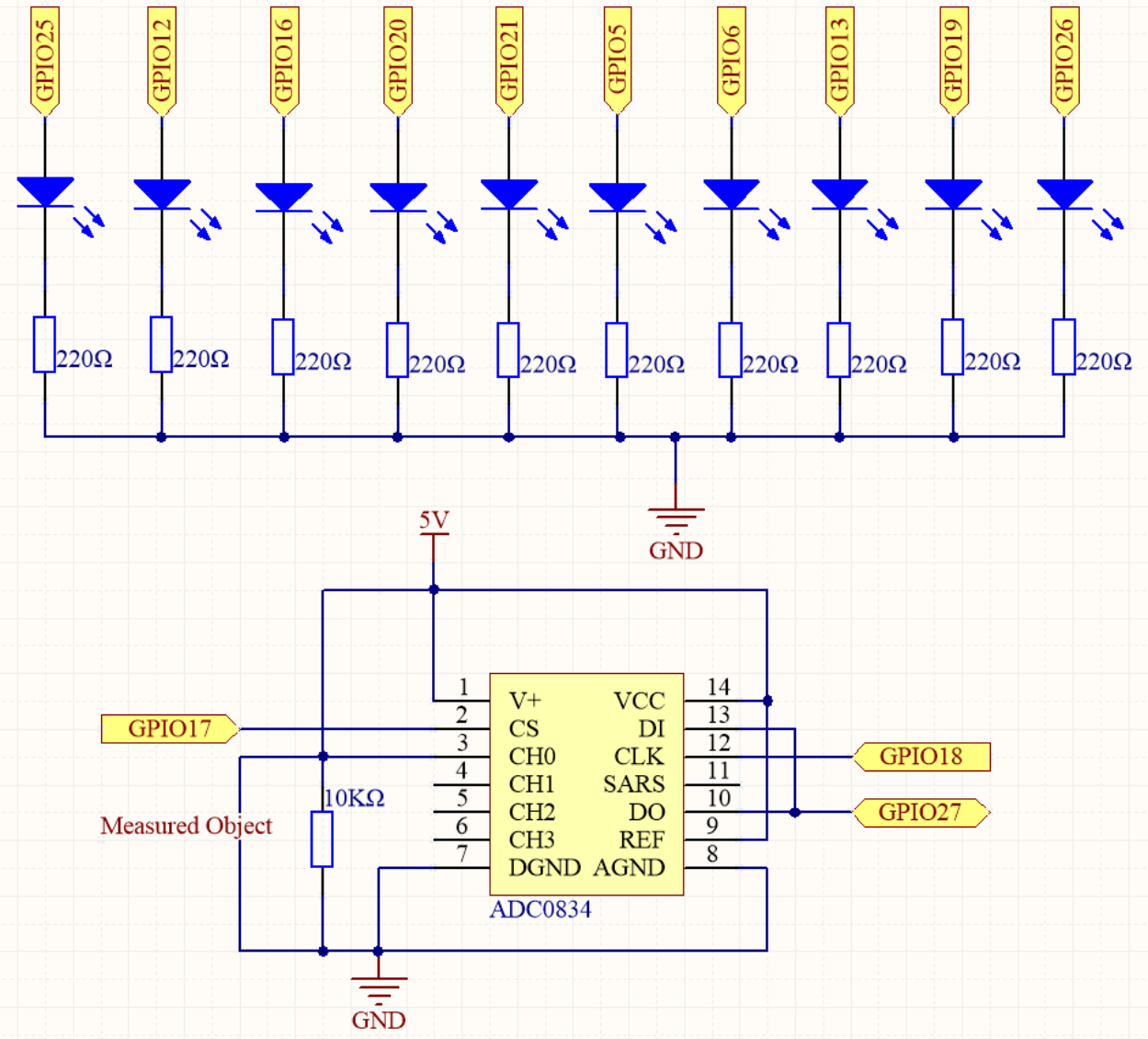
Erforderliche Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



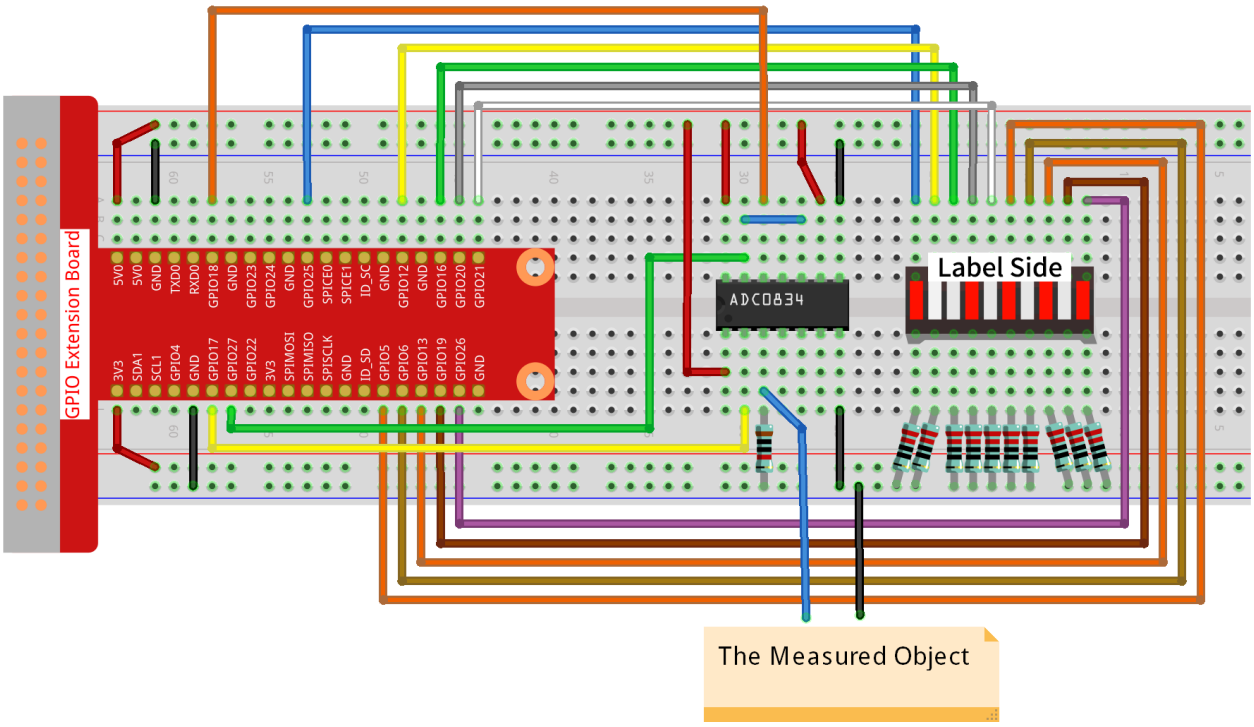
Schaltplan

T-Karte Name	physisch	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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Wechseln Sie in den Ordner mit dem Code.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.5_BatteryIndicator_zero.py
```

Nachdem das Programm ausgeführt wurde, führen Sie das dritte Pin des ADC0834 und die GND separat heraus und führen Sie sie dann separat zu den beiden Polen einer Batterie. Sie können sehen, dass die entsprechende LED auf dem LED-Bargraphen aufleuchtet, um den Energiepegel anzuzeigen (Messbereich: 0-5V).

Code

Bemerkung: Sie können den folgenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Aber bevor Sie das tun, müssen Sie zum Quellcodepfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import LED
import ADC0834
import time

# Liste der GPIO-Pins, an die LEDs angeschlossen sind
ledPins = [25, 12, 16, 20, 21, 5, 6, 13, 19, 26]
# Initialisieren von LED-Objekten für jeden Pin in der Liste
leds = [LED(pin) for pin in ledPins]

# Setup des ADC0834-Moduls
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
ADC0834.setup()

def LedBarGraph(value):
    # Schalte alle LEDs aus
    for i in range(10):
        leds[i].off()
    # Schalte LEDs bis zum angegebenen Wert ein
    for i in range(value):
        leds[i].on()

try:
    # Hauptschleife zur kontinuierlichen Aktualisierung des LED-Bargraphen
    while True:
        # Lese den analogen Wert von ADC0834 aus
        analogVal = ADC0834.getResult()
        # Konvertiere den analogen Wert in den LED-Bargraphen-Level
        LedBarGraph(int(analogVal/25))
except KeyboardInterrupt:
    # Schalte alle LEDs aus, wenn das Programm unterbrochen wird
    for i in range(10):
        leds[i].off()
```

Code-Erklärung

1. Dieser Abschnitt importiert die erforderlichen Bibliotheken. `gpiozero` dient zur Steuerung der LEDs, `ADC0834` zur Kommunikation mit dem ADC-Modul und `time` für zeitbezogene Operationen.

   ```
   #!/usr/bin/env python3
   from gpiozero import LED
   import ADC0834
   import time
   ```

2. Definiert die GPIO-Pins, an die die LEDs angeschlossen sind, und initialisiert ein Array von LED-Objekten für jeden Pin. Dadurch wird eine einfache Steuerung jeder LED im Array ermöglicht.

   ```
   # Liste der GPIO-Pins, an die die LEDs angeschlossen sind
   ledPins = [25, 12, 16, 20, 21, 5, 6, 13, 19, 26]
   # Initialisieren von LED-Objekten für jeden Pin in der Liste
   leds = [LED(pin) for pin in ledPins]
   ```

3. Initialisiert das ADC0834-Modul für die analog-digital-Umwandlung.

   ```
   # Setup des ADC0834-Moduls
   ADC0834.setup()
   ```

4. Diese Funktion schaltet zunächst alle LEDs aus und schaltet dann basierend auf dem Eingangswert eine bestimmte Anzahl von LEDs ein, um effektiv eine Balkendiagrammdarstellung zu erstellen.

   ```
   def LedBalkenDiagramm(wert):
       # Schalte alle LEDs aus
       for i in range(10):
           leds[i].off()
       # Schalte LEDs bis zum angegebenen Wert ein
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    for i in range(wert):
        leds[i].on()
```

5. Liest kontinuierlich den analogen Wert aus dem ADC0834 aus und aktualisiert das LED-Balkendiagramm basierend auf diesem Wert. Der analoge Wert wird auf einen Bereich von 0 bis 10 für die 10 LEDs skaliert.

```
try:
    # Hauptschleife zur kontinuierlichen Aktualisierung des LED-Bargraphen
    while True:
        # Lese den analogen Wert von ADC0834 aus
        analogVal = ADC0834.getResult()
        # Konvertiere den analogen Wert in den LED-Bargraphen-Level
        LedBalkenDiagramm(int(analogVal/25))

except KeyboardInterrupt:
    # Schalte alle LEDs aus, wenn das Programm unterbrochen wird
    for i in range(10):
        leds[i].off()
```

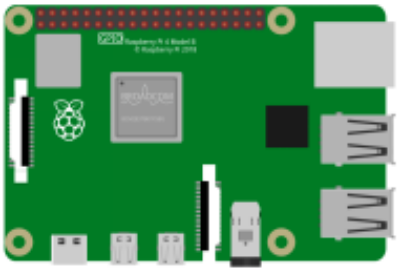
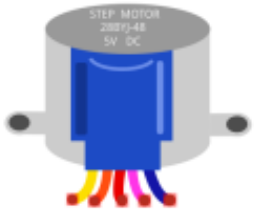
3.1.6 Bewegungssteuerung

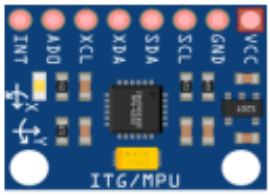
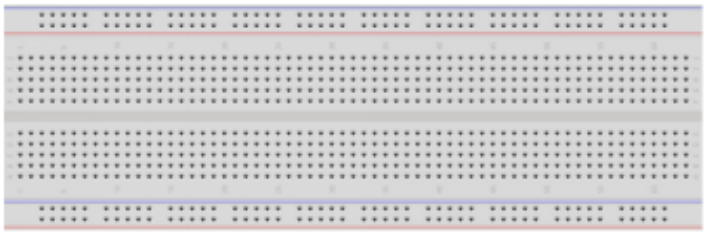
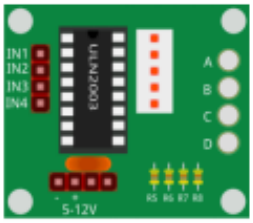
Einführung

In dieser Lektion werden wir ein einfaches Bewegungserkennungs- und -steuerungsgerät erstellen. Der MPU6050 wird als Sensor und der Schrittmotor als gesteuertes Gerät verwendet. Mit dem MPU6050, der am Handschuh befestigt ist, können Sie den Schrittmotor durch Drehen Ihres Handgelenks steuern.

Benötigte Komponenten

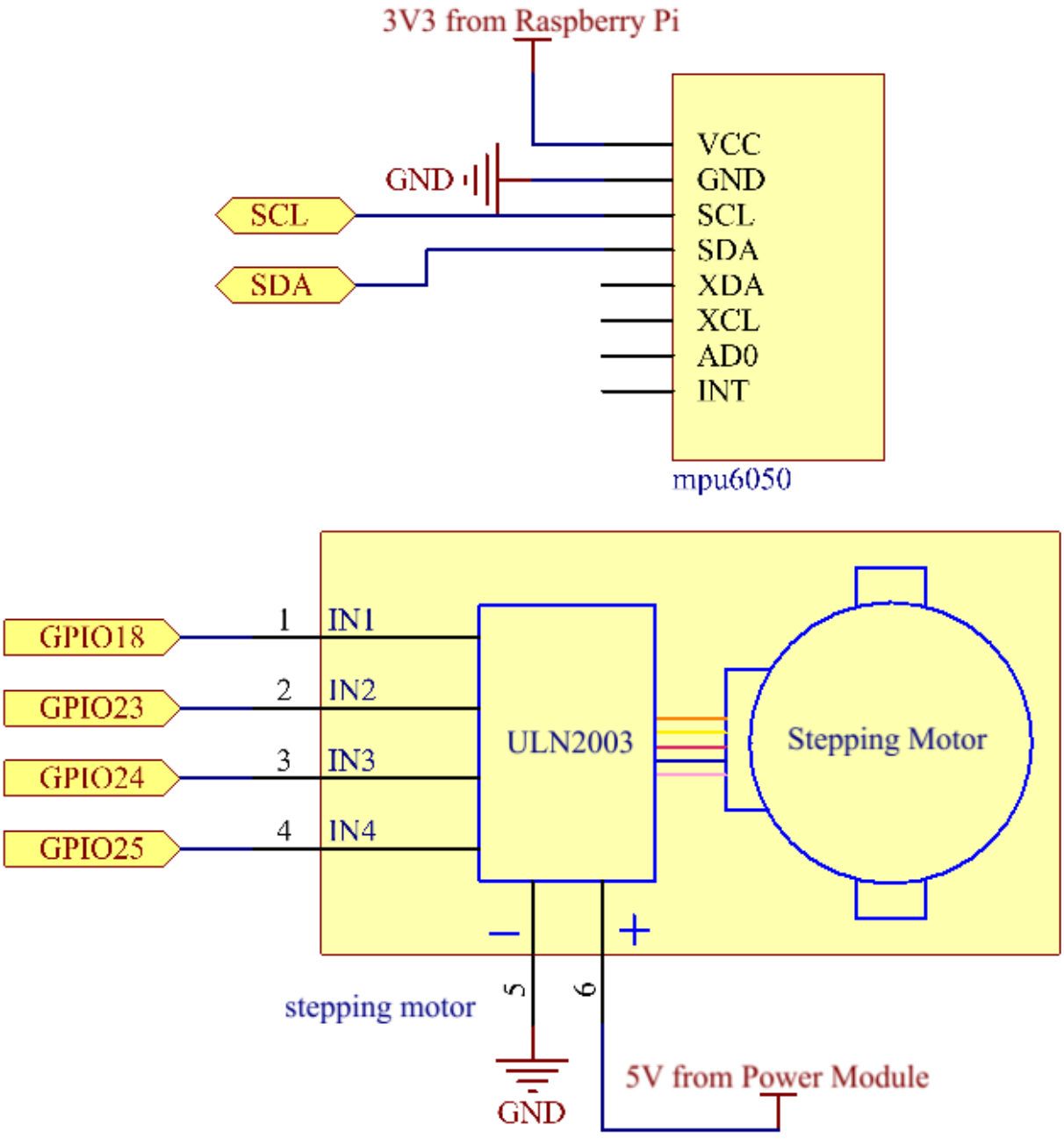
Für dieses Projekt benötigen wir die folgenden Komponenten.

<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

1 * MPU6050 Module
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * ULN2003 Module
  </td></tr>
</table>

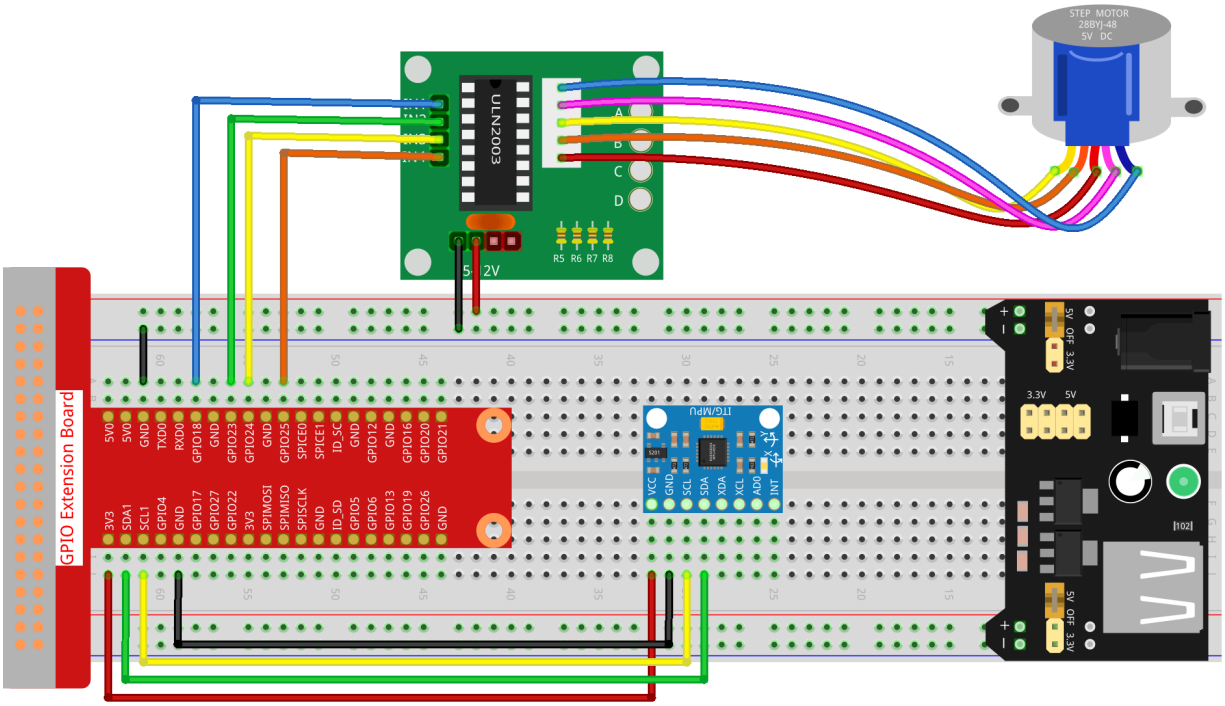
Schaltplan

T-Karte Name	physisch	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		


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

Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.6_MotionControl_zero.py
```

Während der Code ausgeführt wird, wenn der Neigungswinkel des **MPU6050** auf der Y-Achse größer als **45** ℃ ist, rotiert der Schrittmotor gegen den Uhrzeigersinn; wenn er weniger als **-45** ℃ beträgt, rotiert der Schrittmotor im Uhrzeigersinn.

Code

Bemerkung: Sie können den unten stehenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Wirkung zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice
import smbus
import math
import time

# Initialisierung der Leistungsverwaltungsregister für MPU6050
power_mgmt_1 = 0x6b
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
power_mgmt_2 = 0x6c

# Einrichten der I2C-Kommunikation mit MPU6050
bus = smbus.SMBus(1)  # SMBus initialisieren
address = 0x68        # I2C-Adresse des MPU6050
bus.write_byte_data(address, power_mgmt_1, 0)  # MPU6050 aufwecken

# Initialisieren der Motorpins zu den GPIO-Pins 18, 23, 24, 25
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]

# Setzen der Motor-Drehgeschwindigkeitsparameter
rolePerMinute = 15
stepsPerRevolution = 2048
# Berechnen der Verzögerung zwischen Schritten für die gewünschte U/min
stepSpeed = (60 / rolePerMinute) / stepsPerRevolution

# Einzelnes Byte von der angegebenen I2C-Adresse lesen
def read_byte(adr):
    return bus.read_byte_data(address, adr)

# Wort (2 Bytes) von der angegebenen I2C-Adresse lesen
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

# Wort in 2's-Komplement-Form lesen
def read_word_2c(adr):
    val = read_word(adr)
    if val >= 0x8000:
        return -((65535 - val) + 1)
    else:
        return val

# Euklidischer Abstand zwischen zwei Punkten berechnen
def dist(a, b):
    return math.sqrt((a * a) + (b * b))

# Y-Achsenrotation berechnen
def get_y_rotation(x, y, z):
    radians = math.atan2(x, dist(y, z))
    return -math.degrees(radians)

# X-Achsenrotation berechnen
def get_x_rotation(x, y, z):
    radians = math.atan2(y, dist(x, z))
    return math.degrees(radians)

# Neigungswinkel von MPU6050 abrufen
def mpu6050():
    accel_xout = read_word_2c(0x3b)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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

# Steuerung der Schrittmotor-Drehung
def rotary(direction):
    if direction == 'c':
        # Sequenz für die Drehung im Uhrzeigersinn
        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':
        # Sequenz für die Drehung gegen den Uhrzeigersinn
        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)

# Hauptschleife zum kontinuierlichen Lesen des Neigungswinkels und Steuern des Motors
try:
    while True:
        angle = mpu6050()
        if angle >= 45:
            rotary('a')  # Gegen den Uhrzeigersinn für positiven Neigungswinkel drehen
        elif angle <= -45:
            rotary('c')  # Im Uhrzeigersinn für negativen Neigungswinkel drehen
except KeyboardInterrupt:
    # Alle Motorpins bei Tastaturunterbrechung ausschalten
    for pin in motorPin:
        pin.off()
```

Code-Erklärung

1. Das Skript beginnt mit dem Importieren erforderlicher Bibliotheken. `gpiozero` wird für die Steuerung der GPIO-Pins verwendet, `smbus` für die I2C-Kommunikation, `math` für mathematische Operationen und `time` für Verzögerungen.

   ```
   #!/usr/bin/env python3
   from gpiozero import OutputDevice
   import smbus
   import math
   import time
   ```

2. Richten Sie die I2C-Kommunikation mit dem MPU6050-Sensor ein. `power_mgmt_1` und `power_mgmt_2` sind Register zur Verwaltung der Stromversorgung des Sensors. Der Sensor wird durch Schreiben in `power_mgmt_1` „aufgeweckt“.

```
# Initialisieren der Leistungsverwaltungsregister für MPU6050
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

# Richten Sie die I2C-Kommunikation mit MPU6050 ein
bus = smbus.SMBus(1)  # Initialisieren Sie SMBus
address = 0x68        # I2C-Adresse von MPU6050
bus.write_byte_data(address, power_mgmt_1, 0)  # Wecken Sie MPU6050 auf
```

3. Initialisiert die GPIO-Pins (18, 23, 24, 25) auf dem Raspberry Pi zur Steuerung des Schrittmotors. Jeder Pin ist mit einer Spule im Motor verbunden.

```
# Initialisieren Sie Motorpins für GPIO-Pins 18, 23, 24, 25
motorPin = [OutputDevice(pin) for pin in (18, 23, 24, 25)]
```

4. Legt die Umdrehungen pro Minute (RPM) des Motors und die Anzahl der Schritte pro Umdrehung fest. `stepSpeed` berechnet die Verzögerung zwischen Schritten, um die gewünschte RPM zu erreichen und einen reibungslosen Motorbetrieb sicherzustellen.

```
# Setzen Sie Parameter für die Motordrehzahl
rolePerMinute = 15
stepsPerRevolution = 2048
# Berechnen Sie die Verzögerung zwischen den Schritten für die gewünschte RPM
stepSpeed = (60 / rolePerMinute) / stepsPerRevolution
```

5. Diese Funktionen werden für die I2C-Kommunikation verwendet. `read_byte` liest ein einzelnes Byte von einer bestimmten Adresse, während `read_word` zwei Bytes (ein Wort) von der Adresse liest und sie mit bitweisen Operationen (<< und +) zu einem einzelnen Wert kombiniert.

```
# Lesen Sie ein einzelnes Byte von der angegebenen I2C-Adresse
def read_byte(adr):
    return bus.read_byte_data(address, adr)

# Lesen Sie ein Wort (2 Bytes) von der angegebenen I2C-Adresse
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. Diese Funktion konvertiert das gelesene Wort in eine 2's-Komplement-Form, die nützlich ist, um Vorzeichenwerte aus Sensordaten zu interpretieren. Diese Umwandlung ist notwendig, um negative Sensormessungen zu verarbeiten.

```
# Lesen eines Wortes in 2's-Komplement-Form
def read_word_2c(adr):
    val = read_word(adr)
    if val >= 0x8000:
        return -((65535 - val) + 1)
    else:
        return val
```

7. `dist` berechnet die euklidische Entfernung zwischen zwei Punkten, die in den Rotationsberechnungen verwendet wird. `get_y_rotation` und `get_x_rotation` berechnen die Rotationswinkel entlang der Y- bzw. X-Achse, indem sie die `atan2`-Funktion aus der `math`-Bibliothek verwenden und das Ergebnis in Grad umrechnen.

```
# Berechnen der euklidischen Entfernung zwischen zwei Punkten
def dist(a, b):
    return math.sqrt((a * a) + (b * b))

# Berechnen der Y-Achsenrotation
def get_y_rotation(x, y, z):
    radians = math.atan2(x, dist(y, z))
    return -math.degrees(radians)

# Berechnen der X-Achsenrotation
def get_x_rotation(x, y, z):
    radians = math.atan2(y, dist(x, z))
    return math.degrees(radians)
```

8. Diese Funktion liest die Beschleunigungsdaten vom MPU6050-Sensor, skaliert die Messungen und berechnet den Neigungswinkel mithilfe der Funktion `get_y_rotation`. Die Funktion `read_word_2c` liest Sensordaten in 2's-Komplement-Form, um negative Werte zu verarbeiten.

```
# Holen Sie sich den Neigungswinkel von MPU6050
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. Die Funktion `rotary` steuert die Rotation des Schrittmotors. Sie führt eine Schrittsequenz für die Uhrzeiger- oder gegen den Uhrzeigersinn-Drehung basierend auf dem `direction`-Parameter aus. Die Sequenz beinhaltet das Ein- oder Ausschalten bestimmter Motorpins in einem Muster.

```
# Steuern Sie die Rotation des Schrittmotors
def rotary(direction):
    if direction == 'c':
        # Sequenz für die Uhrzeigersinn-Drehung
        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':
        # Sequenz für die gegen den Uhrzeigersinn-Drehung
        for j in range(4):
            for i in range(4):
                if 0x99 << j & (0x08 >> i):
                    motorPin[i].on()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                else:
                    motorPin[i].off()
                time.sleep(stepSpeed)
```

10. Die Hauptschleife liest kontinuierlich den Neigungswinkel aus dem MPU6050-Sensor und steuert die Rotationsrichtung des Motors basierend auf dem Winkel. Wenn das Programm unterbrochen wird (z. B. durch eine Tastaturunterbrechung), schaltet es alle Motorpins aus, um die Sicherheit zu gewährleisten.

```
# Hauptschleife zum kontinuierlichen Lesen des Neigungswinkels und zur Steuerung␣
↪des Motors
try:
    while True:
        angle = mpu6050()
        if angle >= 45:
            rotary('a')  # Gegen den Uhrzeigersinn für positiven Neigungswinkel␣
↪drehen
        elif angle <= -45:
            rotary('c')  # Im Uhrzeigersinn für negativen Neigungswinkel drehen
except KeyboardInterrupt:
    # Schalten Sie bei Tastaturunterbrechung alle Motorpins aus
    for pin in motorPin:
        pin.off()
```

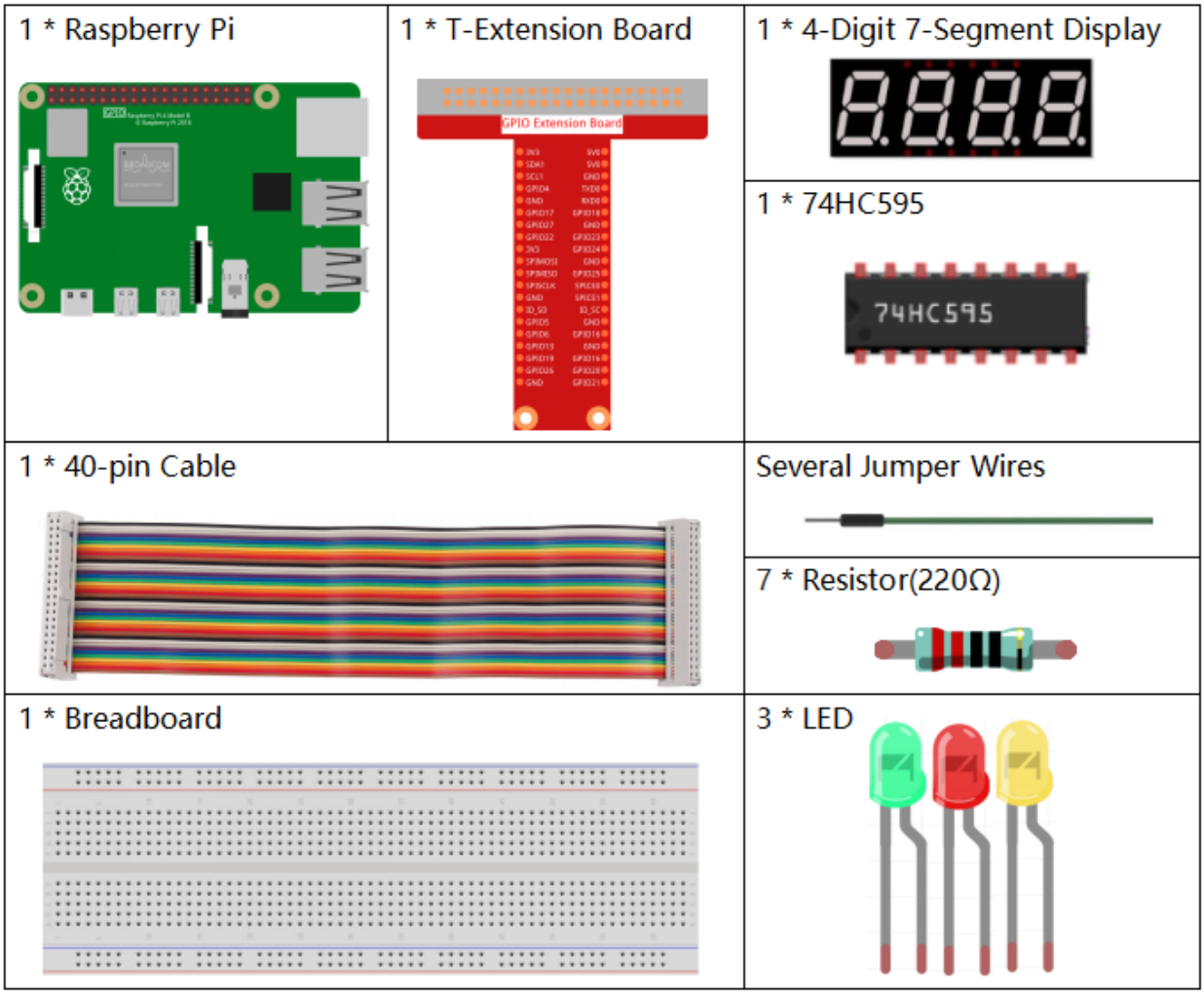
3.1.7 Verkehrsampel

Einführung

In diesem Projekt werden wir LED-Lichter in drei Farben verwenden, um die Änderung der Verkehrsampel zu realisieren, und eine vierstellige 7-Segment-Anzeige wird verwendet, um die Zeitdauer jedes Verkehrszustands anzuzeigen.

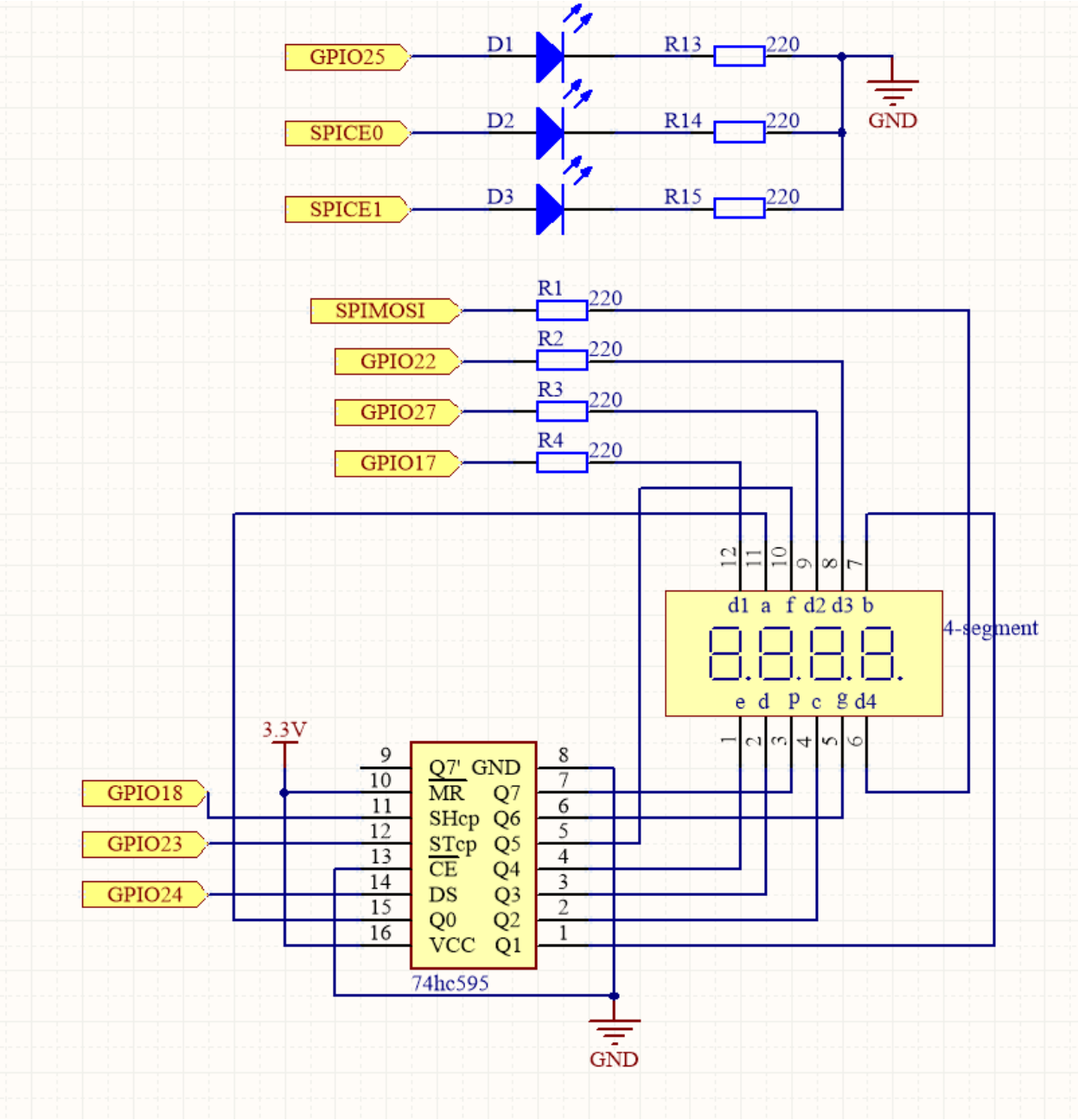
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.



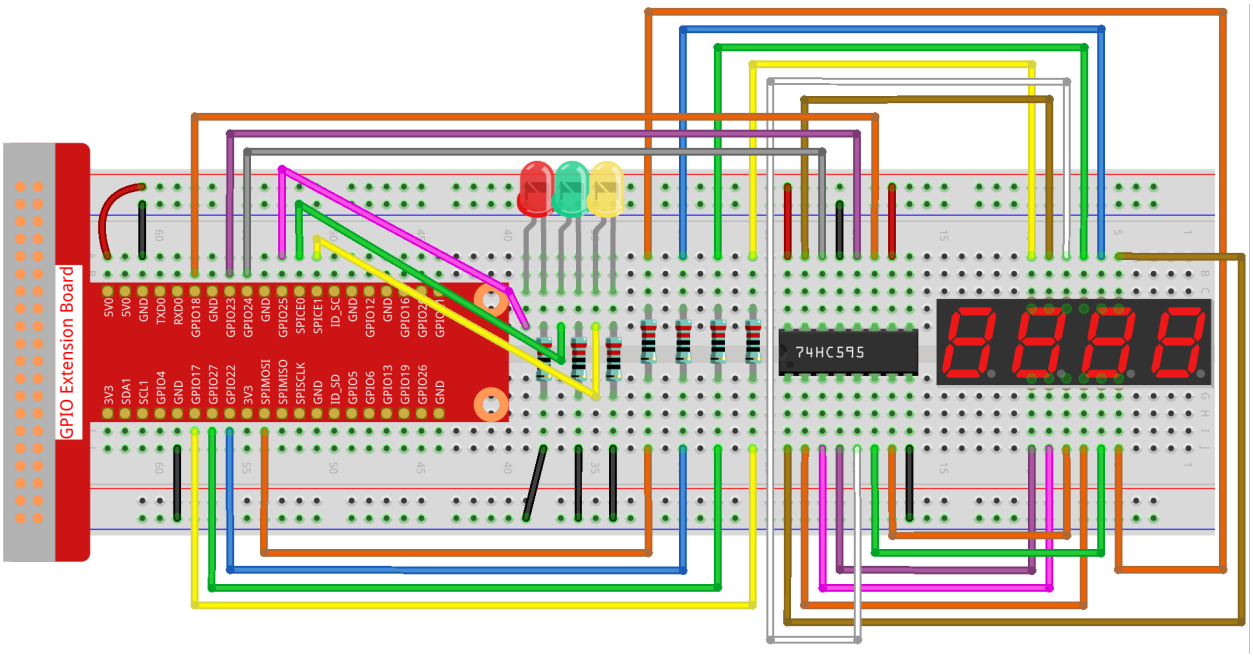
Schaltplan

T-Board-Name	physisch	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



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung auf.



Schritt 2: Wechseln Sie das Verzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.7_TrafficLight_zero.py
```

Während der Code läuft, simulieren LEDs die Farbänderung von Verkehrsampeln. Zuerst leuchtet die rote LED für 60 Sekunden, dann leuchtet die grüne LED für 30 Sekunden; anschließend leuchtet die gelbe LED für 5 Sekunden. Danach leuchtet die rote LED wieder 60 Sekunden lang. Auf diese Weise wird diese Serie von Aktionen wiederholt ausgeführt. Gleichzeitig zeigt die 4-stellige 7-Segment-Anzeige kontinuierlich die Countdown-Zeit an.

Code

Bemerkung: Sie können den unten stehenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie das tun, müssen Sie zum Quellcodepfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Wirkung zu sehen.

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

from gpiozero import OutputDevice, LED
import threading

# GPIO-Pins für 74HC595-Schieberegister einrichten
SDI = OutputDevice(24)   # Serielle Dateneingabe
RCLK = OutputDevice(23)  # Register Clock
SRCLK = OutputDevice(18) # Shift Register Clock

# GPIO-Pins für die Auswahl der Ziffern auf der 7-Segment-Anzeige einrichten
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Segmentcodes für die Zahlen 0-9 auf der 7-Segment-Anzeige
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# GPIO-Pins für die LEDs der Verkehrsampel einrichten
ledPinR = LED(25) # Rote LED
ledPinG = LED(8)  # Grüne LED
ledPinY = LED(7)  # Gelbe LED

# Dauer-Einstellungen für die Verkehrsampel
greenLight = 30
yellowLight = 5
redLight = 60

# Namen der Verkehrsampelfarben
lightColor = ("Rot", "Grün", "Gelb")

# Initialisierung der Statusvariablen
colorState = 0
counter = 60
timer1 = None

def setup():
    """ Initialisieren Sie das Verkehrsampelsystem und starten Sie den Timer. """
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()

def clearDisplay():
    """ Löschen Sie die 7-Segment-Anzeige. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ Verschieben Sie Daten in das 74HC595-Schieberegister zur Anzeige der Ziffer. """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    """ Wählen Sie eine bestimmte Ziffer zur Anzeige auf der 7-Segment-Anzeige aus. """
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
def timer():
    """ Behandeln Sie die Zeitgestaltung für die Änderung der Verkehrsampel. """
    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"Zähler : {counter}    Farbe: {lightColor[colorState]}")

def lightup():
    """ Aktualisieren Sie die LED der Verkehrsampel basierend auf dem aktuellen Zustand.
↪"""
    global colorState
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
    [ledPinR, ledPinG, ledPinY][colorState].on()

def display():
    """ Zeigen Sie den aktuellen Zählerwert auf der 7-Segment-Anzeige an. """
    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():
    """ Hauptloop zur kontinuierlichen Aktualisierung der Anzeige und der LEDs der␣
↪Verkehrsampel. """
    while True:
        display()
        lightup()

def destroy():
    """ Räumen Sie die Ressourcen auf, wenn das Skript beendet wird, wie das Ausschalten␣
↪der LEDs und das Stoppen des Timer-Threads. """
    global timer1
    timer1.cancel()
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()

try:
    setup()
    loop()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

Code Explanation

1. Importiert die Klassen `OutputDevice` und `LED` aus der gpiozero-Bibliothek, um die Steuerung von allgemeinen Ausgabegeräten und insbesondere LEDs zu ermöglichen. Importiert das Python-Modul threading, das für das Erstellen und Verwalten von Threads zur gleichzeitigen Ausführung verwendet wird.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, LED
import threading
```

2. Initialisiert die GPIO-Pins, die mit dem Schieberegister für die serielle Dateneingabe (SDI), die Registeruhr (RCLK) und die Schieberegisteruhr (SRCLK) verbunden sind.

```
# Einrichtung der GPIO-Pins für das 74HC595 Schieberegister
SDI = OutputDevice(24)   # Serielle Dateneingabe
RCLK = OutputDevice(23)  # Registeruhr
SRCLK = OutputDevice(18) # Schieberegisteruhr
```

3. Initialisiert die Pins für jede Ziffer der 7-Segment-Anzeige und definiert die binären Codes für die Anzeige der Zahlen 0-9.

```
# Einrichtung der GPIO-Pins für die Auswahl der Ziffern auf der 7-Segment-Anzeige
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Segmentcodes für die Zahlen 0-9 auf der 7-Segment-Anzeige
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

4. Initialisiert GPIO-Pins für die roten, grünen und gelben LEDs, die in der Verkehrsampelsimulation verwendet werden. Setzt die Dauer (in Sekunden) für jeden Farbzustand in der Verkehrsampel fest. Definiert die Namen der Verkehrsampelfarben zur Referenz.

```
# Einrichtung der GPIO-Pins für die LEDs der Verkehrsampel
ledPinR = LED(25) # Rote LED
ledPinG = LED(8)  # Grüne LED
ledPinY = LED(7)  # Gelbe LED

# Dauereinstellungen für die Verkehrsampel
greenLight = 30
yellowLight = 5
redLight = 60

# Namen der Verkehrsampelfarben
lightColor = ("Rot", "Grün", "Gelb")
```

5. Initialisiert Variablen zur Verfolgung des aktuellen Farbzustands, eines Zählers für die Zeitmessung und einen Platzhalter für ein Timer-Objekt.

```
# Initialisierung der Statusvariablen
colorState = 0
counter = 60
timer1 = None
```

6. Initialisiert das Verkehrsampelsystem und startet den Timer-Thread.

```
def setup():
    """ Initialisieren Sie das Verkehrsampelsystem und starten Sie den Timer. """
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
```

7. Funktionen zur Steuerung der 7-Segment-Anzeige. `clearDisplay` schaltet alle Segmente aus, `hc595_shift` verschiebt Daten in das Schieberegister und `pickDigit` aktiviert eine bestimmte Ziffer auf der Anzeige.

```
def clearDisplay():
    """ Löschen Sie die 7-Segment-Anzeige. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ Verschieben Sie Daten in das 74HC595 Schieberegister zur Anzeige der Ziffer.
    """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    """ Wählen Sie eine bestimmte Ziffer zur Anzeige auf der 7-Segment-Anzeige aus.
    """
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

8. Verwaltet die Zeitsteuerung für die Änderungen der Verkehrsampel und aktualisiert den Zähler und den Farbzustand.

```
def timer():
    """ Behandeln Sie die Zeitsteuerung für die Änderungen der Verkehrsampel. """
    global counter, colorState, timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter -= 1
    if counter == 0:
        counter = [greenLight, yellowLight, redLight][colorState]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        colorState = (colorState + 1) % 3
    print(f"counter : {counter}    color: {lightColor[colorState]}")
```

9. Aktualisiert den Zustand der Verkehrsampel-LEDs basierend auf dem aktuellen Farbzustand.

```
def lightup():
    """ Aktualisieren Sie die LED der Verkehrsampel basierend auf dem aktuellen␣
↪Zustand. """
    global colorState
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
    [ledPinR, ledPinG, ledPinY][colorState].on()
```

10. Berechnet die Ziffer, die auf jedem Segment der 7-Segment-Anzeige angezeigt werden soll, und aktualisiert sie entsprechend.

```
def display():
    """ Zeigen Sie den aktuellen Zählerwert auf der 7-Segment-Anzeige an. """
    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])
```

11. Die Hauptschleife, die die Anzeige und die LEDs der Verkehrsampel kontinuierlich aktualisiert.

```
def loop():
    """ Hauptloop zur kontinuierlichen Aktualisierung der Anzeige und der LEDs der␣
↪Verkehrsampel. """
    while True:
        display()
        lightup()
```

12. Räumt Ressourcen auf, wenn das Skript beendet wird, wie das Ausschalten der LEDs und das Stoppen des Timer-Threads.

```
def destroy():
    """ Räumen Sie die Ressourcen auf, wenn das Skript beendet wird, wie das␣
↪Ausschalten der LEDs und das Stoppen des Timer-Threads. """
    global timer1
    timer1.cancel()
    ledPinR.off()
    ledPinG.off()
    ledPinY.off()
```

3.1.8 Überhitzungsüberwachung

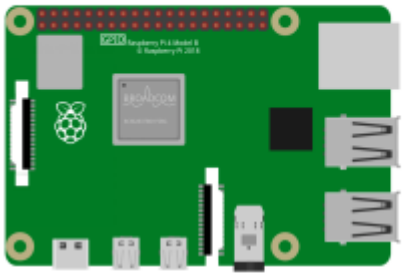
Einleitung

Sie möchten ein Überhwärmeüberwachungsgerät entwickeln, das in verschiedenen Situationen angewendet werden kann, z. B. in einer Fabrik, wenn ein Alarmsignal ausgelöst und die Maschine rechtzeitig ausgeschaltet werden soll, wenn es zu einer Überhitzung des Stromkreises kommt. In diesem Projekt verwenden wir einen Thermistor, einen Joystick, einen Summer, eine LED und ein LCD, um ein intelligentes Temperaturüberwachungsgerät zu erstellen, dessen Schwellenwert einstellbar ist.

Benötigte Komponenten

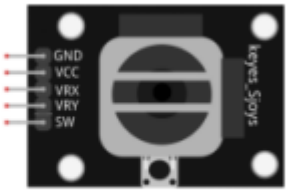
Für dieses Projekt benötigen wir die folgenden Komponenten.

1 * Raspberry Pi



1 * T-Extension Board

1 * Joystick



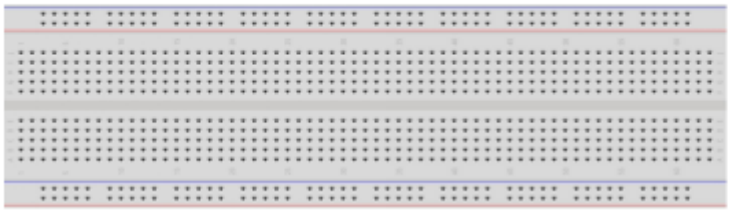
1 * ADC0834

1 * 40-pin Cable

1 * LED

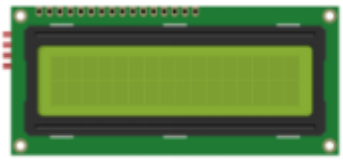
1* S8050 NPN Transistor

1 * Breadboard



Several Jumper Wires

1 * I2C LCD1602



1 * Active Buzzer

1*Thermistor

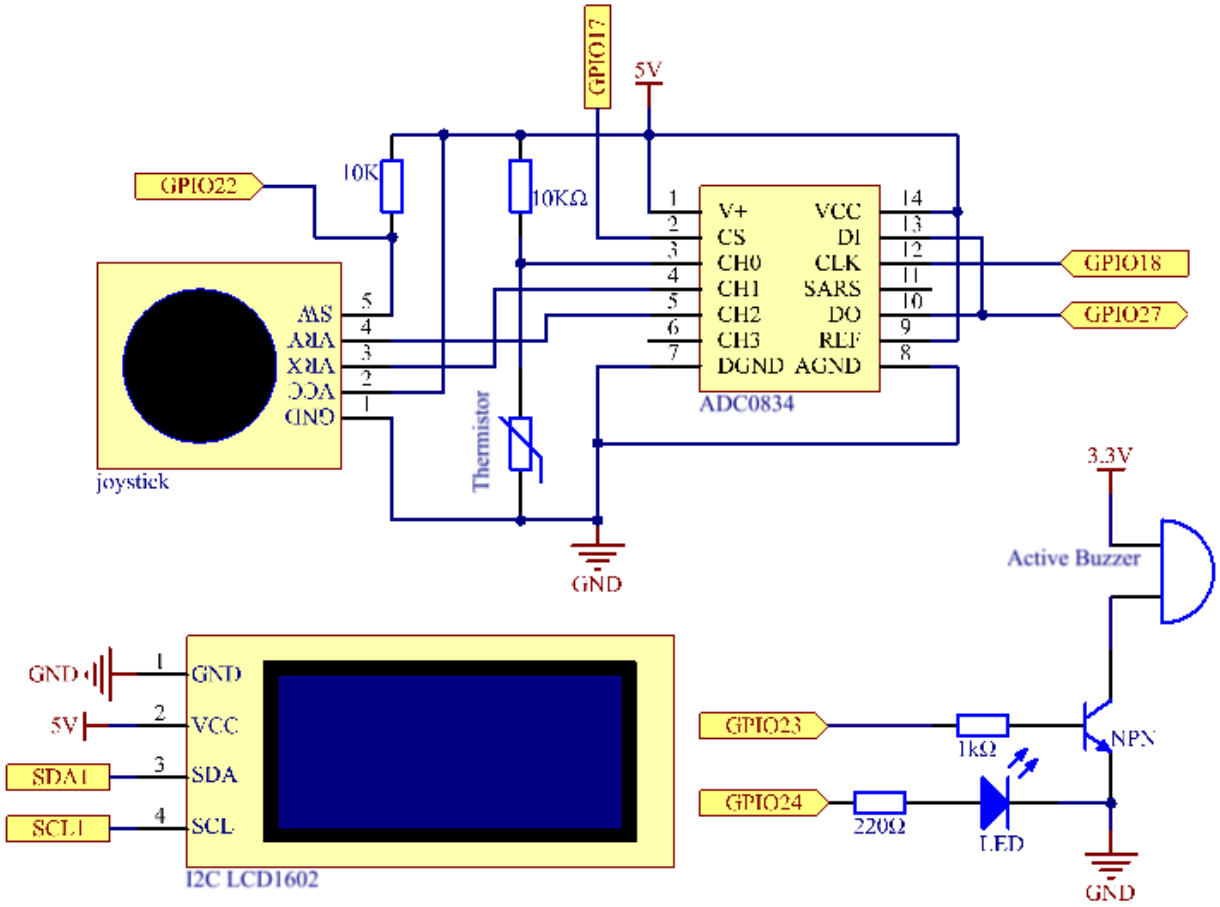
1 * Resistor 220Ω

1 * Resistor 1kΩ

1 * Resistor 10kΩ

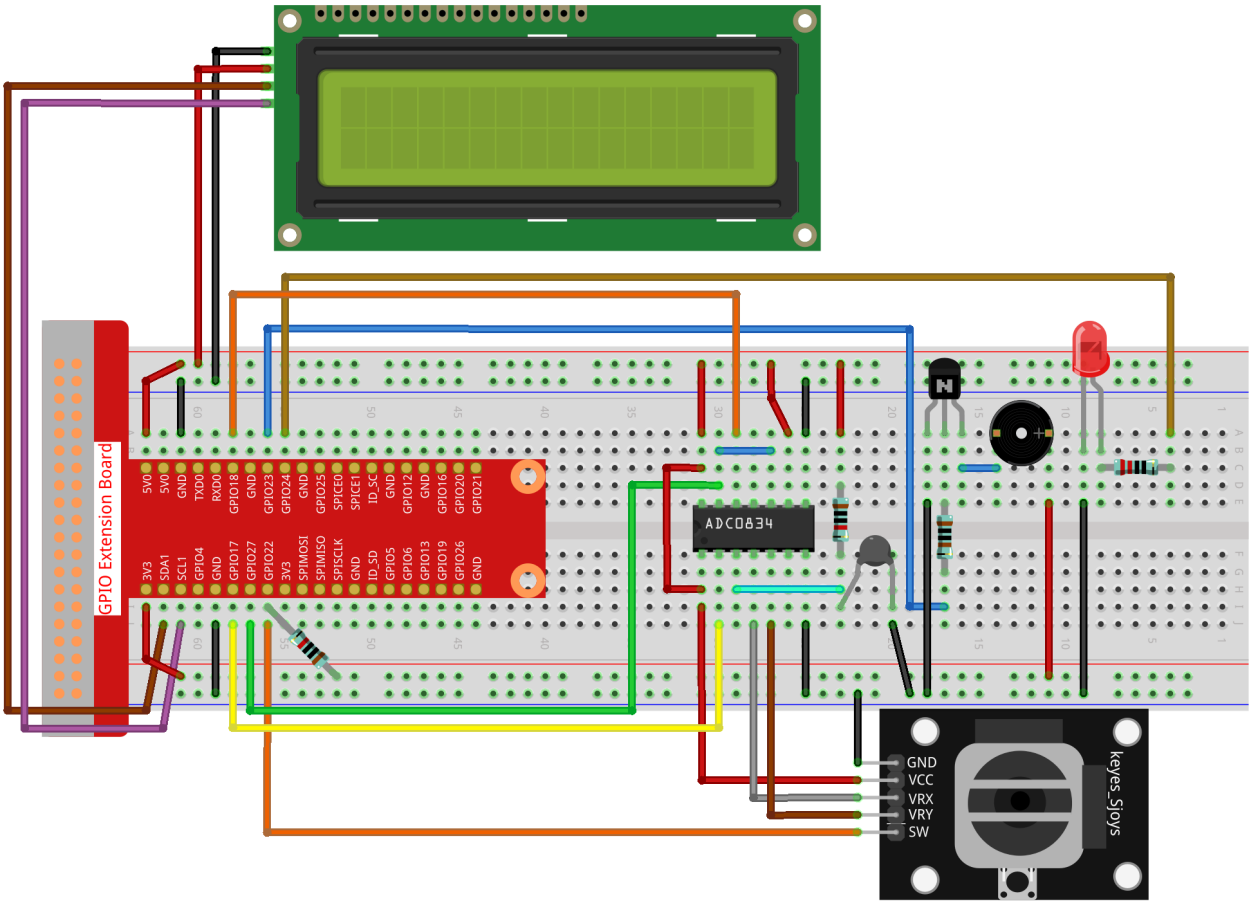
Schaltplan

T-Karte Name	physisch	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		



Versuchsdurchführung

Schritt 1: Bauen Sie die Schaltung.



Schritt 2: Gehen Sie zum Ordner mit dem Code.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.8_OverheatMonitor_zero.py
```

Während der Code ausgeführt wird, werden die aktuelle Temperatur und die Hochtemperaturschwelle von **40** auf einem **I2C LCD1602** angezeigt. Wenn die aktuelle Temperatur höher ist als die Schwelle, wird der Summer und die LED aktiviert, um Sie zu alarmieren.

Der **Joystick** dient dazu, die Hochtemperaturschwelle anzupassen. Das Drehen des **Joystick** in Richtung der X- und Y-Achse kann die aktuelle Hochtemperaturschwelle erhöhen oder verringern. Drücken Sie den **Joystick** erneut, um die Schwelle auf den Ausgangswert zurückzusetzen.

Bemerkung:

- Wenn der Fehler `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'` auftritt, müssen Sie *I2C-Konfiguration* aufrufen, um I2C zu aktivieren.

- Wenn der Fehler `ModuleNotFoundError: No module named 'smbus2'` auftritt, führen Sie `sudo pip3 install smbus2` aus.
- Wenn der Fehler `OSError: [Errno 121] Remote I/O error` auftritt, bedeutet dies, dass das Modul falsch angeschlossen ist oder defekt ist.
- Wenn der Code und die Verkabelung in Ordnung sind, das LCD jedoch keinen Inhalt anzeigt, können Sie den Potenziometer auf der Rückseite drehen, um den Kontrast zu erhöhen.

Code

Bemerkung: Sie können den folgenden Code **ändern/zurücksetzen/kopieren/ausführen/anhalten**. Davor müssen Sie jedoch zum Quellcodepfad wie `raphael-kit/python` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Wirkung zu sehen.

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

import LCD1602
from gpiozero import LED, Buzzer, Button
import ADC0834
import time
import math

# Initialisieren Sie Joystick-Taste, Summer und LED
Joy_BtnPin = Button(22)
buzzPin = Buzzer(23)
ledPin = LED(24)

# Setzen Sie die anfängliche obere Temperaturschwelle
upperTem = 40

# Setup ADC- und LCD-Module
ADC0834.setup()
LCD1602.init(0x27, 1)

def get_joystick_value():
    """
    Liest die Joystick-Werte und gibt einen Änderungswert basierend auf der Position des␣
↪Joysticks zurück.
    """
    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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
def upper_tem_setting():
    """
    Passt die obere Temperaturschwelle an und zeigt sie auf dem LCD an.
    """
    global upperTem
    LCD1602.write(0, 0, 'Oben anpassen: ')
    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():
    """
    Liest die aktuelle Temperatur vom Sensor und gibt sie in Grad Celsius zurück.
    """
    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():
    """
    Überwacht und zeigt die aktuelle Temperatur und die obere Temperaturschwelle an.␣
↪Aktiviert den Summer und die LED, wenn die Temperatur den oberen Grenzwert␣
↪überschreitet.
    """
    global upperTem
    Cel = temperature()
    LCD1602.write(0, 0, 'Temp: ')
    LCD1602.write(0, 1, 'Oben: ')
    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()

# Hauptausführungsschleife
try:
    lastState = 1
    stage = 0
    while True:
        currentState = Joy_BtnPin.value
        # Wechseln Sie zwischen Einstell- und Überwachungsmodus
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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:
    # Bereinigen und beenden
    LCD1602.clear()
    ADC0834.destroy()
```

Code-Erklärung

1. In diesem Abschnitt werden die erforderlichen Bibliotheken für das Projekt importiert. `LCD1602` ist für das LCD-Display, `gpiozero` bietet Klassen für LED, Summer und Taste, `ADC0834` ist für die Analog-Digital-Umwandlung und `time` und `math` sind Python-Standardbibliotheken für zeitbezogene Funktionen bzw. mathematische Operationen.

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

   import LCD1602
   from gpiozero import LED, Buzzer, Button
   import ADC0834
   import time
   import math
   ```

2. Hier werden die Joystick-Taste, der Summer und die LED initialisiert. `Button(22)` erstellt ein Tastenobjekt, das mit dem GPIO-Pin 22 verbunden ist. `Buzzer(23)` und `LED(24)` initialisieren den Summer bzw. die LED an den GPIO-Pins 23 bzw. 24.

   ```
   # Initialisieren Sie Joystick-Taste, Summer und LED
   Joy_BtnPin = Button(22)
   buzzPin = Buzzer(23)
   ledPin = LED(24)
   ```

3. Legt die anfängliche obere Temperaturschwelle fest und initialisiert die ADC- und LCD-Module. Das LCD wird mit einer Adresse (`0x27`) und einem Modus (`1`) initialisiert.

   ```
   # Setzen Sie die anfängliche obere Temperaturschwelle
   upperTem = 40

   # Setup ADC- und LCD-Module
   ADC0834.setup()
   LCD1602.init(0x27, 1)
   ```

4. Diese Funktion liest die X- und Y-Werte des Joysticks mithilfe von ADC0834 aus. Sie gibt einen Änderungswert basierend auf der Position des Joysticks zurück, der zur Anpassung der Temperaturschwelle verwendet wird.

   ```
   def get_joystick_value():
       """
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    Liest die Joystick-Werte und gibt einen Änderungswert basierend auf der␣
↪Position des Joysticks zurück.
    """
    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
```

5. Passt die obere Temperaturschwelle mithilfe der Joystick-Eingabe an. Die neue Schwelle wird auf dem LCD angezeigt.

```
def upper_tem_setting():
    """
    Passt die obere Temperaturschwelle an und zeigt sie auf dem LCD an.
    """
    global upperTem
    LCD1602.write(0, 0, 'Oben anpassen: ')
    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. Liest die aktuelle Temperatur mithilfe des Sensors mit ADC0834 und gibt sie in Grad Celsius zurück.

```
def temperature():
    """
    Liest die aktuelle Temperatur vom Sensor und gibt sie in Grad Celsius zurück.
    """
    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. Überwacht und zeigt die aktuelle Temperatur und die obere Schwelle an. Wenn die Temperatur die obere Schwelle überschreitet, werden der Summer und die LED aktiviert.

```
def monitoring_temp():
    """
    Überwacht und zeigt die aktuelle Temperatur und die obere Temperaturschwelle an.
    Aktiviert den Summer und die LED, wenn die Temperatur den oberen Grenzwert␣
↪überschreitet.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    global upperTem
    Cel = temperature()
    LCD1602.write(0, 0, 'Temp: ')
    LCD1602.write(0, 1, 'Oben: ')
    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. Die Hauptausführungsschleife wechselt basierend auf den Tastendrücken der Joystick-Taste zwischen Einstell- und Überwachungsmodus. Sie aktualisiert kontinuierlich entweder die Temperatureinstellung oder überwacht die aktuelle Temperatur.

```
# Hauptausführungsschleife
try:
    lastState = 1
    stage = 0
    while True:
        currentState = Joy_BtnPin.value
        # Wechseln Sie zwischen Einstell- und Überwachungsmodus
        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:
    # Bereinigen und beenden
    LCD1602.clear()
    ADC0834.destroy()
```

3.1.9 Passwortsperre

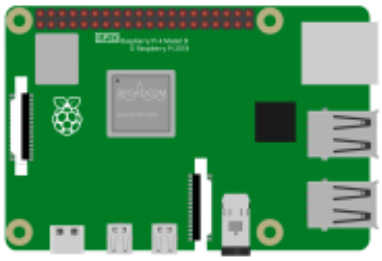
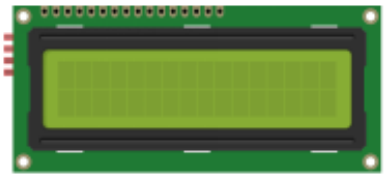
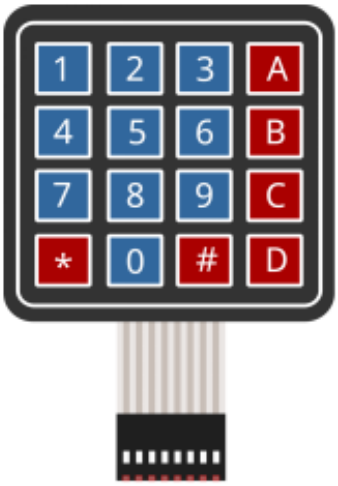
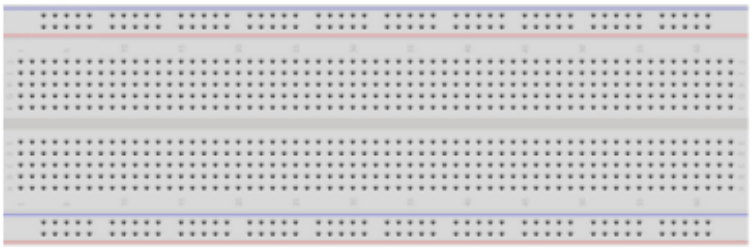
Einführung

In diesem Projekt verwenden wir eine Tastatur und ein LCD, um ein Kombinationsschloss zu erstellen. Das LCD zeigt eine entsprechende Aufforderung an, um Ihr Passwort auf der Tastatur einzugeben. Wenn das Passwort korrekt eingegeben wird, wird „Richtig“ angezeigt.

Auf der Grundlage dieses Projekts können wir zusätzliche elektronische Komponenten wie einen Summer, eine LED usw. hinzufügen, um verschiedene experimentelle Phänomene für die Passworteingabe zu realisieren.

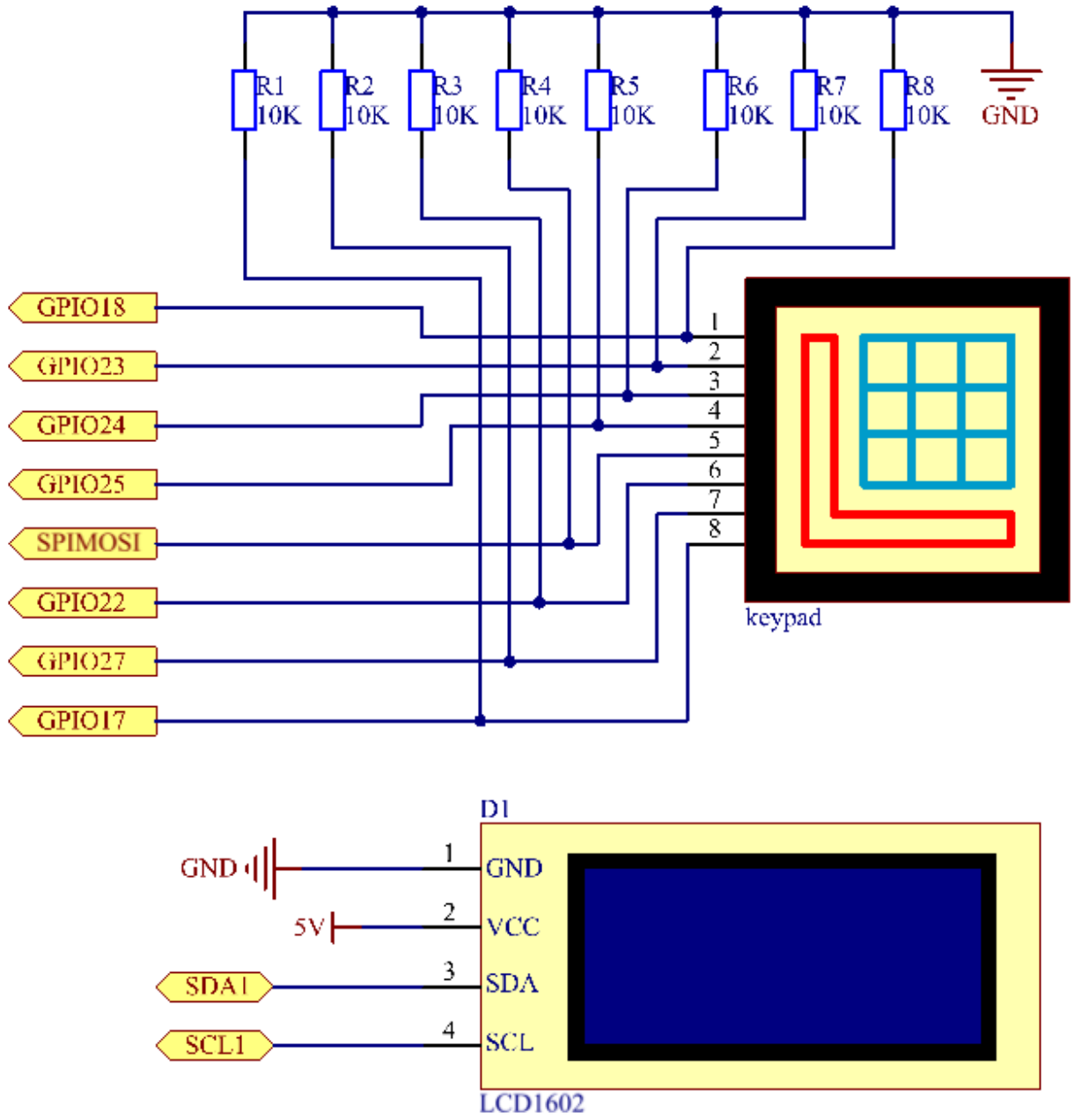
Benötigte Komponenten

Für dieses Projekt benötigen wir die folgenden Komponenten.

