SunFounder raphael-kit

www.sunfounder.com

Nov 08, 2022

CONTENTS

1 About Video Course **3**

2 Components List and Introduction **5**
2.1 Components List . . . 5
2.2 Components Introductions . . . 5

3 Install and Setup Raspberry Pi OS **75**
3.1 What Do We Need? . . . 75
3.2 Installing the OS . . . 76
3.3 Set up Your Raspberry Pi . . . 84

4 GPIO Extension Board **91**

5 Download the Code **93**

6 Play with Python **95**
6.1 Check the `RPi.GPIO` . . . 95
6.2 Output . . . 96
6.3 Input . . . 158
6.4 Audiovisual . . . 261
6.5 IoT . . . 274
6.6 Extension . . . 307

7 Play with C **433**
7.1 Install and Check the WiringPi . . . 433
7.2 Output . . . 434
7.3 Input . . . 500
7.4 Extension . . . 609

8 Play with Processing **693**
8.1 What is Processing . . . 693
8.2 Install the Processing . . . 693
8.3 Install Hardware I/O . . . 694
8.4 Projects . . . 696

9 Play with Nodejs **723**
9.1 What is Nodejs . . . 723
9.2 Install or update nodejs and npm . . . 723
9.3 Check the pigpio . . . 723
9.4 Output . . . 726
9.5 Input . . . 780

9.6 Extension . . . 861

10 Play with Scratch **865**
10.1 Quick Guide on Scratch . . . 865
10.2 Projects . . . 874

11 Appendix **985**
11.1 Install the Libraries . . . 985
11.2 I2C Configuration . . . 987
11.3 SPI Configuration . . . 989
11.4 Audio Configuration . . . 991
11.5 Remote Desktop . . . 997
11.6 Filezilla Software . . . 1006
11.7 How to use Blynk on mobile device? . . . 1008

12 FAQ **1011**
12.1 C code is not working? . . . 1011

13 Thank You **1013**

Are you new to Raspberry Pi? Are you looking for a clear path to learn Raspberry Pi? Would you like to do more than just copy and paste code, but actually write your own? If you answered yes to any of the above questions, then the Raspberry Pi Education Starter Kit - Raphael Kit is right for you.

The kit is unlike other kits that only have a variety of projects. It is a true educational kit designed for beginners, whether you are programming beginners, electronics hobbyists, experienced electrical engineers, students or educators, this kit will meet all your needs.

In addition to including all the hardware and software needed for beginners, the kit also offers an online tutorial with 161 interesting projects in 5 programming languages and 45 free video courses (30 hours) on getting started to mastery.

This video course shows beginners how to set up the Raspberry Pi, use the GPIO pins and sensors, and learn the basics of circuits and programming. Each course has simple and interesting projects for beginners to practice and learn, all you need to do is to follow step by step and eventually you will be able to master the Raspberry Pi and make your own projects.

After learning the video tutorials, you can practice the use of each component from our online tutorials and play with some more advanced and interesting projects.

Alternatively, you can use the components to build projects in other languages, such as C, Scratch, Java (processing) and JavaScript (Nodejs).

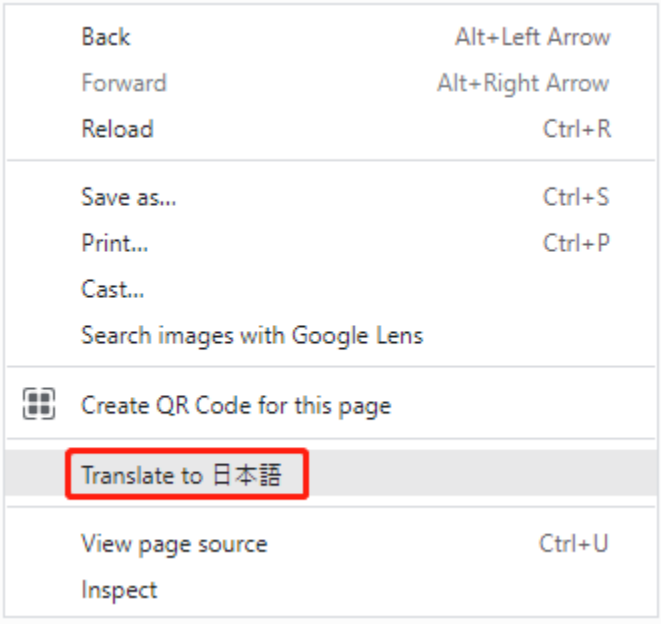
If you have any questions, please send an email to cs@sunfounder.com and we will respond as soon as possible.

About the display language

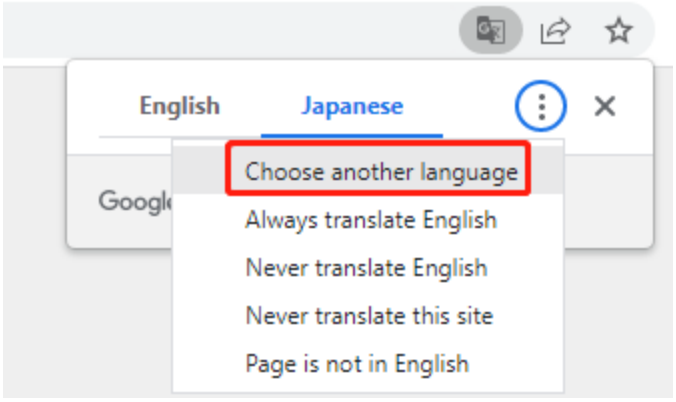
In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

The steps are as follows.

- In this course page, right-click and select **Translate to xx**. If the current language is not what you want, you can change it later.

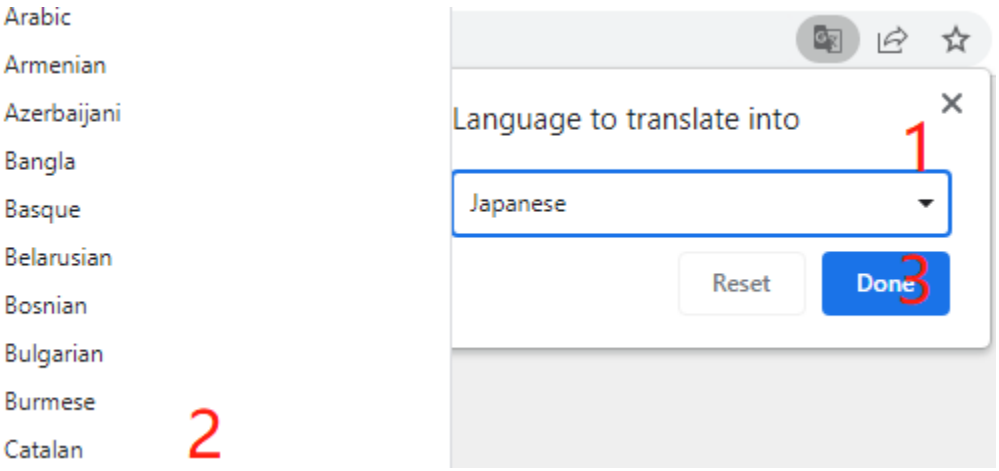


- There will be a language popup in the upper right corner. Click on the menu button to **choose another language**.



- Select the language from the inverted triangle box, and then click **Done**.

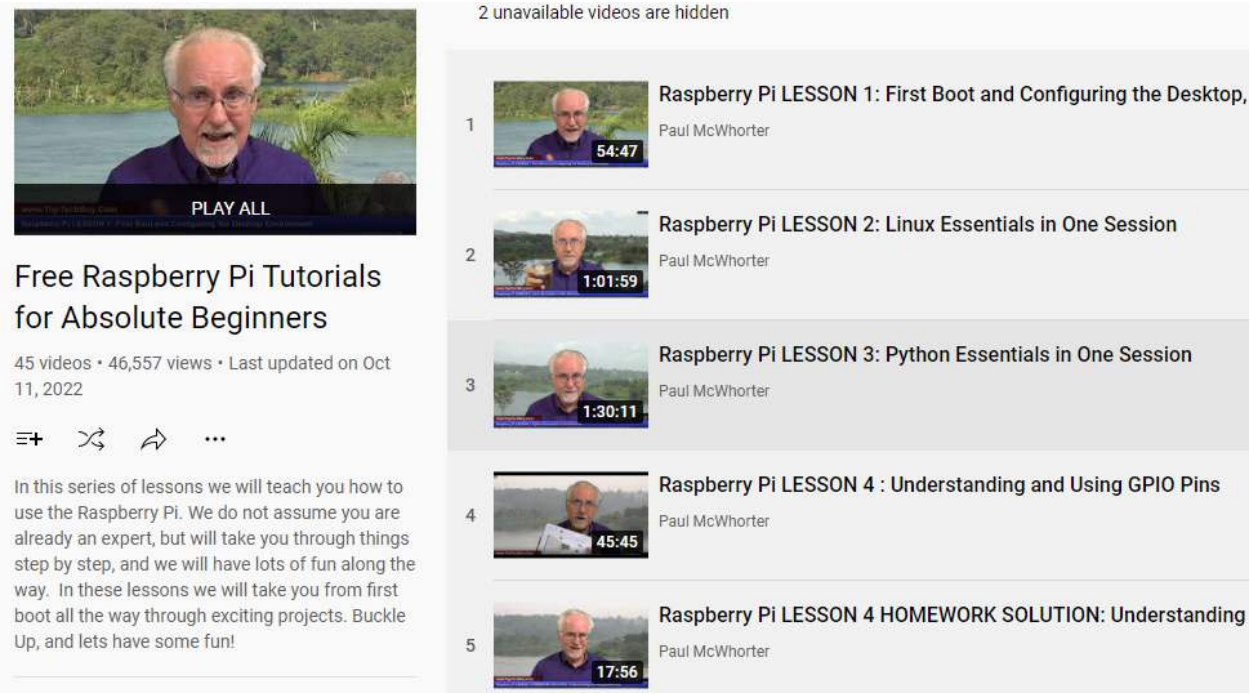
Contents



CHAPTER

ONE

ABOUT VIDEO COURSE



The kit also contains 45 free video lessons (30 hours) produced by SunFounder and Paul McWhorter. Paul is an experienced programming educator with 300,000 followers on YouTube and currently has 45 Raspberry Pi video lessons that continue to be updated weekly.

The video course is made for beginners who have no basic programming skills and use the Sunfounder Ultimate Raspberry Pi Kit - Raphael Kit.

Here is the video linkhttps://www.youtube.com/playlist?list=PLGs0VKk2DiYxdMjCJmcP6jt4Yw6OHK85O

From the video, you can

- Master the installation, configuration and use of the Raspberry Pi system.
- Understanding and using Raspberry Pi GPIO pins.
- Learn Python's programming logic so you can control what happens in your scripts.
- Acquire the skill to create Python libraries to better prototype your ideas.
- Utilize multithreading to run multiple scripts at once.
- Know how to get and use analog signals from ADC module.
- Learn the circuit principles so that you can design the circuit to your own specifications.

- Get to know Joystick, Servo, Ultrasonic module, PIR, DHT11, I2C LCD, Keyboard, etc.

You don't have to worry about learning programming too hard, because we don't consider you an expert and will take you step by step, with simple and interesting projects to learn each point and make sure every beginner can learn.

Through the video tutorials, you can make temperature alarm systems, light sensing systems and security monitoring systems, all of which are very common and interesting projects in life.

CHAPTER

TWO

COMPONENTS LIST AND INTRODUCTION

2.1 Components List

After opening the package, please check whether the quantity of components is compliance with product description and whether all components are in good condition.

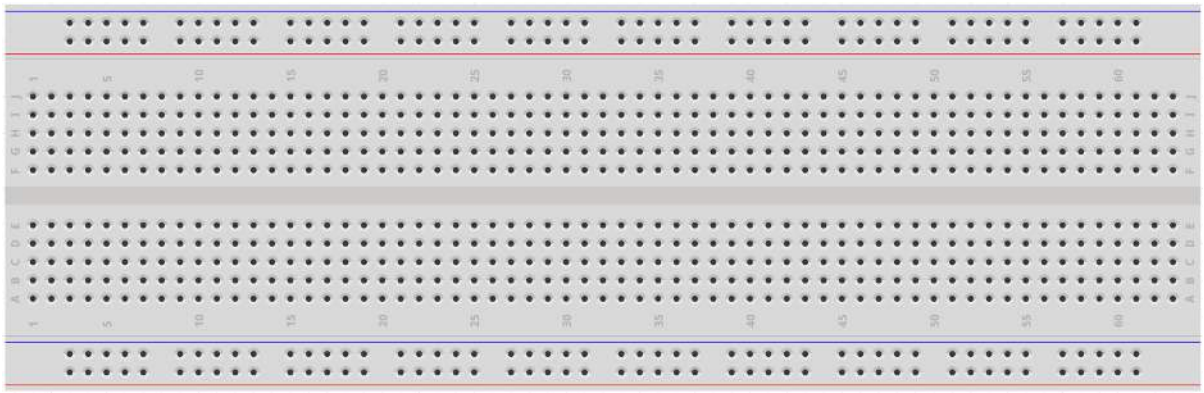
- `Components List`

2.2 Components Introductions

Below is the introduction to each component, which contains the operating principle of the component and the corresponding projects.

Basic

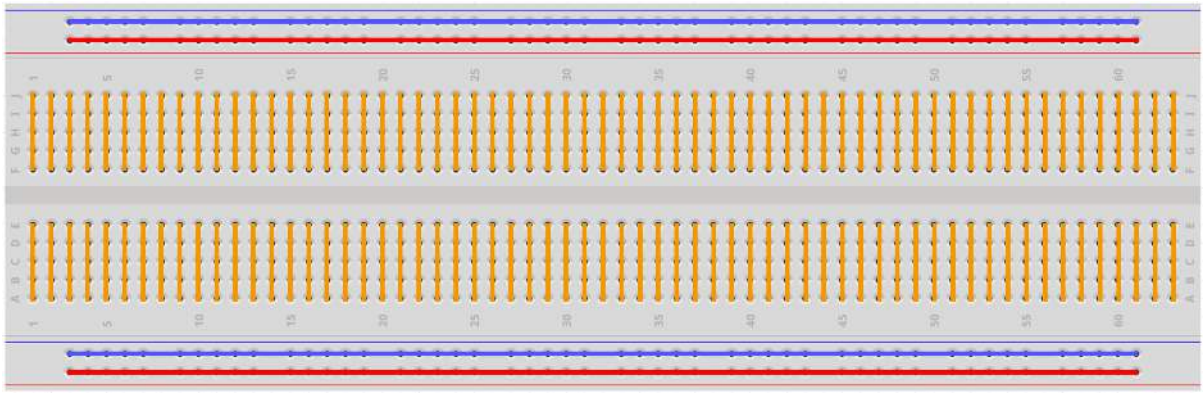
2.2.1 Breadboard



A breadboard is a construction base for prototyping of electronics. Originally the word referred to a literal bread board, a polished piece of wood used for slicing bread.[1] In the 1970s the solderless breadboard (a.k.a. plugboard, a terminal array board) became available and nowadays the term “breadboard” is commonly used to refer to these.

It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

The picture shows the internal structure of a breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



If you want to know more about breadboard, refer to: How to Use a Breadboard - Science Buddies

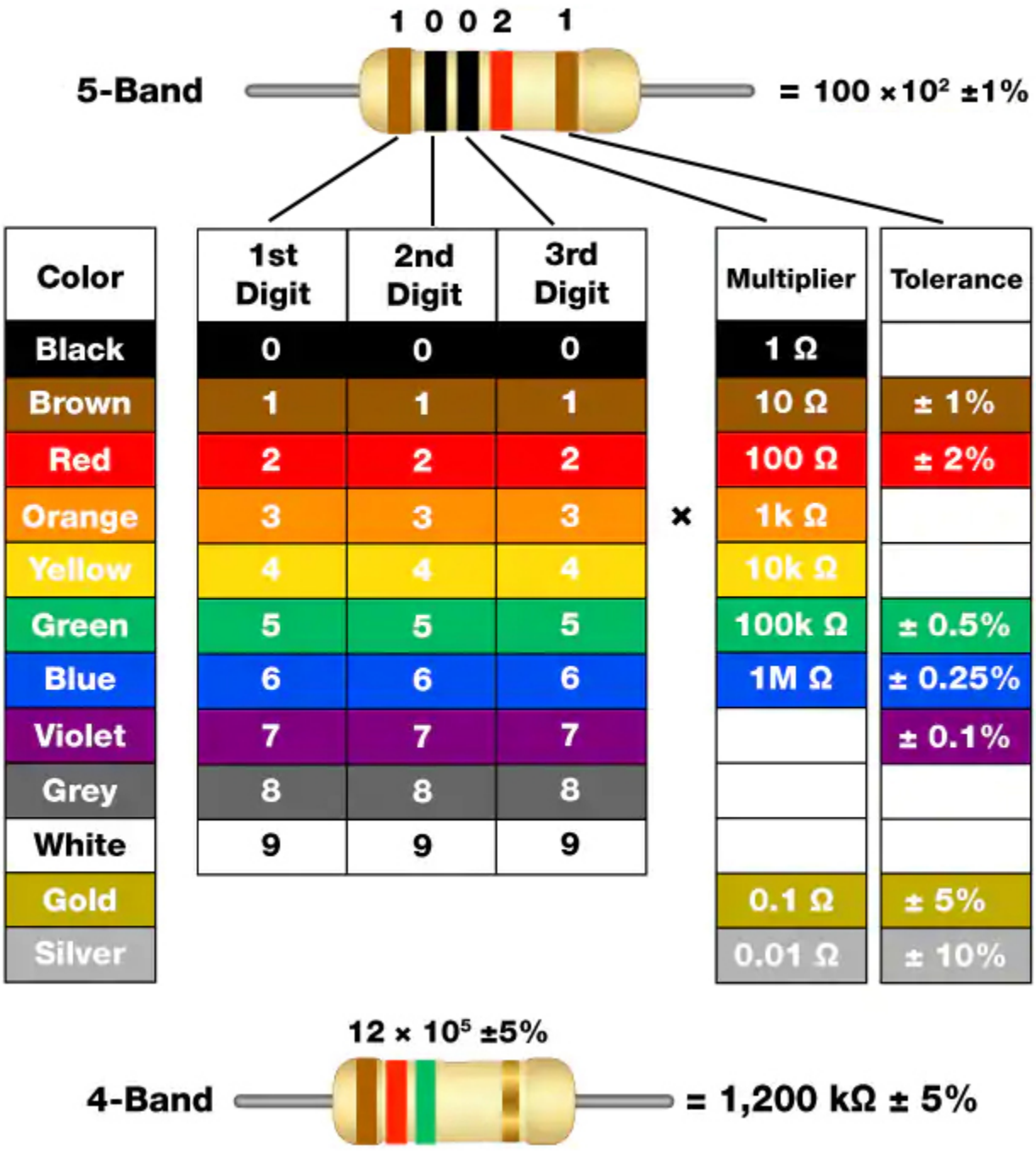
2.2.2 Resistor

Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

Two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.

 is the unit of resistance and the larger units include K, M, etc. Their relationship can be shown as follows: 1 M=1000 K, 1 K = 1000 . Normally, the value of resistance is marked on it.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster.



As shown in the card, each color stands for a number.

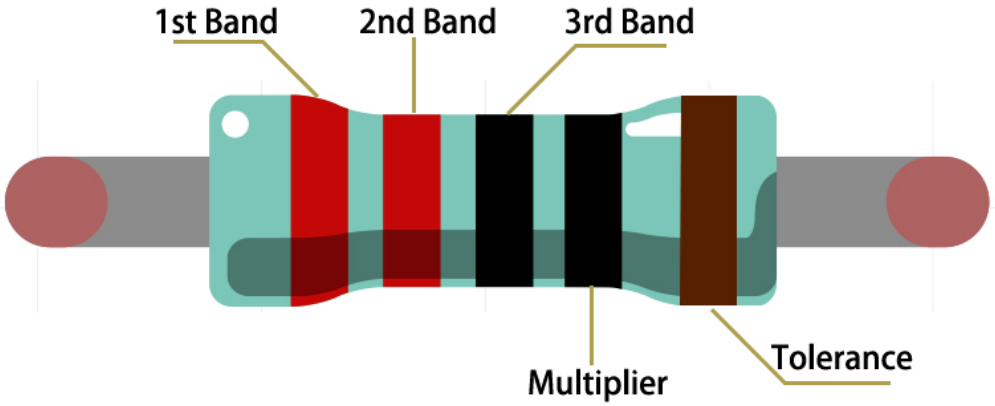
Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White	Gold	Silver
0	1	2	3	4	5	6	7	8	9	0.1	0.01

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands.

Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4th and 5th band will be comparatively larger.

Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.

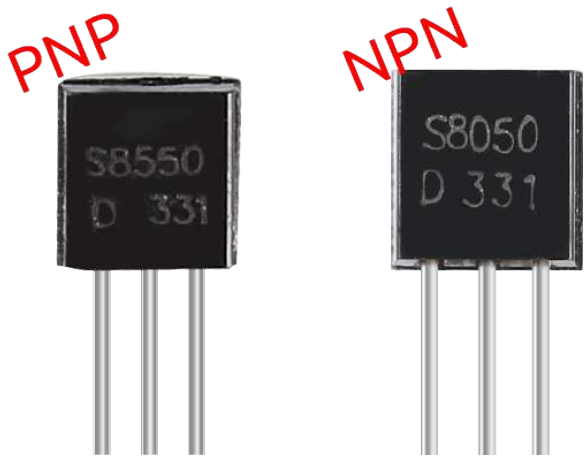
Let's see how to read the resistance value of a 5-band resistor as shown below.



So for this resistor, the resistance should be read from left to right. The value should be in this format: 1st Band 2nd Band 3rd Band x 10^Multiplier () and the permissible error is ±Tolerance%. So the resistance value of this resistor is 2(red) 2(red) 0(black) x 10^0(black) = 220 , and the permissible error is ± 1% (brown).

You can learn more about resistor from Wiki: Resistor - Wikipedia.

2.2.3 Transistor



Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch.

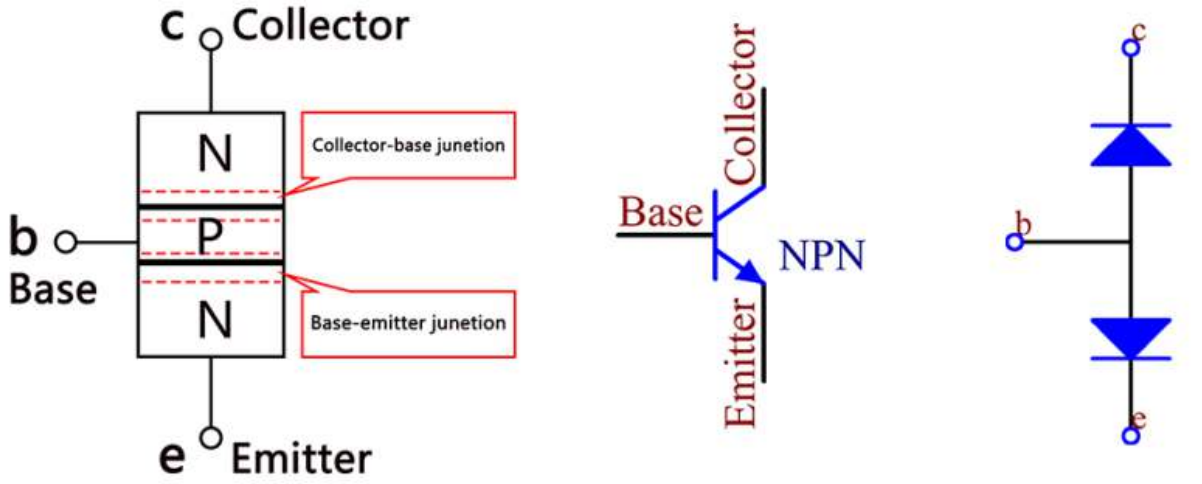
A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier. From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction.

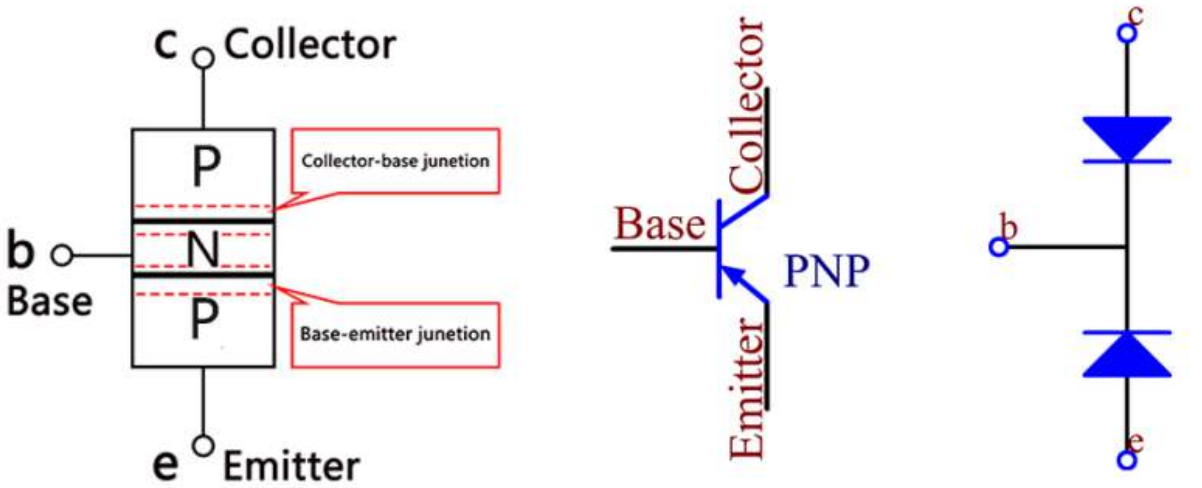
- P–N junction - Wikipedia

Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is

the opposite. See the figure below.

Note: s8550 is PNP transistor and the s8050 is the NPN one, They look very similar, and we need to check carefully to see their labels.





When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

Put the label side facing us and the pins facing down. The pins from left to right are emitter(e), base(b), and collector(c).

- S8050 Transistor Datasheet
- S8550 Transistor Datasheet

Example

- *1.2.1 Active Buzzer* (C Project S8550)
- *1.3.3 Relay* (C Project S8050)
- *1.2.2 Passive Buzzer* (Python Project S8550)
- *1.3.3 Relay* (Python Project S8050)
- *1.14 123 Wooden Man* (Scratch Project S8550)

2.2.4 Capacitor

Capacitor, refers to the amount of charge storage under a given potential difference, denoted as C, and the international unit is farad (F). Generally speaking, electric charges move under force in an electric field. When there is a medium between conductors, the movement of electric charges is hindered and the electric charges accumulate on the conductors, resulting in accumulation of electric charges.

The amount of stored electric charges is called capacitance. Because capacitors are one of the most widely used electronic components in electronic equipment, they are widely used in direct current isolation, coupling, bypass, filtering, tuning loops, energy conversion, and control circuits. Capacitors are divided into electrolytic capacitors, solid capacitors, etc.

According to material characteristics, capacitors can be divided into: aluminum electrolytic capacitors, film capacitors, tantalum capacitors, ceramic capacitors, super capacitors, etc.

In this kit, ceramic capacitors and electrolytic capacitors are used.

- Ceramic Capacitor - Wikipedia
- Electrolytic Capacitor - Wikipedia

There are 103 or 104 label on the ceramic capacitors, which represent the capacitance value, 103=10x10^3pF, 104=10x10^4pF

Unit Conversion

1F=10^3mF=10^6uF=10^9nF=10^12pF

Example

- *2.1.2 Micro Switch* (C Project)
- *2.1.4 Slide Switch* (C Project)
- *3.1.9 Alarm Bell* (C Project)
- *2.1.2 Micro Switch* (Python Project)
- *4.1.15 Alarm Bell* (Python Project)

- *1.8 Service Bell* (scratch Project)

2.2.5 Diode

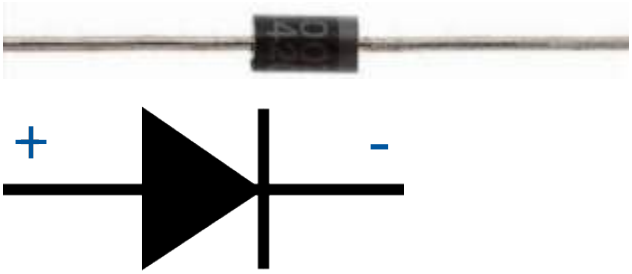
A diode is an electronic component with two electrodes. It allows current to flow in only one direction, which is often called the "Rectifying" function. Thus, a diode can be thought of as an electronic version of a check valve.

Because of its unidirectional conductivity, the diode is used in almost all electronic circuits of some complexity. It is one of the first semiconductor devices and has a wide range of applications.

According to its use classification, it can be divided into detector diodes, rectifier diodes, limiter diodes, voltage regulator diodes, etc.

Rectifier diodes and voltage regulator diodes are included in this kit.

Rectifier Diode



A rectifier diode is a semiconductor diode, used to rectify AC (alternating current) to DC (direct current) using the rectifier bridge application. The alternative of rectifier diode through the Schottky barrier is mainly valued within digital electronics. This diode is capable to conduct the values of current which changes from mA to a few kA & voltages up to a few kV.

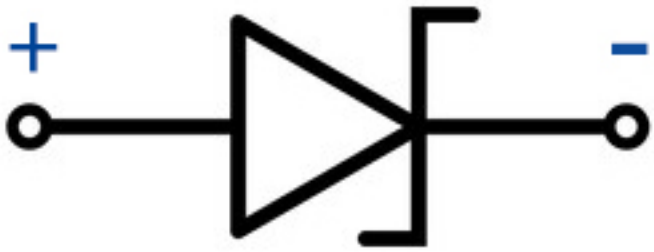
The designing of rectifier diodes can be done with Silicon material and they are capable of conducting high electric current values. These diodes are not famous but still used Ge or gallium arsenide-based semiconductor diodes. Ge diodes have less allowable reversed voltage as well as a lesser allowable junction temperature. The Ge diode has a benefit as compared to Si diode that is low threshold voltage value while operating in a forward-bias.

- 1N400x general-purpose diode - Wikipedia

Zener Diode

A Zener diode is a special type of diode designed to reliably allow current to flow "backwards" when a certain set reverse voltage, known as the Zener voltage, is reached.

This diode is a semiconductor device that has a very high resistance up to the critical reverse breakdown voltage. At this critical breakdown point, the reverse resistance is reduced to a very small value, and the current increases while the voltage remains constant in this low resistance region.



- Zener diode - Wikipedia

Example

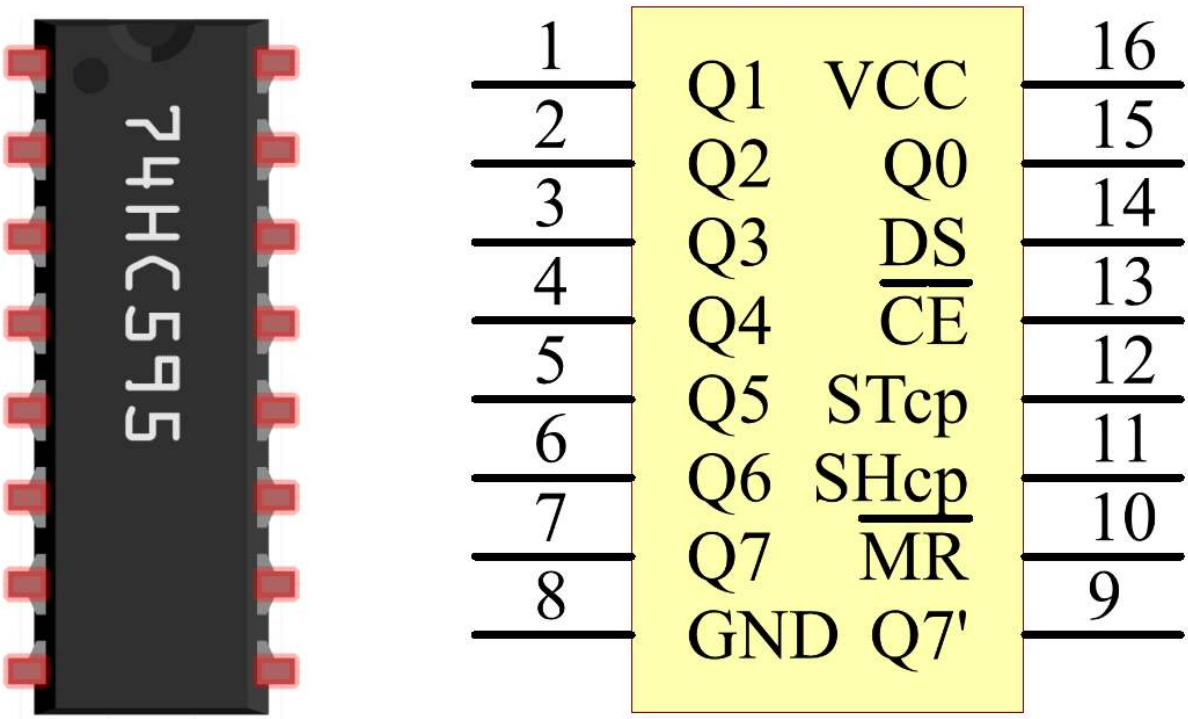
- *1.3.3 Relay* (C Project)
- *1.3.3 Relay* (Python Project)

Chip

2.2.6 74HC595

The 74HC595 consists of an 8bit shift register and a storage register with threestate parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU. When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp. If the two clocks are connected together, the shift register is always one pulse earlier than the memory register. There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register. The memory register outputs a Bus with a parallel 8-bit and in three states. When OE is enabled (low level), the data in memory register is output to the bus.

- 74HC595 Datasheet



Pins of 74HC595 and their functions:

- **Q0-Q7**: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.
- **Q7'**: Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series
- **MR**: Reset pin, active at low level;
- **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**: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.
- **CE**: Output enable pin, active at low level.
- **DS**: Serial data input pin
- **VCC**: Positive supply voltage.
- **GND**: Ground.

Example

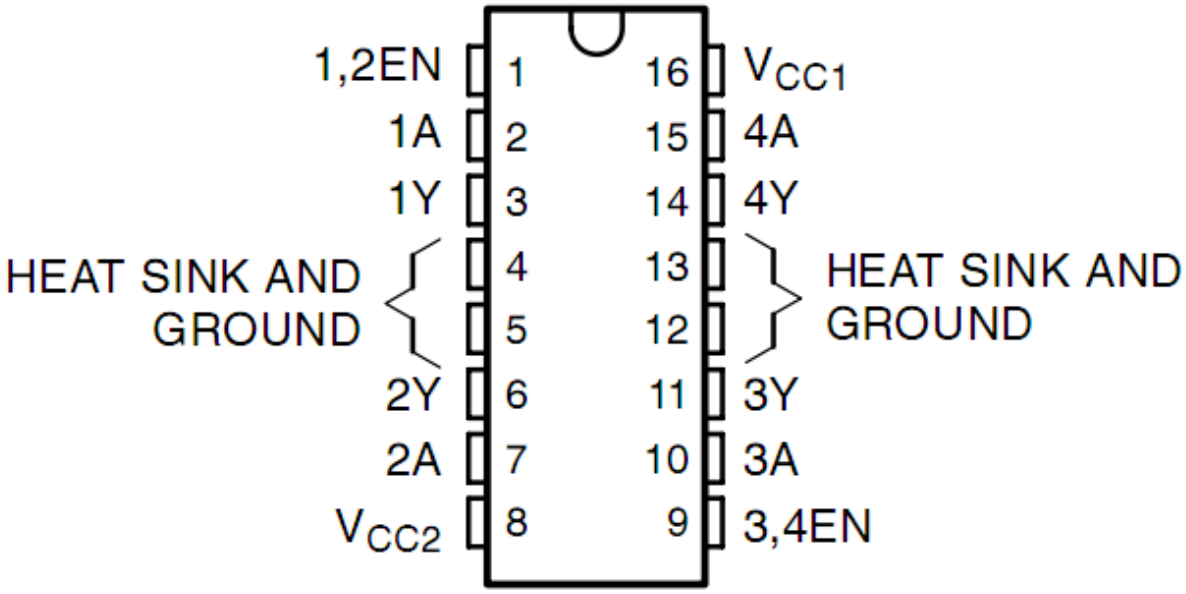
- *1.1.4 7-segment Display* (C Project)
- *1.1.5 4-Digit 7-Segment Display* (C Project)
- *3.1.1 Counting Device* (C Project)
- *3.1.6 Traffic Light* (C Project)
- *3.1.12 GAME - 10 Second* (C Project)
- *1.1.4 7-segment Display* (Python Project)
- *1.1.5 4-Digit 7-Segment Display* (Python Project)
- *4.1.7 Counting Device* (Pyhton Project)

- *4.1.12 Traffic Light* (Python Project)
- *4.1.18 GAME - 10 Second* (Pyhton Project)

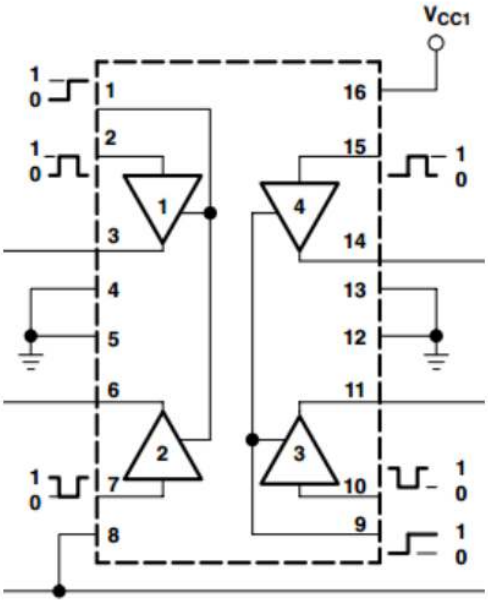
2.2.7 L293D

L293D is a 4-channel motor driver integrated by chip with high voltage and high current. It's designed to connect to standard DTL, TTL logic level, and drive inductive loads (such as relay coils, DC, Stepper Motors) and power switching transistors etc. DC Motors are devices that turn DC electrical energy into mechanical energy. They are widely used in electrical drive for their superior speed regulation performance.

See the figure of pins below. L293D has two pins (Vcc1 and Vcc2) for power supply. Vcc2 is used to supply power for the motor, while Vcc1 to supply for the chip. Since a small-sized DC motor is used here, connect both pins to +5V.



The following is the internal structure of L293D. Pin EN is an enable pin and only works with high level; A stands for input and Y for output. You can see the relationship among them at the right bottom. When pin EN is High level, if A is High, Y outputs high level; if A is Low, Y outputs Low level. When pin EN is Low level, the L293D does not work.



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)

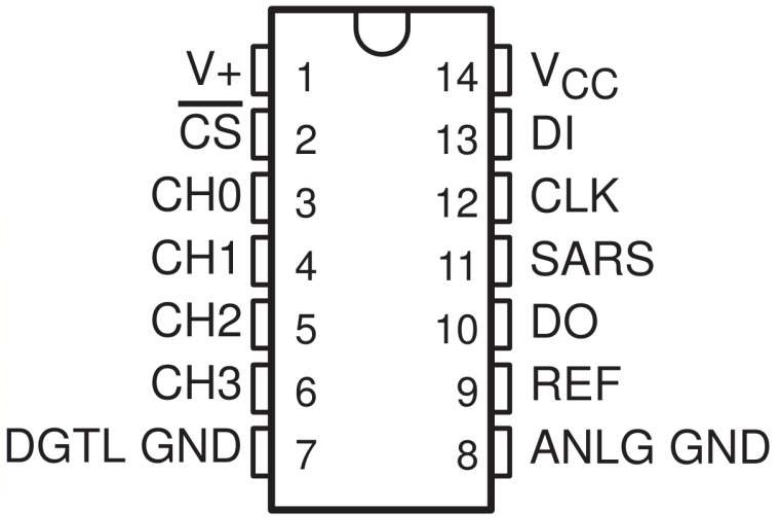
- L293D Datasheet

Example

- *1.3.1 Motor* (C Project)
- *3.1.4 Smart Fan* (C Project)
- *1.3.1 Motor* (Python Project)
- *4.1.10 Smart Fan* (Python Project)
- *1.17 Rotating fan* (Scratch Project)

2.2.8 ADC0834

ADC0834 is an 8-bit successive approximation analog-to-digital converter that is equipped with an input-configurable multichannel multi-plexer and serial input/output. The serial input/output is configured to interface with standard shift registers or microprocessors.

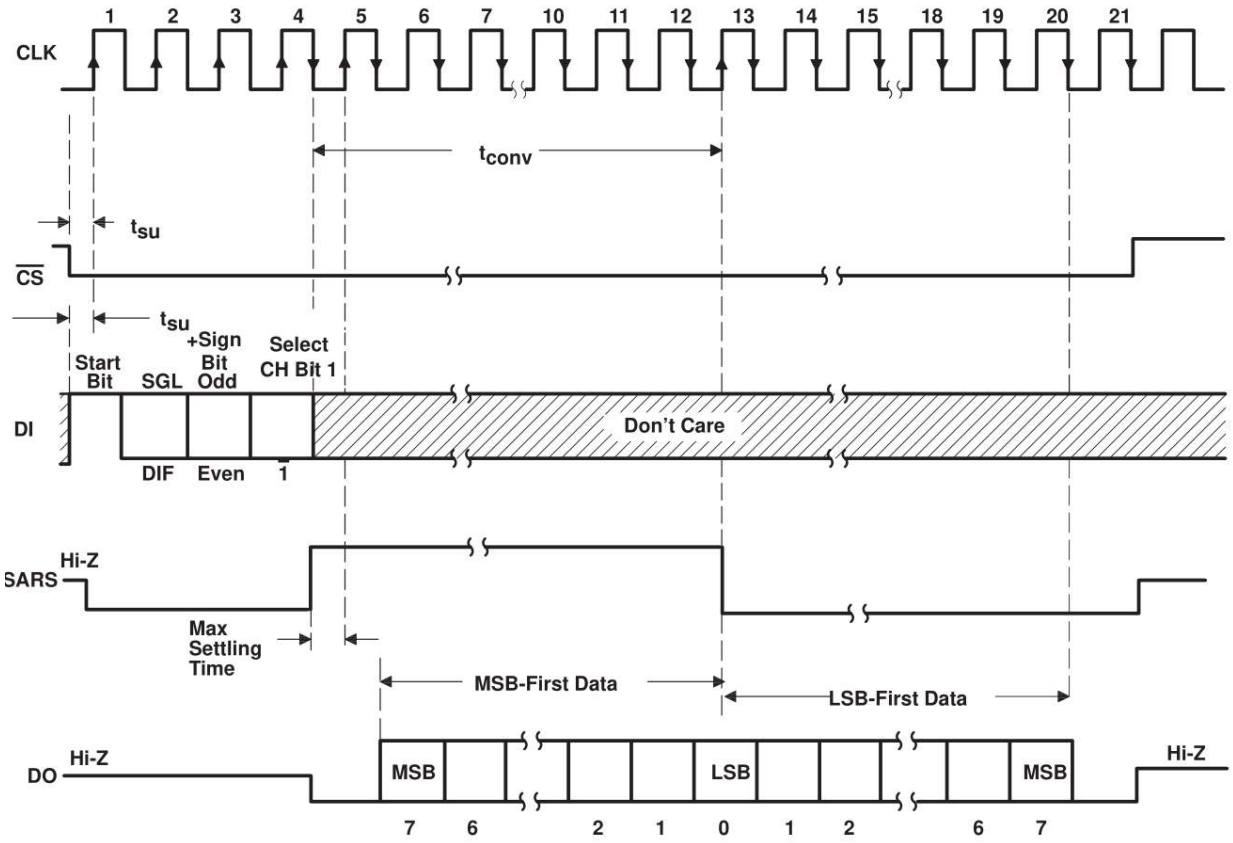


Sequence of Operation

A conversion is initiated by setting CS low, which enables all logic circuits. CS must be held low for the complete conversion process. A clock input is then received from the processor. On each low-to-high transition of the clock input, the data on DI is clocked into the multiplexer address shift register. The first logic high on the input is the start bit. A 3- to 4-bit assignment word follows the start bit. On each successive low-to-high transition of the clock input, the start bit and assignment word are shifted through the shift register. When the start bit is shifted into the start location of the multiplexer register, the input channel is selected and conversion starts. The SAR Statu output (SARS) goes high to indicate that a conversion is in progress, and DI to the multiplexer shift register is disabled the duration of the conversion.

An interval of one clock period is automatically inserted to allow the selected multiplexed channel to settle. The data output DO comes out of the high-impedance state and provides a leading low for this one clock period of multiplexer settling time. The SAR comparator compares successive outputs from the resistive ladder with the incoming analog signal. The comparator output indicates whether the analog input is greater than or less than the resistive ladder output. As the conversion proceeds, conversion data is simultaneously output from the DO output pin, with the most significant bit (MSB) first.

After eight clock periods, the conversion is complete and the SARS output goes low. Finally outputs the least-significant-bit-first data after the MSB-first data stream.



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

- ADC0831 series Datasheet

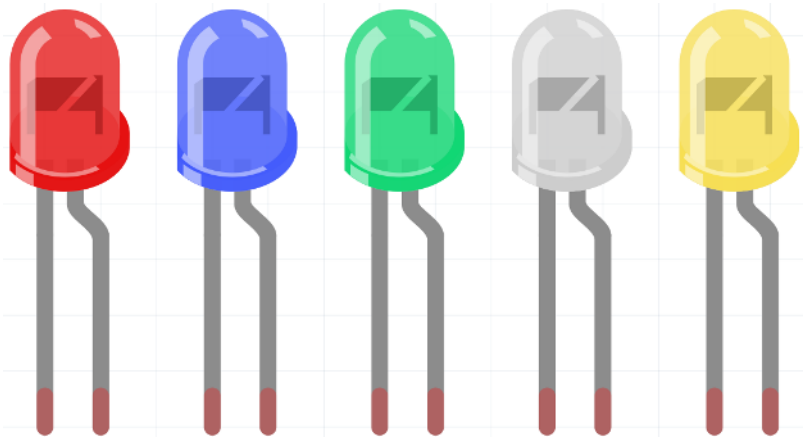
Example

- *2.1.7 Potentiometer* (C Project)
- *2.2.1 Photoresistor* (C Project)
- *2.2.2 Thermistor* (C Project)

- *3.1.4 Smart Fan* (C Project)
- *3.1.5 Battery Indicator* (C Project)
- *3.1.7 Overheat Monitor* (C Project)
- *2.1.7 Potentiometer* (Python Project)
- *2.2.1 Photoresistor* (Pyhton Project)
- *2.2.2 Thermistor* (Pyhton Project)
- *4.1.10 Smart Fan* (Pyhton Project)
- *4.1.11 Battery Indicator* (Pyhton Project)
- *4.1.13 Overheat Monitor* (Pyhton Project)

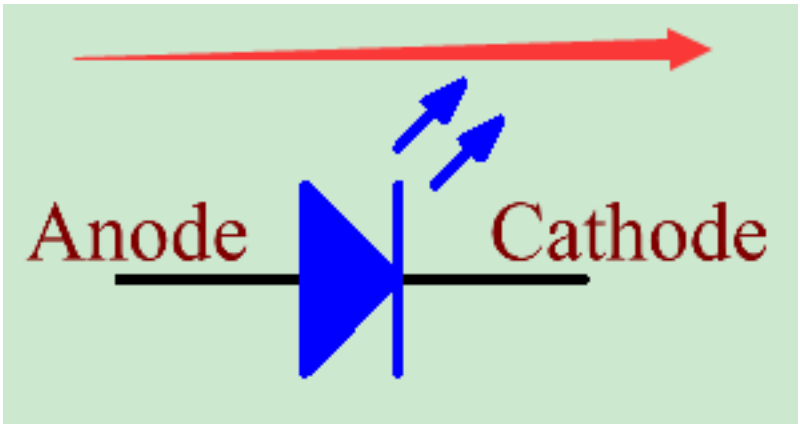
Display

2.2.9 LED



Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode which is usually known as light-emitting diode (LED).

Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative. Thus the LED will light up.



An LED has two pins. The longer one is the anode, and shorter one, the cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$$R = (Vsupply - VD)/I$$

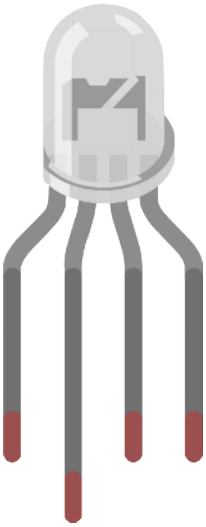
R stands for the resistance value of the current limiting resistor, **Vsupply** for voltage supply, **VD** for voltage drop and **I** for the working current of the LED.

Here is the detailed introduction for the LED: LED - Wikipedia.

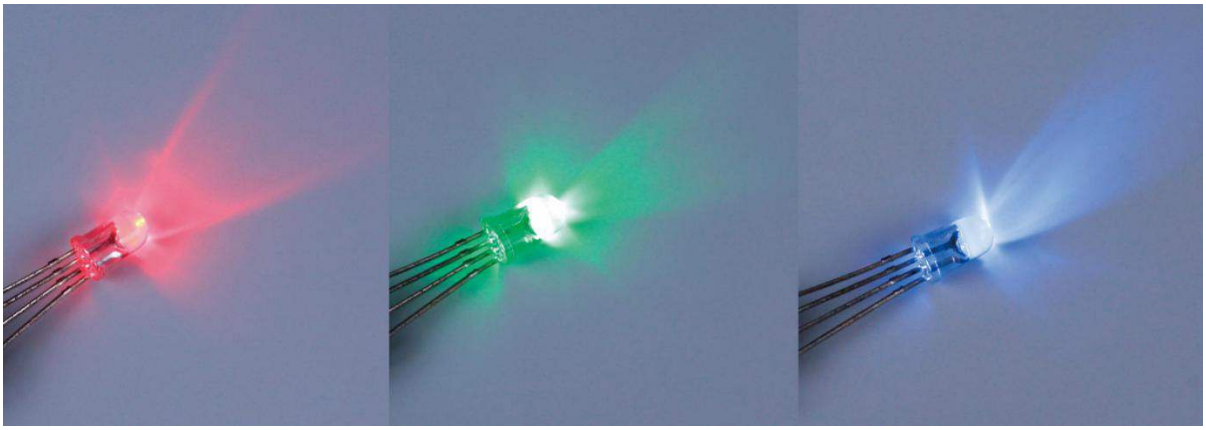
Example

- *1.1.1 Blinking LED* (C Project)
- *3.1.6 Traffic Light* (C Project)
- *1.1.1 Blinking LED* (Python Project)
- *4.1.12 Traffic Light* (Python Project)
- *1.1 Wand* (Scratch Project)

2.2.10 RGB LED

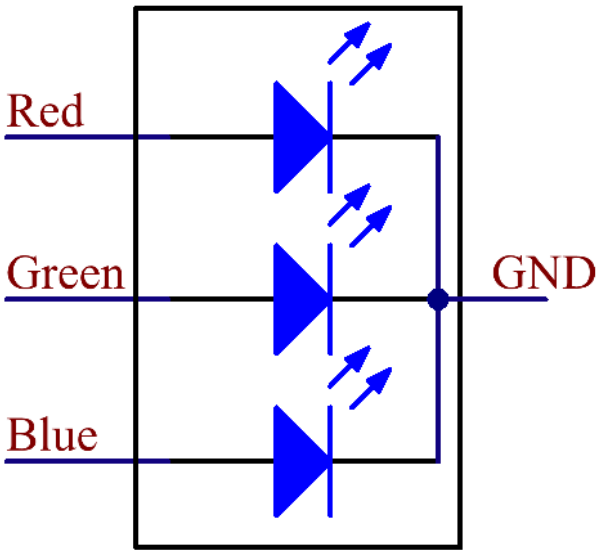


RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216 different colors.

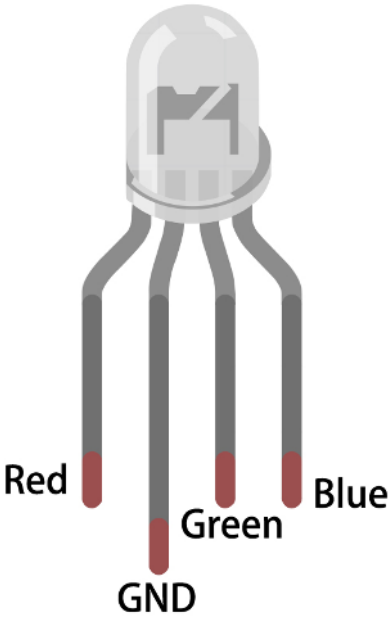


RGB LEDs can be categorized into common anode and common cathode ones. In this kit, the latter is used. The **common cathode**, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

Its circuit symbol is shown as figure.



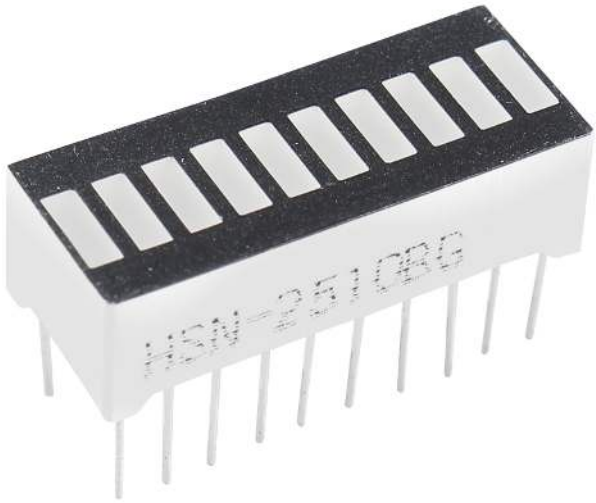
An RGB LED has 4 pins: the longest one is GND; the others are Red, Green and Blue. Touch its plastic shell and you will find a cut. The pin closest to the cut is the first pin, marked as Red, then GND, Green and Blue in turn.



Example

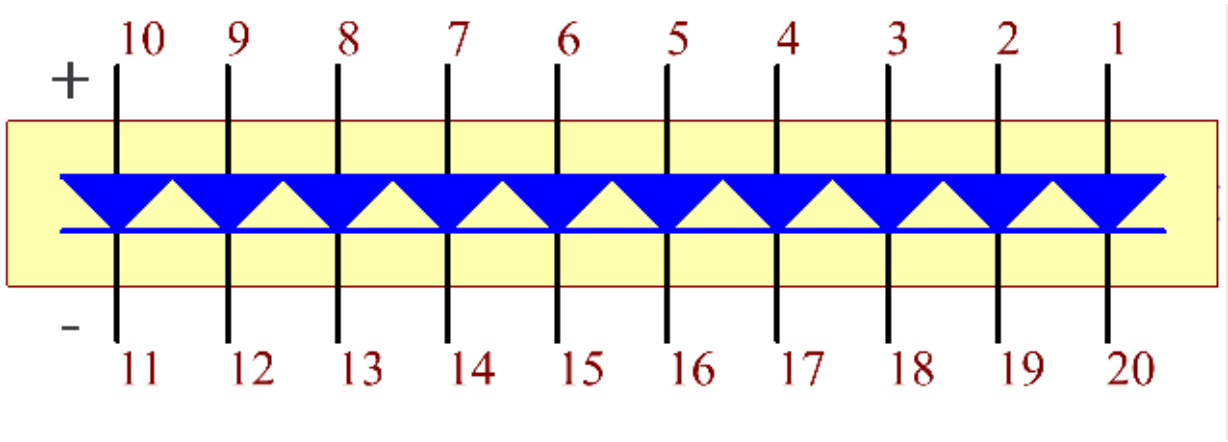
- *1.1.2 RGB LED* (C Project)
- *1.1.2 RGB LED* (Python Project)
- *1.2 Colorful Balls* (Scratch Project)

2.2.11 LED Bar Graph



LED Bar Graph is an LED array, which is used to connect with electronic circuit or microcontroller. It's easy to connect LED bar graph with the circuit like as connecting 10 individual LEDs with 10 output pins. Generally we can use the LED bar graph as a Battery level Indicator, Audio equipments, and Industrial Control panels. There are many other applications of LED bar graphs.

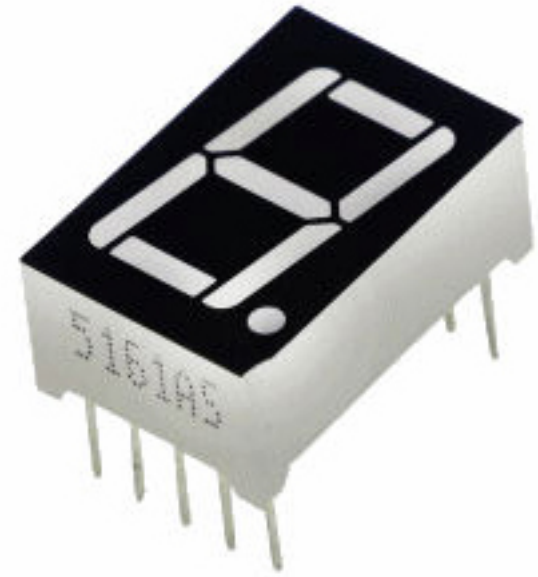
The following is the internal schematic diagram of LED Bar Graph. Generally speaking, the side with the label is the anode and the other side is the cathode.



Example

- *1.1.3 LED Bar Graph* (C Project)
- *3.1.5 Battery Indicator* (C Project)
- *1.1.3 LED Bar Graph* (Python Project)
- *4.1.11 Battery Indicator* (Python Project)
- *1.12 Water Lamp* (Scratch Project)

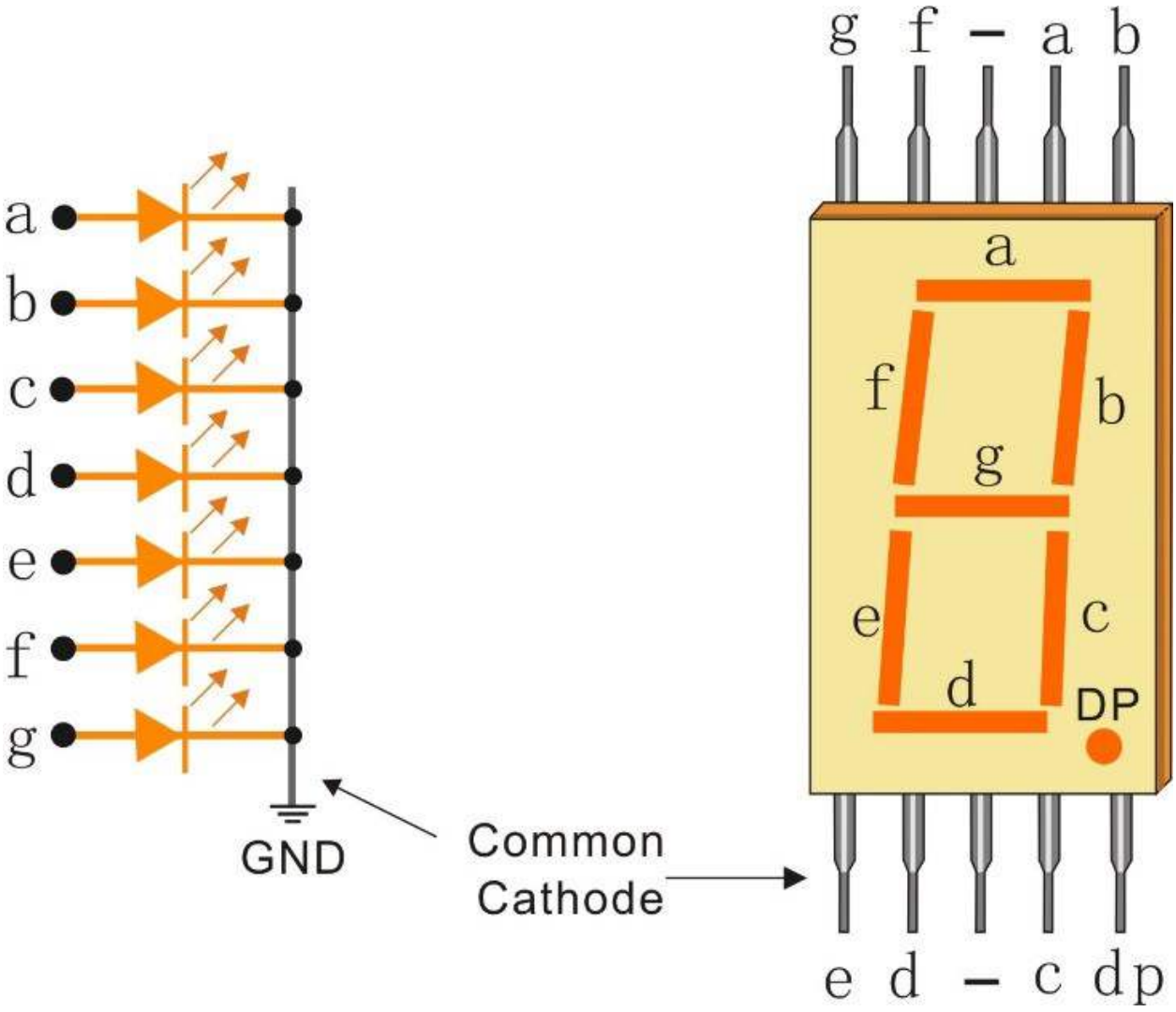
2.2.12 7-segment Display



A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment - when energized, one segment forms part of a numeral to be displayed.

There are two types of pin connection: Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected.

In this kit, we use the Common Cathode 7-segment display, here is the electronic symbol.



Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from “a” through to “g” representing each individual LED. The other LED pins are connected together forming a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

Display Codes

To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

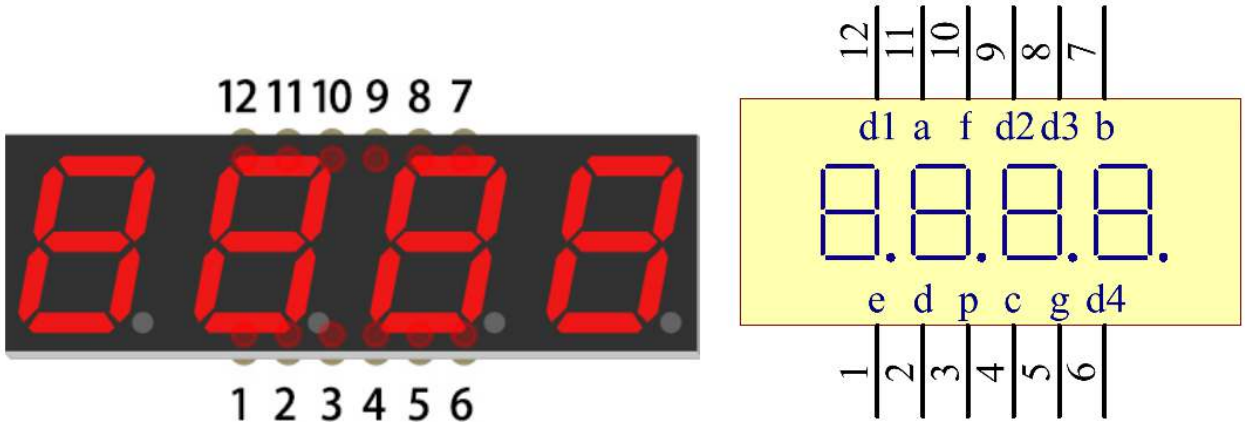
Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39
3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d			
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

Example

- *1.1.4 7-segment Display* (C Project)
- *1.1.4 7-segment Display* (Python Project)

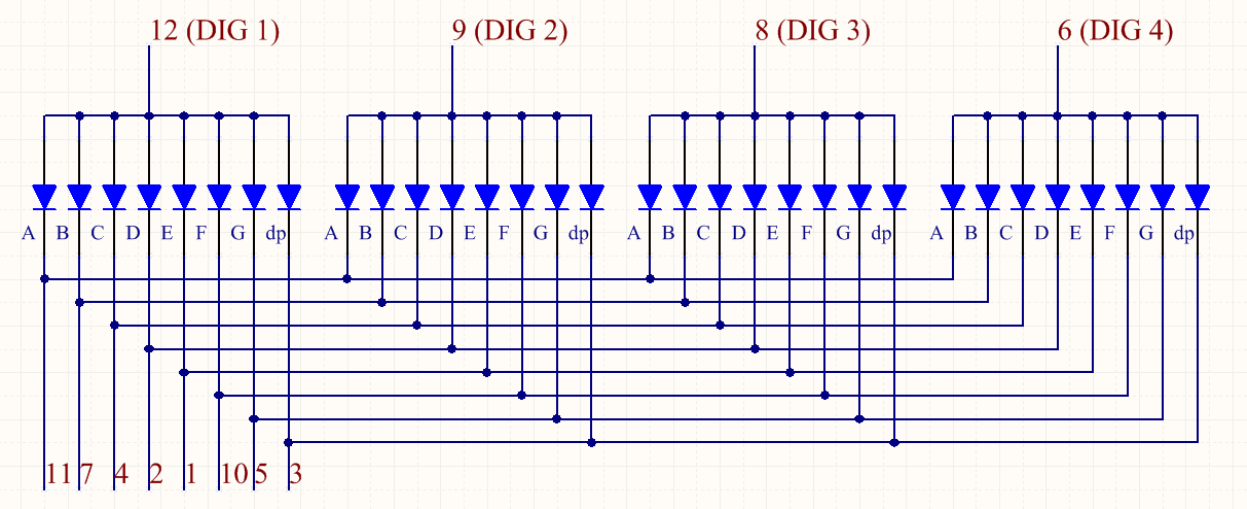
2.2.13 4-Digit 7-Segment Display

4-Digit 7-segment display consists of four 7- segment displays working together.



The 4-digtal 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when "1234" is displayed on the display, "1" is displayed on the first 7-segment, and "234" is not displayed. After a period of time, the second 7-segment shows "2", the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.



Display Codes

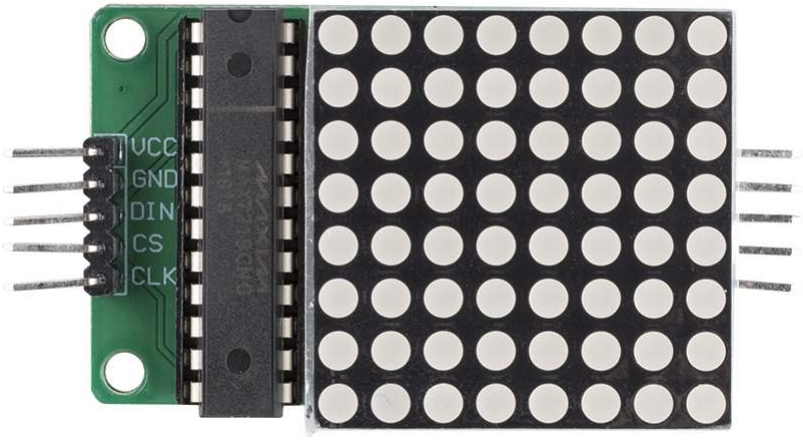
To help you get to know how 7-segment displays(Common Anode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 11000000 means that DP and G are set to 1, while others are set to 0. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

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			

Example

- *1.1.5 4-Digit 7-Segment Display* (C Project)
- *3.1.1 Counting Device* (C Project)
- *3.1.6 Traffic Light* (C Project)
- *3.1.12 GAME - 10 Second* (C Project)
- *1.1.5 4-Digit 7-Segment Display* (Python Project)
- *4.1.3 Speech Clock* (Pyhton Project)
- *4.1.7 Counting Device* (Pyhton Project)
- *4.1.12 Traffic Light* (Pyhton Project)
- *4.1.18 GAME - 10 Second* (Pyhton Project)

2.2.14 LED Matrix Module



This is a common cathode 8x8 dot matrix module driven by MAX7219, the module operating voltage is 5V, the size is 50mmx32mmx15mm, the left side is input port, the right side is output port, support multiple modules cascade.

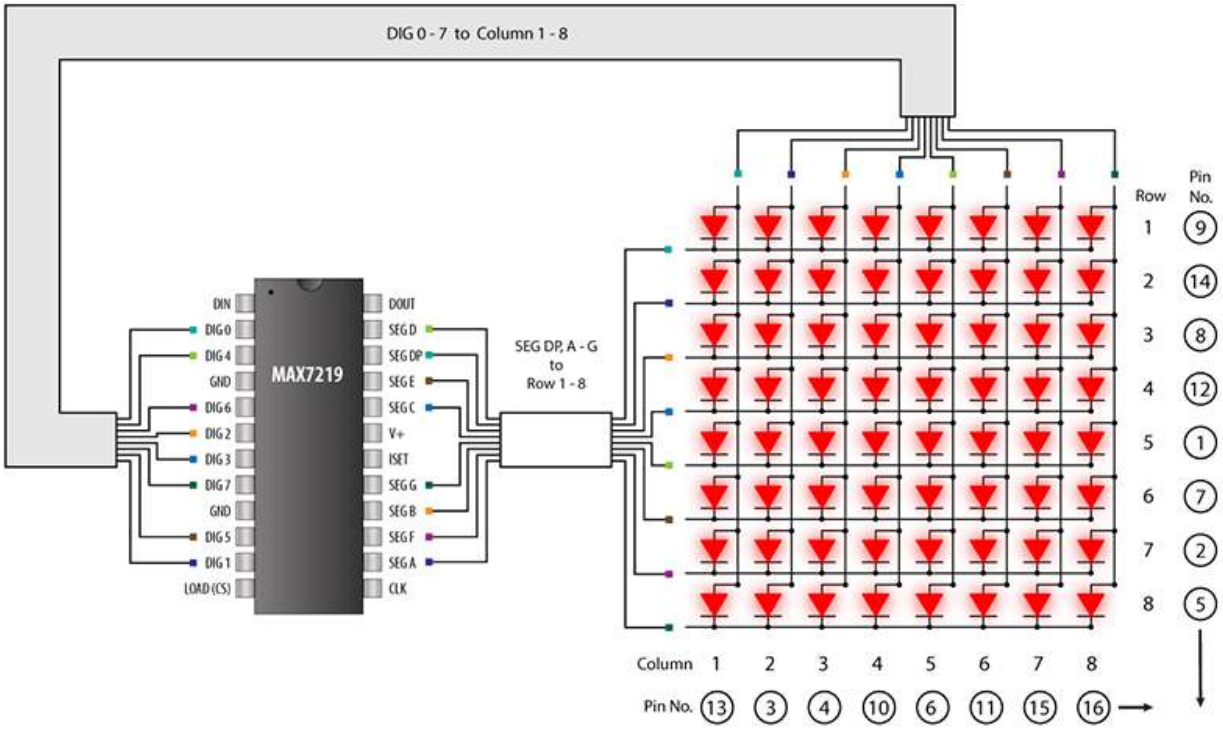
- **VCC**: Positive Supply Voltage. Connect to +5V.
- **GND**: Ground (both GND pins must be connected)
- **DIN**: Serial-Data Input. Data is loaded into the internal 16-bit shift register on CLK's rising edge.
- **CS**: Chip-Select Input. Serial data is loaded into the shift register while CS is low. The last 16 bits of serial data are latched on CS's rising edge.
- **CLK**: Serial-Clock Input. 10MHz maximum rate. On CLK's rising edge, data is shifted into the internal shift register. On CLK's falling edge, data is clocked out of DOUT. On the MAX7221, the CLK input is active only while CS is low.

MAX7219

The MAX7219 is a compact, serial input/output common-cathode display drivers that interface microprocessors (µPs) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs. Included on-chip are a BCD code-B decoder, multiplex scan circuitry, segment and digit drivers, and an 8x8 static RAM that stores each digit.

Only one external resistor is required to set the segment current for all LEDs. The MAX7221 is compatible with SPI™, QSPI™, and MICROWIRE™, and has slewrate-limited segment drivers to reduce EMI.

A convenient 4-wire serial interface connects to all common µPs. Individual digits may be addressed and updated without rewriting the entire display. The MAX7219/MAX7221 also allow the user to select codeB decoding or no-decode for each digit.



- MAX7219 Datasheet

Example

- *1.1.6 LED Dot Matrix Module* (C Project)
- *3.1.12 GAME - 10 Second* (C Project)
- *1.1.6 LED Dot Matrix* (Python Project)
- *4.1.19 AttendanceSystem* (Python Project)

2.2.15 I2C LCD1602

- **GND**: Ground
- **VCC**: Voltage supply, 5V.
- **SDA**: Serial data line. Connect to VCC through a pullup resistor.
- **SCL**: Serial clock line. Connect to VCC through a pullup resistor.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller.

Therefore, LCD1602 with an I2C module is developed to solve the problem. The I2C module has a built-in PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display.

- PCF8574 Datasheet

Backlight can be enabled by jumper cap, unplugg the jumper cap to disable the backlight. The blue potentiometer on the back is used to adjust the contrast (the clarity of the displayed text), which is increased in the clockwise direction and decreased in the counterclockwise direction.

The default address is basically 0x27, in a few cases it may be 0x3F, which can be known by test (*I2C Configuration*).

Taking the default address of 0x27 as an example, the device address can be modified by shorting the A0/A1/A2 pads; in the default state, A0/A1/A2 is 1, and if the pad is shorted, A0/A1/A2 is 0.

Example

- *1.1.7 I2C LCD1602* (C Project)
- *3.1.3 Reversing Alarm* (C Project)
- *3.1.7 Overheat Monitor* (C Project)
- *3.1.8 Password Lock* (C Project)
- *3.1.11 GAME– Guess Number* (C Project)
- *1.1.7 I2C LCD1602* (Python Project)

- *4.1.9 Reversing Alarm* (Python Project)
- *4.1.13 Overheat Monitor* (Python Project)
- *4.1.14 Password Lock* (Python Project)
- *4.1.17 GAME– Guess Number* (Python Project)

Sound

2.2.16 Buzzer

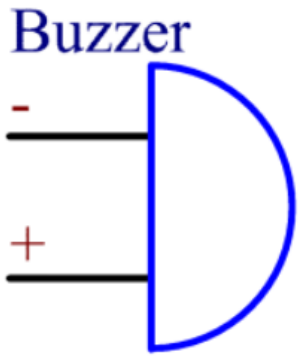
As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices.

Buzzers can be categorized as active and passive ones (see the following picture). Turn the buzzer so that its pins are facing up, and the buzzer with a green circuit board is a passive buzzer, while the one enclosed with a black tape is an active one.

The difference between an active buzzer and a passive buzzer:

An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not beep if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

The following is the electrical symbol of a buzzer. It has two pins with positive and negative poles. With a + in the surface represents the anode and the other is the cathode.



You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

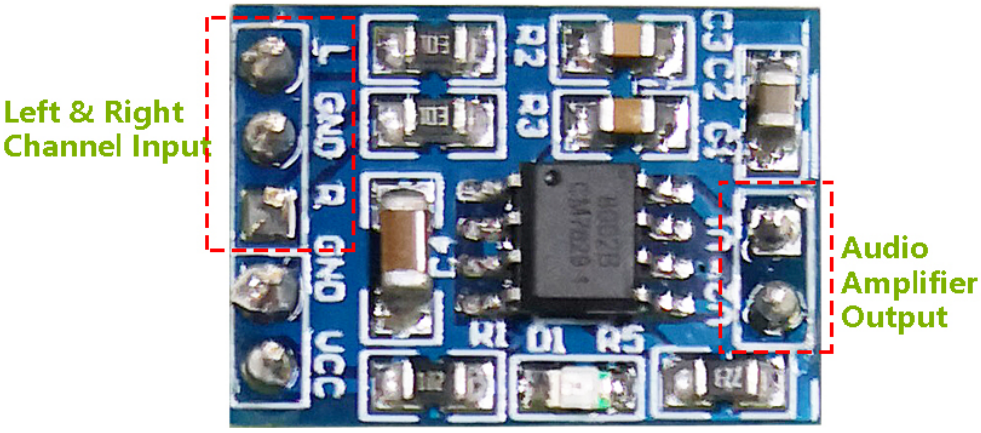
Buzzer - Wikipedia

Example

- *1.2.1 Active Buzzer* (C Project)
- *1.2.2 Passive Buzzer* (C Project)
- *1.2.1 Active Buzzer* (Python Project)
- *1.2.2 Passive Buzzer* (Python Project)
- *1.13 Doorbell* (Scratch Project)
- *1.14 123 Wooden Man* (Scratch Project)

2.2.17 Audio Module and Speaker

Audio Amplifier Module



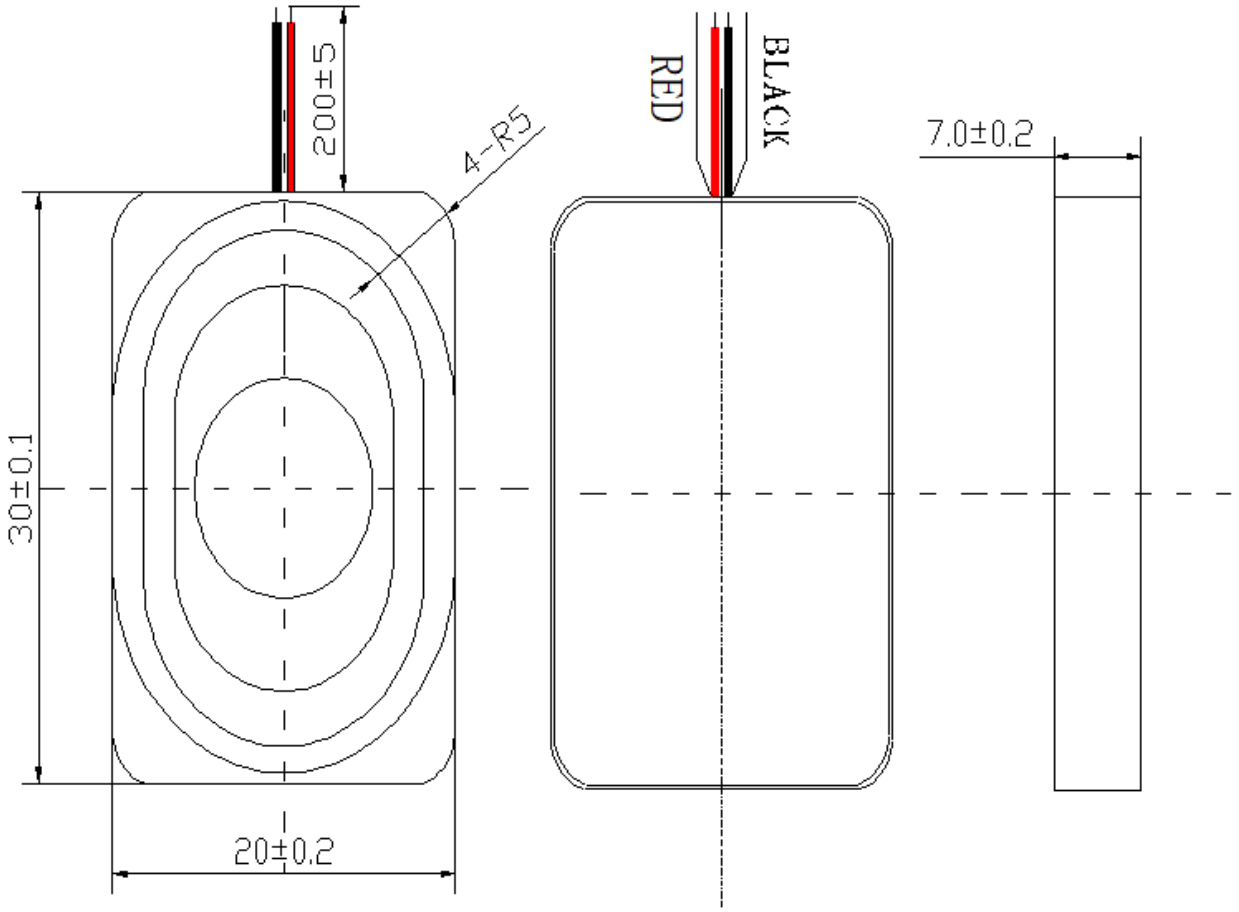
Audio Amplifier Module contains a HXJ8002 audio power amplifier chip. This chip is a power amplifier with low power supply, that can provide 3W average audio power for a 3Ω BTL load with low harmonic distortion (under 10% threshold distortion at 1KHz) from a 5V DC power supply. This chip can amplify audio signals without any coupling capacitors or bootstrap capacitors.

The module can be supplied by a 2.0V up to 5.5V DC with 10mA operating current (0.6uA for typical standby current) power source and produce a powerful amplified sound into a 3, 4, or 8 impedance speaker. This module has an improved pop and clicks circuitry for reducing significantly the transition nose at the powering on and off moment. Tiny size besides high efficiency and low power supplying make it applicable in widely portable and battery-powered projects and microcontrollers.

- **IC**: HXJ8002
- **Input Voltage**: 2V ~ 5.5V
- **Standby Mode Current**: 0.6uA (typical value)
- **Output Power**: 3W (3Ω load) , 2.5W (4Ω load) , 1.5W (8Ω load)
- **Output Speaker Impedance**: 3Ω, 4Ω, 8Ω
- **Size**: 19.8mm x 14.2mm

Speaker

- **Size**: 20x30x7mm
- **Impedance**8ohm
- **Rate Input Power**: 1.5W
- **Max Input Power**: 2.0W
- **Wire Length**: 10cm



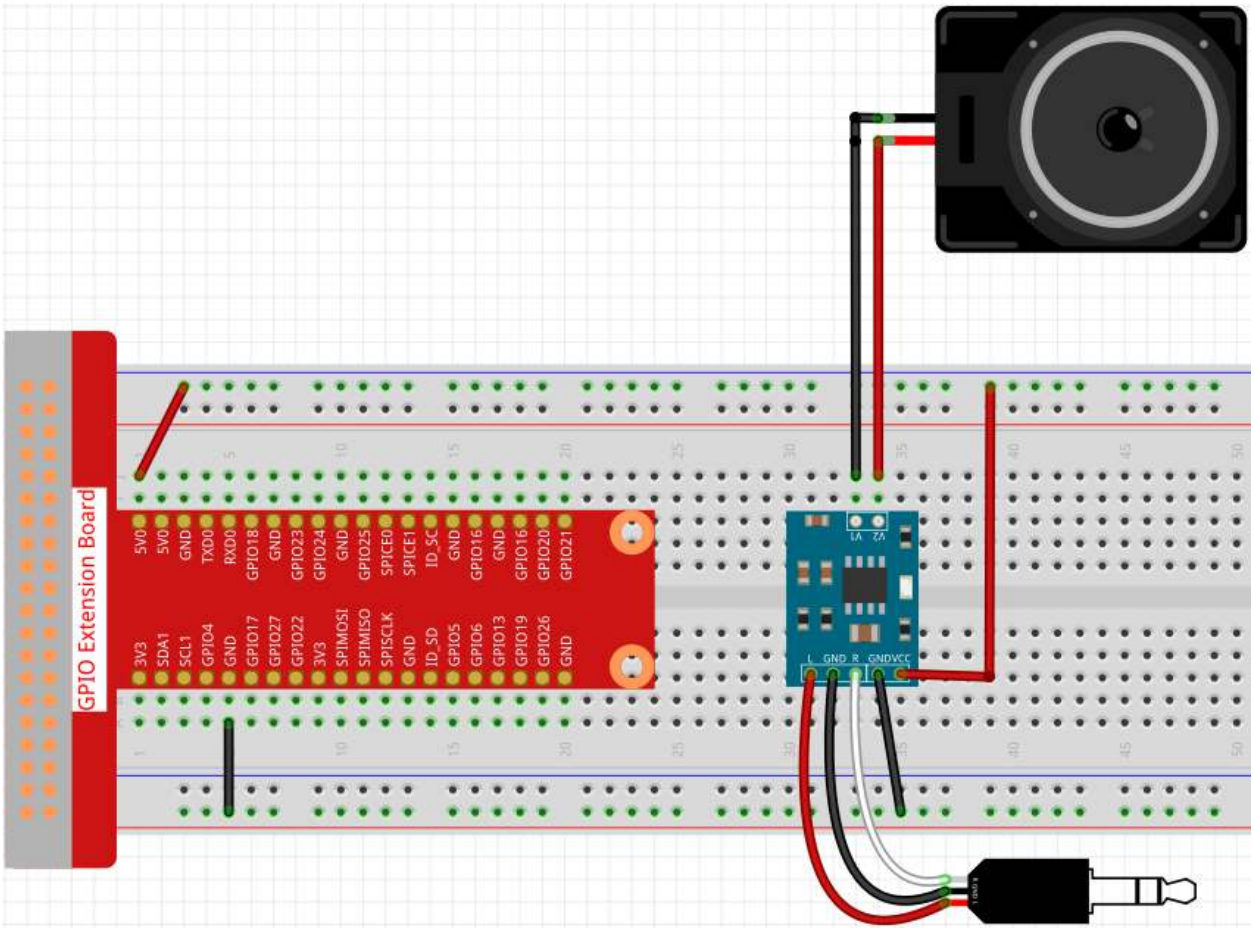
The size chart is as follows

- `2030 Speaker Datasheet`

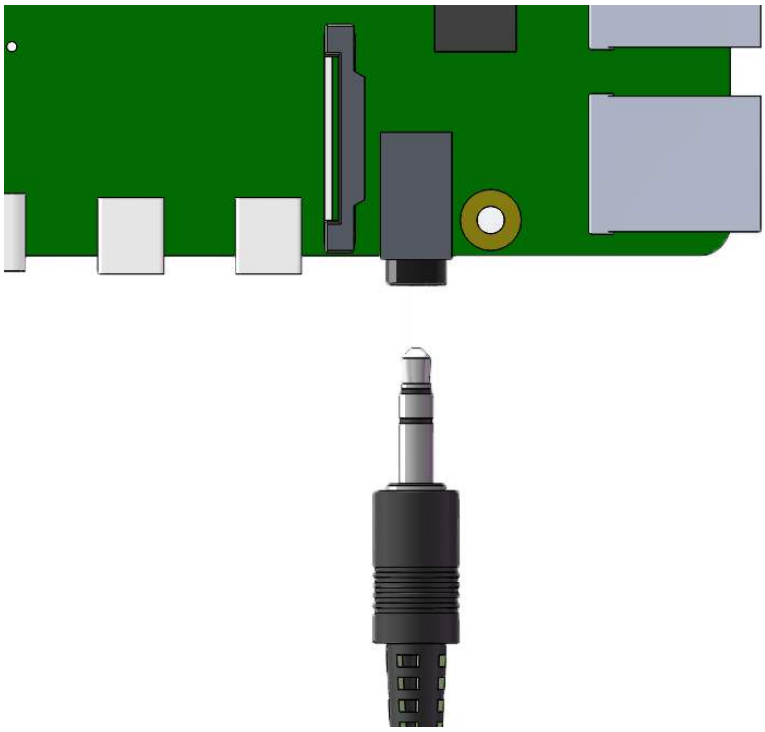
Audio Cable

This is a 3.5mm male audio cable with a total length of 43cm. it has 3 connectors, red for the left channel, white for the right channel, and GND in the middle.

Circuit



After building the circuit according to the above diagram, then plug the audio cable into the Raspberry Pi's 3.5mm audio jack.



If your speaker have no sound, it may be because the Raspberry Pi has selected the wrong audio output (The default is HDMI), you need to *Change Audio Output* to **Headphones**.

If you feel that the volume of the speakers is too low, you can *Adjust Volume*.

Example

- *3.1.3 Audio Module* (Python Project)
- *3.1.4 Text-to-speech* (Python Project)
- *4.1.2 Music Player* (Python Project)
- *4.1.3 Speech Clock* (Python Project)
- *4.1.5 Intelligent Visual Doorbell* (Python Project)
- *1.8 Service Bell* (Scratch Project)
- *1.9 Drumming* (Scratch Project)
- *1.10 Drumming in the Air* (Scratch Project)

Driver

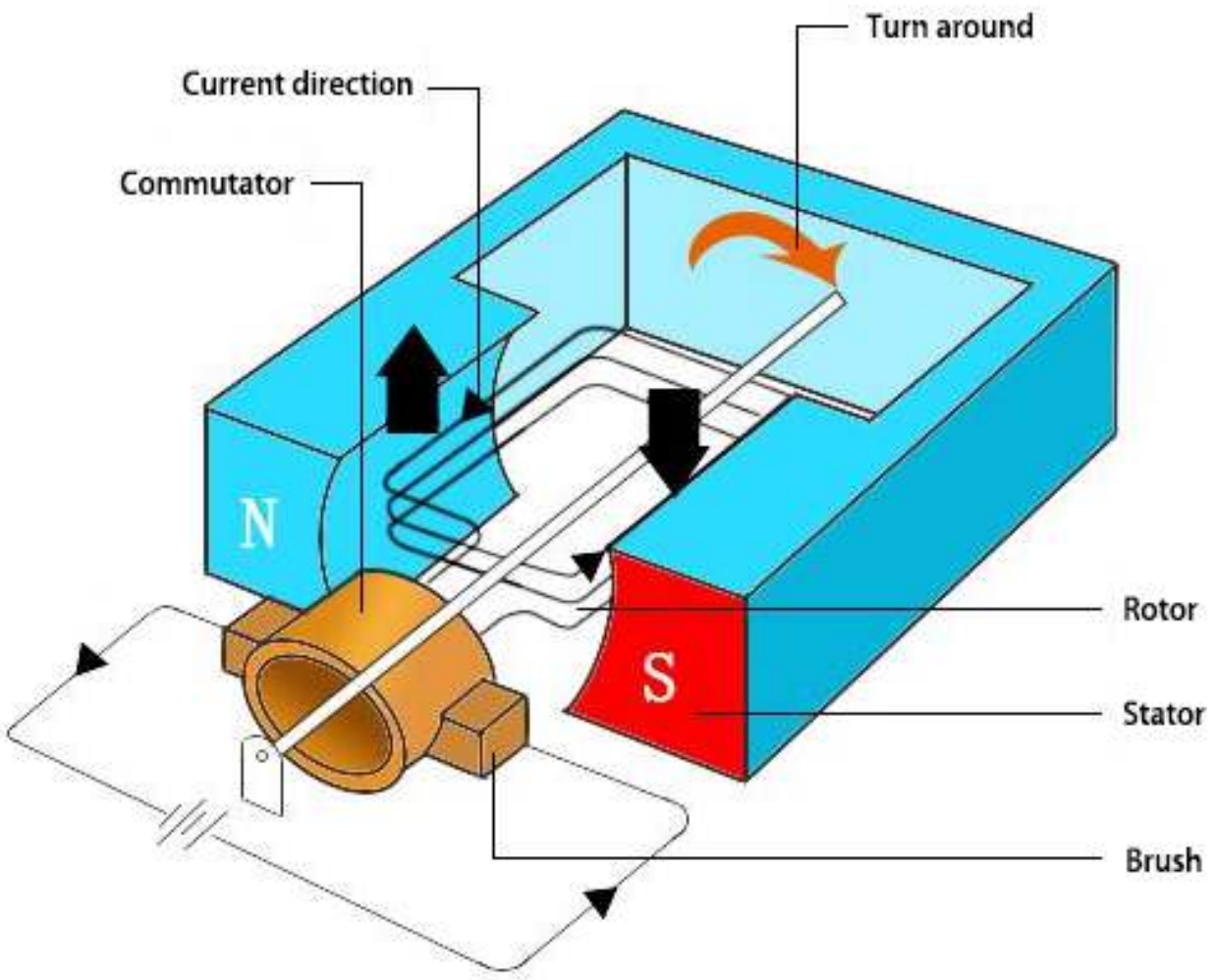
2.2.18 DC Motor

This is a 3V DC motor. When you give a high level and a low level to each of the 2 terminals, it will rotate.

- **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

Direct current (DC) motor is a continuous actuator that converts electrical energy into mechanical energy. DC motors make rotary pumps, fans, compressors, impellers, and other devices work by producing continuous angular rotation.

A DC motor consists of two parts, the fixed part of the motor called the **stator** and the internal part of the motor called the **rotor** (or **armature** of a DC motor) that rotates to produce motion. The key to generating motion is to position the armature within the magnetic field of the permanent magnet (whose field extends from the north pole to the south pole). The interaction of the magnetic field and the moving charged particles (the current-carrying wire generates the magnetic field) produces the torque that rotates the armature.



Current flows from the positive terminal of the battery through the circuit, through the copper brushes to the commutator, and then to the armature. But because of the two gaps in the commutator, this flow reverses halfway through each complete rotation. This continuous reversal essentially converts the DC power from the battery to AC, allowing the armature to experience torque in the right direction at the right time to maintain rotation.

- DC Motor - MagLab

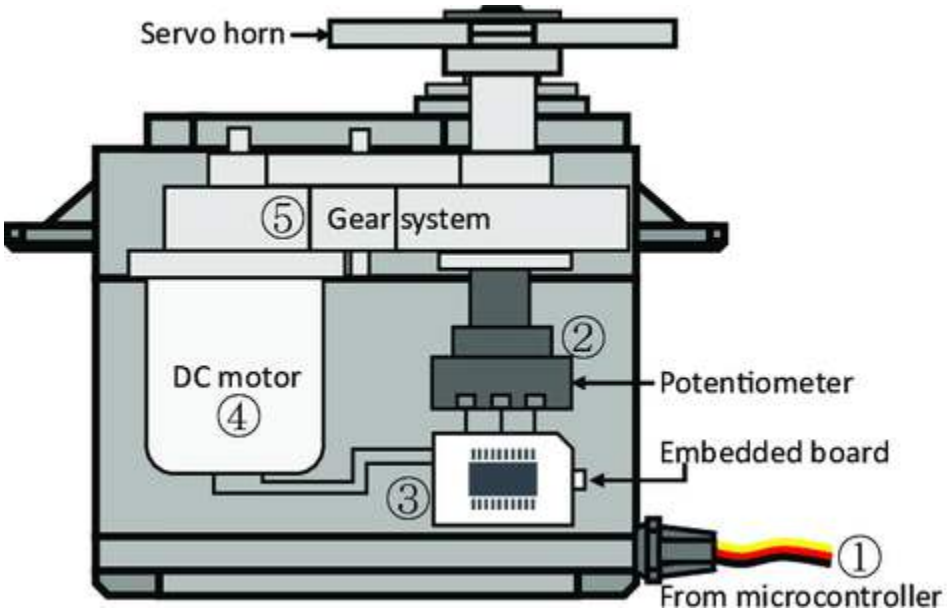
Example

- *1.3.1 Motor* (C Project)
- *3.1.4 Smart Fan* (C Project)
- *1.3.1 Motor* (Python Project)
- *4.1.10 Smart Fan* (Python Project)
- *1.17 Rotating fan* (Scratch Project)

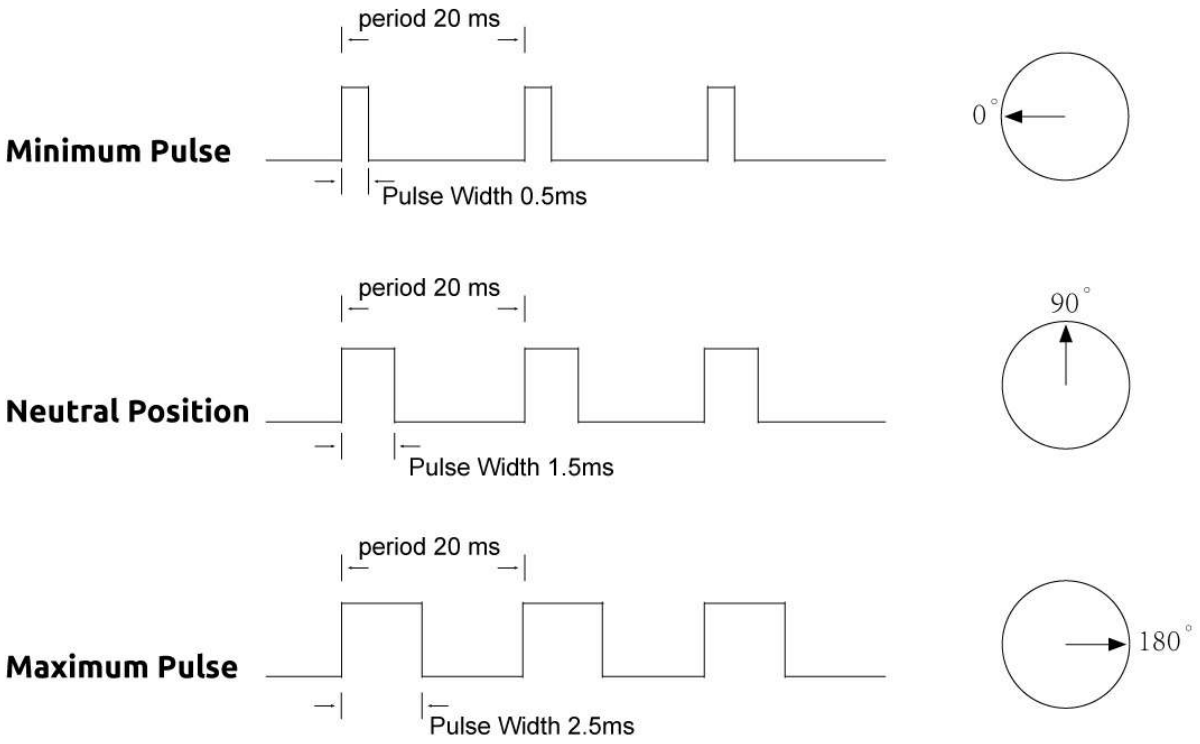
2.2.19 Servo

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.



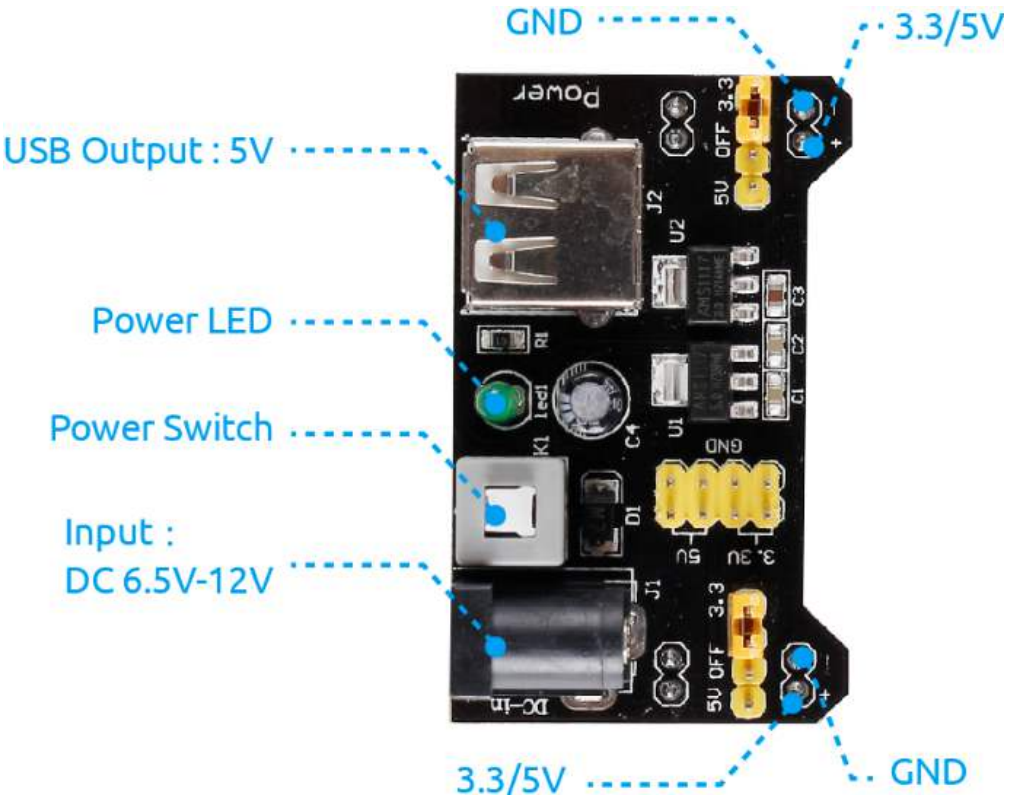
Example

- *1.3.2 Servo* (C Project)
- *3.1.2 Welcome* (C Project)
- *1.3.2 Servo* (Python Project)
- *4.1.8 Welcome* (Python Project)

2.2.20 Power Supply Module

When we need a large current to drive a component, which will severely interfere with the normal work of Raspberry Pi. Therefore, we separately supply power for the component by this module to make it run safely and steadily.

You can just plug it in the breadboard to supply power. It provides a voltage of 3.3V and 5V, and you can connect either via a jumper cap included.



Features and specifications

- Input voltage: 6.5 - 12V
- Two Independent Channel
- Output voltage: 5V, 3.3V (adjustable via jumpers. 0V, 3.3V, and 5V configuration)
- Output current: Maximum output current 700mA
- Onboard berg male header for GND, 5V, 3.3V output
- ON-OFF Switch available.
- USB (Type-A) input available.
- DC Barrel Jack input available.
- Onboard power LED
- Dimension: 53mm x 33mm (L x W)

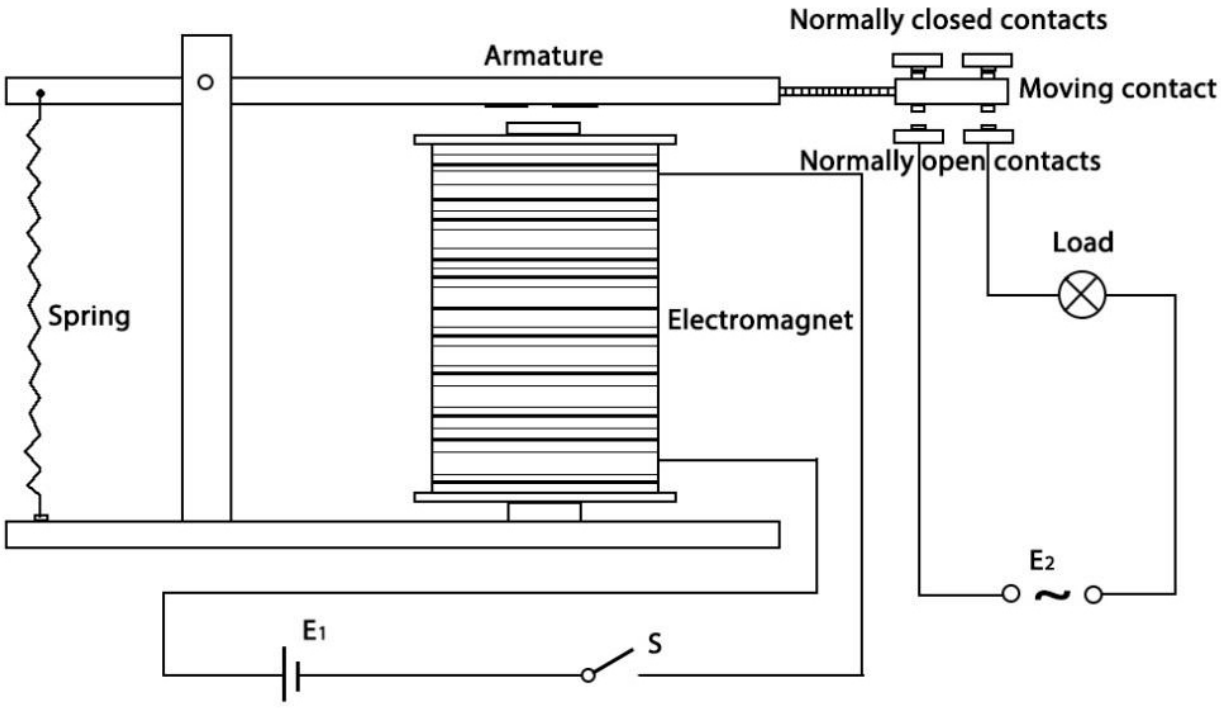
Example

- *1.3.1 Motor* (C Project)
- *3.1.4 Smart Fan* (C Project)
- *1.3.1 Motor* (Python Project)
- *4.1.10 Smart Fan* (Python Project)
- *1.17 Rotating fan* (Scratch Project)

2.2.21 Relay

As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:



Electromagnet - It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

Armature - The movable magnetic strip is known as armature. When current flows through them, the coil is it energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

Spring - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit

cannot be completed.

Set of electrical **contacts** - There are two contact points:

- Normally open - connected when the relay is activated, and disconnected when it is inactive.
- Normally close - not connected when the relay is activated, and connected when it is inactive.

Molded frame - Relays are covered with plastic for protection.

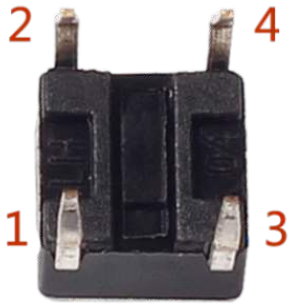
The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

Example

- *1.3.3 Relay* (C Project)
- *1.3.3 Relay* (Python Project)

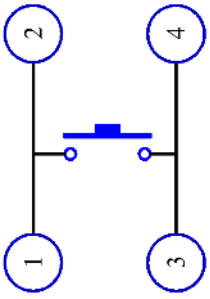
Controller

2.2.22 Button

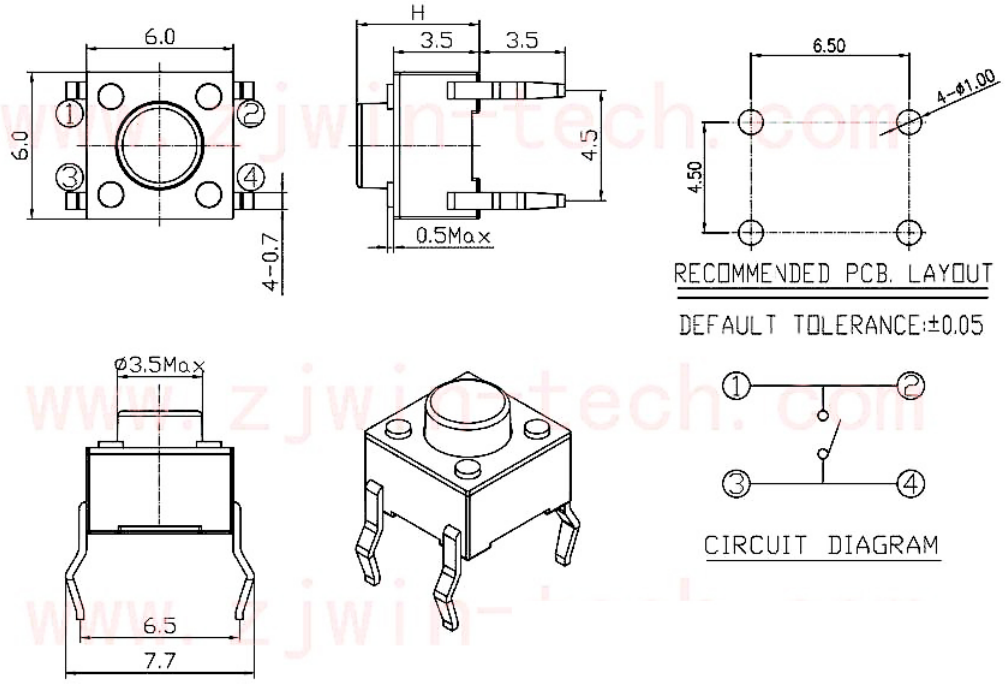


Buttons are a common component used to control electronic devices. They are usually used as switches to connect or break circuits. Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures. Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.

The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.



Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.

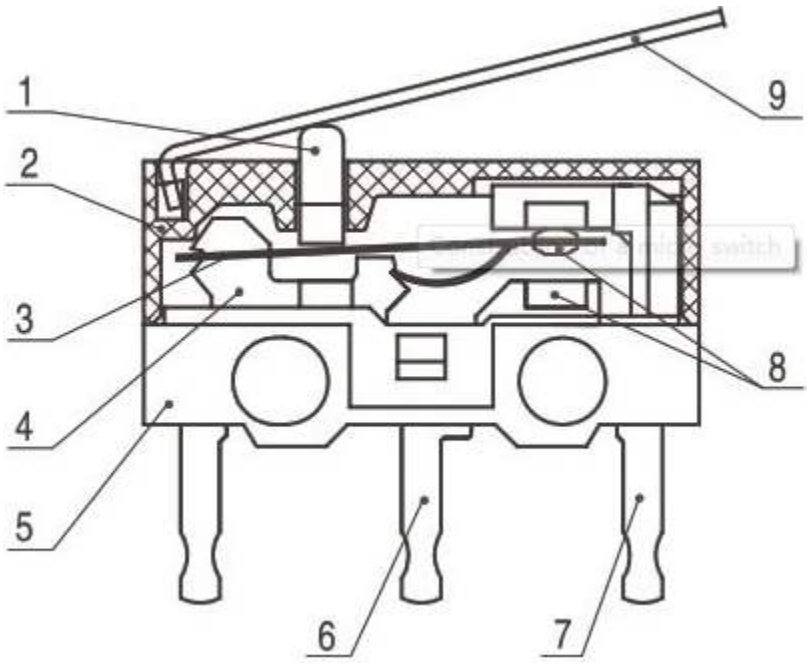


Example

- *2.1.1 Button* (C Project)
- *3.1.4 Smart Fan* (C Project)
- *3.1.13 GAME– NotNot* (C Project)
- *2.1.1 Button* (Python Project)
- *4.1.2 Music Player* (Python Project)
- *4.1.10 Smart Fan* (Python Project)
- *1.4 Hare* (Scratch Project)

2.2.23 Micro Switch

The construction of a micro switch is really simple. The main parts of the switch are:



- 1.Plunger (Actuator)
- 2.Cover
- 3.Moving piece
- 4.Support
- 5.Case
- 6.NO terminal: normally open
- 7.NC terminal: normally closed
- 8.Contact
- 9.Moving arm

After a micro switch makes physical contact with an object, its contacts change position. The basic working principle is as follows.

When the plunger is in the released or rest position.

- The normally closed circuit can carry current.
- The normally open circuit is electrically insulated.

When the plunger is depressed or switched.

- The normally closed circuit is open.
- The normally open circuit is closed.

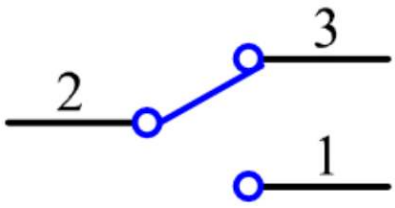
Example

- *2.1.2 Micro Switch* (C Project)
- *2.1.2 Micro Switch* (Python Project)
- *1.8 Service Bell* (Scratch Project)

2.2.24 Slide Switch

A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The slide switch is commonly used in low-voltage circuit. It has the features of flexibility and stability, and applies in electric instruments and electric toys widely. How it works: Set the middle pin as the fixed one. When you pull the slide to the left, the two pins on the left are connected; when you pull it to the right, the two pins on the right are connected. Thus, it works as a switch connecting or disconnecting circuits. See the figure below:

The circuit symbol of the slide switch is shown as below. The pin2 in the figure refers to the middle pin.

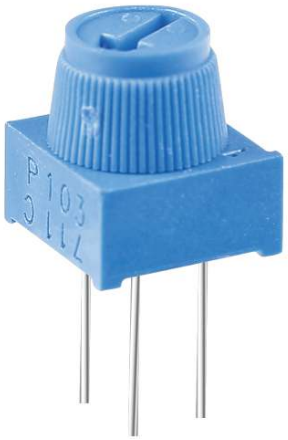


Example

- *2.1.4 Slide Switch* (C Project)
- *3.1.9 Alarm Bell* (C Project)

- *2.1.4 Slide Switch* (Python Project)
- *4.1.15 Alarm Bell* (Python Project)
- *1.15 Inflating the Balloon* (Scratch Project)

2.2.25 Potentiometer

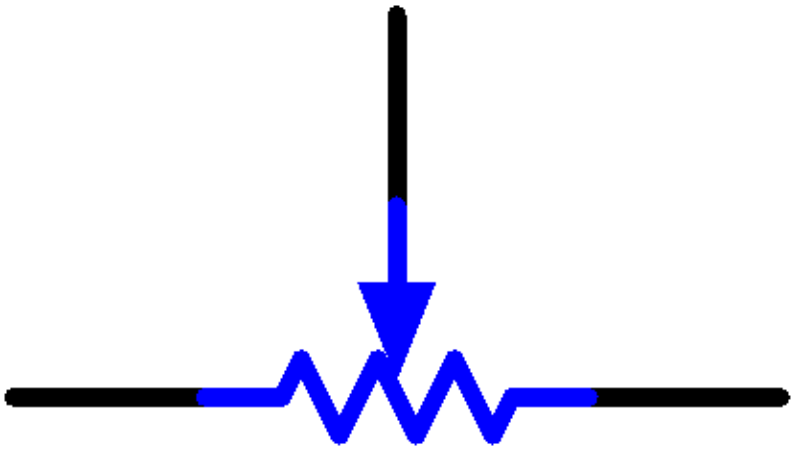


Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

Here is the circuit symbol of potentiometer.



The functions of the potentiometer in the circuit are as follows:

1. Serving as a voltage divider

Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

2. Serving as a rheostat

   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

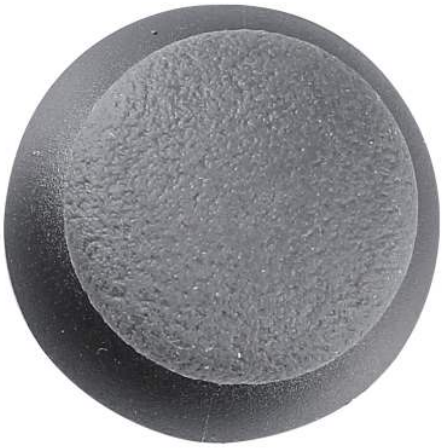
   When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

If you want to know more about potentiometer, refer to: Potentiometer - Wikipedia

Example

- *2.1.7 Potentiometer* (C Project)
- *2.1.7 Potentiometer* (Python Project)

2.2.26 Joystick Module

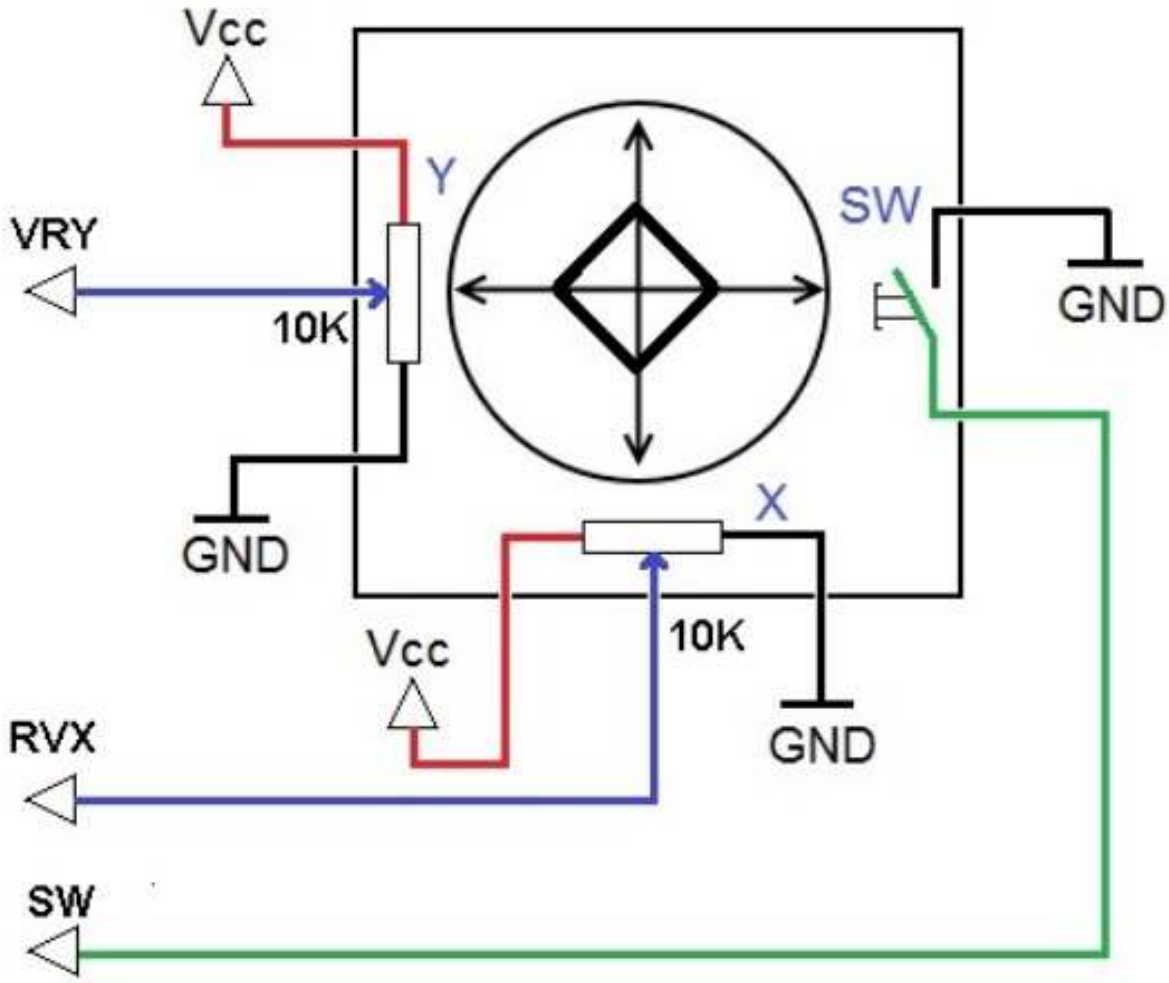


The basic idea of a joystick is to translate the movement of a stick into electronic information that a computer can process.

In order to communicate a full range of motion to the computer, a joystick needs to measure the stick's position on two axes – the X-axis (left to right) and the Y-axis (up and down). Just as in basic geometry, the X-Y coordinates pinpoint the stick's position exactly.

To determine the location of the stick, the joystick control system simply monitors the position of each shaft. The conventional analog joystick design does this with two potentiometers, or variable resistors.

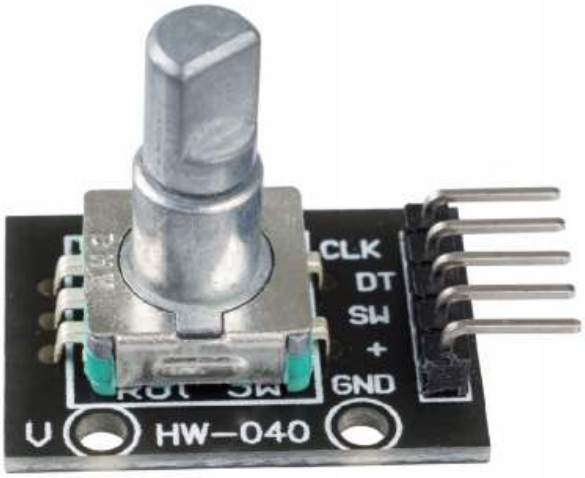
The joystick also has a digital input that is actuated when the joystick is pressed down.



Example

- *2.1.9 Joystick* (C Project)
- *3.1.7 Overheat Monitor* (C Project)
- *2.1.9 Joystick* (Python Project)
- *4.1.13 Overheat Monitor* (Python Project)

2.2.27 Rotary Encoder Module

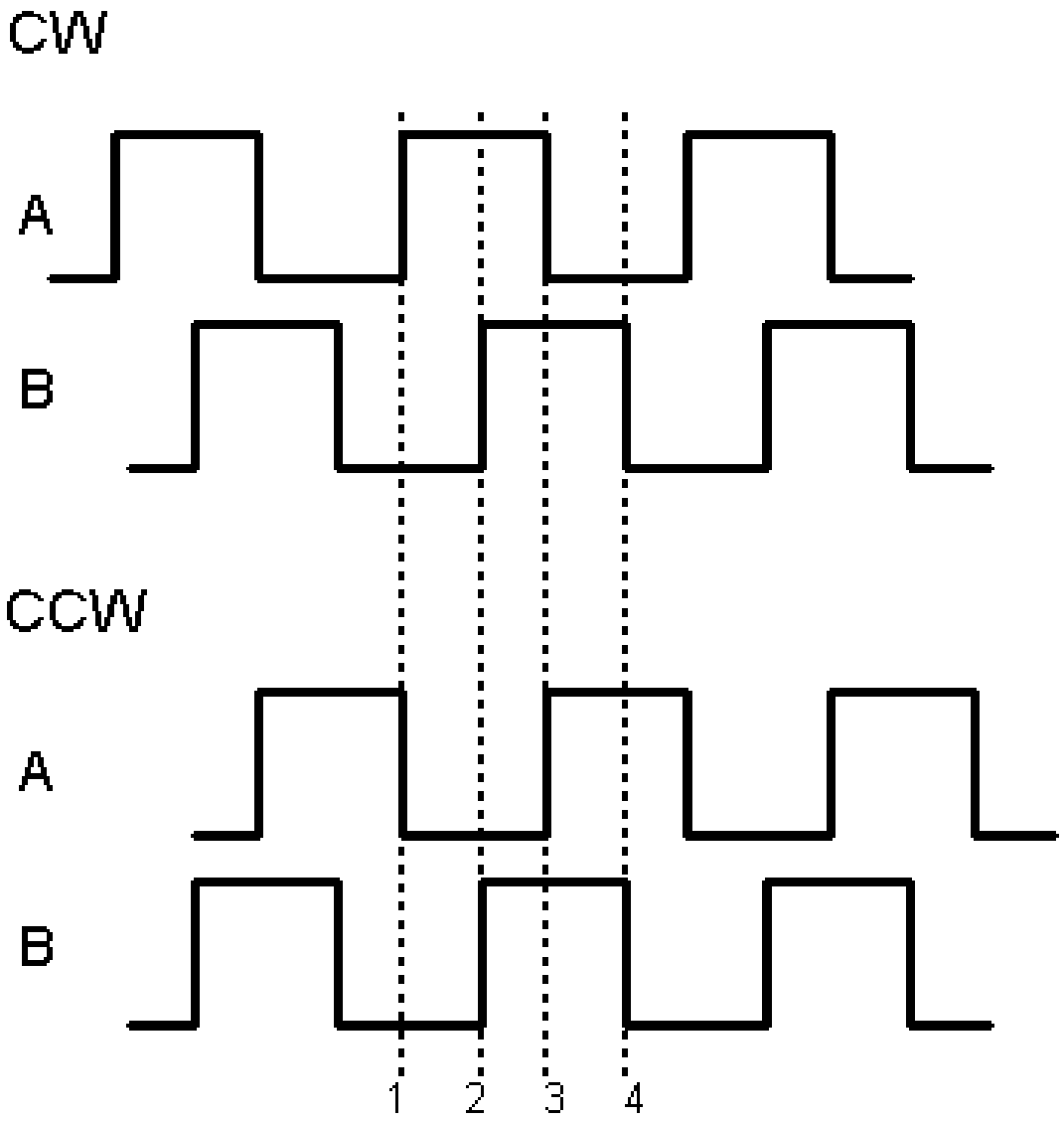


The rotary encoder module counts the number of pulses output in the forward and reverse directions during rotation. unlike a potentiometer, this rotation count is unlimited and the number of pulses per cycle is 20. Press the key (SW) on the rotary encoder to start counting from zero.

There are mainly two types of rotary encoders: absolute and incremental (relative) encoders. An incremental one is used in this kit.

Incremental encoders give two-phase square waves, their phase difference is 90, usually called A channel and B channel.

As shown on the right, when channel A changes from high level to low level, if channel B is high level, it indicates the rotary encoder spins clockwise (CW); if at that moment channel B is low level, it means spins counterclockwise (CCW). So if we read the value of channel B when channel A is low level, we can know in which direction the rotary encoder rotates.

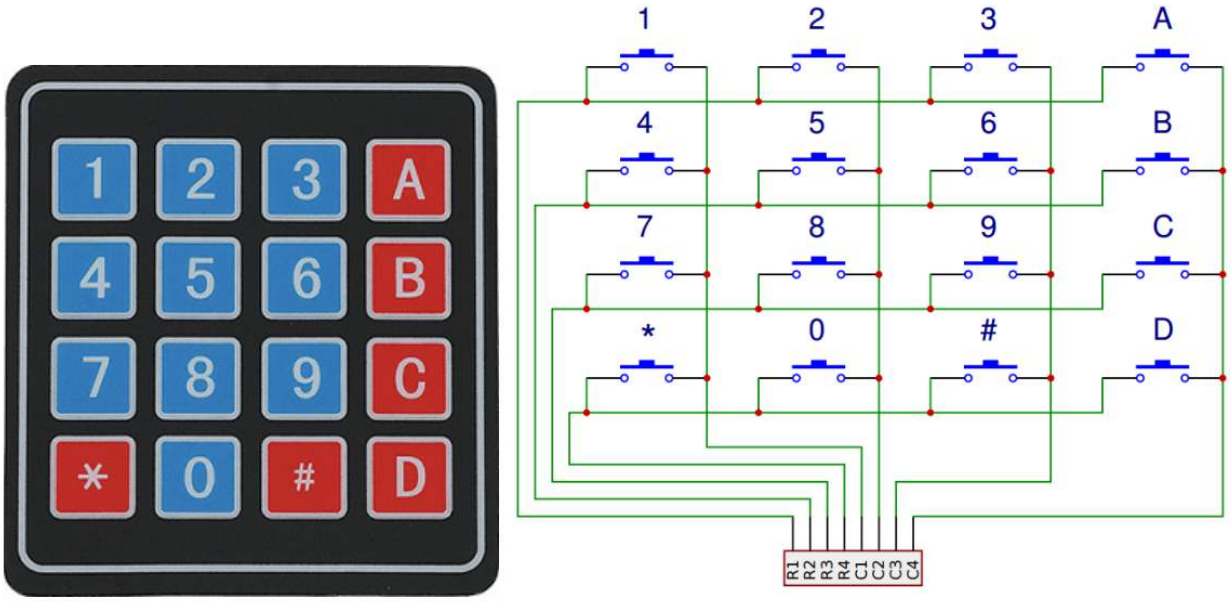


Example

- *2.1.6 Rotary Encoder Module* (C Project)
- *2.1.6 Rotary Encoder Module* (Python Project)

2.2.28 Keypad

A keypad is a rectangular array of 12 or 16 OFF-(ON) buttons. Their contacts are accessed via a header suitable for connection with a ribbon cable or insertion into a printed circuit board. In some keypads, each button connects with a separate contact in the header, while all the buttons share a common ground.



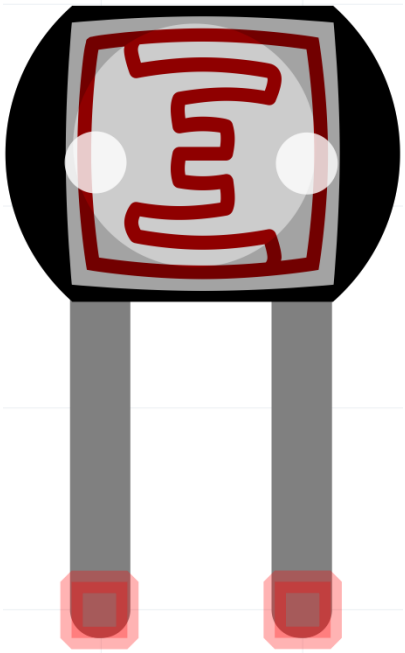
More often, the buttons are matrix encoded, meaning that each of them bridges a unique pair of conductors in a matrix. This configuration is suitable for polling by a microcontroller, which can be programmed to send an output pulse to each of the four horizontal wires in turn. During each pulse, it checks the remaining four vertical wires in sequence, to determine which one, if any, is carrying a signal. Pullup or pulldown resistors should be added to the input wires to prevent the inputs of the microcontroller from behaving unpredictably when no signal is present.

Example

- *2.1.8 Keypad* (C Project)
- *3.1.8 Password Lock* (C Project)
- *3.1.11 GAME– Guess Number* (C Project)
- *2.1.8 Keypad* (Python Project)
- *4.1.14 Password Lock* (Python Project)
- *4.1.17 GAME– Guess Number* (Python Project)

Sensor

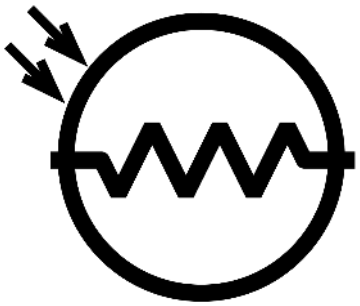
2.2.29 Photoresistor



A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.

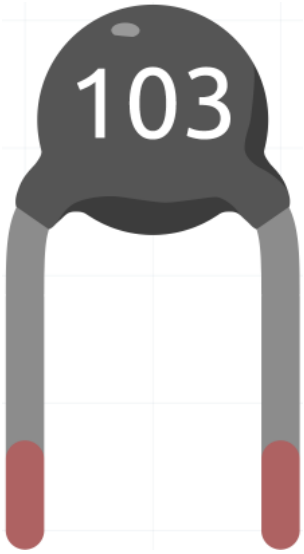


- Photoresistor - Wikipedia

Example

- *2.2.1 Photoresistor* (C Project)
- *2.2.1 Photoresistor* (Python Project)

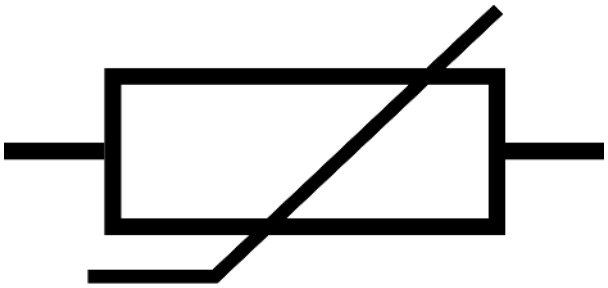
2.2.30 Thermistor



A thermistor is a type of resistor whose resistance is strongly dependent on temperature, more so than in standard resistors. The word is a combination of thermal and resistor. Thermistors are widely used as inrush current limiters, temperature sensors (negative temperature coefficient or NTC type typically), self-resetting overcurrent protectors, and self-regulating heating elements (positive temperature coefficient or PTC type typically).

- Thermistor - Wikipedia

Here is the electronic symbol of thermistor.



Thermistors are of two opposite fundamental types:

- With NTC thermistors, resistance decreases as temperature rises usually due to an increase in conduction electrons bumped up by thermal agitation from valency band. An NTC is commonly used as a temperature sensor, or in series with a circuit as an inrush current limiter.
- With PTC thermistors, resistance increases as temperature rises usually due to increased thermal lattice agitations particularly those of impurities and imperfections. PTC thermistors are commonly installed in series with a circuit, and used to protect against overcurrent conditions, as resettable fuses.

In this kit we use an NTC one. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature:

RT = RN * expB(1/TK – 1/TN)

- **RT** is the resistance of the NTC thermistor when the temperature is TK.

- **RN** is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.
- **TK** is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- **TN** is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And **B(beta)**, the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- **exp** is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

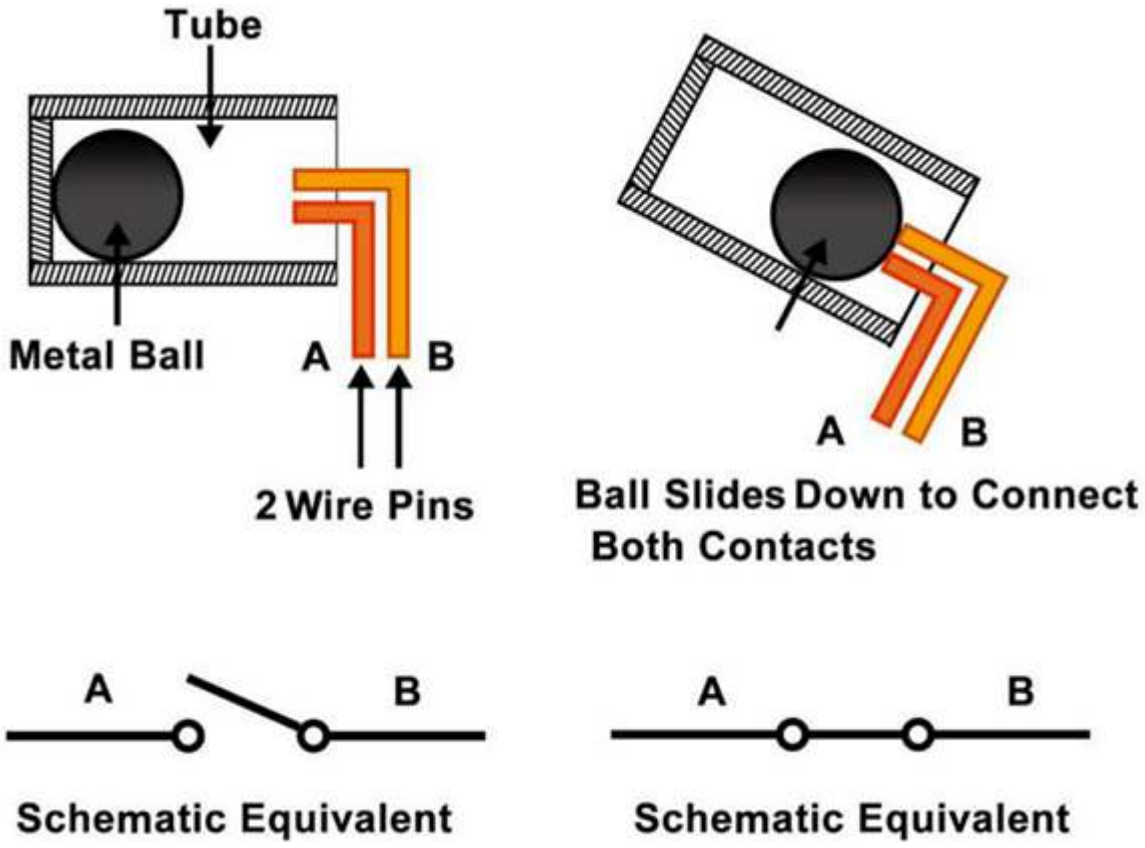
Example

- *2.2.2 Thermistor* (C Project)
- *3.1.4 Smart Fan* (C Project)
- *3.1.7 Overheat Monitor* (C Project)
- *2.2.2 Thermistor* (Python Project)
- *4.1.10 Smart Fan* (Python Project)
- *4.1.13 Overheat Monitor* (Python Project)

2.2.31 Tilt Switch

The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.

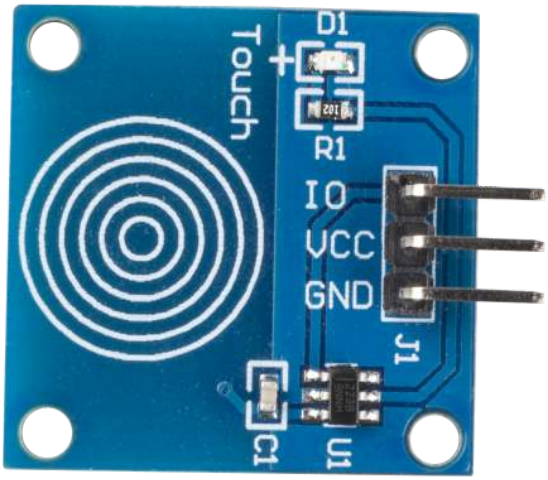


- SW520D Tilt Switch Datasheet

Example

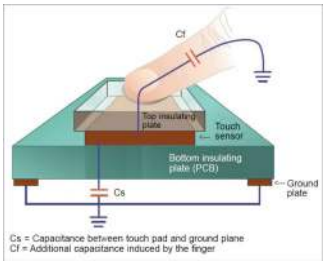
- *2.1.5 Tilt Switch* (C Project)
- *3.1.12 GAME - 10 Second* (C Project)
- *2.1.5 Tilt Switch* (Python Project)
- *4.1.18 GAME - 10 Second* (Python Project)
- *1.3 Tumbler* (Scratch Project)

2.2.32 Touch Switch Module



Touch switch module works by detecting a change in capacitance due to influence of an external object. The touch plate is covered with insulating material, and the user does not come in contact with the electrical circuit.

A capacitive touch switch has different layers—top insulating face plate followed by touch plate, another insulating layer and then ground plate.



In practice, a capacitive sensor can be made on a double-sided PCB by regarding one side as the touch sensor and the opposite side as ground plate of the capacitor. When power is applied across these plates, the two plates get charged. In equilibrium state, the plates have the same voltage as the power source.

The touch detector circuit has an oscillator whose frequency is dependent on capacitance of the touchpad. When a finger is moved close to the touchpad, additional capacitance causes frequency of this internal oscillator to change. The detector circuit tracks oscillator frequency at timed intervals, and when the shift crosses the threshold change, the circuit triggers a key-press event.

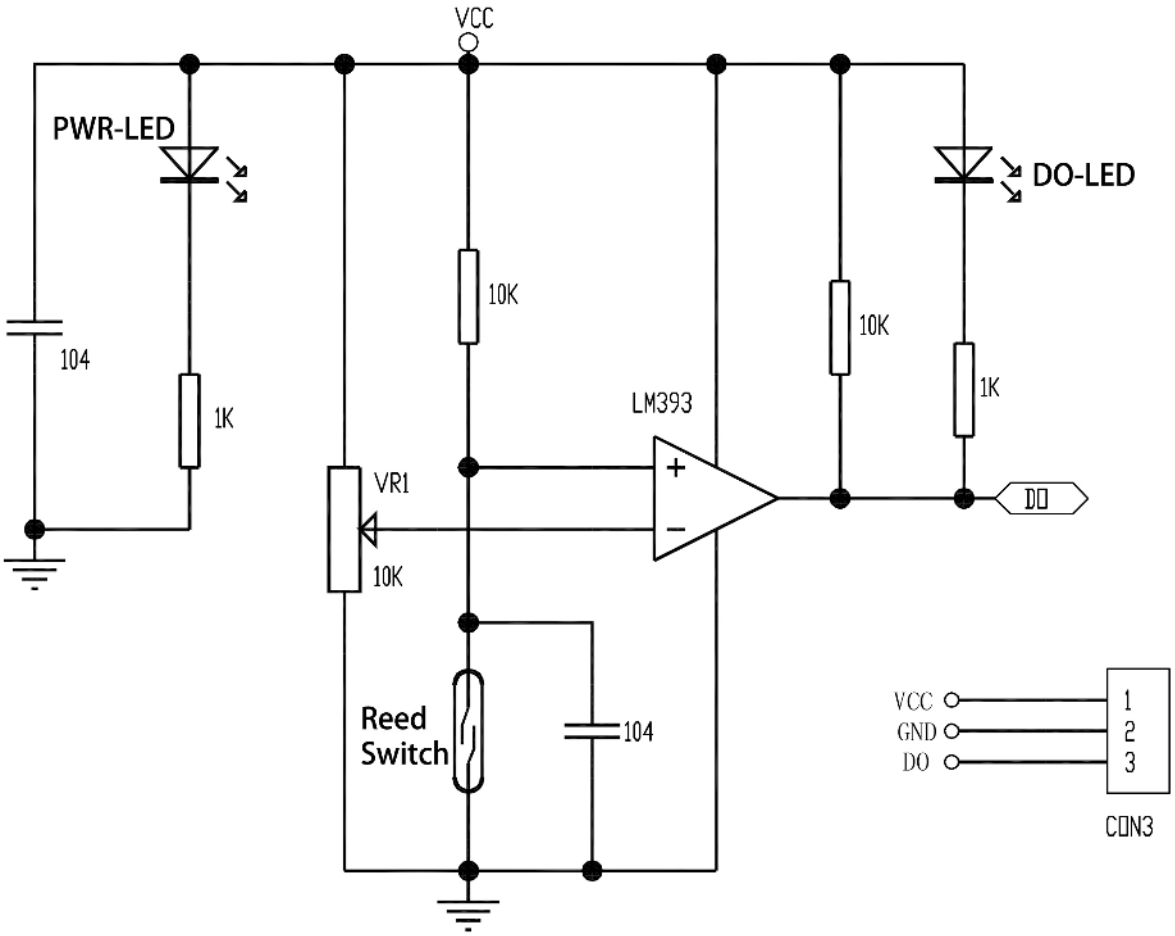
Example

- *2.1.3 Touch Switch Module* (C Project)
- *2.1.3 Touch Switch Module* (Python Project)
- *1.9 Drumming* (Scratch Project)

2.2.33 Reed Switch Module

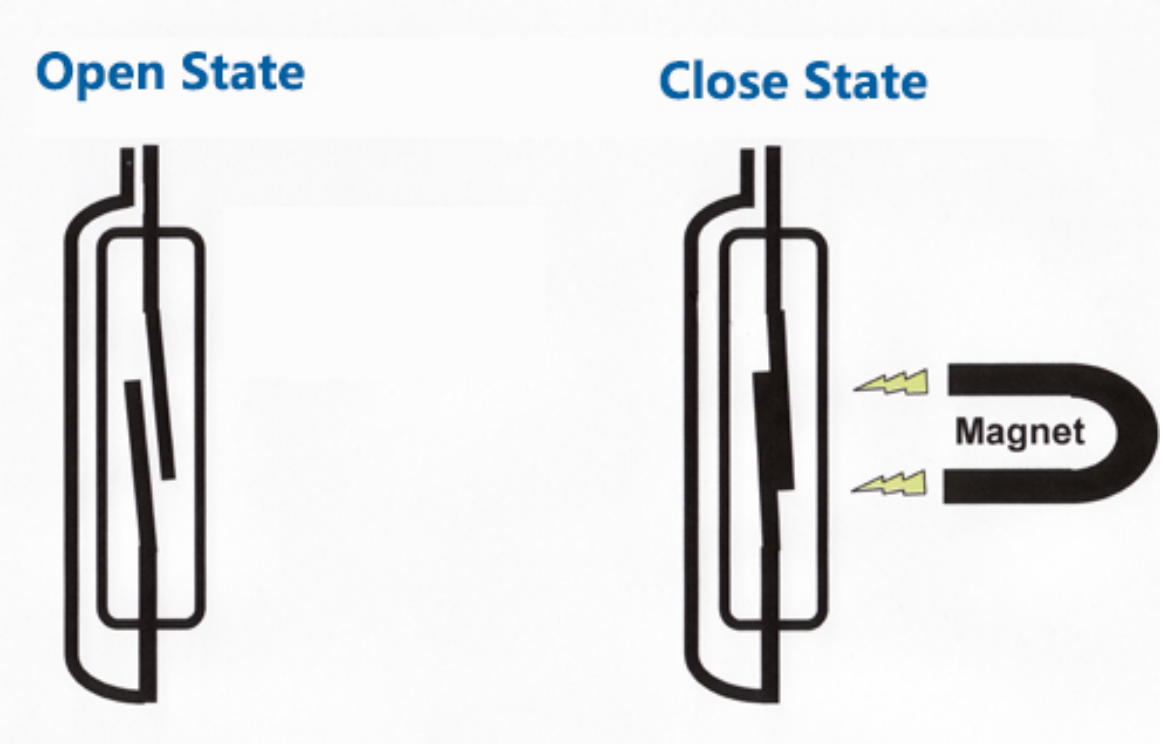
- Using normally open type reed switch.
- Comparator output, clean signal, good waveform, strong driving ability, more than 15mA.
- Working voltage: 3.3V-5V
- Output form: digital switch output (0 and 1).
- With fixed bolt holes for easy installation.
- Small board PCB size: 3.2cm x 1.4cm.
- Use wide voltage LM393 comparator.

The reed switch module consists of a reed switch, potentiometer, LM393 comparator, LED, etc. The internal circuit is shown below, when the magnet is close to the module, it will be on, and the module will output low level; when there is no magnetism, it will be off, and output high level. Reed switch and magnet induction distance should be within 1.5cm, beyond will not be sensitive or no trigger phenomenon, you can also adjust the sensitivity through the potentiometer on the module.



Reed switch, also known as a magnetic switch or reed switch.

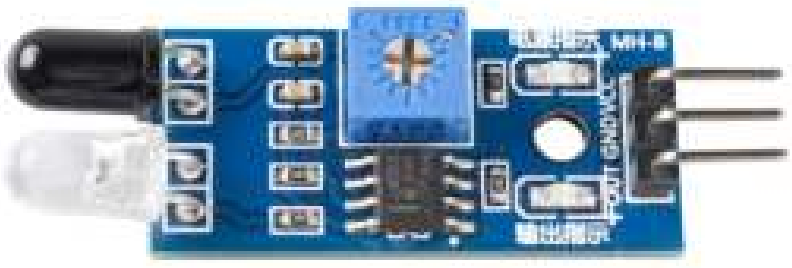
It has two internal metal reeds, sealed in a glass tube, which is filled with inert gas. Normally the two reeds overlap each other, but are separated by a gap, and the circuit is broken. When there is a magnetic object close to the two reeds will produce a mutual attraction of the magnetic force, which will be sucked together, the circuit is connected. Therefore, the reed switch can be used to make a magnetic sensor.



Example

- *2.2.4 Reed Switch Module* (C Project)
- *2.2.4 Reed Switch Module* (Python Project)
- *4.1.6 Magnetic Induction Alarm System* (Python Project)
- *1.6 Vanishing Vase* (Scratch Project)

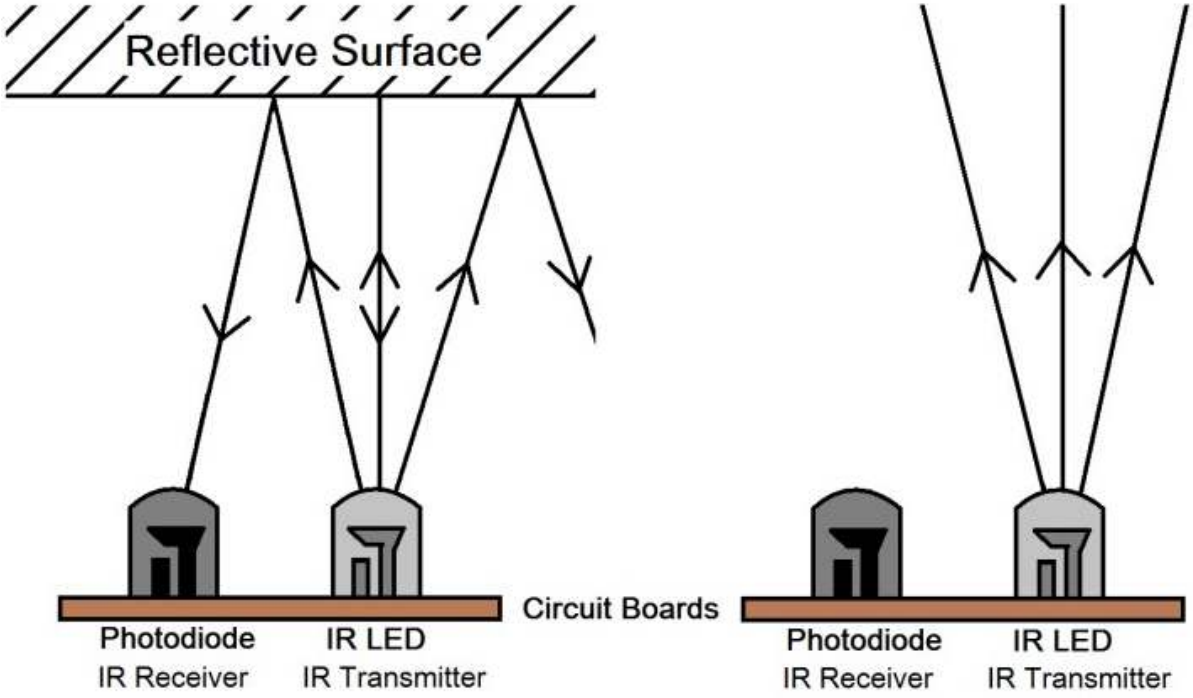
2.2.34 Obstacle Avoidance Module



The IR obstacle avoidance module has strong adaptability to environmental light, it has a pair of infrared transmitting and receiving tubes.

The transmitting tube emits infrared frequency, when the detection direction encounters an obstacle, the infrared radiation is received by the receiving tube, after the comparator circuit processing, the green indicator will light up and output low level signal.

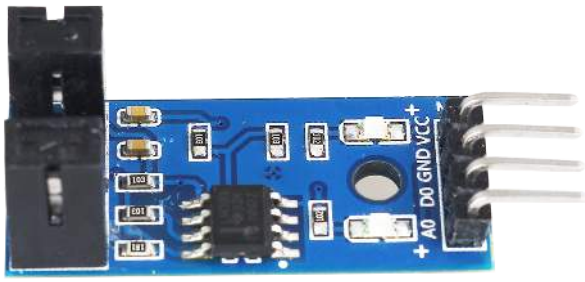
The detection distance can be adjusted by potentiometer, the effective distance range 2-30cm.



Example

- *2.2.5 IR Obstacle Avoidance Module* (C Project)
- *2.2.5 IR Obstacle Avoidance Sensor* (Python Project)
- *1.11 Repelling locusts* (Scratch Project)

2.2.35 Speed Sensor Module



The speed sensor consists of two parts: a transmitter and a receiver. The transmitter emits light, which then enters the receiver.

If the light beam between the emitter and receiver is interrupted by an obstacle, the receiver will not detect the incident light, then the D0 pin will output low level.

Note: The A0 pin on this module is empty and there is no circuit.

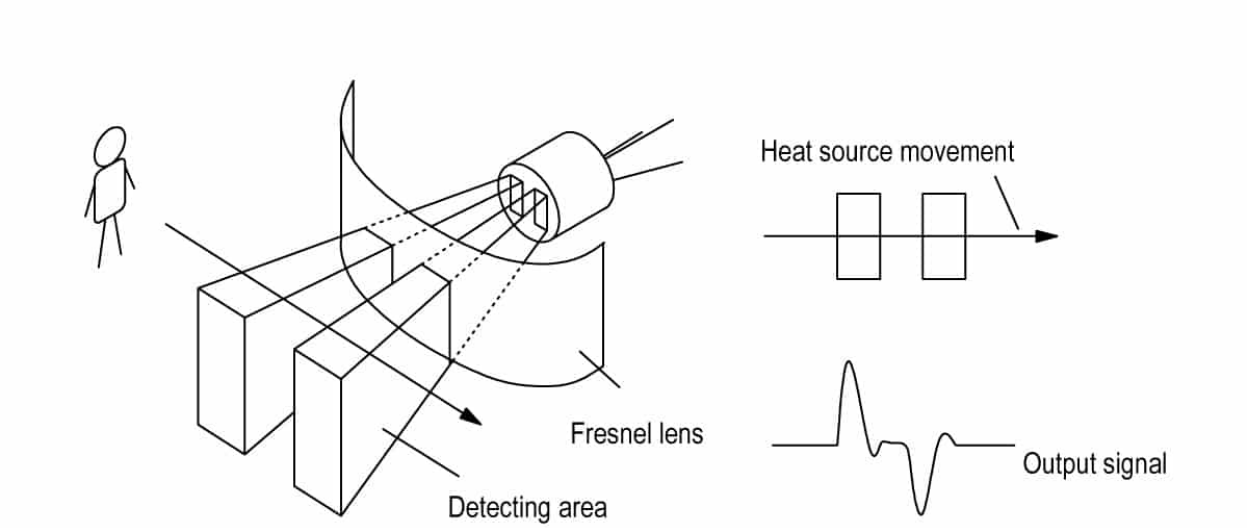
Example

- *2.2.6 Speed Sensor Module* (C Project)
- *2.2.6 Speed Sensor Module* (Python Project)
- *1.7 Piggy Bank* (Scratch Project)

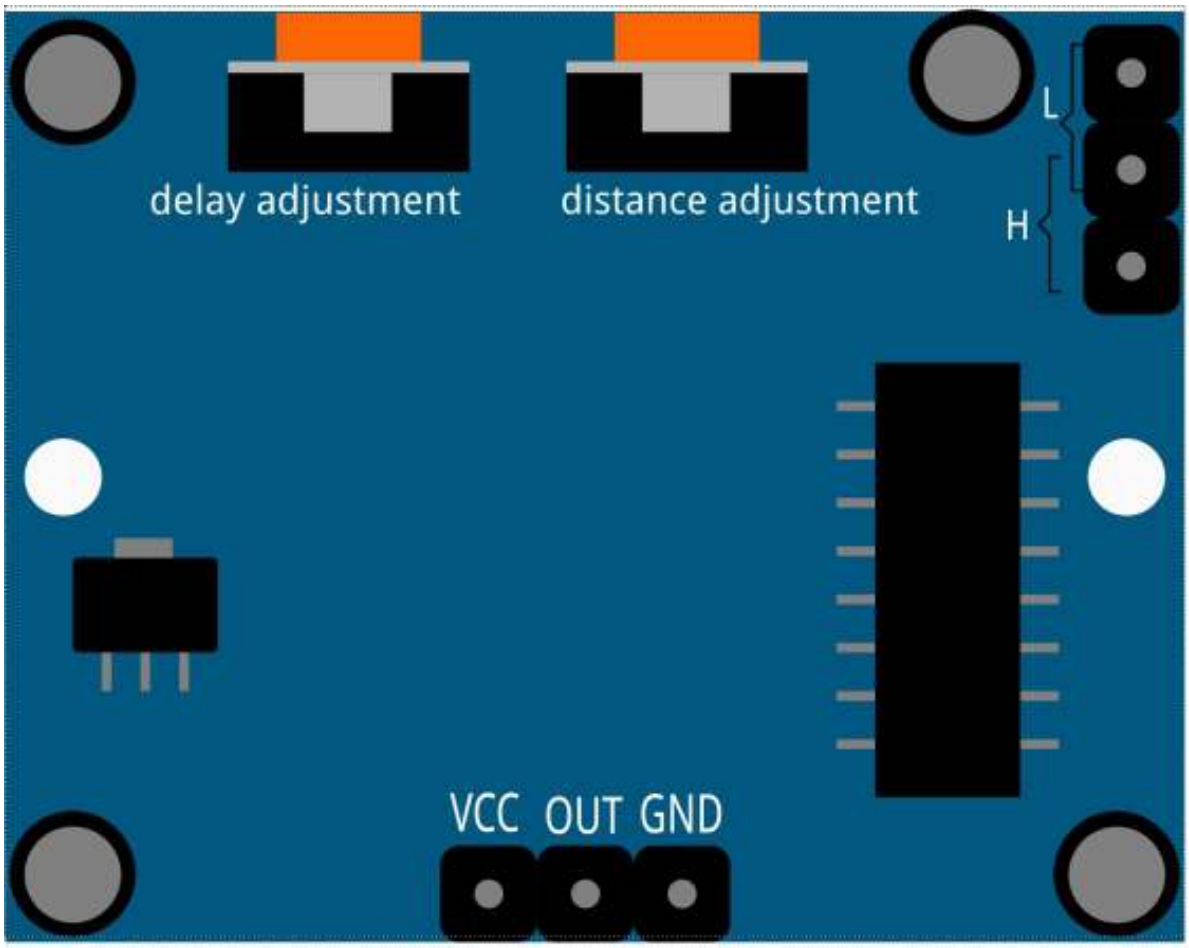
2.2.36 PIR Motion Sensor Module

The PIR sensor detects infrared heat radiation that can be used to detect the presence of organisms that emit infrared heat radiation.

The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other , which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.



After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.



Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.

Delay adjustment

Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

Two Trigger Modes

Choosing different modes by using the jumper cap.

- **H**: Repeatable trigger mode, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range, the output will keep being the high level.
- **L**: Non-repeatable trigger mode, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

Example

- *2.2.7 PIR* (C Project)
- *2.2.7 PIR* (Python Project)
- *4.1.4 Automatic Capture Camera* (Python Project)
- *1.5 Wake up the Owl* (Scratch Project)

2.2.37 Ultrasonic Module

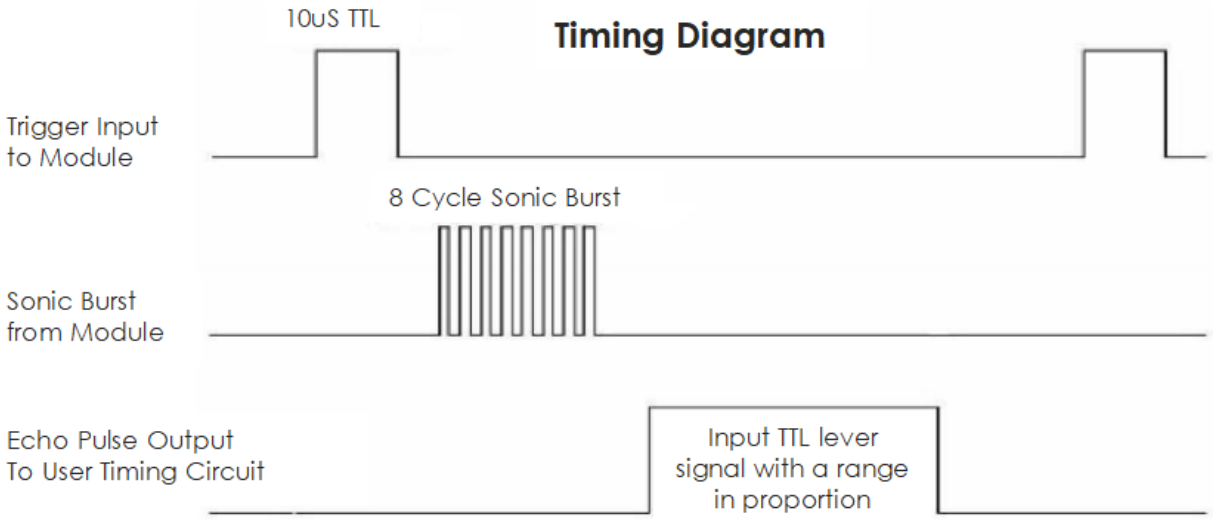
Ultrasonic ranging module provides 2cm - 400cm non-contact measurement function, and the ranging accuracy can reach to 3mm. It can ensure that the signal is stable within 5m, and the signal is gradually weakened after 5m, till the 7m position disappears.

The module includes ultrasonic transmitters, receiver and control circuit. The basic principles are as follows:

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.

The timing diagram is shown below.



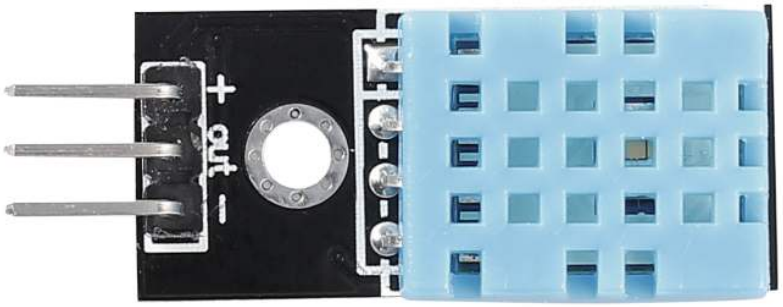
You only need to supply a short 10us pulse for the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula: us / 58 = centimeters or us / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; you are suggested to use measurement cycle over 60ms in order to prevent signal collisions of trigger signal and the echo signal.

Example

- *2.2.8 Ultrasonic Sensor Module* (C Project)
- *3.1.3 Reversing Alarm* (C Project)
- *2.2.8 Ultrasonic Sensor Module* (Python Project)
- *4.1.9 Reversing Alarm* (Python Project)

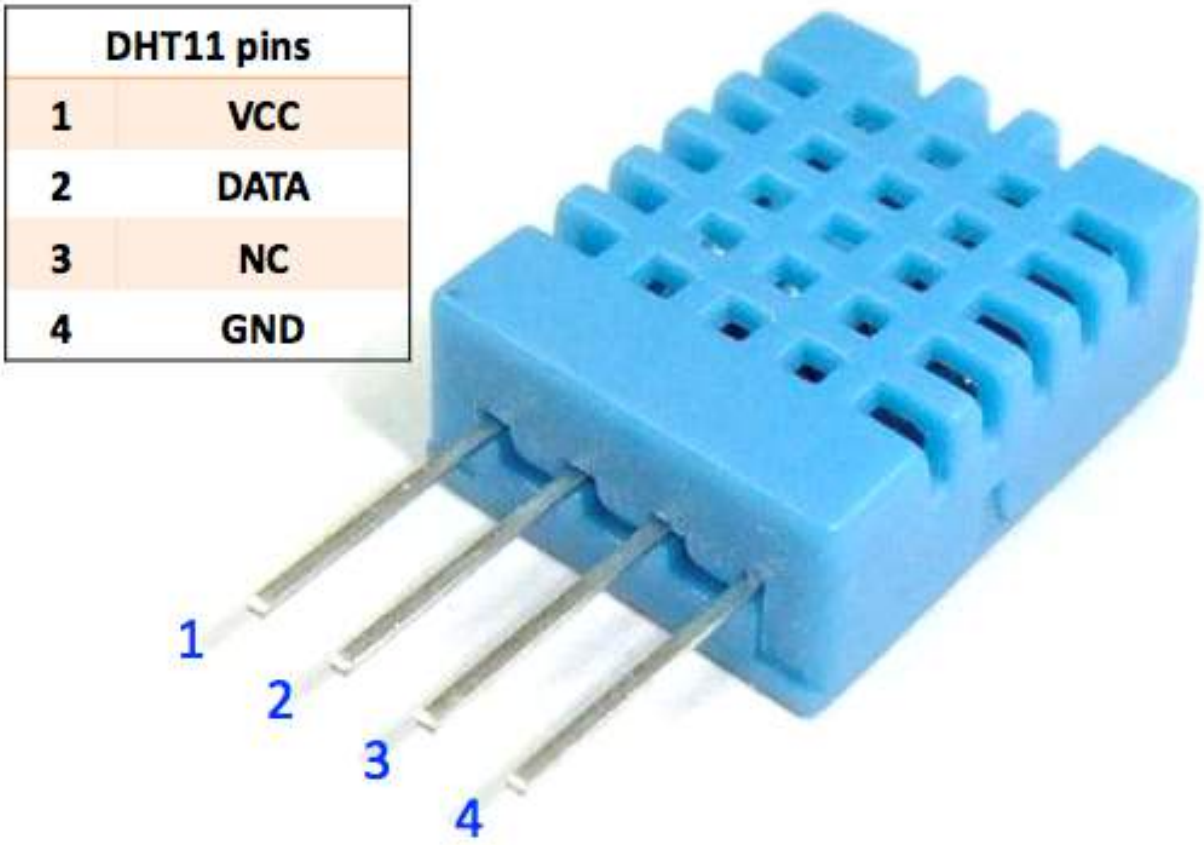
2.2.38 Humiture Sensor Module



The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature

and humidity sensing technology are applied to ensure that the product has high reliability and excellent long-term stability.

Only three pins are available for use: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humiture data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum).

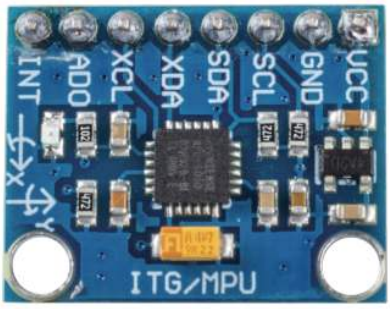


- DHT11 Datasheet

Example

- *2.2.3 DHT-11* (C Project)
- *2.2.3 DHT-11* (Python Project)

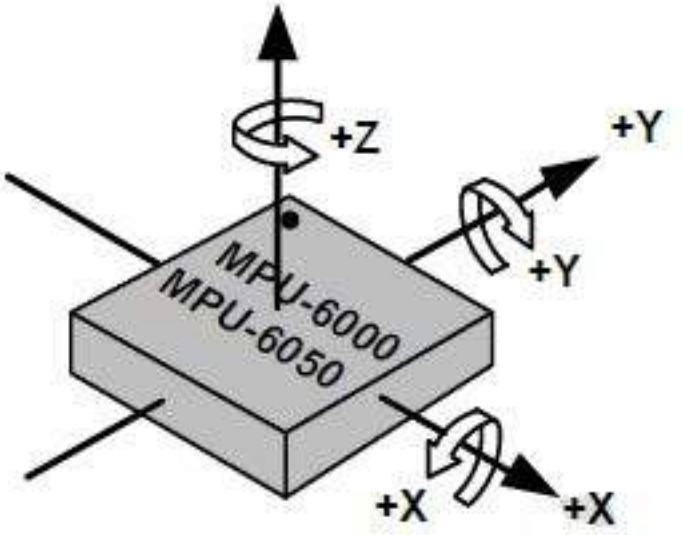
2.2.39 MPU6050 Module



The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.

Its three coordinate systems are defined as follows:

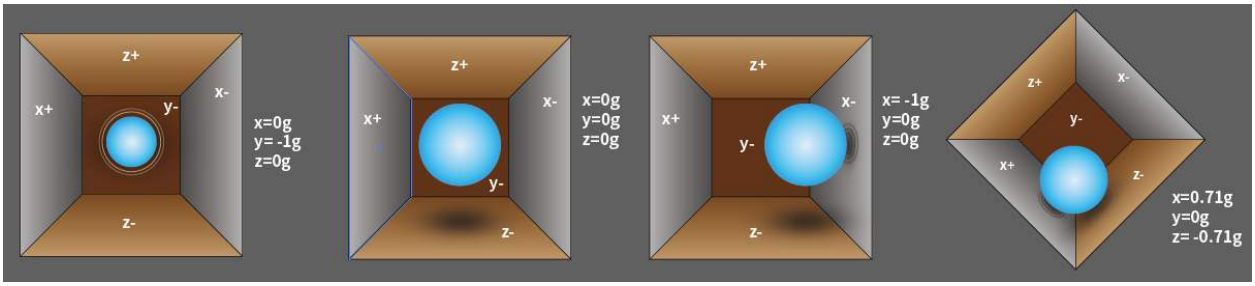
Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



3-axis Accelerometer

The accelerometer works on the principle of piezo electric effect, the ability of certain materials to generate an electric charge in response to applied mechanical stress.

Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude.



We can use the MPU6050 to detect its acceleration on each coordinate axis (in the stationary desktop state, the Z-axis acceleration is 1 gravity unit, and the X and Y axes are 0). If it is tilted or in a weightless/overweight condition, the corresponding reading will change.

There are four kinds of measuring ranges that can be selected programmatically: +/-2g, +/-4g, +/-8g, and +/-16g (2g by default) corresponding to each precision. Values range from -32768 to 32767.

The reading of accelerometer is converted to an acceleration value by mapping the reading from the reading range to the measuring range.

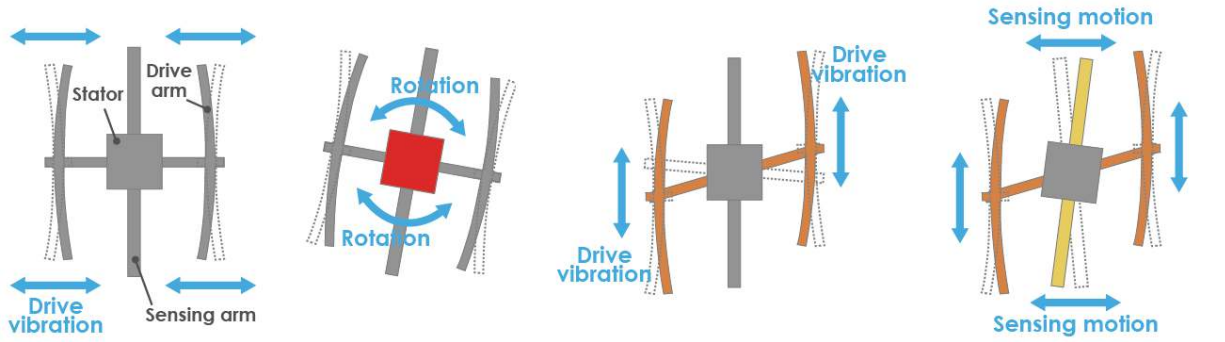
Acceleration = (Accelerometer axis raw data / 65536 * full scale Acceleration range) g

Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as +/-2g:

Acceleration along the X axis = (16384 / 65536 * 4) g =1g

3-axis Gyroscope

Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified.



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.

The Gyroscope also has four kinds of measuring ranges: +/- 250, +/- 500, +/- 1000, +/- 2000. The calculation method and Acceleration are basically consistent.

The formula for converting the reading into angular velocity is as follows:

Angular velocity = (Gyroscope axis raw data / 65536 * full scale Gyroscope range) °/s

The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250°/ s:

Angular velocity along the X axis = (16384 / 65536 * 500)°/s =125°/s

Example

- *2.2.9 MPU6050 Module* (C Project)
- *2.2.9 MPU6050 Module* (Python Project)

2.2.40 MFRC522 Module

RFID

Radio Frequency Identification (RFID) refers to technologies that involve using wireless communication between an object (or tag) and an interrogating device (or reader) to automatically track and identify such objects. The tag transmission range is limited to several meters from the reader. A clear line of sight between the reader and tag is not necessarily required.

Most tags contain at least one integrated circuit (IC) and an antenna. The microchip stores information and is responsible for managing the radio frequency (RF) communication with the reader. Passive tags do not have an independent energy source and depend on an external electromagnetic signal, provided by the reader, to power their operations. Active tags contain an independent energy source, such as a battery. Thus, they may have increased processing, transmission capabilities and range.

MFRC522

MFRC522 is a kind of integrated read and write card chip. It is commonly used in the radio at 13.56MHz. Launched by the NXP Company, it is a low-voltage, low-cost, and small-sized non-contact card chip, a best choice of intelligent instrument and portable handheld device.

The MF RC522 uses advanced modulation and demodulation concept which fully presented in all types of 13.56MHz passive contactless communication methods and protocols. In addition, it supports rapid CRYPTO1 encryption algorithm to verify MIFARE products. MFRC522 also supports MIFARE series of high-speed non-contact communication, with a two-way data transmission rate up to 424kbit/s. As a new member of the 13.56MHz highly integrated reader card series, MF RC522 is much similar to the existing MF RC500 and MF RC530 but there also exists great differences. It communicates with the host machine via the serial manner which needs less wiring. You can choose between SPI, I2C and serial UART mode (similar to RS232), which helps reduce the connection, save PCB board space (smaller size), and reduce cost.

Example

- *2.2.10 MFRC522 RFID Module* (C Project)
- *2.2.10 MFRC522 RFID Module* (Python Project)
- *4.1.19 AttendanceSystem* (Python Project)

2.2.41 Camera Module

Description

This is a 5MP Raspberry Pi camera module with OV5647 sensor. It's plug and play, connect the included ribbon cable to the CSI (Camera Serial Interface) port on your Raspberry Pi and you're ready to go.

The board is small, about 25mm x 23mm x 9mm, and weighs 3g, making it ideal for mobile or other size and weight-critical applications. The camera module has a native resolution of 5 megapixels and has an on-board fixed focus lens that captures still images at 2592 x 1944 pixels, and also supports 1080p30, 720p60 and 640x480p90 video.

Note: The module is only capable of capturing pictures and videos, not sound.

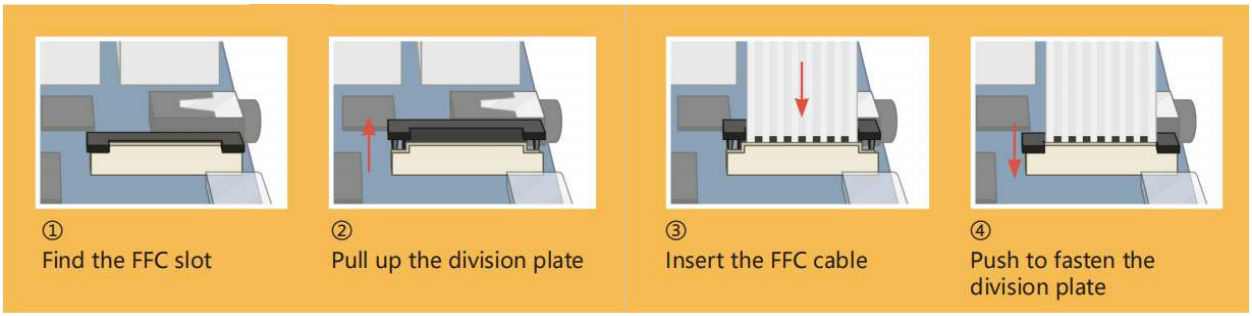
Specification

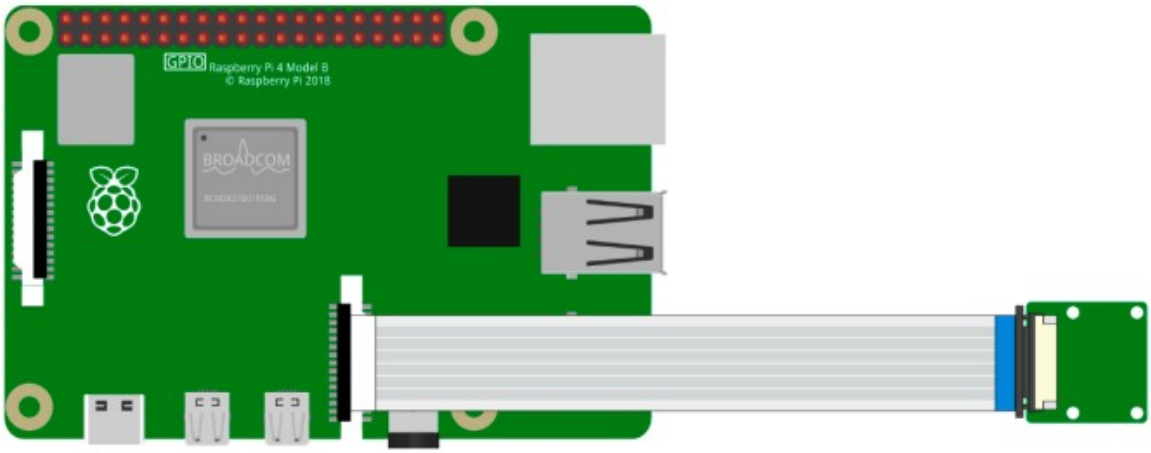
- **Static Images Resolution**: 2592×1944
- **Supported Video Resolution**: 1080p/30 fps, 720p/ 60fps and 640 x480p 60/90 video recording
- **Aperture (F)**: 1.8
- **Visual Angle**: 65 degree
- **Dimension**: 24mmx23.5mmx8mm
- **Weight**: 3g
- **Interface**: CSI connector
- **Supported OS**: Raspberry Pi OS(latest version recommended)

Assemble the Camera Module

On the camera module or Raspberry Pi, you will find a flat plastic connector. Carefully pull out the black fixing switch until the fixing switch is partially pulled out. Insert the FFC cable into the plastic connector in the direction shown and push the fixing switch back into place.

If the FFC wire is installed correctly, it will be straight and will not pull out when you gently pull on it. If not, reinstall it again.





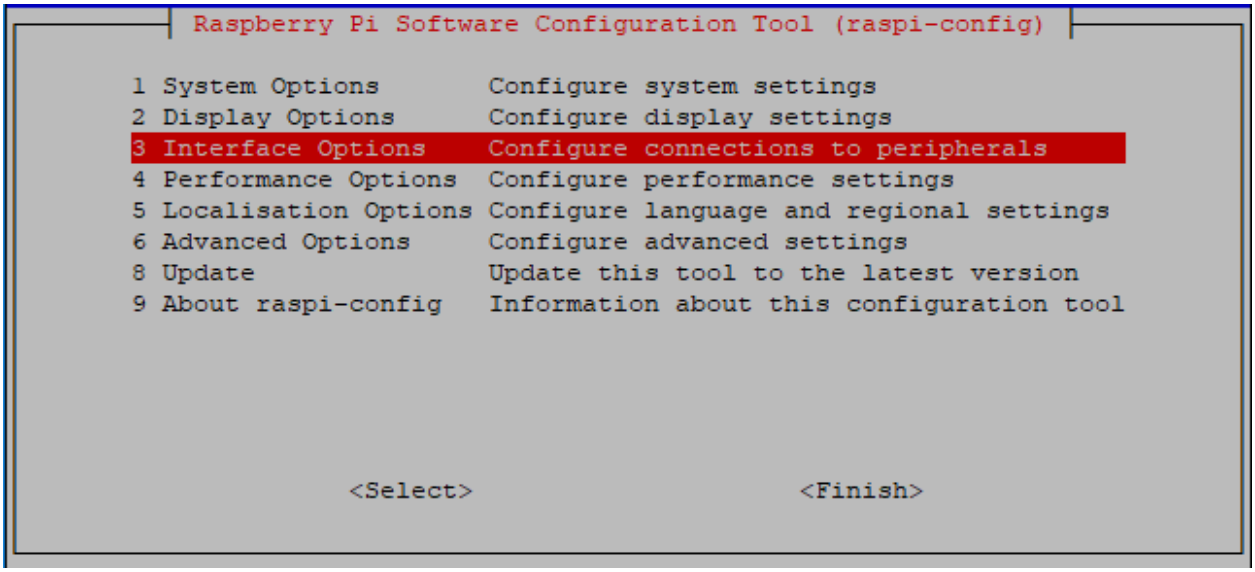
Warning: Do not install the camera with the power on, it may damage your camera.

Enable the Camera Interface

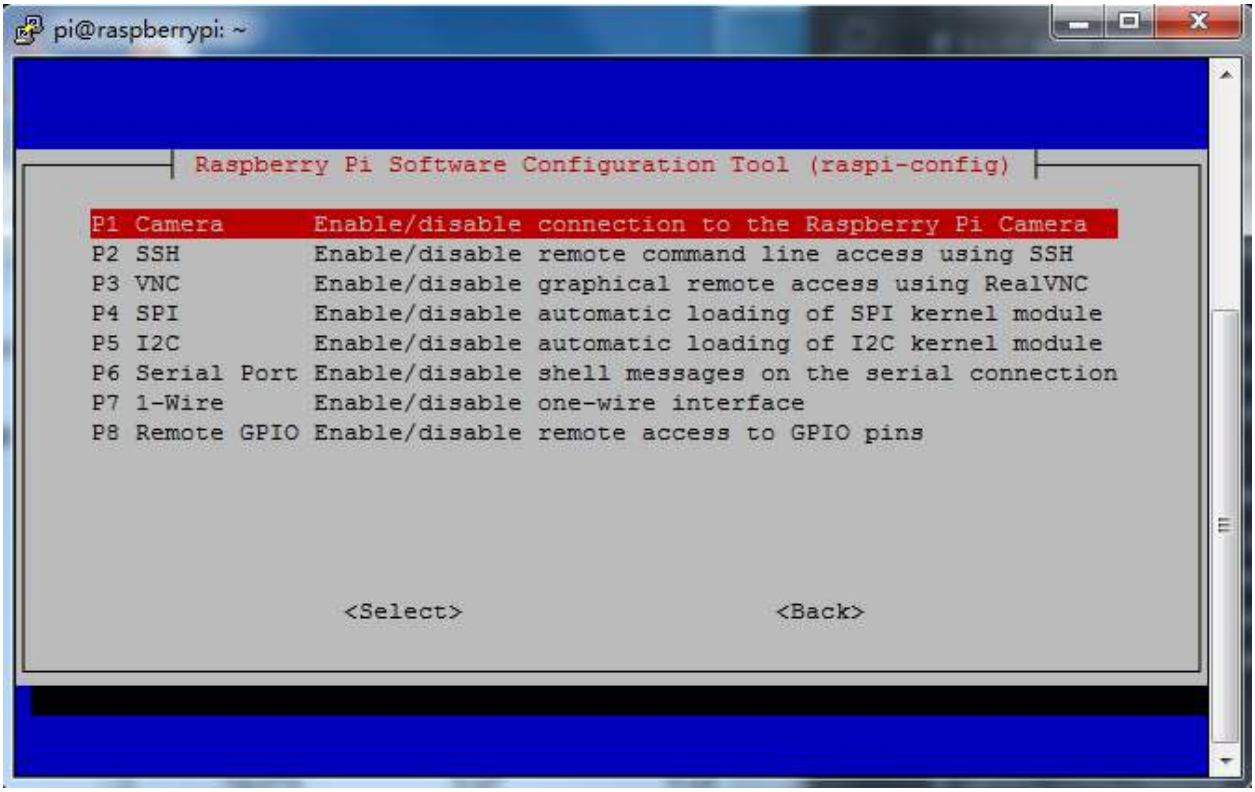
Run the following command to enable the camera interface of your Raspberry Pi. If you have enabled it, skip this; if you do not know whether you have done that or not, please continue.

```
sudo raspi-config
```

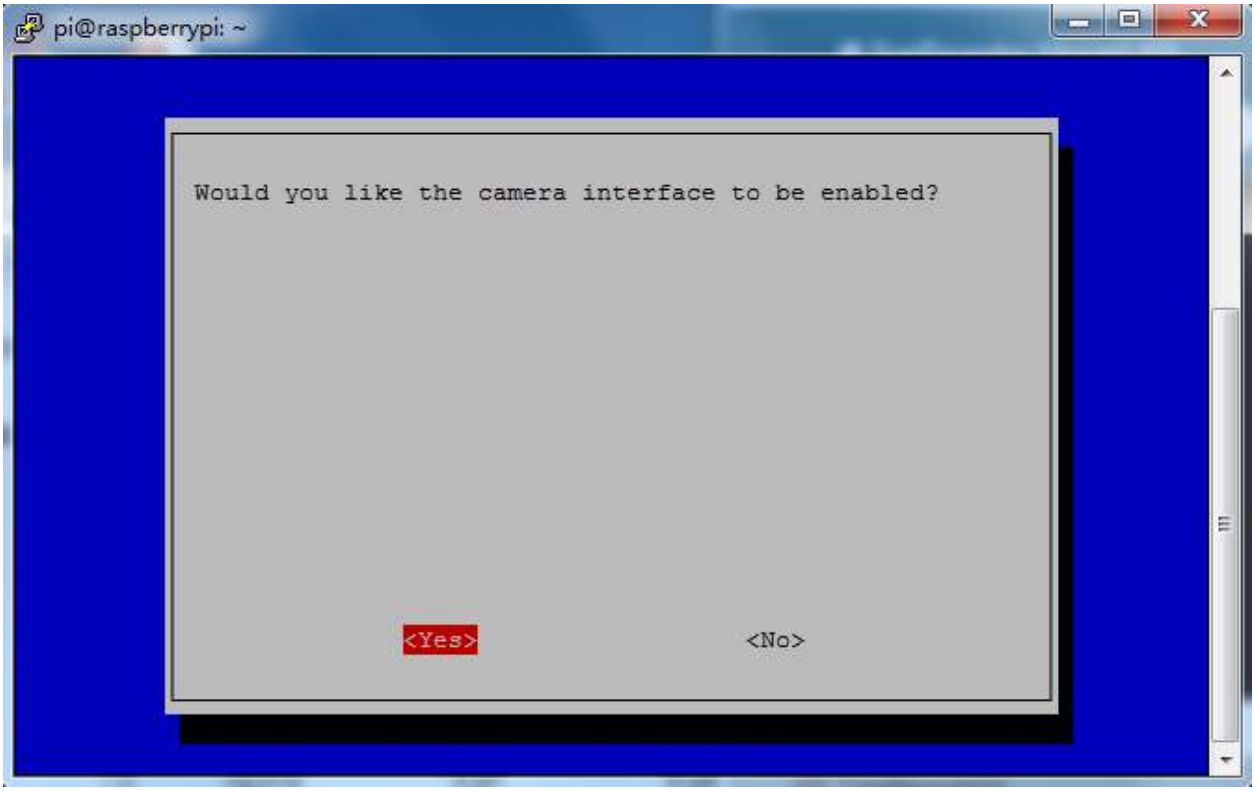
3 Interfacing options



P1 Camera



<Yes>, then <Ok> -> <Finish>



After the configuration is complete, it is recommended to reboot the Raspberry Pi.

```
sudo reboot
```

Example

- *3.1.1 Photograph Module* (Python Project)
- *3.1.2 Video Module* (Python Project)
- *4.1.1 Camera* (Python Project)
- *4.1.4 Automatic Capture Camera* (Python Project)
- *4.1.5 Intelligent Visual Doorbell* (Python Project)
- *1.10 Drumming in the Air* (Scratch Project)
- *1.18 Eating Banana Game* (Scratch Project)

CHAPTER

THREE

INSTALL AND SETUP RASPBERRY PI OS

In this chapter, we firstly learn to start up Raspberry Pi. The content includes installing the OS, Raspberry Pi network and how to open terminal.

Note: You can check the complete tutorial on the official website of the Raspberry Pi: raspberry-pi-setting-up.

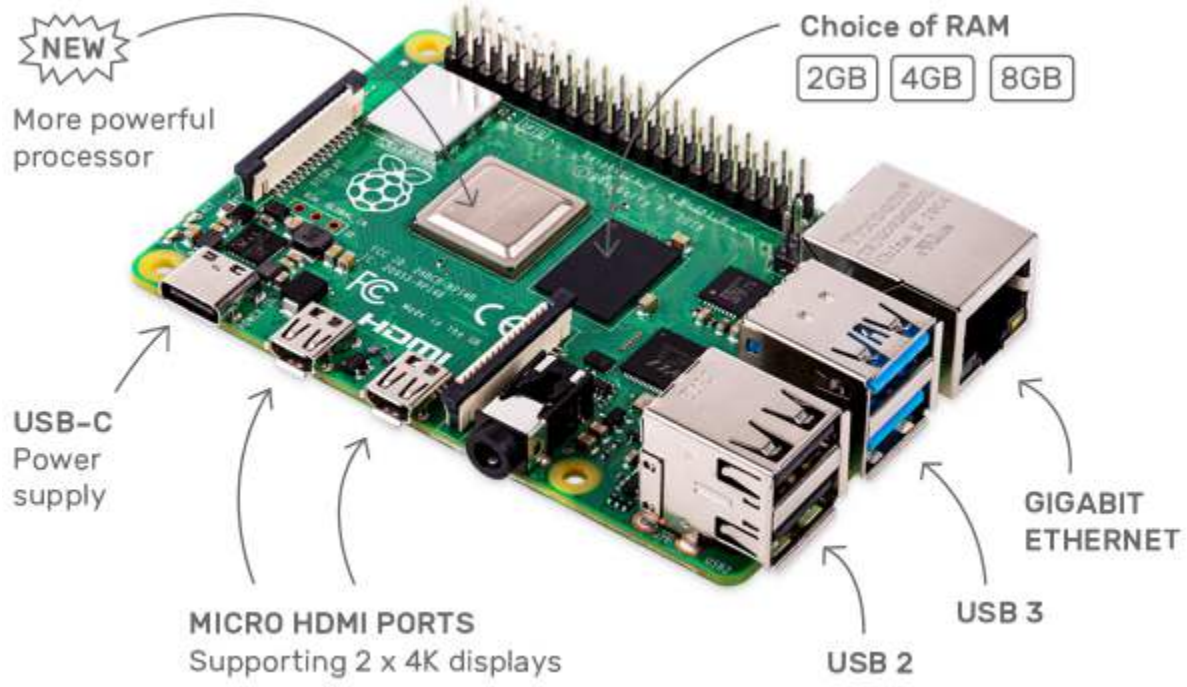
If your Raspberry Pi is set up, you can skip the part and go into the next chapter.

3.1 What Do We Need?

3.1.1 Required Components

Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.



Power Adapter

To connect to a power socket, the Raspberry Pi has a micro USB port (the same found on many mobile phones). You will need a power supply which provides at least 2.5 amps.

Micro SD Card

Your Raspberry Pi needs an Micro SD card to store all its files and the Raspberry Pi OS. You will need a micro SD card with a capacity of at least 8 GB

3.1.2 Optional Components

Screen

To view the desktop environment of Raspberry Pi, you need to use the screen that can be a TV screen or a computer monitor. If the screen has built-in speakers, the Pi plays sounds via them.

Mouse & Keyboard

When you use a screen , a USB keyboard and a USB mouse are also needed.

HDMI

The Raspberry Pi has a HDMI output port that is compatible with the HDMI ports of most modern TV and computer monitors. If your screen has only DVI or VGA ports, you will need to use the appropriate conversion line.

Case

You can put the Raspberry Pi in a case; by this means, you can protect your device.

Sound or Earphone

The Raspberry Pi is equipped with an audio port about 3.5 mm that can be used when your screen has no built-in speakers or when there is no screen operation.

3.2 Installing the OS

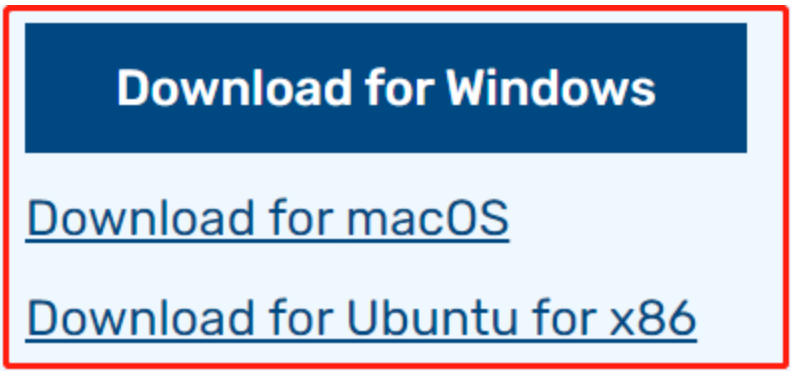
Required Components

Any Raspberry Pi	1 * Personal Computer
1 * Micro SD card	

Step 1

Raspberry Pi have developed a graphical SD card writing tool that works on Mac OS, Ubuntu 18.04 and Windows, and is the easiest option for most users as it will download the image and install it automatically to the SD card.

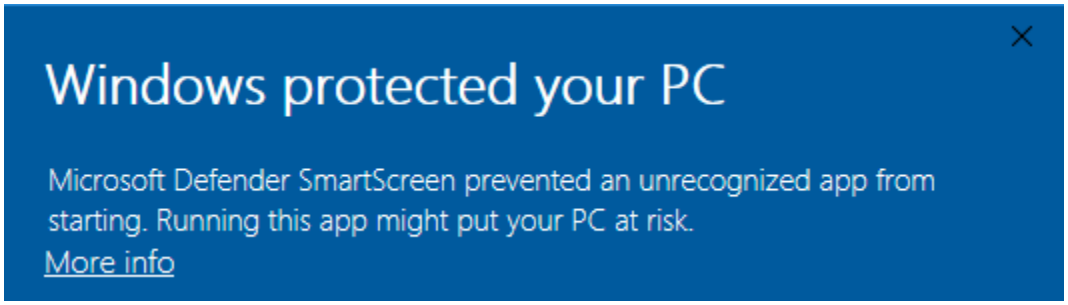
Visit the download page: https://www.raspberrypi.org/software/. Click on the link for the **Raspberry Pi Imager** that matches your operating system, when the download finishes, click it to launch the installer.



Step 2

When you launch the installer, your operating system may try to block you from running it. For example, on Windows I receive the following message:

If this pops up, click on **More info** and then **Run anyway**, then follow the instructions to install the Raspberry Pi Imager.

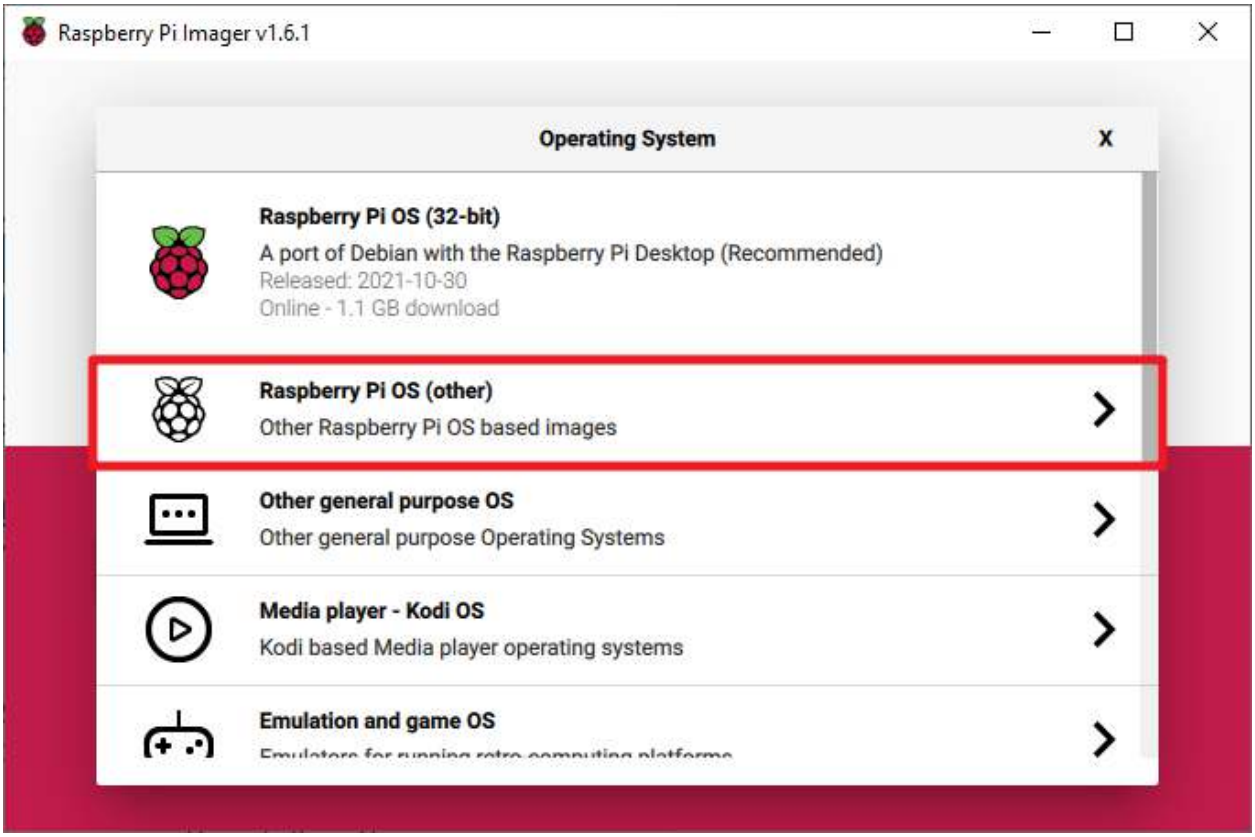


Step 3

Insert your SD card into the computer or laptop SD card slot.

Step 4

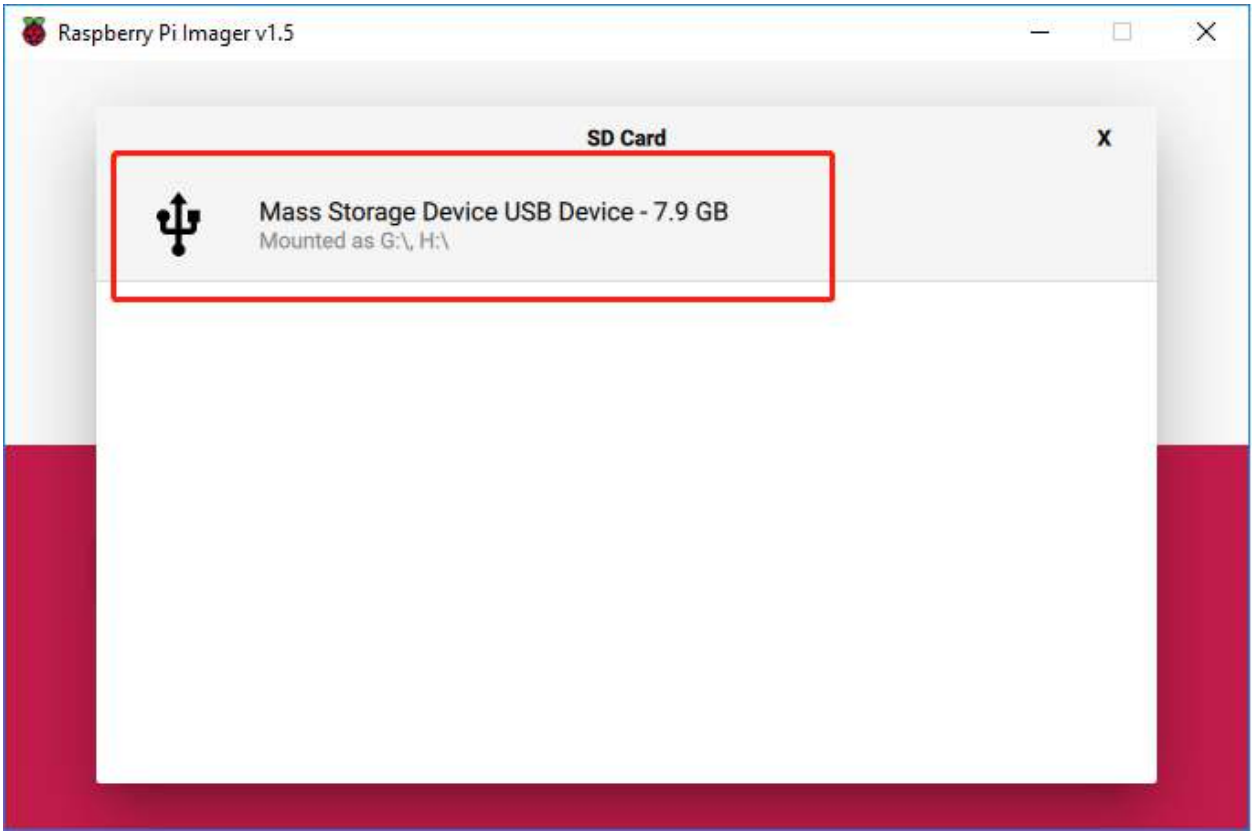
Download the raspios_armhf-2020-05-28 image and select it in Raspberry Pi Imager.



Warning: Raspberry Pi OS has major changes after the 2021-05-28 version, which may cause some functions to be unavailable. Please do not use the latest version for now.

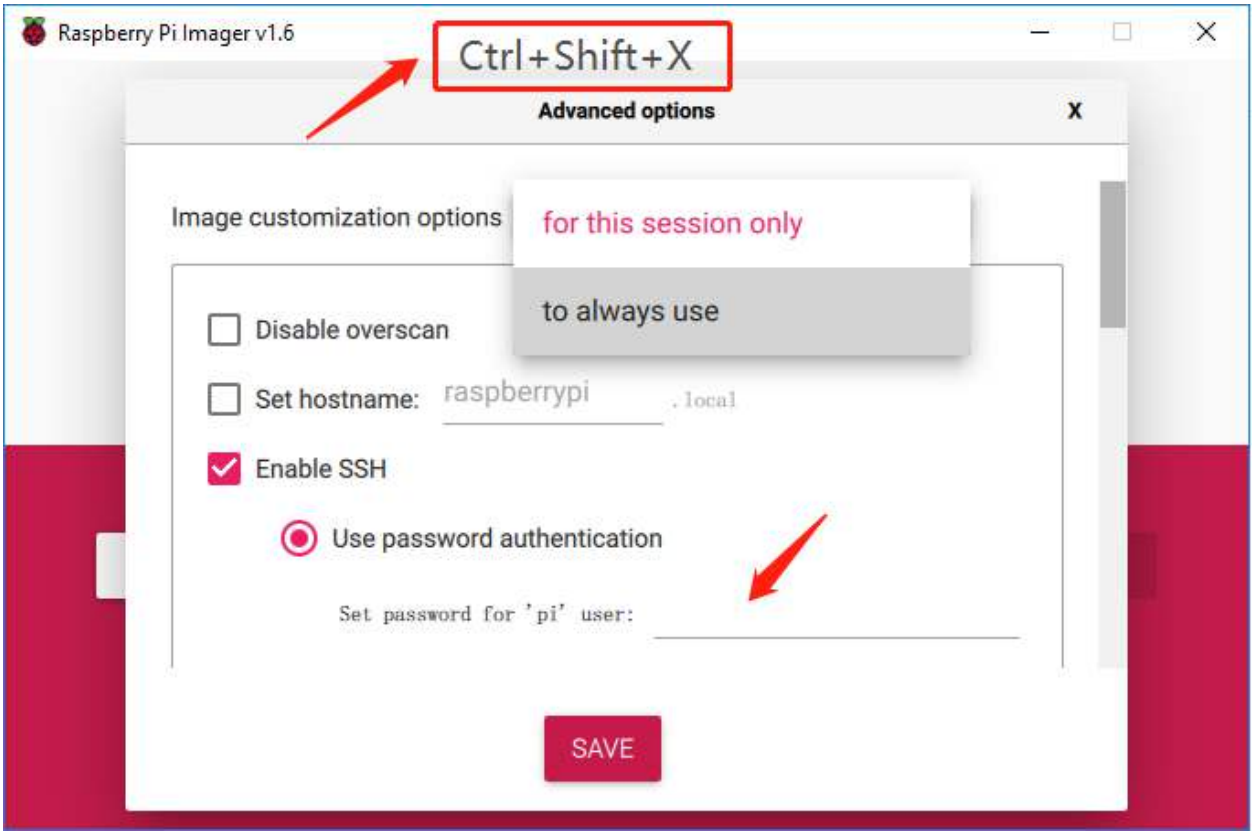
Step 5

Select the SD card you are using.



Step 6

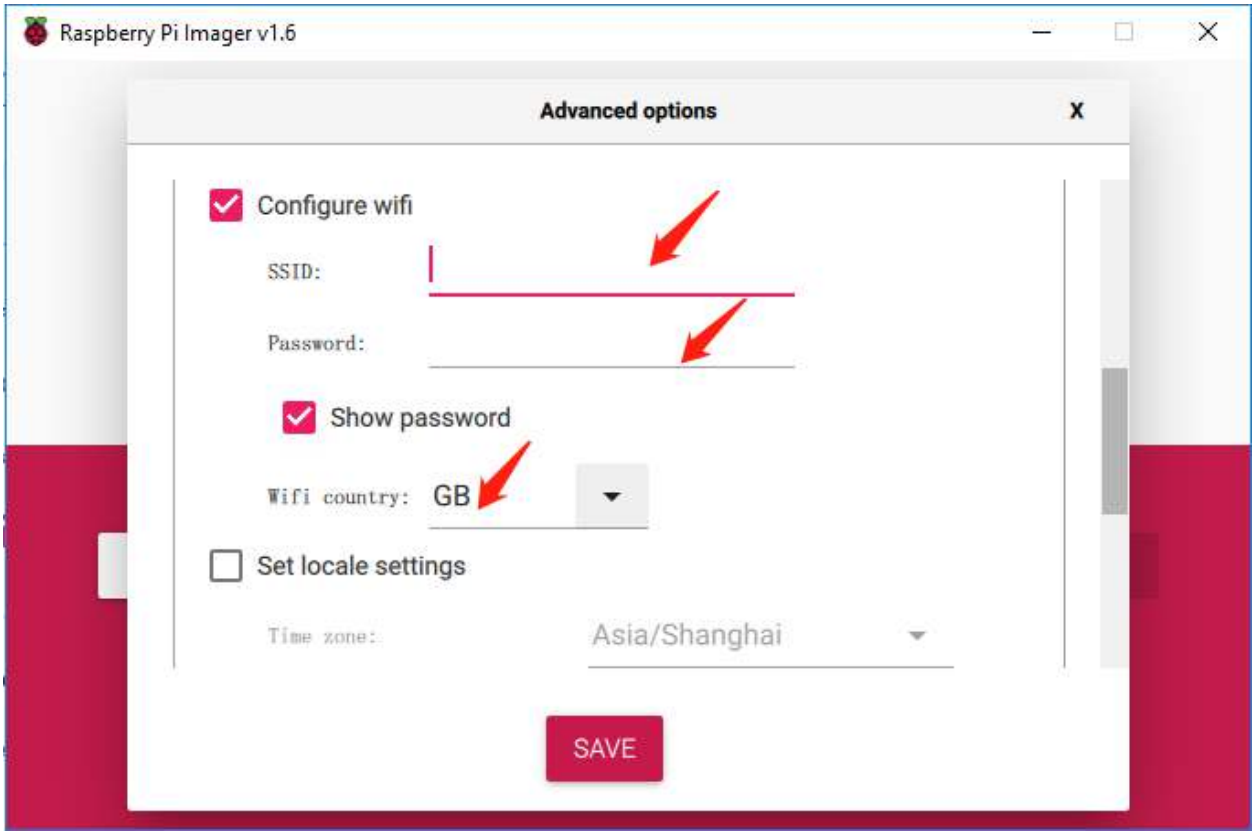
Press **Ctrl+Shift+X** to open the **Advanced options** page to enable SSH and configure wifi, these 2 items must be set, the others depend on your choice . You can choose to always use this image customization options.



Then scroll down to complete the wifi configuration and click **SAVE**.

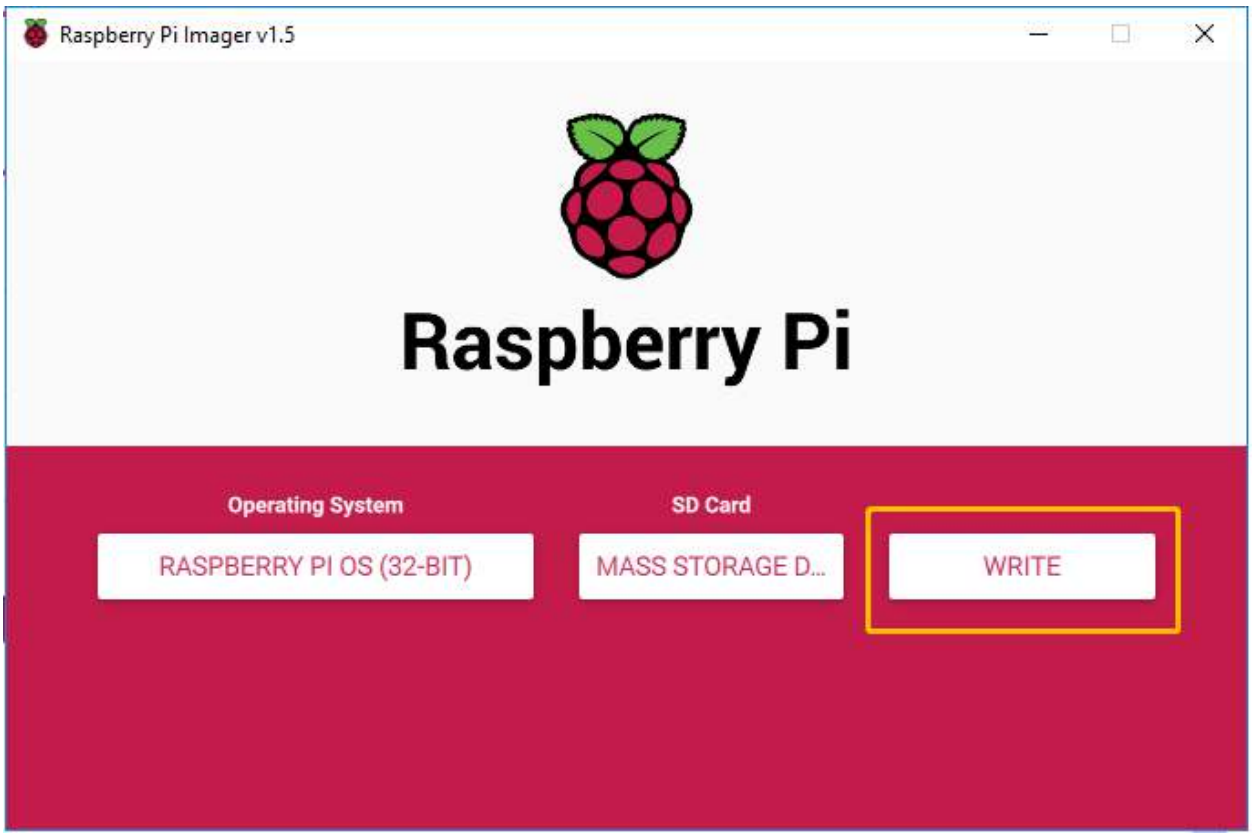
Note: **wifi country** should be set the two-letter ISO/IEC alpha2 code for the country in which you are using your Raspberry Pi, please refer to the following link: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements

---



Step 7

Click the **WRITE** button.

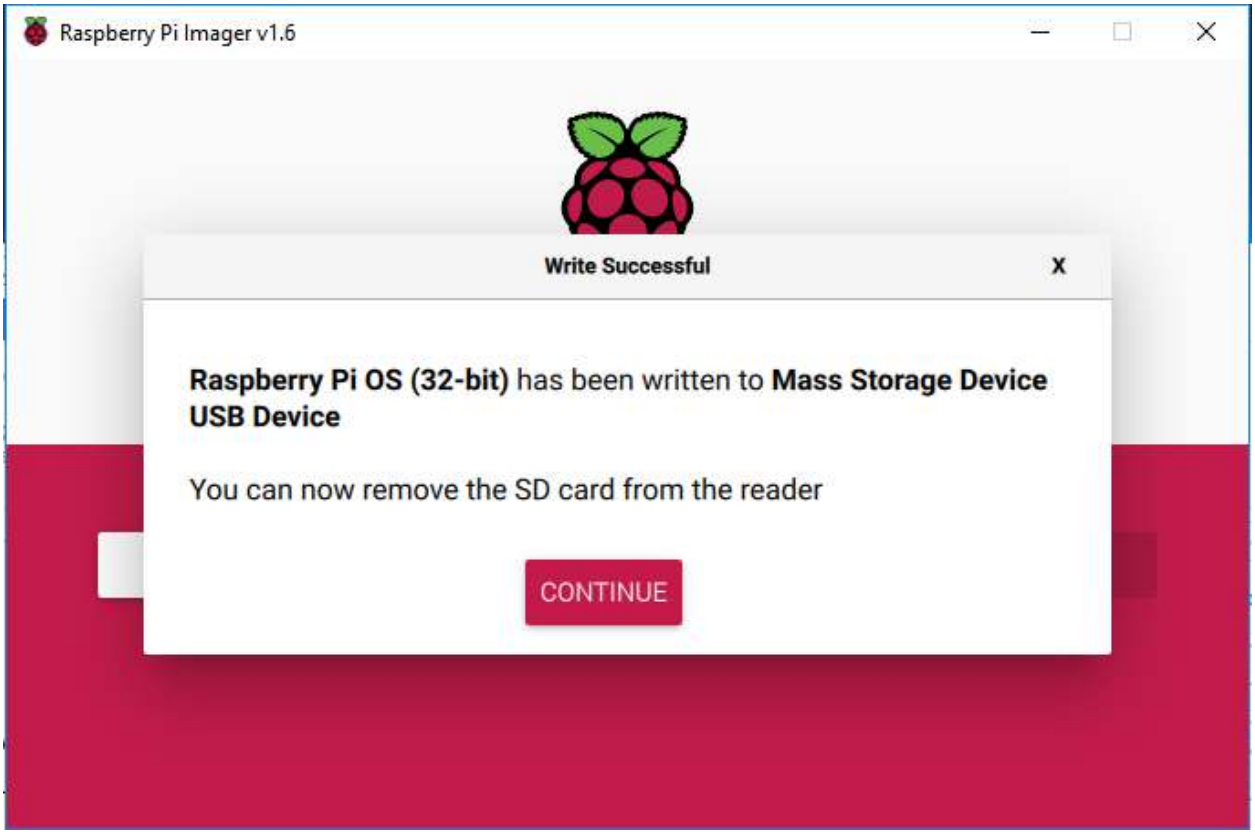


Step 8

If your SD card currently has any files on it, you may wish to back up these files first to prevent you from permanently losing them. If there is no file to be backed up, click **Yes**.

Step 9

After waiting for a period of time, the following window will appear to represent the completion of writing.



3.3 Set up Your Raspberry Pi

3.3.1 If You Have a Screen

If you have a screen, it will be easy for you to operate on the Raspberry Pi.

Required Components

Any Raspberry Pi	1 * Power Adapter
1 * Micro SD card	1 * Screen Power Adapter
1 * HDMI cable	1 * Screen
1 * Mouse	1 * Keyboard

1. Insert the SD card you've set up with Raspberry Pi OS into the micro SD card slot on the underside of your Raspberry Pi.
2. Plug in the Mouse and Keyboard.
3. Connect the screen to Raspberry Pi's HDMI port and make sure your screen is plugged into a wall socket and switched on.

> **Note:** If you use a Raspberry Pi 4, you need to connect the screen to the HDMI0 (nearest the power in port).

4. Use the power adapter to power the Raspberry Pi. After a few seconds, the Raspberry Pi OS desktop will be displayed.

3.3.2 If You Have No Screen

If you don't have a display, you can log in to the Raspberry Pi remotely, but before that, you need to get the IP of the Raspberry Pi.

Get the IP Address

After the Raspberry Pi is connected to WIFI, we need to get the IP address of it. There are many ways to know the IP address, and two of them are listed as follows.

1. Checking via the router

If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the Raspberry Pi OS is **raspberrypi**, and you need to find it. (If you are using ArchLinuxARM system, please find alarmpi.)

2. Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, **Advanced IP scanner** and so on.

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspberry Pi OS is **raspberrypi**, if you haven't modified it.

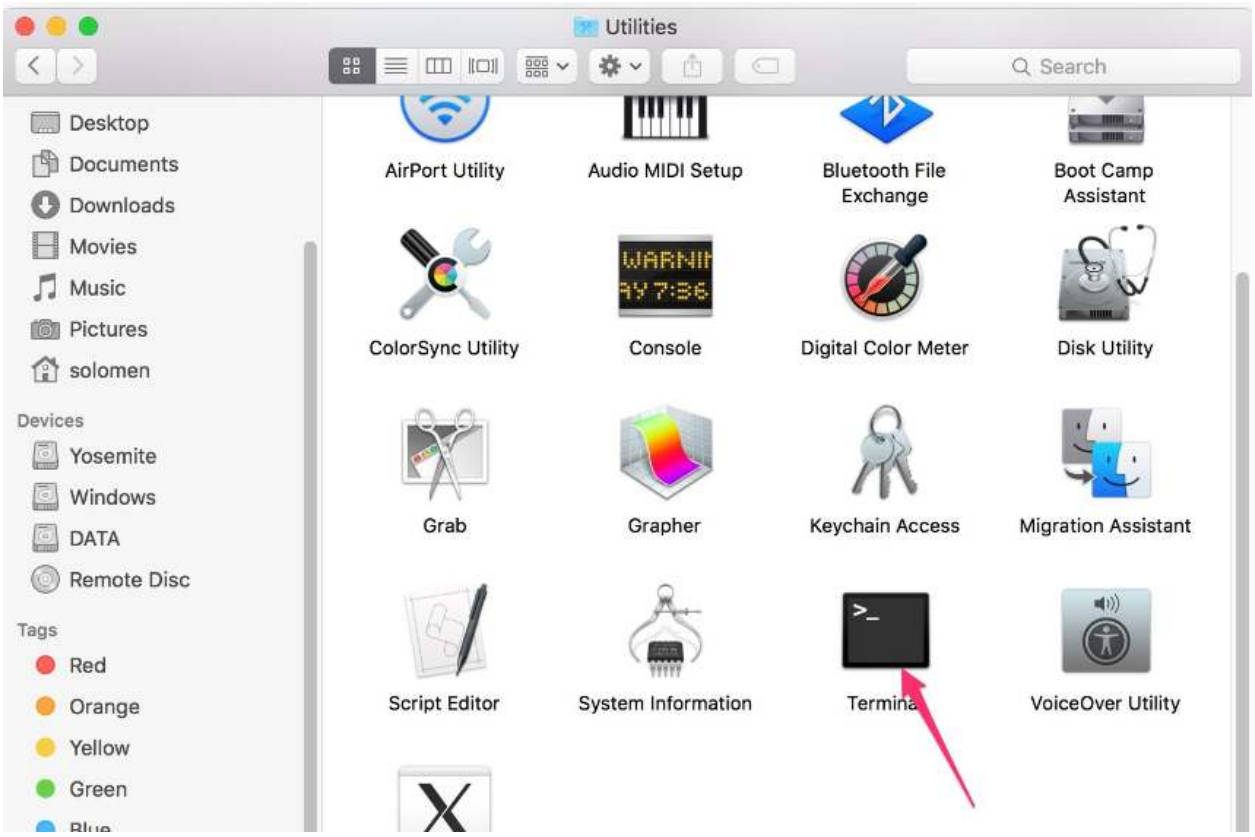
Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

For Linux or/Mac OS X Users

Step 1

Go to **Applications->Utilities**, find the **Terminal**, and open it.



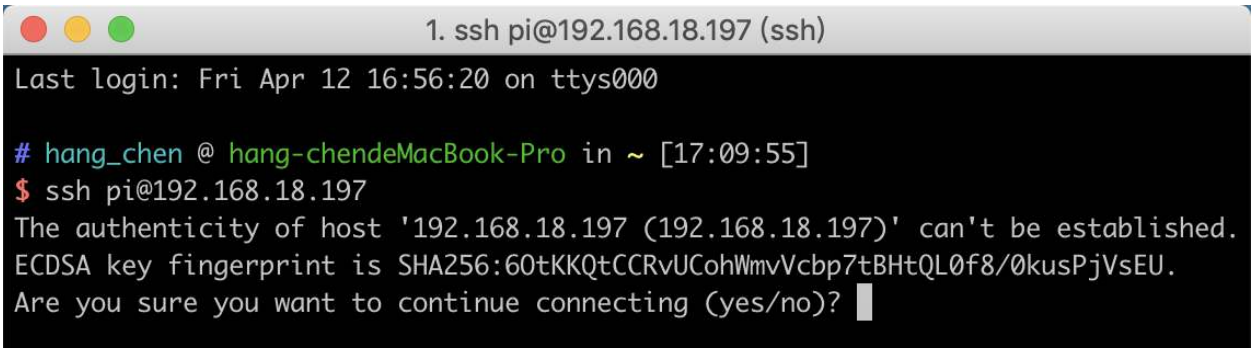
Step 2

Type in **ssh pi@ip_address** . “pi”is your username and “ip_address” is your IP address. For example:

```
ssh pi@192.168.18.197
```

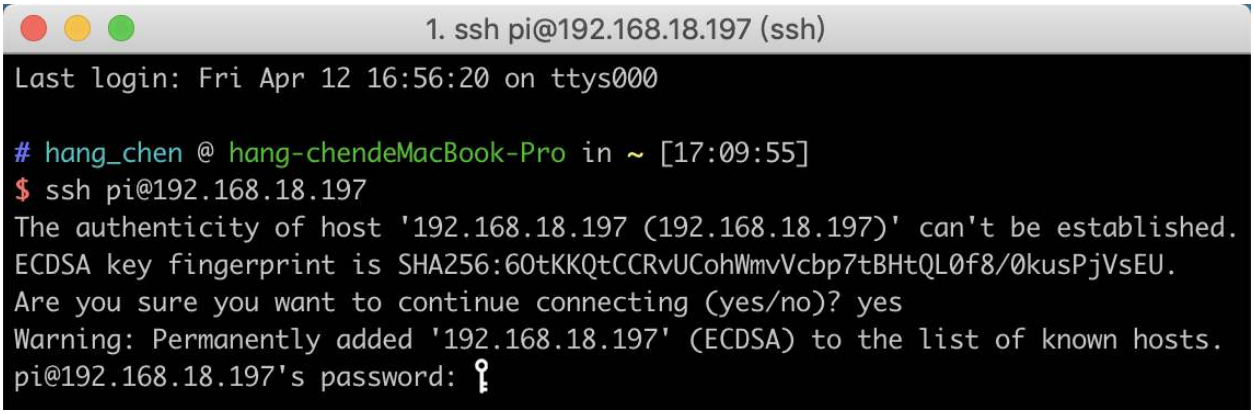
Step 3

Input”yes”.



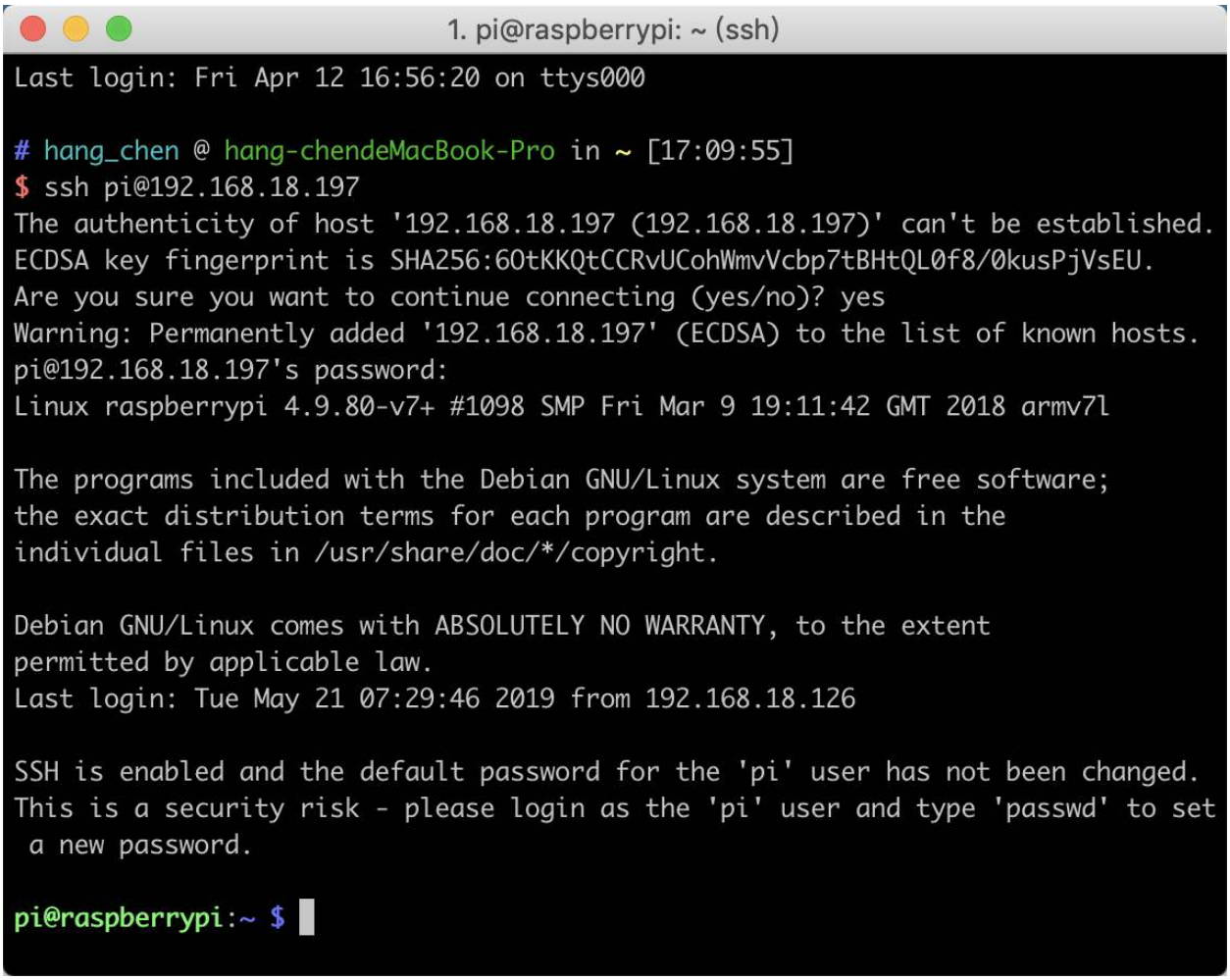
Step 4

Input the passcode and the default password is **raspberry**.



Step 5

We now get the Raspberry Pi connected and are ready to go to the next step.



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

For Windows Users

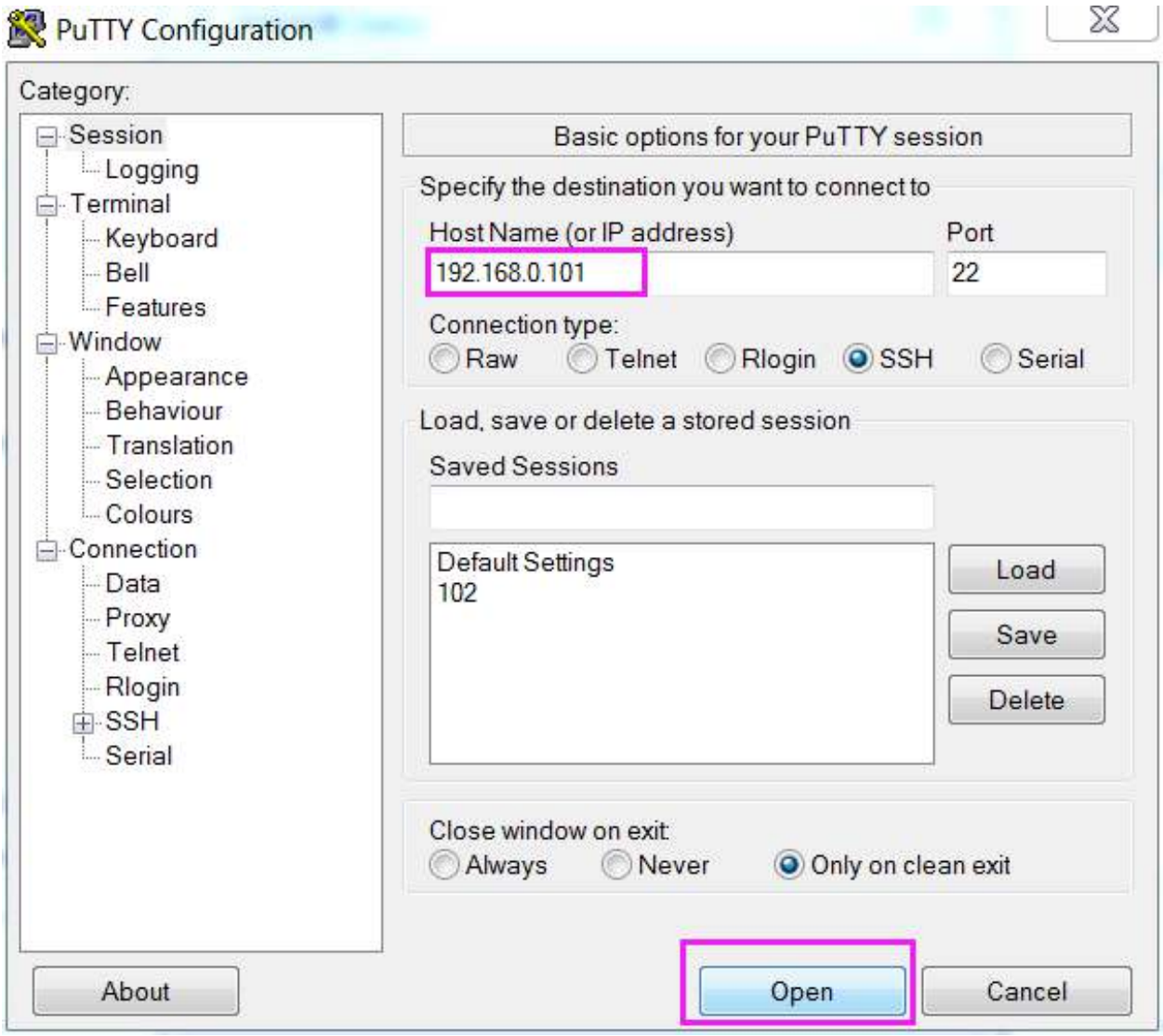
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

Step 1

Download PuTTY.

Step 2

Open PuTTY and click **Session** on the left tree-alike structure. Enter the IP address of the RPi in the text box under **Host Name (or IP address)** and **22** under **Port** (by default it is 22).



Step 3

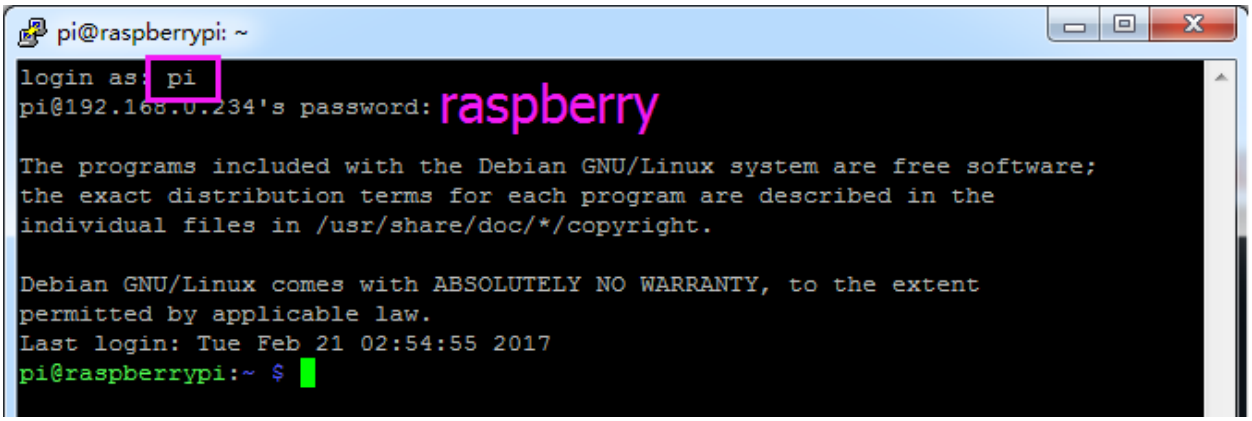
Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, there prompts a security reminder. Just click **Yes**.

Step 4

When the PuTTY window prompts "**login as:"**, type in "**pi"**(the user name of the RPi), and **password: "**raspberry" (the default one, if you haven't changed it).

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

If inactive appears next to PuTTY, it means that the connection has been broken and needs to be reconnected.



Step 5

Here, we get the Raspberry Pi connected and it is time to conduct the next steps.

Note: If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily.

For details on how to do this, please refer to *Remote Desktop*.

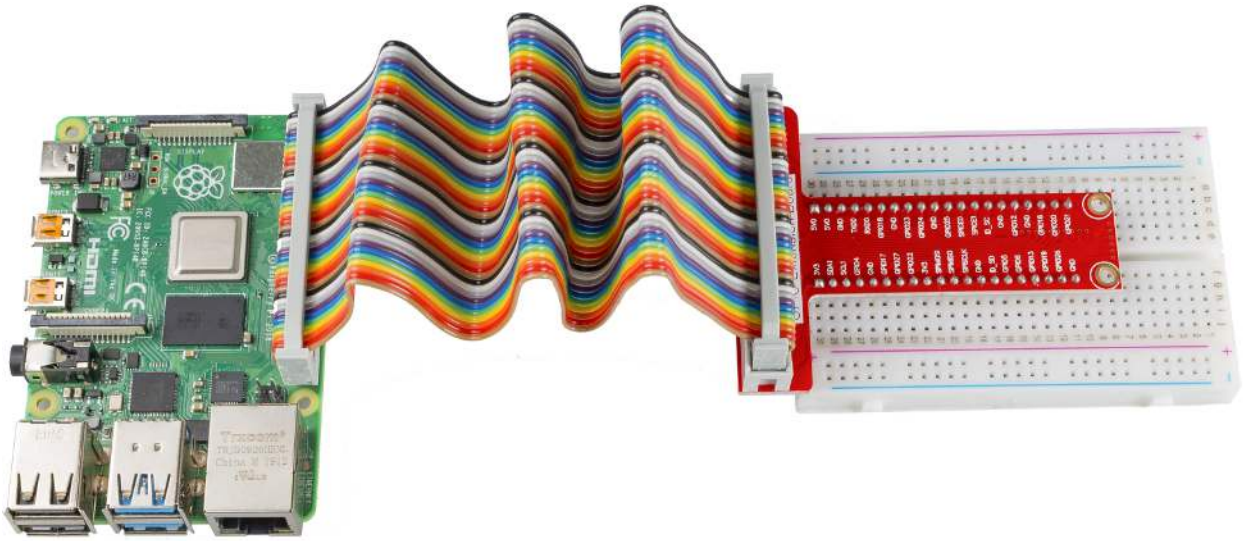
CHAPTER

FOUR

GPIO EXTENSION BOARD

Before starting to learn the commands, you first need to know more about the pins of the Raspberry Pi, which is key to the subsequent study.

We can easily lead out pins of the Raspberry Pi to breadboard by GPIO Extension Board to avoid GPIO damage caused by frequent plugging in or out. This is our 40-pin GPIO Extension Board and GPIO cable for Raspberry Pi model B+, 2 model B and 3, 4 model B.



Pin Number

The pins of Raspberry Pi have three kinds of ways to name and they are wiringPi, BCM and Board.

Among these naming methods, 40-pin GPIO Extension board uses the naming method, BCM. But for some special pins, such as I2C port and SPI port, they use the Name that comes with themselves.

The following table shows us the naming methods of WiringPi, Board and the intrinsic Name of each pin on GPIO Extension board. For example, for the GPIO17, the Board naming method of it is 11, the wiringPi naming method is 0, and the intrinsic naming method of it is GPIO0.

Note: 1In C Language, what is used is the naming method WiringPi.

2In Python Language, the applied naming methods are **Board** and **BCM**, and the function `GPIO.setmode()` is used to set them.

3In Scratch 3 and Processing, the applied naming method is **BCM**.

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

CHAPTER

FIVE

DOWNLOAD THE CODE

Before you download the code, please note that the example code is **ONLY** test on Raspberry Pi OS. We provide two methods for download:

Method 1: Use git clone (Recommended)

Log into Raspberry Pi and then change directory to `/home/pi`.

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

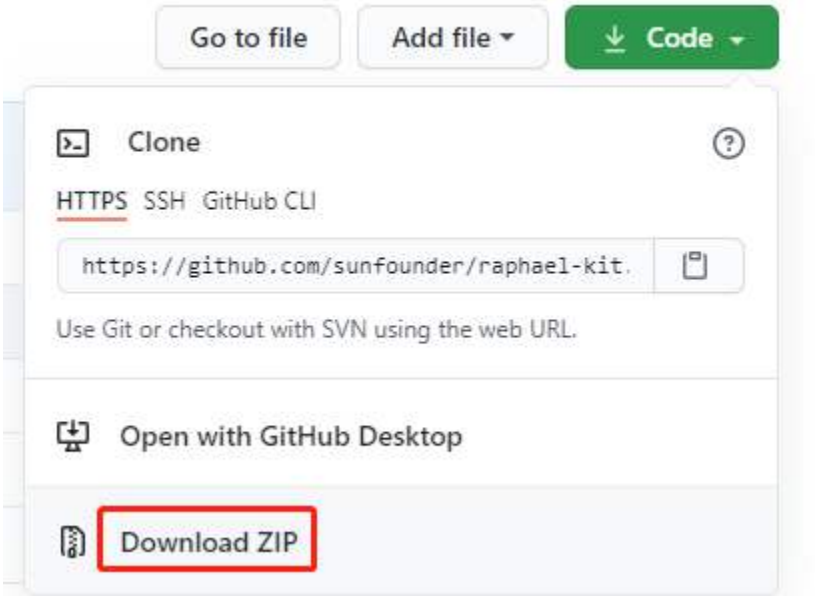
Note: cd to change to the intended directory from the current path. Informally, here is to go to the path `/home/pi/`.

Clone the repository from GitHub.

```
git clone https://github.com/sunfounder/raphael-kit.git
```

Method 2: Download the code

Download the source code from github: https://github.com/sunfounder/raphael-kit



CHAPTER

SIX

PLAY WITH PYTHON

6.1 Check the `RPi.GPIO`

If you are a Python user, you can program GPIOs with API provided by `RPi.GPIO`.

`RPi.GPIO` is a module to control Raspberry Pi GPIO channels. This package provides a class to control the GPIO on a Raspberry Pi. For examples and documents, visit: http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/.

Test whether RPi.GPIO is installed or not, type in python:

```
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.
>>>
```

In Python CLI, input `import RPi.GPIO`, If no error prompts, it means RPi.GPIO is installed.

```
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
>>>
```

If you want to quit python CLI, type in:

```
exit()
```

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

6.2 Output

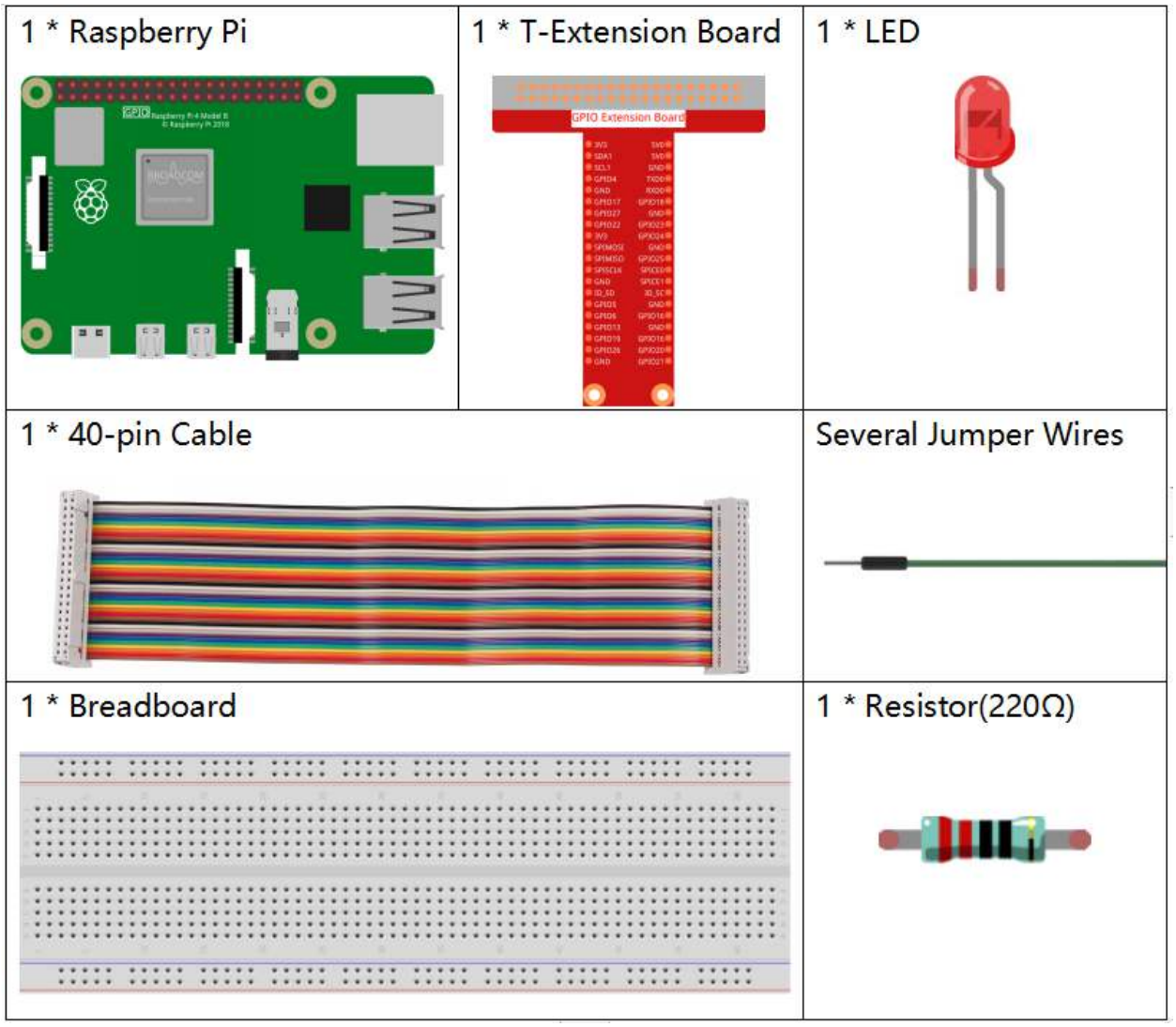
6.2.1 1.1 Displays

1.1.1 Blinking LED

Introduction

In this project, we will learn how to make a blinking LED by programming. Through your settings, your LED can produce a series of interesting phenomena. Now, go for it.

Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*

- *LED*

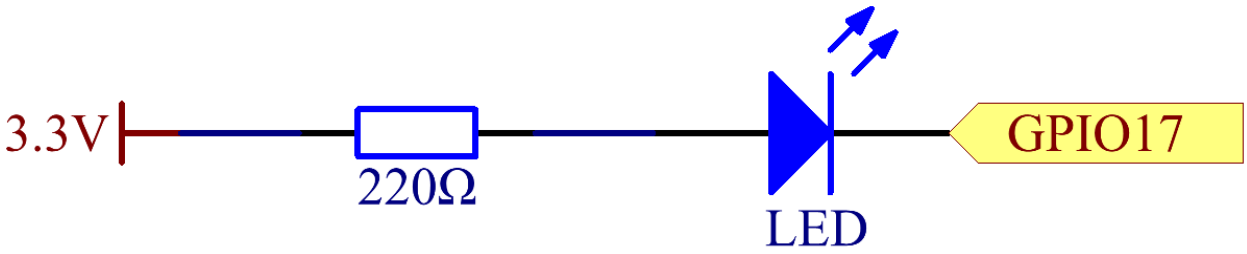
Schematic Diagram

In this experiment, connect a 220 resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to GPIO17 of Raspberry Pi. Therefore, to turn on an LED, we need to make GPIO17 low (0V) level. We can get this phenomenon by programming.

Note: **Pin11** refers to the 11th pin of the Raspberry Pi from left to right, and its corresponding **wiringPi** and **BCM** pin numbers are shown in the following table.

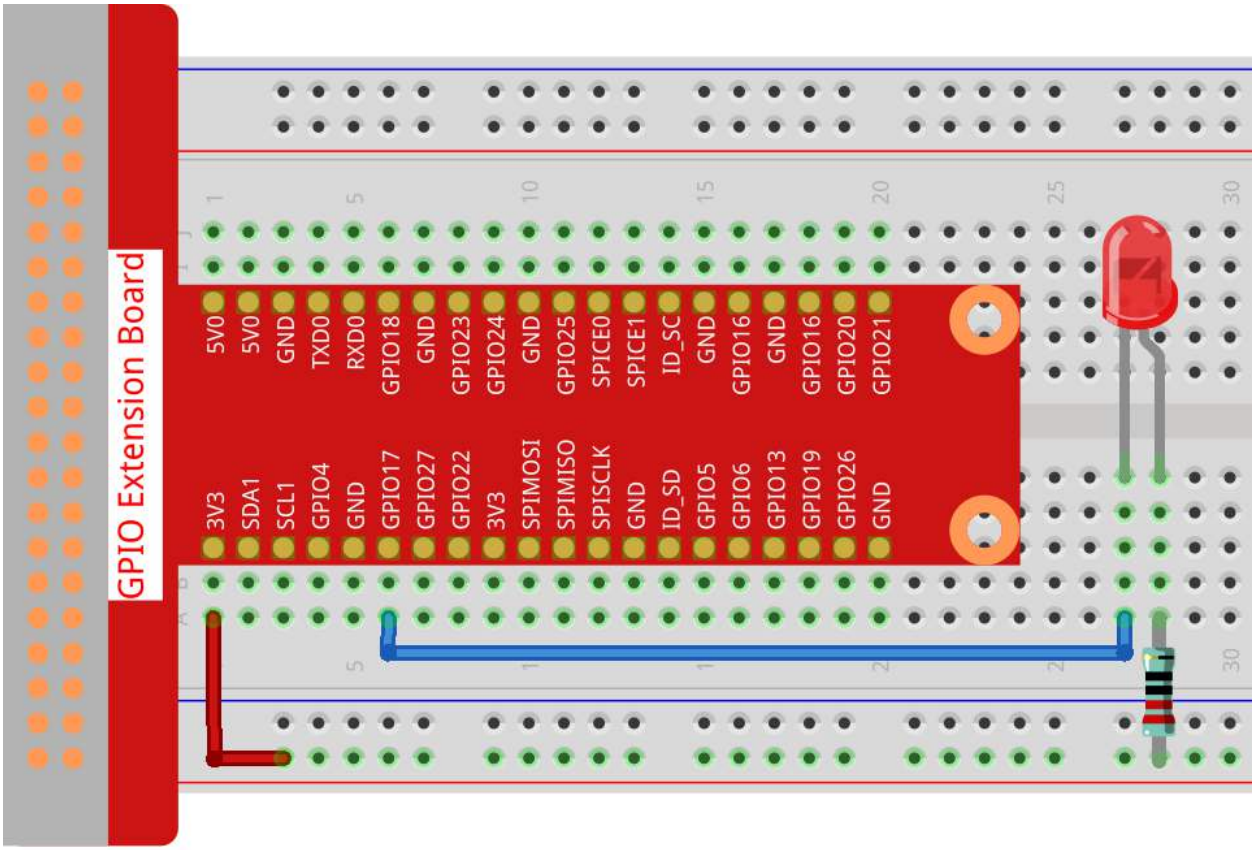
In the C language related content, we make GPIO0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi, Pin 11.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code and run it.

1. If you use a screen, you're recommended to take the following steps.

Find 1.1.1_BlinkingLed.py and double click it to open. Now you're in the file.

Click **Run ->Run Module** in the window and the following contents will appear.

To stop it from running, just click the X button on the top right to close it and then you'll back to the code. If you modify the code, before clicking **Run Module (F5)** you need to save it first. Then you can see the results.

2. If you log into the Raspberry Pi remotely, type in the command:

```
cd /home/pi/raphael-kit/python
```

Note: Change directory to the path of the code in this experiment via `cd`.

Step 3: Run the code

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

Note: Here sudo - superuser do, and python means to run the file by Python.

After the code runs, you will see the LED flashing.

Step 4: If you want to edit the code file 1.1.1_BlinkingLed.py, press `Ctrl + C` to stop running the code. Then type the following command to open 1.1.1_BlinkingLed.py:

```
nano 1.1.1_BlinkingLed.py
```

Note: nano is a text editor tool. The command is used to open the code file 1.1.1_BlinkingLed.py by this tool.

Press `Ctrl+X` to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in `Y` (save) or `N` (don't save).

Then press `Enter` to exit. Type in nano 1.1.1_BlinkingLed.py again to see the effect after the change.

Code

The following is the program code:

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
LedPin = 17
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
# Define a main function for main process
def main():
    while True:
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)
        print ('LED OFF...')
        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)
# Define a destroy function for clean up everything after the script finished
def destroy():
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()
# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program destroy() will be  executed.
    except KeyboardInterrupt:
        destroy()
```

Code Explanation

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

When the system detects this, it will search the installation path of python in the env setting, then call the corresponding interpreter to complete the operation. It's to prevent the user not installing the python onto the `/usr/bin` default path.

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

In this way, import the RPi.GPIO library, then define a variable, GPIO to replace RPI.GPIO in the following code.

```
import time
```

Import time package, for time delay function in the following program.

```
LedPin = 17
```

LED connects to the GPIO17 of the T-shape extension board, namely, BCM 17.

```
def setup():
    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).

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO: BOARD numbers and BCM numbers. In our projects, what we use is BCM numbers. You need to set up every channel you are using as an input or an output.

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

Set GPIO17(BCM17) as 0V (low level). Since the cathode of LED is connected to GPIO17, thus the LED will light up.

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

Delay for 0.5 second. Here, the statement is delay function in C language, the unit is second.

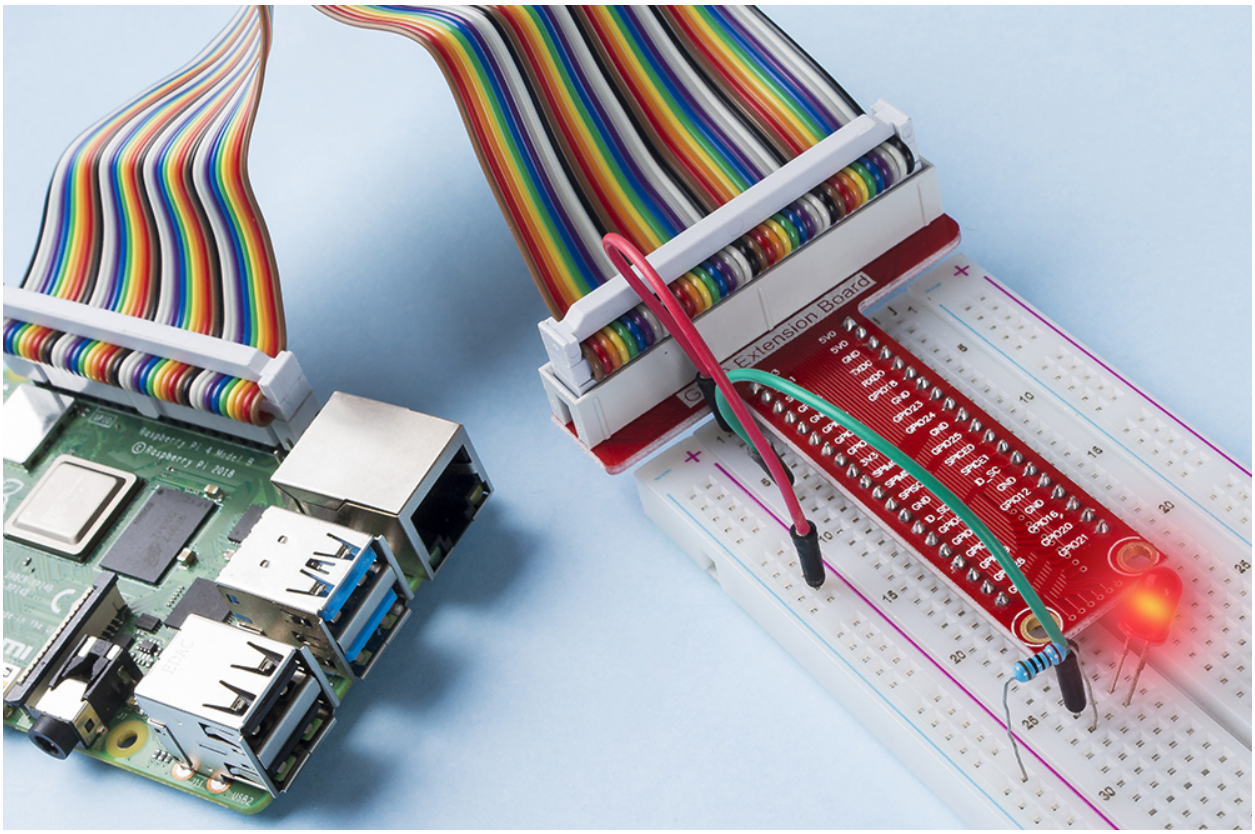
```
def destroy():
    GPIO.cleanup()
```

Define a destroy function for clean up everything after the script finished.

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

This is the general running structure of the code. When the program starts to run, it initializes the pin by running the setup(), and then runs the code in the main() function to set the pin to high and low levels. When `Ctrl+C` is pressed, the program, destroy() will be executed.

Phenomenon Picture

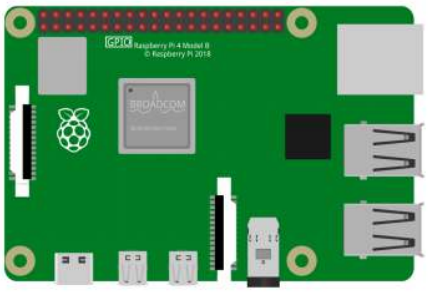
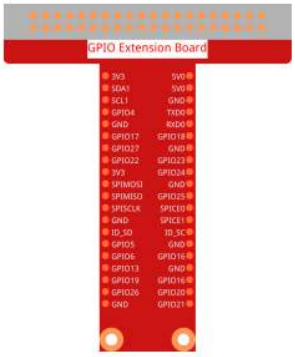
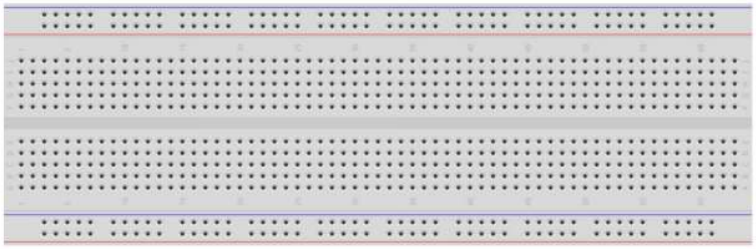


1.1.2 RGB LED

Introduction

In this project, we will control an RGB LED to flash various colors.

Components

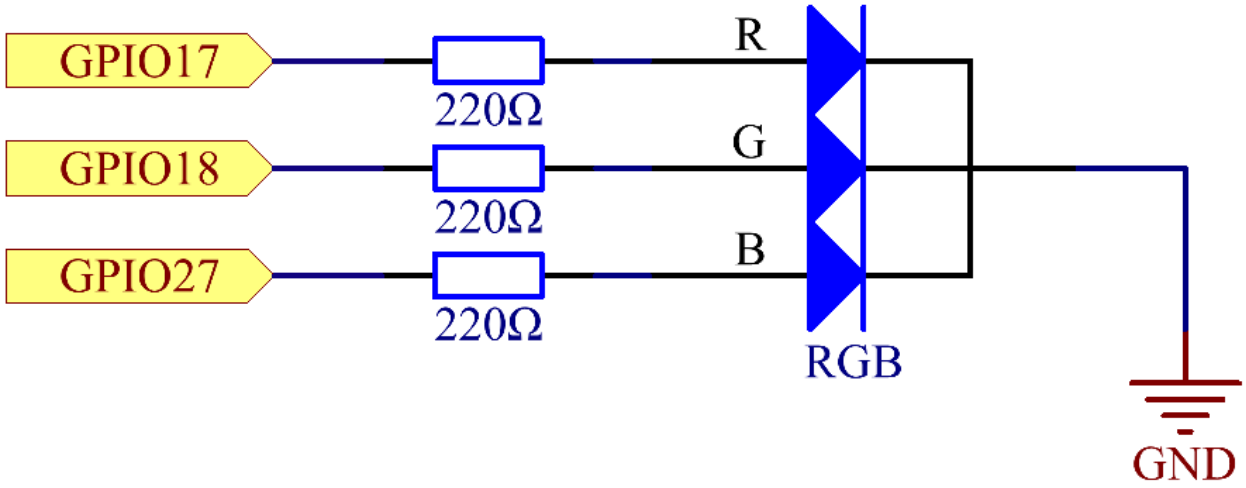
1 * Raspberry Pi 	1 * T-Extension Board 	1 * RGB LED
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Resistor(220Ω)

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*

Schematic Diagram

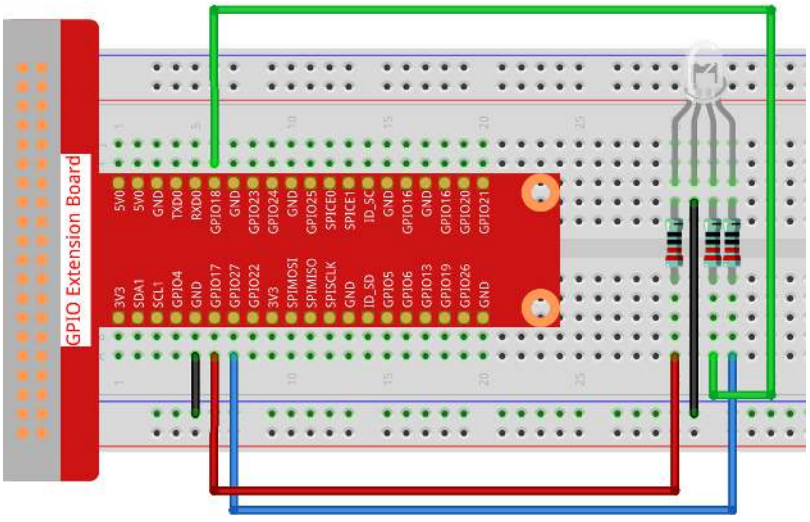
After connecting the pins of R, G, and B to a current limiting resistor, connect them to the GPIO17, GPIO18, and GPIO27 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

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

After the code runs, you will see that RGB displays red, green, blue, yellow, pink, and cyan.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

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

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

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

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

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

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

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

def main():
```

(continues on next page)

(continued from previous page)

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

def destroy():
    # Stop all pwm channel
    p_R.stop()
    p_G.stop()
    p_B.stop()
    # Release resource
    GPIO.cleanup()

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

Code Explanation

```
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)
```

Call the `GPIO.PWM()` function to define Red, Green and Blue as PWM pins and set the frequency of PWM pins to 2000Hz, then Use the `Start()` function to set the initial duty cycle to zero.

```
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 MAP function for mapping values. For instance, x=50, in_min=0, in_max=255, out_min=0, out_max=100. After the map function mapping, it returns `(50-0) \* (100-0)/(255-0) +0=19.6`, meaning that 50 in 0-255 equals 19.6 in 0-100.

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

Configures the three LEDs' luminance with the inputted color value, assign the first two values of the hexadecimal to R_val, the middle two assigned to G_val, the last two values to B_val. For instance, if color=0xFF00FF, 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)
```

Use map function to map the R,G,B value among 0~255 into PWM duty cycle range 0-100.

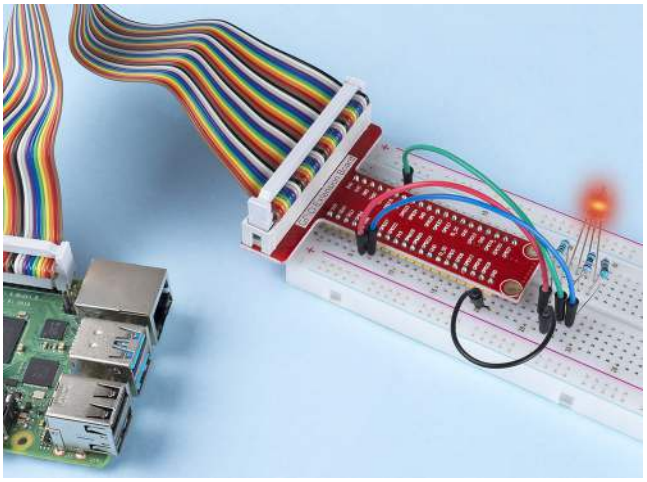
```
p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

Assign the mapped duty cycle value to the corresponding PWM channel to change the luminance.

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

Assign every item in the COLOR list to the color respectively and change the color of the RGB LED via the `setColor()` function.

Phenomenon Picture

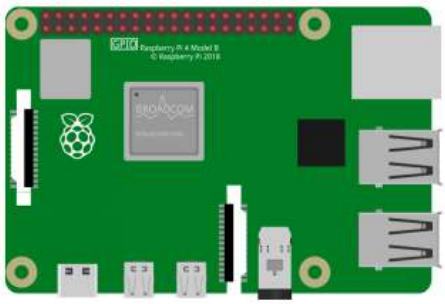
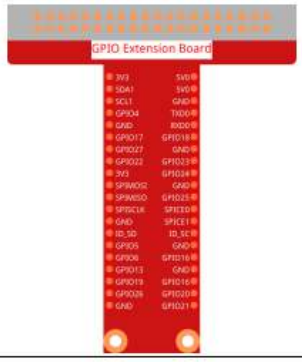
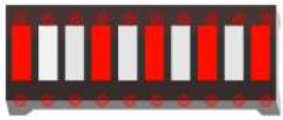
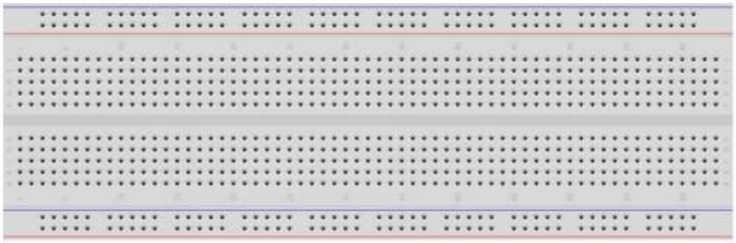


1.1.3 LED Bar Graph

Introduction

In this project, we sequentially illuminate the lights on the LED Bar Graph.

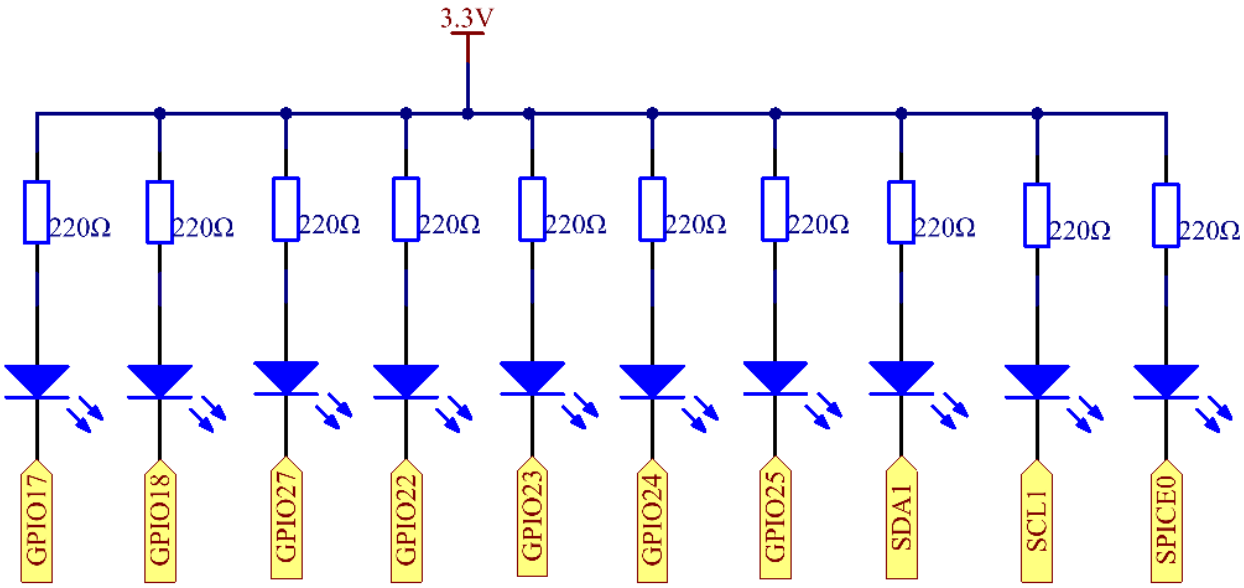
Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*

Schematic Diagram

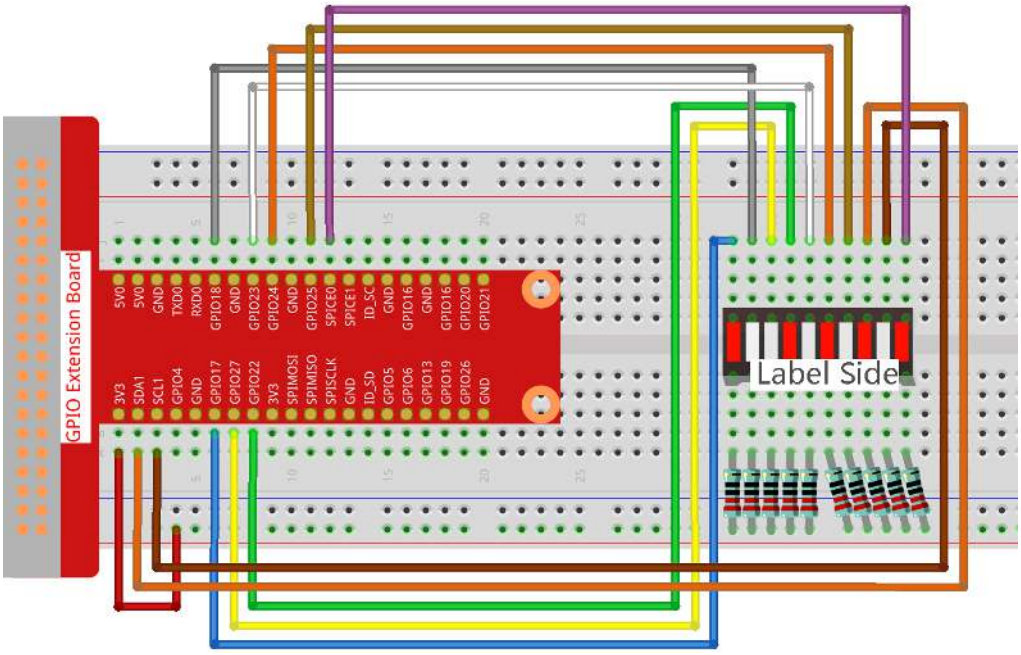
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3	8	2
SCL1	Pin 5	9	3
SPICE0	Pin 24	10	8



Experimental Procedures

Step 1: Build the circuit.

Note: Pay attention to the direction when connecting. If you connect it backwards, it will not light up.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

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

After the code runs, you will see the LEDs on the LED bar turn on and off regularly.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

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

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

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

(continues on next page)

(continued from previous page)

```
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 Explanation

ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24] Create an array and assign it to the pin number corresponding to the LED Bar Graph (11, 12, 13, 15, 16, 18, 22, 3, 5, 24) and the array will be used to control the LED.

```
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)
```

Let the LED on the odd digit of the LED Bar Graph light on in turn.

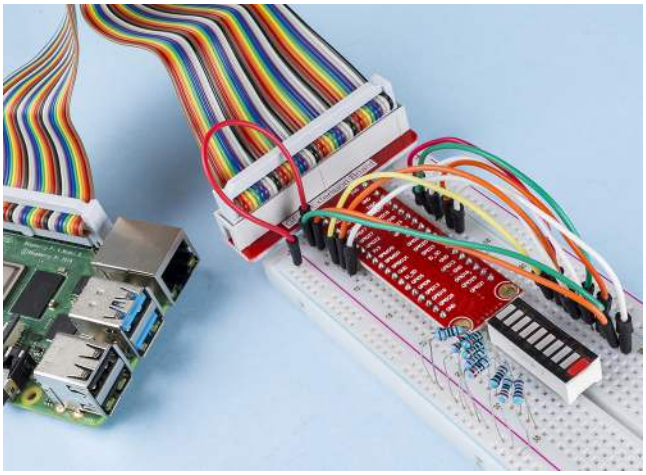
```
def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)
```

Make the LED on the even digit of the LED Bar Graph light on in turn.

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

Let the LED on the LED Bar Graph light on one by one.

Phenomenon Picture

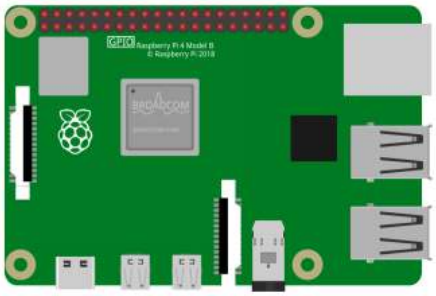
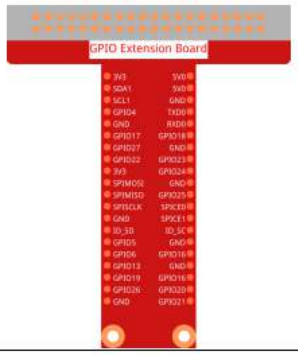
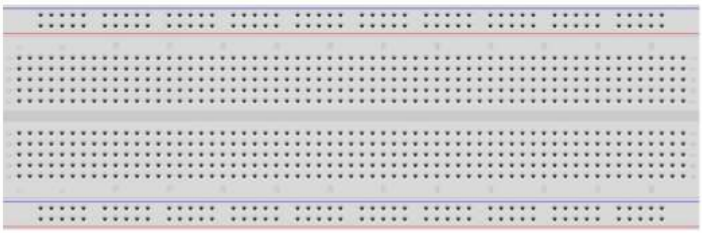


1.1.4 7-segment Display

Introduction

Let's try to drive a 7-segment display to show a figure from 0 to 9 and A to F.

Components

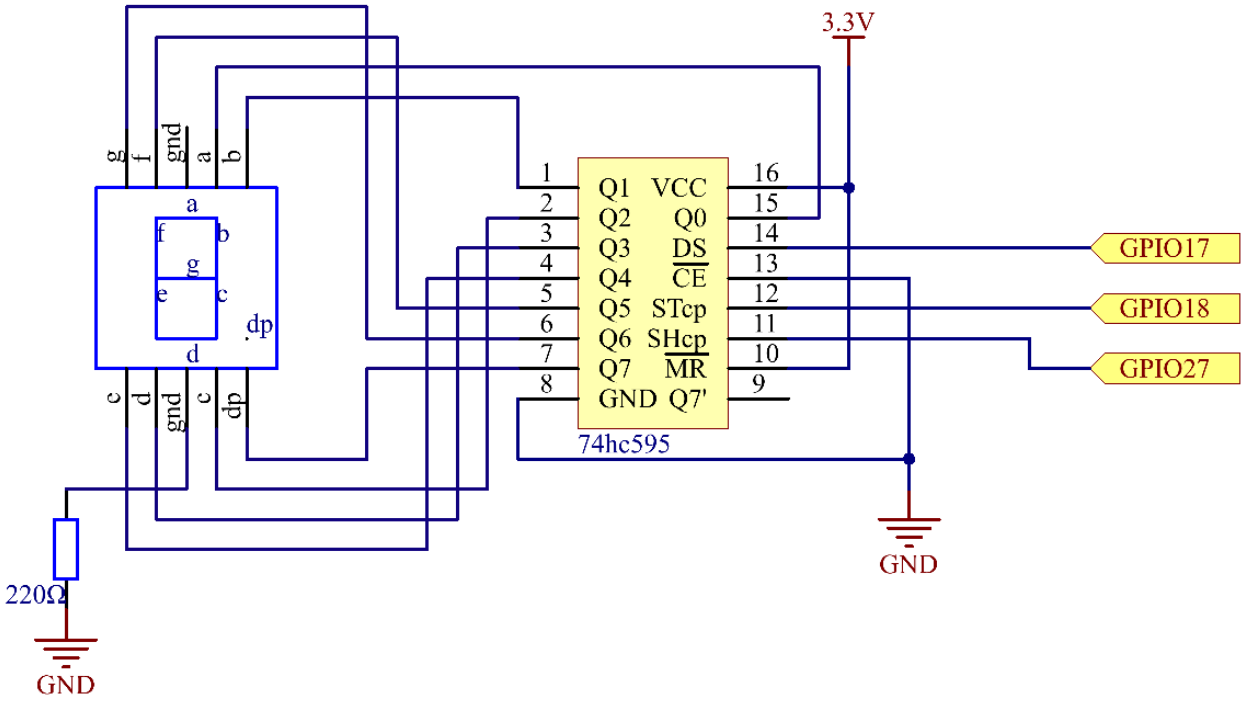
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 7-segment display
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * 74HC595
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *7-segment Display*
- *74HC595*

Schematic Diagram

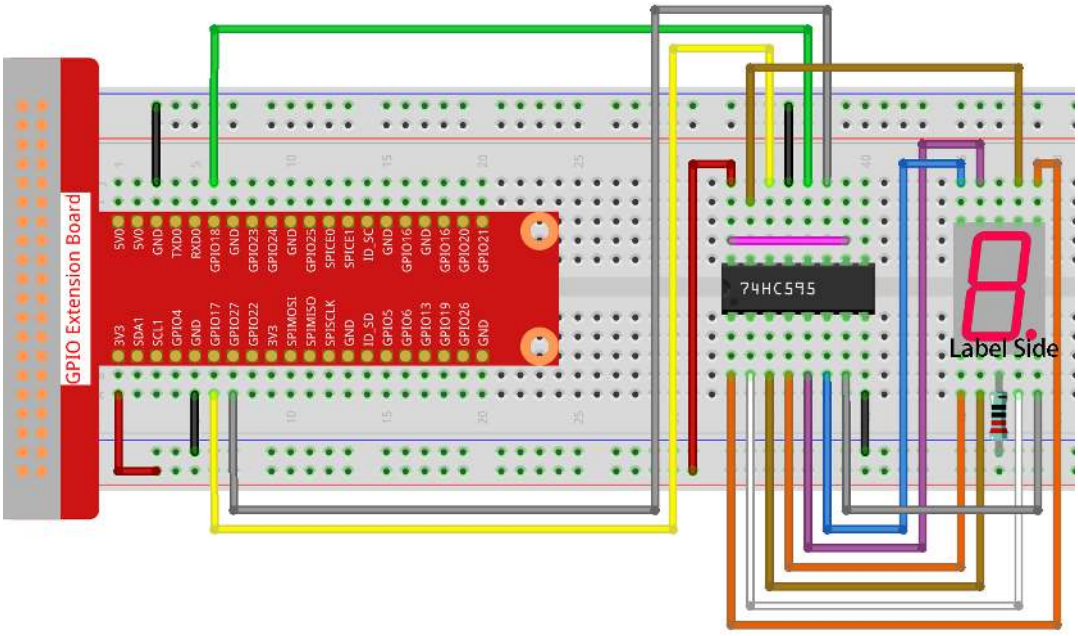
Connect pin ST_CP of 74HC595 to Raspberry Pi GPIO18, SH_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST_CP (the clock input of the memory) is at the rising edge. Then you can control the states of SH_CP and ST_CP via the Raspberry Pi GPIOs to transform serial data input into parallel data output so as to save Raspberry Pi GPIOs and drive the display.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

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

After the code runs, you'll see the 7-segment display display 0-9, A-F.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect. After confirming that there are no problems, you can use the Copy button to copy the modified code, then open the source code in Terminal via `nano` cammand and paste it.

```
import RPi.GPIO as GPIO
import time

# Set up pins
SDI   = 17
RCLK  = 18
SRCLK = 27

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

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

(continues on next page)

(continued from previous page)

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

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

def destroy():
    GPIO.cleanup()

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

Code Explanation

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

A segment code array from 0 to F in Hexadecimal (Common cathode).

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

Set ds, st_cp, sh_cp three pins to output and the initial state as low level.

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

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when bit=2, 0x3f will shift right(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true.

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

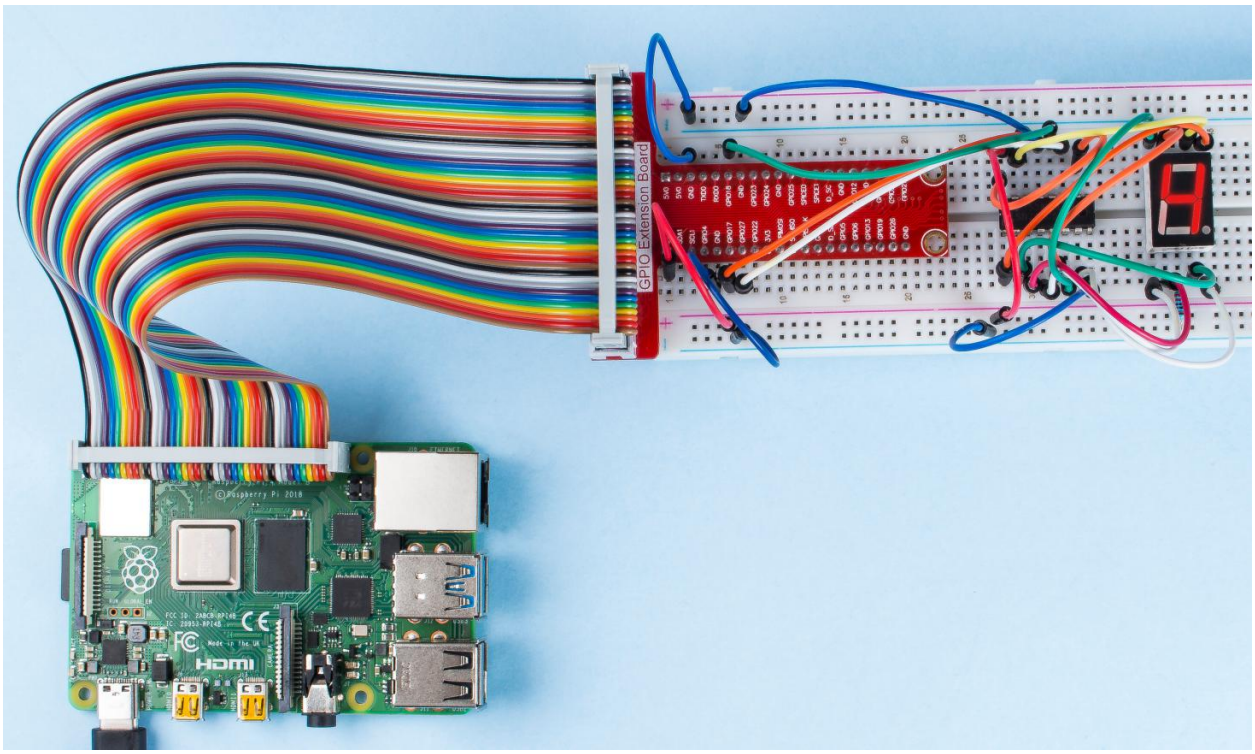
SRCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge pulse, then shift the DS date to shift register.

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

RCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge, then shift data from shift register to storage register.

Note: The hexadecimal format of number 0~15 are (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

Phenomenon Picture

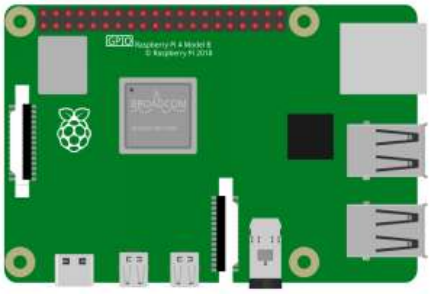
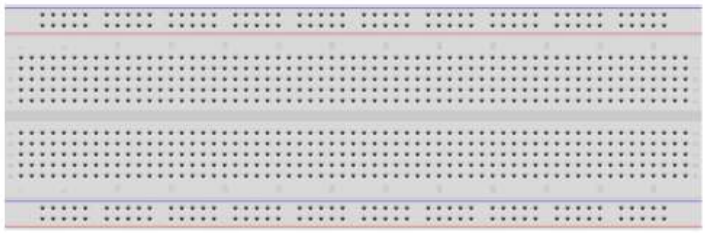


1.1.5 4-Digit 7-Segment Display

Introduction

Next, follow me to try to control the 4-digit 7-segment display.

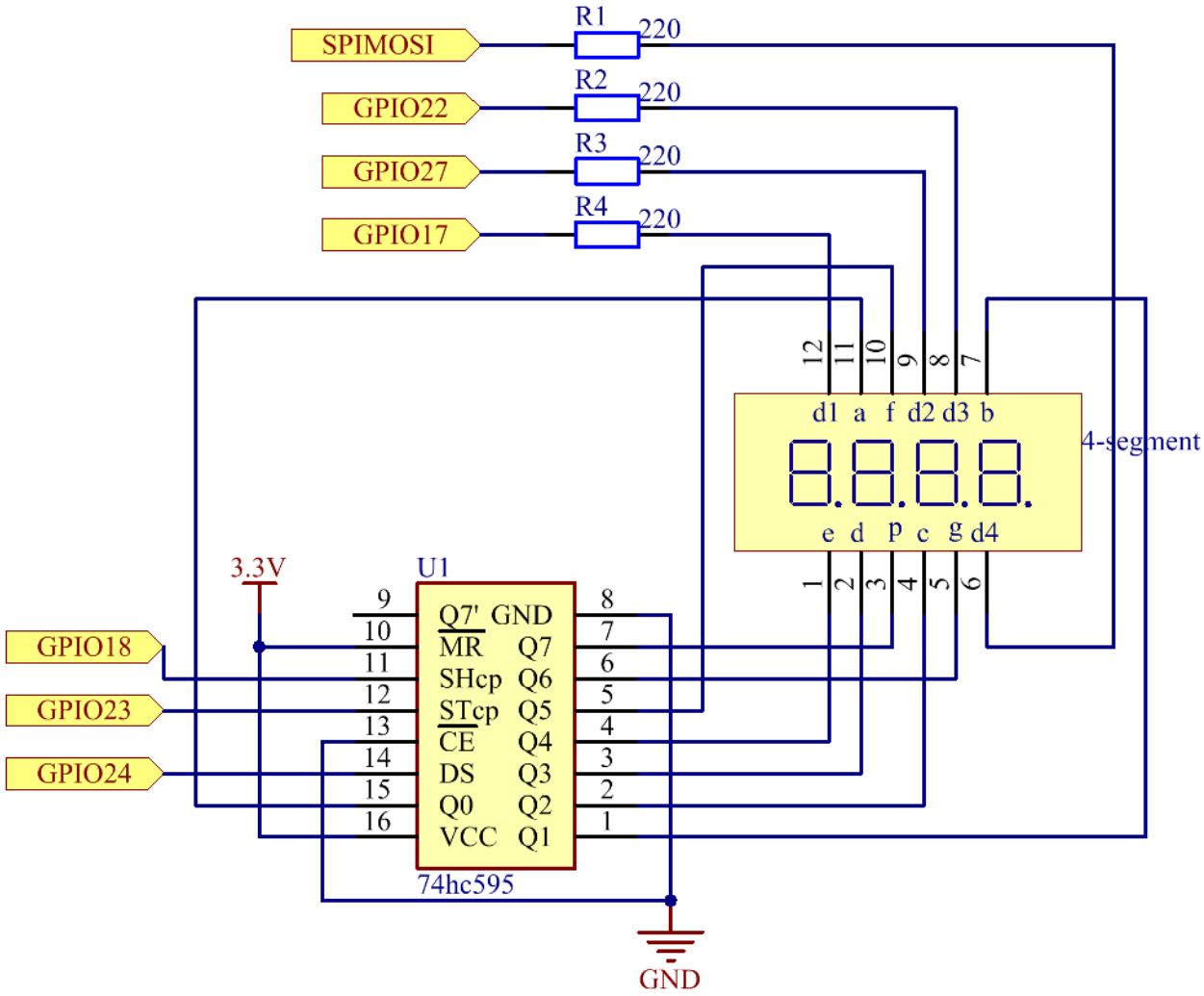
Components

<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>4 * Resistor(220Ω)
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>Several Jumper Wires
</td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*

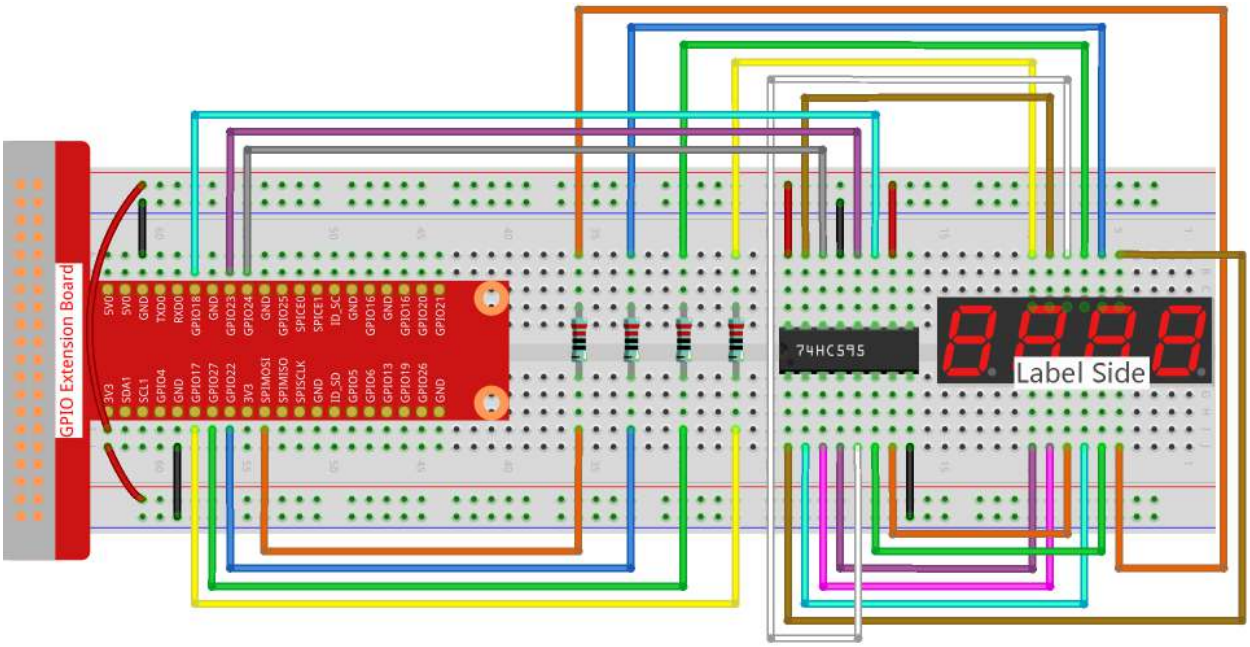
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

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

After the code runs, the program takes a count, increasing by 1 per second, and the 4 digit display displays the count.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
    timer1.start()

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

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

Code Explanation

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

These four pins control the common anode pins of the four-digit 7-segment displays.

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

A segment code array from 0 to 9 in hexadecimal (common 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)
```

Write “1” for eight times in SDI., so that the eight LEDs on the 7-segment Dispaly will turn off so as to clear the displayed content.

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

Select the place of the value. there is only one place that should be enable each time. The enabled place will be written high.

```
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)
```

(continues on next page)

(continued from previous page)

```
        hc595_shift(number[counter % 1000//100])

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

The function is used to set the number displayed on the 4-digit 7-segment Dispaly.

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

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

The module, threading is the common threading module in Pythonand Timer is the subclass of it. The prototype of code is:

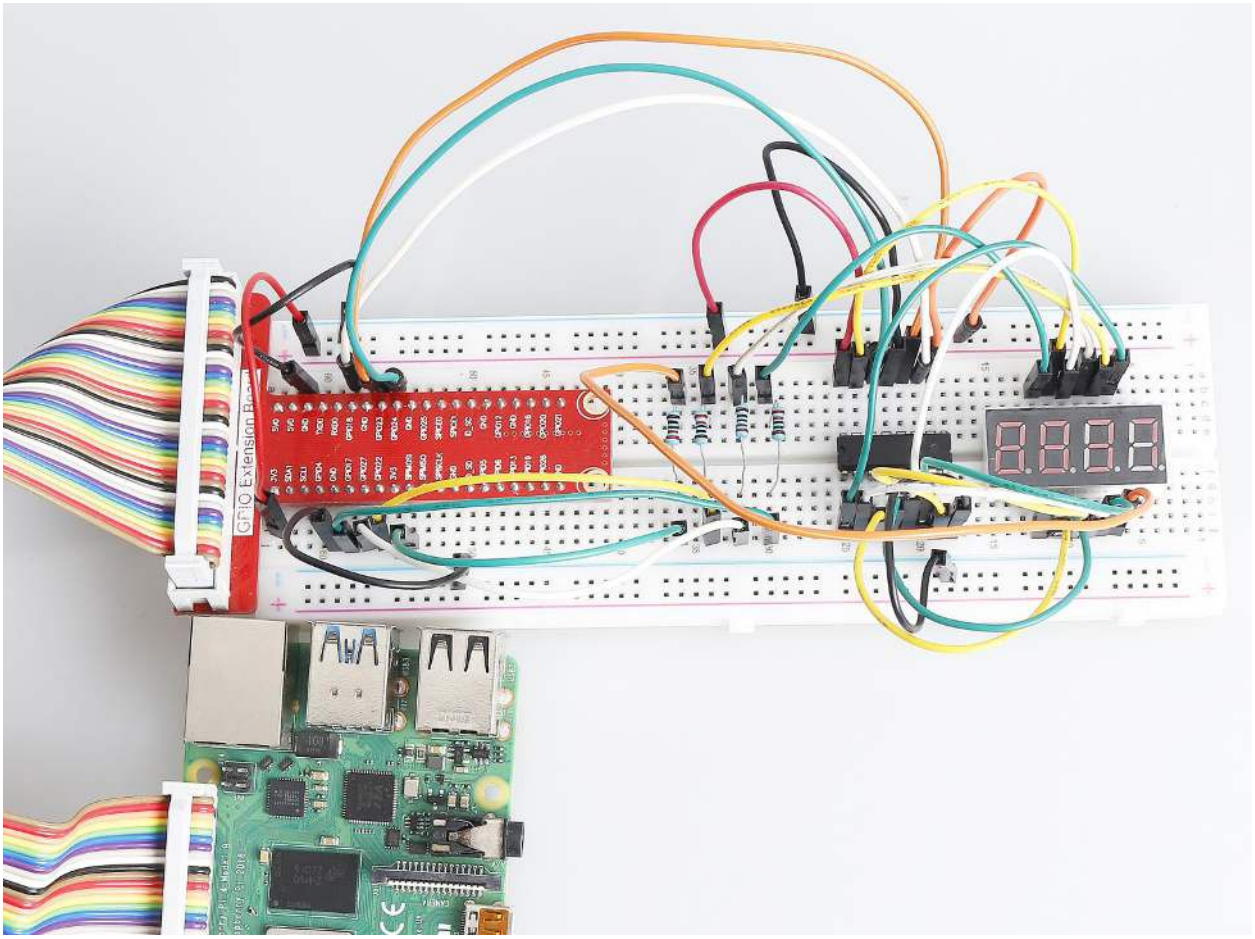
```
class threading.Timer(interval, function, args=[], kwargs={})
```

After the interval, the function will be run. Here, the interval is 1.0and the function is timer(). start () means the Timer will start at this point.

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

After Timer reaches 1.0s, the Timer function is called; add 1 to counter, and the Timer is used again to execute itself repeatedly every second.

Phenomenon Picture

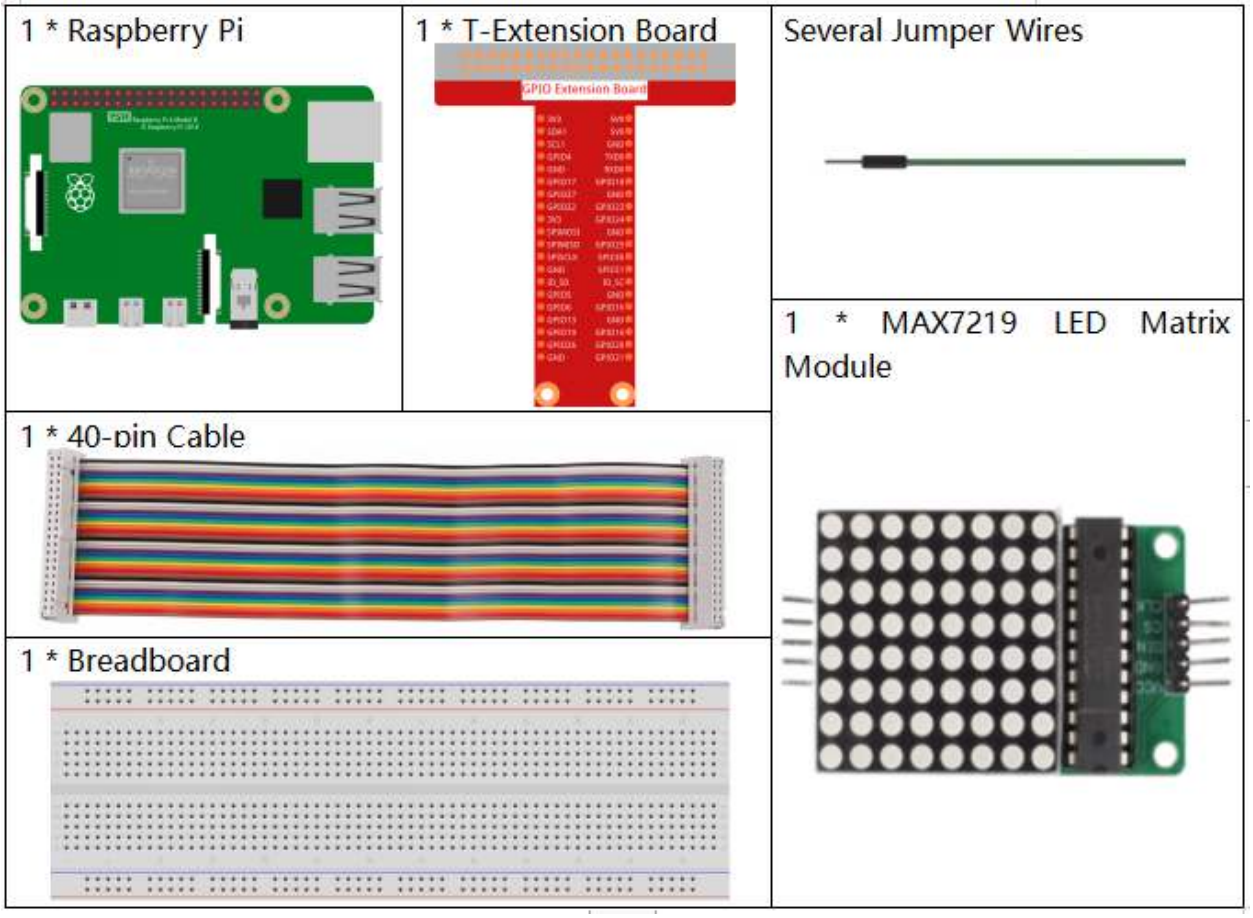


1.1.6 LED Dot Matrix

Introduction

As the name suggests, an LED dot matrix is a matrix composed of LEDs. The lighting up and dimming of the LEDs formulate different characters and patterns.

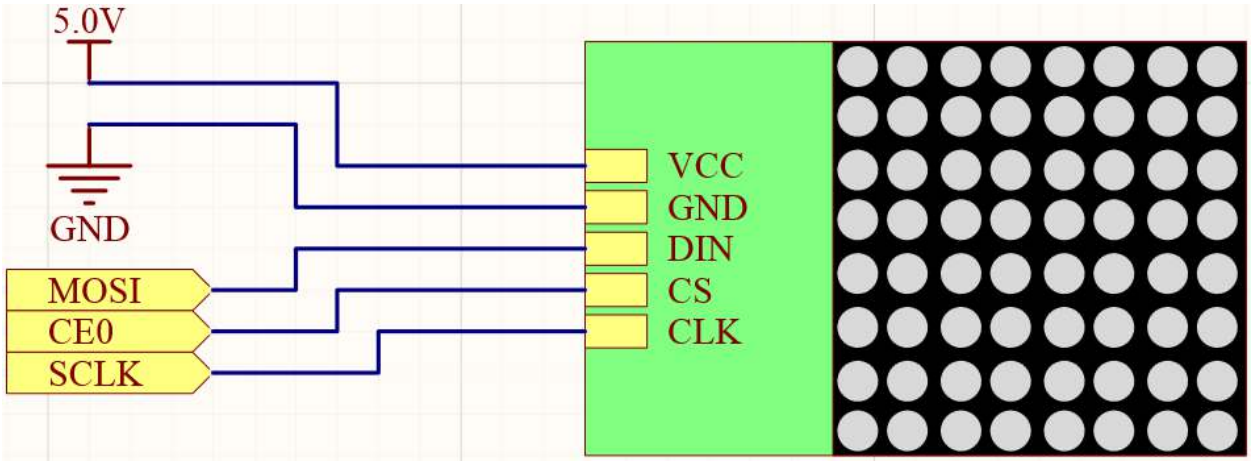
Components



- *GPIO Extension Board*
- *Breadboard*
- *LED Matrix Module*

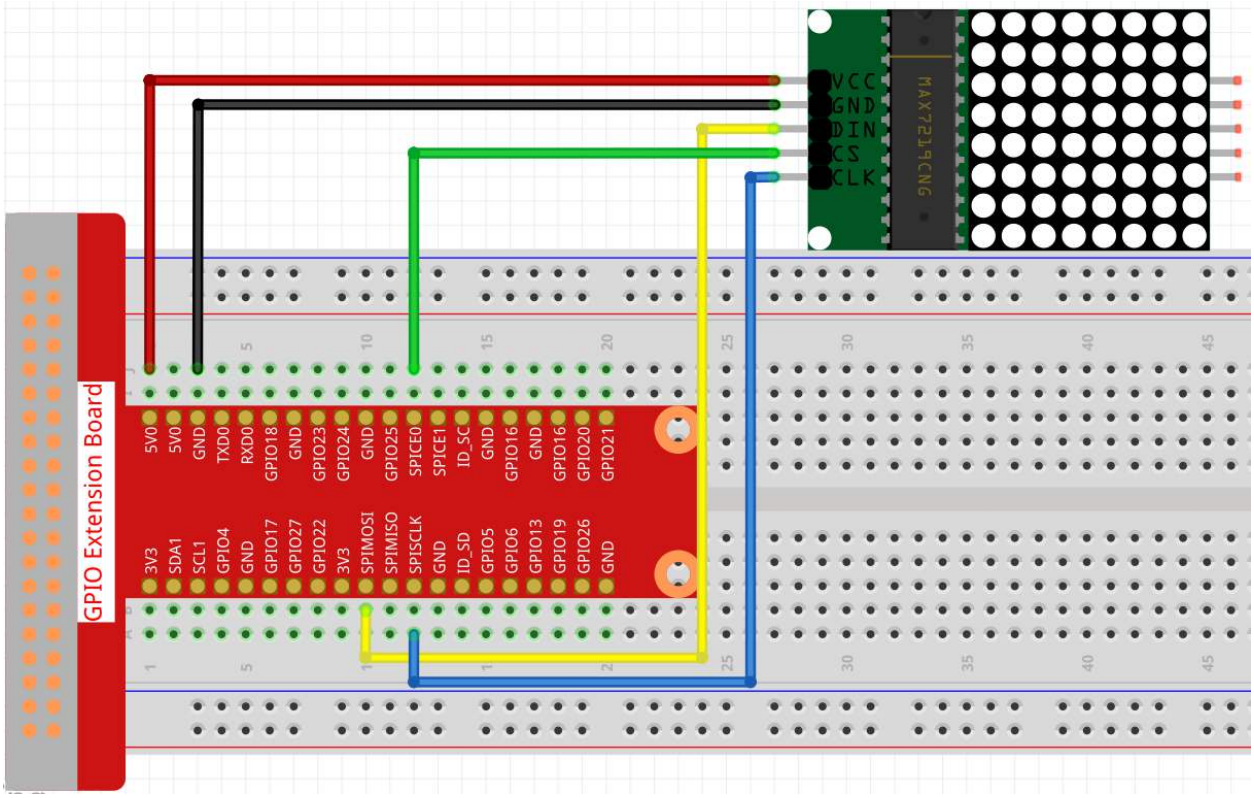
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
SPIMOSI	Pin 19	12	MOSI
SPICE0	pin 24	10	CE0
SPISCLK	Pin 23	14	SCLK



Experimental Procedures

Step 1: Build the circuit.



Note: Turn on the SPI before starting the experiment, refer to *SPI Configuration* for details. And the *Luma.LED_Matrix* module is also needed.

Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

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

After running the code, the LED Matrix will display a rectangle for two seconds, then the text 'A' for two seconds, and finally scroll to display the text "Hello, Nice to meet you!

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.led_matrix.device import max7219
from luma.core.legacy import text
from luma.core.legacy.font import proportional, CP437_FONT, LCD_FONT
import time

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, rotate=1)
virtual = viewport(device, width=200, height=400)

def displayRectangle():
    with canvas(device) as draw:
        draw.rectangle(device.bounding_box, outline="white", fill="black")

def displayLetter():
    with canvas(device) as draw:
        text(draw, (0, 0), "A", fill="white", font=proportional(CP437_FONT))

def scrollToDisplayText():
    with canvas(virtual) as draw:
        text(draw, (0, 0), "Hello, Nice to meet you!", fill="white",␣
↪font=proportional(CP437_FONT))

    for offset in range(150):
        virtual.set_position((offset,0))
        time.sleep(0.1)

def main():
    while True:
        displayRectangle()
        time.sleep(2)
        displayLetter()
        time.sleep(2)
        scrollToDisplayText()

def destroy():
    pass

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

Code Explanation

```
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.led_matrix.device import max7219
from luma.core.legacy import text
from luma.core.legacy.font import proportional, CP437_FONT, LCD_FONT
import time
```

Import the relevant libraries, of which `luma.core` is a component library that provides a Pillow-compatible canvas for Python 3, as well as other drawing primitives and text rendering features that support small displays on Raspberry Pi and other single-board computers. You can visit https://luma-core.readthedocs.io/en/latest/intro.html to learn more.

```
serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, rotate=1)
```

Initialize the `luma.led_matrix.device.max7219` class.

Note: If you want to modify the display direction of the LED Matrix, you can do so by modifying the value of rotate, where 0 means no rotation, 1 means 90° clockwise rotation, 2 means 180° rotation, and 3 means 270° clockwise rotation.

```
def displayRectangle():
    with canvas(device) as draw:
        draw.rectangle(device.bounding_box, outline="white", fill="black")
```

Display a hollow rectangle in the edge area of the LED Matrix and modify the value of `fill` to `white` to display a solid rectangle.

```
def displayLetter():
    with canvas(device) as draw:
        text(draw, (0, 0), "A", fill="white", font=proportional(CP437_FONT))
```

An “A” is displayed on the (0, 0) coordinate of the LED Matrix, where `CP437_FONT` is a font suitable for 8*8 dot matrix screens.

```
virtual = viewport(device, width=200, height=400)
```

There is no way to display a line of text in a single 8x8 LED matrix. We need to use the `luma.core.virtual.viewport` method so that the text can be scrolled through the virtual viewport.

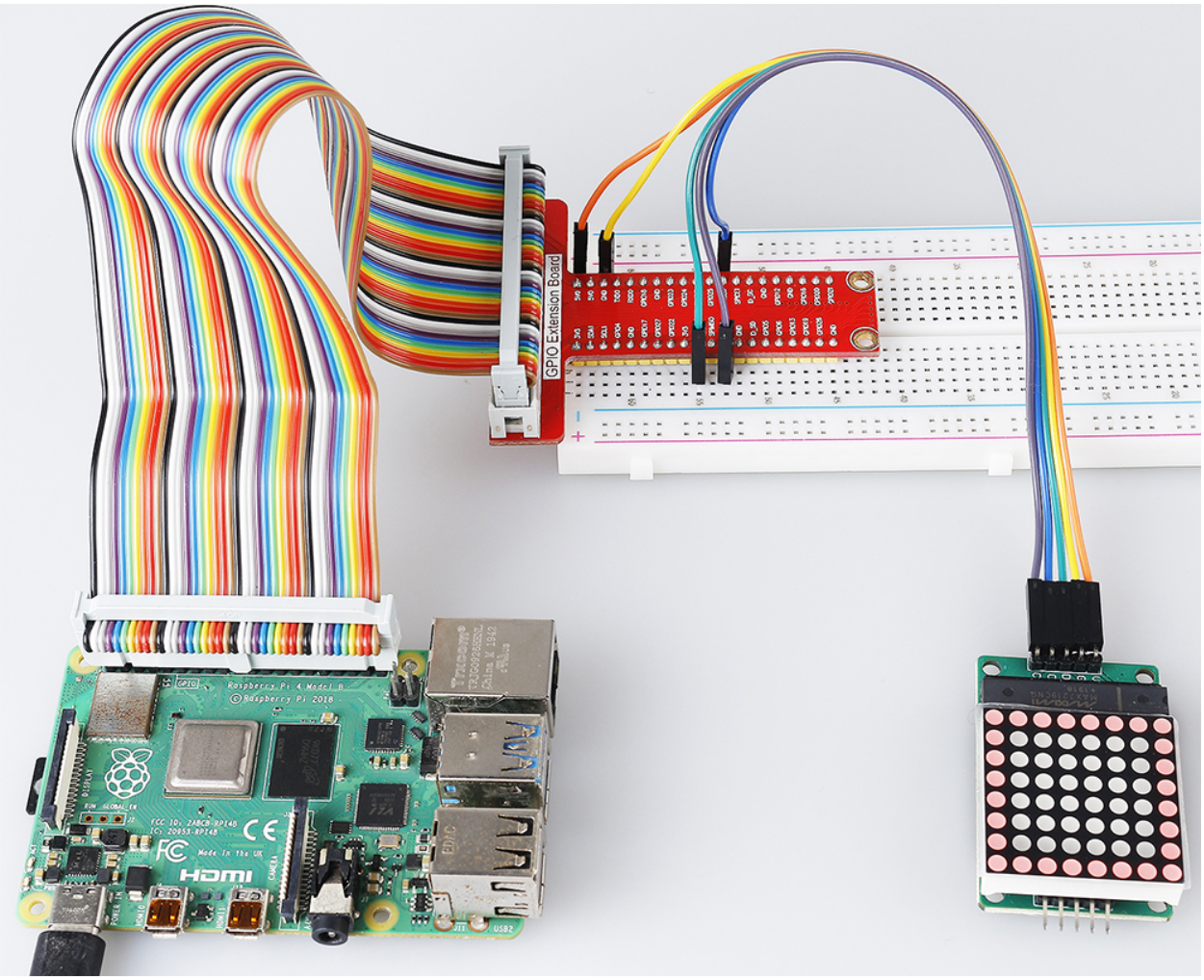
```
def scrollToDisplayText():
    with canvas(virtual) as draw:
        text(draw, (0, 0), "Hello, Nice to meet you!", fill="white",␣
↪font=proportional(CP437_FONT))

    for offset in range(150):
        virtual.set_position((offset,0))
        time.sleep(0.1)
```

`scrollToDisplayText()` implements “Hello, Nice to meet you!” as a scrolling text on the LED Matrix.

First, we pass virtual as an argument to the `canvas()` function, so that we can use the virtual window as the current display window. Then the `text()` function displays “Hello, Nice to meet you!” on the LED Matrix.

Using the for loop function, we move the virtual window in the X direction so that we can see the “Hello, Nice to meet you!” text scrolling.

Phenomenon Picture

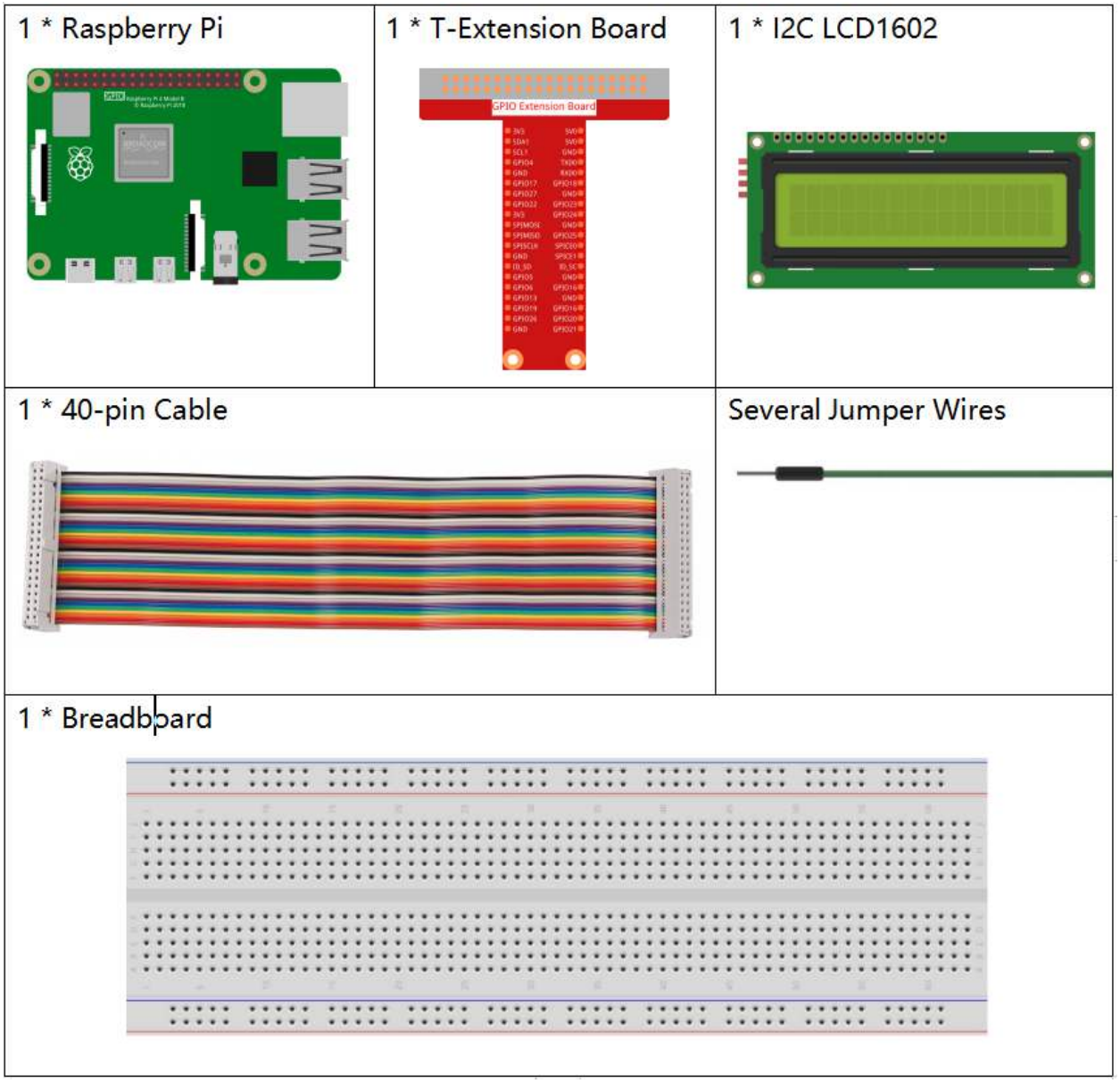


1.1.7 I2C LCD1602

Introduction

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

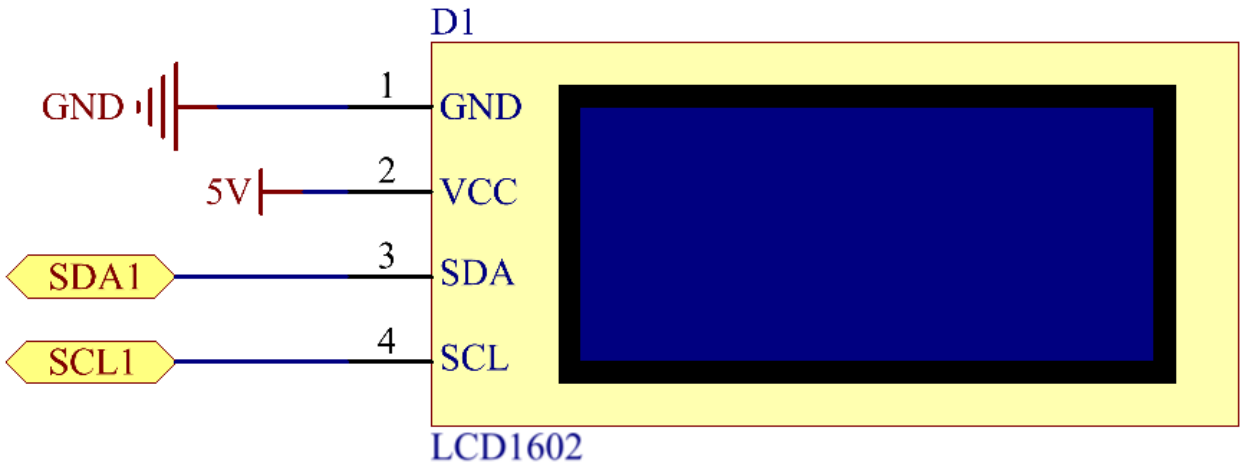
Components



- *GPIO Extension Board*
- *Breadboard*
- *I2C LCD1602*

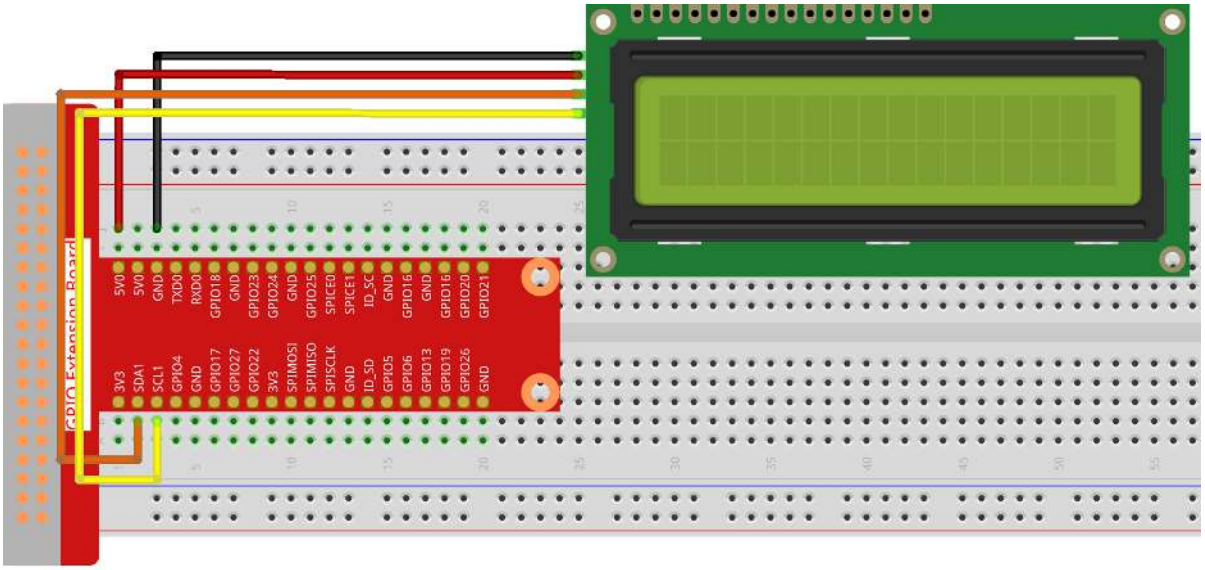
Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 4: Run.

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

After the code runs, you can see `Greetings!, From SunFounder` displaying on the LCD.

Note:

- If you get the error `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'`, you need to refer to *I2C Configuration* to enable the I2C.
- If you get `ModuleNotFoundError: No module named 'smbus2'` error, please run `sudo pip3 install smbus2`.
- If the error `OSError: [Errno 121] Remote I/O error` appears, it means the module is miswired or the module is broken.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import LCD1602
import time

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

def destroy():
    LCD1602.clear()

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

Code Explanation

```
import LCD1602
```

This file is an open source file for controlling I2C LCD1602. It allows us to easily use I2C LCD1602.

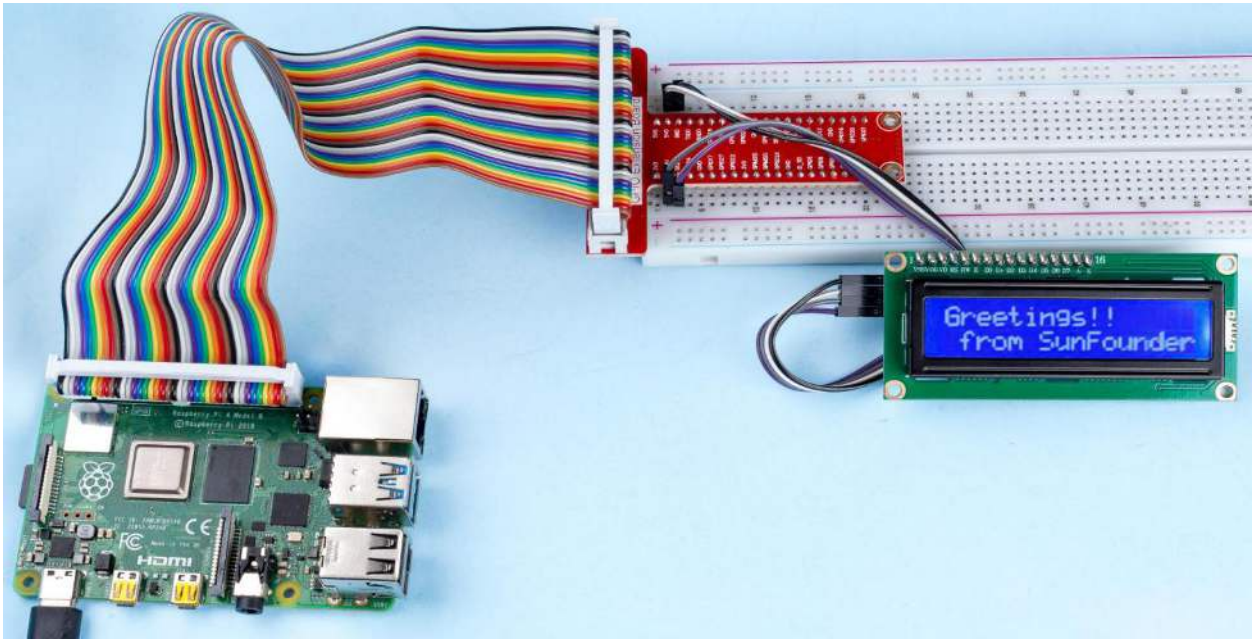
```
LCD1602.init(0x27, 1)
```

The function initializes the I2C system with the designated device symbol. The first parameter is the address of the I2C device, which can be detected through the i2cdetect command (see Appendix for details). The address of I2C LCD1602 is generally 0x27.

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

Within this function, 'Greetings!! ' is the character to be printed on the Row 0+1, column 0+1 on LCD. Now you can see "Greetings!! From SunFounder" displayed on the LCD.

Phenomenon Picture



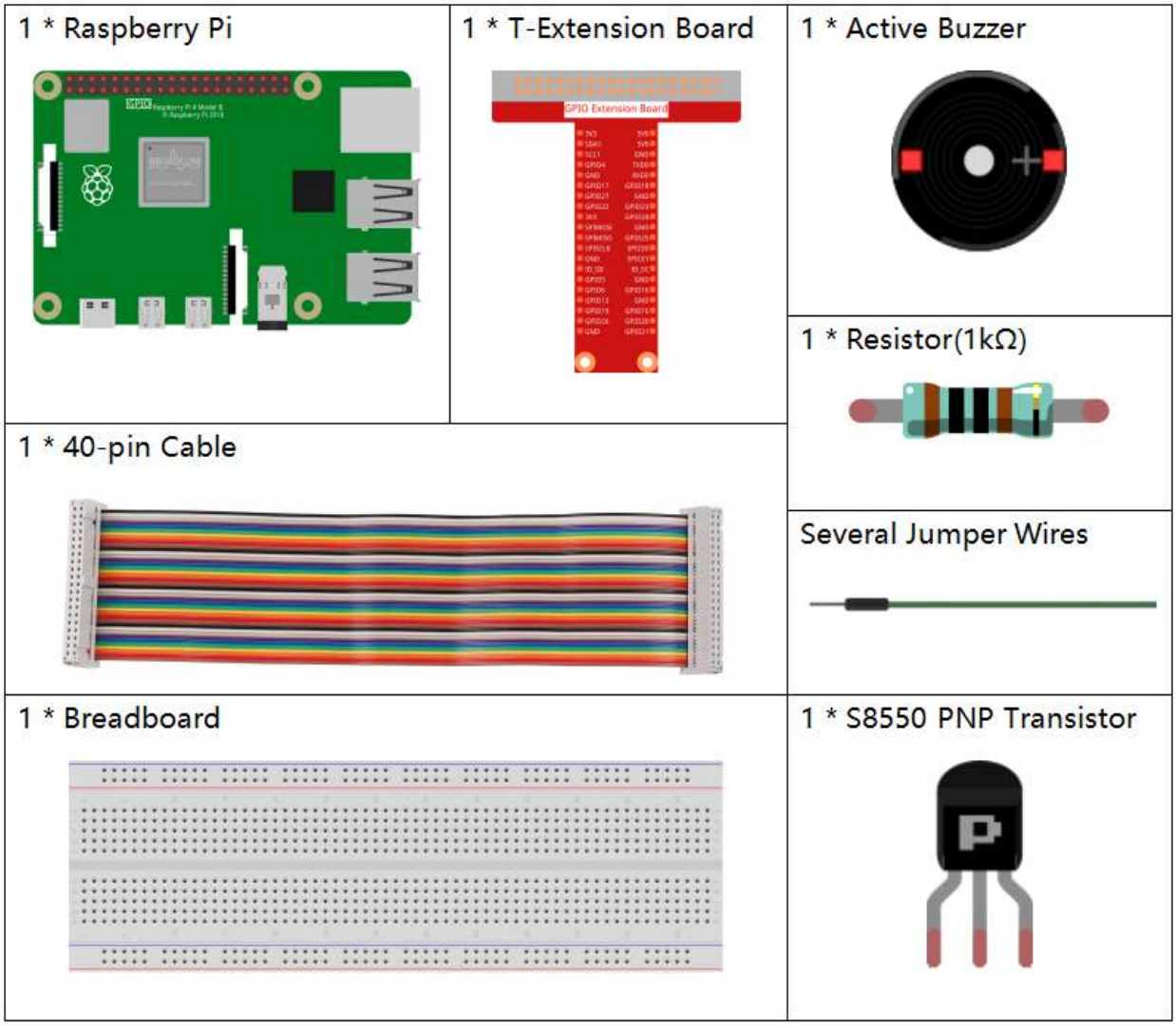
6.2.2 1.2 Sound

1.2.1 Active Buzzer

Introduction

In this project, we will learn how to drive an active buzzer to beep with a PNP transistor.

Components

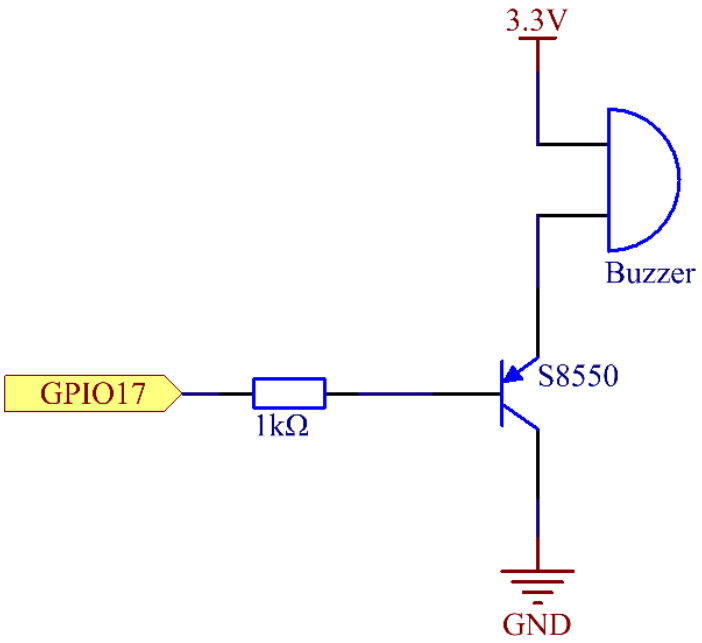


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Schematic Diagram

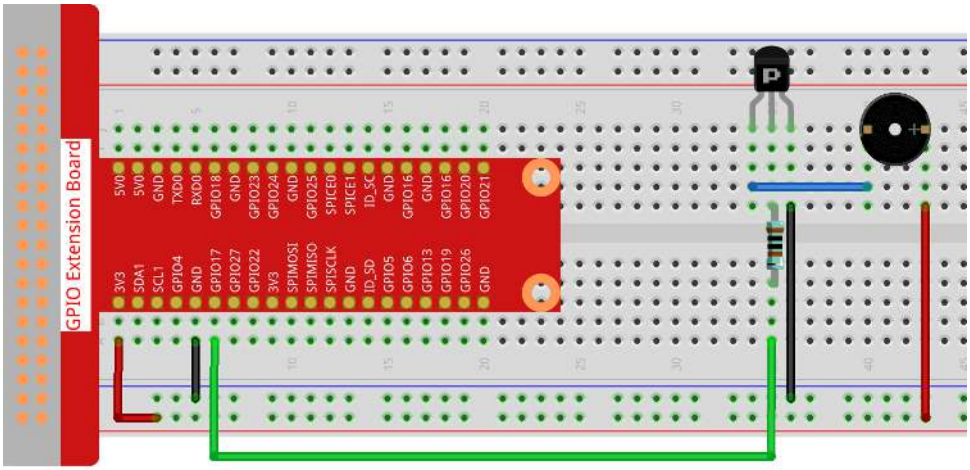
In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and the buzzer will make sounds. But when high level is supplied to the IO of Raspberry Pi, the transistor will be cut off and the buzzer will not make sounds.

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



Experimental Procedures

Step 1: Build the circuit. (The active buzzer has a white table sticker on the surface and a black back.)



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

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

The code run, the buzzer beeps.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as buzzer pin
BeepPin = 17

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

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

def destroy():
    # Turn off buzzer
    GPIO.output(BeepPin, GPIO.HIGH)
```

(continues on next page)

(continued from previous page)

```
    # 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 Explanation

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

Set the buzzer pin as low level to make the buzzer beep.

