SunFounder super-kit-v2-for-pi

www.sunfounder.com

Sep 21, 2022

CONTENTS

About the Super Kit 2.0

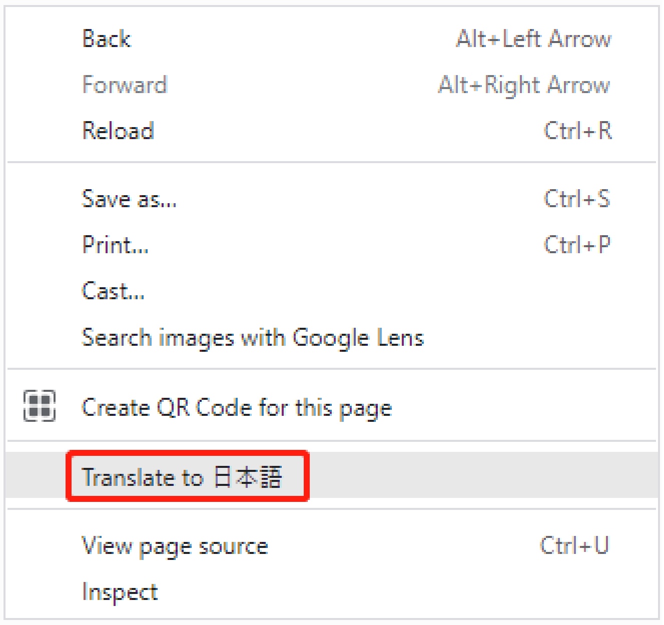
This super kit is suitable for the Raspberry Pi B, model B+, Pi 2 model B, Pi 3 model B , Pi 3 model B+ and 4 Model B. It includes various components and chips that can show different interesting phenomena. You can make it happen by following the experiment instructions, and learn basic knowledge about them. Also you can explore more application after mastering the principle and code. Now get on the road!

About the display language

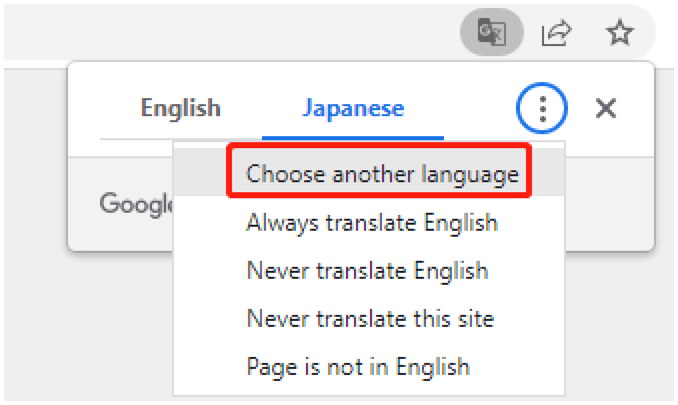
In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

The steps are as follows.

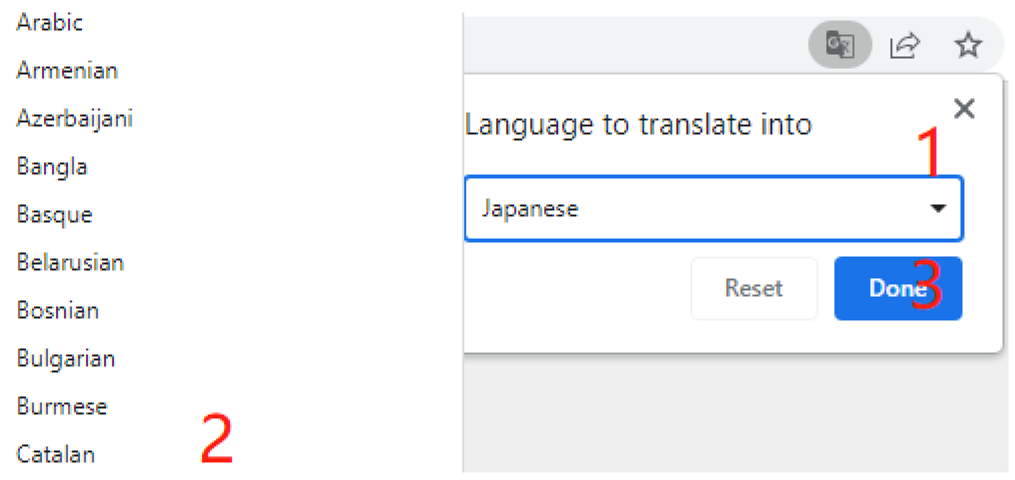
- In this course page, right-click and select **Translate to xx**. If the current language is not what you want, you can change it later.



- 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**.


Arabic
Armenian
Azerbaijani
Bangla
Basque
Belarusian
Bosnian
Bulgarian
Burmese
Catalan
2
Language to translate into
1
Japanese
Reset
Done
3

CHAPTER

ONE

COMPONENTS LIST

Note: After unpacking, please check that the number of components is correct and that all components are in good condition.

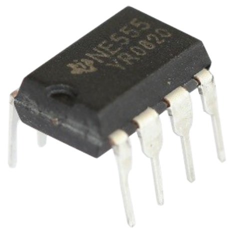
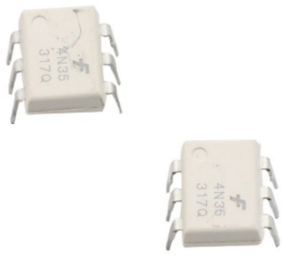
No.	Name	Quantity	Component
1	RGB LED	1	
2	555 Timer IC	1	
3	Optocoupler (4N35)	2	

Table 1 – continued from previous page

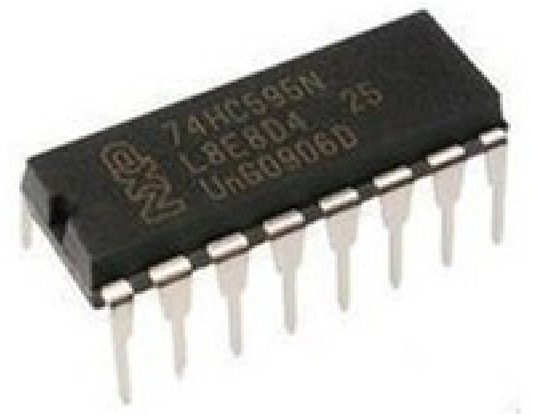
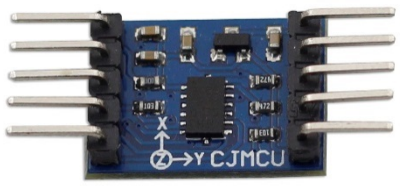
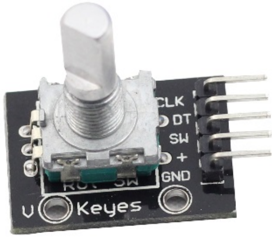
4	Shift Register (74HC595)	2	
5	L293D	1	
6	Accelerometer ADXL345	1	
7	Rotary Encoder	1	

Table 1 – continued from previous page

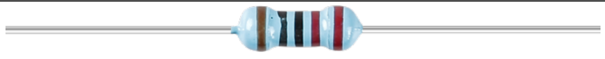
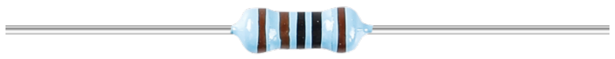
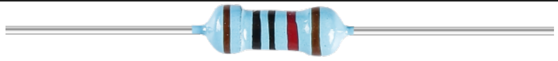
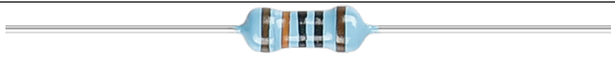
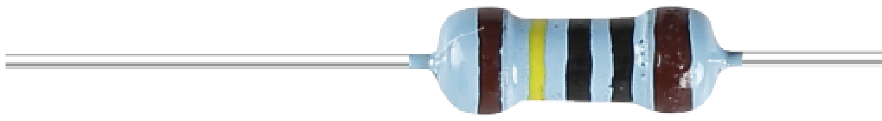
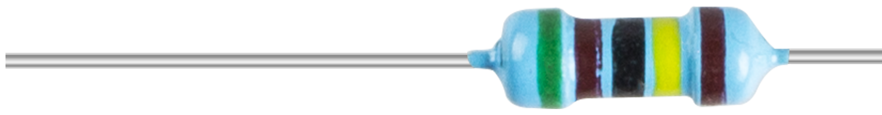
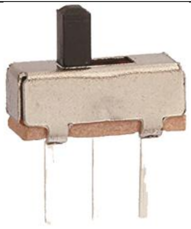
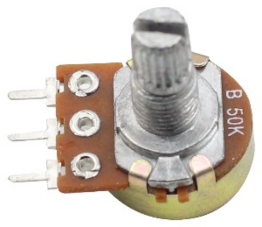
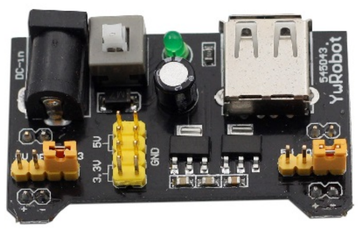
8	Button	5	
9	Resistor (220)	8	
10	Resistor (1k)	8	
11	Resistor (10k)	4	
12	Resistor (100k)	4	
13	Resistor (1M)	1	
14	Resistor (5.1M)	1	
15	Switch	1	
16	Potentiometer (50k)	1	
17	Power Supply Module	1	

Table 1 – continued from previous page

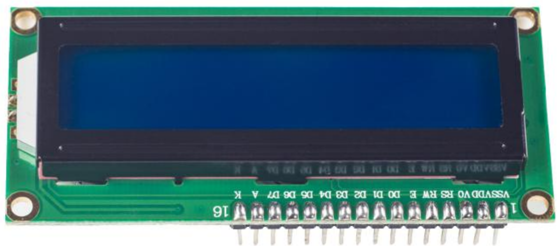
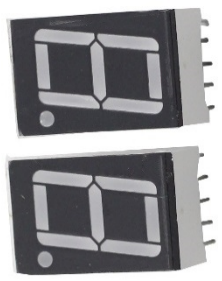
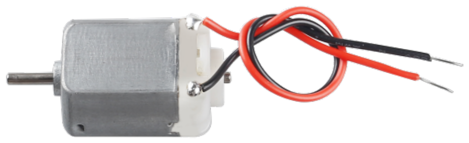
18	LCD1602	1	
19	Dot Matrix Display (8*8)	1	
20	7-Segment Display	2	
21	DC Motor	1	
22	LED (red)	16	
23	LED (white)	2	
24	LED (green)	2	
25	LED (yellow)	2	

Table 1 – continued from previous page

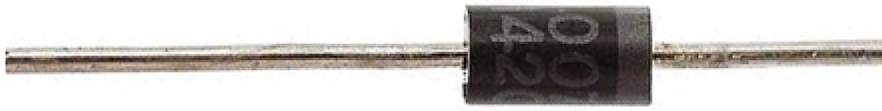
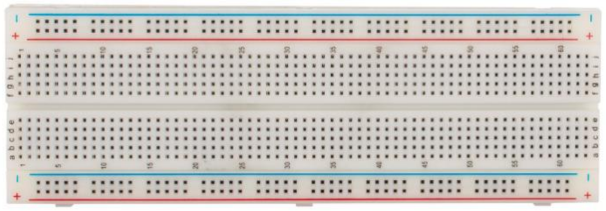
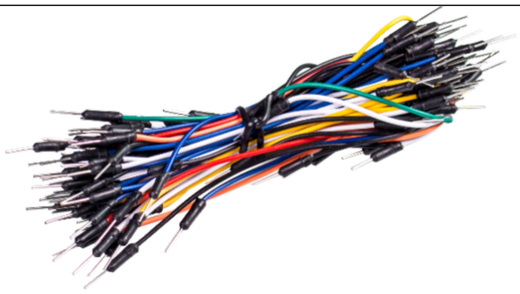
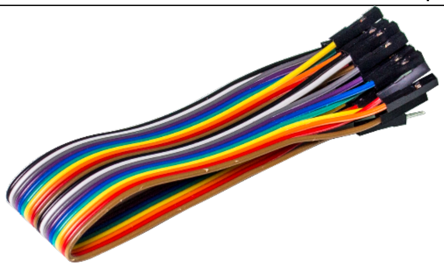
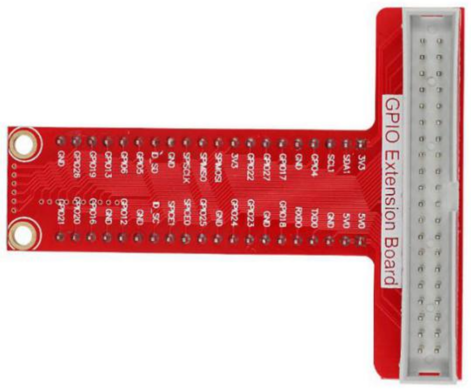
26	NPN Transistor (S8050)	2	
27	PNP Transistor (S8550)	2	
28	Capacitor Ceramic 100nF	4	
29	Capacitor Ceramic 10nF	4	
30	Diode Rectifier	4	
31	Breadboard	1	
32	Male-to-Male Jumper Wire	65	

Table 1 – continued from previous page

33	Female-to-Male Dupont Wire	20	
34	Active Buzzer	1	
35	Fan	1	
36	GPIO Extension Board	1	
37	40-pin Ribbon Cable	1	

CHAPTER

TWO

PREPARATION

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.

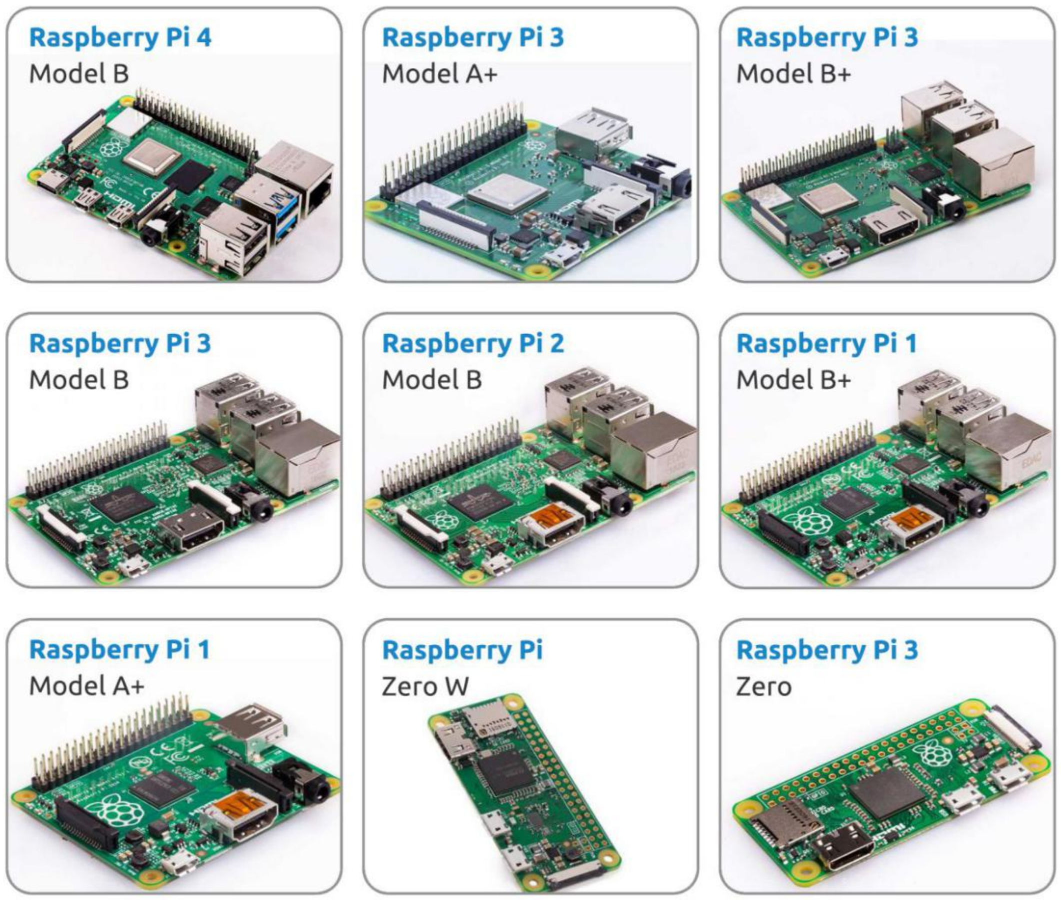
2.1 What Do We Need?

2.1.1 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.

Our kit applies to the following versions of the product of Raspberry Pi:



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 SD card to store all its files and the Raspbian operating system. You will need a micro SD card with a capacity of at least 8 GB.

2.1.2 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.

2.2 Installing the OS

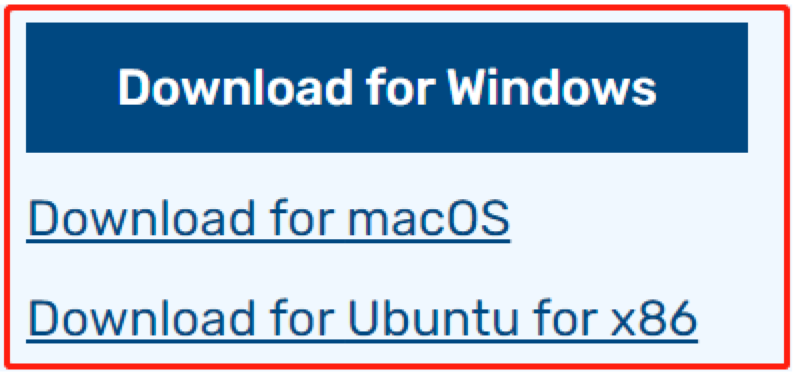
Required Components

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.

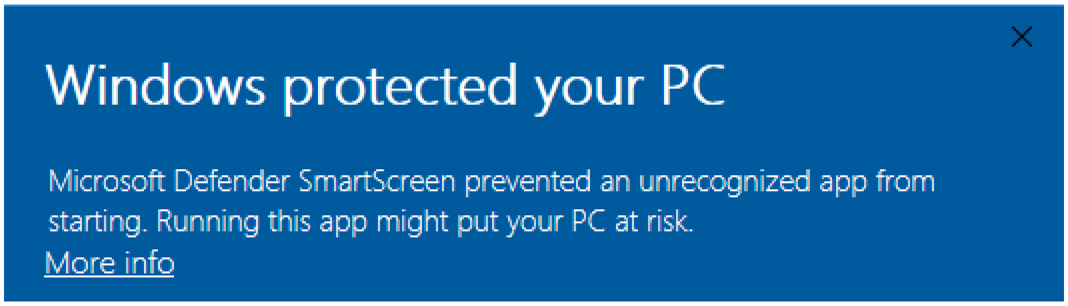
Visit the download page: https://www.raspberrypi.org/software/. Click on the link for the Raspberry Pi Imager that matches your operating system, when the download finishes, click it to launch the installer.



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.


Windows protected your PC

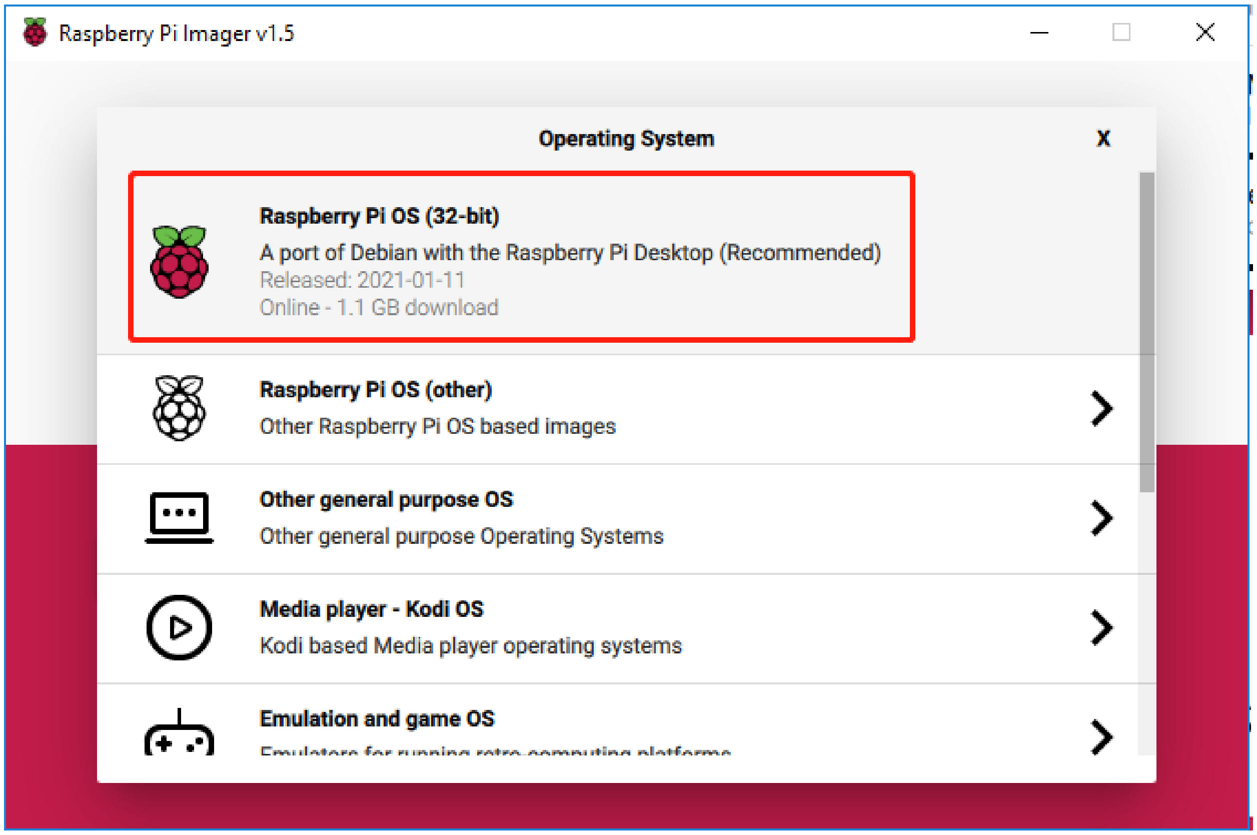
Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.
More info

Step 3

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

Step 4

In the Raspberry Pi Imager, select the OS that you want to install and the SD card you would like to install it on.



Raspberry Pi OS (32-bit)
A port of Debian with the Raspberry Pi Desktop (Recommended)
Released: 2021-01-11
Online - 1.1 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general purpose OS
Other general purpose Operating Systems

Media player - Kodi OS
Kodi based Media player operating systems

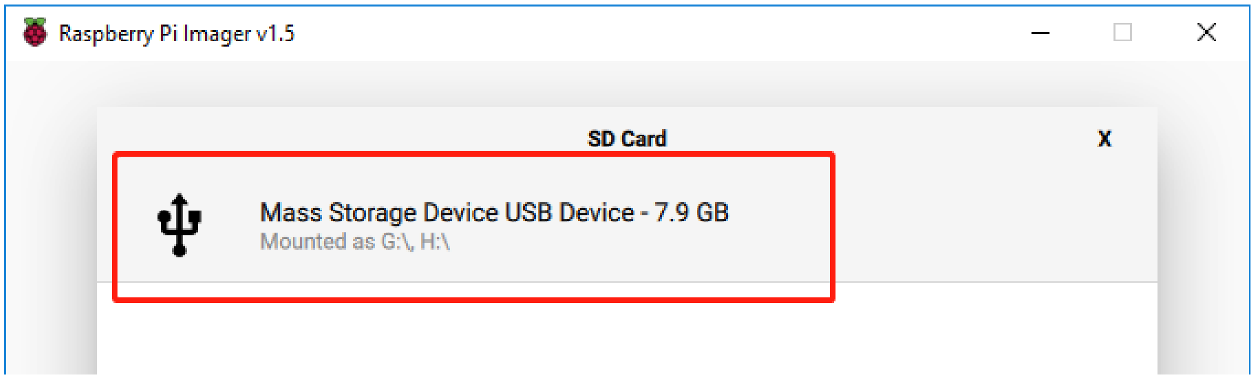
Emulation and game OS

Note:

1) You will need to be connected to the internet the first time.
2) That OS will then be stored for future offline use(lastdownload.cache, C:/Users/yourname/AppData/Local/Raspberry Pi/Imager/cache). So the next time you open the software, it will have the display "Released: date, cached on your computer".

Step 5

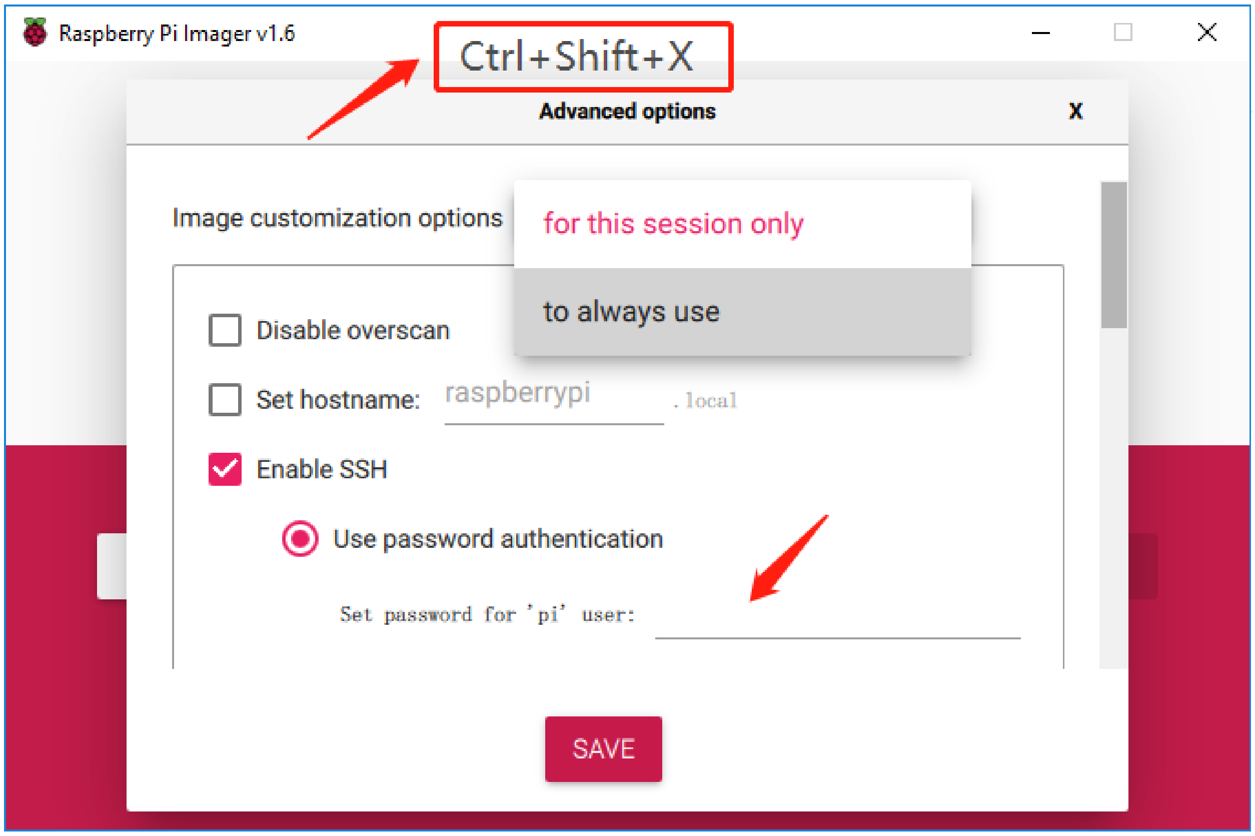
Select the SD card you are using.



Mass Storage Device USB Device - 7.9 GB
Mounted as G:\, H:\

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.


Raspberry Pi Imager v1.6

Ctrl+Shift+X

Advanced options

Image customization options: for this session only / to always use

Disable overscan: unchecked

Set hostname: unchecked, raspberrypi .local

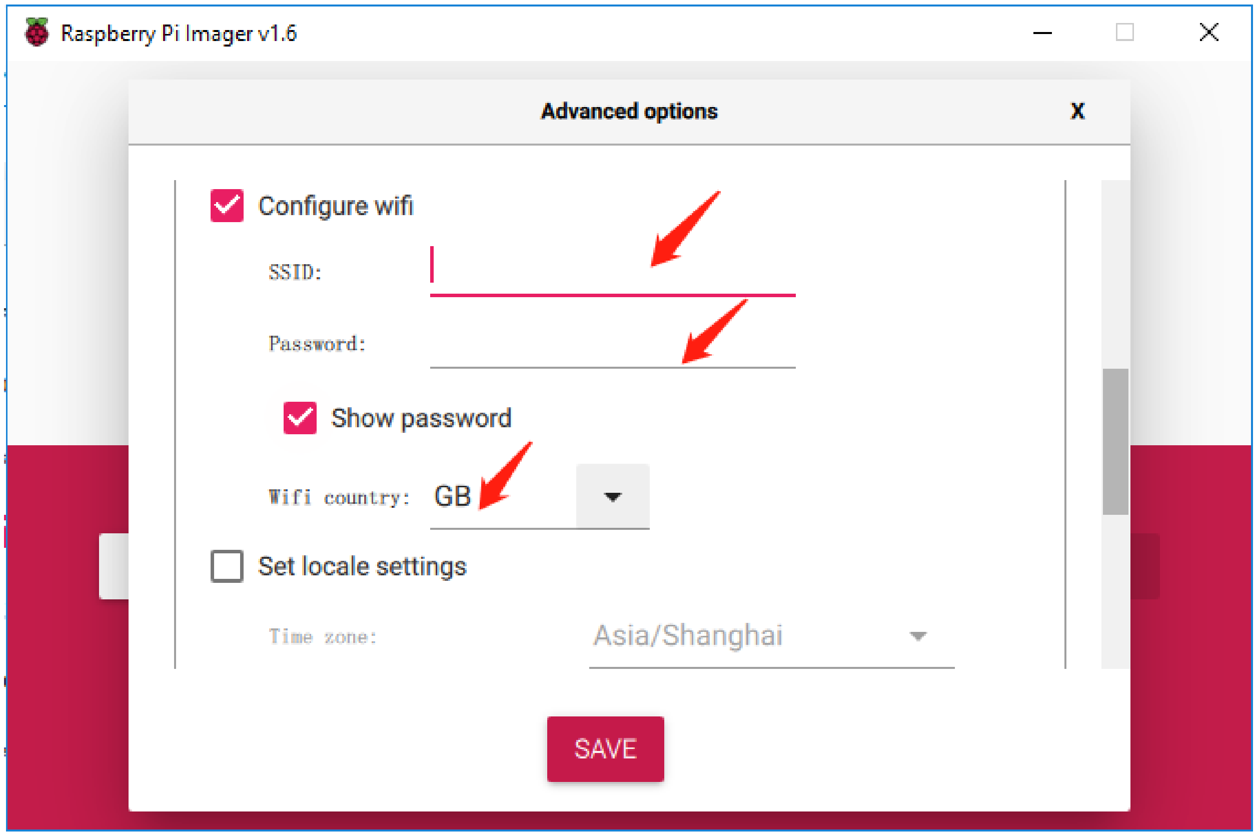
Enable SSH: checked

Use password authentication: selected

Set password for 'pi' user:

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



Configure wifi [✓]

SSID:

Password:

Show password [✓]

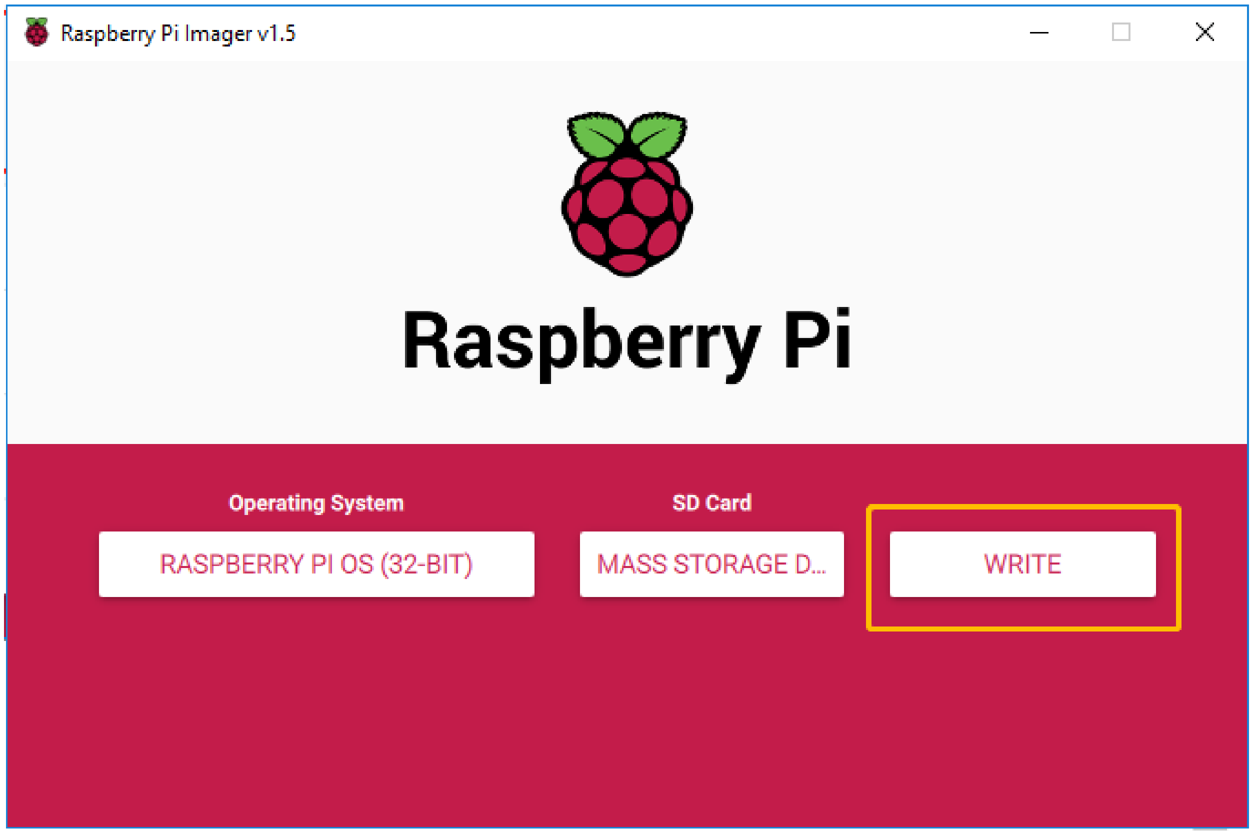
Wifi country: GB

Set locale settings [ ]

Time zone: Asia/Shanghai

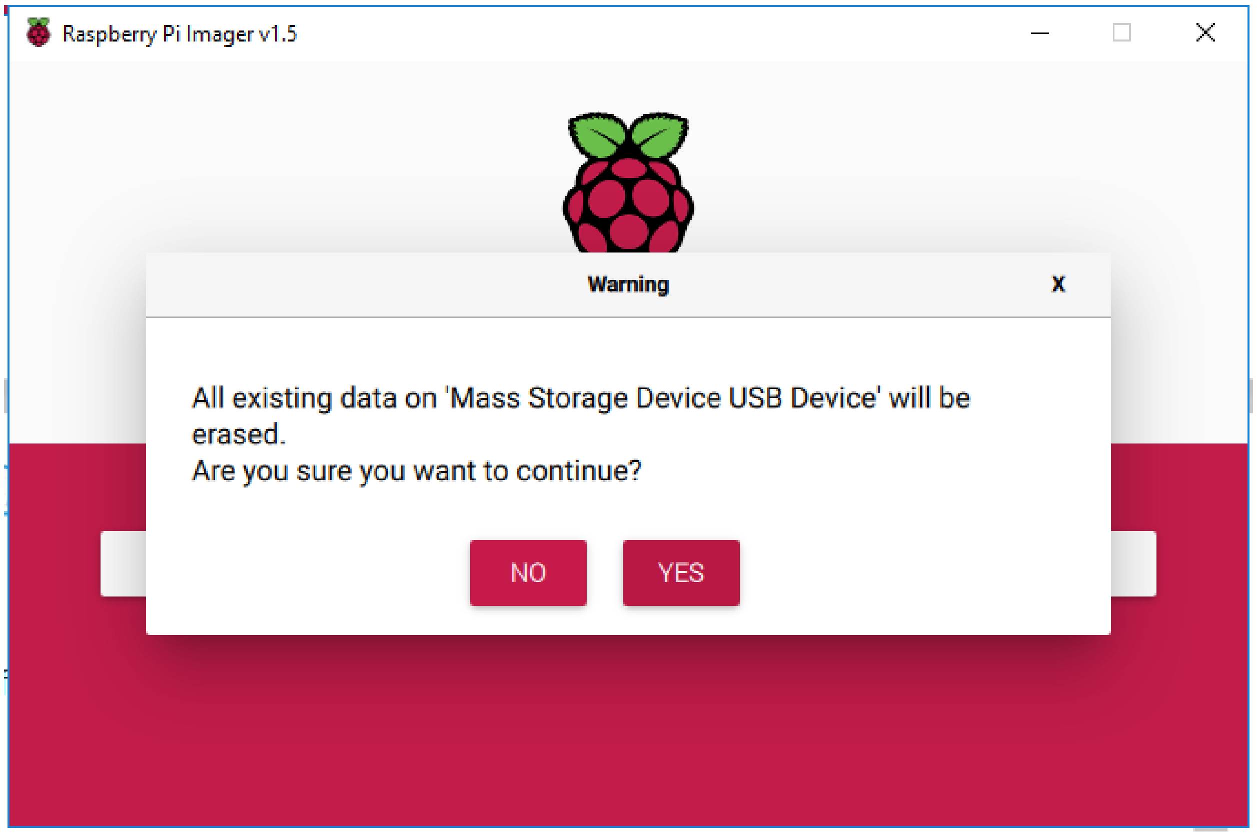
Step 7

Click the **WRITE** button.