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
 </td>
<td>1 * I2C LCD1602
  </td>
</tr>
<tr>
<td rowspan="2">1 * 40-pin Cable
 </td>
<td rowspan="2">1 * Keypad
  </td>
<td>Several Jumper Wires
 </td>
</tr>
<tr>
<td>1 * Breadboard
  </td>
<td>8 * Resistor 10KΩ
 </td>
</tr>
</table>

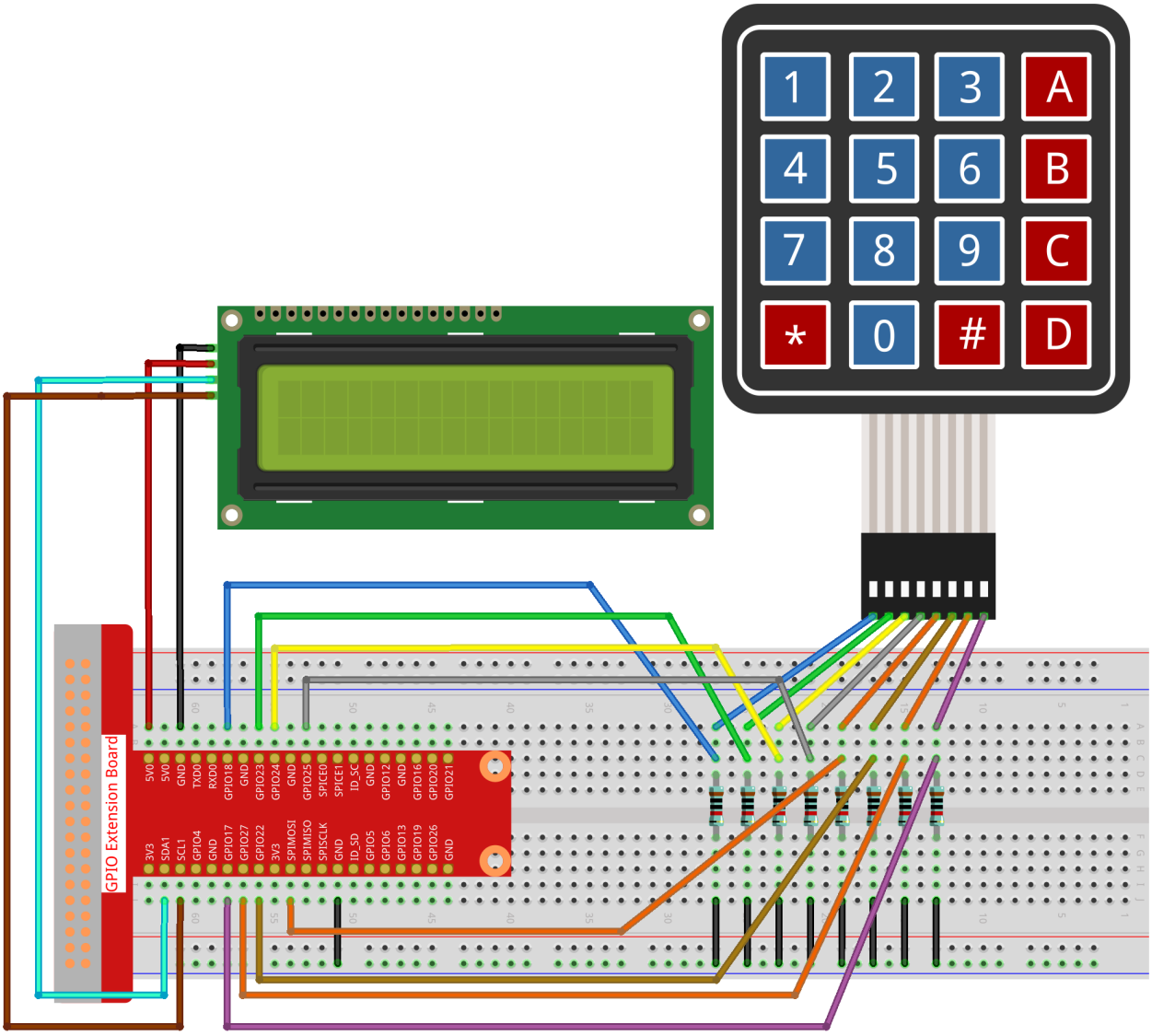
Schaltplan

T-Karte Name	physisch	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		



Experimentelle Vorgehensweise

Schritt 1: Schaltung aufbauen.



Schritt 2: Verzeichnis ändern.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.9_PasswordLock_zero.py
```

Nachdem der Code ausgeführt wurde, wird die Tastatur verwendet, um das Passwort einzugeben: 1984. Wenn auf dem LCD1602 „Richtig“ angezeigt wird, ist das Passwort korrekt eingegeben. Andernfalls wird „FALSCHE TASTE“ angezeigt.

Bemerkung:

- Wenn der Fehler `Datei nicht gefunden: [Errno 2] Keine solche Datei oder Verzeichnis: '/dev/i2c-1'` auftritt, müssen Sie sich auf *I2C-Konfiguration* beziehen, um I2C zu aktivieren.
- Wenn der Fehler `Modul nicht gefunden: Kein Modul mit dem Namen 'smbus2'` auftritt, führen Sie bitte `sudo pip3 install smbus2` aus.

- Wenn der Fehler `OSError: [Errno 121] Remote I/O-Fehler` auftritt, bedeutet dies, dass das Modul falsch angeschlossen ist oder das Modul defekt ist.
- Wenn der Code und die Verkabelung korrekt sind, das LCD jedoch immer noch keinen Inhalt anzeigt, können Sie den Potentiometer auf der Rückseite drehen, um den Kontrast zu erhöhen.

Code

Bemerkung: Sie können den unten stehenden Code **Ändern/Zurücksetzen/Kopieren/Ausführen/Stoppen**. Bevor Sie dies tun, müssen Sie zum Quellcodepfad wie `raphael-kit/python-pi5` gehen. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um die Auswirkungen zu sehen.

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

from gpiozero import DigitalOutputDevice, Button
from time import sleep
import LCD1602

class Tastatur:
    def __init__(self, reihen_pins, spalten_pins, tasten):
        """
        Initialisieren der Tastatur mit festgelegten Reihen- und Spaltenpins und Tasten.
        :param reihen_pins: Liste der GPIO-Pins für die Reihen.
        :param spalten_pins: Liste der GPIO-Pins für die Spalten.
        :param tasten: Liste der Tasten in der Tastaturanordnung.
        """
        self.reihen = [DigitalOutputDevice(pin) for pin in reihen_pins]  # Einrichtung␣
↪der Reihenpins
        self.spalten = [Button(pin, pull_up=False) for pin in spalten_pins]  #␣
↪Einrichtung der Spaltenpins
        self.tasten = tasten  # Tastaturanordnung

    def lesen(self):
        """
        Lesen und Zurückgeben einer Liste der aktuell gedrückten Tasten.
        :return: Liste der gedrückten Tasten.
        """
        gedrueckte_tasten = []
        for i, reihe in enumerate(self.reihen):
            reihe.on()  # Aktiviere aktuelle Reihe
            for j, spalte in enumerate(self.spalten):
                if spalte.is_pressed:
                    index = i * len(self.spalten) + j
                    gedrueckte_tasten.append(self.tasten[index])
            reihe.off()  # Deaktiviere Reihe nach Überprüfung
        return gedrueckte_tasten

# Einrichtung der Passwortüberprüfung
LAENGE = 4
passwort = ['1', '9', '8', '4']  # Voreingestelltes Passwort
testwort = ['0', '0', '0', '0']  # Speicherung der Benutzereingabe
tasteIndex = 0  # Index für Eingabetasten
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
def ueberpruefen():
    """
    Überprüfen, ob das eingegebene Passwort mit dem voreingestellten Passwort␣
↪übereinstimmt.
    :return: 1 bei Übereinstimmung, 0 sonst.
    """
    for i in range(LAENGE):
        if passwort[i] != testwort[i]:
            return 0
    return 1

def einrichten():
    """
    Einrichten der Tastatur und des LCD-Displays.
    """
    global tastatur, letzte_gedrueckte_taste
    # Pin-Konfiguration für die Tastatur
    reihen_pins = [18, 23, 24, 25]
    spalten_pins = [10, 22, 27, 17]
    tasten = ["1", "2", "3", "A",
              "4", "5", "6", "B",
              "7", "8", "9", "C",
              "*", "0", "#", "D"]

    # Tastatur und LCD initialisieren
    tastatur = Tastatur(reihen_pins, spalten_pins, tasten)
    letzte_gedrueckte_taste = []
    LCD1602.init(0x27, 1)  # LCD initialisieren
    LCD1602.clear()
    LCD1602.write(0, 0, 'WILLKOMMEN!')
    LCD1602.write(2, 1, 'Geben Sie das Passwort ein')
    sleep(2)

def schleife():
    """
    Hauptschleife zur Handhabung der Tastatureingabe und Passwortüberprüfung.
    """
    global tasteIndex, LAENGE, tastatur, letzte_gedrueckte_taste
    while True:
        gedrueckte_tasten = tastatur.lesen()
        if gedrueckte_tasten and gedrueckte_tasten != letzte_gedrueckte_taste:
            if tasteIndex < LAENGE:
                LCD1602.clear()
                LCD1602.write(0, 0, "Geben Sie das Passwort ein:")
                LCD1602.write(15 - tasteIndex, 1, gedrueckte_tasten[0])
                testwort[tasteIndex] = gedrueckte_tasten[0]
                tasteIndex += 1

            if tasteIndex == LAENGE:
                if ueberpruefen() == 0:
                    LCD1602.clear()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                    LCD1602.write(3, 0, "FALSCHE TASTE!")
                    LCD1602.write(0, 1, "Bitte erneut versuchen")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "RICHTIG!")
                    LCD1602.write(2, 1, "Willkommen zurück")
                tasteIndex = 0  # Tastenindex nach Überprüfung zurücksetzen

        letzte_gedrueckte_taste = gedrueckte_tasten
        sleep(0.1)

try:
    einrichten()
    schleife()
except KeyboardInterrupt:
    LCD1602.clear()  # LCD-Anzeige bei Unterbrechung löschen
```

Code-Erklärung

1. Das Skript importiert Klassen zur Verwaltung digitaler Ausgabegeräte und Tasten aus der Bibliothek gpiozero. Es importiert auch die sleep-Funktion aus dem time-Modul, um Verzögerungen in der Ausführung des Skripts zu ermöglichen. Darüber hinaus wird die Bibliothek LCD1602 importiert, um das LCD1602-Display zu steuern.

   ```
   #!/usr/bin/env python3
   from gpiozero import DigitalOutputDevice, Button
   from time import sleep
   import LCD1602
   ```

2. Definiert eine benutzerdefinierte Klasse zur Verwaltung der Tastatur. Sie initialisiert die Tastatur mit den angegebenen Reihen- und Spaltenpins und bietet eine Methode `lesen`, um gedrückte Tasten zu erkennen.

   ```
   class Tastatur:
       def __init__(self, reihen_pins, spalten_pins, tasten):
           """
           Initialisieren der Tastatur mit festgelegten Reihen- und Spaltenpins und␣
   ↪Tasten.
           :param reihen_pins: Liste der GPIO-Pins für die Reihen.
           :param spalten_pins: Liste der GPIO-Pins für die Spalten.
           :param tasten: Liste der Tasten in der Tastaturanordnung.
           """
           self.reihen = [DigitalOutputDevice(pin) for pin in reihen_pins]  #␣
   ↪Einrichtung der Reihenpins
           self.spalten = [Button(pin, pull_up=False) for pin in spalten_pins]  #␣
   ↪Einrichtung der Spaltenpins
           self.tasten = tasten  # Tastaturanordnung

       def lesen(self):
           """
           Lesen und Zurückgeben einer Liste der aktuell gedrückten Tasten.
           :return: Liste der gedrückten Tasten.
           """
           gedrueckte_tasten = []
           for i, reihe in enumerate(self.reihen):
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            reihe.on()  # Aktiviere aktuelle Reihe
            for j, spalte in enumerate(self.spalten):
                if spalte.is_pressed:
                    index = i * len(self.spalten) + j
                    gedrueckte_tasten.append(self.tasten[index])
            reihe.off()  # Deaktiviere Reihe nach Überprüfung
        return gedrueckte_tasten
```

3. Richten Sie das Passwortüberprüfungssystem ein. `LAENGE` definiert die Länge des Passworts. `passwort` ist das voreingestellte richtige Passwort, während `testwort` verwendet wird, um die Eingabe des Benutzers zu speichern. `tasteIndex` verfolgt die aktuelle Position in der Benutzereingabe.

```
# Einrichtung der Passwortüberprüfung
LAENGE = 4
passwort = ['1', '9', '8', '4']  # Voreingestelltes Passwort
testwort = ['0', '0', '0', '0']  # Speicherung der Benutzereingabe
tasteIndex = 0  # Index für Eingabetasten
```

4. Funktion zum Vergleichen des eingegebenen Passworts (`testwort`) mit dem voreingestellten Passwort (`passwort`) und zur Rückgabe des Ergebnisses.

```
def ueberpruefen():
    """
    Überprüfen, ob das eingegebene Passwort mit dem voreingestellten Passwort␣
↪übereinstimmt.
    :return: 1 bei Übereinstimmung, 0 sonst.
    """
    for i in range(LAENGE):
        if passwort[i] != testwort[i]:
            return 0
    return 1
```

5. Initialisiert die Tastatur und das LCD-Display. Zeigt eine Begrüßungsnachricht und Anweisungen zum Eingeben des Passworts an.

```
def einrichten():
    """
    Einrichten der Tastatur und des LCD-Displays.
    """
    global tastatur, letzte_gedrueckte_taste
    # Pin-Konfiguration für die Tastatur
    reihen_pins = [18, 23, 24, 25]
    spalten_pins = [10, 22, 27, 17]
    tasten = ["1", "2", "3", "A",
              "4", "5", "6", "B",
              "7", "8", "9", "C",
              "*", "0", "#", "D"]

    # Tastatur und LCD initialisieren
    tastatur = Tastatur(reihen_pins, spalten_pins, tasten)
    letzte_gedrueckte_taste = []
    LCD1602.init(0x27, 1)  # LCD initialisieren
    LCD1602.clear()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    LCD1602.write(0, 0, 'WILLKOMMEN!')
    LCD1602.write(2, 1, 'Geben Sie das Passwort ein')
    sleep(2)
```

6. Die Hauptschleife zur Handhabung der Tastatureingabe und der Passwortüberprüfung. Sie aktualisiert das LCD-Display basierend auf dem eingegebenen Passwort und gibt Rückmeldung, ob das Passwort korrekt oder inkorrekt ist.

```
def schleife():
    """
    Hauptschleife zur Handhabung der Tastatureingabe und Passwortüberprüfung.
    """
    global tasteIndex, LAENGE, tastatur, letzte_gedrueckte_taste
    while True:
        gedrueckte_tasten = tastatur.lesen()
        if gedrueckte_tasten and gedrueckte_tasten != letzte_gedrueckte_taste:
            if tasteIndex < LAENGE:
                LCD1602.clear()
                LCD1602.write(0, 0, "Geben Sie das Passwort ein:")
                LCD1602.write(15 - tasteIndex, 1, gedrueckte_tasten[0])
                testwort[tasteIndex] = gedrueckte_tasten[0]
                tasteIndex += 1

            if tasteIndex == LAENGE:
                if ueberpruefen() == 0:
                    LCD1602.clear()
                    LCD1602.write(3, 0, "FALSCHE TASTE!")
                    LCD1602.write(0, 1, "Bitte erneut versuchen")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "RICHTIG!")
                    LCD1602.write(2, 1, "Willkommen zurück")
                tasteIndex = 0  # Tastenindex nach Überprüfung zurücksetzen

        letzte_gedrueckte_taste = gedrueckte_tasten
        sleep(0.1)
```

7. Führt die Einrichtung aus und tritt in die Hauptschleife ein. Ermöglicht ein sauberes Beenden des Programms mit einer Tastaturunterbrechung (Strg+C) und löscht das LCD-Display.

```
try:
    einrichten()
    schleife()
except KeyboardInterrupt:
    LCD1602.clear()  # LCD-Anzeige bei Unterbrechung löschen
```

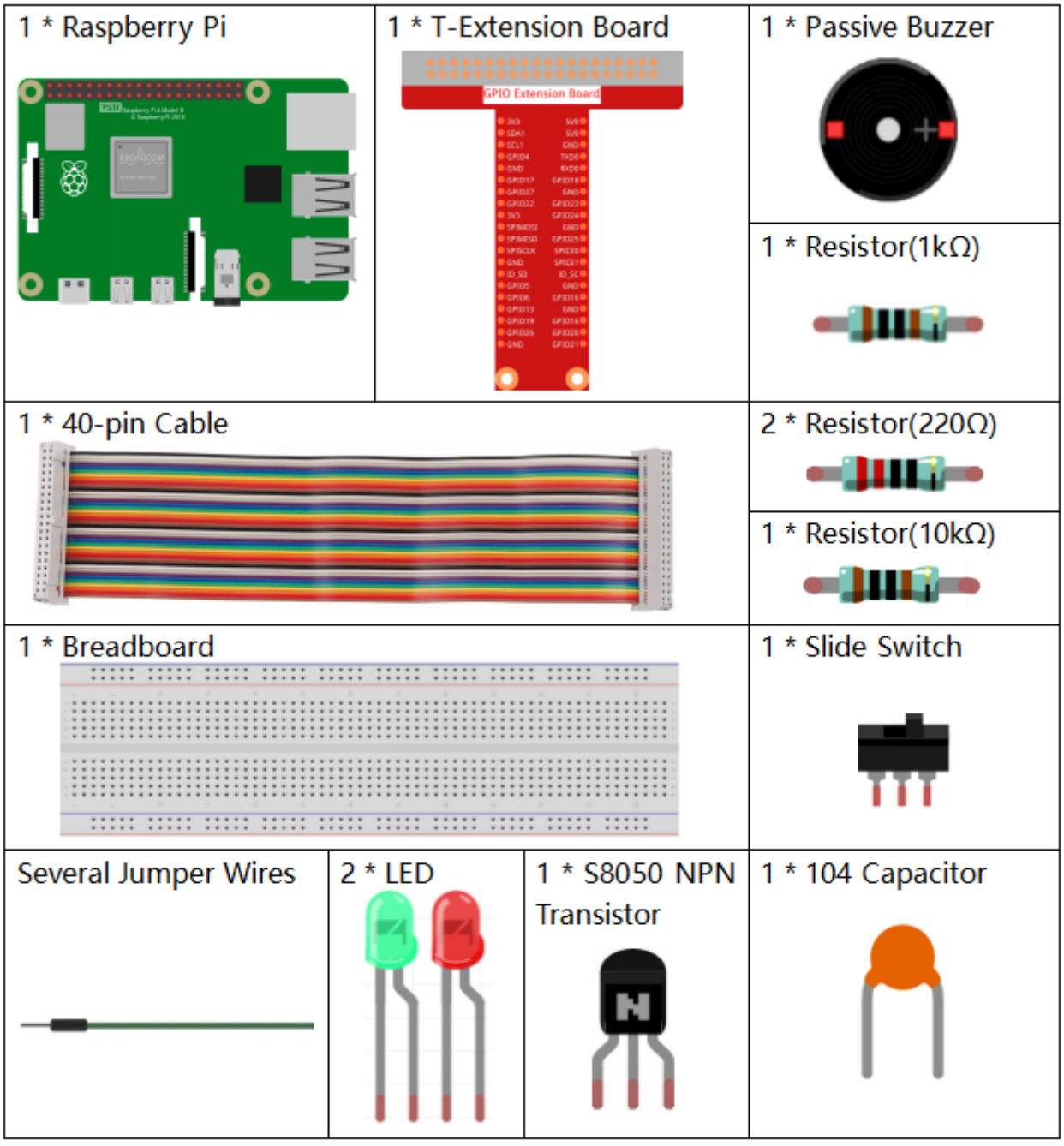
3.1.10 Alarmglocke

Einführung

In diesem Projekt werden wir ein manuelles Alarmsystem erstellen. Sie können den Kippschalter durch einen Thermistor oder einen lichtempfindlichen Sensor ersetzen, um einen Temperaturalarm oder einen Lichtalarm zu realisieren.

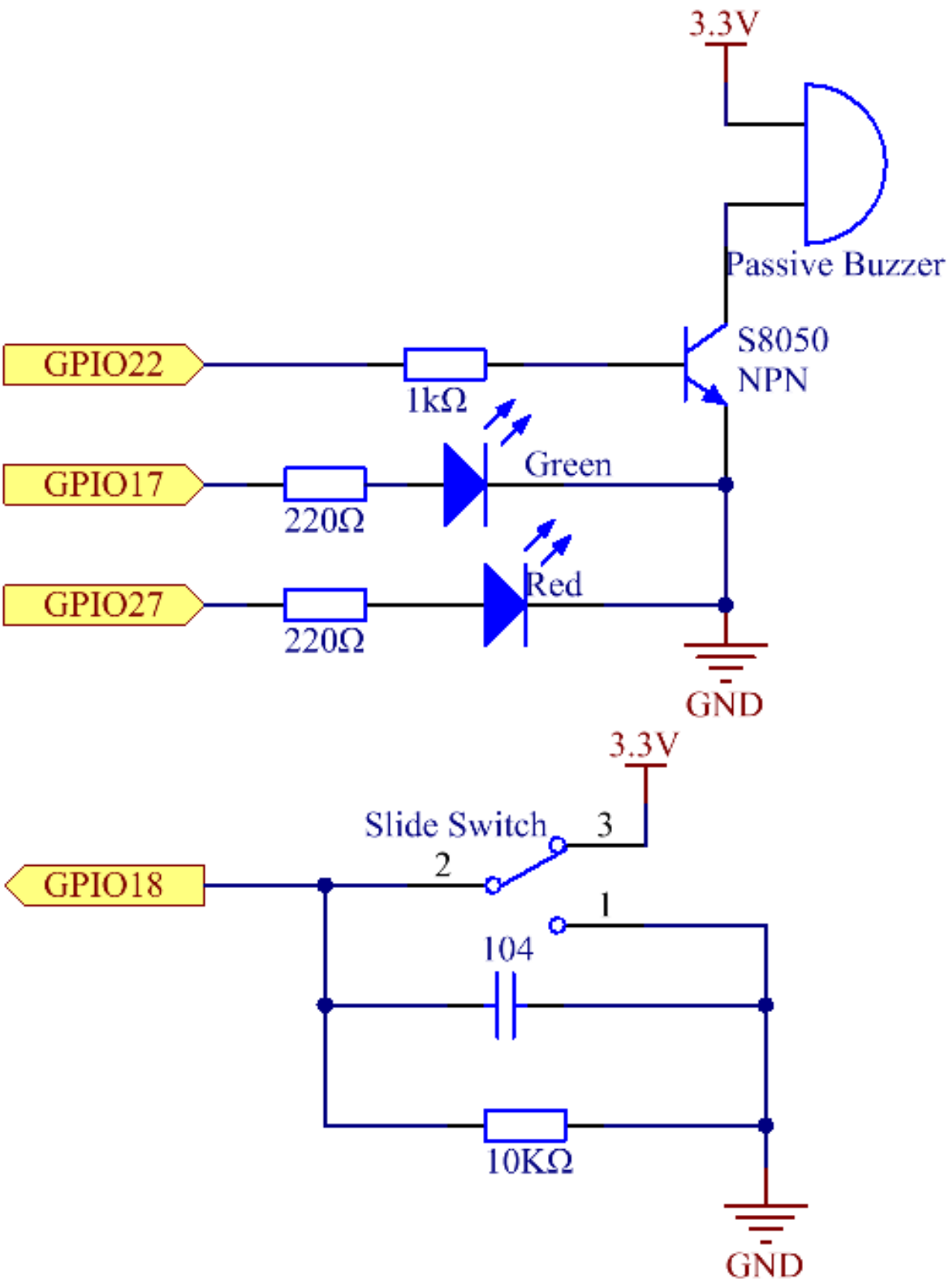
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



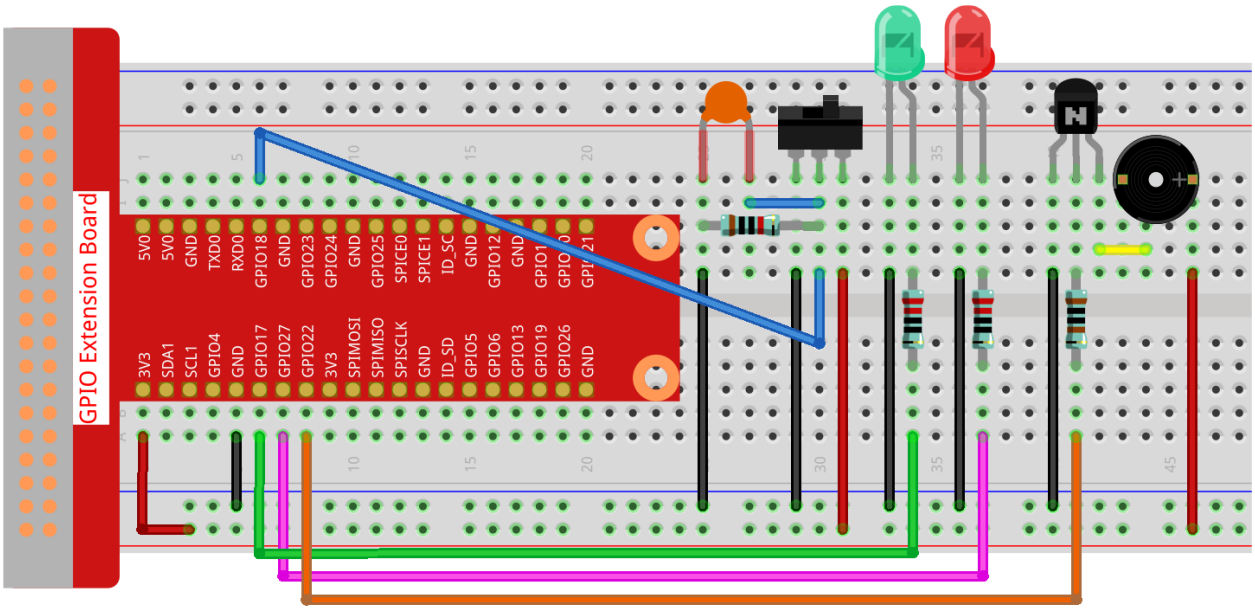
Schaltplan

T-Board Name	physisch	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
220Ω
Green
GPIO27
220Ω
Red
GND
3.3V
Slide Switch
3
2
GPIO18
1
104
10KΩ
GND

Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Wechseln Sie das Verzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.10_AlarmBell_zero.py
```

Nach dem Start des Programms wird der Kippschalter nach rechts umgelegt, und der Summer gibt Alarmtöne aus. Gleichzeitig blinken die rote und grüne LED in einem bestimmten Rhythmus.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln.

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

from gpiozero import LED, Button, TonalBuzzer
import time
import threading

# Initialisierung des TonalBuzzers an GPIO-Pin 22
BeepPin = TonalBuzzer(22)

# Initialisierung der LEDs an GPIO-Pins 17 und 27
ALedPin = LED(17)
BLedPin = LED(27)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Initialisierung des Schalters an GPIO-Pin 18
switchPin = Button(18)

# Globaler Schalter zur Steuerung des Zustands des Buzzers und der LEDs
flag = 0

def ledWork():
    """
    Steuert das Blinkmuster der LEDs basierend auf dem Zustand des Schalters.
    Wenn der Schalter gesetzt ist, blinken die LEDs abwechselnd.
    """
    while True:
        if flag:
            # Alternierendes Blinken der LEDs
            ALedPin.on()
            time.sleep(0.5)
            ALedPin.off()
            BLedPin.on()
            time.sleep(0.5)
            BLedPin.off()
        else:
            # Beide LEDs ausschalten, wenn der Schalter nicht gesetzt ist
            ALedPin.off()
            BLedPin.off()

# Definition der Melodie als Liste von Noten und deren Dauer
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)
]

def buzzerWork():
    """
    Spielt eine Melodie mit dem Summer, basierend auf dem Zustand des Schalters.
    Die Melodie wird nur gespielt, wenn der Schalter gesetzt ist.
    """
    while True:
        for note, duration in tune:
            if flag == 0:
                break
            print(note)  # Aktuelle Note in der Konsole ausgeben
            BeepPin.play(note)  # Aktuelle Note abspielen
            time.sleep(duration)  # Für die Dauer der Note pausieren
        BeepPin.stop()  # Summer nach dem Spielen der Melodie stoppen

def main():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    Überwacht das Drücken des Schalters, um den Zustand des Schalters zu aktualisieren.
    Setzt den Schalter, wenn der Knopf gedrückt wird.
    """
    global flag
    while True:
        flag = 1 if switchPin.is_pressed else 0

try:
    # Initialisierung und Start der Threads für Summer- und LED-Steuerung
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
    main()

except KeyboardInterrupt:
    # Stoppen des Summers und Ausschalten der LEDs bei Programmabbruch
    BeepPin.stop()
    ALedPin.off()
    BLedPin.off()
```

Code-Erklärung

1. Dieser Abschnitt beinhaltet den Import wesentlicher Bibliotheken zur Implementierung von Verzögerungen und Threading. Es werden auch die Klassen LED, Button und TonalBuzzer aus der gpiozero-Bibliothek importiert, die für die Steuerung von GPIO-Geräten auf einem Raspberry Pi entscheidend sind.

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

   from gpiozero import LED, Button, TonalBuzzer
   import time
   import threading
   ```

2. Stellt einen Summer an GPIO-Pin 22, zwei LEDs an GPIO-Pins 17 und 27 ein und initialisiert einen Knopf an GPIO-Pin 18. Ein globaler Schalter wird ebenfalls definiert, um den Zustand des Buzzers und der LEDs zu verwalten.

   ```
   # Initialisierung des TonalBuzzers an GPIO-Pin 22
   BeepPin = TonalBuzzer(22)

   # Initialisierung der LEDs an GPIO-Pins 17 und 27
   ALedPin = LED(17)
   BLedPin = LED(27)

   # Initialisierung des Schalters an GPIO-Pin 18
   switchPin = Button(18)

   # Globaler Schalter zur Steuerung des Zustands des Buzzers und der LEDs
   flag = 0
   ```

3. Diese Funktion steuert das Blinken der LEDs basierend auf dem Zustand des Schalters. Wenn der Schalter gesetzt ist (1), wechselt sie abwechselnd zwischen dem Ein- und Ausschalten jeder LED. Wenn nicht gesetzt (0), werden beide LEDs ausgeschaltet.

```
def ledWork():
    """
    Steuert das Blinkmuster der LEDs basierend auf dem Zustand des Schalters.
    Wenn der Schalter gesetzt ist, blinken die LEDs abwechselnd.
    """
    while True:
        if flag:
            # Alternierendes Blinken der LEDs
            ALedPin.on()
            time.sleep(0.5)
            ALedPin.off()
            BLedPin.on()
            time.sleep(0.5)
            BLedPin.off()
        else:
            # Beide LEDs ausschalten, wenn der Schalter nicht gesetzt ist
            ALedPin.off()
            BLedPin.off()
```

4. Die Melodie wird als eine Sequenz von Noten (Frequenz) und Dauern (Sekunden) definiert.

```
# Definition der Melodie als Liste von Noten und deren Dauer
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. Spielt eine vordefinierte Melodie ab, wenn der Schalter gesetzt ist. Die Melodie stoppt, wenn der Schalter während des Spielens zurückgesetzt wird.

```
def buzzerWork():
    """
    Spielt eine Melodie mit dem Summer, basierend auf dem Zustand des Schalters.
    Die Melodie wird nur gespielt, wenn der Schalter gesetzt ist.
    """
    while True:
        for note, duration in tune:
            if flag == 0:
                break
            print(note)  # Aktuelle Note in der Konsole ausgeben
            BeepPin.play(note)  # Aktuelle Note abspielen
            time.sleep(duration)  # Für die Dauer der Note pausieren
        BeepPin.stop()  # Summer nach dem Spielen der Melodie stoppen
```

6. Überprüft kontinuierlich den Zustand des Knopfes, um den Schalter zu setzen oder zurückzusetzen.

```
def main():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    Überwacht das Drücken des Schalters, um den Zustand des Schalters zu␣
↪aktualisieren.
    Setzt den Schalter, wenn der Knopf gedrückt wird.
    """
    global flag
    while True:
        flag = 1 if switchPin.is_pressed else 0
```

7. Threads für `buzzerWork` und `ledWork` werden gestartet, wodurch sie gleichzeitig mit der Hauptfunktion laufen können.

```
try:
    # Initialisierung und Start der Threads für Summer- und LED-Steuerung
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
    main()
```

8. Stoppt den Summer und schaltet die LEDs aus, wenn das Programm unterbrochen wird, um einen sauberen Ausstieg zu gewährleisten.

```
except KeyboardInterrupt:
    # Stoppen des Summers und Ausschalten der LEDs bei Programmabbruch
    BeepPin.stop()
    ALedPin.off()
    BLedPin.off()
```

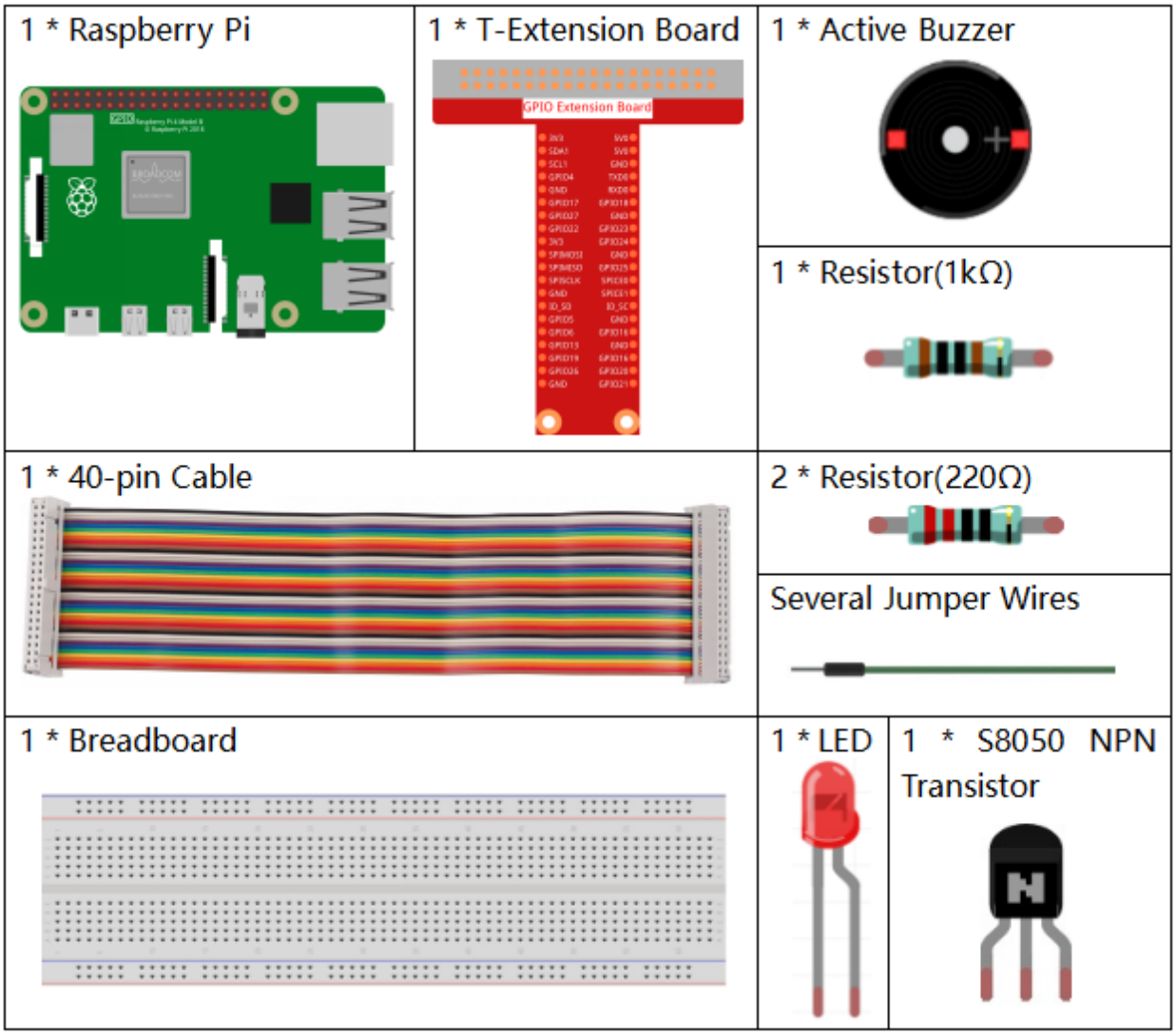
3.1.11 Morsecode-Generator

Einführung

In diesem Projekt werden wir einen Morsecode-Generator erstellen, bei dem Sie eine Reihe von englischen Buchstaben in den Raspberry Pi eingeben, um diese als Morsecode darzustellen.

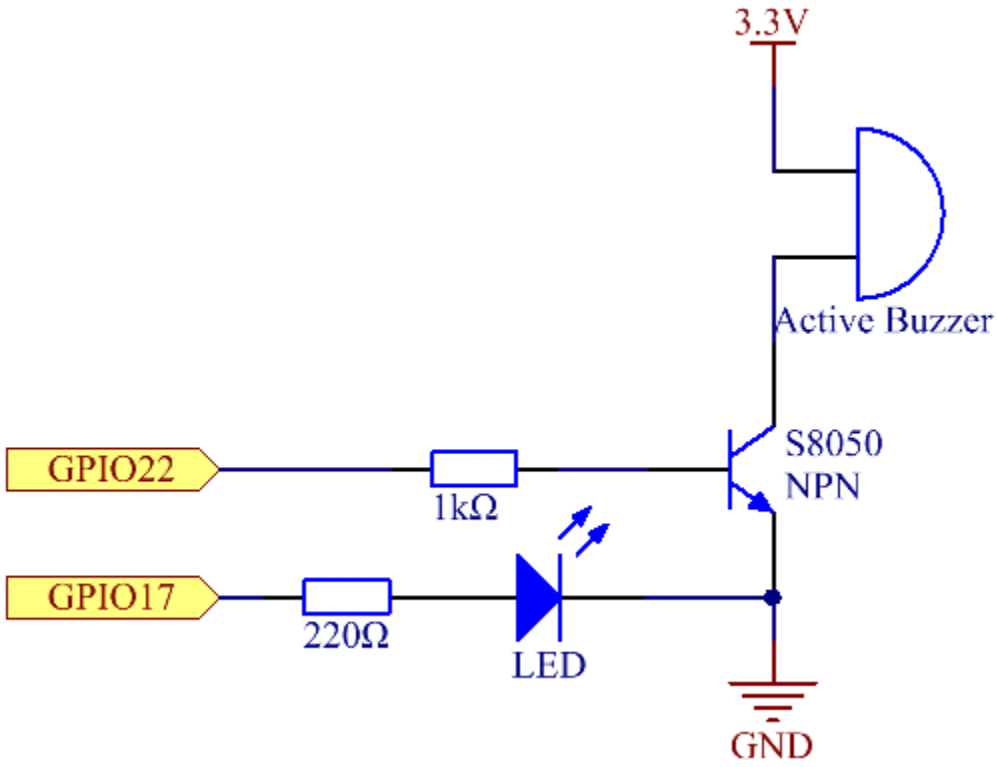
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



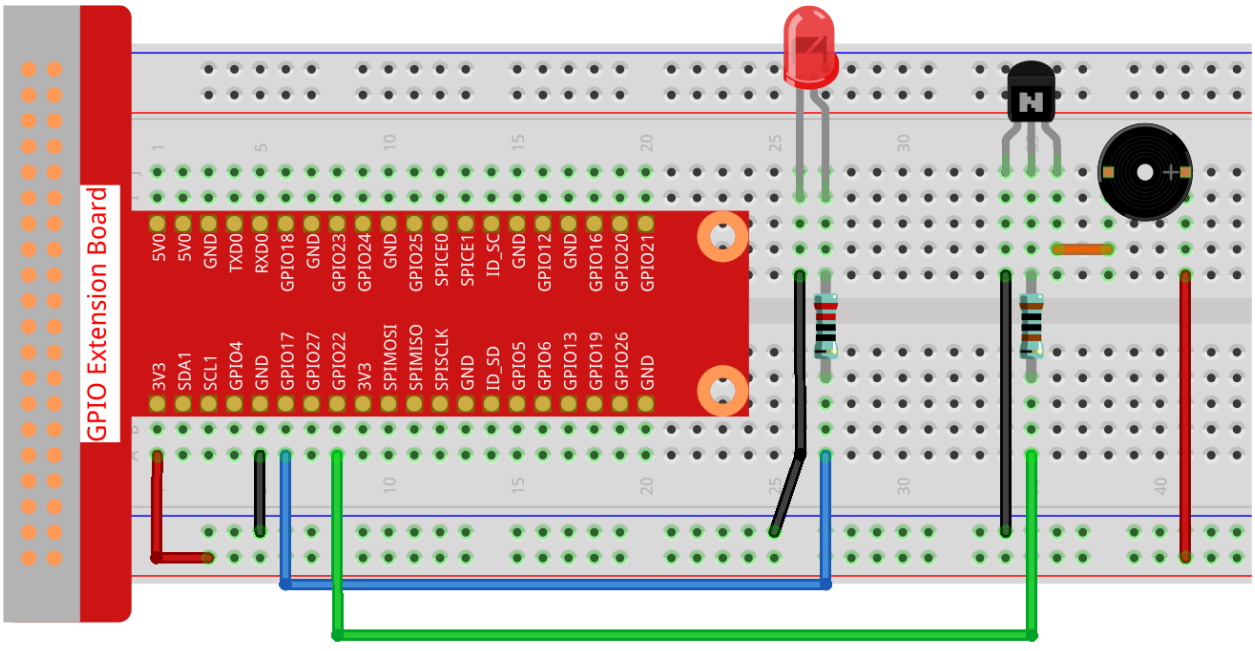
Schaltplan

T-Board Name	physisch	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO22	Pin 15	3	22



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis. (Achten Sie auf die Pole des Buzzers: Der mit + markierte ist der positive Pol, der andere der negative.)



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Ausführen.

```
sudo python3 3.1.11_MorseCodeGenerator_zero.py
```

Nachdem das Programm gestartet wurde, geben Sie eine Reihe von Zeichen ein, und der Summer sowie die LED senden die entsprechenden Morsecode-Signale.

Code

```
#!/usr/bin/env python3
from gpiozero import Buzzer, LED
import time

# Initialisierung des Buzzers und der LED an GPIO-Pins
BeepPin = Buzzer(22)
ALedPin = LED(17)

# Morsecode-Darstellung für Zeichen
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 on():
    """ Schaltet den Summer und die LED ein. """
    BeepPin.on()
    ALedPin.on()

def off():
    """ Schaltet den Summer und die LED aus. """
    BeepPin.off()
    ALedPin.off()

def beep(dt):  # dt für Verzögerungszeit.
    """
    Erzeugt einen Piepton und ein LED-Blinken für die angegebene Dauer.
    :param dt: Dauer für den Piepton und das Blinken.
    """
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)

def morsecode(code):
    """
    Konvertiert den eingegebenen Code in Morsecode und signalisiert ihn mit dem Summer␣
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪und der LED.
    :param code: Der in Morsecode umzuwandelnde Text.
    """
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause / 2)  # Kurzer Piepton für Punkt
            if tap == '1':
                beep(pause)      # Langer Piepton für Strich
        time.sleep(pause)  # Pause zwischen Buchstaben

def destroy():
    """ Ressourcen bei Skriptbeendigung aufräumen. """
    print("")
    BeepPin.off()
    ALedPin.off()

try:
    while True:
        code = input("Bitte geben Sie die Nachricht ein:")
        code = code.upper()  # Umwandlung in Großbuchstaben für Morsecode-Suche
        print(code)
        morsecode(code)
except KeyboardInterrupt:
    destroy()
```

Code-Erklärung

1. Dieser Code importiert die Klassen Buzzer und LED aus der gpiozero-Bibliothek. Diese Klassen sind wesentlich für die Steuerung der entsprechenden GPIO-Geräte auf dem Raspberry Pi.

   ```
   #!/usr/bin/env python3
   from gpiozero import Buzzer, LED
   import time
   ```

2. Initialisiert den Summer an GPIO-Pin 22 und die LED an GPIO-Pin 17 und erleichtert so die Steuerung dieser Komponenten.

   ```
   # Initialisierung des Buzzers und der LED an GPIO-Pins
   BeepPin = Buzzer(22)
   ALedPin = LED(17)
   ```

3. Definiert die Struktur „MORSE“, ein Wörterbuch, das Morsecode-Darstellungen für die Buchstaben A-Z, Zahlen 0-9 und Symbole wie „?“, „/“, „:“, „„“, „.“, „;“, „!“, „@“ enthält, wobei „0“ einen Punkt und „1“ einen Strich bedeutet.

   ```
   # Morsecode-Darstellung für Zeichen
   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',
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    '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. Die Funktion „on()“ startet den Summer und die LED. Die Funktion „off()“ wird verwendet, um den Summer und die LED auszuschalten.

```
def on():
    """ Schaltet den Summer und die LED ein. """
    BeepPin.on()
    ALedPin.on()

def off():
    """ Schaltet den Summer und die LED aus. """
    BeepPin.off()
    ALedPin.off()
```

5. Definiert eine Funktion „beep()“, um den Summer und die LED in einem bestimmten Intervall von „dt“ Töne abzugeben und zu blinken.

```
def beep(dt):  # dt für Verzögerungszeit.
    """
    Erzeugt einen Piepton und ein LED-Blinken für die angegebene Dauer.
    :param dt: Dauer für den Piepton und das Blinken.
    """
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

6. Die Funktion „morsecode()“ wird verwendet, um den Morsecode der eingegebenen Zeichen zu verarbeiten, indem das „1“ des Codes weiterhin Töne oder Lichter abgibt und das „0“ kurz Töne oder Lichter abgibt, z.B. Eingabe „SOS“, und es wird ein Signal mit drei kurzen, drei langen und dann drei kurzen Segmenten „ · · · - - - · · · “ erzeugt.

```
def morsecode(code):
    """
    Konvertiert den eingegebenen Code in Morsecode und signalisiert ihn mit dem␣
↪Summer und der LED.
    :param code: Der in Morsecode umzuwandelnde Text.
    """
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause / 2)  # Kurzer Piepton für Punkt
            if tap == '1':
                beep(pause)      # Langer Piepton für Strich
        time.sleep(pause)  # Pause zwischen Buchstaben
```

7. Definiert eine Funktion namens „destroy“, die sowohl den Summer als auch die LED ausschaltet. Diese Funktion soll aufgerufen werden, wenn das Skript beendet wird, um sicherzustellen, dass die GPIO-Pins nicht in einem aktiven Zustand belassen werden.

```
def destroy():
    """ Ressourcen bei Skriptbeendigung aufräumen. """
    print("")
    BeepPin.off()
    ALedPin.off()
```

8. Wenn Sie die relevanten Zeichen mit der Tastatur eingeben, konvertiert „upper()“ die eingegebenen Buchstaben in ihre Großbuchstabenform. „printf()“ druckt dann den Klartext auf dem Computerbildschirm aus, und die Funktion „morsecod()“ veranlasst den Summer und die LED, Morsecode auszusenden.

```
try:
    while True:
        code = input("Bitte geben Sie die Nachricht ein:")
        code = code.upper()  # Umwandlung in Großbuchstaben für Morsecode-Suche
        print(code)
        morsecode(code)
except KeyboardInterrupt:
    destroy()
```

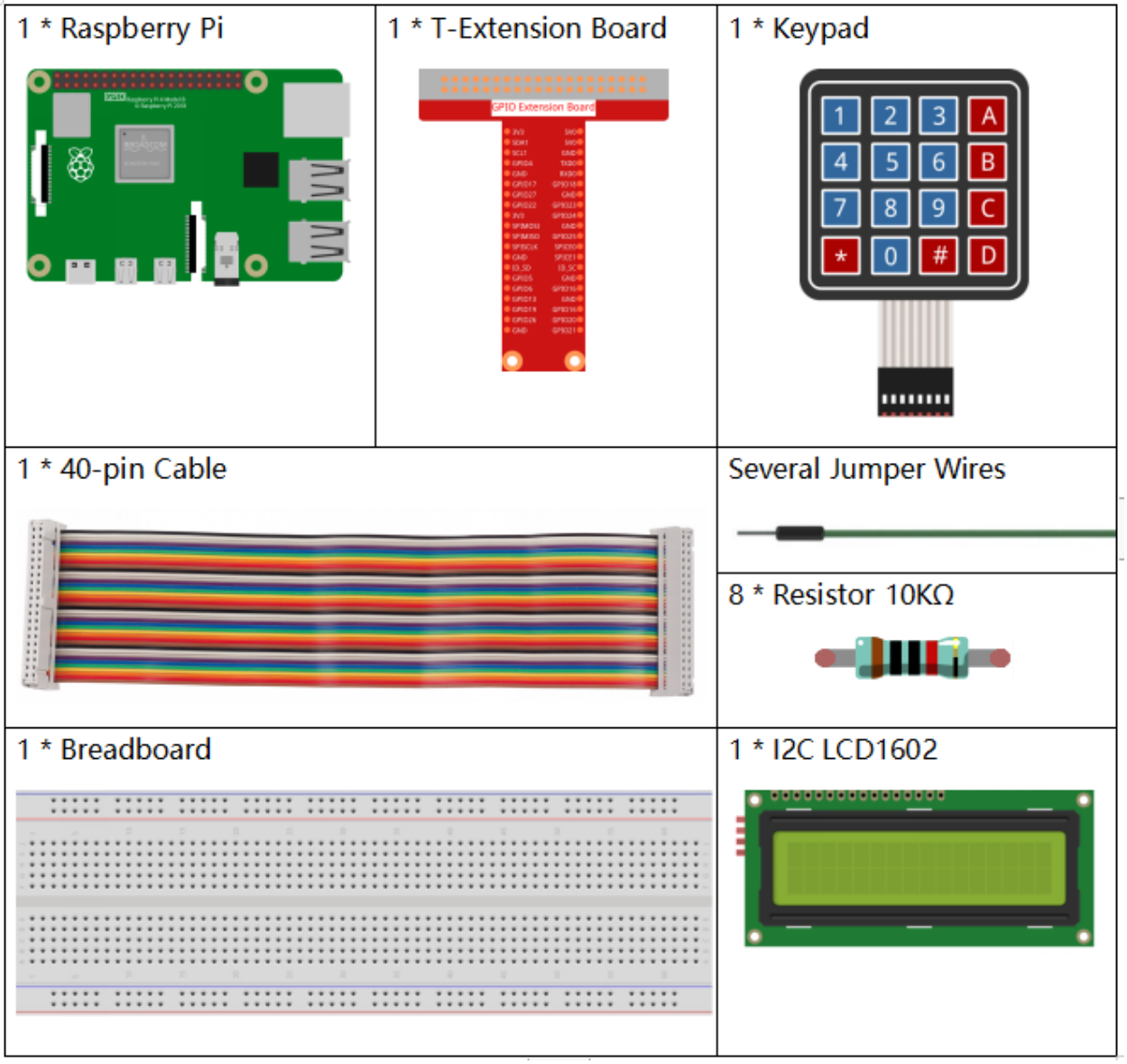
3.1.12 SPIEL – Zahl raten

Einführung

„Zahl raten“ ist ein unterhaltsames Partyspiel, bei dem Sie und Ihre Freunde abwechselnd eine Zahl (0~99) eingeben. Der Bereich wird mit der Eingabe der Zahl kleiner, bis ein Spieler das Rätsel richtig beantwortet. Dann wird dieser Spieler besiegt und bestraft. Zum Beispiel, wenn die Glückszahl 51 ist, die die Spieler nicht sehen können, und Spieler gibt 50 ein, ändert sich der Bereich der Zahlen zu 50~99; gibt Spieler 70 ein, kann der Bereich der Zahlen 50~70 sein; gibt Spieler 51 ein, ist dieser Spieler der Pechvogel. Hier verwenden wir ein Keypad zur Eingabe von Zahlen und ein LCD zur Ausgabe von Ergebnissen.

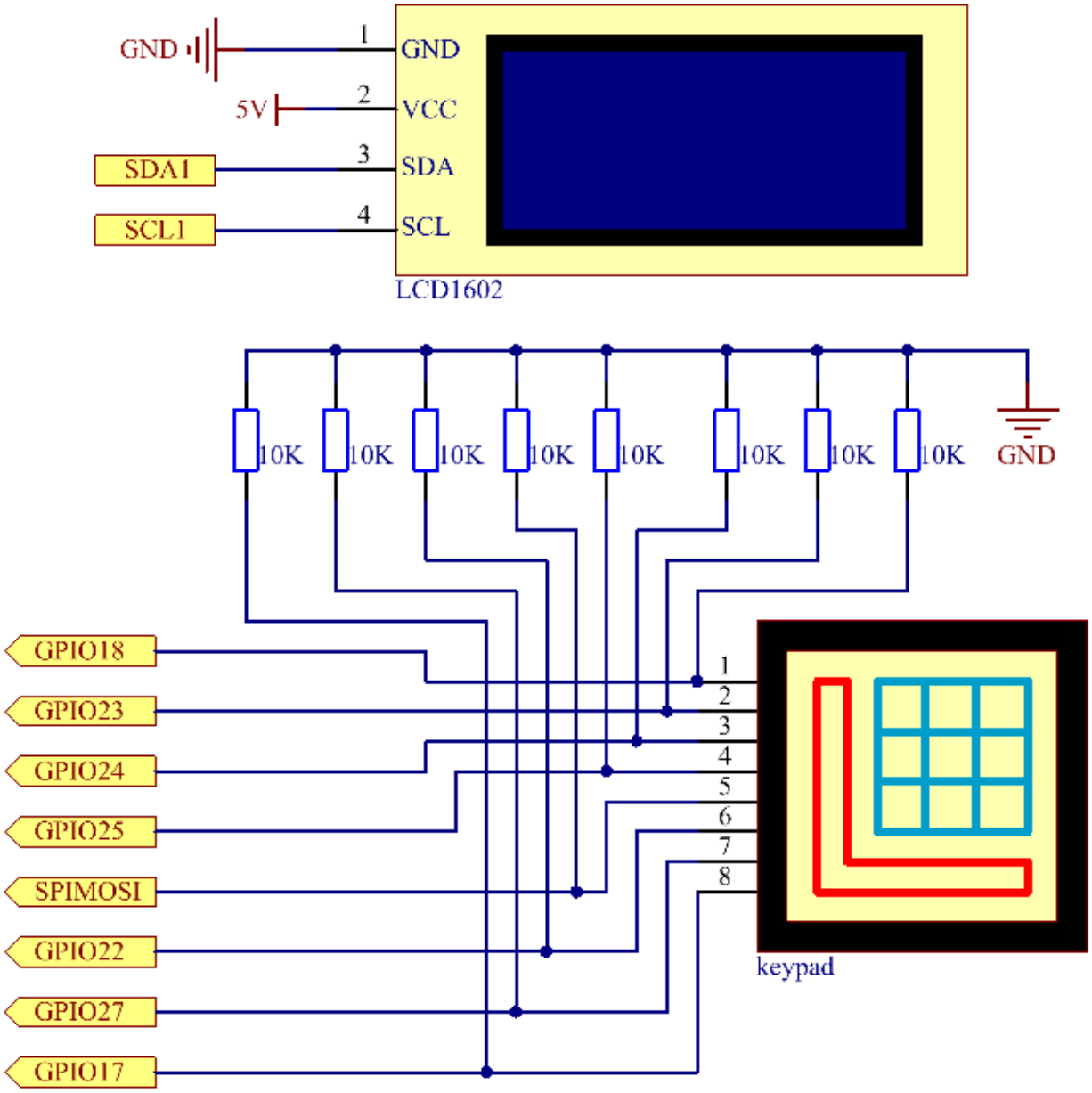
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



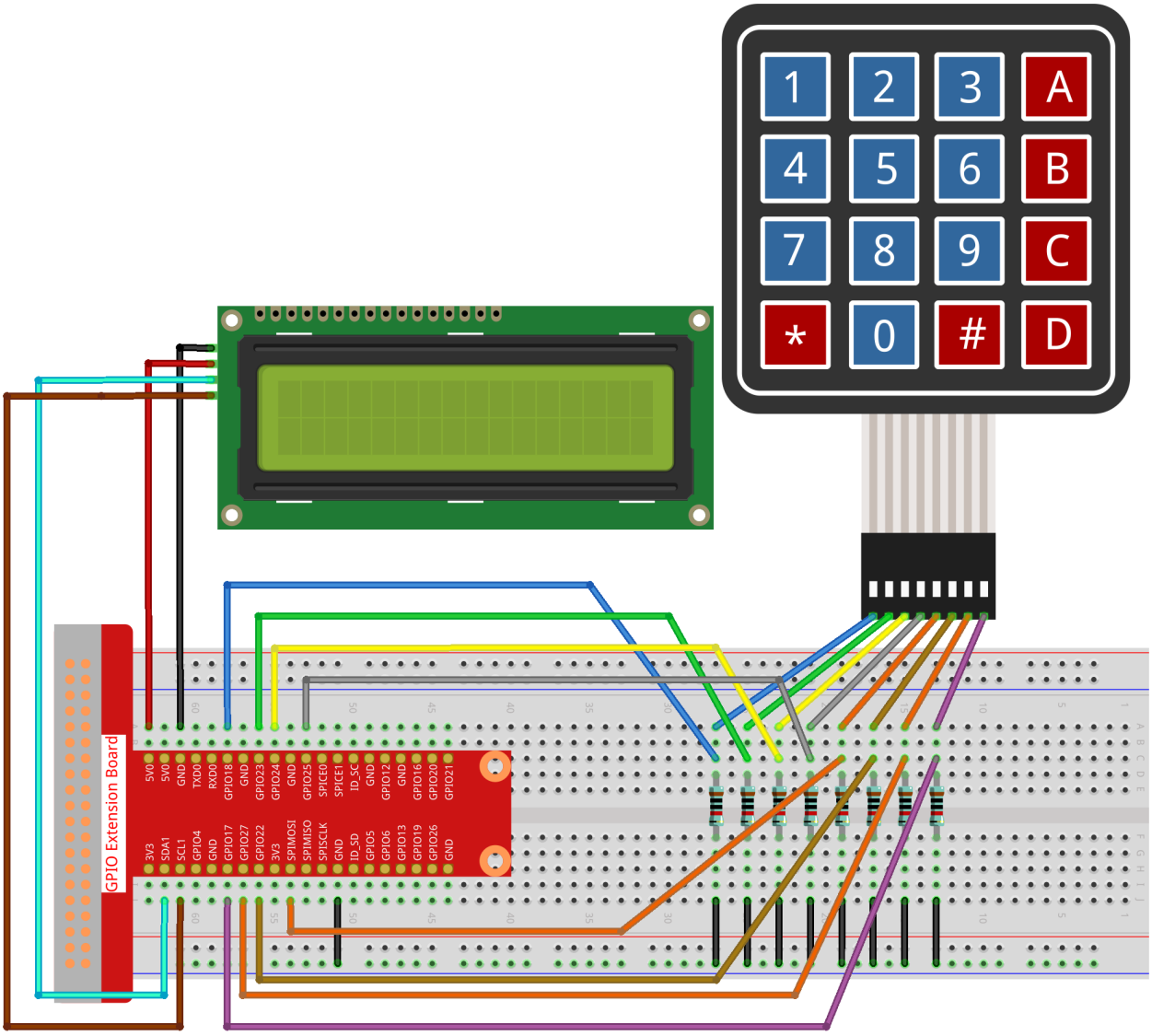
Schaltplan

T-Board Name	physisch	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)



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Einrichten von I2C (siehe *I2C-Konfiguration*.)

Schritt 3: Wechseln Sie das Verzeichnis.

```
cd ~/raphael-kit/python-pi5
```

Schritt 4: Ausführen.

```
sudo python3 3.1.12_GAME_GuessNumber_zero.py
```

Nachdem das Programm ausgeführt wurde, erscheint die Startseite auf dem LCD:

```
Willkommen!
Drücken Sie A, um zu starten!
```

Drücken Sie „A“, und das Spiel beginnt und die Spielseite erscheint auf dem LCD.

```
Zahl eingeben:
0 ‹Punkt‹ 99
```

Eine zufällige Zahl „**Punkt**“ wird erzeugt, aber nicht auf dem LCD angezeigt, wenn das Spiel beginnt, und Ihre Aufgabe ist es, sie zu erraten. Die von Ihnen eingegebene Zahl erscheint am Ende der ersten Zeile, bis die endgültige Berechnung abgeschlossen ist. (Drücken Sie „D“, um den Vergleich zu starten, und wenn die eingegebene Zahl größer als **10** ist, startet der automatische Vergleich.)

Der Zahlenbereich von „Punkt“ wird in der zweiten Zeile angezeigt. Und Sie müssen die Zahl innerhalb des Bereichs eingeben. Wenn Sie eine Zahl eingeben, wird der Bereich enger; wenn Sie die Glückszahl glücklicherweise oder unglücklicherweise erraten, erscheint „Sie haben es geschafft!“

Bemerkung:

- Wenn Sie den Fehler „FileNotFoundError: [Errno 2] No such file or directory: ‚/dev/i2c-1‘“ erhalten, beziehen Sie sich bitte auf *I2C-Konfiguration*, um den I2C zu aktivieren.
- Bei dem Fehler „ModuleNotFoundError: No module named ‚smbus2‘“ führen Sie bitte „sudo pip3 install smbus2“ aus.
- Wenn der Fehler „OSError: [Errno 121] Remote I/O error“ auftritt, bedeutet dies, dass das Modul falsch verkabelt ist oder defekt ist.
- Wenn der Code und die Verkabelung in Ordnung sind, das LCD jedoch keinen Inhalt anzeigt, können Sie das Potentiometer auf der Rückseite drehen, um den Kontrast zu erhöhen.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

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

from gpiozero import DigitalOutputDevice, Button
from time import sleep
import LCD1602
import random

class Keypad:
    def __init__(self, rows_pins, cols_pins, keys):
        """
        Initialisiert das Keypad mit den angegebenen Reihen- und Spalten-Pins und dem␣
↪Tastaturlayout.
        :param rows_pins: Liste der GPIO-Pins für die Reihen.
        :param cols_pins: Liste der GPIO-Pins für die Spalten.
        :param keys: Layout der Tasten auf dem Keypad.
        """
        self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # Reihen-Pins␣
↪einrichten
        self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # Spalten-Pins␣
↪einrichten
        self.keys = keys  # Tastaturlayout definieren

    def read(self):
        """
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        Liest und gibt die derzeit gedrückten Tasten zurück.
        :return: Liste der gedrückten Tasten.
        """
        pressed_keys = []
        for i, row in enumerate(self.rows):
            row.on()  # Aktiviere aktuelle Reihe
            for j, col in enumerate(self.cols):
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])  # Gedrückte Taste hinzufügen
            row.off()  # Deaktiviere Reihe
        return pressed_keys

# Spielbezogene Variablen
count = 0
pointValue = 0
upper = 99
lower = 0

def setup():
    """
    Einrichtungsfunktion zum Initialisieren des Keypads und des LCD-Displays.
    """
    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 initialisieren
    LCD1602.clear()
    LCD1602.write(0, 0, 'Willkommen!')
    LCD1602.write(0, 1, 'Drücken Sie A zum Starten!')

def init_new_value():
    """
    Initialisiert einen neuen Zielwert und setzt die Spielparameter zurück.
    """
    global pointValue, upper, lower, count
    pointValue = random.randint(0, 99)
    upper = 99
    lower = 0
    count = 0
    print('Punkt ist %d' % pointValue)

def detect_point():
    """
    Überprüft, ob die geratene Zahl das Ziel ist, zu hoch oder zu niedrig ist.
    :return: 1, wenn die Vermutung korrekt ist, 0 andernfalls.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    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
    count = 0
    return 0

def lcd_show_input(result):
    """
    Zeigt den aktuellen Spielstand und die Ergebnisse auf dem LCD an.
    :param result: Ergebnis der letzten Vermutung (0 oder 1).
    """
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0, 1, 'Sie haben es geschafft!')
        sleep(5)
        init_new_value()
        lcd_show_input(0)
    else:
        LCD1602.write(0, 0, 'Zahl eingeben:')
        LCD1602.write(13, 0, str(count))
        LCD1602.write(0, 1, str(lower))
        LCD1602.write(3, 1, ' < Punkt < ')
        LCD1602.write(13, 1, str(upper))

def loop():
    """
    Hauptschleife für die Behandlung der Keypad-Eingabe und die Aktualisierung des␣
↪Spielstands.
    """
    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", "#", "*"]:
                    continue
                count = count * 10 + int(pressed_keys[0])
                if count >= 10:
                    result = detect_point()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                lcd_show_input(result)
            print(pressed_keys)
        last_key_pressed = pressed_keys
        sleep(0.1)

try:
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # LCD bei Unterbrechung löschen
```

Code-Erklärung

1. Dieser Abschnitt importiert wesentliche Klassen aus der GPIO Zero-Bibliothek, um digitale Ausgabegeräte und Tasten zu verwalten. Darüber hinaus beinhaltet er die Sleep-Funktion aus dem Time-Modul, um Verzögerungen im Skript zu ermöglichen. Die LCD1602-Bibliothek wird für den Betrieb des LCD-Displays importiert, was nützlich ist, um Text oder Datenausgaben anzuzeigen. Außerdem wird die Random-Bibliothek integriert, die Funktionen zur Generierung von Zufallszahlen bietet, was für verschiedene Aspekte des Projekts vorteilhaft sein kann.

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

   from gpiozero import DigitalOutputDevice, Button
   from time import sleep
   import LCD1602
   import random
   ```

2. Definiert eine Klasse für das Keypad, initialisiert es mit Reihen- und Spalten-Pins und definiert eine Methode, um gedrückte Tasten zu lesen.