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

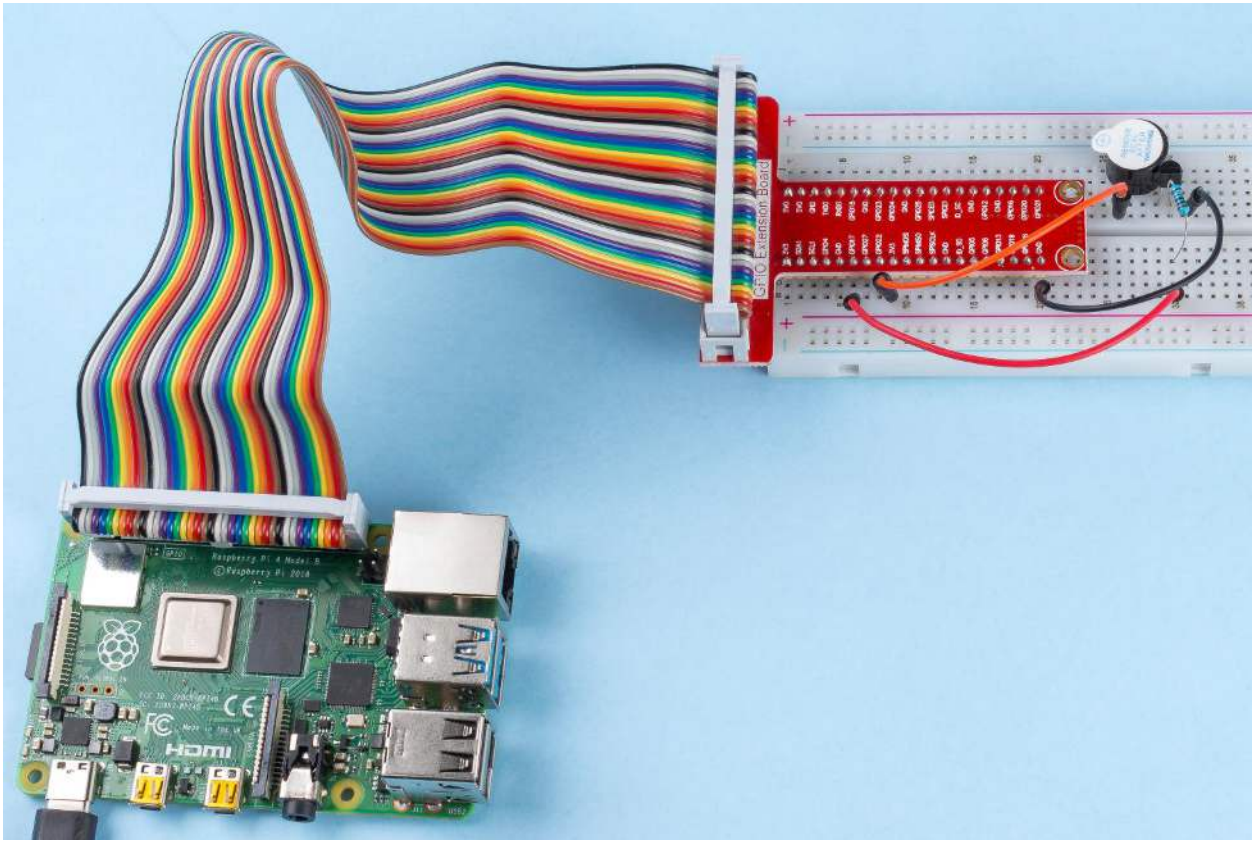
Wait for 0.1 second. Change the switching frequency by changing this parameter.

Note: Not the sound frequency. Active Buzzer cannot change sound frequency.

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

Close the buzzer.

Phenomenon Picture

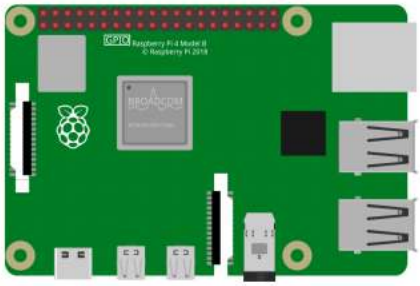
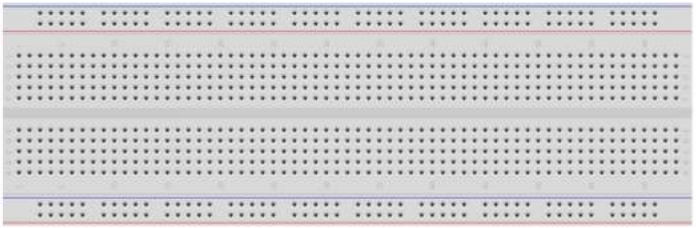
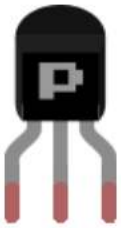


1.2.2 Passive Buzzer

Introduction

In this project, we will learn how to make a passive buzzer play music.

Components

<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>

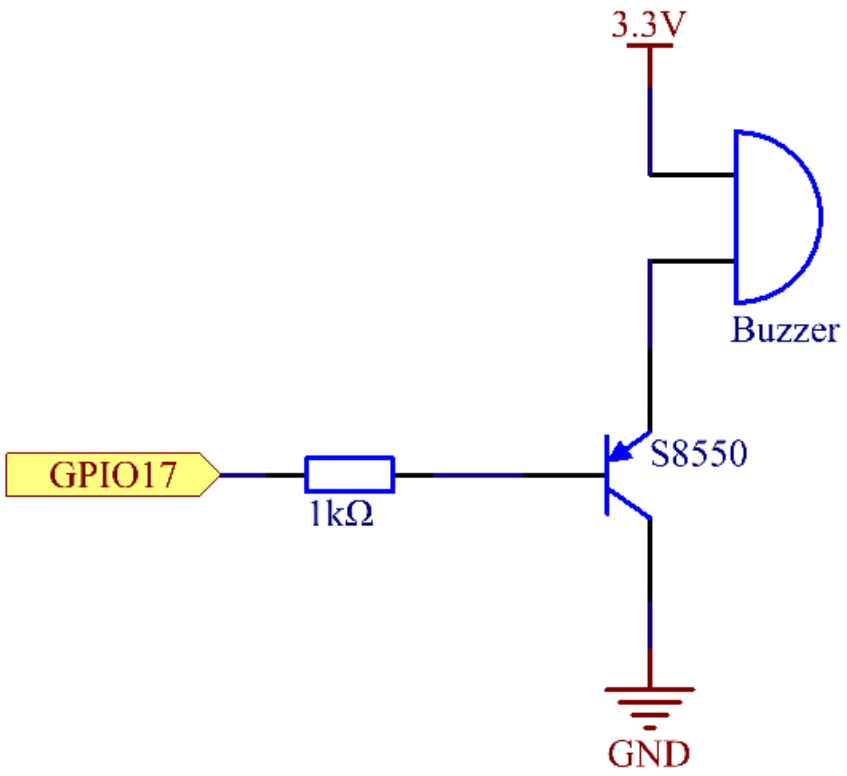
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Schematic Diagram

In this experiment, a passive buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor.

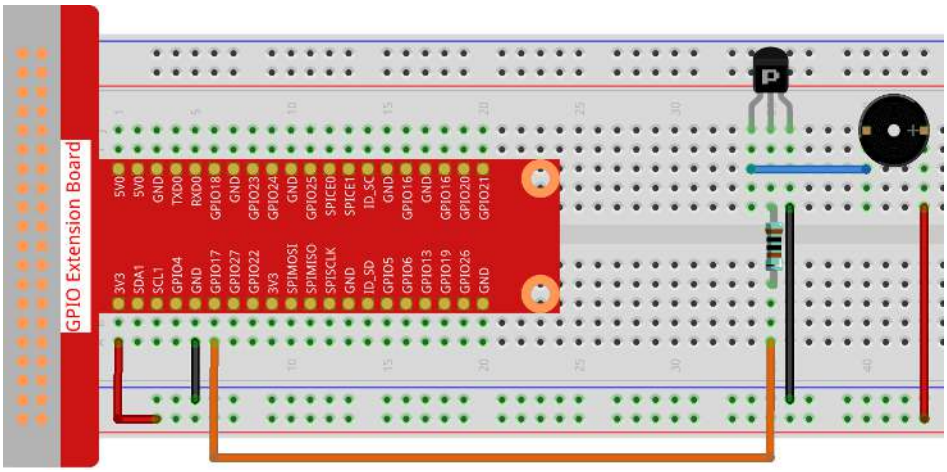
When GPIO17 is given different frequencies, the passive buzzer will emit different sounds; in this way, the buzzer plays music.

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



Experimental Procedures

Step 1: Build the circuit. (The Passive buzzer with green circuit board on the back.)



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

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

The code run, the buzzer plays a piece of music.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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 ]
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

```
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
```

These are the frequencies of each note. The first 0 is to skip CL[0] so that the number 1-7 corresponds to the CDEFGAB of the tone.

```
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],
```

(continues on next page)

(continued from previous page)

```
            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] ]
```

These arrays are the notes of a song.

```
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 ]
```

Every sound beat (each number) represents the beat, or 0.5s

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

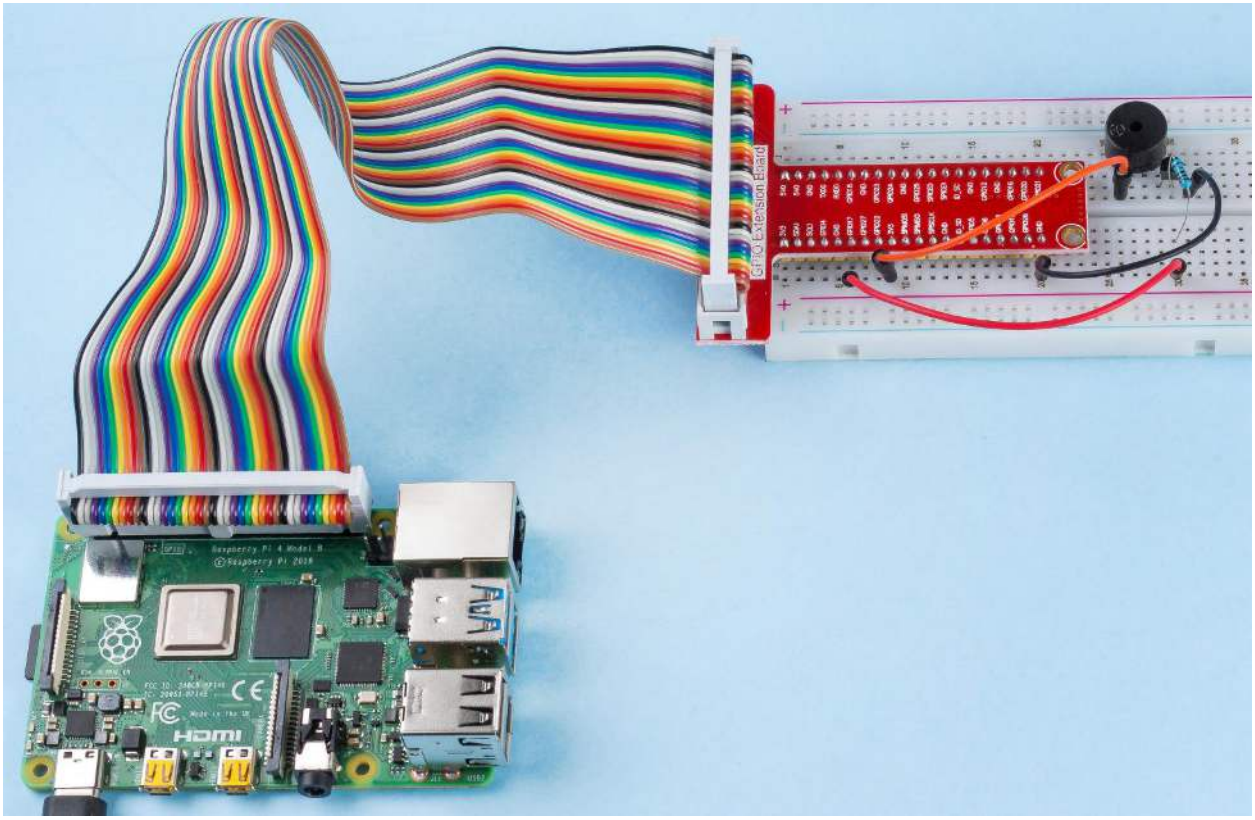
Define pin Buzzer as PWM pin, then set its frequency to 440 and Buzz.start(50) is used to run PWM. What's more, set the duty cycle to 50%.

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

Run a for loop, then the buzzer will play the notes in the array song_1[] with the beats in the beat_1[] array, .

Now you can hear the passive buzzer playing music.

Phenomenon Picture



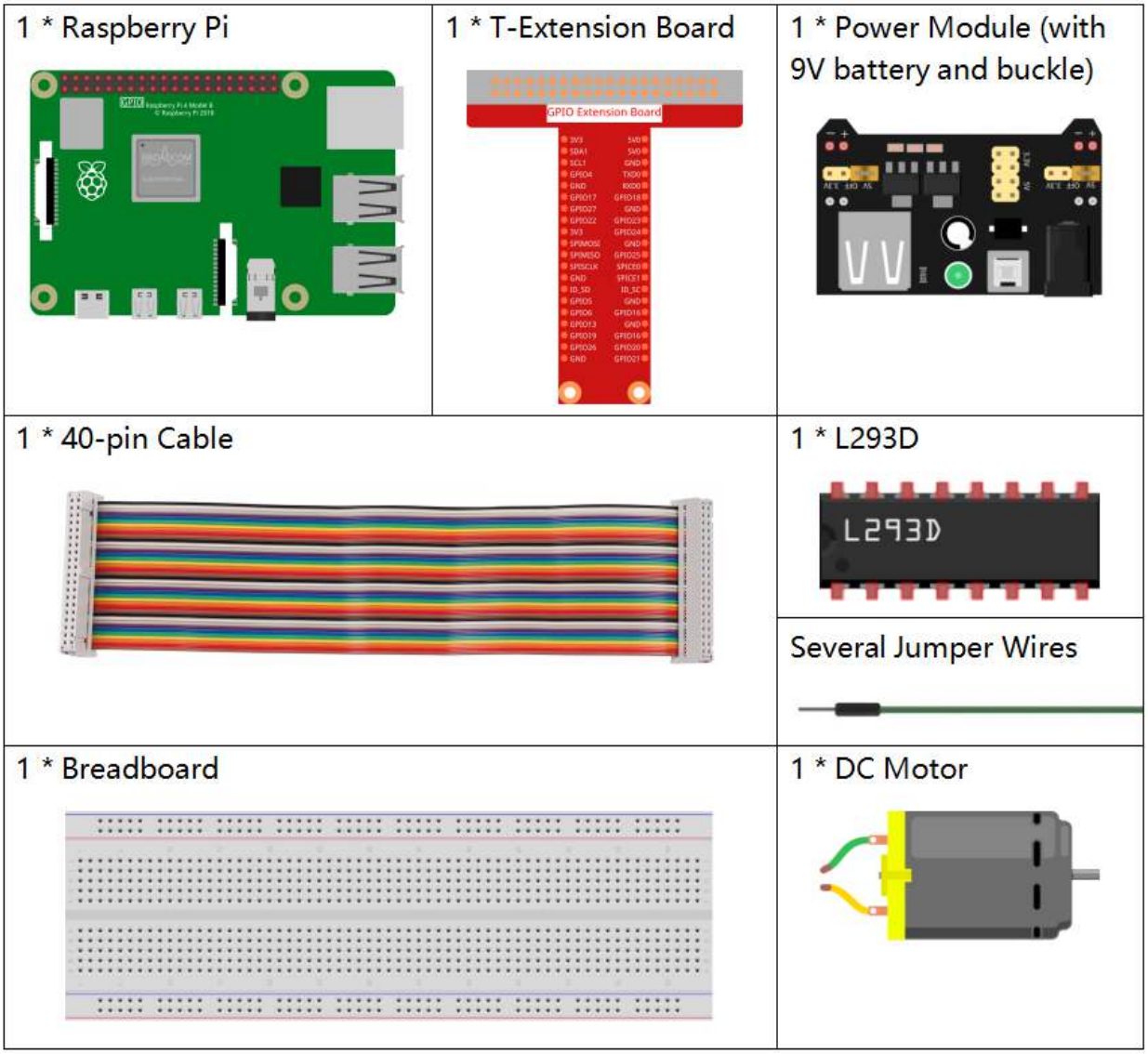
6.2.3 1.3 Drivers

1.3.1 Motor

Introduction

In this project, we will learn to how to use L293D to drive a DC motor and make it rotate clockwise and counterclockwise. Since the DC Motor needs a larger current, for safety purpose, here we use the Power Supply Module to supply motors.

Components



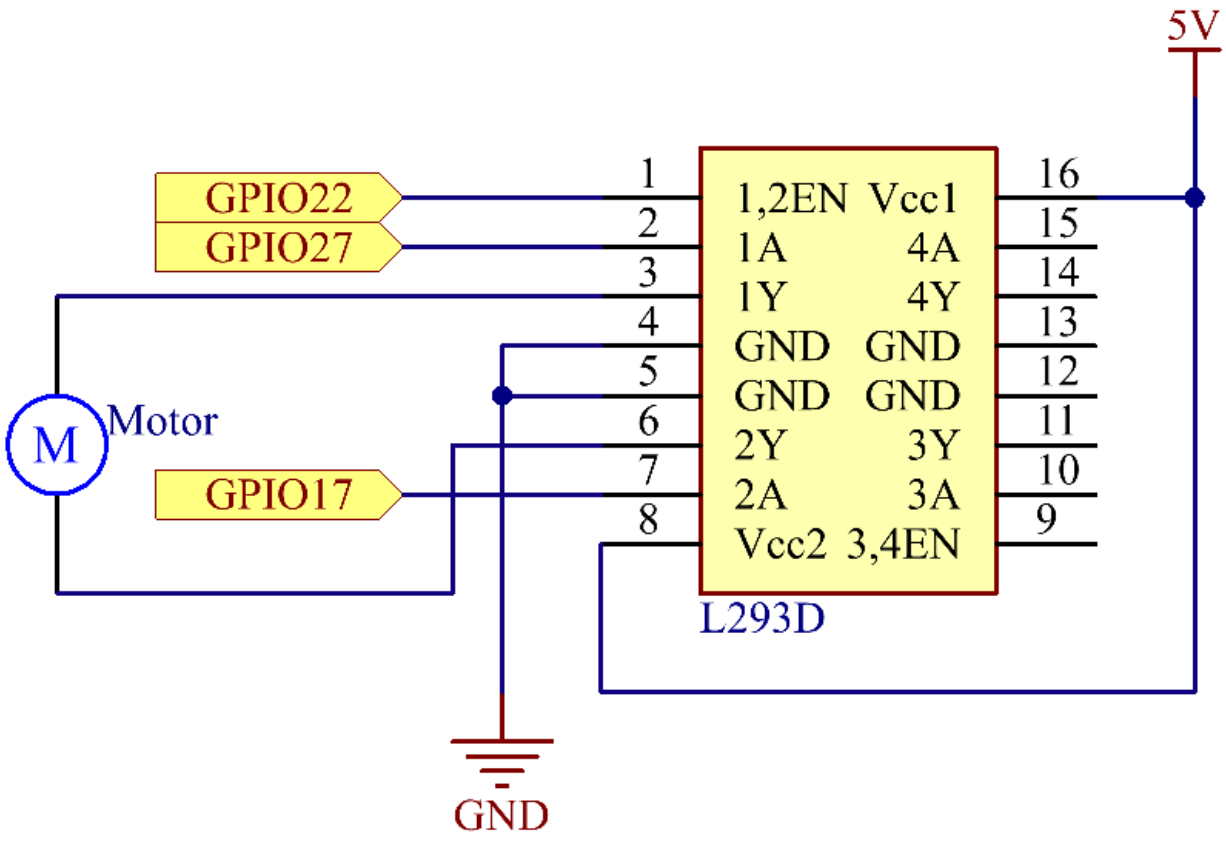
- *GPIO Extension Board*
- *Breadboard*
- *Power Supply Module*

- *L293D*
- *DC Motor*

Schematic Diagram

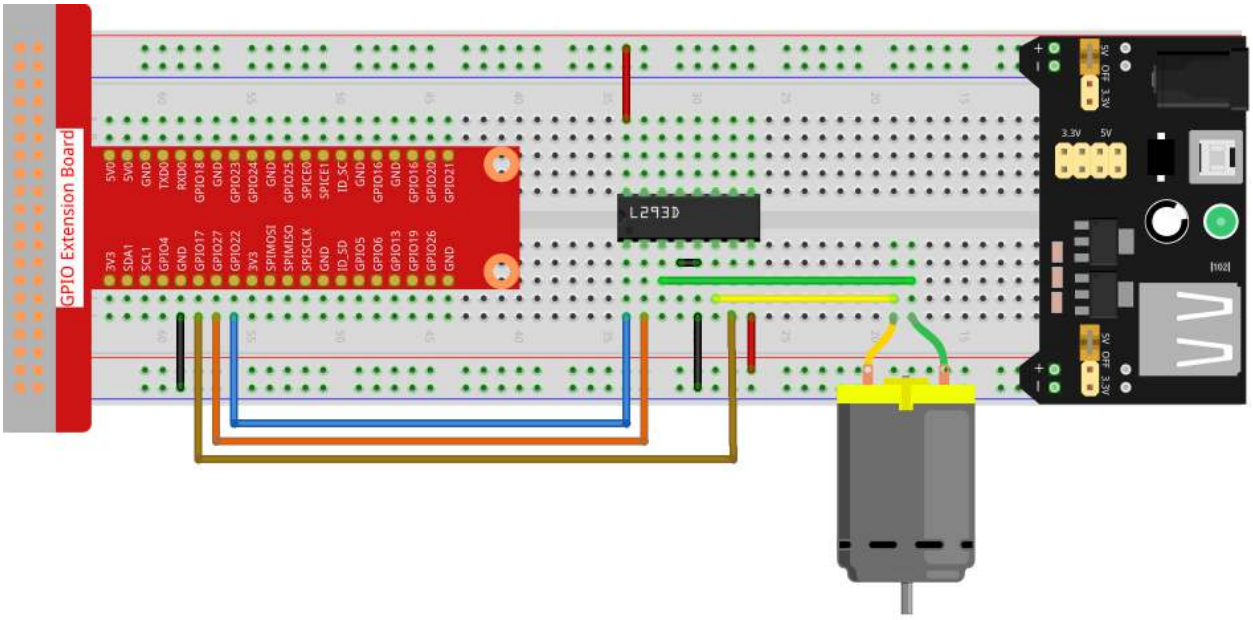
Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to GPIO22, and set it as high level. Connect pin2 to GPIO27, and pin7 to GPIO17, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

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



Experimental Procedures

Step 1: Build the circuit.



Note: 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.

Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

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

As the code runs, the motor first rotates clockwise for 5s then stops for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

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

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

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

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

(continues on next page)

(continued from previous page)

```
        motor(directions['STOP'])
        time.sleep(5)

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

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

Code Explanation

```
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")
...
```

Create a function, `motor()` whose variable is direction. As the condition that direction=1 is met, the motor rotates clockwise; when direction=-1, the motor rotates anticlockwise; and under the condition that direction=0, it stops rotating.

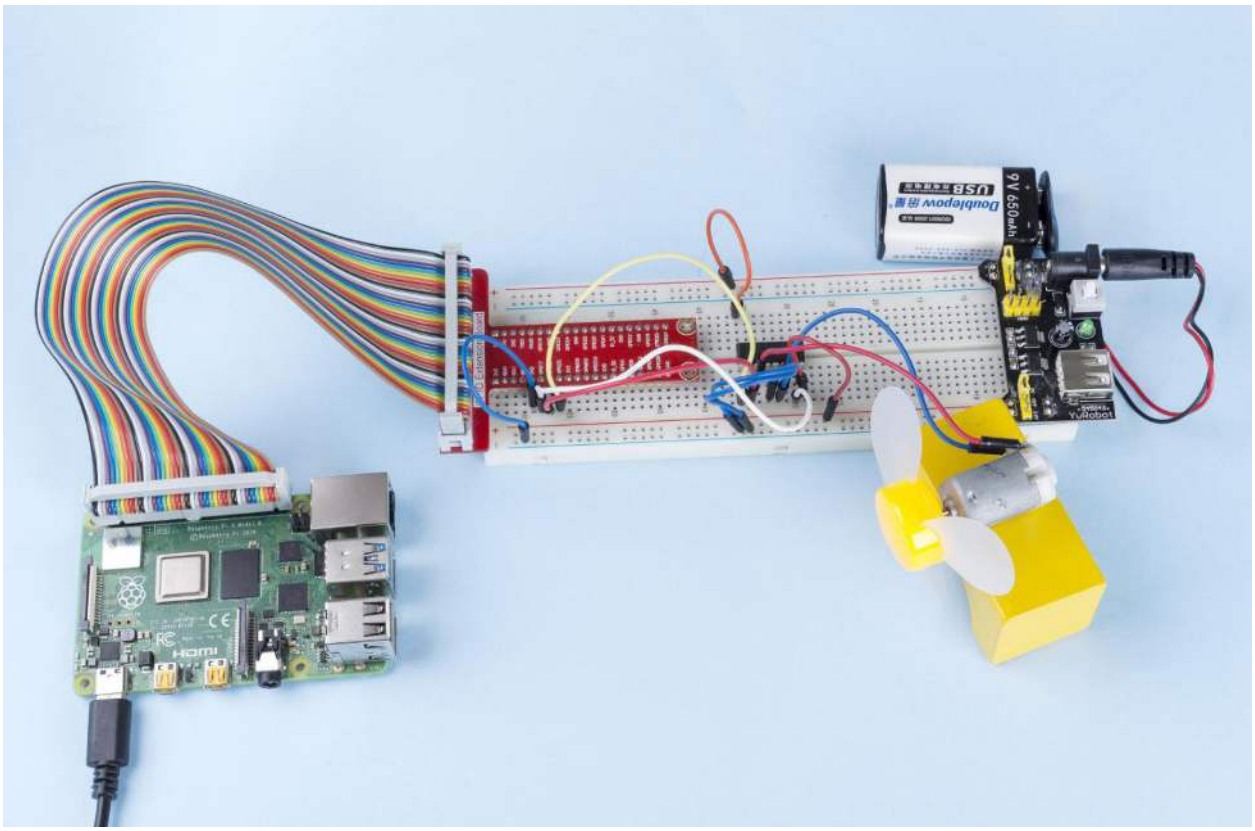
```
def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}
    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])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

In the main() function, create an array, directions[], in which CW is equal to 1, the value of CCW is -1, and the number 0 refers to Stop.

As the code runs, the motor first rotates clockwise for 5s then stop for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

Now, you should see the motor blade rotating.

Phenomenon Picture

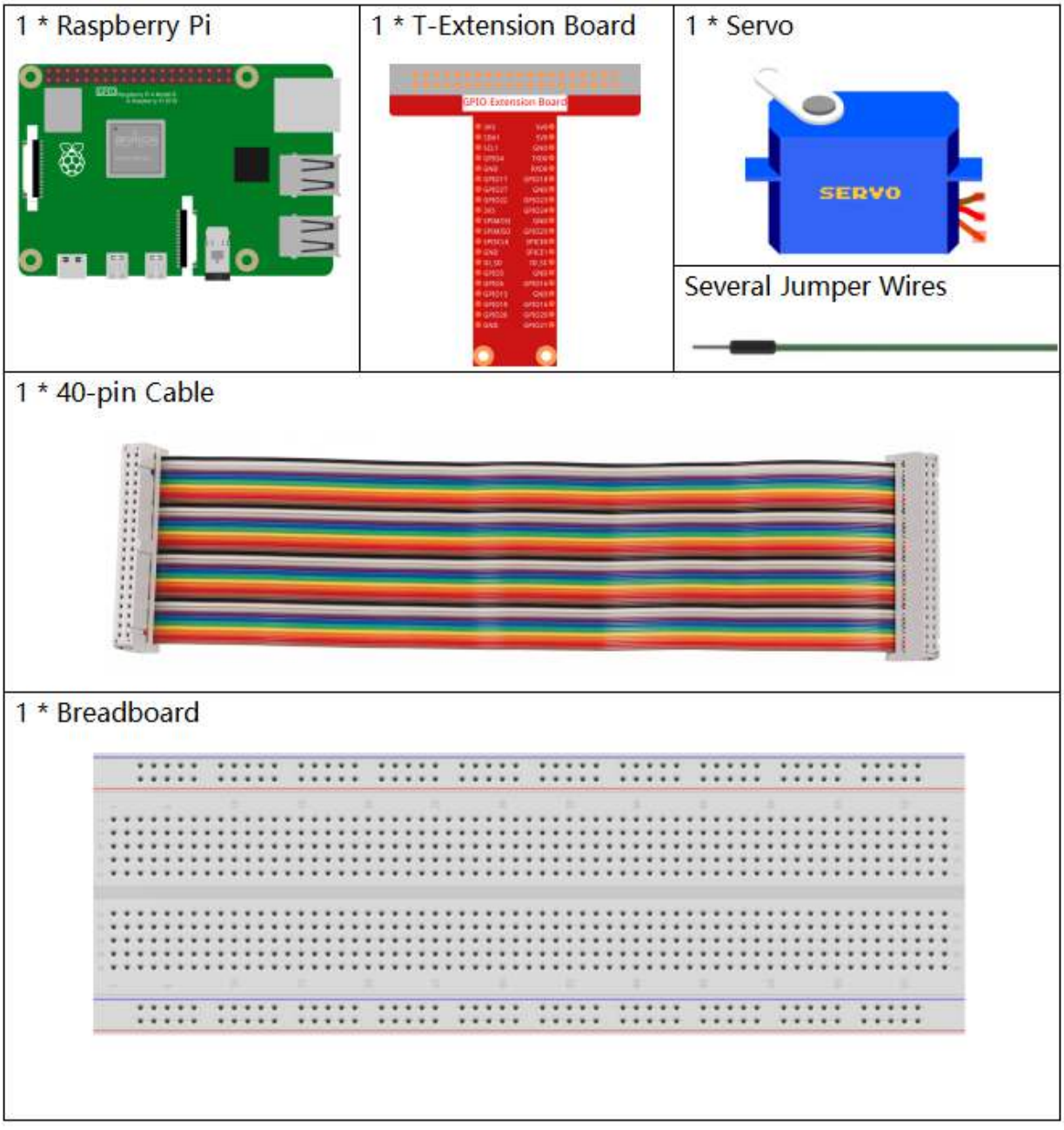


1.3.2 Servo

Introduction

In this project, we will learn how to make the servo rotate.

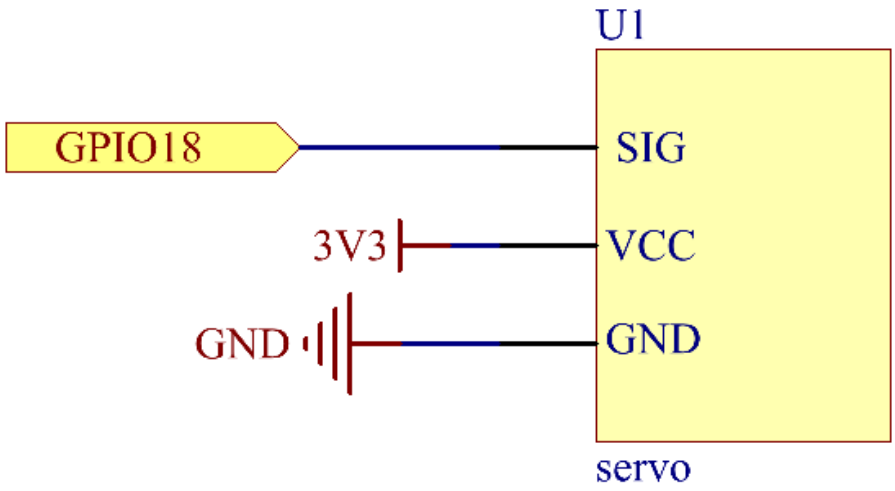
Components



- *GPIO Extension Board*
- *Breadboard*
- *Servo*

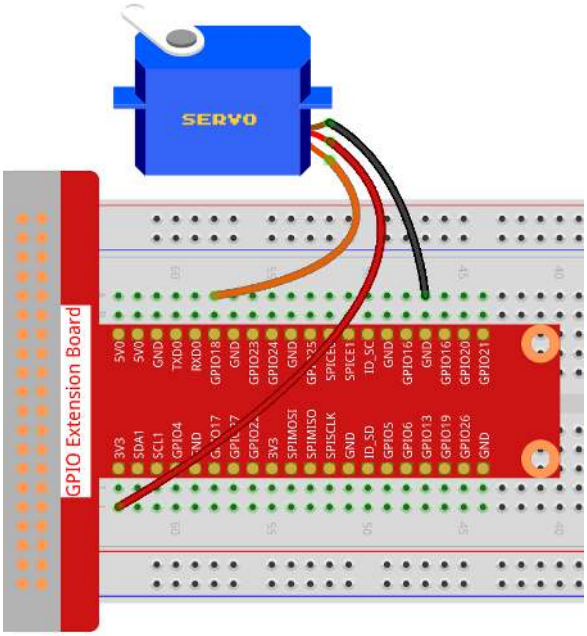
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

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

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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:
```

(continues on next page)

(continued from previous page)

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

Code Explanation

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

Set the servoPin to PWM pin, then the frequency to 50hz, and the period to 20ms.

p.start(0): Run the PWM functionand set the initial value to 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
```

Create a function, setAngle() to write angle that ranges from 0 to 180 into the servo.

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

This code is used to limit the angle within the range 0-180°.

The min() function returns the minimum of the input values. If 180<angle, then return 180,if not, return angle.

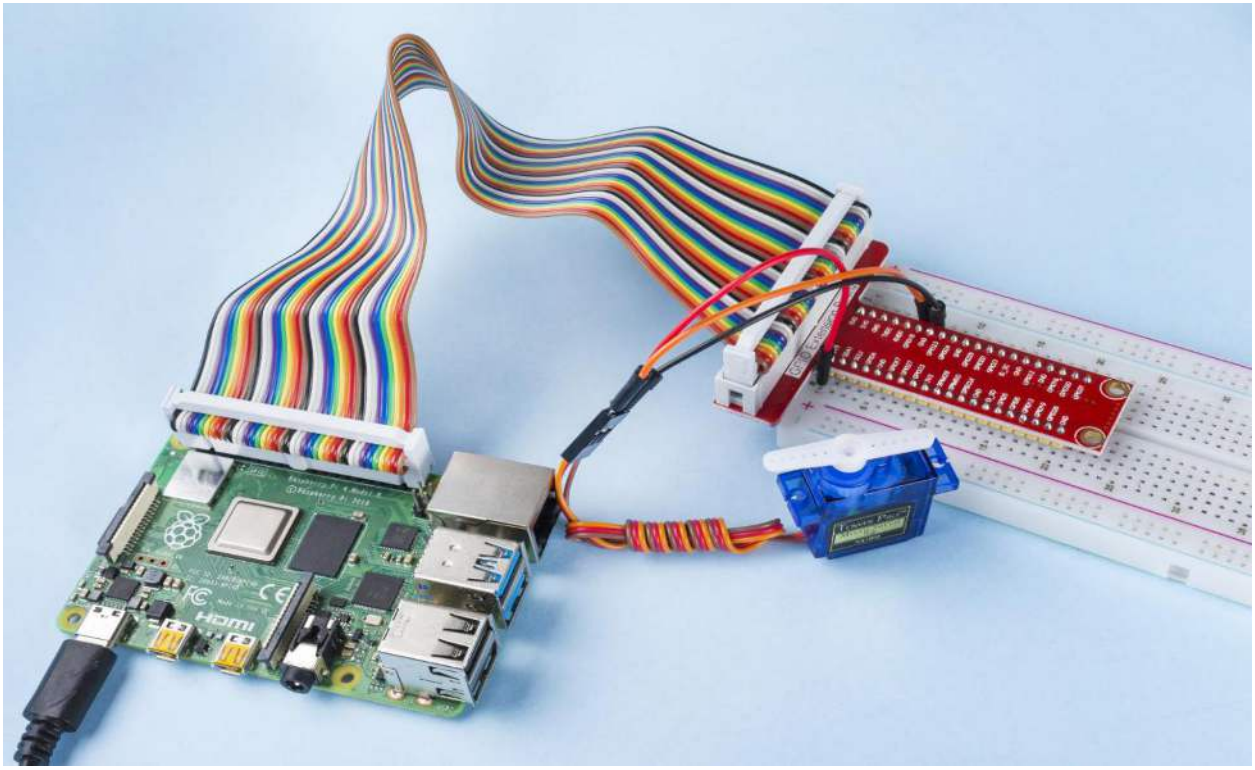
The max() method returns the maximum element in an iterable or largest of two or more parameters. If 0>angle, then return 0, if not, return angle.

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

To render a range 0 ~ 180 ° to the servo, the pulse width of the servo is set to 0.5ms(500us)-2.5ms(2500us).

The period of PWM is 20ms(20000us), thus the duty cycle of PWM is (500/20000)%-(2500/20000)%, and the range 0 ~ 180 is mapped to 2.5 ~ 12.5.

Phenomenon Picture

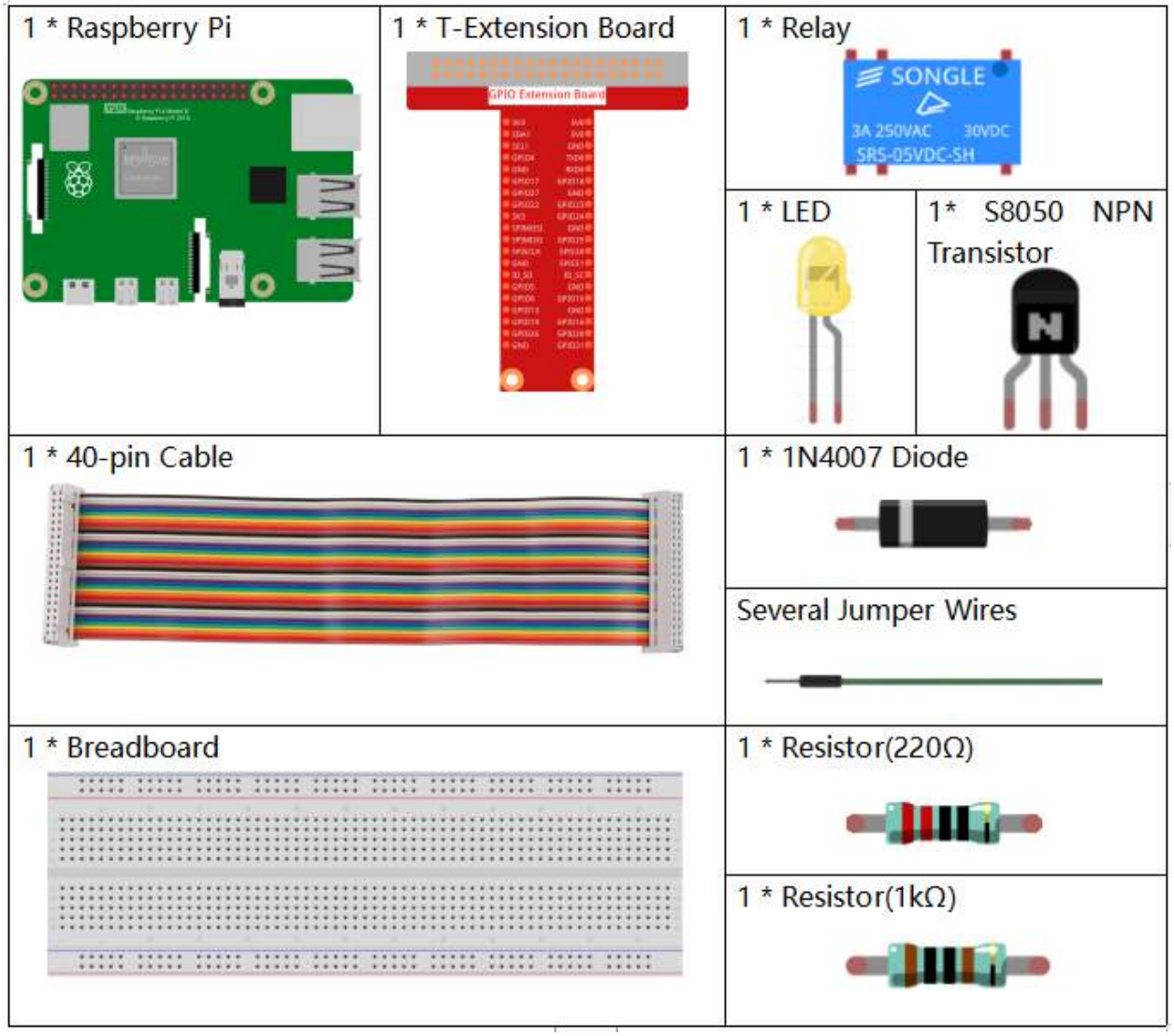


1.3.3 Relay

Introduction

In this project, we will learn to use a relay. It is one of the commonly used components in automatic control system. When the voltage, current, temperature, pressure, etc., reaches, exceeds or is lower than the predetermined value, the relay will connect or interrupt the circuit, to control and protect the equipment.

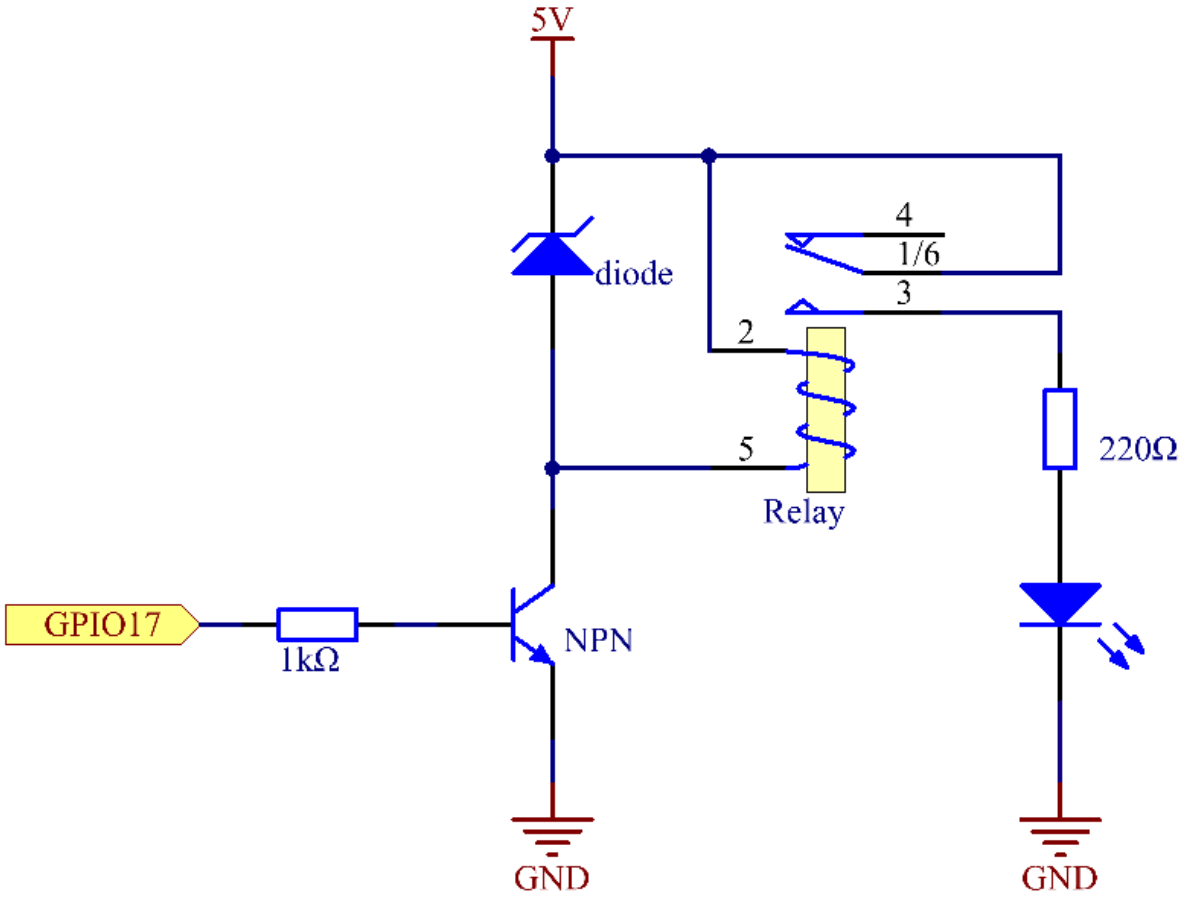
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Transistor*
- *Relay*
- *Diode*

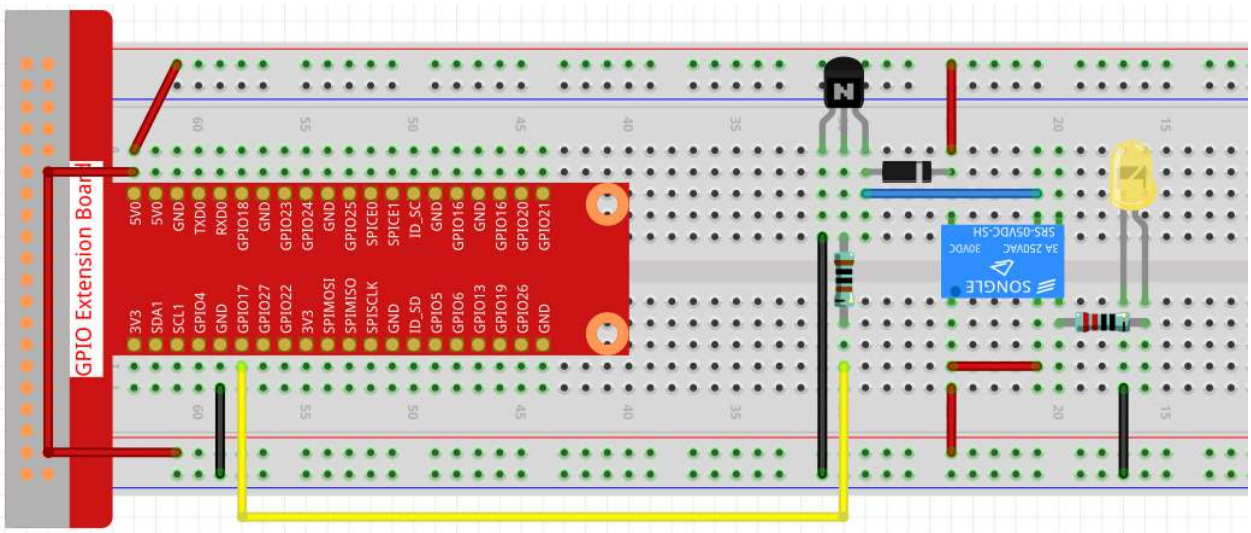
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 1.3.3_Relay.py
```

While the code is running, the LED lights up. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import time

# Set GPIO17 as control pin
relayPin = 17

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

# Define a main function for main process
def main():
    while True:
        print ('Relay open...')
        # Tick
        GPIO.output(relayPin, GPIO.LOW)
```

(continues on next page)

(continued from previous page)

```
        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 Explanation

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

Set the pins of transistor as low level to let the relay open, LED does not turn on.

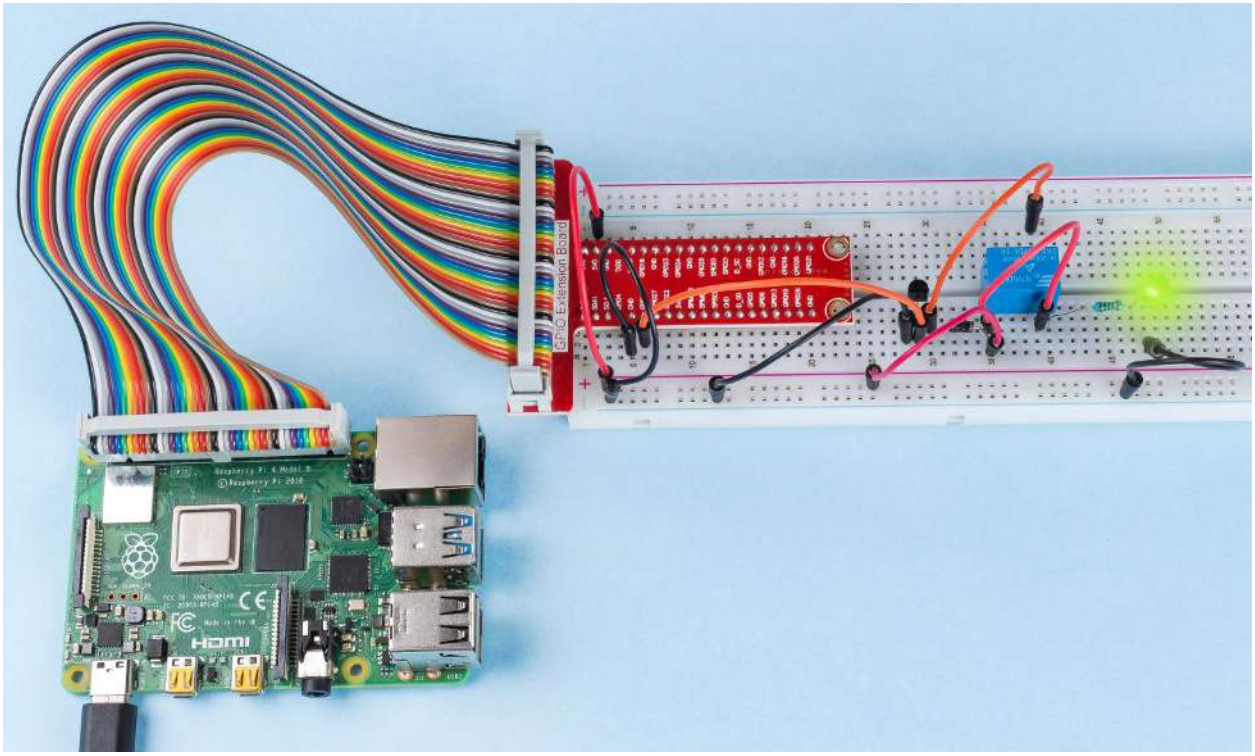
```
time.sleep(1)
```

wait for 1 second.

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

Set the pins of the transistor as low level to actuate the relay, LED lights up.

Phenomenon Picture



6.3 Input

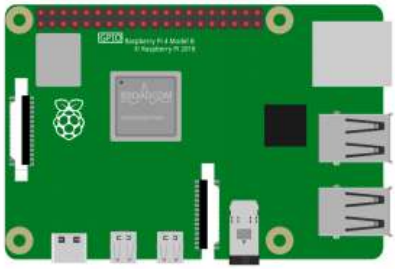
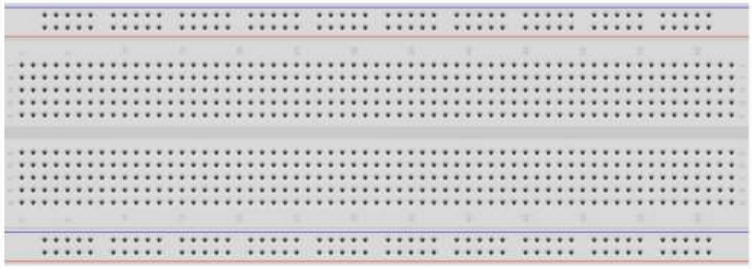
6.3.1 2.1 Controllers

2.1.1 Button

Introduction

In this project, we will learn how to turn on or off the LED by using a button.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>1 * LED
 </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor 10KΩ
 </td></tr>
<tr><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>Several Jumper Wires

1 * Button
 </td></tr>
</table>

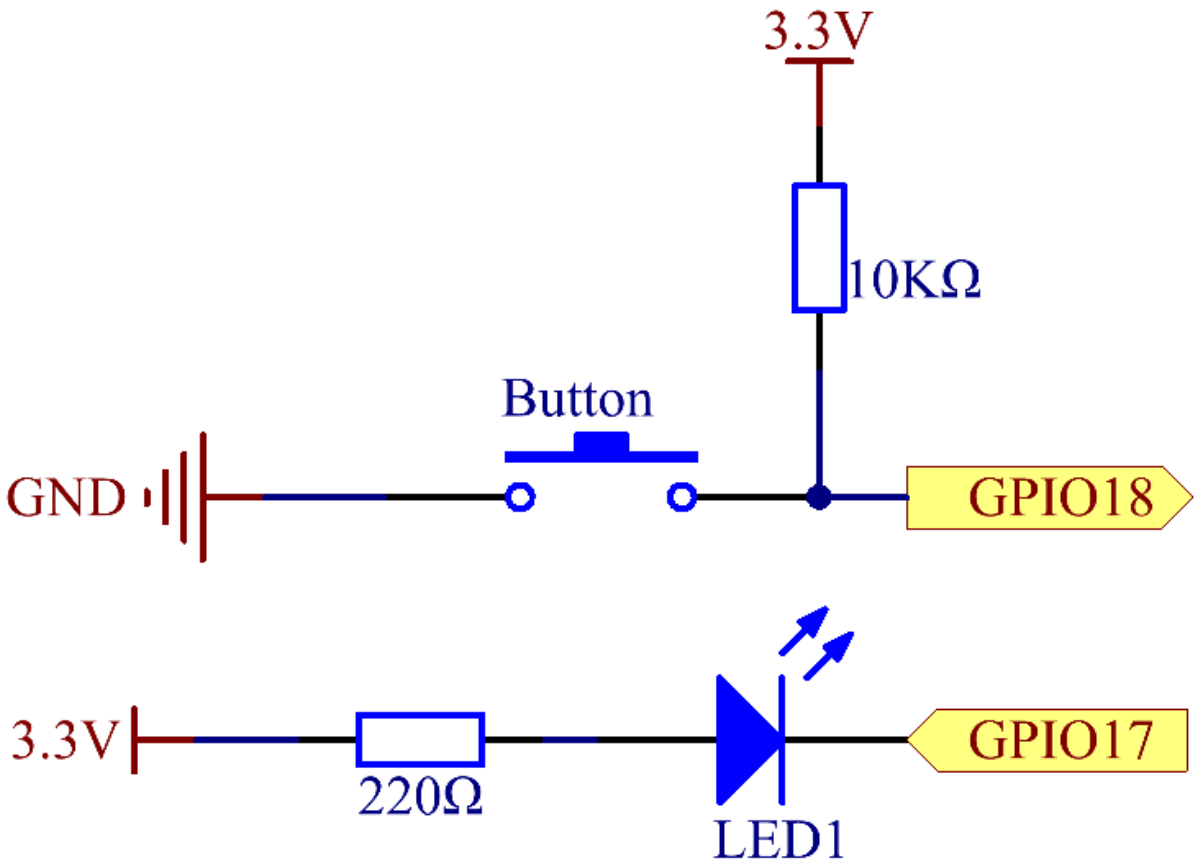
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Button*

Schematic Diagram

Use a normally open button as the input of Raspberry Pi, the connection is shown in the schematic diagram below. When the button is pressed, the GPIO18 will turn into low level (0V). We can detect the state of the GPIO18 through programming. That is, if the GPIO18 turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

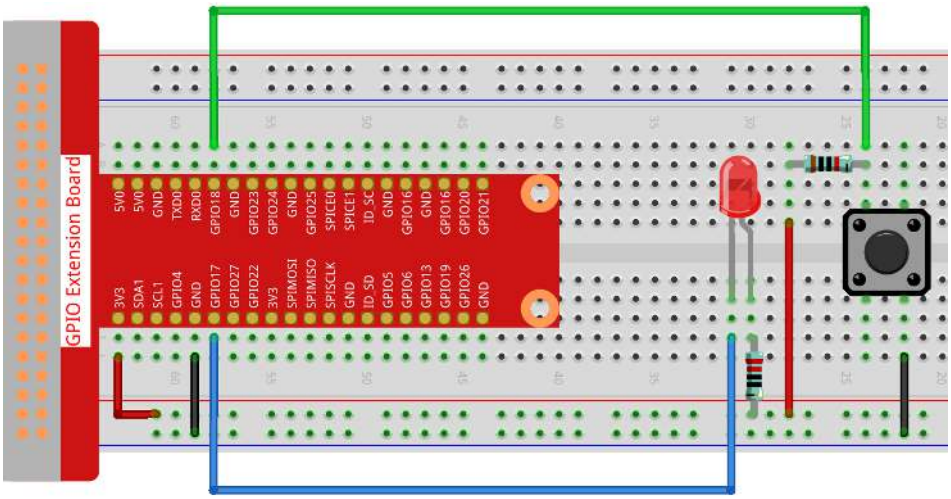
Note: The longer pin of the LED is the anode and the shorter one is the cathode.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run the code.

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

Now, press the button, and the LED will light up; press the button again, and the LED will go out. At the same time, the state of the LED will be printed on the screen.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time
LedPin = 17  # Set GPIO17 as LED pin
BtnPin = 18  # Set GPIO18 as button pin

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

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,
    # and initial level to high (3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    # Set BtnPin's mode to input,
    # and pull up to high (3.3V)
    GPIO.setup(BtnPin, GPIO.IN)
```

(continues on next page)

(continued from previous page)

```
# Define a callback function for button callback
def swLed(ev=None):
    global Led_status
    # Switch led status(on-->off; off-->on)
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
    if Led_status:
        print ('LED OFF...')
    else:
        print ('...LED ON')

# Define a main function for main process
def main():
# Set up a falling detect on BtnPin,
    # and callback function to swLed
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
    while True:
        # Don't do anything.
        time.sleep(1)

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

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

Code Explanation

```
LedPin = 17
```

Set GPIO17 as LED pin

```
BtnPin = 18
```

Set GPIO18 as button pin

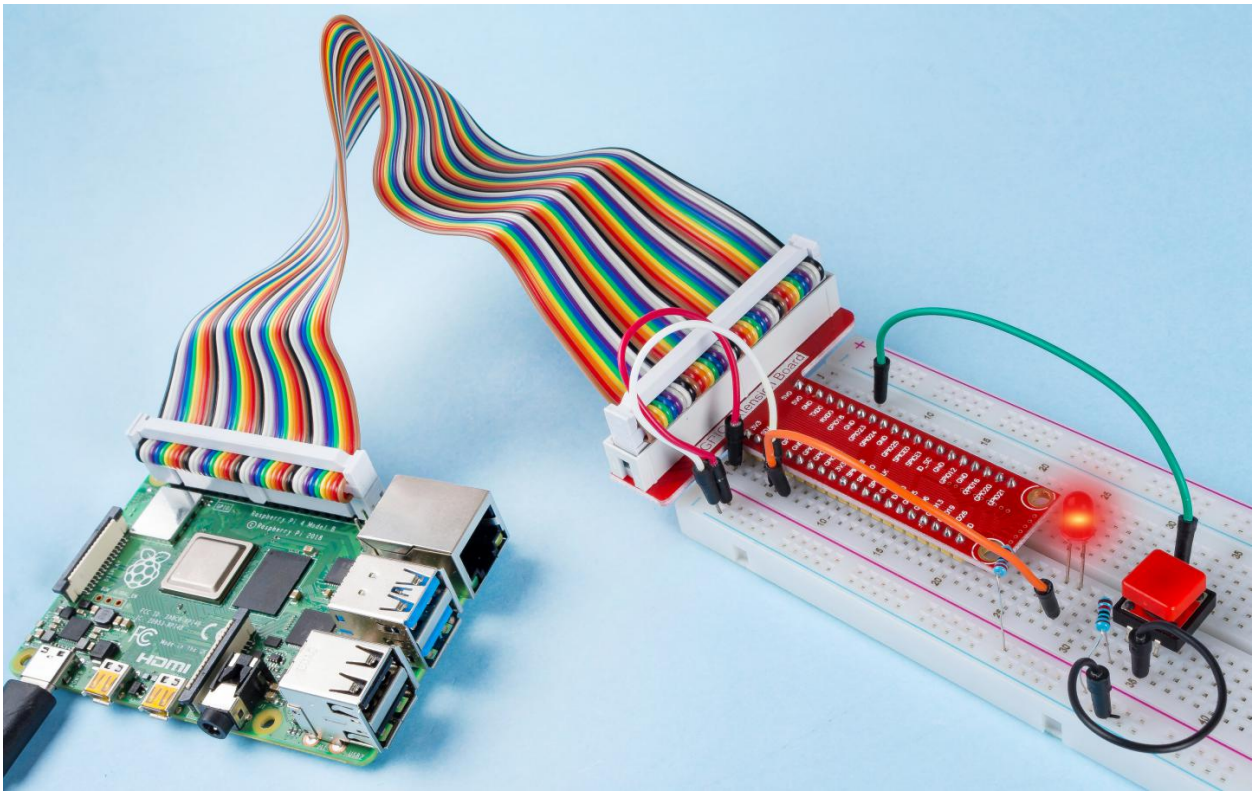
```
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

Set up a falling detect on BtnPin, and then when the value of BtnPin changes from a high level to a low level, it means that the button is pressed. The next step is calling the function, 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)
```

Define a callback function as button callback. When the button is pressed at the first timeand the condition, not Led_status is false, GPIO.output() function is called to light up the LED. As the button is pressed once again, the state of LED will be converted from false to true, thus the LED will turn off.

Phenomenon Picture



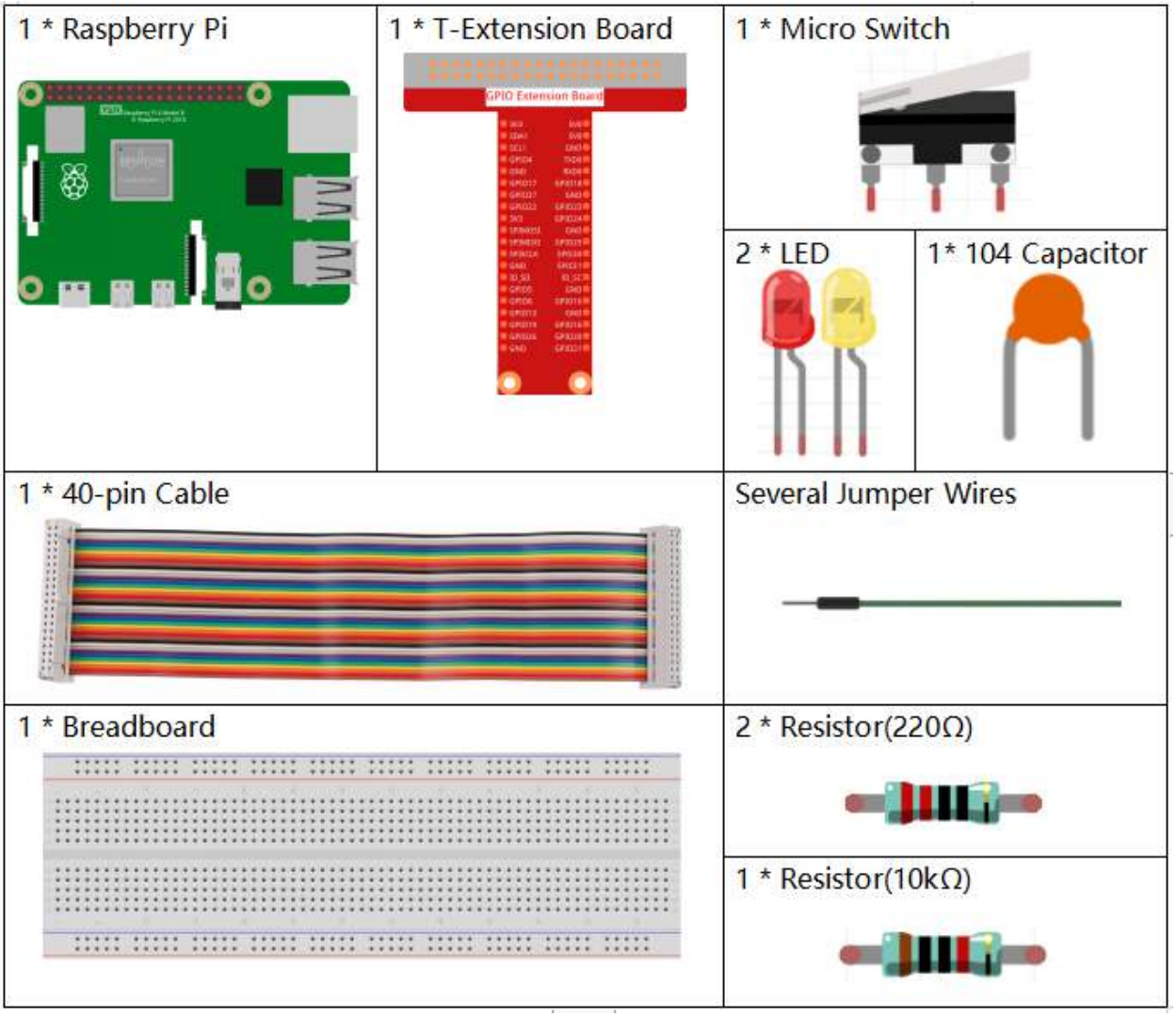
2.1.2 Micro Switch

Introduction

In this project, we will learn how to use Micro Switch. A Micro Switch is a small, very sensitive switch which requires minimum compression to activate. Because they are reliable and sensitive, micro switches are often used as a safety device.

They are used to prevent doors from closing if something or someone is in the way and other applications similar.

Components

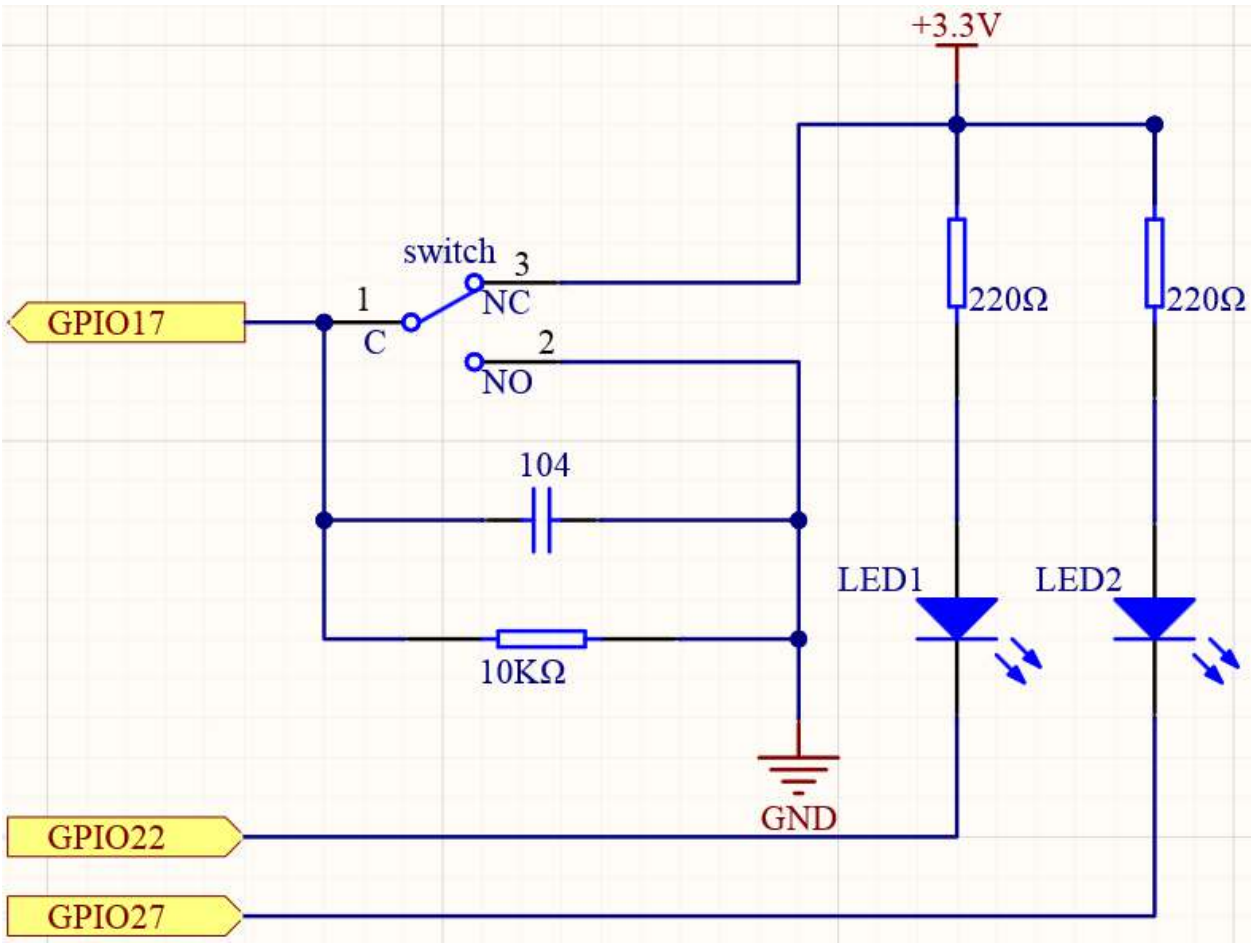


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Micro Switch*
- *Capacitor*

Schematic Diagram

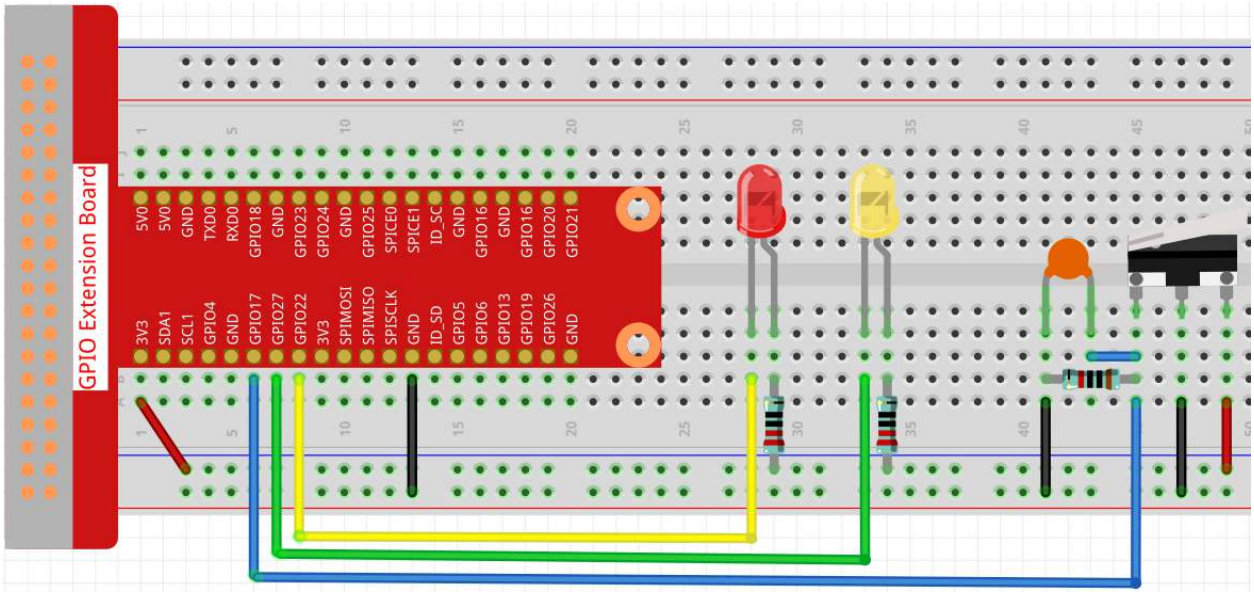
Connect the left pin of the Micro Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you press and release the move arm of the Micro Switch, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 2.1.2_MicroSwitch.py
```

While the code is running, press the moving arm, then the yellow LED lights up; release the moving arm, the red LED turns on.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import time

# Set #17 as micro switch pin, #22 as led1 pin, #27 as led2 pin
microPin = 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 microPin input
```

(continues on next page)

(continued from previous page)

```
    # Set ledPin output,
    # and initial level to High(3.3v)
    GPIO.setup(microPin, 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:
        # micro switch high, led1 on
        if GPIO.input(microPin) == 1:
            print ('LED1 ON')
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)

        # micro switch low, led2 on
        if GPIO.input(microPin) == 0:
            print ('    LED2 ON')
            GPIO.output(led2Pin, GPIO.LOW)
            GPIO.output(led1Pin, GPIO.HIGH)

        time.sleep(0.5)
# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(led1Pin, GPIO.HIGH)
    GPIO.output(led2Pin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

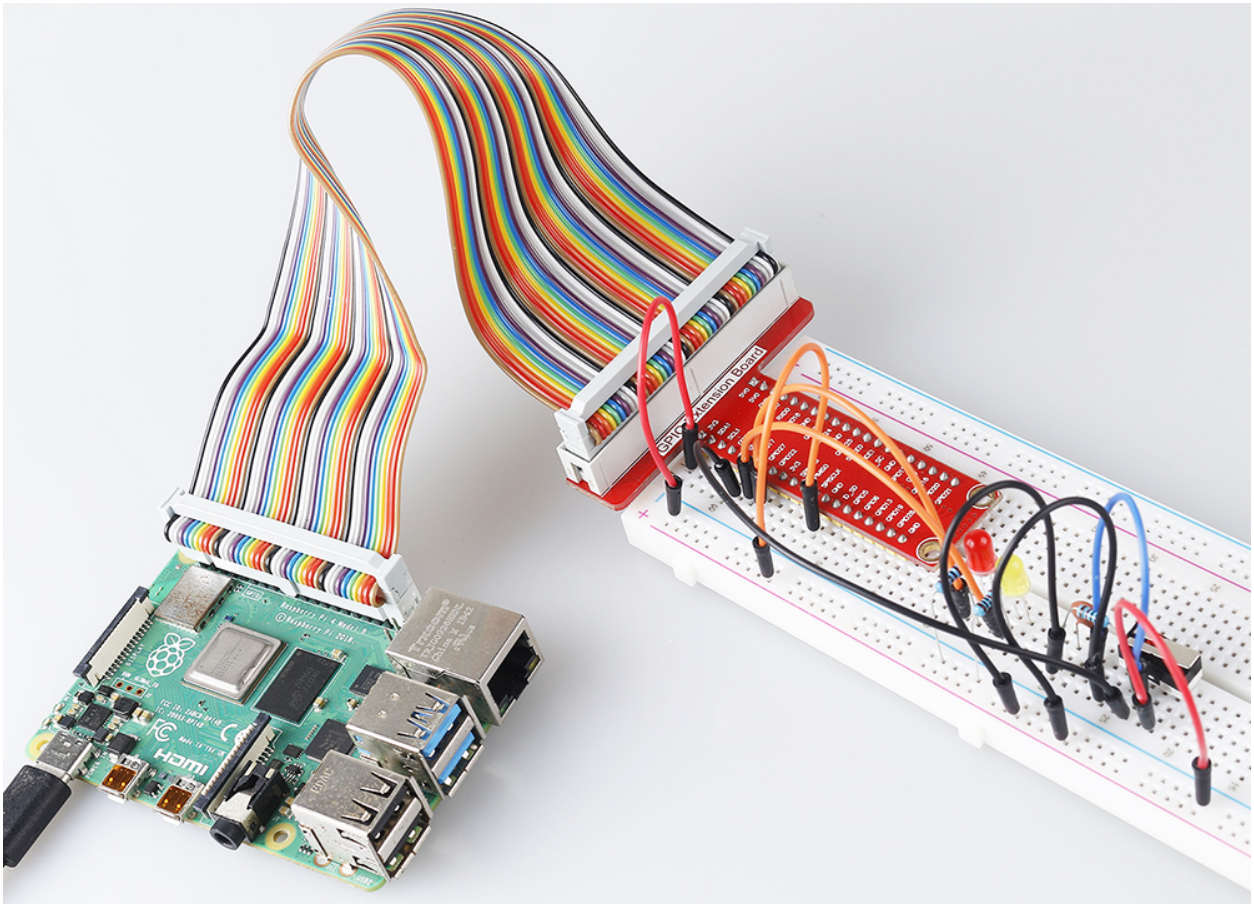
```
if GPIO.input(slidePin) == 1:
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)
```

When the moving arm of the micro switch is released, the left pin is connected to the right pin; at this time, a high level will be read on GPIO17, and then LED1 will be on and LED2 will be off.

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

When the move arm is pressed, the left pin and the middle pin are connected. At this point a low level will be read on GPIO17, then turns LED2 on and LED1 off.

Phenomenon Picture

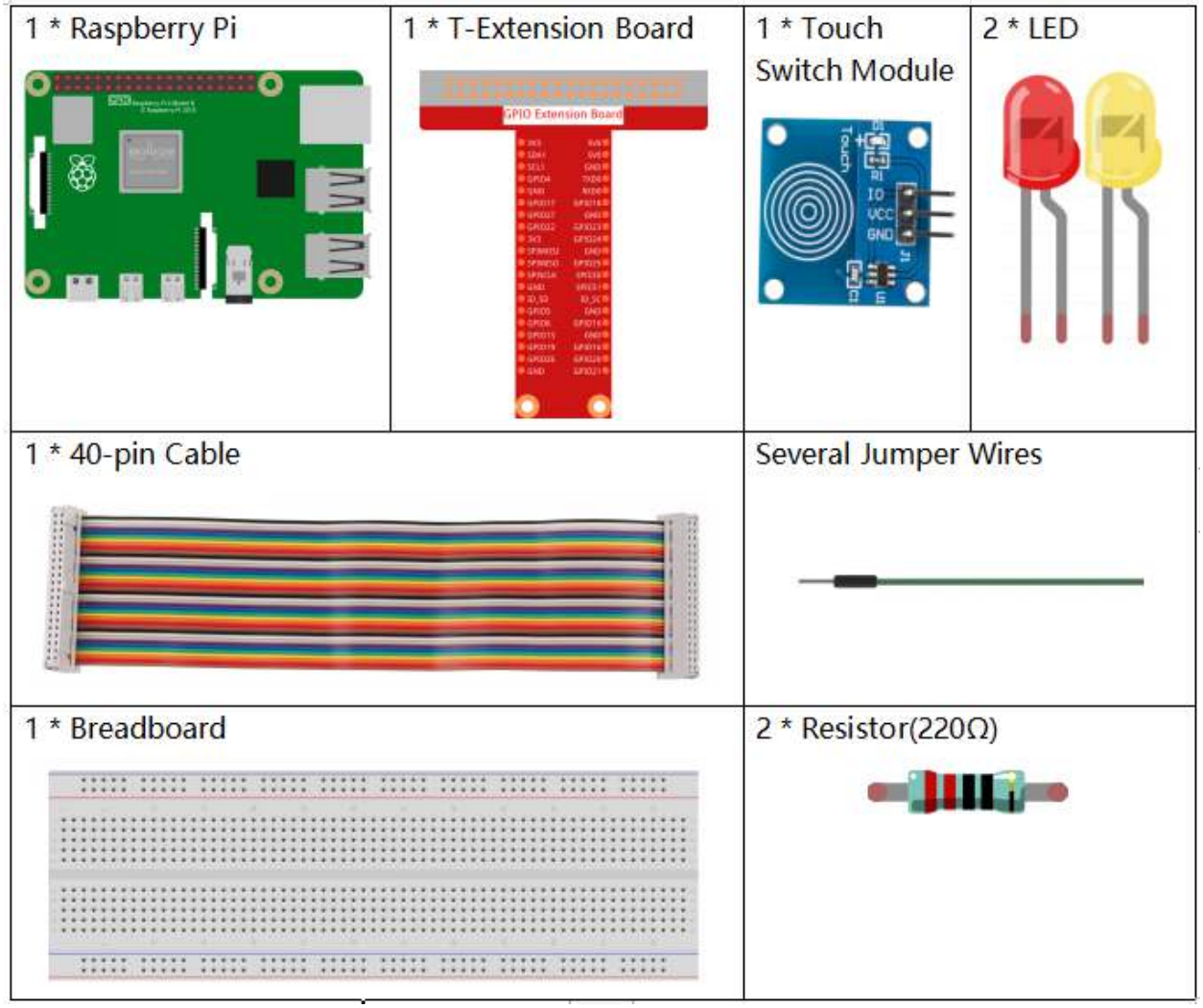


2.1.3 Touch Switch Module

Introduction

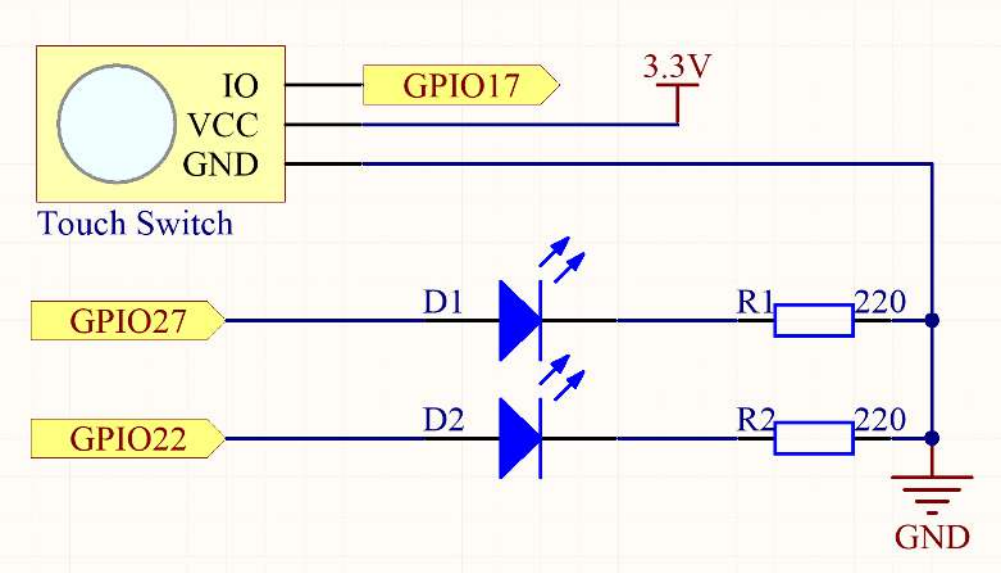
In this project, you will learn about touch switch module. It can replace the traditional kinds of switch with these advantages: convenient operation, fine touch sense, precise control and least mechanical wear.

Components



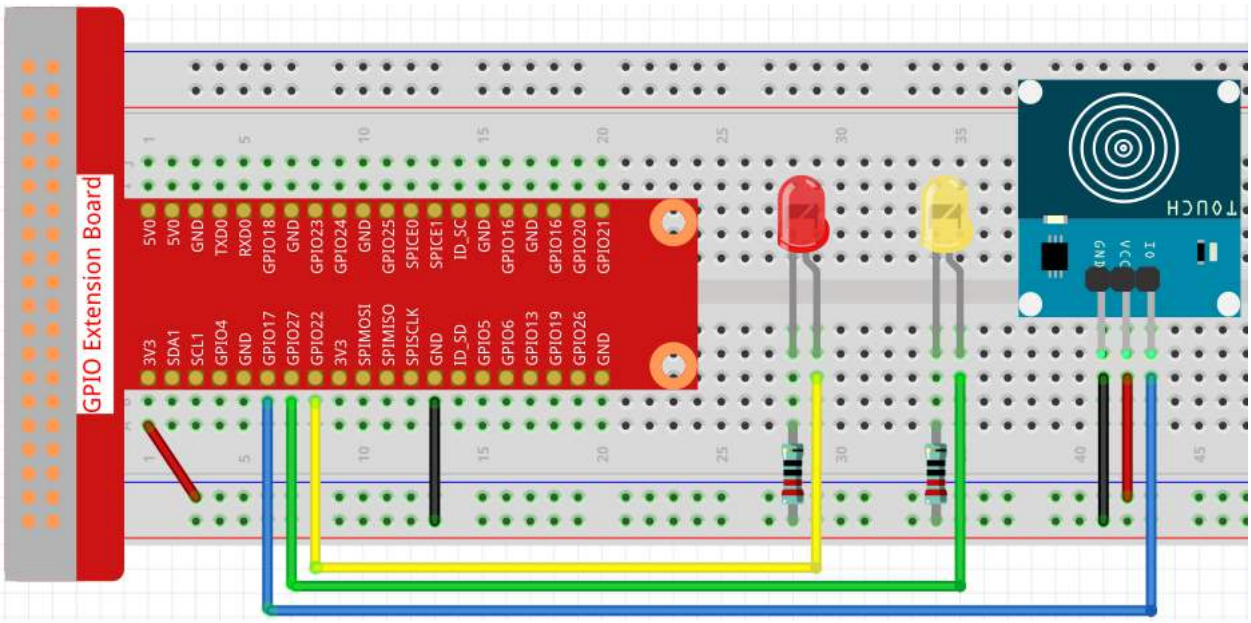
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Touch Switch Module*

Schematic Diagram



Experimental Procedures

Step 1:: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.3_TouchSwitch.py
```

While the code is running, the red LED lights up; when you tap on the touch switch module, the yellow LED turns on.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import time

# Set #17 as touch switch pin, #22 as led1 pin, #27 as led2 pin
touchPin = 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 touchPin input
    # Set ledPin output,
    # and initial level to High(3.3v)
    GPIO.setup(touchPin, 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:
        # touch switch high, led1 on
        if GPIO.input(touchPin) == 1:
            print ('You touch it!')
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)

        # touch switch low, led2 on
        if GPIO.input(touchPin) == 0:
            GPIO.output(led2Pin, GPIO.LOW)
            GPIO.output(led1Pin, GPIO.HIGH)

        time.sleep(0.5)
# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(led1Pin, GPIO.HIGH)
    GPIO.output(led2Pin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

(continues on next page)

(continued from previous page)

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

Code Explanation

```
touchPin = 17
led1Pin = 22
led2Pin = 27
```

`touchPin`, `led1Pin` and `led2Pin` connects to the GPIO17, GPIO22 and GPIO27, namely BCM17, BCM22 and BCM27.

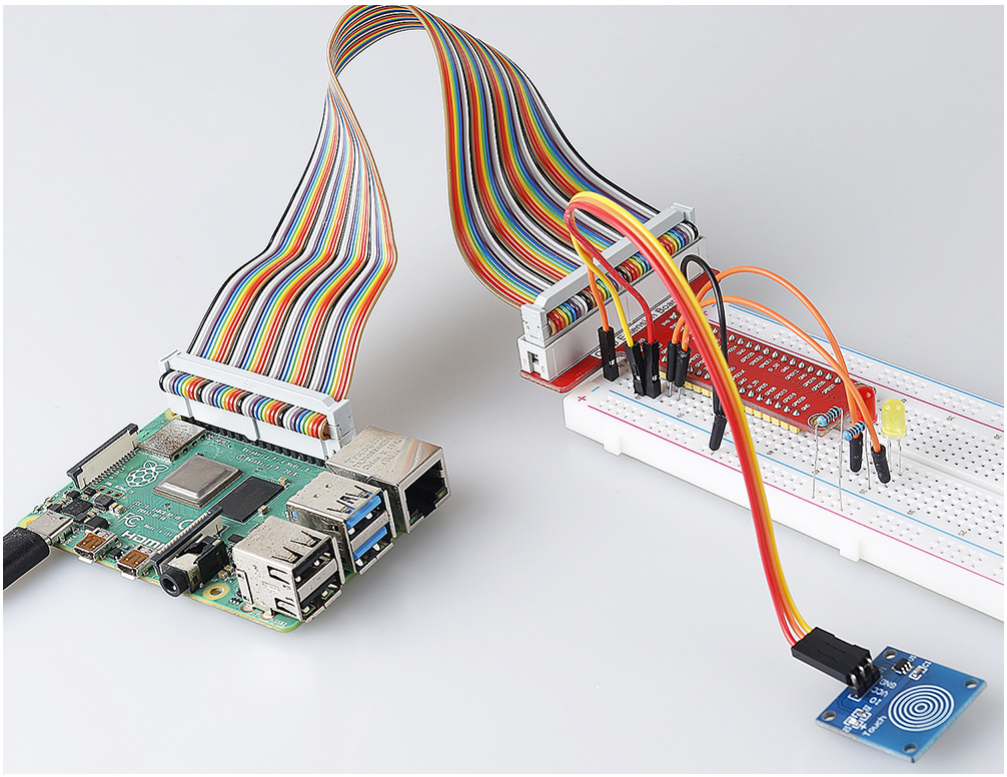
```
GPIO.setmode(GPIO.BCM)
    GPIO.setup(touchPin, GPIO.IN)
    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.HIGH)
```

Set the GPIO modes to BCM Numbering. Set `led1Pin`, `led2Pin` to output mode and initial their level to High (3.3v).

```
# touch switch high, led1 on
if GPIO.input(touchPin) == 1:
    print ('You touch it!')
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.HIGH)

# touch switch low, led2 on
if GPIO.input(touchPin) == 0:
    GPIO.output(led2Pin, GPIO.LOW)
    GPIO.output(led1Pin, GPIO.HIGH)
```

When you tap on the touch switch module, `touchPin` is high, led1 will light up and print “You touch it!”. When touchPin is low, led2 will light up.

Phenomenon Picture

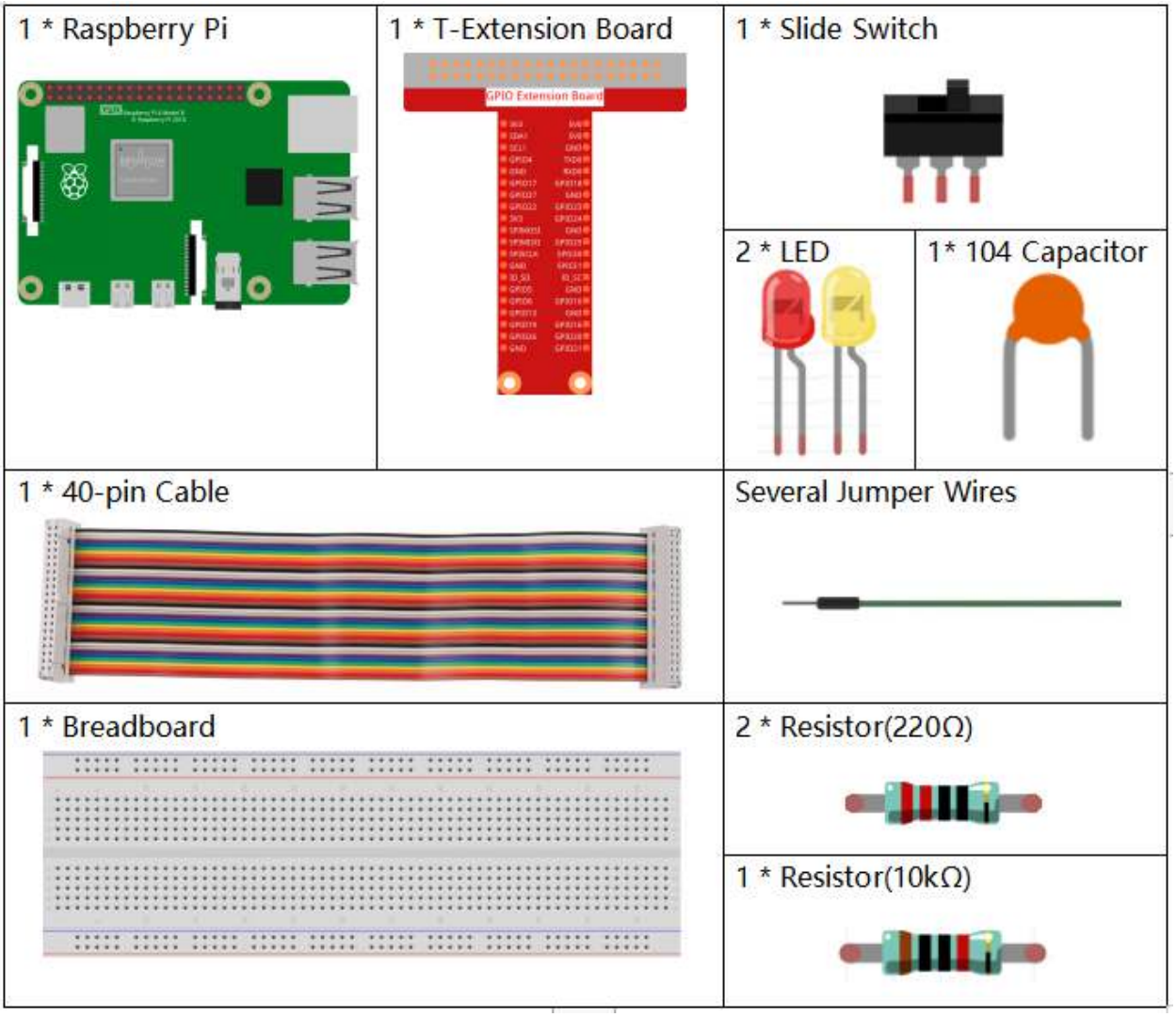


2.1.4 Slide Switch

Introduction

In this project, we will learn how to use a slide switch. Usually,the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.

Components

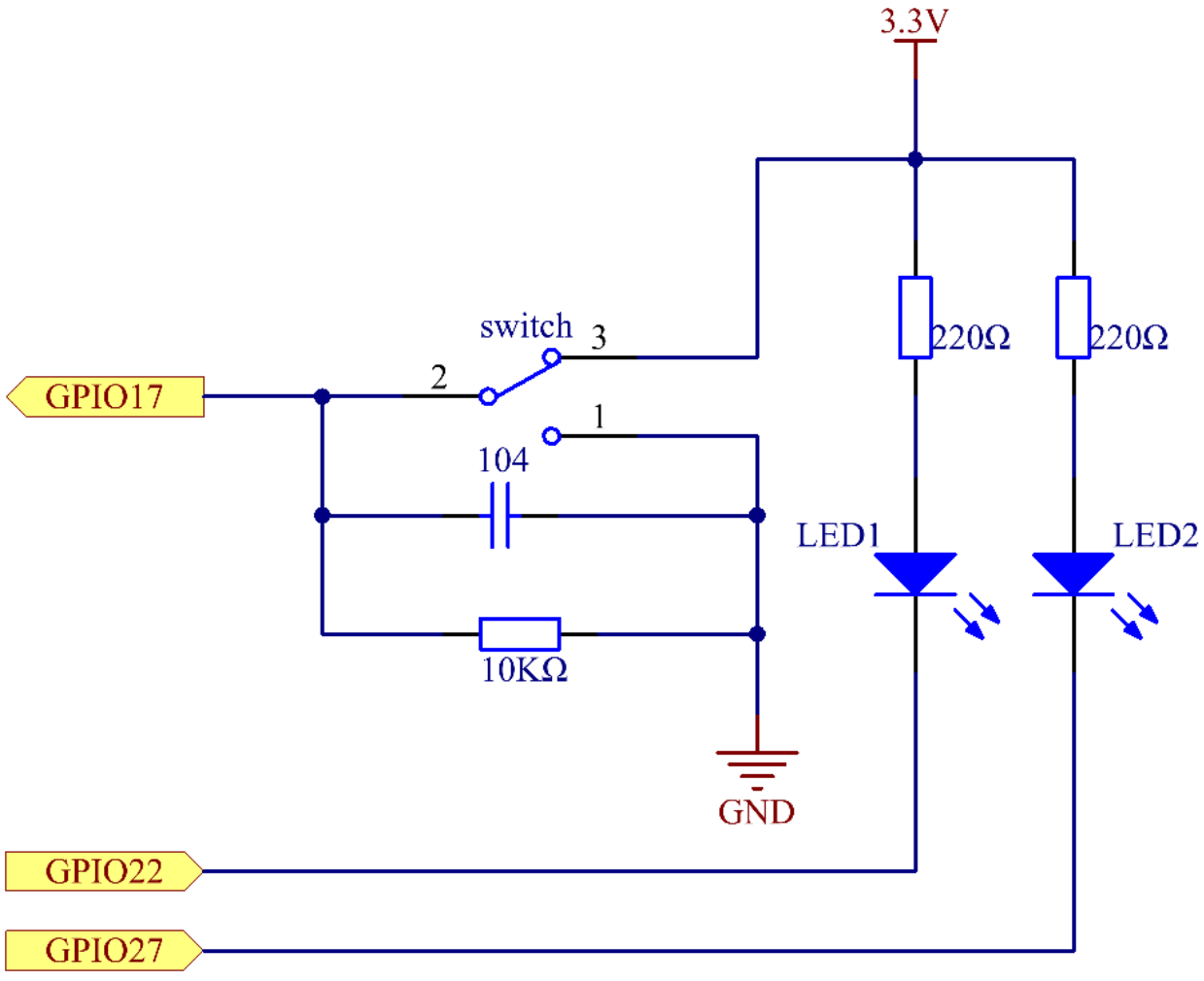


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Slide Switch*
- *Capacitor*

Schematic Diagram

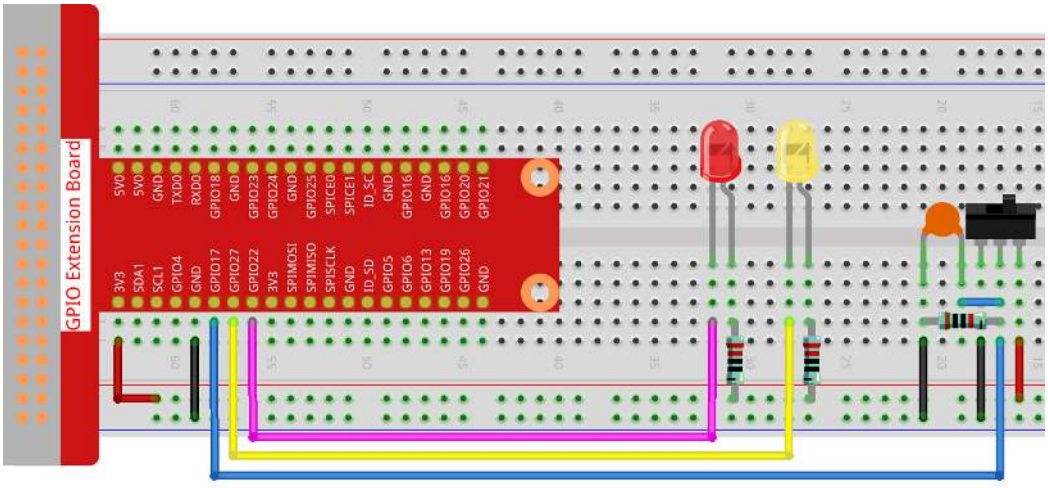
Connect the middle pin of the Slide Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 2.1.4_Slider.py
```

While the code is running, get the switch connected to the left, then the yellow LED lights up; to the right, the red light turns on.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as slide switch pin, GPIO22 as led1 pin, GPIO27 as led2 pin
slidePin = 17
led1Pin = 22
led2Pin = 27

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set slidePin input
    # Set ledPin output,
    # and initial level to High(3.3v)
    GPIO.setup(slidePin, GPIO.IN)
    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.HIGH)
```

(continues on next page)

(continued from previous page)

```
# Define a main function for main process
def main():
    while True:
        # slide switch high, led1 on
        if GPIO.input(slidePin) == 1:
            print ('    LED1 ON    ')
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)

        # slide switch low, led2 on
        if GPIO.input(slidePin) == 0:
            print ('    LED2 ON    ')
            GPIO.output(led2Pin, GPIO.LOW)
            GPIO.output(led1Pin, GPIO.HIGH)

        time.sleep(0.5)
# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(led1Pin, GPIO.HIGH)
    GPIO.output(led2Pin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

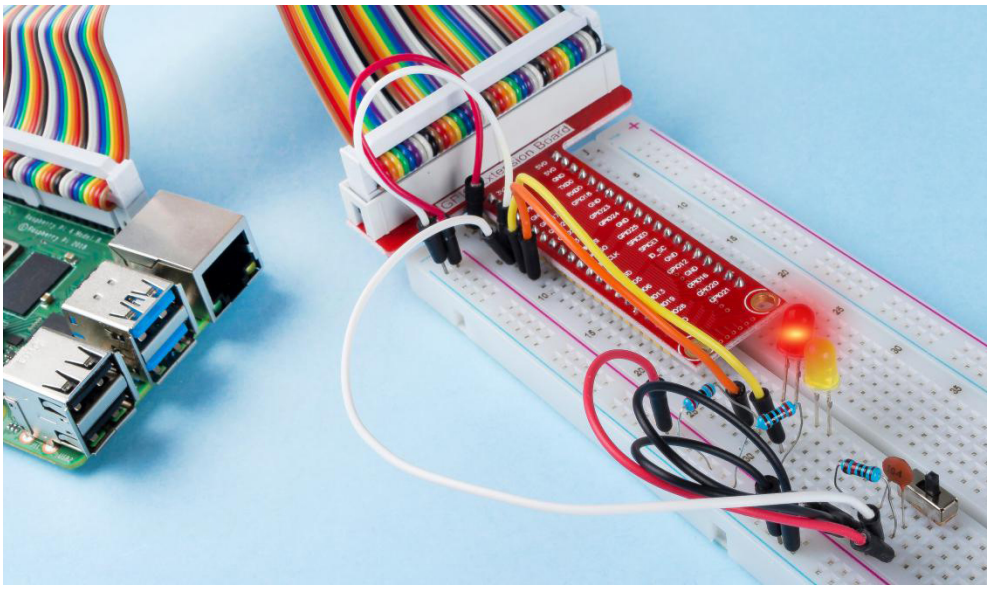
```
if GPIO.input(slidePin) == 1:
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off.

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

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off.

Phenomenon Picture

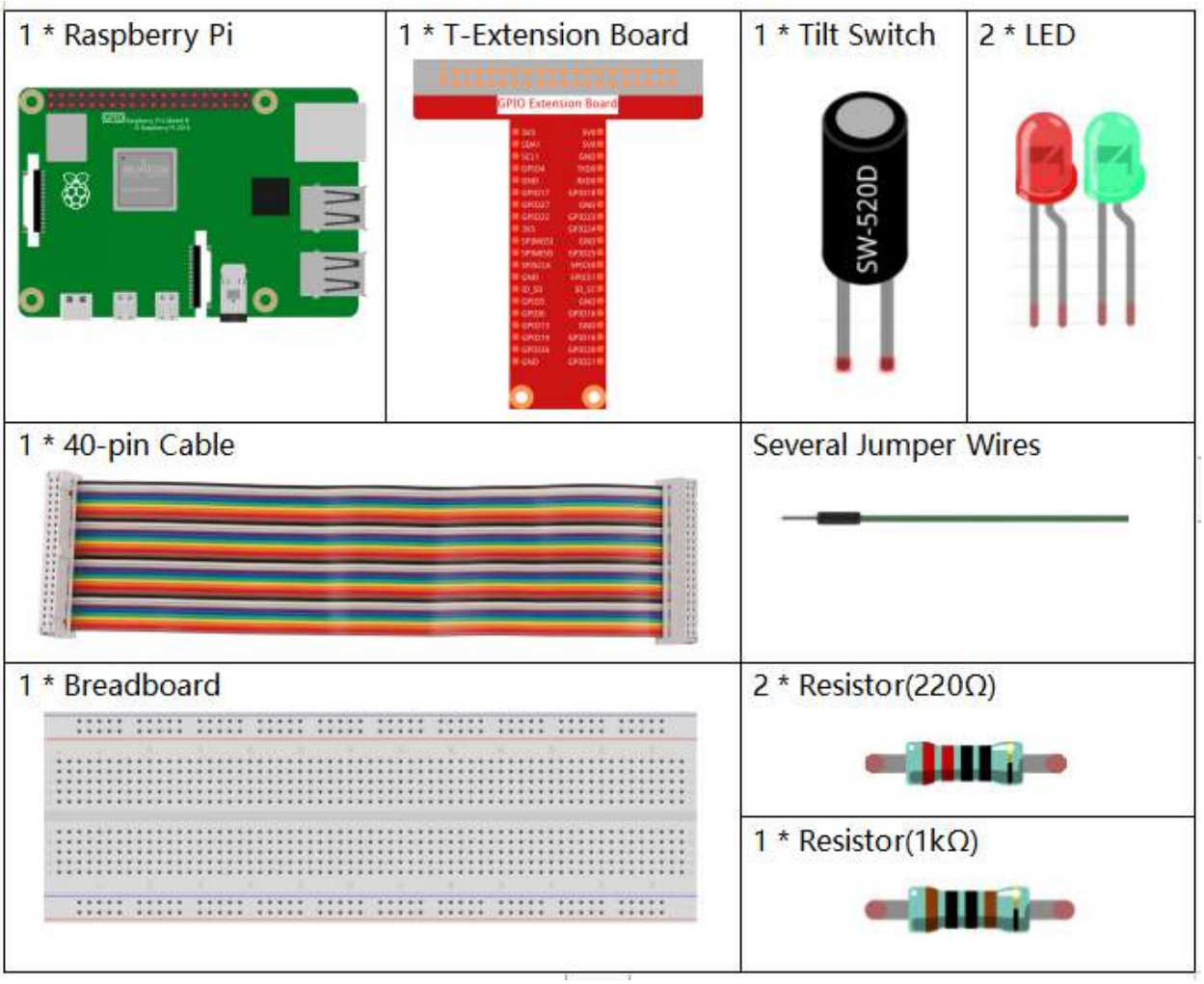


2.1.5 Tilt Switch

Introduction

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

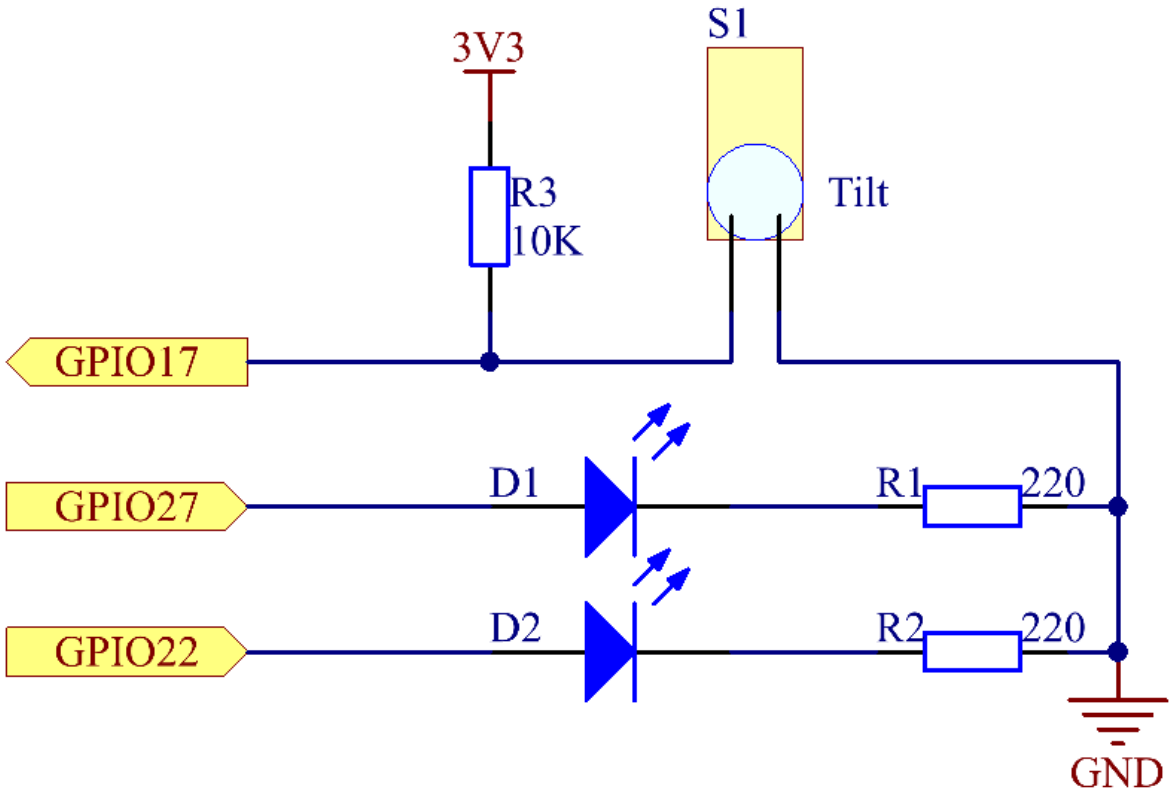
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Tilt Switch*

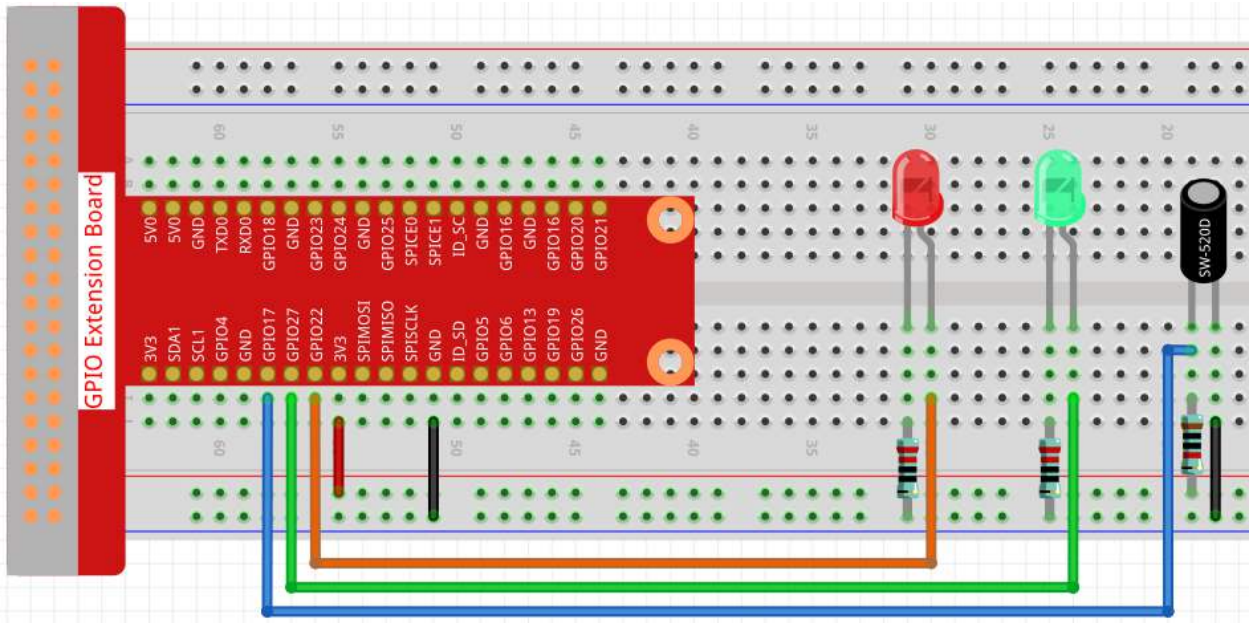
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.5_Tilt.py
```

Place the tilt vertically, and the green LED will turns on. If you tilt it, "Tilt!" will be printed on the screen and the red LED will turns on. Place it vertically again, and the green LED will lights on.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO

TiltPin = 17
Gpin    = 27
Rpin    = 22

def setup():
    GPIO.setmode(GPIO.BCM)        # Numbers GPIOs by physical location
    GPIO.setup(Gpin, GPIO.OUT)      # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)      # Set Red Led Pin mode to output
    GPIO.setup(TiltPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # Set BtnPin's mode is␣
↪input, and pull up to high level(3.3V)
    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
```

(continues on next page)

(continued from previous page)

```
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)

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

def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.HIGH)       # Green led off
    GPIO.output(Rpin, GPIO.HIGH)       # Red led off
    GPIO.cleanup()                     # Release resource

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

Code Explanation

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

Set up a detect on TiltPin, and callback function to detect.

```
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)
```

Define a function Led() to turn the two LEDs on or off. If x=0, the red LED lights up; otherwise, the green LED will be lit.

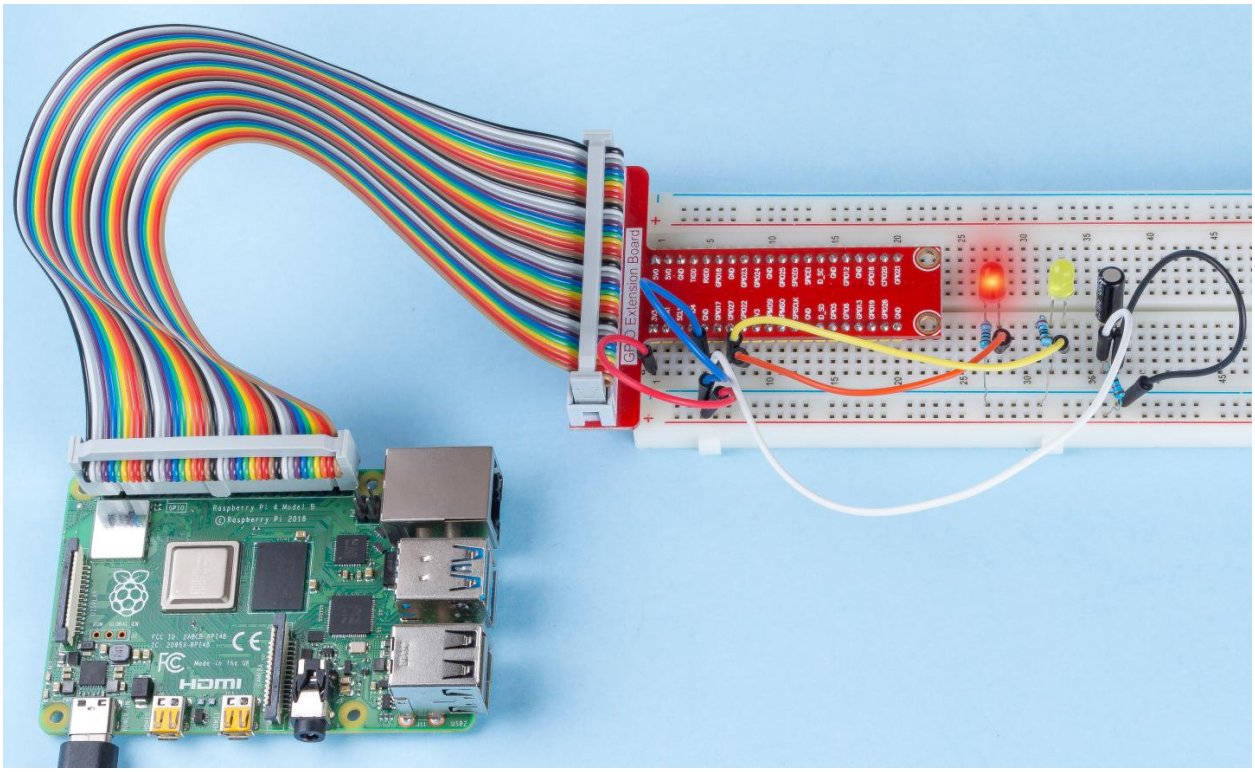
```
def Print(x):
    if x == 0:
        print ('    *************')
        print ('    *   Tilt!   *')
        print ('    *************')
```

Create a function, Print() to print the characters above on the screen.

```
def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))
```

Define a callback function for tilt callback. Get the read value of the tilt switch then the function Led() controls the turning on or off of the two LEDs that is depended on the read value of the tilt switch.

Phenomenon Picture

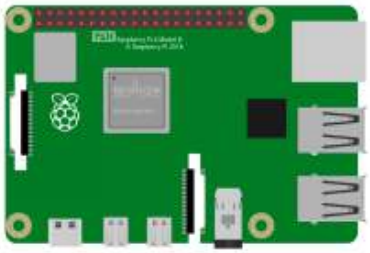
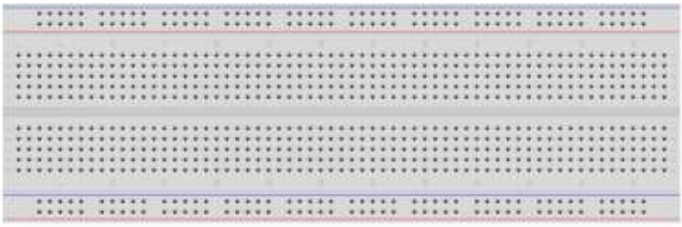


2.1.6 Rotary Encoder Module

Introduction

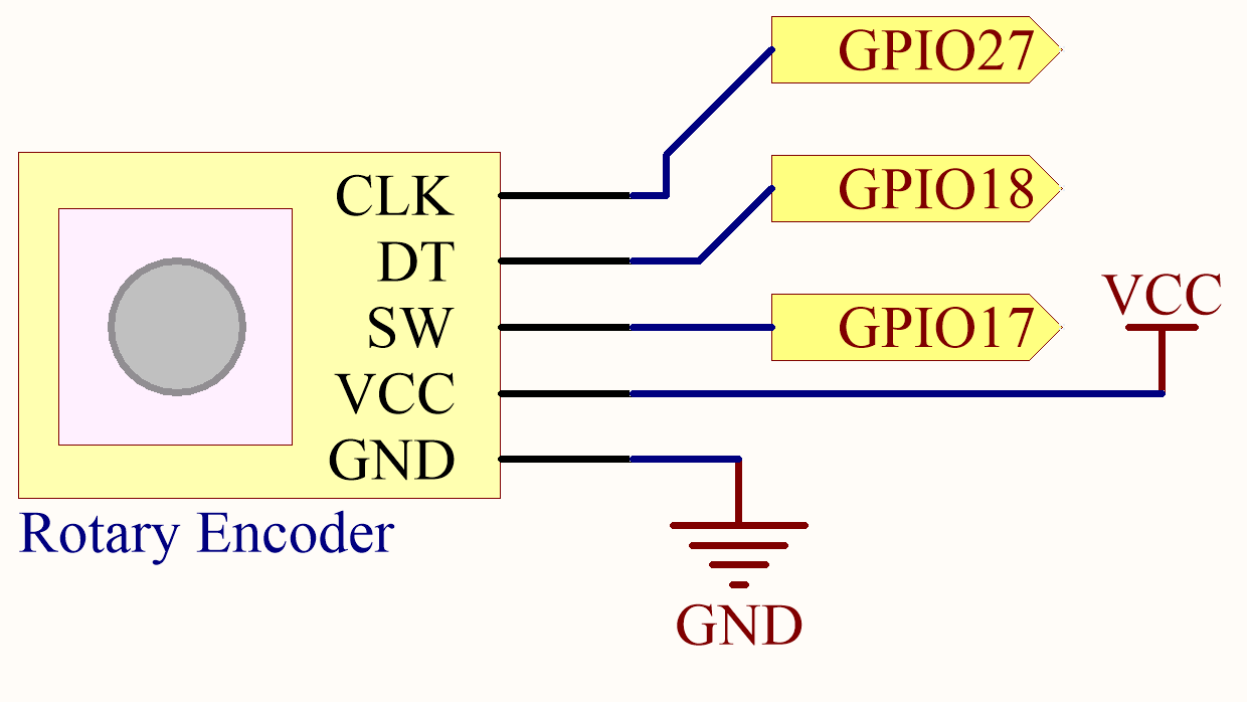
In this project, you will learn about Rotary Encoder. A rotary encoder is an electronic switch with a set of regular pulses in strictly timing sequence. When used with IC, it can achieve increment, decrement, page turning and other operations such as mouse scrolling, menu selection, and so on.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td rowspan="2">1 * Rotary Encoder
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td></tr>
</table>

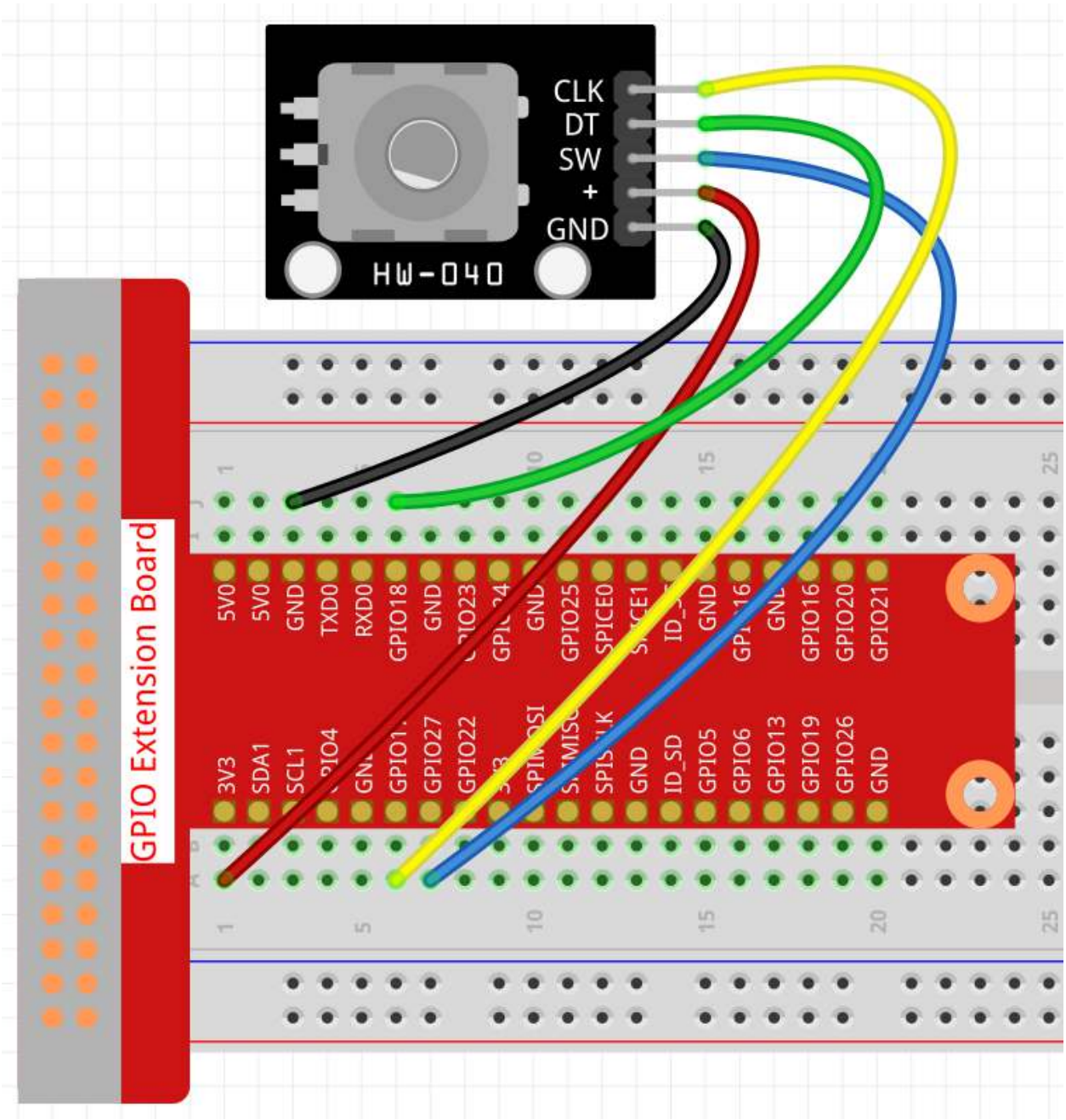
- *GPIO Extension Board*
- *Breadboard*
- *Rotary Encoder Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



In this example, we can connect the Rotary Encoder pin directly to the Raspberry Pi using a breadboard and 40-pin Cable, connect the GND of the Rotary Encoder to GND, +to 5V, SW to digital GPIO27, DT to digital GPIO18, and CLK to digital GPIO 17.

Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.6_RotaryEncoder.py
```

You will see the count on the shell. When you turn the rotary encoder clockwise, the count is increased; when turn

it counterclockwise, the count is decreased. If you press the switch on the rotary encoder, the readings will return to zero.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time

clkPin = 17    # CLK Pin
dtPin = 18    # DT Pin
swPin = 27    # Button Pin

globalCounter = 0

flag = 0
Last_dt_Status = 0
Current_dt_Status = 0

def setup():
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by physical location
    GPIO.setup(clkPin, GPIO.IN)    # input mode
    GPIO.setup(dtPin, GPIO.IN)
    GPIO.setup(swPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def rotaryDeal():
    global flag
    global Last_dt_Status
    global Current_dt_Status
    global globalCounter
    Last_dt_Status = GPIO.input(dtPin)
    while(not GPIO.input(clkPin)):
        Current_dt_Status = GPIO.input(dtPin)
        flag = 1
    if flag == 1:
        flag = 0
        if (Last_dt_Status == 0) and (Current_dt_Status == 1):
            globalCounter = globalCounter - 1
        if (Last_dt_Status == 1) and (Current_dt_Status == 0):
            globalCounter = globalCounter + 1

def swISR(channel):
    global globalCounter
    globalCounter = 0

def loop():
    global globalCounter
    tmp = 0    # Rotary Temperary

    GPIO.add_event_detect(swPin, GPIO.FALLING, callback=swISR)
    while True:
        rotaryDeal()
        if tmp != globalCounter:
```

(continues on next page)

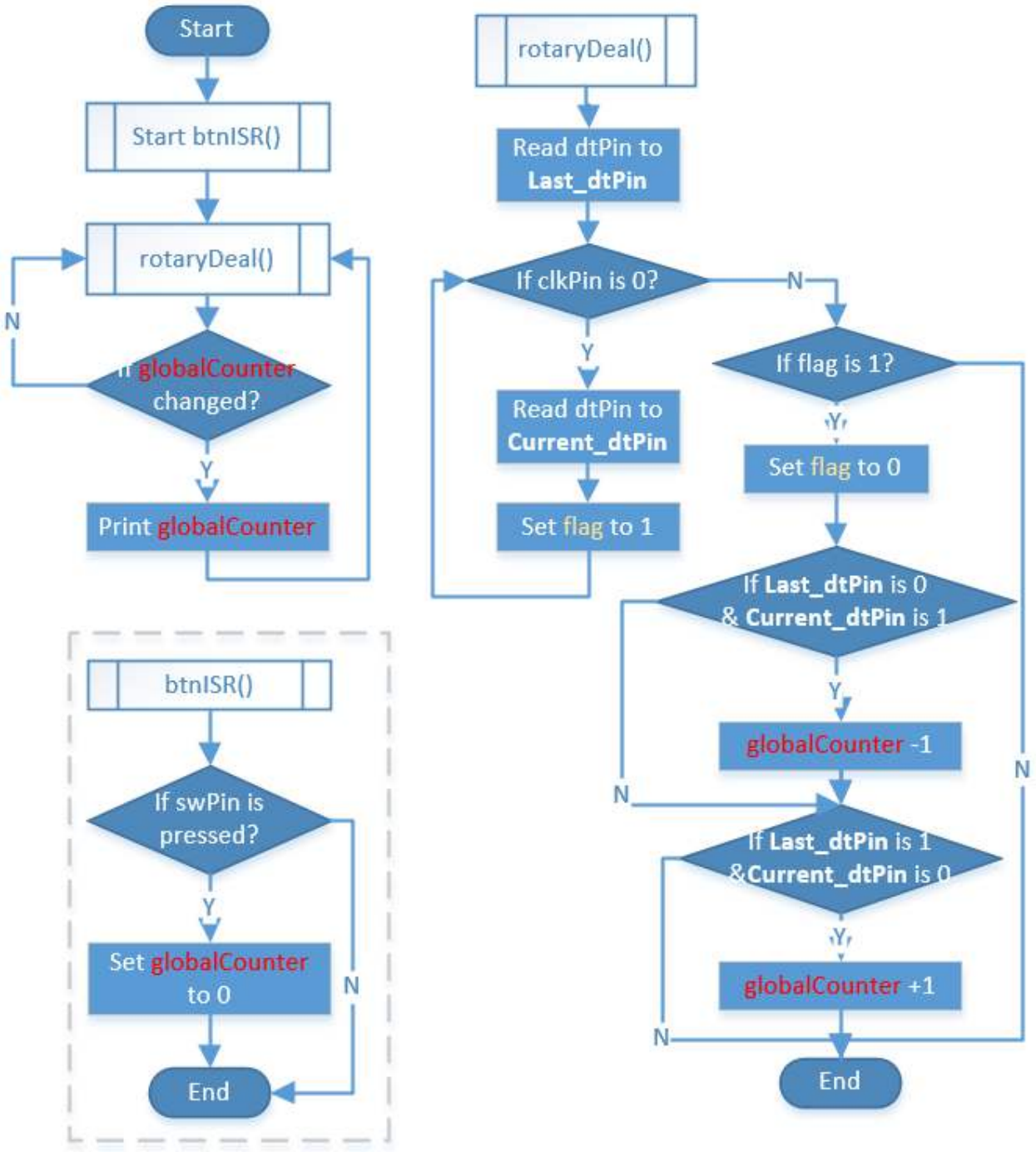
(continued from previous page)

```
        print ('globalCounter = %d' % globalCounter)
        tmp = globalCounter

def destroy():
    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()
```

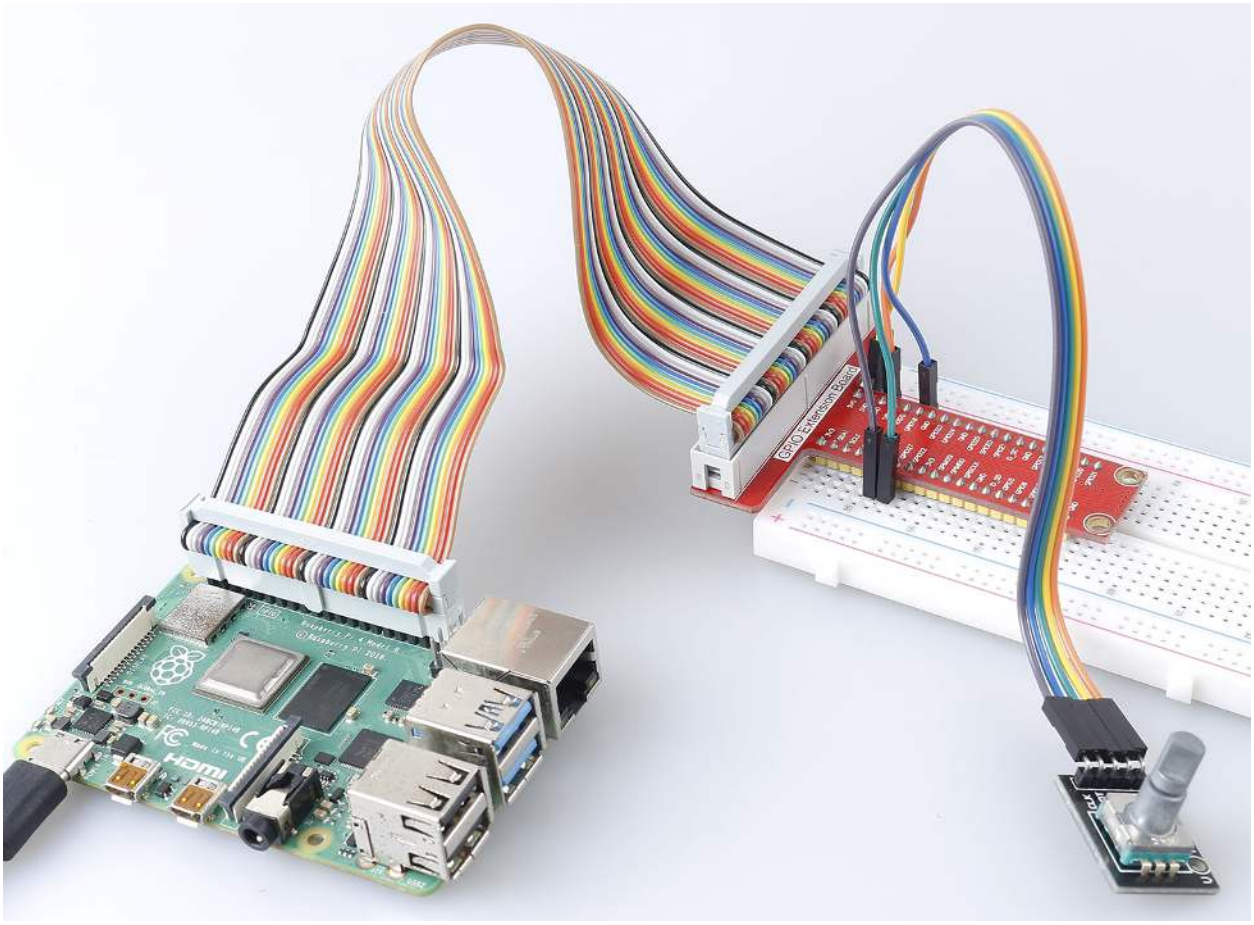
Code Analysis

- Read dtPin value when clkPin is low.
- When clkPin is high, if dtPin goes from low to high, the count decreases, otherwise the count increases.
- swPin will output low when the shaft is pressed.

From this, the program flow is shown below:


Start
Start btnISR()
rotaryDeal()
N
globalCounter changed?
Y
Print globalCounter
btnISR()
If swPin is pressed?
Y
Set globalCounter to 0
N
End
rotaryDeal()
Read dtPin to Last_dtPin
If clkPin is 0?
N
Y
Read dtPin to Current_dtPin
Set flag to 1
If flag is 1?
Y
Set flag to 0
If Last_dtPin is 0 & Current_dtPin is 1
Y
globalCounter -1
N
If Last_dtPin is 1 &Current_dtPin is 0
Y
globalCounter +1
N
N
End

Phenomenon Picture

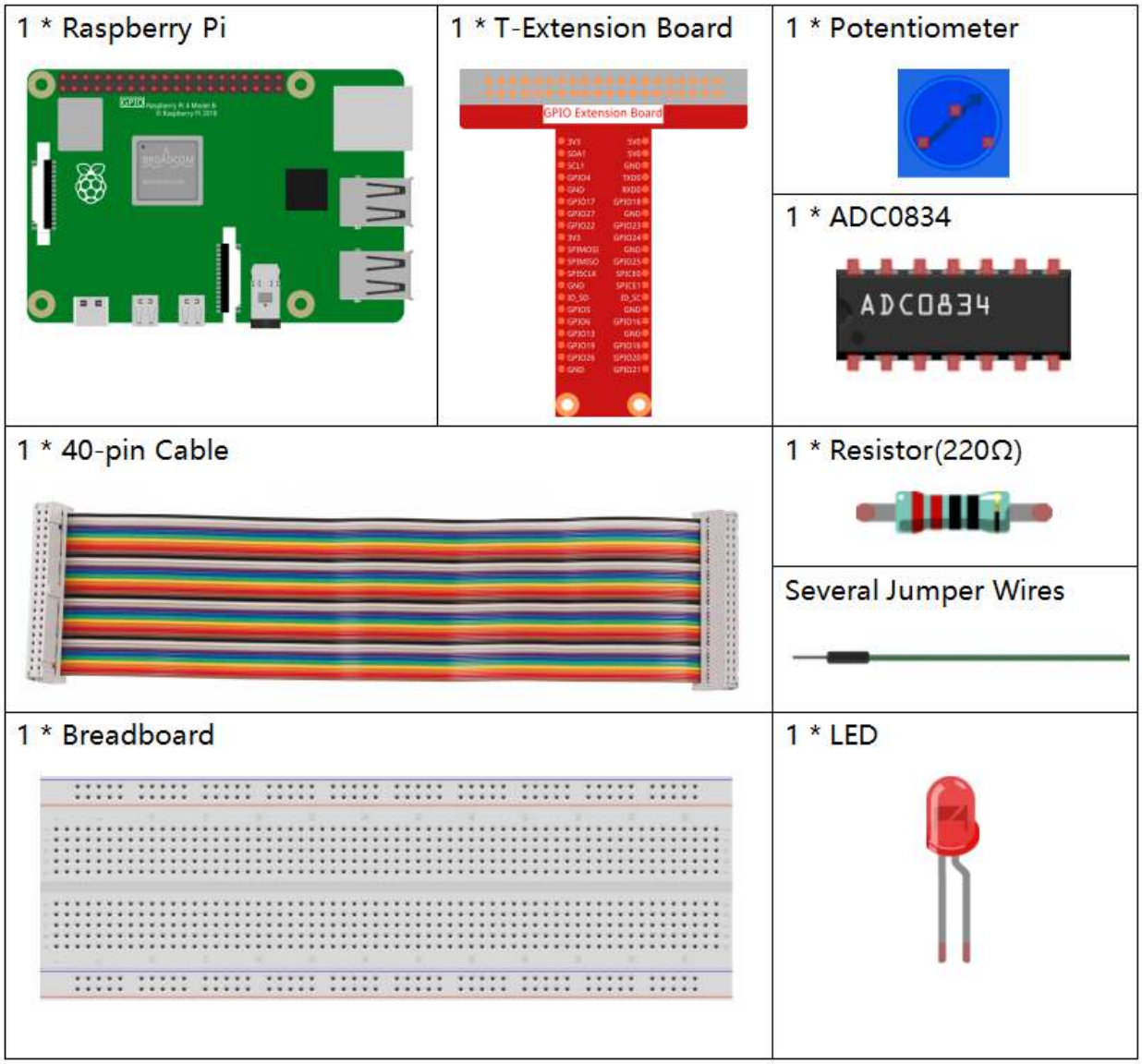


2.1.7 Potentiometer

Introduction

The ADC function can be used to convert analog signals to digital signals, and in this experiment, ADC0834 is used to get the function involving ADC. Here, we implement this process by using potentiometer. Potentiometer changes the physical quantity – voltage, which is converted by the ADC function.

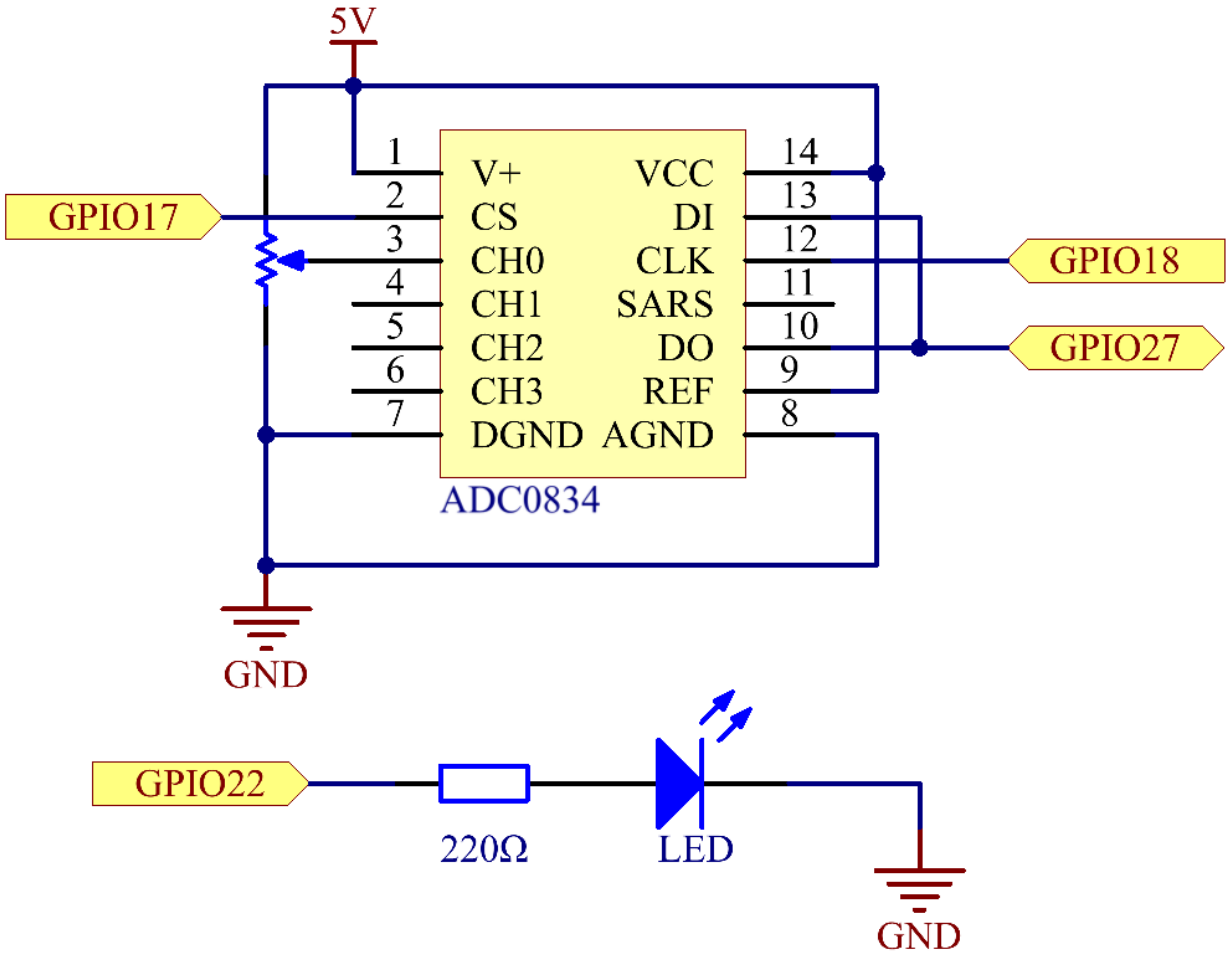
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Potentiometer*
- *ADC0834*

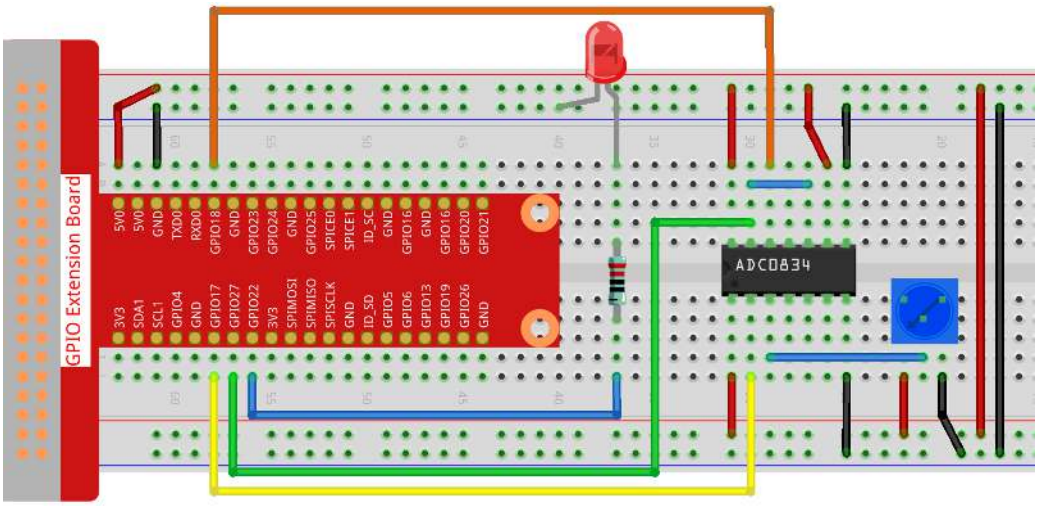
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Note: Please place the chip by referring to the corresponding position depicted in the picture. Note that the grooves on the chip should be on the left when it is placed.

Step 2: Open the code file

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.7_Potentiometer.py
```

After the code runs, rotate the knob on the potentiometer, the intensity of LED will change accordingly.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import ADC0834
import time

LedPin = 22

def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
```

(continues on next page)

(continued from previous page)

```
    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)

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

def destroy():
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()

def loop():
    while True:
        res = ADC0834.getResult()
        print ('res = %d' % res)
        R_val = MAP(res, 0, 255, 0, 100)
        led_val.ChangeDutyCycle(R_val)
        time.sleep(0.2)

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

Code Explanation

```
import ADC0834
```

import ADC0834 library. You can check the content of the library by calling the command nano ADC0834.py.

```
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)
```

In setup(), define the naming method as BCM, set LedPin as PWM channel and render it a frequency of 2Khz.

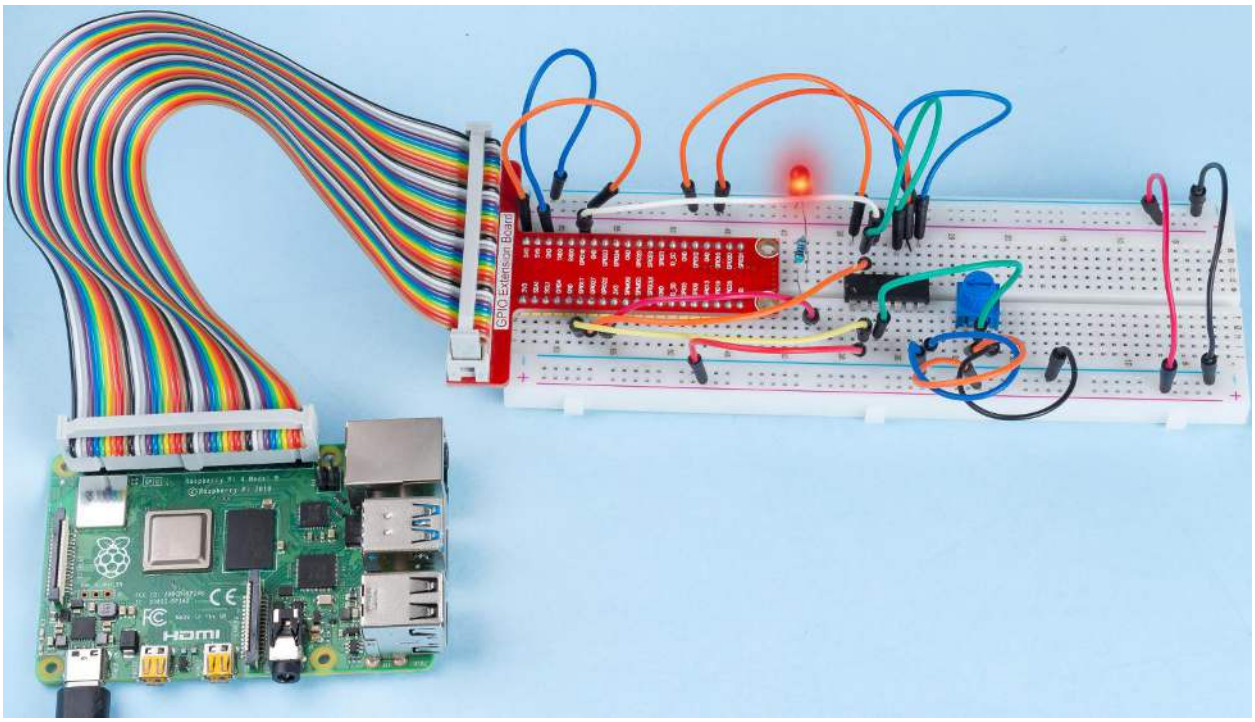
ADC0834.setup(): Initialize ADC0834, and connect the defined CS, CLK, DIO of ADC0834 to GPIO17, GPIO18 and GPIO27 respectively.

```
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)
```

The function getResult() is used to read the analog values of the four channels of ADC0834. By default, the function reads the value of CH0, and if you want to read other channels, please input the channel number in ( ), ex. getResult(1).

The function loop() first reads the value of CH0, then assign the value to the variable res. After that, call the function MAP to map the read value of potentiometer to 0~100. This step is used to control the duty cycle of LedPin. Now, you may see that the brightness of LED is changing with the value of potentiometer.

Phenomenon Picture

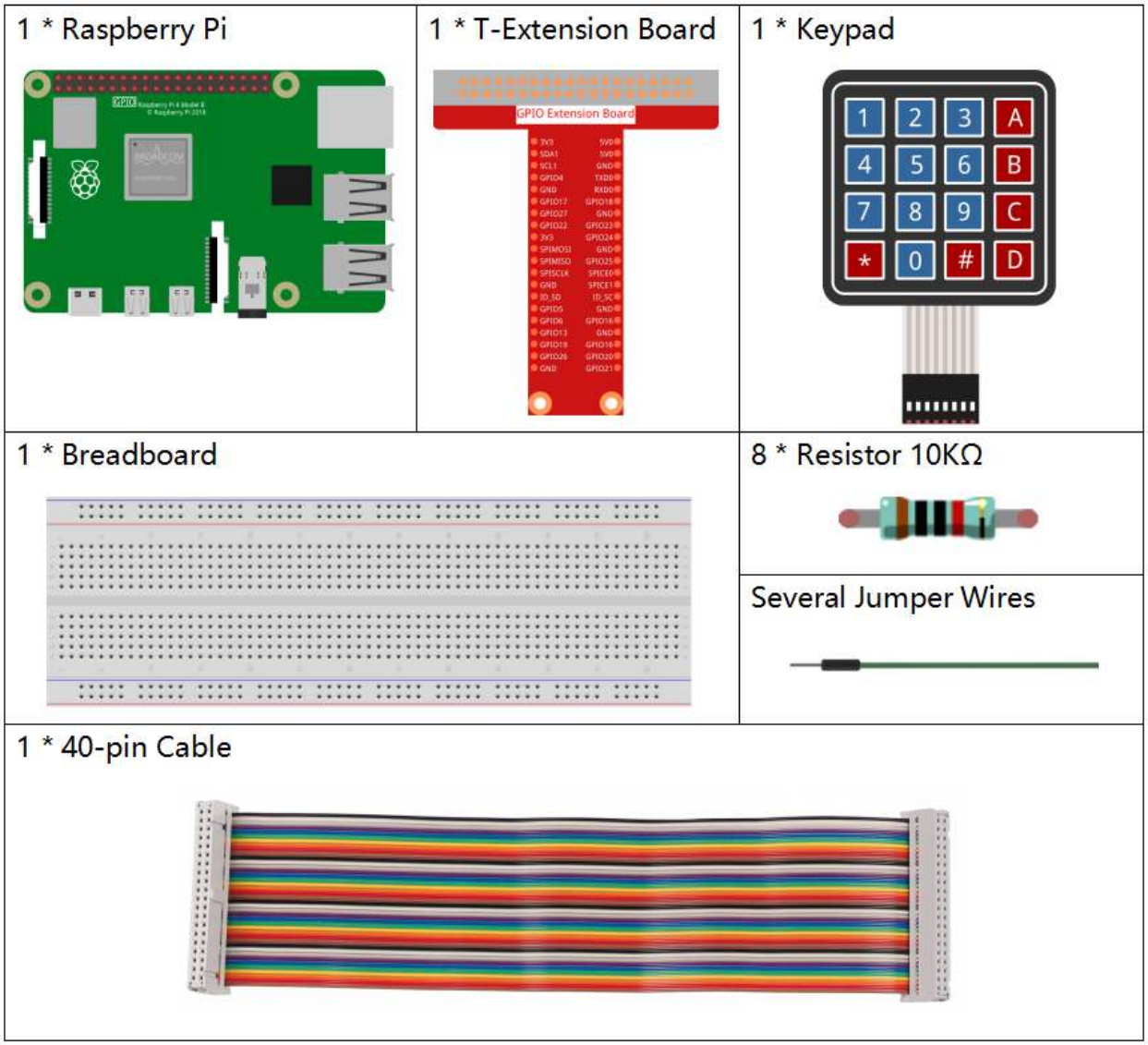


2.1.8 Keypad

Introduction

A keypad is a rectangular array of buttons. In this project, We will use it input characters.

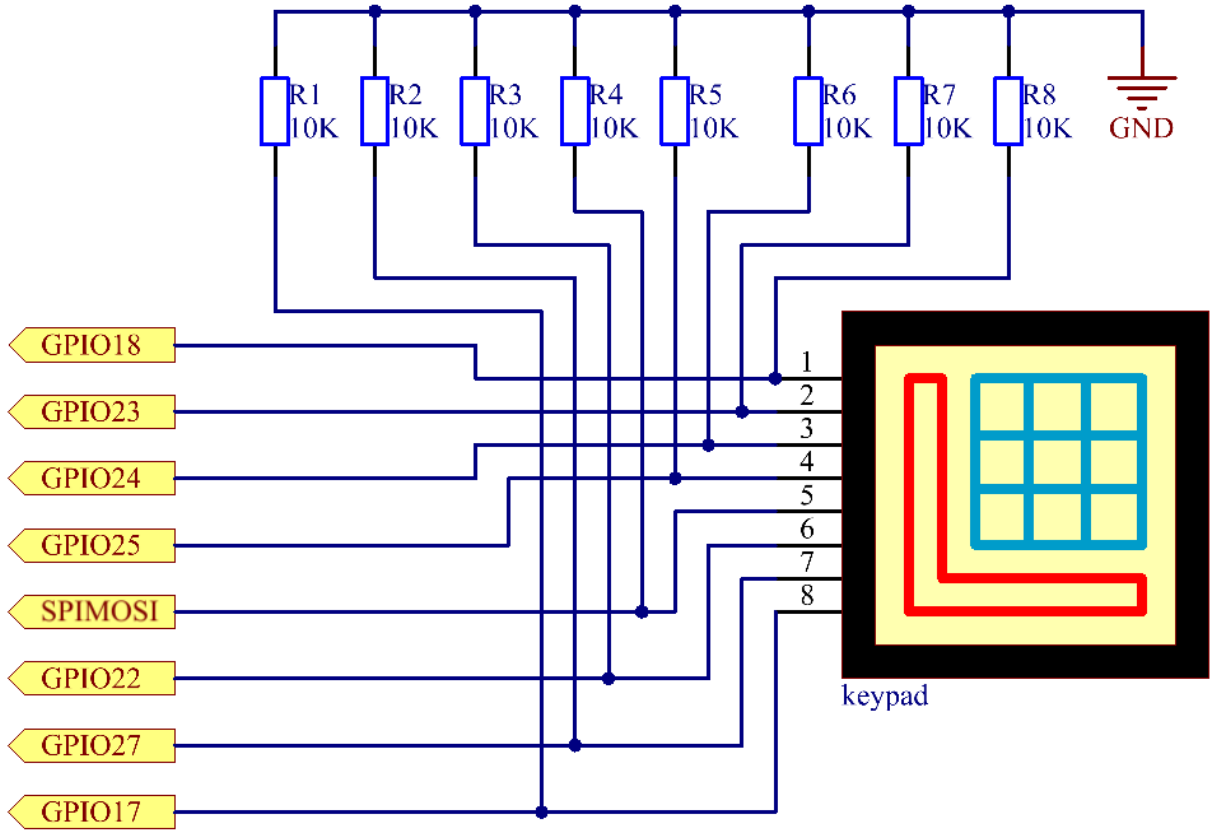
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Keypad*

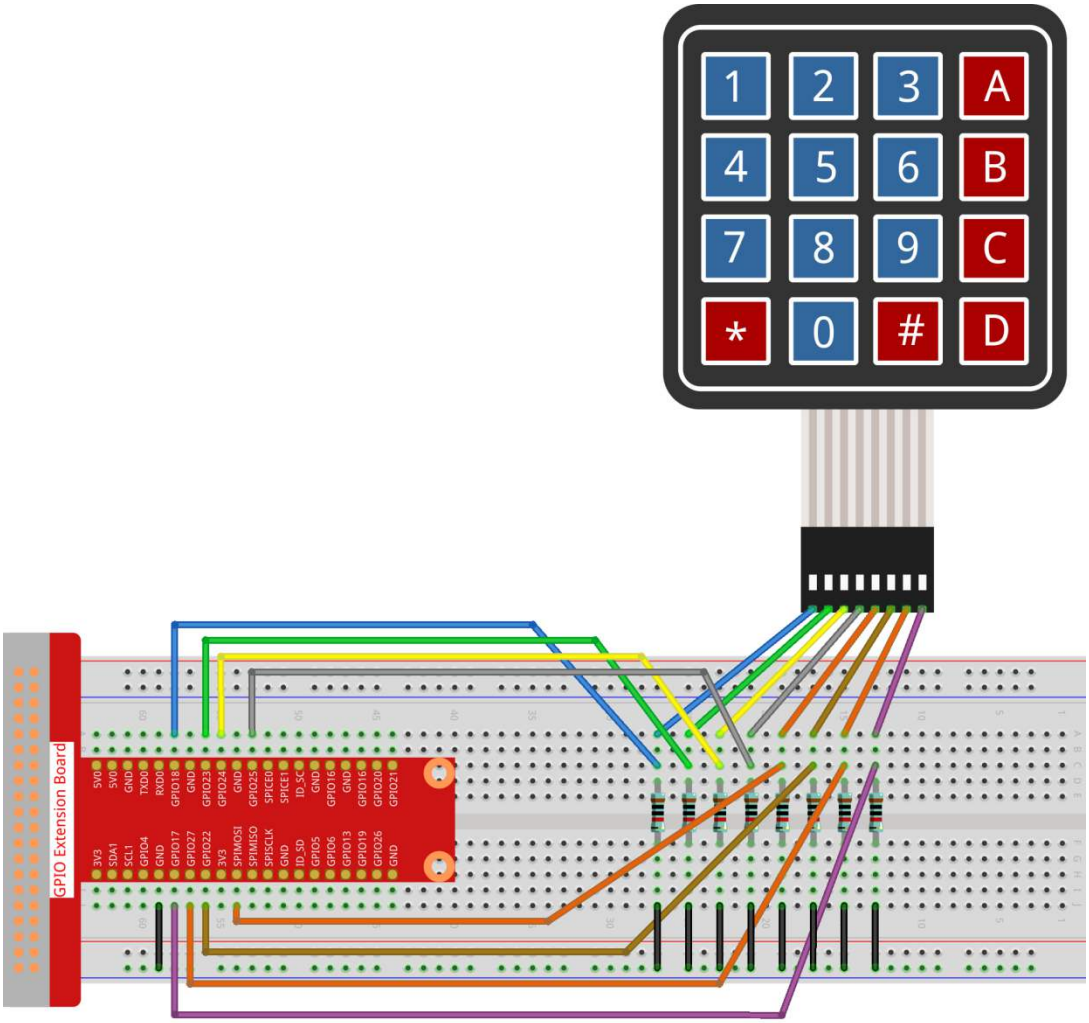
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.8_Keypad.py
```

After the code runs, the values of pressed buttons on keypad (button Value) will be printed on the screen.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

class Keypad():

    def __init__(self, rowsPins, colsPins, keys):
```

(continues on next page)

(continued from previous page)

```
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys

def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
    colsPins = [10,22,27,17]
    keys = ["1","2","3","A",
            "4","5","6","B",
            "7","8","9","C",
            "*","0","#","D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []

def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)

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

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

Code Explanation

```
def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
```

(continues on next page)

(continued from previous page)

```
    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 = []
```

Declare each key of the matrix keyboard to the array keys[] and define the pins on each row and column.

```
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)
```

This is the part of the main function that reads and prints the button value.

The function `keyRead()` will read the state of every button.

The statement `if len(pressed_keys) != 0` and `last_key_pressed != pressed_keys` is used to judge

whether a key is pressed and the state of the pressed button. (If you press '3' when you press '1', the judgement is tenable.)

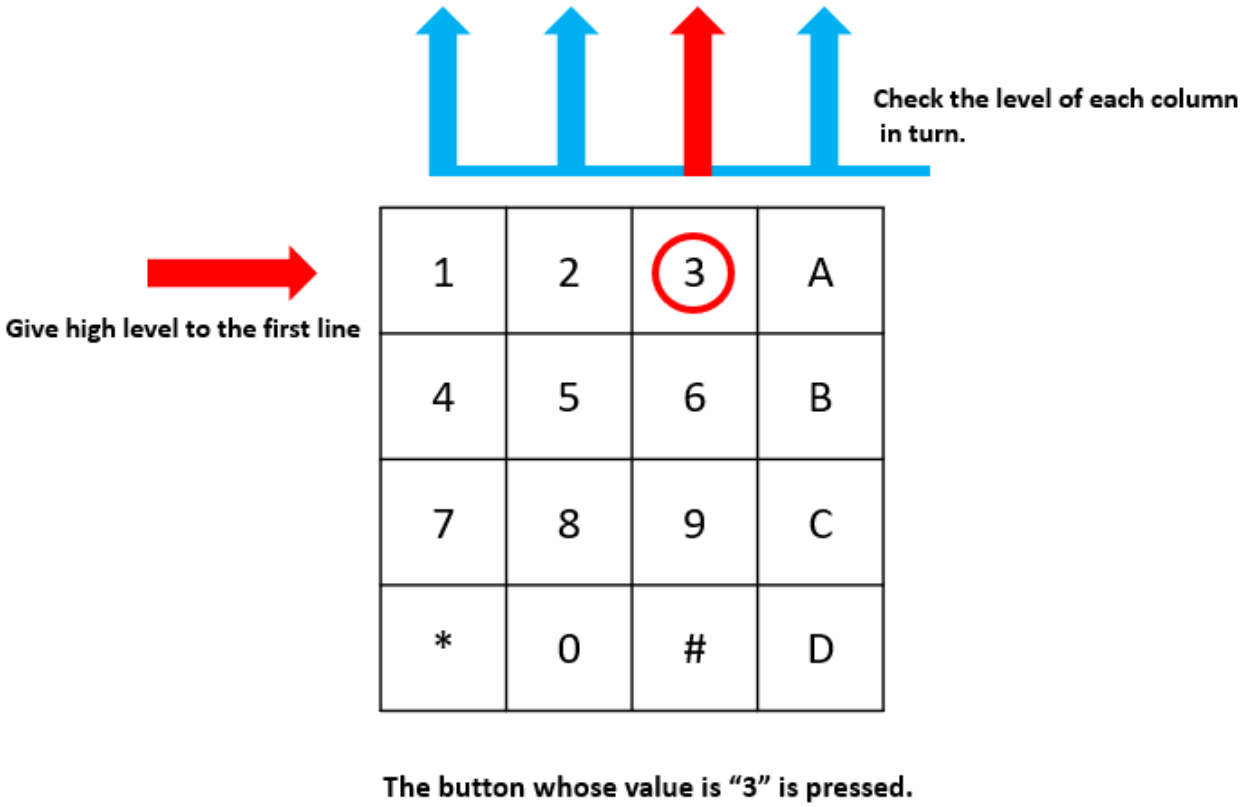
Prints the value of the currently pressed key when the condition is tenable.

The statement `last_key_pressed = pressed_keys` assigns the state of each judgment to an array `last_key_pressed` to facilitate the next round of conditional judgment.

```
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
```

This function assigns a high level to each row in turn, and when the button in the column is pressed, the column in which the key is located gets a high level. After the two-layer loop is judged, the value of the button whose state is 1 is stored in the array `pressed_keys`.

If you press the key '3':



rowPins[0] is written in high level, and colPins[2] gets high level.

colPins[0]colPins[1]colPins[3] get low level.

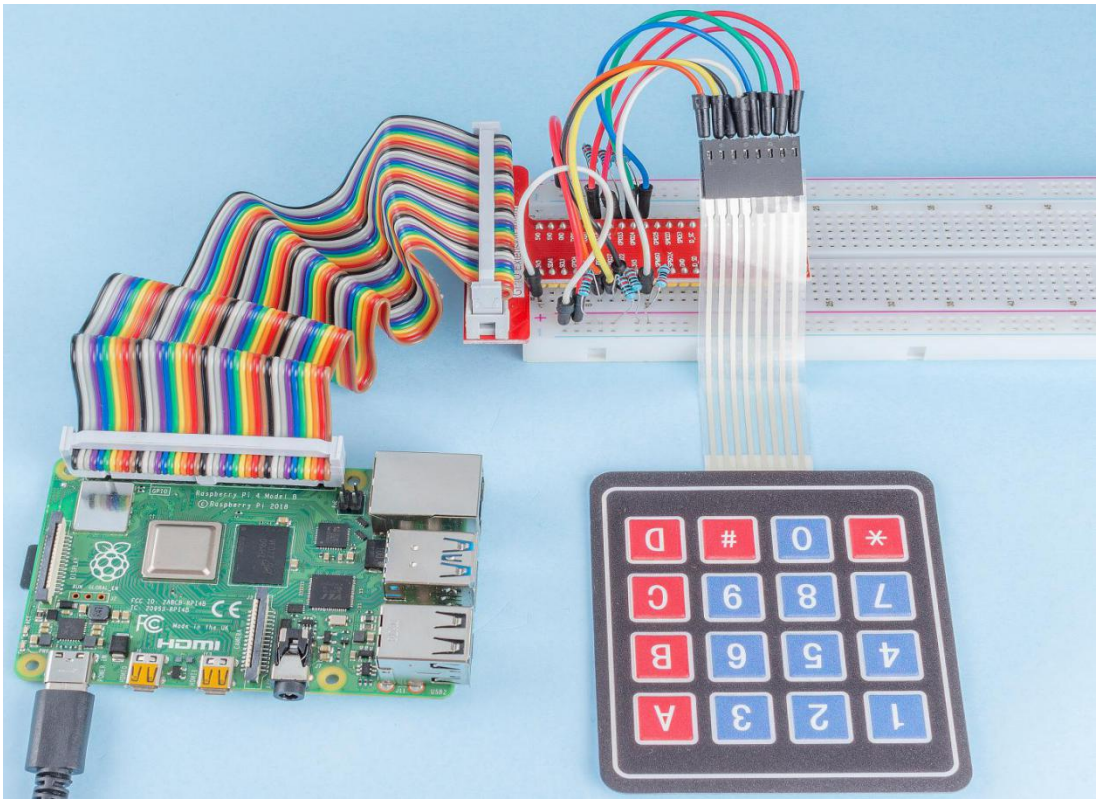
There are four states: 0, 0, 1, 0; and we write ‘3’ into `pressed_keys`.

When rowPins[1] , rowPins[2] , rowPins[3] are written into high level, colPins[0] ~ colPins[4] get low level.

The loop stopped, there returns `pressed_keys = '3'`.

If you press the buttons ‘1’ and ‘3’, there will return pressed_keys = [‘1’,’3’].

Phenomenon Picture

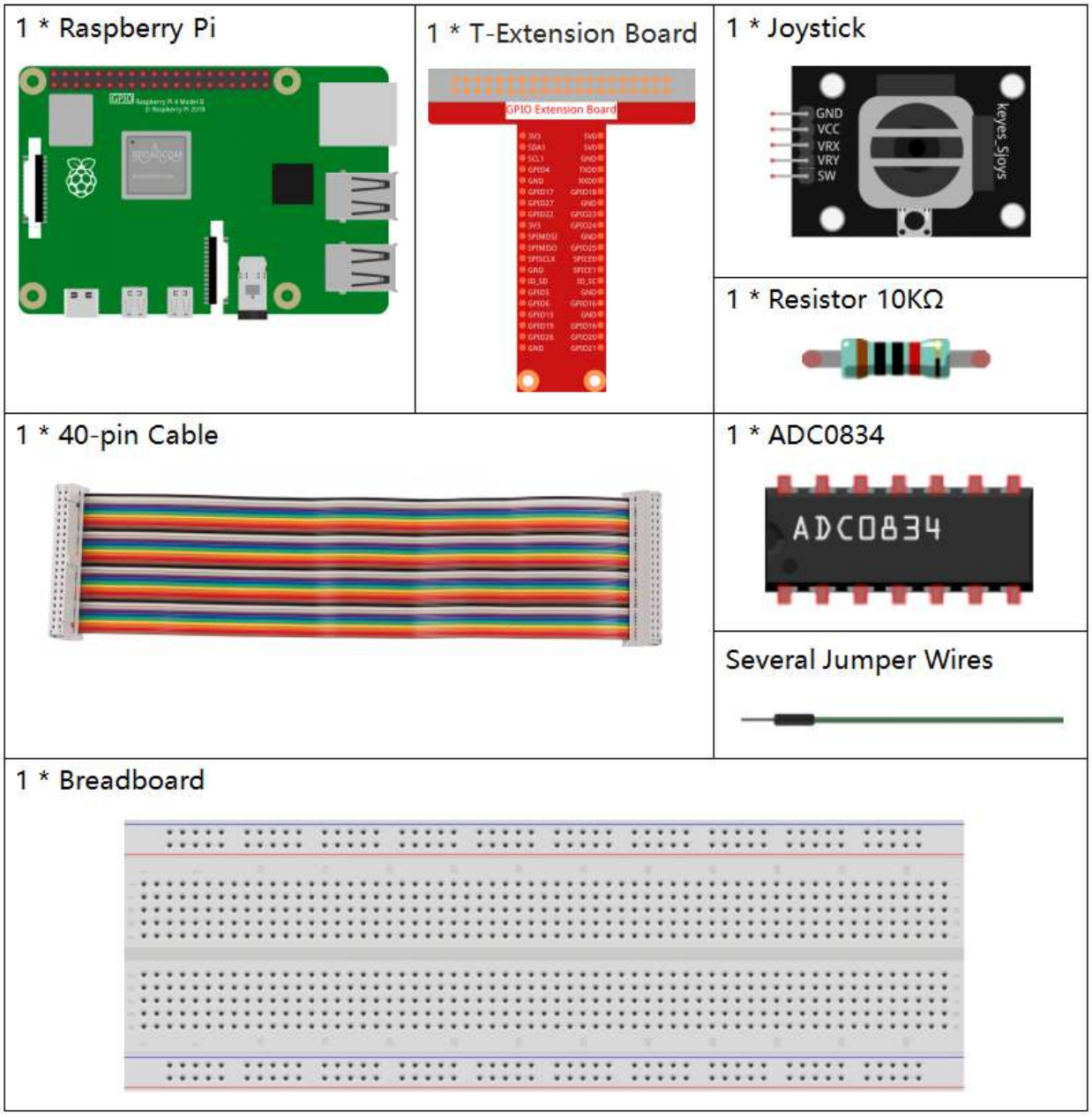


2.1.9 Joystick

Introduction

In this project, We're going to learn how joystick works. We manipulate the Joystick and display the results on the screen.

Components

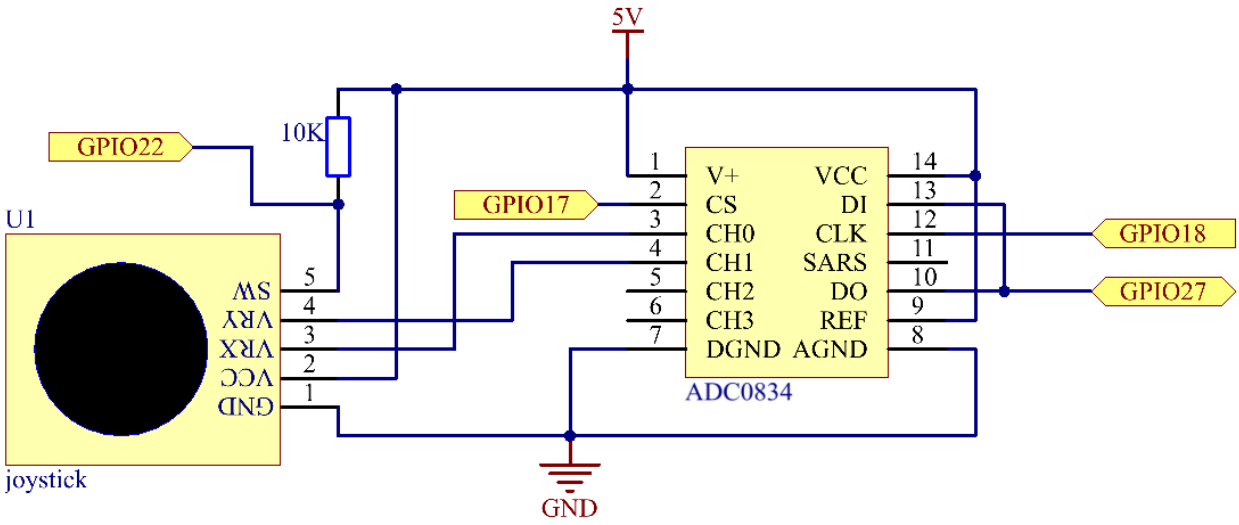


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Joystick Module*
- *ADC0834*

Schematic Diagram

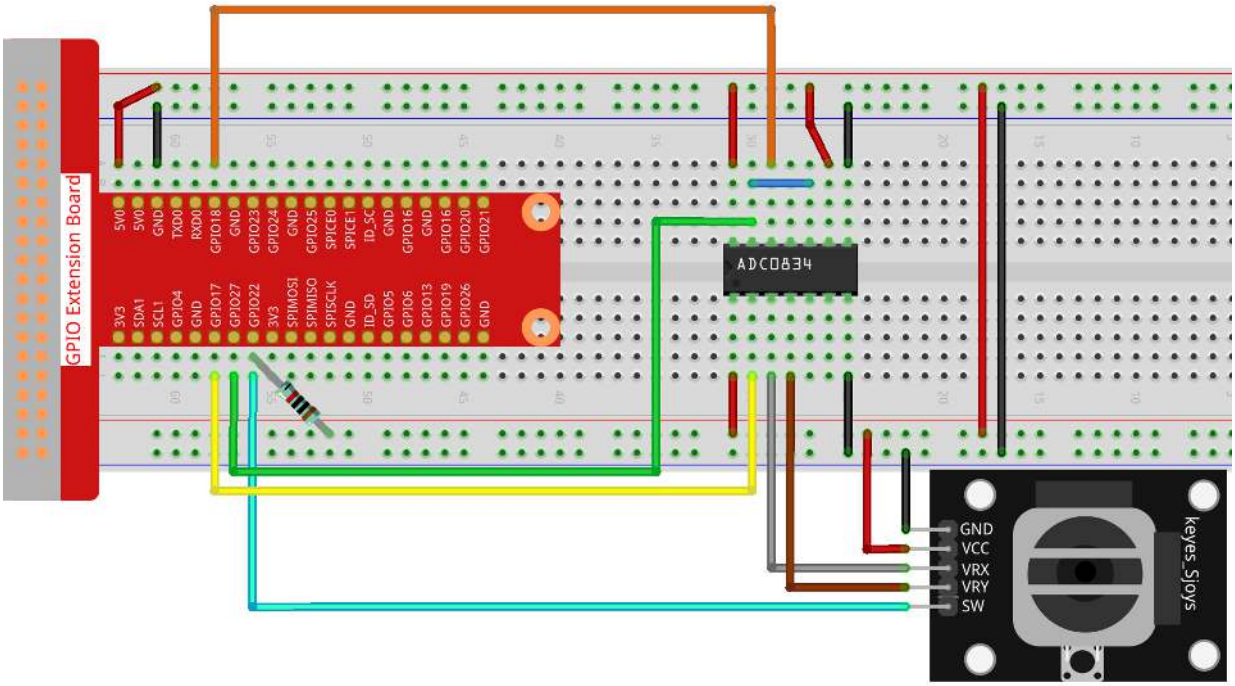
When the data of joystick is read, there are some differents between axis: data of X and Y axis is analog, which need to use ADC0834 to convert the analog value to digital value. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.1.9_Joystick.py
```

After the code runs, turn the Joystick, then the corresponding values of x, y, Btn are displayed on screen.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import ADC0834
import time

BtnPin = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC0834.setup()

def destroy():
    # Release resource
    GPIO.cleanup()
```

(continues on next page)

(continued from previous page)

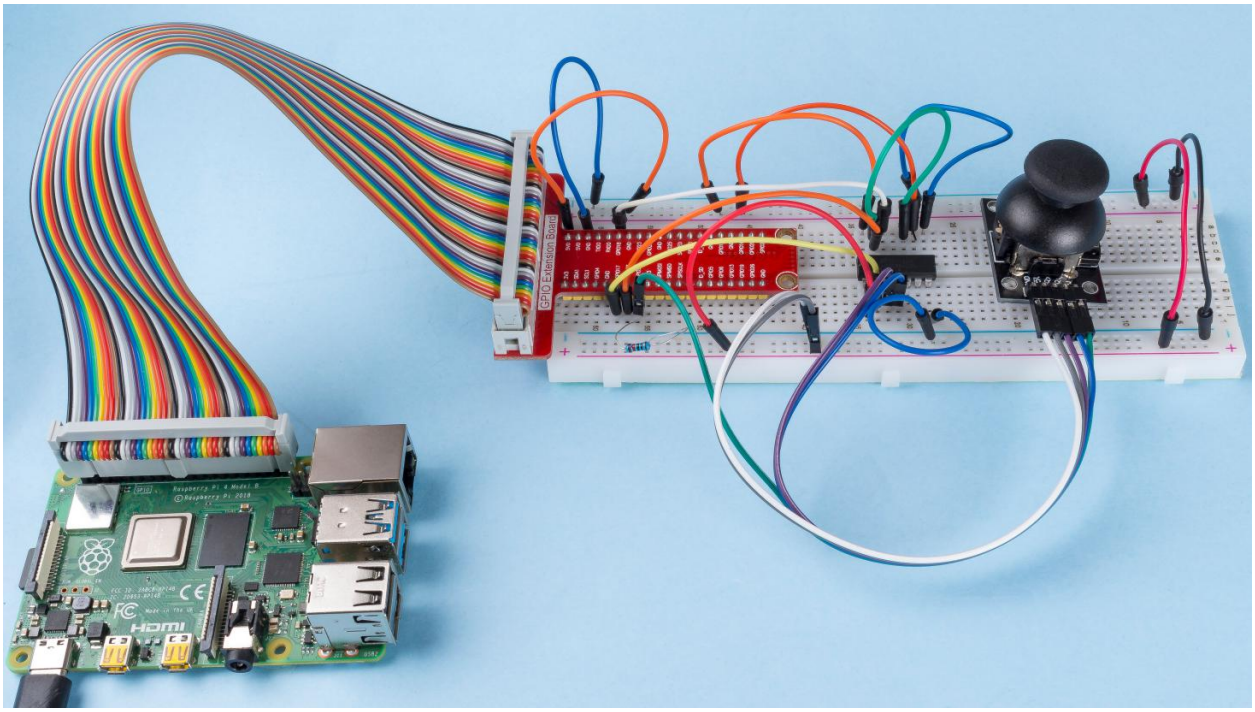
```
def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d  Y: %d  Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)

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

Code Explanation

```
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 and VRY of Joystick are connected to CH0, CH1 of ADC0834 respectively. So the function getResult() is called to read the values of CH0 and CH1. Then the read values should be stored in the variables x_val and y_val. In addition, read the value of SW of joystick and store it into the variable Btn_val. Finally, the values of x_val, y_val and Btn_val shall be printed with print() function.

Phenomenon Picture



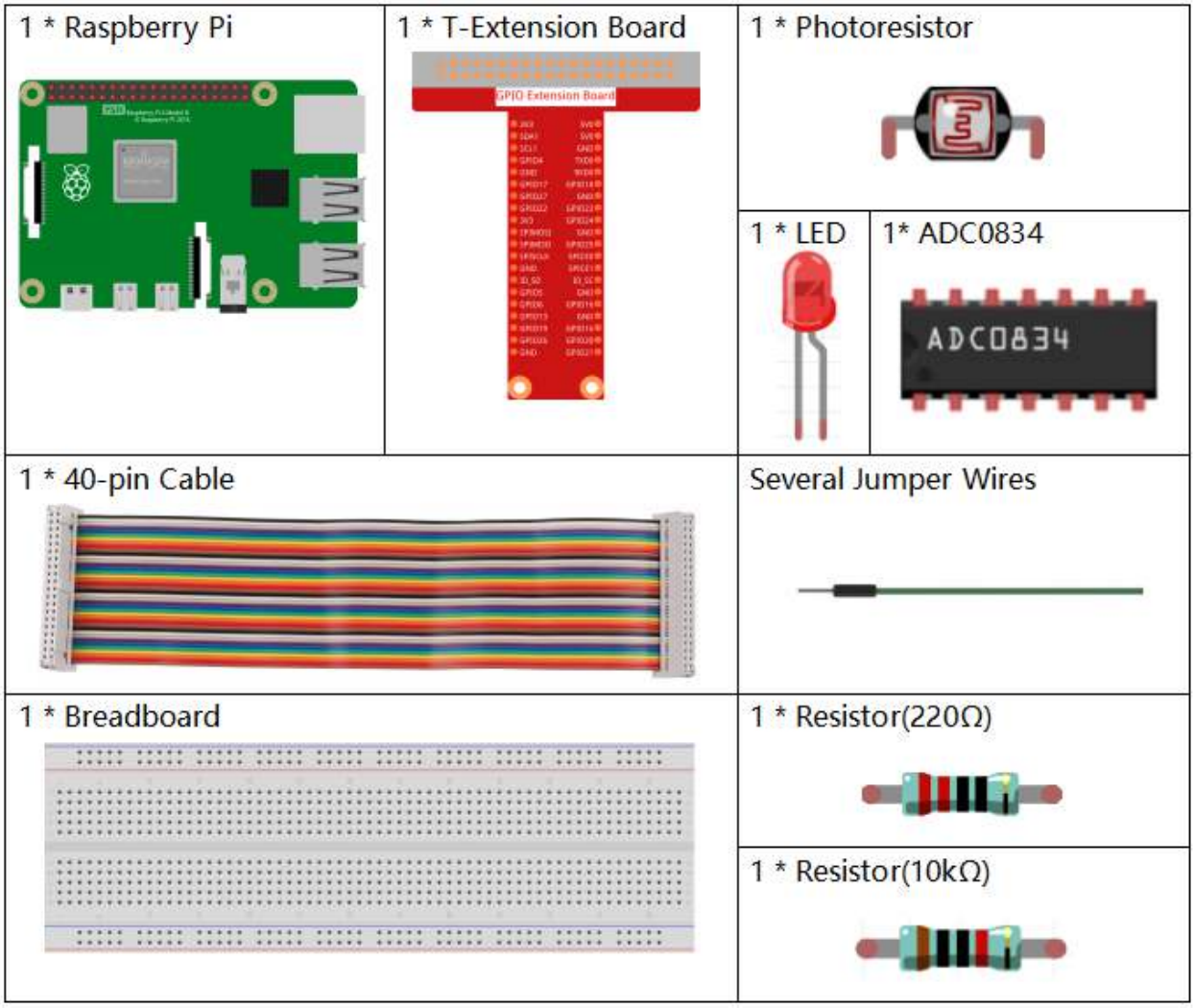
6.3.2 2.2 Sensors

2.2.1 Photoresistor

Introduction

Photoresistor is a commonly used component of ambient light intensity in life. It helps the controller to recognize day and night and realize light control functions such as night lamp. This project is very similar to potentiometer, and you might think it changing the voltage to sensing light.

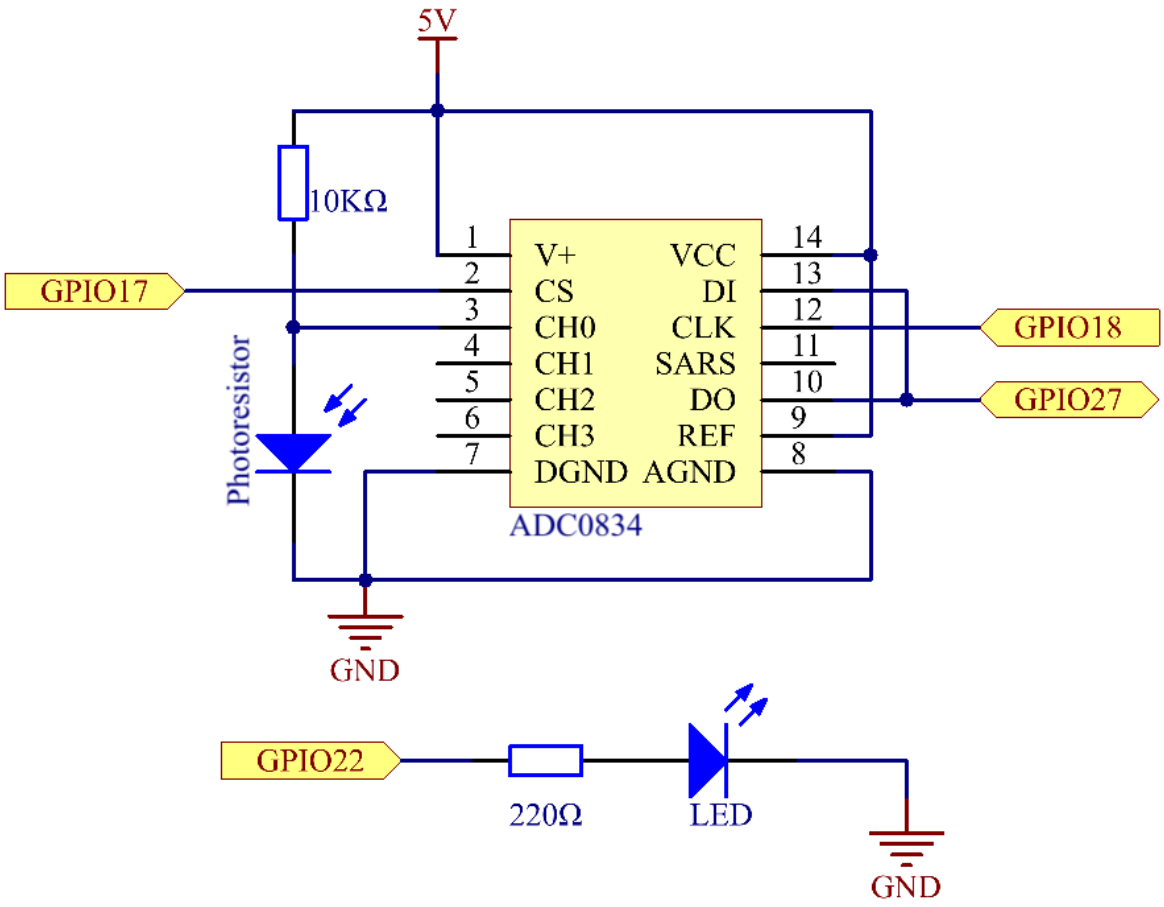
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *ADC0834*
- *Photoresistor*

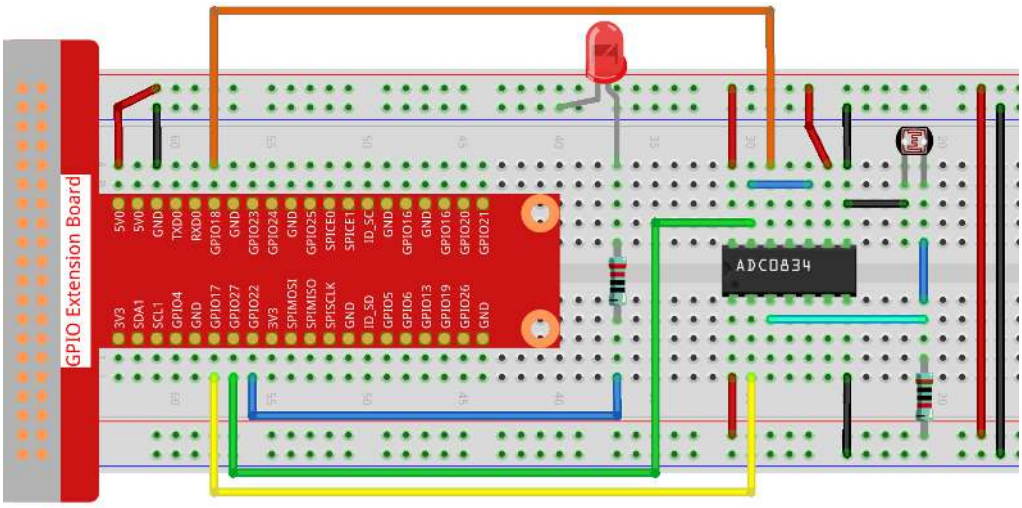
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 2.2.1_Photoresistor.py
```

When the code is running, the brightness of the LED will change according to the light intensity sensed by the photoresistor.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import ADC0834
import time
LedPin = 22
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and 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
```

(continues on next page)

(continued from previous page)

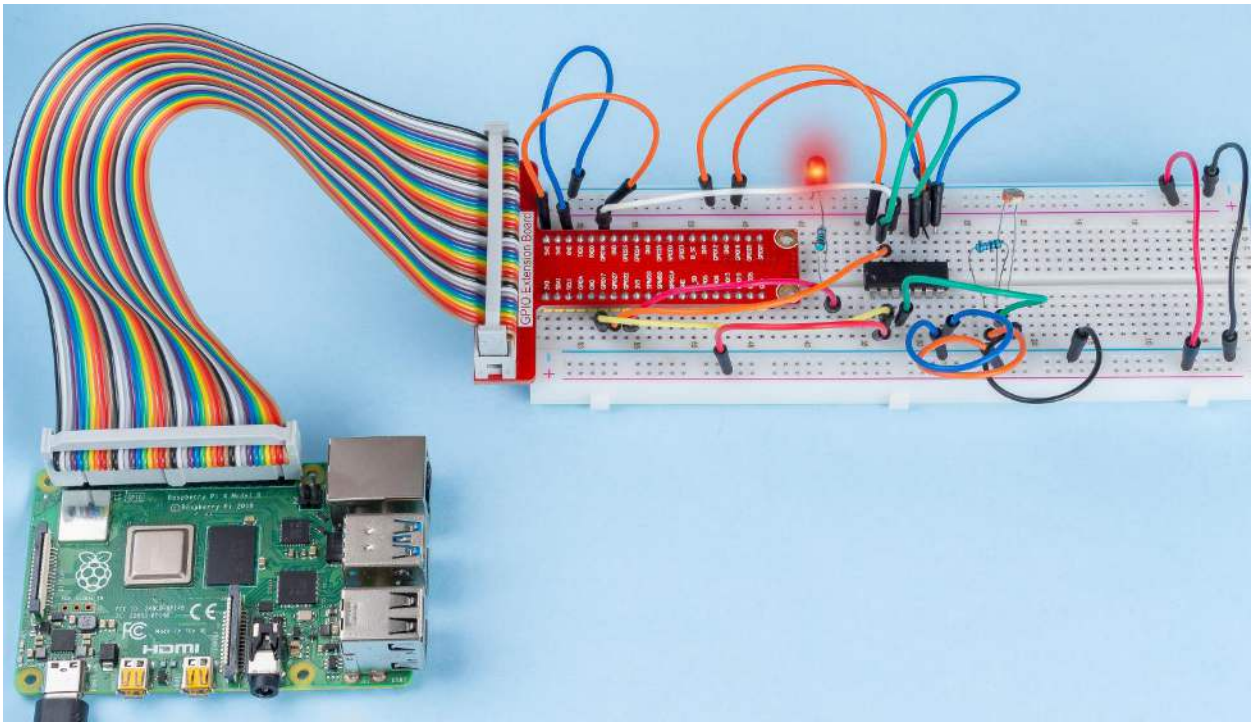
```
    led_val.stop()
    # Release resource
    GPIO.cleanup()
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will␣
↪be executed.
        destroy()
```

Code Explanation

```
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
```

Read the analog value of CH0 of ADC0834. By default, the function `getResult()` is used to read the value of CH0, so if you want to read other channels, please input 1, 2, or 3 into () of the function `getResult()`. Next, what you need is to print the value via the print function. Because the changing element is the duty cycle of `LedPin`, the computational formula, `analogVal*100/255` is needed to convert analogVal into percentage. Finally, `ChangeDutyCycle()` is called to write the percentage into `LedPin`.

Phenomenon Picture

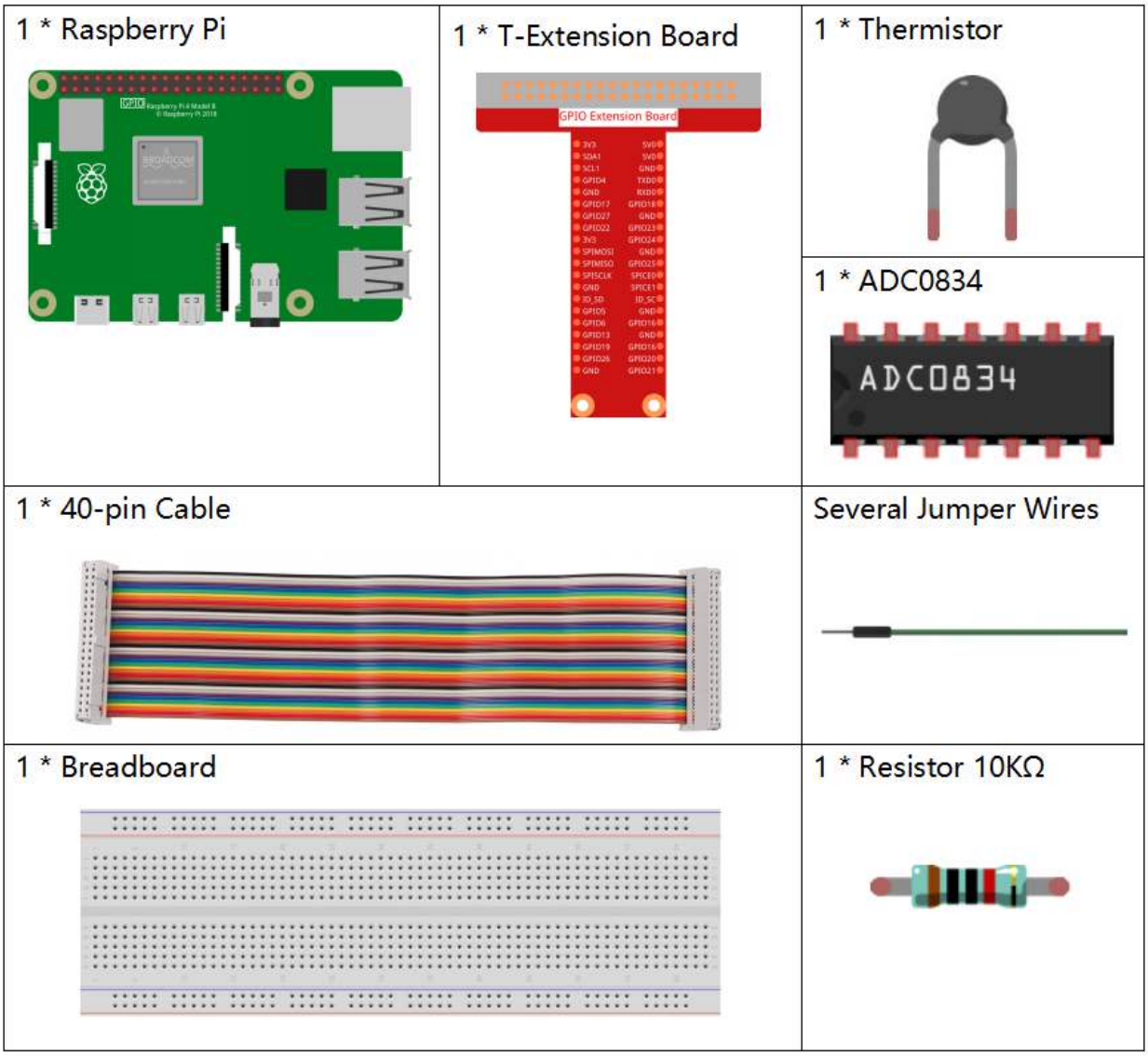


2.2.2 Thermistor

Introduction

Just like photoresistor can sense light, thermistor is a temperature sensitive electronic device that can be used for realizing functions of temperature control, such as making a heat alarm.

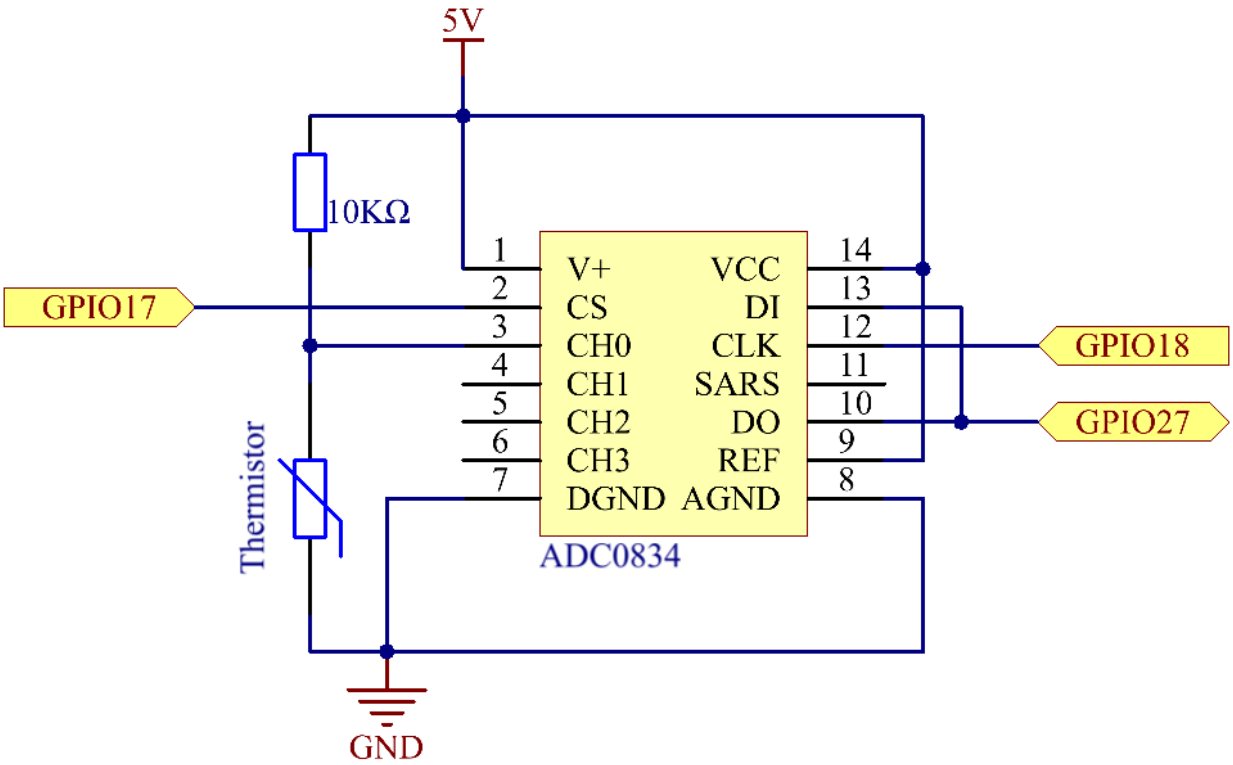
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Thermistor*
- *ADC0834*

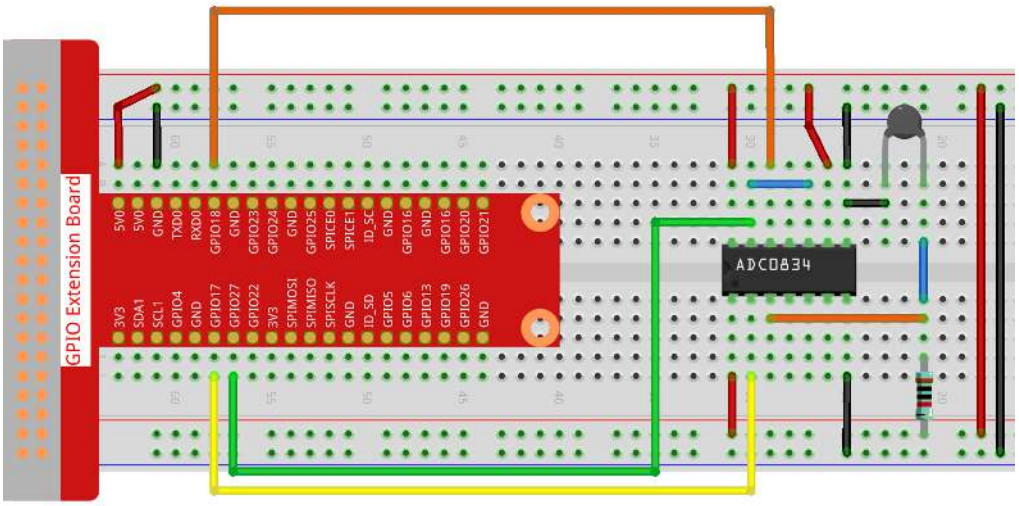
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file

```
sudo python3 2.2.2_Thermistor.py
```

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

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

def init():
    ADC0834.setup()

def loop():
    while True:
        analogVal = ADC0834.getResult()
        Vr = 5 * float(analogVal) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
        Cel = temp - 273.15
```

(continues on next page)

(continued from previous page)

```
        Fah = Cel * 1.8 + 32
        print ('Celsius: %.2f °C  Fahrenheit: %.2f ' % (Cel, Fah))
        time.sleep(0.2)

if __name__ == '__main__':
    init()
    try:
        loop()
    except KeyboardInterrupt:
        ADC0834.destroy()
```

Code Explanation

```
import math
```

There is a numerics library which declares a set of functions to compute common mathematical operations and transformations.

```
analogVal = ADC0834.getResult()
```

This function is used to read the value of the thermistor.

```
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))
```

These calculations convert the thermistor values into centigrade degree and Fahrenheit degree.

```
Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)
```

These two lines of codes are calculating the voltage distribution with the read value analog so as to get Rt (resistance of thermistor).

```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

This code refers to plugging Rt into the formula **TK=1/(ln(RT/RN)/B+1/TN)** to get Kelvin temperature.

```
temp = temp - 273.15
```

Convert Kelvin temperature into centigrade degree.

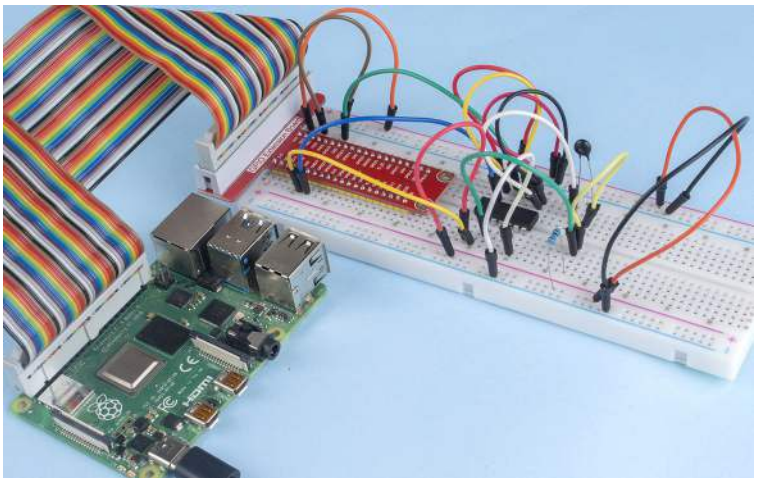
```
Fah = Cel * 1.8 + 32
```

Convert the centigrade degree into Fahrenheit degree.

```
print ('Celsius: %.2f °C  Fahrenheit: %.2f ' % (Cel, Fah))
```

Print centigrade degree, Fahrenheit degree and their units on the display.

Phenomenon Picture



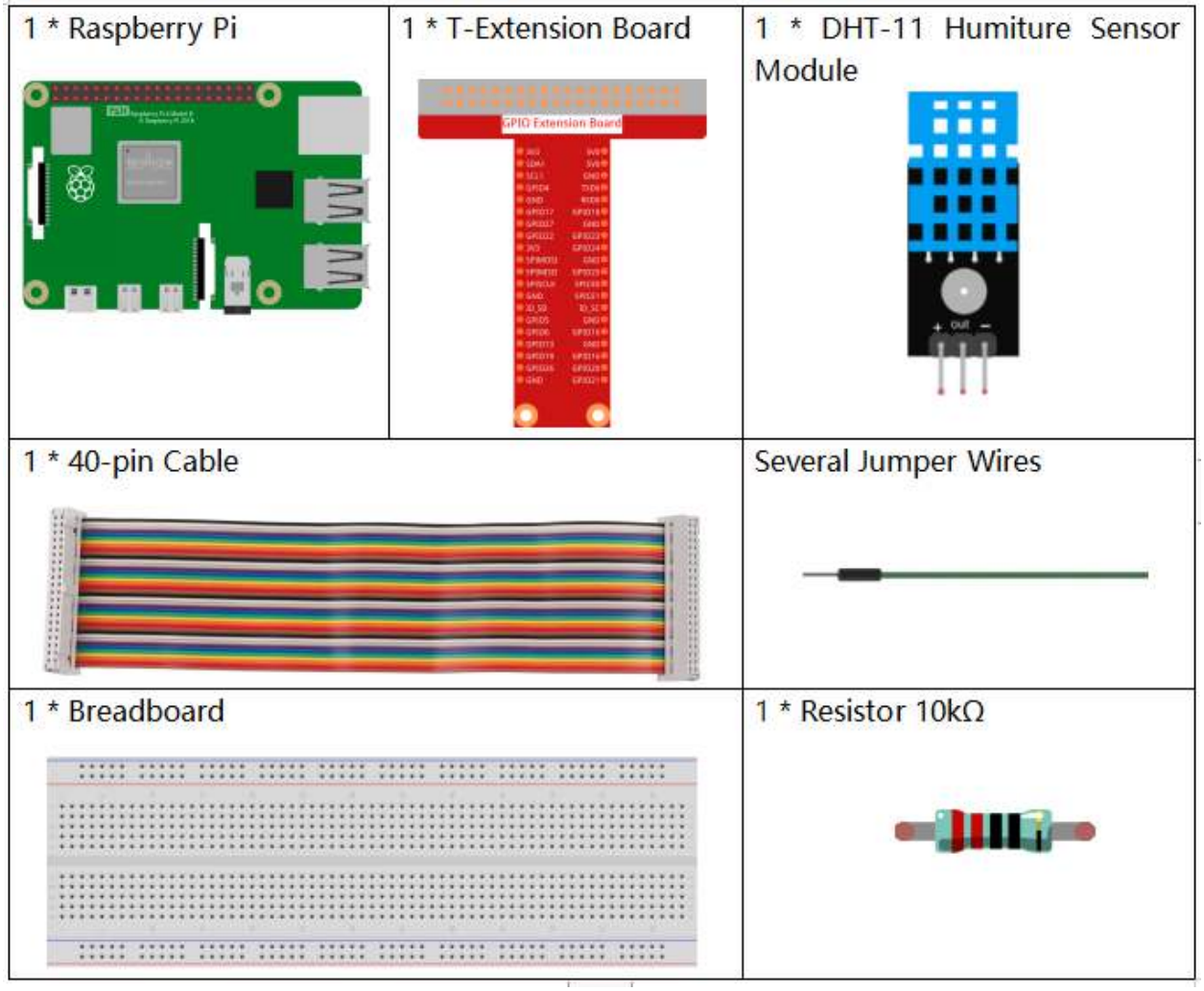
2.2.3 DHT-11

Introduction

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the technology of the temperature and humidity sensing are applied to ensure that the product has high reliability and excellent stability.

The sensors include a wet element resistive sensor and a NTC temperature sensor and they are connected to a high performance 8-bit microcontroller.

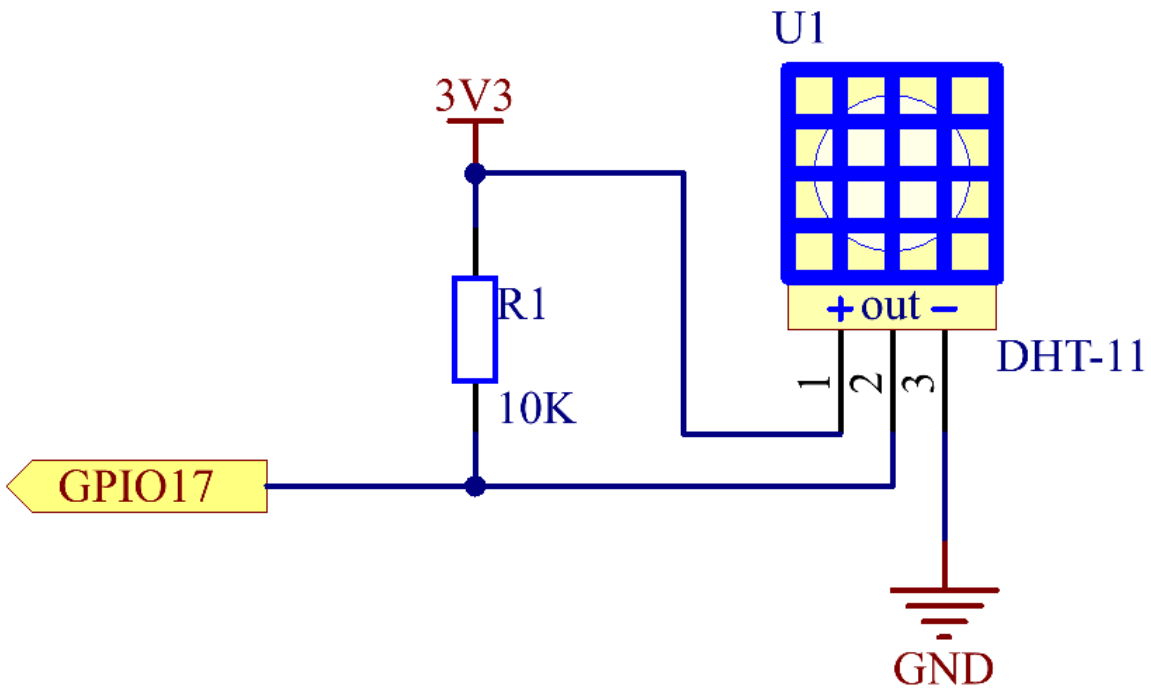
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Humiture Sensor Module*

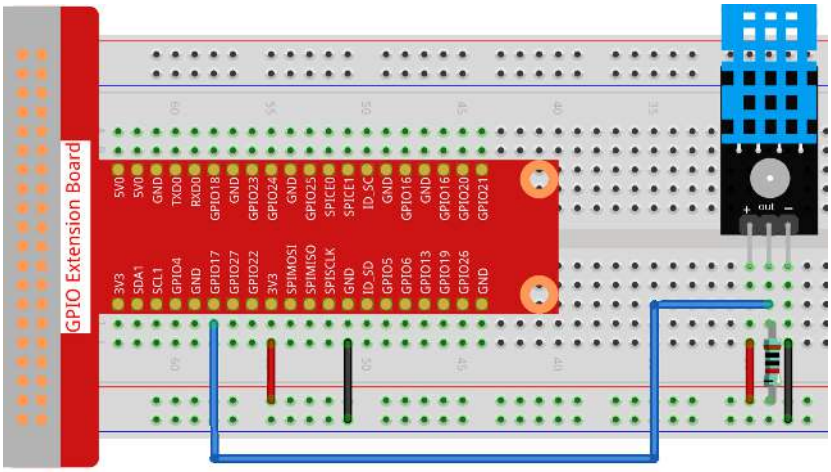
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 2.2.3_DHT.py
```

After the code runs, the program will print the temperature and humidity detected by DHT11 on the computer screen.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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
```

(continues on next page)

(continued from previous page)

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

    state = STATE_INIT_PULL_DOWN

    lengths = []
    current_length = 0

    for current in data:
        current_length += 1

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

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

(continues on next page)

(continued from previous page)

```
    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 Explanation

```
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 = []
    #...
```

This function is used to implement the functions of DHT11. It stores the detected data in the the_bytes[] array. DHT11

transmits data of 40 bits at a time. The first 16 bits are related to humidity, the middle 16 bits are related to temperature, and the last eight bits are used for verification. The data format is:

8bit humidity integer data +8bit humidity decimal data +8bit temperature integer data + 8bit temperature decimal data + 8bit check bit.

When the validity is detected via the check bit, the function returns two results: 1. error; 2. humidity and temperature.

```
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]
```

For example, if the received date is 00101011(8-bit value of humidity integer) 00000000 (8-bit value of humidity decimal) 00111100 (8-bit value of temperature integer) 00000000 (8-bit value of temperature decimal) 01100111 (check bit)

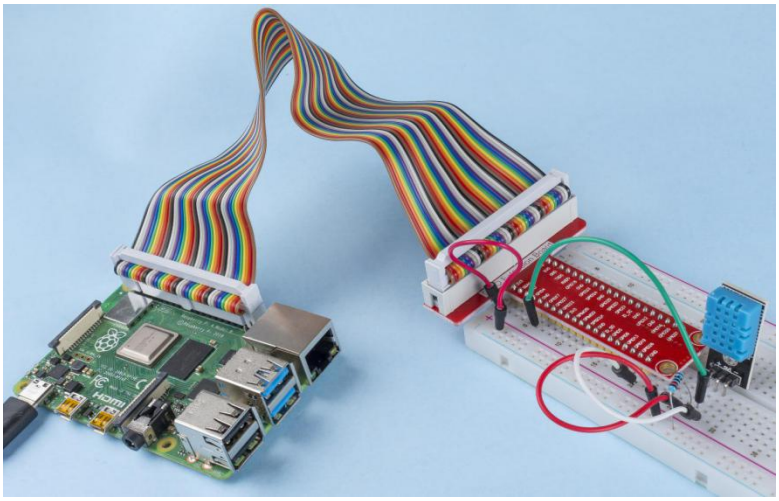
Calculation:

00101011+00000000+00111100+00000000=01100111.

If the final result is equal to the check bit data, the data transmission is abnormal: return False.

If the final result is equal to the check bit data, the received data is correct, then there will return the_bytes[0] and the_bytes[2] and output “Humidity =43%Temperature =60C”.

Phenomenon Picture

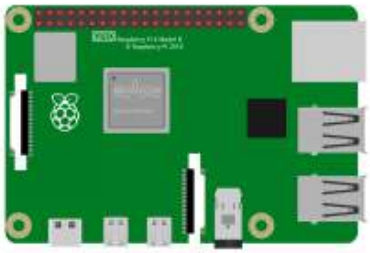
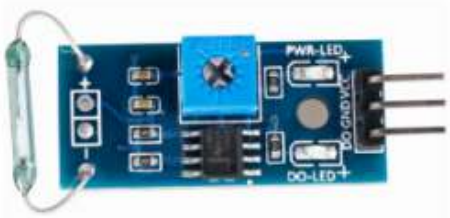
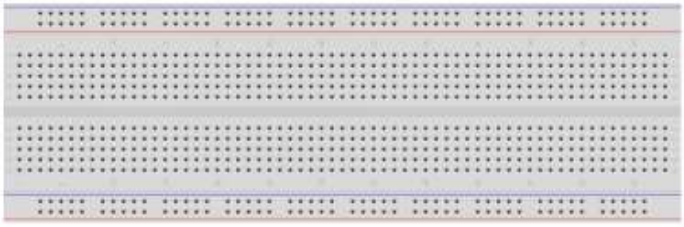
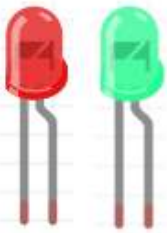
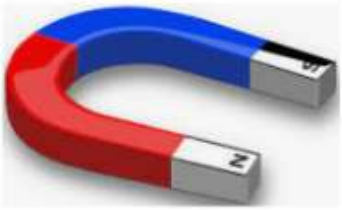


2.2.4 Reed Switch Module

Introduction

In this project, we will learn about the reed switch, which is an electrical switch that operates by means of an applied magnetic field.

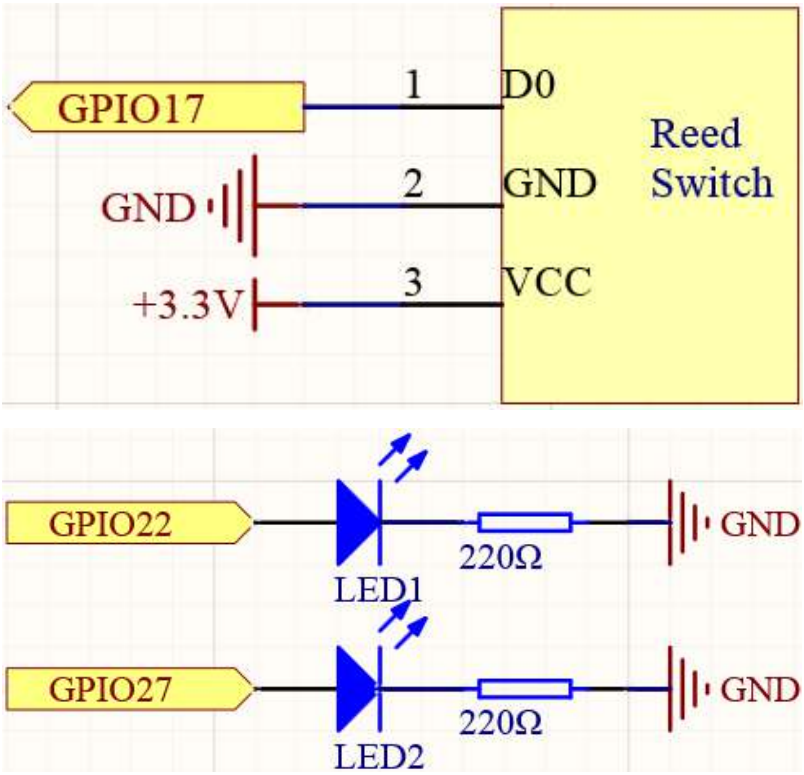
Components

1 * Raspberry Pi 	1 * T-Extension Board	1 * Reed Switch Module 
1 * 40-pin Cable		2 * Resistor(220Ω)
1 * Breadboard 		2 * LED 
1 * Magnet 		Several Jumper Wires

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Reed Switch Module*

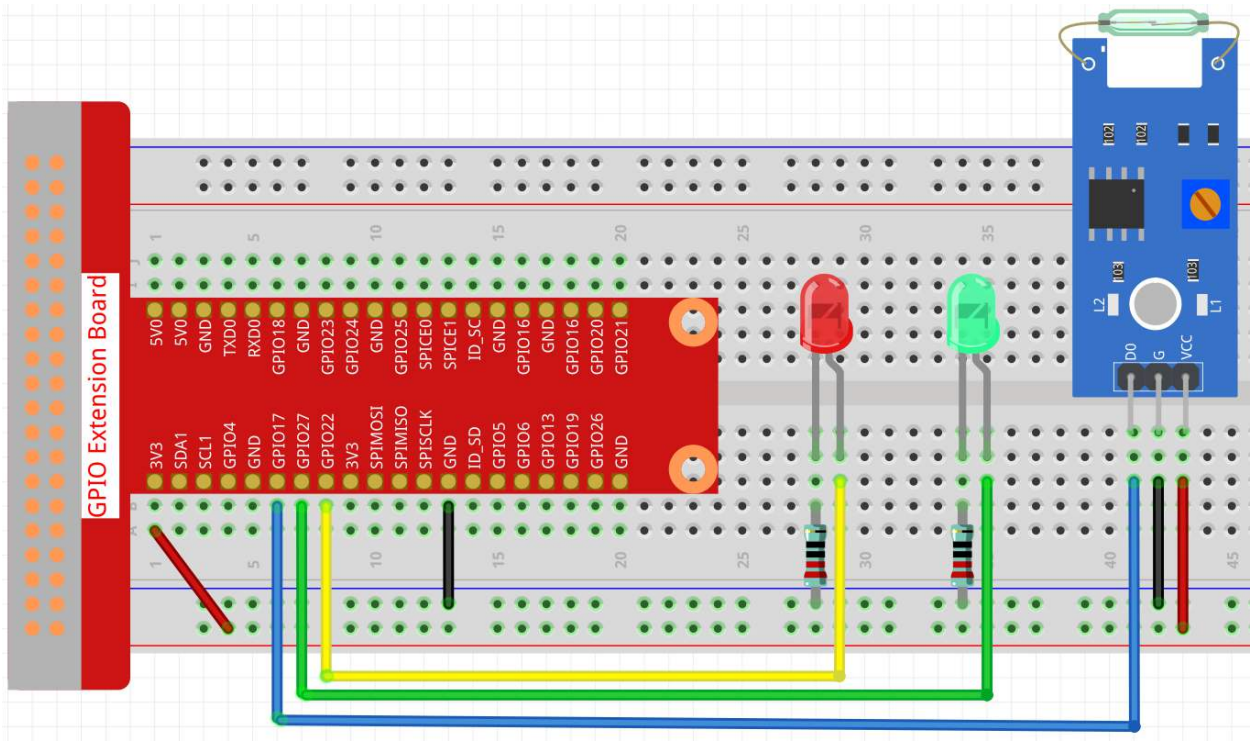
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 2.2.4_ReedSwitch.py
```

The green LED will light up when the code is run. If a magnet is placed close to the reed switch module, the red LED lights up; take away the magnet and the green LED lights up again.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time

ReedPin = 17
Gpin    = 27
Rpin    = 22

def setup():
    GPIO.setmode(GPIO.BCM)        #
    GPIO.setup(Gpin, GPIO.OUT)      # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)      # Set Red Led Pin mode to output
    GPIO.setup(ReedPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # Set ReedPin's mode is␣
↪input, and pull up to high level(3.3V)
```

(continues on next page)

(continued from previous page)

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

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)

def detect(self):
    Led(GPIO.input(ReedPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.HIGH)       # Green led on
    GPIO.output(Rpin, GPIO.LOW)       # Red led off
    GPIO.cleanup()                     # Release resource

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

Code Explanation

```
ReedPin = 17
Gpin    = 27
Rpin    = 22

def setup():
    GPIO.setmode(GPIO.BCM)       #
    GPIO.setup(Gpin, GPIO.OUT)     # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)     # Set Red Led Pin mode to output
    GPIO.setup(ReedPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set ReedPin's mode is␣
↪input, and pull up to high level(3.3V)
    GPIO.add_event_detect(ReedPin, GPIO.BOTH, callback=detect, bouncetime=200)
```

Set the GPIO modes to BCM Numbering. `ReedPin`, `Gpin` and `Rpin` connects to the GPIO17, GPIO27 and GPIO22.

`GPIO.add_event_detect()` is used to add an event that is triggered by a change in the value (level) of `ReedPin`, at which point the callback function `detect()` is called.

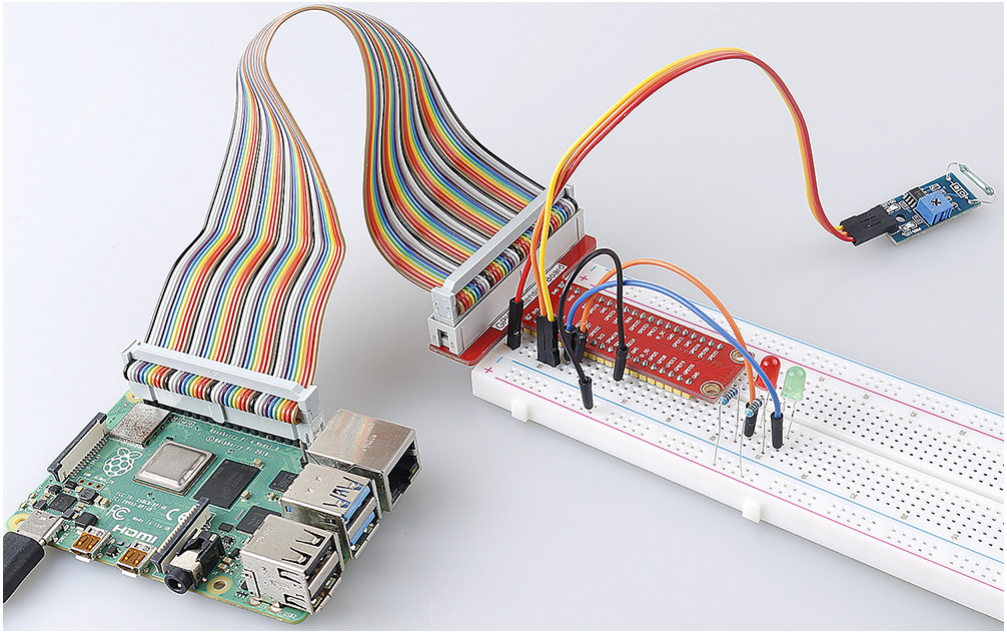
```
def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)
```

Define a function `Led()` to turn the two LEDs on or off. If `x=0`, the red LED lights up; otherwise, the green LED will be lit.

```
def detect(self):
Led(GPIO.input(ReedPin))
```

In this callback function, the value of the reed switch is used to control the 2 LEDs.

Phenomenon Picture

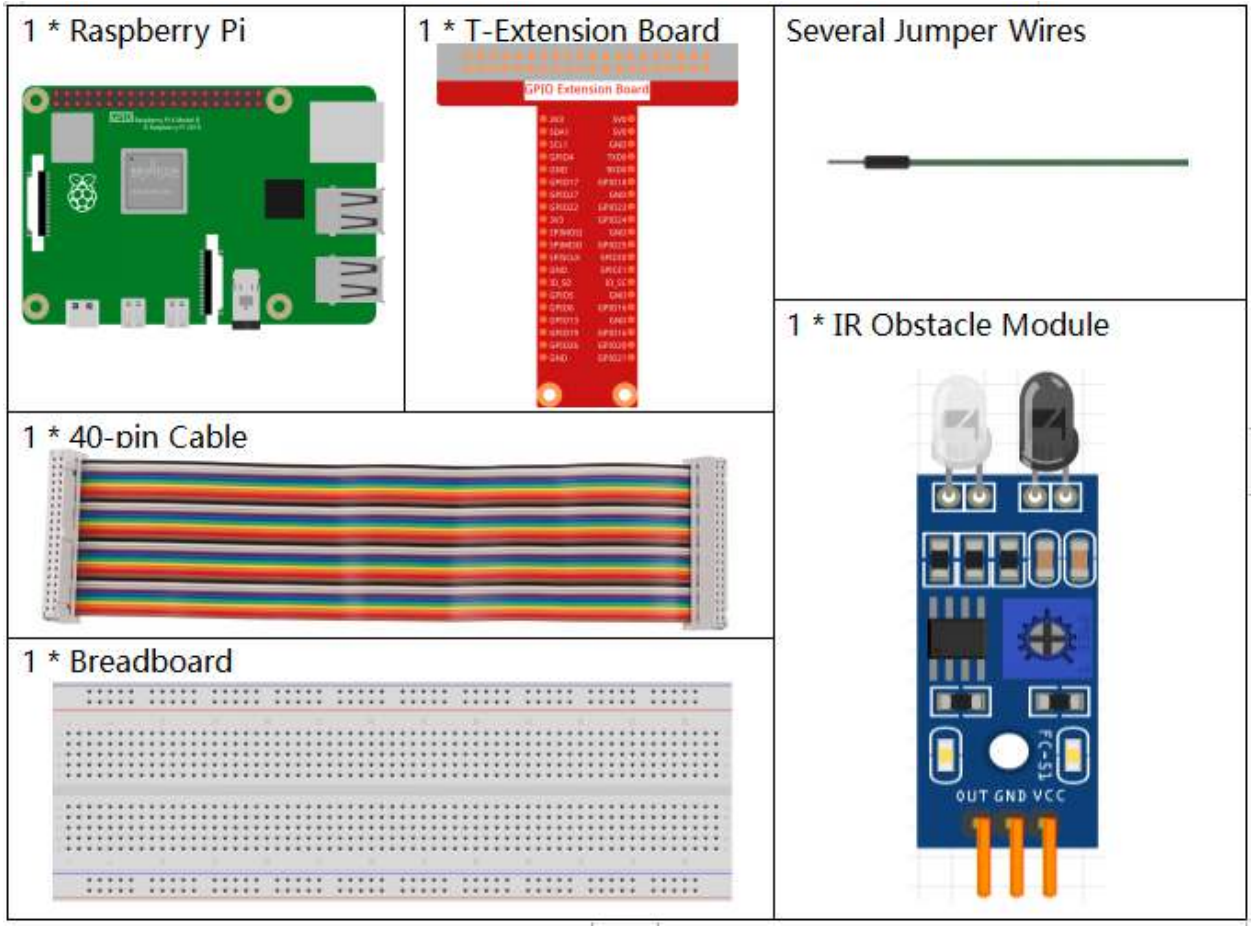


2.2.5 IR Obstacle Avoidance Sensor

Introduction

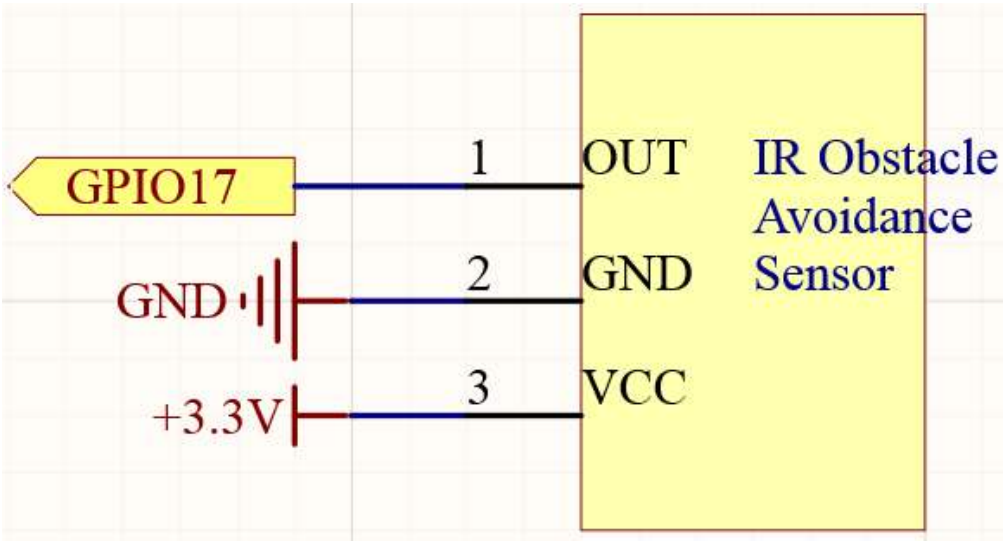
In this project, we will learn IR obstacle avoidance module, which is a sensor module that can be used to detect obstacles at short distances, with small interference, easy to assemble, easy to use, etc. It can be widely used in robot obstacle avoidance, obstacle avoidance trolley, assembly line counting, etc.

Components Required



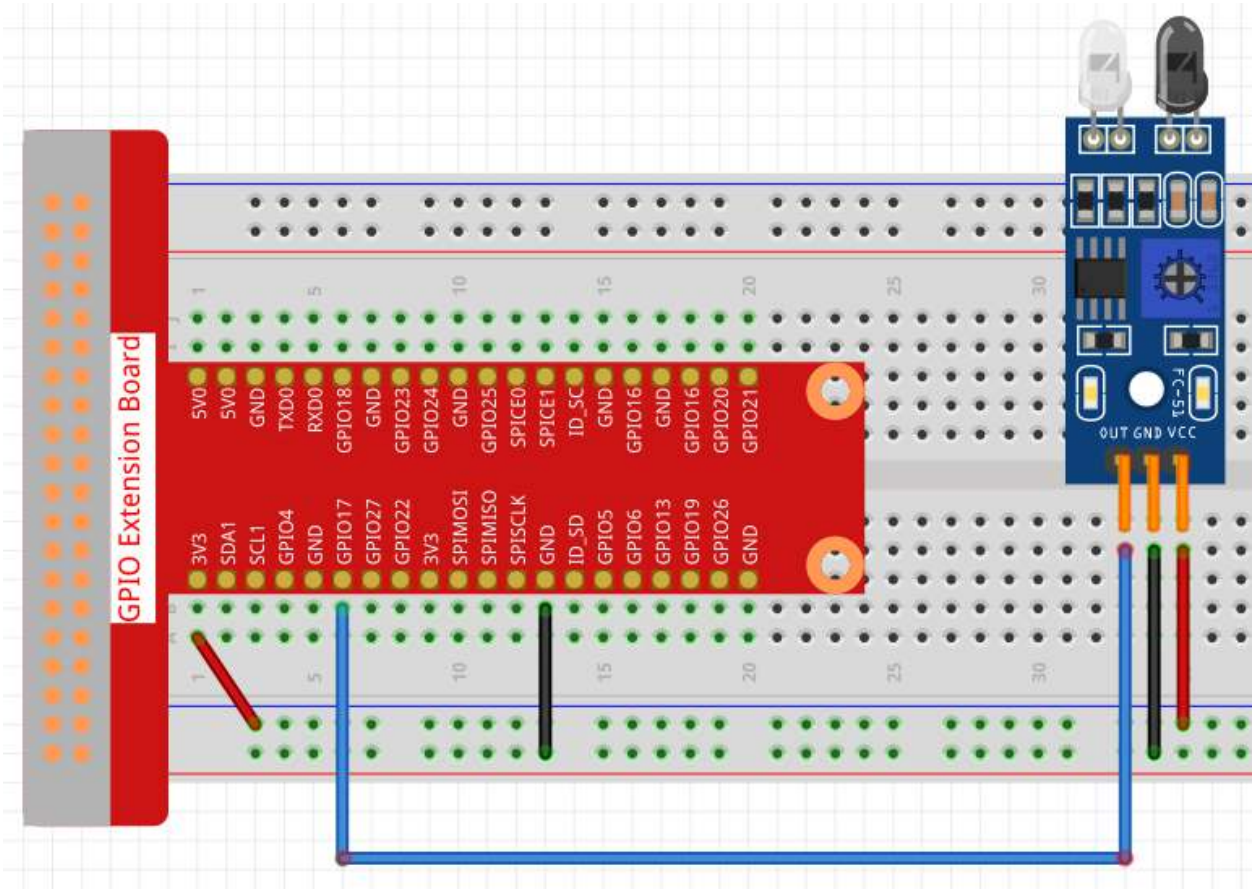
- *GPIO Extension Board*
- *Breadboard*
- *Obstacle Avoidance Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 2.2.5_IrObstacle.py
```

After the code runs, when you put your hand in front of the module's probe, the output indicator on the module lights up and the "Detected Barrier!" will be repeatedly printed on the screen until the your hand is removed.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time

ObstaclePin = 17

def setup():
    GPIO.setmode(GPIO.BCM)        # Numbers GPIOs by physical location
    GPIO.setup(ObstaclePin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def loop():
    while True:
        if (0 == GPIO.input(ObstaclePin)):
            print ("Detected Barrier!")
                        time.sleep(1)


def destroy():
    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 Explanation

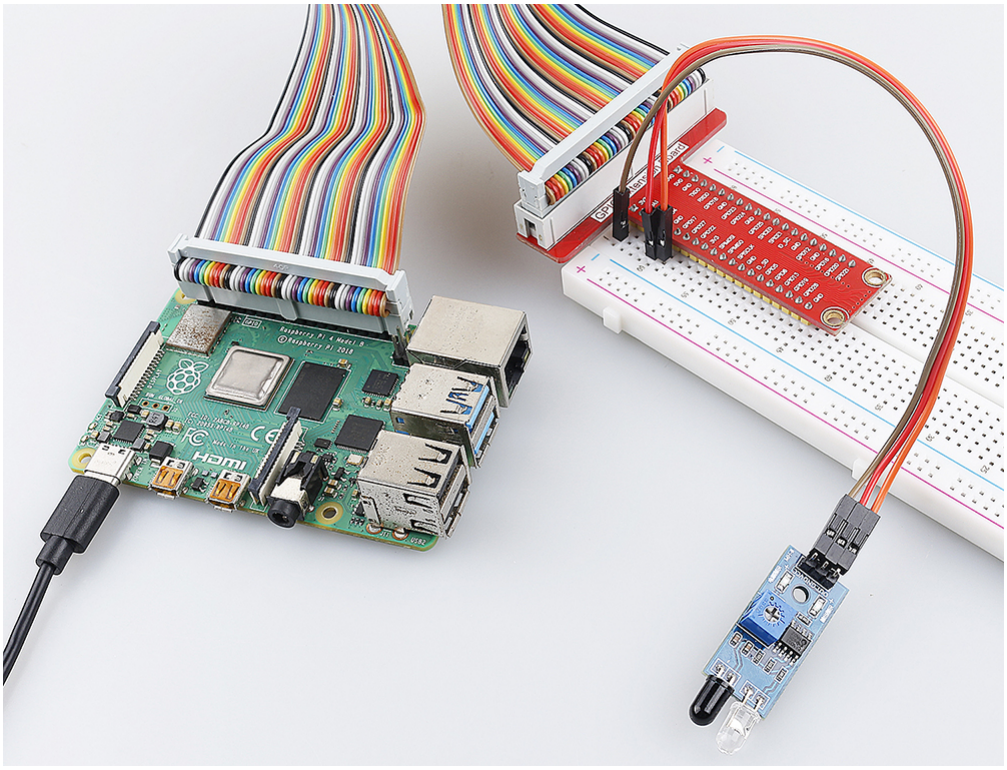
```
def setup():
GPIO.setmode(GPIO.BCM)        # Numbers GPIOs by physical location
GPIO.setup(ObstaclePin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Set the GPIO mode to BCM Numbering. Set `ObstaclePin` to input mode and initial it to High level (3.3v).

```
def loop():
    while True:
        if (0 == GPIO.input(ObstaclePin)):
            print ("Detected Barrier!")
```

When `ObstaclePin` is low level, print "Detected Barrier!". It means that an obstacle is detected.

Phenomenon Picture

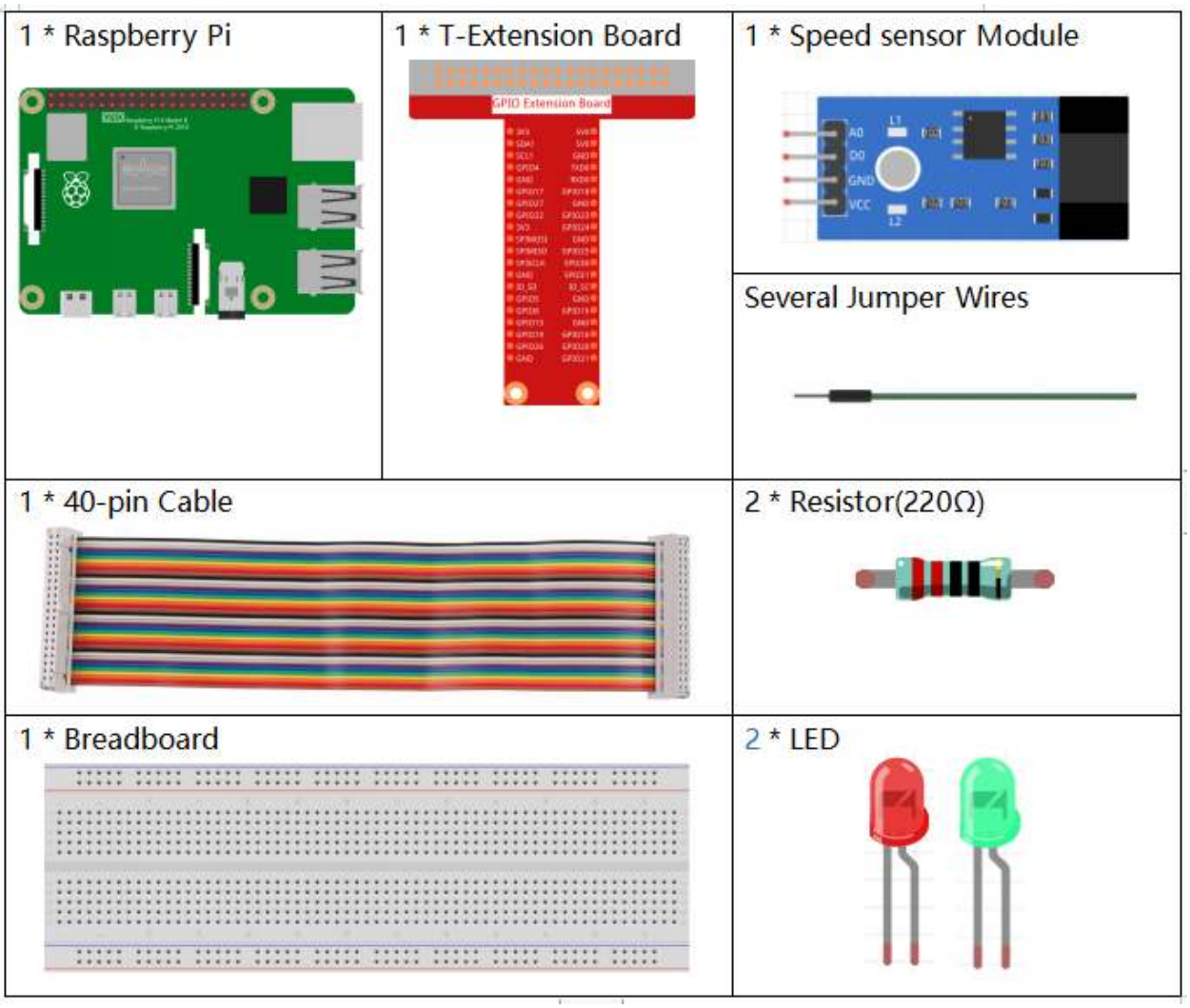


2.2.6 Speed Sensor Module

Introduction

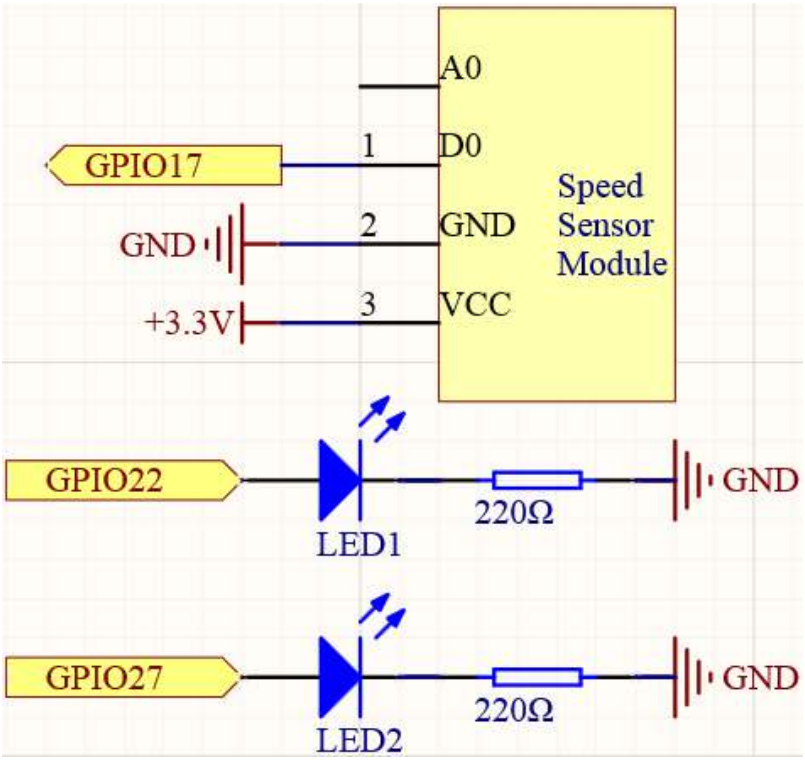
In this project, we will learn the use of the speed sensor module. A Speed sensor module is a type of tachometer that is used to measure the speed of a rotating object like a motor.

Components



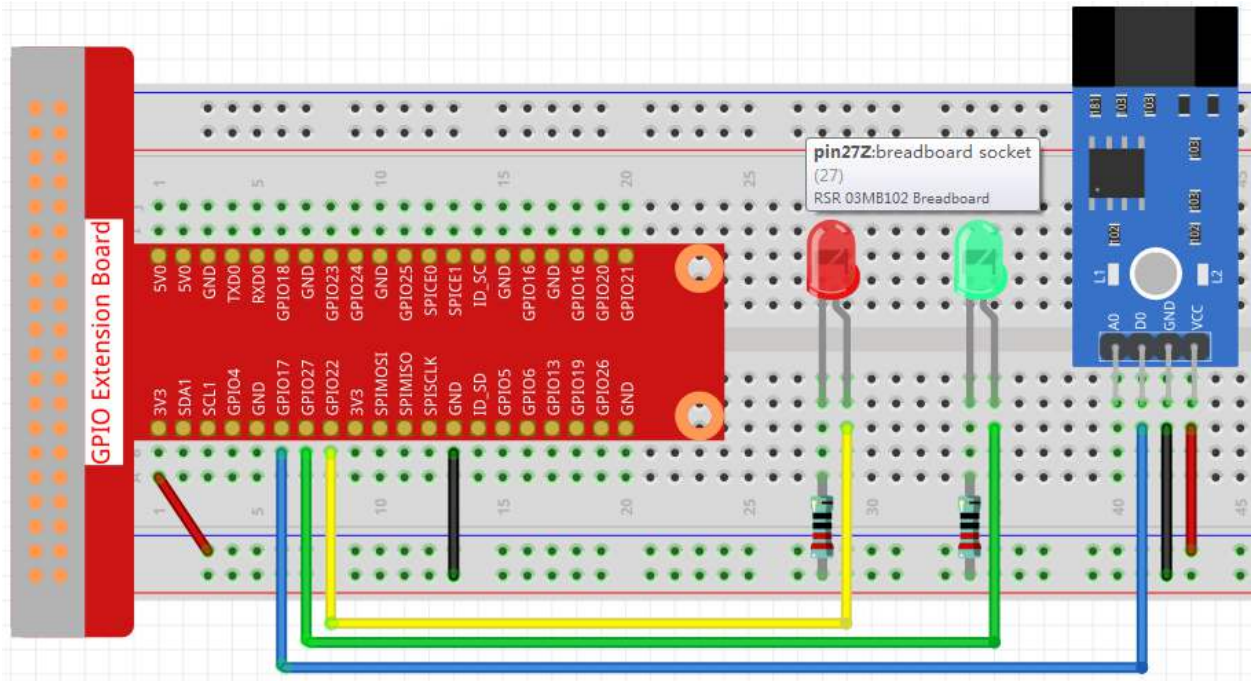
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Speed Sensor Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 2.2.6_speed_sensor_module.py
```

After the code runs, the green LED will light up. If you place an obstacle in the gap of the speed sensor module, the "light blocked" will be printed on the screen and the red LED will be lit. Remove the obstacle and the green LED will light up again.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO

speedPin  = 17
Gpin   = 27
Rpin   = 22

def setup():
    GPIO.setmode(GPIO.BCM)       #
    GPIO.setup(Gpin, GPIO.OUT)     # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)     # Set Red Led Pin mode to output
    GPIO.setup(speedPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set speedPin's mode␣
↪is input, and pull up to high level(3.3V)
    GPIO.add_event_detect(speedPin, GPIO.BOTH, callback=detect, bouncetime=200)

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)
    if x == 1:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
        print ('Light was blocked')

def detect(chn):
    Led(GPIO.input(speedPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.LOW)       # Green led off
    GPIO.output(Rpin, GPIO.LOW)       # Red led off
    GPIO.cleanup()                     # Release resource

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

(continues on next page)

(continued from previous page)

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

Code Explanation

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

Add an event here, triggered by a change in the level of `speedPin` and call `detect()` to control the 2 LEDs on and off.

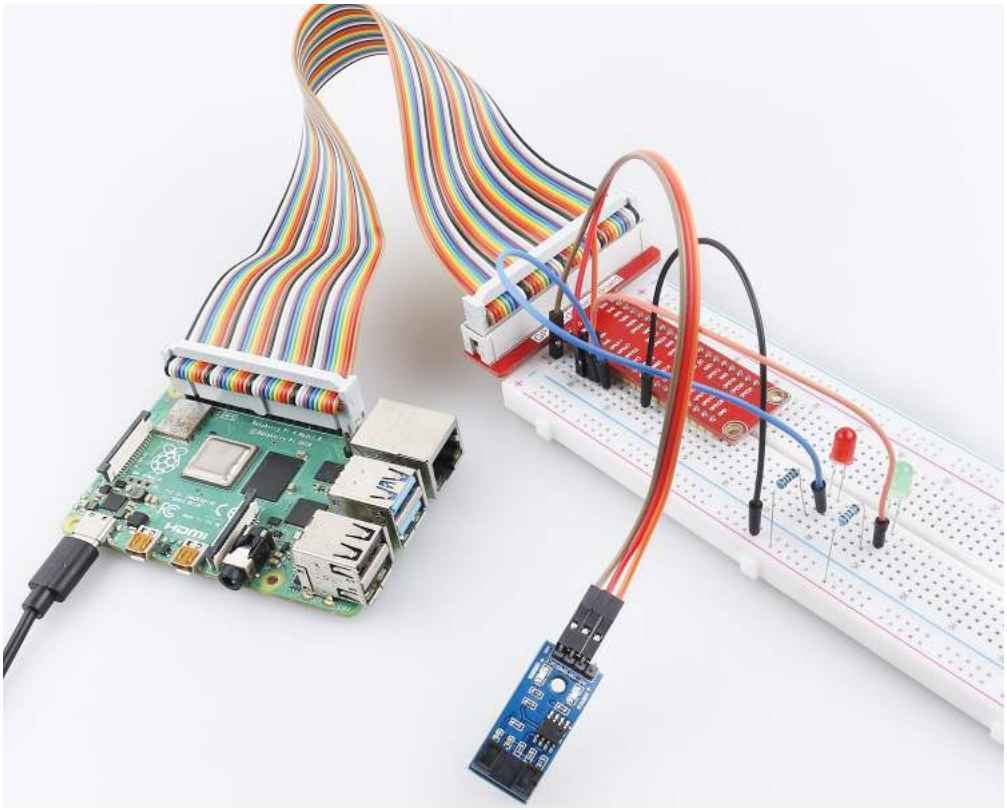
```
def Led(x):
    if x == 0:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)
    if x == 1:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
        print ('Light was blocked')
```

Define a function `Led()` that turns the red LED on and prints `Light was blocked` when the parameter is 1; turn the green LED on when the parameter is 0.

```
def detect(chn):
    Led(GPIO.input(speedPin))
```

Define a callback function where the value of `speedPin` will control the turning on or off of the 2 LEDs.

Phenomenon Picture

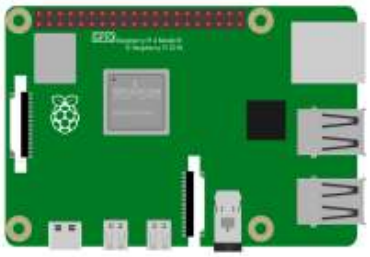
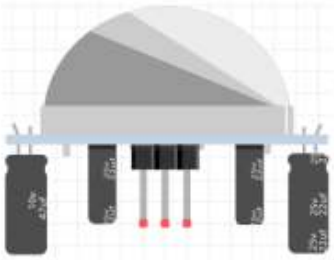
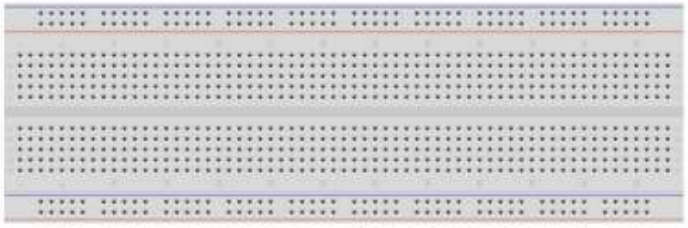


2.2.7 PIR

Introduction

In this project, we will make a device by using the human body infrared pyroelectric sensors. When someone gets closer to the LED, the LED will turn on automatically. If not, the light will turn off. This infrared motion sensor is a kind of sensor that can detect the infrared emitted by human and animals.

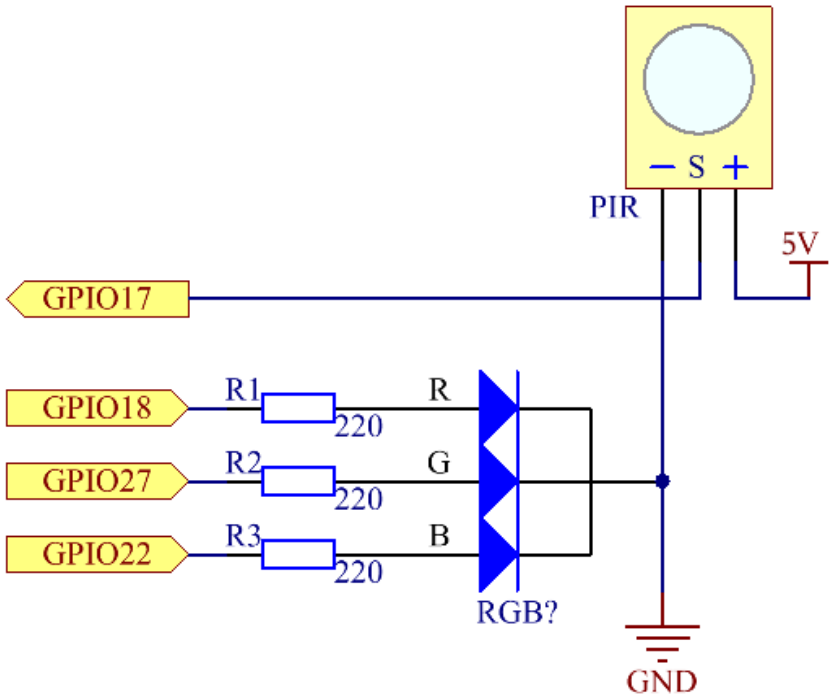
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>1 * PIR
  </td></tr>
<tr><td>1 * RGB LED
 </td><td>3 * Resistor 220Ω
 </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>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*
- *PIR Motion Sensor Module*

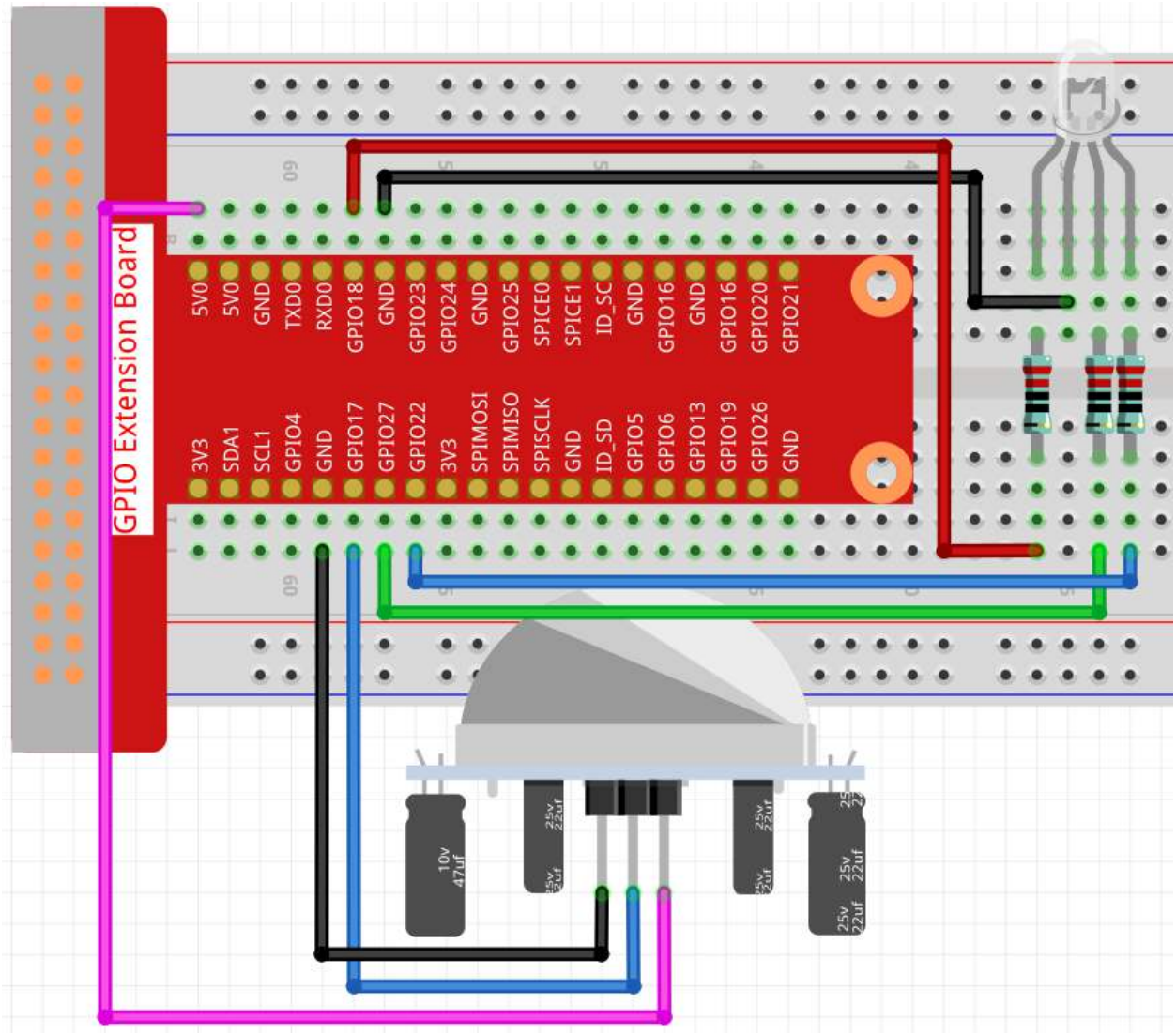
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

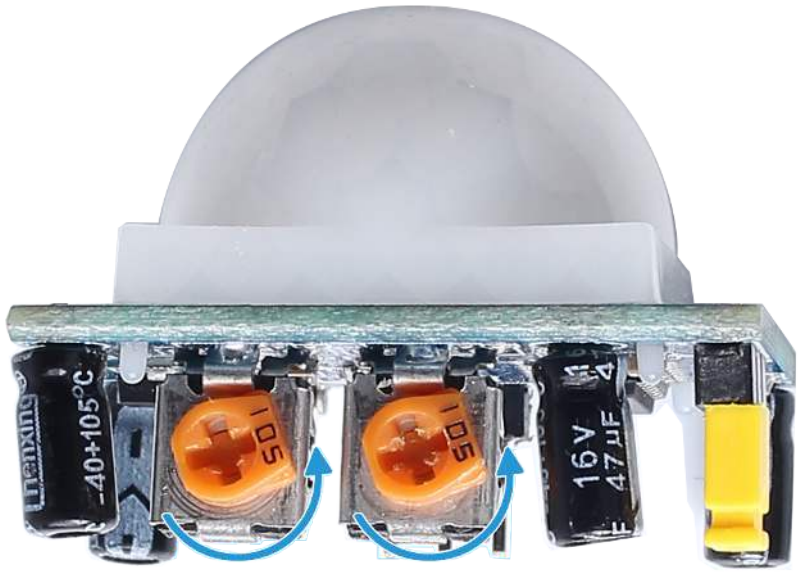
```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 2.2.7_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

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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):
```

(continues on next page)

(continued from previous page)

```
# configures the three LEDs' luminance with the inputted color value .
    # Devide colors from 'color' veriable
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0
    # Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

    #Assign the mapped duty cycle value to the corresponding PWM channel to change␣
↪the luminance.
    p_R.ChangeDutyCycle(R_val)
    p_G.ChangeDutyCycle(G_val)
    p_B.ChangeDutyCycle(B_val)
    #print ("color_msg: R_val = %s, G_val = %s, B_val = %s"%(R_val, G_val, B_val))

def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)

def destroy():
    p_R.stop()
    p_G.stop()
    p_B.stop()
    GPIO.cleanup()                     # Release resource

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

Code Explanation

```
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):
```

(continues on next page)

(continued from previous page)

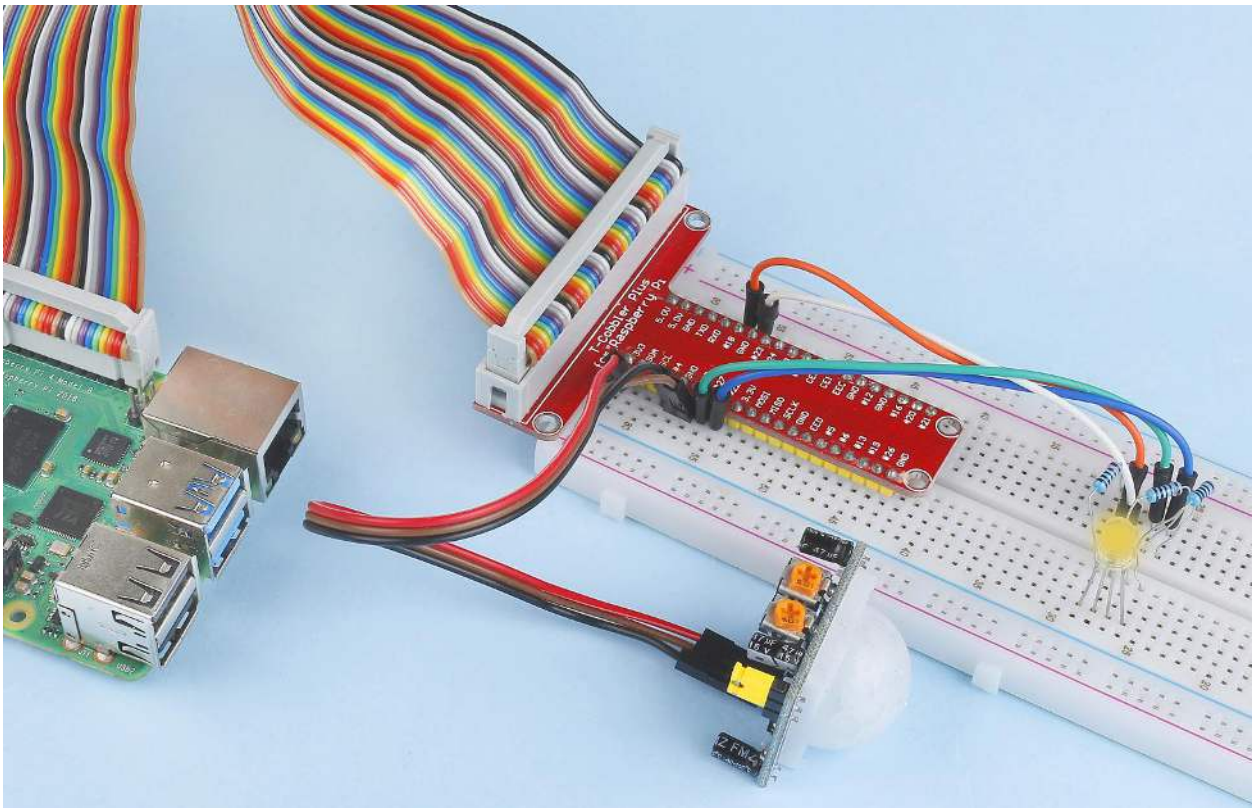
```
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def setColor(color):
...
```

These codes are used to set the color of the RGB LED, and please refer to *1.1.2 RGB LED* for more details.

```
def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)
```

When PIR detects the human infrared spectrum, RGB LED emits the yellow light; if not, emits the blue light.

Phenomenon Picture

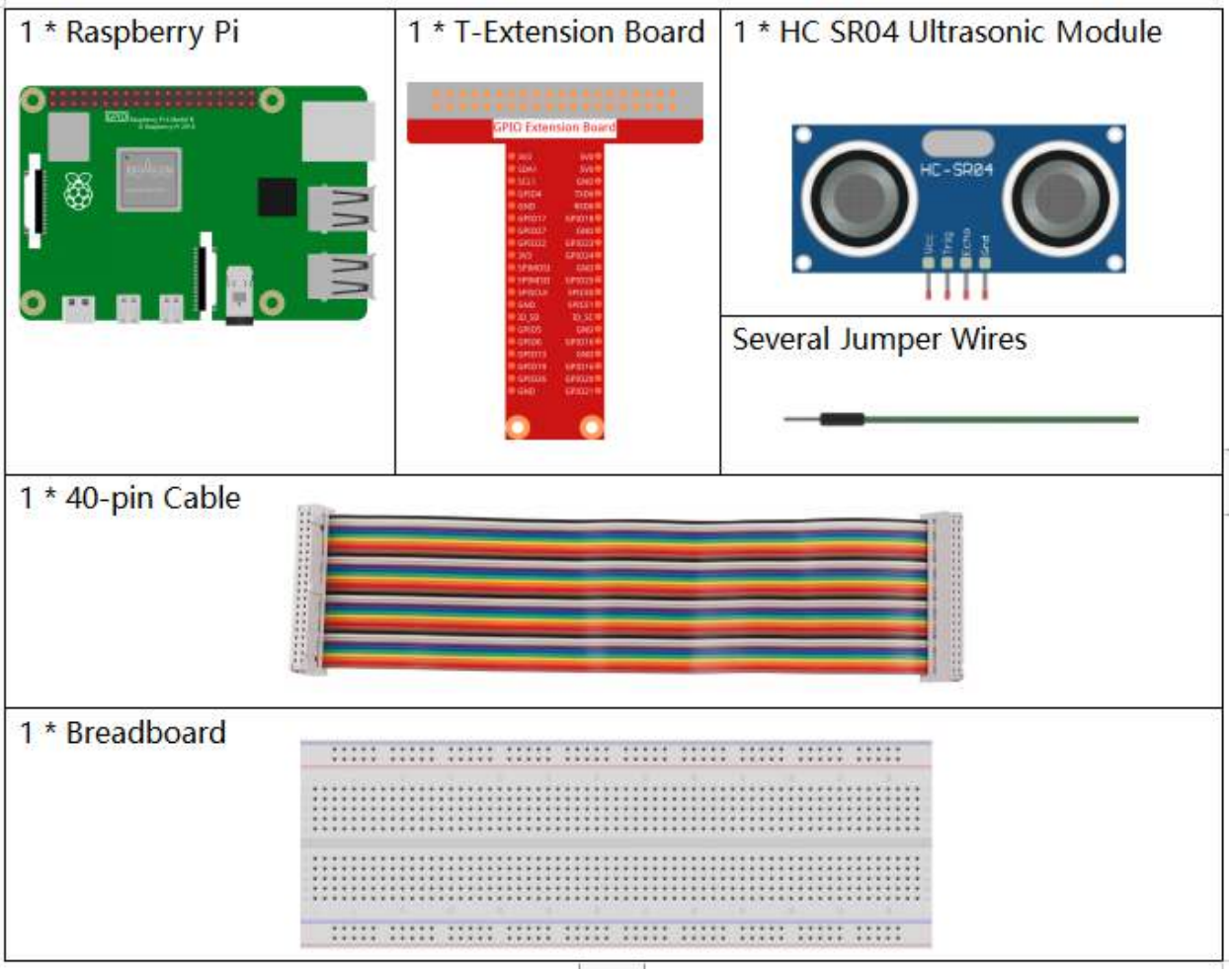


2.2.8 Ultrasonic Sensor Module

Introduction

The ultrasonic sensor uses ultrasonic to accurately detect objects and measure distances. It sends out ultrasonic waves and converts them into electronic signals.

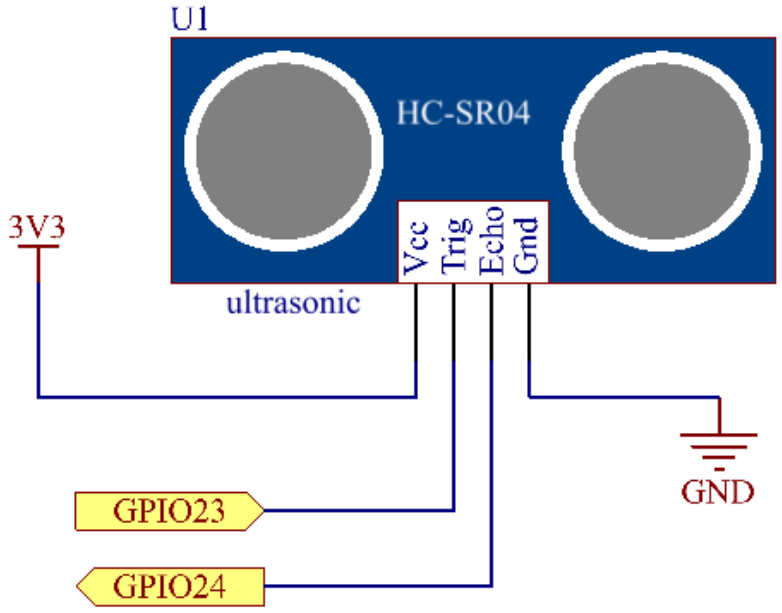
Components



- *GPIO Extension Board*
- *Breadboard*
- *Ultrasonic Module*

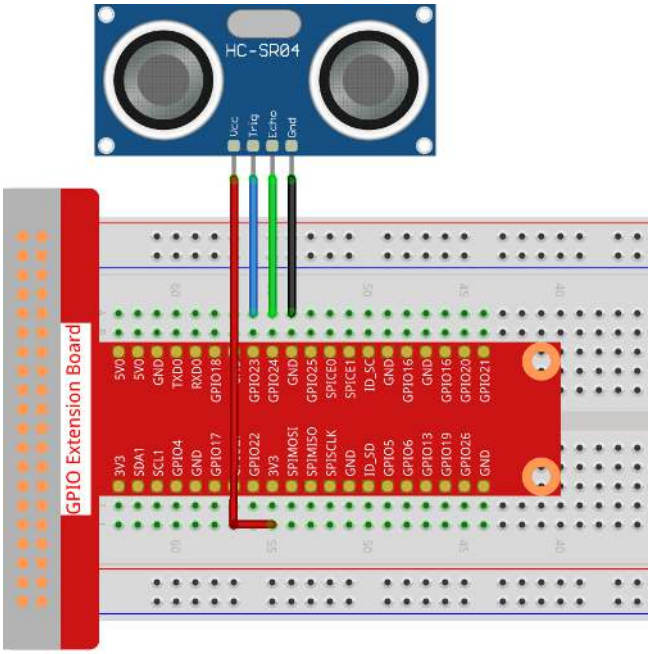
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 2.2.8_Ultrasonic.py
```

With the code run, the ultrasonic sensor module detects the distance between the obstacle ahead and the module itself, then the distance value will be printed on the screen.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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)
```

(continues on next page)

(continued from previous page)

```
    GPIO.output(TRIG, 0)

    while GPIO.input(ECHO) == 0:
        a = 0
    time1 = time.time()
    while GPIO.input(ECHO) == 1:
        a = 1
    time2 = time.time()

    during = time2 - time1
    return during * 340 / 2 * 100

def loop():
    while True:
        dis = distance()
        print ('Distance: %.2f' % dis )
        time.sleep(0.3)

def destroy():
    GPIO.cleanup()

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

Code Explanation

```
def distance():
```

This function is used to realize the function of ultrasonic sensor by calculating the return detection distance.

```
GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)
```

This is sending out a 10us ultrasonic pulse.

```
while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
```

This empty loop is used to ensure that when the trigger signal is sent, there is no interfering echo signal and then get the current time.

```
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()
```

This empty loop is used to ensure that the next step is not performed until the echo signal is received and then get the current time.

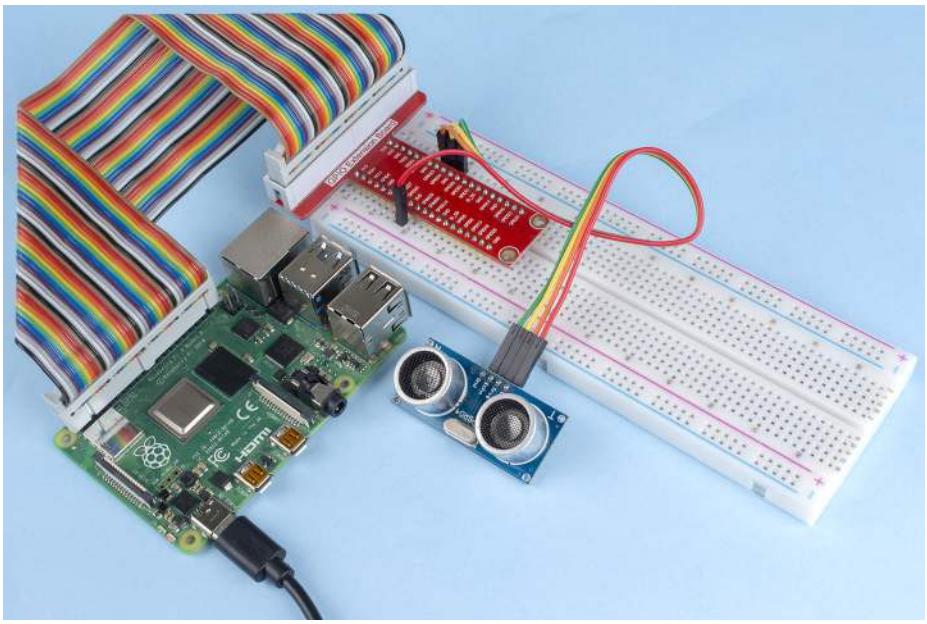
```
during = time2 - time1
```

Execute the interval calculation.

```
return during * 340 / 2 * 100
```

The distance is calculated in the light of time interval and the speed of sound propagation. The speed of sound in the air: 340m/s.

Phenomenon Picture



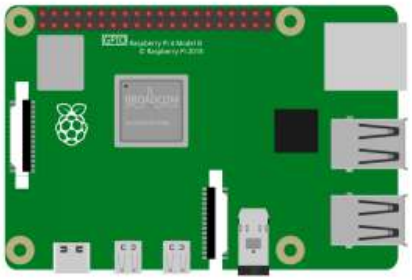
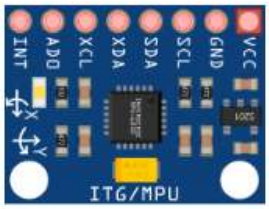
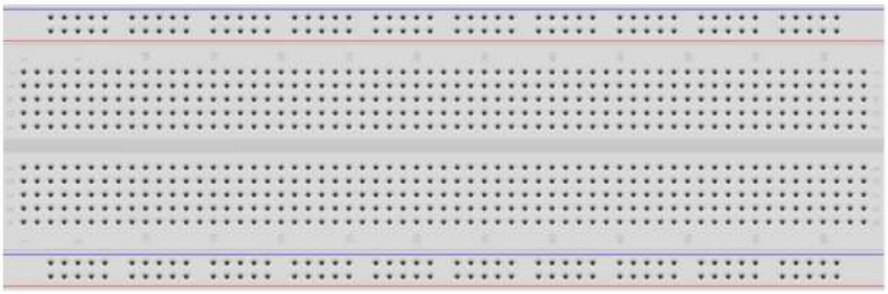
2.2.9 MPU6050 Module

Introduction

The MPU-6050 is the world's first and only 6-axis motion tracking devices (3-axis Gyroscope and 3-axis Accelerometer) designed for smartphones, tablets and wearable sensors that have these features, including the low power, low cost, and high performance requirements.

In this experiment, use I2C to obtain the values of the three-axis acceleration sensor and three-axis gyroscope for MPU6050 and display them on the screen.

Components

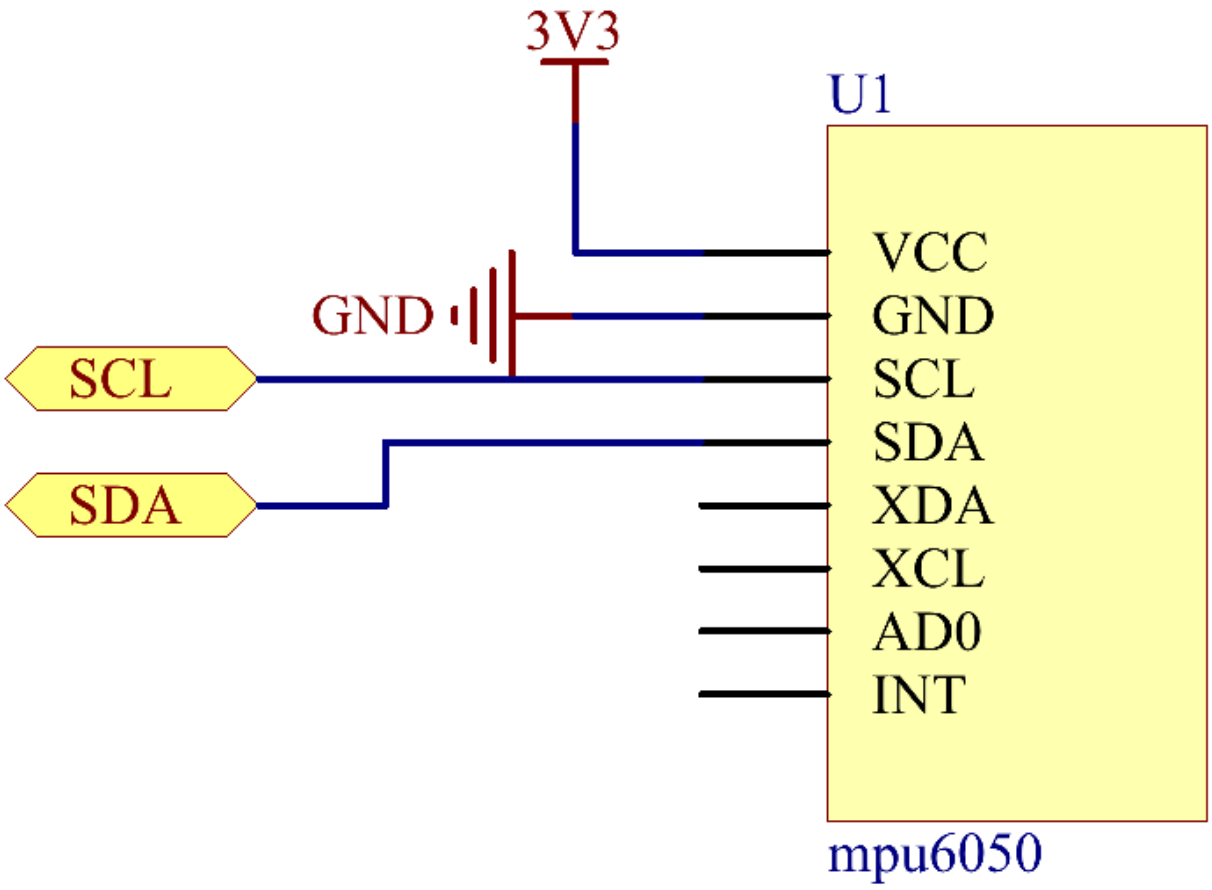
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
</td><td>1 * MPU6050
 
</td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *MPU6050 Module*

Schematic Diagram

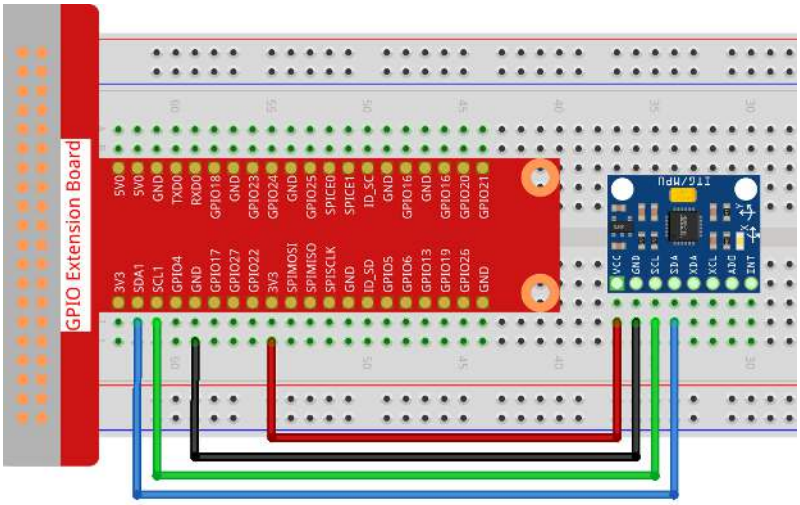
MPU6050 communicates with the microcontroller through the I2C bus interface. The SDA1 and SCL1 need to be connected to the corresponding pin.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see Appendix *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 4: Run the executable file.

```
sudo python3 2.2.9_mpu6050.py
```

With the code run, the angle of deflection of the x-axis and y-axis and the acceleration, angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

Note:

- If you get the error `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'`, you need to refer to *I2C Configuration* to enable the I2C.
- If you get `ModuleNotFoundError: No module named 'smbus2'` error, please run `sudo pip3 install smbus2`.
- If the error `OSError: [Errno 121] Remote I/O error` appears, it means the module is miswired or the module is broken.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import smbus
import math
import time

```

(continues on next page)

(continued from previous page)

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

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)


bus = smbus.SMBus(1) # or bus = smbus.SMBus(1) for Revision 2 boards
address = 0x68       # This is the address value read via the i2cdetect command

# Now wake the 6050 up as it starts in sleep mode
bus.write_byte_data(address, power_mgmt_1, 0)

while True:
    time.sleep(0.1)
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))

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

    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
```

(continues on next page)

(continued from previous page)

```
    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(1)
```

Code Explanation

```
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
```

Read sensor data sent from MPU6050.

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

Calculate the deflection angle of the y-axis.

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

Calculate the deflection angle of the x-axis.

```
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))
```

Read the values of the x axis, y axis and z axis on the gyroscope sensor, convert the metadata to angular velocity values, and then print them.

```
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
```

(continues on next page)

(continued from previous page)

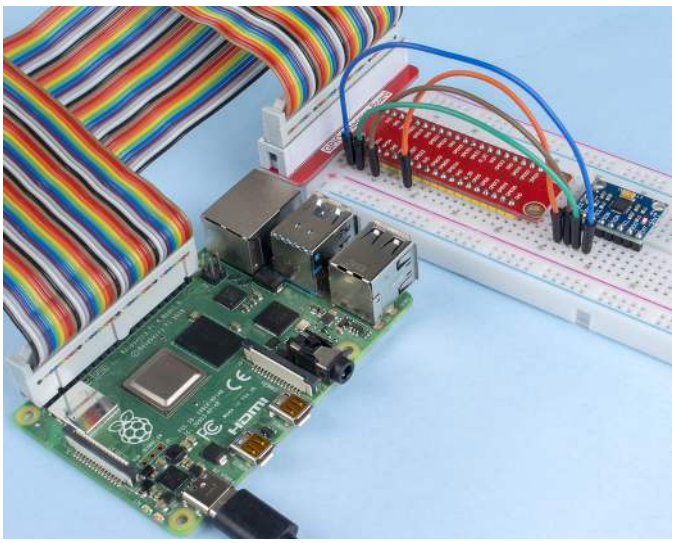
```
accel_zout_scaled = accel_zout / 16384.0

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)
```

Read the values of the x axis, y axis and z axis on the acceleration sensor, convert the elements to accelerated speed value (gravity unit), and print them.

```
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))
```

Print the deflection angles of the x-axis and y-axis.

Phenomenon Picture



2.2.10 MFRC522 RFID Module

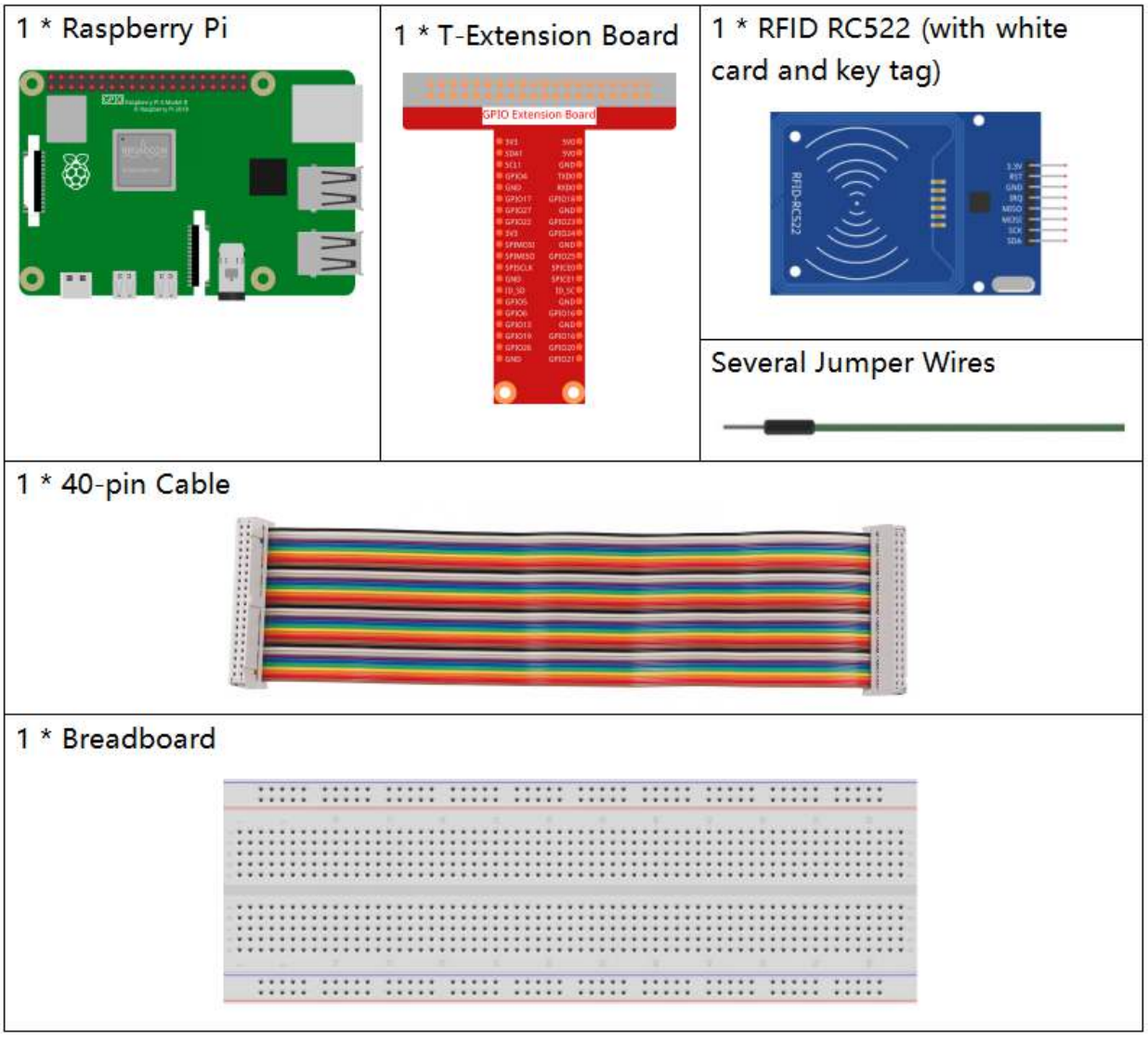
Introduction

Radio Frequency Identification (RFID) refers to technologies that use wireless communication between an object (or tag) and interrogating device (or reader) to automatically track and identify such objects.

Some of the most common applications for this technology include retail supply chains, military supply chains, automated payment methods, baggage tracking and management, document tracking and pharmaceutical management, to name a few.

In this project, we will use RFID for reading and writing.

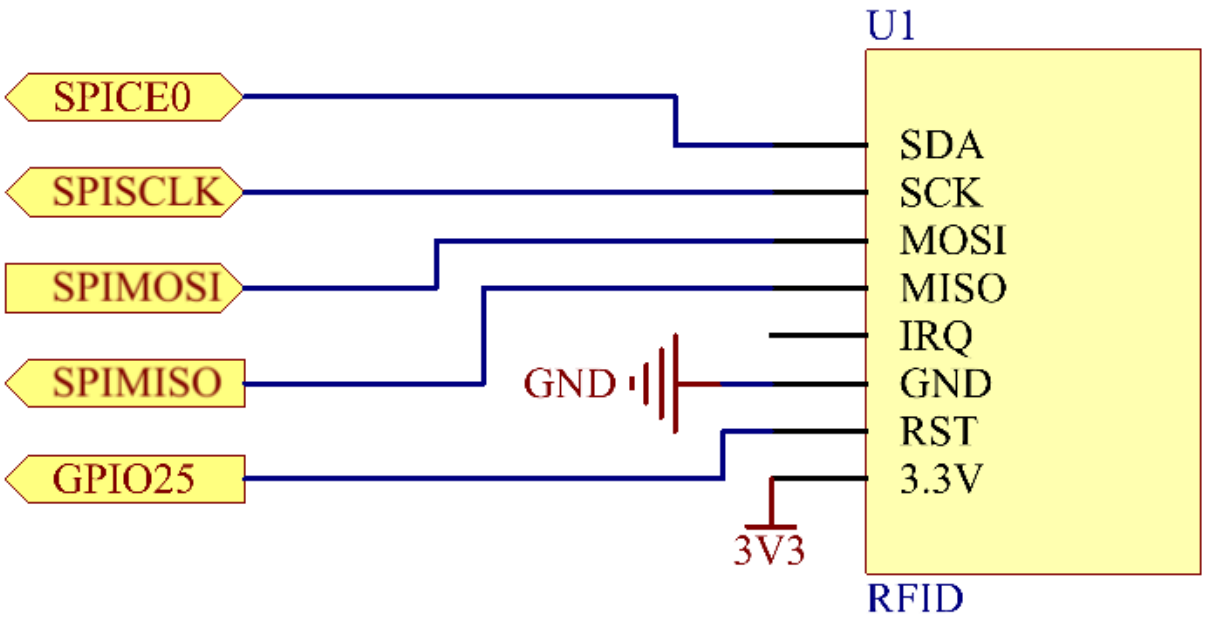
Components



- *GPIO Extension Board*
- *Breadboard*
- *MFRC522 Module*

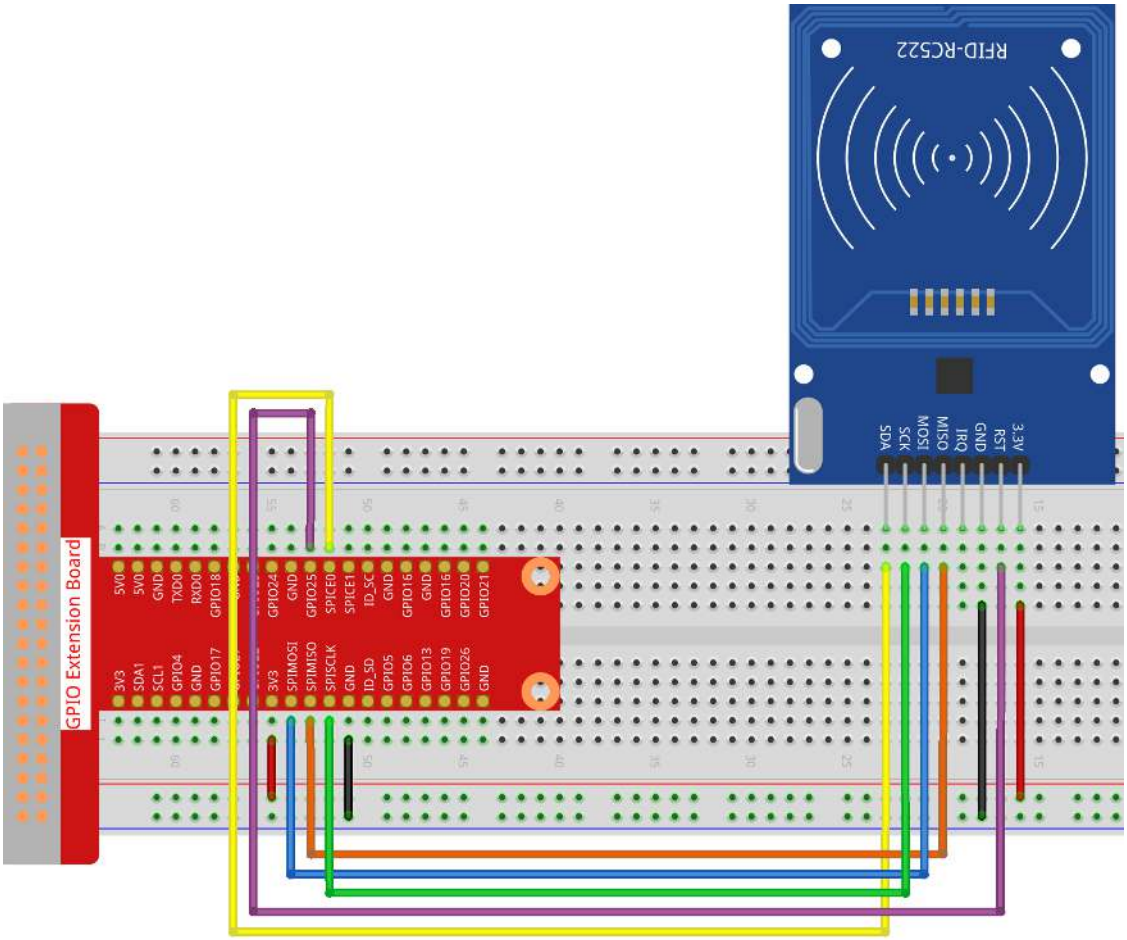
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Install the *Spidev and MFRC522* libraries.

Step 3: Set up SPI (refer to *SPI Configuration* for more details. If you have set SPI, skip this step.)

Step 4: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 5: After running `2.2.10_write.py`. You need to write a message first, press `Enter` to confirm, then put the card on the MFRC522 module, wait for "Data writing is complete" to appear and take the card away, or rewrite the message to another card and exit by `Ctrl+C`.

```
sudo python3 2.2.10_write.py
```

```
pi@raspberrypi:~/raphael-kit/python $ sudo python3 2.2.10_write.py
Please write new data:John
Please place the card to complete writing
Data writing is complete
Please write new data:
```

Step 6: Now run `2.2.10_read.py` to read the information of the tag or card you have written.

```
sudo python3 2.2.10_read.py
```

Code Explanation

```
reader = SimpleMFRC522()
```

Instantiate `SimpleMFRC522()` class.

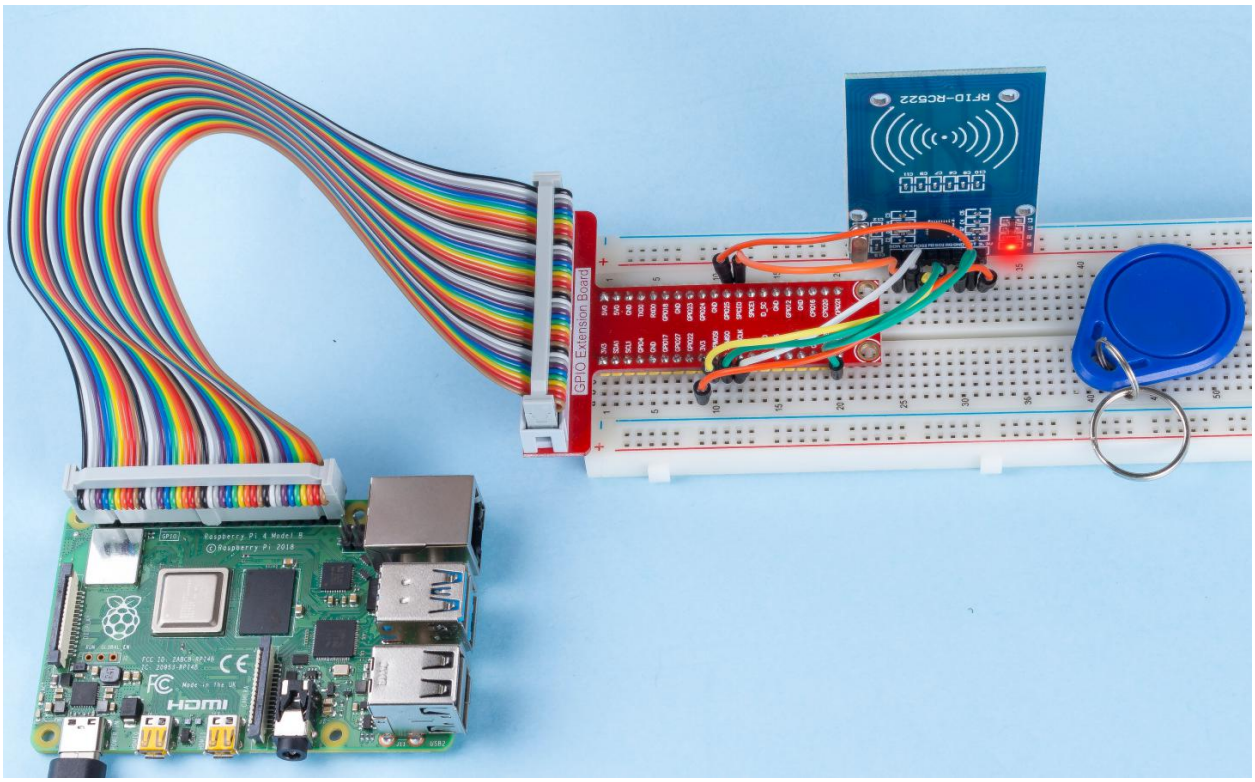
```
reader.read()
```

This function is used to read card data. If the reading is successful, id and text will be returned.

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

This function is used to write information to the card, press `Enter` key to finish writing. `text` is the information to be written to the card.

Phenomenon Picture



6.4 Audiovisual

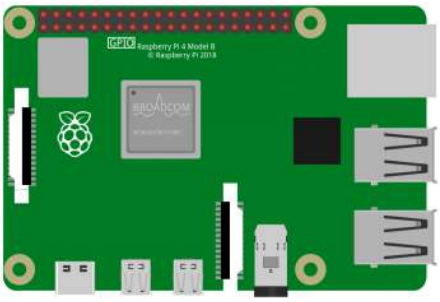
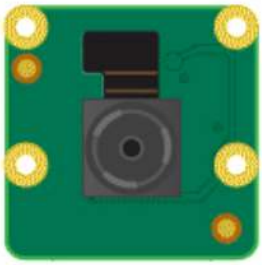
Note: When use the camera module, you may need a screen for a better experience, refer to: Connect your Raspberry Pi. Of course, if you don't have a screen, you can also access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

6.4.1 3.1.1 Photograph Module

Introduction

In this kit, equipped with a camera module, let's try to take a picture with Raspberry Pi.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * Camera Module
  </td></tr>
<tr><td colspan="2">1 * FFC Cable
 </td></tr>
</table>

For more information on how to connect the camera module and its configuration, please refer to *Camera Module*.

Experimental Procedures

Step 1: Go into the Raspberry Pi Desktop. You may need a screen for a better experience, refer to: Connect your Raspberry Pi. Or access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

Step 2: Open a Terminal and get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 3.1.1_PhotographModule.py
```

After the code runs, the camera will take a photo. Now you can see the photo named `my_photo.jpg` in the `/home/pi` directory.

Note: You can also open `3.1.1_PhotographModule.py` in the `/home/pi/raphael-kit/python/` path with a Python IDE, click Run button to run, and stop the code with Stop button.

If you want to download the photo to your PC, please refer to *Filezilla Software*.

Code

```
from picamera import PiCamera

camera = PiCamera()

def setup():
    camera.start_preview(alpha=200)

def main():
    camera.capture('/home/pi/my_photo.jpg')
    while True:
        pass

def destroy():
    camera.stop_preview()

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

Code Explanation

```
from picamera import PiCamera

camera = PiCamera()
```

Import the `picamera` library and instantiate the `PiCamera` class to use the camera module.

```
start_preview(**options)
```

Show the preview overlay and change the transparency level of the preview with `alpha` - from 0 to 255. This method starts a camera preview as an overlay on the Pi's primary display (HDMI or composite). By default, the renderer will

be opaque and fullscreen.

This means the default preview overrides whatever is currently visible on the display. More specifically, the preview does not rely on a graphical environment like X-Windows (it can run quite happily from a TTY console); it is simply an overlay on the Pi's video output. To stop the preview and reveal the display again, call `stop_preview()` . The preview can be started and stopped multiple times during the lifetime of the `PiCamera` object.

```
camera.capture('/home/pi/my_photo.jpg')
```

Capture an image from the camera, storing it in `/home/pi/`.

Note: You can use `camera.capture()` function and `for` loop together to achieve continuous shooting. And use the `delay` function to adjust the time interval for taking pictures.

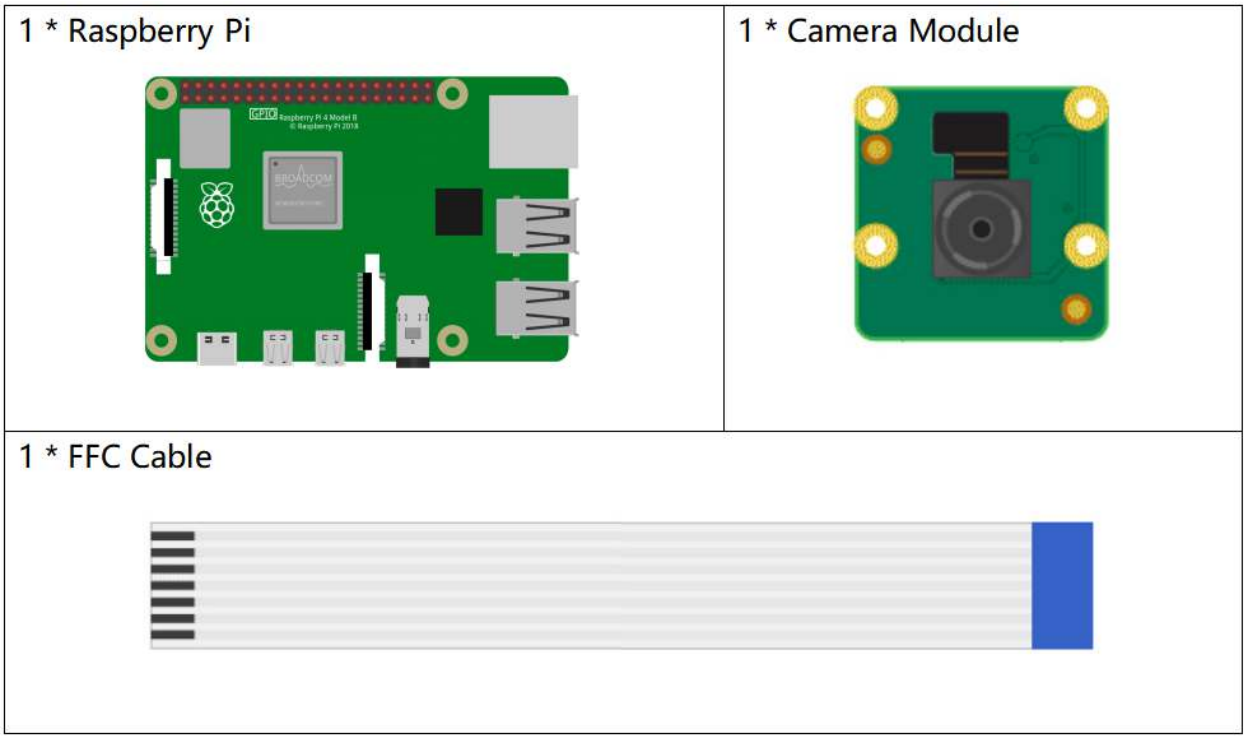
```
for i in 5:
    camera.capture('/home/pi/my_photo%s.jpg' % i)
```

6.4.2 3.1.2 Video Module

Introduction

In addition to taking photos, the Camera Module can also be used to record videos.

Components



For more information on how to connect the camera module and its configuration, please refer to *Camera Module*.

Experimental Procedures

Step 1: Go into the Raspberry Pi Desktop. You may need a screen for a better experience, refer to: Connect your Raspberry Pi. Or access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

Step 2: Open a Terminal and get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 3.1.2_VideoModule.py
```

Run the code to start recording. Press `Ctrl+C` to end the recording. Name the video `my_video.h264` and store it in the `/home/pi` directory.

Note: You can also open `3.1.2_PhotographModule.py` in the `/home/pi/raphael-kit/python/` path with a Python IDE, click Run button to run, and stop the code with Stop button.

If you want to send photos to your PC, please refer to *Filezilla Software*.

Code

```
from picamera import PiCamera

camera = PiCamera()

def setup():
    camera.start_preview(alpha=200)

def main():
    camera.start_recording('/home/pi/my_video.h264')
    while True:
        pass

def destroy():
    camera.stop_recording()
    camera.stop_preview()

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

Code Explanation

```
start_recording(output, format=None, resize=None, splitter_port=1, **options)
```

Start recording video from the camera, storing it in `output`.

```
camera.stop_recording()
```

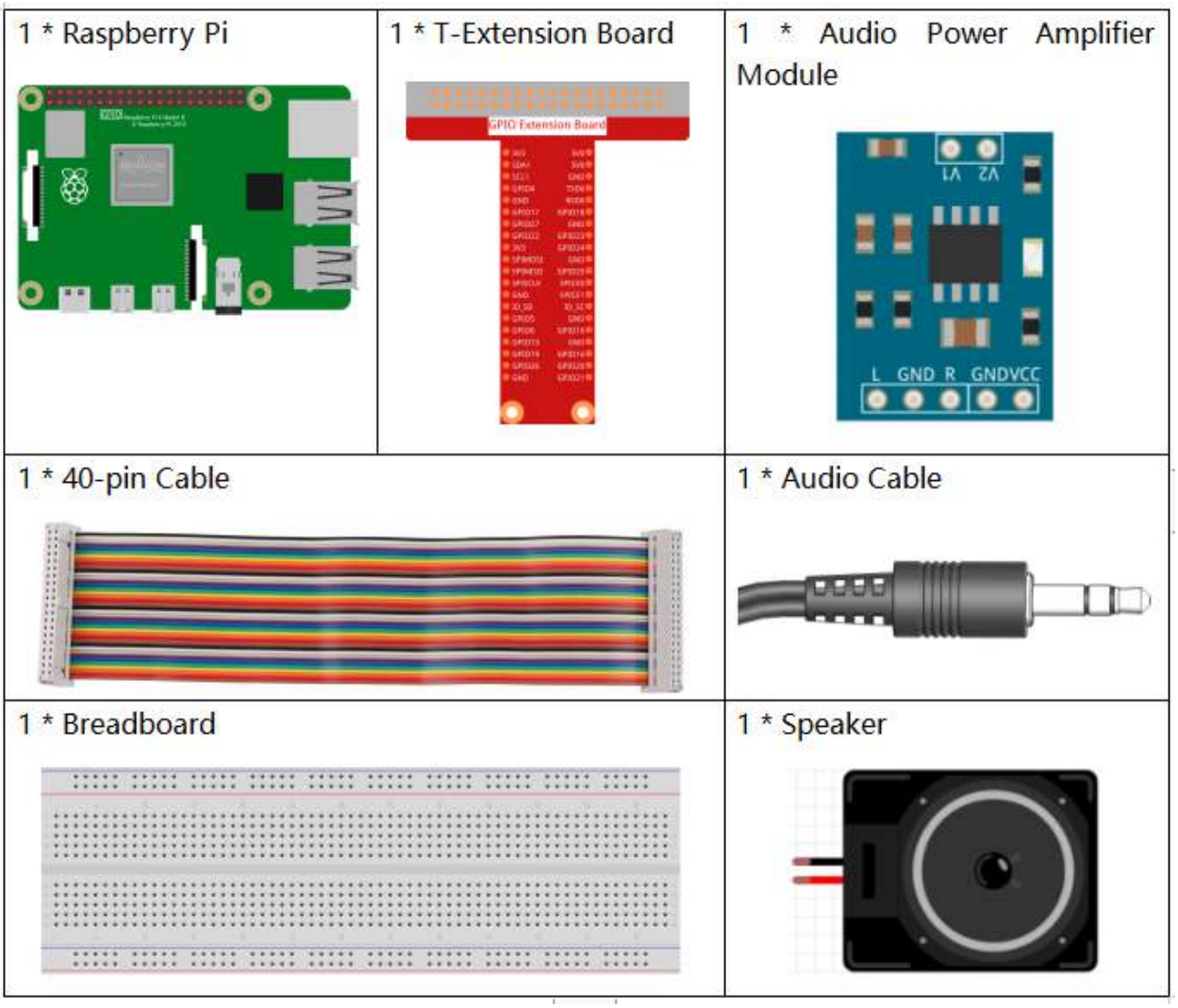
End the recording.

6.4.3 3.1.3 Audio Module

Introduction

In this project, let's make a DIY stereo with an audio power amplifier module, 8ohm/2w speakers and a 3.5mm Audio cable.

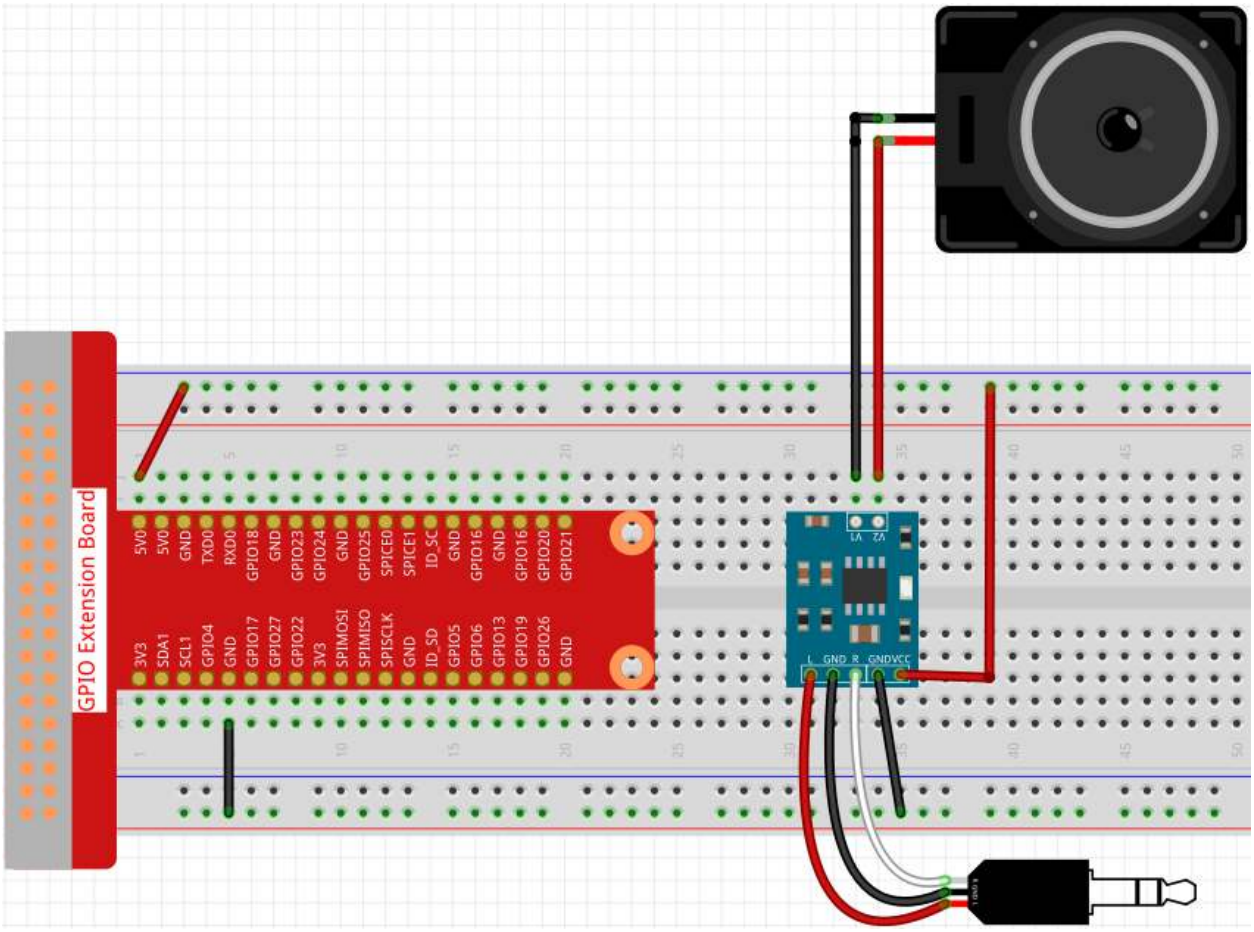
Components



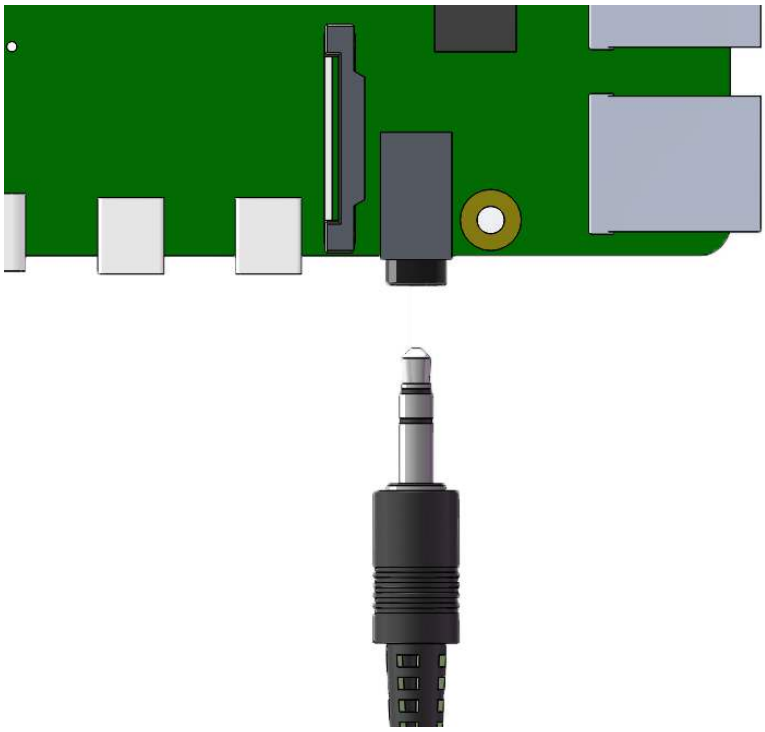
- *GPIO Extension Board*
- *Breadboard*
- *Audio Module and Speaker*

Experimental Procedures

Step 1: Build the circuit.



After building the circuit according to the above diagram, then plug the audio cable into the Raspberry Pi's 3.5mm audio jack.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
python3 3.1.3_AudioModule.py
```

After the code runs, you can enjoy the music.

Note: If your speaker have no sound, it may be because the Raspberry Pi has selected the wrong audio output (The default is HDMI), you need to *Change Audio Output* to Headphones.

If you feel that the volume of the speakers is too low, you can *Adjust Volume*.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
from pygame import mixer

mixer.init()

def main():
    mixer.music.load('/home/pi/raphael-kit/music/my_music.mp3')
    mixer.music.set_volume(0.7)
    mixer.music.play()
    while True:
```

(continues on next page)

(continued from previous page)

```
        pass# Don't do anything.

def destroy():
    mixer.music.stop()

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

Code Explanation

```
from pygame import mixer

mixer.init()
```

Import the `mixer` method in the `pygame` library and initialize the method.

```
mixer.music.load('/home/pi/raphael-kit/music/my_music.mp3')
mixer.music.set_volume(0.7)
mixer.music.play()
```

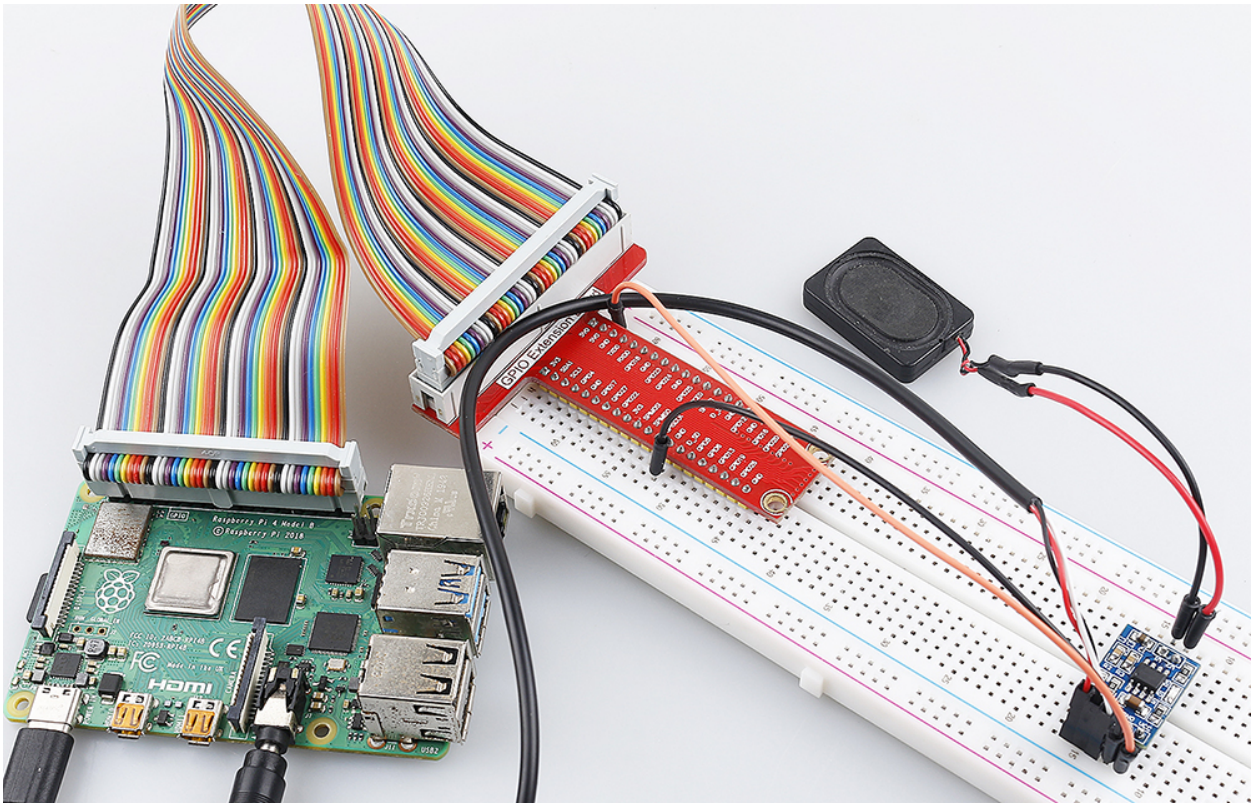
This code reads the `my_music.mp3` file in the `/home/pi/raphael-kit/music` directory and sets the volume to 0.7(The range is 0~1). The Raspberry Pi will start playing audio When `mixer.music.play()` is called.

Note: You can also upload other music files to your Raspberry Pi. For a detailed tutorial, please refer to: *Filezilla Software*.

```
mixer.music.stop()
```

Calling `mixer.music.stop()` will stop playing audio. In addition, you can also pause with `mixer.music.pause()` and continue with `mixer.music.unpause()`.

Phenomenon Picture



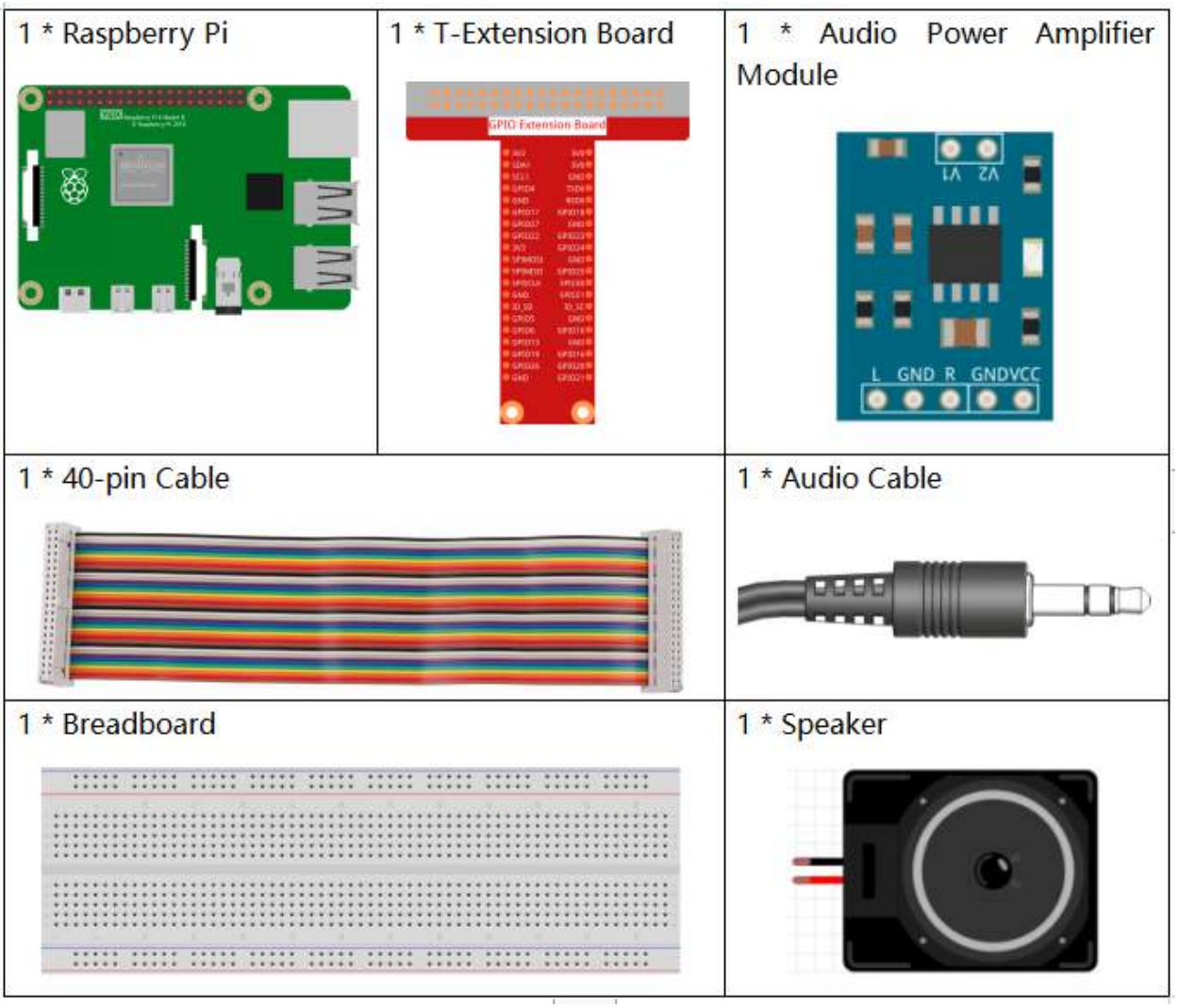
6.4.4 3.1.4 Text-to-speech

Introduction

In many places, we can come into contact with TTS (Text-to-speech) technology, which converts text into natural-sounding speech and brings people a good interactive experience.

Let's try to make your project speak.

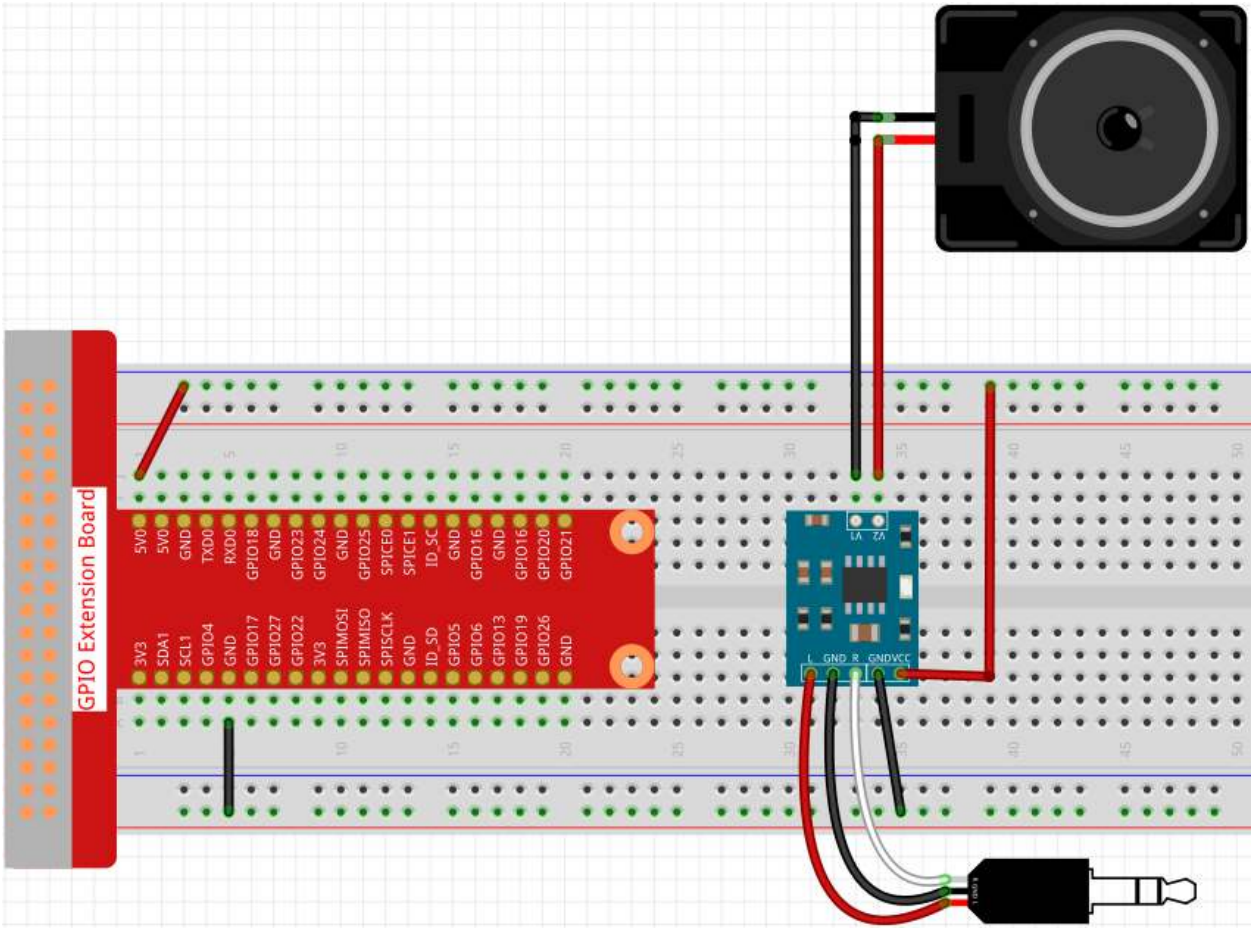
Components



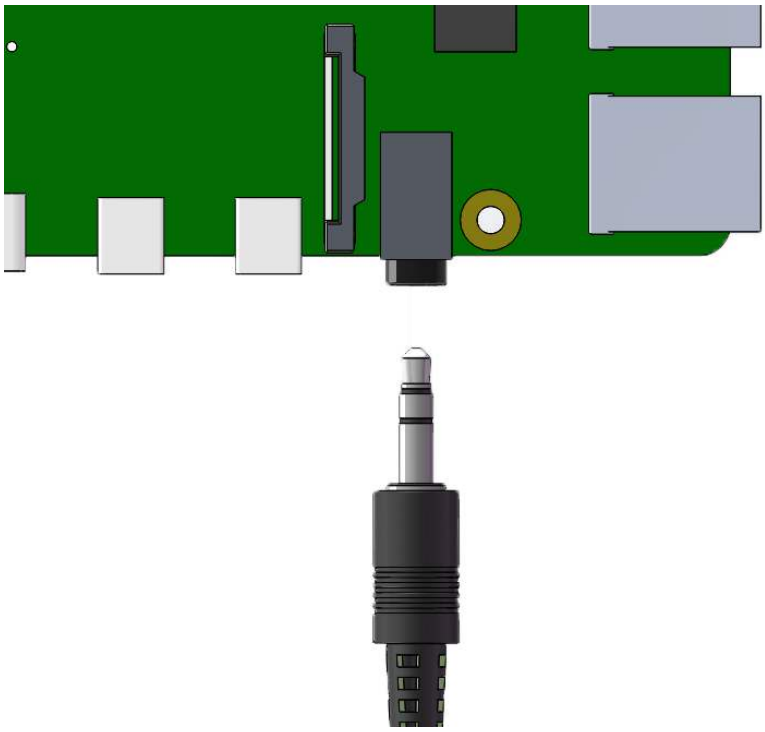
- *GPIO Extension Board*
- *Breadboard*
- *Audio Module and Speaker*

Experimental Procedures

Step 1: Build the circuit.



After building the circuit according to the above diagram, then plug the audio cable into the Raspberry Pi's 3.5mm audio jack.



Step 2: Install `espeak` module.

```
sudo apt-get install espeak -y
```

Step 3: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 4: Run.

```
python3 3.1.4_Text-to-speech.py
```

Raspberry pi will greet you kindly after the code runs, and it will say goodbye to you when the code stops.

Note: If your speaker have no sound, it may be because the Raspberry Pi has selected the wrong audio output (The default is HDMI), you need to *Change Audio Output* to Headphones.

If you feel that the volume of the speakers is too low, you can *Adjust Volume*.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect. After confirming that there are no problems, you can use the Copy button to copy the modified code, then open the source code in Terminal via `nano` cammand and paste it.