Step 8

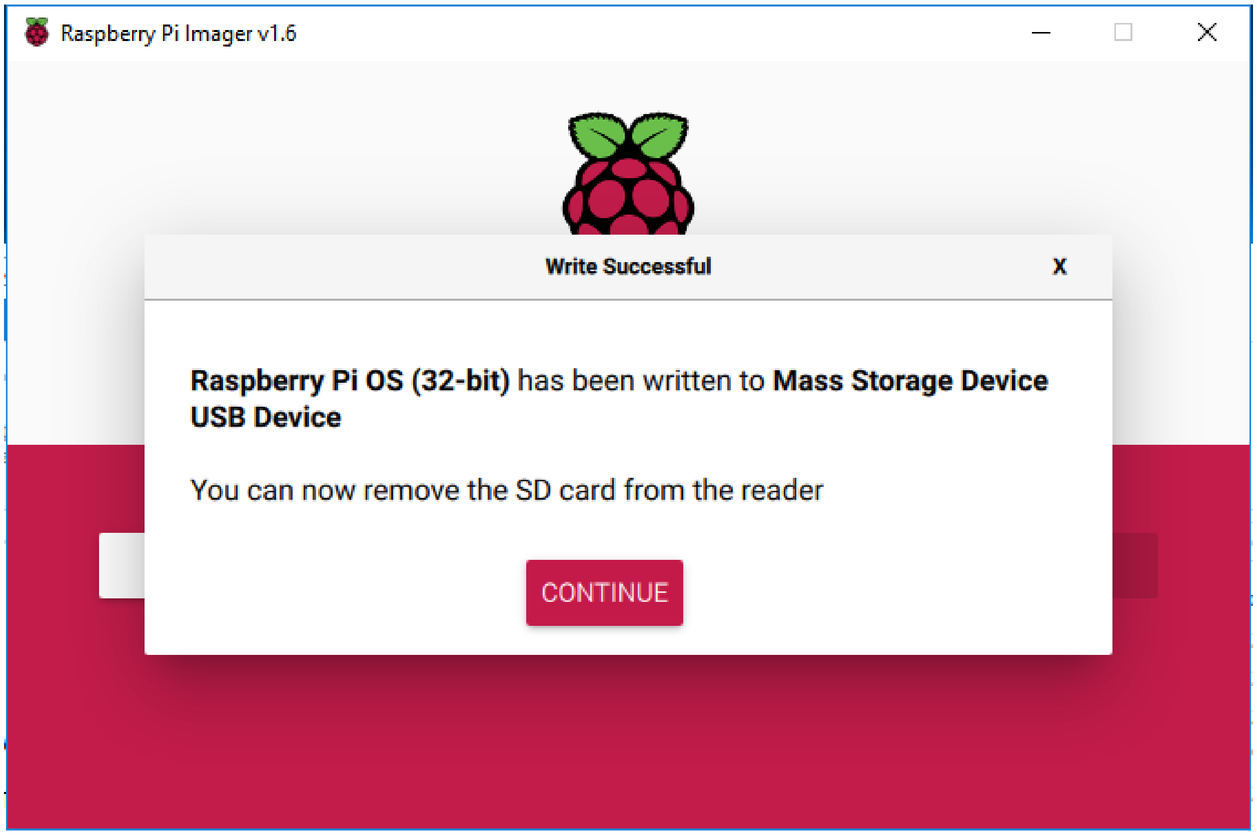
If your SD card currently has any files on it, you may wish to back up these files first to prevent you from permanently losing them. If there is no file to be backed up, click **Yes**.



All existing data on 'Mass Storage Device USB Device' will be erased.
Are you sure you want to continue?

Step 9

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



Raspberry Pi OS (32-bit) has been written to **Mass Storage Device USB Device**

You can now remove the SD card from the reader

2.3 Set up Your Raspberry Pi

2.3.1 If You Have a Screen

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

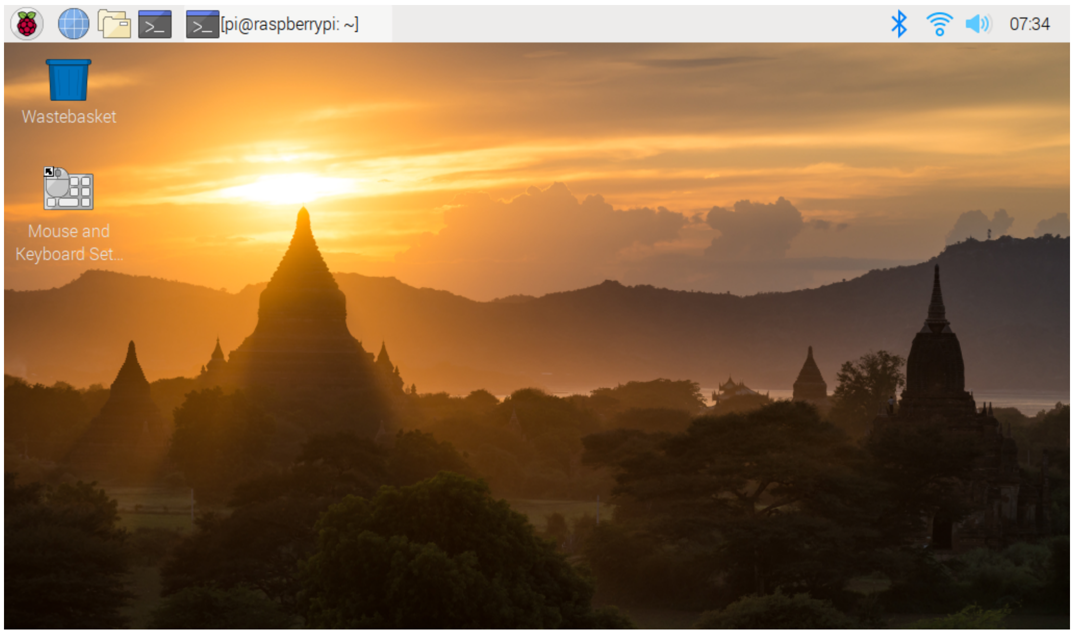
Required Components

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.



2.3.2 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.

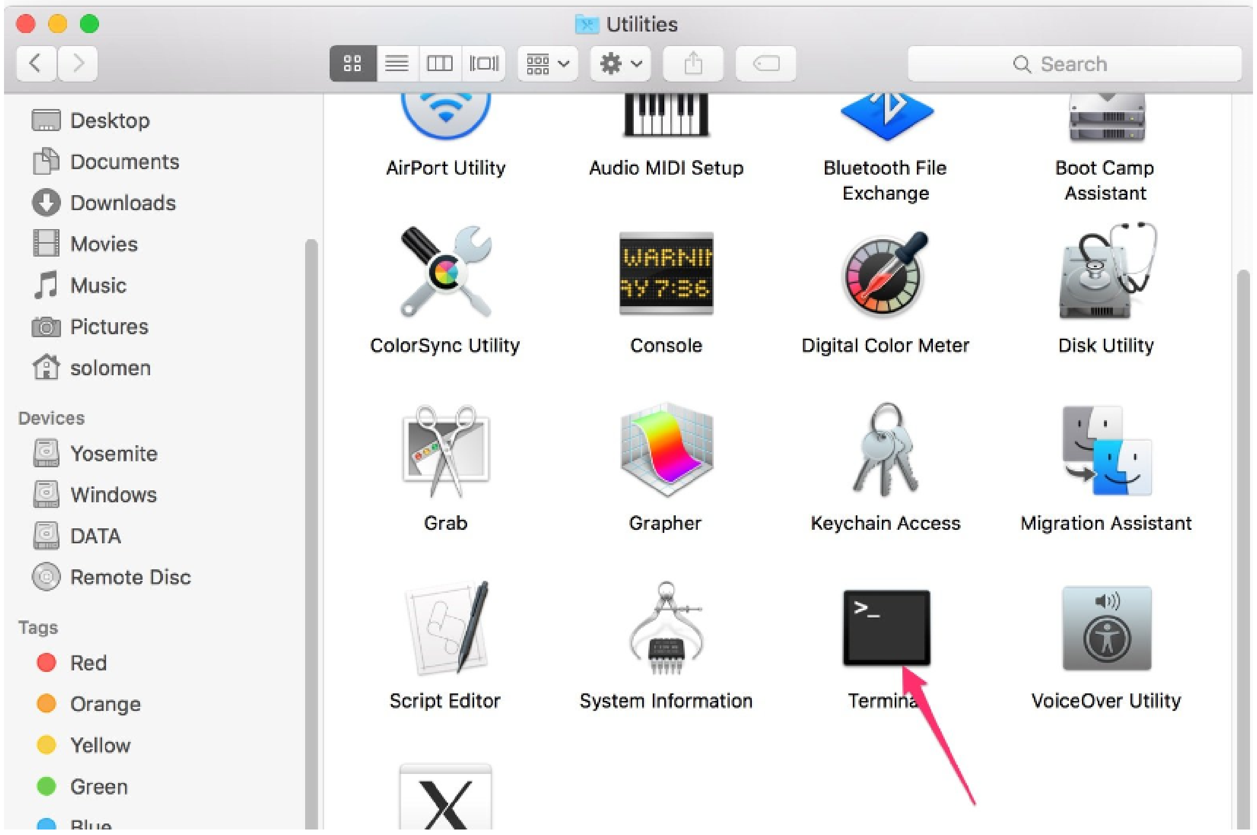
Use the SSH Remote Control

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.



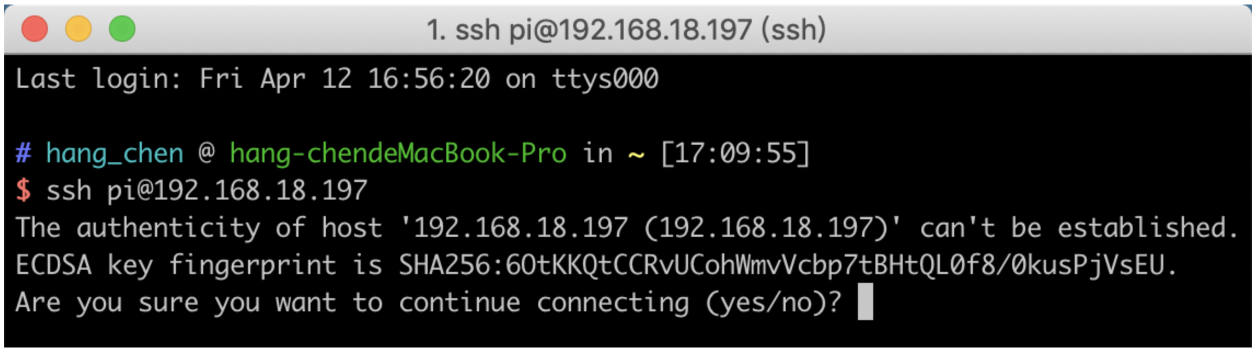
Step 2

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

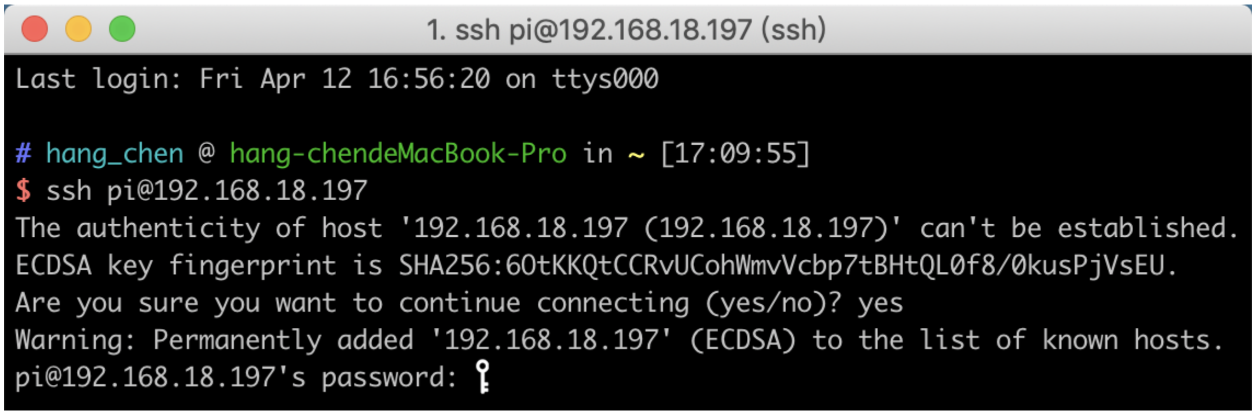
Input"yes".



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

Step 4

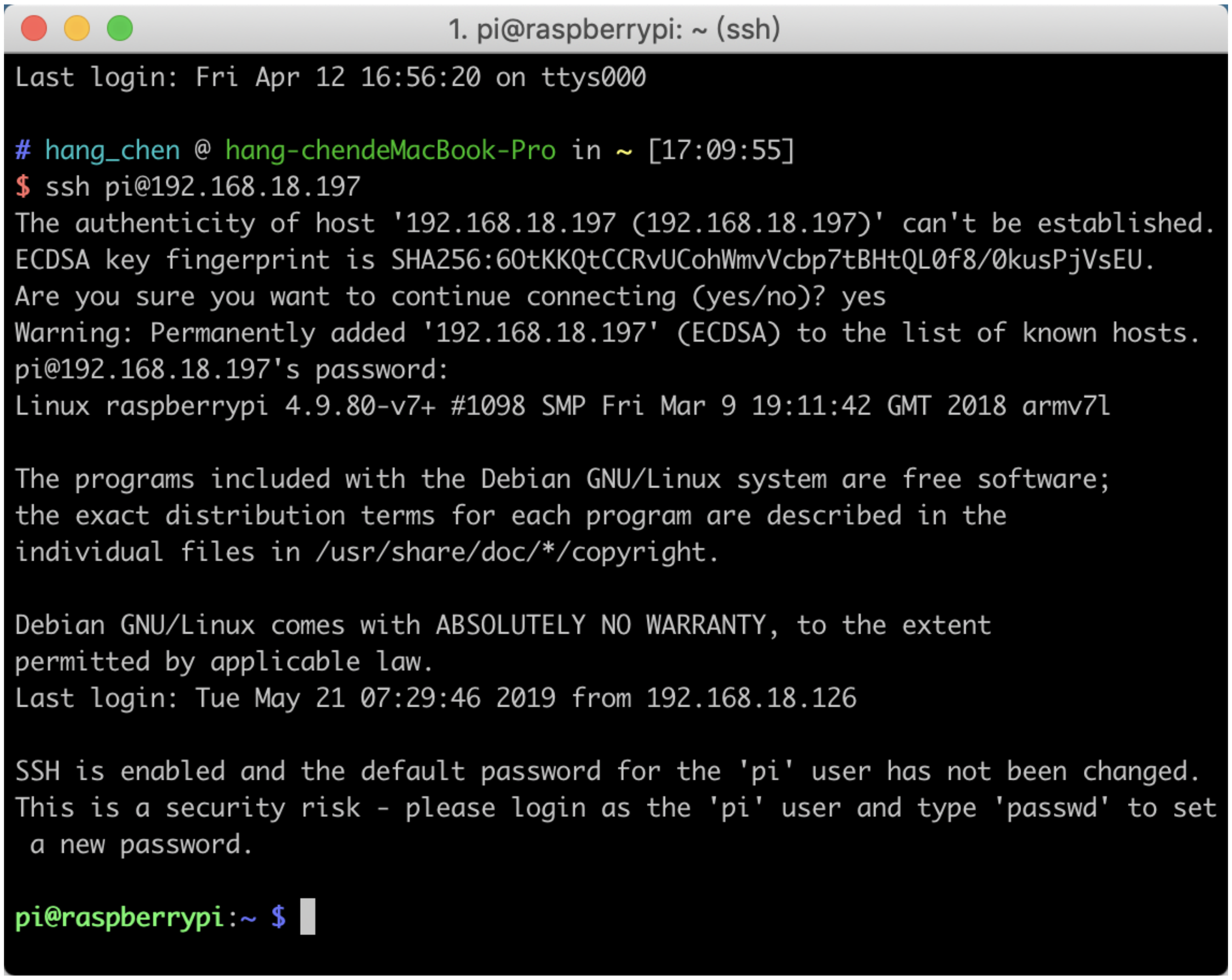
Input the passcode and the default password is **raspberry**.



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
```

Step 5

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



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $
```

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

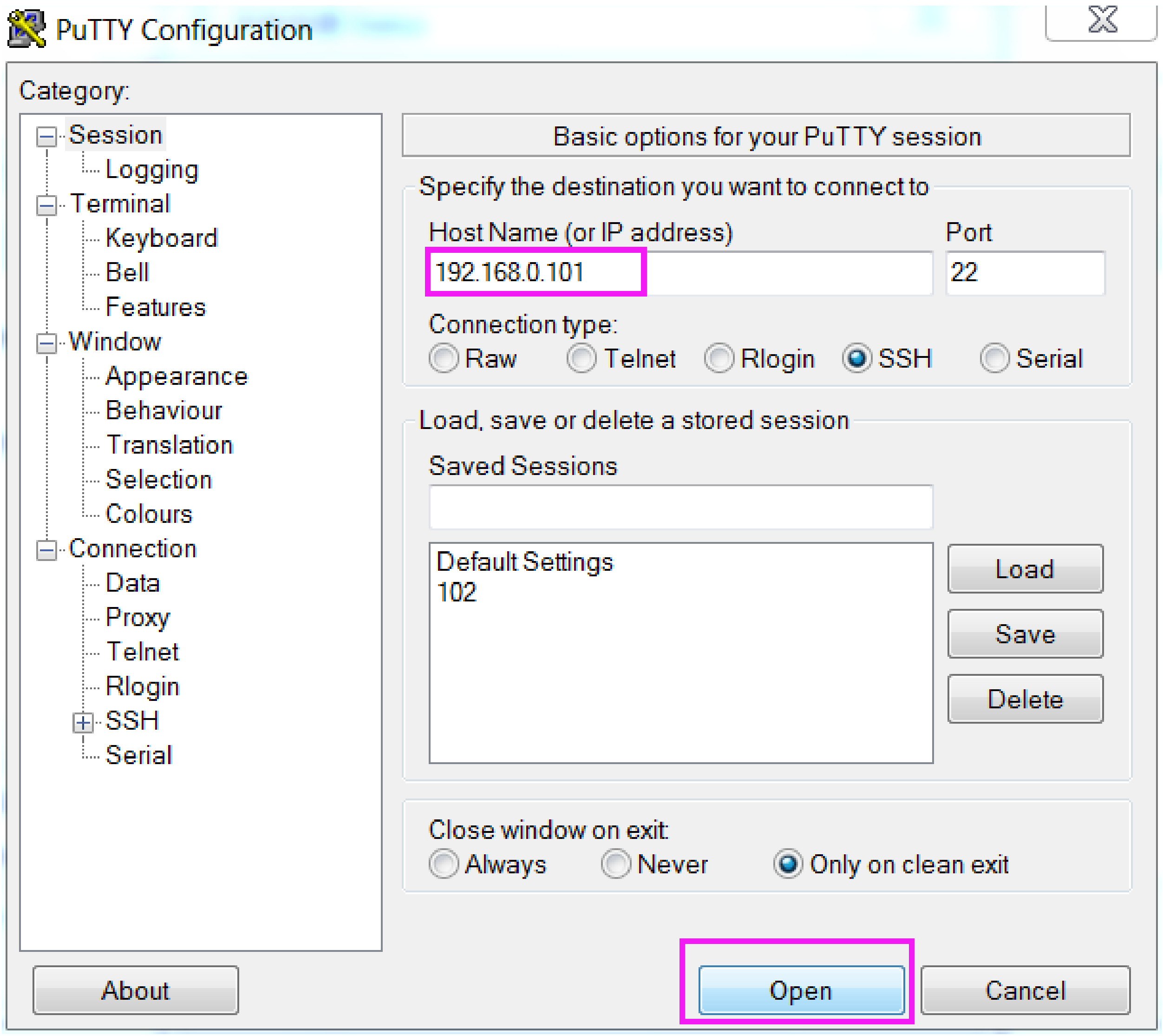
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

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).



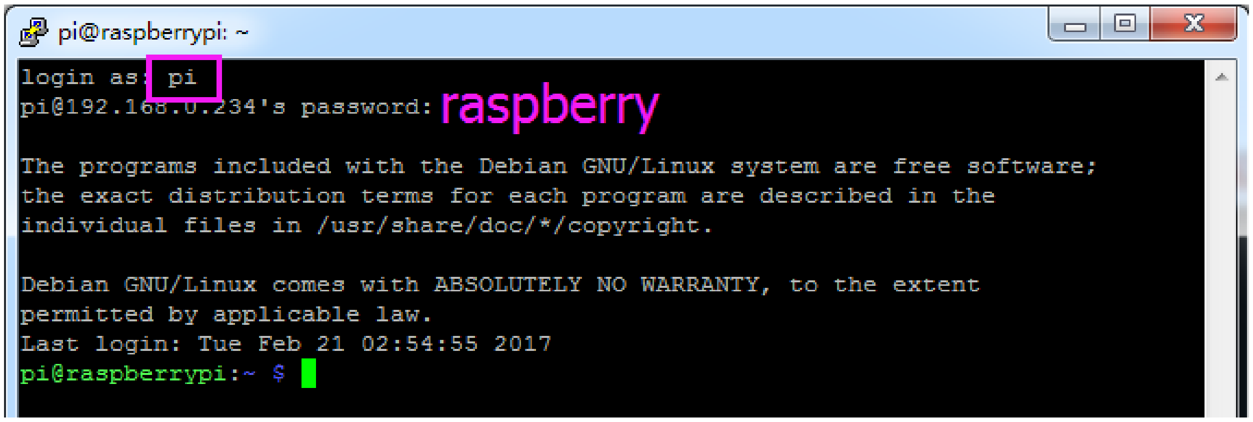
Host Name (or IP address): 192.168.0.101
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Default Settings
102
Close window on exit: Always Never [✓ Only on clean exit]

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).



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

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

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

Step 5

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

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.

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

THREE

LIBRARIES

Two important libraries are used in programming with Raspberry Pi, and they are wiringPi and RPi.GPIO. The Raspbian OS image of Raspberry Pi installs them by default, so you can use them directly.

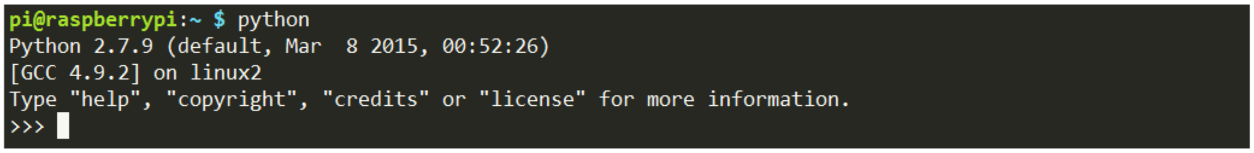
3.1 RPi.GPIO

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:

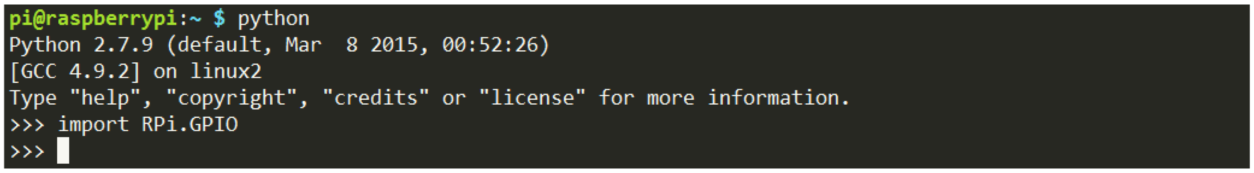
```
python
```



```
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.

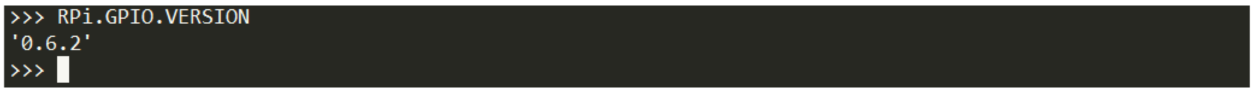
```
import RPi.GPIO
```



```
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
>>>
```

Then, type in RPi.GPIO.VERSION to check its version.

```
RPi.GPIO.VERSION
```



```
>>> RPi.GPIO.VERSION
'0.6.2'
>>>
```

If you want to quit python CLI, type in:

```
exit()
```

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

3.2 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.

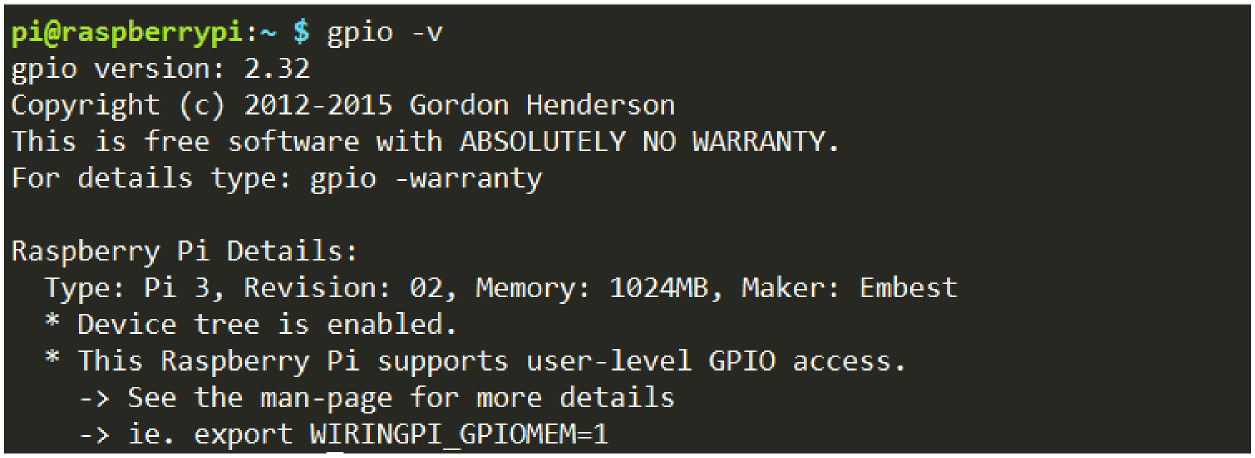
`wiringPi` includes lots of GPIO commands which enable you to control all kinds of interfaces on Raspberry Pi.

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.

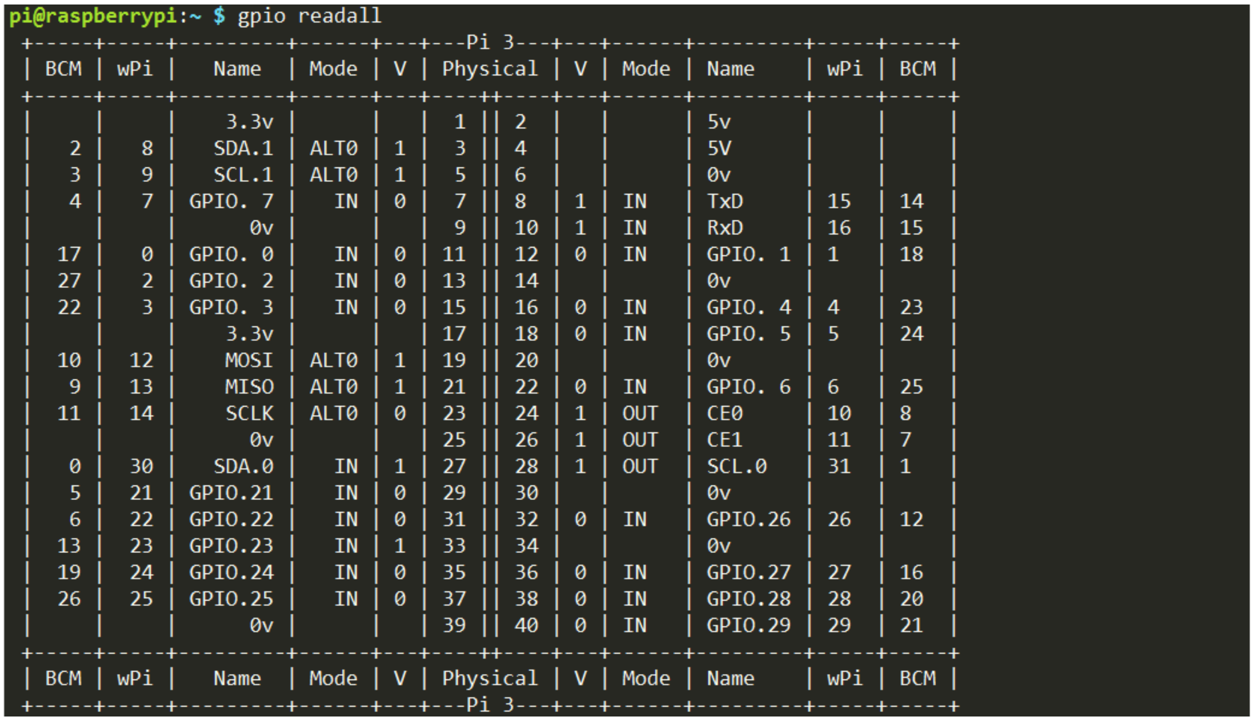
```
gpio -v
```



```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.32
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Embest
  * Device tree is enabled.
  * This Raspberry Pi supports user-level GPIO access.
    -> See the man-page for more details
    -> ie. export WIRINGPI_GPIOMEM=1
```

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.

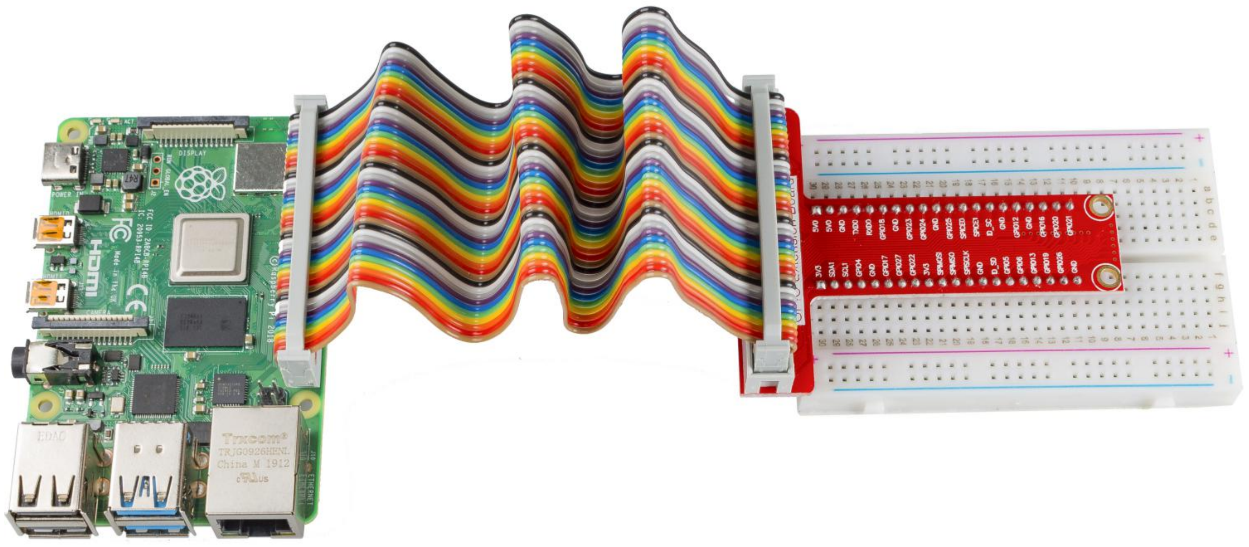
CHAPTER

FOUR

RASPBERRY PI GPIO EXTENSION BOARD

We apply the GPIO Extension Board to extend the pins of Raspberry Pi to the breadboard and avoid damage caused by frequent plugging and unplugging.

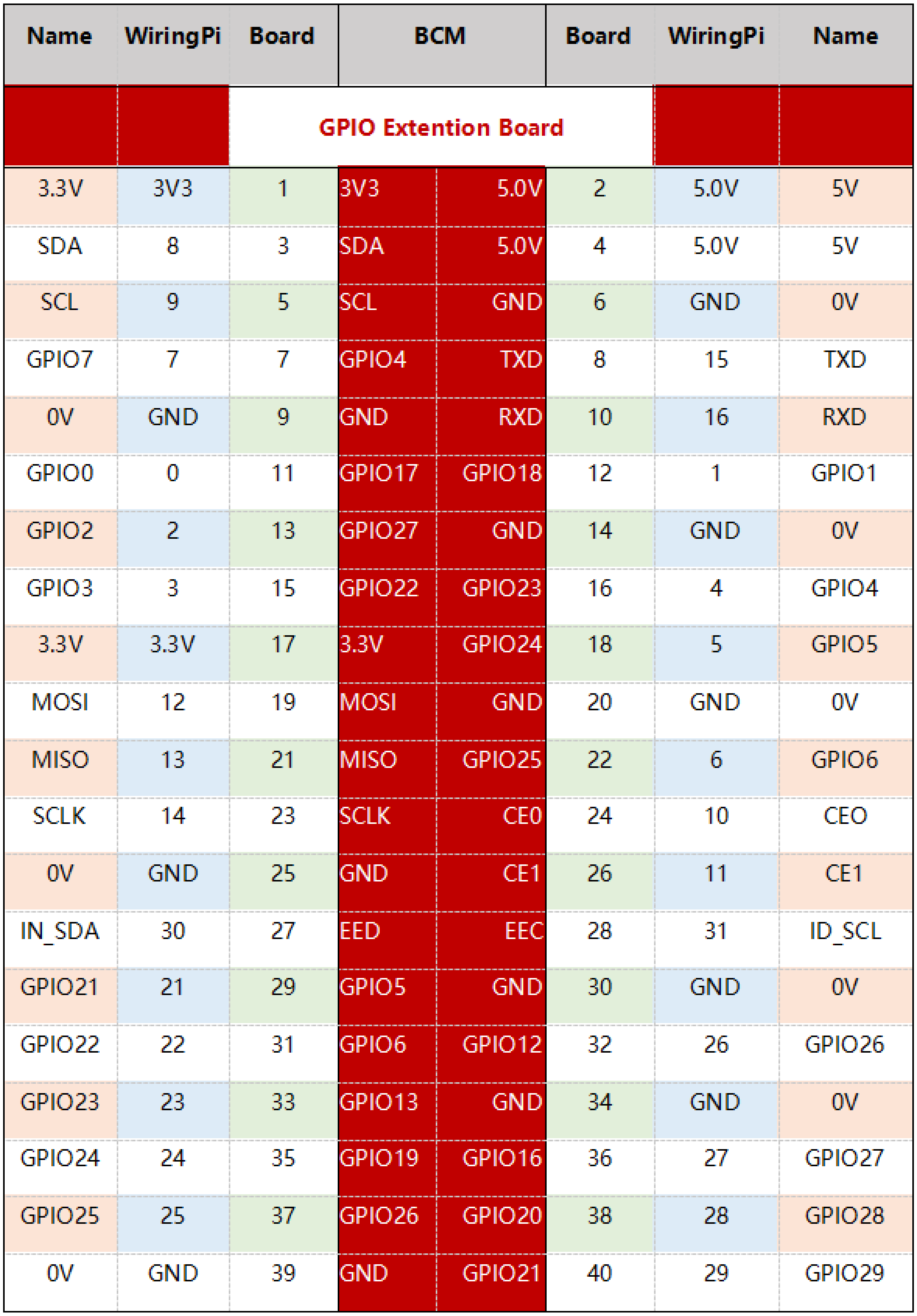
Here, we apply a 40-pin GPIO Extension board and a 40-pin GPIO cable. In case of the potential risk of short circuit, you must build your circit in strict accordance with the following picture.



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.



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

Change directory to */home/pi*.

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

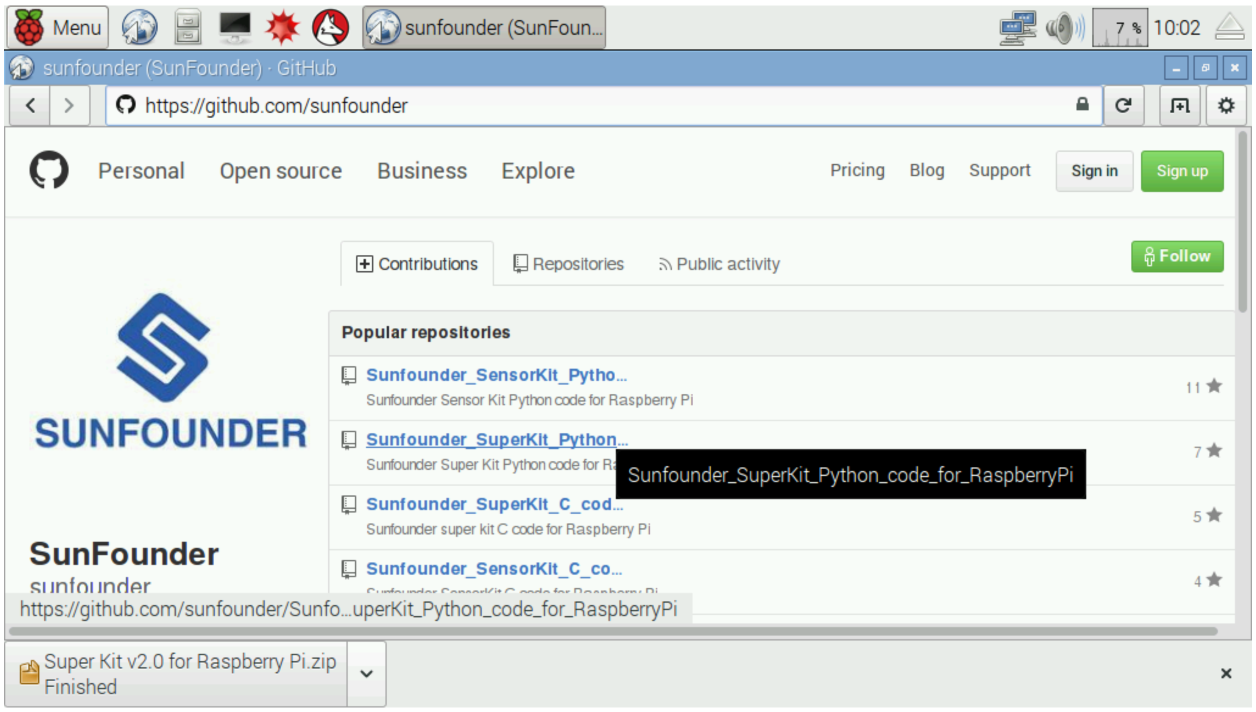
Note: cd, short for change directory is to change to the intended directory from the current path. Informally, here is to go to the path */home/pi/*.

Clone the repository from GitHub (C code and python code).