   ```
   class Keypad:
       def __init__(self, rows_pins, cols_pins, keys):
           """
           Initialisiert das Keypad mit den angegebenen Reihen- und Spalten-Pins und dem␣
   ↪Tastaturlayout.
           :param rows_pins: Liste der GPIO-Pins für die Reihen.
           :param cols_pins: Liste der GPIO-Pins für die Spalten.
           :param keys: Layout der Tasten auf dem Keypad.
           """
           self.rows = [DigitalOutputDevice(pin) for pin in rows_pins]  # Reihen-Pins␣
   ↪einrichten
           self.cols = [Button(pin, pull_up=False) for pin in cols_pins]  # Spalten-Pins␣
   ↪einrichten
           self.keys = keys  # Tastaturlayout definieren

       def read(self):
           """
           Liest und gibt die derzeit gedrückten Tasten zurück.
           :return: Liste der gedrückten Tasten.
           """
           pressed_keys = []
           for i, row in enumerate(self.rows):
                   row.on()  # Aktiviere aktuelle Reihe
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            for j, col in enumerate(self.cols):
                if col.is_pressed:
                    index = i * len(self.cols) + j
                    pressed_keys.append(self.keys[index])  # Gedrückte Taste␣
↪hinzufügen
            row.off()  # Deaktiviere Reihe
        return pressed_keys
```

3. Initialisiert eine Variable „count“ als null, die möglicherweise für die Verfolgung von Versuchen oder bestimmten Werten im Spiel verwendet wird. Konfiguriert das Keypad und das LCD-Display mit einer Willkommensnachricht und Anweisungen. Initialisiert die Variable „pointValue“ auf null, die möglicherweise einen Zielpunktestand oder -wert im Spiel darstellt. Definiert eine „upper“-Grenze für das Spiel, die zunächst auf 99 festgelegt wird, was in einem Zahlraten-Spiel das Maximum sein könnte. Setzt die „lower“-Grenze beginnend von null, die wahrscheinlich als minimale Grenze im Spiel verwendet wird.

```
# Spielbezogene Variablen
count = 0
pointValue = 0
upper = 99
lower = 0
```

4. Richtet das Keypad und das LCD-Display ein, zeigt eine Willkommensnachricht und Anweisungen an.

```
def setup():
    """
    Einrichtungsfunktion zum Initialisieren des Keypads und des LCD-Displays.
    """
    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 initialisieren
    LCD1602.clear()
    LCD1602.write(0, 0, 'Willkommen!')
    LCD1602.write(0, 1, 'Drücken Sie A zum Starten!')
```

5. Initialisiert einen neuen Zielwert für das Spiel und setzt die Spielparameter zurück.

```
def init_new_value():
    """
    Initialisiert einen neuen Zielwert und setzt die Spielparameter zurück.
    """
    global pointValue, upper, lower, count
    pointValue = random.randint(0, 99)
    upper = 99
    lower = 0
    count = 0
    print('Punkt ist %d' % pointValue)
```

6. Überprüft, ob die geratene Zahl mit dem Ziel übereinstimmt, und aktualisiert entsprechend den Bereich der Vermutungen.

```
def detect_point():
    """
    Überprüft, ob die geratene Zahl das Ziel ist, zu hoch oder zu niedrig ist.
    :return: 1, wenn die Vermutung korrekt ist, 0 andernfalls.
    """
    global count, upper, lower
    if count > pointValue and count < upper:
        upper = count
    elif count < pointValue und count > lower:
        lower = count
    elif count == pointValue:
        count = 0
        return 1
    count = 0
    return 0
```

7. Zeigt den Spielstand auf dem LCD an, zeigt die aktuelle Vermutung, den Bereich und das Ergebnis an.

```
def lcd_show_input(result):
    """
    Zeigt den aktuellen Spielstand und die Ergebnisse auf dem LCD an.
    :param result: Ergebnis der letzten Vermutung (0 oder 1).
    """
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0, 1, 'Sie haben es geschafft!')
        sleep(5)
        init_new_value()
        lcd_show_input(0)
    else:
        LCD1602.write(0, 0, 'Zahl eingeben:')
        LCD1602.write(13, 0, str(count))
        LCD1602.write(0, 1, str(lower))
        LCD1602.write(3, 1, ' < Punkt < ')
        LCD1602.write(13, 1, str(upper))
```

8. Die Hauptschleife für die Handhabung der Keypad-Eingabe, die Aktualisierung des Spielstands und die Anzeige der Ergebnisse auf dem LCD.

```
def loop():
    """
    Hauptschleife für die Behandlung der Keypad-Eingabe und die Aktualisierung des␣
↪Spielstands.
    """
    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()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                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", "#", "*"]:
                    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. Führt die Einrichtung aus und betritt die Hauptschleife des Spiels, ermöglicht einen sauberen Ausstieg mit einem Tastaturinterrupt.

```
try:
    setup()
    loop()
except KeyboardInterrupt:
    LCD1602.clear()  # LCD bei Unterbrechung löschen
```

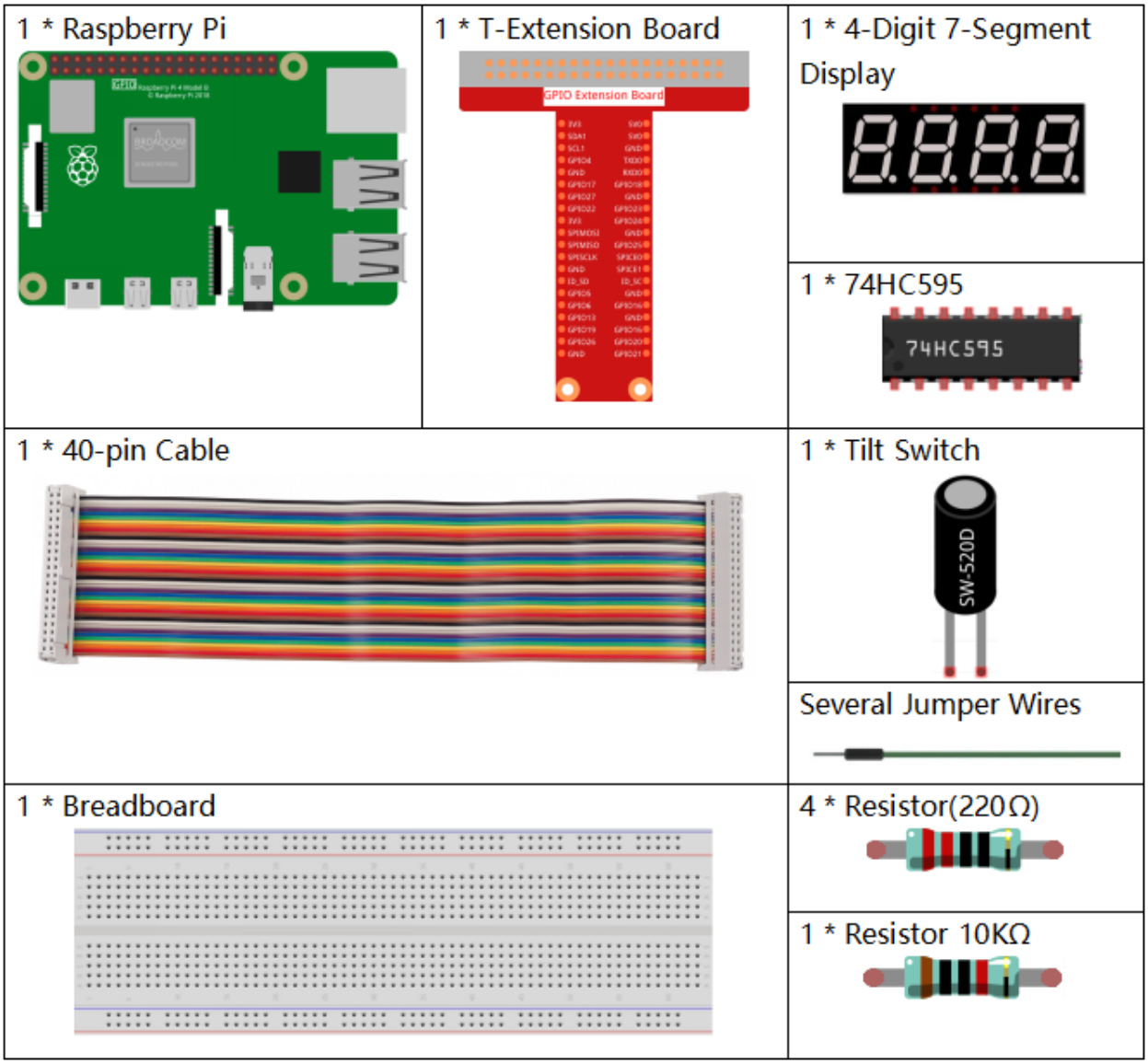
3.1.13 SPIEL - 10 Sekunden

Einführung

Als Nächstes machen wir ein Spielgerät, um Ihre Konzentration herauszufordern. Binden Sie den Kippschalter an einen Stab, um einen Zauberstab zu machen. Schütteln Sie den Zauberstab, die 4-stellige Segmentanzeige beginnt zu zählen, ein erneutes Schütteln stoppt die Zählung. Wenn es Ihnen gelingt, den angezeigten Zähler auf **10.00** zu halten, dann haben Sie gewonnen. Sie können das Spiel mit Ihren Freunden spielen, um zu sehen, wer der Zeitmagier ist.

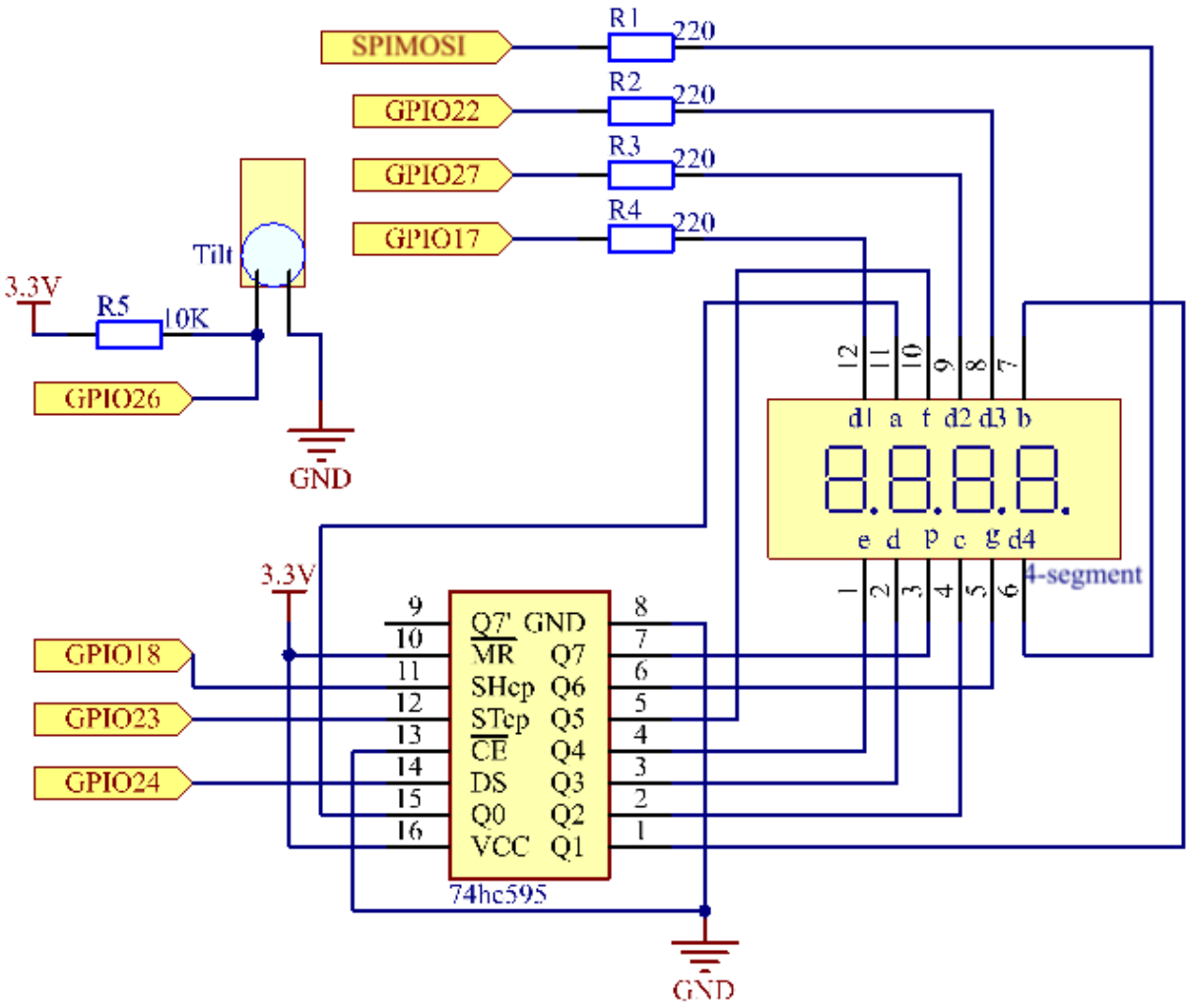
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



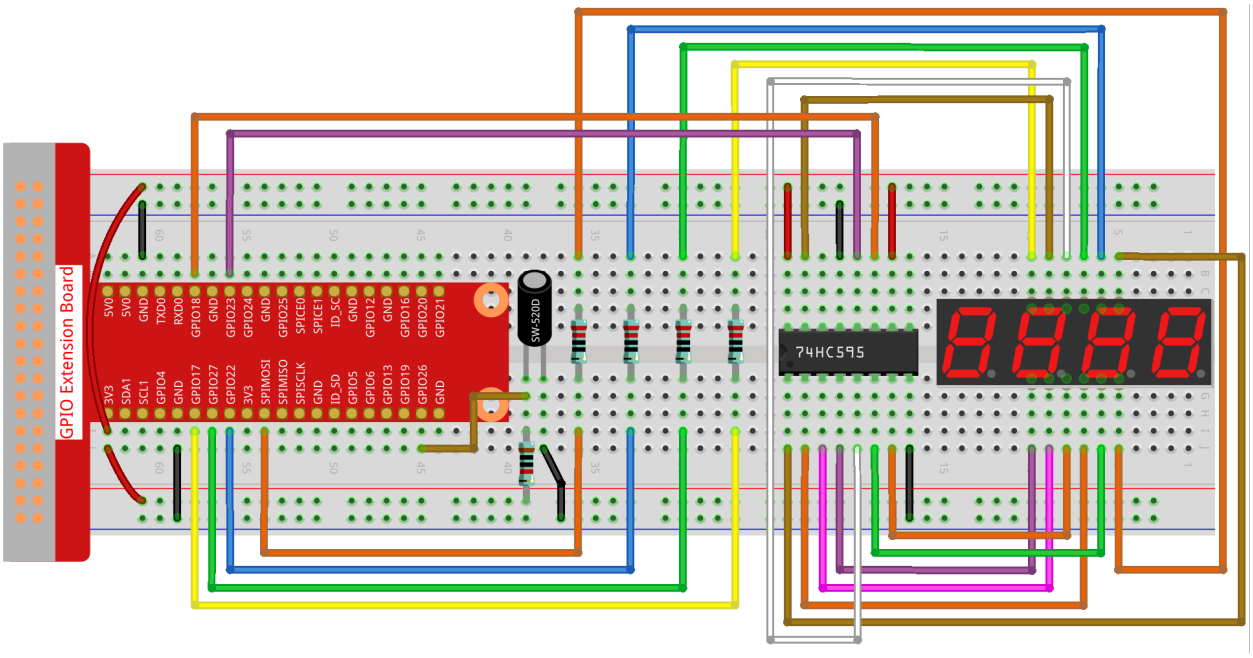
Schaltplan

T-Board Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Gehen Sie zum Ordner mit dem Code.

```
cd ~/raphael-kit/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.13_GAME_10Second_zero.py
```

Schütteln Sie den Zauberstab, die 4-stellige Segmentanzeige beginnt zu zählen, ein weiteres Schütteln stoppt die Zählung. Wenn Sie es schaffen, den angezeigten Zähler bei **10.00** zu halten, dann haben Sie gewonnen. Schütteln Sie es ein weiteres Mal, um die nächste Runde des Spiels zu starten.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `raphael-kit/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

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

from gpiozero import OutputDevice, Button
import time
import threading

# Initialisierung des Knopfes, verbunden mit GPIO 26
sensorPin = Button(26)

# Definition der GPIO-Pins, verbunden mit dem 74HC595 Schieberegister
SDI = OutputDevice(24)   # Serieller Dateneingang
RCLK = OutputDevice(23)  # Registeruhr
SRCLK = OutputDevice(18) # Schieberegisteruhr
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Definition der GPIO-Pins für die Ziffernauswahl auf der 7-Segment-Anzeige
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Definition der Segmentcodes für Zahlen 0 bis 9 auf der 7-Segment-Anzeige
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

# Zähler und Timer-Variablen
counter = 0
timer1 = None
gameState = 0

def clearDisplay():
    """ Alle Segmente auf der 7-Segment-Anzeige löschen. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ Verschiebt Daten in das 74HC595 Schieberegister, um eine Ziffer anzuzeigen. """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    """ Wählt aus, welche Ziffer auf der 7-Segment-Anzeige angezeigt werden soll. """
    for pin in placePin:
        pin.off()
    placePin[digit].on()

def display():
    """ Den aktuellen Zählerwert auf der 7-Segment-Anzeige anzeigen. """
    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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

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

def stateChange():
    """ Behandelt Zustandsänderungen des Zählers aufgrund von Knopfdrücken. """
    global gameState, counter, timer1
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState == 1 und timer1 ist nicht None:
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState + 1) % 2

def loop():
    """ Hauptloop, um Knopfdrücke zu prüfen und die Anzeige zu aktualisieren. """
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = sensorPin.value
        if (currentState == 0) und (lastState == 1):
            stateChange()
        lastState = currentState

def timer():
    """ Timer-Funktion, die den Zähler alle 0,01 Sekunden erhöht. """
    global counter, timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
    counter += 1

try:
    loop()
except KeyboardInterrupt:
    if timer1:
        timer1.cancel()
```

Code-Erklärung

1. Dieser Abschnitt importiert notwendige Module. Die Bibliothek „gpiozero“ wird verwendet, um mit GPIO-Geräten wie Tasten zu interagieren, und die Module „time“ und „threading“ können für zeitbezogene Aufgaben oder gleichzeitige Operationen verwendet werden.

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

   from gpiozero import OutputDevice, Button
   import time
   import threading
   ```

2. Initialisiert ein „Button“-Objekt aus der GPIO Zero-Bibliothek und verbindet es mit GPIO-Pin 26. Diese Konfiguration ermöglicht die Erkennung von Tastendrücken.

```
# Initialize the button connected to GPIO 26
sensorPin = Button(26)
```

3. Initialisiert GPIO-Pins, die mit den Eingängen für die serielle Dateneingabe (SDI), Registeruhr (RCLK) und Schieberegisteruhr (SRCLK) des Schieberegisters verbunden sind.

```
# Define GPIO pins connected to the 74HC595 shift register
SDI = OutputDevice(24)   # Serial Data Input
RCLK = OutputDevice(23)  # Register Clock
SRCLK = OutputDevice(18) # Shift Register Clock
```

4. Initialisiert die Pins für jede Ziffer der 7-Segment-Anzeige und definiert die Binärcodes für die Anzeige der Zahlen 0-9.

```
# Define GPIO pins for digit selection on the 7-segment display
placePin = [OutputDevice(pin) for pin in (10, 22, 27, 17)]

# Define segment codes for numbers 0 to 9 on the 7-segment display
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
```

5. Funktionen zur Steuerung der 7-Segment-Anzeige. „clearDisplay“ schaltet alle Segmente aus, „hc595_shift“ verschiebt Daten in das Schieberegister und „pickDigit“ aktiviert eine bestimmte Ziffer auf der Anzeige.

```
def clearDisplay():
    """ Clear all segments on the 7-segment display. """
    for _ in range(8):
        SDI.on()
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def hc595_shift(data):
    """ Shift data to the 74HC595 shift register to display a digit. """
    for i in range(8):
        SDI.value = 0x80 & (data << i)
        SRCLK.on()
        SRCLK.off()
    RCLK.on()
    RCLK.off()

def pickDigit(digit):
    """ Select which digit to display on the 7-segment display. """
    for pin in placePin:
        pin.off()
    placePin[digit].on()
```

6. Funktion zur Anzeige des aktuellen Zählerwerts auf der 7-Segment-Anzeige.

```
def display():
    """ Display the current counter value on the 7-segment display. """
    global counter
    clearDisplay()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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. Funktion zur Behandlung von Zustandsänderungen (Start/Stop) des Zählers basierend auf Tastendrücken.

```
def stateChange():
    """ Handle state changes for the counter based on button presses. """
    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
```

8. Hauptschleife, die kontinuierlich den Tastenzustand überprüft und die Anzeige aktualisiert. Sie ruft „stateChange“ auf, wenn sich der Tastenzustand ändert.

```
def loop():
    """ Main loop to check for button presses and update the display. """
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState = sensorPin.value
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState = currentState
```

9. Timer-Funktion, die den Zähler in regelmäßigen Abständen (alle 0,01 Sekunden) erhöht.

```
def timer():
    """ Timer function that increments the counter every 0.01 second. """
    global counter, timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
    counter += 1
```

10. Führt die Hauptschleife aus und ermöglicht einen sauberen Ausstieg aus dem Programm mit einem Tastaturinterrupt (Strg+C).

```
try:
    loop()
except KeyboardInterrupt:
    if timer1:
        timer1.cancel()
```

3.1.14 SPIEL – Not Not

Einführung

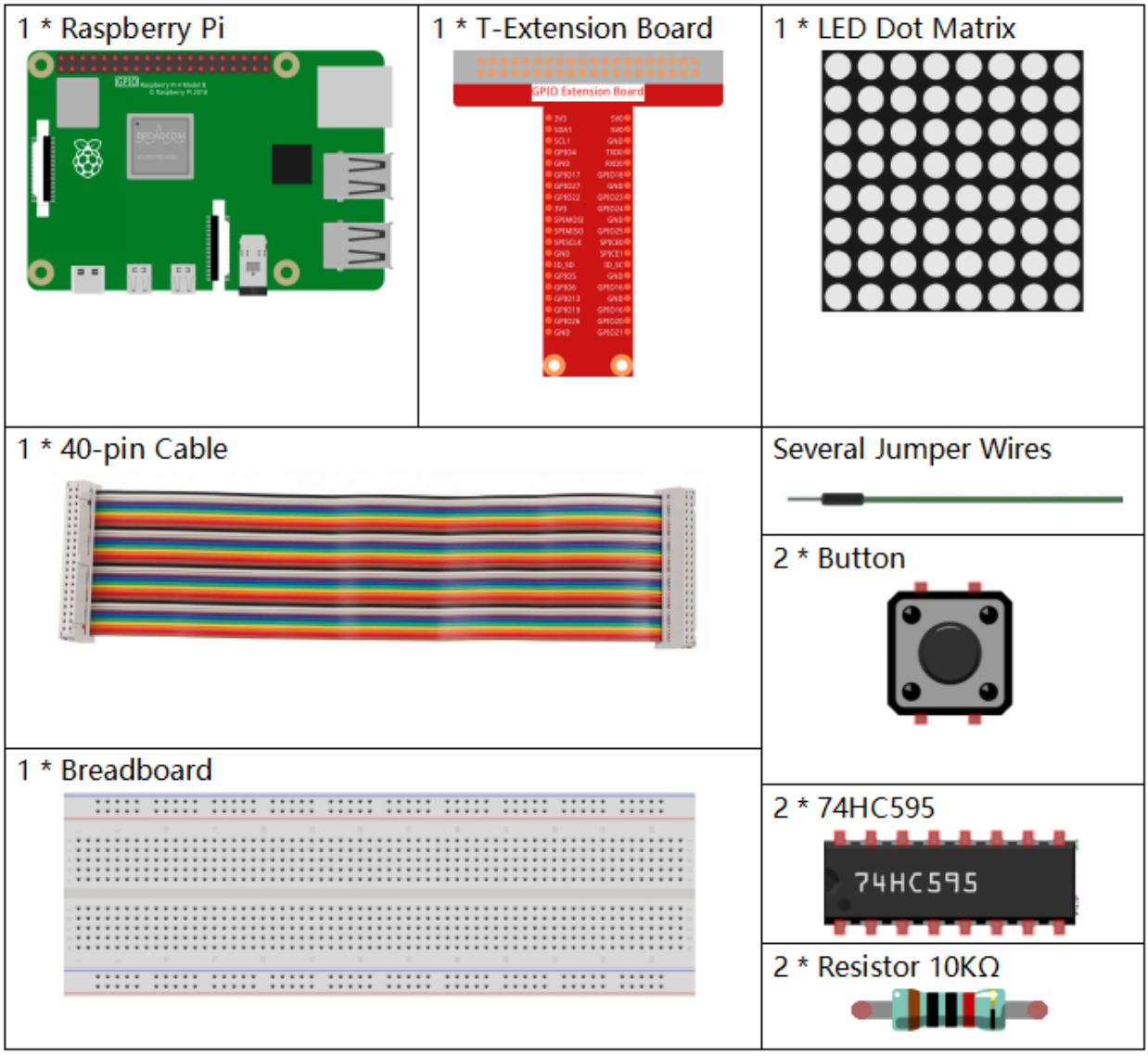
In dieser Lektion werden wir ein interessantes Spielgerät bauen, das wir „Not Not“ nennen.

Während des Spiels wird auf der Punkt-Matrix zufällig ein Pfeil angezeigt. Ihre Aufgabe ist es, innerhalb einer begrenzten Zeit die Taste in die entgegengesetzte Richtung des Pfeils zu drücken. Ist die Zeit abgelaufen oder wird die Taste in die gleiche Richtung wie der Pfeil gedrückt, sind Sie raus.

Dieses Spiel trainiert wirklich Ihr umgekehrtes Denken, und nun, wollen wir es ausprobieren?

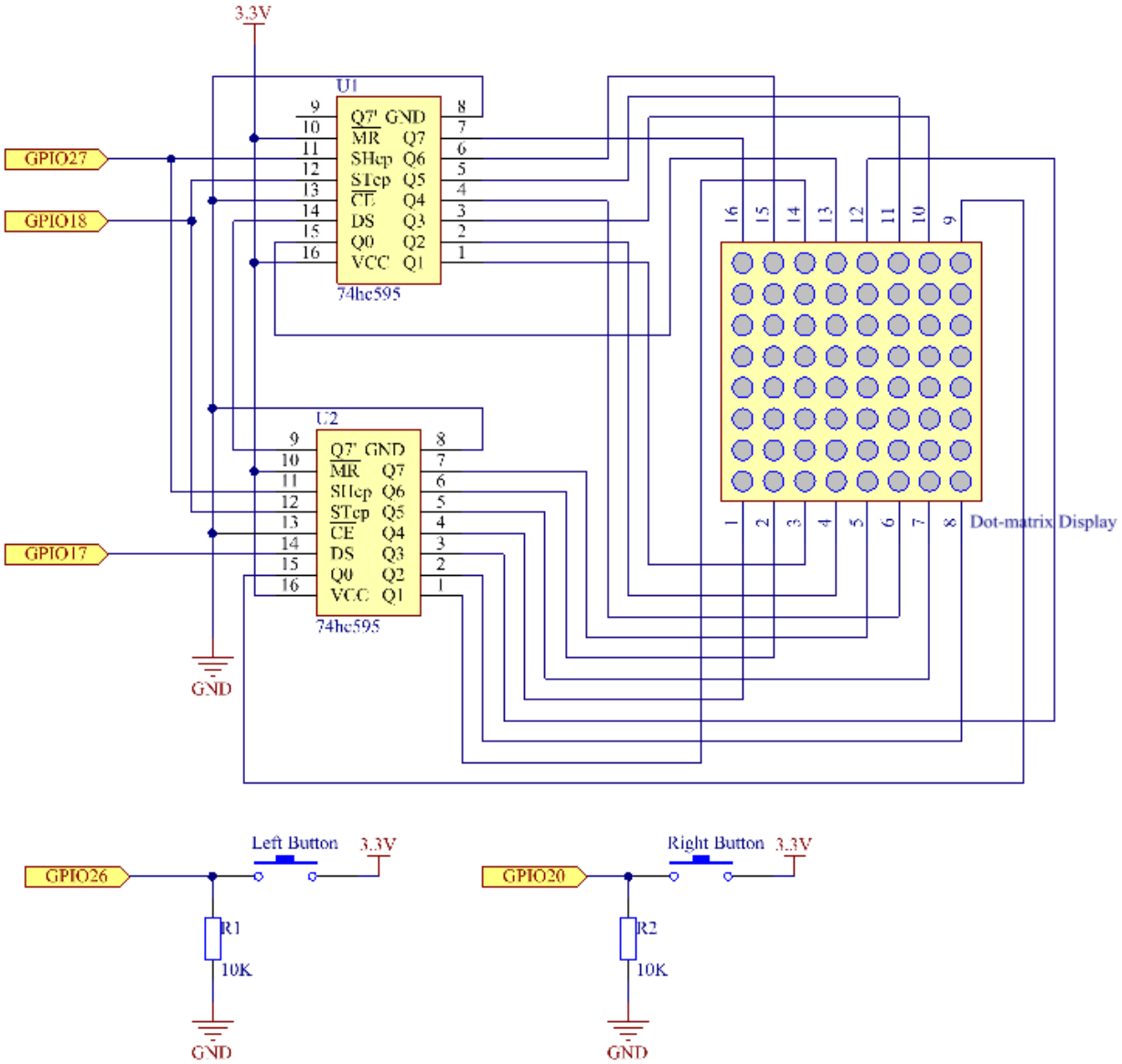
Benötigte Komponenten

Für dieses Projekt benötigen wir folgende Komponenten.



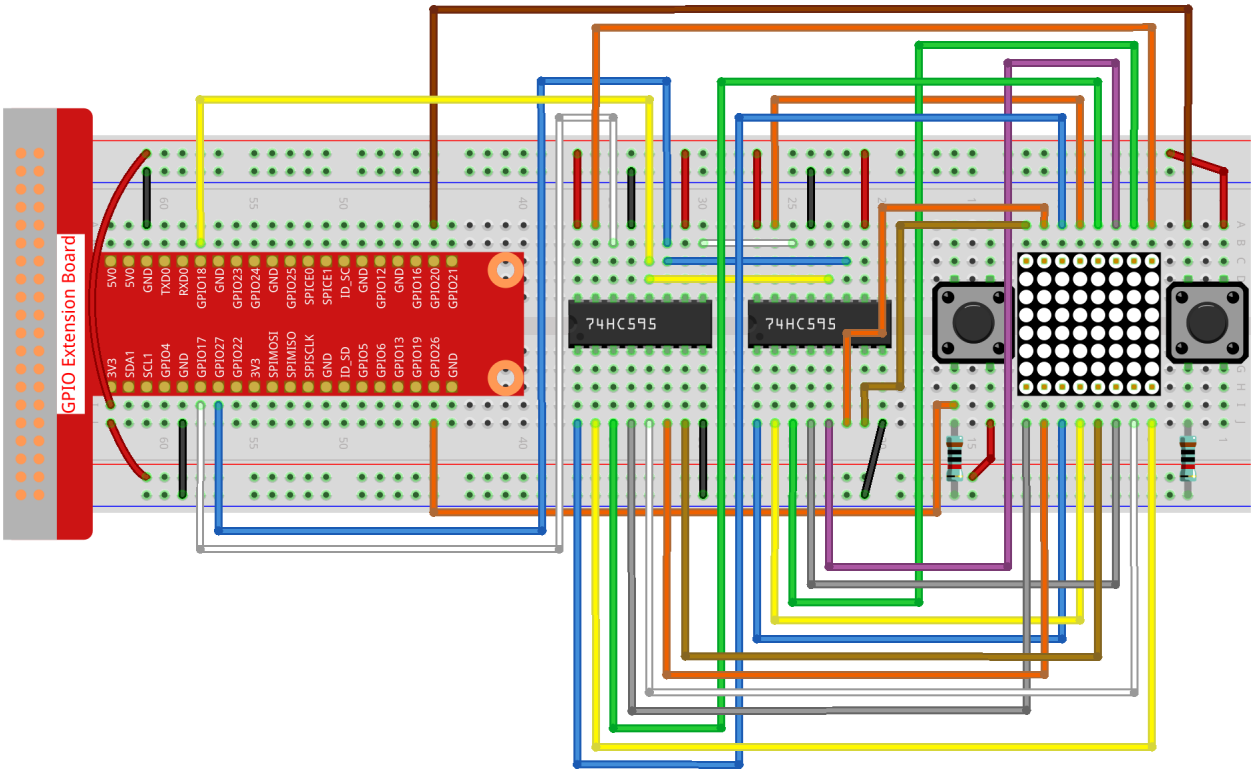
Schaltplan

T-Board Name	physisch	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



Experimentelle Verfahren

Schritt 1: Bauen Sie den Schaltkreis.



Schritt 2: Öffnen Sie die Code-Datei.

```
cd ~/davinci-kit-for-raspberry-pi/python-pi5
```

Schritt 3: Führen Sie die ausführbare Datei aus.

```
sudo python3 3.1.14_MotionControl_zero.py
```

Nachdem das Programm gestartet ist, erscheint auf der Punkt-Matrix ein Pfeil, der nach rechts oder links zeigt. Ihre Aufgabe ist es, innerhalb einer begrenzten Zeit die Taste in die entgegengesetzte Richtung des Pfeils zu drücken. Dann erscheint auf der Punkt-Matrix ein „“. Wenn die Zeit abgelaufen ist oder die Taste in die gleiche Richtung wie der Pfeil gedrückt wird, sind Sie raus, und die Punkt-Matrix zeigt ein „x“ an. Sie können auch 2 neue Tasten hinzufügen oder sie durch Joystick-Tasten für oben, unten, links und rechts – 4 Richtungen ersetzen, um den Schwierigkeitsgrad des Spiels zu erhöhen.

Code

Bemerkung: Sie können den untenstehenden Code **modifizieren/zurücksetzen/kopieren/ausführen/stoppen**. Bevor Sie dies tun, sollten Sie jedoch zum Quellcodepfad wie `davinci-kit-for-raspberry-pi/python-pi5` wechseln. Nachdem Sie den Code geändert haben, können Sie ihn direkt ausführen, um das Ergebnis zu sehen.

```
#!/usr/bin/env python3
from gpiozero import OutputDevice, Button
import time
import threading
import random

# GPIO-Pins für das 74HC595 Schieberegister
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
SDI = OutputDevice(17)   # Serieller Dateneingang
RCLK = OutputDevice(18)  # Registeruhr
SRCLK = OutputDevice(27) # Schieberegisteruhr

# GPIO-Pins für Tasten
AButtonPin = Button(20)  # Taste A
BButtonPin = Button(26)  # Taste B

# Initialisierung von Spielvariablen
timerPlay = 0
timerCheck = 0
waypoint = "NULL"
stage = "NULL"

# Pfeil-Glyphen für die LED-Matrix-Anzeige
arrow = {
    "right": [0xFF, 0xEF, 0xDF, 0x81, 0xDF, 0xEF, 0xFF, 0xFF],
    "left": [0xFF, 0xF7, 0xFB, 0x81, 0xFB, 0xF7, 0xFF, 0xFF]
}

# Feedback-Glyphen für richtige/falsche Antworten
check = {
    "wrong": [0xFF, 0xBB, 0xD7, 0xEF, 0xD7, 0xBB, 0xFF, 0xFF],
    "right": [0xFF, 0xFF, 0xF7, 0xEB, 0xDF, 0xBF, 0xFF, 0xFF]
}

def hc595_shift(dat):
    """ Verschiebt Daten in das 74HC595 Schieberegister. """
    for i in range(8):
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()

def display(glyphCode):
    """ Zeigt ein Glyph auf der LED-Matrix an. """
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80 >> i)
        RCLK.on()
        RCLK.off()

def creatGlyph():
    """ Erstellt ein neues Glyph für das Spiel und startet den Spiel-Timer. """
    global waypoint, stage, timerPlay
    waypoint = random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()

def checkPoint(inputKey):
    """ Überprüft die Eingabe des Spielers und aktualisiert den Spielstatus. """
    global waypoint, stage, timerCheck
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    if inputKey == "empty" oder inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()

def timeOut():
    """ Behandelt das Szenario eines Spielzeit-Überlaufs. """
    checkPoint("empty")

def getKey():
    """ Erkennt Tastendruck und löst Checkpoint aus. """
    if AButtonPin.is_pressed und nicht BButtonPin.is_pressed:
        checkPoint("right")
    elif nicht AButtonPin.is_pressed und BButtonPin.is_pressed:
        checkPoint("left")

def main():
    """ Hauptspielschleife. """
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])

def destroy():
    """ Räumt Ressourcen bei Programmende auf. """
    global timerPlay, timerCheck
    timerPlay.cancel()  # Spiel-Timer abbrechen
    timerCheck.cancel()  # Checkpoint-Timer abbrechen

# Spiel ausführen, KeyboardInterrupt für sauberen Ausstieg abfangen
try:
    main()
except KeyboardInterrupt:
    destroy()
```

Code-Erklärung

Basierend auf **1.1.6 LED-Punkt-Matrix** fügt diese Lektion **2** Tasten hinzu, um ein unterhaltsames Spielgerät zu machen. Wenn Sie also nicht sehr vertraut mit der Punkt-Matrix sind, beziehen Sie sich bitte auf *1.1.6 LED-Punktmatrix*.

1. Der Code beginnt mit dem Import der notwendigen Bibliotheken. „gpiozero“ wird für die Interaktion mit GPIO-Pins wie Tasten und Ausgabegeräten verwendet. „time“ ermöglicht das Hinzufügen von Verzögerungen, „threading“ ermöglicht das gleichzeitige Ausführen mehrerer Aufgaben und „random“ ist nützlich, um Zufälligkeit im Projekt einzuführen.

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
from gpiozero import OutputDevice, Button
import time
import threading
import random
```

2. Initialisiert GPIO-Pins für das Schieberegister („SDI“, „RCLK“, „SRCLK“) und Tasten („AButtonPin“, „BButtonPin“). Das Schieberegister wird verwendet, um mehrere LEDs mit weniger GPIO-Pins zu steuern, was für die LED-Matrixanzeige entscheidend ist.

```
# GPIO-Pins für das 74HC595 Schieberegister
SDI = OutputDevice(17)   # Serieller Dateneingang
RCLK = OutputDevice(18)  # Registeruhr
SRCLK = OutputDevice(27) # Schieberegisteruhr

# GPIO-Pins für Tasten
AButtonPin = Button(20)  # Taste A
BButtonPin = Button(26)  # Taste B
```

3. Initialisiert Variablen, die in der Spiellogik verwendet werden, wie Timer und Spielzustandsindikatoren.

```
# Spielvariablen-Initialisierung
timerPlay = 0
timerCheck = 0
waypoint = "NULL"
stage = "NULL"
```

4. Definiert binäre Muster zur Anzeige von Pfeilen und Feedback (richtig/falsch) auf der LED-Matrix. Jedes Array-Element repräsentiert eine Reihe der LED-Matrix, wobei „1“ und „0“ entsprechend bedeuten, dass eine LED an oder aus ist.

```
# Pfeil-Glyphen für die LED-Matrixanzeige
arrow = {
    "right": [0xFF, 0xEF, 0xDF, 0x81, 0xDF, 0xEF, 0xFF, 0xFF],
    "left": [0xFF, 0xF7, 0xFB, 0x81, 0xFB, 0xF7, 0xFF, 0xFF]
}

# Feedback-Glyphen für richtige/falsche Antworten
check = {
    "wrong": [0xFF, 0xBB, 0xD7, 0xEF, 0xD7, 0xBB, 0xFF, 0xFF],
    "right": [0xFF, 0xFF, 0xF7, 0xEB, 0xDF, 0xBF, 0xFF, 0xFF]
}
```

5. Diese Funktion verschiebt ein Byte Daten in das 74HC595 Schieberegister. Sie iteriert über jedes Bit des „dat“-Bytes, setzt den „SDI“-Pin entsprechend hoch oder niedrig und toggelt den „SRCLK“-Pin, um das Bit in das Register zu schieben.

```
def hc595_shift(dat):
    """ Daten in das 74HC595 Schieberegister verschieben. """
    for i in range(8):
        SDI.value = 0x80 & (dat << i)
        SRCLK.on()
        SRCLK.off()
```

6. Diese Funktion zeigt ein Glyph auf der LED-Matrix an. Sie sendet jede Reihe des Glyphs (repräsentiert durch

„glyphCode“) und die Adresse der Reihe an das Schieberegister mit „hc595_shift“ und toggelt dann den „RCLK“-Pin, um die Anzeige zu aktualisieren.

```
def display(glyphCode):
    """ Ein Glyph auf der LED-Matrix anzeigen. """
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80 >> i)
        RCLK.on()
        RCLK.off()
```

7. Diese Funktion wählt zufällig ein Glyph aus dem „arrow“-Wörterbuch aus, startet den Spiel-Timer und setzt die Spielphase auf „PLAY“. Der „threading.Timer“ wird für die Zeitsteuerung im Spiel verwendet.

```
def creatGlyph():
    """ Ein neues Glyph für das Spiel erstellen und den Spiel-Timer starten. """
    global waypoint, stage, timerPlay
    waypoint = random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()
```

8. Diese Funktion überprüft die Eingabe des Spielers gegenüber dem aktuellen Glyph. Ist die Eingabe korrekt, wird das Wegpunkt auf „right“ gesetzt, andernfalls auf „wrong“. Anschließend wird der aktuelle Spiel-Timer abgebrochen und ein neuer Timer für das nächste Glyph gestartet.

```
def checkPoint(inputKey):
    """ Spieler-Eingabe überprüfen und Spielstatus aktualisieren. """
    global waypoint, stage, timerCheck
    if inputKey == "empty" oder inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()
```

9. Diese Funktion wird aufgerufen, wenn das Spiel zeitlich abläuft. Sie ruft „checkPoint“ mit „empty“ auf, um anzuzeigen, dass keine Taste rechtzeitig gedrückt wurde.

```
def timeOut():
    """ Szenario eines Spielzeit-Überlaufs behandeln. """
    checkPoint("empty")
```

10. Diese Funktion überprüft den Zustand der Tasten. Wenn „AButtonPin“ gedrückt wird (und „BButtonPin“ nicht), ruft sie „checkPoint“ mit „right“ auf. Wenn „BButtonPin“ gedrückt wird (und „AButtonPin“ nicht), ruft sie „checkPoint“ mit „left“ auf.

```
def getKey():
    """ Tastendruck erkennen und Checkpoint auslösen. """
    if AButtonPin.is_pressed und nicht BButtonPin.is_pressed:
        checkPoint("right")
    elif nicht AButtonPin.is_pressed und BButtonPin.is_pressed:
        checkPoint("left")
```

11. Die „main“-Funktion steuert den Spielablauf. Sie beginnt mit der Erstellung eines Glyphs und überprüft kontinuierlich die Spielphase. In der Phase „PLAY“ wird das aktuelle Glyph angezeigt und auf Tastendrücke geprüft. In der Phase „CHECK“ wird das Feedback basierend auf der Aktion des Spielers angezeigt.

```
def main():
    """ Hauptspiel-Schleife. """
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])
```

12. Diese Funktion bricht alle laufenden Timer ab, wenn das Programm beendet wird, um einen sauberen Abschluss zu gewährleisten.

```
def destroy():
    """ Ressourcen bei Programmende aufräumen. """
    global timerPlay, timerCheck
    timerPlay.cancel()  # Spiel-Timer abbrechen
    timerCheck.cancel()  # Checkpoint-Timer abbrechen
```

13. Das Spiel wird in einem „try“-Block ausgeführt. Tritt eine „KeyboardInterrupt“ (wie das Drücken von Strg+C) auf, fängt es die Ausnahme ab und ruft „destroy“ auf, um vor dem Beenden aufzuräumen.

```
# Spiel ausführen, KeyboardInterrupt für sauberen Ausstieg abfangen
try:
    main()
except KeyboardInterrupt:
    destroy()
```

1.12 Spielen Sie mit der Processing (nicht für Pi 5)

1.12.1 Was ist Processing?

Processing ist eine einfache Programmierumgebung, die entwickelt wurde, um die Entwicklung visuell orientierter Anwendungen mit Schwerpunkt auf Animation zu erleichtern und Benutzern durch Interaktion sofortiges Feedback zu geben. Die Entwickler wollten ein Mittel, um Ideen im Code zu „skizzieren“. Da sich seine Fähigkeiten in den letzten zehn Jahren erweitert haben, wird Processing neben seiner Skizzierfunktion auch für anspruchsvollere Arbeiten auf Produktionsebene eingesetzt. Ursprünglich als domänenspezifische Erweiterung für Java für Künstler und Designer entwickelt, hat sich Processing zu einem ausgewachsenen Design- und Prototyping-Tool entwickelt, das für groß angelegte Installationsarbeiten, Motion Graphics und komplexe Datenvisualisierungen verwendet wird.

Processing basiert auf Java, aber da die Programmelemente in Processing recht einfach sind, können Sie es auch ohne Java-Kenntnisse erlernen. Wenn Sie mit Java vertraut sind, vergessen Sie am besten, dass Processing für eine Weile etwas mit Java zu tun hat, bis Sie sich mit der Funktionsweise der API vertraut gemacht haben.

Dieser Text stammt aus dem Tutorial Processing Overview.

1.12.2 Installieren Sie die Processing

Bemerkung: Bevor Sie Processing verwenden können, müssen Sie aus der Ferne auf den Raspberry Pi-Desktop zugreifen (*Remotedesktop*) oder ein Display für den Raspberry Pi anschließen.

Besuchen Sie zuerst https://processing.org/download und wählen Sie die Version „Linux ARM32-bit“ aus

Sie laden dann in eine `.tar.gz`-Datei herunter, die den meisten Linux-Benutzern bekannt sein sollte. Laden Sie die Datei in Ihr Home-Verzeichnis herunter, öffnen Sie dann ein Terminalfenster und geben Sie Folgendes ein:

```
tar xvfz processing-xxxx.tgz
```

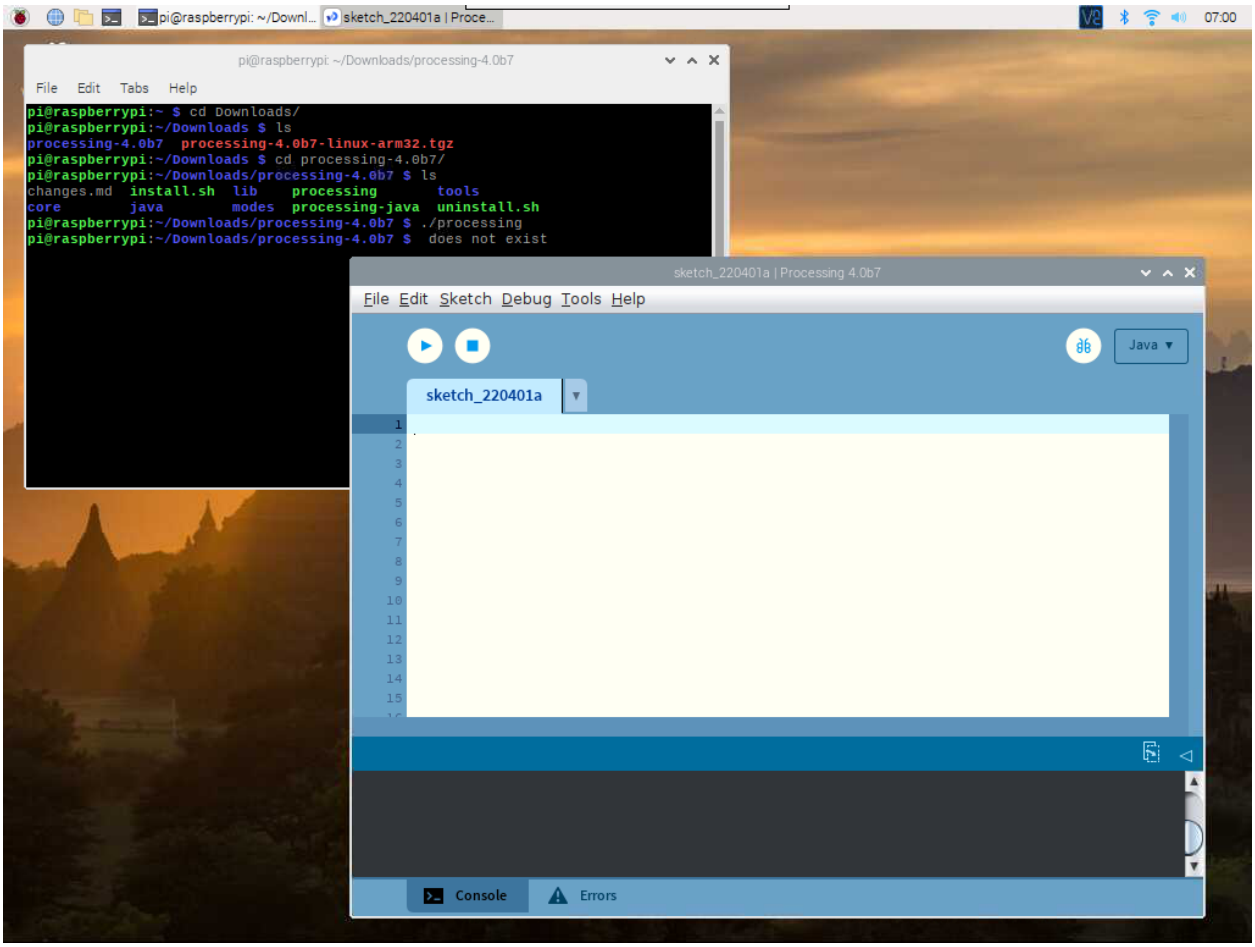
(Ersetzen Sie xxxx durch den Rest des Dateinamens, also die Versionsnummer.) Dadurch wird ein Ordner mit dem Namen processing-2.0 oder ähnlich erstellt. Wechseln Sie dann in dieses Verzeichnis:

```
cd processing-xxxx
```

und führe es aus:

```
./processing
```

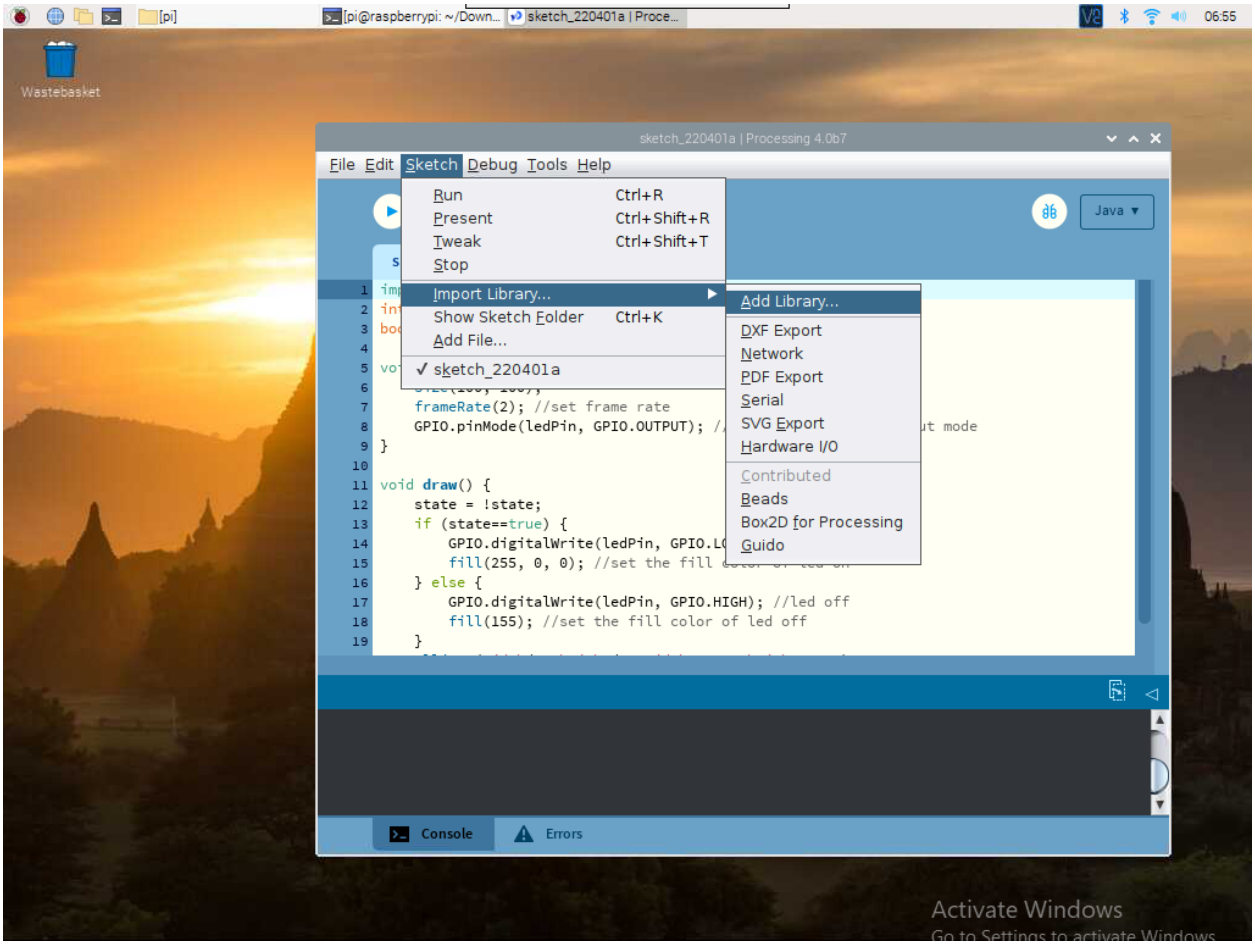
Mit etwas Glück wird nun das Hauptverarbeitungsfenster sichtbar.



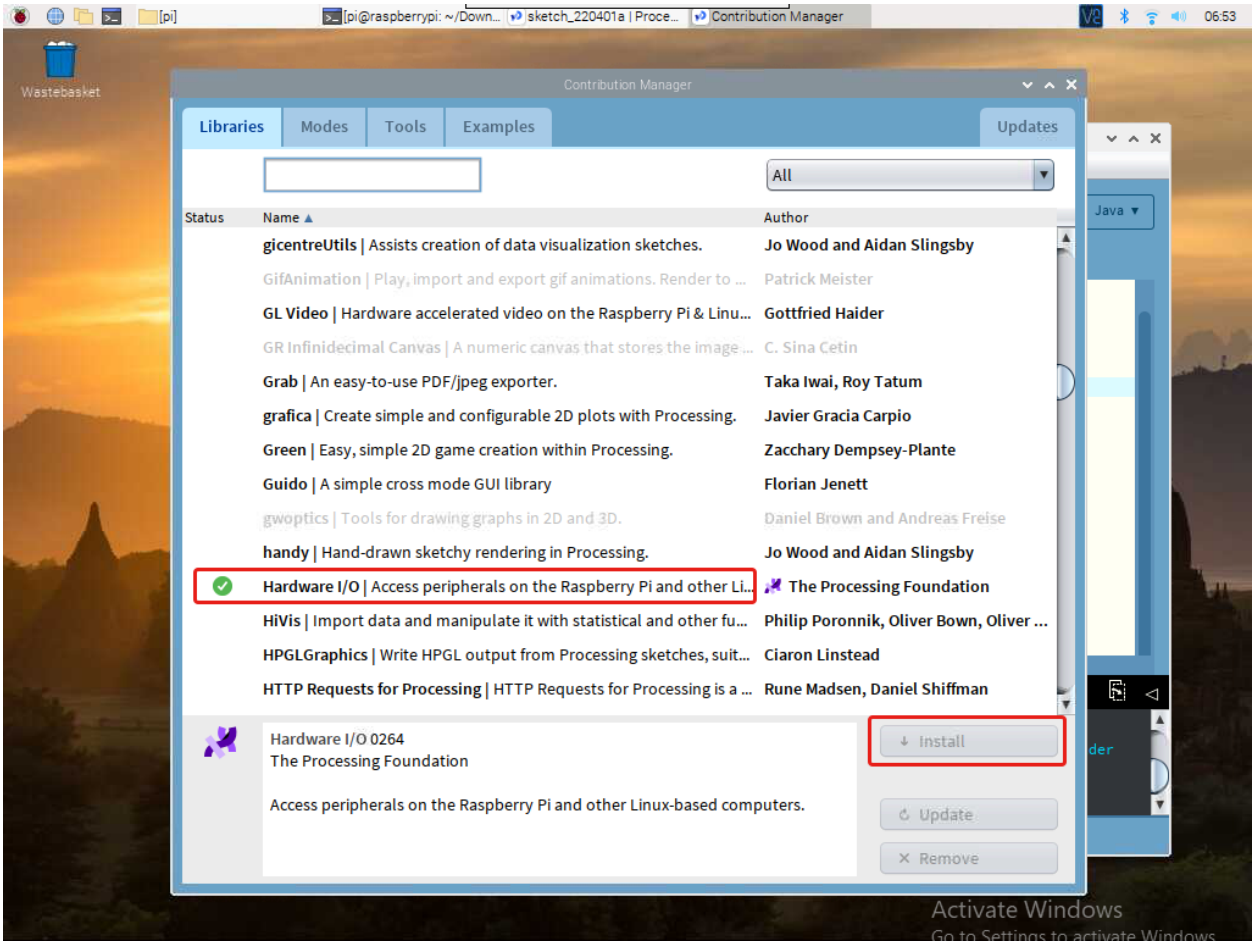
1.12.3 Hardware-I/O installieren

Um den GPIO des Raspberry Pi zu verwenden, müssen Sie manuell eine Hardware-I/O-Bibliothek hinzufügen.

Klicken `Sketch` -> `Import Library` -> `Add Library...`



Suchen Sie Hardware I/O , wählen Sie es aus und klicken Sie dann auf Installieren. Wenn Sie fertig sind, erscheint ein Häkchen-Symbol.



1.12.4 Projekte

Zeichne einen Matchman

Sie führen jetzt die Processing Development Environment (oder PDE) aus. Es ist nicht viel dabei; der große Bereich ist der Texteditor, und oben gibt es eine Reihe von Schaltflächen; Dies ist die Symbolleiste. Unterhalb des Editors befindet sich der Nachrichtenbereich und darunter die Konsole. Der Nachrichtenbereich wird für einzeilige Nachrichten verwendet, und die Konsole wird für weitere technische Details verwendet.

Machen wir uns mit der Verwendung von Processing vertraut und ziehen wir einen Matchman.

Skizzieren

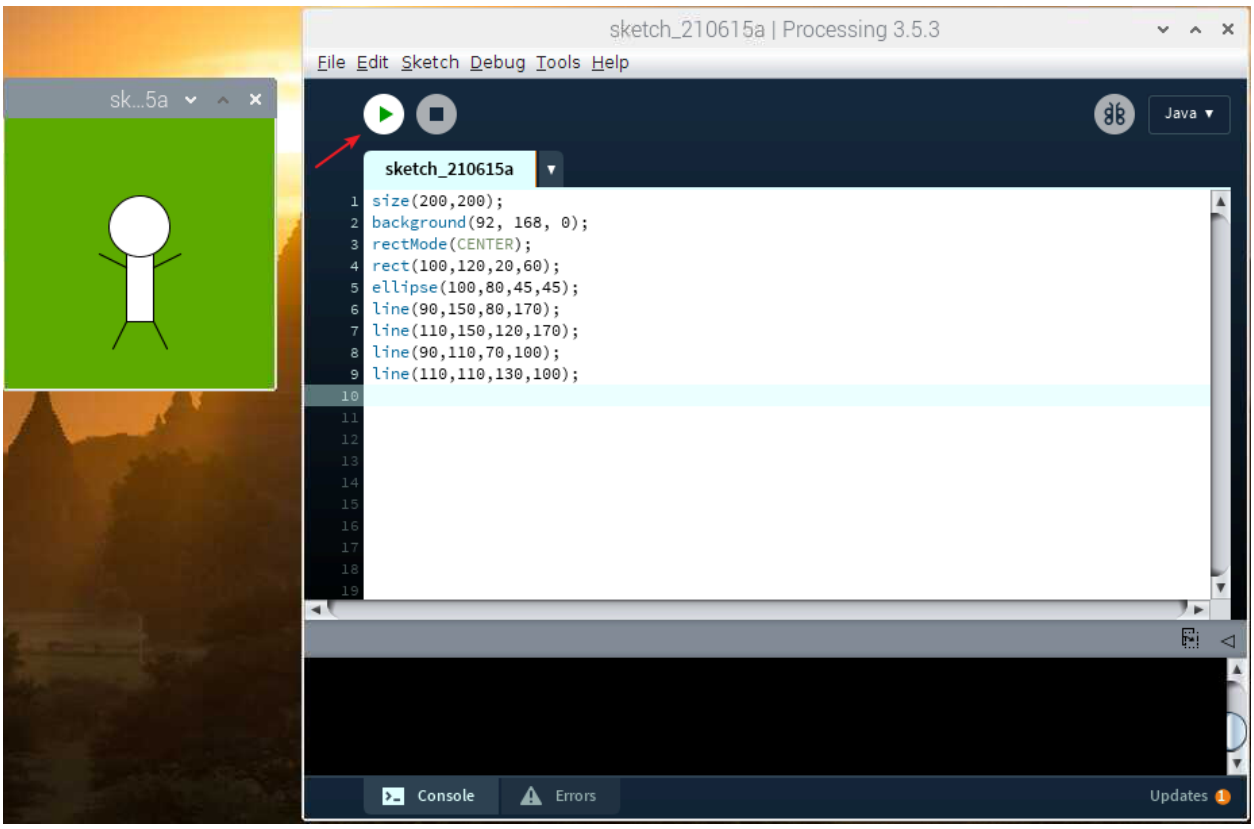
Kopieren Sie die folgende Skizze in Processing und führen Sie sie aus. Ein neues Anzeigefenster erscheint und ein jubelnder Matchmaker wird ausgelost.

```
size(200,200);
background(92, 168, 0);
rectMode(CENTER);
rect(100,120,20,60);
ellipse(100,80,45,45);
line(90,150,80,170);
line(110,150,120,170);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
line(90,110,70,100);
line(110,110,130,100);
```



Bemerkung: Wenn Sie es ausführen und der Nachrichtenbereich rot wird und einige Fehler melden, dann stimmt etwas mit der Skizze nicht. Stellen Sie sicher, dass Sie die Beispielskizze genau kopieren: Zahlen sollten in Klammern eingeschlossen werden, mit Kommas zwischen jeder Zahl, und Zeilen sollten mit Semikolon enden.

Wie es funktioniert?

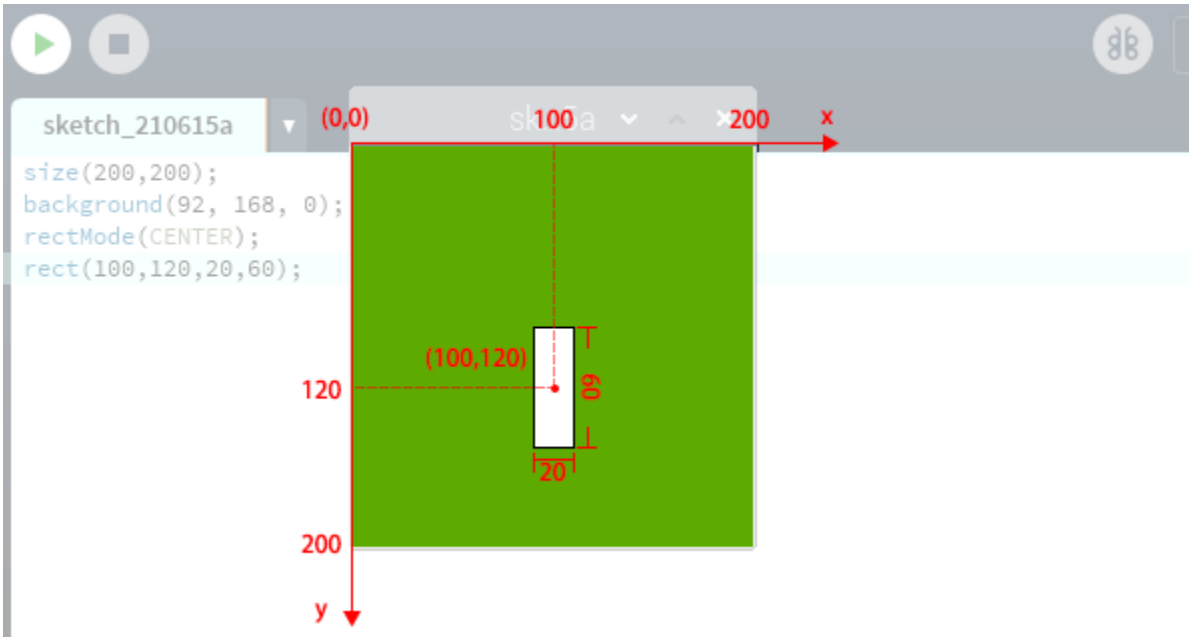
Der Schlüssel hier ist zu erkennen, dass das Anzeigefenster wie ein Quadrat aus Papier behandelt werden kann.

Jedes Pixel des Anzeigefensters ist eine Koordinate `(x,y)` die die Position eines Punktes im Raum bestimmt. Der Ursprung `(0,0)` der Koordinaten liegt in der oberen linken Ecke, die positive Richtung der X-Achse liegt horizontal nach rechts und die positive Richtung der Y-Achse ist vertikal nach unten.

Was wir tun müssen, ist anzugeben, welche Form und Farbe an diesen Pixelkoordinaten erscheinen soll.

Zeichnen Sie beispielsweise ein Rechteck der Breite 20 und Höhe 60 mit den Koordinate `(100,120)` als Mittelpunkt.

```
rectMode(CENTER);
rect(100,120,20,60);
```



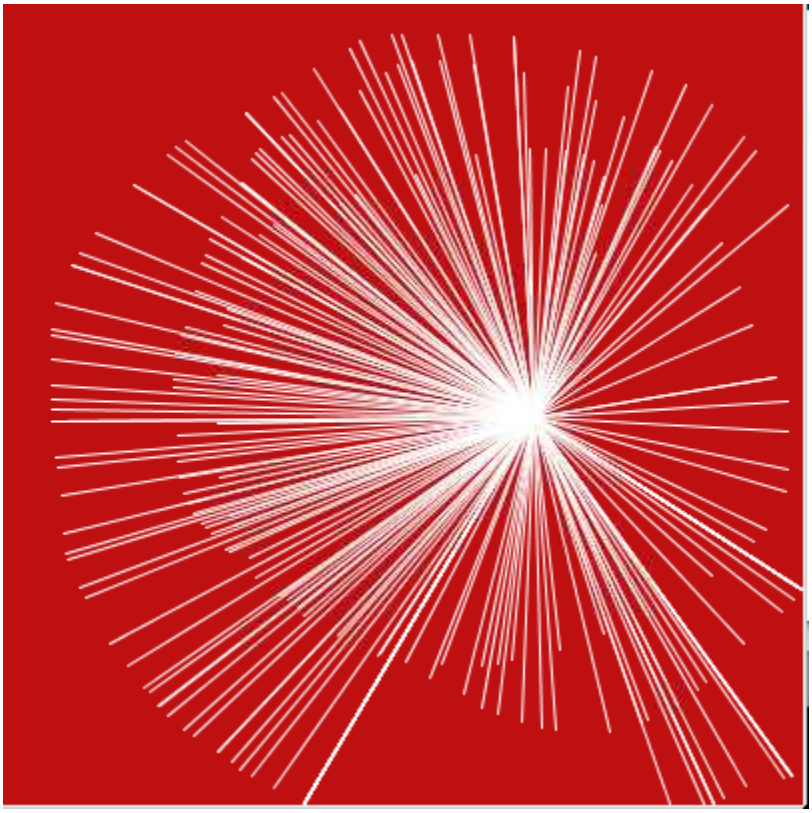
Sobald wir die Beziehung zwischen dem Anzeigefenster und den Achsen verstanden haben, fällt uns diese Skizze nicht schwer, wir müssen nur einige einfache grafische Zeichenanweisungen verstehen.

- `size(width, height)` : Definiert die Dimension der Breite und Höhe des Anzeigefensters in Pixeleinheiten.
- `background(red, green, blue)` : Stellen Sie die Hintergrundfarbe des Anzeigefensters ein.
- `rectMode(mode)` : Ändert die Position, von der aus Rechtecke gezeichnet werden, indem die Art und Weise geändert wird, wie die an `rect()` übergebenen Parameter interpretiert werden.
- `rect(x, y, width, height)` : Zeichnet ein Rechteck auf den Bildschirm.
- `ellipse(x, y, width, height)` : Zeichnet eine Ellipse (oval) auf den Bildschirm.
- `line(x1, y1, x2, y2)` : Zeichnet eine Linie (einen direkten Pfad zwischen zwei Punkten) zum Bildschirm.

Weitere Informationen finden Sie unter Processing Reference.

Hallo Maus

In diesem Projekt schießt Ihre Maus weiterhin Linien auf einen Punkt; Bewegen Sie die Maus und Sie werden eine einzigartige Sternenlinie zeichnen. Drücken Sie die Maus, um die Zeichnung neu zu starten.



Skizzieren

```
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;
    pointY=mouseY;
    background(192, 16, 18);
}
```

Wie es funktioniert?

Das vorherige Projekt zeichnete ein einzelnes Bild ohne Animation oder Interaktion.

Wenn wir eine interaktive Skizze erstellen möchten, müssen wir die Funktionen `setup()` und `draw()` (dies sind integrierte Funktionen, die automatisch aufgerufen werden) hinzufügen, um den Rahmen zu erstellen.

- `setup()`: Wird nur einmal zu Beginn des Sketches ausgeführt.

- `draw()`: Wird wiederholt ausgeführt, wobei wir normalerweise die Skizze zum Zeichnen der Animation hinzufügen.