```
from tts import TTS

tts = TTS(engine="espeak")
```

(continues on next page)

(continued from previous page)

```
tts.lang('en-US')

def main():
    tts.say('Hello, nice to meet you!')

def destroy():
    tts.say('See you later')

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

Code Explanation

```
from tts import TTS

tts = TTS(engine="espeak")
```

Import the TTS class and instantiate an object.

```
tts.lang('en-US')
```

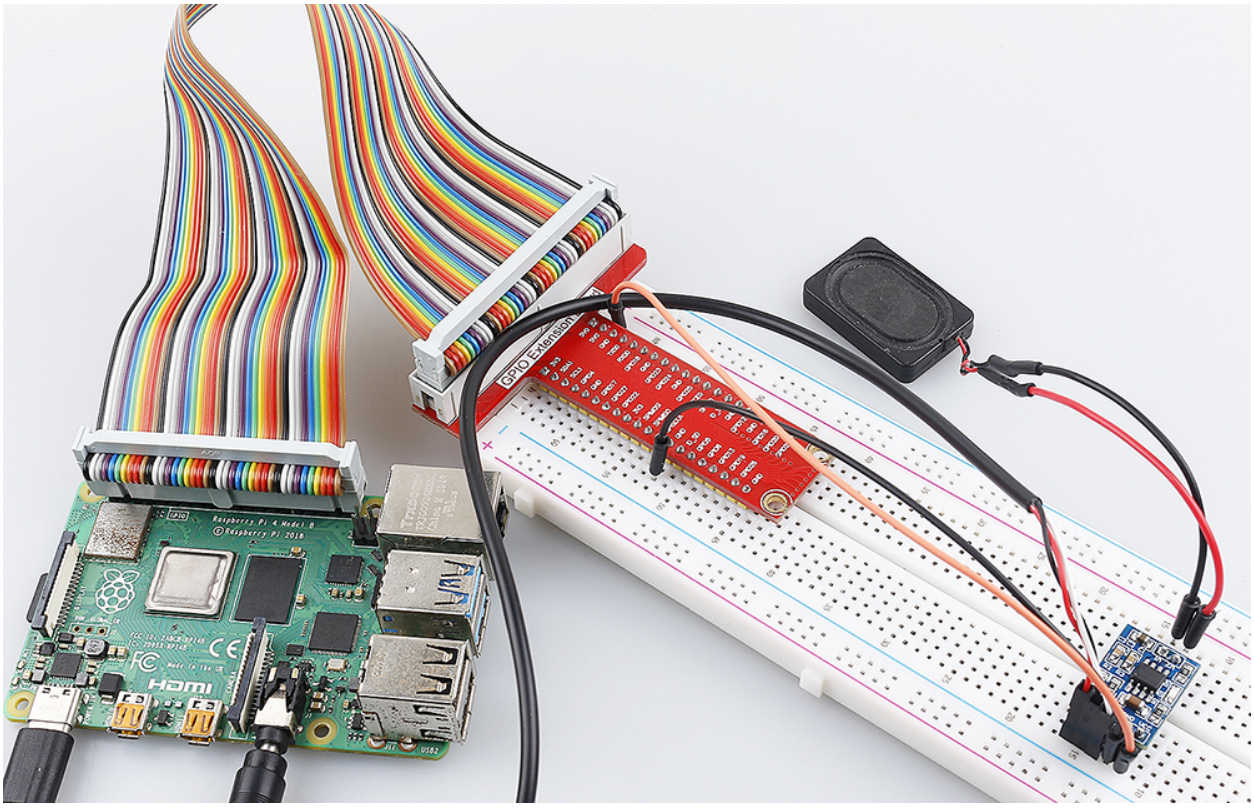
Set the language.

Note: Currently the switchable language only supports English.

```
tts.say("Hello, nice to meet you!")
```

Fill in the text to be said as a parameter, after executing `tts.say()`, Raspberry Pi will say the text you wrote.

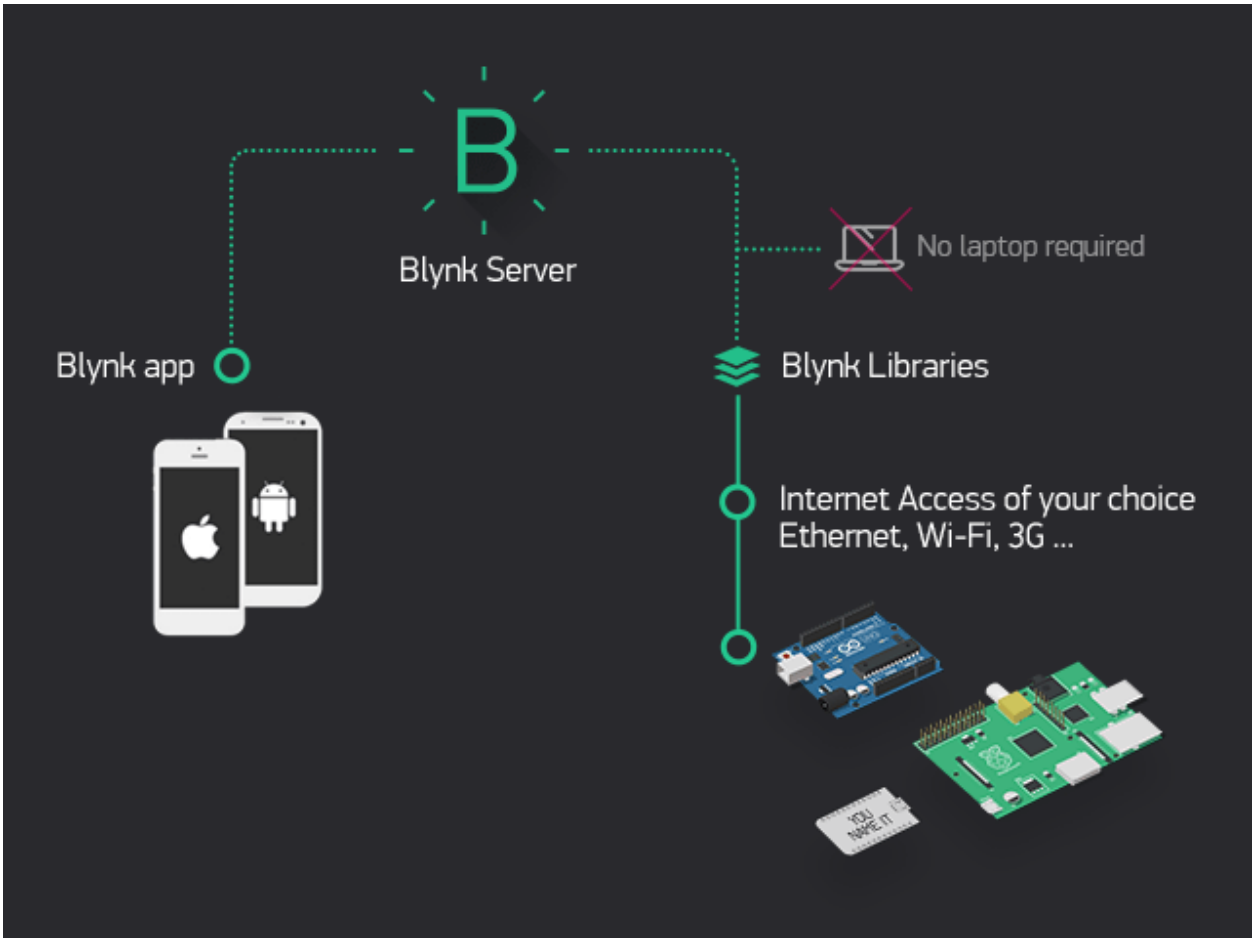
Phenomenon Picture



6.5 IoT

In this article, we'll show you how to connect your Raspberry Pi to the **Blynk** platform to make some interesting IoT projects.

Blynk is a full suite of software required to prototype, deploy, and remotely manage connected electronic devices at any scale: from personal IoT projects to millions of commercial connected products. With Blynk anyone can connect their hardware to the cloud and build a no-code iOS, Android, and web applications to analyze real-time and historical data coming from devices, control them remotely from anywhere in the world, receive important notifications, and much more. . .



6.5.1 Get Start with Blynk

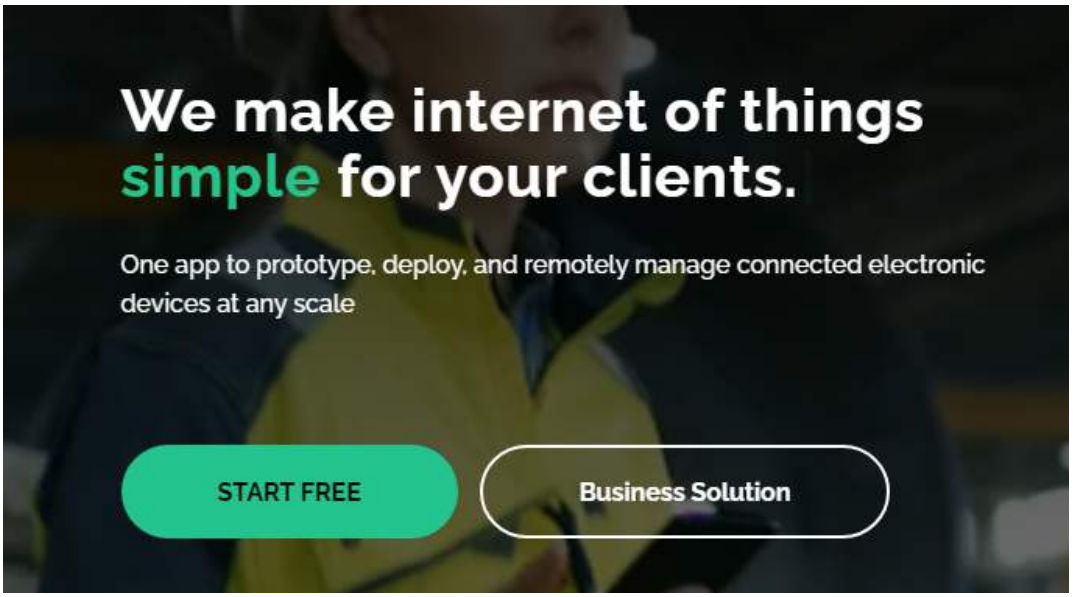
You will learn how to use Blynk in this project.

In the event that you trigger widgets on Blynk, your Raspberry Pi will print out their values.

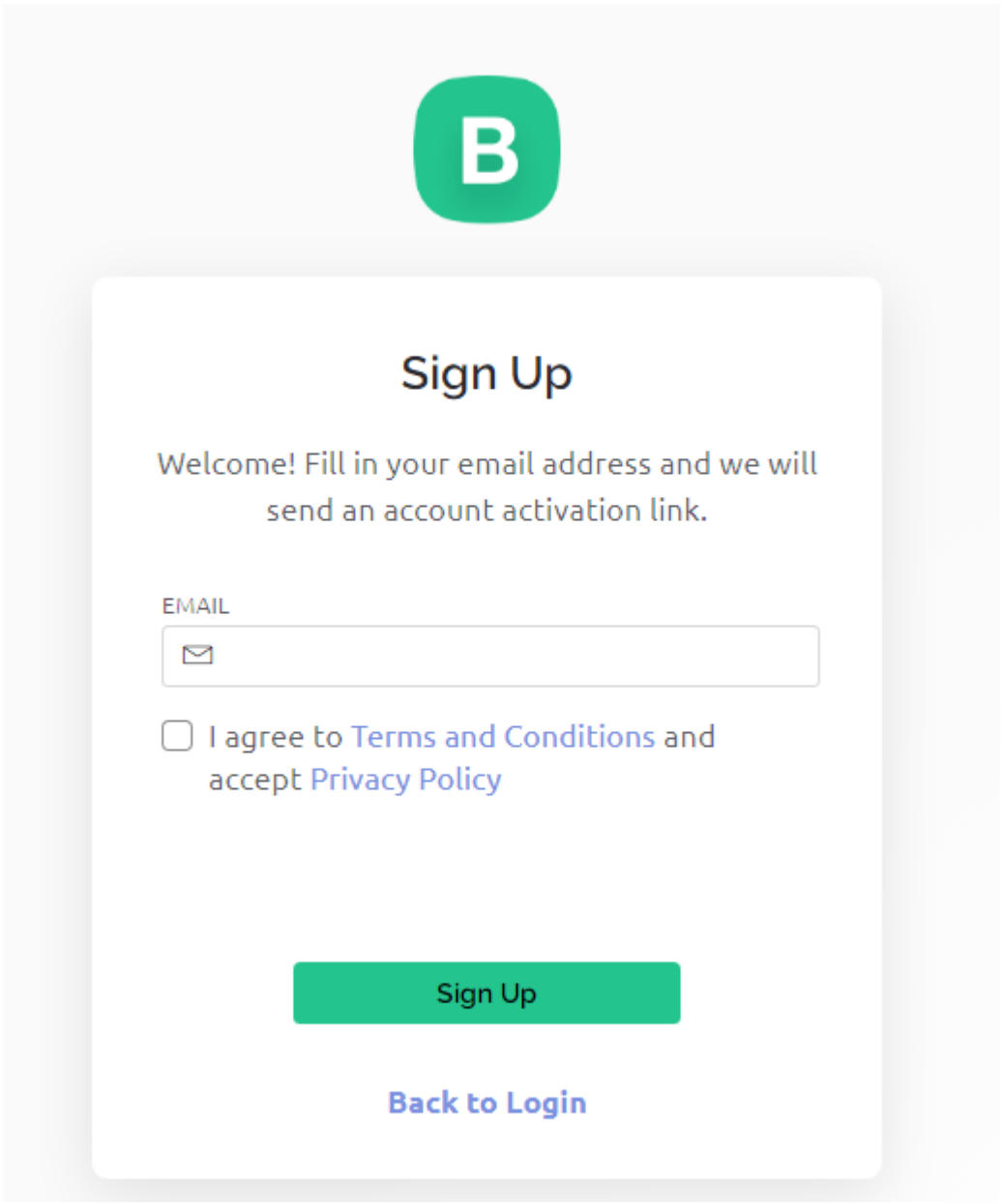
Follow the steps below, and note that you must do them in order and not skip any chapters.

1. Configuring the Blynk

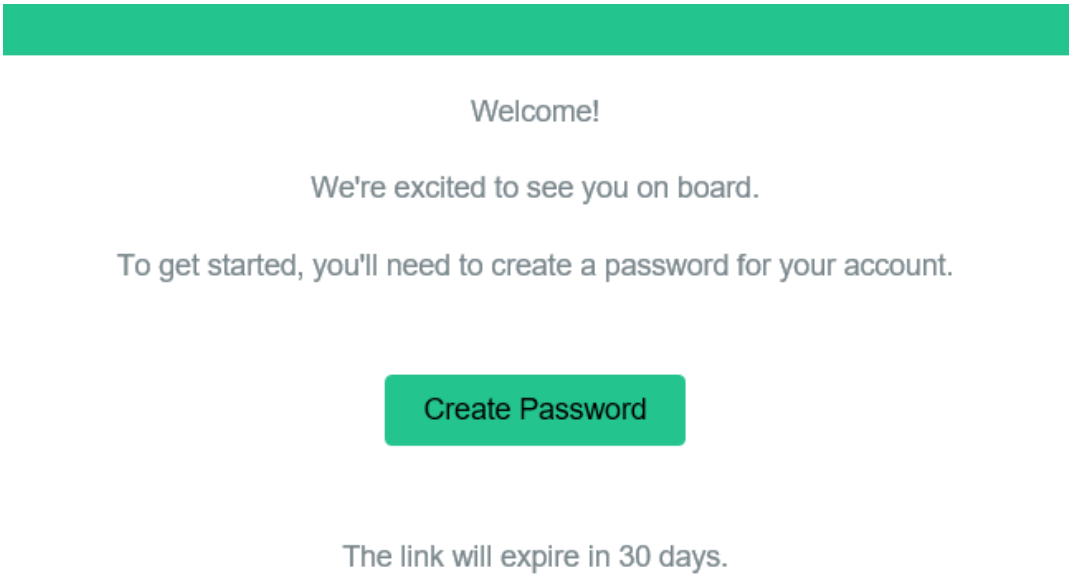
1. Go to the BLYNK and click **START FREE**.



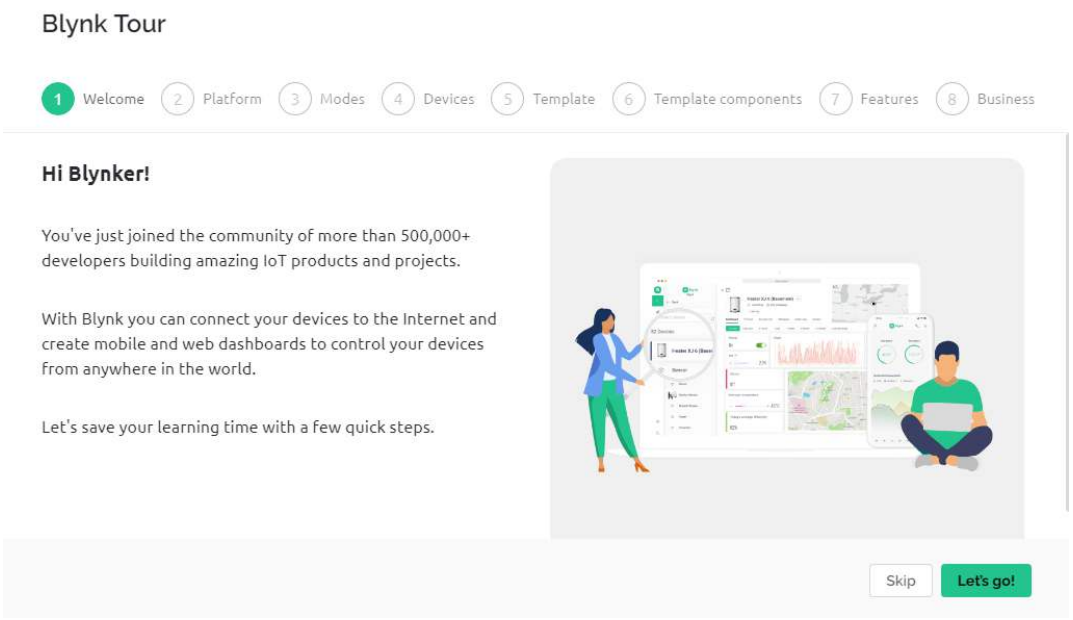
2. Fill in your email address to register an account.



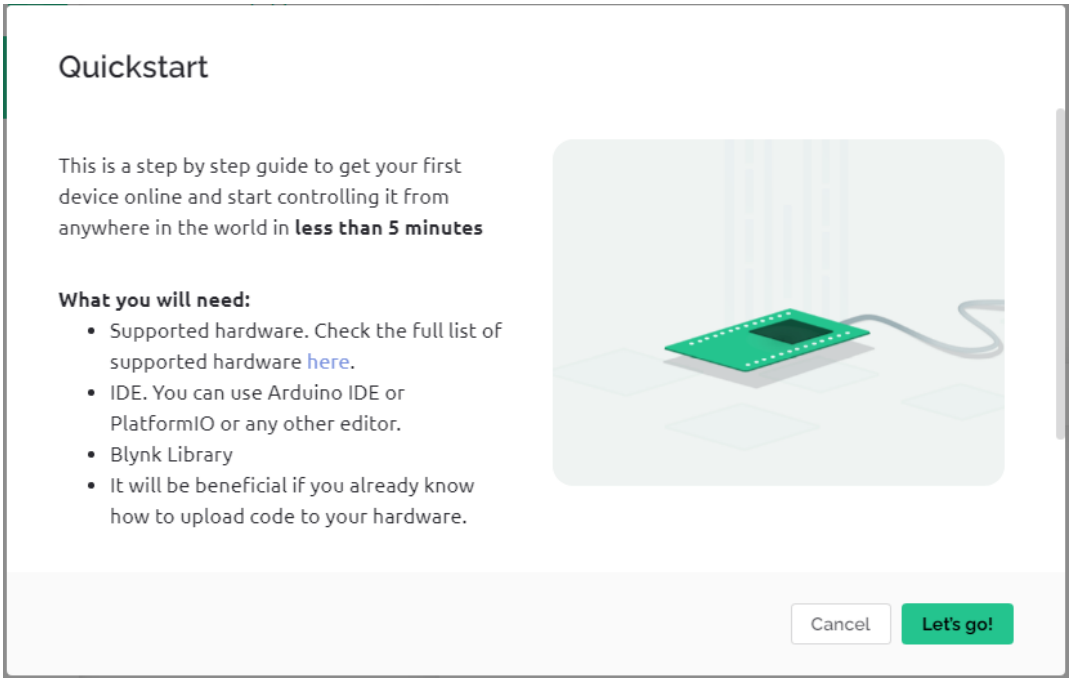
3. Go to your email address to complete your account registration.



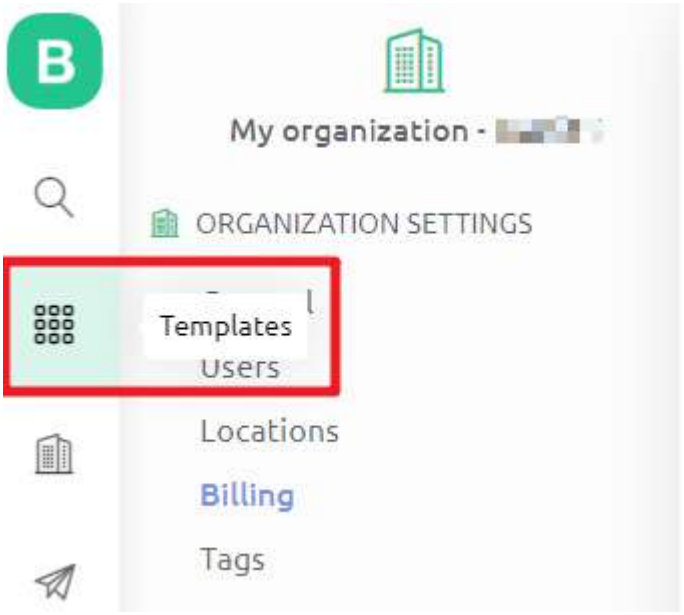
4. Afterwards, **Blynk Tour** will appear and you can read it to learn the basic information about the Blynk.



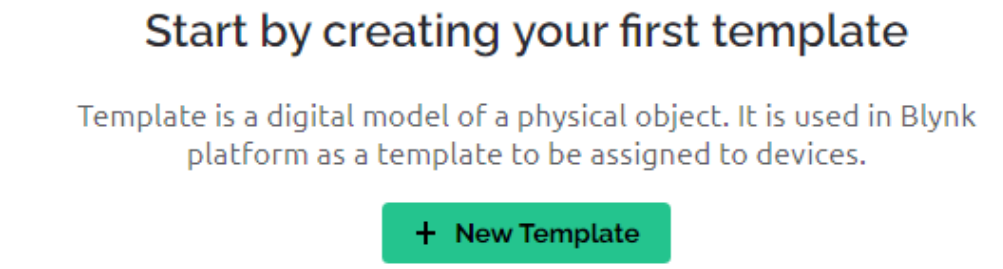
5. Next we need to create a template and device, click **Cancel**.



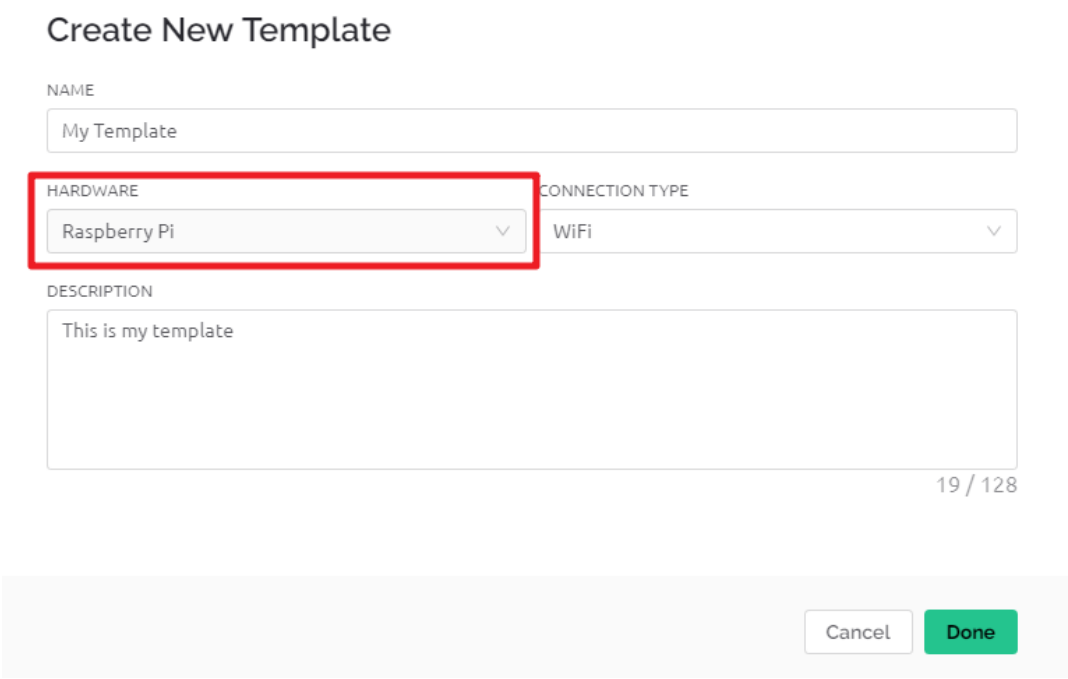
6. Go to Template from the navigation bar.



7. Create New Template

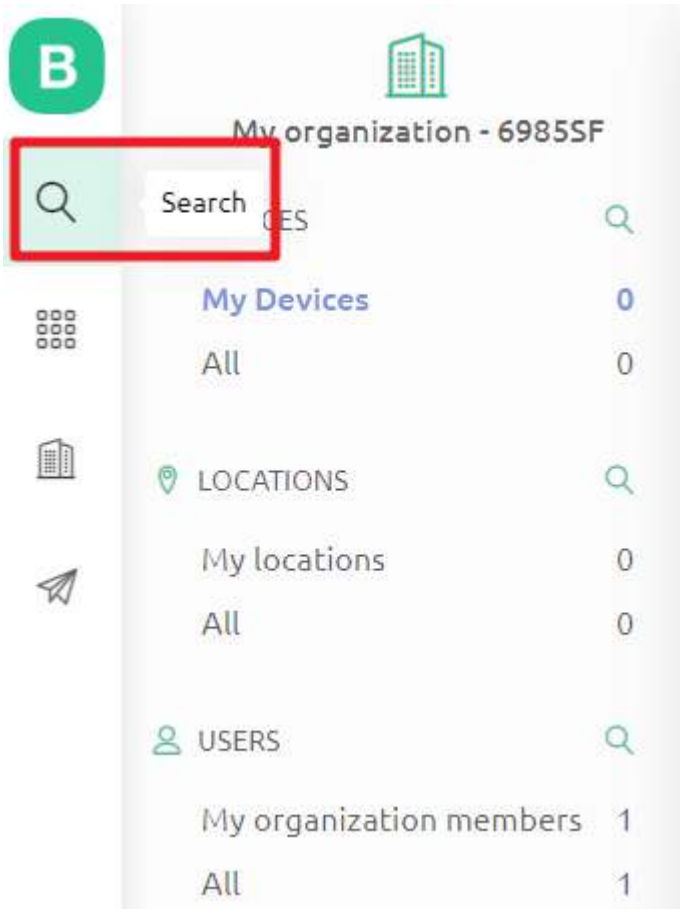


8. Fill in **NAME**, feel free to do so; **HARDWARE** should be **Raspberry Pi**. Then **Done**.

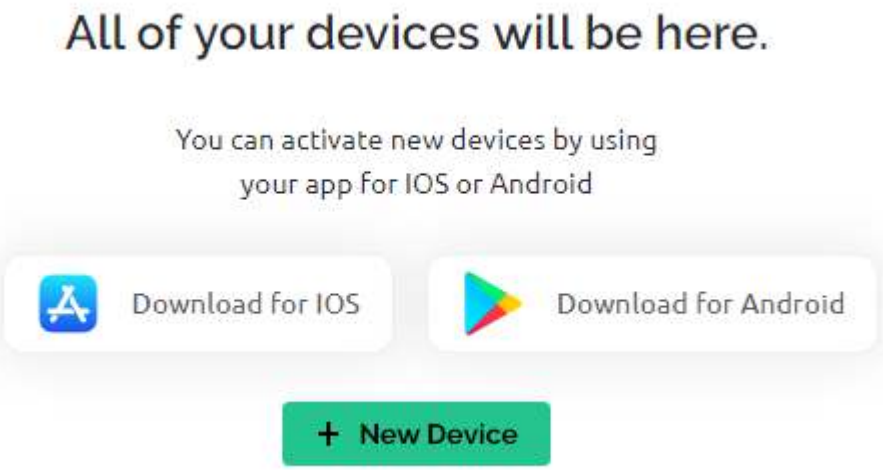


9. You will be redirected to the Info page, just click on save in the top right corner.

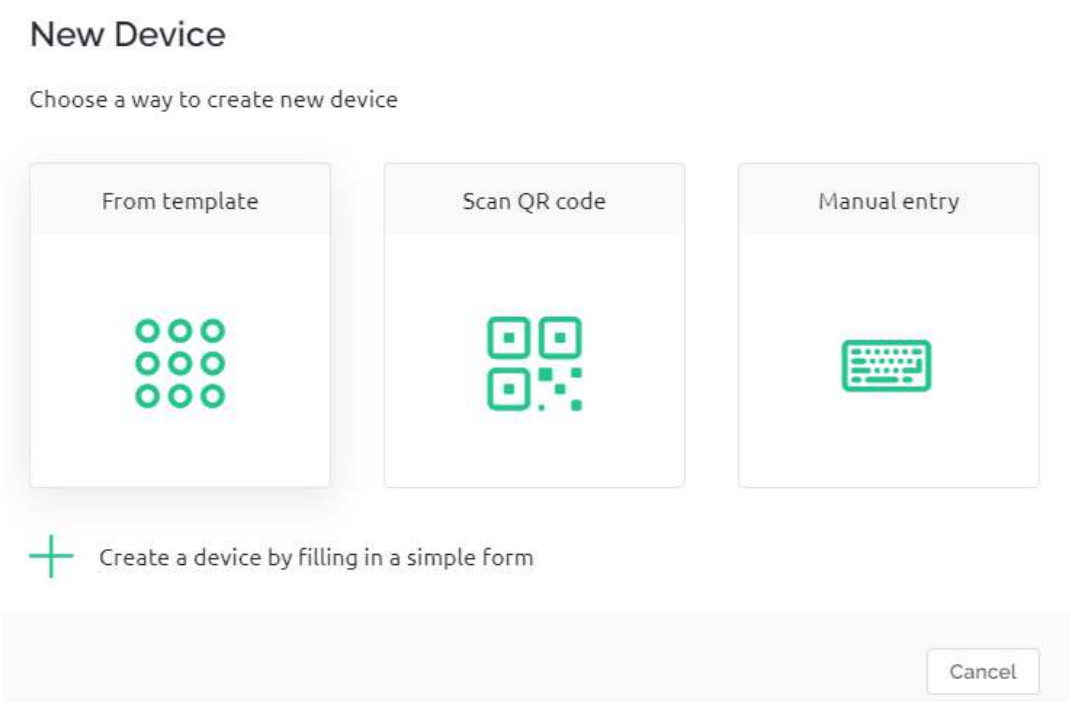
10. Go to **Search** page from the navigation bar.



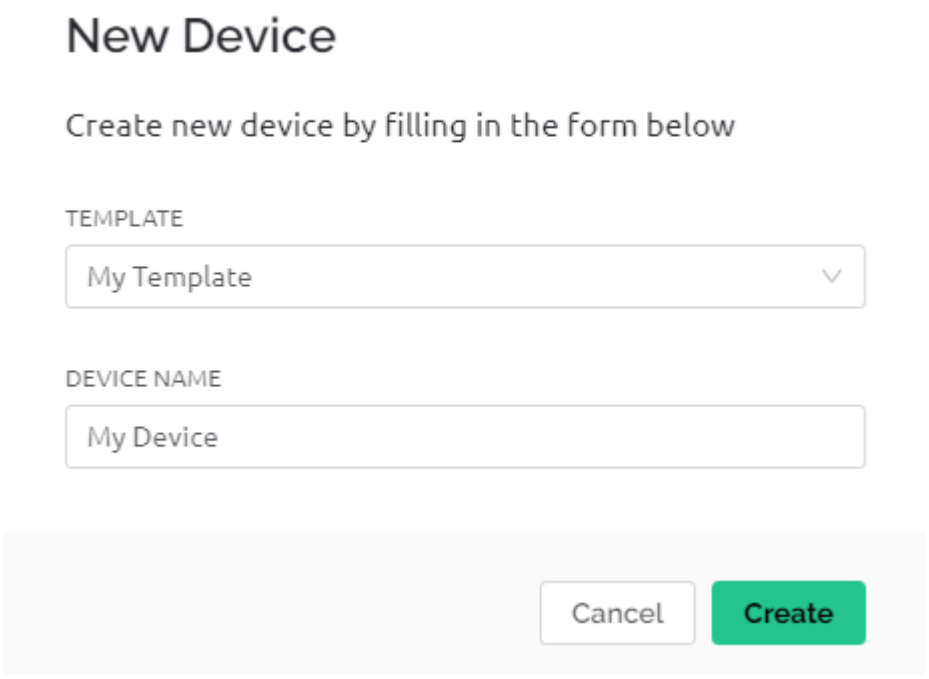
11. Create New Device.



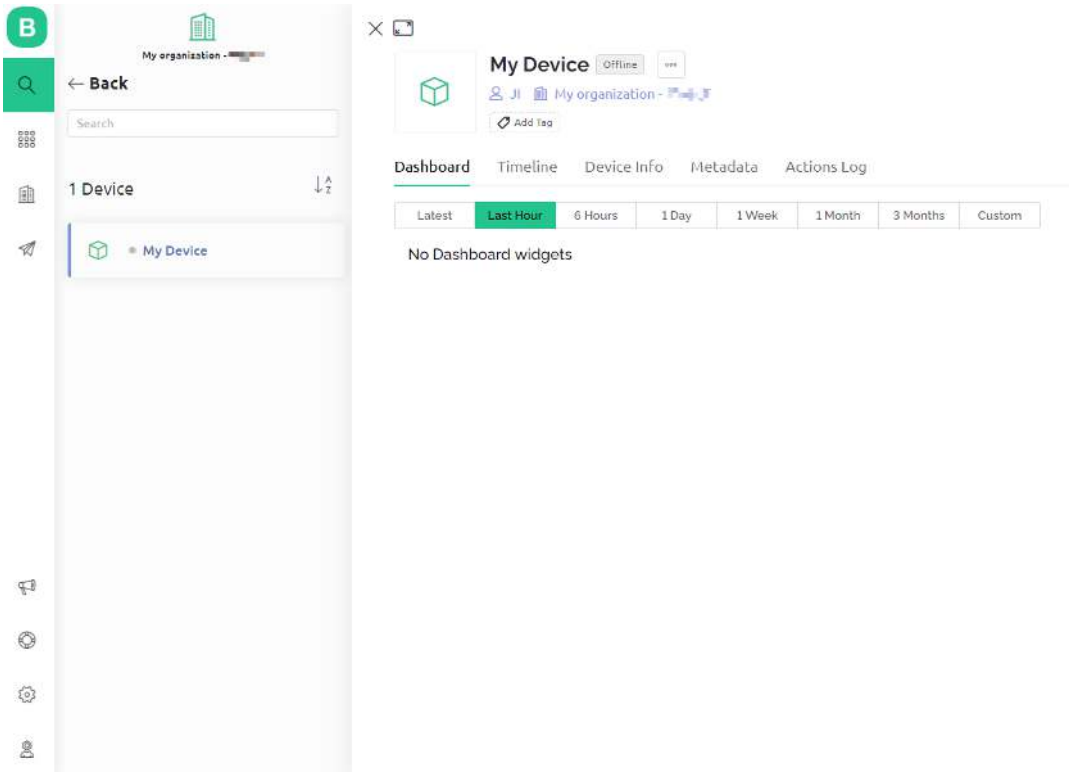
12. From template.



13. Select **TEMPLATE** as the one you just set, **DEVICE NAME** can be customized. Then click Create.

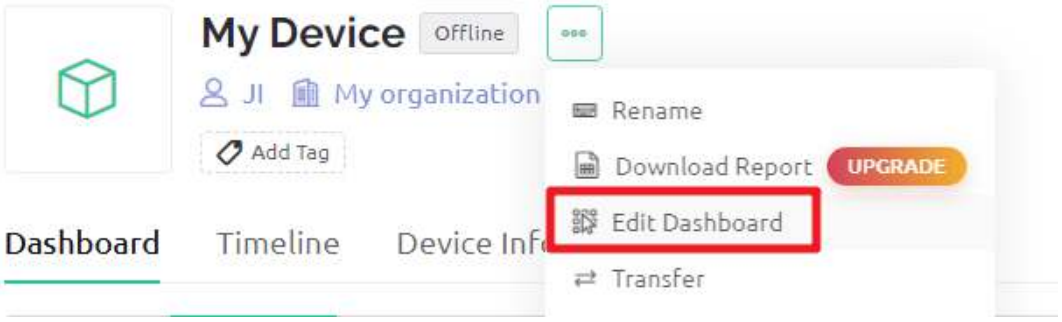


14. You should now see a page like this one, which means that your initial Blynk setup is complete.

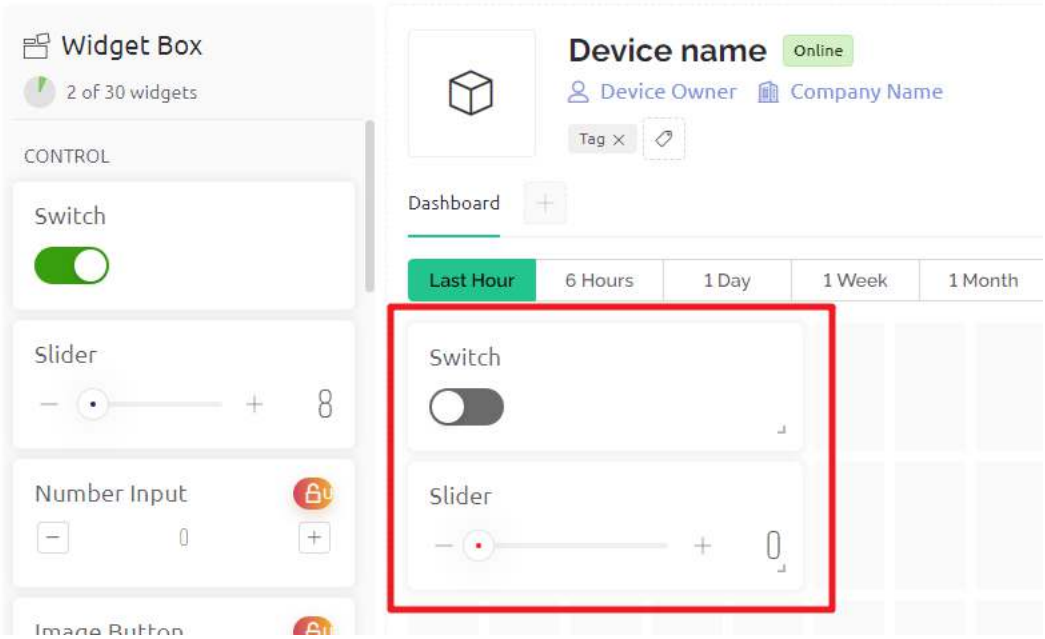


2. Edit Dashboard

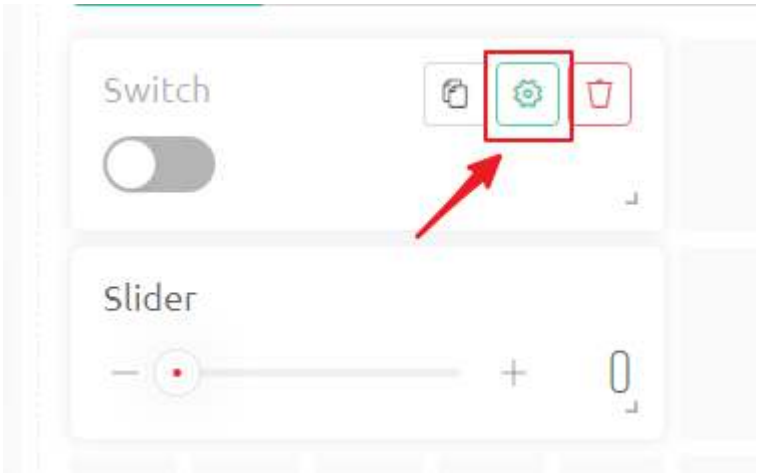
1. Click on the menu icon in the upper right corner and select edit dashboard.



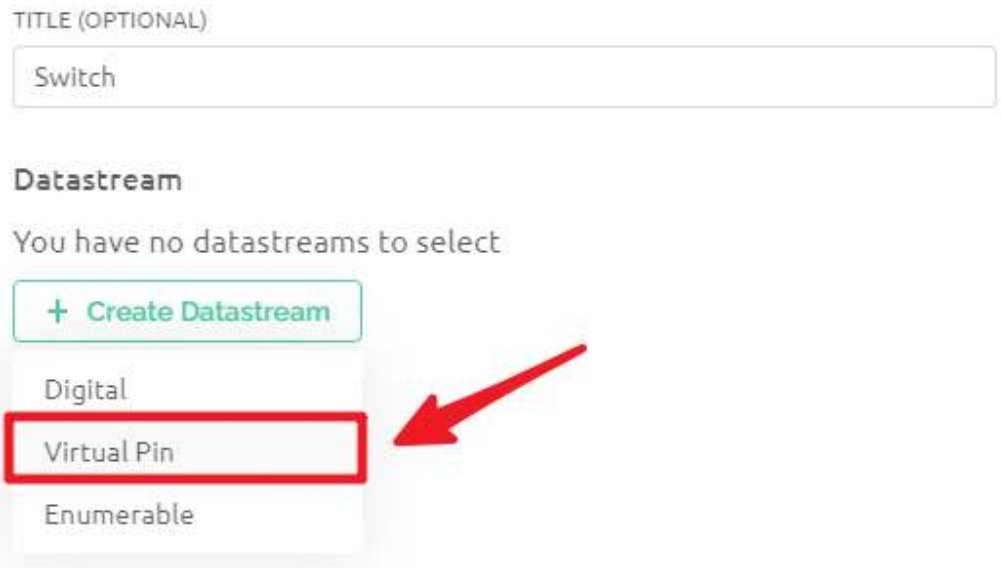
2. Then drag any CONTROL Widgets you want onto the Dashboard. I dragged a Switch and a Slider.



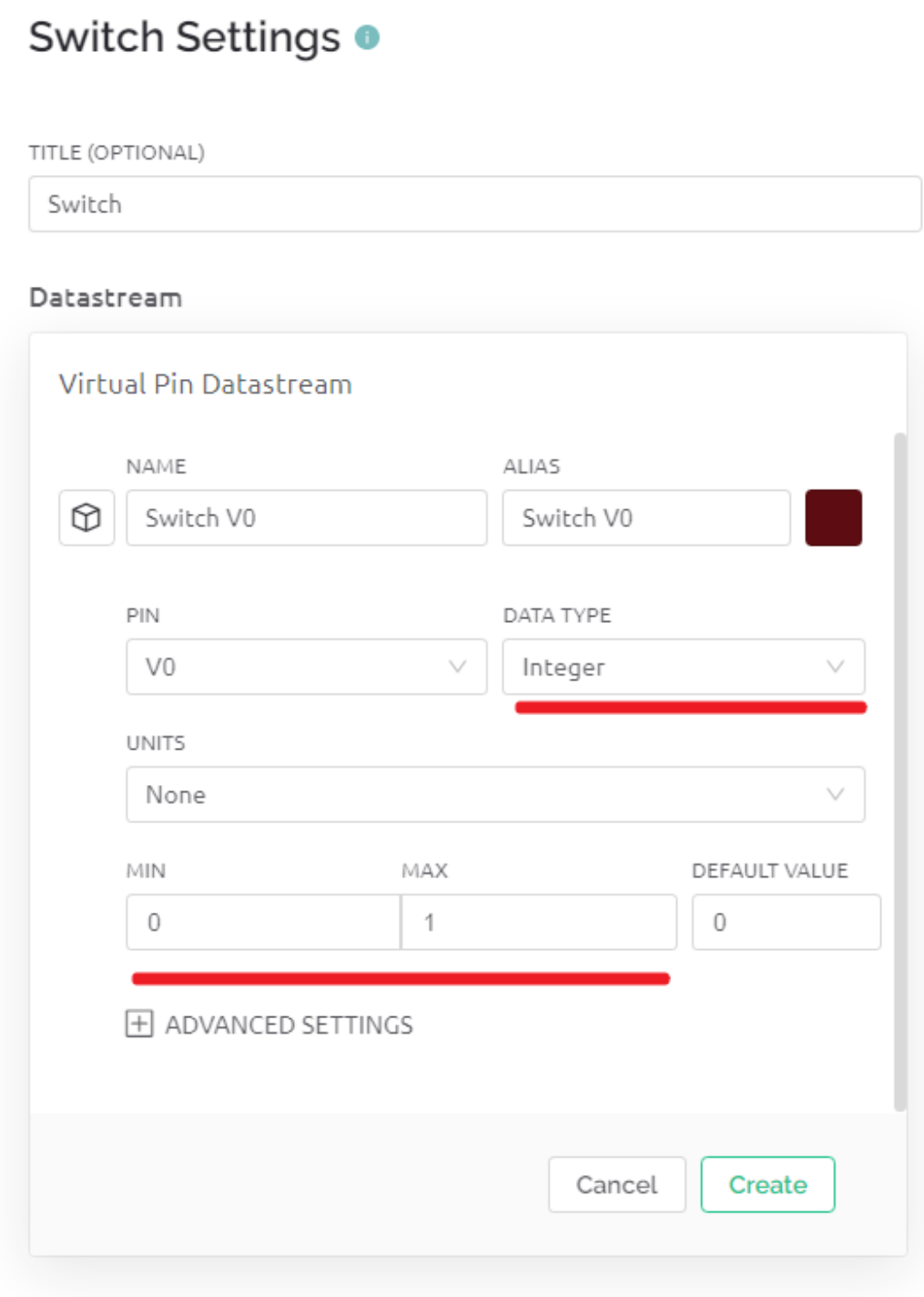
3. Tap the setting icon on the widget.



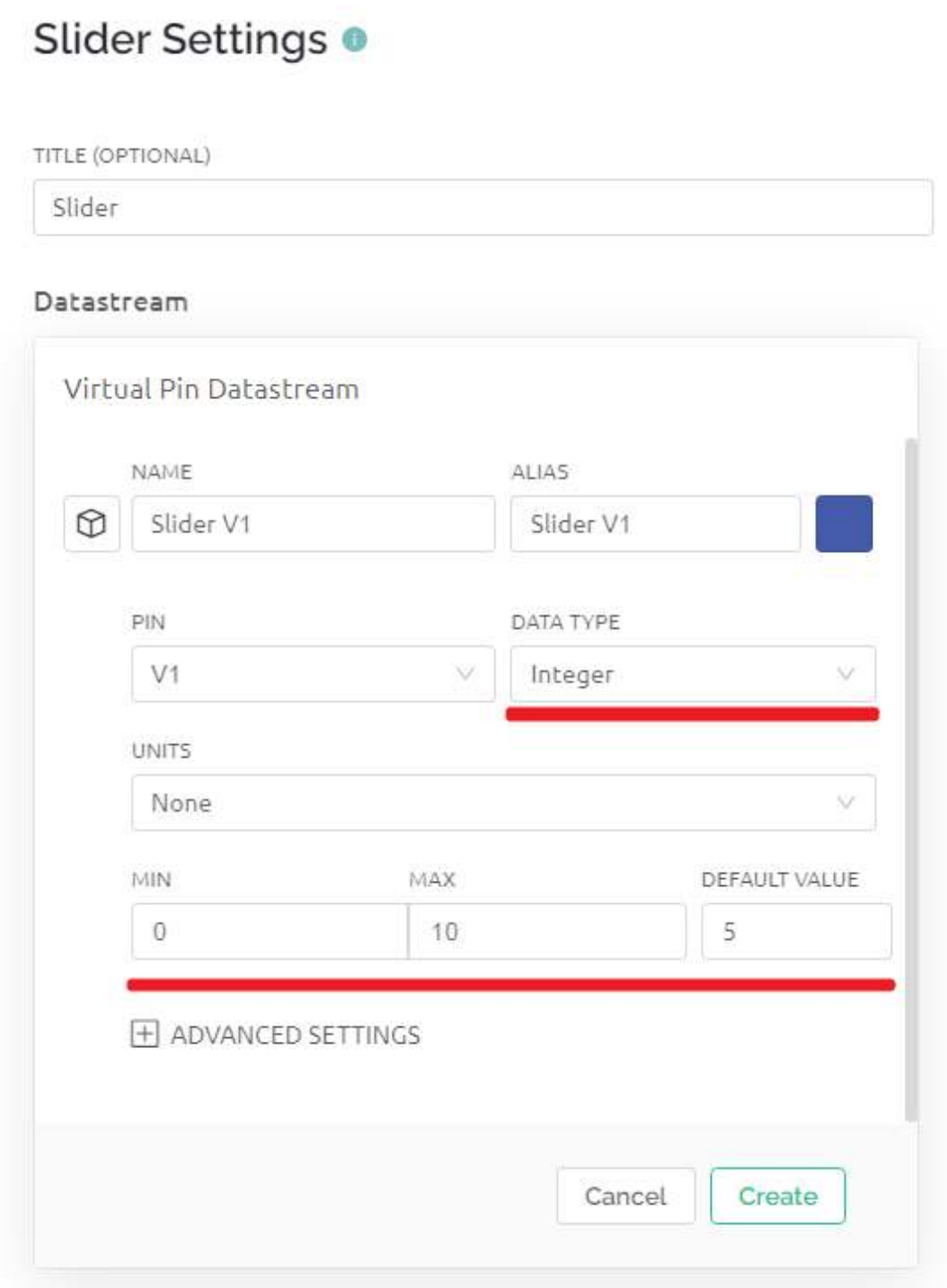
4. Create Datastream, select Virtual Pin



5. Complete the datastream setup. Here is the datastream created for Switch, so **DATA TYPE** select `Interger`, **MIN** and **MAX** set to `0` and `1`. Create and then Save.



6. Use the same steps to create a Datastream for the slider widget, and again, modify **DATA TYPE**, **MIN** and **MAX** to what you need.



7. When finished, click Save And Apply at the top right.

3. Install the Blynk library

Run the following command to install.

```
cd /home/pi
git clone https://github.com/vshymanskyy/blynk-library-python.git
cd blynk-library-python
sudo python3 setup.py
```

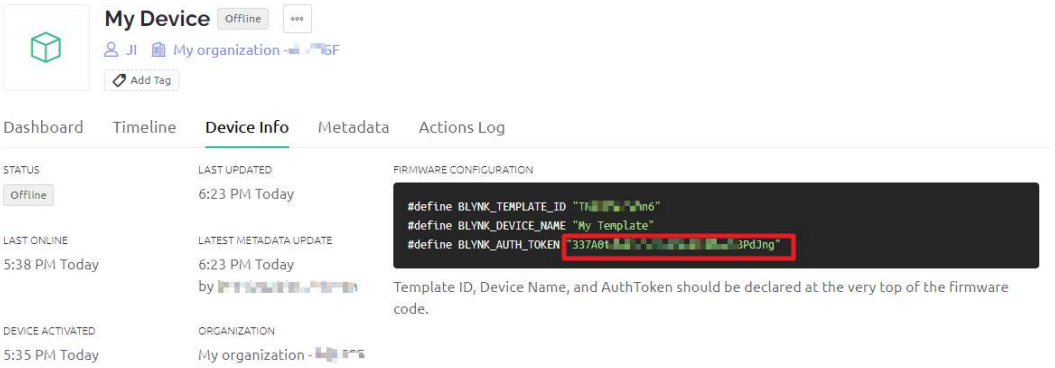
4. Download the Example

We have provided some examples, please run the following command to download them.

```
cd /home/pi
git clone https://github.com/sunfounder/blynk-raspberrypi-python.git
```

5. Run the Code

1. Go to Blynk's Device Info page, you will see some information under **FIRMWARE CONFIGURATION**, you need to copy **BLYNK_AUTH_TOKEN** down.



2. Edit the code.

```
cd /home/pi/blynk-raspberrypi-python
sudo nano blynk_start.py
```

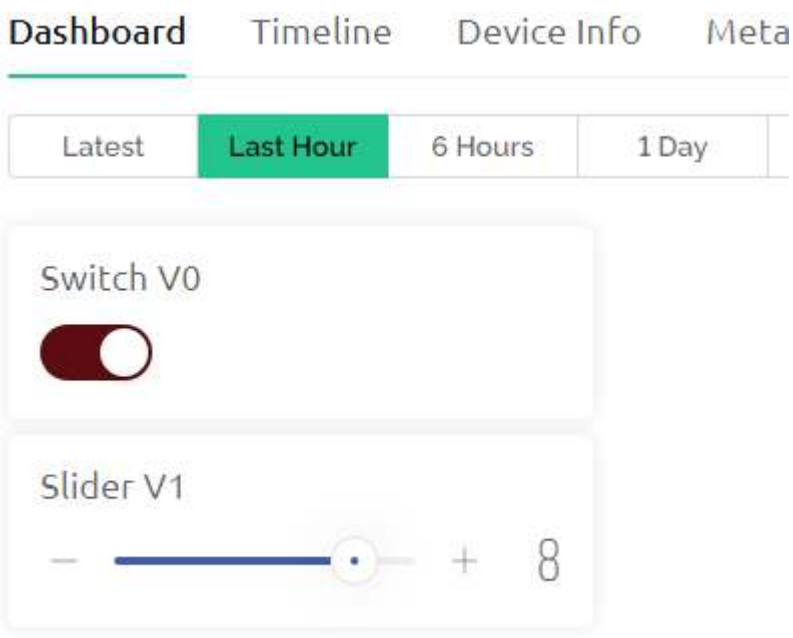
3. Find the line below and past your `BLYNK_AUTH_TOKEN`.

```
BLYNK_AUTH = 'YourAuthToken'
```

4. Run the code.

```
sudo python3 blynk_start.py
```

5. Go to Blynk, and operate the widget on Dashboard.



6. Now you will be able to see your actions on the terminal.

```
..
   ___  __          __
  / _ )/ /_ _____  / /__
 / _  / / // / _ \/  '_/
/____/_/\_, /_//_/_/\_\
       /___/ for Python v1.0.0 (linux)

Connecting to blynk.cloud:443...
Blynk ready. Ping: 142 ms
V0 value: ['1']
V0 value: ['0']
V1 value: ['3']
V1 value: ['8']
V0 value: ['1']
```

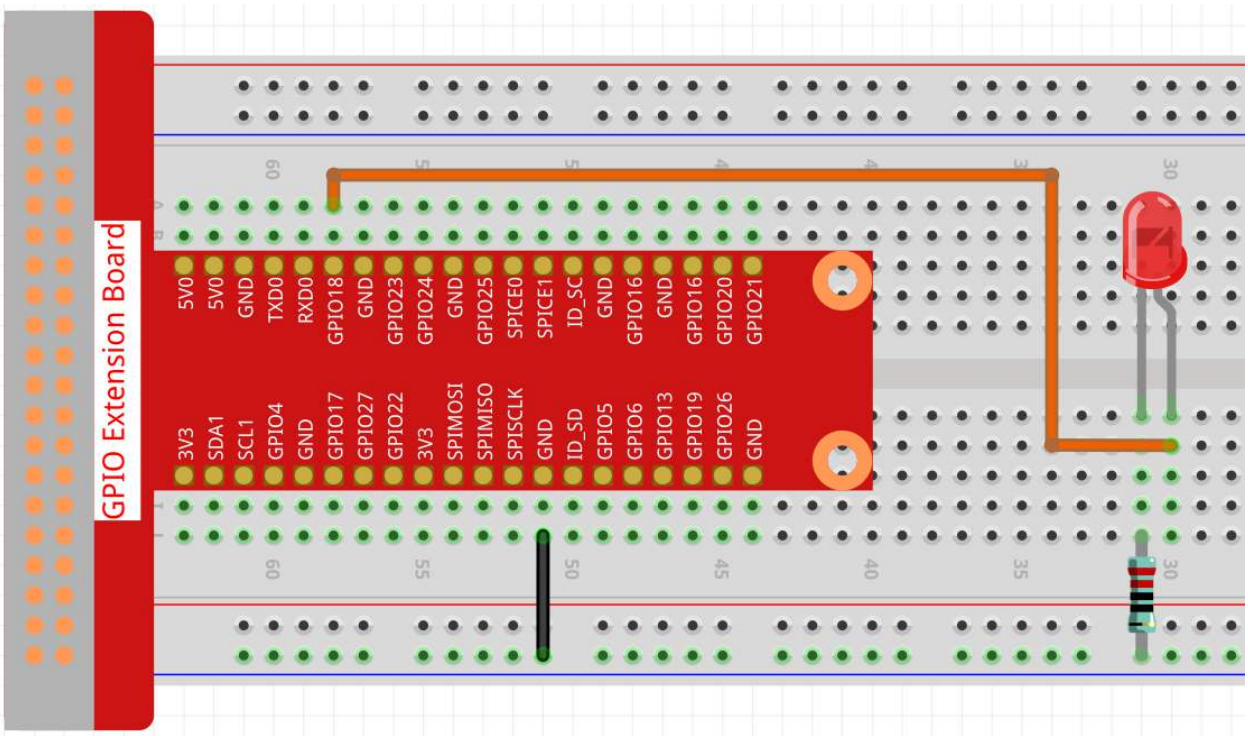
6.5.2 Smart Light

In this project, we use Blynk's Silder to control the brightness of the LED, turning it on and off with Switch.

Note: Before starting this project, we recommend that you complete *Get Start with Blynk*. The following will give you a clear understanding of Blynk.

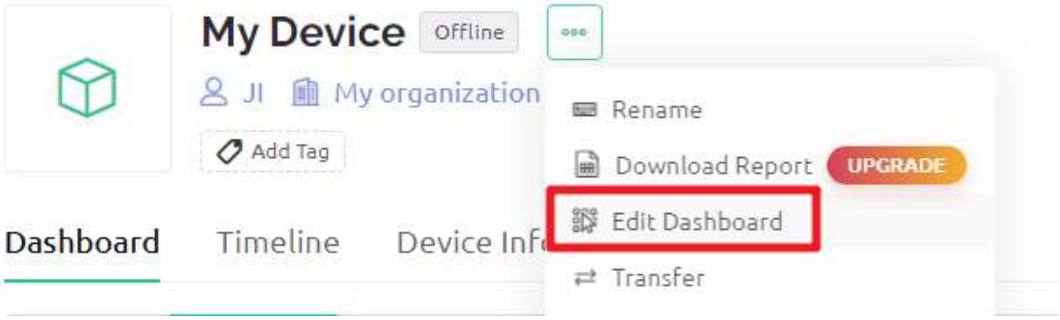
---

1. Wiring

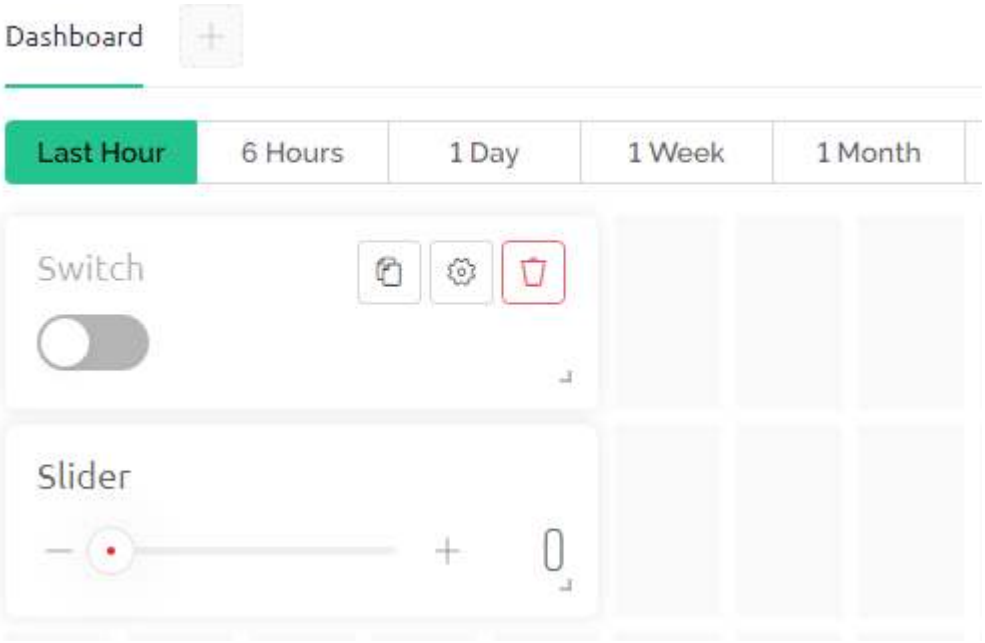


2. Create Widget and Datastream

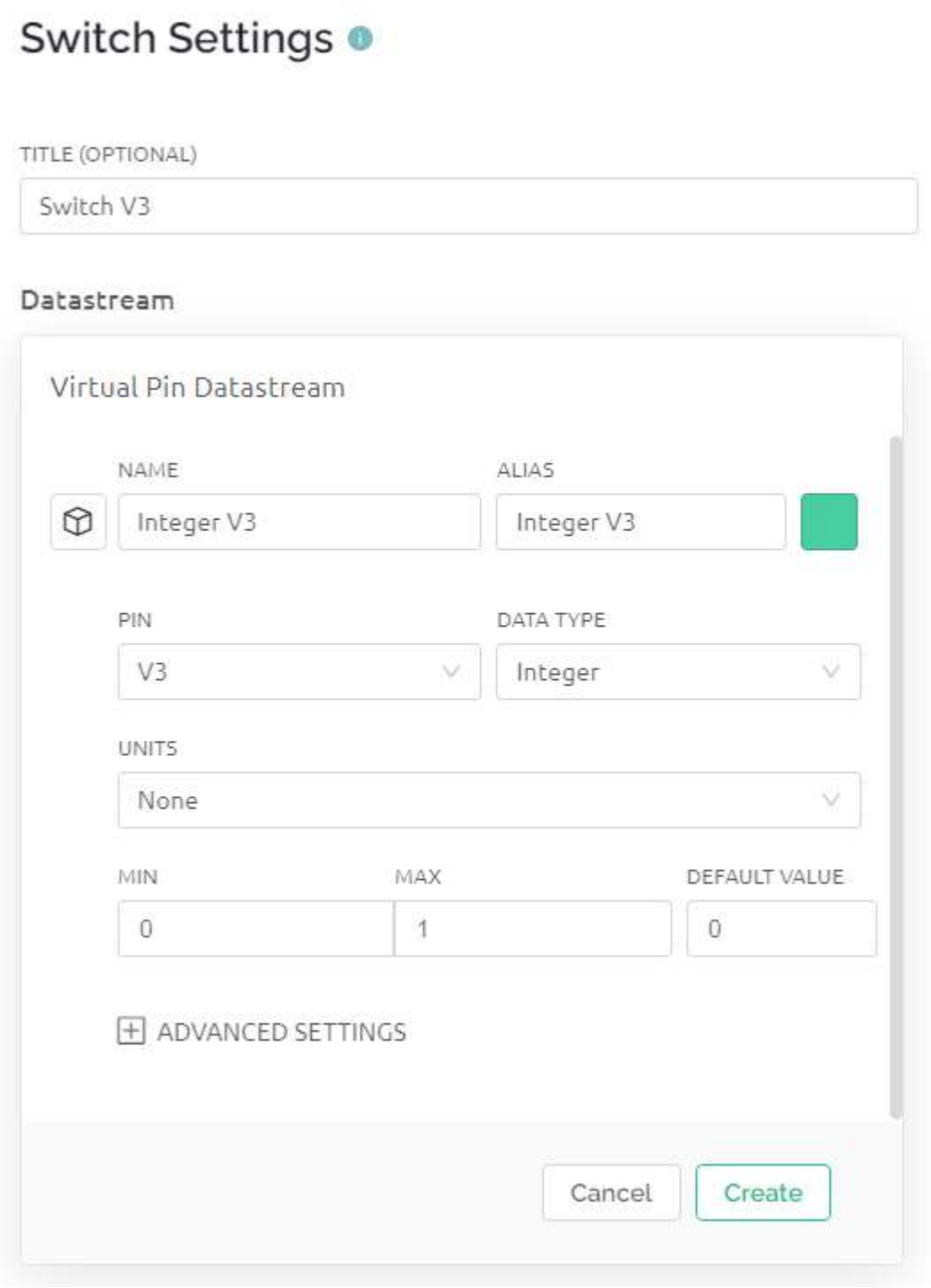
1. Click on the menu icon in the upper right corner and select edit dashboard.



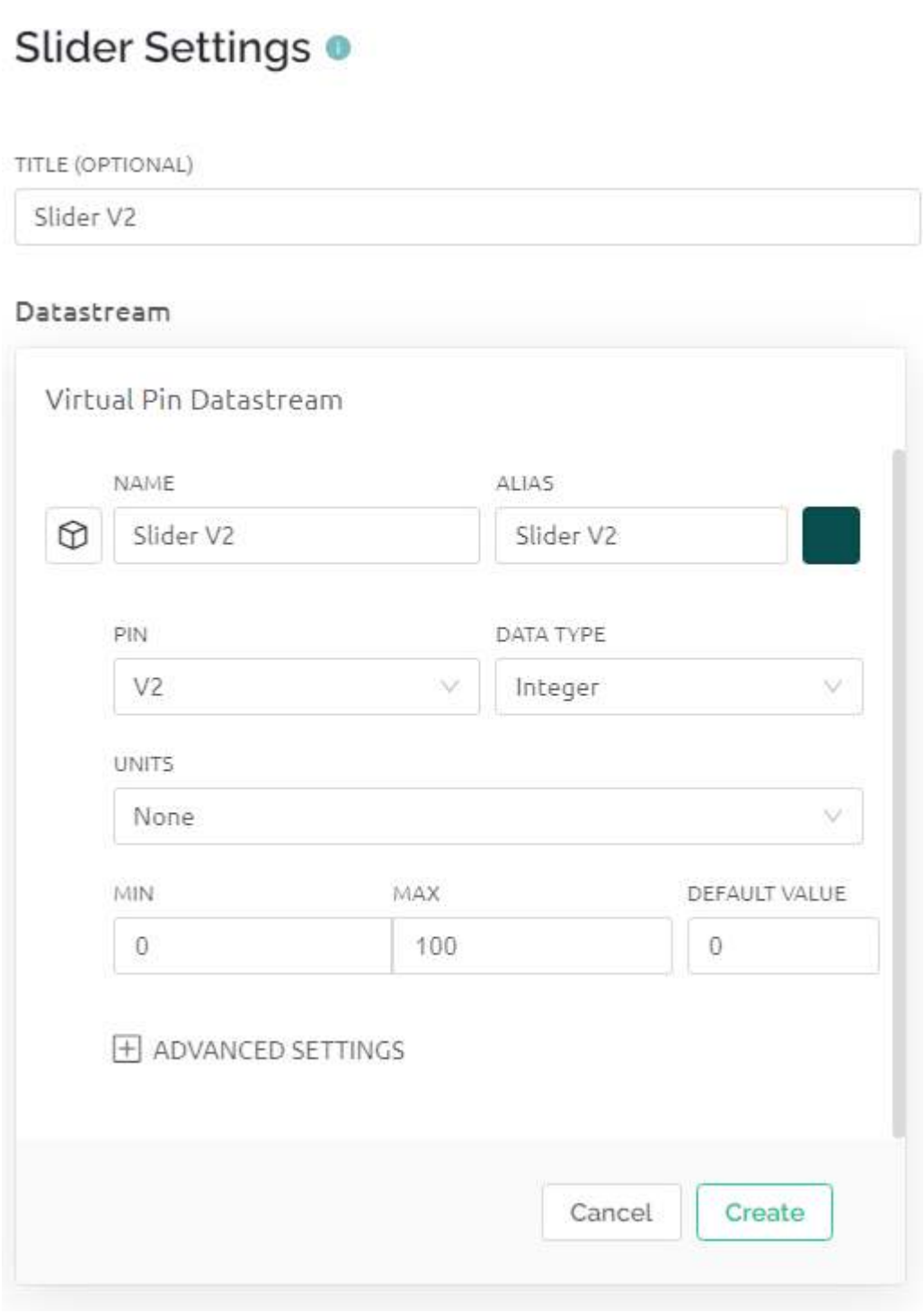
2. Add a Switch widget and a Slider widget to the Dashboard.



3. Create a Datastream for the Switch widget (I used V3). It will be used to control the turning on and off of the LED.



4. Create a Datastream for the Slider widget (I used V2), its value range is 0 to 100, it will be used to control the brightness of the LED.



5. When finished, click Save And Apply at the top right.

3. Run the Code

1. Edit the code

```
cd /home/pi/blynk-raspberrypi-python
sudo nano blynk_light.py
```

2. Find the line below and past your `BLYNK_AUTH_TOKEN`.

```
BLYNK_AUTH = 'YourAuthToken'
```

3. Run the code.

```
sudo python3 blynk_light.py
```

4. Go to Blynk, operate widget on Dashboard. now you click switch widget will turn on/off LED. slide Slider widget will change LED brightness.
5. If you want to use Blynk on mobile devices, please refer to *How to use Blynk on mobile device?*.

6.5.3 Door Window Sensor

When you're outside, you've probably had this confusion. "Are the doors and windows of my house closed?"

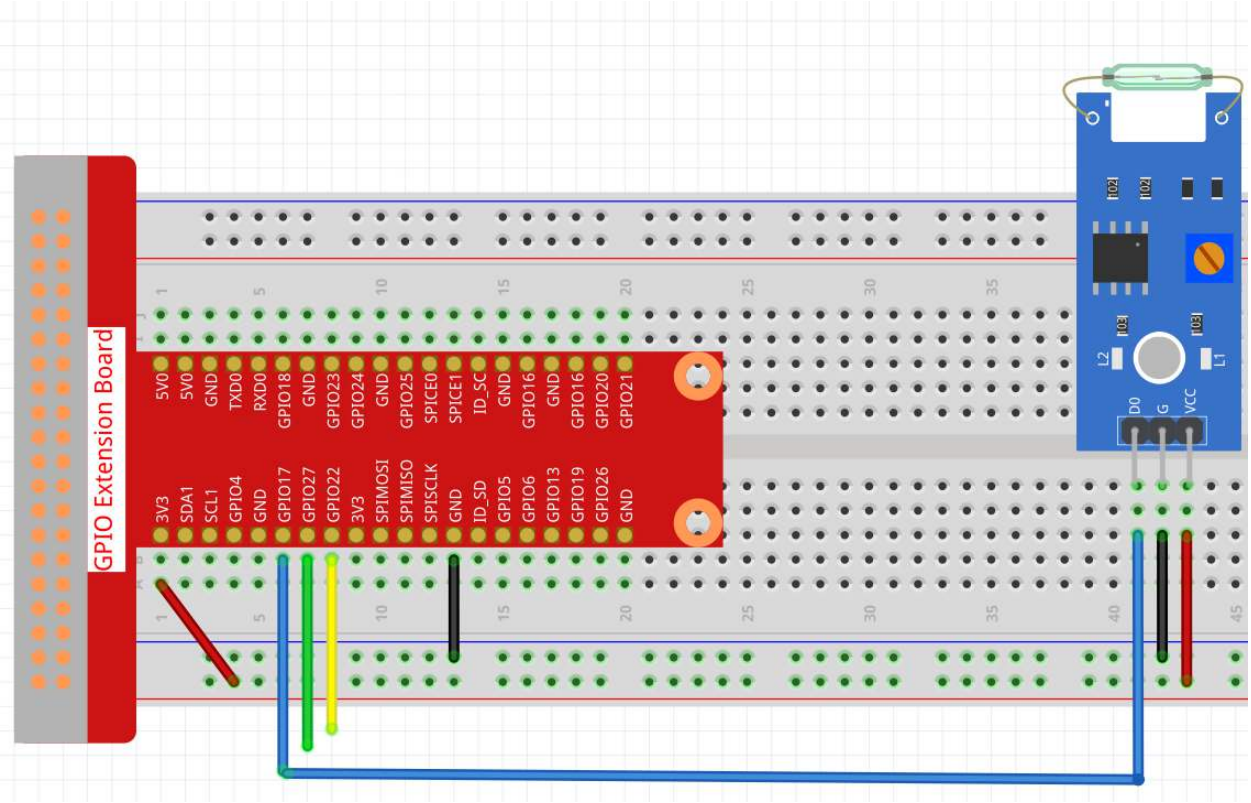
To solve this problem, in this project, we will build a door and window sensor with Reed Switch and magnets.

Install this sensor and magnet on both sides of the door or window. You will be able to check whether your doors and windows are closed or not from the Blynk APP on your phone.

Note: Before starting this project, we recommend that you complete *Get Start with Blynk*. The following will give you a clear understanding of Blynk.

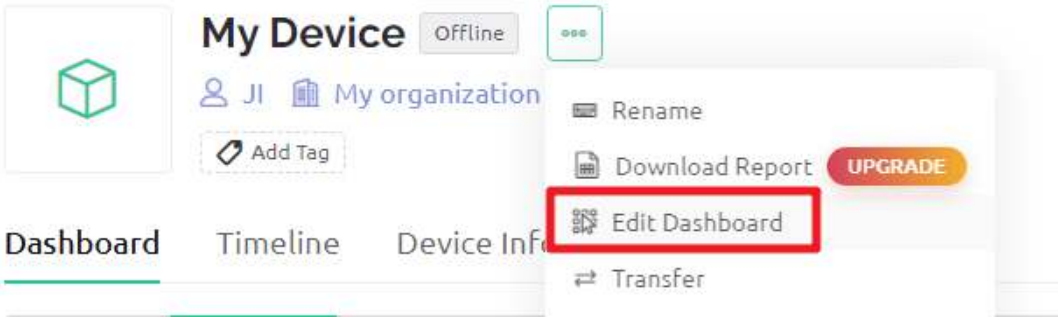
---

1. Wiring

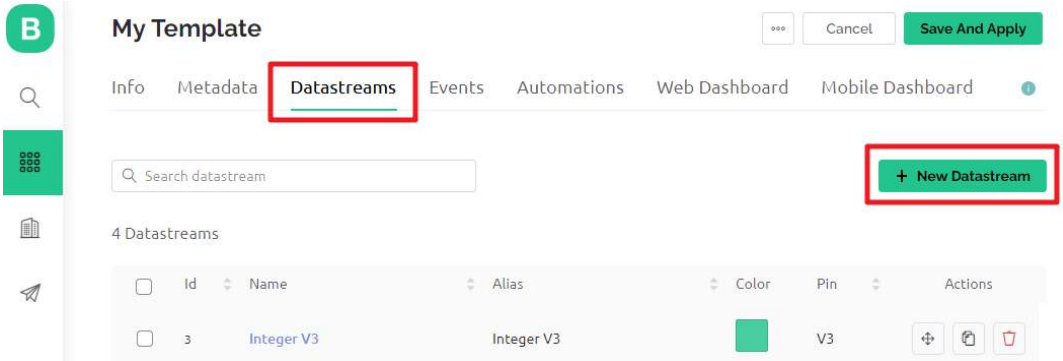


2. Create Datastream

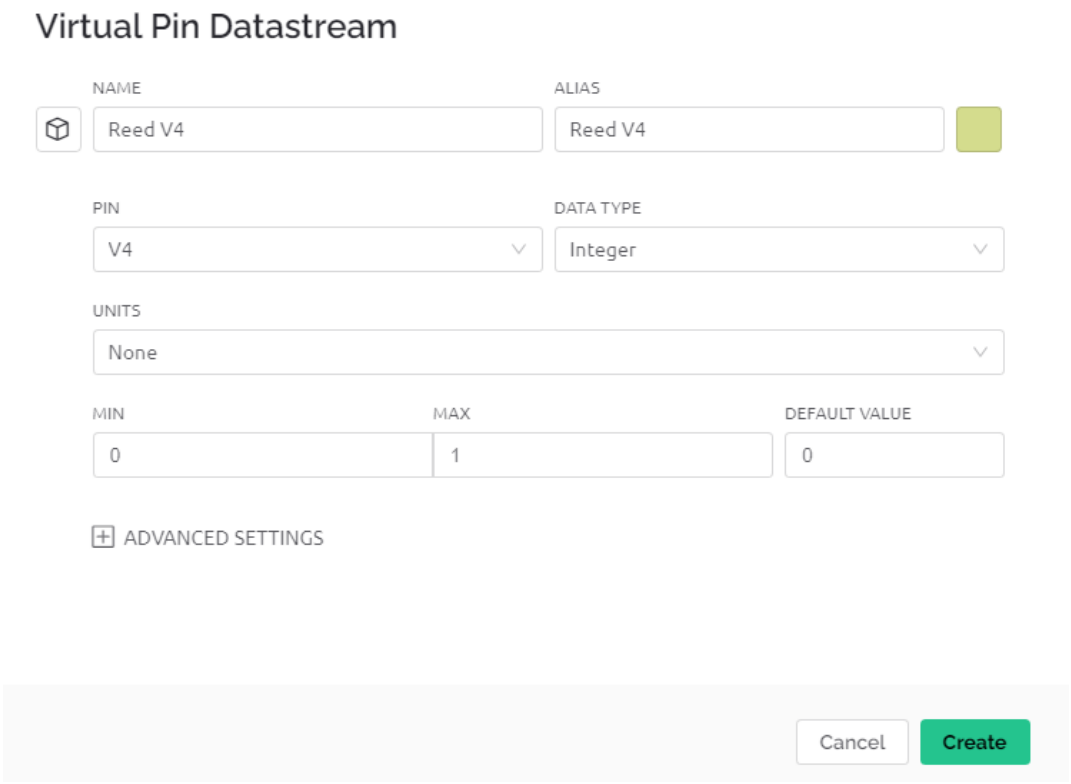
1. Click on the menu icon in the upper right corner and select edit dashboard.



2. Go to the Datastreams page and create a New Datastream.



3. Create a Virtual Pin V4.



4. When finished, click Save And Apply at the top right.

3. Run the Code

1. Edit the code

```
cd /home/pi/blynk-raspberrypi-python
sudo nano blynk_reed.py
```

2. Find the line below and past your `BLYNK_AUTH_TOKEN`.

```
BLYNK_AUTH = 'YourAuthToken'
```

3. Run the code.

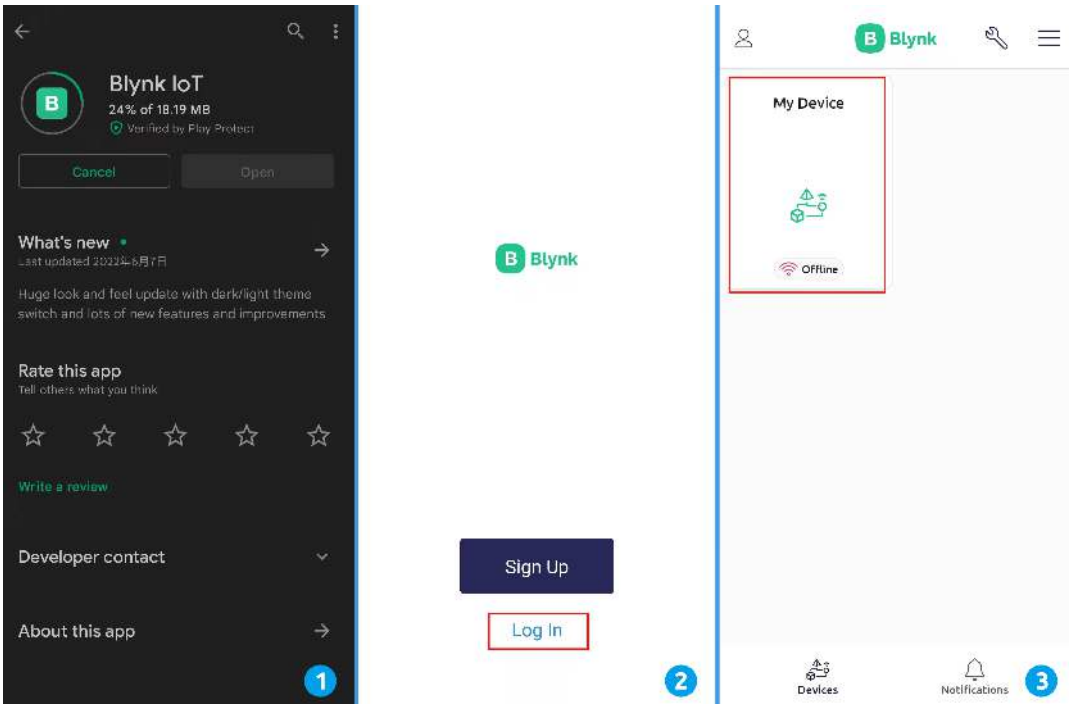
```
sudo python3 blynk_reed.py
```

4. Open Blynk APP

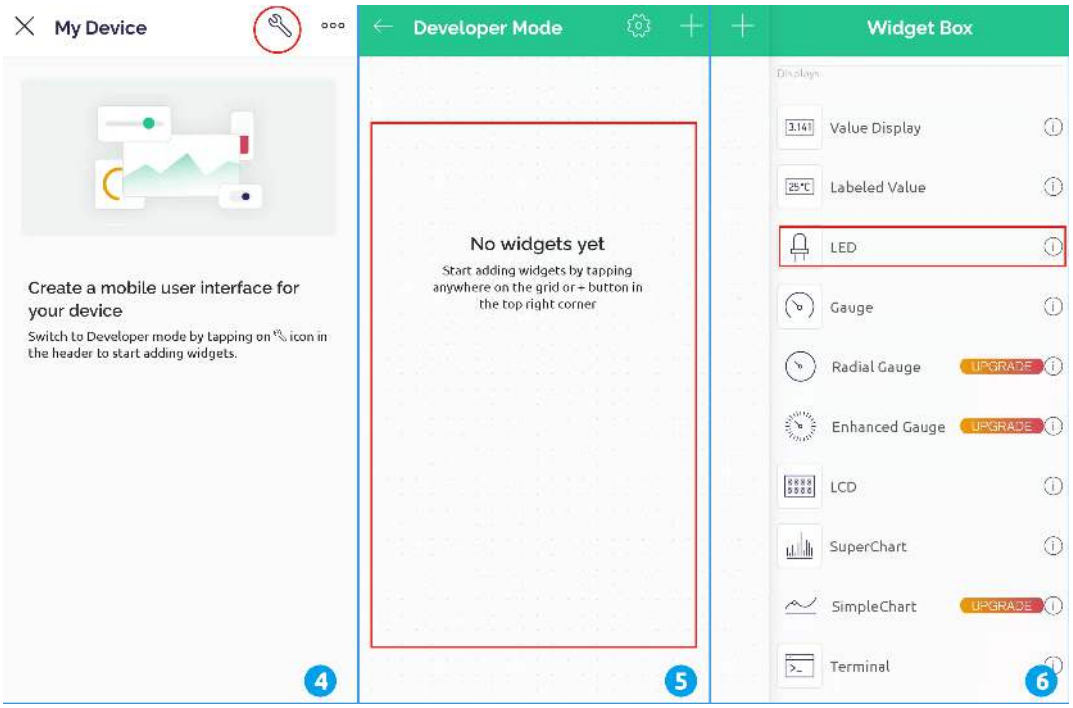
Note: As datastreams can only be created in Blynk on the web, you will need to reference different projects to create datastreams on the web, then follow the tutorial below to create widgets in Blynk on your mobile device.

---

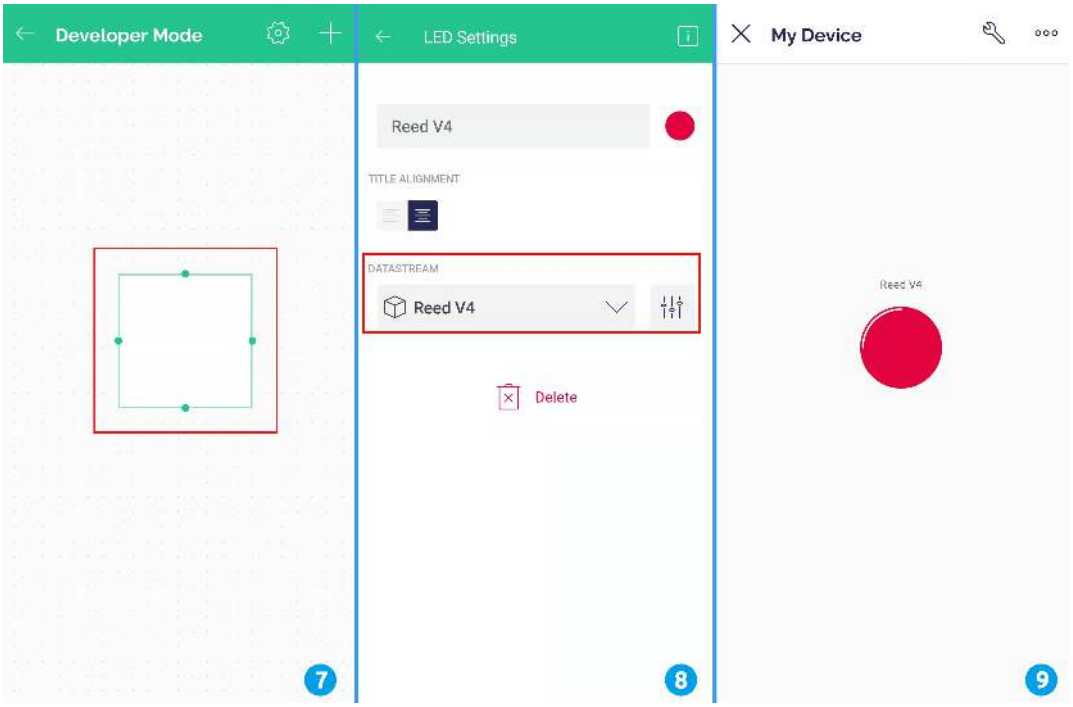
1. Open Google Play or APP Store on your mobile device and search for “Blynk IoT” (not Blynk(legacy)) to download.
2. After opening the APP, login in, this account should be the same as the account used on the web client.
3. Then go to **Dashboard** (if you don’t have one, create one) and you will see that the **Dashboard** for mobile and web are independent of each other.



4. Click **Edit** Icon.
5. Click on the blank area.
6. Choose **LED** widget.



7. Now you will see a **LED** widget appear in the blank area, even if it looks like a blank grid, click on it.
8. **LED** Settings will appear, select the **V4** datastreams you just set in the web page. Note that each widget corresponds to a different datastream in each project.
9. Go back to the **Dashboard** page. Now if you see that the **LED** widget is filled with color, your door or window is open; if the **LED** widget is not filled with color, the door or window is closed.

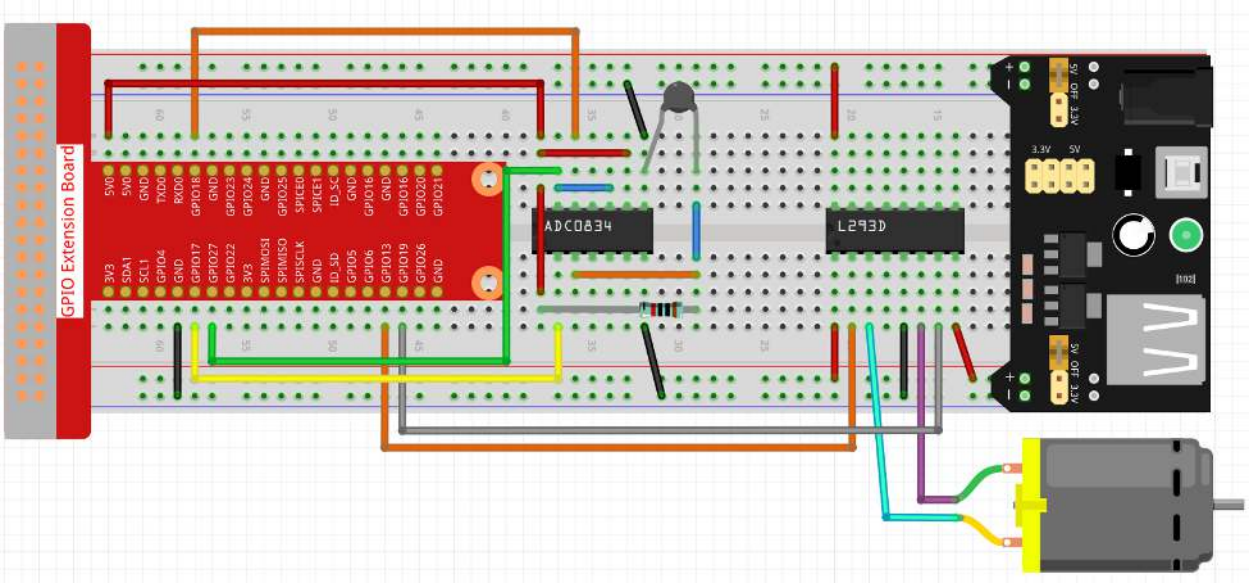


6.5.4 Smart Fan

In this project, you can see the temperature from Blynk and turn on the fan remotely.

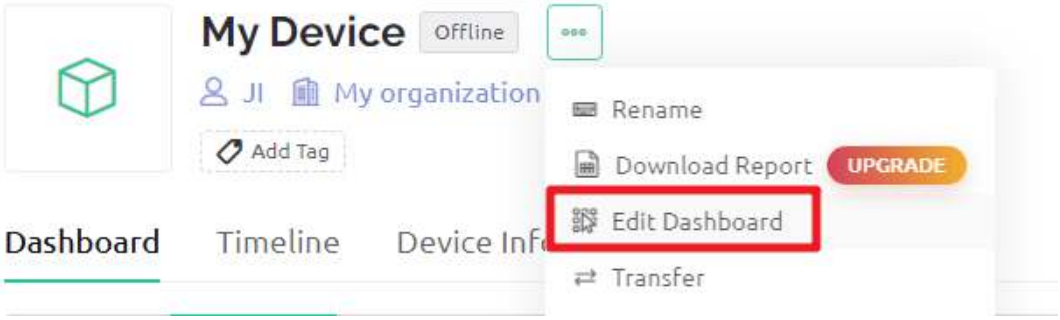
Note: Before starting this project, we recommend that you complete *Get Start with Blynk*. The following will give you a clear understanding of Blynk.

1. Wiring

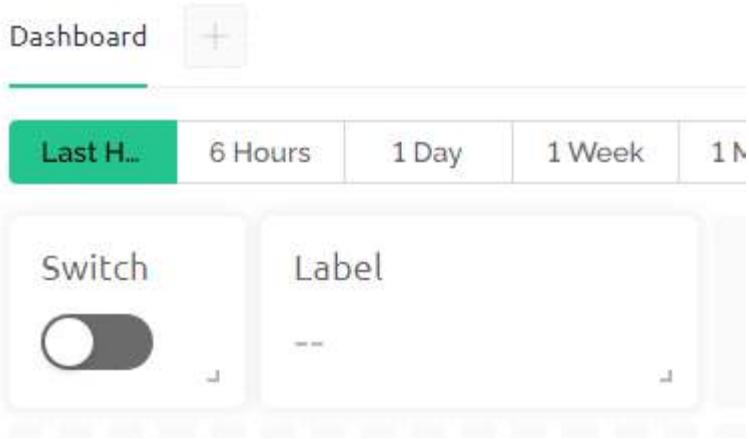


2. Create Widget and Datastream

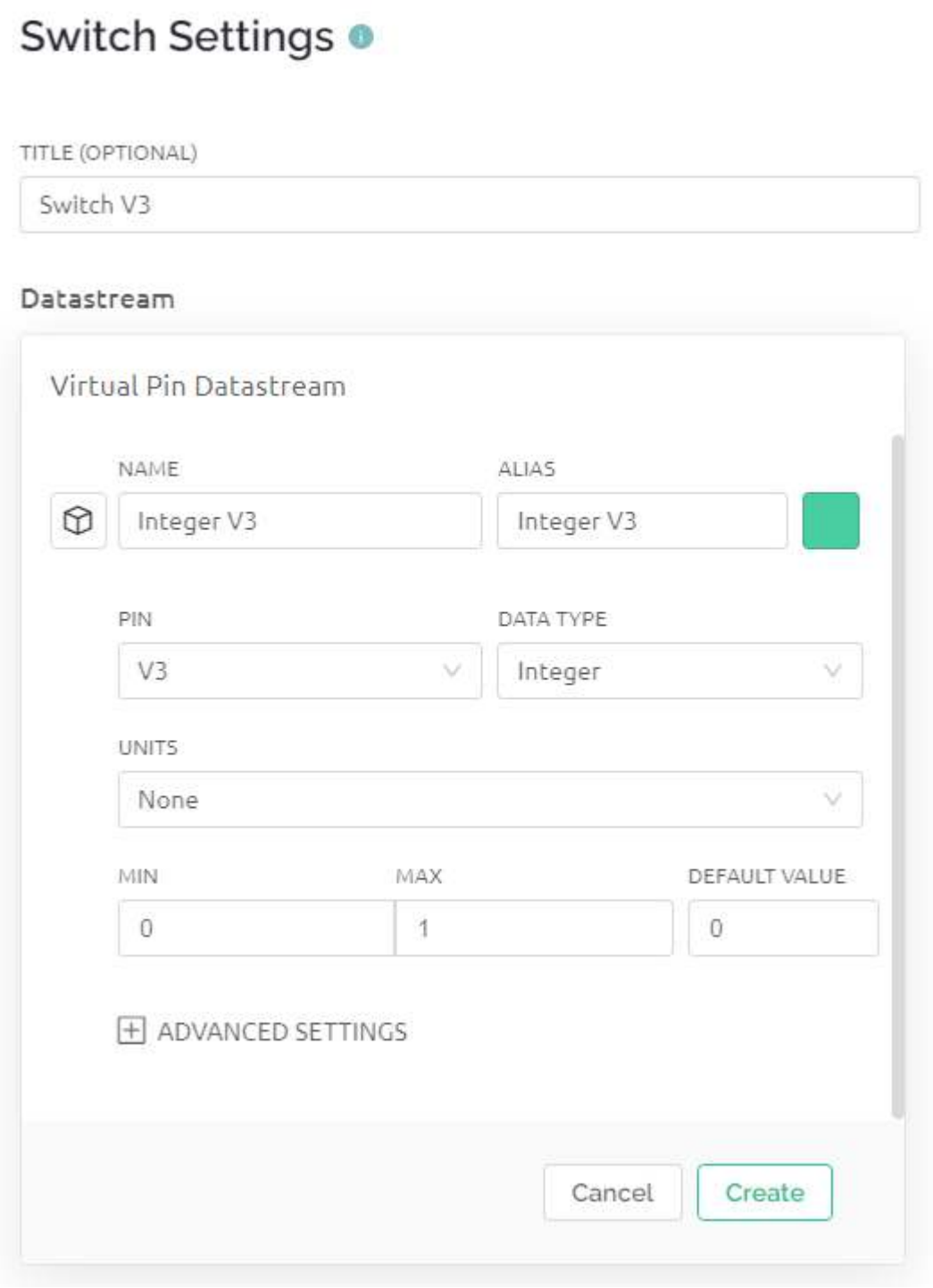
1. Click on the menu icon in the upper right corner and select edit dashboard.



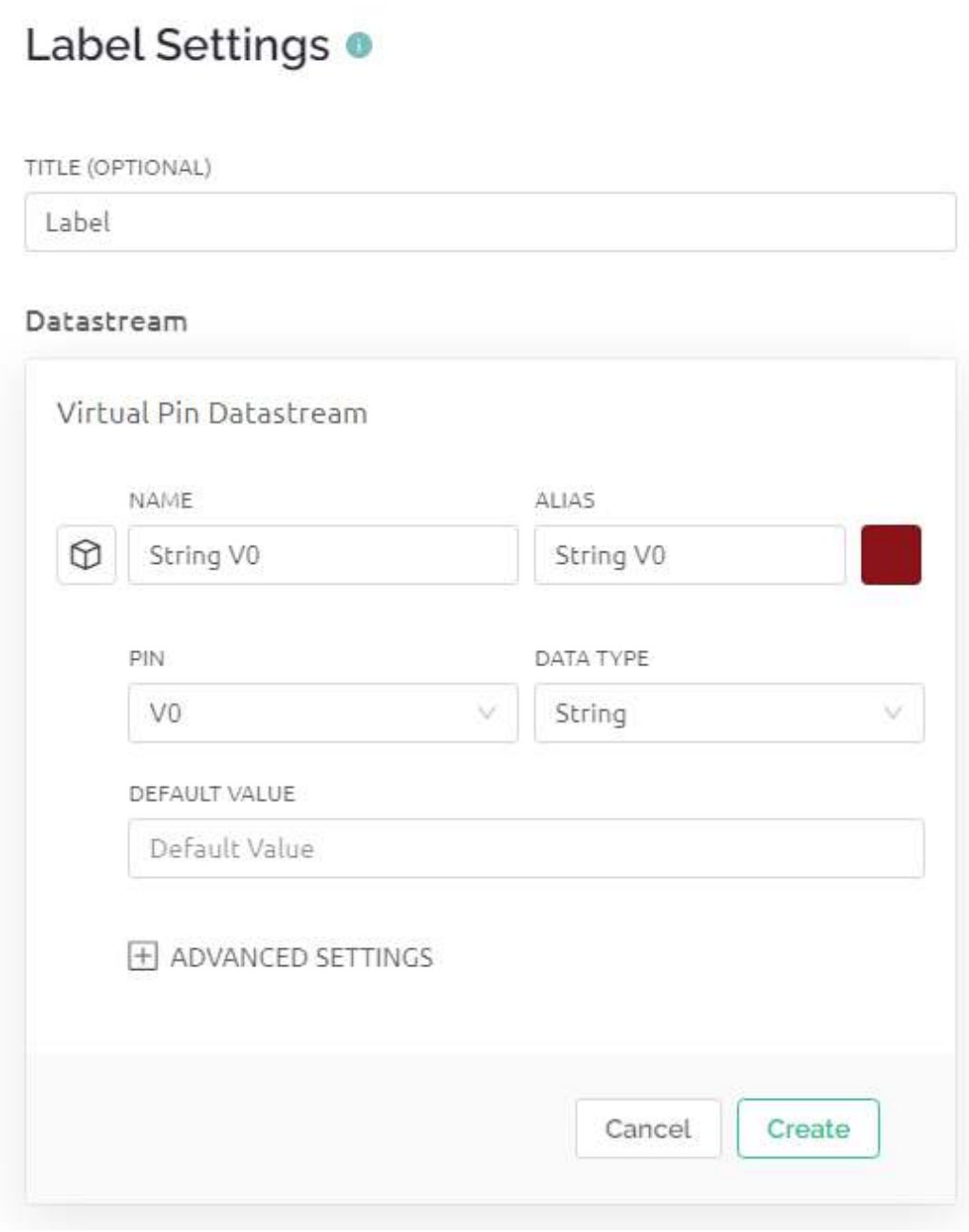
2. Add a Switch widget and a Label widget to the Dashboard.



3. Create a Datastream (I used V3) for the Switch widget. It will be used to turn on the Motor.



4. Create a Datastream for the Label widget(I used V0). It will be used to display the temperature. Set **DATA TYPE** to String.



5. When finished, click Save And Apply at the top right.

3. Run the Code

1. Edit the code

```
cd /home/pi/blynk-raspberrypi-python
sudo nano blynk_motor.py
```

2. Find the line below and past your `BLYNK_AUTH_TOKEN`.

```
BLYNK_AUTH = 'YourAuthToken'
```

3. Run the code.

```
sudo python3 blynk_motor.py
```

4. Go to Blynk, on the Dashboard you can check the temperature via Label widget; you can turn on/off the fan via Switch widget.
5. If you want to use Blynk on mobile devices, please refer to *How to use Blynk on mobile device?*.

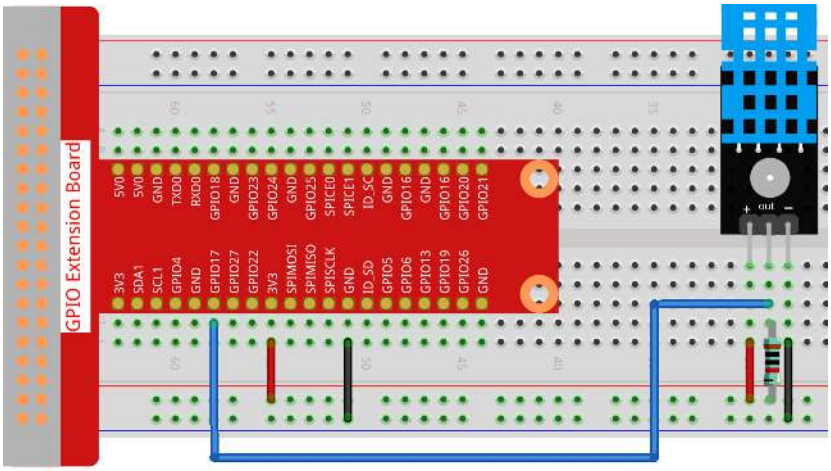
6.5.5 Temperature Recorder

In this project, you can see the current temperature and the temperature change line graph from Blynk.

Note: Before starting this project, we recommend that you complete *Get Start with Blynk*. The following will give you a clear understanding of Blynk.

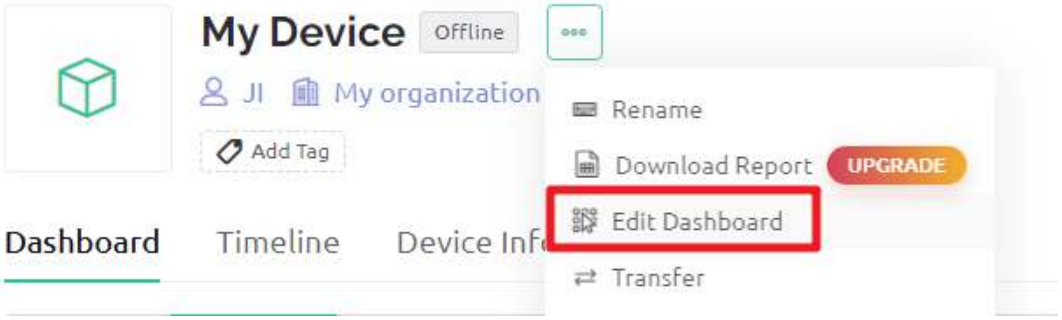
---

1. Wiring

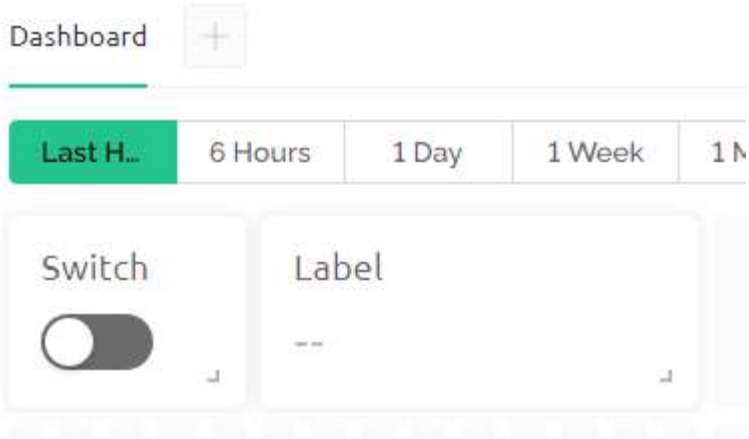


2. Create Widget and Datastream

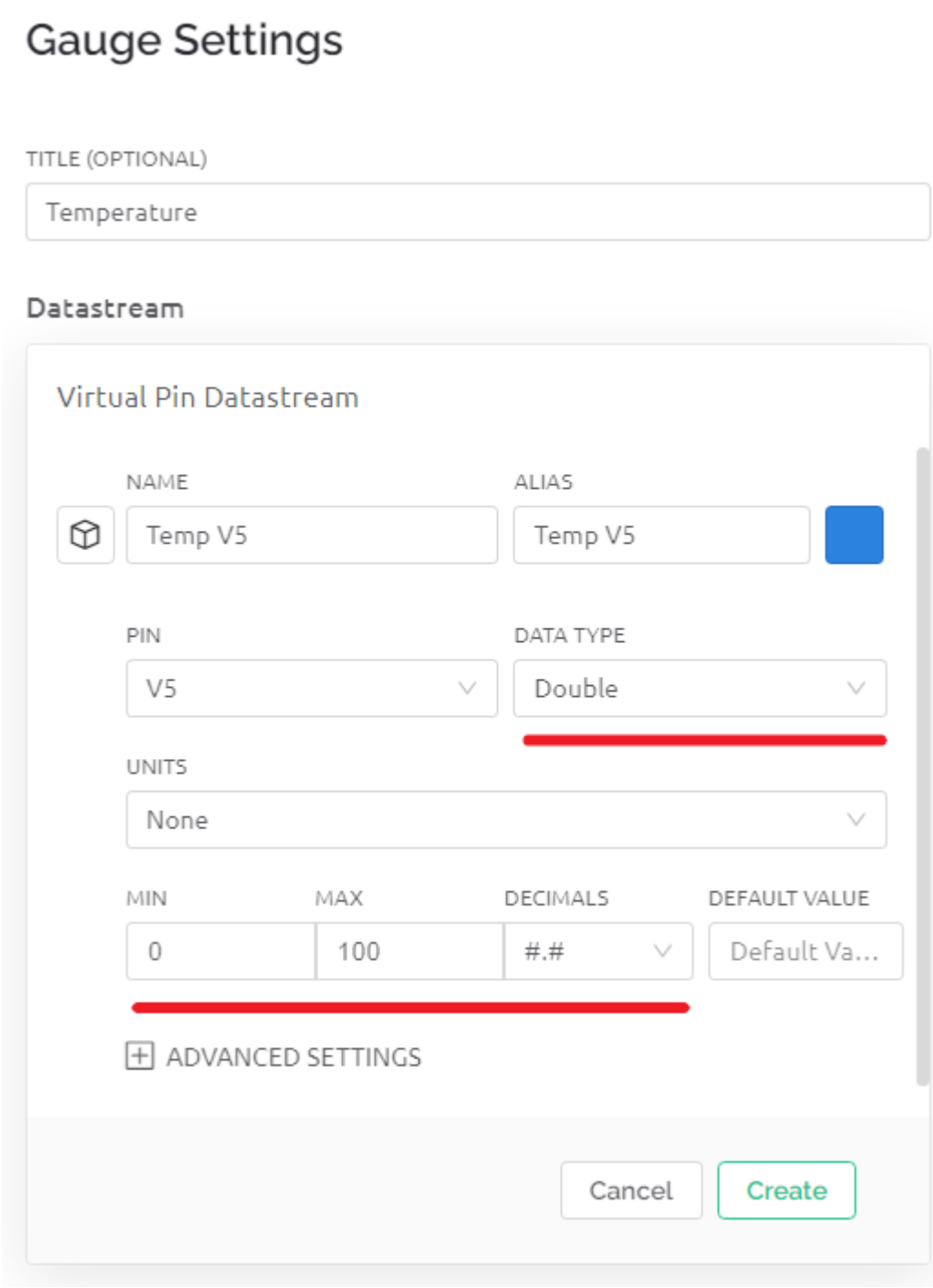
1. Click on the menu icon in the upper right corner and select edit dashboard.



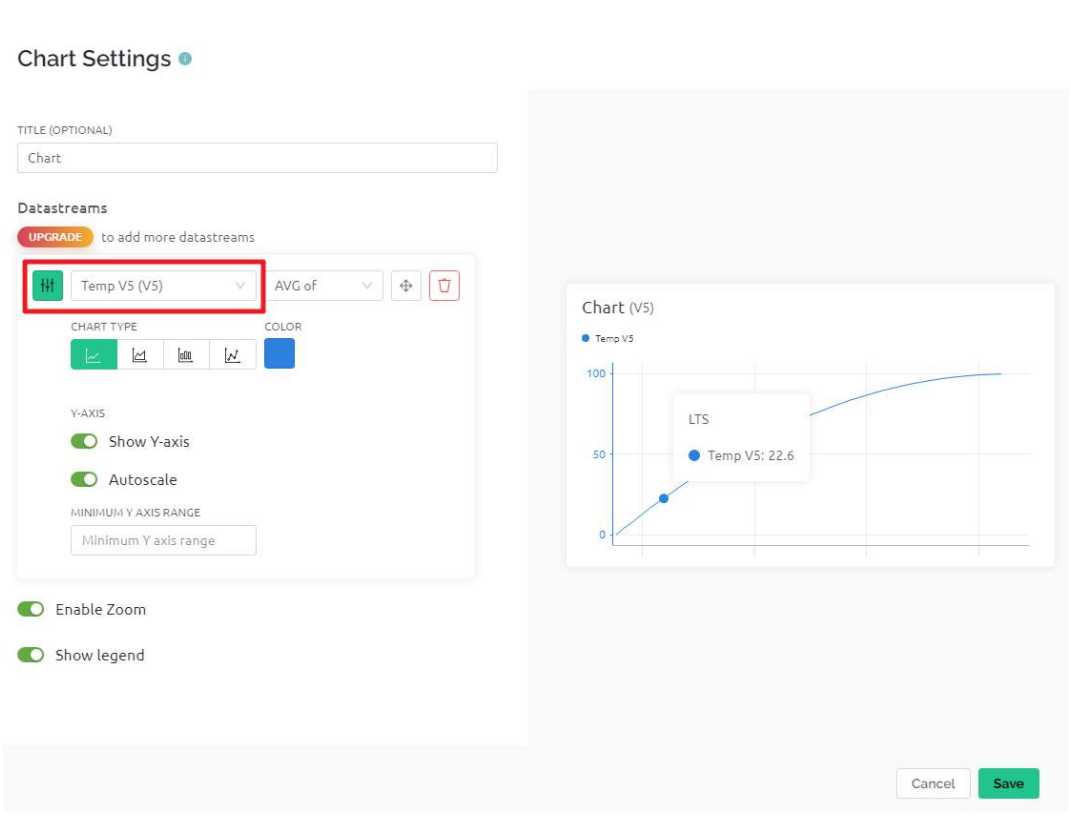
2. Add a Gauge widget and a Chart widget to the Dashboard.



3. Create a Datastream for the Gauge widget (I used V5). It will be used to display the temperature. Set **DATA TYPE** to `Double`, **DECIMALS** to `#.##` (two valid decimal places).



4. Add the V5 Datastream you just created to the Chart widget.



5. When finished, click Save And Apply at the top right.

3. Run the Code

1. Edit the code

```
cd /home/pi/blynk-raspberrypi-python
sudo nano blynk_temp.py
```

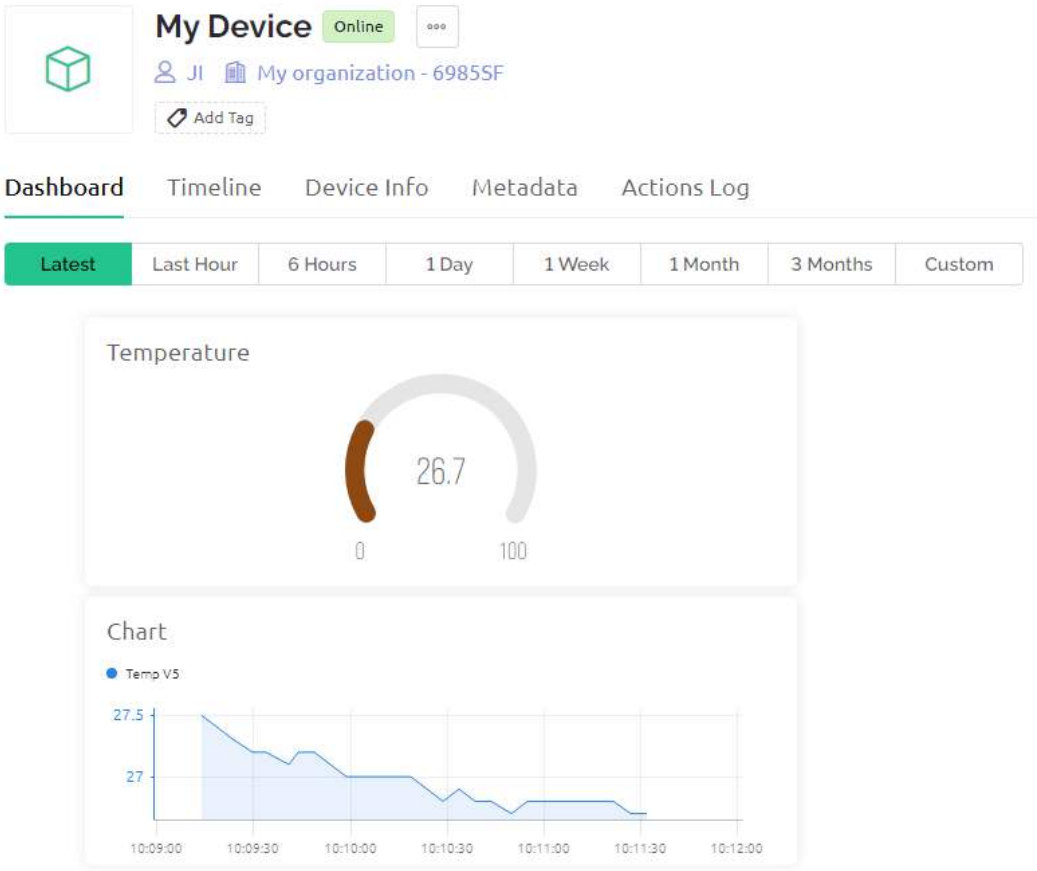
2. Find the line below and past your `BLYNK_AUTH_TOKEN`.

```
BLYNK_AUTH = 'YourAuthToken'
```

3. Run the code.

```
sudo python3 blynk_temp.py
```

4. Go to Blynk. Now you can view the temperature and temperature change line graph on the Dashboard.



5. If you want to use Blynk on mobile devices, please refer to *How to use Blynk on mobile device?*.

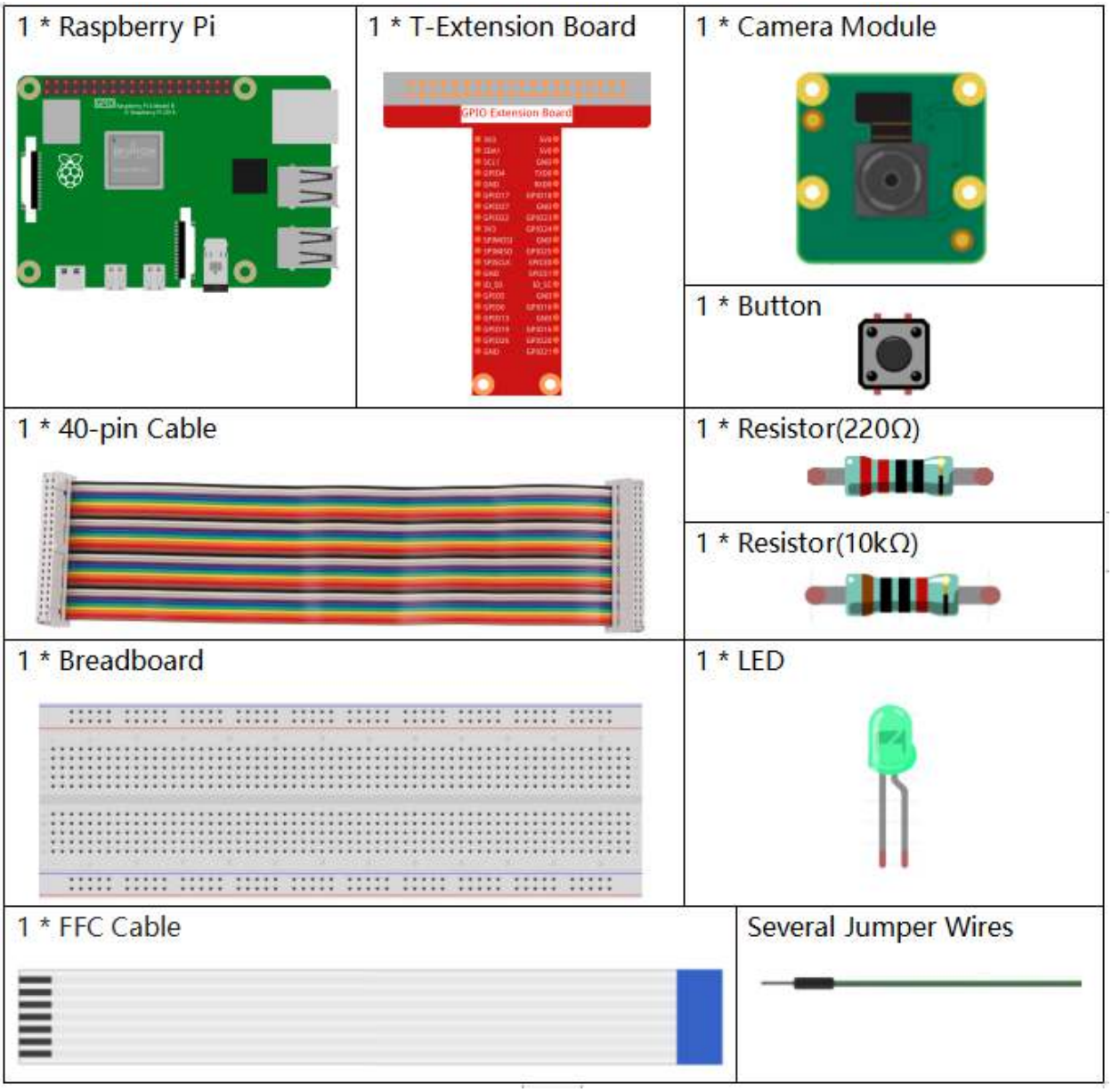
6.6 Extension

6.6.1 4.1.1 Camera

Introduction

Here we will make a camera with a shutter, when you press the button, the camera shoots while the LED flashes.

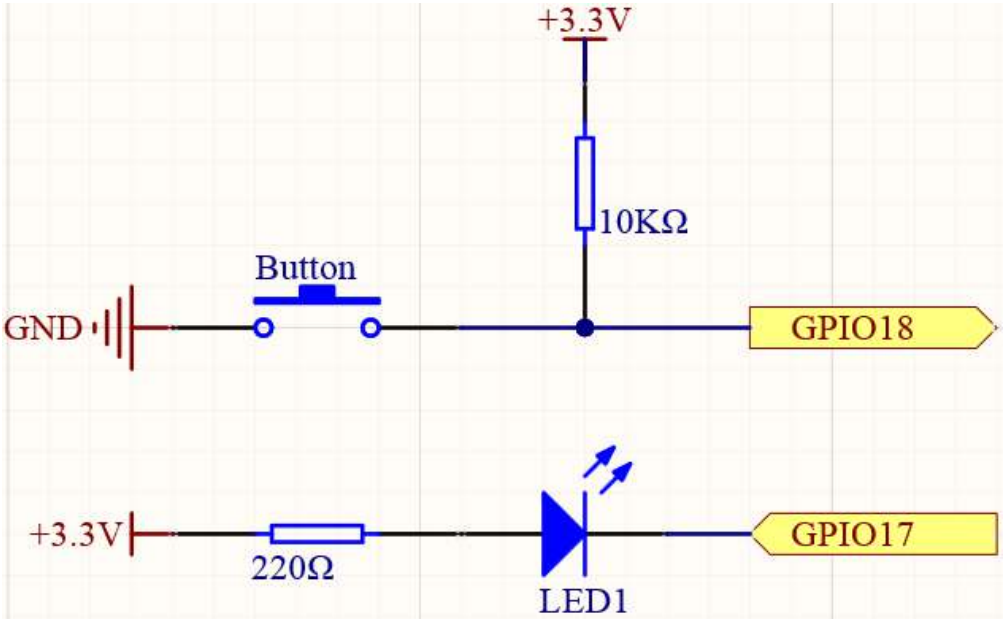
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Button*
- *Camera Module*

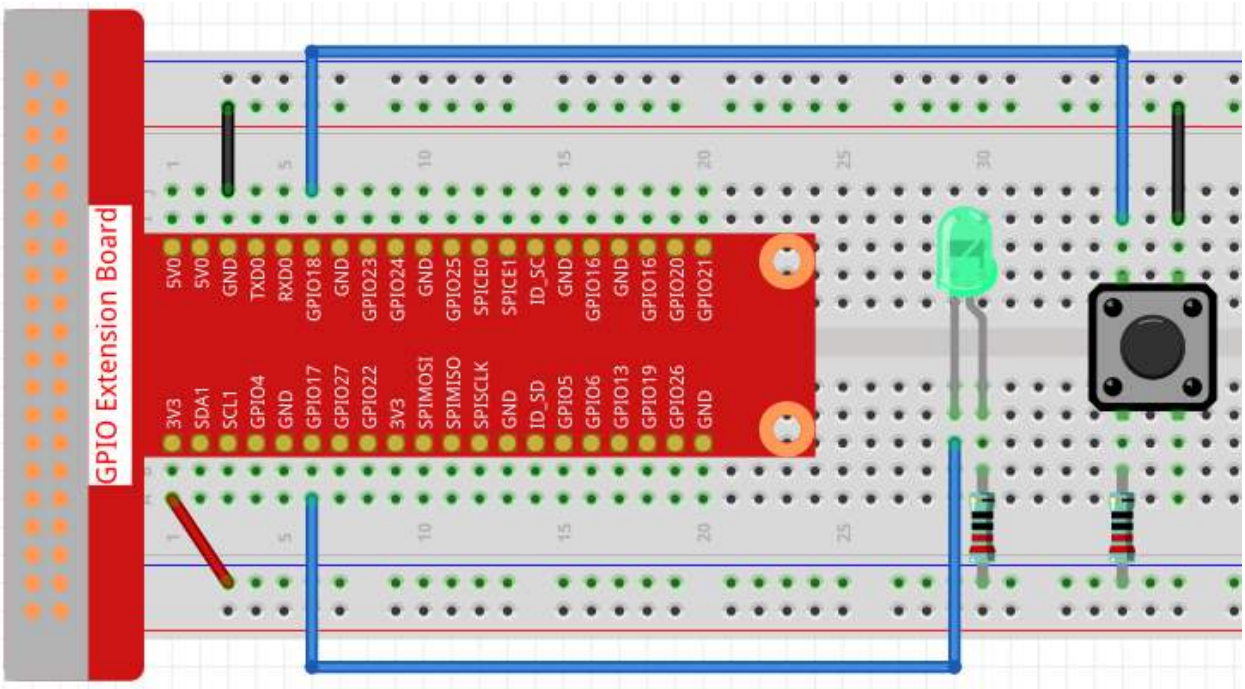
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: To connect the camera module and complete the configuration, please refer to: *Camera Module*.

Step 3: Go into the Raspberry Pi Desktop. You may need a screen for a better experience, refer to: Connect your Raspberry Pi. Or access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

Step 4: Open a Terminal and get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 5: Run.

```
sudo python3 4.1.1_Camera.py
```

After the code runs, press the button, the Raspberry Pi will flash the LED and take a picture. The photo will be named `my_photo.jpg` and stored in the `/home/pi` directory.

Note: You can also open `4.1.1_Camera.py` in the `/home/pi/raphael-kit/python/` path with a Python IDE, click Run button to run, and stop the code with Stop button.

If you want to download the photo to your PC, please refer to *Filezilla Software*.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

from picamera import PiCamera
import RPi.GPIO as GPIO
import time

camera = PiCamera()

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

status = False

def setup():
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
        GPIO.setup(BtnPin, GPIO.IN)
        camera.start_preview(alpha=200)

def takePhotos(pin):
        global status
        status = True

def main():
        global status
        GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=takePhotos)
        while True:
                if status:
                        for i in range(5):
```

(continues on next page)

(continued from previous page)

```
                                GPIO.output(LedPin, GPIO.LOW)
                                time.sleep(0.1)
                                GPIO.output(LedPin, GPIO.HIGH)
                                time.sleep(0.1)
                        camera.capture('/home/pi/my_photo.jpg')
                        print ('Take a photo!')
                        status = False
                else:
                        GPIO.output(LedPin, GPIO.HIGH)
                time.sleep(1)

def destroy():
        camera.stop_preview()
        GPIO.output(LedPin, GPIO.HIGH)
        GPIO.cleanup()

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

Code Explanation

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

Set the event of `BtnPin`, when the button is pressed (the level signal changes from high to low) , call the function `takePhotos()`.

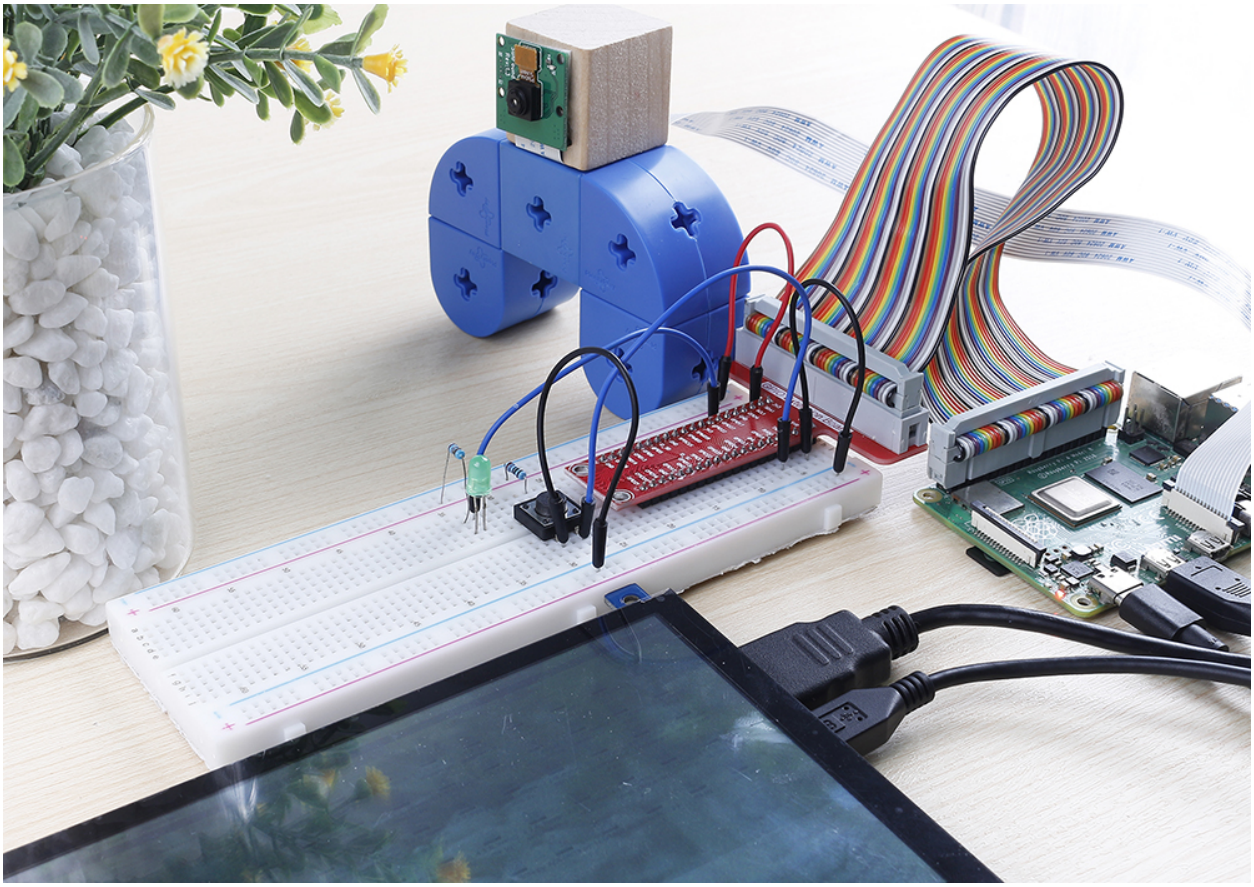
```
def takePhotos(pin):
    global status
    status = True
```

When `takePhotos()` is called, modify the `status` to `True`.

```
if status:
    for i in range(5):
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.1)
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.1)
    camera.capture('/home/pi/my_photo.jpg')
    print ('Take a photo!')
    status = False
else:
    GPIO.output(LedPin, GPIO.HIGH)
time.sleep(1)
```

When `status` is `True`, the Raspberry Pi will flash the LED and take a picture. The photo will be named `my_photo.jpg` and stored in the `/home/pi` directory.

Phenomenon Picture

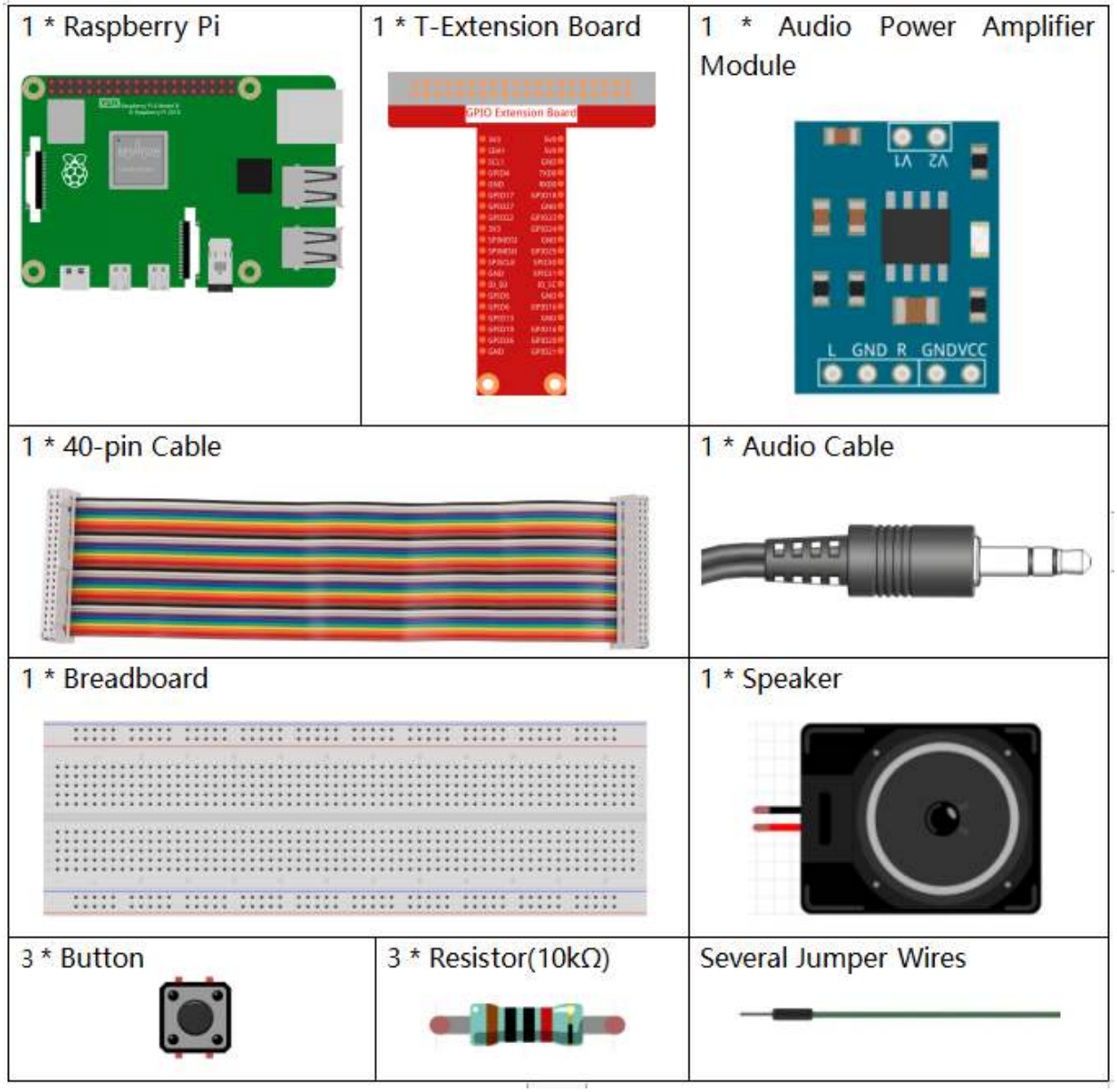


6.6.2 4.1.2 Music Player

Introduction

In project *3.1.3 Audio Module*, let speaker play a song. Now we add 3 buttons to control the play/pause and volume of the music.

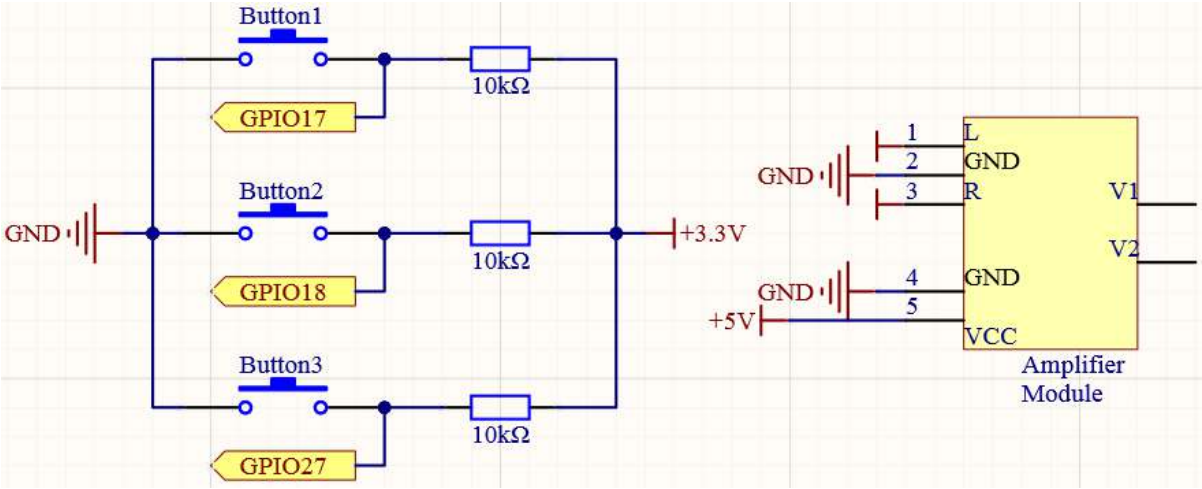
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Audio Module and Speaker*
- *Button*

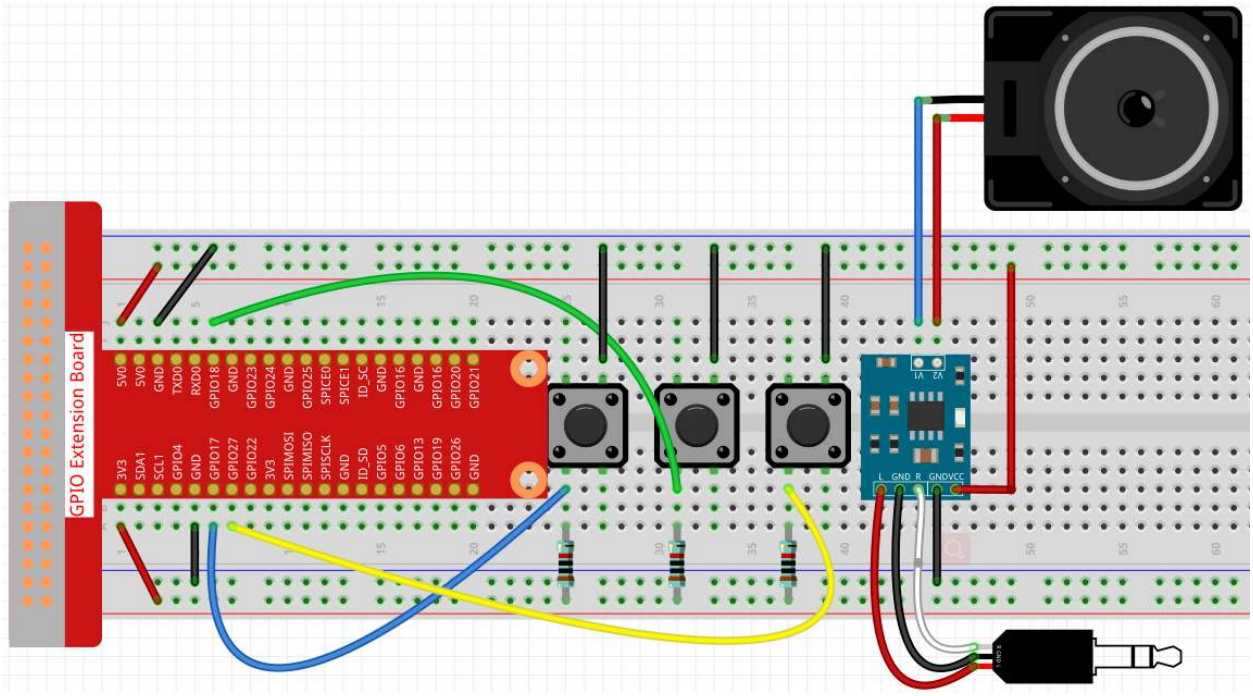
Schematic Diagram

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

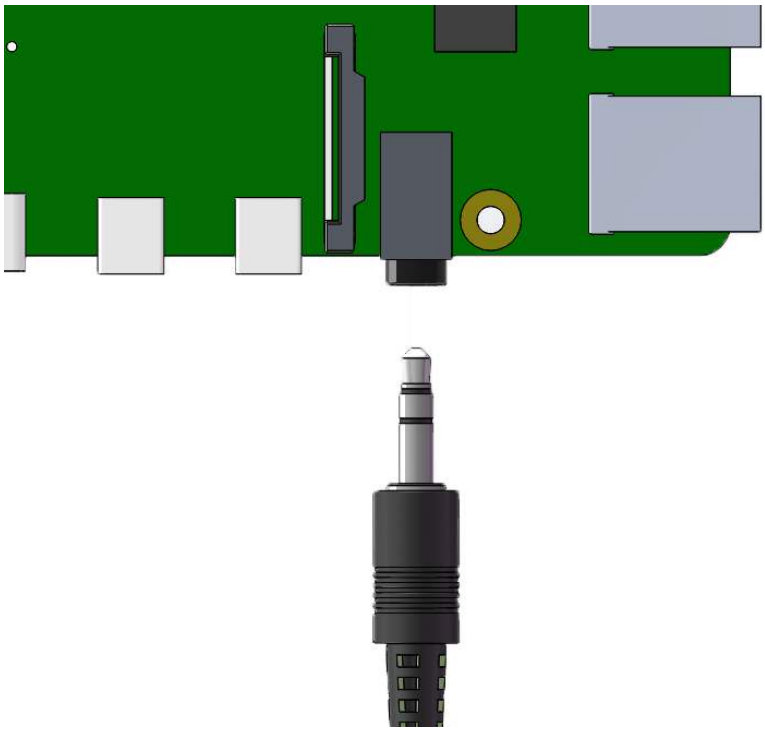


Experimental Procedures

Step 1: Build the circuit.



After building the circuit according to the above diagram, then plug the audio cable into the Raspberry Pi's 3.5mm audio jack.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
python3 4.1.2_MusicPlayer.py
```

After the code runs, Raspberry Pi will play the `my_music.mp3` file in the `/home/pi/raphael-kit/music` directory.

- Button 1 pauses/play the music.
- Button 2 decreases the volume.
- Button 3 increases the volume.

If you want to upload other music files to Raspberry Pi, you can refer to *Filezilla Software*.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
from pygame import mixer
import RPi.GPIO as GPIO
import time

BtnPin1 = 18
BtnPin2 = 17
BtnPin3 = 27
volume = 0.7
```

(continues on next page)

(continued from previous page)

```
status = False
upPressed = False
downPressed = False
playPressed = False

def setup():
    mixer.init()
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin1, GPIO.IN, GPIO.PUD_UP)
    GPIO.setup(BtnPin2, GPIO.IN, GPIO.PUD_UP)
    GPIO.setup(BtnPin3, GPIO.IN, GPIO.PUD_UP)

def clip(x,min,max):
    if x < min:
        return min
    elif x > max:
        return max
    return x

def play(pin):
    global playPressed
    playPressed = True

def volDown(pin):
    global downPressed
    downPressed = True

def volUp(pin):
    global upPressed
    upPressed = True

def main():
    global volume, status
    global downPressed, upPressed, playPressed
    mixer.music.load('/home/pi/raphael-kit/music/my_music.mp3')
    mixer.music.set_volume(volume)
    mixer.music.play()
    GPIO.add_event_detect(BtnPin1, GPIO.FALLING, callback=play)
    GPIO.add_event_detect(BtnPin2, GPIO.FALLING, callback=volDown)
    GPIO.add_event_detect(BtnPin3, GPIO.FALLING, callback=volUp)
    while True:
        if upPressed:
            volume = volume + 0.1
            upPressed = False
        if downPressed:
            volume = volume - 0.1
            downPressed = False
        if playPressed:
            if status:
                mixer.music.pause()
                status = not status
            else:
                mixer.music.unpause()
                status = not status
            playPressed = False
            time.sleep(0.5)
```

(continues on next page)

(continued from previous page)

```
        volume = clip(volume,0.2,1)
        mixer.music.set_volume(volume)
        time.sleep(0.1)

def destroy():
    # Release resource
    GPIO.cleanup()
    mixer.music.stop()

# 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 Explanation

```
from pygame import mixer

mixer.init()
```

Import the `Mixer` method in the `pygame` library and initialize the method.

```
BtnPin1 = 18
BtnPin2 = 17
BtnPin3 = 27
volume = 0.7
```

Define the pin ports of the three buttons and set the initial volume to 0.7.

```
upPressed = False
downPressed = False
playPressed = False
```

`UpPressed`, `downPressed` and `playPressed` are all interrupt flags, the corresponding task will be executed When they are `True`.

```
def clip(x,min,max):
if x < min:
    return min
elif x > max:
    return max
return x
```

The `clip()` function is used to set the upper and lower limits of input parameters.

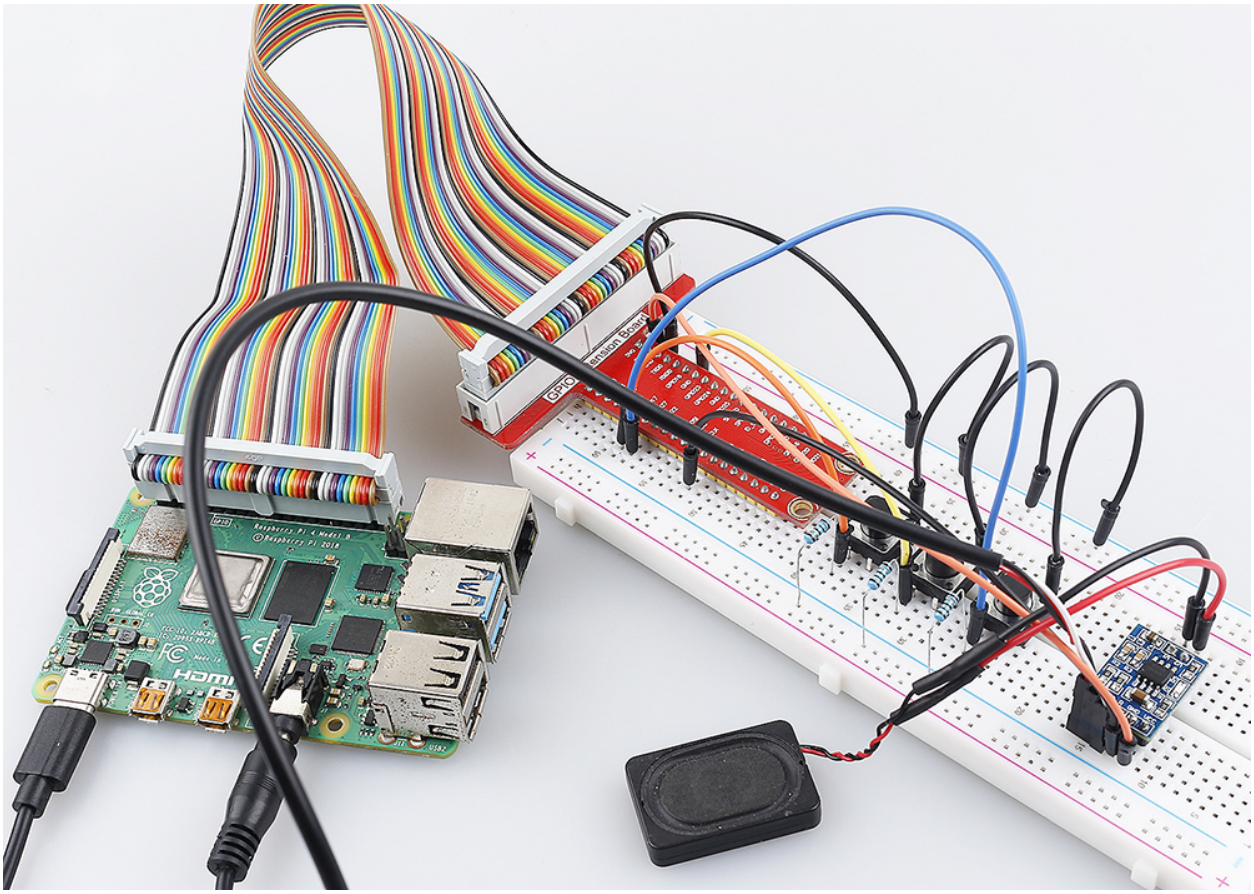
```
GPIO.add_event_detect(BtnPin1, GPIO.FALLING, callback=play)
GPIO.add_event_detect(BtnPin2, GPIO.FALLING, callback=volDown)
GPIO.add_event_detect(BtnPin3, GPIO.FALLING, callback=volUp)
```

Set the key detection events of `BtnPin1`, `BtnPin2` and `BtnPin3`.

- When `BtnPin1` is pressed, the interrupt function `play()` is executed.

- when `BtnPin2` is pressed, the interrupt function `volDown()` is executed.
- When `BtnPin3` is pressed, the interrupt function `volUp()` is executed.

Phenomenon Picture

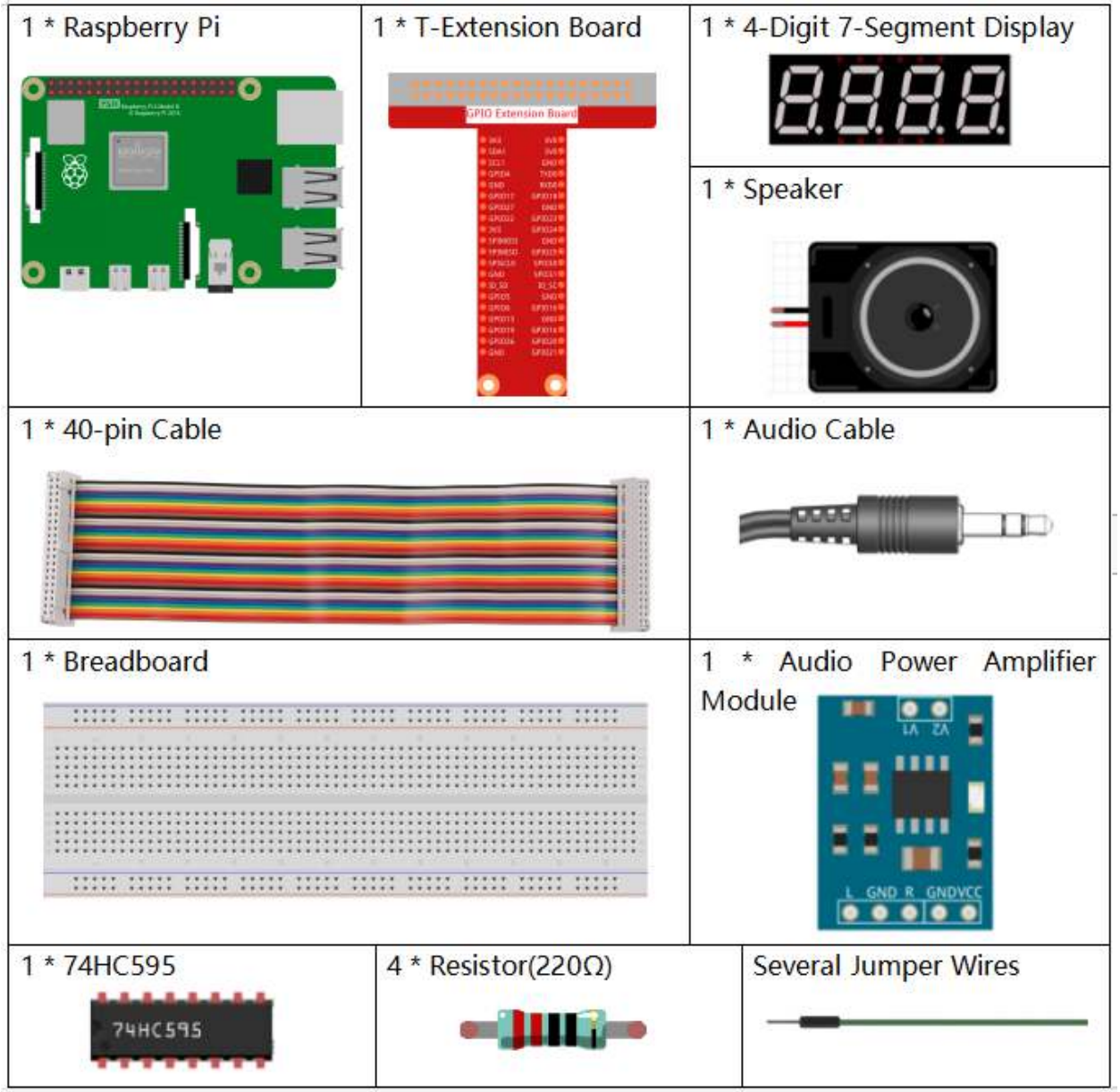


6.6.3 4.1.3 Speech Clock

Introduction

In this project, let's make a voice clock with a speaker and a 4-digit 7-segment display. The 4-digit 7-segment display will display the time, and the speaker will broadcast the time every hour.

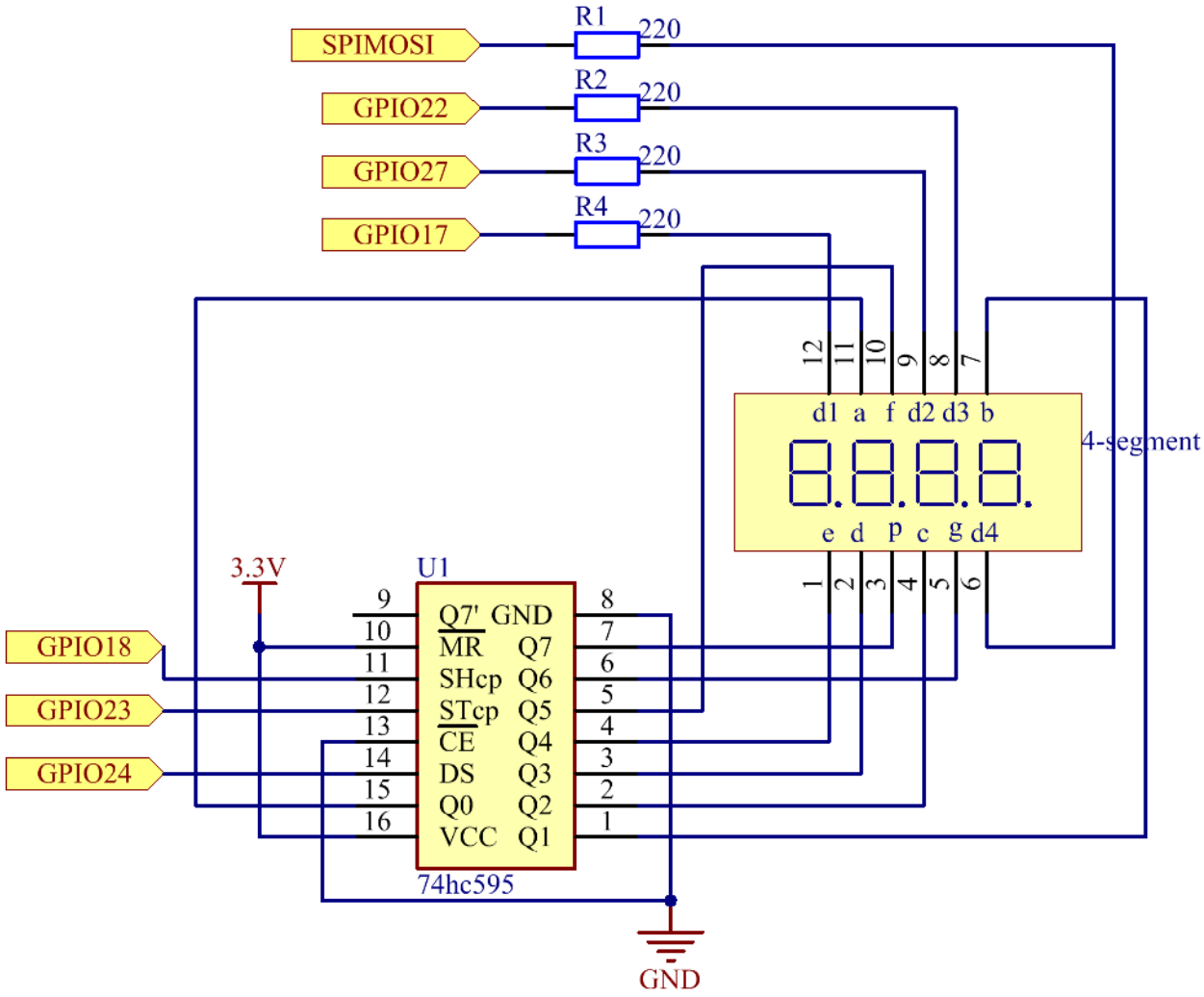
Components

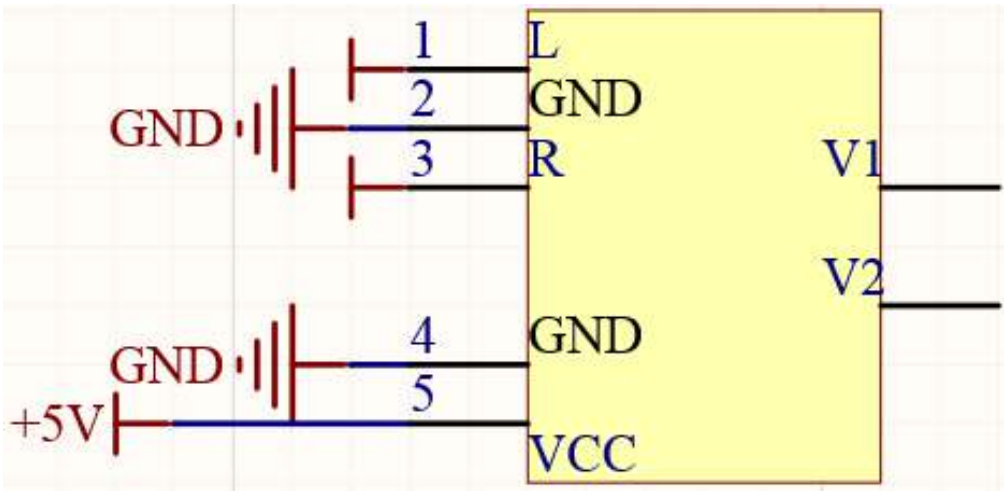


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Audio Module and Speaker*
- *4-Digit 7-Segment Display*
- *74HC595*

Schematic Diagram

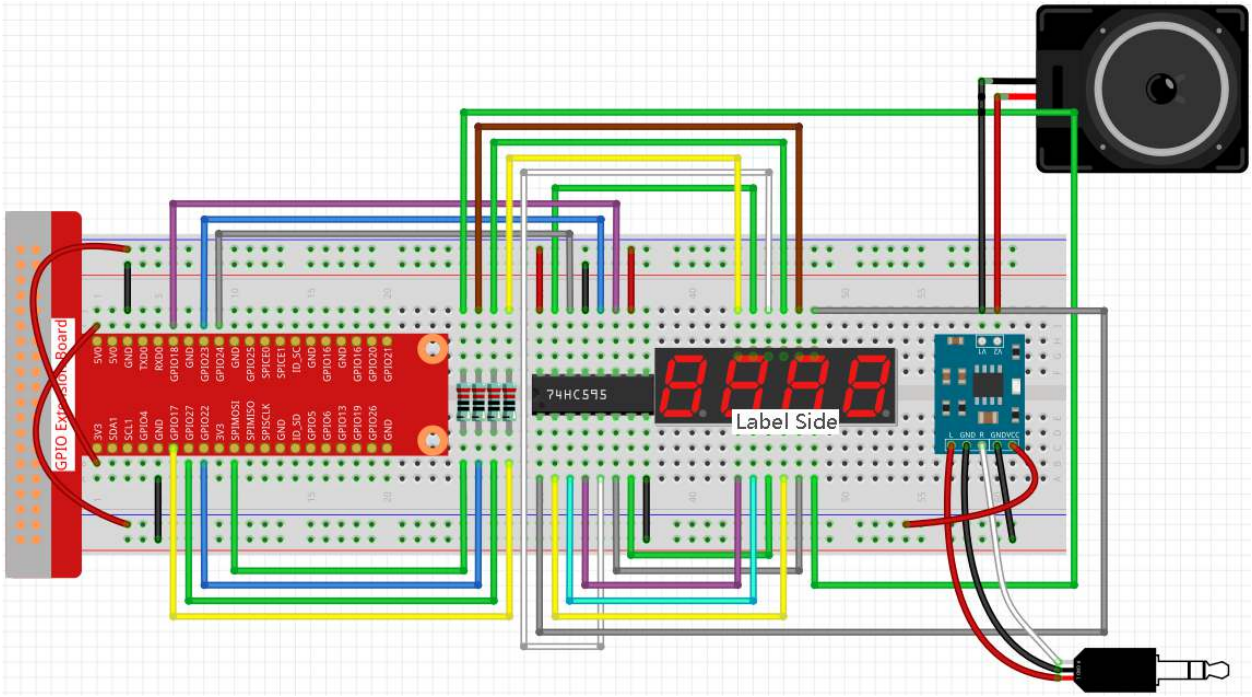
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24





Experimental Procedures

Step 1: Build the circuit.



Before this project, you need to make sure you complete *3.1.4 Text-to-speech*.

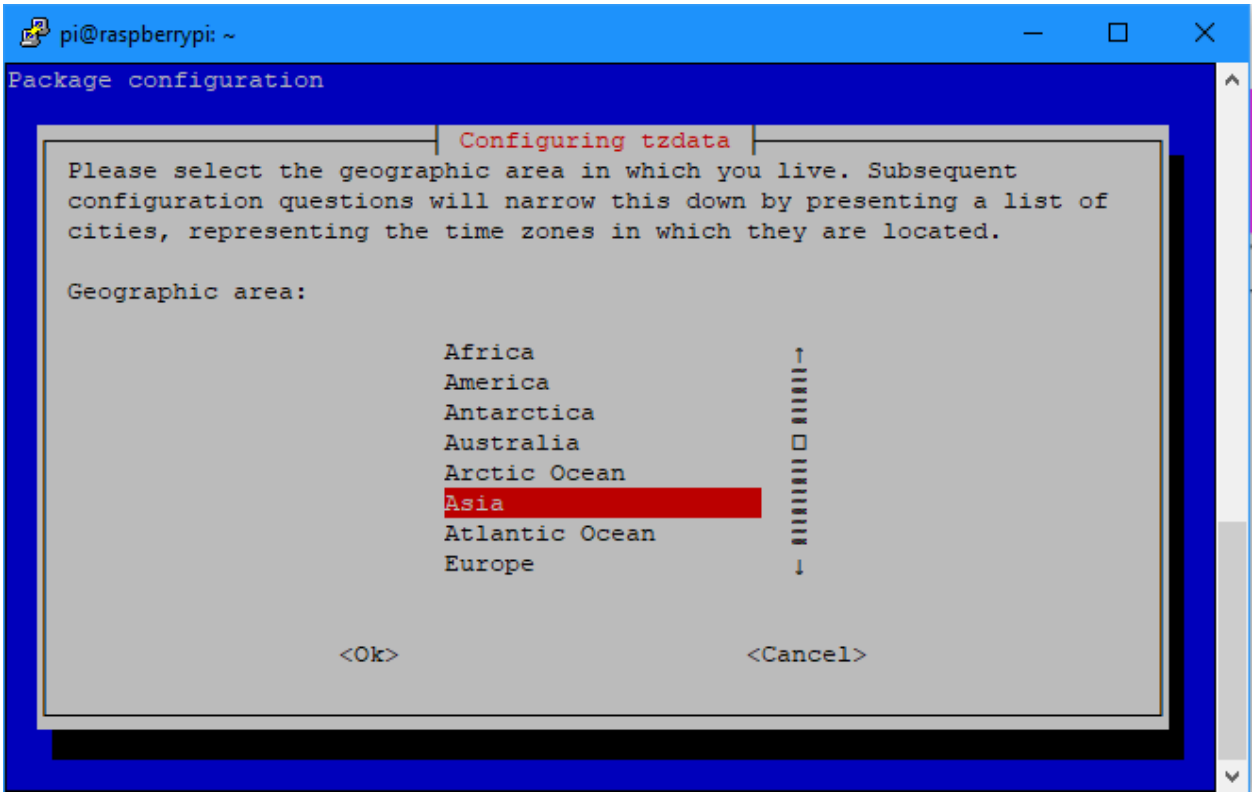
Step 2: Use the command `date` to view the local time.

```
date
```

If the local time is different from the real time, you need to use the following command to set the time zone.

```
sudo dpkg-reconfigure tzdata
```

Choose your time zone.



Step 3: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
python3 4.1.3_SpeechClock.py
```

When the code is run, the 4-digit 7-segment will display the time and chime on every hour.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
from tts import TTS
import time

tts = TTS(engine="espeak")
tts.lang('en-US')

SDI = 24
RCLK = 23
SRCLK = 18

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

(continues on next page)

(continued from previous page)

```
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)

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 loop():
    status = 0
    while True:
        time.localtime(time.time())
        hour = int(time.strftime('%H',time.localtime(time.time())))
        minute = int(time.strftime('%M',time.localtime(time.time())))

        clearDisplay()
        pickDigit(0)
        hc595_shift(number[minute % 10])

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

        clearDisplay()
        pickDigit(2)
        hc595_shift(number[hour % 10])

        clearDisplay()
        pickDigit(3)
        hc595_shift(number[hour % 100//10])

        if minute == 0 and status == 0:
            tts.say('The time is now ' + str(hour) + ' hours and ' + str(minute) + '␣
↪minutes')
            status = 1
        elif minute != 0:
```

(continues on next page)

(continued from previous page)

```
            status = 0

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 Explanation

```
time.localtime(time.time())
hour = int(time.strftime('%H',time.localtime(time.time())))
minute = int(time.strftime('%M',time.localtime(time.time())))
```

Through the function `time.time()`, we can get the timestamp of the current time (the number of floating-point seconds that have passed since the 1970 epoch), and then use the time formatting method of the time module `(time.localtime(time.time()))` to process the current timestamp, so that we can format the timestamp as a local time.

The input result is:

```
time.struct_time(tm_year=2021, tm_mon=5, tm_mday=28, tm_hour=13, tm_min=54,
tm_sec=26, tm_wday=4, tm_yday=148, tm_isdst=0)
```

Finally, we use the `time.strftime()` method to format the large string of information into what we want. If you want to get the current hour, you can get it through the function `time.strftime('%H',time.localtime(time.time()))`.

The output of the specified formatted string obtained by modifying the first parameter are listed below.

%y	Two-digit year representation(00-99)
%Y	Four-digit year representation(000-9999)
%m	month(01-12)
%H	Day of the month(0-31)
%I	Hours in a 24-hour clock(0-23)
%M	Hours in 12-hour clock(01-12)
%y	Minutes(00=59)
%S	second(00-59)
%a	Local simplified week name
%A	Full local week name
%b	Local simplified month name
%B	Local full month name
%c	Local corresponding date and time display
%j	Day of the year(001-366)
%p	The equivalent of local A.M. or P.M.
%U	Num of weeks of one year(00-53)starting with Sunday
%w	Week (0-6), starting with Sunday
%W	Num of weeks of one year(00-53)starting with Monday
%x	Local corresponding date representation
%X	Local corresponding time representation
%Z	The name of the current time zone

Note: The output of the `time.strftime()` method is all string variables. Before using it, remember to do a coercive type conversion.

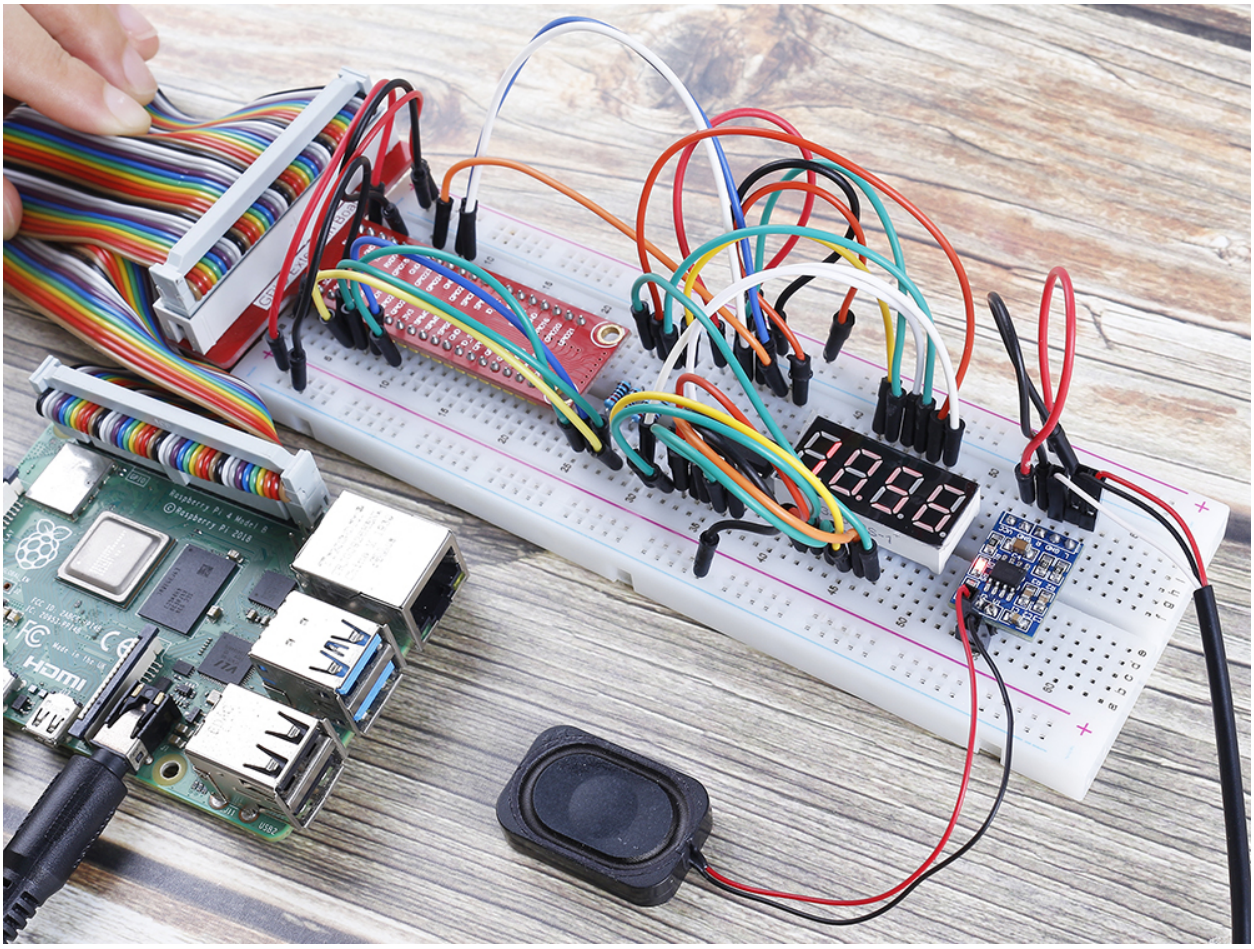
```
clearDisplay()
pickDigit(0)
hc595_shift(number[minute % 10])

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

clearDisplay()
pickDigit(2)
hc595_shift(number[hour % 10])

clearDisplay()
pickDigit(3)
hc595_shift(number[hour % 100//10])
```

The tens digit of the hour is displayed on the first 7-segment digital display, and the ones digit is displayed on the second. Then the tens digit of the minutes is displayed on the third digital display, and the ones digit are displayed on the last.

```
if minute == 0 and status == 0:
    tts.say('The time is now ' + str(hour) + ' hours and ' + str(minute) + ' minutes')
    status = 1
elif minute != 0:
    status = 0
```

When the number of minutes is 0 (by hour), the Raspberry Pi will use TTS to announce the time for us.

Phenomenon Picture

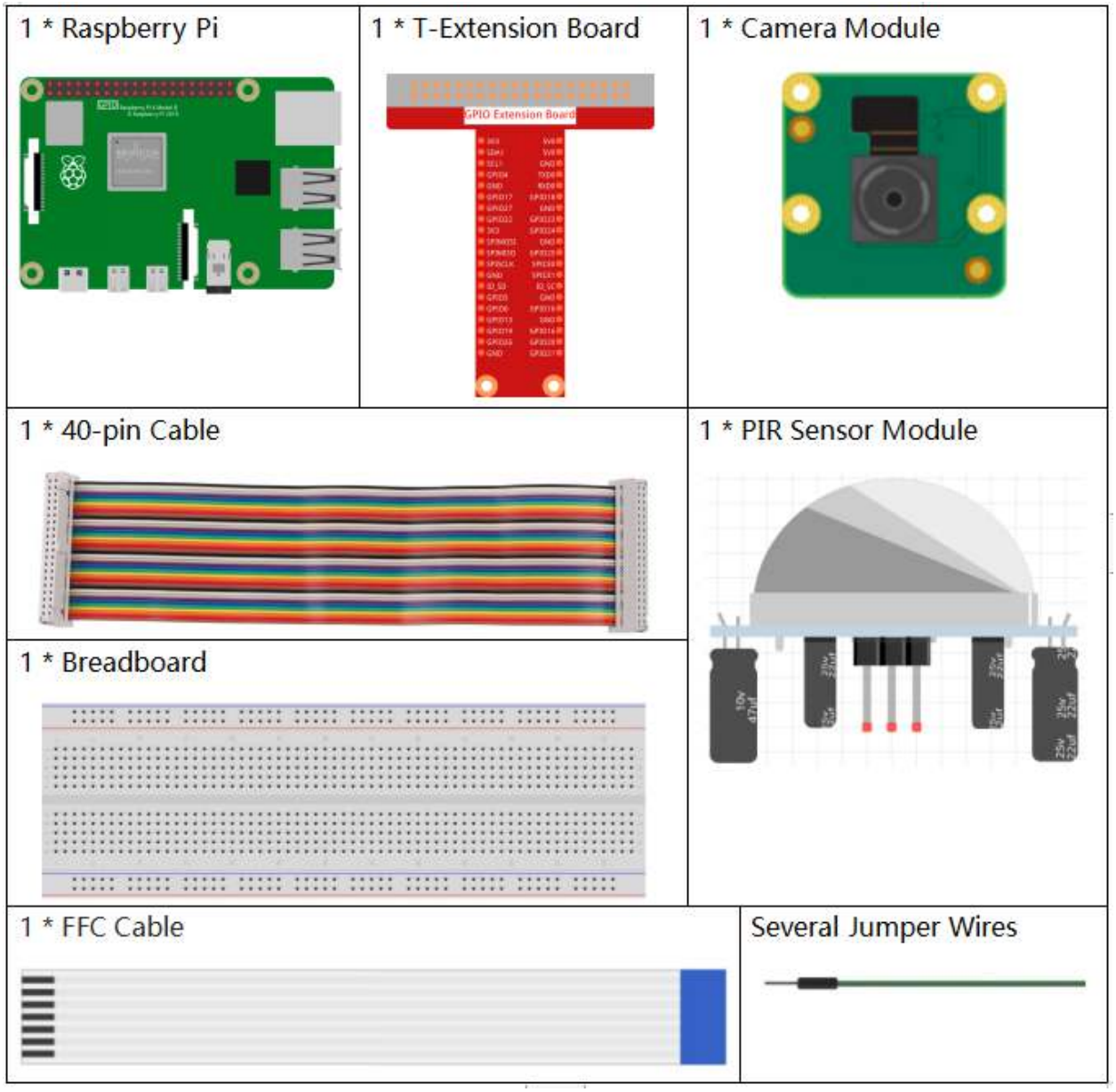


6.6.4 4.1.4 Automatic Capture Camera

Introduction

When you are out, the little squirrels in the woods might visit your windowsill. Let's make a automatic capture camera to leave pictures of these little cuties!

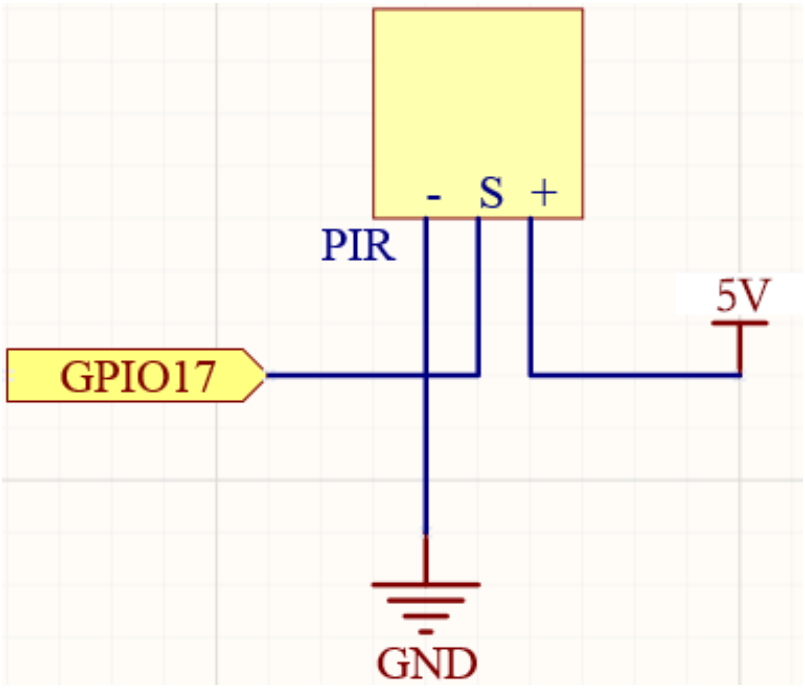
Components



- *GPIO Extension Board*
- *Breadboard*
- *Camera Module*
- *PIR Motion Sensor Module*

Schematic Diagram

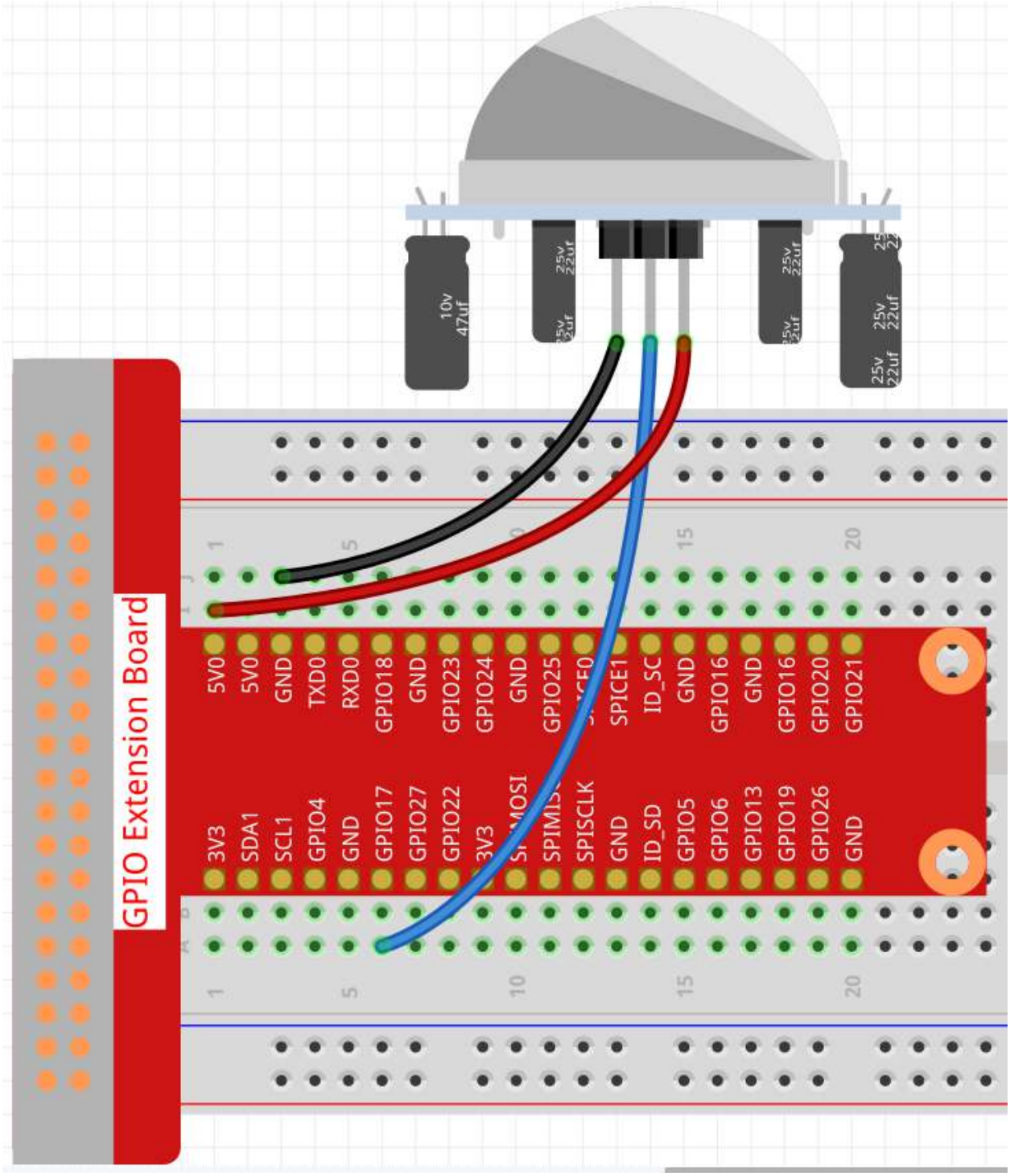
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



Experimental Procedures

Before this project, you need to make sure you complete *3.1.1 Photograph Module* .

Step 1: Build the circuit.



Step 2: To connect the camera module and complete the configuration, please refer to: *Camera Module*.

Step 3: Go into the Raspberry Pi Desktop. You may need a screen for a better experience, refer to: Connect your Raspberry Pi. Or access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

Step 4: Open a Terminal and get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 5: Run.

```
sudo python3 4.1.4_AutomaticCaptureCamera.py
```

After the code runs, PIR will detect the surrounding environment, and if it senses the little squirrel passing by, the camera will take a photo. The photo interval is 3 seconds, and the total number of photos taken will be displayed through the print window.

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.

Note: You can also open `4.1.4_AutomaticCaptureCamera.py` in the `/home/pi/raphael-kit/python/` path with a Python IDE, click Run button to run, and stop the code with Stop button.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

from picamera import PiCamera
import RPi.GPIO as GPIO
import time

camera = PiCamera()

pirPin = 17    # the pir connect to pin17

def setup():
   GPIO.setmode(GPIO.BCM)
   GPIO.setup(pirPin, GPIO.IN)
   camera.start_preview(alpha=200)
```

(continues on next page)

(continued from previous page)

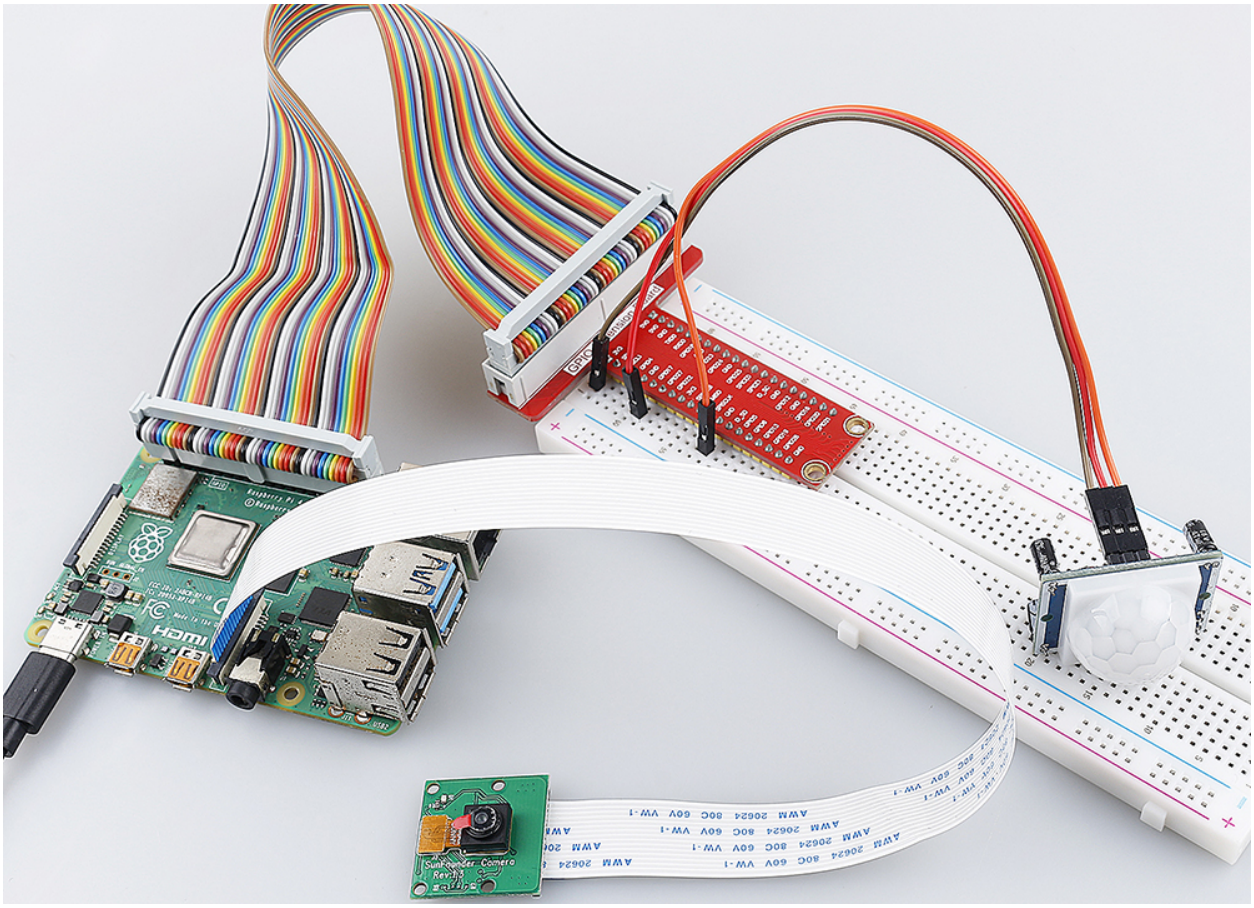
```
def main():
    i = 1
    while True:
        pirVal = GPIO.input(pirPin)
        if pirVal==GPIO.HIGH:
                camera.capture('/home/pi/capture%s.jpg' % i)
                print('The number is %s' % i)
                time.sleep(3)
                i = i + 1

def destroy():
    GPIO.cleanup()
    camera.stop_preview()

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

Code Explanation

```
pirVal = GPIO.input(pirPin)
if pirVal==GPIO.HIGH:
        camera.capture('/home/pi/capture%s.jpg' % i)
        print('The number is %s' % i)
        time.sleep(3)
        i = i + 1
```

Every time a little squirrel is detected by the PIR module, the Raspberry Pi will take a photo and tell you through the print window how many pictures have been taken. The interval between each photo is 3s.

Phenomenon Picture

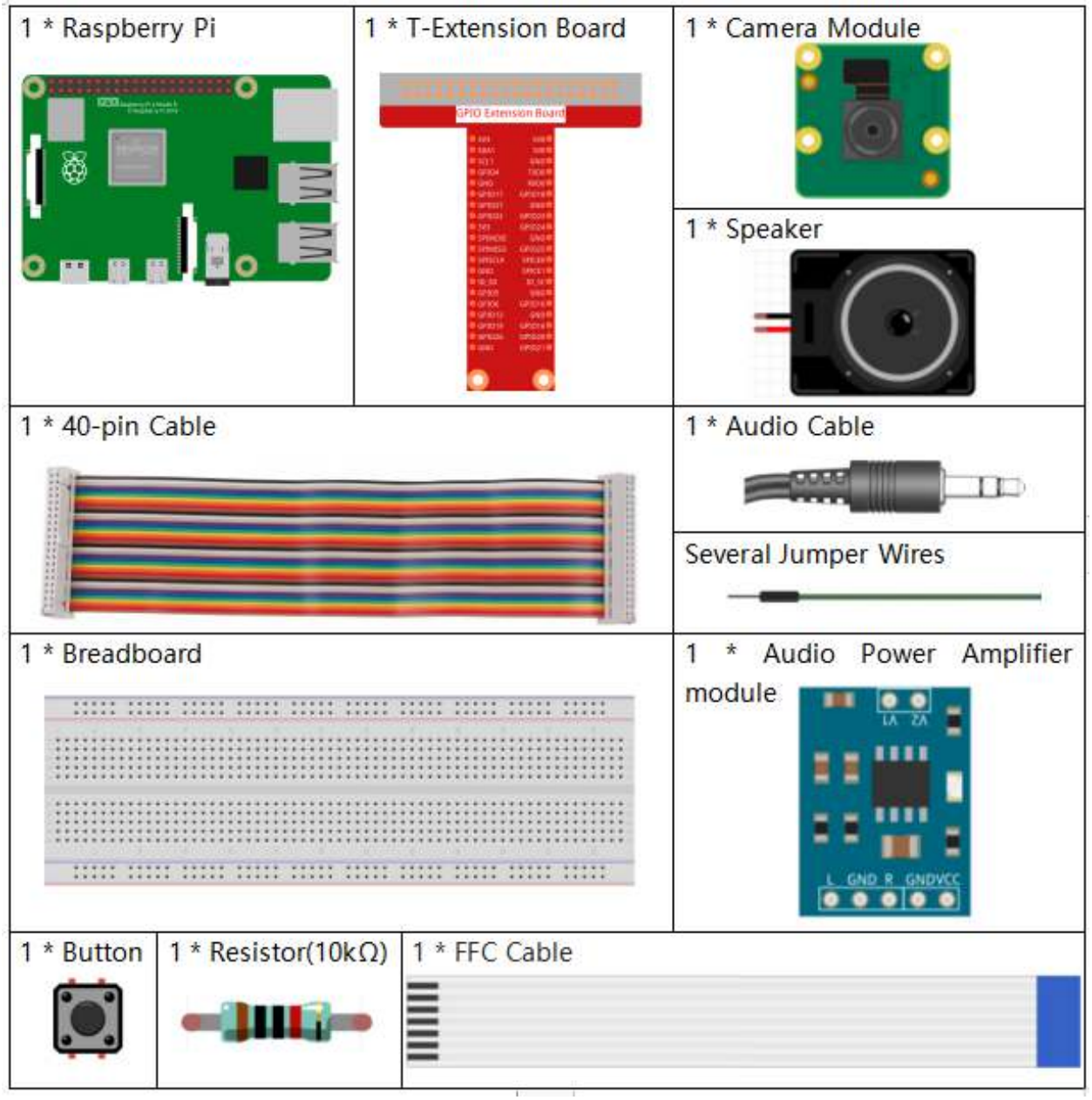


6.6.5 4.1.5 Intelligent Visual Doorbell

Introduction

In this project, let's make a DIY intelligent visual doorbell.

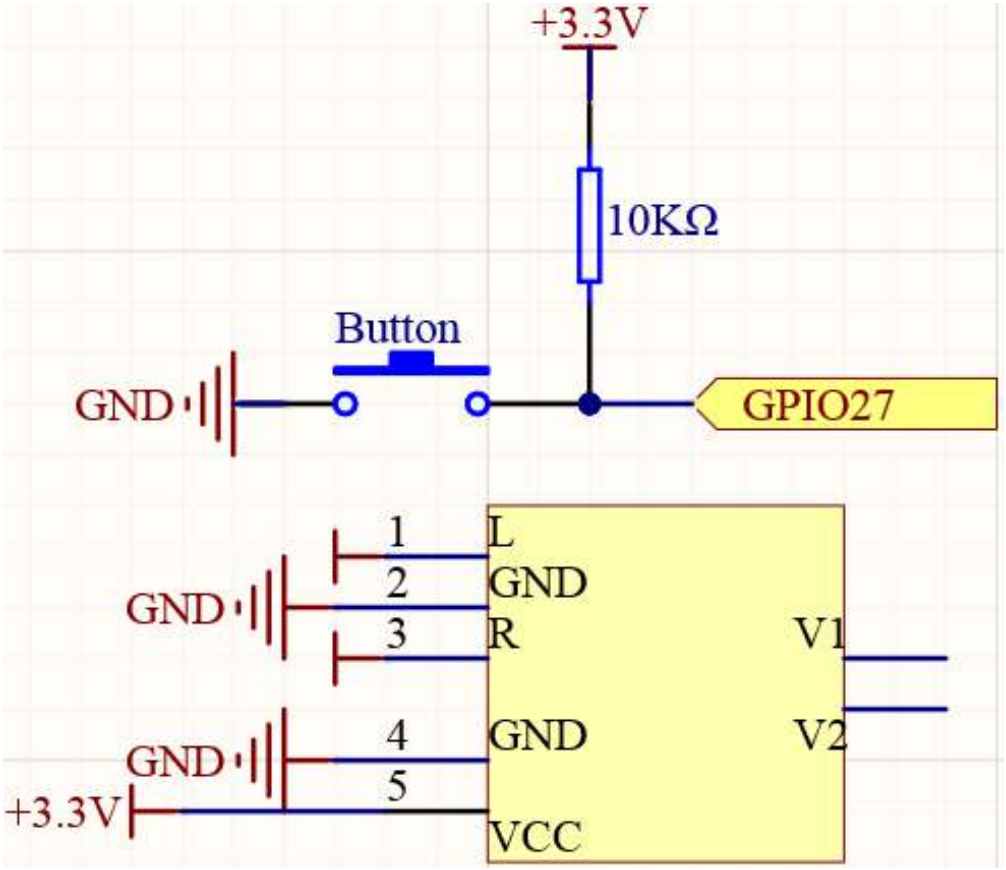
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Button*
- *Audio Module and Speaker*
- *Camera Module*

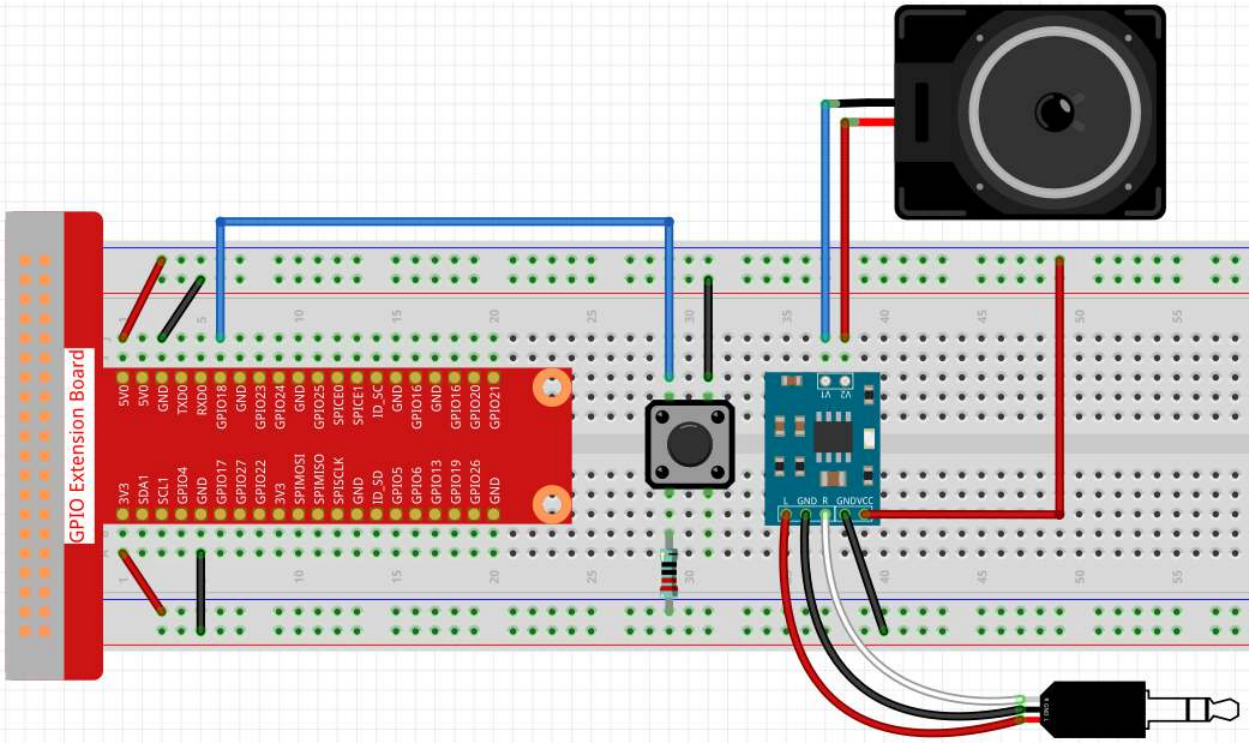
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO27	Pin 13	2	27



Experimental Procedures

Step 1: Build the circuit.



Before this project, you need to make sure you complete *3.1.3 Audio Module* & *3.1.2 Video Module*.

Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
python3 4.1.5_DoorBell.py
```

After the code runs, when the button is pressed, a bell will sound, and the camera will record a 5s video, which is stored as the `visitor.h264` file in the `/home/pi` directory. If you have a screen, you can also view visitors by previewing the video in real time.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
from picamera import PiCamera
from pygame import mixer
import RPi.GPIO as GPIO
import time

camera = PiCamera()

BtnPin = 18
status = False
```

(continues on next page)

(continued from previous page)

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN, GPIO.PUD_UP)
    mixer.init()

def takePhotos(pin):
    global status
    status = True

def main():
    global status
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=takePhotos)
    while True:
        if status:
            mixer.music.load('/home/pi/raphael-kit/music/doorbell.wav')
            mixer.music.set_volume(0.7)
            mixer.music.play()
            camera.start_preview(alpha=200)
            camera.start_recording('/home/pi/visitor.h264')
            print ('Have a visitor')
            time.sleep(5)
            mixer.music.stop()
            camera.stop_preview()
            camera.stop_recording()
            status = False

def destroy():
    GPIO.cleanup()
    mixer.music.stop()
    camera.stop_preview()
    camera.stop_recording()

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

Code Explanation

```
status = False
```

This is a flag used to record whether the doorbell is used.

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

Set the event of `BtnPin`, when the button is pressed (the level signal changes from high to low) , call the function `takePhotos()`.

```
if status:
    mixer.music.load('/home/pi/raphael-kit/music/doorbell.wav')
    mixer.music.set_volume(0.7)
    mixer.music.play()
    camera.start_preview(alpha=200)
    camera.start_recording('/home/pi/visitor.h264')
```

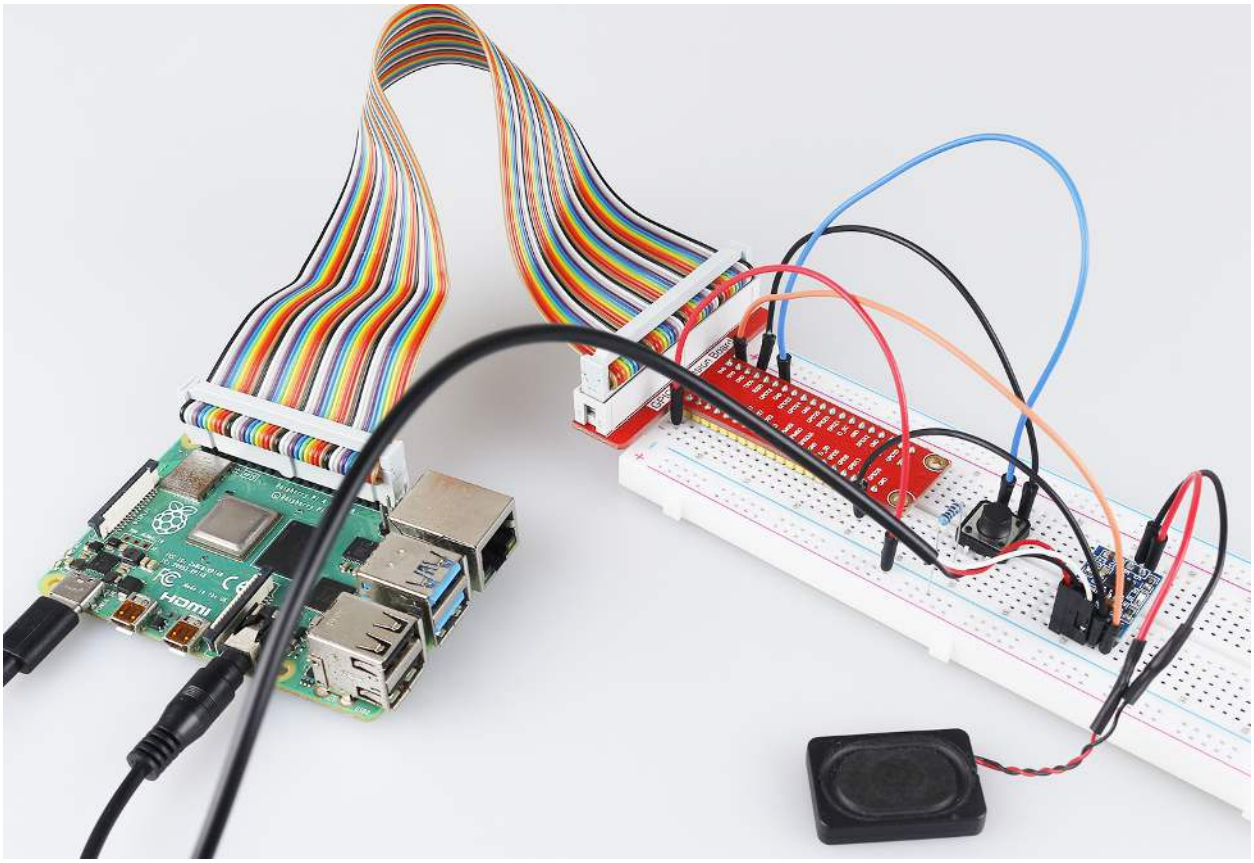
(continues on next page)

(continued from previous page)

```
    print ('Have a visitor')
    time.sleep(5)
    mixer.music.stop()
    camera.stop_preview()
    camera.stop_recording()
    status = False
```

Five seconds are used here to play music and record videos, thus functioning as a doorbell.

Phenomenon Picture

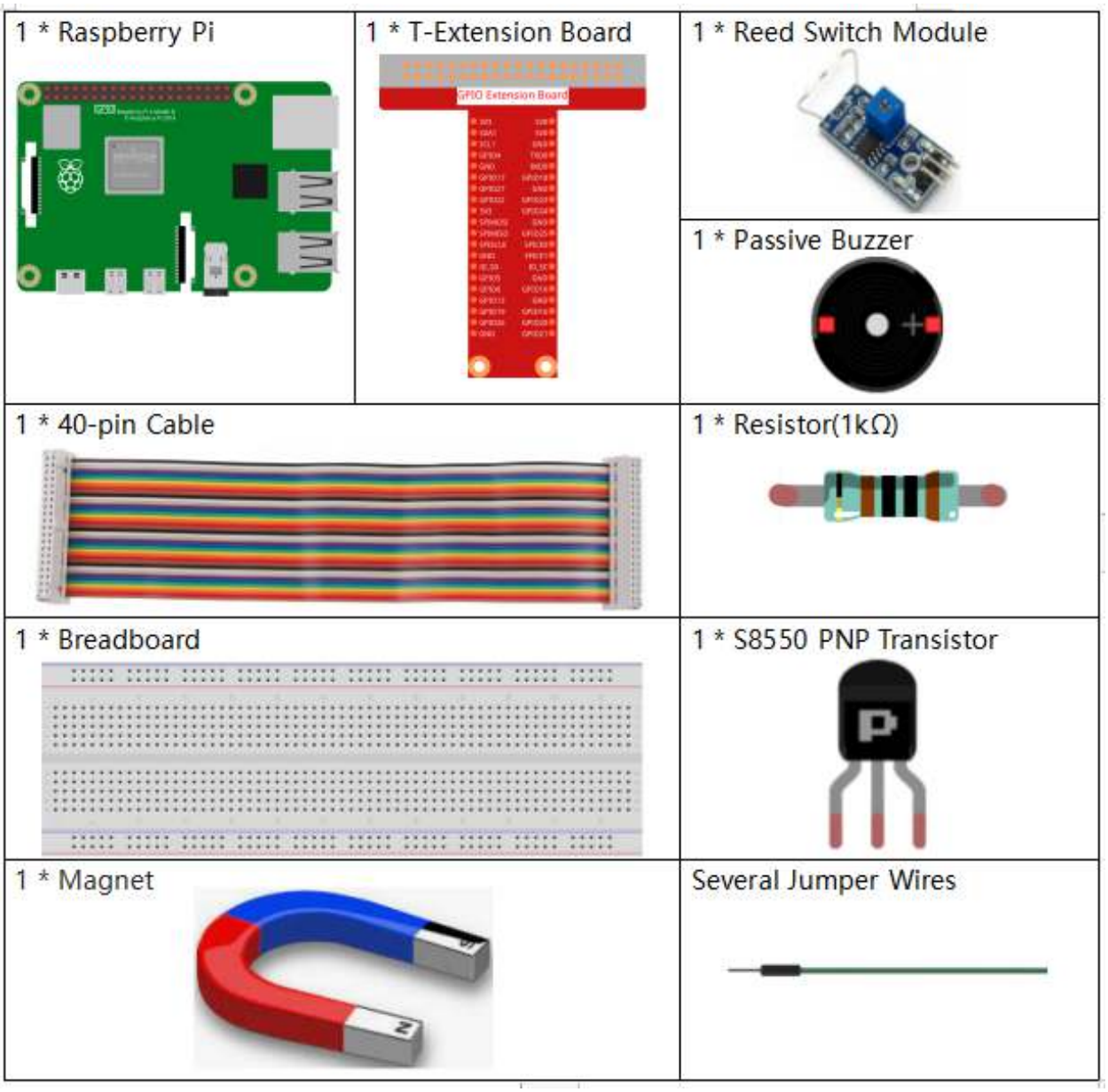


6.6.6 4.1.6 Magnetic Induction Alarm System

Introduction

When you get a precious vase, you can make a magnetic induction alarm system for it, no matter who moves it, you can hear the alarm in time.

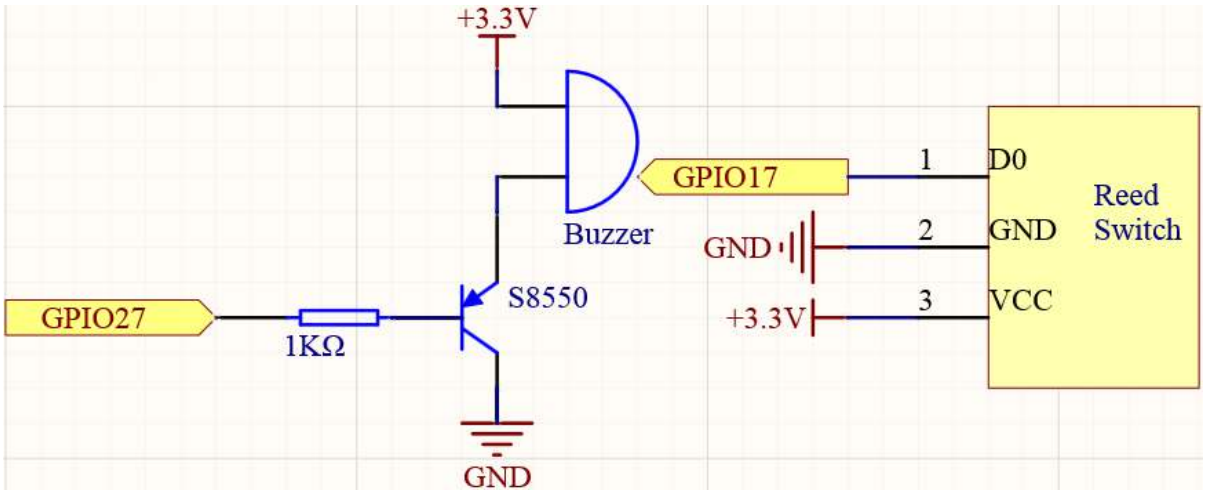
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*
- *Reed Switch Module*

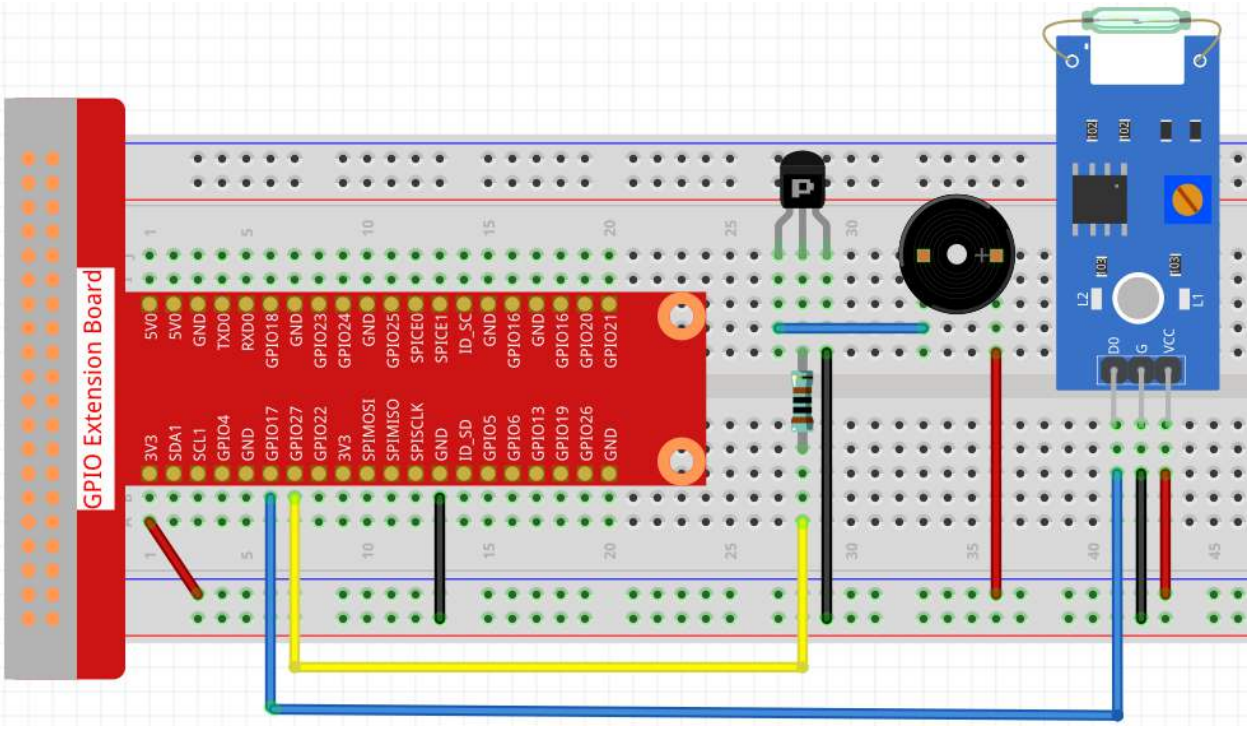
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.6_MagneticAlarmSystem.py
```

If the reed switch is affected by the magnet (for example, the reed switch is placed on the base and the magnet is placed in the vase), the object is safe. At this time, the reed switch is in the closed state, and the buzzer is silent. After removing the magnet (such as the vase being stolen), the reed switch is not affected by the magnetic, the switch opens, and the buzzer sounds an alarm.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time

BeepPin = 17
ReedPin = 18

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(ReedPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def loop():
    while True:
        if GPIO.input(ReedPin) == 0:
            GPIO.output(BeepPin, GPIO.HIGH)
        else:
            GPIO.output(BeepPin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(BeepPin, GPIO.HIGH)
            time.sleep(0.1)

def destroy():
    GPIO.output(BeepPin, GPIO.HIGH)
    GPIO.cleanup()

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

Code Explanation

```
def loop():
    while True:
        if GPIO.input(ReedPin) == 0:
            GPIO.output(BeepPin, GPIO.HIGH)
```

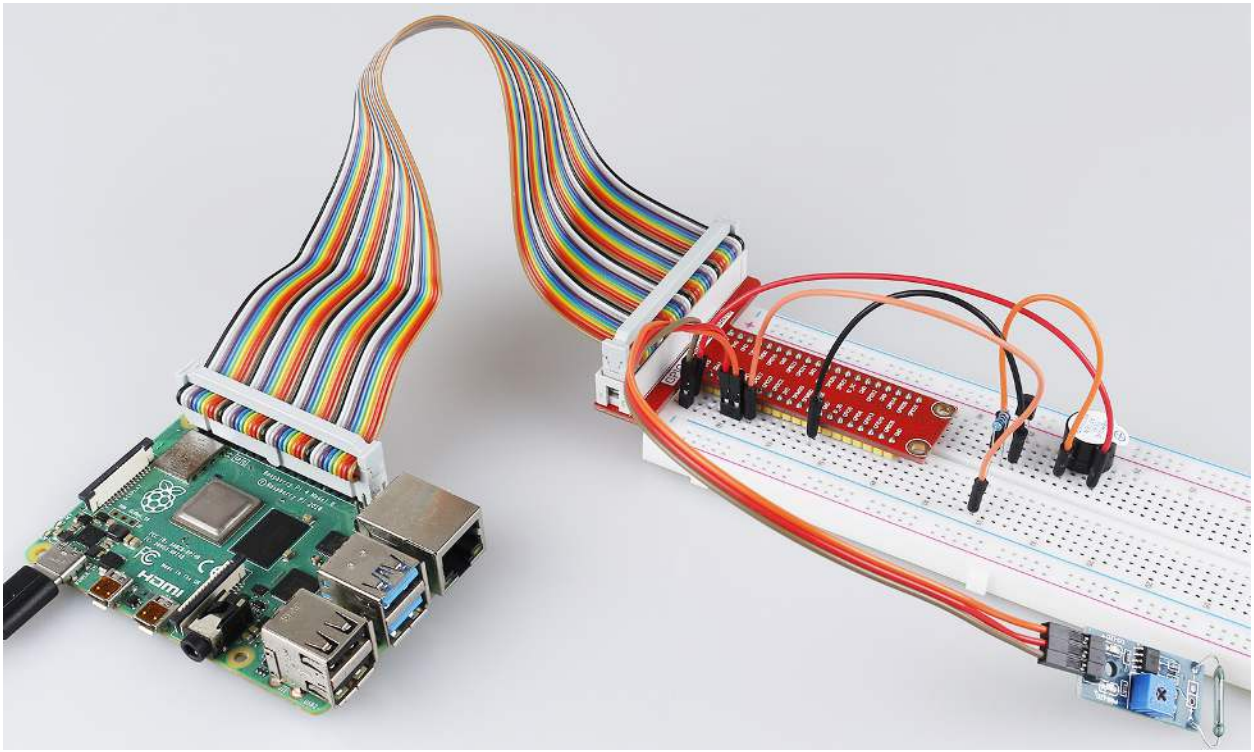
(continues on next page)

(continued from previous page)

```
        else:
            GPIO.output(BeepPin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(BeepPin, GPIO.HIGH)
            time.sleep(0.1)
```

We judge the state of the reed switch in the main loop. If the reed switch is closed, the buzzer does not work; otherwise, the buzzer beeps.

Phenomenon Picture

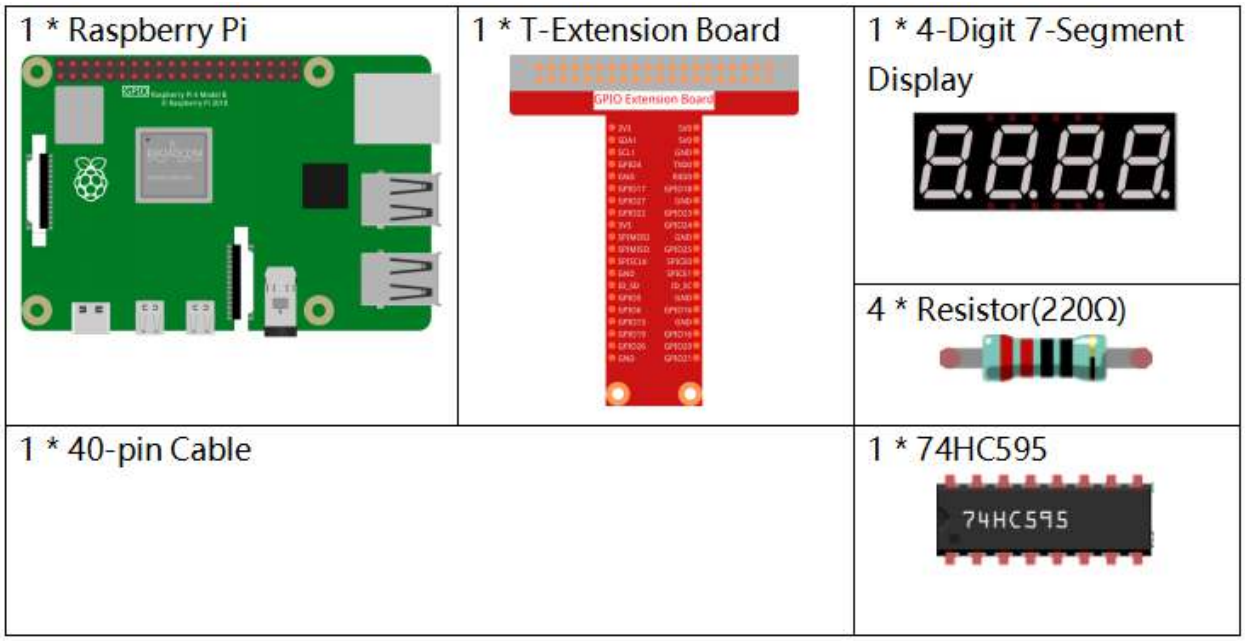


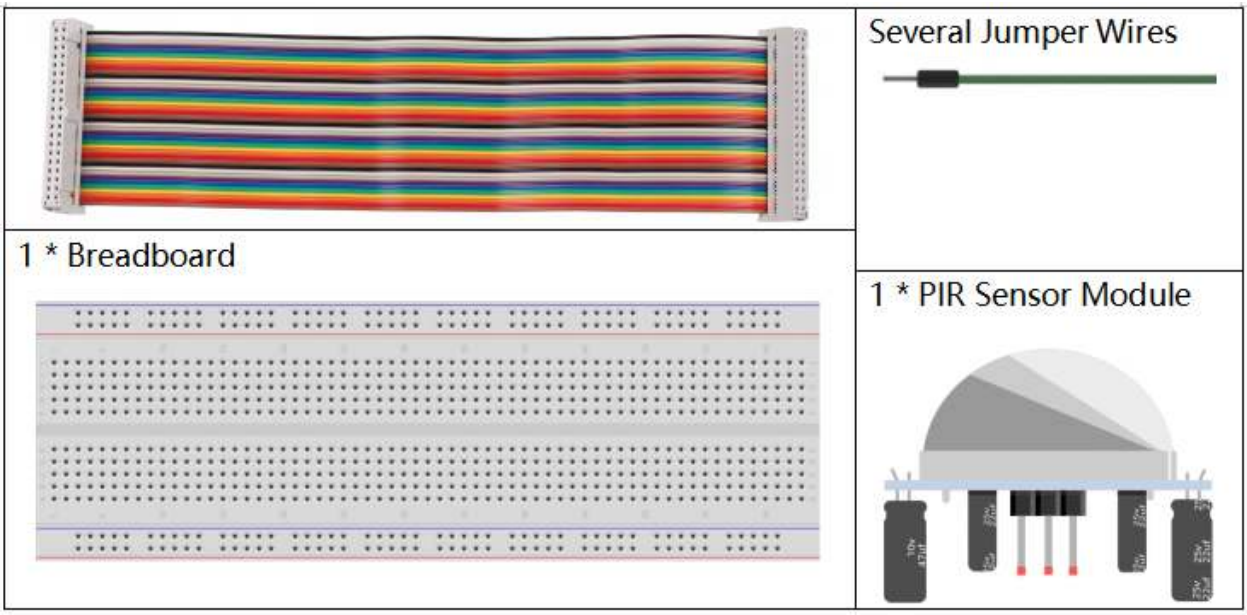
6.6.7 4.1.7 Counting Device

Introduction

Here we will make a number-displaying counter system, consisting of a PIR sensor and a 4-digit segment display. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1. You can use this counter to count the number of people walking through the passageway.

Components

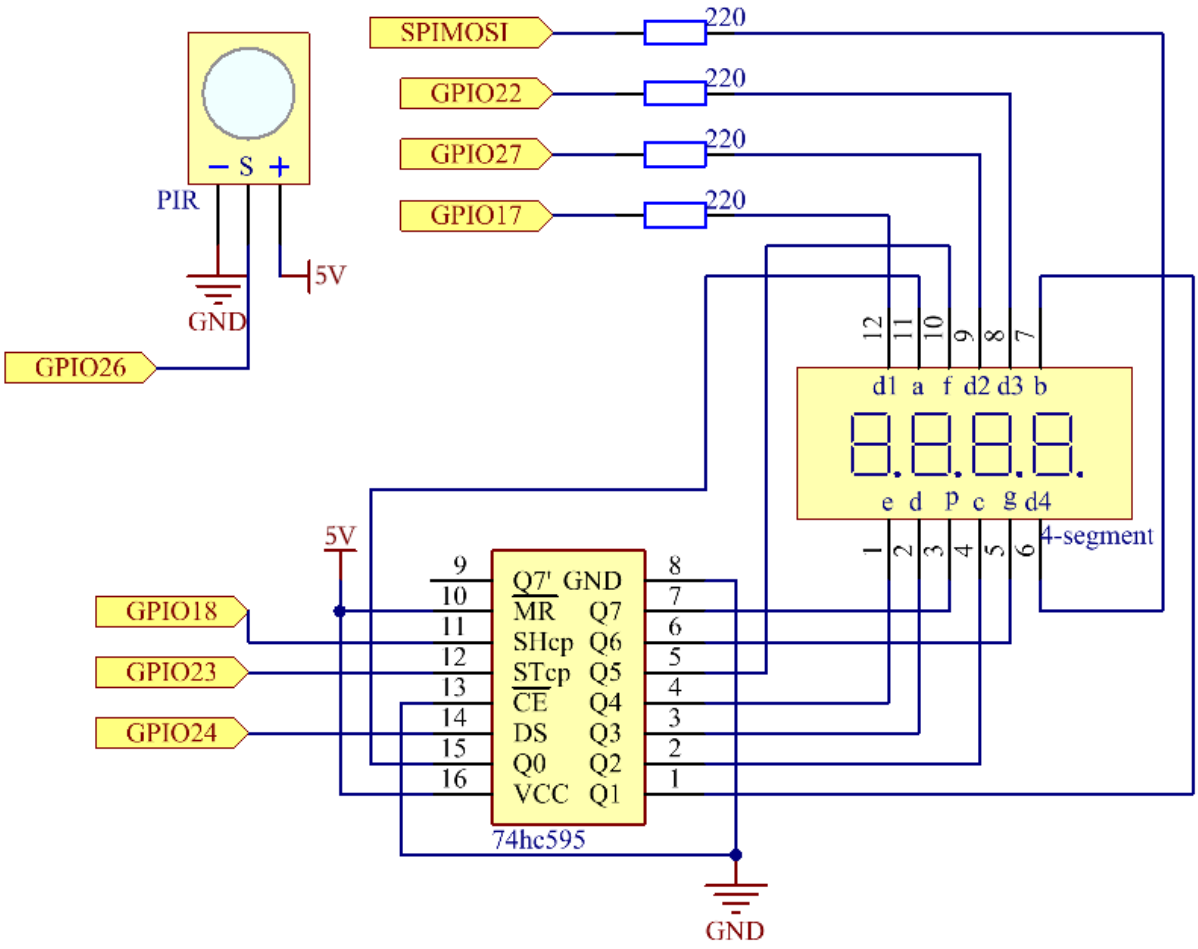




- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*
- *PIR Motion Sensor Module*

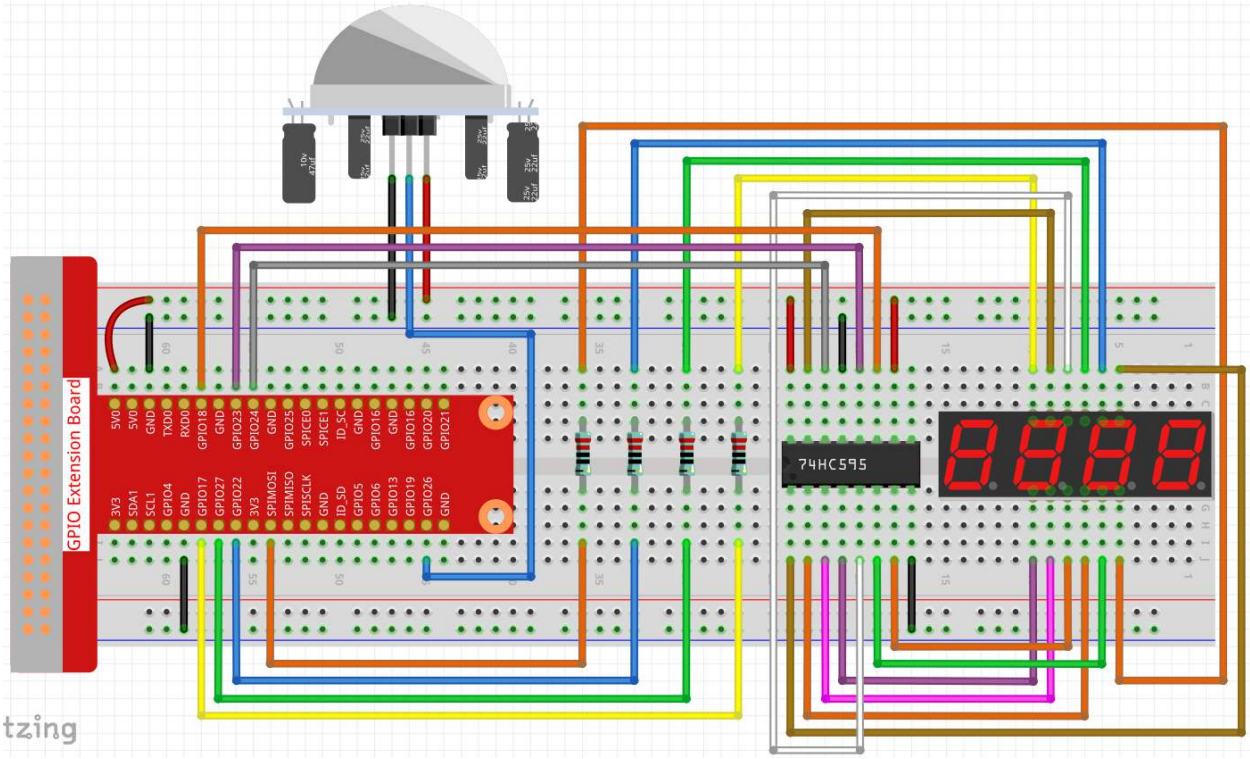
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 4.1.7_CountingDevice.py
```

After the code runs, when the PIR detects that someone is passing by, the number on the 4-digit segment display will add 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

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

(continues on next page)

(continued from previous page)

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

def display():
    global counter
    clearDisplay()
    pickDigit(0)
    hc595_shift(number[counter % 10])

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

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

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

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 Explanation

Based on *1.1.5 4-Digit 7-Segment Display*, this project adds **PIR module** to change the automatic counting into count detecting. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

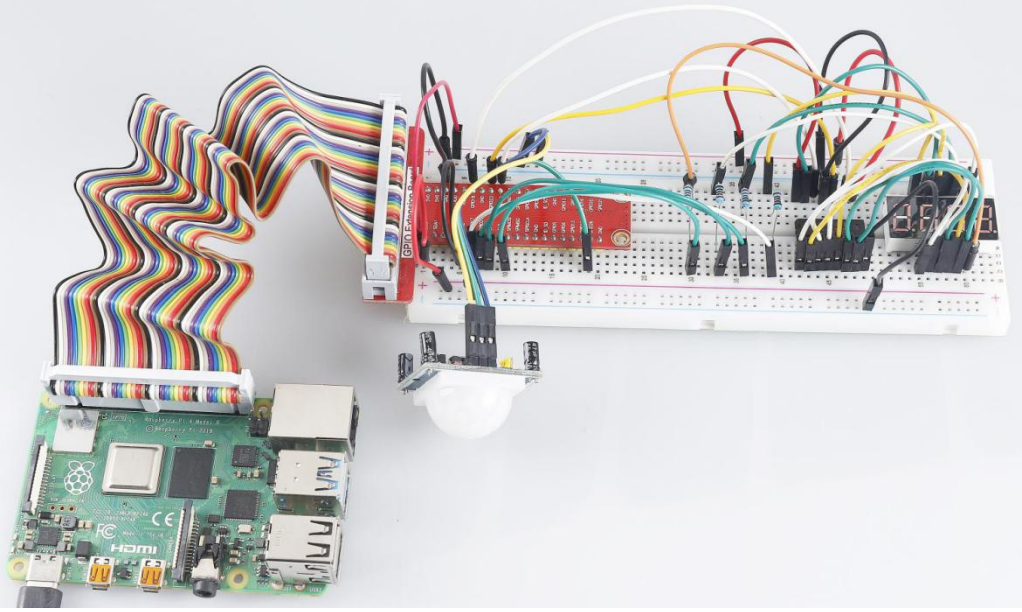
```
def display():
    global counter
    clearDisplay()
    pickDigit(0)
    hc595_shift(number[counter % 10])

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

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

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

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
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
```

This is the main function: display the number on the 4-digit segment display and read the PIR value. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

Phenomenon Picture

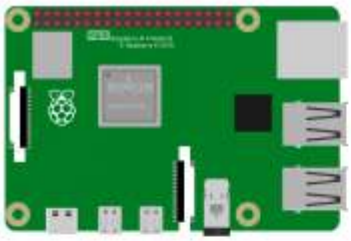
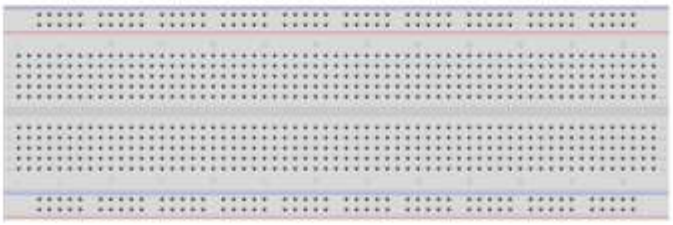


6.6.8 4.1.8 Welcome

Introduction

In this project, we will use PIR to sense the movement of pedestrians, and use servos, LED, buzzer to simulate the work of the sensor door of the convenience store. When the pedestrian appears within the sensing range of the PIR, the indicator light will be on, the door will be opened, and the buzzer will play the opening bell.

Components

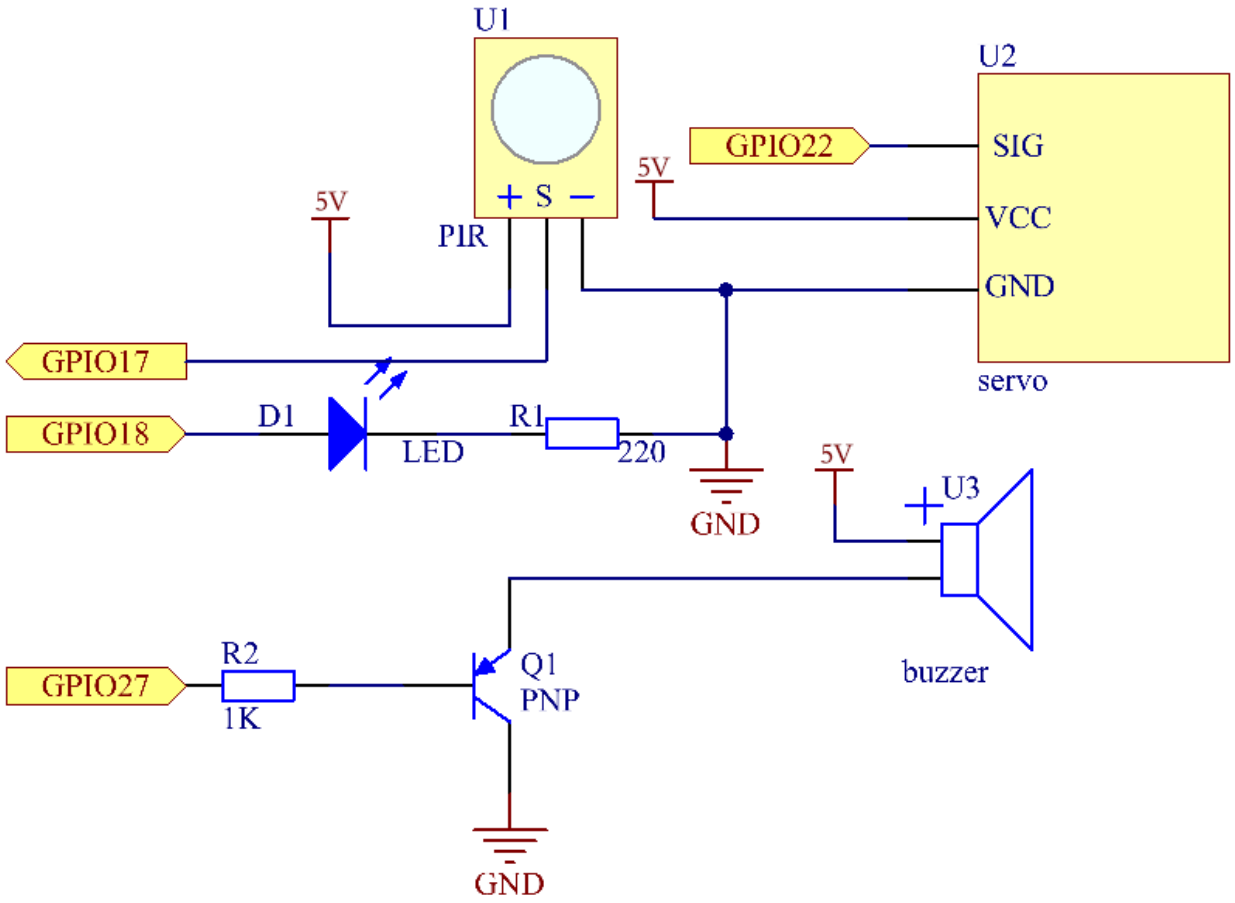
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td colspan="2">1 * PIR
 </td></tr>
<tr><td>1 * Servo
 </td><td>1 * Passive Buzzer
 </td><td>1 * S8550 PNP Transistor
 </td><td>1 * LED
 </td></tr>
<tr><td>1 * Resistor 1kΩ
 </td><td>1 * Resistor 220Ω
 </td><td colspan="2">Several Jumper Wires
</td></tr>
<tr><td colspan="4">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="4">1 * Breadboard
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *PIR Motion Sensor Module*
- *Servo*

- *Buzzer*
- *Transistor*

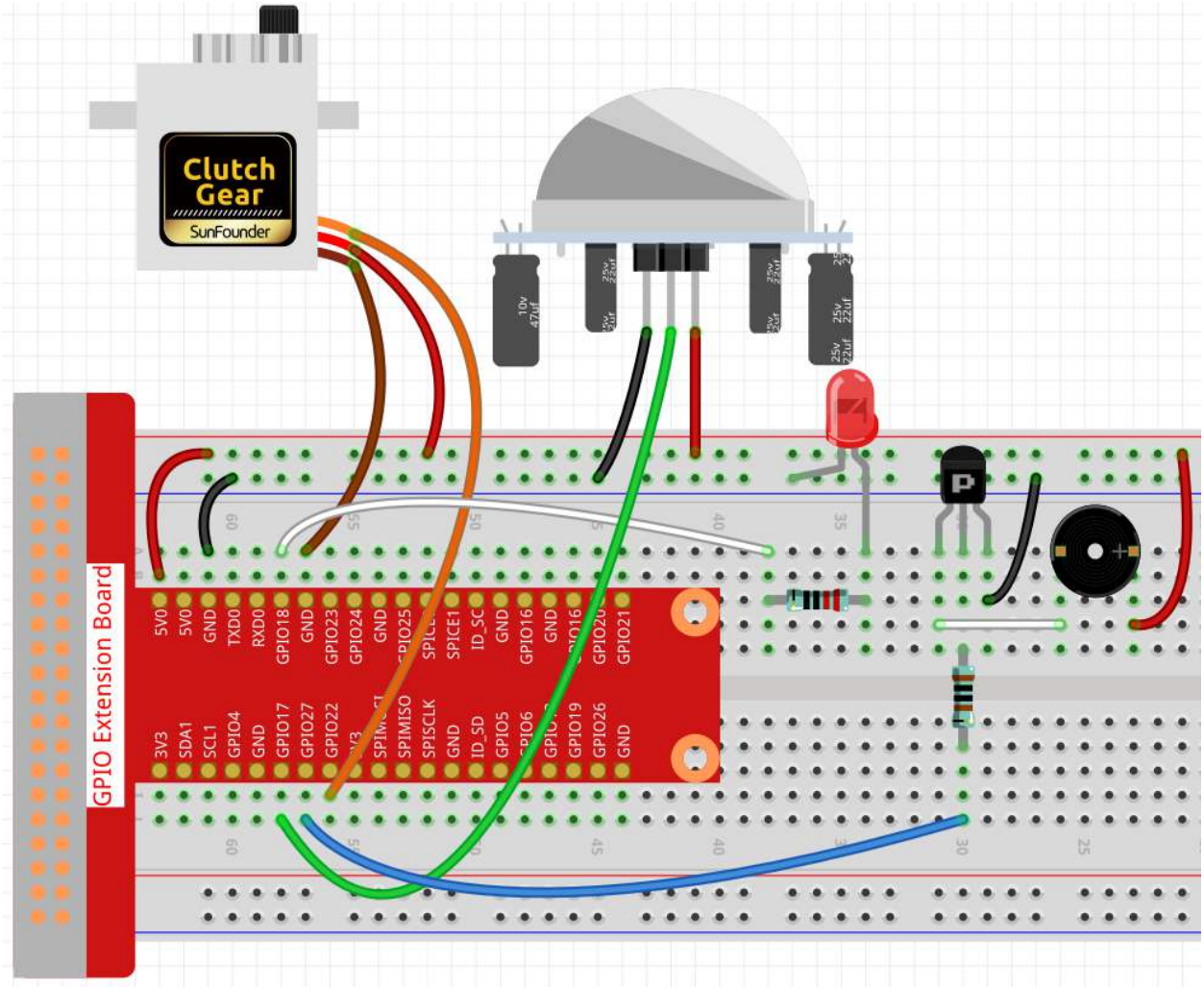
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.8_Welcome.py
```

After the code runs, if the PIR sensor detects someone passing by, the door will automatically open (simulated by the servo), turn on the indicator and play the doorbell music. After the doorbell music plays, the system will automatically close the door and turn off the indicator light, waiting for the next time someone passes 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

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import time

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

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


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

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

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

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

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


def setup():
    global p
    global Buzz                         # Assign a global variable to replace GPIO.PWM
    GPIO.setmode(GPIO.BCM)        # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT)    # Set ledPin's mode is output
```

(continues on next page)

(continued from previous page)

```
    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.HIGH)
    for i in range(0, 181, 1):   #make servo rotate from 0 to 180 deg
        setAngle(i)      # Write to servo
        time.sleep(0.001)
    time.sleep(1)
    doorbell()
    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()
```

(continues on next page)

(continued from previous page)

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

Code Explanation

```
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
```

These statements are used to initialize the pins of each component.

```
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
```

Create a function, servowrite to write the angle in the servo that is 0-180.

```
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
```

Create a function, doorbell to enable the buzzer to play music.

```
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)
```

Close the door and turn off the indicator light.

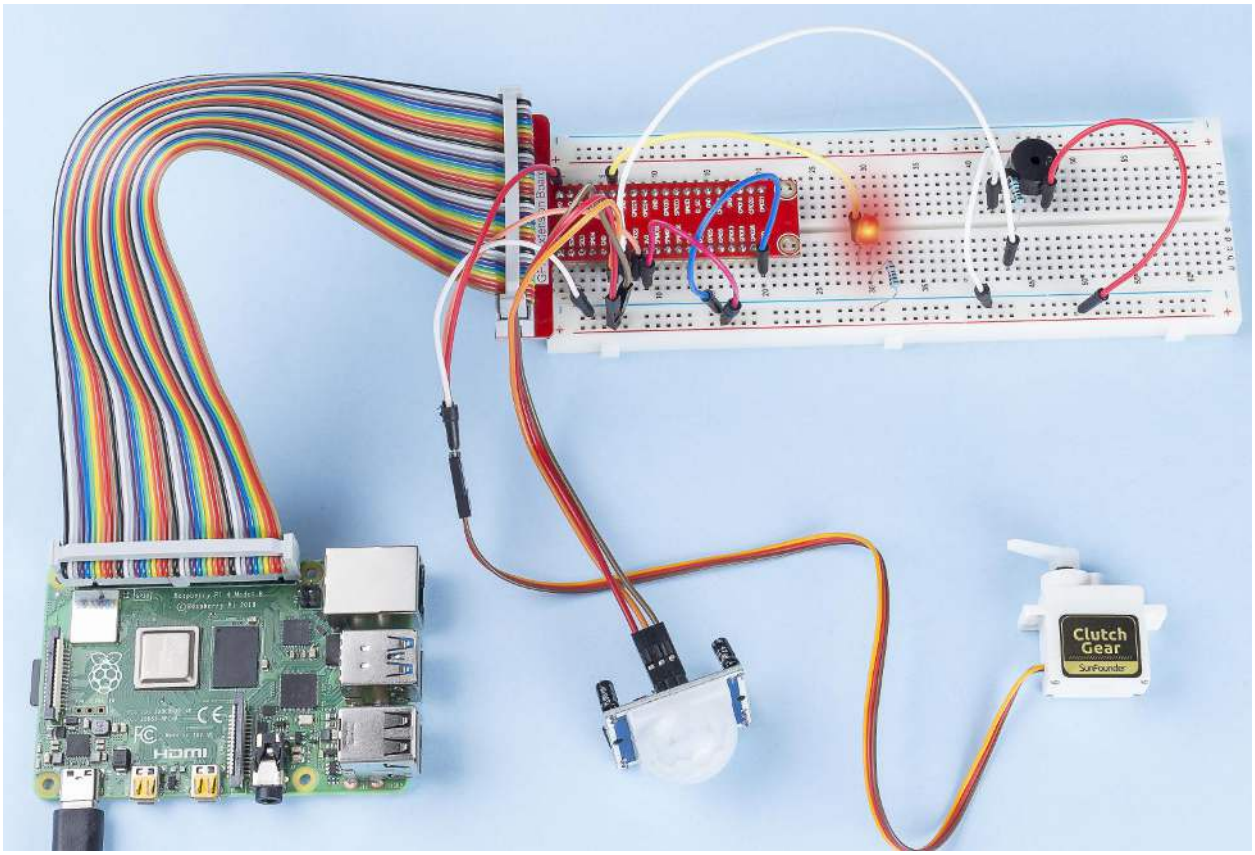
```
def opendoor():
    GPIO.output(ledPin, GPIO.HIGH)
    for i in range(0, 181, 1):    #make servo rotate from 0 to 180 deg
        setAngle(i)      # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()
```

The function, `opendoor()` consists of several parts: turn on the indicator light, turn the servo (to simulate the action of opening the door), play the doorbell music of the convenience store, and call the function , `closedoor()` after playing music.

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

When PIR senses that someone is passing by, it calls the function, `opendoor()` .

Phenomenon Picture

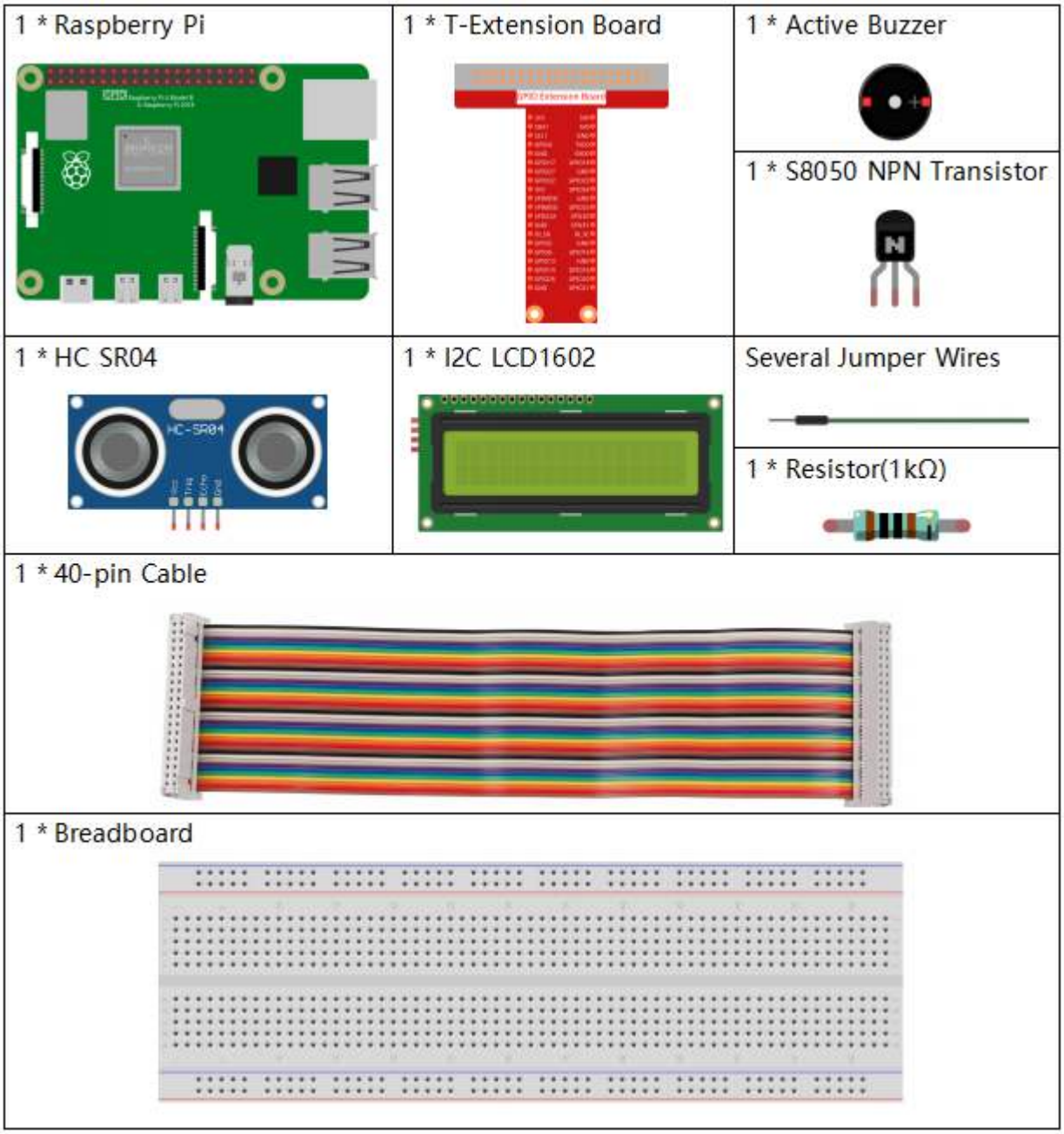


6.6.9 4.1.9 Reversing Alarm

Introduction

In this project, we will use LCD, buzzer and ultrasonic sensors to make a reverse assist system. We can put it on the remote control vehicle to simulate the actual process of reversing the car into the garage.

Components



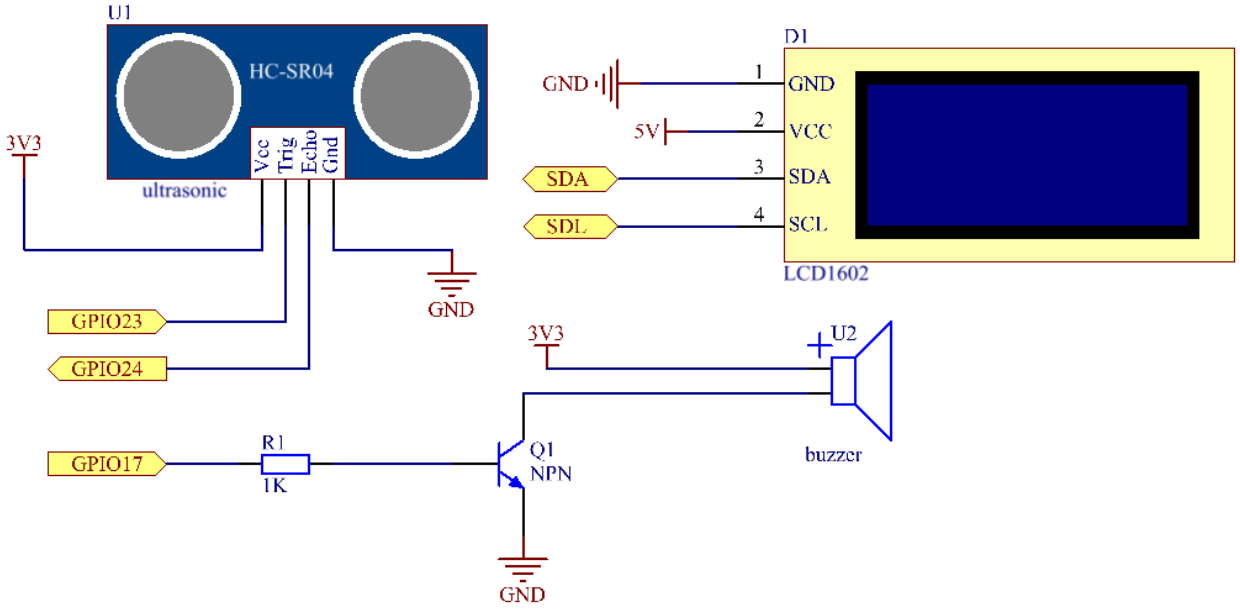
- *GPIO Extension Board*
- *Breadboard*

- *Resistor*
- *Buzzer*
- *Transistor*
- *Ultrasonic Module*
- *I2C LCD1602*

Schematic Diagram

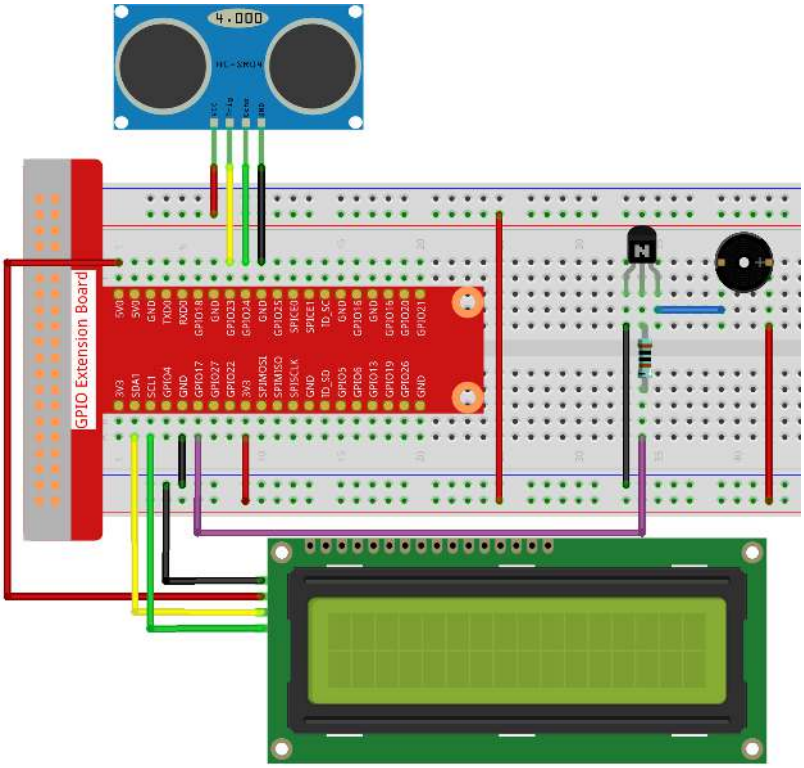
Ultrasonic sensor detects the distance between itself and the obstacle that will be displayed on the LCD in the form of code. At the same time, the ultrasonic sensor let the buzzer issue prompt sound of different frequency according to different distance value.

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.9_ReversingAlarm.py
```

As the code runs, ultrasonic sensor module detects the distance to the obstacle and then displays the information about the distance on LCD1602; besides, buzzer emits warning tone whose frequency changes with the distance.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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')
```

(continues on next page)

(continued from previous page)

```
time.sleep(2)

def setup():
GPIO.setmode(GPIO.BOARD)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
lcdsetup()

def distance():
GPIO.output(TRIG, 0)
time.sleep(0.000002)

GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def destroy():
GPIO.output(BUZZER, GPIO.LOW)
GPIO.cleanup()
LCD1602.clear()

def loop():
while True:
    dis = distance()
    print (dis, 'cm')
    print ('')
    GPIO.output(BUZZER, GPIO.LOW)
    if (dis > 400):
        LCD1602.clear()
        LCD1602.write(0, 0, 'Error')
        LCD1602.write(3, 1, 'Out of range')
        time.sleep(0.5)
    else:
        LCD1602.clear()
        LCD1602.write(0, 0, 'Distance is')
        LCD1602.write(5, 1, str(round(dis,2)) +' cm')
        if(dis>=50):
            time.sleep(0.5)
        elif(dis<50 and dis>20):
            for i in range(0,2,1):
                GPIO.output(BUZZER, GPIO.HIGH)
                time.sleep(0.05)
                GPIO.output(BUZZER, GPIO.LOW)
                time.sleep(0.2)
        elif(dis<=20):
            for i in range(0,5,1):
```

(continues on next page)

(continued from previous page)

```
                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 Explanation

```
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 this program, we apply the previously used components synthetically. Here we use buzzers, LCD and ultrasonic. We can initialize them in the same way as we did before.

```
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')
```

Here we get the values of the ultrasonic sensor and get the distance through calculation. If the value of distance is greater than the range of value to be detected, an error message is printed on the LCD. And if the distance is within the working range, the corresponding results will be output.

```
LCD1602.write(5, 1, str(round(dis,2)) +' cm')
```

Since the LCD output only supports character types, we need to use **str ()** to convert numeric values to characters. We are going to round it to two decimal places.

```
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)
```

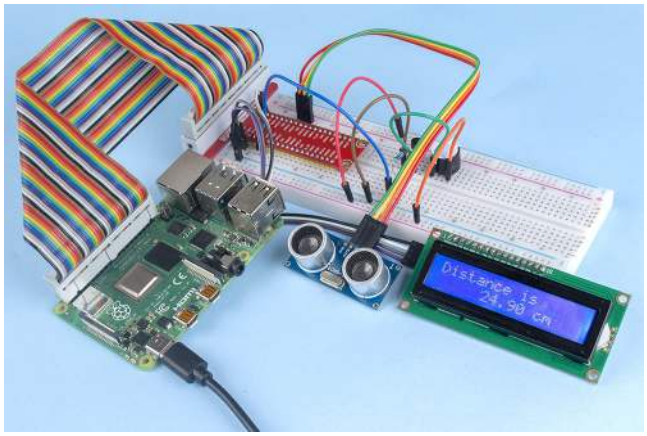
(continues on next page)

(continued from previous page)

```
        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)
```

This judgment condition is used to control the sound of the buzzer. According to the difference in distance, it can be divided into three cases, in which there will be different sound frequencies. Since the total value of delay is 500, all of them can provide a 500ms interval for the ultrasonic sensor to work.

Phenomenon Picture

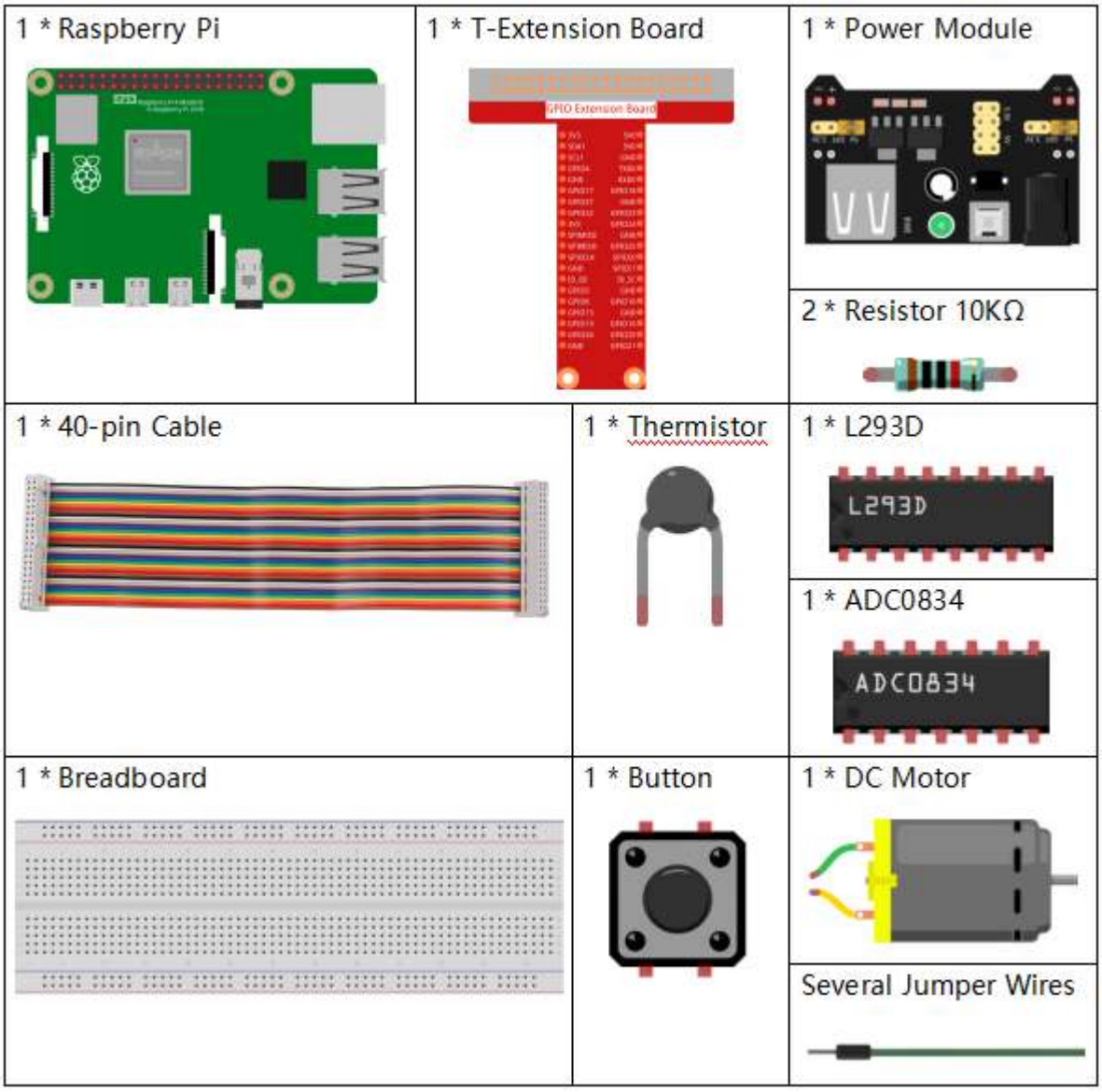


6.6.10 4.1.10 Smart Fan

Introduction

In this project, we will use motors, buttons and thermistors to make a manual + automatic smart fan whose wind speed is adjustable.

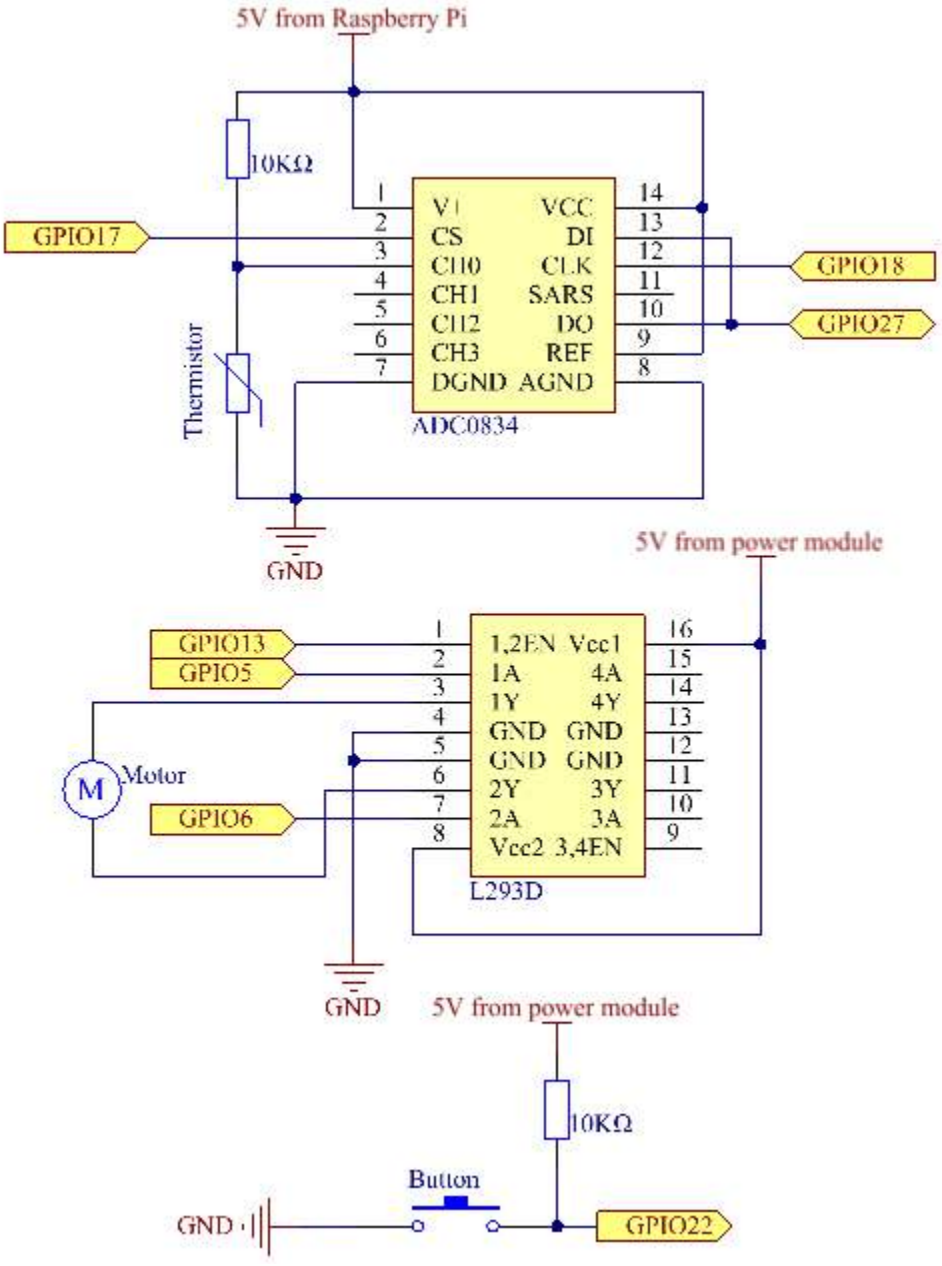
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Power Supply Module*
- *Thermistor*
- *L293D*
- *ADC0834*
- *Button*
- *DC Motor*

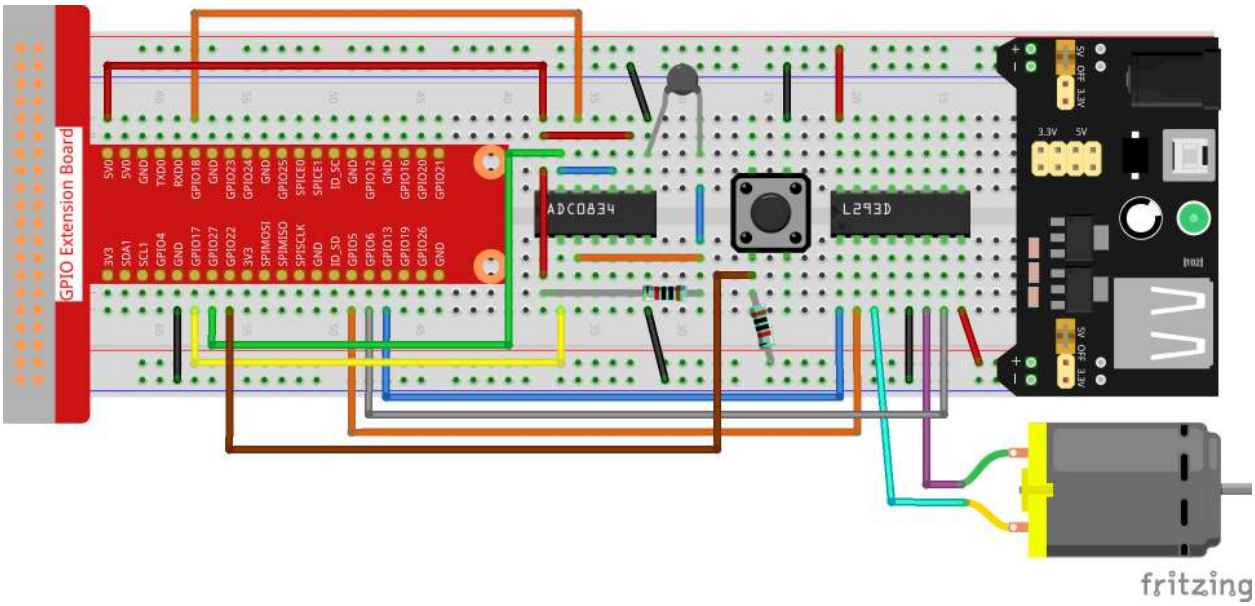
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13



Experimental Procedures

Step 1: Build the circuit.



Note: 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.

Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 4.1.10_SmartFan.py
```

As the code runs, start the fan by pressing the button. Every time you press, 1 speed grade is adjusted up or down. There are **5** kinds of speed grades: **0~4**. When set to the 4th speed grade and you press the button, the fan stops working with a **0** wind speed.

Once the temperature goes up or down for more than 2°C, the speed automatically gets 1-grade faster or slower.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

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


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

def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return Cel

def motor(level):
    if level == 0:
        GPIO.output(MotorEnable, GPIO.LOW)
        return 0
    if level>=4:
        level = 4
    GPIO.output(MotorEnable, GPIO.HIGH)
    p_M1.ChangeDutyCycle(level*25)
    return level


def main():
    lastState=0
    level=0
    markTemp = temperature()
    while True:
        currentState =GPIO.input(BtnPin)
```

(continues on next page)

(continued from previous page)

```
        currentTemp=temperature()
        if currentState == 1 and lastState == 0:
            level=(level+1)%5
            markTemp = currentTemp
            time.sleep(0.5)
        lastState=currentState
        if level!=0:
            if currentTemp-markTemp <= -2:
                level = level -1
                markTemp=currentTemp
            if currentTemp-markTemp >= 2:
                level = level +1
                markTemp=currentTemp
        level = motor(level)


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

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

Code Explanation

```
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
```

`temperture()` works by converting thermistor values read by **ADC0834** into temperature values. Refer to *2.2.2 Thermistor* for more 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
```

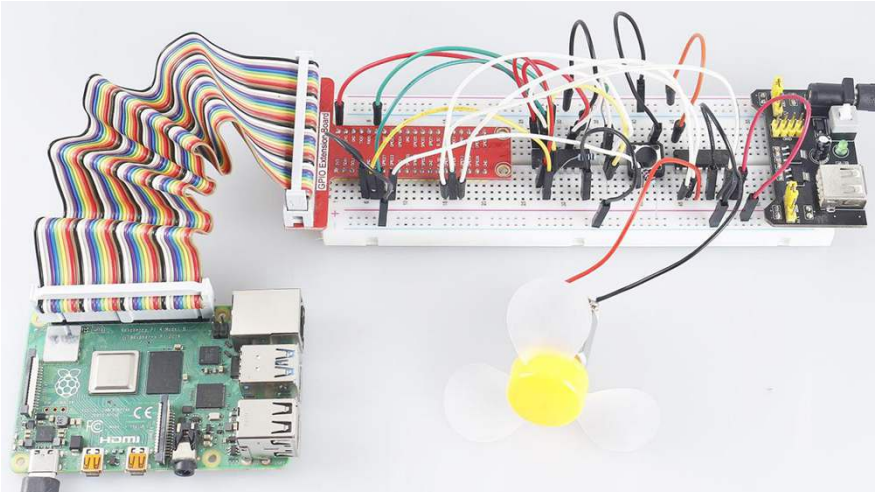
This function controls the rotating speed of the motor. The range of the **Lever**: **0-4** (level **0** stops the working motor). One level adjustment stands for a **25%** change of the wind speed.

```
def main():
    lastState=0
    level=0
    markTemp = temperature()
    while True:
        currentState =GPIO.input(BtnPin)
        currentTemp=temperature()
        if currentState == 1 and lastState == 0:
            level=(level+1)%5
            markTemp = currentTemp
            time.sleep(0.5)
        lastState=currentState
        if level!=0:
            if currentTemp-markTemp <= -2:
                level = level -1
                markTemp=currentTemp
            if currentTemp-markTemp >= 2:
                level = level +1
                markTemp=currentTemp
        level = motor(level)
```

The function **main()** contains the whole program process as shown:

1) Constantly read the button state and the current temperature.
2) Every press makes level**+1** and at the same time, the temperature is updated. The **Level** ranges **1~4**.
3) As the fan works (the level is **not 0**), the temperature is under detection. A **2°C+** change causes the up and down of the level.
4) The motor changes the rotating speed with the **Level**.

Phenomenon Picture

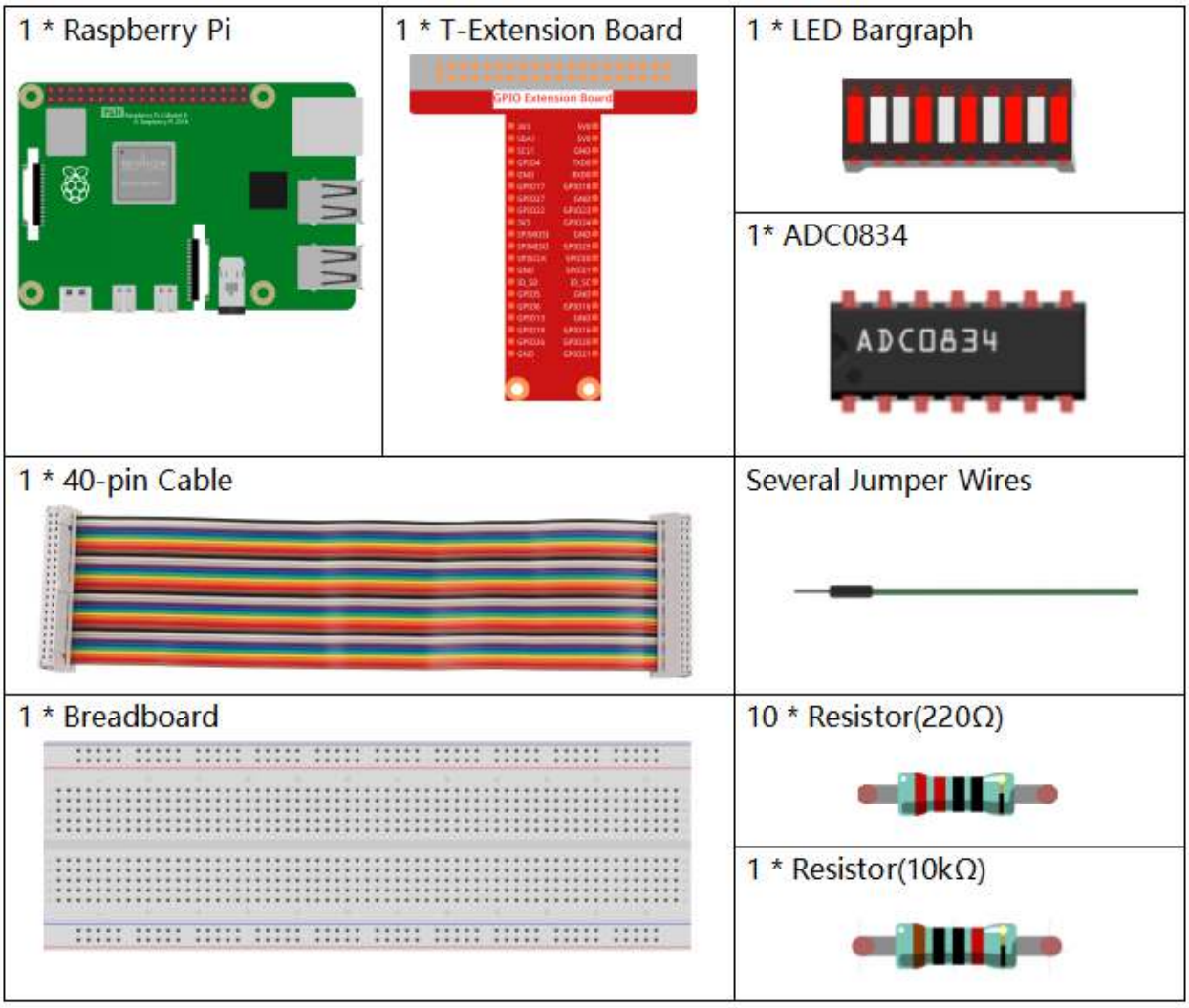


6.6.11 4.1.11 Battery Indicator

Introduction

In this project, we will make a battery indicator device that can visually display the battery level on the LED Bargraph.

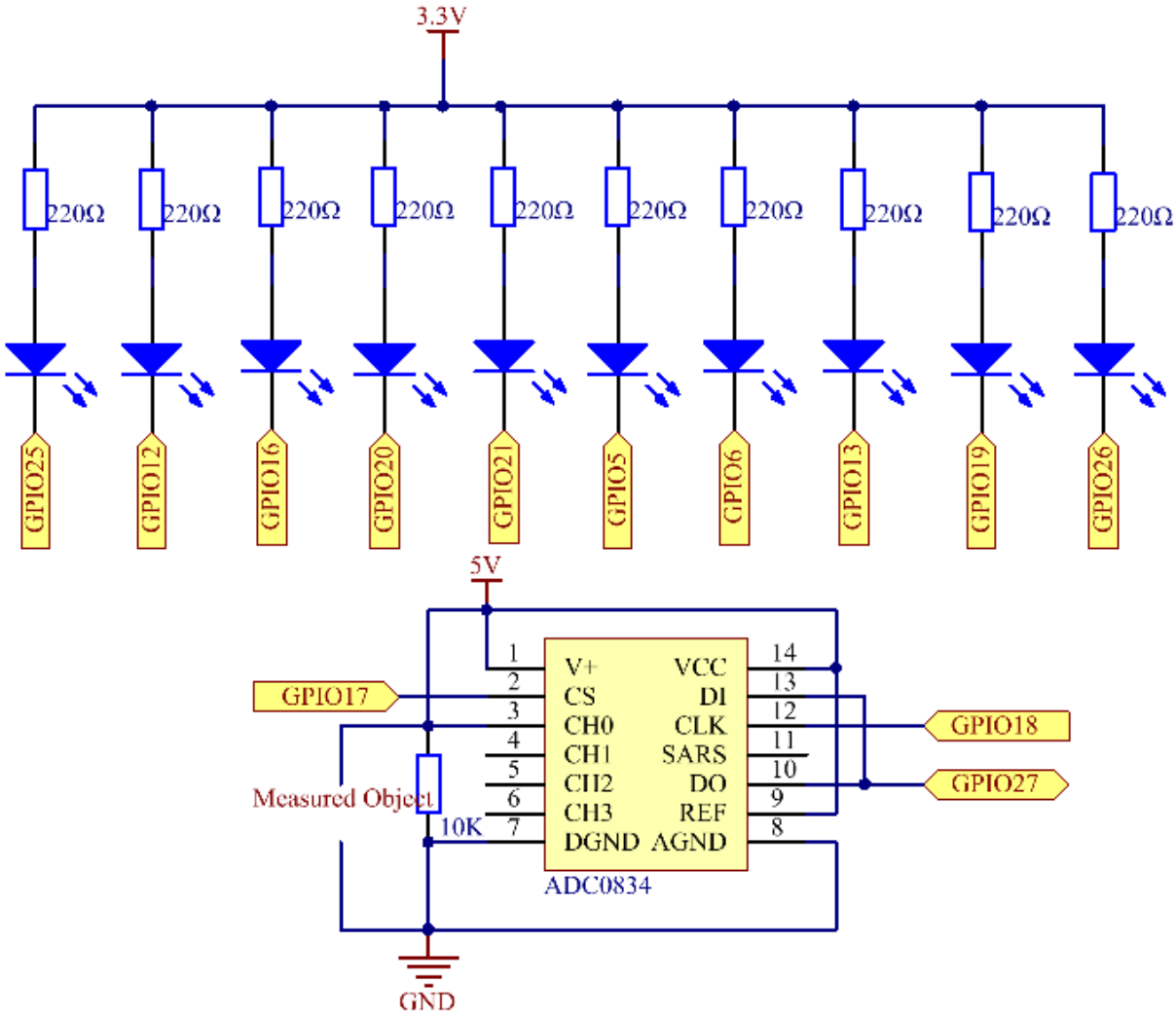
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*
- *ADC0834*

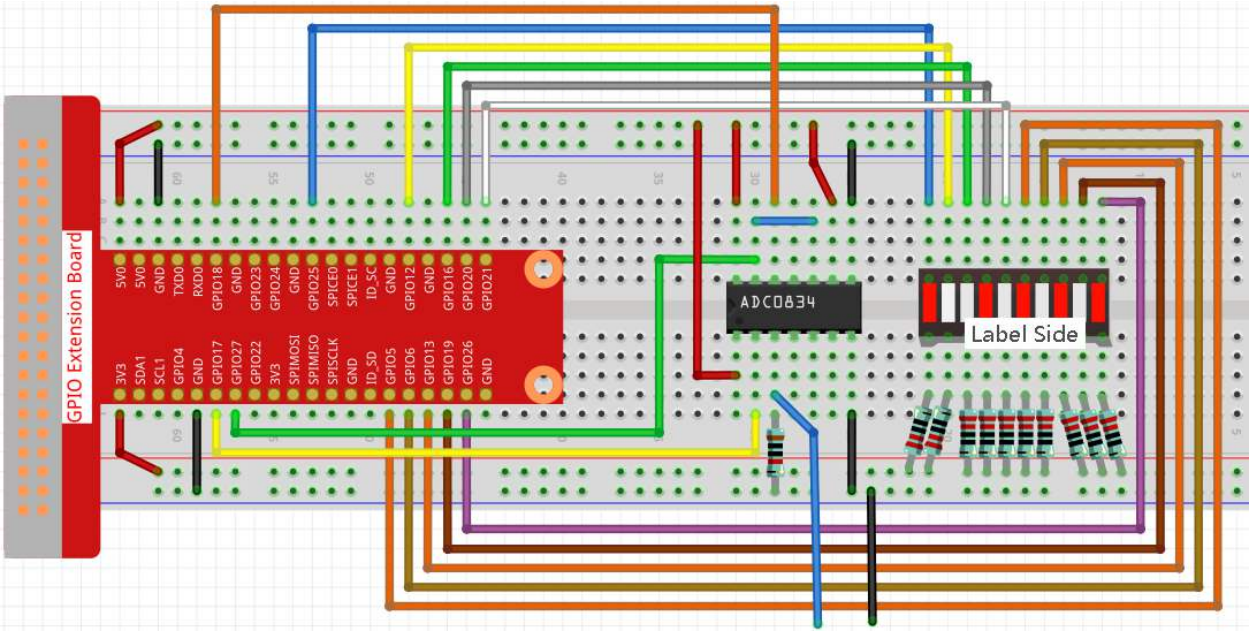
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO25	Pin 22	6	25
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	20
GPIO21	Pin 40	29	21
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13
GPIO19	Pin 35	24	19
GPIO26	Pin 37	25	26



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 4.1.11_BatteryIndicator.py
```

After the program runs, give the 3rd pin of ADC0834 and the GND a lead-out wire separately and then lead them to the two poles of a battery separately. You can see the corresponding LED on the LED Bargraph is lit up to display the power level (measuring range: 0-5V).

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
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)
```

(continues on next page)

(continued from previous page)

```
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 Explanation

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

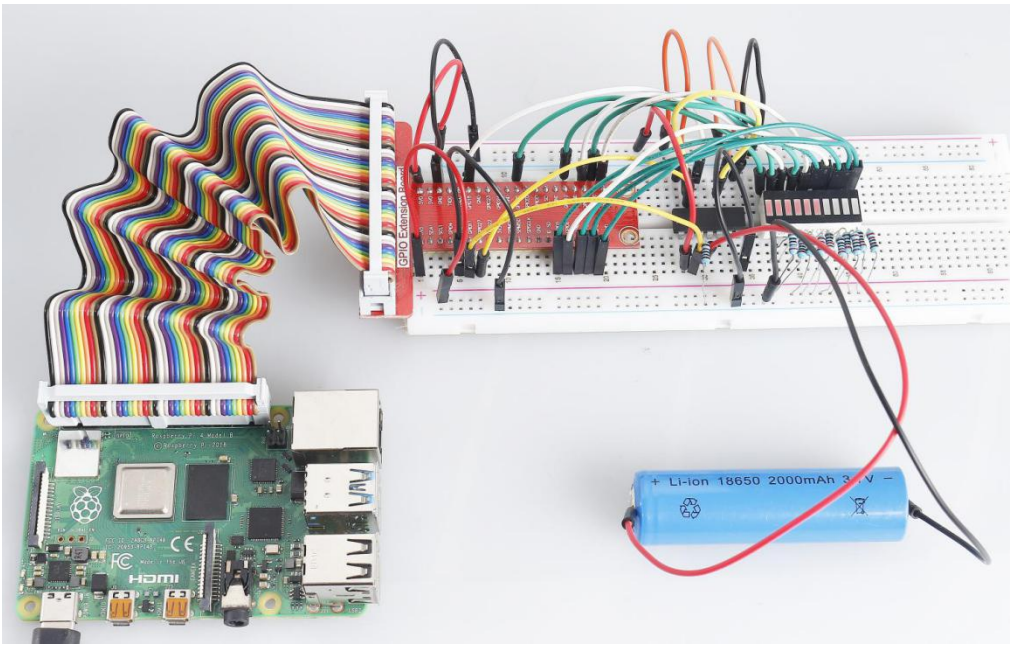
This function works for controlling the turning on or off of the **10** LEDs on the LED Bargraph. We give these **10** LEDs high levels to let they are **off** at first, then decide how many LEDs are lit up by changing the received analog value.

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

analogVal produces values (**0-255**) with varying voltage values (**0-5V**), ex., if a 3V is detected on a battery, the corresponding value **152** is displayed on the voltmeter.

The **10** LEDs on the LED Bargraph are used to display the **analogVal** readings. 255/10=25, so every **25** the analog value increases, one more LED turns on, ex., if "analogVal=150 (about 3V), there are 6 LEDs turning on."

Phenomenon Picture

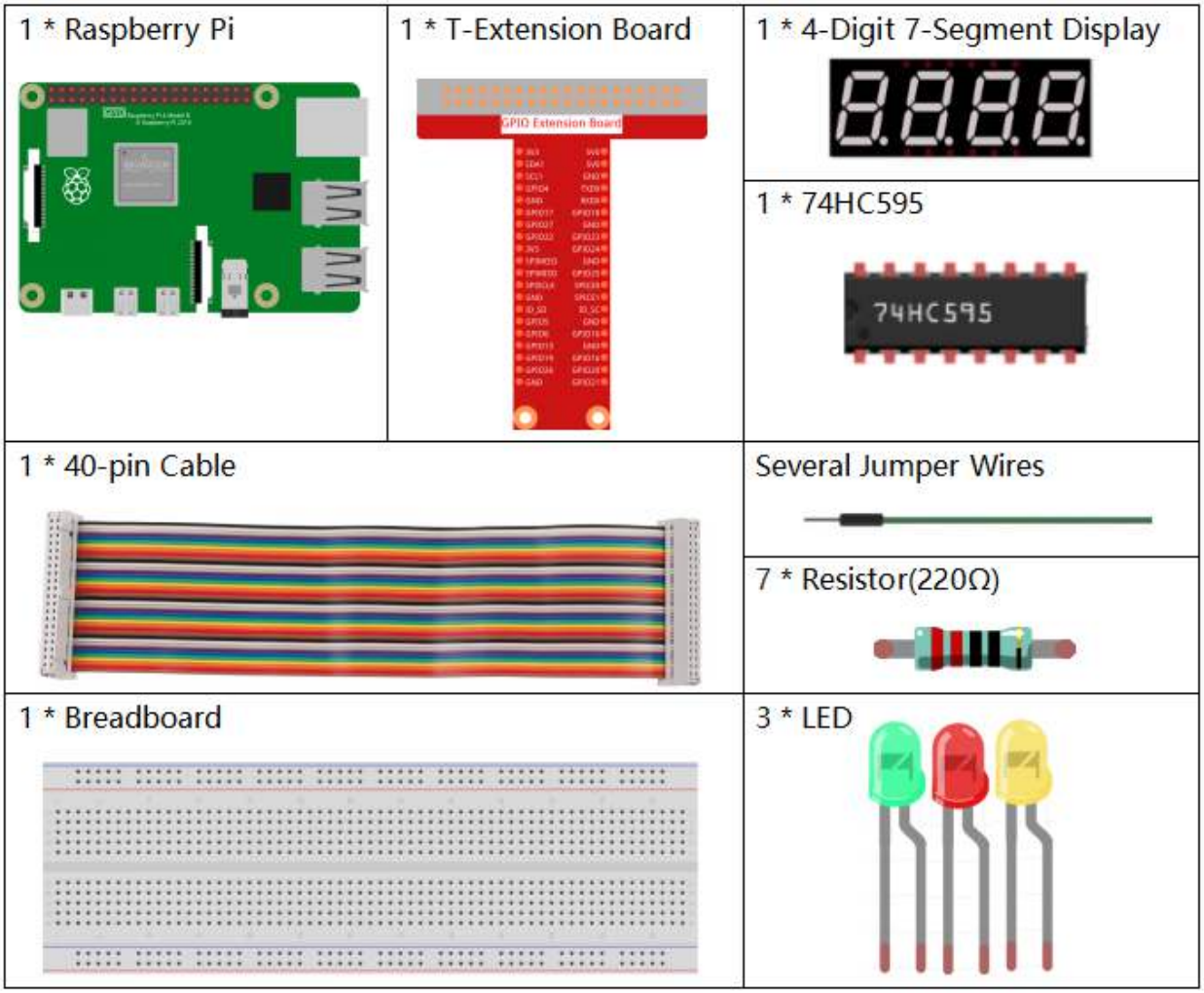


6.6.12 4.1.12 Traffic Light

Introduction

In this project, we will use LED lights of three colors to realize the change of traffic lights and a four-digit 7-segment display will be used to display the timing of each traffic state.

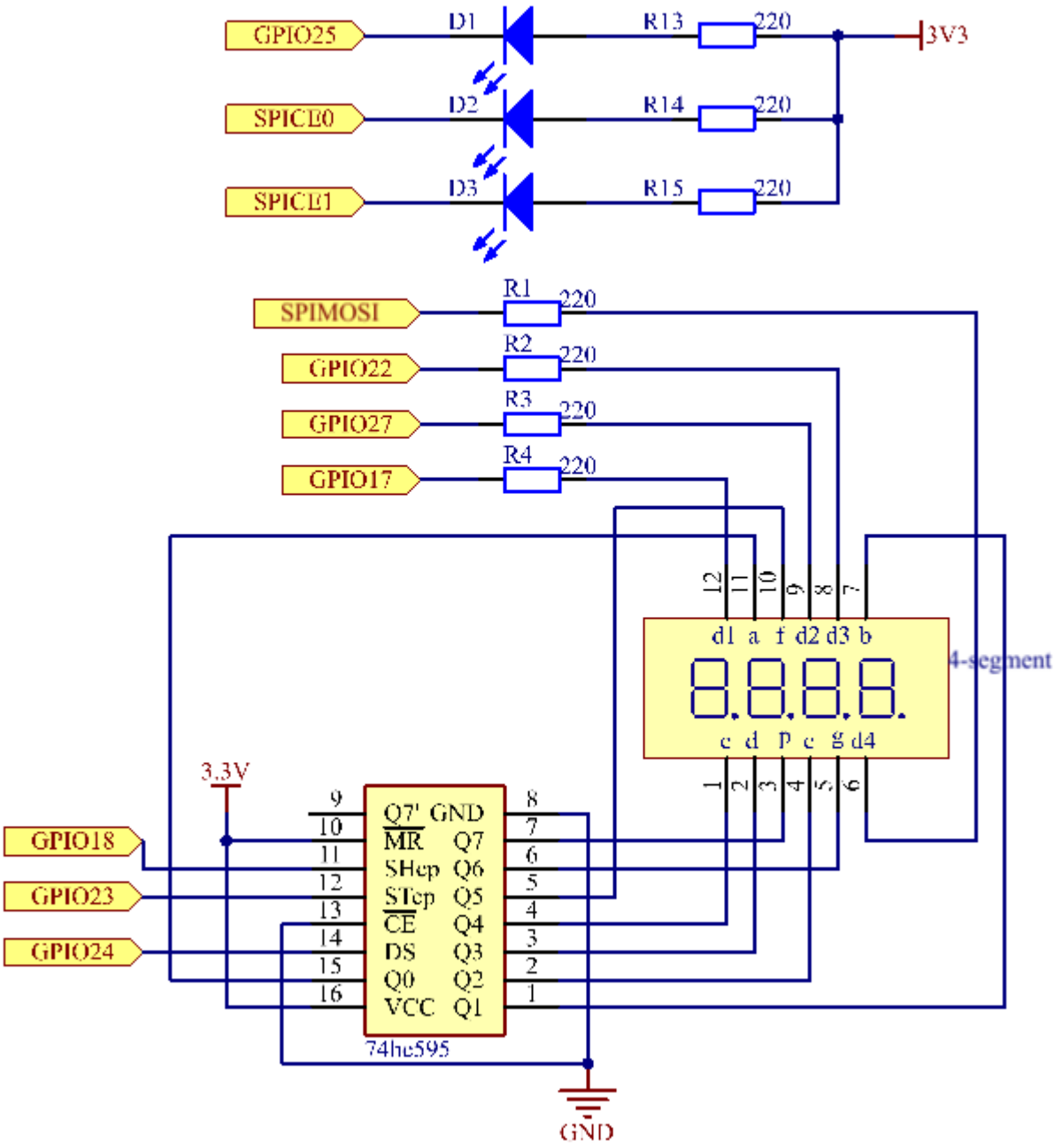
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *4-Digit 7-Segment Display*
- *74HC595*

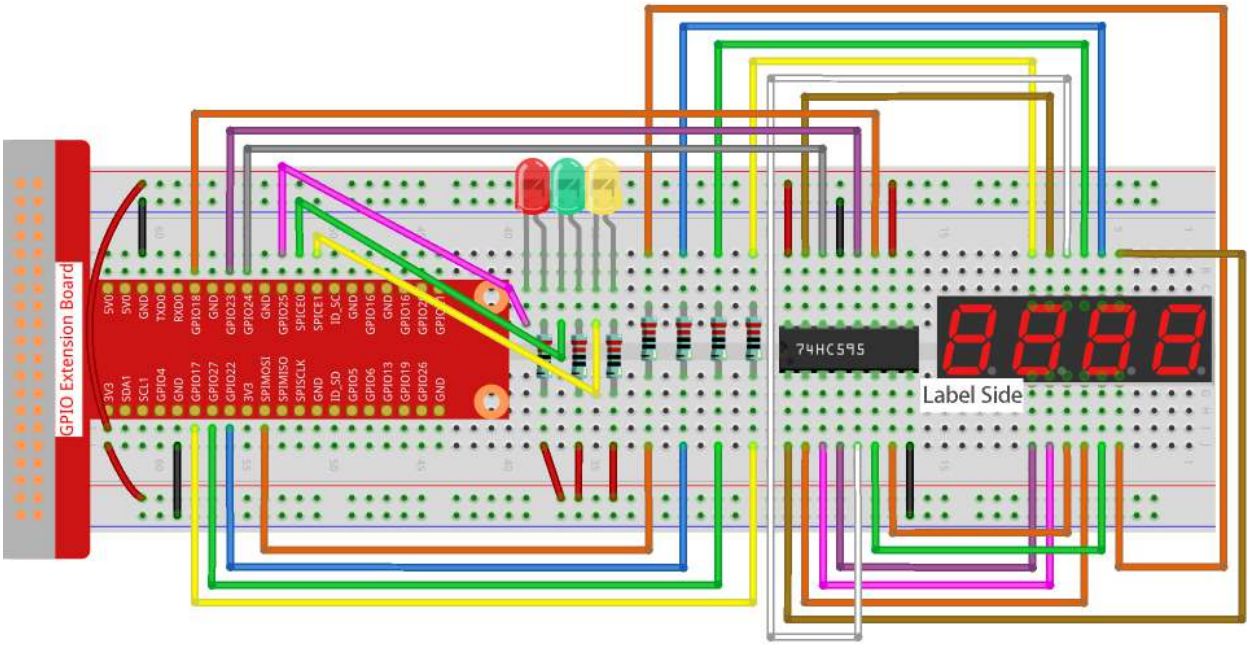
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	Pin 26	11	7


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

Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.12_TrafficLight.py
```

As the code runs, LEDs will simulate the color changing of traffic lights. Firstly, the red LED lights up for 60s, then the green LED lights up for 30s; next, the yellow LED lights up for 5s. After that, the red LED lights up for 60s once again. In this way, this series of actions will be executed repeatedly. Meanwhile, the 4-digit 7-segment display displays the countdown time continuously.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

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)
```

(continues on next page)

(continued from previous page)

```
SRCLK = 18      #shift register clock input(SHCP)
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)

placePin = (10,22,27,17)
ledPin =(25,8,7)

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

colorState=0
counter = 60
timer1 = 0


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

(continues on next page)

(continued from previous page)

```
    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]))

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()

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

(continues on next page)

(continued from previous page)

```
    global timer1
    GPIO.cleanup()
    timer1.cancel()      #cancel the timer

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

Code Explanation

```
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():
```

These codes are used to realize the function of number display of 4-Digit 7-Segment. Refer to chapter 1.1.5 of the document for more details. Here, we use the codes to display countdown of traffic light time.

```
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)
```

The codes are used to switch the LED on and off.

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

colorState=0
counter = 60
timer1 = 0

def timer():        #timer function
    global counter
    global colorState
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()
    counter-=1
    if (counter is 0):
```

(continues on next page)

(continued from previous page)

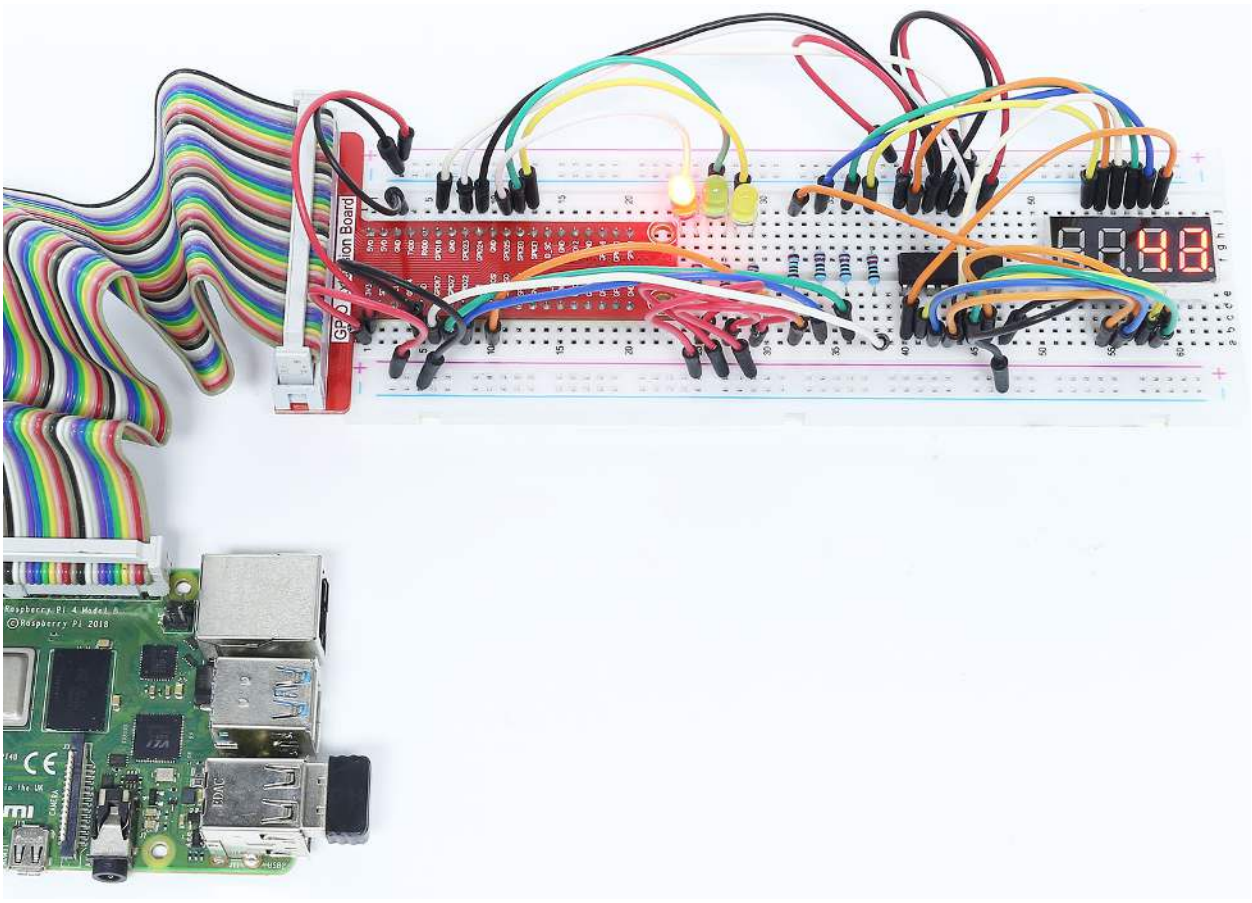
```
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
        colorState=(colorState+1)%3
    print ("counter : %d    color: %s "%(counter,lightColor[colorState]))
```

The codes are used to switch the timer on and off. Refer to chapter 1.1.5 for more details. Here, when the timer returns to zero, colorState will be switched so as to switch LED, and the timer will be assigned to a new value.

```
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()
```

In `setup()` function, start the timer. In `loop()` function, a **while True** is used: call the relative functions of 4-Digit 7-Segment and LED circularly.

Phenomenon Picture

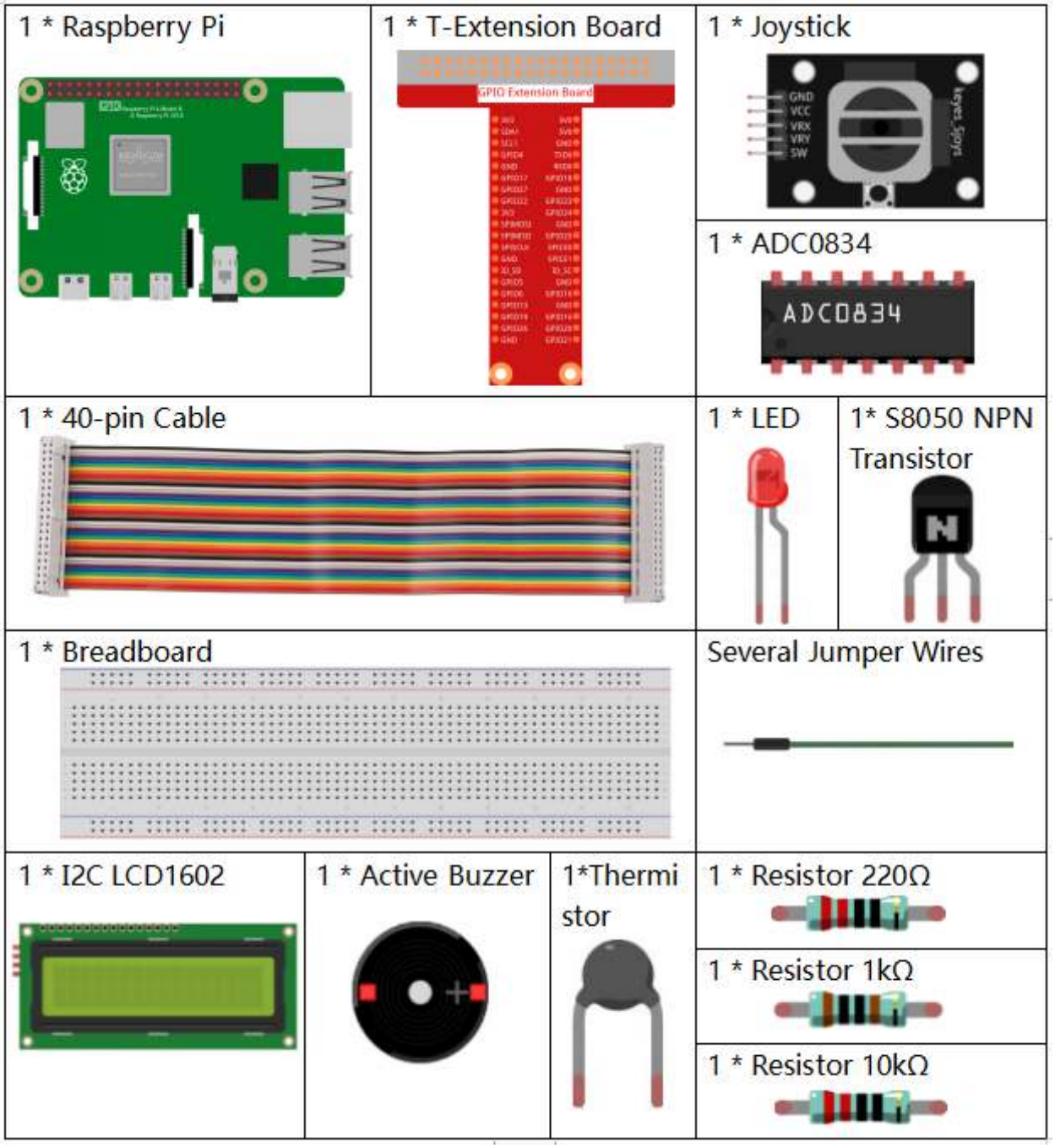


6.6.13 4.1.13 Overheat Monitor

Introduction

You may want to make an overheat monitoring device that applies to various situations, ex., in the factory, if we want to have an alarm and the timely automatic turning off of the machine when there is a circuit overheating. In this project, we will use thermistor, joystick, buzzer, LED and LCD to make an smart temperature monitoring device whose threshold is adjustable.

Components

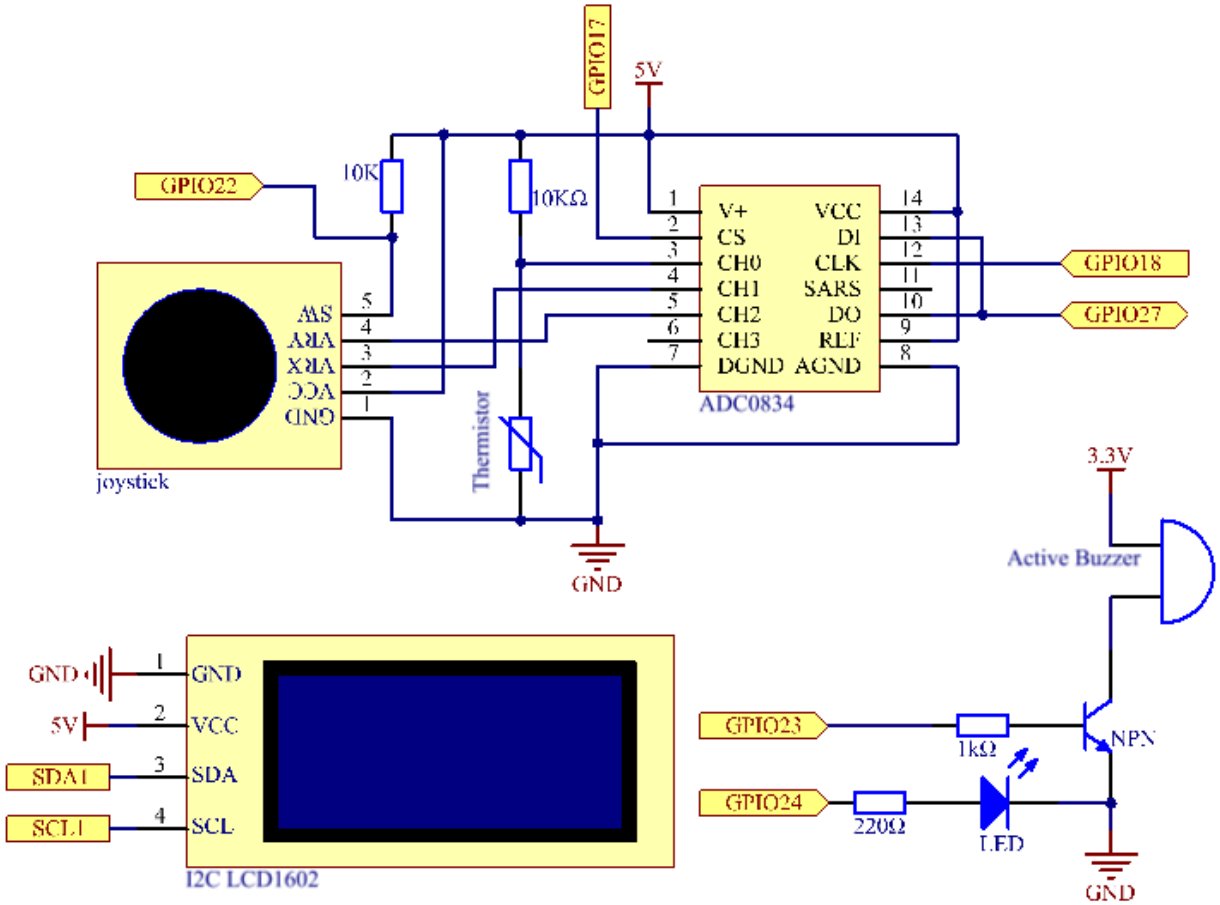


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Joystick Module*
- *ADC0834*

- *Transistor*

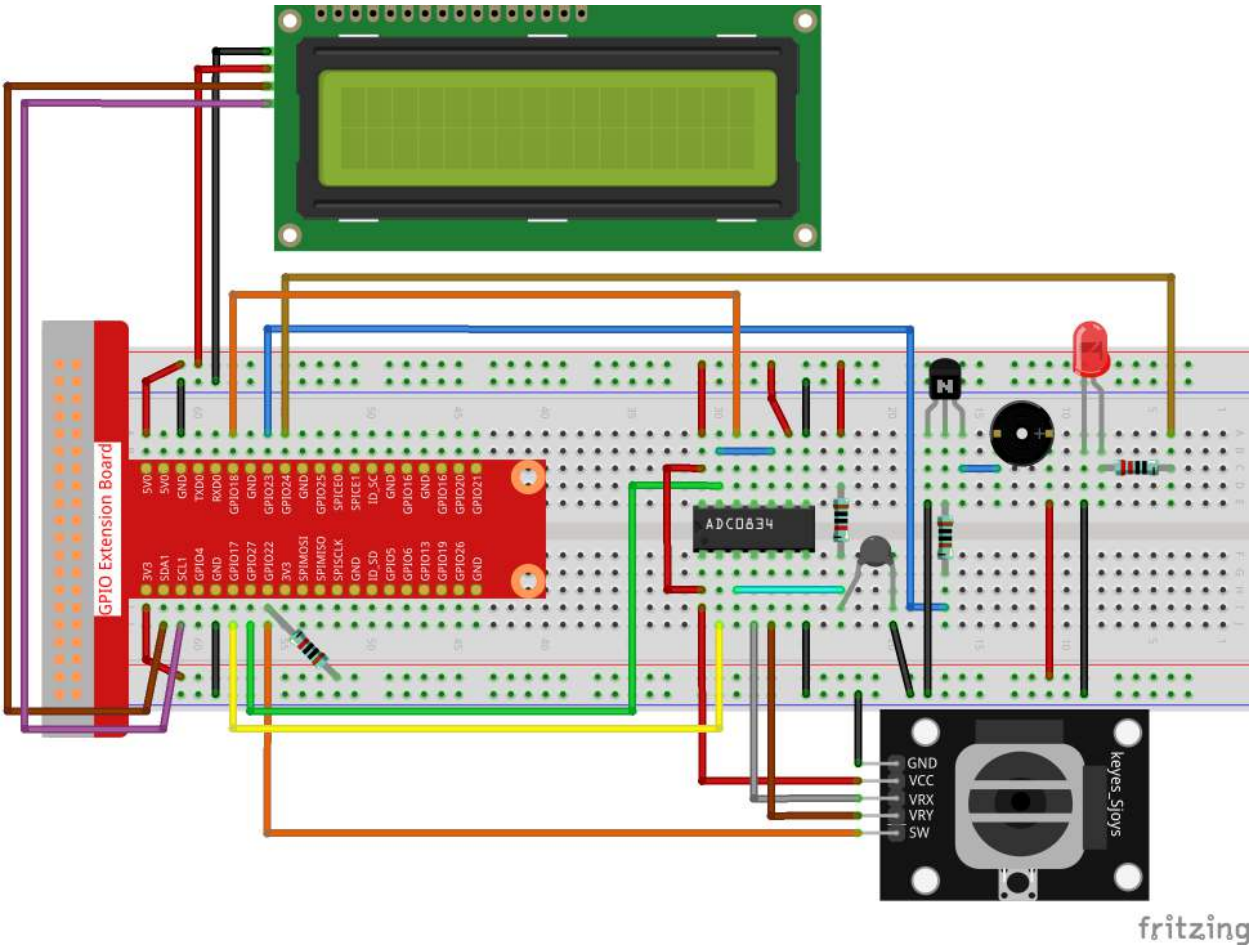
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22
GPIO23	Pin16	4	23
GPIO24	Pin18	5	24
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 4.1.13_OverheatMonitor.py
```

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

Joystick here is for your pressing to adjust the high-temperature threshold. Toggling the **Joystick** in the direction of X-axis and Y-axis can adjust (turn up or down) the current high-temperature threshold. Press the **Joystick** once again to reset the threshold to initial value.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

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)
    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
```

(continues on next page)

(continued from previous page)

```
    Cel=temperature()
    LCD1602.write(0, 0, 'Temp: ')
    LCD1602.write(0, 1, 'Upper: ')
    LCD1602.write(6, 0, str(Cel))
    LCD1602.write(7, 1, str(upperTem))
    time.sleep(0.1)
    if Cel >= upperTem:
        GPIO.output(buzzPin, GPIO.HIGH)
        GPIO.output(ledPin, GPIO.HIGH)
    else:
        GPIO.output(buzzPin, GPIO.LOW)
        GPIO.output(ledPin, GPIO.LOW)

def loop():
    lastState=1
    stage=0
    while True:
        currentState=GPIO.input(Joy_BtnPin)
        if currentState==1 and lastState ==0:
            stage=(stage+1)%2
            time.sleep(0.1)
            LCD1602.clear()
        lastState=currentState
        if stage == 1:
            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 Explanation

```
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
```

This function reads values of X and Y. If **X>200**, there will return "**1**"; **X<50**, return "**-1**"; **y>200**, return "**-10**", and **y<50**, return "**10**".

```
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)
```

This function is for adjusting the threshold and displaying it on the 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)
```

Read the analog value of the **CH0** (thermistor) of **ADC0834** and then convert it to temperature value.

```
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)
```

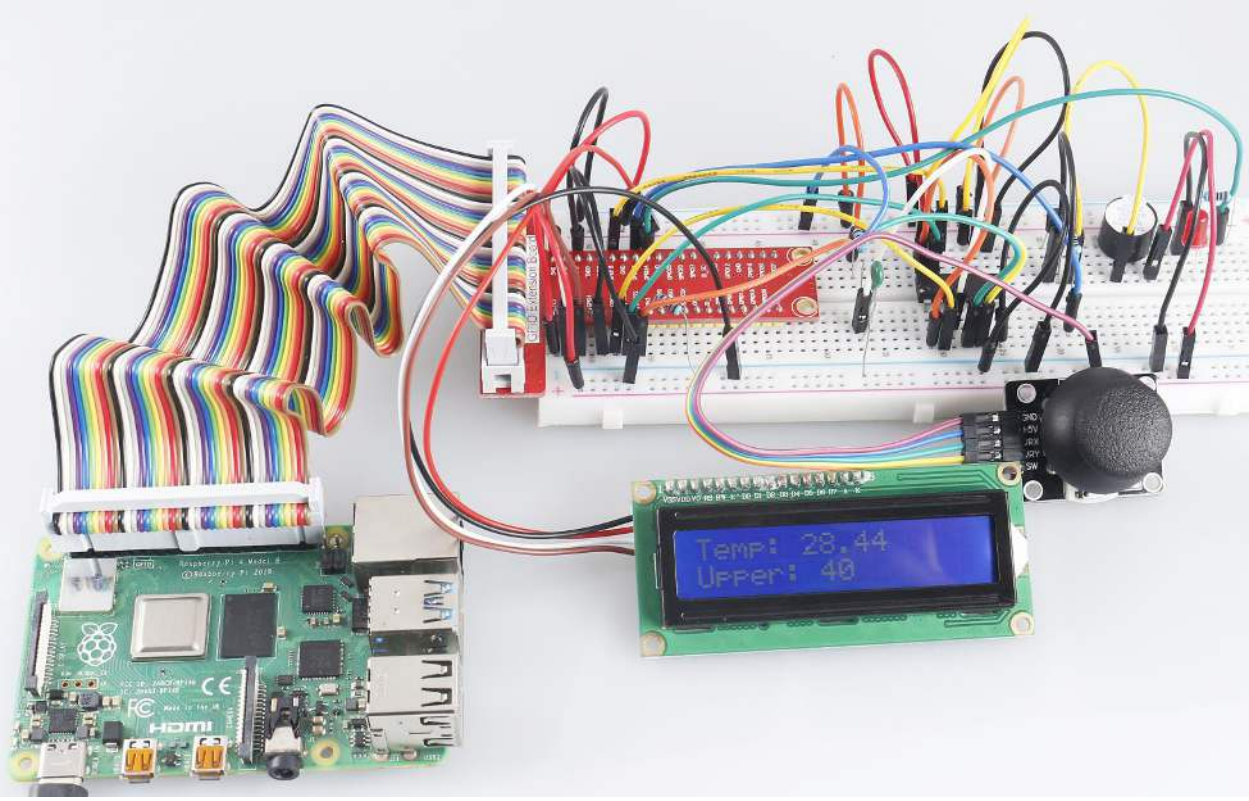
As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

```
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()
```

The function `main()` contains the whole program process as shown:

1) When the program starts, the initial value of **stage** is **0**, and the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and the LED are started to alarm you.
2) Press the Joystick, and **stage** will be **1** and you can adjust the high-temperature threshold. Toggling the Joystick in the direction of X-axis and Y-axis can adjust (turn up or down) the current high-temperature threshold. Press the Joystick once again to reset the threshold to initial value.

Phenomenon Picture



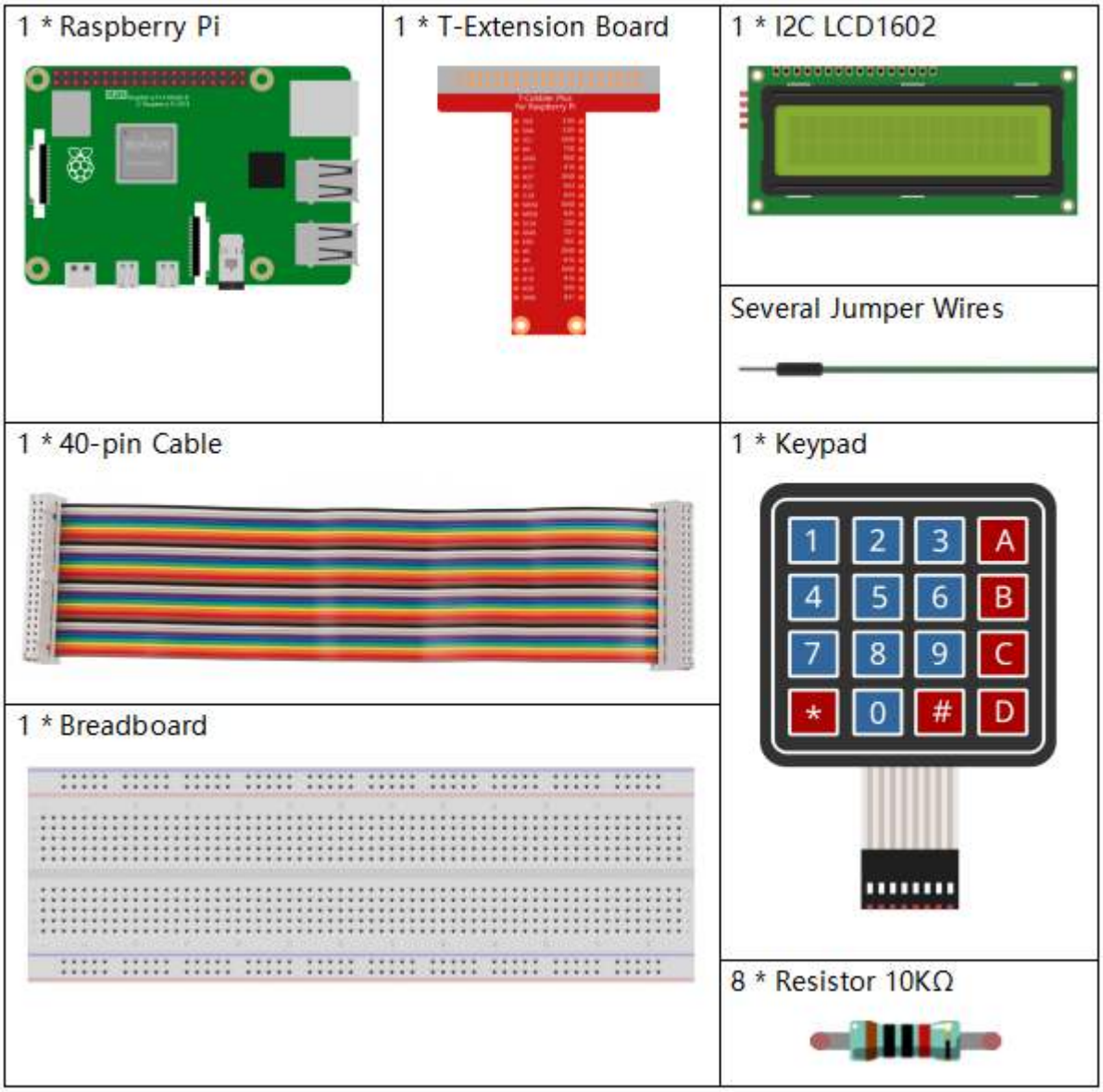
6.6.14 4.1.14 Password Lock

Introduction

In this project, we will use a keypad and a LCD to make a combination lock. The LCD will display a corresponding prompt for you to type your password on the Keypad. If the password is input correctly, “Correct” will be displayed.

On the basis of this project, we can add additional electronic components, such as buzzer, LED and so on, to add different experimental phenomena for password input.

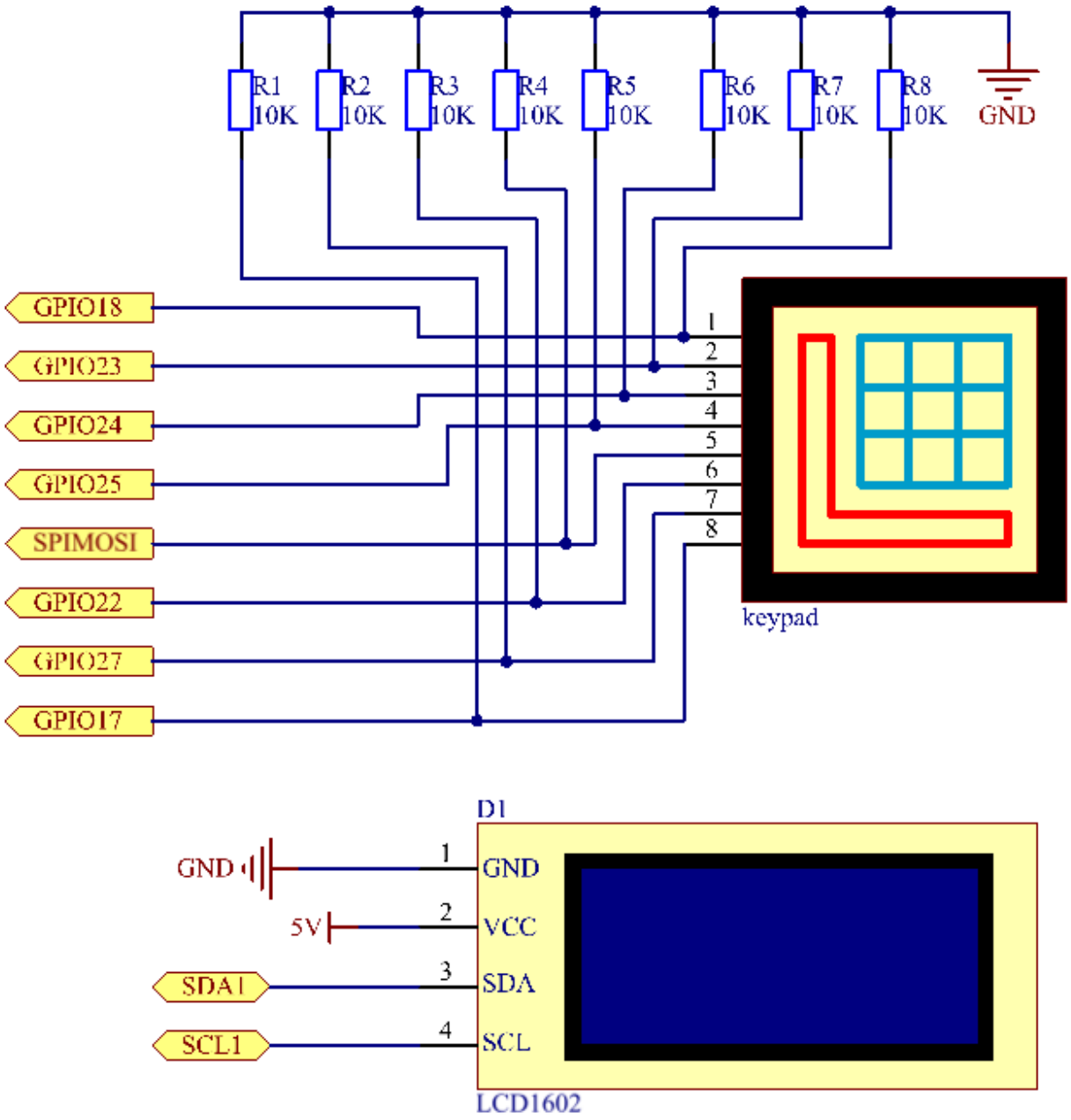
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *I2C LCD1602*
- *Keypad*

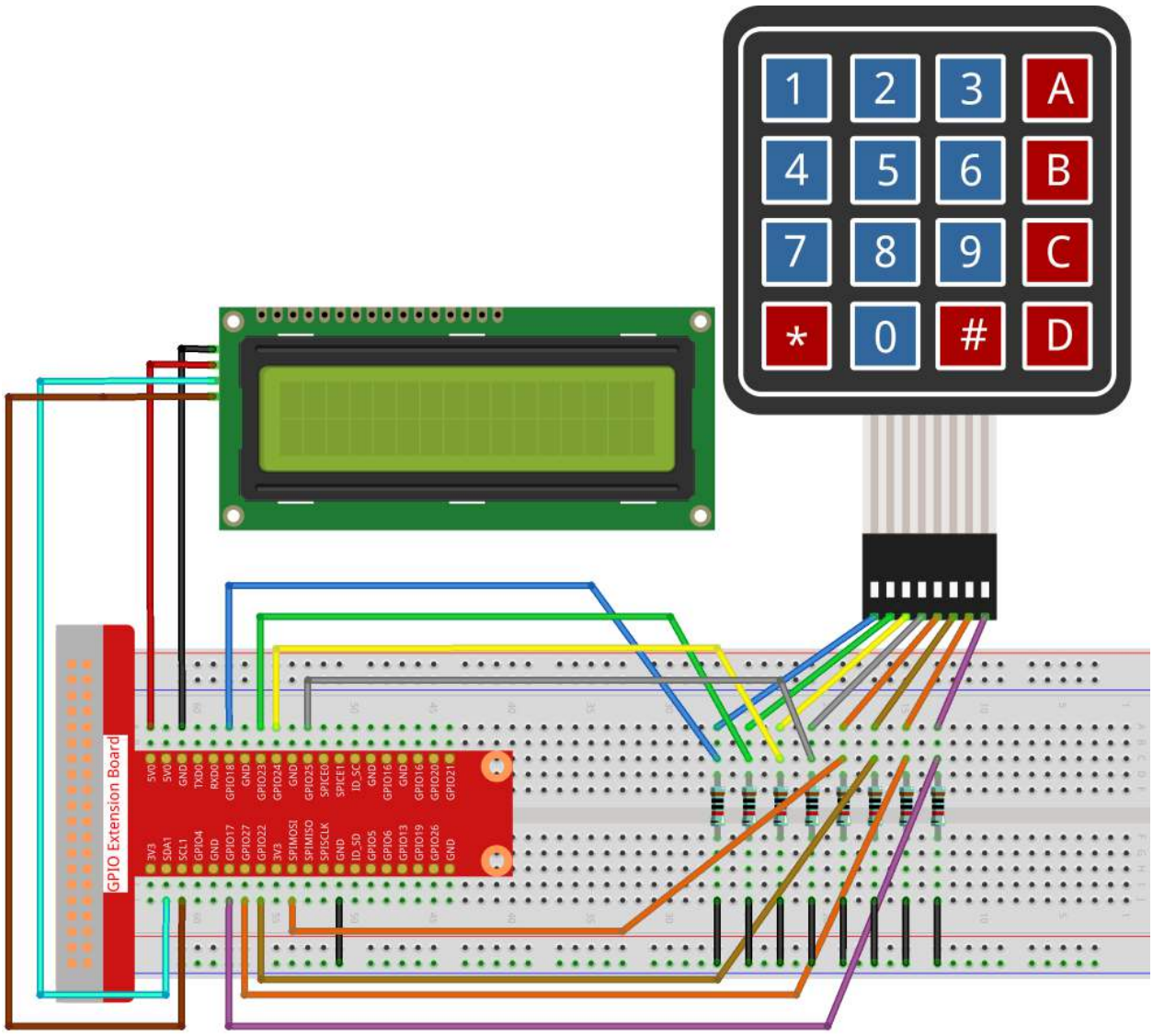
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.14_PasswordLock.py
```

After the code runs, keypad is used to input password: 1984. If the "CORRECT" appears on LCD1602, there is no wrong with the password; otherwise, "WRONG KEY" will appear.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

import RPi.GPIO as GPIO
import time
import LCD1602

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

    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys

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

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')
```

(continues on next page)

(continued from previous page)

```
    time.sleep(2)

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

def loop():
    global keyIndex
    global LENS
    global keypad, last_key_pressed
    while(True):
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, pressed_keys)
            testword[keyIndex]=pressed_keys
            keyIndex+=1
            if (keyIndex is LENS):
                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 Explanation

```
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"]
```

Here, we define the length of the password LENS, the array keys that store the matrix keyboard keys, and the array password that stores the correct password.