```
git clone https://github.com/sunfounder/Sunfounder_SuperKit_C_code_for_RaspberryPi.git
```

```
git clone https://github.com/sunfounder/Sunfounder_SuperKit_Python_code_for_
↪RaspberryPi.git
```

The advantage of this method is that, you can download the latest code any time you want, and then place the code under the path */home/pi/*. But in case of incorrect typing which is possible especially when you're strange to the commands, you can just enter github.com/sunfounder at the address bar of a web browser, and on the page directed find the code for Super Kit.



Popular repositories

Sunfounder_SensorKit_Pytho...
Sunfounder Sensor Kit Python code for Raspberry Pi
11

Sunfounder_SuperKit_Python...
Sunfounder Super Kit Python code for Ra
7

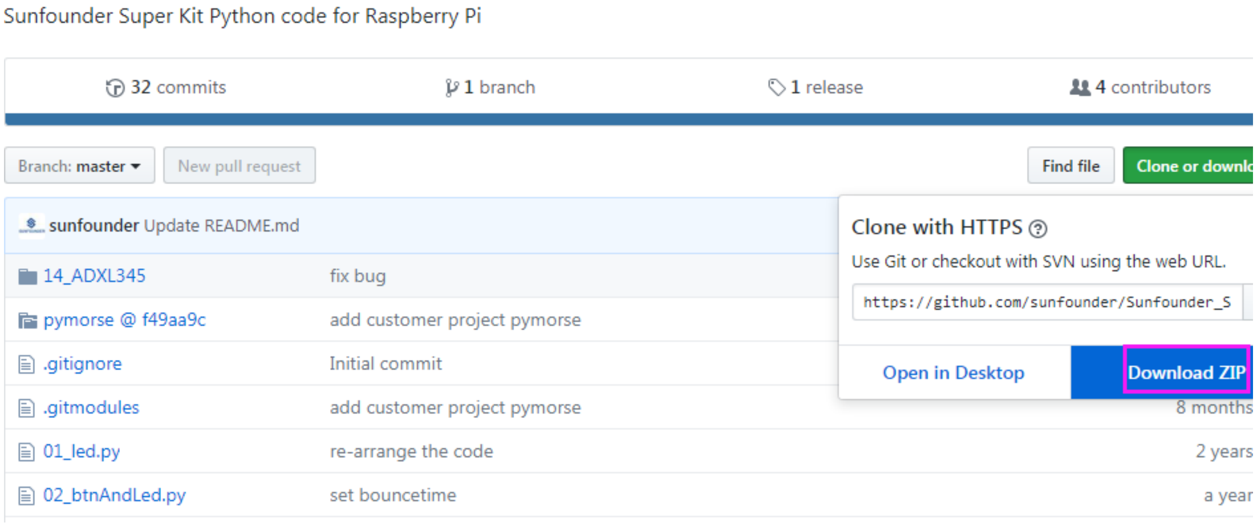
Sunfounder_SuperKit_Python_code_for_RaspberryPi

Sunfounder_SuperKit_C_cod...
Sunfounder super kit C code for Raspberry Pi
5

Sunfounder_SensorKit_C_co...
4

https://github.com/sunfounder/Sunfo...uperKit_Python_code_for_RaspberryPi

Click on the repository. On the page directed, click **Clone or download** on the right side.



Sunfounder Super Kit Python code for Raspberry Pi

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

https://github.com/sunfounder/Sunfounder_S

sunfounder Update README.md

14_ADXL345	fix bug	
pymorse @ f49aa9c	add customer project pymorse	
.gitignore	Initial commit	
.gitmodules	add customer project pymorse	8 months
01_led.py	re-arrange the code	2 years
02_btnAndLed.py	set bouncetime	a year

After download, transfer the package to */home/pi/*.

Now you can start the experiments. Let’s rock!

CHAPTER

SIX

LESSONS

6.1 Lesson 1 Blinking LED

6.1.1 Introduction

In this lesson, we will learn how to program Raspberry Pi to make an LED blink. You can play numerous tricks with an LED as you want. Now get to start and you will enjoy the fun of DIY at once!

6.1.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * LED

- 1 * Resistor (220)

- Jumper wires

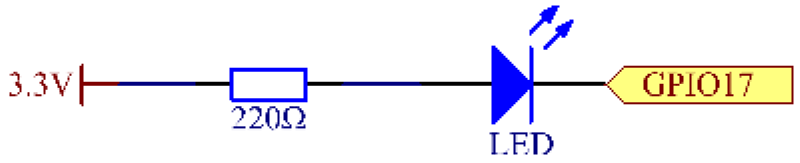
6.1.3 Principle

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).

When 2V-3V forward voltage is supplied to an LED, it will blink only if forward currents flow through the LED. Usually there are red, yellow, green, blue and color-changing LEDs which change color with different voltages. LEDs are widely used due to their low operating voltage, low current, luminescent stability and small size.

LEDs are diodes too. Hence they have a voltage drop which usually varies from 1V to 3V depending on their types. Generally, they brighten if supplied with a 5mA–30mA current, and we usually use 10mA–20mA. Thus when an LED is used, it is necessary to connect a current-limiting resistor to protect it from being burnt.

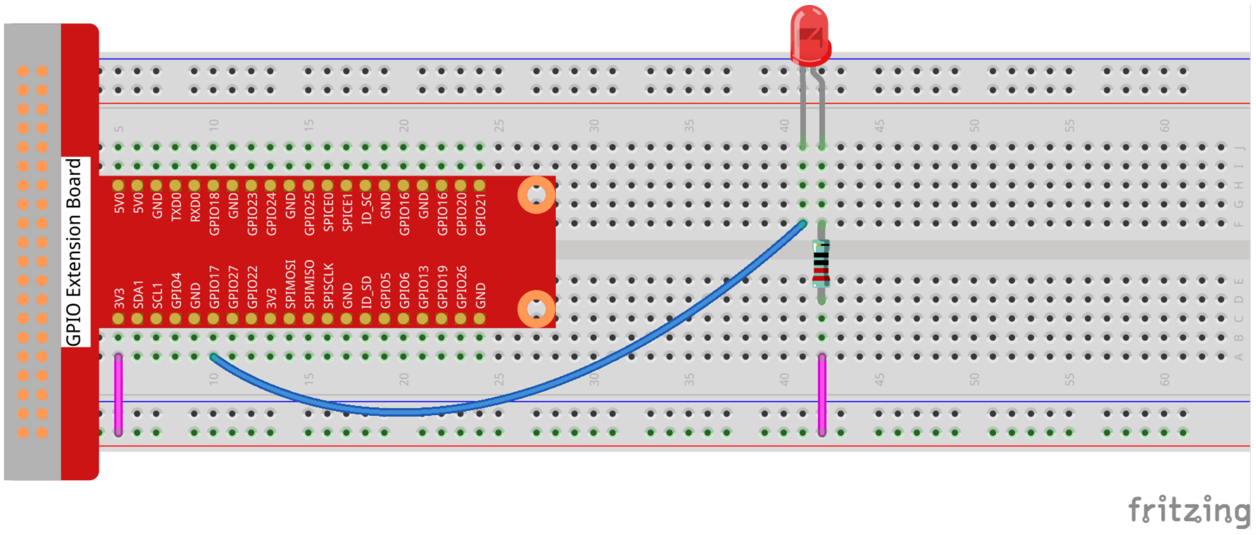
6.1.4 Schematic Diagram



In this experiment, connect a 220 resistor to the anode of the LED, then the resistor to 3.3 V, and connect the cathode of the LED to GPIO17 (See Raspberry Pi Pin Number Introduction). Write 1 to GPIO17, and the LED will stay off; write 0 to GPIO17, and then the LED will blink, just as indicated by the principle above.

6.1.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/01_LED
```

Step 3: Compile.

```
gcc led.c -o led -lwiringPi
```

Step 4: Run.

```
sudo ./led
```

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

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    printf("linker LedPin : GPIO %d(wiringPi pin)\n",LedPin); //when initialize␣
↪wiring successfully,print message to screen

    pinMode(LedPin, OUTPUT);

    while(1){
            digitalWrite(LedPin, LOW);  //led on
            printf("led on...\n");
            delay(500);
            digitalWrite(LedPin, HIGH);  //led off
            printf("...led off\n");
            delay(500);
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 01_led.py
```

Now, you should see the LED blink.

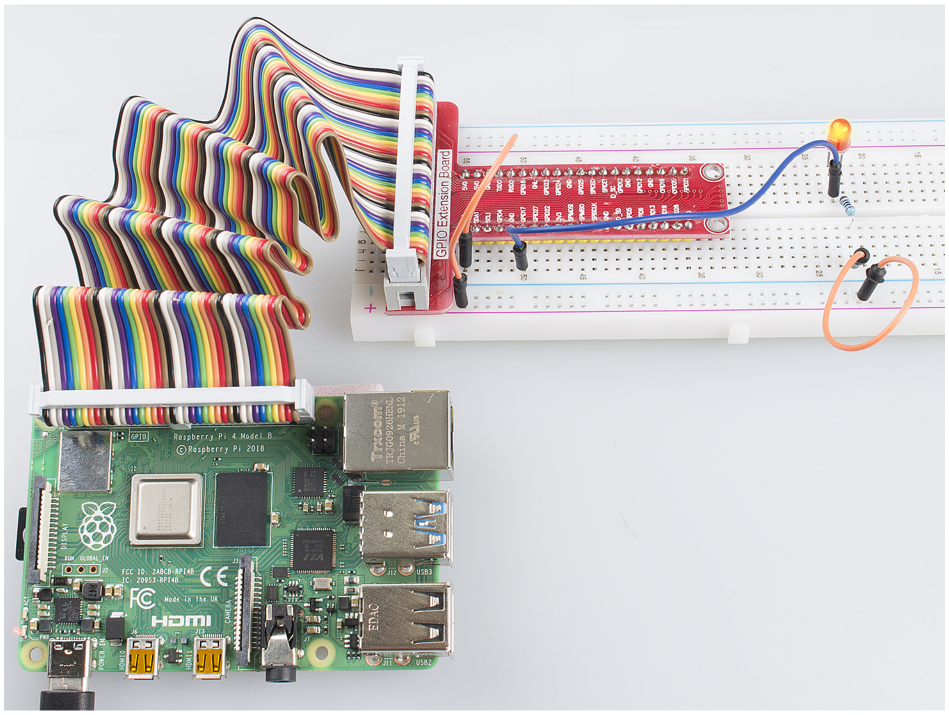
Code

```
import RPi.GPIO as GPIO
import time

LedPin = 17

def setup():
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop():
    while True:
        print ("...led on")
```

(continues on next page)

(continued from previous page)

```
        GPIO.output(LedPin, GPIO.LOW)   # led on
        time.sleep(0.5)
        print ("led off...")
        GPIO.output(LedPin, GPIO.HIGH) # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(LedPin, GPIO.HIGH)     # led off
    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()
```



6.1.6 Further Exploration

If you want the LED to speed up the blinking, just change the delay time. For example, change the time to *delay (200)* in the program, recompile and run, and then you will see the LED blink faster.

6.1.7 Summary

Raspberry Pi packages many low-level detail designs, which enable you to explore your own apps more conveniently. Maybe that is the charm of Raspberry Pi.

Now you have already learnt how to use the Raspberry Pi GPIO to blink an LED. Keep moving to the next contents.

Tips

For any **TECHNICAL** questions, add a topic under **FORUM** section on our website www.sunfounder.com and we'll reply as soon as possible. For **NON-TECH** questions like order issues, please **email** service@sunfounder.com.

6.2 Lesson 2 Controlling an LED by a Button

6.2.1 Introduction

In this lesson, we will learn how to turn an LED on or off by a button.

6.2.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * LED

- 1 * Button

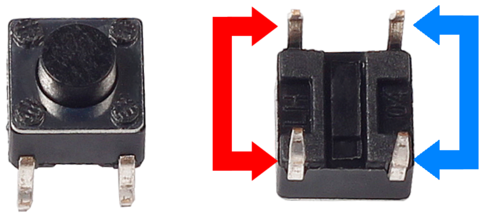
- 1 * Resistor (220)

- Jumper wires

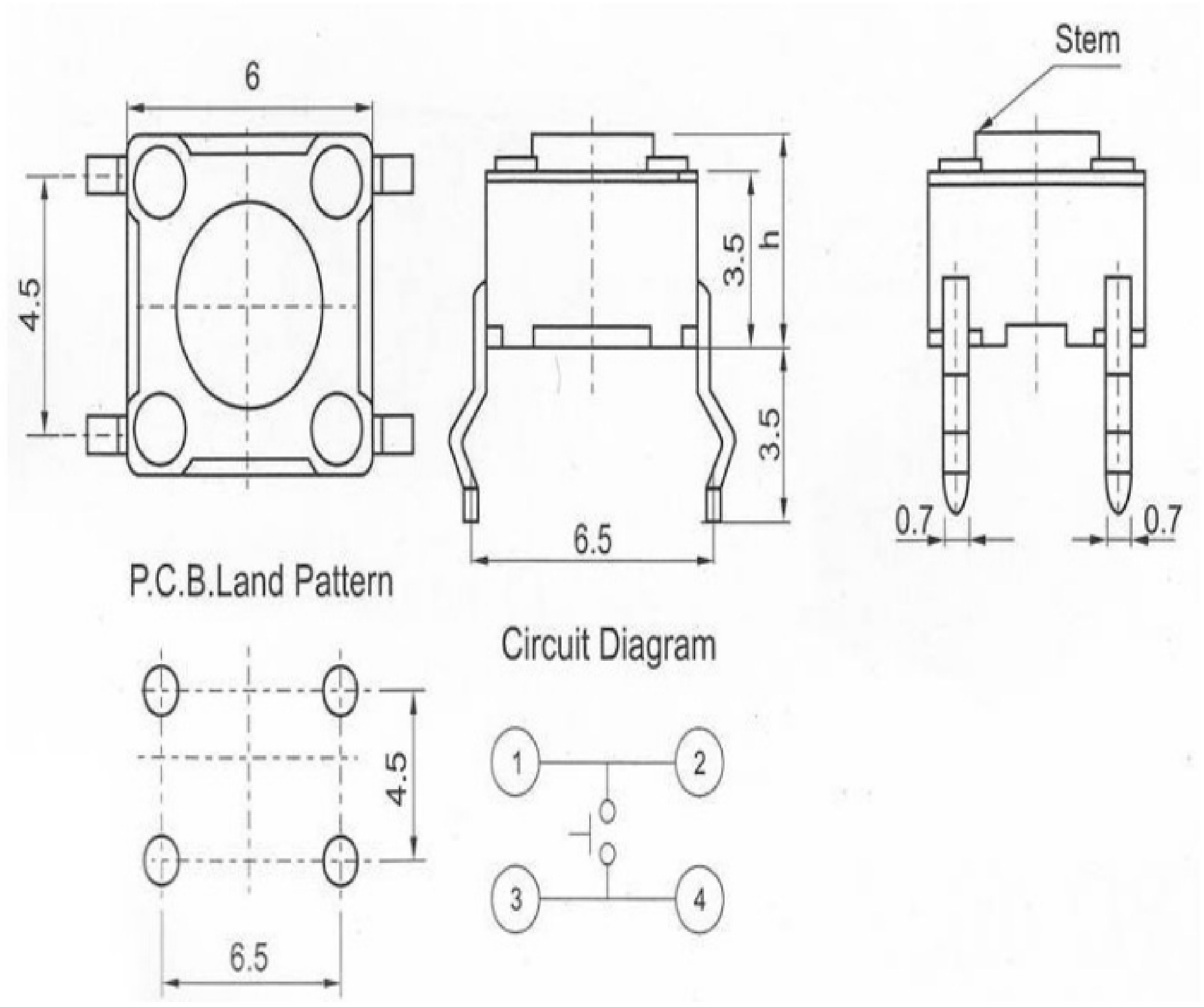
6.2.3 Principle

Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect 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. Pins pointed out by the arrows of same color are meant to be connected.



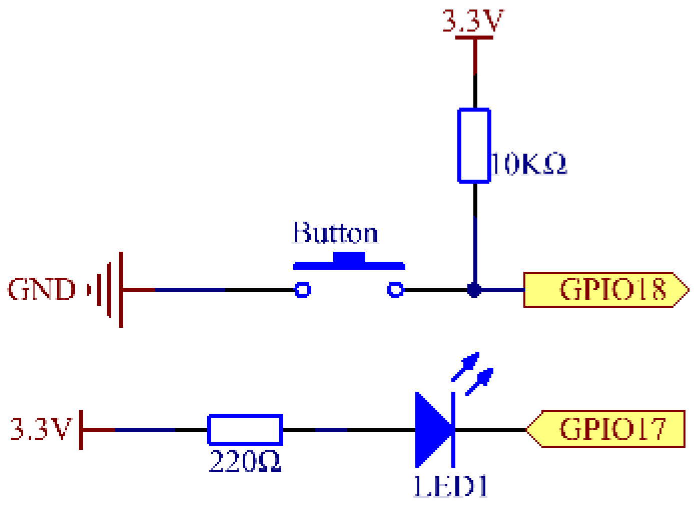
When the button is pressed, the pins pointed by the blue arrow will connect to the pins pointed by the red arrow (see the above figure), thus closing the circuit, as shown in the following diagrams.



Generally, the button can be connected directly to the LED in a circuit to turn on or off the LED, which is comparatively simple. However, sometimes the LED will brighten automatically without any button pressed, which is caused by various kinds of external interference. In order to avoid this interference, a pull-up resistor is used – usually connect a 1K–10K resistor between the button and VCC. It can be connected to VCC to consume the interference when the button is off.

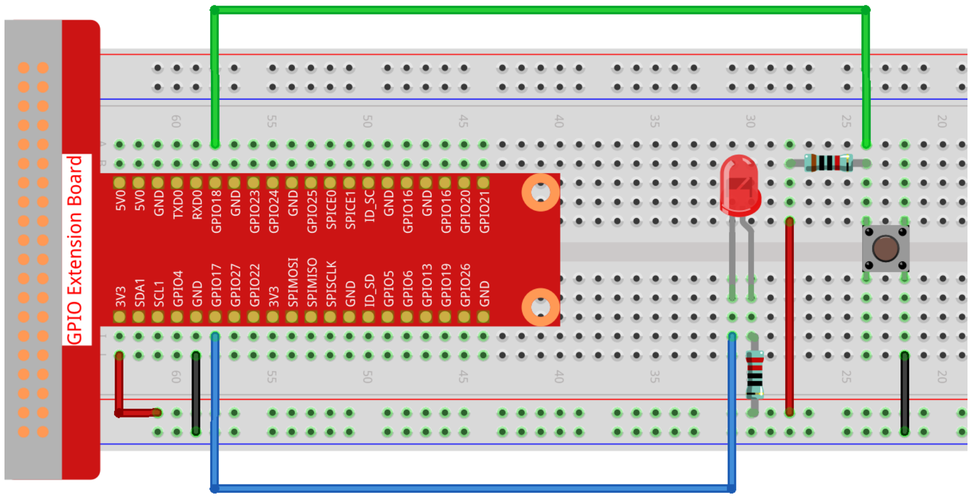
6.2.4 Schematic Diagram

Use a normally open button as the input of Raspberry Pi. When the button is pressed, the GPIO connected to the button will turn into low level (0V). We can detect the state of the GPIO connected to the button through programming. That is, if the GPIO 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.



6.2.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/02_BtnAndLed/
```

Step 3: Compile.

```
gcc BtnAndLed.c -o BtnAndLed -lwiringPi
```

Step 4: Run.

```
sudo ./BtnAndLed
```

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

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

    pullUpDnControl(ButtonPin, PUD_UP);  //pull up to 3.3V,make GPIO1 a stable level
    while(1){
        digitalWrite(LedPin, HIGH);
        if(digitalRead(ButtonPin) == 0){ //indicate that button has pressed down
            digitalWrite(LedPin, LOW);   //led on
        }
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 02_btnAndLed.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.

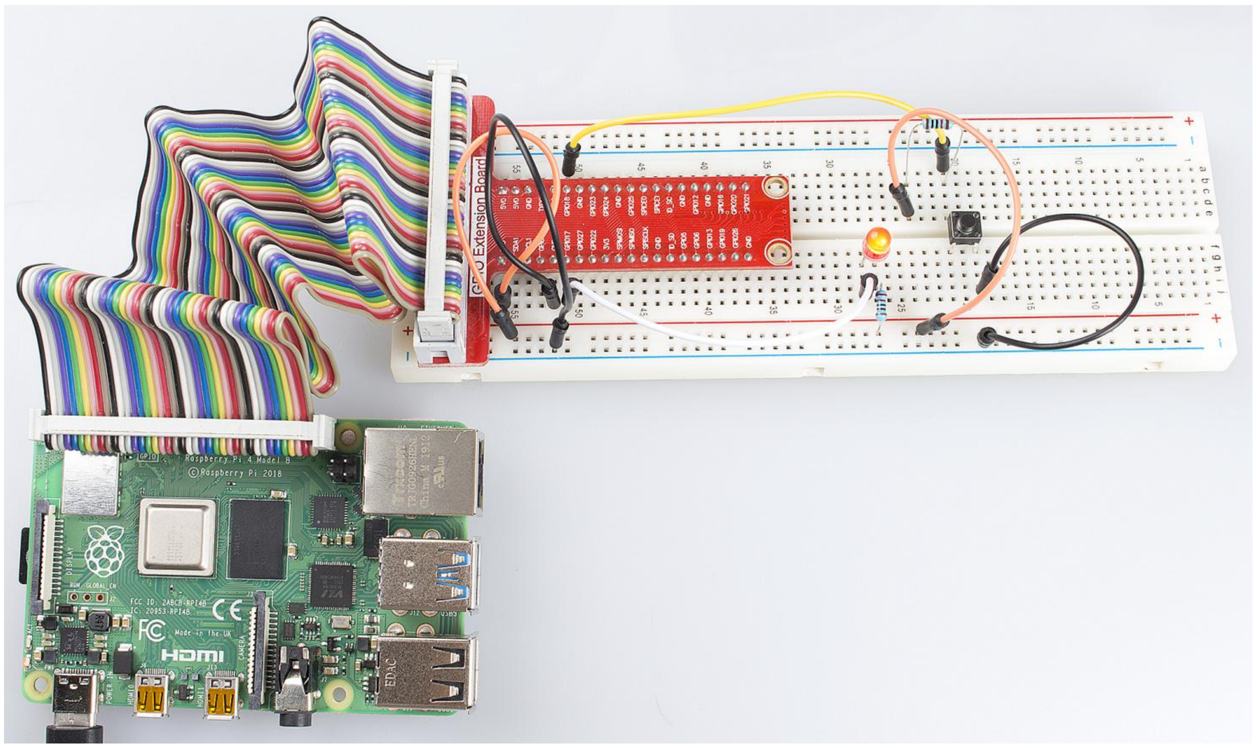
Code

```
import RPi.GPIO as GPIO
import time

LedPin = 17
BtnPin = 18

Led_status = 1

def setup():
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
```

(continues on next page)

(continued from previous page)

```
    GPIO.setup(LedPin, GPIO.OUT)    # Set LedPin's mode is output
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # Set BtnPin's mode is␣
↪input, and pull up to high level(3.3V)
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def swLed(ev=None):
    global Led_status
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)  # switch led status(on-->off; off-->on)
    if Led_status == 1:
        print ("led off...")
    else:
        print ("...led on")

def loop():
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed, bouncetime=200) #␣
↪wait for falling and set bouncetime to prevent the callback function from being␣
↪called multiple times when the button is pressed
    while True:
        time.sleep(1)   # Don't do anything

def destroy():
    GPIO.output(LedPin, GPIO.HIGH)     # led off
    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()
```



6.2.6 Summary

Through this experiment, you have learnt how to control the GPIOs of the Raspberry Pi by programming.

6.3 Lesson 3 Flowing LED Lights

6.3.1 Introduction

In this lesson, we will learn how to make eight LEDs blink in various effects as you want based on Raspberry Pi.

6.3.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

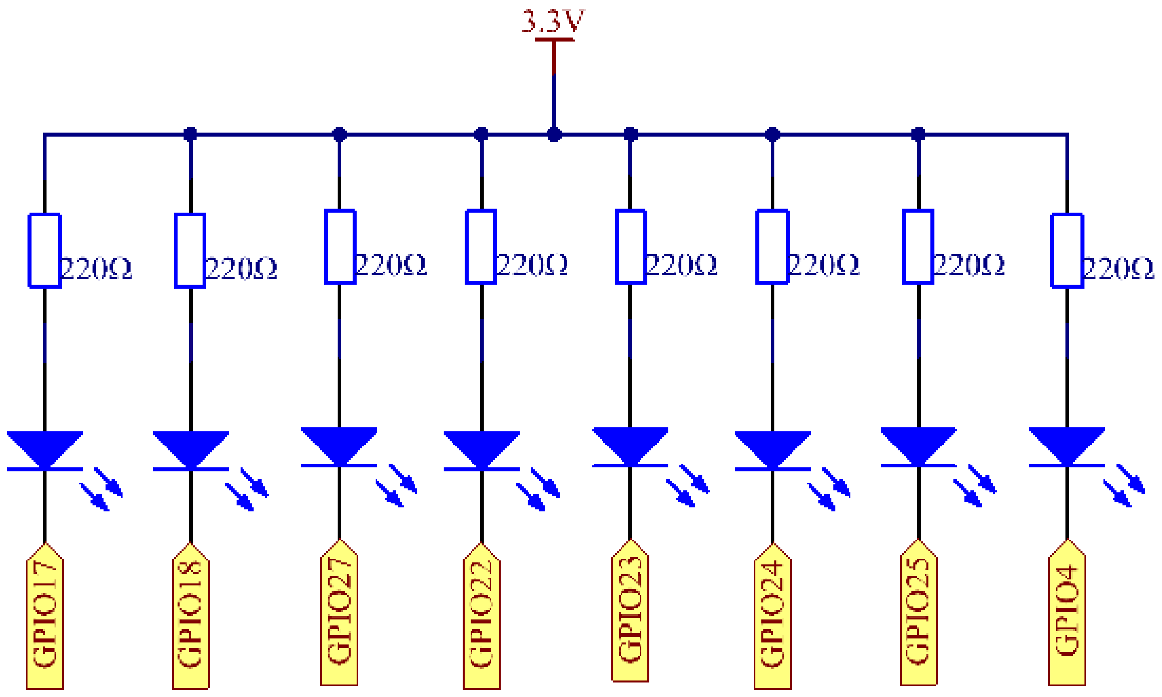
- 8 * LED

- 8 * Resistor (220)

- Jumper wires

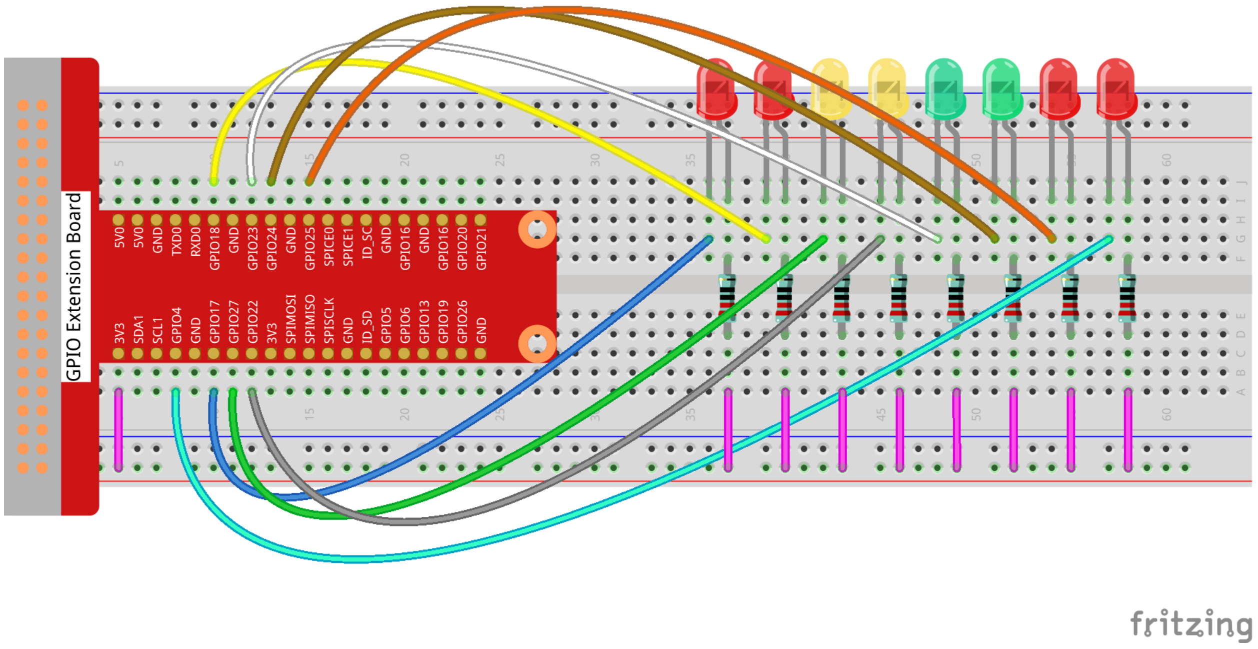
6.3.3 Schematic Diagram

Set GPIO17-GPIO25 to low level in turn by programming, and then LED0-LED7 will light up in turn. You can make eight LEDs blink in different effects by controlling their delay time and the order of lighting up.



6.3.4 Experimental Procedures

Step 1: Build the circuit.



Step 2: GPIO4 is the default pin for onewire driver (w1-gpio). In this lesson, we need to disable the onewire function and use it as an output pin.

```
sudo nano /boot/config.txt
```

Commit the following line.

```
#dtoverlay = w1-gpio
```

For C Language Users:

Step 3: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/03_8Led/
```

Step 4: Compile.

```
gcc 8Led.c -o 8Led -lwiringPi
```

Step 5: Run.

```
sudo ./8Led
```

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)

```
//make led_n on
void led_on(int n)
{
    digitalWrite(n, LOW);
}

//make led_n off
void led_off(int n)
{
    digitalWrite(n, HIGH);
}

int main(void)
{
    int i;

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

    for(i=0;i<8;i++){
        printf("linker LedPin : GPIO %d(wiringPi pin)\n",i); //when initialize wiring␣
↪successfully,print message to screen
    }

    for(i=0;i<8;i++){       //make 8 pins' mode is output
        pinMode(i, OUTPUT);
    }

    while(1){
        for(i=0;i<8;i++){   //make led on from left to right
            led_on(i);
            delay(100);
            led_off(i);
        }
    //      delay(500);
        for(i=8;i>=0;i--){  //make led off from right to left
            led_on(i);
            delay(100);
            led_off(i);
        }
    }

    return 0;
}
```

For Python Users:

Step 3: Change directory.

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

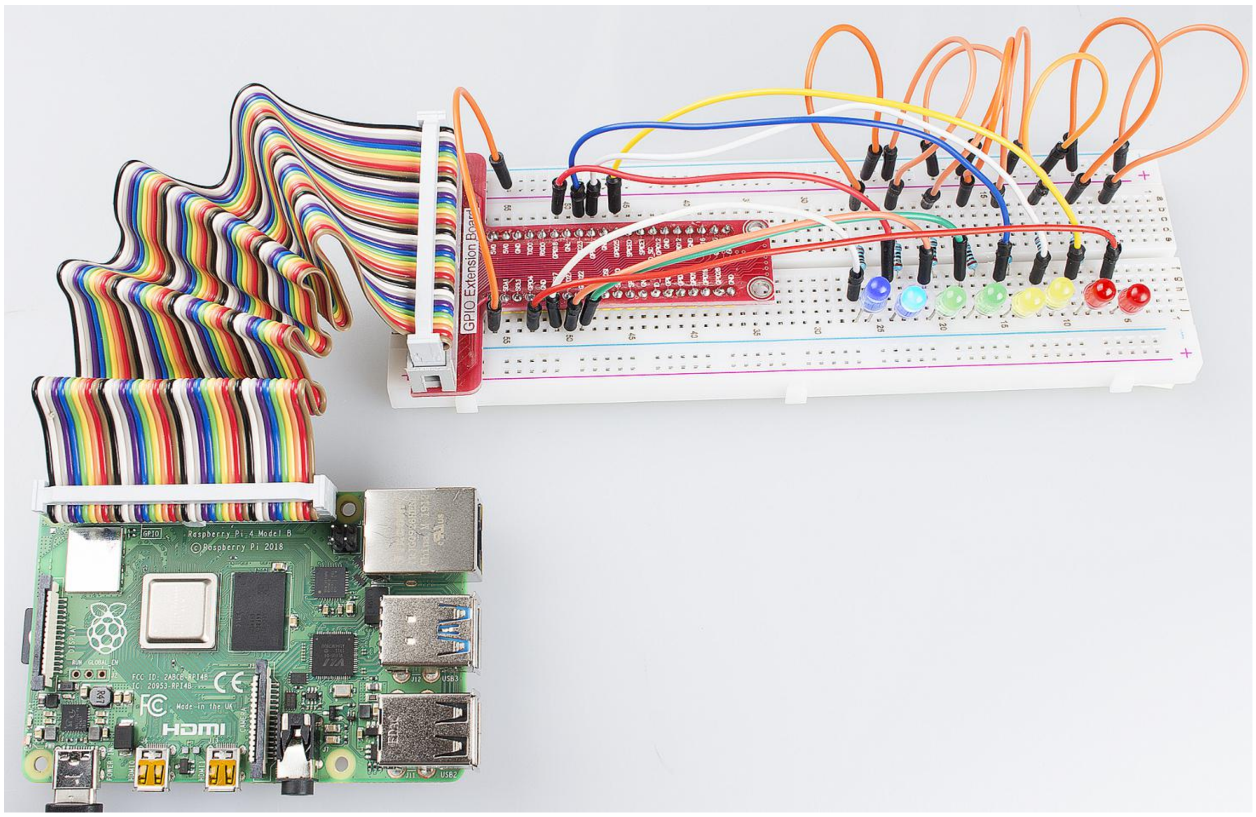
Step 4: Run.