```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}

void draw() {
    line(pointX, pointY, mouseX, mouseY);
}
```

Diese obige Skizze funktioniert bereits reibungslos als interaktive Skizze.

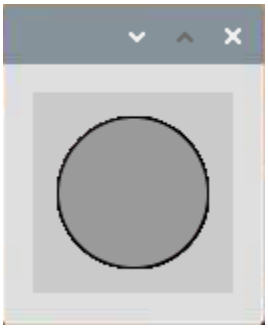
Als nächstes können Sie ein Mausklick-Ereignis hinzufügen. Dieses Ereignis kann mit der Funktion `mousePressed()` implementiert werden, wo wir Anweisungen hinzufügen, um den Zielpunkt zu aktualisieren und den Bildschirm zu leeren.

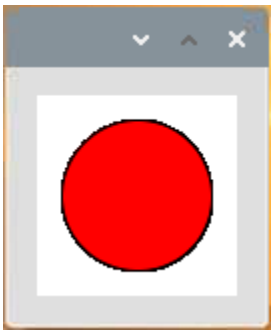
```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}

void draw() {
    line(pointX, pointY, mouseX, mouseY);
}

void mousePressed() {
    pointX=mouseX;
    pointY=mouseY;
    background(192, 16, 18);
}
```

Weitere Informationen finden Sie unter Processing Reference.

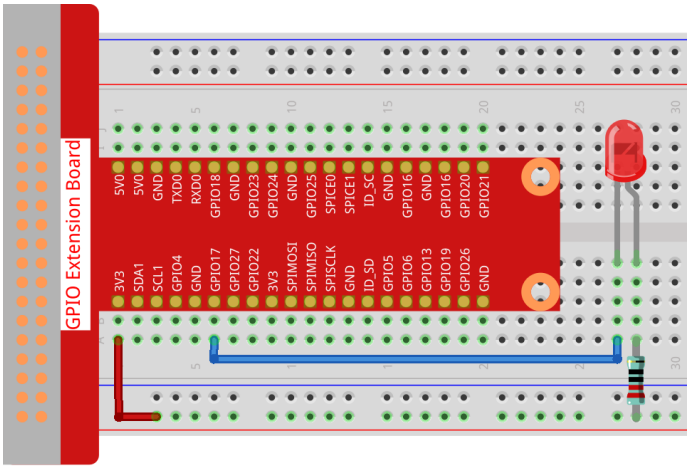
Blinkender Punkt

In diesem Projekt werden wir einen Punkt auf Processing zeichnen, der synchron mit der LED blinkt. Bitte bauen Sie die Schaltung wie im Diagramm gezeigt auf und führen Sie die Skizze aus.





Verdrahtung



Skizzieren

```
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}

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);
}
```

Wie es funktioniert?

Am Anfang der Skizze müssen Sie die GPIO-Funktionsbibliothek von Processing durch `import processing.io.*;` einbetten, was für Schaltungsexperimente unverzichtbar ist.

Framerate ist die Frequenz der auf der Platine erscheinenden Bitmaps, ausgedrückt in Hertz (Hz). Mit anderen Worten, es ist auch die Häufigkeit, mit der die Funktion `draw()` aufgerufen wird. Wenn Sie in `setup()` die **Framerate** auf 2 setzen, wird `draw()` alle 0,5s aufgerufen.

Jeder Aufruf der Funktion `draw()` nimmt die Umkehrung von `state` und bestimmt diese anschließend. Wenn der Wert `true` ist, leuchtet die LED und der Pinsel ist rot gefüllt; wenn nicht, wird die LED ausgeschaltet und der Pinsel wird mit Grau gefüllt.

Verwenden Sie nach Abschluss der Beurteilung die Funktion `ellipse()` , um einen Kreis zu zeichnen. Es sollte beachtet werden, dass `width` und `height` Systemvariablen sind, die verwendet werden, um die Breite und Höhe des Anzeigefensters zu speichern.

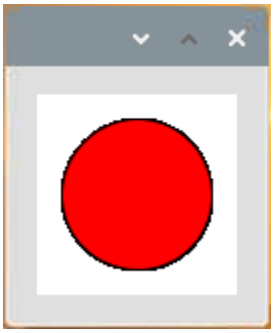
Es sind noch zwei weitere Punkte zu beachten. Wenn Sie GPIOs verwenden, müssen Sie die Funktion `GPIO.pinMode()` verwenden, um den INPUT/OUTPUT-Zustand des Pins festzulegen, und dann die Funktion `GPIO.digitalWrite()` verwenden, um einen Wert (HIGH) // (LOW) an den Stift .

Bemerkung: Bitte vermeiden Sie die Verwendung von `delay()` in `draw()` , da dies die Aktualisierung des Anzeigefensters beeinflusst.

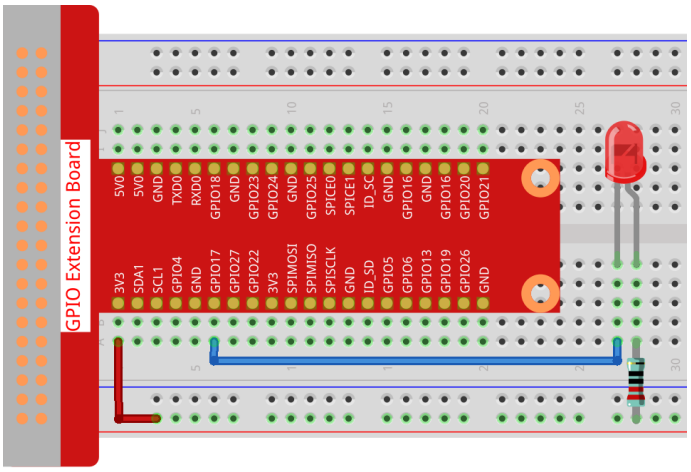
Weitere Informationen finden Sie unter Processing Reference.

Anklickbarer Punkt

Wir haben versucht, Animationen zu zeichnen, auf Mausereignisse zu reagieren und die LED zu steuern. Wir können diese Funktionen also genauso gut kombinieren, einen anklickbaren Punkt zeichnen, um die LED zu steuern!



Verdrahtung



Skizzieren

```
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);
}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
void mouseClicked() {
    //  toggles state:
    if (2*dist(mouseX,mouseY,width/2, height/2)<=width*0.75)
        {state = !state;}
}
```

Wie es funktioniert?

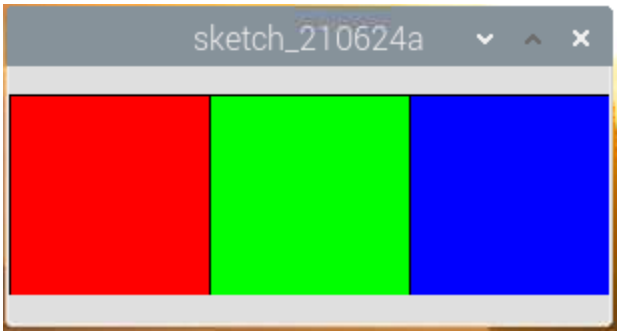
Dieses Projekt hat viel mit *Blinkender Punkt* gemeinsam, der Unterschied besteht darin, dass es den Toggle-Status in das Mausereignis versetzt. Dadurch blinkt die LED nicht automatisch, sondern leuchtet auf und erlischt mit einem Mausklick.

Und im Ereignis `mouseClicked()` wird die Funktion `dist()` verwendet, um die Position der Maus zum Zeitpunkt des Klickens zu bestimmen, und der Punkt wird nur dann als angeklickt betrachtet, wenn der Abstand zwischen der Maus und der Mittelpunkt des Punktes ist kleiner als der Radius.

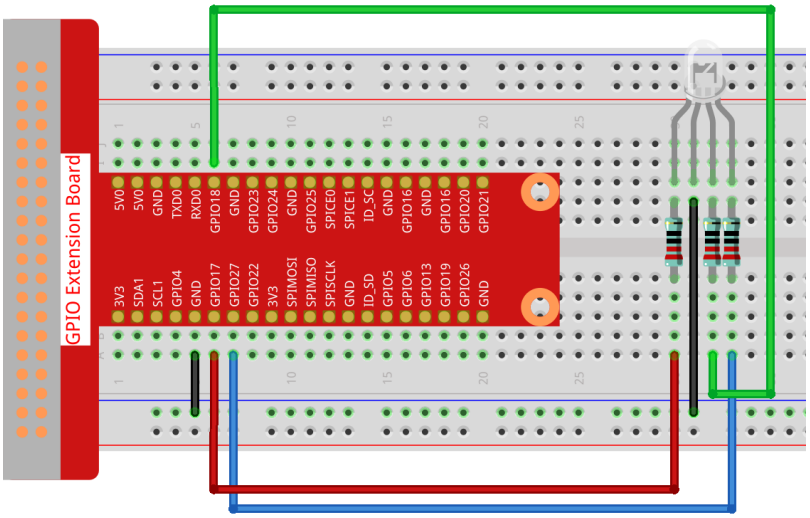
Weitere Informationen finden Sie unter Processing Reference.

Anklickbare Farbblöcke

Wir haben bereits versucht, einen anklickbaren Punkt zu zeichnen, um die LED zu steuern, also gehen wir noch einen Schritt weiter und zeichnen 3 farbige Quadrate, um die RGB-Farben anzupassen!



Verdrahtung



Skizzieren

```
import processing.io.*; // use the GPIO library

int[] pins = { 17, 18, 27 };

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);
    }
}
```

Wie es funktioniert?

Dieses Projekt hat viel mit *Anklickbarer Punkt* gemeinsam, außer dass es die Bedingungen für die Bestimmung des Mausklick-Ereignisses verfeinert.

Zeichnen Sie zuerst drei Farbblöcke in `draw()` , ermitteln Sie dann, welcher Farbblock basierend auf dem Wert von mouseX (der X-Achsen-Koordinate der Maus) angeklickt wurde, und lassen Sie schließlich RGB die entsprechende Farbe aufleuchten.

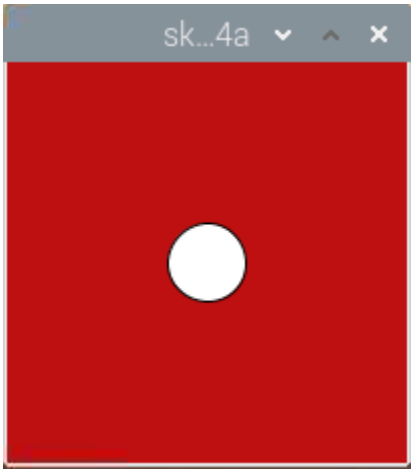
Was mehr?

Basierend auf der Zugabe von Licht können wir eine siebenfarbige RGB-LED-Anzeige erstellen - das Hinzufügen von Rot zu Grün erzeugt Gelb; Wenn man alle drei Primärfarben zusammen addiert, erhält man Weiß. Jetzt können Sie es selbst ausprobieren.

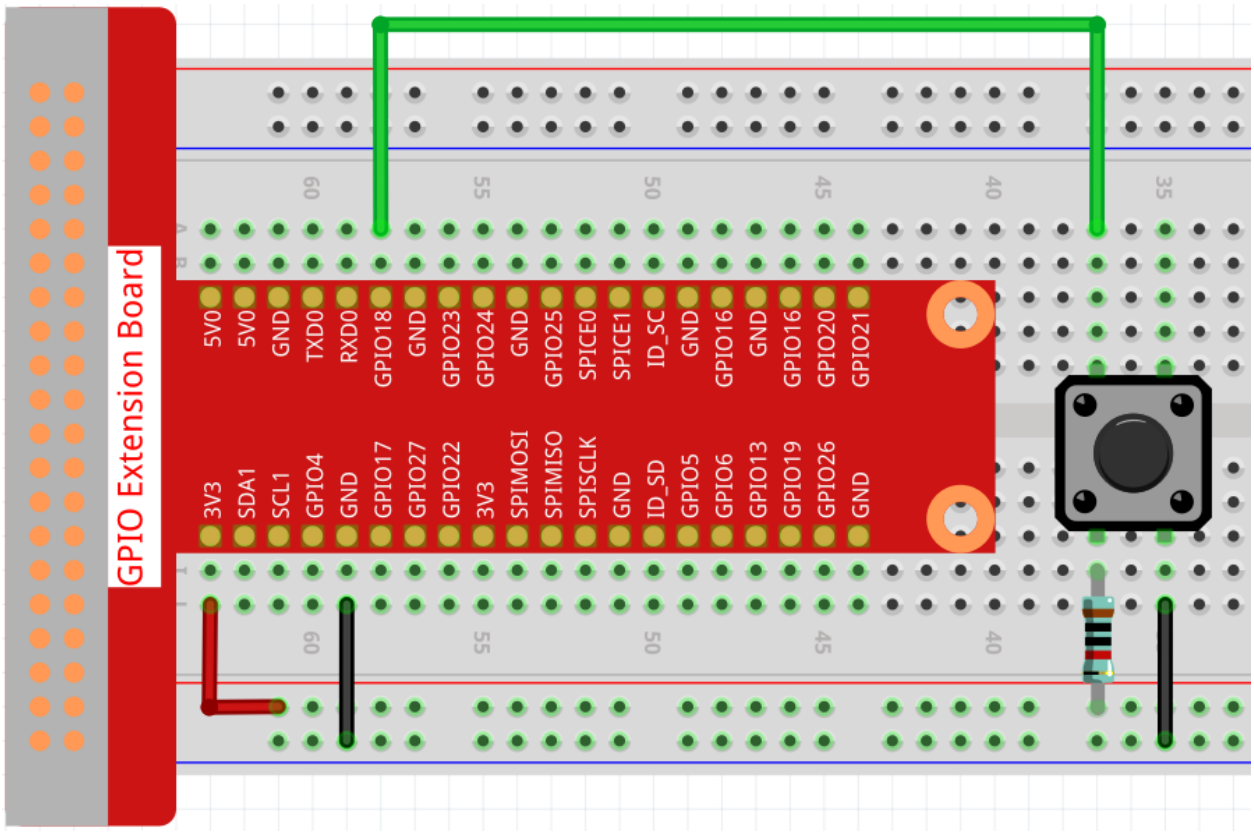
Weitere Informationen finden Sie unter Processing Reference.

Aufblasen des Dot

Als nächstes bauen wir eine Schaltung, die es der Schaltfläche ermöglicht, die Größe des Punkts zu steuern. Wenn wir die Taste drücken, wird der Punkt schnell größer; Wenn wir die Taste loslassen, wird der Punkt allmählich kleiner, wodurch der Punkt wie ein aufgeblasener Ballon aussieht.



Verdrahtung



Skizzieren

```
import processing.io.*;
int buttonPin = 18;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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--;}
    }
    background(192, 16, 18);
    ellipse(width/2, height/2,diameter, diameter);
}
```

Wie es funktioniert?

Dieses Projekt verwendet die Eingabefunktion im Vergleich zu den vorherigen 2 Projekten, die die Ausgabefunktion des GPIO verwendet haben.

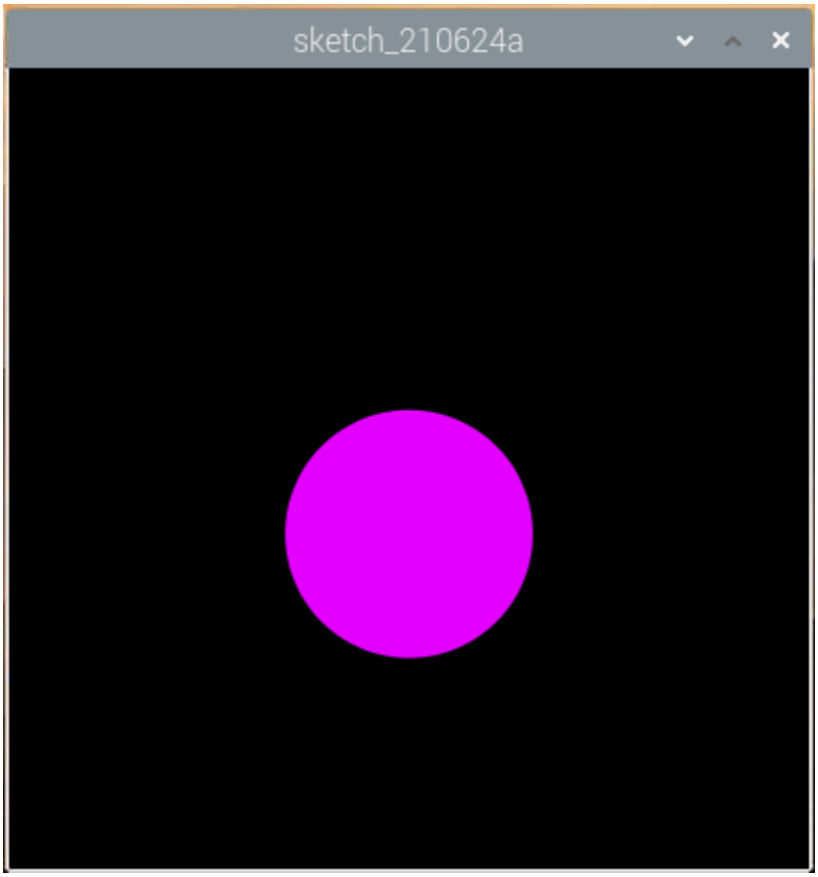
Die Funktion `GPIO.pinMode()` wird verwendet, um `buttonPin` in den Pull-Up-Eingangsmodus zu setzen, wodurch der Pin im Standardzustand automatisch hoch wird.

Verwenden Sie dann die Funktion `GPIO.digitalRead()` , um den Wert von `buttonPin` auszulesen. Wenn der Wert NIEDRIG ist, bedeutet dies, dass die Taste gedrückt wird. Lassen Sie dann den Durchmesser des Punkts um 5 zunehmen; Wird die Taste losgelassen, verringert sich der Durchmesser des Punktes um 1.

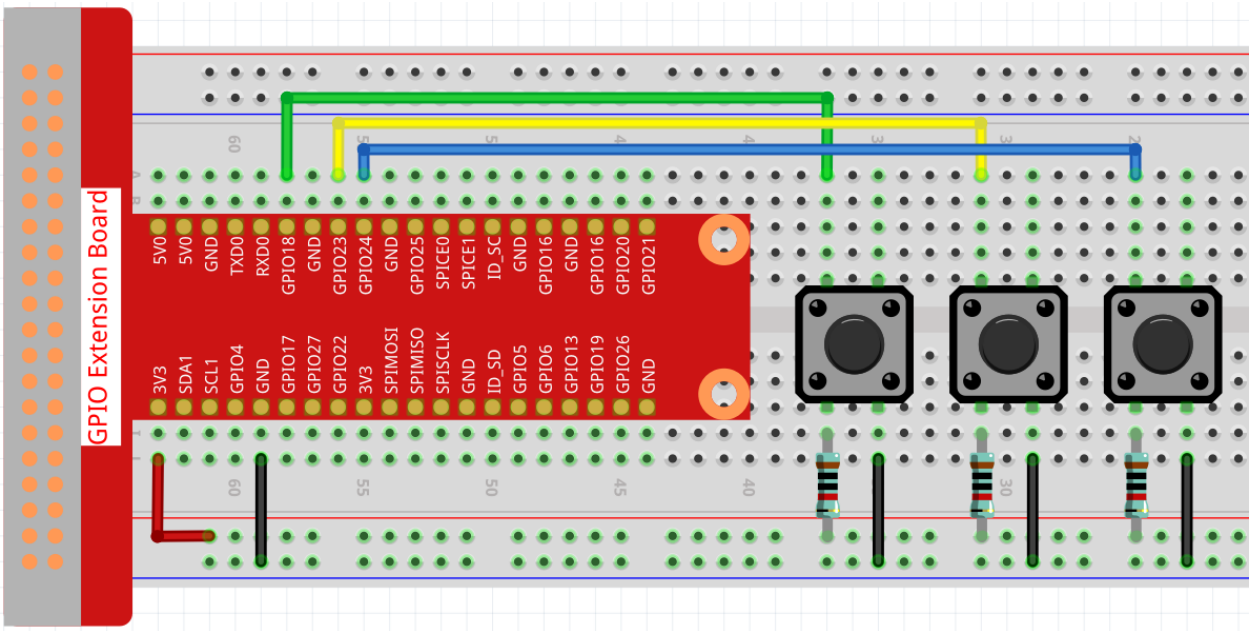
Weitere Informationen finden Sie unter Processing Reference.

Punkt auf der Schaukel

In diesem Projekt sind 3 Schaltflächen verbunden, eine zum Ändern der Punktgröße, eine zum Ändern der Position und die letzte zum Ändern der Farbe. Wenn Sie alle 3 Tasten gleichzeitig drücken, erhalten Sie einen schwingenden Punkt mit variabler Farbe.



Verdrahtung



Skizzieren

```
import processing.io.*;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
// 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;
    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;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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 ));
    }

    void update() {
        timer += speed;
    }

    void show() {
        fill(currentColor, 100, 100);
        ellipse(currentX, currentY, currentSize, currentSize);
    }
}
```

Wie es funktioniert?

Anstatt einen Punkt direkt zu zeichnen, erstellen wir hier eine `Dot` -Klasse. Dann deklarieren Sie das Objekt (in diesem Fall `myDot`).

Dies ist eine einfache Möglichkeit, Punkte mit mehreren identischen Eigenschaften zu zeichnen. Wenn wir beispielsweise dem Punkt in diesem Projekt drei Funktionen hinzufügen - Größe ändern, Position ändern und Farbe ändern -, dann hat jeder von uns deklarierte Punkt dieselbe Funktion. Wir können dieselbe Schaltfläche verwenden, damit sie dasselbe tun, oder wir können verschiedene Schaltflächen verwenden, um jeden Punkt separat zu steuern.

Die Verwendung von **classes** macht Ihre Skizze schön, kraftvoll und flexibel.

Class (Computerprogrammierung) - Wikipedia

Als nächstes werfen wir einen genaueren Blick auf die Klasse `Dot` .

```
Dot(float x, float y, float s, int c)
```

In der Deklaration muss es vier Parameter übergeben, nämlich den X- und Y-Koordinatenwert der Position, die Größe und die Farbe (hier ist es auf den HSB color mode).

Jedem Parameter werden 2 Wertesätze zugewiesen (Anfangswert und aktueller Wert).

```
float initX;
float initY;
float currentX;
float currentY;
int positionRange = 60;

float initSize;
float currentSize;
int sizeRange = 50;

int initColor;
int currentColor;
int ColorRange = 80;
```

Neben dem Anfangswert und dem aktuellen Wert gibt es auch eine Reihe von Bereichswerten. Es ist nicht schwer zu verstehen, dass der Anfangswert verwendet wird, um den Anfangszustand des Punktes (bestimmt durch die eingehenden Parameter) zu bestimmen, während sich der aktuelle Wert innerhalb des Bereichs ändert, um den Punkt zu bewegen.

Daher werden die aktuellen Werte der anderen drei Parameter mit Ausnahme des X-Koordinatenwertes wie folgt berechnet:

```
void setSize() {
    currentSize = initSize + sizeRange * sin( timer );
}

void setPosition() {
    currentY = initY + positionRange * cos( timer *2);
}

void setColor() {
    currentColor = int(initColor + ColorRange * sin( timer ));
}
```

Wenn Sie mit trigonometrischen Funktionen vertraut sind, sollte es nicht schwer sein, sine and cosine zu verstehen, was eine gleichmäßige periodische Änderung (von -1 zu 1) ergibt. des aktuellen Wertes des Punktes.

Wir müssen auch einen Startwert, `timer` , für die periodische Variation hinzufügen. Es fügt den festen Wert in die Methode `update()` ein und wird in `draw()` aufgerufen.

```
void update() {
    timer += speed;
}
```

Schließlich wird der Punkt mit der Methode `show()` entsprechend dem aktuellen Wert angezeigt, die auch in `draw()` aufgerufen wird.

```
void show() {
    fill(currentColor, 100, 100);
    ellipse(currentX, currentY, currentSize, currentSize);
}
```

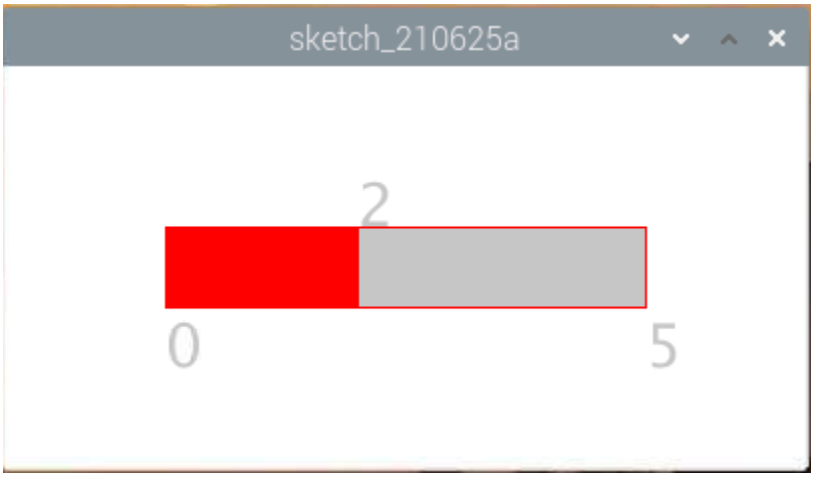
Was mehr?

Wenn Sie die Verwendung von Klassen beherrschen, können Sie bereits mehrere Punkte mit den gleichen Eigenschaften zeichnen. Warum also nicht etwas Cooleres versuchen? Wie wäre es zum Beispiel, ein stabiles Doppelsternsystem zu zeichnen oder ein ‚DUET'-Spiel zu machen?

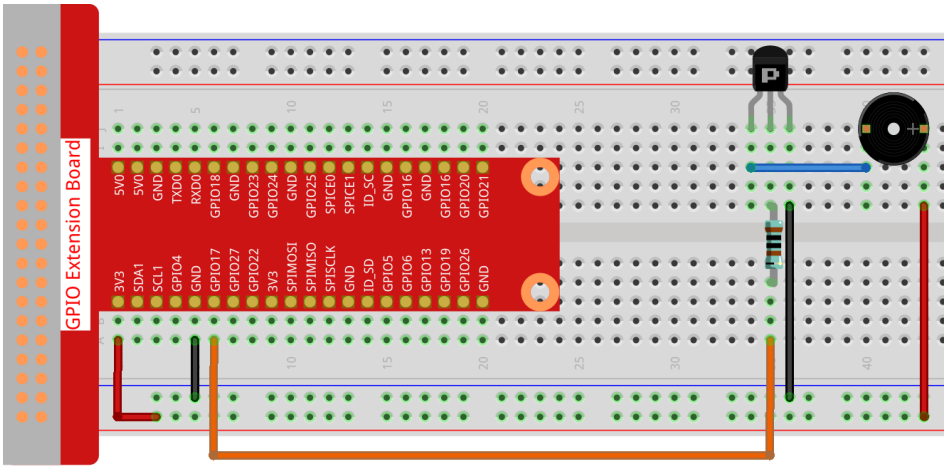
Weitere Informationen finden Sie unter Processing Reference.

Metronom

Hier machen wir ein Metronom, das Metronom ist in 5 Stufen unterteilt, je höher die Stufe, desto dringender der Summerruf.



Verdrahtung



Bemerkung: Hier wird ein aktiver Summer verwendet, der mit einem weißen Aufkleber versehen ist.

Skizzieren

```
import processing.io.*;

int level = 0;
int buzzerPin = 17;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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;
    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;
    }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    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;
    }
}
```

Wie es funktioniert?

Hier haben wir eine `Slider` -Klasse erstellt und sie als **WIDGET** fungieren lassen.

```
Slider(ax, ay, bx, by, min, max, v)
```

In der Deklaration muss es in 7 Parametern übergeben werden.

Die ersten vier Parameter bestimmen die Größe des Widgets, gefolgt von den Koordinaten (x1, y1) des Startpunkts in der oberen linken Ecke und (x2, y2) in der unteren rechten Ecke.

Die letzten drei Parameter bestimmen seinen Zahlenbereich (min bis max) und den Anfangswert.

Es hat zwei Methoden, die Wirkung von `dragPoint()` besteht darin, den Schieberegler verschiebbar zu machen und den aktuellen Positionswert des Schiebereglers zurückzugeben.

```
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;
}
```

Eine andere Methode `show()` besteht darin, den Slider anzuzeigen. Gleichzeitig werden der Bereichswert und der aktuelle Wert an der entsprechenden Stelle angezeigt.

```
void show(){
    rectMode(CORNERS);
    fill(200);
    stroke(255,0,0);
    rect(slotPointAX, slotPointAY, slotPointBX, slotPointBY);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

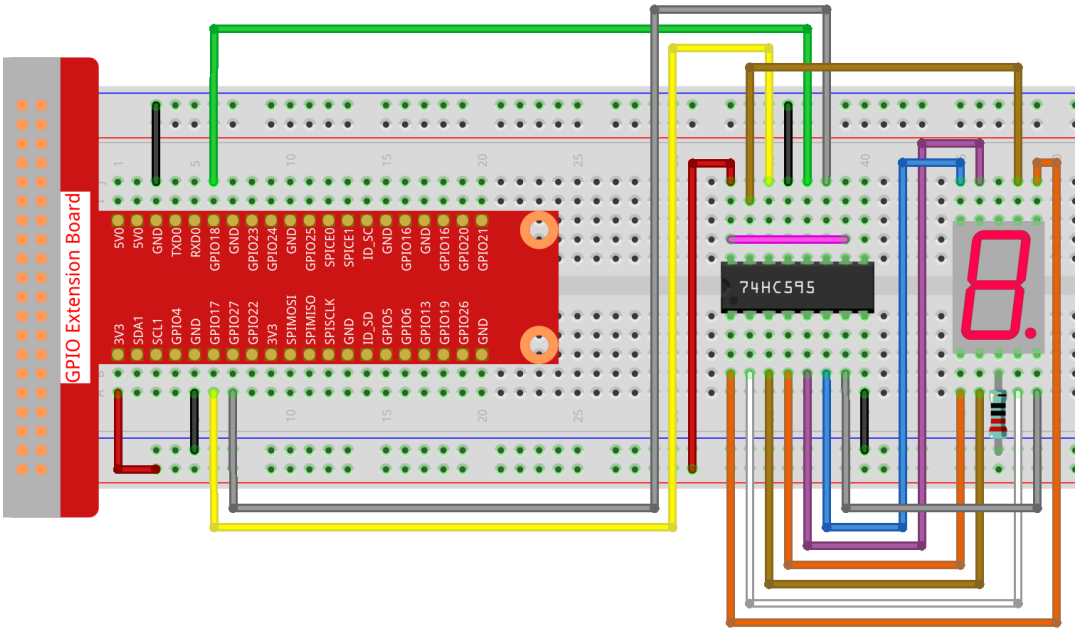
```
    fill(255,0,0);
    rect(slotPointAX, slotPointAY, linePoint, slotPointBY);
    fill(200);
    textSize(depth);
    text(minRange, slotPointAX, slotPointBY+depth);
    text(maxRange, slotPointBX, slotPointBY+depth);
    text(value, linePoint, slotPointAY);
}
```

Weitere Informationen finden Sie unter Processing Reference.

Anzahl anzeigen

In dieser Lektion verwenden wir die Verarbeitung, um eine 7-Segment-Anzeige anzusteuern, um eine Zahl von 0 bis 9 und A bis F anzuzeigen.

Verdrahtung



Skizzieren

```
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;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  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(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]);
}
```

Wie es funktioniert?

Importieren Sie `processing.io.*` und verwenden Sie die GPIO-Funktionsbibliothek, um die digitalen Röhrenpins zu steuern.

Definiere das Array `SegCode = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71}` was ein Segmentcode-Array von 0 bis. darstellt F in Hexadezimal (gemeinsame Kathode).

Die Funktion `setup()` setzt die drei Pins SDI,RCLK und SRCLK als Ausgang und die Anfangsdaten auf 0.

Die Funktion `hc595_shift(int dat)` wird verwendet, um den `SegCode` auf 74HC595 zu verschieben.

```
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))` bedeutet, dat um `i` Bits nach links zu verschieben und dann die & Operation mit 0x80 auszuführen.

Die Regel der & -Operation lautet, dass, wenn beide Seiten von & 1 sind, das Ergebnis 1 ist, andernfalls ist das Ergebnis 0.

Zum Beispiel nehmen wir an `dat=0x3f` , `i=2` (0011 1111 << 2 Verschiebung zu 1111 1100), dann 1111 1100 & 1000 0000 (0x80)) = 1000 0000.

Zuletzt weisen Sie SDI(DS) die Dat-Daten bitweise zu.

`digitalWrite(SRCLK, 1)` wenn SRCLK einen Anstiegsflankenimpuls von 0 bis 1 erzeugt, werden die Daten vom DS-Register zum Schieberegister übertragen;

`digitalWrite(RCLK, 1)` Wenn RCLK einen Anstiegsflankenimpuls von 0 auf 1 erzeugt, werden die Daten vom Schieberegister zum Speicherregister übertragen.

```
fill(0,25,88);
textAlign(CENTER,CENTER);
textSize(height*0.8);
```

- `fill()` : Legt die Farbe fest, die zum Füllen von Formen verwendet wird.
- `textAlign(CENTER,CENTER)` : Legt die aktuelle Ausrichtung zum Zeichnen von Text fest. Die Parameter `LEFT`, `CENTER`, und `RIGHT` setzen die Darstellungseigenschaften der Buchstaben in Relation zu den Werten für die x- und y-Parameter der Funktion `text()` .
- `textSize()` : Setzt die aktuelle Schriftgröße. Diese Größe wird in allen nachfolgenden Aufrufen der Funktion `text()` verwendet. Die Schriftgröße wird in Pixeleinheiten gemessen.

Diese Funktionen können den bei der Verarbeitung angezeigten Textstil anpassen.

```
void draw() {

        background(255);
        int number = (frameCount%100)/10;
        text(number, width/2, height/2);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        hc595_shift(SegCode[number]);
}
```

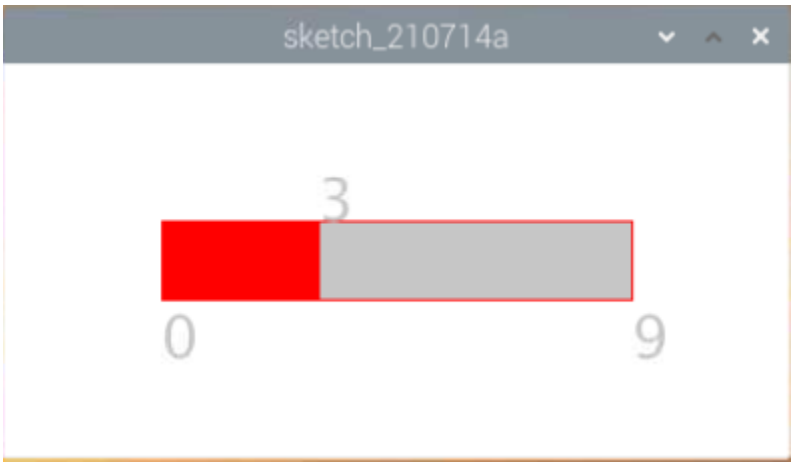
Der `frameCount` ist ein Seed, der mit `frameRate` verwandt ist. Standardmäßig ist `frameRate` 60, was bedeutet, dass `frameCount` 60, was bedeutet, dass

Dann können wir Verarbeitung und 7-Segment-Anzeige die Zahl von 0 bis 9 und A bis F gleichzeitig anzeigen lassen.

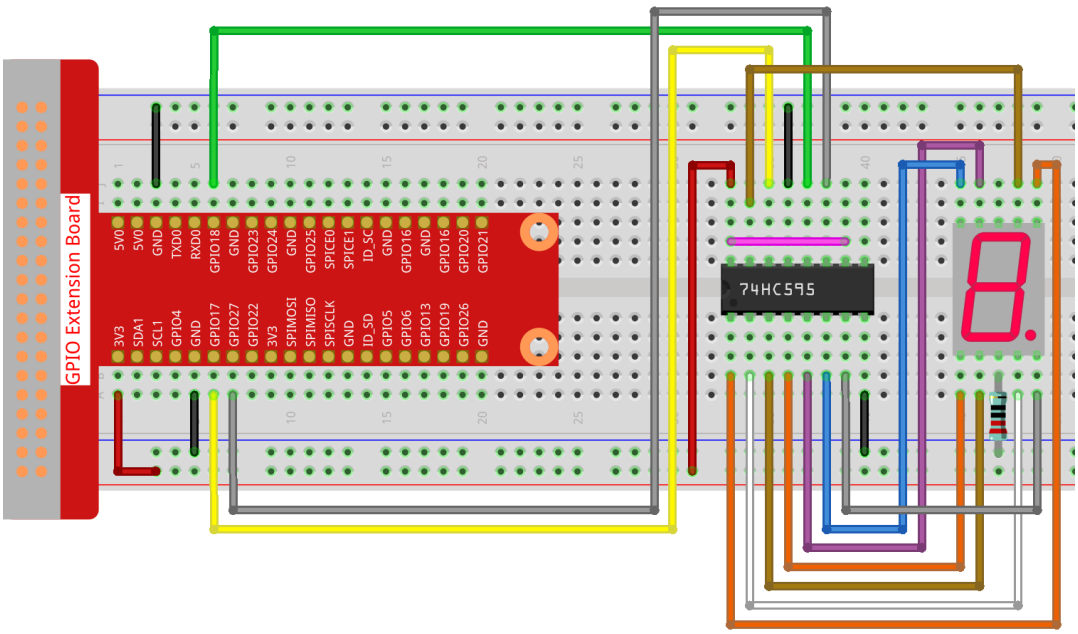
Weitere Informationen finden Sie unter Processing Reference.

Zahl ziehen

Lassen Sie uns einen Schieberegler zeichnen, um die 7-Segment-Anzeige zu steuern.



Verdrahtung



Skizzieren

```
import processing.io.*;

int number = 0;
int levelRange=9;
Slider mySlider;

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);
    } 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);
}

void draw() {

    background(255);
    mySlider.show();
    hc595_shift(SegCode[number]);
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}