```
class Keypad():
```

(continues on next page)

(continued from previous page)

```
    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)
...
```

This class is the code that reads the values of the pressed keys. Refer to *2.1.8 Keypad* of this document for more details.

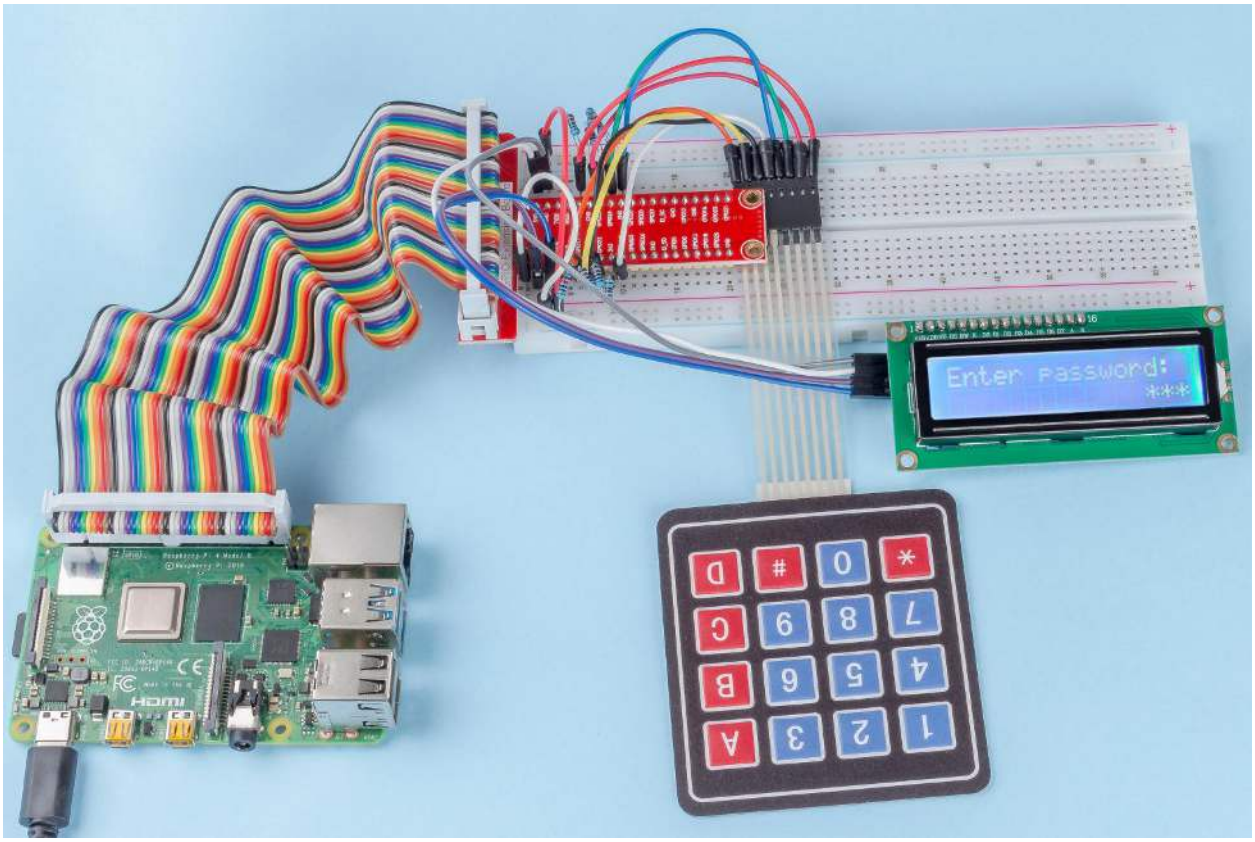
```
while(True):
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, pressed_keys)
            testword[keyIndex]=pressed_keys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

Read the key value and store it in the test array testword. If the number of stored key values is more than 4, the correctness of the password is automatically verified, and the verification results are displayed on the LCD interface.

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

Verify the correctness of the password. Return 1 if the password is entered correctly, and 0 if not.

Phenomenon Picture

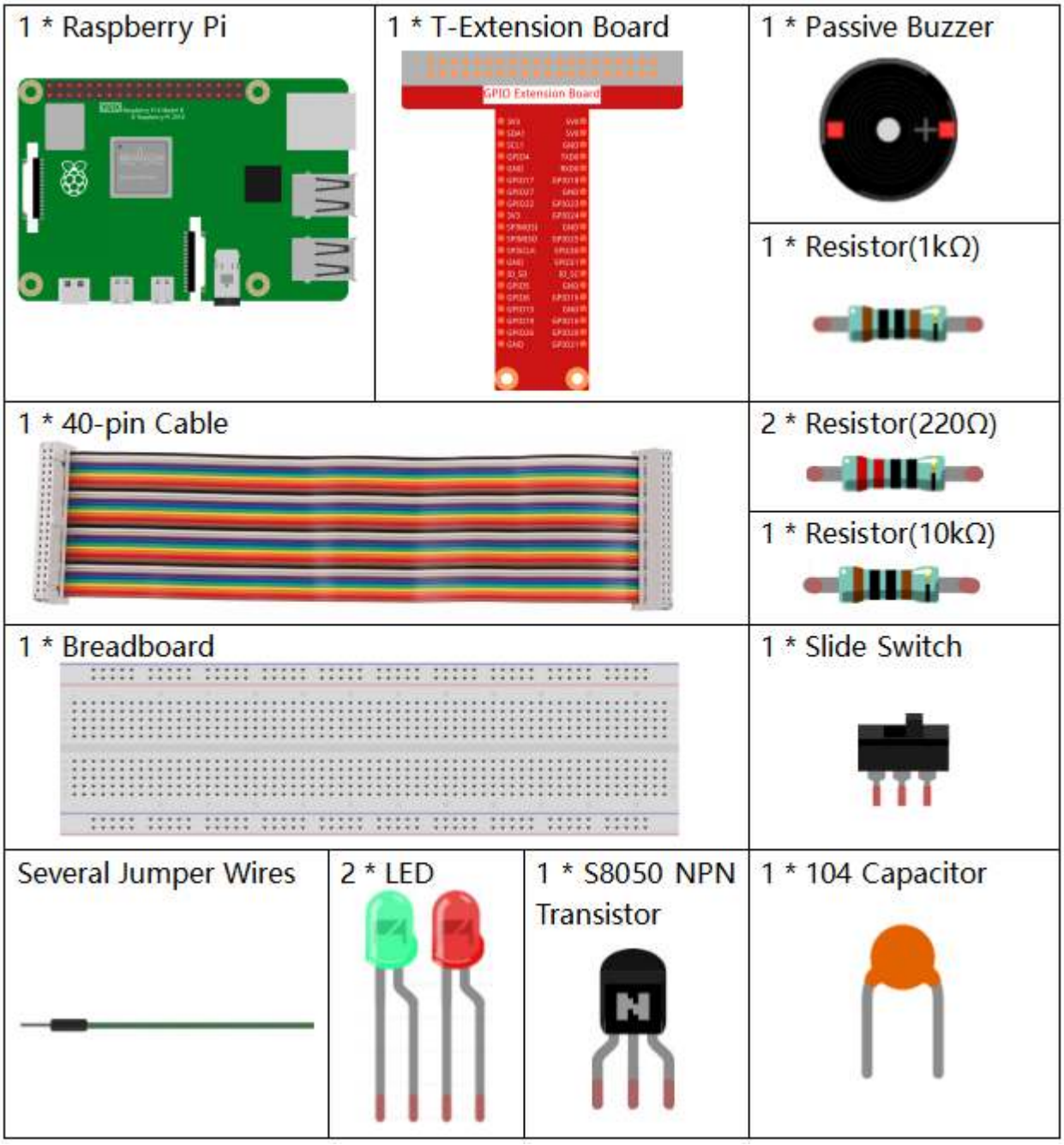


6.6.15 4.1.15 Alarm Bell

Introduction

In this project, we will make a manual alarm device. You can replace the toggle switch with a thermistor or a photosensitive sensor to make a temperature alarm or a light alarm.

Components

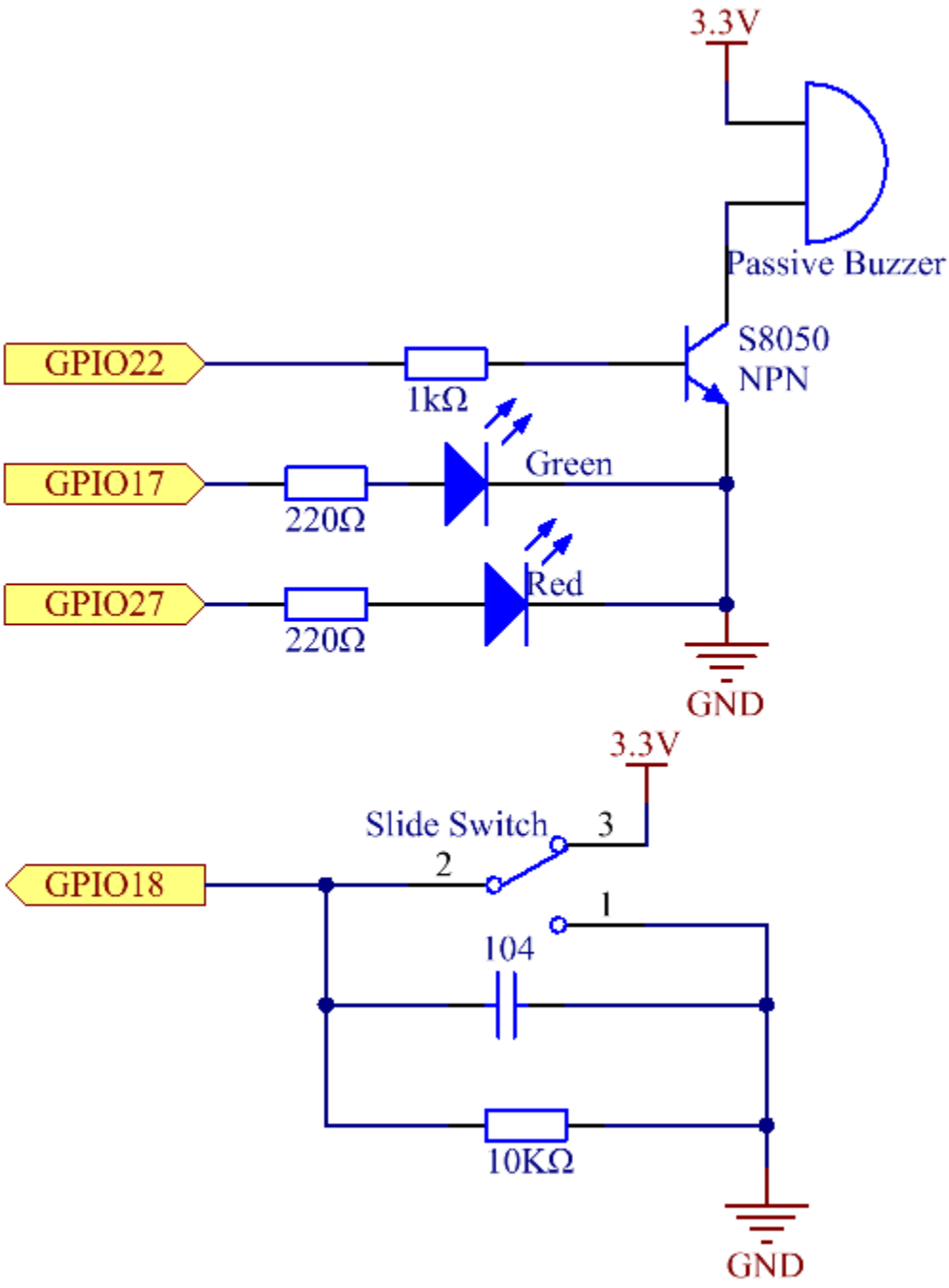


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Buzzer*
- *Slide Switch*

- *Transistor*
- *Capacitor*

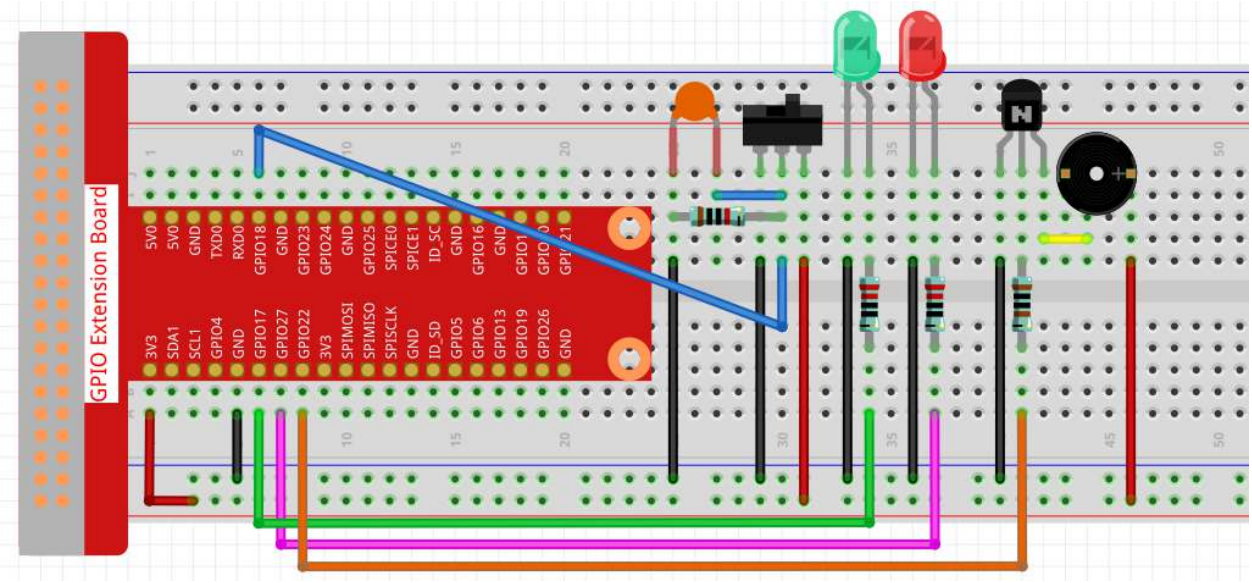
Schematic Diagram

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


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

Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run.

```
sudo python3 4.1.15_AlarmBell.py
```

After the program starts, the toggle switch will be toggled to the right, and the buzzer will give out alarm sounds. At the same time, the red and green LEDs will flash at a certain frequency.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`.

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

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
```

(continues on next page)

(continued from previous page)

```
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():
    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()
```

(continues on next page)

(continued from previous page)

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

Code Explanation

```
import threading
```

Here, we import the `Threading` module and it allows you to do multiple things at once, while normal programs can only execute code from top to bottom. With `Threading` modules, the LED and the buzzer can work separately.

```
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)
```

The function `ledWork()` helps to set the working state of these 2 LEDs: it keeps the green LED lighting up for 0.5s and then turns off; similarly, keeps the red LED lighting up for 0.5s and then turns off.

```
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)
```

The function `buzzWork()` is used to set the working state of the buzzer. Here we set the frequency as between 130 and 800, to accumulate or decay at an interval of 20.

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

In the function `on()` :

1) Define the mark “flag=1”, indicating the ending of the control thread.
2) Start the Buzz, and set the duty cycle to 50%.
3) Create **2** separate threads so that the LED and the buzzer can work at the same time.

`threading.Thread()` function is used to create the thread and its prototype is as follows:

```
class threading.Thread(group=None, target=None, name=None, args=(),
kwargs={}, *, daemon=None)
```

Among the construction methods, the principal parameter is `target`, we need to assign a callable object (here are the functions `ledWork` and `BuzzWork`) to `target`.

Next `start()` is called to start the thread object, ex., `tBuzz.start()` is used to start the newly installed `tBuzz` thread.

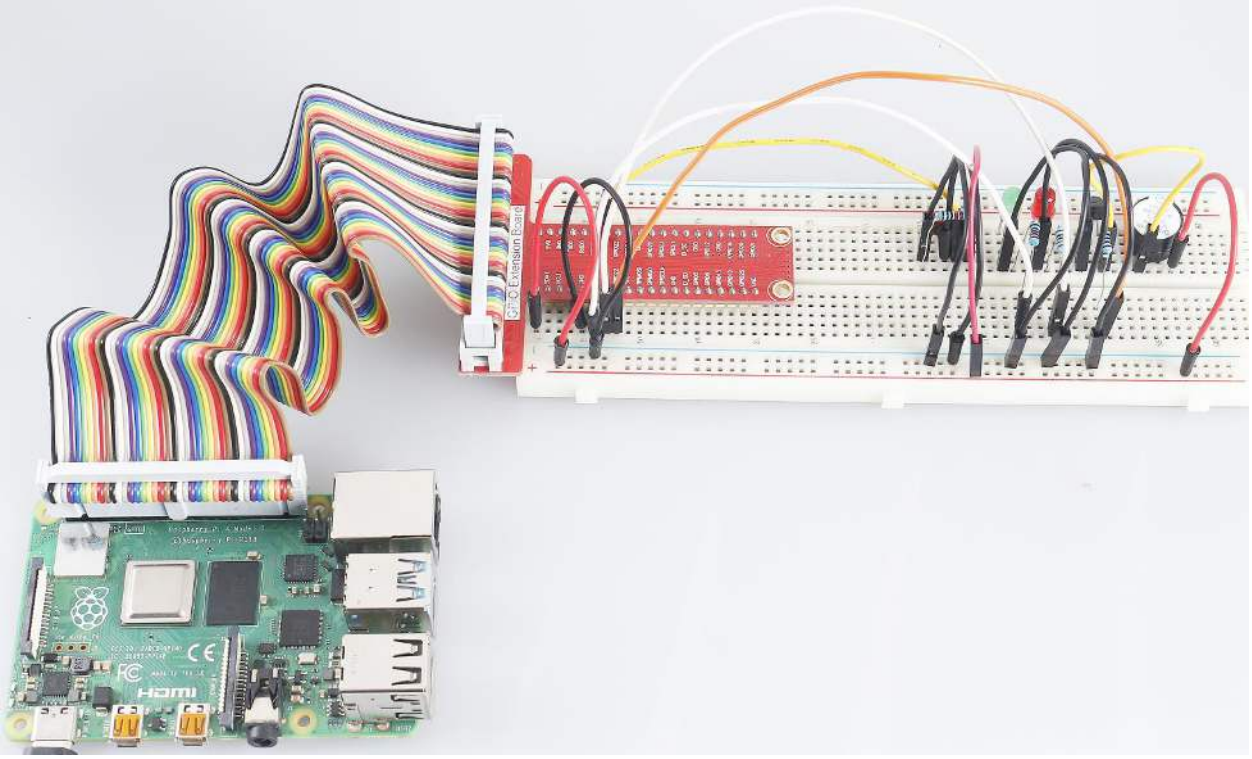
```
def off():
    global flag
    flag = 0
    Buzz.stop()
    GPIO.output(ALedPin,GPIO.LOW)
    GPIO.output(BLedPin,GPIO.LOW)
```

The function `Off()` defines "flag=0" so as to exit the threads **ledWork** and **BuzzWork** and then turn off the buzzer and the LED.

```
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()` contains the whole process of the program: firstly read the value of the slide switch; if the toggle switch is toggled to the right (the reading is 1), the function `on()` is called, the buzzer is driven to emit sounds and the the red and the green LEDs blink. Otherwise, the buzzer and the LED don't work.

Phenomenon Picture

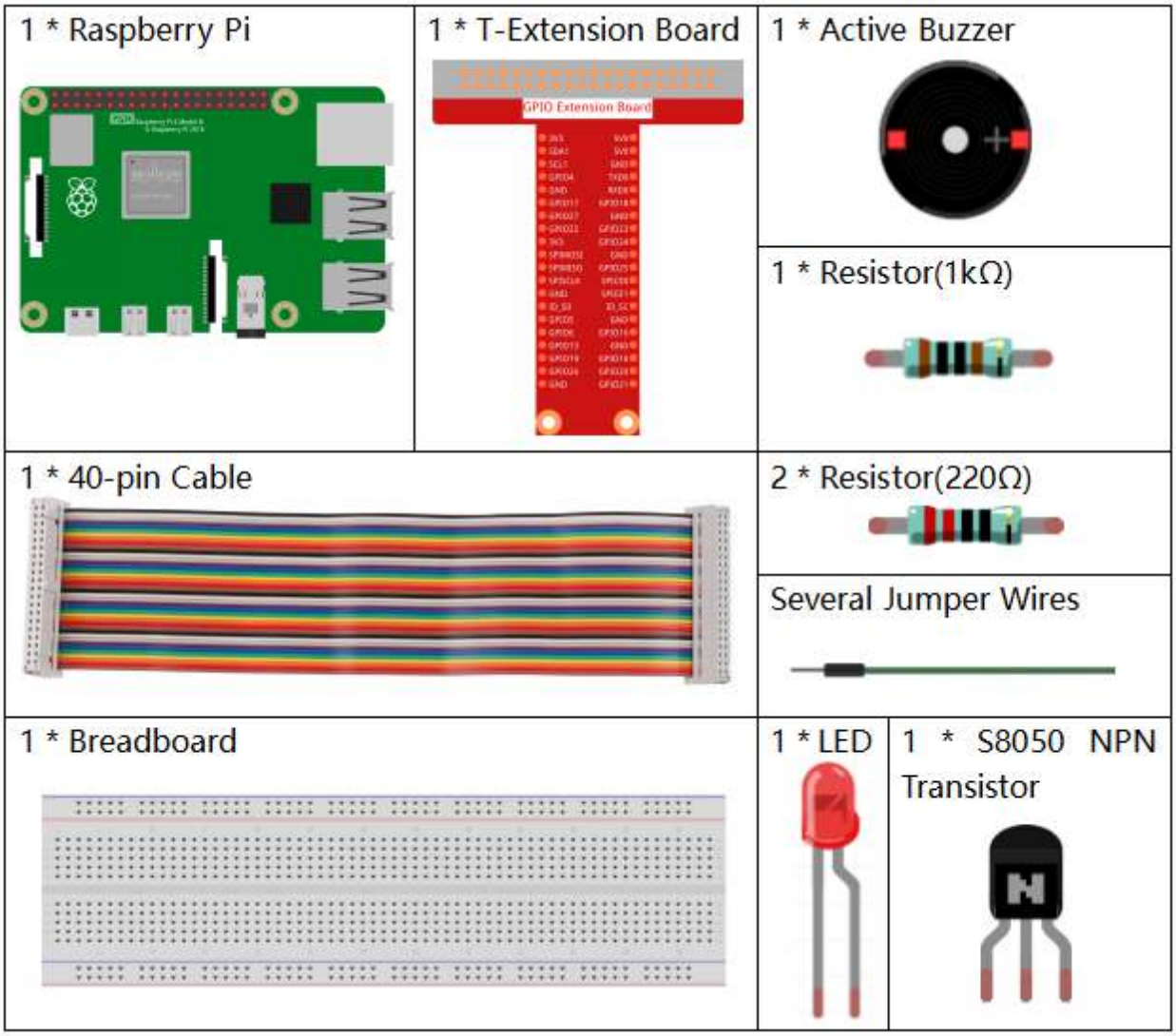


6.6.16 4.1.16 Morse Code Generator

Introduction

In this project, we'll make a Morse code generator, where you type in a series of English letters in the Raspberry Pi to make it appear as Morse code.

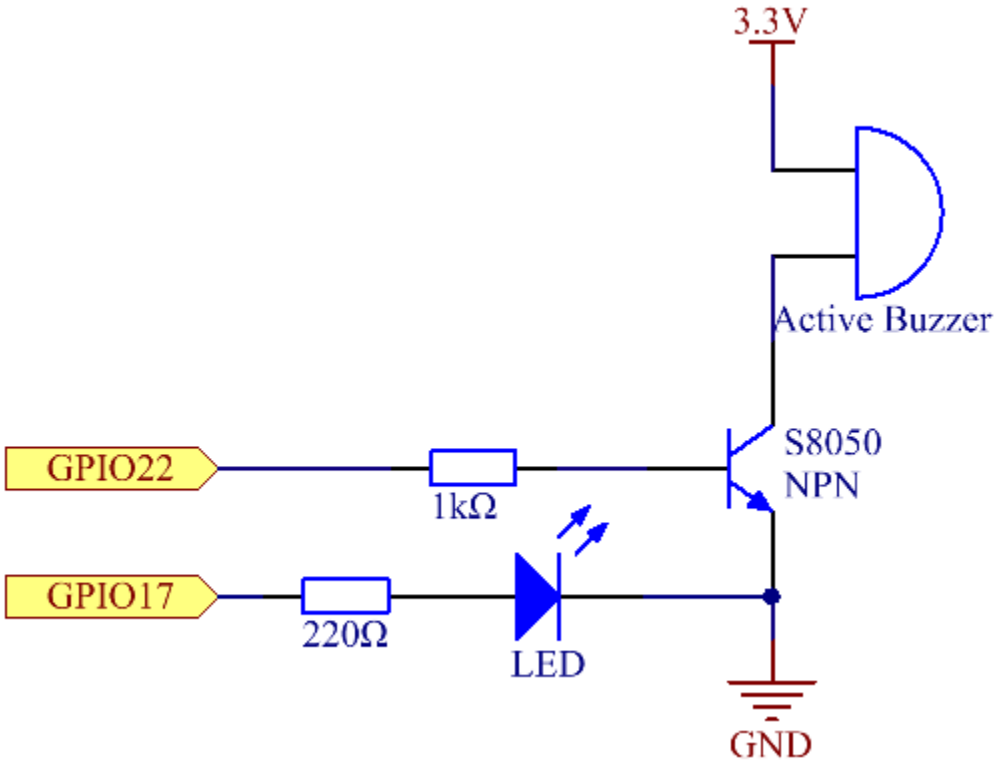
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Buzzer*
- *Transistor*

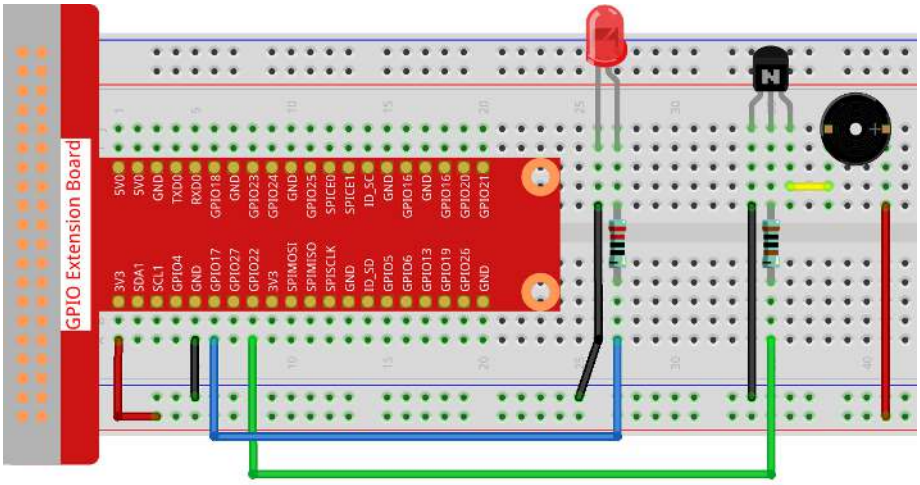
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit. (Pay attention to poles of the buzzer: The one with + label is the positive pole and the other is the negative.)



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/python
```

Step 3: Run.

```
sudo python3 4.1.16_MorseCodeGenerator.py
```

After the program runs, type a series of characters, and the buzzer and the LED will send the corresponding Morse code signals.

Code

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

(continues on next page)

(continued from previous page)

```
        time.sleep(pause)

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

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

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

Code Explanation

```
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',
    }
```

This structure MORSE is the dictionary of the Morse code, containing characters A-Z, numbers 0-9 and marks “?” “/” “:” “,” “.” “;” “!” “@” .

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

The function `on()` starts the buzzer and the LED.

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

The function `off()` is used to turn off the buzzer and the LED.

```
def beep(dt):   # x for dalay time.
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

Define a function `beep()` to make the buzzer and the LED emit sounds and blink in a certain interval of `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)
```

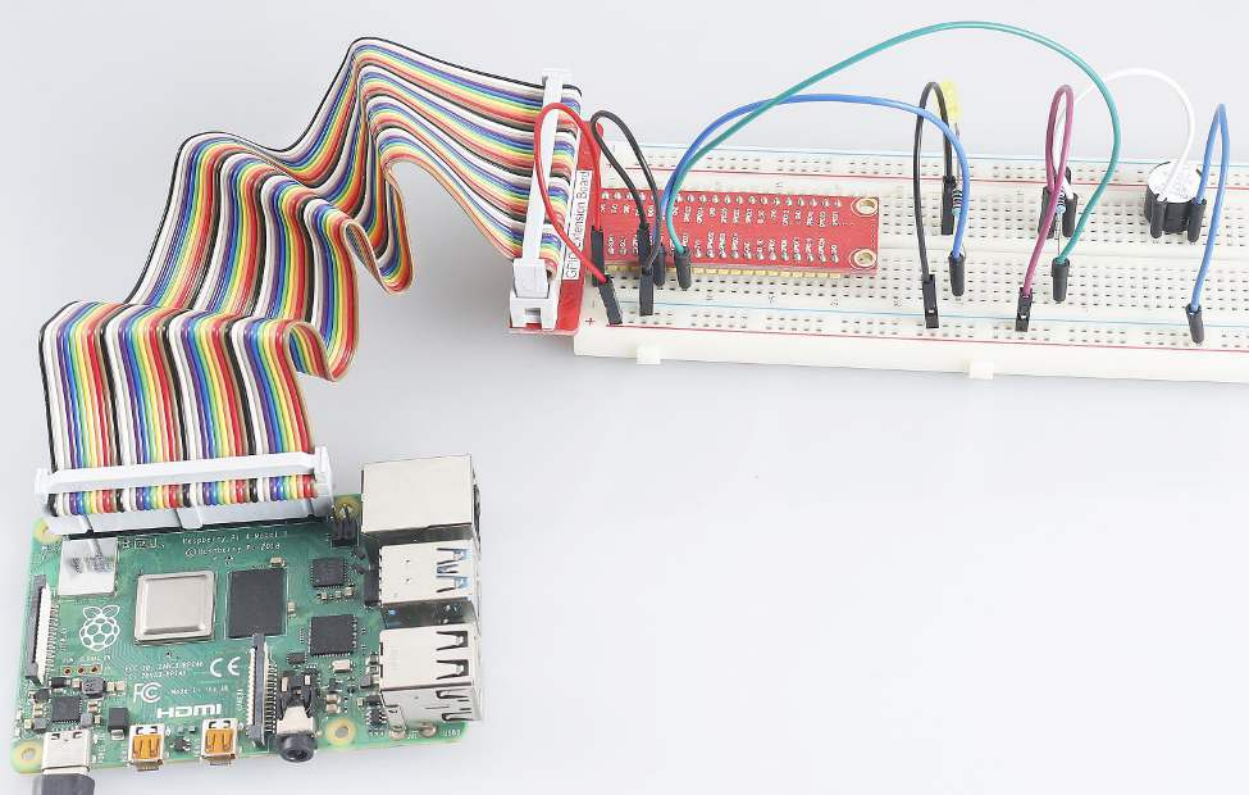
The function `morsecode()` is used to process the Morse code of input characters by making the "1" of the code keep emitting sounds or lights and the "0"shortly emit sounds or lights, ex., input "SOS", and there will be a signal containing three short three long and then three short segments " · · · — — — · · · ".

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

When you type the relevant characters with the keyboard, `upper()` will convert the input letters to their capital form.

`printf()` then prints the clear text on the computer screen, and the `morsecod()` function causes the buzzer and the LED to emit Morse code.

Phenomenon Picture

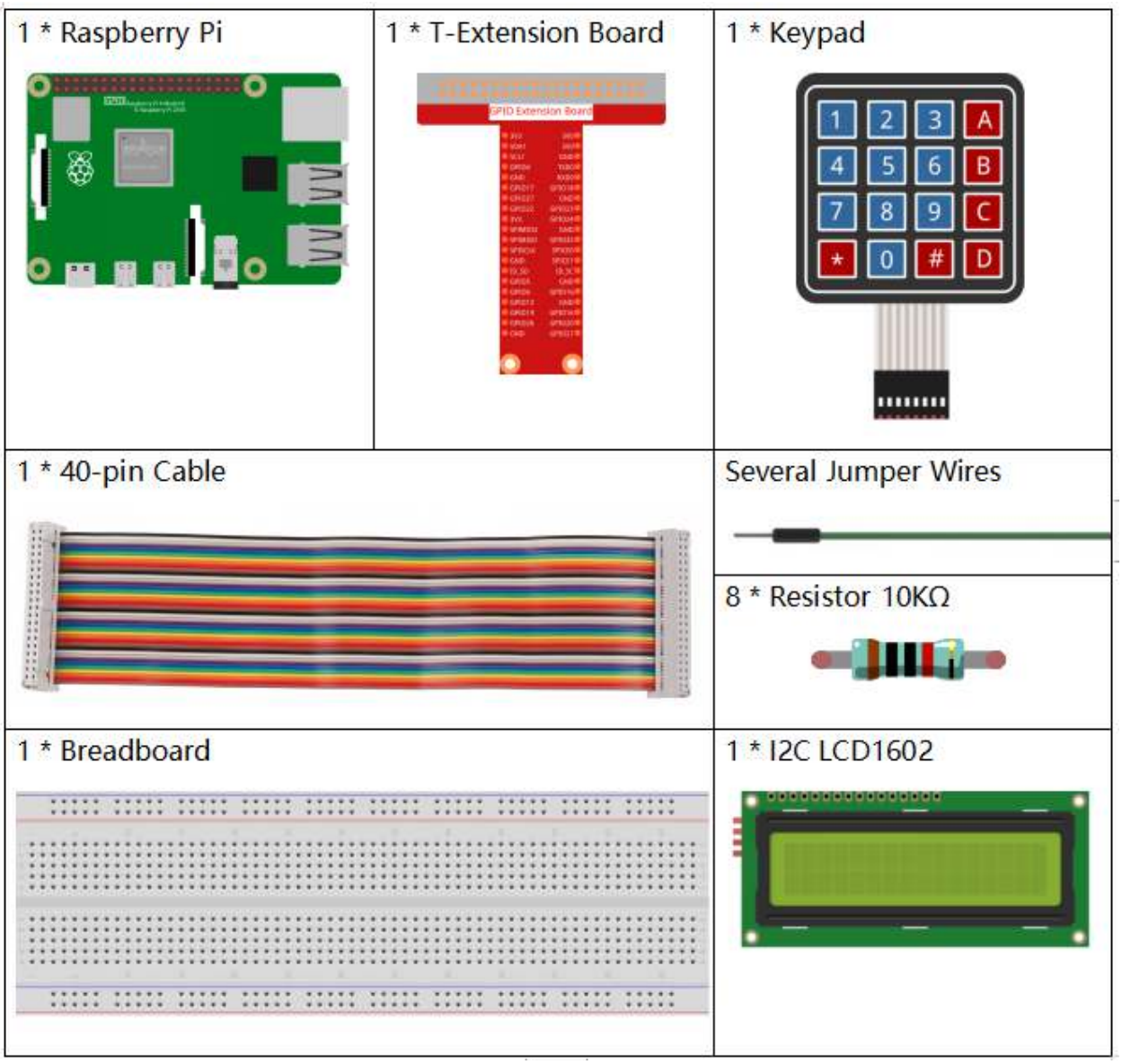


6.6.17 4.1.17 GAME– Guess Number

Introduction

Guessing Numbers is a fun party game where you and your friends take turns inputting a number (0~99). The range will be smaller with the inputting of the number till a player answers the riddle correctly. Then the player is defeated and punished. For example, if the lucky number is 51 which the players cannot see, and the player inputs 50, the prompt of number range changes to 50~99; if the player inputs 70, the range of number can be 50~70; if the player inputs 51, this player is the unlucky one. Here, we use keypad to input numbers and use LCD to output outcomes.

Components

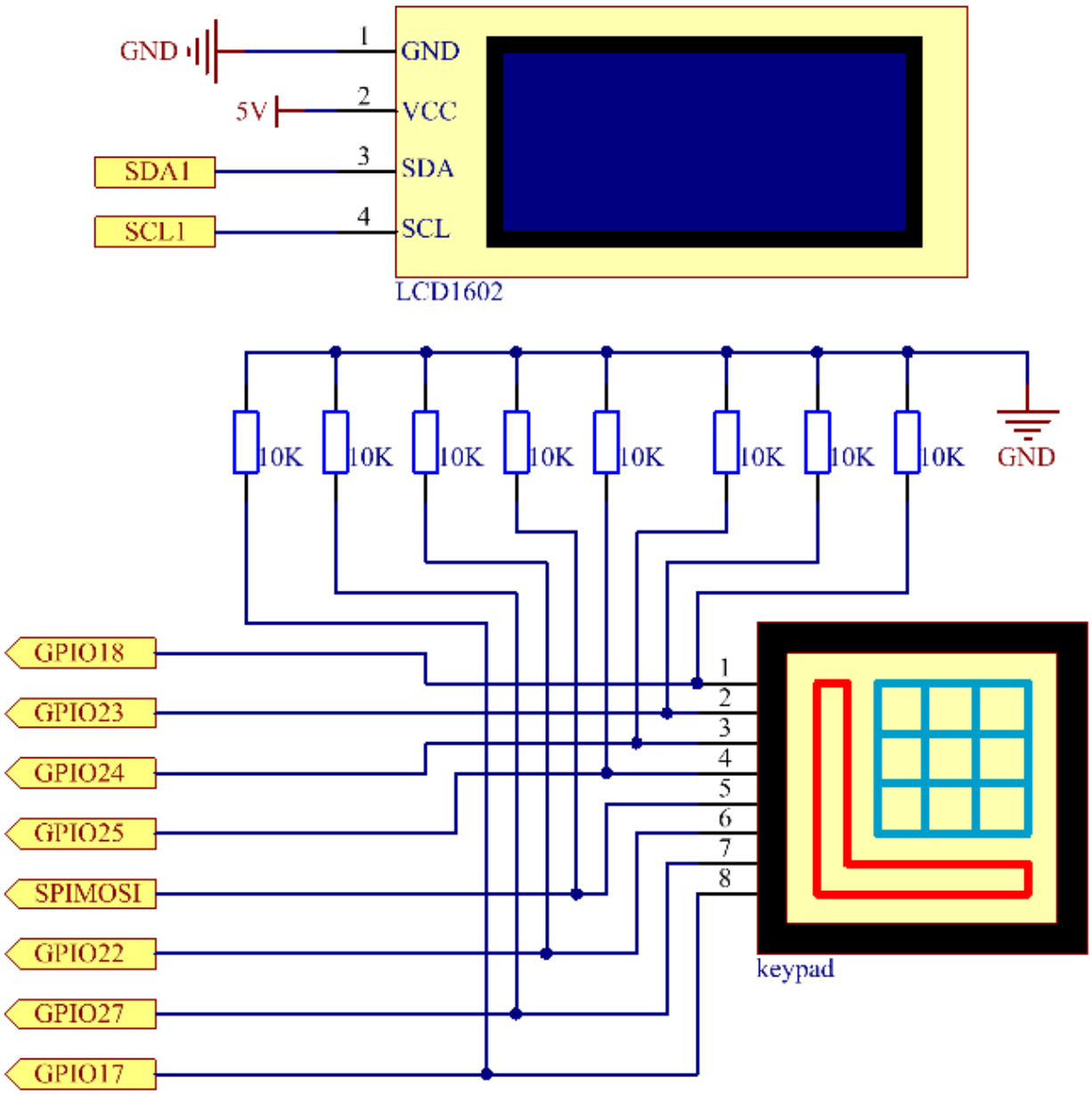


- *GPIO Extension Board*
- *Breadboard*

- *Resistor*
- *Keypad*
- *I2C LCD1602*

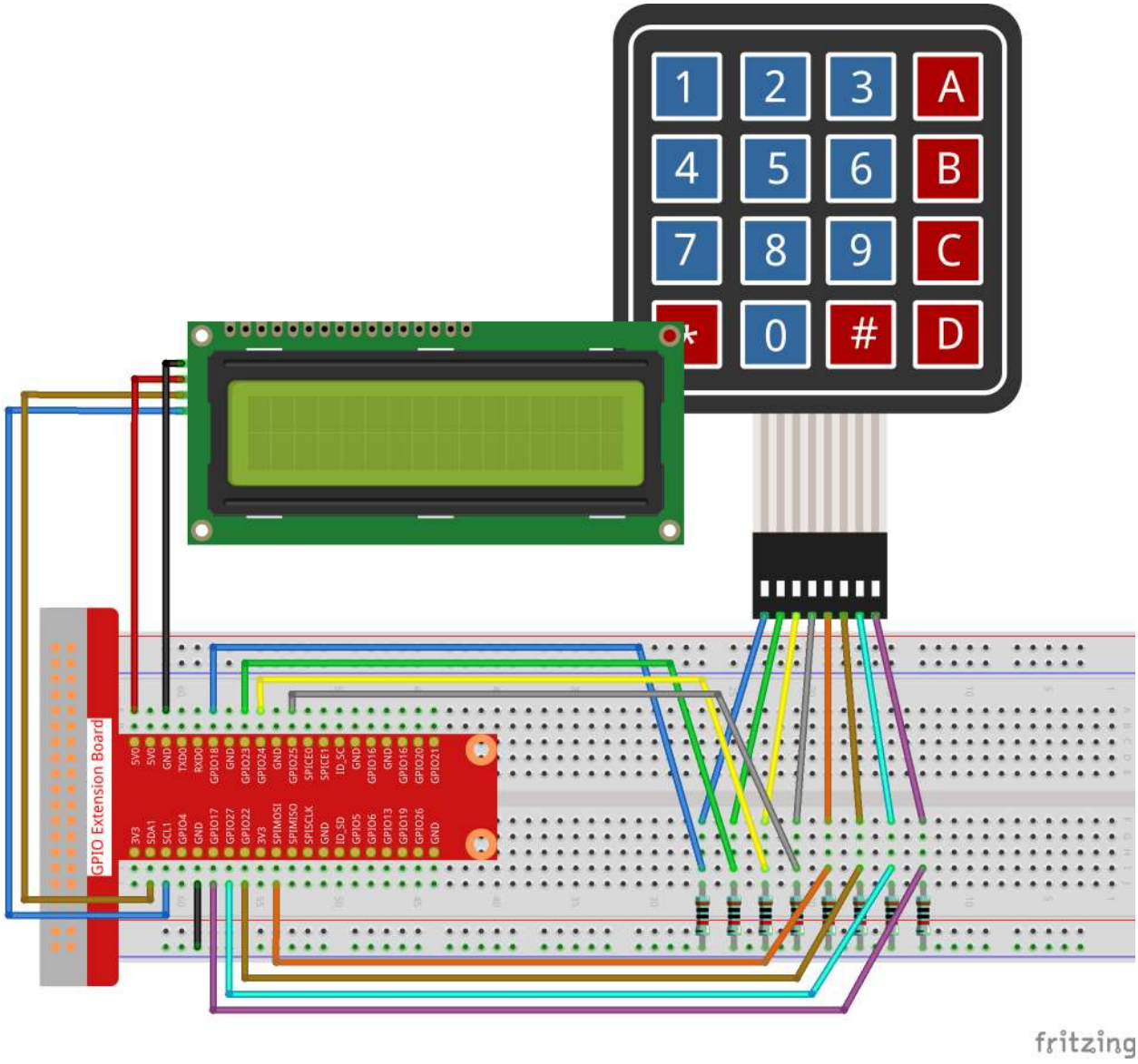
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17
SDA1	Pin 3	SDA1(8)	SDA1(2)
SCL1	Pin 5	SCL1(9)	SDA1(3)



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see *I2C Configuration*.)

Step 3: Change directory.

```
cd /home/pi/raphael-kit/python/
```

Step 4: Run.

```
sudo python3 4.1.17_GAME_GuessNumber.py
```

After the program runs, there displays the initial page on the LCD:

```
Welcome!
Press A to go!
```

Press 'A', and the game will start and the game page will appear on the LCD.

```
Enter number:
0 ‹point‹ 99
```

A random number '**point**' is produced but not displayed on the LCD when the game starts, and what you need to do is to guess it. The number you have typed appears at the end of the first line till the final calculation is finished. (Press 'D' to start the comparation, and if the input number is larger than **10**, the automatic comparation will start.)

The number range of 'point' is displayed on the second line. And you must type the number within the range. When you type a number, the range narrows; if you got the lucky number luckily or unluckily, there will appear "You've got it!"

Note:

- If you get the error `FileNotFoundError: [Errno 2] No such file or directory: '/dev/i2c-1'`, you need to refer to *I2C Configuration* to enable the I2C.
- If you get `ModuleNotFoundError: No module named 'smbus2'` error, please run `sudo pip3 install smbus2`.
- If the error `OSError: [Errno 121] Remote I/O error` appears, it means the module is miswired or the module is broken.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

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

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

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

   def __init__(self, rowsPins, colsPins, keys):
      self.rowsPins = rowsPins
      self.colsPins = colsPins
      self.keys = keys
      GPIO.setwarnings(False)
      GPIO.setmode(GPIO.BCM)
      GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
      GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

   def read(self):
      pressed_keys = []
      for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
               index = i * len(self.colsPins) + j
               if (GPIO.input(col) == 1):
```

(continues on next page)

(continued from previous page)

```
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
      return pressed_keys

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

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

def setup():
   global keypad, last_key_pressed,keys
   rowsPins = [18,23,24,25]
   colsPins = [10,22,27,17]
   keys = ["1","2","3","A",
           "4","5","6","B",
           "7","8","9","C",
           "*","0","#","D"]
   keypad = Keypad(rowsPins, colsPins, keys)
   last_key_pressed = []
   LCD1602.init(0x27, 1)     # init(slave address, background light)
   LCD1602.clear()
   LCD1602.write(0, 0, 'Welcome!')
   LCD1602.write(0, 1, 'Press A to Start!')

def init_new_value():
   global pointValue,upper,count,lower
   pointValue = random.randint(0,99)
   upper = 99
   lower = 0
   count = 0
   print('point is %d' %(pointValue))

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

(continues on next page)

(continued from previous page)

```
    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
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 Explanation

At the beginning part of the code are the functional functions of **keypad** and **I2C LCD1602**. You can learning more details about them in *1.1.7 I2C LCD1602* and *2.1.8 Keypad*.

Here, what we need to know is as follows:

```
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))
```

The function produces the random number '**point**' and resets the range hint of the point.

```
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()` compares the input number (**count**) with the produced "**point**". If the comparing outcome is that they are not same, **count** will assign values to **upper** and **lower** and return '**0**'; otherwise, if the outcome indicates they are same, there returns '**1**'.

```
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))
```

This function works for displaying the game page.

`str(count)`: Because `write()` can only support the data type — **string**, `str()` is needed to convert the **number** into **string**.

```
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()
```

(continues on next page)

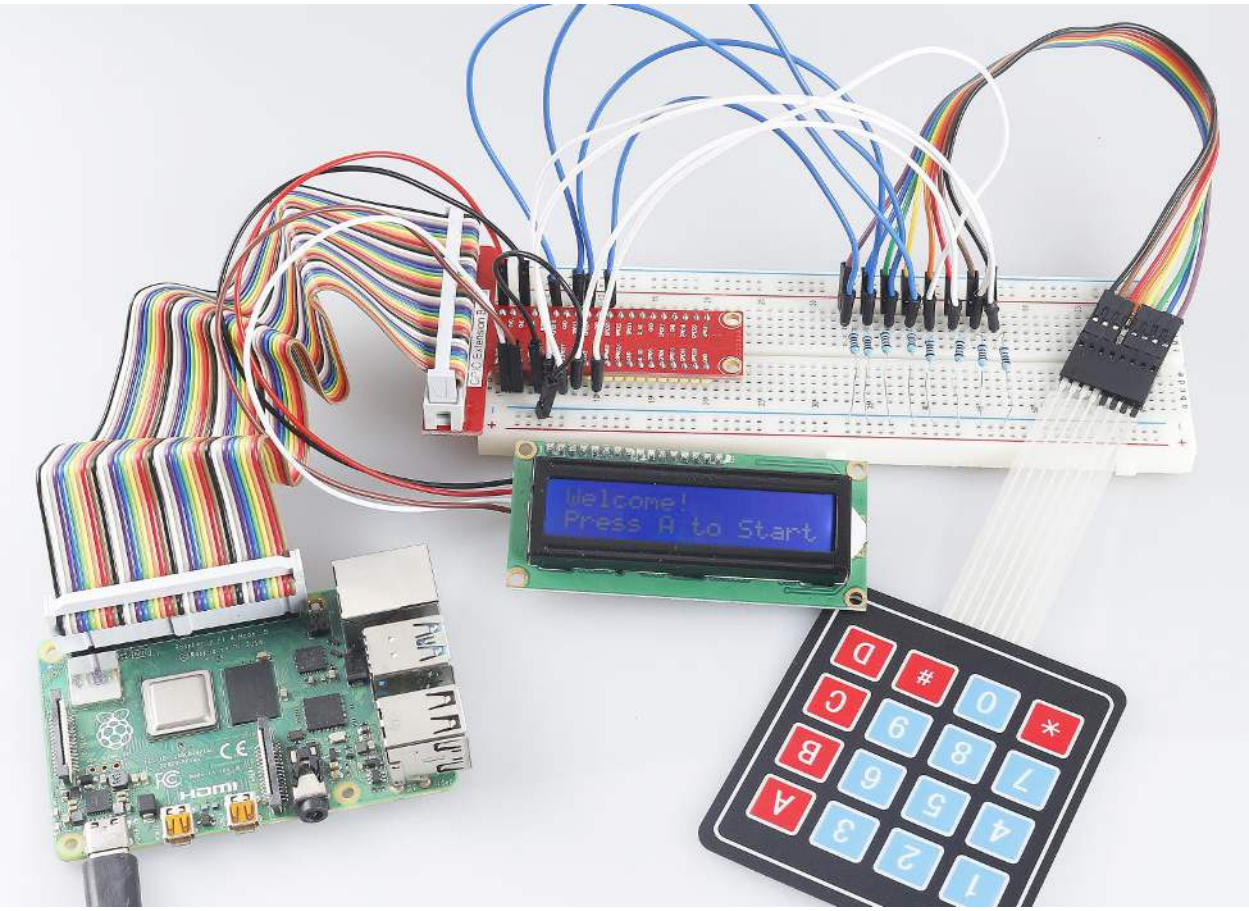
(continued from previous page)

```
                lcd_show_input(result)
            print(pressed_keys)
        last_key_pressed = pressed_keys
        time.sleep(0.1)
```

`main()` contains the whole process of the program, as show below:

1) Initialize **I2C LCD1602** and **Keypad**.
2) Judge whether the button is pressed and get the button reading.
3) If the button '**A**' is pressed, a random number **0-99** will appear then the game starts.
4) If the button '**D**' is detected to have been pressed, the program will enter into the outcome judgement.
5) If the button **0-9** is pressed, the value of **count** will be changed; if the **count** is larger than **10**, then the judgement starts.
6) The changes of the game and its values are displayed on **LCD1602**.

Phenomenon Picture

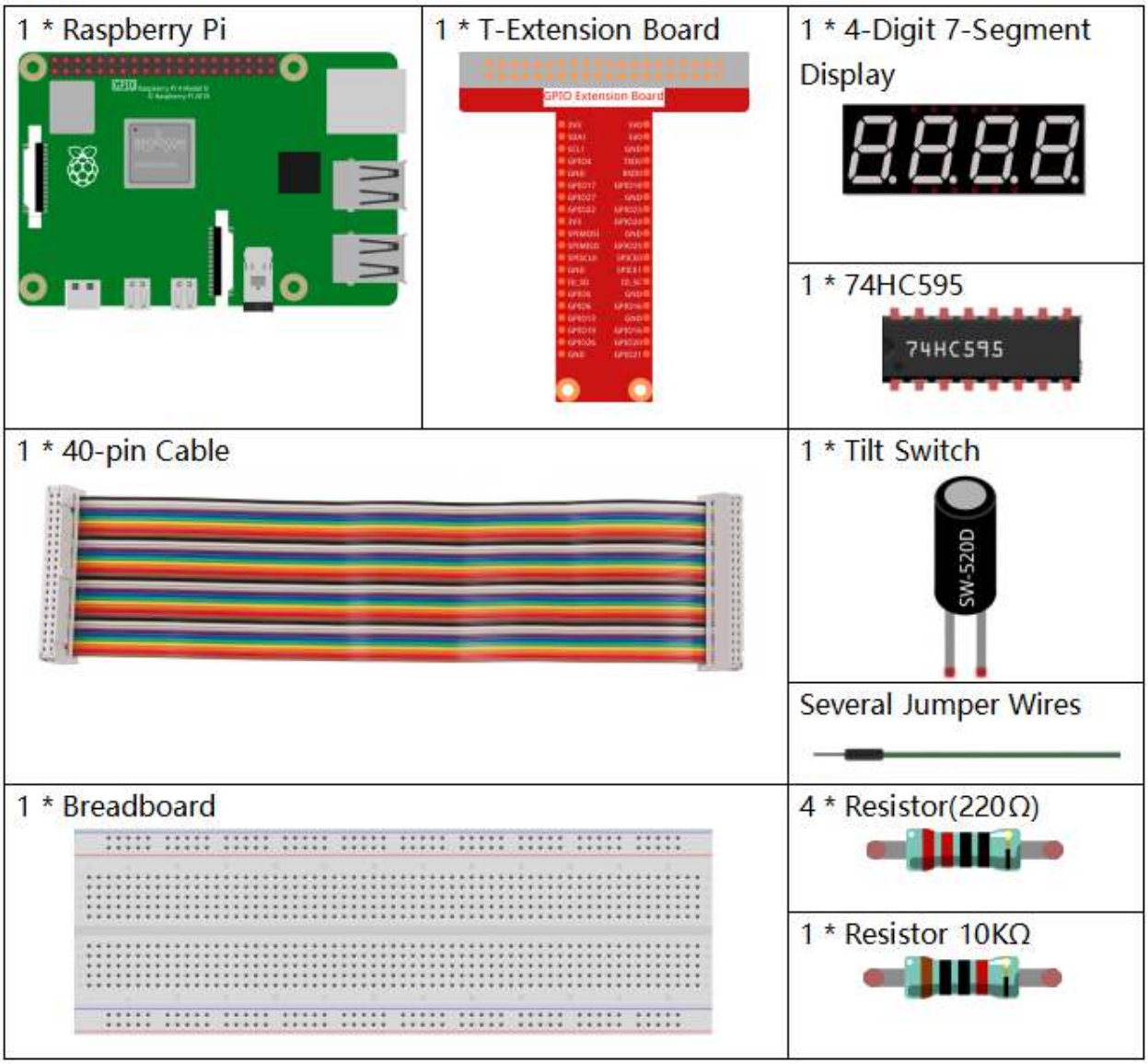


6.6.18 4.1.18 GAME - 10 Second

Introduction

Next, follow me to make a game device to challenge your concentration. Tie the tilt switch to a stick to make a magic wand. Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. You can play the game with your friends to see who is the time wizard.

Components

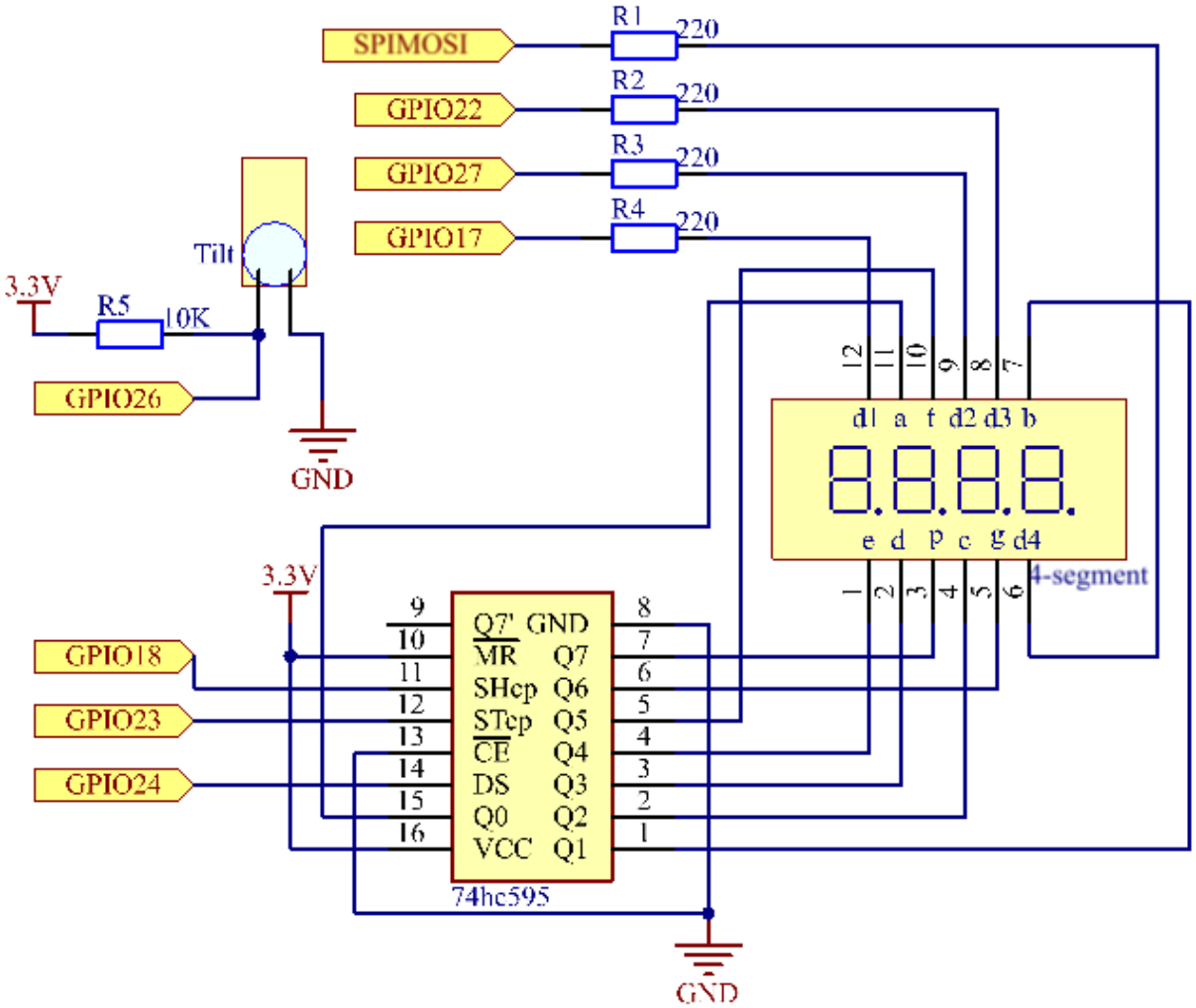


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*

- *4-Digit 7-Segment Display*
- *74HC595*
- *Tilt Switch*

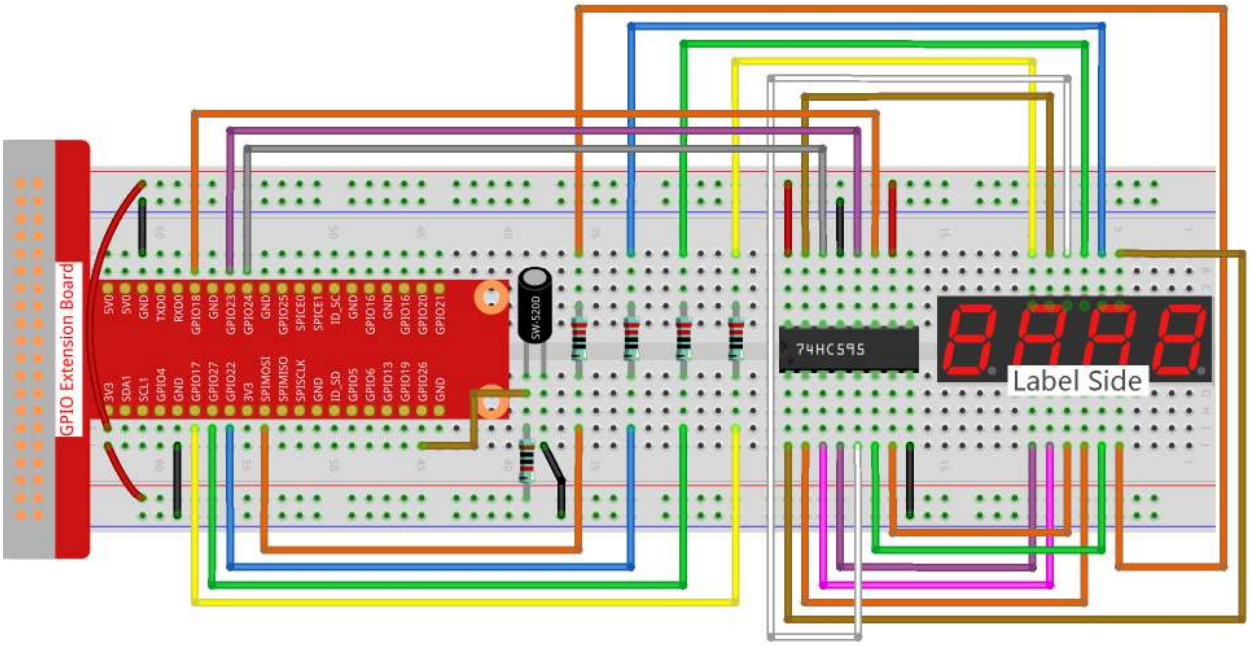
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/python/
```

Step 3: Run the executable file.

```
sudo python3 4.1.18_GAME_10Second.py
```

Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. Shake it one more time to start the next round of the game.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
import threading

sensorPin = 26

SDI = 24
RCLK = 23
SRCLK = 18

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

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
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 Explanation

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

The game is divided into two modes:

`gameState==0` is the “start” mode, in which the time is timed and displayed on the segment display, and the tilting switch is shaken to enter the “show” mode.

`gameState==1` is the "show" mode, which stops the timing and displays the time on the segment display. Shaking the tilt switch again will reset the timer and restart the game.

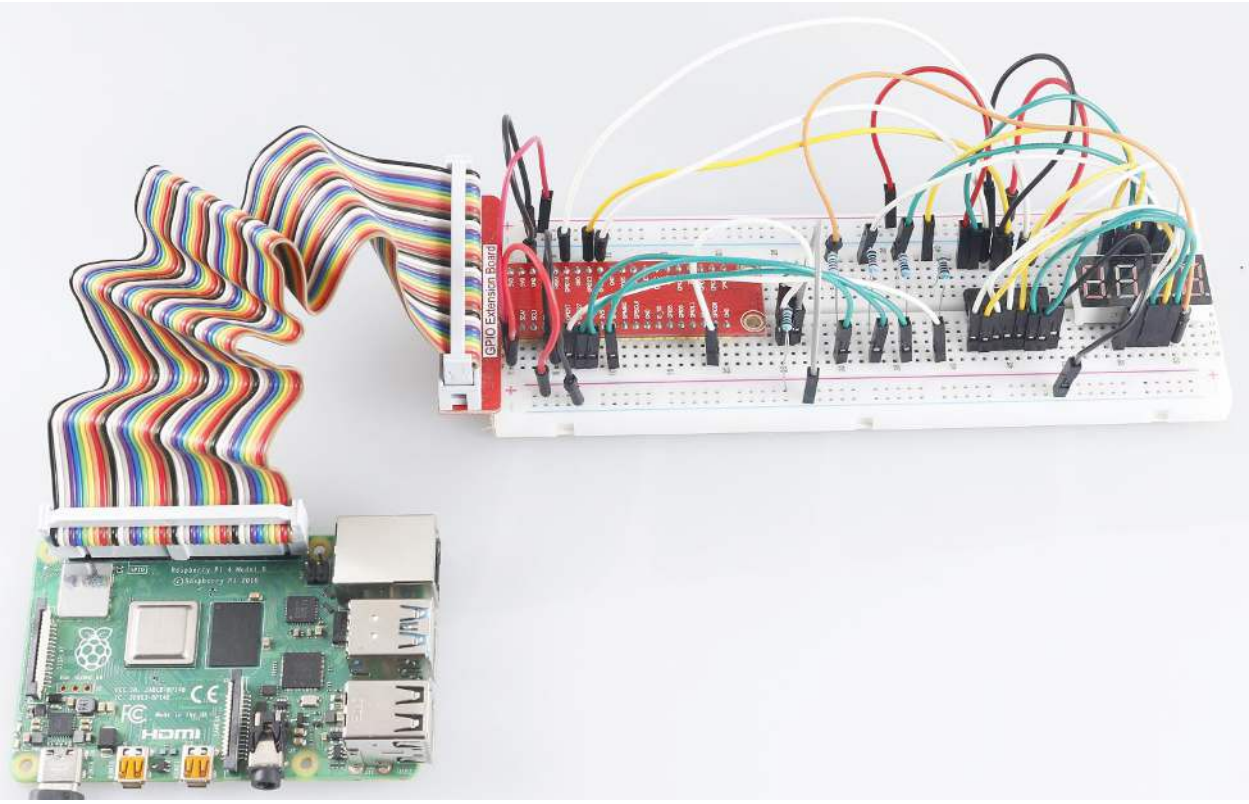
```
def loop():
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState=currentState
```

`loop()` is the main function. First, the time is displayed on the 4-bit segment display and the value of the tilt switch is read. If the state of the tilt switch has changed, `stateChange()` is called.

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

After the interval reaches 0.01s, the timer function is called; add 1 to counter, and the timer is used again to execute itself repeatedly every 0.01s.

Phenomenon Picture

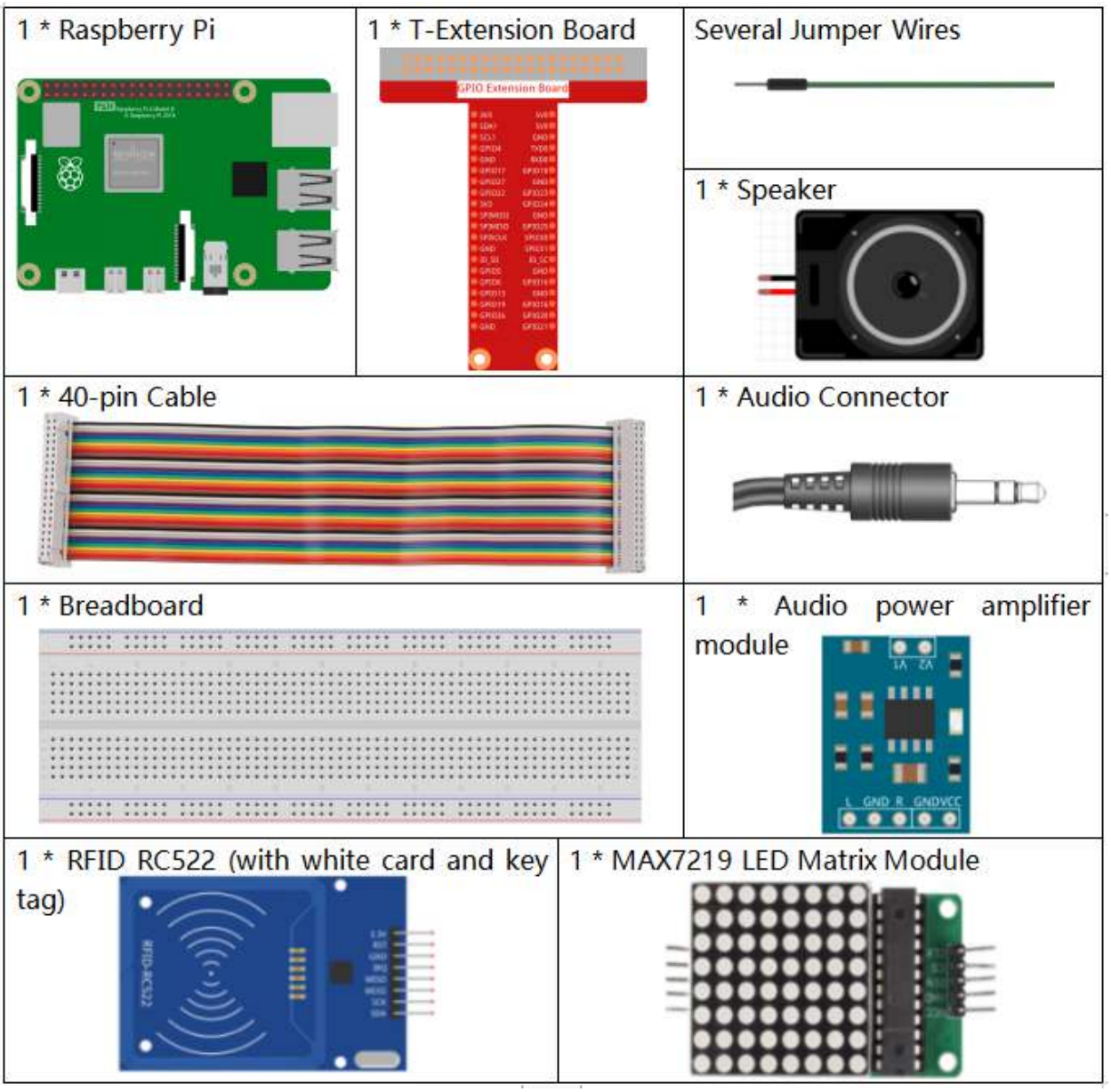


6.6.19 4.1.19 AttendanceSystem

Introduction

Let's make a simple attendance system. When we scan the card, the Raspberry Pi will record our information and generate a csv file.

Components

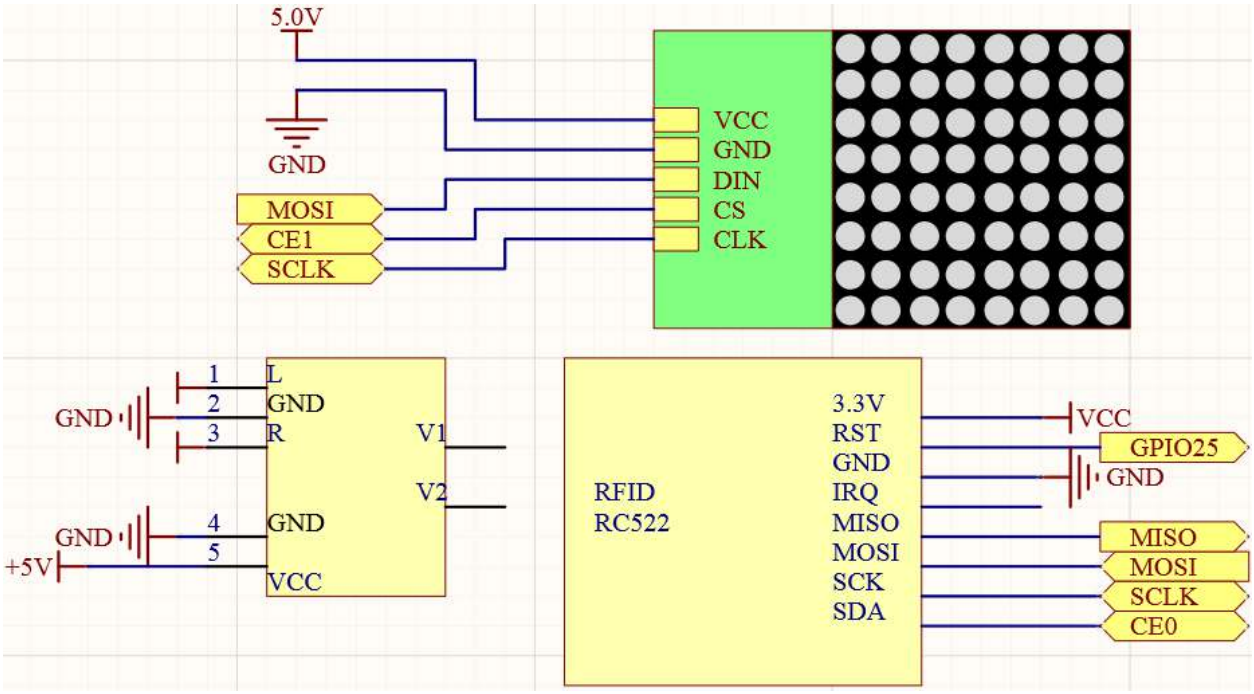


- *GPIO Extension Board*
- *Breadboard*
- *Audio Module and Speaker*

- *LED Matrix Module*
- *MFRC522 Module*

Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	MOSI
SPIMISO	Pin 19	12	MISO
SPICE0	pin 24	10	CE0
SPICE1	pin 26	11	CE1
SPISCLK	Pin 23	14	SCLK

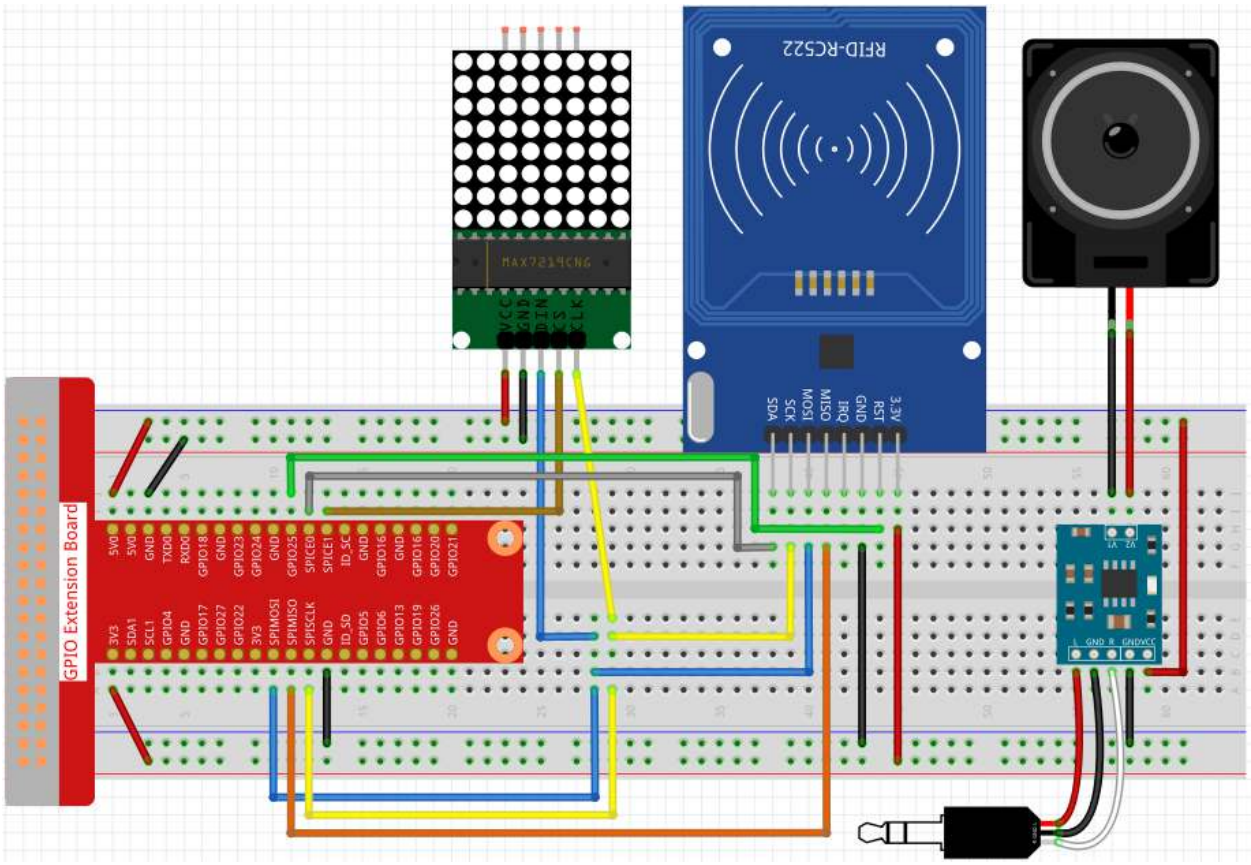


Experimental Procedures

Note: Turn on the SPI before starting the experiment, refer to *SPI Configuration* for details.

The *Luma.LED_Matrix* and the *Spidev and MFRC522* libraries are also needed.

Step 1: Build the circuit.



Step 2: Run the `2.2.10_write.py` file to modify the content of the rfid card.

```
cd /home/pi/raphael-kit/python
sudo python3 2.2.10_write.py
```

Step 3: Enter the name (here we use `John``as an example) and press ``Enter` to confirm, then put the card on the MFRC522 module, wait for "Data writing is complete" to appear and take the card away, or rewrite the message to another card and exit by `Ctrl+C`.

```
pi@raspberrypi:~/raphael-kit/python $ sudo python3 2.2.10_write.py
Please write new data:John
Please place the card to complete writing
Data writing is complete
Please write new data:
```

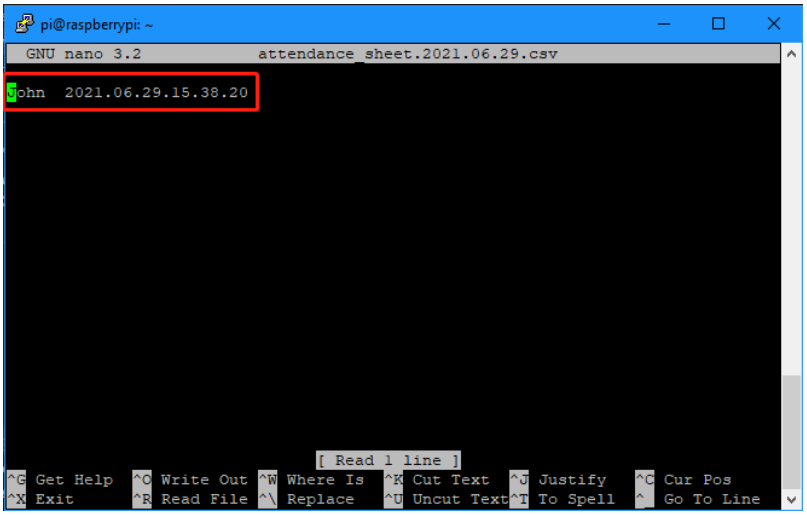
Step 4: Get into the folder of code and run.

```
cd /home/pi/raphael-kit/python
sudo python3 4.1.19_Attendance_Machine.py
```

After starting the program, we put the RFID card close to the MFRC522 RFID Module, the Raspberry Pi will send out a voice to greet you and display it on the LED matrix.

We can also find a `.csv` file that records the time and list in the same directory. Open it with the nano command and you will see the record just now.

```
sudo nano attendance_sheet.2021.06.29.csv
```



Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `raphael-kit/python`. After modifying the code, you can run it directly to see the effect.

```
import time
from tts import TTS
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.led_matrix.device import max7219
from luma.core.legacy import text
from luma.core.legacy.font import proportional, CP437_FONT, LCD_FONT

serial = spi(port=0, device=1, gpio=noop())
device = max7219(serial, rotate=1)
virtual = viewport(device, width=200, height=400)

reader = SimpleMFRC522()

tts = TTS(engine="espeak")
tts.lang('en-US')

attendance_statistics = {}

def get_time():
    time.time()
    year = str(time.strftime('%Y',time.localtime(time.time())))
    month = str(time.strftime('%m',time.localtime(time.time())))
    day = str(time.strftime('%d',time.localtime(time.time())))
    hour = str(time.strftime('%H',time.localtime(time.time())))
    minute = str(time.strftime('%M',time.localtime(time.time())))
    second = str(time.strftime('%S',time.localtime(time.time())))
```

(continues on next page)

(continued from previous page)

```
    present_time = year + '.' + month + '.' + day + '.' + hour + '.' + minute + '.' +␣
↪second
    present_date = year + '.' + month + '.' + day
    return present_date, present_time

def main():
    while True:
        print("Reading...Please place the card...")
        id, name = reader.read()
        print(id,name)
        greeting = name.rstrip() + ", Welcome!"
        present_date, present_time = get_time()
        attendance_statistics[name.rstrip()] = present_time
        tts.say(greeting)
        with open('attendance_sheet.' + present_date + '.csv', 'w') as f:
            [f.write('{0}  {1}\n'.format(key, value)) for key, value in attendance_
↪statistics.items()]
        with canvas(virtual) as draw:
            text(draw, (0, 0), greeting, fill="white", font=proportional(CP437_FONT))
        for offset in range(95):
            virtual.set_position((offset,0))
            time.sleep(0.1)

def destroy():
    GPIO.cleanup()
    pass

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

Code Explanation

In order to better understand the program, you may need to complete *1.1.6 LED Dot Matrix* , *2.2.10 MFRC522 RFID Module* and *3.1.4 Text-to-speech* first.

```
def get_time():
        time.time()
        year = str(time.strftime('%Y',time.localtime(time.time())))
        month = str(time.strftime('%m',time.localtime(time.time())))
        day = str(time.strftime('%d',time.localtime(time.time())))
        hour = str(time.strftime('%H',time.localtime(time.time())))
        minute = str(time.strftime('%M',time.localtime(time.time())))
        second = str(time.strftime('%S',time.localtime(time.time())))
        present_time = year + '.' + month + '.' + day + '.' + hour + '.' + minute + '.
↪' + second
        present_date = year + '.' + month + '.' + day
        return present_date, present_time
```

Use the `get_time()` function to get the current timestamp and return two values. Among them, `present_date` is accurate to the number of days of the current timestamp, and `present_time` is accurate to the number of seconds of the current timestamp.

```
id, name = reader.read()
greeting = name.rstrip() + ", Welcome!"
present_date, present_time = get_time()
attendance_statistics[name.rstrip()] = present_time
```

The `reader.read()` function reads the name information, and then creates a greeting. Then an `attendance_statistics` dictionary is generated, and `name.rstrip()` and `present_time` are stored as keys and values.

```
tts.say(greeting)
```

Say a greeting through the speaker.

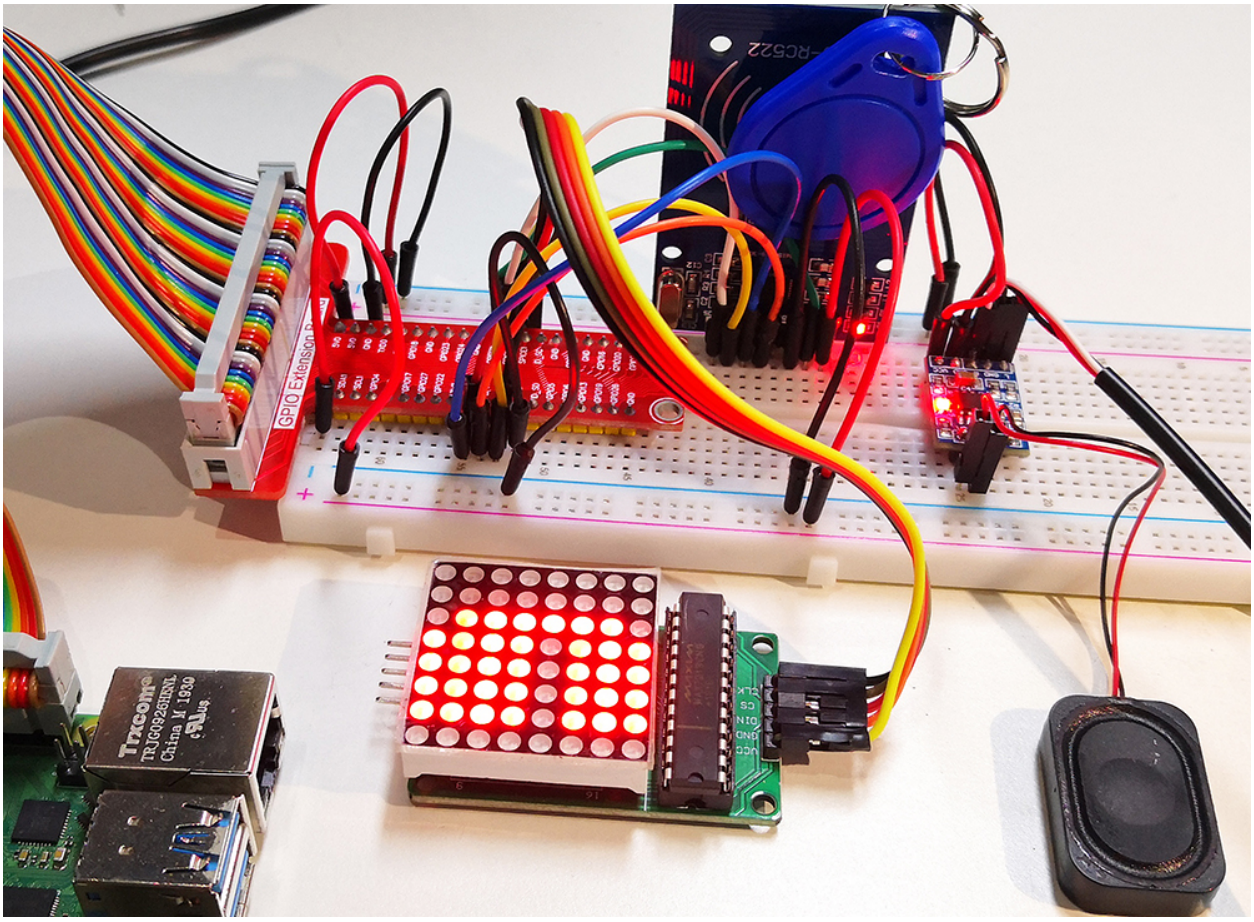
```
with open('attendance_sheet.' + present_date + '.csv', 'w') as f:
    [f.write('{0}  {1}\n'.format(key, value)) for key, value in attendance_statistics.
↪items()]
```

Write the `attendance_statistics` to the .csv file.

```
with canvas(virtual) as draw:
    text(draw, (0, 0), greeting, fill="white", font=proportional(CP437_FONT))
for offset in range(95):
    virtual.set_position((offset,0))
    time.sleep(0.1)
```

Scroll to display this greeting.

Phenomenon Picture


GPIO Extension
Trxcom
TRJG0926HENL
China M 1930

CHAPTER

SEVEN

PLAY WITH C

7.1 Install and Check the WiringPi

`wiringPi` is a C language GPIO library applied to the Raspberry Pi. It complies with GUN Lv3. The functions in wiringPi are similar to those in the wiring system of Arduino. They enable the users familiar with Arduino to use wiringPi more easily.

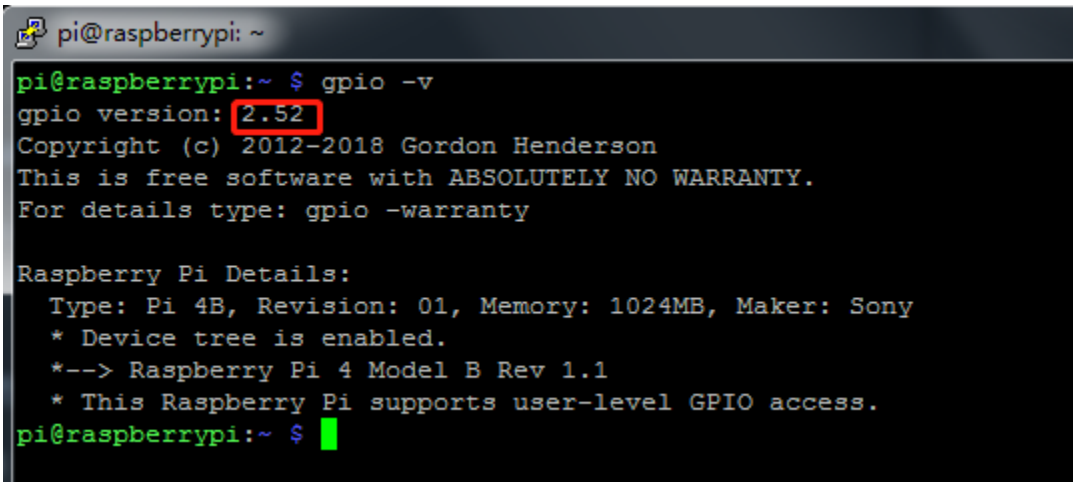
`wiringPi` includes lots of GPIO commands which enable you to control all kinds of interfaces on Raspberry Pi.

Please run the following command to install `wiringPi` library.

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

You can test whether the wiringPi library is installed successfully or not by the following instruction.

```
gpio -v
```



Check the GPIO with the following command:

```
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---+---+------+---------+-----+-----+
```

For more details about wiringPi, you can refer to WiringPi.

7.2 Output

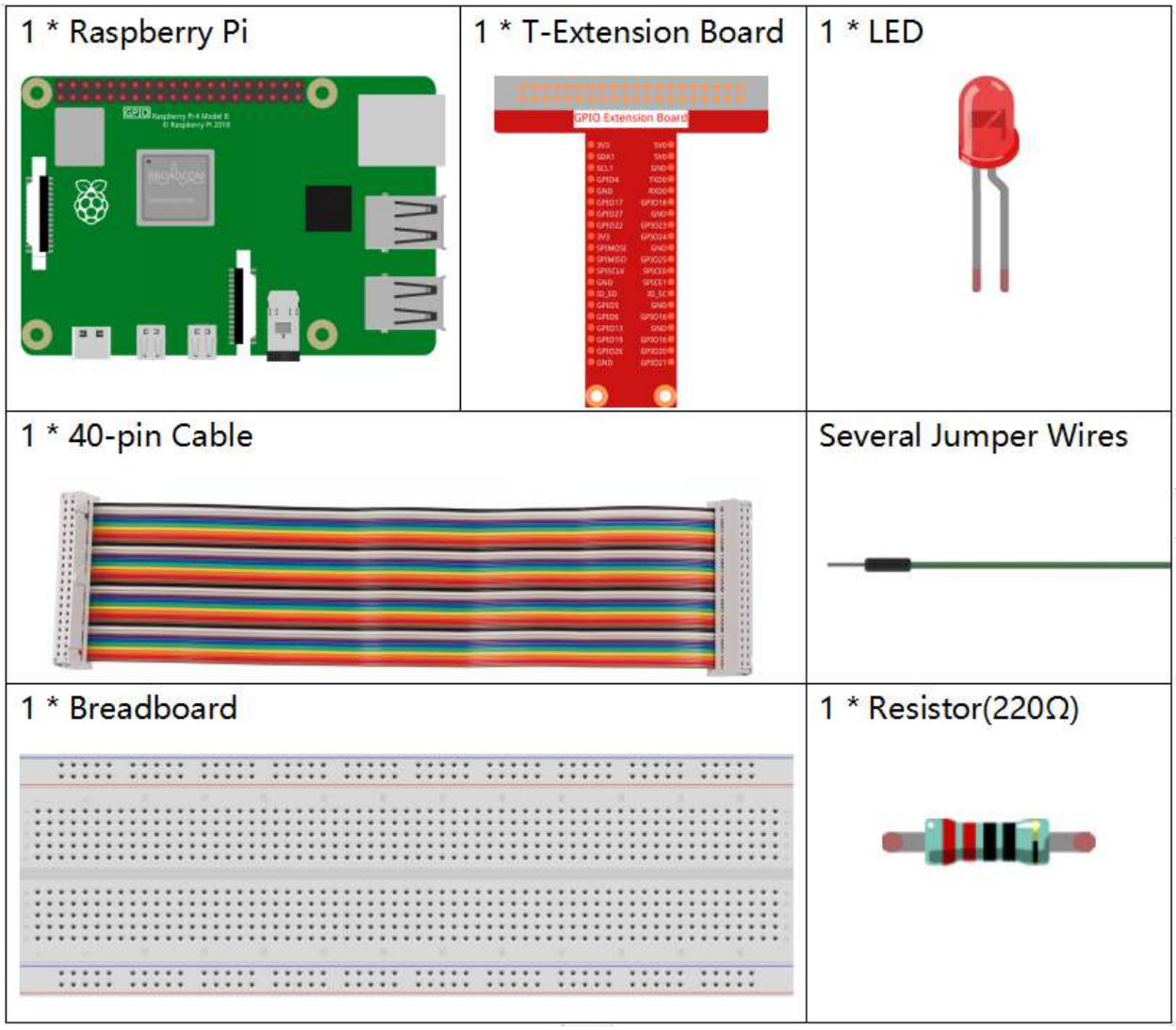
7.2.1 1.1 Displays

1.1.1 Blinking LED

Introduction

In this project, we will learn how to make a blinking LED by programming. Through your settings, your LED can produce a series of interesting phenomena. Now, go for it.

Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*

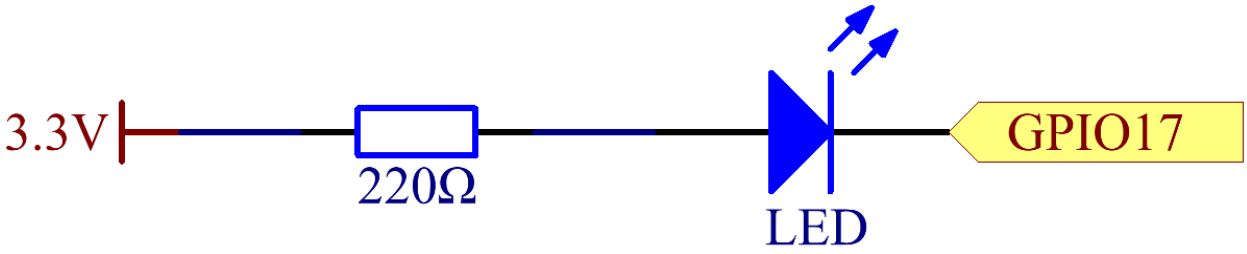
Schematic Diagram

In this experiment, connect a 220 resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to GPIO17 of Raspberry Pi. Therefore, to turn on an LED, we need to make GPIO17 low (0V) level. We can get this phenomenon by programming.

Note: **Pin11** refers to the 11th pin of the Raspberry Pi from left to right, and its corresponding **wiringPi** and **BCM** pin numbers are shown in the following table.

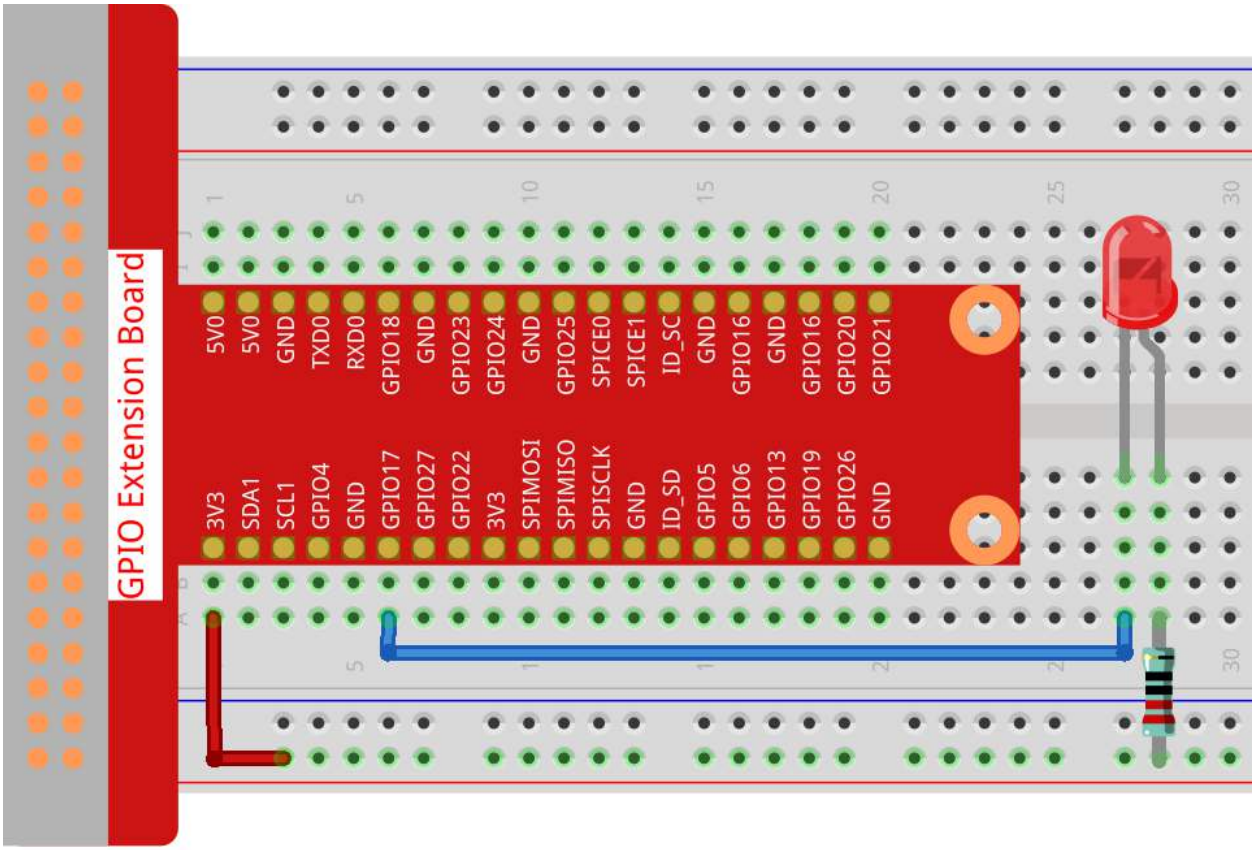
In the C language related content, we make GPIO0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi, Pin 11.

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



Experimental Procedures

Step 1: Build the circuit.

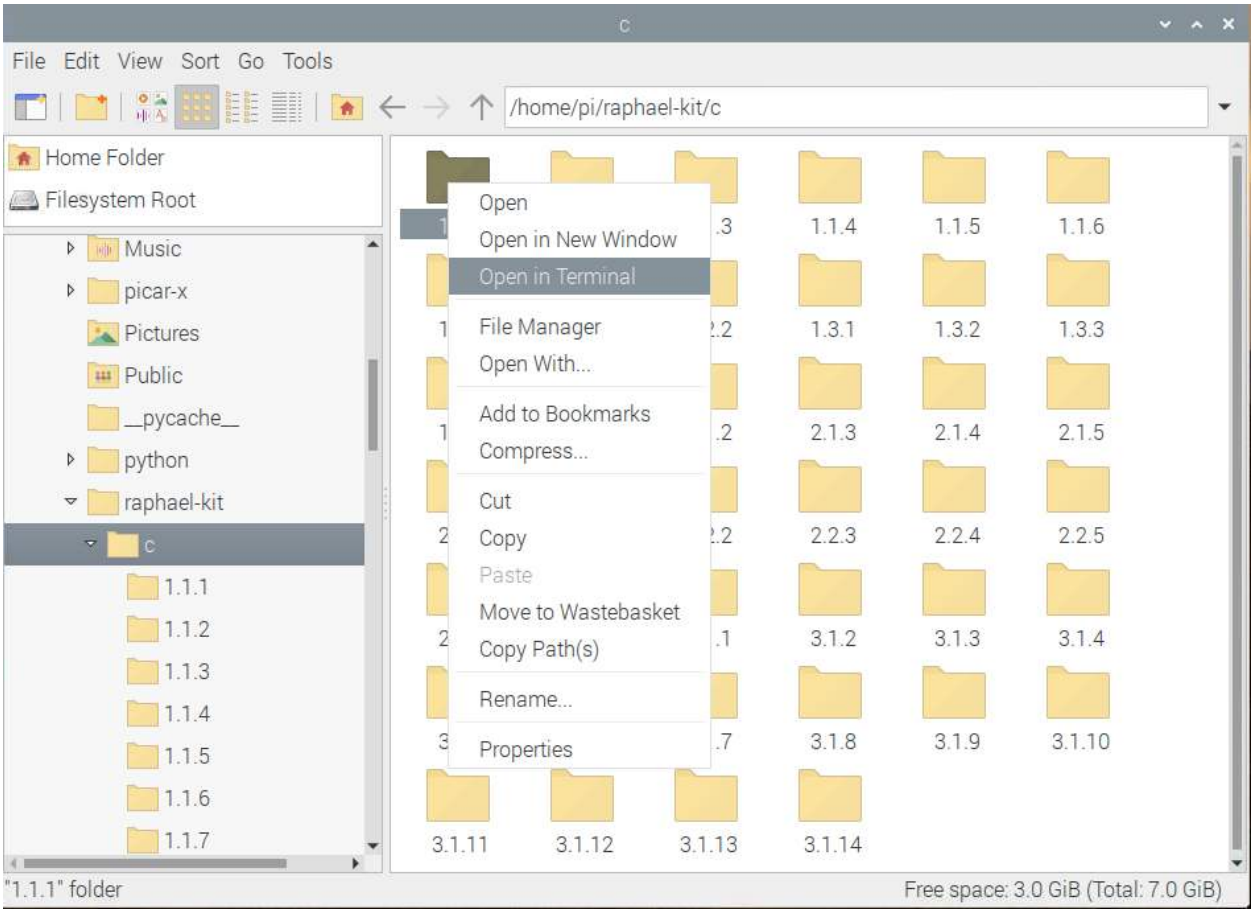


Step 2: Go to the folder of the code.

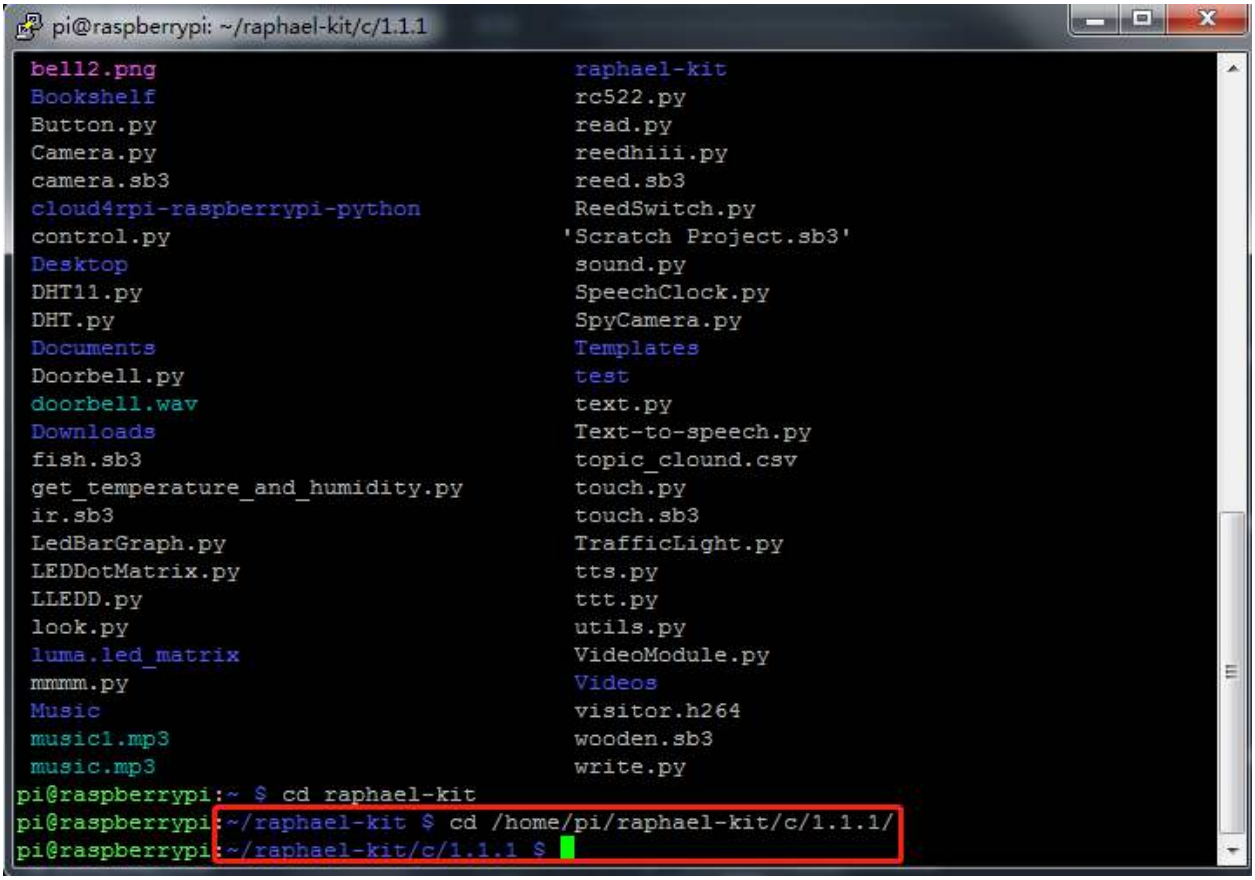
1) If you use a screen, you're recommended to take the following steps.

Go to **/home/pi/** and find the folder **raphael-kit**.

Find **C** in the folder, right-click on it and select **Open in Terminal**.



Then a window will pop up as shown below. So now you've entered the path of the code **1.1.1_BlinkingLed.c** .



In the following projects, we will use command to enter the code file instead of right-clicking. But you can choose the method you prefer.

2) If you log into the Raspberry Pi remotely, use `cd` to change directory:

```
cd /home/pi/raphael-kit/c/1.1.1/
```

Note: Change directory to the path of the code in this experiment via cd.

In either way, now you are in the folder C. The subsequent procedures based on these two methods are the same. Let's move on.

Step 3: Compile the code

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

Note: gcc is GNU Compiler Collection. Here, it functions like compiling the C language file `1.1.1_BlinkingLed.c` and outputting an executable file.

In the command, `-o` means outputting (the character immediately following -o is the filename output after compilation, and an executable named `BlinkingLed` will generate here) and `-lwiringPi` is to load the library wiringPi (`l` is the abbreviation of library).

Step 4: Run the executable file output in the previous step.

```
sudo ./BlinkingLed
```

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

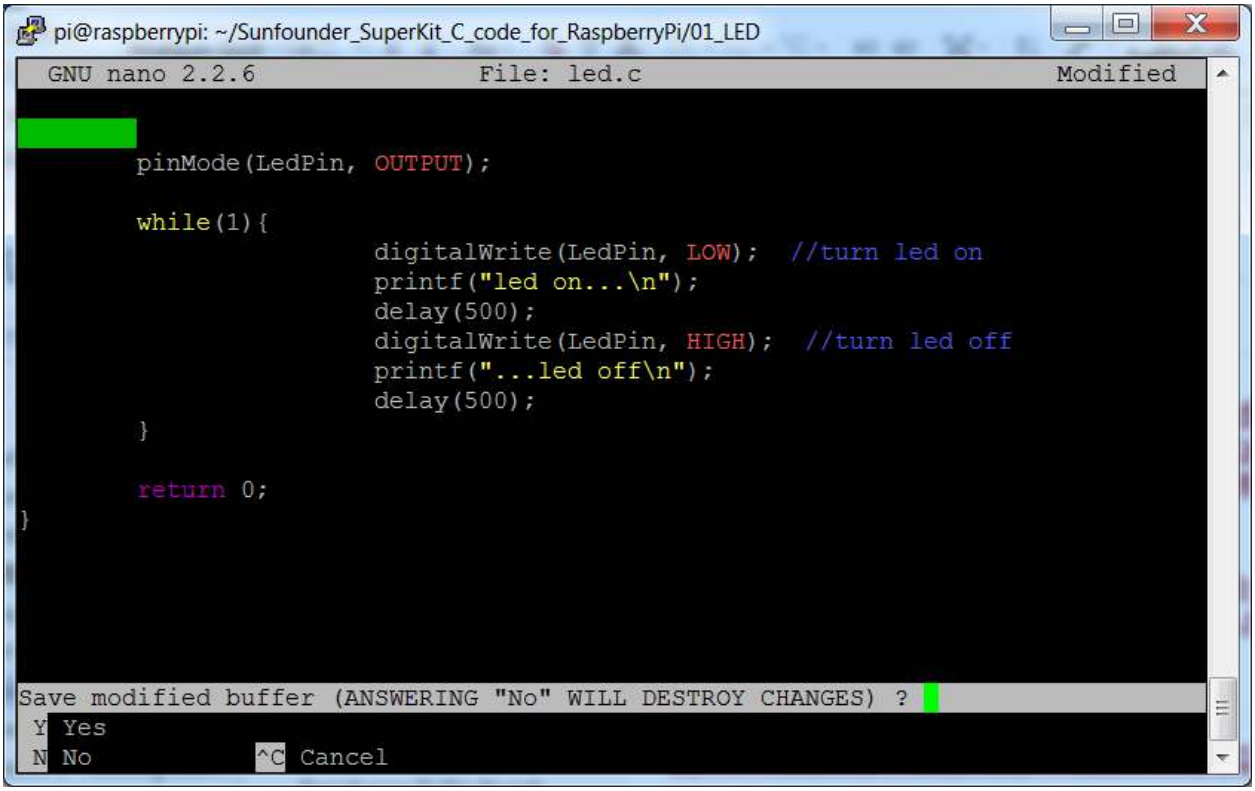
To control the GPIO, you need to run the program, by the command, `sudo` (superuser do). The command `./` indicates the current directory. The whole command is to run the `BlinkingLed` in the current directory.

After the code runs, you will see the LED flashing.

If you want to edit the code file `1.1.1_BlinkingLed.c`, stop the code and then type the following command to open it:

```
nano 1.1.1_BlinkingLed.c
```

Press `Ctrl+X` to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in `Y` (save) or `N` (don't save). Then press `Enter` to exit. Repeat `Step 3` and `Step 4` to see the effect after modifying.



Code

The program code is shown as follows:

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin      0
int main(void)
{
   // When initialize wiring failed, print message to screen
```

(continues on next page)

(continued from previous page)

```
    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 Explanation

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

The hardware drive library is designed for the C language of Raspberry Pi. Adding this library is conducive to the initialization of hardware, and the output of I/O ports, PWM, etc.

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

Standard I/O library. The pintf function used for printing the data displayed on the screen is realized by this library. There are many other performance functions for you to explore.

```
#define LedPin 0
```

Pin GPIO17 of the T_Extension Board is corresponding to the GPIO0 in wiringPi. Assign GPIO0 to LedPin, LedPin represents GPIO0 in the code later.

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

This initialises wiringPi and assumes that the calling program is going to be using the wiringPi pin numbering scheme.

This function needs to be called with root privileges. When initialize wiring failed, print message to screen. The function `return` is used to jump out of the current function. Using return in `main()` function will end the program.

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

Set `LedPin` as output to write value to it.

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

Set GPIO0 as 0V (low level). Since the cathode of LED is connected to GPIO0, thus the LED will light up if GPIO0 is set low. On the contrary, set GPIO0 as high level, LED will go out.

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

The printf function is a standard library function and its function prototype is in the header file `stdio.h`.

The general form of the call is: `printf(" format control string ", output table columns)`. The format control string is used to specify the output format, which is divided into format string and non-format string. The format string starts with `%` followed by format characters, such as `%d` for decimal integer output. Unformatted strings are printed as prototypes. What is used here is a non-format string, followed by `\n` that is a newline character, representing automatic line wrapping after printing a string.

```
delay(500);
```

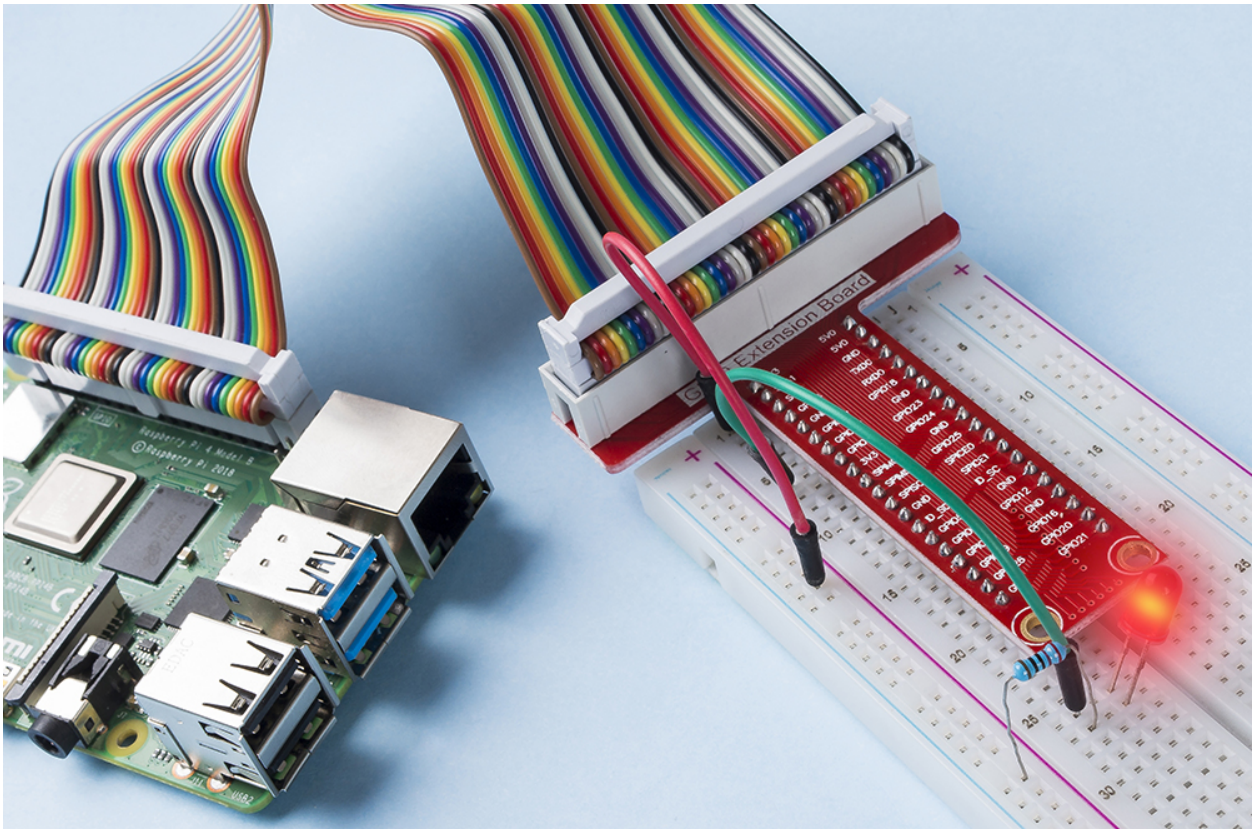
Keeps the current HIGH or LOW state for 500ms.

This is a function that suspends the program for a period of time. And the speed of the program is determined by our hardware. Here we turn on or off the LED. If there is no delay function, the program will run the whole program very fast and continuously loop. So we need the delay function to help us write and debug the program.

```
return 0;
```

Usually, it is placed behind the main function, indicating that the function returns 0 on successful execution.

Phenomenon Picture

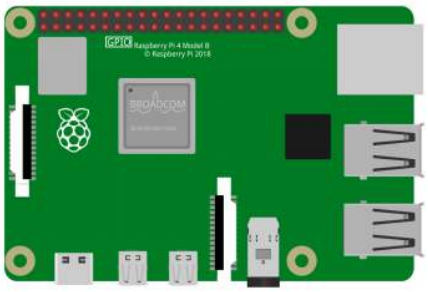
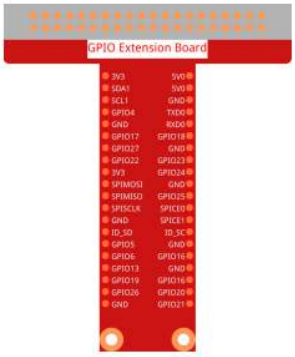
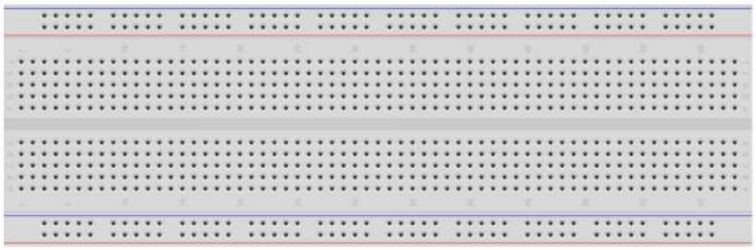


1.1.2 RGB LED

Introduction

In this project, we will control an RGB LED to flash various colors.

Components

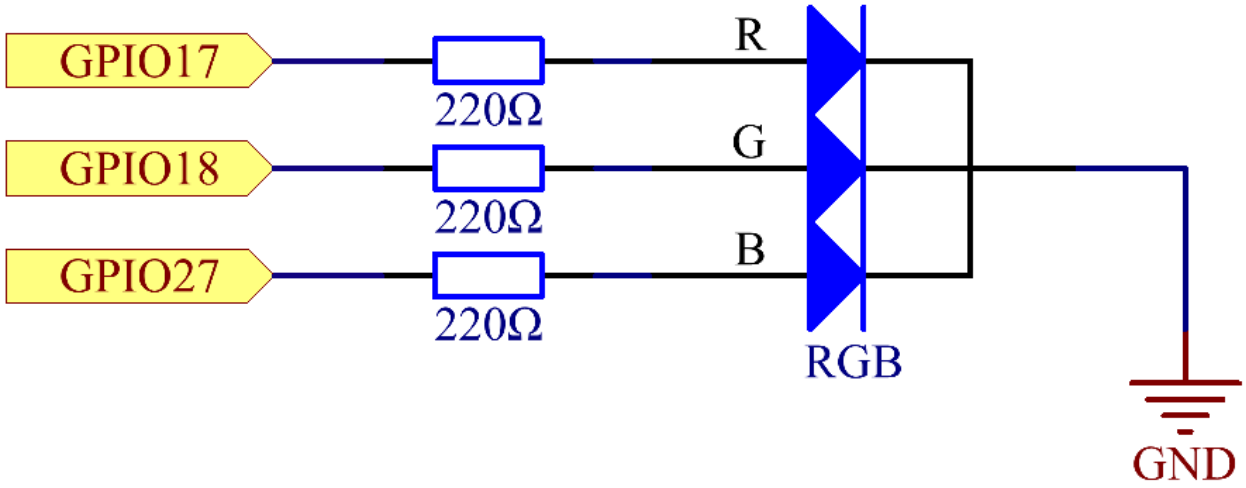
1 * Raspberry Pi 	1 * T-Extension Board 	1 * RGB LED
1 * 40-pin Cable		Several Jumper Wires
1 * Breadboard 		3 * Resistor(220Ω)

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*

Schematic Diagram

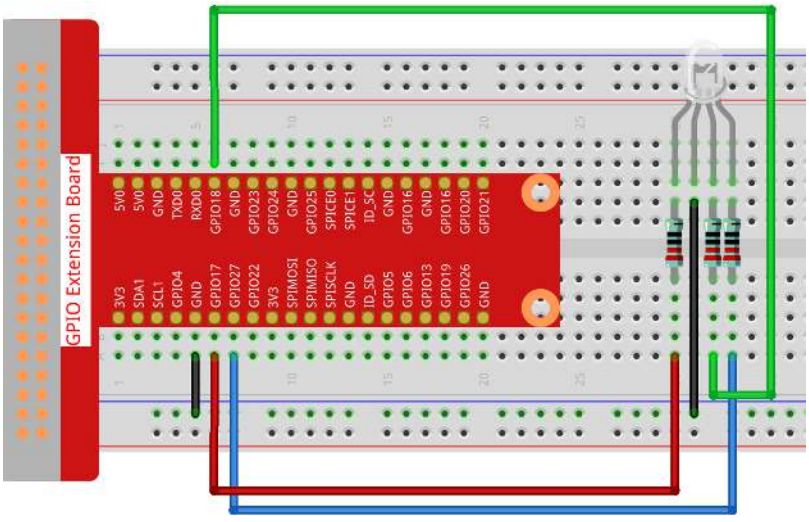
After connecting the pins of R, G, and B to a current limiting resistor, connect them to the GPIO17, GPIO18, and GPIO27 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/1.1.2/
```

Step 3: Compile the code.

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

Note: When the instruction `gcc` is executed, if `-o` is not called, then the executable file is named `a.out`.

Step 4: Run the executable file.

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

After the code runs, you will see that RGB displays red, green, blue, yellow, pink, and cyan.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

```
#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
```

(continues on next page)

(continued from previous page)

```
        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 Explanation

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

Library used for realizing the pwm function of the software.

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

The function is to use software to create a PWM pin, set its period between 0x100us-100x100us.

The prototype of the function softPwmCreate(LedPinRed, 0, 100) is as follows:

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

- **Parameter pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.
- **Parameter initialValue:** The initial pulse width is that initialValue times100us.
- **Parameter pwmRange:** the period of PWM is that 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);
}
```

This function is to set the colors of the LED. Using RGB, the formal parameter **r_val** represents the luminance of the red one, **g_val** of the green one, **b_val** of the blue one.

The prototype of the function softPwmWrite(LedPinBlue, b_val) is as follows

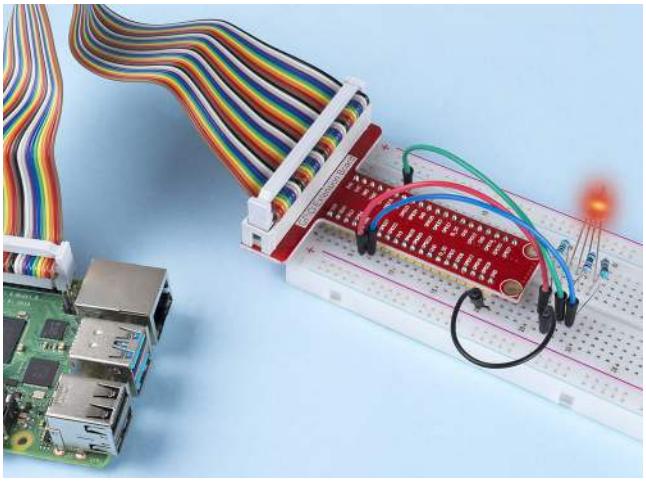
```
void softPwmWrite (int pin, int value) ;
```

- **Parameter pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.
- **Parameter Value:** The pulse width of PWM is value times 100us. Note that value can only be less than pwmRange defined previously, if it is larger than pwmRange, the value will be given a fixed value, pwmRange.

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

Call the function defined before. Write 0xff into LedPinRed and 0x00 into LedPinGreen and LedPinBlue. Only the Red LED lights up after running this code. If you want to light up LEDs in other colors, just modify the parameters.

Phenomenon Picture

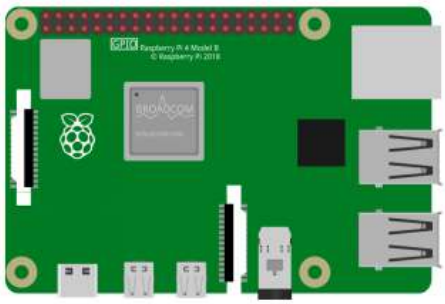
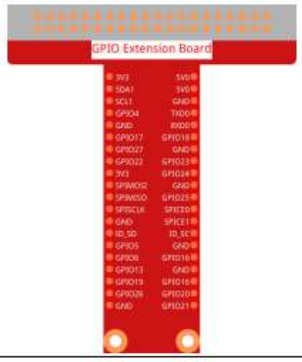
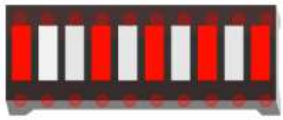
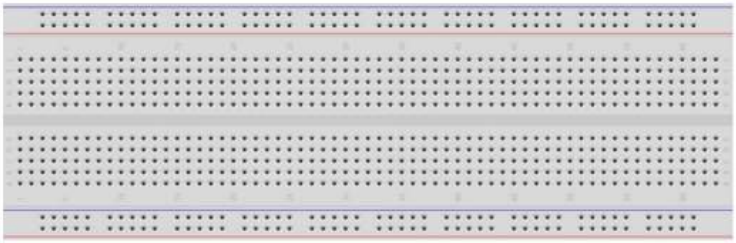


1.1.3 LED Bar Graph

Introduction

In this project, we sequentially illuminate the lights on the LED Bar Graph.

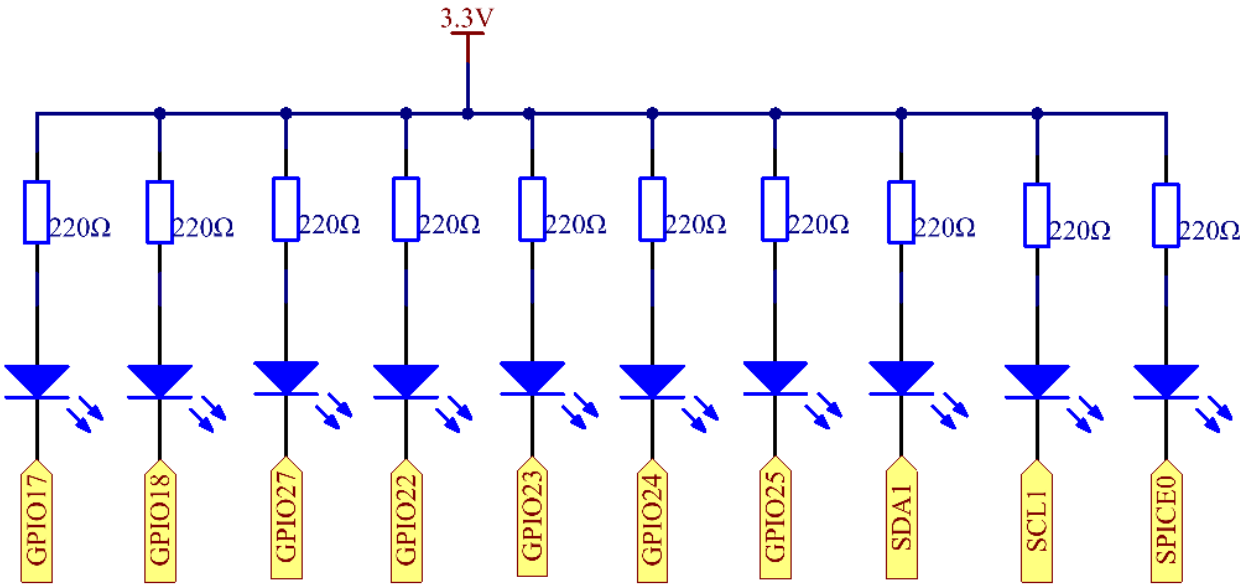
Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*

Schematic Diagram

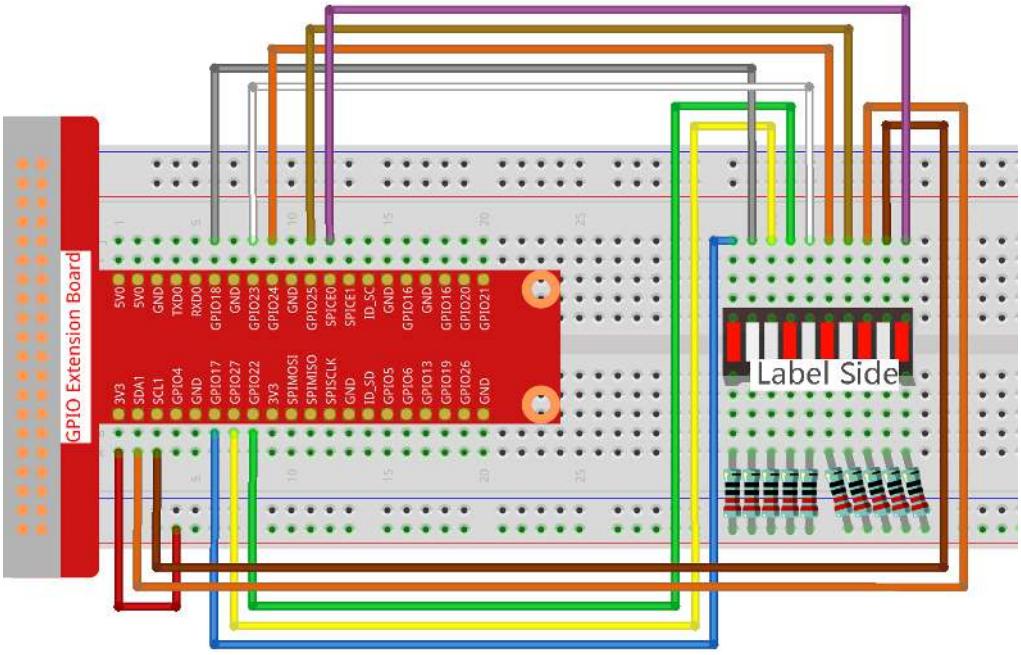
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3	8	2
SCL1	Pin 5	9	3
SPICE0	Pin 24	10	8



Experimental Procedures

Step 1: Build the circuit.

Note: Pay attention to the direction when connecting. If you connect it backwards, it will not light up.



Step 2: Go to the folder of the code.

```
cd ~/raphael-kit/c/1.1.3/
```

Step 3: Compile the code.

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

Step 4: Run the executable file.

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

After the code runs, you will see the LEDs on the LED bar turn on and off regularly.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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

(continues on next page)

(continued from previous page)

```
        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 Explanation

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

Create an array and assign it to the pin number corresponding to the LED Bar Graph (0,1,2,3,4,5,6,8,9,10) and the array will be used to control the LED.

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

Let the LED on the odd digit of the LED Bar Graph light on in turn.

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

(continues on next page)

(continued from previous page)

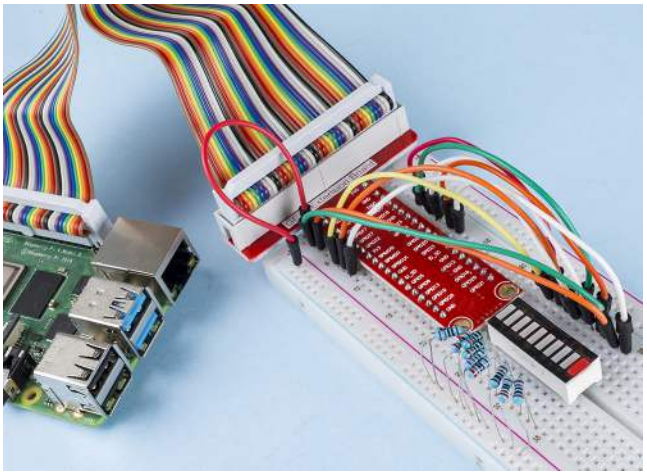
```
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
```

Make the LED on the even digit of the LED Bar Graph light on in turn.

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

Let the LED on the LED Bar Graph light on one by one.

Phenomenon Picture

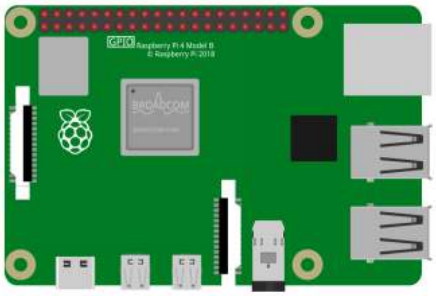
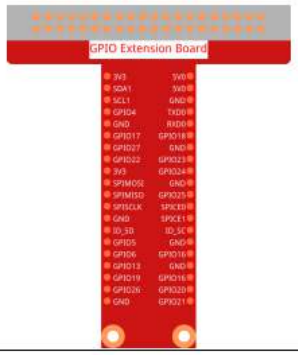
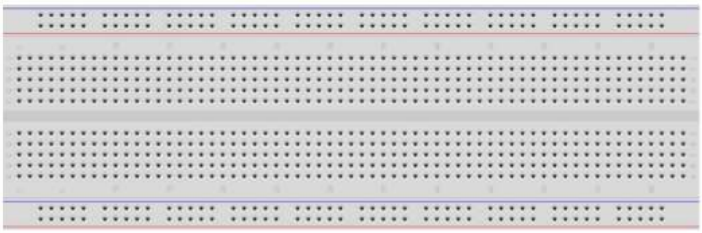


1.1.4 7-segment Display

Introduction

Let's try to drive a 7-segment display to show a figure from 0 to 9 and A to F.

Components

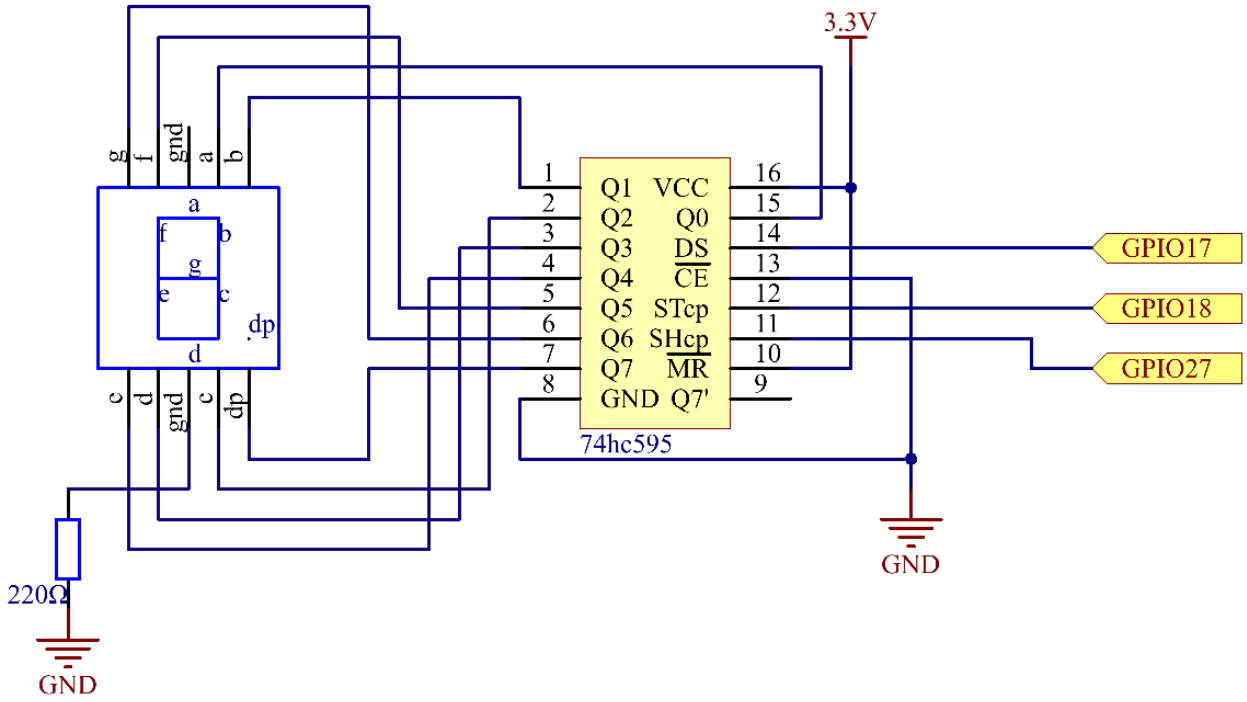
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 7-segment display
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * 74HC595
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *7-segment Display*
- *74HC595*

Schematic Diagram

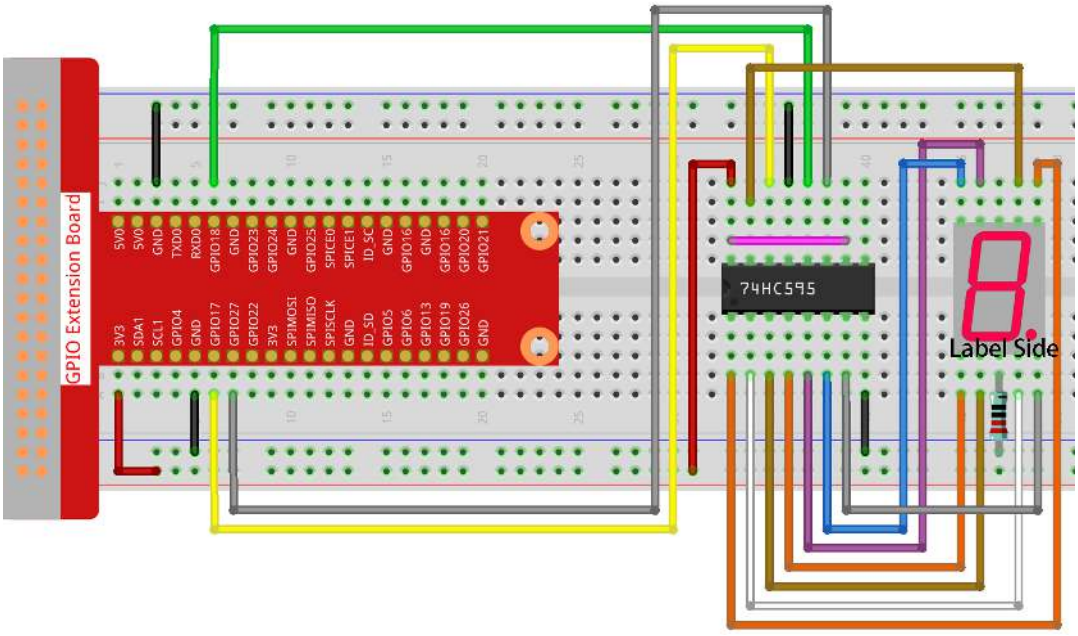
Connect pin ST_CP of 74HC595 to Raspberry Pi GPIO18, SH_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST_CP (the clock input of the memory) is at the rising edge. Then you can control the states of SH_CP and ST_CP via the Raspberry Pi GPIOs to transform serial data input into parallel data output so as to save Raspberry Pi GPIOs and drive the display.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/c/1.1.4/
```

Step 3: Compile.

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

Step 4: Run the executable file above.

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

After the code runs, you'll see the 7-segment display display 0-9, A-F.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
```

(continues on next page)

(continued from previous page)

```
}

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

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

Code Explanation

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

A segment code array from 0 to F in Hexadecimal (Common cathode).

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

Set ds, st_cp, sh_cp three pins to OUTPUT, and the initial state as 0.

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

To assign 8 bit value to 74HC595's shift register.

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

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when i=2, 0x3f will shift left(<<) 2 bits.

1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true.

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

SRCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge pulse, then shift the DS date to shift register.

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

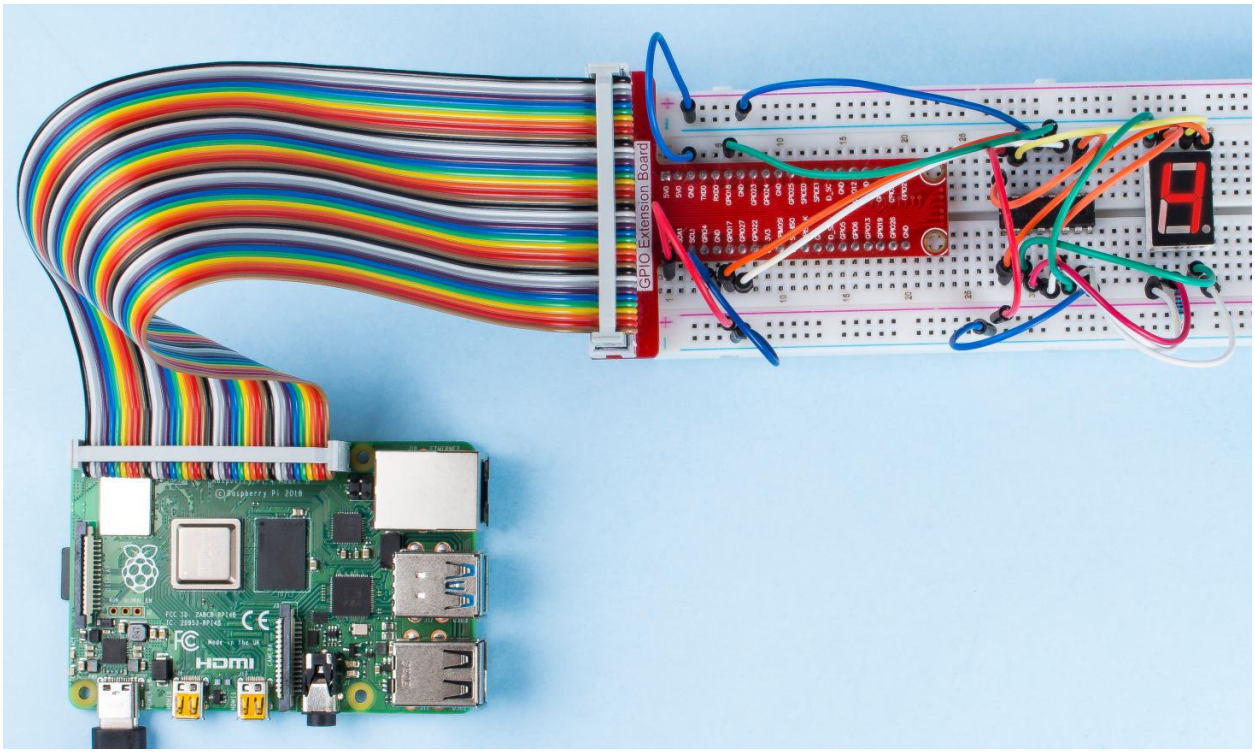
RCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge, then shift data from shift register to storage register.

```
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);
        }
    }
```

In this for loop, we use `%1X` to output i as a hexadecimal number. Apply i to find the corresponding segment code in the `SegCode[]` array, and employ `hc595_shift()` to pass the SegCode into 74HC595's shift register.

Note: The hexadecimal format of number 0~15 are (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

Phenomenon Picture

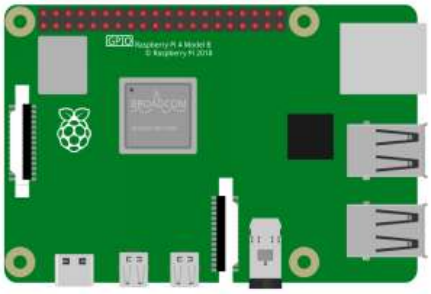
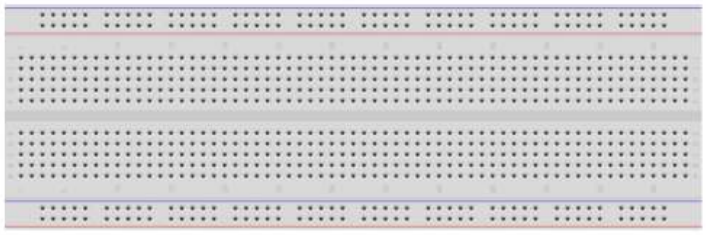


1.1.5 4-Digit 7-Segment Display

Introduction

Next, follow me to try to control the 4-digit 7-segment display.

Components

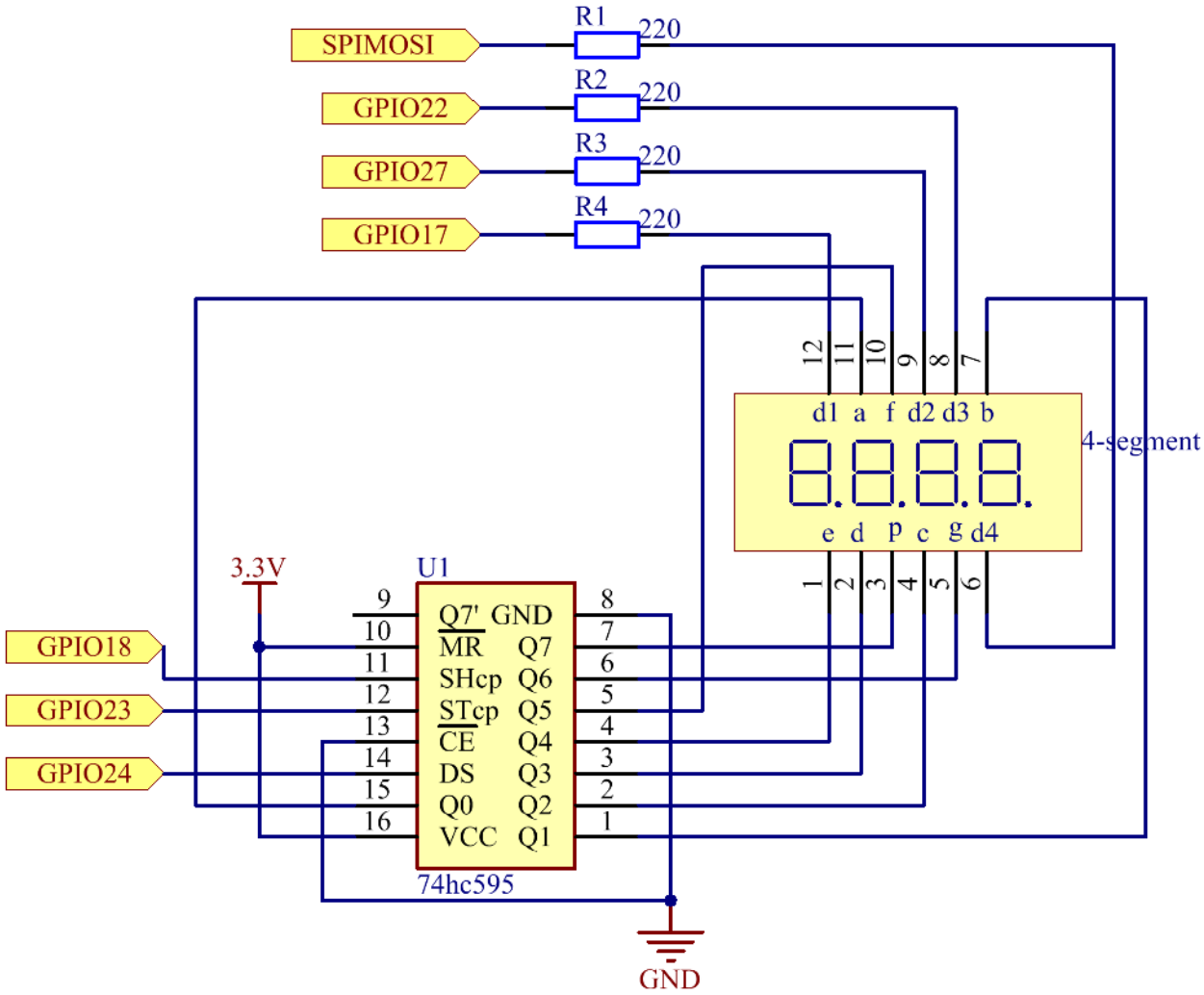
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
</td><td>1 * 4-Digit 7-Segment Display
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>1 * 74HC595
</td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>4 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td>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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*

Note: In this projiect, for the 4-Digit 7-Segment Display we should use BS model,if you use AS model it may not light up.

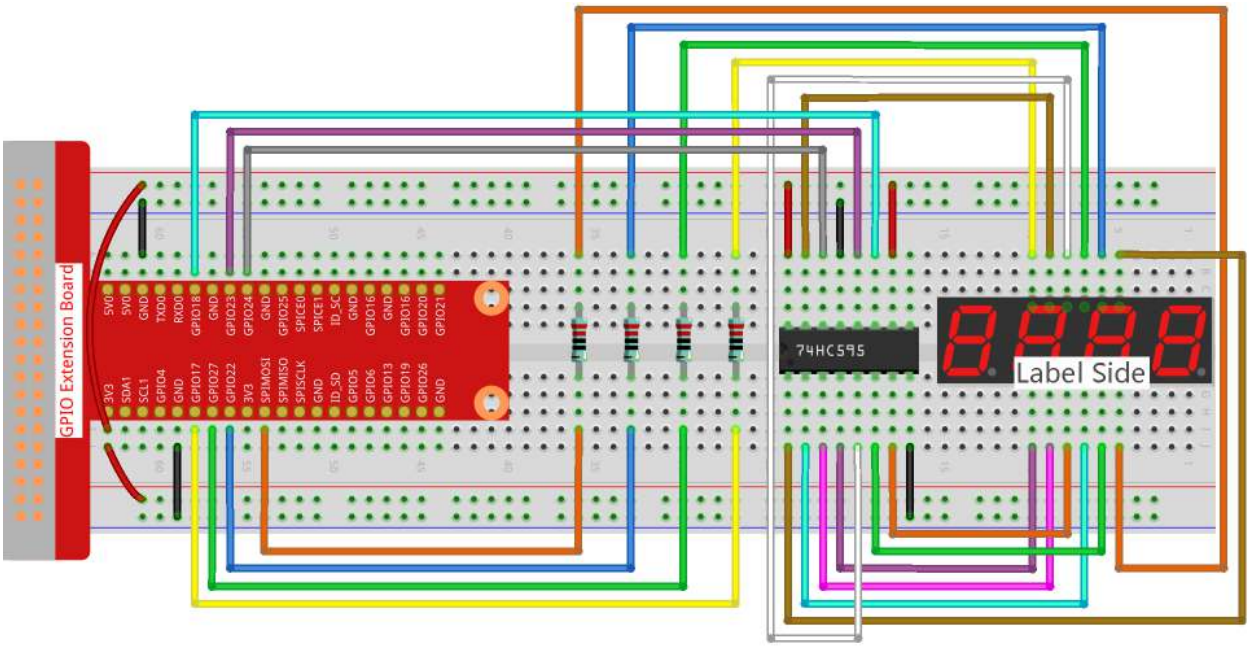
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/1.1.5/
```

Step 3: Compile the code.

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

Step 4: Run the executable file.

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

After the code runs, the program takes a count, increasing by 1 per second, and the 4-digit 7-segment display displays the count.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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
```

(continues on next page)

(continued from previous page)

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

int counter = 0;

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

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

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

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

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

    clearDisplay();
```

(continues on next page)

(continued from previous page)

```
    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 Explanation

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

These four pins control the common anode pins of the four-digit 7-segment display.

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

A segment code array from 0 to 9 in Hexadecimal (Common anode).

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

(continues on next page)

(continued from previous page)

```
    digitalWrite(placePin[digit], 1);
}
```

Select the place of the value. there is only one place that should be enable each time. The enabled place will be written high.

```
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]);
    }
}
```

The functionis used to set the number displayed on the 4-digit 7-segment display.

- `clearDisplay()` write in 11111111 to turn off these eight LEDs on 7-segment display so as to clear the displayed content.
- `pickDigit(0)` pick the fourth 7-segment display.
- `hc595_shift(number[counter%10])` the number in the single digit of counter will display on the forth segment.

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

This is a system-provided function, the prototype of code is:

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

After executing the `signal()` , once the process receives the corresponding signum (in this case SIGALRM), it immediately pauses the existing task and processes the set function (in this case `timer(sig)`).

```
alarm(1);
```

This is also a system-provided function. The code prototype is:

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

It generates a SIGALRM signal after a certain number of seconds.

```
void timer(int timer1)
{
    if (timer1 == SIGALRM)
```

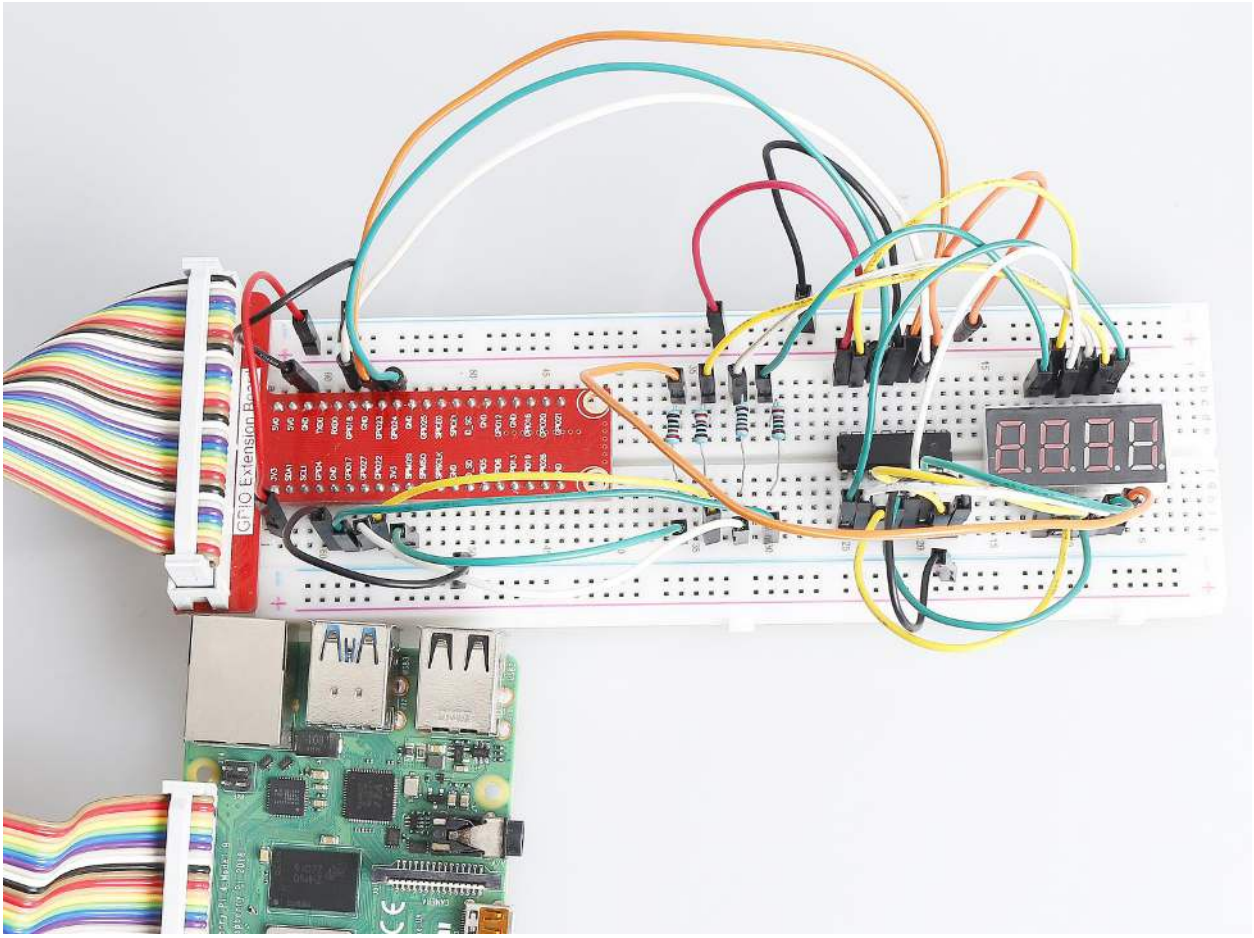
(continues on next page)

(continued from previous page)

```
    {
        counter++;
        alarm(1);
        printf("%d\n", counter);
    }
}
```

We use the functions above to implement the timer function. After the `alarm()` generates the SIGALRM signal, the timer function is called. Add 1 to counter, and the function, `alarm(1)` will be repeatedly called after 1 second.

Phenomenon Picture

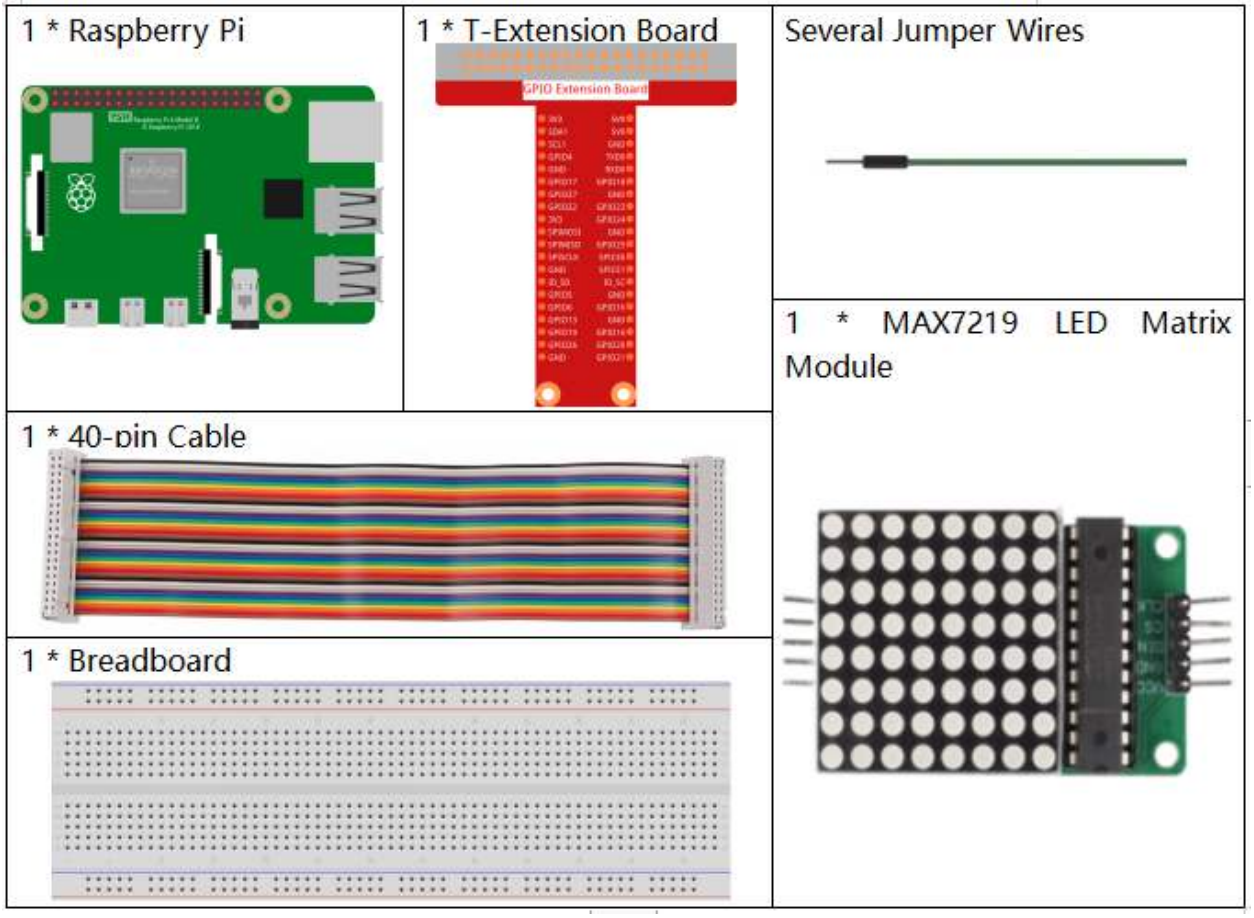


1.1.6 LED Dot Matrix Module

Introduction

In this project, you will learn about LED Matrix Module. LED Matrix Module uses the MAX7219 driver to drive the 8 x 8 LED Matrix.

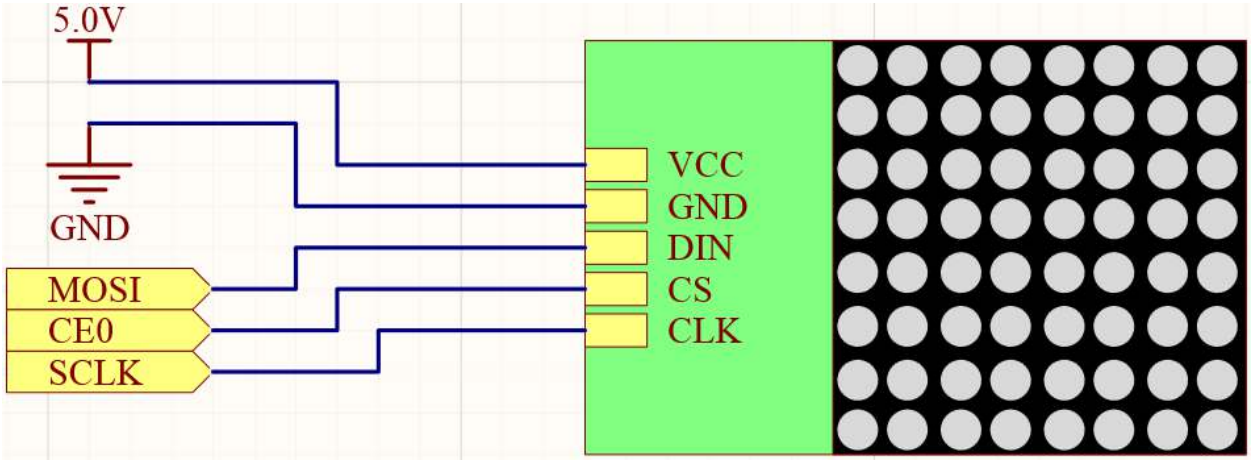
Components



- *GPIO Extension Board*
- *Breadboard*
- *LED Matrix Module*

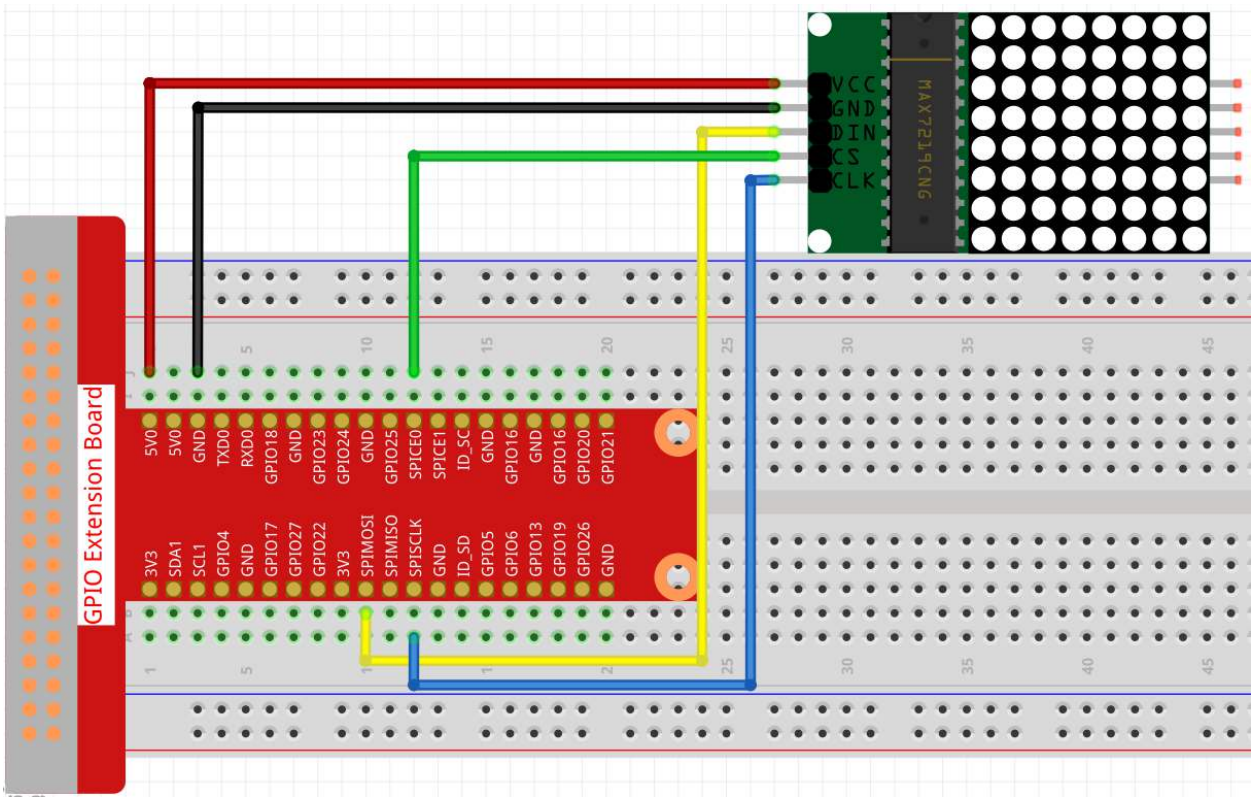
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
SPIMOSI	Pin 19	12	MOSI
SPICE0	pin 24	10	CE0
SPISCLK	Pin 23	14	SCLK



Experimental Procedures

Step 1: Build the circuit.



Note: Turn on the SPI before starting the experiment, refer to *SPI Configuration* for details. And the *BCM2835* library is also needed.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/1.1.6/
```

Step 3: Compile the code.

```
make
```

Step 4:: Run the executable file.

```
sudo ./1.1.6_LedMatrix
```

After running the code, the LED Dot Matrix displays from 0 to 9 and A to Z in sequence.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

```
#include <bcm2835.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define uchar unsigned char
#define uint unsigned int

#define Max7219_pinCS  RPI_GPIO_P1_24

uchar disp1[36][8]={
{0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C},//0
{0x08,0x18,0x28,0x08,0x08,0x08,0x08,0x08},//1
{0x7E,0x2,0x2,0x7E,0x40,0x40,0x40,0x7E},//2
{0x3E,0x2,0x2,0x3E,0x2,0x2,0x3E,0x0},//3
{0x8,0x18,0x28,0x48,0xFE,0x8,0x8,0x8},//4
{0x3C,0x20,0x20,0x3C,0x4,0x4,0x3C,0x0},//5
{0x3C,0x20,0x20,0x3C,0x24,0x24,0x3C,0x0},//6
{0x3E,0x22,0x4,0x8,0x8,0x8,0x8,0x8},//7
{0x0,0x3E,0x22,0x22,0x3E,0x22,0x22,0x3E},//8
{0x3E,0x22,0x22,0x3E,0x2,0x2,0x2,0x3E},//9
{0x8,0x14,0x22,0x3E,0x22,0x22,0x22,0x22},//A
{0x3C,0x22,0x22,0x3E,0x22,0x22,0x3C,0x0},//B
{0x3C,0x40,0x40,0x40,0x40,0x40,0x3C,0x0},//C
{0x7C,0x42,0x42,0x42,0x42,0x42,0x7C,0x0},//D
{0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x7C},//E
{0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x40},//F
{0x3C,0x40,0x40,0x40,0x40,0x44,0x44,0x3C},//G
{0x44,0x44,0x44,0x7C,0x44,0x44,0x44,0x44},//H
{0x7C,0x10,0x10,0x10,0x10,0x10,0x10,0x7C},//I
{0x3C,0x8,0x8,0x8,0x8,0x8,0x48,0x30},//J
{0x0,0x24,0x28,0x30,0x20,0x30,0x28,0x24},//K
```

(continues on next page)

(continued from previous page)

```
{0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x7C},//L
{0x81,0xC3,0xA5,0x99,0x81,0x81,0x81,0x81},//M
{0x0,0x42,0x62,0x52,0x4A,0x46,0x42,0x0},//N
{0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C},//O
{0x3C,0x22,0x22,0x22,0x3C,0x20,0x20,0x20},//P
{0x1C,0x22,0x22,0x22,0x22,0x26,0x22,0x1D},//Q
{0x3C,0x22,0x22,0x22,0x3C,0x24,0x22,0x21},//R
{0x0,0x1E,0x20,0x20,0x3E,0x2,0x2,0x3C},//S
{0x0,0x3E,0x8,0x8,0x8,0x8,0x8,0x8},//T
{0x42,0x42,0x42,0x42,0x42,0x42,0x22,0x1C},//U
{0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18},//V
{0x0,0x49,0x49,0x49,0x49,0x2A,0x1C,0x0},//W
{0x0,0x41,0x22,0x14,0x8,0x14,0x22,0x41},//X
{0x41,0x22,0x14,0x8,0x8,0x8,0x8,0x8},//Y
{0x0,0x7F,0x2,0x4,0x8,0x10,0x20,0x7F},//Z
};

void Delay_xms(uint x)
{
    bcm2835_delay(x);
}
//-------------------------

void Write_Max7219_byte(uchar DATA)
{
    bcm2835_gpio_write(Max7219_pinCS,LOW);
    bcm2835_spi_transfer(DATA);
}

void Write_Max7219(uchar address1,uchar dat1)
{
    bcm2835_gpio_write(Max7219_pinCS,LOW);
    Write_Max7219_byte(address1);
    Write_Max7219_byte(dat1);
    bcm2835_gpio_write(Max7219_pinCS,HIGH);
}

void Init_MAX7219()
{
    Write_Max7219(0x09,0x00);
    Write_Max7219(0x0a,0x03);
    Write_Max7219(0x0b,0x07);
    Write_Max7219(0x0c,0x01);
    Write_Max7219(0x0f,0x00);
}

void Init_BCM2835()
{
    bcm2835_spi_begin();
    bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);
    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);
    bcm2835_gpio_fsel(Max7219_pinCS, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_write(disp1[0][0],HIGH);
}

int main(void)
```

(continues on next page)

(continued from previous page)

```
{
    uchar i,j;

    if (!bcm2835_init())
    {
        printf("Unable to init bcm2835.\n");
        return 1;
    }
    Init_BCM2835();
    Delay_xms(50);
    Init_MAX7219();
    while(1)
    {
        for(j=0;j<36;j++)
        {
            for(i=1;i<9;i++)
            {
                Write_Max7219(i,disp1[j][i-1]);
            }
            Delay_xms(1000);
        }
    }
    // bcm2835_spi_end();
    // bcm2835_close();
    return 0;
}
```

Code Explanation

```
#define Max7219_pinCS  24
```

The cs pin of the LED Dot Matrix is connected to pin24.

Note: When you have multiple devices that need spi communication, just connect the cs pins on different pins.

```
if (!bcm2835_init())
    {
            printf("Unable to init bcm2835.\n");
            return 1;
    }
```

Check if the bcm2835 library is successfully installed, if not, print the message “Unable to init bcm2835”.

```
Init_BCM2835();
    Delay_xms(50);
    Init_MAX7219();
```

Initialize libraries and module.

```
while(1)
{
    for(j=0;j<36;j++)
    {
        for(i=1;i<9;i++)
        {
```

(continues on next page)

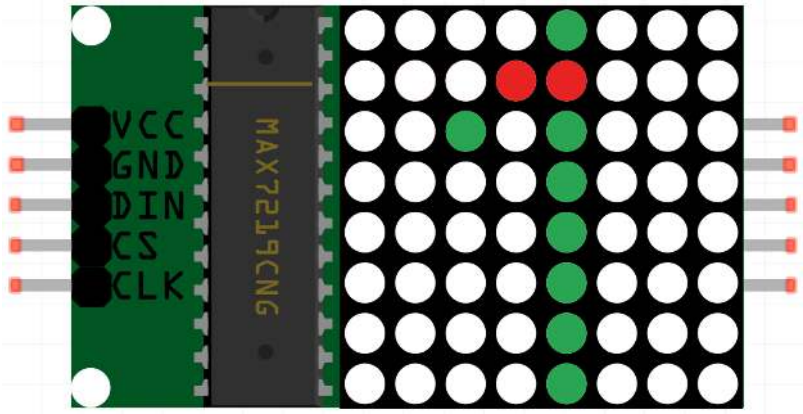
(continued from previous page)

```
            Write_Max7219(i,disp1[j][i-1]);
        }
        Delay_xms(1000);
    }
}
```

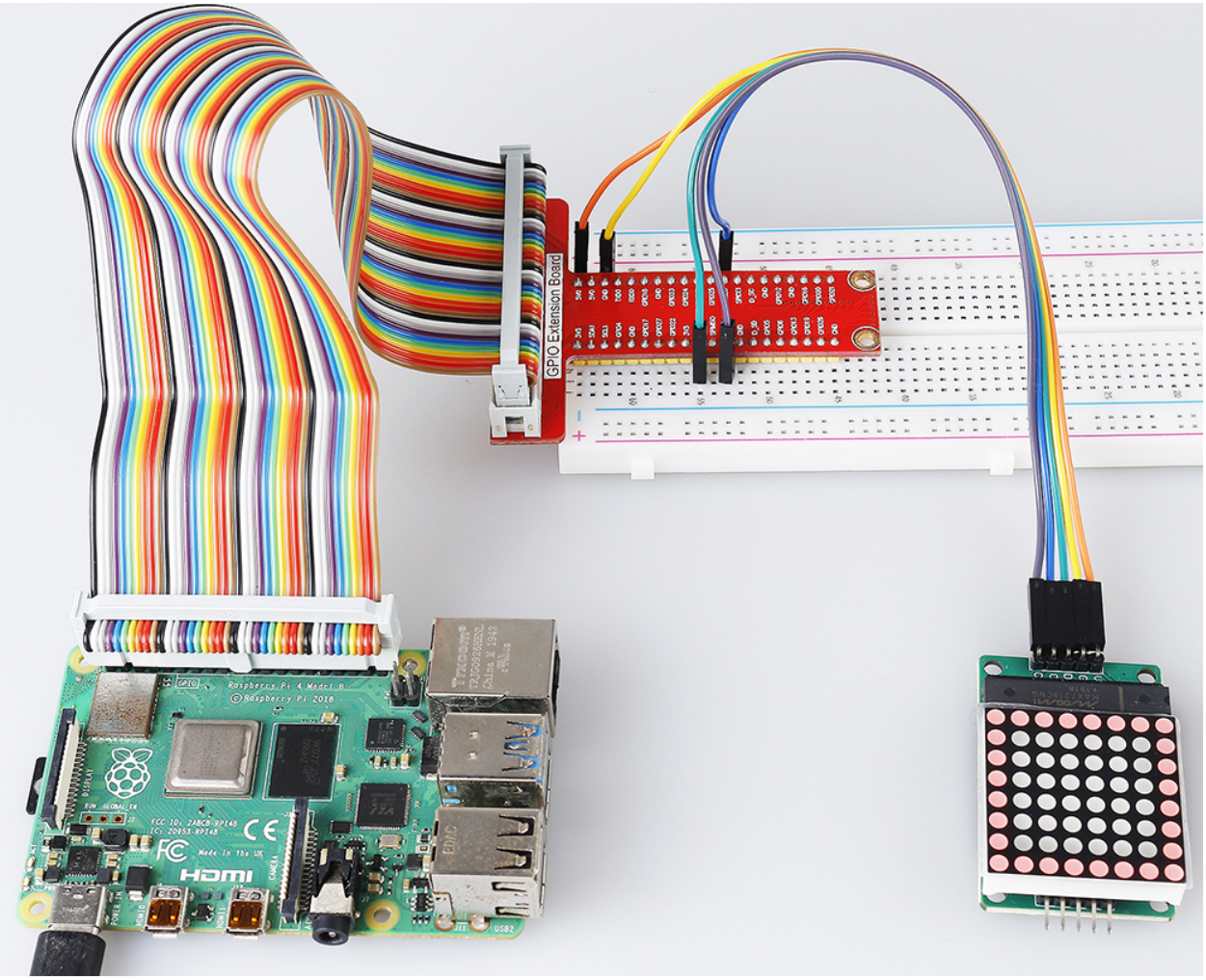
The `Write_Max7219()` function allows you to display the specified character on the LED Dot Matrix, where the first parameter inputs the row in which it is displayed, and the second parameter inputs an 8-bit binary number or a hexadecimal number that indicates the light on or off in that row (0 means off, 1 means lit).

The variable `j` represents the rows in the array `disp1[]` (35 rows) and the variable `i` represents the column (8 columns).

For example, when j=1 and i=2, the value `disp1[1][1]` (0x18) is displayed on the dot matrix. i loops 8 times to display the full 1 on the dot matrix. After 35 cycles of j, 0-9 and A-Z are displayed on the dot matrix.



Phenomenon Picture

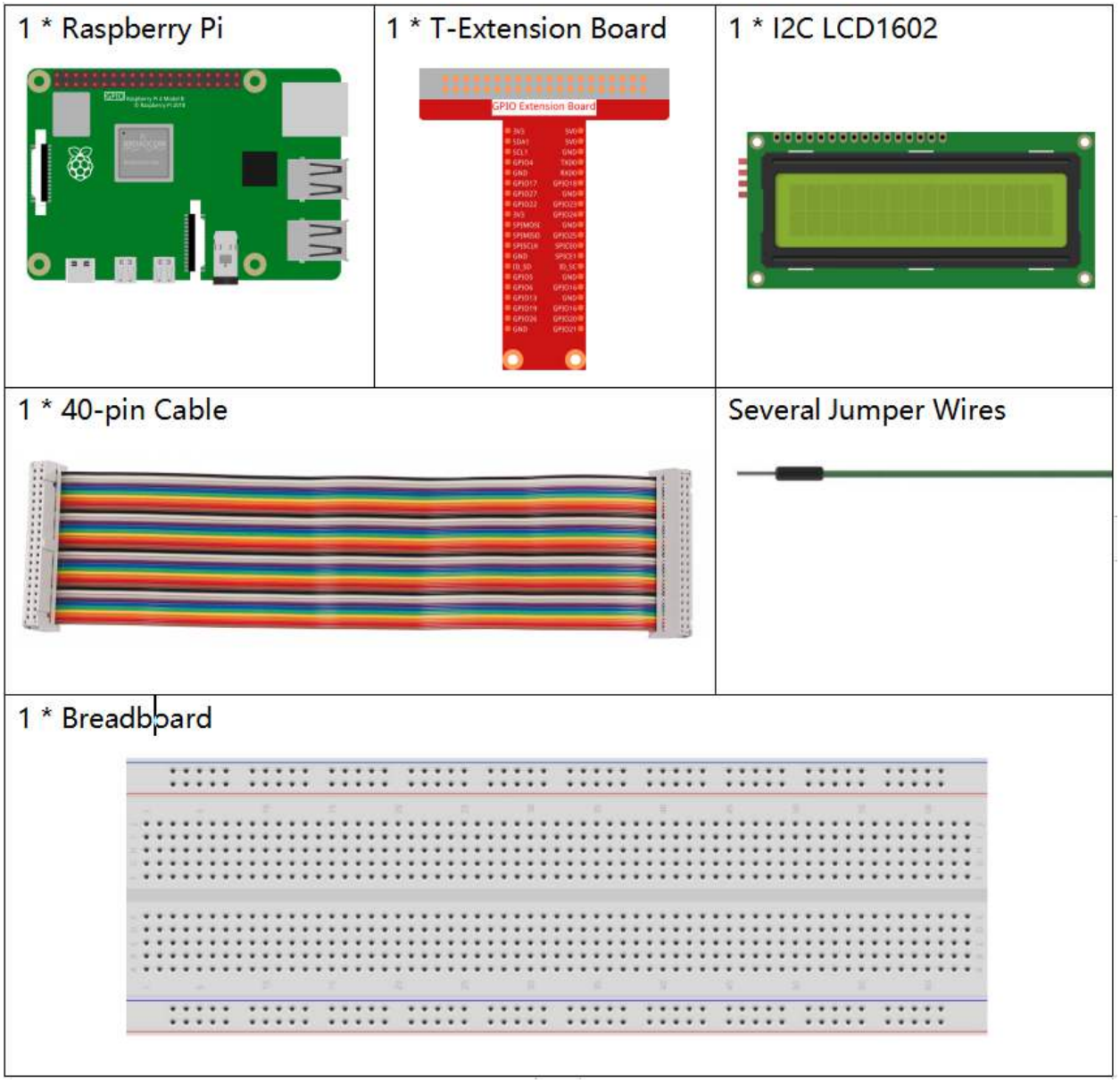


1.1.7 I2C LCD1602

Introduction

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

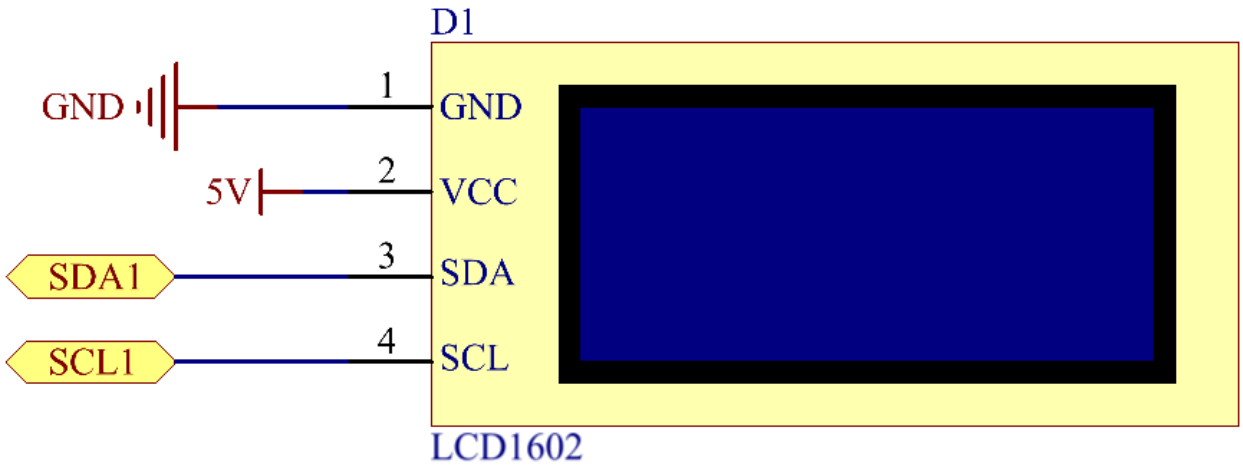
Components



- *GPIO Extension Board*
- *Breadboard*
- *I2C LCD1602*

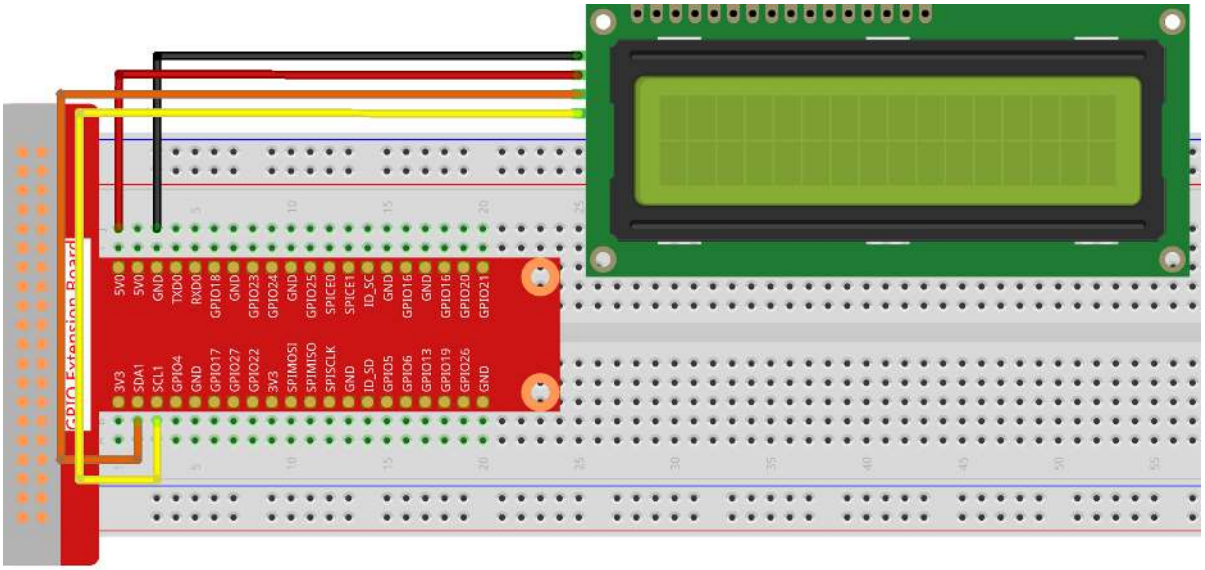
Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Change directory.

```
cd /home/pi/raphael-kit/c/1.1.7/
```

Step 4: Compile.

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

Step 5: Run.

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

After the code runs, you can see `Greetings!, From SunFounder` displaying on the LCD.

Note:

- If there is an error prompt `wiringPi.h: No such file or directory`, please refer to *C code is not working?*.
- If you get `Unable to open I2C device: No such file or directory` error, you need to refer to *I2C Configuration* to enable I2C and check if the wiring is correct.

Code

- 1.1.7_Lcd1602.c

Code Explanation

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

These functions are used to control I2C LCD1602 open source code. They allow us to easily use I2C LCD1602. Among these functions, `init()` is used for initialization, `clear()` is used to clear the screen, `write()` is used to write what is displayed, and other functions support the above functions.

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

This function initializes the I2C system with the specified device symbol. The prototype of the function:

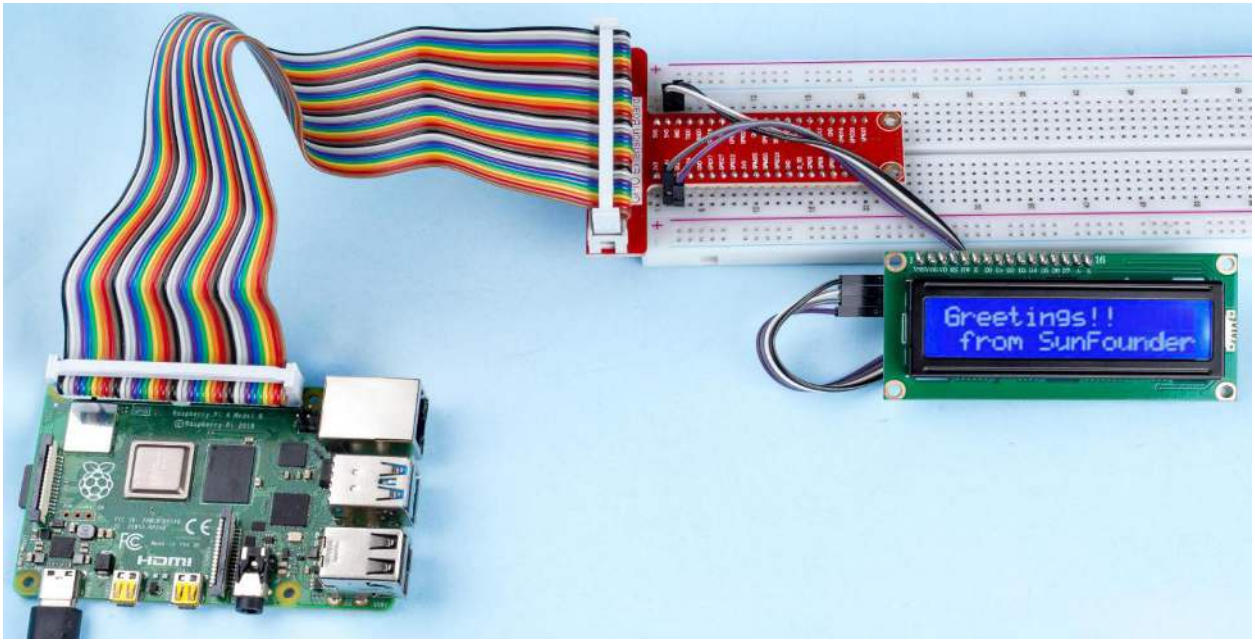
```
int wiringPiI2CSetup(int devId);
```

Parameters devId is the address of the I2C device, it can be found through the i2cdetect command(see Appendix) and the devId of I2C LCD1602 is generally 0x27.

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

In this function, `data[]` is the character to be printed on the LCD, and the parameters x and y determine the printing position (line y+1, column x+1 is the starting position of the character to be printed).

Phenomenon Picture



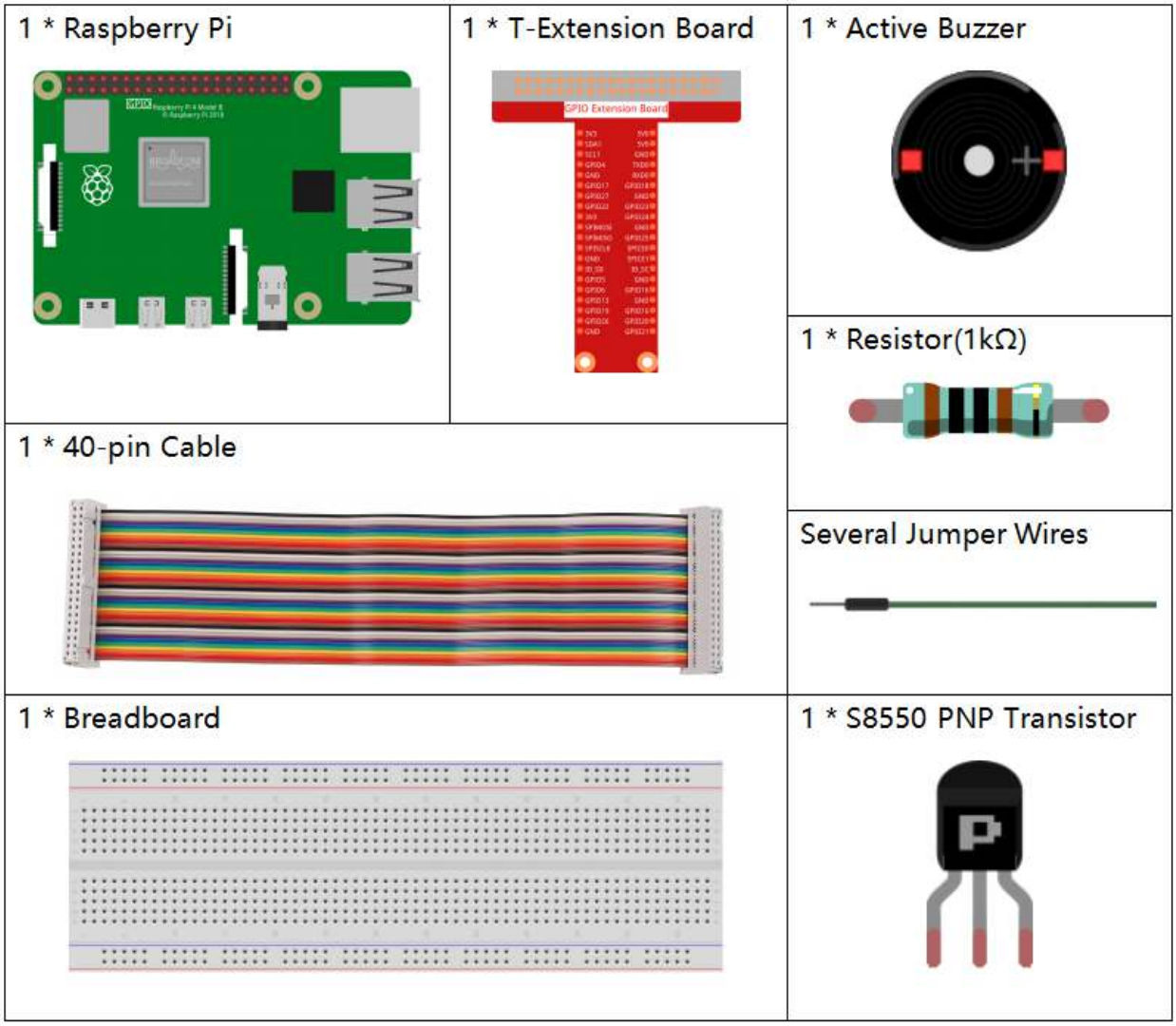
7.2.2 1.2 Sound

1.2.1 Active Buzzer

Introduction

In this project, we will learn how to drive an active buzzer to beep with a PNP transistor.

Components

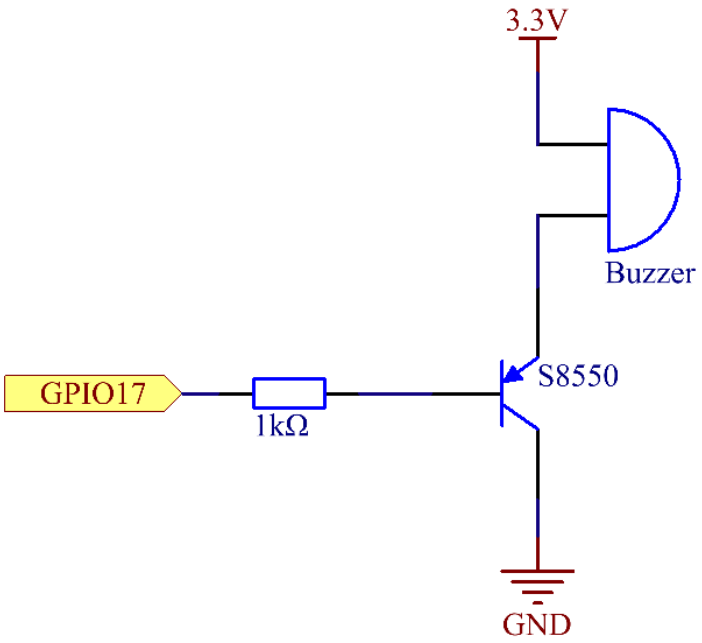


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Schematic Diagram

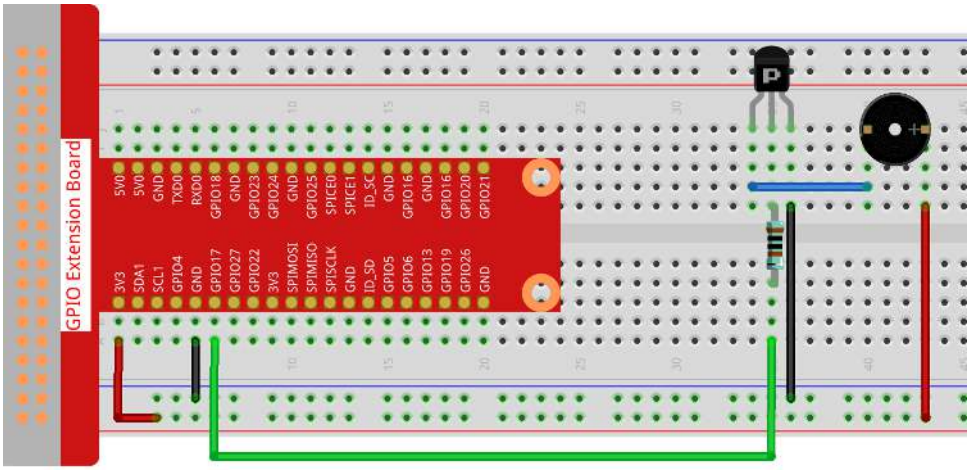
In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and the buzzer will make sounds. But when high level is supplied to the IO of Raspberry Pi, the transistor will be cut off and the buzzer will not make sounds.

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



Experimental Procedures

Step 1: Build the circuit. (The active buzzer has a white table sticker on the surface and a black back.)



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/1.2.1/
```

Step 3: Compile the code.

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

Step 4: Run the executable file above.

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

The code run, the buzzer beeps.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
```

(continues on next page)

(continued from previous page)

```
    }
    return 0;
}
```

Code Explanation

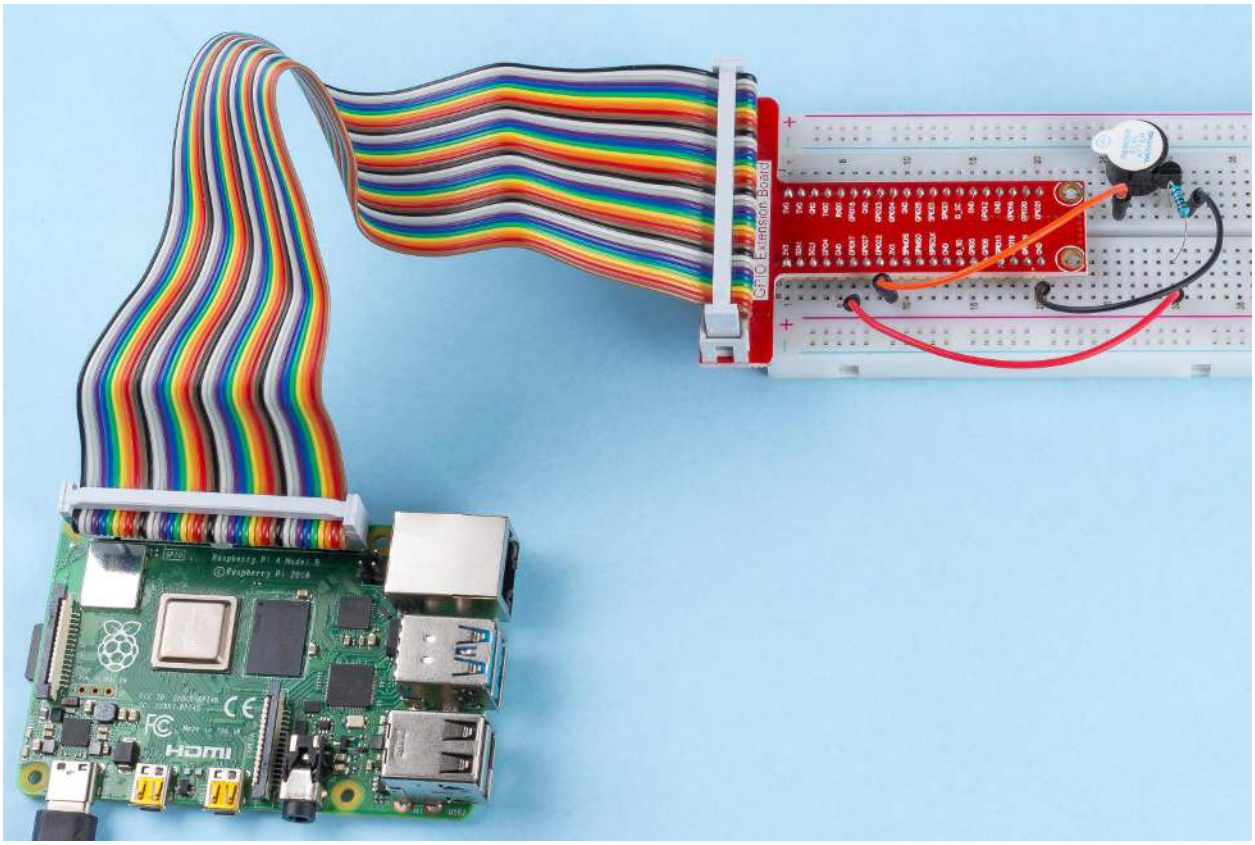
```
digitalWrite(BeepPin, LOW);
```

We use an active buzzer in this experiment, so it will make sound automatically when connecting to the direct current. This sketch is to set the I/O port as low level (0V), thus to manage the transistor and make the buzzer beep.

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

To set the I/O port as high level(3.3V), thus the transistor is not energized and the buzzer doesn't beep.

Phenomenon Picture

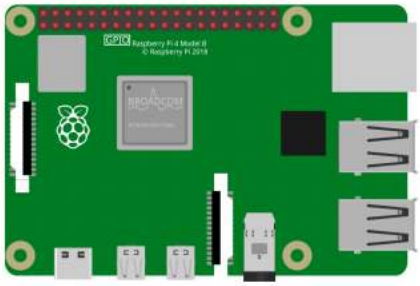
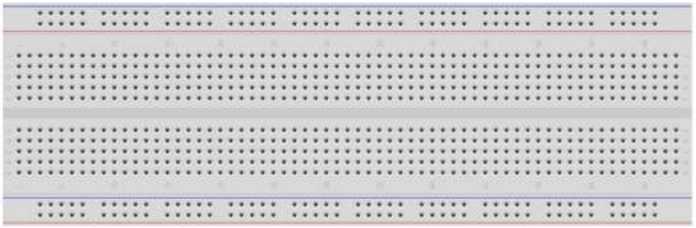
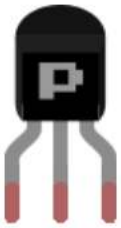


1.2.2 Passive Buzzer

Introduction

In this project, we will learn how to make a passive buzzer play music.

Components

<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>

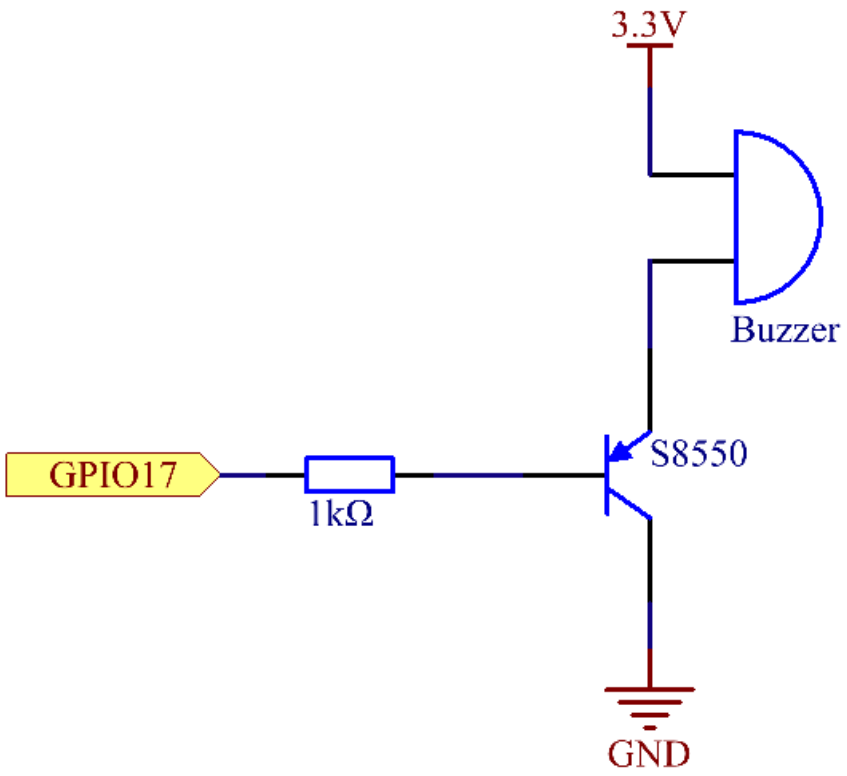
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Schematic Diagram

In this experiment, a passive buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor.

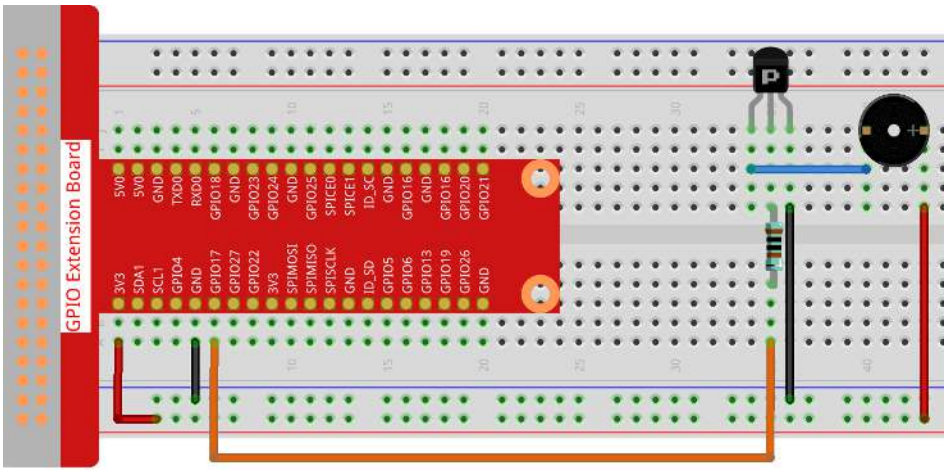
When GPIO17 is given different frequencies, the passive buzzer will emit different sounds; in this way, the buzzer plays music.

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



Experimental Procedures

Step 1: Build the circuit. (The Passive buzzer with green circuit board on the back.)



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/1.2.2/
```

Step 3: Compile.

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

Step 4: Run.

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

The code run, the buzzer plays a piece of music.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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
#define  CL6  221
#define  CL7  248

#define  CM1  262
#define  CM2  294
```

(continues on next page)

(continued from previous page)

```
#define  CM3  330
#define  CM4  350
#define  CM5  393
#define  CM6  441
#define  CM7  495

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

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

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


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

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

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

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

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

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

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

    return 0;
}
```

Code Explanation

```
#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
```

These frequencies of each note are as shown. CL refers to low note, CM middle note, CH high note, 1-7 correspond to the notes 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[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,2,1,1,
                1,1,1,1,1,1,3};
```

The array, `song_1[]` stores a musical score of a song in which `beat_1[]` refers to the beat of each note in the song (0.5s for each beat).

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

This creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the `wiringPiSetup()` function you used. The return value is 0 for success. Anything else and you should check the global errnovariable to see what went wrong.

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

Employ a for statement to play song_1.

In the judgment condition, **i<sizeof(song_1)/4**"devide by 4" is used because the array `song_1[]` is an array of the data type of integer, and each element takes up four bytes.

The number of elements in `song_1` (the number of musical notes) is gotten by deviding `sizeof(song_1)` by 4.

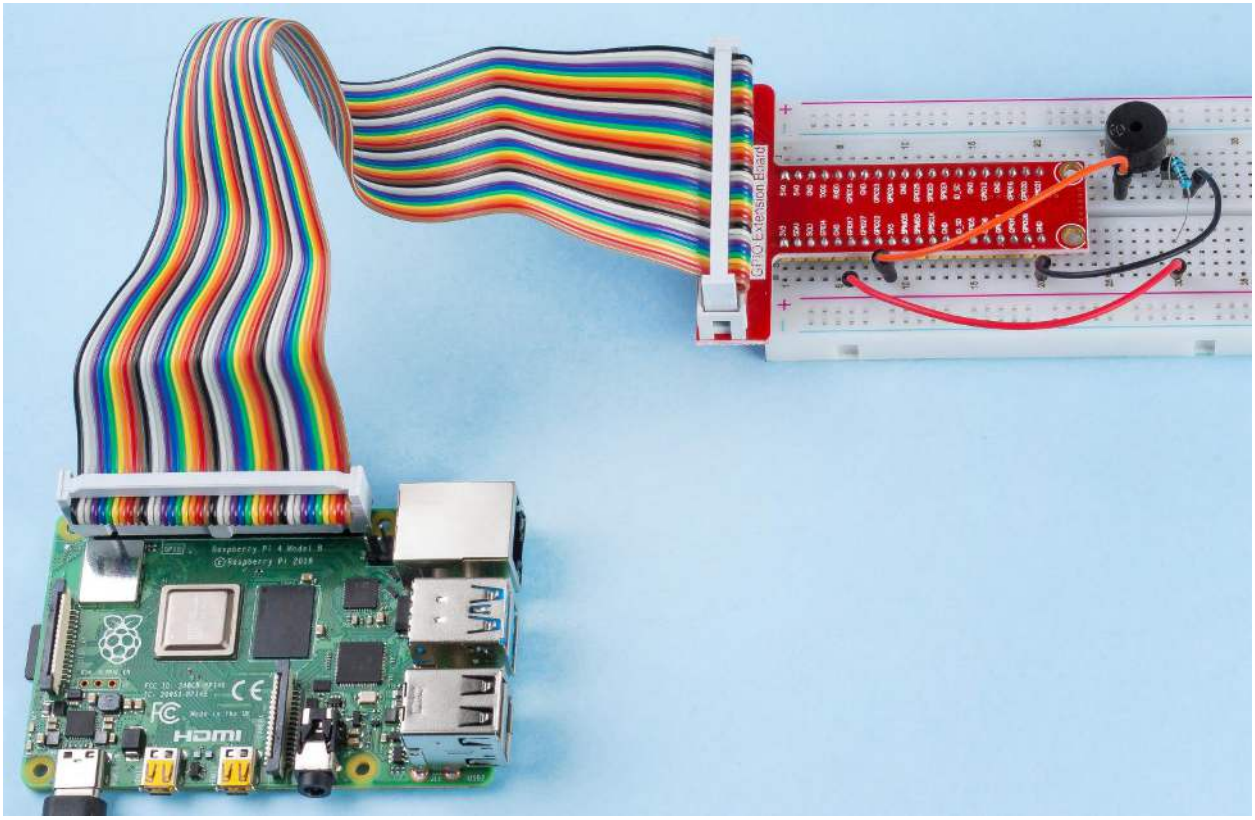
To enable each note to play for beat * 500ms, the function `delay(beat_1[i] \* 500)` is called.

The prototype of `softToneWrite(BuzPin, song_1[i])` is

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

This updates the tone frequency value on the given pin. The tone does not stop playing until you set the frequency to 0.

Phenomenon Picture



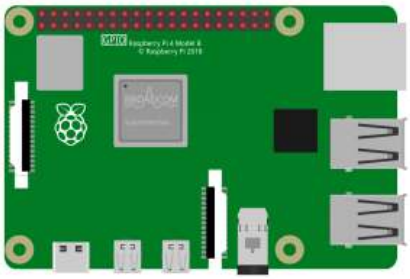
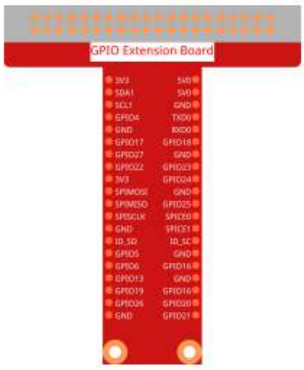
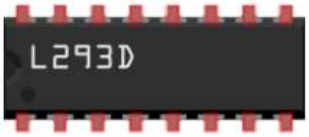
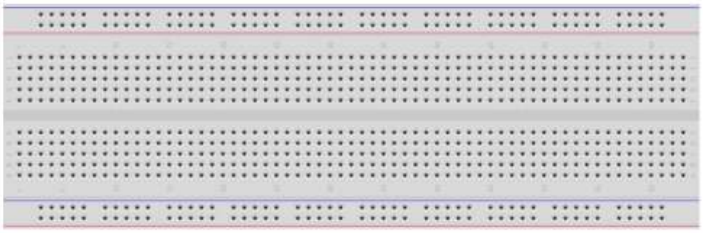
7.2.3 1.3 Drivers

1.3.1 Motor

Introduction

In this project, we will learn to how to use L293D to drive a DC motor and make it rotate clockwise and counterclockwise. Since the DC Motor needs a larger current, for safety purpose, here we use the Power Supply Module to supply motors.

Components

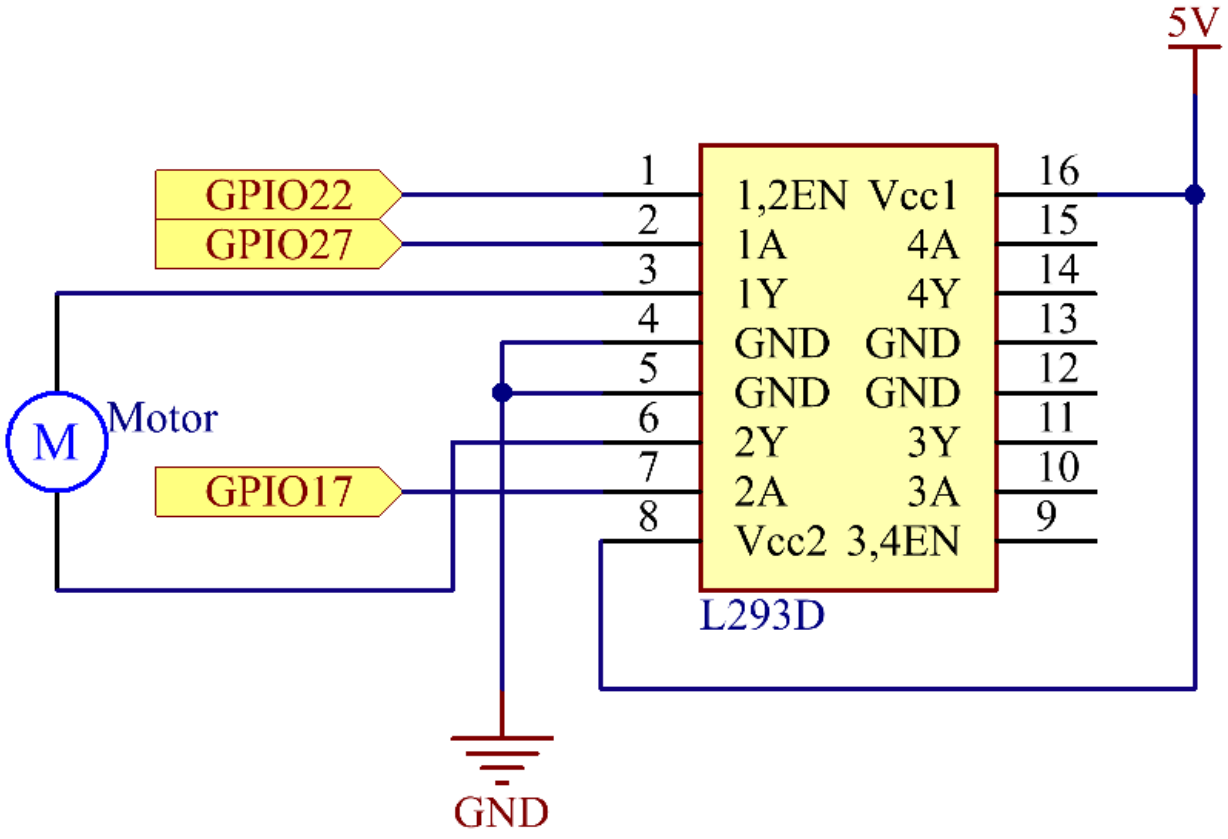
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Power Module (with 9V battery and buckle)
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * L293D
  </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * DC Motor
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Power Supply Module*
- *L293D*
- *DC Motor*

Schematic Diagram

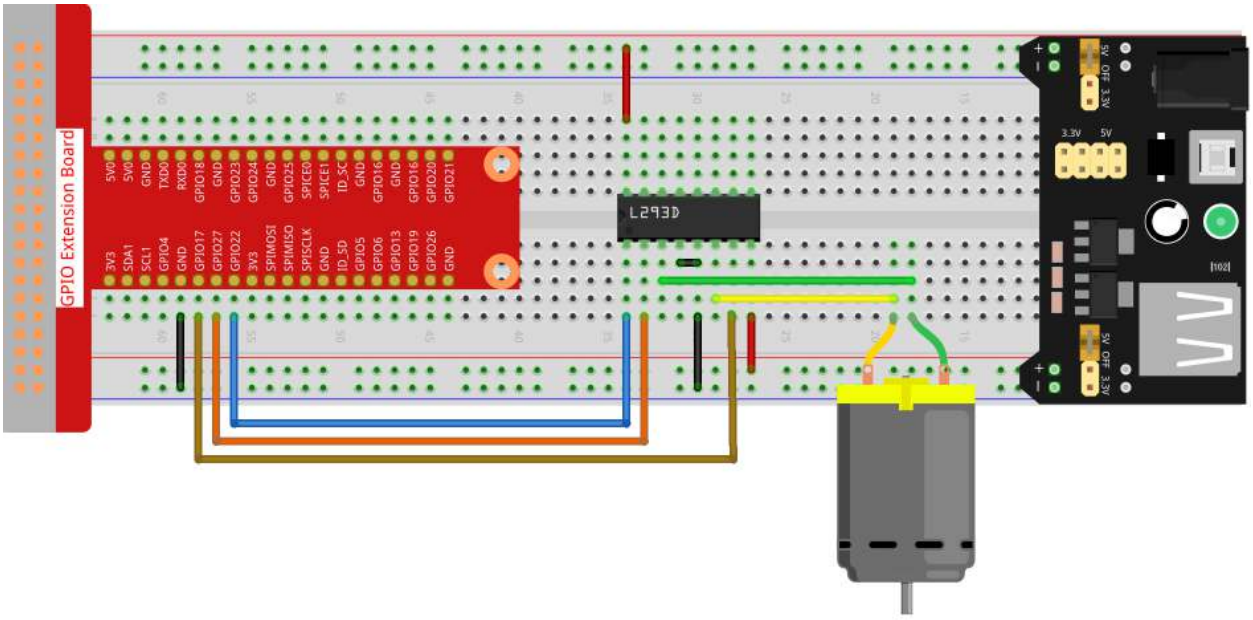
Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to GPIO22, and set it as high level. Connect pin2 to GPIO27, and pin7 to GPIO17, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

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



Experimental Procedures

Step 1: Build the circuit.



Note: 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.

Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/c/1.3.1/
```

Step 3: Compile.

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

Step 4: Run the executable file above.

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

As the code runs, the motor first rotates clockwise for 5s then stops for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please

refer to *C code is not working?*.

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");
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, HIGH);
        digitalWrite(MotorPin2, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

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

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

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

Code Explanation

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

Enable the L239D.

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

Set a high level for 2A(pin 7); since 1,2EN(pin 1) is in high level, 2Y will output high level.

Set a low level for 1A, then 1Y will output low level, and the motor will rotate.

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

this loop is to delay for 3*1000ms.

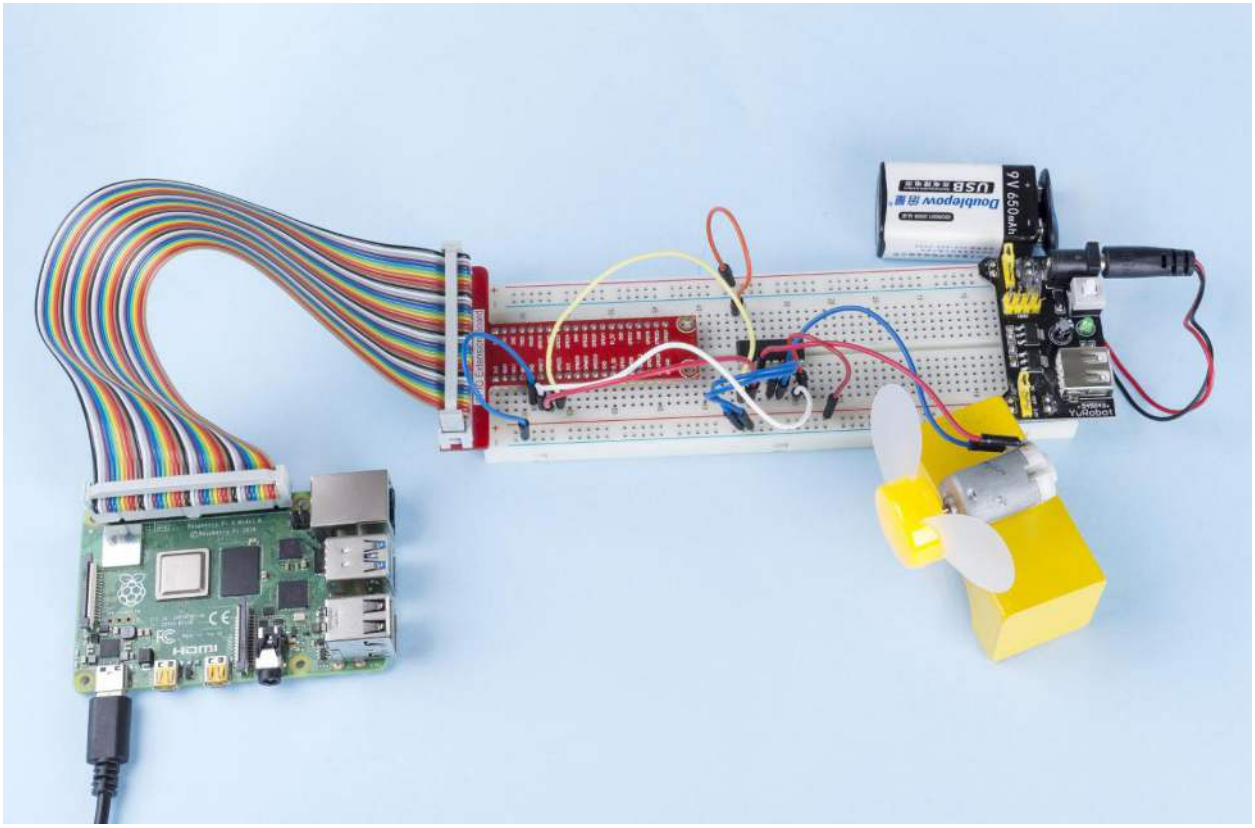
```
digitalWrite(MotorEnable, LOW)
```

If 1,2EN (pin1) is in low level, L293D does not work. Motor stops rotating.

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

Reverse the current flow of the motor, then the motor will rotate reversely.

Phenomenon Picture

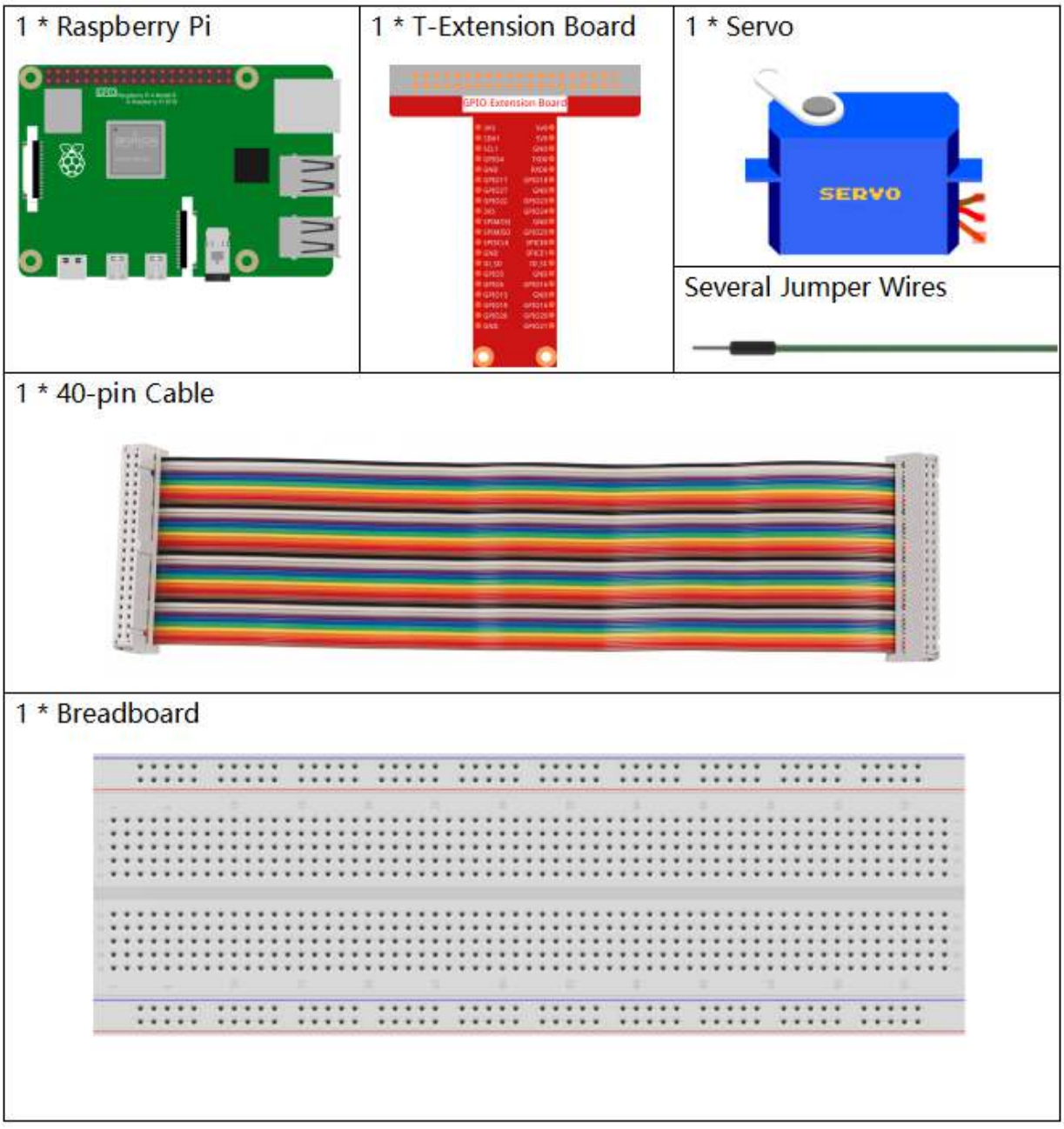


1.3.2 Servo

Introduction

In this project, we will learn how to make the servo rotate.

Components

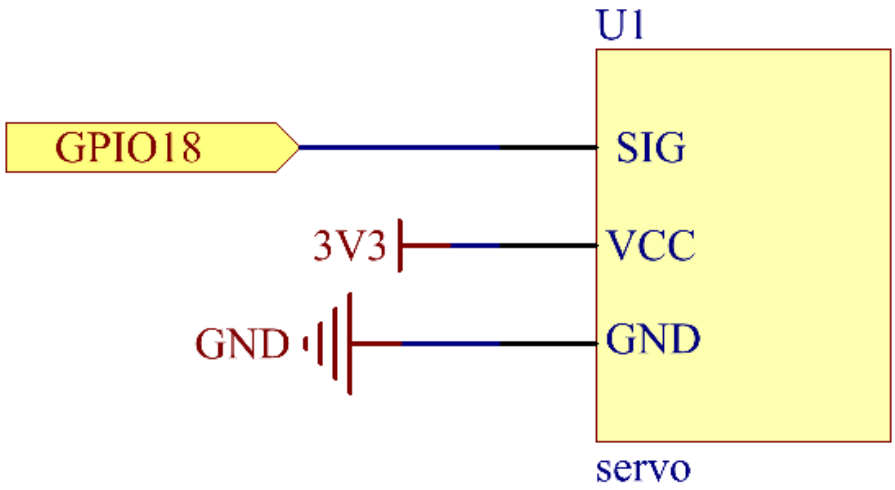


- *GPIO Extension Board*
- *Breadboard*

- *Servo*

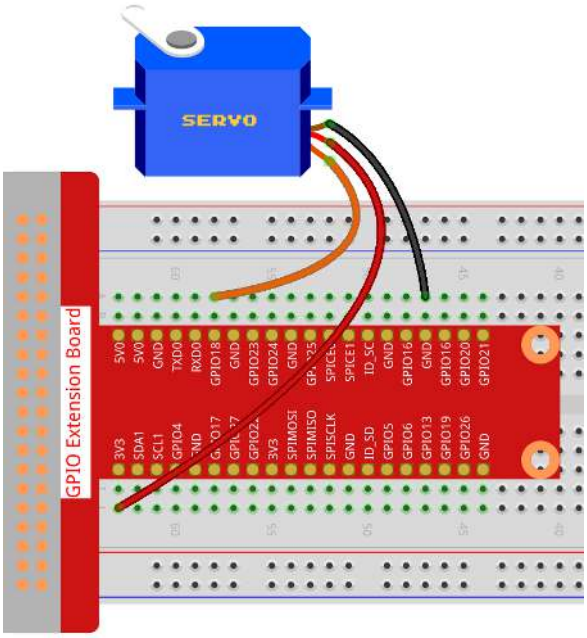
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/1.3.2
```

Step 3: Compile the code.

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

Step 4: Run the executable file.

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

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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;
```

(continues on next page)

(continued from previous page)

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

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

Code Explanation

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

Create a `Map()` function to map value in the following code.

```
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));
}
```

Create a funtion, `setAngle()` to write angle to the servo.

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

This function can change the duty cycle of the PWM.

To make the servo rotate to 0 ~ 180 °, the pulse width should change within the range of 0.5ms ~ 2.5ms when the period is 20ms; in the function, `softPwmCreate()` , we have set that the period is 200x100us=20ms, thus we need to map 0 ~ 180 to 5x100us ~ 25x100us.

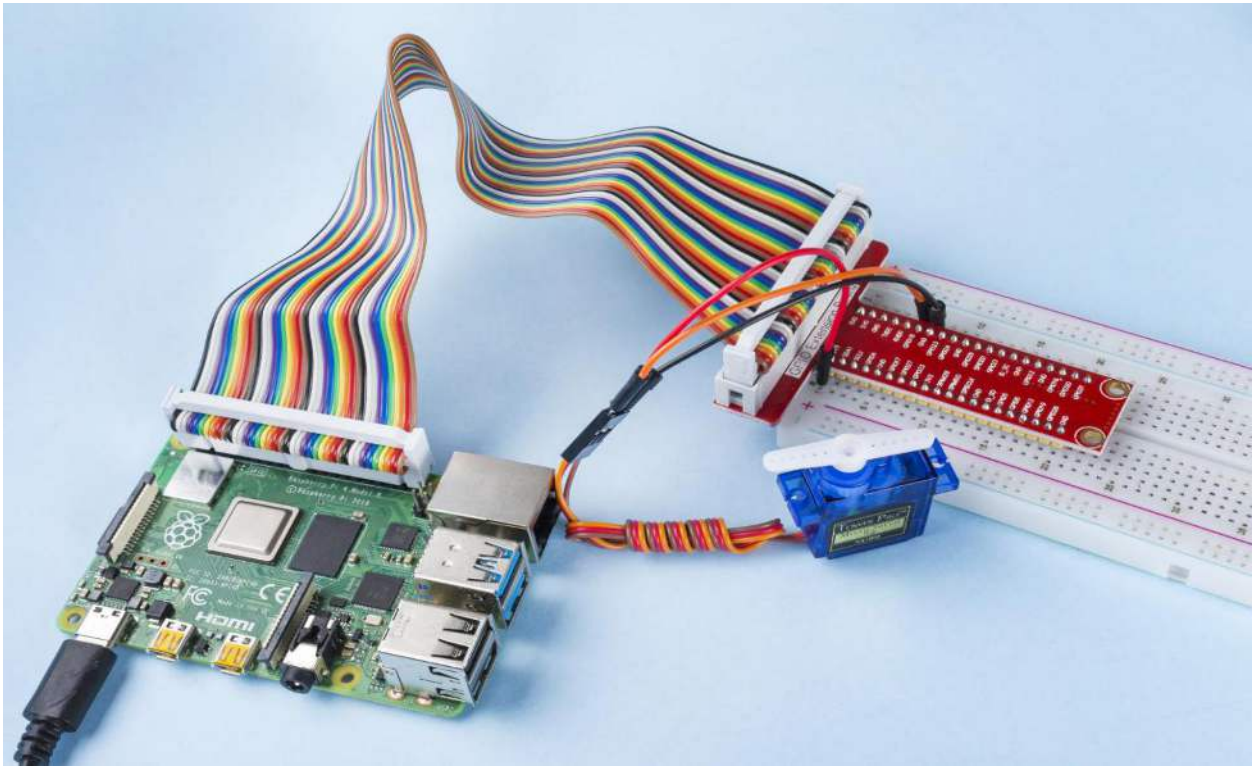
The prototype of this function is shown below.

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

- `pin`: Any GPIO pin of Raspberry Pi can be set as PWM pin.

- `initialValue`: The initial pulse width is that initialValue times 100us.
- `pwmRange`: the period of PWM is that pwmRange times 100us.

Phenomenon Picture

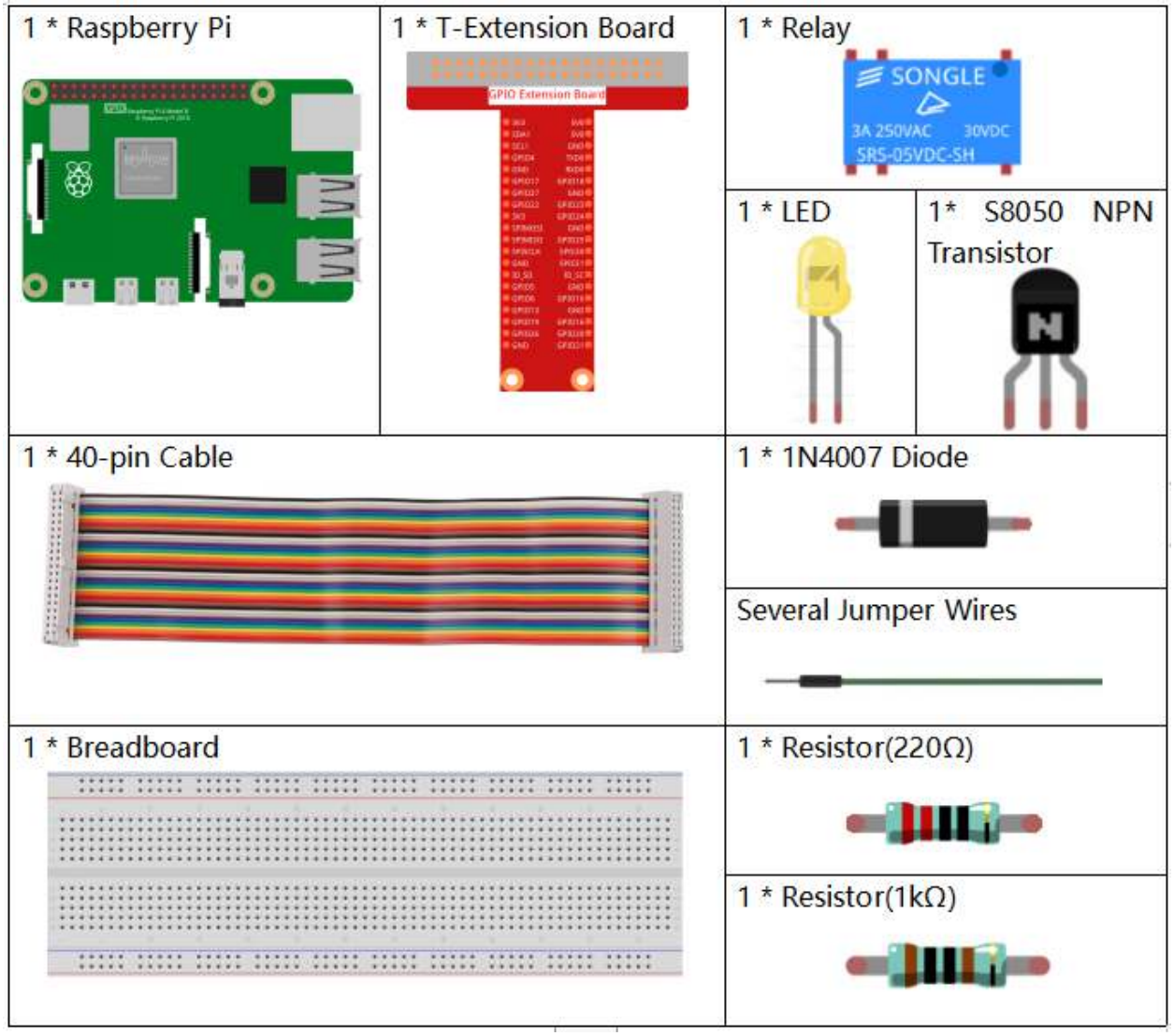


1.3.3 Relay

Introduction

In this project, we will learn to use a relay. It is one of the commonly used components in automatic control system. When the voltage, current, temperature, pressure, etc., reaches, exceeds or is lower than the predetermined value, the relay will connect or interrupt the circuit, to control and protect the equipment.

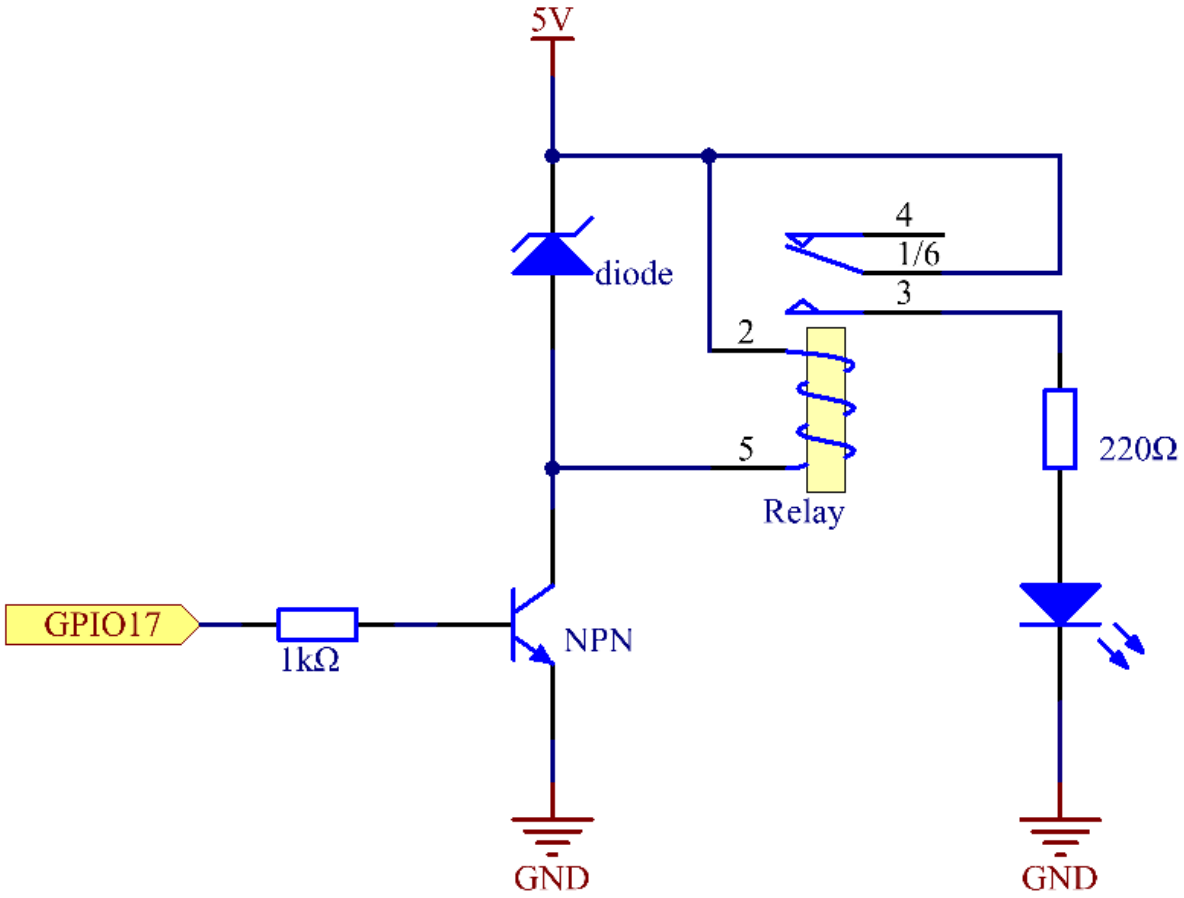
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Transistor*
- *Relay*
- *Diode*

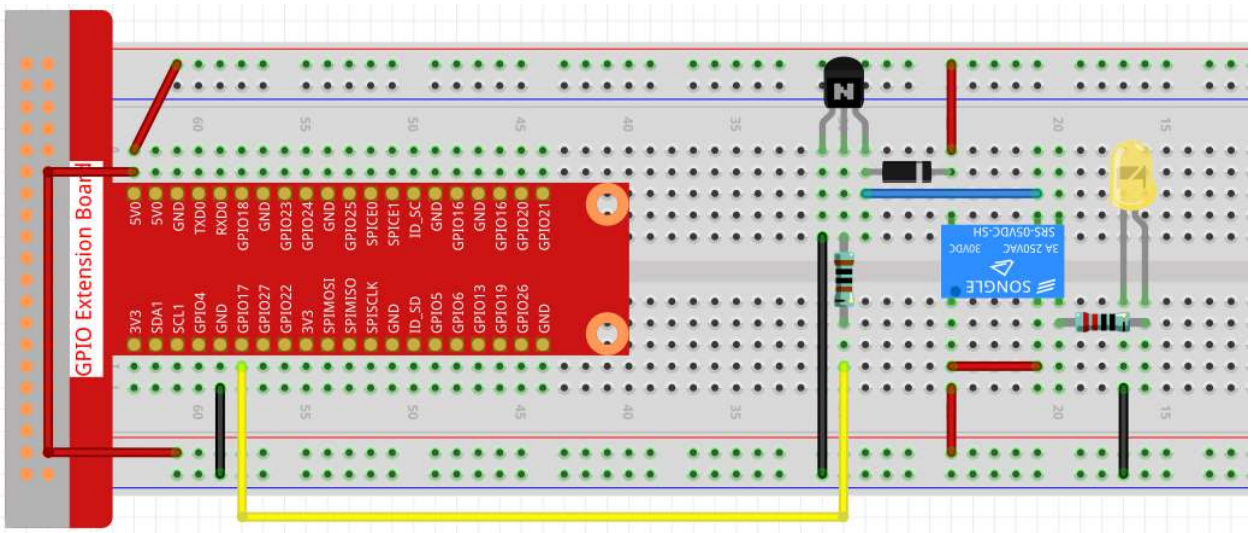
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/1.3.3
```

Step 3: Compile the code.

```
gcc 1.3.3_Relay.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, the LED will light up. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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");
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
```

(continues on next page)

(continued from previous page)

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

    return 0;
}
```

Code Explanation

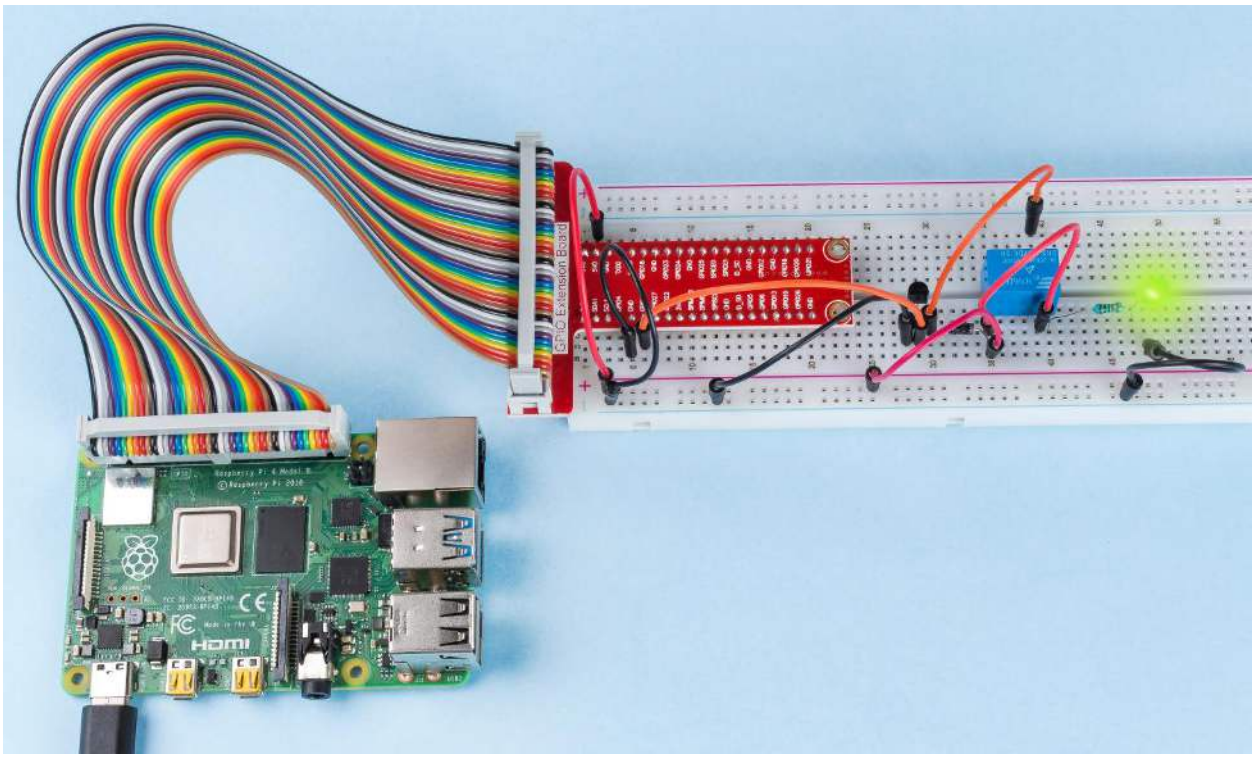
```
digitalWrite(RelayPin, LOW);
```

Set the I/O port as low level (0V), thus the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens, LED does not turn on.

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

set the I/O port as high level (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes, LED lights up.

Phenomenon Picture



7.3 Input

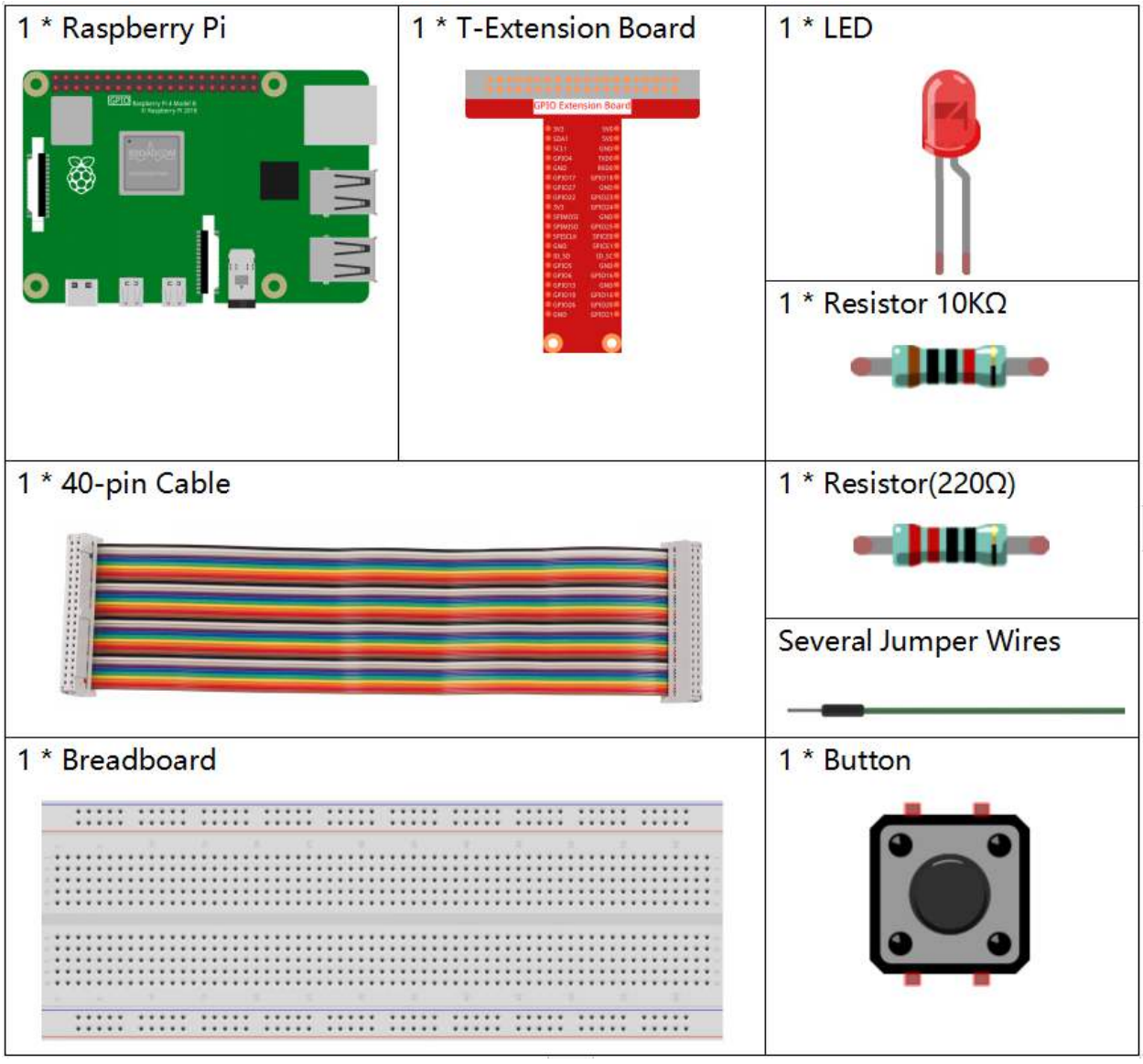
7.3.1 2.1 Controllers

2.1.1 Button

Introduction

In this project, we will learn how to turn on or off the LED by using a button.

Components



- *GPIO Extension Board*
- *Breadboard*

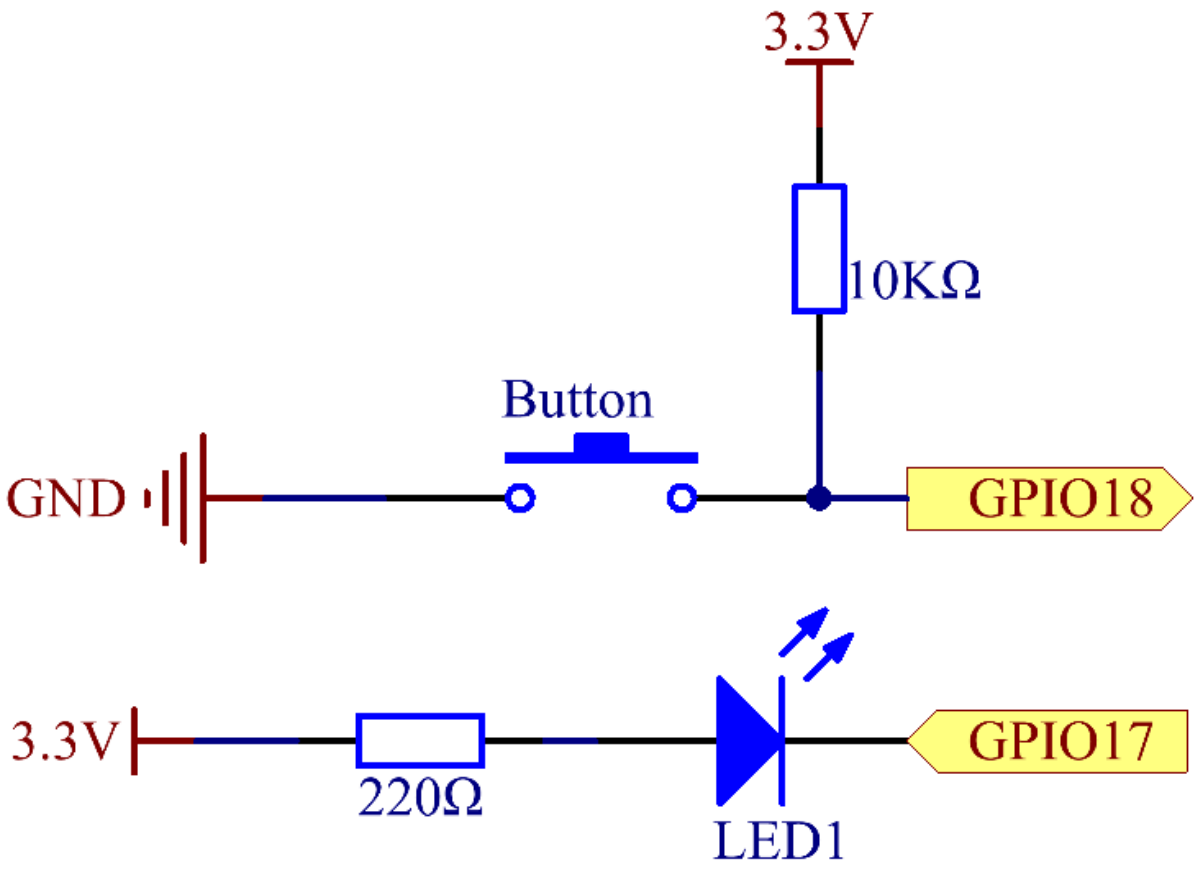
- *Resistor*
- *LED*
- *Button*

Schematic Diagram

Use a normally open button as the input of Raspberry Pi, the connection is shown in the schematic diagram below. When the button is pressed, the GPIO18 will turn into low level (0V). We can detect the state of the GPIO18 through programming. That is, if the GPIO18 turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

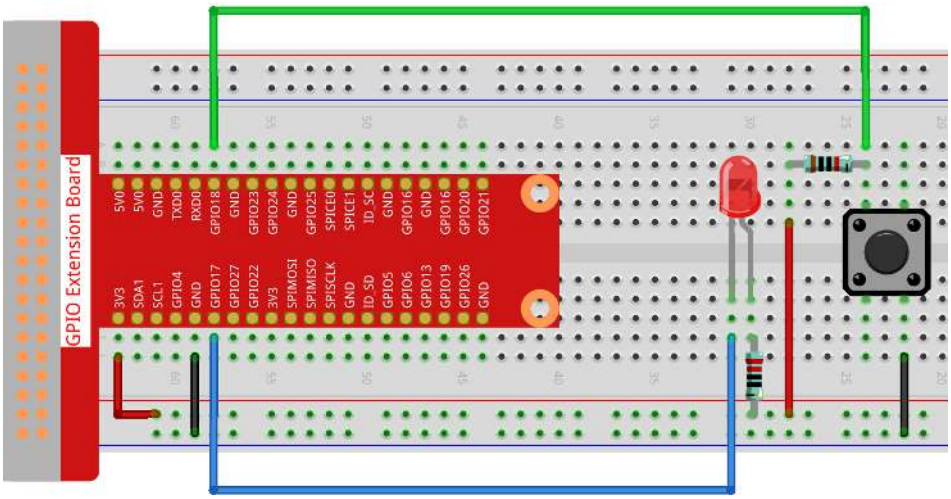
Note: The longer pin of the LED is the anode and the shorter one is the cathode.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/2.1.1/
```

Note: Change directory to the path of the code in this experiment via **cd**.

Step 3: Compile the code.

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

Step 4: Run the executable file.

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

After the code runs, press the button, the LED lights up; otherwise, turns off.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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;
```

(continues on next page)

(continued from previous page)

```
    }

    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 Explanation

```
#define LedPin      0
```

Pin GPIO17 in the T_Extension Board is equal to the GPIO0 in the wiringPi.

```
#define ButtonPin   1
```

ButtonPin is connected to GPIO1.

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

Set LedPin as output to assign value to it.

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

Set ButtonPin as input to read the value of ButtonPin.

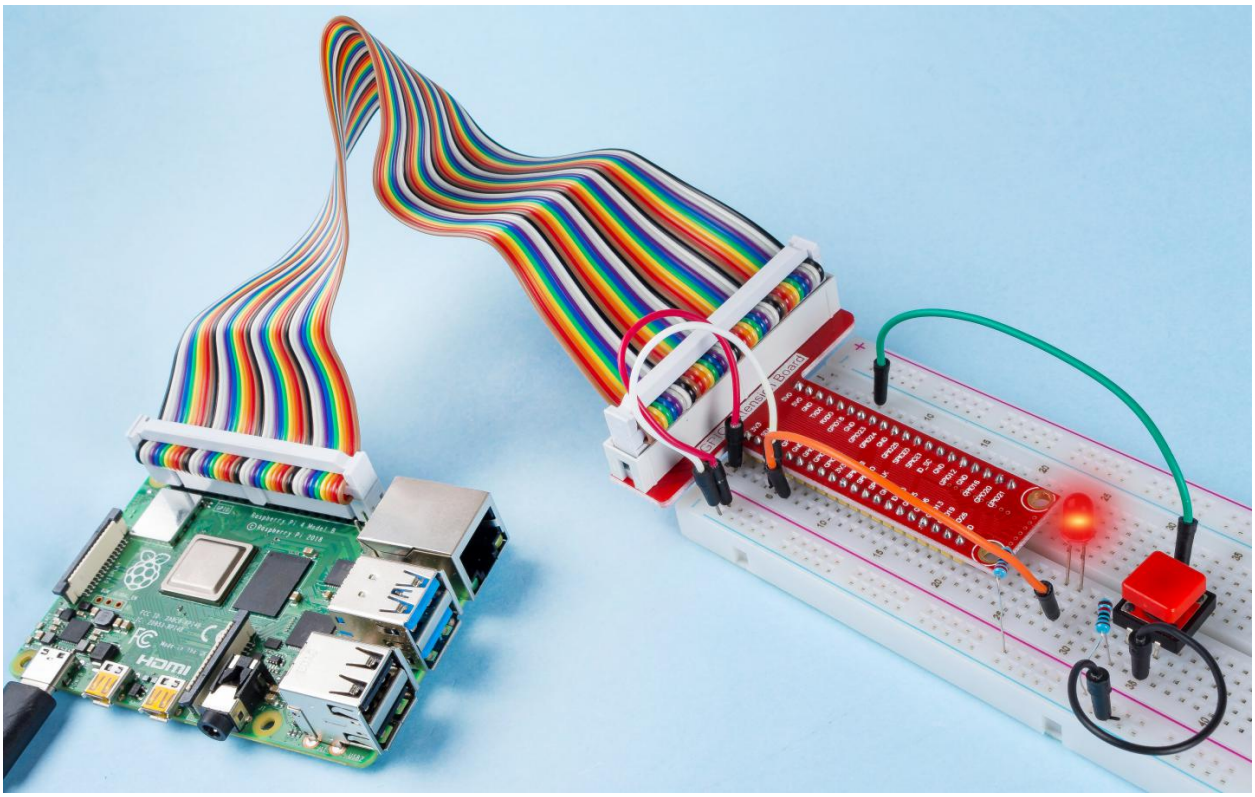
```
while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 0){
            // Led on
            digitalWrite(LedPin, LOW);
        //  printf("...LED on\n");
        }
        else{
            // Led off
            digitalWrite(LedPin, HIGH);
        //  printf("LED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

`if (digitalRead (ButtonPin) == 0) :` check whether the button has been pressed. Execute `digitalWrite(LedPin, LOW)` when button is pressed to light up LED.

`digitalRead()` function is to read HIGH (high level) or LOW (low level) of the input parameter word pin, it returns 1 when pin is HIGH and returns 0 when pin is LOW.

`digitalWrite()` function is to write HIGH (high level) or LOW (low level) to the input parameter word pin.

Phenomenon Picture



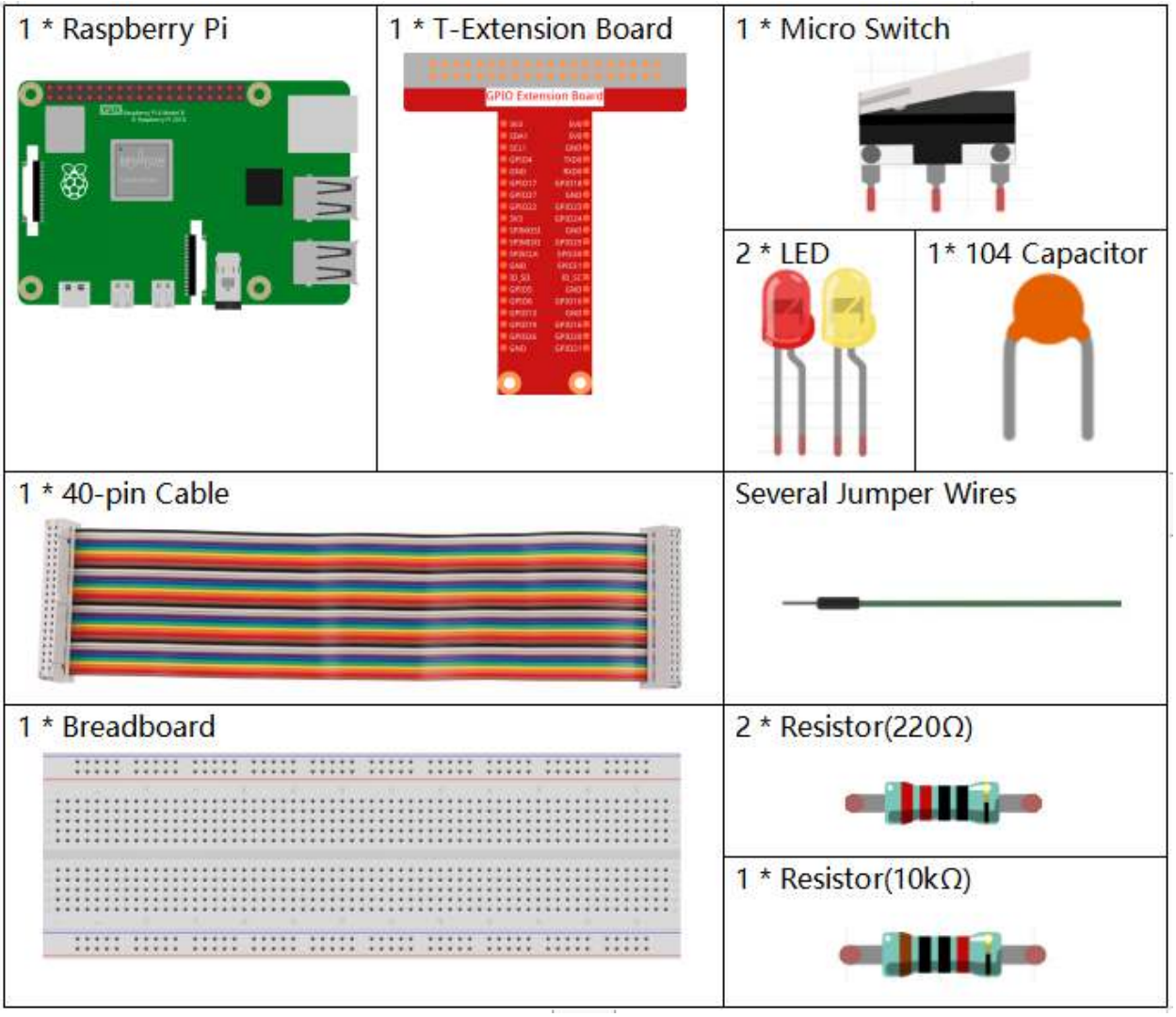
2.1.2 Micro Switch

Introduction

In this project, we will learn how to use Micro Switch. A Micro Switch is a small, very sensitive switch which requires minimum compression to activate. Because they are reliable and sensitive, micro switches are often used as a safety device.

They are used to prevent doors from closing if something or someone is in the way and other applications similar.

Components

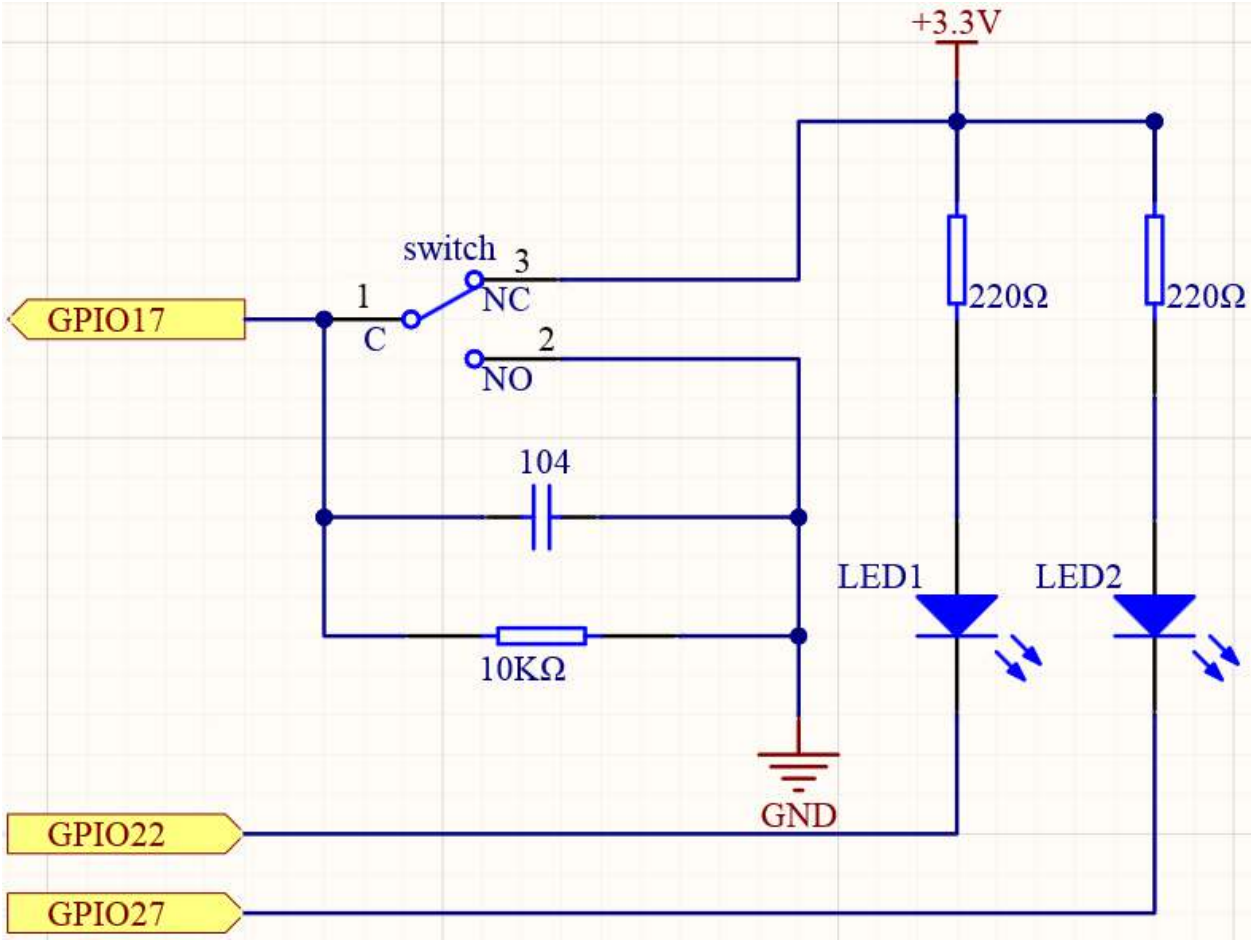


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Micro Switch*
- *Capacitor*

Schematic Diagram

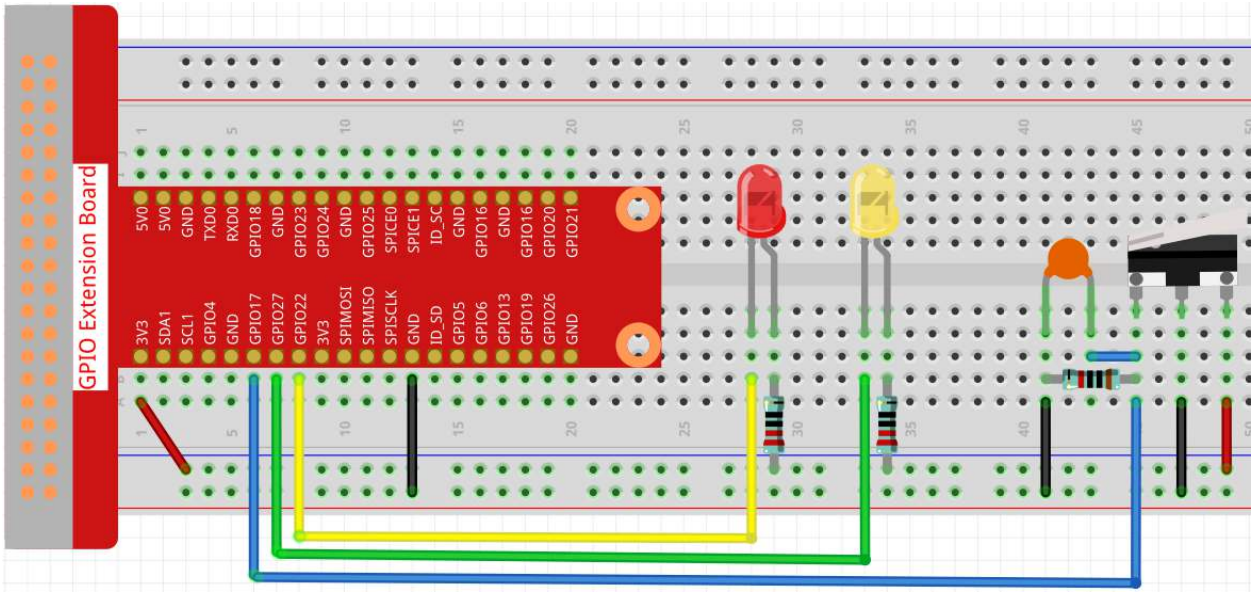
Connect the left pin of the Micro Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you press and release the move arm of the Micro Switch, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.1.2
```

Step 3: Compile.

```
gcc 2.1.2_MicroSwitch.c -lwiringPi
```

Step 4: Run the executable file above.

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

While the code is running, press the Micro Switch, then the yellow LED lights up; release the moving arm, the red LED turns on.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define microPin            0
#define led1                        3
#define led2                        2

int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
```

(continues on next page)

(continued from previous page)

```
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(microPin, INPUT);
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

    while(1){
        // micro switch high, led1 on
        if(digitalRead(microPin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("LED1 on\n");
        }
        // micro switch low, led2 on
        if(digitalRead(microPin) == 0){
            digitalWrite(led2, LOW);
            digitalWrite(led1, HIGH);
            printf(".....LED2 on\n");
        }
        delay(500);
    }

    return 0;
}
```

Code Explanation

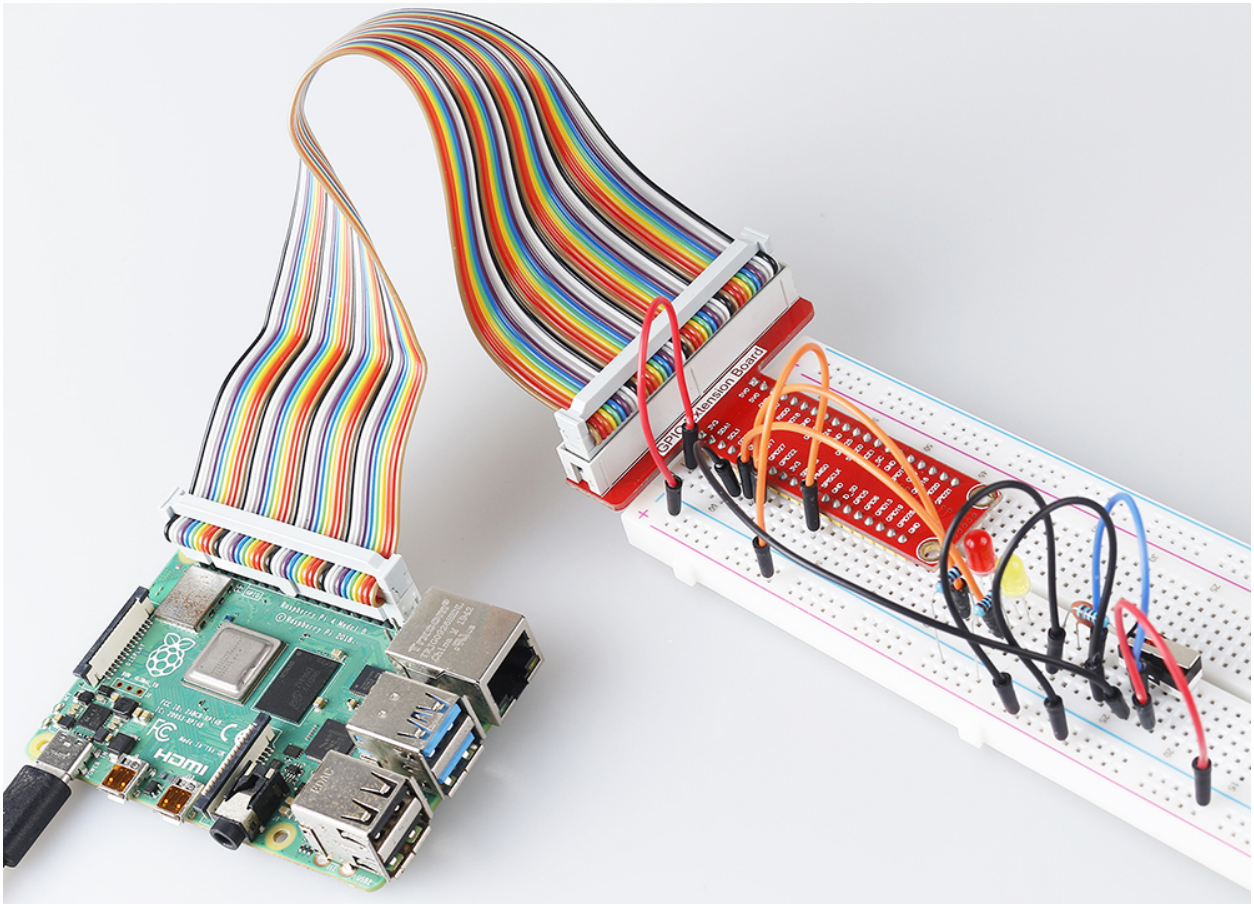
```
if(digitalRead(slidePin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("LED1 on\n");
    }
```

When the moving arm of the micro switch is released, the left pin is connected to the right pin; at this time, a high level will be read on GPIO17, and then LED1 will be on and LED2 will be off.

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

When the move arm is pressed, the left pin and the middle pin are connected. At this point a low level will be read on GPIO17, then turns LED2 on and LED1 off.

Phenomenon Picture

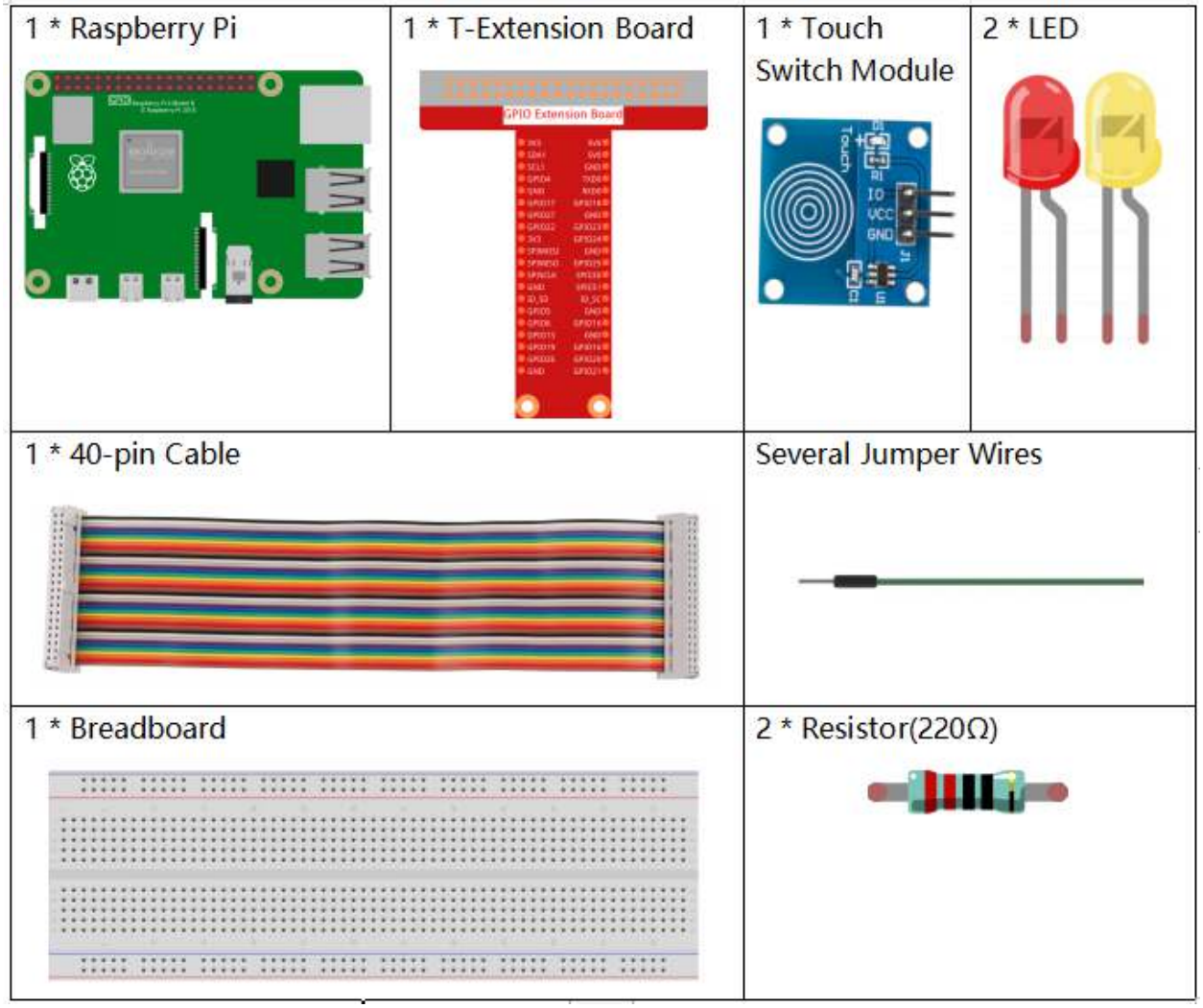


2.1.3 Touch Switch Module

Introduction

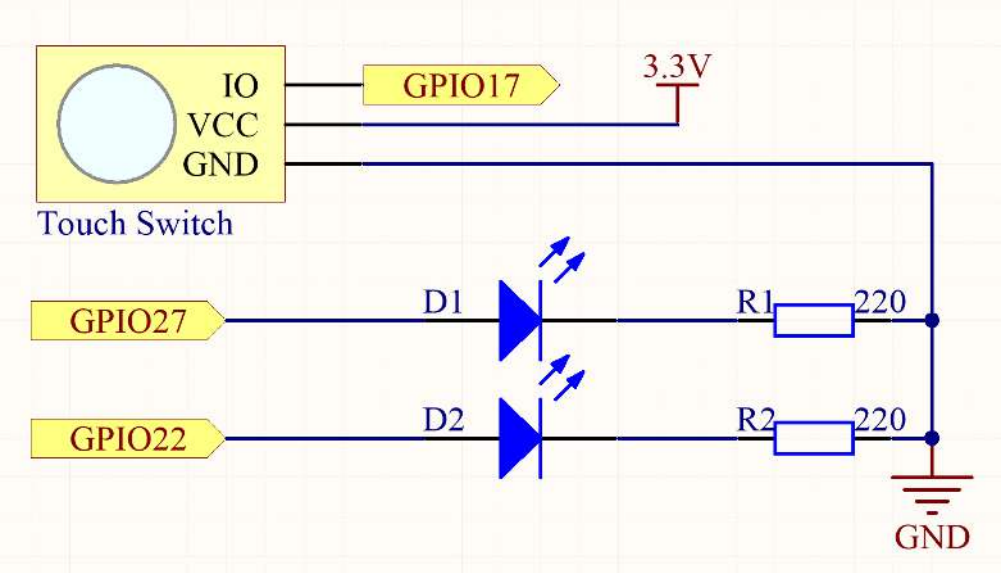
In this project, you will learn about touch switch module. It can replace the traditional kinds of switch with these advantages: convenient operation, fine touch sense, precise control and least mechanical wear.

Components



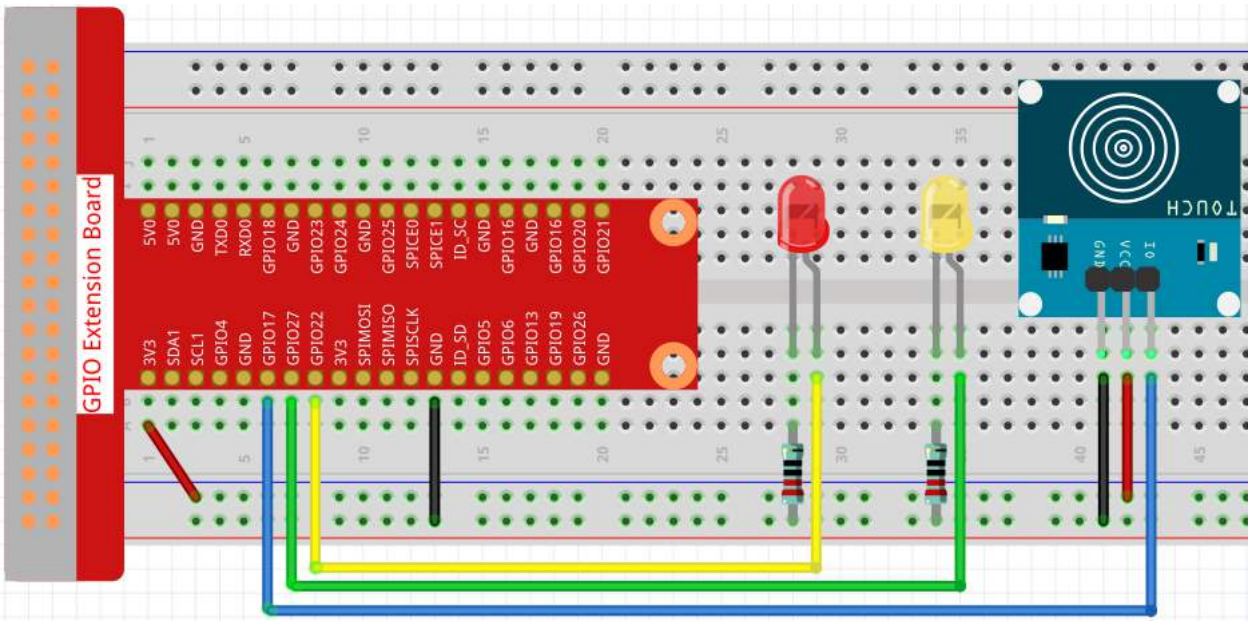
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Touch Switch Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/2.1.3/
```

Step 3: Compile.

```
gcc 2.1.3_TouchSwitch.c -lwiringPi
```

Step 4: Run.

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

While the code is running, the red LED lights up; when you tap on the touch switch module, the yellow LED turns on.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define touchPin                0
#define led1                    3
#define led2                    2

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

    pinMode(touchPin, INPUT);
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

    while(1){
        // touch switch high, led1 on
        if(digitalRead(touchPin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("You touch it! \r\n");
        }
        // touch switch low, led2 on
        if(digitalRead(touchPin) == 0){
            digitalWrite(led2, LOW);
            digitalWrite(led1, HIGH);
        }
    }

    return 0;
}
```

Code Explanation

```
#define touchPin                0
#define led1                    3
#define led2                    2
```

Pin GPIO17, GPIO22 and GPIO27 of the T_Extension Board is corresponding to the GPIO0, GPIO3 and GPIO2 in wiringPi. Assign GPIO0, GPIO3 and GPIO2 to `touchPin`, `led1` and `led2`.

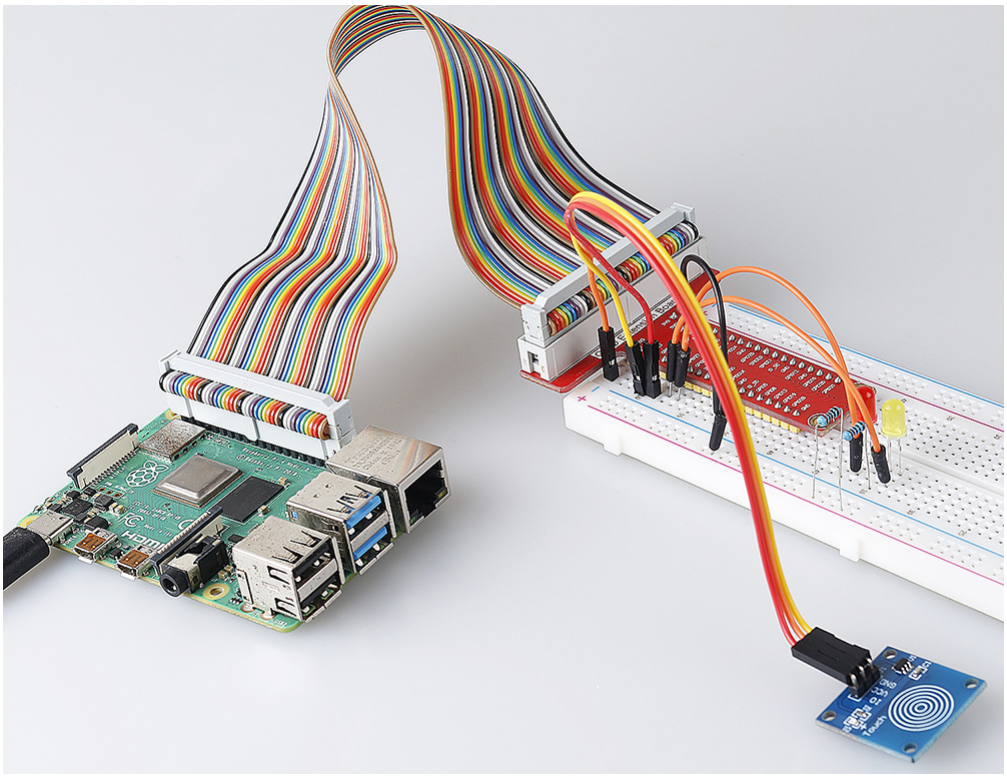
```
pinMode(touchPin, INPUT);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
```

Set `led1`, `led2` as output to write value to them and set `touchPin` as input to read value from it.

```
while(1){
    // touch switch high, led1 on
        if(digitalRead(touchPin) == 1){
        digitalWrite(led1, LOW);
        digitalWrite(led2, HIGH);
        printf("You touch it! \r\n");
    }
    // touch switch low, led2 on
    if(digitalRead(touchPin) == 0){
        digitalWrite(led2, LOW);
        digitalWrite(led1, HIGH);
    }
}
```

Set an infinite loop, when you tap on the touch switch module, `touchPin` is high, led1 will light up and print “You touch it!” . When touchPin is low, led2 will light up.

Phenomenon Picture

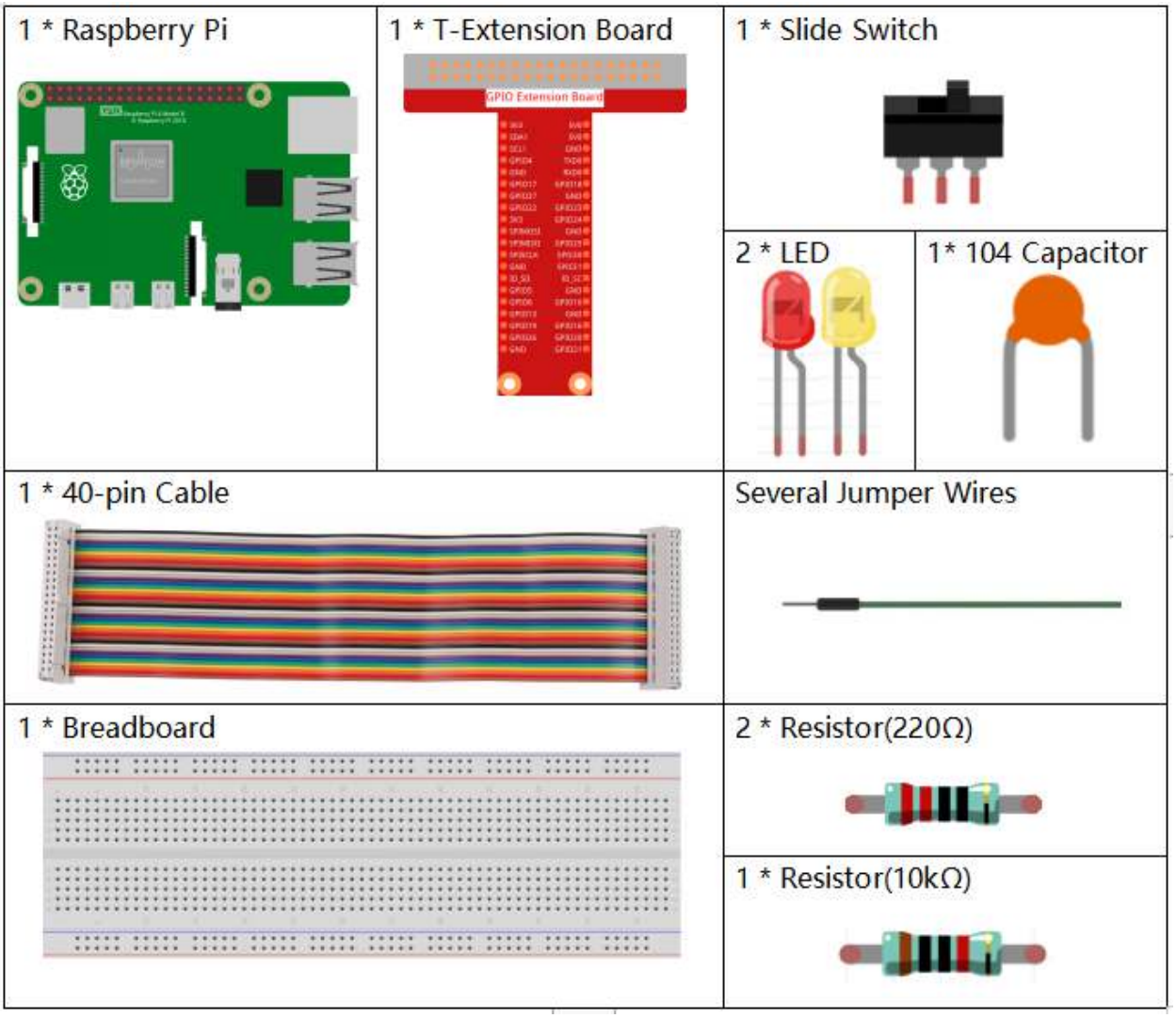


2.1.4 Slide Switch

Introduction

In this project, we will learn how to use a slide switch. Usually,the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.

Components



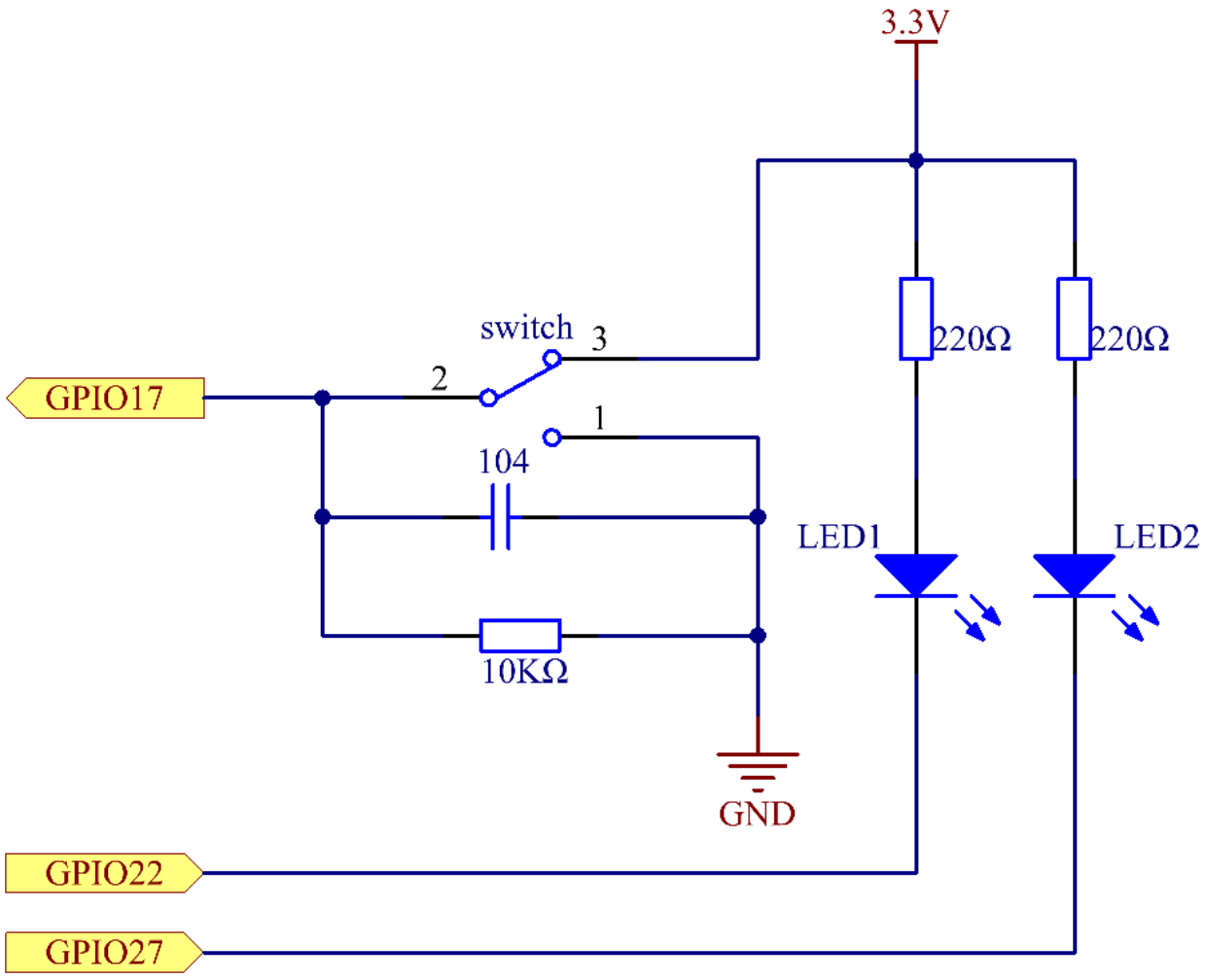
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Slide Switch*

- *Capacitor*

Schematic Diagram

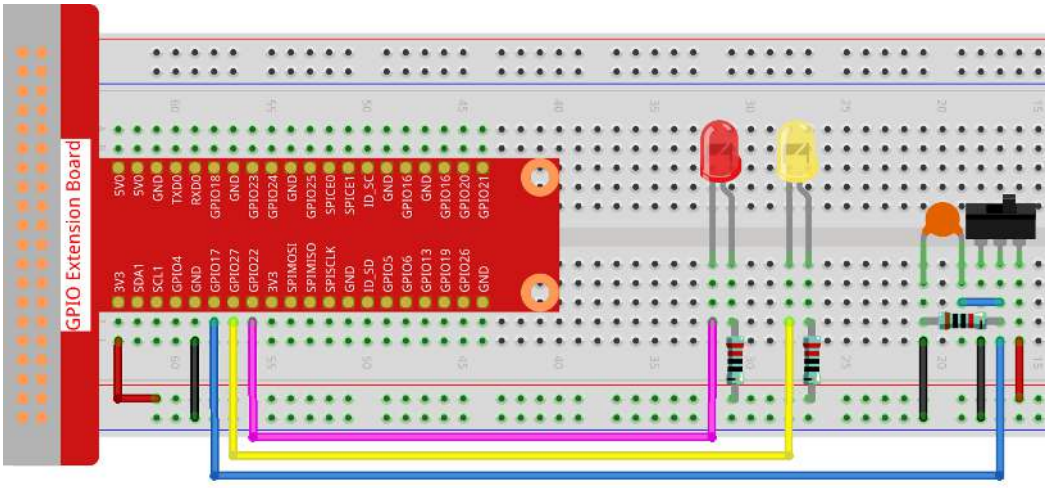
Connect the middle pin of the Slide Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.1.4
```

Step 3: Compile.

```
gcc 2.1.4_Slider.c -lwiringPi
```

Step 4: Run the executable file above.

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

While the code is running, get the switch connected to the left, then the yellow LED lights up; to the right, the red light turns on.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
```

(continues on next page)

(continued from previous page)

```
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
    while(1){
        // slide switch high, led1 on
        if(digitalRead(slidePin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("LED1 on\n");
        }
        // slide switch low, led2 on
        if(digitalRead(slidePin) == 0){
            digitalWrite(led2, LOW);
            digitalWrite(led1, HIGH);
            printf(".....LED2 on\n");
        }
    }
    return 0;
}
```

Code Explanation

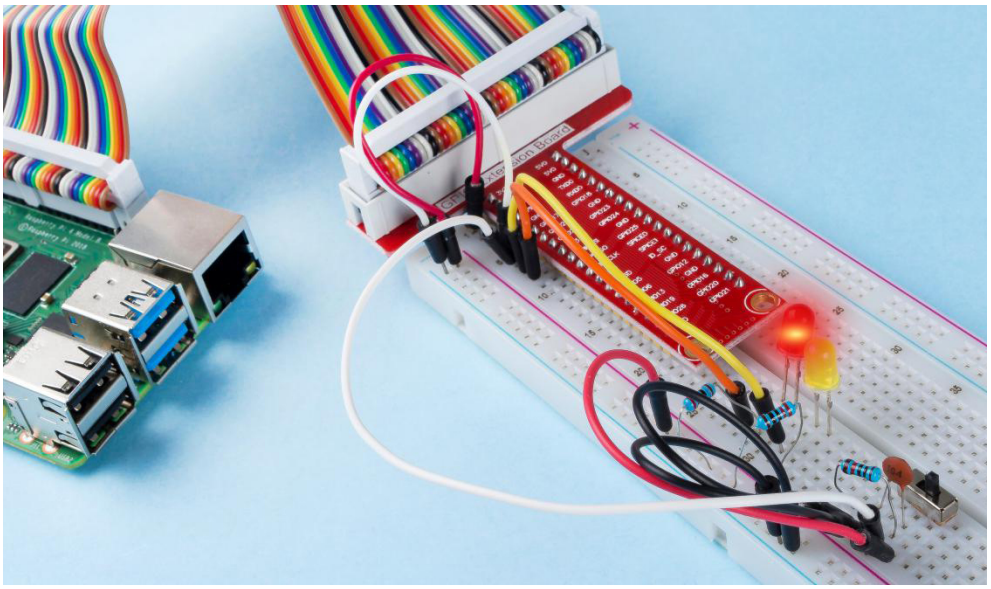
```
if(digitalRead(slidePin) == 1){
            digitalWrite(led1, LOW);
            digitalWrite(led2, HIGH);
            printf("LED1 on\n");
    }
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off

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

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off

Phenomenon Picture

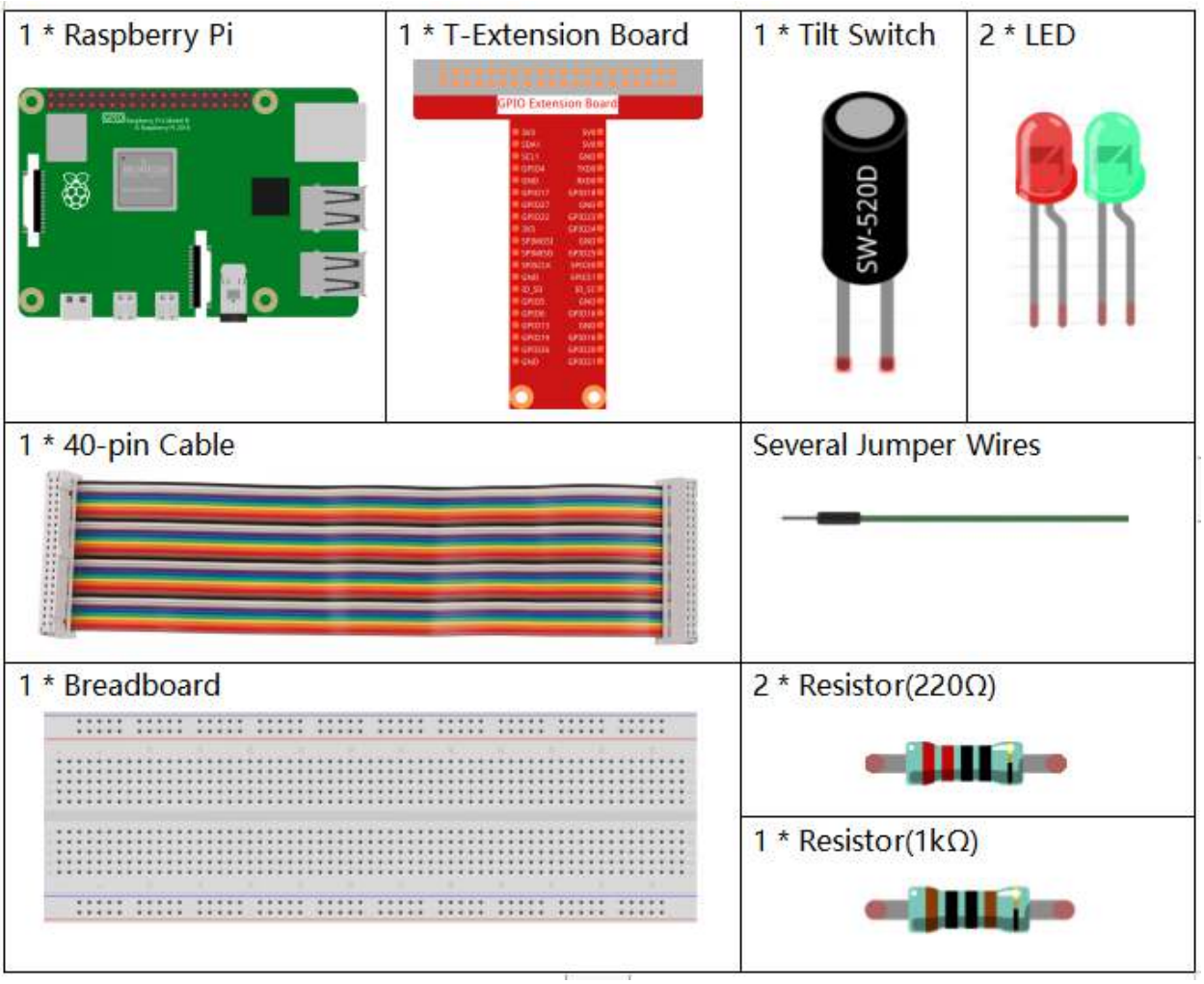


2.1.5 Tilt Switch

Introduction

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

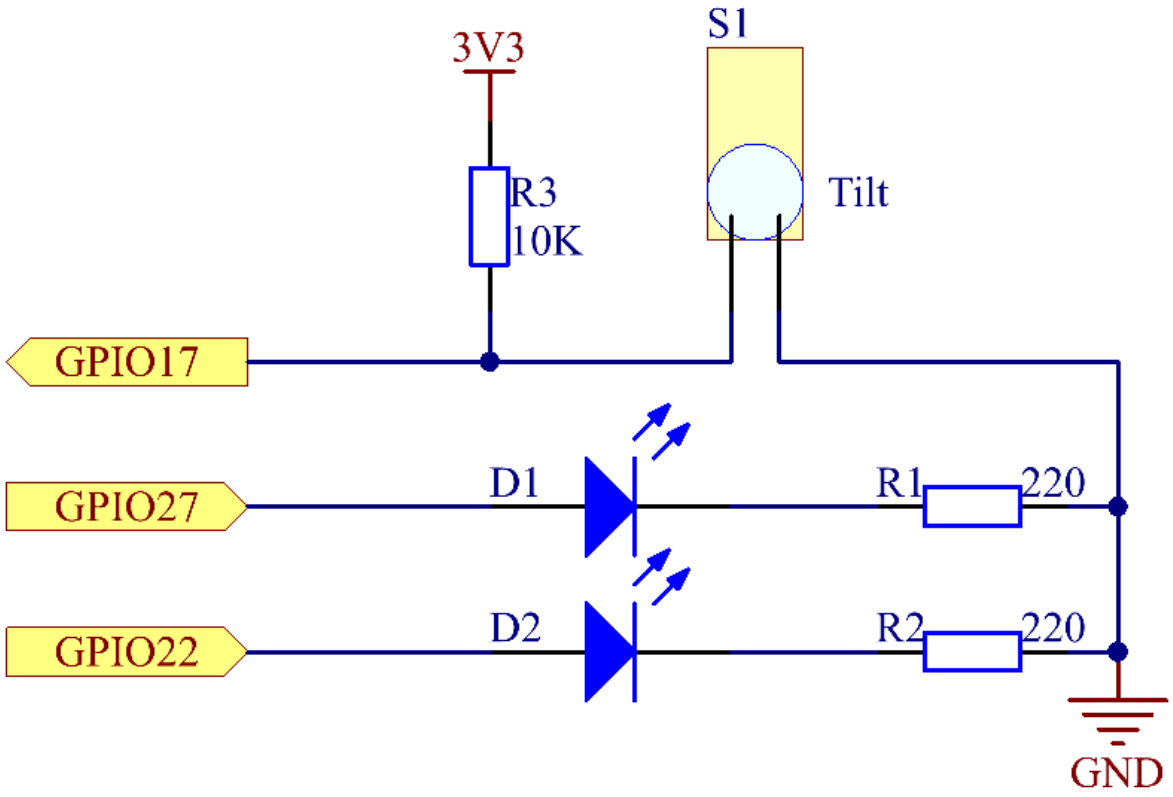
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Tilt Switch*

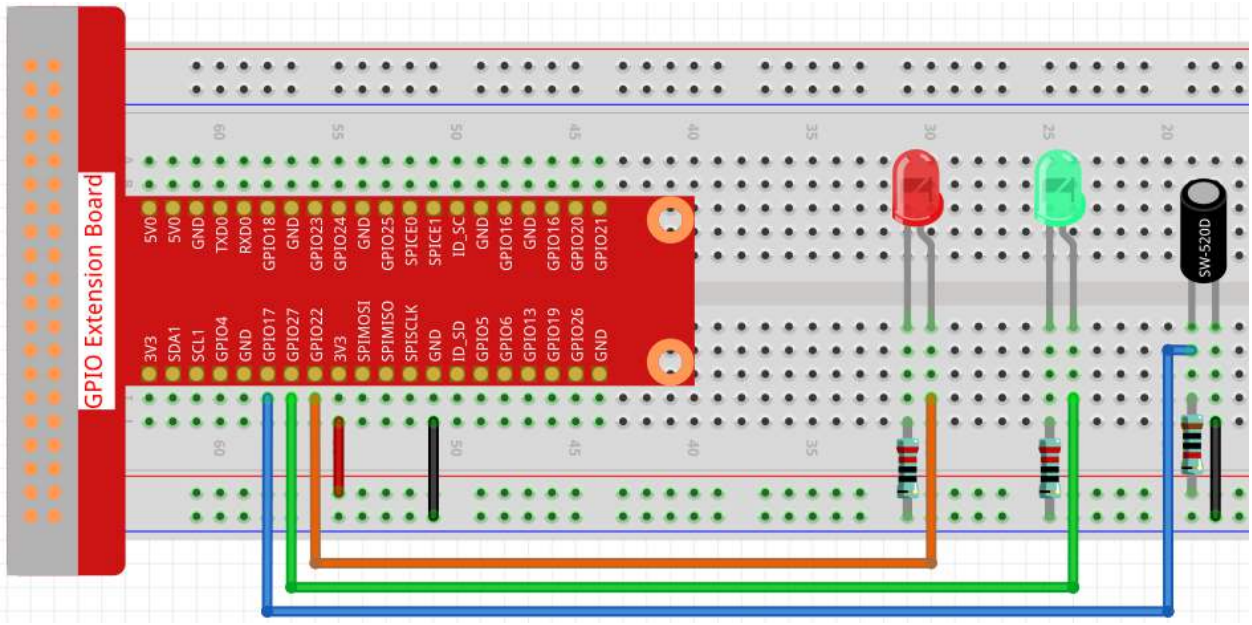
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/2.1.5/
```

Step 3: Compile.

```
gcc 2.1.5_Tilt.c -lwiringPi
```

Step 4: Run.

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

Place the tilt vertically, and the green LED will turns on. If you tilt it, "Tilt!" will be printed on the screen and the red LED will lights on. Place it vertically again, and the green LED will turns on again.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
```

(continues on next page)

(continued from previous page)

```
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}

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

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

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

Code Explanation

```
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");
```

(continues on next page)

(continued from previous page)

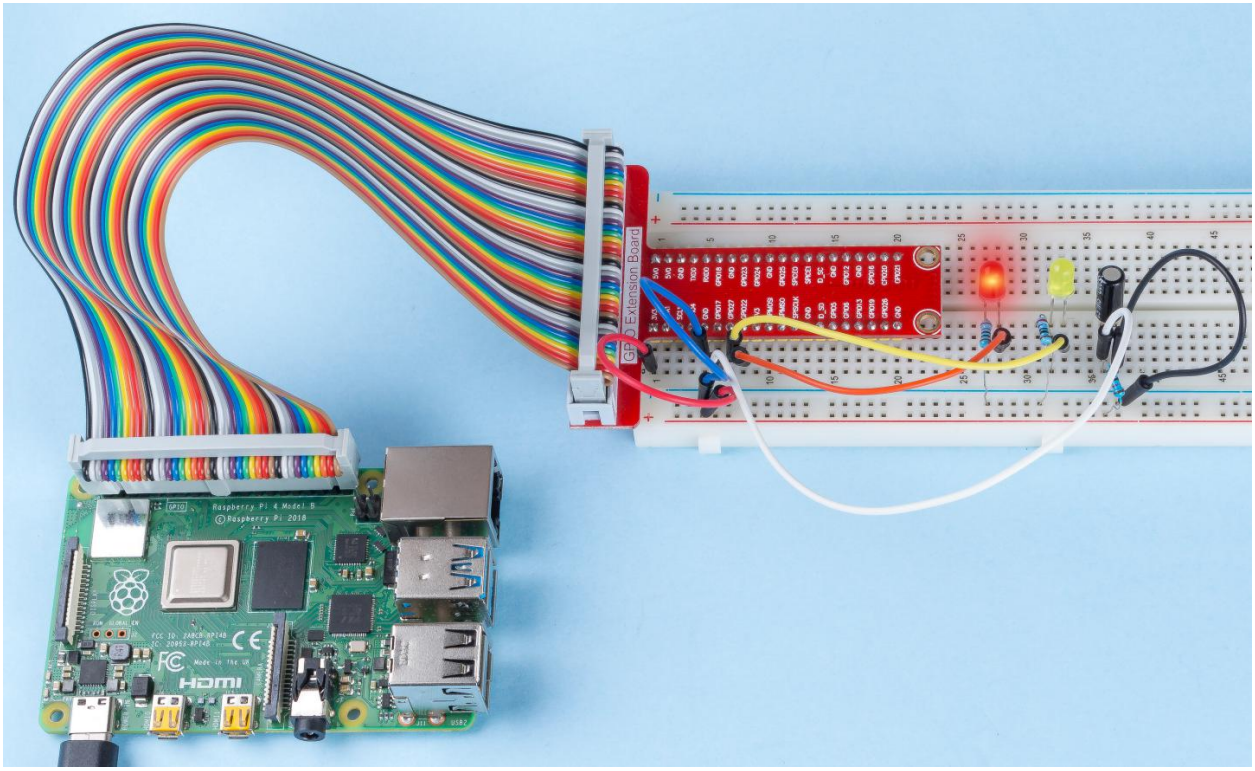
```
}
```

Define a function `LED()` to turn the two LEDs on or off. If the parameter color is RED, the red LED lights up; similarly, if the parameter color is GREEN, the green LED will turns on.

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

If the read value of tilt switch is 0, it means that the tilt switch is tilted then you write the parameter ”RED” into function LED to get the red LED lighten up; otherwise, the green LED will lit.

Phenomenon Picture

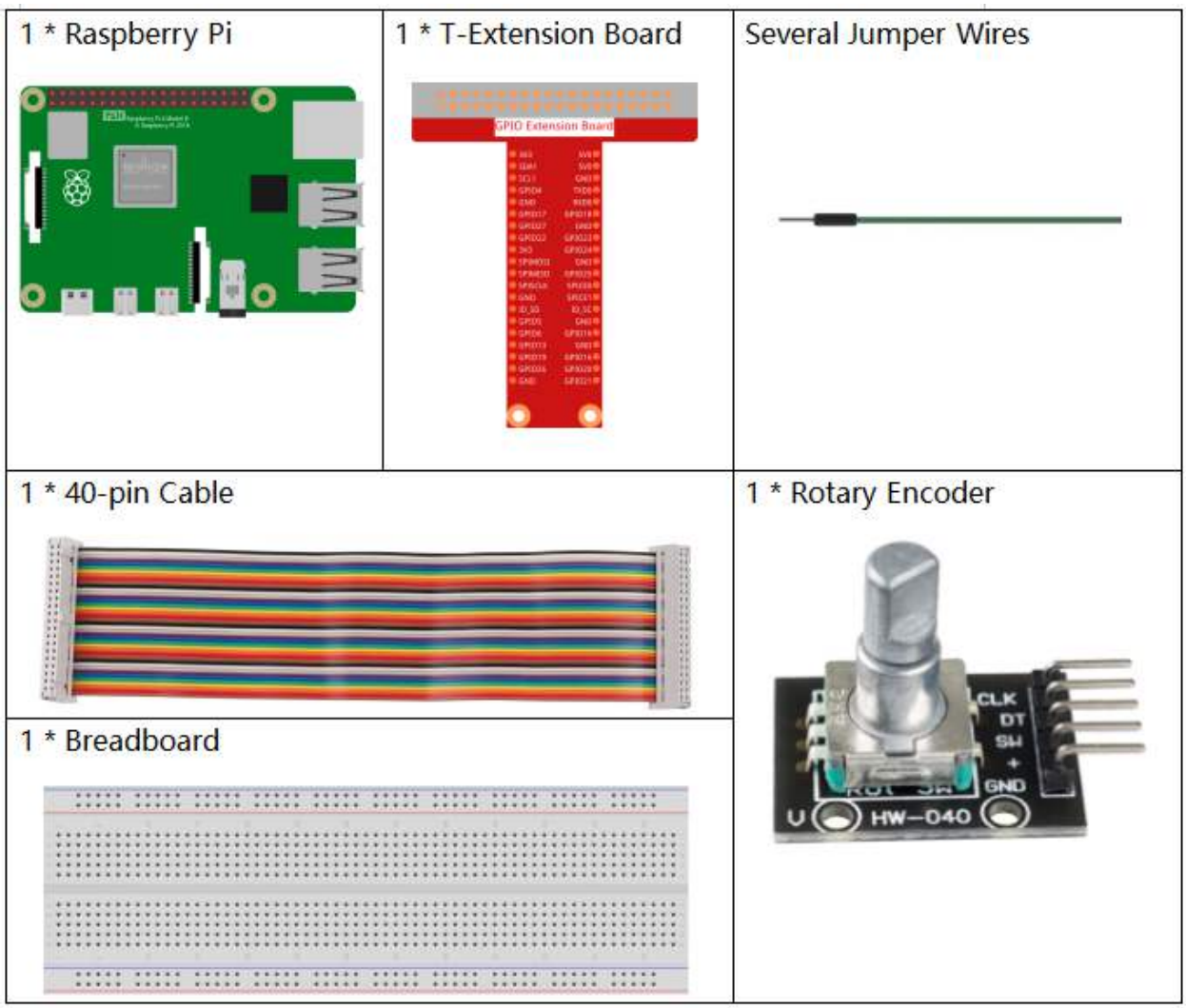


2.1.6 Rotary Encoder Module

Introduction

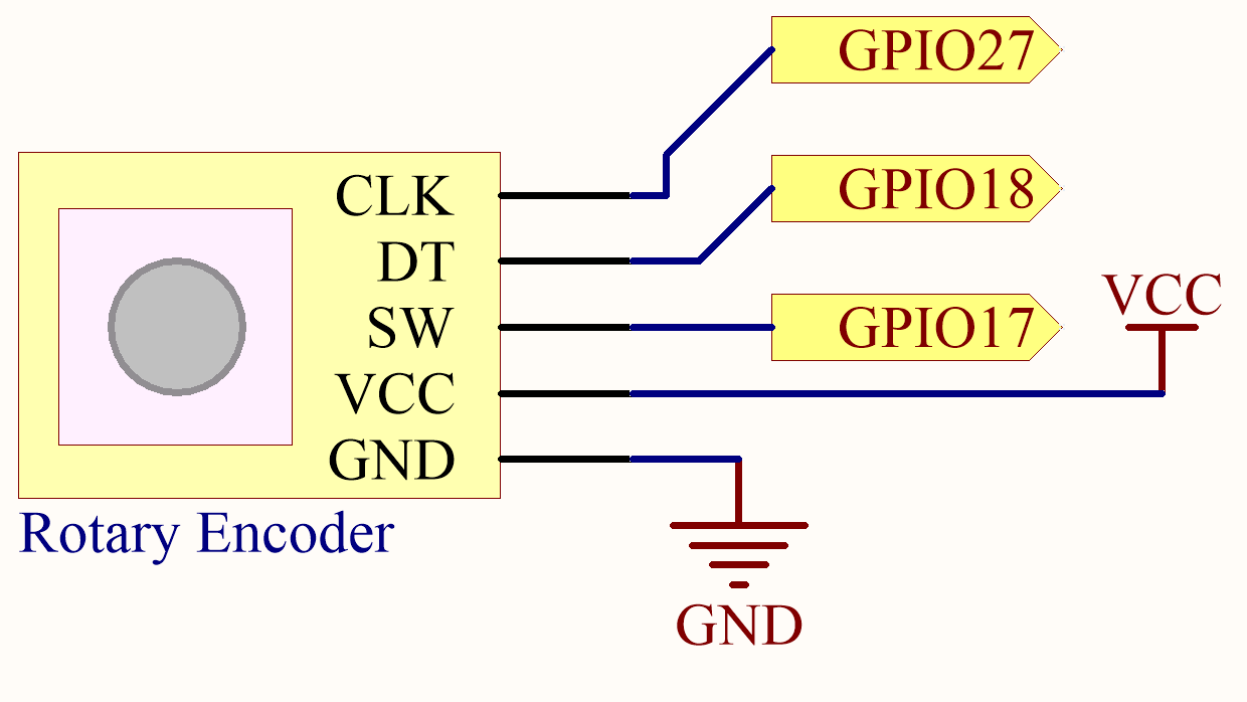
In this project, you will learn about Rotary Encoder. A rotary encoder is an electronic switch with a set of regular pulses in strictly timing sequence. When used with IC, it can achieve increment, decrement, page turning and other operations such as mouse scrolling, menu selection, and so on.

Components



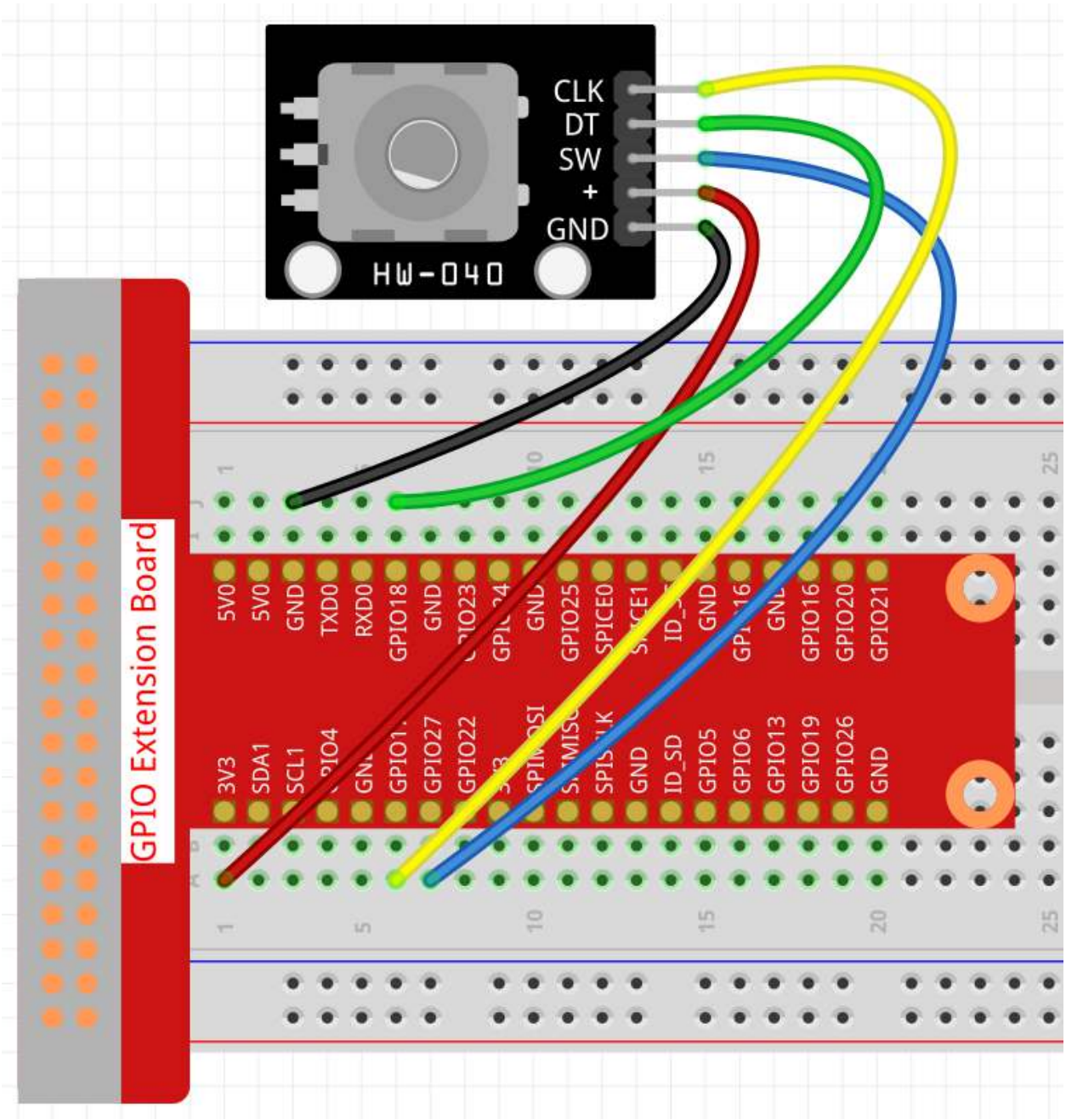
- *GPIO Extension Board*
- *Breadboard*
- *Rotary Encoder Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/2.1.6/
```

Step 3: Compile the code.

```
gcc 2.1.6_RotaryEncoder.c -lwiringPi
```

Step 4: Run.

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

You will see the count on the shell. When you turn the rotary encoder clockwise, the count is increased; when turn

it counterclockwise, the count is decreased. If you press the switch on the rotary encoder, the readings will return to zero.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <wiringPi.h>

#define  clkPin    0
#define  dtPin    1
#define  swPin     2

static volatile int globalCounter = 0 ;

unsigned char flag;
unsigned char Last_dtPin_Status;
unsigned char Current_dtPin_Status;

void btnISR(void)
{
   globalCounter = 0;
}

void rotaryDeal(void)
{
   Last_dtPin_Status = digitalRead(dtPin);

   while(!digitalRead(clkPin)){
      Current_dtPin_Status = digitalRead(dtPin);
      flag = 1;
   }

   if(flag == 1){
      flag = 0;
      if((Last_dtPin_Status == 0)&&(Current_dtPin_Status == 1)){
         globalCounter --;
      }
      if((Last_dtPin_Status == 1)&&(Current_dtPin_Status == 0)){
         globalCounter ++;
      }
   }
}

int main(void)
{
   if(wiringPiSetup() < 0){
      fprintf(stderr, "Unable to setup wiringPi:%s\n",strerror(errno));
      return 1;
   }
```

(continues on next page)

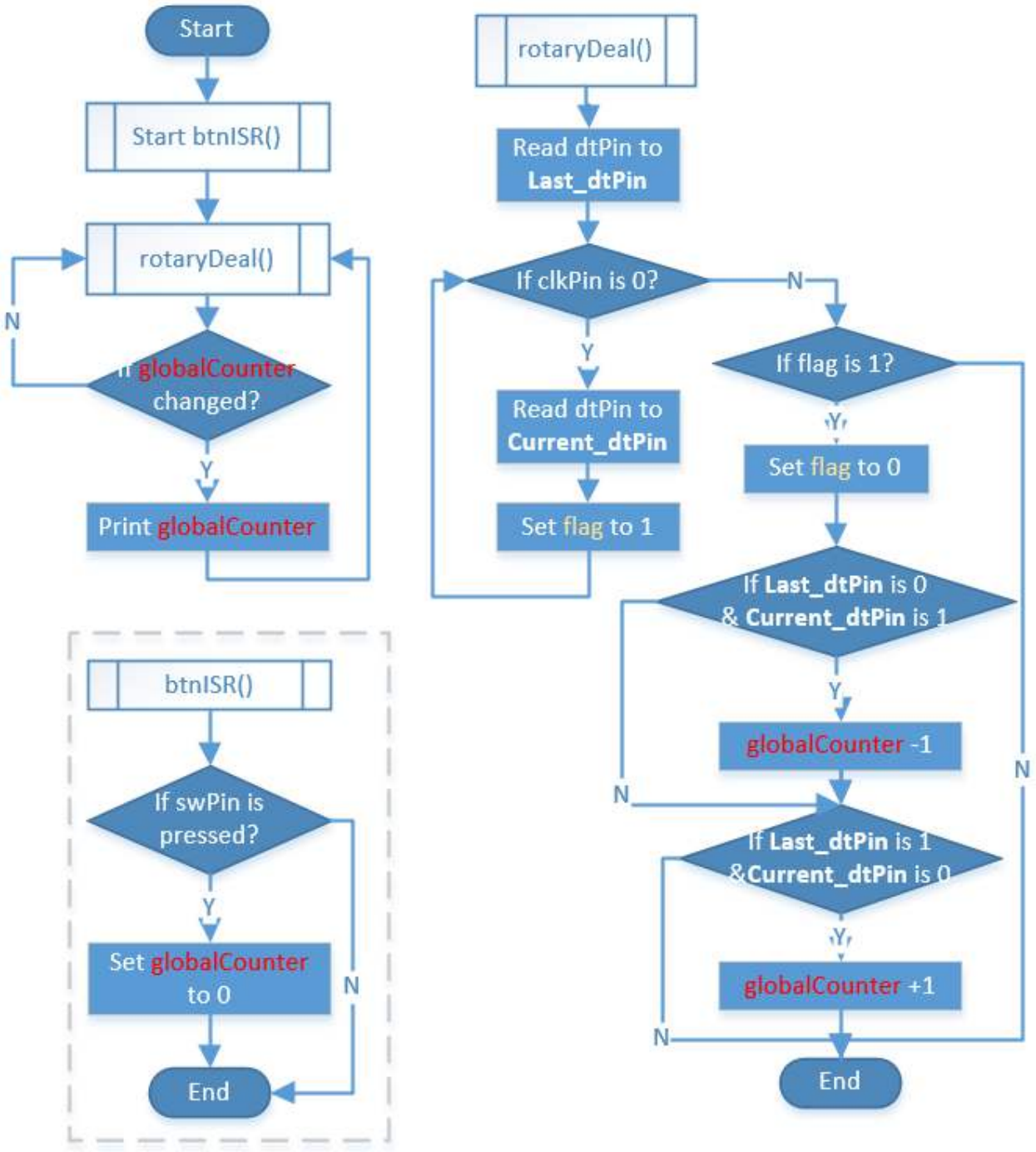
(continued from previous page)

```
    pinMode(swPin, INPUT);
    pinMode(clkPin, INPUT);
    pinMode(dtPin, INPUT);

    pullUpDnControl(swPin, PUD_UP);

    if(wiringPiISR(swPin, INT_EDGE_FALLING, &btnISR) < 0){
        fprintf(stderr, "Unable to init ISR\n",strerror(errno));
        return 1;
    }

    int tmp = 0;

    while(1){
        rotaryDeal();
        if (tmp != globalCounter){
            printf("%d\n", globalCounter);
            tmp = globalCounter;
        }
    }

    return 0;
}
```

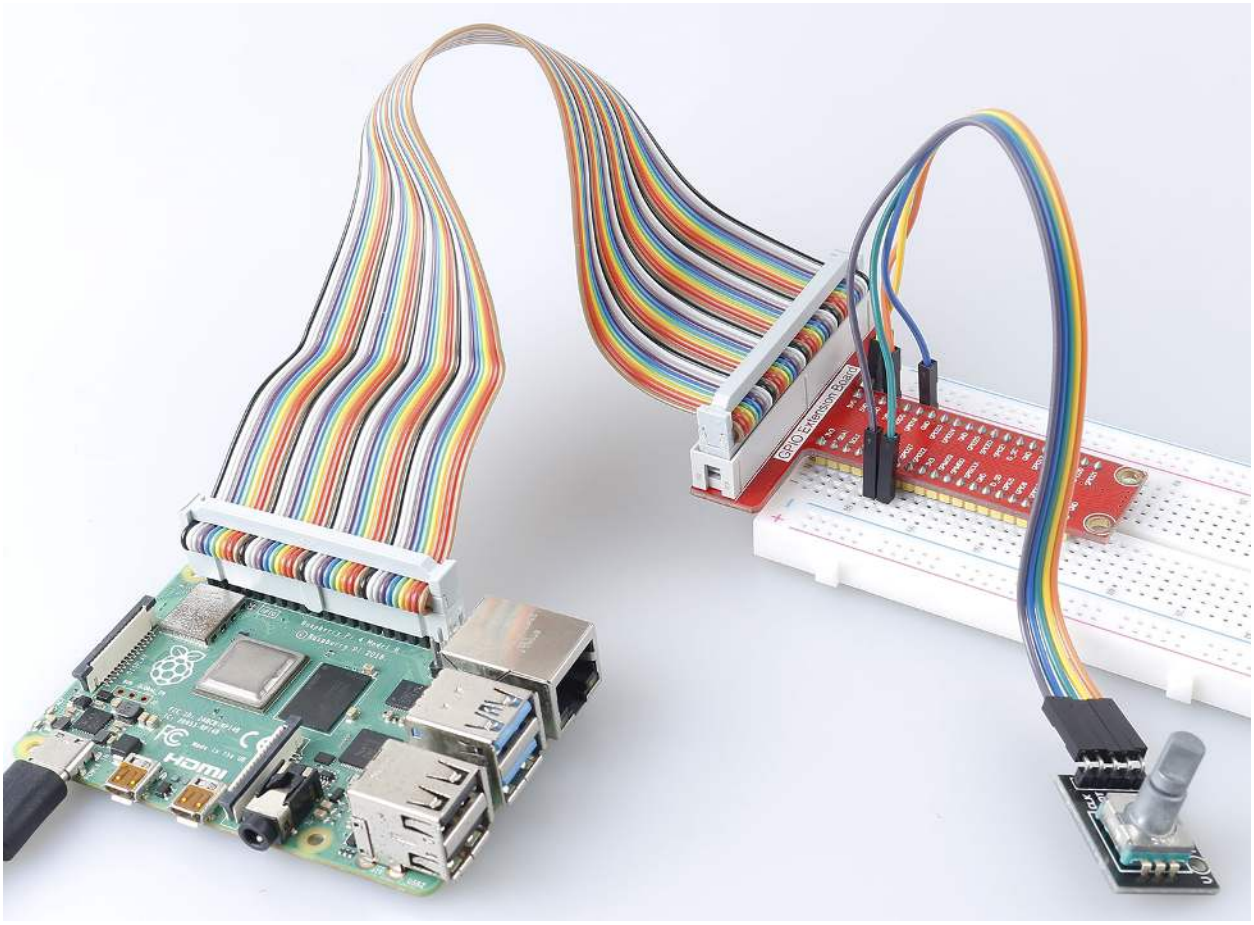
Code Analysis

- Read dtPin value when clkPin is low.
- When clkPin is high, if dtPin goes from low to high, the count decreases, otherwise the count increases.
- swPin will output low when the shaft is pressed.