```
sudo python3 03_8Led.py
```

Then you will see eight LEDs brighten and dim left to right and right to left circularly, just like flowing water.

Code

```
import RPi.GPIO as GPIO
import time

pins = [17, 18, 27, 22, 23, 24, 25, 4]

def setup():
    GPIO.setmode(GPIO.BCM)        # Numbers GPIOs by BCM
    for pin in pins:
        GPIO.setup(pin, GPIO.OUT)   # Set all pins' mode is output
        GPIO.output(pin, GPIO.HIGH) # Set all pins to high(+3.3V) to off led

def loop():
    while True:
        for pin in pins:
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.05)
            GPIO.output(pin, GPIO.HIGH)
        for pin in reversed(pins):
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.05)
            GPIO.output(pin, GPIO.HIGH)

def destroy():
    for pin in pins:
        GPIO.output(pin, GPIO.HIGH)    # 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 child program␣
↪destroy() will be  executed.
        destroy()
```



Further Exploration

You can write the blinking effects of LEDs in an array. If you want to use one of these effects, you can call it in the *main()* function directly.

6.4 Lesson 4 Breathing LED

6.4.1 Introduction

In this lesson, we will try something interesting – gradually increase and decrease the luminance of an LED with PWM, just like breathing. So we give it a magical name - Breathing LED.

6.4.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * LED

- 1 * Resistor (220)

- Jumper wires

6.4.3 Principle

PWM

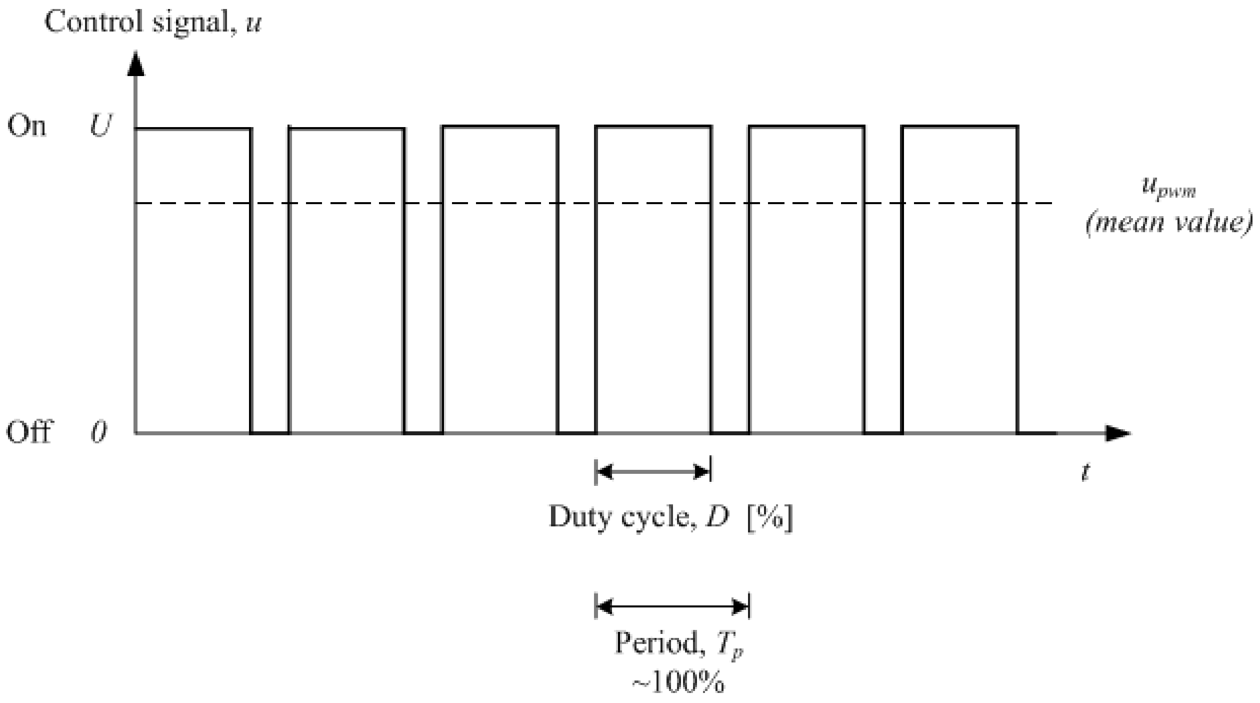
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (3.3 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you change, or modulate, that width. If you repeat this on-off pattern fast enough with some device, an LED for example, the result would be like this: the signal is a steady voltage between 0 and 3.3v controlling the brightness of the LED. (See the PWM description on the official website of Arduino)

Duty Cycle

A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

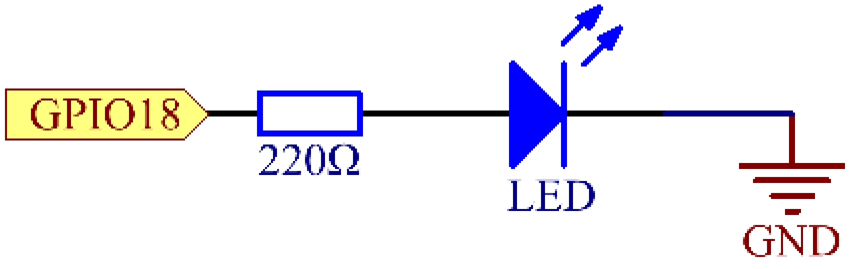
$$D = \frac{\mathrm{T}}{\mathrm{P}} x 100\%$$

where **D** is the duty cycle, **T** is the time the signal is active, and **P** is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.



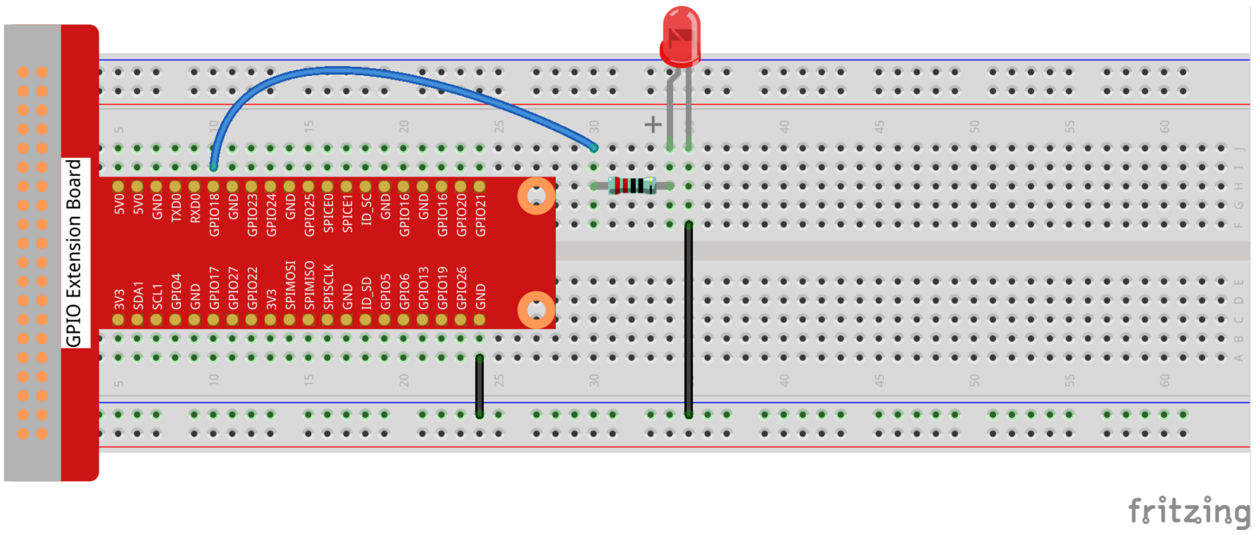
In this experiment, we use this technology to make the LED brighten and dim slowly so it looks like our breath.

6.4.4 Schematic Diagram



6.4.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/04_PwmLed
```

Step 3: Compile.

```
gcc PwmLed.c -o PwmLed -lwiringPi -lpthread
```

Step 4: Run.

```
sudo ./PwmLed
```

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 LedPin    1

int main(void)
{
    int i;

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

    softPwmCreate(LedPin, 0, 100);

    while(1){
        for(i=0;i<=100;i++){
            softPwmWrite(LedPin, i);
            delay(20);
        }
        delay(1000);
        for(i=100;i>=0;i--){
            softPwmWrite(LedPin, i);
            delay(20);
        }
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 04_pwmLed.py
```

Now you will see the gradual change of the LED luminance, between bright and dim.

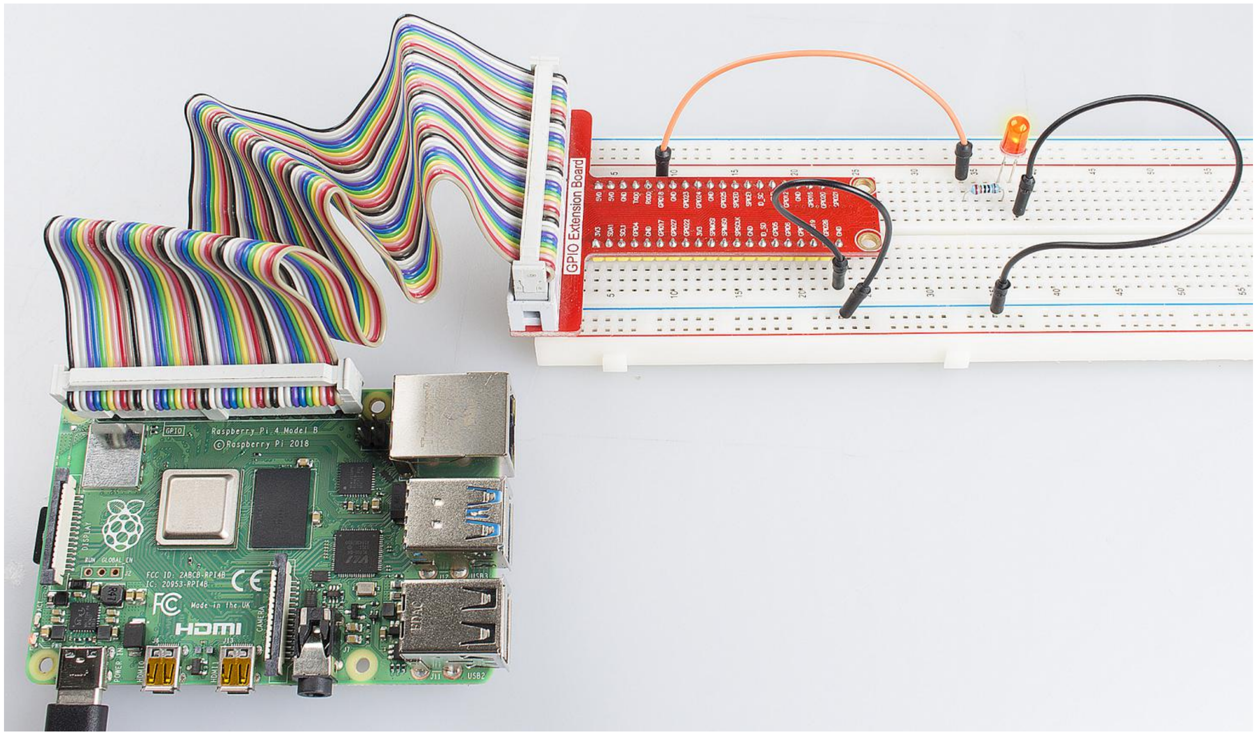
Code

```
import RPi.GPIO as GPIO
import time

LedPin = 18

def setup():
    global p
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
```

(continues on next page)

(continued from previous page)

```
    GPIO.output(LedPin, GPIO.LOW)   # Set LedPin to low(0V)

    p = GPIO.PWM(LedPin, 1000)      # set Frequece to 1KHz
    p.start(0)                      # Duty Cycle = 0

def loop():
    while True:
        for dc in range(0, 101, 4):    # Increase duty cycle: 0~100
            p.ChangeDutyCycle(dc)      # Change duty cycle
            time.sleep(0.05)
        time.sleep(1)
        for dc in range(100, -1, -4): # Decrease duty cycle: 100~0
            p.ChangeDutyCycle(dc)
            time.sleep(0.05)
        time.sleep(1)

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

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



6.4.6 Summary

Through this experiment, you should have mastered the principle of PWM and how to program Raspberry Pi with PWM. You can try to apply this technology to DC motor speed regulation later.

6.5 Lesson 5 RGB LED

6.5.1 Introduction

Previously we've used the PWM technology to control an LED brighten and dim. In this lesson, we will use it to control an RGB LED to flash various kinds of colors.

6.5.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * RGB LED

- 3 * Resistor (220)

- Several jumper wires

6.5.3 Principle

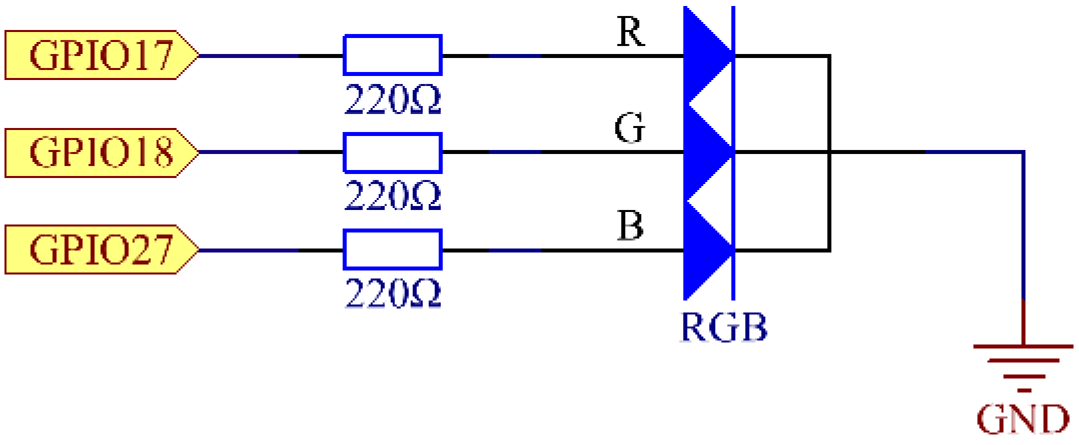
RGB

RGB LEDs emit light in various colors. RGB stands for the red, green, and blue color channels and is an industry color standard. They package three LEDs of red, green, and blue into a transparent or semitransparent plastic shell and have four pins. An RGB LED can display various new colors by changing the three channels and superimposing them, which, according to statistics, can create 16,777,216 different colors.

The three primary colors can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with the hardware PWM of Raspberry Pi. Fortunately, the *softPwm* library simulates PWM (softPwm) by programming. You only need to include the header file *softPwm.h* (for C language users), and then call the API it provides to easily control the RGB LED by multi-channel PWM output, so as to display all kinds of color.

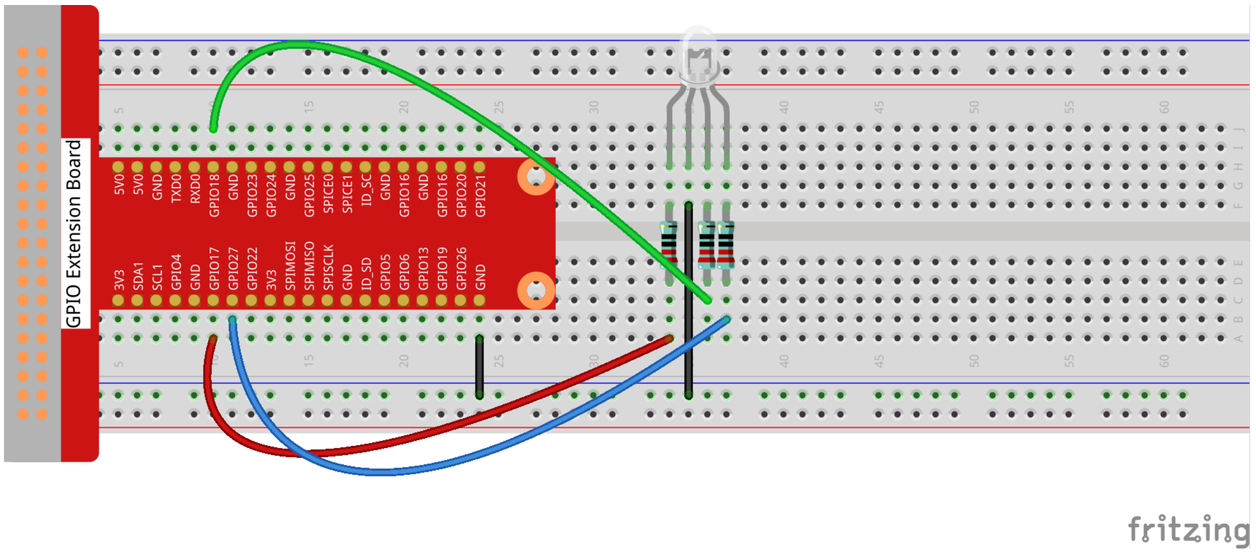
RGB LEDs can be categorized into common anode and common cathode ones. In this experiment, the latter is used.

6.5.4 Schematic Diagram



6.5.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/05_RGB/
```

Step 3: Compile.

```
gcc rgb.c -o rgb -lwiringPi -lpthread
```

Step 4: Run.

```
sudo ./rgb
```

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)
{
    int i;

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    //printf("linker LedPin : GPIO %d(wiringPi pin)\n",LedPin); //when initialize␣
↪wiring successfully,print message to screen

    ledInit();

    while(1){
        ledColorSet(0xff,0x00,0x00);   //red
        delay(500);
        ledColorSet(0x00,0xff,0x00);   //green
        delay(500);
        ledColorSet(0x00,0x00,0xff);   //blue
        delay(500);

        ledColorSet(0xff,0xff,0x00);   //yellow
        delay(500);
        ledColorSet(0xff,0x00,0xff);   //pick
```

(continues on next page)

(continued from previous page)

```
        delay(500);
        ledColorSet(0xc0,0xff,0x3e);
        delay(500);

        ledColorSet(0x94,0x00,0xd3);
        delay(500);
        ledColorSet(0x76,0xee,0x00);
        delay(500);
        ledColorSet(0x00,0xc5,0xcd);
        delay(500);

    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 05_rgb.py
```

Here you should see the RGB LED flash different colors in turn.

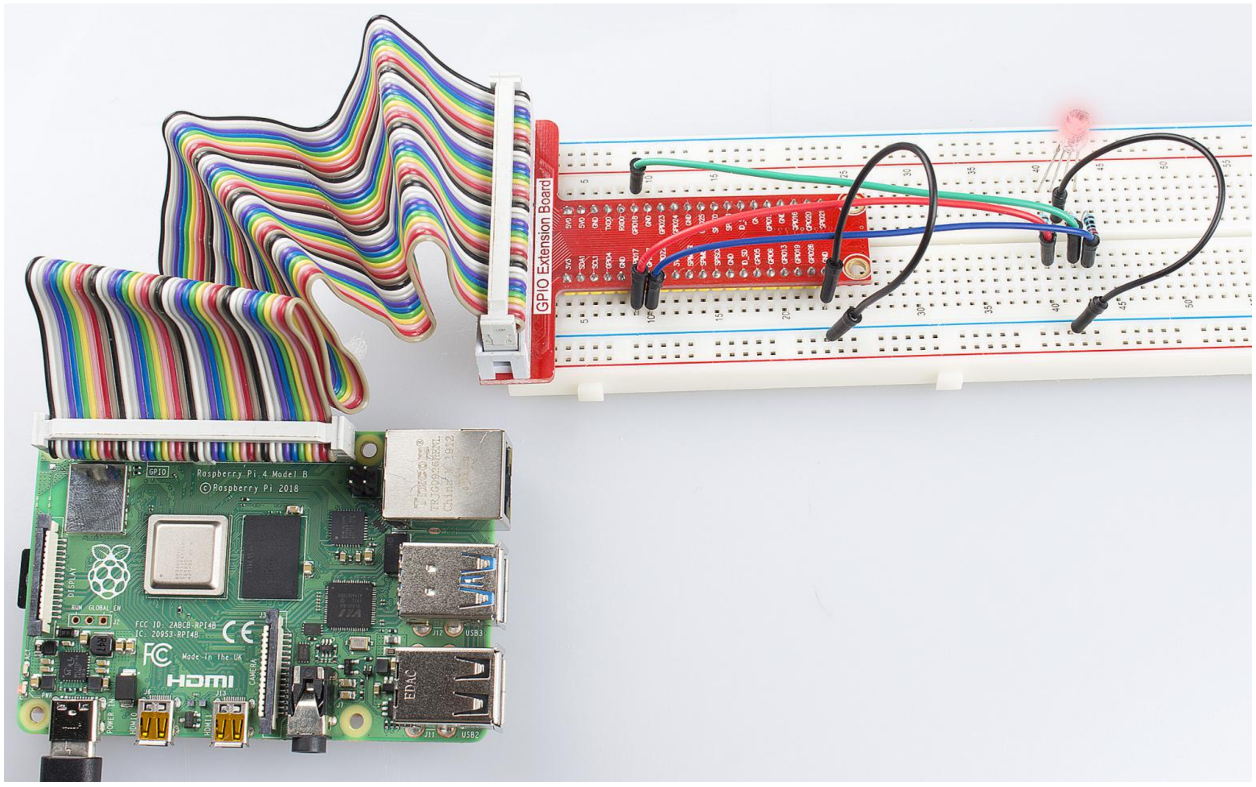
Code

```
import RPi.GPIO as GPIO
import time

colors = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]
pins = {'pin_R':17, 'pin_G':18, 'pin_B':27}  # pins is a dict

GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
for i in pins:
    GPIO.setup(pins[i], GPIO.OUT)   # Set pins' mode is output
    GPIO.output(pins[i], GPIO.HIGH) # Set pins to high(+3.3V) to off led

p_R = GPIO.PWM(pins['pin_R'], 2000)  # set Frequece to 2KHz
p_G = GPIO.PWM(pins['pin_G'], 2000)
p_B = GPIO.PWM(pins['pin_B'], 5000)

p_R.start(0)      # Initial duty Cycle = 0(leds off)
p_G.start(0)
p_B.start(0)

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

def setColor(col):   # For example : col = 0x112233
    R_val = (col & 0xFF0000) >> 16
    G_val = (col & 0x00FF00) >> 8
```

(continues on next page)

(continued from previous page)

```
    B_val = (col & 0x0000FF) >> 0

    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)

    p_R.ChangeDutyCycle(R_val)     # Change duty cycle
    p_G.ChangeDutyCycle(G_val)
    p_B.ChangeDutyCycle(B_val)

try:
    while True:
        for col in colors:
            setColor(col)
            time.sleep(0.5)
except KeyboardInterrupt:
    p_R.stop()
    p_G.stop()
    p_B.stop()
    for i in pins:
        GPIO.output(pins[i], GPIO.HIGH)    # Turn off all leds
    GPIO.cleanup()
```



6.5.6 Further Exploration

You can modify the parameters of the function *ledColorSet( )* by yourself, and then compile and run the code to see the color changes of the RGB LED.

6.5.7 Experimental Summary

In this experiment, you have learnt how to control RGB LEDs with the softPwm of Raspberry Pi in this experiment. Try to apply the softPwm to DC motor speed regulation.

6.6 Lesson 6 Buzzer

6.6.1 Introduction

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

6.6.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * Buzzer (Active)

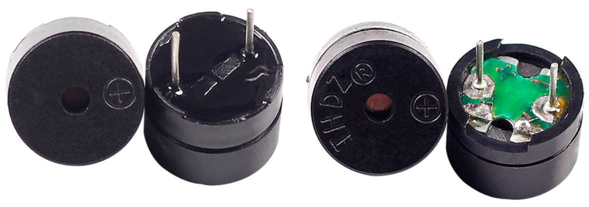
- 1 * PNP transistor (8550)

- 1 * Resistor (1K)

- Jumper wires

6.6.3 Principle

As a type of electronic buzzer with 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 for voice devices. Buzzers can be categorized as active and passive ones (see the following picture). Turn the pins of two buzzers face up, and the one with a green circuit board is a passive buzzer, while the other enclosed with a black tape is an active one.

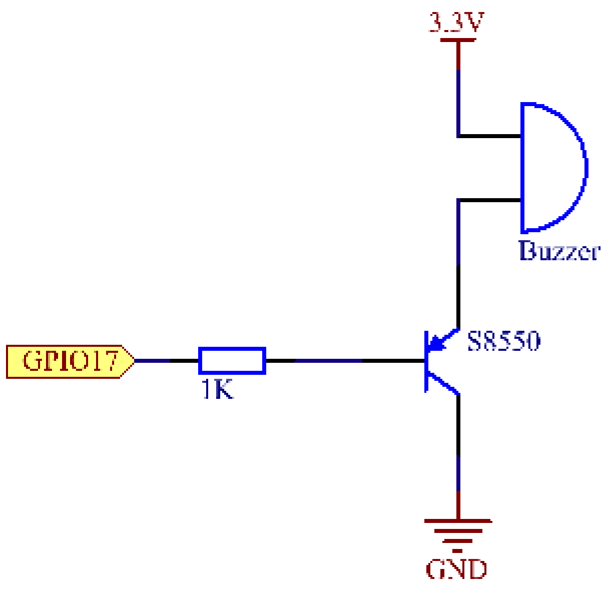


The difference between an active buzzer and a passive buzzer is:

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.

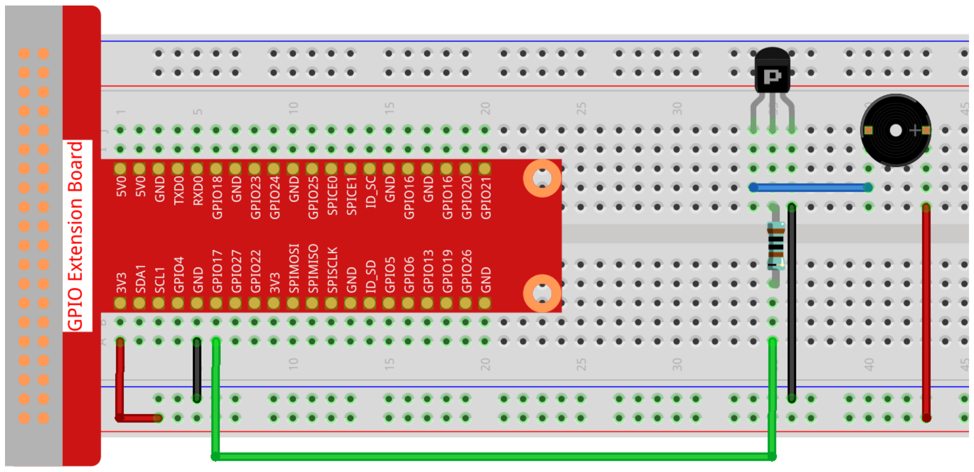
6.6.4 Schematic Diagram

In this experiment, an active buzzer is used. When the GPIO 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.



6.6.5 Experimental Procedures

Step 1: Build the circuit (Pay attention to the positive and negative poles of the buzzer)



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/06_Beep/
```

Step 3: Compile.

```
gcc beep.c -o beep -lwiringPi
```

Step 4: Run.

```
sudo ./beep
```

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){
        digitalWrite(BeepPin, LOW);  //beep on
        delay(100);                  //delay
        digitalWrite(BeepPin, HIGH); //beep off
        delay(100);                  //delay
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

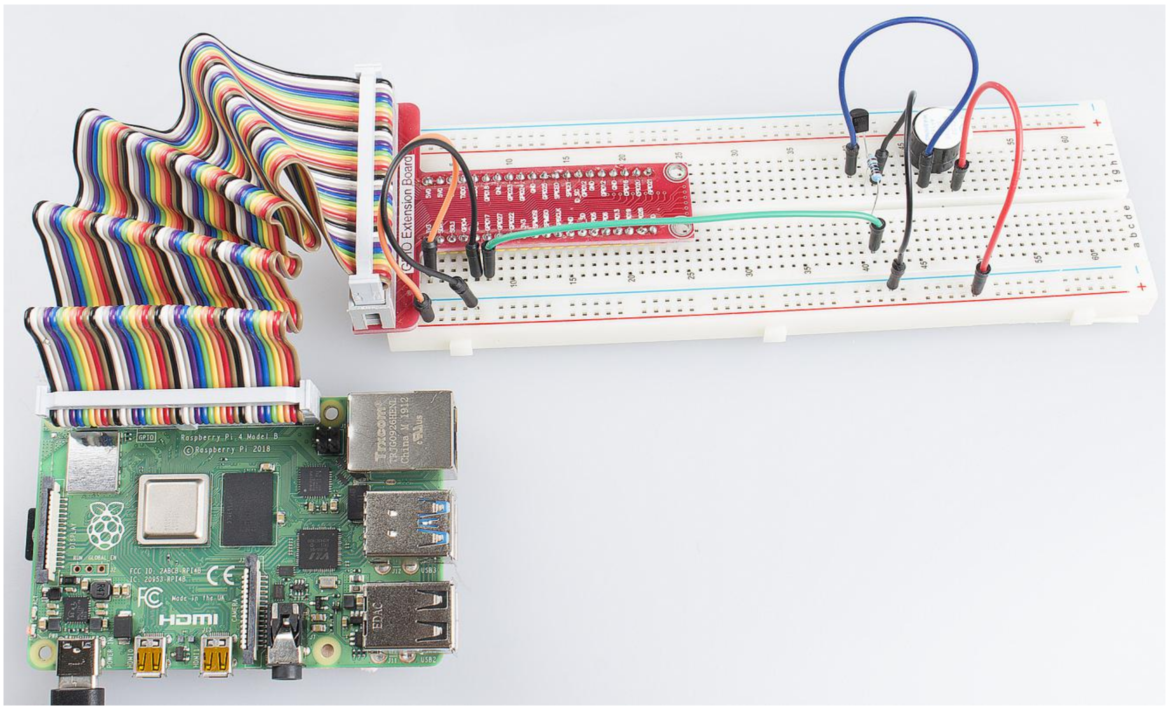
Step 3: Run.

```
sudo python3 06_beep.py
```

Now, you should hear the buzzer make sounds.

Code

```
import RPi.GPIO as GPIO
import time

BeepPin = 17

def setup():
    GPIO.setmode(GPIO.BCM)         # Numbers GPIOs by BCM
    GPIO.setup(BeepPin, GPIO.OUT)    # Set BeepPin's mode is output
    GPIO.output(BeepPin, GPIO.HIGH) # Set BeepPin high(+3.3V) to off beep

def loop():
    while True:
        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)     # beep off
    GPIO.cleanup()                      # Release resource

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



6.6.6 Further Exploration

If you have a passive buzzer in hand, you can replace the active buzzer with it. Now you can make a buzzer sound like "do re mi fa so la si do" with just some basic knowledge of programming. Give a try!

6.7 Lesson 7 How to Drive a DC Motor

6.7.1 Introduction

In this lesson, we will learn to how to use L293D to drive a DC motor and make it rotate clockwise and counterclockwise.

6.7.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * L293D

- 1 * DC motor

- 1 * Power Module

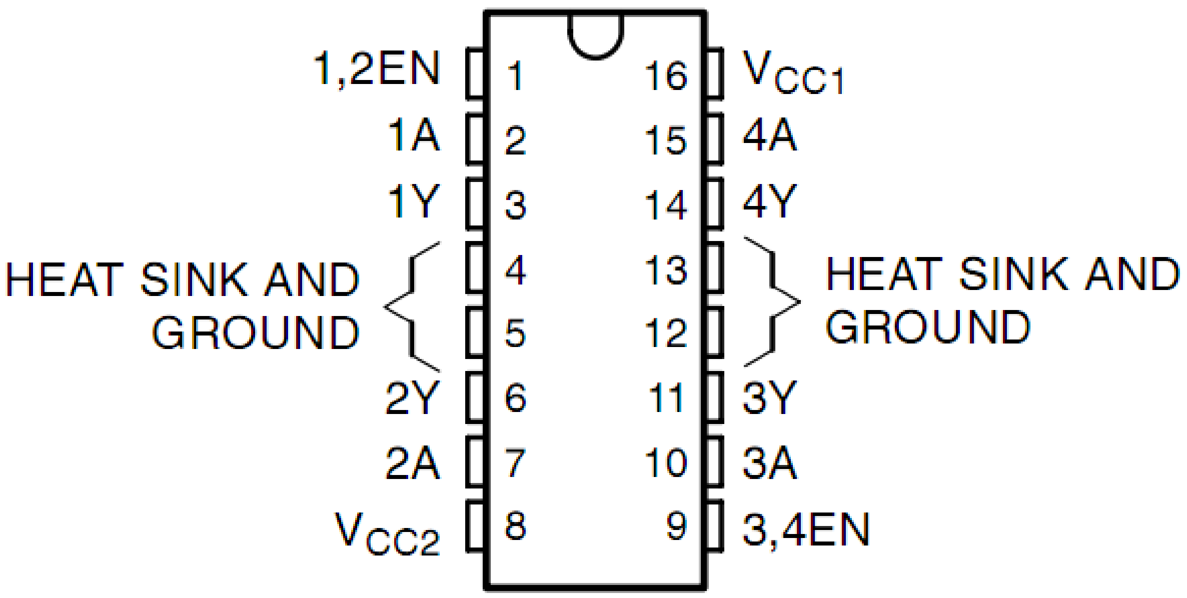
- Jumper wires

6.7.3 Principle

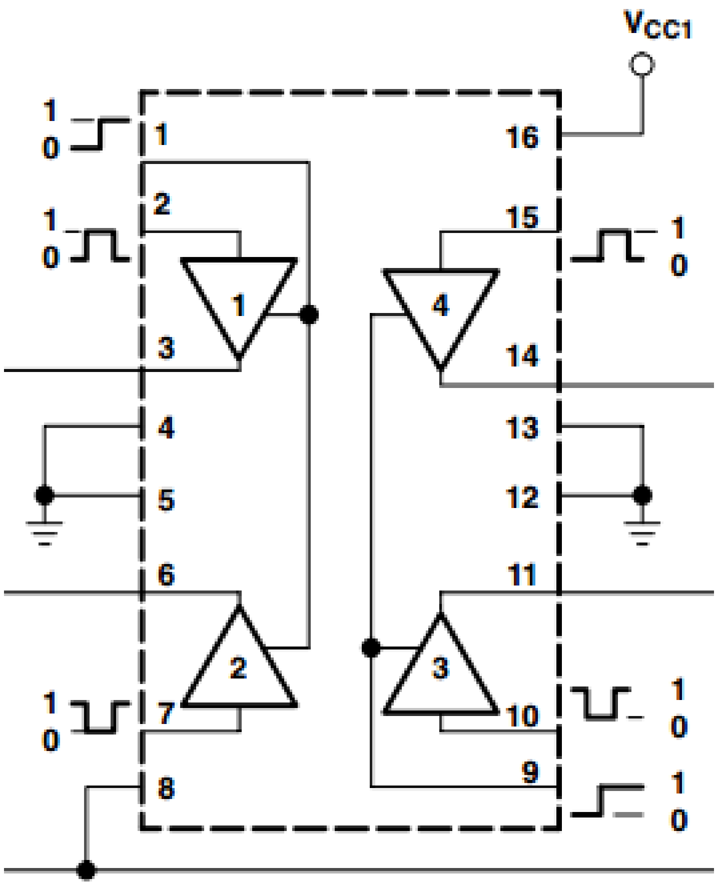
L293D

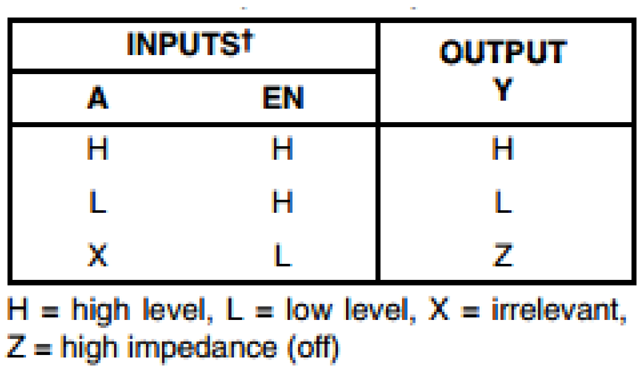
L293D is a 4-channel motor driver integrated by chip with high voltage and high current. It's designed to connect to standard DTL, TTL logic level, and drive inductive loads (such as relay coils, DC, Stepper Motors) and power switching transistors etc. DC Motors are devices that turn DC electrical energy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



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)

DC Motor

This is a 5V DC motor. It will rotate when you give the two terminals of the copper sheet one high and one low level. For convenience, you can weld the pins to it.

Size: 25*20*15MM

Operation Voltage: 1-6V

Free-run current (3V): 70mA

Free-run speed (3V): 13000RPM

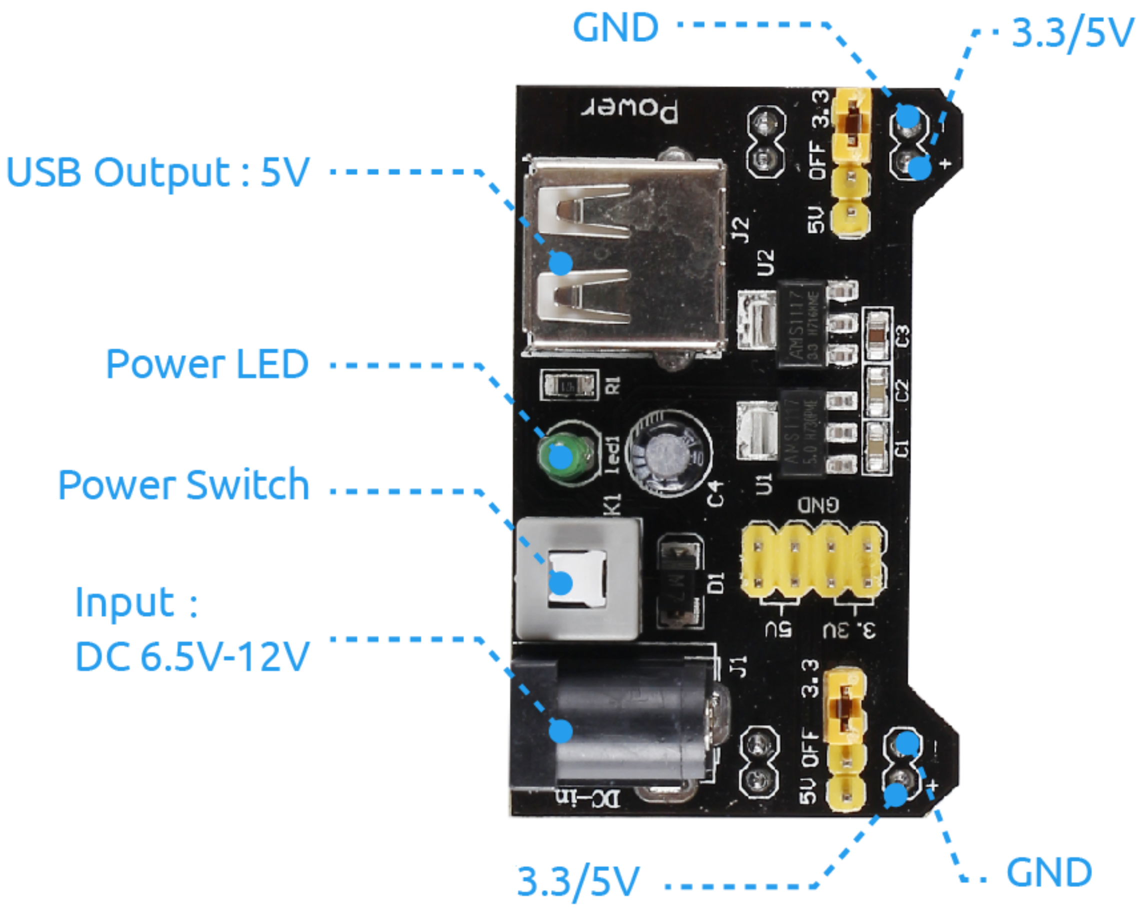
Stall current (3V): 800mA

Shaft diameter: 2mm

Power Supply Module

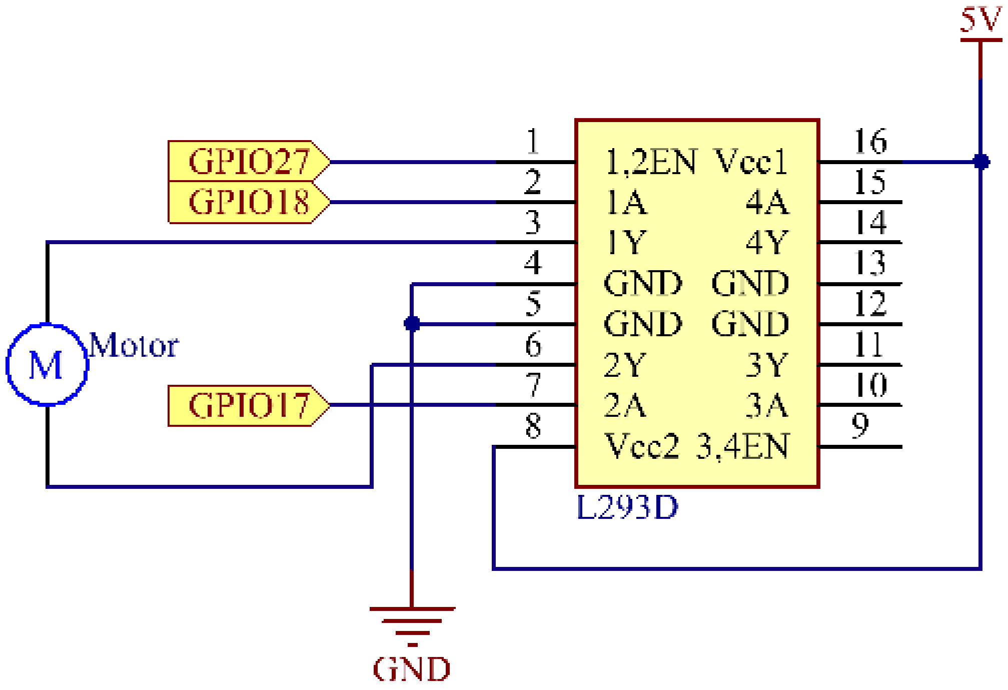
In this experiment, it needs large currents to drive the motor especially when it starts and stops, which will severely interfere with the normal work of Raspberry Pi. Therefore, we separately supply power for the motor by this module to make it run safely and steadily.