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;
    }

    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;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
}
```

Wie es funktioniert?

Dieses Projekt integriert den Slider und die 7-Segment-Anzeige des vorherigen Kurses. Spezifische Wissenspunkte finden Sie unter *Anzahl anzeigen* and *Metronom*.

1.13 Spielen mit Scratch (nicht für Pi 5)

Scratch ist eine blockbasierte visuelle Programmiersprache und Website, die sich in erster Linie an Kinder von 8 bis 16 Jahren als pädagogisches Werkzeug zum Programmieren richtet. Benutzer der Website können Projekte im Web über eine blockartige Oberfläche erstellen. Der Service wird vom MIT Media Lab entwickelt, wurde in mehr als 70 Sprachen übersetzt und wird in den meisten Teilen der Welt verwendet.

Hier lernen Sie, Scratch 3 im Raspberry Pi zu verwenden und mit Scratch 3 auf Raspberry Pi GPIO zuzugreifen, was mit Online Scratch nicht möglich ist.

1.13.1 Kurzanleitung zu Scratch

Bemerkung: Wenn Sie mit Scratch 3 programmieren, benötigen Sie möglicherweise einen Bildschirm, um eine bessere Erfahrung zu machen. Wenn Sie keinen Bildschirm haben, können Sie natürlich auch VNC verwenden, um aus der Ferne auf den Raspberry Pi-Desktop zuzugreifen; eine detaillierte Anleitung finden Sie unter *VNC*.

Darüber hinaus benötigt Scratch 3 mindestens 1 GB RAM, und wir empfehlen einen Raspberry Pi 4 mit mindestens 2 GB RAM. Sie können Scratch 3 zwar auf einem Raspberry Pi 2, 3, 3B+ oder einem Raspberry 4 mit 1 GB RAM ausführen, aber die Leistung ist bei diesen Modellen reduziert und je nachdem, welche andere Software Sie gleichzeitig ausführen, kann Scratch 3 aufgrund von Speichermangel nicht starten.

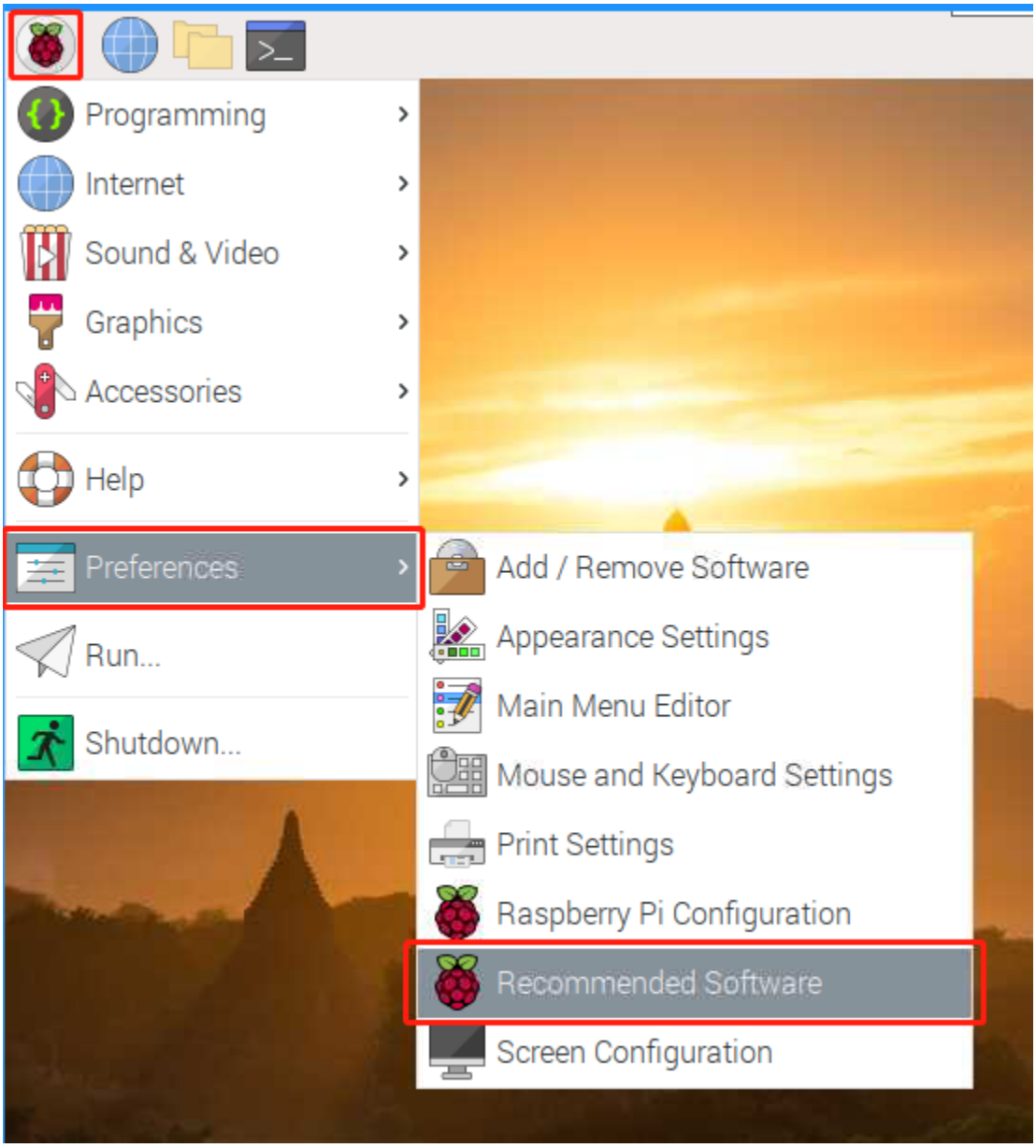
Scratch 3 installieren

Bei der Installation von Raspberry Pi OS (*Installieren des Betriebssystems*)müssen Sie die Version mit Desktop auswählen, entweder nur mit Desktop oder mit Desktop und empfohlener Software.

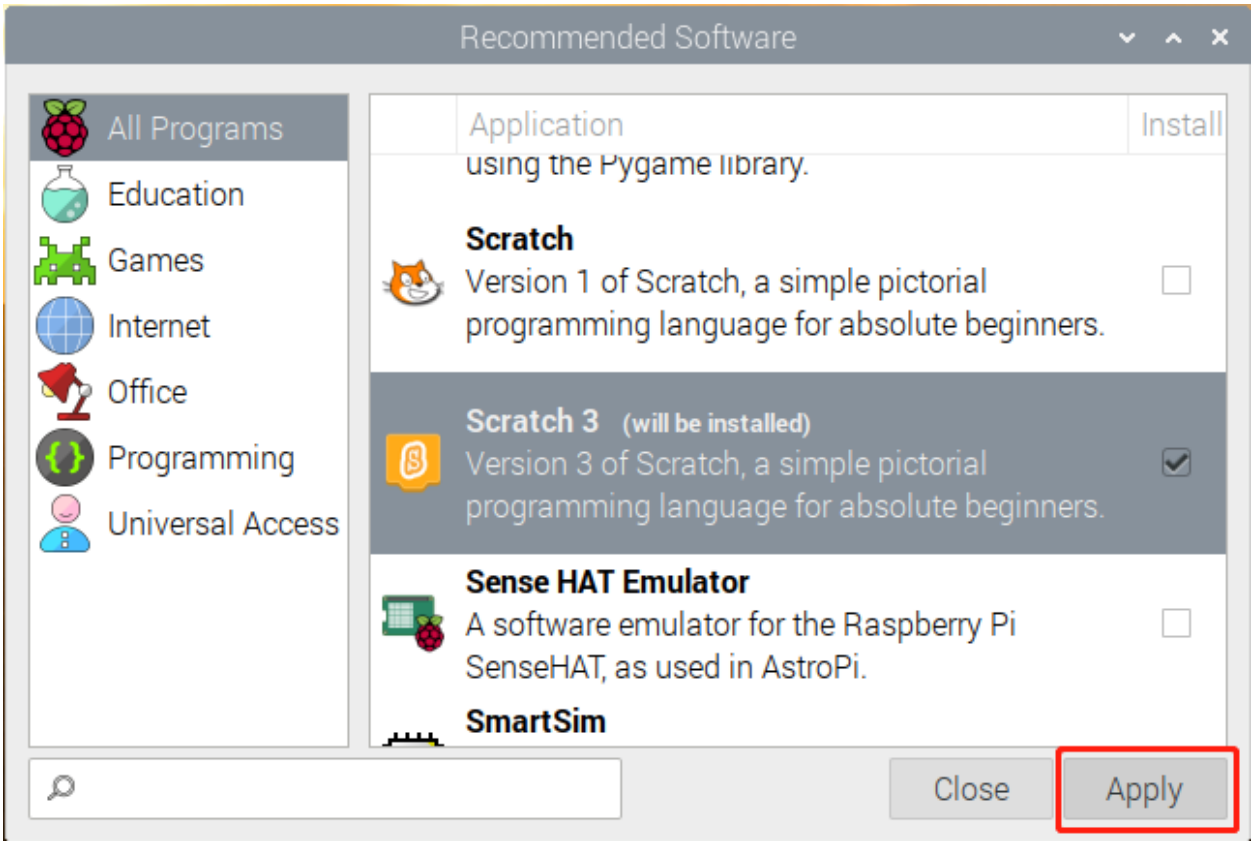
Wenn Sie die Version mit der empfohlenen Software installieren, können Sie Scratch 3 im Systemmenü unter **Programming** sehen.

Wenn Sie die reine Desktop-Version installiert haben, müssen Sie Scratch 3 manuell installieren, wie unten beschrieben.

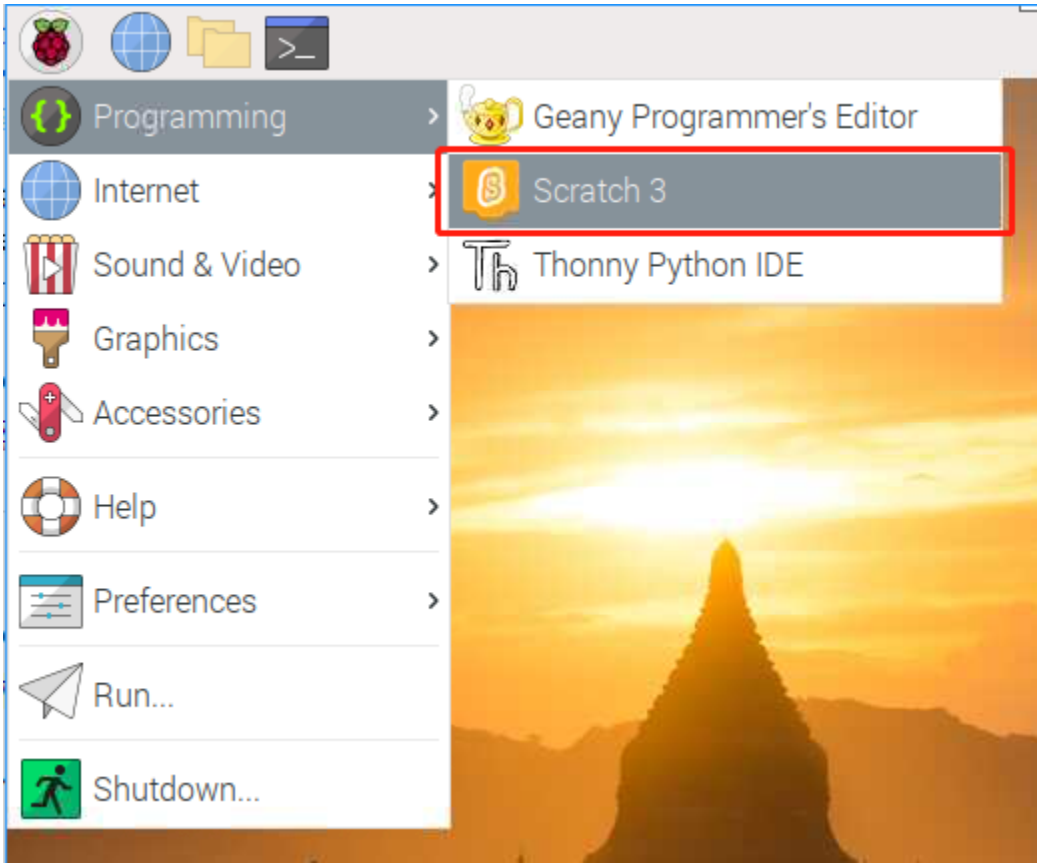
Öffnen Sie das Menü und klicken Sie auf **Preferences** -> **Recommended Software**.



Suchen Sie Scratch 3 und markieren Sie es, klicken Sie dann auf **Apply** und warten Sie schließlich, bis die Installation abgeschlossen ist.



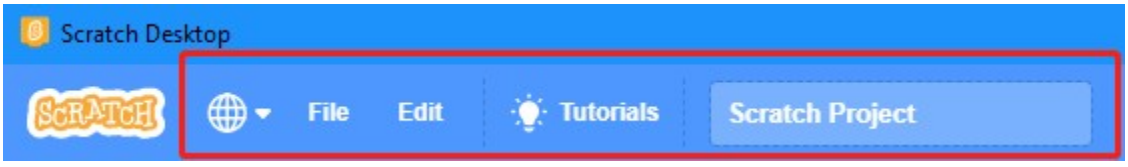
Sobald die Installation abgeschlossen ist, sollten Sie sie unter **Programming** im Systemmenü finden.



Über die Benutzeroberfläche von Scratch 3

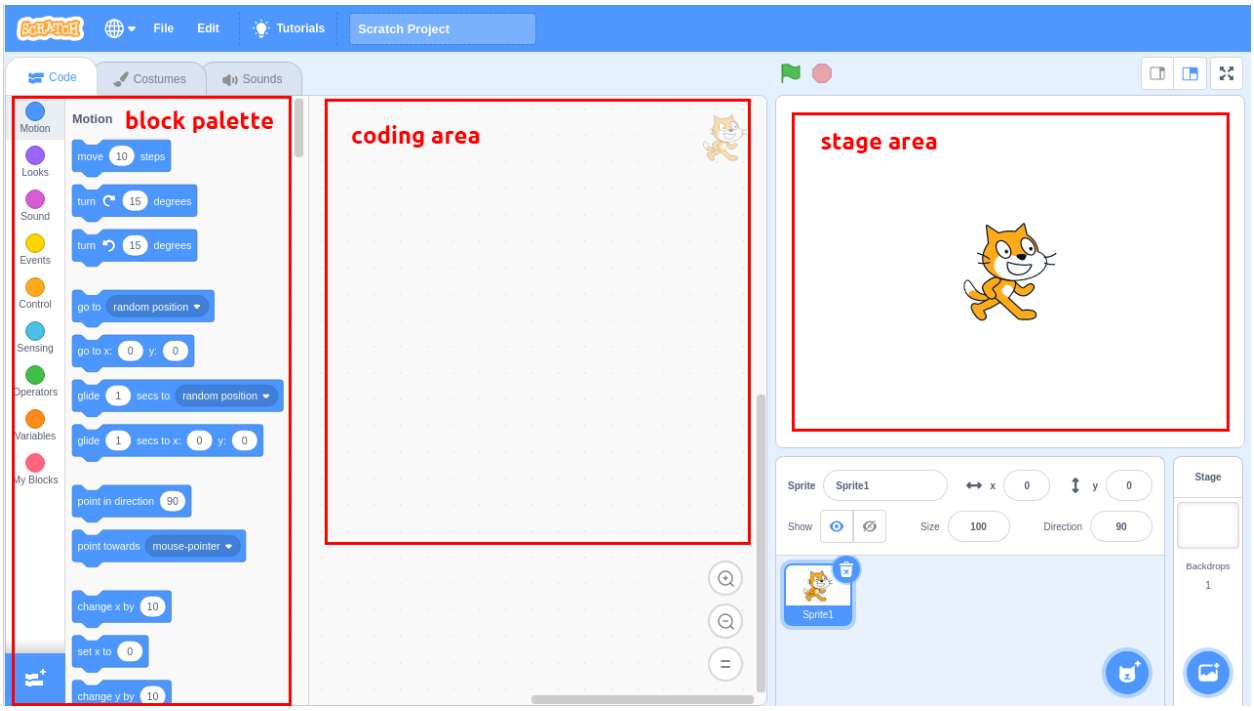
Scratch 3 ist so konzipiert, dass es Spaß macht, lehrreich und leicht zu erlernen ist. Es verfügt über Werkzeuge zum Erstellen von interaktiven Geschichten, Spielen, Kunst, Simulationen und mehr, die auf blockbasierter Programmierung basieren. Scratch hat auch einen eigenen Mal- und Sound-Editor integriert.

Oben in Scratch 3 gibt es einige grundlegende Optionen, die erste von links nach rechts ist die Sprachoption, mit der Sie verschiedene Sprachen für die Programmierung auswählen können. Die zweite ist die Option **File** . Mit dieser Option können Sie neue Dateien erstellen, lokale Dateien lesen und aktuelle Dateien speichern. Die dritte ist die Option **Edit** , mit der Sie einige Löschvorgänge fortsetzen und den Beschleunigungsmodus aktivieren können (in dem die Sprite-Bewegung besonders schnell wird). Die vierte Option ist die Option **Tutorials** , mit der Sie sich Tutorials für einige Projekte ansehen können. Die fünfte Option ist die Option Dateinamen, mit der Sie das Projekt umbenennen können.

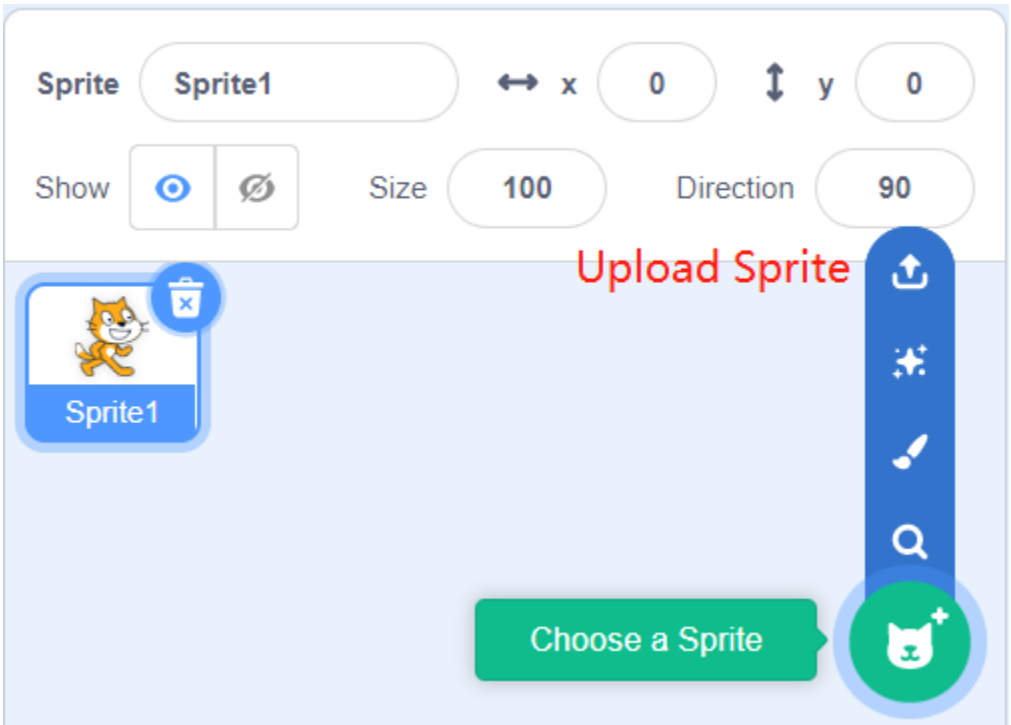


Skripte

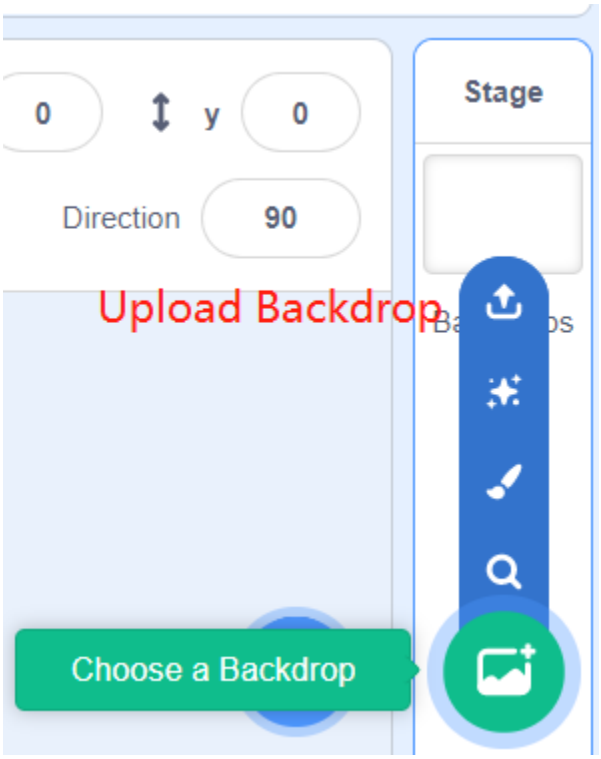
Es besteht aus drei Hauptbereichen: einem Bühnenbereich, einer Blockpalette und einem Codierbereich. Programmierung durch Anklicken und Ziehen des Blocks auf der Blockpalette in den Codierbereich und schließlich werden Ihre Programmierergebnisse im Bühnenbereich angezeigt.



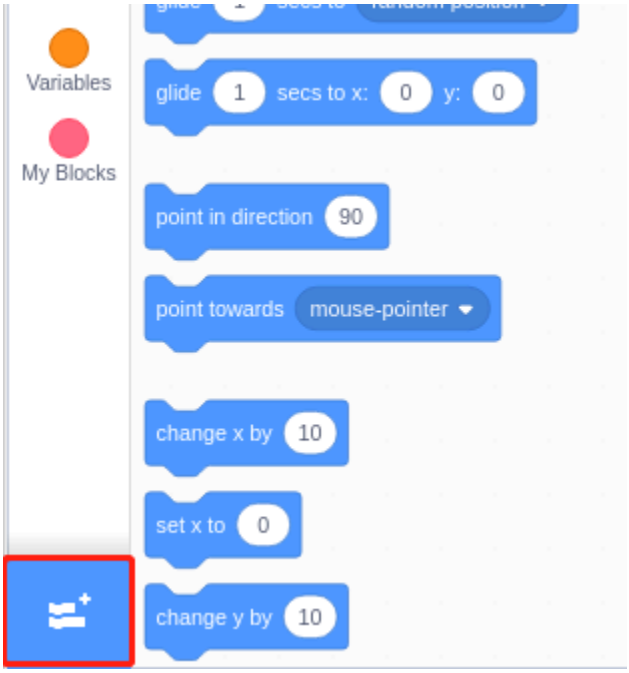
Hier ist der Sprites-Bereich von Scratch 3. Oberhalb des Bereichs befinden sich die Grundparameter der Sprites. Sie können Sprites hinzufügen, die mit Scratch 3 geliefert werden, oder lokale Sprites hochladen.



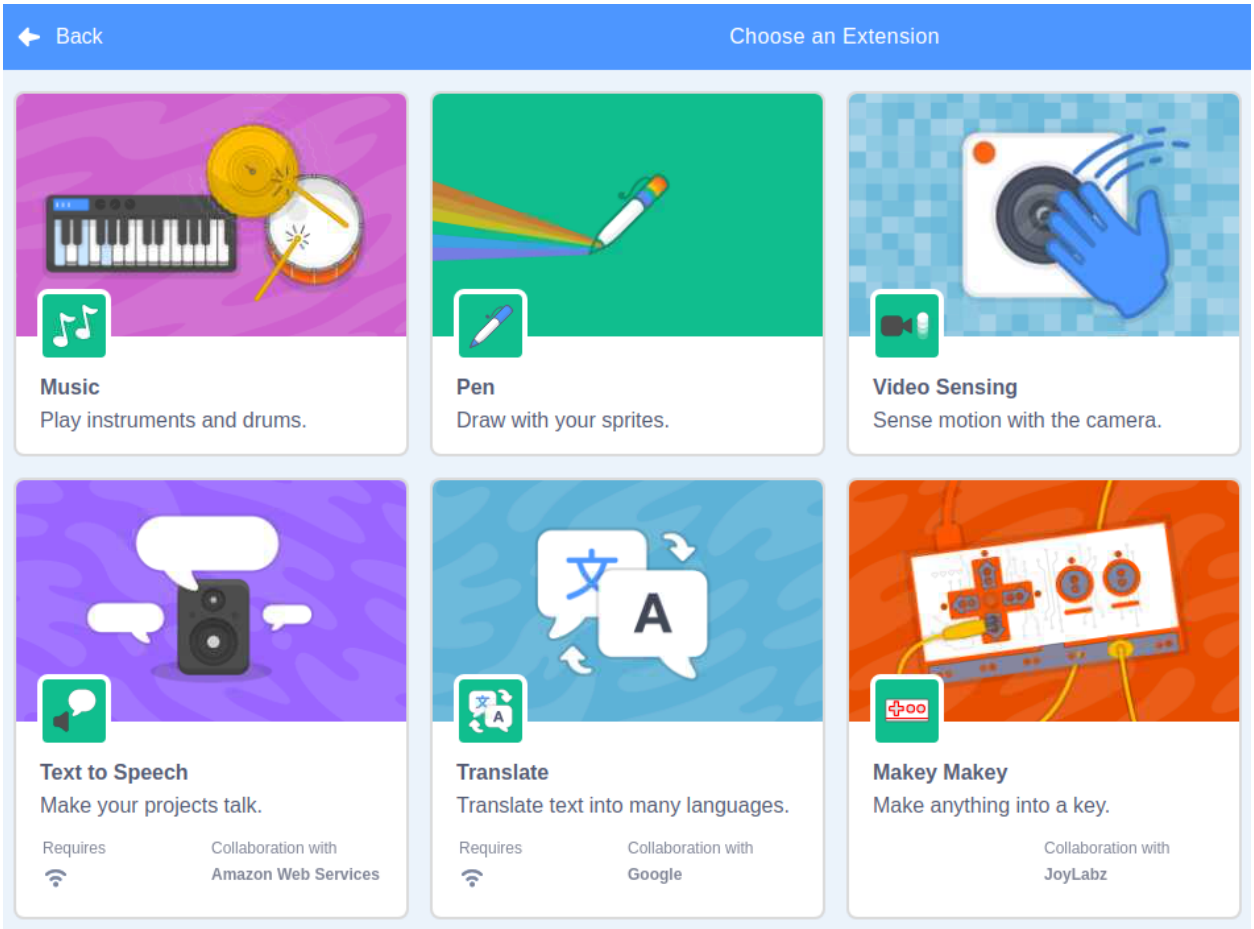
Hier ist der Scratch 3 Hintergrundbereich, hauptsächlich um einen passenden Hintergrund für deine Bühne hinzuzufügen. Du kannst den Hintergrund, der mit Scratch 3 geliefert wird, hinzufügen oder einen lokalen Hintergrund hochladen.



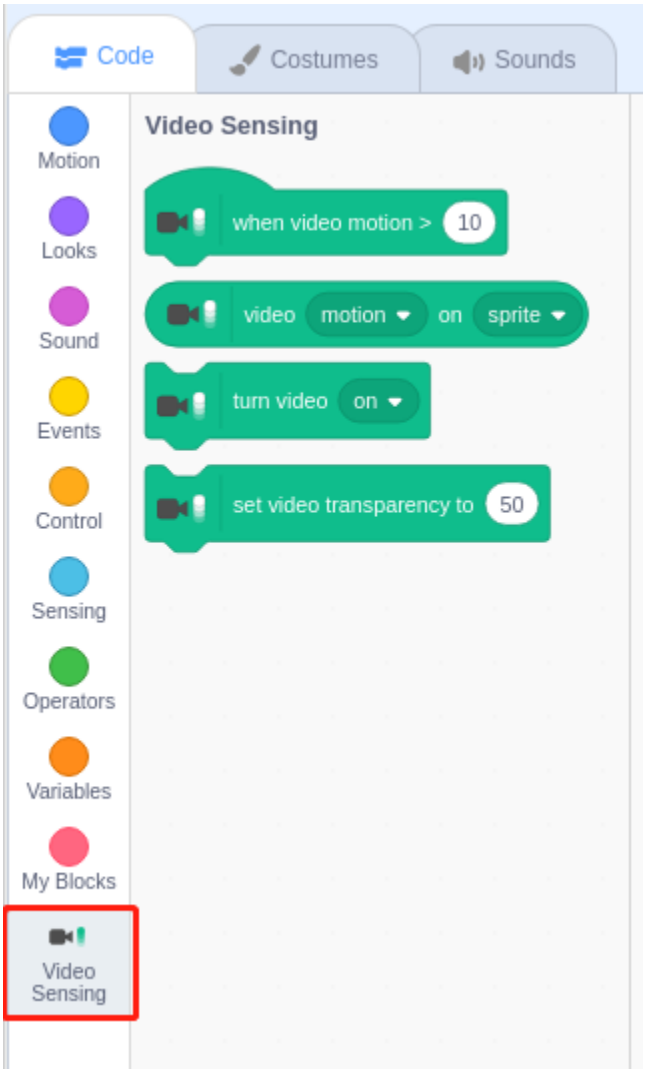
Dies ist eine Schaltfläche **Add Extension** .



In Scratch 3 können wir alle Arten von nützlichen Erweiterungen hinzufügen, hier nehmen wir **Video Sensing** als Beispiel und klicken es an.

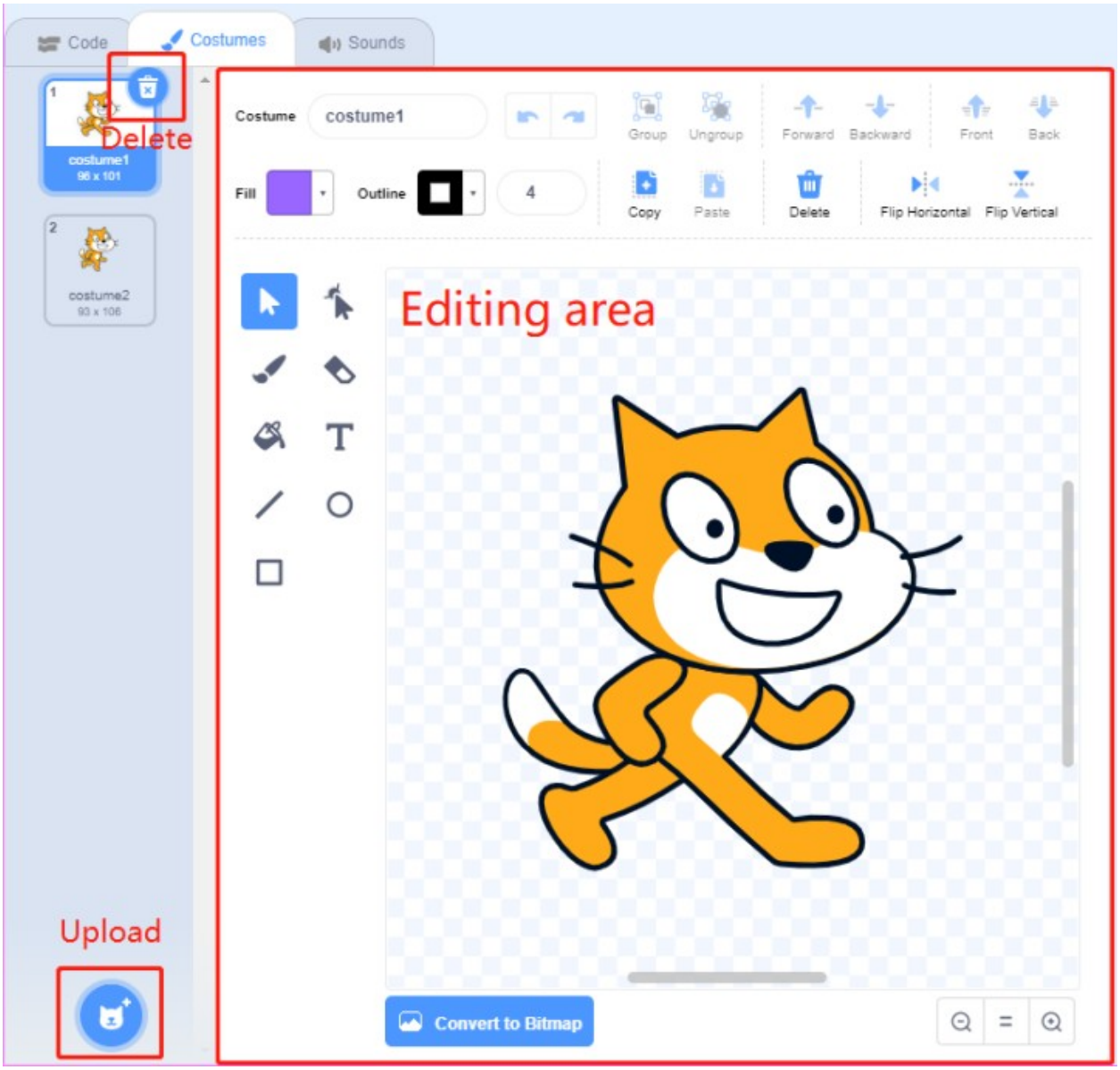


Sie wird in der Blockpalette angezeigt und Sie können die mit dieser Erweiterung verbundenen Funktionen nutzen. Wenn Sie eine Kamera angeschlossen haben, sehen Sie den Kamerabildschirm im Bühnenbereich.



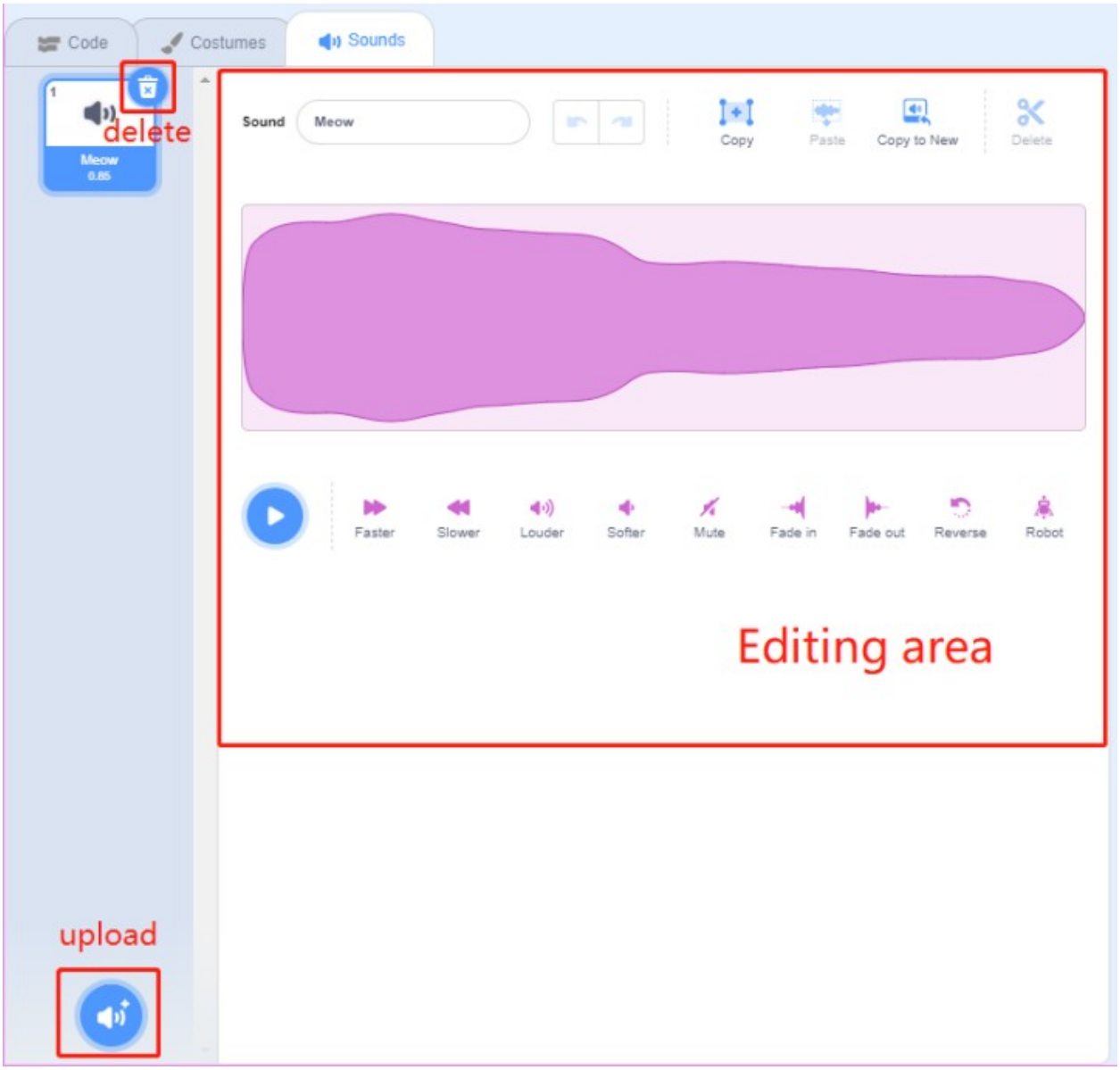
Kostüme

Klicken Sie auf die Option **Costumes** in der oberen linken Ecke, um die Kostümpalette zu öffnen. Verschiedene Kostüme ermöglichen den Sprites unterschiedliche statische Bewegungen, und wenn diese statischen Bewegungen zusammengefügt werden, bilden sie eine kohärente dynamische Bewegung.



Klänge

Möglicherweise müssen Sie einige Musikclips verwenden, um Ihre Experimente interessanter zu gestalten. Klicken Sie auf die Option **Sounds** in der oberen linken Ecke und Sie können den aktuellen Sound bearbeiten oder einen neuen Sound auswählen bzw. hochladen.



1.13.2 Projekte

In diesem Kapitel spielen Sie mit Scratch 3 durch 18 Projekte.

Wenn Sie ein Benutzer sind, der gerade Scratch 3 verwendet hat, empfehlen wir Ihnen, die Projekte der Reihe nach auszuprobieren, damit Sie schnell mit Scratch 3 beginnen können.

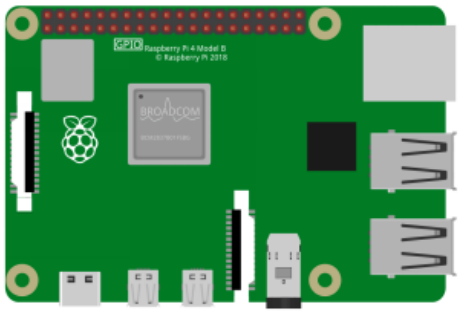
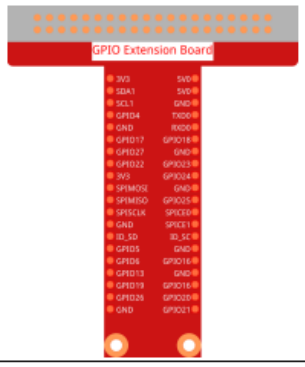
Bemerkung: Bevor Sie die Projekte ausprobieren, sollten Sie die entsprechenden Materialien und Codedateien heruntergeladen haben. Öffnen Sie ein Terminal und geben Sie den folgenden Befehl ein, um sie von github herunterzuladen.

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

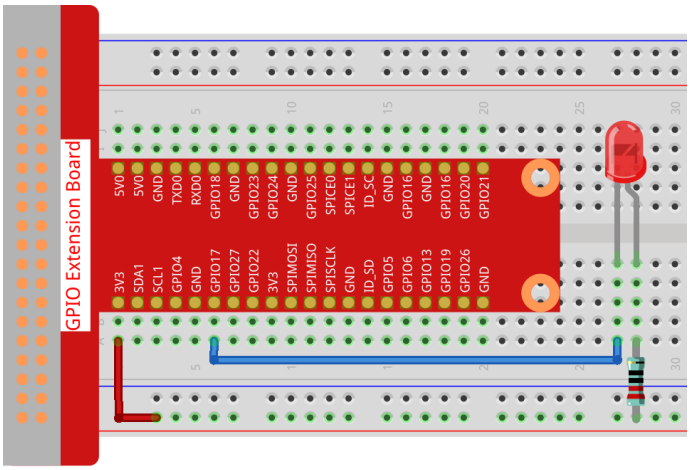
1.1 Zauberstab

Heute werden wir LED, Raspberry Pi und Scratch verwenden, um ein lustiges Spiel zu machen. Wenn wir den Zauberstab schwingen, blinkt die LED.

Erforderliche Komponenten

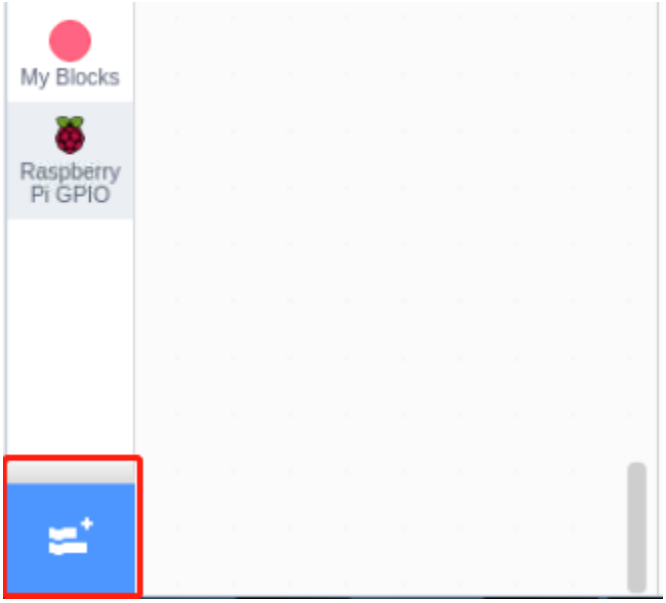
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard		1 * Resistor(220Ω)

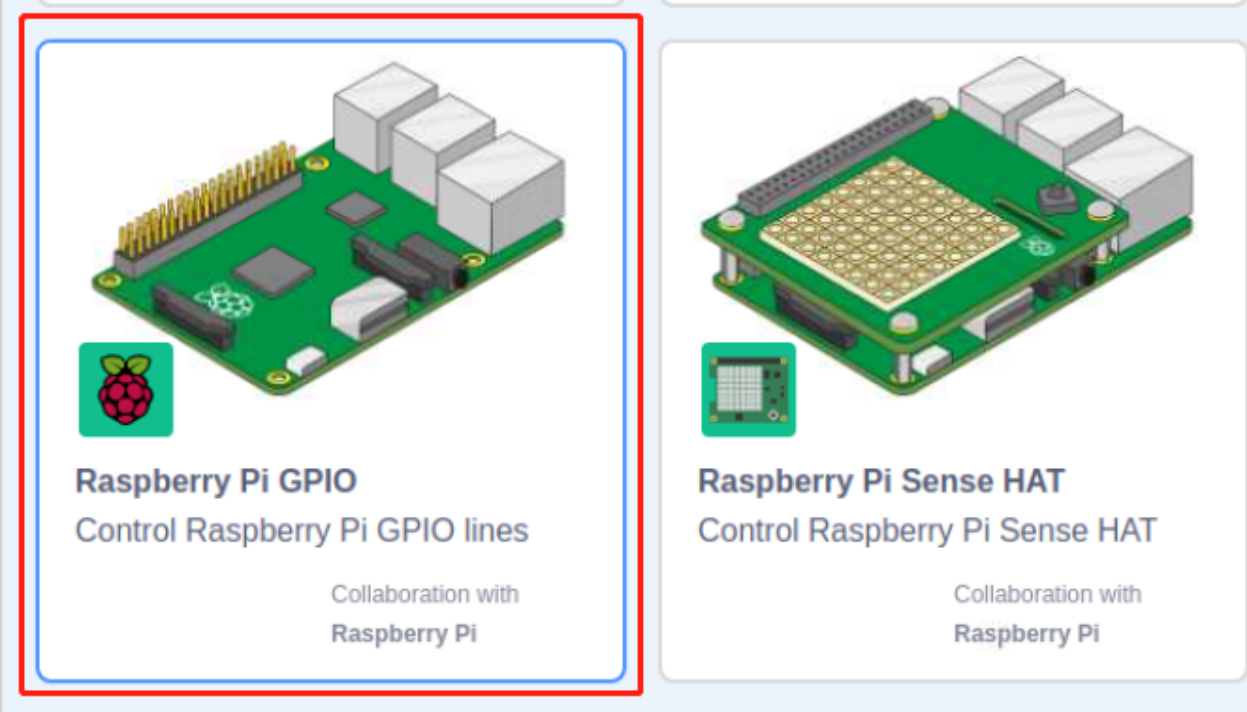
Bauen Sie den Stromkreis auf

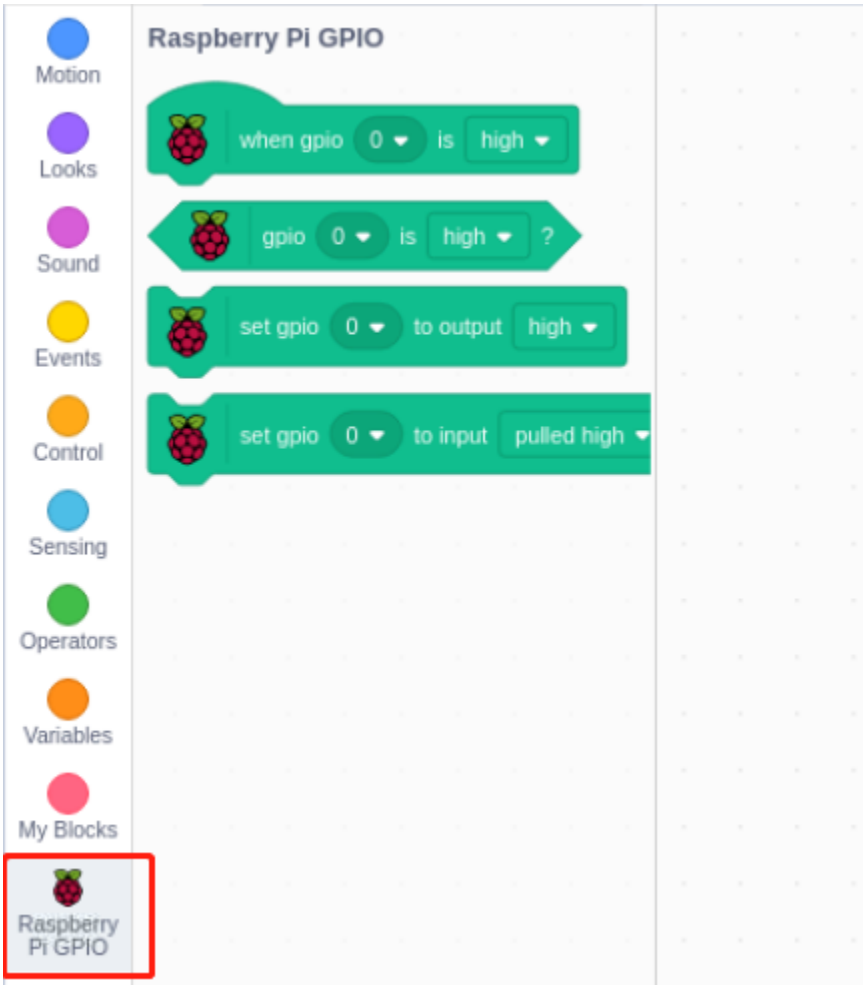


GPIO-Erweiterung hinzufügen

Klicken Sie auf die Schaltfläche **Add Extension** in der unteren linken Ecke und fügen Sie dann **Raspberry Pi GPIO** hinzu, eine Erweiterung, die wir für alle unsere Scratch-Projekte verwenden.

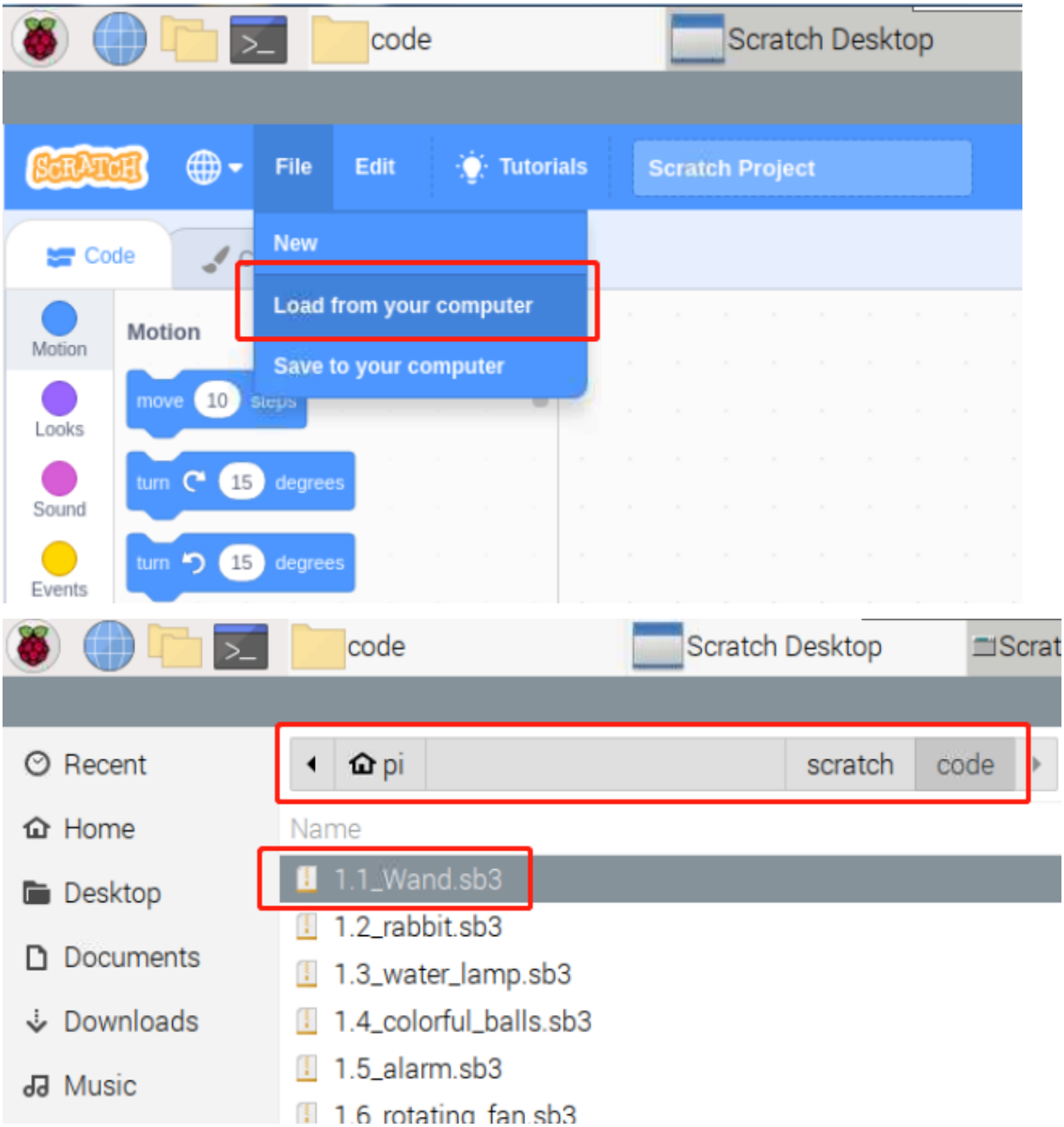



Raspberry Pi GPIO
Control Raspberry Pi GPIO lines
Collaboration with
Raspberry Pi
Raspberry Pi Sense HAT
Control Raspberry Pi Sense HAT
Collaboration with
Raspberry Pi

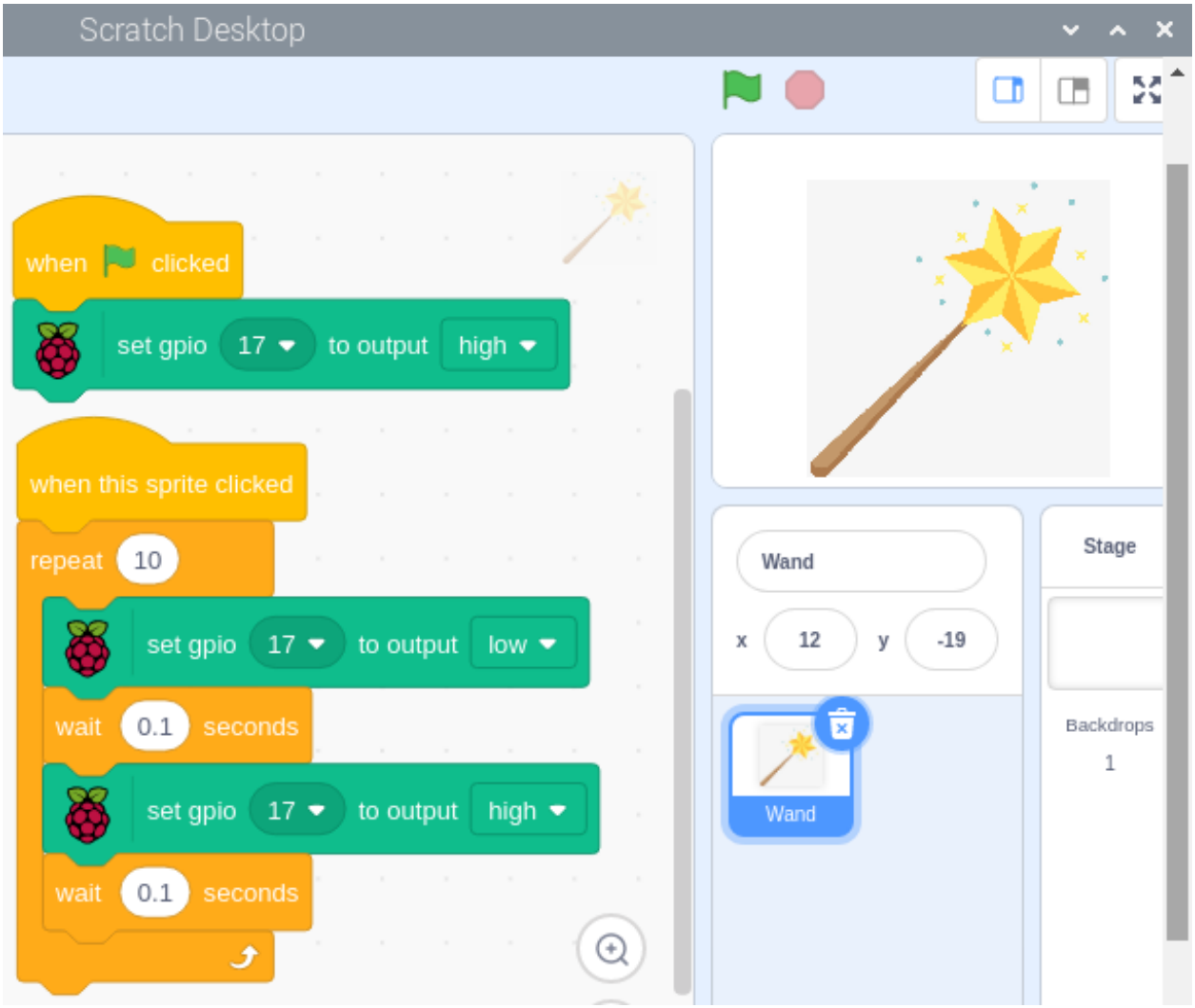

Motion
Looks
Sound
Events
Control
Sensing
Operators
Variables
My Blocks
Raspberry Pi GPIO
Raspberry Pi GPIO
when gpio 0 is high
gpio 0 is high ?
set gpio 0 to output high
set gpio 0 to input pulled high

Laden Sie den Code und sehen Sie, was passiert

Lade die Code-Datei von deinem Computer (`home/pi/davinci-kit-for-raspberry-pi/scratch/code`) in Scratch 3.

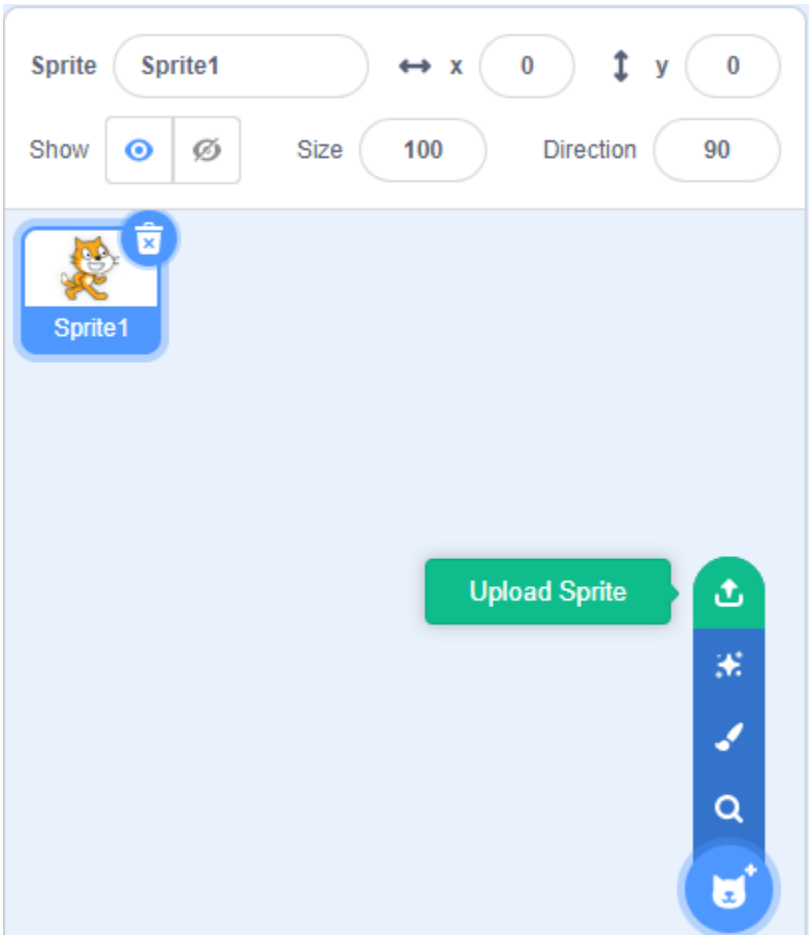


Nachdem du den Zauberstab im Bühnenbereich angeklickt hast, blinkt die LED zwei Sekunden lang.

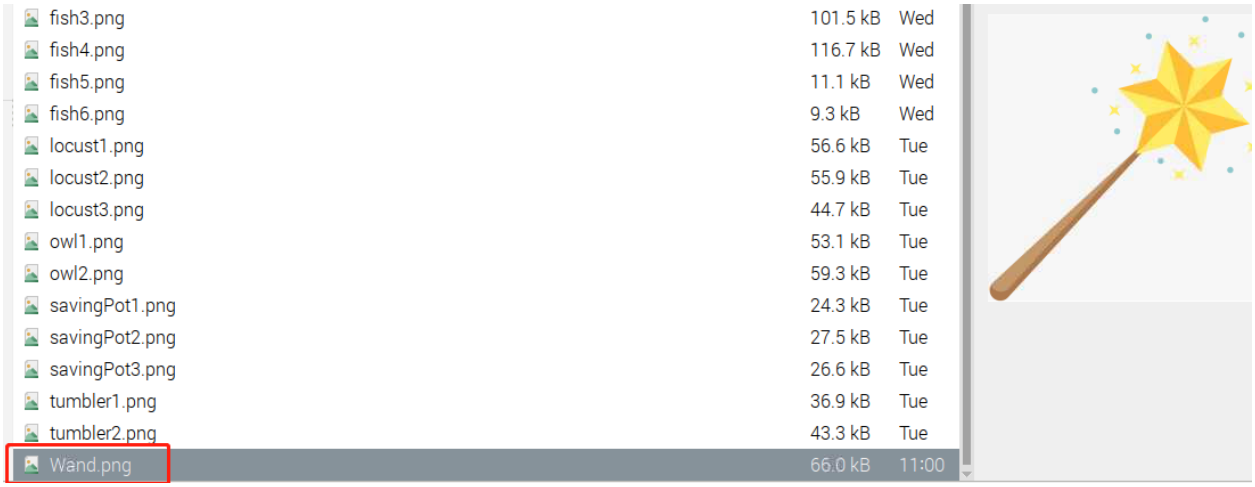


Tipps zu Figur

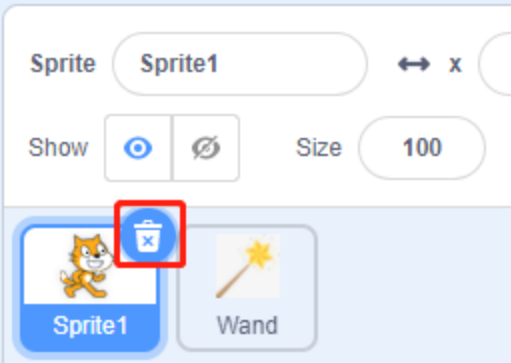
Klicken Sie auf die Schaltfläche **Upload Sprite**.



Lade **Wand.png** aus dem Pfad `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` in Scratch 3 hoch.

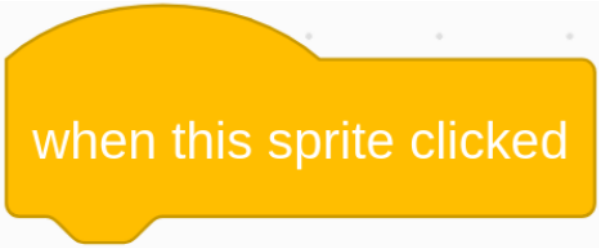


Löschen Sie schließlich die **Sprite1**.

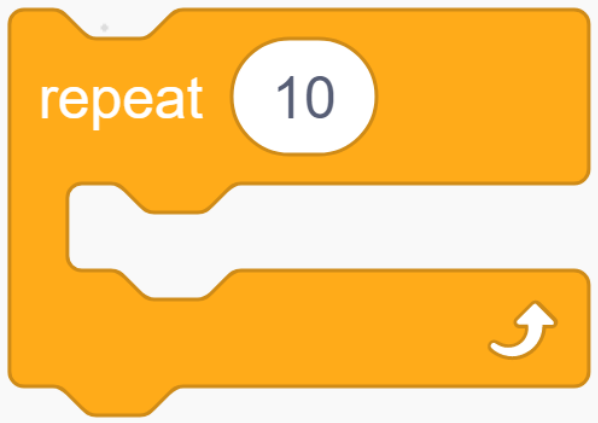


Tipps zu Codes

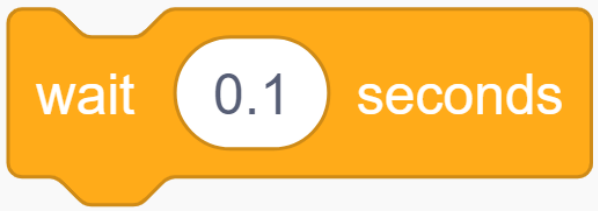
Dies ist ein Ereignisblock, dessen Auslösebedingung das Klicken auf die grüne Flagge auf der Bühne ist. Ein Auslöseereignis ist am Anfang aller Codes erforderlich, und Sie können andere Auslöseereignisse in der Kategorie **Events** der **block palette** auswählen.



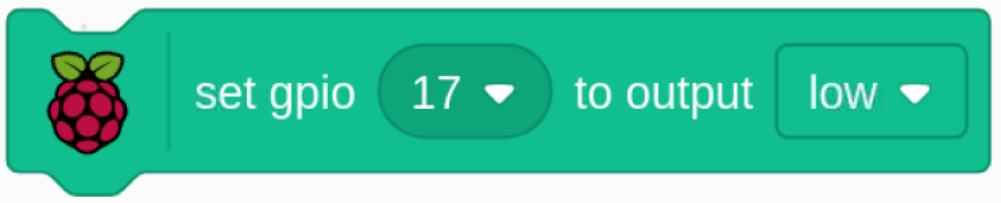
Zum Beispiel können wir jetzt das Auslöseereignis in einen Klick auf das Figur ändern.



Dies ist ein Block mit einer bestimmten Anzahl von Zyklen. Wenn wir die Zahl 10 eingeben, werden die Ereignisse im Block 10 Mal ausgeführt.



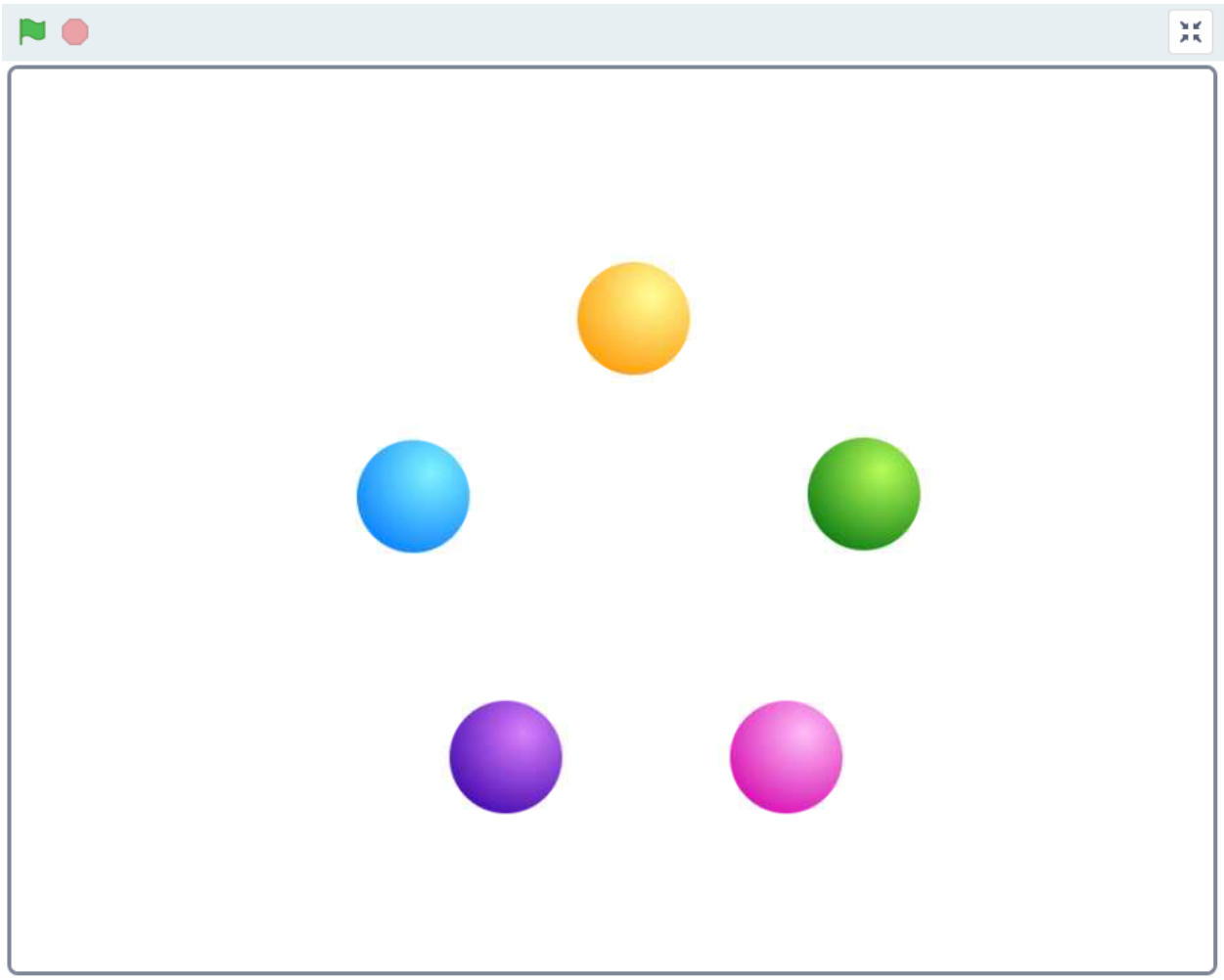
Mit diesem Satz wird das Programm für eine bestimmte Zeitspanne in Sekunden angehalten.



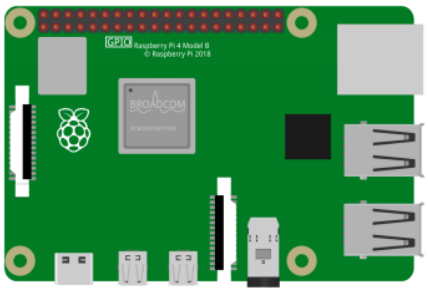
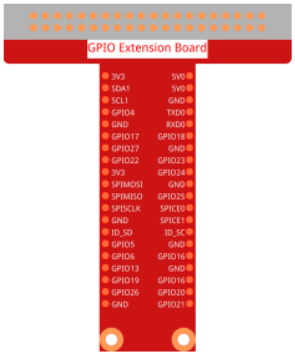
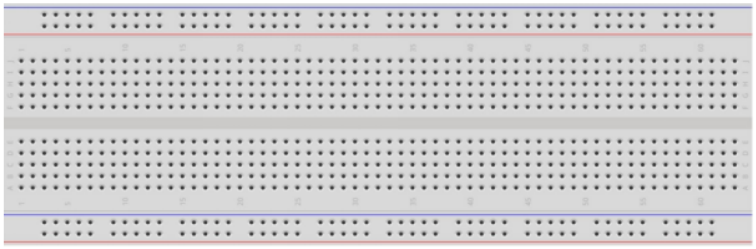
Da in Scratch die BCM-Benennungsmethode verwendet wird, setzt dieser Code GPIO17(BCM17) auf 0V (Low Level). Da die Kathode der LED mit GPIO17 verbunden ist, wird die LED aufleuchten. Wenn Sie dagegen GPIO(BCM17) auf einen hohen Wert einstellen, wird die LED ausgeschaltet.

1.2 Bunte Kugeln

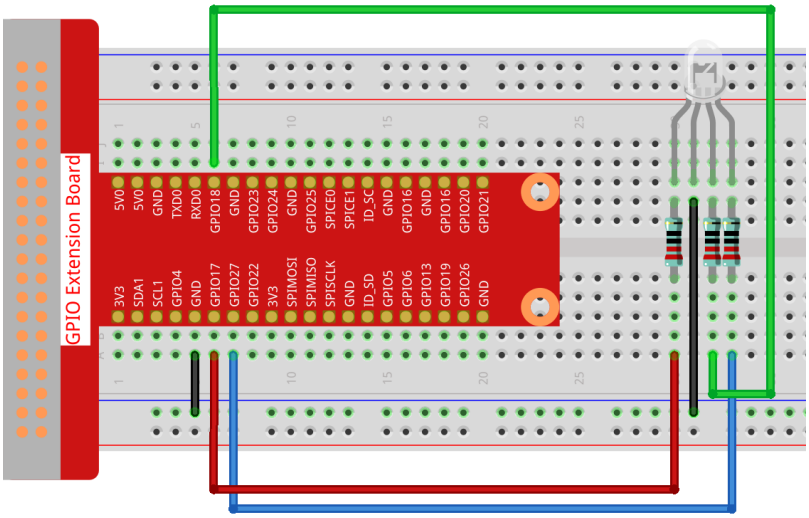
Wenn du auf verschiedenfarbige Kugeln auf der Bühne klickst, leuchtet die RGB-LED in verschiedenen Farben auf.



Erforderliche Komponenten

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

Bauen Sie den Stromkreis auf

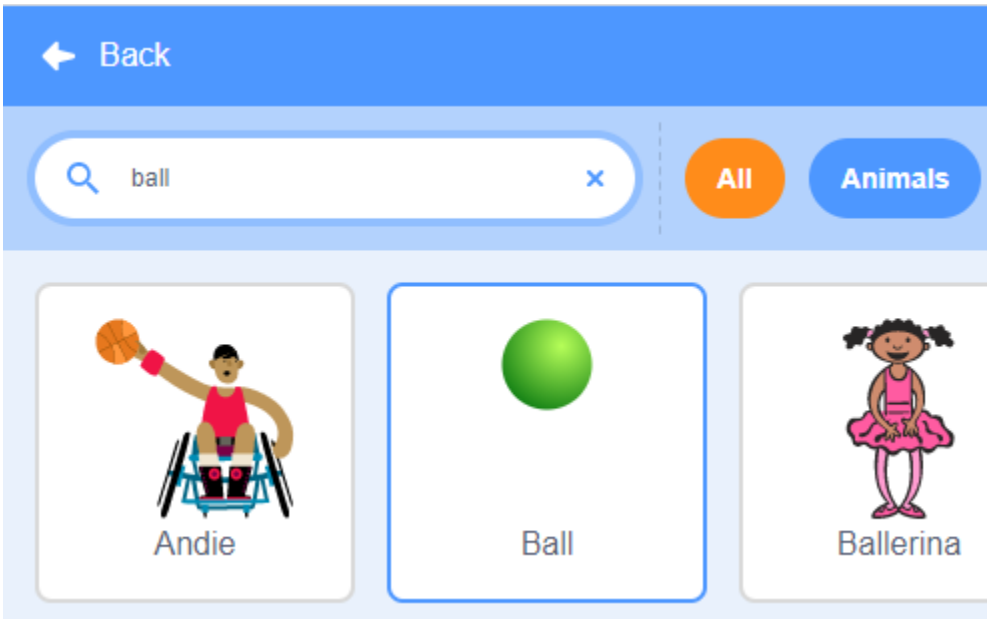


Laden Sie den Code und sehen Sie, was passiert

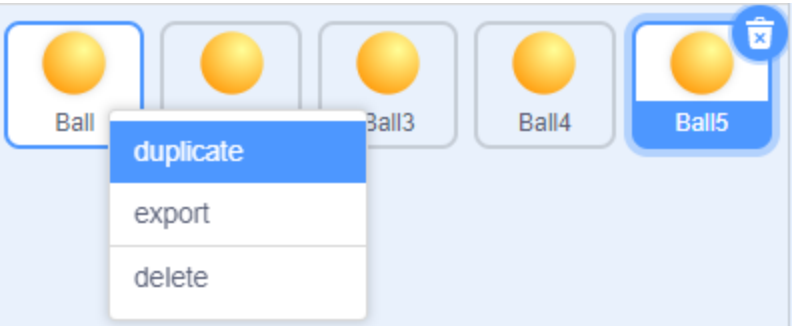
Nachdem du die Codedatei (`1.2_colorful_balls.sb3`) in Scratch 3 geladen hast, leuchtet die RGB-LED gelb, blau, rot, grün oder lila, wenn du auf den entsprechenden Ball klickst.

Tipps zu Figur

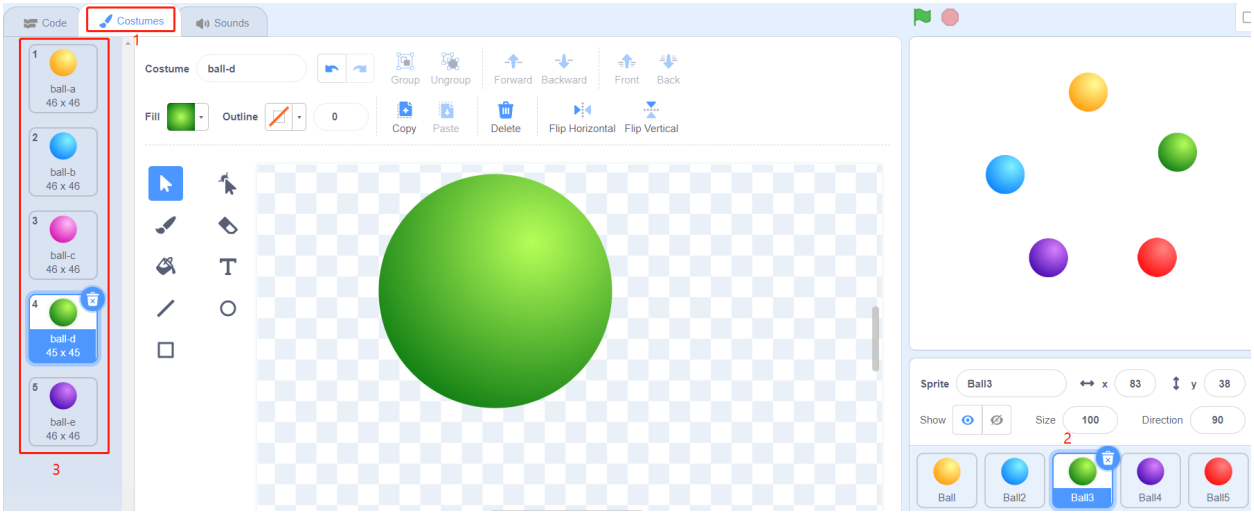
Löschen Sie das Standard-Sprite und wählen Sie dann das Sprite **Ball**.



Und duplizieren Sie es 5 Mal.



Wähle verschiedene Kostüme für diese 5 **Ball** -Sprites und bewege sie an die entsprechenden Positionen.

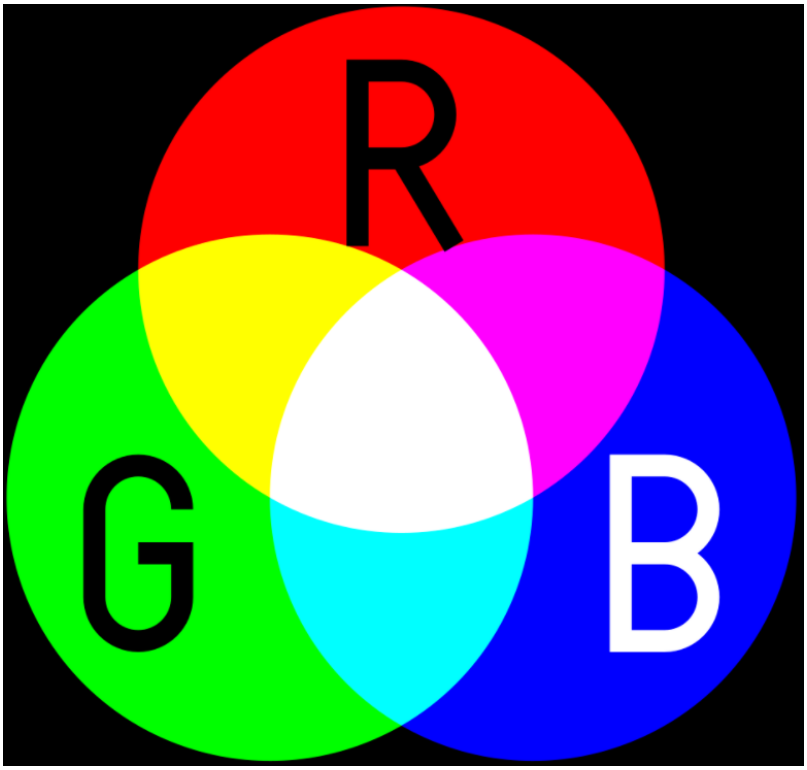


Tipps zu Codes

Bevor wir den Code verstehen, müssen wir das RGB color model verstehen.

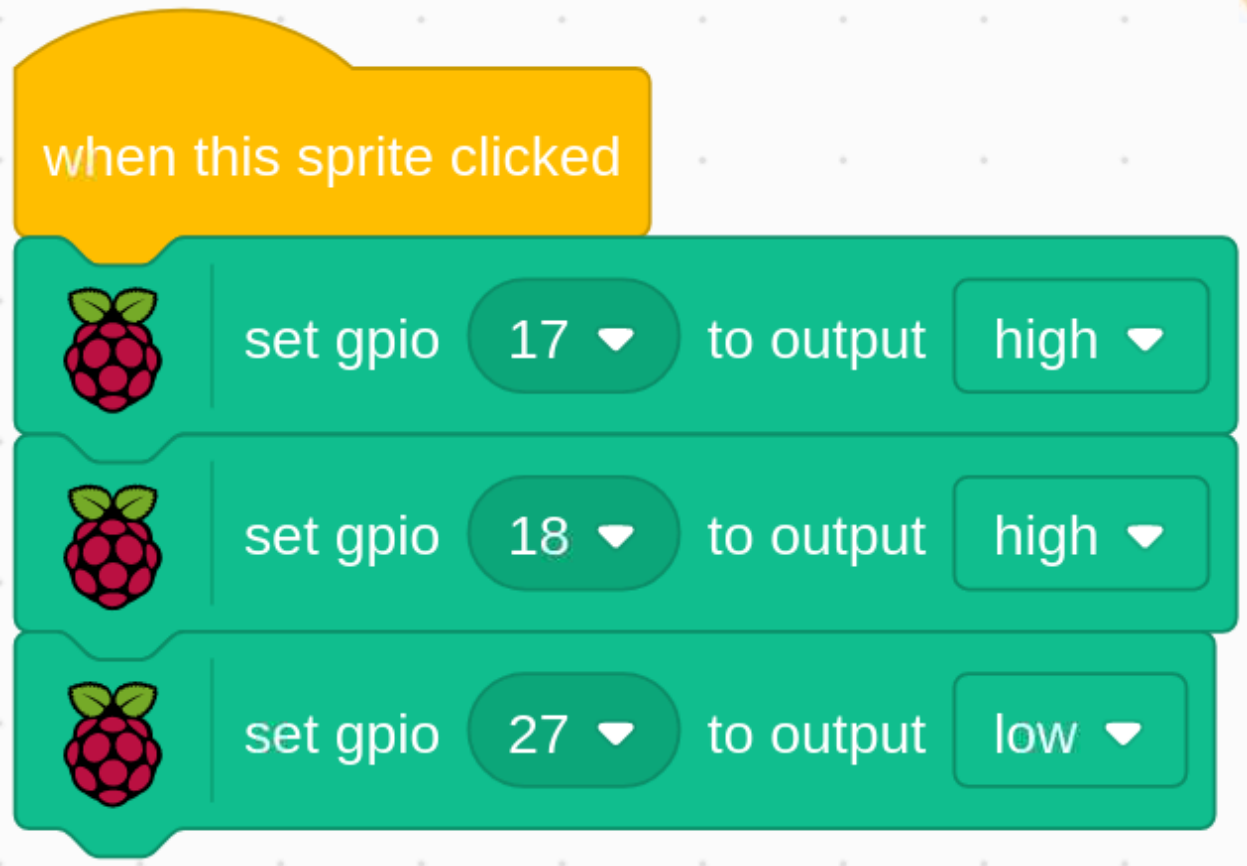
Das RGB-Farbmodell ist ein additives Farbmodell, bei dem rotes, grünes und blaues Licht auf verschiedene Weise addiert werden, um eine breite Palette von Farben zu reproduzieren.

Additive Farbmischung: Wenn man Rot zu Grün addiert, erhält man Gelb; wenn man Grün zu Blau addiert, erhält man Cyan; wenn man Blau zu Rot addiert, erhält man Magenta; wenn man alle drei Grundfarben addiert, erhält man Weiß.



Eine RGB-LED ist eine Kombination von 3 LEDs (rote LED, grüne LED, blaue LED) in nur einem Gehäuse, Sie können fast jede Farbe durch die Kombination dieser drei Farben erzeugen. Sie hat 4 Pins, von denen einer GND ist, und die anderen 3 Pins steuern jeweils 3 LEDs.

Der Code, um die RGB-LED gelb leuchten zu lassen, lautet also wie folgt.

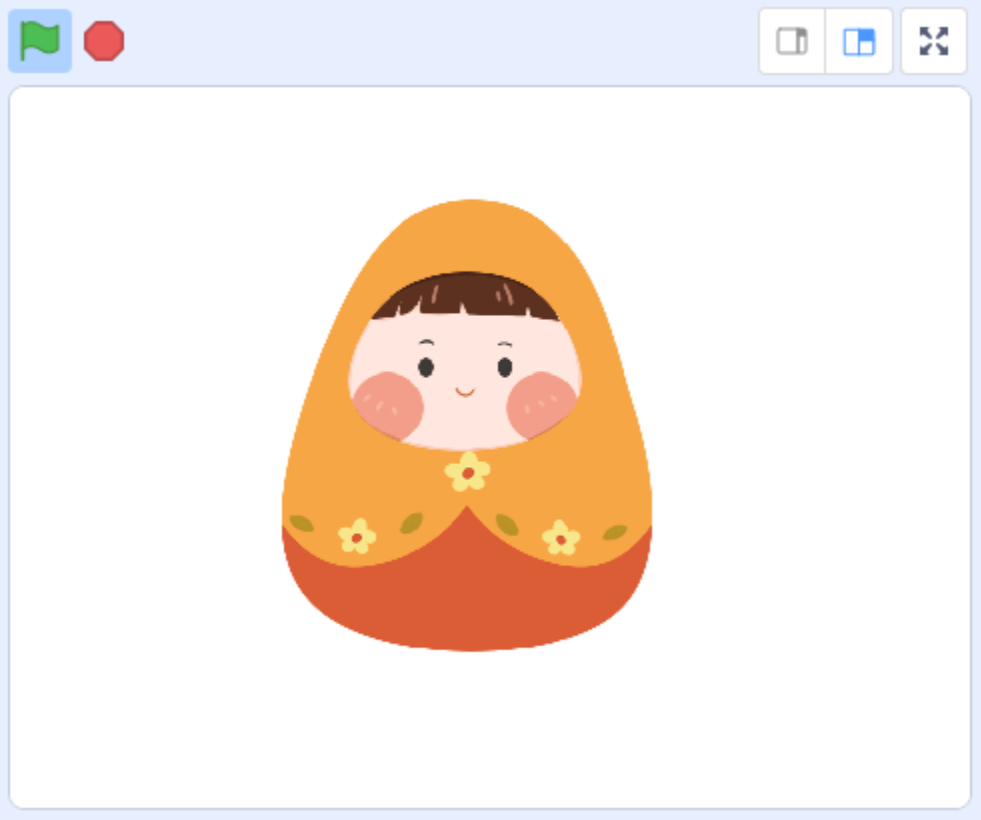


Wenn das Ball-Sprite (gelber Ball) angeklickt wird, setzen wir gpio17 hoch (rote LED an), gpio18 hoch (grüne LED an) und gpio27 niedrig (blaue LED aus), so dass die RGB-LED gelb leuchtet.

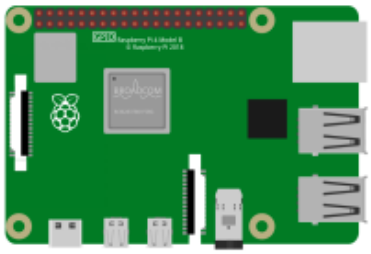
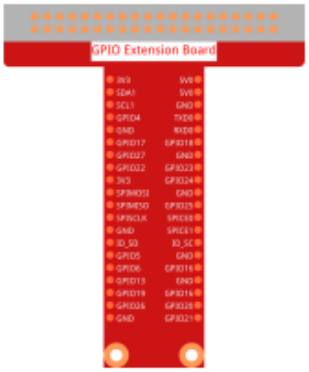
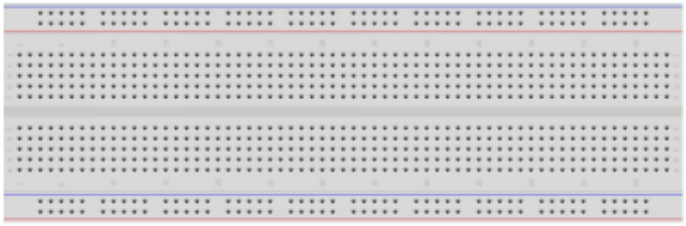
Du kannst auf die gleiche Weise Codes in andere Sprites schreiben, um die RGB-LEDs in den entsprechenden Farben aufleuchten zu lassen.

1.3 Becher

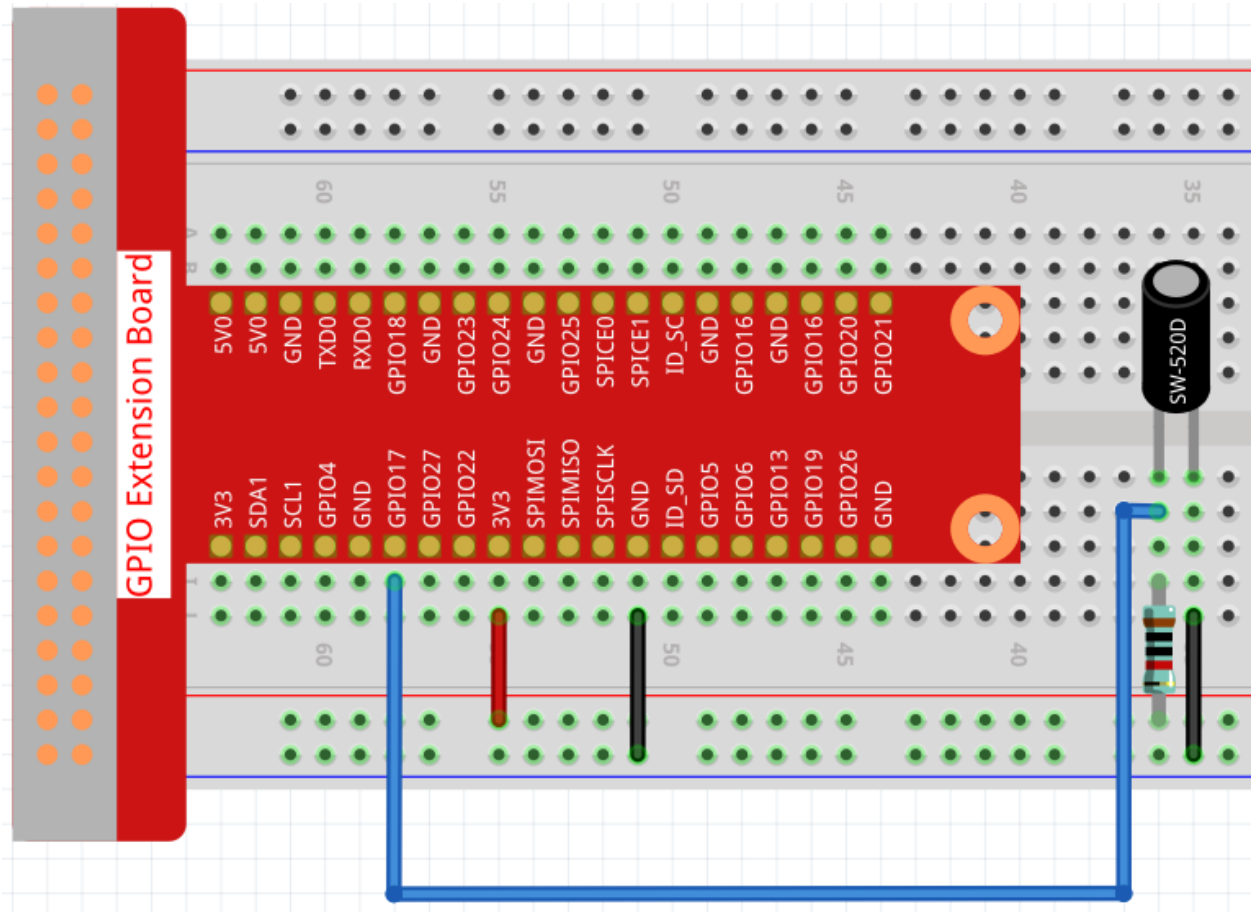
In diesem Projekt werden wir ein kippschaltergesteuertes Tumbler-Spielzeug bauen.



Erforderliche Komponenten

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Tilt Switch
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		1 * Resistor(10kΩ)

Bauen Sie den Stromkreis auf



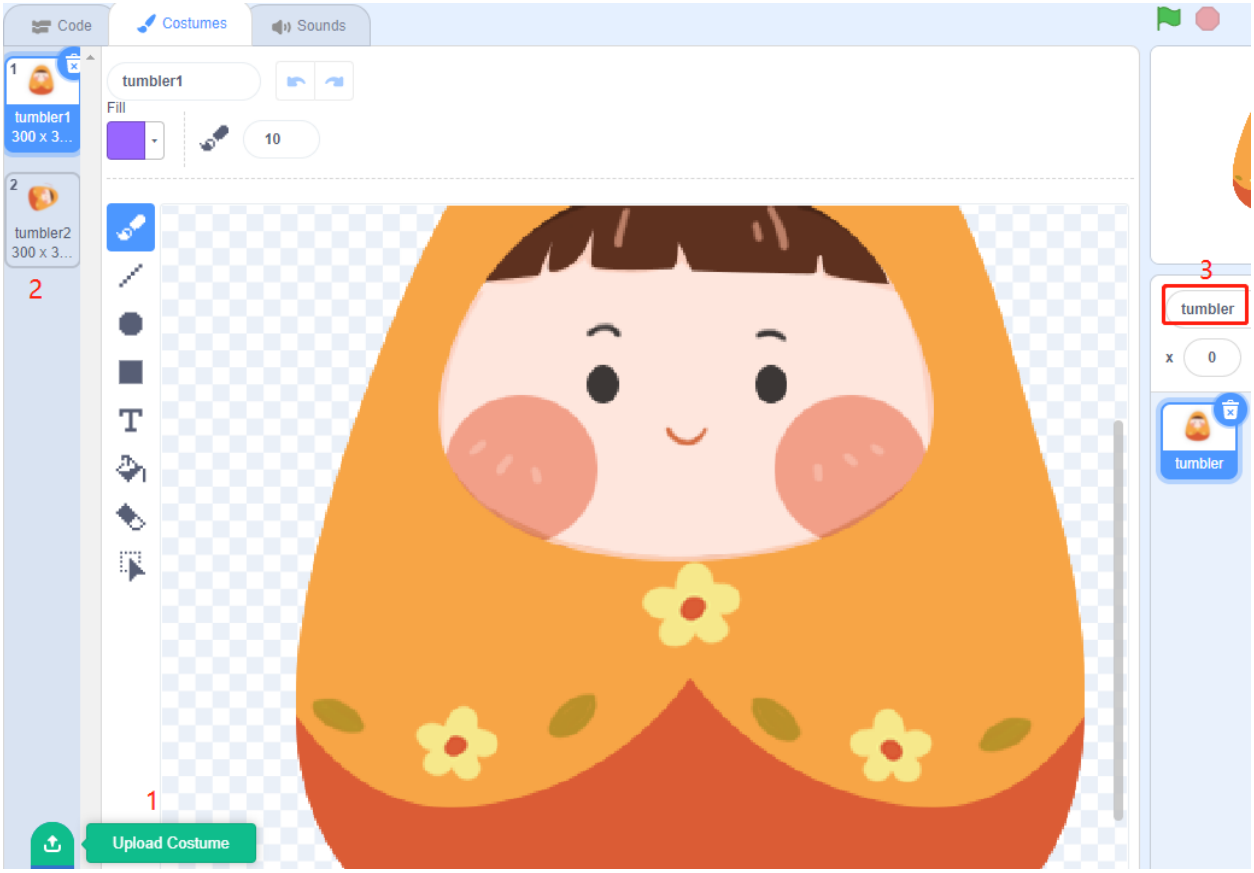
Laden Sie den Code und sehen Sie, was passiert s

Lade die Code-Datei (`1.3_tumbler.sb3`) in Scratch 3.

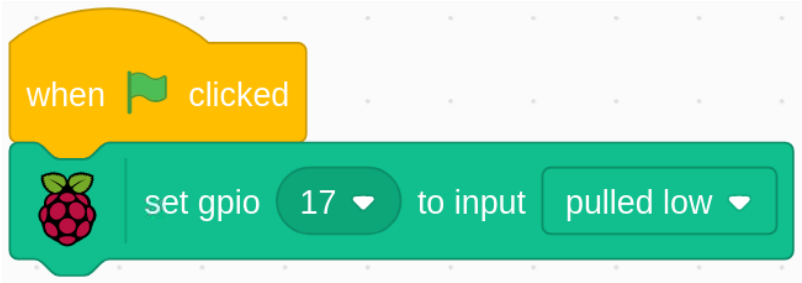
Wenn der Kippschalter senkrecht steht, bleibt der Becher stehen. Wenn Sie ihn kippen, fällt auch der Becher um. Stellen Sie ihn wieder aufrecht, und der Becher steht wieder aufrecht.

Tipps zu Figur

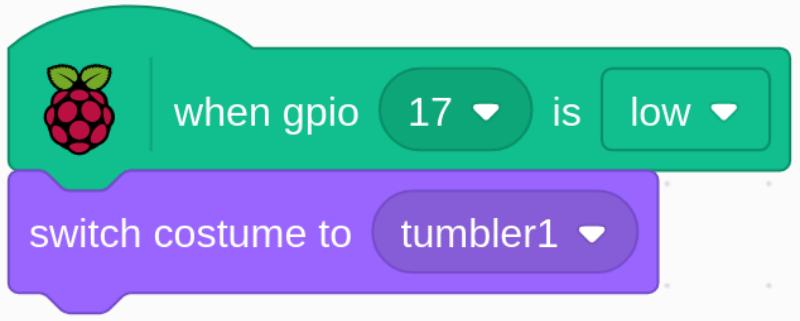
Wähle Sprite1 und klicke oben links auf **Costumes**; Laden Sie `tumbler1.png` und `tumbler2.png` aus dem Pfad `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` über die Schaltfläche Kostüm hochladen hoch; Löschen Sie die 2 Standardkostüme und benennen Sie das Sprite in Tumbler um.



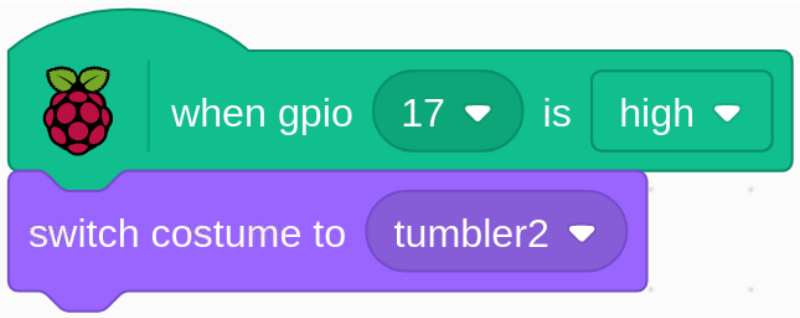
Tipps zu Codes



Wenn die grüne Flagge angeklickt wird, wird der Ausgangszustand von gpio17 auf low gesetzt.



Wenn Pin17 niedrig ist (der Kippschalter ist aufrecht), schalten wir das Kostüm des Tumbler-Sprites auf Tumbler1 (aufrechter Zustand).

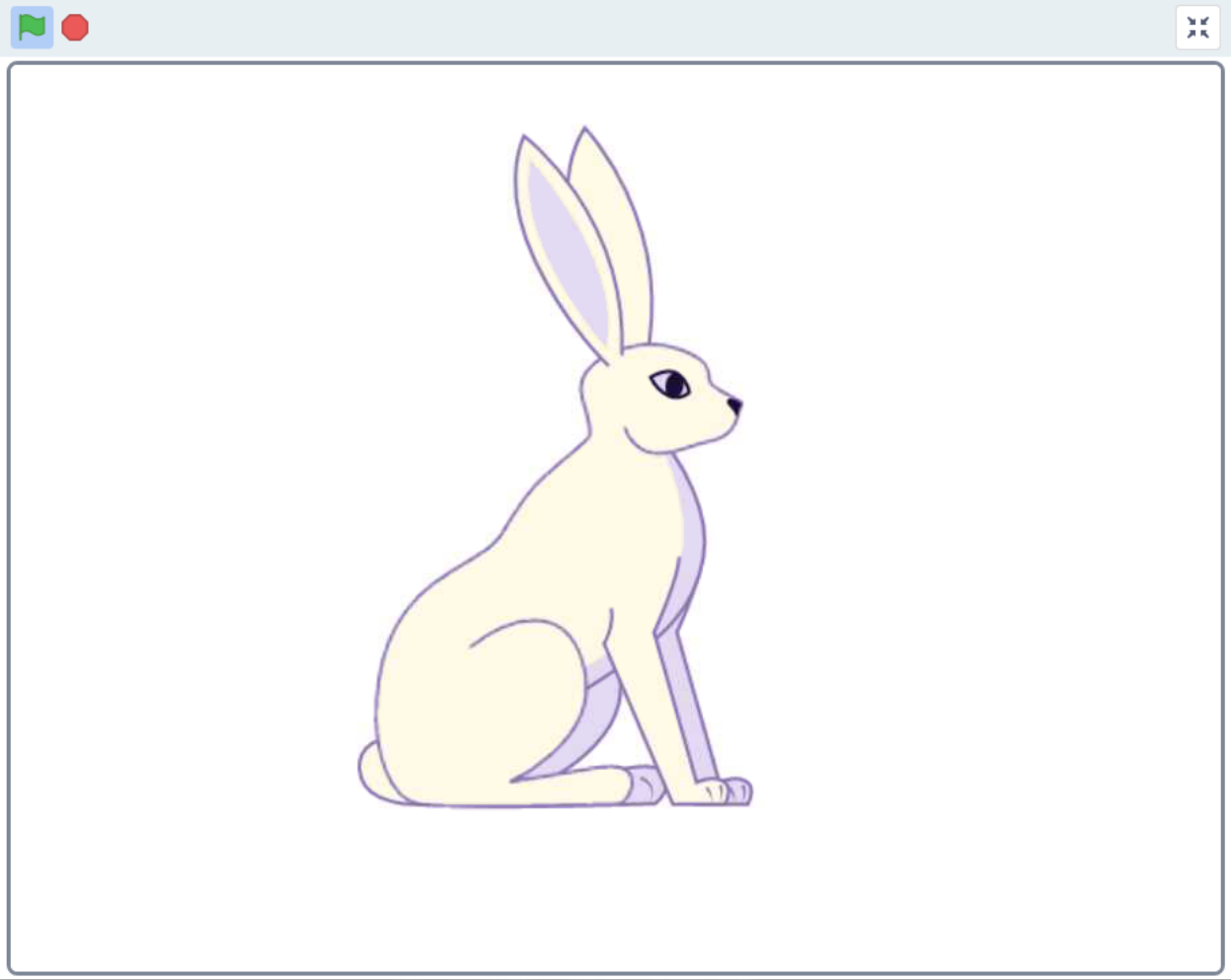


Wenn Pin17 High ist (Kippschalter ist gekippt), schaltet das Kostüm des Tumbler-Sprites auf Tumbler2 (Kippzustand).

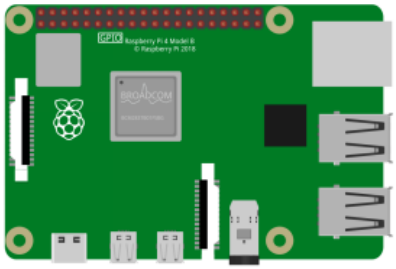
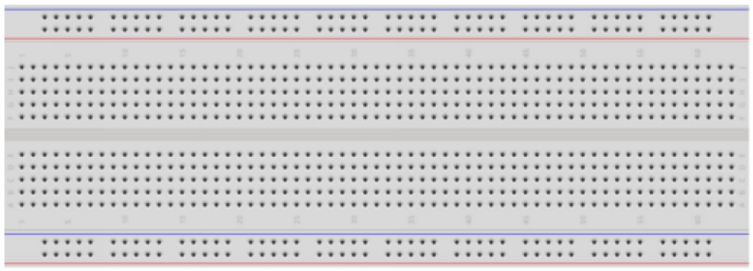
1.4 Hase

Heute werden wir Button, Raspberry Pi und Scratch verwenden, um einen Hasen mit verschiedenen Veränderungen zu erstellen!

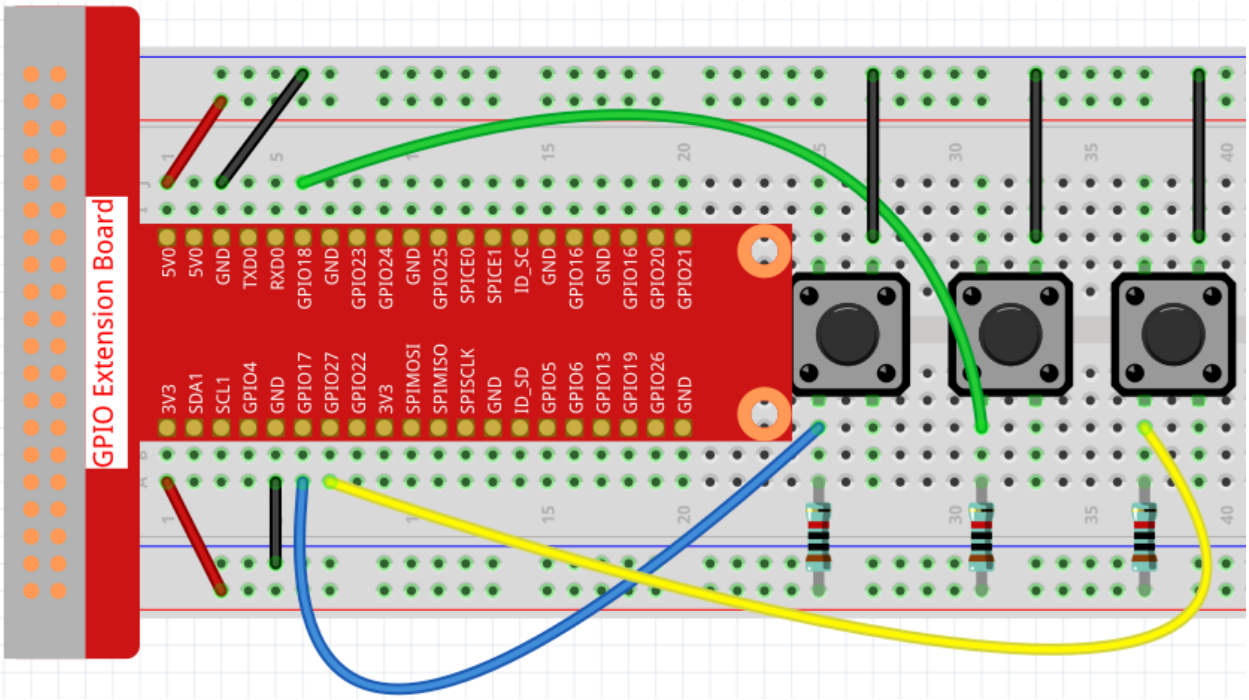
Wenn wir den ersten Knopf drücken, ändert der Hase im Bühnenbereich seine Körperfarbe; wenn wir den zweiten Knopf drücken, ändert der Hase seine Körpergröße; wenn wir den dritten Knopf drücken, macht der Hase einen Schritt nach vorne.



Erforderliche Komponenten

1 * Raspberry Pi 	1 * T-Extension Board	3* Resistor 10KΩ
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Button

Bauen Sie den Stromkreis auf



Laden Sie den Code und sehen Sie, was passiert

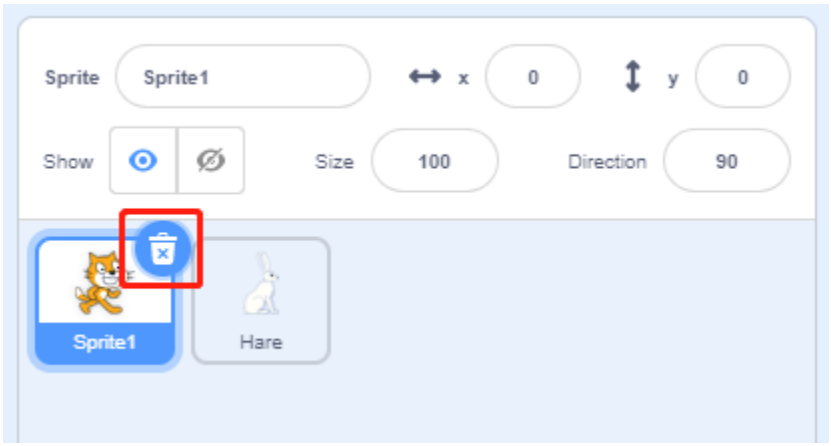
Lade die Code-Datei (`1.4_hare.sb3`) in Scratch 3.

Jetzt kannst du versuchen, jeden der 3 Knöpfe zu drücken, um zu sehen, wie sich der Hase auf der Bühne verändert.

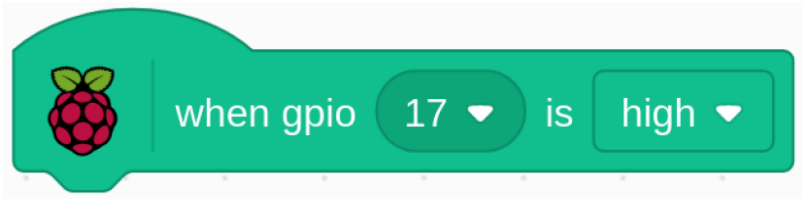
Tipps zu Figur

Klicken Sie auf die Schaltfläche **Choose a Sprite** in der unteren rechten Ecke des Sprite-Bereichs, geben Sie **Hare** in das Suchfeld ein, und klicken Sie dann, um es hinzuzufügen.

Löschen Sie Figur1.

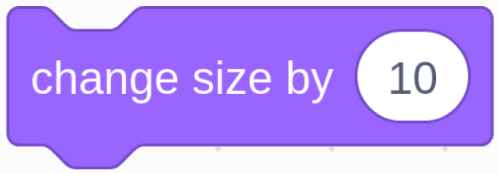


Tipps zu Codes



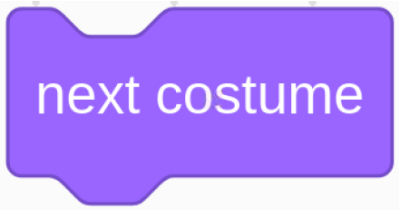
Dies ist ein Ereignisblock, der ausgelöst wird, wenn der Pegel von GPIO17 hoch ist, was bedeutet, dass die Taste zu diesem Zeitpunkt gedrückt wird.

Dies ist ein Block, um die Farbe von **Hare** zu ändern, der Bereich des Wertes ist 0 ~ 199, über 199 wird wieder von 0 geändert.



Dies ist ein Block, der verwendet wird, um die Größe für das Sprite zu ändern. Je höher der Wert, desto größer das Sprite.

Bemerkung: Das Sprite ist auch nicht unendlich groß, und seine maximale Größe hängt von der Größe des Originalbildes ab.



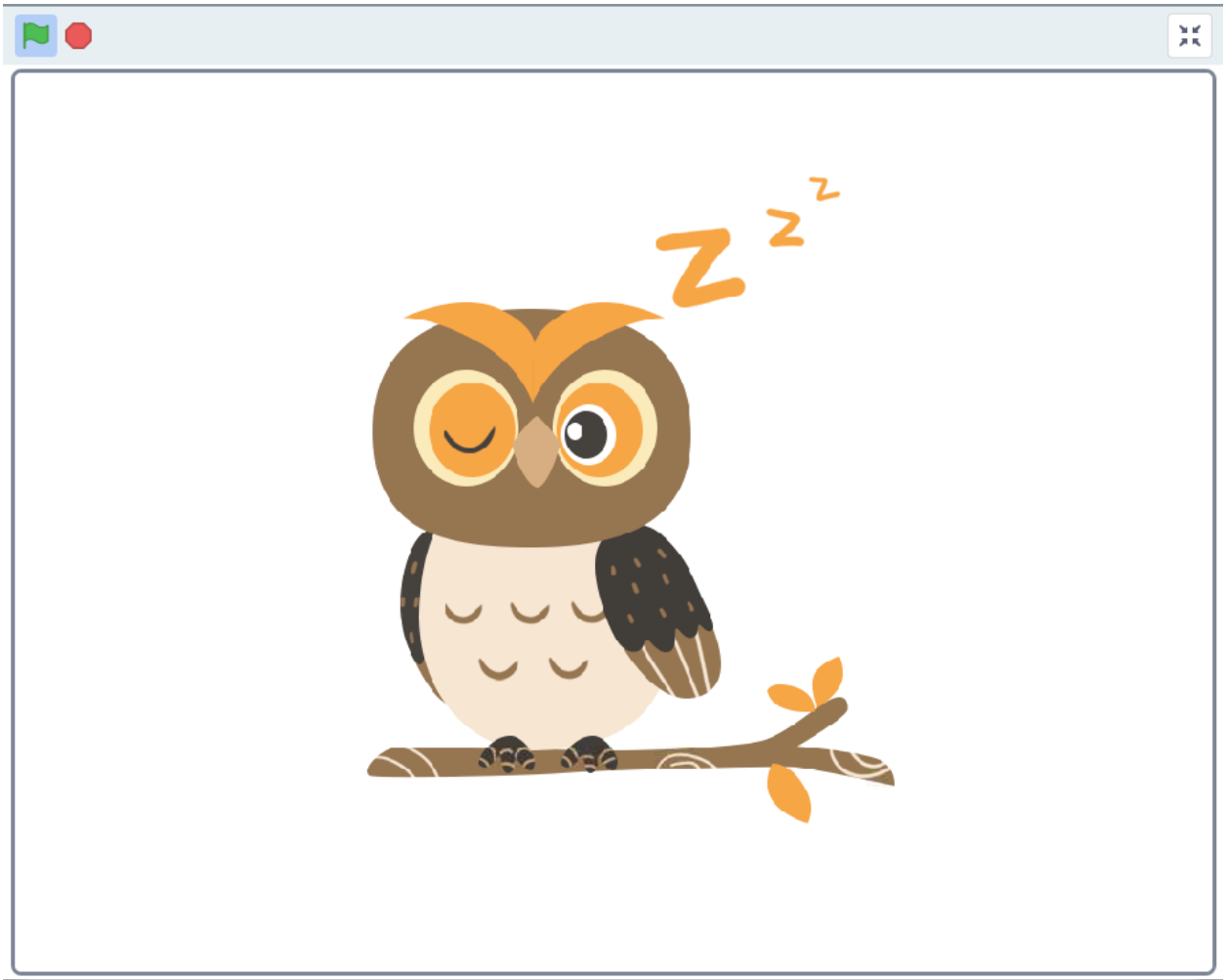
Dies ist ein Block, der die Kostüme der Sprites wechselt. Wenn **Hare** sein Kostüm wechselt, führt er eine Reihe von kohärenten Aktionen aus. In diesem Projekt soll **Hare** zum Beispiel einen Schritt nach vorne machen.

1.5 Weck die Eule auf

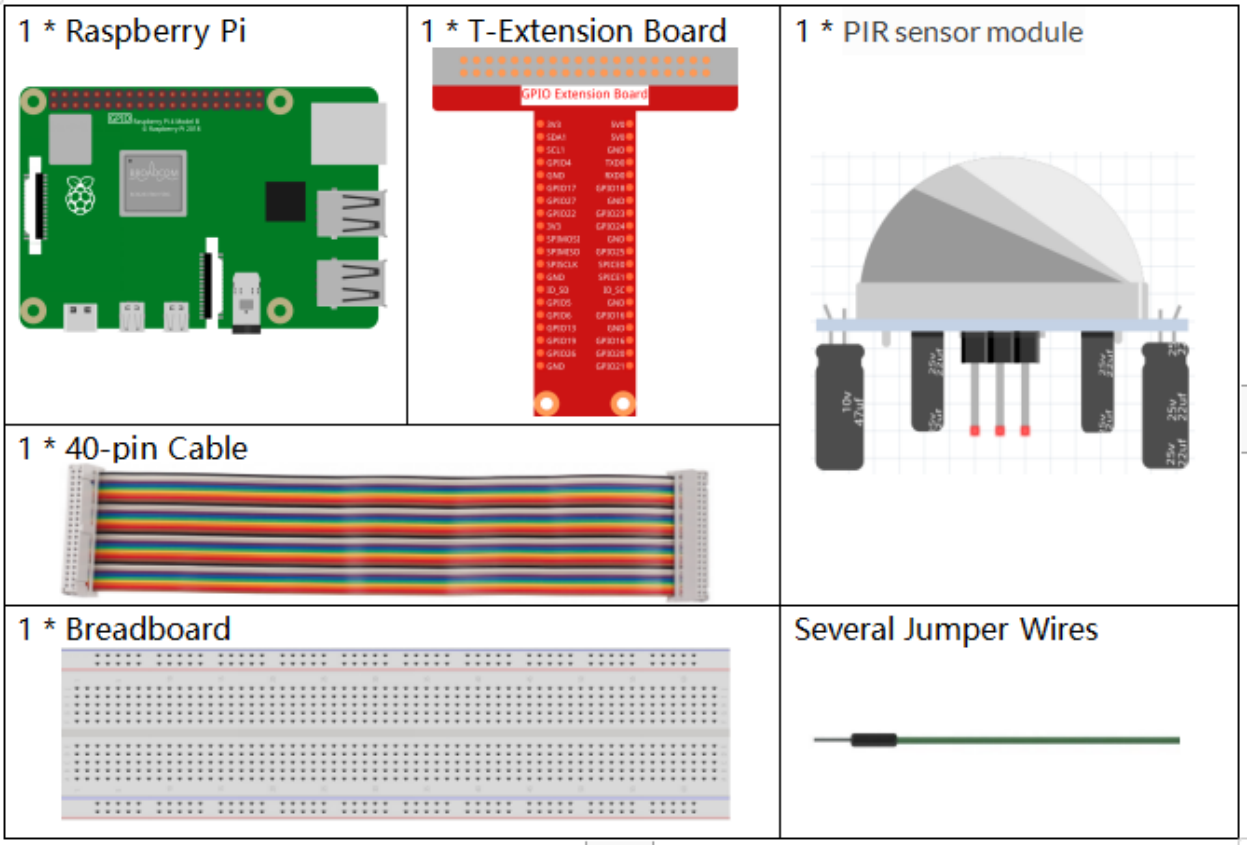
Heute werden wir ein Spiel spielen, bei dem es darum geht, die Eule zu wecken.

Wenn sich jemand dem PIR-Sensormodul nähert, erwacht die Eule aus dem Schlaf.

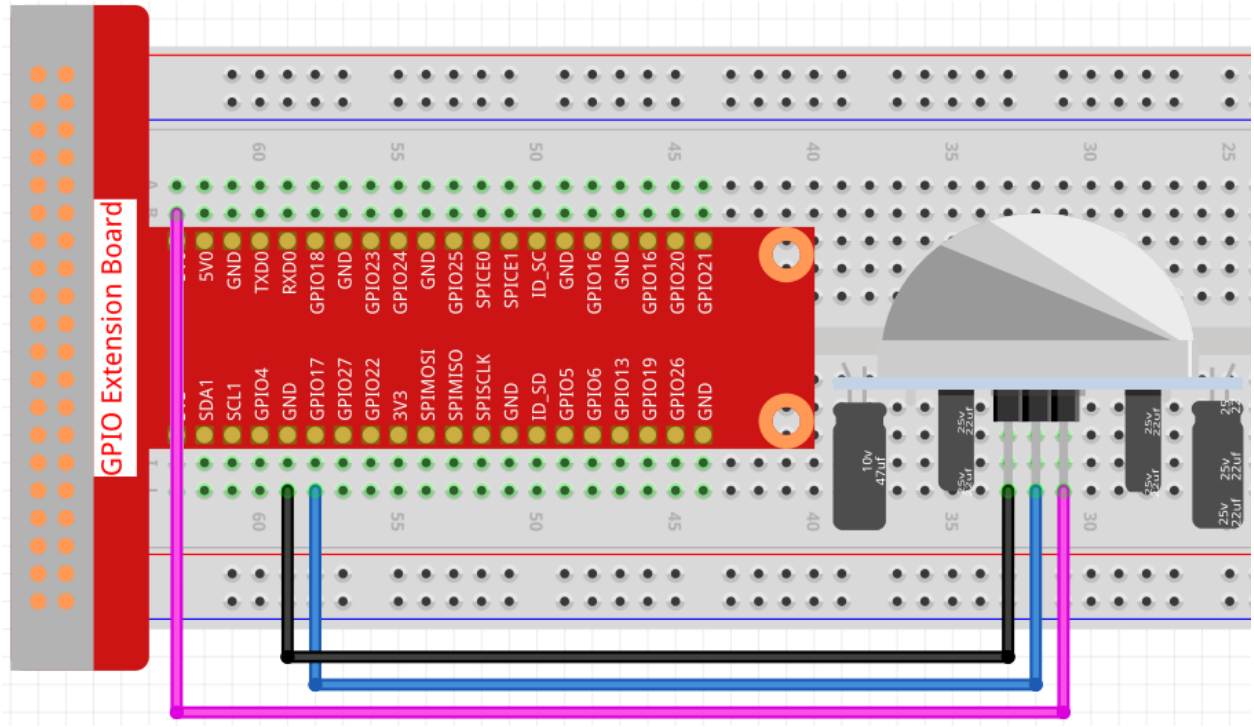
There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. To make the PIR module work better, you You need to turn both of them counterclockwise to the end.



Erforderliche Komponenten



Bauen Sie den Stromkreis auf



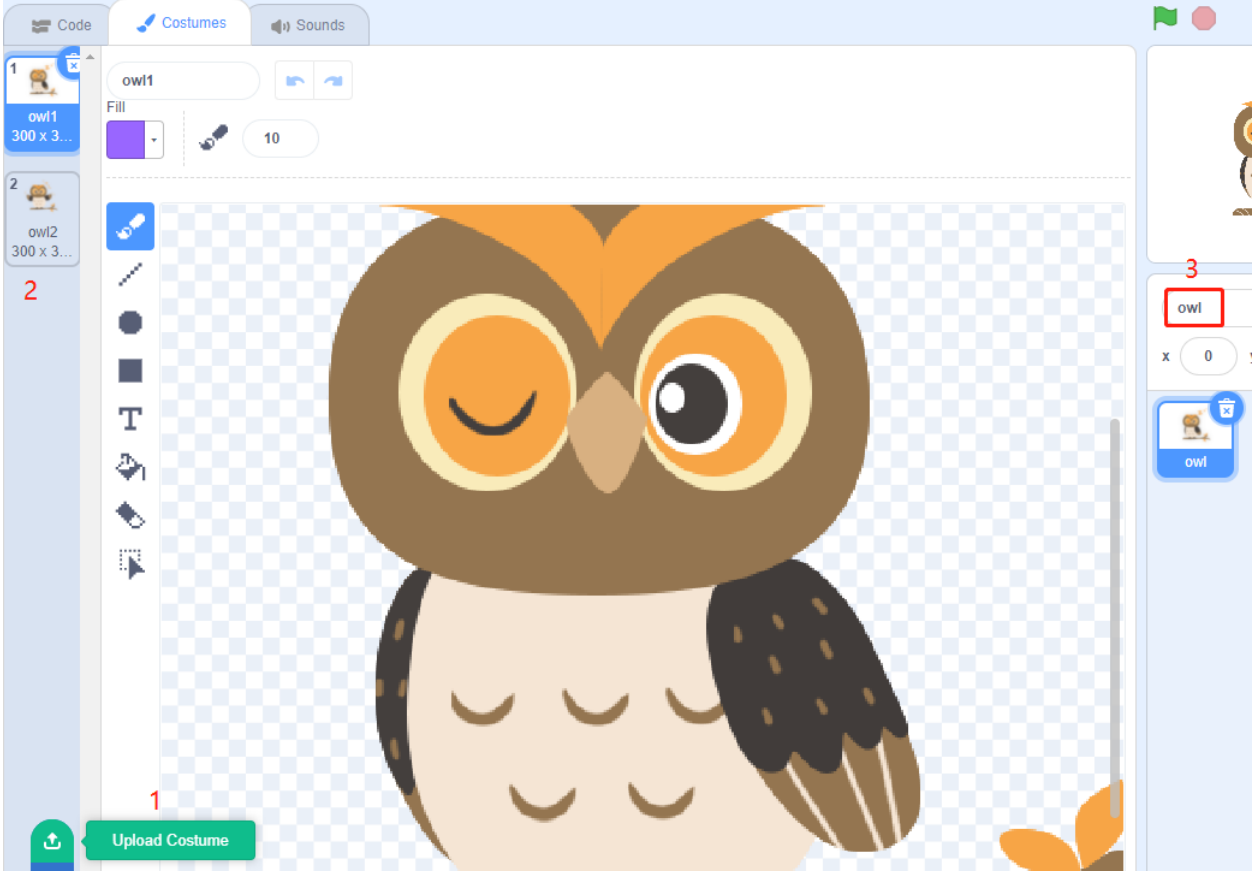
Laden Sie den Code und sehen Sie, was passiert