From this, the program flow is shown below:


Start
Start btnISR()
rotaryDeal()
N
globalCounter changed?
Y
Print globalCounter
btnISR()
If swPin is pressed?
Y
Set globalCounter to 0
N
End
rotaryDeal()
Read dtPin to Last_dtPin
If clkPin is 0?
N
Y
Read dtPin to Current_dtPin
Set flag to 1
If flag is 1?
Y
Set flag to 0
If Last_dtPin is 0 & Current_dtPin is 1
Y
globalCounter -1
N
If Last_dtPin is 1 &Current_dtPin is 0
Y
globalCounter +1
N
N
End

Phenomenon Picture

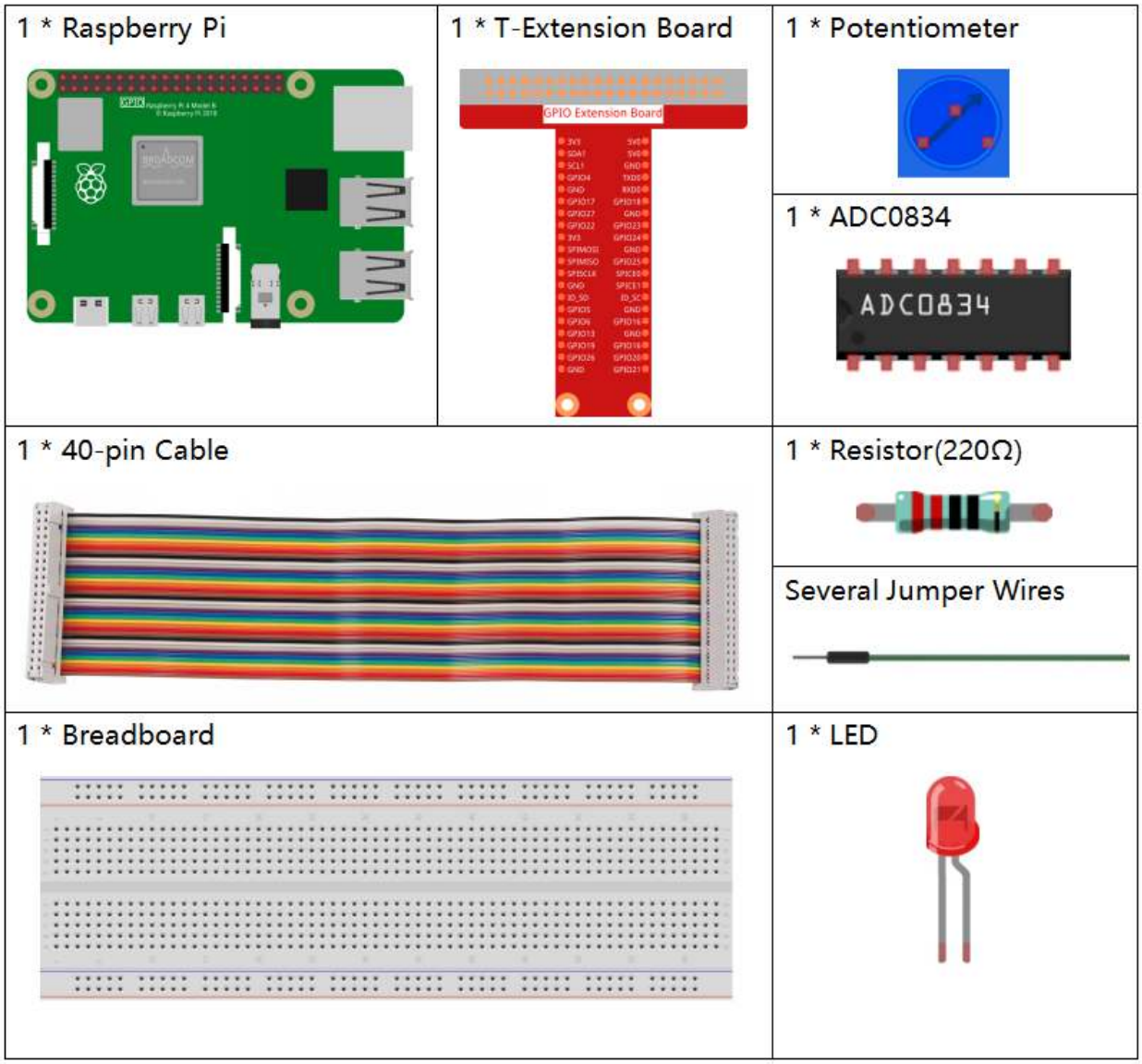


2.1.7 Potentiometer

Introduction

The ADC function can be used to convert analog signals to digital signals, and in this experiment, ADC0834 is used to get the function involving ADC. Here, we implement this process by using potentiometer. Potentiometer changes the physical quantity – voltage, which is converted by the ADC function.

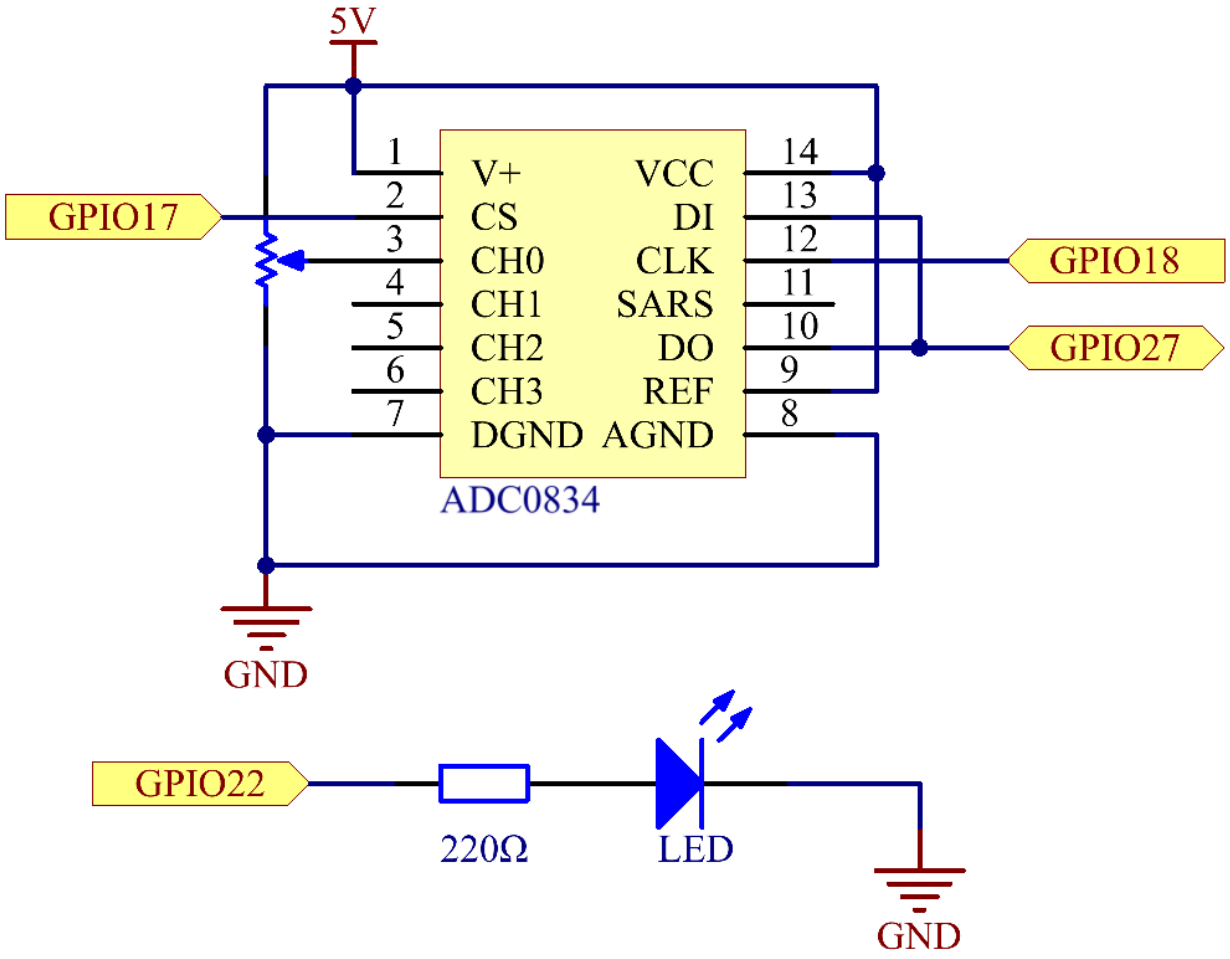
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Potentiometer*
- *ADC0834*

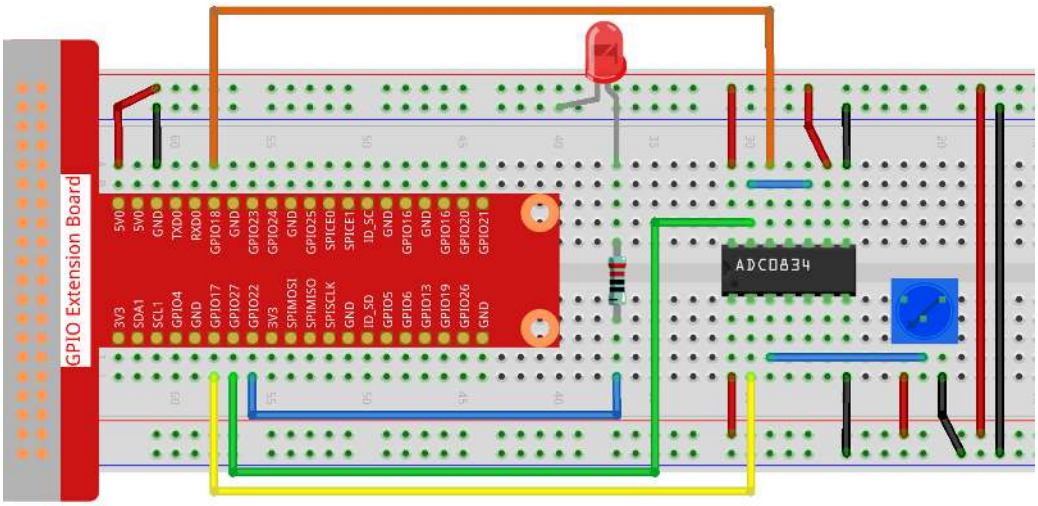
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Note: Please place the chip by referring to the corresponding position depicted in the picture. Note that the grooves on the chip should be on the left when it is placed.

Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/2.1.7/
```

Step 3: Compile the code.

```
gcc 2.1.7_Potentiometer.c -lwiringPi
```

Step 4: Run.

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

After the code runs, rotate the knob on the potentiometer, the intensity of LED will change accordingly.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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
```

(continues on next page)

(continued from previous page)

```
#define     LedPin    3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
//Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
```

(continues on next page)

(continued from previous page)

```
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(LedPin,  0, 100);
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

Code Explanation

```
#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2
#define     LedPin    3
```

Define CS, CLK, DIO of ADC0834, and connect them to GPIO0, GPIO1 and GPIO2 respectively. Then attach LED to GPIO3.

```
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++)
```

(continues on next page)

(continued from previous page)

```
    {
        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;
}
```

There is a function of ADC0834 to get Analog to Digital Conversion. The specific workflow is as follows:

```
digitalWrite(ADC_CS, 0);
```

Set CS to low level and start enabling AD conversion.

```
// Start bit
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

When the low-to-high transition of the clock input occurs at the first time, set DIO to 1 as Start bit. In the following three steps, there are 3 assignment words.

```
//Single End mode
digitalWrite(ADC_CLK,0);
igitalWrite(ADC_DIO,1);    delayMicroseconds(2);
gitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

As soon as the low-to-high transition of the clock input occurs for the second time, set DIO to 1 and choose SGL mode.

```
// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

Once occurs for the third time, the value of DIO is controlled by the variable **odd**.

```
//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);
```

The pulse of CLK converted from low level to high level for the forth time, the value of DIO is controlled by the variable **sel**.

Under the condition that channel=0, sel=0, odd=0, the operational formulas concerning **sel** and **odd** are as follows:

```
int sel = channel > 1 & 1;
int odd = channel & 1;
```

When the condition that channel=1, sel=0, odd=1 is met, please refer to the following address control logic table. Here CH1 is chosen, and the start bit is shifted into the start location of the multiplexer register and conversion starts.

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);
```

Here, set DIO to 1 twice, please ignore it.

```
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);
    }
```

In the first for() statement, as soon as the fifth pulse of CLK is converted from high level to low level, set DIO to input mode. Then the conversion starts and the converted value is stored in the variable dat1. After eight clock periods, the conversion is complete.

```
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);
    }
```

In the second for() statement, output the converted values via DO after other eight clock periods and store them in the variable dat2.

```
digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
return(dat1==dat2) ? dat1 : 0;
```

return(dat1==dat2) ? dat1 : 0 is used to compare the value gotten during the conversion and the output value. If they are equal to each other, output the converting value dat1; otherwise, output 0. Here, the workflow of ADC0834 is complete.

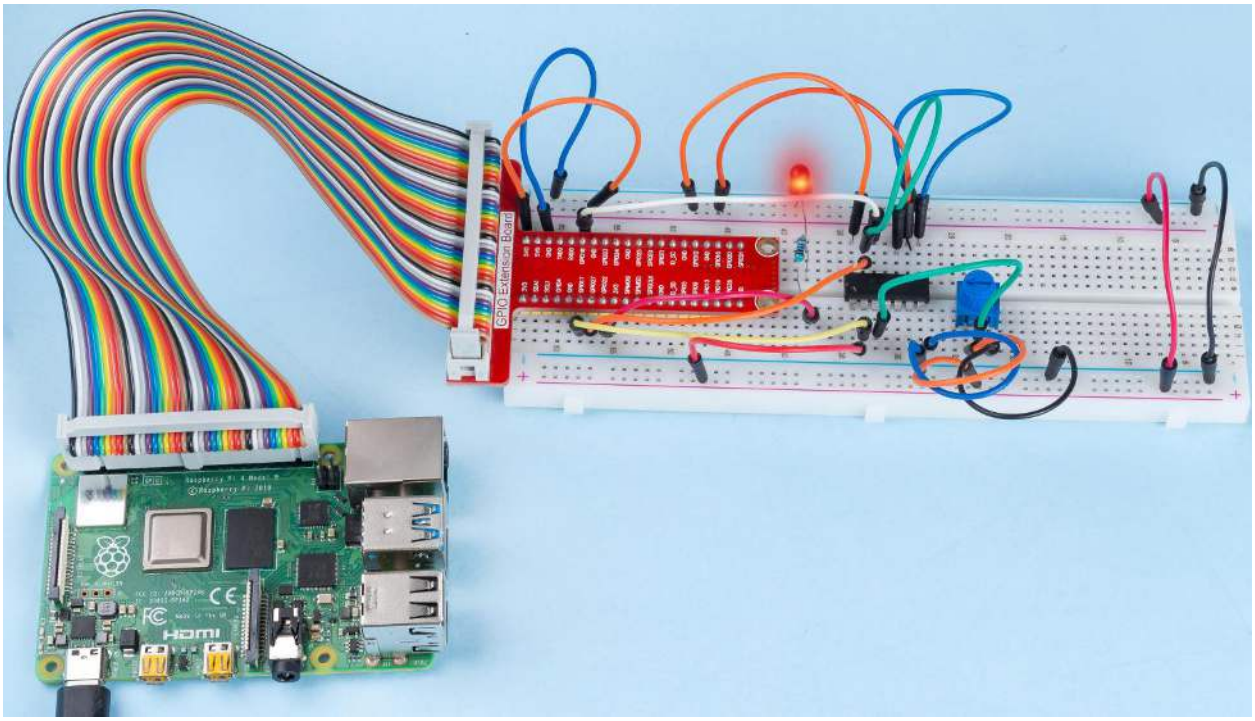
```
softPwmCreate(LedPin,  0, 100);
```

The function is to use software to create a PWM pin, LedPin, then the initial pulse width is set to 0, and the period of PWM is 100 x 100us.

```
while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
```

In the main program, read the value of channel 0 that has been connected with a potentiometer. And store the value in the variable analogVal then write it in LedPin. Now you can see the brightness of LED changing with the value of the potentiometer.

Phenomenon Picture

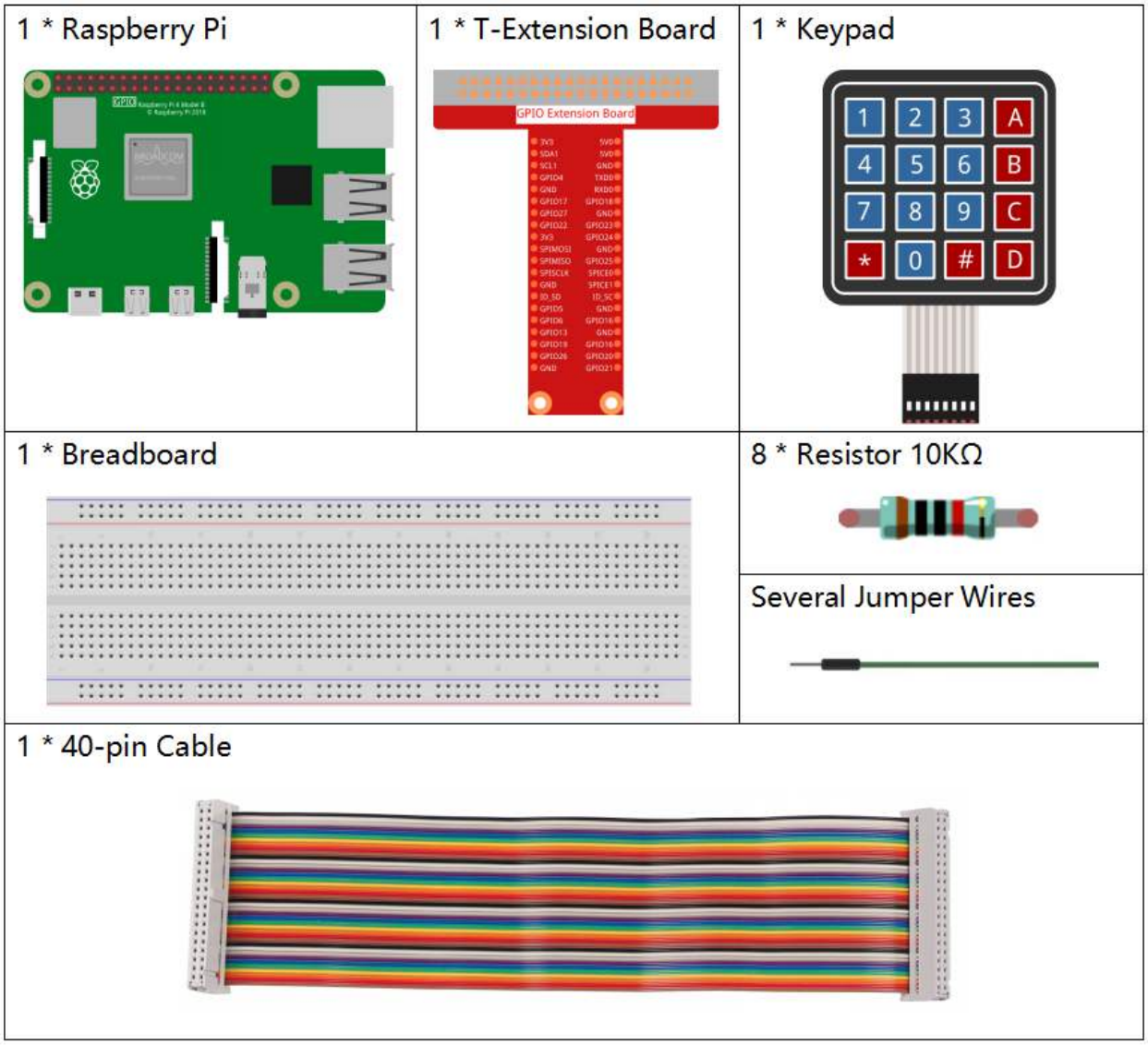


2.1.8 Keypad

Introduction

A keypad is a rectangular array of buttons. In this project, We will use it input characters.

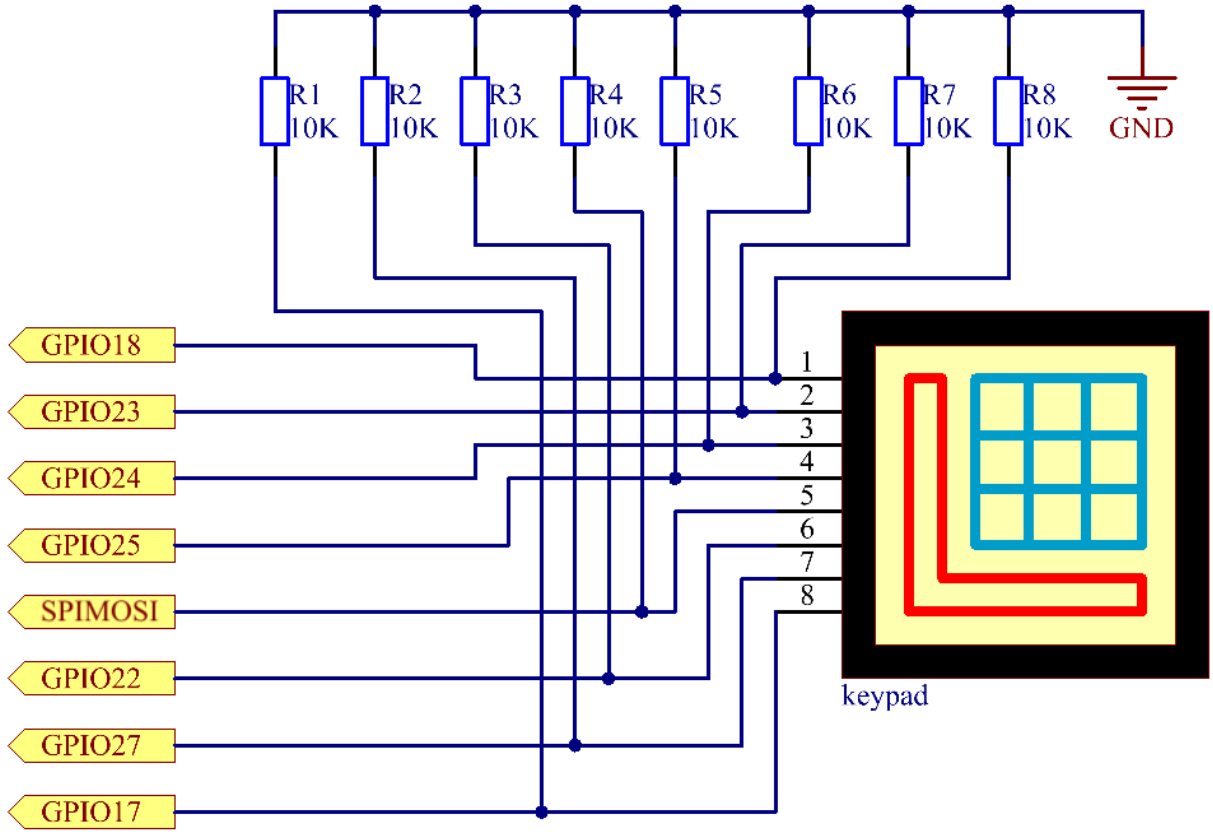
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Keypad*

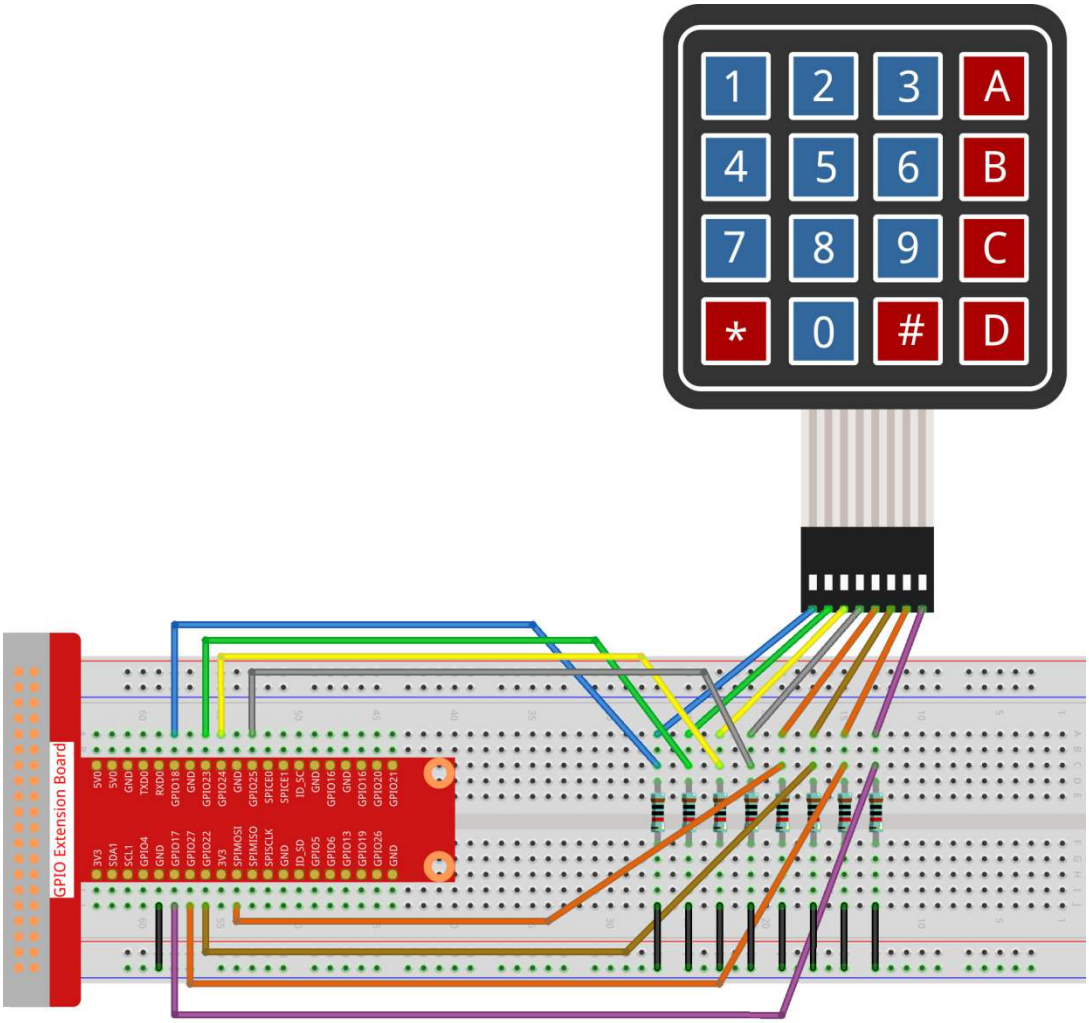
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/2.1.8/
```

Step 3: Compile the code.

```
gcc 2.1.8_Keypad.cpp -lwiringPi
```

Step 4: Run.

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

After the code runs, the values of pressed buttons on keypad (button Value) will be printed on the screen.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

(continues on next page)

(continued from previous page)

```
#define ROWS  4
#define COLS  4
#define BUTTON_NUM (ROWS * COLS)

unsigned char KEYS[BUTTON_NUM] {
'1','2','3','A',
'4','5','6','B',
'7','8','9','C',
'*','0','#','D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};

void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);

void init(void) {
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
        pinMode(colPins[i], INPUT);
    }
}

int main(void){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            keyPrint(pressed_keys);
            keyCopy(last_key_pressed, pressed_keys);
        }
        delay(100);
    }
    return 0;
}

void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
    for(int i=0 ; i<ROWS ; i++ ){
        digitalWrite(rowPins[i], HIGH);
        for(int j =0 ; j < COLS ; j++){
            index = i * ROWS + j;
```

(continues on next page)

(continued from previous page)

```
            if(digitalRead(colPins[j]) == 1){
                result[count]=KEYS[index];
                count += 1;
            }
        }
        delay(1);
        digitalWrite(rowPins[i], LOW);
    }
}

bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}

void keyPrint(unsigned char* a){
    if (a[0] != 0){
        printf("%c",a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c",a[i]);
        }
    }
    printf("\n");
}

void keyClear(unsigned char* a){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = 0;
    }
}

int keyIndexOf(const char value){
    for (int i=0; i<BUTTON_NUM; i++){
        if ((const char)KEYS[i] == value){
            return i;
        }
    }
    return -1;
}
```

Code Explanation

```
unsigned char KEYS[BUTTON_NUM] {
'1','2','3','A',
'4','5','6','B',
```

(continues on next page)

(continued from previous page)

```
'7','8','9','C',
'*','0','#','D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};
```

Declare each key of the matrix keyboard to the array `keys[]` and define the pins on each row and column.

```
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);
    }
```

This is the part of the main function that reads and prints the button value.

The function `keyRead()` will read the state of every button.

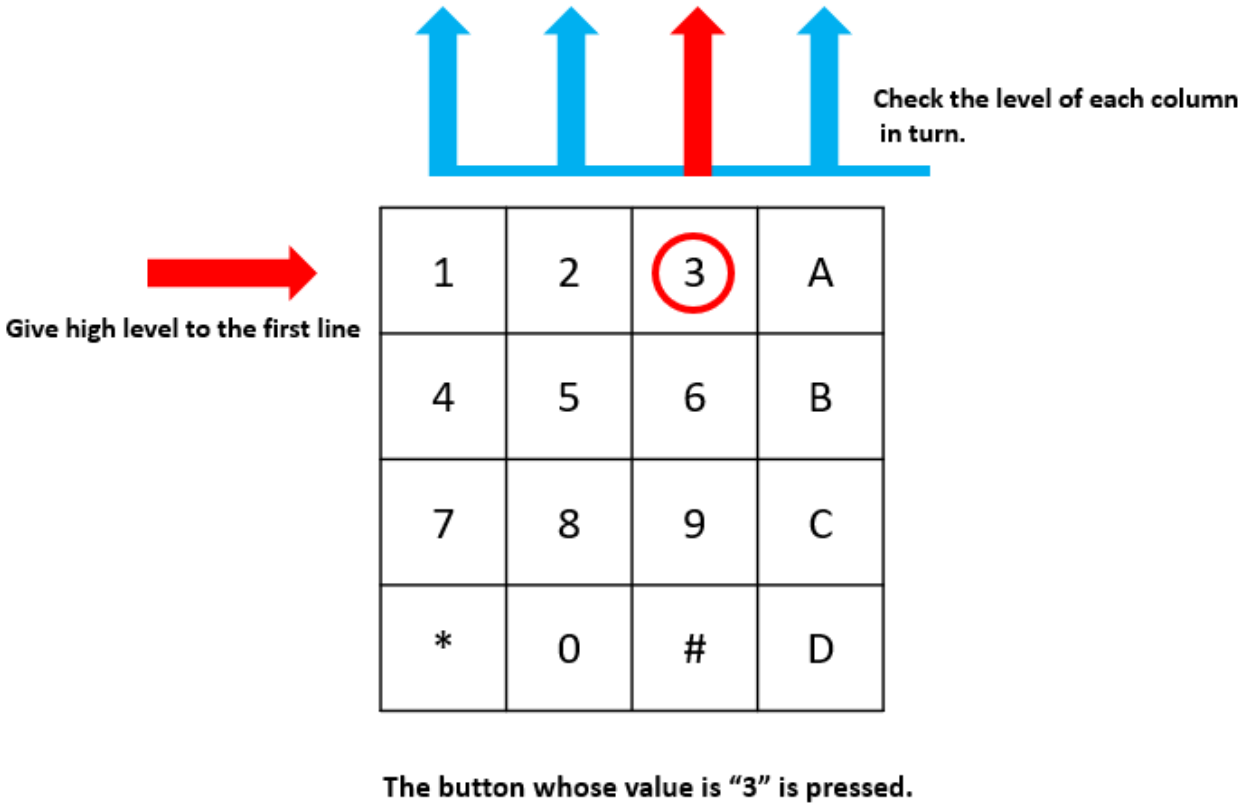
`KeyCompare()` and `keyCopy()` are used to judge whether the state of a button has changed (that is, a button has been pressed or released).

`keyPrint()` will print the button value of the button whose current level is high level (the button is pressed).

```
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);
    }
}
```

This function assigns a high level to each row in turn, and when the key in the column is pressed, the column in which the key is located gets a high level. After the two-layer loop judgment, the key state compilation will generate an array (`reasult[]`).

When pressing button 3:



The button whose value is "3" is pressed.

`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.

After the loop judgment is completed, an array will be generated:

```
result[BUTTON_NUM] {
0, 0, 1, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0};
```

```
bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}
```

These two functions are used to judge whether the key state has changed, for example when you release your hand

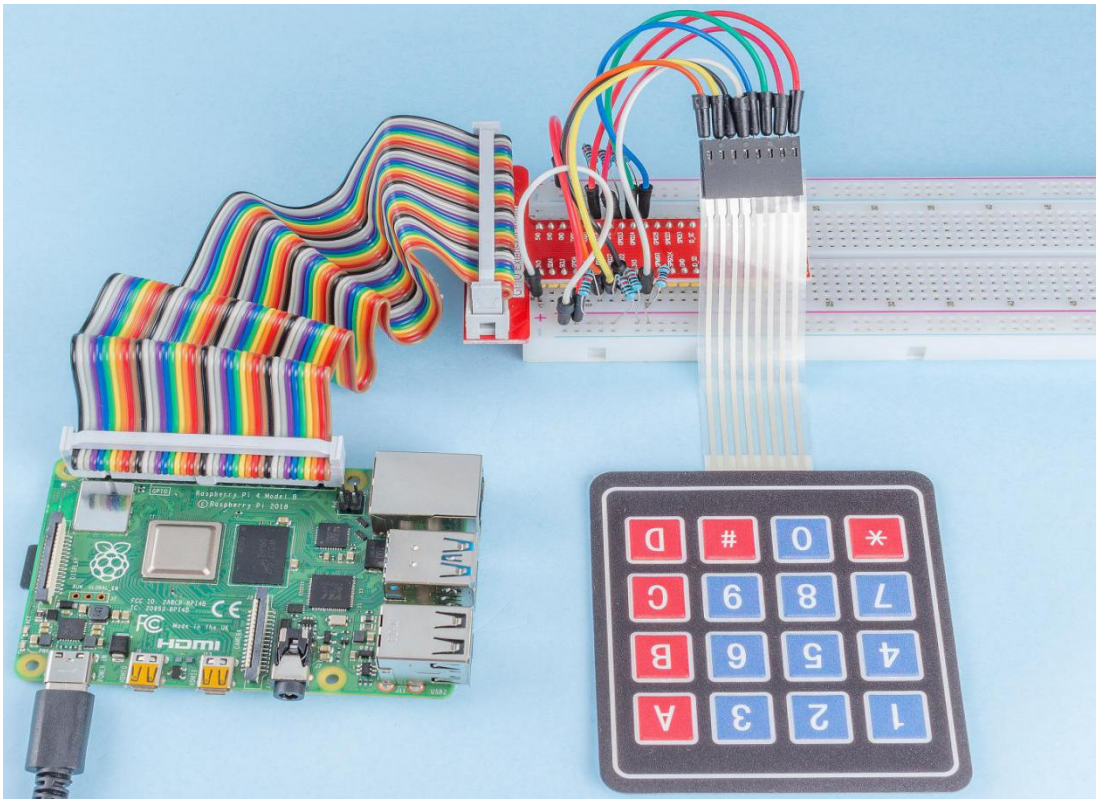
when pressing '3' or pressing '2', keyCompare() returns false.

KeyCopy() is used to re-write the current button value for the a array (last_key_pressed[BUTTON_NUM]) after each comparison. So we can compare them next time.

```
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");
}
```

This function is used to print the value of the button currently pressed. If the button '1' is pressed, the '1' will be printed. If the button '1' is pressed and the button '3' is pressed, the '1, 3' will be printed.

Phenomenon Picture

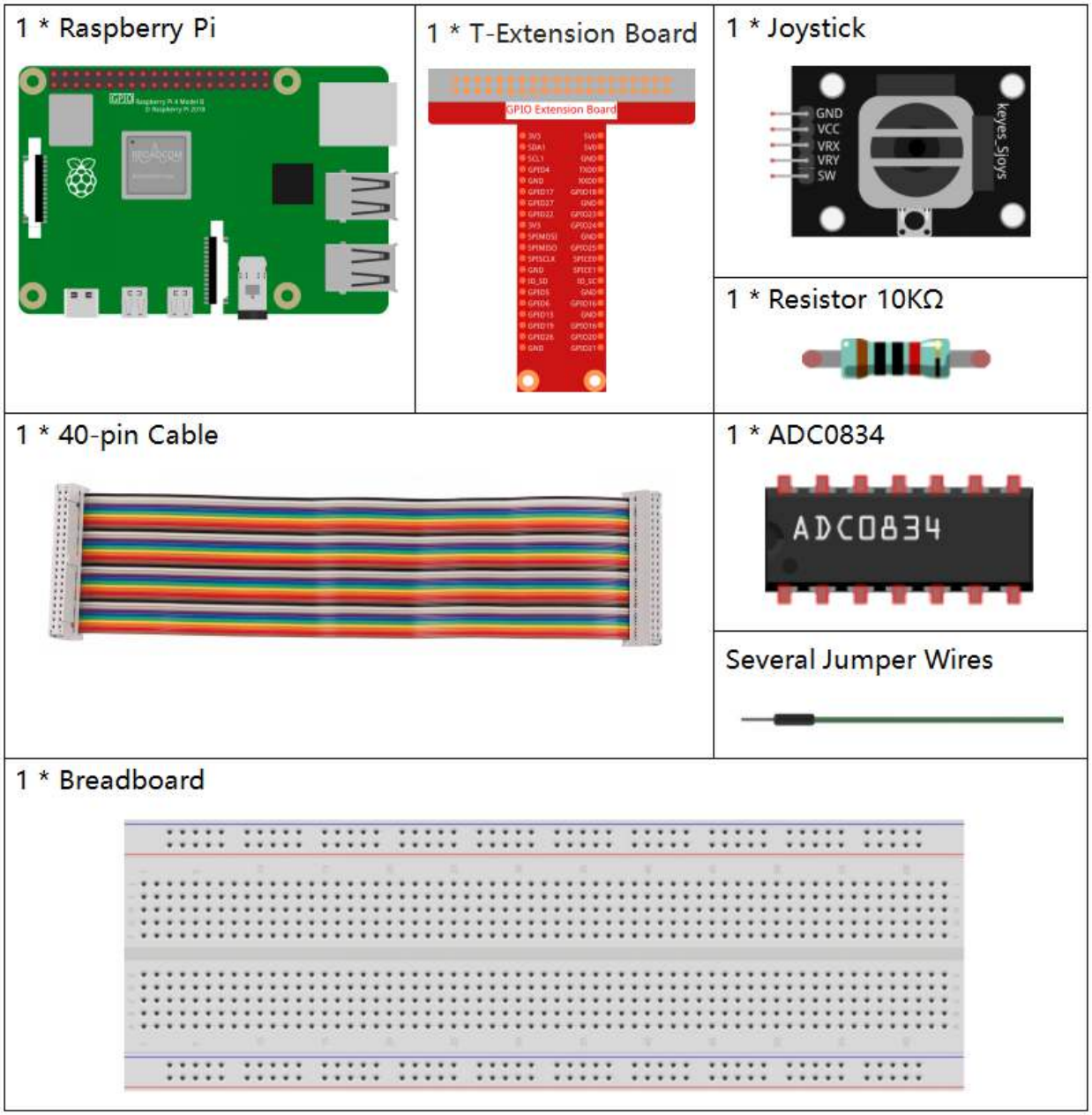


2.1.9 Joystick

Introduction

In this project, We're going to learn how joystick works. We manipulate the Joystick and display the results on the screen.

Components



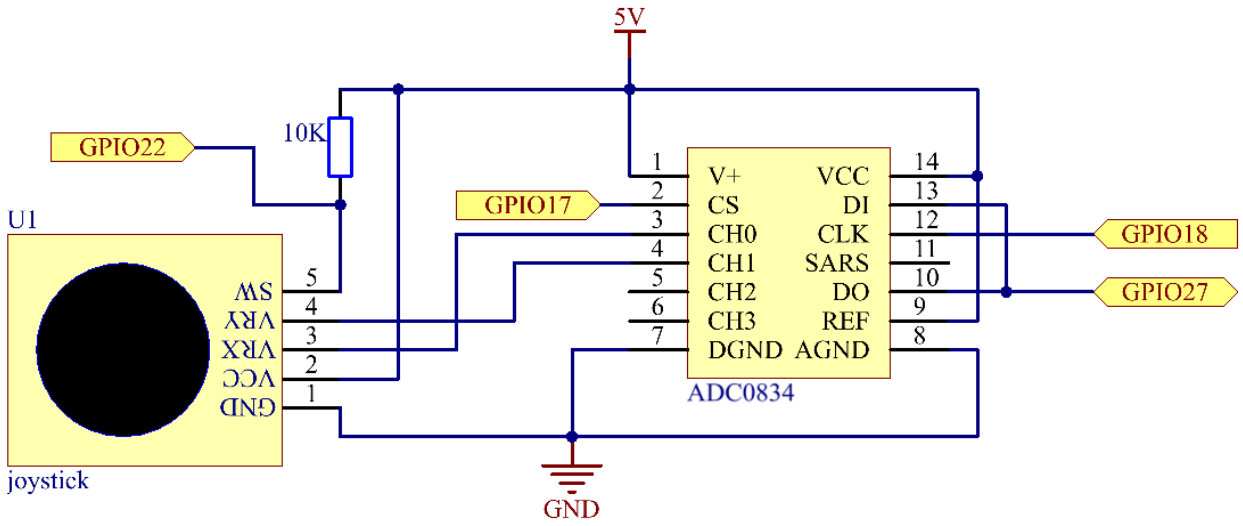
- *GPIO Extension Board*
- *Breadboard*

- *Resistor*
- *Joystick Module*
- *ADC0834*

Schematic Diagram

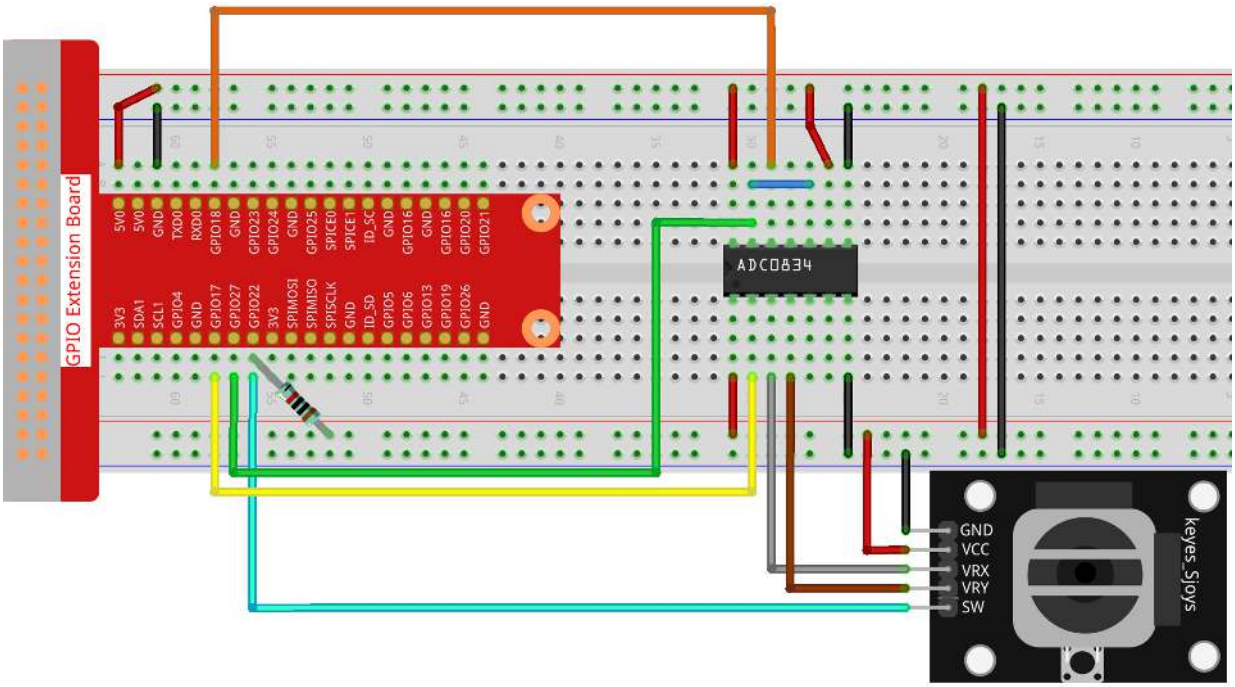
When the data of joystick is read, there are some differents between axis: data of X and Y axis is analog, which need to use ADC0834 to convert the analog value to digital value. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.1.9/
```

Step 3: Compile the code.

```
gcc 2.1.9_Joystick.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, turn the Joystick, then the corresponding values of x, y, Btn are displayed on screen.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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
```

(continues on next page)

(continued from previous page)

```
#define     ADC_DIO   2
#define     BtnPin    3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
//Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
digitalWrite(ADC_CLK,1);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }
    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }
    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}
int main(void)
{
    uchar x_val;
    uchar y_val;
    uchar btn_val;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(BtnPin,  INPUT);
```

(continues on next page)

(continued from previous page)

```
    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 Explanation

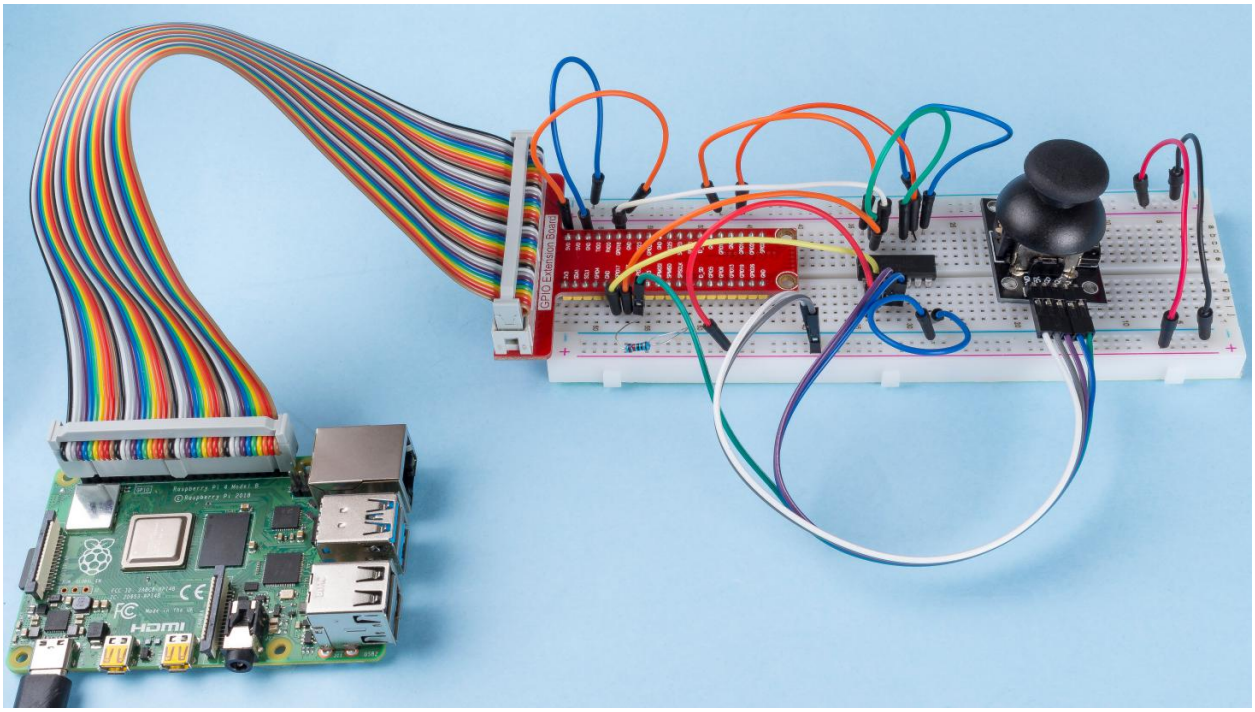
```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    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);
    ......
```

The working process of the function is detailed in 2.1.4 Potentiometer.

```
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 and VRY of Joystick are connected to CH0, CH1 of ADC0834 respectively. So the function getResult() is called to read the values of CH0 and CH1. Then the read values should be stored in the variables x_val and y_val. In addition, read the value of SW of joystick and store it into the variable Btn_val. Finally, the values of x_val, y_val and Btn_val shall be printed with print() function.

Phenomenon Picture



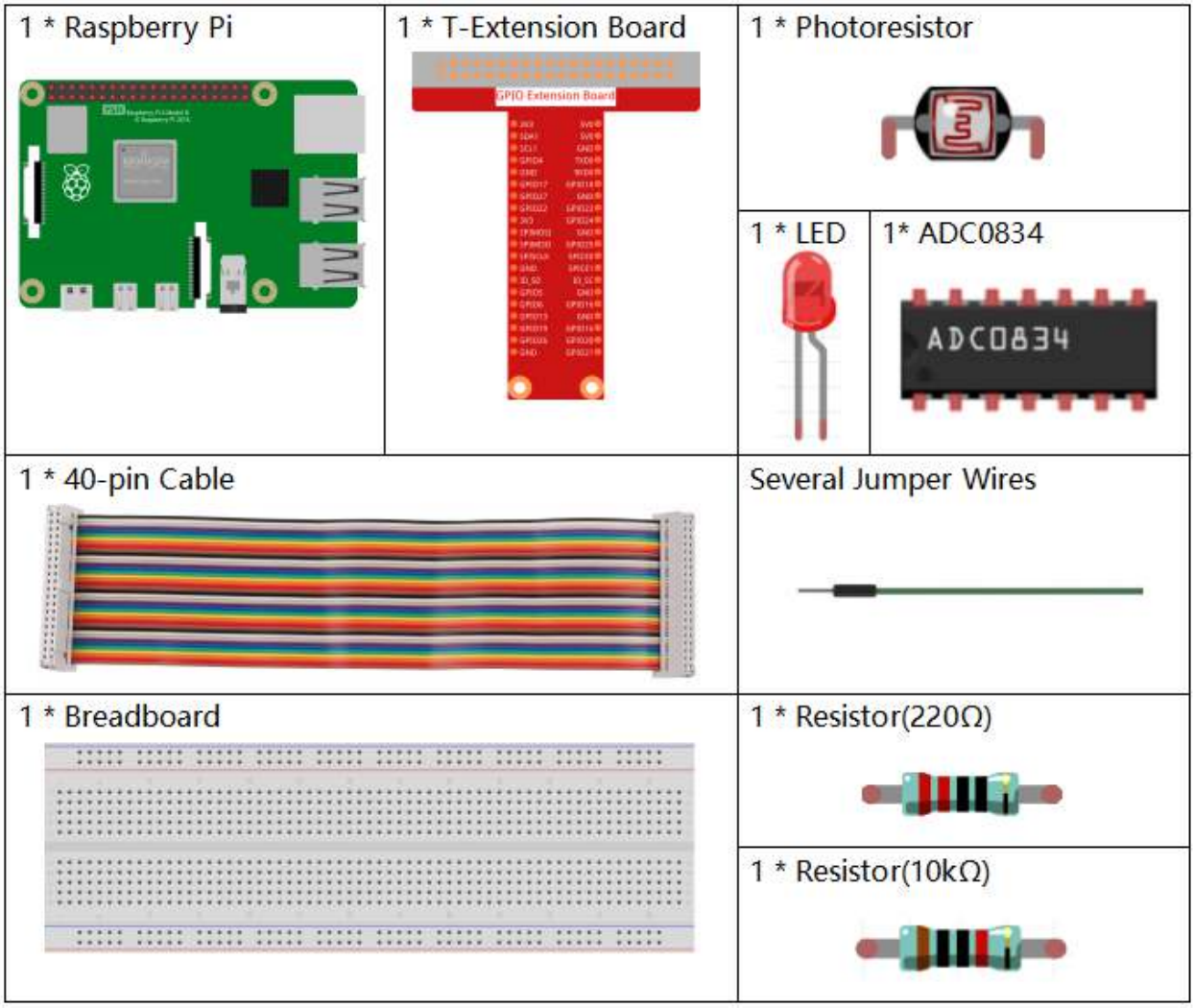
7.3.2 2.2 Sensors

2.2.1 Photoresistor

Introduction

Photoresistor is a commonly used component of ambient light intensity in life. It helps the controller to recognize day and night and realize light control functions such as night lamp. This project is very similar to potentiometer, and you might think it changing the voltage to sensing light.

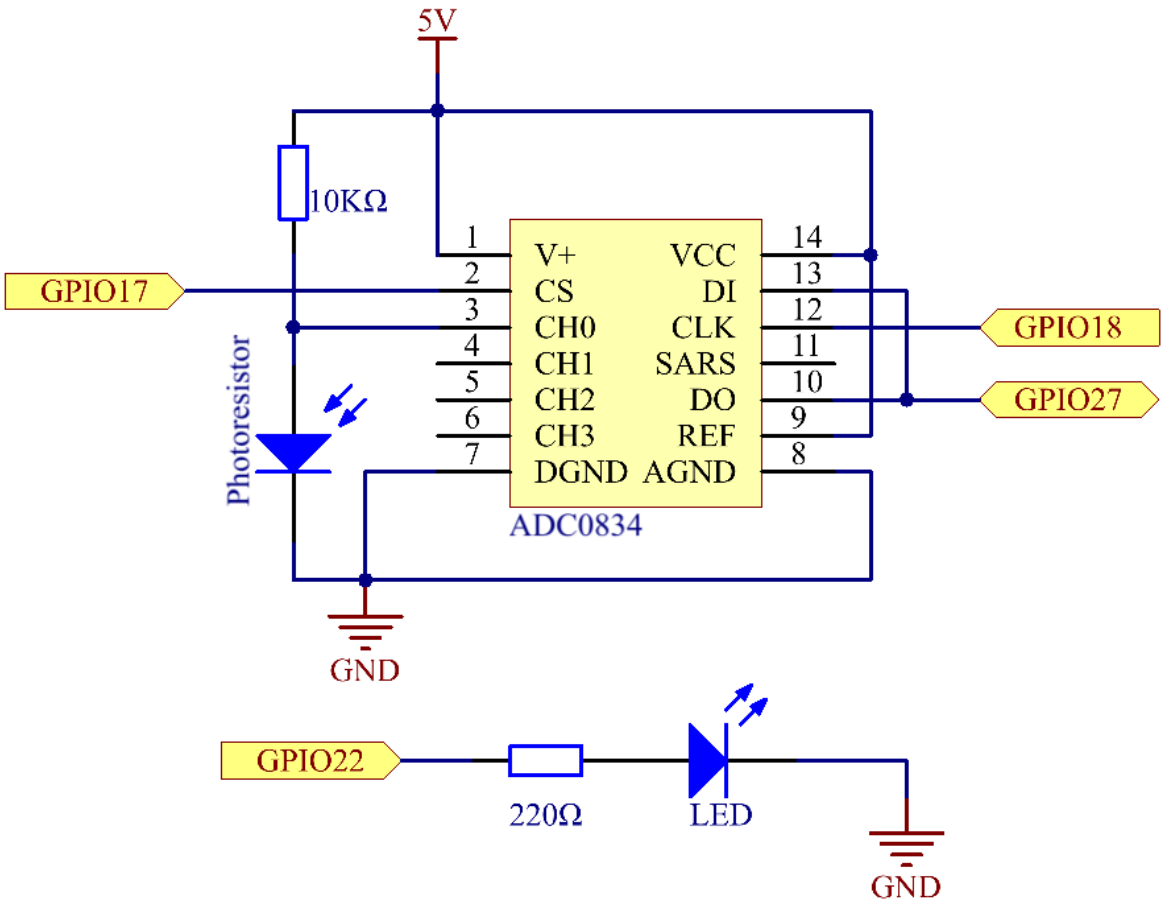
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *ADC0834*
- *Photoresistor*

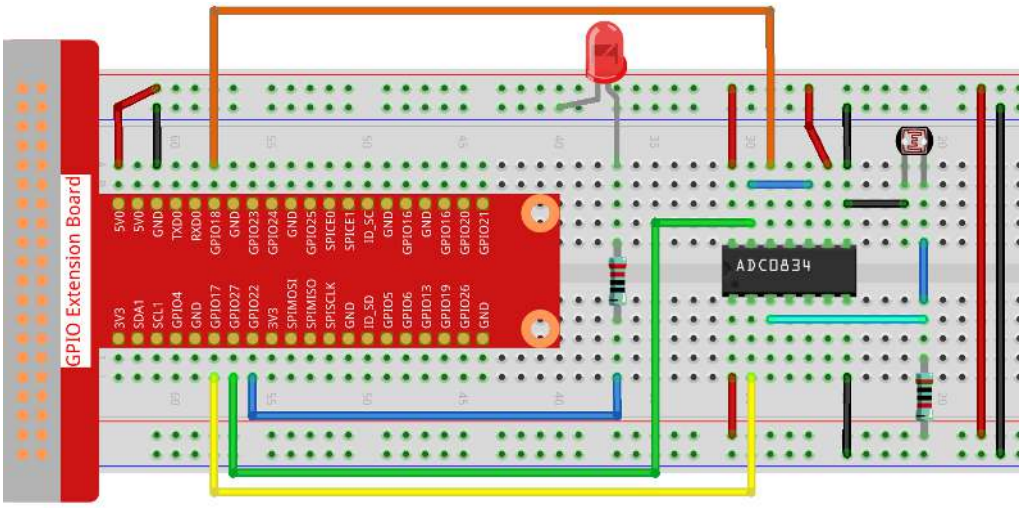
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.1/
```

Step 3: Compile the code.

```
gcc 2.2.1_Photoresistor.c -lwiringPi
```

Step 4: Run the executable file.

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

When the code is running, the brightness of the LED will change according to the light intensity sensed by the photoresistor.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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)
{
```

(continues on next page)

(continued from previous page)

```
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(LedPin,  0, 100);
```

(continues on next page)

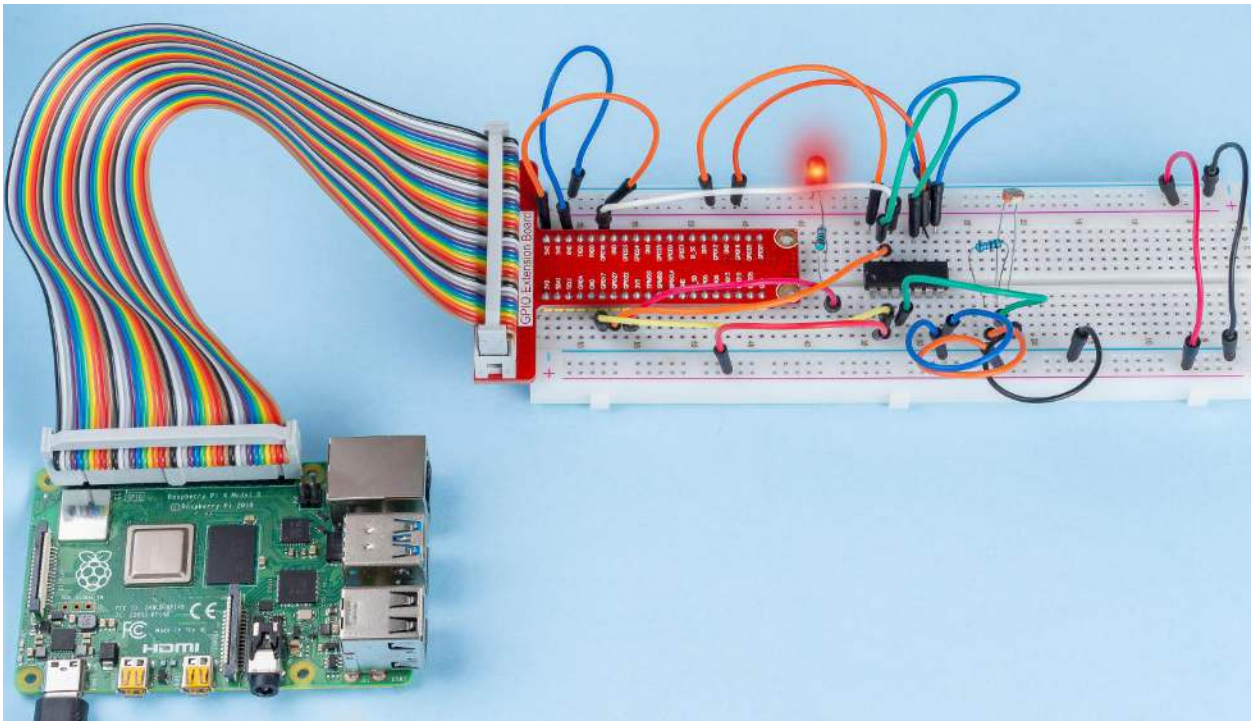
(continued from previous page)

```
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

Code Explanation

The codes here are the same as that in 2.1.4 Potentiometer. If you have any other questions, please check the code explanation of *2.1.7 Potentiometer* for details.

Phenomenon Picture

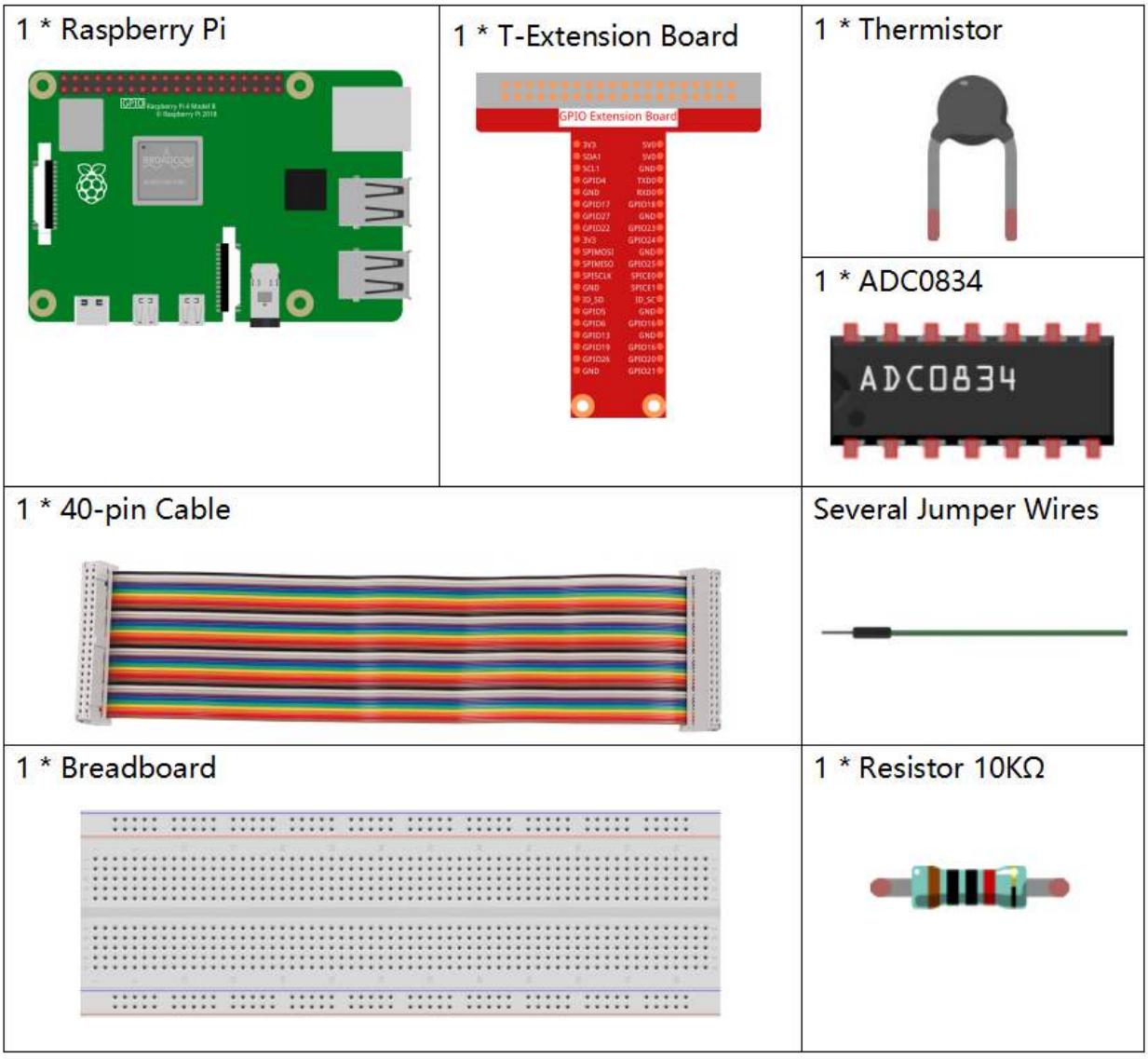


2.2.2 Thermistor

Introduction

Just like photoresistor can sense light, thermistor is a temperature sensitive electronic device that can be used for realizing functions of temperature control, such as making a heat alarm.

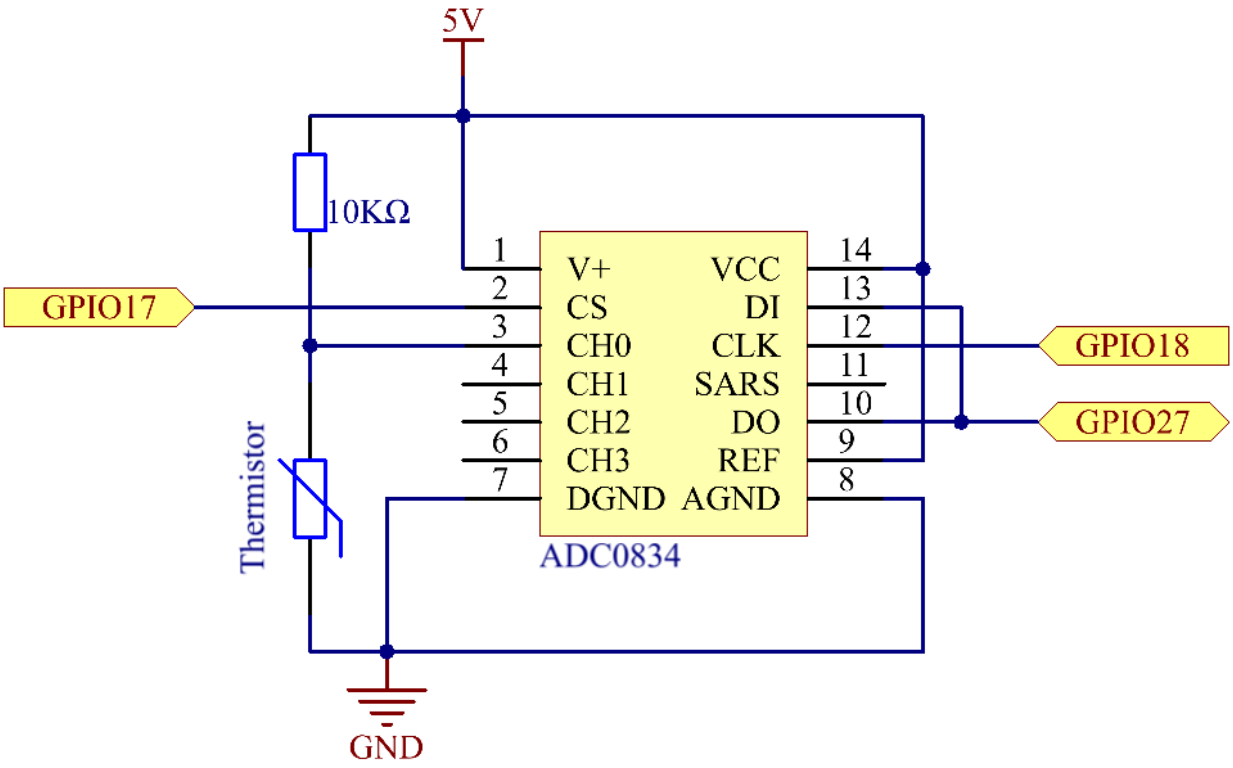
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Thermistor*
- *ADC0834*

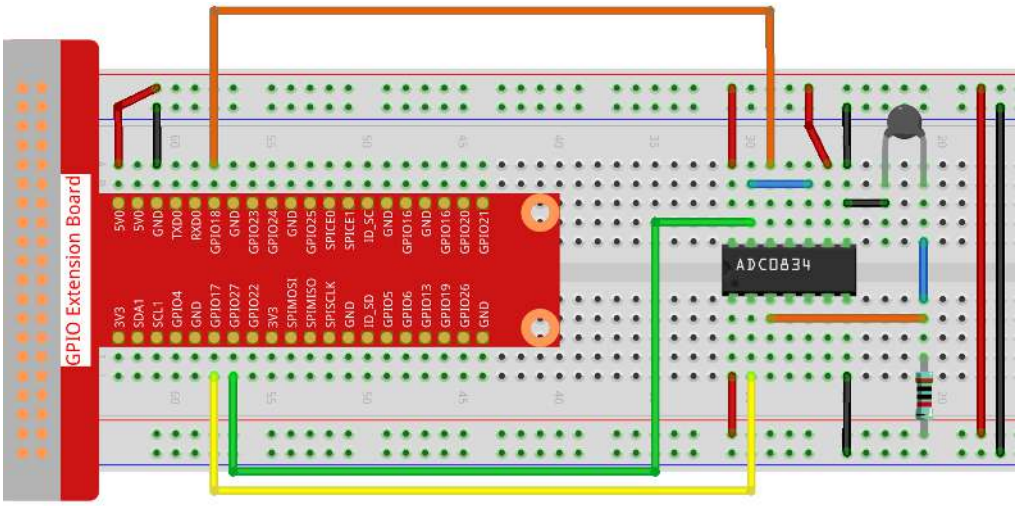
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.2/
```

Step 3: Compile the code.

```
gcc 2.2.2_Thermistor.c -lwiringPi -lm
```

Note: -lm is to load the library math. Do not omit, or you will make an error.

Step 4: Run the executable file.

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

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;

#define      ADC_CS    0
#define      ADC_CLK   1
#define      ADC_DIO   2
```

(continues on next page)

(continued from previous page)

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    unsigned char analogVal;
double Vr, Rt, temp, cel, Fah;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
```

(continues on next page)

(continued from previous page)

```
        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 Explanation

```
#include <math.h>
```

There is a C numerics library which declares a set of functions to compute common mathematical operations and transformations.

```
analogVal = get_ADC_Result(0);
```

This function is used to read the value of the thermistor.

```
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);
```

These calculations convert the thermistor values into Celsius values.

```
Vr = 5 * (double)(analogVal) / 255;
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
```

These two lines of codes are calculating the voltage distribution with the read value analog so as to get Rt (resistance of thermistor).

```
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
```

This code refers to plugging Rt into the formula **TK=1/(ln(RT/RN)/B+1/TN)** to get Kelvin temperature.

```
temp = temp - 273.15;
```

Convert Kelvin temperature into degree Celsius.

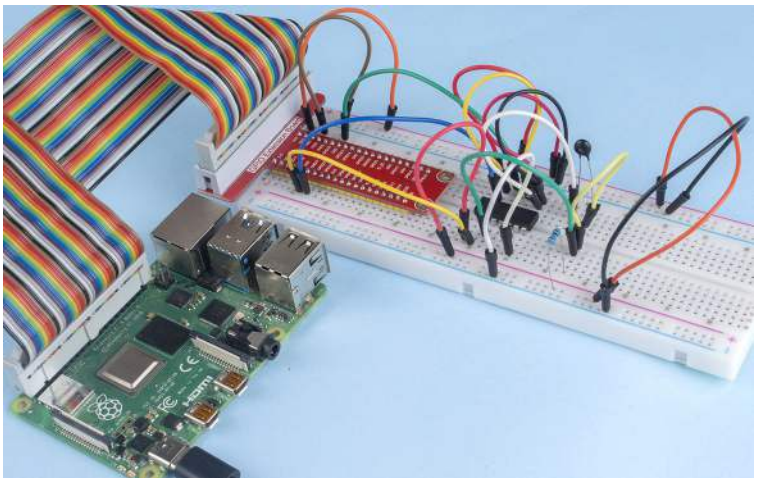
```
Fah = cel * 1.8 +32;
```

Convert degree Celsius into Fahrenheit.

```
printf("Celsius: %.2f C  Fahrenheit: %.2f F\n", cel, Fah);
```

Print centigrade degree, Fahrenheit degree and their units on the display.

Phenomenon Picture



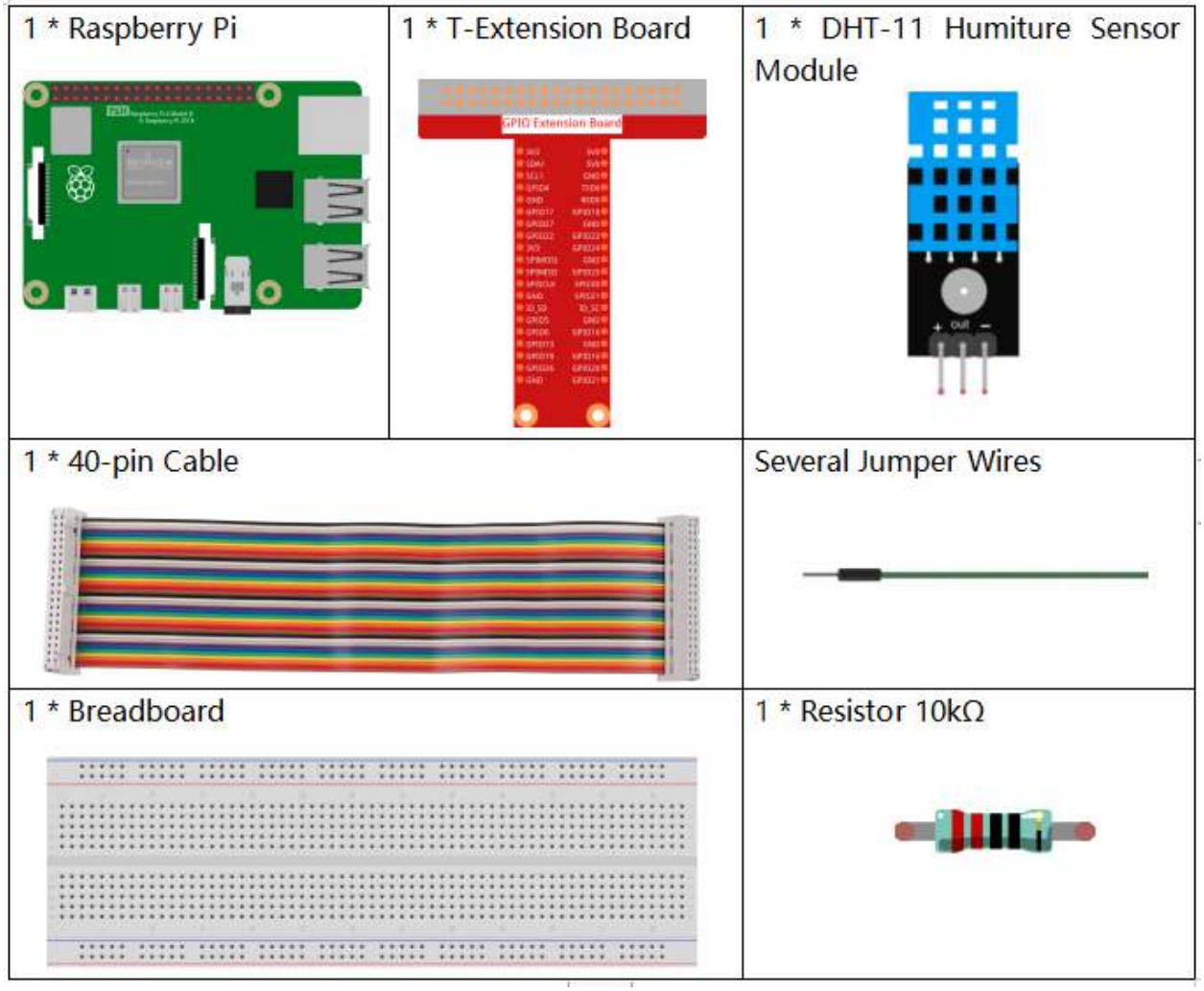
2.2.3 DHT-11

Introduction

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the technology of the temperature and humidity sensing are applied to ensure that the product has high reliability and excellent stability.

The sensors include a wet element resistive sensor and a NTC temperature sensor and they are connected to a high performance 8-bit microcontroller.

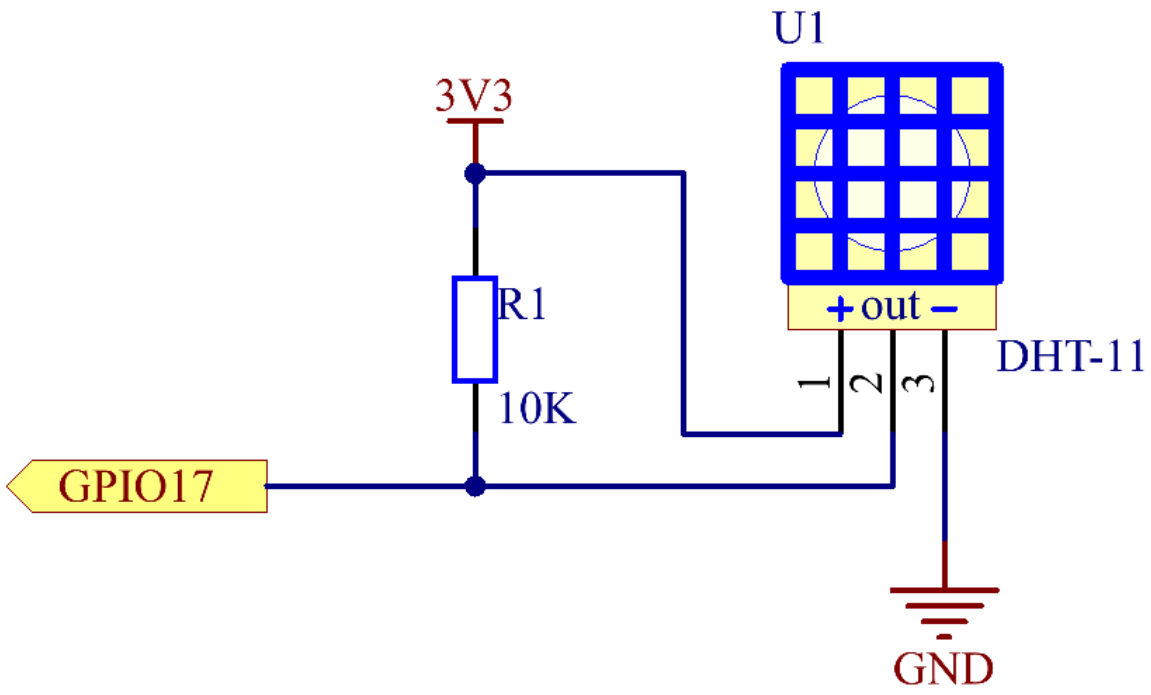
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Humiture Sensor Module*

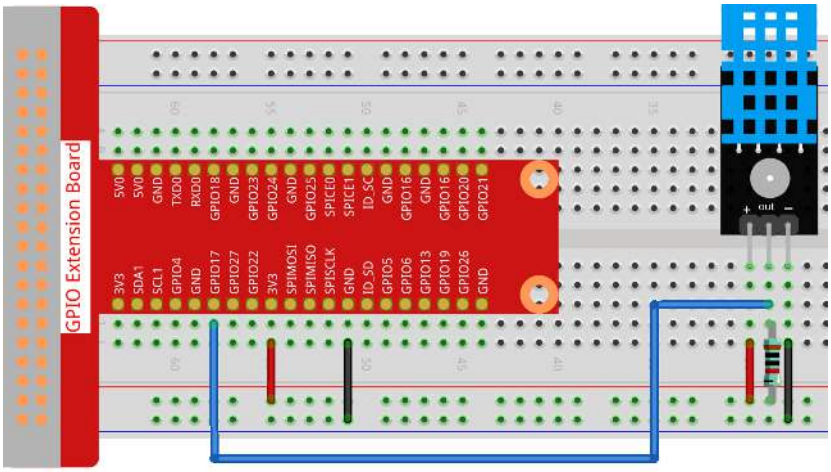
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.3/
```

Step 3: Compile the code.

```
gcc 2.2.3_DHT.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, the program will print the temperature and humidity detected by DHT11 on the computer screen.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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() {
    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
```

(continues on next page)

(continued from previous page)

```
    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;
    }
    while (1) {
        readDht11();
        delay(500); // wait 1sec to refresh
    }
    return 0 ;
}
```

Code Explanation

```
void readDht11() {
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit
```

(continues on next page)

(continued from previous page)

```
    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // ...
}
```

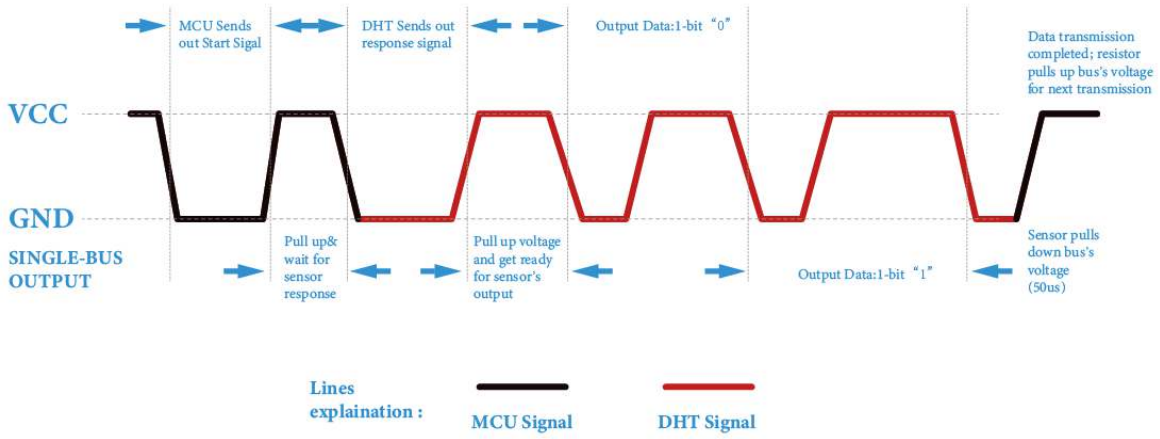
This function is used to realize the function of DHT11.

It generally can be divided into 3 parts:

1. prepare to read the pin:

```
// 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);
```

Its communication flow is determined by work timing.



When DHT11 starts up, MCU will send a low level signal and then keep the signal at high level for 40us. After that, the detection of the condition of external environment will start.

2. read data:

```
// 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
```

(continues on next page)

(continued from previous page)

```
        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++;
        }
    }
```

The loop stores the detected data in the dht11_dat[] array. DHT11 transmits data of 40 bits at a time. The first 16 bits are related to humidity, the middle 16 bits are related to temperature, and the last eight bits are used for verification. The data format is:

8bit humidity integer data + 8bit humidity decimal data + 8bit temperature integer data + 8bit temperature decimal data + 8bit check bit.

3. Print Humidity & Temperature.

```
// 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);
}
```

When the data storage is up to 40 bits, check the validity of the data through the **check bit (dht11_dat[4])**, and then print the temperature and humidity.

For example, if the received data is 00101011(8-bit value of humidity integer) 00000000 (8-bit value of humidity decimal) 00111100 (8-bit value of temperature integer) 00000000 (8-bit value of temperature decimal) 01100111 (check bit)

Calculation:

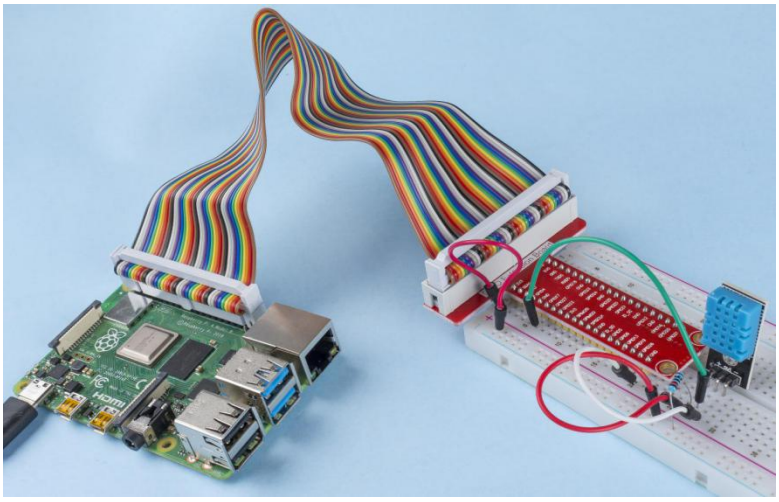
00101011+00000000+00111100+00000000=01100111.

The final result is equal to the check bit data, then the received data is correct:

Humidity =43%Temperature =60*C.

If it is not equal to the check bit data, the data transmission is not normal and the data is received again.

Phenomenon Picture

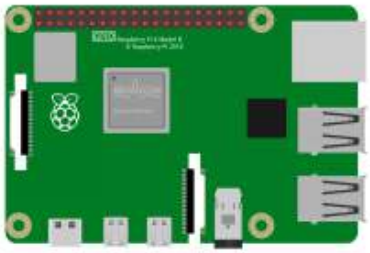
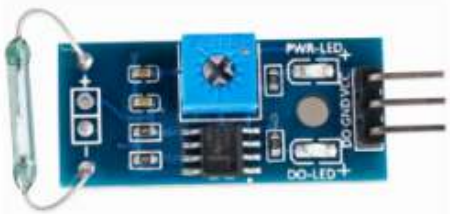
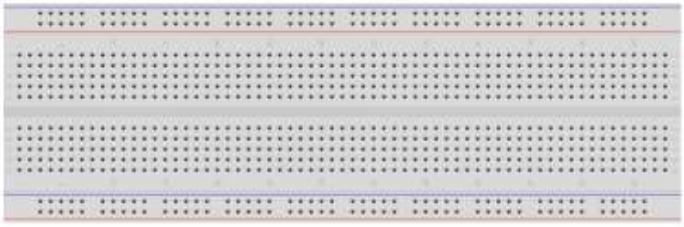
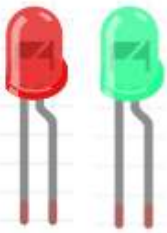
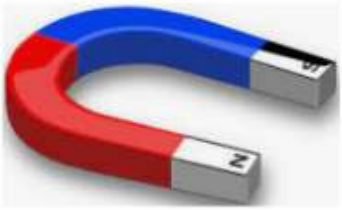


2.2.4 Reed Switch Module

Introduction

In this project, we will learn about the reed switch, which is an electrical switch that operates by means of an applied magnetic field.

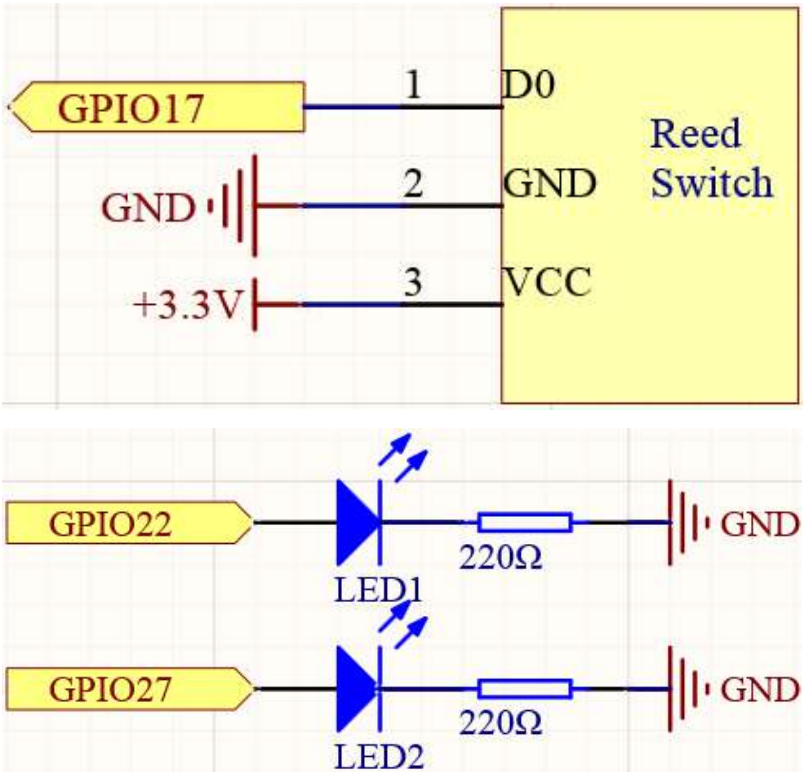
Components

1 * Raspberry Pi 	1 * T-Extension Board	1 * Reed Switch Module 
1 * 40-pin Cable		2 * Resistor(220Ω)
1 * Breadboard 		2 * LED 
1 * Magnet 		Several Jumper Wires

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Reed Switch Module*

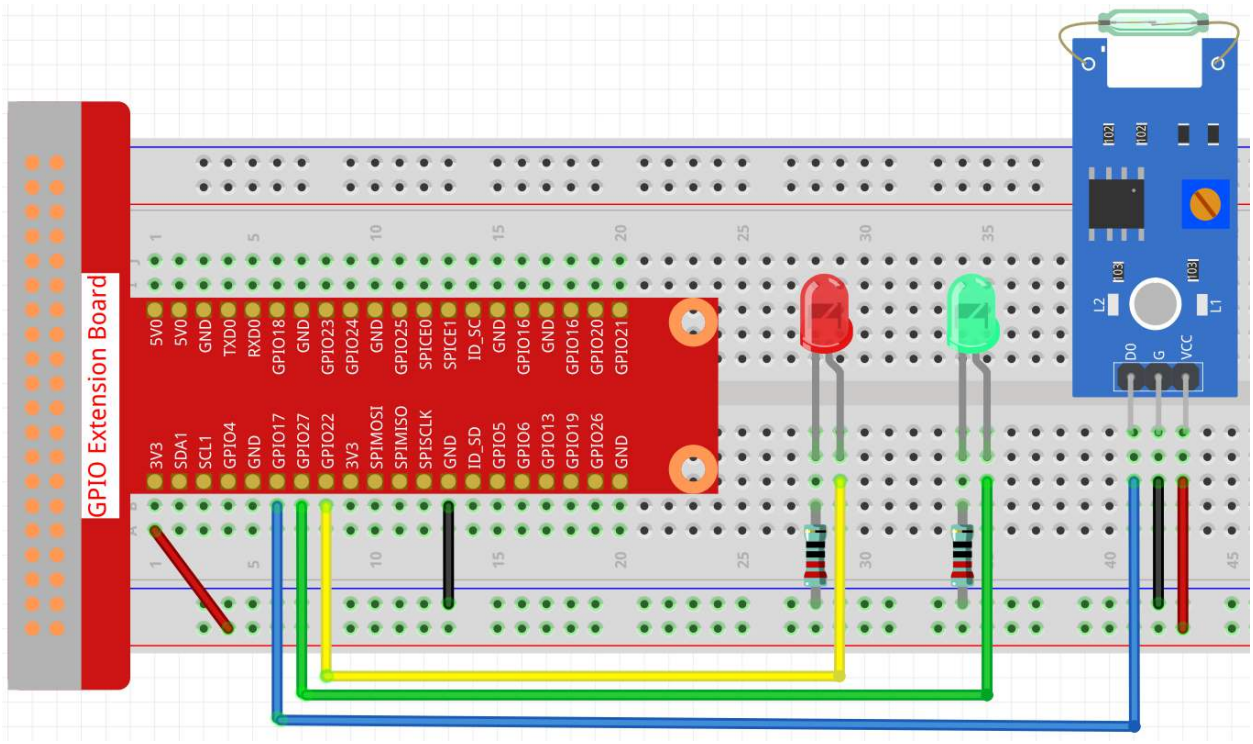
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/2.2.4/
```

Step 3: Compile.

```
gcc 2.2.4_ReedSwitch.c -lwiringPi
```

Step 4: Run.

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

The green LED will light up when the code is run. If a magnet is placed close to the reed switch module, the red LED lights up; take away the magnet and the green LED lights up again.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define ReedPin             0
#define Gpin                2
#define Rpin                3

void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
```

(continues on next page)

(continued from previous page)

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

    pinMode(ReedPin, INPUT);
    LED("GREEN");

    while(1){
        if(0 == digitalRead(ReedPin)){
            delay(10);
            if(0 == digitalRead(ReedPin)){
                LED("RED");
                printf("Detected Magnetic Material!\n");
            }
        }
        else if(1 == digitalRead(ReedPin)){
            delay(10);
            if(1 == digitalRead(ReedPin)){
                while(!digitalRead(ReedPin));
                LED("GREEN");
            }
        }
    }
    return 0;
}
```

Code Explanation

```
#define ReedPin             0
#define Gpin                2
#define Rpin                3
```

Pin GPIO17, GPIO27 and GPIO22 of the T_Extension Board is corresponding to the GPIO0, GPIO2 and GPIO3 in wiringPi. Assign GPIO0, GPIO2 and GPIO3 to `ReedPin`, `Gpin` and `Rpin`.

```
void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
```

(continues on next page)

(continued from previous page)

```
    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");
}
```

Set a `LED()` function to control the 2 LEDs, the parameter of this function is `color`.

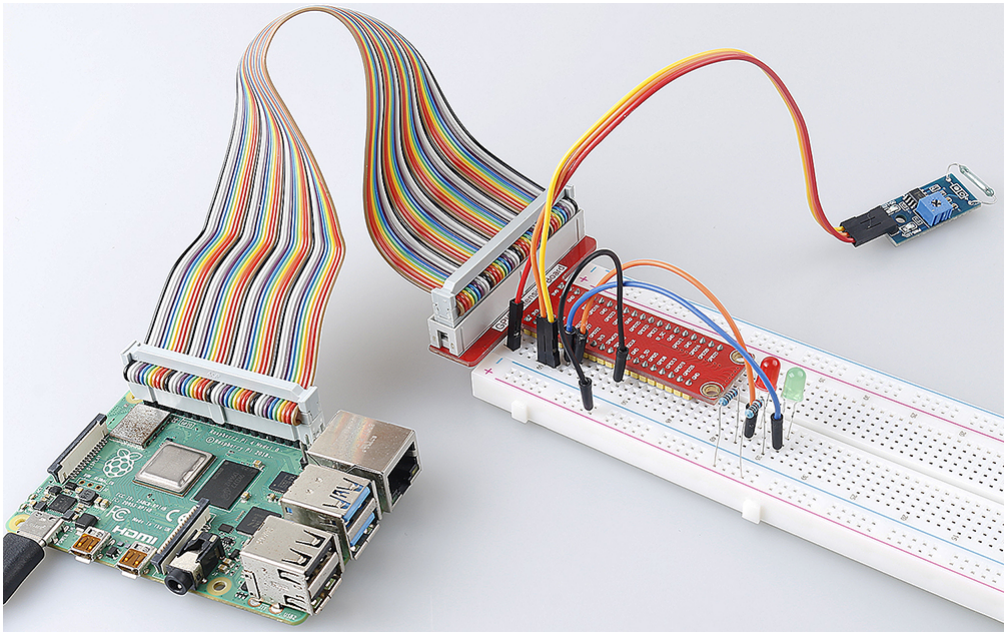
When `color` is `"RED"`, set `Rpin` to `HIGH` (light up the red LED) and `Gpin` to `LOW` (turn off the green LED); when `color` is `"GREEN"`, then light up the green LED and turn off the red LED.

```
while(1){
        if(0 == digitalRead(ReedPin)){
            delay(10);
            if(0 == digitalRead(ReedPin)){
                LED("RED");
                printf("Detected Magnetic Material!\n");
            }
        }
        else if(1 == digitalRead(ReedPin)){
            delay(10);
            if(1 == digitalRead(ReedPin)){
                while(!digitalRead(ReedPin));
                LED("GREEN");
            }
        }
    }
```

Read the value of the reed switch module, if the value read 2 times is 0, call `LED("RED")` to light up the red LED and print `"Magnetic material detected!"`.

If the value is 1, the green LED is lit.

Phenomenon Picture

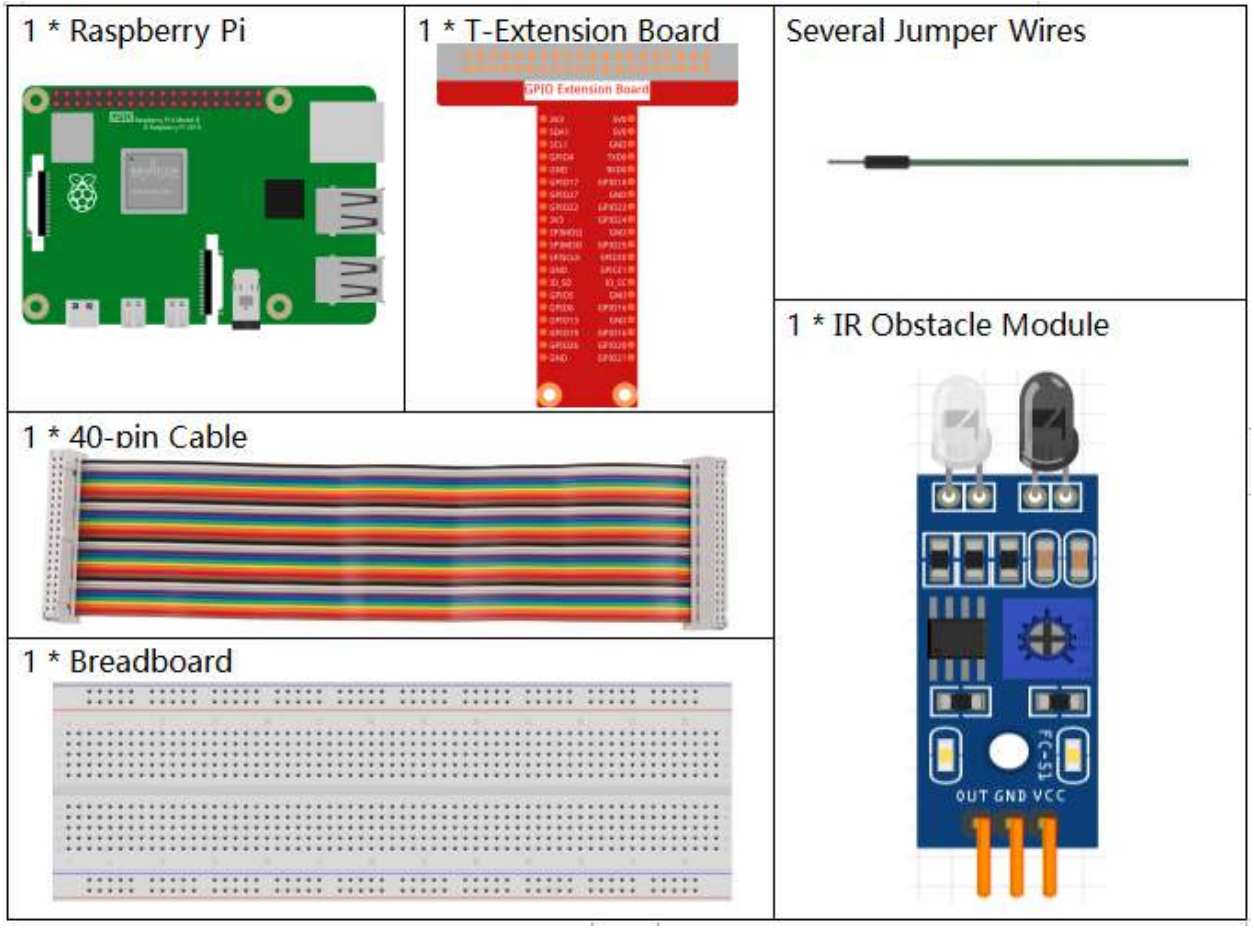


2.2.5 IR Obstacle Avoidance Module

Introduction

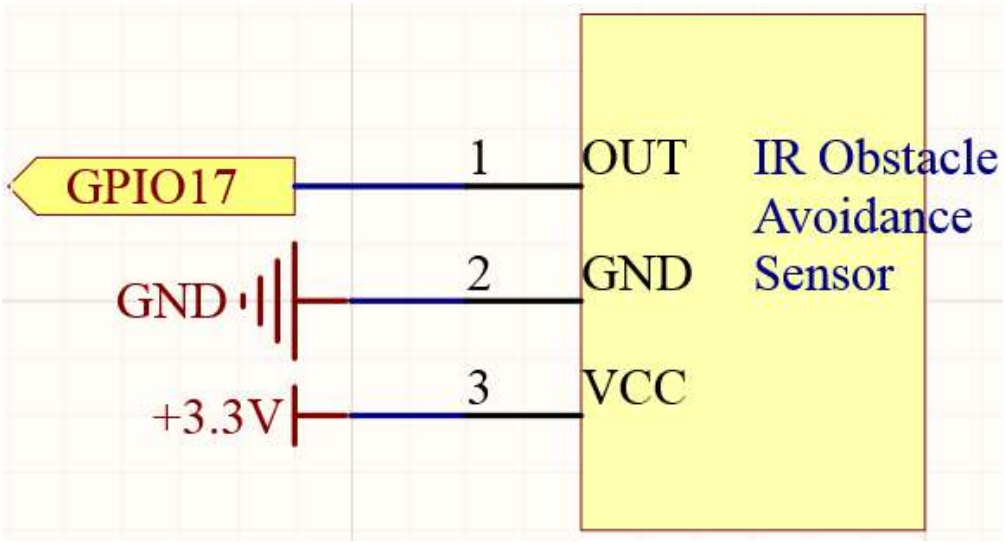
In this project, we will learn IR obstacle avoidance module, which is a sensor module that can be used to detect obstacles at short distances, with small interference, easy to assemble, easy to use, etc. It can be widely used in robot obstacle avoidance, obstacle avoidance trolley, assembly line counting, etc.

Components



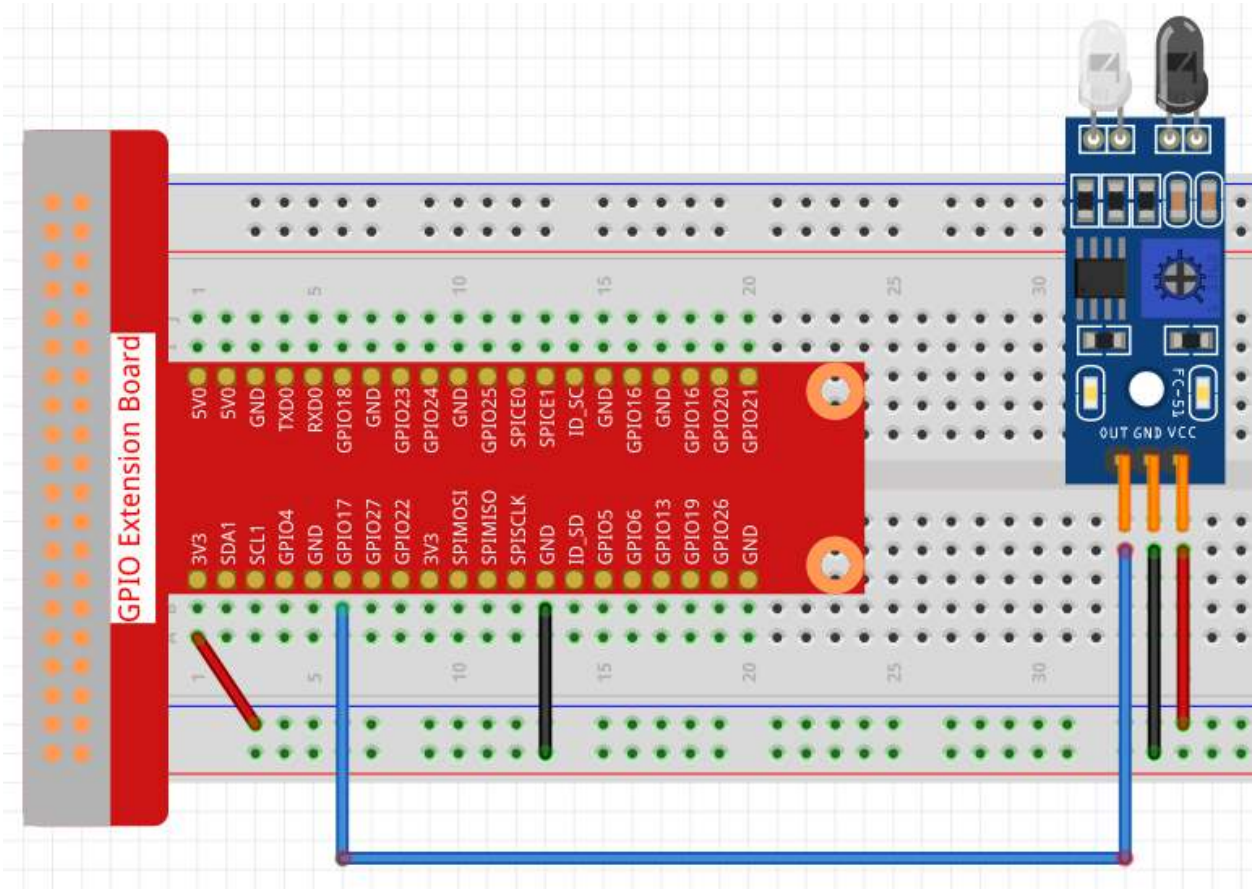
- *GPIO Extension Board*
- *Breadboard*
- *Obstacle Avoidance Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/2.2.5/
```

Step 3: Compile.

```
gcc 2.2.5_IrObstacle.c -lwiringPi
```

Step 4: Run.

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

After the code runs, when you put your hand in front of the module's probe, the output indicator on the module lights up and the "Detected Barrier!" will be repeatedly printed on the screen.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define ObstaclePin      0

void myISR(void)
{
   printf("Detected Barrier !\n");
}

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

   if(wiringPiISR(ObstaclePin, INT_EDGE_FALLING, &myISR) < 0){
      printf("Unable to setup ISR !!!\n");
      return 1;
   }

   while(1){
      ;
   }

   return 0;
}
```

Code Explanation

```
void myISR(void)
{
   printf("Detected Barrier !\n");
}
```

Define a function `myISR()` to print `obstacle detected` , indicating that an obstacle is detected.

```
if(wiringPiISR(ObstaclePin, INT_EDGE_FALLING, &myISR) < 0){
    printf("Unable to setup ISR !!!\n");
    return 1;
}
```

This `wiringPiISR()` function registers a `myISR()` function to received interrupts on the specified `ObstaclePin`.

When ObstaclePin changes from high to low, it means that an obstacle is detected. At this time, call the `myISR()` function to print "Detected Barrier !"

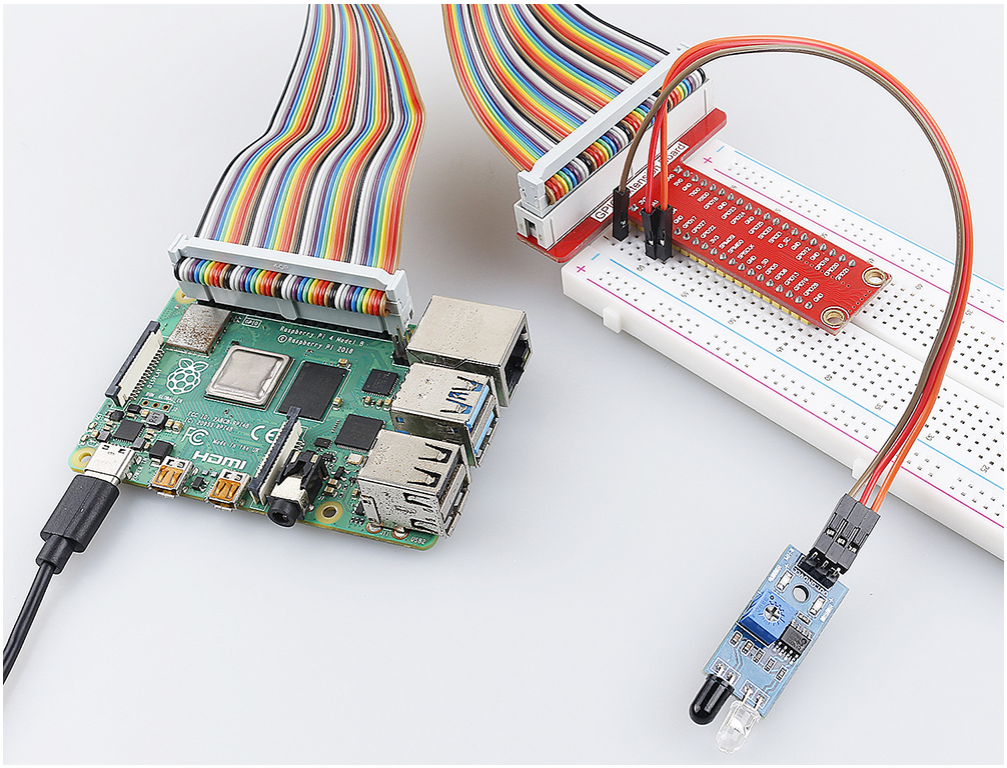
The prototype of this `wiringPiISR()` function is shown below.

```
int wiringPiISR (int pin, int edgeType,  void (*function)(void)) ;
```

The edgeType parameter is either `INT_EDGE_FALLING` , `INT_EDGE_RISING` , `INT_EDGE_BOTH` or `INT_EDGE_SETUP` . If it is `INT_EDGE_SETUP` then no initialisation of the pin will happen – it's assumed that you have already setup the pin elsewhere (e.g. with the gpio program), but if you specify one of the other types, then the pin will be exported and initialised as specified.

For more information, please refer to: wiringPi-Functions (API).

Phenomenon Picture

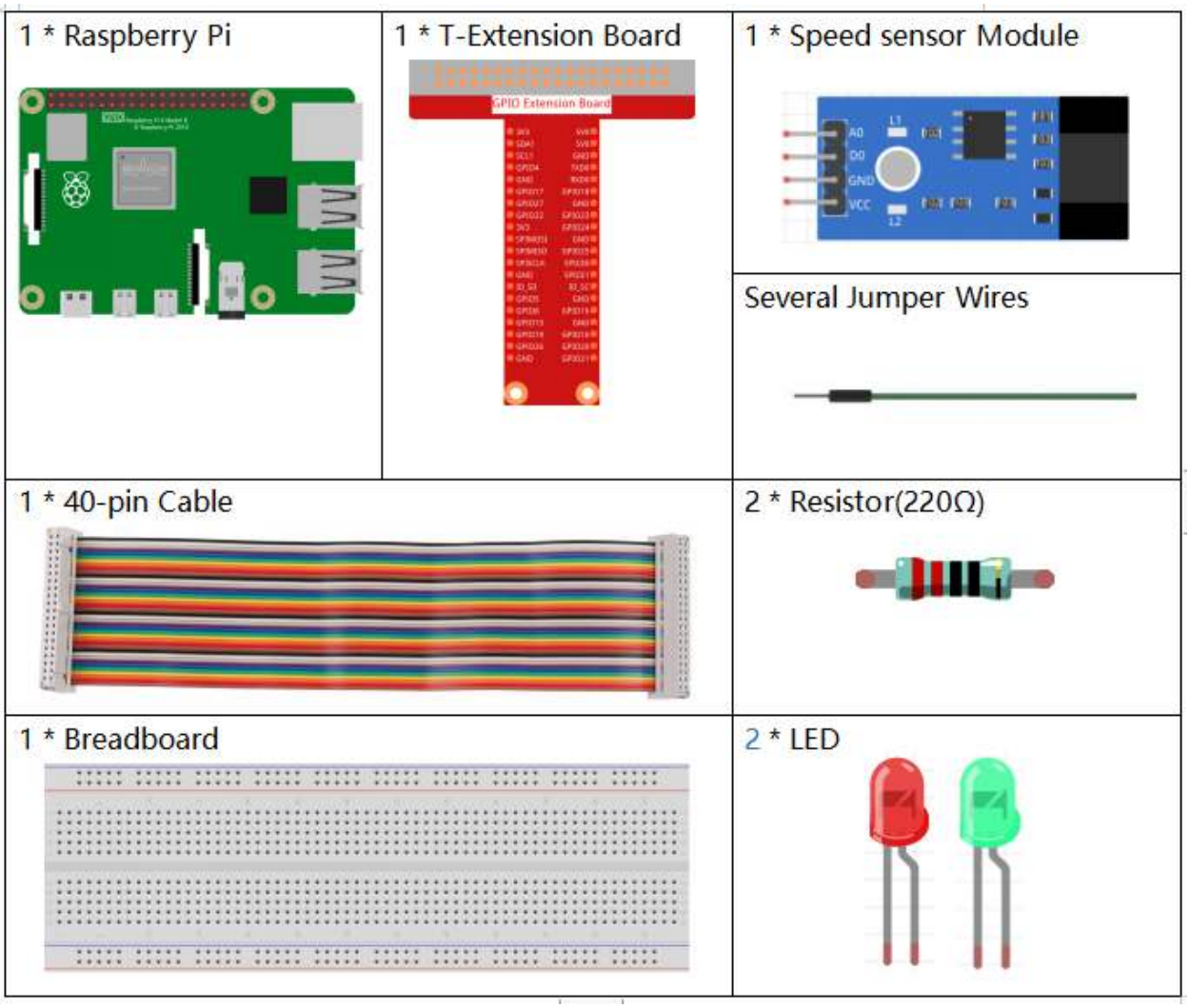


2.2.6 Speed Sensor Module

Introduction

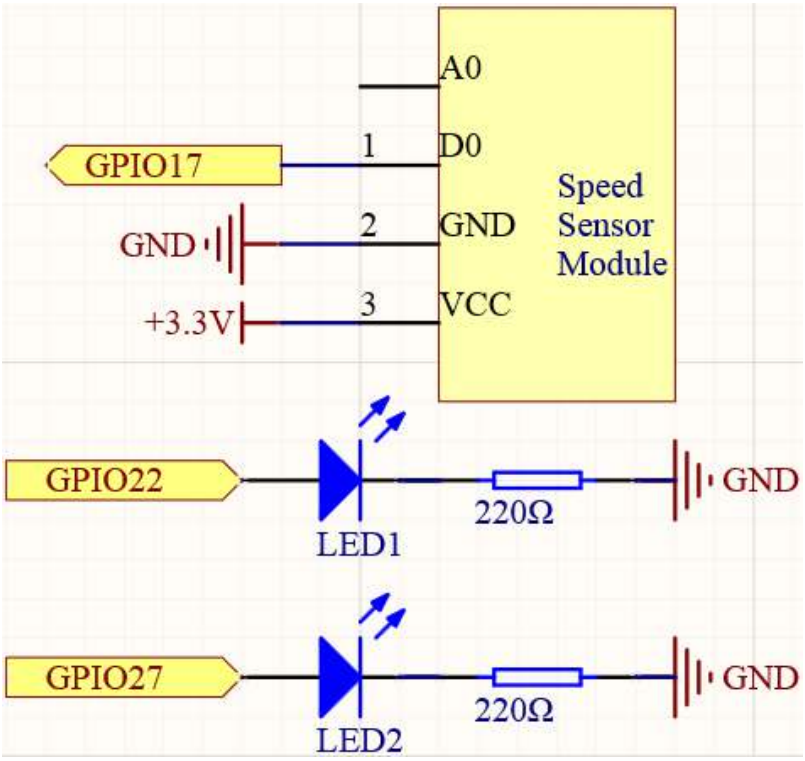
In this project, we will learn the use of the speed sensor module. A Speed Sensor Module is a type of tachometer that is used to measure the speed of a rotating object like a motor.

Components



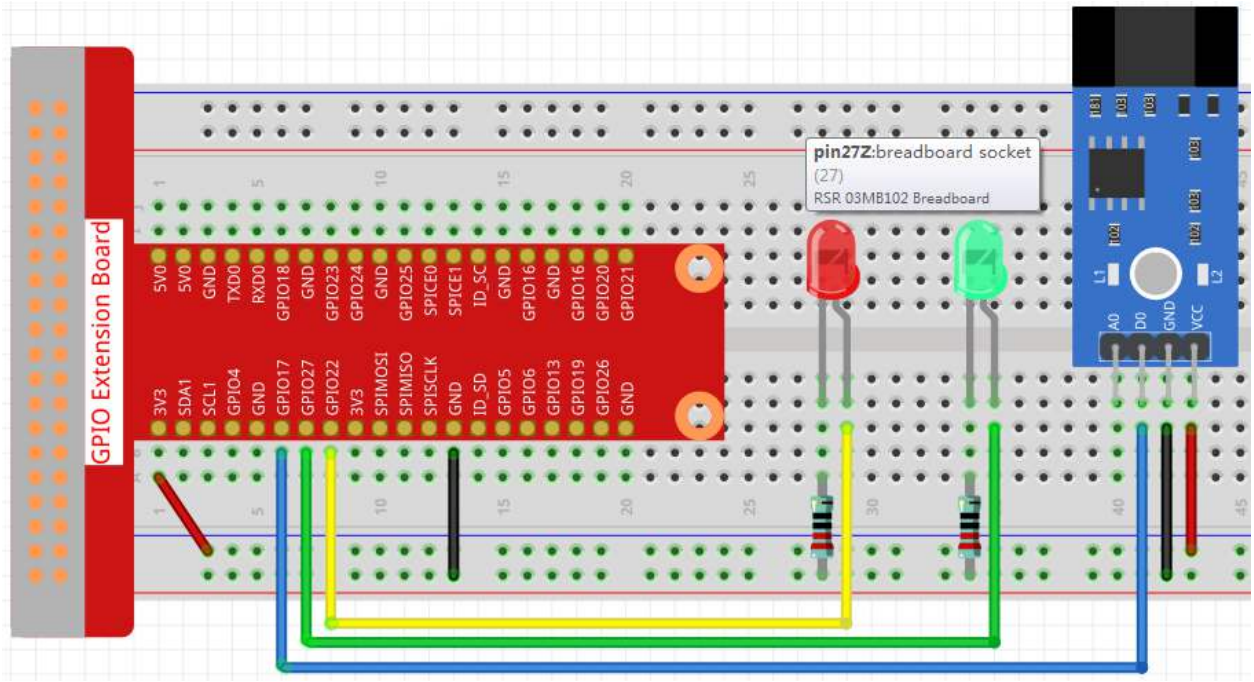
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Speed Sensor Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/2.2.6/
```

Step 3: Compile.

```
gcc 2.2.6_speed_sensor_module.c -lwiringPi
```

Step 4: Run.

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

After the code runs, the green LED will light up. If you place an obstacle in the gap of the speed sensor module, the "light blocked" will be printed on the screen and the red LED will be lit. Remove the obstacle and the green LED will light up again.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define speedPin            0
#define Gpin                2
#define Rpin                3

void LED(int color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == 0){
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == 1){
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
}

void Print(int x){
    if ( x == 0 ){
        printf("Light was blocked\n");
    }
}

int main(void){

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

    pinMode(speedPin, INPUT);
```

(continues on next page)

(continued from previous page)

```
    int temp;
    while(1){
        //Reverse the input of speedPin
        if ( digitalRead(speedPin) == 0 ){
            temp = 1;
        }
        if ( digitalRead(speedPin) == 1 ){
            temp = 0;
        }

        LED(temp);
        Print(temp);
    }
    return 0;
}
```

Code Explanation

```
void LED(int color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == 0){
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == 1){
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
}
```

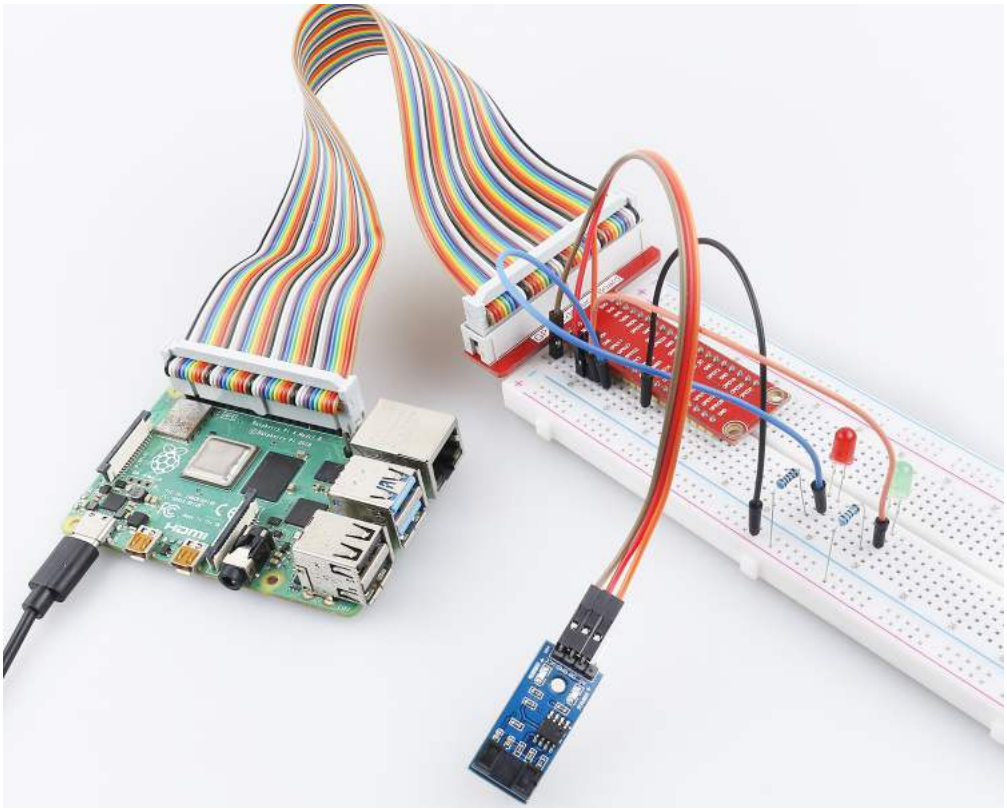
Set a `LED()` function to control the 2 LEDs, the parameter of this function is `color`.

When `color` is 0, set `Rpin` to `HIGH` (light up the red LED) and `Gpin` to `LOW` (turn off the green LED); when `color` is 1, then light up the green LED and turn off the red LED.

```
while(1){
    //Reverse the input of speedPin
    if ( digitalRead(speedPin) == 0 ){
        temp = 1;
    }
    if ( digitalRead(speedPin) == 1 ){
        temp = 0;
    }

    LED(temp);
    Print(temp);
}
```

When you place an obstacle in the gap of the speed sensor module, `speedPin` is low level (0), then call `LED(1)` function to light up the green LED and "Light was blocked!" is printed.

Phenomenon Picture

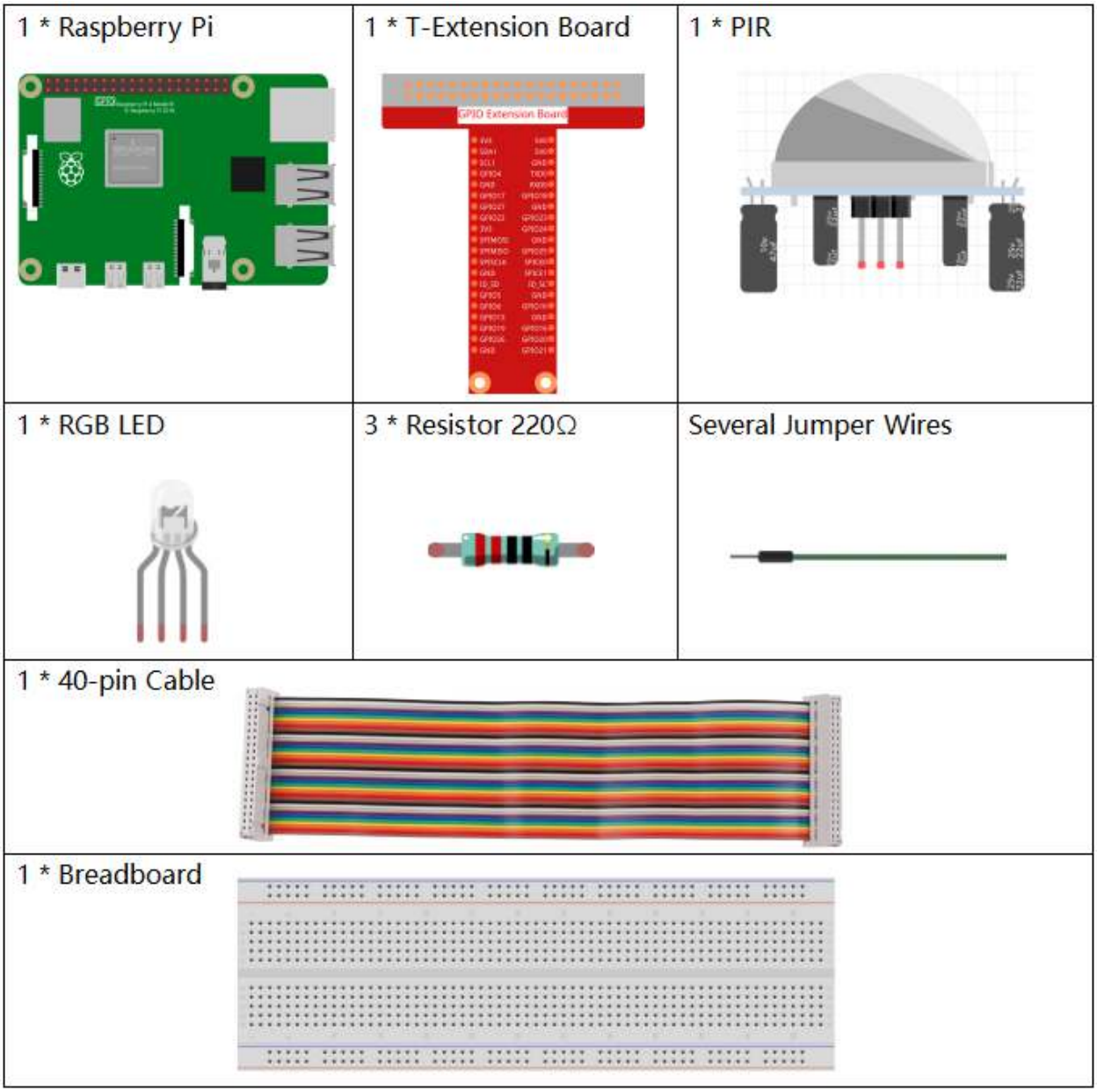


2.2.7 PIR

Introduction

In this project, we will make a device by using the human body infrared pyroelectric sensors. When someone gets closer to the LED, the LED will turn on automatically. If not, the light will turn off. This infrared motion sensor is a kind of sensor that can detect the infrared emitted by human and animals.

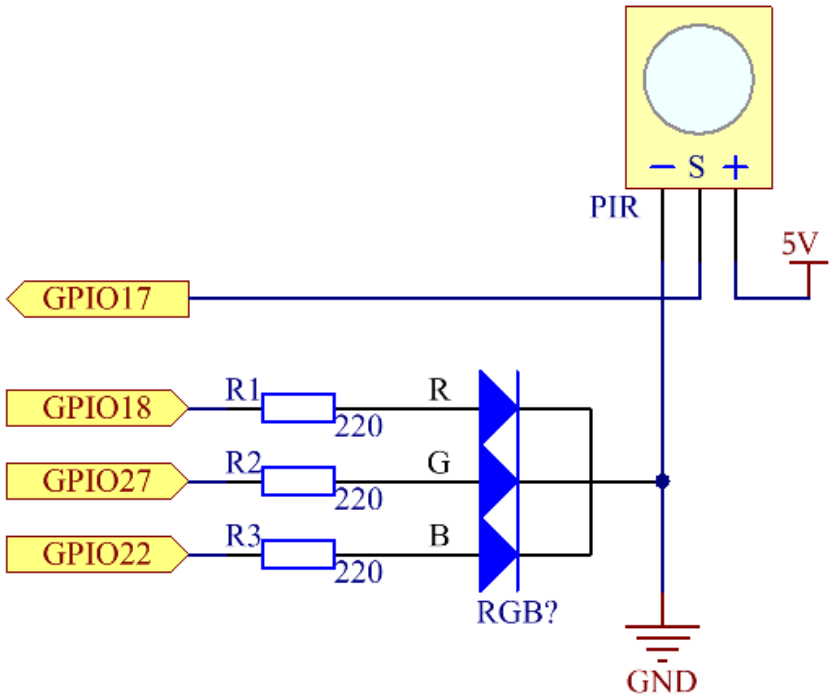
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*
- *PIR Motion Sensor Module*

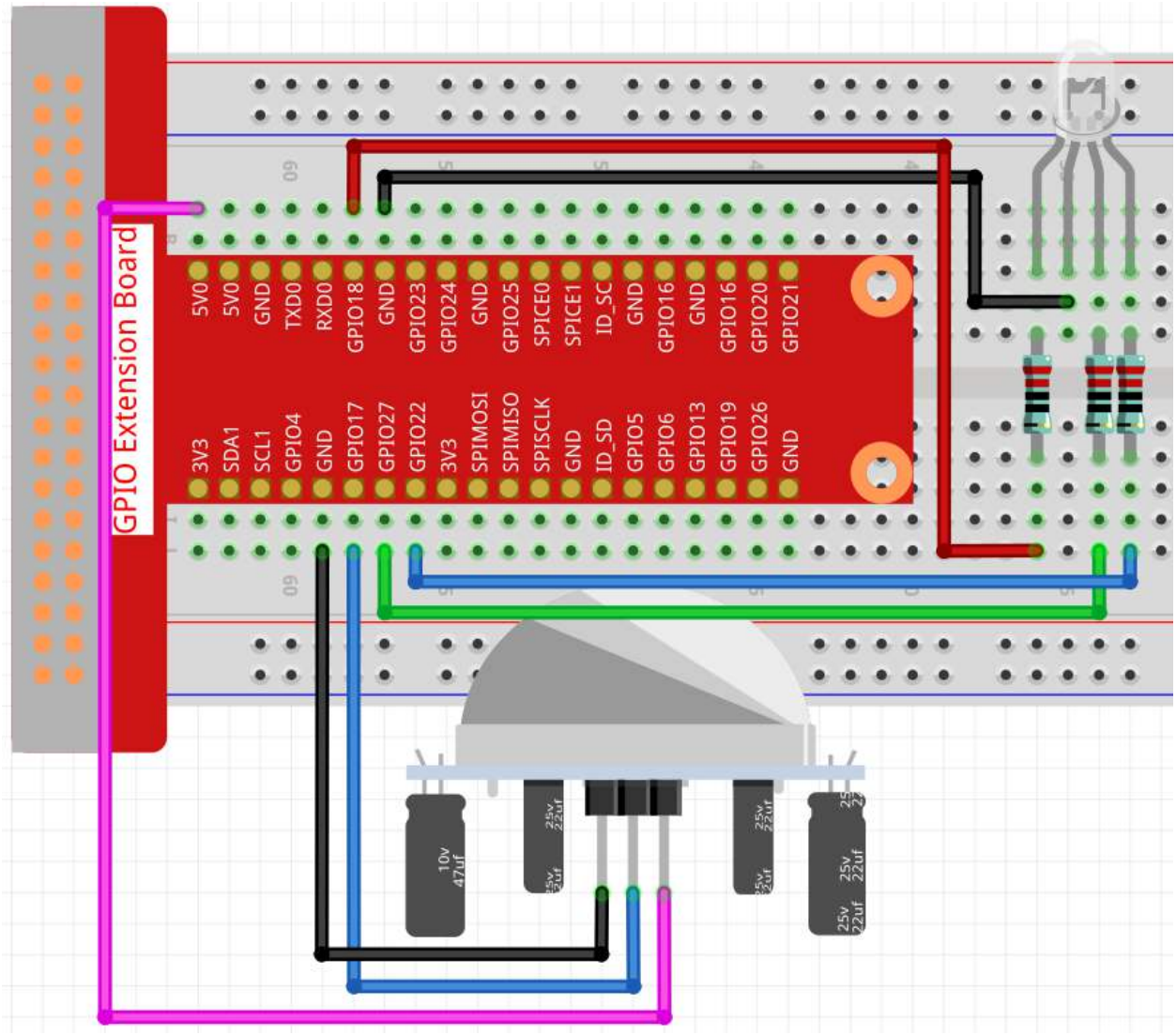
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.7/
```

Step 3: Compile the code.

```
gcc 2.2.7_PIR.c -lwiringPi
```

Step 4: Run the executable file.

```
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.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
    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){
```

(continues on next page)

(continued from previous page)

```
    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 Explanation

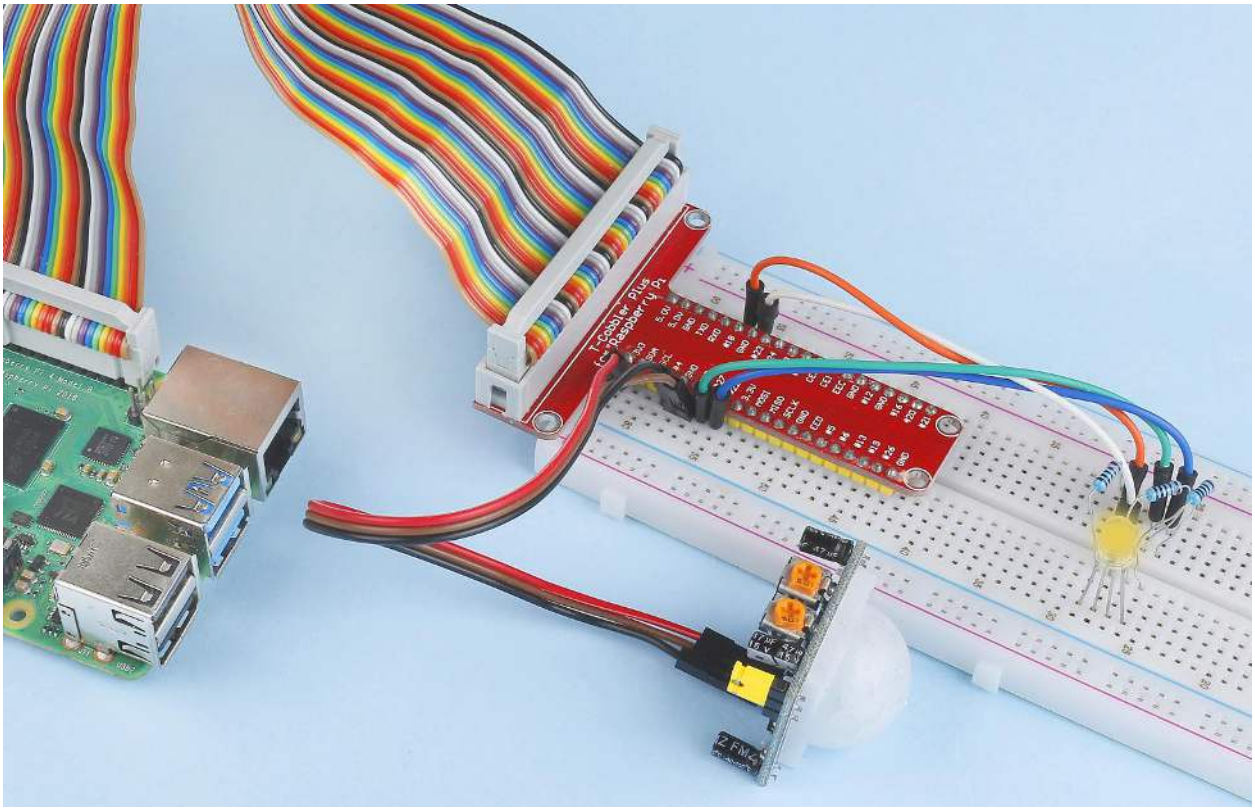
```
void ledInit(void);
void ledColorSet(uchar r_val, uchar g_val, uchar b_val);
```

These codes are used to set the color of the RGB LED, and please refer to *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;
}
```

When PIR detects the human infrared spectrum, RGB LED emits the yellow light; if not, emits the blue light.

Phenomenon Picture

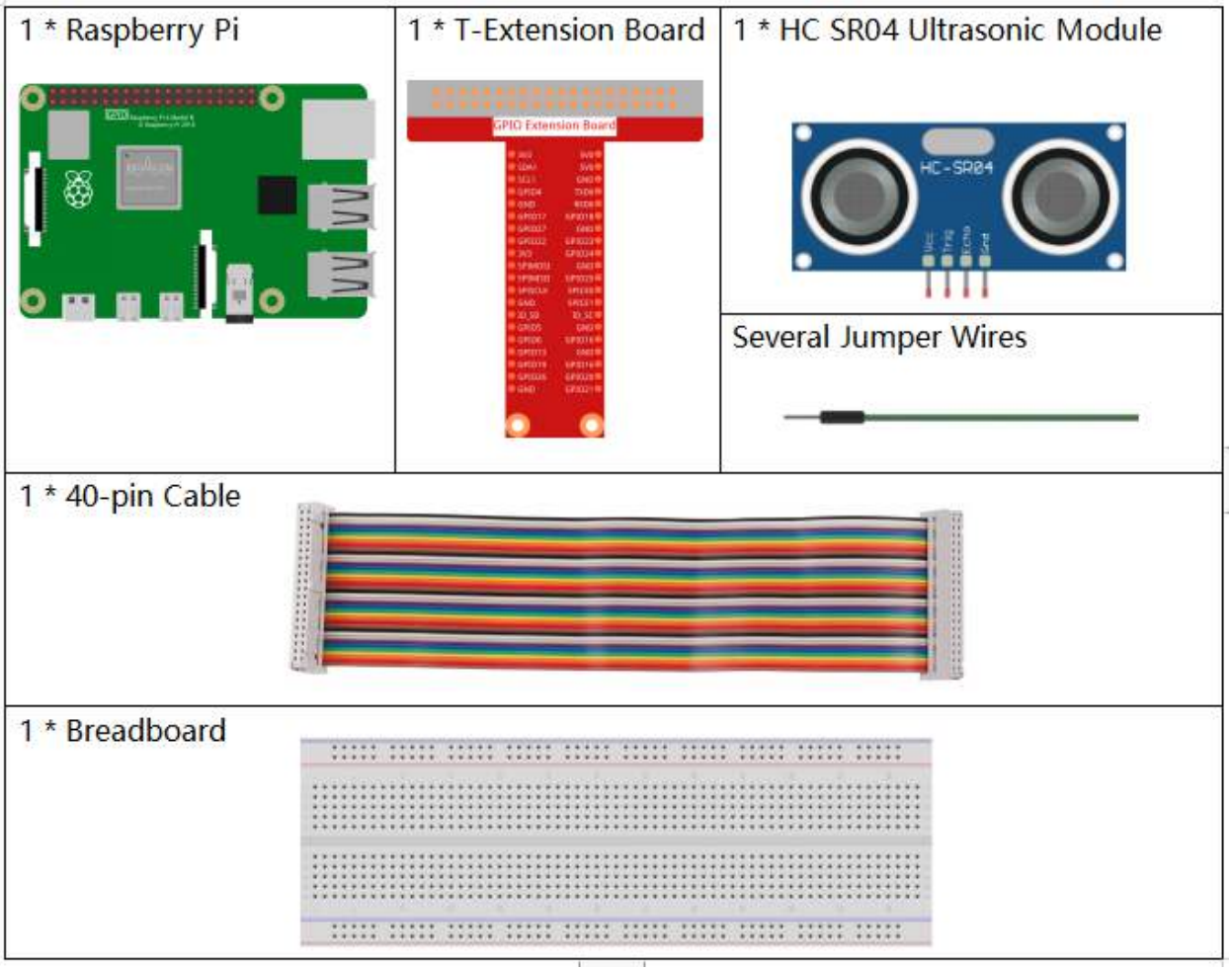


2.2.8 Ultrasonic Sensor Module

Introduction

The ultrasonic sensor uses ultrasonic to accurately detect objects and measure distances. It sends out ultrasonic waves and converts them into electronic signals.

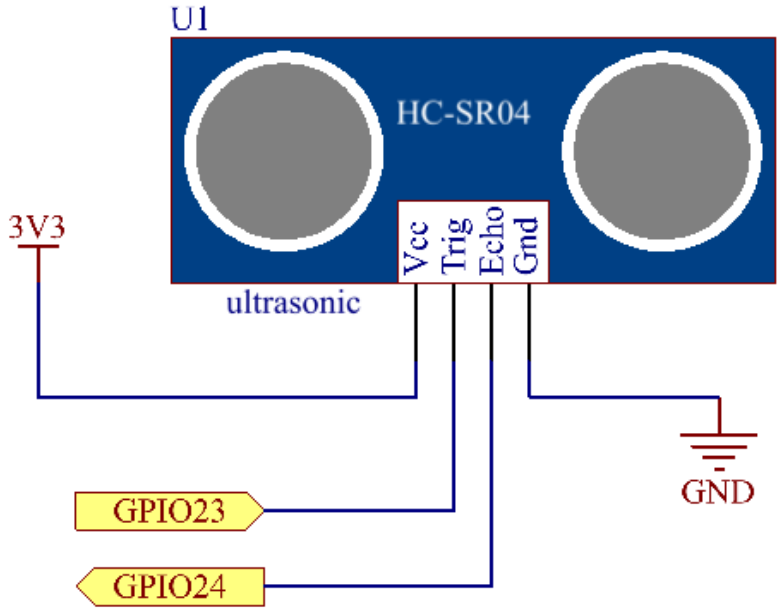
Components



- *GPIO Extension Board*
- *Breadboard*
- *Ultrasonic Module*

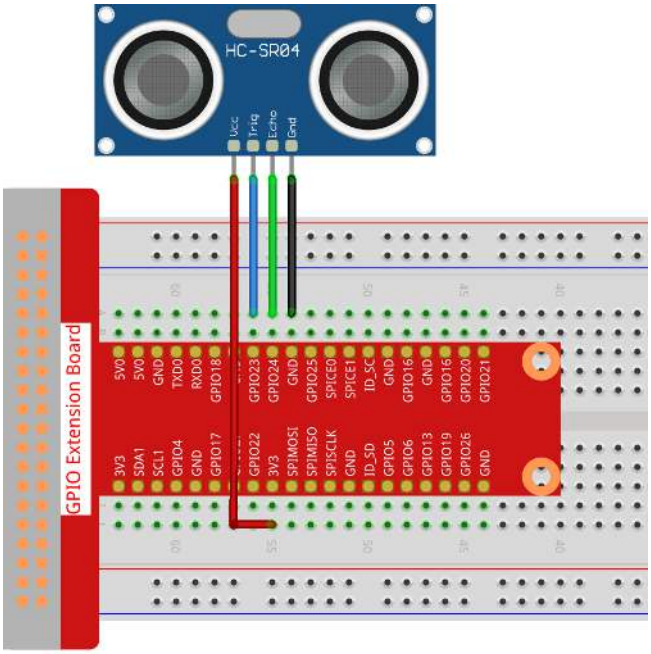
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.8/
```

Step 3: Compile the code.

```
gcc 2.2.8_Ultrasonic.c -lwiringPi
```

Step 4: Run the executable file.

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

With the code run, the ultrasonic sensor module detects the distance between the obstacle ahead and the module itself, then the distance value will be printed on the screen.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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);
}
```

(continues on next page)

(continued from previous page)

```
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 Explanation

```
void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
```

Initialize the ultrasonic pin; meanwhile, set Echo to input, Trig to output.

```
float disMeasure(void){};
```

This function is used to realize the function of ultrasonic sensor by calculating the return detection distance.

```
struct timeval tv1;
struct timeval tv2;
```

Struct timeval is a structure used to store the current time. The complete structure is as follows:

```
struct timeval
{
__time_t tv_sec;         /* Seconds. */
__suseconds_t tv_usec;   /* Microseconds. */
};
```

Here, tv_sec represents the seconds that Epoch spent when creating struct timeval. Tv_usec stands for microseconds or a fraction of seconds.

```
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
```

A 10us ultrasonic pulse is being sent out.

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);
```

This empty loop is used to ensure that when the trigger signal is sent, there is no interfering echo signal and then get the current time.

```
while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);
```

This empty loop is used to ensure that the next step is not performed until the echo signal is received and then get the current time.

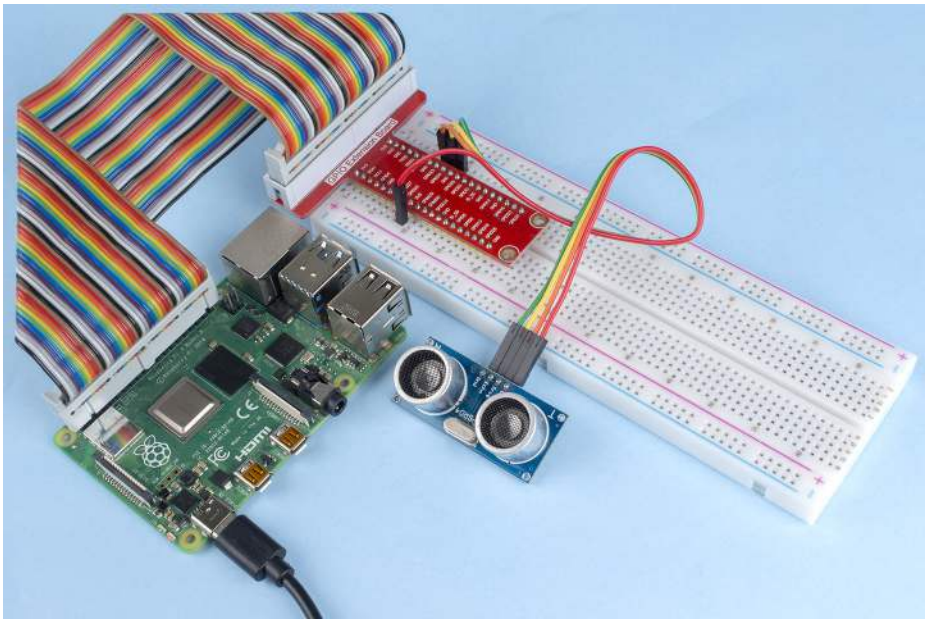
```
time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
time2  = tv2.tv_sec * 1000000 + tv2.tv_usec;
```

Convert the time stored by struct timeval into a full microsecond time.

```
dis = (float)(time2 - time1) / 1000000 * 34000 / 2;
```

The distance is calculated by the time interval and the speed of sound propagation. The speed of sound in the air: 34000cm/s.

Phenomenon Picture



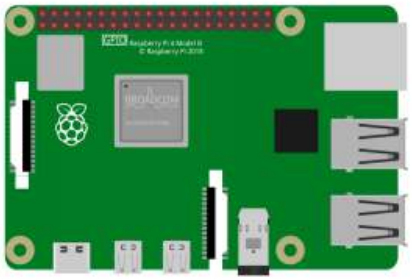
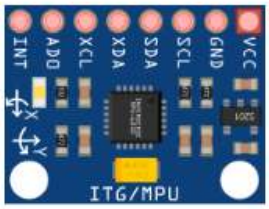
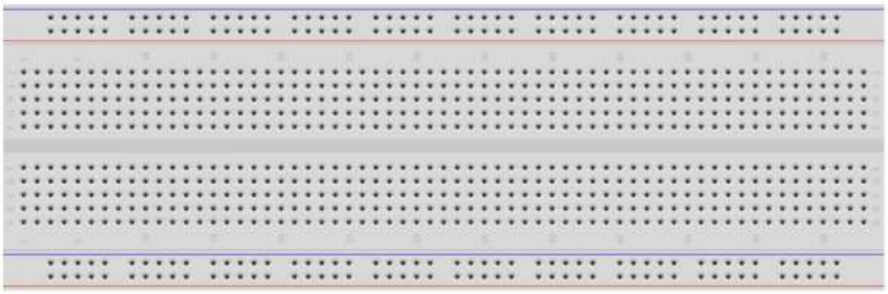
2.2.9 MPU6050 Module

Introduction

The MPU-6050 is the world's first and only 6-axis motion tracking devices (3-axis Gyroscope and 3-axis Accelerometer) designed for smartphones, tablets and wearable sensors that have these features, including the low power, low cost, and high performance requirements.

In this experiment, use I2C to obtain the values of the three-axis acceleration sensor and three-axis gyroscope for MPU6050 and display them on the screen.

Components

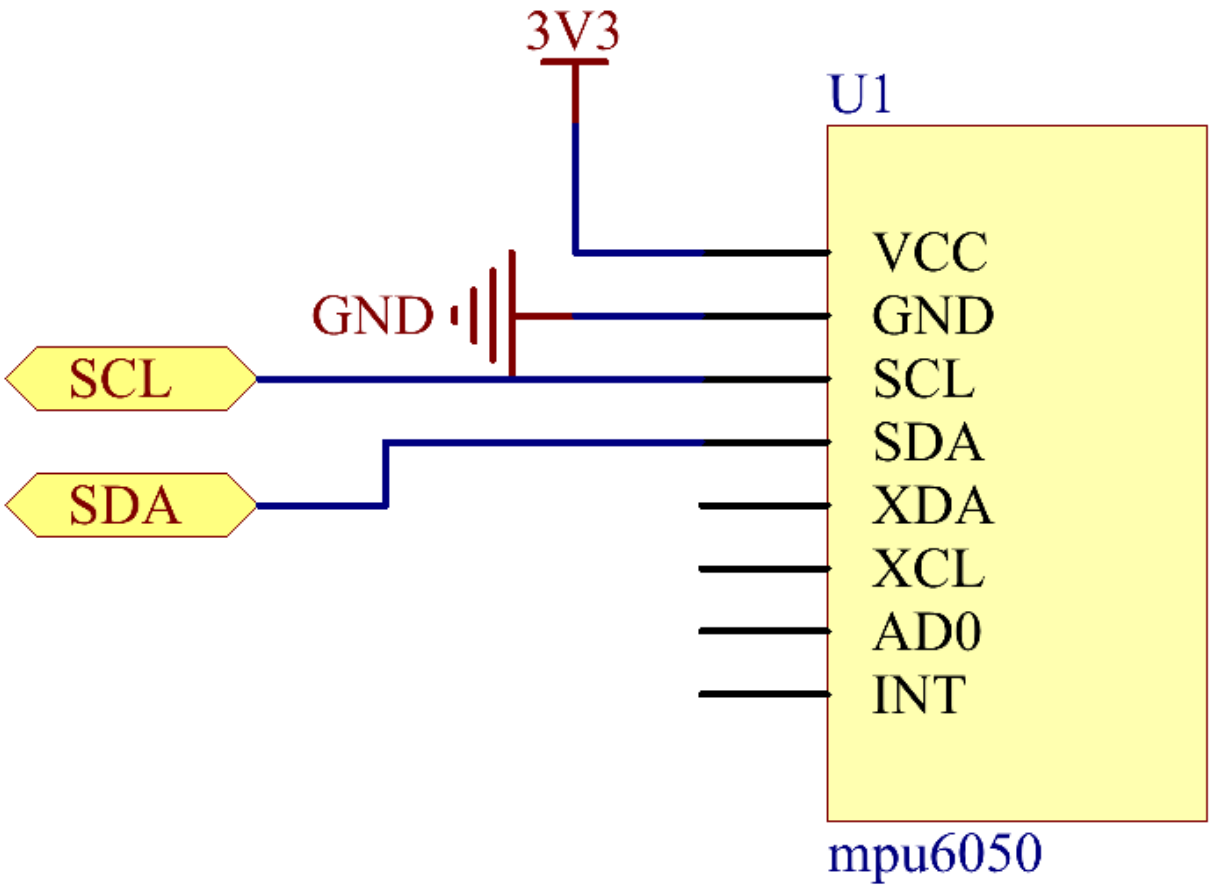
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
</td><td>1 * MPU6050
 
</td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *MPU6050 Module*

Schematic Diagram

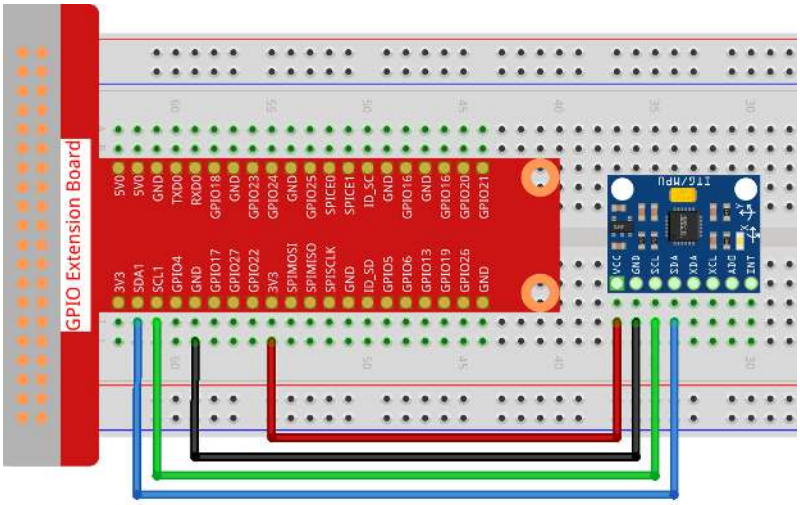
MPU6050 communicates with the microcontroller through the I2C bus interface. The SDA1 and SCL1 need to be connected to the corresponding pin.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see Appendix *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.9/
```

Step 4: Compile the code.

```
gcc 2.2.9_mpu6050.c -lwiringPi -lm
```

Step 5: Run the executable file.

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

With the code run, deflection angle of x axis, y axis and the acceleration, angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

Note:

- If there is an error prompt `wiringPi.h: No such file or directory`, please refer to *C code is not working?*.
- If you get `Unable to open I2C device: No such file or directory` error, you need to refer to *I2C Configuration* to enable I2C and check if the wiring is correct.

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;
```

(continues on next page)

(continued from previous page)

```
double gyroX_scaled, gyroY_scaled, gyroZ_scaled;

int read_word_2c(int addr)
{
int val;
val = wiringPiI2CReadReg8(fd, addr);
val = val << 8;
val += wiringPiI2CReadReg8(fd, addr+1);
if (val >= 0x8000)
    val = -(65536 - val);
return val;
}

double dist(double a, double b)
{
return sqrt((a*a) + (b*b));
}

double get_y_rotation(double x, double y, double z)
{
double radians;
radians = atan2(x, dist(y, z));
return -(radians * (180.0 / M_PI));
}

double get_x_rotation(double x, double y, double z)
{
double radians;
radians = atan2(y, dist(x, z));
return (radians * (180.0 / M_PI));
}

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

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

    acclX = read_word_2c(0x3B);
    acclY = read_word_2c(0x3D);
    acclZ = read_word_2c(0x3F);
```

(continues on next page)

(continued from previous page)

```
    acclX_scaled = acclX / 16384.0;
    acclY_scaled = acclY / 16384.0;
    acclZ_scaled = acclZ / 16384.0;

    //Print the X, Y, and Z values of the acceleration sensor.
    printf("My acclX_scaled: %f\n", acclX_scaled);
    printf("My acclY_scaled: %f\n", acclY_scaled);
    printf("My acclZ_scaled: %f\n", acclZ_scaled);

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

    delay(100);
}
return 0;
}
```

Code Explanation

```
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;
}
```

Read sensor data sent from MPU6050.

```
double get_y_rotation(double x, double y, double z)
{
double radians;
radians = atan2(x, dist(y, z));
return -(radians * (180.0 / M_PI));
}
```

We get the deflection angle on the Y-axis.

```
double get_x_rotation(double x, double y, double z)
{
double radians;
radians = atan2(y, dist(x, z));
return (radians * (180.0 / M_PI));
}
```

Calculate the deflection angle of the X-axis.

```
gyroX = read_word_2c(0x43);
gyroY = read_word_2c(0x45);
gyroZ = read_word_2c(0x47);
```

(continues on next page)

(continued from previous page)

```
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);
```

Read the values of the x axis, y axis and z axis on the gyroscope sensor, convert the metadata to angular velocity values, and then print them.

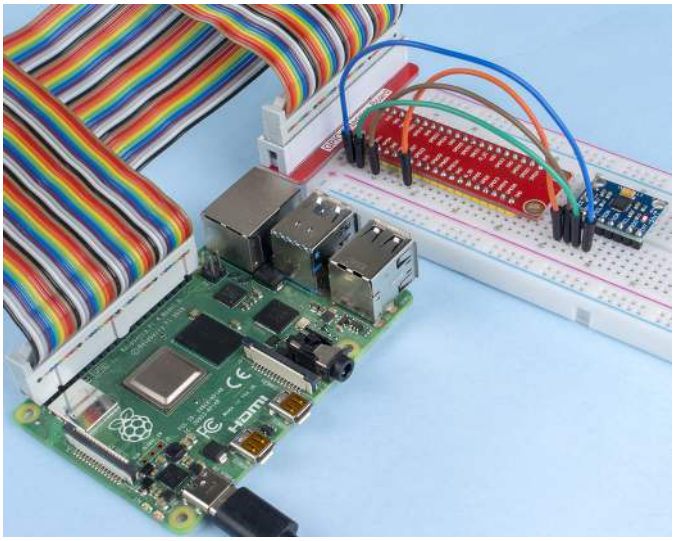
```
acclX = read_word_2c(0x3B);
acclY = read_word_2c(0x3D);
acclZ = read_word_2c(0x3F);

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);
```

Read the values of the x axis, y axis and z axis on the acceleration sensor, convert the metadata to accelerated speed values (gravity unit), and then print them.

```
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));
```

Print the deflection angles of the x-axis and y-axis.

Phenomenon Picture



2.2.10 MFRC522 RFID Module

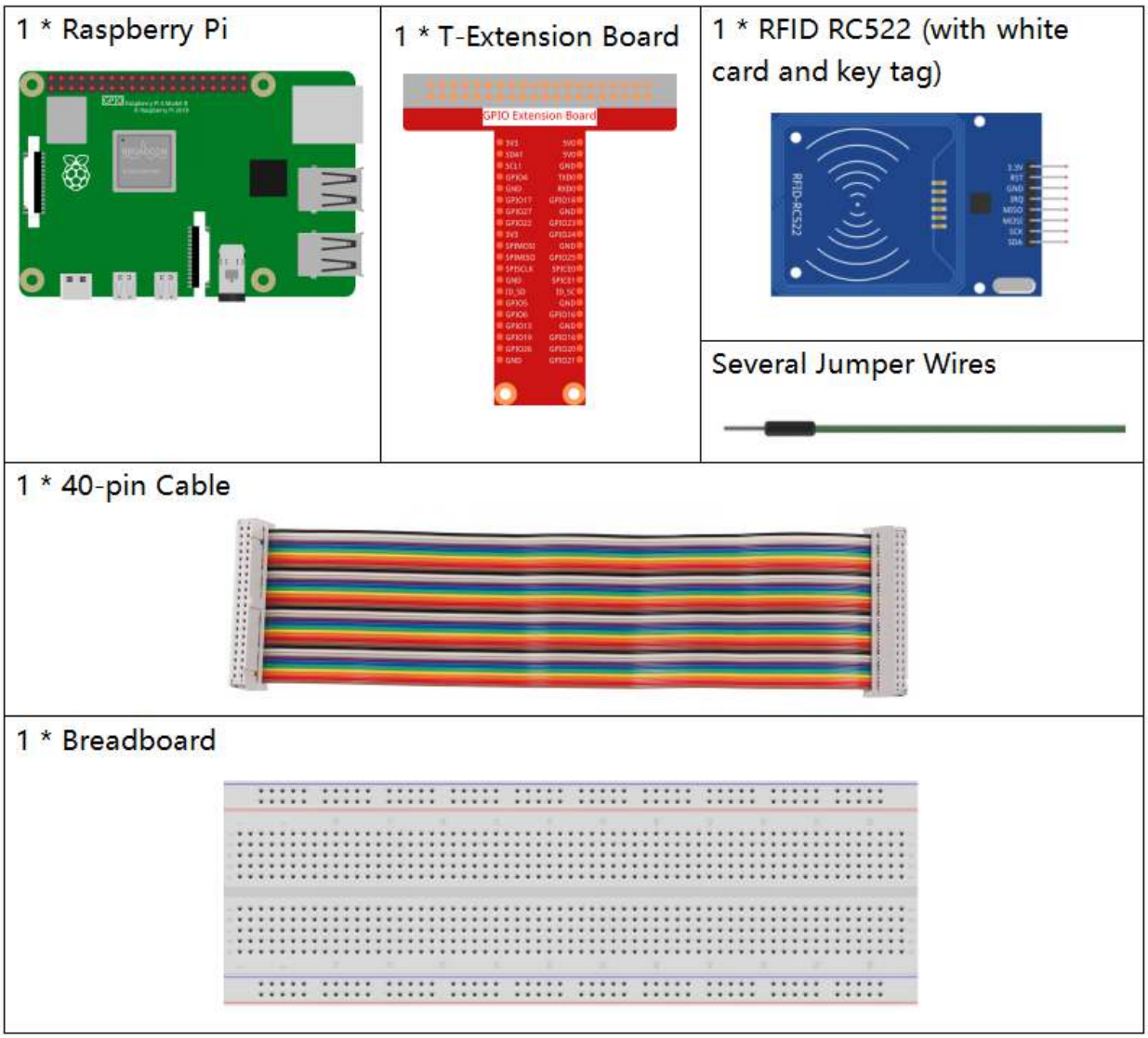
Introduction

Radio Frequency Identification (RFID) refers to technologies that use wireless communication between an object (or tag) and interrogating device (or reader) to automatically track and identify such objects.

Some of the most common applications for this technology include retail supply chains, military supply chains, automated payment methods, baggage tracking and management, document tracking and pharmaceutical management, to name a few.

In this project, we will use RFID for reading and writing.

Components

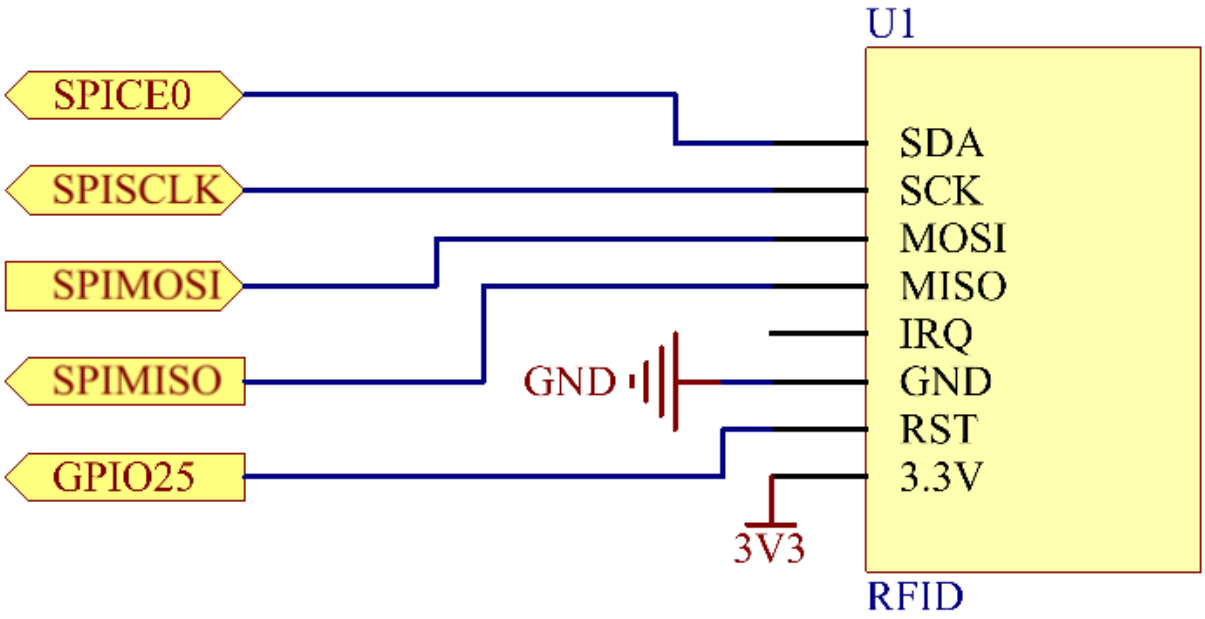


- *GPIO Extension Board*
- *Breadboard*

- *MFRC522 Module*

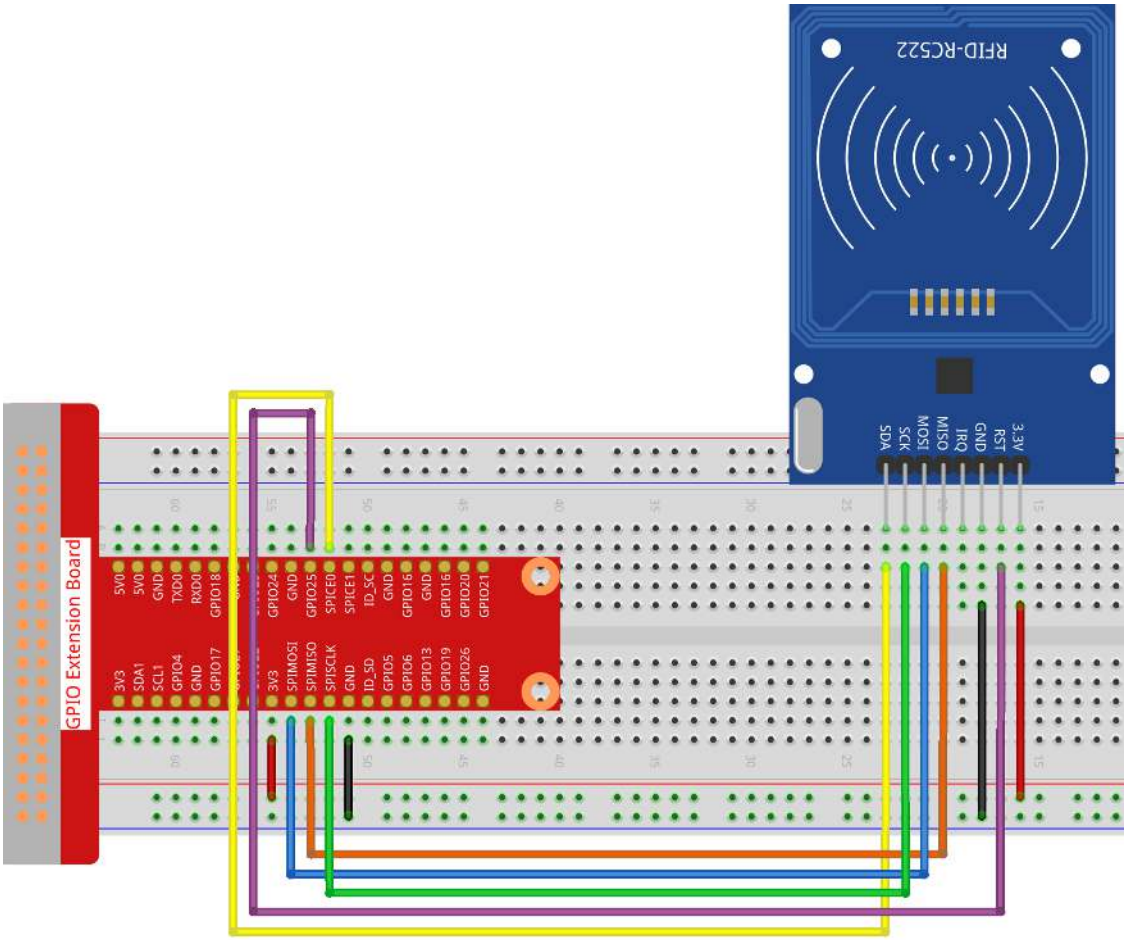
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Set up SPI (refer to *SPI Configuration* for more details. If you have set SPI, skip this step.)

Step 3: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/2.2.10/
```

Step 4: Compile the code.

```
make read
make write
```

Step 5: After running `./write`, enter the information, such as the name of the person, and then put the tag or card on the MRC522 module and wait for the writing to be completed.

```
sudo ./write
```

Step 6: Now run `./read` to read the information of the tag or card you have written.

```
sudo ./read
```

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please

refer to *C code is not working?*.

Code Explanation

```
InitRc522();
```

This function is used to initialize the RFID RC522 module.

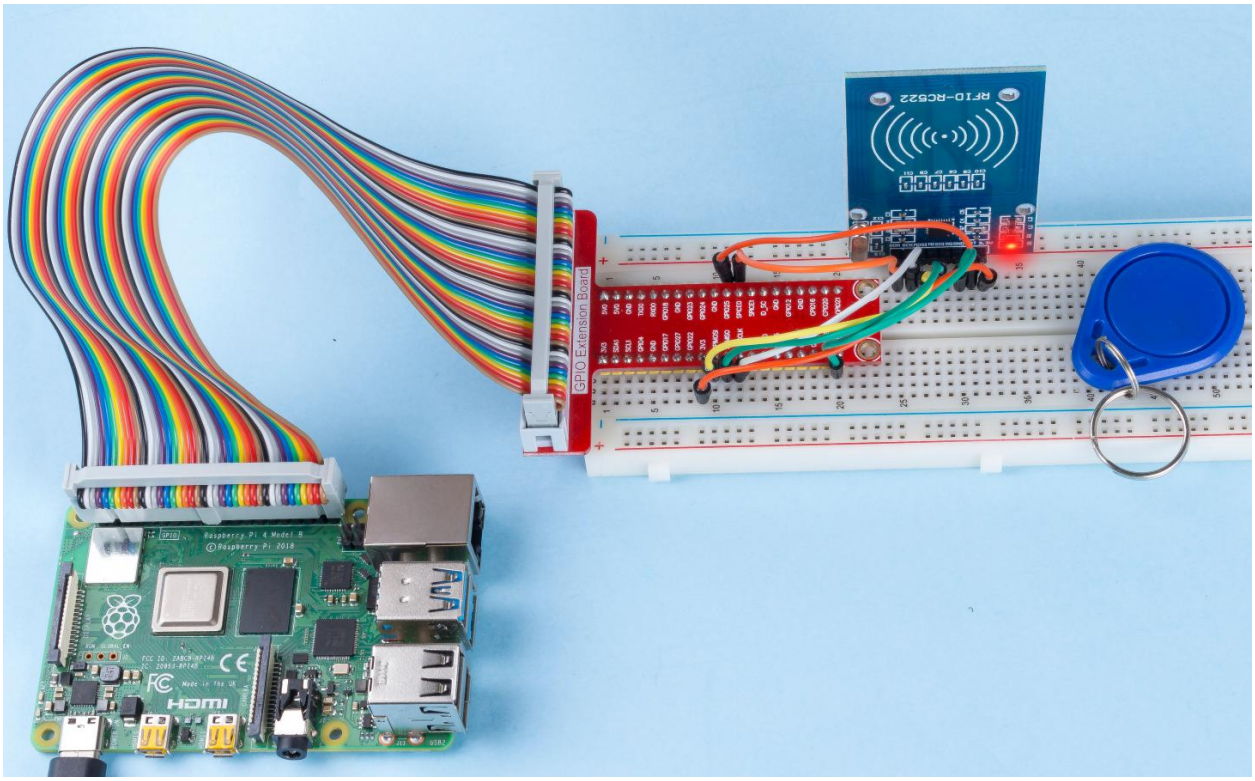
```
uint8_t read_card_data();
```

This function is used to read the data of the card, and if the read is successful, it will return "1".

```
uint8_t write_card_data(uint8_t *data);
```

This function is used to write the data of card and returns "1" if the write is successful. `*data` is the information that will be written to the card.

Phenomenon Picture



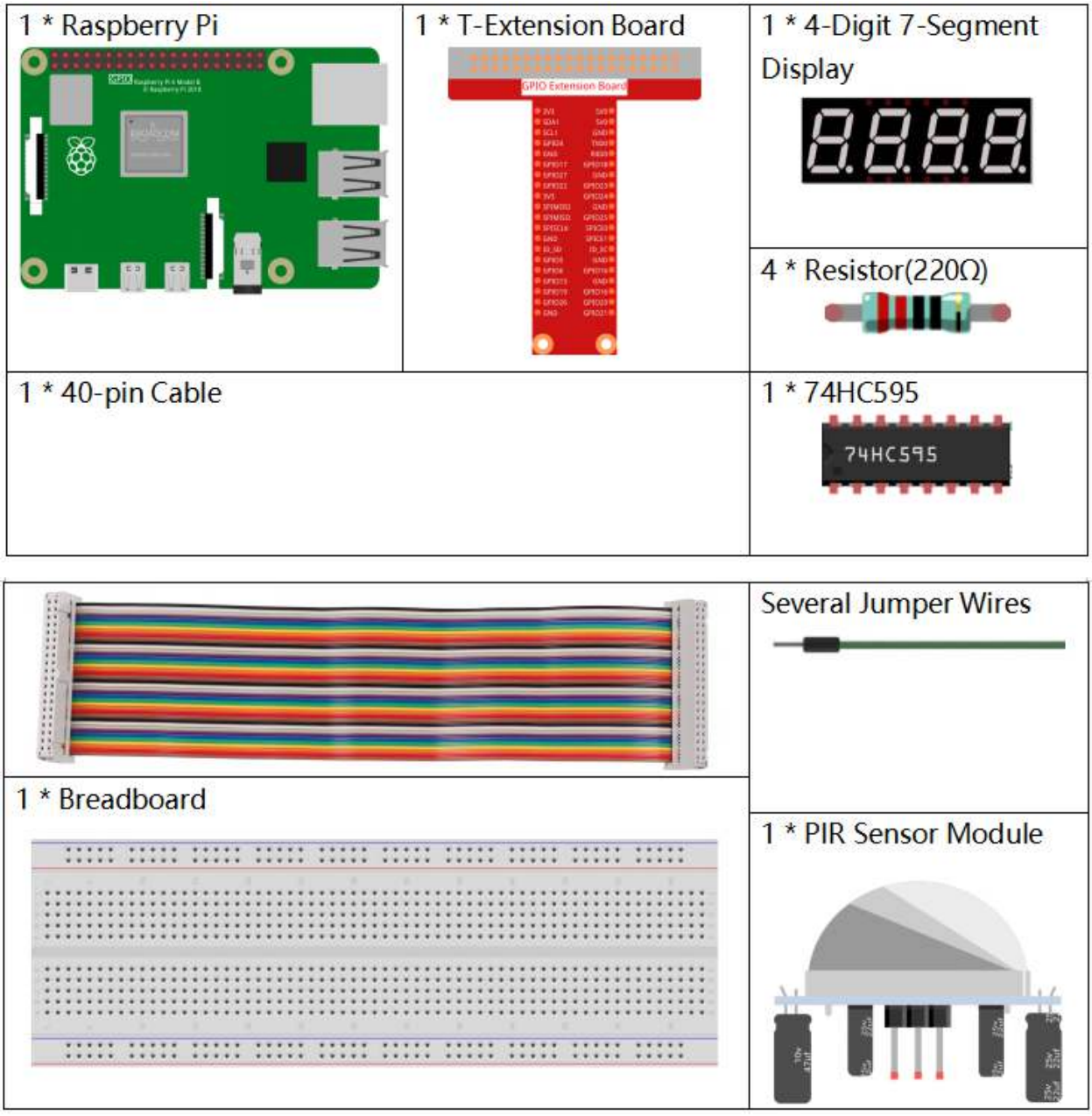
7.4 Extension

7.4.1 3.1.1 Counting Device

Introduction

Here we will make a number-displaying counter system, consisting of a PIR sensor and a 4-digit segment display. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1. You can use this counter to count the number of people walking through the passageway.

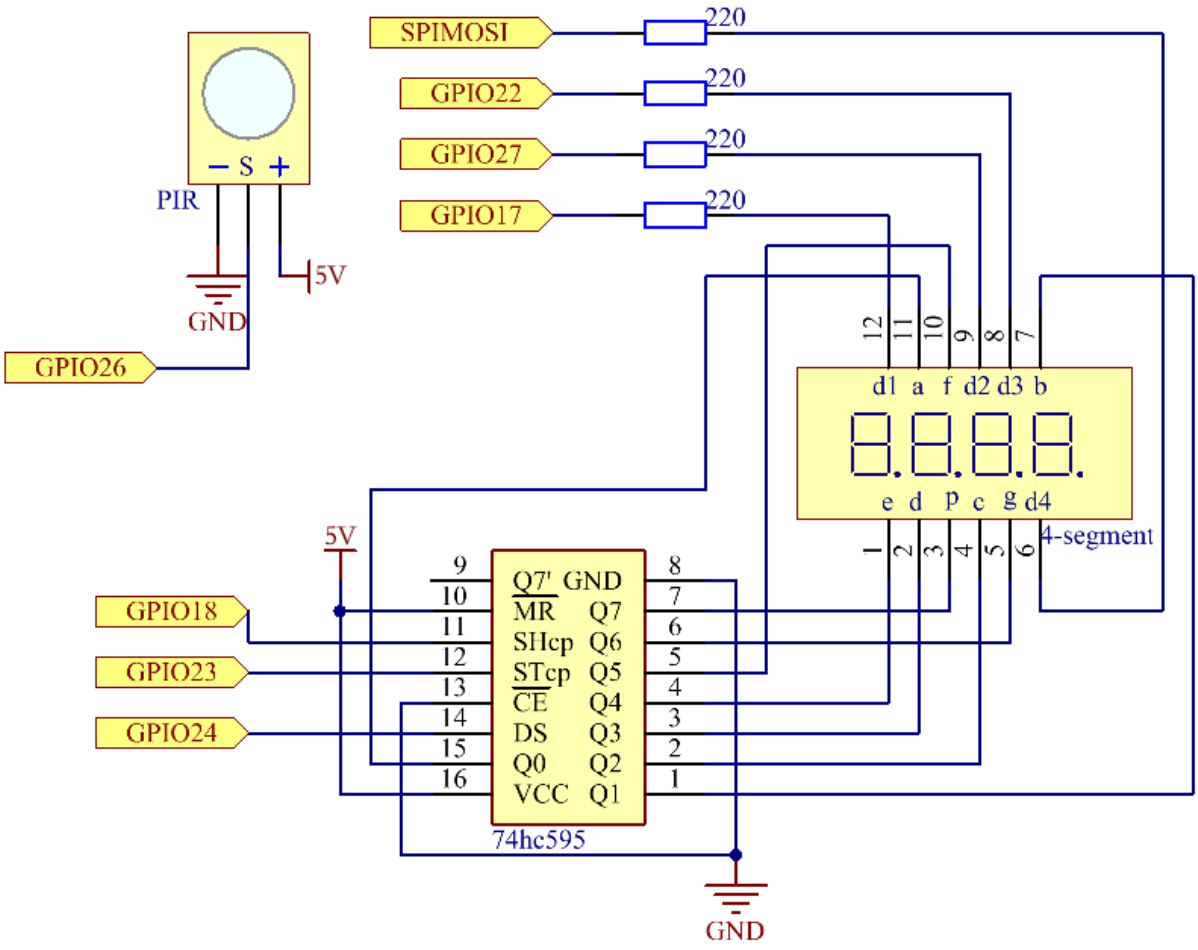
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*
- *PIR Motion Sensor Module*

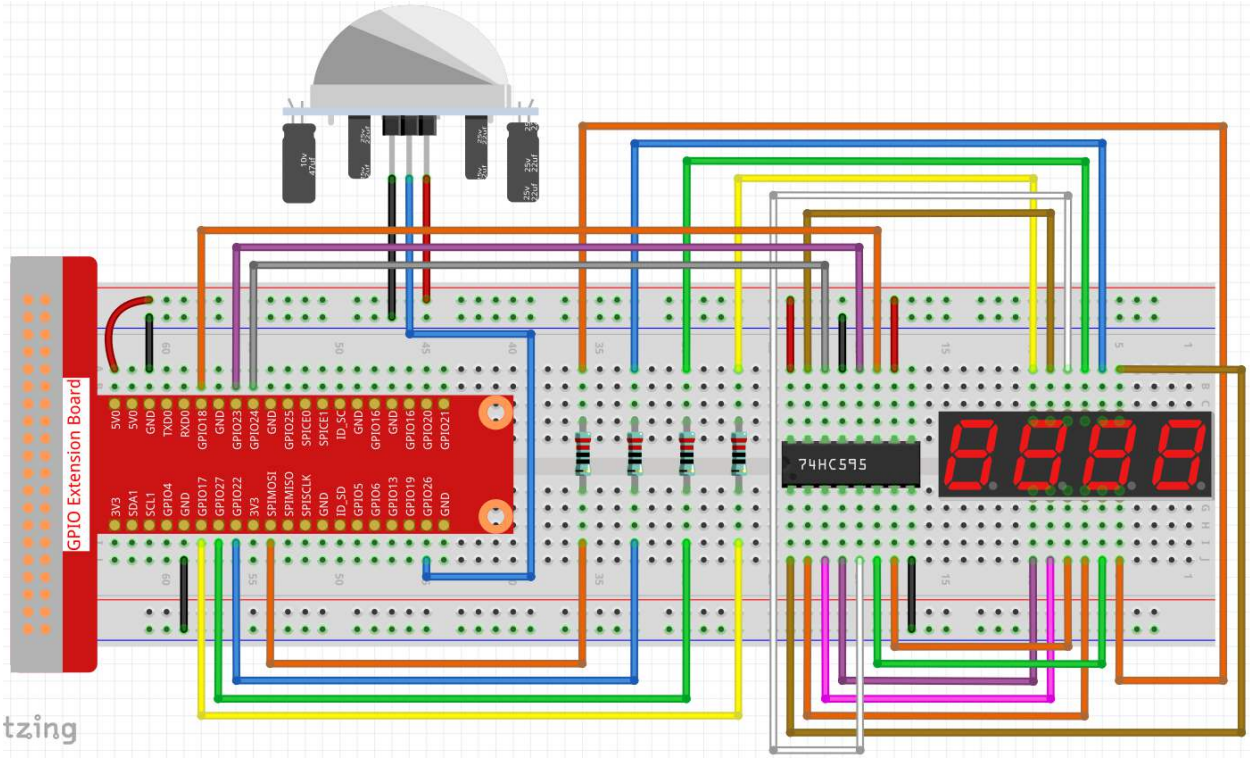
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.1/
```

Step 3: Compile the code.

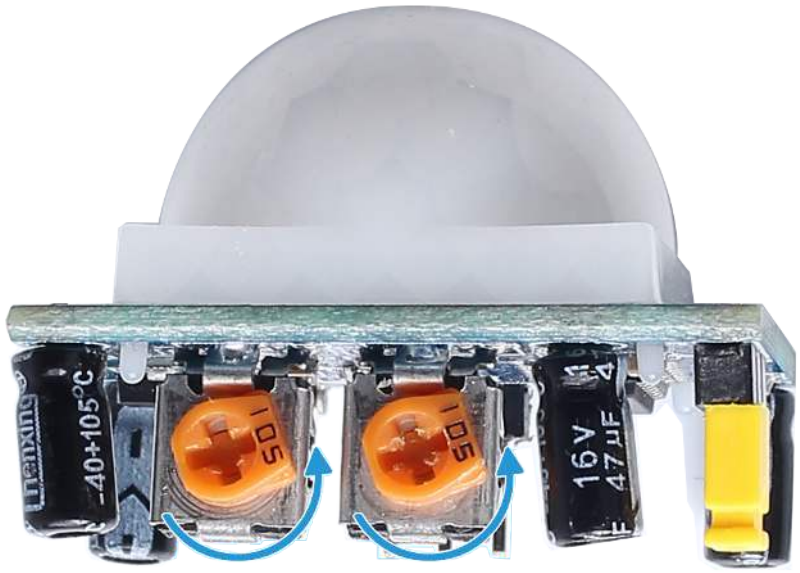
```
gcc 3.1.1_CountingDevice.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, when the PIR detects that someone is passing by, the number on the 4-digit segment display will add 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.



Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
void display()
{
    clearDisplay();
    pickDigit(0);
    hc595_shift(number[counter % 10]);

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

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

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

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
```

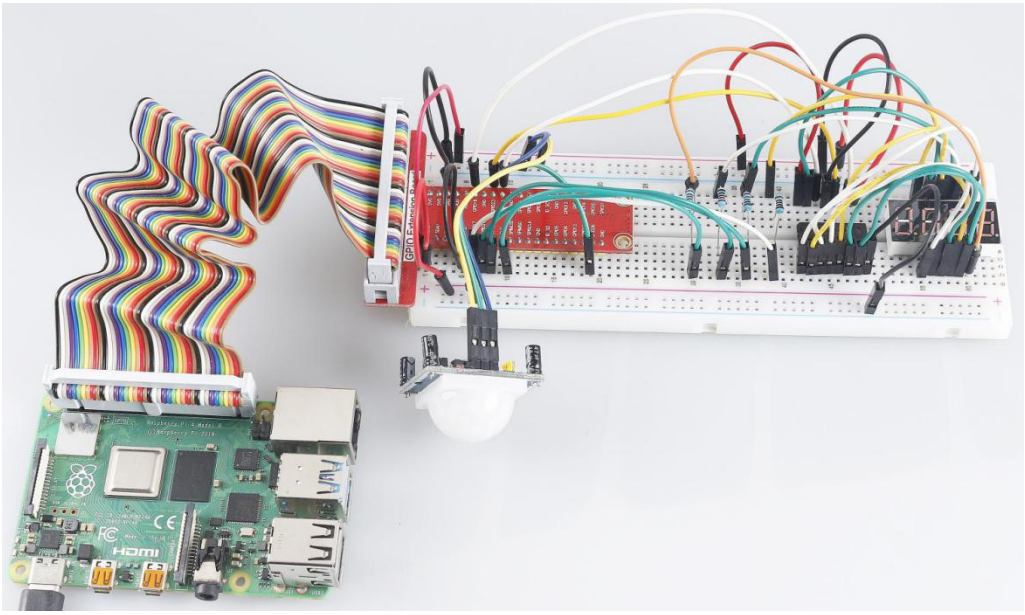
(continues on next page)

(continued from previous page)

```
            counter +=1;
        }
        lastState=currentState;
    }
}
```

This is the main function: display the number on the 4-digit segment display and read the PIR value. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

Phenomenon Picture

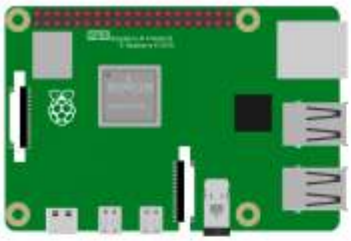
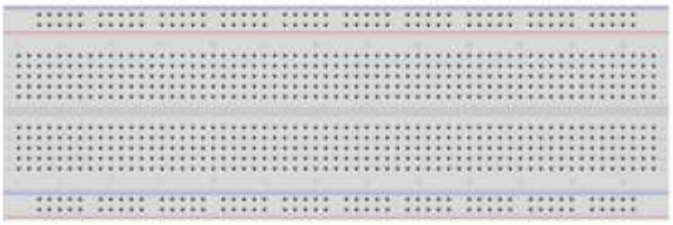


7.4.2 3.1.2 Welcome

Introduction

In this project, we will use PIR to sense the movement of pedestrians, and use servos, LED, buzzer to simulate the work of the sensor door of the convenience store. When the pedestrian appears within the sensing range of the PIR, the indicator light will be on, the door will be opened, and the buzzer will play the opening bell.

Components

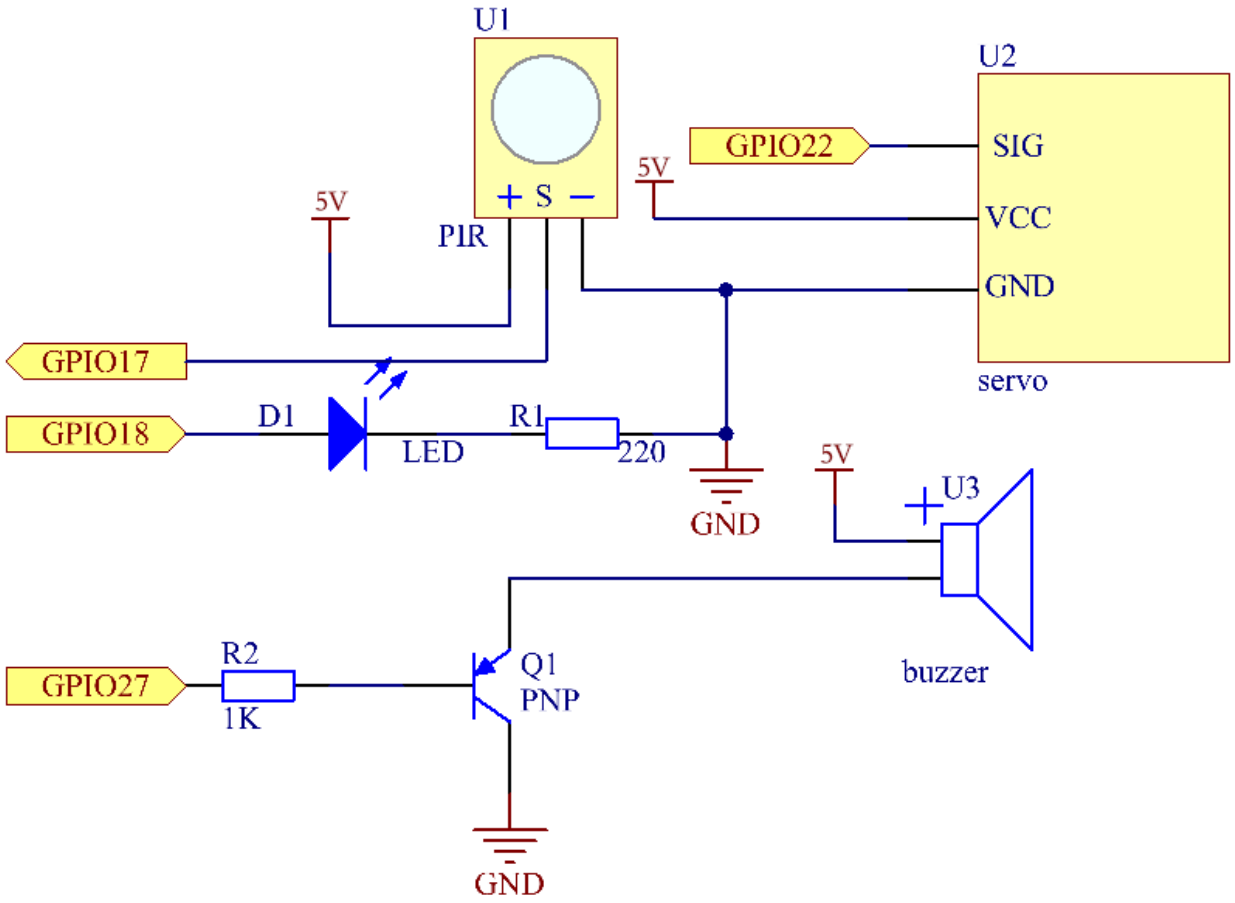
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td colspan="2">1 * PIR
 </td></tr>
<tr><td>1 * Servo
 </td><td>1 * Passive Buzzer
 </td><td>1 * S8550 PNP Transistor
 </td><td>1 * LED
 </td></tr>
<tr><td>1 * Resistor 1kΩ
 </td><td>1 * Resistor 220Ω
 </td><td colspan="2">Several Jumper Wires
</td></tr>
<tr><td colspan="4">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="4">1 * Breadboard
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *PIR Motion Sensor Module*
- *Servo*

- *Buzzer*
- *Transistor*

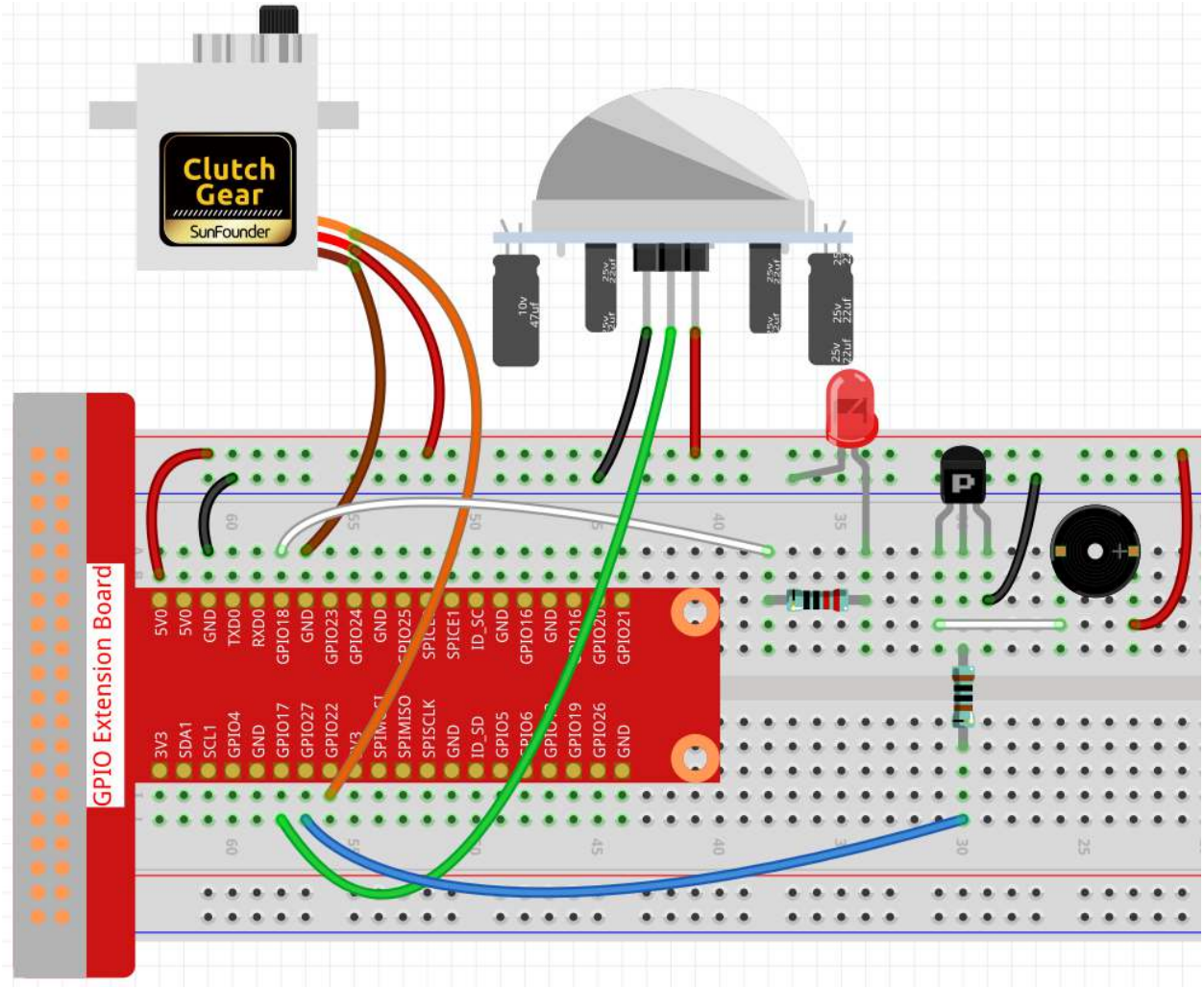
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.2/
```

Step 3: Compile.

```
gcc 3.1.2_Welcome.c -lwiringPi
```

Step 4: Run.

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

After the code runs, if the PIR sensor detects someone passing by, the door will automatically open (simulated by the servo), turn on the indicator and play the doorbell music. After the doorbell music plays, the system will automatically close the door and turn off the indicator light, waiting for the next time someone passes 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.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
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));
}
```

Create a function, setAngle to write the angle in the servo that is 0-180.

```
void doorbell(){
for(int i=0;i<sizeof(song)/4;i++){
        softToneWrite(BuzPin, song[i]);
        delay(beat[i] * 250);
    }
```

Create a function, doorbell to enable the buzzer to play music.

```
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);
    }
}
```

Create a closedoor function to simulate closing the door, turn off the LED and let the servo turn from 180 degrees to 0 degree.

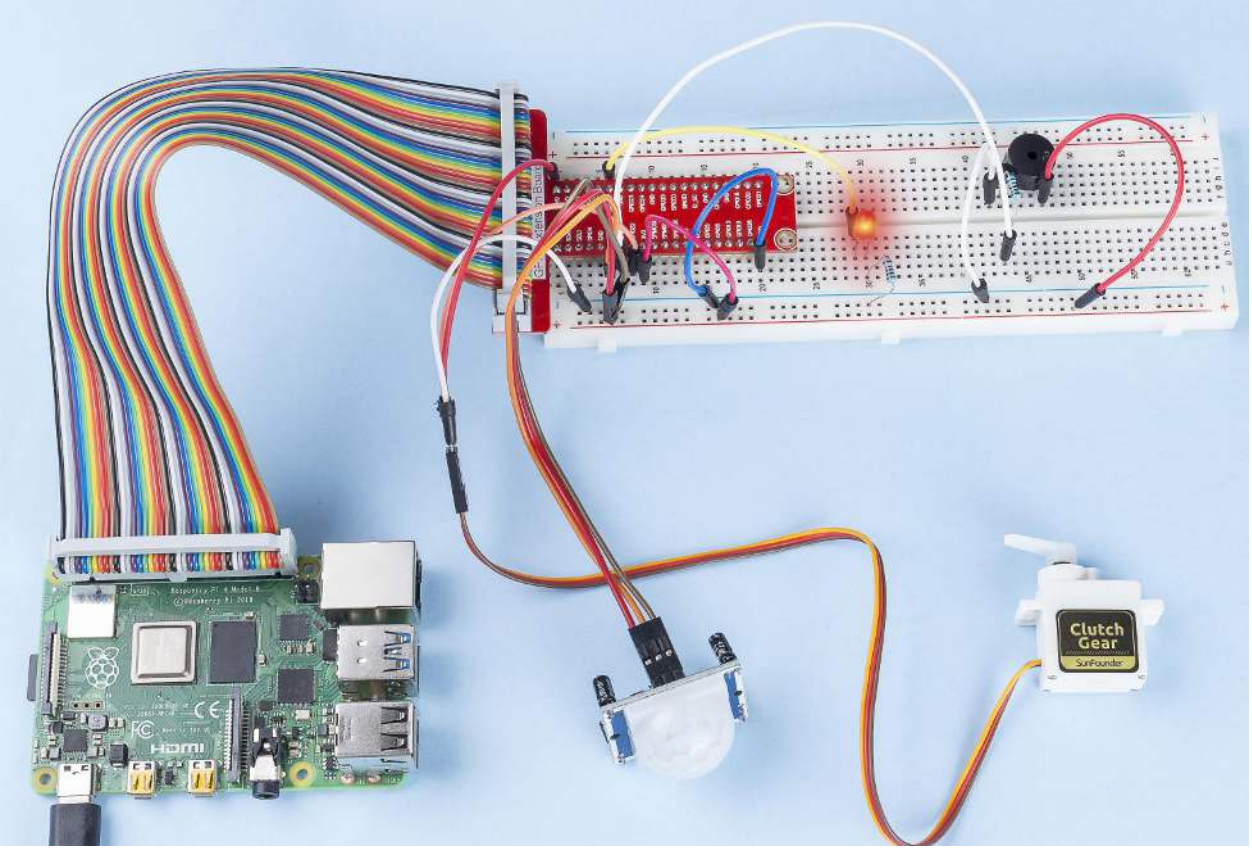
```
void opendoor(){
    digitalWrite(ledPin, HIGH);   //led on
    for(int i=0;i<181;i++){  //make servo rotate from minimum angle to maximum angle
        setAngle(servoPin,i);
        delay(1);
    }
    doorbell();
    closedoor();
}
```

The function opendoor() includes several parts: turn on the indicator light, turn the servo (simulate the action of opening the door), play the doorbell music of the convenience store, and call the function closedoor() after playing music.

```
    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;
......
```

In the function main(), initialize library wiringPi and setup softTone, then set ledPin to output state and pirPin to input state. If the PIR sensor detects someone passing by, the function opendoor will be called to simulate opening the door.

Phenomenon Picture

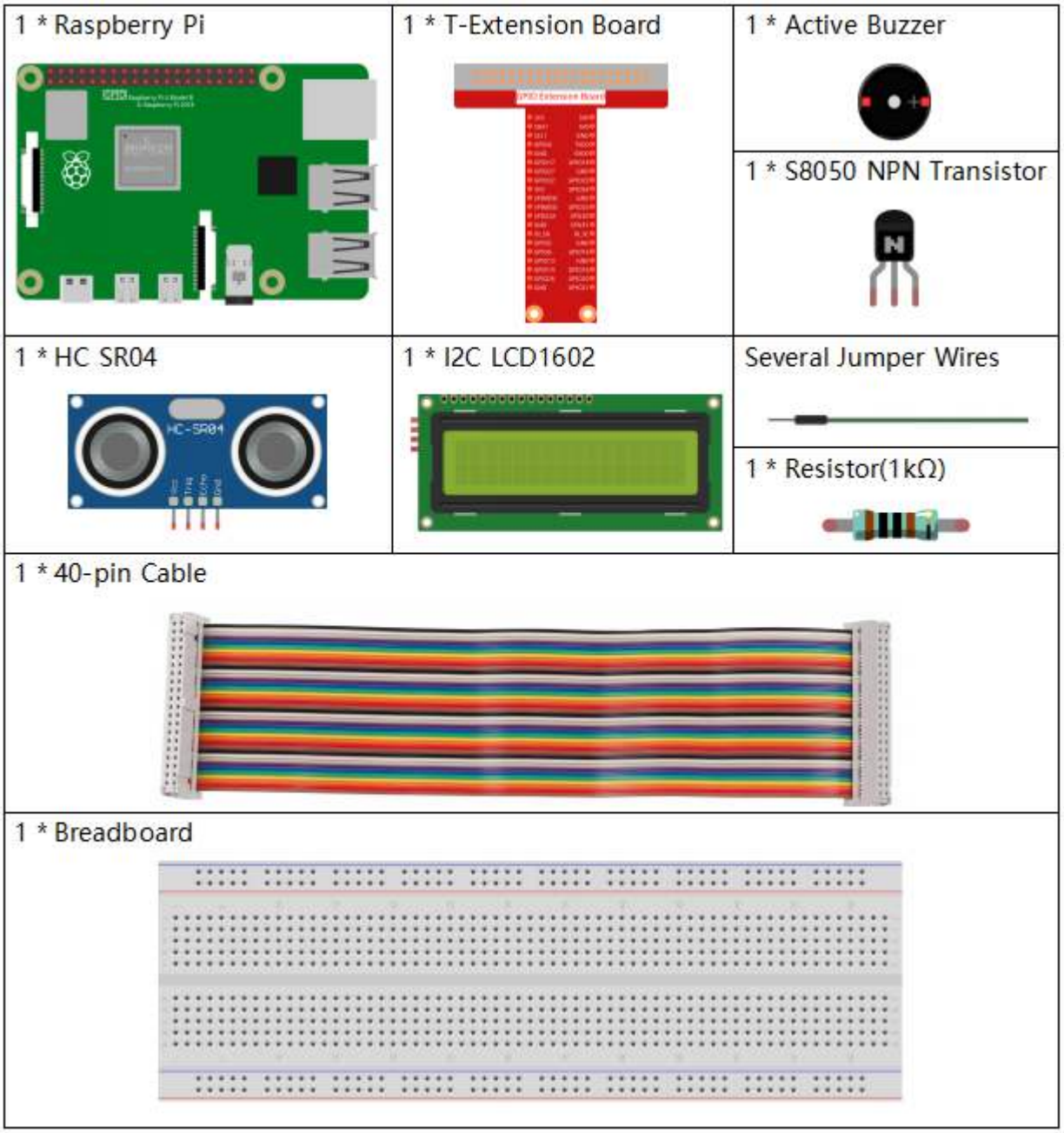


7.4.3 3.1.3 Reversing Alarm

Introduction

In this project, we will use LCD, buzzer and ultrasonic sensors to make a reverse assist system. We can put it on the remote control vehicle to simulate the actual process of reversing the car into the garage.

Components

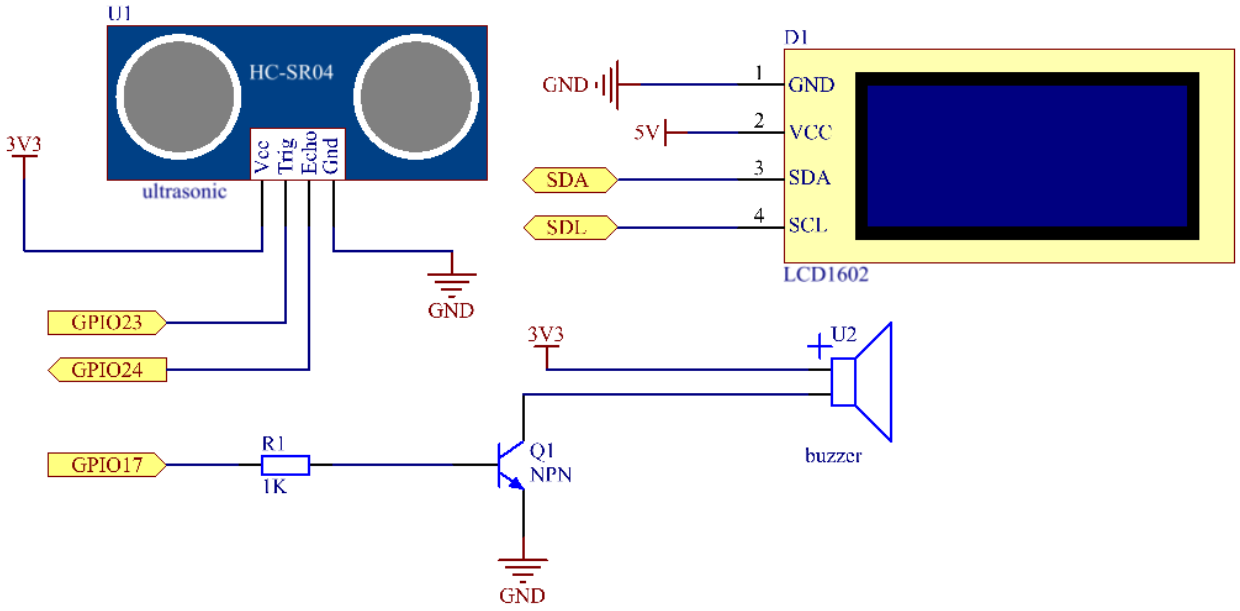


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*
- *Ultrasonic Module*
- *I2C LCD1602*

Schematic Diagram

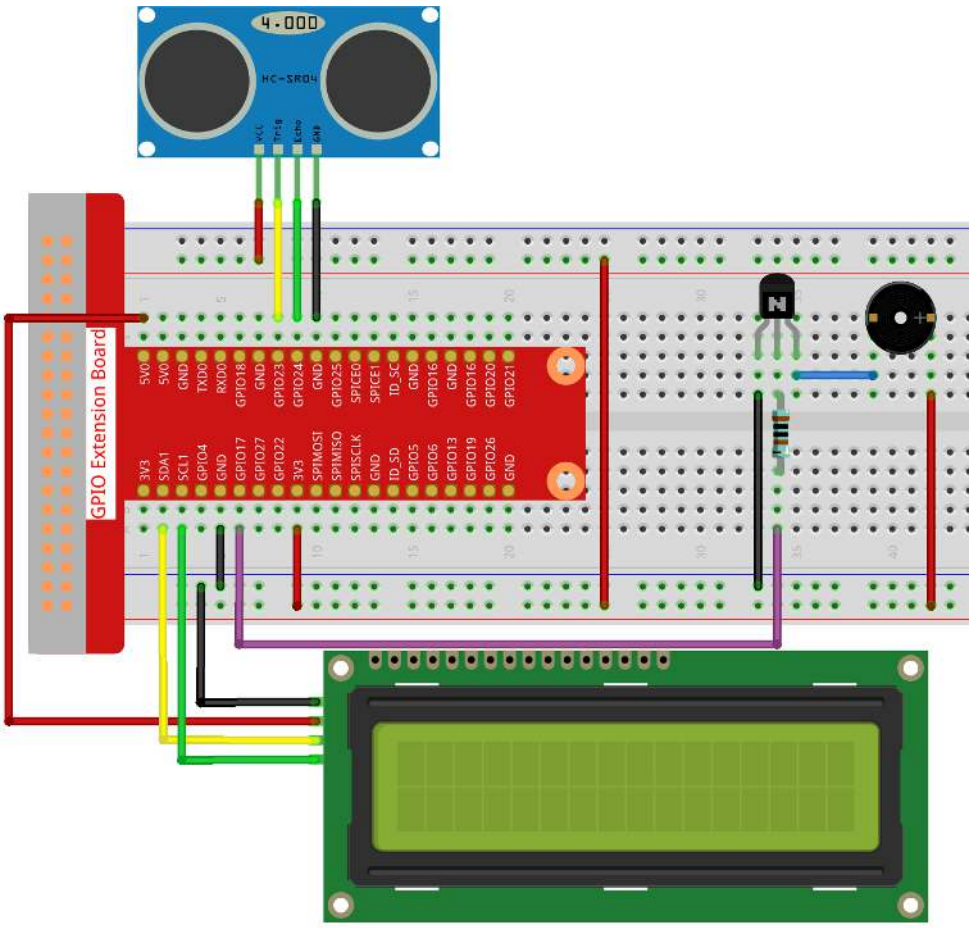
Ultrasonic sensor detects the distance between itself and the obstacle that will be displayed on the LCD in the form of code. At the same time, the ultrasonic sensor let the buzzer issue prompt sound of different frequency according to different distance value.

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.3/
```

Step 3: Compile.

```
gcc 3.1.3_ReversingAlarm.c -lwiringPi
```

Step 4: Run.

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

As the code runs, ultrasonic sensor module detects the distance to the obstacle and then displays the information about the distance on LCD1602; besides, buzzer emits warning tone whose frequency changes with the distance.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

Note: The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.1_ReversingAlarm.c.

```
#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){
        dis = disMeasure();
        printf("%.2f cm \n",dis);
        digitalWrite(Buzzer,LOW);
        if (dis > 400){
```

(continues on next page)

(continued from previous page)

```
            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 Explanation

```
pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();
```

In this program, we apply previous components synthetically. Here we use buzzers, LCD and ultrasonic. We can initialize them the same way as we did before.

```
dis = disMeasure();
 printf("%.2f cm \n",dis);
digitalWrite(Buzzer,LOW);
if (dis > 400){
     write(0, 0, "Error");
     write(3, 1, "Out of range");
}
else
{
    write(0, 0, "Distance is");
```

(continues on next page)

(continued from previous page)

```
    sprintf(result,"%.2f cm",dis);
    write(5, 1, result);
    }
```

Here we get the value of the ultrasonic sensor and get the distance through calculation.

If the value of distance is greater than the range value to be detected, an error message is printed on the LCD. And if the distance value is within the range, the corresponding results will be output.

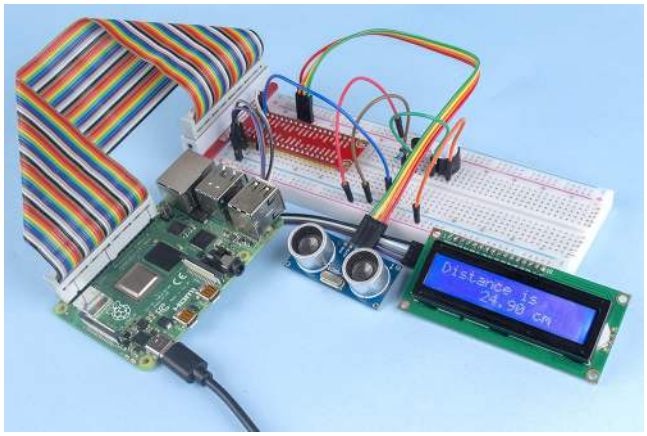
```
sprintf(result,"%.2f cm",dis);
```

Since the output mode of LCD only supports character type, and the variable dis stores the value of float type, we need to use sprintf(). The function converts the float type value to a character and stores it on the string variable result[]. %.2f means to keep two decimal places.

```
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);
    }
}
```

This judgment condition is used to control the sound of the buzzer. According to the difference in distance, it can be divided into three cases, in which there will be different sound frequencies. Since the total value of delay is 500, all of the cases can provide a 500ms interval for the ultrasonic sensor.

Phenomenon Picture

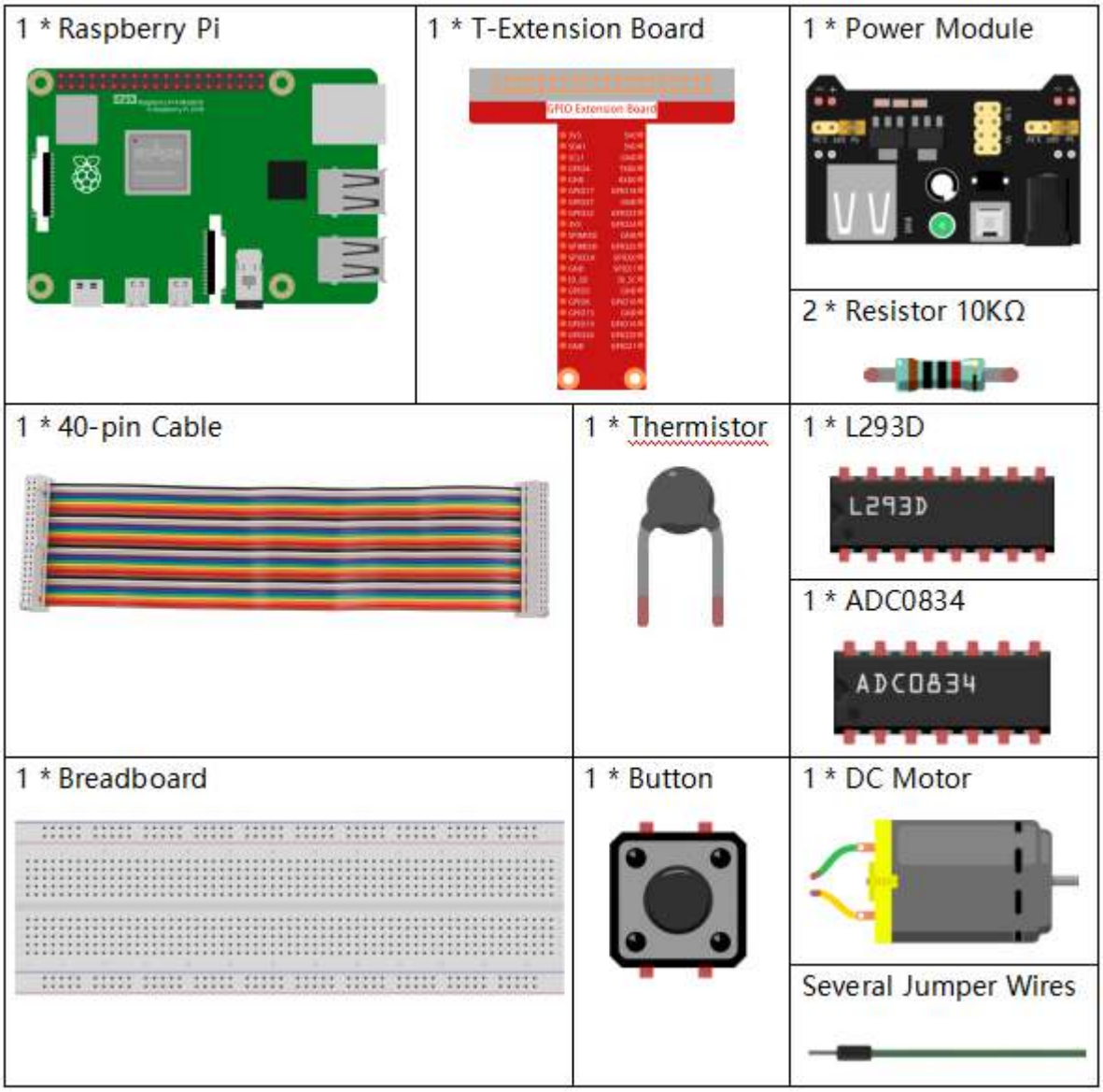


7.4.4 3.1.4 Smart Fan

Introduction

In this project, we will use motors, buttons and thermistors to make a manual + automatic smart fan whose wind speed is adjustable.

Components

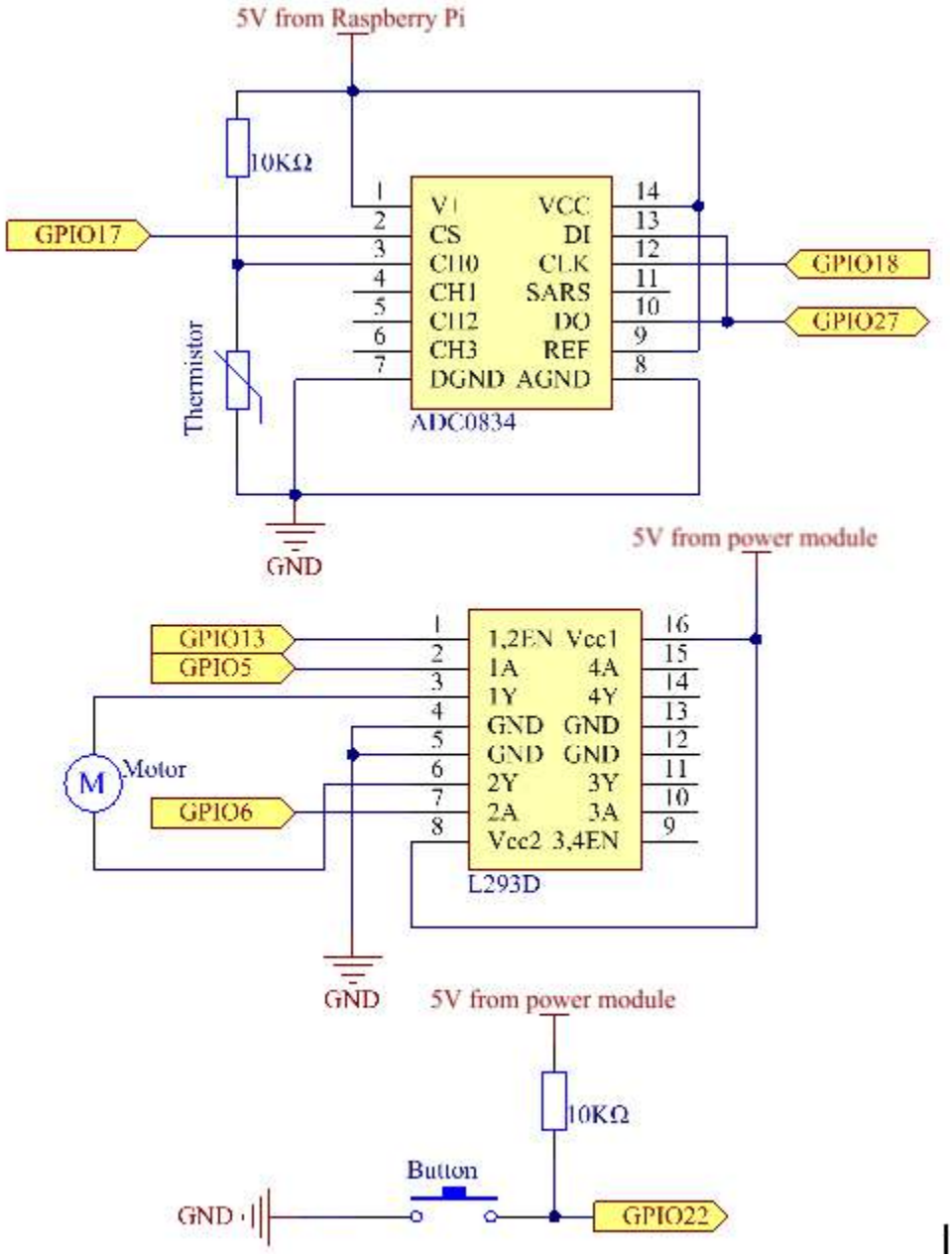


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Power Supply Module*

- *Thermistor*
- *L293D*
- *ADC0834*
- *Button*
- *DC Motor*

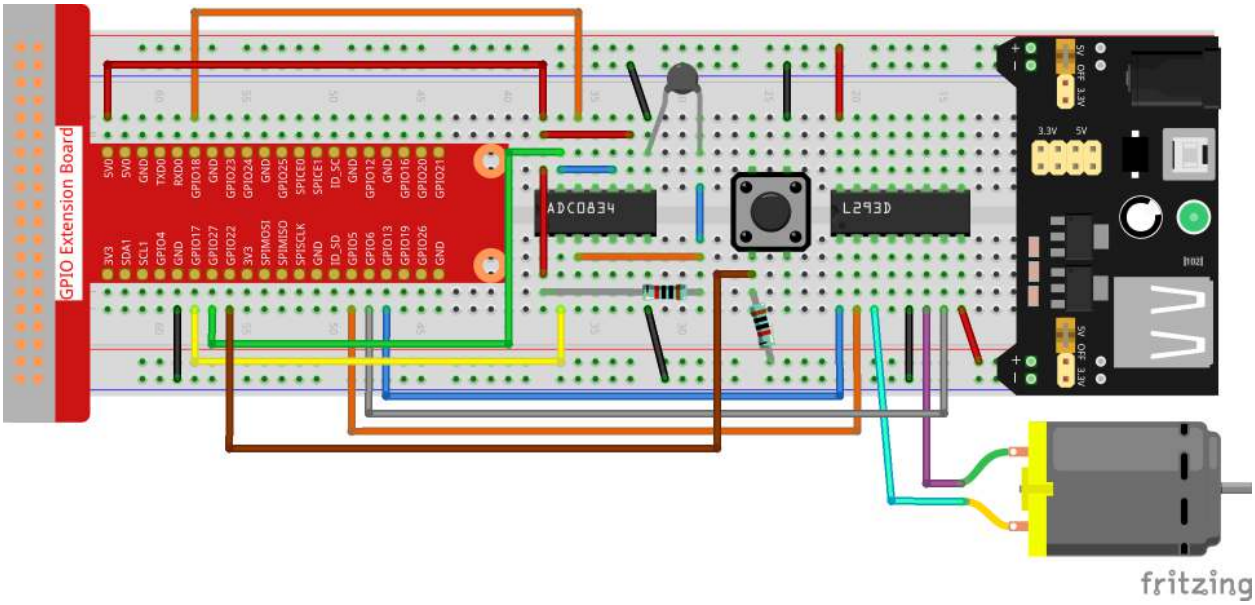
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13



Experimental Procedures

Step 1: Build the circuit.



Note: 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.

Step 2: Get into the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.4/
```

Step 3: Compile.

```
gcc 3.1.4_SmartFan.c -lwiringPi -lm
```

Step 4: Run the executable file above.

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

As the code runs, start the fan by pressing the button. Every time you press, 1 speed grade is adjusted up or down. There are **5** kinds of speed grades: **0~4**. When set to the 4th speed grade and you press the button, the fan stops working with a **0** wind speed.

Once the temperature goes up or down for more than 2°C, the speed automatically gets 1-grade faster or slower.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;

#define ADC_CS      0
#define ADC_CLK     1
#define ADC_DIO     2
#define MotorPin1   21
#define MotorPin2   22
#define MotorEnable 23
#define BtnPin      3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
//Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);  delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);    delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
```

(continues on next page)

(continued from previous page)

```
        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 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;
}

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;
}
void setup(){
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return;
    }
    softPwmCreate(MotorPin1,  0, 100);
    softPwmCreate(MotorPin2,  0, 100);
    pinMode(MotorEnable,OUTPUT);
    pinMode(BtnPin,INPUT);
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);
}

int main(void)
```

(continues on next page)

(continued from previous page)

```
{
    setup();
    int currentState,lastState=0;
    int level = 0;
    int currentTemp,markTemp=0;
    while(1){
        currentState=digitalRead(BtnPin);
        currentTemp=temperture();
        if (currentTemp<=0){continue;}
        if (currentState==1&&lastState==0){
            level=(level+1)%5;
            markTemp=currentTemp;
            delay(500);
        }
        lastState=currentState;
        if (level!=0){
            if (currentTemp-markTemp<=-2){
                level=level-1;
                markTemp=currentTemp;
            }
            if (currentTemp-markTemp>=2){
                level=level+1;
                markTemp=currentTemp;
            }
        }
        level=motor(level);
    }
    return 0;
}
```

Code Explanation

```
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() works by converting thermistor values read by ADC0834 into temperature values. Refer to *2.2.2 Thermistor* for more details.

```
int motor(int level){
    if(level==0){
        digitalWrite(MotorEnable,LOW);
        return 0;
    }
    if (level>=4){
        level =4;
```

(continues on next page)

(continued from previous page)

```
    }
    digitalWrite(MotorEnable,HIGH);
    softPwmWrite(MotorPin1, level*25);
    return level;
}
```

This function controls the rotating speed of the motor. The range of the **Level**: **0-4** (level **0** stops the working motor). One level adjustment stands for a **25%** change of the wind speed.

```
int main(void)
{
    setup();
    int currentState,lastState=0;
    int level = 0;
    int currentTemp,markTemp=0;
    while(1){
        currentState=digitalRead(BtnPin);
        currentTemp=temperture();
        if (currentTemp<=0){continue;}
        if (currentState==1&&lastState==0){
            level=(level+1)%5;
            markTemp=currentTemp;
            delay(500);
        }
        lastState=currentState;
        if (level!=0){
            if (currentTemp-markTemp<=-2){
                level=level-1;
                markTemp=currentTemp;
            }
            if (currentTemp-markTemp>=2){
                level=level+1;
                markTemp=currentTemp;
            }
        }
        level=motor(level);
    }
    return 0;
}
```

The function **main()** contains the whole program process as shown:

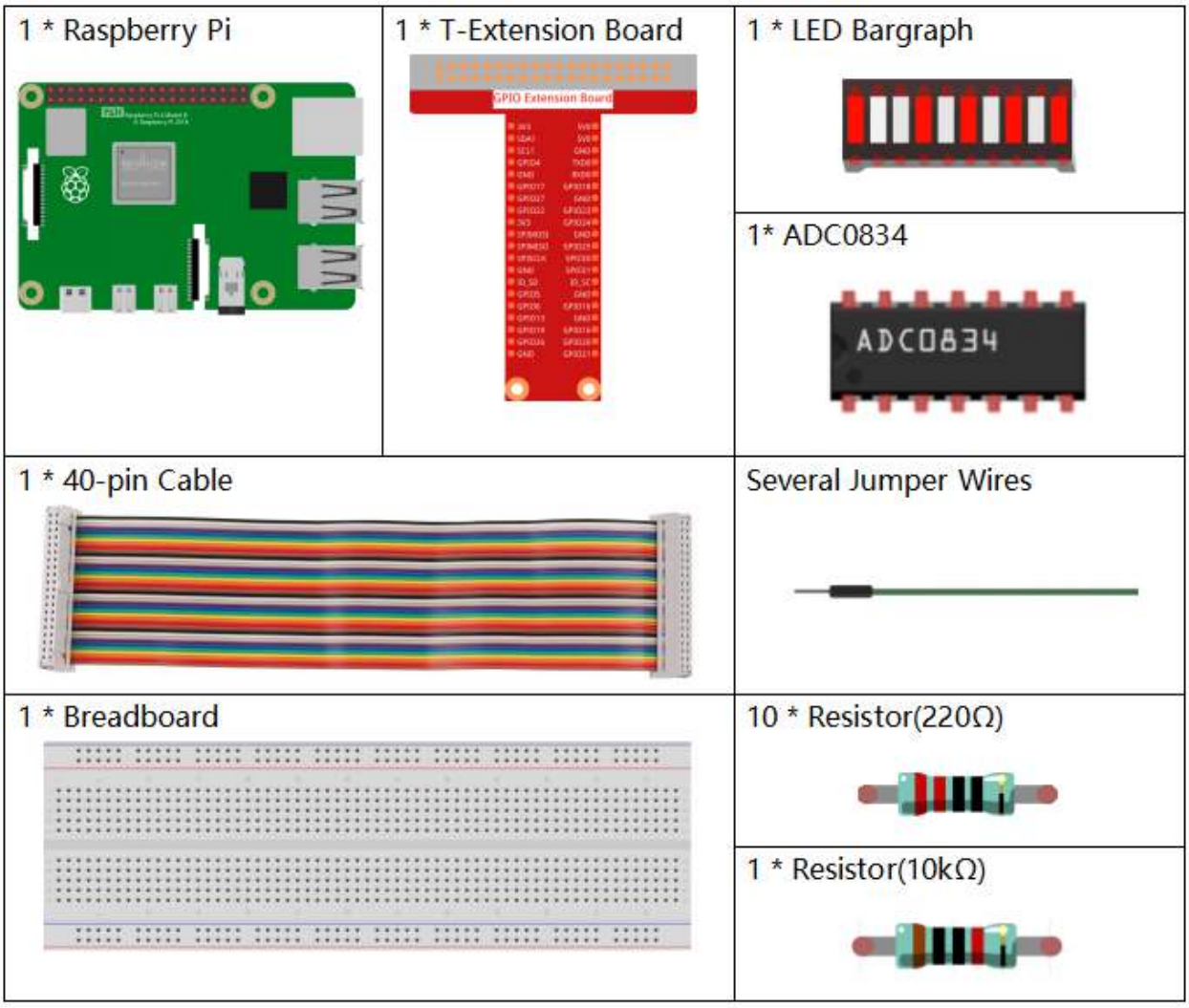
1) Constantly read the button state and the current temperature.
2) Every press makes level**+1** and at the same time, the temperature is updated. The **Level** ranges **1~4**.
3) As the fan works (the level is **not 0**), the temperature is under detection. A **2°C+** change causes the up and down of the level.
4) The motor changes the rotating speed with the **Level**.

7.4.5 3.1.5 Battery Indicator

Introduction

In this project, we will make a battery indicator device that can visually display the battery level on the LED Bargraph.

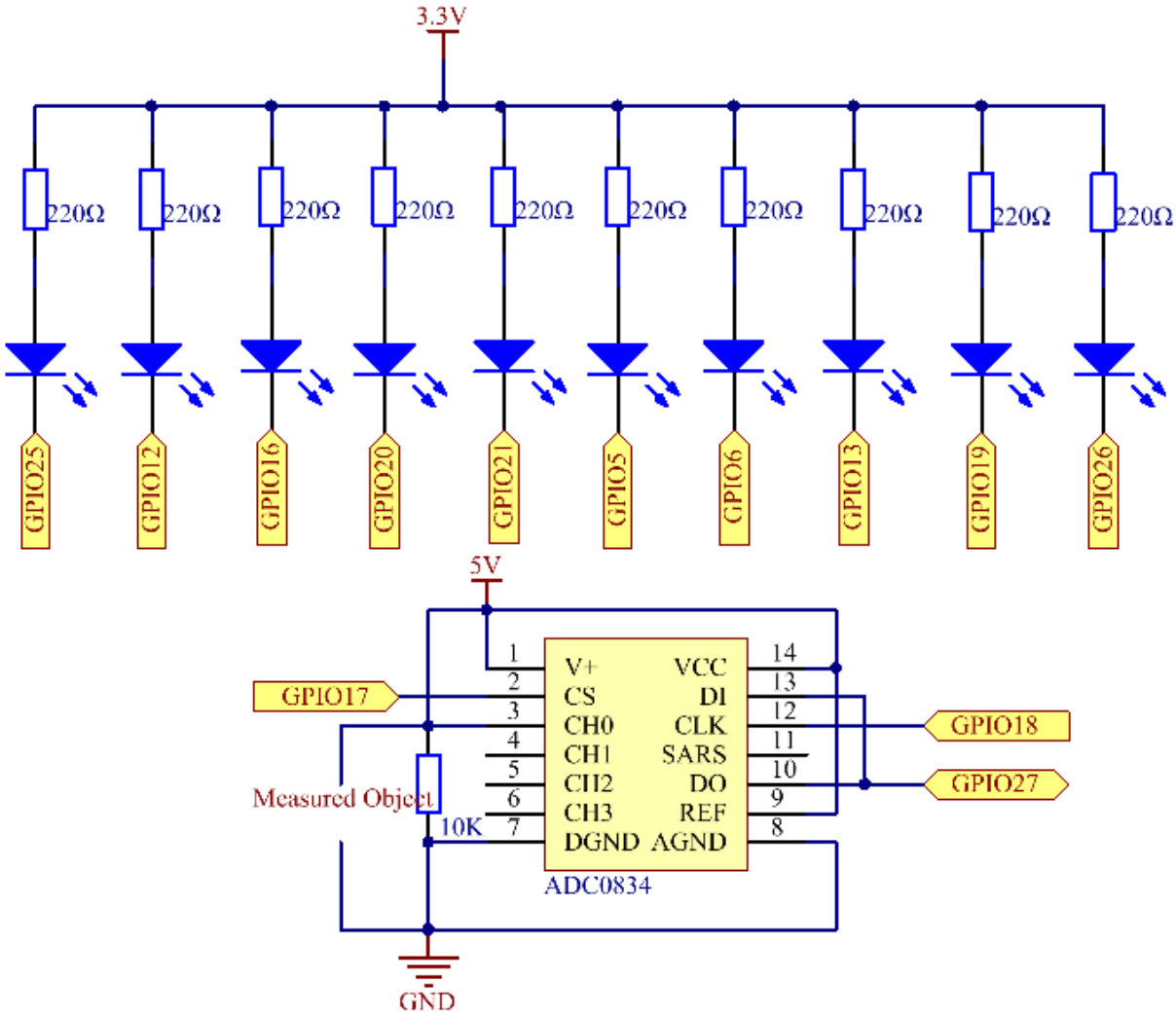
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*
- *ADC0834*

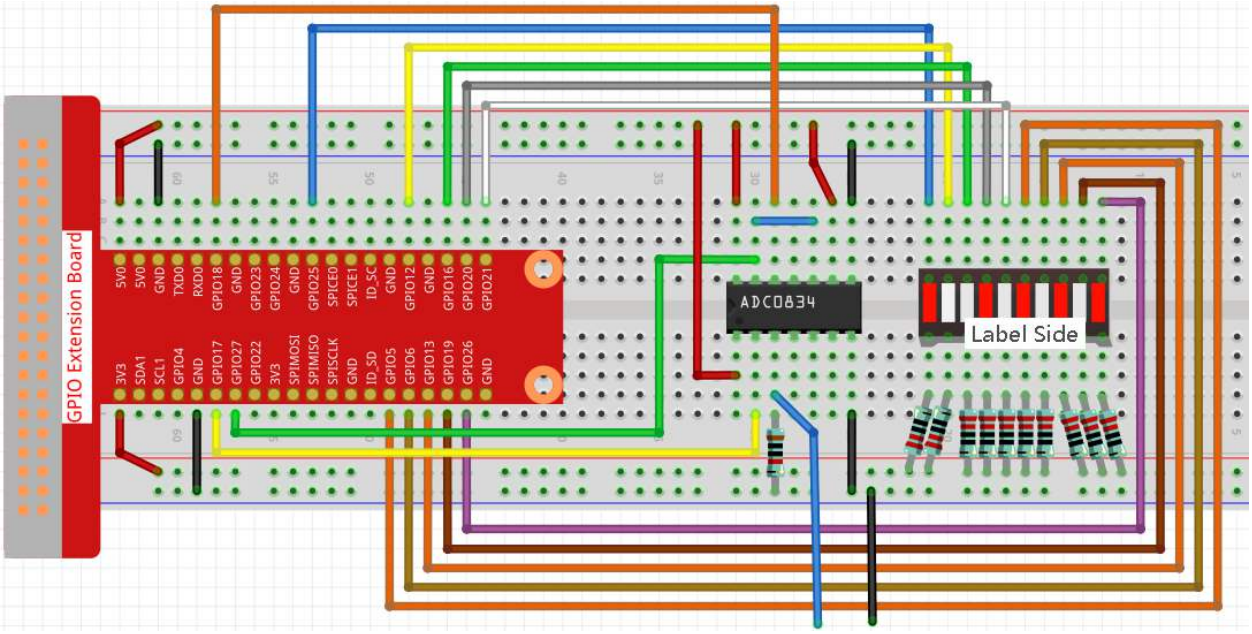
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO25	Pin 22	6	25
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	20
GPIO21	Pin 40	29	21
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13
GPIO19	Pin 35	24	19
GPIO26	Pin 37	25	26



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.5/
```

Step 3: Compile the code.

```
gcc 3.1.5_BatteryIndicator.c -lwiringPi
```

Step 4: Run the executable file.

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

After the program runs, give the 3rd pin of ADC0834 and the GND a lead-out wire separately and then lead them to the two poles of a battery separately. You can see the corresponding LED on the LED Bargraph is lit up to display the power level (measuring range: 0-5V).

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
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);
    }
}
```

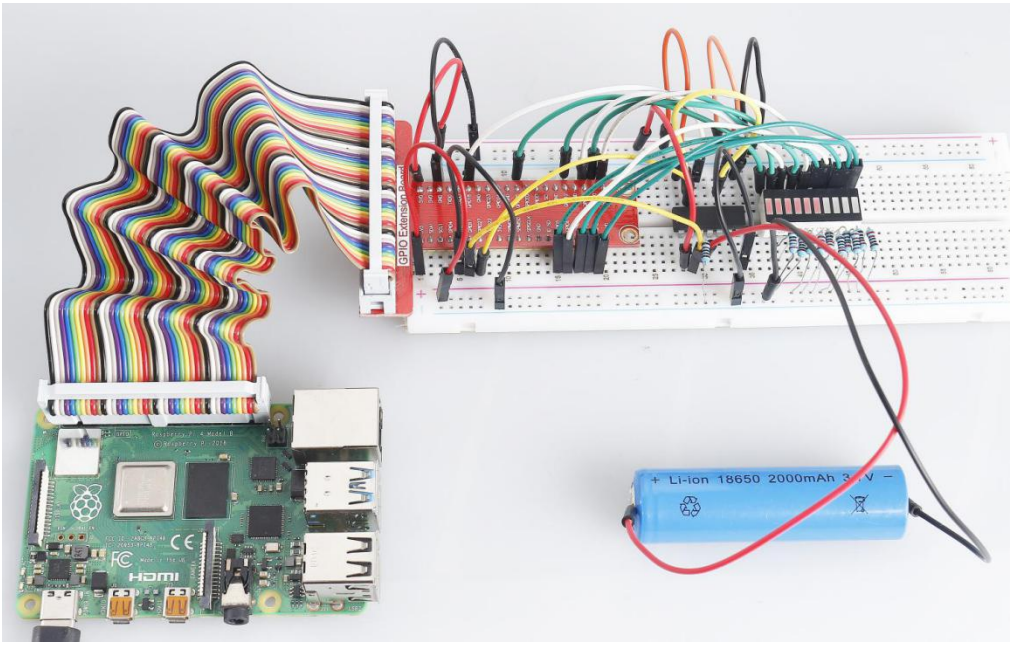
This function works for controlling the turning on or off of the 10 LEDs on the LED Bargraph. We give these 10 LEDs high levels to let they are off at first, then decide how many LEDs are lit up by changing the received analog value.

```
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 produces values (**0-255**) with varying voltage values (**0-5V**), ex., if a 3V is detected on a battery, the corresponding value **152** is displayed on the voltmeter.

The **10** LEDs on the LED Bargraph are used to display the **analogVal** readings. 255/10=25, so every **25** the analog value increases, one more LED turns on, ex., if "analogVal=150 (about 3V), there are 6 LEDs turning on."

Phenomenon Picture

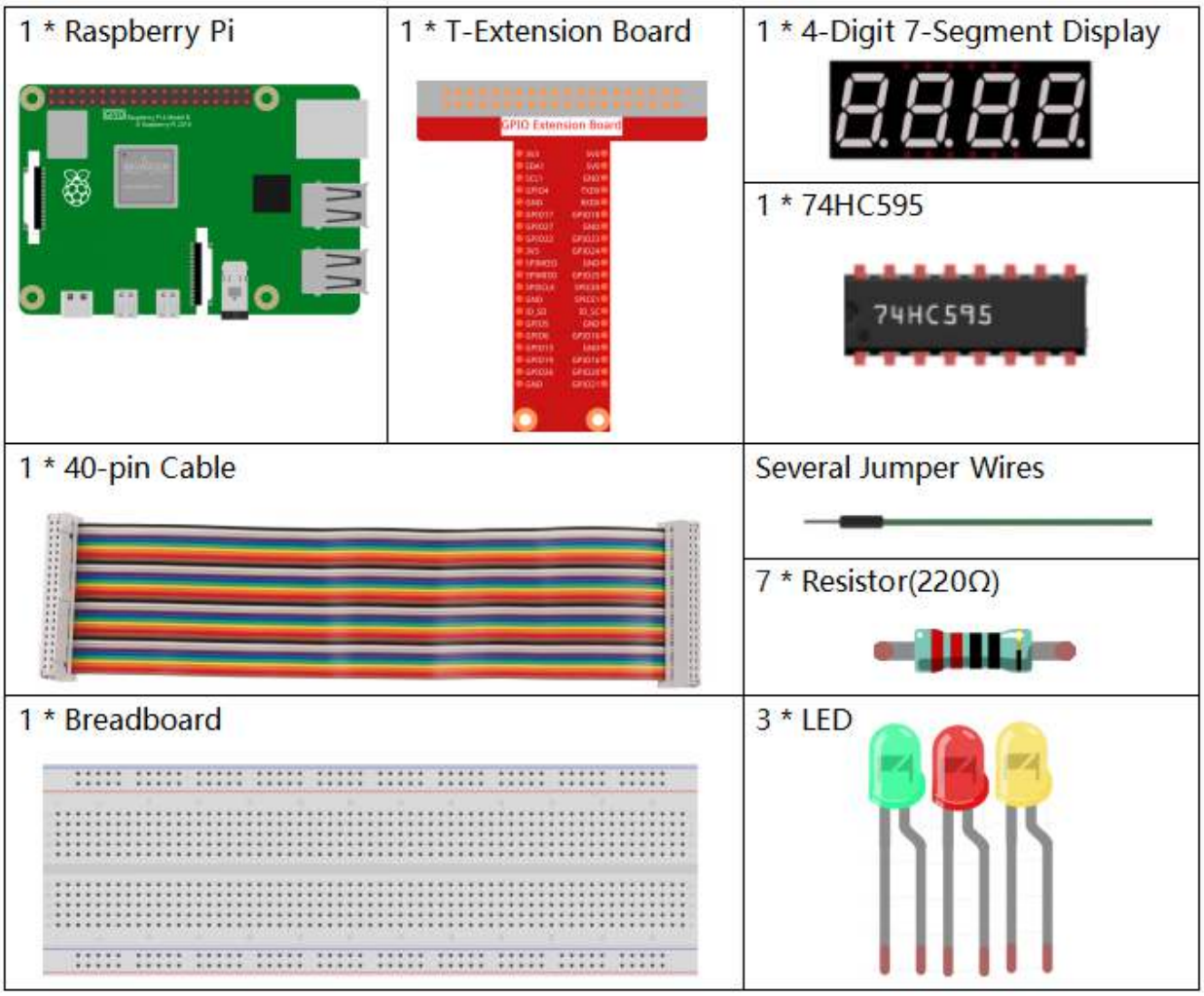


7.4.6 3.1.6 Traffic Light

Introduction

In this project, we will use LED lights of three colors to realize the change of traffic lights and a four-digit 7-segment display will be used to display the timing of each traffic state.

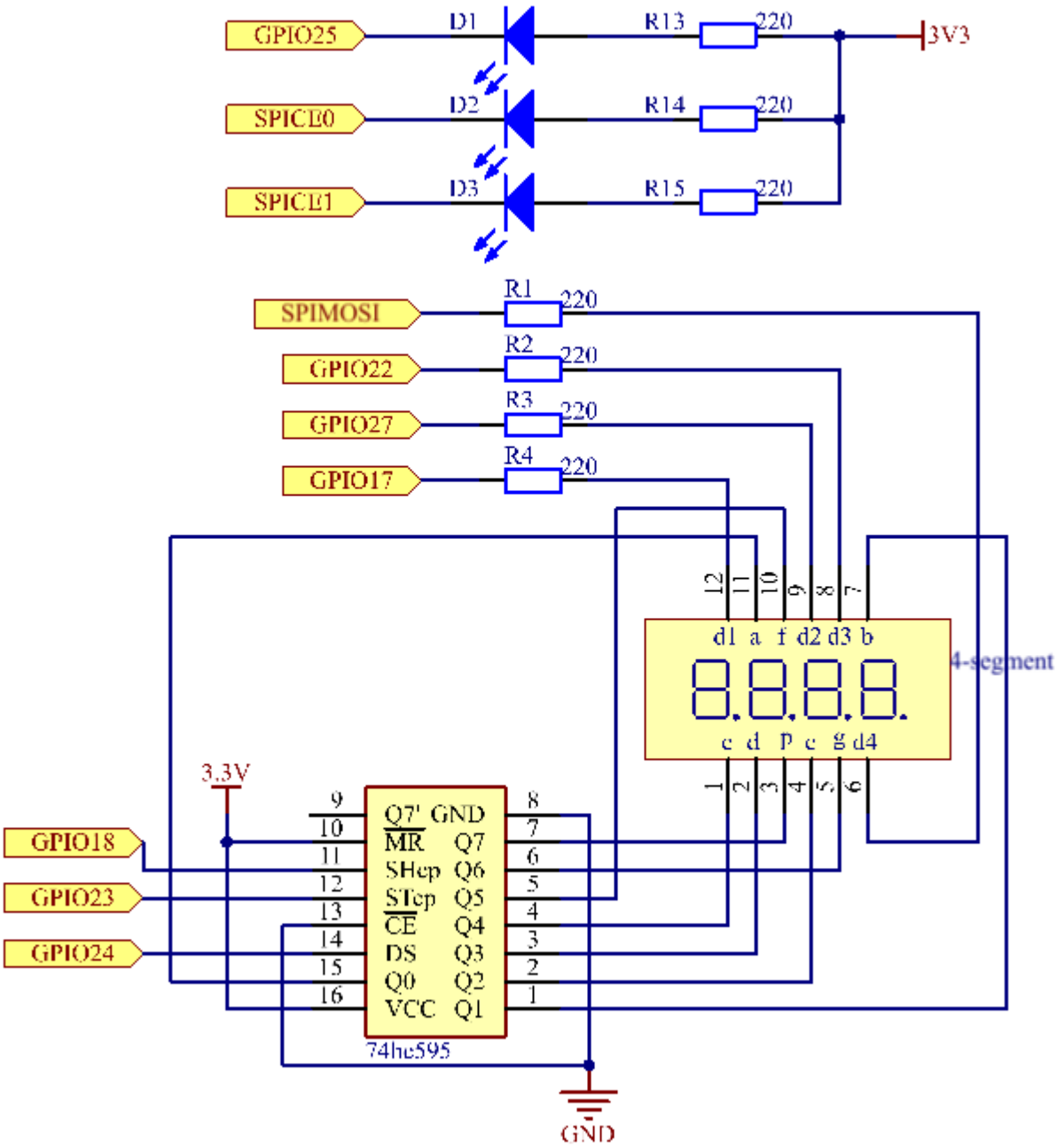
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *4-Digit 7-Segment Display*
- *74HC595*

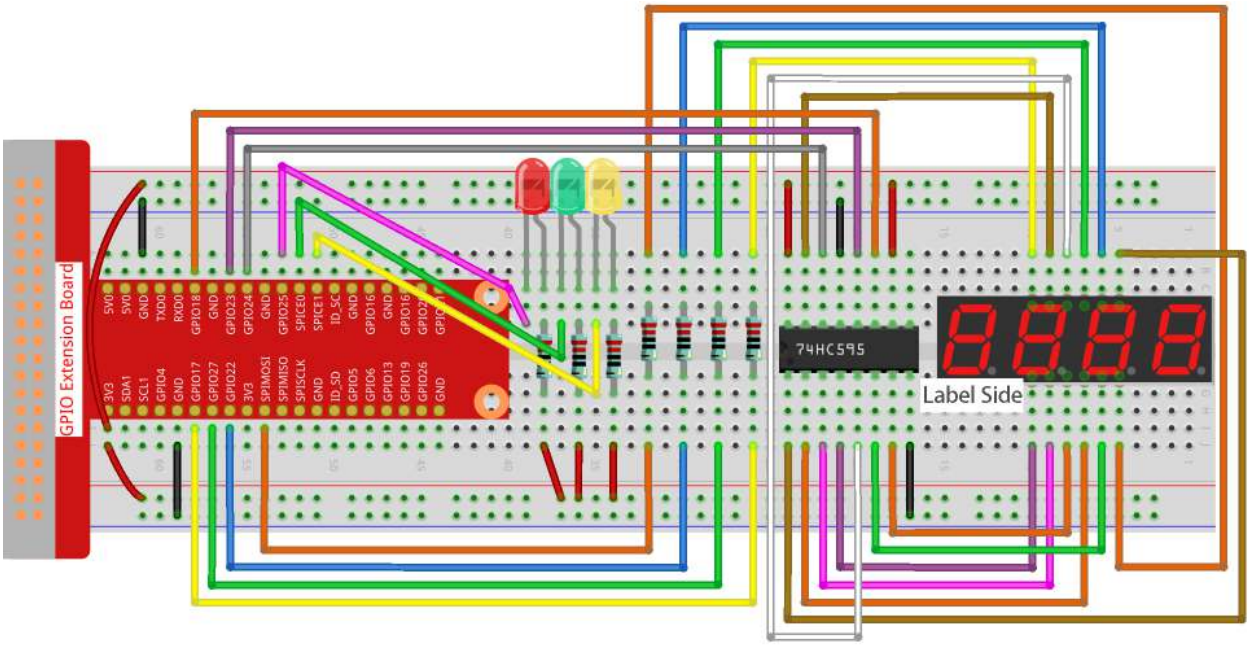
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	Pin 26	11	7


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
4-segment
c d p c g d4
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

Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.6/
```

Step 3: Compile.

```
gcc 3.1.6_TrafficLight.c -lwiringPi
```

Step 4: Run.

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

As the code runs, LEDs will simulate the color changing of traffic lights. Firstly, the red LED lights up for 60s, then the green LED lights up for 30s; next, the yellow LED lights up for 5s. After that, the red LED lights up for 60s once again. In this way, this series of actions will be executed repeatedly.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

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 ledPin[]={6,10,11};
const int placePin[] = {12, 3, 2, 0};
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 60;

void lightup()
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],HIGH);
    }
    digitalWrite(ledPin[colorState],LOW);
}

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;
```

(continues on next page)

(continued from previous page)

```
    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 display()
{
    int a,b,c;

    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]);
    }
}

void loop()
```

(continues on next page)

(continued from previous page)

```
{
    while(1){
    display();
    lightup();
    }
}


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]);
    }
}

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(SDI,OUTPUT);
    pinMode(RCLK,OUTPUT);
    pinMode(SRCLK,OUTPUT);
    for(i=0;i<4;i++){
        pinMode(placePin[i],OUTPUT);
        digitalWrite(placePin[i],HIGH);
    }
    for(i=0;i<3;i++){
        pinMode(ledPin[i],OUTPUT);
        digitalWrite(ledPin[i],HIGH);
    }
    signal(SIGALRM,timer);
    alarm(1);
    loop();
    return 0;
}
```

Code Explanation

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

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

void pickDigit(int digit);
void hc595_shift(int8_t data);
void clearDisplay();
void display();
```

These codes are used to realize the function of number display of 4-Digit 7-Segment Displays. Refer to chapter 1.1.5 of the document for more details. Here, we use the codes to display countdown of traffic light time.

```
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);
}
```

The codes are used to switch the LED on and off.

```
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]);
    }
}
```

The codes are used to switch the timer on and off. Refer to chapter 1.1.5 for more details. Here, when the timer returns to zero, colorState will be switched so as to switch LED, and the timer will be assigned to a new value.

```
void loop()
{
```

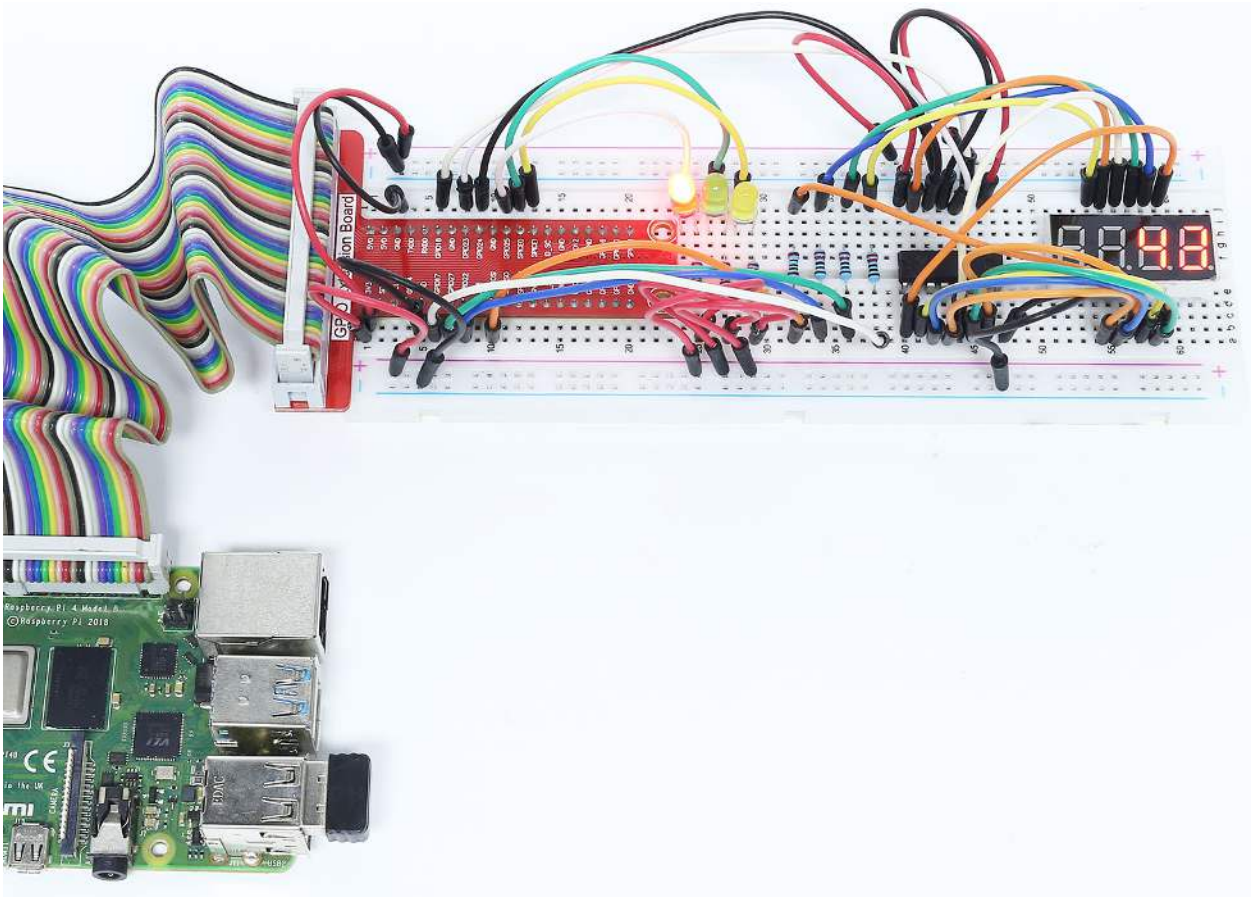
(continues on next page)

(continued from previous page)

```
    while(1){
    display();
    lightup();
    }
}

int main(void)
{
    //...
    signal(SIGALRM,timer);
    alarm(1);
    loop();
    return 0;
}
```

The timer is started in the main() function. In loop() function, use **while(1)** loop and call the functions of 4-Digit 7-Segment and LED.

Phenomenon Picture

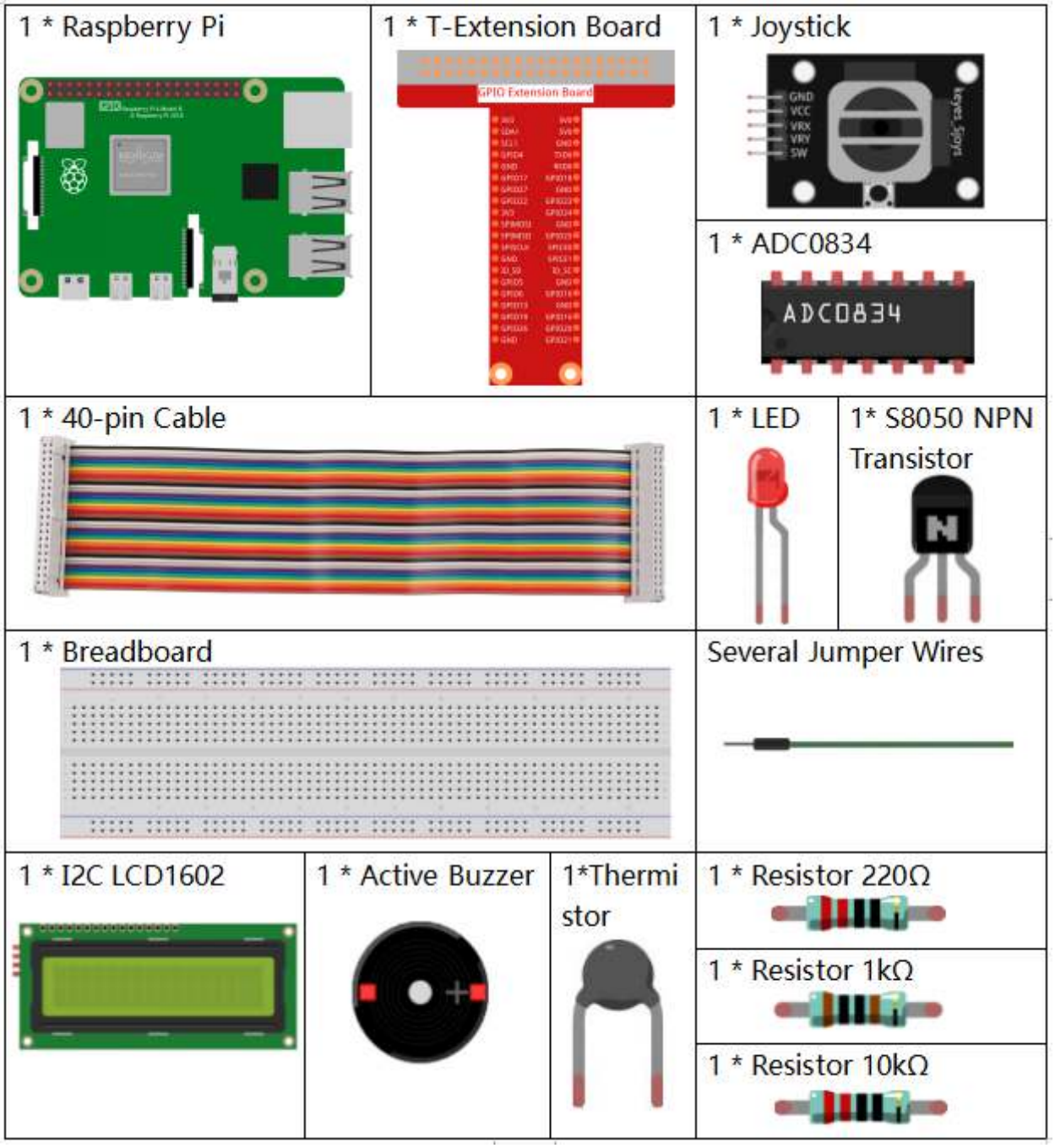


7.4.7 3.1.7 Overheat Monitor

Introduction

You may want to make an overheat monitoring device that applies to various situations, ex., in the factory, if we want to have an alarm and the timely automatic turning off of the machine when there is a circuit overheating. In this project, we will use thermistor, joystick, buzzer, LED and LCD to make an smart temperature monitoring device whose threshold is adjustable.

Components

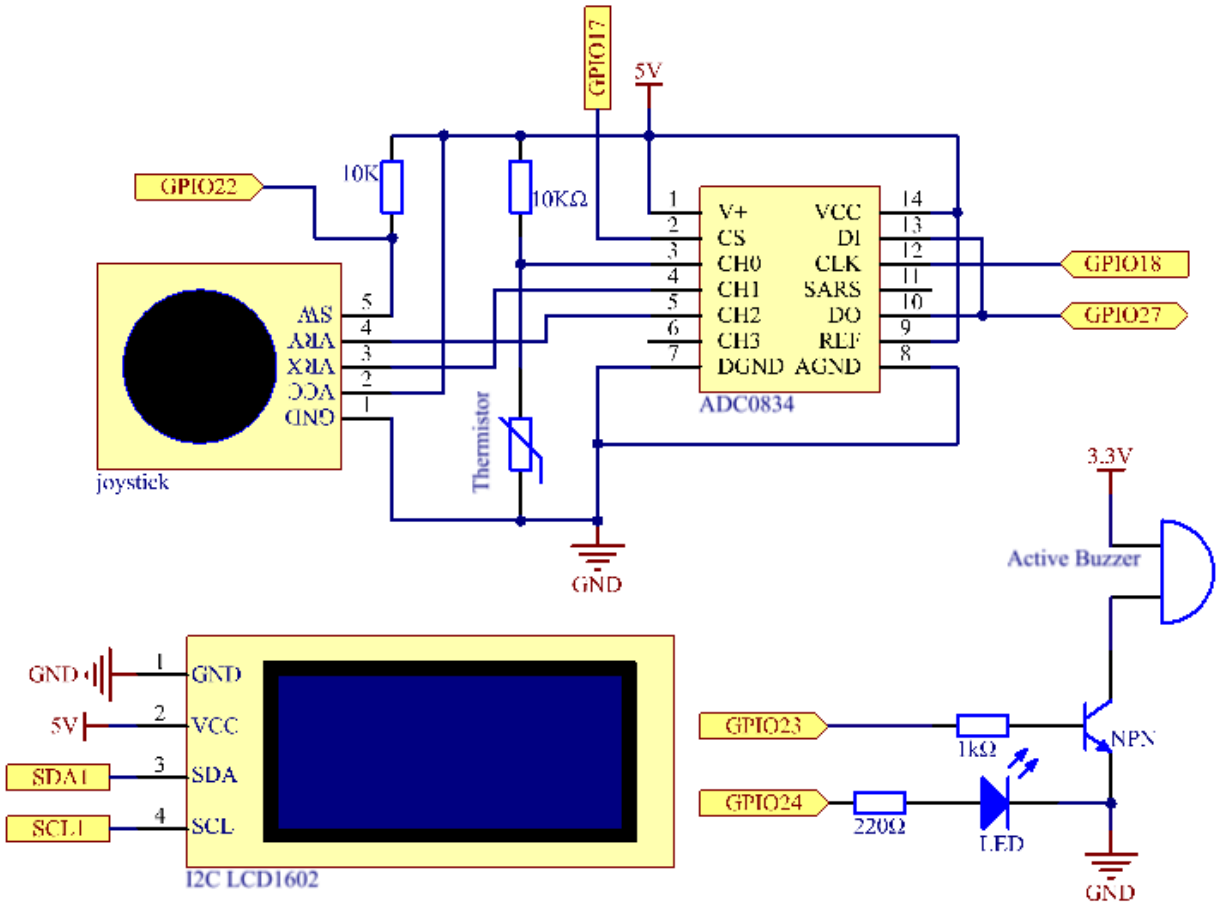


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Joystick Module*
- *Thermistor*

- *ADC0834*
- *Transistor*

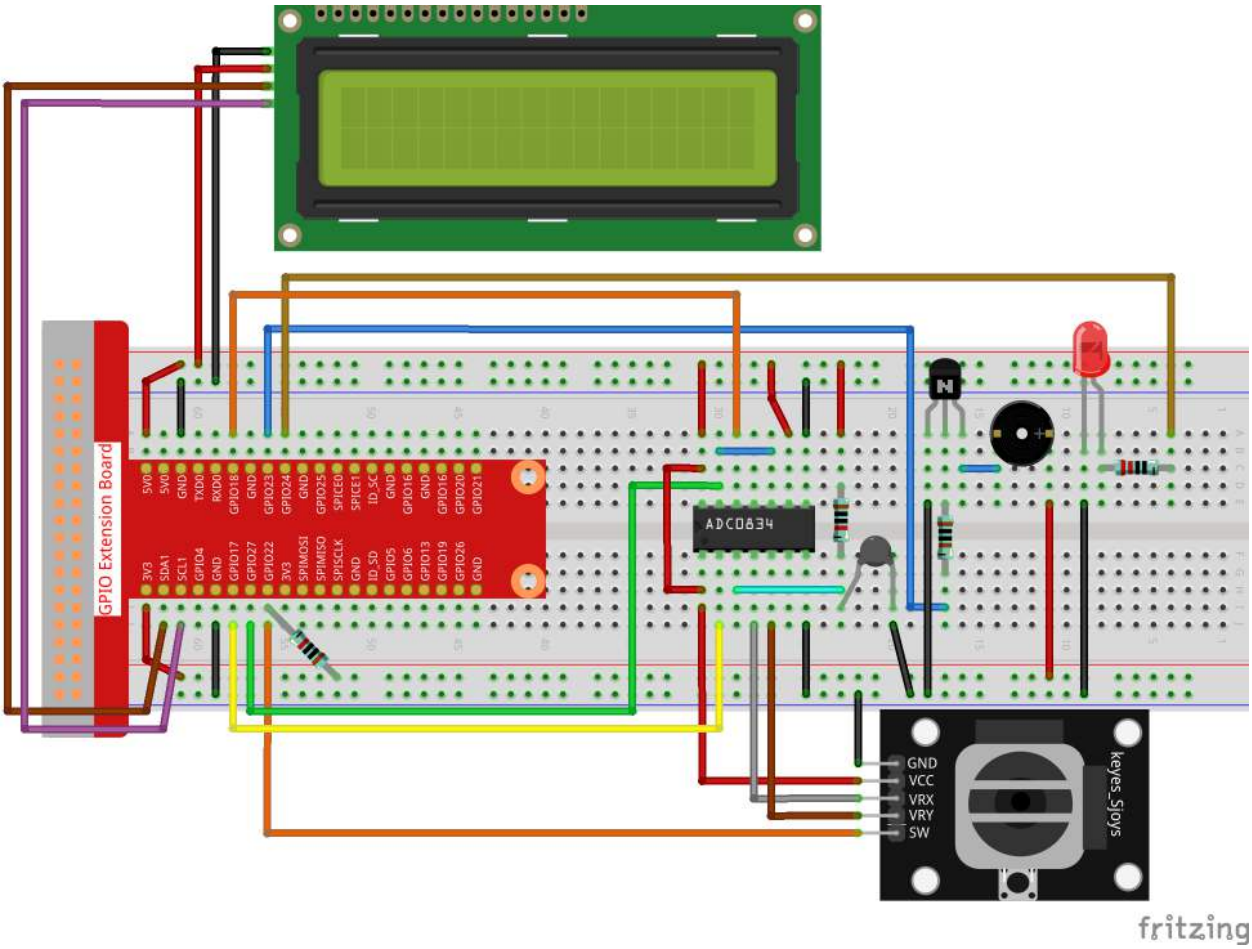
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22
GPIO23	Pin16	4	23
GPIO24	Pin18	5	24
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.7/
```

Step 3: Compile the code.

```
gcc 3.1.7_OverheatMonitor.c -lm -lwiringPi
```

Step 4: Run the executable file.

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

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

Joystick here is for your pressing to adjust the high-temperature threshold. Toggling the **Joystick** in the direction of X-axis and Y-axis can adjust (turn up or down) the current high-temperature threshold. Press the **Joystick** once again to reset the threshold to initial value.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please

refer to *C code is not working?*.

Code Explanation

```
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;
    }
}
```

This function reads values of X and Y. If `X>200`, there will return `1`; `X<50`, return `-1`; `y>200`, return `-10`, and `y<50`, return `10`.

```
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;
}
```

Read the analog value of the **CH0** (thermistor) of **ADC0834** and then convert it to temperature value.

```
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);
    }
}
```

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

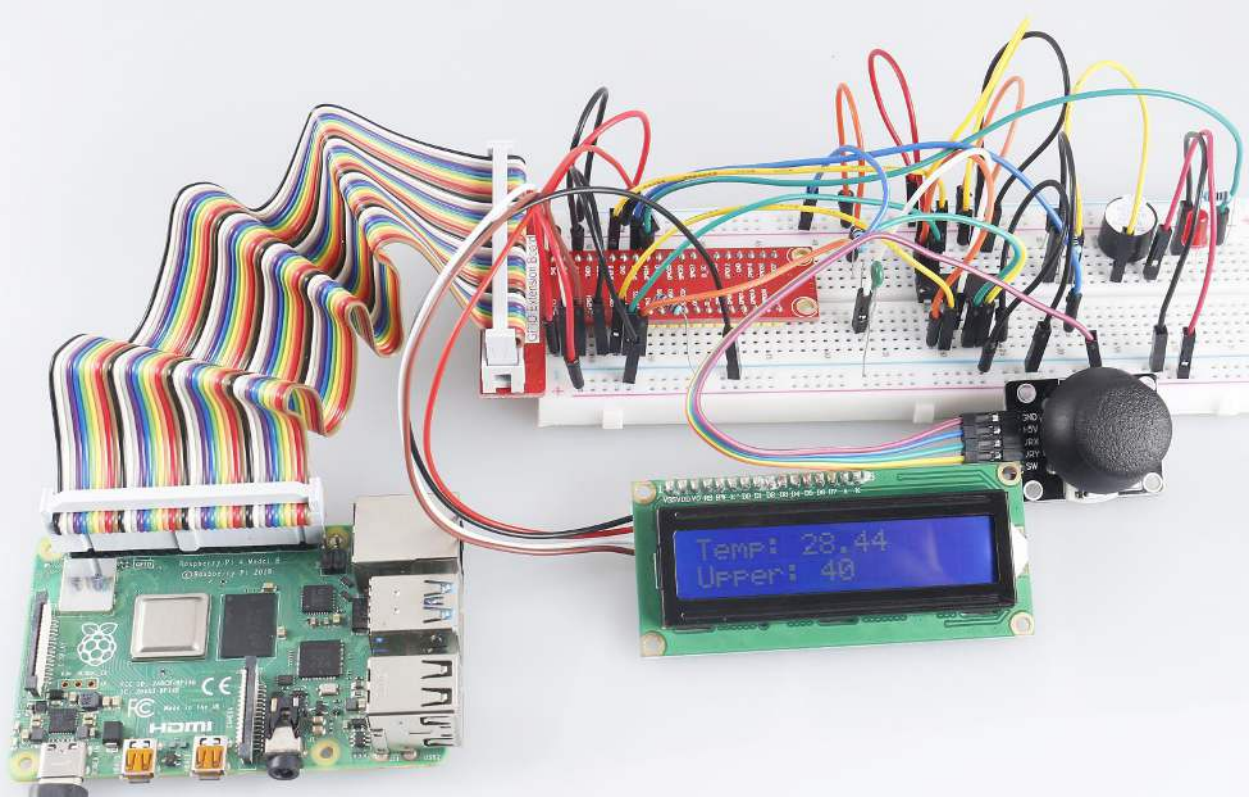
```
int main(void)
{
    setup();
    int lastState =1;
    int stage=0;
    while (1)
    {
        int currentState = digitalRead(Joy_BtnPin);
        if(currentState==1 && lastState == 0){
            stage=(stage+1)%2;
            delay(100);
            lcd_clear();
        }
        lastState=currentState;
        if (stage==1){
            upper_tem_setting();
        }
        else{
            monitoring_temp();
        }
    }
    return 0;
}
```

The function `main()` contains the whole program process as shown:

1) When the program starts, the initial value of **stage** is **0**, and the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and the LED are started to alarm you.

2) Press the Joystick, and **stage** will be **1** and you can adjust the high-temperature threshold. Toggling the Joystick in the direction of X-axis and Y-axis can adjust (turn up or down) the current threshold. Press the Joystick once again to reset the threshold to initial value.

Phenomenon Picture



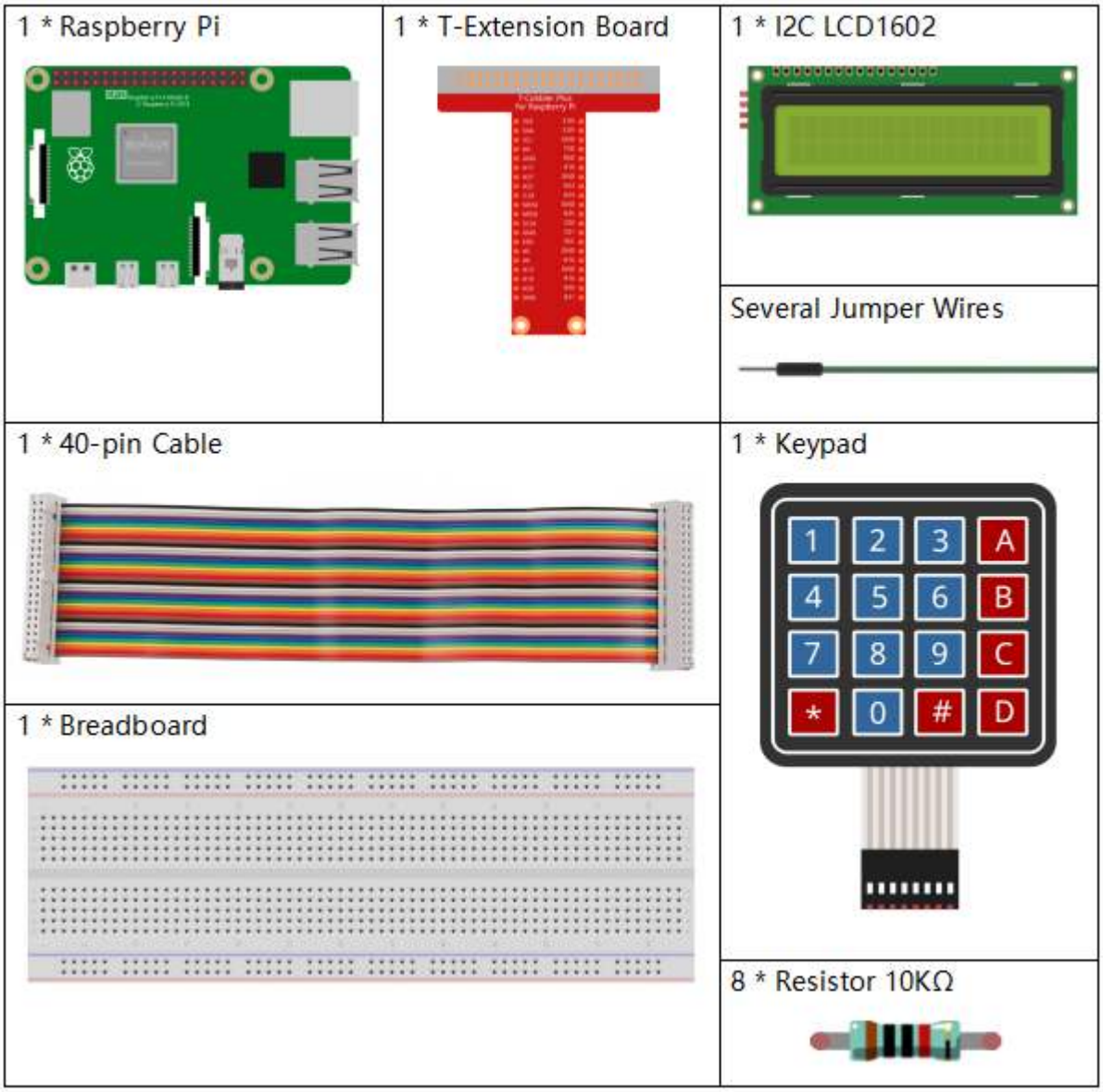
7.4.8 3.1.8 Password Lock

Introduction

In this project, we will use a keypad and a LCD to make a combination lock. The LCD will display a corresponding prompt for you to type your password on the Keypad. If the password is input correctly, “Correct” will be displayed.

On the basis of this project, we can add additional electronic components, such as buzzer, LED and so on, to add different experimental phenomena for password input.

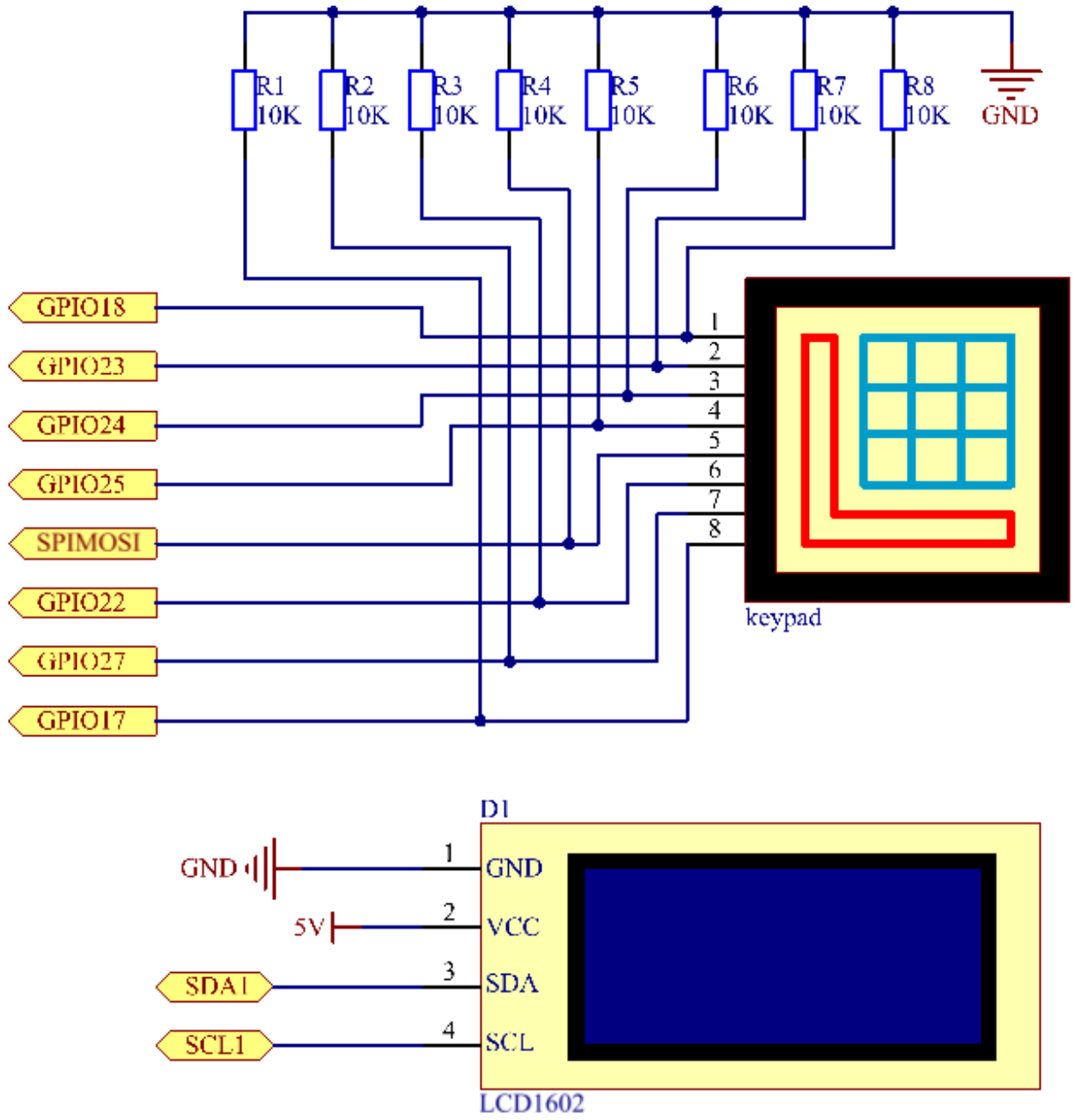
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *I2C LCD1602*
- *Keypad*

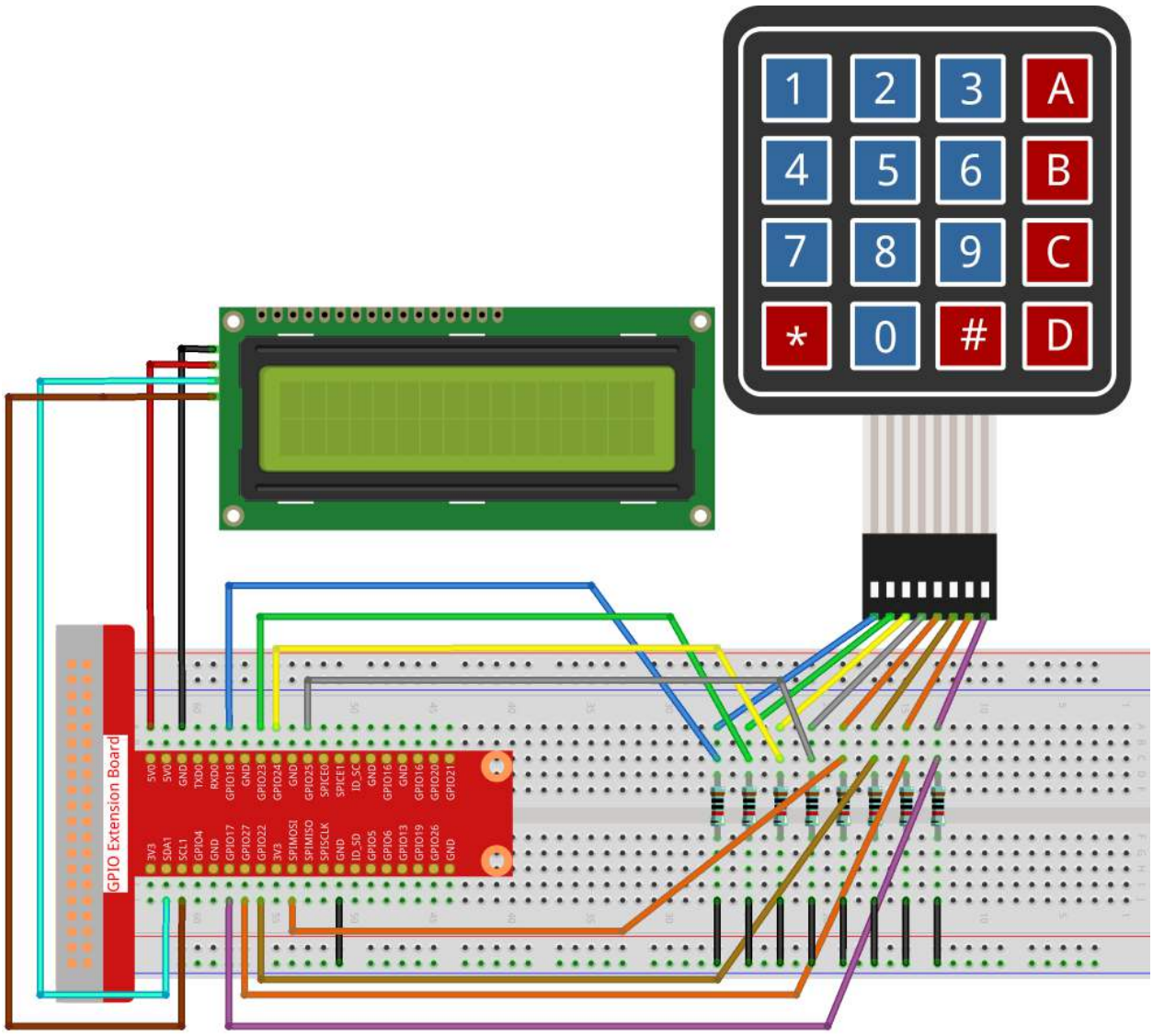
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		



Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.8/
```

Step 3: Compile.

```
gcc 3.1.8_PasswordLock.cpp -lwiringPi
```

Step 4: Run.

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

After the code runs, use the keypad to enter the correct password: 1984. If the "CORRECT" appears on LCD1602, there is no wrong with the password; otherwise, "WRONG KEY" will appear.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
#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'};
```

Here, we define the length of the password LENS, storage matrix keyboard key value array KEYS and the array that stores the correct password.

```
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);
```

There is a declaration of the subfunctions of the matrix keyboard code, refer to *2.1.8 Keypad* of this document for more 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[]);
```

There is a declaration of the subfunctions of LCD1062 code, refer to *1.1.7 I2C LCD1602* of this document for more 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");
                        }
                ...
```

Read the key value and store it in the test array testword. If the number of stored key values is more than 4, the correctness of the password is automatically verified, and the verification results are displayed on the LCD interface.

```
int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
```

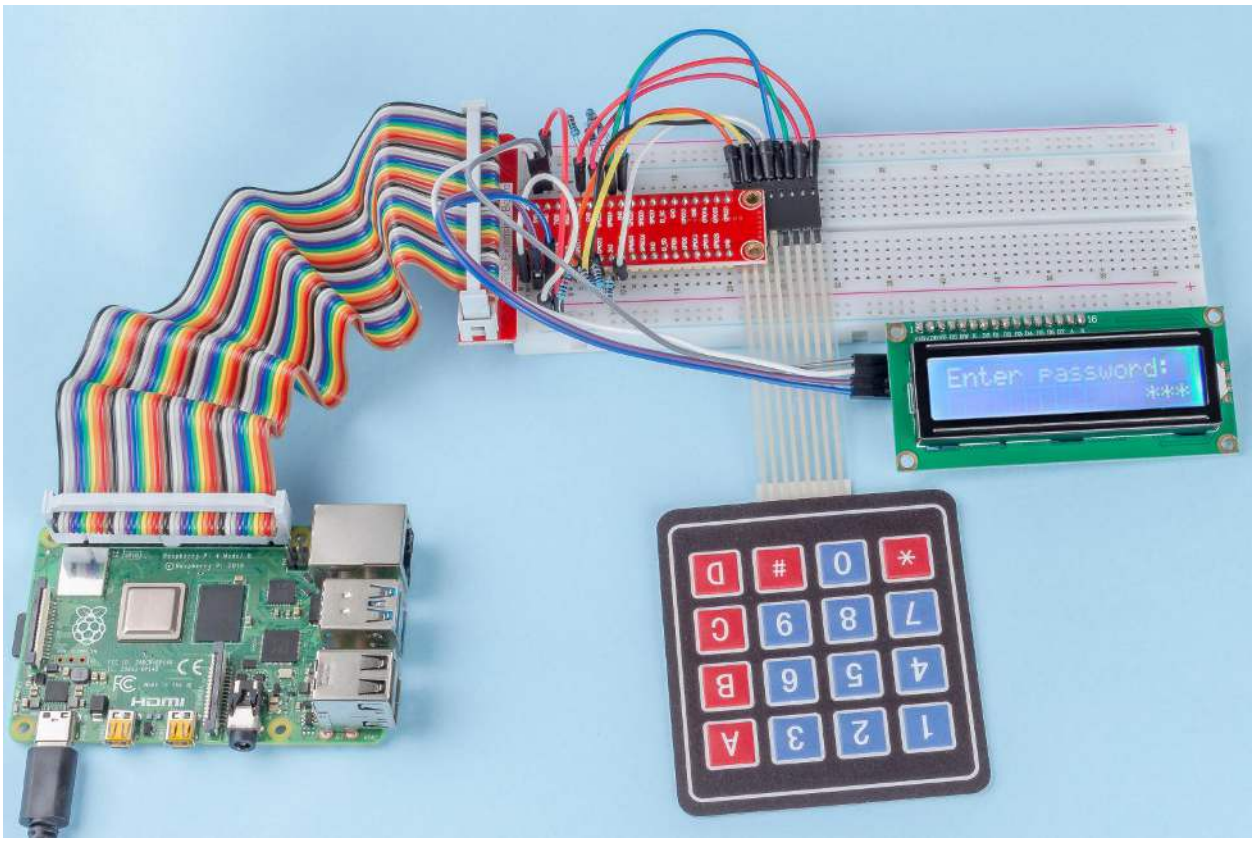
(continues on next page)

(continued from previous page)

```
            {return 0;}
        }
        return 1;
}
```

Verify the correctness of the password. Return 1 if the password is entered correctly, and 0 if not.

Phenomenon Picture

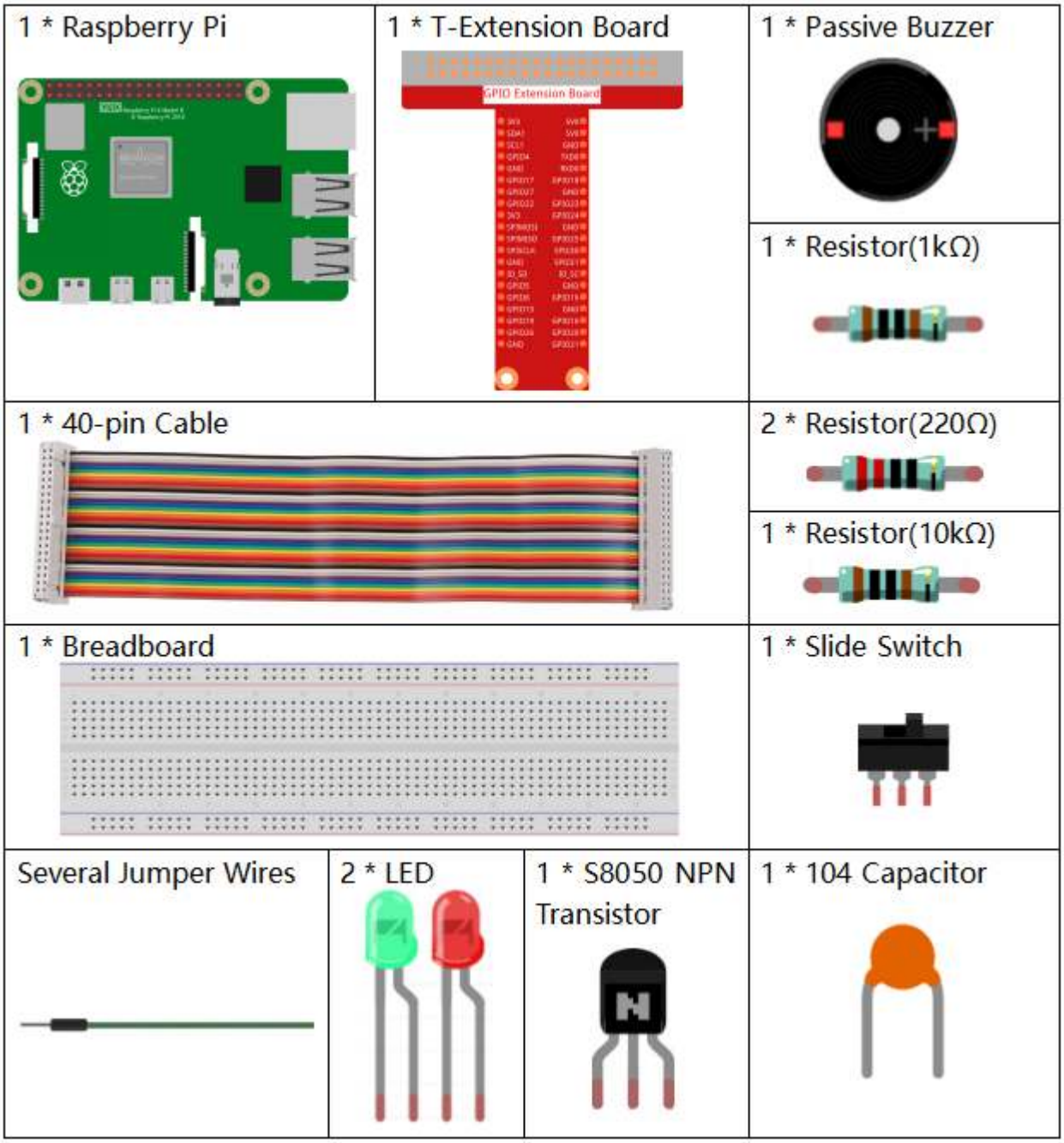


7.4.9 3.1.9 Alarm Bell

Introduction

In this project, we will make a manual alarm device. You can replace the toggle switch with a thermistor or a photosensitive sensor to make a temperature alarm or a light alarm.

Components

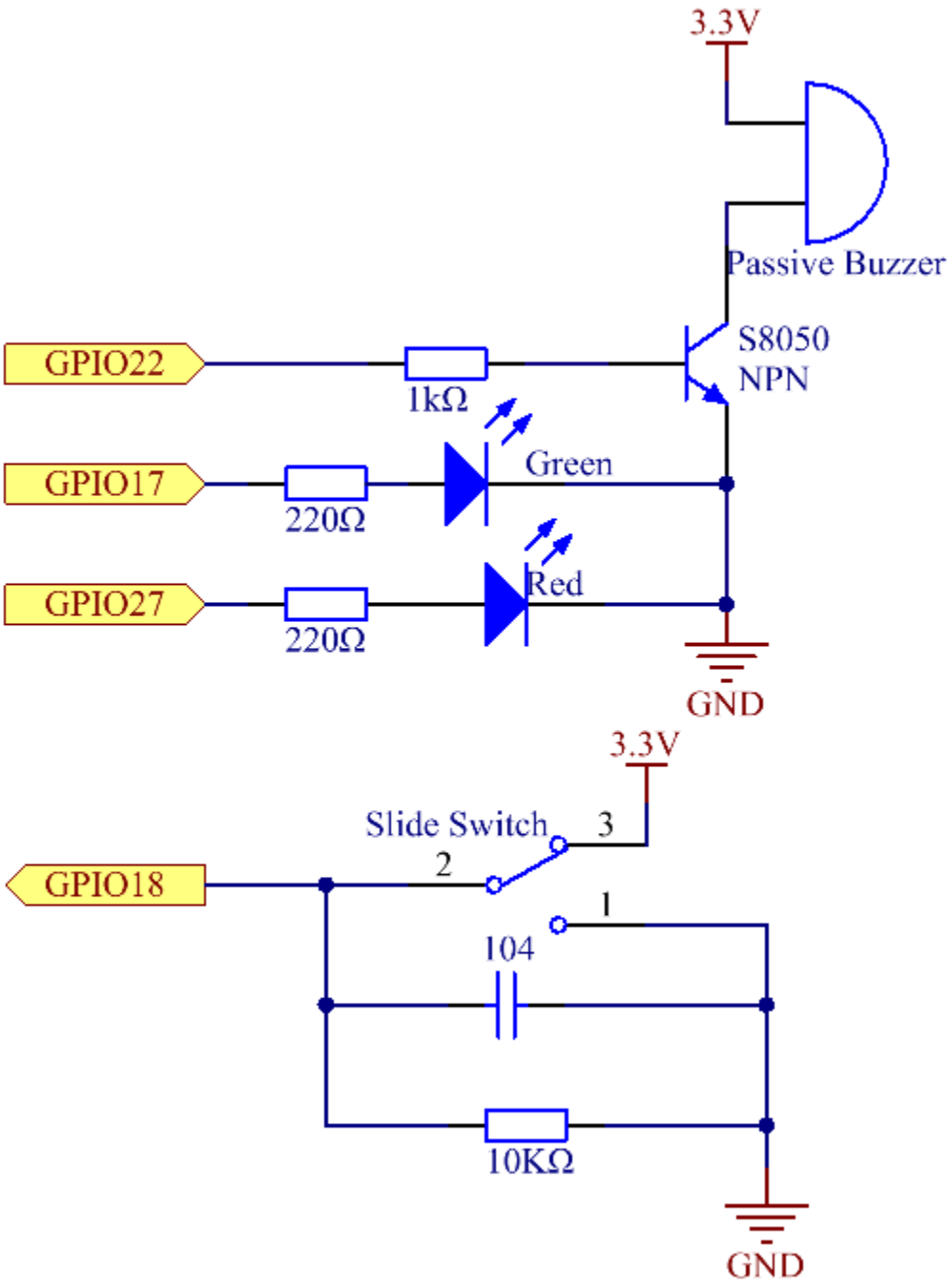


- *GPIO Extension Board*

- *Breadboard*
- *Resistor*
- *LED*
- *Buzzer*
- *Slide Switch*
- *Transistor*
- *Capacitor*

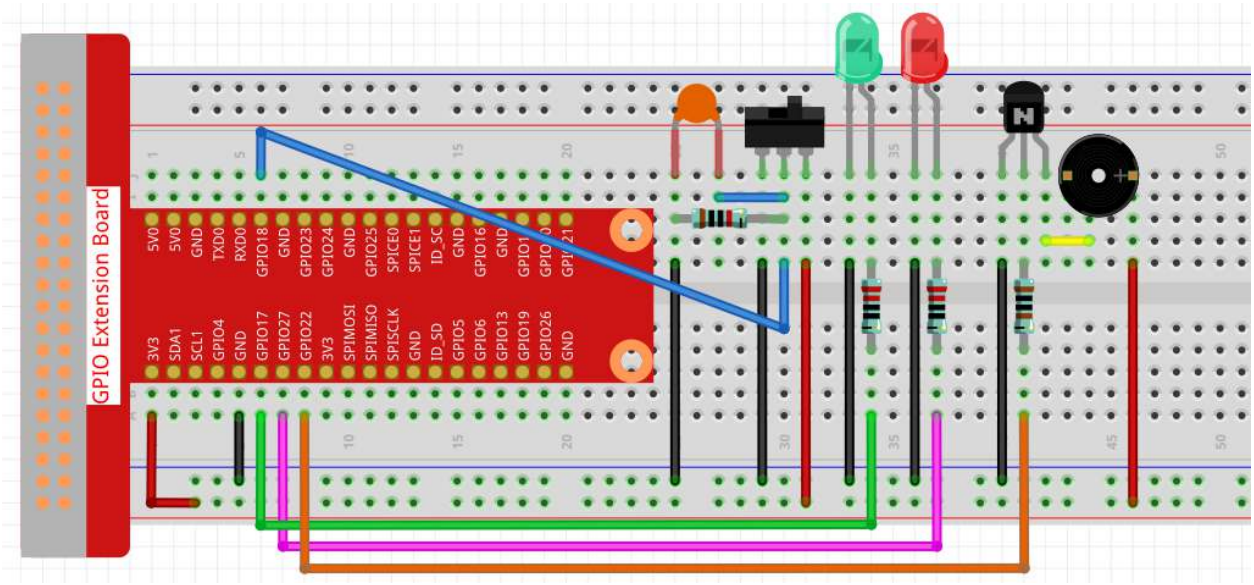
Schematic Diagram

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


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

Experimental Procedures

Step 1: Build the circuit.



Step 2: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.9/
```

Step 3: Compile.

```
gcc 3.1.9_AlarmBell.c -lwiringPi -lpthread
```

Step 4: Run.

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

After the program starts, put the slide switch to the right, and the buzzer will give out alarm sounds. At the same time, the red and green LEDs will flash at a certain frequency.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
#include <pthread.h>
```

In this code, you'll use a new library, `pthread.h`, which is a set of common thread libraries and can realize multithreading. We add the `-lpthread` parameter at compile time for the independent working of the LED and the buzzer.

```
void *ledWork(void *arg){
    while(1)
    {
        if(flag==0){
            pthread_exit(NULL);
```

(continues on next page)

(continued from previous page)

```
        }
        digitalWrite(ALedPin,HIGH);
        delay(500);
        digitalWrite(ALedPin,LOW);
        digitalWrite(BLedPin,HIGH);
        delay(500);
        digitalWrite(BLedPin,LOW);
    }
}
```

The function `ledWork()` helps to set the working state of these 2 LEDs: it keeps the green LED lighting up for 0.5s and then turns off; similarly, keeps the red LED lighting up for 0.5s and then turns off.

```
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);
    }
}
```

The function `buzzWork()` is used to set the working state of the buzzer. Here we set the frequency as between 130 and 800, to accumulate or decay at an interval of 20.

```
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 the function on():

1) Define the mark `flag=1`, indicating the ending of the control thread.
2) Create a software-controlled tone pin `BeepPin`.
3) Create two separate threads so that the LED and the buzzer can work at the same time.

- `pthread_t tLed`: Declare a thread `tLed`.
- `pthread_create(&tLed,NULL,ledWork,NULL)`: Create the thread and its prototype is as follows:

```
int pthread_create(pthread_t *restrict tidp,const pthread_attr_t*restrict_attr,
↪void**start_rtn)(void*),void *restrict arg);
```

If successful, return 0 otherwise, return the fall number -1.

- The first parameter is a pointer to the thread identifier.
- The second one is used to set the thread attribute.
- The third one is the starting address of the thread running function.
- The last one is the one that runs the function.

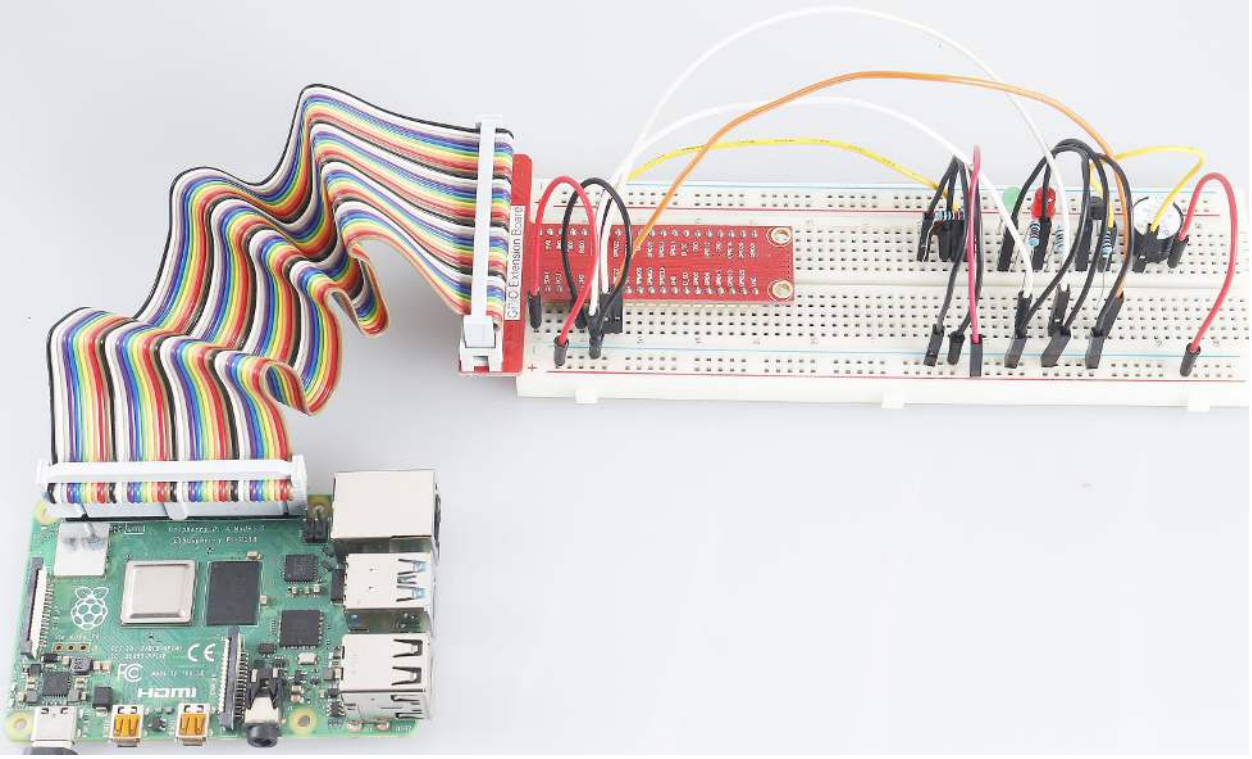
```
void off(){
    flag = 0;
    softToneStop(BeepPin);
    digitalWrite(ALedPin,LOW);
    digitalWrite(BLedPin,LOW);
}
```

The function `Off()` defines "flag=0" so as to exit the threads **ledWork** and **BuzzWork** and then turn off the buzzer and the LED.

```
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() contains the whole process of the program: firstly read the value of the slide switch; if the toggle switch is toggled to the right (the reading is 1), the function `on()` is called, the buzzer is driven to emit sounds and the the red and the green LEDs blink. Otherwise, the buzzer and the LED don't work.

Phenomenon Picture

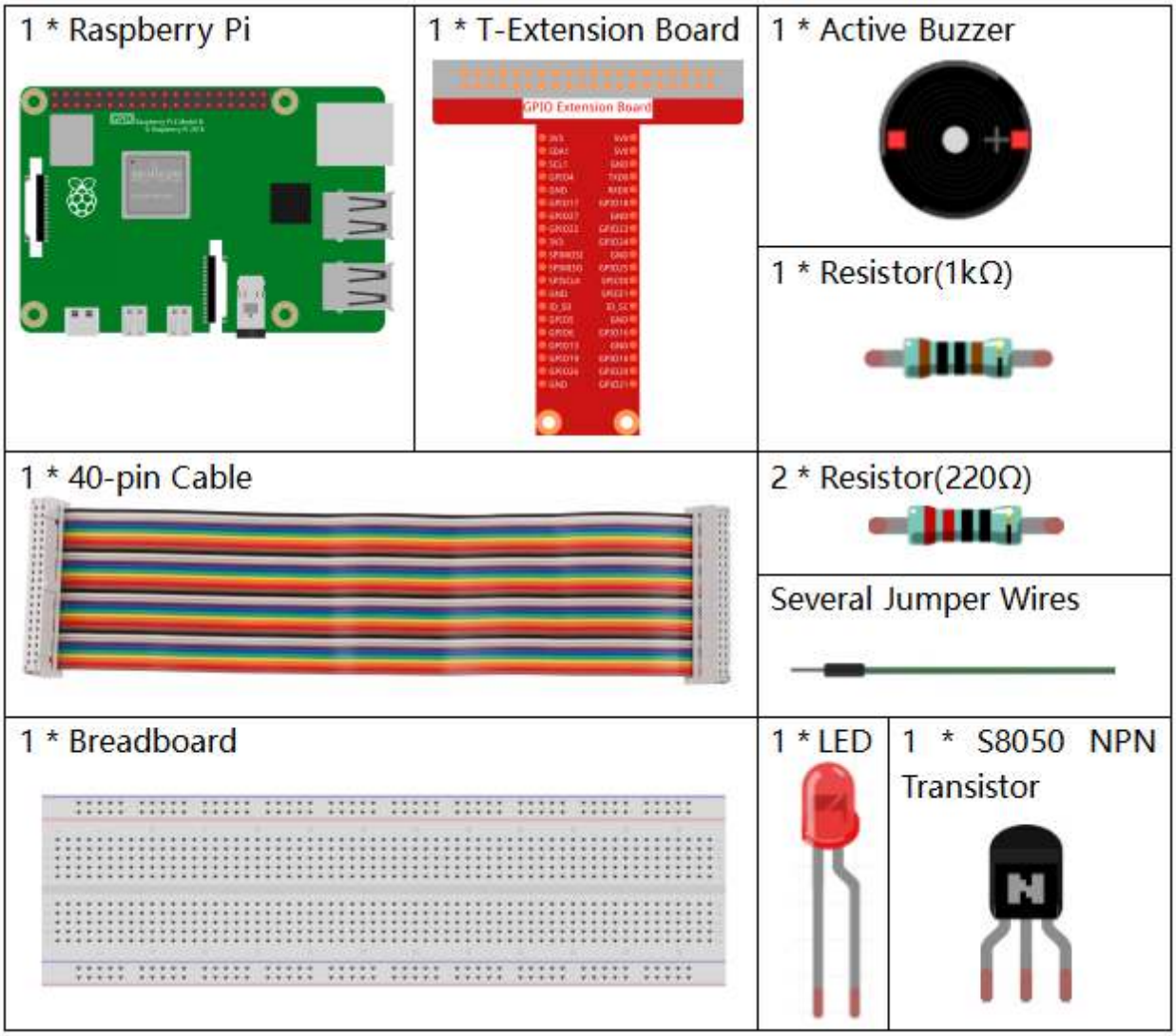


7.4.10 3.1.10 Morse Code Generator

Introduction

In this project, we'll make a Morse code generator, where you type in a series of English letters in the Raspberry Pi to make it appear as Morse code.

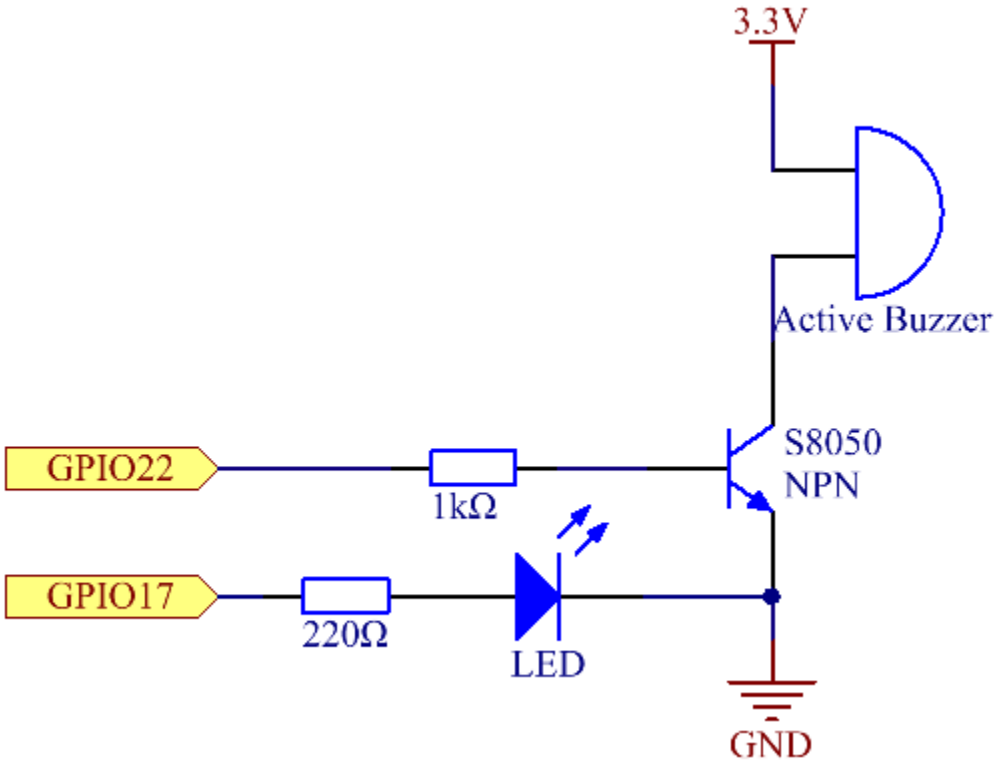
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Buzzer*
- *Transistor*

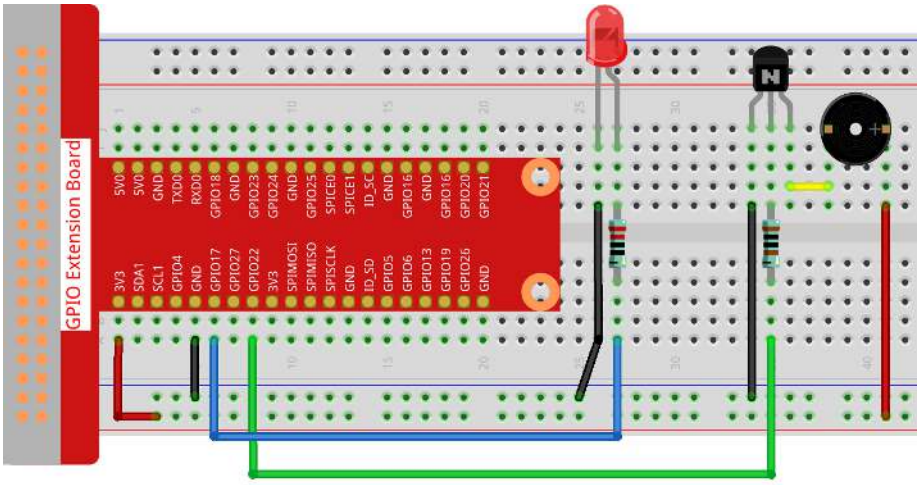
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit. (Pay attention to poles of the buzzer: The one with + label is the positive pole and the other is the negative.)



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/c/3.1.10/
```

Step 3: Compile the code.

```
gcc 3.1.10_MorseCodeGenerator.c -lwiringPi
```

Step 4: Run the executable file above.

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

After the program runs, type a series of characters, and the buzzer and the LED will send the corresponding Morse code signals.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
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"}
};
```

This structure MORSE is the dictionary of the Morse code, containing characters A-Z, numbers 0-9 and marks "?" "/" ":" "," "." ";" "!" "@" .

```
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;
        }
    }
}
```

The function `lookup()` works by "checking the dictionary". Define a `key` search the same words as `key` in the structure `morseDict` and return the corresponding information— `code` of the certain word.

```
void on(){
    digitalWrite(ALedPin,HIGH);
```

(continues on next page)

(continued from previous page)

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

Create a function `on()` to start the buzzer and the LED.

```
void off(){
    digitalWrite(ALedPin,LOW);
    digitalWrite(BeepPin,LOW);
}
```

The function `off()` turns off the buzzer and the LED.

```
void beep(int dt){
    on();
    delay(dt);
    off();
    delay(dt);
}
```

Define a function `beep()` to make the buzzer and the LED emit sounds and blink in a certain interval of **dt**.

```
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);
        }
    }
}
```

The function `morsecode()` is used to process the Morse code of input characters by making the "1" of the code keep emitting sounds or lights and the "0"shortly emit sounds or lights, ex., input "SOS", and there will be a signal containing three short three long and then three short segments "· · · — — — · · · ".

```
int toupper(int c)
{
    if ((c >= 'a') && (c <= 'z'))
        return c + ('A' - 'a');
    return c;
}
char *strupr(char *str)
{
    char *orign=str;
    for (; *str!='\0'; str++)
        *str = toupper(*str);
```

(continues on next page)

(continued from previous page)

```
return orign;
}
```

Before coding, you need to unify the letters into capital letters.

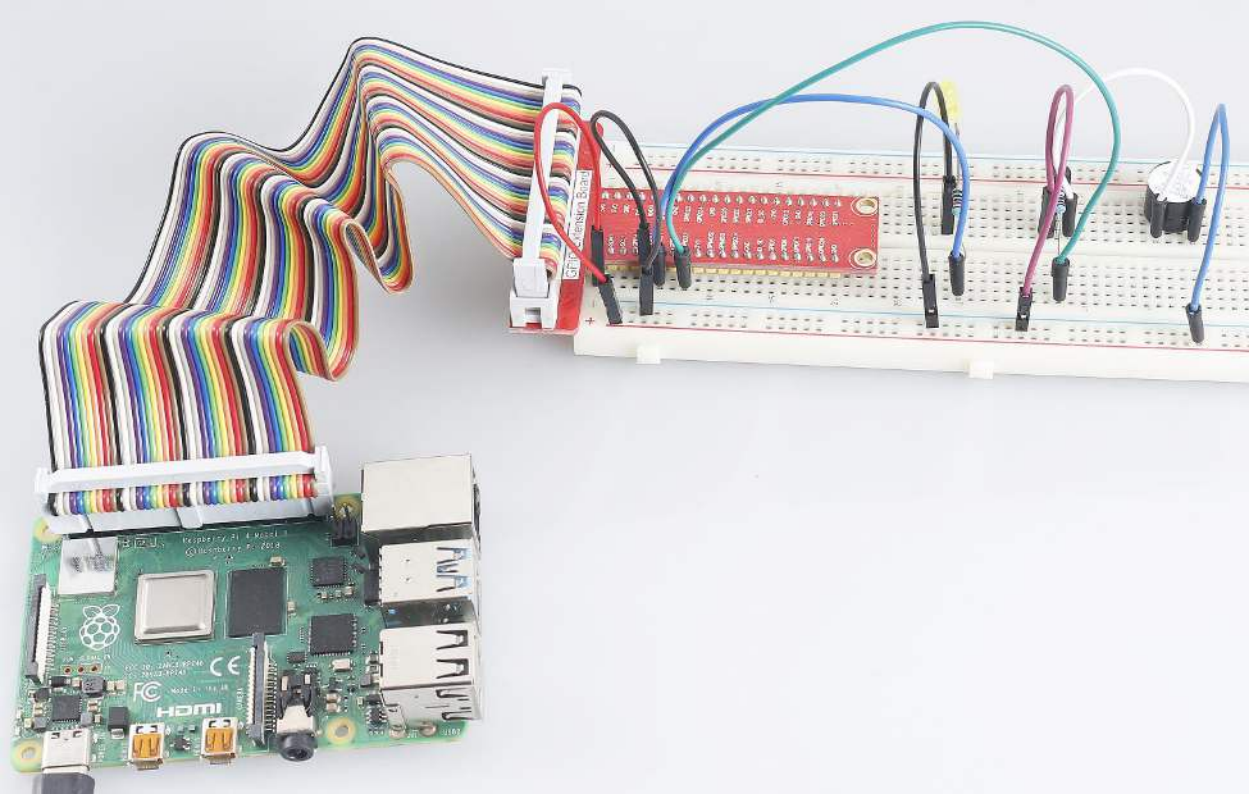
```
void main(){
    setup();
    char *code;
    int length=8;
    code = (char*)malloc(sizeof(char)*length);
    while (1){
        printf("Please input the messenger:");
        scanf("%s",code);
        code=strupr(code);
        printf("%s\n",code);
        morsecode(code);
    }
}
```

When you type the relevant characters with the keyboard, `code=strupr(code)` will convert the input letters to their capital form.

`Printf()` then prints the clear text on the computer screen, and the `morsecod()` function causes the buzzer and the LED to emit Morse code.

Note that the length of the input character mustn't exceed the **length** (can be revised).

Phenomenon Picture

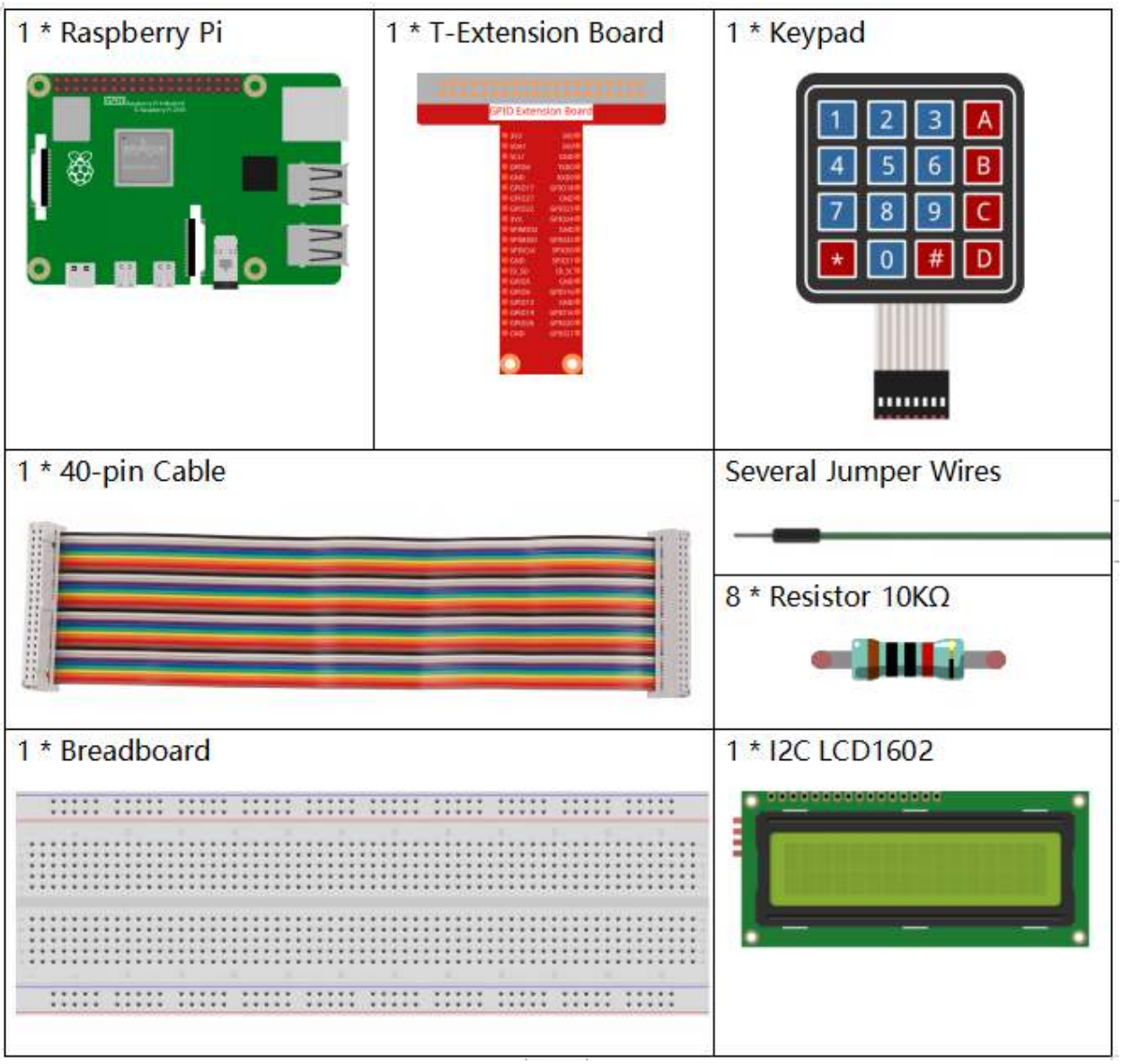


7.4.11 3.1.11 GAME– Guess Number

Introduction

Guessing Numbers is a fun party game where you and your friends take turns inputting a number (0~99). The range will be smaller with the inputting of the number till a player answers the riddle correctly. Then the player is defeated and punished. For example, if the lucky number is 51 which the players cannot see, and the player inputs 50, the prompt of number range changes to 50~99; if the player inputs 70, the range of number can be 50~70; if the player inputs 51, this player is the unlucky one. Here, we use keypad to input numbers and use LCD to output outcomes.

Components

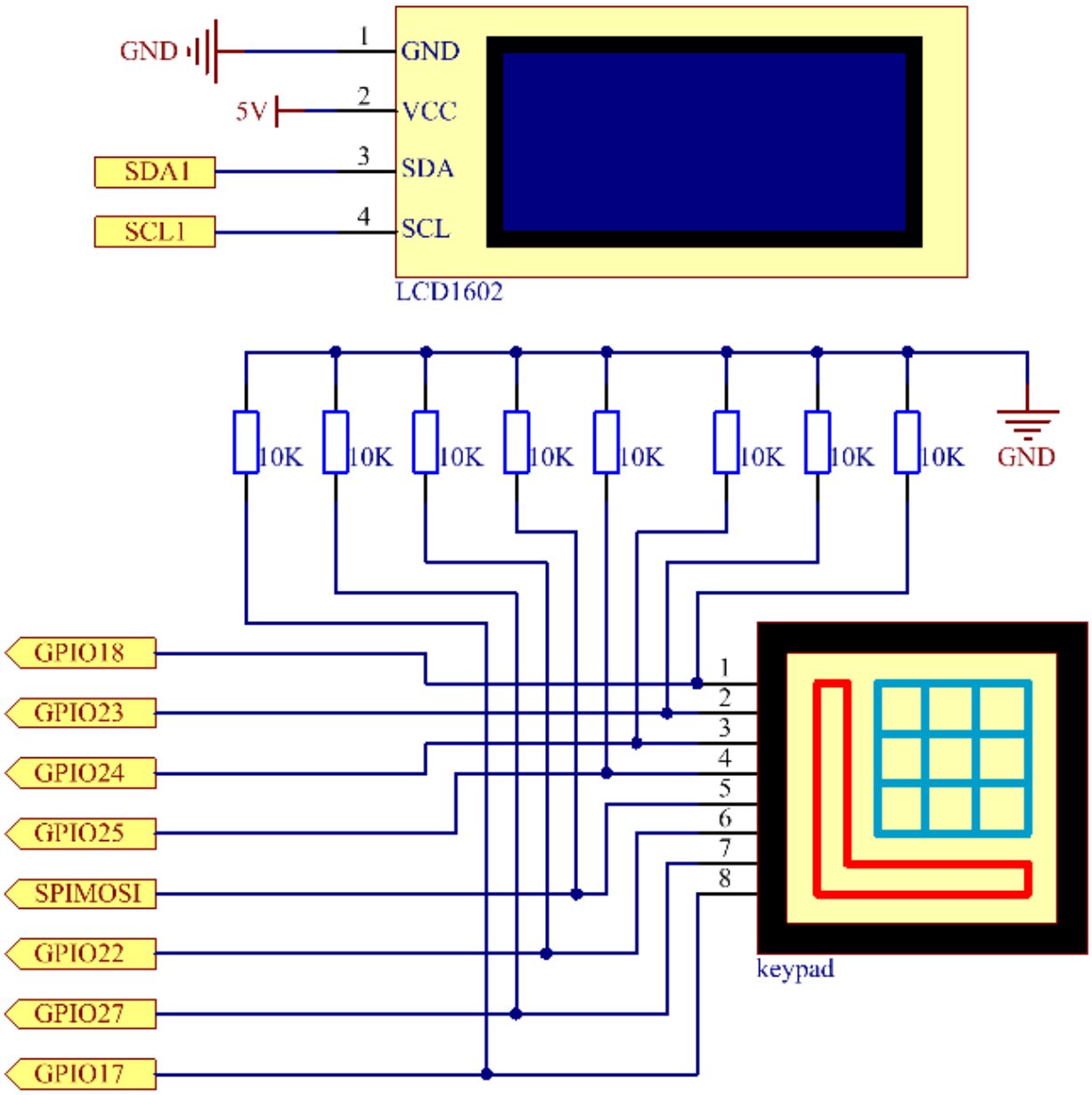


- *GPIO Extension Board*
- *Breadboard*

- *Resistor*
- *Keypad*
- *I2C LCD1602*

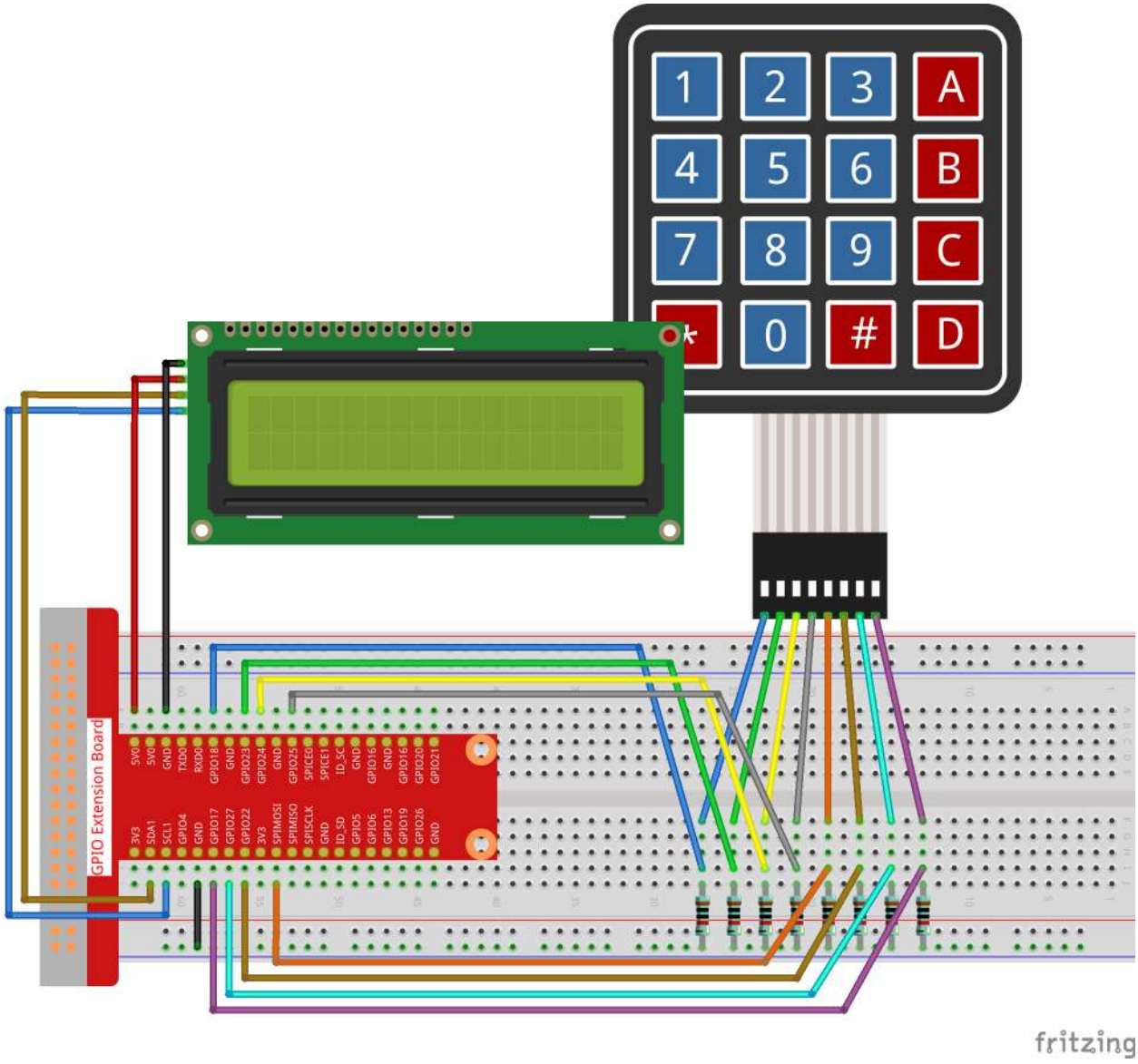
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17
SDA1	Pin 3	SDA1(8)	SDA1(2)
SCL1	Pin 5	SCL1(9)	SDA1(3)



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see Appendix *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Change directory.