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.



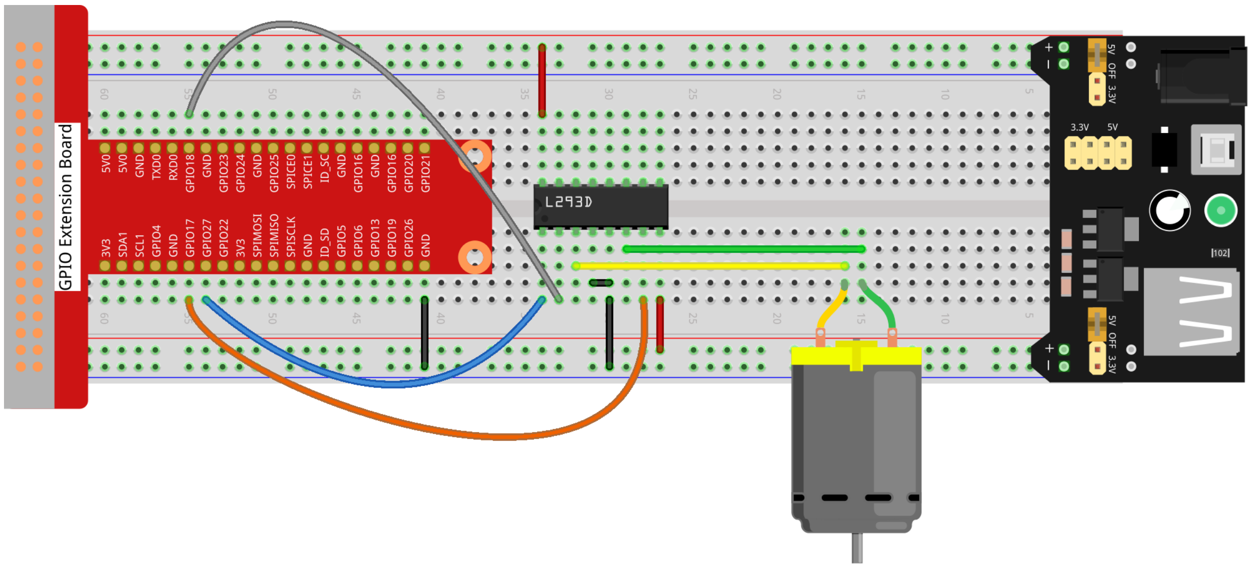
Schematic Diagram

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.



6.7.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/07_Motor/
```

Step 3: Compile.

```
gcc motor.c -o motor -lwiringPi
```

Step 4: Run.

```
sudo ./motor
```

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    1
#define MotorEnable  2

int main(void)
{
    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);

    int i;

    while(1){
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, HIGH);
        digitalWrite(MotorPin2, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        digitalWrite(MotorEnable, LOW);
            delay(1000);

        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, LOW);
        digitalWrite(MotorPin2, HIGH);
        for(i=0;i<3;i++){
            delay(1000);
        }
```

(continues on next page)

(continued from previous page)

```
        digitalWrite(MotorEnable, LOW);
                        delay(1000);

    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 07_motor.py
```

Now, you should see the motor blade rotating.

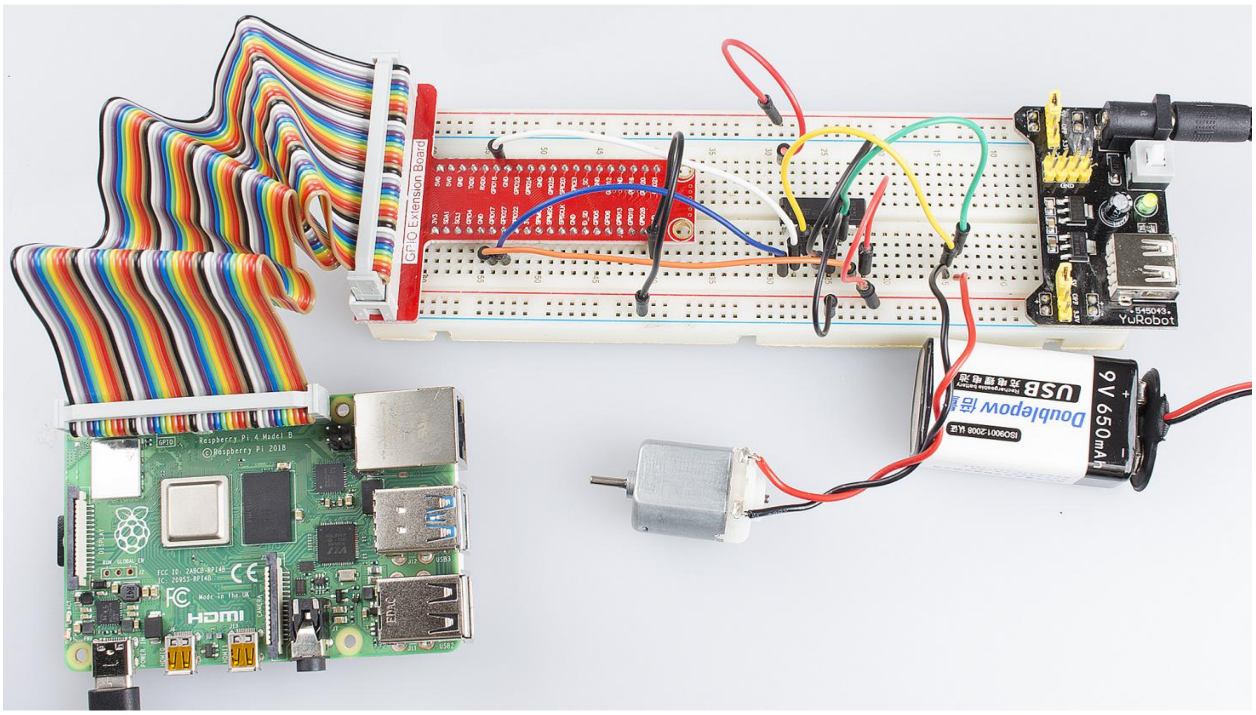
Code

```
import RPi.GPIO as GPIO
import time

MotorPin1   = 17
MotorPin2   = 18
MotorEnable = 27

def setup():
    GPIO.setmode(GPIO.BCM)          # Numbers GPIOs by BCM
    GPIO.setup(MotorPin1, GPIO.OUT)   # mode --- output
    GPIO.setup(MotorPin2, GPIO.OUT)
    GPIO.setup(MotorEnable, GPIO.OUT)
    GPIO.output(MotorEnable, GPIO.LOW) # motor stop

def loop():
    while True:
        print ("Press Ctrl+C to end the program...")
        GPIO.output(MotorEnable, GPIO.HIGH) # motor driver enable
        GPIO.output(MotorPin1, GPIO.HIGH)  # clockwise
        GPIO.output(MotorPin2, GPIO.LOW)
        time.sleep(5)

        GPIO.output(MotorEnable, GPIO.LOW) # motor stop
        time.sleep(5)

        GPIO.output(MotorEnable, GPIO.HIGH) # motor driver enable
        GPIO.output(MotorPin1, GPIO.LOW)   # anticlockwise
        GPIO.output(MotorPin2, GPIO.HIGH)
        time.sleep(5)

        GPIO.output(MotorEnable, GPIO.LOW) # motor stop
        time.sleep(5)
```

(continues on next page)

(continued from previous page)

```
def destroy():
    GPIO.output(MotorEnable, GPIO.LOW) # motor 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()
```



6.7.5 Further Exploration

You can use buttons to control the clockwise and counterclockwise rotation of the motor blade based on the previous lessons. Also you can apply the PWM technology to control the rotation.

6.7.6 Summary

Through this lesson, you have learnt the relative principle and driving mode of DC motors, as well as how to drive a motor by Raspberry Pi. You should also pay special attention to the fact that a DC motor will greatly interfere with the whole circuit when it works, so you need to adopt photoelectric isolation and provide separate power supply. A freewheeling diode is also necessary for the whole system to work reliably and steadily.

6.8 Lesson 8 Rotary Encoder

6.8.1 Introduction

A rotary encoder is a type of electro-mechanical device that converts the angular position or motion of a shaft or axle to an analog or digital code. In this lesson, we will learn how to use this device.

6.8.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

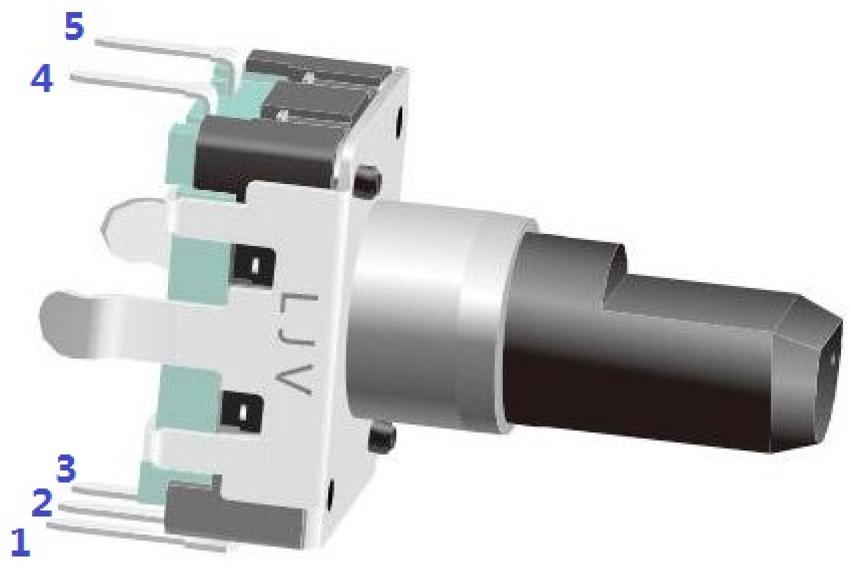
- 1 * Rotary Encoder Module

- Jumper wires

6.8.3 Principle

A rotary encoder is an electronic switch with a set of regular pulses with strictly timing sequence. When used with IC, it can achieve increment, decrement, page turning and other operations such as mouse scrolling, menu selection, acoustic sound regulation, frequency regulation, toaster temperature regulation, and so on.

There are mainly two types of rotary encoders: absolute and incremental (relative) encoders. The output of absolute encoders indicates the current position of the shaft, making them angle transducers. The output of incremental encoders provides information about the motion of the shaft, which is typically further processed elsewhere into information such as speed, distance, and position.



Most rotary encoders have 5 pins with three functions of turning left, turning right and pressing down:

Pin 4 & 5: switching wiring terminals for pressing down (no different from the buttons mentioned previously, so no more details will be provided here.)

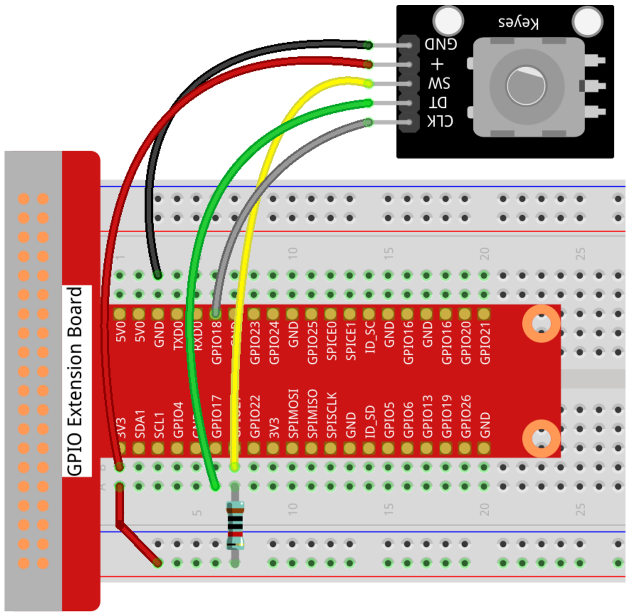
Pin 2: generally connected to ground.

Pin 1 & 3: first connected to a pull-up resistor and then to a microprocessor (in this experiment, to GPIO0 and GPIO1 of Raspberry Pi); when you spin the knob of the encoder clockwise and counterclockwise, there will be pulse outputs in pin 1 and pin 3.

If both GPIO0 and GPIO1 are at high level, the switch rotates clockwise; if GPIO0 is at high level but GPIO1 is low, the switch rotates counterclockwise. Therefore, when programming, you only need to check the state of pin 3 when pin 1 is at high level, and then you can tell whether the switch rotates clockwise or counterclockwise.

Step 1: Build the circuit.

Raspberry Pi	Rotary Encoder
3.3V	+
GND	GND
GPIO17	DT
GPIO18	CLK
GPIO27	SW



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/08_RotaryEncoder/
```

Step 3: Compile.

```
gcc rotaryEncoder.c -o rotaryEncoder -lwiringPi
```

Step 4: Run.

```
sudo ./rotaryEncoder
```

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  RoAPin    0
#define  RoBPin    1
#define  RoSPin    2

static volatile int globalCounter = 0 ;

unsigned char flag;
unsigned char Last_RoB_Status;
unsigned char Current_RoB_Status;

void rotaryDeal(void)
{
    Last_RoB_Status = digitalRead(RoBPin);

    while(!digitalRead(RoAPin)){
        Current_RoB_Status = digitalRead(RoBPin);
        flag = 1;
    }

    if(flag == 1){
        flag = 0;
        if((Last_RoB_Status == 0)&&(Current_RoB_Status == 1)){
            globalCounter ++;
            printf("globalCounter : %d\n",globalCounter);
        }
        if((Last_RoB_Status == 1)&&(Current_RoB_Status == 0)){
            globalCounter --;
            printf("globalCounter : %d\n",globalCounter);
        }

    }
}

void rotaryClear(void)
{
    if(digitalRead(RoSPin) == 0)
    {
        globalCounter = 0;
        printf("globalCounter : %d\n",globalCounter);
        delay(1000);
    }
}

int main(void)
{
    if(wiringPiSetup() < 0){
        fprintf(stderr, "Unable to setup wiringPi:%s\n",strerror(errno));
```

(continues on next page)

(continued from previous page)

```
        return 1;
    }

    pinMode(RoAPin, INPUT);
    pinMode(RoBPin, INPUT);
    pinMode(RoSPin, INPUT);

    pullUpDnControl(RoSPin, PUD_UP);

    while(1){
        rotaryDeal();
        rotaryClear();
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 08_rotaryEncoder.py
```

Now, gently rotate the encoder to change the value of the variable in the above program, and you will see the value printed on the screen. Rotate the encoder clockwise, the value will increase; or rotate it counterclockwise, the value will decrease.

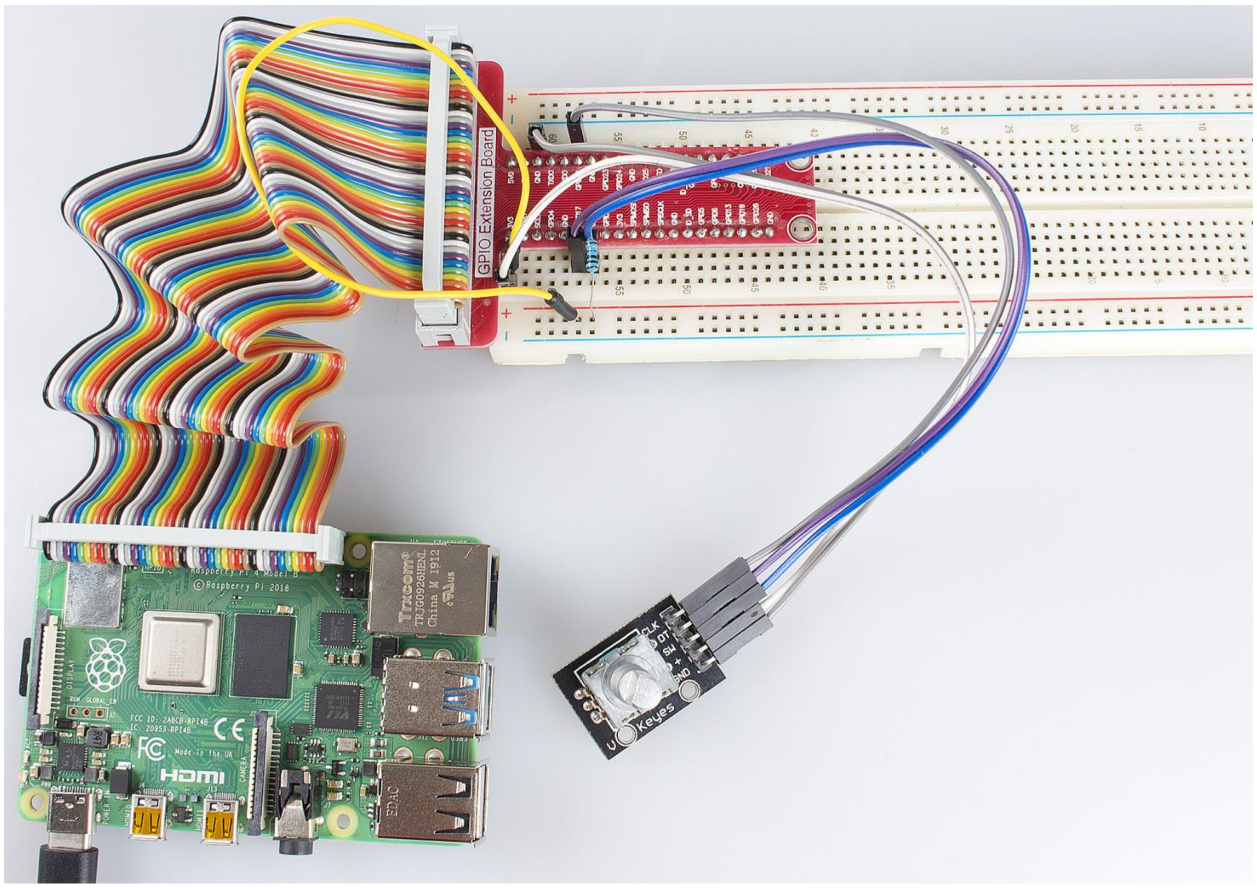
Code

```
import RPi.GPIO as GPIO
import time

# Set up pins
# Rotary A Pin
RoAPin = 17
# Rotary B Pin
RoBPin = 18
# Rotary Switch Pin
RoSPin = 27

def setup():
    global counter
    global Last_RoB_Status, Current_RoB_Status
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(RoAPin, GPIO.IN)
    GPIO.setup(RoBPin, GPIO.IN)
    GPIO.setup(RoSPin,GPIO.IN, pull_up_down=GPIO.PUD_UP)
    # Set up a falling edge detect to callback clear
    GPIO.add_event_detect(RoSPin, GPIO.FALLING, callback=clear)

    # Set up a counter as a global variable
```

(continues on next page)

(continued from previous page)

```
    counter = 0
    Last_RoB_Status = 0
    Current_RoB_Status = 0

# Define a function to deal with rotary encoder
def rotaryDeal():
    global counter
    global Last_RoB_Status, Current_RoB_Status

    flag = 0
    Last_RoB_Status = GPIO.input(RoBPin)
    # When RoAPin level changes
    while(not GPIO.input(RoAPin)):
        Current_RoB_Status = GPIO.input(RoBPin)
        flag = 1
    if flag == 1:
        # Reset flag
        flag = 0
        if (Last_RoB_Status == 0) and (Current_RoB_Status == 1):
            counter = counter + 1
        if (Last_RoB_Status == 1) and (Current_RoB_Status == 0):
            counter = counter - 1
        print ("counter = %d" % counter)

# Define a callback function on switch, to clean "counter"
def clear(ev=None):
    global counter
    counter = 0
    print ("counter = %d" % counter)

def main():
    while True:
        rotaryDeal()

def destroy():
    # 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()
```



6.8.4 Further Exploration

In this experiment, the pressing down function of rotary encoder is not involved. Try to explore this function by yourself!

6.9 Lesson 9 555 Timer

6.9.1 Introduction

The NE555 Timer, a mixed circuit composed of analog and digital circuits, integrates analog and logical functions into an independent IC, thus tremendously expanding the applications of analog integrated circuits. It is widely used in various timers, pulse generators, and oscillators. In this lesson, we will learn how to use the 555 timer.

6.9.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * NE555

- 3 * Resistor (1 * 1K, 2 * 10K)

- 2 * Capacitor (100nF)

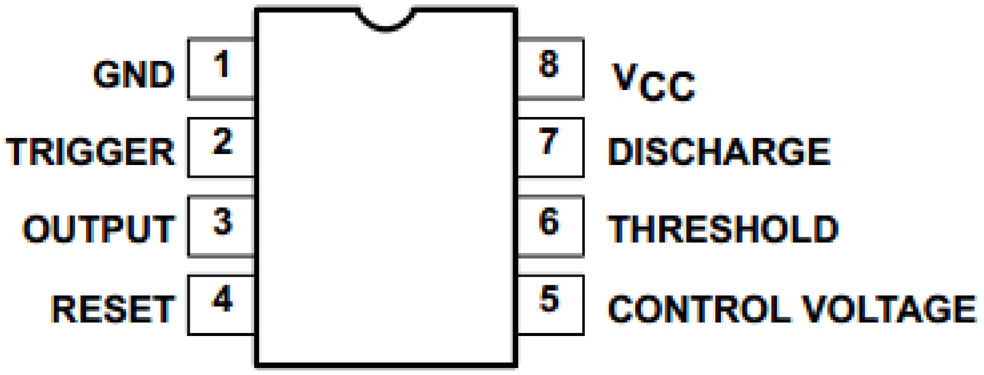
- Jumper wires

6.9.3 Principle

555 IC

A 555 timer is a medium-sized IC device which combines analog and digital functions. With low cost and reliable performance, it just attaches external resistors and capacitors so as to achieve the functions of multivibrator, monostable trigger, Schmitt trigger and other circuits which can generate and transform pulses. It is also used frequently as a timer and widely applied to instruments, household appliances, electronic measurement, automatic control and other fields.

Its pins and their functions:



As shown in the picture, the pins are set dual in-line with the 8-pin package.

- Pin 1 (**GND**): the ground

- Pin 2 (**TRIGGER**): the input of lower comparator

- Pin 3 (**OUTPUT**): having two states of 0 and 1 decided by the input electrical level

- Pin 4 (**RESET**): outputting low level when supplied a low one

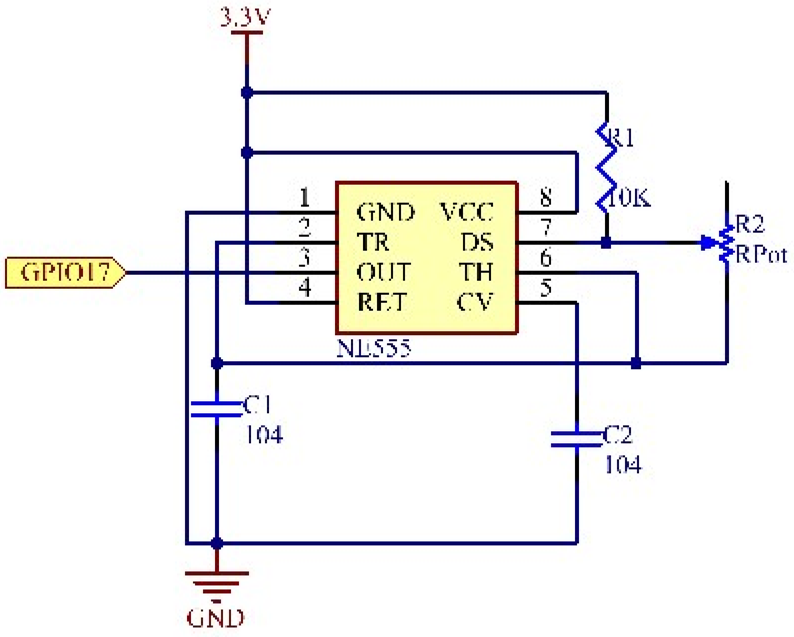
- Pin 5 (**CONTROL VOLTAGE**): changing the upper and lower level trigger values

- Pin 6 (**THRESHOLD**): the input of the upper comparator

- Pin 7 (**DISCHARGE**): having two states of suspension and ground connection also decided by the input, and the output of the internal discharge tube

- Pin 8 (**VCC**): the power supply

The 555 timer can work under three modes. In this experiment, the astable mode is used to generate square waves.



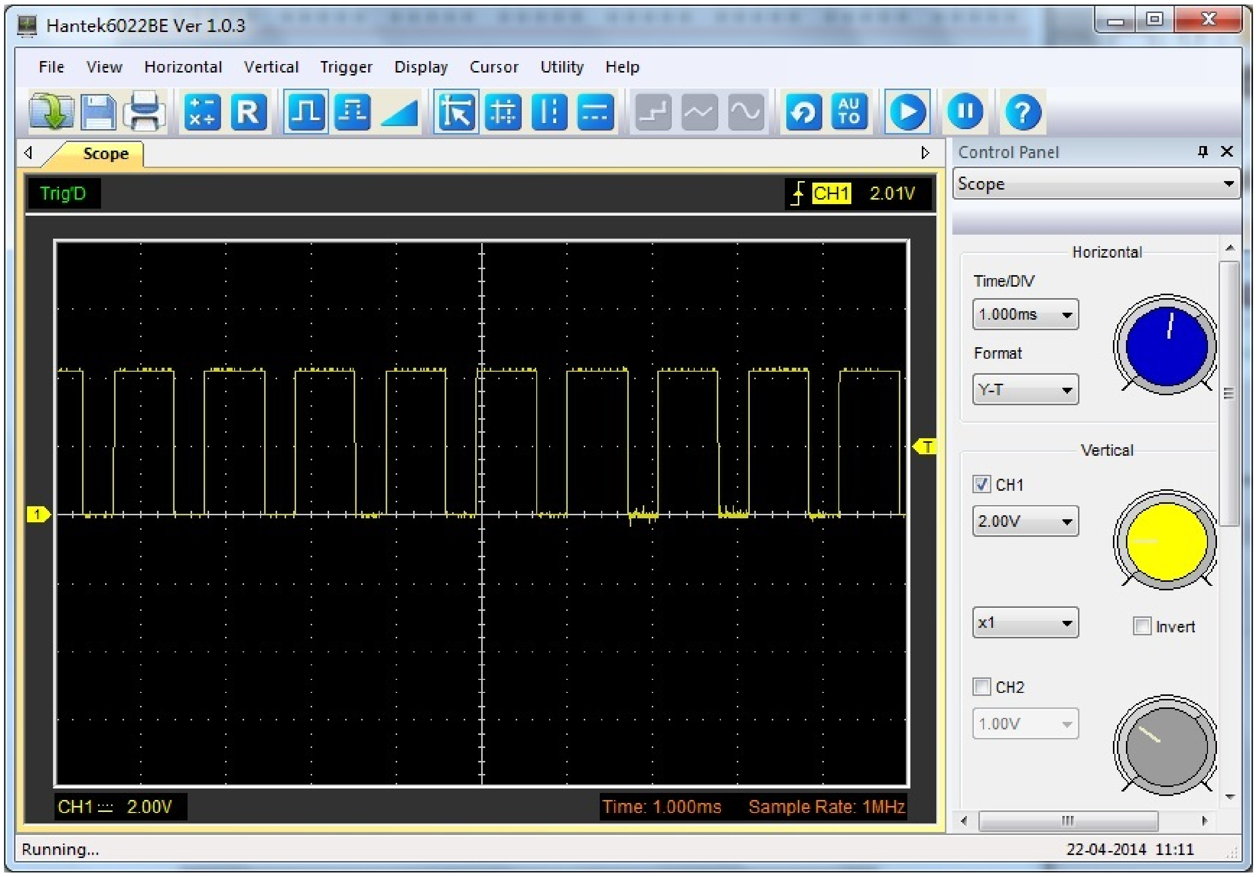
Under the astable mode, the frequency of output waveform of the 555 timer is determined by R_1, R_2 and C_2 :

$$f = \frac{1}{ln2 \, * \, C_2 \, * \, (R_1 \, + \, 2R_2)}$$

In the above circuit, R_1= R_2=10K= 10^4; =100nF=10^{-7}F, so we can get the frequency:

$$f = \frac{1}{ln2 * \, 10^{-7} * (10^4 + \, 2 * 10^4)} \approx 481Hz$$

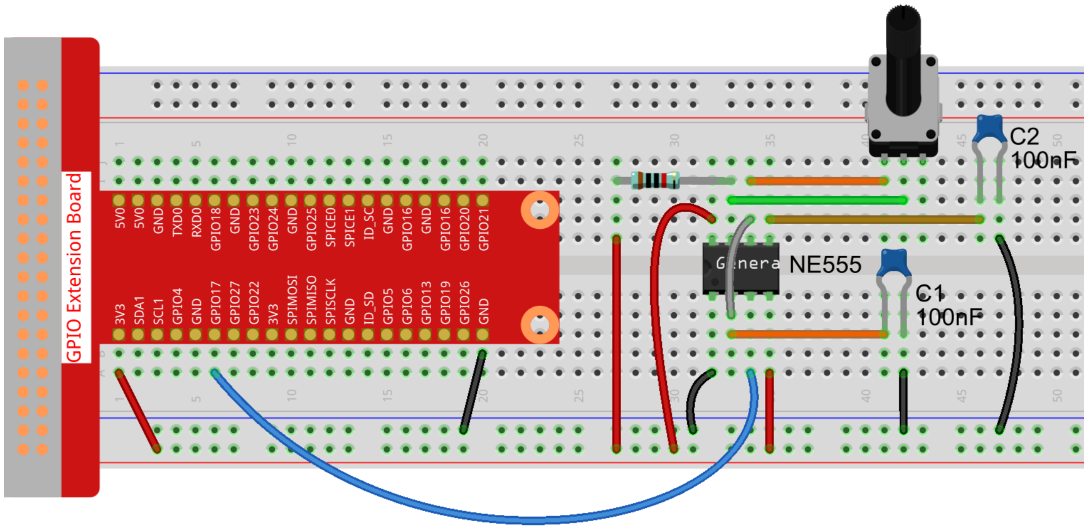
After connecting the circuit according to the schematic diagram, use an oscilloscope to observe the frequency of the output waveform. We can see it is consistent with the above calculated result.



Attach the output pin (e.g. pin 3) of the 555 timer to GPIO17 of the Raspberry Pi, configure GPIO17 as the mode of the rising edge interrupt by programming, and then detect the square wave pulses generated by the 555 timer with interrupt. The work of Interrupt Service Routine (ISR) is to add 1 to a variable.

6.9.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/09_Timer555/
```

Step 3: Compile.

```
gcc timer555.c -o timer555 -lwiringPi
```

Step 4: Run.

```
sudo ./timer555
```

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  Pin0  0

static volatile int globalCounter = 0 ;

void exInt0_ISR(void)  //GPIO0 interrupt service routine
{
    ++globalCounter;
}

int main (void)
{
  if(wiringPiSetup() < 0){
      fprintf(stderr, "Unable to setup wiringPi:%s\n",strerror(errno));
    return 1;
  }

  wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR);

   while(1){
    printf("Current pluse number is : %d\n", globalCounter);
    delay(100);
  }

  return 0;
}
```

For Python Users:

Step 2: Change directory.

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

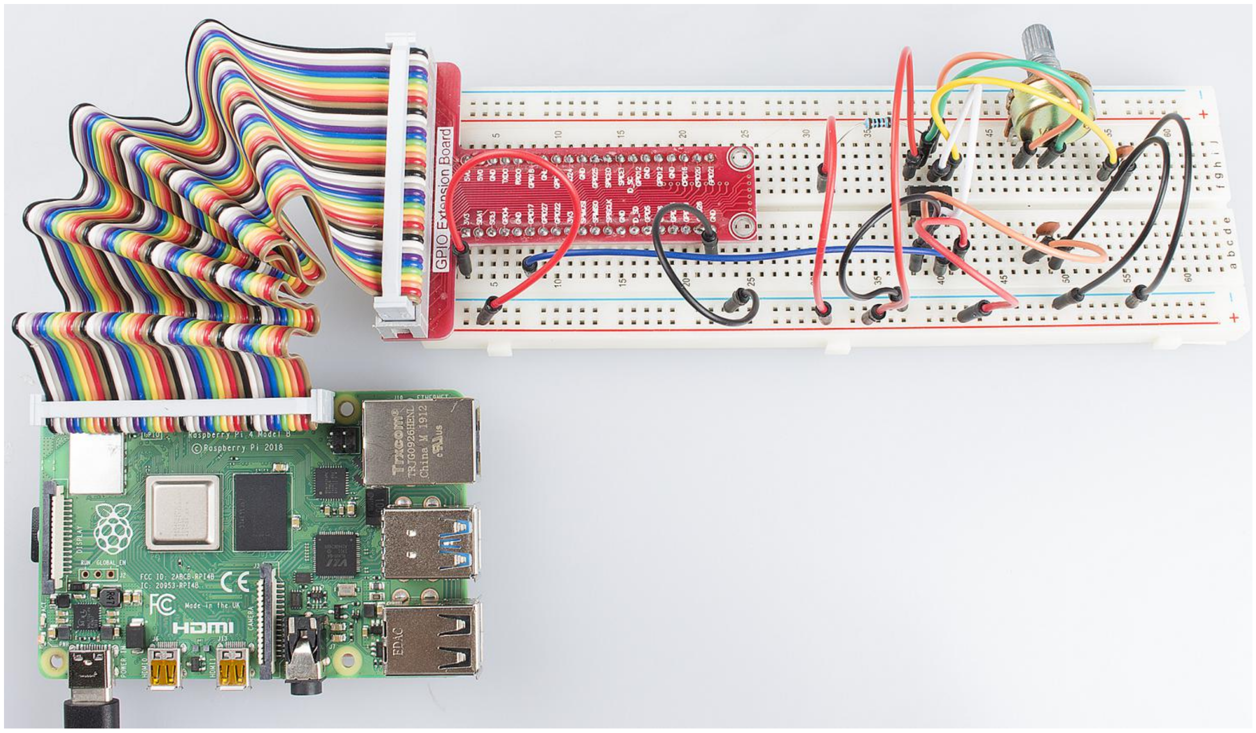
Step 3: Run.

```
sudo python3 09_timer555.py
```

Now, you should see data printed on the display, which are square waves generated by the 555 timer. The program counts pulses by interrupt as we have learned previously.

Code

```
import RPi.GPIO as GPIO
import time

SigPin = 17

g_count = 0

def count(ev=None):
    global g_count
    g_count += 1

def setup():
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
    GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set Pin's mode is␣
↪input, and pull up to high level(3.3V)
    GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # wait for rasing

def loop():
    while True:
        print ("g_count = %d" % g_count)
        time.sleep(0.2)

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()
```



6.10 Lesson 10 Driving LEDs by 74HC595

6.10.1 Introduction

In this lesson, we will learn how to use 74HC595 to make eight LEDs blink regularly.

6.10.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * 74HC595

- 8 * LED

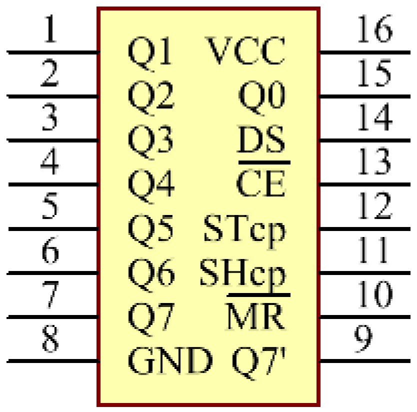
- 8 * Resistor (220)

- Jumper wires

6.10.3 Principle

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 that you can save IO ports of an MCU. The 74HC595 is widely used to indicate multipath LEDs and drive multi-bit segment displays. “Three-state” mentioned above refers to the fact that you can set the output pins as either high, low or high impedance. With data latching, the instant output will not be affected during the shifting; with data output, you can cascade 74HC595s more easily. Compatible with low voltage TTL circuit, 74HC595 can transform serial input of 8-bit data into parallel output of 8-bit data. So it is often used to extend GPIO for embedded system and drive low power devices.



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; here it is directly connected to 5V.

SH_CP: 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.

ST_CP: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

OE: Output enable pin, active at low level, connected to GND.