Laden Sie die Codedatei (`1.5_wake_up_the_owl.sb3`) in Scratch 3.

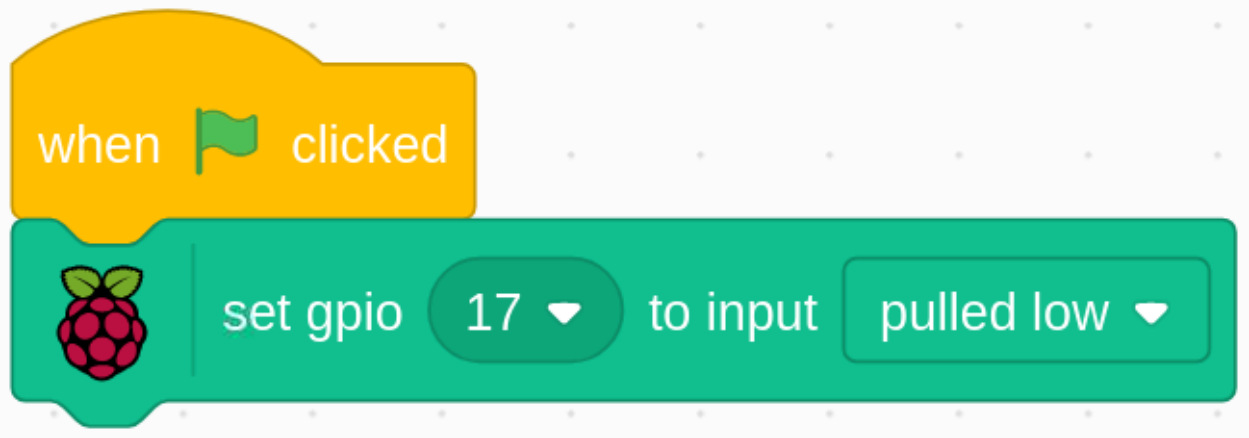
Wenn Sie sich dem PIR-Sensormodul nähern, sehen Sie, wie die Eule auf dem Bühnenbereich ihre Flügel öffnet und aufwacht, und wenn Sie gehen, schläft die Eule wieder ein.

Tipps zu Figur

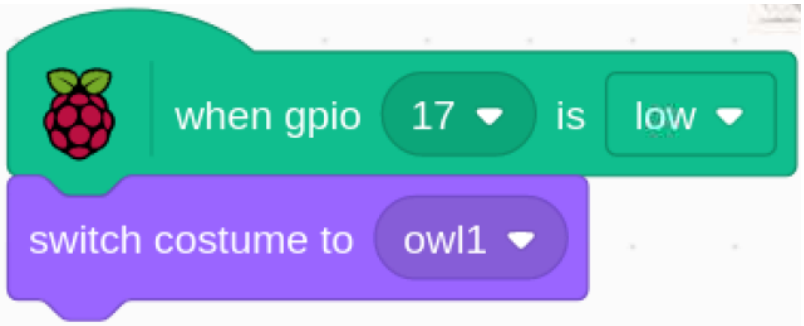
Wählen Sie Sprite1 und klicken Sie auf **Costumes** in der oberen linken Ecke; laden Sie **owl1.png** und **owl2.png** aus dem Pfad `home/pi/davinci-kit-for-raspberry-pi/scratch/picture` über die Schaltfläche **Upload Costume** hoch; löschen Sie die 2 Standardkostüme und benennen Sie das Sprite in **owl** um.



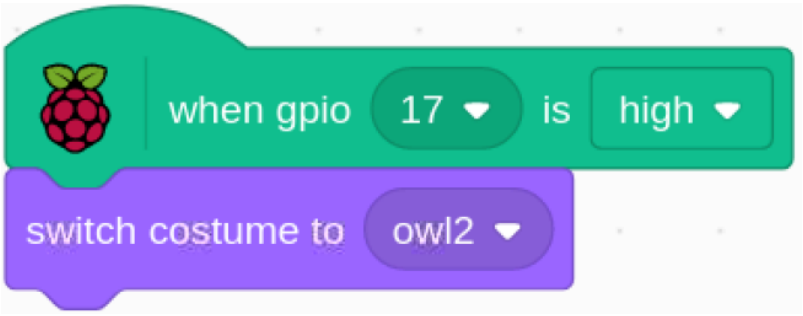
Tipps zu Codes



Wenn die grüne Flagge angeklickt wird, wird der Ausgangszustand von gpio17 auf low gesetzt.



Wenn Pin17 niedrig ist (es nähert sich niemand), schalte das Kostüm des Eulen-Sprites auf Eule1 (Schlafzustand).

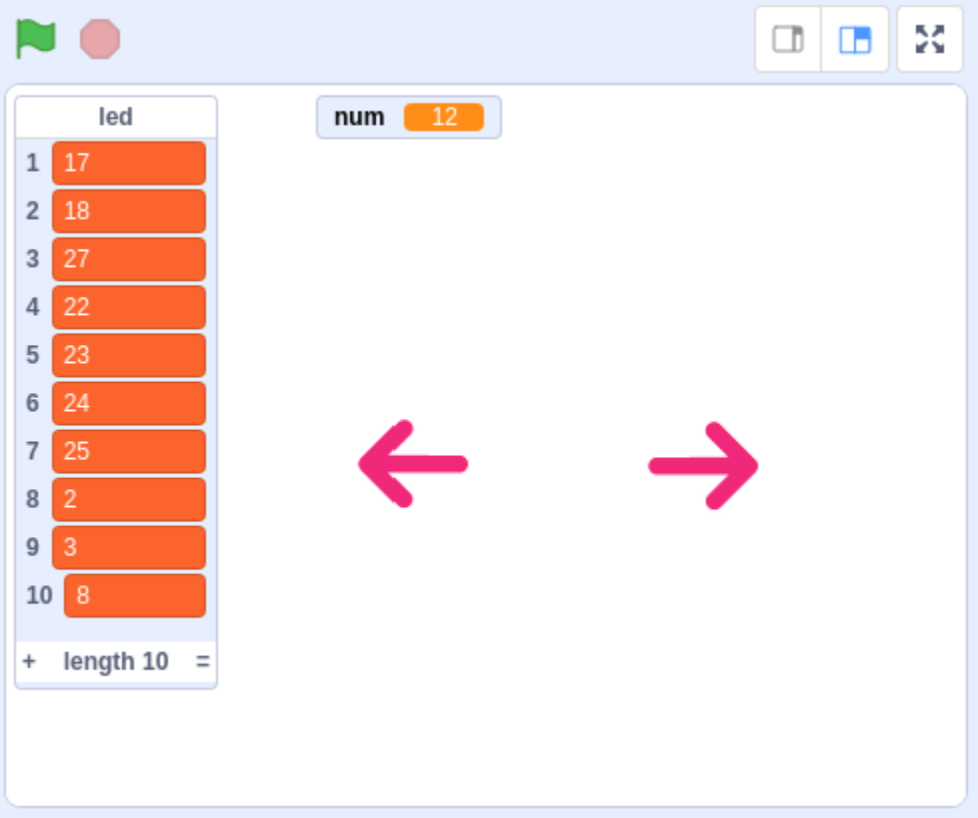


Wenn Pin17 hoch ist (jemand nähert sich), schalten wir das Kostüm des Eulen-Sprites auf Eule2 (Wachzustand).

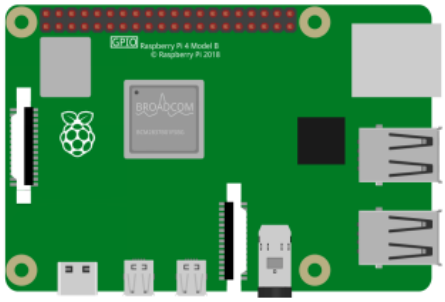
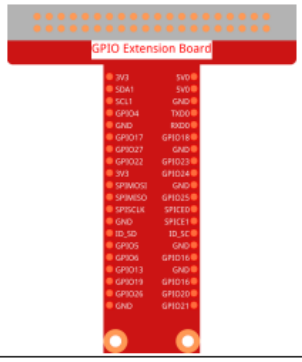
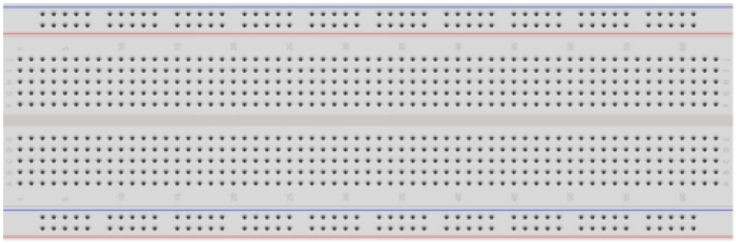
1.6 Water Lamp

Heute werden wir mit LED-Balkengrafik, Raspberry Pi und Scratch eine Wasserlampe bauen.

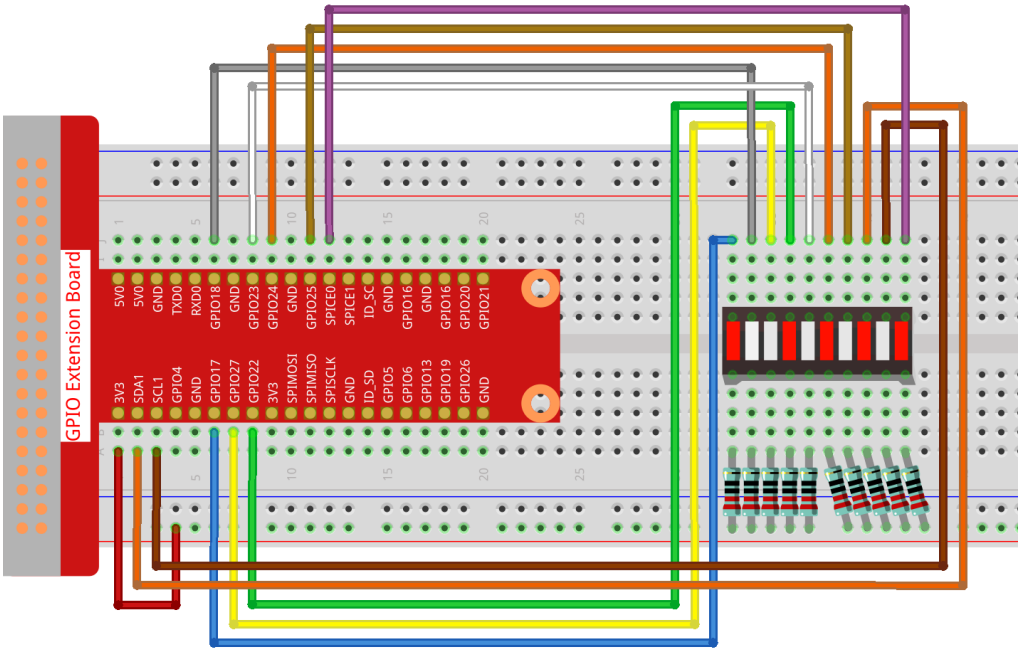
Die LED-Balkenanzeige leuchtet in der Reihenfolge der Pfeile auf der Bühne auf.


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

Erforderliche Komponenten

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

Bauen Sie den Stromkreis auf



Laden Sie den Code und sehen Sie, was passiert

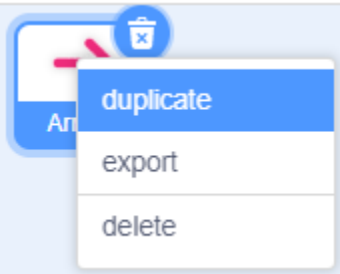
Lade die Code-Datei (`1.6_water_lamp.sb3`) von deinem Computer in Scratch 3.

Wenn Sie auf **Arrow1** klicken, leuchten die LEDs auf der LED-Leiste nacheinander von links nach rechts auf (eine nach der anderen) und gehen dann aus. Klicken Sie auf **Arrow2** und die LEDs leuchten in der umgekehrten Reihenfolge auf.

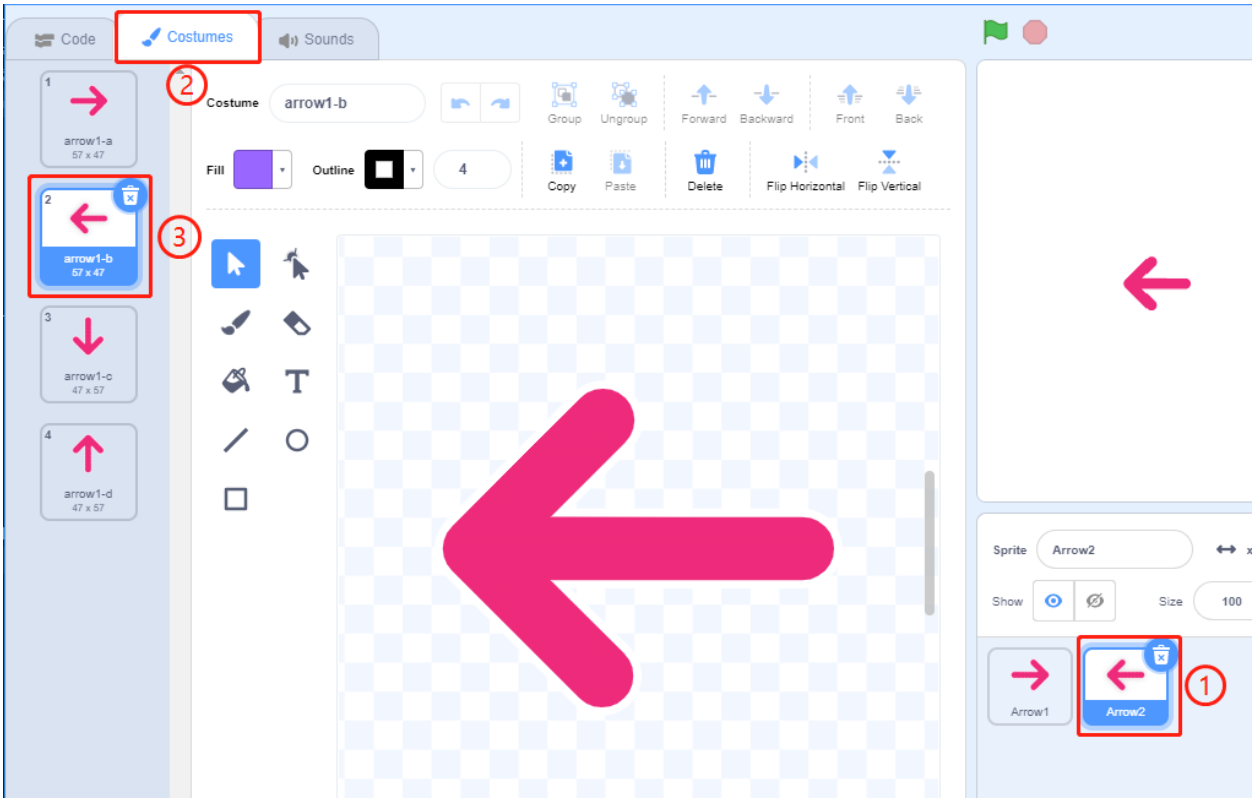
Tipps zu Figur

Löschen Sie das Standard-Figur und wählen Sie das Figur **Arrow1** .

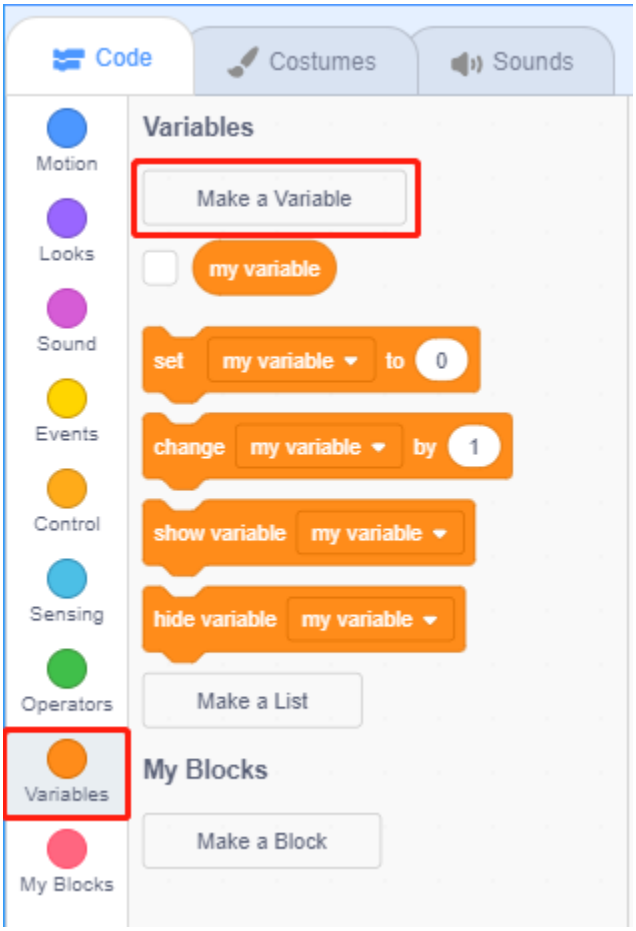
Hier benötigen wir 2 **Arrow1** -Sprites, die mit dem Duplikat-Button erstellt werden können.



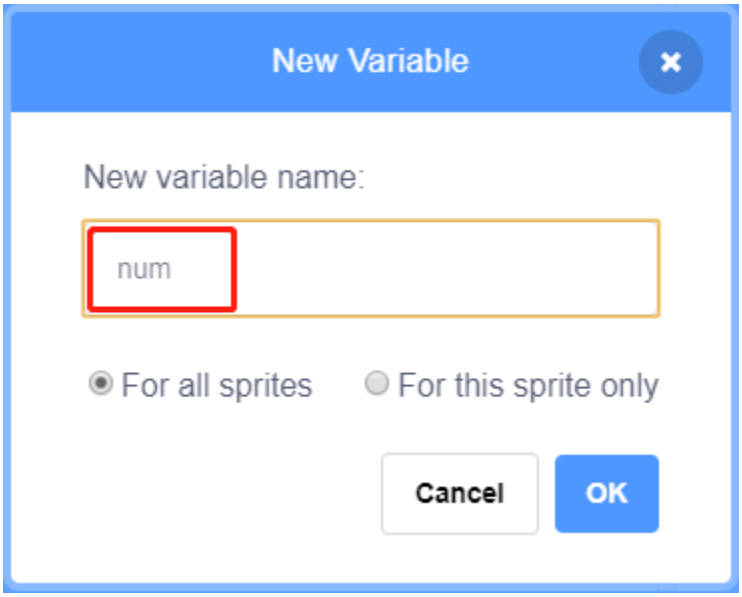
Klicken Sie auf das Sprite **Arrow 2** und ändern Sie die Richtung des Pfeils, indem Sie Kostüm 2 wählen.



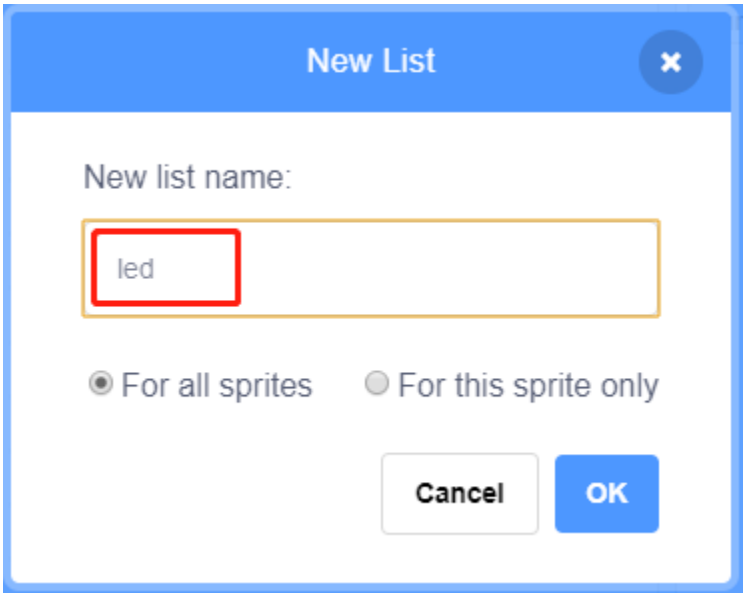
Nun wollen wir eine Variable erstellen.



Benennen Sie es als `num` .



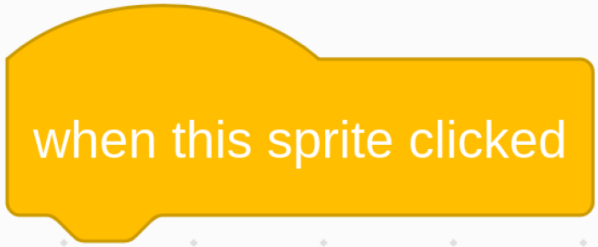
Gehen Sie genauso vor, um eine Liste namens `led` zu erstellen.



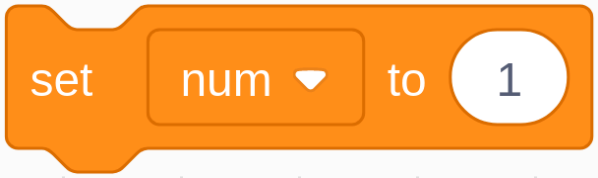
Nach dem Hinzufügen sollten Sie die Variable `num` und die Liste `led` im Bühnenbereich sehen.

Klicken Sie auf +, um 10 Listenelemente hinzuzufügen, und geben Sie die Pin-Nummern der Reihe nach ein (17,18,27,22,23,24,25,2,3,8).

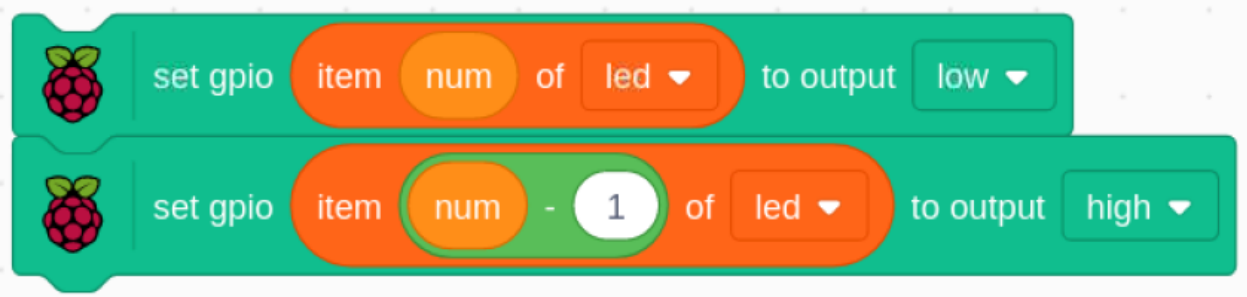
Tipps zu Codes



Dies ist ein Ereignisblock, der ausgelöst wird, wenn das aktuelle Sprite angeklickt wird.



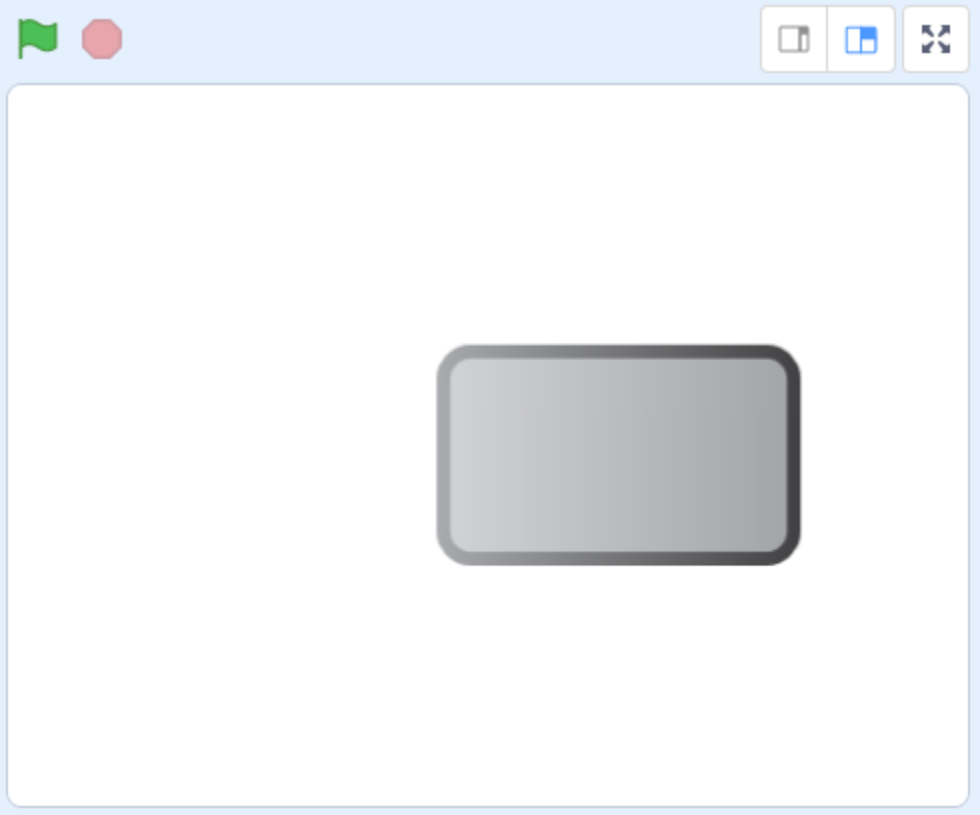
Der Anfangswert der Variablen `num` bestimmt, welche LED zuerst leuchtet.



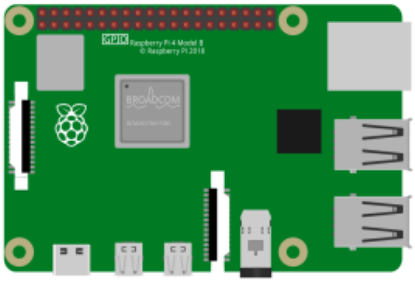
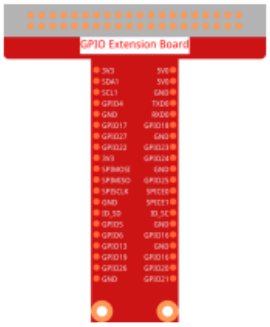
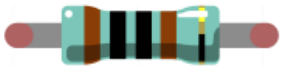
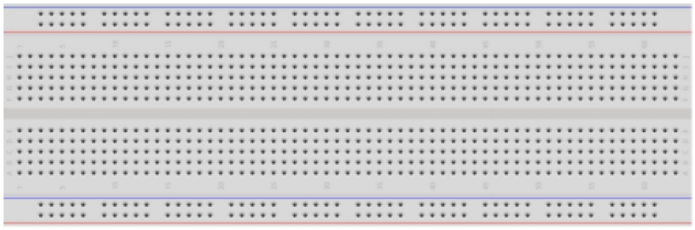
Setzen Sie den Pin mit der Nummer `num` in der LED-Liste auf low, um die LED zum Leuchten zu bringen, und setzen Sie dann den Pin mit der Nummer `num-1` auf high, um die vorherige LED auszuschalten.

1.7 Türklingel

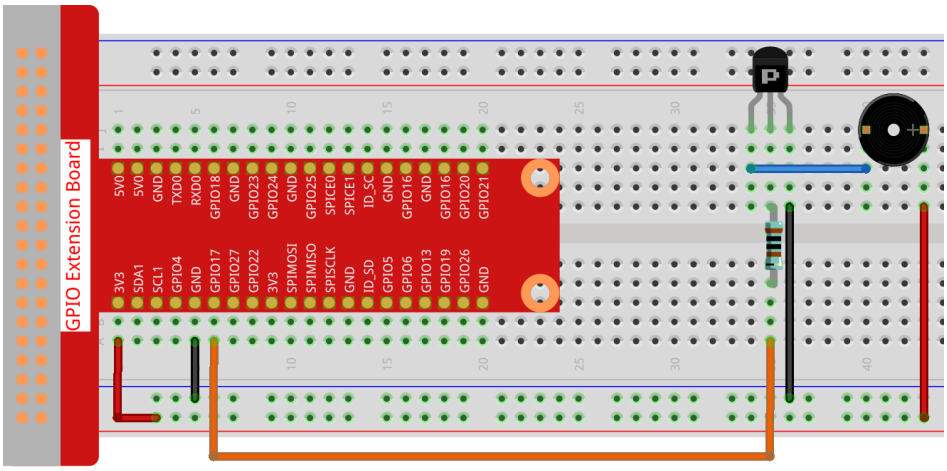
Heute werden wir eine Türklingel bauen, klicke auf das Button3-Sprite auf der Bühne, der Summer ertönt; klicke erneut, der Summer hört auf zu ertönen.



Erforderliche Komponenten

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

Bauen Sie den Stromkreis auf



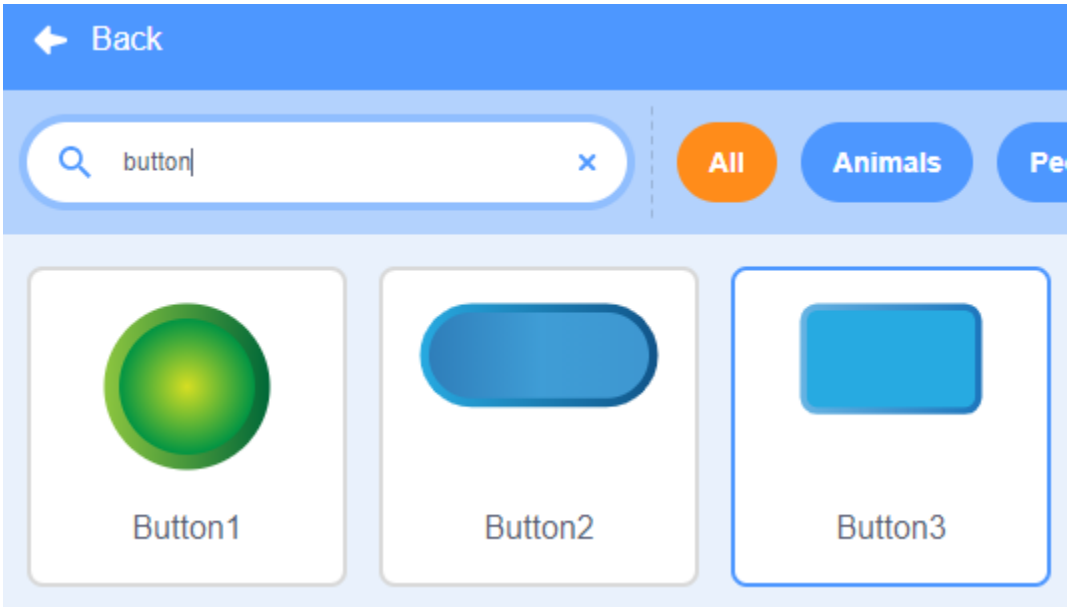
Laden Sie den Code und sehen Sie, was passiert

Lade die Code-Datei (`1.7_doorbell.sb3`) in Scratch 3.

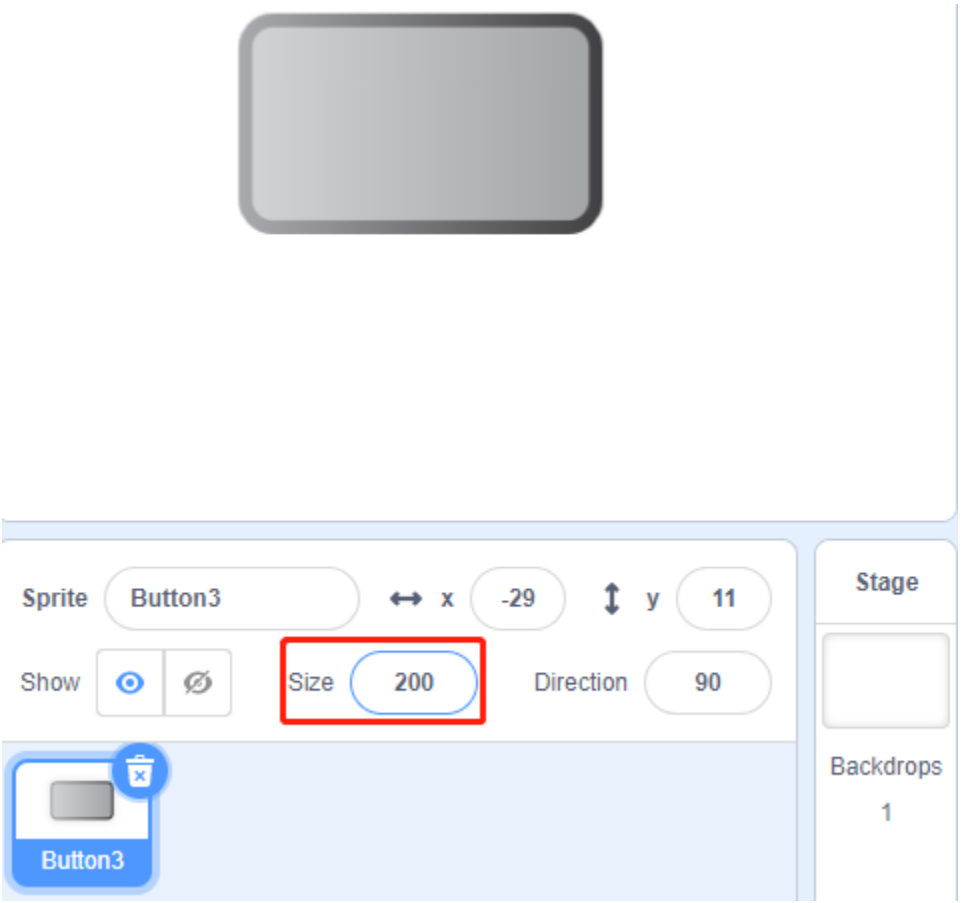
Klicken Sie auf die grüne Flagge auf der Bühne. Wenn wir auf das Sprite **Button3** klicken, wird es blau und der Summer ertönt. Wenn wir erneut klicken, wird das Sprite Button3 wieder grau und der Summer hört auf zu ertönen.

Tipps zu Figur

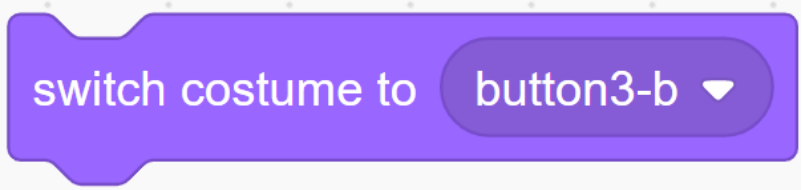
Löschen Sie das Standard-Sprite und wählen Sie dann das Sprite **Button3** .



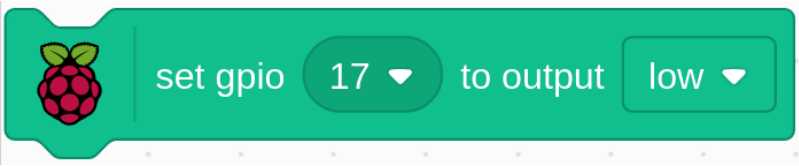
Dann setzen Sie die Größe auf 200.



Tipps zu Codes



Mit diesem Block kannst du das Kostüm des Sprites wechseln.

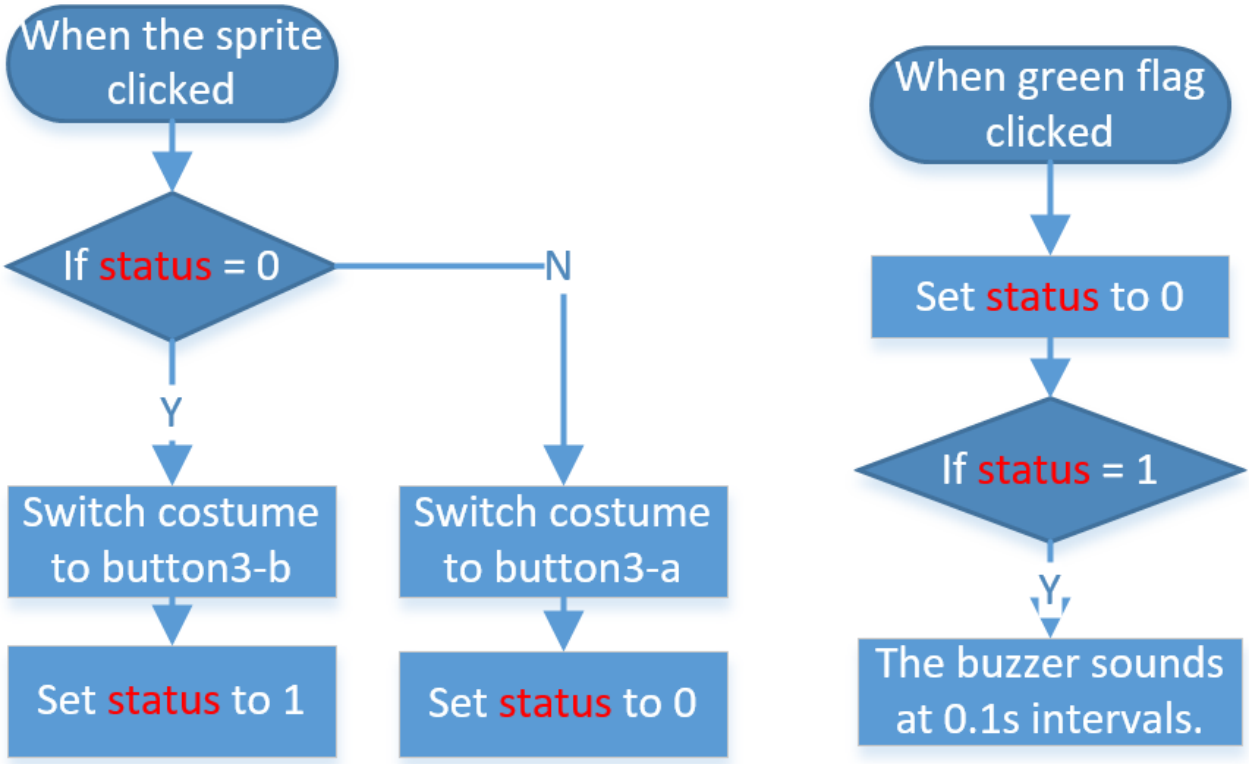


Setzen Sie gpio17 auf niedrig, damit der Summer ertönt; stellen Sie ihn auf hoch und der Summer ertönt nicht.

Hier wird der Schalter status verwendet, und wir verwenden ein Flussdiagramm, um Ihnen das Verständnis des gesamten Codes zu erleichtern.

Wenn das grüne Flag angeklickt wird, wird der **Status** zuerst auf 0 gesetzt und gewartet, bis das Sprite zu diesem Zeitpunkt angeklickt wird; Wenn **button3** geklickt wird, wechselt es zum Kostüm als **button-b** Kostüm (blau) und der **Status** wird auf 1 gesetzt. Wenn das Hauptprogramm den **Status** erhält als 1 wird der Summer im 0,1s-Intervall ertönt.

Wenn **button3** erneut geklickt wird, wechselt es zu **button-a** Kostüm (grau) und **status** wird wieder auf 0 gesetzt.



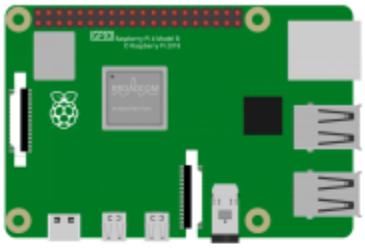
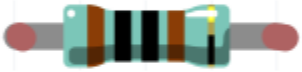
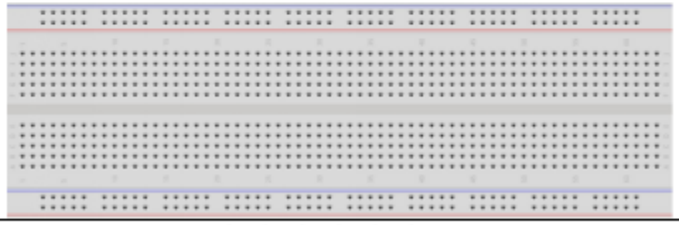
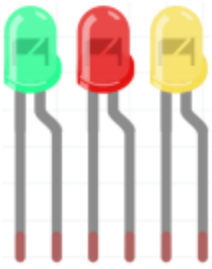
1.8 123 Holzmann

Heute spielen wir 123 Holzmann.

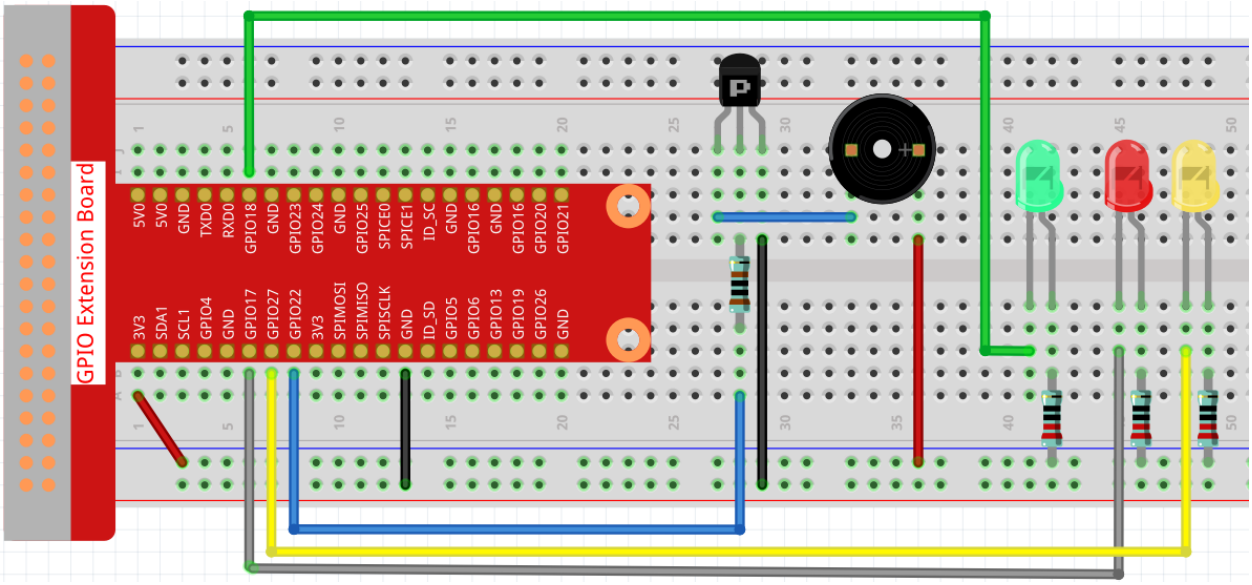
Klicken Sie auf die grüne Flagge, um das Spiel zu starten, halten Sie die rechte Pfeiltaste auf der Tastatur gedrückt, um das Sprite nach rechts zu bewegen. Wenn das grüne Licht leuchtet, kann sich das Sprite bewegen; aber wenn die rote LED leuchtet, müssen Sie die Bewegung des Sprites stoppen; andernfalls klingelt der Summer weiter.

status
1

Erforderliche Komponenten

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

Baue die Schaltung



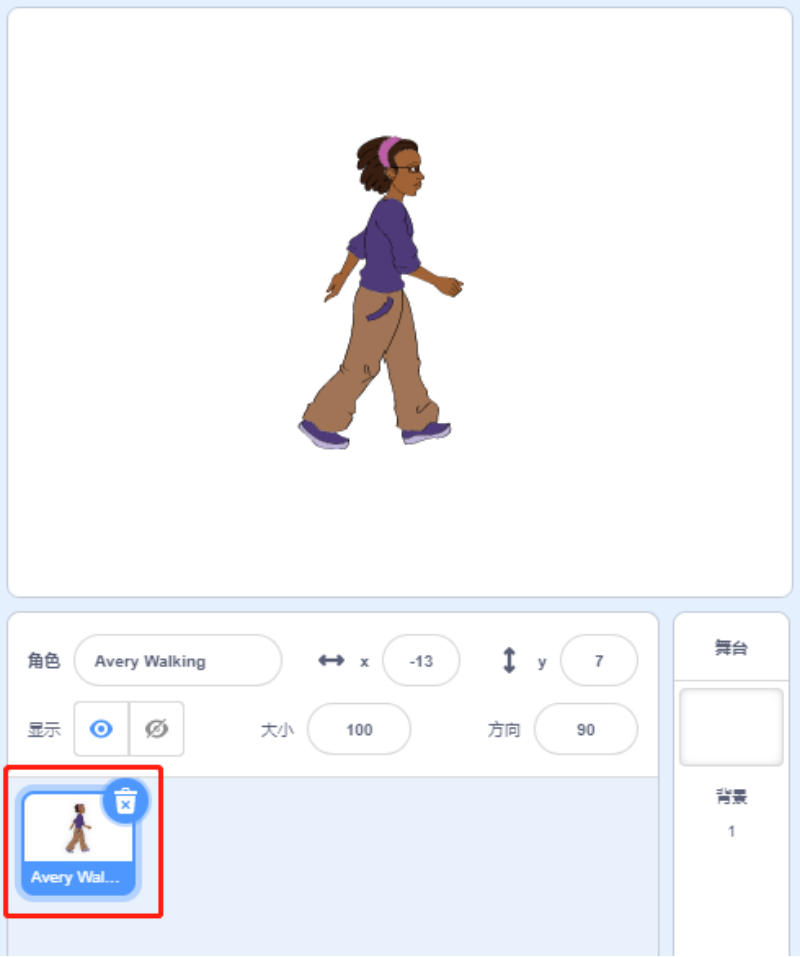
Laden Sie den Code und sehen Sie, was passiert

Laden Sie die Codedatei (`1.8_123_wooden_man.sb3`) in Scratch 3.

Wenn die grüne LED leuchtet, können Sie **Avery** mit der rechten Pfeiltaste so steuern, dass sie nach rechts geht; Wenn die rote LED leuchtet und Sie **Avery** weiter nach rechts bewegen lassen, ertönt ein Alarm.

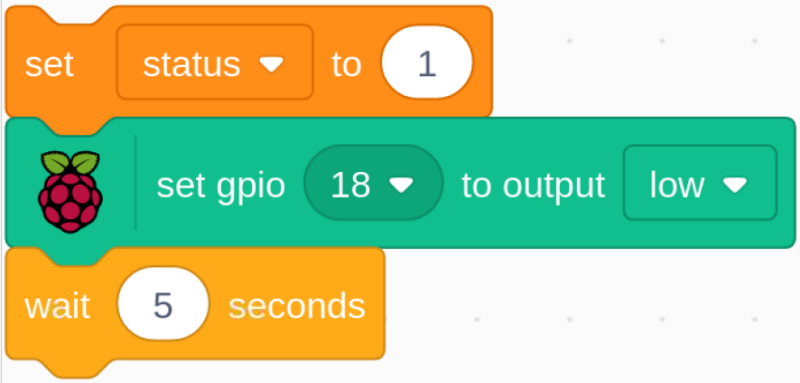
Tipps zu Sprite

Löschen Sie das Standard-Sprite und wählen Sie dann das **Avery Walking** -Sprite.

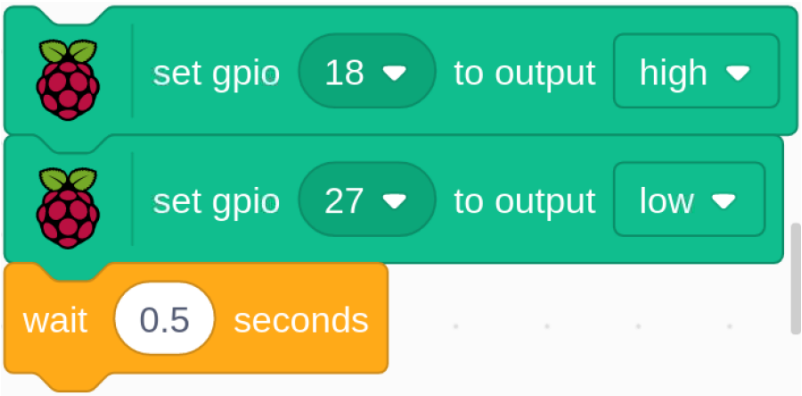


Tipps zu Codes

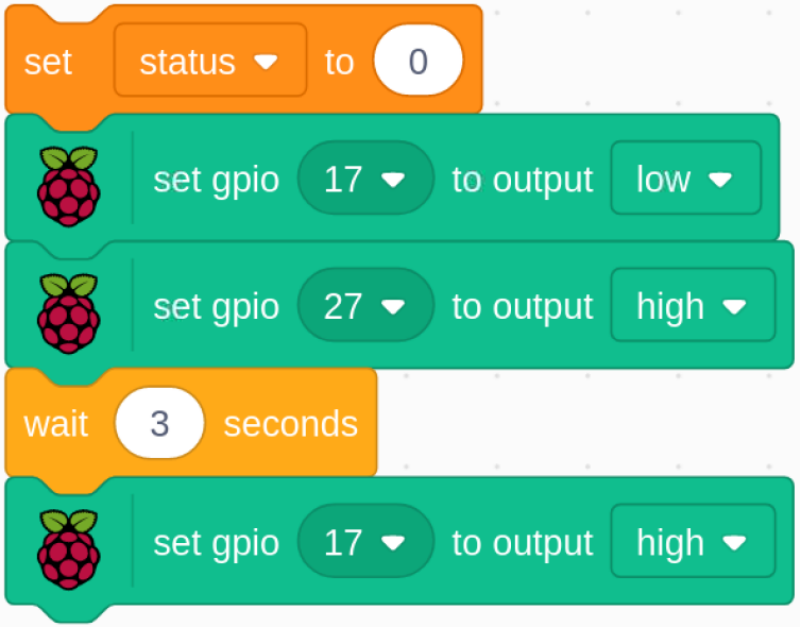
Initialisieren Sie alle Pins auf High.



Wenn das Spiel startet, weisen Sie die Statusvariable auf 1 zu, um anzuzeigen, dass das Avery Walking-Sprite beweglich ist, und setzen Sie dann gpio18 auf niedrig, wodurch die grüne LED 5 Sekunden lang aufleuchtet.



Setzen Sie gpio18 auf hoch und dann gpio27 auf niedrig, d.h. schalten Sie die grüne LED aus und leuchten Sie die gelbe LED für 0,5s auf.



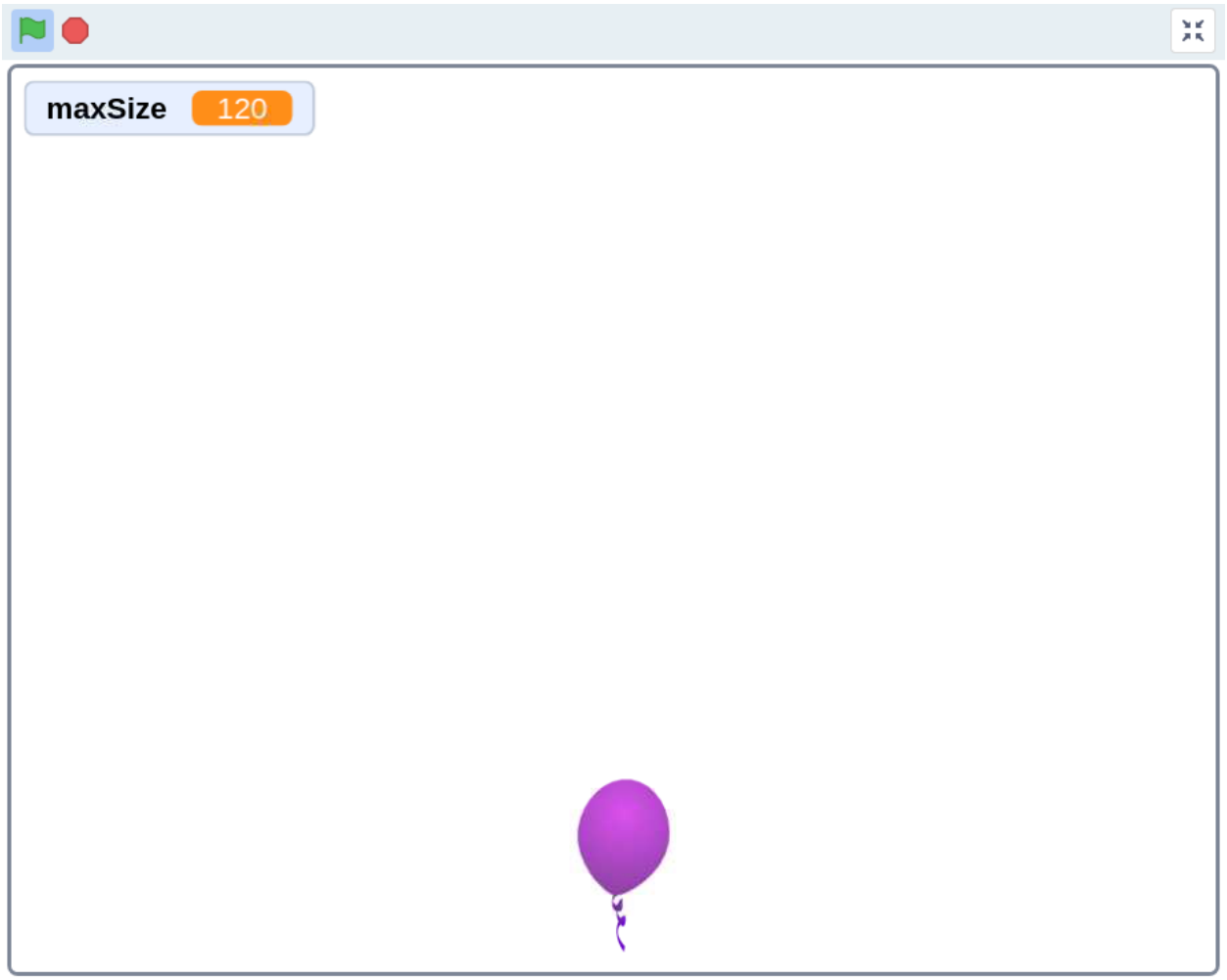
Weisen Sie die Statusvariable auf 0 zu, was bedeutet, dass sich das Avery Walking-Sprite nicht bewegt; setze dann gpio27 auf low und gpio17 auf high, was die gelbe LED ausschaltet und dann die rote LED für 3s aufleuchtet. Setzen Sie schließlich gpio17 auf hoch, um die rote LED auszuschalten.

Wenn wir die rechte Pfeiltaste auf der Tastatur drücken, müssen wir das Sprite Avery Walking auf das nächste Kostüm umschalten, damit wir Avery nach rechts gehen sehen. Dann müssen wir den Wert der Variable status bestimmen. Wenn es 0 ist, bedeutet dies, dass sich das Avery Walking-Sprite in diesem Moment nicht bewegt, und der Summer ertönt, um Sie zu warnen, dass Sie die rechte Pfeiltaste nicht erneut drücken können.

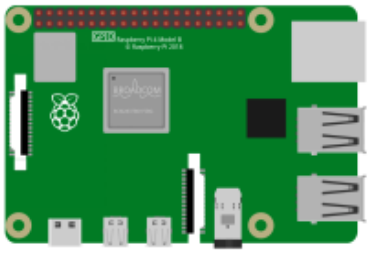
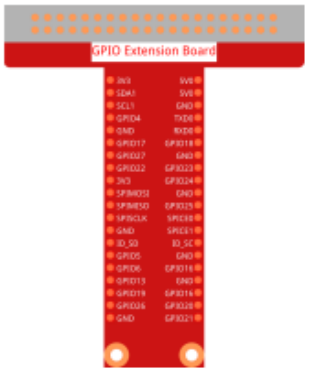
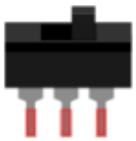
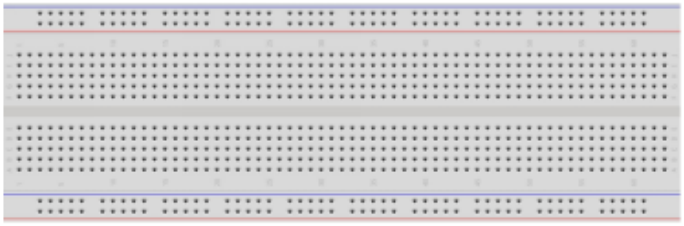
1.9 Aufblasen des Ballons

Hier spielen wir eine Ballonfahrt.

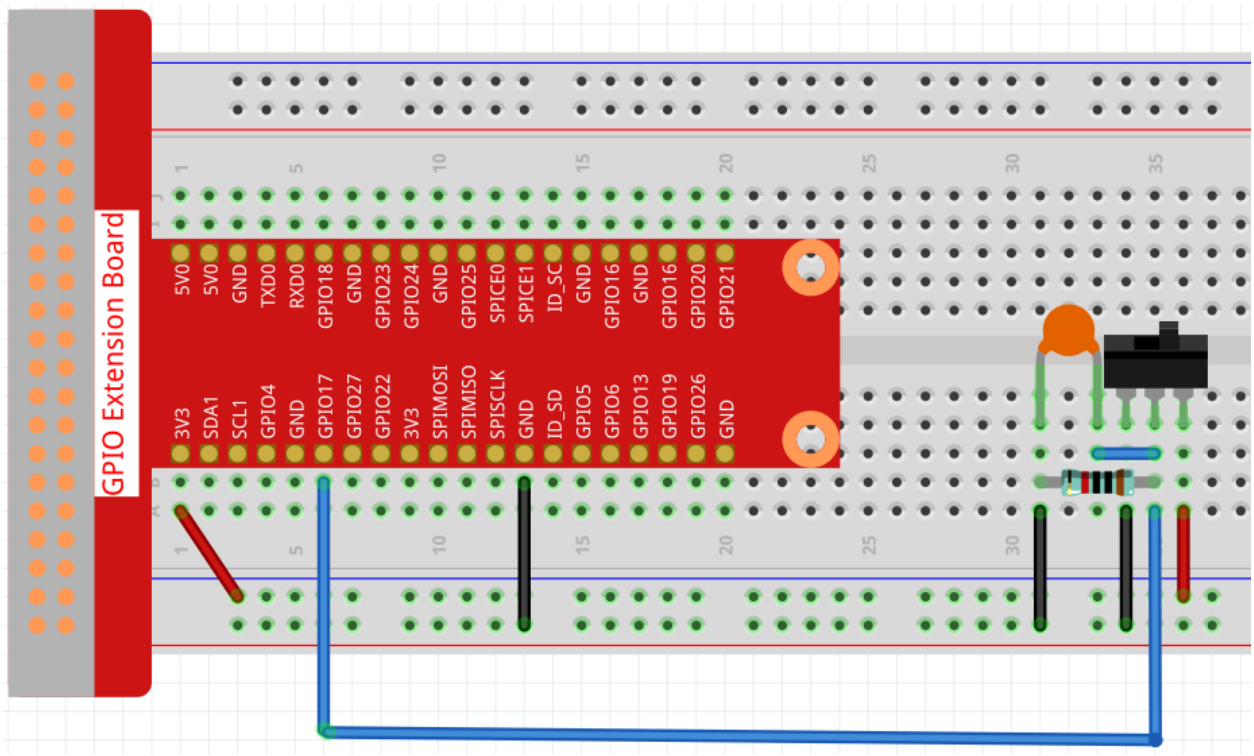
Durch Umschalten von Slide nach links, um den Ballon aufzublasen, wird der Ballon zu diesem Zeitpunkt immer größer. Wenn der Ballon zu groß ist, bläst er auf; Wenn der Ballon zu klein ist, schwebt er nicht in der Luft. Sie müssen beurteilen, wann Sie den Schalter nach rechts schieben müssen, um das Pumpen zu stoppen.


maxSize
120

Erforderliche Komponenten

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)

Baue die Schaltung



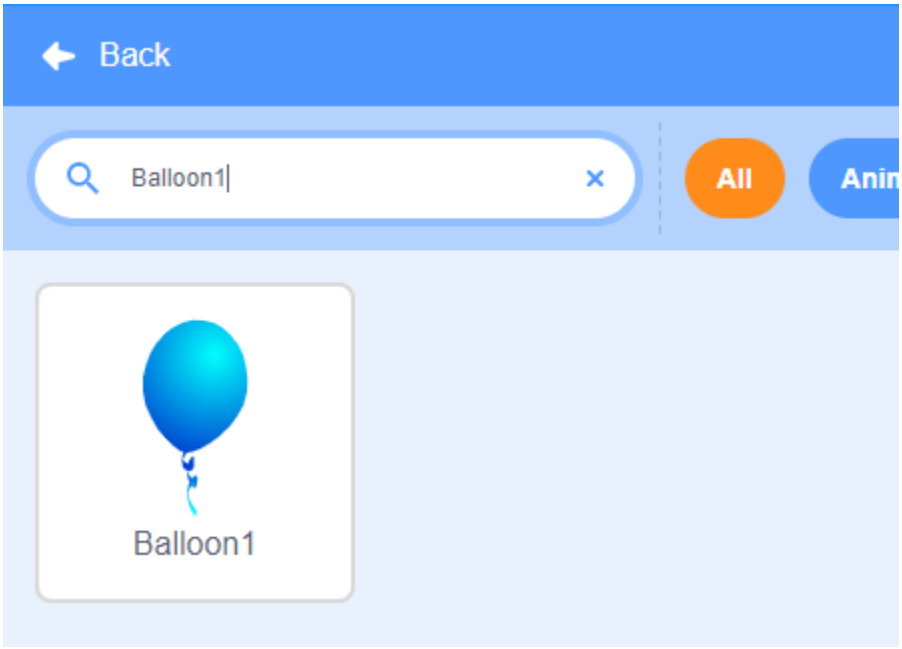
Laden Sie den Code und sehen Sie, was passiert

Laden Sie die Codedatei (`1.9_inflating_the_balloon.sb3`) in Scratch 3.

Durch Umschalten des Schiebereglers nach links, um mit dem Aufblasen des Ballons zu beginnen, wird der Ballon zu diesem Zeitpunkt immer größer. Wenn der Ballon zu groß ist, bläst er auf; Wenn der Ballon zu klein ist, schwebt er nicht in der Luft. Sie müssen beurteilen, wann Sie den Schalter nach rechts schieben müssen, um das Pumpen zu stoppen.

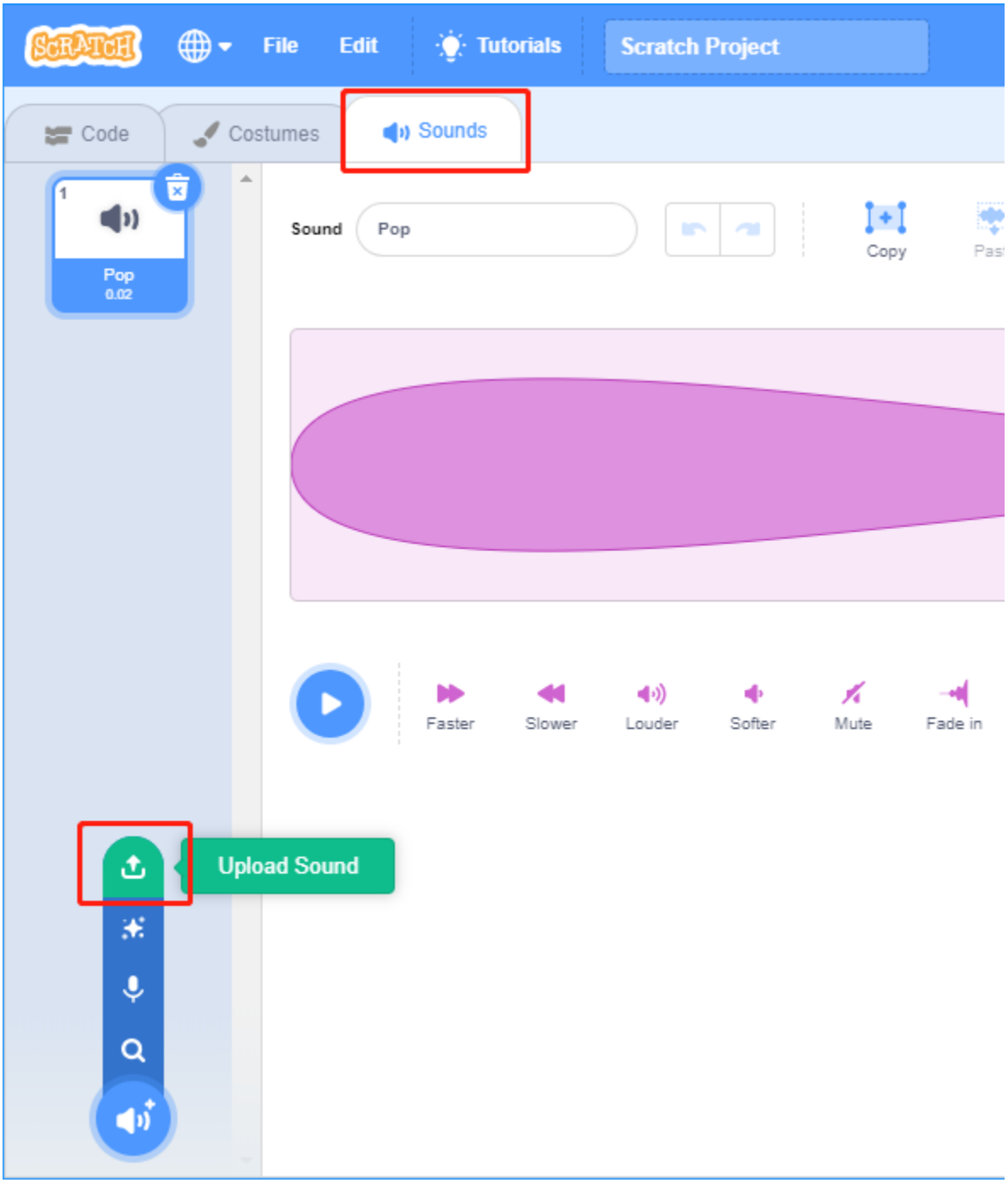
Tipps zu Figur

Löschen Sie das vorherige Sprite1-Sprite und fügen Sie dann das **Balloon1** -Sprite hinzu.

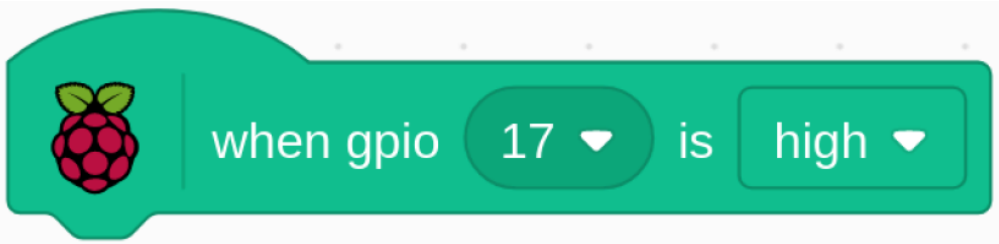


In diesem Projekt wird ein Ballonexplosions-Soundeffekt verwendet. Sehen wir uns also an, wie er hinzugefügt wurde.

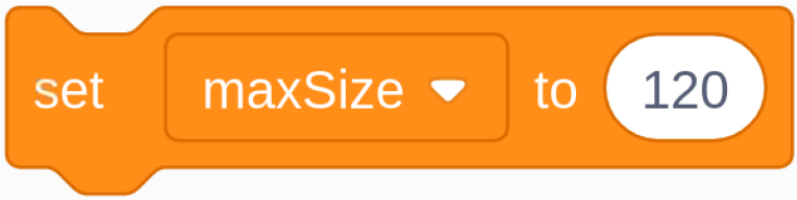
Klicken Sie oben auf die Option **Sound** und dann auf **Upload Sound** um `boom.wav` von `home/pi/davinci-kit-for-raspberry-pi/scratch/sound` hochzuladen. Pfad zu Scratch 3.


File
Edit
Tutorials
Scratch Project
Code
Costumes
Sounds
Pop
0.02
Sound
Pop
Copy
Faster
Slower
Louder
Softer
Mute
Fade in
Upload Sound

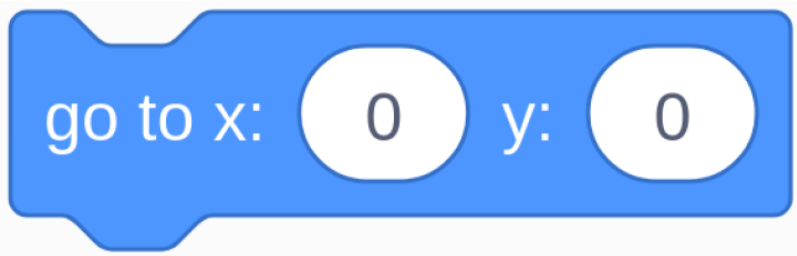
Tipps zu Codes



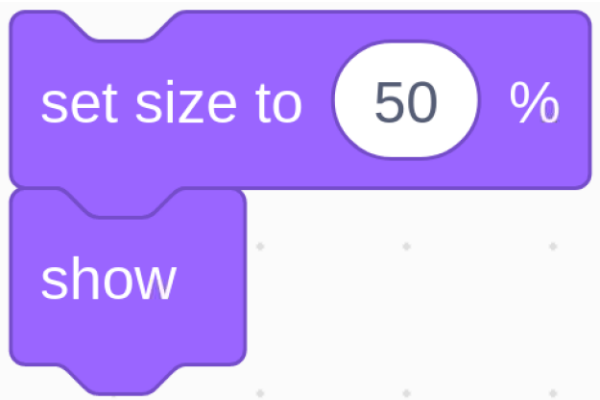
Dies ist ein Ereignisblock, und die Triggerbedingung ist, dass gpio17 hoch ist, dh der Schalter wird nach links geschaltet.



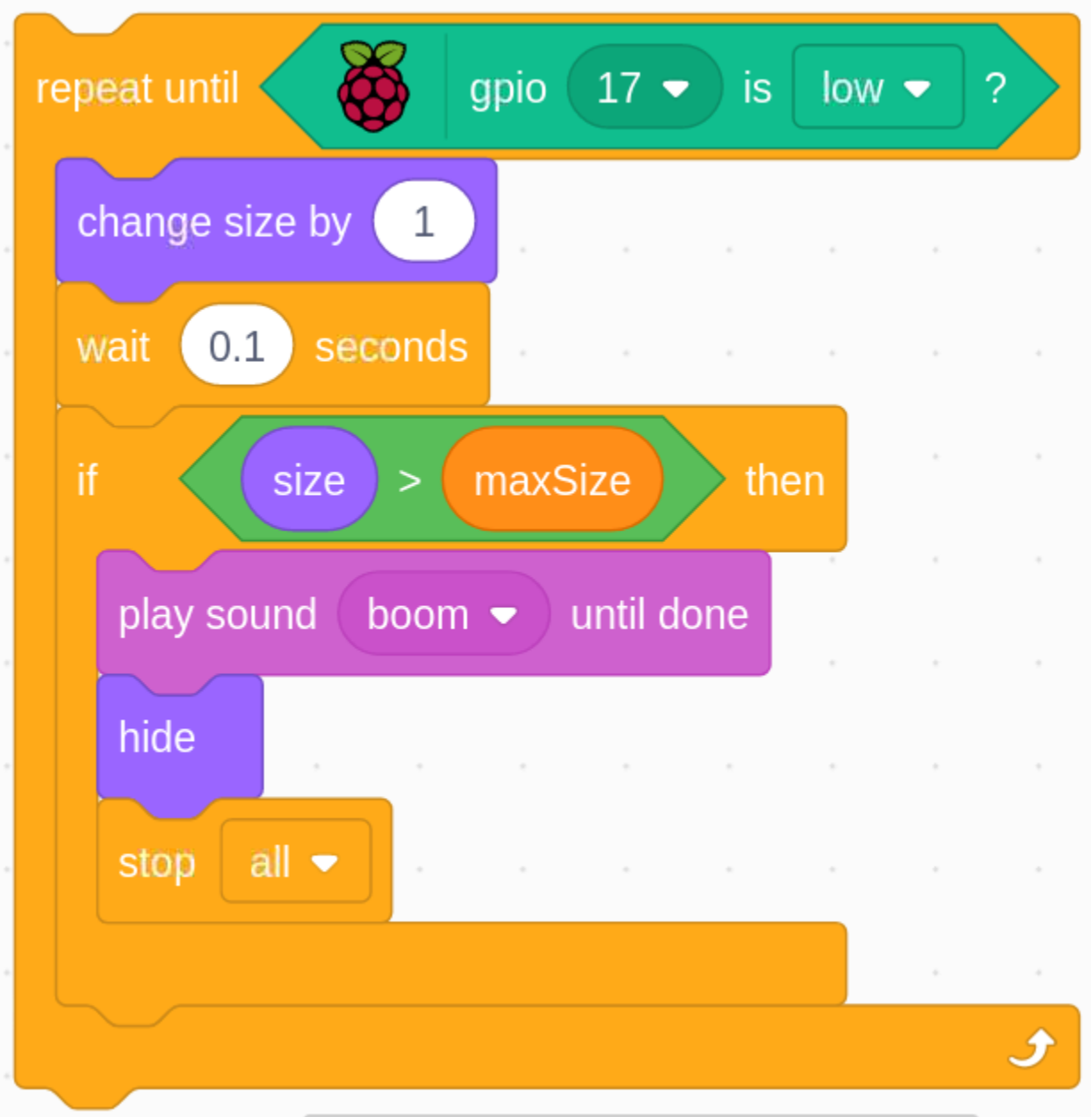
Stellen Sie die Größenschwelle des Ballon1-Sprites auf 120 .



Verschieben Sie die Koordinaten des Sprites Balloon1 auf (0,0), die Mitte des Bühnenbereichs.

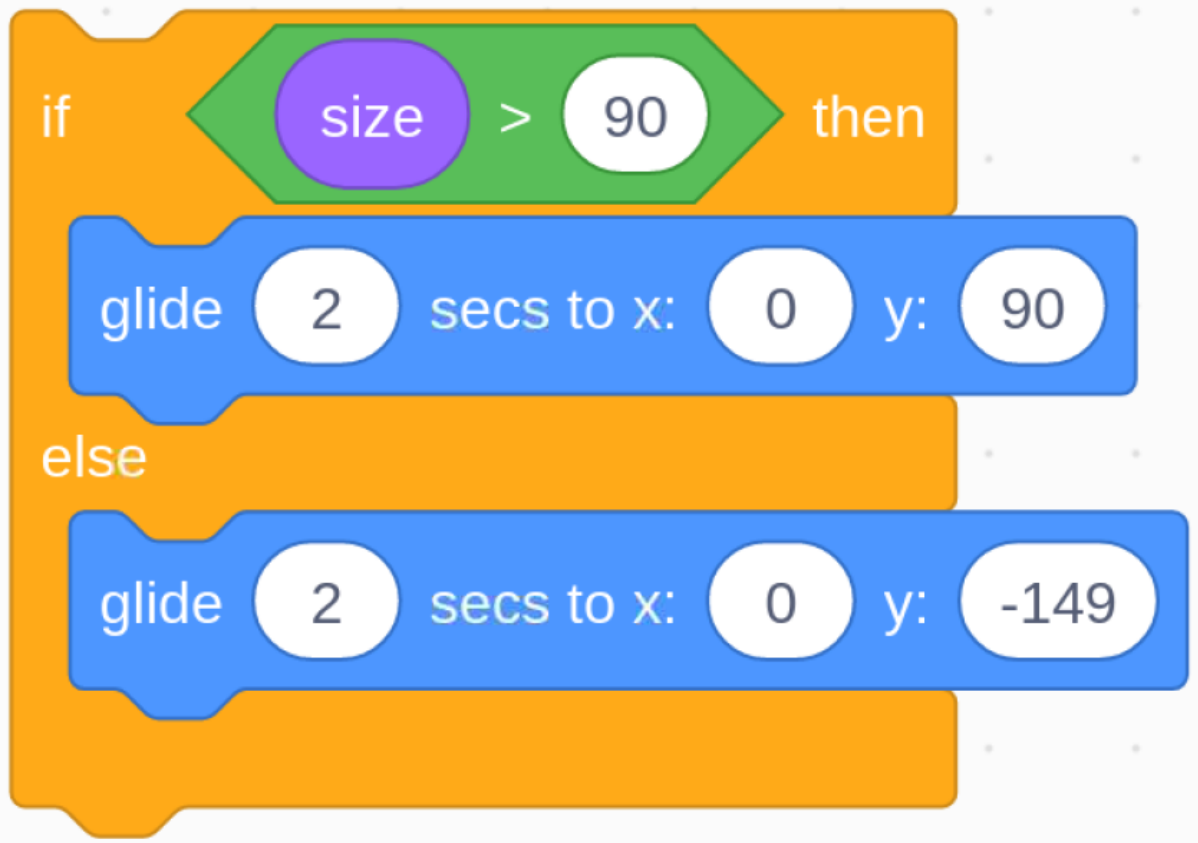


Stellen Sie die Größe des Balloon1-Sprites auf 50 ein und zeigen Sie es im Bühnenbereich an.



Richten Sie eine Schlaufe zum Aufblasen des Ballons ein. Diese Schlaufe stoppt, wenn der Schieberegler nach rechts geschoben wird.

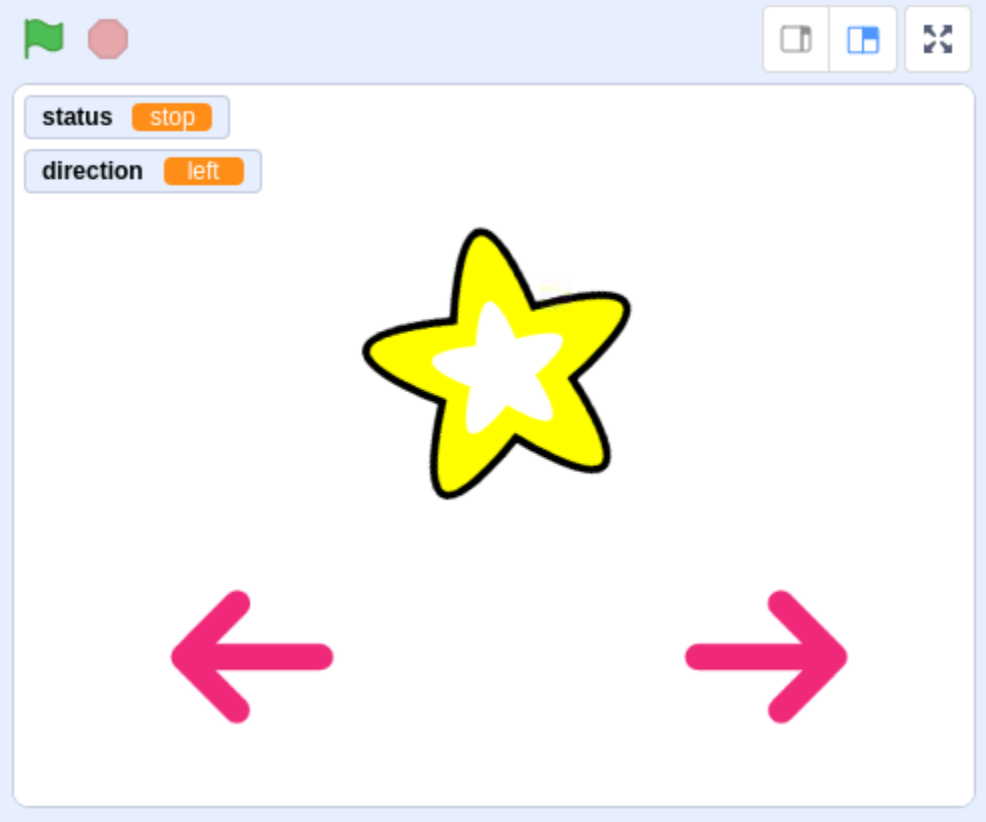
Innerhalb dieser Schleife wird die Ballongröße alle 0,1s um 1 erhöht, und wenn sie größer als `maxSize` ist, platzt die Sprechblase, woraufhin das Boom-Geräusch erzeugt und der Code verlassen wird.



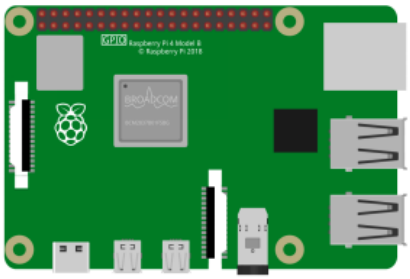
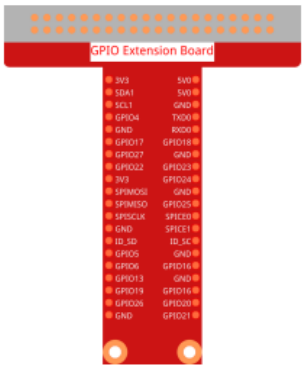
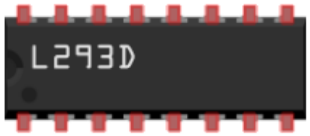
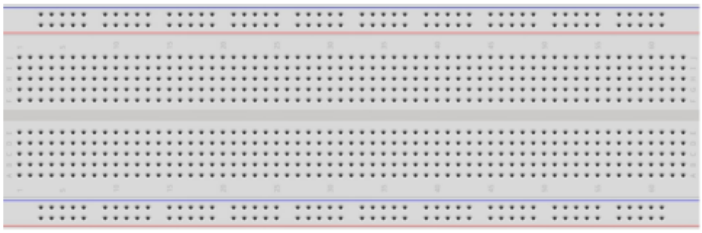
Nachdem die letzte Schleife beendet wurde (Slider schaltet nach rechts), bestimmen Sie die Position des Balloon1-Sprites basierend auf seiner Größe. Wenn die Größe des Balloon1-Sprites größer als 90 ist, heben Sie ab (bewegen Sie die Koordinaten auf (0, 90), andernfalls landen Sie (bewegen Sie die Koordinaten auf (0, -149).

1.10 Rotierender Lüfter

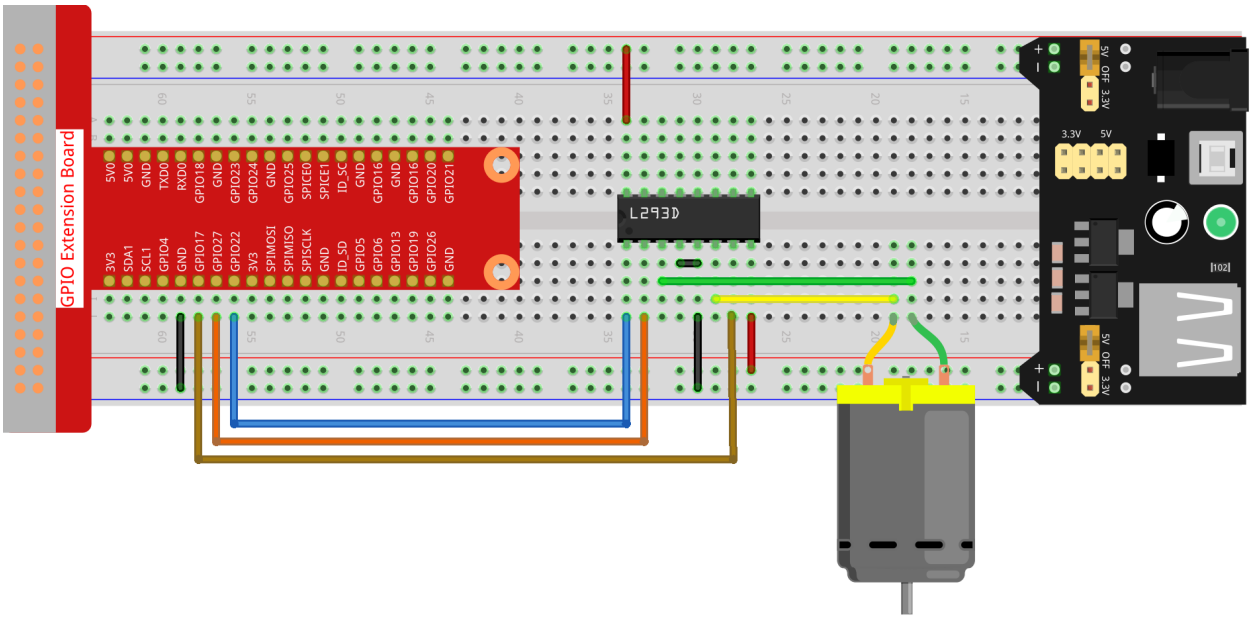
In diesem Projekt werden wir ein sich drehendes Stern-Sprite und einen Fächer herstellen.


status stop
direction left

Erforderliche Komponenten

<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" rowspan="2">1 * Breadboard
  </td>
<td>Several Jumper Wires
 </td>
</tr>
<tr>
<td>1 * DC Motor
 </td>
</tr>
</table>

Baue die Schaltung



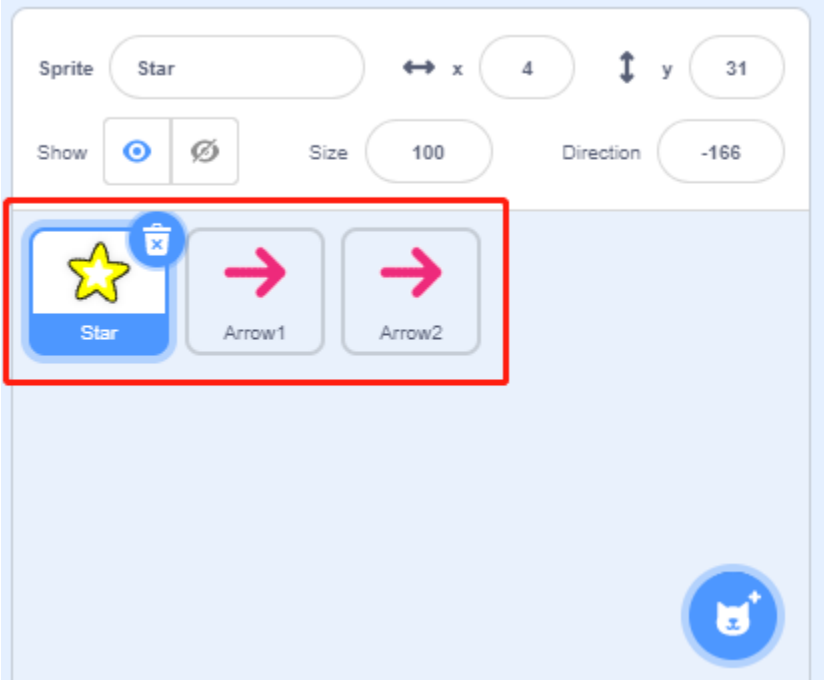
Laden Sie den Code und sehen Sie, was passiert

Laden Sie die Codedatei (`1.10_rotating_fan.sb3`) in Scratch 3.

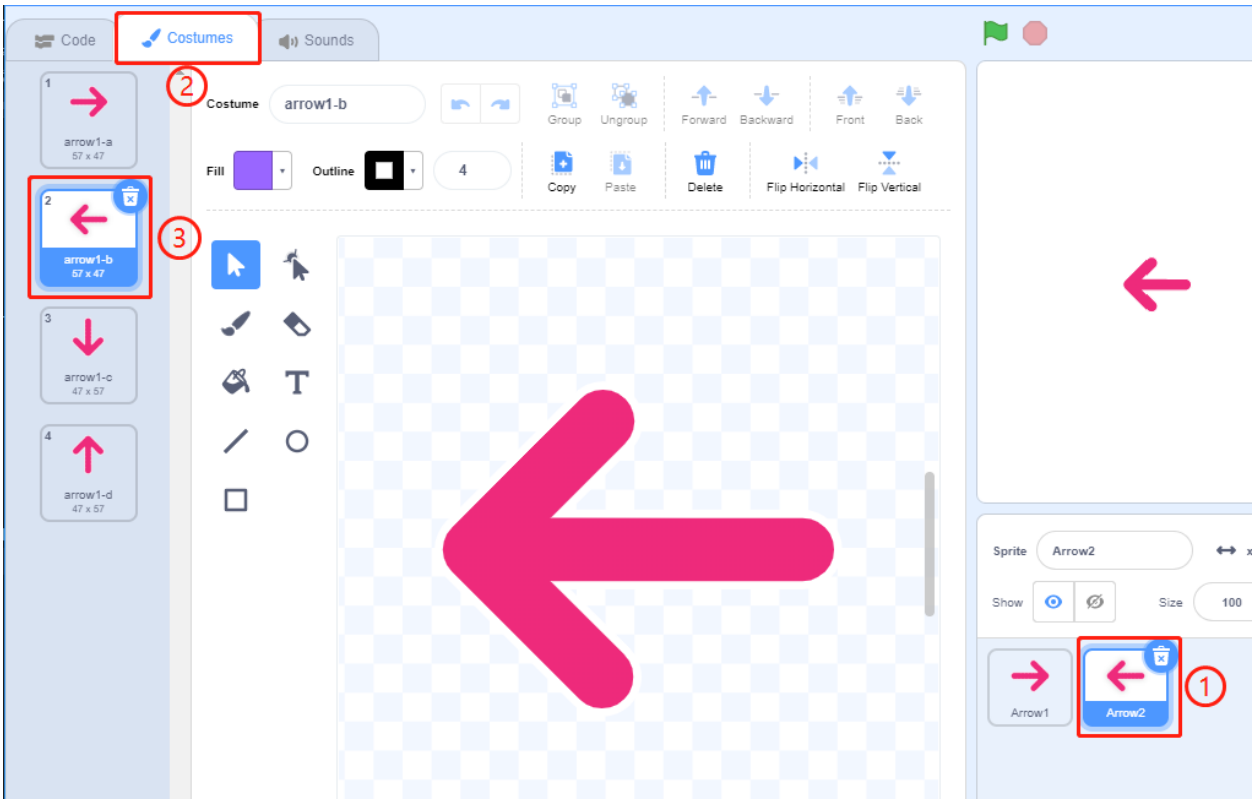
Nachdem Sie auf die grüne Flagge auf der Bühne geklickt haben, klicken Sie auf das Stern-Sprite, dann dreht sich der Motor im Uhrzeigersinn. Sie können die Drehrichtung ändern, indem Sie auf die beiden Pfeil-Sprites klicken. Wenn Sie erneut auf das Stern-Sprite klicken, hören es und der Motor auf zu drehen.

Tipps zu Figur

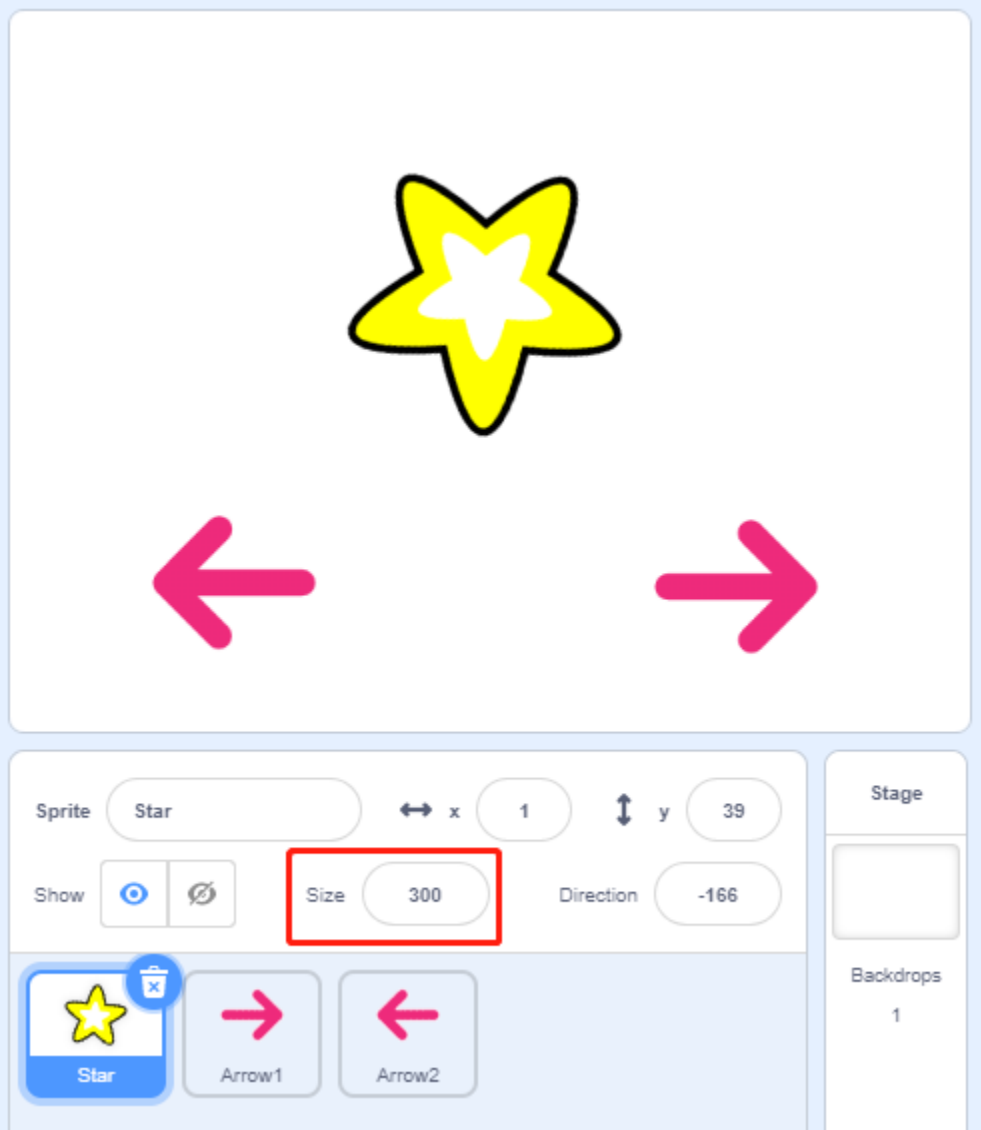
Löschen Sie das Standard-Sprite, wählen Sie dann das **Star** -Sprite und das **Arrow1** -Sprite aus und kopieren Sie Arrow1 einmal.



Ändern Sie in der Option **Costumes** das **Arrow2** -Sprite in ein anderes Richtungskostüm.

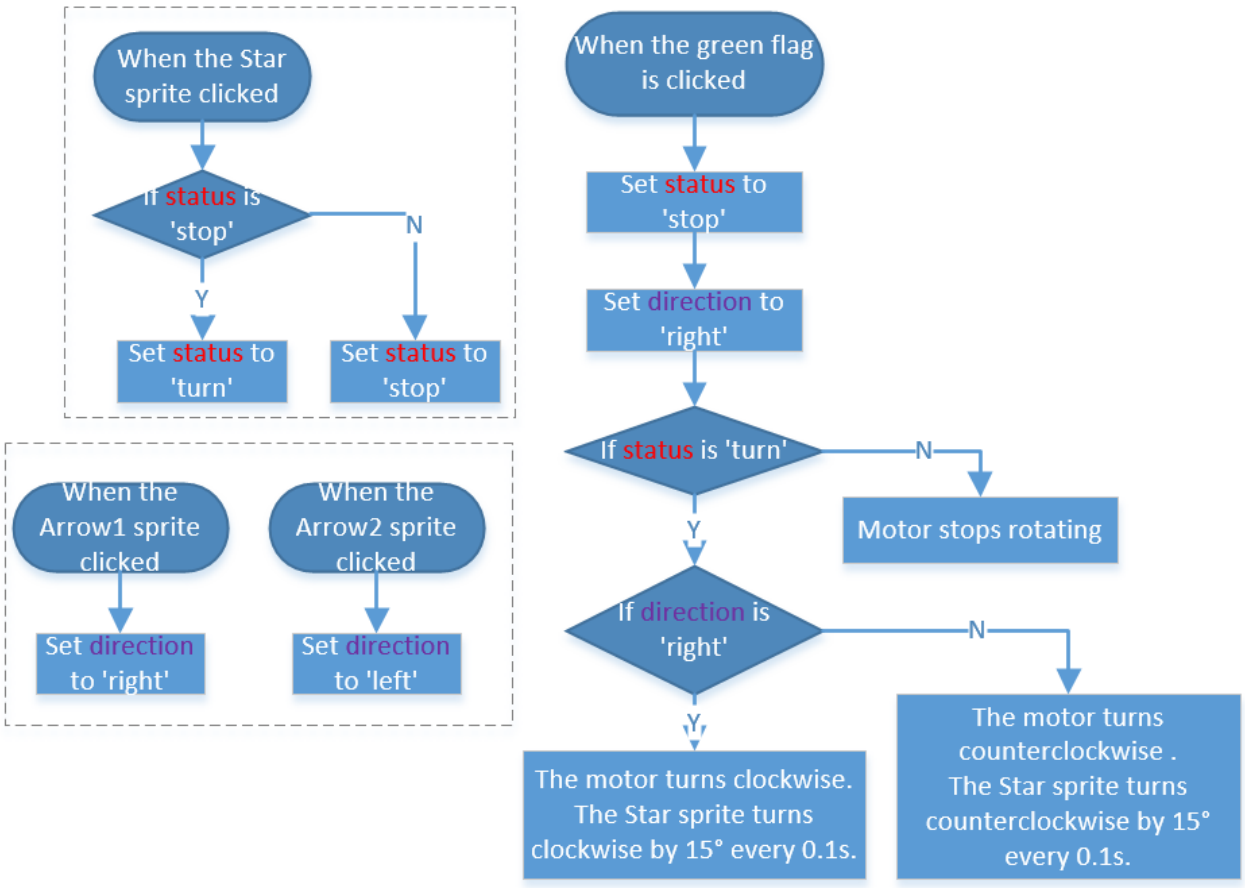


Passen Sie die Größe und Position des Sprites entsprechend an.



Tipps zu Codes

Flussdiagramm

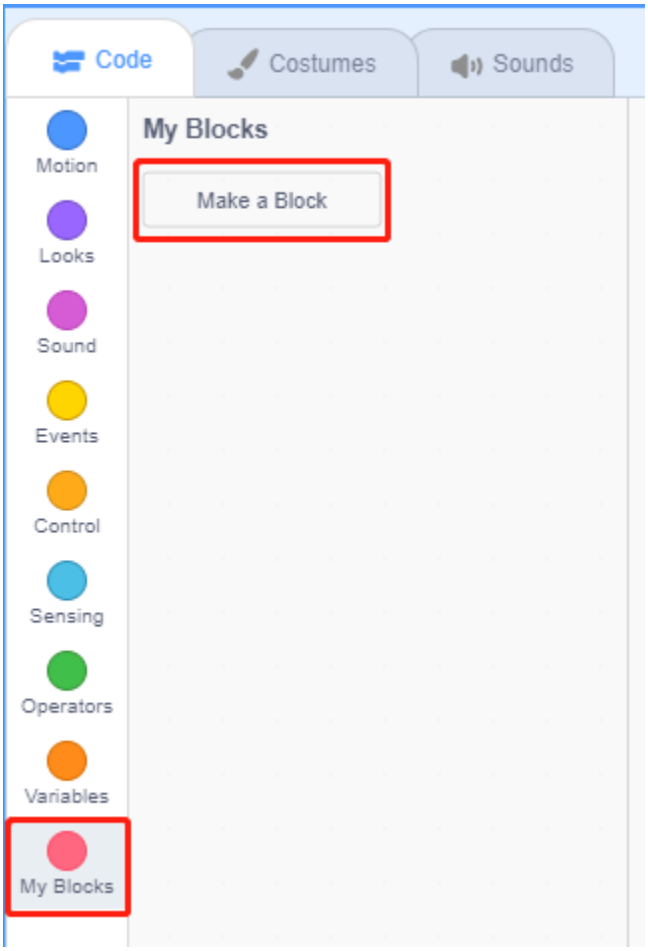


In diesem Code sehen Sie 2 rosa Blöcke, biegen Sie links ab und biegen Sie rechts ab, die unsere benutzerdefinierten Blöcke (Funktionen) sind.

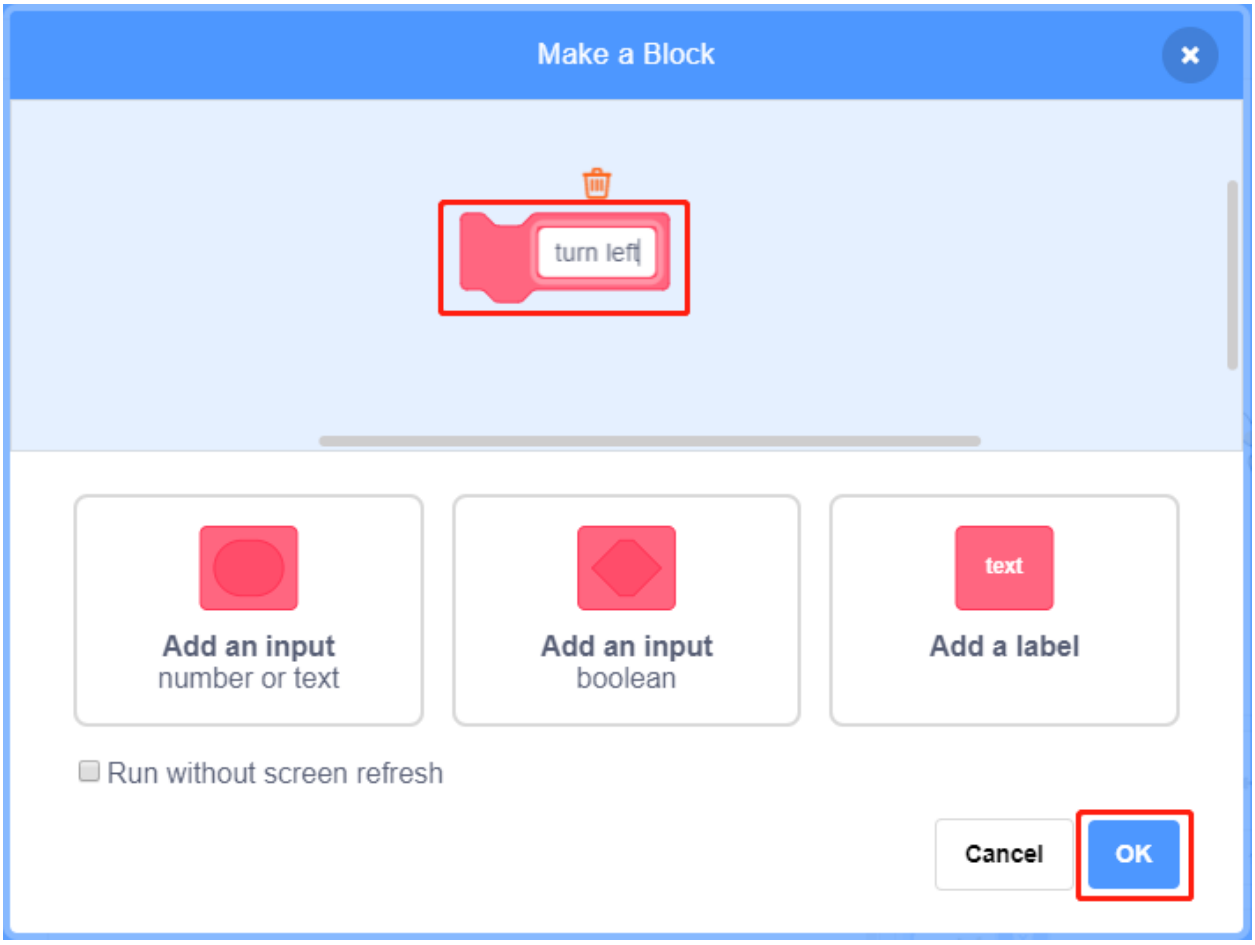
Wie erstelle ich einen Block?

Lassen Sie uns lernen, wie man einen Block (eine Funktion) erstellt. Der Block (Funktion) kann verwendet werden, um Ihr Programm zu vereinfachen, insbesondere wenn Sie dieselbe Operation mehrmals ausführen. Das Einfügen dieser Operationen in einen neu deklarierten Block kann für Sie sehr praktisch sein.

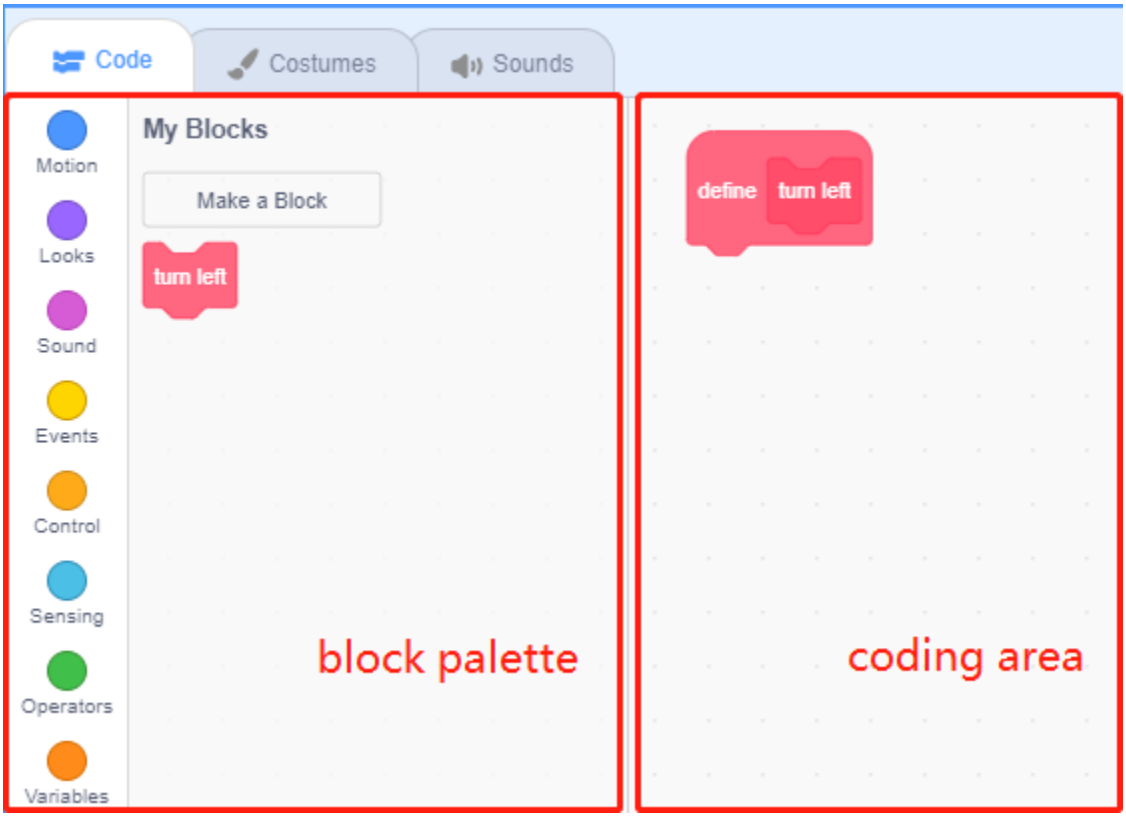
Suchen Sie zuerst **My Blocks** in der Blockpalette und wählen Sie dann **Make a Block**.



Geben Sie den Namen des neuen Blocks ein.

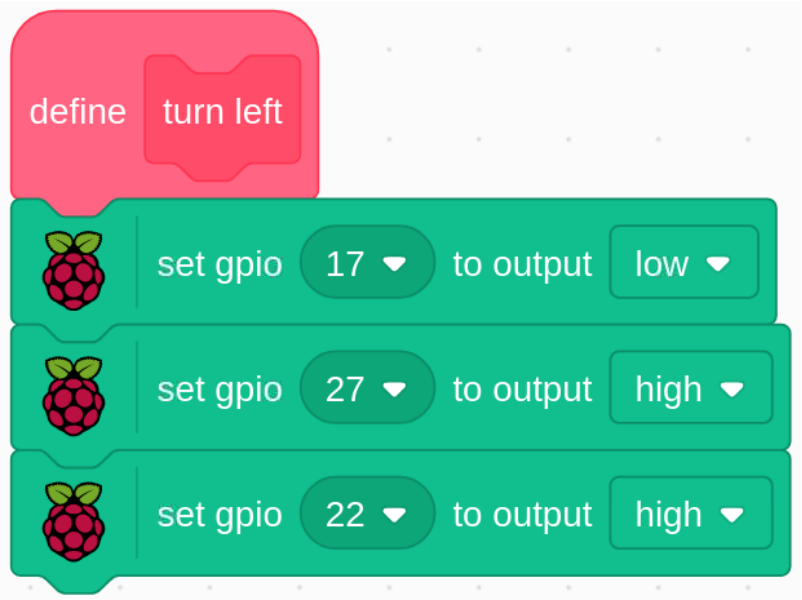


Nachdem Sie die Funktion des neuen Blocks in den Codierbereich geschrieben haben, speichern Sie diese und Sie finden den Block dann in der Blockpalette.



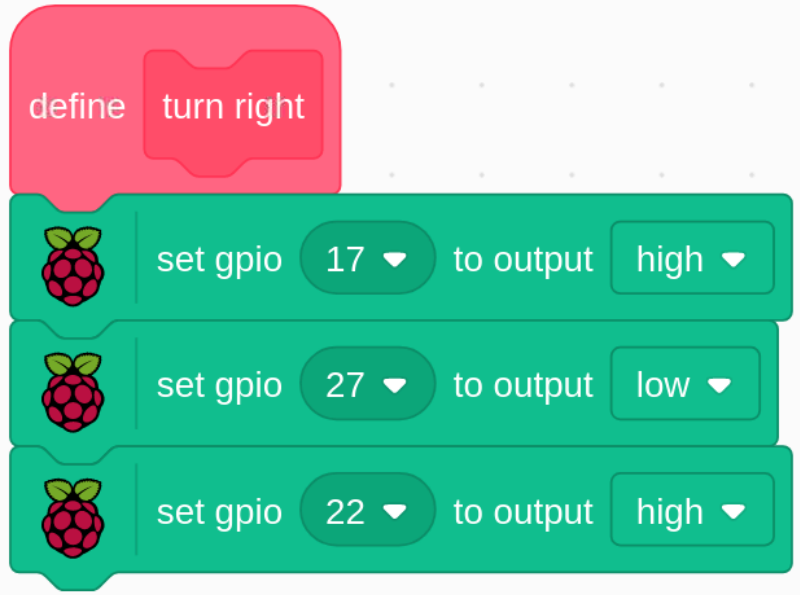
Biegen Sie links ab

Dies ist der Code innerhalb des Linksabbiegeblocks, um den Motor gegen den Uhrzeigersinn drehen zu lassen.



Biegen Sie rechts ab

Dies ist der Code im Block rechts drehen, um den Motor im Uhrzeigersinn drehen zu lassen.



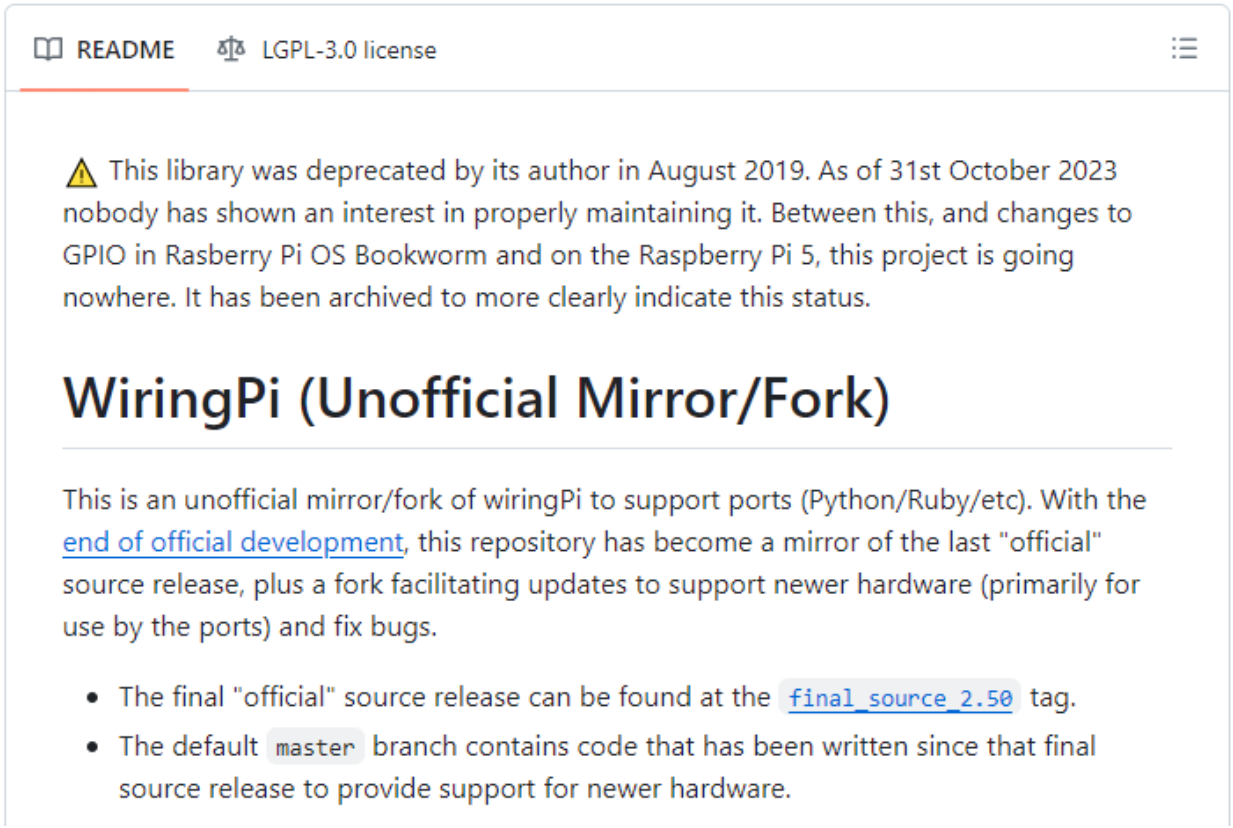
1.14 Andere Sprache (für Pi 5)

Die Veröffentlichung des Raspberry Pi 5 hat uns ein leistungsfähigeres Modell gebracht, aber sie hat auch einige Änderungen eingeführt, insbesondere bei den GPIOs.

Obwohl es seine standardmäßige 40-Pin-Schnittstelle beibehält, hat sich die Funktionalität aufgrund seiner Verbindung mit dem neu integrierten RP1-Southbridge-Chip verschoben. Dieser benutzerdefinierte RP1-Chip kümmert sich nun um die Peripheriegeräte auf dem Pi 5 und hat verschiedene Kompatibilitätsbedenken zur Folge.

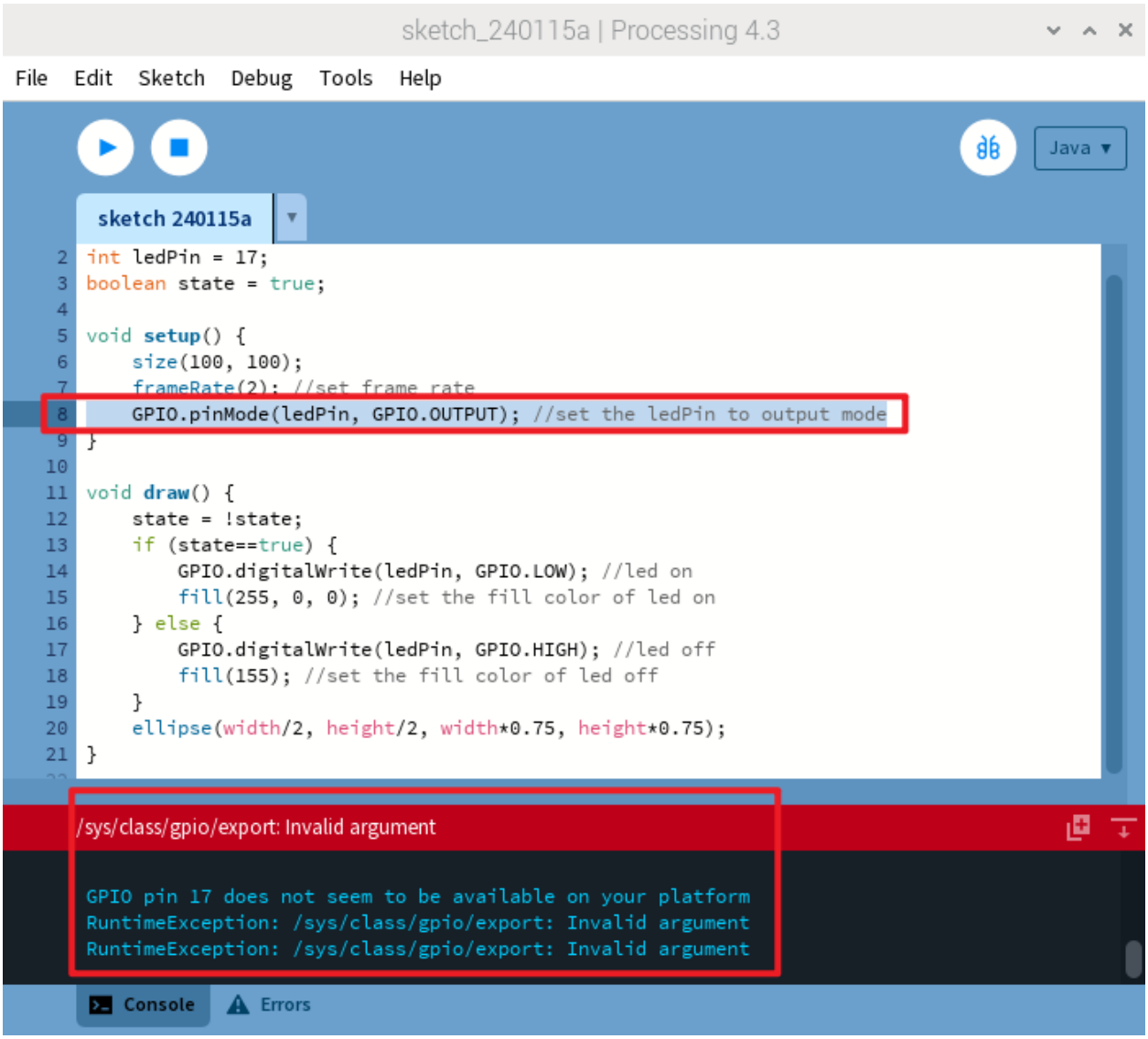
1.14.1 C-Sprache

Die Implementierung in C-Sprache stützt sich auf die wiringPi-Bibliothek. Jedoch ist die wiringPi-Community-Bibliothek nun archiviert und erhält keine Updates mehr, was sie für Projekte mit dem Raspberry Pi 5 ungeeignet macht. Für weitere Informationen siehe: https://github.com/WiringPi/WiringPi



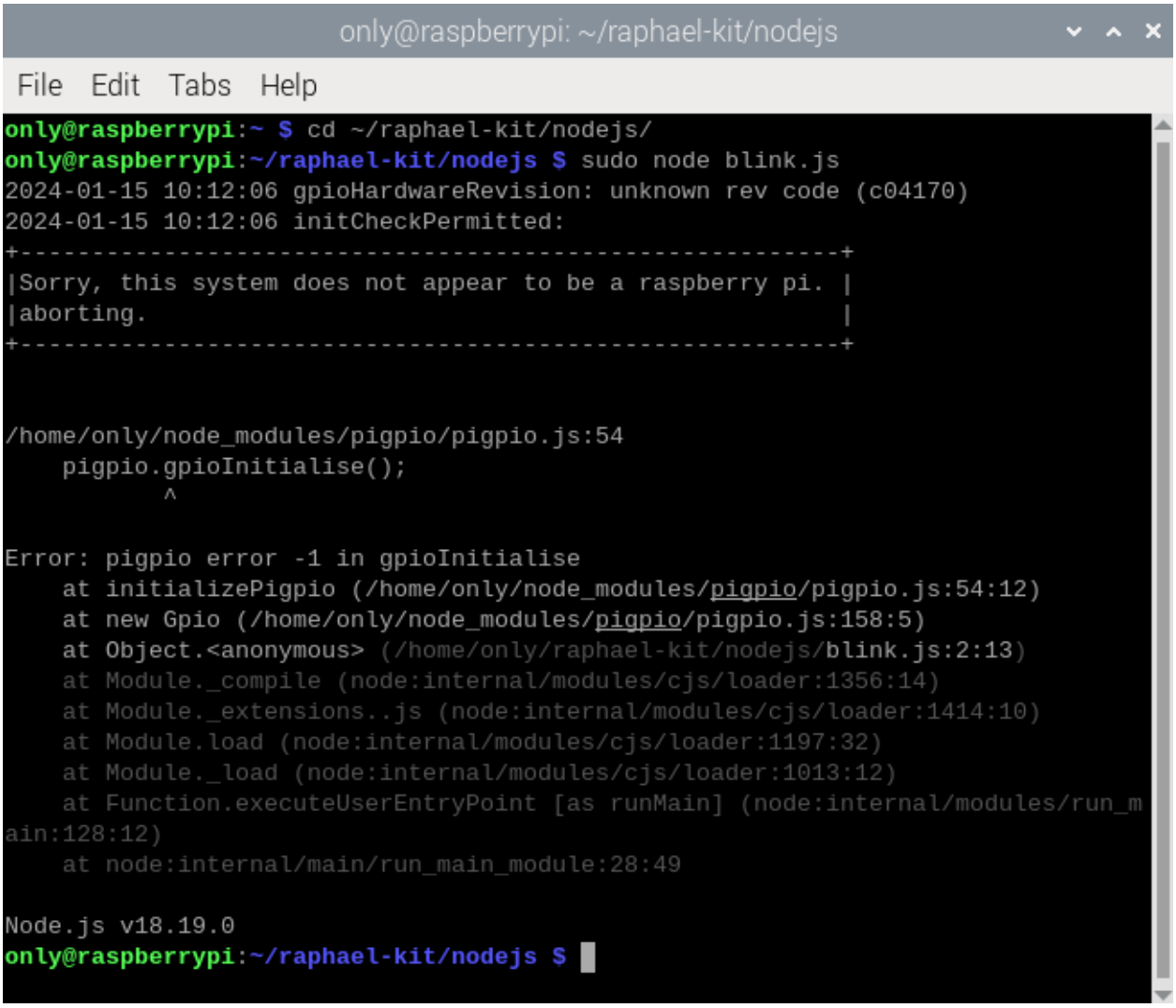
1.14.2 Processing

Beim Einsatz von Processing 4 auf dem Raspberry Pi 5 stoßen GPIO-Programmierungen auf Herausforderungen. Fehler wie „Invalid argument“ und „GPIO pin 17 seems to be unavailable on your platform“ treten während der Ausführung von GPIO-bezogenem Code auf (wie im beigefügten Bild dargestellt). Für weitere Details besuchen Sie: https://github.com/benfry/processing4/issues/807



1.14.3 Node.js

Node.js nutzt die pigpio-Bibliothek, die bislang den Raspberry Pi 5 nicht unterstützt. Für mehr Einblicke besuchen Sie: https://github.com/joan2937/pigpio/issues/589



1.14.4 Scratch

Auf einem 64-Bit-System stößt der Import der Raspberry Pi GPIO-Bibliothek auf Probleme, was zu Unresponsiveness führt. Für weitere Informationen besuchen Sie: https://github.com/raspberrypi/bookworm-feedback/issues/91

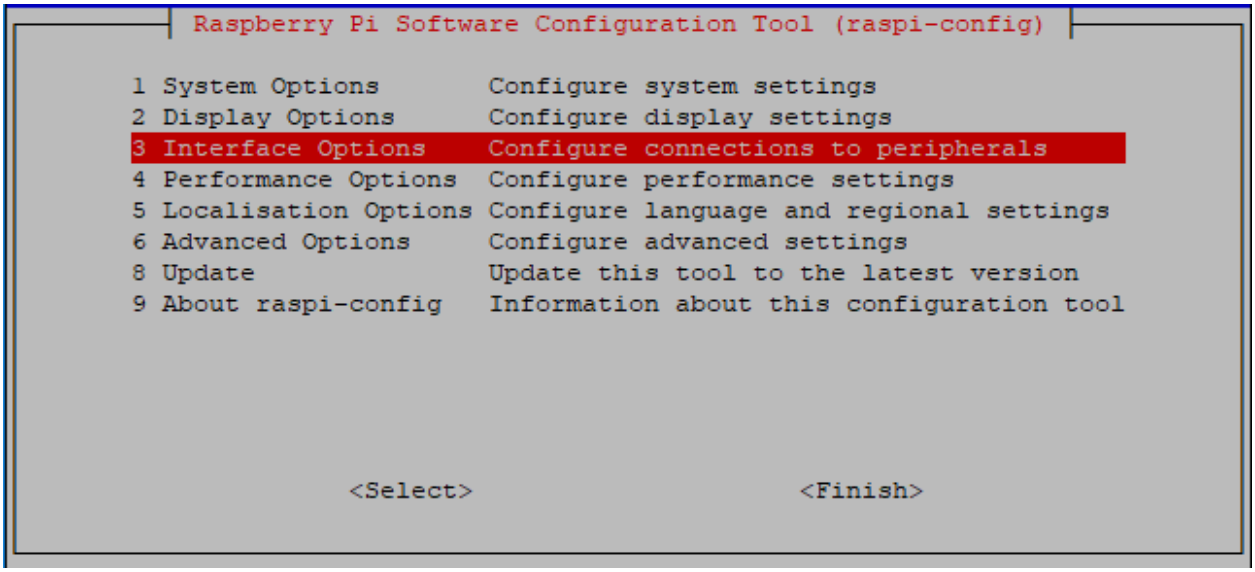
1.15 Anhang

1.15.1 I2C-Konfiguration

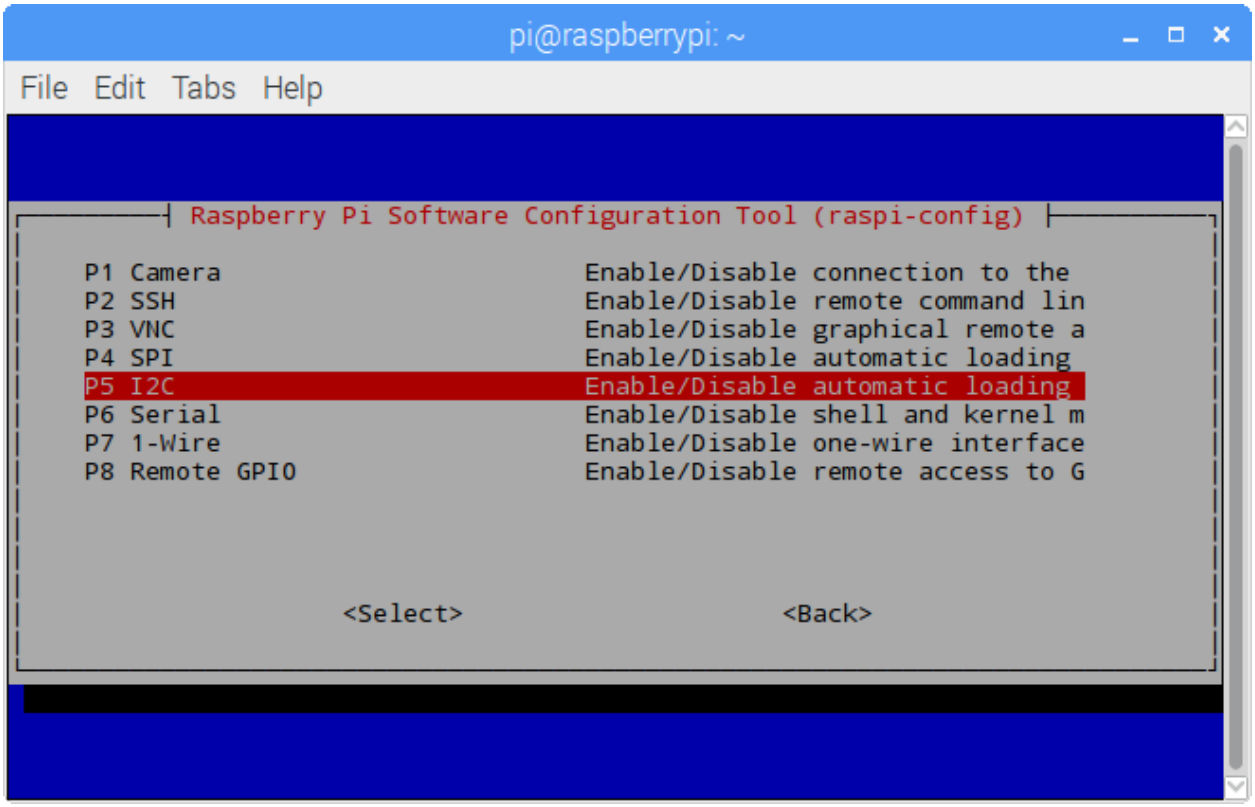
Schritt 1: Aktivieren Sie den I2C-Port Ihres Raspberry Pi (Wenn Sie ihn aktiviert haben, überspringen Sie diesen; wenn Sie nicht wissen, ob Sie das getan haben oder nicht, fahren Sie bitte fort).