```
cd /home/pi/raphael-kit/c/3.1.11/
```

Step 4: Compile.

```
gcc 3.1.11_GAME_GuessNumber.c -lwiringPi
```

Step 5: Run.

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

After the program runs, there displays the initial page on the LCD:

Note:

- If there is an error prompt `wiringPi.h: No such file or directory`, please refer to *C code is not working?*.
- If you get `Unable to open I2C device: No such file or directory` error, you need to refer to *I2C Configuration* to enable I2C and check if the wiring is correct.

```
Welcome!
Press A to go!
```

Press 'A', and the game will start and the game page will appear on the LCD.

```
Enter number:
0 <point< 99
```

A random number '**point**' is produced but not displayed on the LCD when the game starts, and what you need to do is to guess it. The number you have typed appears at the end of the first line till the final calculation is finished. (Press 'D' to start the comparation, and if the input number is larger than **10**, the automatic comparation will start.)

The number range of 'point' is displayed on the second line. And you must type the number within the range. When you type a number, the range narrows; if you got the lucky number luckily or unluckily, there will appear "You've got it!"

Code Explanation

At the beginning part of the code are the functional functions of **keypad** and **I2C LCD1602**. You can learning more details about them in *1.1.7 I2C LCD1602* and *2.1.8 Keypad*.

Here, what we need to know is as follows:

```
/*****************************************/
//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!");
}
```

This function is used to initially define **I2C LCD1602** and **Keypad** and to display "Welcome!" and "Press A to go!".

```
void init_new_value(void){
    srand(time(0));
    pointValue = rand()%100;
    upper = 99;
    lower = 0;
    count = 0;
    printf("point is %d\n",pointValue);
}
```

The function produces the random number '**point**' and resets the range hint of the point.

```
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() compares the input number with the produced "point". If the comparing outcome is that they are not same, **count** will assign values to **upper** and **lower** and return '**0**'; otherwise, if the outcome indicates they are same, there returns '**1**'.

```
void lcd_show_input(bool result){
    char *str=NULL;
    str =(char*)malloc(sizeof(char)*3);
    lcd_clear();
    if (result == 1){
        write(0,1,"You've got it!");
        delay(5000);
        init_new_value();
        lcd_show_input(0);
        return;
    }
    write(0,0,"Enter number:");
    Int2Str(str,count);
    write(13,0,str);
    Int2Str(str,lower);
    write(0,1,str);
    write(3,1,"<Point<");
    Int2Str(str,upper);
    write(12,1,str);
}
```

This function works for displaying the game page. Pay attention to the function **Int2Str(str,count)**, it converts these variables **count**, **lower**, and **upper** from **integer** to **character string** for the correct display of **lcd**.

```
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){
```

(continues on next page)

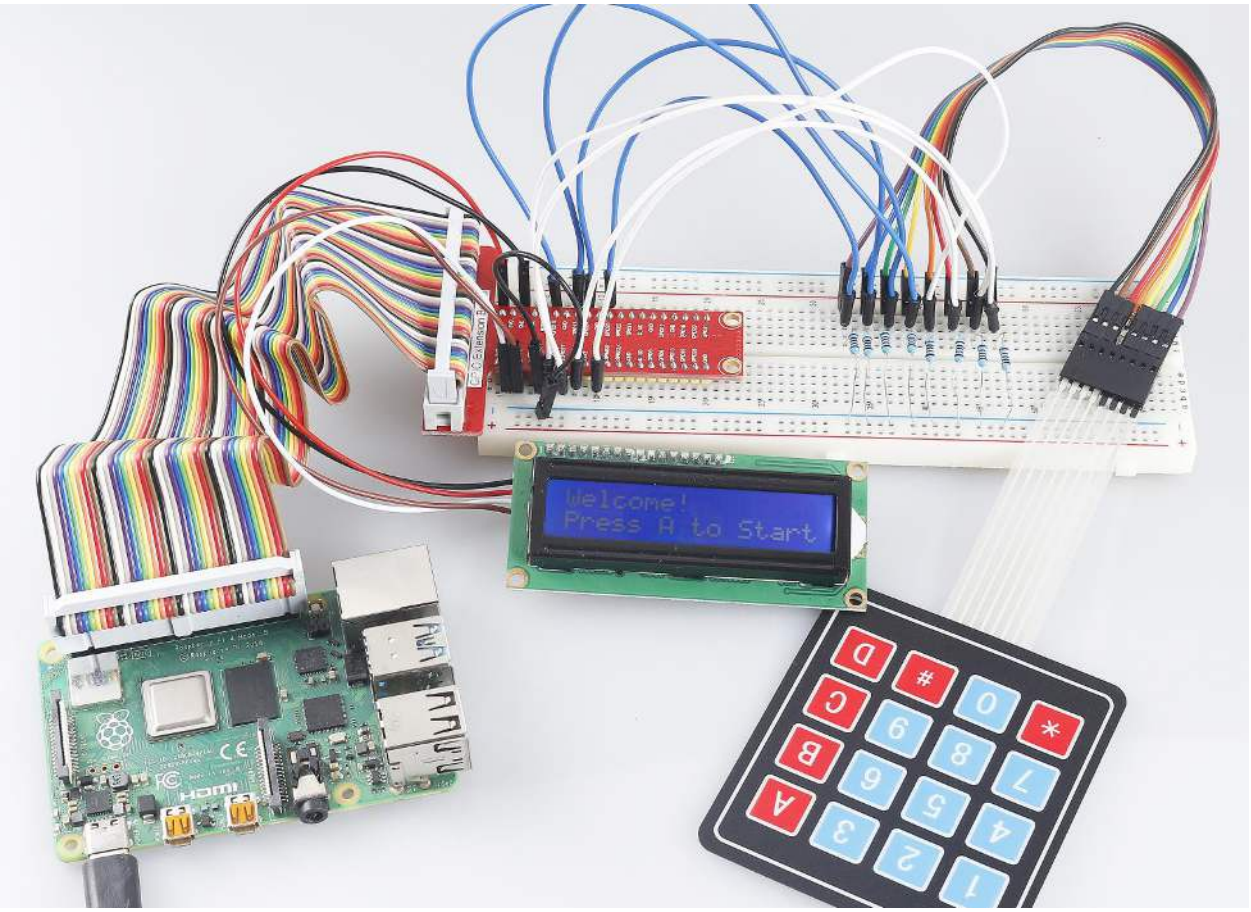
(continued from previous page)

```
        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() contains the whole process of the program, as show below:

1) Initialize **I2C LCD1602** and **Keypad**.
2) Use **init_new_value()** to create a random number **0-99**.
3) Judge whether the button is pressed and get the button reading.
4) If the button '**A**' is pressed, a random number **0-99** will appear then the game starts.
5) If the button '**D**' is detected to have been pressed, the program will enter into the outcome judgement and will display the outcome on the LCD. This step helps that you can also judge the outcome when you press only one number and then the button '**D**'.
6) If the button **0-9** is pressed, the value of **count** will be changed; if the **count** is larger than **10**, then the judgement starts.
7) The changes of the game and its values are displayed on **LCD1602**.

Phenomenon Picture

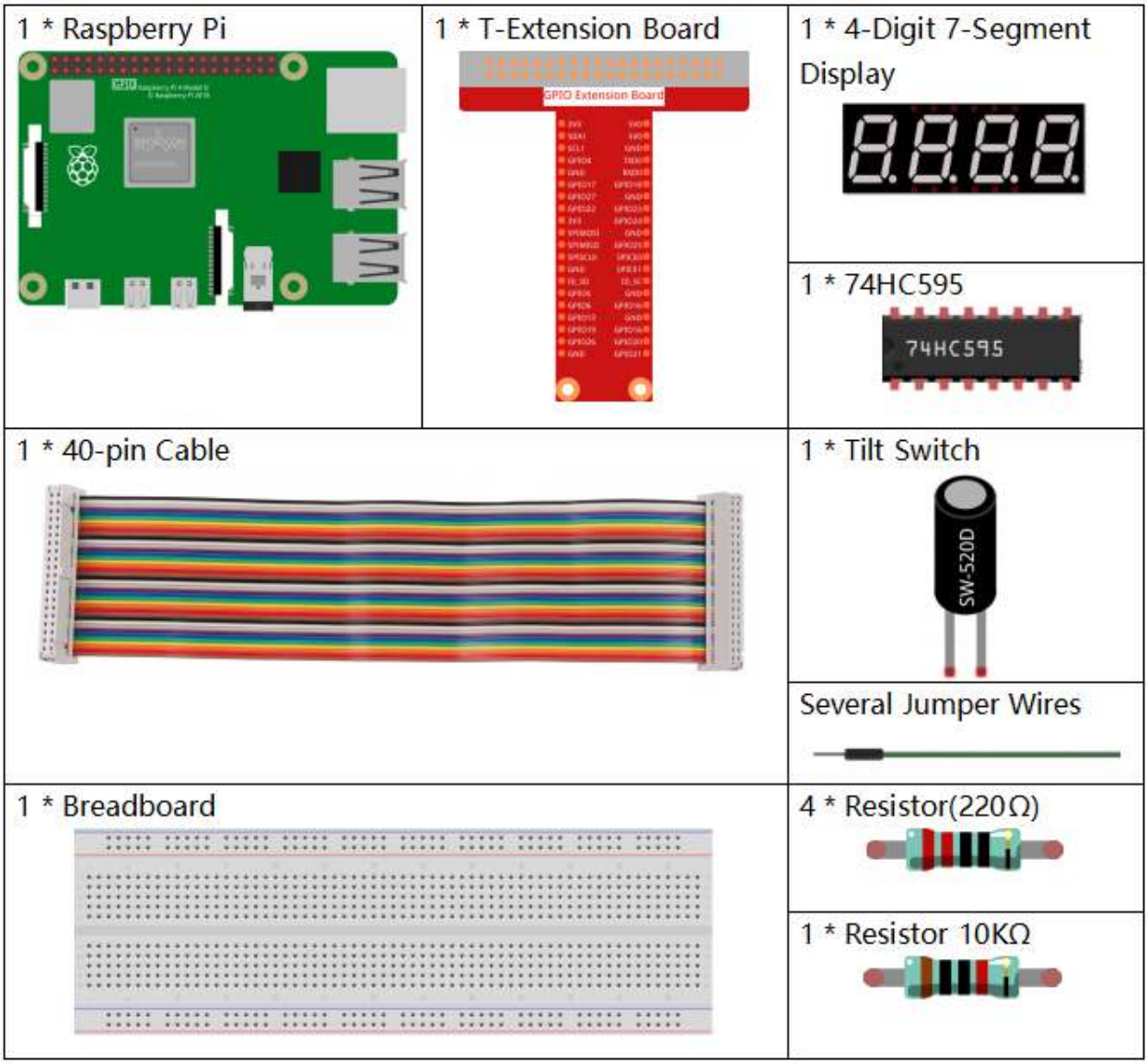


7.4.12 3.1.12 GAME - 10 Second

Introduction

Next, follow me to make a game device to challenge your concentration. Tie the tilt switch to a stick to make a magic wand. Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. You can play the game with your friends to see who is the time wizard.

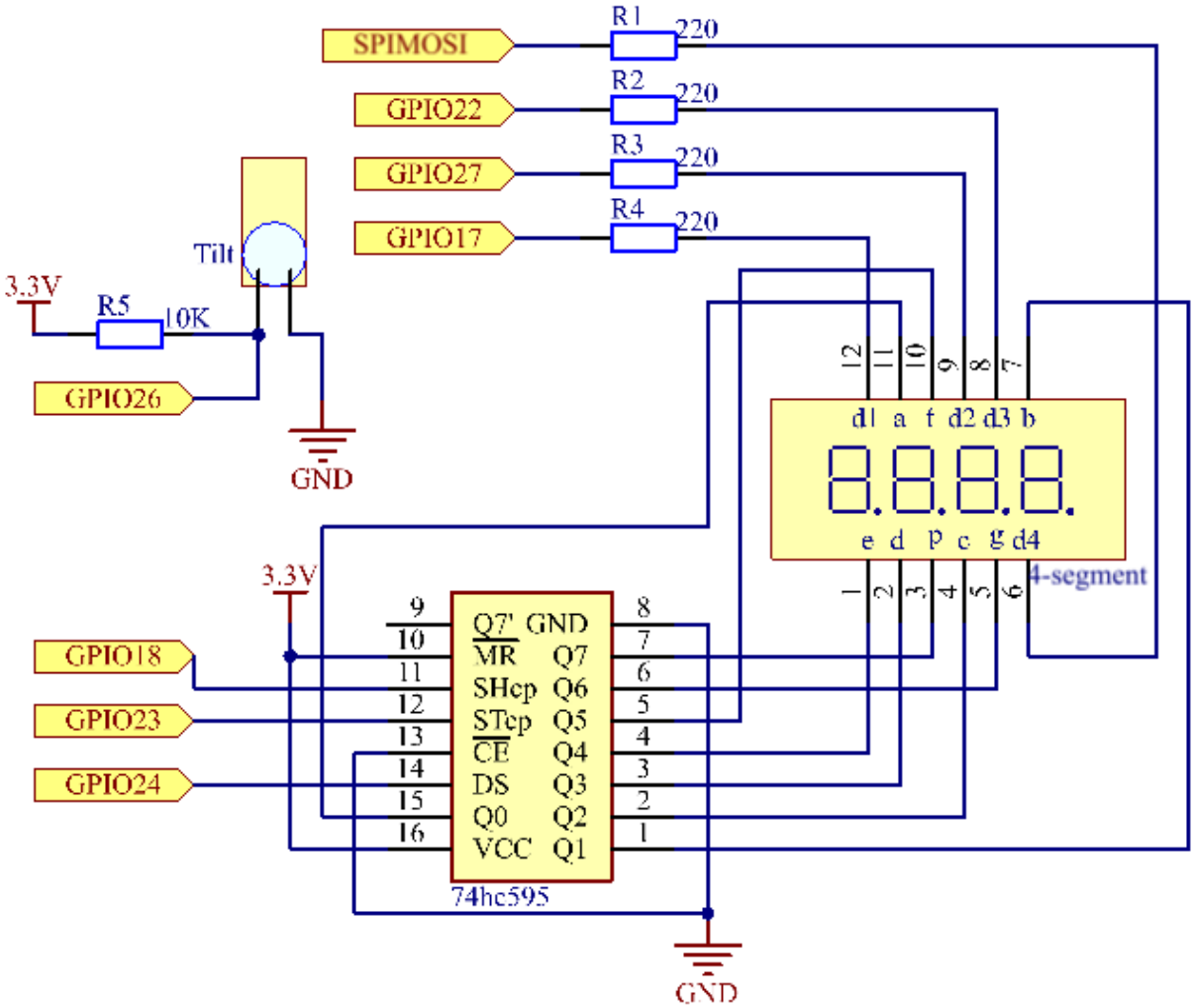
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*
- *Tilt Switch*

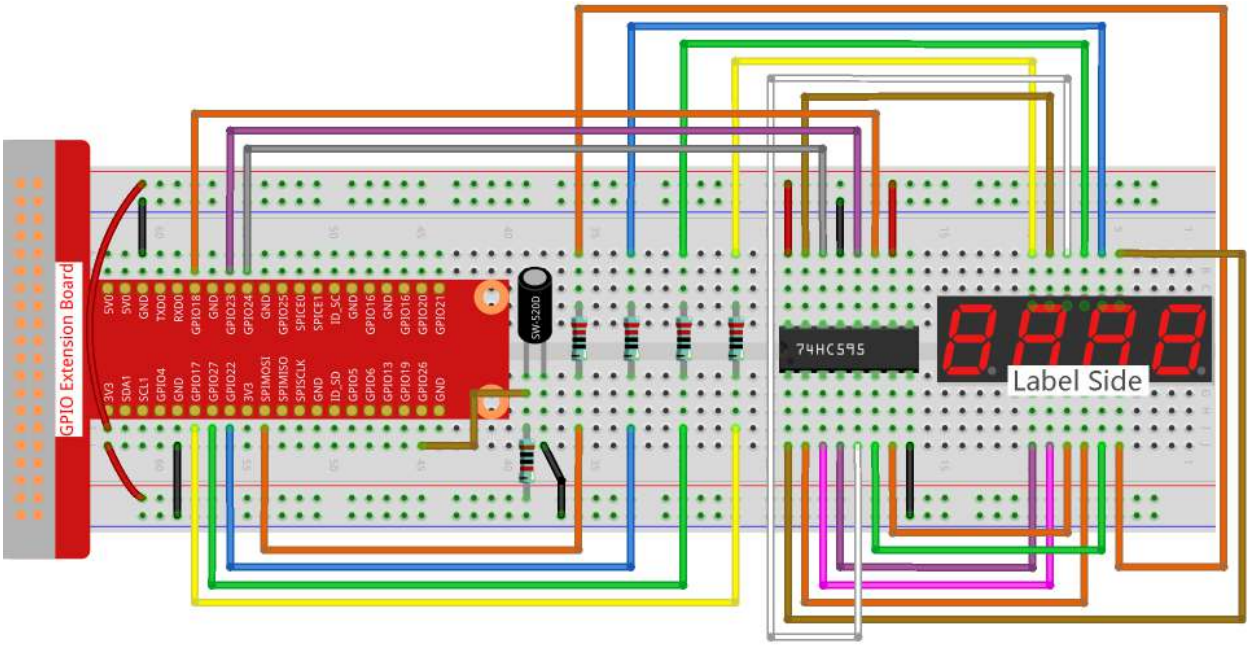
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.12/
```

Step 3: Compile the code.

```
gcc 3.1.12_GAME_10Second.c -lwiringPi
```

Step 4: Run the executable file.

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

Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. Shake it one more time to start the next round of the game.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

```
void stateChange(){
    if (gameState == 0){
        counter = 0;
        delay(1000);
        ualarm(10000,10000);
    }else{
        alarm(0);
        delay(1000);
    }
```

(continues on next page)

(continued from previous page)

```
    gameState = (gameState + 1)%2;
}
```

The game is divided into two modes:

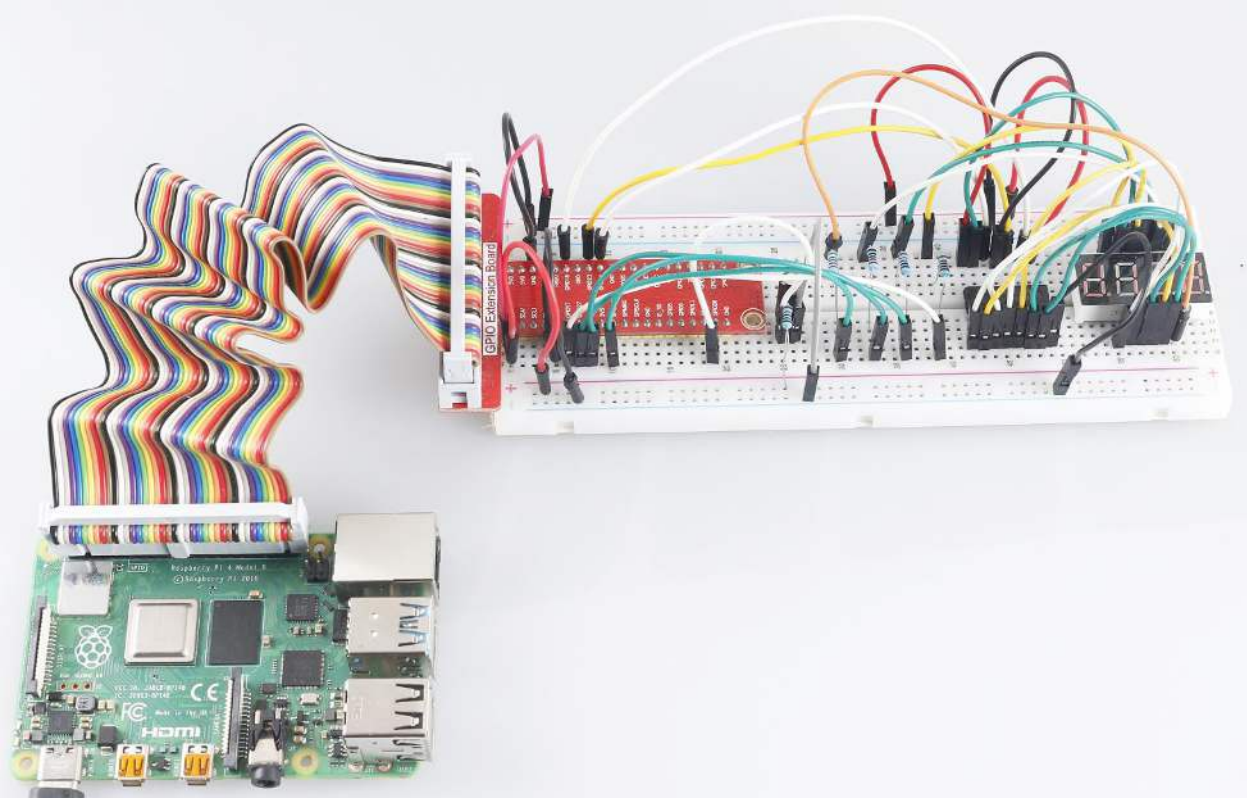
gameState=0 is the "start" mode, in which the time is timed and displayed on the segment display, and the tilting switch is shaken to enter the "show" mode.

GameState =1 is the "show" mode, which stops the timing and displays the time on the segment display. Shaking the tilt switch again will reset the timer and restart the game.

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            stateChange();
        }
        lastState=currentState;
    }
}
```

Loop() is the main function. First, the time is displayed on the 4-bit segment display and the value of the tilt switch is read. If the state of the tilt switch has changed, stateChange() is called.

Phenomenon Picture



7.4.13 3.1.13 GAME– NotNot

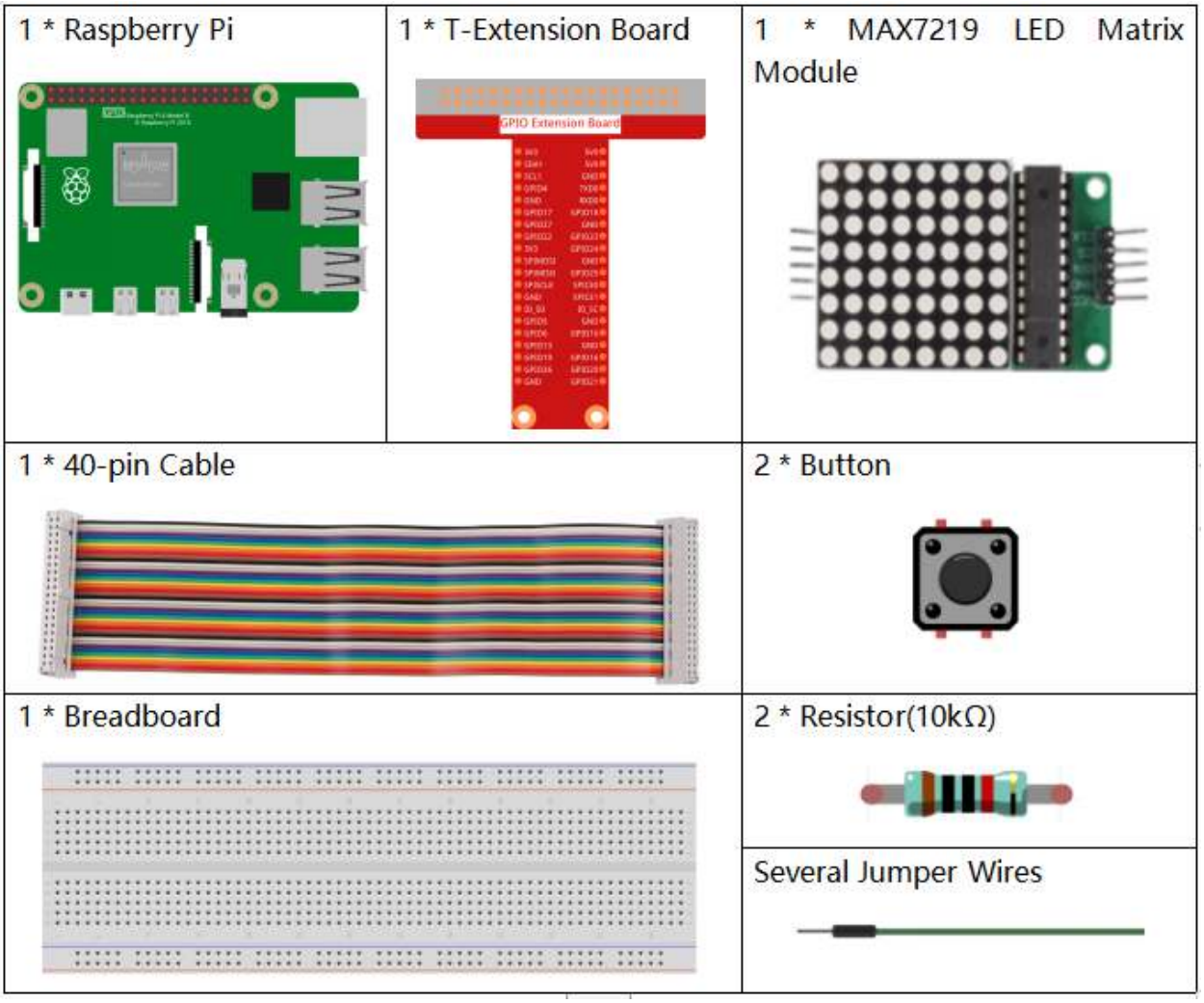
Introduction

In this project, we will make an interesting game device, and we call it “Not Not”.

During the game, the dot matrix will refresh an arrow randomly. What you need to do is to press the button in the opposite direction of the arrow within a limited time. If the time is up, or if the button in the same direction as the arrow is pressed, you are out.

This game can really practice your reverse thinking, and now shall we have a try?

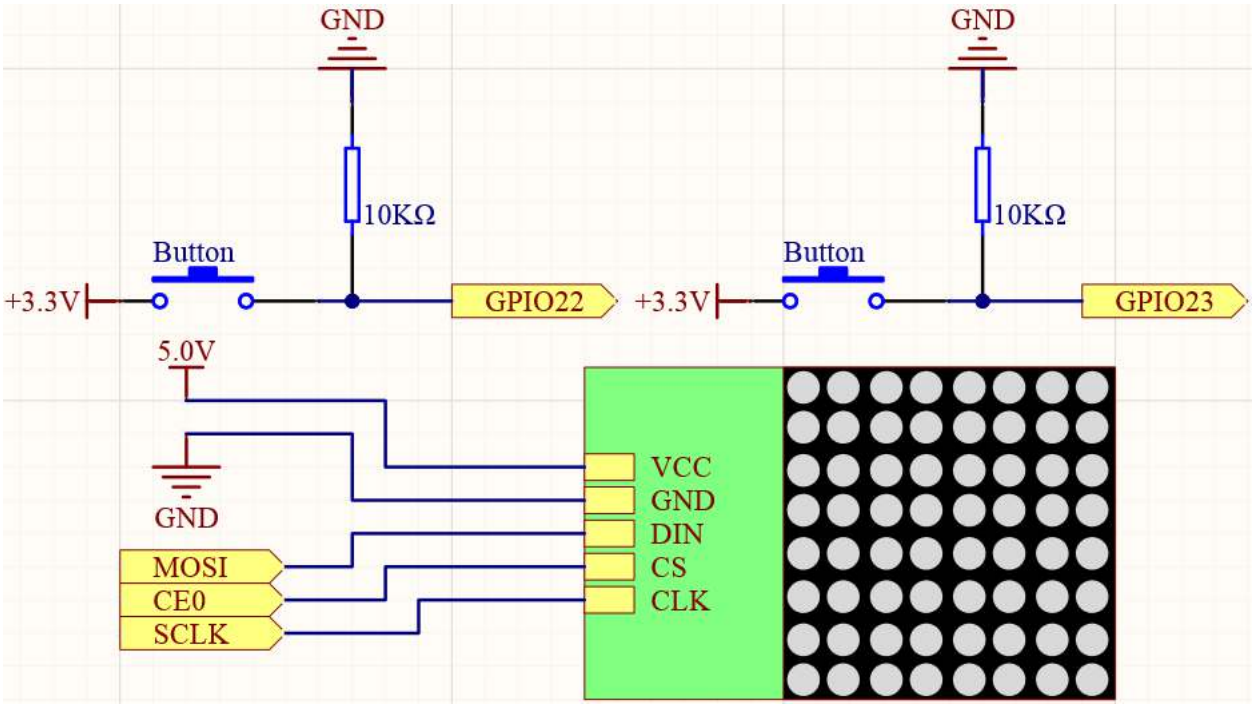
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Matrix Module*
- *Button*

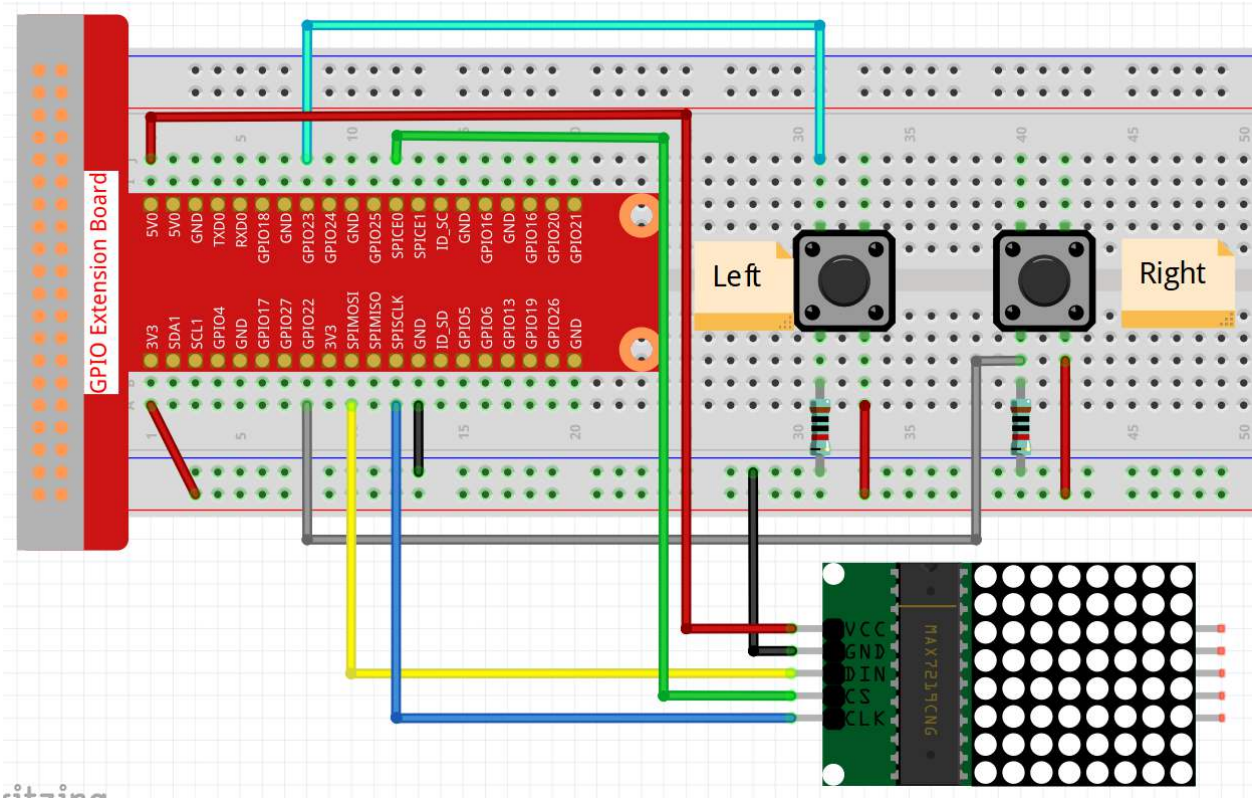
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO22	Pin 15	3	22
GPIO23	Pin 16	4	23
SPIMOSI	Pin 19	12	MOSI
SPICE0	pin 24	10	CE0
SPISCLK	Pin 23	14	SCLK



Experimental Procedures

Step 1: Build the circuit.



Note: Turn on the SPI before starting the experiment, refer to *SPI Configuration* for details. And the *BCM2835* library is also needed.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/c/3.1.13/
```

Step 3: Compile the code.

```
make
```

Step 4: Run the executable file.

```
sudo ./3.1.13_GAME_NotNot
```

After the program starts, a left or right arrow will be refreshed at random on the dot matrix. What you need to do is to press the button in the opposite direction of the arrow, then "" appears on the dot matrix. If the button in the same direction as the arrow is pressed, you are out and the dot matrix displays "x". You can also add 2 new buttons or replace them with Joystick keys for up, down, left and right— 4 directions to increase the difficulty of the game.

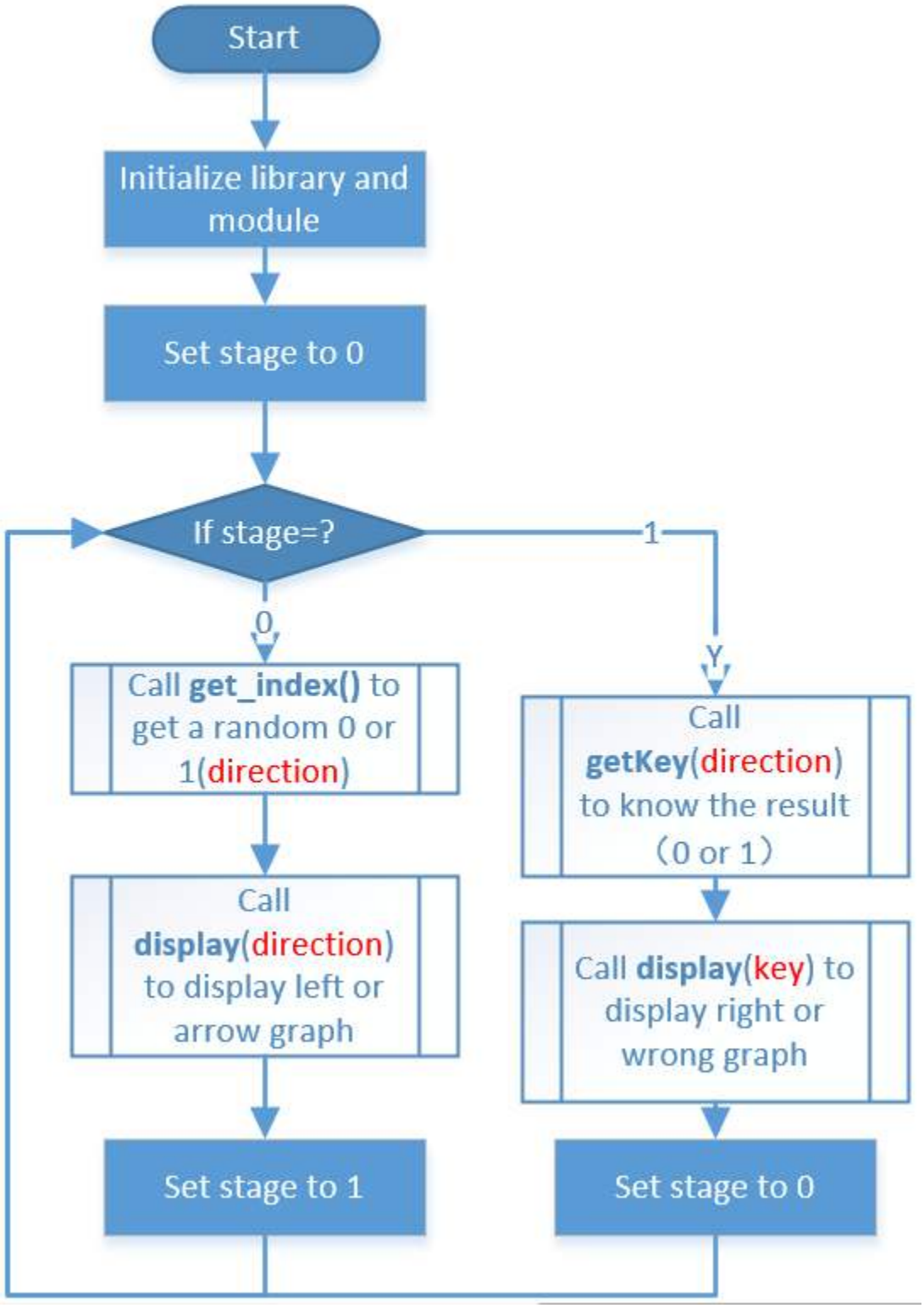
Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code Explanation

Based on *1.1.6 LED Dot Matrix Module*, this project adds 2 buttons to make an amusing game device.

The whole program process is as below:

1. Use system time to generate a random 0 or 1.
2. Display a random left or right arrow pattern.
3. Press the key and determine the result.
4. Display the right or wrong pattern.




```
int get_index()
{
    srand((unsigned)time(NULL));
    return rand()%2;
```

(continues on next page)

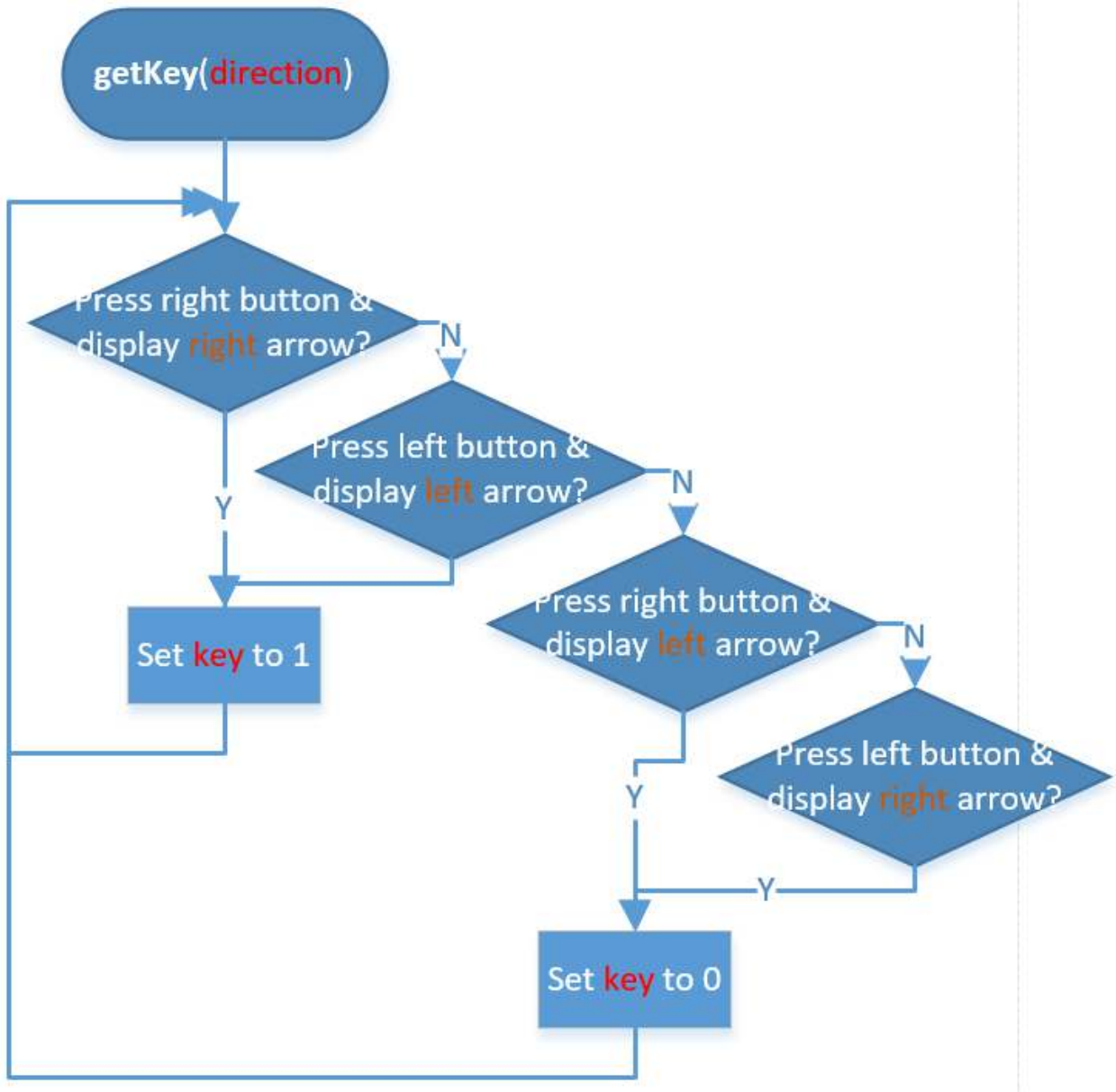
(continued from previous page)

```
}
```

The seed of the system is changed by the system time, i.e. `srand((unsigned)time(NULL))`, so that each time the rand function is called the value obtained is completely random, and finally the result obtained is divided by 2, so that the values obtained are 0 and 1.

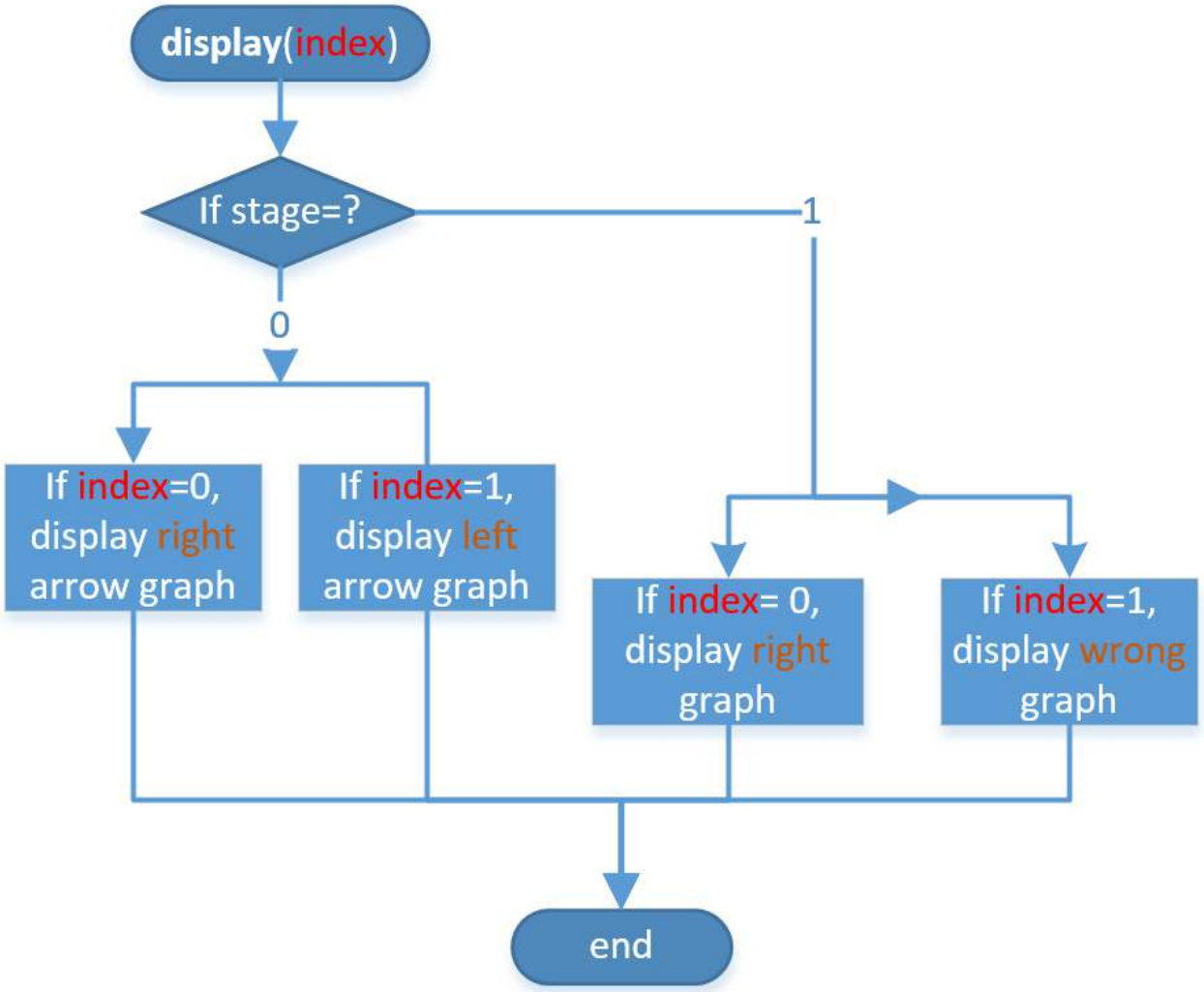
```
int get_key(uint num)
{
    while (1)
    {
        if (1 == bcm2835_gpio_lev(AButtonPin) && num == 0){
            return 1;
        }
        else if (1 == bcm2835_gpio_lev(BButtonPin) && num == 1){
            return 1;
        }
        else if (1 == bcm2835_gpio_lev(AButtonPin) && num == 1){
            return 0;
        }
        else if (1 == bcm2835_gpio_lev(BButtonPin) && num == 0){
            return 0;
        }
    }
}
```

Determines which button was pressed and compares it to the direction of the arrow on the dot matrix and gives the final result of 0 or 1.




```
void display(uint index){
    uchar i;
    if (stage == 0){
        for(i=1;i<9;i++)
        {
            Write_Max7219(i,arrow[index][i-1]);
        }
    }
    else if(stage == 1){
        for(i=1;i<9;i++)
        {
            Write_Max7219(i,check[index][i-1]);
        }
    }
}
```

Depending on the value of the `stage` and `index` to display the `left or right` arrow or the `right or wrong` pattern.



CHAPTER

EIGHT

PLAY WITH PROCESSING

8.1 What is Processing

Processing is a simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction. The developers wanted a means to "sketch" ideas in code. As its capabilities have expanded over the past decade, Processing has come to be used for more advanced production-level work in addition to its sketching role. Originally built as a domain-specific extension to Java targeted towards artists and designers, Processing has evolved into a full-blown design and prototyping tool used for large-scale installation work, motion graphics, and complex data visualization.

Processing is based on Java, but because program elements in Processing are fairly simple, you can learn to use it even if you don't know any Java. If you're familiar with Java, it's best to forget that Processing has anything to do with Java for a while, until you get the hang of how the API works.

This text is from the tutorial, Processing Overview.

8.2 Install the Processing

Note: Before you can use Processing, you need to access the Raspberry Pi desktop remotely (*Remote Desktop*) or connect a display for the Raspberry Pi.

1. First visit https://processing.org/download and select the `LinuxRaspberry Pi 32-bit` or `LinuxRaspberry Pi 64-bit` version. Using this method, you can always download the latest version.

   Or you can use the following command to download the Processing from the Terminal.

   ```
   git clone https://github.com/processing/processing4/releases/download/
   ↪processing-1286-4.0.1/processing-4.0.1-linux-arm32.tgz
   ```

   ```
   git clone https://github.com/processing/processing4/releases/download/
   ↪processing-1286-4.0.1/processing-4.0.1-linux-arm64.tgz
   ```

2. A `.tar.gz` file will be downloaded, which most Linux users should be familiar with. Extract the file you just downloaded from its location.

   ```
   tar xvfz processing-xxxx.tgz
   ```

   Replace xxxx with the rest of the file's name, which is the version number. This will create a folder named processing-xxxx or something similar.

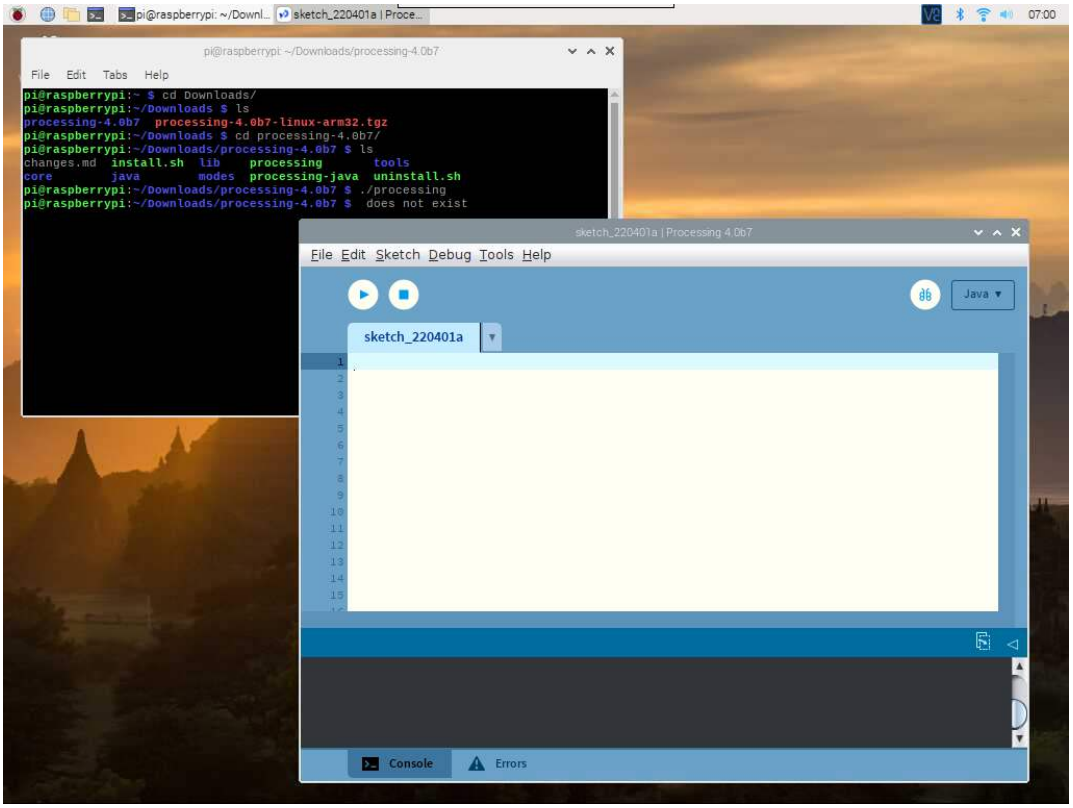
3. Then go to that directory:

```
cd processing-xxxx
```

4. And run it:

```
./processing
```

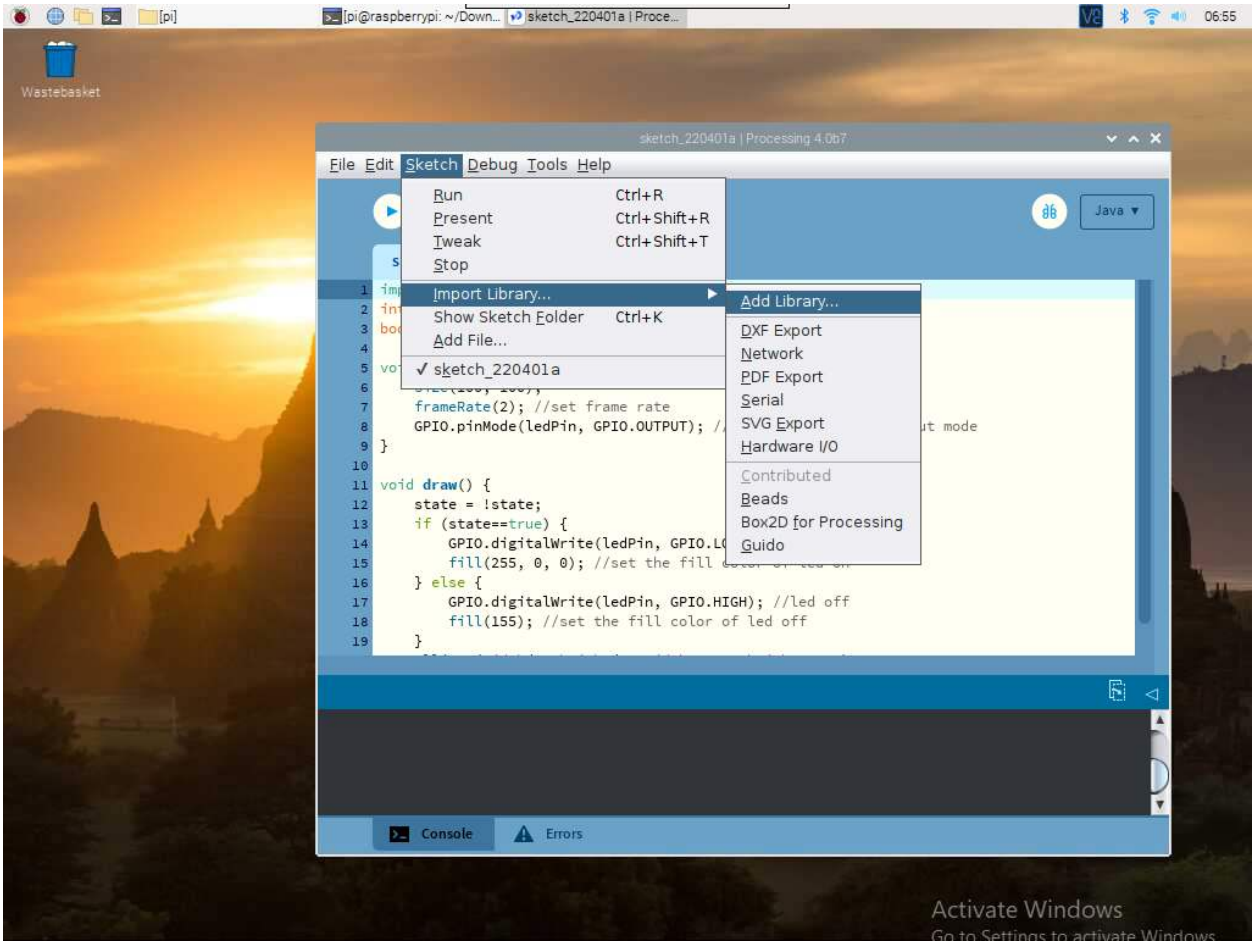
1. With any luck, the main Processing window will now be visible.



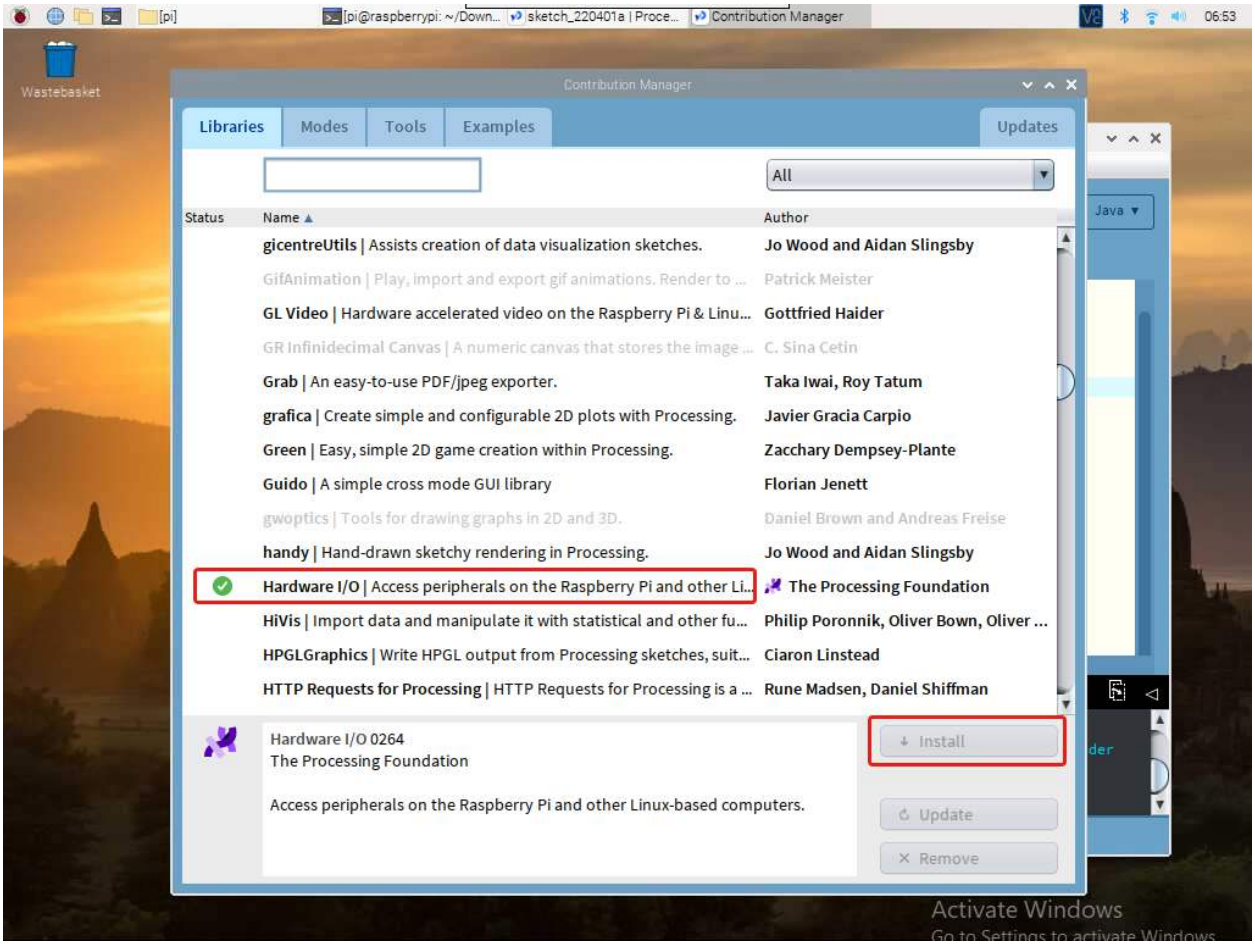
8.3 Install Hardware I/O

In order to use the Raspberry Pi's GPIO, you need to manually add a Hardware I/O library.

Click `Sketch` -> `Import Library` -> `Add Library...`



Find Hardware I/O , select it, and then click Install. When done, a checkmark icon will appear.



8.4 Projects

8.4.1 Draw a Matchmaker

You're now running the Processing Development Environment (or PDE). There's not much to it; the large area is the Text Editor, and there's a row of buttons across the top; this is the toolbar. Below the editor is the Message Area, and below that is the Console. The Message Area is used for one line messages, and the Console is used for more technical details.

Let's get familiar with the usage of Processing and draw a matchmaker.

Sketch

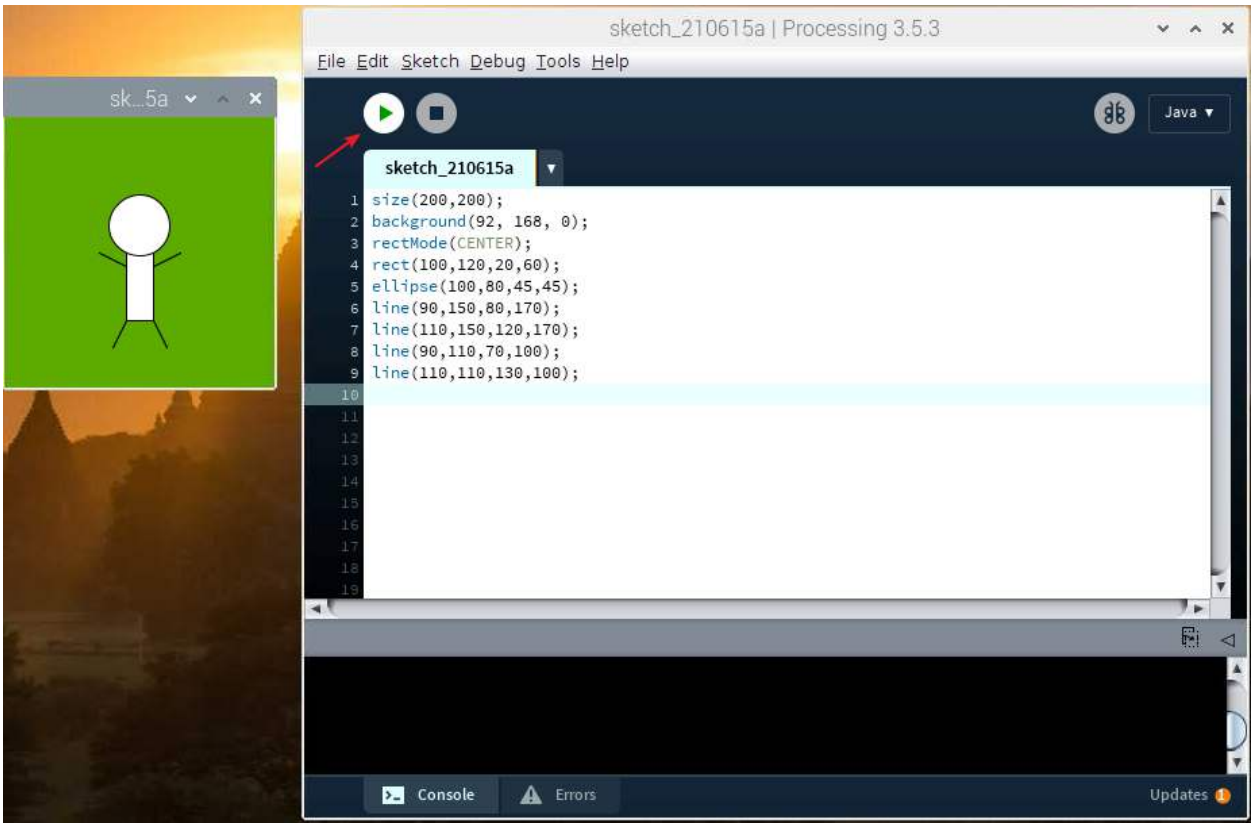
Copy the sketch below into Processing and run it. A new display window will appear and a cheering matchmaker will be drawn.

```
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);
```

(continues on next page)

(continued from previous page)

```
line(90,110,70,100);
line(110,110,130,100);
```



Note: If you run it and the message area turns red and reports some errors, then there is something wrong with the sketch. Make sure you copy the sample sketch exactly: numbers should be enclosed in parentheses, with commas between each number, and lines should end with semicolons.

How it works?

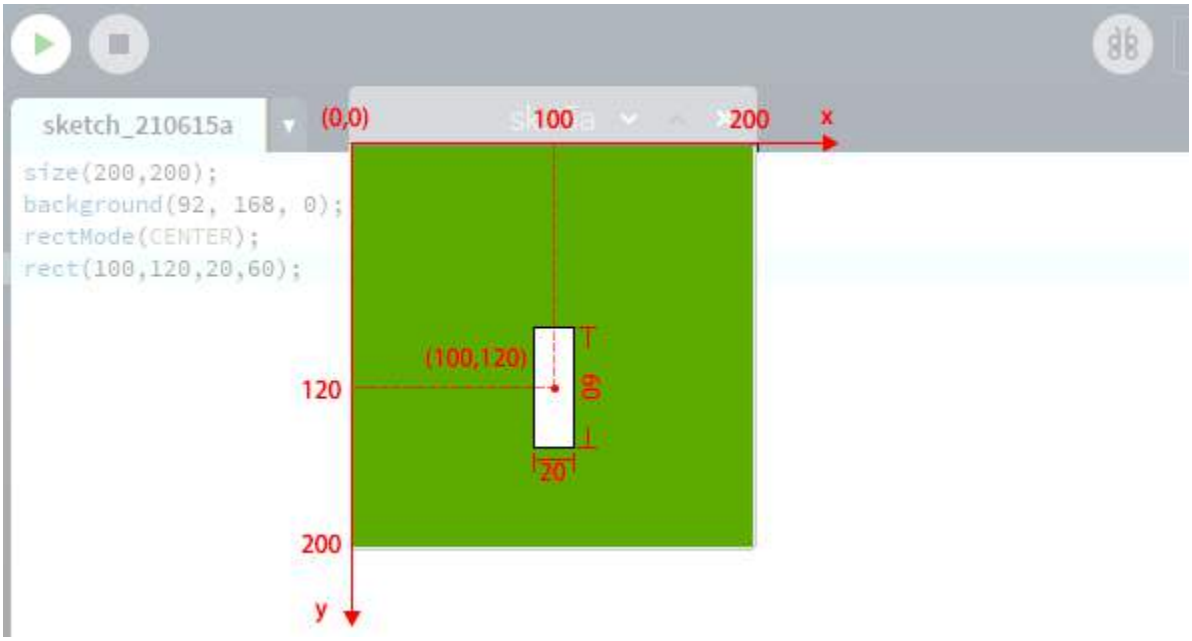
The key here is to realize that the display window can be treated as a square of paper.

Each pixel of the display window is a coordinate (x,y) that determines the position of a point in space. The origin (0,0) of the coordinates is in the upper left corner, the positive direction of the X-axis is horizontally to the right, and the positive direction of the Y-axis is vertically down.

What we have to do is to specify what shape and color should appear at these pixel coordinates.

For example, draw a rectangle of width 20 and height 60 with coordinates (100,120) as the midpoint.

```
rectMode(CENTER);
rect(100,120,20,60);
```



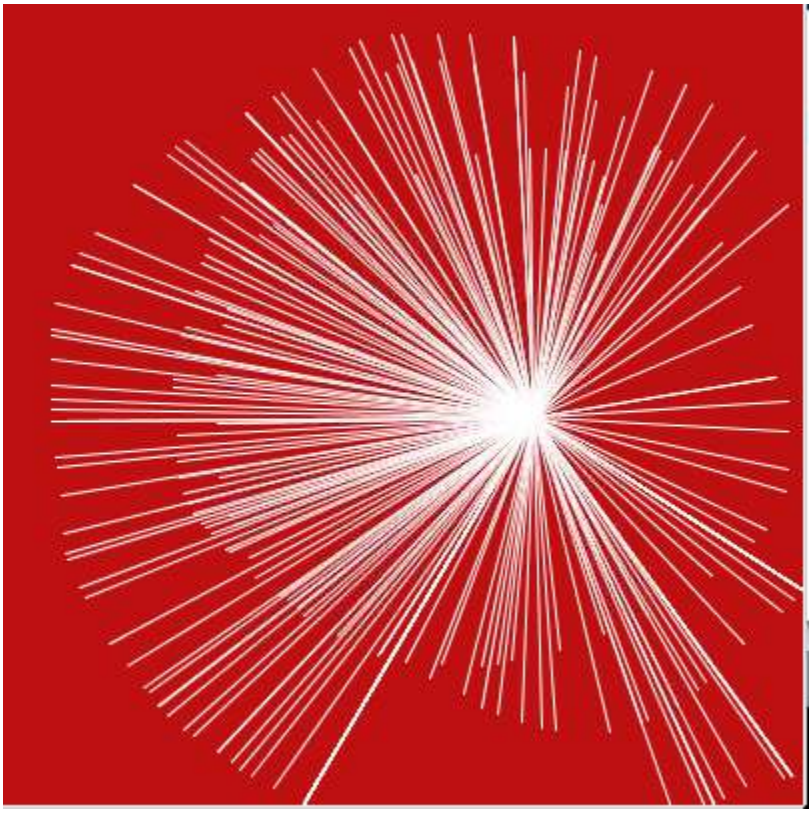
Once we understand the relationship between the display window and the axes, this sketch is not difficult for us, we just need to understand some simple graphic drawing statements.

- `size(width, height)`: Defines the dimension of the display window width and height in units of pixels.
- `background(red, green, blue)`: Set the background color of the display window.
- `rectMode(mode)`: Modifies the location from which rectangles are drawn by changing the way in which parameters given to `rect()` are intepreted.
- `rect(x, y, width, height)`: Draws a rectangle to the screen.
- `ellipse(x, y, width, height)`: Draws an ellipse (oval) to the screen.
- `line(x1, y1, x2, y2)`: Draws a line (a direct path between two points) to the screen.

For more please refer to Processing Reference.

8.4.2 Hello Mouse

In this project, your mouse will keep shooting lines towards a point; move the mouse and you will draw a unique line of stars. Press the mouse to restart the drawing.



Sketch

```
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);
}
```

How it works?

The previous project was drawing a single image without any animation or interaction.

If we want to make an interactive sketch, we need to add the `setup()` and `draw()` functions (these are built-in functions that are called automatically) to build the frame.

- `setup()`: Executed only once at the start of the sketch.
- `draw()`: Executed repeatedly, where we usually add the sketch for drawing the animation.

```
int pointX = 172;
int pointY = 88;

void setup() {
    size(400, 400);
    stroke(255);
    background(192, 16, 18);
}

void draw() {
    line(pointX, pointY, mouseX, mouseY);
}
```

This sketch above already works smoothly as an interactive sketch.

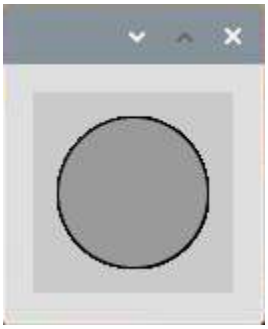
Next you can add a mouse click event. This event can be implemented with the `mousePressed()` function, where we add statements to refresh the target point and clear the screen.

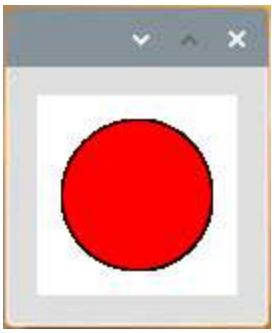
```
int pointX = 172;
int pointY = 88;

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);
}
```

For more please refer to Processing Reference.

8.4.3 Blinking Dot

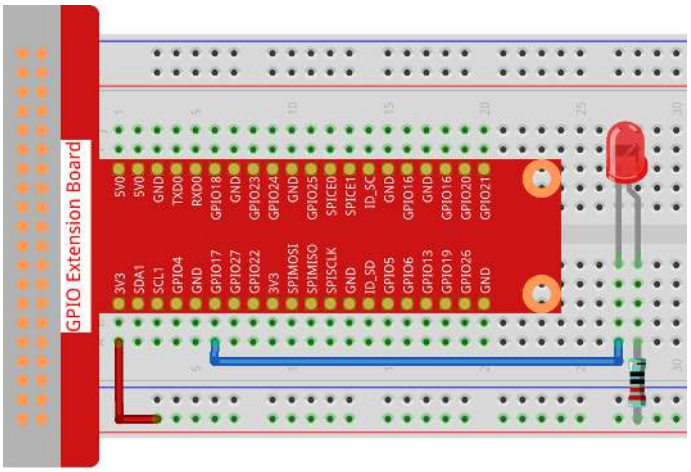
In this project, we will draw a dot on Processing, which blinks synchronously with the LED. Please build the circuit as shown in the diagram and run the sketch.





Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*



Sketch

```
import processing.io.*;
int ledPin = 17;
boolean state = true;

void setup() {
    size(100, 100);
    frameRate(2); //set frame rate
    GPIO.pinMode(ledPin, GPIO.OUTPUT); //set the ledPin to output mode
}

void draw() {
    state = !state;
    if (state==true) {
        GPIO.digitalWrite(ledPin, GPIO.LOW); //led on
        fill(255, 0, 0); //set the fill color of led on
    } else {
        GPIO.digitalWrite(ledPin, GPIO.HIGH); //led off
        fill(155); //set the fill color of led off
    }
```

(continues on next page)

(continued from previous page)

```
    ellipse(width/2, height/2, width*0.75, height*0.75);
}
```

How it works?

At the beginning of the sketch, you need to embed Processing's GPIO function library by `import processing.io.*;`, which is indispensable for circuit experiments.

Frame rate is the frequency of bitmaps appearing on the board, expressed in hertz (Hz). In other words, it is also the frequency at which the `draw()` function is called. In `setup()`, setting the **frame rate** to 2 will call `draw()` every 0.5s.

Each call of the `draw()` function takes the inverse of `state` and subsequently determines it. If the value is `true`, the LED is lit and the brush is filled with red; if not, the LED is turned off and the brush is filled with gray.

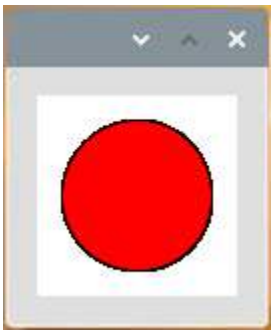
After completing the judgment, use the `ellipse()` function to draw a circle. It should be noted that `width` and `height` are system variables used to store the width and height of the display window.

There are two other points to note. When using GPIOs, you need to use the `GPIO.pinMode()` function to set the INPUT/OUTPUT state of the pin, and then use the `GPIO.digitalWrite()` function to assign a value (HIGH/LOW) to the pin .

Note: Please try to avoid using `delay()` in `draw()` because it will affect the display window refresh.

8.4.4 Clickable Dot

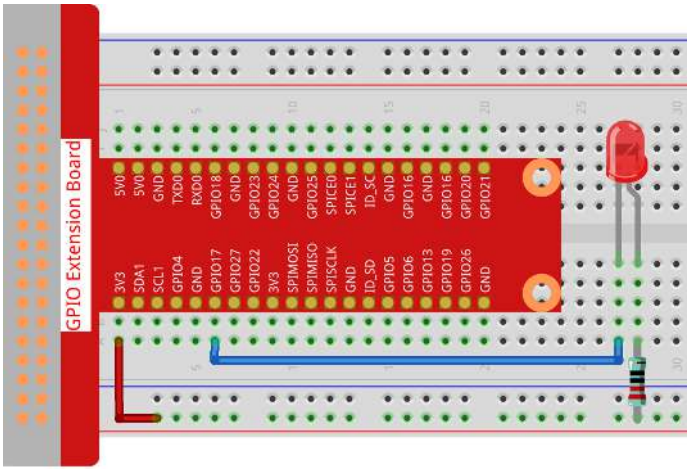
We've tried drawing motion graphic, responding to mouse event, and controlling LED. So, we might as well combine these functions, draw a clickable dot, to control the LED!



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*

Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*



Sketch

```
import processing.io.*;
boolean state = false;
int ledPin = 17;

void setup() {
    GPIO.pinMode(ledPin, GPIO.OUTPUT);
    background(255);
}

void draw() {
    if (state == true) {
        GPIO.digitalWrite(ledPin, GPIO.LOW);
        fill(255, 0, 0);
    }else {
        GPIO.digitalWrite(ledPin, GPIO.HIGH);
        fill(155);
    }
    ellipse(width/2, height/2, width*0.75, height*0.75);
}

void mouseClicked() {
    //  toggles state:
    if (2*dist(mouseX,mouseY,width/2, height/2)<=width*0.75)
        {state = !state;}
}
```

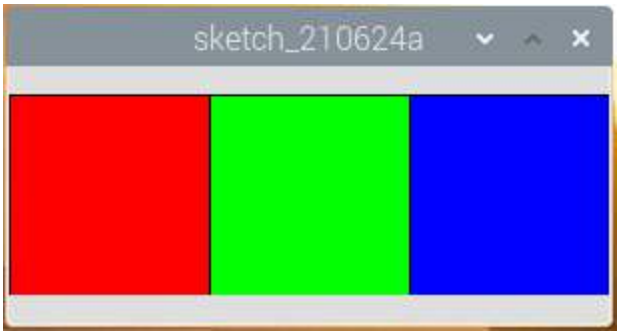
How it works?

This project has a lot in common with *Blinking Dot*, the difference is that it puts the toggle state in the mouse event. This causes the LED to not blink automatically, but to light up and go off with a mouse click.

And in the `mouseClicked()` event, the `dist()` function is used to determine the position of the mouse at the time of the click, and the dot is considered clicked only if the distance between the mouse and the center of the dot is less than the radius.

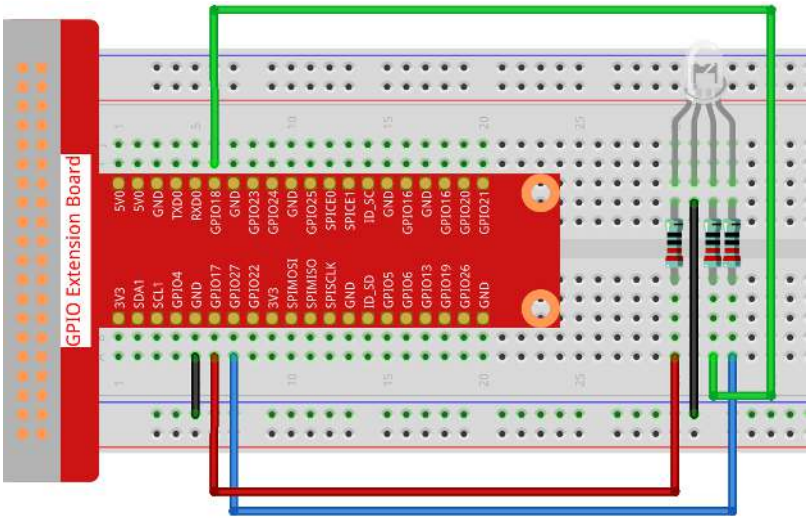
8.4.5 Clickable Color Blocks

We've already tried drawing a clickable dot to control the LED, so let's take it a step further and draw 3 colored squares to adjust the RGB colors!



Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*



Sketch

```
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);
}
```

(continues on next page)

(continued from previous page)

```
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);
    }
}
```

How it works?

This project has a lot in common with *Clickable Dot*, except that it refines the conditions for determining mouse click event.

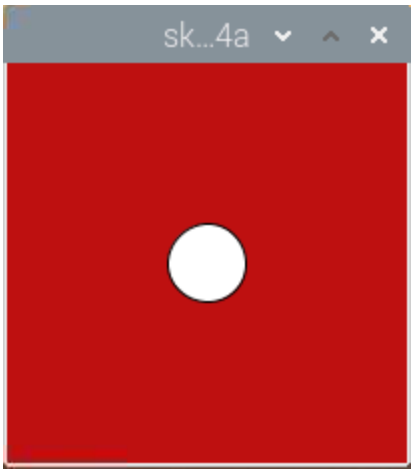
First draw three color blocks in `draw()`, then get which color block was clicked based on the value of mouseX (the X-axis coordinate of the mouse), and finally make RGB light up the corresponding color.

What more?

Based on the addition of light, we can make RGB LED display seven colors - adding red to green produces yellow; adding all three primary colors together produces white. Now you can try it out for yourself.

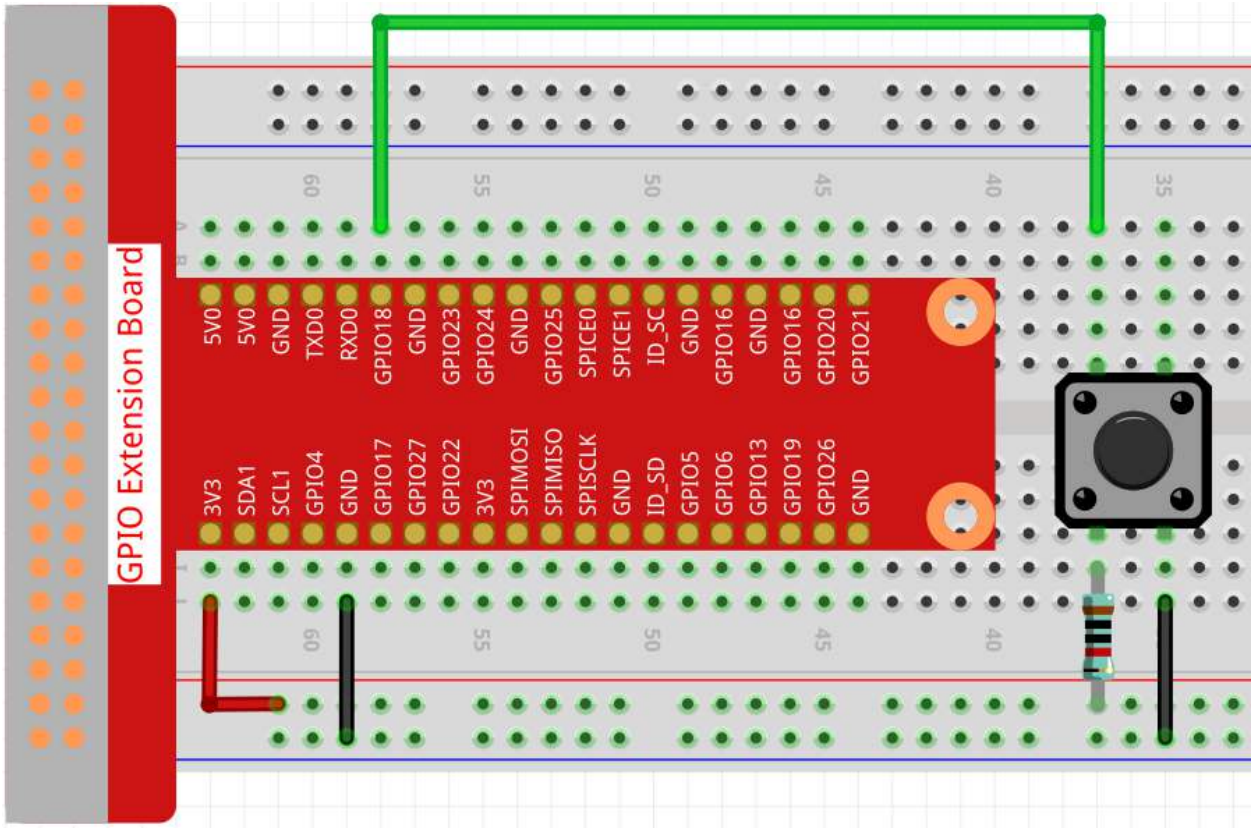
8.4.6 Inflating the Dot

Next, let's build a circuit that allows the button to control the size of the dot. When we press the button, the dot will quickly get bigger; when we release the button, the dot will gradually get smaller, which makes the dot look like a balloon being inflated.



Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Button*



Sketch

```
import processing.io.*;
int buttonPin = 18;
```

(continues on next page)

(continued from previous page)

```
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);
}
```

How it works?

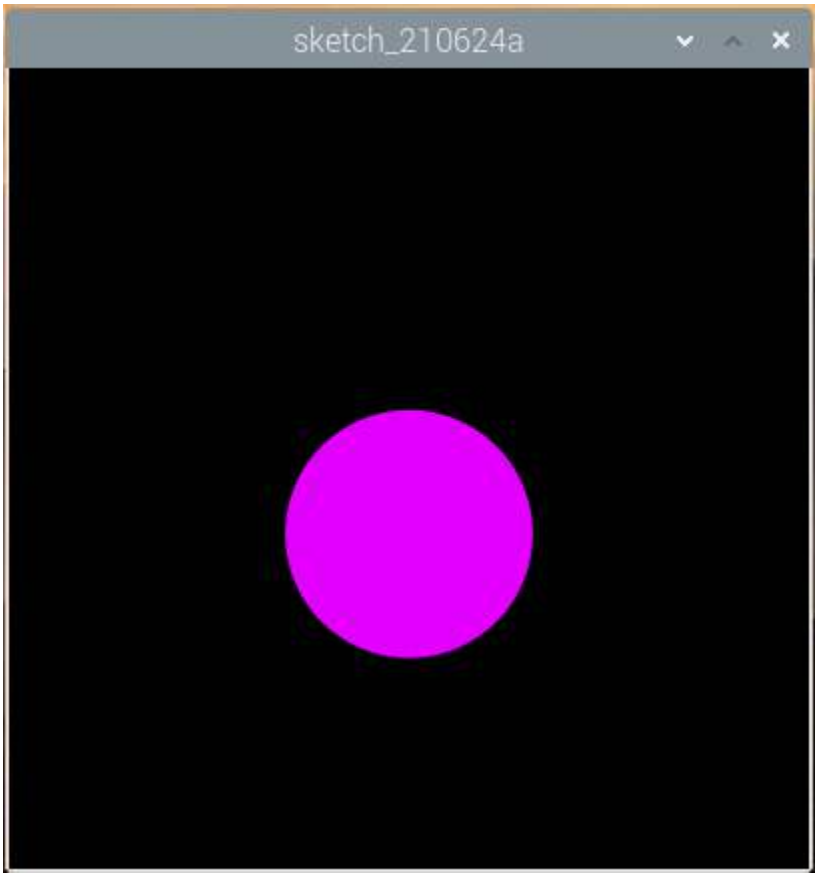
This project uses the input function compared to the previous 2 projects that used the output function of the GPIO.

The `GPIO.pinMode()` function is used to set `buttonPin` to pull-up input mode, which makes the pin get high automatically in the default state.

Then use the `GPIO.digitalRead()` function to read the value of `buttonPin`. When the value is LOW, it means the button is pressed, at which point let the diameter of the dot increase by 5; if the button is released, then the diameter of the dot will decrease by 1.

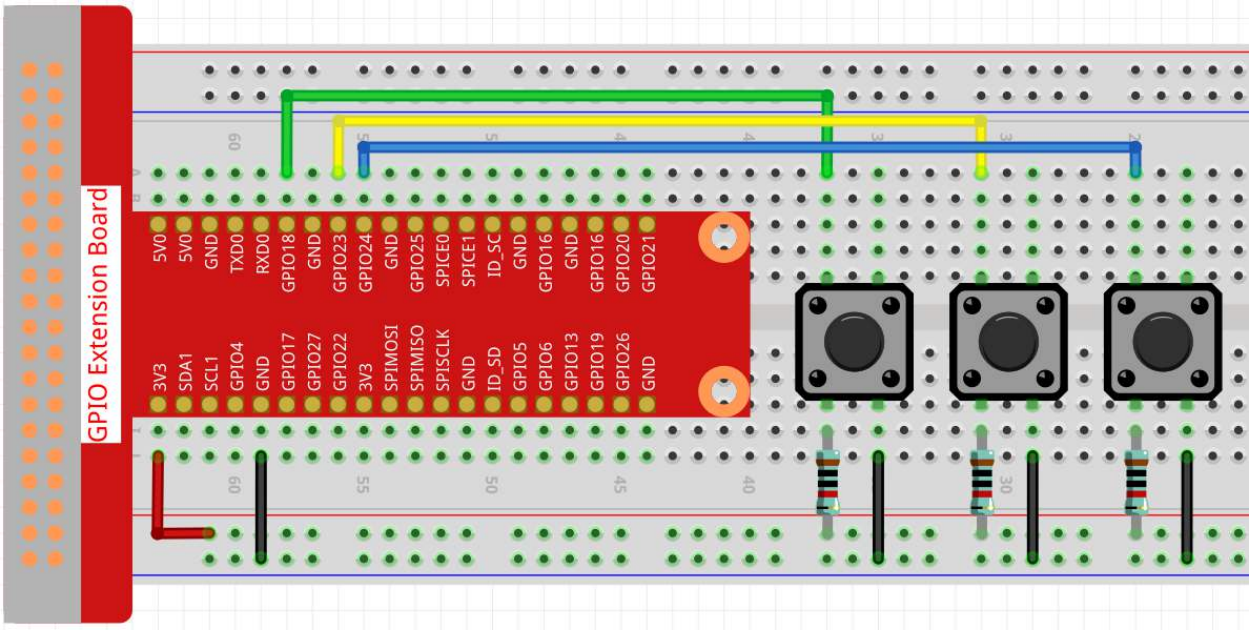
8.4.7 Dot on the Swing

In this project, 3 buttons are connected, one to change the size of the dot, one to change the position and the last one to change the color. If you press all 3 buttons at the same time, you will get a dot that is swinging and has a variable color.



Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Button*



Sketch

```
import processing.io.*;

// Define an instance of the Dot object
Dot myDot;

// Define the pins that will be reading button presses
int[] pins = { 18, 23, 24 };

void setup() {
    size(400, 400);
    // Change the color mode of the sketch to HSB
    colorMode(HSB, 360, 100, 100);
    noStroke();

    for (int i = 0; i < pins.length; i++) {
        GPIO.pinMode(pins[i], GPIO.INPUT_PULLUP);
    }

    // Create a Dot in the middle of the screen
    myDot = new Dot(width / 2, height / 2, 100, 255);
}

void draw() {
    background(0);

    // Modify attributes of the Dot depending on which buttons are pressed
    if (GPIO.digitalRead(pins[0]) == GPIO.LOW) {myDot.setSize();}
    if (GPIO.digitalRead(pins[1]) == GPIO.LOW) {myDot.setPosition();}
    if (GPIO.digitalRead(pins[2]) == GPIO.LOW) {myDot.setColor();}

    // Update the Dot state
    myDot.update();
    // And draw it to the screen
    myDot.show();
```

(continues on next page)

(continued from previous page)

```
}

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;
    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);
    }
}
```

How it works?

Instead of drawing dot directly, we create a `Dot` class here. Then, declare the object (in this case `myDot`).

This is a simple way to draw dots with multiple identical properties. For example, if we add three functions to the dot in this project - change size, change position and change color - then each dot we declare will have the same function. We can use the same button to make them do the same thing, or we can use different buttons to control each dot separately.

Using **classes** makes your sketch beautiful, powerful and flexible.

Class (computer programming) - Wikipedia

Next, let's take a closer look at the `Dot` class.

```
Dot(float x, float y, float s, int c)
```

In the declaration, it needs to pass in four parameters, which are the X and Y coordinate value of the position, the size, and the color (here it is set to the HSB color mode).

Each parameter will be assigned to 2 sets of values (initial value and current value).

```
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;
```

In addition to the initial value and the current value, there is also a set of range values. It is not difficult to understand that the initial value is used to determine the initial state of the dot (determined by the incoming parameters), while the current value will change within the range to make the dot move.

Therefore, except for the X coordinate value, the current values of the other three parameters are calculated as follows:

```
void setSize() {
    currentSize = initSize + sizeRange * sin( timer );
}

void setPosition() {
    currentY = initY + positionRange * cos( timer *2);
}

void setColor() {
    currentColor = int(initColor + ColorRange * sin( timer ));
}
```

If you are familiar with trigonometric functions, it should not be difficult to understand sine and cosine, which gives a smooth periodic change (from -1 to 1) of the current value of the dot.

We also need to add a seed, `timer`, for the periodic variation. It adds the fixed value in the method `update()` and is called in `draw()`.

```
void update() {
    timer += speed;
}
```

Finally, the dot is displayed according to the current value using the method `show()`, which is also called in `draw()`.

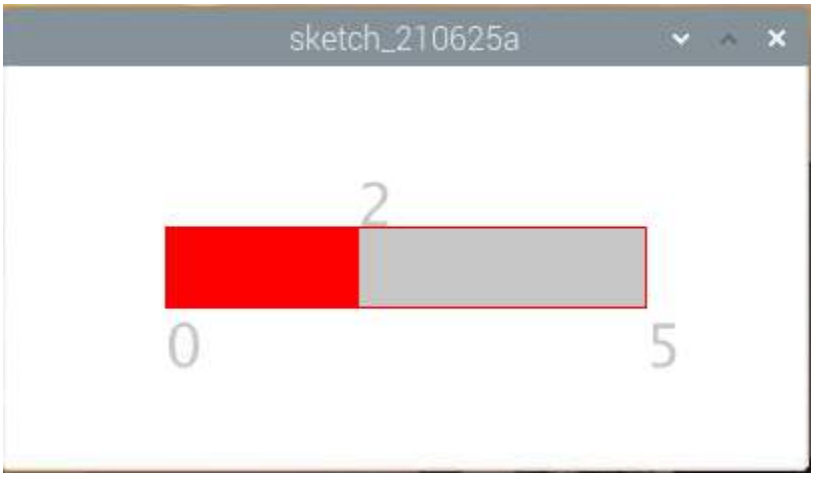
```
void show() {
    fill(currentColor, 100, 100);
    ellipse(currentX, currentY, currentSize, currentSize);
}
```

What more?

Having mastered the use of classes, you can already draw multiple dots with the same properties, so why not try to do something cooler. For example, how about drawing a stable binary star system, or making a 'DUET' game?

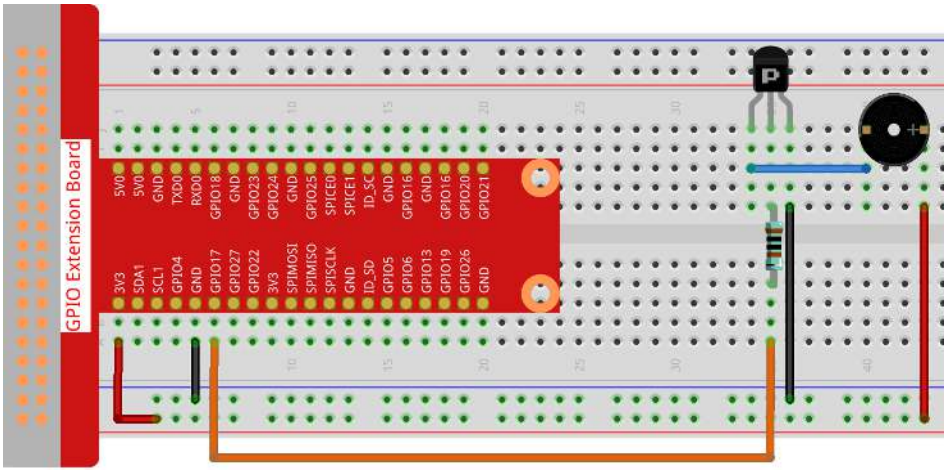
8.4.8 Metronome

Here we will make a metronome, the metronome is divided into 5 levels, the higher the level, the more urgent the buzzer call.



Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*



Note: An active buzzer is used here, and it has a white sticker on it.

Sketch

```
import processing.io.*;

int level = 0;
int buzzerPin = 17;
int levelRange=5;
Slider mySlider;

void setup() {
    size(400, 200);
    frameRate(50);
    mySlider = new Slider(width * 0.2,height * 0.4,width * 0.8,height * 0.6,0,
↪levelRange,level);
    GPIO.pinMode(buzzerPin, GPIO.OUTPUT);
}

void draw() {

    background(255);
    mySlider.show();
    if(level==0){
        GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    }else if((frameCount/5) % (levelRange-level+1) ==0){
        GPIO.digitalWrite(buzzerPin, GPIO.LOW);
    }else{
        GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    }
}

void mouseDragged(){
    level = mySlider.dragPoint(mouseX,mouseY);
}

class Slider{
    float slotPointAX;
    float slotPointBX;
    float slotPointAY;
```

(continues on next page)

(continued from previous page)

```
    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;
    }
}
```

How it works?

Here, we created a `Slider` class and made it act as **WIDGET**.

```
Slider(ax, ay, bx, by, min, max, v)
```

In the declaration, it needs to be passed in 7 parameters.

The first four parameters determine the size of the widget, followed by the coordinates (x1, y1) of the starting point in the upper left corner and (x2, y2) in the lower right corner.

The last three parameters determine its numerical range (min to max) and initial value.

It has two methods, the effect of `dragPoint()` is to make the slider draggable and return the slider's current position value.

```
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;
}
```

Another method `show()` is to show the Slider. At the same time, the range value and current value are displayed in the corresponding position.

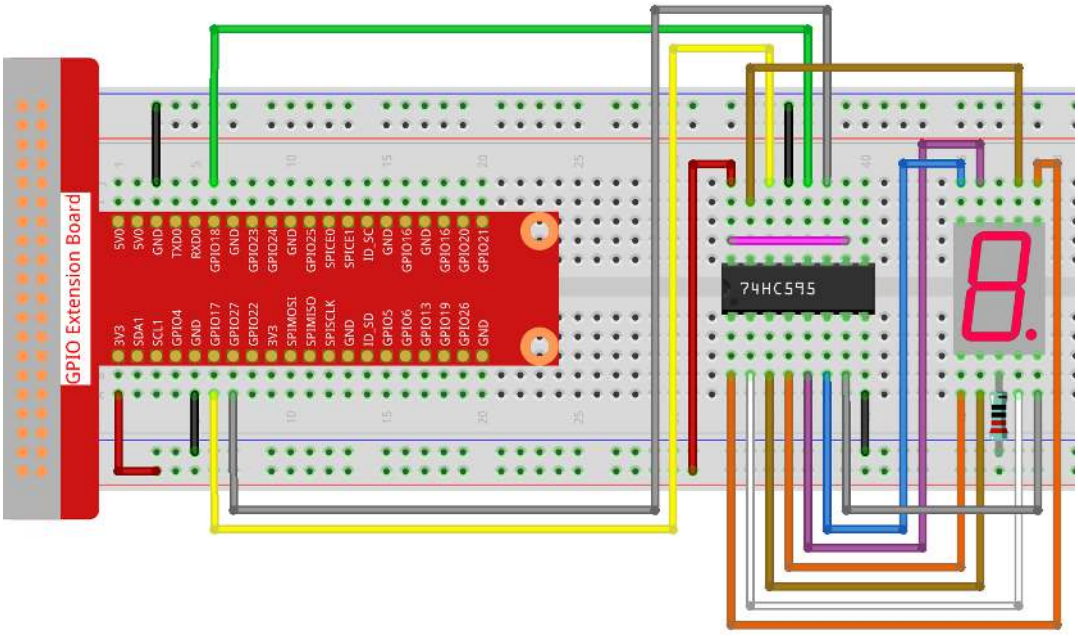
```
void show(){
    rectMode(CORNERS);
    fill(200);
    stroke(255,0,0);
    rect(slotPointAX, slotPointAY, slotPointBX, slotPointBY);
    fill(255,0,0);
    rect(slotPointAX, slotPointAY, linePoint, slotPointBY);
    fill(200);
    textSize(depth);
    text(minRange, slotPointAX, slotPointBY+depth);
    text(maxRange, slotPointBX, slotPointBY+depth);
    text(value, linePoint, slotPointAY);
}
```

8.4.9 Show Number

In this project, we use processing to drive a 7-segment display to show a figure from 0 to 9 and A to F.

Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *7-segment Display*
- *74HC595*



Sketch

```
import processing.io.*;

int SDI=17;   //serial data input
int RCLK=18;  //memory clock input(STCP)
int SRCLK =27;   //shift register clock input(SHCP)


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

void hc595_shift(int dat){
  int i;

  for(i=0;i<8;i++){
        int n=(0x80 & (dat << i));
        if ( n==0){
          GPIO.digitalWrite(SDI, 0);
        } 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);
```

(continues on next page)

(continued from previous page)

```
        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]);
}
```

How it works?

Import `processing.io.*` and use the GPIO function library to control the digital tube pins.

Define array `SegCode = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71}` which represents a segment code array from 0 to F in Hexadecimal (Common cathode).

`setup()` function sets the three pins SDI,RCLK and SRCLK as output, and the initial data as 0.

`hc595_shift(int dat)` function is used to shift the `SegCode` to 74HC595.

```
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))` means to shift dat to the left by `i` bits and then do the `&` operation with 0x80.

The rule of `&` operation is that when both sides of `&` are 1, the result is 1, otherwise the result is 0.

For example, we assume dat=0x3f,i=2(0011 1111 << 2 shift to 1111 1100), then 1111 1100 & 1000 0000 (0x80)) = 1000 0000.

At last assign the dat data to SDI(DS) by bits.

`digitalWrite(SRCLK, 1)` when SRCLK generates a rising edge pulse from 0 to 1, the data will be transferred from the DS register to the shift register;

`digitalWrite(RCLK, 1)` when RCLK generates a rising edge pulse from 0 to 1, the data will be transferred from the shift register to the storage register.

```
fill(0,25,88);
textAlign(CENTER,CENTER);
textSize(height*0.8);
```

The `fill()` function used in `setup()` can fill the text color, `textAlign(CENTER,CENTER)` is used to center the text, `textSize(height*0.8)` change the text height to 0.8 times the original. These functions can customize the text style displayed on the processing

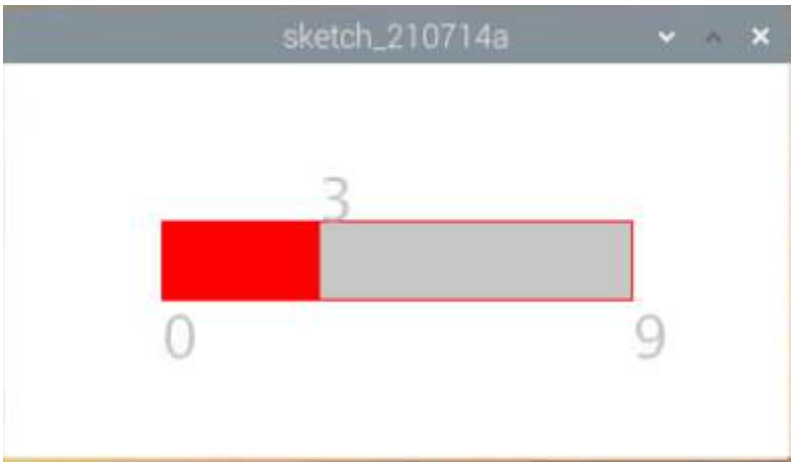
```
void draw() {

        background(255);
        int number = (frameCount%100)/10;
        text(number, width/2, height/2);
        hc595_shift(SegCode[number]);
}
```

The `frameCount` is a seed, which is related to `frameRate`. By default `frameRate` is 60, which means that `frameCount` will accumulate 60 times per second.

Then we can let processing and 7-segment display to show the figure from 0 to 9 and A to F simultaneously.

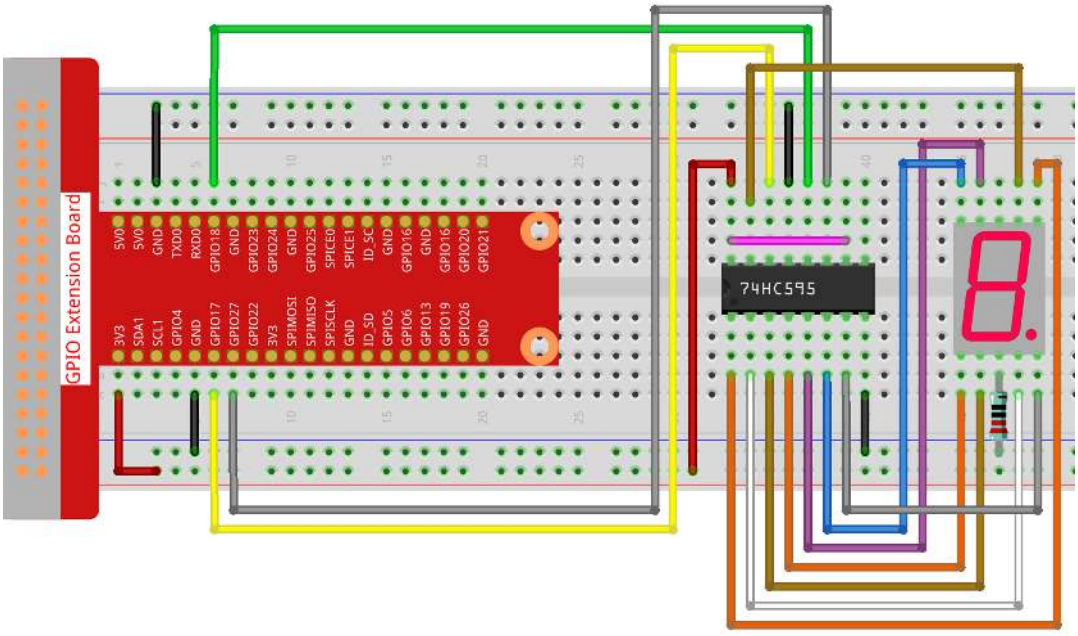
8.4.10 Drag Number

Let's draw a slider bar to control the 7-segment Display.



Wiring

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *7-segment Display*
- *74HC595*



Sketch

```
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);
}
```

(continues on next page)

(continued from previous page)

```
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]);
}

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);
```

(continues on next page)

(continued from previous page)

```
        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;
    }
}
```

How it works?

This project integrates the Slider and 7-segment Display of the previous project. For specific knowledge points, please refer to *Show Number* and *Metronome*.

CHAPTER

NINE

PLAY WITH NODEJS

9.1 What is Nodejs

Node.js was released in May 2009 and developed by Ryan Dahl. It is a JavaScript runtime environment based on the Chrome V8 engine. It uses an event-driven, non-blocking I/O model to allow JavaScript to run on the server-side development platform.

Simply put, Node.js is JavaScript running on the server. If you are familiar with Javascript, then you will easily learn Node.js.

Nodejs usually uses the command `npm install xxx` to install third-party packages, which requires us to install the npm tool, similar to the pip tool in python.

9.2 Install or update nodejs and npm

Run the following commands to install and update nodejs and npm.

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
sudo npm install npm -g
```

Then check the current Node version with the following command.

```
node -v
```

The following command checks the current npm version.

```
npm -v
```

9.3 Check the pigpio

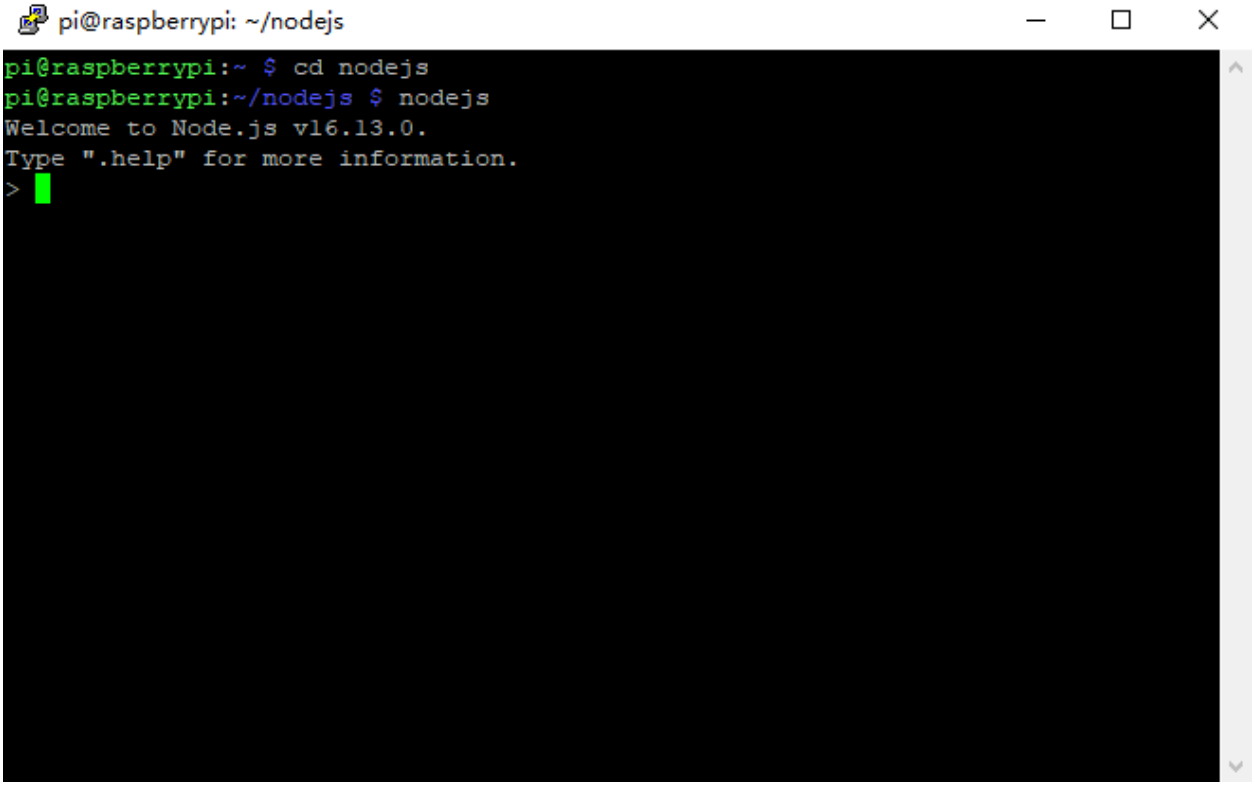
pigpio is a module used to control Raspberry Pi GPIO channels. This package provides some methods to control GPIO on Raspberry Pi. For examples and documentation, please visit: https://www.npmjs.com/package/pigpio.

Enter the following command to install the pigpio library.

```
npm install pigpio
```

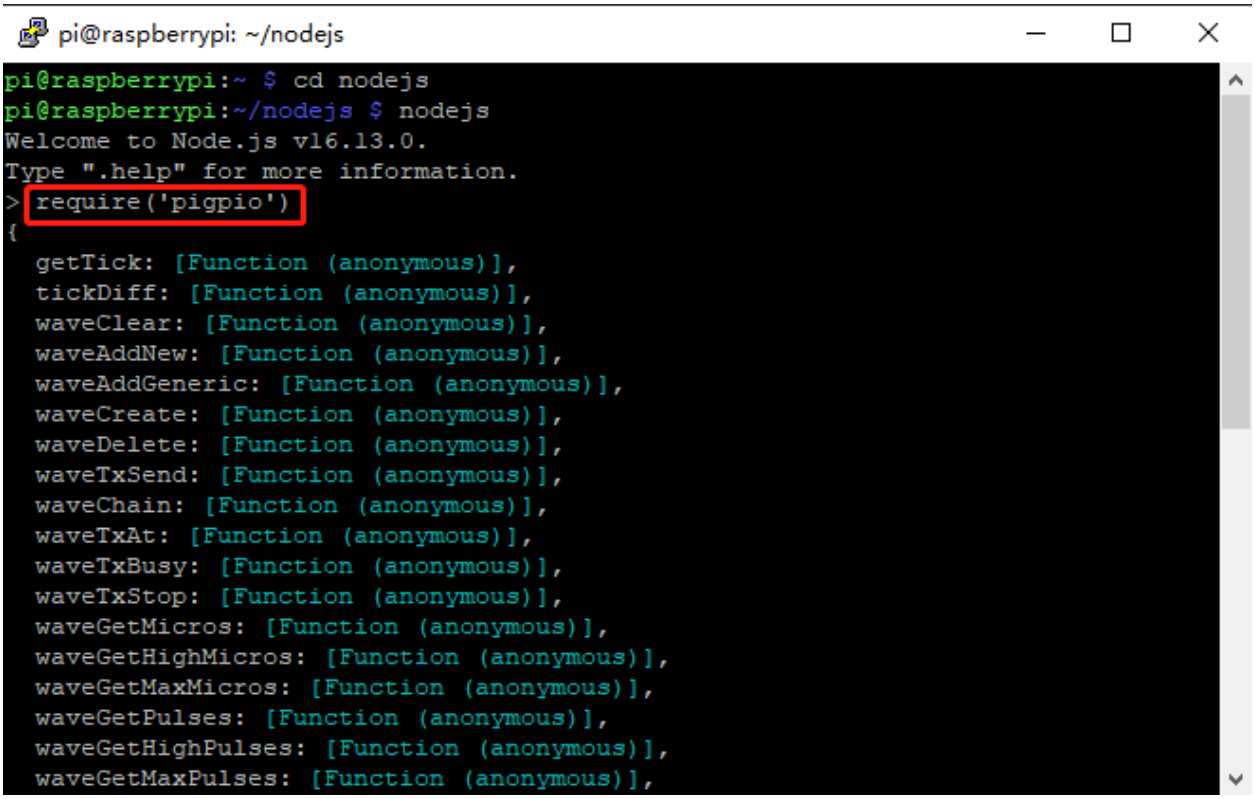
Check if the library is installed successfully, change the directory and enter nodejs:

```
cd /home/pi/raphael-kit/nodejs
nodejs
```



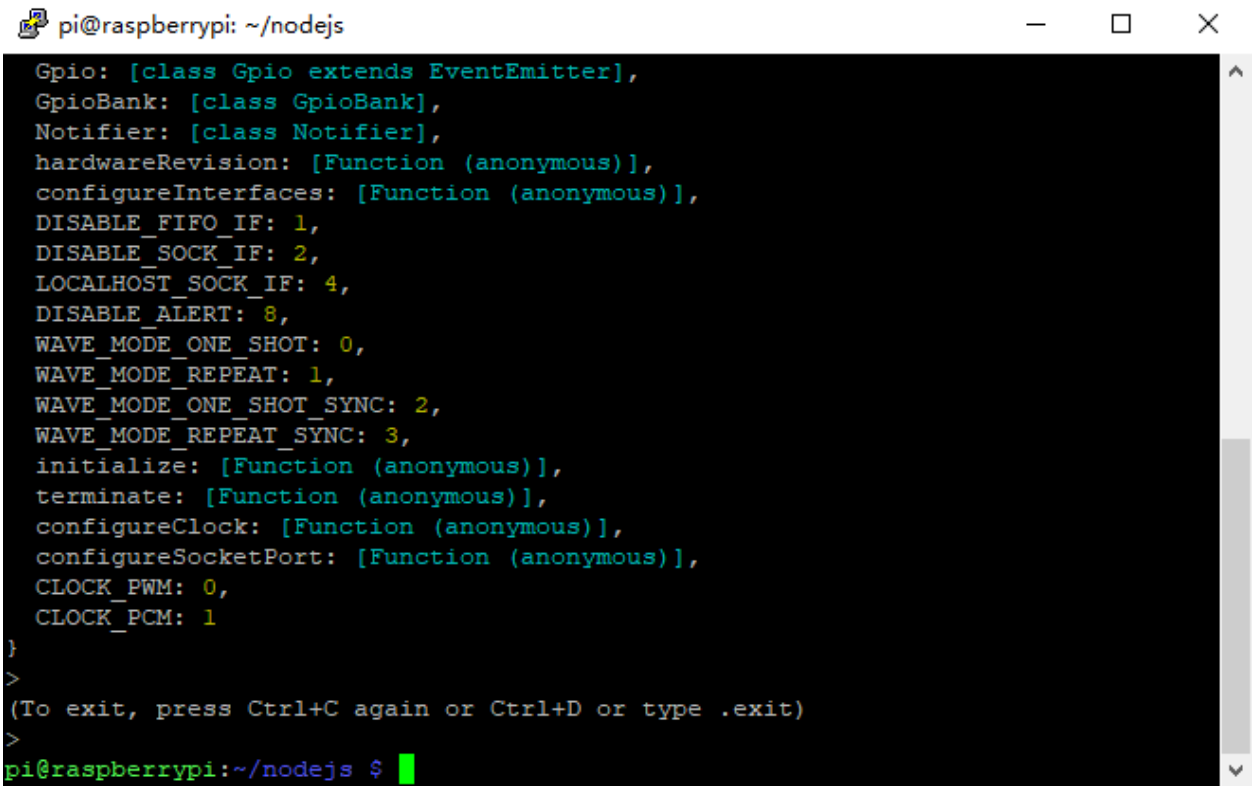
Then enter require('pigpio'):

```
require('pigpio')
```



If the above screen appears, the library installation is successful.

If you want to exit node CLI, please press Ctrl+C twice.



9.4 Output

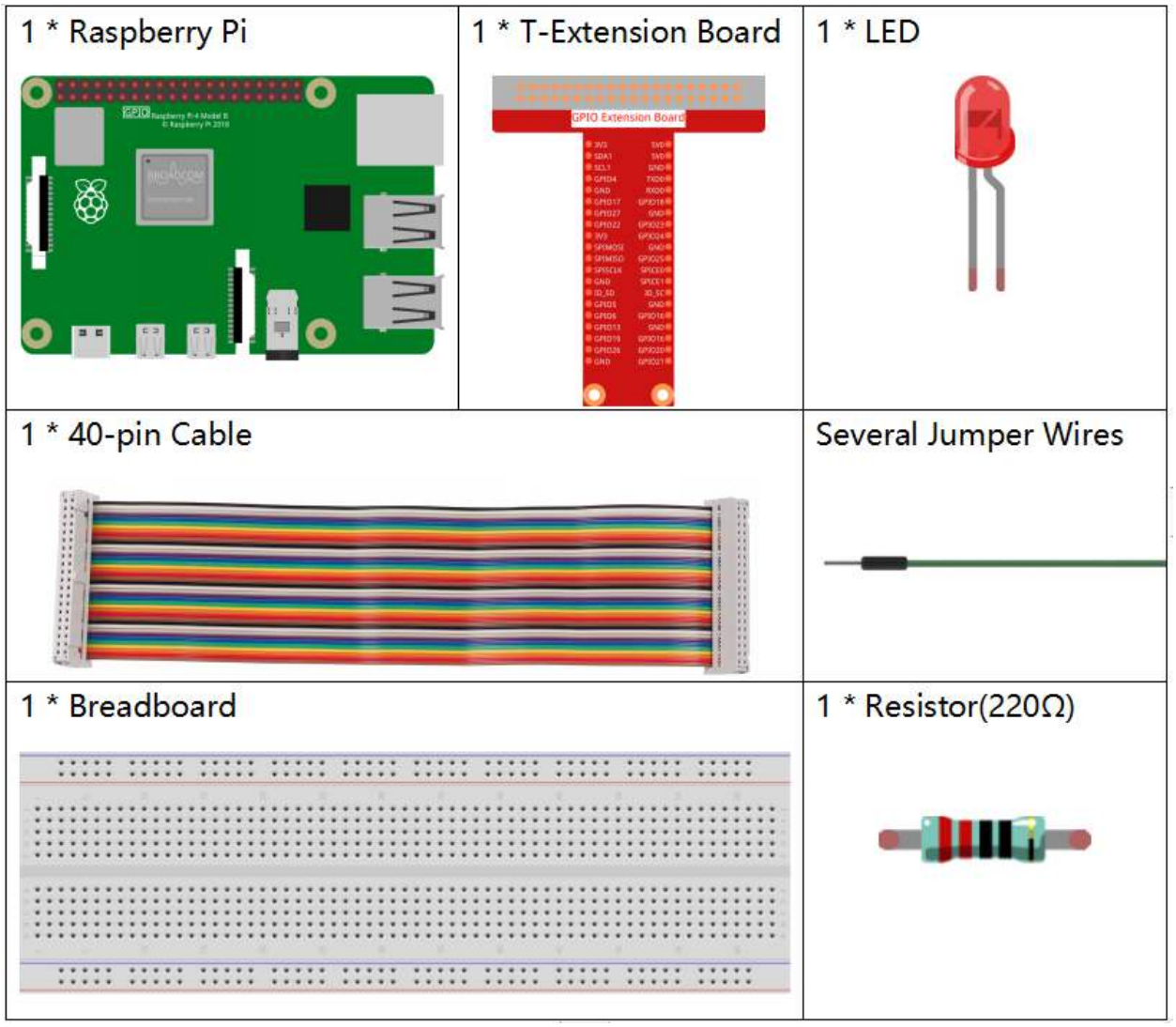
9.4.1 1.1 Displays

1.1.1 Blinking LED

Introduction

In this project, we will learn how to make a blinking LED by programming. Through your settings, your LED can produce a series of interesting phenomena. Now, go for it.

Components

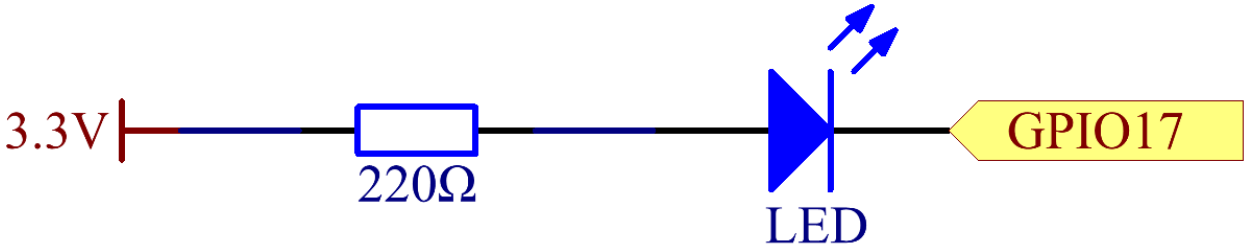


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*

- *LED*

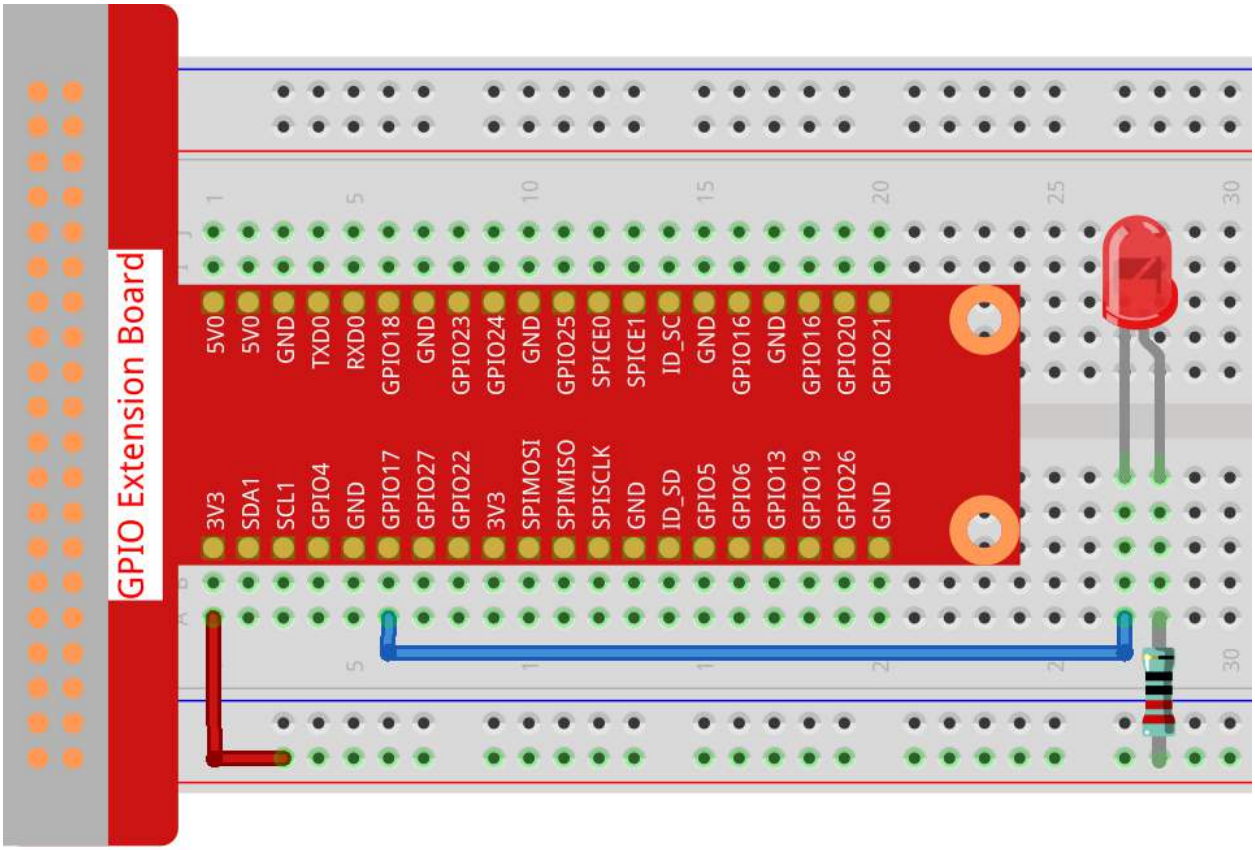
Schematic Diagram

In this experiment, connect a 220 resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to GPIO17 of Raspberry Pi. Therefore, to turn on an LED, we need to make GPIO17 low (0V) level. We can get this phenomenon by programming.



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Note: Change directory to the path of the code in this experiment via `cd`.

Step 3: Run the code

```
sudo node blink.js
```

Note: Here sudo - superuser do, and python means to run the file by Python.

After the code runs, you will see the LED flashing.

Step 4: If you want to edit the code file blink.js, press `Ctrl + C` to stop running the code. Then type the following command to open blink.js:

```
nano blink.js
```

Note: nano is a text editor tool. The command is used to open the code file blink.js by this tool.

Press `Ctrl+X` to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in `Y` (save) or `N` (don't save).

Then press `Enter` to exit. Type in nano blink.js again to see the effect after the change.

Code

The following is the program code:

```
const Gpio = require('pigpio').Gpio;
const led = new Gpio(17,{mode: Gpio.OUTPUT});

var led_state = 0;

function blink_led(){
   led.digitalWrite(led_state);
   led_state = !led_state;
}

setInterval(blink_led,300);
```

Alternatively, write the code as a more js-specific arrow function:

```
const Gpio = require('pigpio').Gpio;
const led = new Gpio(17,{mode: Gpio.OUTPUT});

var led_state = 0;

setInterval(() => {
   led.digitalWrite(led_state);
   led_state = !led_state;
}, 300);
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;
```

Import the Gpio constructor in the pigpio package in this way, and then define a constant `Gpio` to represent this constructor.

With a variety of constructor, we can use js to control electronic devices. `pigpio` can be used to implement fast GPIO, PWM, servo control, state change notification and interrupt handling.

- pigpio - github

```
const led = new Gpio(17,{mode: Gpio.OUTPUT});
```

The `new` keyword is used to construct instantiated objects of the class.

Connect the LED to the GPIO17 of the T-shaped expansion board, set the LedPin mode to output and assign it to the constant `led`, that is, construct a GPIO17 object `led`, and its mode is the output mode.

There are two ways to number the IO pins on the Raspberry Pi: BOARD number and BCM number. In our project, we use the BCM number. You need to set each channel used as input or output.

```
function blink_led(){
   led.digitalWrite(led_state);
   led_state = !led_state;
}

setInterval(blink_led,300);
```

The `setInterval()` method can call functions or compute expressions with a specified period (in milliseconds). Here we change the operating state of the LED with a period of 300ms.

- setInerval

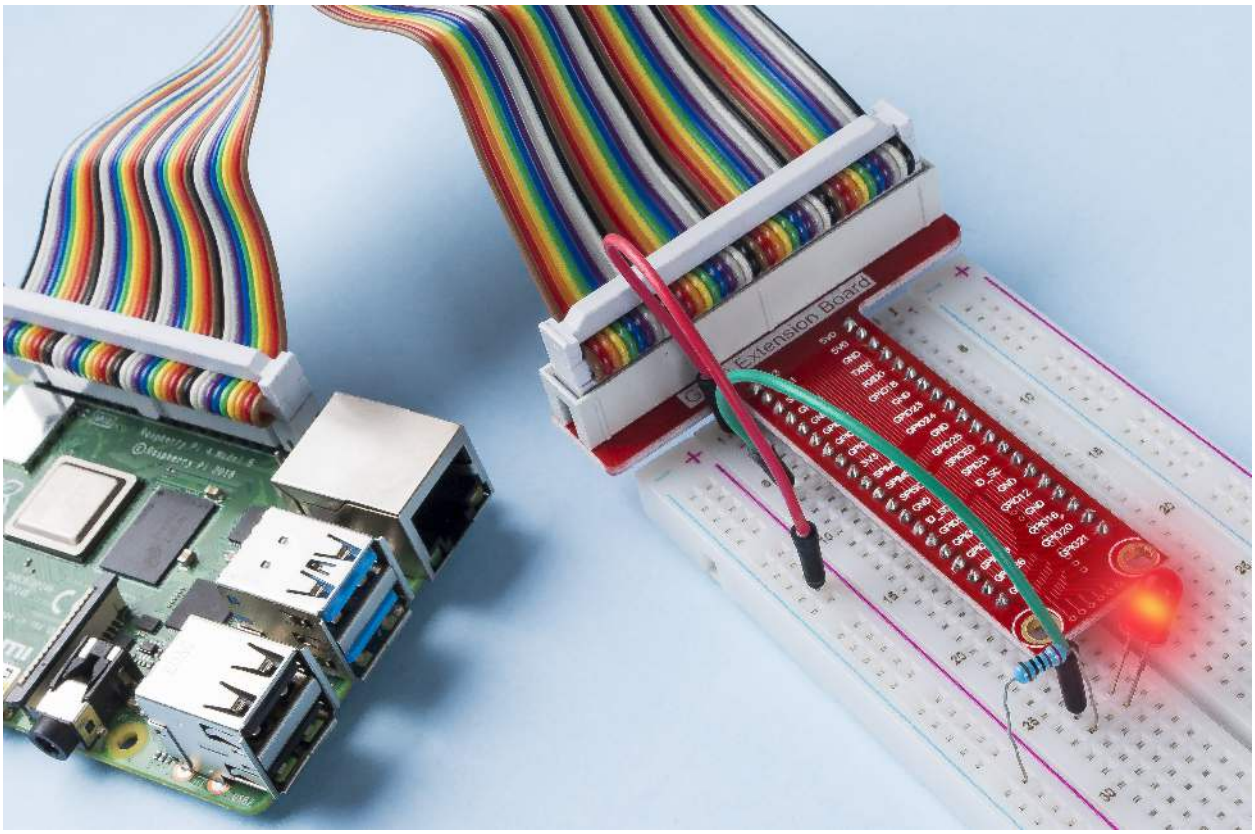
The `gpio.digitalWrite(level)` method sets the gpio level to 1 or 0.

```
var led_state = 0;

setInterval(() => {
   led.digitalWrite(led_state);
   led_state = !led_state;
}, 300);
```

Rewrite the code as an arrow function to make the code shorter.

- Arrow Functions

Phenomenon Picture

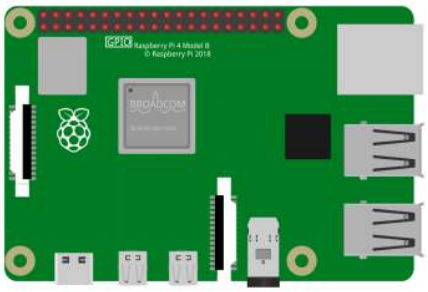
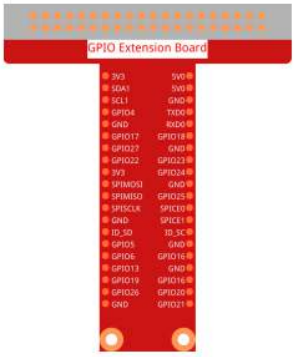
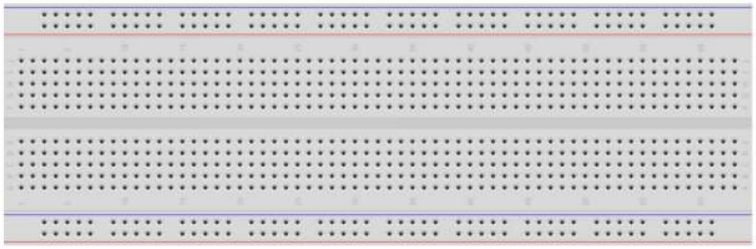


1.1.2 RGB LED

Introduction

In this project, we will control an RGB LED to flash various colors.

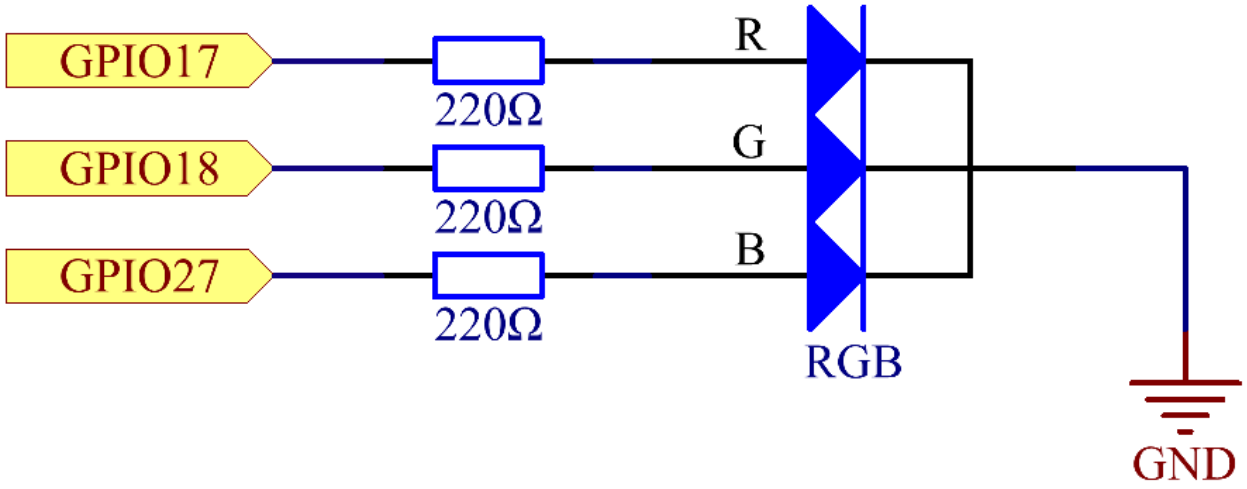
Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*

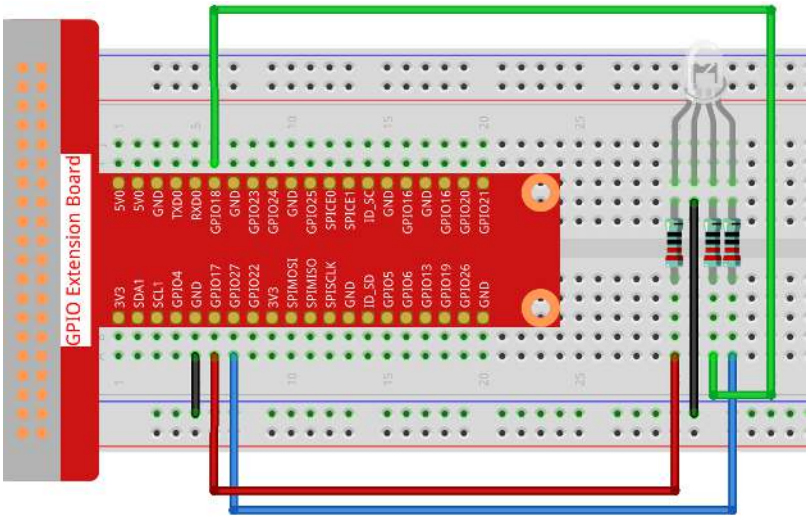
Schematic Diagram

After connecting the pins of R, G, and B to a current limiting resistor, connect them to the GPIO17, GPIO18, and GPIO27 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node rgb_led.js
```

After the code runs, you will see that RGB displays red, green, blue, yellow, pink, and cyan.

Code

```
const Gpio = require('pigpio').Gpio;
const ledred = new Gpio(17, { mode: Gpio.OUTPUT });
const ledgreen = new Gpio(18, { mode: Gpio.OUTPUT });
const ledblue = new Gpio(27, { mode: Gpio.OUTPUT });

function colorset(r, g, b) {
    ledred.pwmWrite(r);
    ledgreen.pwmWrite(g);
    ledblue.pwmWrite(b);
}

var color_index = -1;

setInterval(() => {
color_index += 1;
switch (color_index) {
    case 0:
        colorset(0xff, 0x00, 0xFF);
        break;
    case 1:
        colorset(0x00, 0xff, 0x00);
        break;
    case 2:
        colorset(0x00, 0x00, 0xff);
        break;
    case 3:
        colorset(0xff, 0xff, 0x00);
        break;
    case 4:
        colorset(0xff, 0x00, 0xff);
        break;
    case 5:
        colorset(0xc0, 0xff, 0x3e);
        break;
    default:
        color_index=-1;
    }
}, 500);
```

Code Explanation

```
const ledred = new Gpio(17,{mode: Gpio.OUTPUT});
const ledgreen = new Gpio(18,{mode: Gpio.OUTPUT});
const ledblue = new Gpio(27,{mode: Gpio.OUTPUT});
```

Initialize pins 17, 18, and 27 to output mode, and assign them to the constants `ledred`, `ledgreen`, and `ledblue` respectively.

```
function colorset(r,g,b){
    ledred.pwmWrite(r);
    ledgreen.pwmWrite(g);
    ledblue.pwmWrite(b);
}
```

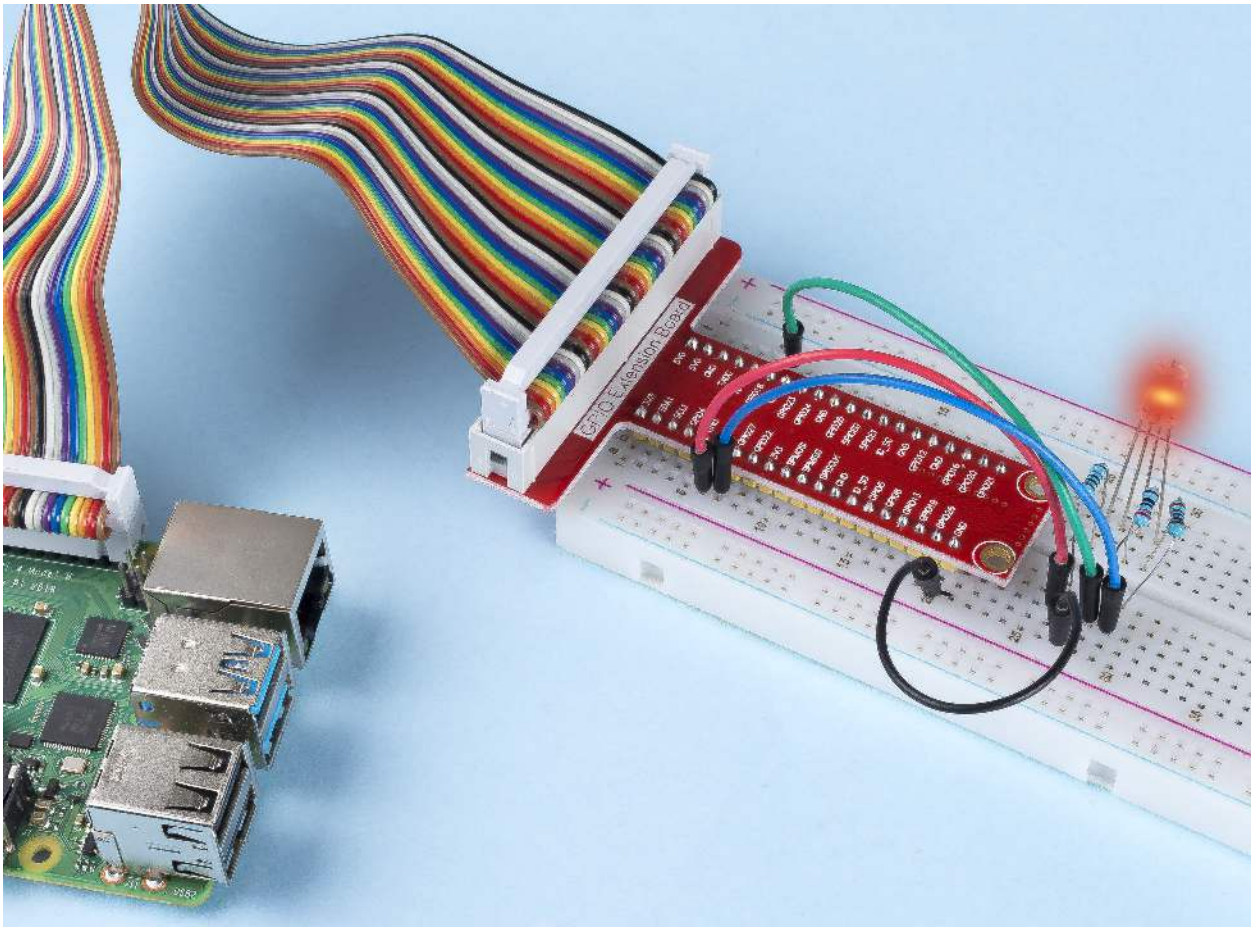
Implement a `colorset(r,g,b)` function, which is used to write pulse values to pins 17, 18, 27. The Gpio library encapsulates the function `pwmWrite()` to write to pins Pulse value, the value is 0x00 to 0xff. Then you can write RGB values to the RGB LED through the `colorset(r,g,b)` function, so that it can display a variety of colors.

Note: For questions about RGB, please refer to the website: https://www.rapidtables.com/web/color/RGB_Color.html

```
var color_index = -1;

setInterval(() => {
color_index += 1;
switch (color_index) {
    case 0:
        colorset(0xff, 0x00, 0xFF);
        break;
    case 1:
        colorset(0x00, 0xff, 0x00);
        break;
    case 2:
        colorset(0x00, 0x00, 0xff);
        break;
    case 3:
        colorset(0xff, 0xff, 0x00);
        break;
    case 4:
        colorset(0xff, 0x00, 0xff);
        break;
    case 5:
        colorset(0xc0, 0xff, 0x3e);
        break;
    default:
        color_index=-1;
    }
}, 500);
```

The RGB LED is controlled by `colorset()` executed every 500ms. A switch case is used here to select the color emitted by the RGB LEDs. Since `color_index` is changed by one every cycle, the color of this one RGB LED will change in order.

Phenomenon Picture

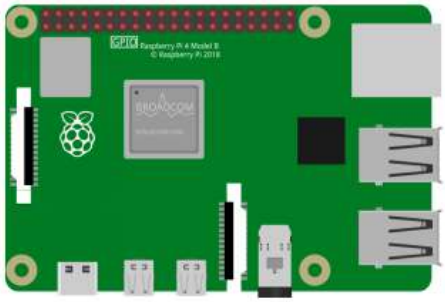
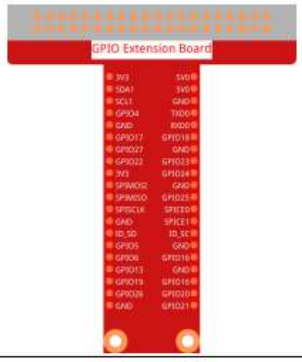
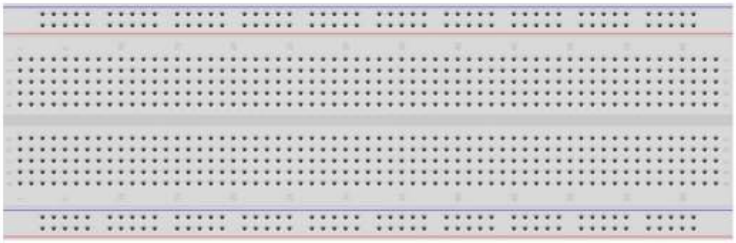


1.1.3 LED Bar Graph

Introduction

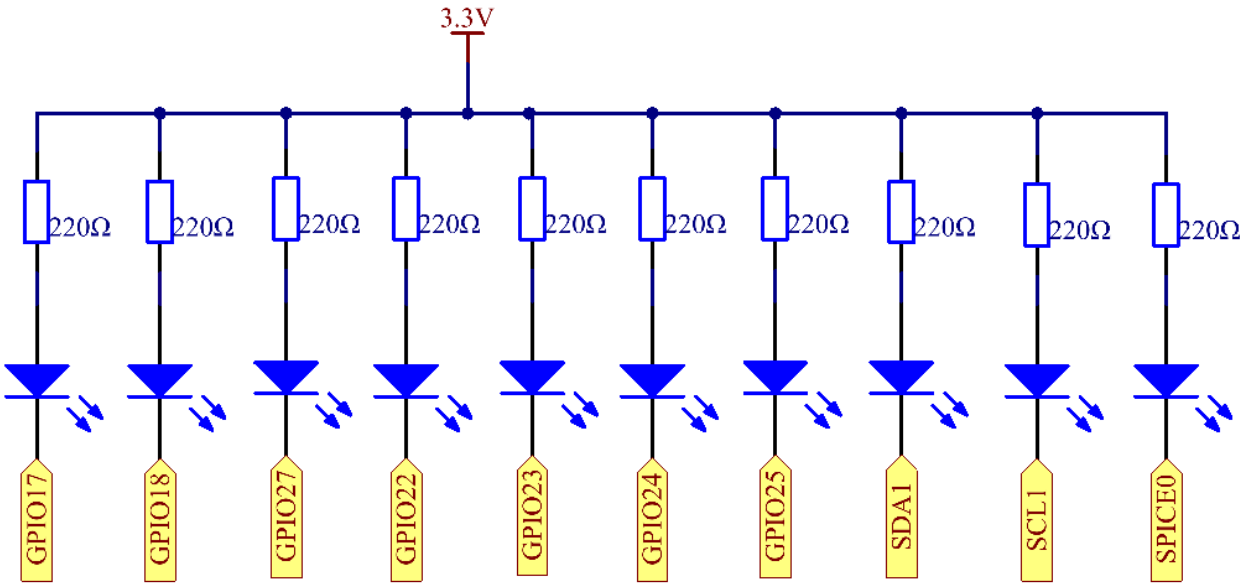
In this project, we sequentially illuminate the lights on the LED Bar Graph.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * LED Bargraph
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>Several Jumper Wires
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>10 * Resistor(220Ω)
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*

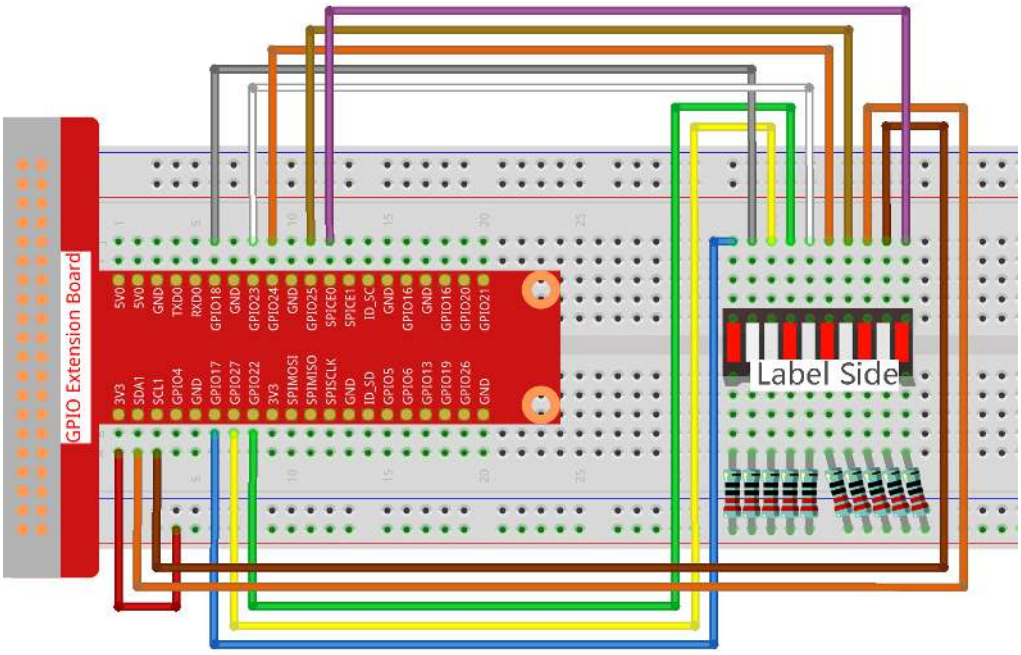
Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.

Note: Pay attention to the direction when connecting. If you connect it backwards, it will not light up.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Compile the code.

```
sudo node led_bar_graph.js
```

After the code runs, you will see the LEDs on the LED bar turn on and off regularly.

Code

```
const Gpio = require('pigpio').Gpio;

var pins = [17, 18, 27, 22, 23, 24, 25, 2, 3, 8];
var leds = [];
for (let i = 0; i < pins.length; i++) {
    leds[i] = new Gpio(pins[i], { mode: Gpio.OUTPUT });
}

function oddLedBarGraph() {
    for (let i = 0; i < leds.length; i++) {
        if (i % 2 == 1) {
            leds[i].digitalWrite(1);
        } else {
            leds[i].digitalWrite(0);
        }
    }
}

function evenLedBarGraph() {
    for (let i = 0; i < leds.length; i++) {
        if (i % 2 == 0) {
            leds[i].digitalWrite(1);
        } else {
            leds[i].digitalWrite(0);
        }
    };
}

var odd_even = 0;

setInterval(() => {
    odd_even = (odd_even + 1) % 2;
    if (odd_even == 1) {
        oddLedBarGraph();
    } else {
        evenLedBarGraph();
    }
}, 500);
```

Code Explanation

```
var pins = [17,18,27,22,23,24,25,2,3,8];
```

Because the led bar graph will use multiple pins, we create a constant array `pins` to store them in batches.

```
var leds = [];
for (let i = 0; i < pins.length; i++) {
```

(continues on next page)

(continued from previous page)

```
    leds[i] = new Gpio(pins[i], { mode: Gpio.OUTPUT });
}
```

Instantiate these pins as leds objects with a for loop.

```
function oddLedBarGraph() {
    for (let i = 0; i < leds.length; i++) {
        if (i % 2 == 1) {
            leds[i].digitalWrite(1);
        } else {
            leds[i].digitalWrite(0);
        }
    }
}
```

Implement an `oddLedBarGraph()` function to make the LEDs on the odd digits of the LED bar graph light up.

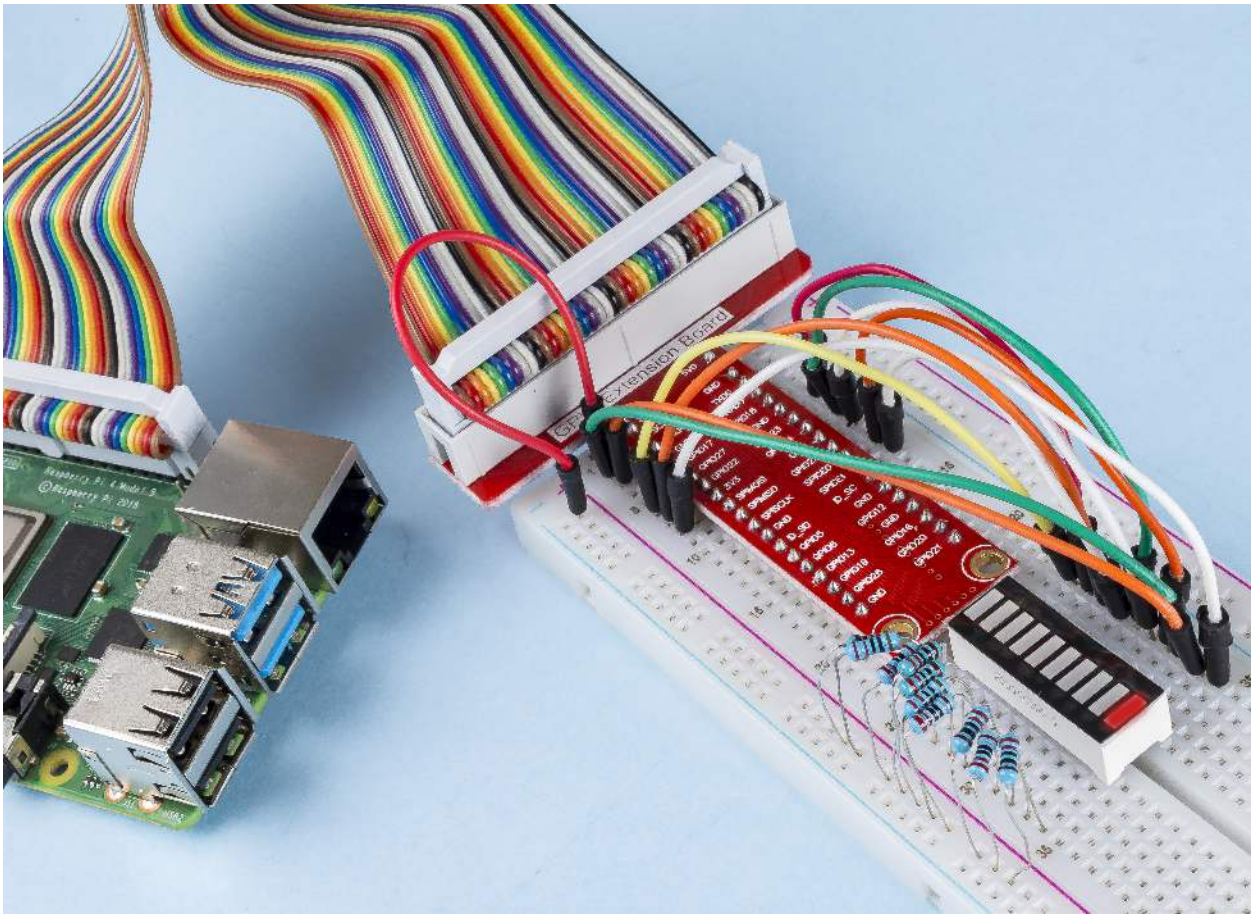
```
function evenLedBarGraph() {
    for (let i = 0; i < leds.length; i++) {
        if (i % 2 == 0) {
            leds[i].digitalWrite(1);
        } else {
            leds[i].digitalWrite(0);
        }
    };
}
```

Implement an `evenLedBarGraph()` function to make the LEDs on the even digits of the LED bar graph light up.

```
var odd_even = 0;

setInterval(() => {
    odd_even = (odd_even + 1) % 2;
    if (odd_even == 1) {
        oddLedBarGraph();
    } else {
        evenLedBarGraph();
    }
}, 500);
```

The working state of the LED is switched every 500ms.

Phenomenon Picture

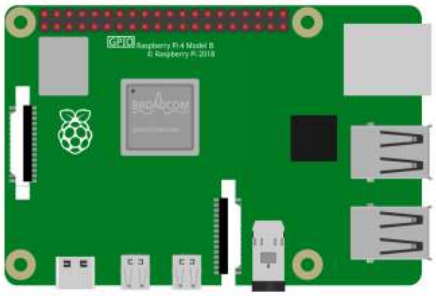
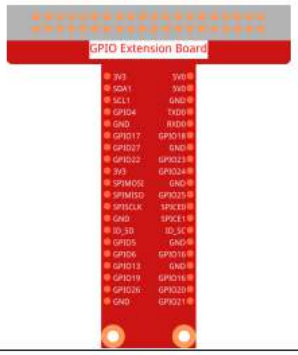
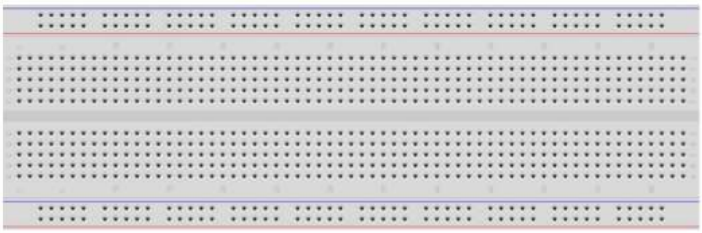


1.1.4 7-segment Display

Introduction

Let's try to drive a 7-segment display to show a figure from 0 to 9 and A to F.

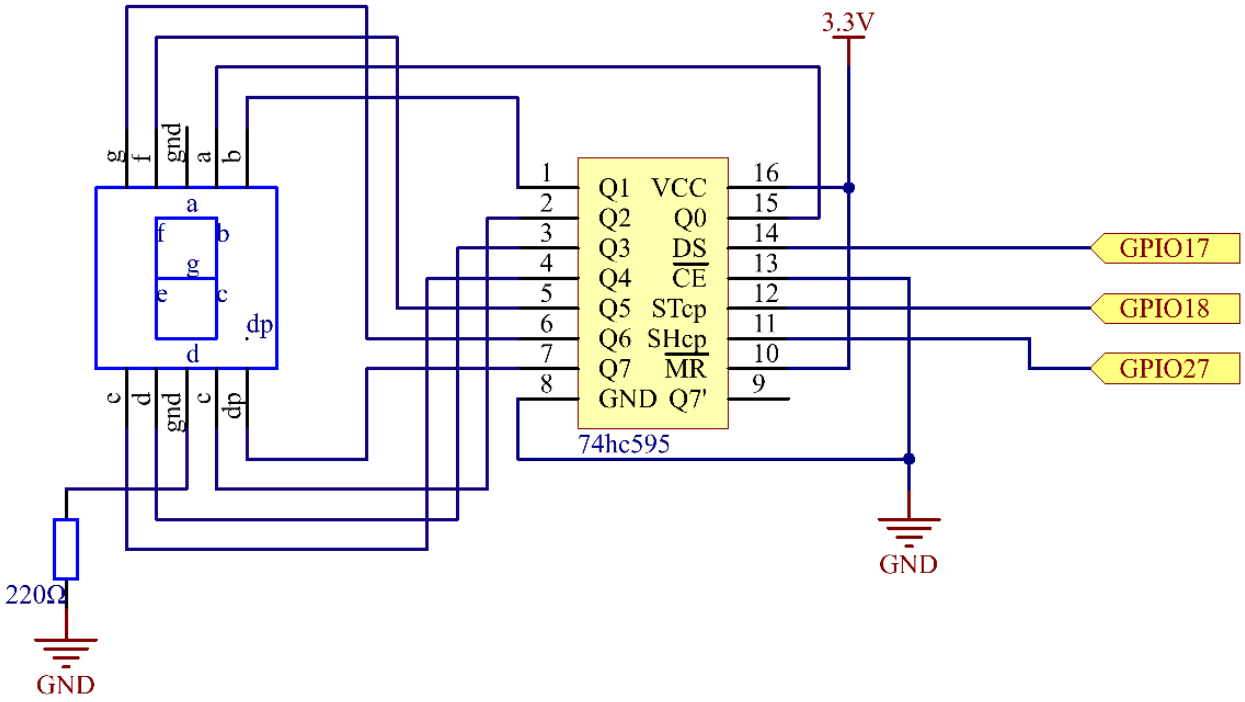
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 7-segment display
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * 74HC595
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *7-segment Display*
- *74HC595*

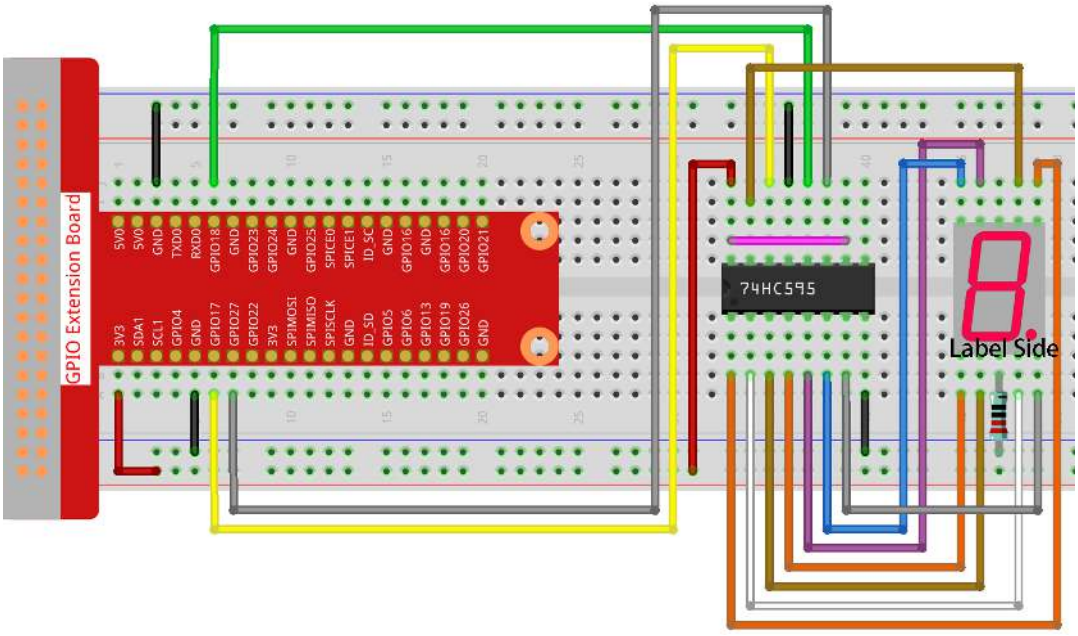
Schematic Diagram

Connect pin ST_CP of 74HC595 to Raspberry Pi GPIO18, SH_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST_CP (the clock input of the memory) is at the rising edge. Then you can control the states of SH_CP and ST_CP via the Raspberry Pi GPIOs to transform serial data input into parallel data output so as to save Raspberry Pi GPIOs and drive the display.



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node 7-segment_display.js
```

After the code runs, you'll see the 7-segment display display 0-9, A-F.

Code

```
const Gpio = require('pigpio').Gpio;

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

const SDI = new Gpio(17, { mode: Gpio.OUTPUT });
const RCLK = new Gpio(18, { mode: Gpio.OUTPUT });
const SRCLK = new Gpio(27, { mode: Gpio.OUTPUT });

function hc595_shift(dat) {
    for (let j = 0; j < 8; j++) {
        let code = 0x80 & (dat << j);
        if (code != 0) {
            code = 1;
        }
        SDI.digitalWrite(code);
        SRCLK.trigger(1,1);
    }
    RCLK.trigger(1,1);
}

let index = -1;
setInterval(() => {
    index = (index+1)%16;
```

(continues on next page)

(continued from previous page)

```
    hc595_shift(segCode[index]);
}, 1000);
```

Code Explanation

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

Define a hexadecimal (common cathode) segment code array from 0 to F.

```
const SDI = new Gpio(17, { mode: Gpio.OUTPUT });
const RCLK = new Gpio(18, { mode: Gpio.OUTPUT });
const SRCLK = new Gpio(27, { mode: Gpio.OUTPUT });
```

Initialize pins 17, 18, and 27 as output mode, and copy them to `SDI`, `RCLK`, and `SRCLK` respectively.

```
function hc595_shift(dat) {
    for (let j = 0; j < 8; j++) {
        let code = 0x80 & (dat << j);
        if (code != 0) {
            code = 1;
        }
        SDI.digitalWrite(code);
        SRCLK.trigger(1,1);
    }
    RCLK.trigger(1,1);
}
```

Implement a `hc595_shift` function to convert the fields in the array `segCode` into numbers and display them on the digital tube.

```
let code = 0x80 & (dat << j);
if (code != 0) {
    code = 1;
}
SDI.digitalWrite(code);
```

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when j=2, 0x3f will shift right(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true. At this time, 1 is written to SDI.

```
SRCLK.trigger(1,1);
```

Generate a rising edge pulse and move the DS data to the shift register.

`trigger(pulseLen, level)`

- pulseLen - pulse length in microseconds (1 - 100)
- level - 0 or 1

Sends a trigger pulse to the GPIO. The GPIO is set to level for pulseLen microseconds and then reset to not level.

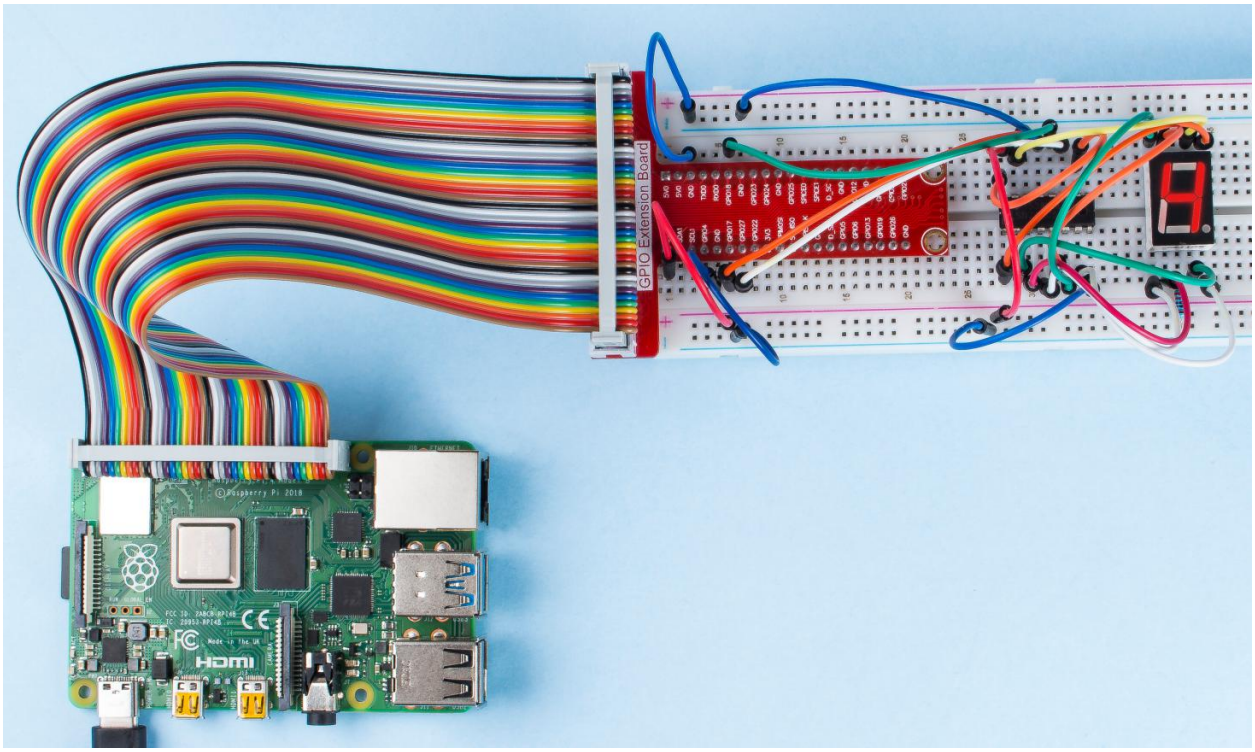
```
RCLK.trigger(1,1);
```

Generate a rising edge pulse and move the data from the shift register to the storage register.

```
let index = -1;
setInterval(() => {
    index = (index+1)%16;
    hc595_shift(segCode[index]);
}, 1000);
```

Finally, use the function `hc595_shift()` to convert the fields in `segCode` and display them through the digital tube.

Phenomenon Picture

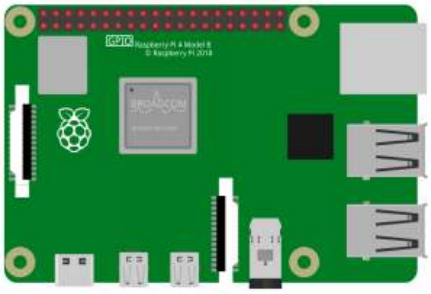
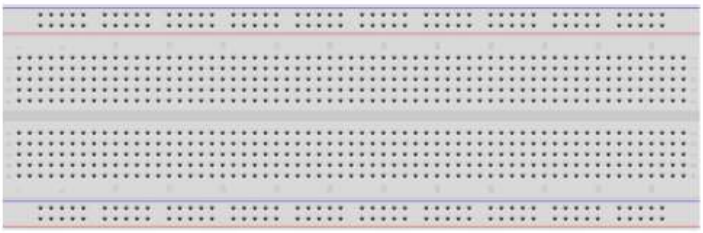


1.1.5 4-Digit 7-Segment Display

Introduction

Next, follow me to try to control the 4-digit 7-segment display.

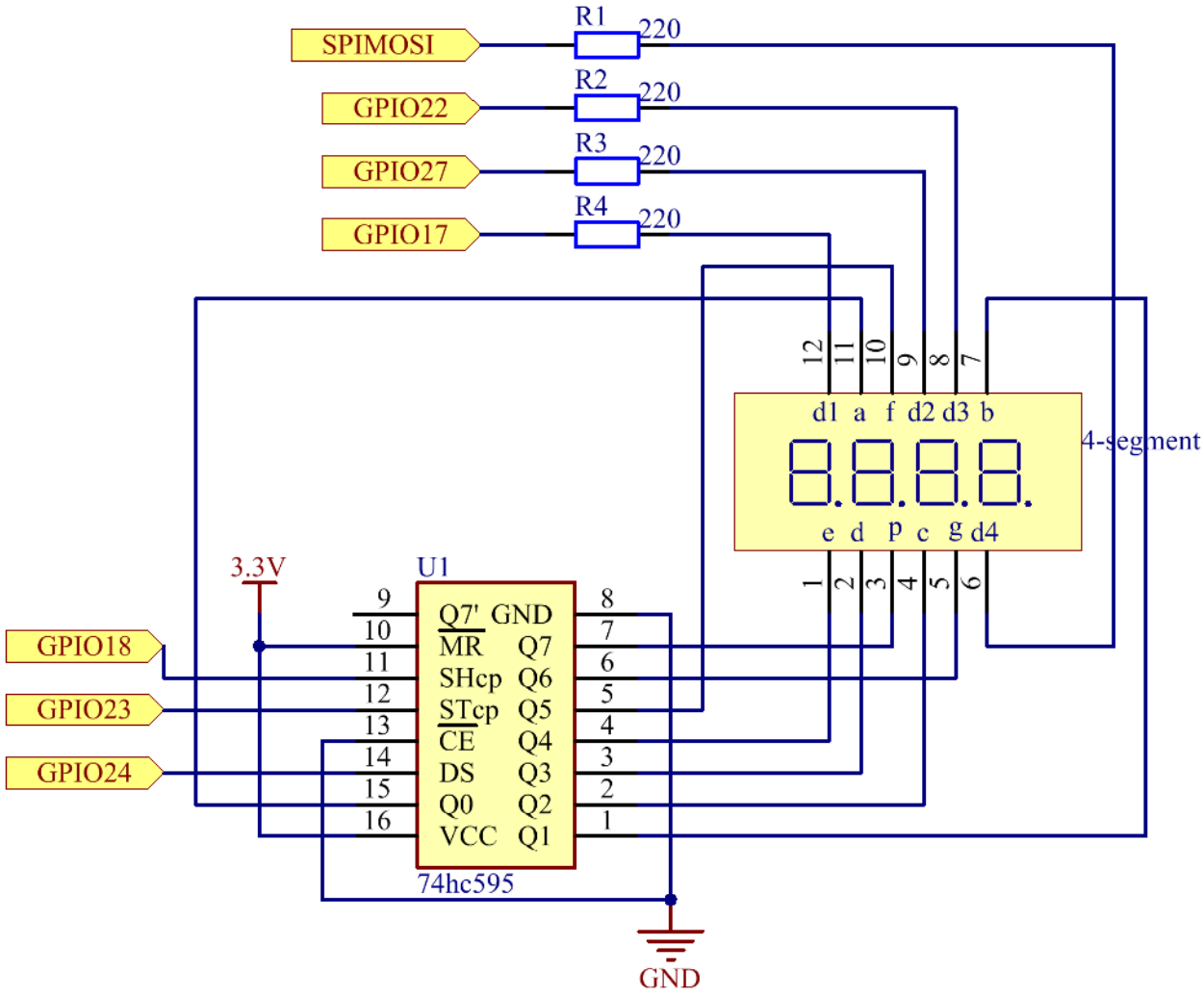
Components

<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>4 * Resistor(220Ω)
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>Several Jumper Wires
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *4-Digit 7-Segment Display*
- *74HC595*

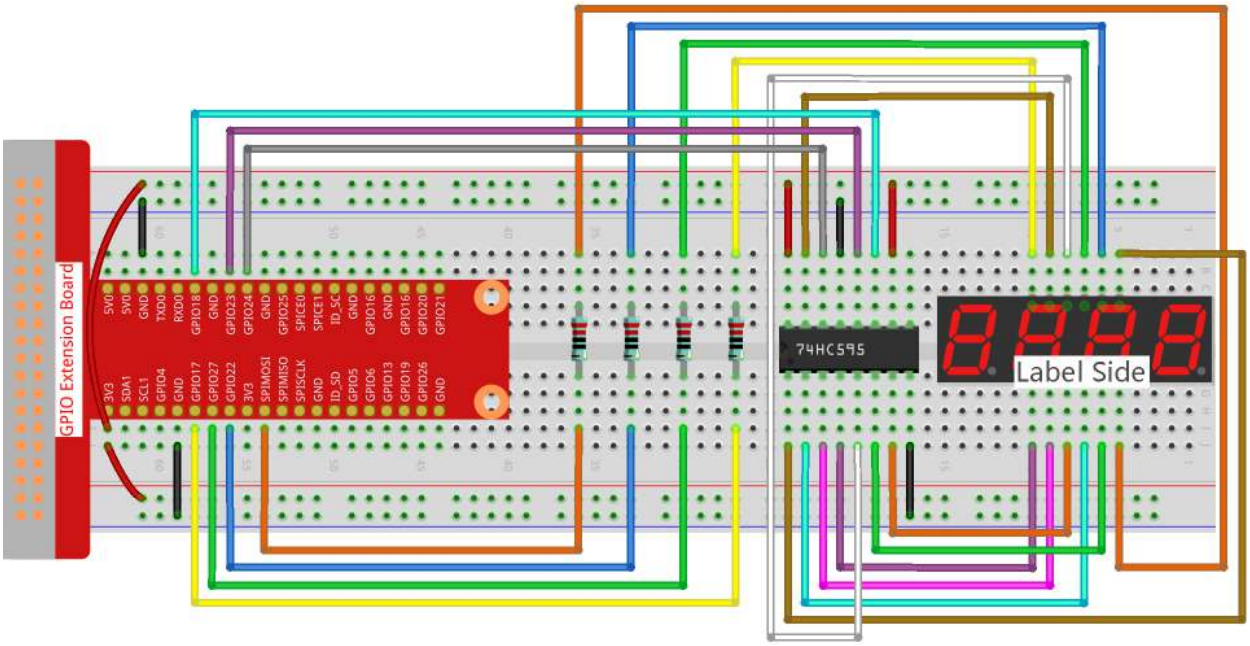
Note: In this projiect, for the 4-Digit 7-Segment Display we should use BS model,if you use AS model it may not light up.

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node 4_digit_7_segment_display.js
```

After the code runs, the program takes a count, increasing by 1 per second, and the 4-digit 7-segment display displays the count.

Code

```
const Gpio = require('pigpio').Gpio;

var counter = 0;

const number = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90]; //for BS

const SDI = new Gpio(24, { mode: Gpio.OUTPUT });
const RCLK = new Gpio(23, { mode: Gpio.OUTPUT });
const SRCLK = new Gpio(18, { mode: Gpio.OUTPUT });

const pin1 = new Gpio(10, { mode: Gpio.OUTPUT });
const pin2 = new Gpio(22, { mode: Gpio.OUTPUT });
const pin3 = new Gpio(27, { mode: Gpio.OUTPUT });
const pin4 = new Gpio(17, { mode: Gpio.OUTPUT });
const placePin = [pin1, pin2, pin3, pin4];

function clearDisplay() {
    hc595_shift(0xff); //for BS
}

function hc595_shift(dat) {
    for (let j = 0; j < 8; j++) {
```

(continues on next page)

(continued from previous page)

```
        let code = 0x80 & (dat << j);
        if (code != 0) {
            code = 1;
        }
        SDI.digitalWrite(code);
        SRCLK.trigger(1,1);
    }
    RCLK.trigger(1,1);
}

function pickDigit(digit) {
    for(let i=0;i<4;i++){
        placePin[i].digitalWrite(0);
    }
    placePin[digit].digitalWrite(1);
}


let digit = -1
setInterval(() => {
    digit = (digit +1)% 4
    clearDisplay();
    pickDigit(digit);
    switch(digit){
        case 0:
            hc595_shift(number[Math.floor(counter % 10)]);
            break;
        case 1:
            hc595_shift(number[Math.floor(counter % 100 / 10)]);
            break;
        case 2:
            hc595_shift(number[Math.floor(counter % 1000 / 100)]);
            break;
        case 3:
            hc595_shift(number[Math.floor(counter % 10000 / 1000)]);
            break;
    }
}, 5);

setInterval(() => {
    counter++;
}, 1000);
```

Code Explanation

```
const pin1 = new Gpio(10, {mode: Gpio.OUTPUT});
const pin2 = new Gpio(25, {mode: Gpio.OUTPUT});
const pin3 = new Gpio(27, {mode: Gpio.OUTPUT});
const pin4 = new Gpio(17, {mode: Gpio.OUTPUT});
const placePin = [pin1,pin2,pin3,pin4];
```

Initialize pins 10, 25, 27, and 17 as output modes and place them in the array `placePin` to facilitate control of the common anode of the four-digit 7-segment display.

```
const number = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90];
```

Define a constant array `number` to represent the hexadecimal segment code from 0 to 9 (common anode).

```
function clearDisplay() {
    hc595_shift(0xff);
}
```

Write 0xff to turn off the digital tube.

```
function pickDigit(digit) {
    for(let i=0;i<4;i++){
        placePin[i].digitalWrite(0);
    }
    placePin[digit].digitalWrite(1);
}
```

Select the place of the value. there is only one place that should be enable each time. The enabled place will be written high.

```
let digit = -1
setInterval(() => {
    digit = (digit +1)% 4
    clearDisplay();
    pickDigit(digit);
    switch(digit){
        case 0:
            hc595_shift(number[Math.floor(counter % 10)]);
            break;
        case 1:
            hc595_shift(number[Math.floor(counter % 100 / 10)]);
            break;
        case 2:
            hc595_shift(number[Math.floor(counter % 1000 / 100)]);
            break;
        case 3:
            hc595_shift(number[Math.floor(counter % 10000 / 1000)]);
            break;
    }
}, 5);
```

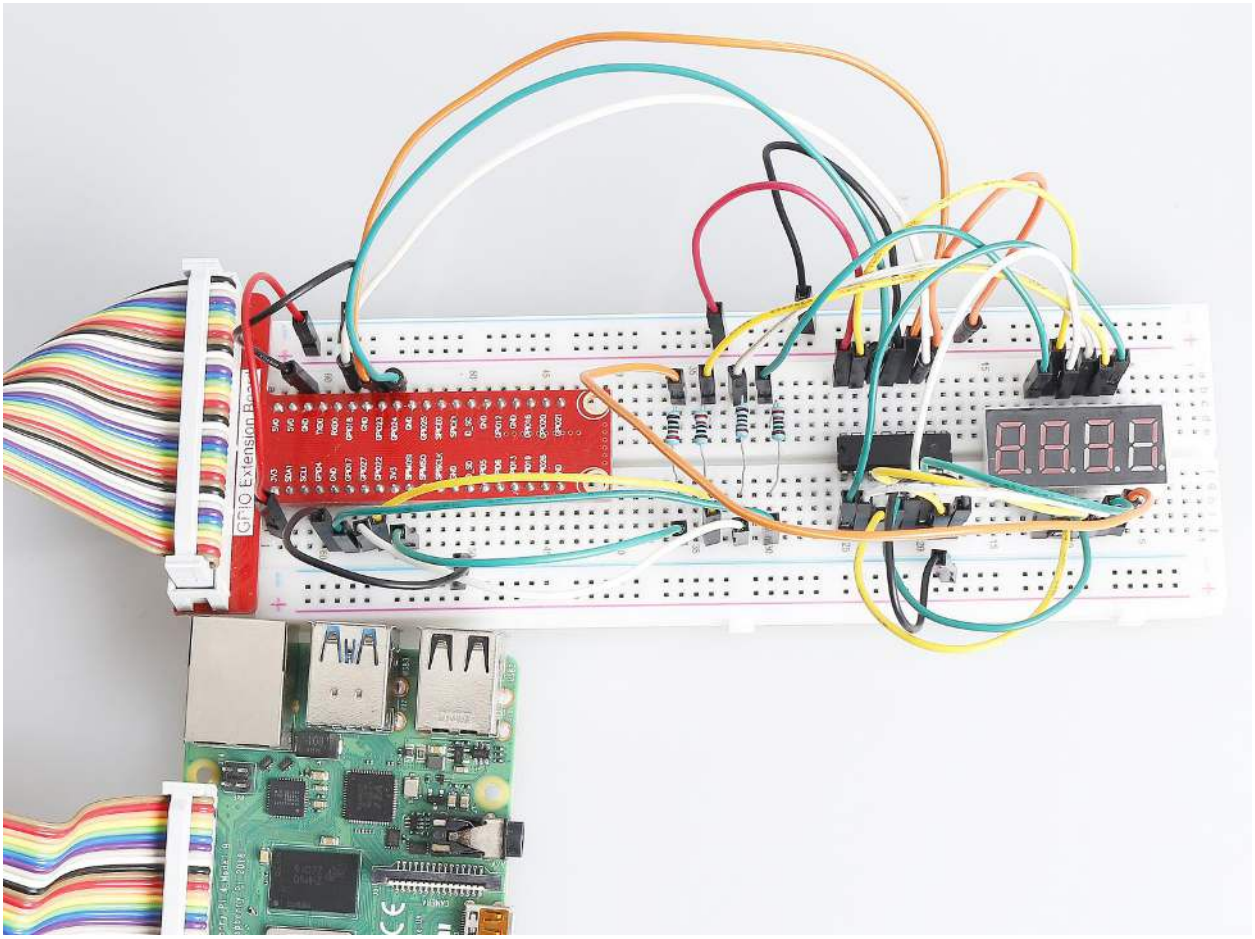
this code is used to set the number displayed on the 4-digit 7-segment Dispaly.

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
setInterval(() => {
    counter++;
}, 1000);
```

Add one to the `counter` (the four-digit digital tube displays the number plus one) every second that passes.

Phenomenon Picture

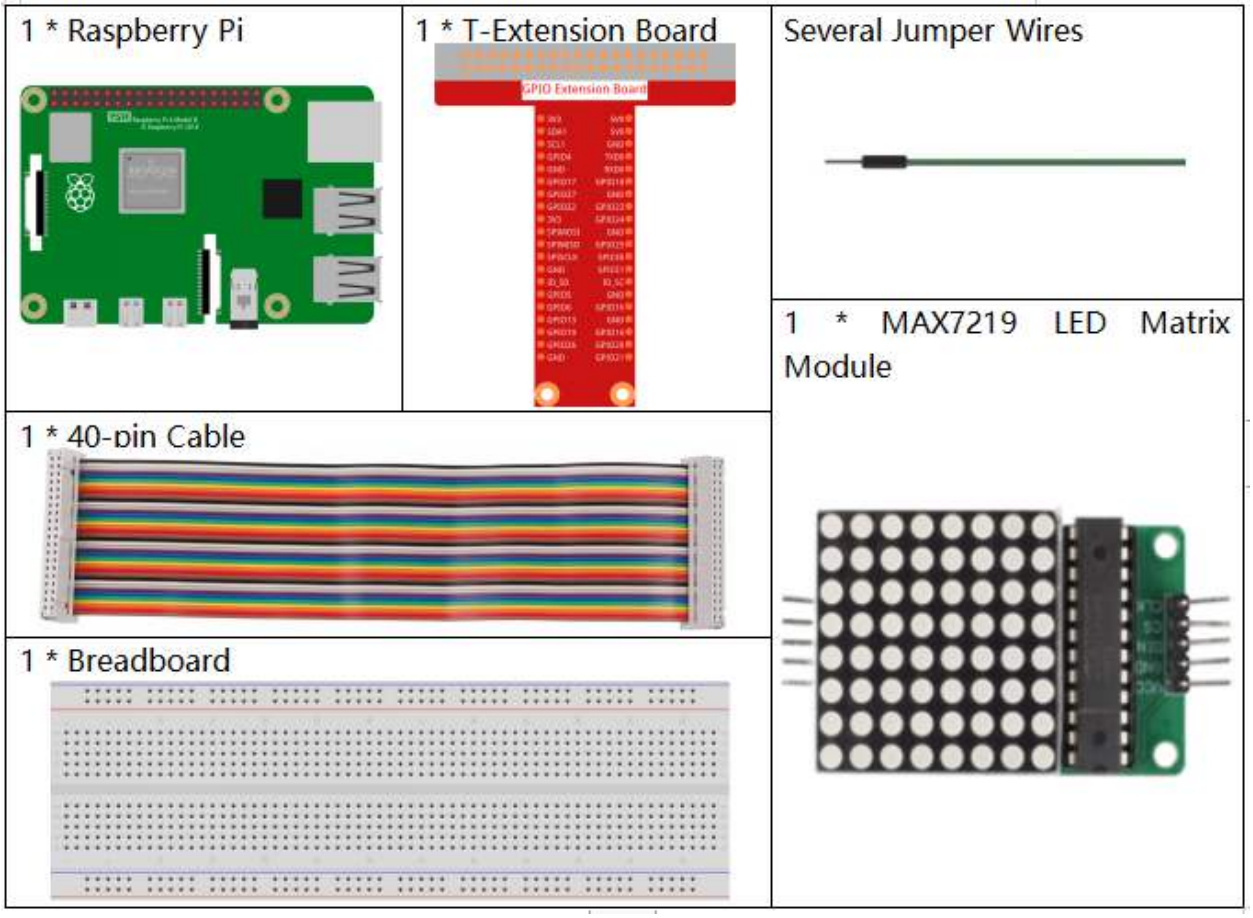


1.1.6 LED Dot Matrix Module

Introduction

In this project, you will learn about LED Matrix Module. LED Matrix Module uses the MAX7219 driver to drive the 8 x 8 LED Matrix.

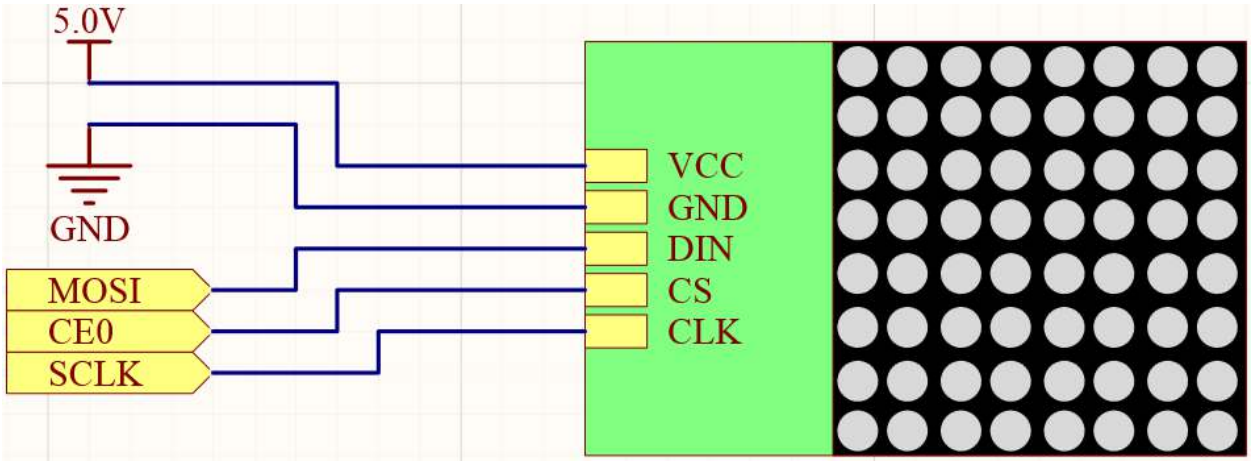
Components



- *GPIO Extension Board*
- *Breadboard*
- *LED Matrix Module*

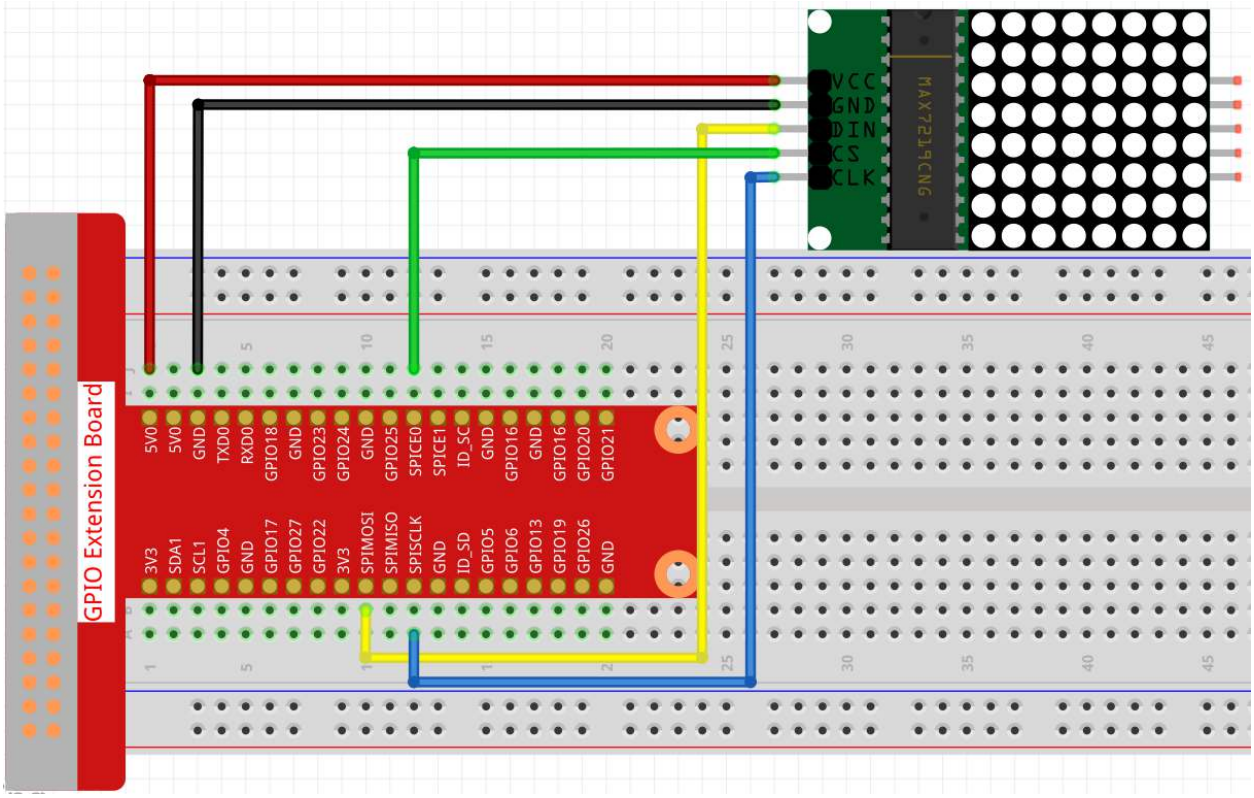
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
SPIMOSI	Pin 19	12	MOSI
SPICE0	pin 24	10	CE0
SPISCLK	Pin 23	14	SCLK



Experimental Procedures

Step 1: Build the circuit.



Note: Turn on the SPI before starting the experiment, refer to *SPI Configuration* for details.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Install dependencies.

```
sudo npm install spi-device
```

Step 4: Run the code.

```
sudo node max7219_led_matrix.js
```

After running the code, the LED Dot Matrix displays from 0 to 9 and A to Z in sequence.

Code

```
const Gpio = require('pigpio').Gpio;
const spi = require('spi-device');

class MAX7219_LED_MATRIX {
    constructor(bus, device) {
        this.bus = bus;
        this.device = device;
    }
    delay(ms) {
        return new Promise((resolve, reject) => {setTimeout(resolve, ms)});
    }

    async write(addr, data) {
        return new Promise((resolve, reject)=>{
            const max7219 = spi.open(this.bus, this.device, (err)=>{
                if (err) reject(err);

                const message = [{
                    sendBuffer: Buffer.from([addr, data]),
                    receiveBuffer: Buffer.alloc(2),
                    byteLength: 2,
                    speedHz: 20000
                }];

                max7219.transfer(message, (err, message)=>{
                    if (err) reject(err);
                    else resolve();
                })
            })
        })
    }
    async init() {
        await this.write(0x09, 0x00);
        await this.write(0x0a, 0x03);
        await this.write(0x0b, 0x07);
        await this.write(0x0c, 0x01);
        await this.write(0x0f, 0x00);
    }
}

const DISP=[
    [0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C],//0
    [0x08,0x18,0x28,0x08,0x08,0x08,0x08,0x08],//1
    [0x7E,0x2,0x2,0x7E,0x40,0x40,0x40,0x7E],//2
    [0x3E,0x2,0x2,0x3E,0x2,0x2,0x3E,0x0],//3
    [0x8,0x18,0x28,0x48,0xFE,0x8,0x8,0x8],//4
    [0x3C,0x20,0x20,0x3C,0x4,0x4,0x3C,0x0],//5
    [0x3C,0x20,0x20,0x3C,0x24,0x24,0x3C,0x0],//6
```

(continues on next page)

(continued from previous page)

```
    [0x3E,0x22,0x4,0x8,0x8,0x8,0x8,0x8],//7
    [0x0,0x3E,0x22,0x22,0x3E,0x22,0x22,0x3E],//8
    [0x3E,0x22,0x22,0x3E,0x2,0x2,0x2,0x3E],//9
    [0x8,0x14,0x22,0x3E,0x22,0x22,0x22,0x22],//A
    [0x3C,0x22,0x22,0x3E,0x22,0x22,0x3C,0x0],//B
    [0x3C,0x40,0x40,0x40,0x40,0x40,0x3C,0x0],//C
    [0x7C,0x42,0x42,0x42,0x42,0x42,0x7C,0x0],//D
    [0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x7C],//E
    [0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x40],//F
    [0x3C,0x40,0x40,0x40,0x40,0x44,0x44,0x3C],//G
    [0x44,0x44,0x44,0x7C,0x44,0x44,0x44,0x44],//H
    [0x7C,0x10,0x10,0x10,0x10,0x10,0x10,0x7C],//I
    [0x3C,0x8,0x8,0x8,0x8,0x8,0x48,0x30],//J
    [0x0,0x24,0x28,0x30,0x20,0x30,0x28,0x24],//K
    [0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x7C],//L
    [0x81,0xC3,0xA5,0x99,0x81,0x81,0x81,0x81],//M
    [0x0,0x42,0x62,0x52,0x4A,0x46,0x42,0x0],//N
    [0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C],//O
    [0x3C,0x22,0x22,0x22,0x3C,0x20,0x20,0x20],//P
    [0x1C,0x22,0x22,0x22,0x22,0x26,0x22,0x1D],//Q
    [0x3C,0x22,0x22,0x22,0x3C,0x24,0x22,0x21],//R
    [0x0,0x1E,0x20,0x20,0x3E,0x2,0x2,0x3C],//S
    [0x0,0x3E,0x8,0x8,0x8,0x8,0x8,0x8],//T
    [0x42,0x42,0x42,0x42,0x42,0x42,0x22,0x1C],//U
    [0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18],//V
    [0x0,0x49,0x49,0x49,0x49,0x2A,0x1C,0x0],//W
    [0x0,0x41,0x22,0x14,0x8,0x14,0x22,0x41],//X
    [0x41,0x22,0x14,0x8,0x8,0x8,0x8,0x8],//Y
    [0x0,0x7F,0x2,0x4,0x8,0x10,0x20,0x7F],//Z
];

lm = new MAX7219_LED_MATRIX(0, 0);

async function main(){
    lm.init();
    while(1){
        for(let j=0;j<36;j++){
            for(let i=1;i<9;i++){
                lm.write(i, DISP[j][i-1]);
            }
            await lm.delay(1000);
        }
    }
}
main();
```

Code Explanation

```
const spi = require('spi-device');
```

Import the modules needed for spi communication.

Note: When you have multiple devices that need spi communication, just connect the cs pins to different pins.

```
class MAX7219_LED_MATRIX {
    constructor(bus, device) {
        this.bus = bus;
        this.device = device;
    }
    delay(ms) {
        return new Promise((resolve, reject) => {setTimeout(resolve, ms)});
    }

    async write(addr, data) {
        return new Promise((resolve, reject)=>{
            const max7219 = spi.open(this.bus, this.device, (err)=>{
                if (err) reject(err);

                const message = [{
                    sendBuffer: Buffer.from([addr, data]),
                    receiveBuffer: Buffer.alloc(2),
                    byteLength: 2,
                    speedHz: 20000
                }];

                max7219.transfer(message, (err, message)=>{
                    if (err) reject(err);
                    else resolve();
                })
            })
        })
    }
```

Implement a `MAX7219_LED_MATRIX` class, and the `write()` function encapsulated in it can light up the matrix.

Note: The `async` keyword is used to modify the function and is usually matched with the `await` keyword. The statement modified by the `await` keyword needs to wait for the previous code to finish running before executing, achieving the effect of synchronous blocking.

- Async Function

```
lm = new MAX7219_LED_MATRIX(0, 0);
```

Instantiate an object lm of the `MAX7219_LED_MATRIX` class, so that we can call the encapsulated `write()` function inside.

```
while(1){
    for(let j=0;j<36;j++){
        for(let i=0;i<8;i++){
            lm.write(i, DISP[j][i]);
        }
        await lm.delay(1000);
    }
}
```

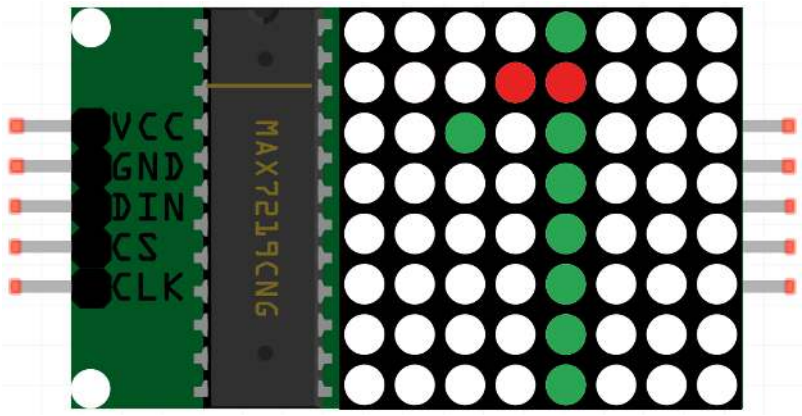
The `write(row,date)` function allows you to display specified characters on the LED dot matrix, The first parameter selects the row of the LED Matrix (8 rows in total), The second parameter enters an 8-bit binary number to control the 8 LEDs of the row (0 means off, 1 means on).

The variable `j` is used to select the glyph, which is `DISP[]` . There are a total of 35 glyphs, 0~9 and A~Z.

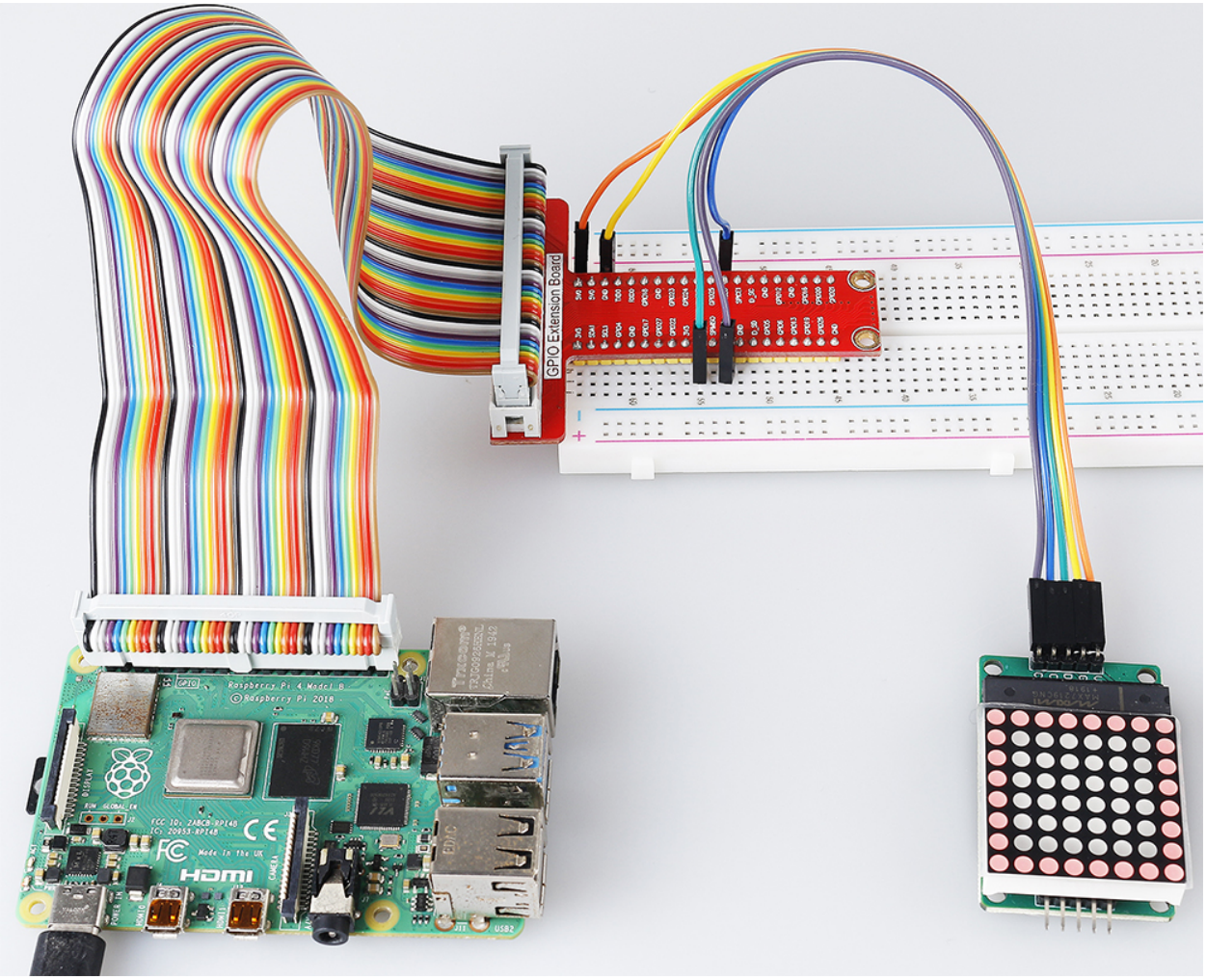
For example, when j=1, the LED Maxtrix should display the image **1** .

The variable `i` is used to write the 8 data in the `DISP[]` glyph into the LED Matrix in turn. After the loop is completed, an 8x8 graphic can be generated.

For example, when j=1, i=1, the data of `DISP[1][1]` will be written here, that is, `0x18`, This will cause the second row of the LED Maxtrix to display the image `00011000`.



Phenomenon Picture

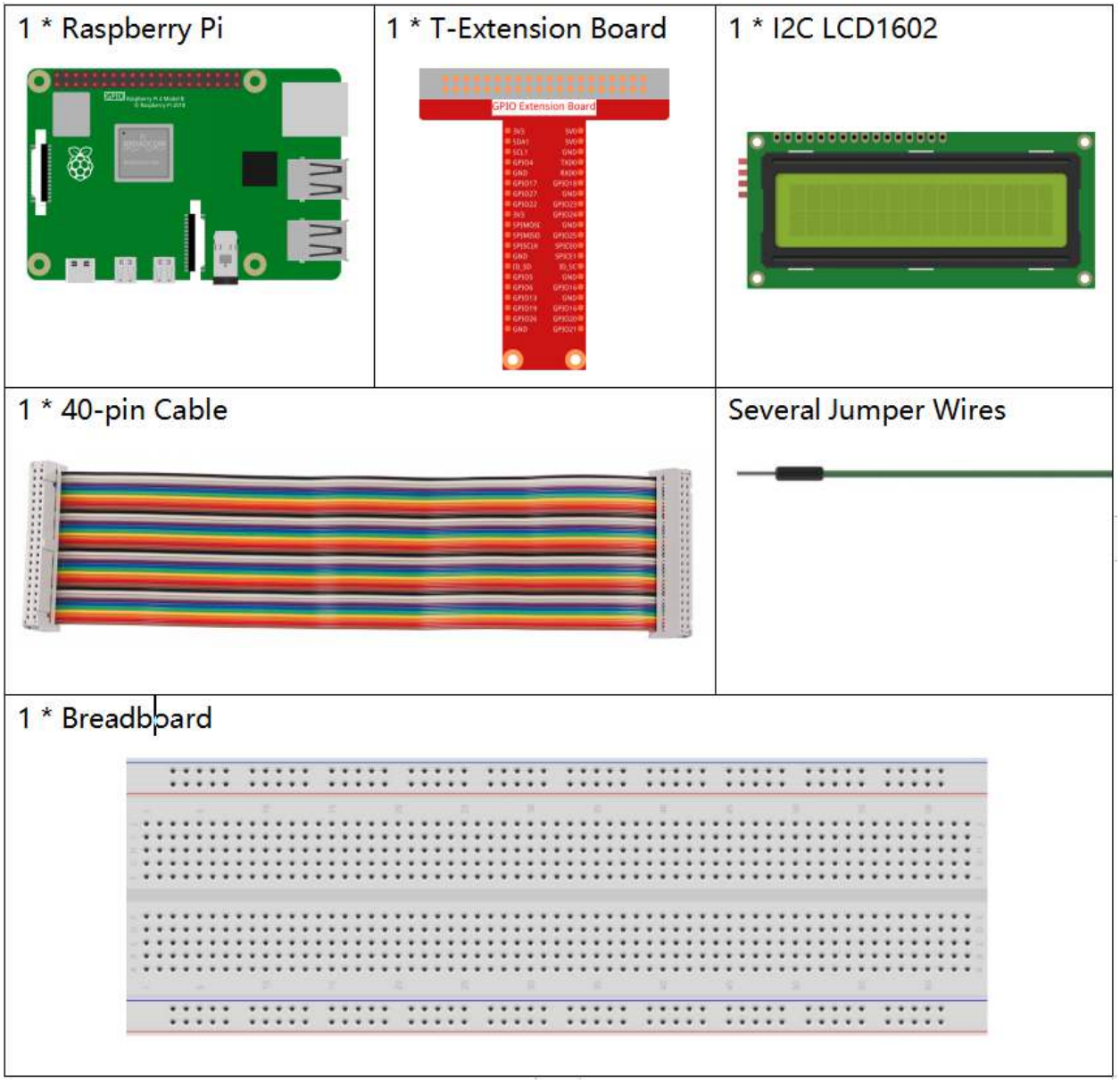


1.1.7 I2C LCD1602

Introduction

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

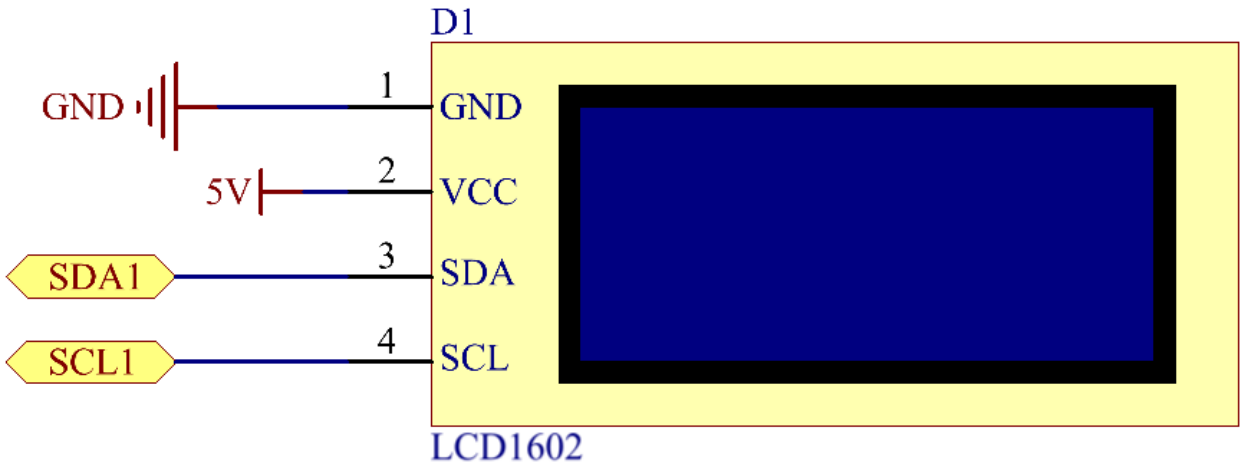
Components



- *GPIO Extension Board*
- *Breadboard*
- *I2C LCD1602*

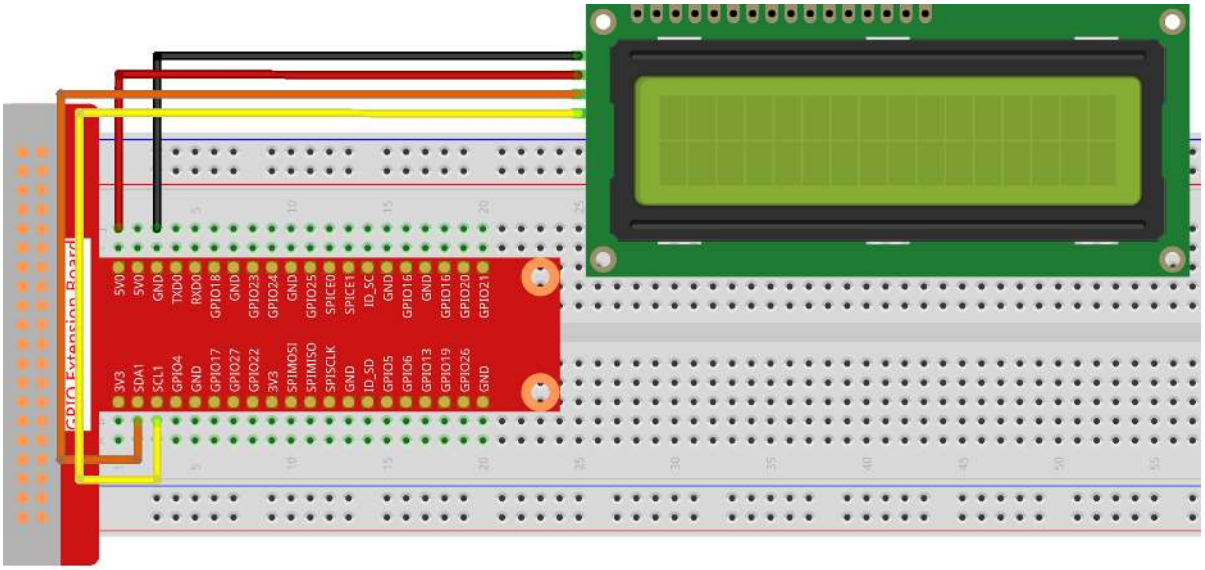
Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see *I2C Configuration*. If you have set I2C, skip this step.)

Step 3: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 4: Install dependencies.

```
sudo npm install @oawu/lcd1602
```

Step 5: Run the code.

```
sudo node i2c_lcd1602.js
```

After the code runs, you can see `Greetings!!, From SunFounder` displaying on the LCD.

Code

```
const LCD = require('@oawu/lcd1602');
const lcd = new LCD();

lcd.text(0, 0, 'Greetings!!');
lcd.text(1, 1, 'from SunFounder');
```

Code Explanation

```
const LCD = require('@oawu/lcd1602');
const lcd = new LCD();
```

Import the `lcd1602` module and represent it with `lcd`.

Note: For the lcd1602 module, please refer to: https://www.npmjs.com/package/@oawu/lcd1602

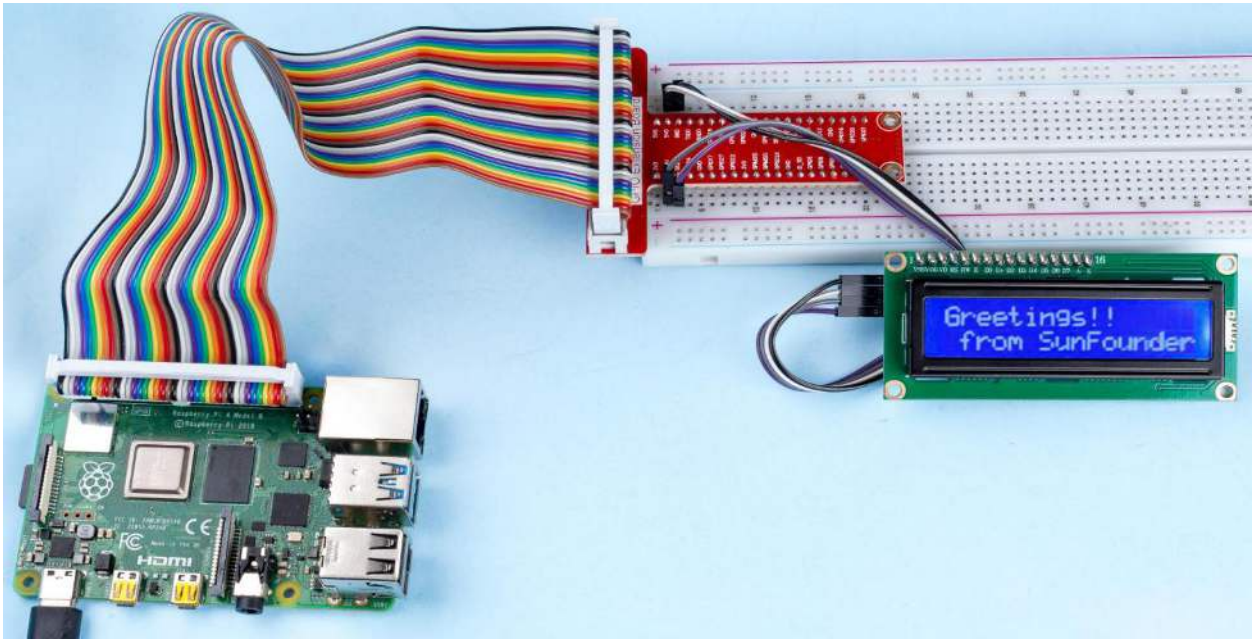
```
lcd.text(0, 0, 'Greetings!!');
lcd.text(1, 1, 'from SunFounder');
```

Calling the encapsulated `text()` function in the `LCD` class can make the lcd1602 display the text we want.

The `text()` function receives three parameters, the first parameter is the line of lcd1602, the second parameter represents the position of the displayed text, and the third parameter represents the text we want to display.

The **1602** number in the LCD model means it has 2 rows of 16 cells each.

Phenomenon Picture



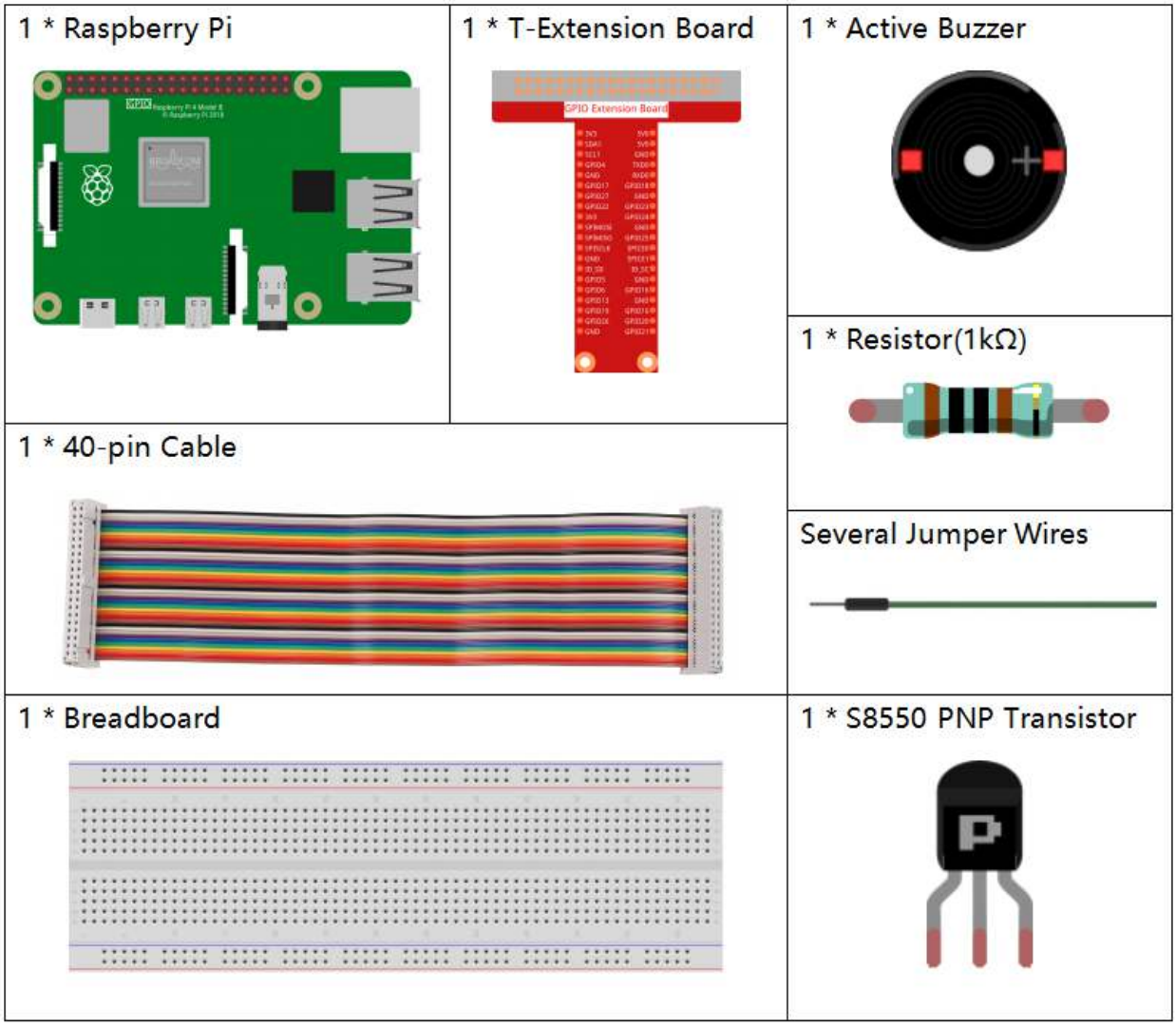
9.4.2 1.2 Sound

1.2.1 Active Buzzer

Introduction

In this project, we will learn how to drive an active buzzer to beep with a PNP transistor.

Components

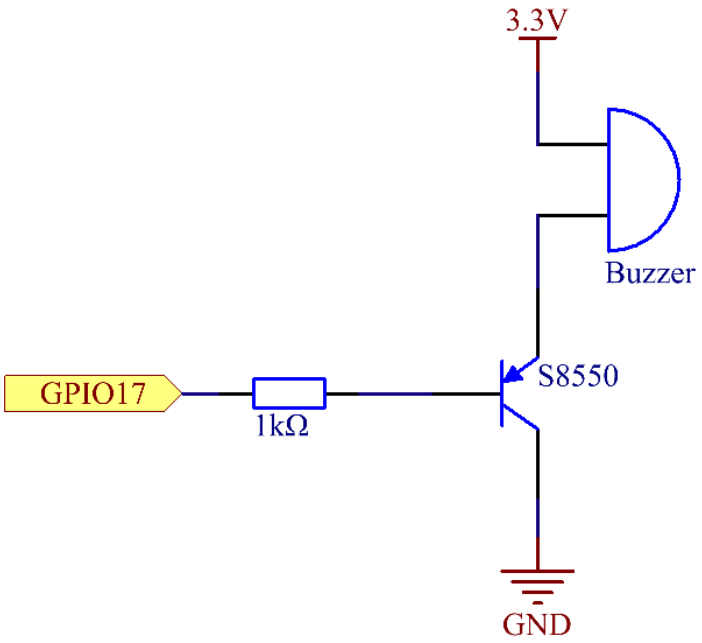


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Schematic Diagram

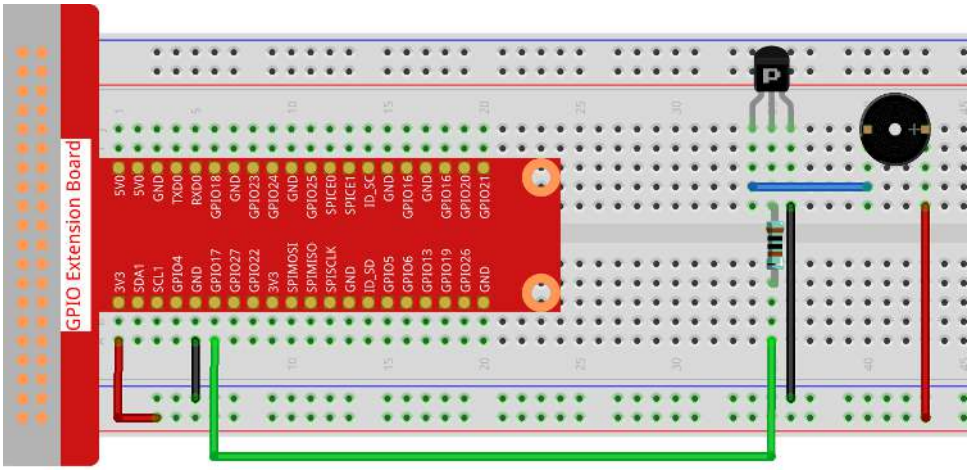
In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and the buzzer will make sounds. But when high level is supplied to the IO of Raspberry Pi, the transistor will be cut off and the buzzer will not make sounds.

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



Experimental Procedures

Step 1: Build the circuit. (The active buzzer has a white table sticker on the surface and a black back.)



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run.

```
sudo node active_buzzer.js
```

The code run, the buzzer beeps.

Code

```
const Gpio = require('pigpio').Gpio;
const active = new Gpio(17,{mode: Gpio.OUTPUT});

setInterval(() => {
  active.digitalWrite(!active.digitalRead());
}, 500);

process.on('SIGINT',function(){
  active.digitalWrite(1);
  process.exit();
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;
const active = new Gpio(17,{mode: Gpio.OUTPUT});
```

Import the `pigpio` module, and instantiate an object active to control the IO port Gpio17, and the mode is set to output mode.

```
setInterval(() => {
  active.digitalWrite(!active.digitalRead());
}, 500);
```

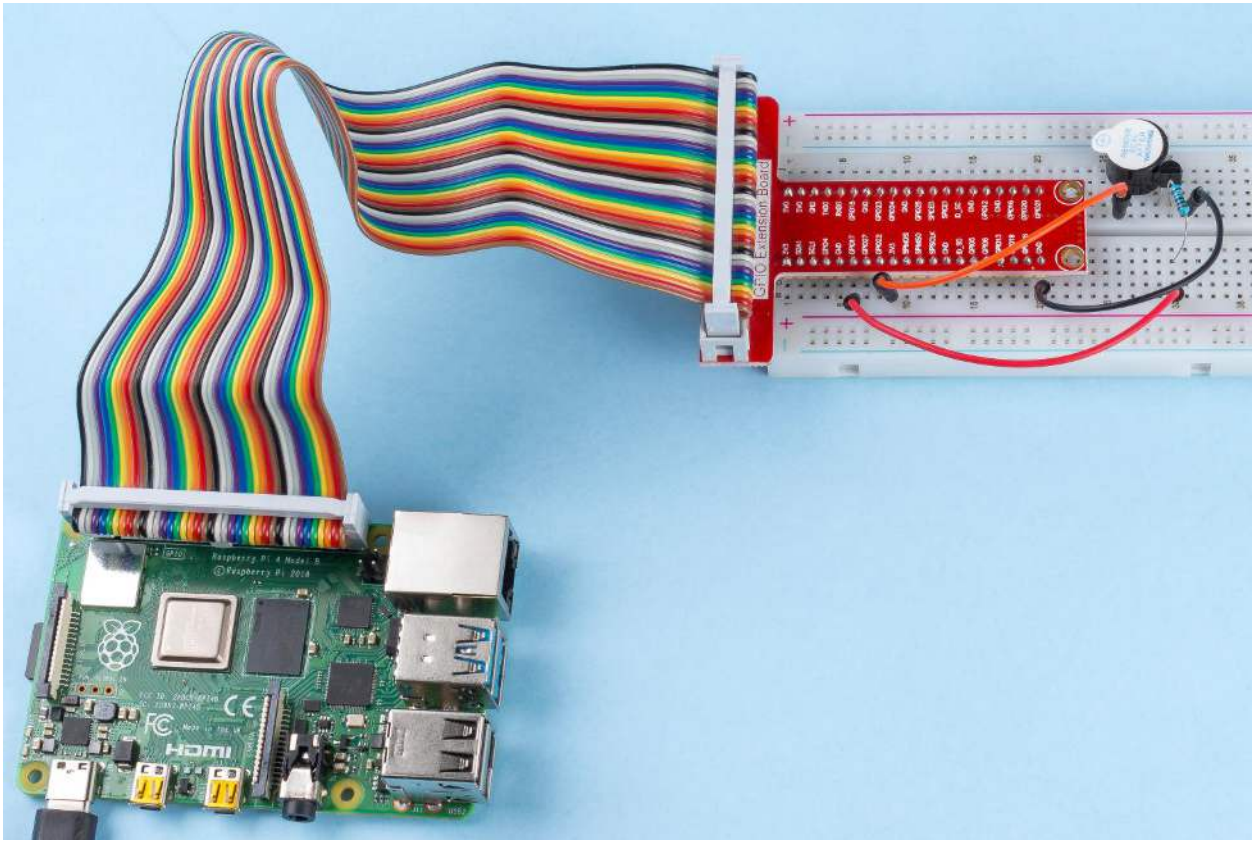
The active buzzer is similar to the LED in usage and can be controlled with `digitalWrite()`, and `digitalRead()` is used to read the current pin level. Here we make the active buzzer change its working state every 500ms.

```
process.on('SIGINT', function() {
    /* DO SOME STUFF HERE */

    process.exit()
})
```

Handle Ctrl+C, here is used to stop the buzzer sounding when exiting the program.

Process - NodeJS

Phenomenon Picture



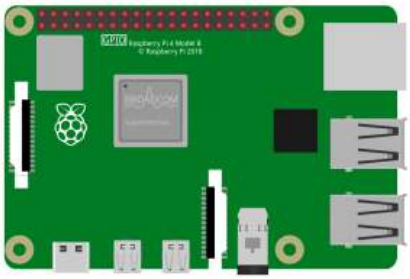
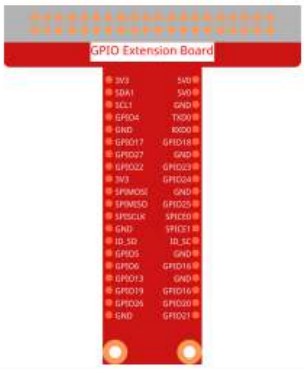
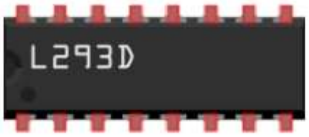
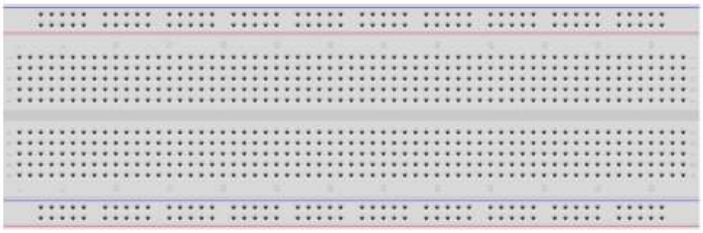
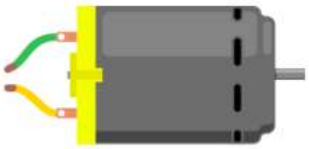
9.4.3 1.3 Drivers

1.3.1 Motor

Introduction

In this project, we will learn to how to use L293D to drive a DC motor and make it rotate clockwise and counterclockwise. Since the DC Motor needs a larger current, for safety purpose, here we use the Power Supply Module to supply motors.

Components

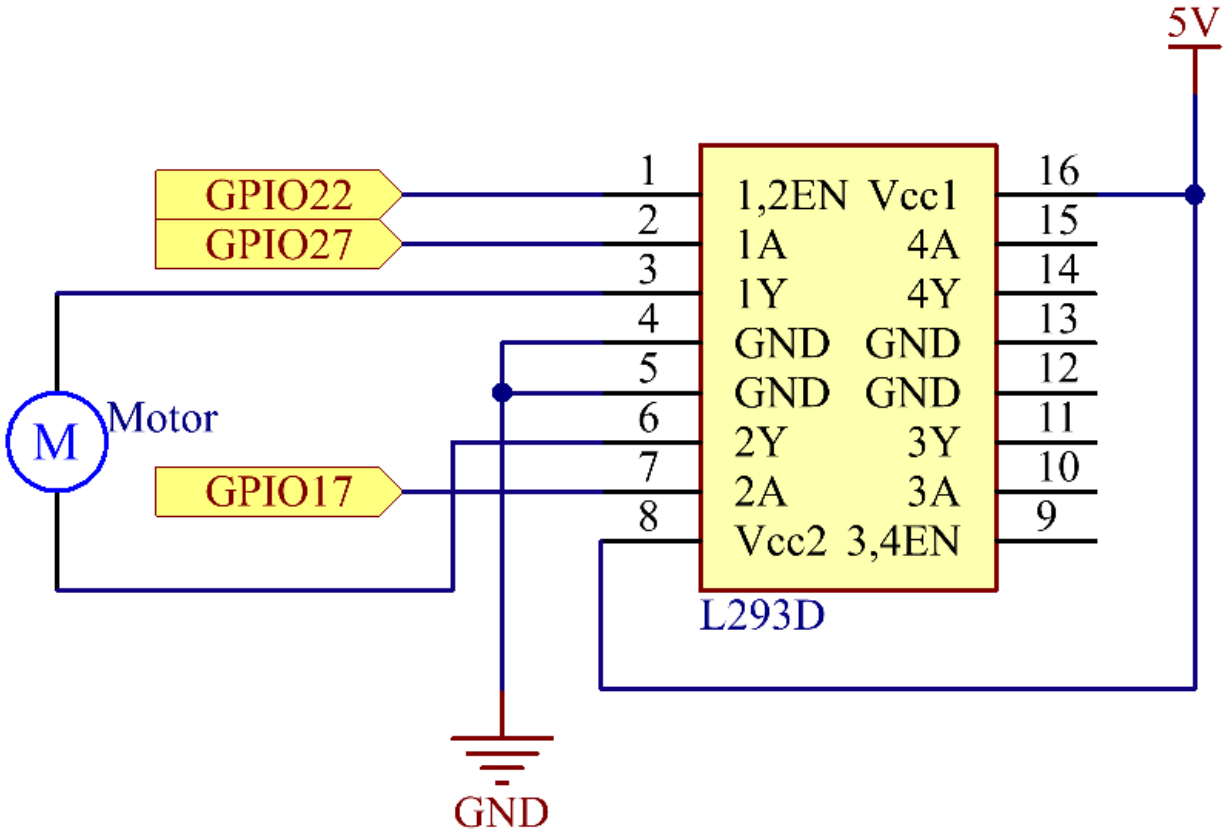
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Power Module (with 9V battery and buckle)
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * L293D
  </td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * DC Motor
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Power Supply Module*
- *L293D*
- *DC Motor*

Schematic Diagram

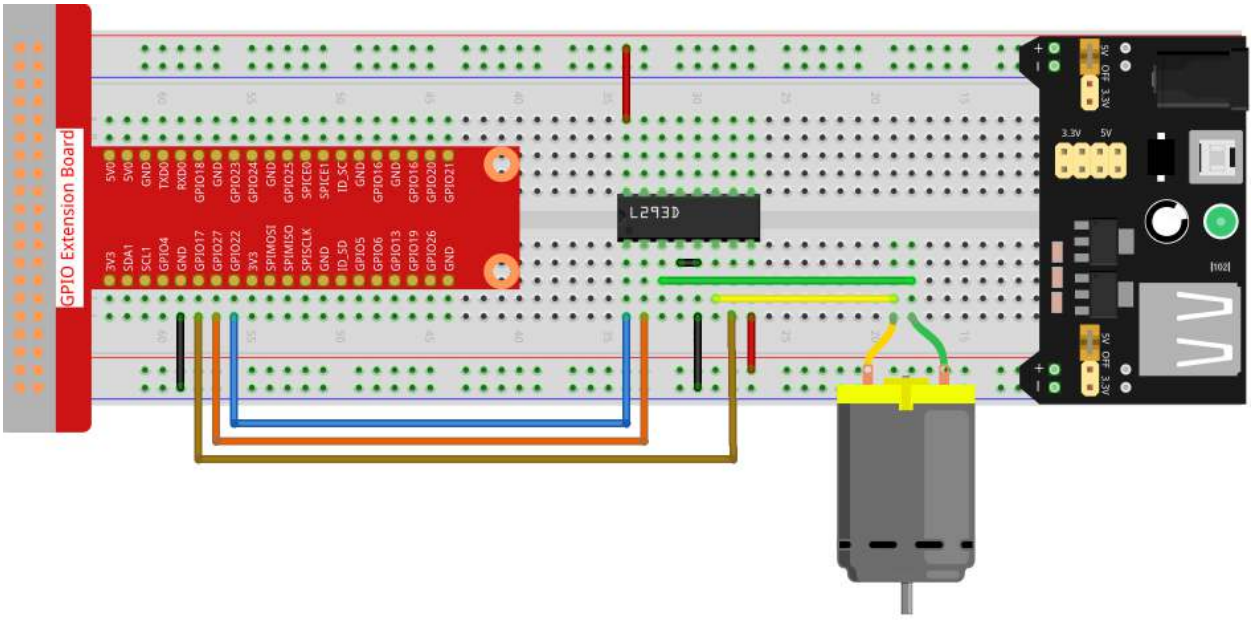
Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to GPIO22, and set it as high level. Connect pin2 to GPIO27, and pin7 to GPIO17, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

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



Experimental Procedures

Step 1: Build the circuit.



Note: 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.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 4: Run the code.

```
sudo node motor.js
```

As the code runs, the motor first rotates clockwise for 1s then stops for 1s, after that, it rotates anticlockwise for 1s; subsequently, the motor stops for 1s. This series of actions will be executed repeatedly.

Code

```
const Gpio = require('pigpio').Gpio;

MotorPin1 = new Gpio(17, { mode: Gpio.OUTPUT });
MotorPin2 = new Gpio(27, { mode: Gpio.OUTPUT });
```

(continues on next page)

(continued from previous page)

```
MotorEnable = new Gpio(22, { mode: Gpio.OUTPUT });


// Define a motor function to spin the motor
// direction should be
// 2(clockwise), 1(counterclockwise), 0(stop)
function motor(direction) {
    switch (direction) {
        case 2: // Clockwise
            // Set direction
            MotorPin1.digitalWrite(1)
            MotorPin2.digitalWrite(0)
            // Enable the motor
            MotorEnable.digitalWrite(1)
            console.log('Clockwise')
            break;
        case 1:  // Counterclockwise
            // Set direction
            MotorPin1.digitalWrite(0)
            MotorPin2.digitalWrite(1)
            // Enable the motor
            MotorEnable.digitalWrite(1)
            console.log('Counterclockwise')
            break;
        case 0: // Stop
            // Disable the motor
            MotorEnable.digitalWrite(0)
            console.log('Stop')

    }
}

process.on('SIGINT', function () {
    MotorEnable.digitalWrite(0)
    process.exit();
})

let index=-1
setInterval(() => {
    index=(index+1)%3
    motor(index)
}, 1000)
```

Code Explanation

```
MotorPin1 = new Gpio(17, { mode: Gpio.OUTPUT });
MotorPin2 = new Gpio(27, { mode: Gpio.OUTPUT });
MotorEnable = new Gpio(22, { mode: Gpio.OUTPUT });
```

Import the pigpio module and create three Gpio class objects to control the three IO ports of Gpio17, Gpio27, and Gpio22.

```
function motor(direction) {
    switch (direction) {
        case 2: // Clockwise
            // Set direction
```

(continues on next page)

(continued from previous page)

```
            MotorPin1.digitalWrite(1)
            MotorPin2.digitalWrite(0)
            // Enable the motor
            MotorEnable.digitalWrite(1)
            console.log('Clockwise')
            break;
        case 1:  // Counterclockwise
            // Set direction
            MotorPin1.digitalWrite(0)
            MotorPin2.digitalWrite(1)
            // Enable the motor
            MotorEnable.digitalWrite(1)
            console.log('Counterclockwise')
            break;
        case 0: // Stop
            // Disable the motor
            MotorEnable.digitalWrite(0)
            console.log('Stop')

    }
}
```

Define a motor() function to control the motor,

1. When the direction is equal to 2, the MotorPin1 port writes a high level, the MotorPin2 port writes a low level, and the enable port MotorEnable writes a high level, and the motor rotates clockwise.
2. When the direction is equal to 1, the MotorPin1 port writes a low level, the MotorPin2 port writes a high level, and the enable port MotorEnable writes a high level, and the motor rotates counterclockwise.
3. When the direction is equal to 0, the enable port MotorEnable is written to a low level, and the motor stops rotating.

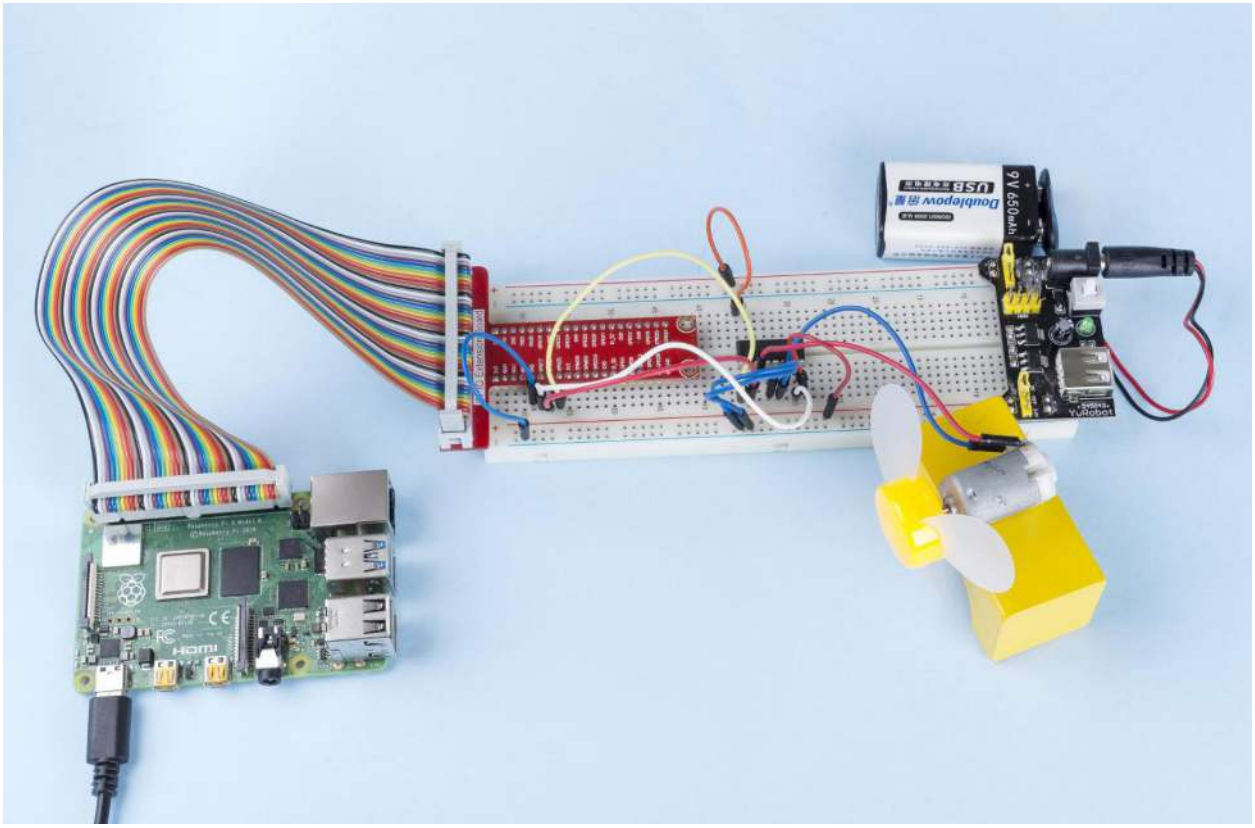
```
let index=-1
setInterval(() => {
    index=(index+1)%3
    motor(index)
}, 1000)
```

Let the motor rotate clockwise and counterclockwise alternately, with an interval of 1 second.

```
process.on('SIGINT', function () {
    MotorEnable.digitalWrite(0)
    process.exit();
})
```

When it is detected that **ctrl+c is** pressed, MotorEnable is written low to stop the motor from spinning.

Phenomenon Picture

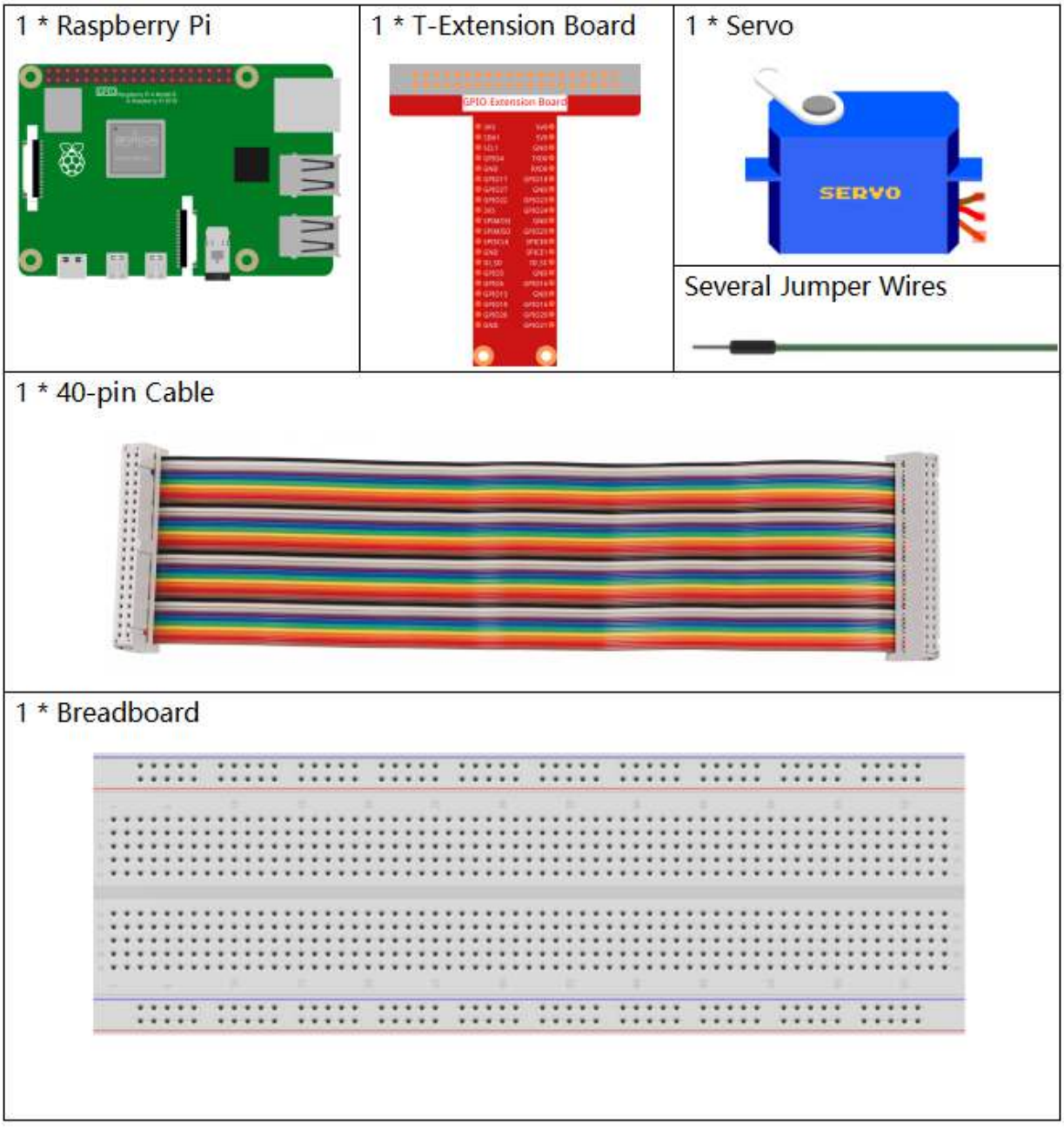


1.3.2 Servo

Introduction

In this project, we will learn how to make the servo rotate.

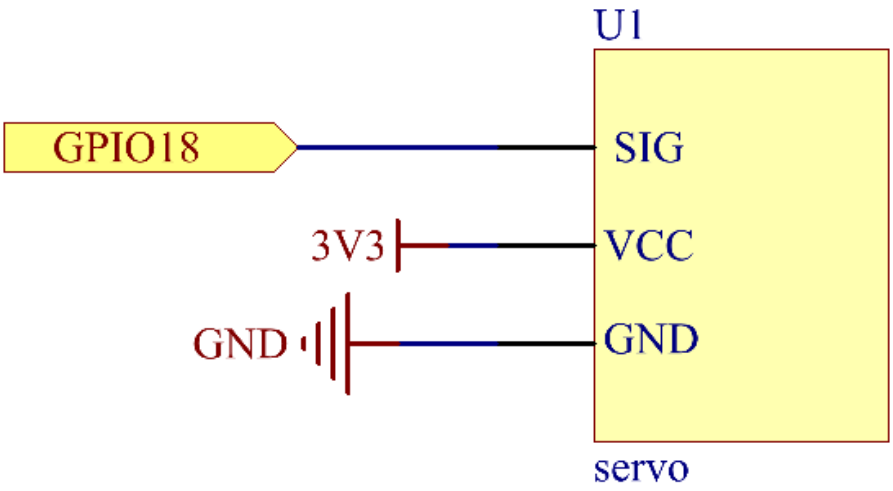
Components



- *GPIO Extension Board*
- *Breadboard*
- *Servo*

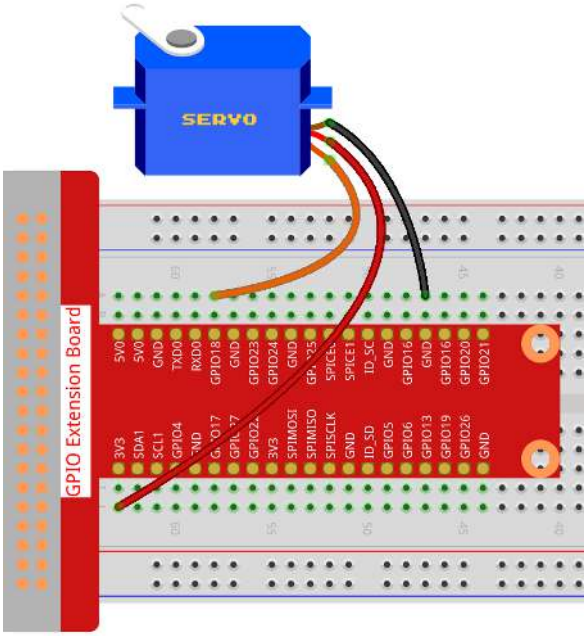
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node servo.js
```

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

Code

```
const Gpio = require('pigpio').Gpio;

SERVO_MIN_ANGLE = 0
SERVO_MAX_ANGLE = 180
SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

ServoPin = new Gpio(18,{mode: Gpio.OUTPUT})

function map(value, inMin, inMax, outMin, outMax){
    return (outMax - outMin) * (value - inMin) / (inMax - inMin) + outMin
}

function angle2pulse(angle){
    return Math.floor(map(angle,SERVO_MIN_ANGLE,SERVO_MAX_ANGLE,SERVO_MIN_PULSE ,
↪SERVO_MAX_PULSE))
}

let angle=90;
let step=5;
setInterval(() => {
    if(angle>=180||angle<=0){
        step=-step
    }
    angle+=step;
    ServoPin.servoWrite(angle2pulse(angle));
}, 20);
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

ServoPin = new Gpio(18,{mode: Gpio.OUTPUT})
```

Import the pigpio module and create an object of class Gpio, ServoPin, to control the output of Gpio18.

```
SERVO_MIN_ANGLE = 0
SERVO_MAX_ANGLE = 180
SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

function map(value, inMin, inMax, outMin, outMax){
    return (outMax - outMin) * (value - inMin) / (inMax - inMin) + outMin
}
```

(continues on next page)

(continued from previous page)

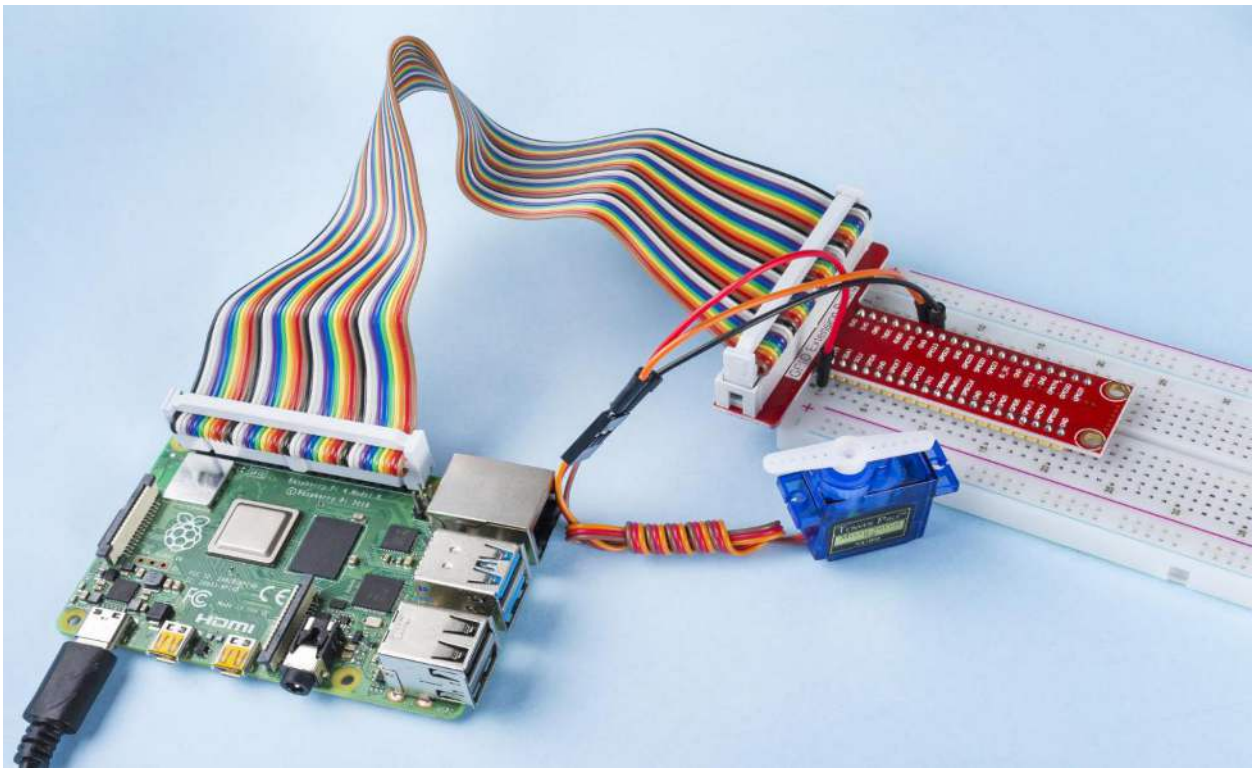
```
function angle2pulse(angle){
    return Math.floor(map(angle,SERVO_MIN_ANGLE,SERVO_MAX_ANGLE,SERVO_MIN_PULSE ,
↪SERVO_MAX_PULSE))
}
```

The function that maps the angle to the pulse width is defined here. This is because the servo control function `servoWrite(pulseWidth)` encapsulated in the Gpio class needs to write pulse width instead of angle. The angle range of the servo we use is 0~180, which needs to be mapped to the range of pulseWidth, 500~2500.

```
let angle=90;
let step=5;
setInterval(() => {
    if(angle>=180||angle<=0){
        step=-step
    }
    angle+=step;
    ServoPin.servoWrite(angle2pulse(angle));
}, 20);
```

Let the servo angle deflect back and forth from 0 to 180.

Phenomenon Picture

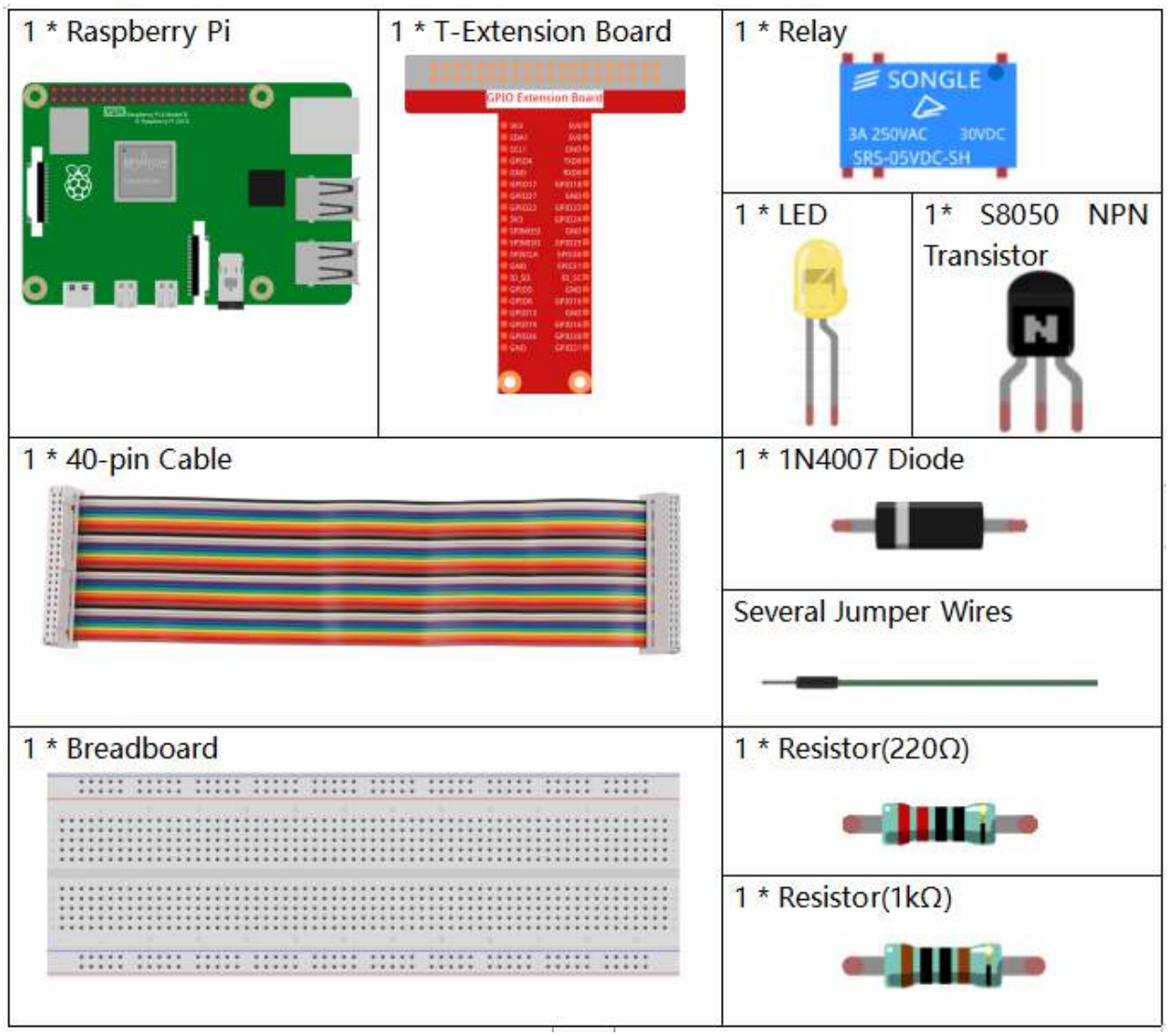


1.3.3 Relay

Introduction

In this project, we will learn to use a relay. It is one of the commonly used components in automatic control system. When the voltage, current, temperature, pressure, etc., reaches, exceeds or is lower than the predetermined value, the relay will connect or interrupt the circuit, to control and protect the equipment.

Components

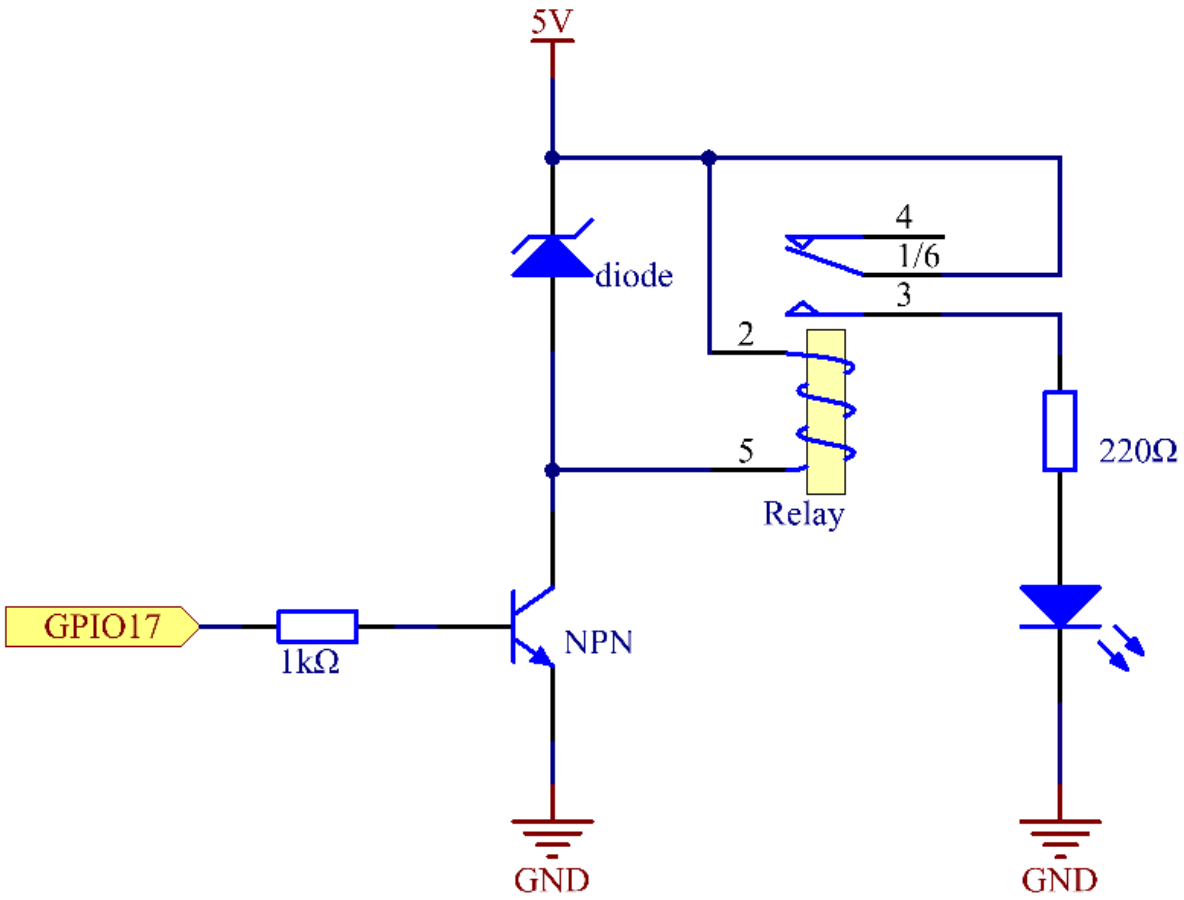


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Transistor*

- *Relay*
- *Diode*

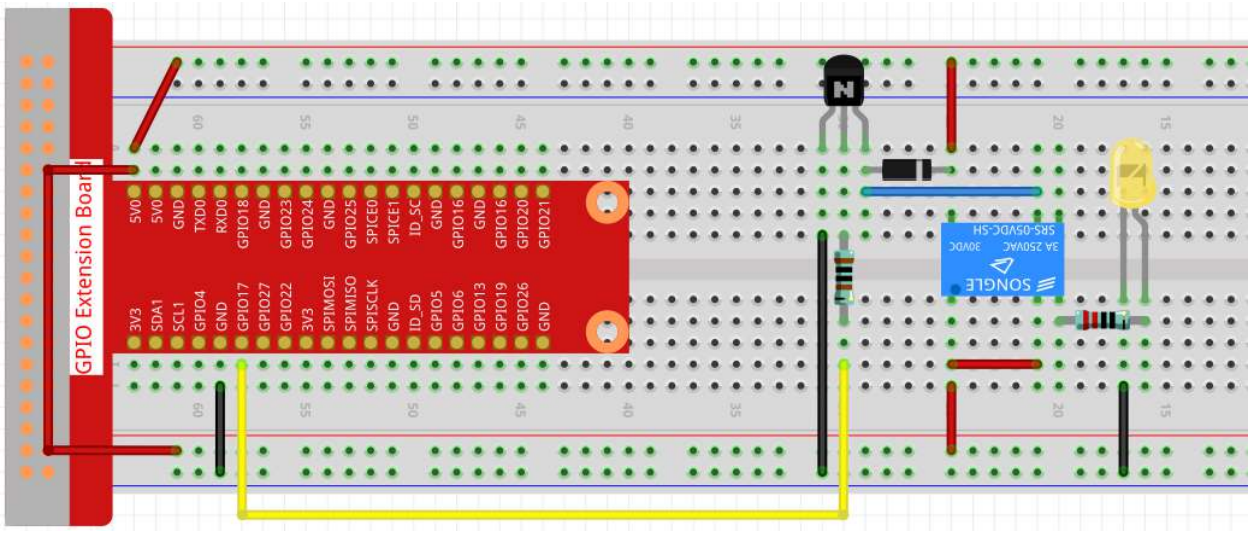
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node relay.js
```

While the code is running, the LED lights up. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.

Code

```
const Gpio = require('pigpio').Gpio;
const relay = new Gpio(17,{mode: Gpio.OUTPUT});

setInterval(() => {
    relay.digitalWrite(!relay.digitalRead());
}, 500);

process.on('SIGINT',function(){
    relay.digitalWrite(0);
    process.exit();
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;
const relay = new Gpio(17,{mode: Gpio.OUTPUT});
```

Import the `pigpio` module and instantiate an object relay of Gpio to control the IO port Gpio17, and set it to output mode.

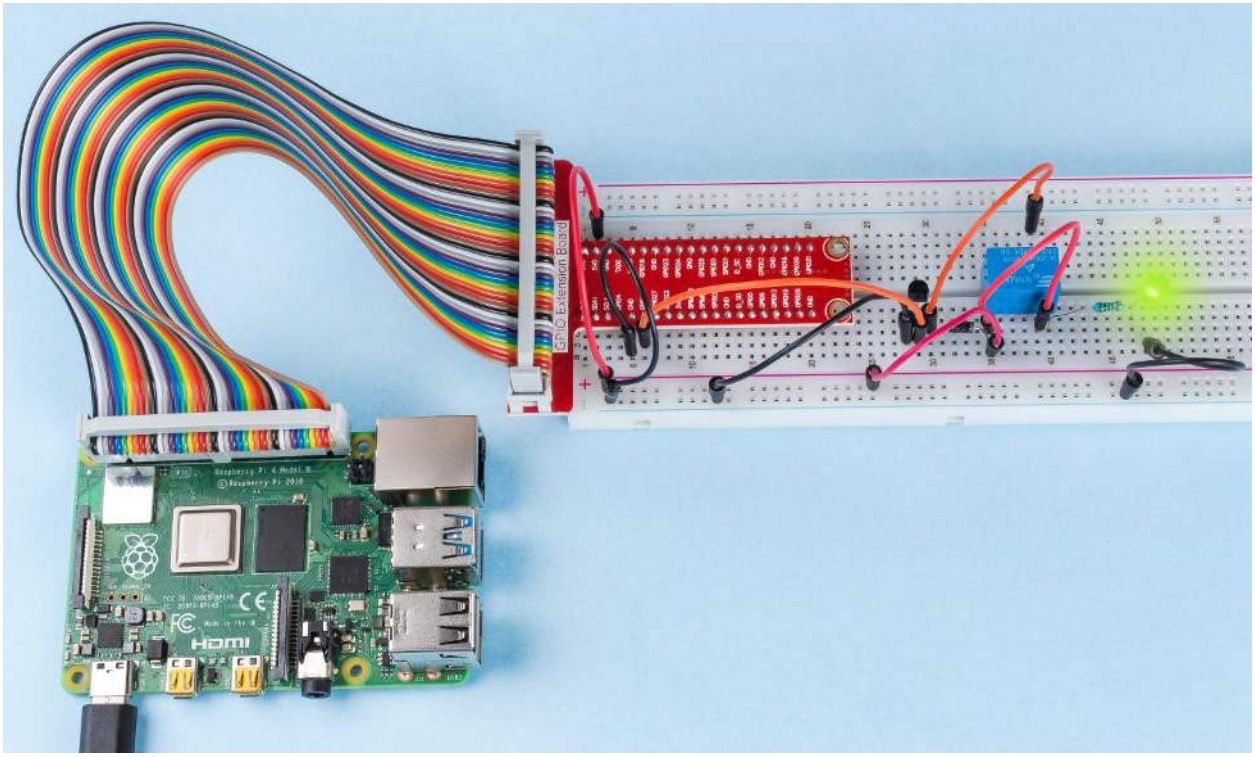
```
setInterval(() => {
    relay.digitalWrite(!relay.digitalRead());
}, 500);
```

The relay is opened and closed continuously, and the LEDs will also be on and off continuously at intervals of 500ms.

```
process.on('SIGINT',function(){
    relay.digitalWrite(0);
    process.exit();
});
```

When ctrl+c is caught, the relay is opened.

Phenomenon Picture



9.5 Input

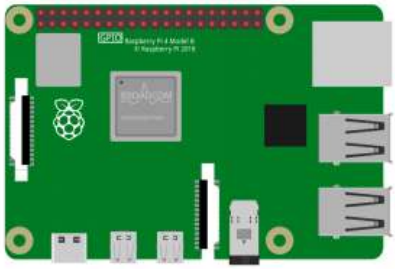
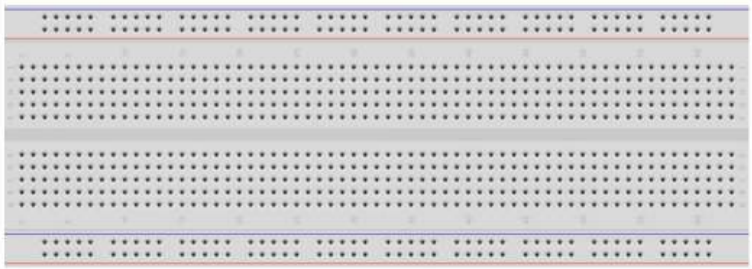
9.5.1 2.1 Controllers

2.1.1 Button

Introduction

In this project, we will learn how to turn on or off the LED by using a button.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>1 * LED
 </td></tr>
<tr><td></td><td></td><td>1 * Resistor 10KΩ
 </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td>1 * Resistor(220Ω)
 </td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * Button
 </td></tr>
</table>

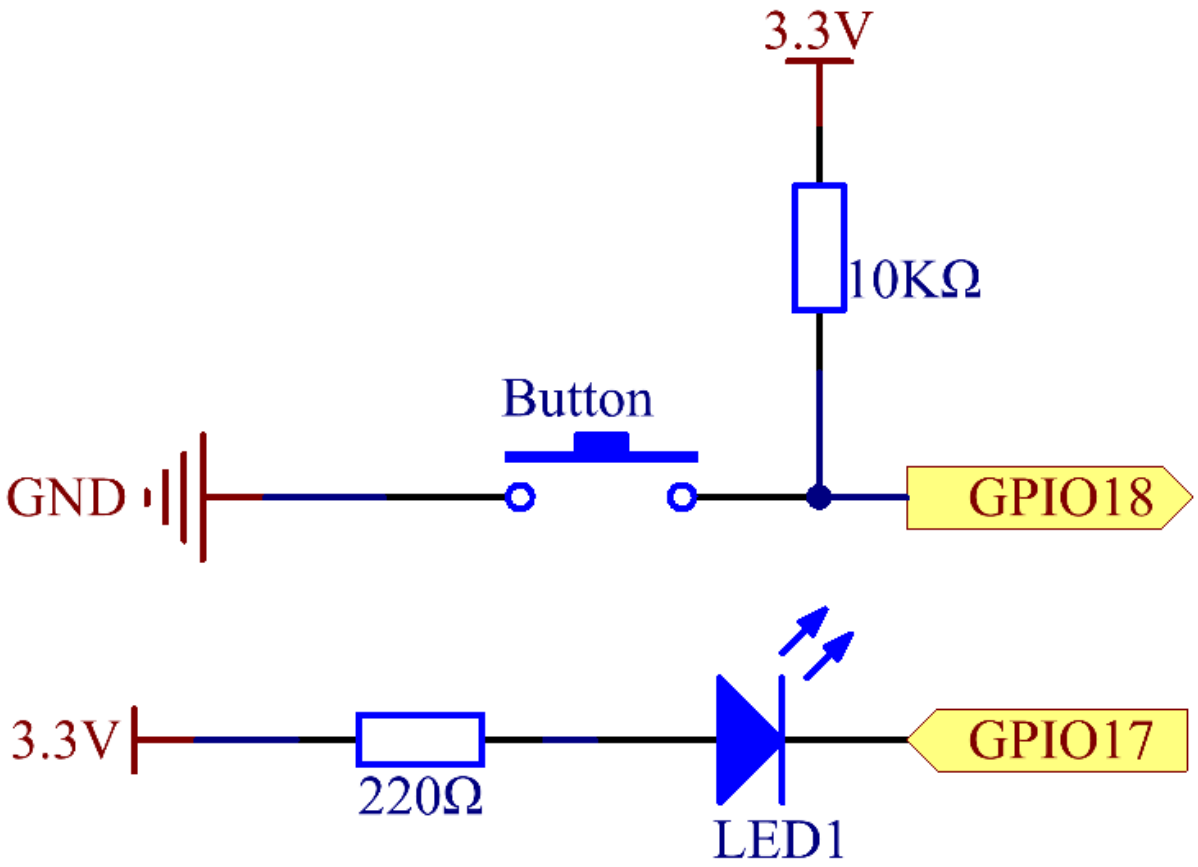
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Button*

Schematic Diagram

Use a normally open button as the input of Raspberry Pi, the connection is shown in the schematic diagram below. When the button is pressed, the GPIO18 will turn into low level (0V). We can detect the state of the GPIO18 through programming. That is, if the GPIO18 turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

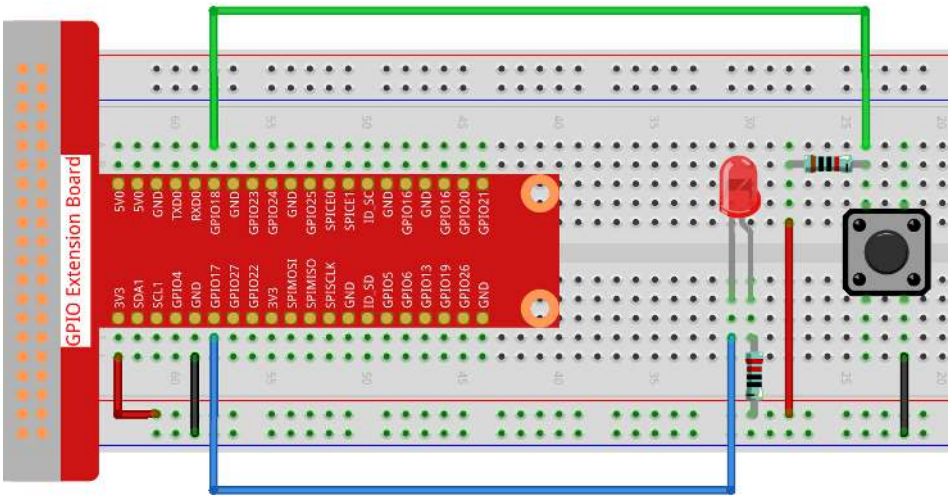
Note: The longer pin of the LED is the anode and the shorter one is the cathode.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node button.js
```

Now, press the button, and the LED will light up; release the button, and the LED will go out.

Code

```
const Gpio = require('pigpio').Gpio;


const led = new Gpio(17, {mode: Gpio.OUTPUT});

const button = new Gpio(18, {
  mode: Gpio.INPUT,
  pullUpDown: Gpio.PUD_DOWN,
  edge: Gpio.EITHER_EDGE
});


button.on('interrupt', (level) => {
  led.digitalWrite(level);
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

const led = new Gpio(17, {mode: Gpio.OUTPUT});
```

Import the `pigpio` module, create a led object to control the IO port Gpio17, and set it to output mode.

```
const button = new Gpio(18, {
  mode: Gpio.INPUT,
  pullUpDown: Gpio.PUD_DOWN,
  edge: Gpio.EITHER_EDGE
});
```

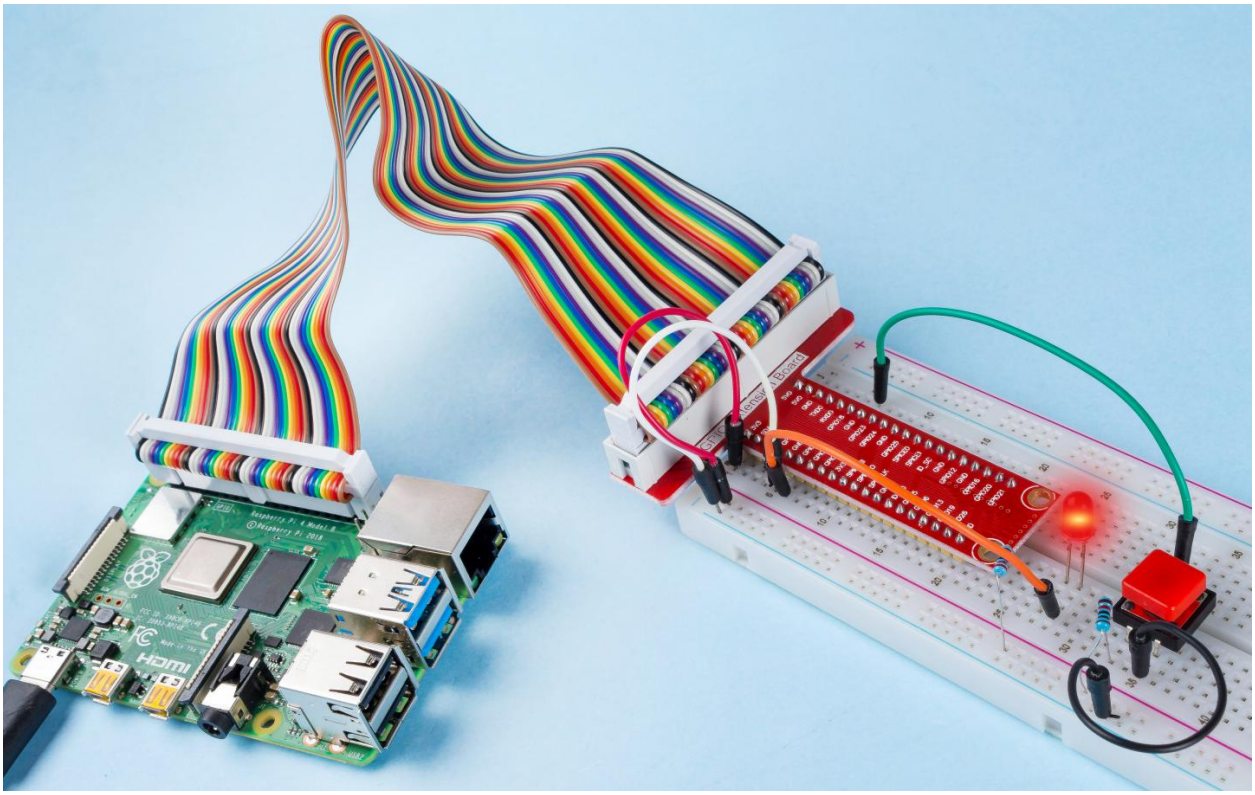
Create a button object to control the IO port Gpio18, set it to input mode, Pull down (low when the button is not pressed, high when the button is pressed). And set the interrupt function, the mode is EITHER_EDGE, that is, both rising and falling edges will trigger the interrupt function.

```
button.on('interrupt', (level) => {
  led.digitalWrite(level);
});
```

Write an interrupt function, when the button is pressed, it is a falling edge, triggering the interrupt function, At this time, write the low level of the button IO port to the IO port of the led, and the led lights up.

When the button is released, it is a rising edge, triggering the interrupt function, At this time, the high level of the button IO port is written to the IO port of the led, and the led is off.

Phenomenon Picture



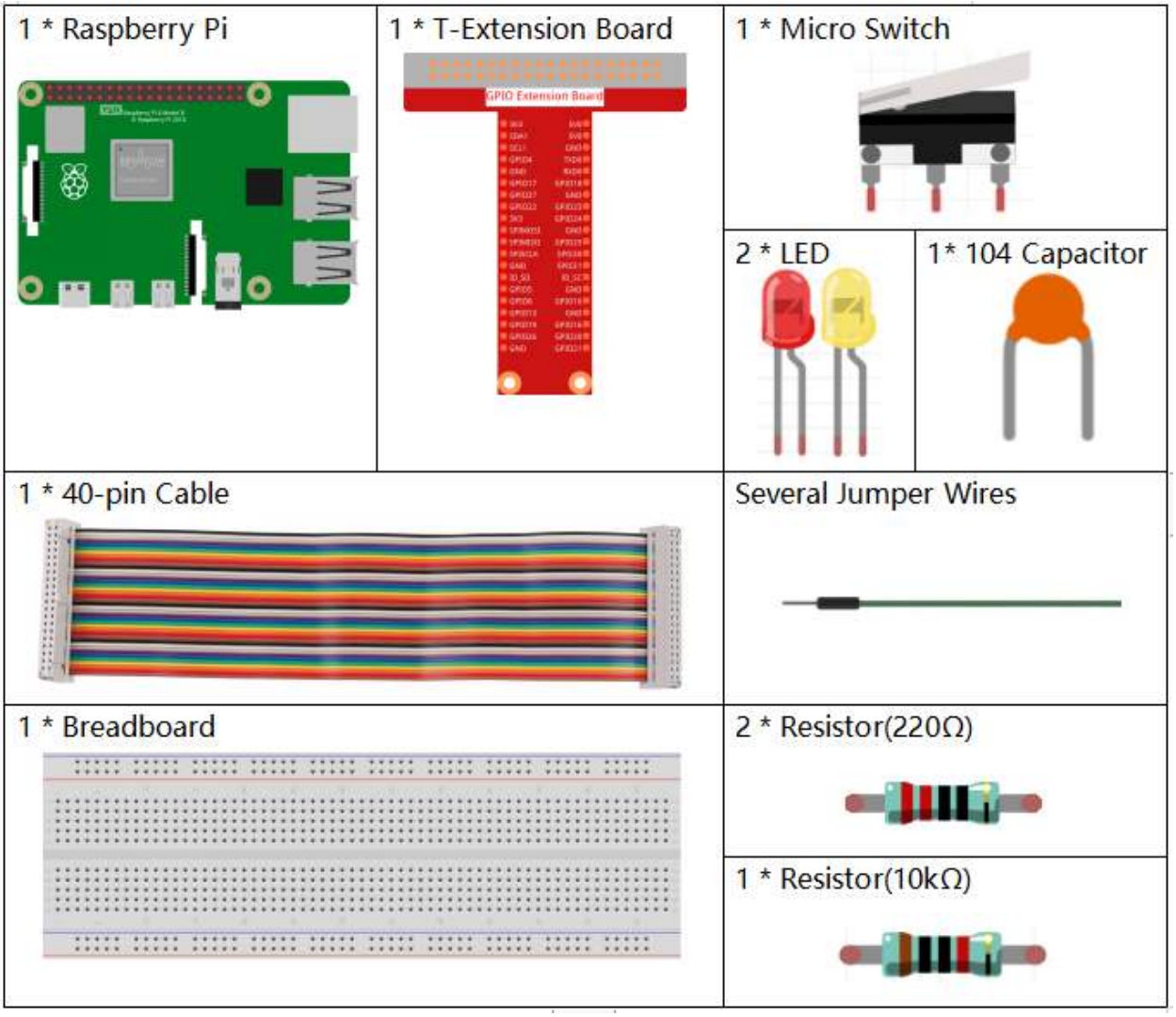
2.1.2 Micro Switch

Introduction

In this project, we will learn how to use Micro Switch. A Micro Switch is a small, very sensitive switch which requires minimum compression to activate. Because they are reliable and sensitive, micro switches are often used as a safety device.

They are used to prevent doors from closing if something or someone is in the way and other applications similar.

Components



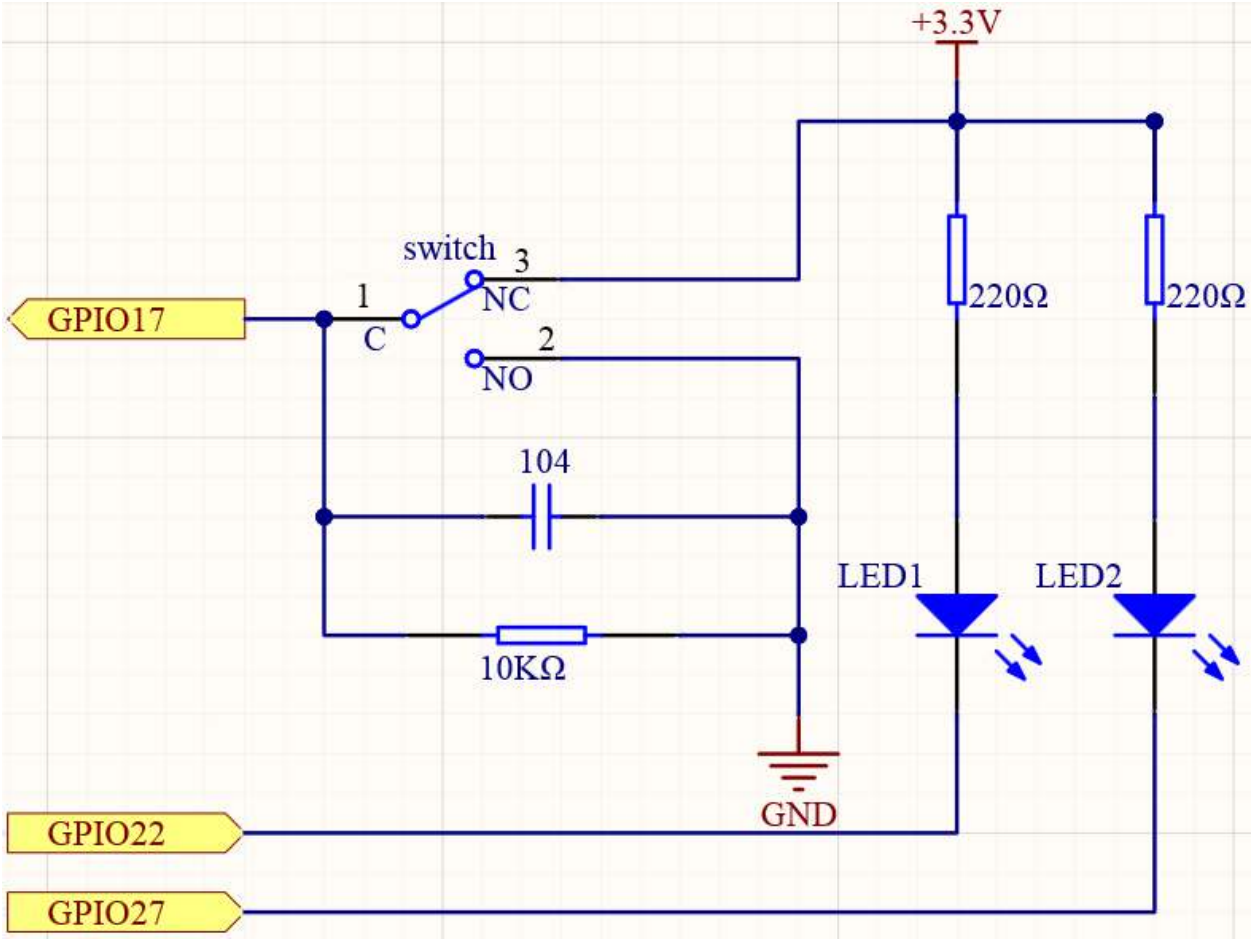
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*

- *Micro Switch*
- *Capacitor*

Schematic Diagram

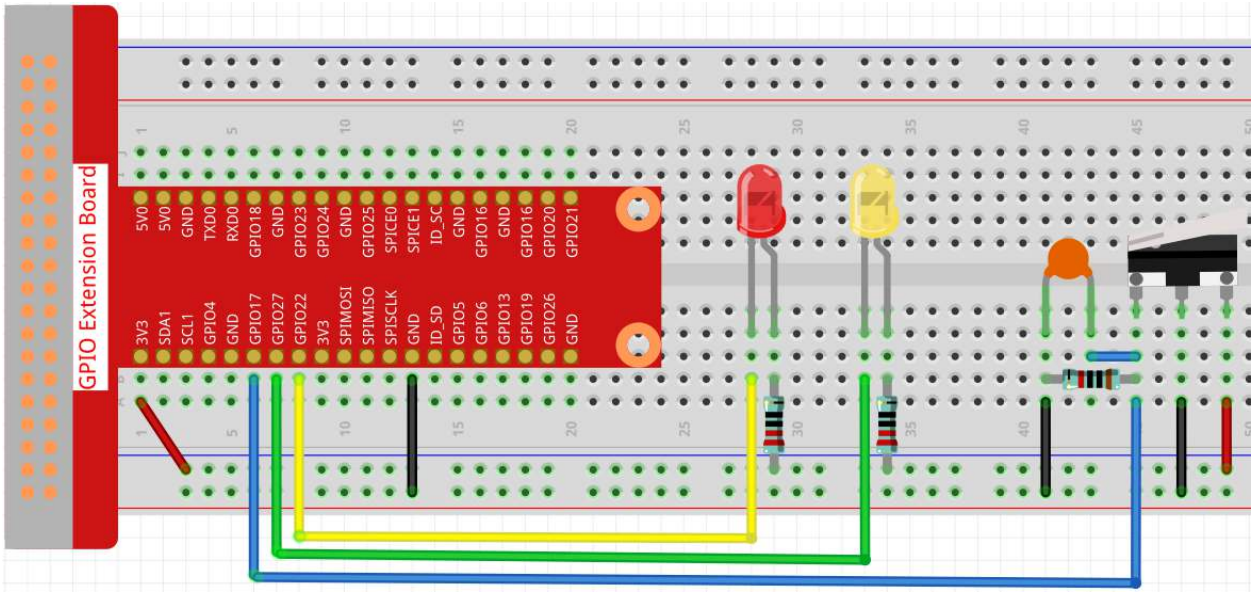
Connect the left pin of the Micro Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you press and release the move arm of the Micro Switch, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node micro_switch.js
```

While the code is running, press the Micro Switch, then the yellow LED lights up; release the moving arm, the red LED turns on.

Code

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const microSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

microSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;
```

(continues on next page)

(continued from previous page)

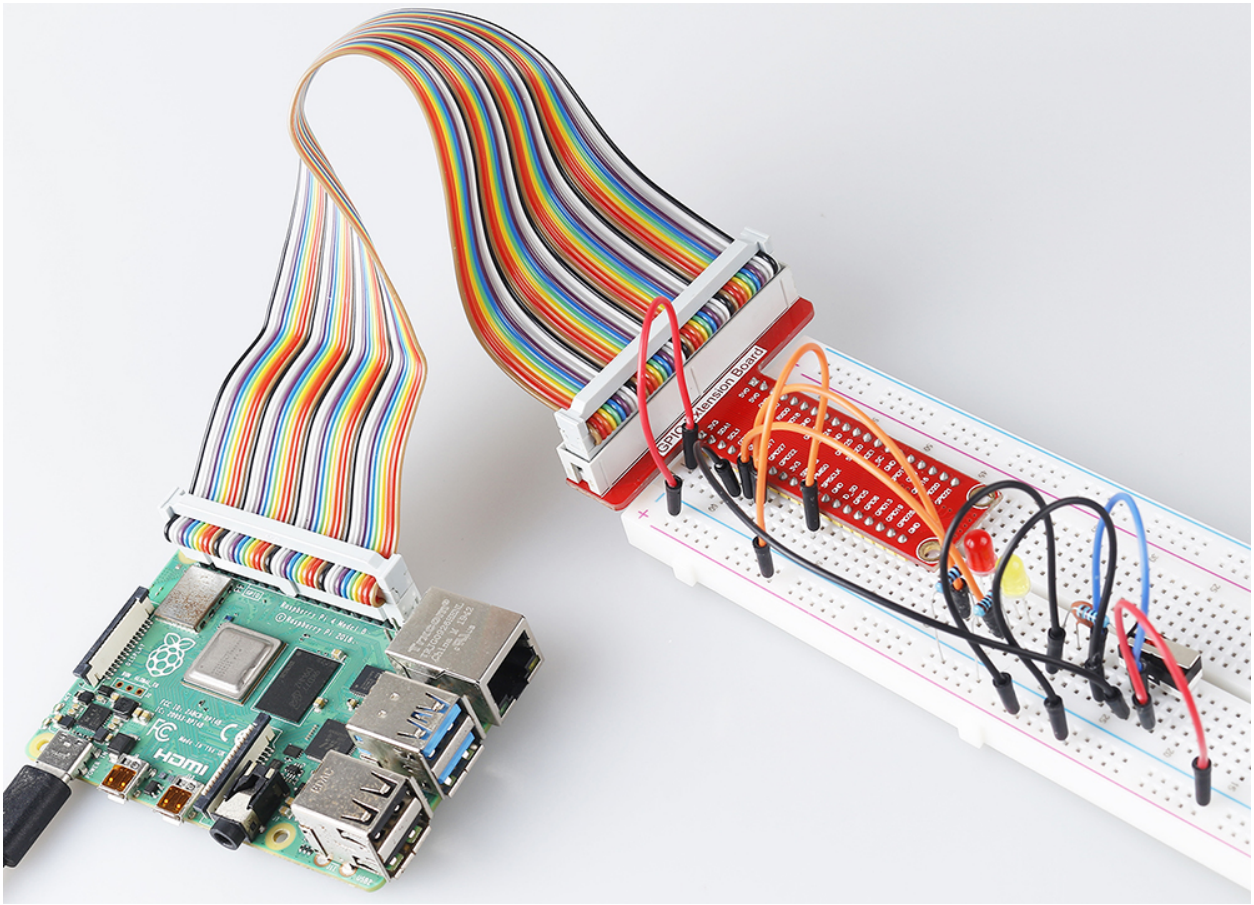
```
const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const microSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Import the `pigpio` module and create three objects led1, led2, micro, By reading the level of the micro IO port, the on and off of led1 and led2 are controlled.

```
microSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

When the level of the read micro IO port changes, Write the same level to led1 and the opposite level to led2.

Phenomenon Picture

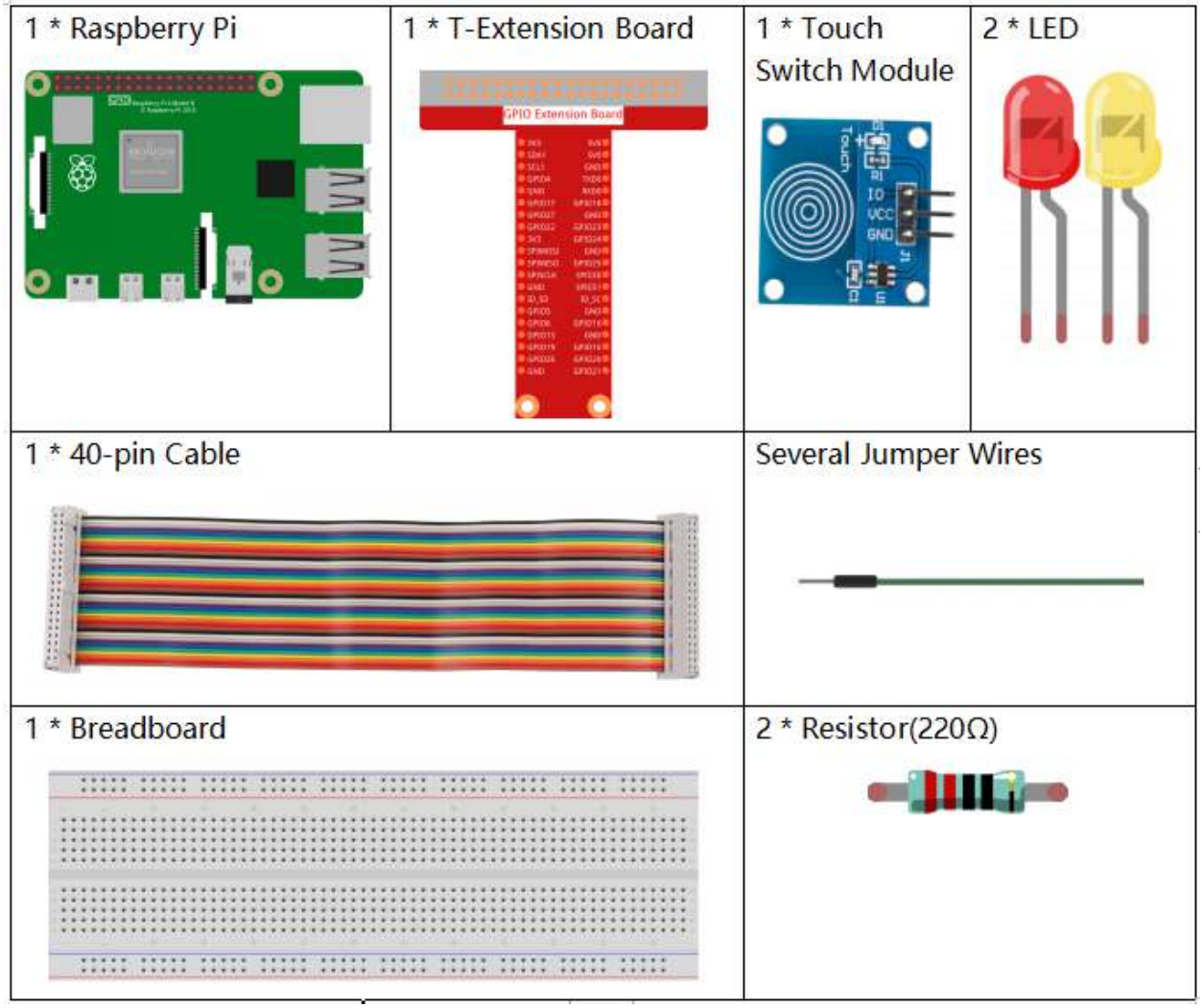


2.1.3 Touch Switch Module

Introduction

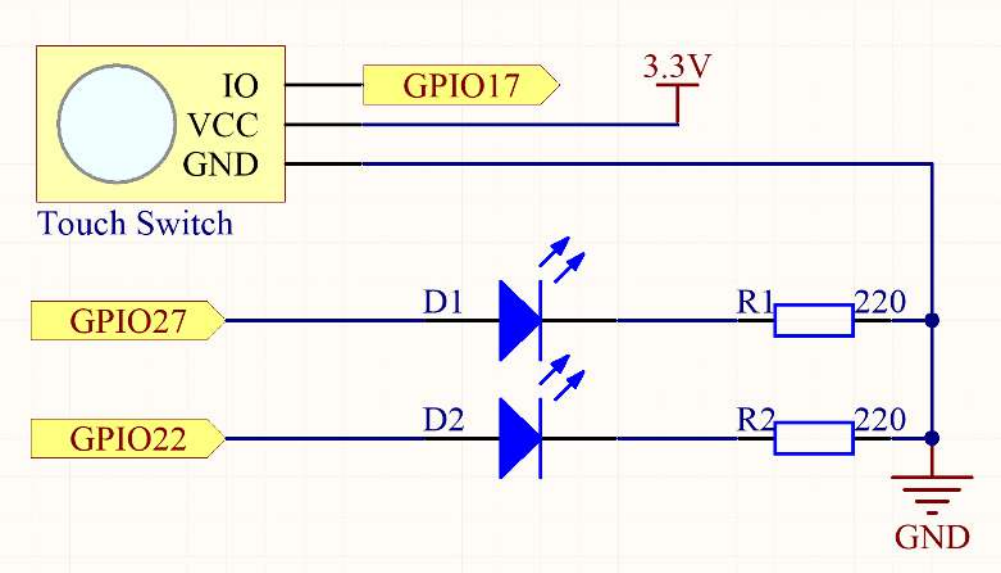
In this project, you will learn about touch switch module. It can replace the traditional kinds of switch with these advantages: convenient operation, fine touch sense, precise control and least mechanical wear.

Components



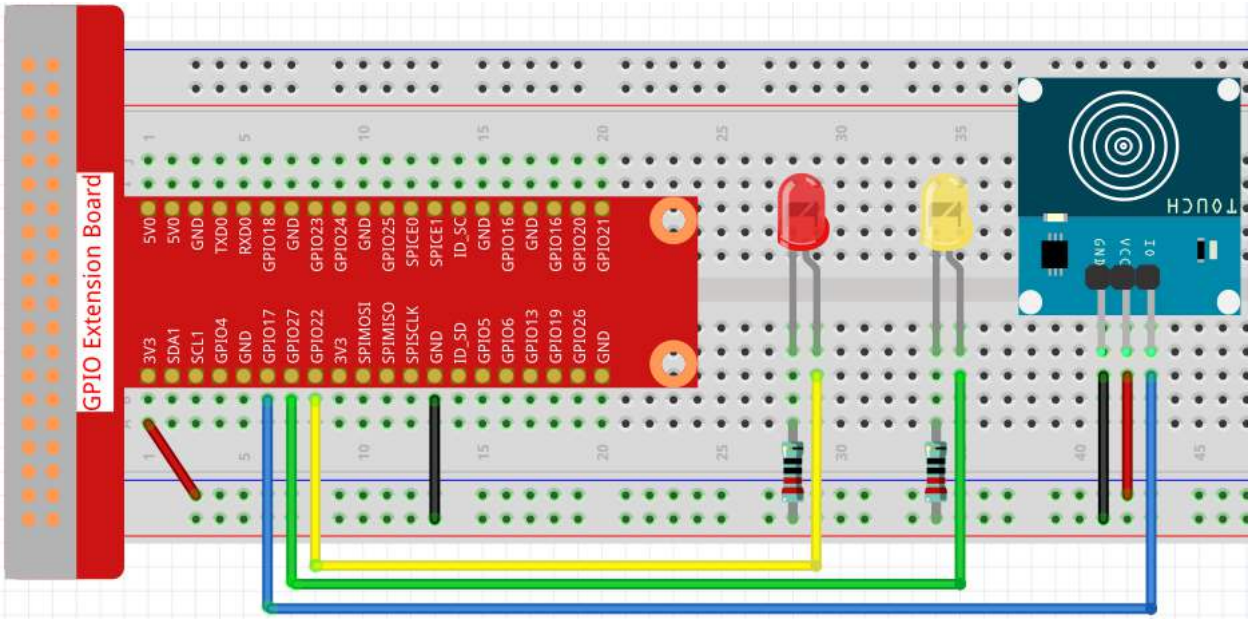
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Touch Switch Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node touch_switch.js
```

While the code is running, the red LED lights up; when you tap on the touch switch module, the yellow LED turns on.

Code

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const touchSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

touchSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

Code Explanation

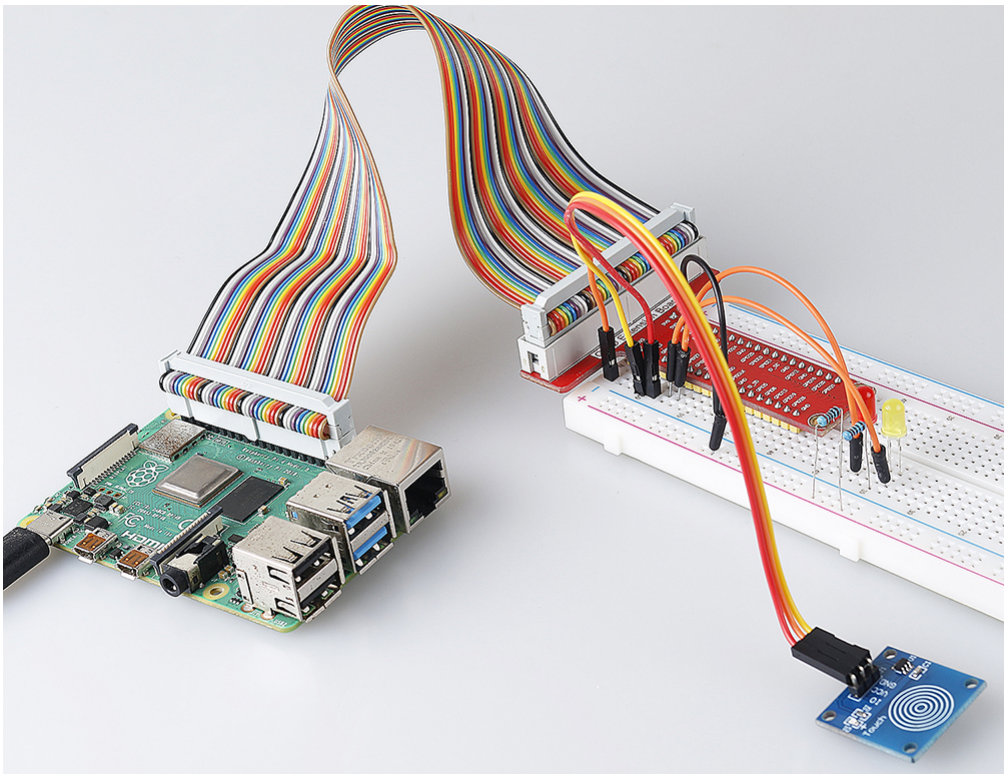
```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const touchSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Import the `pigpio` module and create three objects led1, led2, touchSwitch, By reading the level of the touchSwitch IO port, the on and off of led1 and led2 are controlled.

```
touchSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

When the level of the read touchSwitch IO port changes, Write the same level to led1 and the opposite level to led2.

Phenomenon Picture

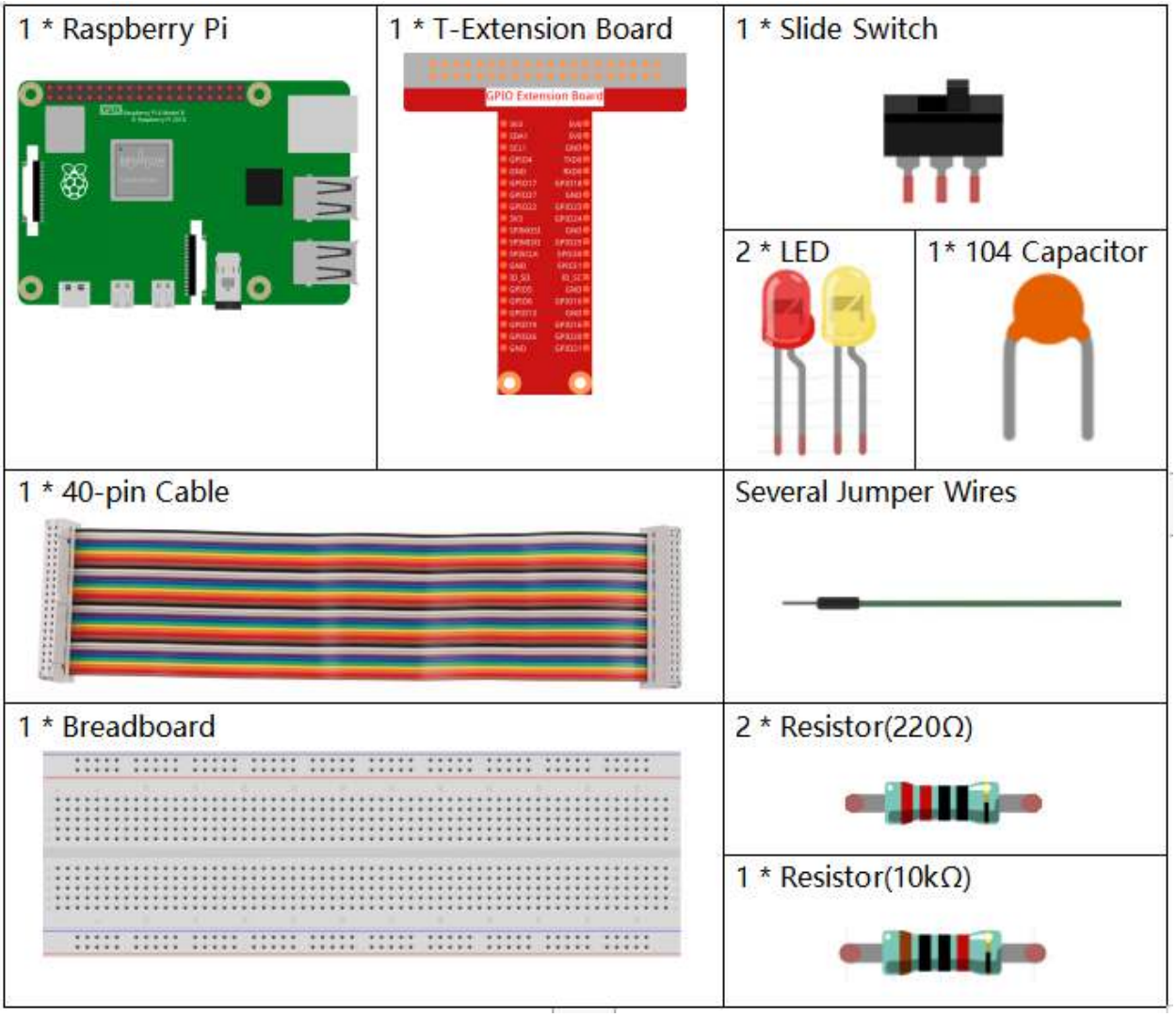


2.1.4 Slide Switch

Introduction

In this project, we will learn how to use a slide switch. Usually,the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.

Components

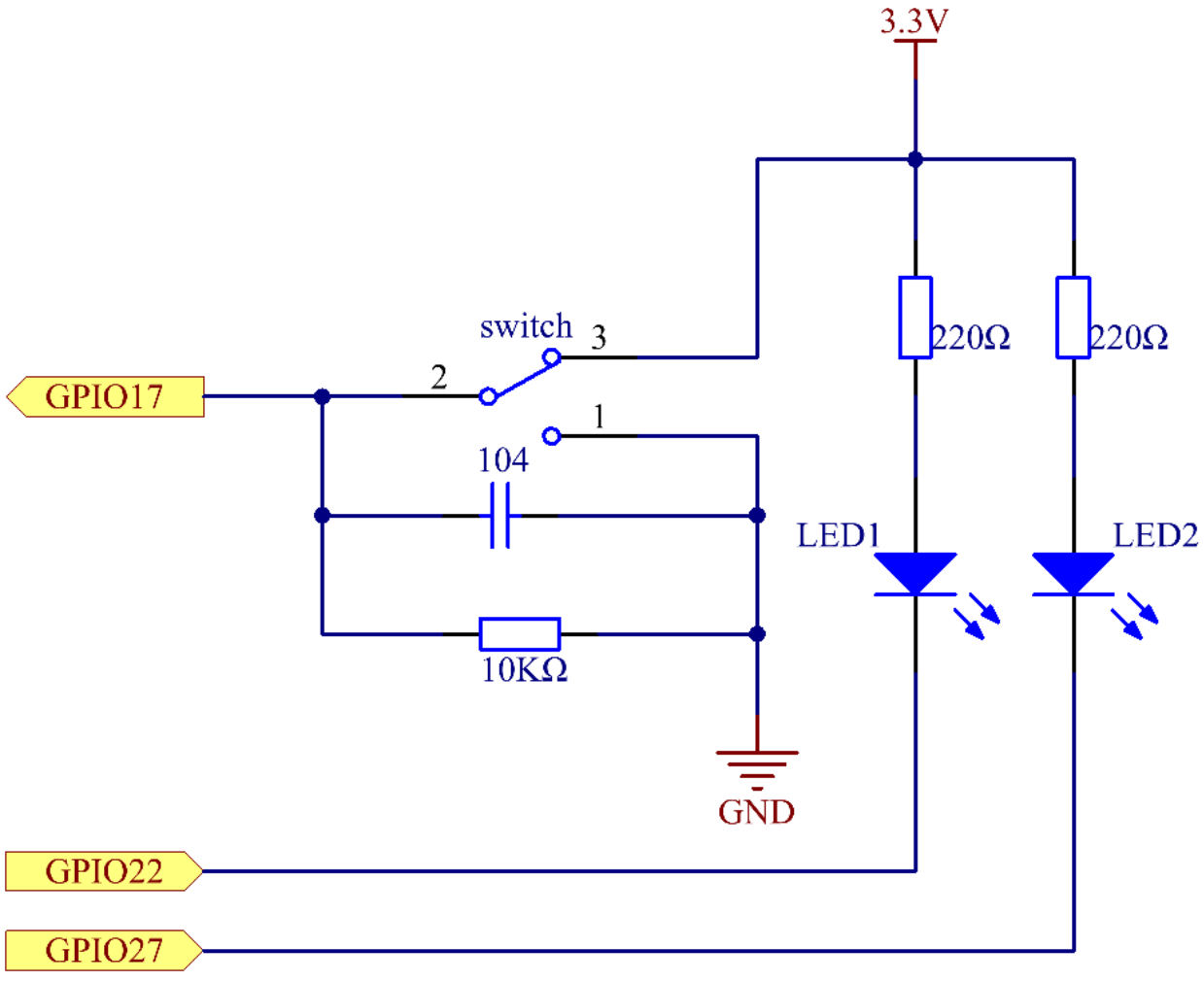


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Slide Switch*
- *Capacitor*

Schematic Diagram

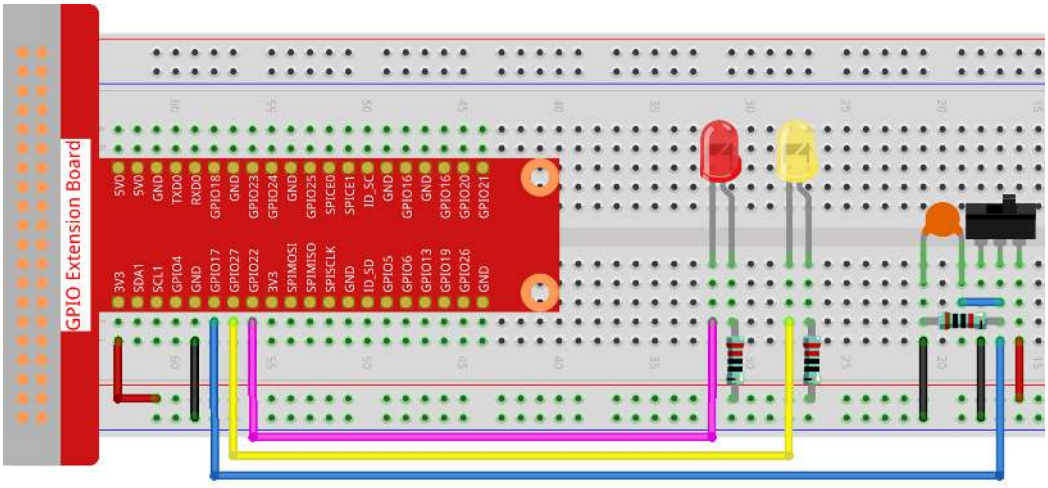
Connect the middle pin of the Slide Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node slide_switch.js
```

While the code is running, get the switch connected to the left, then the yellow LED lights up; to the right, the red light turns on.

Code

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const slideSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

slideSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});
```

(continues on next page)

(continued from previous page)

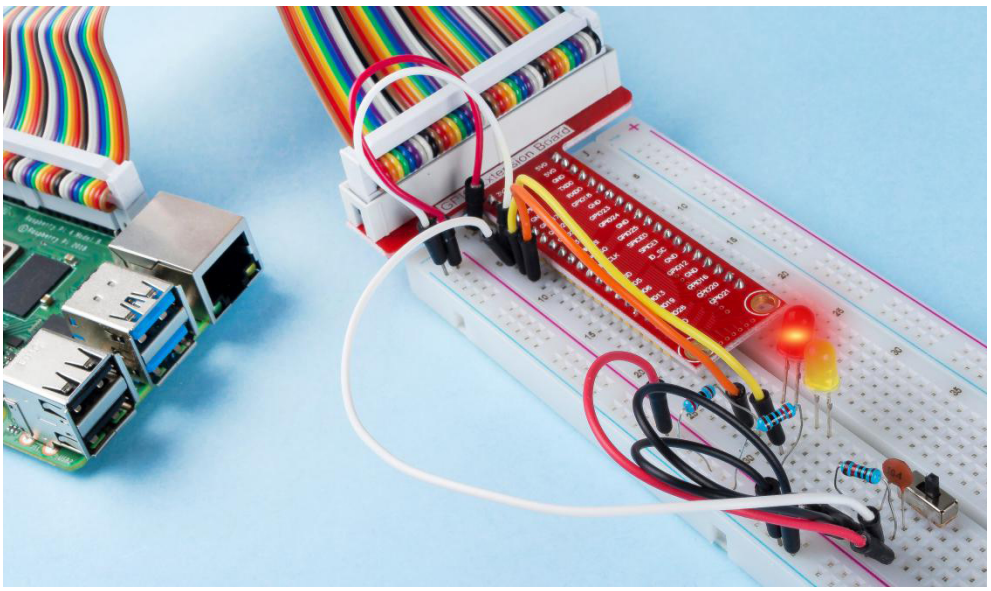
```
const slideSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Import the pigpio module, and create three objects led1, led2, slideSwitch, and control the on and off of led1 and led2 by reading the level of the slideSwitch IO port.

```
slideSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

When the read level of the slideSwitch IO port changes, Write the same level to led1 and the opposite level to led2.