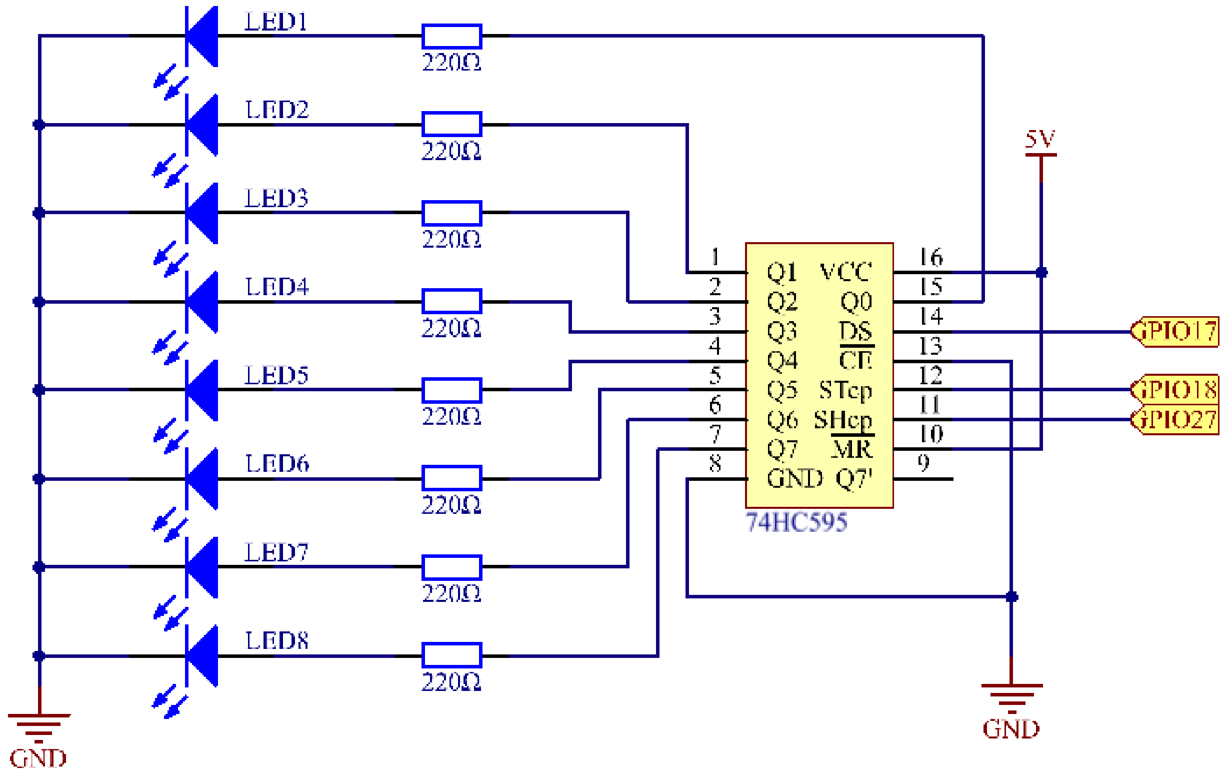
DS: Serial data input pin

VCC: Positive supply voltage

GND: Ground

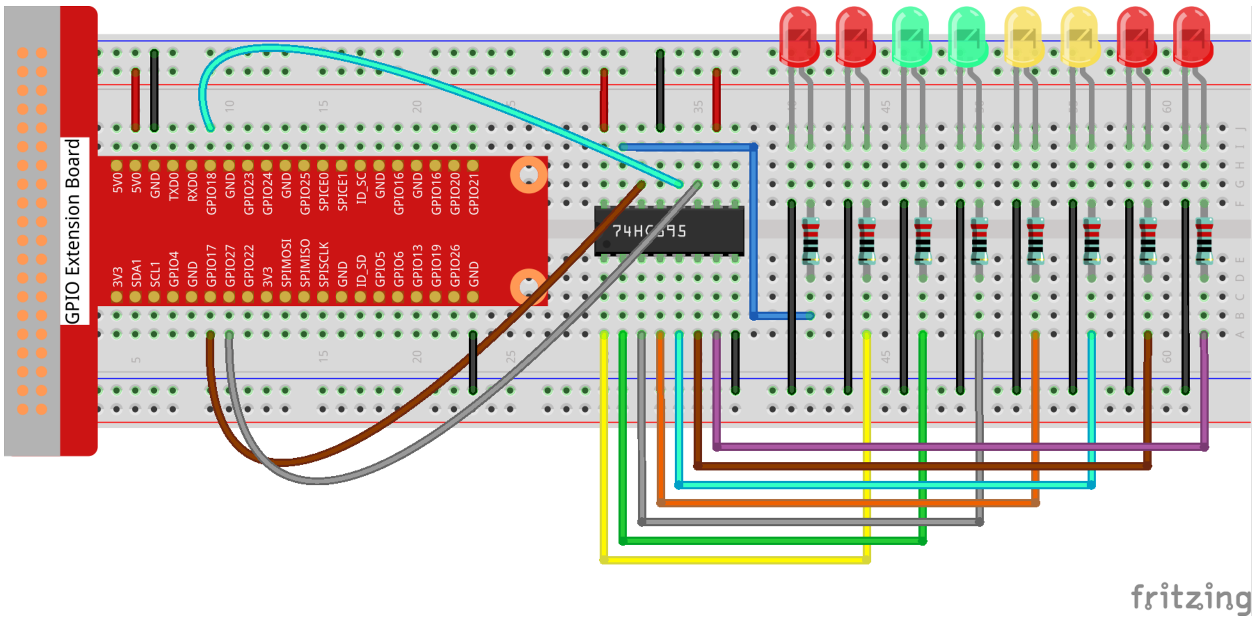
6.10.4 Schematic Diagram

In this experiment, connect ST_CP to Raspberry Pi GPIO18, SH_CP to GPIO27, and DS to GPIO17. Input data in DS pin to the 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 Raspberry Pi GPIO to transform serial input data into parallel output data so as to save Raspberry Pi GPIOs.



6.10.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/10_74HC595_LED/
```

Step 3: Compile.

```
gcc 74HC595_LED.c -o 74HC595_LED -lwiringPi
```

Step 4: Run.

```
sudo ./74HC595_LED
```

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 LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};


void pulse(int pin)
{
    digitalWrite(pin, 0);
    digitalWrite(pin, 1);
```

(continues on next page)

(continued from previous page)

```
}

void SIPO(unsigned char byte)
{
    int i;

    for(i=0;i<8;i++){
        digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));
        pulse(SRCLK);
    }
}

void init(void)
{
    pinMode(SDI, OUTPUT); //make P0 output
    pinMode(RCLK, OUTPUT); //make P0 output
    pinMode(SRCLK, OUTPUT); //make P0 output

    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 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<8;i++){
            SIPO(LED[i]);
            pulse(RCLK);
            delay(150);
            //printf("i = %d\n",i);
        }
        delay(500);

        for(i=0;i<3;i++){
            SIPO(0xff);
            pulse(RCLK);
            delay(100);
            SIPO(0x00);
            pulse(RCLK);
            delay(100);
        }
        delay(500);
//          digitalWrite(RCLK,0);

        for(i=0;i<8;i++){
            SIPO(LED[8-i-1]);
            pulse(RCLK);
```

(continues on next page)

(continued from previous page)

```
            delay(150);
        }
        delay(500);

        for(i=0;i<3;i++){
            SIPO(0xff);
            pulse(RCLK);
            delay(100);
            SIPO(0x00);
            pulse(RCLK);
            delay(100);
        }
        delay(500);
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 10_74HC595_LED.py
```

Here you should see eight LEDs blink regularly.

Code

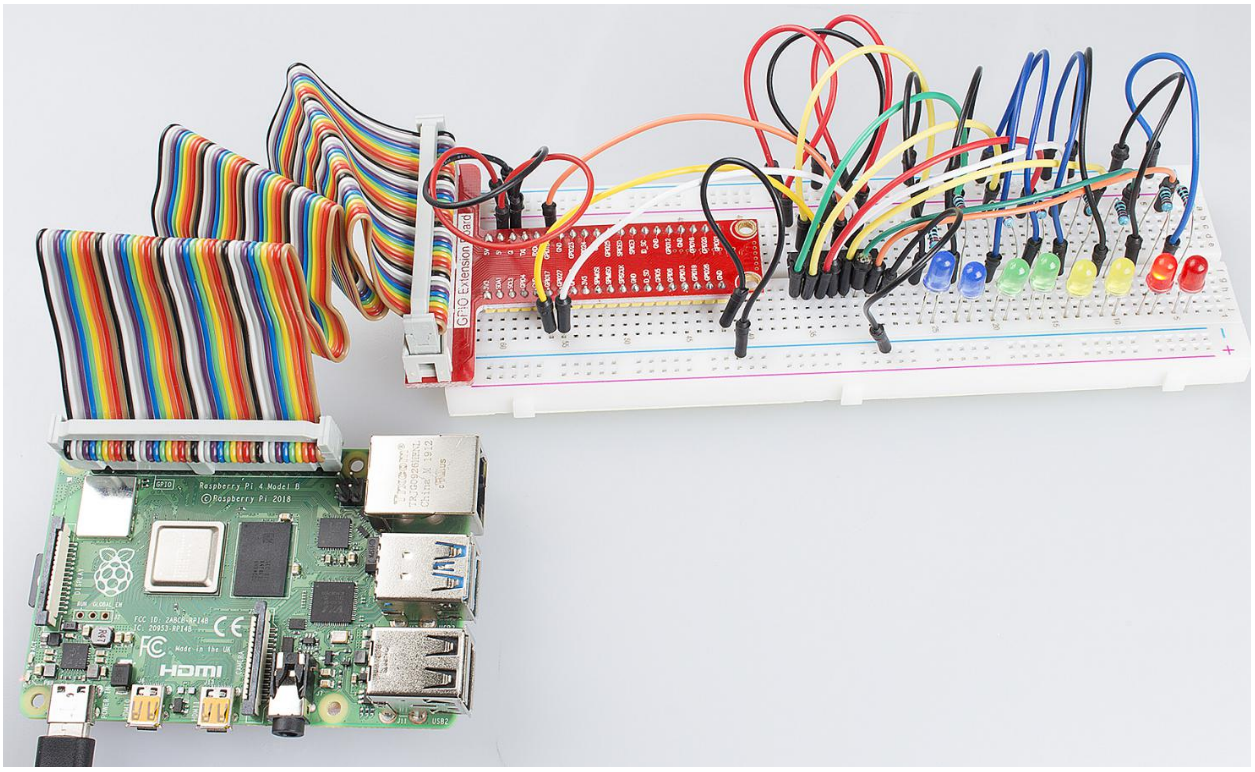
```
import RPi.GPIO as GPIO
import time

SDI   = 17
RCLK  = 18
SRCLK = 27

#================   LED Mode Defne ================
#   You can define yourself, in binay, and convert it to Hex
#   8 bits a group, 0 means off, 1 means on
#   like : 0101 0101, means LED1, 3, 5, 7 are on.(from left to right)
#   and convert to 0x55.

LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80]    #original mode
LED1 = [0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff]    #blink mode 1
LED2 = [0x01,0x05,0x15,0x55,0xb5,0xf5,0xfb,0xff]    #blink mode 2
LED3 = [0x02,0x03,0x0b,0x0f,0x2f,0x3f,0xbf,0xff]    #blink mode 3
#=================================================

def print_msg():
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
```

(continues on next page)

(continued from previous page)

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

def hc595_in(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)

def hc595_out():
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)

def loop():
    WhichLeds = LED0        # Change Mode, modes from LED0 to LED3
    sleeptime = 0.1         # Change speed, lower value, faster speed
    while True:
        for i in range(0, len(WhichLeds)):
            hc595_in(WhichLeds[i])
            hc595_out()
            time.sleep(sleeptime)

        for i in range(len(WhichLeds)-1, -1, -1):
            hc595_in(WhichLeds[i])
            hc595_out()
            time.sleep(sleeptime)

def destroy():   # When program ending, the function is executed.
    GPIO.cleanup()

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

6.10.6 Further Exploration



In this experiment, three Raspberry Pi GPIOs are used to separately control 8 LEDs based on 74HC595. In fact, 74HC595 has another powerful function – cascade. With cascade, you can use a microprocessor to control more peripherals. We'll check more details later.

6.11 Lesson 11 Driving 7-Segment Display by 74HC595

6.11.1 Introduction

Since we've got some knowledge of the 74HC595 in the previous lesson, now let's try to use it and drive a 7-segment display to show a figure from 0 to 9.

6.11.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * 74HC595

- 1 * 7-segment display

- 2 * Resistor (220K,10K)

- 1 * Button

- Jumper wires

6.11.3 Principle

7-Segment Display

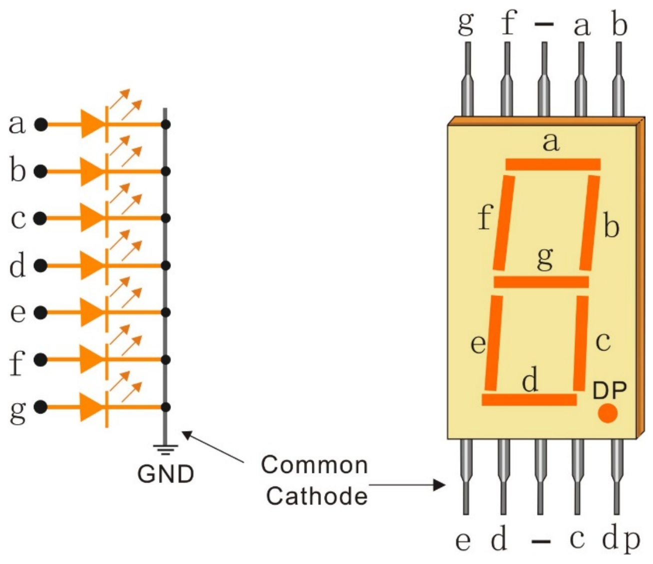
A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral (both decimal and hexadecimal) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

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.

The common pin of the display generally tells its type. There are two types of pin connection: a pin of connected cathodes and one of connected anodes, indicating 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.

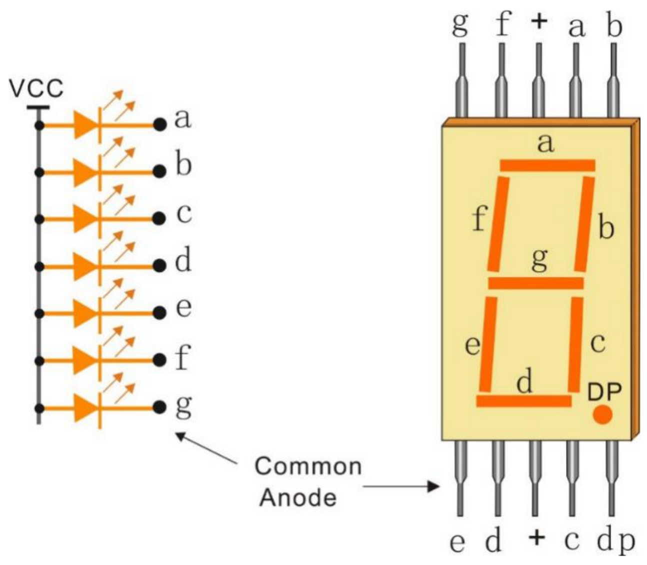
Common Cathode 7-Segment Display

In a common cathode display, the cathodes of all the LED segments are connected to the logic “0” or ground. Then an individual segment (a-g) is energized by a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the anode of the segment.



Common Anode 7-Segment Display

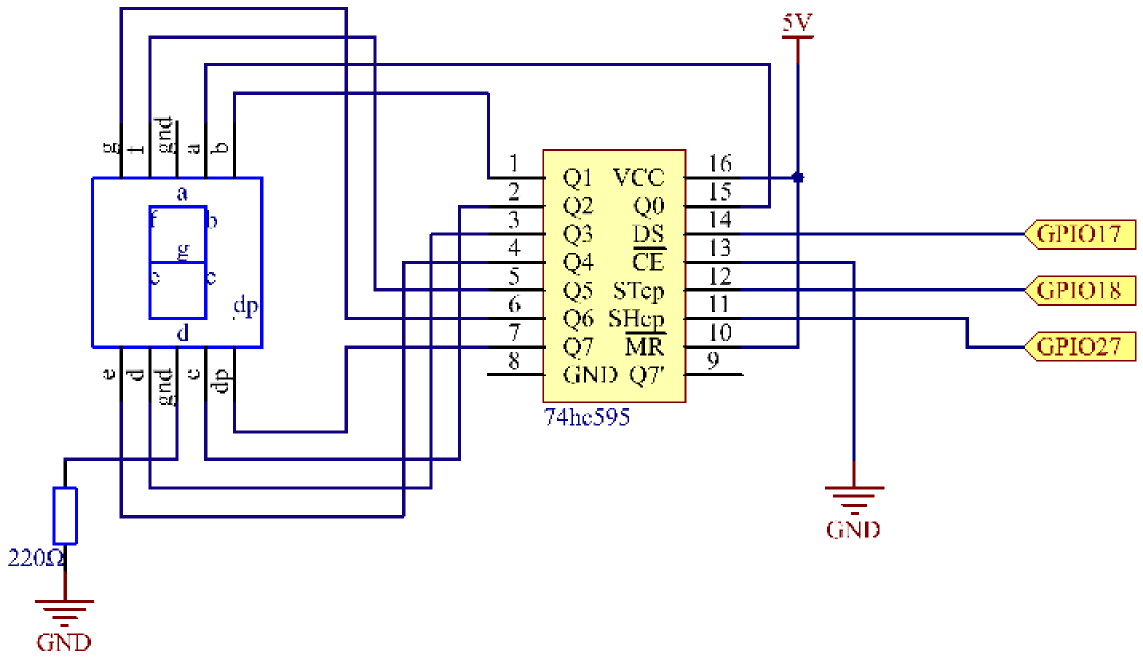
In a common anode display, the anodes of all the LED segments are connected to the logic "1". Then an individual segment (a-g) is energized by a ground, logic "0" or "LOW" signal via a current limiting resistor to the cathode of the segment.



In this experiment, a common cathode 7-segment display is use. It should be connected to ground. When the anode of an LED in a certain segment is at high level, the corresponding segment will light up; when it is at low, the segment will stay dim.

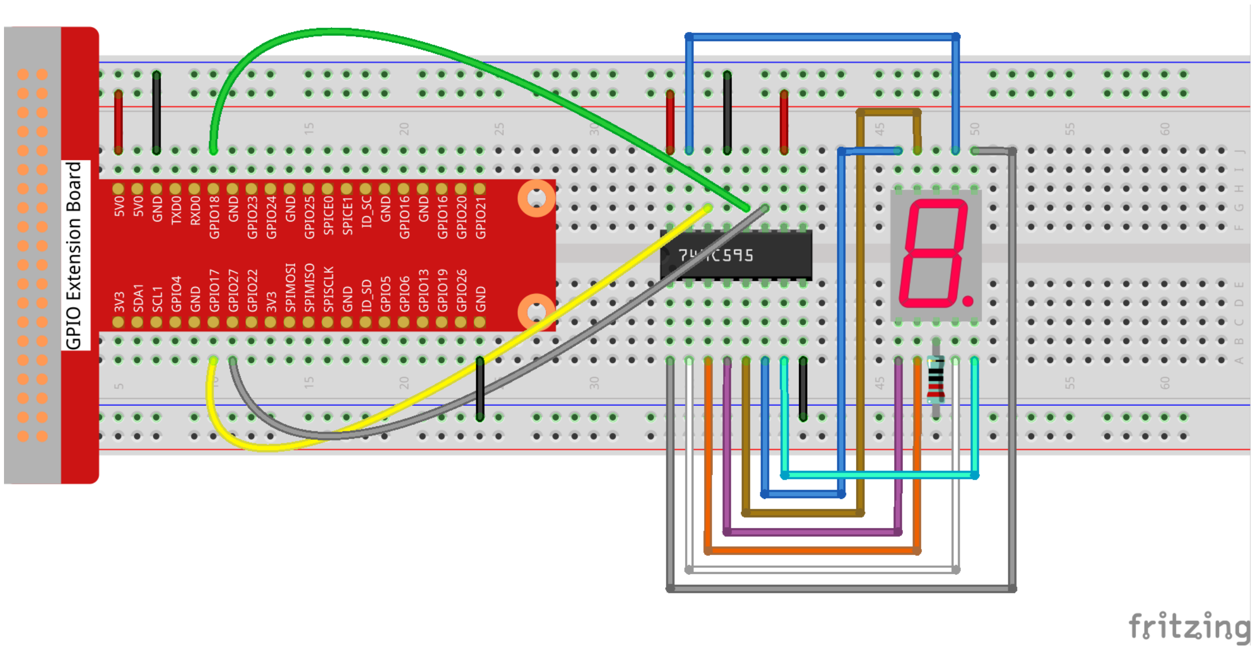
6.11.4 Schematic Diagram

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.



6.11.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/11_Segment/
```

Step 3: Compile.

```
gcc segment1.c -o segment1 -lwiringPi
```

Step 4: Run.

```
sudo ./segment1
```

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[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,
↪0x7c,0x39,0x5e,0x79,0x71,0x80};

void init(void)
{
    pinMode(SDI, OUTPUT); //make P0 output
```

(continues on next page)

(continued from previous page)

```
    pinMode(RCLK, OUTPUT); //make P0 output
    pinMode(SRCLK, OUTPUT); //make P0 output

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

void hc595_shift(unsigned char dat)
{
    int i;

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

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

int main(void)
{
    int i;

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

    init();

    while(1){
        for(i=0;i<17;i++){
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

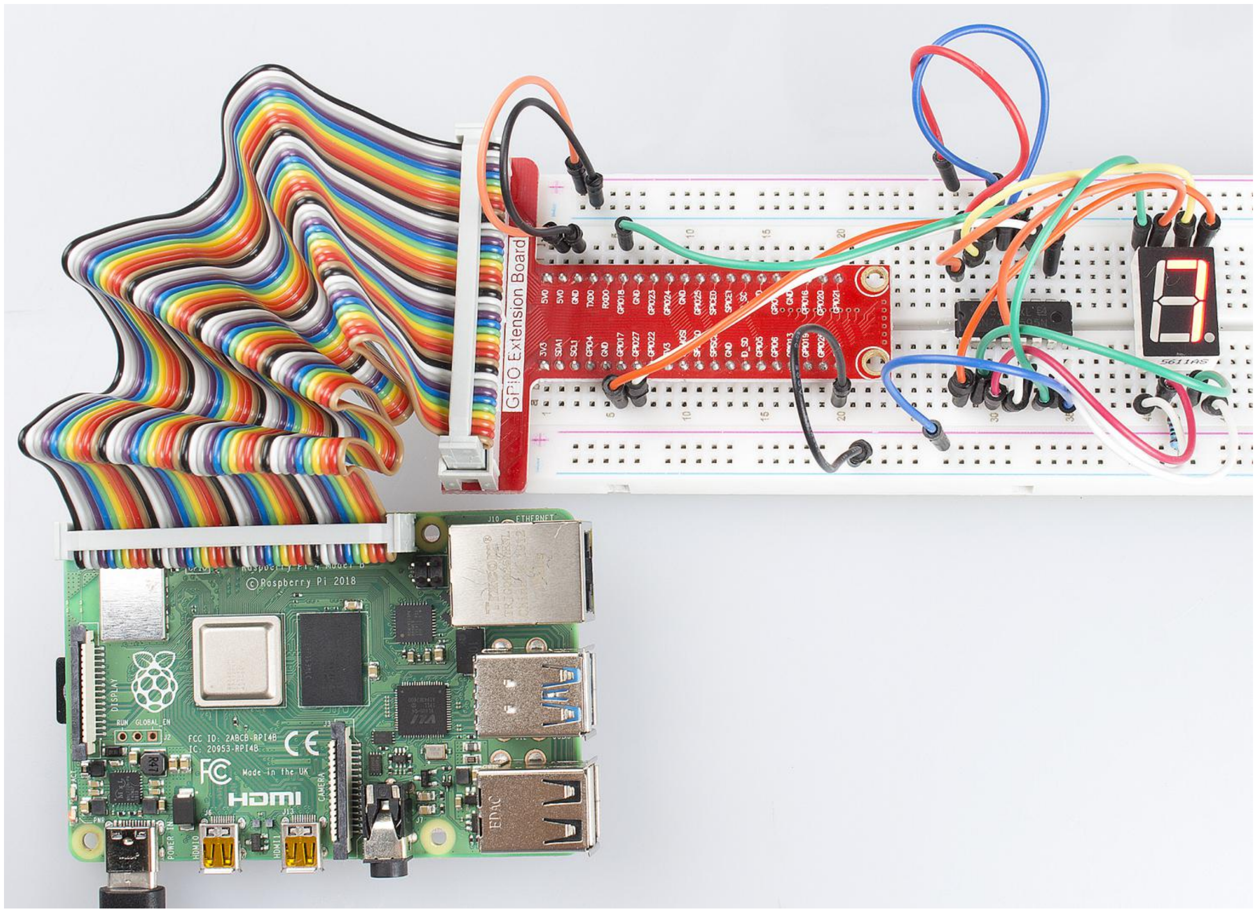
Step 3: Run.

```
sudo python3 11_segment.py
```

You should see the 7-segment display from 0 to 9, and A to F.

Code

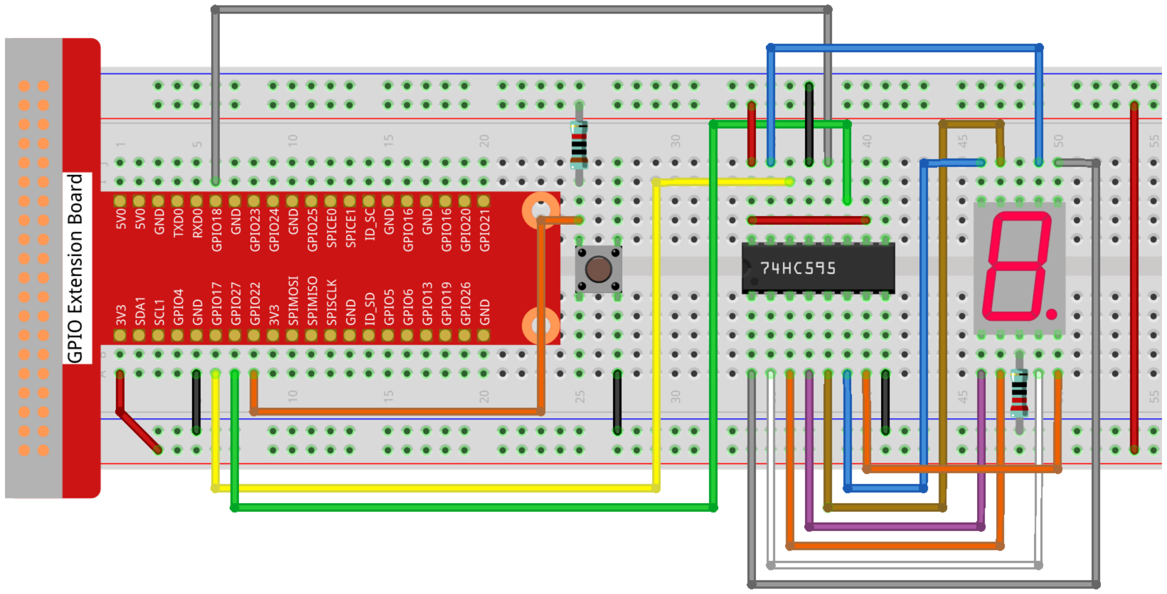
```python
import RPi.GPIO as GPIO
import time

SDI   = 17
RCLK  = 18
SRCLK = 27

segCode = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,
↪0x71,0x80]

def print_msg():
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")

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

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 loop():
    while True:
        for i in range(0, len(segCode)):
            hc595_shift(segCode[i])
            time.sleep(0.5)

def destroy():   #When program ending, the function is executed.
    GPIO.cleanup()

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



6.11.6 Further Exploration

You can slightly modify the hardware and software based on this experiment to make a dice. For hardware, add a button to the original board.

6.11.7 Build the circuit:



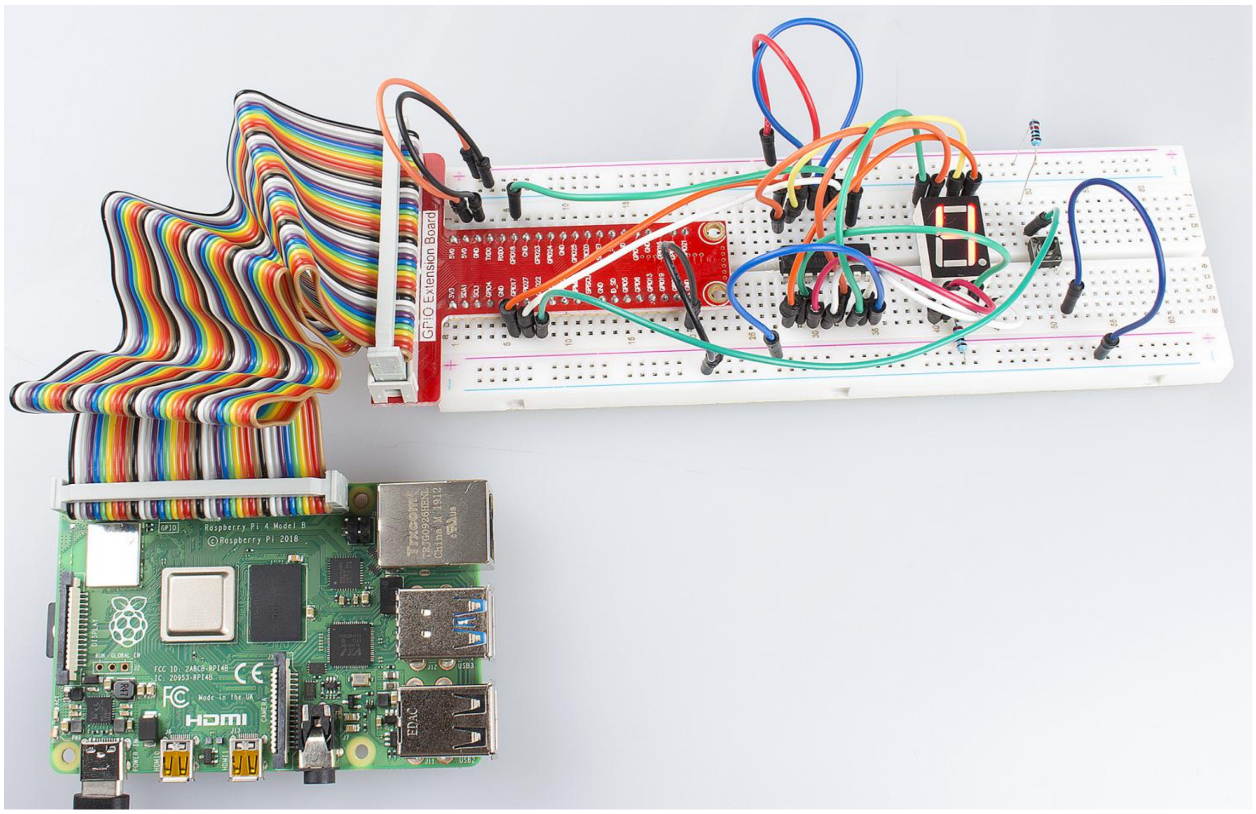
Next, go to *11_Segment*, and compile *dice.c*

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/11_Segment/

gcc dice.c -lwiringPi
```

Run.

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

Now you should see a number flashing between 0 and 6 quickly on the segment display. Press the button on the breadboard, and the display will statically display a random number between 0 and 6 for 2 seconds and then circularly flash randomly between 0 and 6 again.



6.11.8 Summary

Through this lesson, you may have mastered the basic principle and programming for 7-segment display based on Raspberry Pi, as well as more knowledge about using 74HC595. Now you can apply what you've learnt and put it into practice to create your own works!

6.12 Lesson 12 Driving Dot-Matrix by 74HC595

6.12.1 Introduction

With low-voltage scanning, dot matrix LED displays have advantages such as power saving, long service life, low cost, high brightness, a wide angle of view, long visual range, waterproofness, and so on. They can meet the needs of different applications and thus have a broad development prospect. In this lesson, we will learn how to use 74HC595 to drive an LED dot-matrix.

6.12.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 2 * 74HC595

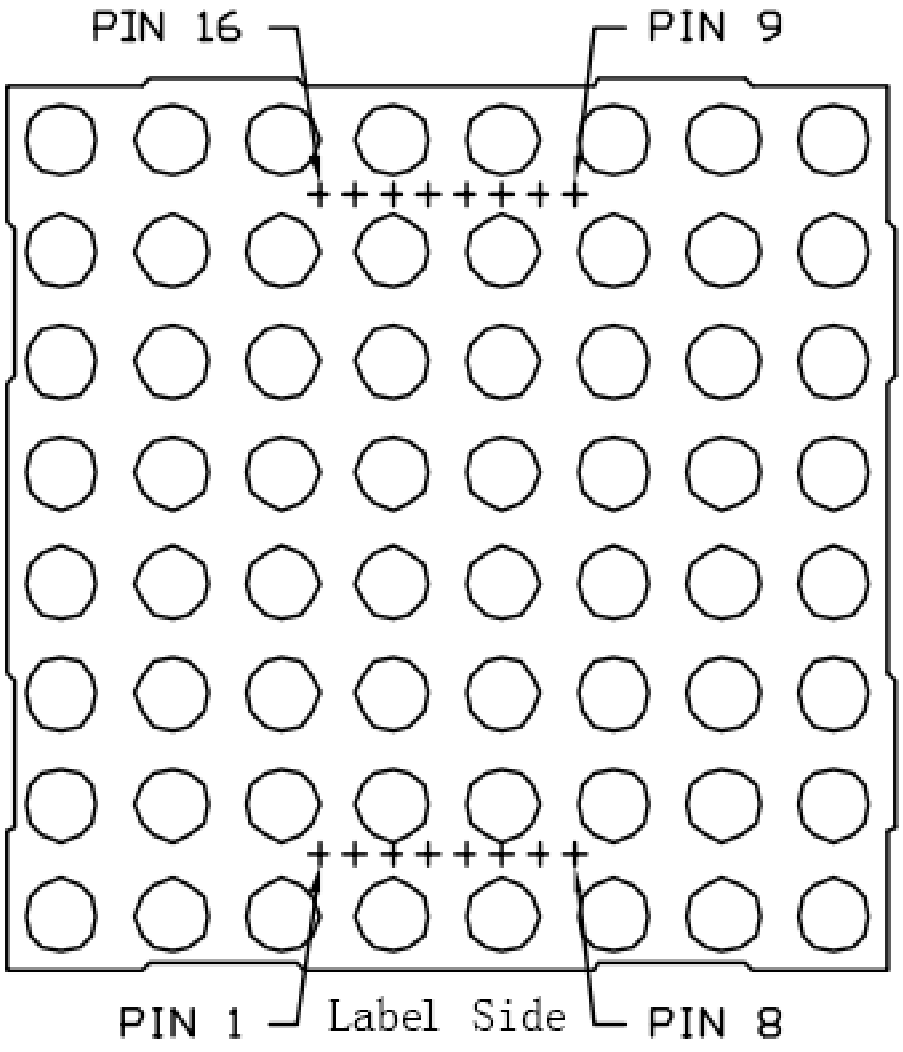
- 1 * Dot-Matrix

- Jumper wires

6.12.3 Principle

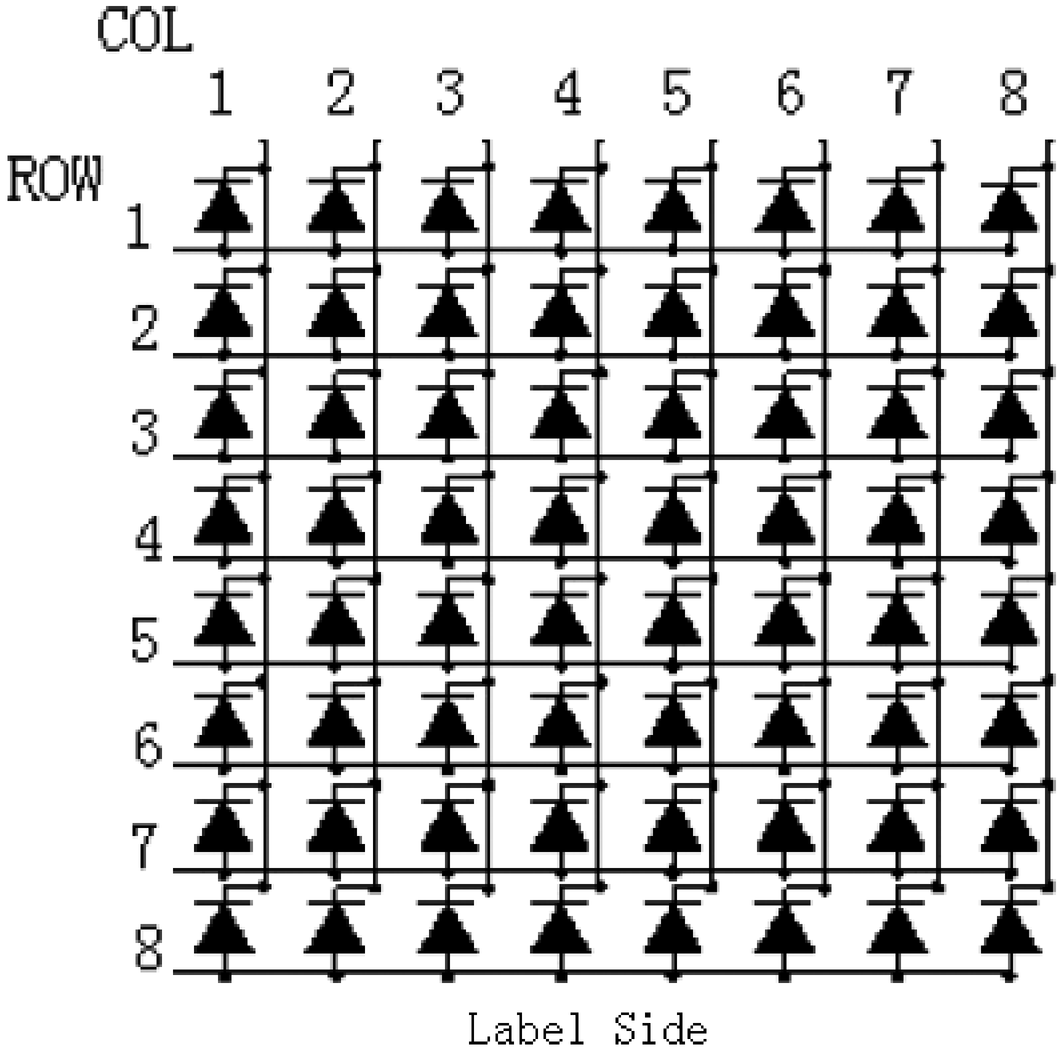
Dot Matrix

The external view:



Pin definition:

Define the row and column numbering at first (only for the dot matrix whose model number ends with BS)



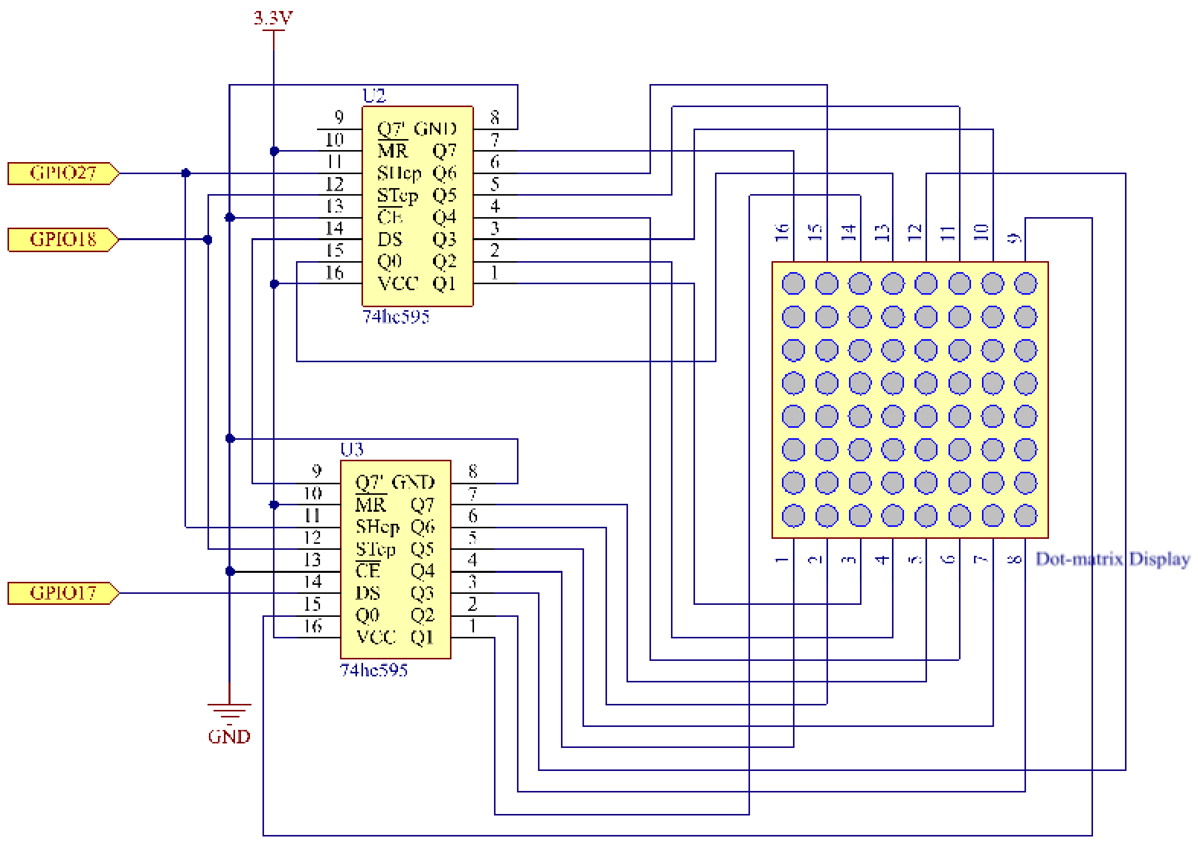
Pin numbering corresponding to the above rows and columns:

COL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**13**	**3**	**4**	**10**	**6**	**11**	**15**	**16**
ROW	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Pin No.	**9**	**14**	**8**	**12**	**1**	**7**	**2**	**5**

The 8*8 dot matrix is made up of sixty-four LEDs and each LED is placed at the cross point of a row and a column. When the electrical level of a certain row is High and the electrical level of a certain column is Low, the corresponding LED at their cross point will light up. For example, to turn on the LED at the first dot, you should set ROW 1 to high level and COL 1 to low, so the LED at the first dot brightens; to turn on all the LEDs on the first row, set ROW 1 to high level and COL 1-8 to low, and then all the LEDs on the first row will light up; similarly, set COL 1 to low level and ROW 1-8 to high level, and all the LEDs on the first column will light up.

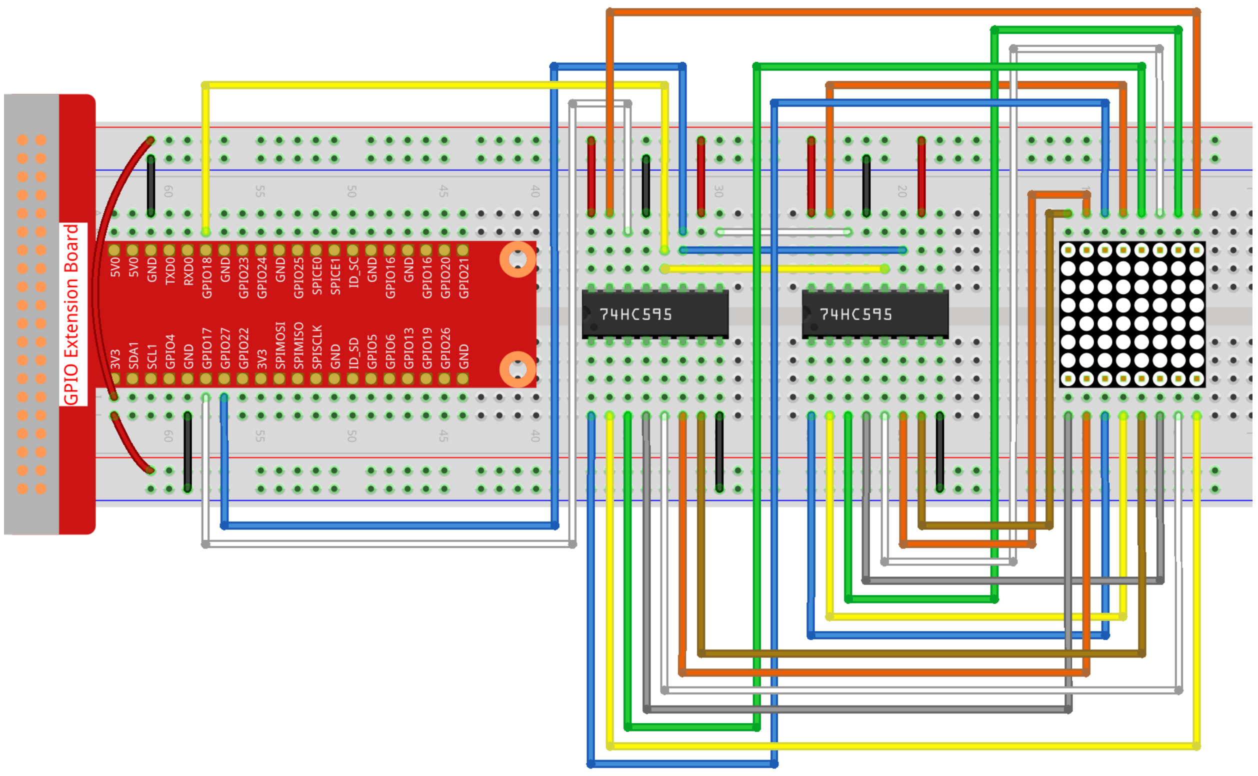
The principle of 74HC595 has been illustrated previously. One chip is used to control the rows of the dot matrix while the other, the columns.

6.12.4 Schematic Diagram



6.12.5 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/12_DotMatrix/
```