```
sudo raspi-config
```

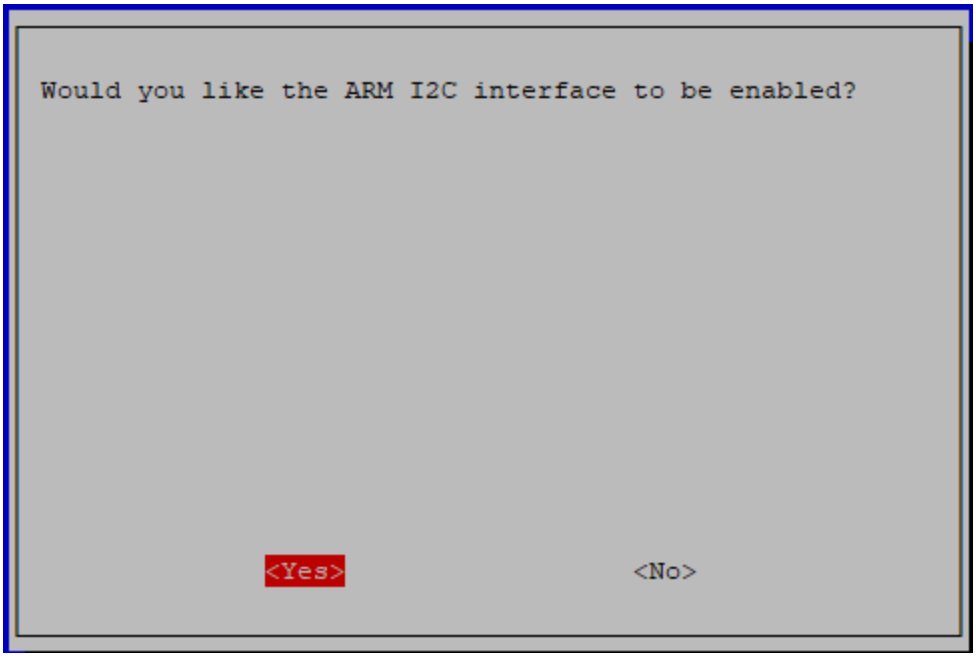
3 Schnittstellenoptionen



P5 I2C



<Yes>, dann <Ok> -> <Finish>



Schritt 2: Prüfen Sie, ob die i2c-Module geladen und aktiv sind.