Phenomenon Picture

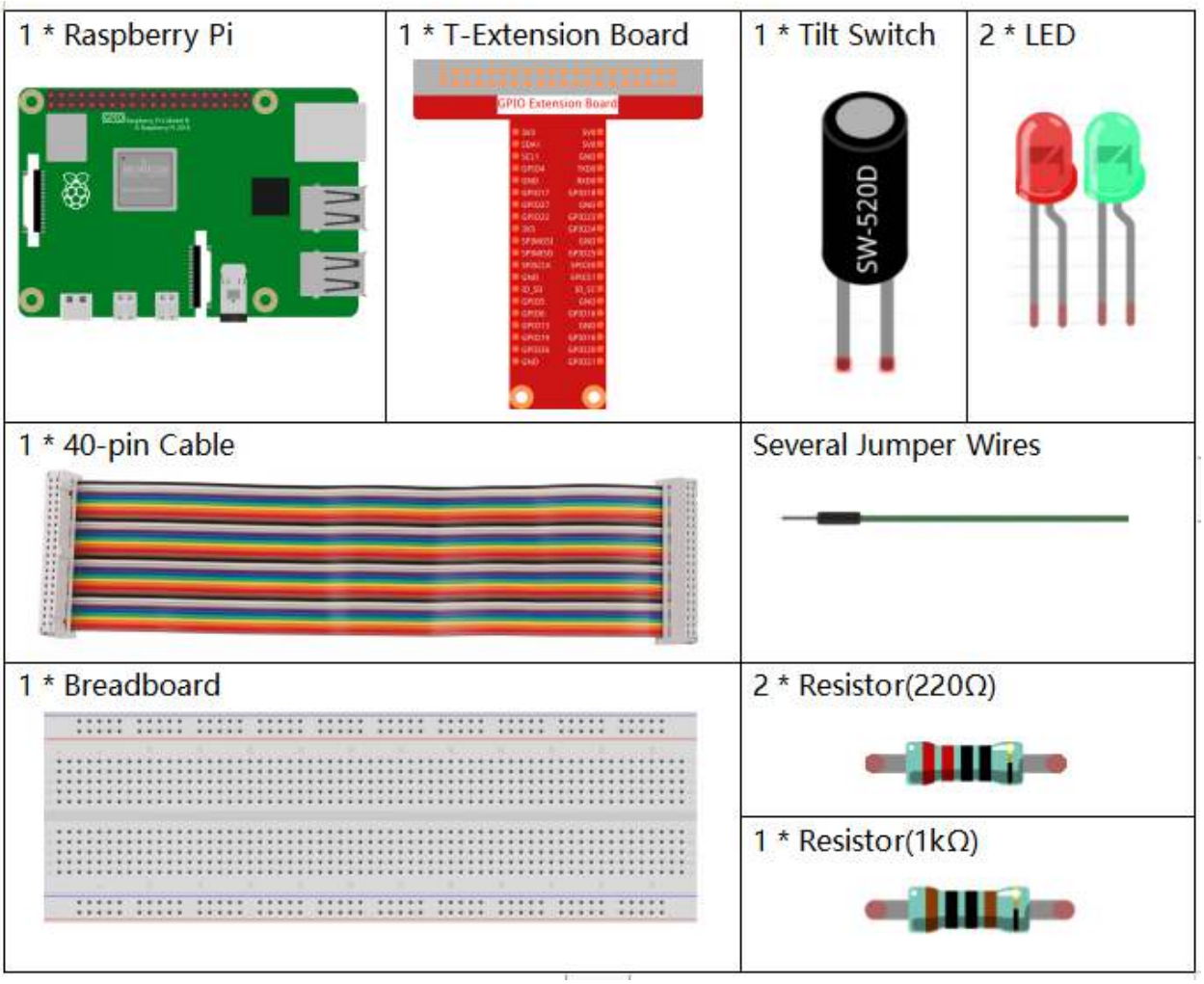


2.1.5 Tilt Switch

Introduction

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

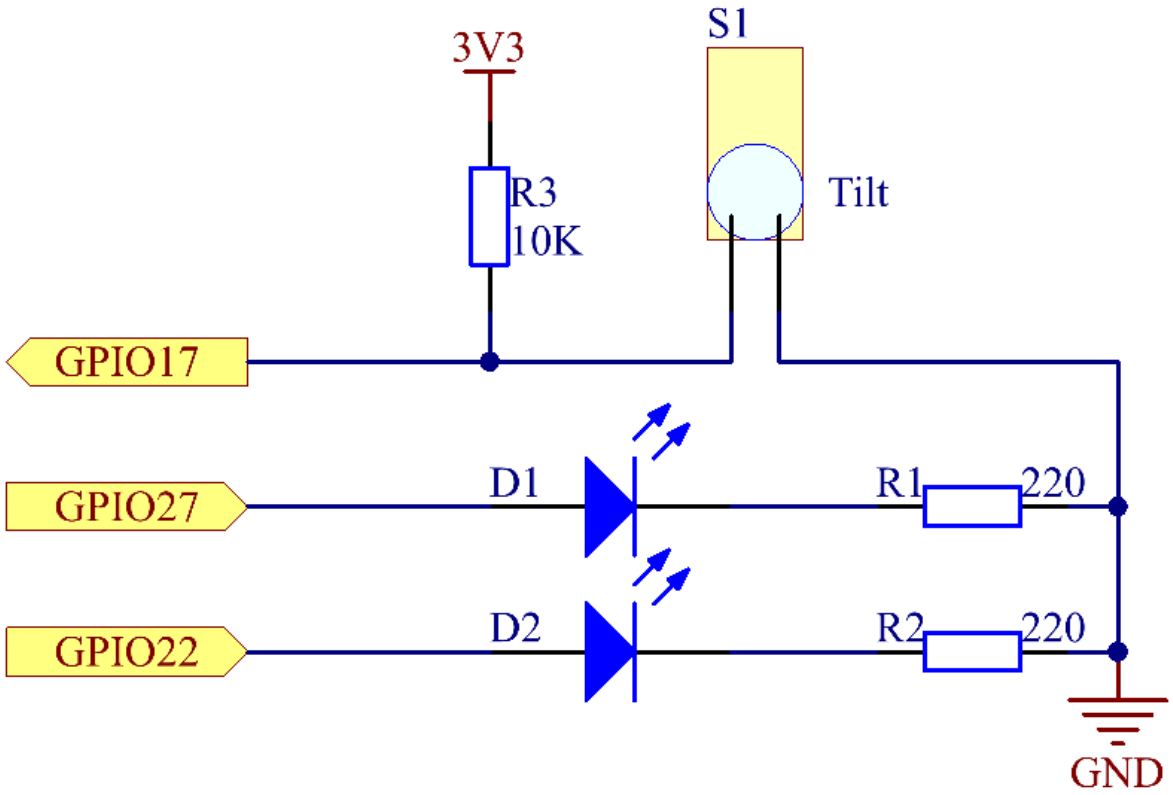
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Tilt Switch*

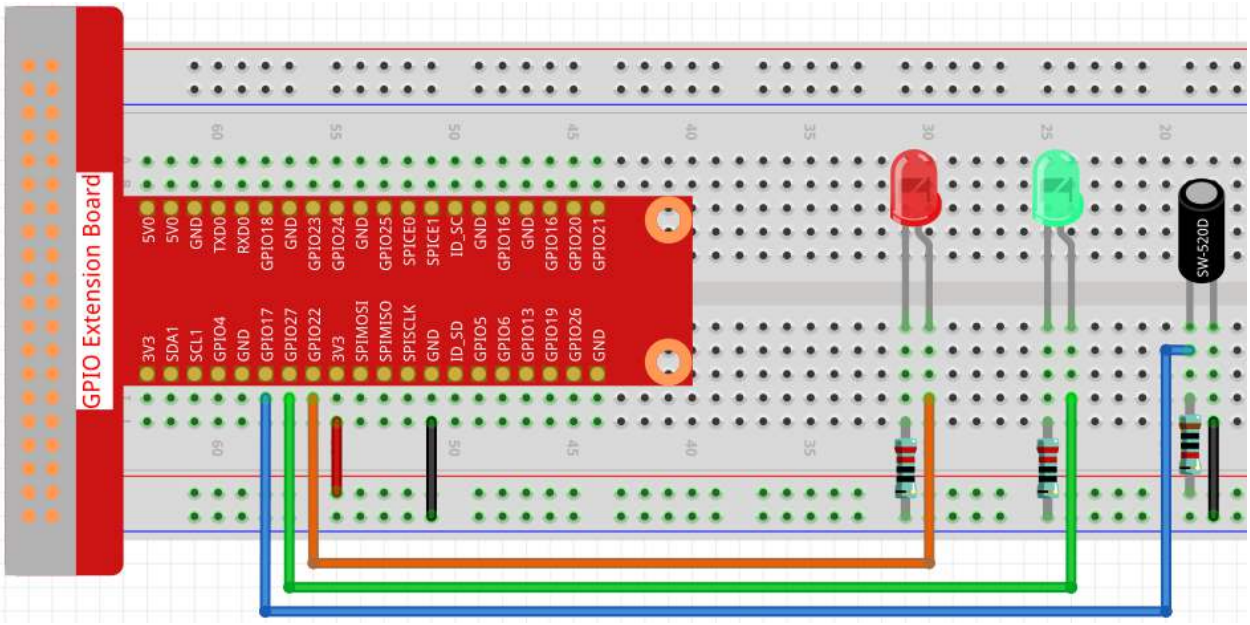
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node tilt_switch.js
```

Place the tilt vertically, and the green LED will turns on. If you tilt it, the red LED will turns on. Place it vertically again, and the green LED will lights on.

Code

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, { mode: Gpio.OUTPUT });
const led2 = new Gpio(27, { mode: Gpio.OUTPUT });

const tilt = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

tilt.on('interrupt', (level) => {
    if (level) {
        console.log("Horizontally");
    }
    else {
        console.log("Vertically");
    }
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, { mode: Gpio.OUTPUT });
const led2 = new Gpio(27, { mode: Gpio.OUTPUT });

const tilt = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Import the pigpio module and create three objects led1, led2, tilt, By reading the level of the tilt IO port, the on and off of led1 and led2 are controlled.

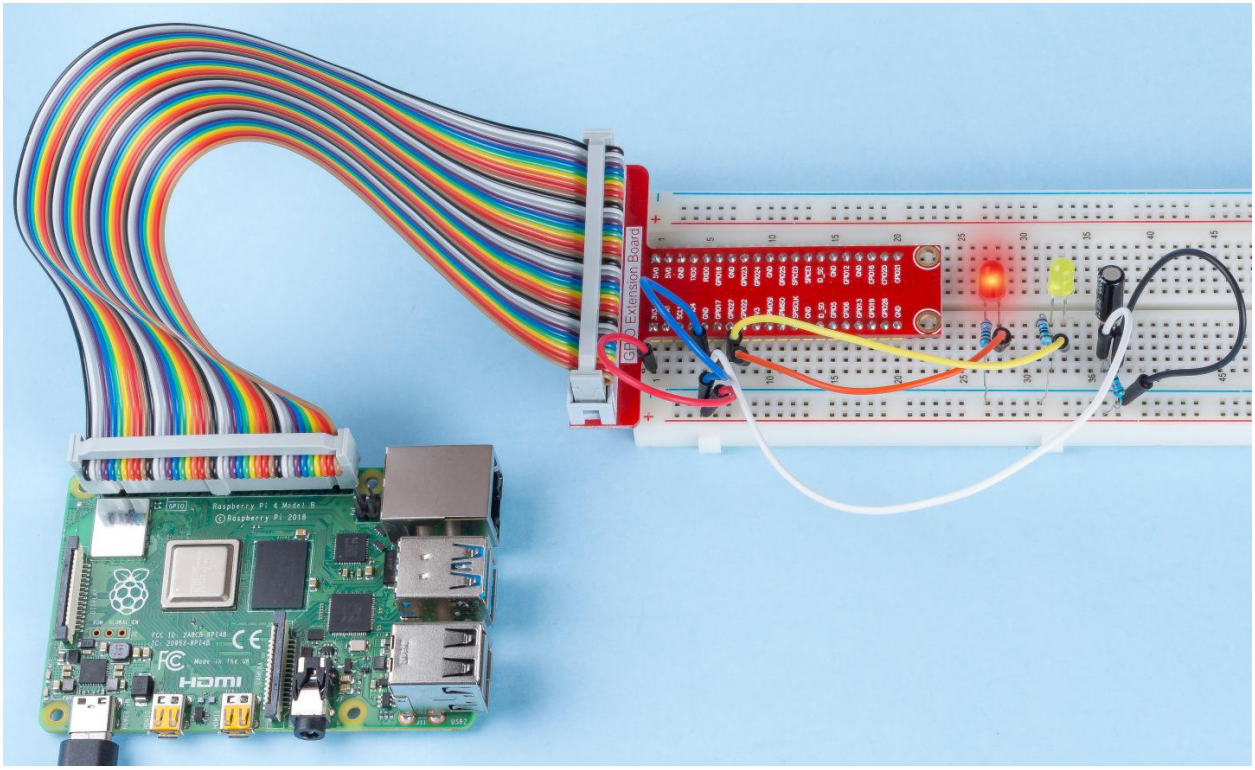
```
const tilt = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Create a tilt object to control the IO port Gpio17, set it to input mode, pull-down resistor (initially low level). And set the interrupt function, the mode is EITHER_EDGE, that is, both rising and falling edges will trigger the interrupt function.

```
tilt.on('interrupt', (level) => {
    if (level) {
        console.log("Horizontally");
    }
    else {
        console.log("Vertically");
    }
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

When the interrupt is triggered, write the same level to led1, and write the opposite level to led2. When the tilt IO port is high, the terminal prints “Horizontally”; When the tilt IO port is low, the terminal prints “Vertically”.

Phenomenon Picture

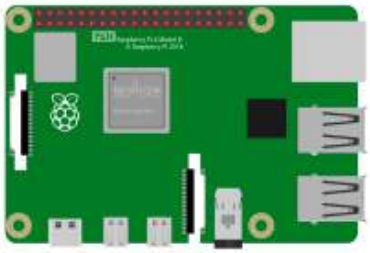
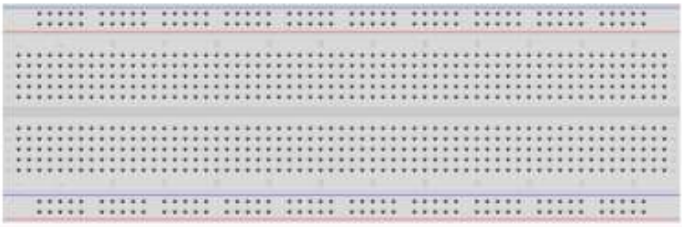


2.1.6 Rotary Encoder Module

Introduction

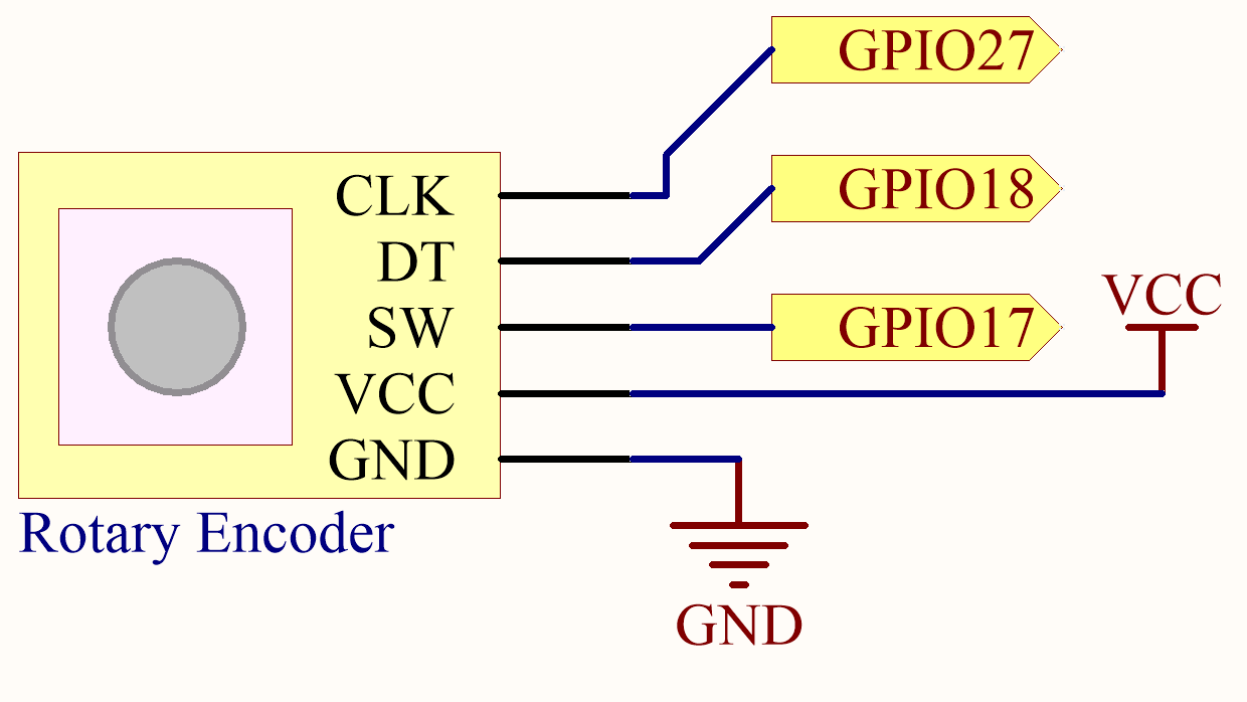
In this project, you will learn about Rotary Encoder. A rotary encoder is an electronic switch with a set of regular pulses in strictly timing sequence. When used with IC, it can achieve increment, decrement, page turning and other operations such as mouse scrolling, menu selection, and so on.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="2">1 * 40-pin Cable
 </td><td rowspan="2">1 * Rotary Encoder
 </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td></tr>
</table>

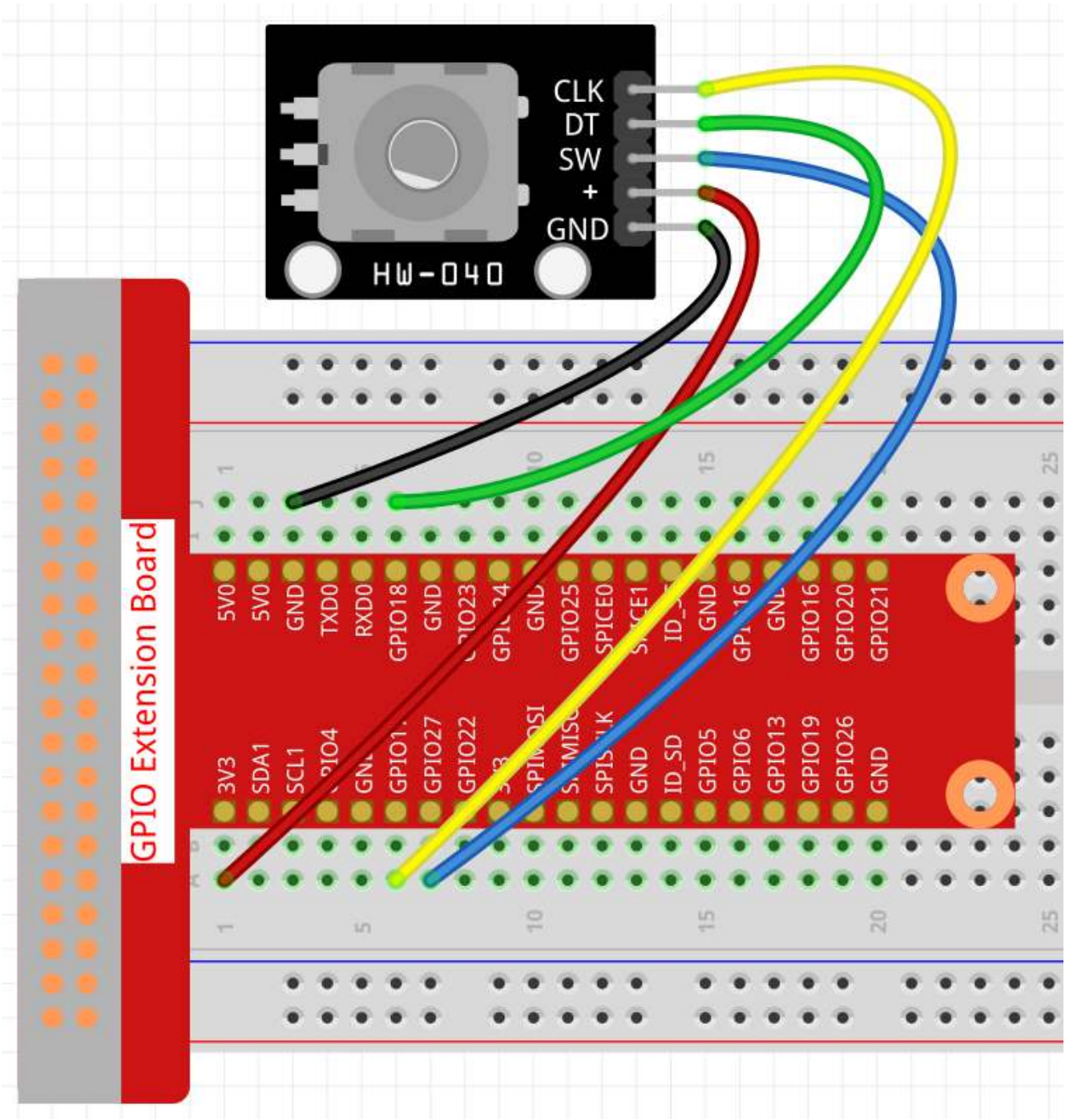
- *GPIO Extension Board*
- *Breadboard*
- *Rotary Encoder Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



In this example, we can connect the Rotary Encoder pin directly to the Raspberry Pi using a breadboard and 40-pin Cable, connect the GND of the Rotary Encoder to GND, +to 5V, SW to digital GPIO27, DT to digital GPIO18, and CLK to digital GPIO 17.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node rotary_encoder_module.js
```

You will see the count on the shell. When you turn the rotary encoder clockwise, the count is increased; when turn

it counterclockwise, the count is decreased. If you press the switch on the rotary encoder, the readings will return to zero.

Code

```
const Gpio = require('pigpio').Gpio;

const clkPin = new Gpio(17, {
  mode: Gpio.INPUT,
  pullUpDown: Gpio.PUD_DOWN,
  edge: Gpio.RISING_EDGE
});
const dtPin = new Gpio(18, {
  mode: Gpio.INPUT,
  pullUpDown: Gpio.PUD_DOWN,
});
const swPin = new Gpio(27, {
  mode: Gpio.INPUT,
  pullUpDown: Gpio.PUD_UP,
  edge: Gpio.FALLING_EDGE
});

var globalCounter = 0;

clkPin.on('interrupt',()=>{
  if(dtPin.digitalRead()==1){
      globalCounter--;
  }
  else{
      globalCounter++;
  }
  console.log(`globalCounter = ${globalCounter}`);
});

swPin.on('interrupt', () => {
  globalCounter = 0;
  console.log(`globalCounter = ${globalCounter}`);
});
```

Code Explanation

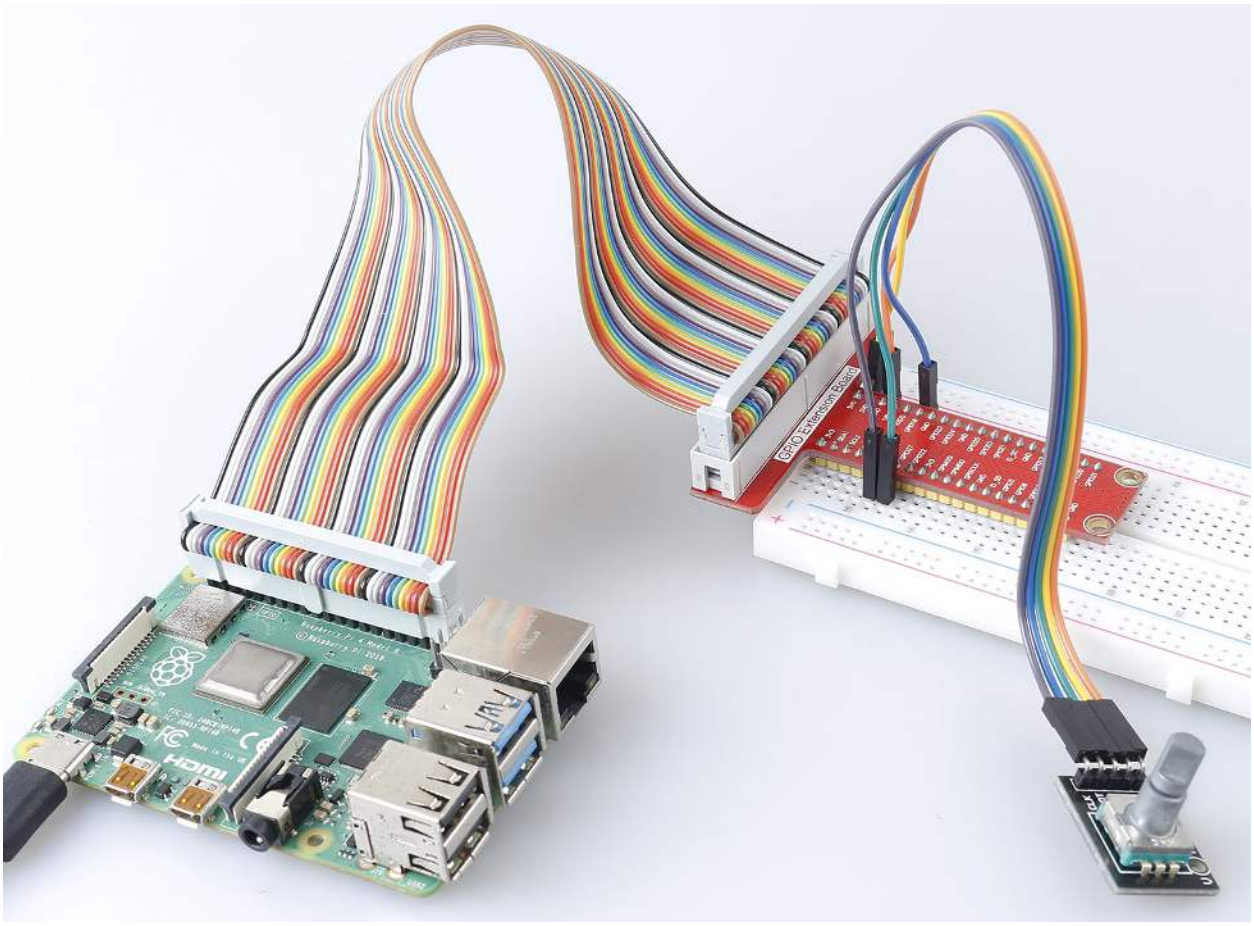
```
var globalCounter = 0;

clkPin.on('interrupt',()=>{
  if(dtPin.digitalRead()==1){
      globalCounter--;
  }
  else{
      globalCounter++;
  }
  console.log(`globalCounter = ${globalCounter}`);
});
```

When dtPin goes from low to high, if clkPin is high, the count decreases, otherwise the count increases.

```
swPin.on('interrupt', () => {
  globalCounter = 0;
  console.log(`globalCounter = ${globalCounter}`);
});
```

The swPin will output low when the shaft is pressed. Let the globalCounter go to zero at this point

Phenomenon Picture

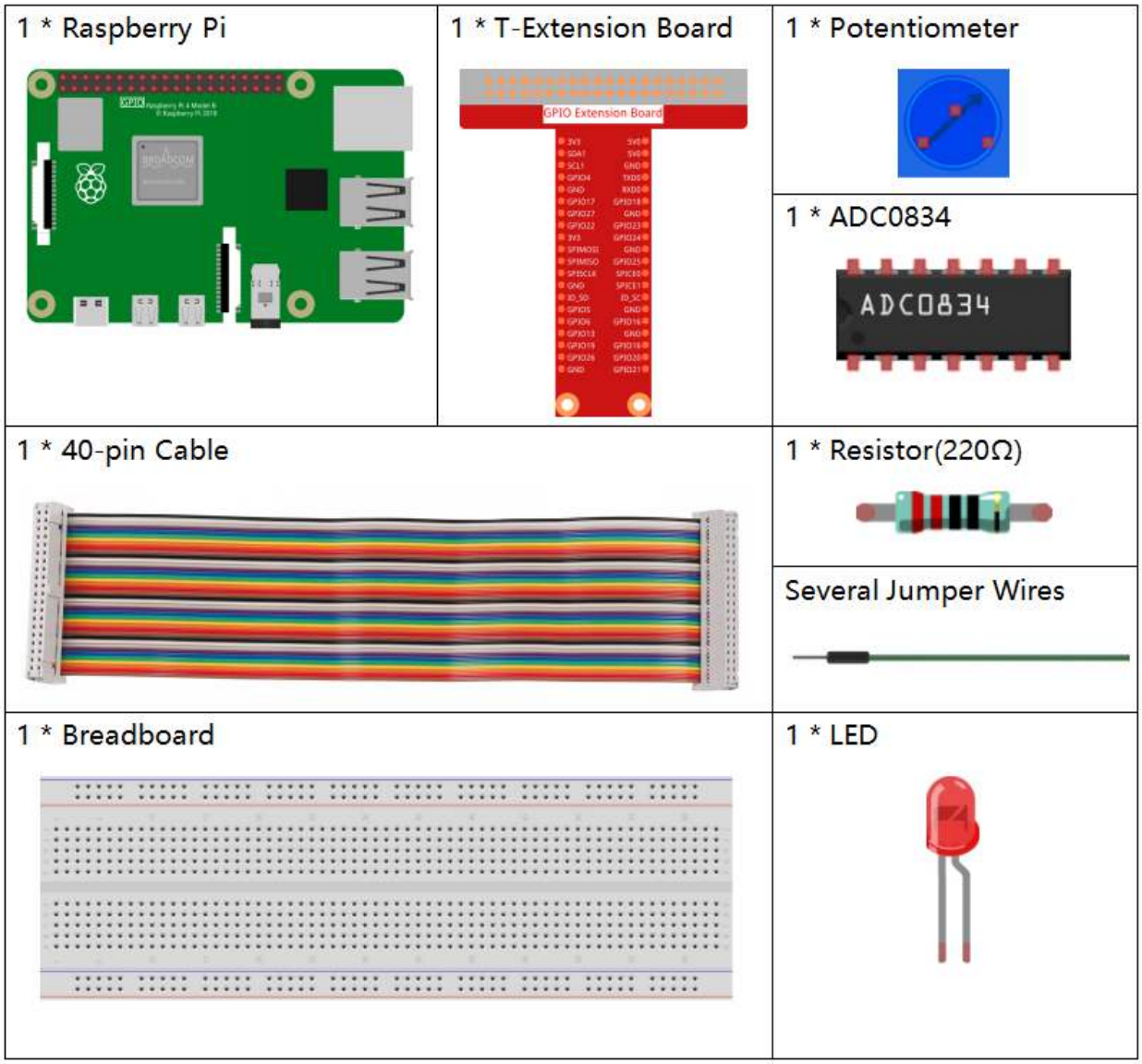


2.1.7 Potentionmeter

Introduction

The ADC function can be used to convert analog signals to digital signals, and in this experiment, ADC0834 is used to get the function involving ADC. Here, we implement this process by using potentiometer. Potentiometer changes the physical quantity – voltage, which is converted by the ADC function.

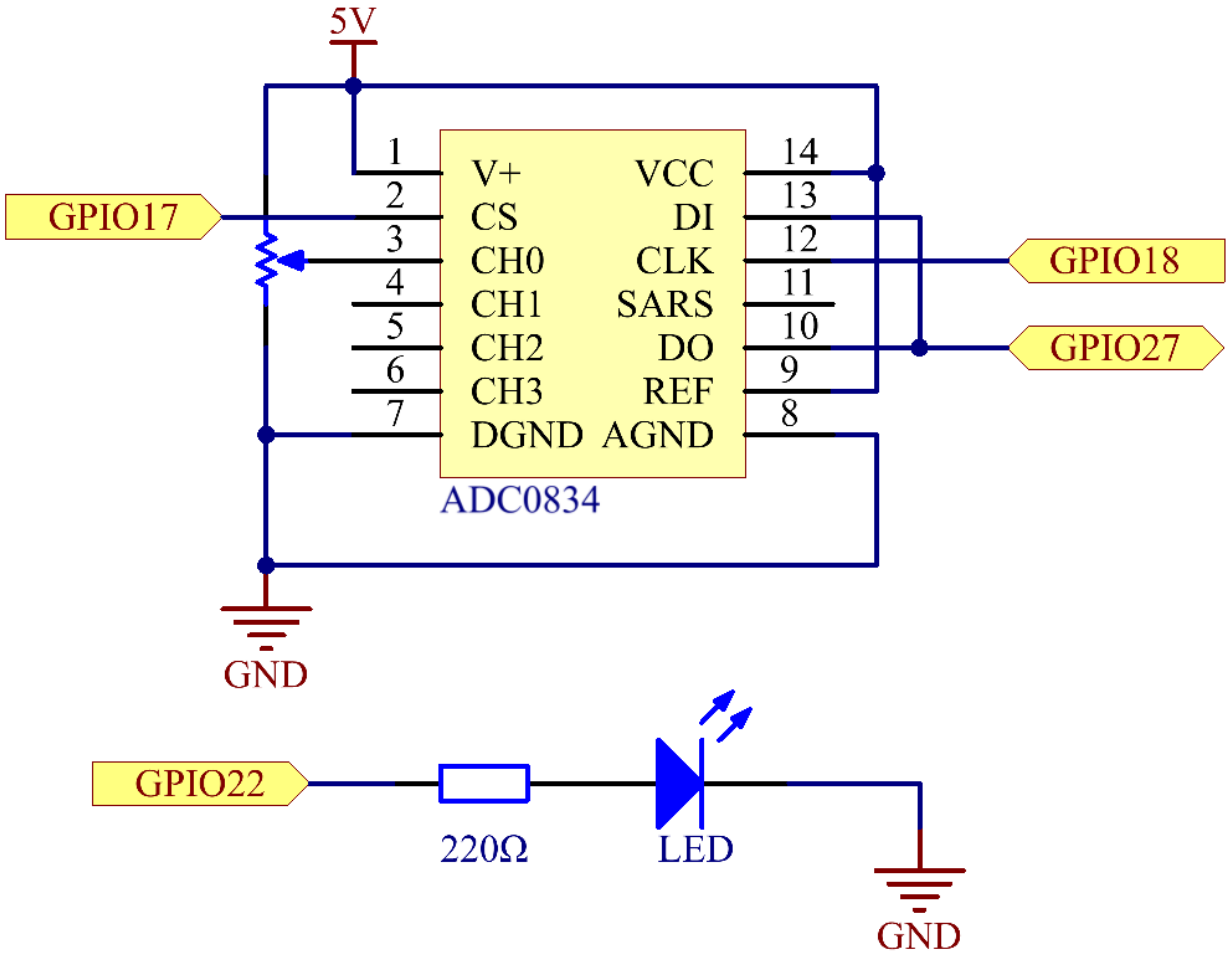
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Potentiometer*
- *ADC0834*

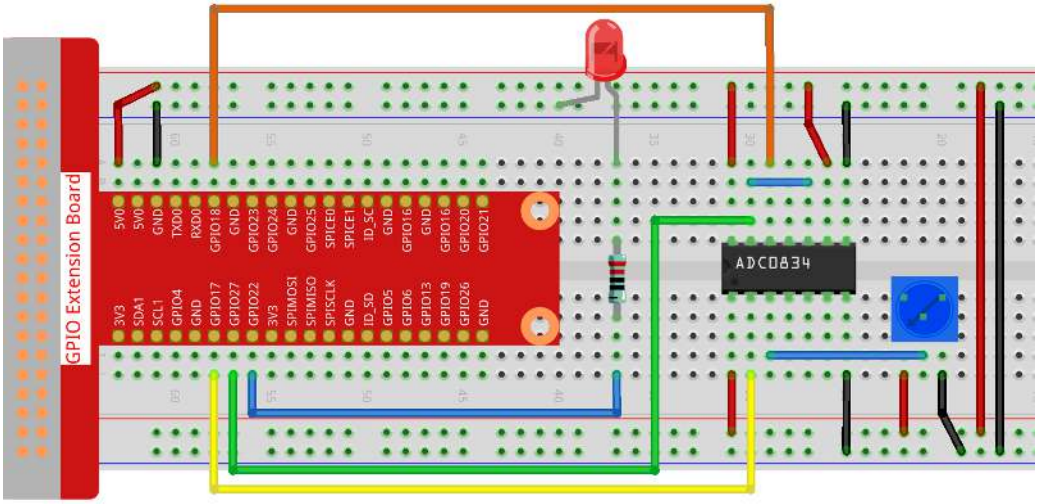
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Note: Please place the chip by referring to the corresponding position depicted in the picture. Note that the grooves on the chip should be on the left when it is placed.

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node potentionmeter.js
```

After the code runs, rotate the knob on the potentiometer, the intensity of LED will change accordingly.

Code

```
const Gpio = require('pigpio').Gpio;
const ADC0834 = require('./adc0834.js').ADC0834;

const adc = new ADC0834(17, 18, 27);
const led = new Gpio(22, {mode: Gpio.OUTPUT});

setInterval(() => {
    adc.read(0).then((value) => {
        console.log(`Current analogVal: ${value}\n`);
        led.pwmWrite(value);
    }, (error)=>{
        console.log("Error: " + error);
    });
}, 100);
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;
```

Import the `pigpio` module.

```
const ADC0834 = require('./adc0834.js').ADC0834;
```

We import an `ADC0834` constructor to use the adc0834 module.

```
const adc = new ADC0834(17, 18, 27);
```

Instantiate an ADC0834 object, the three parameters are its three pins.

This is a promise object, you may need to understand the concept from the following link.

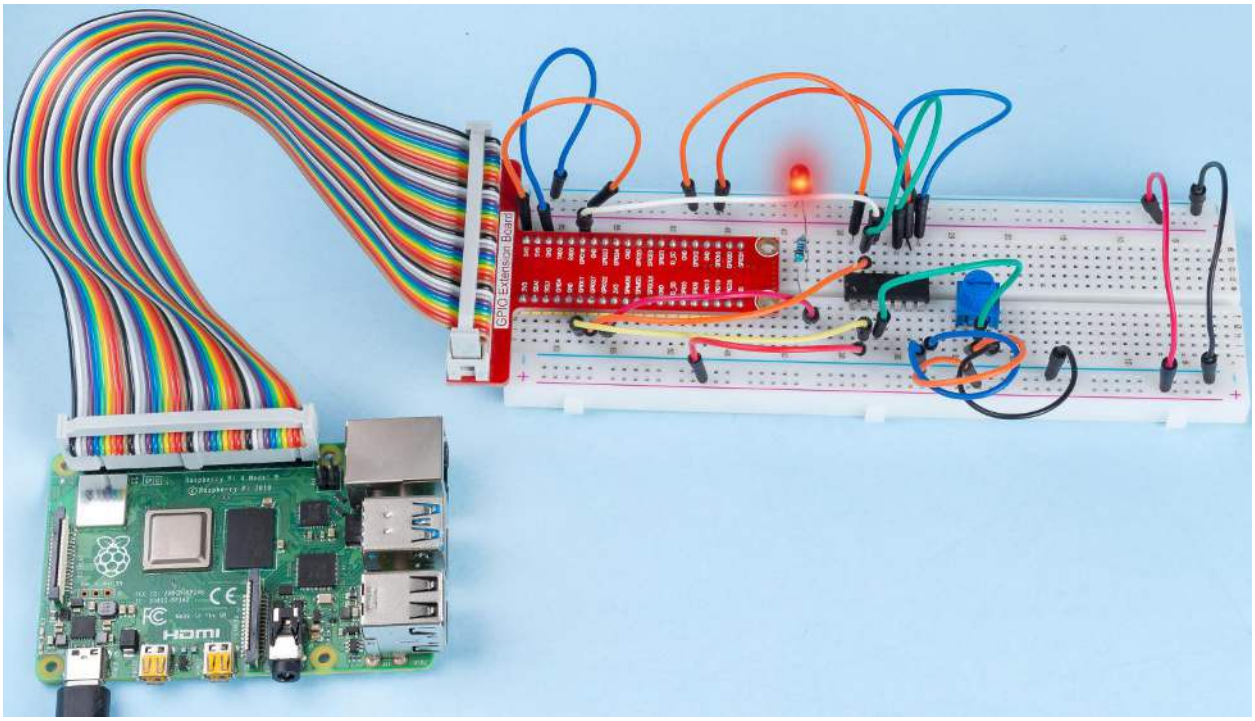
- Promise

```
setInterval(() => {
    adc.read(0).then((value) => {
        console.log(`Current analogVal: ${value}\n`);
        led.pwmWrite(value);
    }, (error)=>{
        console.log("Error: " + error);
    });
}, 100);
```

The value of ADC0834 channel 0 (channel 0 is connected to the potentiometer) is read every 100ms, and the value will be stored in `value`.

Print `value` and use it to control the brightness of the LED, now you can see that the brightness of the LED changes with the value of the potentiometer.

Phenomenon Picture

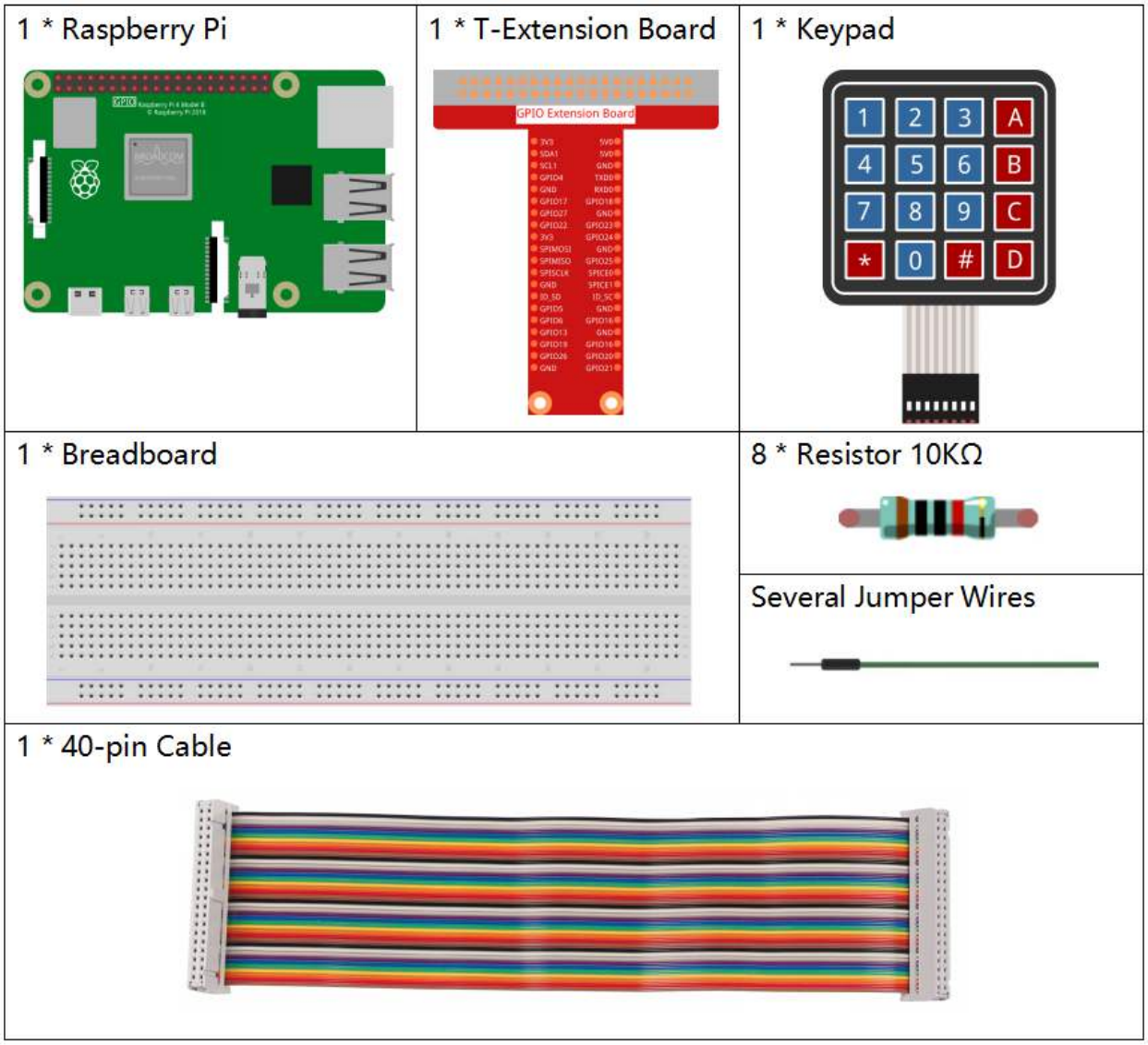


2.1.8 Keypad

Introduction

A keypad is a rectangular array of buttons. In this project, We will use it input characters.

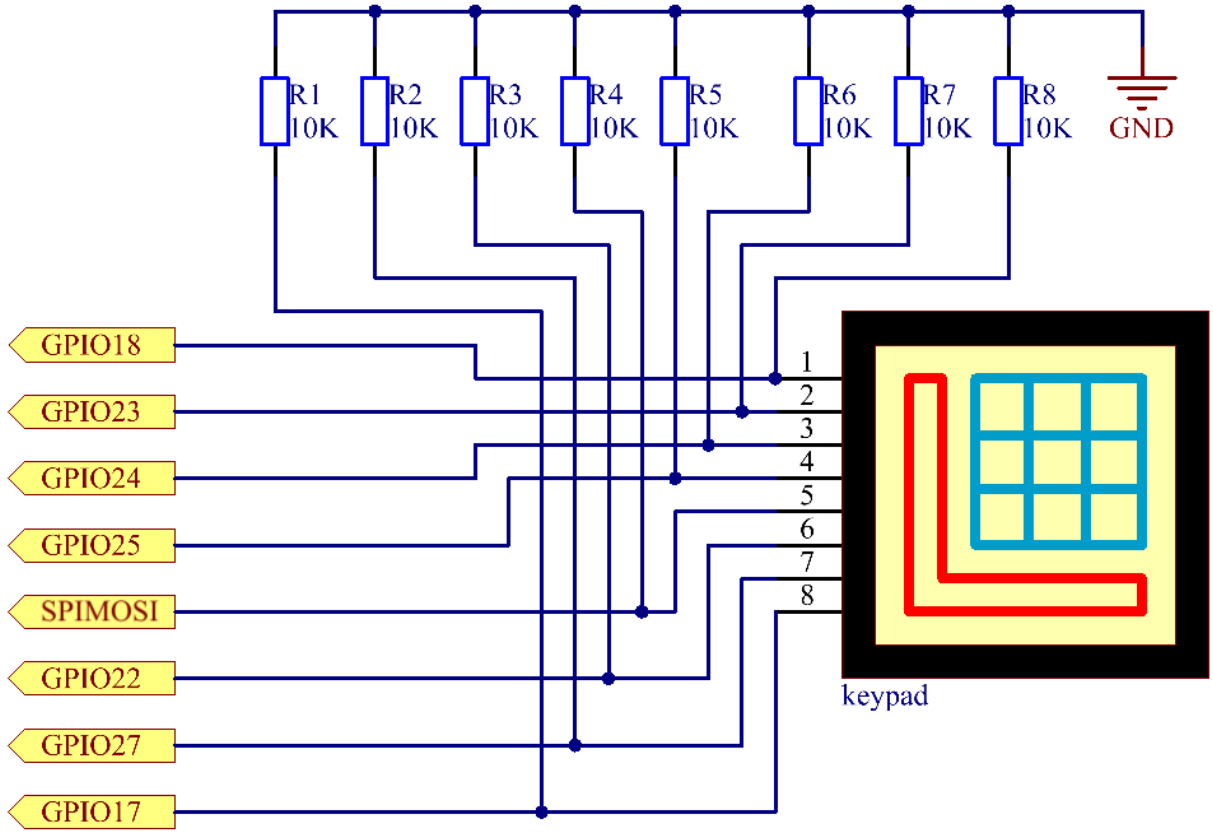
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Keypad*

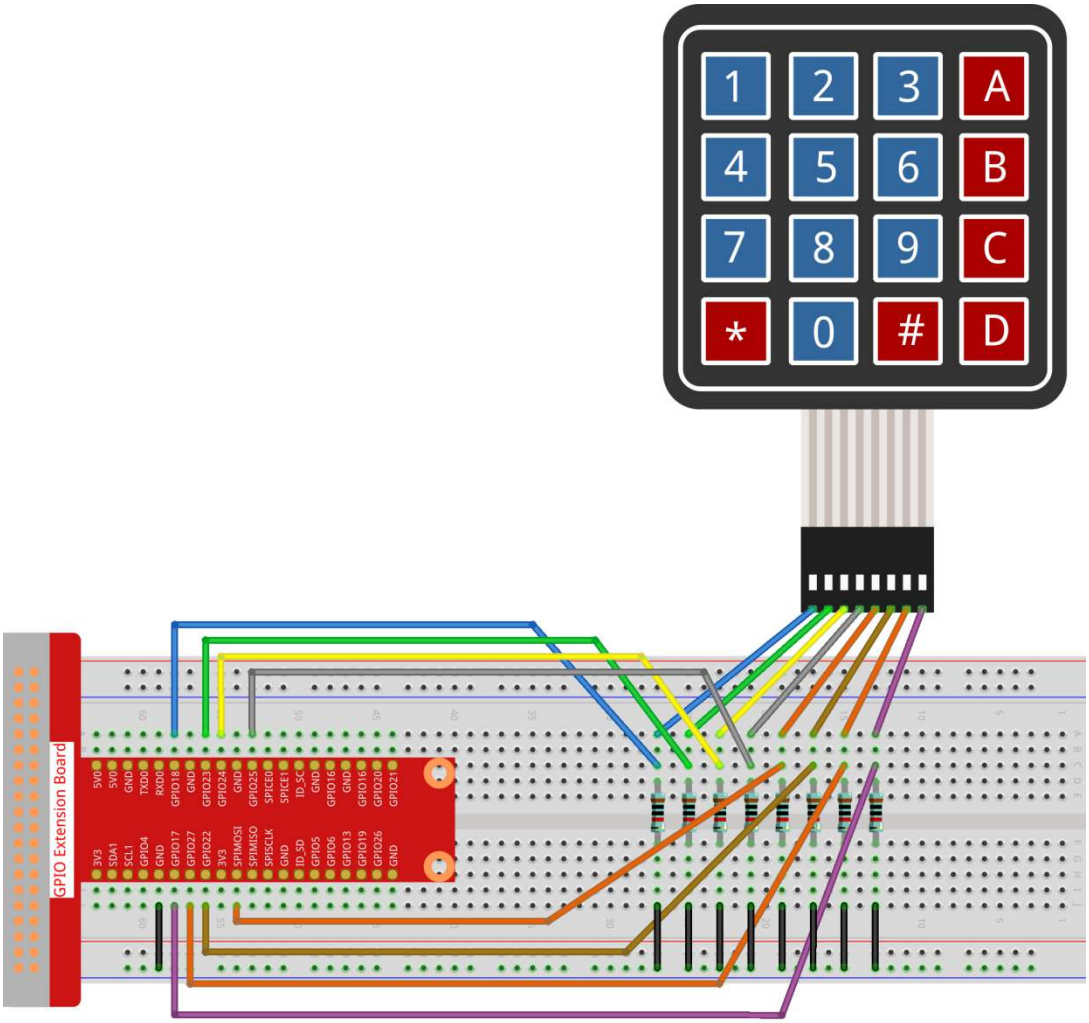
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Open the code file.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run.

```
sudo node keypad.js
```

After the code runs, the values of pressed buttons on keypad (button Value) will be printed on the screen.

Code

```
const Gpio = require('pigpio').Gpio;

var rowsPins = [18,23,24,25];
var colsPins = [10,6,27,17];
var keys = ["1","2","3","A",
        "4","5","6","B",
        "7","8","9","C",
        "*","0","#","D"];

for(let i=0;i<rowsPins.length;i++){
    rowsPins[i] = new Gpio(rowsPins[i],{mode: Gpio.OUTPUT})
```

(continues on next page)

(continued from previous page)

```
}
for(let i=0;i<colsPins.length;i++){
    colsPins[i] = new Gpio(colsPins[i],{
        mode: Gpio.INPUT,
        pullUpDown: Gpio.PUD_DOWN,
        edge: Gpio.RISING_EDGE
    })
}

var last_key_pressed = 0;

var col=-1;
for(let i=0;i<colsPins.length;i++){
    colsPins[i].on('interrupt',()=>{
        col=i;
        pressed_keys=keys[row*colsPins.length+col];
        if(last_key_pressed!=pressed_keys){
            console.log(`${pressed_keys}`);
        }
        last_key_pressed = pressed_keys;
    });
}

var row=-1;
setInterval(() => {
    row=(row+1)%rowsPins.length;
    for(let i=0;i<rowsPins.length;i++){
        rowsPins[i].digitalWrite(0);
    }
    rowsPins[row].digitalWrite(1);
}, 10);
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

var rowsPins = [18,23,24,25];
var colsPins = [10,6,27,17];
var keys = ["1","2","3","A",
        "4","5","6","B",
        "7","8","9","C",
        "*","0","#","D"];

for(let i=0;i<rowsPins.length;i++){
    rowsPins[i] = new Gpio(rowsPins[i],{mode: Gpio.OUTPUT})
}
for(let i=0;i<colsPins.length;i++){
    colsPins[i] = new Gpio(colsPins[i],{
        mode: Gpio.INPUT,
        pullUpDown: Gpio.PUD_DOWN,
        edge: Gpio.RISING_EDGE
    })
}
```

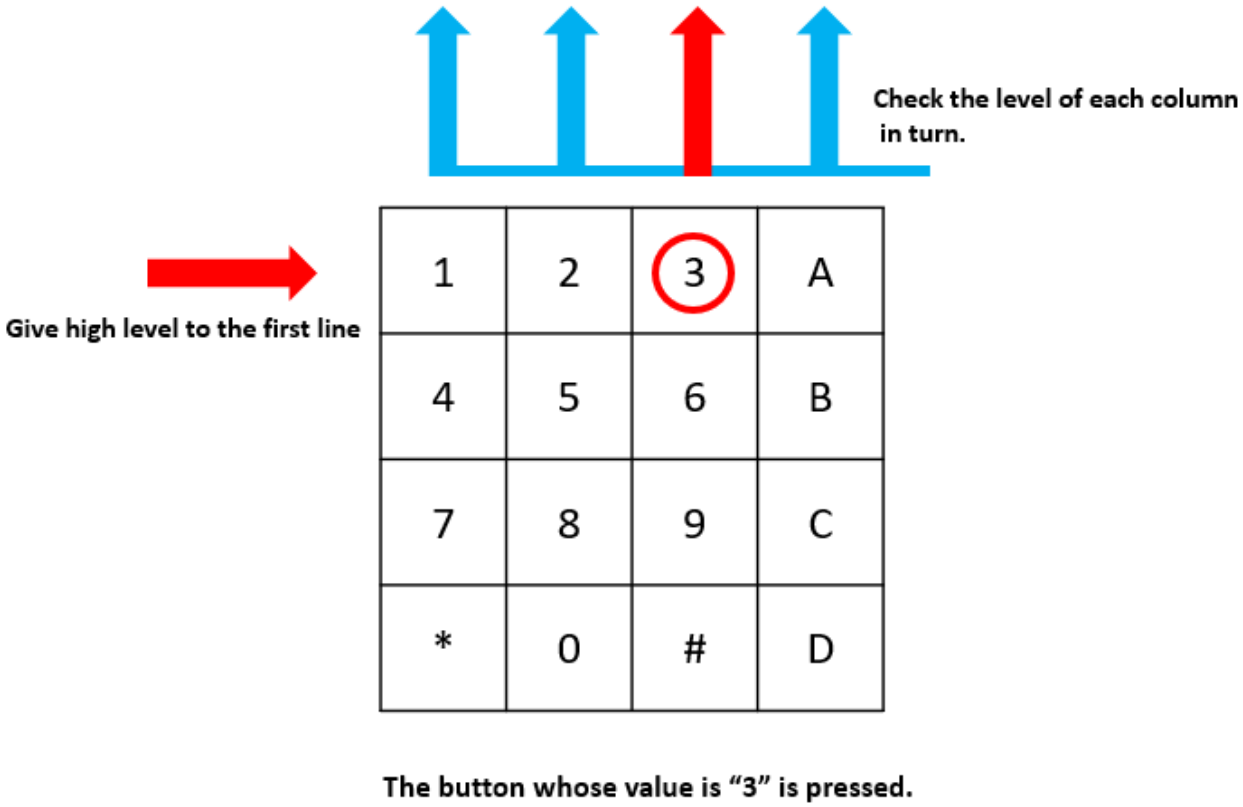
In the style of the keypad, declare two sets of pins and a matrix.

Four of the pins are the row of the keypad, set these pins to OUTPUT mode; The other four pins are the columns of the keypad, set these pins to INPUT mode, and set the rising edge interrupt for them.

The principle that the main controller can obtain the key value is as follows: The four row pins provide high level in turn, if a button is pressed, The corresponding column pin will get the high level released by the row pin, which will trigger the rising edge interrupt.

For example, if I press the button **3** , when the 2nd, 3rd, and 4th row pins release the high level, there is no effect; When the row pin in the first position releases the high level, the third column pin will be able to obtain the high level and trigger the rising edge interrupt. At this time, the 1st, 2nd, and 4th column pins do not trigger any events.

Then, according to the 1st row pin that releases the high level and the 3rd column pin that gets the high level, the main controller will be able to know that the position of the pressed button is (1,3), which is the button **3**.




```
var row=-1;
setInterval(() => {
    row=(row+1)%rowsPins.length;
    for(let i=0;i<rowsPins.length;i++){
        rowsPins[i].digitalWrite(0);
    }
    rowsPins[row].digitalWrite(1);
}, 10);
```

The four row pins are periodically supplied high, and the variable `row` is used to locate the currently working row pin.

```
var col=-1;
for(let i=0;i<colsPins.length;i++){
    colsPins[i].on('interrupt',()=>{
        col=i;
        // pressed_keys=keys[row*colsPins.length+col];
        // if(last_key_pressed!=pressed_keys){
        //     console.log(`${pressed_keys}`);
```

(continues on next page)

(continued from previous page)

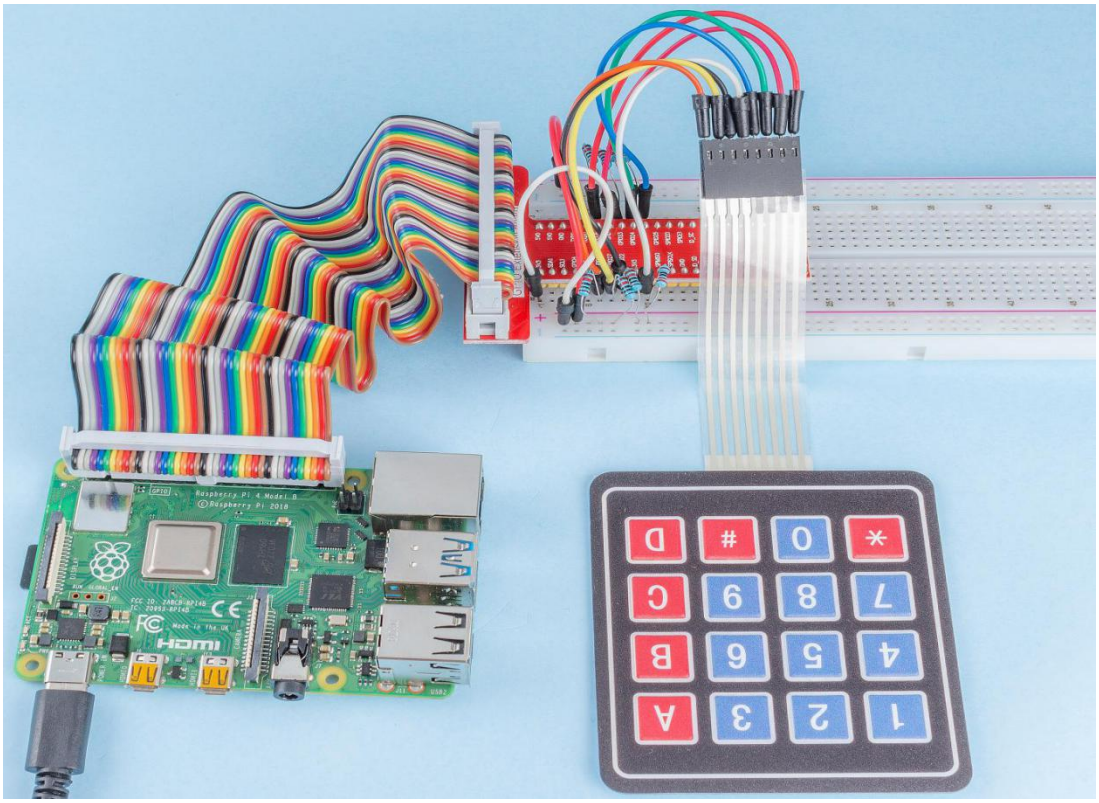
```
        // }
        // last_key_pressed = pressed_keys;
    });
}
```

Set up interrupt functions for the four column pins, and the variable `col` is used to locate the column pins that trigger the rising edge interrupt event.

```
pressed_keys=keys[row*colsPins.length+col];
if(last_key_pressed!=pressed_keys){
    console.log(`${pressed_keys}`);
}
last_key_pressed = pressed_keys;
```

There is also a piece of code in the break function to get the specific key value from the `keys` matrix according to `row` and `col`. And every time you get a new key value, print the value.

Phenomenon Picture

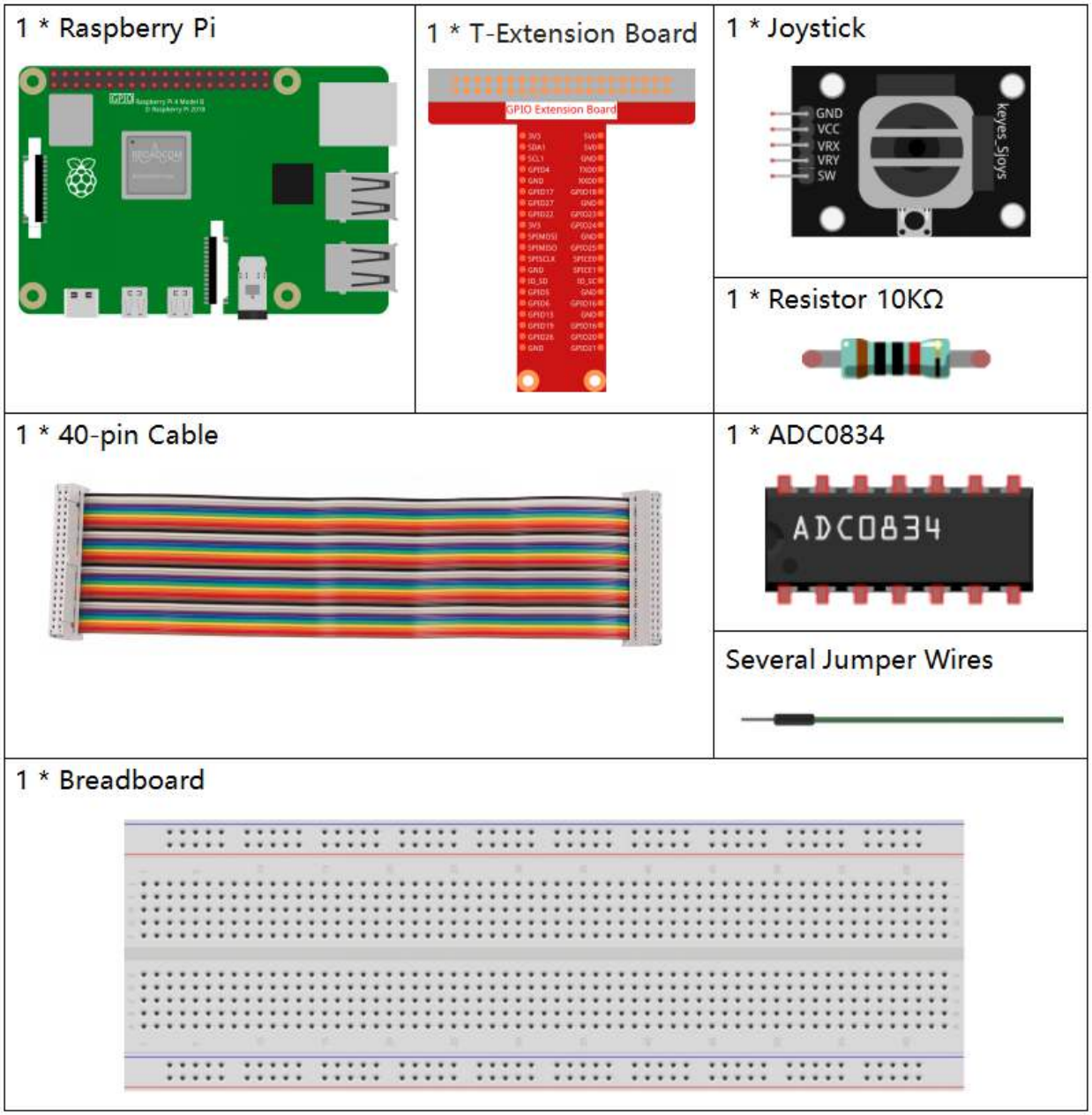


2.1.9 Joystick

Introduction

In this project, We're going to learn how joystick works. We manipulate the Joystick and display the results on the screen.

Components



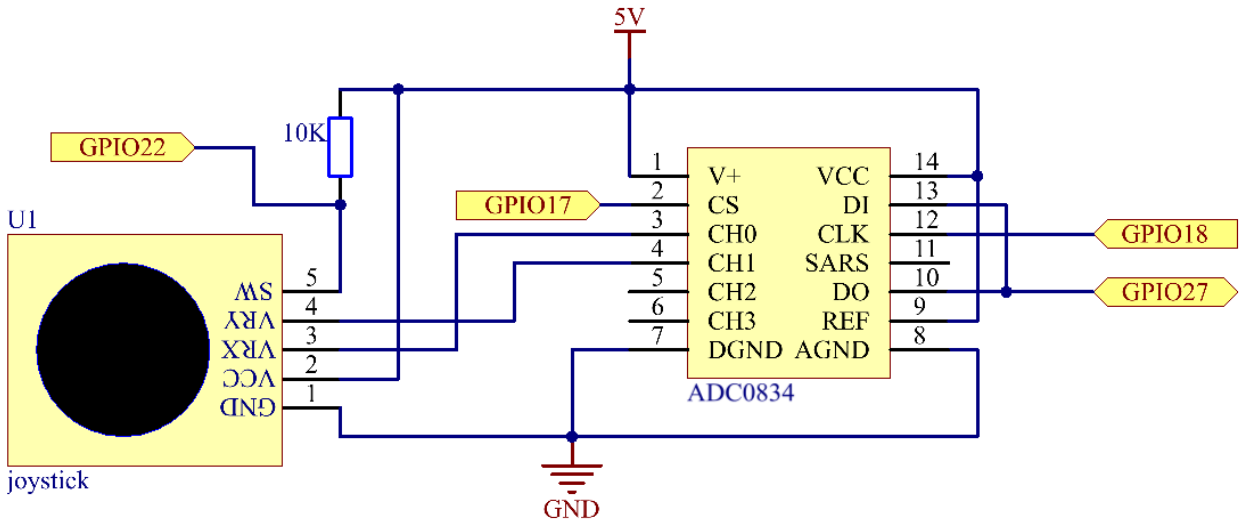
- *GPIO Extension Board*
- *Breadboard*

- *Resistor*
- *Joystick Module*
- *ADC0834*

Schematic Diagram

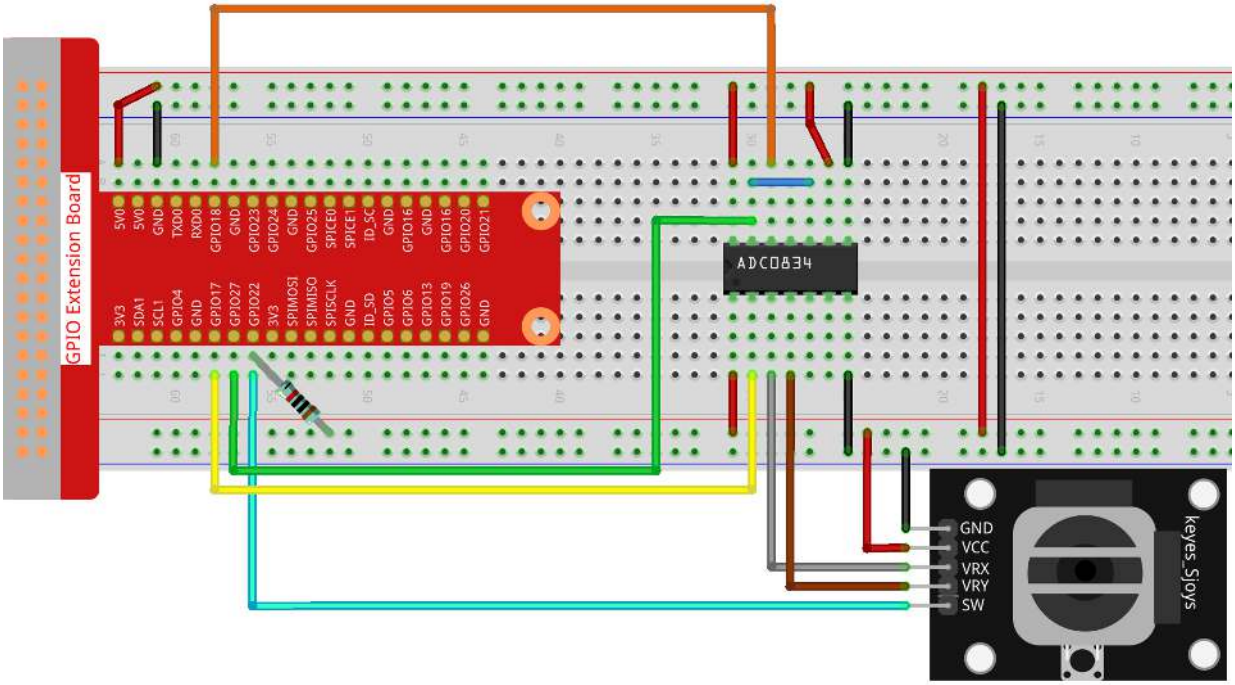
When the data of joystick is read, there are some differents between axis: data of X and Y axis is analog, which need to use ADC0834 to convert the analog value to digital value. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node joystick.js
```

After the code runs, turn the Joystick, then the corresponding values of x, y, Btn are displayed on screen.

Code

```
const Gpio = require('pigpio').Gpio;
const ADC0834 = require('./adc0834.js').ADC0834;

const adc = new ADC0834(17, 18, 22);
const btn = new Gpio(25, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_UP,
});

setInterval(async() => {

    x_val = await adc.read(0);
    y_val = await adc.read(1);

    btn_val = btn.digitalRead();
    console.log(`x = ${x_val}, y = ${y_val}, btn = ${btn_val}\n`);
}, 100);
```

Code Explanation

```
const ADC0834 = require('./adc0834.js').ADC0834;
```

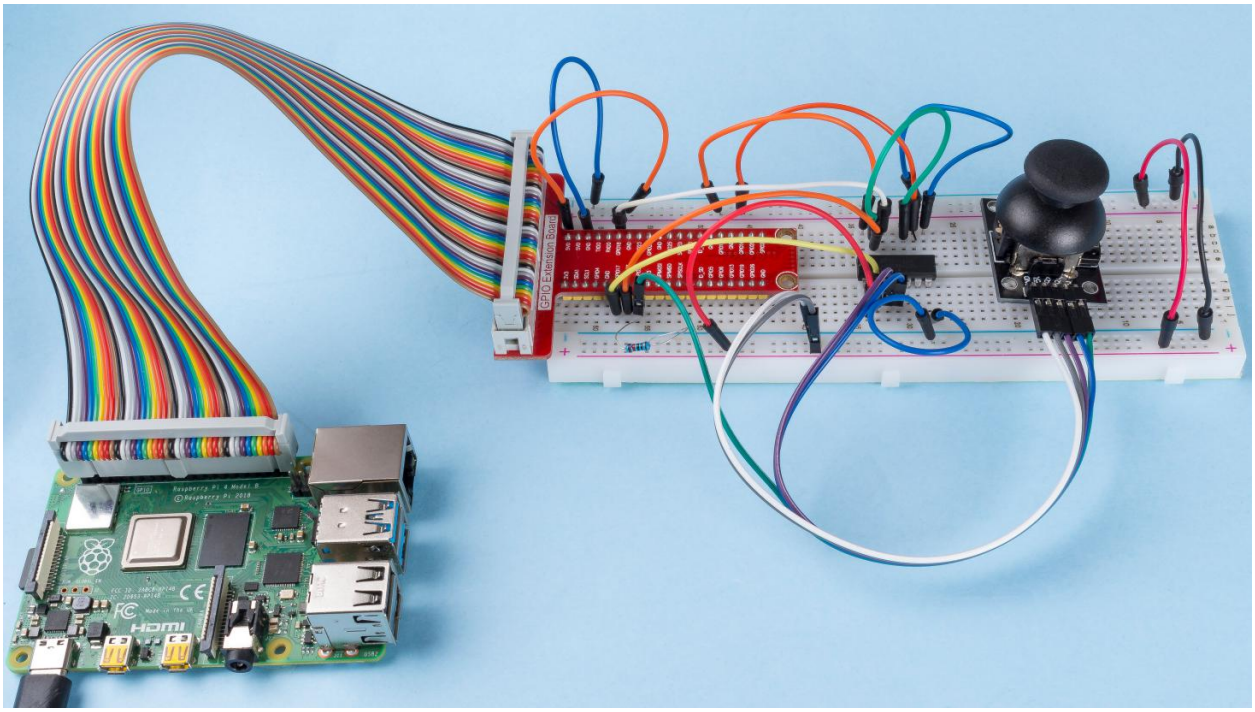
We import an `ADC0834` constructor to use the adc0834 module.

```
setInterval(async() => {

    x_val = await adc.read(0);
    y_val = await adc.read(1);

    btn_val = btn.digitalRead();
    console.log(`x = ${x_val}, y = ${y_val}, btn = ${btn_val}\n`);
}, 100);
```

When reading the values of multiple channels of ADC0834 at the same time, asynchronous programming is required. We build a promise function here, And use the await instruction of async function to elegantly write this complex asynchronous task.

- Promise
- Async Function

Phenomenon Picture



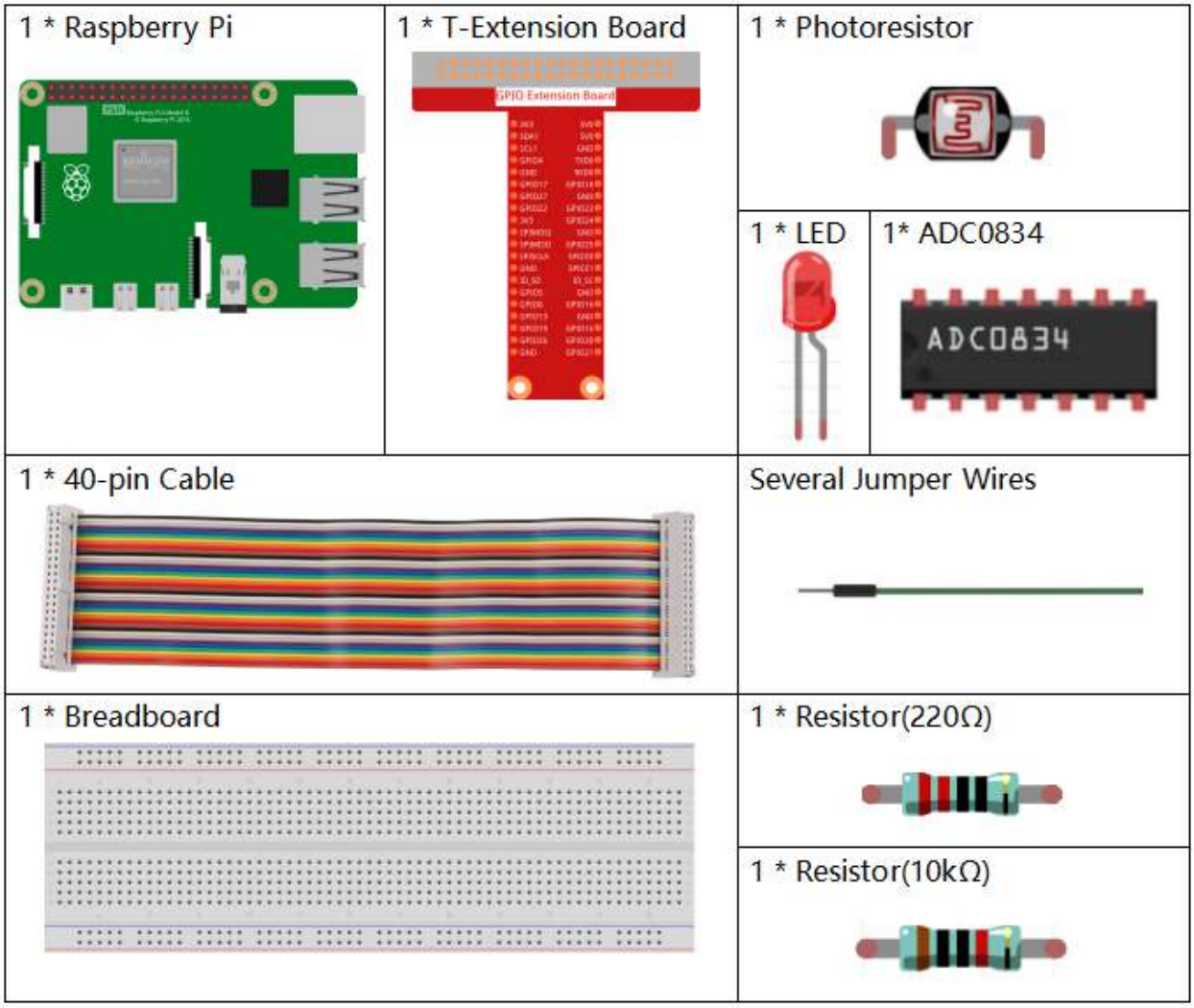
9.5.2 2.2 Sensors

2.2.1 Photoresistor

Introduction

Photoresistor is a commonly used component of ambient light intensity in life. It helps the controller to recognize day and night and realize light control functions such as night lamp. This project is very similar to potentiometer, and you might think it changing the voltage to sensing light.

Components

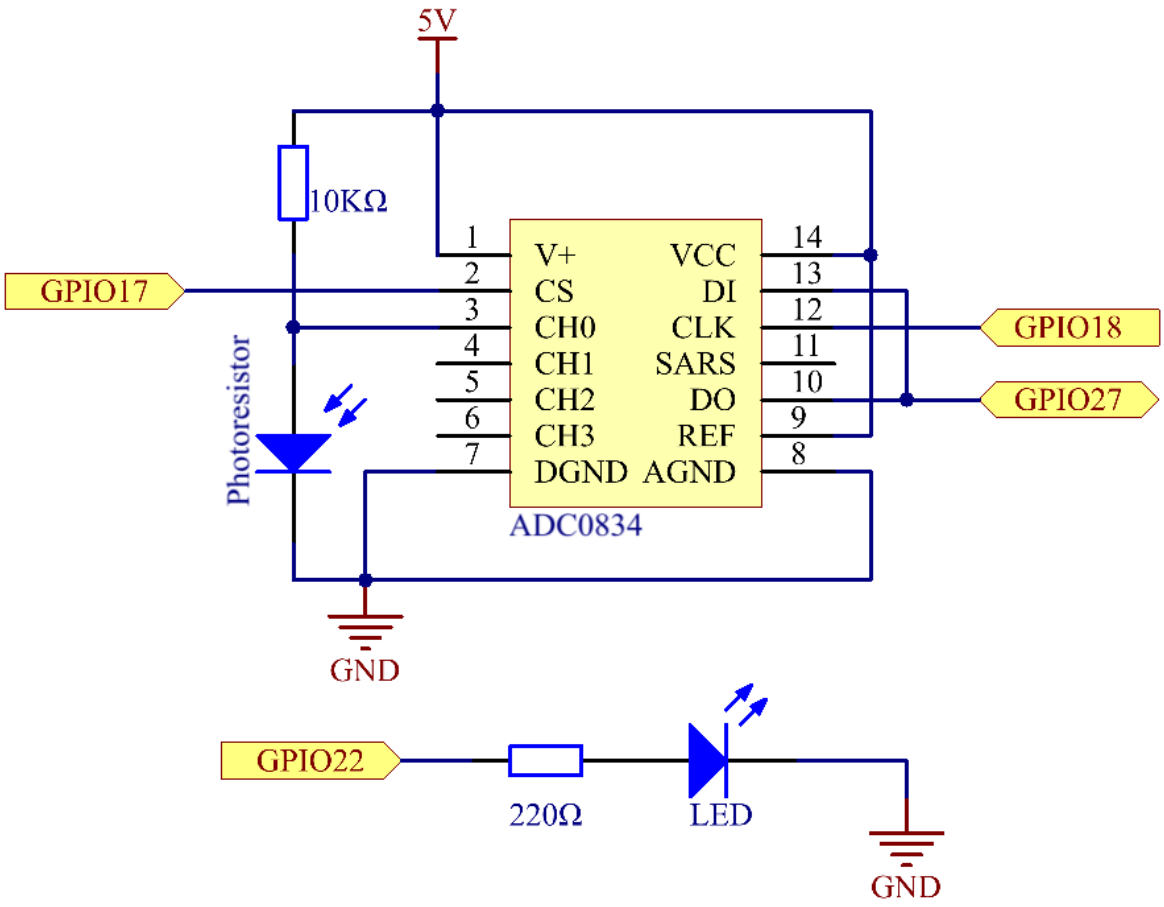


- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*

- *ADC0834*
- *Photoresistor*

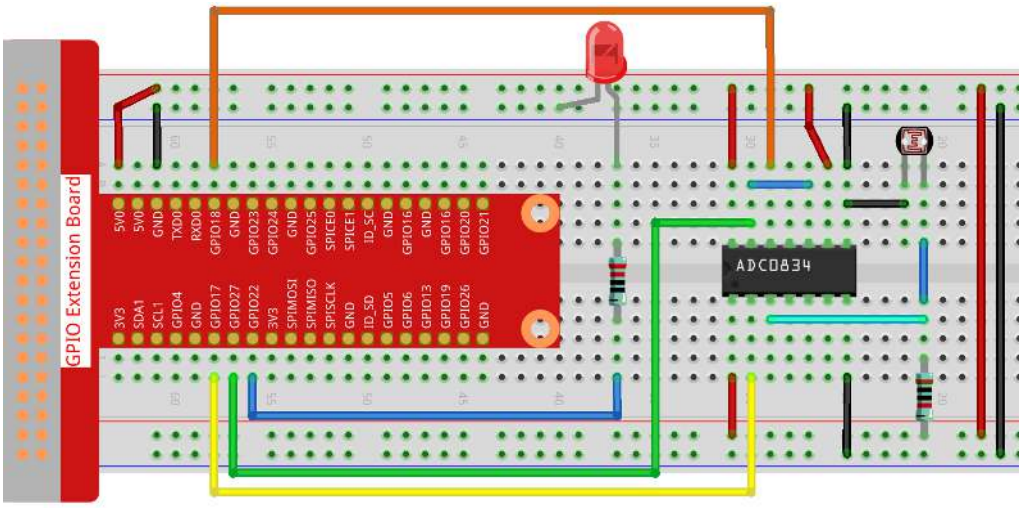
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node photoresistor.js
```

When the code is running, the brightness of the LED will change according to the light intensity sensed by the photoresistor.

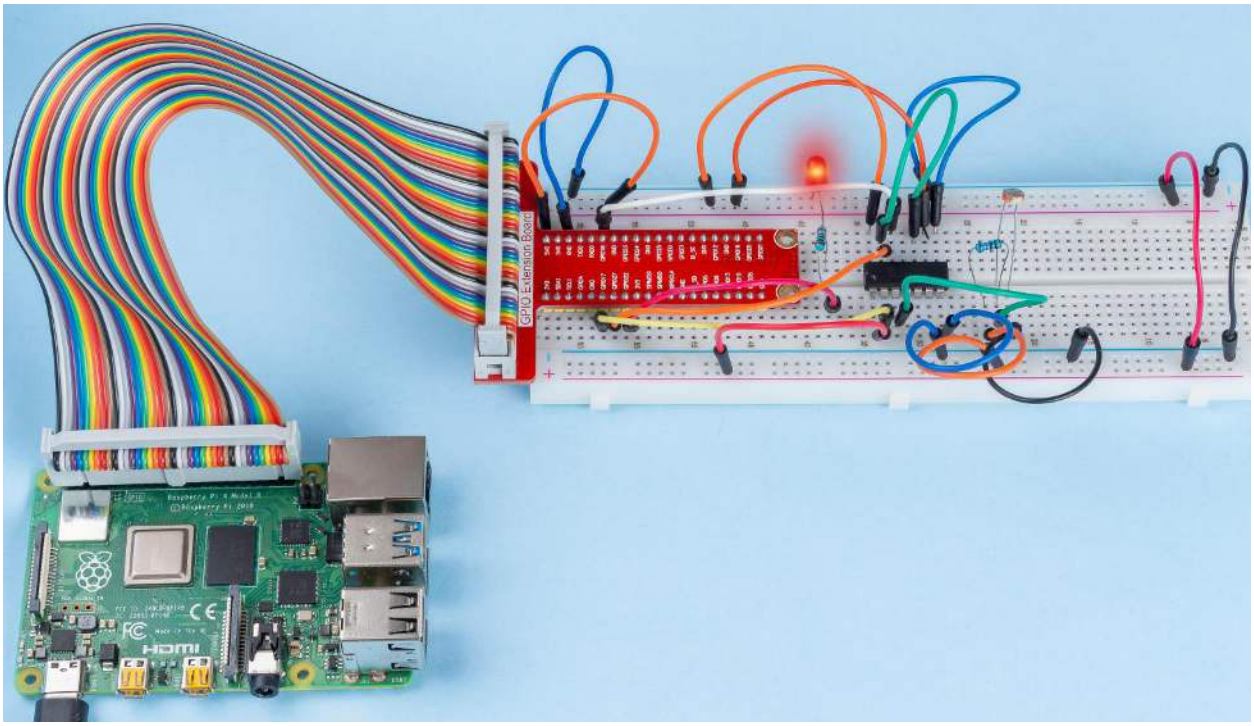
Code

```
const Gpio = require('pigpio').Gpio;
const ADC0834 = require('./adc0834.js').ADC0834;

exports.ADC0834 = ADC0834;

const adc = new ADC0834(17, 18, 27);

const led = new Gpio(22, {mode: Gpio.OUTPUT});

setInterval(() => {
  adc.read(0).then((value) => {
    console.log(value);
    led.pwmWrite(value);
  }, (error)=>{
    console.log("Error: " + error);
  });
}, 100);
```

Code Explanation

The codes here are the same as that in 2.1.7 Potentiometer. Please check the code explanation of *2.1.7 Potentiometer* for details.

Phenomenon Picture

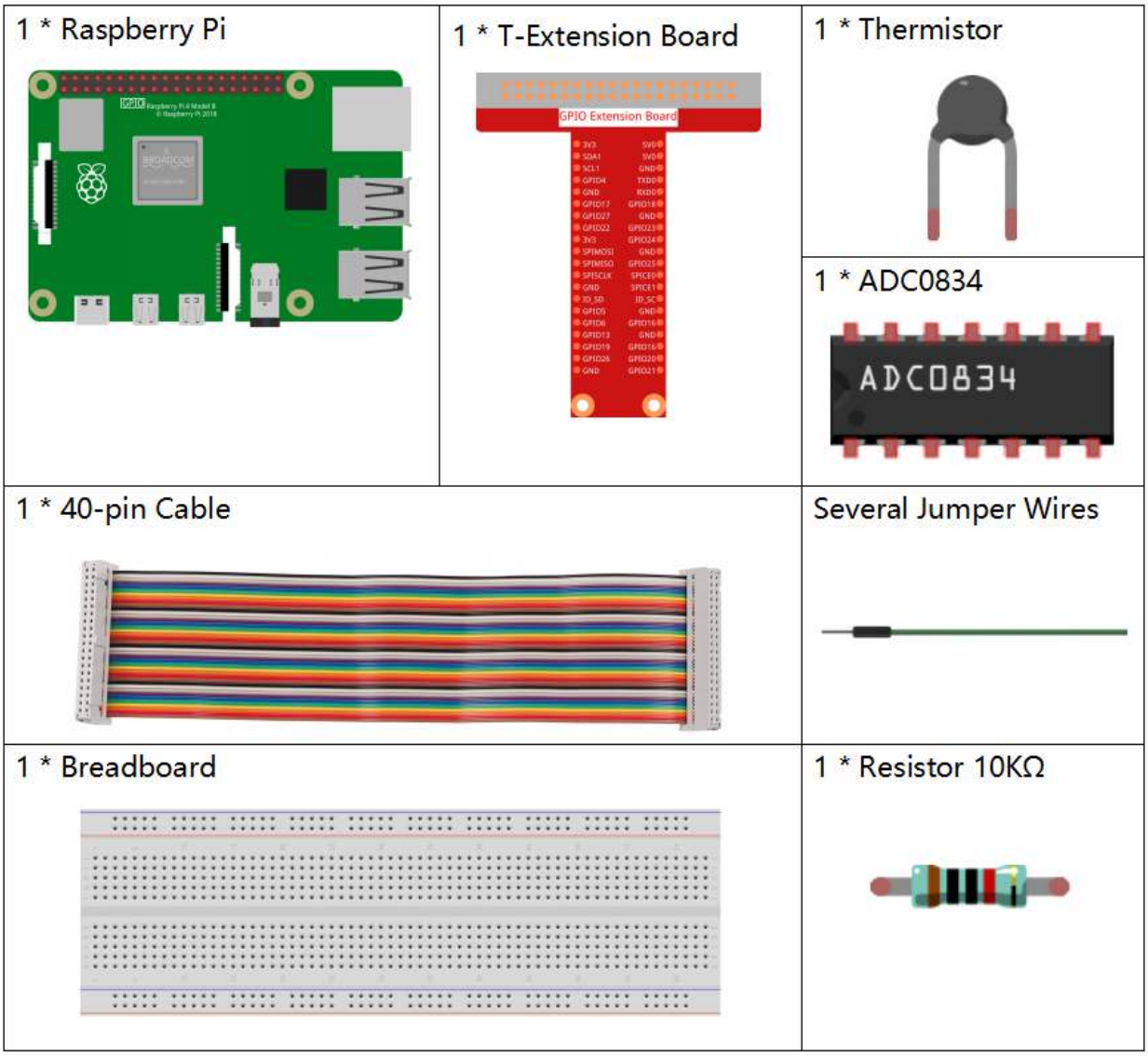


2.2.2 Thermistor

Introduction

Just like photoresistor can sense light, thermistor is a temperature sensitive electronic device that can be used for realizing functions of temperature control, such as making a heat alarm.

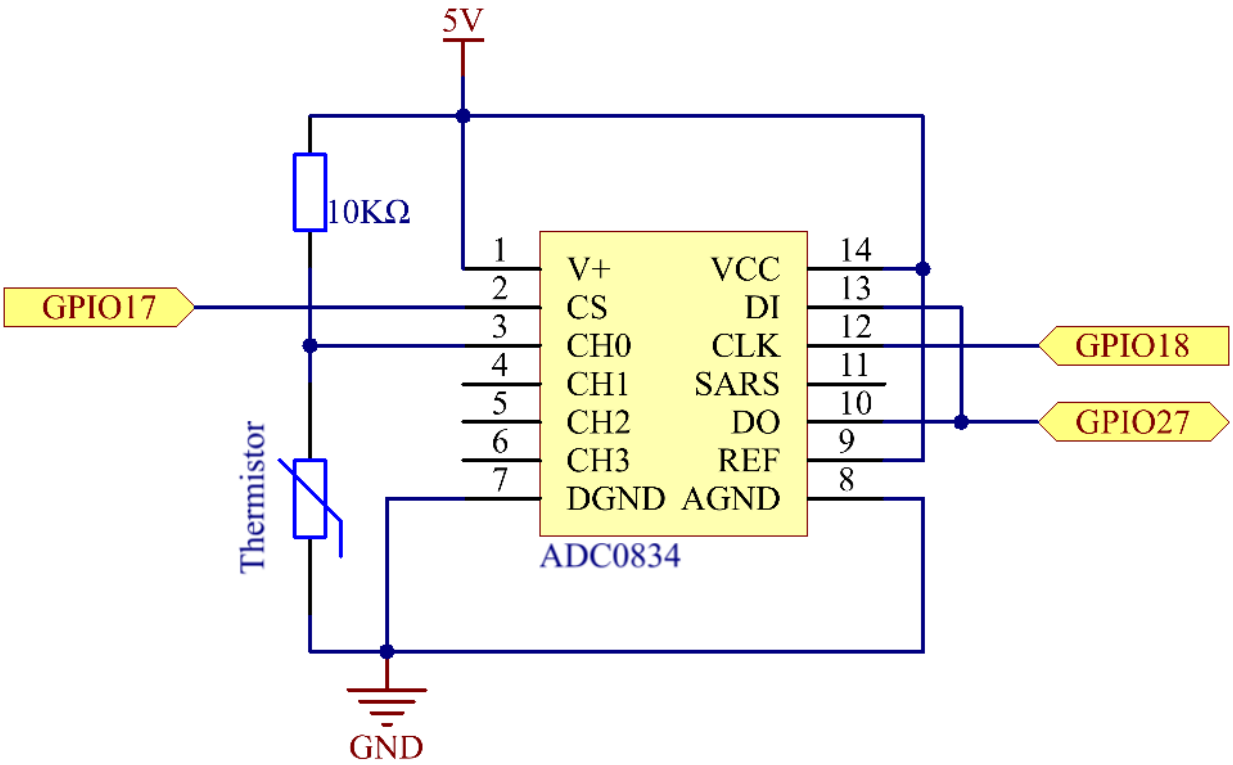
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Thermistor*
- *ADC0834*

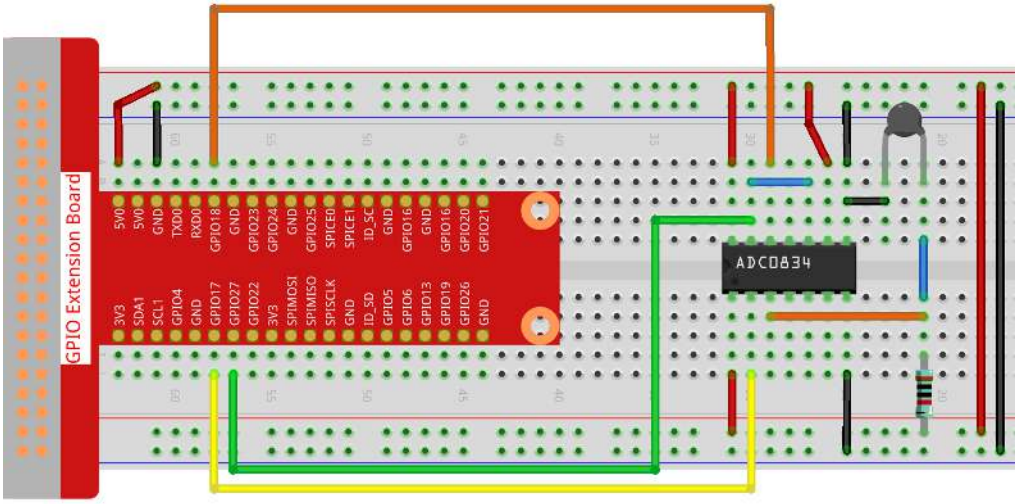
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node thermistor.js
```

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

Code

```
const Gpio = require('pigpio').Gpio;
const ADC0834 = require('./adc0834.js').ADC0834;

exports.ADC0834 = ADC0834;

const adc = new ADC0834(17, 18, 27);

setInterval(() => {
  adc.read(0).then((value) => {
    var Vr = 5 * value / 255;
    var Rt = 10000 * Vr / (5 - Vr);
    var temp = 1 / ((Math.log(Rt/10000) / 3950)+(1 / (273.15 + 25)));
    var cel = (temp - 273.15).toFixed(2);
    var Fah = (cel * 1.8 + 32).toFixed(2);
    console.log(`Celsius: ${cel} C  Fahrenheit: ${Fah} F\n`);
  }, (error)=>{
    console.log("Error: " + error);
  });
}, 1000);
```

Code Explanation

```
setInterval(() => {
  adc.read(0).then((value) => {
    var Vr = 5 * value / 255;
    var Rt = 10000 * Vr / (5 - Vr);
    var temp = 1 / ((Math.log(Rt/10000) / 3950)+(1 / (273.15 + 25)));
    var cel = (temp - 273.15).toFixed(2);
    var Fah = (cel * 1.8 + 32).toFixed(2);
    console.log(`Celsius: ${cel} C  Fahrenheit: ${Fah} F\n`);
  }, (error)=>{
    console.log("Error: " + error);
  });
}, 1000);
```

We can read the `value` of the thermistor through the statement `adc.read(0).then((value) => {...})`

```
var Vr = 5 * value / 255;
var Rt = 10000 * Vr / (5 - Vr);
var temp = 1 / ((Math.log(Rt/10000) / 3950)+(1 / (273.15 + 25)));
var cel = (temp - 273.15).toFixed(2);
var Fah = (cel * 1.8 + 32).toFixed(2);
console.log(`Celsius: ${cel} C  Fahrenheit: ${Fah} F\n`);
```

These operations convert the thermistor value to a Celsius temperature value.

```
var Vr = 5 * value / 255;
var Rt = 10000 * Vr / (5 - Vr);
```

These two lines of code are used to calculate the voltage distribution from the read values, resulting in Rt (resistance of the thermistor).

```
var temp = 1 / ((Math.log(Rt/10000) / 3950)+(1 / (273.15 + 25)));
```

This code refers to substituting Rt into the formula **TK=1/(ln(RT/RN)/B+1/TN)** to get the temperature in Kelvin.

```
var cel = (temp - 273.15).toFixed(2);
```

This paragraph is to convert the Kelvin temperature to Celsius with two decimal places.

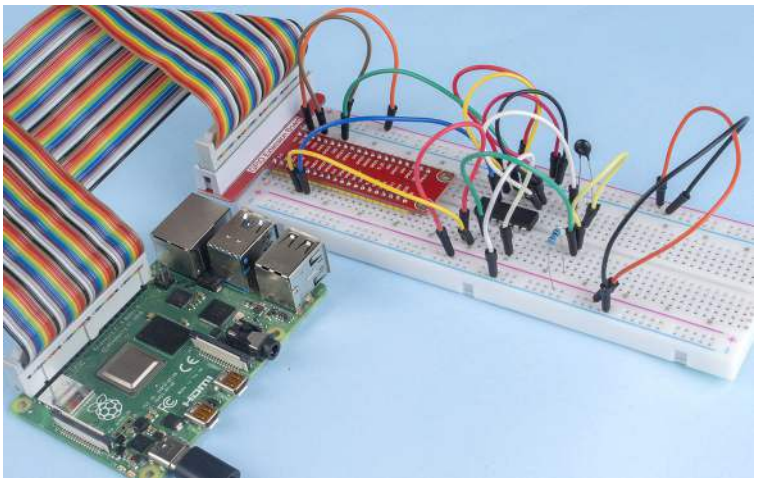
```
var Fah = (cel * 1.8 + 32).toFixed(2);
```

This paragraph converts Celsius to Fahrenheit with two decimal places.

```
console.log(`Celsius: ${cel} C  Fahrenheit: ${Fah} F\n`);
```

Print Celsius, Fahrenheit and their units on the terminal.

Phenomenon Picture



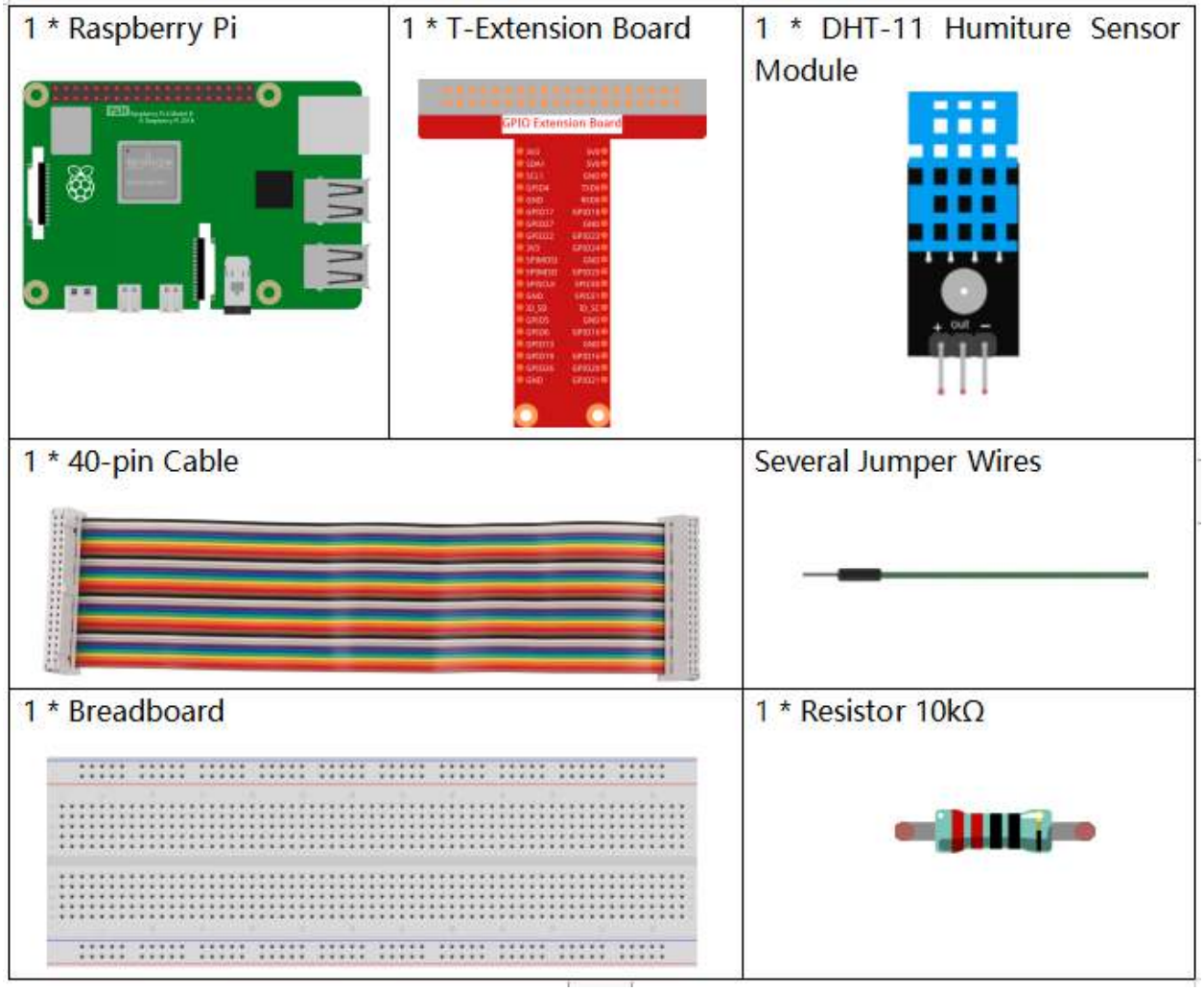
2.2.3 DHT-11

Introduction

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the technology of the temperature and humidity sensing are applied to ensure that the product has high reliability and excellent stability.

The sensors include a wet element resistive sensor and a NTC temperature sensor and they are connected to a high performance 8-bit microcontroller.

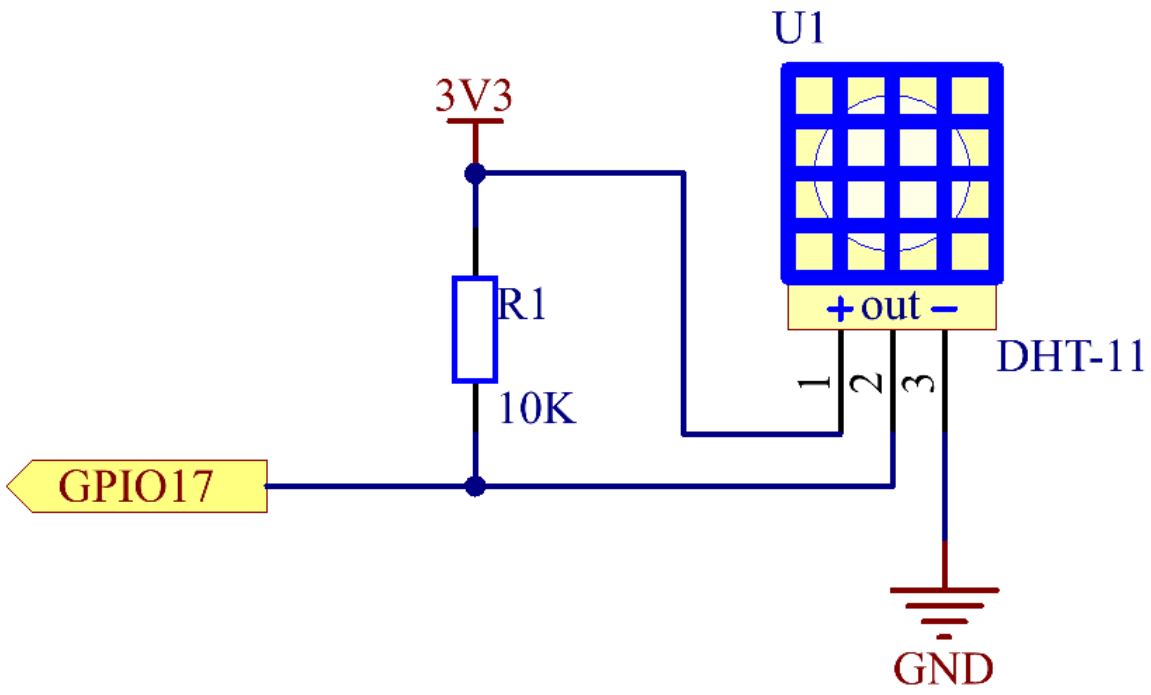
Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Humiture Sensor Module*

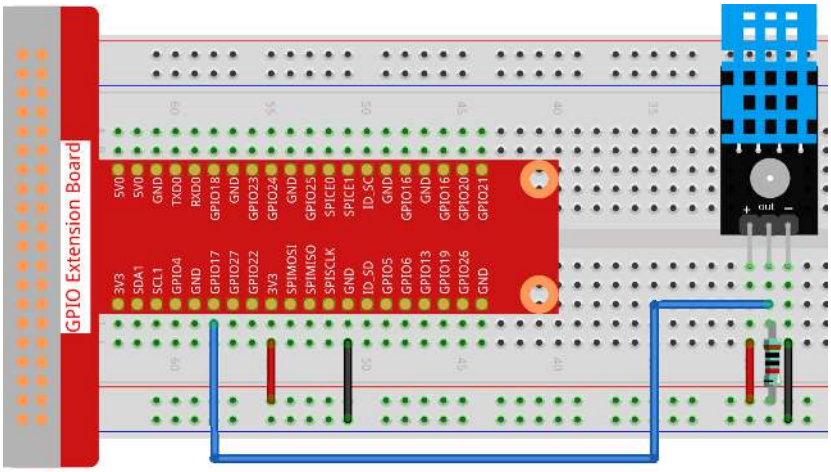
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Install dependencies.

```
sudo npm install node-dht-sensor
```

Step 4: Run the code.

```
sudo node dht11.js
```

After the code runs, the program will print the temperature and humidity detected by DHT11 on the computer screen.

Code

```
var sensor = require("node-dht-sensor");

setInterval(function(){
  sensor.read(11, 17, function(err, temperature, humidity) {
    if (!err) {
      console.log(`temp: ${temperature}\`C, humidity: ${humidity}%`);
    }
  });
},1000);
```

Code Explanation

```
var sensor = require("node-dht-sensor");
```

Import the module `node-dht-sensor` module, which provides functions for us to read the value of DHT-11.

Note: For more details, please refer to: https://www.npmjs.com/package/node-dht-sensor

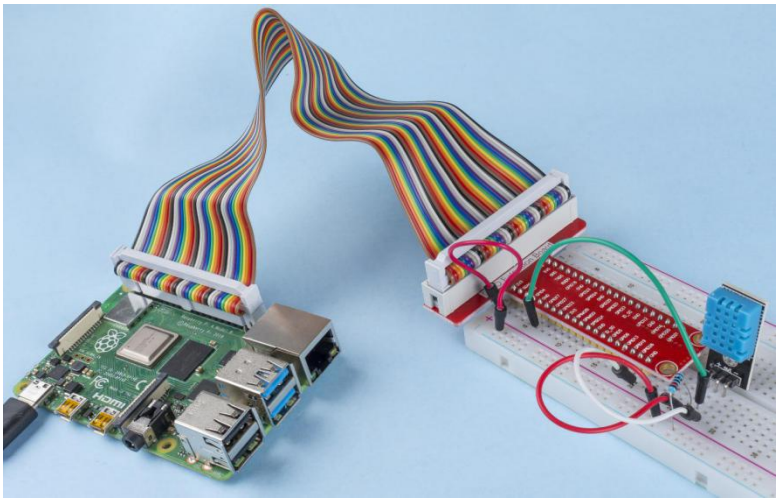
```
sensor.read(11, 17, function(err, temperature, humidity){...})
```

The `node-dht-sensor` module provides the `read()` function for us to read the measured temperature and humidity, where **11** means DHT-11, **17** means and its connected to GPIO17.

```
if (!err) {
  console.log(`temp: ${temperature}\`C, humidity: ${humidity}%`);
}
```

When there is no error in the reading, the temperature and humidity values are printed on the terminal.

Phenomenon Picture

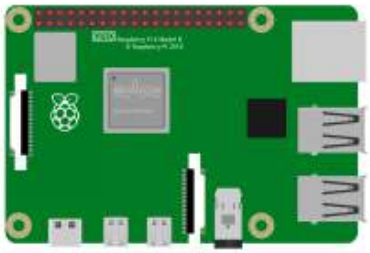
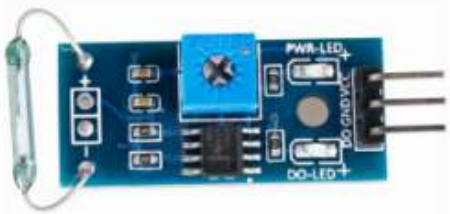
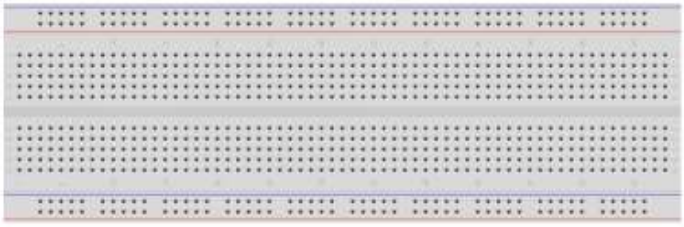
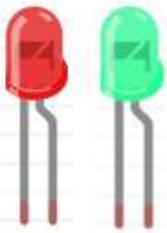
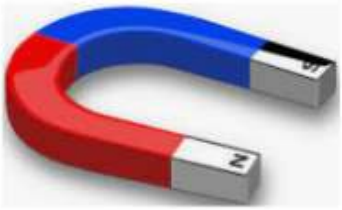


2.2.4 Reed Switch Module

Introduction

In this project, we will learn about the reed switch, which is an electrical switch that operates by means of an applied magnetic field.

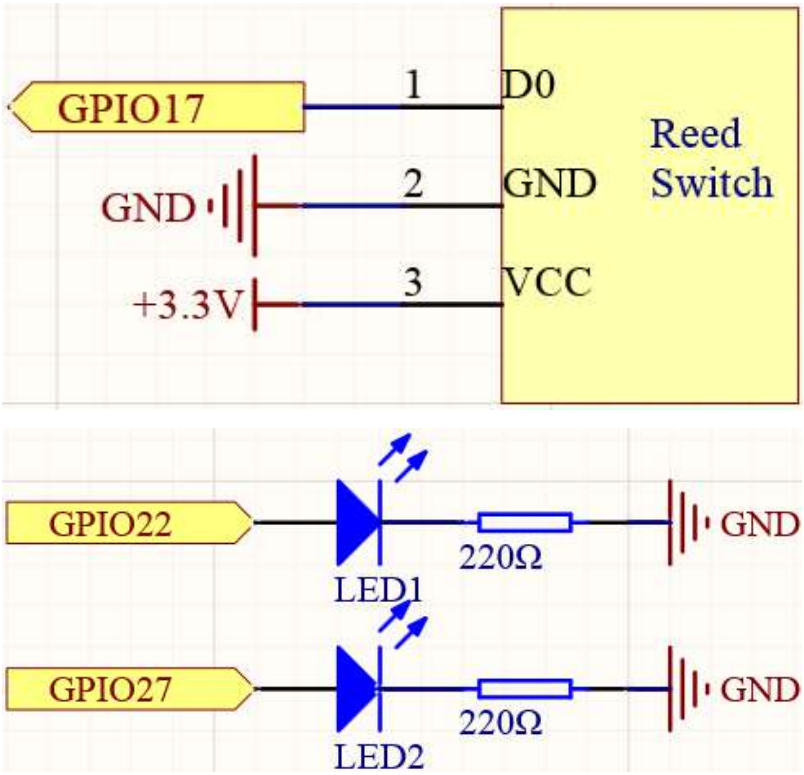
Components

1 * Raspberry Pi 	1 * T-Extension Board	1 * Reed Switch Module 
1 * 40-pin Cable		2 * Resistor(220Ω)
1 * Breadboard 		2 * LED 
1 * Magnet 		Several Jumper Wires

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Reed Switch Module*

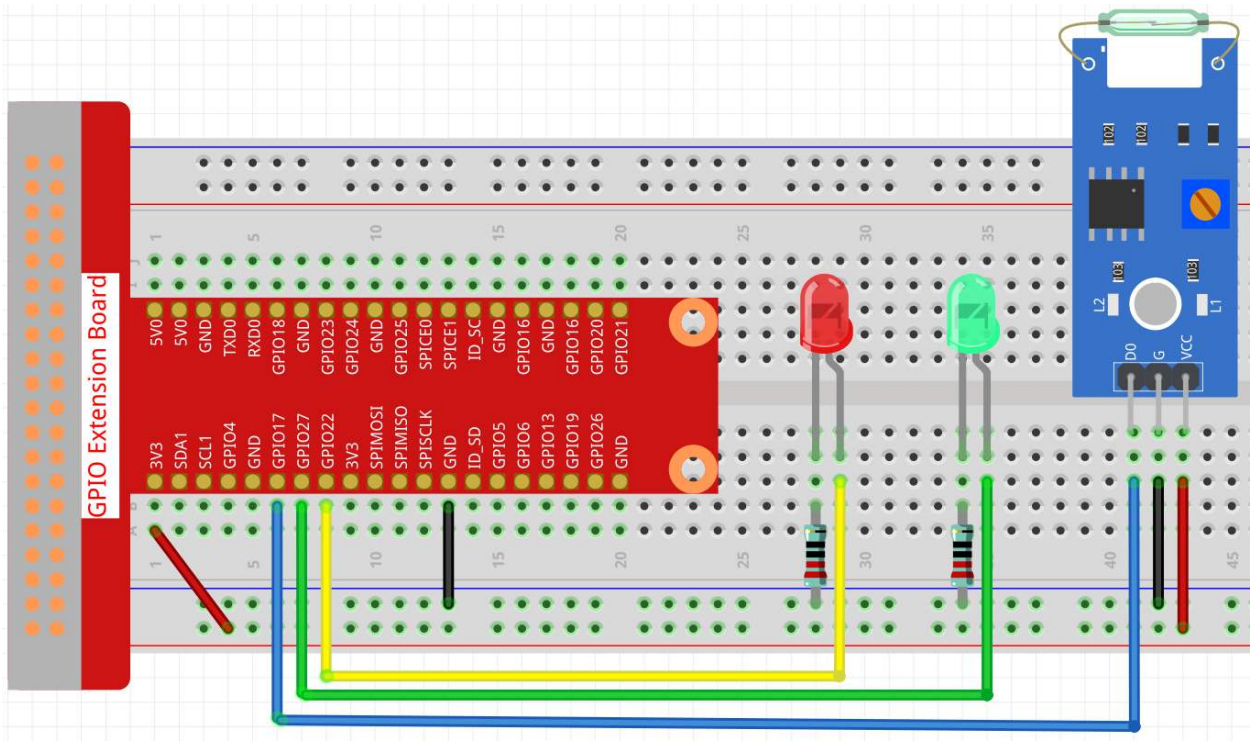
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node reed_switch_module.js
```

The green LED will light up when the code is run. If a magnet is placed close to the reed switch module, the red LED lights up; take away the magnet and the green LED lights up again.

Code

```
const Gpio = require('pigpio').Gpio;

const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});

const reedSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

reedSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

Code Explanation

```
const Gpio = require('pigpio').Gpio;

const reedSwitch = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});
```

Import the pigpio module, create a ReedPin object to control the IO port, set it to input mode, pull down (initially low level), and set an interrupt.

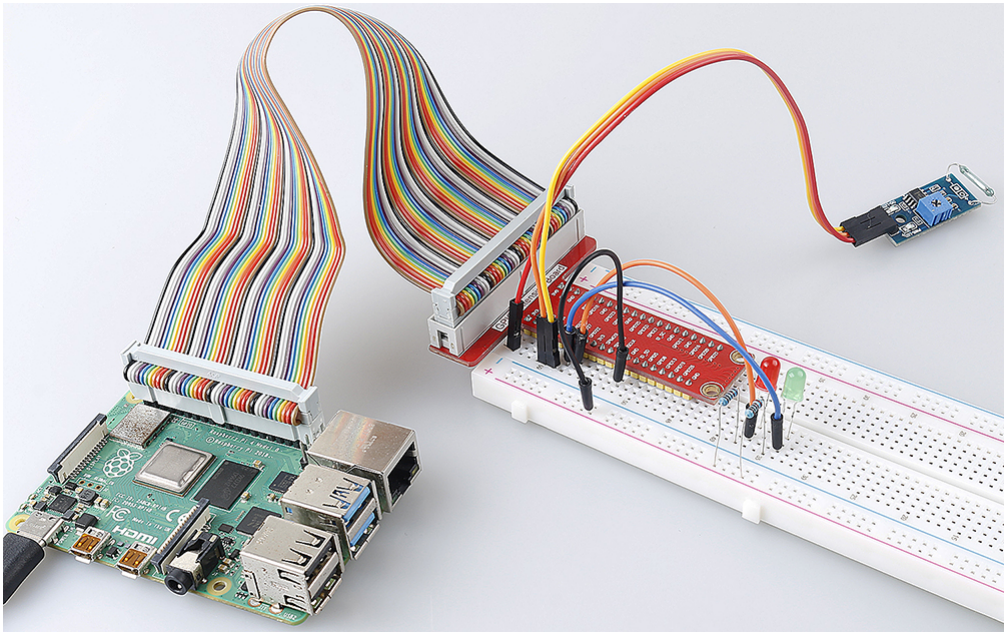
```
const led1 = new Gpio(22, {mode: Gpio.OUTPUT});
const led2 = new Gpio(27, {mode: Gpio.OUTPUT});
```

Create two objects led1, led2 to control the IO ports Gpio22 and Gpio27, and set them to output mode.

```
reedSwitch.on('interrupt', (level) => {
    led1.digitalWrite(level);
    led2.digitalWrite(!level);
});
```

When the interrupt is triggered, write the same level to led1, and write the opposite level to led2.

Phenomenon Picture

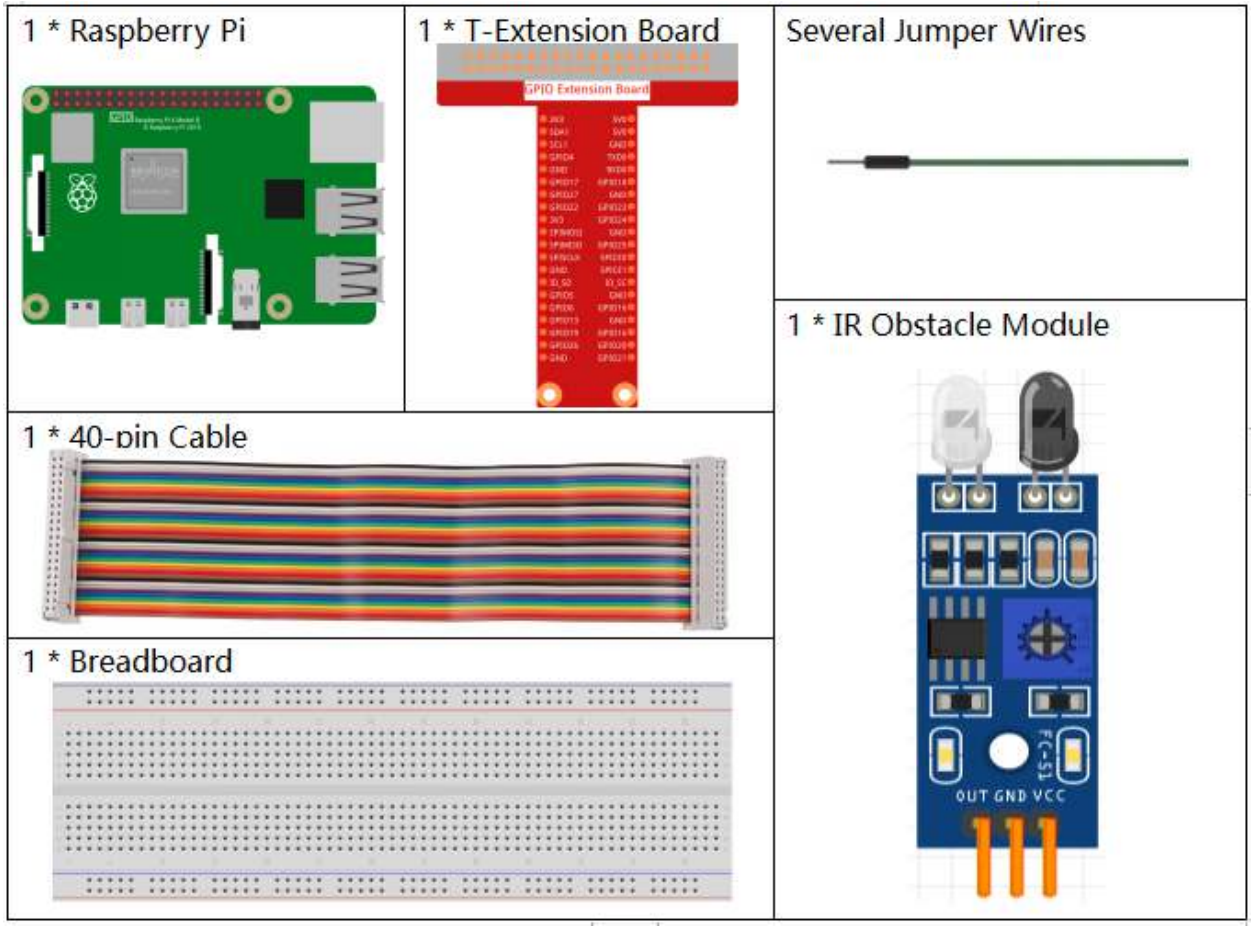


2.2.5 IR Obstacle Avoidance Sensor

Introduction

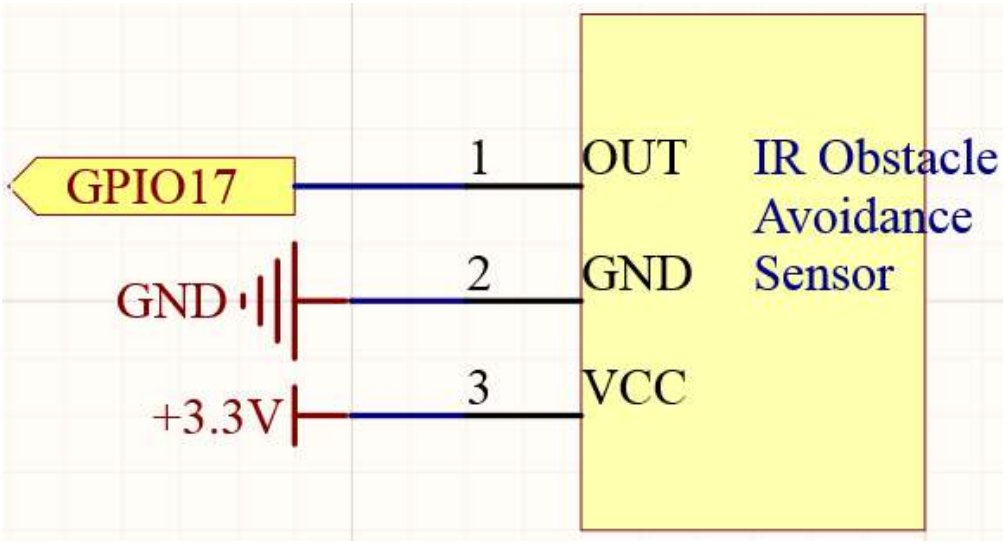
In this project, we will learn IR obstacle avoidance module, which is a sensor module that can be used to detect obstacles at short distances, with small interference, easy to assemble, easy to use, etc. It can be widely used in robot obstacle avoidance, obstacle avoidance trolley, assembly line counting, etc.

Components Required



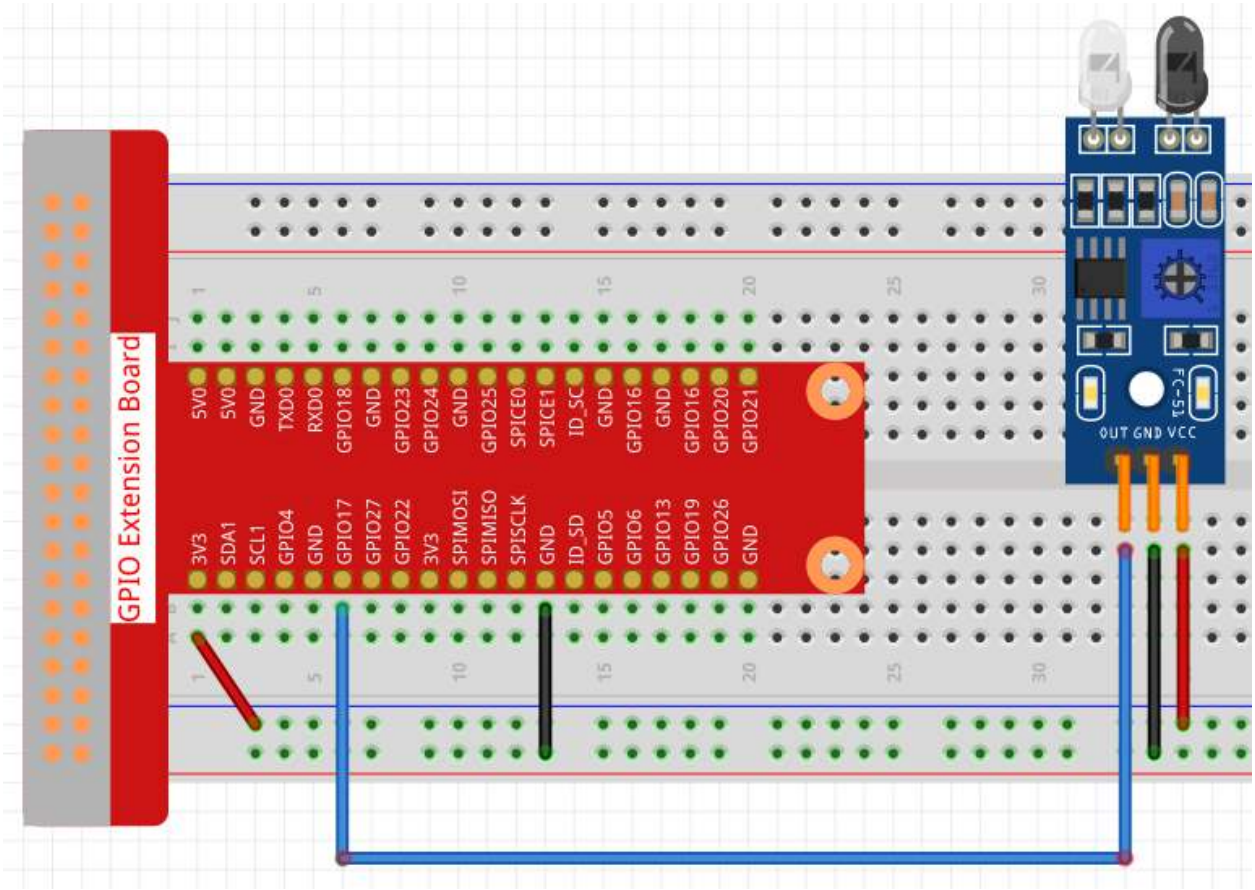
- *GPIO Extension Board*
- *Breadboard*
- *Obstacle Avoidance Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node ir_obstacle.js
```

After the code runs, when you put your hand in front of the module's probe, the output indicator on the module lights up and the "Detected Barrier!" will be repeatedly printed on the screen until the your hand is removed.

Code

```
const Gpio = require('pigpio').Gpio;

const ir_ob = new Gpio(17, {
   mode: Gpio.INPUT,
   pullUpDown: Gpio.PUD_DOWN,
   edge: Gpio.FALLING_EDGE
});

ir_ob.on('interrupt', () => {
   console.log('Detected Barrier!');
});
```

Code Explanation

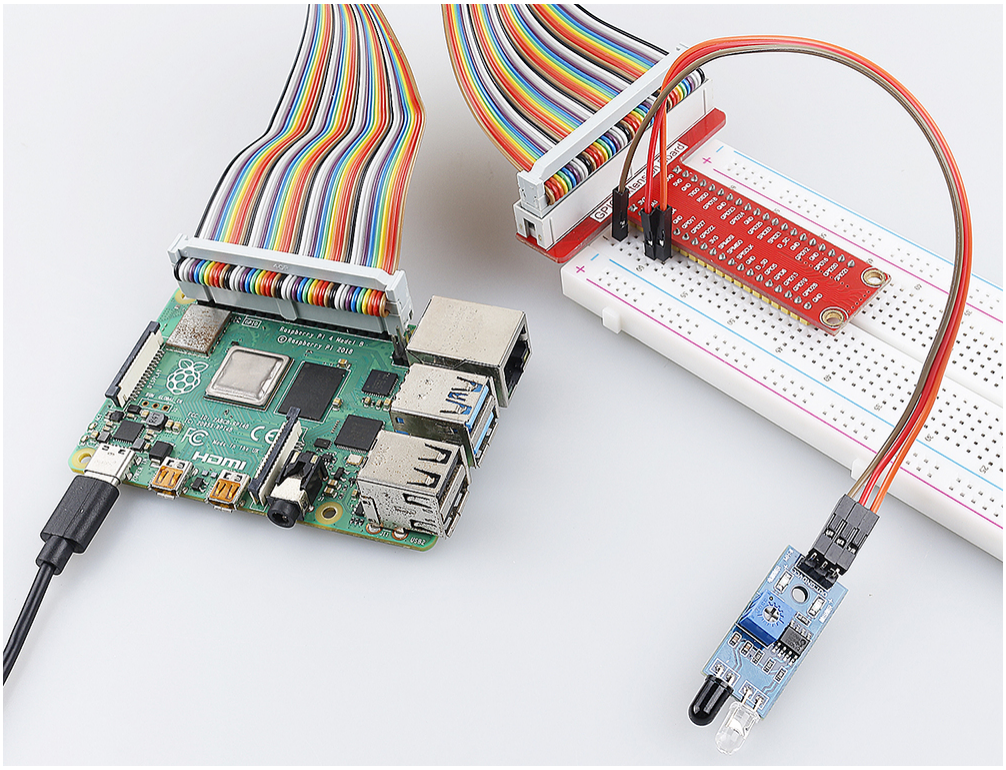
```
const Gpio = require('pigpio').Gpio;

const ir_ob = new Gpio(17, {
   mode: Gpio.INPUT,
   pullUpDown: Gpio.PUD_DOWN,
   edge: Gpio.FALLING_EDGE
});
```

Import the pigpio module, create an object to control the IO port Gpio17, Set it to input mode and interrupt on falling edge.

```
ir_ob.on('interrupt', () => {
   console.log('Detected Barrier!');
});
```

When an interrupt is triggered, meaning an obstacle is detected, print "Detected Barrier!".

Phenomenon Picture

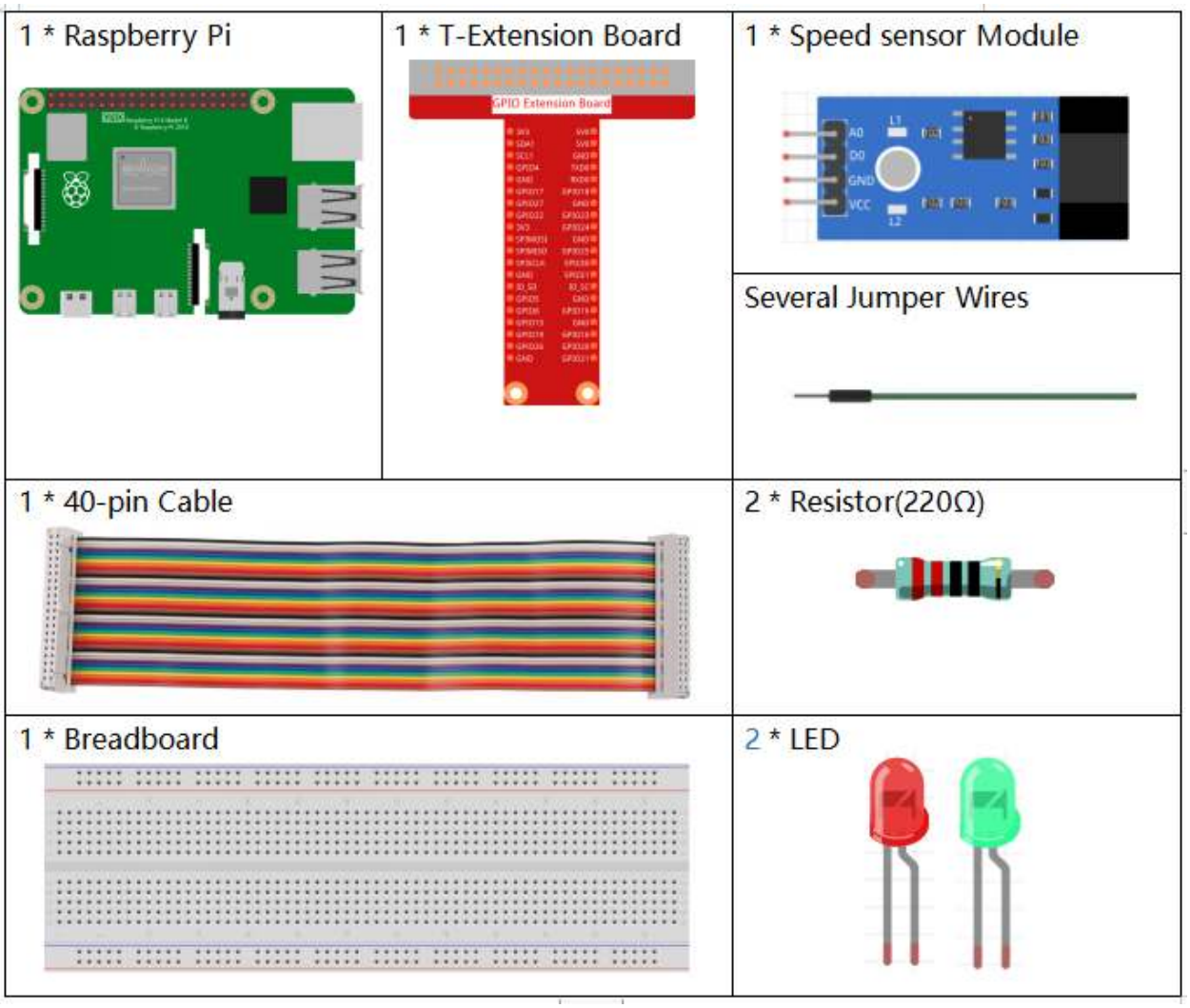


2.2.6 Speed Sensor Module

Introduction

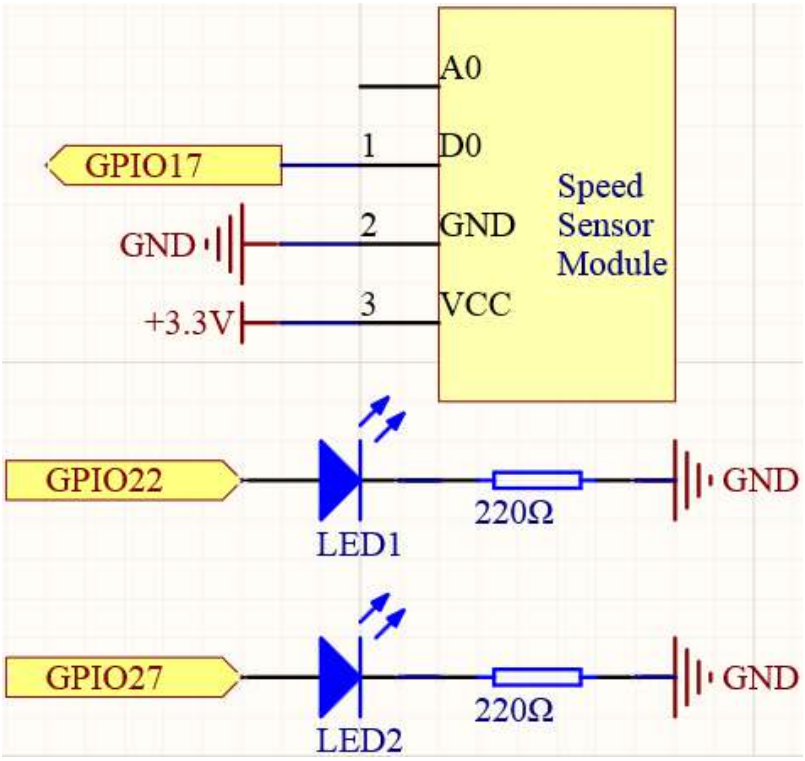
In this project, we will learn the use of the speed sensor module. A Speed sensor module is a type of tachometer that is used to measure the speed of a rotating object like a motor.

Components



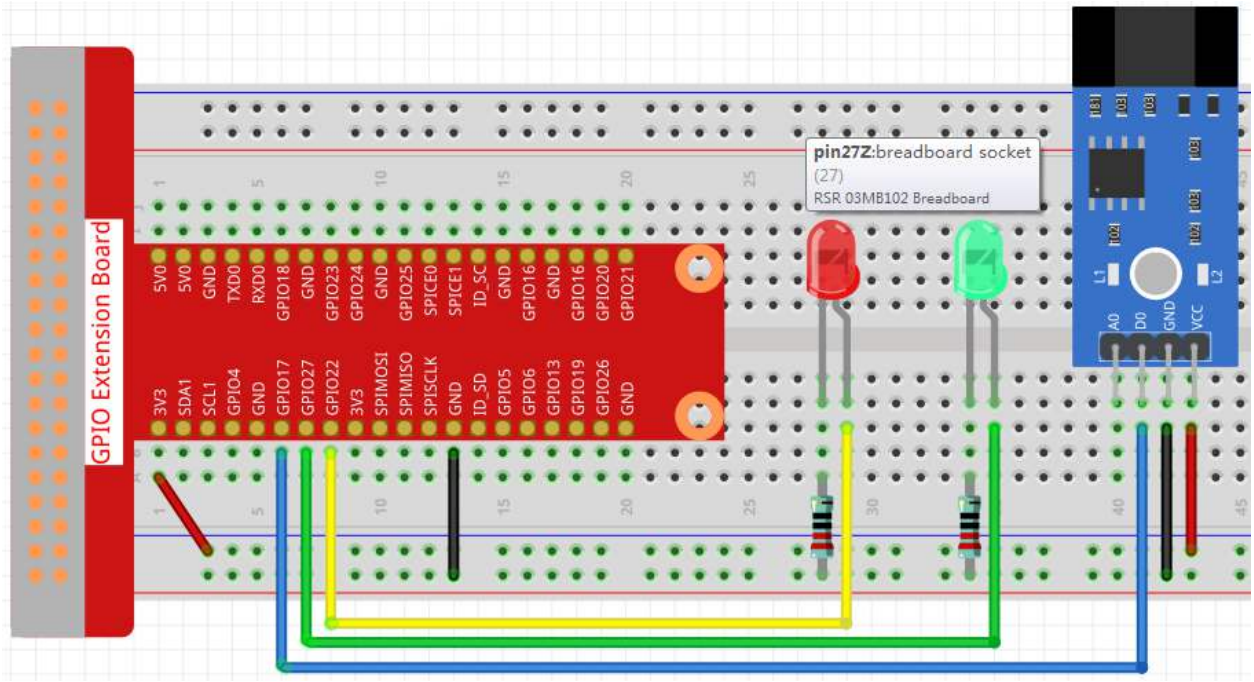
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*
- *Speed Sensor Module*

Schematic Diagram



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node speed_sensor_module.js
```

After the code runs, the green LED will light up. If you place an obstacle in the gap of the speed sensor module, the "light blocked" will be printed on the screen and the red LED will be lit. Remove the obstacle and the green LED will light up again.

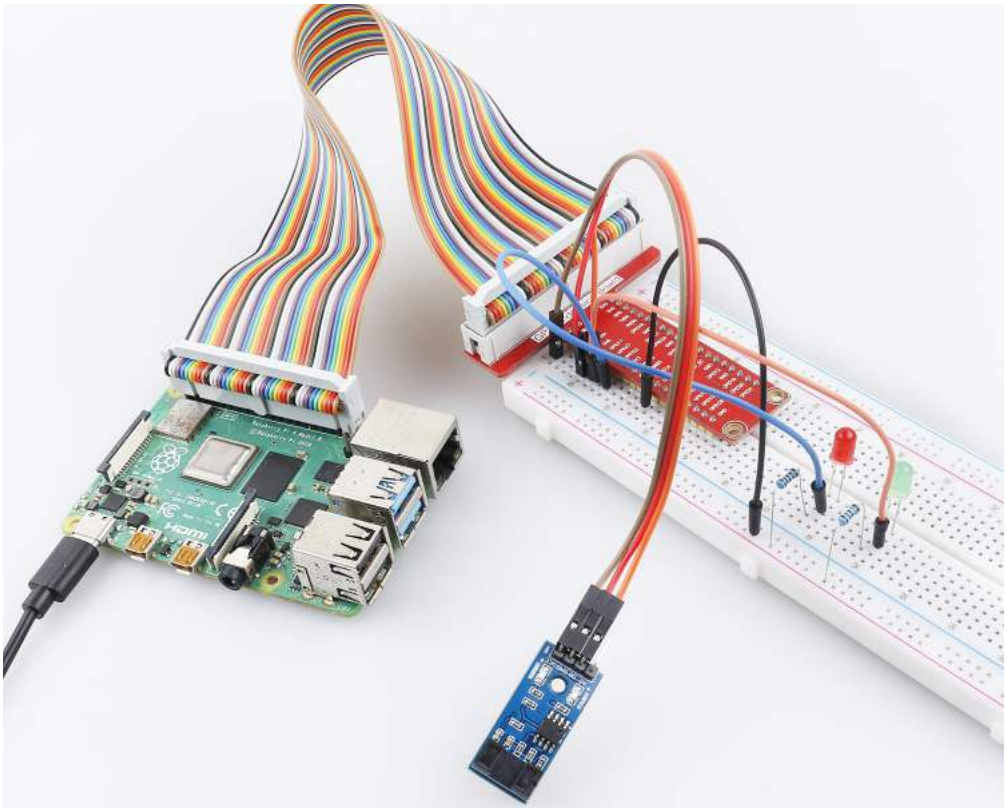
Code

```
const Gpio = require('pigpio').Gpio;

const Rpin = new Gpio(22, { mode: Gpio.OUTPUT });
const Gpin = new Gpio(27, { mode: Gpio.OUTPUT });

const speedPin = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
});

speedPin.on('interrupt', (level) => {
    if (level) {
        console.log("Light was blocked");
    }
    Rpin.digitalWrite(level);
    Gpin.digitalWrite(!level);
});

process.on('SIGINT', function () {
    Rpin.digitalWrite(0);
    Gpin.digitalWrite(0);
    process.exit();
});
```

Code Explanation

The code of this example is almost the same as *2.1.5 Tilt Switch*, so no need to repeat it.

Phenomenon Picture

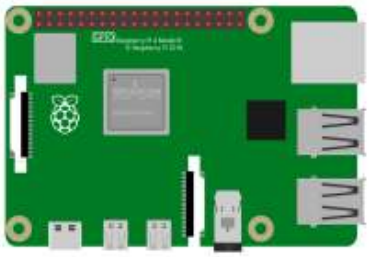
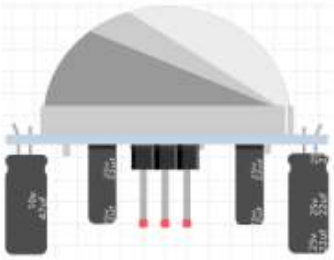
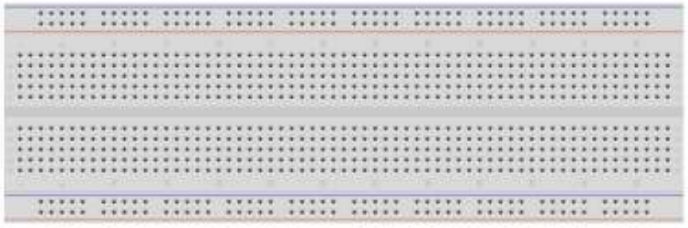


2.2.7 PIR

Introduction

In this project, we will make a device by using the human body infrared pyroelectric sensors. When someone gets closer to the LED, the LED will turn on automatically. If not, the light will turn off. This infrared motion sensor is a kind of sensor that can detect the infrared emitted by human and animals.

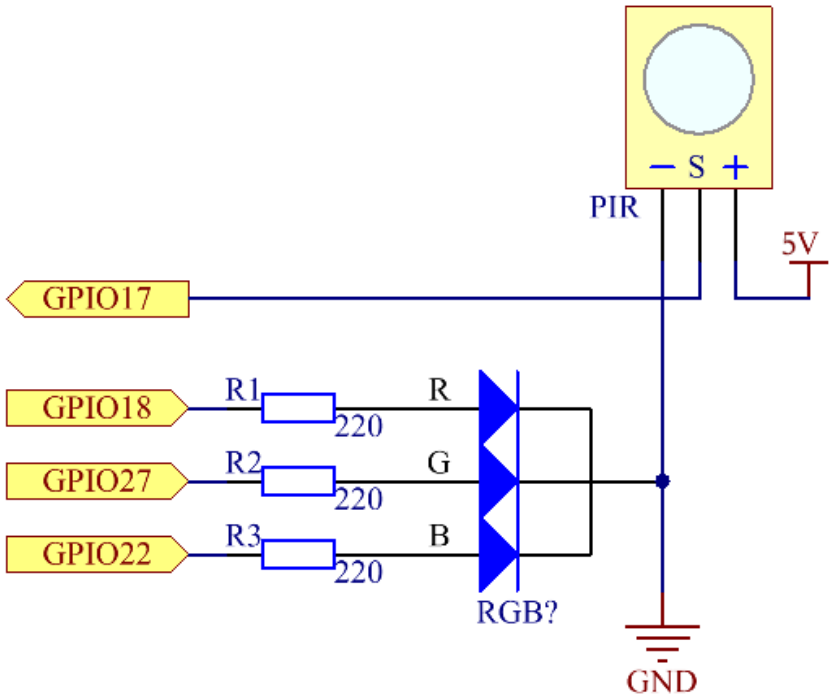
Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*
- *PIR Motion Sensor Module*

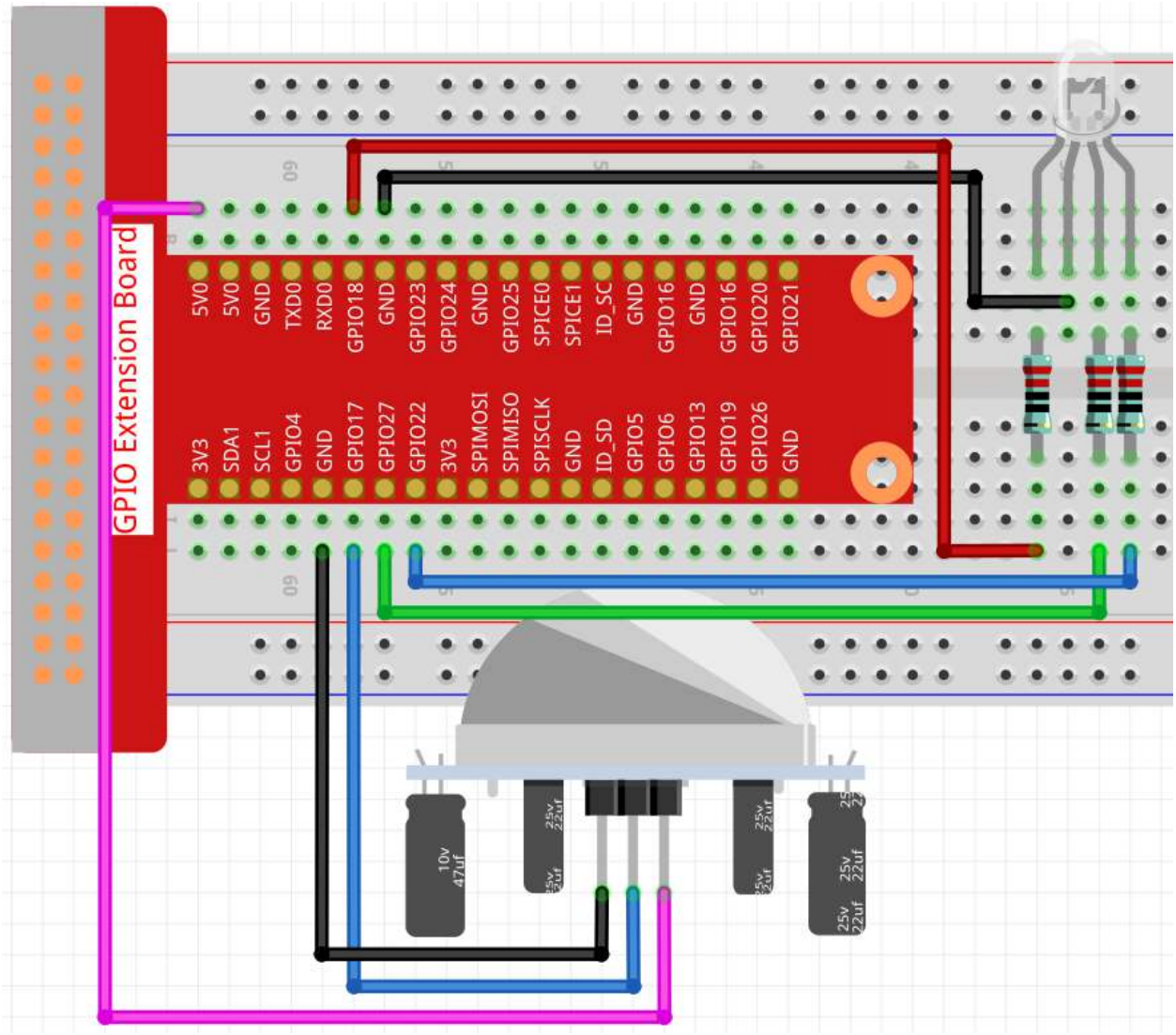
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	3	22



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node pir.js
```

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. In order to make the PIR module work better, you need to try to adjust these two potentiometers.

Code

```
const Gpio = require('pigpio').Gpio;
```

(continues on next page)

(continued from previous page)

```
const pirPin = new Gpio(17, {
    mode: Gpio.INPUT,
    pullUpDown: Gpio.PUD_DOWN,
    edge: Gpio.EITHER_EDGE
})    // the pir connect to pin17

const redPin = new Gpio(18, { mode: Gpio.OUTPUT, })
const greenPin = new Gpio(27, { mode: Gpio.OUTPUT, })
const bluePin = new Gpio(22, { mode: Gpio.OUTPUT, })
//'Red':18, 'Green':27, 'Blue':22

var p_R, p_G, p_B

// Set all led as pwm channel and frequece to 2KHz
p_R = redPin.pwmFrequency(2000)
p_G = greenPin.pwmFrequency(2000)
p_B = bluePin.pwmFrequency(2000)

// Set all begin with value 0
p_R.pwmWrite(0)
p_G.pwmWrite(0)
p_B.pwmWrite(0)

// Define a MAP function for mapping values.  Like from 0~255 to 0~100
function 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
function 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.pwmWrite(R_val)
    p_G.pwmWrite(G_val)
    p_B.pwmWrite(B_val)
    //print ("color_msg: R_val = %s, G_val = %s, B_val = %s"%(R_val, G_val, B_val))
}

pirPin.on('interrupt', (level) => {
    if (level) {
        setColor(0xFFFF00)
    }else{
        setColor(0x0000FF)
    }
});

process.on('SIGINT', function () {
```

(continues on next page)

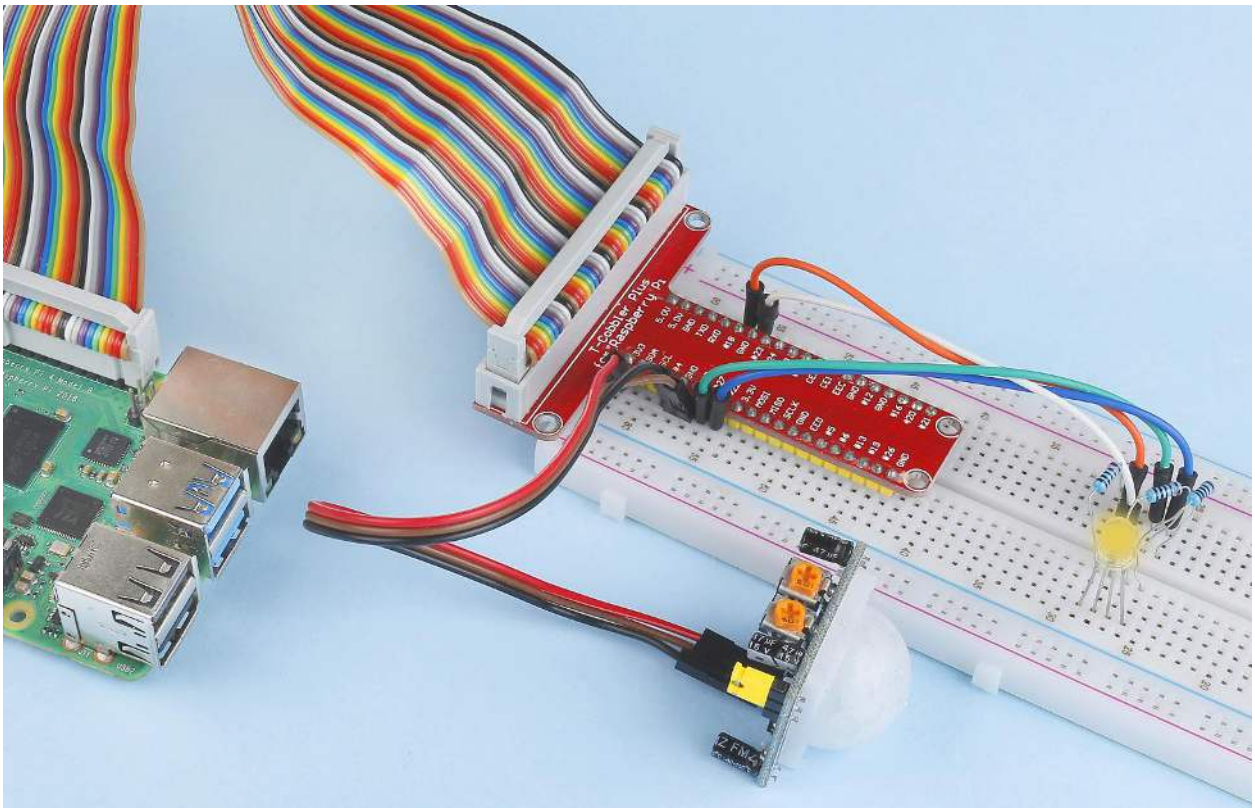
(continued from previous page)

```
    p_R.pwmWrite(0)
    p_G.pwmWrite(0)
    p_B.pwmWrite(0)
    process.exit();
})
```

Code Explanation

The code for this example is a combination of *2.1.1 Button* and *1.1.2 RGB LED*, no need to go into details.

Phenomenon Picture

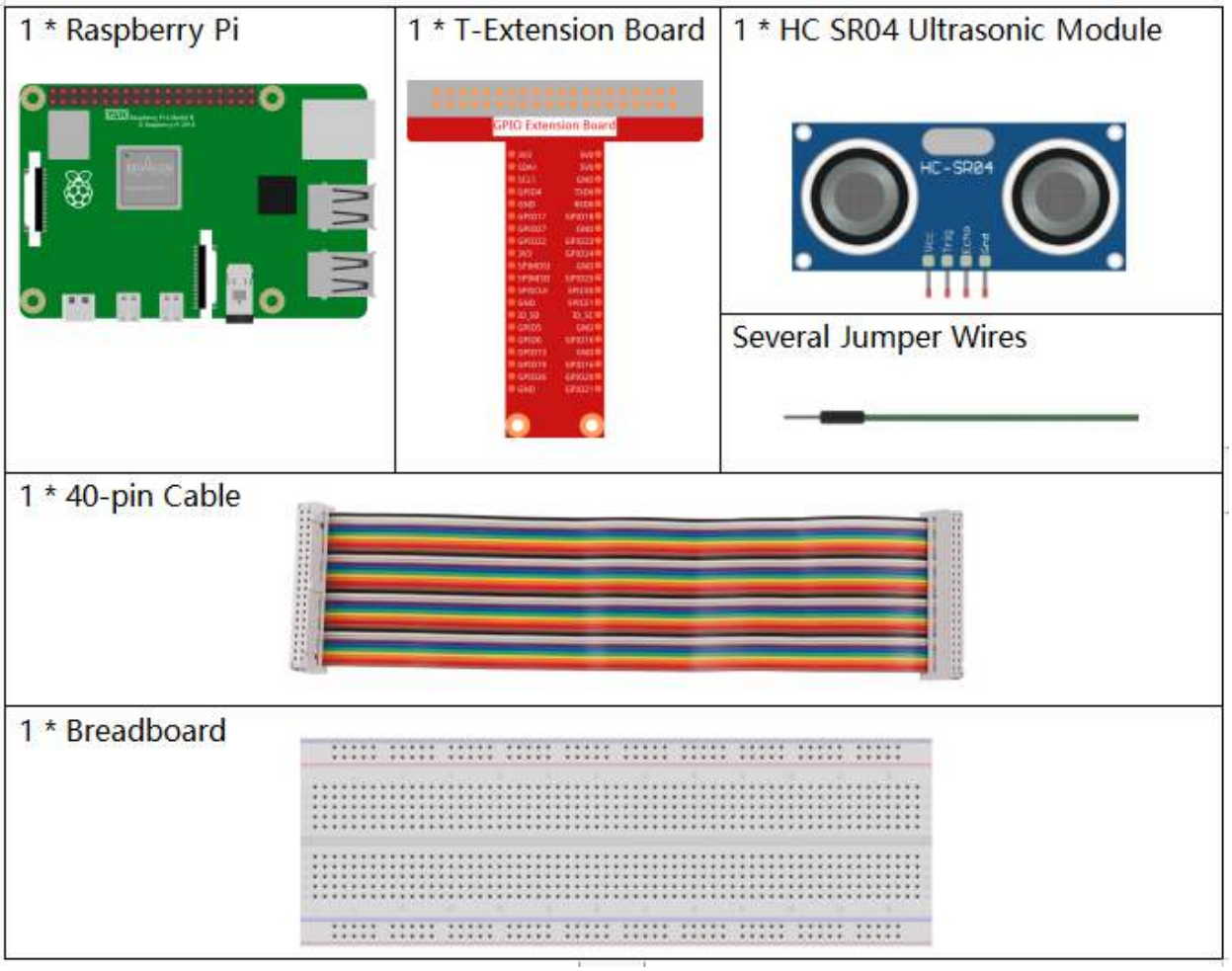


2.2.8 Ultrasonic Sensor Module

Introduction

The ultrasonic sensor uses ultrasonic to accurately detect objects and measure distances. It sends out ultrasonic waves and converts them into electronic signals.

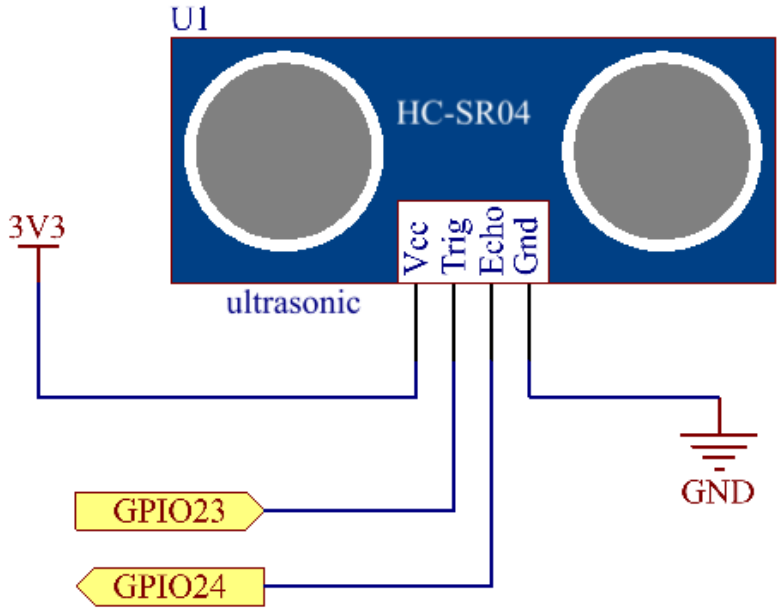
Components



- *GPIO Extension Board*
- *Breadboard*
- *Ultrasonic Module*

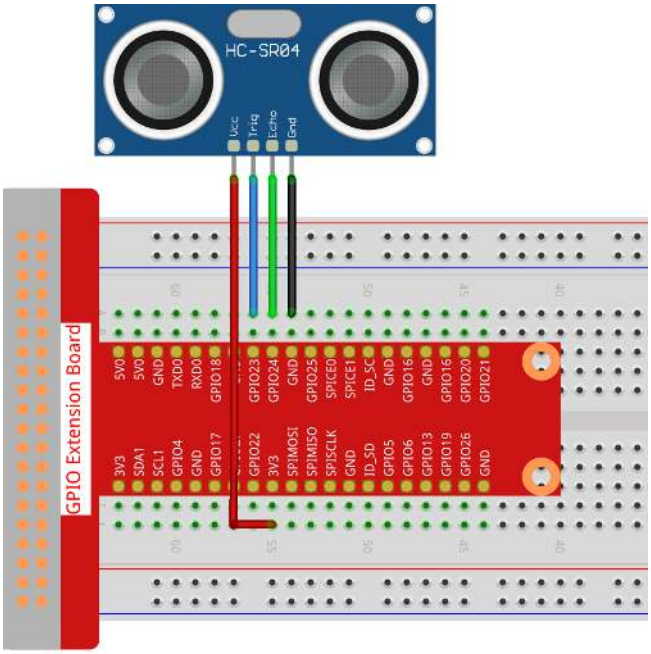
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



Experimental Procedures

Step 1: Build the circuit.



Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node ultrasonic_sensor.js
```

With the code run, the ultrasonic sensor module detects the distance between the obstacle ahead and the module itself, then the distance value will be printed on the screen.

Code

```
const Gpio = require('pigpio').Gpio;

// The number of microseconds it takes sound to travel 1cm at 20 degrees celcius
const MICROSECDONDS_PER_CM = 1e6/34321;

const trigger = new Gpio(23, {mode: Gpio.OUTPUT});
const echo = new Gpio(24, {mode: Gpio.INPUT, alert: true});

trigger.digitalWrite(0); // Make sure trigger is low

const watchHCSR04 = () => {
  let startTick;

  echo.on('alert', (level, tick) => {
    if (level === 1) {
      startTick = tick;
    } else {
      const endTick = tick;
      const diff = (endTick >> 0) - (startTick >> 0); // Unsigned 32 bit arithmetic
      console.log(diff / 2 / MICROSECDONDS_PER_CM);
    }
  });
```

(continues on next page)

(continued from previous page)

```
};

watchHCSR04();

// Trigger a distance measurement once per second
setInterval(() => {
  trigger.trigger(10, 1); // Set trigger high for 10 microseconds
}, 1000);
```

Code Explanation

The `trigger` function can be used to generate a pulse on a GPIO and `alerts` can be used to determine the time of a GPIO state change accurate to a few microseconds.

These two features can be combined to measure distance using a HC-SR04 ultrasonic sensor.

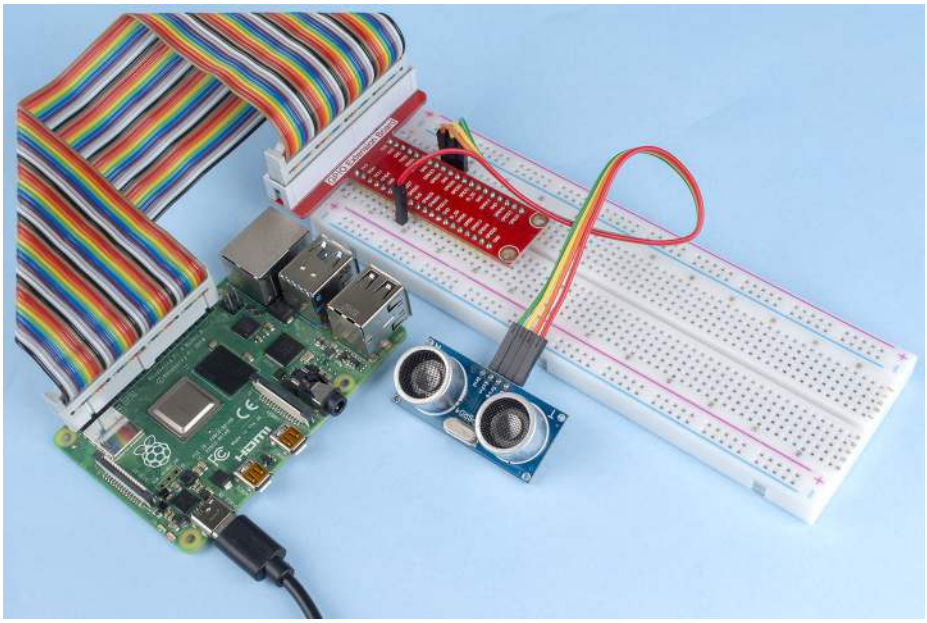
```
setInterval(() => {
  trigger.trigger(10, 1); // Set trigger high for 10 microseconds
}, 1000);
```

This is to periodically send out a 10us ultrasonic pulse.

```
const watchHCSR04 = () => {

  echo.on('alert', (level, tick) => {
      if (level === 1) {
        startTick = tick;
      } else {
        const endTick = tick;
        const diff = (endTick >> 0) - (startTick >> 0); // Unsigned 32 bit arithmetic
        console.log(diff / 2 / MICROSECDONDS_PER_CM);
      }
  });
};
```

This function sets an alert that will record the time between sending the pulse (level is 1) and receiving the echo (level is 0). By multiplying the time difference by the speed of sound (and dividing by 2), you can get the distance to the obstacle ahead.

Phenomenon Picture



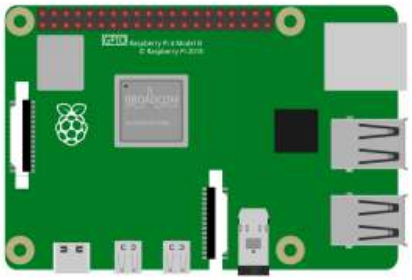
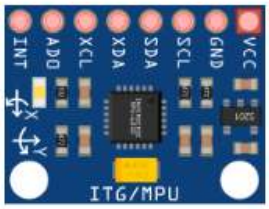
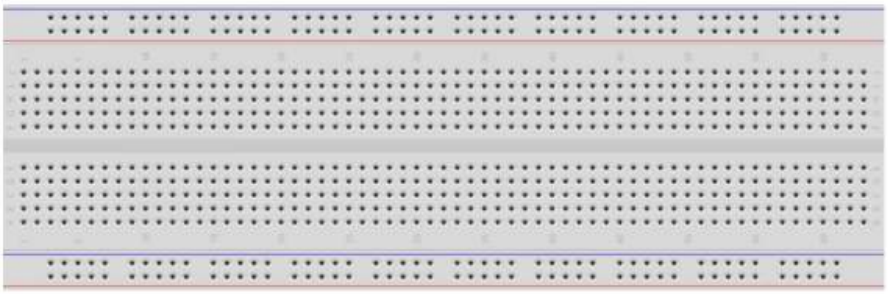
2.2.9 MPU6050 Module

Introduction

The MPU-6050 is the world's first and only 6-axis motion tracking devices (3-axis Gyroscope and 3-axis Accelerometer) designed for smartphones, tablets and wearable sensors that have these features, including the low power, low cost, and high performance requirements.

In this experiment, use I2C to obtain the values of the three-axis acceleration sensor and three-axis gyroscope for MPU6050 and display them on the screen.

Components

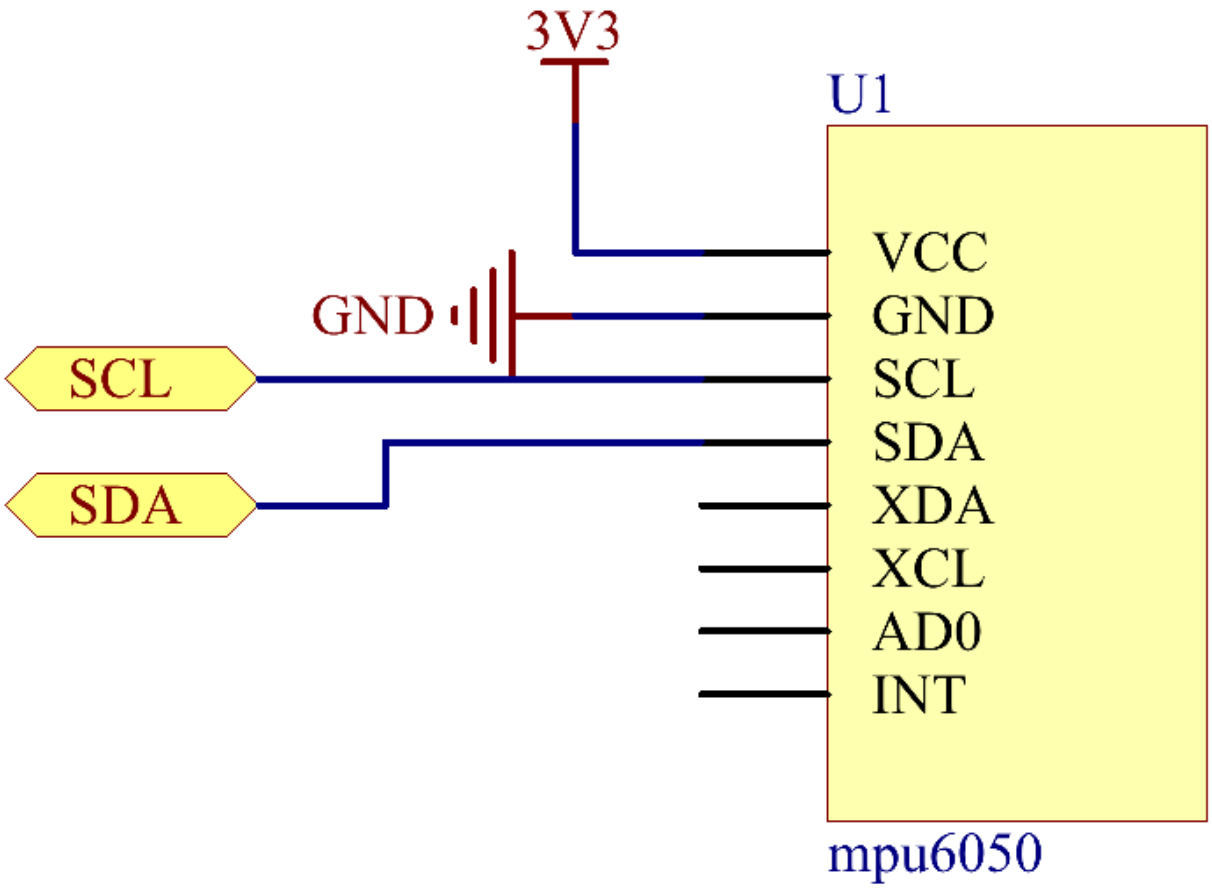
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
</td><td>1 * MPU6050
 
</td></tr>
<tr><td colspan="2"></td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="3">1 * 40-pin Cable
 </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *MPU6050 Module*

Schematic Diagram

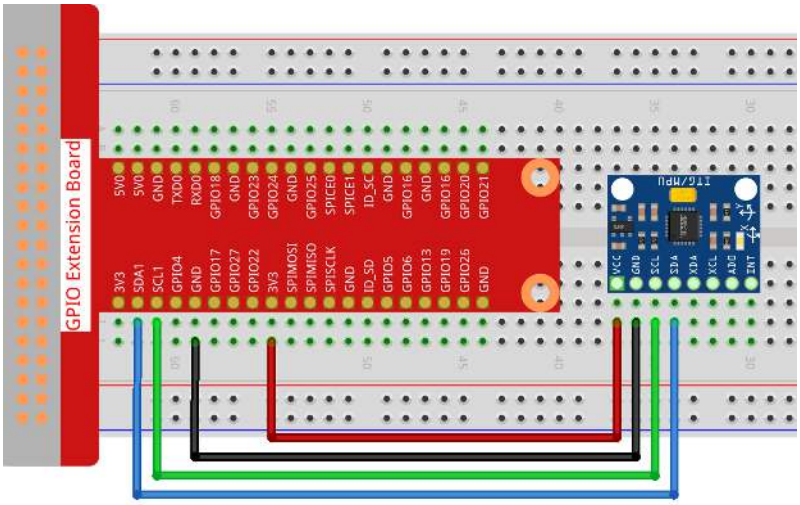
MPU6050 communicates with the microcontroller through the I2C bus interface. The SDA1 and SCL1 need to be connected to the corresponding pin.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



Step 2: Setup I2C (see Appendix *I2C Configuration*. If you have set I2C, skip this step.)

Step 2: Go to the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Install dependencies.

```
sudo npm install mpu6050-gyro
```

Step 4: Run the code.

```
sudo node mpu6050_module.js
```

With the code run, the angle of deflection of the x-axis and y-axis and the acceleration, angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

Code

```
var gyro = require("mpu6050-gyro");

var address = 0x68; //MPU6050 address
var bus = 1; //i2c bus used
var gyro = new gyro( bus,address );

async function update_telemetry() {

    var gyro_xyz = gyro.get_gyro_xyz();
    var accel_xyz = gyro.get_accel_xyz();

    var gyro_data = {
        gyro_xyz: gyro_xyz,
        accel_xyz: accel_xyz,
        rollpitch: gyro.get_roll_pitch( gyro_xyz, accel_xyz )
    }
```

(continues on next page)

(continued from previous page)

```
    console.log(gyro_data);

    setTimeout(update_telemetry, 500);
}

if ( gyro ) {
    update_telemetry();
}
```

Code Explanation

```
var gyro = require("mpu6050-gyro");

var address = 0x68; //MPU6050 address
var bus = 1; //i2c bus used
var gyro = new gyro( bus,address );
```

Import the `mpu6050-gyro` module, determine the MPU6050 address and the bus creation object gyro, It is convenient to call the encapsulated functions in the module.

Note: About this module, please refer to: https://www.npmjs.com/package/mpu6050-gyro

```
var gyro_xyz = gyro.get_gyro_xyz();
var accel_xyz = gyro.get_accel_xyz();

var gyro_data = {
    gyro_xyz: gyro_xyz,
    accel_xyz: accel_xyz,
    rollpitch: gyro.get_roll_pitch( gyro_xyz, accel_xyz )
}

console.log(gyro_data);

setTimeout(update_telemetry, 500);
```

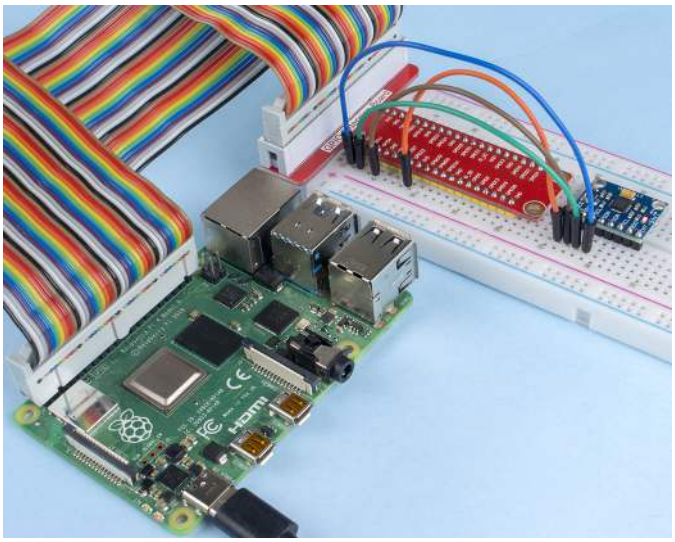
The module encapsulates three available functions:

`gyro.get_gyro_xyz()`: Returns JSON object with raw x,y,z datas from gyroscope.

`gyro.get_accel_xyz()`: Returns JSON object with raw x,y,z datas from accelerometer.

`gyro.get_roll_pitch( gyro_xyz, accel_xyz )`: Returns JSON object with roll and pitch in degrees.

Phenomenon Picture



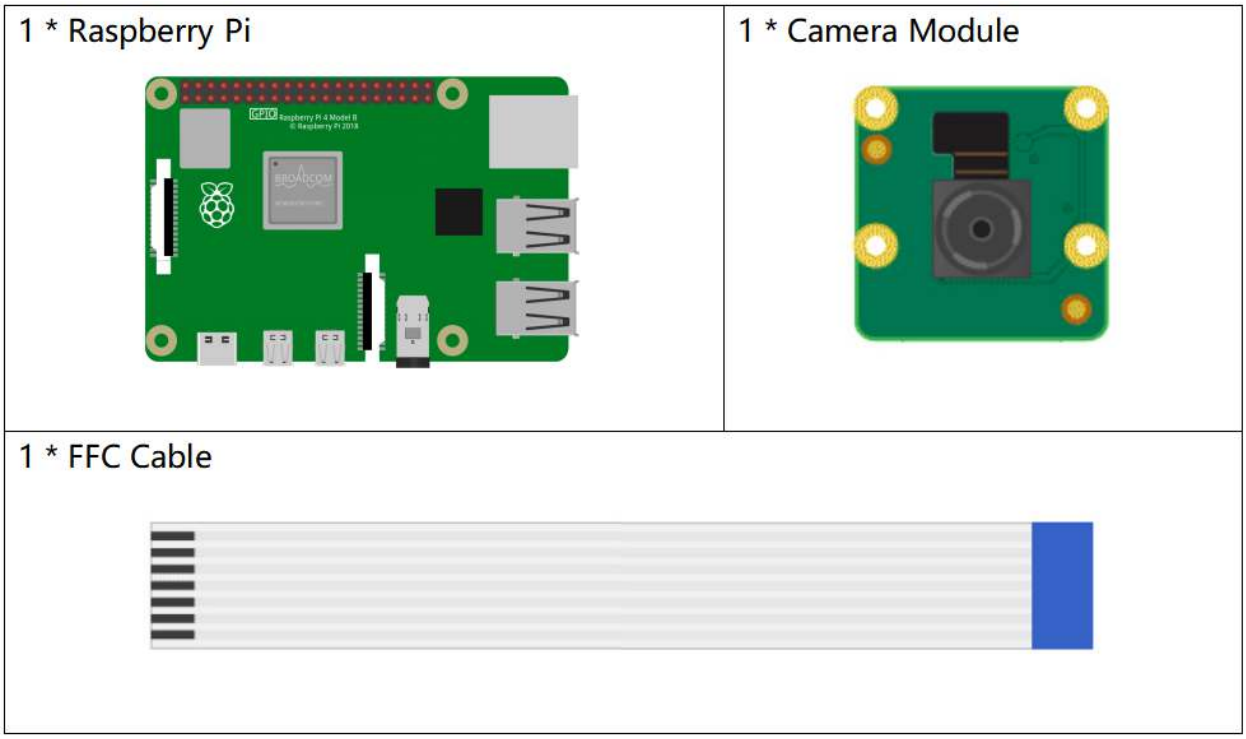
9.6 Extension

9.6.1 3.1.1 Photograph Module

Introduction

In this kit, equipped with a camera module, let's try to take a picture with Raspberry Pi.

Components



For more information on how to connect the camera module and its configuration, please refer to *Camera Module*.

Experimental Procedures

Step 1: Go into the Raspberry Pi Desktop. You may need a screen for a better experience, refer to: Connect your Raspberry Pi. Or access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

Step 2: Open a Terminal and get into the folder of the code.

```
cd /home/pi/raphael-kit/nodejs/
```

Step 3: Run the code.

```
sudo node camera.js
```

After the code runs, the camera will take a photo. Now you can see the photo named `test.jpg` in current directory.

Code

```
const exec = require('child_process').exec;

exec('libcamera-still -o test.jpg');
```

Code Explanation

```
const exec = require('child_process').exec;
```

Import the `child_process` module, which allows nodejs to perform various operations on child processes, including creating child processes to directly execute system commands.

Note: About this module, please refer to: https://nodejs.org/api/child_process.html

```
exec('libcamera-still -o test.jpg');
```

After enabling the Camera function, you can directly use the command `libcamera-still -o test.jpg` to capture photos in the terminal. We can also use the method provided by the `child_process` module `child_process.exec(cmd, [options] , callback)` to create child processes to run system commands.

By adding loop and delay functions, we can also achieve the effect of timing photos or time-lapse video.

CHAPTER

TEN

PLAY WITH SCRATCH

Scratch is a block-based visual programming language and website targeted primarily at children 8-16 as an educational tool for coding. Users of the site can create projects on the web using a block-like interface. The service is developed by the MIT Media Lab, has been translated into 70+ languages, and is used in most parts of the world.

Here you will learn to use Scratch 3 in Raspberry Pi and access Raspberry Pi GPIO with Scratch 3, which is not possible with online Scratch.

Note: When programming with Scratch 3, you may need a screen for a better experience, refer to: Connect your Raspberry Pi. Of course, if you don't have a screen, you can also access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

10.1 Quick Guide on Scratch

Note: When programming with Scratch 3, you may need a screen for a better experience, refer to: Connect your Raspberry Pi. Of course, if you don't have a screen, you can also access the Raspberry Pi desktop remotely, for a detailed tutorial please refer to *Remote Desktop*.

In addition, Scratch 3 needs at least 1GB of RAM to run, and we recommend a Raspberry Pi 4 with at least 2GB RAM. While you can run Scratch 3 on a Raspberry Pi 2, 3, 3B+, or a Raspberry 4 with 1GB RAM, performance on these models is reduced, and depending on what other software you run at the same time, Scratch 3 may fail to start due to lack of memory.

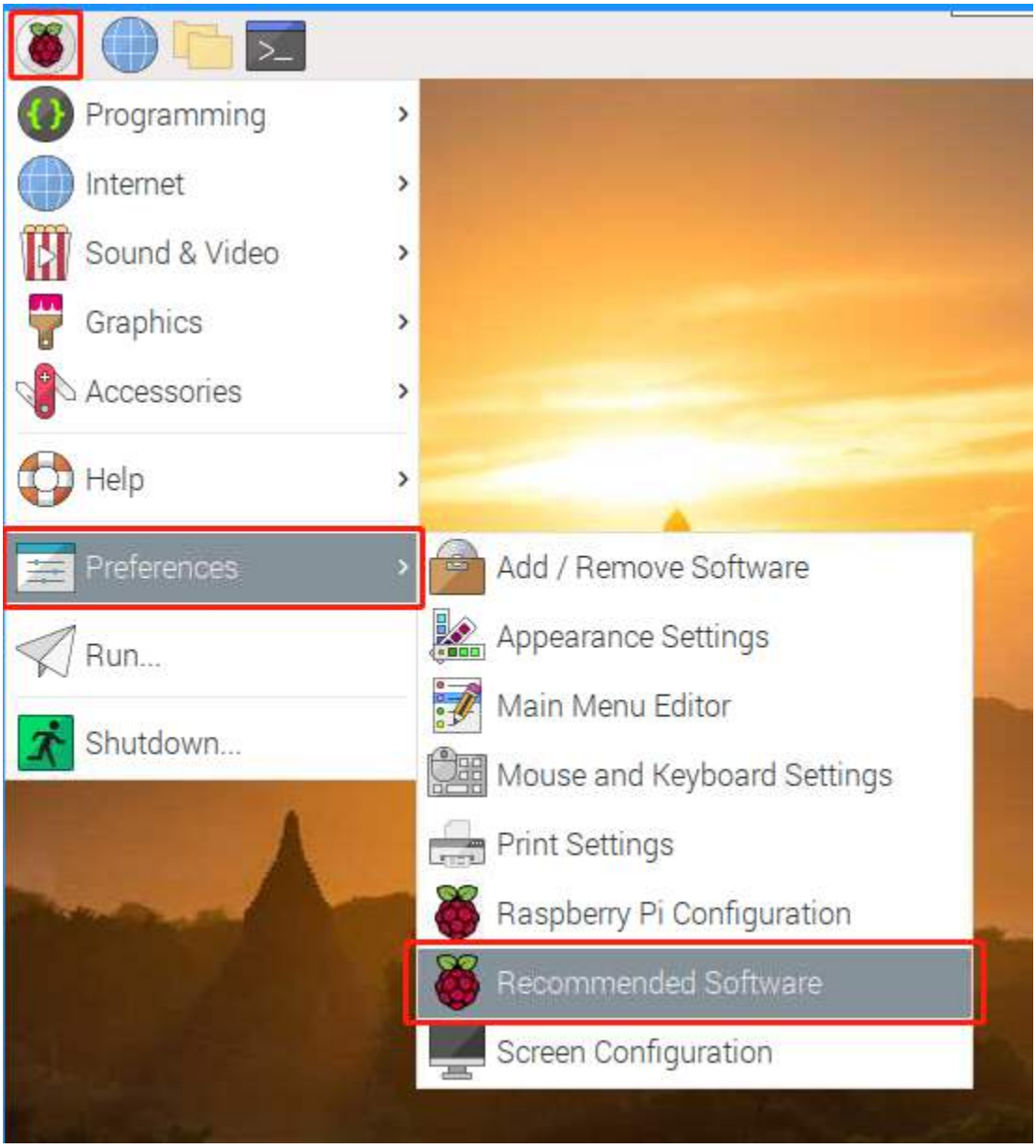
10.1.1 Install Scratch 3

When installing Raspberry Pi OS (*Installing the OS*), you have to choose the version with desktop, either with desktop only or with desktop and recommended software.

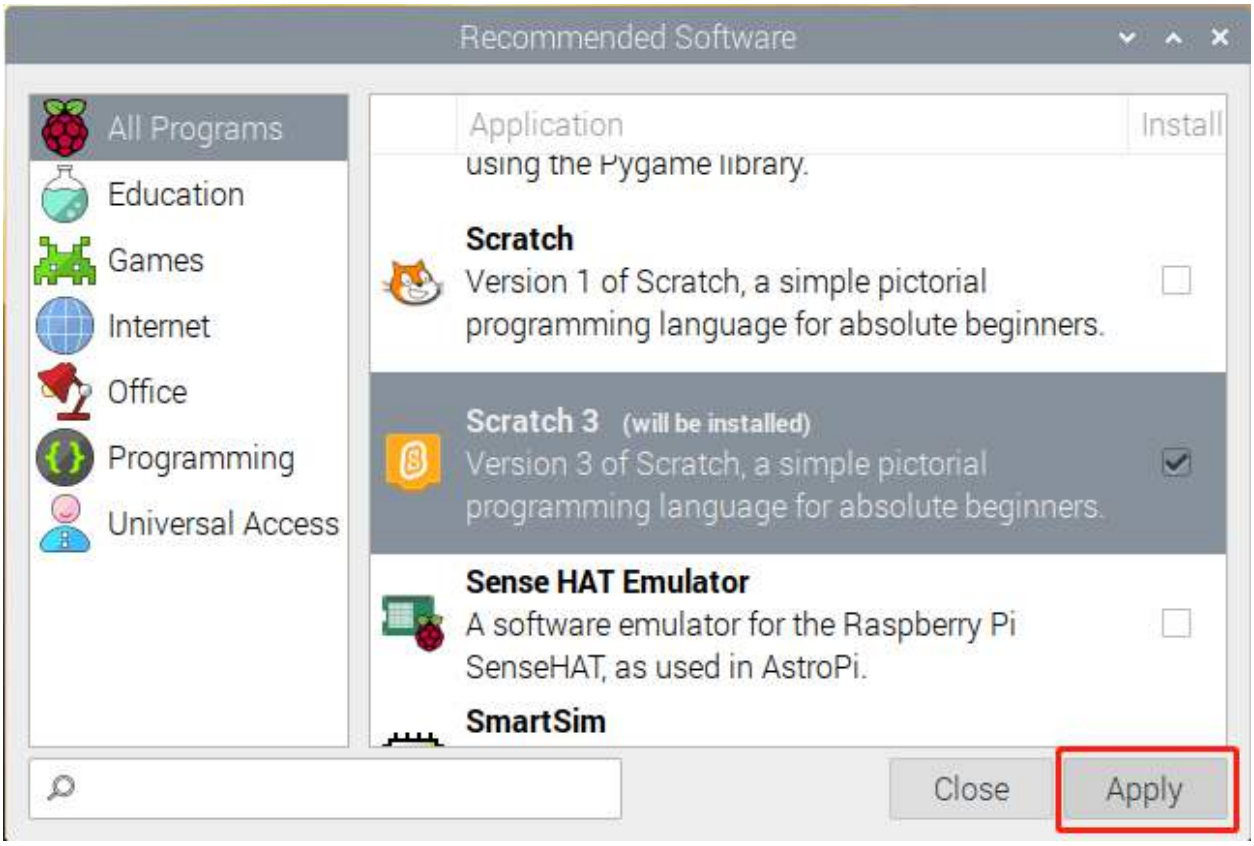
If you install the version with the recommended software, you can see Scratch 3 in the system menu at **Programming**.

If you installed the desktop-only version, you will need to install Scratch 3 manually, as described below.

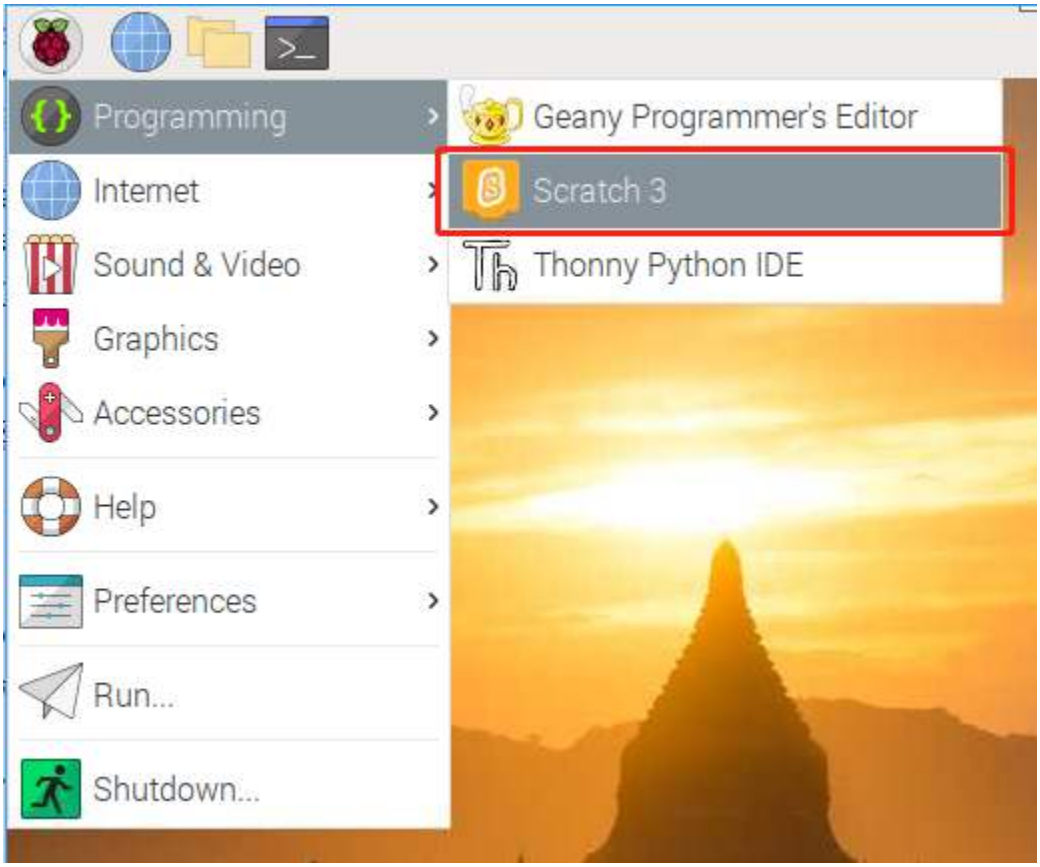
Open up the menu, click on **Preferences** -> **Recommended Software**.



Find scratch 3 and check it, then click **Apply** and finally wait for the installation to finish.



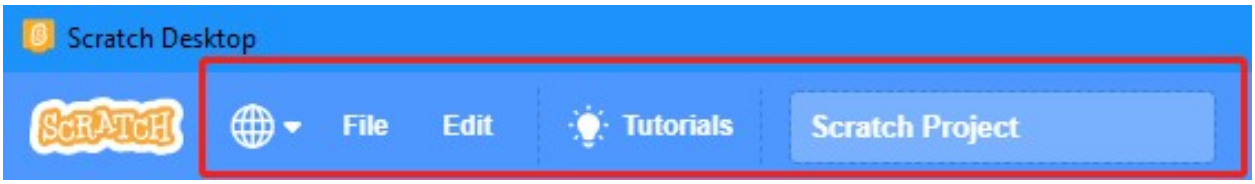
Once the installation is complete, you should see it on the **Programming** in the system menu.



10.1.2 About Scratch 3's Interface

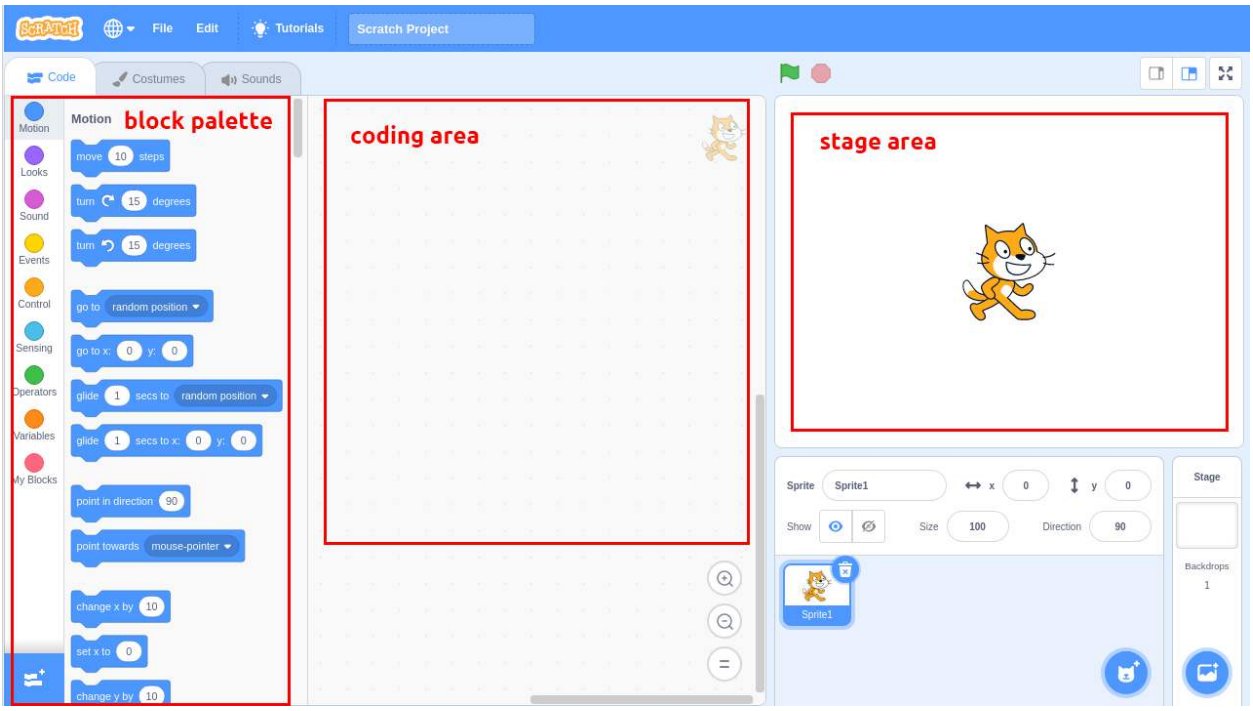
Scratch 3 is designed to be fun, educational, and easy to learn. It has tools for creating interactive stories, games, art, simulations, and more, using block-based programming. Scratch also has its own paint editor and sound editor built-in.

The top of scratch 3 has some basic options, the first one from left to right is the language option, you can choose different languages for programming. The second one is the **File** option, you can create new files, read local files and save current files with this option. The third is the **Edit** option, which allows you to resume some deletion operations and enable the acceleration mode (in which the sprite movement becomes particularly fast). The fourth is the **Tutorials** option, which allows you to view tutorials for some projects. The fifth is the file naming option, where you can rename the project.

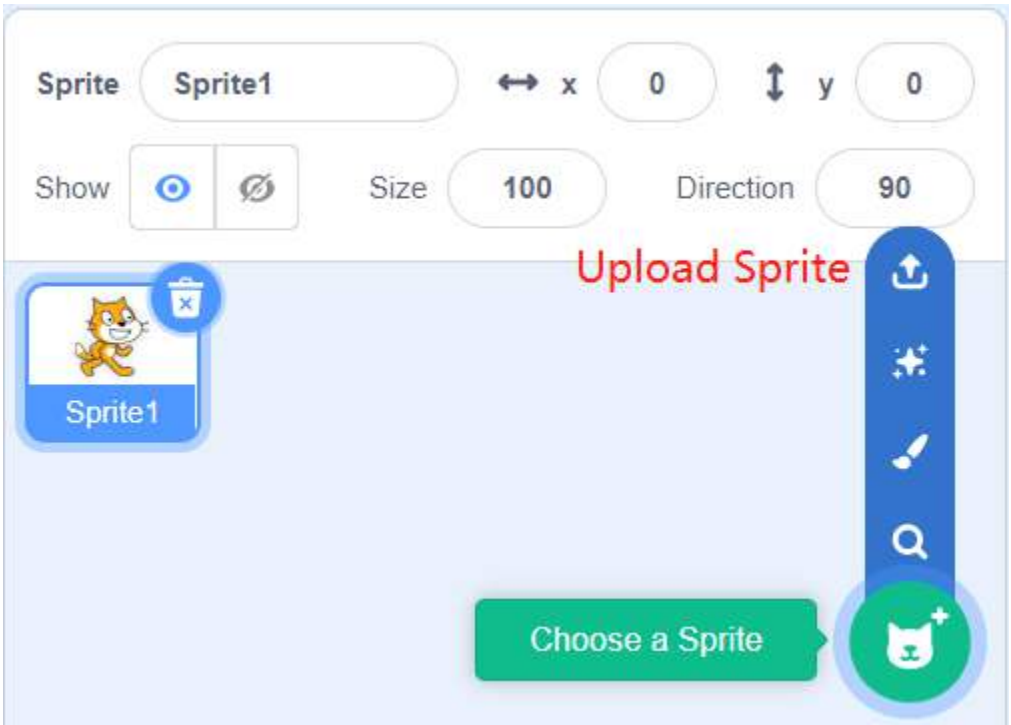


Code

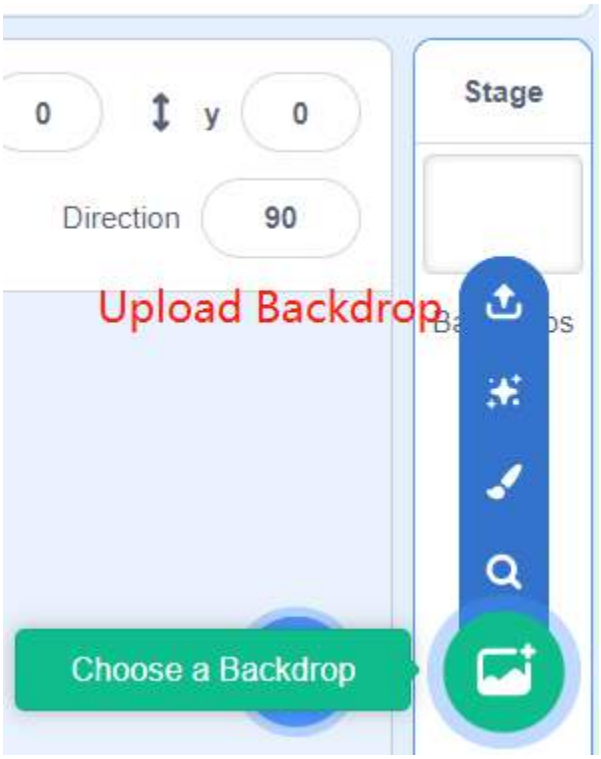
It has three main sections: a stage area, block palette, and coding area. Programming by clicking and dragging the block on the block palette to the coding area, and finally your programming results will be displayed on the stage area.



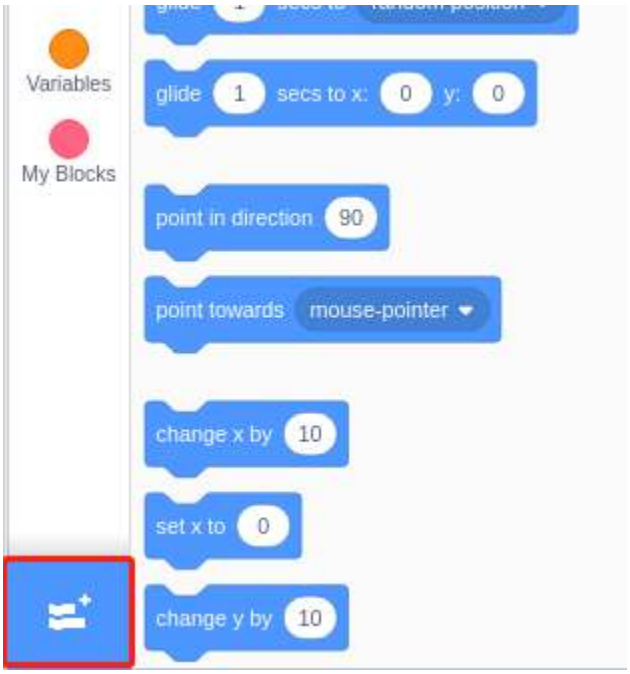
Here is the sprites area of Scratch 3. Above the area are the basic parameters of the sprites, you can add sprites that come with Scratch 3 or upload local sprites.



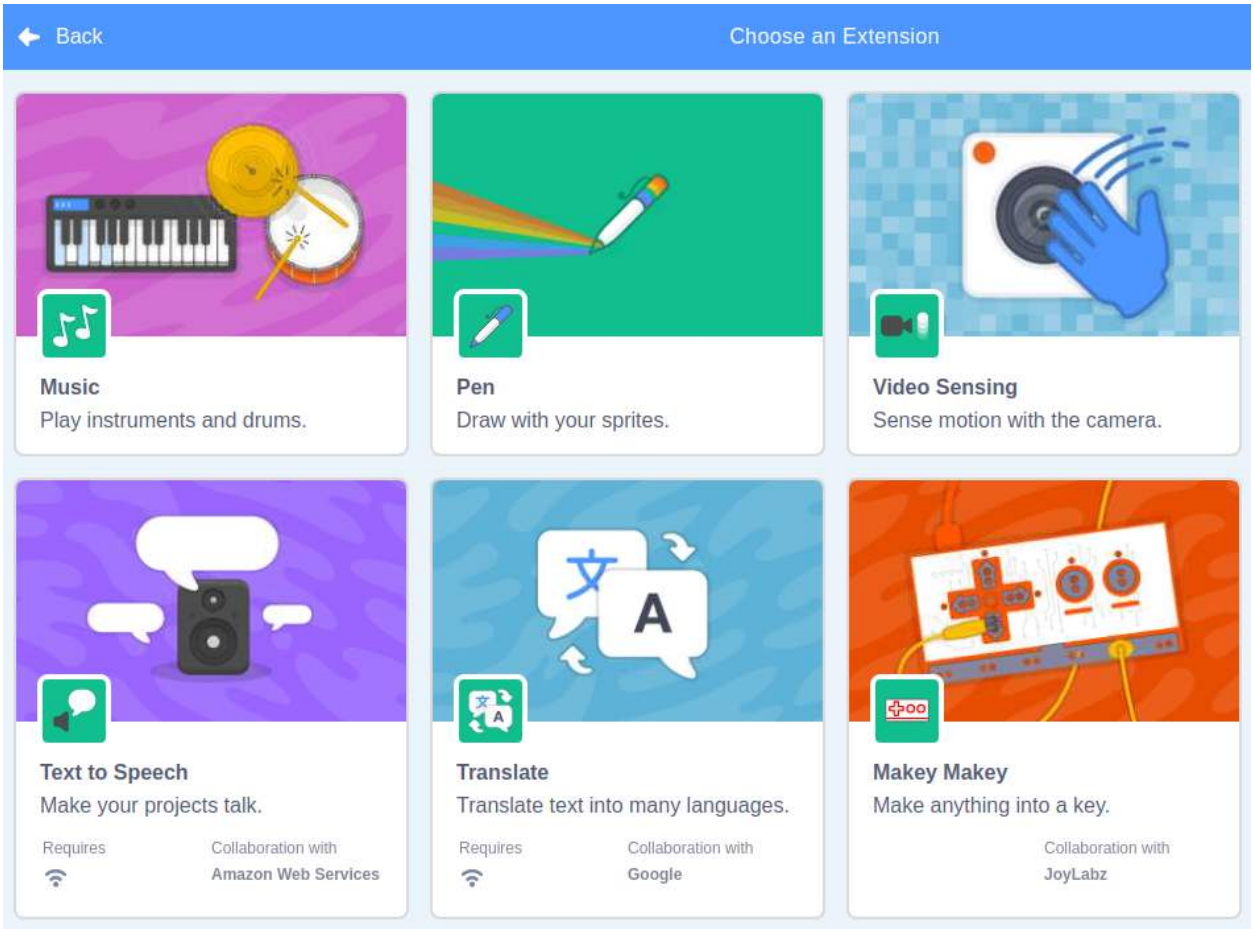
Here is the Scratch 3 backdrop area, mainly to add a suitable backdrop for your stage, you can add the backdrop that comes with Scratch 3 or upload a local one.



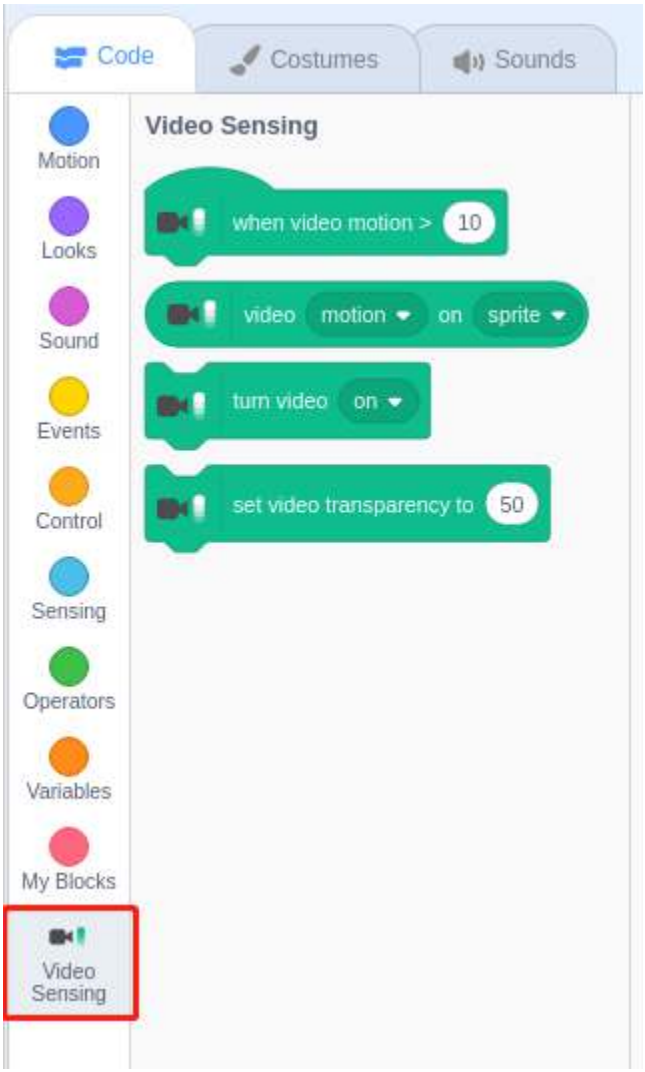
This is a **Add Extension** button.



In Scratch 3, we can add all kinds of useful extensions, here we take **Video Sensing** as an example and click on it.

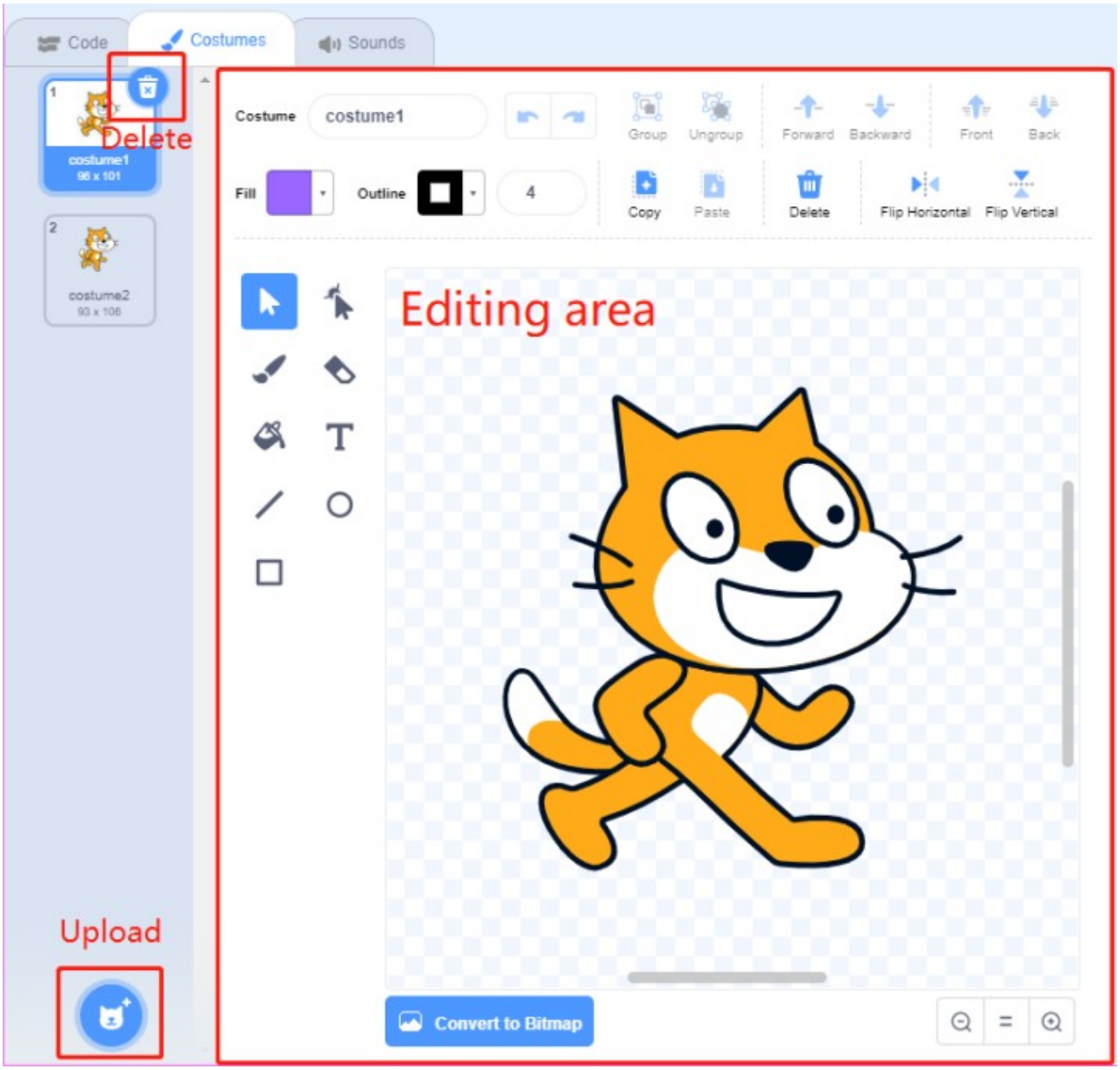


You will see it on the block palette and you can use the functions associated with this extension. If you have a camera connected, you will see the camera screen on the stage area.



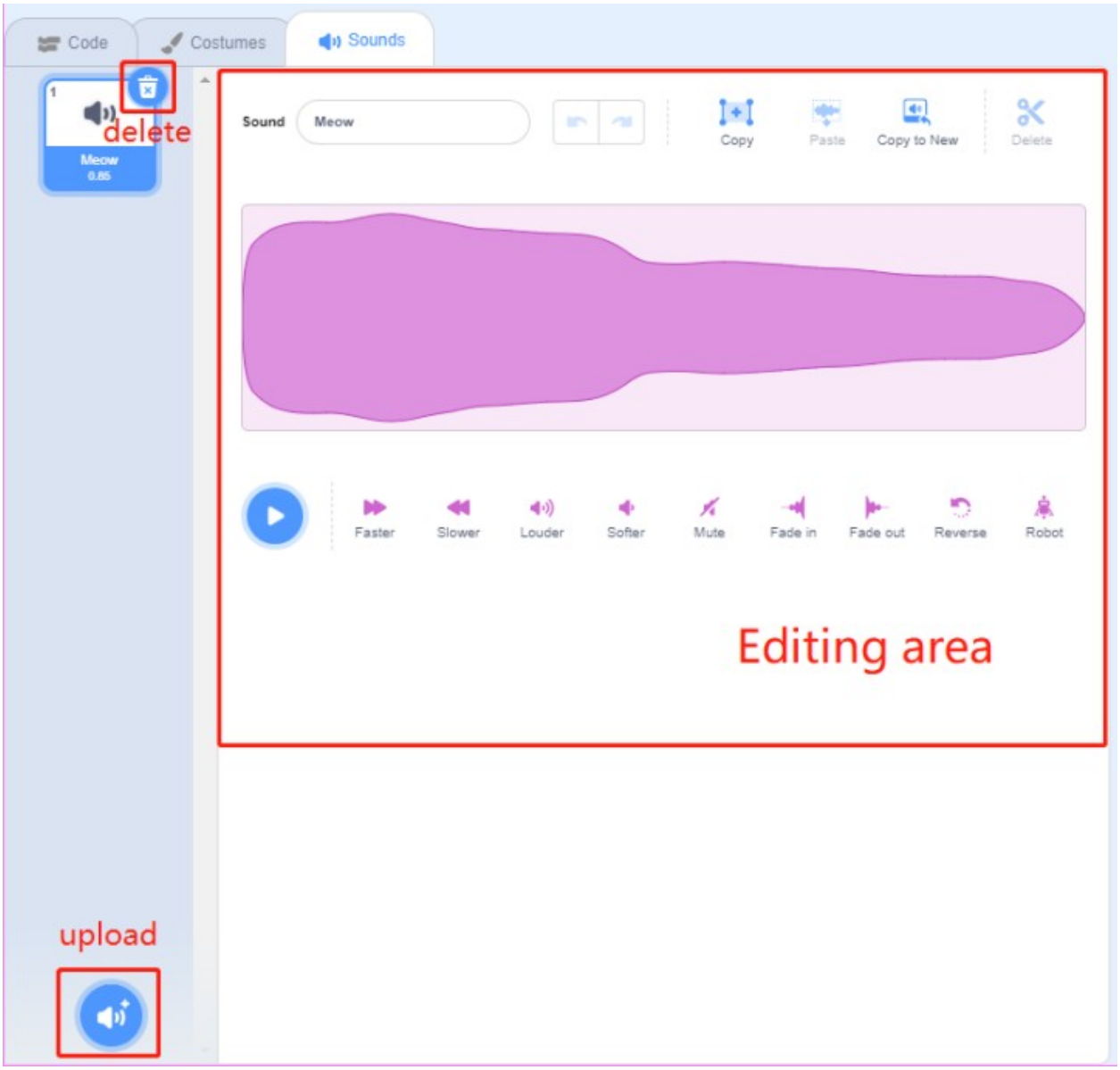
Costumes

Click on the **Costumes** option in the upper left corner to enter the costumes palette. Different costumes allow the sprites to have different static movements, and when these static movements are stitched together, they form a coherent dynamic movement.



Sounds

You may need to use some music clips to make your experiments more interesting. Click on the **Sounds** option in the upper left corner and you can edit the current sound or select/upload a new one.



10.2 Projects

In this chapter, you will play with Scratch 3 through 18 projects.

If you are a user who has just used Scratch 3, we recommend that you try the projects in order so that you can quickly get started with Scratch 3.

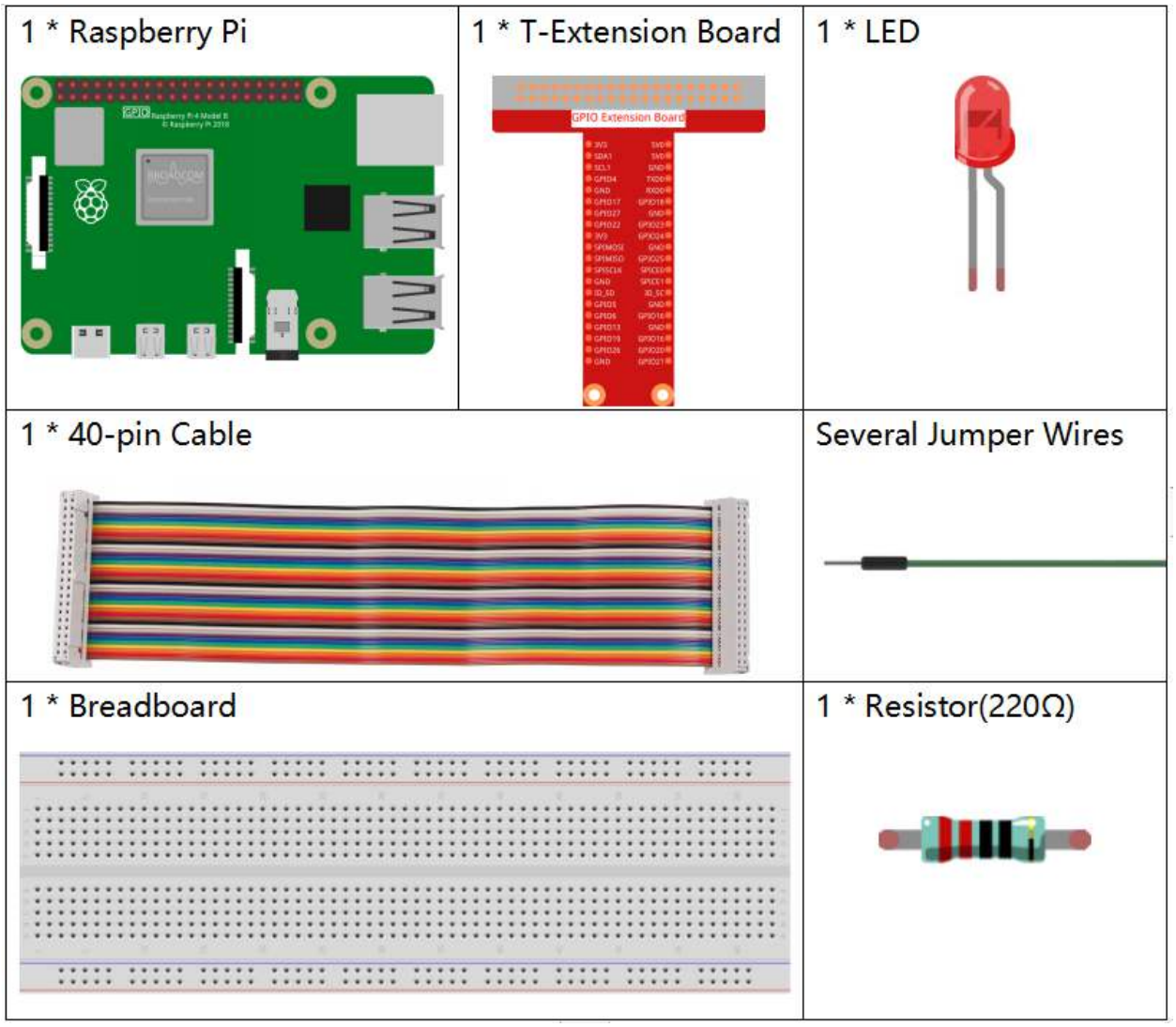
Note: Before trying the projects, you should have downloaded the relevant materials and code files. Open a Terminal and enter the following command to download them from github.

```
git clone https://github.com/sunfounder/raphael-kit.git
```

10.2.1 1.1 Wand

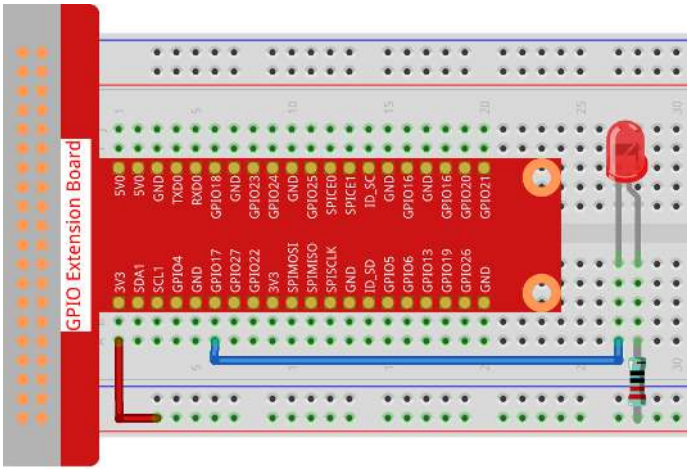
Today we will use LED, Raspberry Pi and Scratch to make a fun game. When we wave the magic wand, the LED will blink.

Required Components



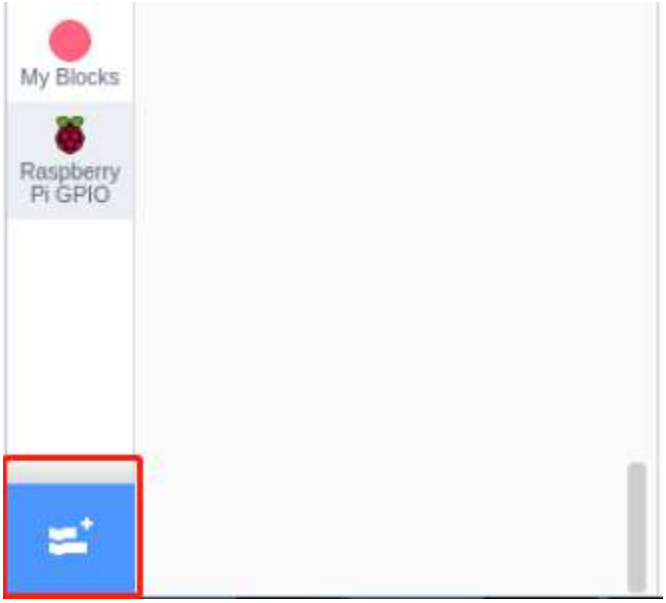
- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED*

Build the Circuit

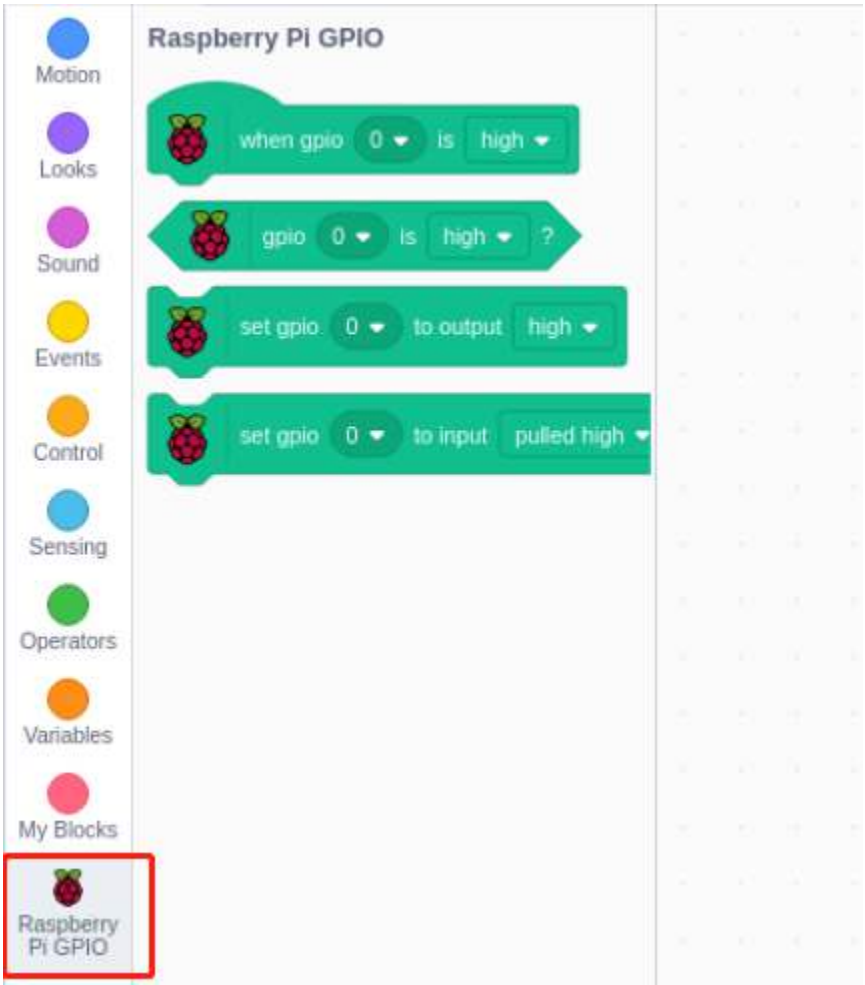


Add GPIO Extension

Click on the **Add Extension** button in the bottom left cornerthen add the **Raspberry Pi GPIO**, an extension we use for all of our Scratch projects.

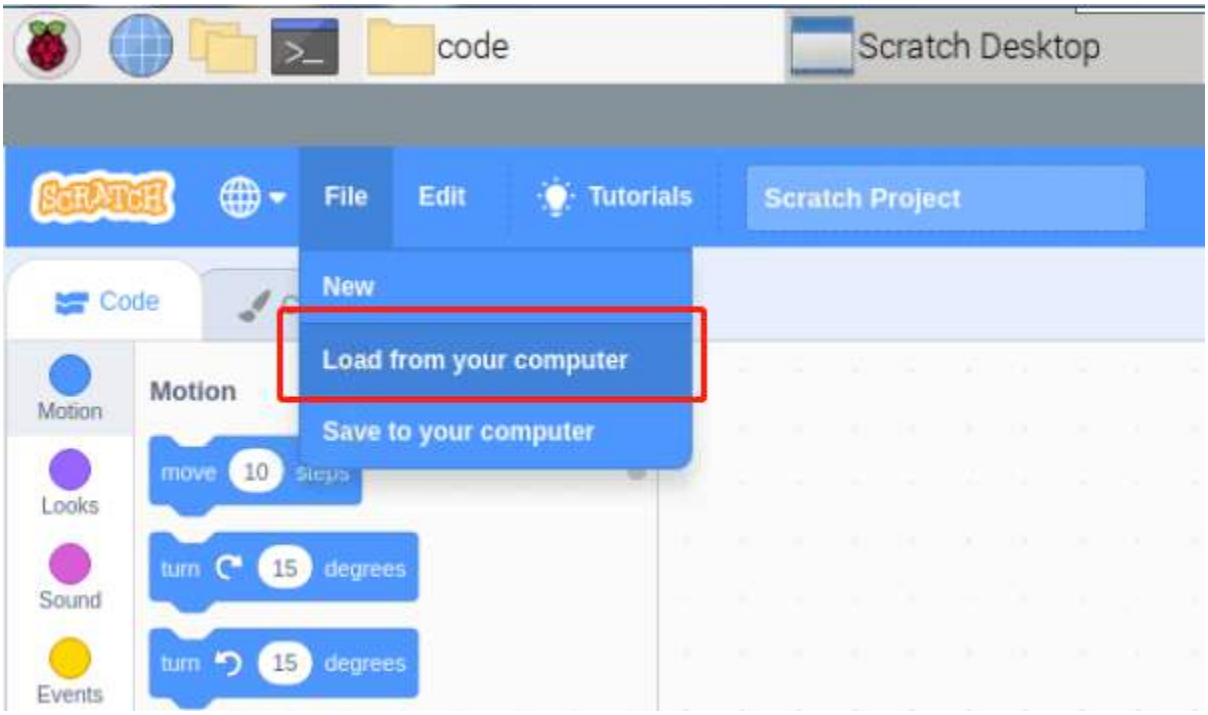


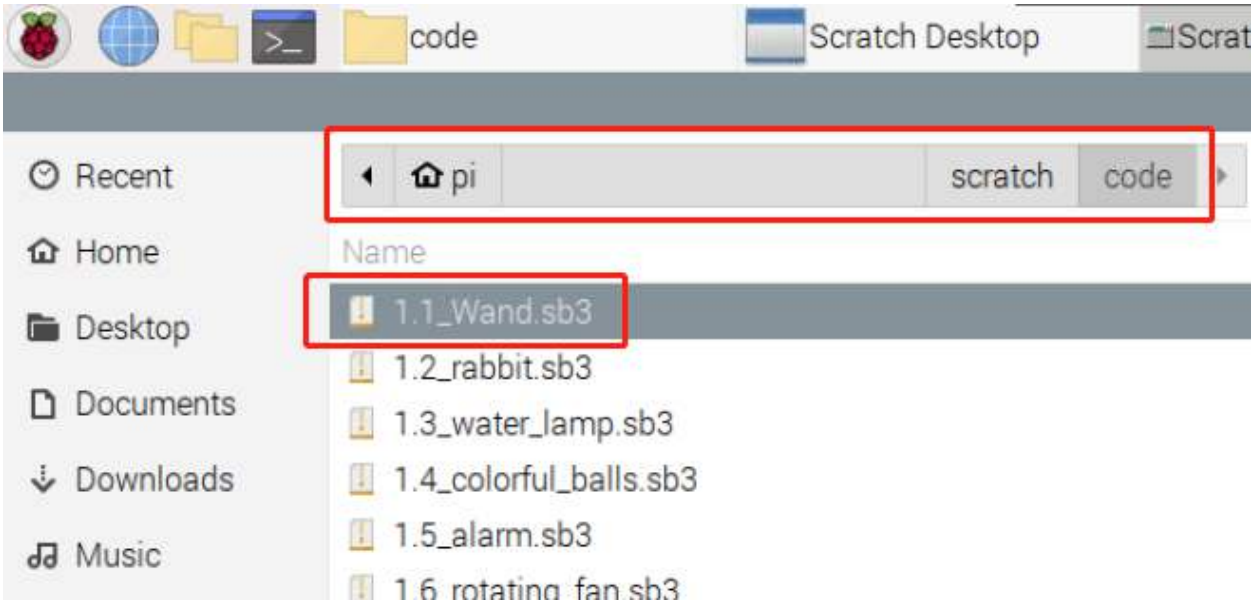
Raspberry Pi GPIO
Control Raspberry Pi GPIO lines
Collaboration with
Raspberry Pi
Raspberry Pi Sense HAT
Control Raspberry Pi Sense HAT
Collaboration with
Raspberry Pi


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

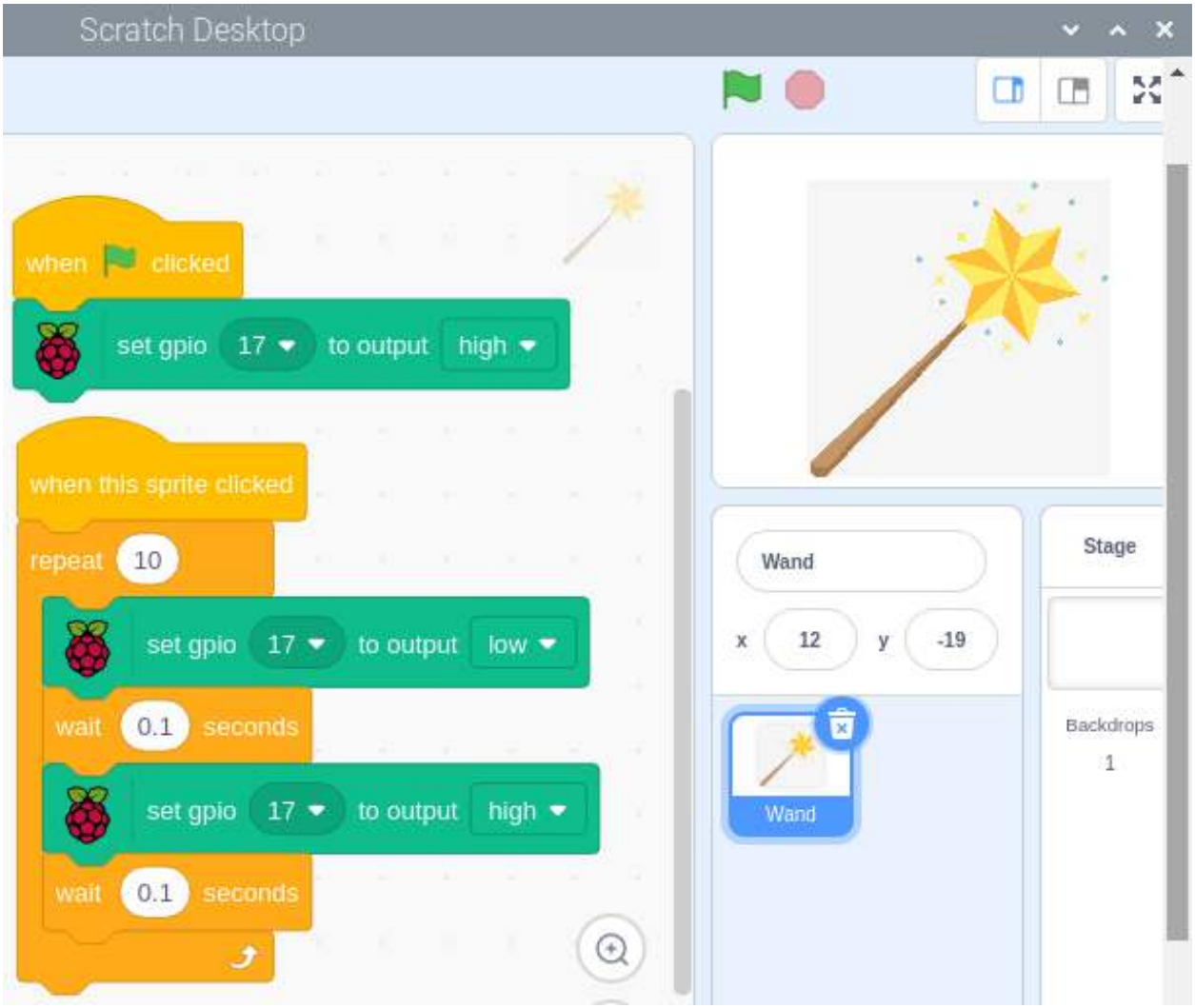
Load the Code and See What Happens

Load the code file from your computer(`home/pi/raphael-kit/scratch/code`) to Scratch 3.



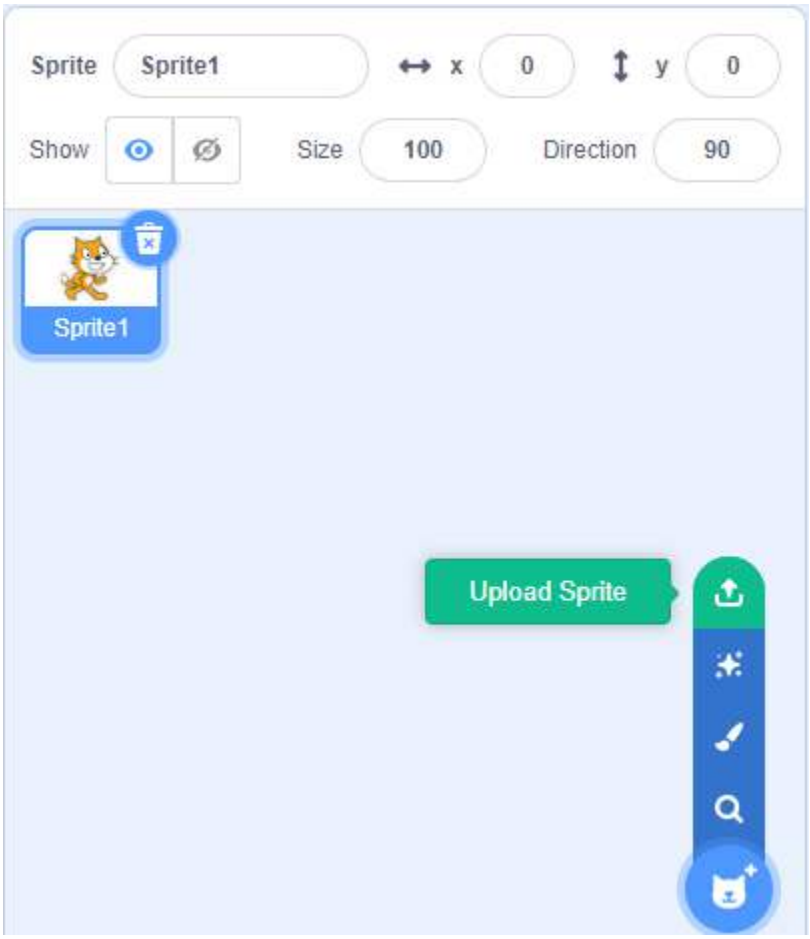


After clicking the magic wand in the stage area, you will see the LED will blink for two seconds.

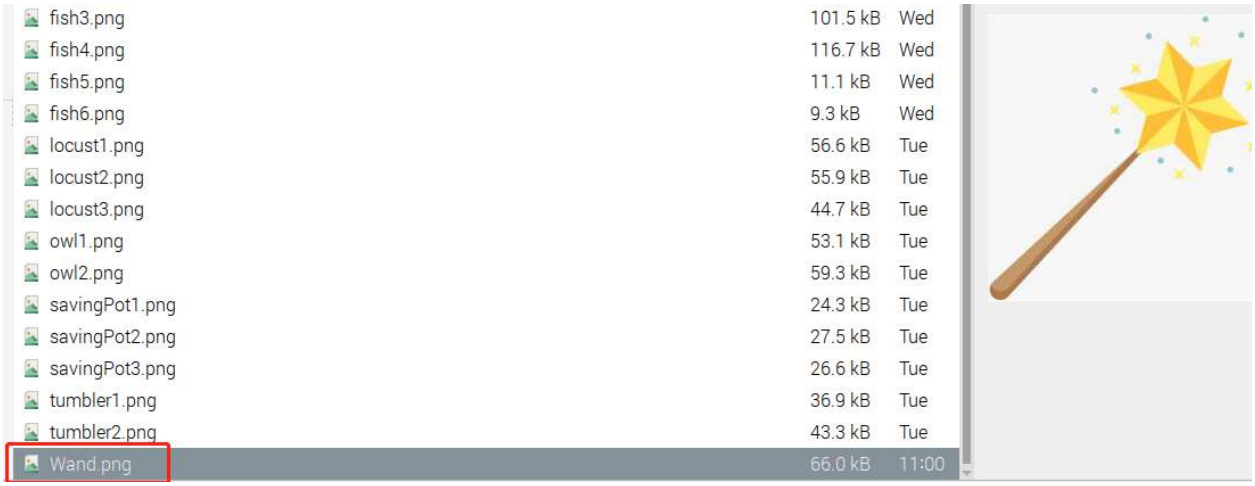


Tips on Sprite

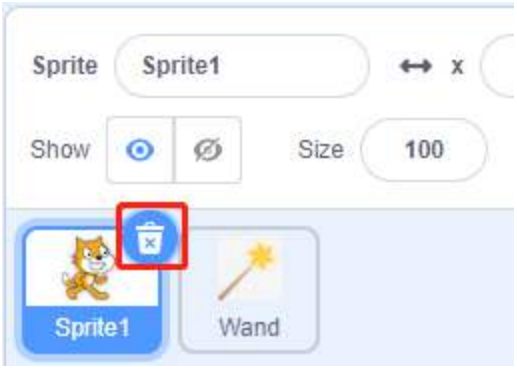
Click on the **Upload Sprite**.



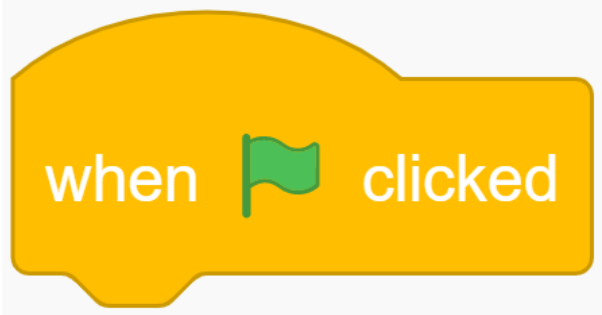
Upload **Wand.png** from the `home/pi/raphael-kit/scratch/picture` path to Scratch 3.



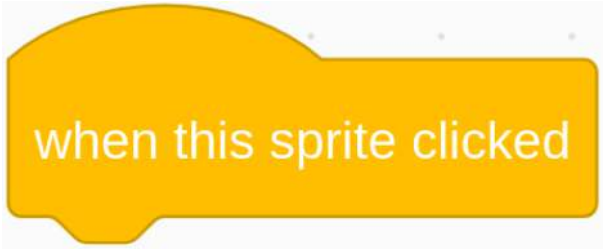
Finally, delete the **Sprite1**.



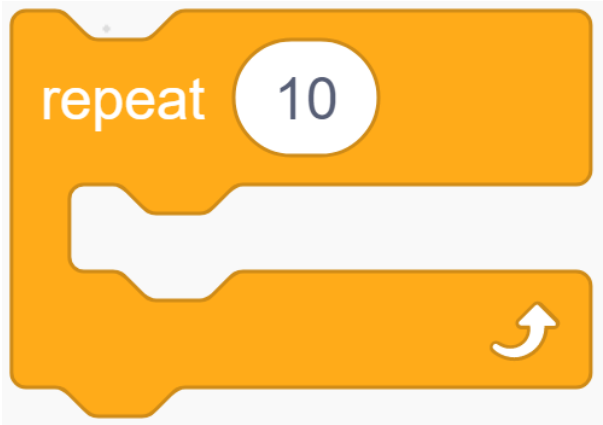
Tips on Codes



This is an event block whose trigger condition is to click on the green flag on the stage. A trigger event is required at the beginning of all codes, and you can select other trigger events in the **Events** category of the **block palette**.

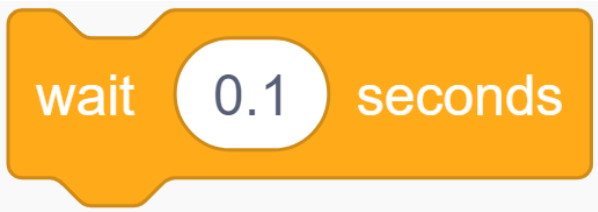


For example, we can now change the trigger event to a click on the sprite.

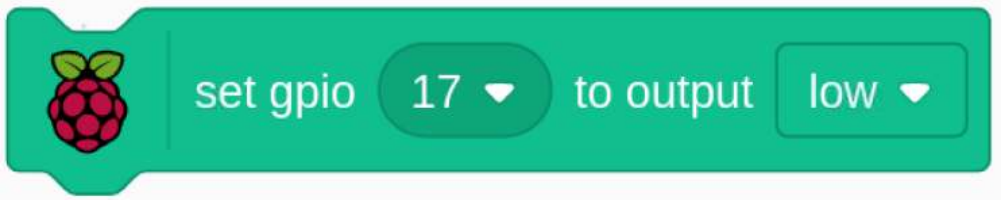


This is a block with a set number of cycles. When we fill in the number 10, the events in the block will be executed 10

times.



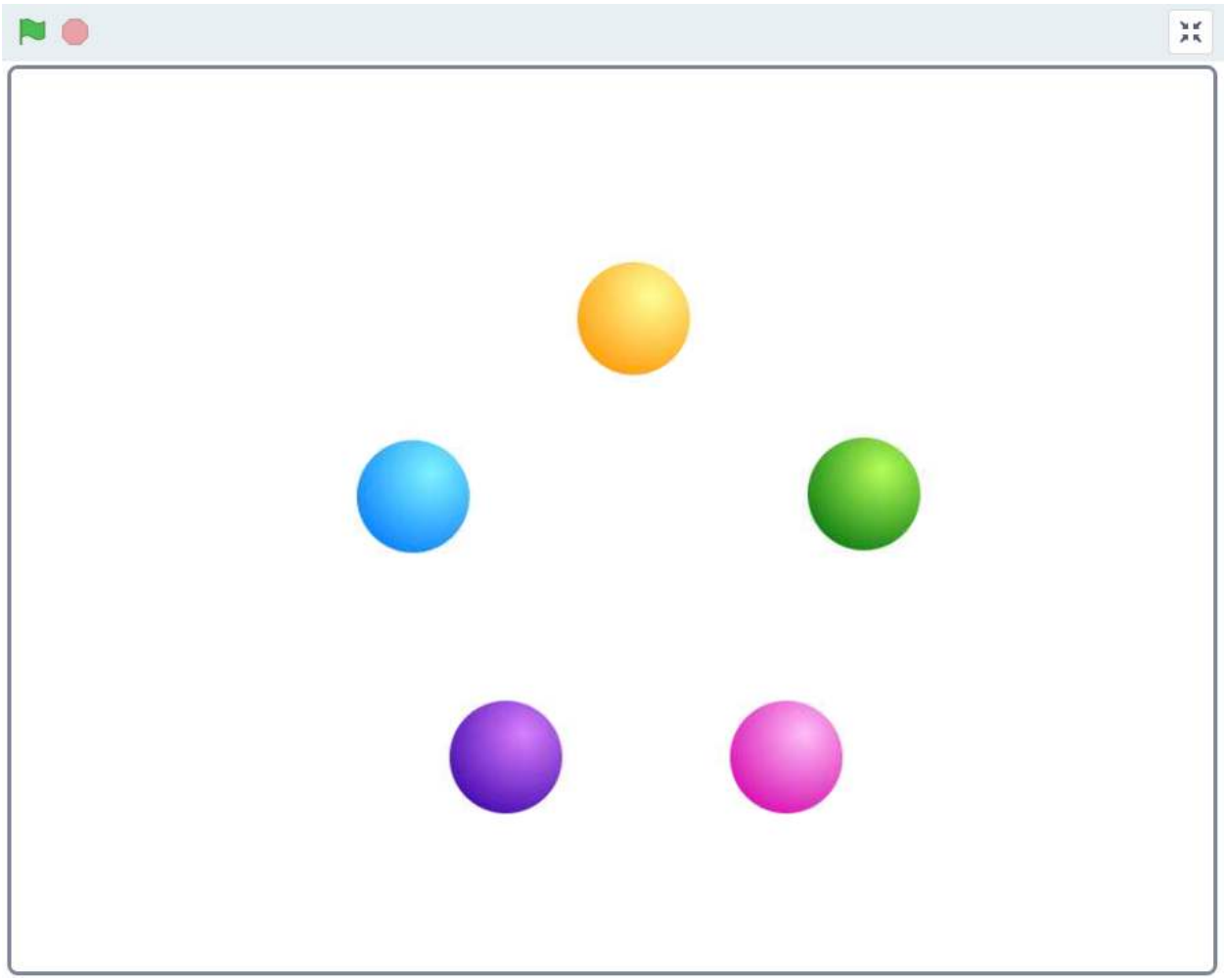
This block is used to pause the program for a period of time in seconds.



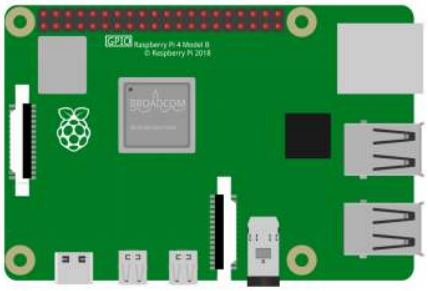
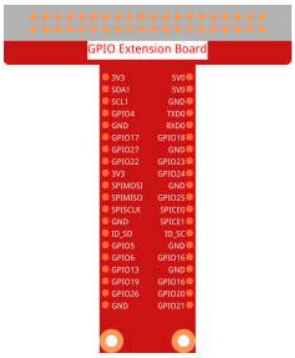
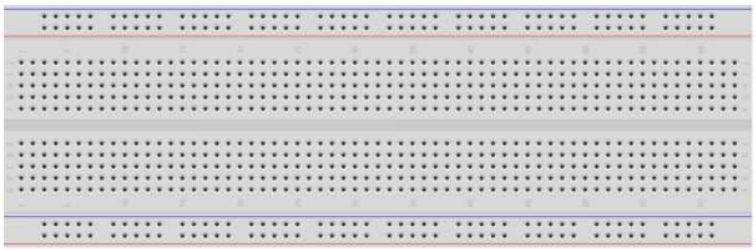
Since the BCM naming method is used in Scratch, this code is setting GPIO17(BCM17) as 0V (low level). Since the cathode of LED is connected to GPIO17, thus the LED will light up. On the contrary, if you set GPIO(BCM17) as high, the LED will turn off.

10.2.2 1.2 Colorful Balls

Clicking on different colored balls on the stage area will cause the RGB LED to light up in different colors.

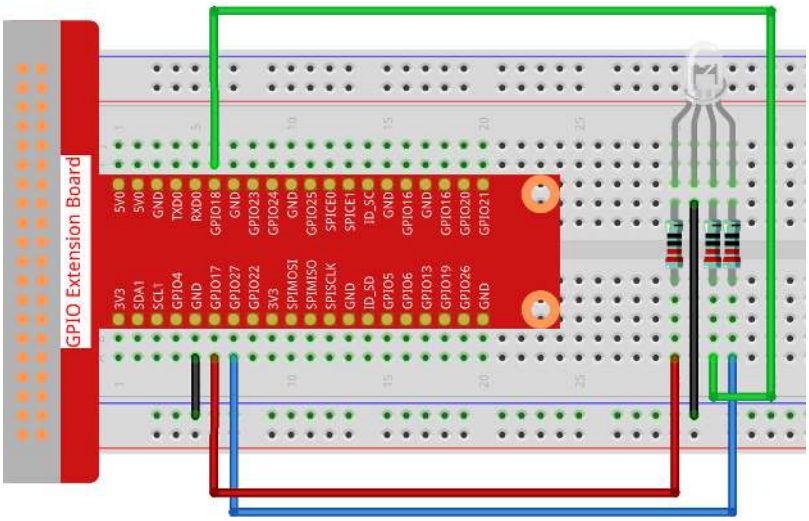


Required Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *RGB LED*

Build the Circuit

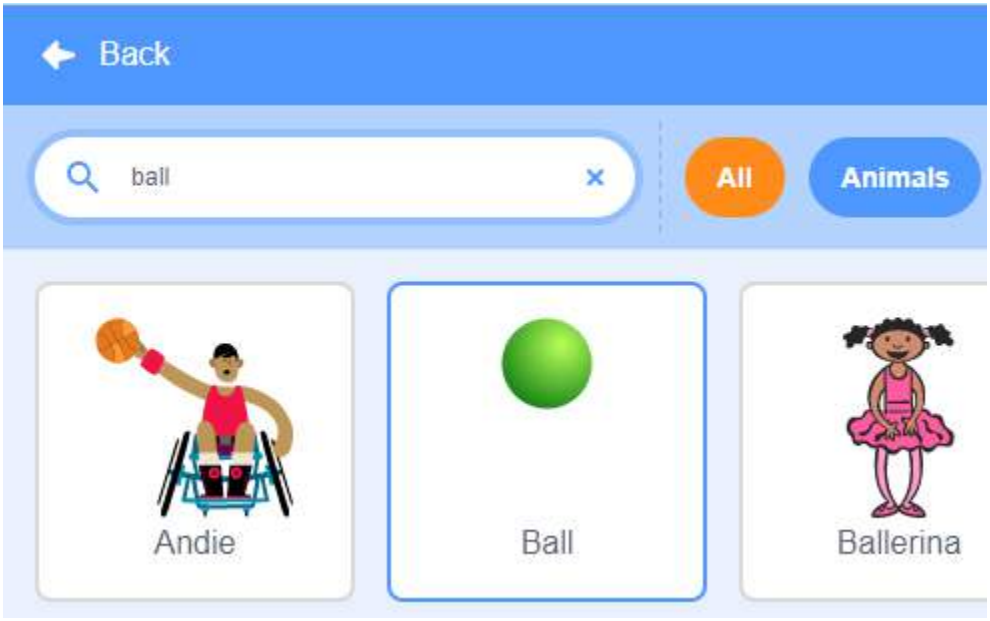


Load the Code and See What Happens

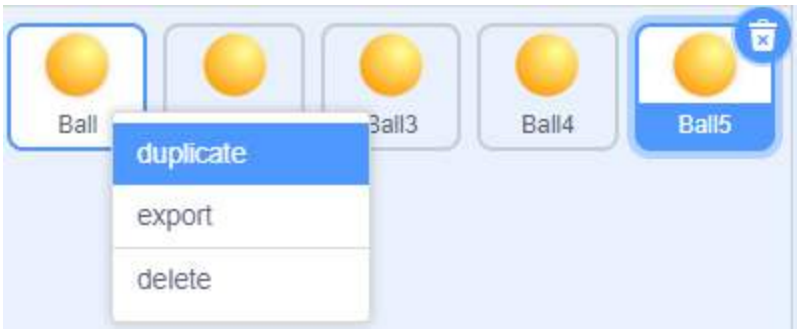
After loading the code file (`1.2_colorful_balls.sb3`) into Scratch 3, the RGB LED will light up yellow, blue, red, green or purple respectively when you click on the corresponding ball.

Tips on Sprites

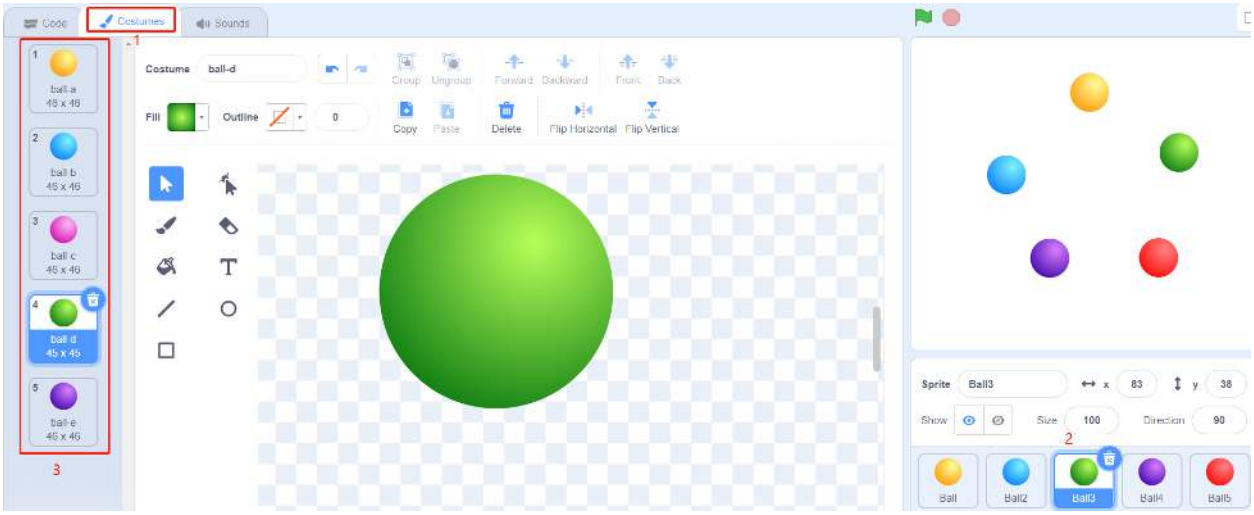
Delete the default sprite, then choose the **Ball** sprite.



And duplicate it 5 times.



Choose different costumes for these 5 **Ball** sprites and move them to the corresponding positions.

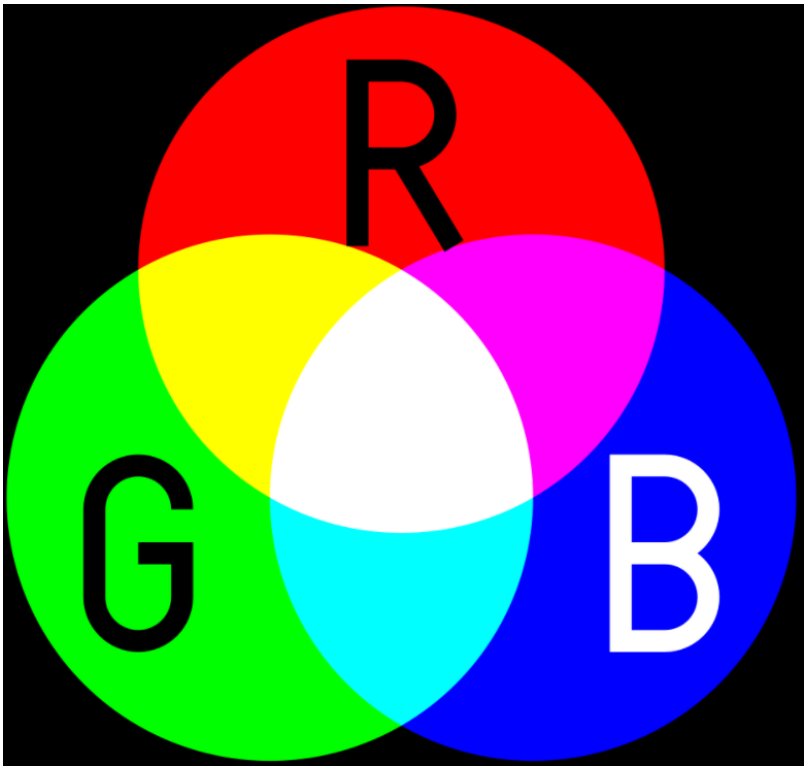


Tips on Codes

Before understanding the code, we need to understand the RGB color model.

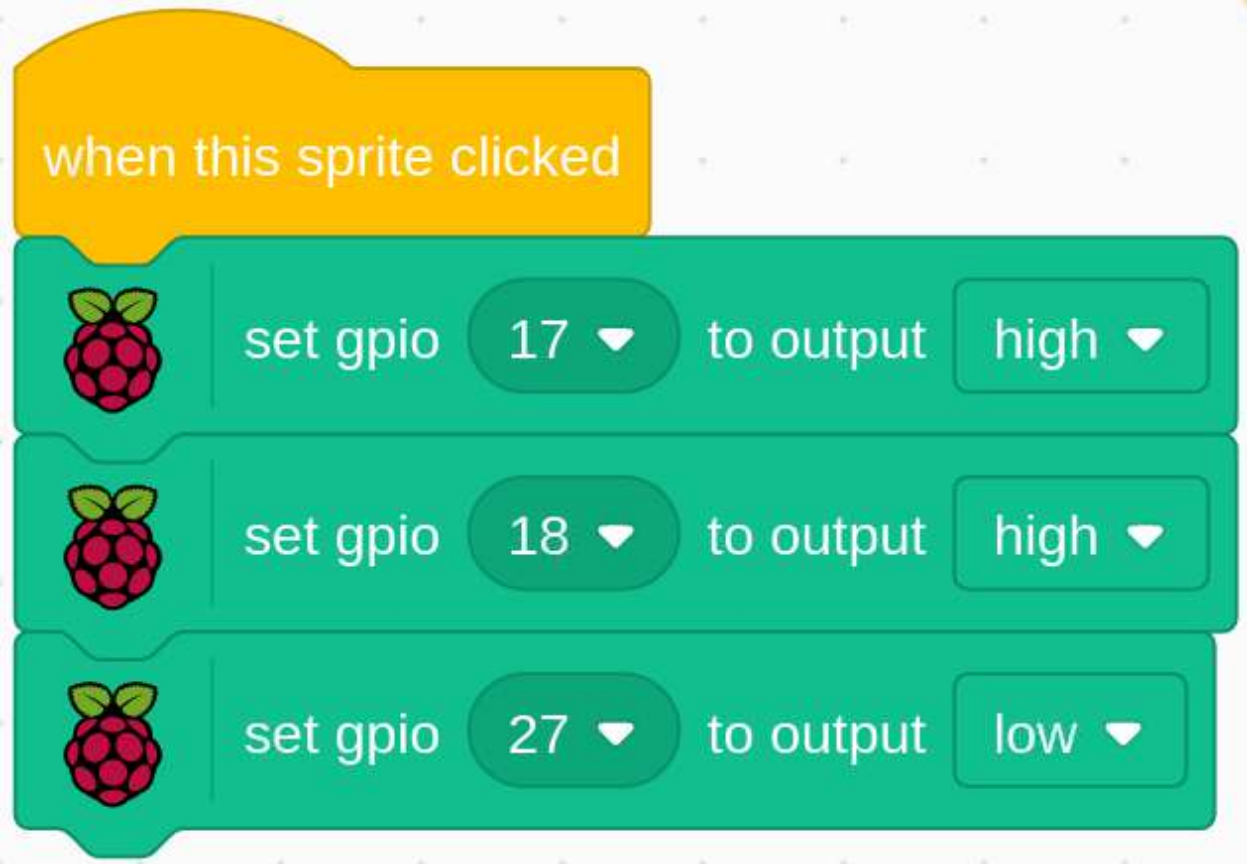
The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.

Additive color mixing: adding red to green yields yellow; adding green to blue yields cyan; adding blue to red yields magenta; adding all three primary colors together yields white.



An RGB LED is a combination of 3 LEDs(red LED, green LED, blue LED) in just one package, you can produce almost any color by combining those three colors. It has 4 pins, one of which is GND, and the other 3 pins control 3 LEDs respectively.

So the code to make the RGB LED light yellow is as follows.

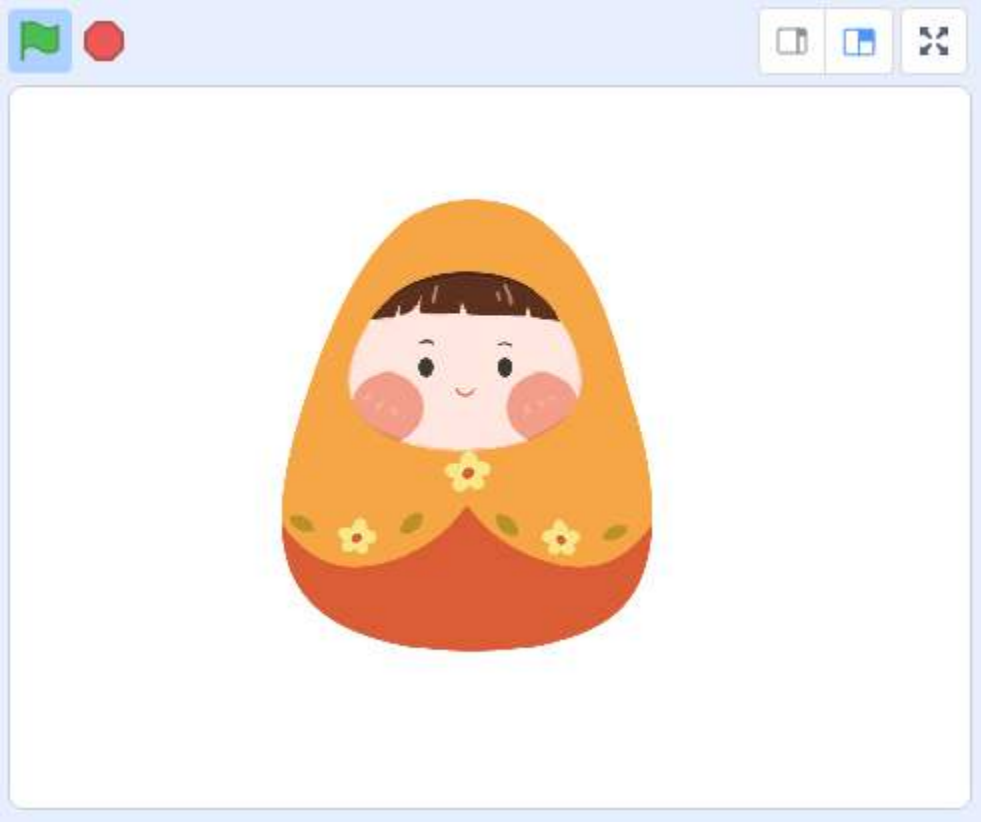


When the Ball sprite (yellow ball) is clicked, we set gpio17 high (red LED on), gpio18 high (green LED on) and gpio27 low (blue LED off) so that the RGB LED will light yellow.

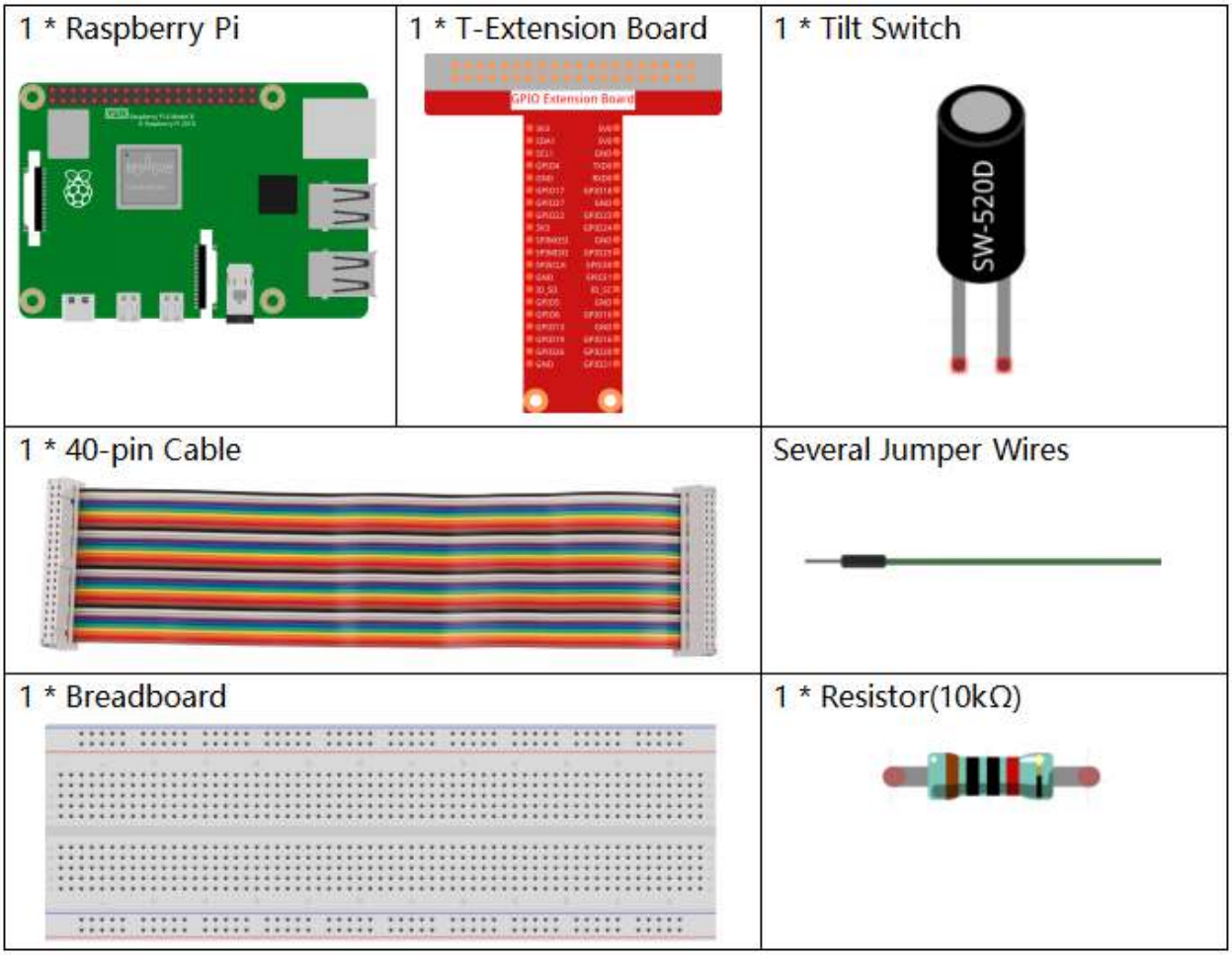
You can Write codes to other sprites in the same way to make the RGB LEDs light up in the corresponding colors.

10.2.3 1.3 Tumbler

In this project, we will make a tilt switch controlled tumbler toy.

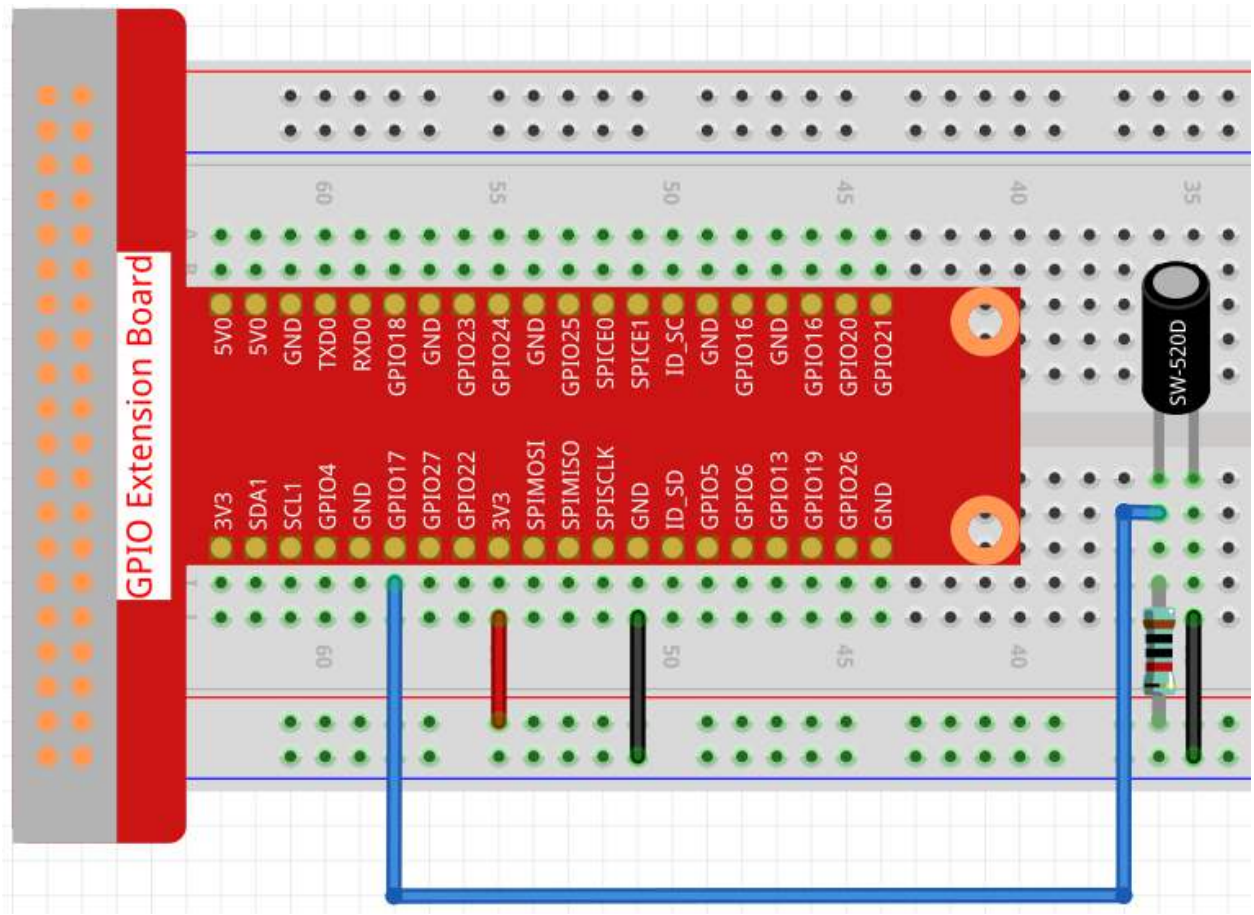


Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Tilt Switch*

Build the Circuit



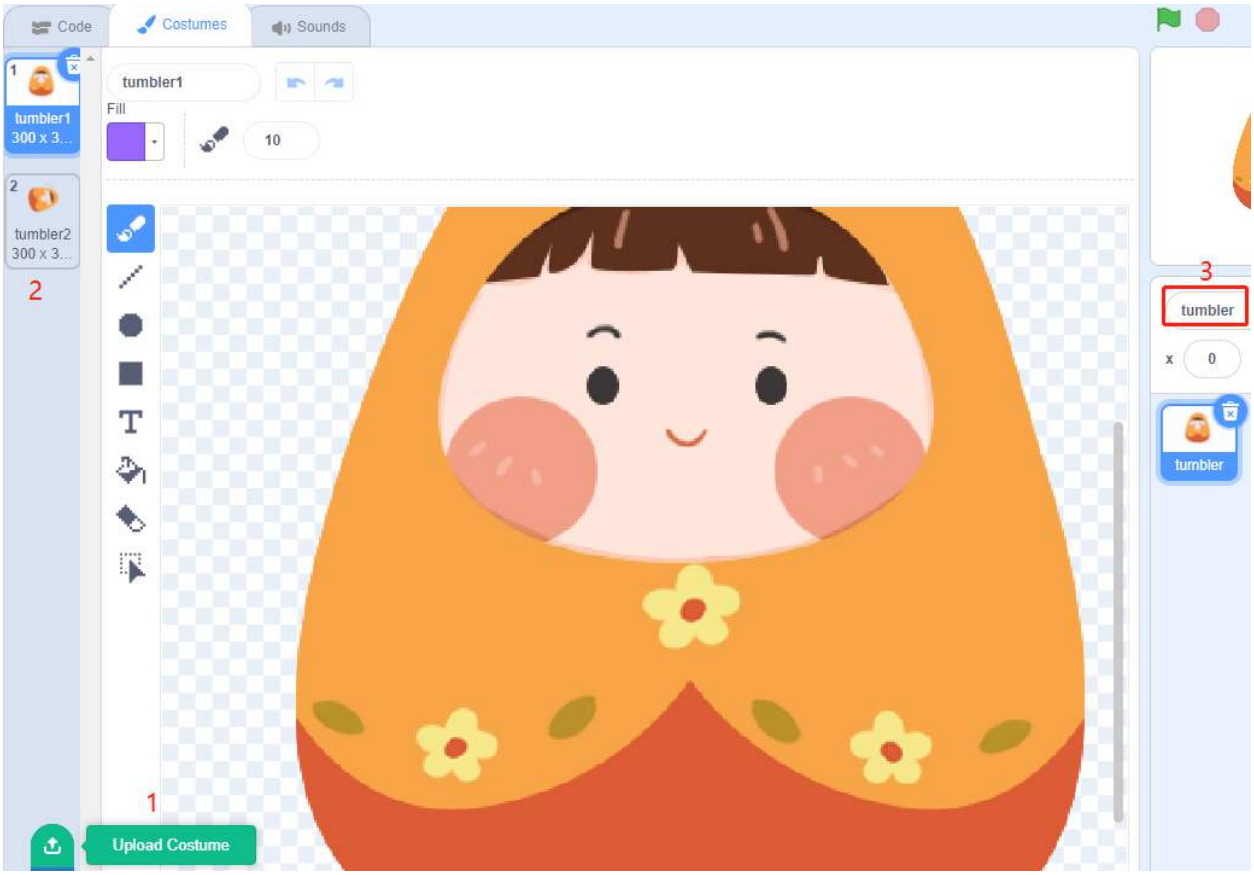
Load the Code and See What Happens

Load the code file (`1.3_tumbler.sb3`) to Scratch 3.

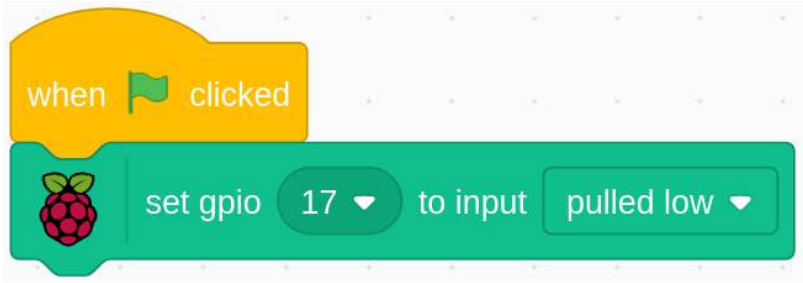
When the tilt switch is placed upright, the tumbler is standing. If you tilt it, the tumbler will also fall. Place it upright again, and the tumbler will stand up again.

Tips on Sprite

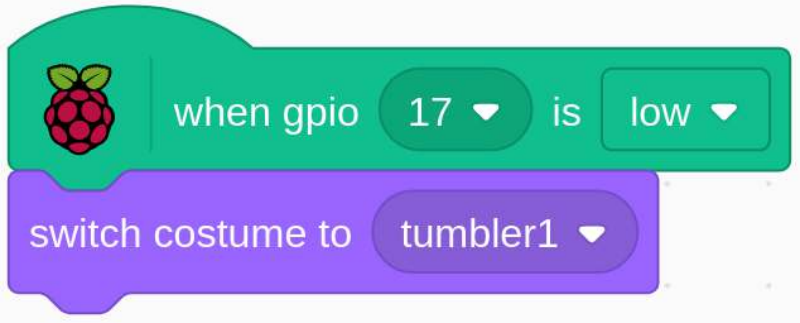
Select Sprite1 and click **Costumes** in the top left corner; upload **tumbler1.png** and **tumbler2.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **tumbler**.



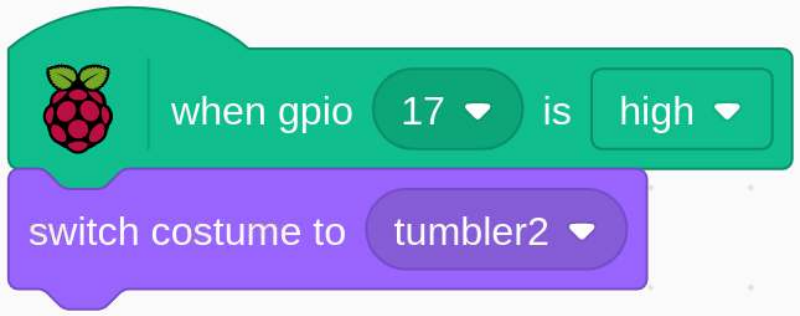
Tips on Codes



When the green flag is clicked, the initial state of gpio17 is set to low.



When pin17 is low (the tilt switch is placed upright), we switch the tumbler sprite's costume to tumbler1 (upright state).

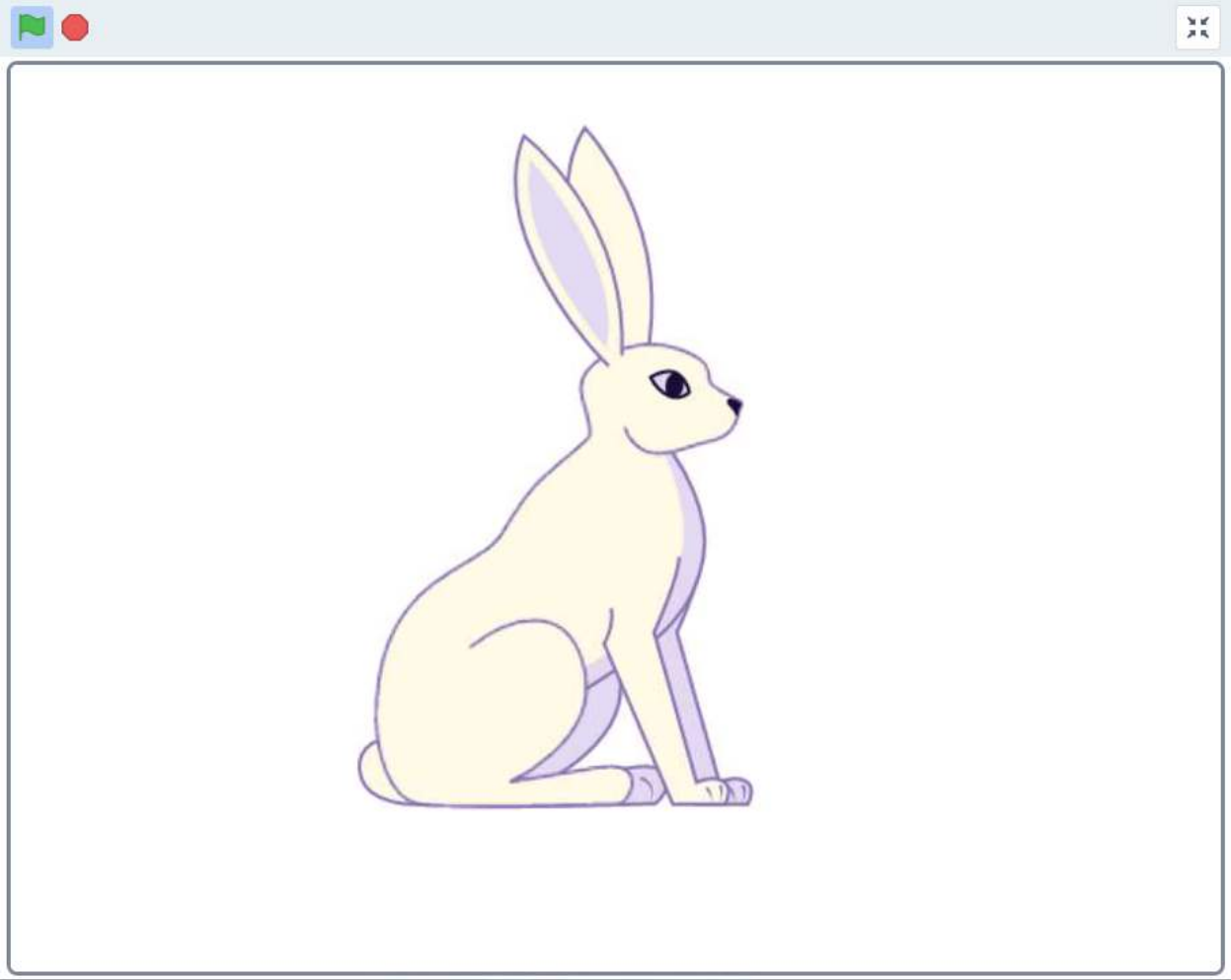


When pin17 is high (tilt switch is tilted), switch the tumbler sprite's costume to tumbler2 (tilt state).

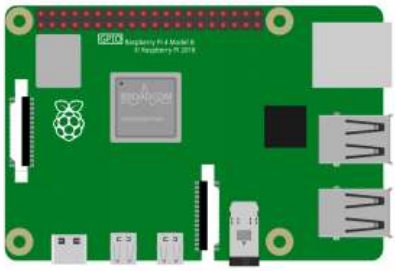
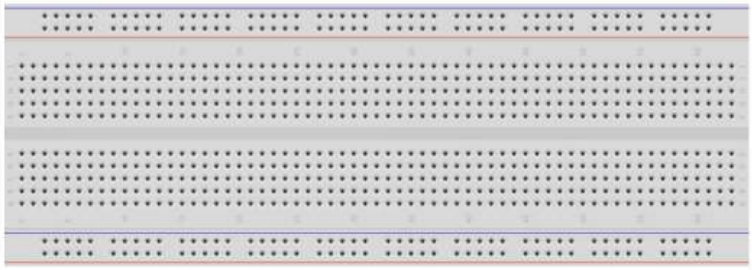
10.2.4 1.4 Hare

Today, we will use Button, Raspberry Pi and Scratch to create a hare with various changes!

When we press the first button, the hare in the stage area will change its body color; when we press the second button, the hare will change its body size; when we press the third button, the hare will take a step forward.

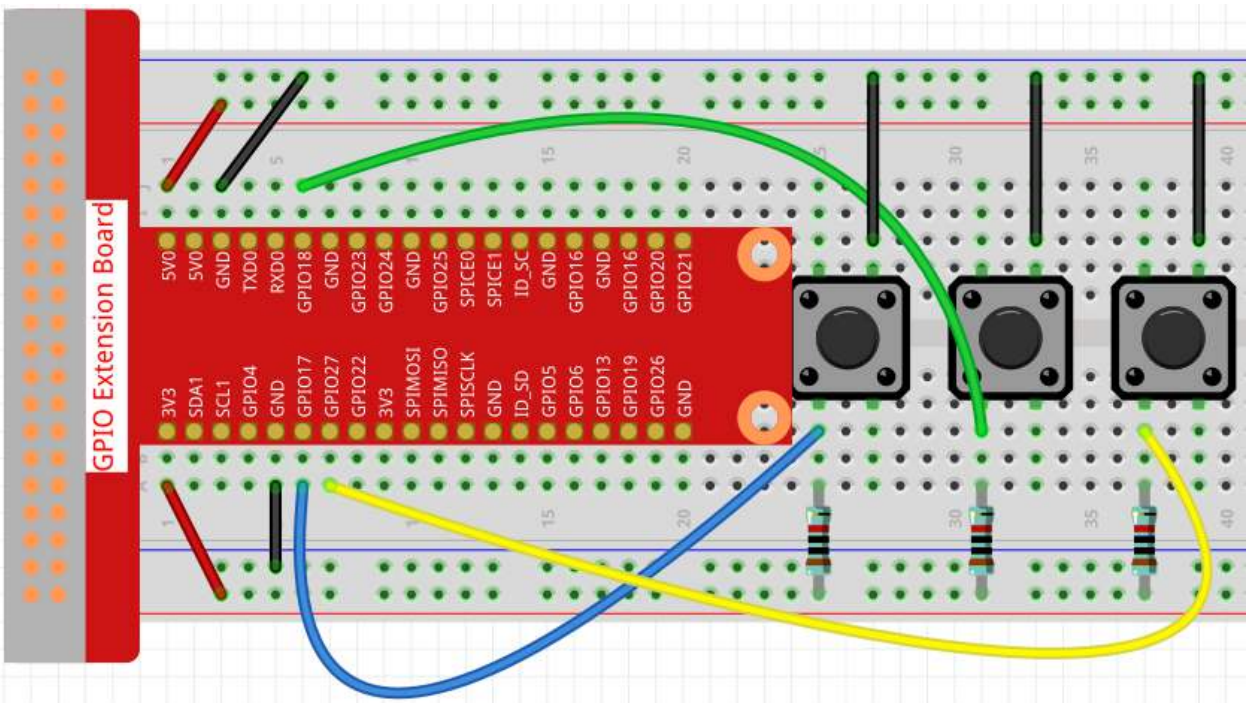


Required Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 </td><td>
3* Resistor 10KΩ
 </td></tr>
<tr><td>1 * 40-pin Cable
 </td><td>
Several Jumper Wires
 </td></tr>
<tr><td>1 * Breadboard
  </td><td>3 * Button
 </td></tr>
</table>

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Button*

Build the Circuit



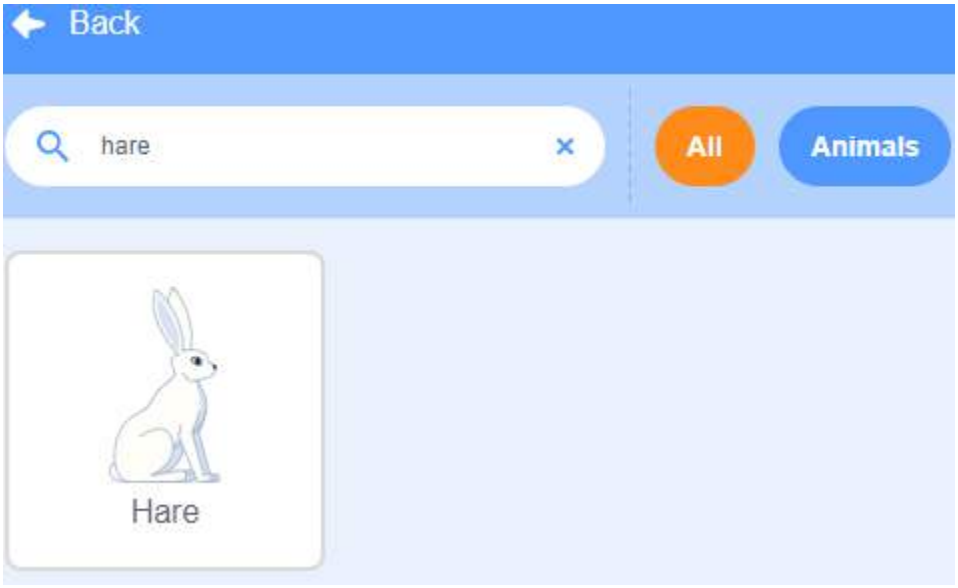
Load the Code and See What Happens

Load the code file (`1.4_hare.sb3`) into Scratch 3.

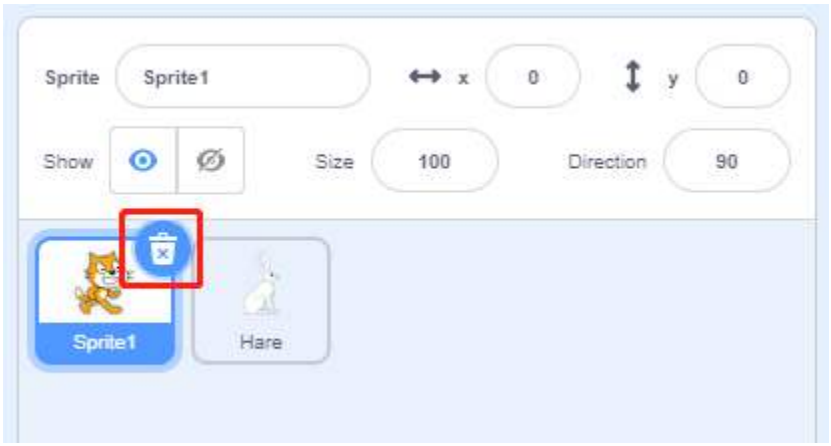
Now you can try to press each of the 3 buttons to see how the Hare on the stage will change.

Tips on Sprite

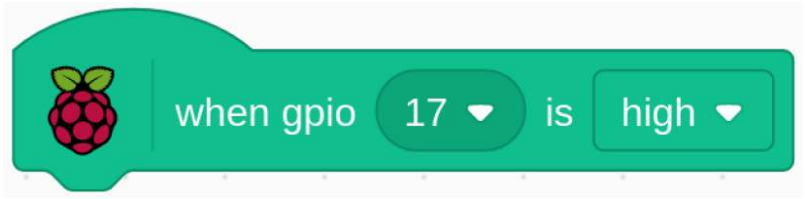
Click the **Choose a Sprite** button in the lower right corner of the sprite area, enter **Hare** in the search box, and then click to add it.



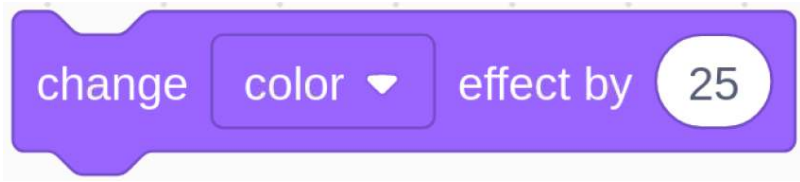
Delete Sprite1.



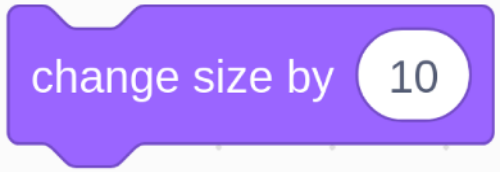
Tips on Codes



This is an event block that is triggered when the level of GPIO17 is high, which means that the button is pressed at that moment.

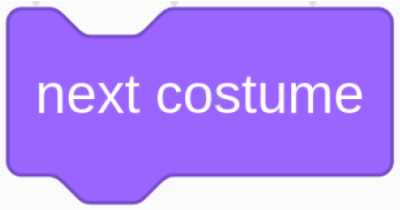


This is a block to change the color of **Hare**, the range of the value is 0 ~ 199, beyond 199 will change from 0 again.



This is a block used to change the size for sprite, the higher the value, the larger the sprite.

Note: The sprite is also not infinitely large, and its maximum size is related to the original image size.

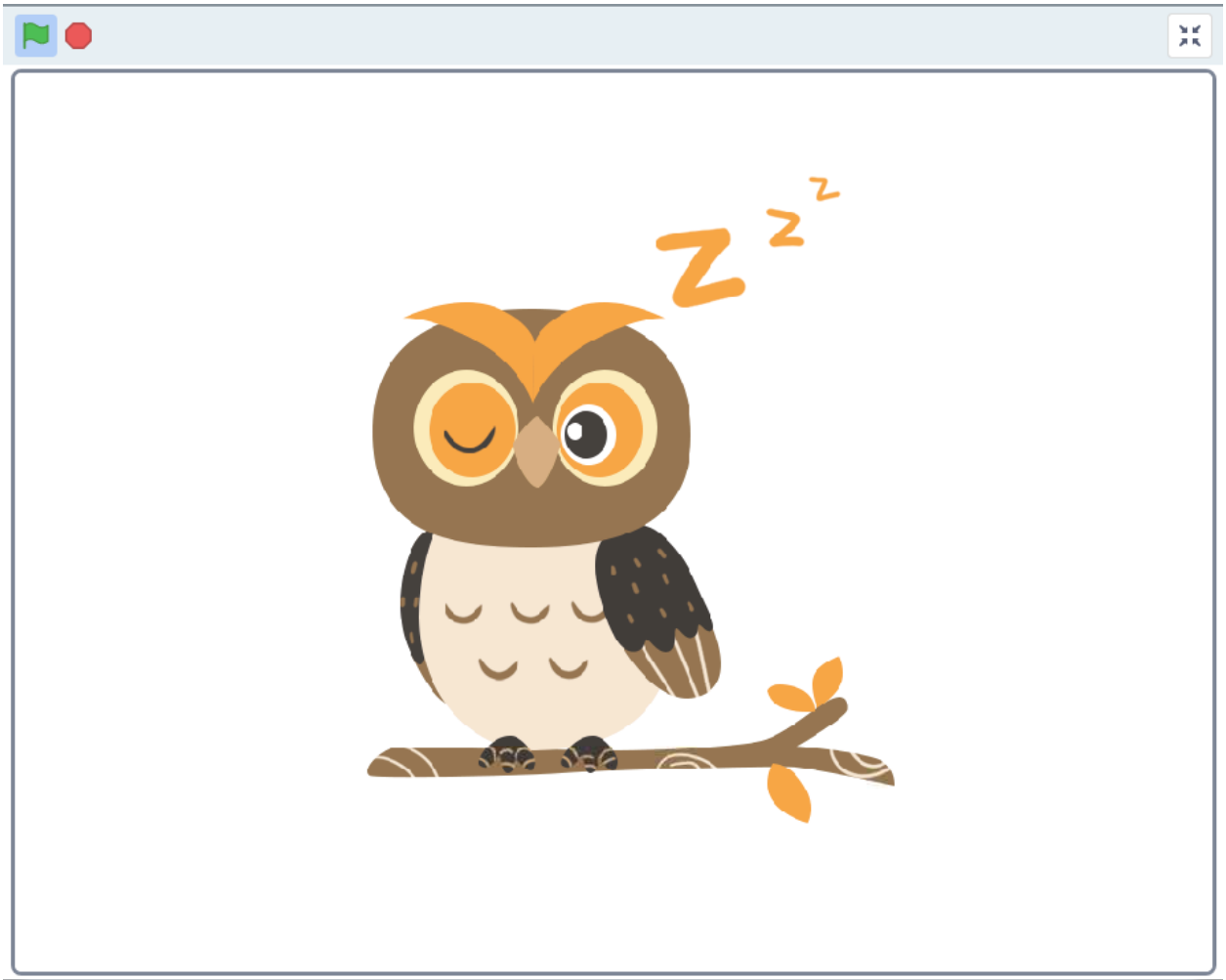


This is a block that switches sprite costumes, and when **Hare**'s costume keeps switching, it does a series of coherent actions. For example, in this project, make **Hare** take a step forward.

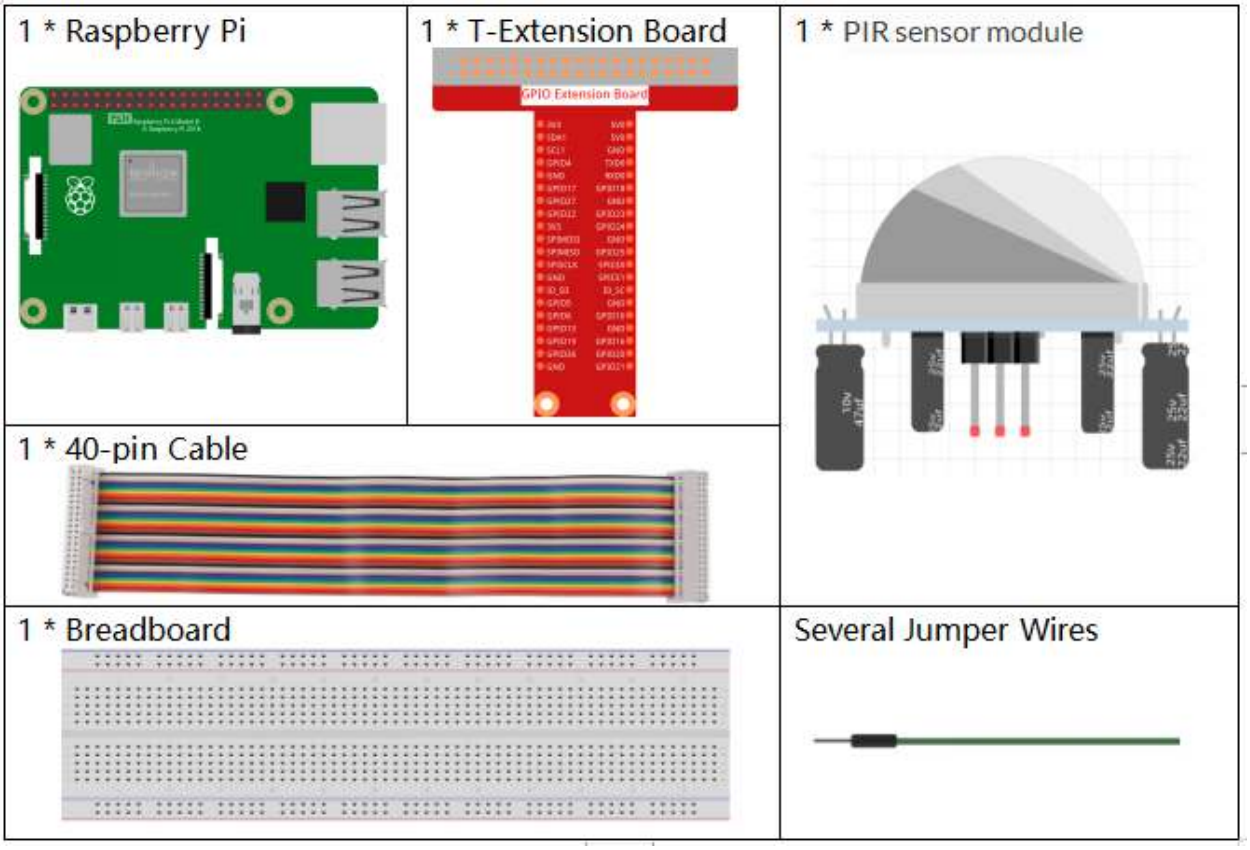
10.2.5 1.5 Wake up the Owl

Today we are going to play a game of waking up the owl.

When someone approaches the PIR sensor module, the owl will wake up from sleep.

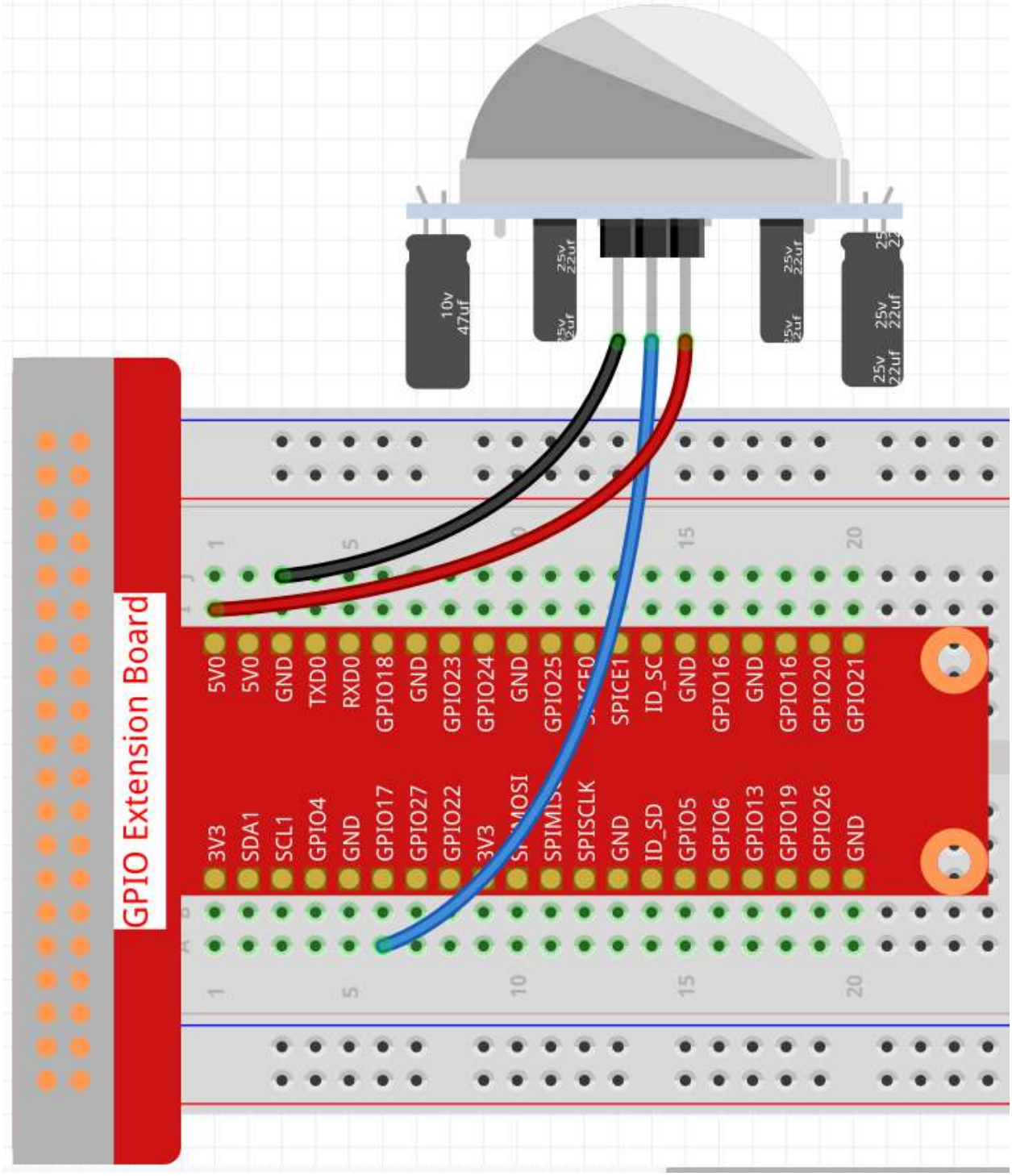


Required Components



- *GPIO Extension Board*
- *Breadboard*
- *PIR Motion Sensor Module*

Build the Circuit



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.

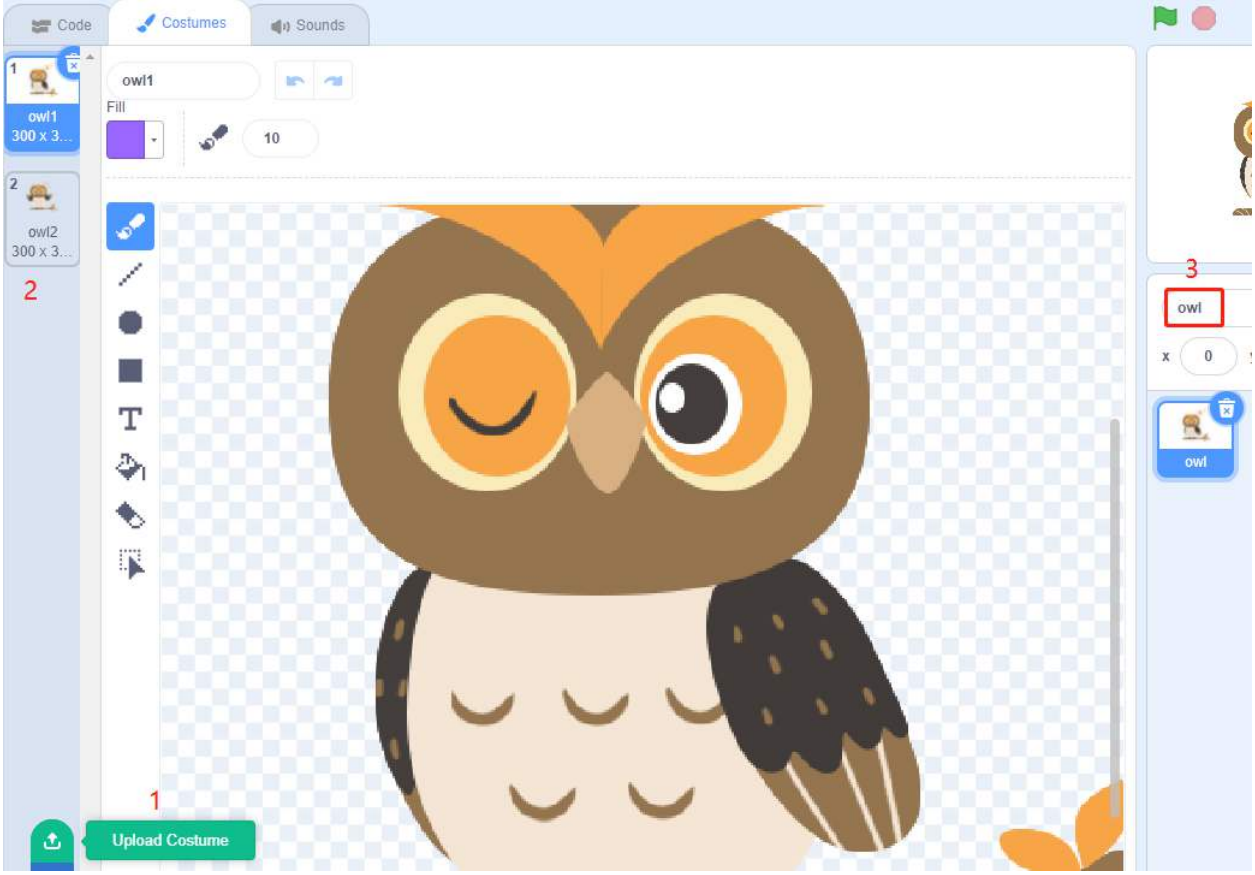
Load the Code and See What Happens

Load the code file (`1.5_wake_up_the_owl.sb3`) to Scratch 3.

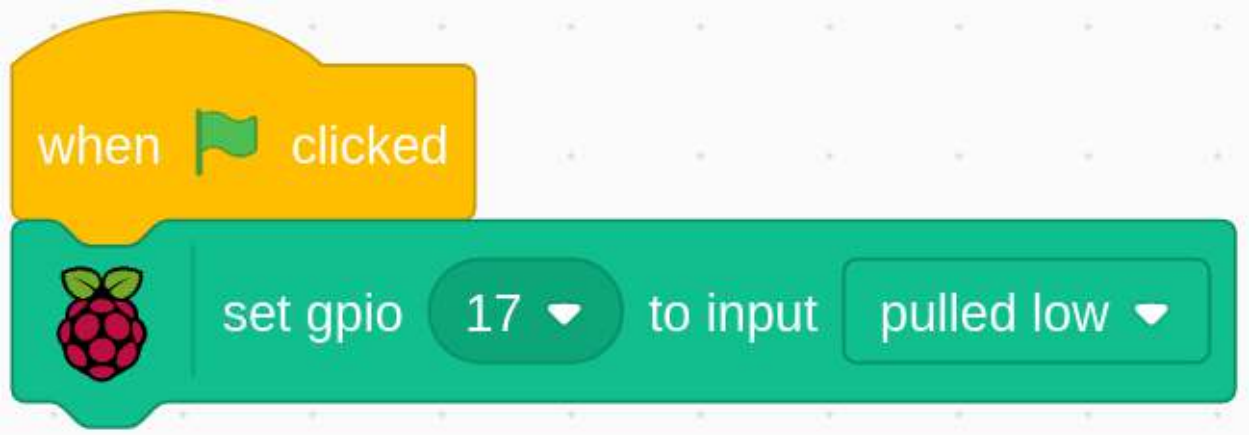
When you approach the PIR sensor module, you will see the owl on the stage area open its wings and wake up, and when you leave, the owl will go back to sleep again.

Tips on Sprite

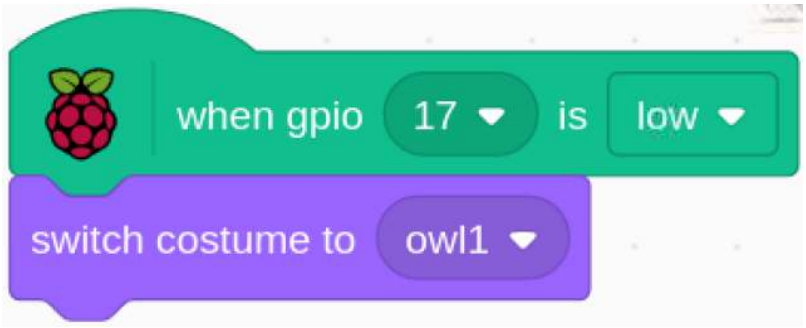
Select Sprite1 and click **Costumes** in the top left corner; upload **owl1.png** and **owl2.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **owl**.



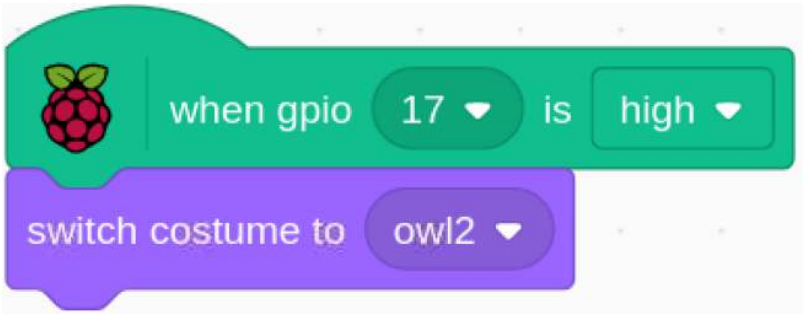
Tips on Codes



When the green flag is clicked, the initial state of gpio17 is set to low.



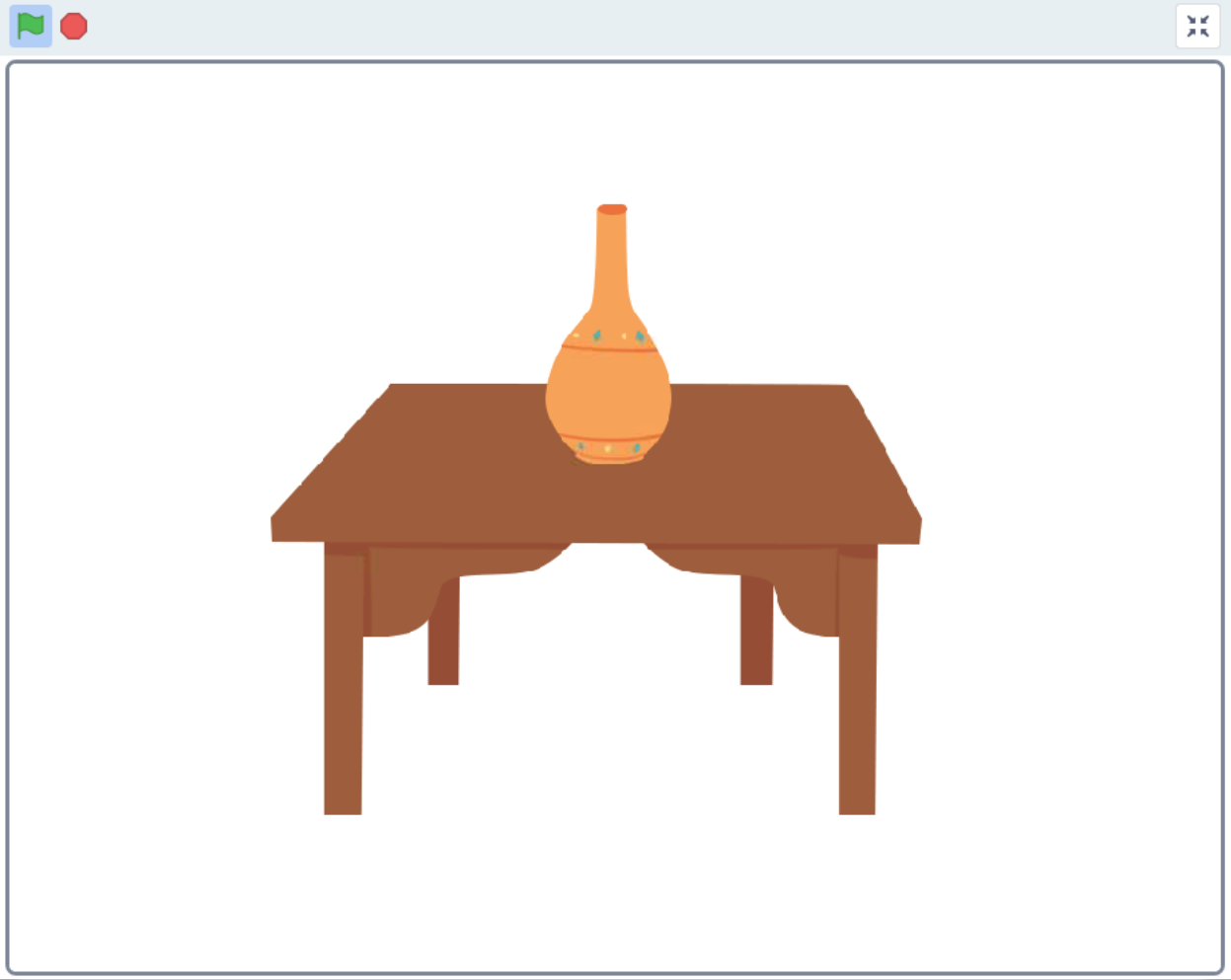
When pin17 is low (no one is approaching), switch the costume of the owl sprite to owl1 (sleeping state).



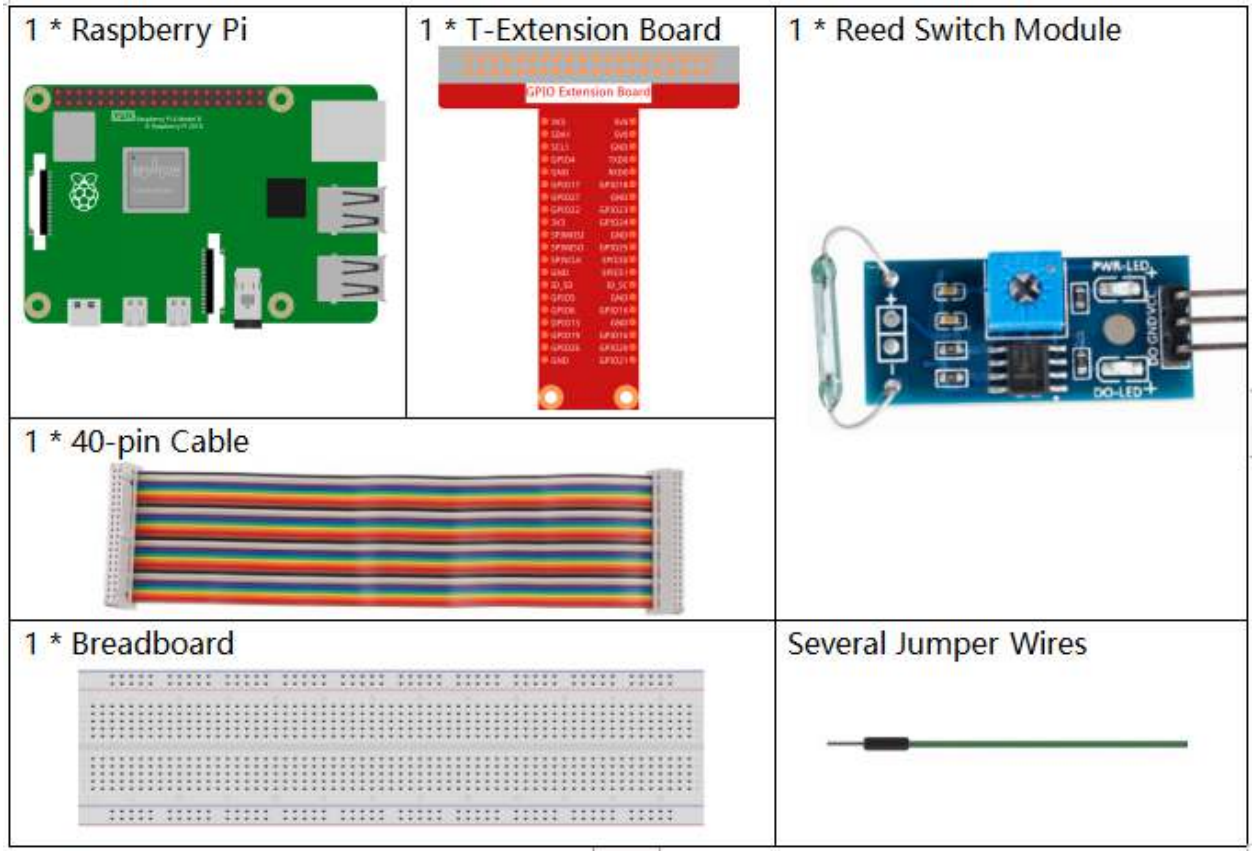
When pin17 is high (someone is approaching), we switch the costume of owl sprite to owl2 (wake up state).

10.2.6 1.6 Vanishing Vase

Now let's do a little magic trick, do nothing, and then the vase somehow disappears.

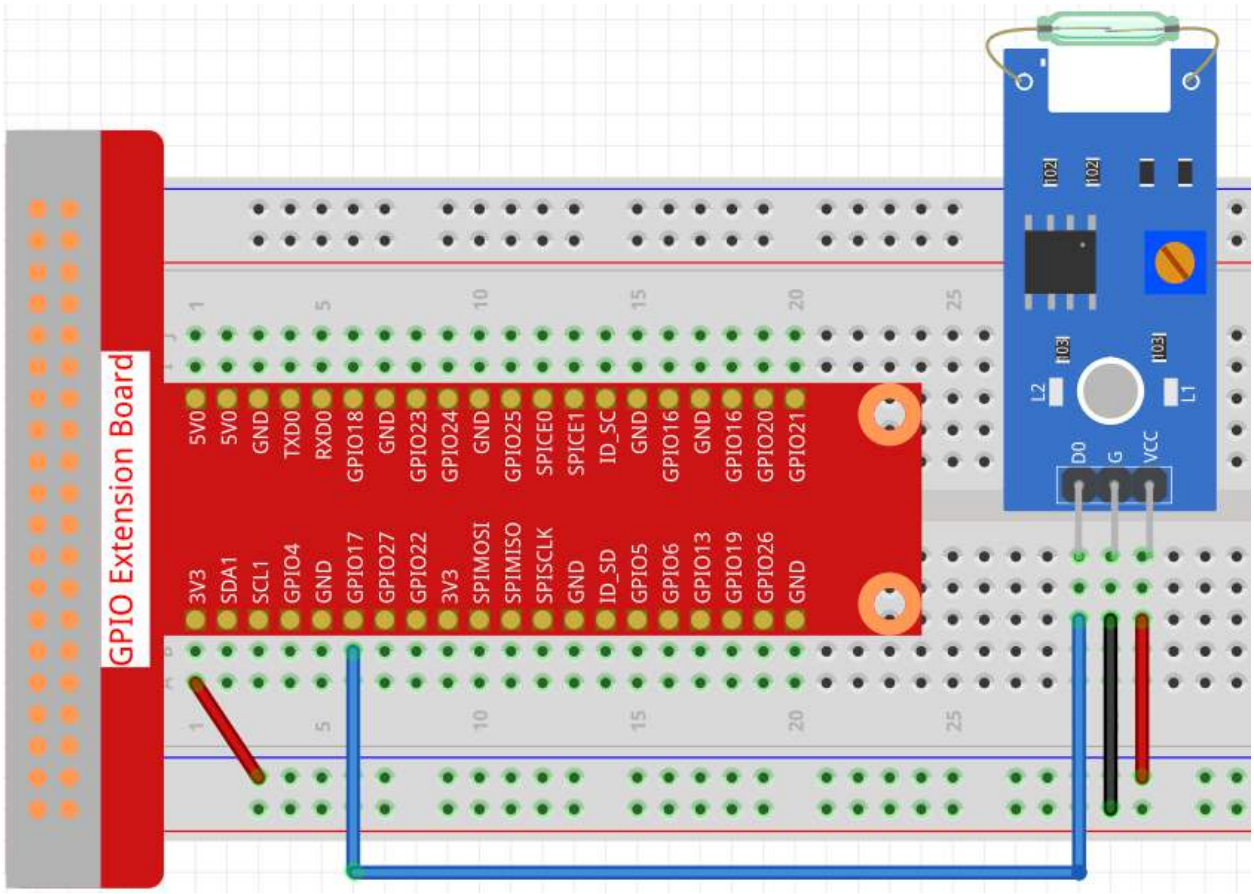


Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Reed Switch Module*

Build the Circuit



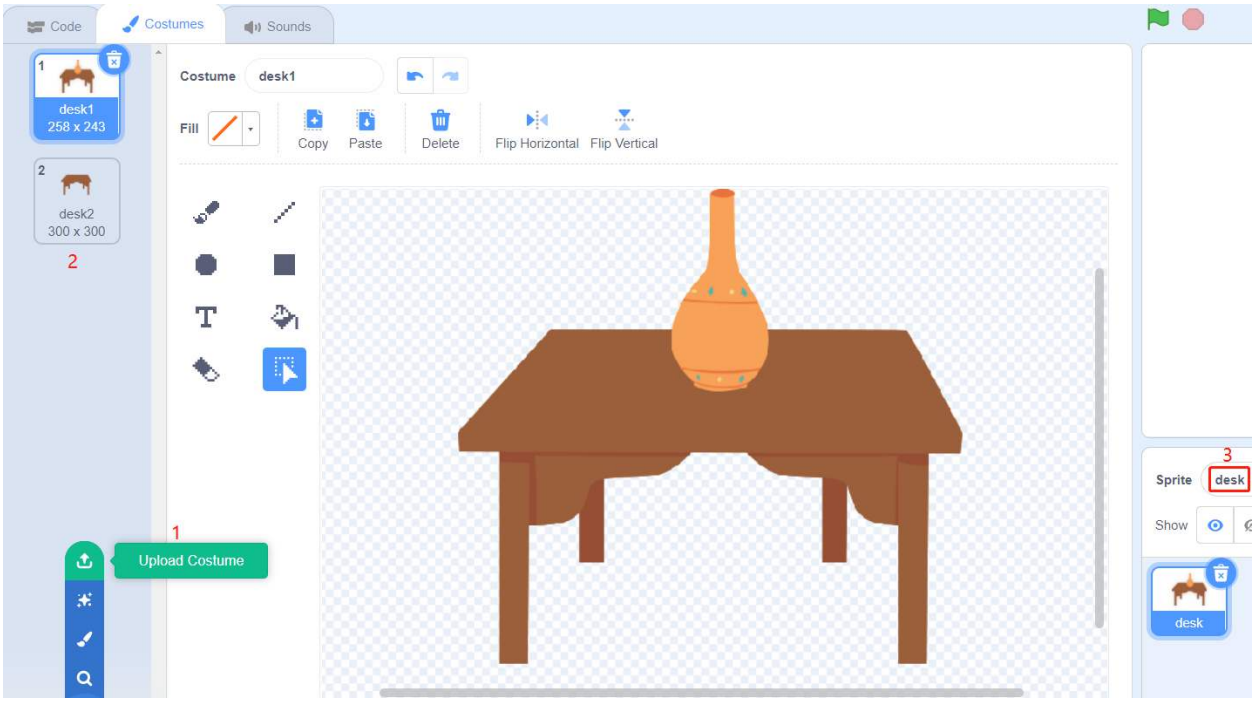
Load the Code and See What Happens

Load the code file (`1.6_vanishing_vase.sb3`) to Scratch 3.

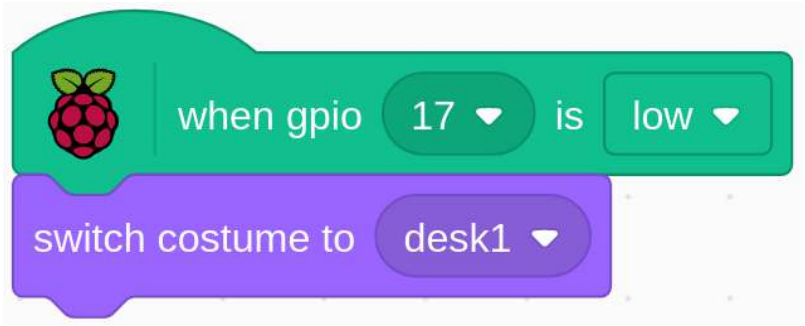
When you use a magnet near the reed switch module, a vase will appear on the stage, take away the magnet and the vase will disappear.

Tips on Sprite

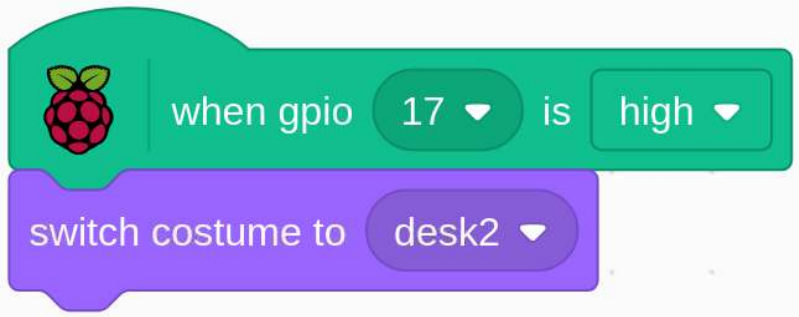
Select Sprite1 and click **Costumes** in the top left corner; upload **desk1.png** and **desk2.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **desk**.



Tips on Codes



When the magnet is close to the reed switch module, gpio17 is low, and the costume of the **desk** sprite is switched to **desk1** (the vase is still on the desk).

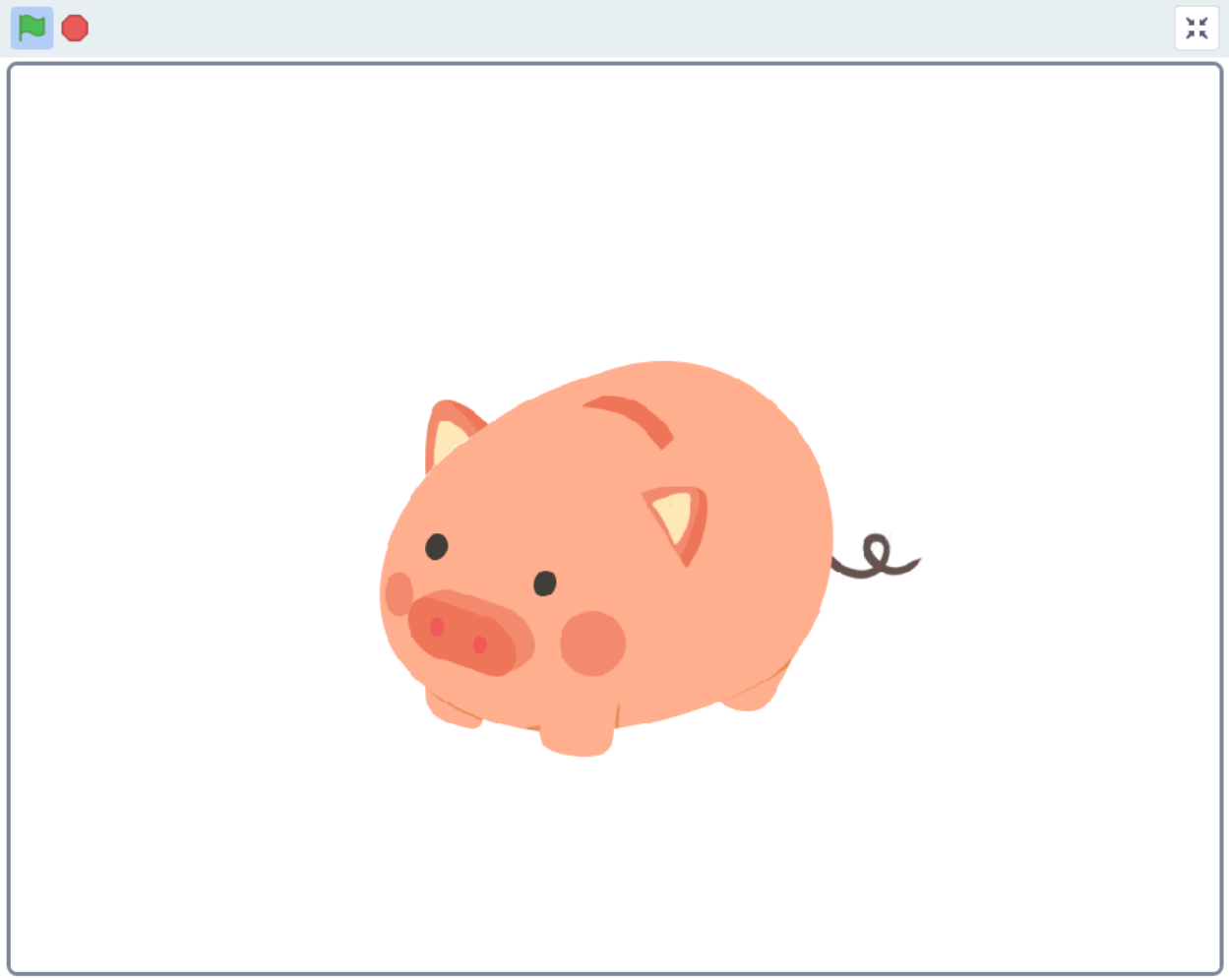


After taking away the magnet, gpio17 is high, at this time the costume of the **desk** sprite is switched to **desk2** (only one desk).