Step 3: Compile.

```
gcc dotMatrix.c -o dotMatrix -lwiringPi
```

Step 4: Run.

```
sudo ./dotMatrix
```

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

(continues on next page)

(continued from previous page)

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

//unsigned char code_L[8] = {0x00,0x00,0x3c,0x42,0x42,0x3c,0x00,0x00};
//unsigned char code_H[8] = {0xff,0xe7,0xdb,0xdb,0xdb,0xdb,0xe7,0xff};

//unsigned char code_L[8] = {0xff,0xff,0xc3,0xbd,0xbd,0xc3,0xff,0xff};
//unsigned char code_H[8] = {0x00,0x18,0x24,0x24,0x24,0x24,0x18,0x00};

void init(void)
{
    pinMode(SDI, OUTPUT); //make P0 output
    pinMode(RCLK, OUTPUT); //make P0 output
    pinMode(SRCLK, OUTPUT); //make P0 output

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

void hc595_in(unsigned char dat)
{
    int i;

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

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

int main(void)
{
    int i;

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

    init();

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

(continues on next page)

(continued from previous page)

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

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 12_dotMatrix.py
```

You should see LEDs light up as you control.

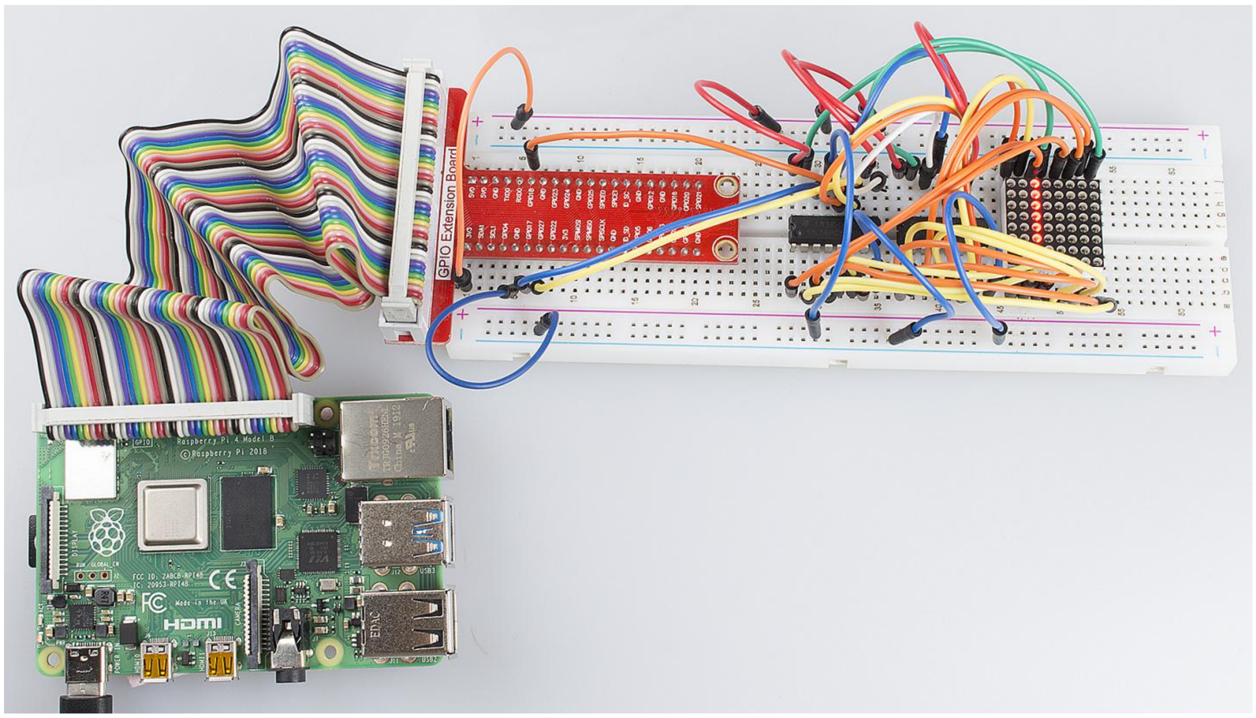
Code

```
import RPi.GPIO as GPIO
import time

SDI   = 17
RCLK  = 18
SRCLK = 27

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


def print_msg():
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")

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

def hc595_in(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
```

(continues on next page)

(continued from previous page)

```
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)

def hc595_out():
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)


def loop():
    while True:
        for i in range(0, len(code_H)):
            hc595_in(code_L[i])
            hc595_in(code_H[i])
            hc595_out()
            time.sleep(0.1)

        for i in range(len(code_H)-1, -1, -1):
            hc595_in(code_L[i])
            hc595_in(code_H[i])
            hc595_out()
            time.sleep(0.1)

def destroy():   # When program ending, the function is executed.
    GPIO.cleanup()

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



6.12.6 Summary

Through this lesson, you have got the basic principle of LED dot matrix and how to program the Raspberry Pi to drive an LED dot matrix based on 74HC595 cascade. With the knowledge learnt, try more fascinating creations!

6.13 Lesson 13 LCD1602

6.13.1 Introduction

In this lesson, we will learn how to use LCD1602 to display characters and strings.

6.13.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * LCD1602

- 1 * Potentiometer

- Jumper wires

6.13.3 Principle

LCD1602, or character type LCD1602, is a dot matrix LCD module specially used to display letters, figures, symbols, and so on. It consists of many 16*2 dot matrixes, and each one is composed of 5*7 or 5*11 character bit. Each character bit can display one character. There is a dot space between each adjacent character bit. Also there is a dot space between each row. The dot space functions as a character space or line space; thus, LCD1602 cannot display graphics very well. It is widely used in pocket instruments and low power application systems due to its micro power consumption, small size, richness in contents, ultra-thinness and lightness.

LCD1602 uses the standard 16-pin port, among which:

Pin 1 (GND): connected to Ground

Pin 2 (Vcc): connected to 5V power supply

Pin 3 (Vo): used to adjust the contrast of LCD1602; the level is lowest when it's connected to a positive power supply, and highest when connected to ground (you can connect a 10K potentiometer to adjust its contrast when using LCD1602)

Pin 4 (RS): register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

Pin 5 (R/W): to read/write signals; it reads signals when supplied with high level (1), and writes when low level (0) (in this experiment, you only need to write data to LCD1602, so just connect this pin to ground)

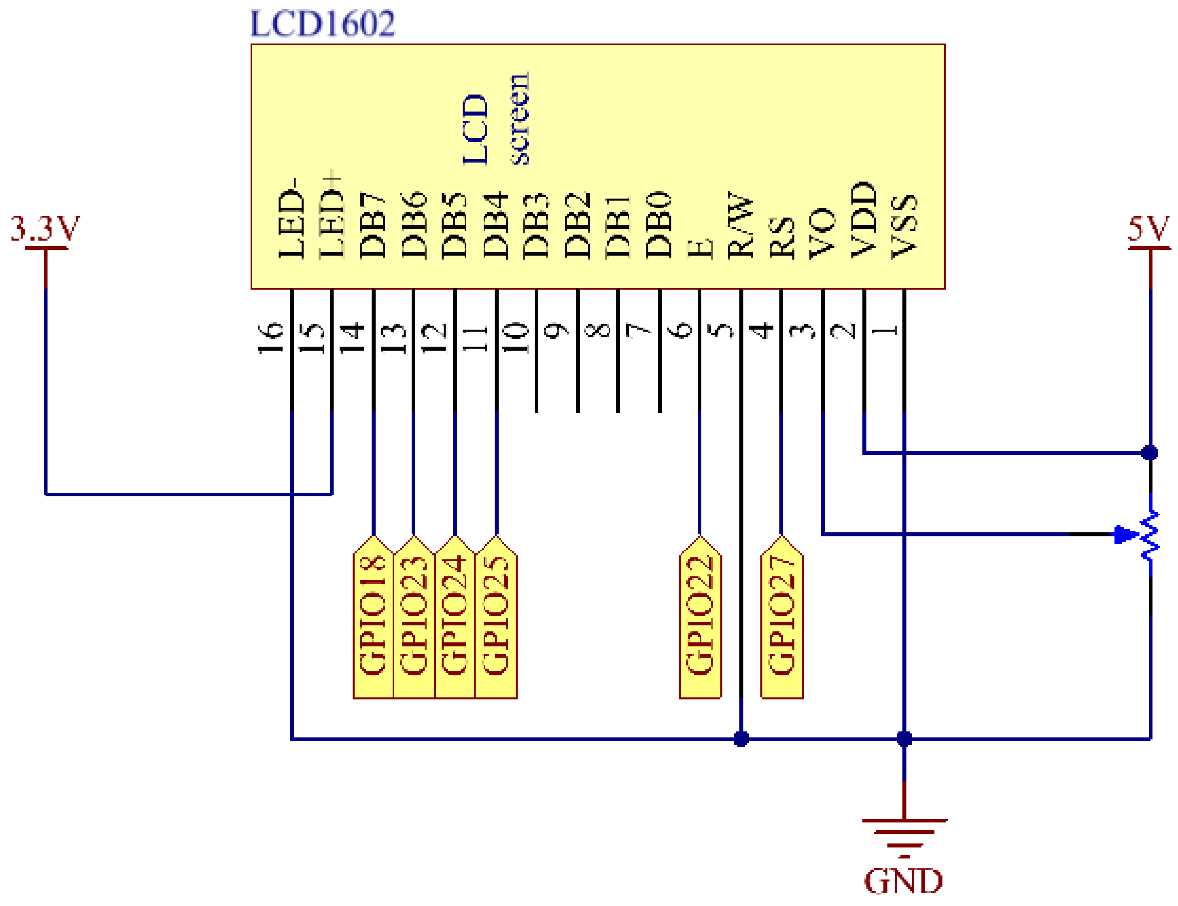
Pin 6 (E): An enable pin that, when low-level energy is supplied, causes the LCD module to execute relevant instructions

Pin 7 (D0-D7): pins that read and write data

A and K: controlling LCD backlight

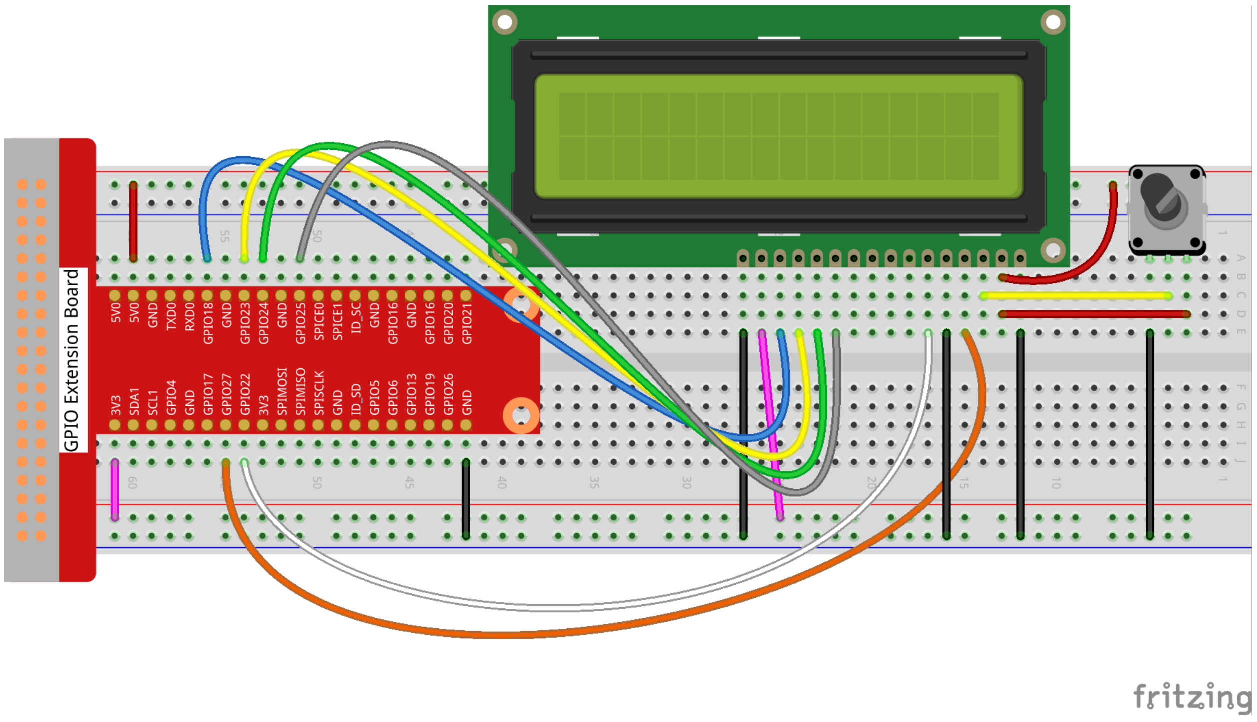
LCD1602 has two operation modes: 4-bit and 8-bit. When the IOs of microprocessor (MCU) are insufficient, you can choose 4-bit mode, under which only pins D4~D7 are used. After connecting the circuit, you can operate LCD1602 by Raspberry Pi.

6.13.4 Schematic Diagram



6.13.5 Experimental Procedures

Step1: Build the circuit (please be sure the pins are connected correctly. Otherwise, characters will not be displayed properly):



For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/13_LCD1602/
```

Step 3: Compile.

```
gcc lcd1602_2.c -o lcd1602_2 -lwiringPiDev -lwiringPi
```

Step 4: Run.

```
sudo ./lcd1602_2
```

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 <stdlib.h>
#include <wiringPi.h>
#include <lcd.h>

const unsigned char Buf[] = "---SUNFOUNDER---";
const unsigned char myBuf[] = "  sunfounder.com";

int main(void)
{
```

(continues on next page)

(continued from previous page)

```
    int fd;
    int i;
    if (wiringPiSetup() == -1){
        exit(1);
    }

    fd = lcdInit(2,16,4, 2,3, 6,5,4,1,0,0,0,0); //see /usr/local/include/lcd.h
    printf("%d", fd);
    if (fd == -1){
        printf("lcdInit 1 failed\n") ;
        return 1;
    }
    delay(1000);

    lcdClear(fd);
    lcdPosition(fd, 0, 0);
    lcdPuts(fd, "Welcom To--->");

    lcdPosition(fd, 0, 1);
    lcdPuts(fd, "  sunfounder.com");

    delay(1000);
    lcdClear(fd);

    while(1){
        for(i=0;i<sizeof(Buf)-1;i++){
            lcdPosition(fd, i, 1);
            lcdPutchar(fd, *(Buf+i));
            delay(200);
        }
        lcdPosition(fd, 0, 1);
        lcdClear(fd);
        delay(500);
        for(i=0; i<16; i++){
            lcdPosition(fd, i, 0);
            lcdPutchar(fd, *(myBuf+i));
            delay(100);
        }
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 13_lcd1602.py
```

You should see two lines of characters displayed on the LCD1602: "**SUNFOUNDER**" and "**Hello World ! :)**".

Code

```
from time import sleep

class LCD:
    # commands
    LCD_CLEARDISPLAY        = 0x01
    LCD_RETURNHOME          = 0x02
    LCD_ENTRYMODESET        = 0x04
    LCD_DISPLAYCONTROL      = 0x08
    LCD_CURSORSHIFT         = 0x10
    LCD_FUNCTIONSET         = 0x20
    LCD_SETCGRAMADDR        = 0x40
    LCD_SETDDRAMADDR        = 0x80

    # flags for display entry mode
    LCD_ENTRYRIGHT      = 0x00
    LCD_ENTRYLEFT       = 0x02
    LCD_ENTRYSHIFTINCREMENT     = 0x01
    LCD_ENTRYSHIFTDECREMENT     = 0x00

    # flags for display on/off control
    LCD_DISPLAYON       = 0x04
    LCD_DISPLAYOFF      = 0x00
    LCD_CURSORON        = 0x02
    LCD_CURSOROFF       = 0x00
    LCD_BLINKON         = 0x01
    LCD_BLINKOFF        = 0x00

    # flags for display/cursor shift
    LCD_DISPLAYMOVE     = 0x08
    LCD_CURSORMOVE      = 0x00

    # flags for display/cursor shift
    LCD_DISPLAYMOVE     = 0x08
    LCD_CURSORMOVE      = 0x00
    LCD_MOVERIGHT       = 0x04
    LCD_MOVELEFT        = 0x00

    # flags for function set
    LCD_8BITMODE        = 0x10
    LCD_4BITMODE        = 0x00
    LCD_2LINE           = 0x08
    LCD_1LINE           = 0x00
    LCD_5x10DOTS        = 0x04
    LCD_5x8DOTS         = 0x00

    def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO = None):
        # Emulate the old behavior of using RPi.GPIO if we haven't been given
        # an explicit GPIO interface to use
        if not GPIO:
            import RPi.GPIO as GPIO
            self.GPIO = GPIO
            self.pin_rs = pin_rs
            self.pin_e = pin_e
            self.pins_db = pins_db

            self.used_gpio = self.pins_db[:]
            self.used_gpio.append(pin_e)
```

(continues on next page)

(continued from previous page)

```
            self.used_gpio.append(pin_rs)

            self.GPIO.setwarnings(False)
            self.GPIO.setmode(GPIO.BCM)
            self.GPIO.setup(self.pin_e, GPIO.OUT)
            self.GPIO.setup(self.pin_rs, GPIO.OUT)

            for pin in self.pins_db:
                self.GPIO.setup(pin, GPIO.OUT)

        self.write4bits(0x33) # initialization
        self.write4bits(0x32) # initialization
        self.write4bits(0x28) # 2 line 5x7 matrix
        self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor
        self.write4bits(0x06) # shift cursor right

        self.displaycontrol = self.LCD_DISPLAYON | self.LCD_CURSOROFF | self.LCD_
↪BLINKOFF

        self.displayfunction = self.LCD_4BITMODE | self.LCD_1LINE | self.LCD_5x8DOTS
        self.displayfunction |= self.LCD_2LINE

        """ Initialize to default text direction (for romance languages) """
        self.displaymode =  self.LCD_ENTRYLEFT | self.LCD_ENTRYSHIFTDECREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode) #  set the entry␣
↪mode

        self.clear()

    def begin(self, cols, lines):
        if (lines > 1):
            self.numlines = lines
            self.displayfunction |= self.LCD_2LINE
            self.currline = 0

    def home(self):
        self.write4bits(self.LCD_RETURNHOME) # set cursor position to zero
        self.delayMicroseconds(3000) # this command takes a long time!

    def clear(self):
        self.write4bits(self.LCD_CLEARDISPLAY) # command to clear display
        self.delayMicroseconds(3000)    # 3000 microsecond sleep, clearing the␣
↪display takes a long time

    def setCursor(self, col, row):
        self.row_offsets = [ 0x00, 0x40, 0x14, 0x54 ]

        if ( row > self.numlines ):
            row = self.numlines - 1 # we count rows starting w/0

        self.write4bits(self.LCD_SETDDRAMADDR | (col + self.row_offsets[row]))

    def noDisplay(self):
        # Turn the display off (quickly)
        self.displaycontrol &= ~self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)
```

(continues on next page)

(continued from previous page)

```
    def display(self):
        # Turn the display on (quickly)
        self.displaycontrol |= self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noCursor(self):
        # Turns the underline cursor on/off
        self.displaycontrol &= ~self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def cursor(self):
        # Cursor On
        self.displaycontrol |= self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def scrollDisplayLeft(self):
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE | self.LCD_
↪MOVELEFT)

    def scrollDisplayRight(self):
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE | self.LCD_
↪MOVERIGHT);

    def leftToRight(self):
        # This is for text that flows Left to Right
        self.displaymode |= self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode);

    def rightToLeft(self):
        # This is for text that flows Right to Left
        self.displaymode &= ~self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def autoscroll(self):
        # This will 'right justify' text from the cursor
        self.displaymode |= self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def noAutoscroll(self):
        # This will 'left justify' text from the cursor
        self.displaymode &= ~self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def write4bits(self, bits, char_mode=False):
        # Send command to LCD
```

(continues on next page)

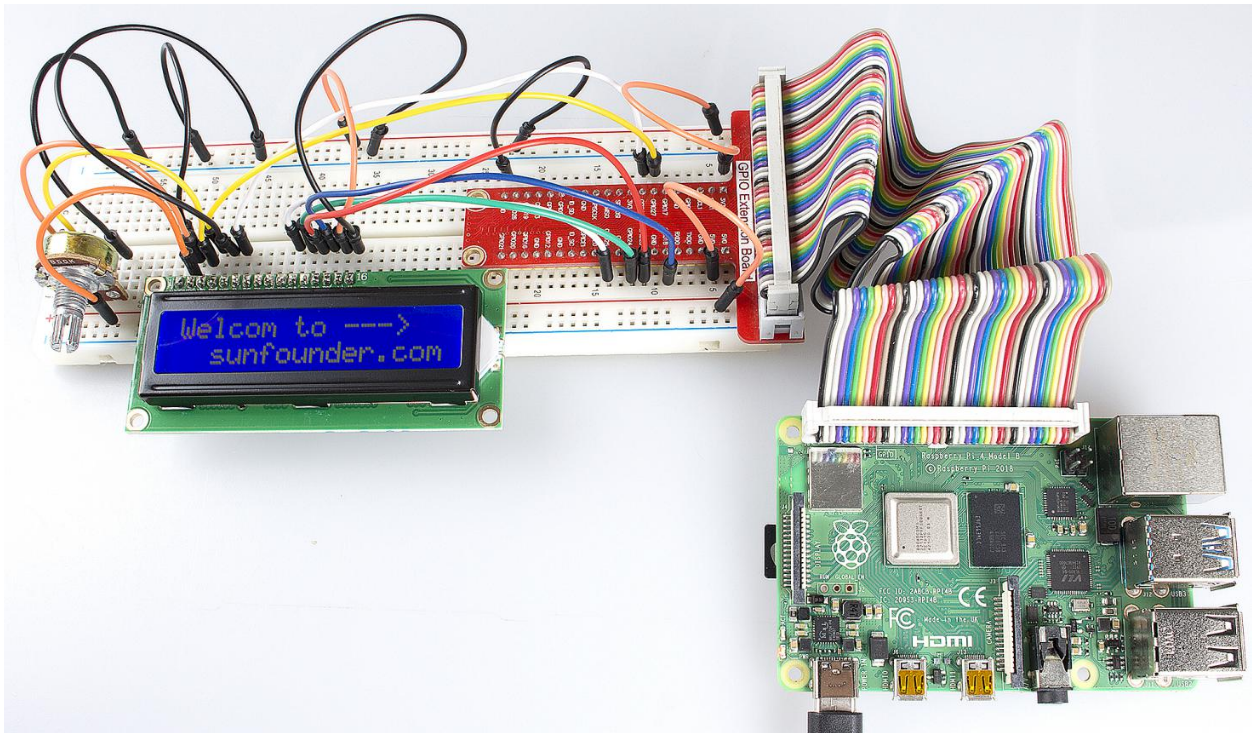
(continued from previous page)

```
        self.delayMicroseconds(1000) # 1000 microsecond sleep
        bits=bin(bits)[2:].zfill(8)
        self.GPIO.output(self.pin_rs, char_mode)
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i], True)
        self.pulseEnable()
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4,8):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i-4], True)
        self.pulseEnable()

    def delayMicroseconds(self, microseconds):
        seconds = microseconds / float(1000000) # divide microseconds by 1 million␣
↪for seconds
        sleep(seconds)

    def pulseEnable(self):
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)       # 1 microsecond pause - enable pulse must be >
↪ 450ns
        self.GPIO.output(self.pin_e, True)
        self.delayMicroseconds(1)       # 1 microsecond pause - enable pulse must be >
↪ 450ns
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)       # commands need > 37us to settle

    def message(self, text):
        # Send string to LCD. Newline wraps to second line
        print ("message: %s"%text)
        for char in text:
            if char == '\n':
                self.write4bits(0xC0) # next line
            else:
                self.write4bits(ord(char),True)

    def destroy(self):
        #print ("clean up used_gpio")
        self.GPIO.cleanup(self.used_gpio)

def loop():
    global lcd
    lcd = LCD()
    while True:
        lcd.clear()
        lcd.message(" LCD 1602 Test \n123456789ABCDEF")
        sleep(2)
        lcd.clear()
        lcd.message("   SUNFOUNDER \nHello World ! :)")
        sleep(2)
        lcd.clear()
        lcd.message("Welcom to --->\n  sunfounder.com")
        sleep(2)
```

(continues on next page)

(continued from previous page)

```
def destroy():
    lcd.destroy()

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



6.13.6 Further Exploration

In this experiment, the LCD1602 is driven in the 4-bit mode. You can try programming by yourself to drive it in the 8-bit mode.

6.14 Lesson 14 ADXL345

6.14.1 Introduction

In this lesson, we will learn how to use the acceleration sensor ADXL345.

6.14.2 Components

- 1 * Raspberry Pi

- 1 * Breadboard

- 1 * ADXL345 module

- Jumper wires

6.14.3 Principle

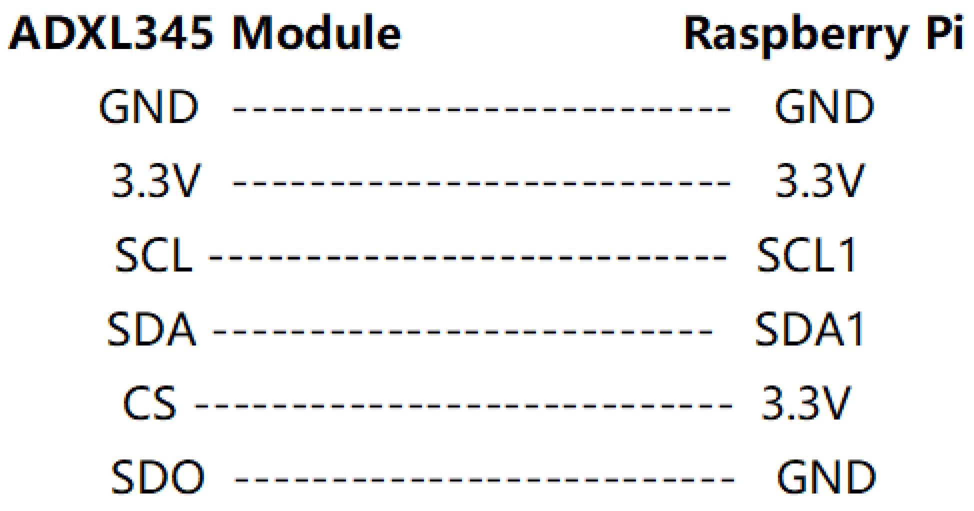
ADXL345

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. Digital output data is formatted as 16-bit two's complement and is accessible through either an SPI (3- or 4-wire) or I2C digital interface.

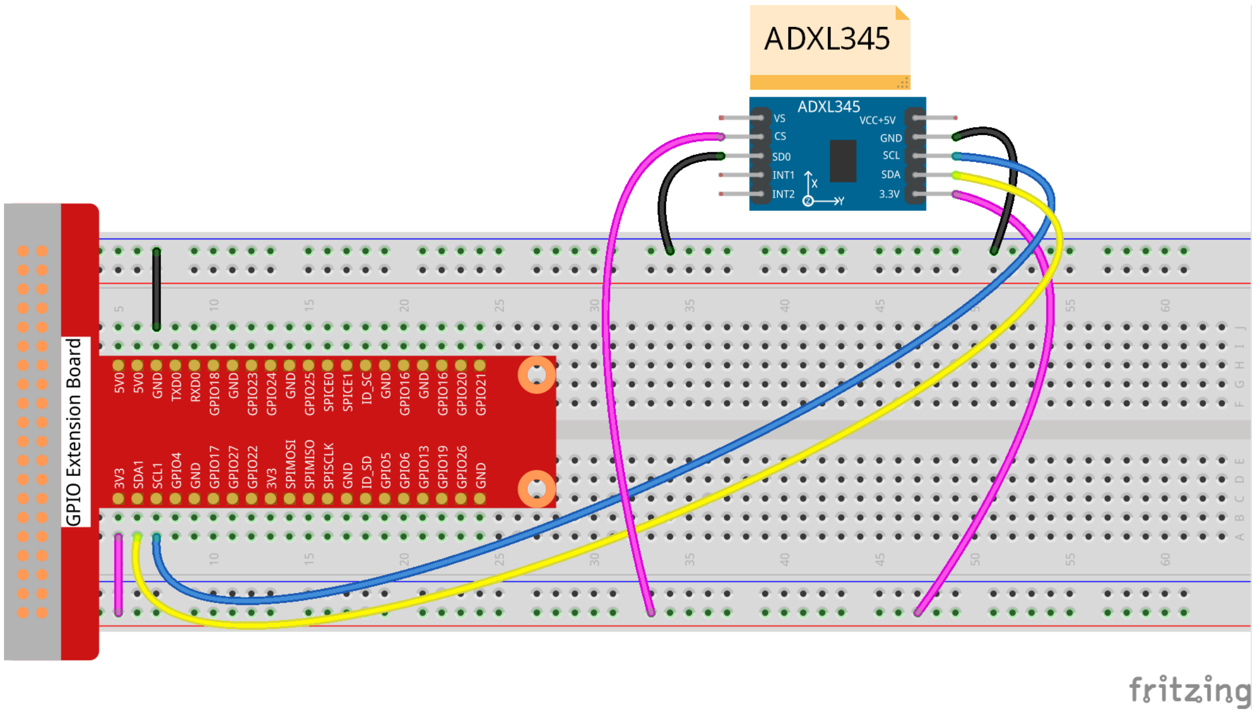
The ADXL345 is well suited to measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables the inclination change measurement by less than 1.0°. And the excellent sensitivity (3.9mg/LSB @2g) provides a high-precision output of up to ±16g. In this experiment, I2C digital interface is used.

6.14.4 Experimental Procedures

Step 1: Build the circuit.



ADXL345 Module		Raspberry Pi
GND	--------------------------	GND
3.3V	--------------------------	3.3V
SCL	--------------------------	SCL1
SDA	--------------------------	SDA1
CS	--------------------------	3.3V
SDO	--------------------------	GND



The I2C interface is used in the following program. Before running the program, please make sure the I2C driver module of Raspberry Pi has loaded normally(Refer to Appendix).

For C Language Users:

Step 2: Change directory.

```
cd /home/pi/Sunfounder_SuperKit_C_code_for_RaspberryPi/14_ADXL345/
```

Step 3: Compile.

```
gcc adxl345.c -o adxl345 -lwiringPi
```

Step 4: Run.