```
lsmod | grep i2c
```

Dann erscheinen die folgenden Codes (die Nummer kann unterschiedlich sein).

```
i2c_dev                 6276    0
i2c_bcm2708             4121    0
```

Schritt 3: Installieren Sie i2c-tools.

```
sudo apt-get install i2c-tools
```

Schritt 4: Überprüfen Sie die Adresse des I2C-Geräts.

```
i2cdetect -y 1      # For Raspberry Pi 2 and higher version
```

```
i2cdetect -y 0      # For Raspberry Pi 1
```

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Wenn ein I2C-Gerät angeschlossen ist, wird die Adresse des Geräts angezeigt.

Schritt 5:

Für Benutzer der Sprache C: Installieren Sie libi2c-dev.

```
sudo apt-get install libi2c-dev
```

Für Python-Benutzer:

1. Aktivierung der virtuellen Umgebung.

Bemerkung:

- Bevor Sie die Aktivierung durchführen, müssen Sie sicherstellen, dass Sie eine virtuelle Umgebung erstellt haben. Bitte beachten Sie: *Erstellen einer virtuellen Umgebung*.
- Jedes Mal, wenn Sie den Raspberry Pi neu starten oder ein neues Terminal öffnen, müssen Sie erneut den folgenden Befehl ausführen, um die virtuelle Umgebung zu aktivieren.

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

Sobald die virtuelle Umgebung aktiviert ist, sehen Sie den Umgebungsnamen vor dem Befehlszeilen-Prompt, was darauf hinweist, dass Sie innerhalb der virtuellen Umgebung arbeiten.

2. Installation von smbus für I2C.

```
sudo pip3 install smbus2
```

3. Beenden der virtuellen Umgebung.

Wenn Sie Ihre Arbeit abgeschlossen haben und die virtuelle Umgebung verlassen möchten, führen Sie einfach folgenden Befehl aus:

```
deactivate
```

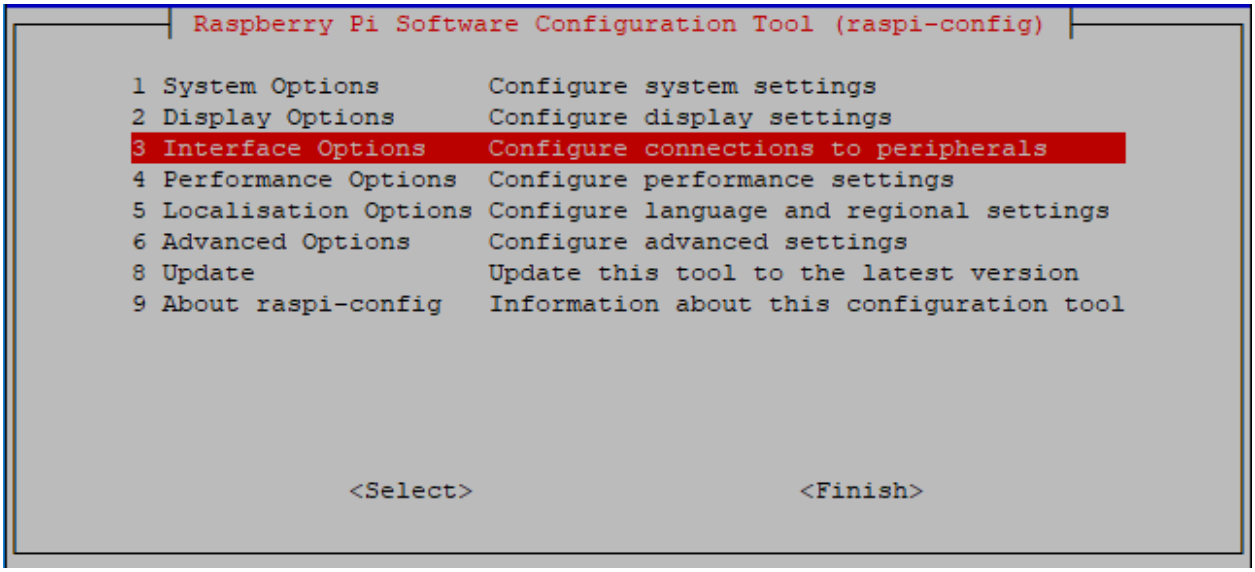
Dadurch kehren Sie zur globalen Python-Umgebung des Systems zurück.

1.15.2 SPI-Konfiguration

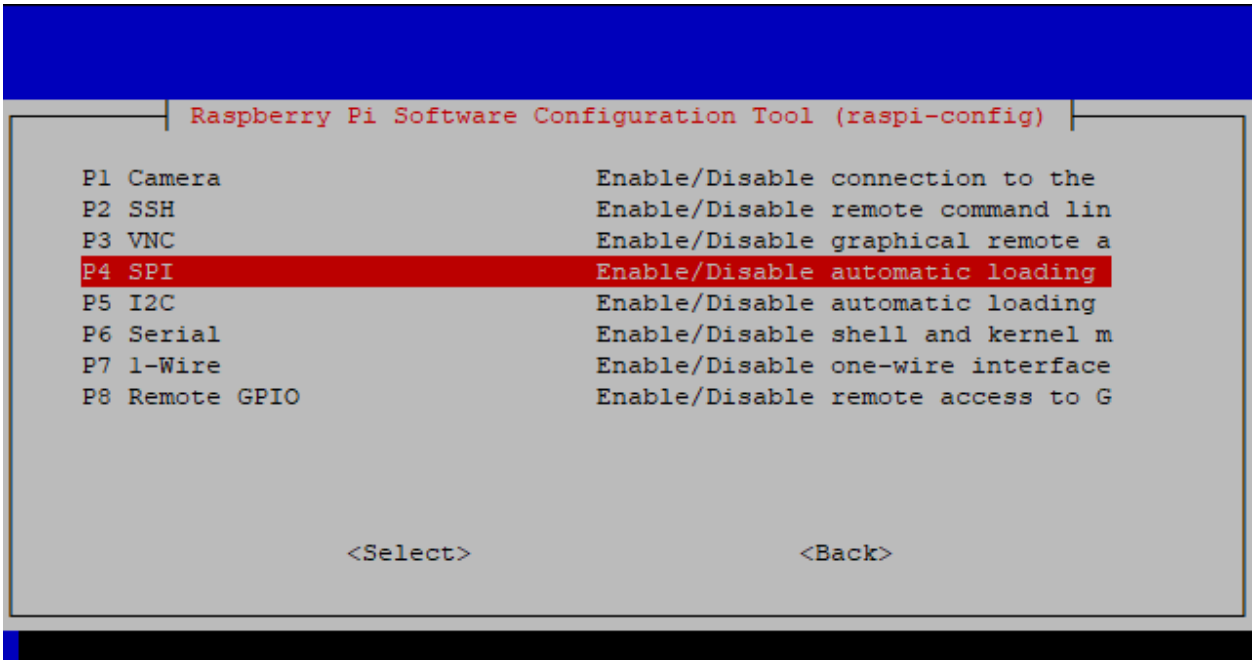
Schritt 1: Aktivieren Sie den SPI-Port Ihres Raspberry Pi (Wenn Sie ihn aktiviert haben, überspringen Sie diesen; wenn Sie nicht wissen, ob Sie das getan haben oder nicht, fahren Sie bitte fort).

```
sudo raspi-config
```

3 Schnittstellenoptionen



P4 SPI



<YES>, dann auf **<OK>** und **<Finish>** klicken. Jetzt können Sie den Raspberry Pi mit dem Befehl `sudo reboot` neu starten.

Schritt 2: Überprüfen Sie, ob die Spi-Module geladen und aktiv sind.

```
ls /dev/sp*
```

Dann erscheinen die folgenden Codes (die Nummer kann abweichen).

```
/dev/spidev0.0  /dev/spidev0.1
```

Schritt 3: Installieren Sie das Python-Modul SPI-Py.

```
git clone https://github.com/lthiery/SPI-Py.git
cd SPI-Py
sudo python3 setup.py install
```

Bemerkung: Dieser Schritt ist für Python-Benutzer, wenn Sie die Sprache C verwenden, überspringen Sie bitte.

1.15.3 Remotedesktop

Es gibt zwei Möglichkeiten, den Desktop des Raspberry Pi aus der Ferne zu steuern:

VNC und **XRDP** , Sie können jeden von ihnen verwenden.

VNC

Sie können die Funktion des Remote-Desktops über VNC verwenden.

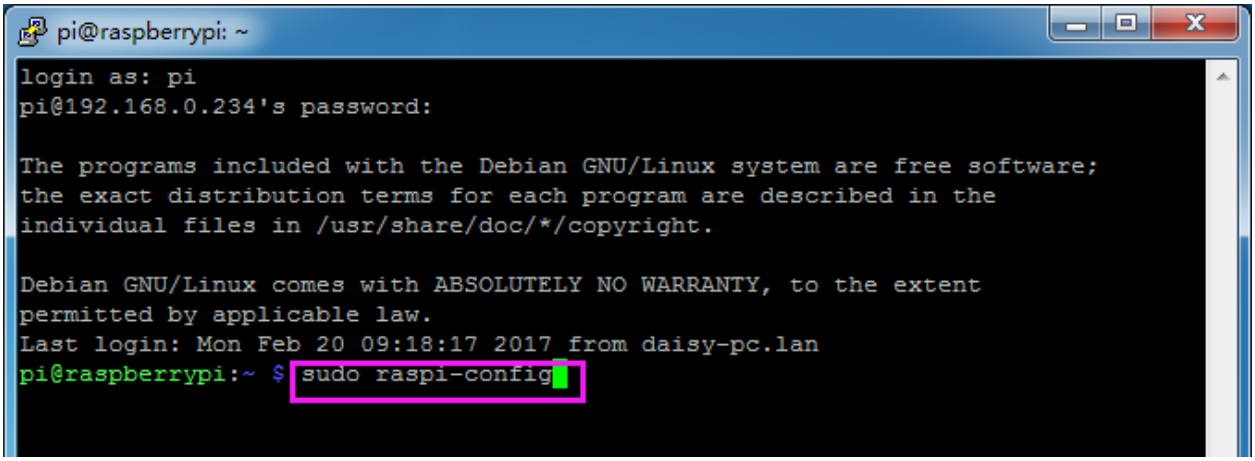
VNC-Dienst aktivieren

Der VNC-Dienst wurde im System installiert. VNC ist standardmäßig deaktiviert. Sie müssen es in der Konfiguration aktivieren.

Schritt 1

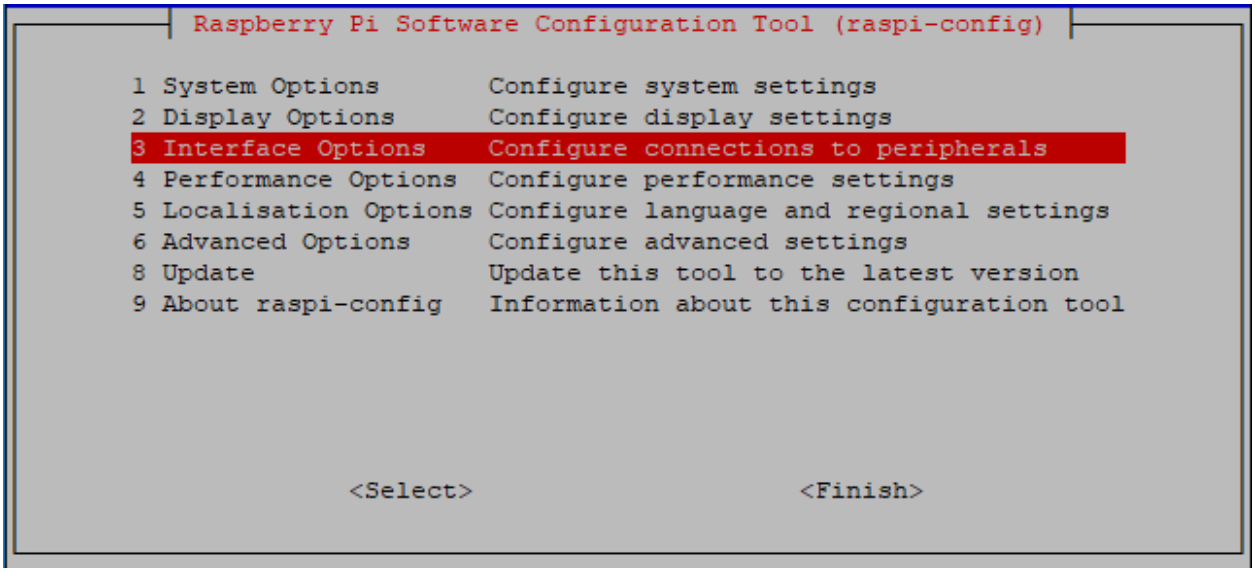
Geben Sie den folgenden Befehl ein:

```
sudo raspi-config
```



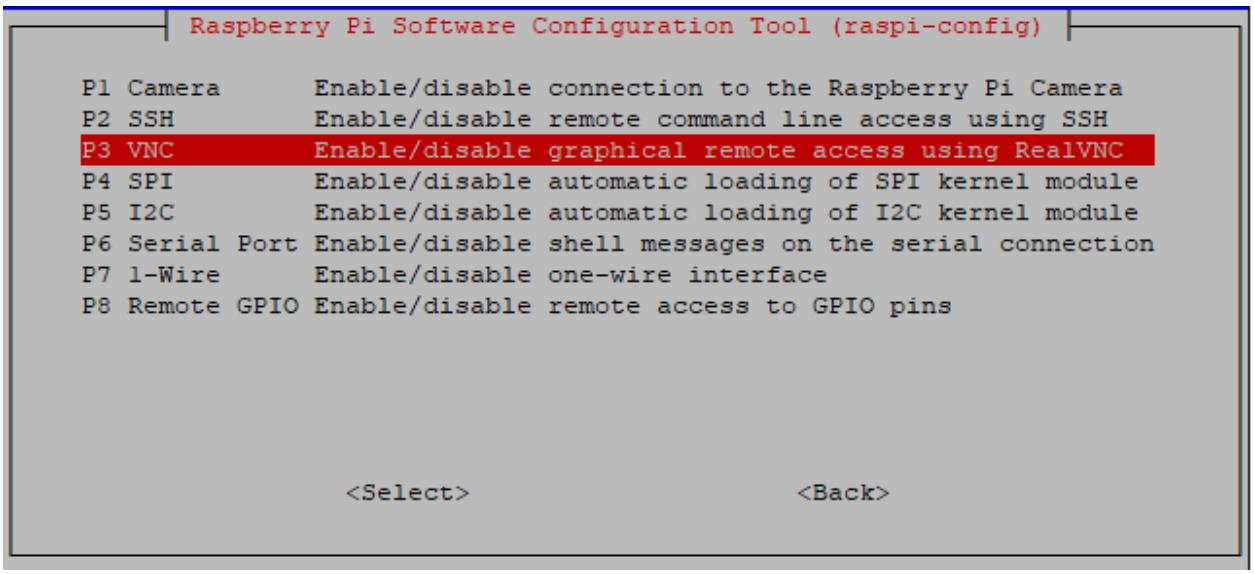
Schritt 2

Wählen Sie **3 Interfacing Options** , indem Sie die Abwärtspfeiltaste auf Ihrer Tastatur drücken und dann die **Enter** drücken.



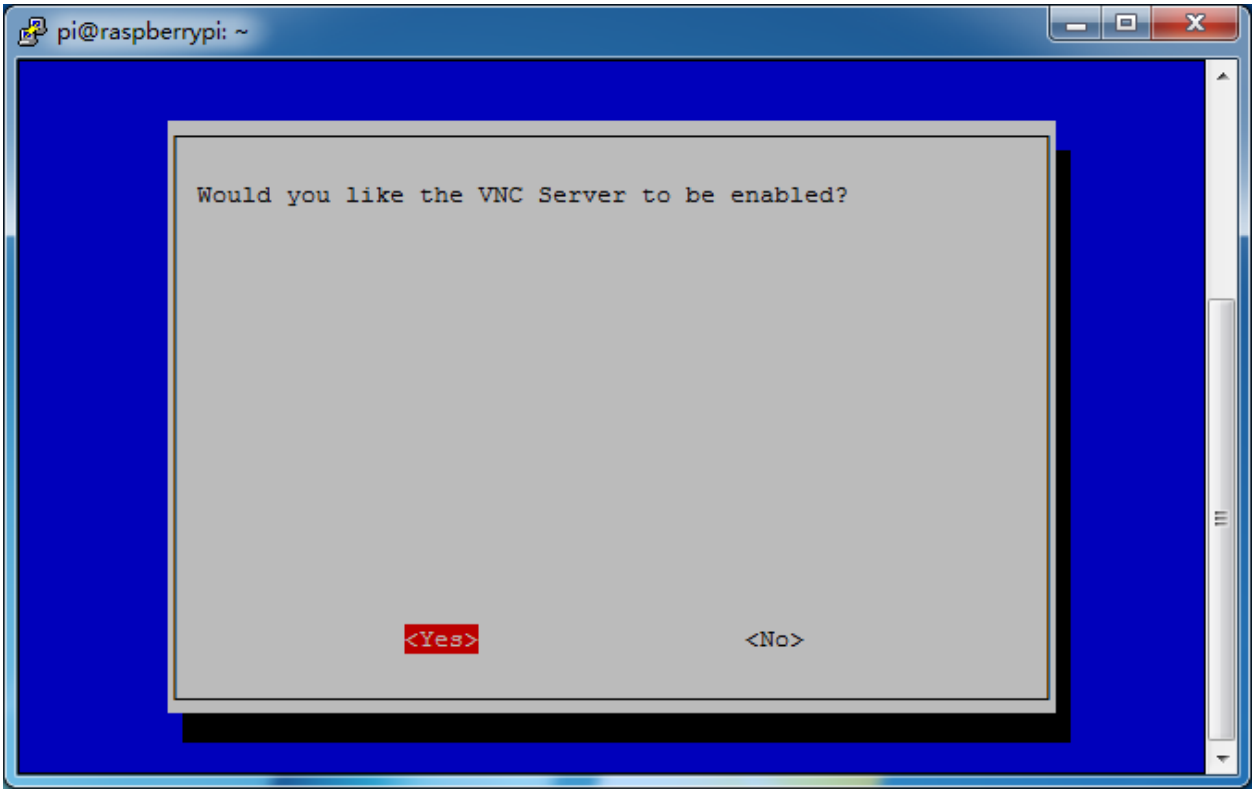
Schritt 3

P3-VNC



Schritt 4

Wählen Sie **Yes -> OK -> Finish** , um die Konfiguration zu beenden.



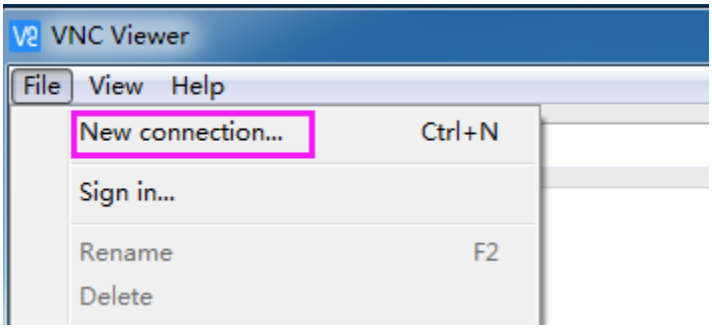
Bei VNC anmelden

Schritt 1

Sie müssen den VNC Viewer herunterladen und auf Ihrem PC installieren. Nachdem die Installation abgeschlossen ist, öffnen Sie es.

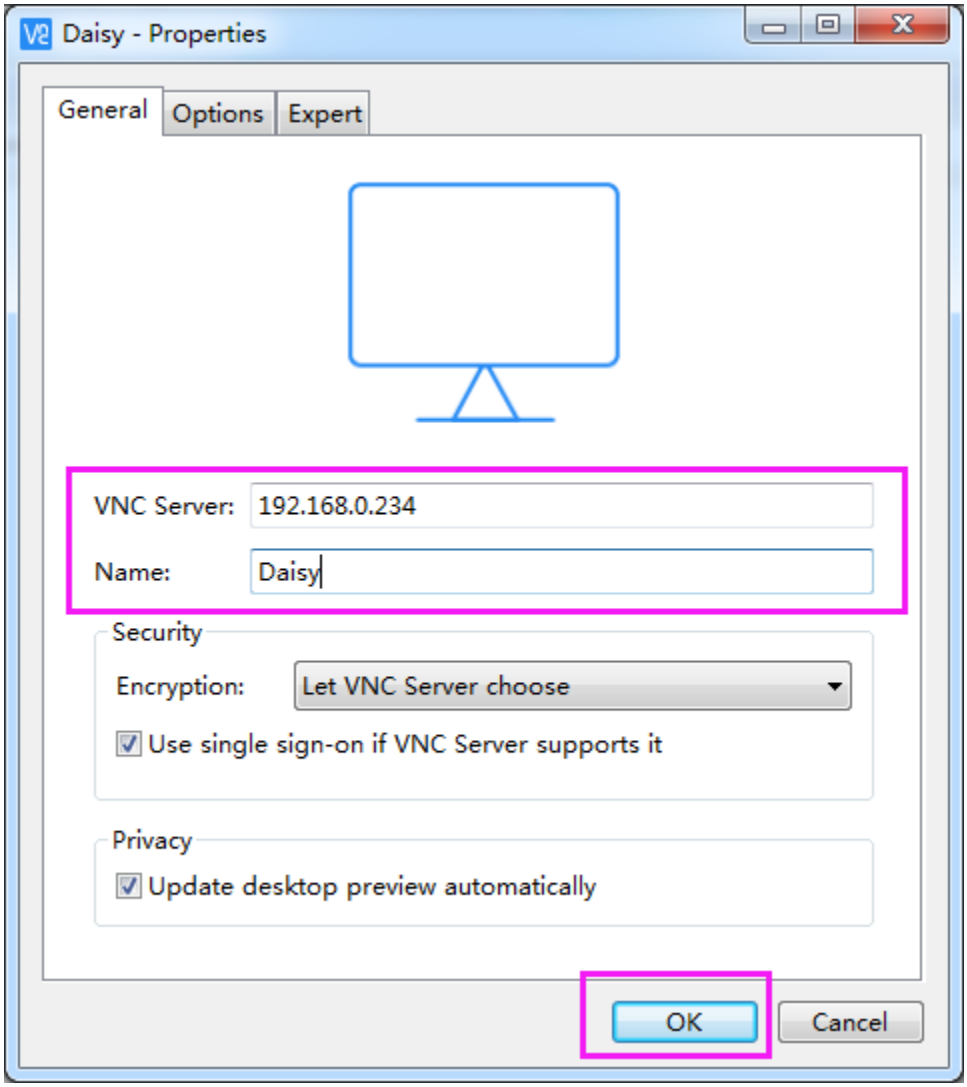
Schritt 2

Wählen Sie dann "**New connection**".



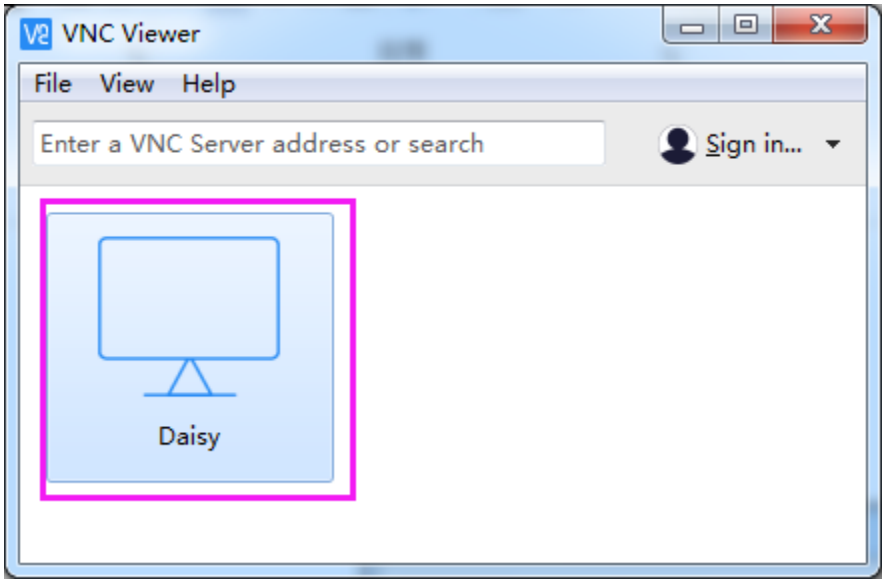
Schritt 3

Geben Sie die IP-Adresse des Raspberry Pi und einen beliebigen **Name** ein.



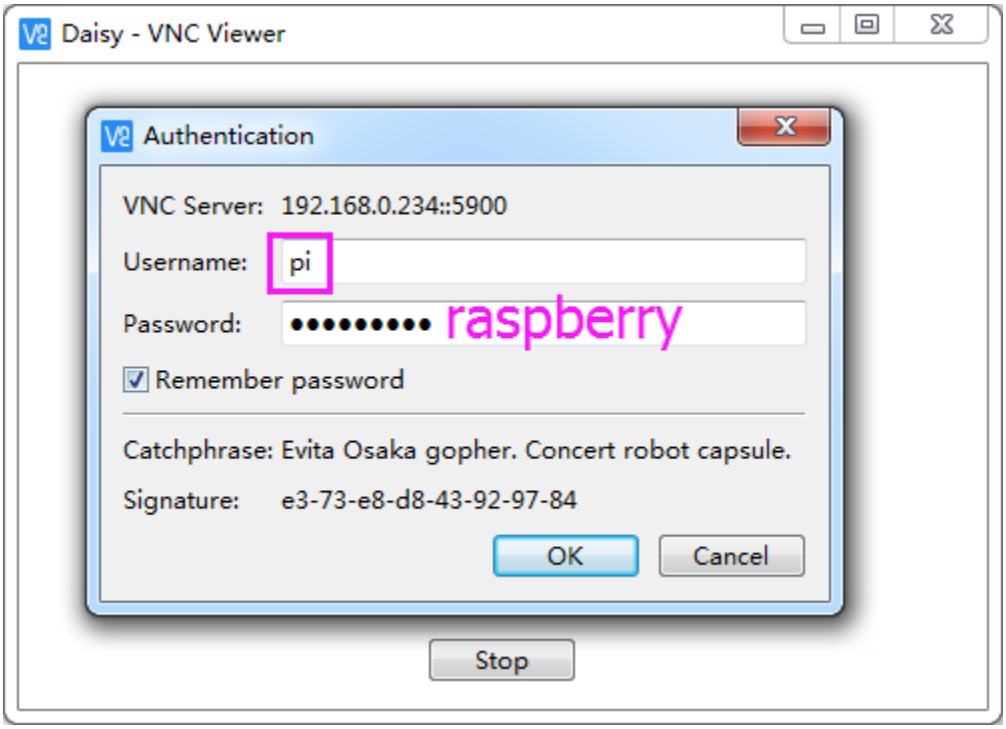
Schritt 4

Doppelklicken Sie auf die soeben erstellte **connection** :



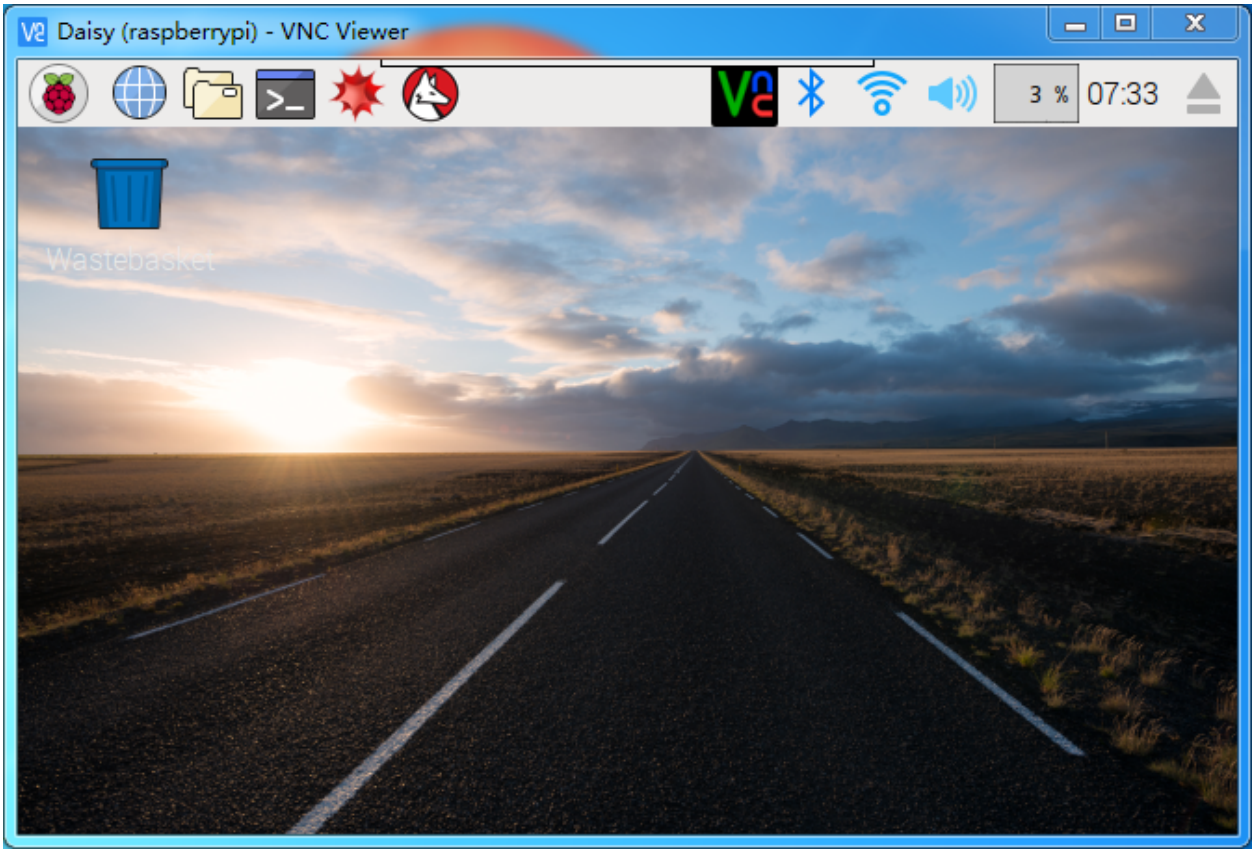
Schritt 5

Geben Sie den Benutzernamen (`pi`) und das Passwort (standardmäßig `raspberry`) ein.



Schritt 6

Nun sehen Sie den Desktop des Raspberry Pi:



Das ist das Ende des VNC-Teils.

XRDP

Eine andere Methode des Remote-Desktops ist XRDP. Sie bietet eine grafische Anmeldung an Remote-Computern mithilfe von RDP (Microsoft Remote Desktop Protocol).

XRDP installieren

Schritt 1

Melden Sie sich mit SSH beim Raspberry Pi an.

Schritt 2

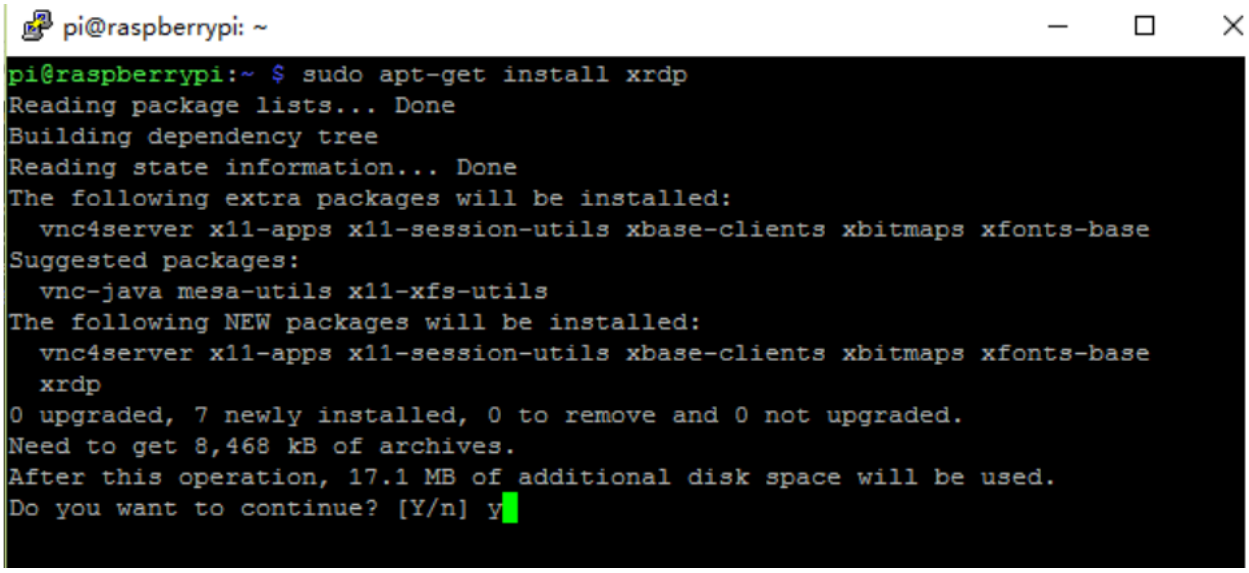
Geben Sie die folgenden Anweisungen ein, um XRDP zu installieren.

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

Schritt 3

Später startet die Installation.

Geben Sie „Y“ ein, drücken Sie zur Bestätigung die Taste „Enter“.



Schritt 4

Nachdem Sie die Installation abgeschlossen haben, sollten Sie sich mit Windows-Remote-Desktop-Anwendungen bei Ihrem Raspberry Pi anmelden.

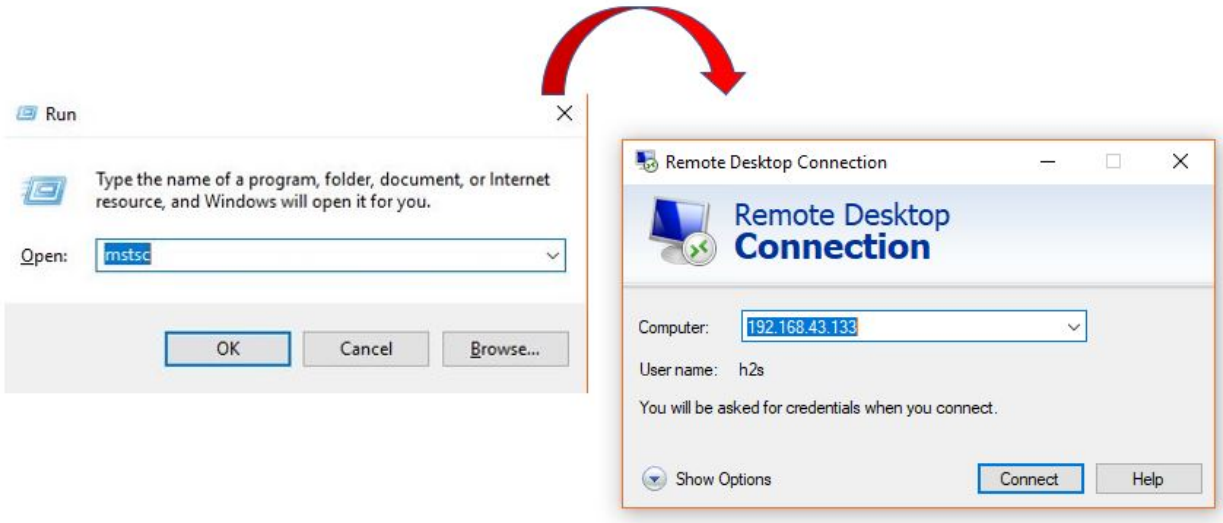
Bei XRDP anmelden

Schritt 1

Wenn Sie ein Windows-Benutzer sind, können Sie die mit Windows gelieferte Remotedesktopfunktion verwenden. Wenn Sie ein Mac-Benutzer sind, können Sie Microsoft Remote Desktop aus dem APP Store herunterladen und verwenden, und es gibt keinen großen Unterschied zwischen den beiden. Das nächste Beispiel ist der Windows-Remotedesktop.

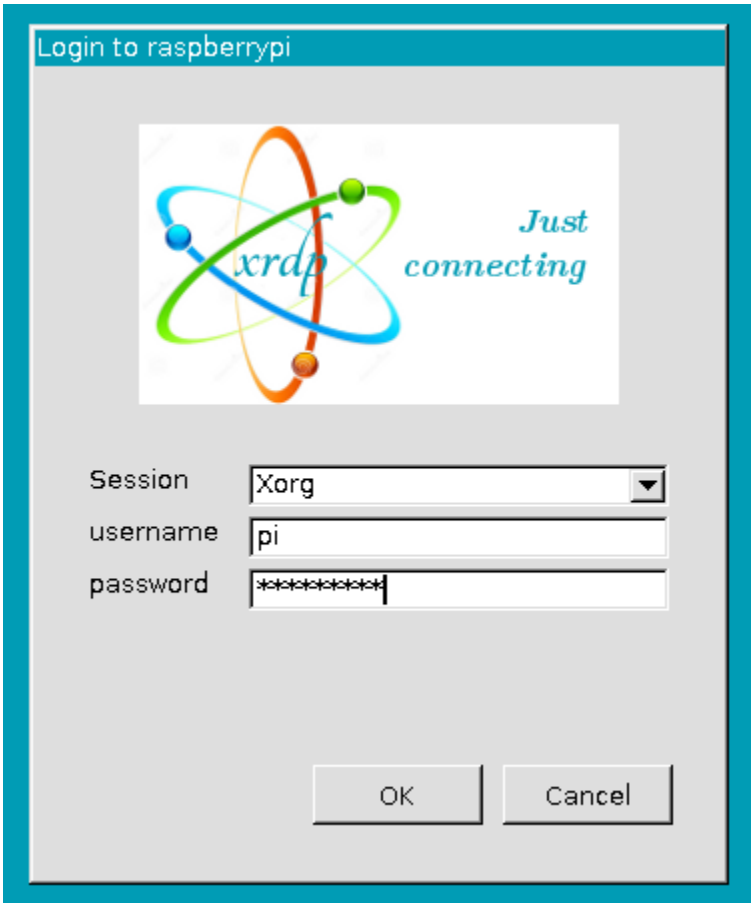
Schritt 2

Geben Sie „mstsc“ in Run (WIN+R) ein, um die Remotedesktopverbindung zu öffnen, geben Sie die IP-Adresse des Raspberry Pi ein und klicken Sie dann auf „Connect“.



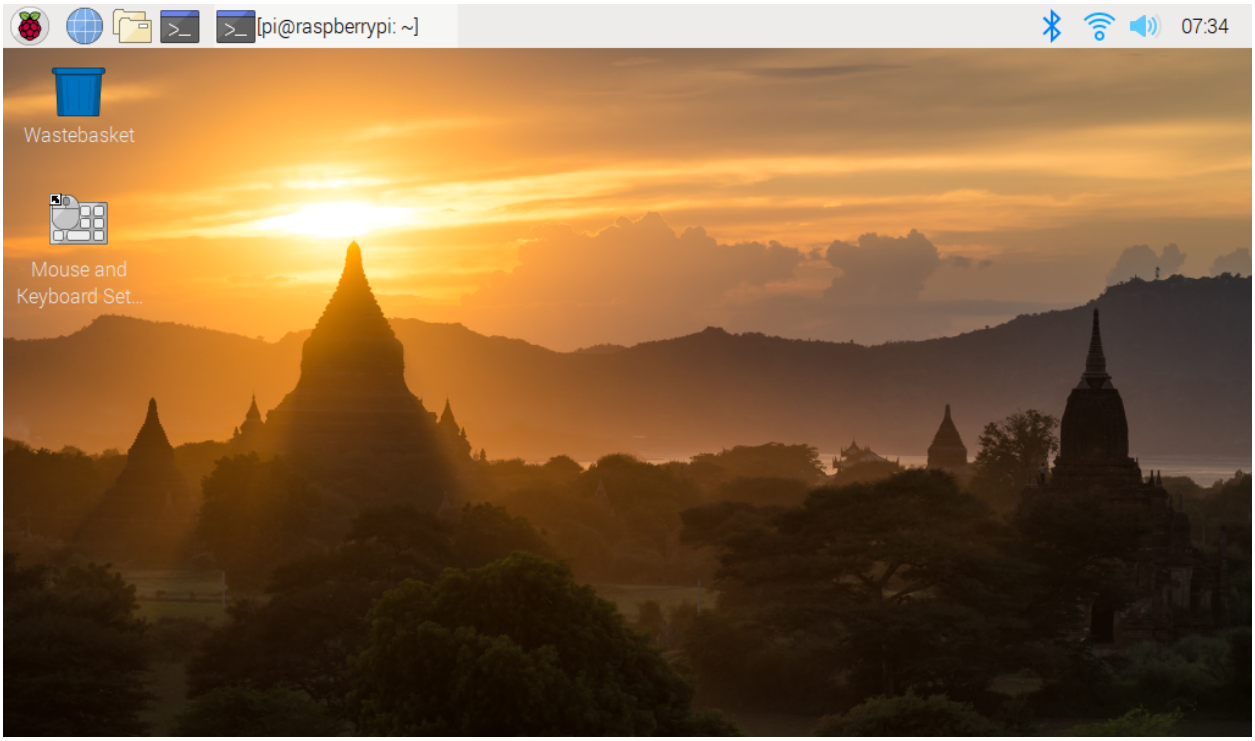
Schritt 3

Dann erscheint die xrdp-Anmeldeseite. Bitte geben Sie Ihren Benutzernamen und Ihr Passwort ein. Danach klicken Sie bitte auf „OK“. Bei der ersten Anmeldung lautet Ihr Benutzername „pi“ und das Passwort „raspberry“.



Schritt 4

Hier melden Sie sich erfolgreich über den Remote-Desktop bei RPi an.



1.15.4 Erstellen einer virtuellen Umgebung

Bei Verwendung von Raspberry Pi oder ähnlichen Geräten wird empfohlen, Pakete mit `pip` in einer virtuellen Umgebung zu installieren. Diese bietet eine Abhängigkeitsisolierung, erhöht die Systemsicherheit, gewährleistet die Sauberkeit des Systems und erleichtert Projektmigration und -freigabe, was die Abhängigkeitsverwaltung vereinfacht. Diese Vorteile machen virtuelle Umgebungen zu einem äußerst wichtigen und nützlichen Werkzeug in der Python-Entwicklung.

Im Folgenden finden Sie die Schritte zur Erstellung einer virtuellen Umgebung:

1. Erstellen einer virtuellen Umgebung

Zunächst müssen Sie sicherstellen, dass Ihr System Python installiert hat. Python-Version 3.3 und später werden mit dem `venv`-Modul geliefert, um virtuelle Umgebungen zu erstellen, was die separate Installation überflüssig macht. Wenn Sie Python 2 oder eine Version vor Python 3.3 verwenden, müssen Sie `virtualenv` installieren.

- Für Python 3:

Python-Versionen 3.3 und später können direkt das `venv`-Modul verwenden:

```
python3 -m venv myenv
```

Dies erstellt eine virtuelle Umgebung namens `myenv` im aktuellen Verzeichnis.

- Für Python 2:

Wenn Sie immer noch Python 2 verwenden, müssen Sie zuerst `virtualenv` installieren:

```
pip install virtualenv
```

Erstellen Sie dann eine virtuelle Umgebung:

```
virtualenv myenv
```

Dies erstellt ebenfalls eine virtuelle Umgebung namens `myenv` im aktuellen Verzeichnis.

2. Aktivieren der virtuellen Umgebung

Nachdem Sie die virtuelle Umgebung erstellt haben, müssen Sie sie für die Verwendung aktivieren.

Bemerkung: Jedes Mal, wenn Sie den Raspberry Pi neu starten oder ein neues Terminal öffnen, müssen Sie erneut den folgenden Befehl ausführen, um die virtuelle Umgebung zu aktivieren.

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

Sobald die virtuelle Umgebung aktiviert ist, sehen Sie den Umgebungsnamen vor dem Befehlszeilen-Prompt, was darauf hinweist, dass Sie innerhalb der virtuellen Umgebung arbeiten.

3. Abhängigkeiten installieren

Mit der aktivierten virtuellen Umgebung können Sie pip verwenden, um die erforderlichen Abhängigkeiten zu installieren. Zum Beispiel:

```
pip install requests
```

Dies installiert die Requests-Bibliothek in die aktuelle virtuelle Umgebung anstelle der globalen Umgebung. Dieser Schritt muss nur einmal ausgeführt werden.

4. Verlassen der virtuellen Umgebung

Wenn Sie Ihre Arbeit abgeschlossen haben und die virtuelle Umgebung verlassen möchten, führen Sie einfach folgenden Befehl aus:

```
deactivate
```

Dadurch kehren Sie zur globalen Python-Umgebung des Systems zurück.

5. Löschen der virtuellen Umgebung

Wenn Sie eine bestimmte virtuelle Umgebung nicht mehr benötigen, können Sie einfach das Verzeichnis löschen, das die virtuelle Umgebung enthält:

```
rm -rf myenv
```

1.16 FAQ

1.16.1 C-Code funktioniert nicht?

- Überprüfen Sie Ihre Verkabelung auf Probleme.
- Check if the code is reporting errors, if so, refer to: *Installieren und überprüfen Sie das WiringPi* .
- Wurde der Code vor der Ausführung kompiliert.
- If all the above 3 conditions are OK, it may be that your wiringPi version (2.50) is not compatible with your Raspberry Pi 4B and above, refer to Check and Install the WiringPi to manually upgrade it to version 2.52.

KAPITEL 2

Urheberrechtshinweis

Alle Inhalte, einschließlich, aber nicht beschränkt auf Texte, Bilder und Kode in diesem Handbuch, sind Eigentum der SunFounder Unternehmen. Sie sollten es nur für persönliche Studien, Nachforschungen, Genüsse oder andere nicht-kommerzielle oder gemeinnützige Zwecke gemäß den entsprechenden Bestimmungen und Urheberrechtsgesetzen verwenden, ohne die gesetzlichen Rechte des Autors und der relevanten Rechteinhaber zu verletzen. Für jede Person oder Organisation, die diese ohne Erlaubnis für kommerzielle Zwecke nutzt, behält sich das Unternehmen das Recht vor, rechtliche Schritte einzuleiten.