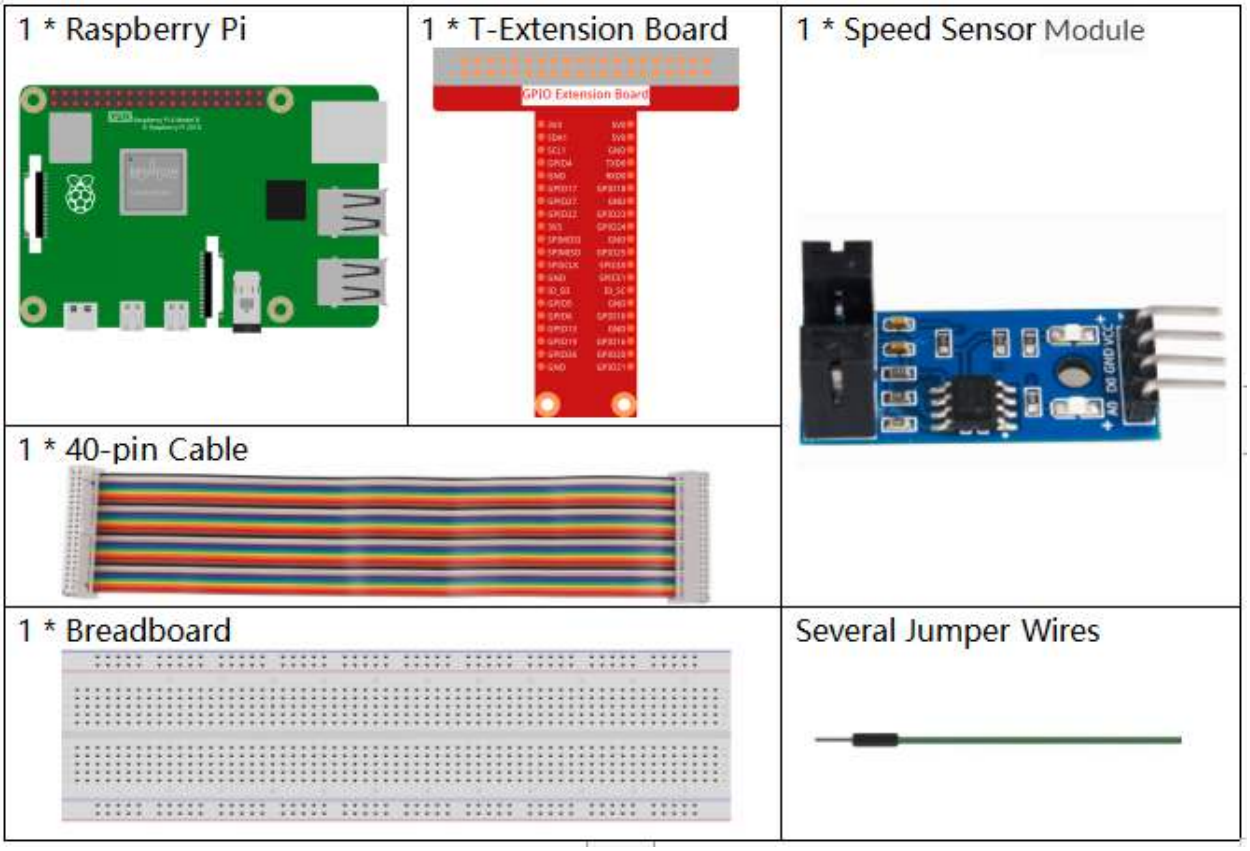
10.2.7 1.7 Piggy Bank

In this project we will use Speed sensor module, Raspberry Pi and Scratch to make a Piggy Bank.

Place a piece of paper in the middle of the Speed sensor module and you will see a coin fall into the Piggy Bank on the stage.

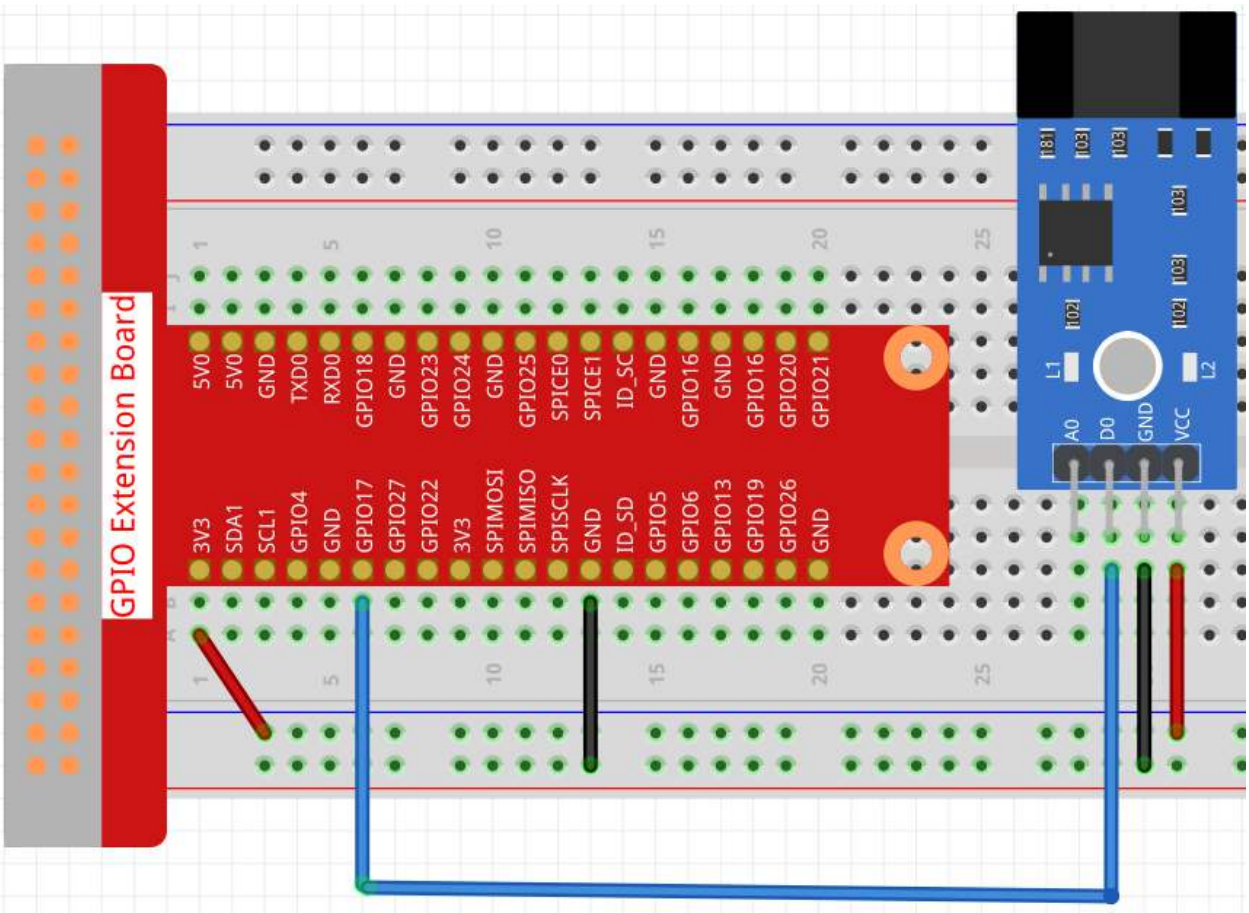


Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Speed Sensor Module*

Build the Circuit



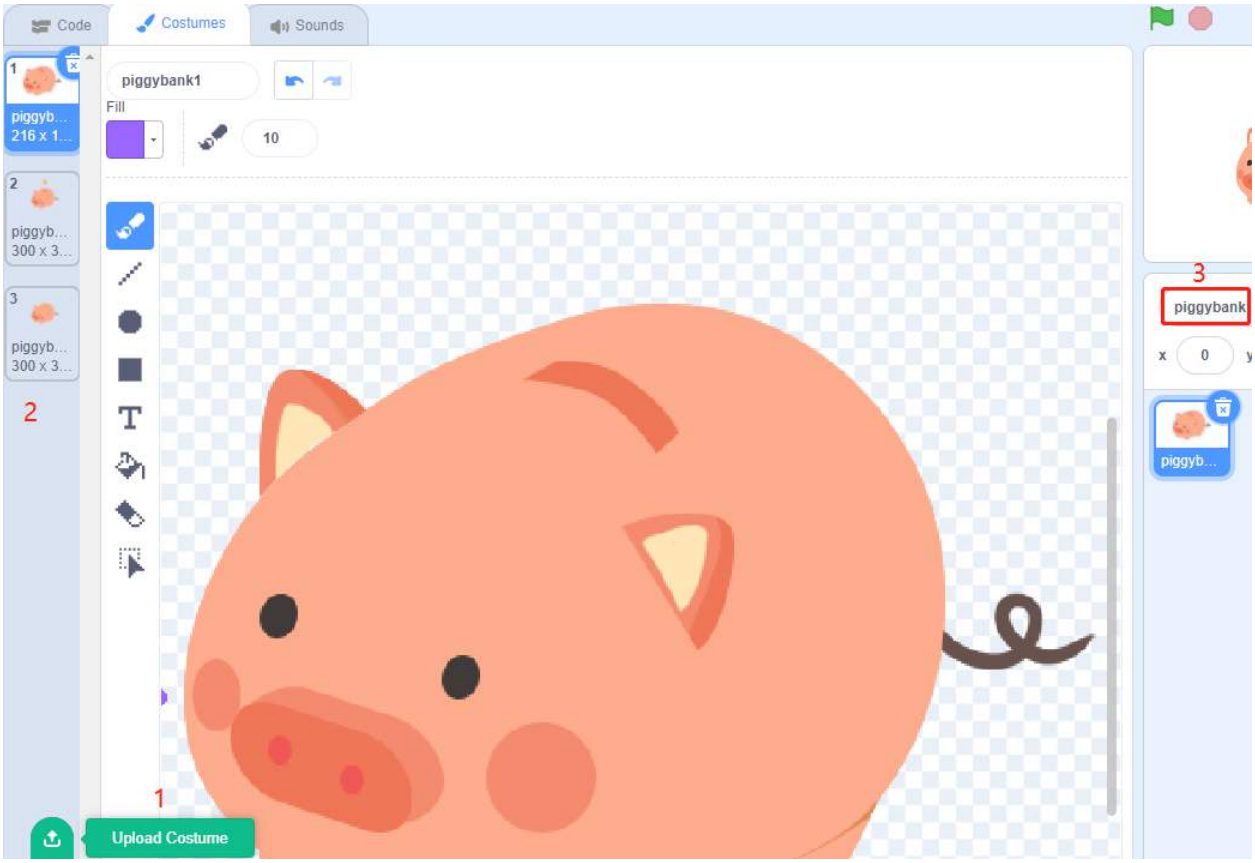
Load the Code and See What Happens

Load the code file (`1.7_piggy_bank.sb3`) to Scratch 3.

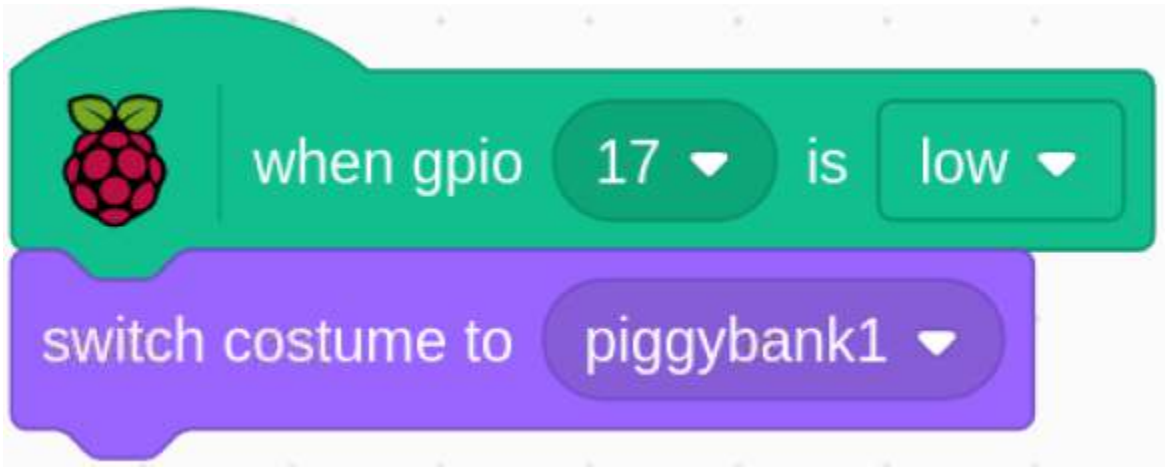
The 2 terminals in the middle of the speed sensor, one is to send light, one is to receive light; if you put a piece of paper in the middle to isolate the light transmission, thus the speed sensor will output a high level. At this point Scratch receives the high level, then switch the costumes of the sprite and you will see a coin fall into the Piggy Bank on the stage.

Tips on Sprite

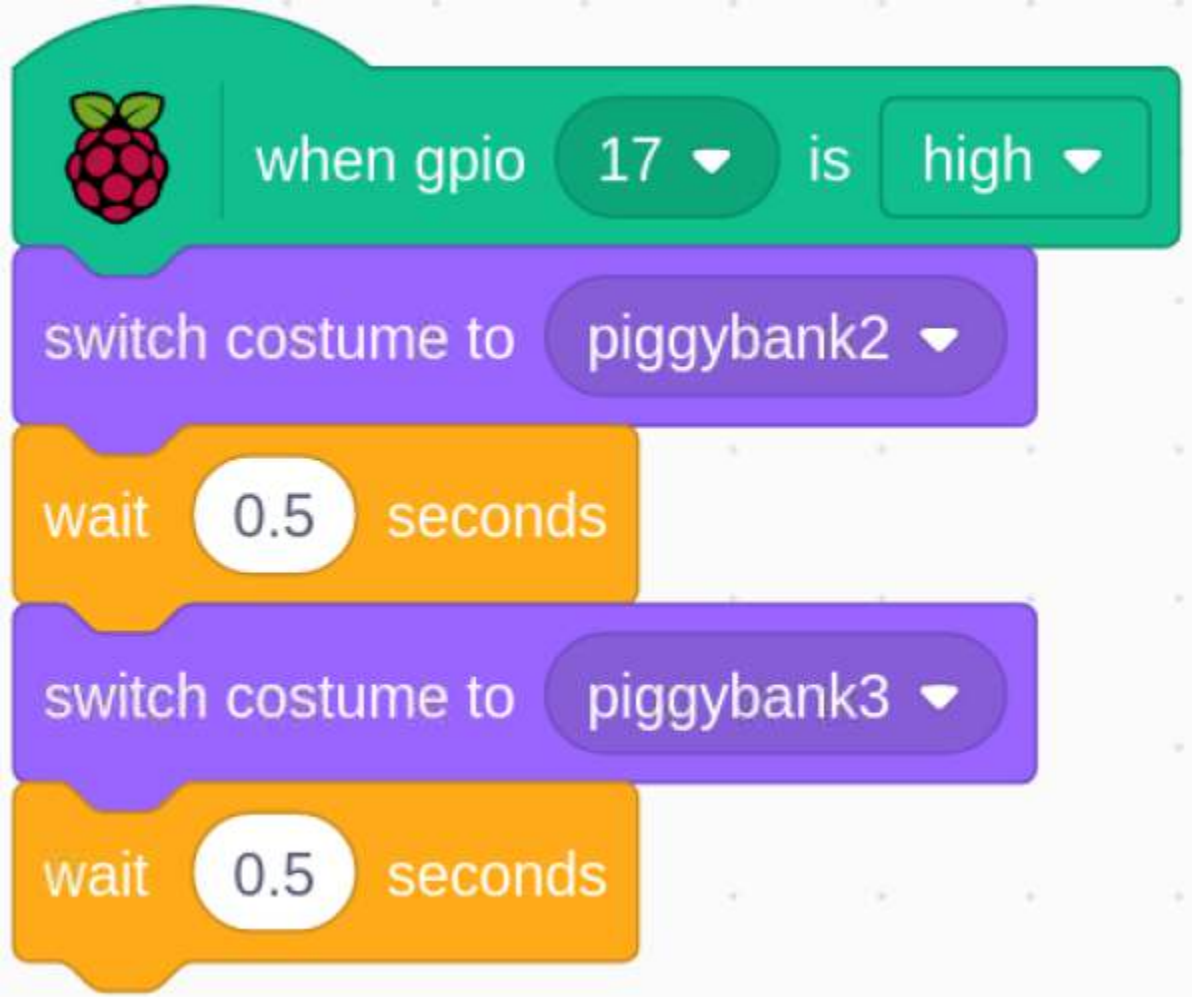
Select Sprite1 and click **Costumes** in the top left corner; upload **piggybank1.png**, **piggybank2.png** and **piggybank3.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **piggybank**.



Tips on Codes



When pin17 is low (no coins are put in), switch the sprite's costume to **piggybank1**.



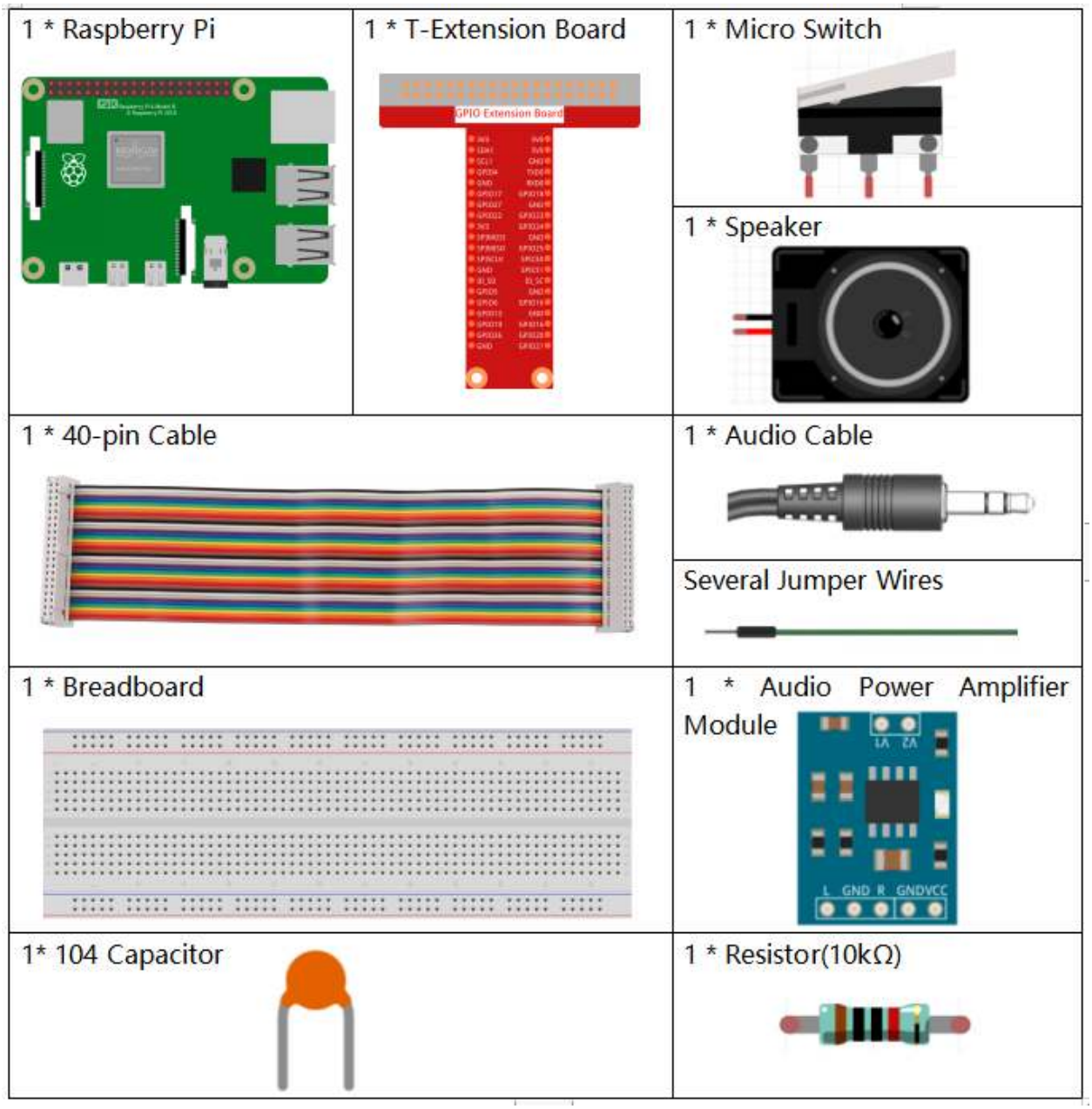
When pin17 is high (a coin is put in), switch the sprite's costume to **piggybank2**, and after 0.5s switch to **piggybank3**, so that we can see a coin falling into the Piggy Bank on the stage.

10.2.8 1.8 Service Bell

Today, we will use Micro Switch, speakers, audio amplifier module, Raspberry Pi and scratch to make a service bell.

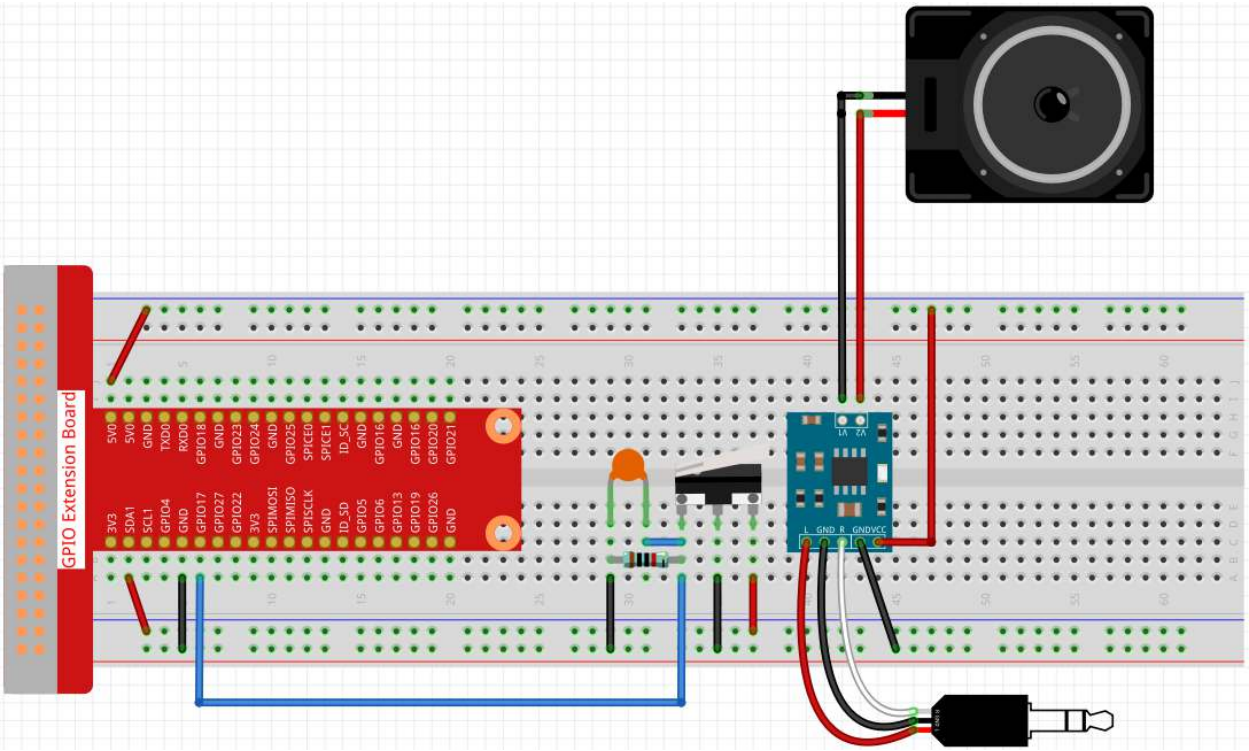
Tap the Micro Switch to make the service bell sound.

Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Micro Switch*
- *Capacitor*
- *Audio Module and Speaker*

Build the Circuit



Load the Code and See What Happens

Load the code file (`1.8_service_bell.sb3`) to Scratch 3.

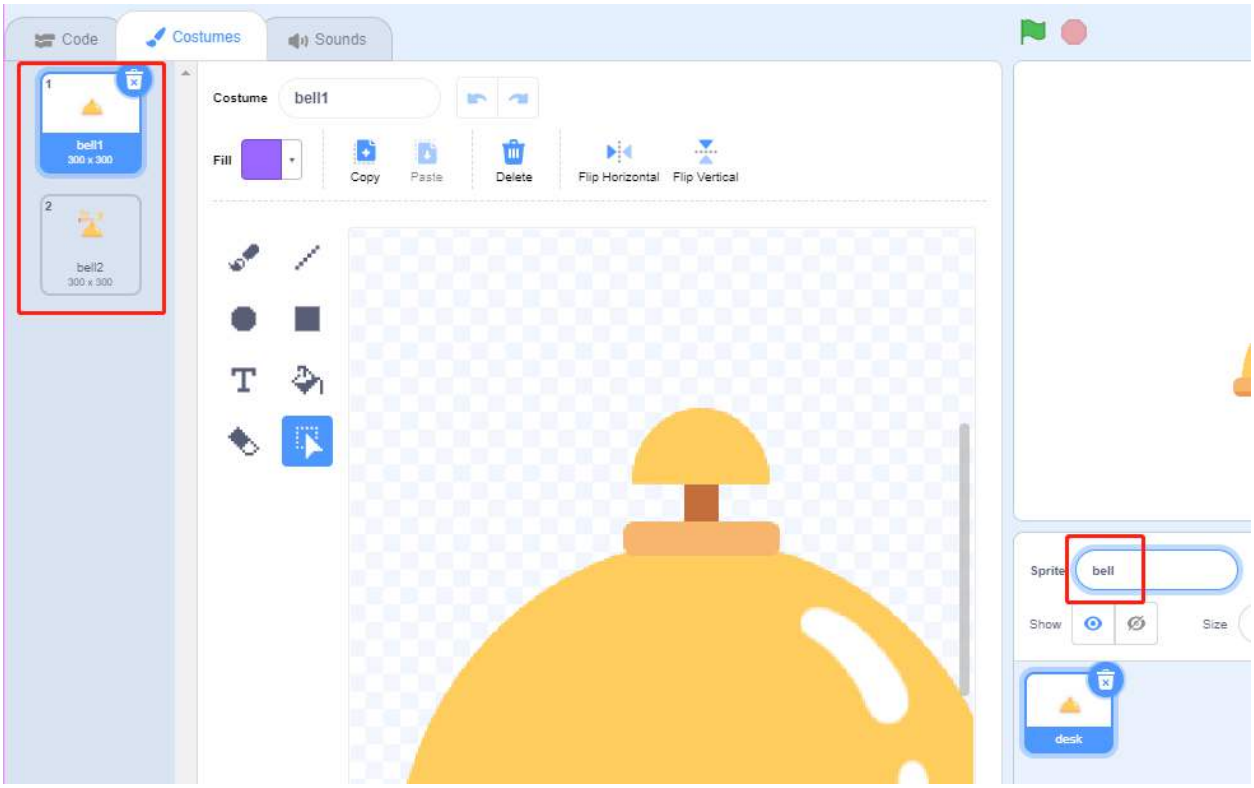
Press the micro switch and the service bell will ring once.

Note: If your Raspberry Pi is connected to a screen with speakers, it may cause no sound from this external speaker, please refer to *Change Audio Output* for the solution.

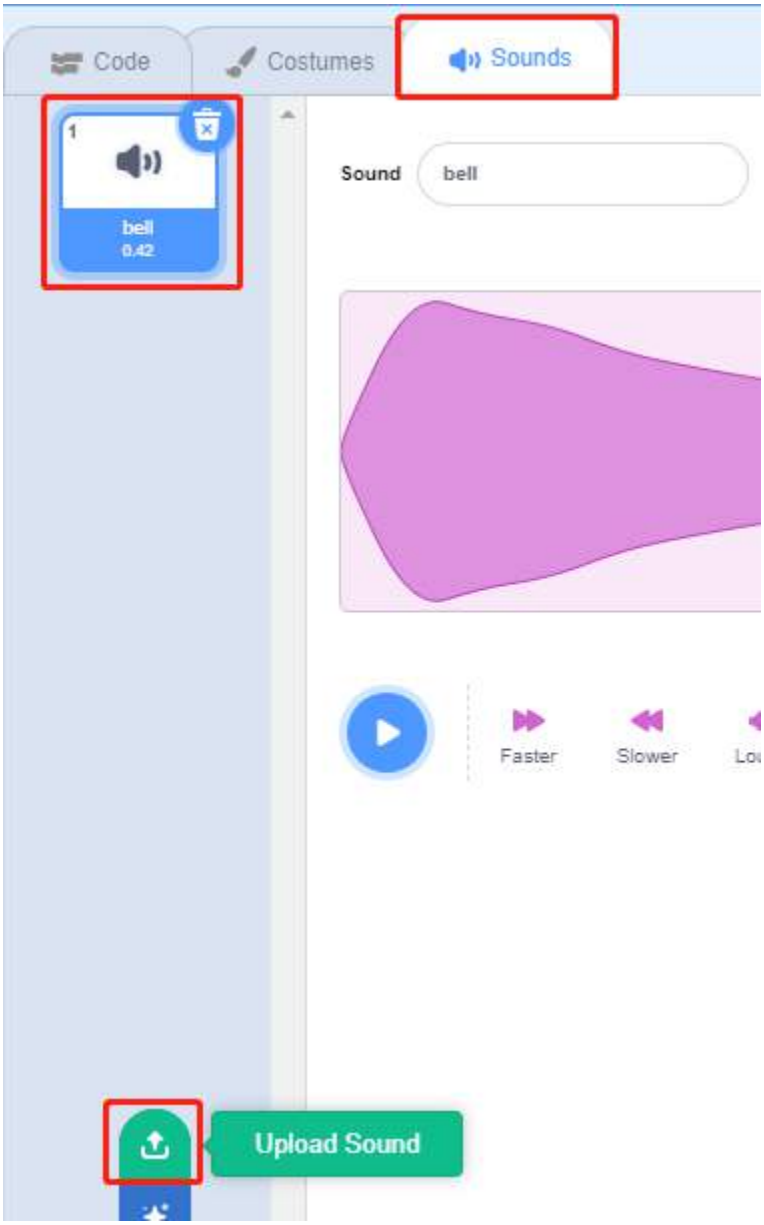
Also, if you want to adjust the volume level, please refer to *Adjust Volume*.

Tips on Sprite

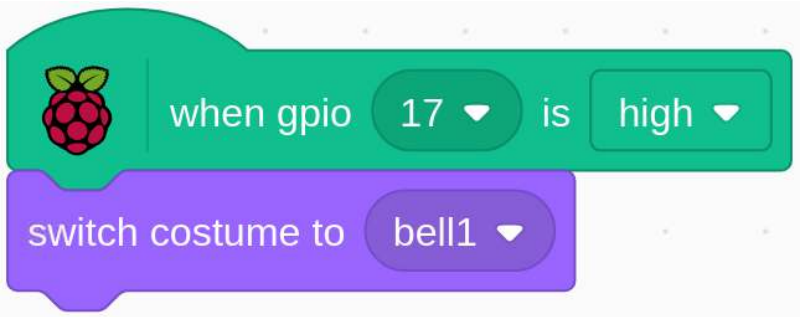
Select Sprite1 and click **Costumes** in the top left corner; upload **bell1.png** and **bell2.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **bell**.



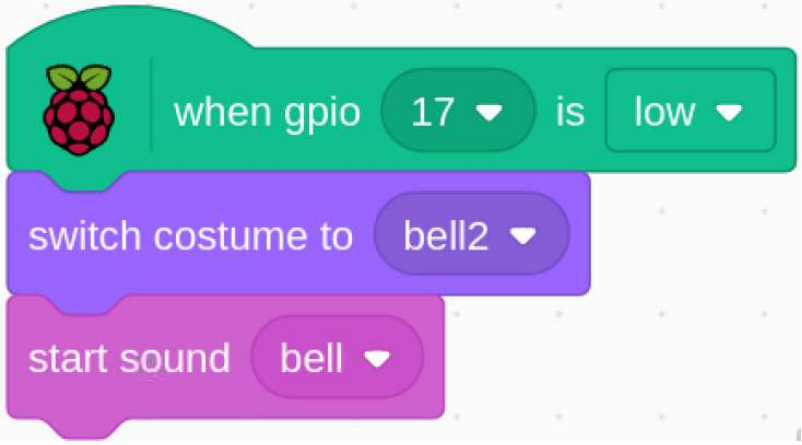
In the **Sounds** option, upload the `bell.wav` from the `home/pi/raphael-kit/scratch/sound` path to Scratch 3.



Tips on Codes



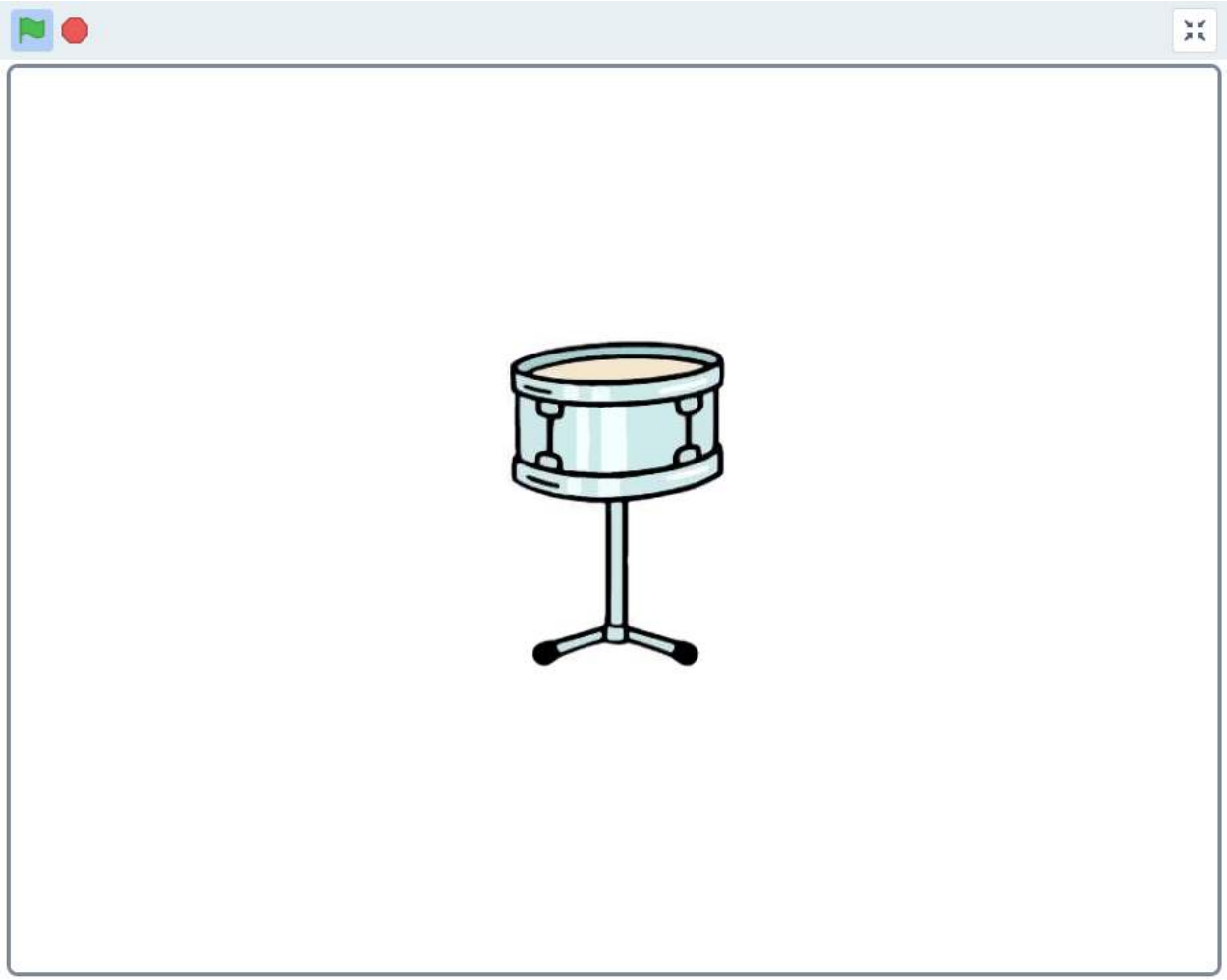
When pin17 is high (the Micro switch is not pressed), switch the costume of the **bell** sprite to **bell1** (released state).



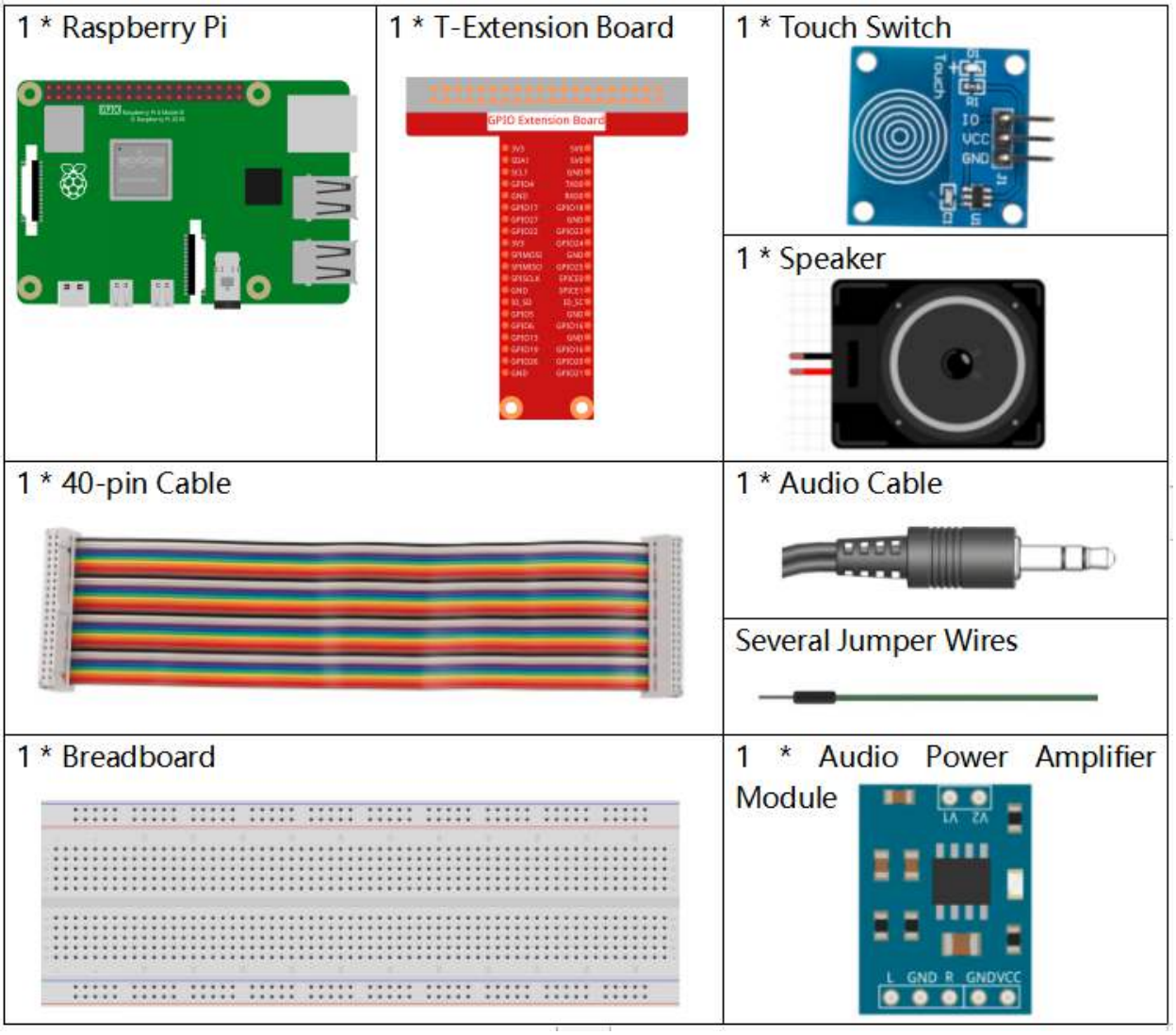
Press the micro switch, gpio17 is low level. At this time, switch the costume of the **bell** sprite to **bell2** (press state), and play a sound effect through the speaker.

10.2.9 1.9 Drumming

In this project, we play the drum with a touch switch module.

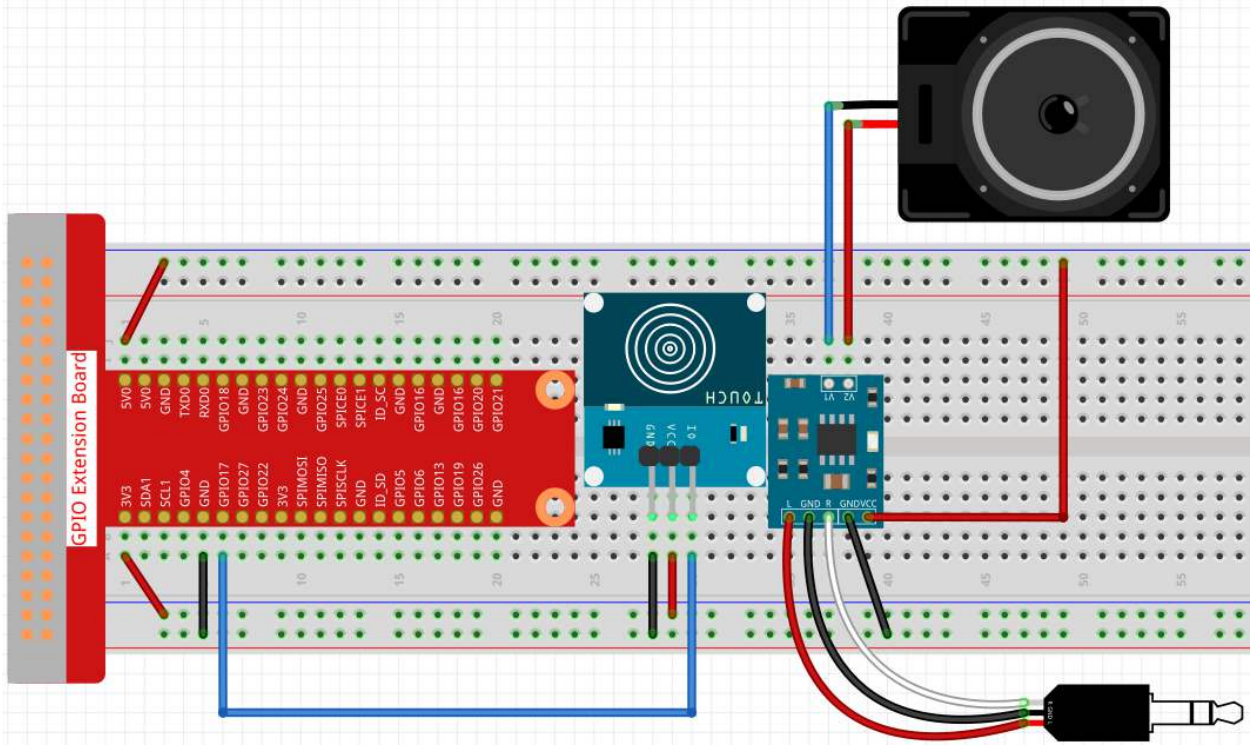


Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Touch Switch Module*
- *Audio Module and Speaker*

Build the Circuit



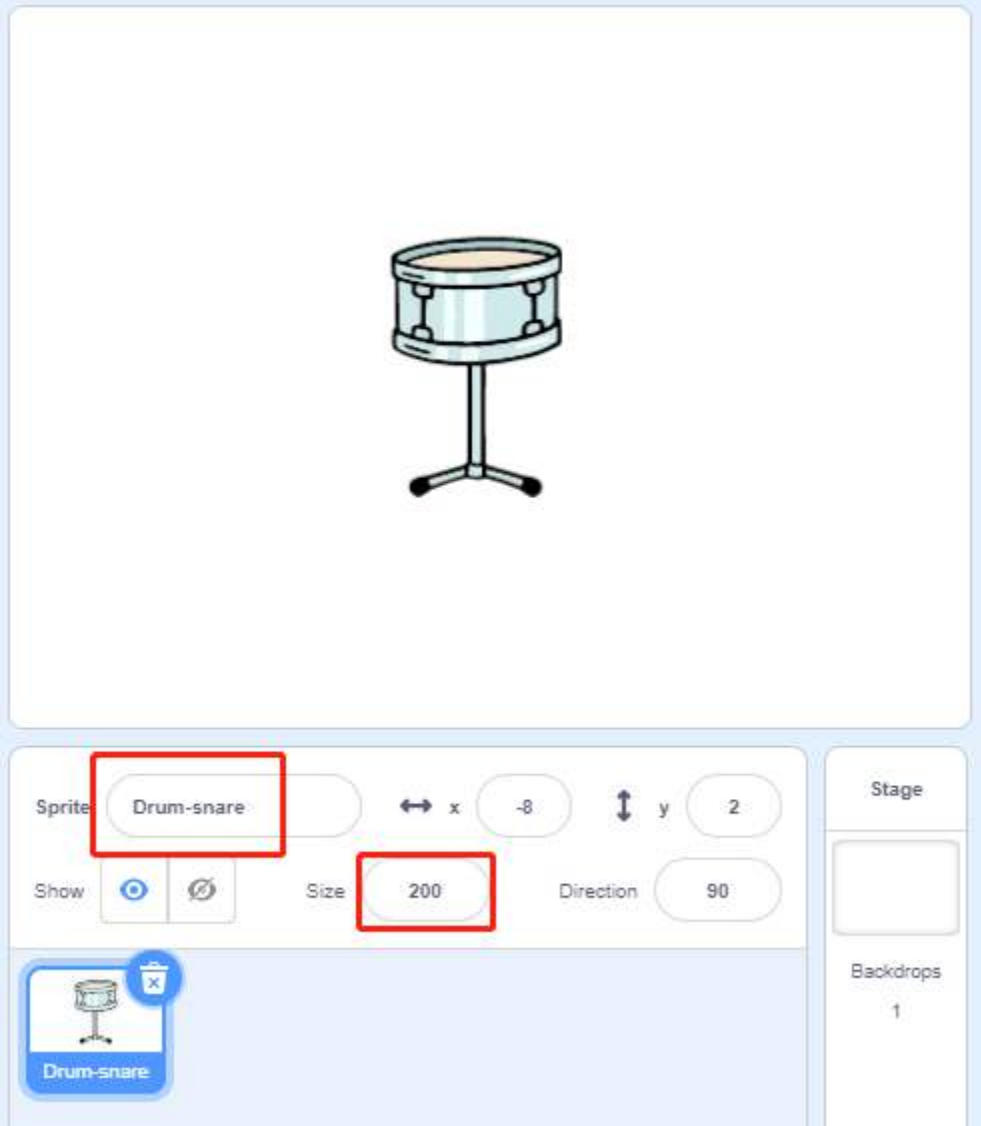
Load the Code and See What Happens

Load the code file (`1.9_drumming.sb3`) to Scratch 3.

When you tap on the touch switch module, you will hear the sound of drums coming from the speaker.

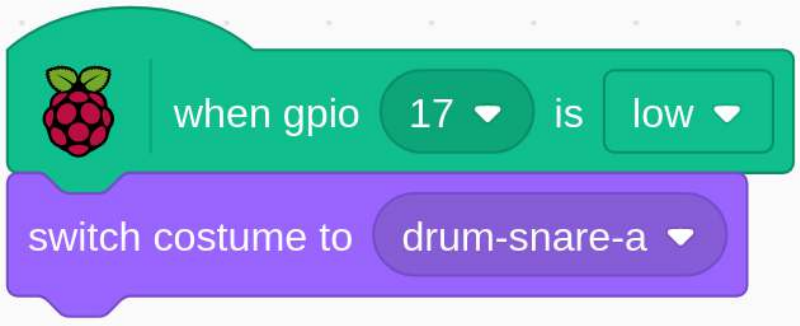
Tips on Sprite

Delete the default sprite, then find the **Drum-snare** sprite and add it, and change the size to 200.

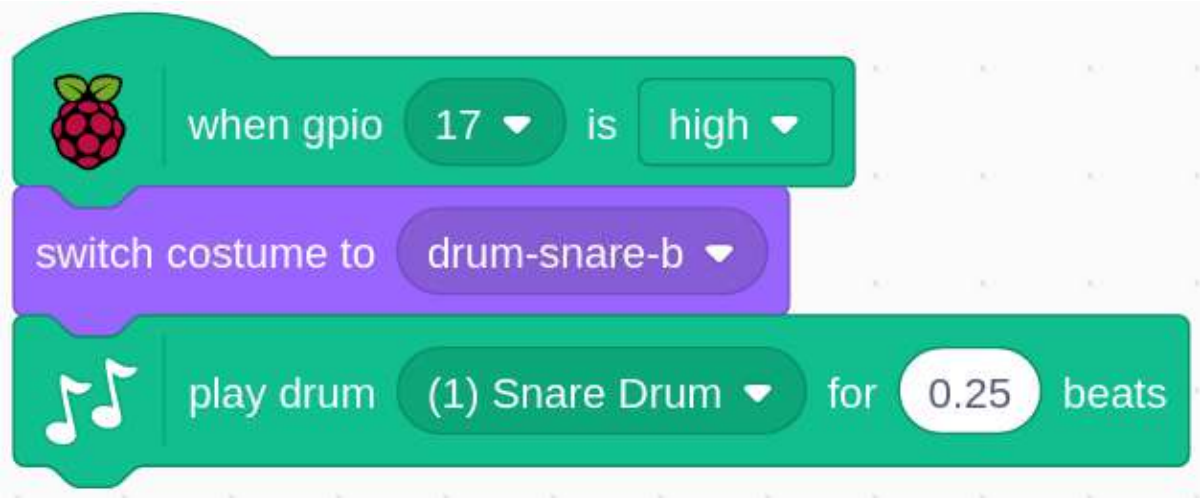


Scratch has a **Music** extension to play instruments and drums, now add it via the **Add Extension** button.

Tips on Codes



When pin17 is low (not tapped on the touch switch module), switch the **Drum-snare** sprite costume to **drum-snare-a**.

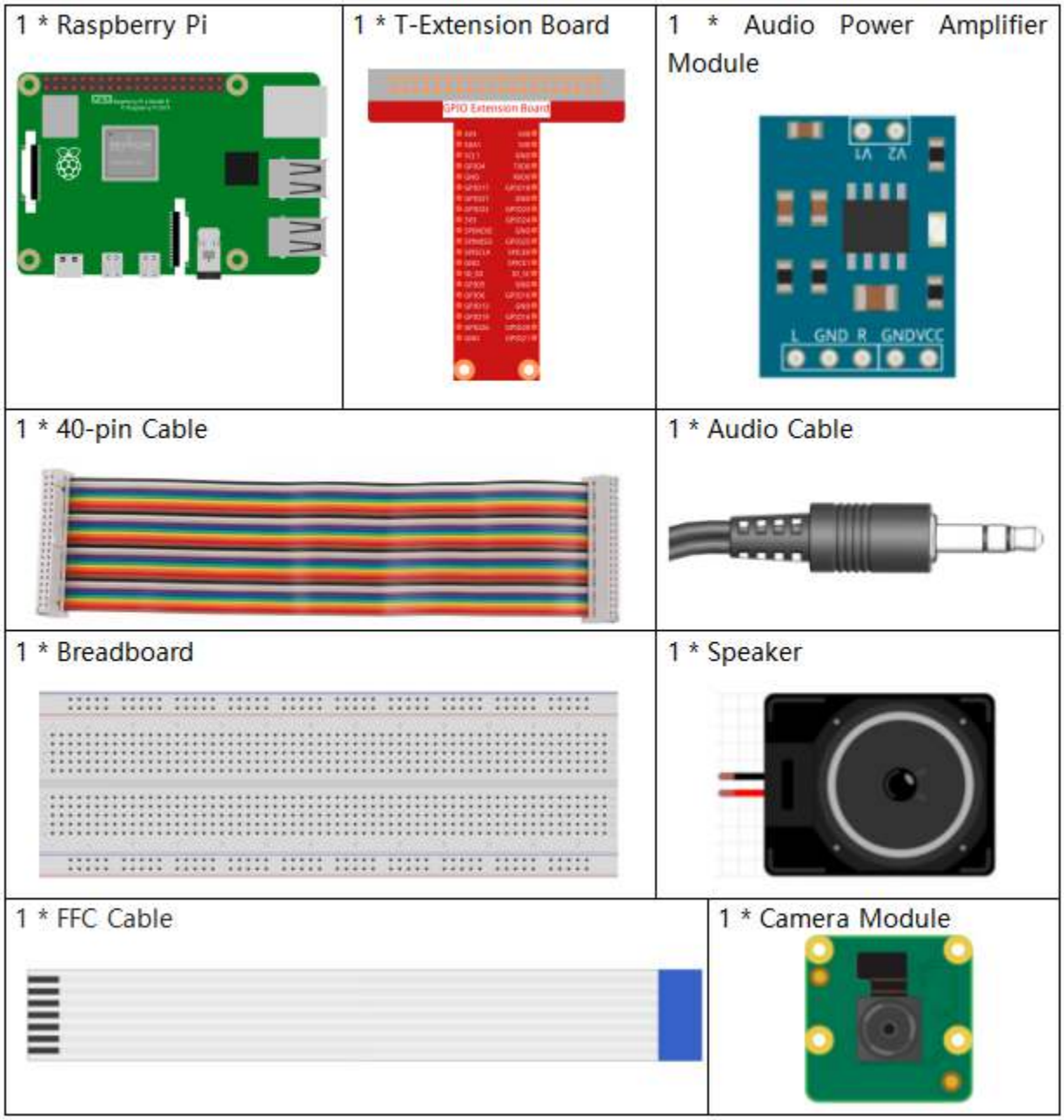


When you tap on the touch switch module, gpio17 is low. At this point, the **Drum-snare** sprite costume is switched to **drum-snare-b** and the drum sound played on speaker.

10.2.10 1.10 Drumming in the Air

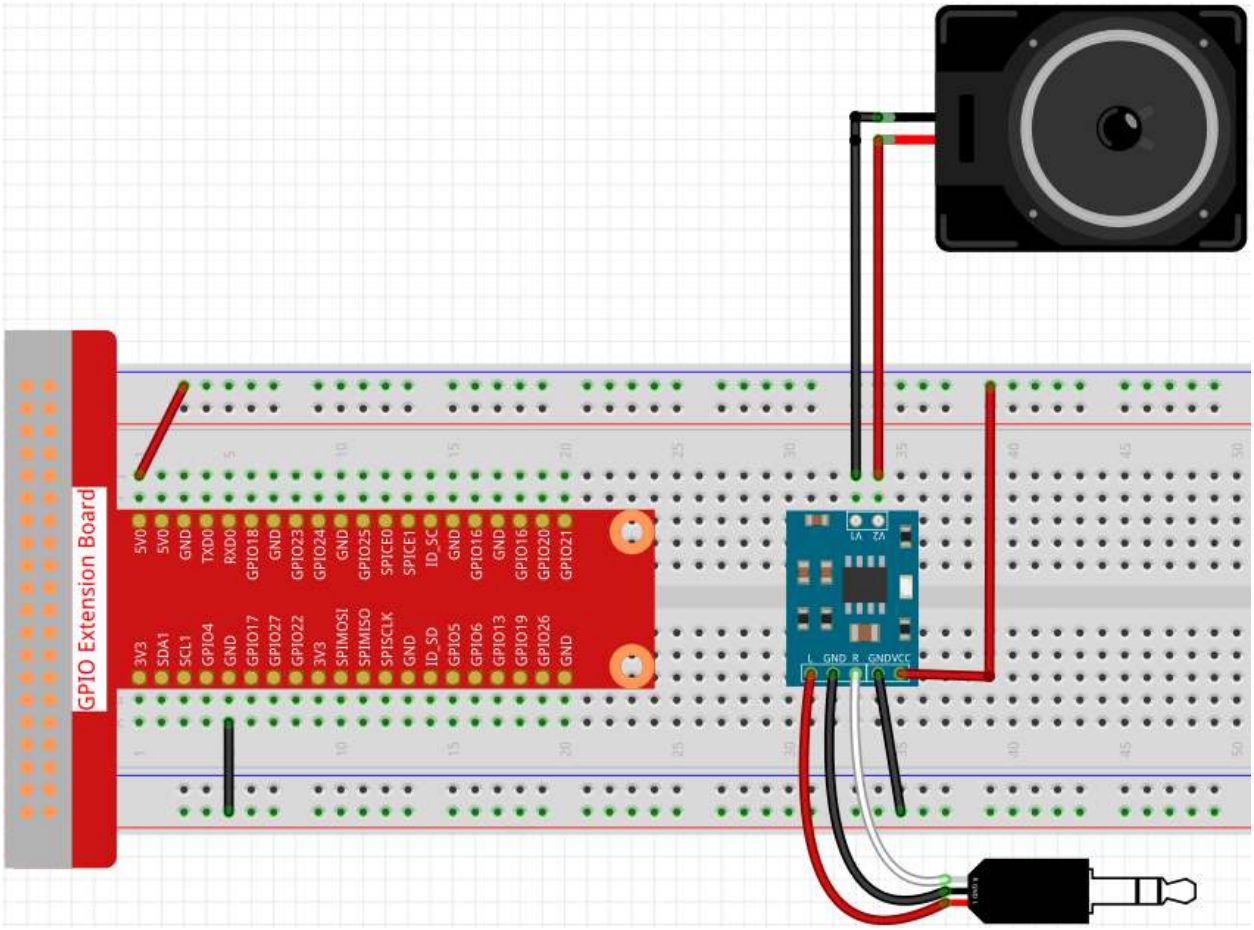
Today we will learn to use the Raspberry Pi camera, Scratch has an expansion module for Video Sensing which turns on the camera in Scratch and detects the movement of objects on the stage.

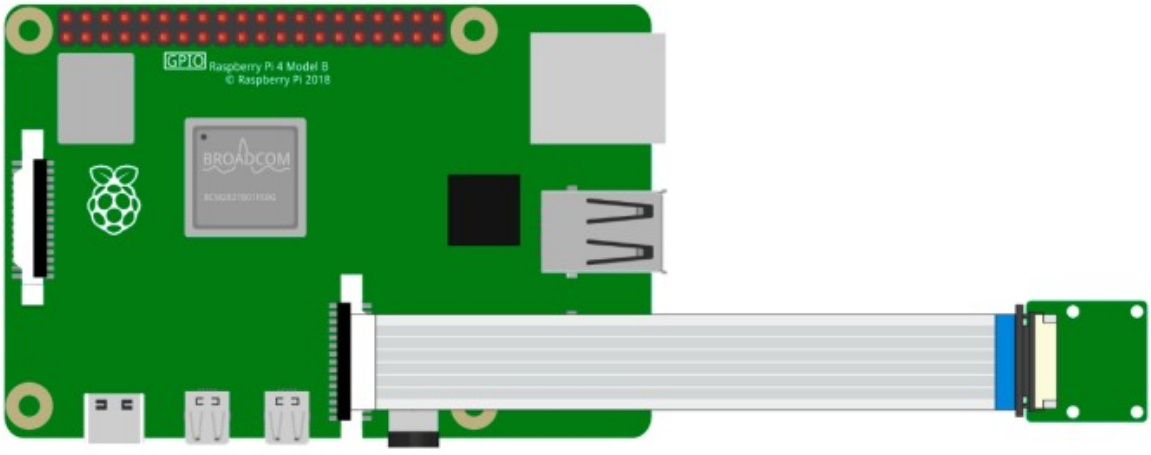
Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Audio Module and Speaker*
- *Camera Module*

Build the Circuit





Note: You need to refer to *Camera Module* to connect the camera module and enable the Raspberry Pi camera interface.

Load the Code and See What Happens

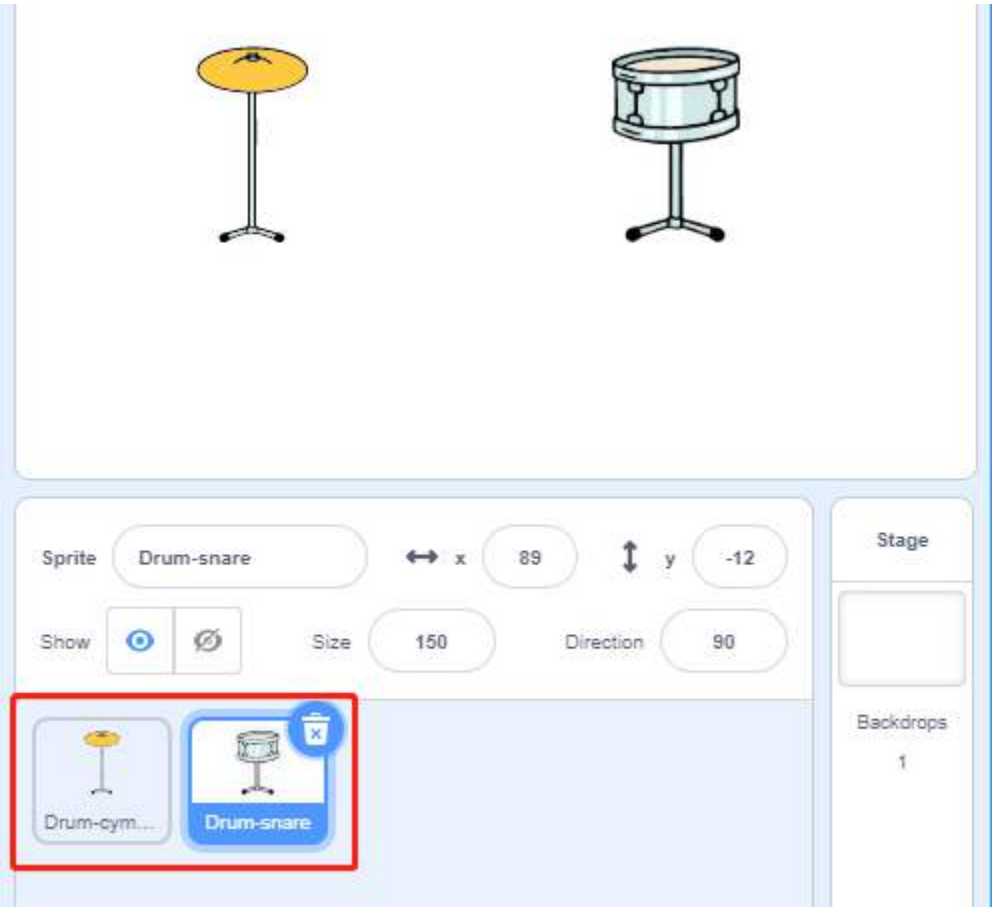
Load the code file (`1.10_drumming_in_the_air.sb3`) to Scratch 3.

Click on the green flag to start the game, place your hand in front of the camera module and Scratch 3 will make instrument sounds when your hand is shown touching an instrument on the stage area.

Note: For a better gaming experience, please try to play on a white background to avoid interference with the camera from other objects.

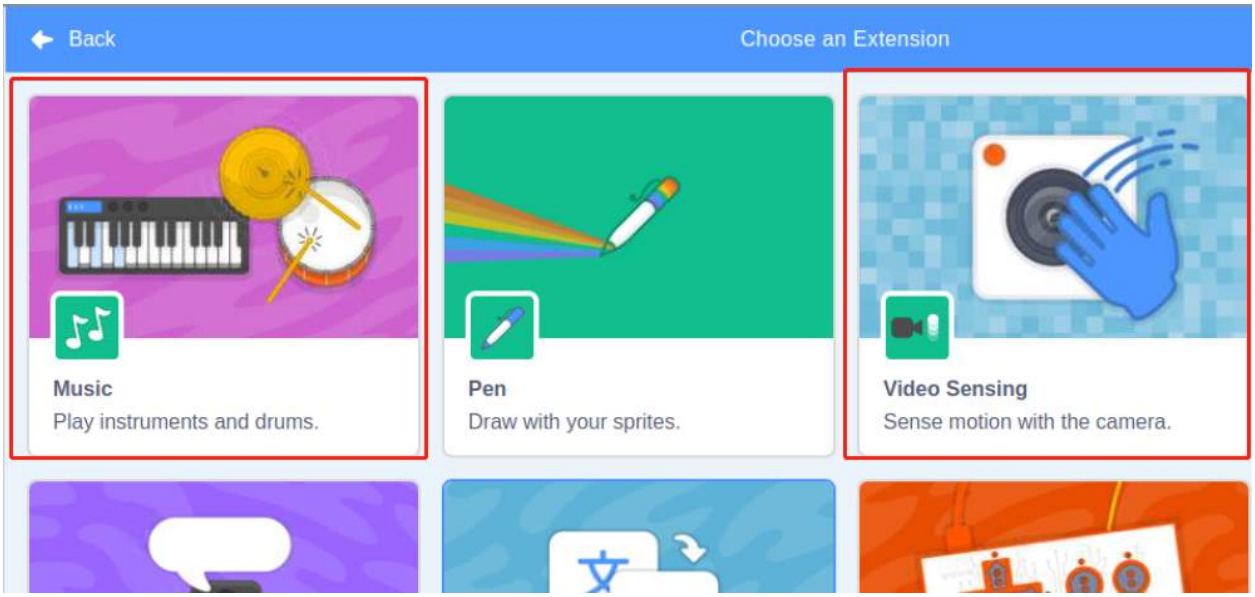
Tips on Sprite

First delete the default sprites, then find the **Drum-cymbal** sprite and **Drum-snare** sprite and add them.

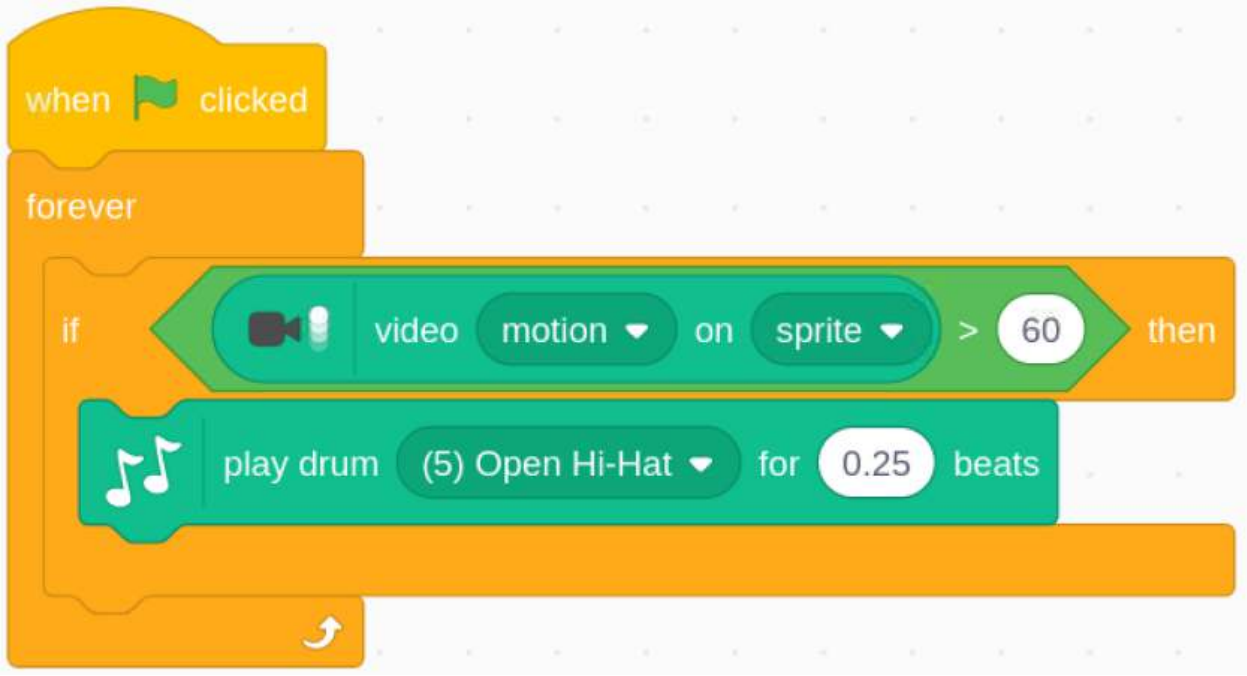


Click the **Add Extension** icon at the bottom left of Scratch and add the **Music** and **Video Sensing** extensions to it.

Variables
My Blocks
glide 1 secs to x: 0 y: 0
point in direction 90
point towards mouse-pointer
change x by 10
set x to 0
change y by 10

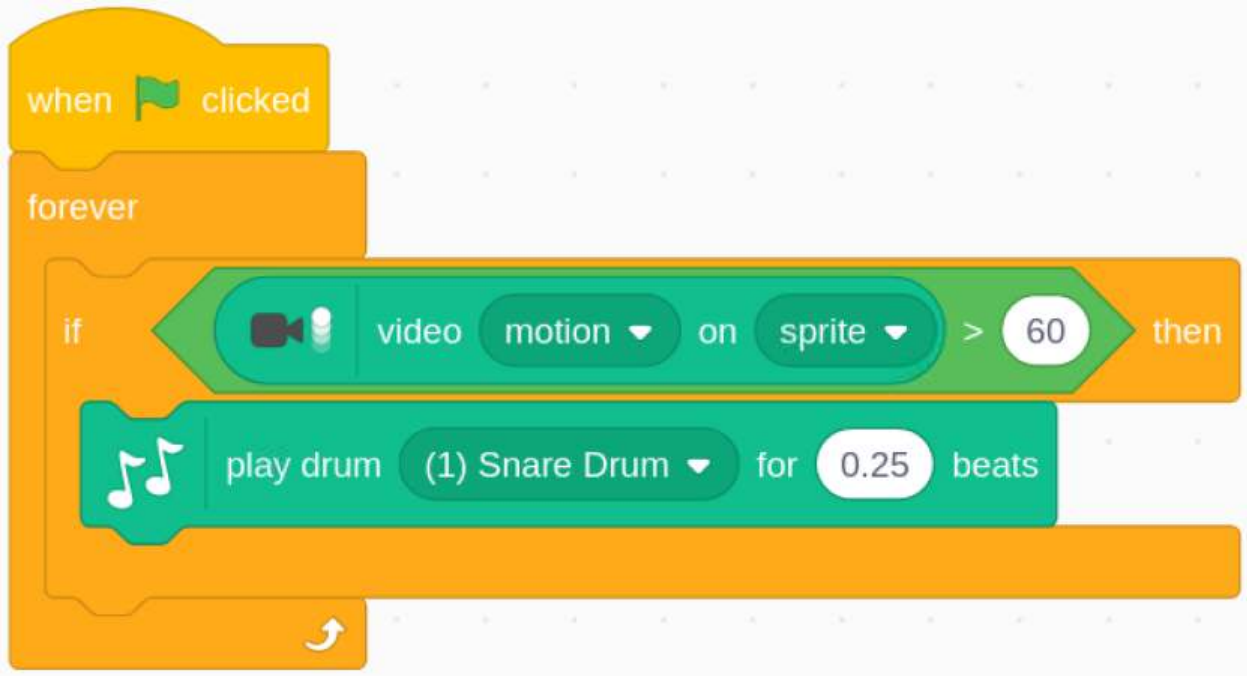

Back
Choose an Extension
Music
Play instruments and drums.
Pen
Draw with your sprites.
Video Sensing
Sense motion with the camera.

Tips on Codes



When the green flag is clicked, it keeps cycling to detect if our hand is moving over the **Drum-cymbal** sprite by more than 60. if so, it is assumed that our hand touched the sprite, at which point the Open Hi-Hat instrument sound is played.

Note: The movement magnitude refers to the change in coordinates on the stage area, which is calculated with respect to the amount of change in the coordinates of the detection target on the stage area.



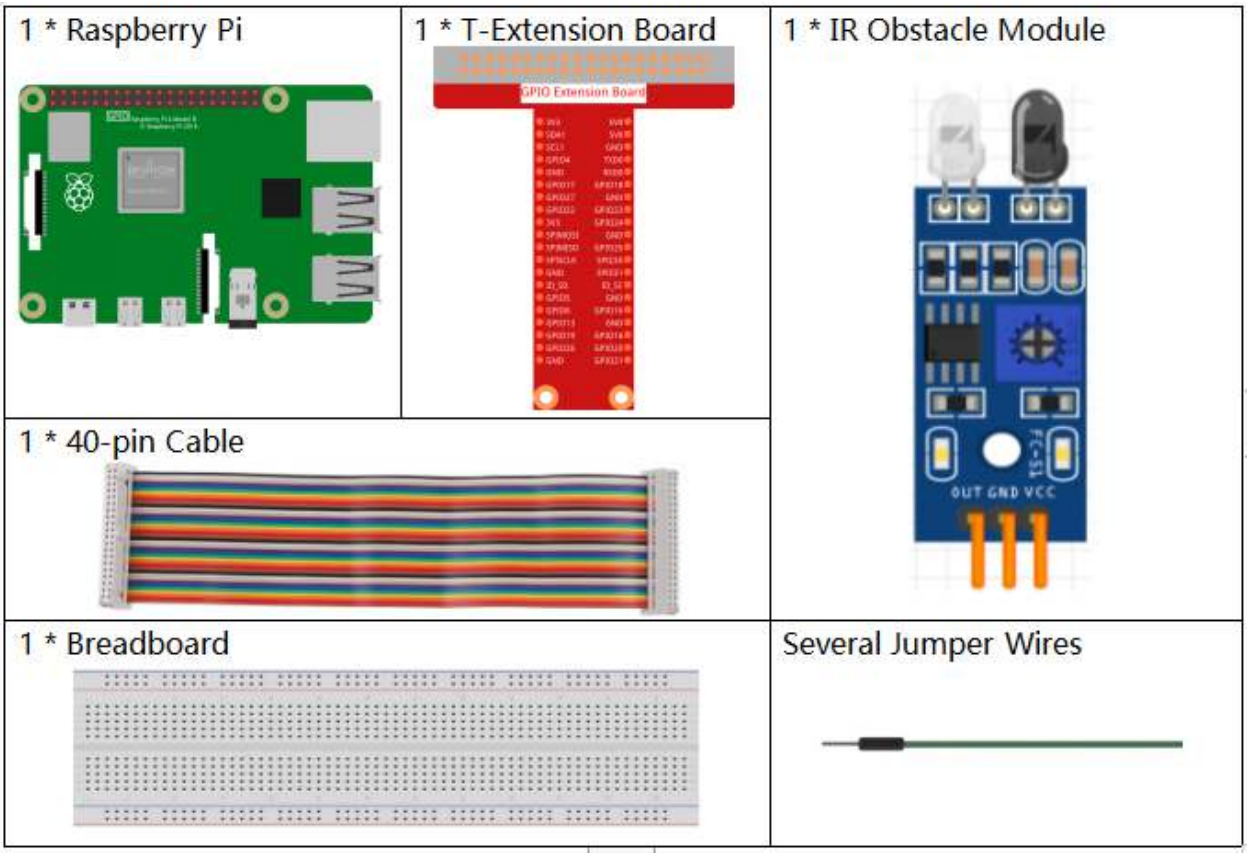
Similarly, if the movement of our hand on the **Drum-snare** sprite is detected to be greater than 60, our hand is considered to have touched the sprite and the sound of the snare drum instrument is played.

10.2.11 1.11 Repelling locusts

Today, we will use IR obstacle avoidance module, Raspberry Pi and Scratch to make a locust repelling game.

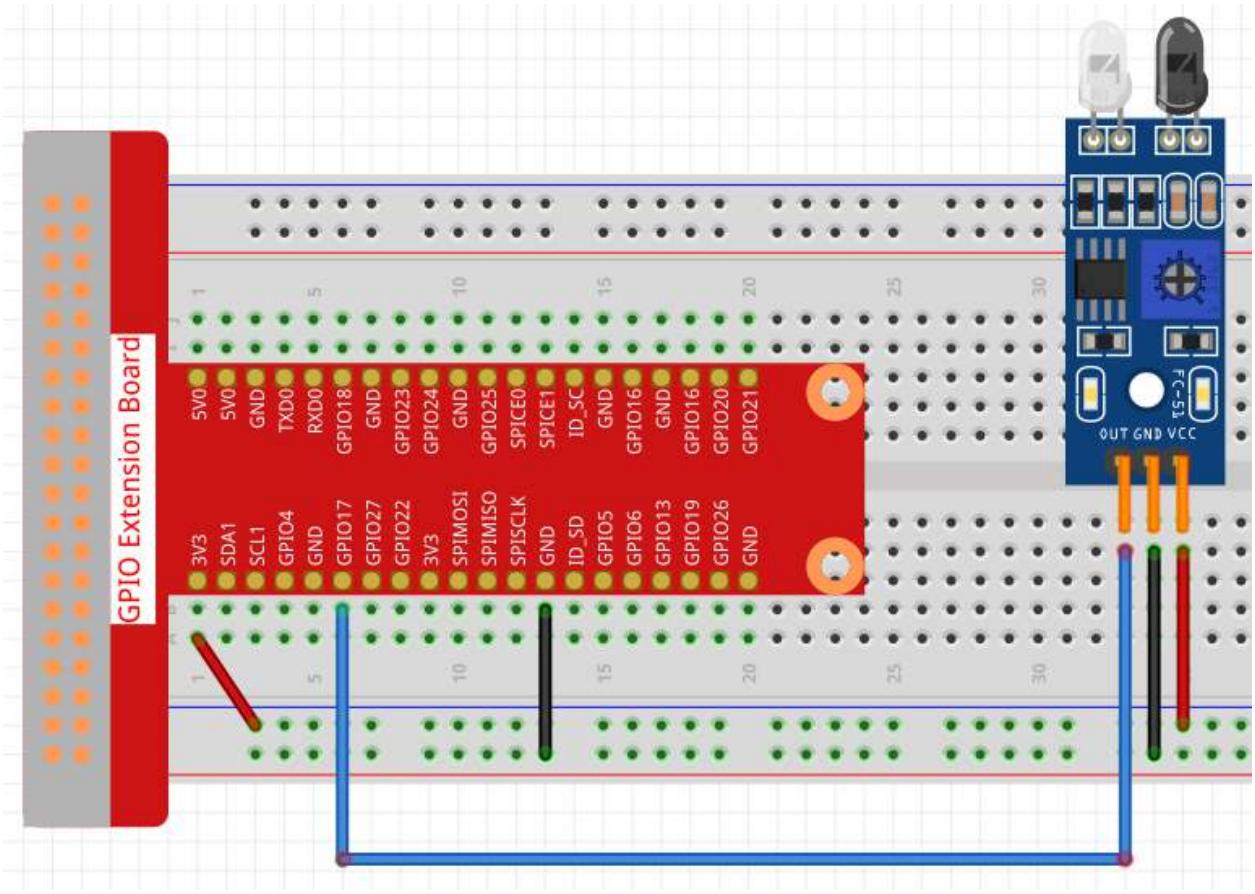
Place your hand in front of the obstacle avoidance module and you will see the locusts being chased away.

Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Obstacle Avoidance Module*

Build the Circuit



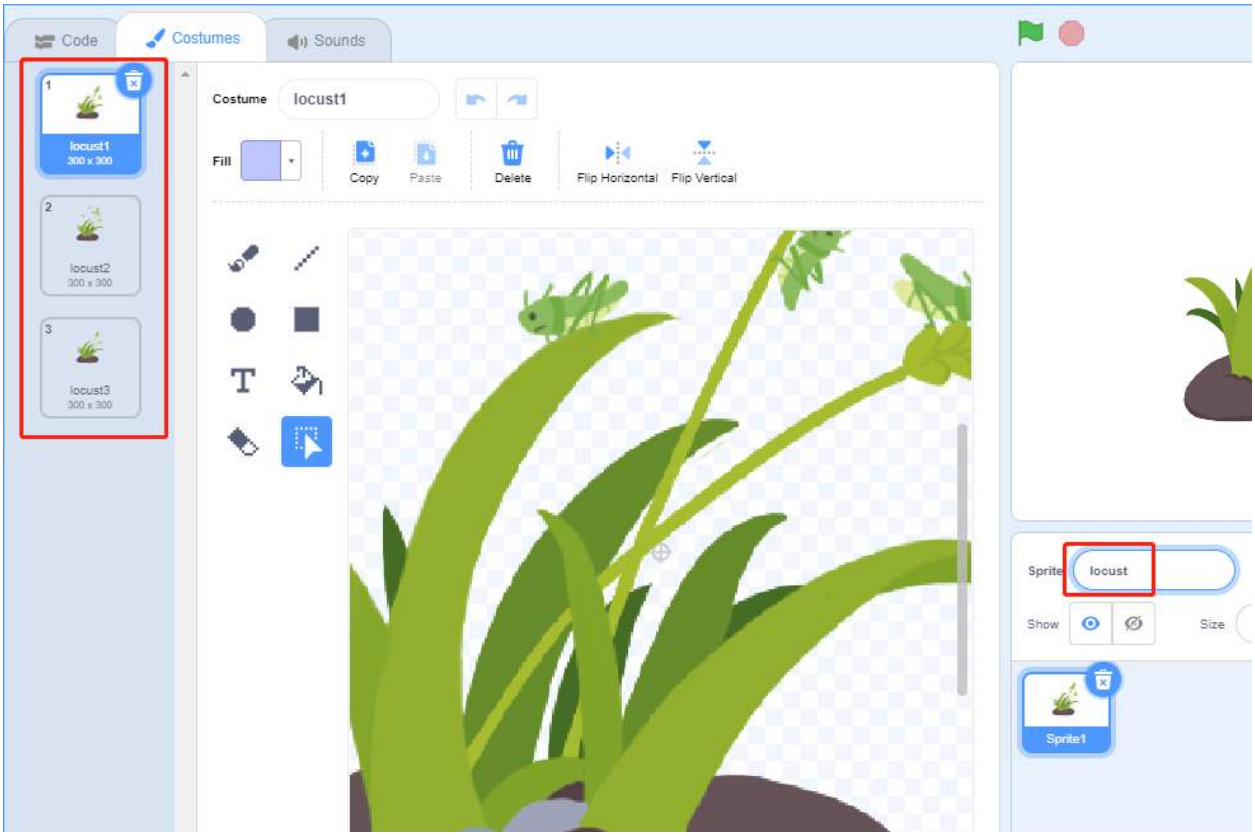
Load the Code and See What Happens

Load the code file (`1.11_repelling_locusts.sb3`) to Scratch 3.

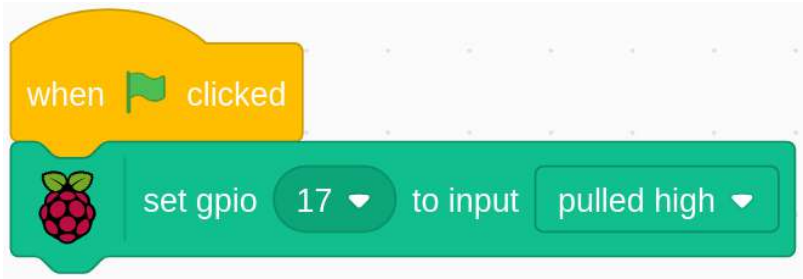
Place your hand in front of the obstacle avoidance module and you will see the locusts being chased away.

Tips on Sprite

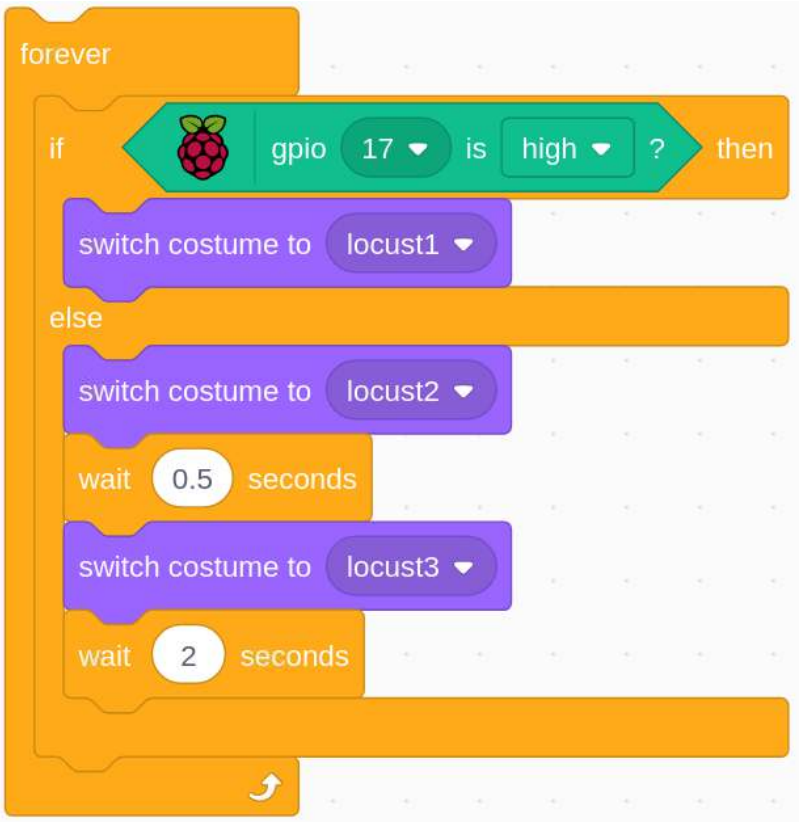
Select Sprite1 and click **Costumes** in the top left corner; upload **locust1.png**, **locust1.png** and **locust3.png** from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes, and rename the sprite to **locust**.



Tips on Codes



When the IR obstacle avoidance module does not detect an obstacle (no hand is placed in front of the probe), the gpio is high.

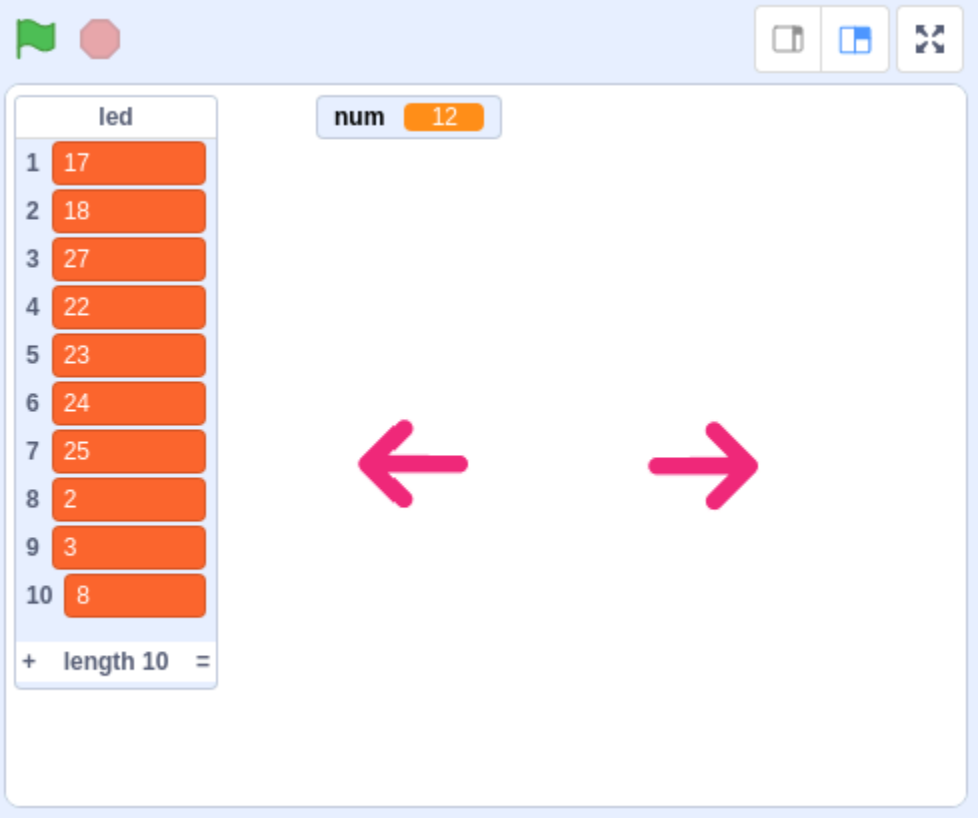


When gpio17 is high (no obstacles go in front of the IR obstacle avoidance module), switch the locust sprite's costume to locust1 (locusts gather in wheat). Conversely when gpio17 is low (put your hand in front of the IR obstacle avoidance module), switch the locust sprite's costume to locust2 (expel locusts), then switch the locust sprite's costume to locust3 (locusts are completely expelled) after 0.5s.

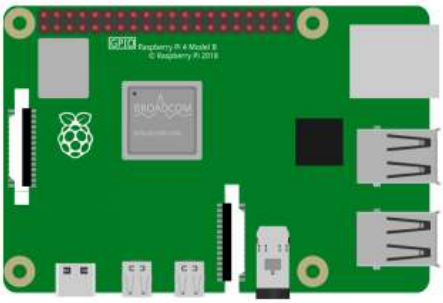
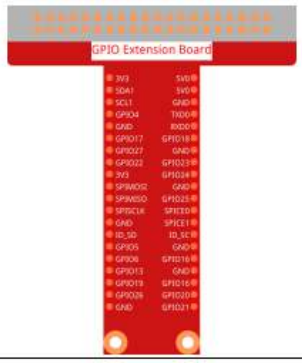
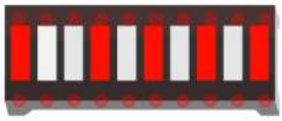
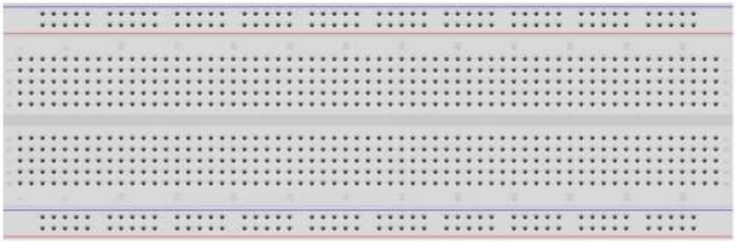
10.2.12 1.12 Water Lamp

Today, we will use LED Bar Graph, Raspberry Pi and scratch to make a Water Lamp.

The LED Bar Graph will light up in order with the direction of the arrows on the stage.

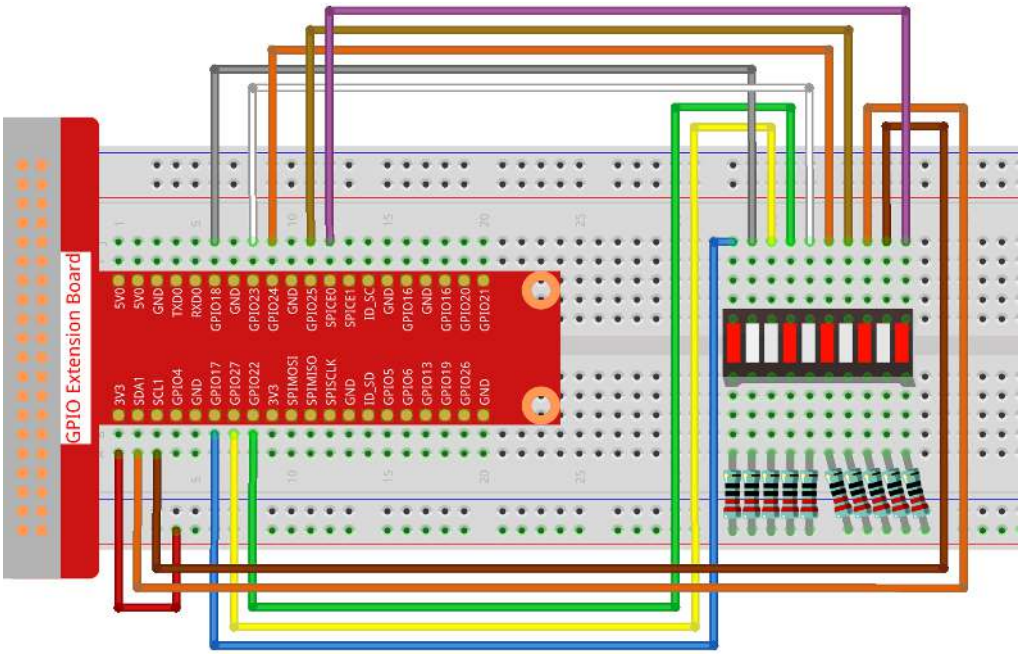

led
1 17
2 18
3 27
4 22
5 23
6 24
7 25
8 2
9 3
10 8
+ length 10 =
num 12

Required Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *LED Bar Graph*

Build the Circuit



Load the Code and See What Happens

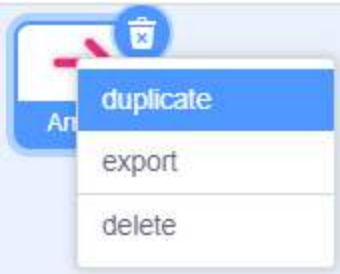
Load the code file (`1.12_water_lamp.sb3`) from your computer to Scratch 3.

By clicking on **Arrow1**, the LEDs on the LED bar are lit in sequence from the left to the right (one at a time) and then off. Click **Arrow2** and the LEDs light up in the opposite order.

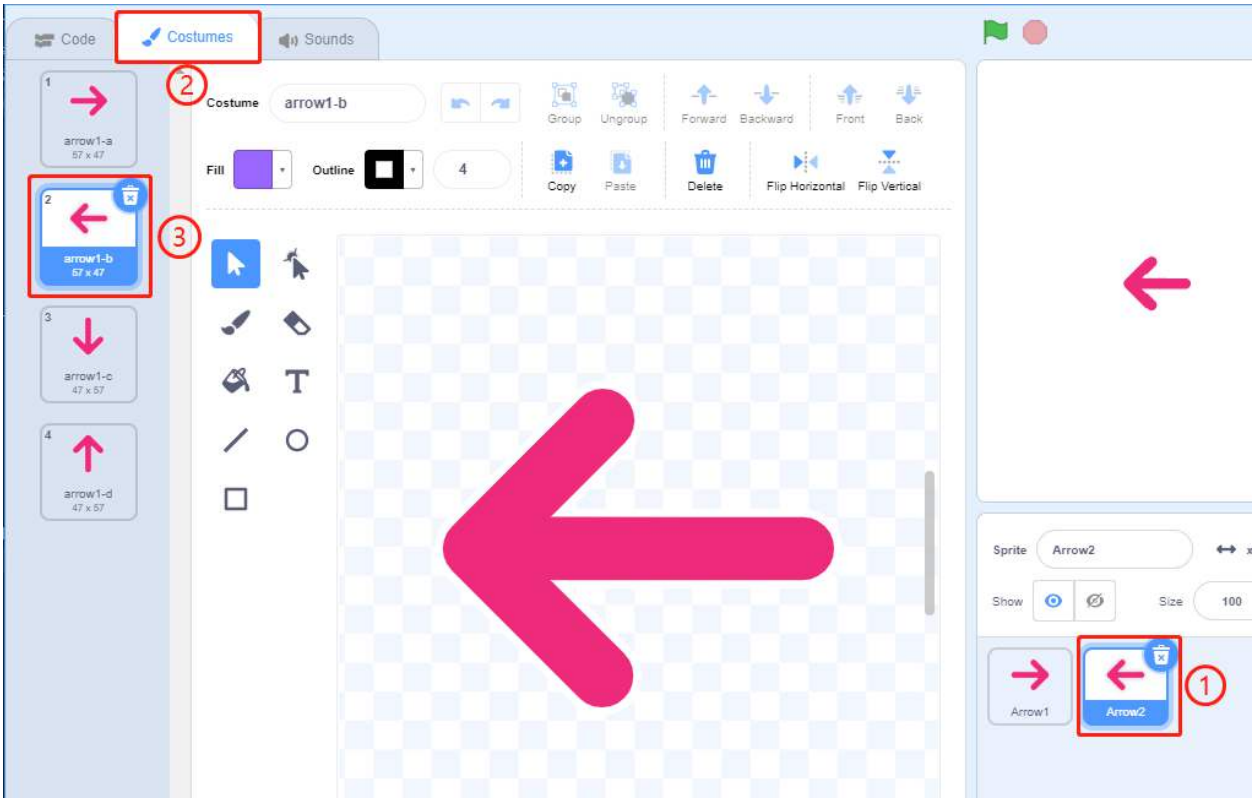
Tips on Sprites

Delete the default sprite and choose the **Arrow1** sprite.

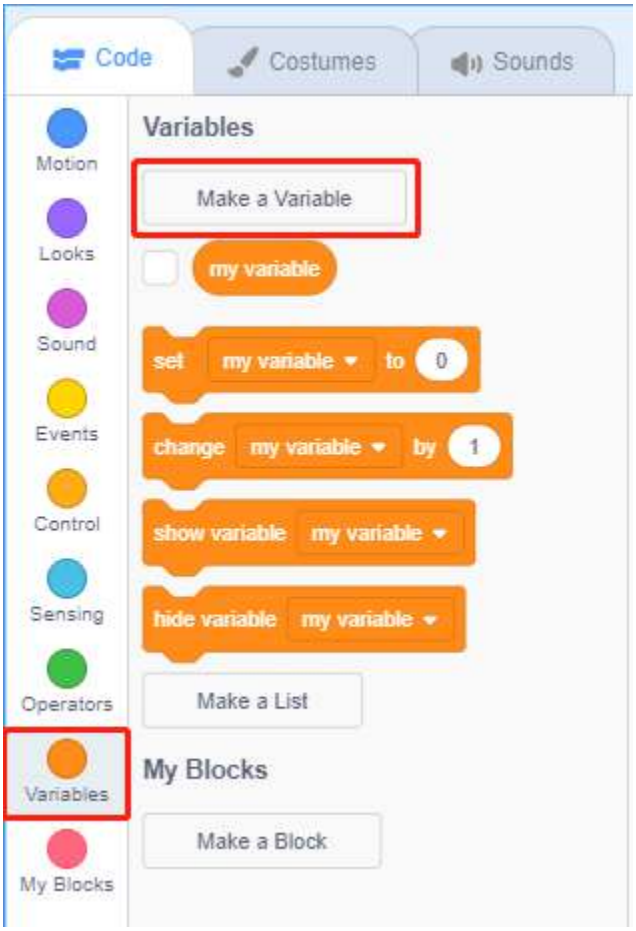
Here we will need 2 **Arrow1** sprites, which can be done with the duplicate button.



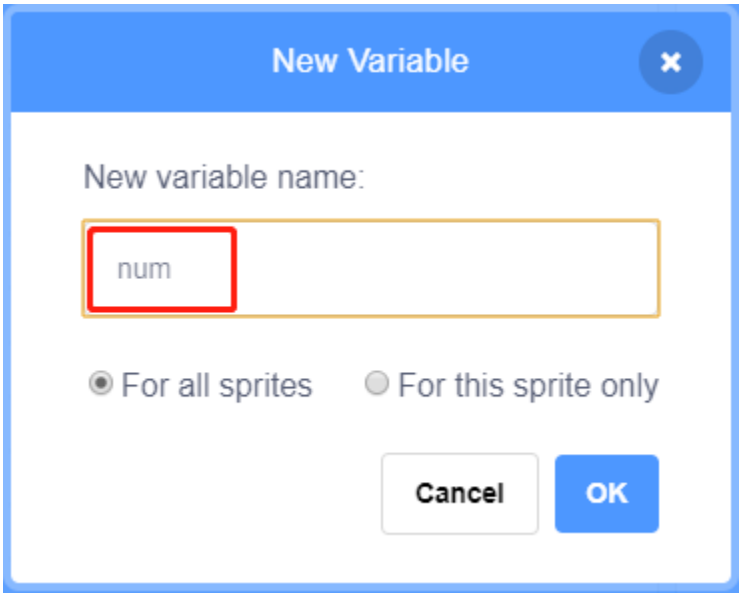
Click on the **Arrow 2** sprite and change the direction of the arrow by selecting costume 2.



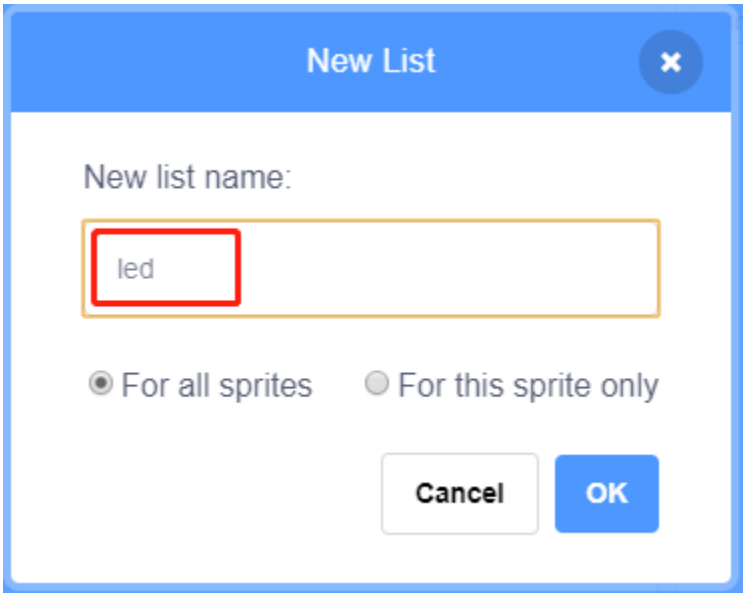
Now let's make a variable.



Name it as **num**.

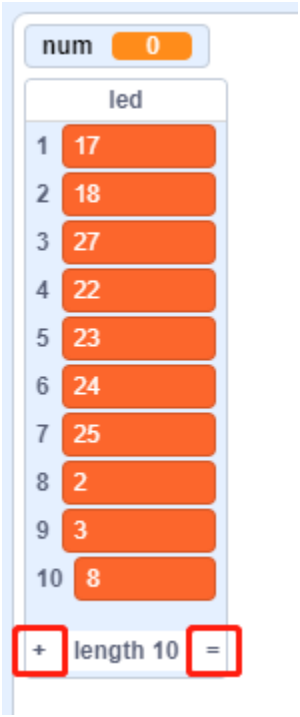


Follow the same method to create a list called **led**.

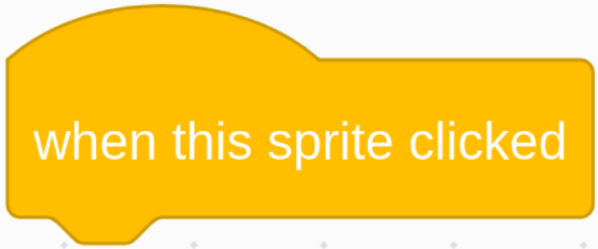


After adding, you should see the **num** variable and the **led** list on the stage area.

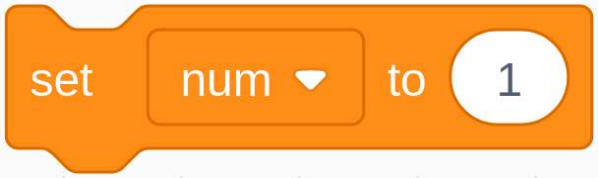
Click **+** to add 10 list items and enter the pin numbers in order (17,18,27,22,23,24,25,2,3,8).



Tips on Codes



This is an event block that is triggered when the current sprite is clicked.

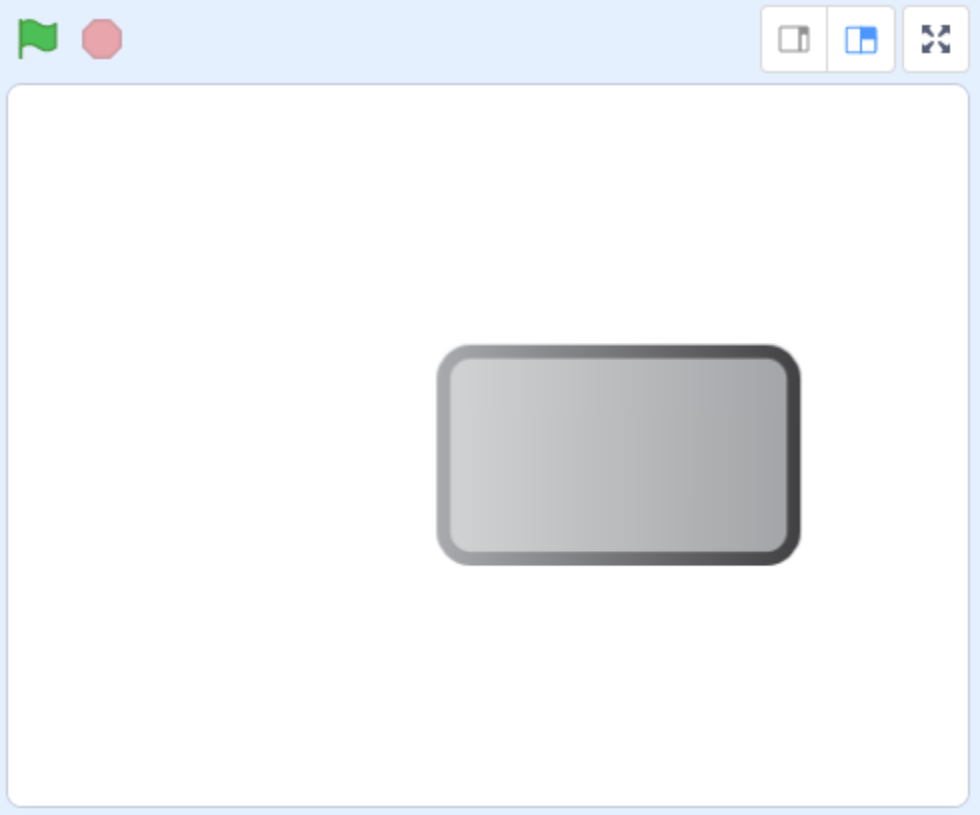


The initial value of the **num** variable determines which LED is lit first.

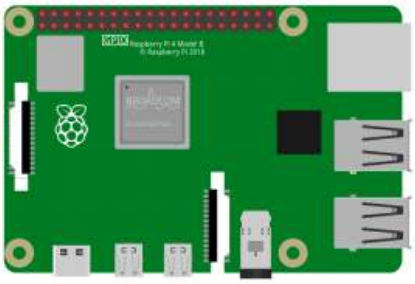
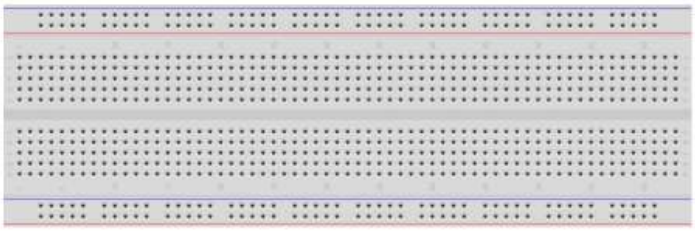
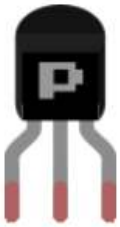
Set the pin corresponding to **num** in the led list to low to light the LED, and then set the pin corresponding to **num-1** to high to turn off the previous LED.

10.2.13 1.13 Doorbell

Today we will make a doorbell, click the button3 sprite on the stage, the buzzer will sound; click again, the buzzer will stop sounding.

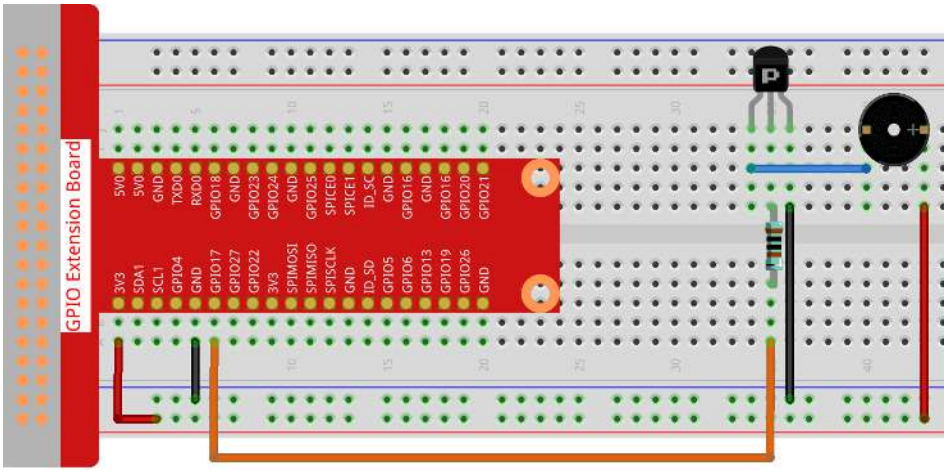


Required Components

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 

- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *Transistor*

Build the Circuit



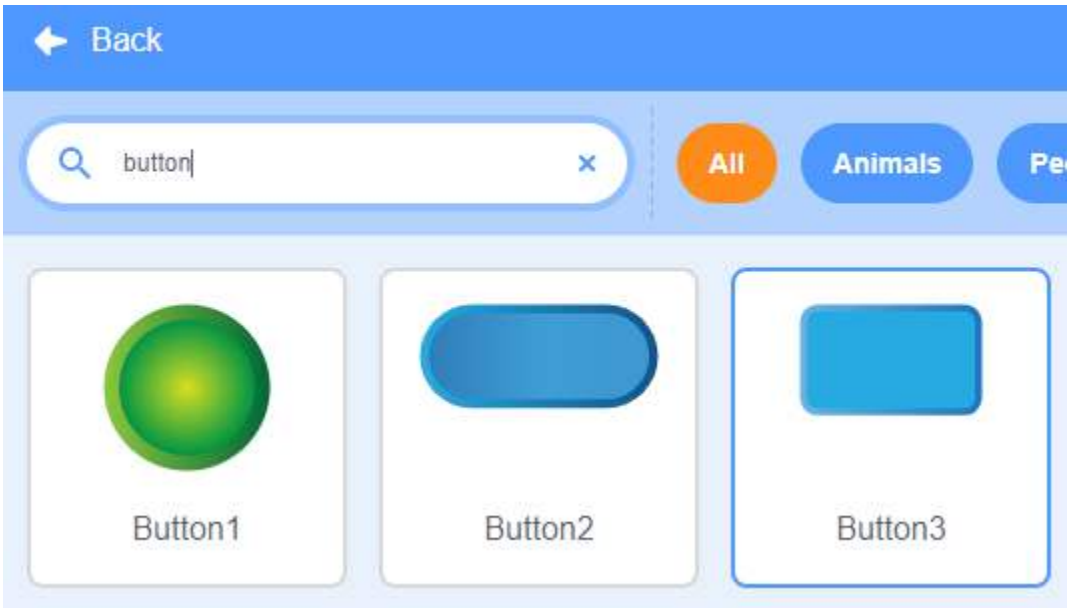
Load the Code and See What Happens

Load the code file (`1.13_doorbell.sb3`) to Scratch 3.

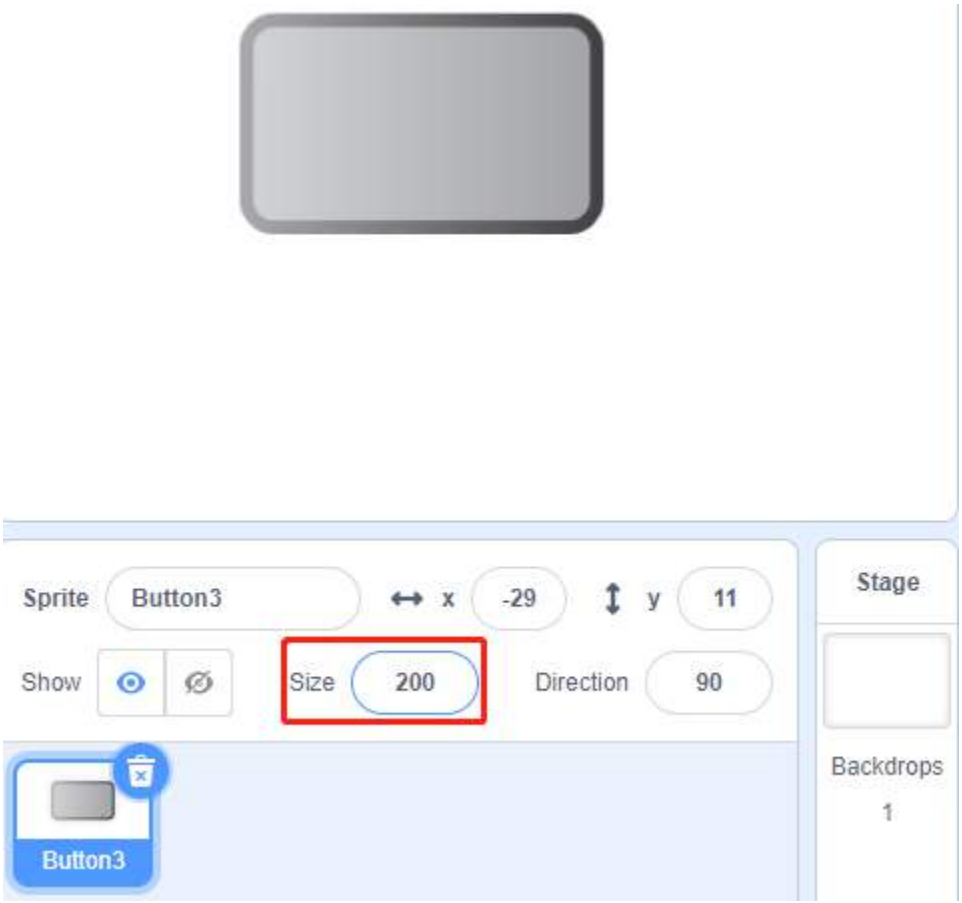
Click on the green flag on the stage. When we click on the Button 3 sprite, it will turn blue and then the buzzer will sound; when we click again, the **Button3** sprite reverts to gray and the buzzer stops sounding.

Tips on Sprite

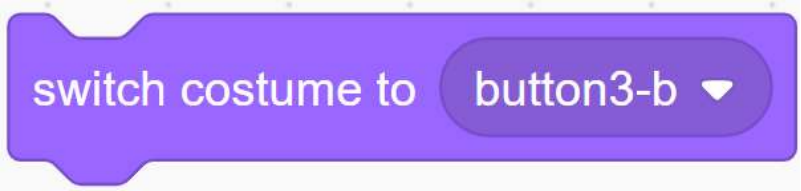
Delete the default sprite, then choose the **Button 3** sprite.



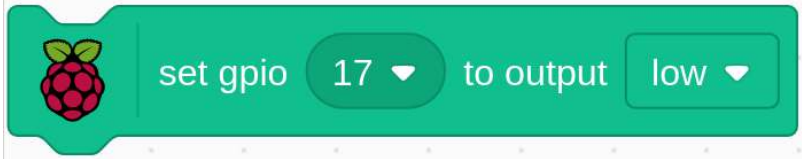
Then set the size to 200.



Tips on Codes



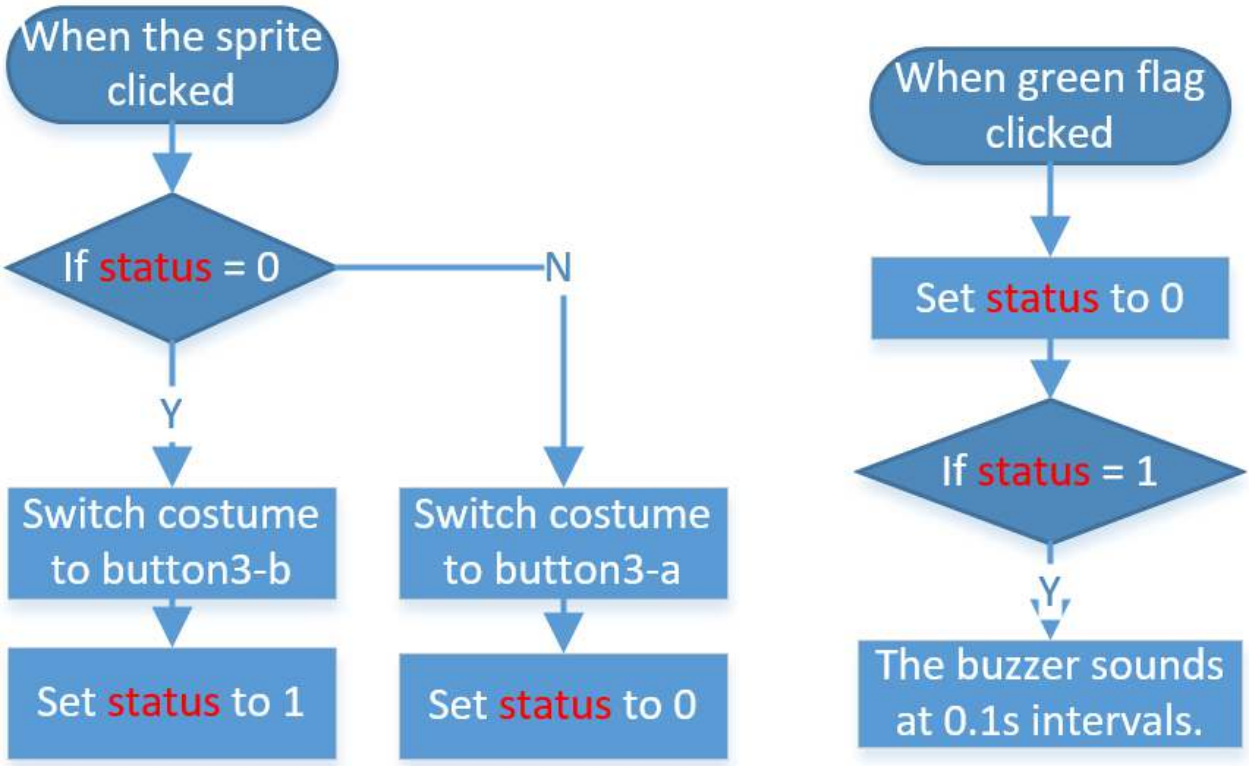
This block allows you to switch the sprite's costume.



Set gpio17 to low to make the buzzer sound; set it to high and the buzzer will not sound.

The **status** switch is used here, and we will use a flowchart to help you understand the whole code.

When the green flag is clicked, the **status** will be set to 0 first, and wait for the sprite to be clicked at this time; if **button3** sprite is clicked, it will switch to costume as **button-b** costume (blue) and the **status** will be set to 1. When the main program receives the **status** as 1, it will let the buzzer sound at 0.1s interval. If **button3** is clicked again, it will switch to **button-a** costume (gray) and **status** will be set to 0 again.



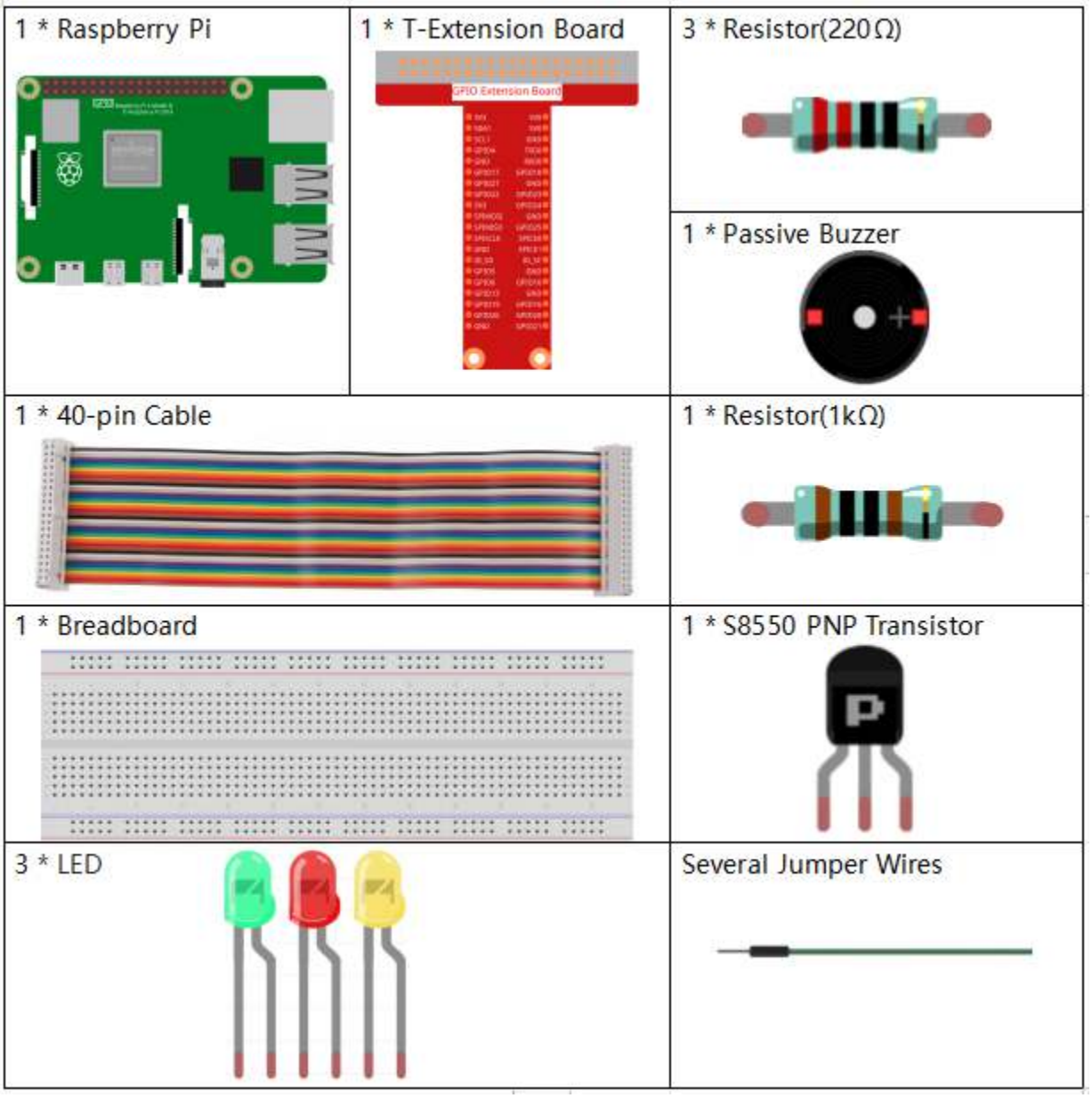
10.2.14 1.14 123 Wooden Man

Today, we are going to play a game of 123 wooden man.

Click on the green flag to start the game, hold down the right arrow key on the keyboard to make the sprite go right. If the green light is on, the sprite can move; but when the red LED is on, you have to stop the sprite from moving; otherwise the buzzer will keep ringing.

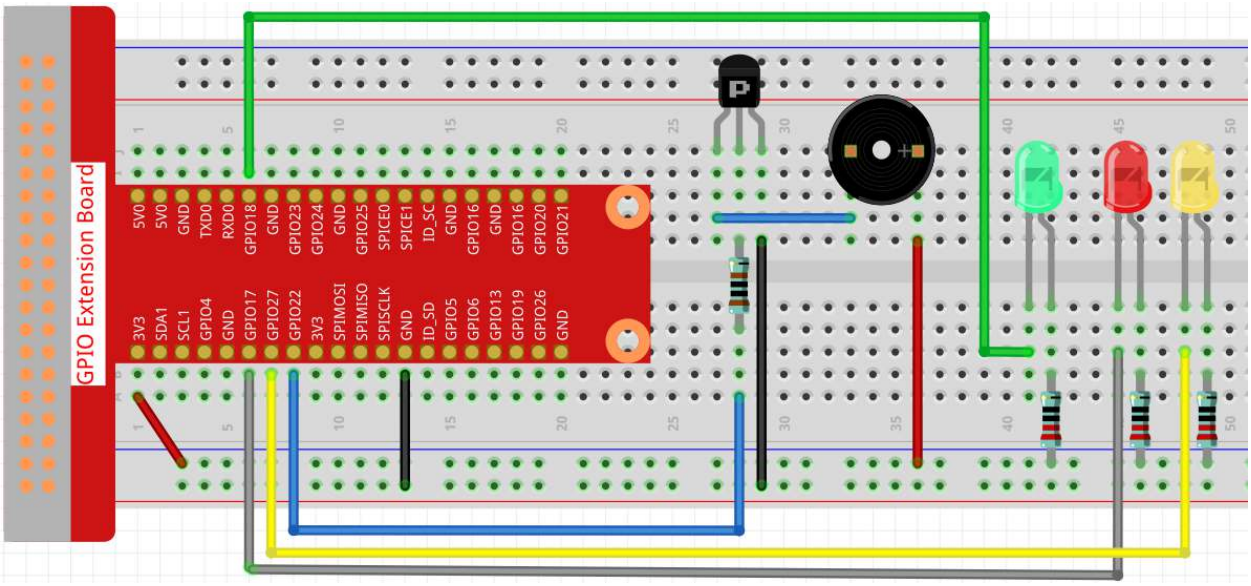
status
1

Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Buzzer*
- *LED*
- *Transistor*

Build the Circuit



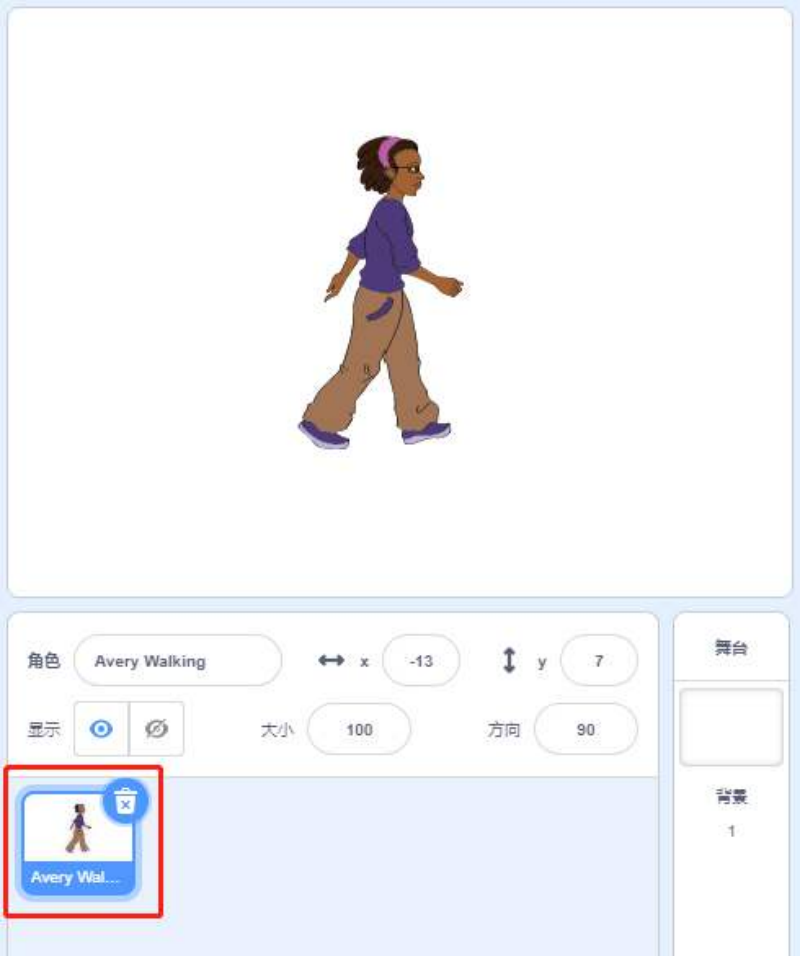
Load the Code and See What Happens

Load the code file (`1.14_123_wooden_man.sb3`) to Scratch 3.

When the green LED is on, you can use the right arrow key to control **Avery** to walk to the right; when the red LED is on, if you continue to let **Avery** move to the right, then an alarm will sound.

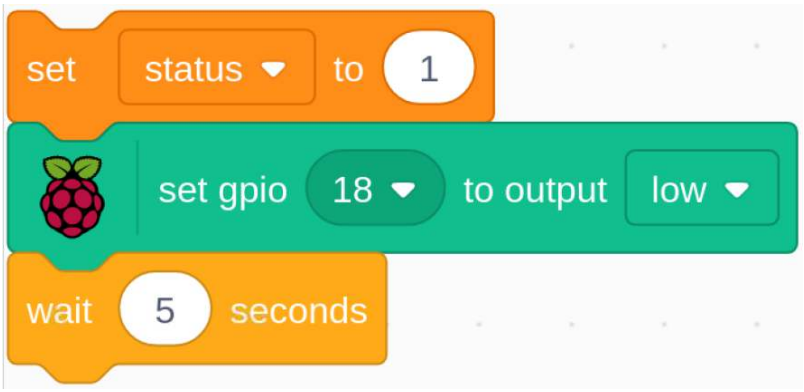
Tips on Sprite

Delete the default sprite, then choose the **Avery Walking** sprite.

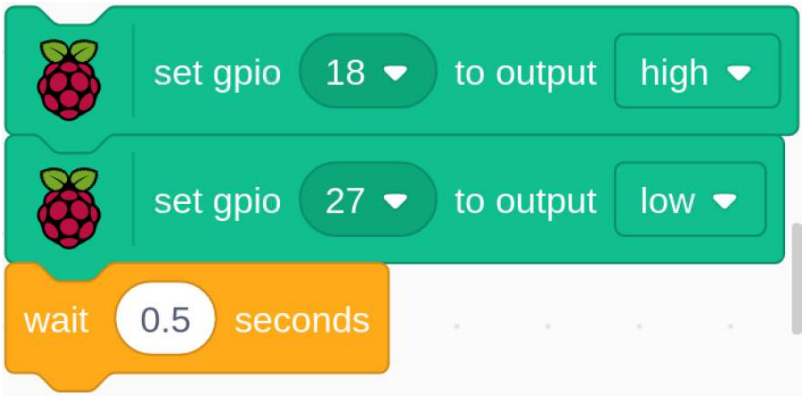


Tips on Codes

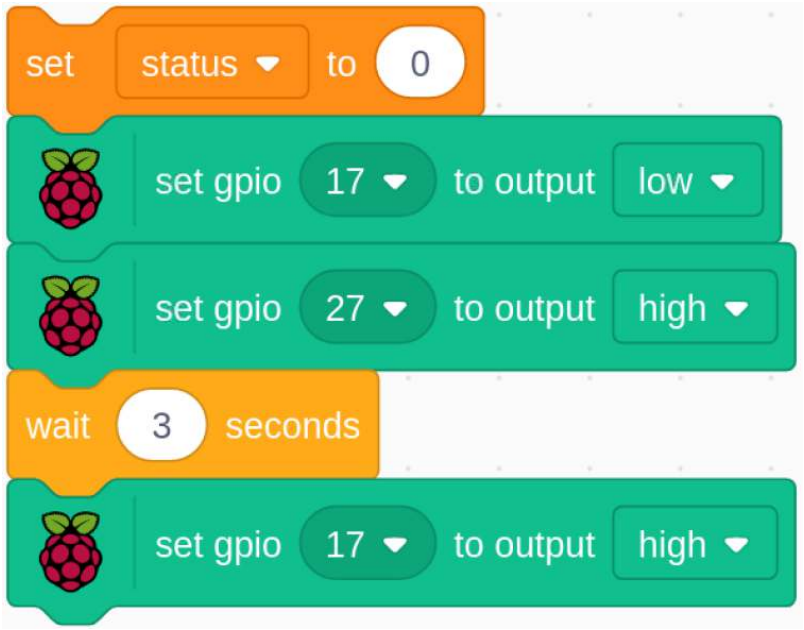
Initialize all pins to high.



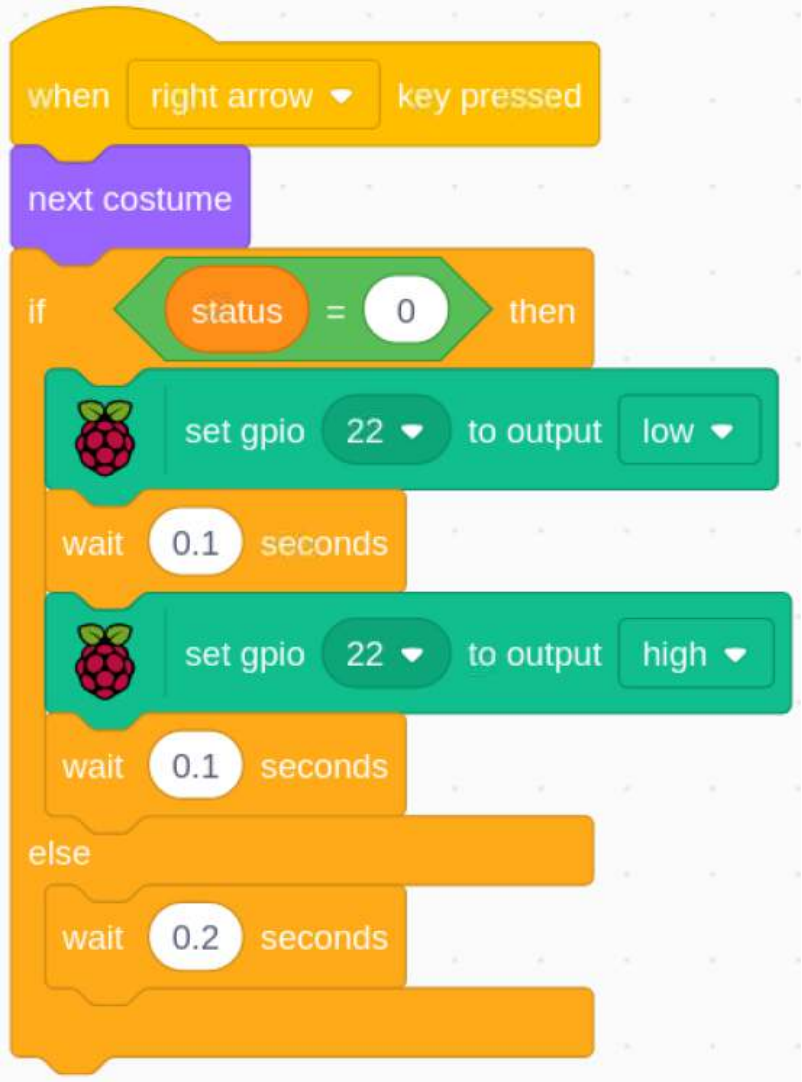
When the game starts, assign the status variable to 1, indicating that the Avery Walking sprite is movable, and then set gpio18 to low, which lights up the green LED for 5s.



Set gpio18 to high, then set gpio27 to low, which means turn off the green LED and light up the yellow LED for 0.5s.



Assign the status variable to 0, which means the Avery Walking sprite is not moving; then set gpio27 to low and gpio17 to high, which turns off the yellow LED and then lights up the red LED for 3s. Finally, set gpio17 to high to turn off the red LED.

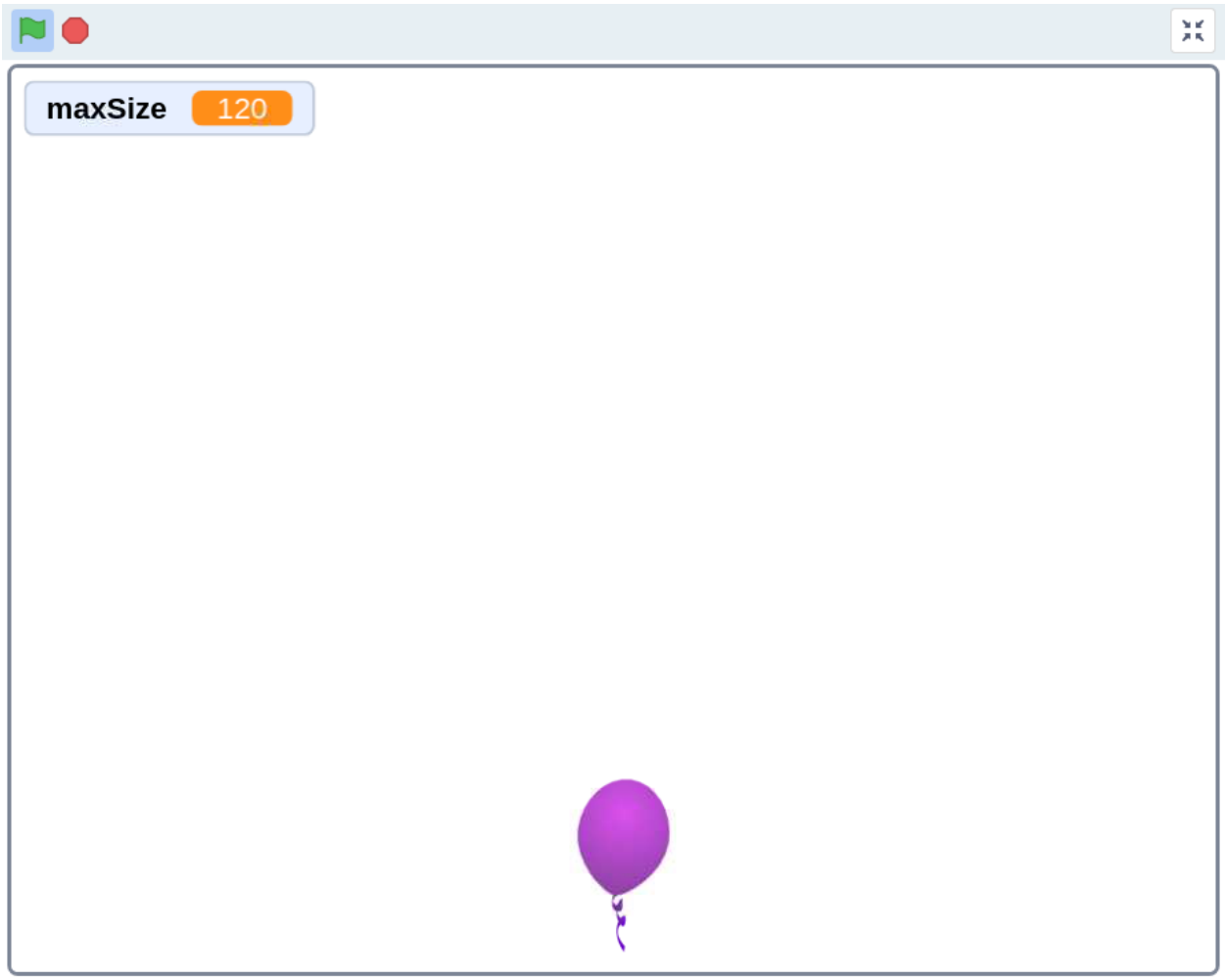


When we press the right arrow key on the keyboard, we need to switch the **Avery Walking** sprite to the next costume so that we can see Avery walking to the right. Then we need to determine the value of the **status** variable. If it is 0, it means that the Avery Walking sprite is not moving at this moment, and the buzzer will sound to warn you that you cannot press the right arrow key again.

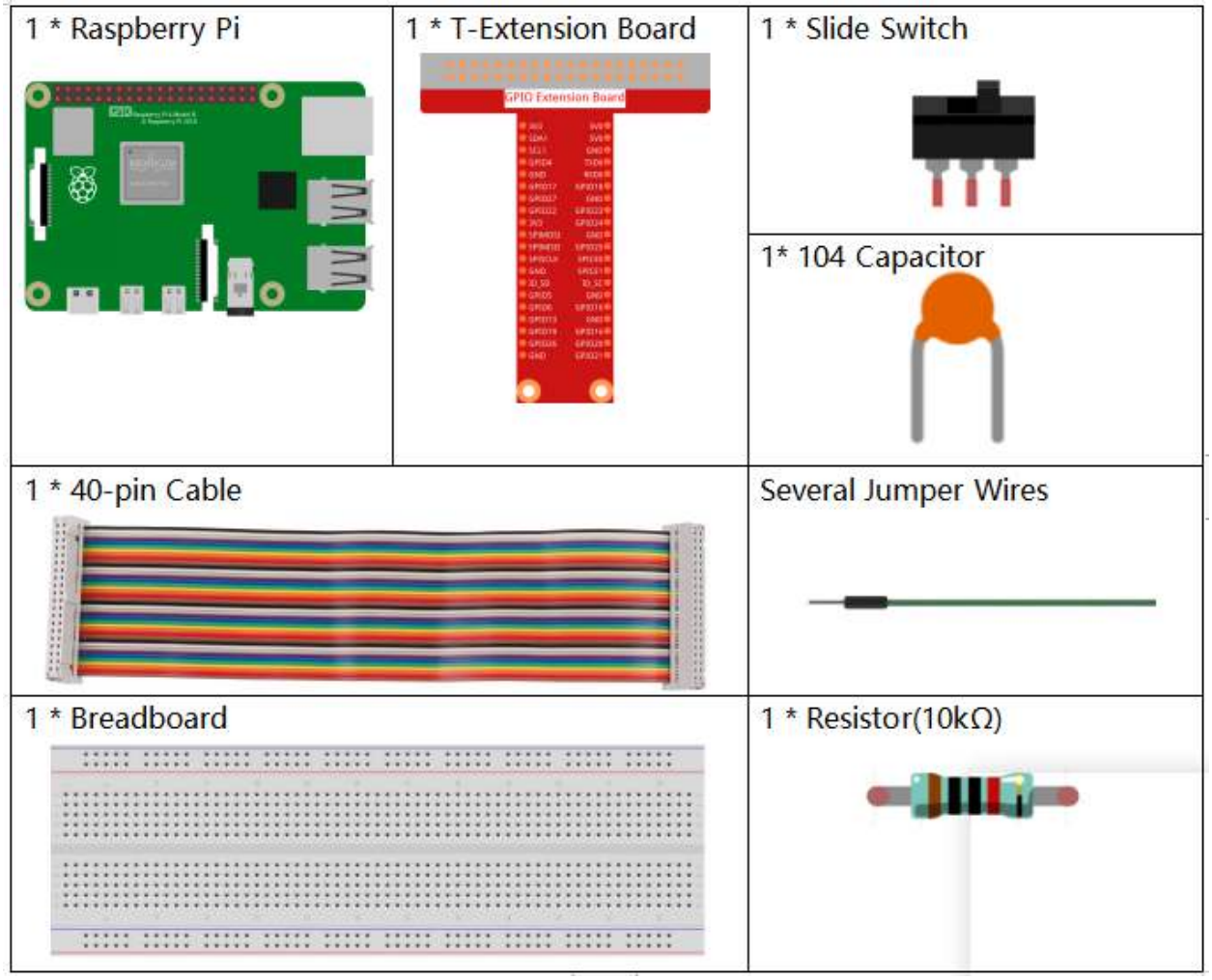
10.2.15 1.15 Inflating the Balloon

Here, we will play a game of ballooning.

By toggling Slide to the left to start to inflate the balloon, at this time the balloon will get bigger and bigger. If the balloon is too large will blow up; if the balloon is too small, it will not float into the air. You need to judge when to toggle the switch to the right to stop pumping.

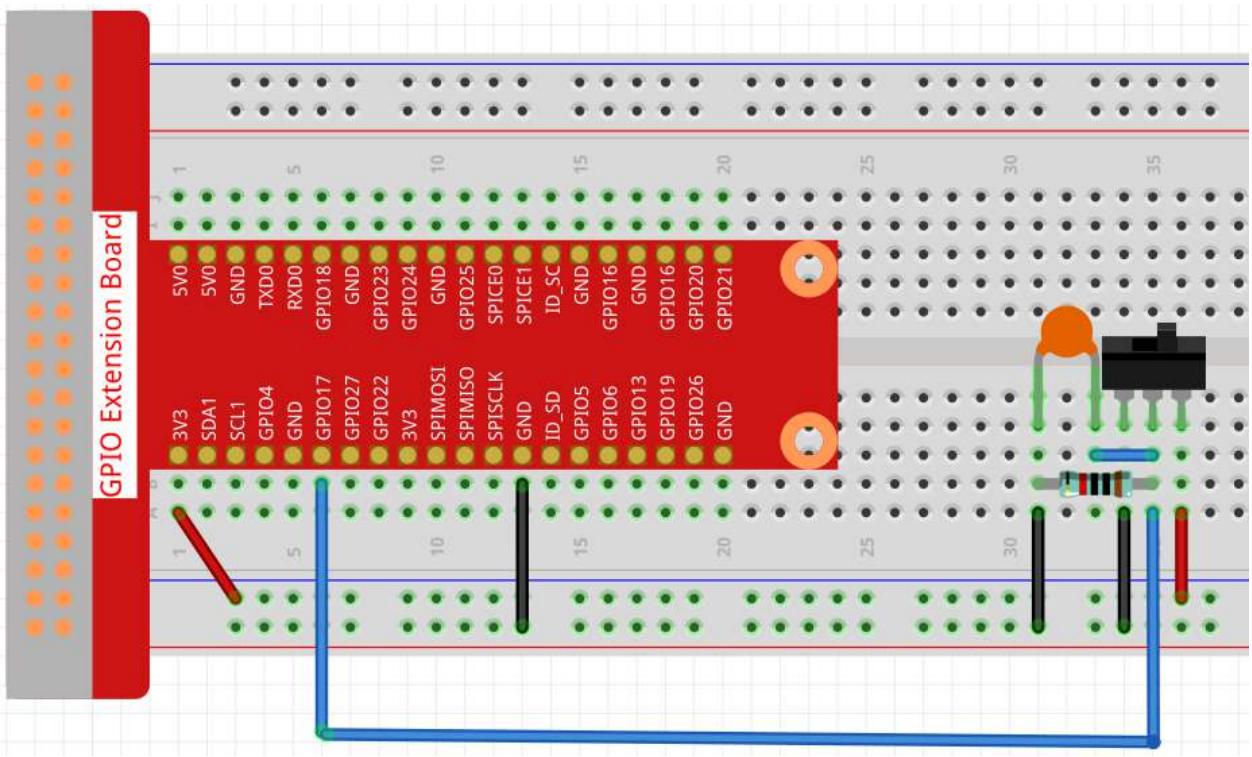

maxSize
120

Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Slide Switch*
- *Capacitor*

Build the Circuit



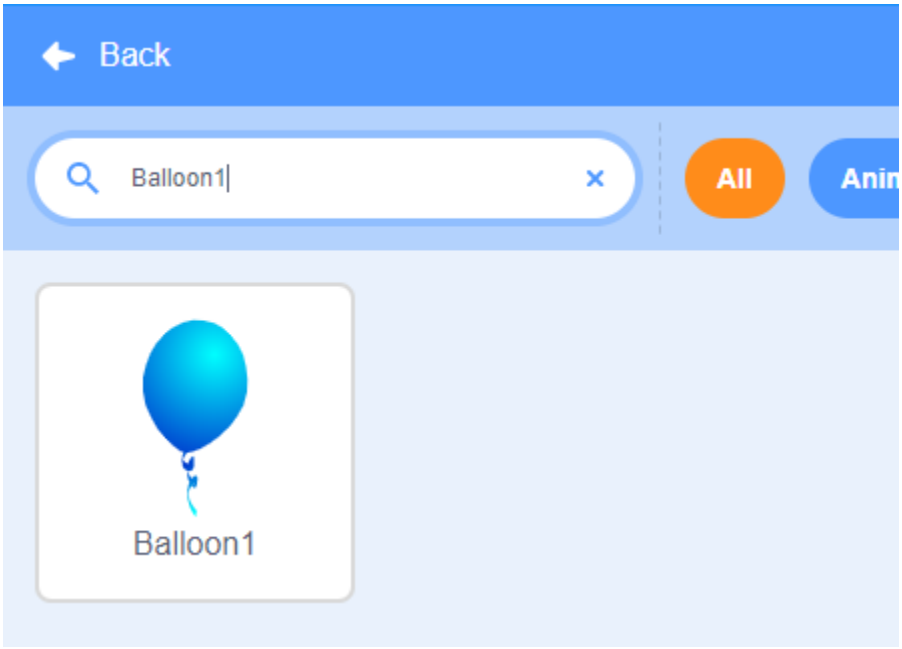
Load the Code and See What Happens

Load the code file (`1.15_inflating_the_balloon.sb3`) to Scratch 3.

By toggling Slider to the left to start to inflate the balloon, at this time the balloon will get bigger and bigger. If the balloon is too large will blow up; if the balloon is too small, it will not float into the air. You need to judge when to toggle the switch to the right to stop pumping.

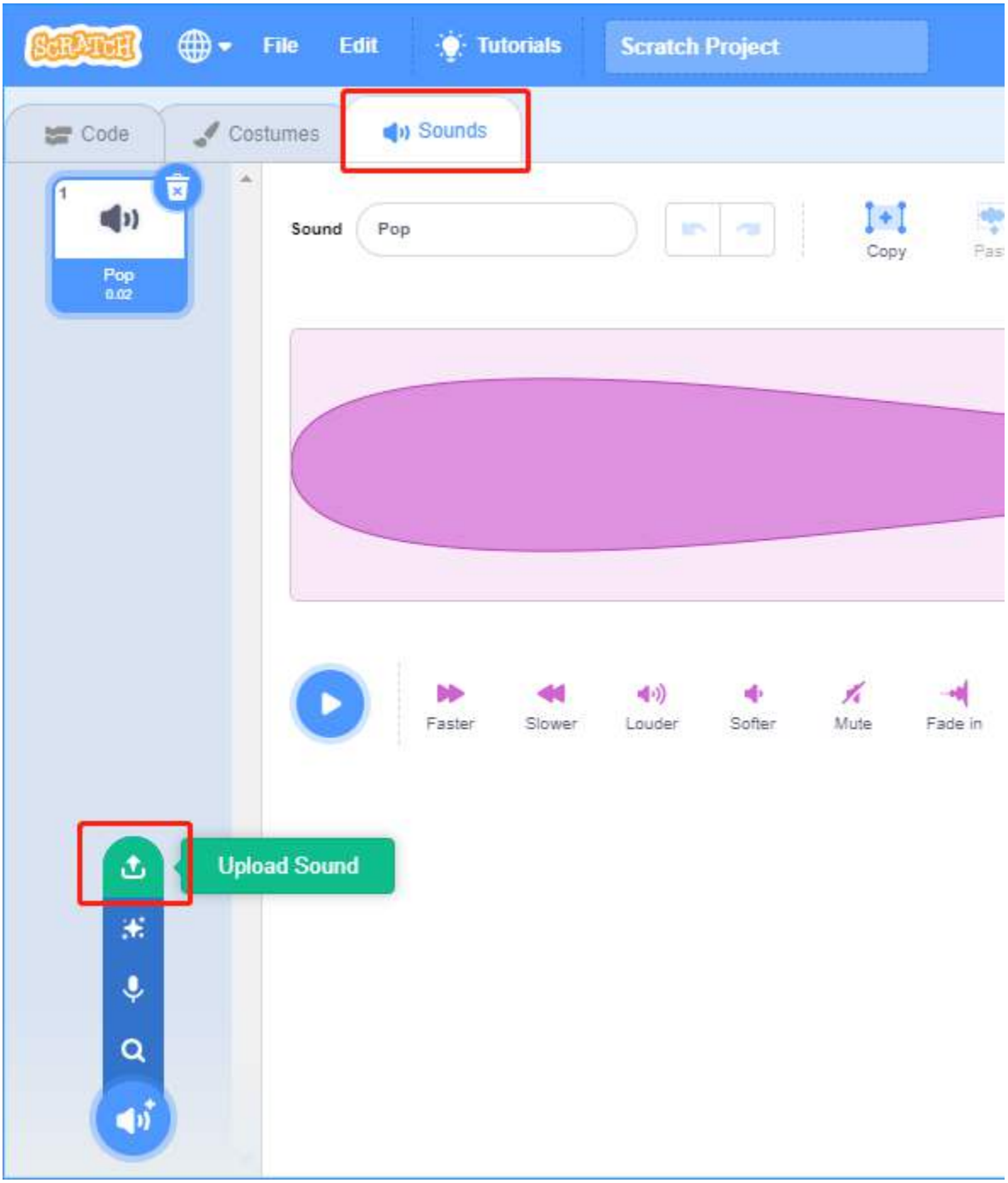
Tips on Sprite

Delete the previous Sprite1 sprite, then add the **Balloon1** sprite.

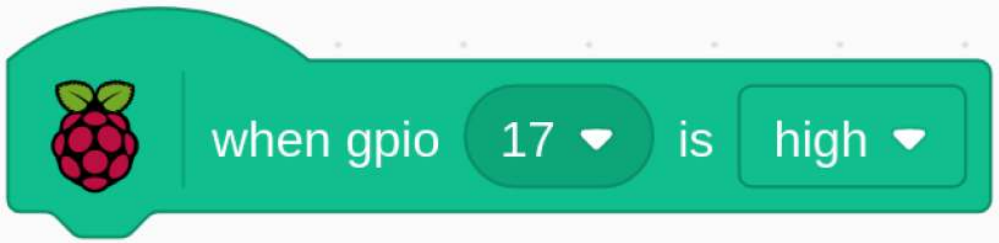


A balloon explosion sound effect is used in this project, so let's see how it was added.

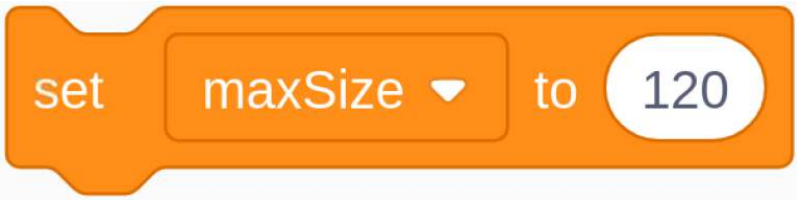
Click the **Sound** option at the top, then click **Upload Sound** to upload `boom.wav` from the `home/pi/raphael-kit/scratch/sound` path to Scratch 3.


File
Edit
Tutorials
Scratch Project
Code
Costumes
Sounds
1
Pop
0.02
Sound
Pop
Copy
Faster
Slower
Louder
Softer
Mute
Fade in
Upload Sound

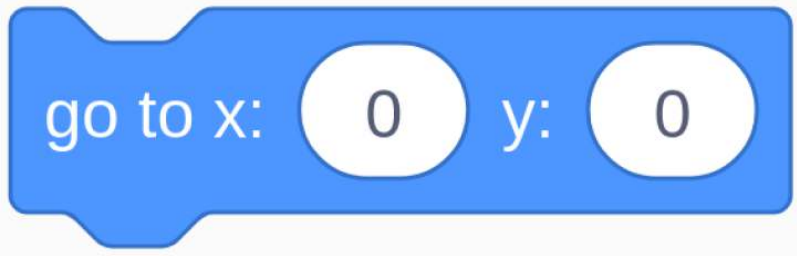
Tips on Codes



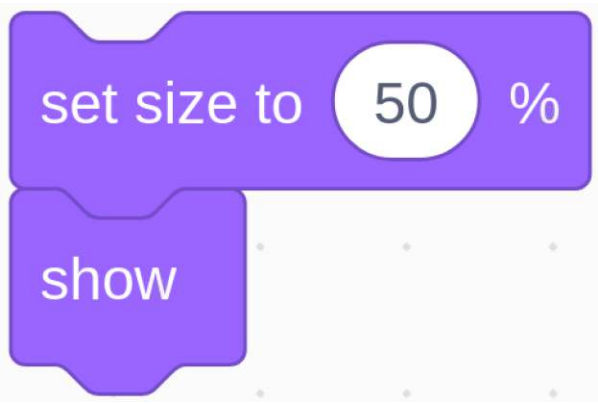
This is an event block, and the trigger condition is that gpio17 is high, that is, the switch is toggled to the left.



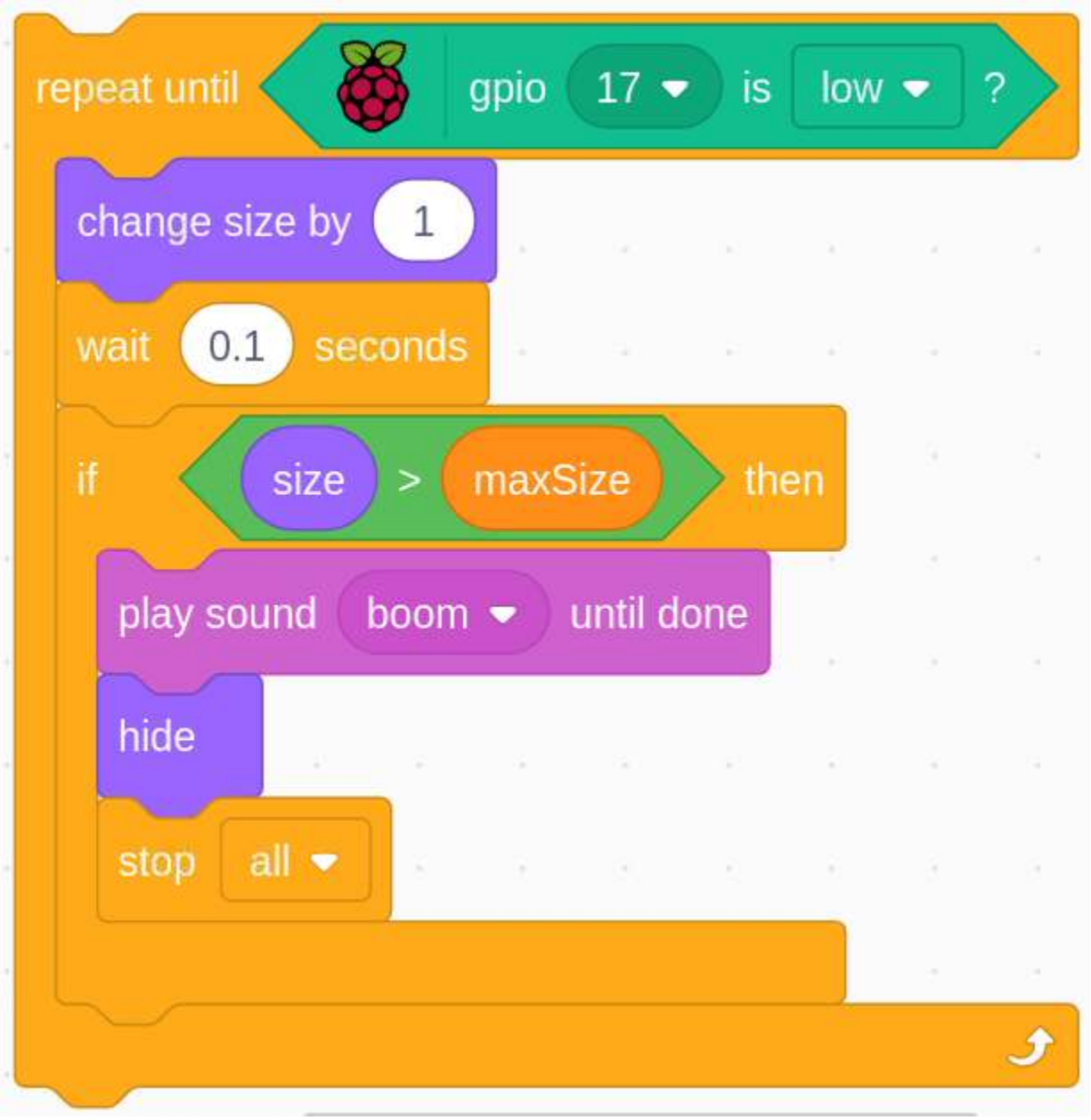
Set the size threshold of the Ballon1 sprite to 120



Move the coordinates of the Balloon1 sprite to (0,0), which is the center of the stage area.

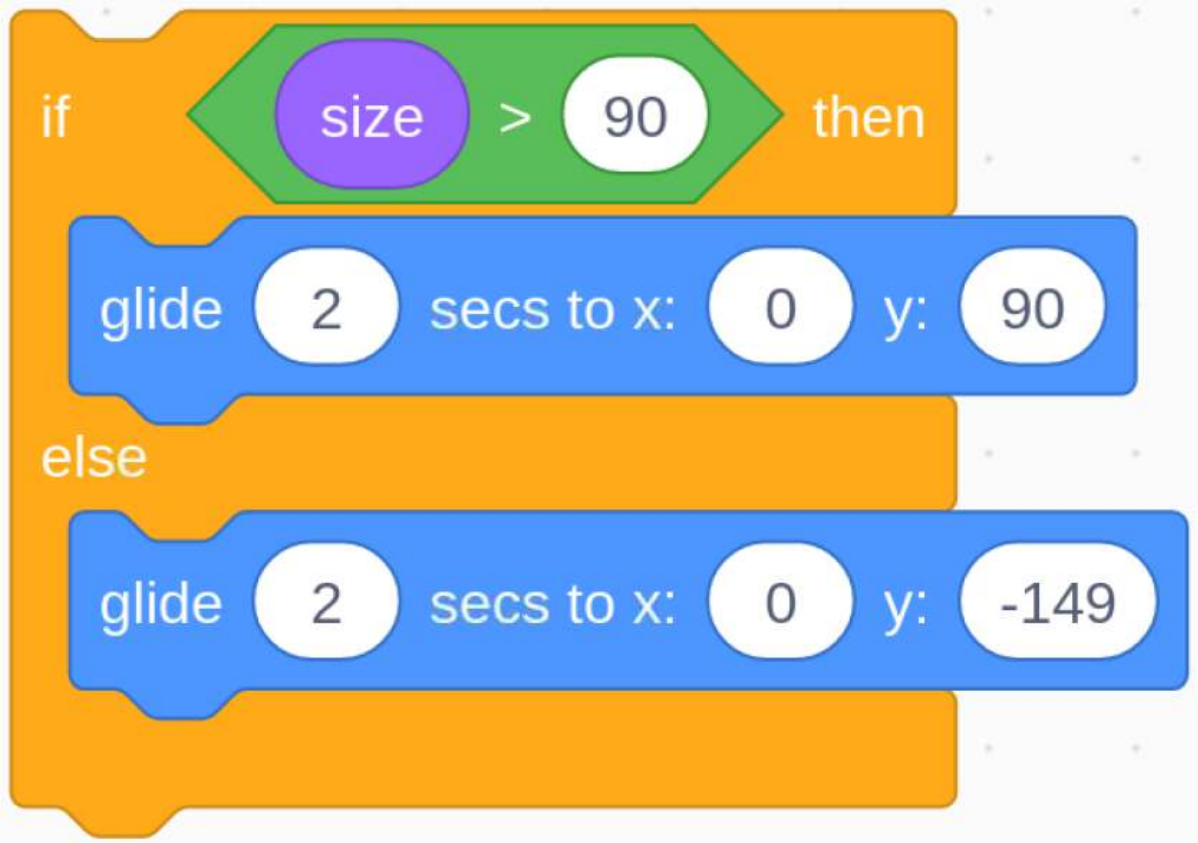


Set the size of the Balloon1 sprite to 50 and show it in the stage area.



Set up a loop to inflate the balloon, this loop stops when the slider switch is toggled to the right.

Within this loop, the balloon size is increased by 1 every 0.1s, and if it is larger than `maxSize`, the balloon will burst, at which point the boom sound is made and the code is exited.



After the last loop exits (Slider toggles to the right), determine the position of the Balloon1 sprite based on its size. If the size of the Balloon1 sprite is greater than 90, lift off (move the coordinates to (0, 90), otherwise land (move the coordinates to (0, -149).

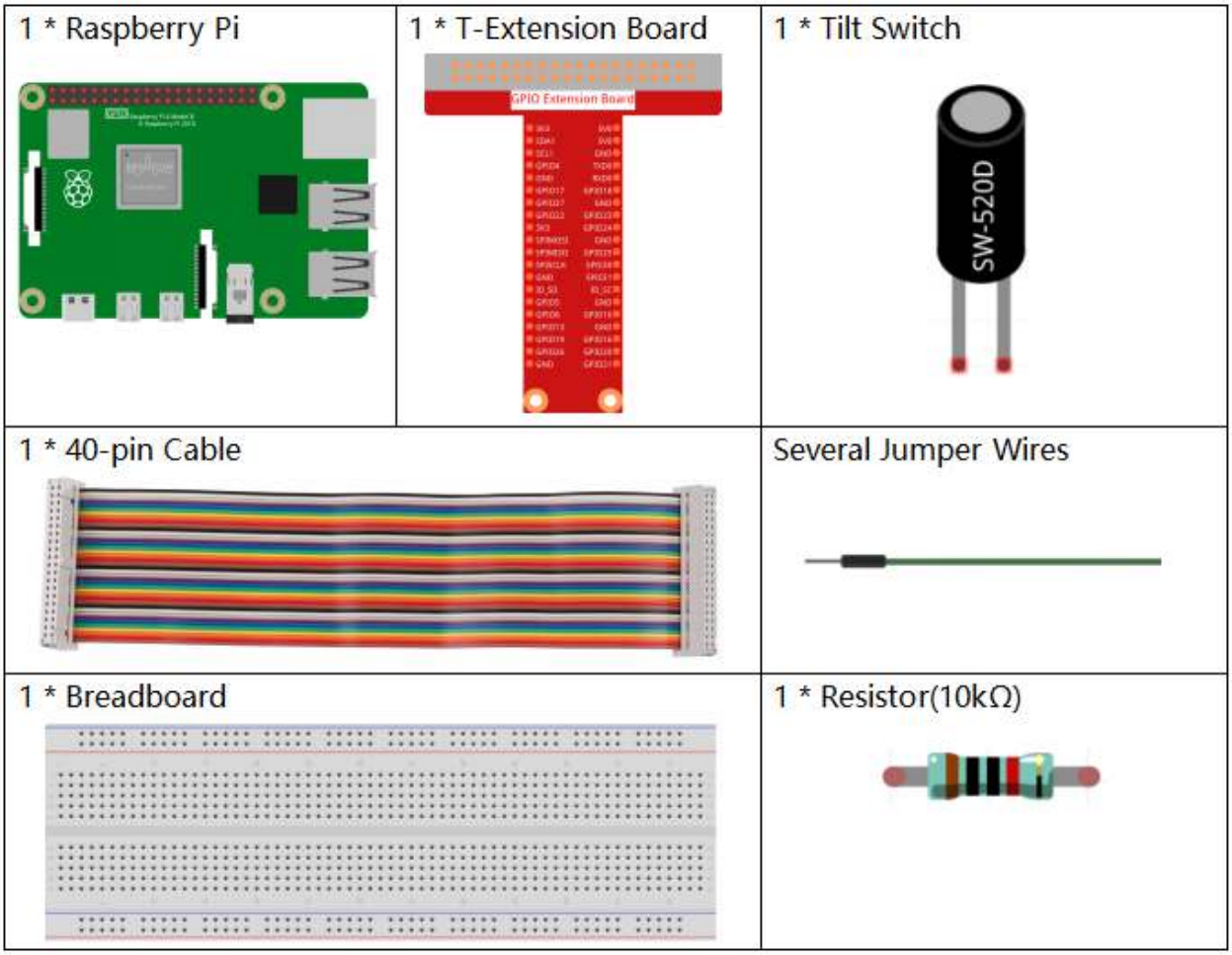
10.2.16 1.16 Fishing Game

Today, we will make a fishing game.

Observe the water on the stage area and if you find a fish on the hook, remember to tilt the switch to catch it.

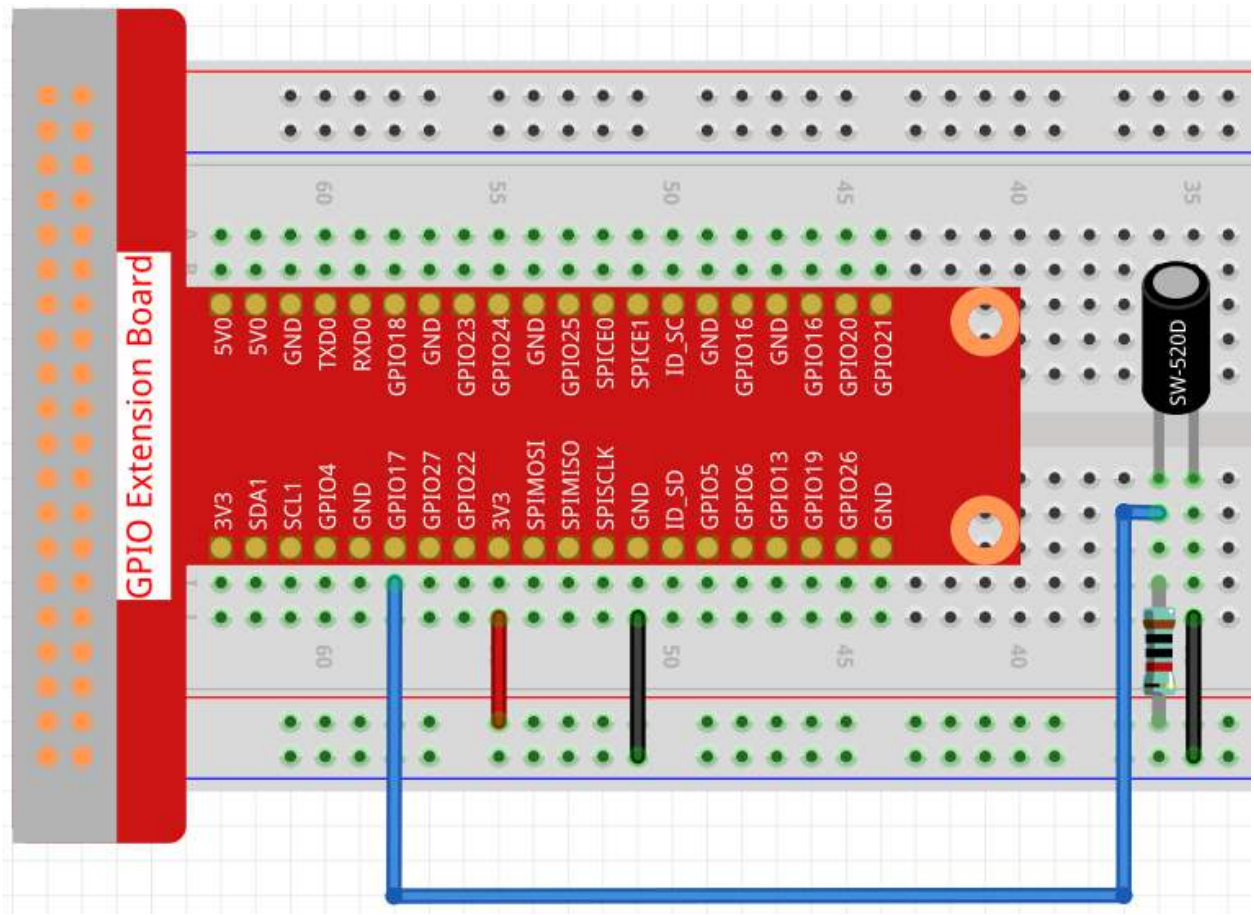
fish_status
1

Required Components



- *GPIO Extension Board*
- *Breadboard*
- *Resistor*
- *Tilt Switch*

Build the Circuit



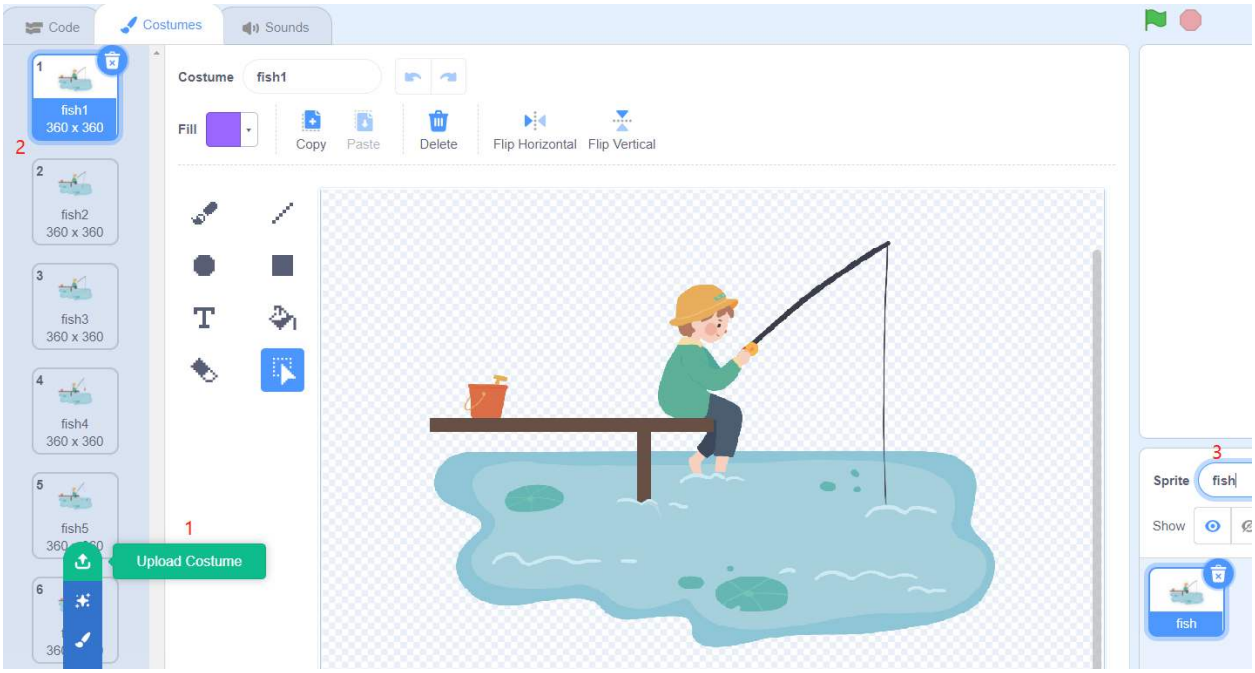
Load the Code and See What Happens

Load the code file (`1.16_fishing_game.sb3`) to Scratch 3.

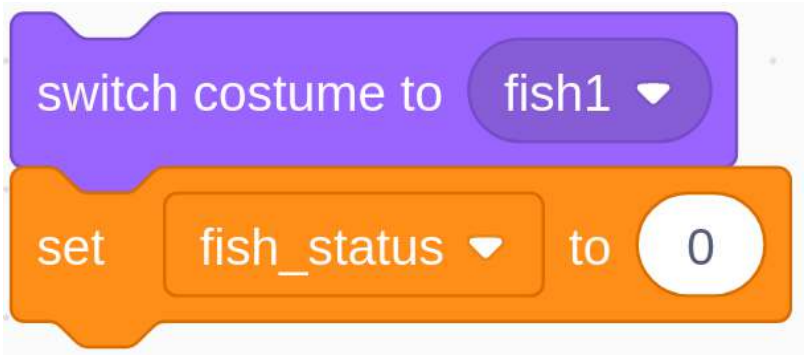
You will see a child is fishing, after a period of time when the water surface movement, you can shake the tilt switch to catch the fish. Remember, if you do not keep shaking the switch, the fish will escape.

Tips on Sprite

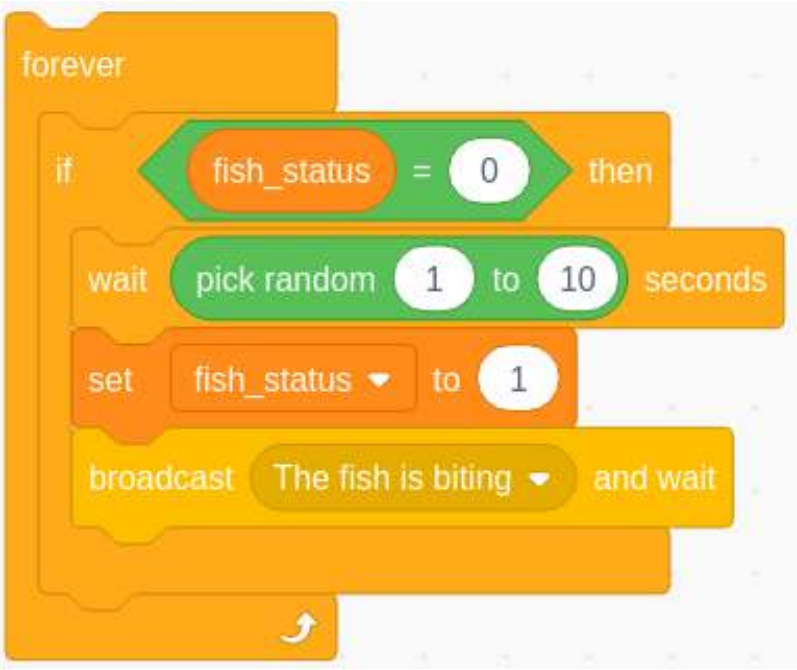
Select Sprite1, click **Costumes** in the upper left corner; upload 6 pictures (**fish1** to **fish6**) from the `home/pi/raphael-kit/scratch/picture` path via the **Upload Costume** button; delete the default 2 costumes and rename the sprite to **fish**.



Tips on Codes

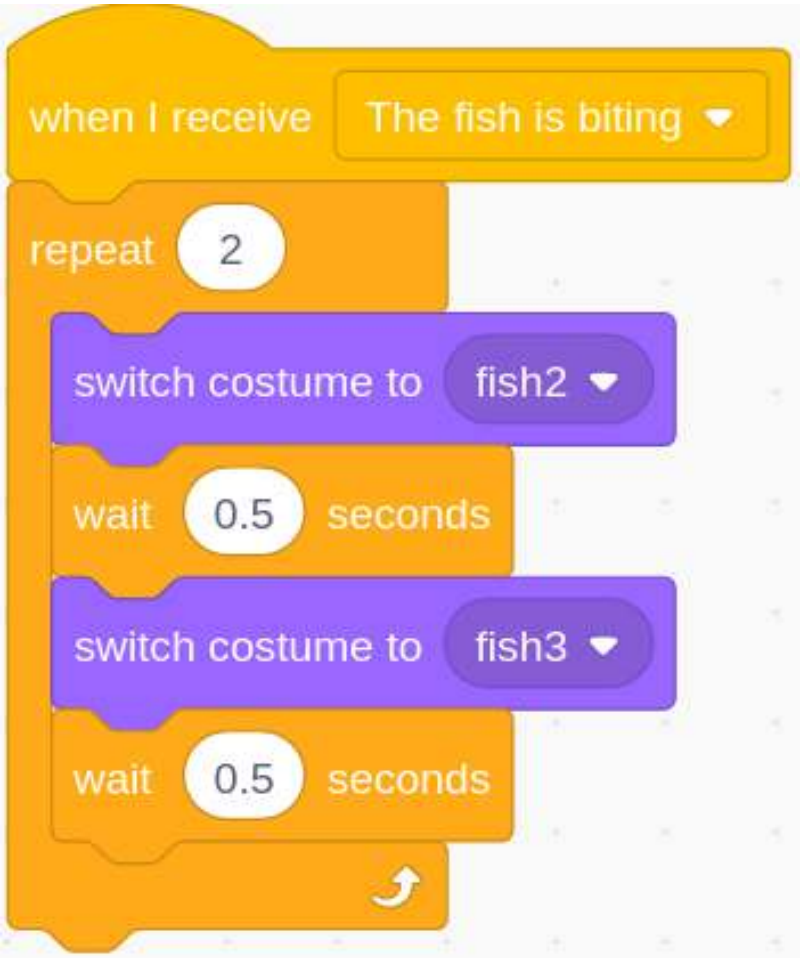


Set the initial costume of the **fish** sprite to **fish1** and assign the value of **fish_status** to 0 (when **fish_status=0**, it means the fish is not hooked, when **fish_status=1**, it means the fish is hooked).

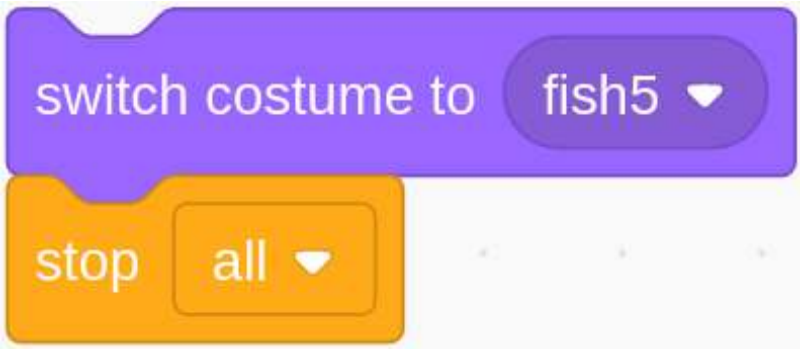


When **fish_status=0**, i.e. the fish is not hooked yet, start the fishing game. Wait for a random time from 0 to 10 seconds, then assign **fish_status** to 1, which means the fish is hooked, and broadcast a message "The fish is biting".

Note: The purpose of the broadcast block is to send a message to other code blocks or other sprites. The message can be either a request or a command.



When the message "The fish is biting" is received, let the fish sprite switch between the **fish2** and **fish3** costumes so that we can see the fish biting.

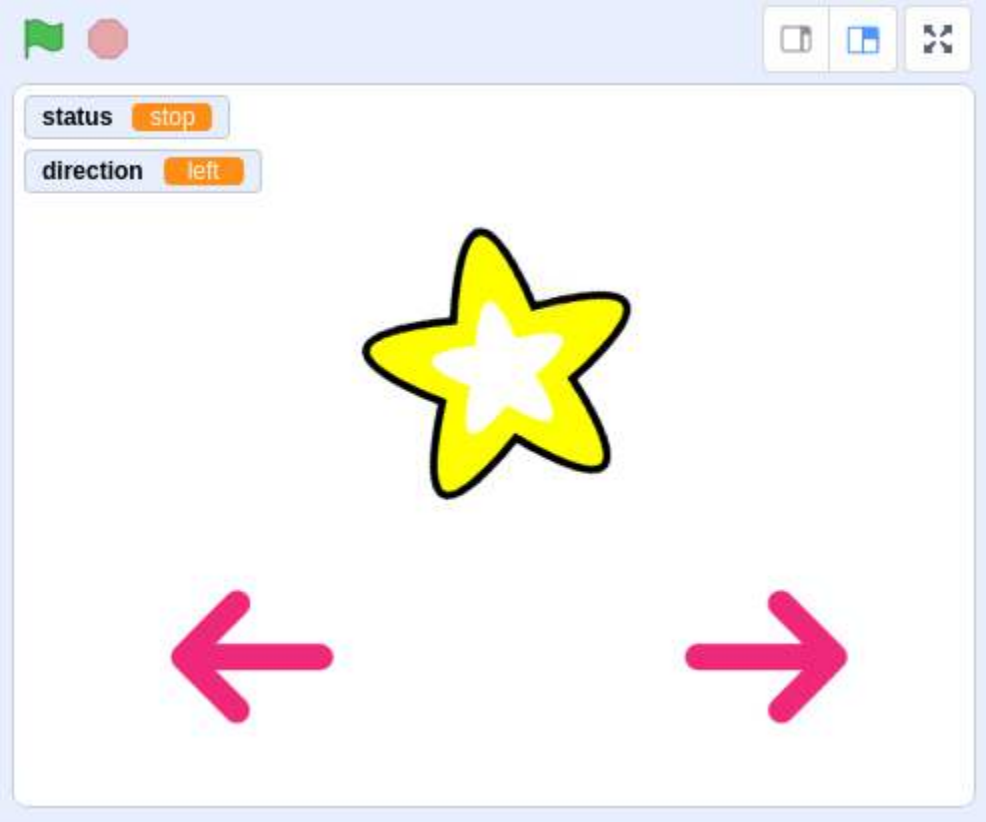


After switching the costume, if the game is not over, it means that the fish is off the hook and gone, so that we will switch the **fish** sprite costume to **fish6** (fish slipped state).

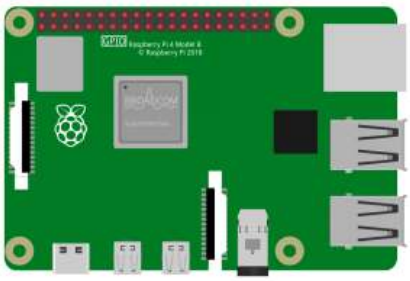
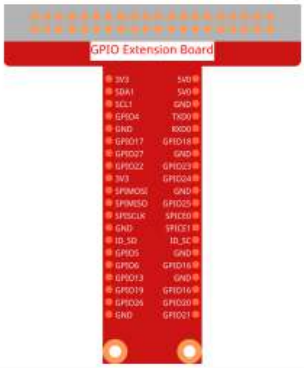
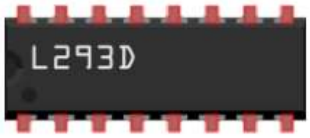
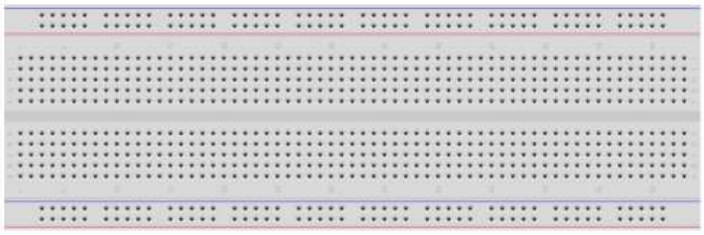
When gpio17 is high (the tilt switch is tilted), it means the fishing rod is pulled up. At this time, the value of fish_status is judged. If it is 1, it means that the fishing rod was pulled up when the fish was hooked and switched to fish4 costume (fish was caught). On the contrary, it means that the fishing rod pulled up when the fish is not hooked is switched to the fish5 costume (nothing is caught).

10.2.17 1.17 Rotating fan

In this project, we will make a spinning star sprite and fan.

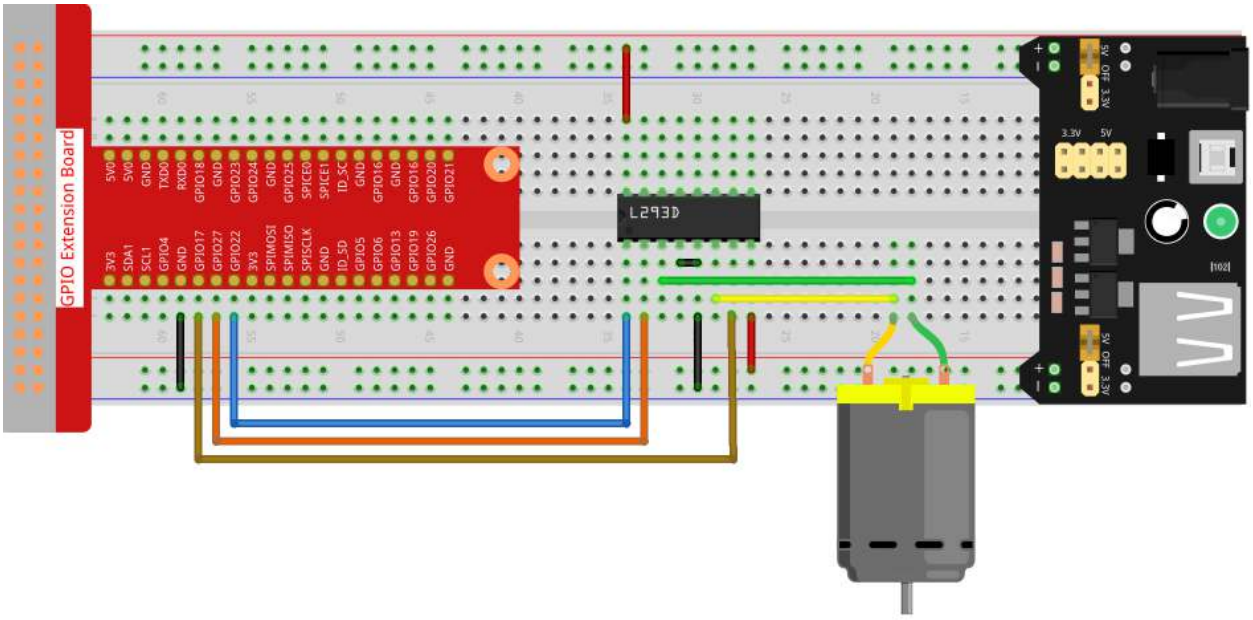

status stop
direction left

Required Components

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

- *GPIO Extension Board*
- *Breadboard*
- *Power Supply Module*
- *L293D*
- *DC Motor*

Build the Circuit



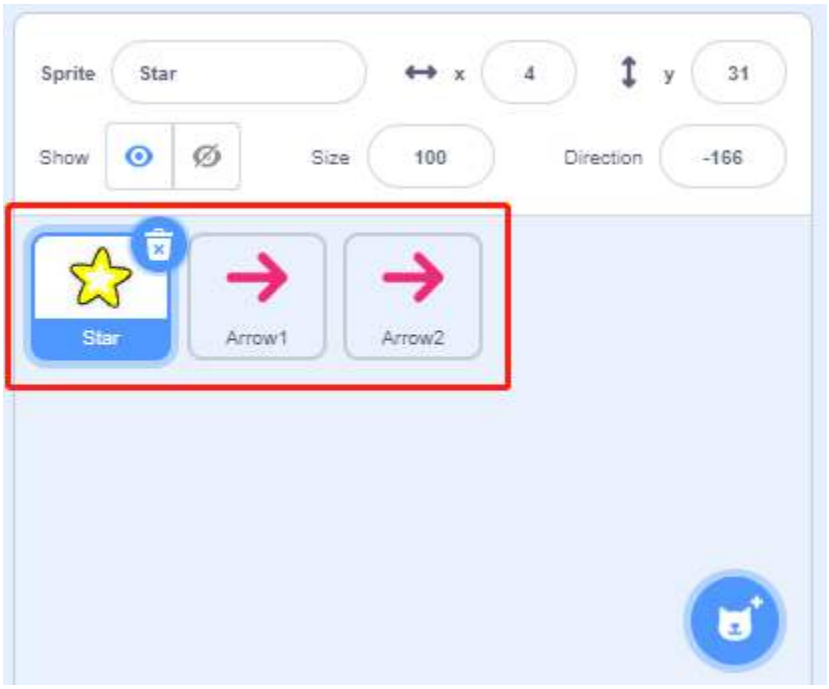
Load the Code and See What Happens

Load the code file (`1.17_rotating_fan.sb3`) to Scratch 3.

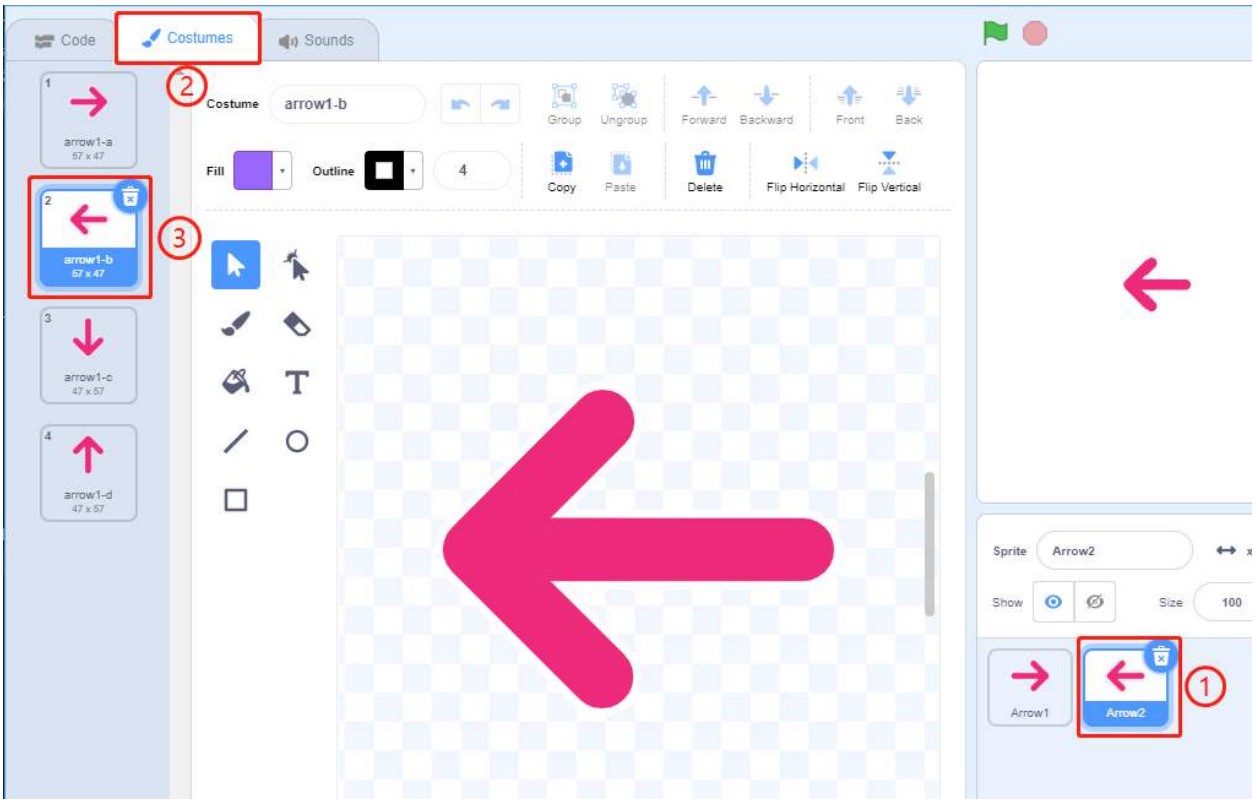
After clicking the green flag on the stage, click on the star spritethen it and the motor will rotate clockwise; you can change the direction of rotation by clicking on the two **arrow** sprites. When you click on the **star** sprite again, it and the motor will stop rotating.

Tips on Sprite

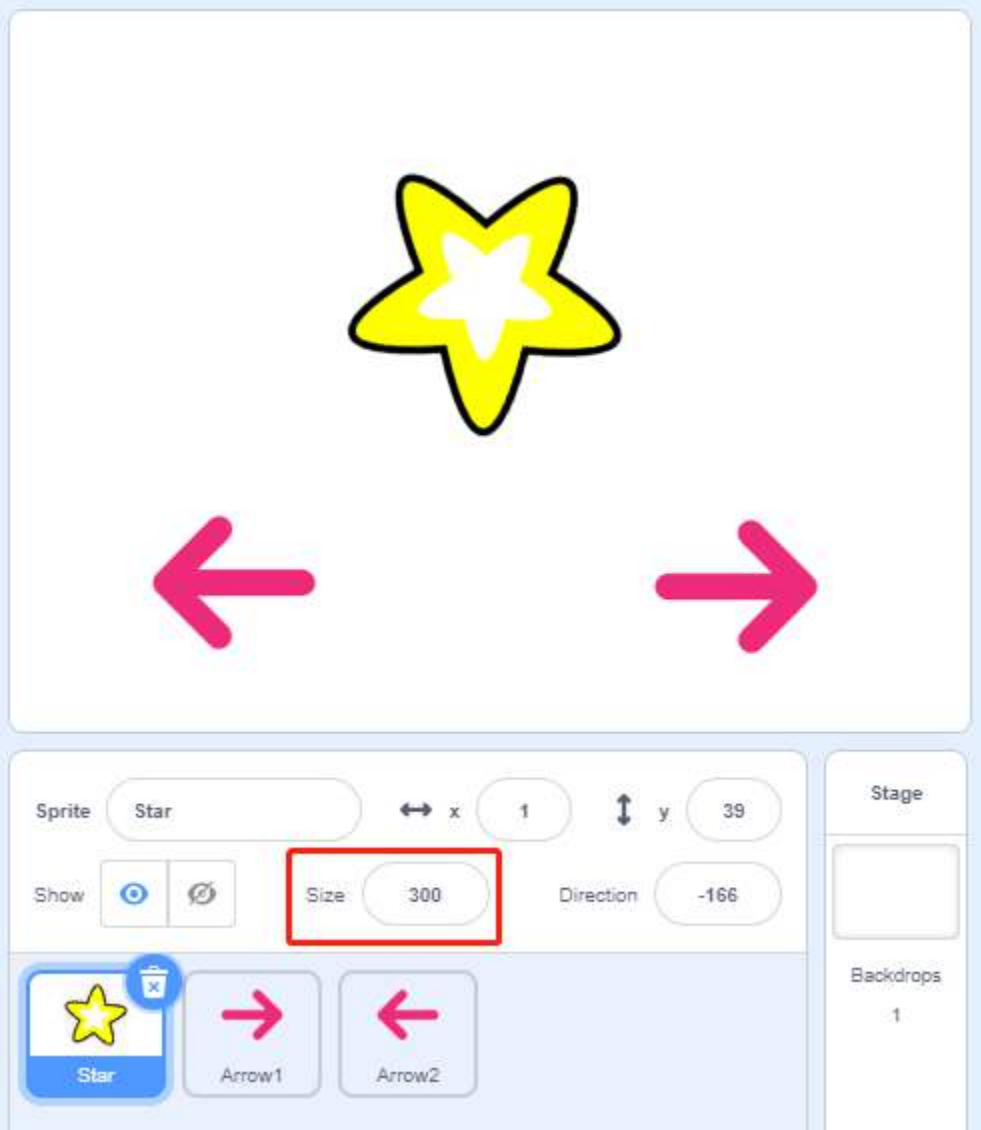
Delete the default sprite, then select the **Star** sprite and the **Arrow1** sprite, and copy Arrow1 once.



In the **Costumes** option, change the Arrow2 sprite to a different direction costume.

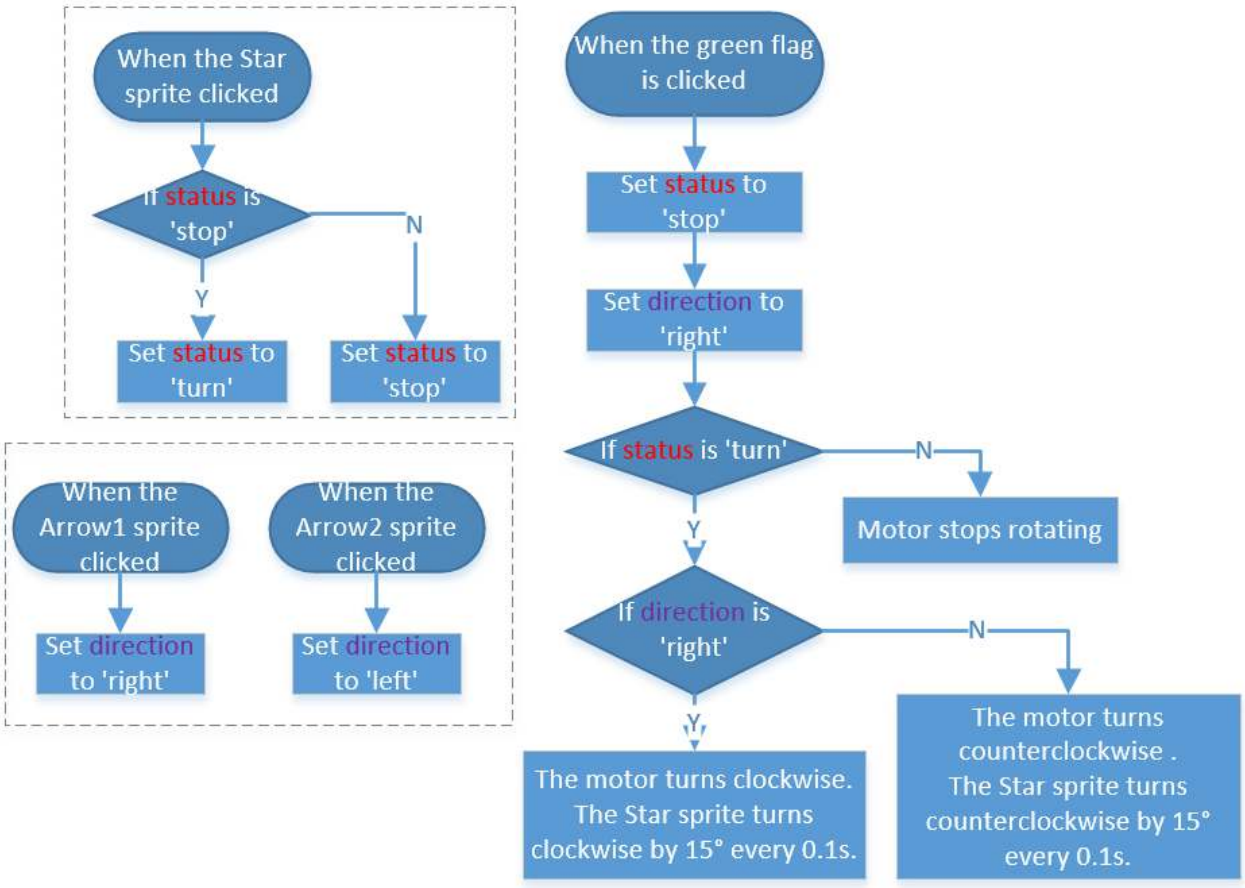


Adjust the size and position of the sprite appropriately.



Tips on Codes

Flow Chart

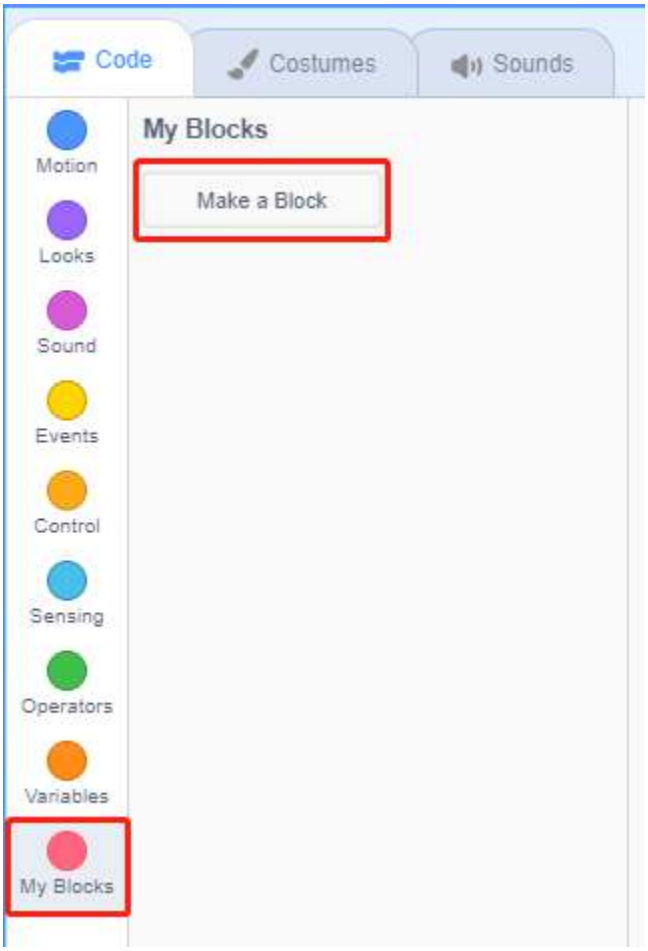


In this code, you will see 2 pink blocks, turn left and turn right, which are our custom blocks (functions).

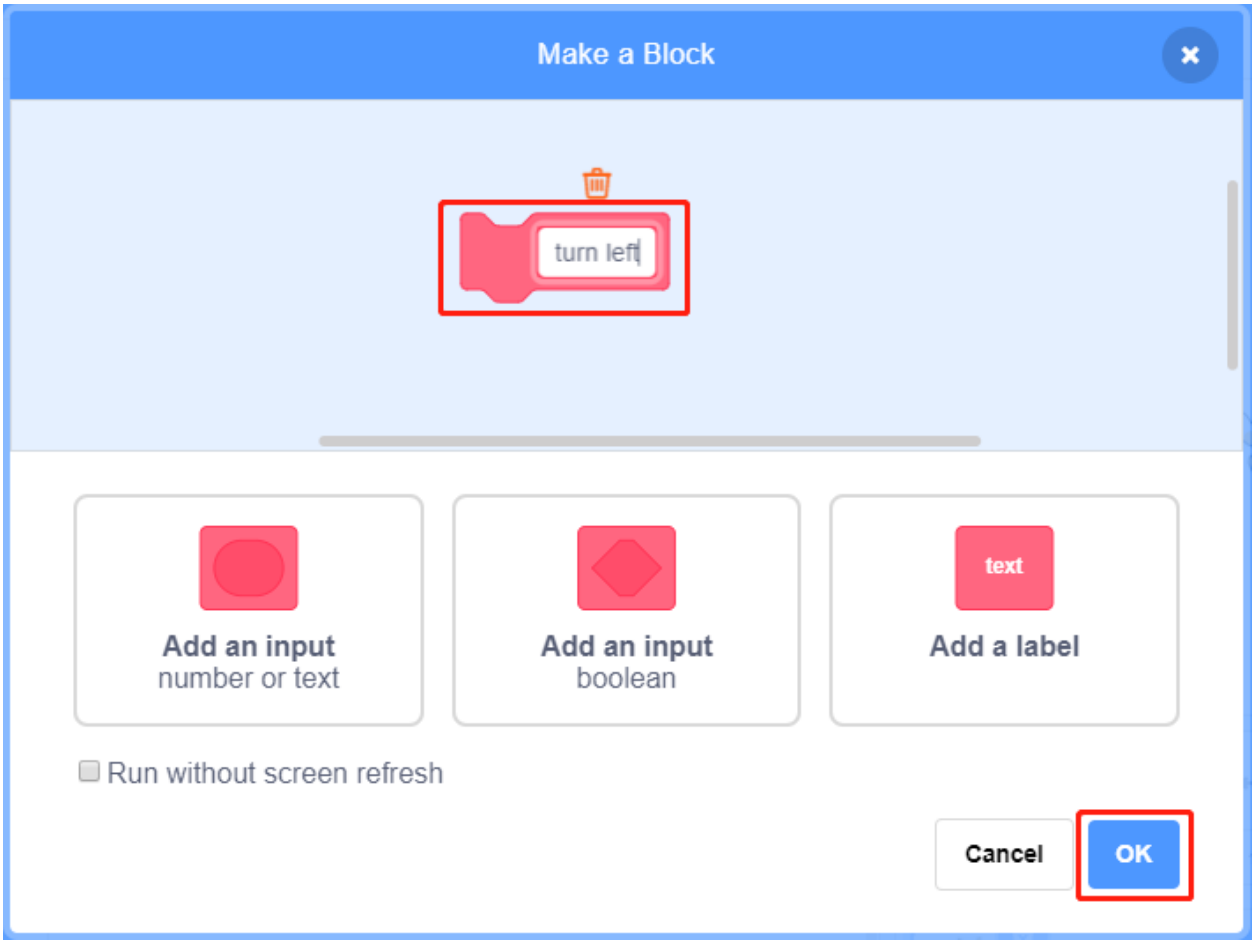
How to Make a Block?

Let's learn how to make a block (function). The block (function) can be used to simplify your program, especially if you perform the same operation multiple times. Putting these operations into a newly declared block can be very convenient for you.

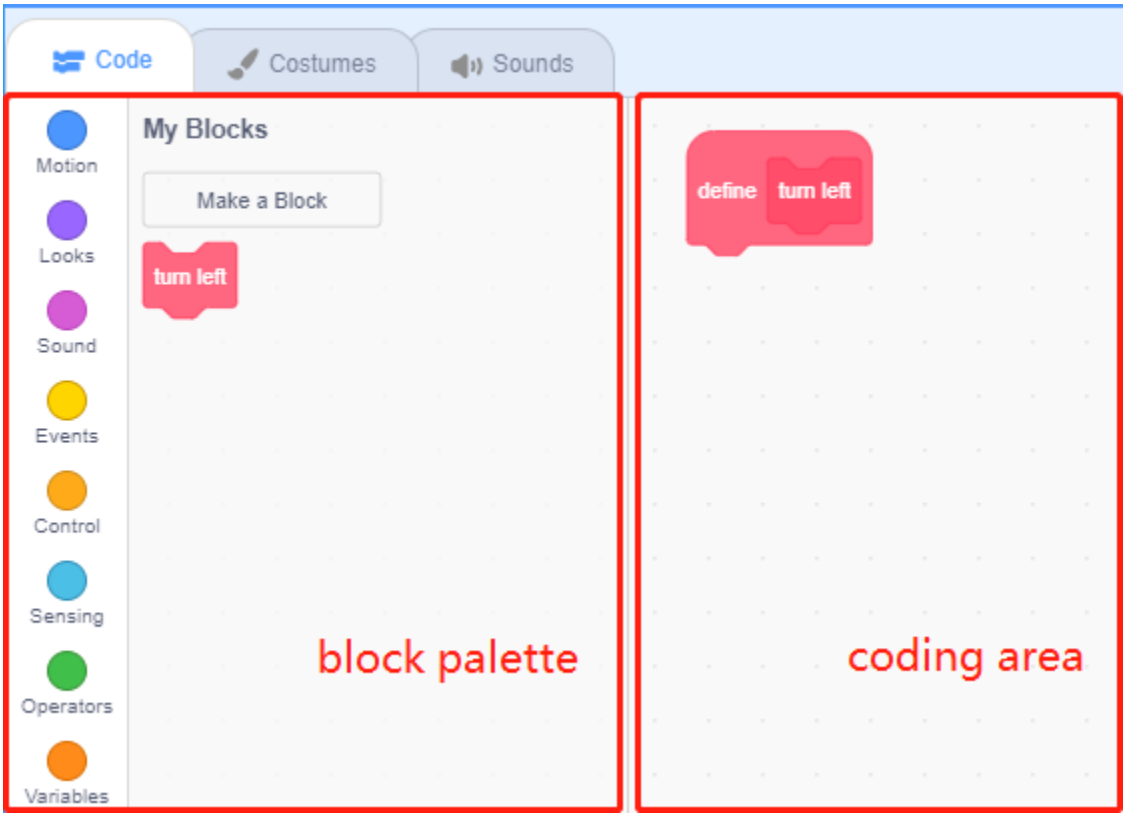
First find **My Blocks** in the block palette, then select **Make a Block**.



Enter the name of the new block.

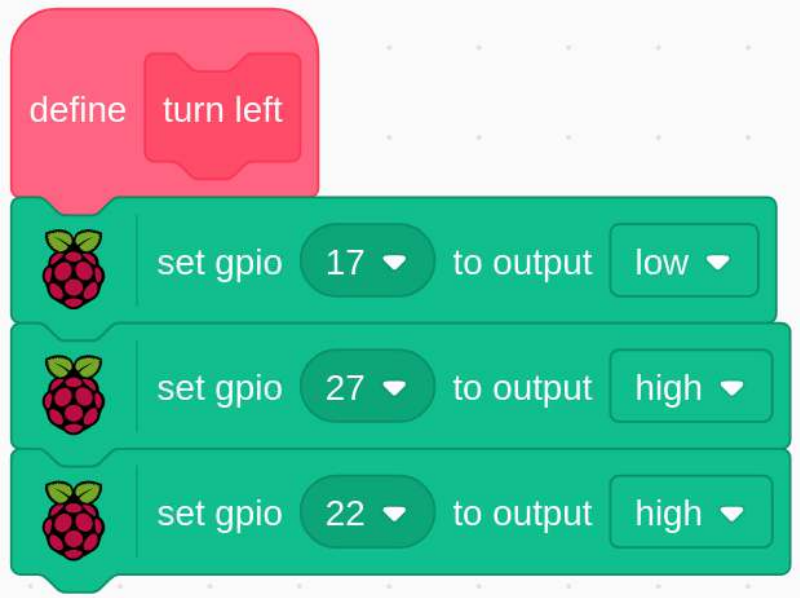


After writing the function of the new block in the coding area, save it and then you can find the block in the blocks palette.



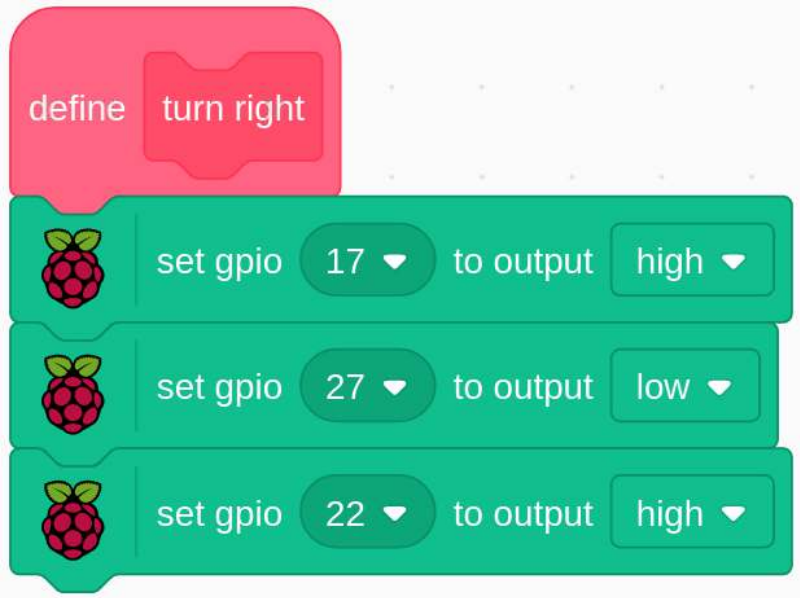
turn left

This is the code inside the turn left block to make the motor rotate counterclockwise.



turn right

This is the code inside the turn right block to make the motor rotate clockwise.



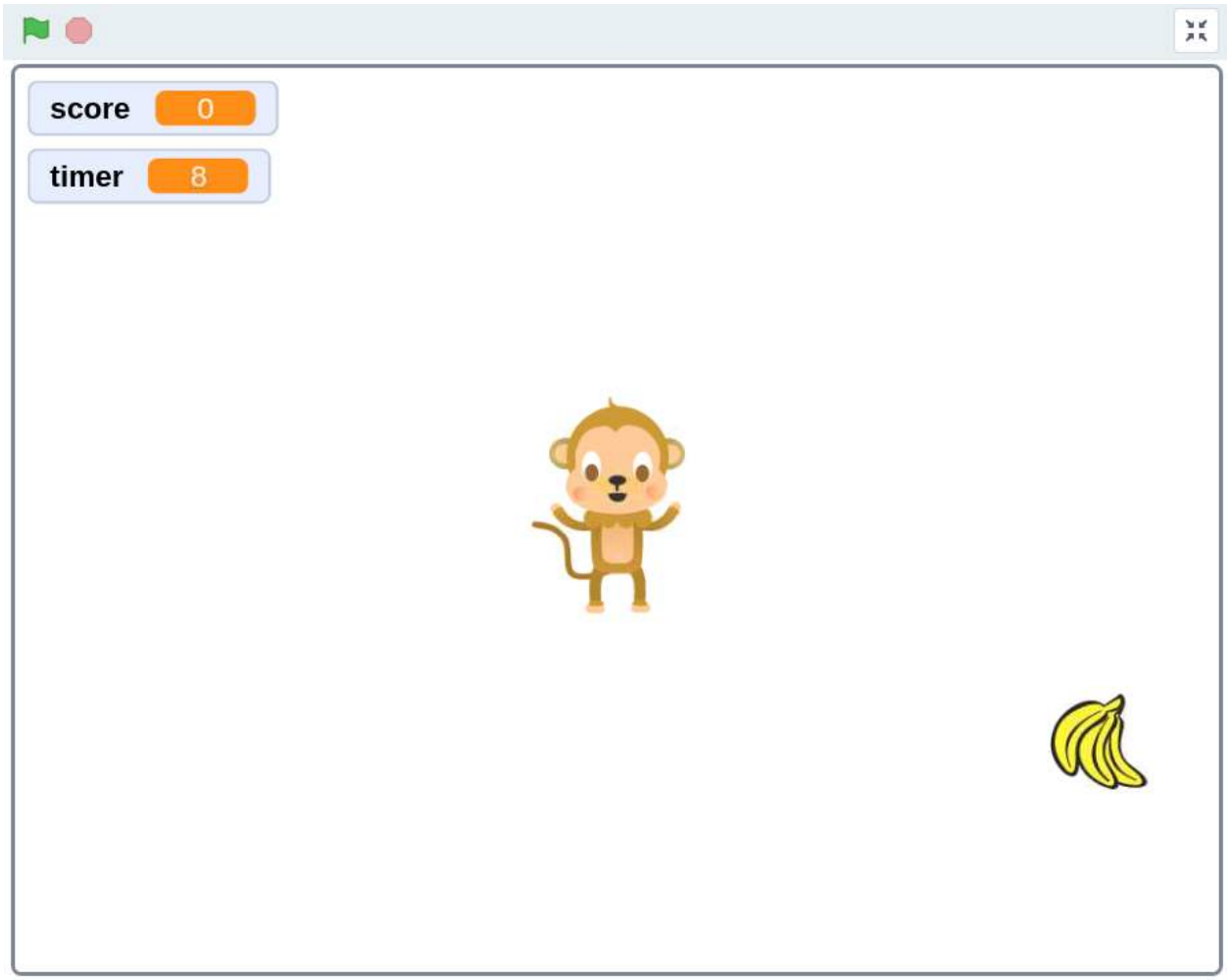
10.2.18 1.18 Eating Banana Game

Description

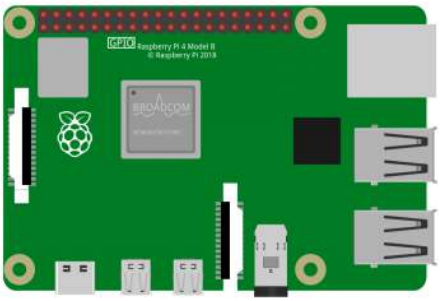
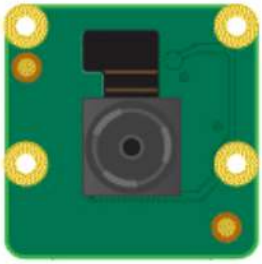
Scratch has a Video Sensing expansion module, which can turn on the camera in Scratch and detect the movement of objects on the camera screen.

Today, we will use the camera to make a eating banana game. In the stipulated time, help the Monkey eat more bananas.

To play the game against a white background, click on the green flag to start. Move colored objects in front of the camera to control the Monkey sprite.

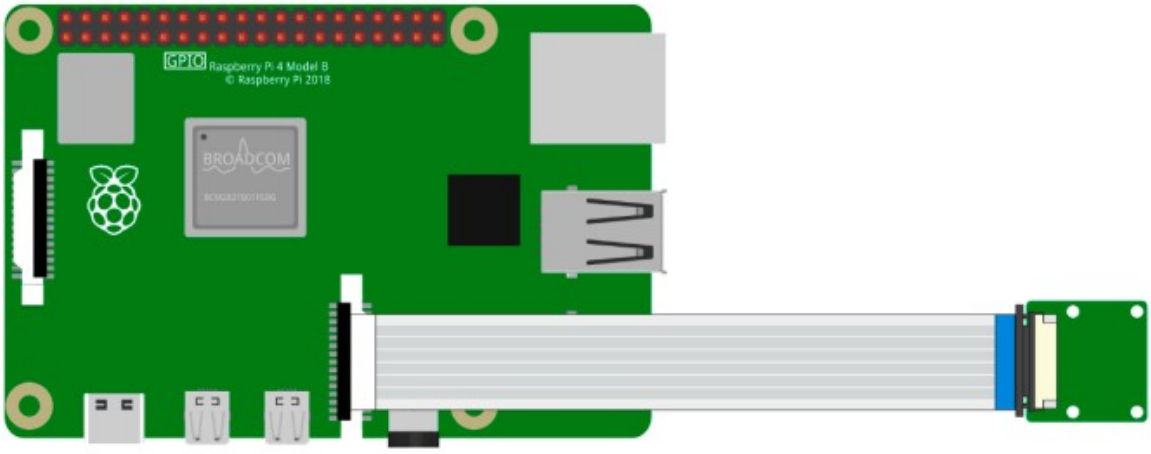

score
0
timer
8

Required Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * Camera Module
  </td></tr>
<tr><td colspan="2">1 * FFC Cable
 </td></tr>
</table>

- *Camera Module*

Build the Circuit



Note: You need to refer to *Camera Module* to connect the camera module and enable the Raspberry Pi camera interface.

Load the Code and See What Happens

Load the code file (`1.18_eating_banana_game.sb3`) to Scratch 3.

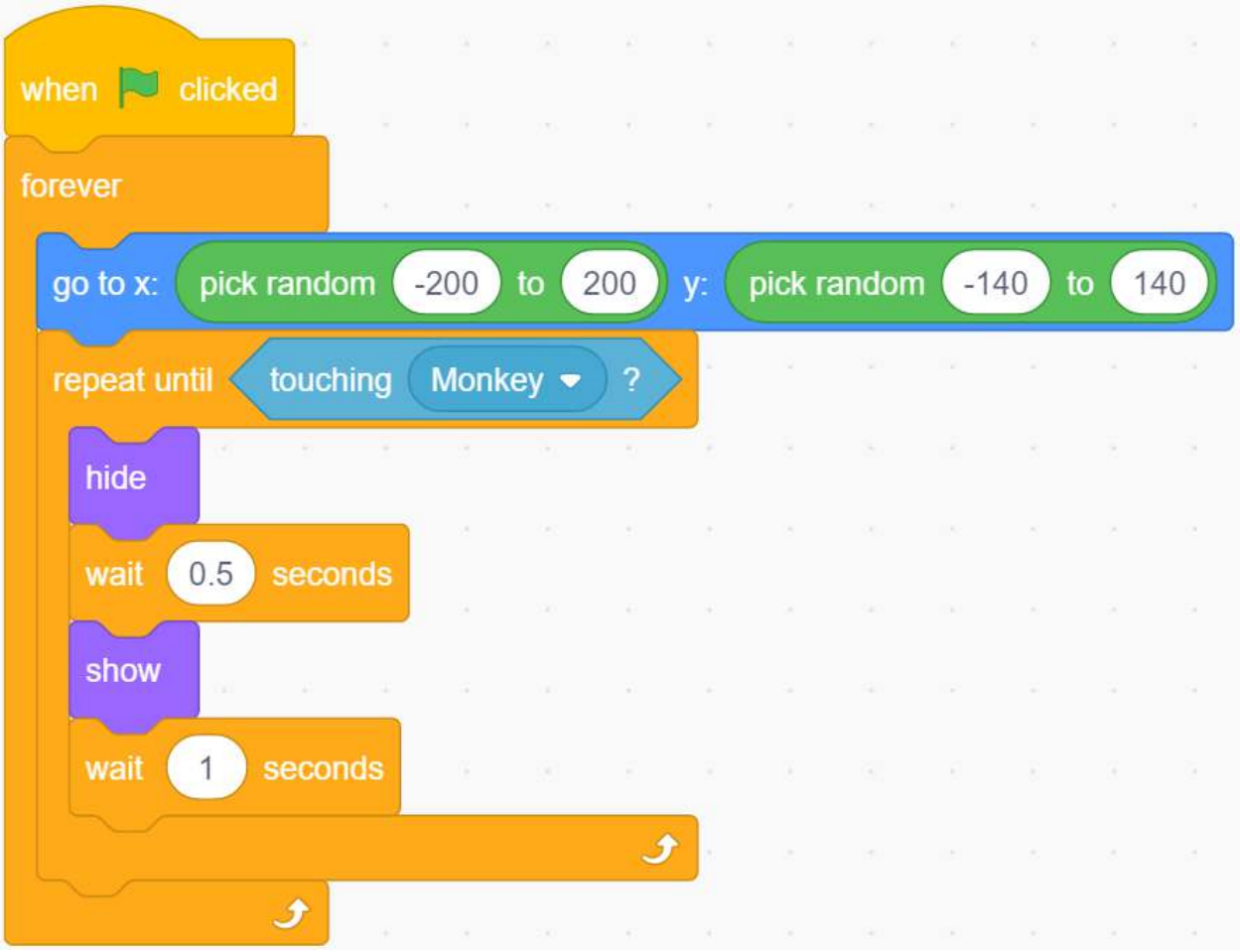
Tips on Codes

Arrange monkeys and bananas

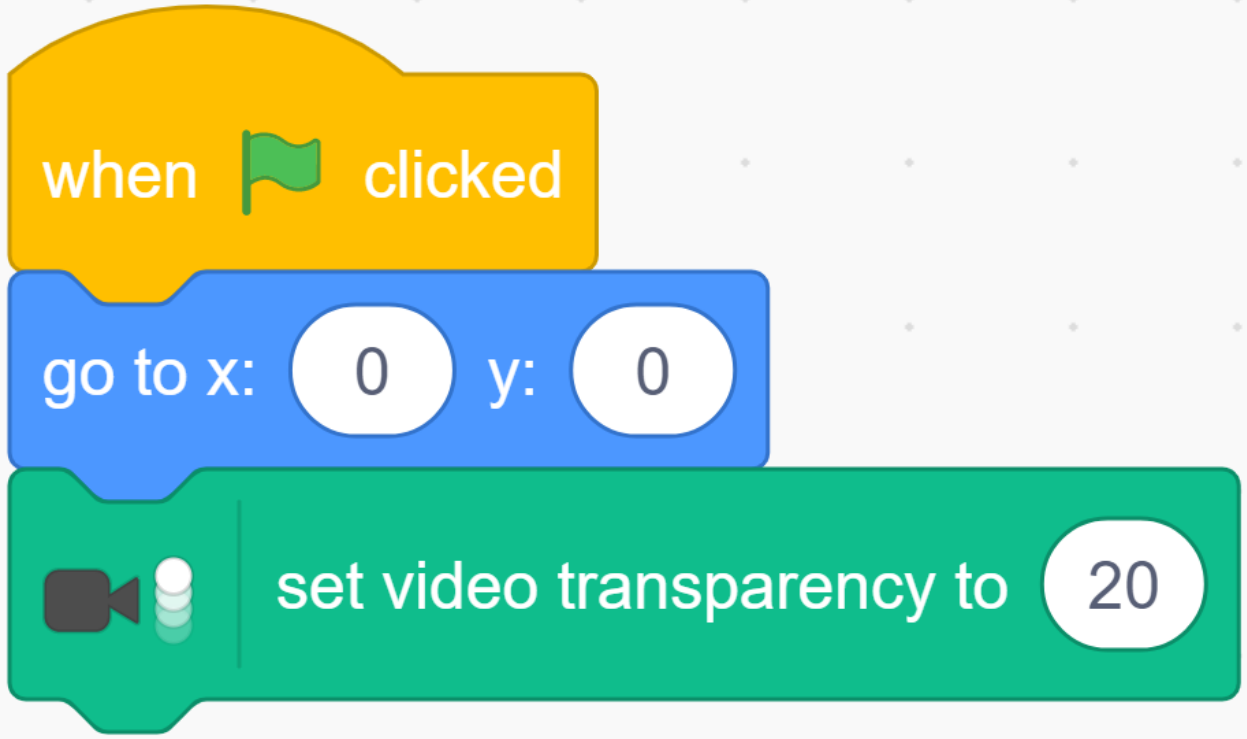
First, we delete the original sprite, then add Monkey sprite and Bananas sprite, and change their sizes to 50.

Let Bananas appear randomly.

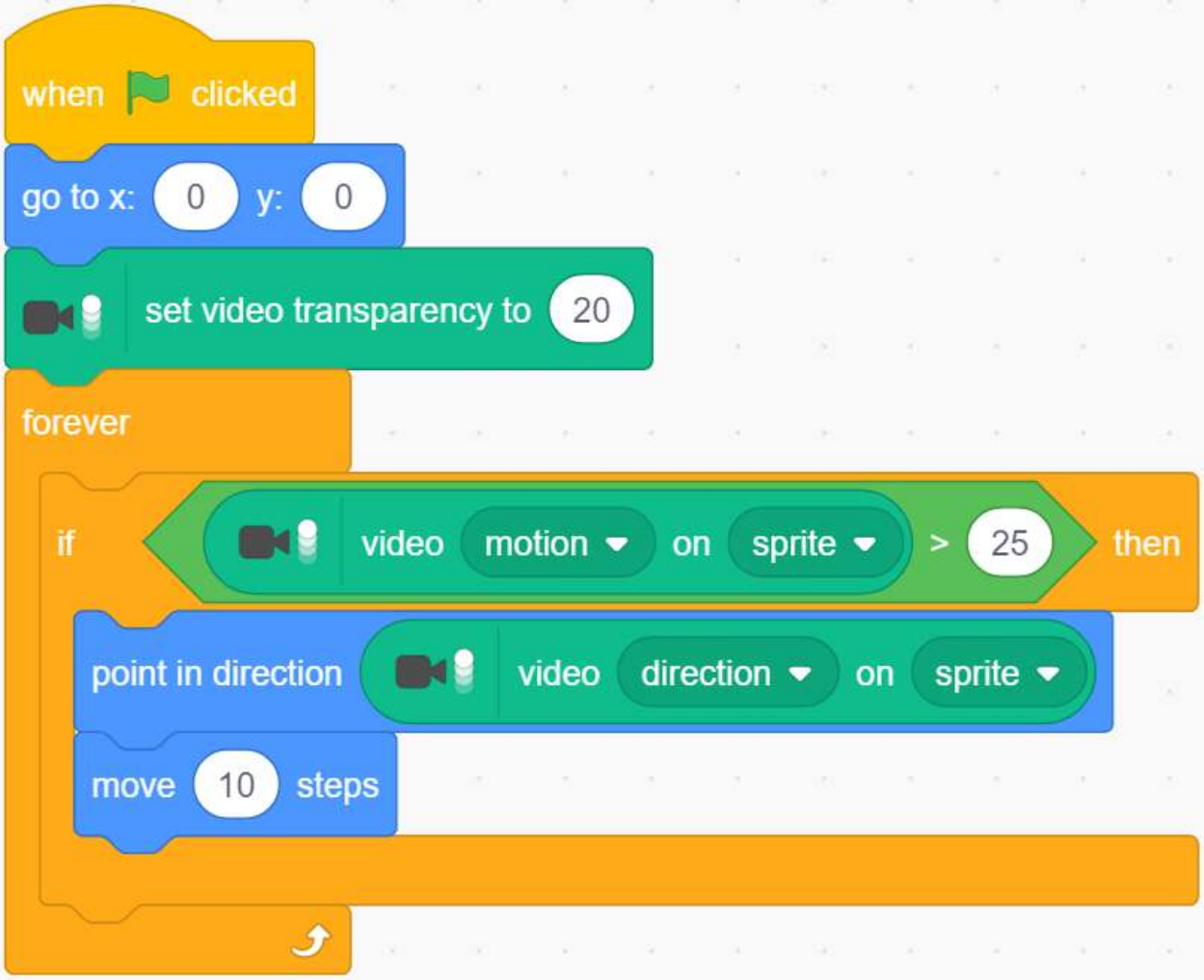
Bananas disappears after encountering the Monkey, which means it was eaten by the Monkey and reappears randomly.



Let the Monkey appear in the center of the stage and initialize the camera data (transparency is set to 20).



If the camera detects an object moving, let the Monkey move towards the object.



Now, click on the green flag at the top of the stage area to start the game.

Let the Monkey eat bananas, it is very hungry! Try to play this game on a white background to prevent interference from other objects.

Challenge

I believe that you will be smart enough to program and implement this game soon. Next, we will add some challenges to enrich our game content.

· When Monkey eats banana, we add 1 to the score. Within 30s, see who has the highest score!

· When Monkey eats a banana, it emits a suitable sound effect.

CHAPTER

ELEVEN

APPENDIX

11.1 Install the Libraries

11.1.1 For C User

BCM2835

This is a C library for Raspberry Pi (RPi). It provides access to GPIO and other IO functions on the Broadcom BCM 2835 chip, as used in the RaspberryPi, allowing access to the GPIO pins on the 26 pin IDE plug on the RPi board so you can control and interface with various external devices.

It provides functions for reading digital inputs and setting digital outputs, using SPI and I2C, and for accessing the system timers. Pin event detection is supported by polling (interrupts are not supported).

Works on all versions upt to and including RPI 4. Works with all versions of Debian up to and including Debian Buster 10.

Open a terminal and download the `bcm2835` library to the `/home/pi` path.

```
cd /home/pi
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.69.tar.gz
```

Unzip the package.

```
tar zxvf bcm2835-1.69.tar.gz
```

Install the BCM2835 library with the following commands.

```
cd bcm2835-1.69
./configure
make
sudo make check
sudo make install
```

- Reference: bcm2835

11.1.2 For Python User

Luma.LED_Matrix

This is a Python 3 library interfacing LED matrix displays with the MAX7219 driver (using SPI), WS2812 (NeoPixels, inc Pimoroni Unicorn pHat/Hat and Unicorn Hat HD) and APA102 (DotStar) on the Raspberry Pi and other Linux-based single board computers.

Install the dependencies for library first with:

```
sudo usermod -a -G spi,gpio pi
sudo apt install build-essential python3-dev python3-pip libfreetype6-dev libjpeg-dev␣
↪libopenjp2-7 libtiff5
```

Note: warning

The default pip and setuptools bundled with apt on Raspbian are really old, and can cause components to not be installed properly. Make sure they are up to date by upgrading them first:

```
sudo -H pip install --upgrade --ignore-installed pip setuptools
```

Proceed to install latest version of the luma.led_matrix library directly from PyPI:

```
sudo python3 -m pip install --upgrade luma.led_matrix
```

- Reference: Luma.LED_Matrix

Spidev and MFRC522

The `spidev` library helps handle interactions with the SPI and is a key component to this tutorial as we need it for the Raspberry Pi to interact with the RFID RC522.

Run the following command to install `spidev` to your Raspberry Pi via `pip`.

```
sudo pip3 install spidev
```

Continue to install the MFRC522 library.

```
sudo pip3 install mfrc522
```

The MFRC522 library contains two files: `MFRC522.py` and `SimpleMFRC522.py`.

Among them `MFRC522.py` is the realization of RFID RC522 interface, this library handles all the heavy work of communicating with RFID through Pi's SPI interface.

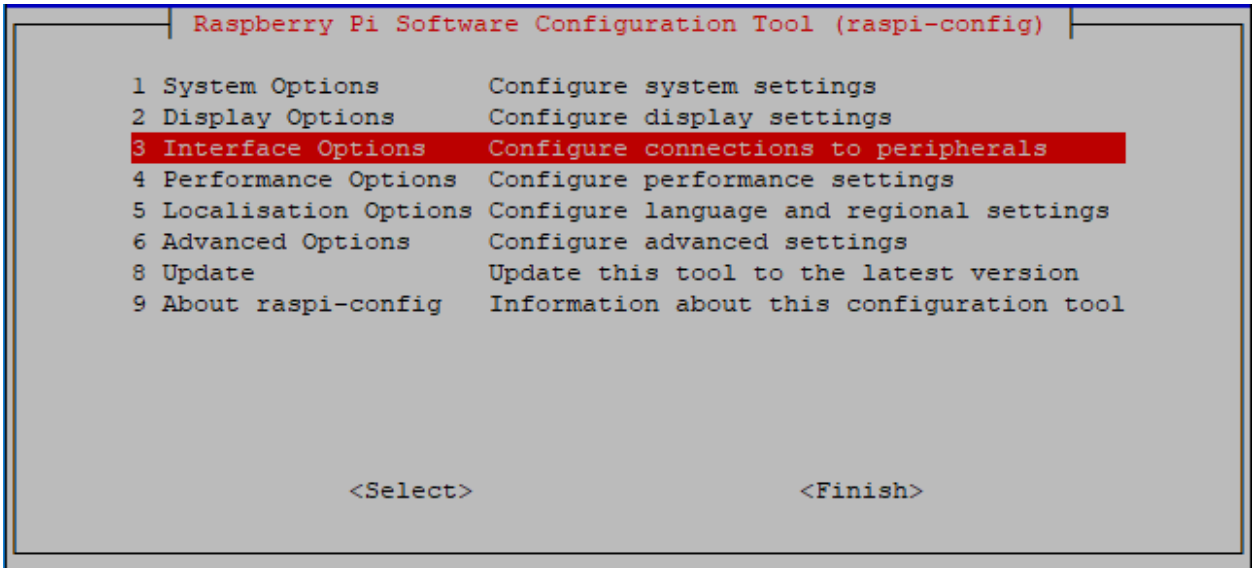
`SimpleMFRC522.py` takes the `MFRC522.py` file and greatly simplifies it by allowing you to deal with only a few functions instead of a few functions.

11.2 I2C Configuration

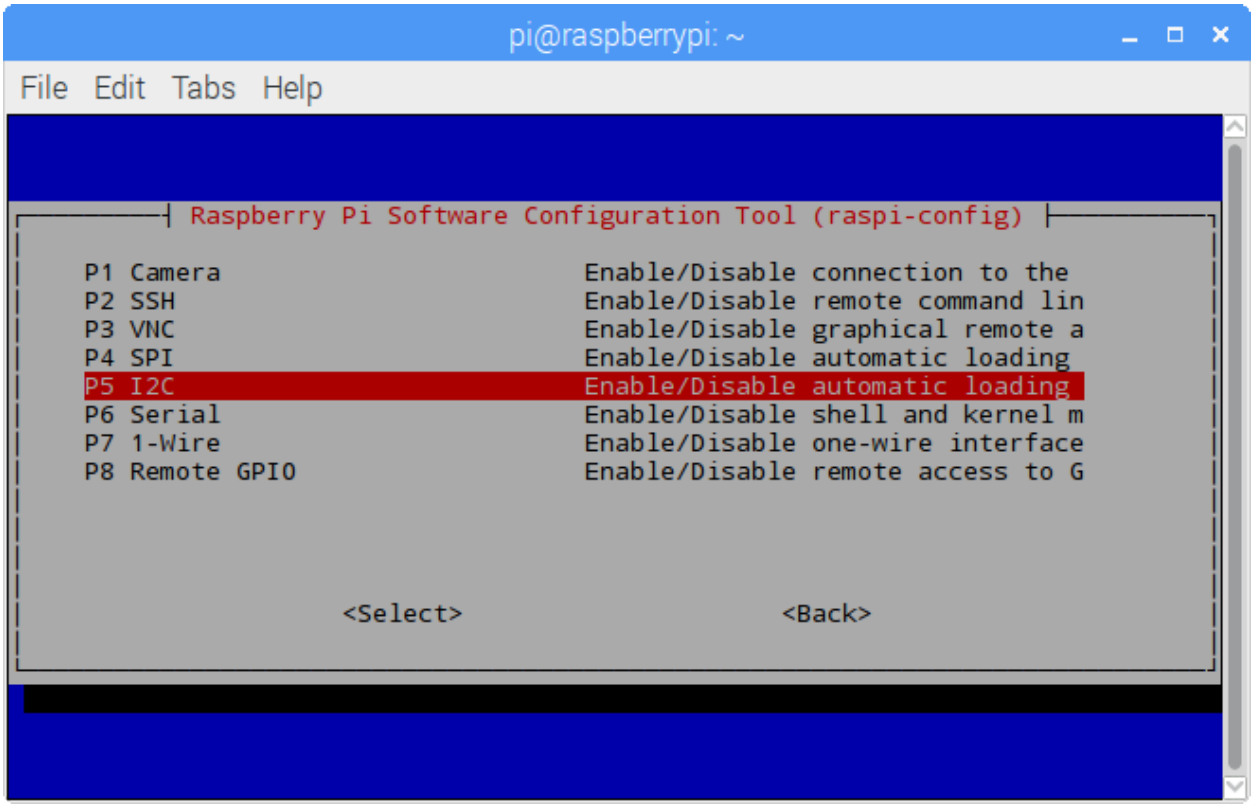
Step 1: Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

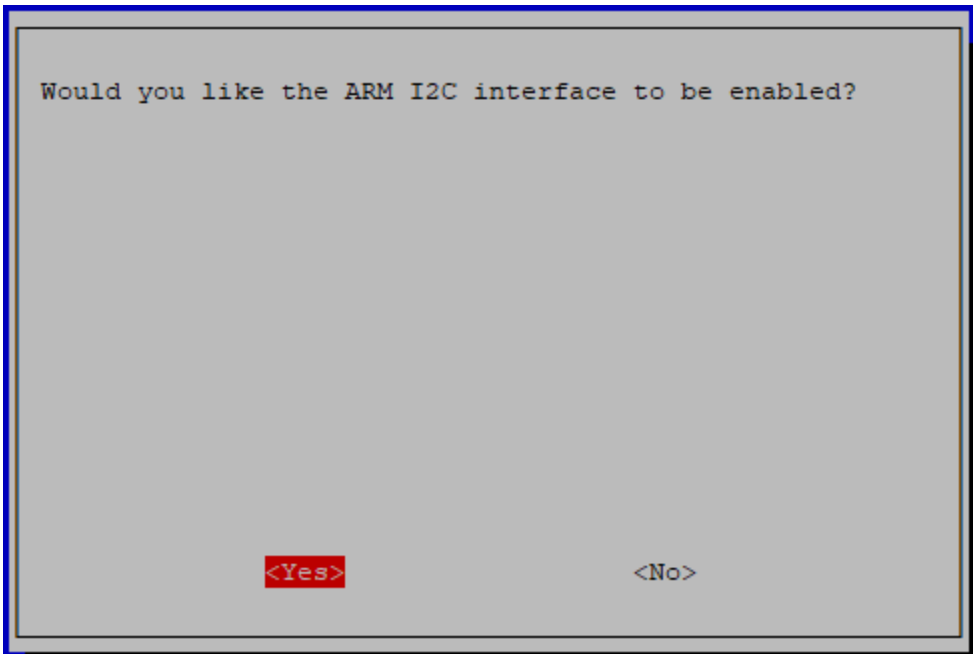
3 Interfacing options



P5 I2C



<Yes>, then <Ok> -> <Finish>



Step 2: Check whether the i2c modules are loaded and active.

```
lsmod | grep i2c
```

Then the following codes will appear (the number may be different), if it does not appear, please reboot the Raspberry Pi with `sudo reboot`.

```
i2c_dev                 6276    0
i2c_bcm2708             4121    0
```

Step 3: Install i2c-tools.

```
sudo apt-get install i2c-tools
```

Step 4: Check the address of the I2C device.

```
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: -- -- -- -- -- -- -- --
```

If there is an I2C device connected, the address of the device will be displayed.

Step 5:

For C language users: Install libi2c-dev.

```
sudo apt-get install libi2c-dev
```

For Python users: Install smbus for I2C.

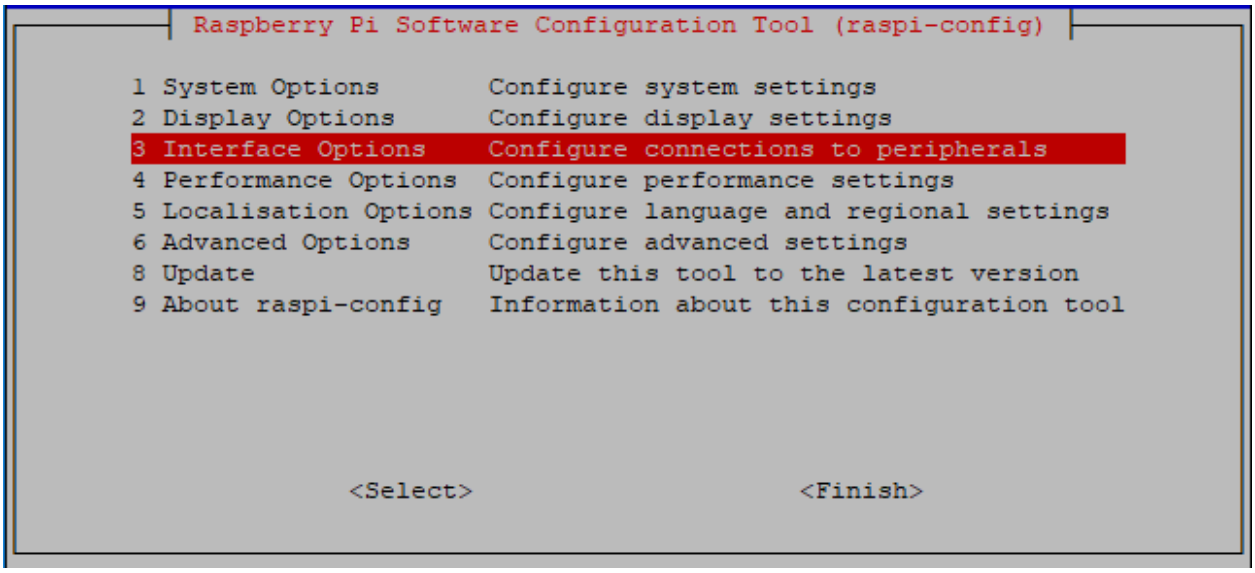
```
sudo pip3 install smbus2
```

11.3 SPI Configuration

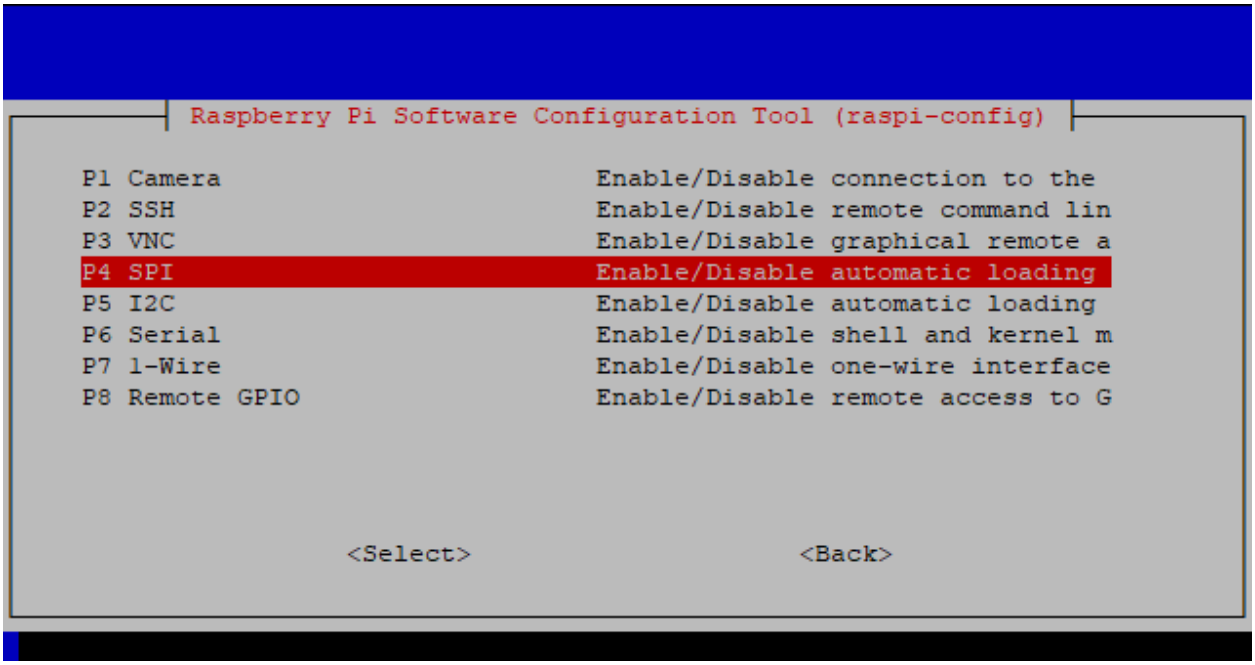
Step 1: Enable the SPI port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

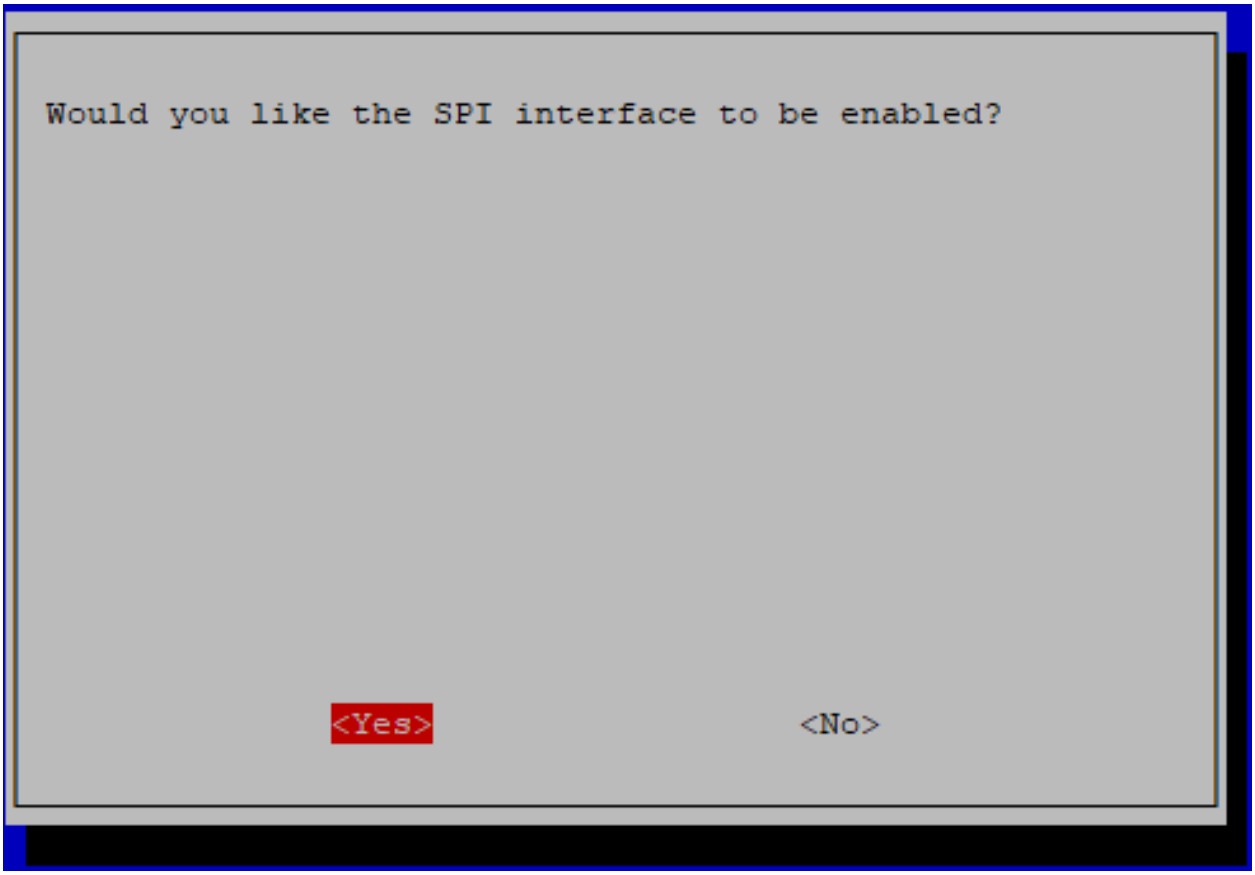
3 Interfacing options



P4 SPI



<YES>, then click <OK> and <Finish>.



Step 2: Check that the spi modules are loaded and active.

```
ls /dev/sp*
```

Then the following codes will appear (the number may be different).

```
/dev/spidev0.0  /dev/spidev0.1
```

Step 3: Install Python module SPI-Py.

```
git clone https://github.com/lthiery/SPI-Py.git
cd SPI-Py
sudo python3 setup.py install
```

Note: This step is for python users, if you use C language, please skip.

11.4 Audio Configuration

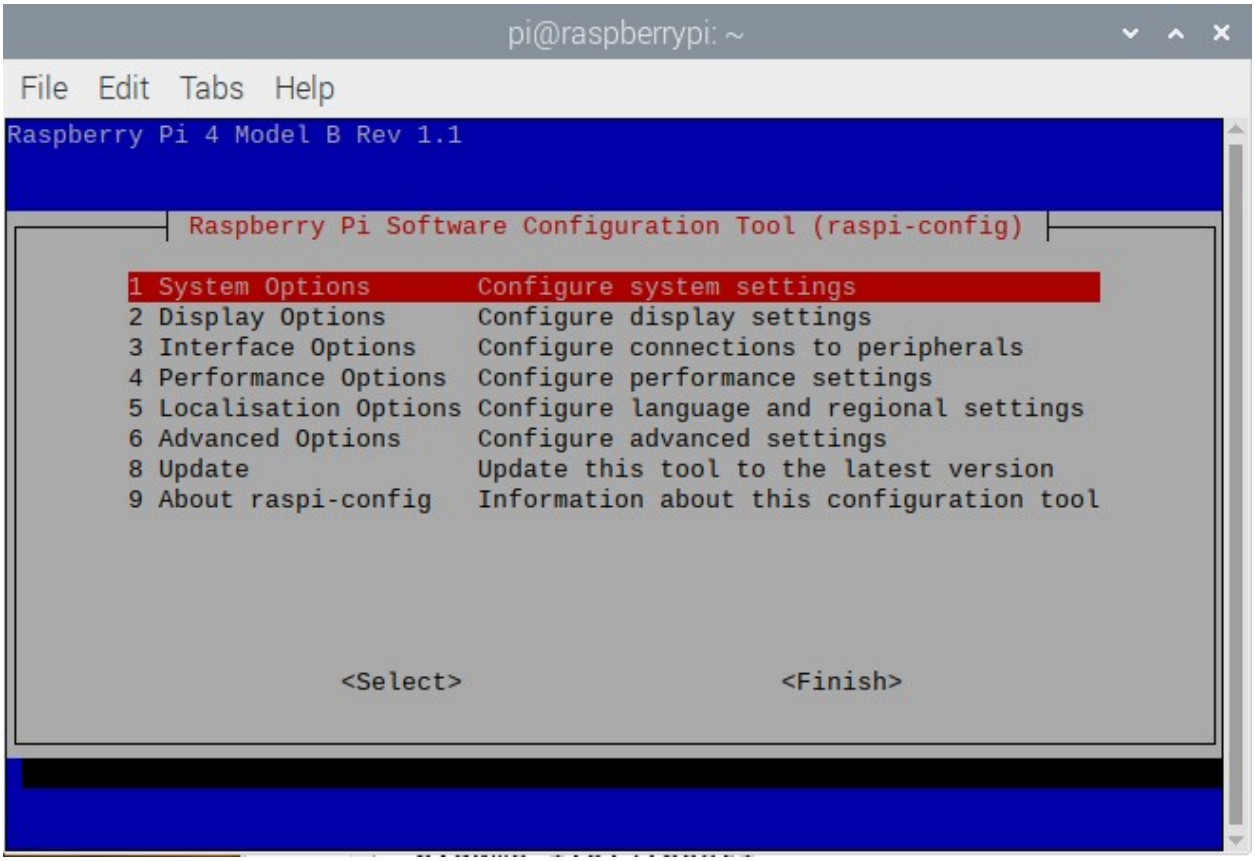
11.4.1 Change Audio Output

If your speaker have no sound, it may be because the Raspberry Pi has selected the wrong audio output, the correct one should be **Headphones**. You can change the audio output by following these steps.

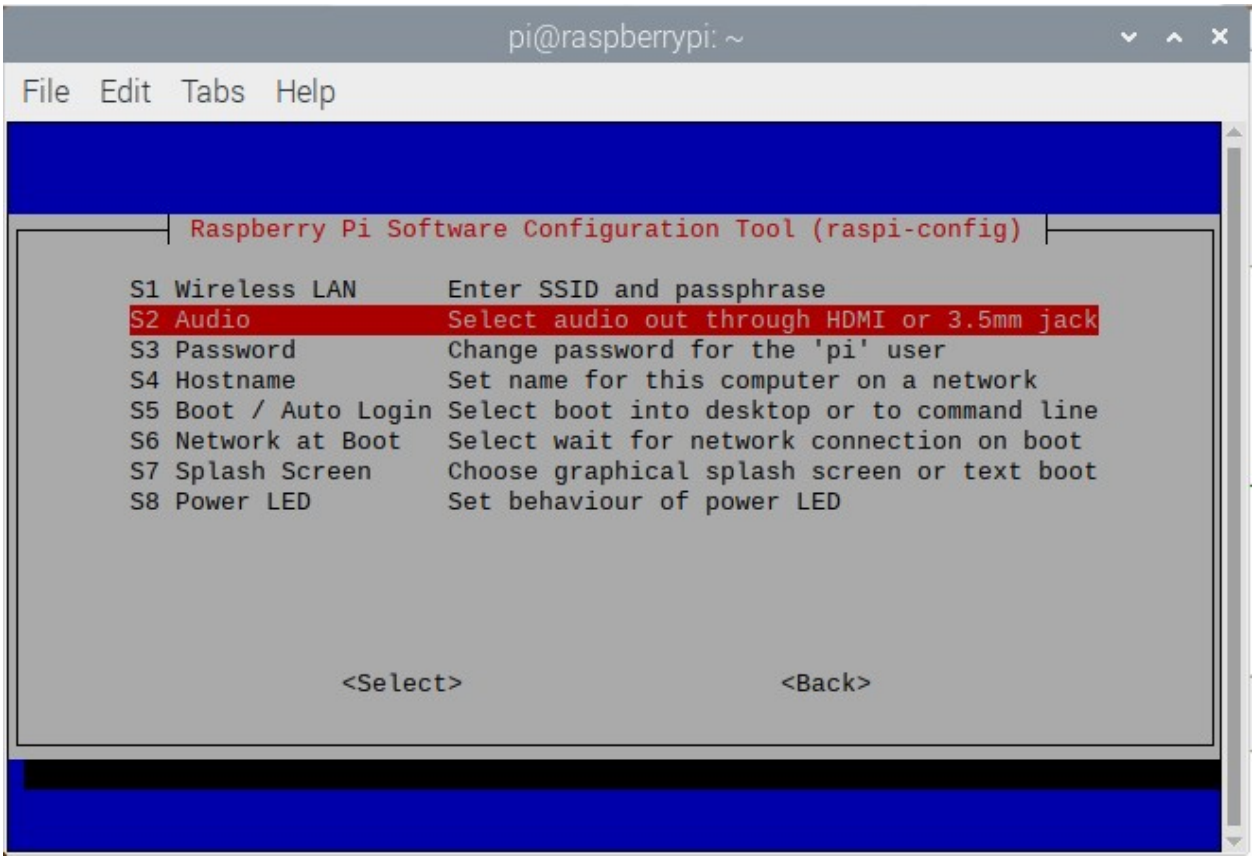
Enter the following command.

```
sudo raspi-config
```

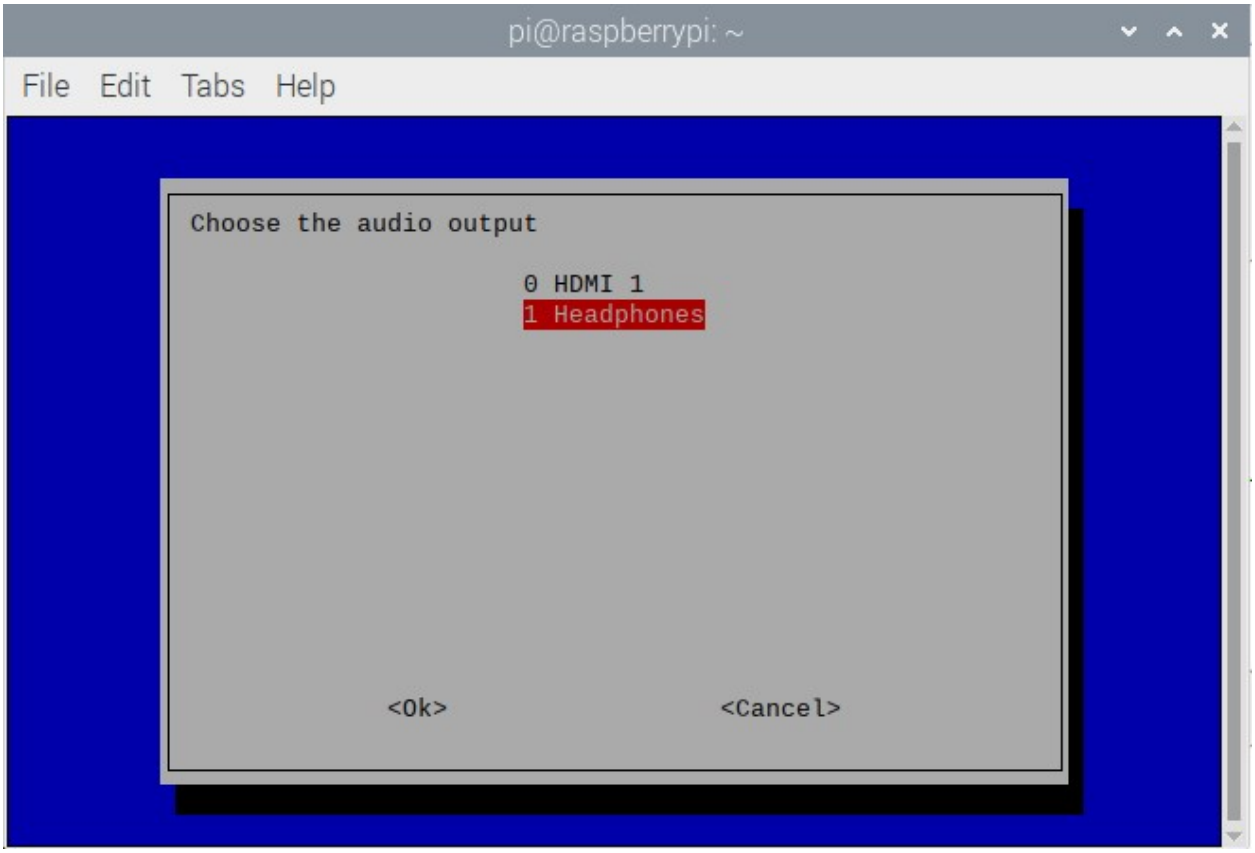
Select **1 System Options**.



Then **S2 Audio**.



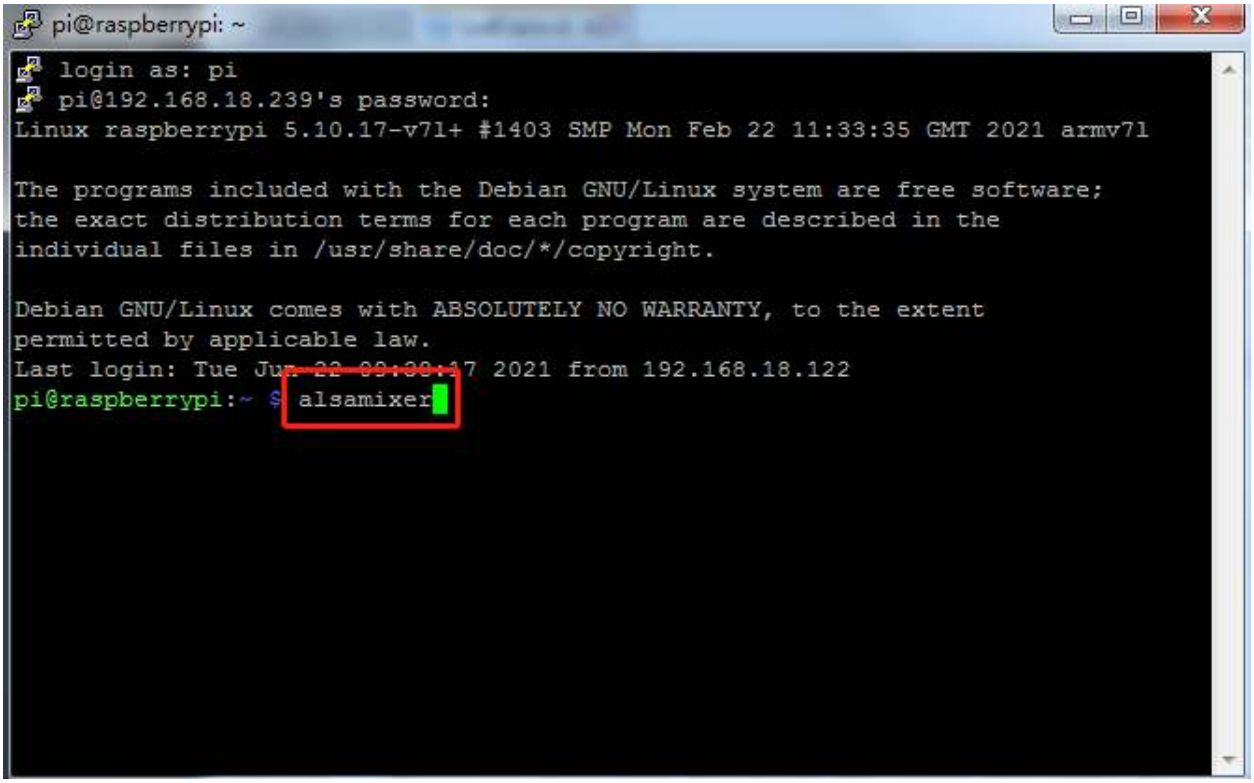
After selecting **1 Headphones**, press `Enter` to confirm and select `Finish` to exit.



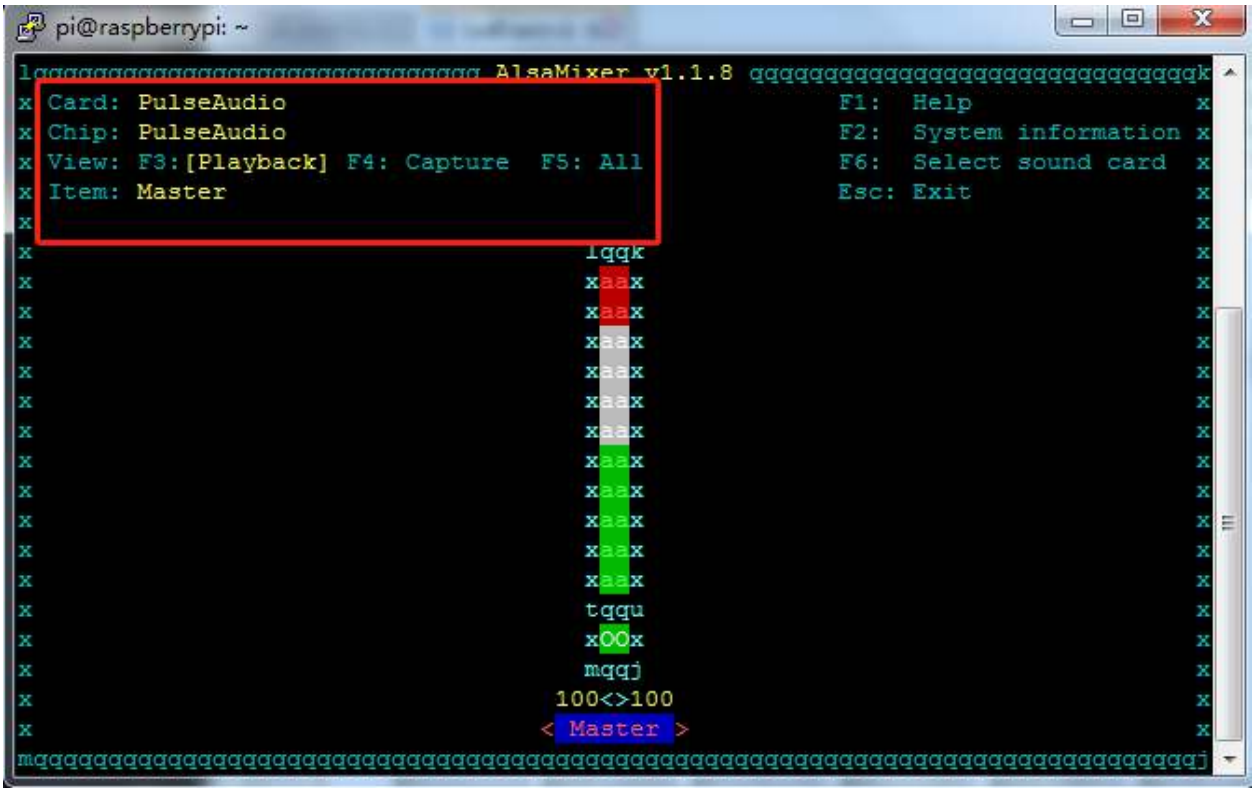
11.4.2 Adjust Volume

If you feel that the volume of the speakers is too low, you can adjust it by entering the following command.

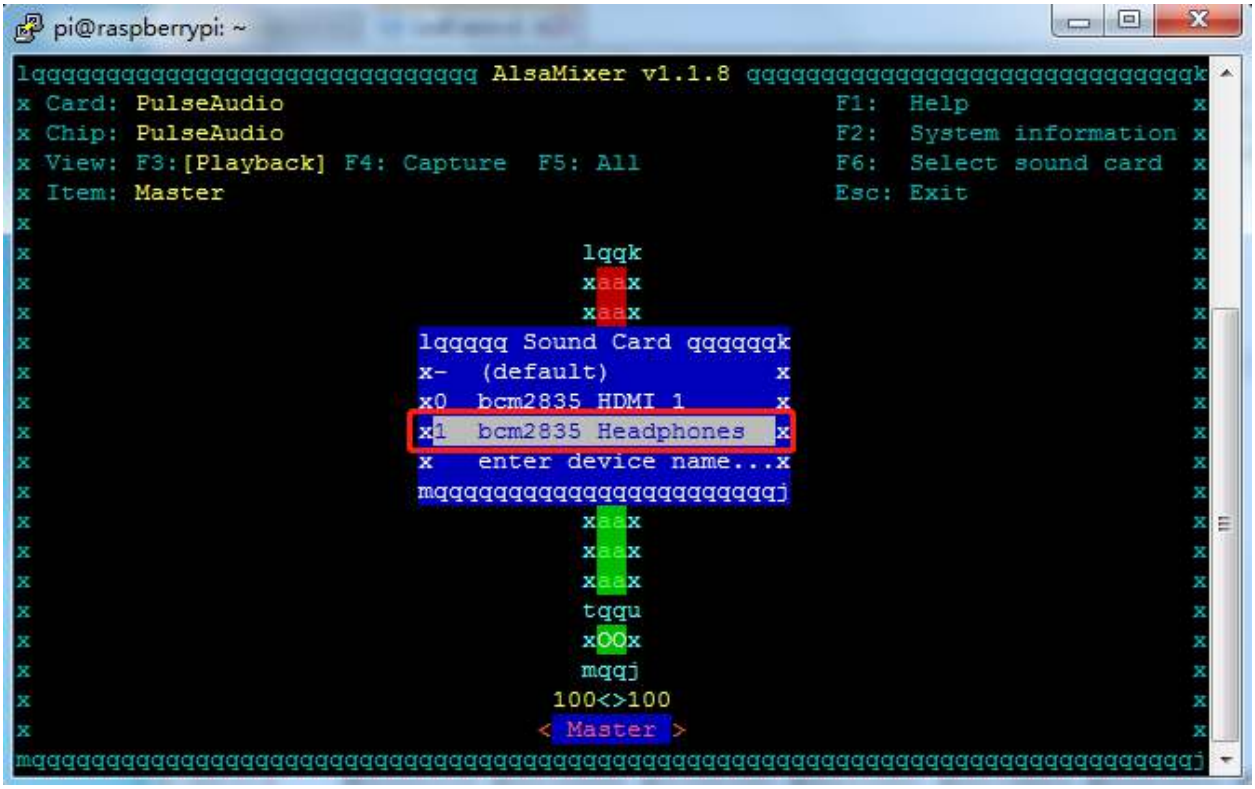
```
alsamixer
```



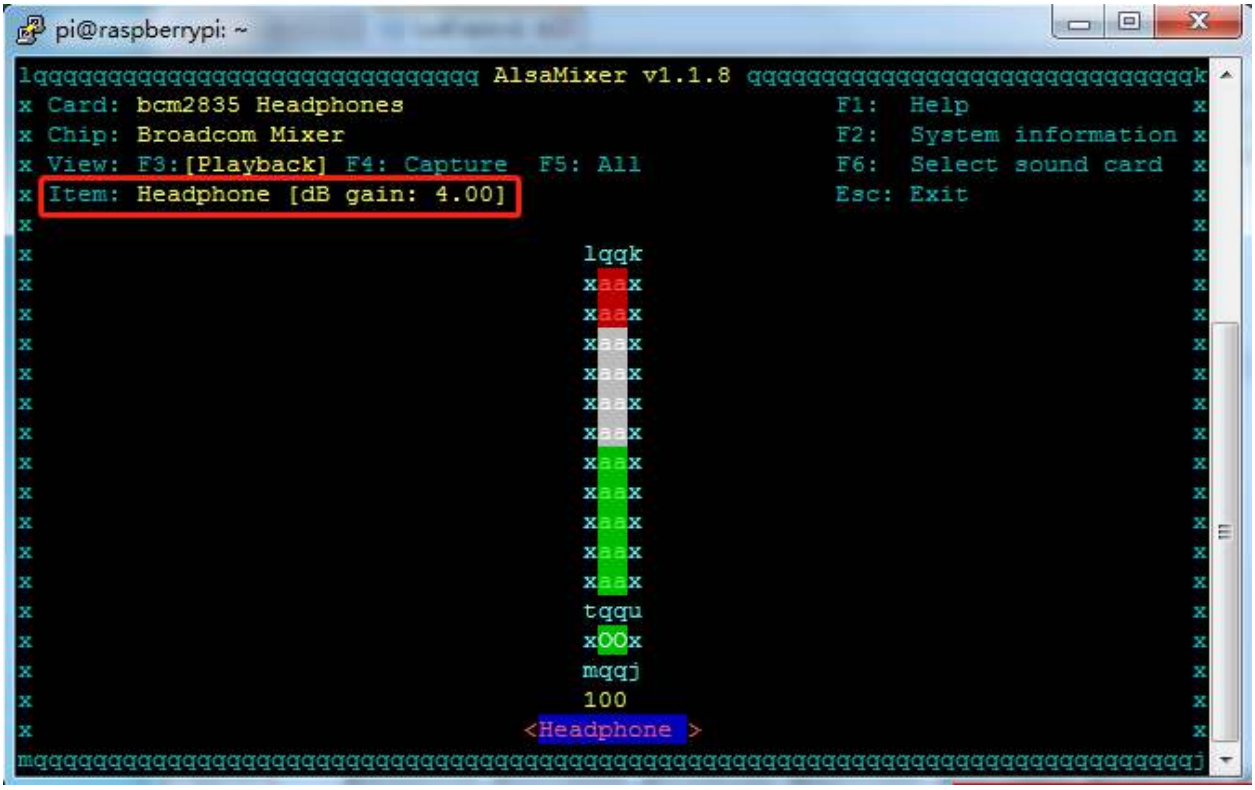
The default page is shown below.



Press F6 to select **Headphones** mode.



Then press the arrow keys up and down to adjust the volume level, and press `ESC` to exit.



11.5 Remote Desktop

There are two ways to control the desktop of the Raspberry Pi remotely:

VNC and **XRDP**, you can use any of them.

11.5.1 VNC

You can use the function of remote desktop through VNC.

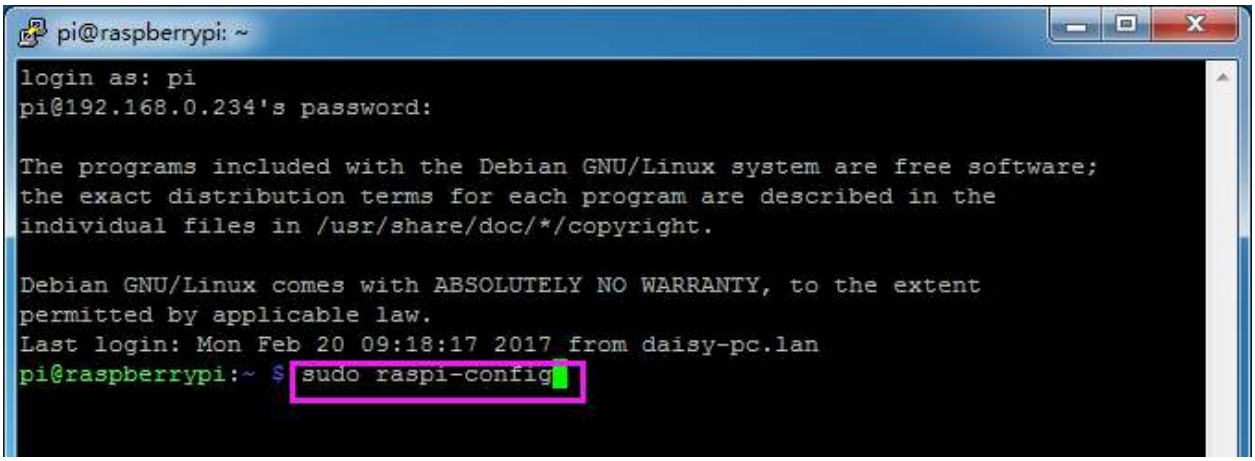
Enable VNC service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

Step 1

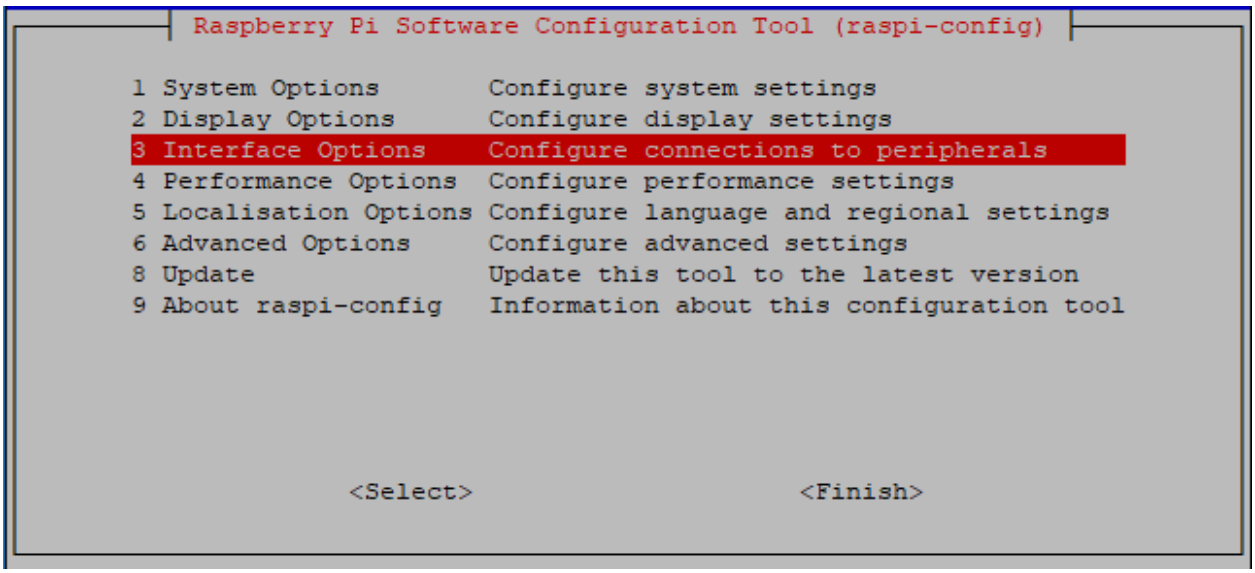
Input the following command:

```
sudo raspi-config
```



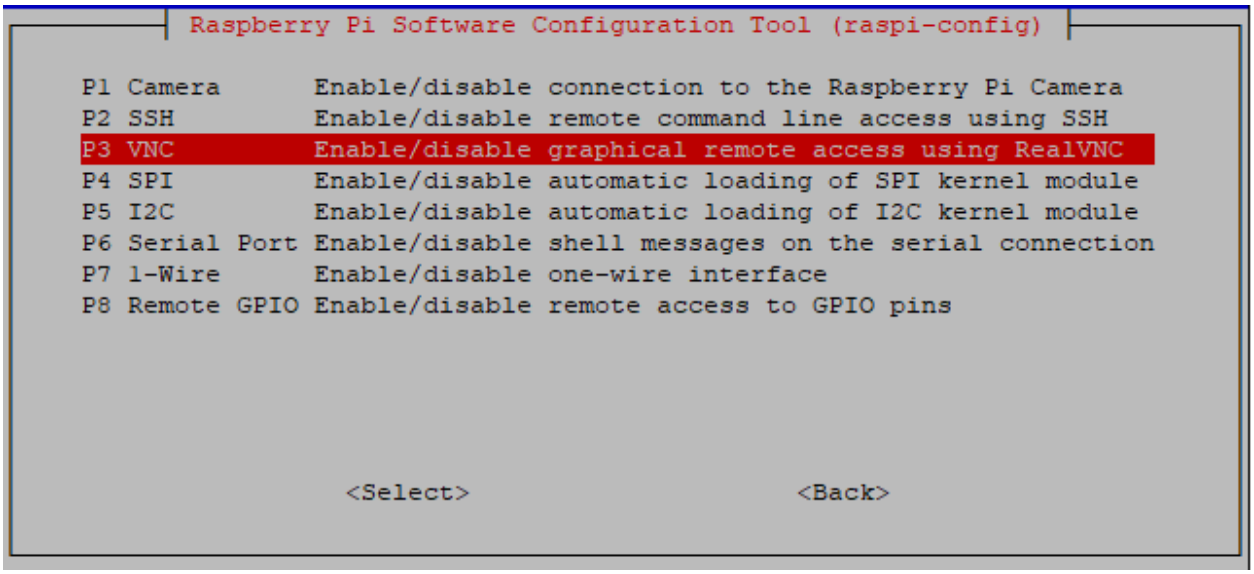
Step 2

Choose **3 Interfacing Options** by press the down arrow key on your keyboard, then press the **Enter** key.



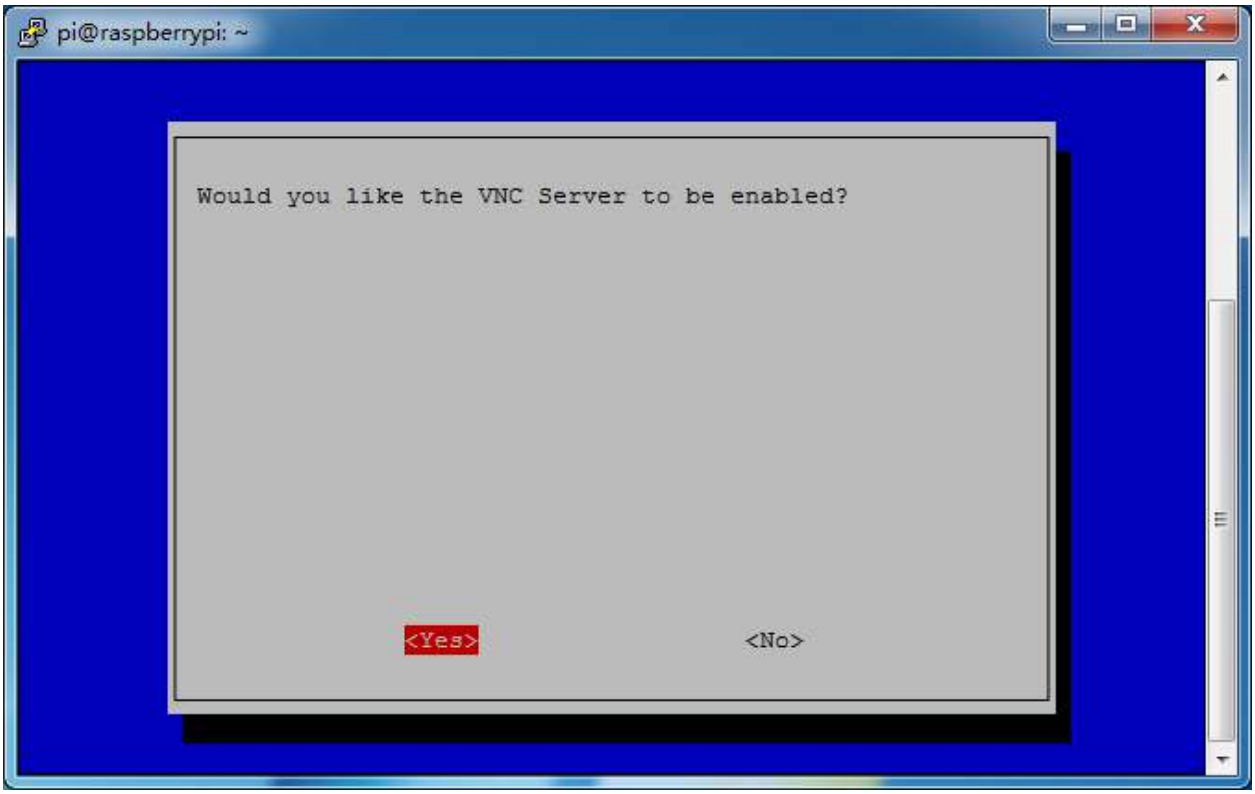
Step 3

P3 VNC



Step 4

Select **Yes -> OK -> Finish** to exit the configuration.



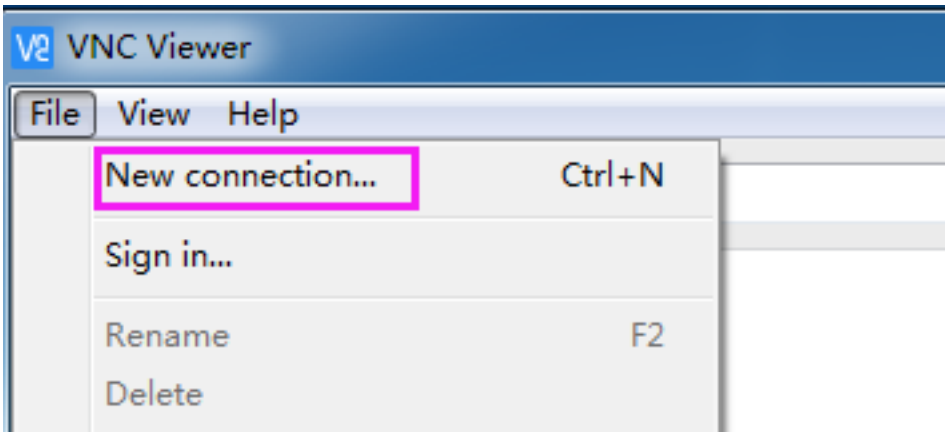
Login to VNC

Step 1

You need to download and install the VNC Viewer on personal computer. After the installation is done, open it.

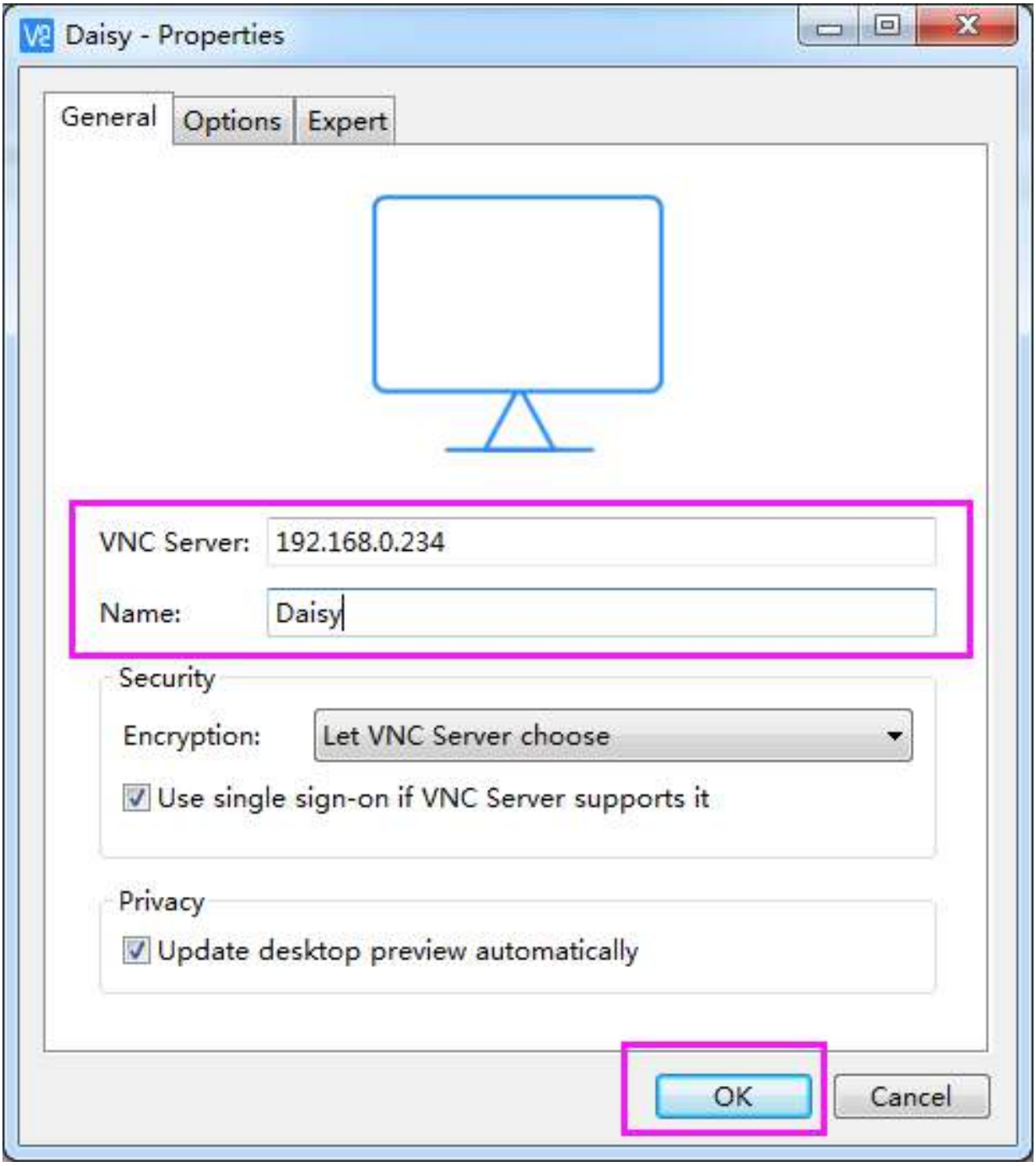
Step 2

Then select "**New connection**".



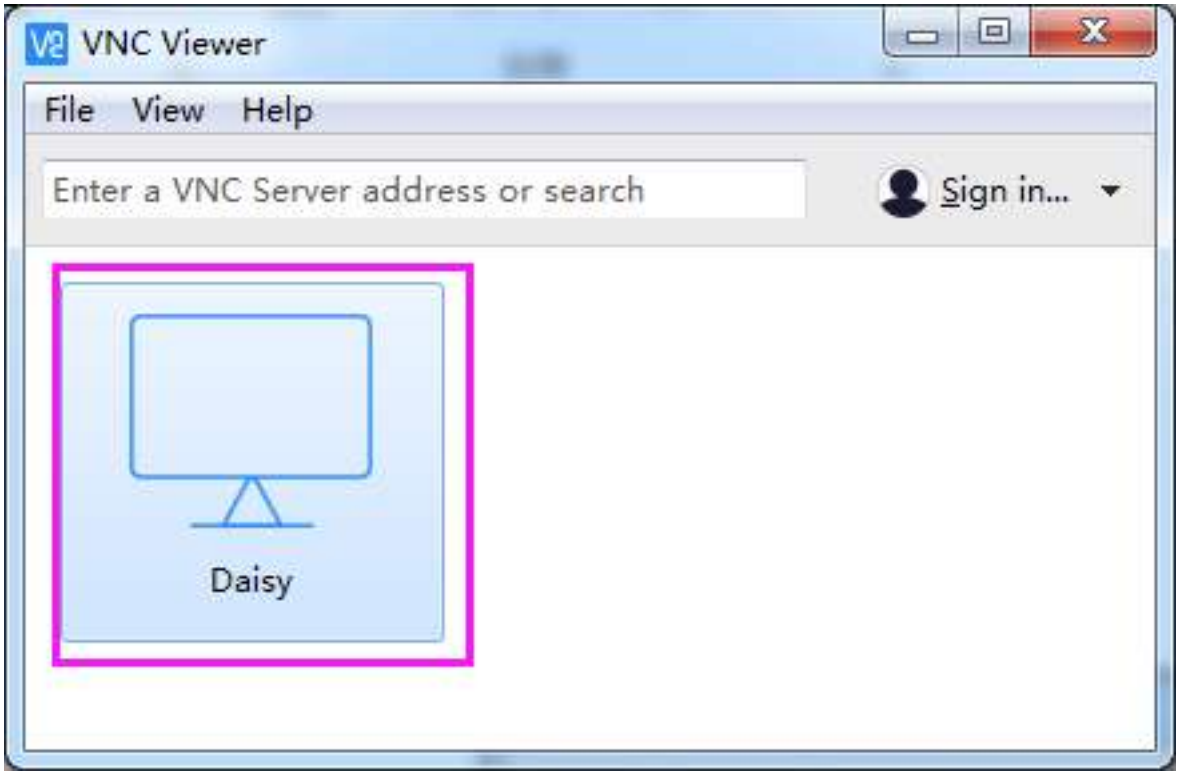
Step 3

Input IP address of Raspberry Pi and any **Name**.



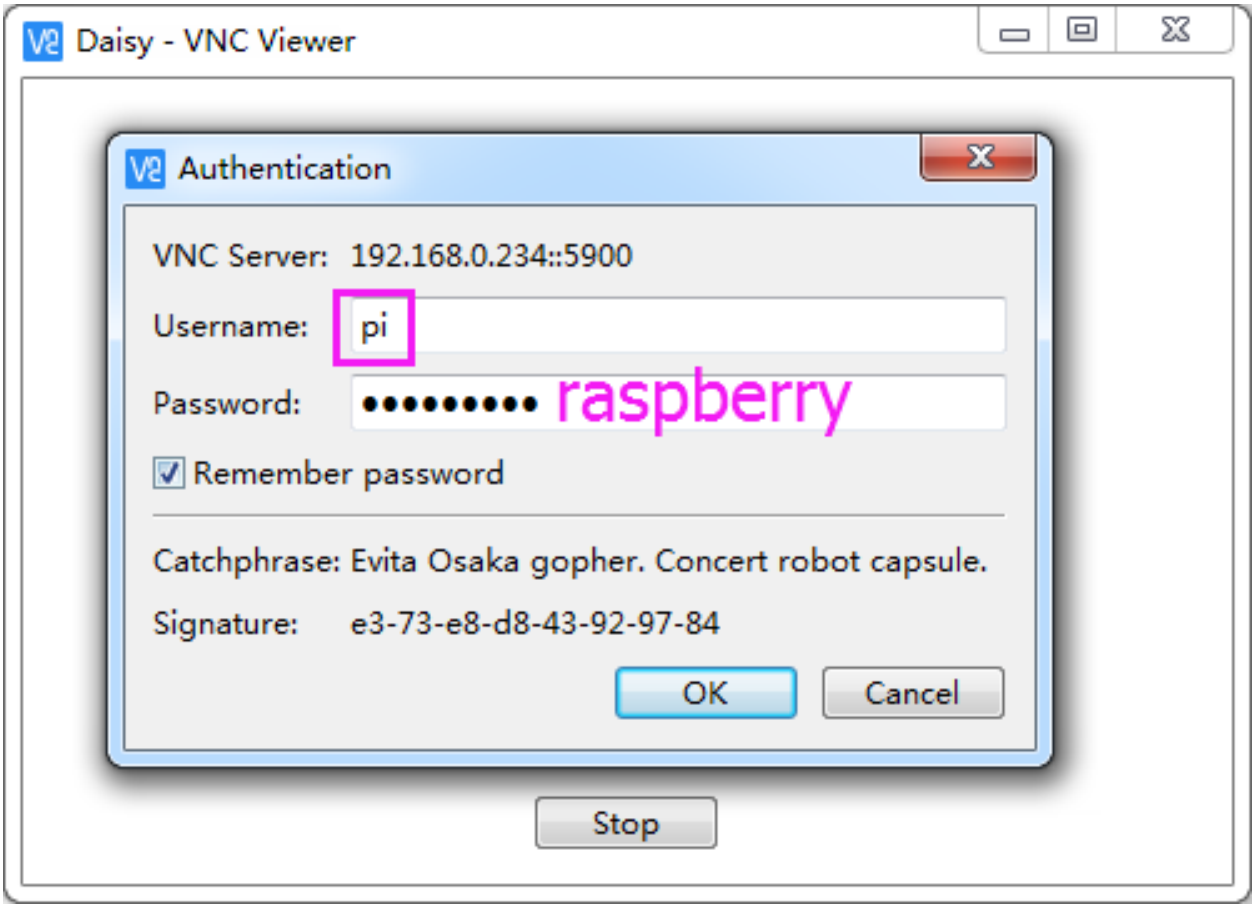
Step 4

Double click the **connection** just created:



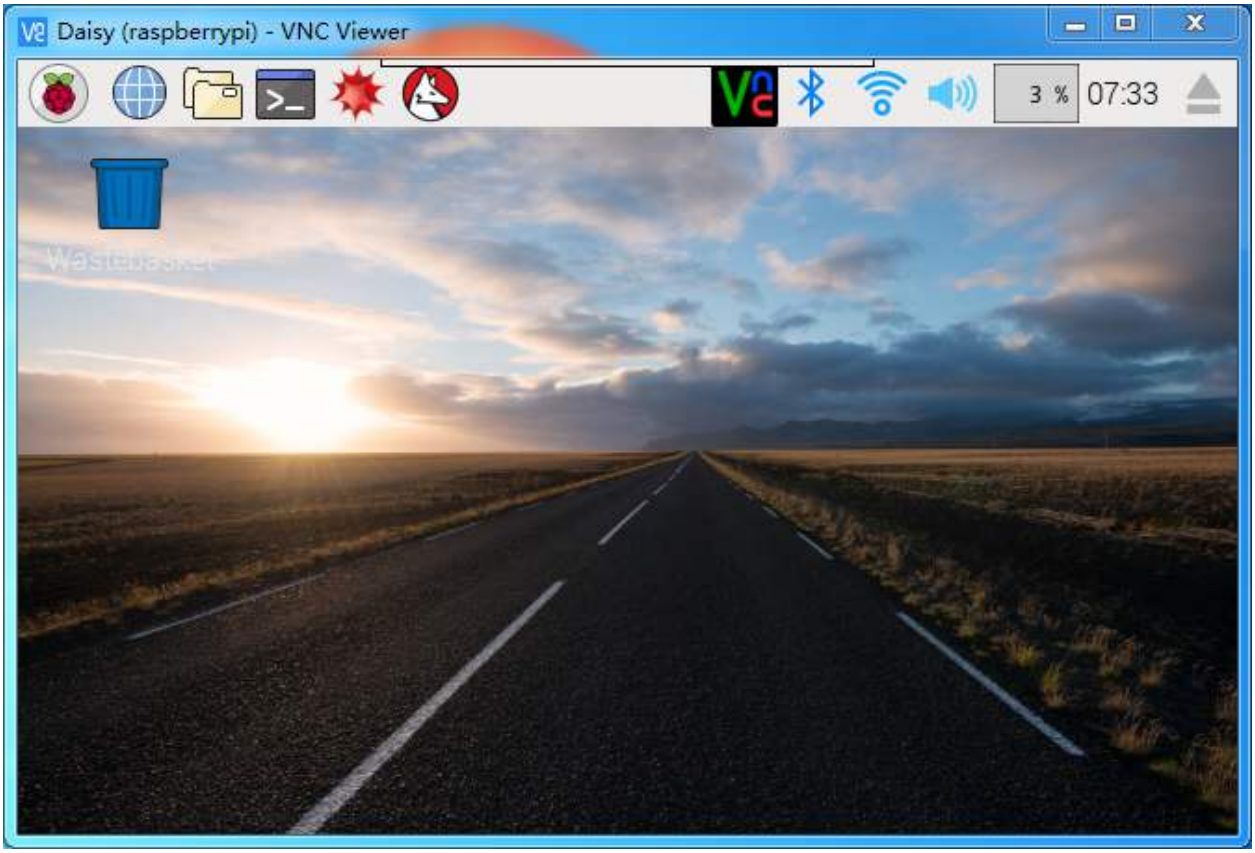
Step 5

Enter Username (**pi**) and Password (**raspberry** by default).



Step 6

Now you can see the desktop of the Raspberry Pi:



That's the end of the VNC part.

11.5.2 XRDP

Another method of remote desktop is XRDP, it provides a graphical login to remote machines using RDP (Microsoft Remote Desktop Protocol).

Install XRDP

Step 1

Login to Raspberry Pi by using SSH.

Step 2

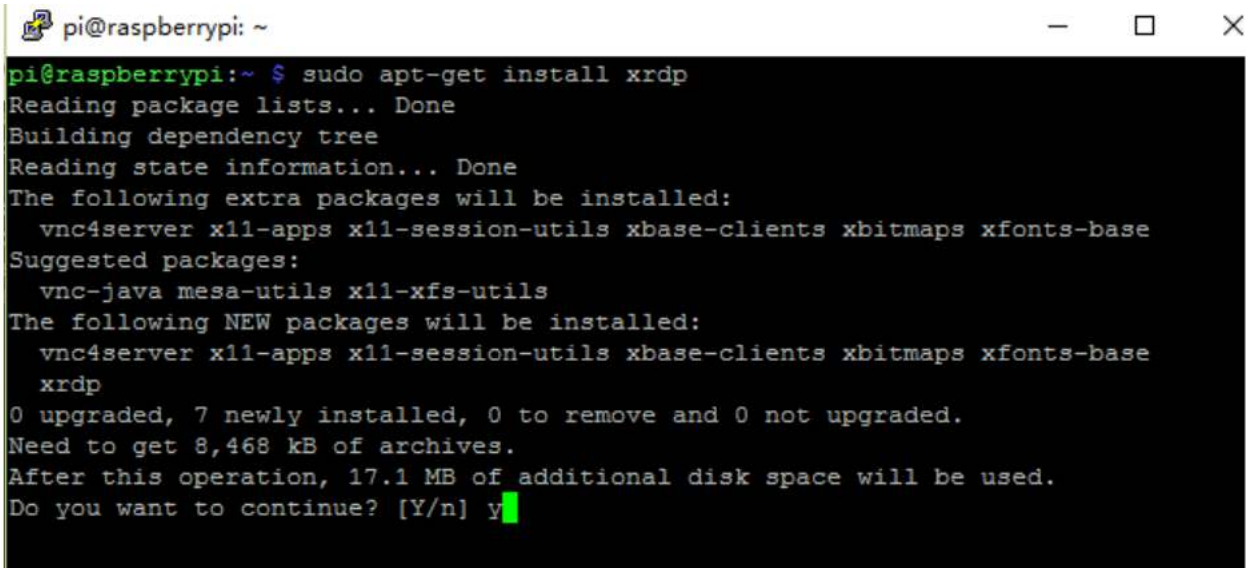
Input the following instructions to install XRDP.

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

Step 3

Later, the installation starts.

Enter "Y", press key "Enter" to confirm.



Step 4

Finished the installation, you should login to your Raspberry Pi by using Windows remote desktop applications.

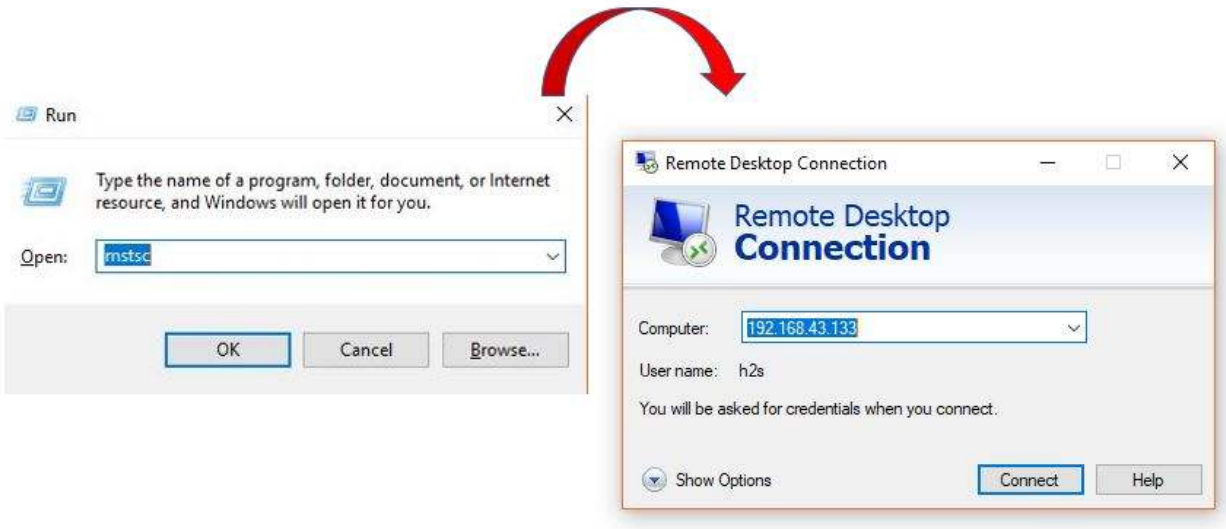
Login to XRDP

Step 1

If you are a Windows user, you can use the Remote Desktop feature that comes with Windows. If you are a Mac user, you can download and use Microsoft Remote Desktop from the APP Store, and there is not much difference between the two. The next example is Windows remote desktop.

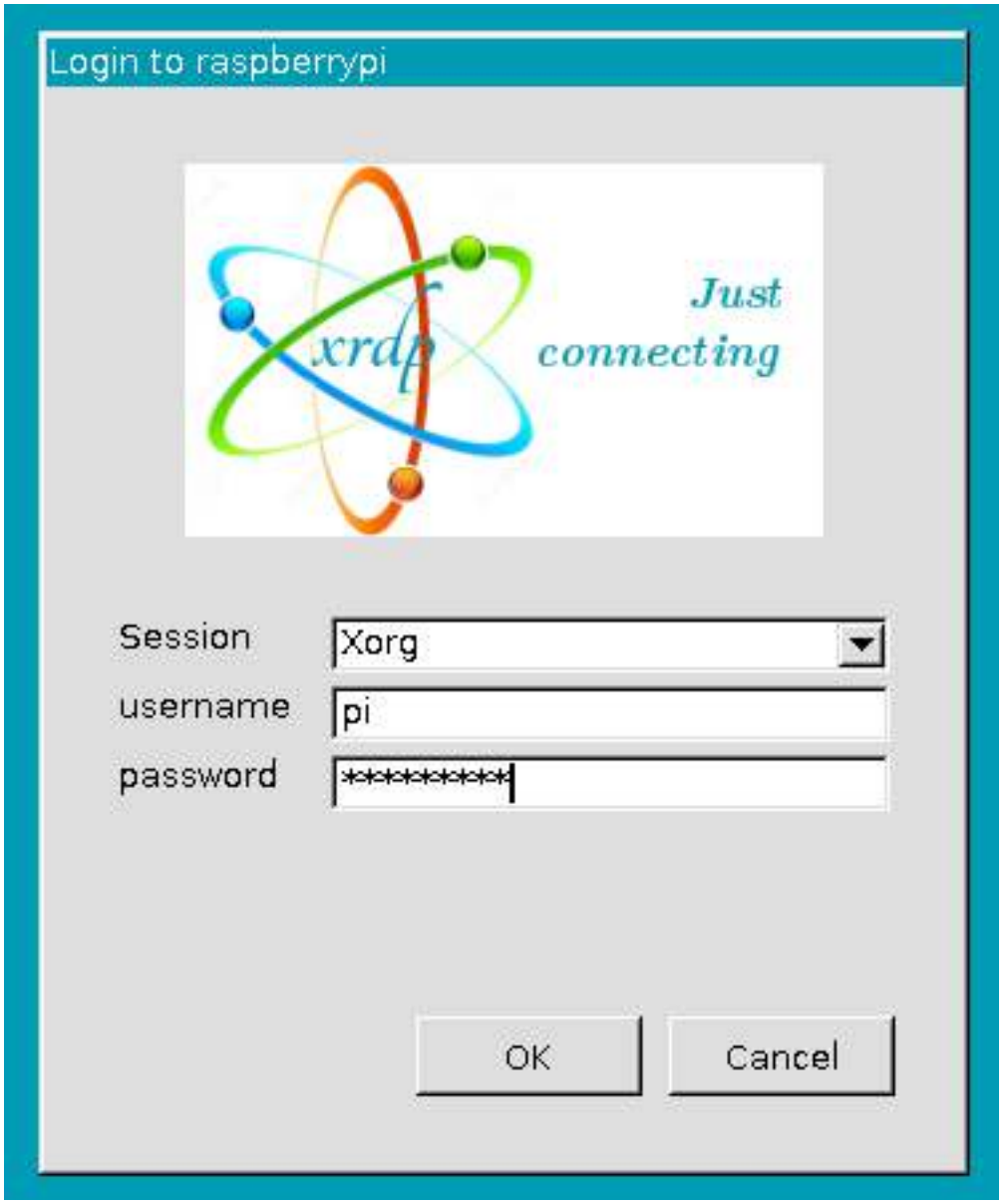
Step 2

Type in "**mstsc**" in Run (WIN+R) to open the Remote Desktop Connection, and input the IP address of Raspberry Pi, then click on "Connect".



Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click “OK”. At the first time you log in, your username is “pi” and the password is “raspberry”.



Step 4

Here, you successfully login to RPi by using the remote desktop.

Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.

11.6 Filezilla Software

The File Transfer Protocol (FTP) is a standard communication protocol used for the transfer of computer files from a server to a client on a computer network.

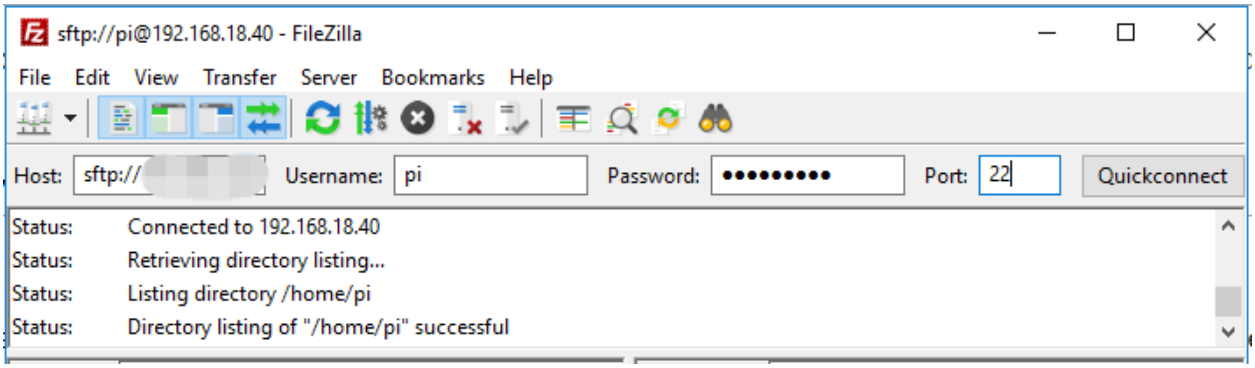
Filezilla is an open source software that not only supports FTP, but also FTP over TLS (FTPS) and SFTP. We can use Filezilla to upload local files (such as pictures and audio, etc.) to the Raspberry Pi, or download files from the Raspberry Pi to the local.

Step 1: Download Filezilla.

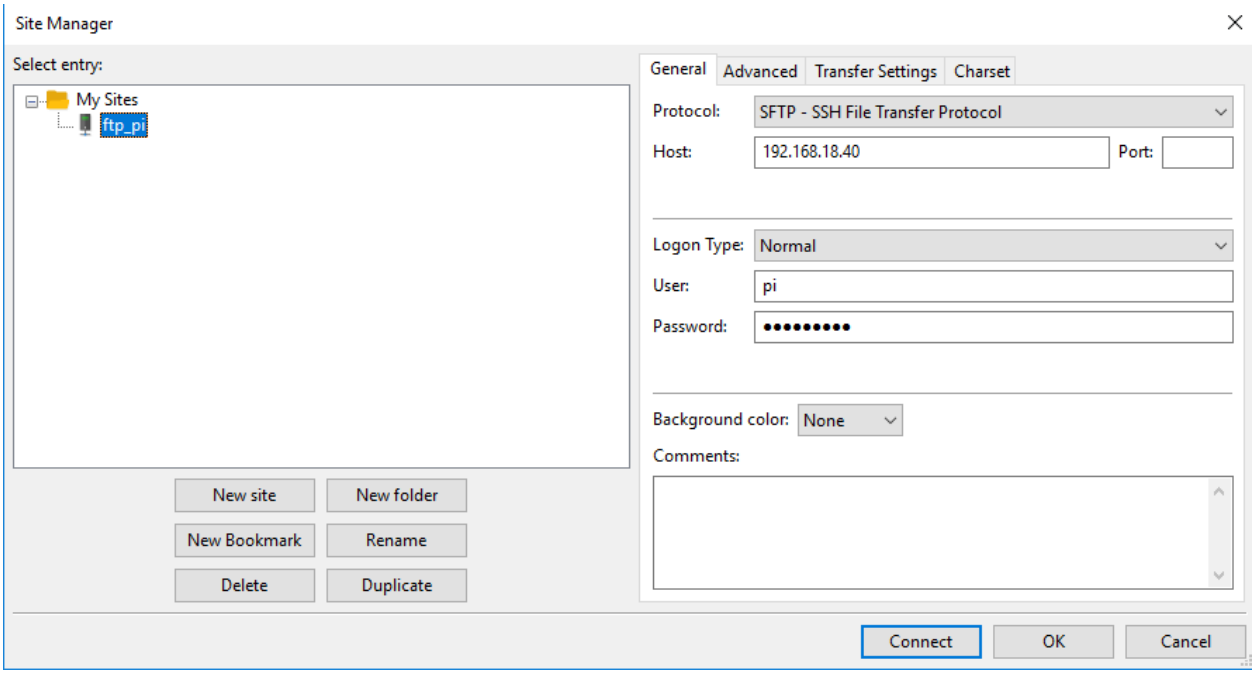
Download the client from Filezilla's official website, Filezilla has a very good tutorial, please refer to: Documentation - Filezilla.

Step 2: Connect to Raspberry Pi

After a quick install open it up and now connect it to an FTP server. It has 3 ways to connect, here we use the **Quick Connect** bar. Enter the **hostname/IP**, **username**, **password** and **port (22)**, then click **Quick Connect** or press **Enter** to connect to the server.

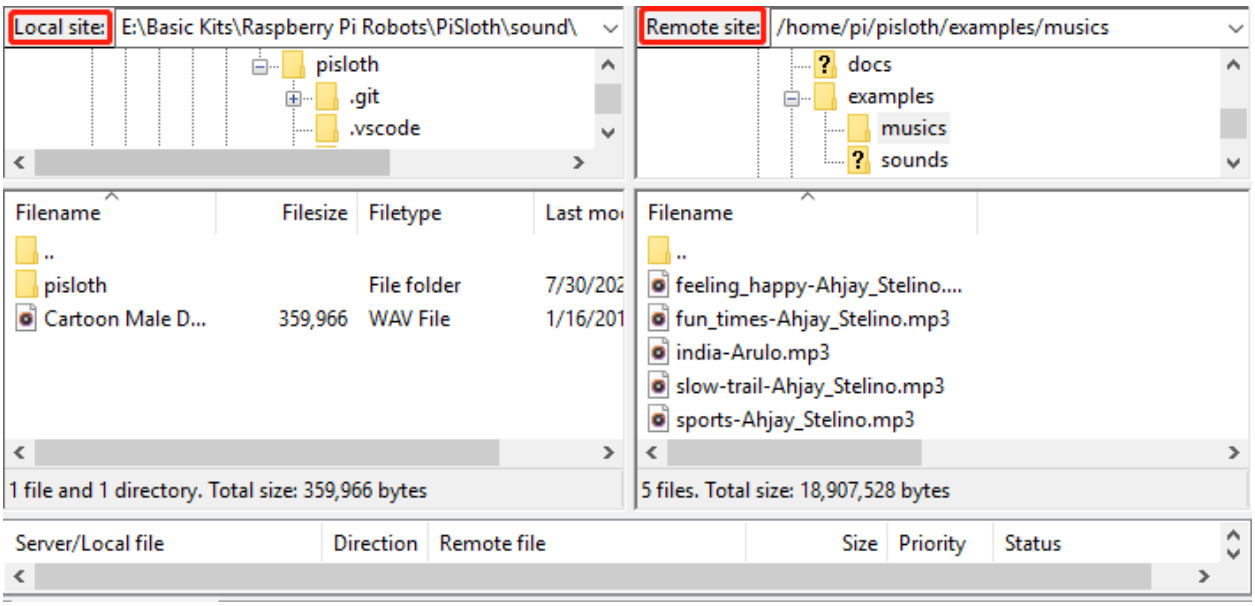


Note: Quick Connect is a good way to test your login information. If you want to create a permanent entry, you can select **File-> Copy Current Connection to Site Manager** after a successful Quick Connect, enter the name and click **OK**. Next time you will be able to connect by selecting the previously saved site inside **File -> Site Manager**.



Step 3: Upload/download files.

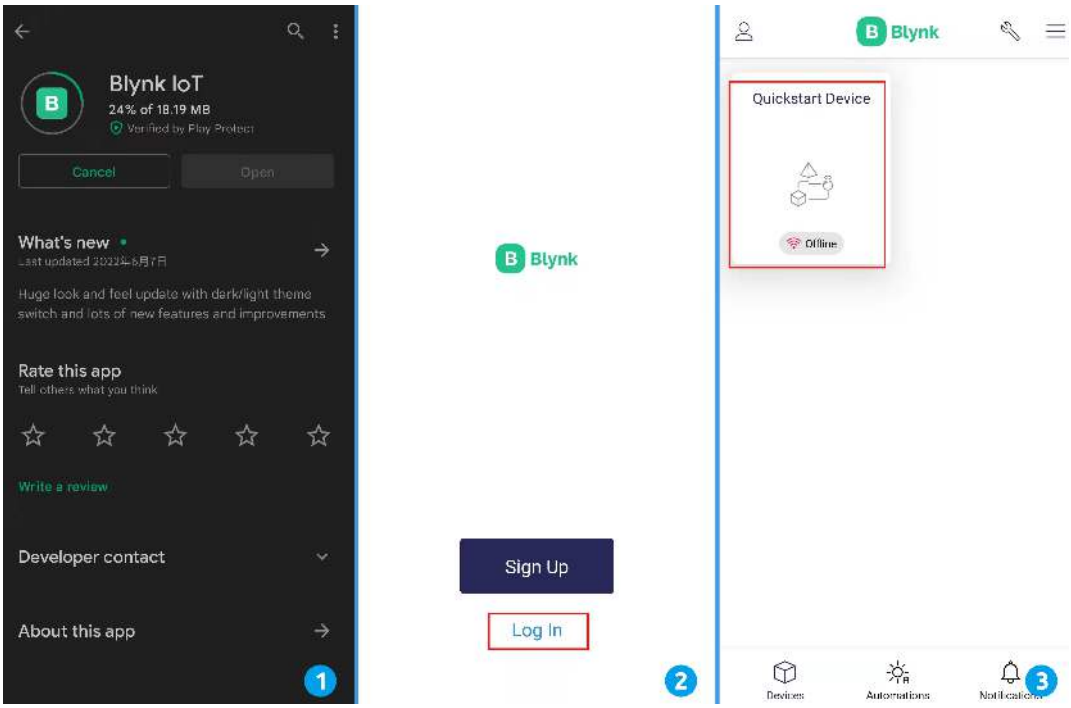
You can upload local files to Raspberry Pi by dragging and dropping them, or download the files inside Raspberry Pi files locally.



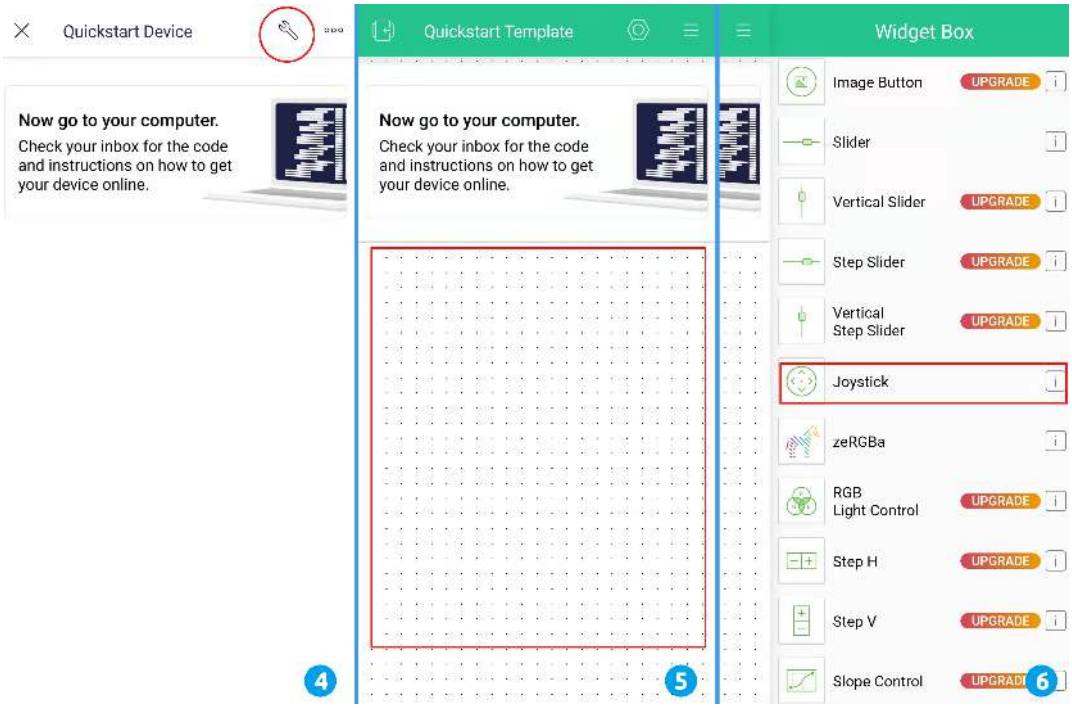
11.7 How to use Blynk on mobile device?

Note: As datastreams can only be created in Blynk on the web, you will need to reference different projects to create datastreams on the web, then follow the tutorial below to create widgets in Blynk on your mobile device.

1. Open Google Play or APP Store on your mobile device and search for "Blynk IoT" (not Blynk(legacy)) to download.
2. After opening the APP, login in, this account should be the same as the account used on the web client.
3. Then go to **Dashboard** (if you don't have one, create one) and you will see that the **Dashboard** for mobile and web are independent of each other.

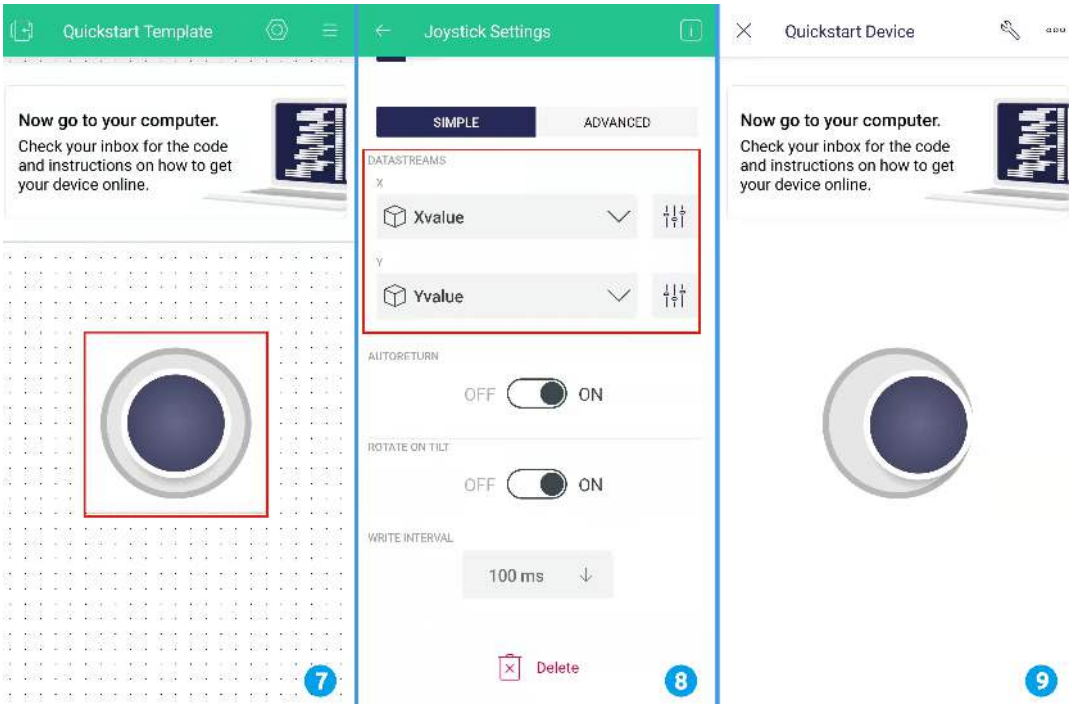


4. Click **Edit** Icon.
5. Click on the blank area.
6. Choose the same widget as on the web page, such as select a **Joystick** widget.



7. Now you will see a **Joystick** widget appear in the blank area, click on it.
8. **Joystick** Settings will appear, select the **Xvalue** and **Yvalue** datastreams you just set in the web page. Note that each widget corresponds to a different datastream in each project.

9. Go back to the **Dashboard** page and you can operate the **Joystick** when you want.



CHAPTER

TWELVE

FAQ

12.1 C code is not working?

- Check your wiring for problems.
- Check if the code is reporting errors, if so, refer to: *Install and Check the WiringPi*.
- Has the code been compiled before running.
- 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 *Install and Check the WiringPi* to manually upgrade it to version 2.52.

CHAPTER

THIRTEEN

THANK YOU

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study,investigation, enjoyment, or other non-commercial or nonprofit purposes, under therelated regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.