```
sudo ./adxl345
```

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 <wiringPiI2C.h>
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define  DevAddr  0x53  //device address
```

(continues on next page)

(continued from previous page)

```c
struct acc_dat{
    int x;
    int y;
    int z;
};

void adxl345_init(int fd)
{
    wiringPiI2CWriteReg8(fd, 0x31, 0x0b);
    wiringPiI2CWriteReg8(fd, 0x2d, 0x08);
//  wiringPiI2CWriteReg8(fd, 0x2e, 0x00);
    wiringPiI2CWriteReg8(fd, 0x1e, 0x00);
    wiringPiI2CWriteReg8(fd, 0x1f, 0x00);
    wiringPiI2CWriteReg8(fd, 0x20, 0x00);

    wiringPiI2CWriteReg8(fd, 0x21, 0x00);
    wiringPiI2CWriteReg8(fd, 0x22, 0x00);
    wiringPiI2CWriteReg8(fd, 0x23, 0x00);

    wiringPiI2CWriteReg8(fd, 0x24, 0x01);
    wiringPiI2CWriteReg8(fd, 0x25, 0x0f);
    wiringPiI2CWriteReg8(fd, 0x26, 0x2b);
    wiringPiI2CWriteReg8(fd, 0x27, 0x00);

    wiringPiI2CWriteReg8(fd, 0x28, 0x09);
    wiringPiI2CWriteReg8(fd, 0x29, 0xff);
    wiringPiI2CWriteReg8(fd, 0x2a, 0x80);
    wiringPiI2CWriteReg8(fd, 0x2c, 0x0a);
    wiringPiI2CWriteReg8(fd, 0x2f, 0x00);
    wiringPiI2CWriteReg8(fd, 0x38, 0x9f);
}

struct acc_dat adxl345_read_xyz(int fd)
{
    char x0, y0, z0, x1, y1, z1;
    struct acc_dat acc_xyz;

    x0 = 0xff - wiringPiI2CReadReg8(fd, 0x32);
    x1 = 0xff - wiringPiI2CReadReg8(fd, 0x33);
    y0 = 0xff - wiringPiI2CReadReg8(fd, 0x34);
    y1 = 0xff - wiringPiI2CReadReg8(fd, 0x35);
    z0 = 0xff - wiringPiI2CReadReg8(fd, 0x36);
    z1 = 0xff - wiringPiI2CReadReg8(fd, 0x37);

    printf("  x0 = %d   ",x0);printf("x1 = %d  \n",x1);
    printf("  y0 = %d   ",y0);printf("y1 = %d  \n",y1);
    printf("  z0 = %d   ",z0);printf("z1 = %d  \n",z1);

    acc_xyz.x = (int)(x1 << 8) + (int)x0;
    acc_xyz.y = (int)(y1 << 8) + (int)y0;
    acc_xyz.z = (int)(z1 << 8) + (int)z0;

    if(acc_xyz.x > 32767){
        acc_xyz.x -= 65536;
    }
    if(acc_xyz.y > 32767){
        acc_xyz.y -= 65536;
```

(continues on next page)

(continued from previous page)

```
    }
    if(acc_xyz.z > 32767){
        acc_xyz.z -= 65536;
    }

    return acc_xyz;
}

int main(void)
{
    int fd;
    struct acc_dat acc_xyz;

    fd = wiringPiI2CSetup(DevAddr);

    if(-1 == fd){
        perror("I2C device setup error");
    }

    adxl345_init(fd);

    while(1){
        acc_xyz = adxl345_read_xyz(fd);
        printf("x: %d  y: %d  z: %d\n", acc_xyz.x, acc_xyz.y, acc_xyz.z);
        delay(1000);
    }

    return 0;
}
```

For Python Users:

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 14_ADXL345.py
```

Now, rotate the acceleration sensor, and you should see the values printed on the screen change.

Code

```
from I2C import I2C
from time import sleep
import RPi.GPIO as GPIO
from sys import version_info

if version_info.major == 3:
    raw_input = input

class ADXL345(I2C):

    ADXL345_ADDRESS          = 0x53
```

(continues on next page)

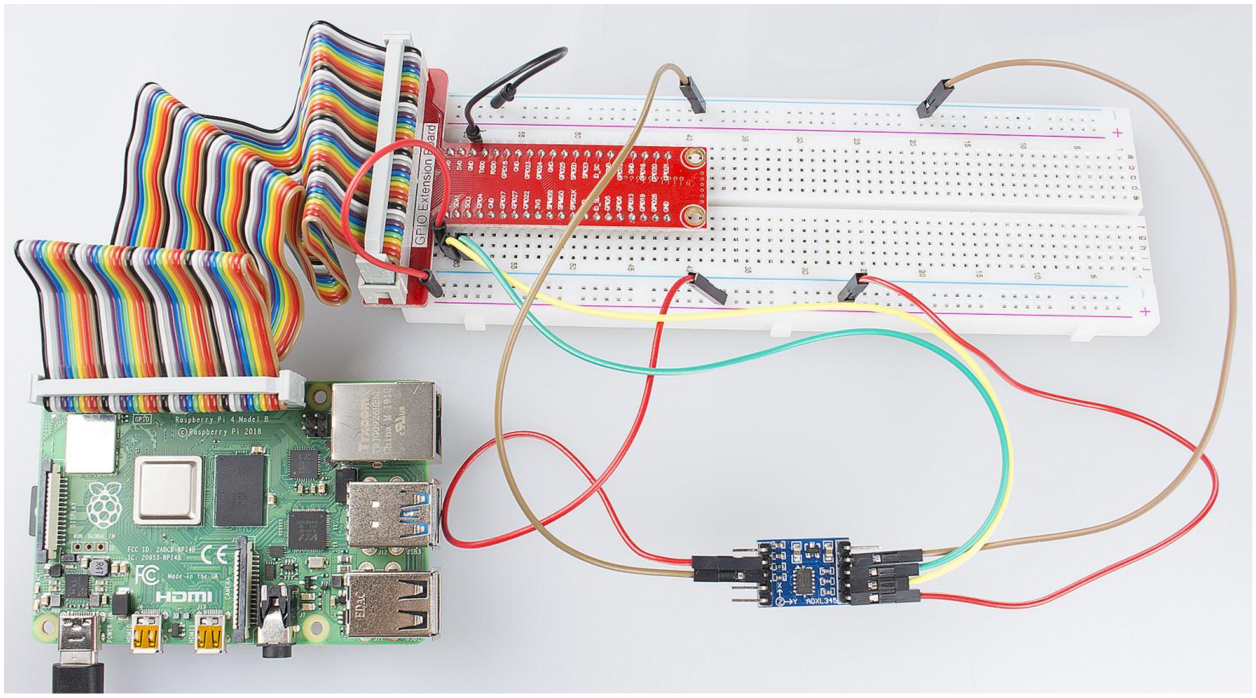
(continued from previous page)

```
    ADXL345_REG_DEVID         = 0x00 # Device ID
    ADXL345_REG_DATAX0        = 0x32 # X-axis data 0 (6 bytes for X/Y/Z)
    ADXL345_REG_POWER_CTL     = 0x2D # Power-saving features control

    ADXL345_DATARATE_0_10_HZ = 0x00
    ADXL345_DATARATE_0_20_HZ = 0x01
    ADXL345_DATARATE_0_39_HZ = 0x02
    ADXL345_DATARATE_0_78_HZ = 0x03
    ADXL345_DATARATE_1_56_HZ = 0x04
    ADXL345_DATARATE_3_13_HZ = 0x05
    ADXL345_DATARATE_6_25HZ  = 0x06
    ADXL345_DATARATE_12_5_HZ = 0x07
    ADXL345_DATARATE_25_HZ   = 0x08
    ADXL345_DATARATE_50_HZ   = 0x09
    ADXL345_DATARATE_100_HZ  = 0x0A # (default)
    ADXL345_DATARATE_200_HZ  = 0x0B
    ADXL345_DATARATE_400_HZ  = 0x0C
    ADXL345_DATARATE_800_HZ  = 0x0D
    ADXL345_DATARATE_1600_HZ = 0x0E
    ADXL345_DATARATE_3200_HZ = 0x0F

    ADXL345_RANGE_2_G         = 0x00 # +/-  2g (default)
    ADXL345_RANGE_4_G         = 0x01 # +/-  4g
    ADXL345_RANGE_8_G         = 0x02 # +/-  8g
    ADXL345_RANGE_16_G        = 0x03 # +/- 16g

    def __init__(self, busnum=1, debug=False):
        self.accel = I2C(self.ADXL345_ADDRESS, busnum, debug)
        if self.accel.readU8(self.ADXL345_REG_DEVID) == 0xE5:
            # Enable the accelerometer
            self.accel.write8(self.ADXL345_REG_POWER_CTL, 0x08)

    def setRange(self, range):
        # Read the data format register to preserve bits.  Update the data
        # rate, make sure that the FULL-RES bit is enabled for range scaling
        format = ((self.accel.readU8(self.ADXL345_REG_DATA_FORMAT) & ~0x0F) |
          range | 0x08)
        # Write the register back to the IC
        seld.accel.write8(self.ADXL345_REG_DATA_FORMAT, format)

    def getRange(self):
        return self.accel.readU8(self.ADXL345_REG_DATA_FORMAT) & 0x03

    def setDataRate(self, dataRate):
        # Note: The LOW_POWER bits are currently ignored,
        # we always keep the device in 'normal' mode
        self.accel.write8(self.ADXL345_REG_BW_RATE, dataRate & 0x0F)

    def getDataRate(self):
        return self.accel.readU8(self.ADXL345_REG_BW_RATE) & 0x0F

    # Read the accelerometer
    def read(self):
        raw = self.accel.readList(self.ADXL345_REG_DATAX0, 6)
        print (raw)
        res = []
```

(continues on next page)

(continued from previous page)

```
        for i in range(0, 6, 2):
            g = raw[i] | (raw[i+1] << 8)
            g = 65535 - g
            if g > 32767:
                g -= 65535
            res.append(g)
        return res

    # Simple example prints accelerometer data once per second:
    def main():
            accel = ADXL345()
            accel.setRange(accel.ADXL345_RANGE_16_G)
            while True:
                    x, y, z = accel.read()
                    print('X: %.2f, Y: %.2f, Z: %.2f'%(x, y, z))
                    sleep(1) # Output is fun to watch if this is commented out

    def destroy():
            exit()

def destroy():
    exit()

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



CHAPTER

SEVEN

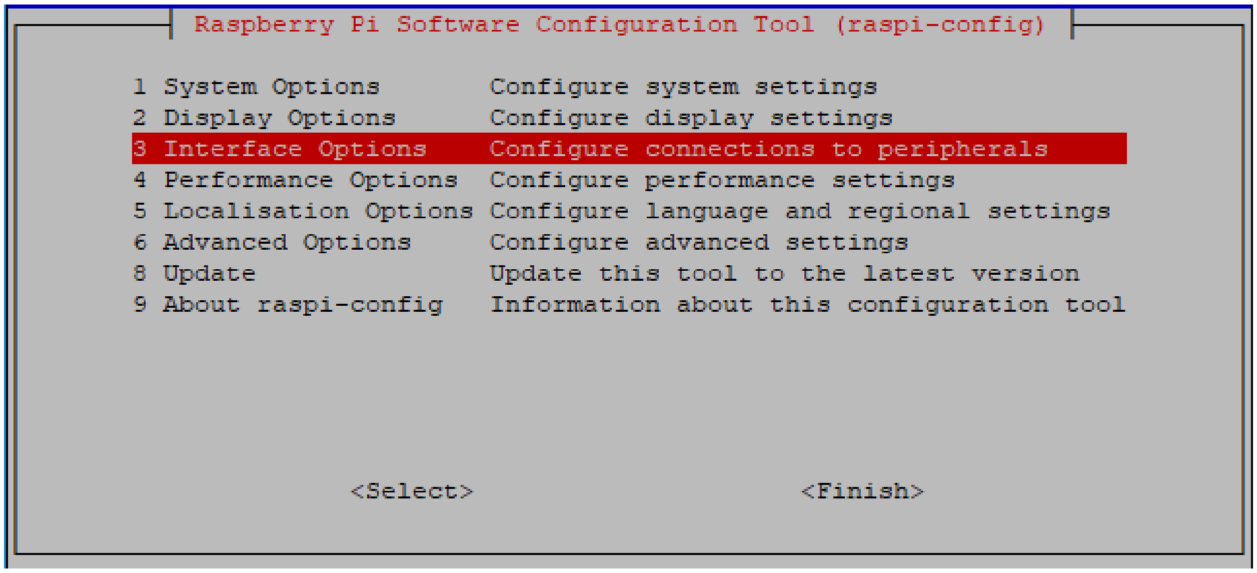
APPENDIX

7.1 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).

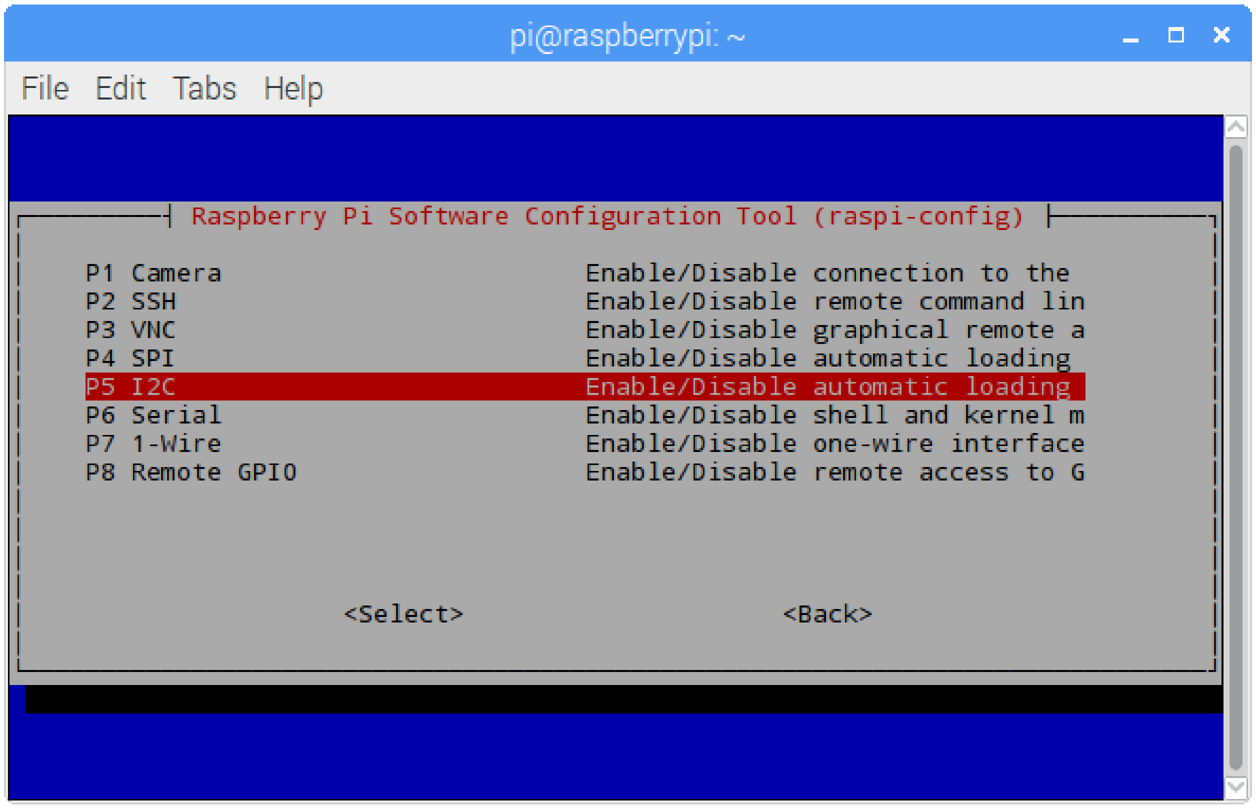
```
sudo raspi-config
```

3 Interfacing options



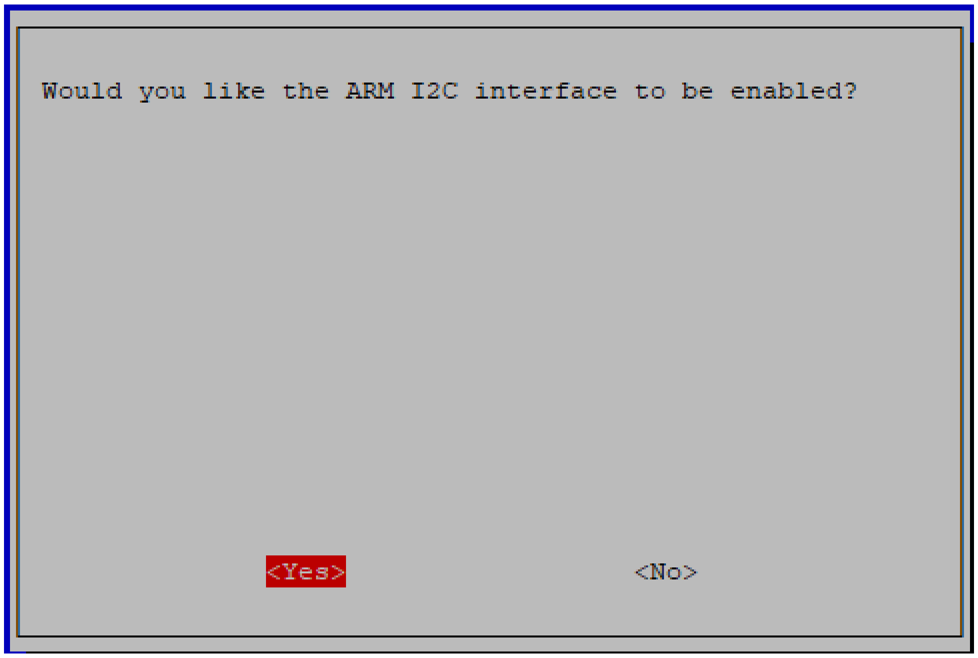
```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

              <Select>                     <Finish>
```

P5 I2C



```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera          Enable/Disable connection to the
P2 SSH             Enable/Disable remote command lin
P3 VNC             Enable/Disable graphical remote a
P4 SPI             Enable/Disable automatic loading
P5 I2C             Enable/Disable automatic loading
P6 Serial          Enable/Disable shell and kernel m
P7 1-Wire          Enable/Disable one-wire interface
P8 Remote GPIO     Enable/Disable remote access to G

        <Select>                <Back>
```

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



```
Would you like the ARM I2C interface to be enabled?

        <Yes>                <No>
```

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).

```
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.

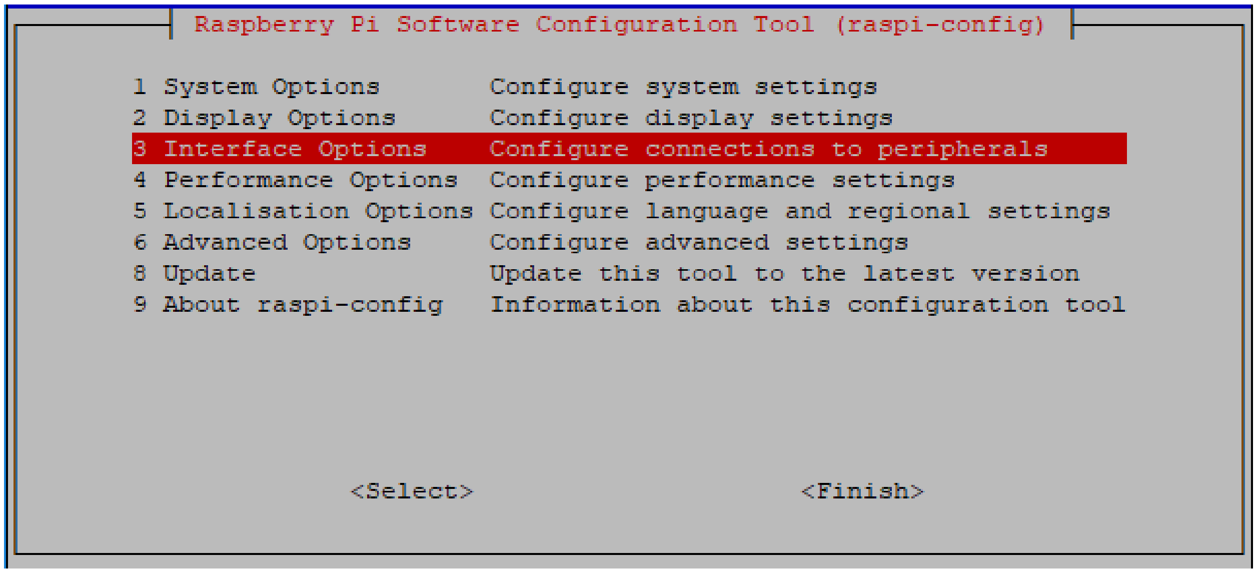
```
sudo pip3 install smbus2
```

7.2 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).

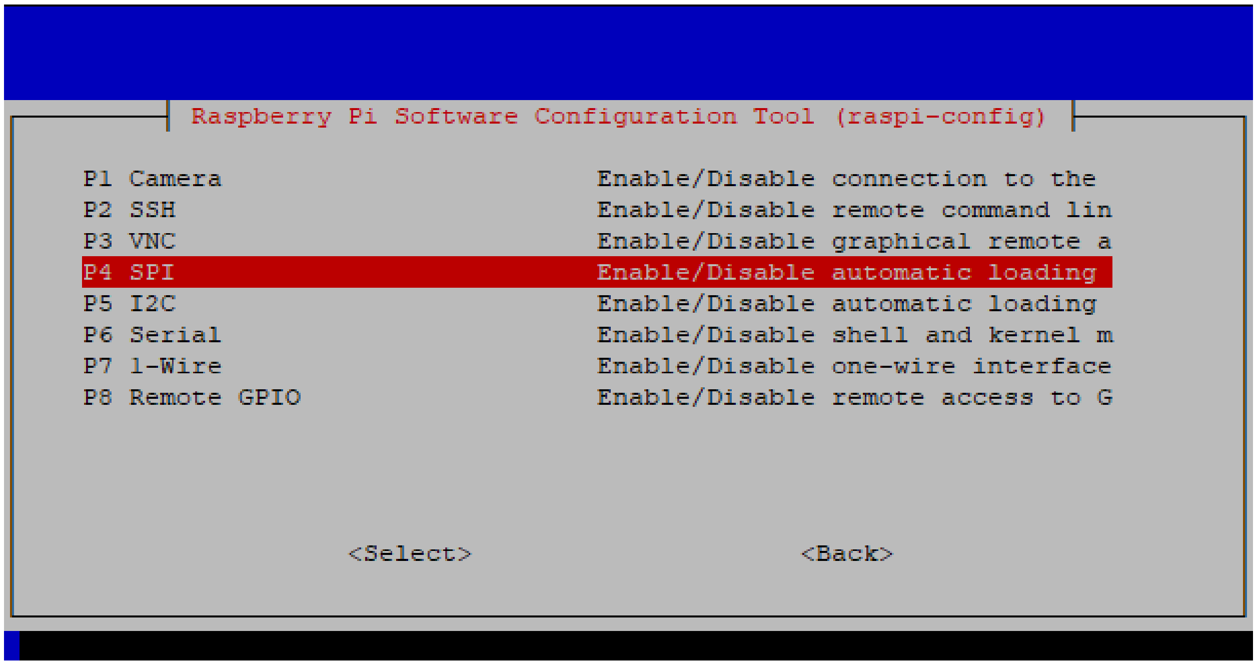
```
sudo raspi-config
```

3 Interfacing options



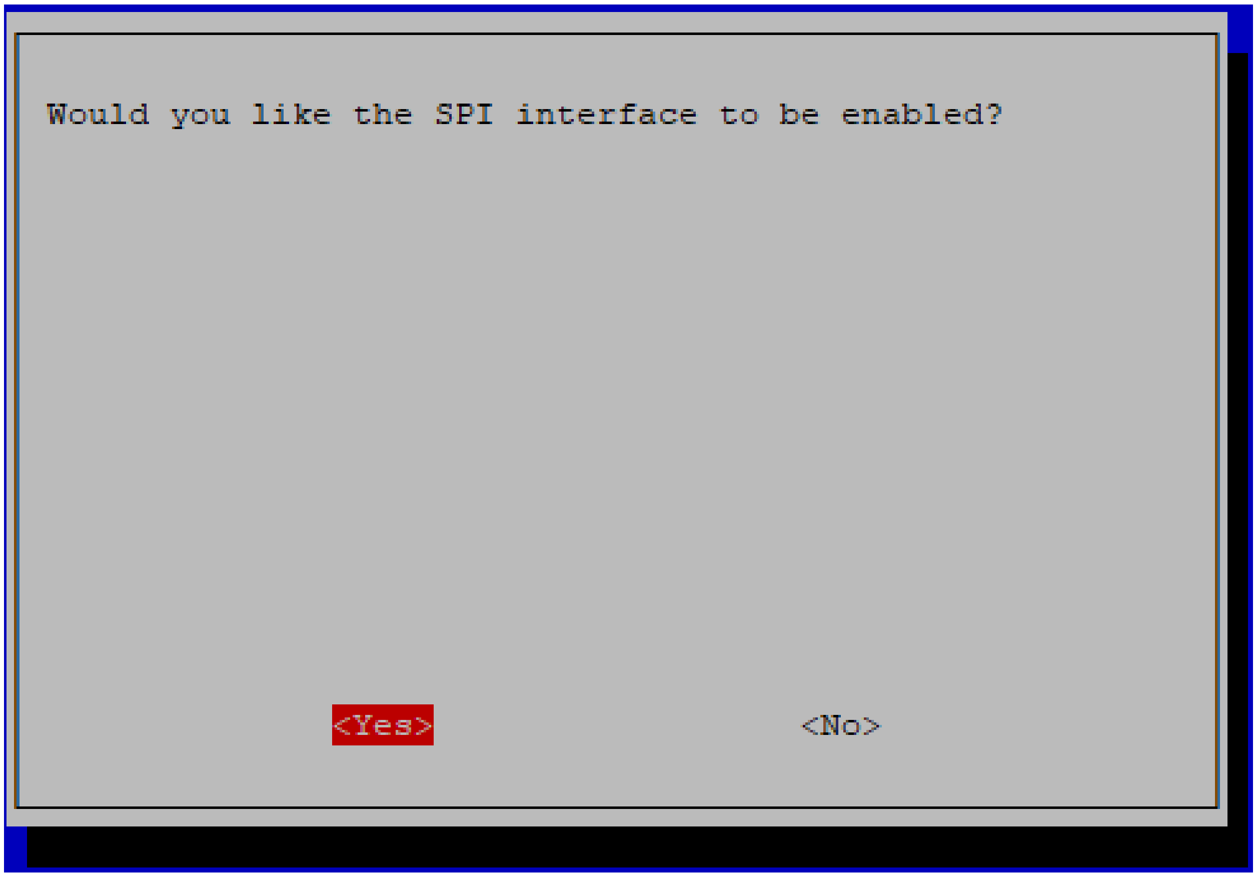
```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options        Configure system settings
2 Display Options       Configure display settings
3 Interface Options     Configure connections to peripherals
4 Performance Options   Configure performance settings
5 Localisation Options  Configure language and regional settings
6 Advanced Options      Configure advanced settings
8 Update                Update this tool to the latest version
9 About raspi-config    Information about this configuration tool

              <Select>                    <Finish>
```

P4 SPI



```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera                         Enable/Disable connection to the
P2 SSH                            Enable/Disable remote command lin
P3 VNC                            Enable/Disable graphical remote a
P4 SPI                            Enable/Disable automatic loading
P5 I2C                            Enable/Disable automatic loading
P6 Serial                         Enable/Disable shell and kernel m
P7 1-Wire                         Enable/Disable one-wire interface
P8 Remote GPIO                    Enable/Disable remote access to G

              <Select>                    <Back>
```

<YES>, then click <OK> and <Finish>.



```
Would you like the SPI interface to be enabled?
```

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.

7.3 Remote Desktop

There are two ways to control the desktop of the Raspberry Pi remotely:

VNC and **XRDP**, you can use any of them.

7.3.1 VNC

You can use the function of remote desktop through VNC.

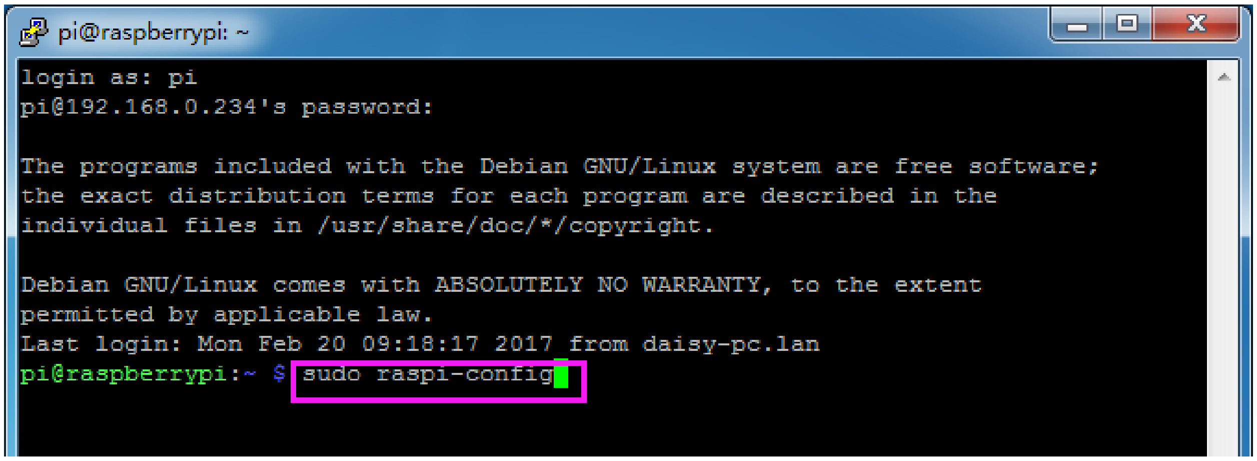
Enable VNC service

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:

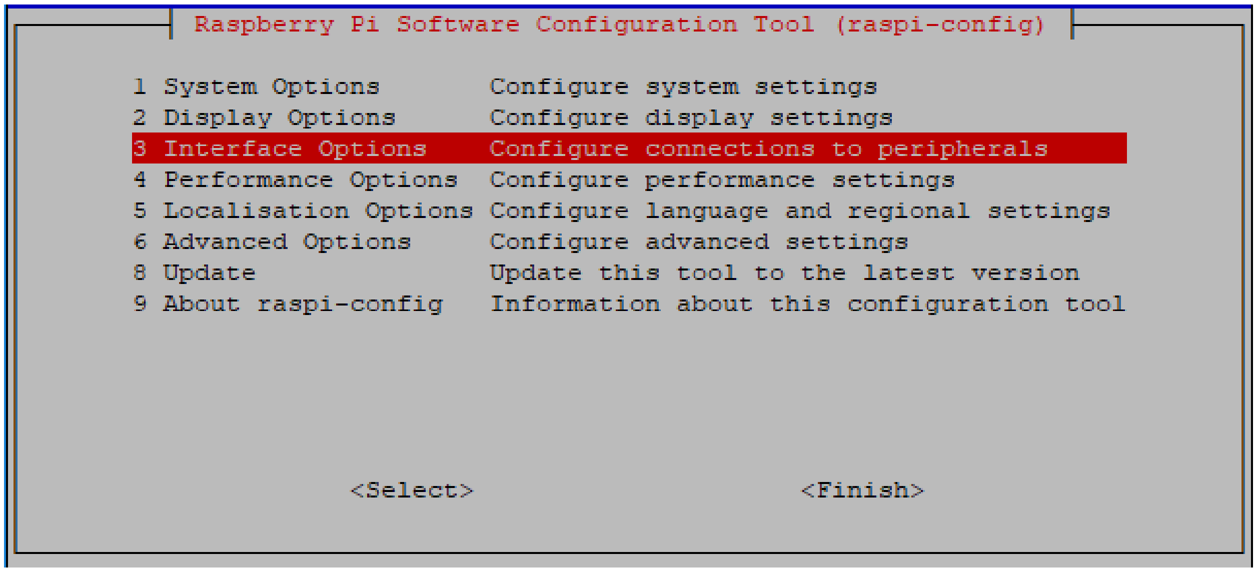
```
sudo raspi-config
```



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

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

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

Step 2

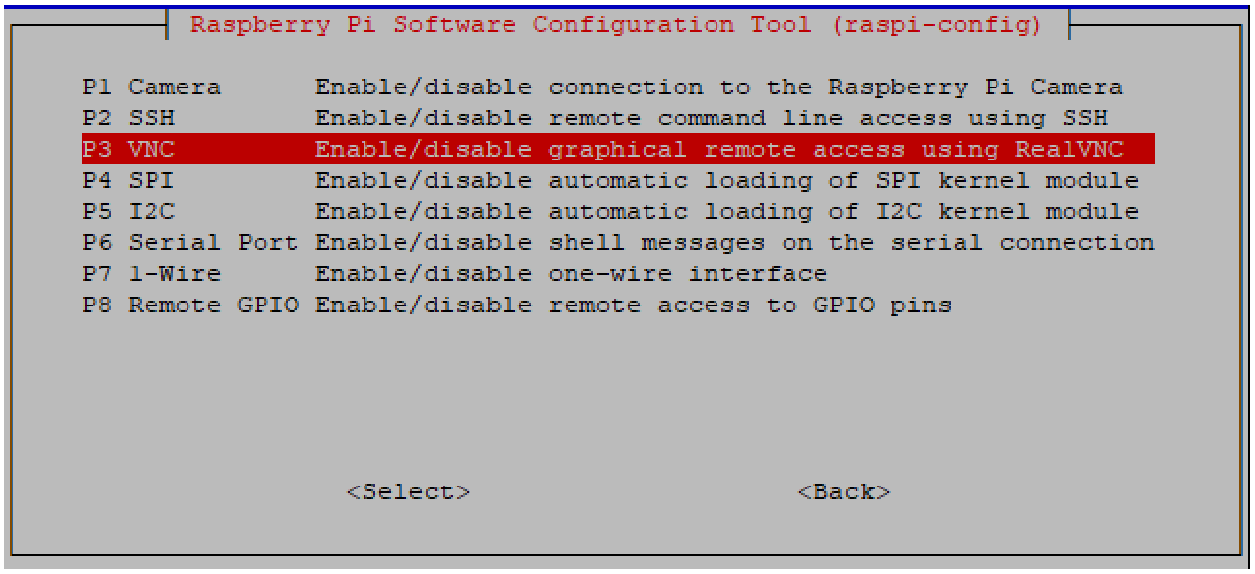
Choose **3 Interfacing Options** by press the down arrow key on your keyboard, then press the **Enter** key.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

              <Select>                     <Finish>
```

Step 3

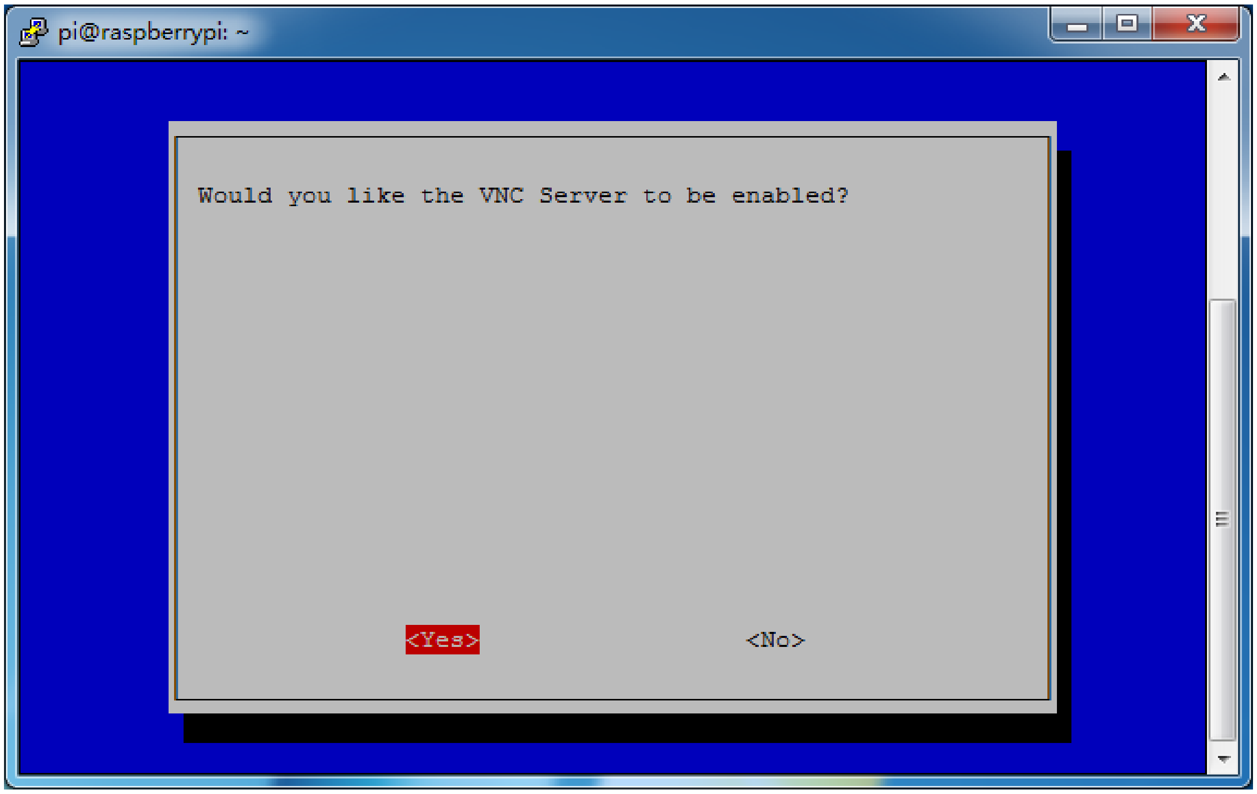
P3 VNC



```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/disable connection to the Raspberry Pi Camera
P2 SSH         Enable/disable remote command line access using SSH
P3 VNC         Enable/disable graphical remote access using RealVNC
P4 SPI         Enable/disable automatic loading of SPI kernel module
P5 I2C         Enable/disable automatic loading of I2C kernel module
P6 Serial Port Enable/disable shell messages on the serial connection
P7 1-Wire      Enable/disable one-wire interface
P8 Remote GPIO Enable/disable remote access to GPIO pins

                 <Select>                    <Back>
```

Step 4

Select **Yes -> OK -> Finish** to exit the configuration.



```
pi@raspberrypi: ~

Would you like the VNC Server to be enabled?

          <Yes>                 <No>
```

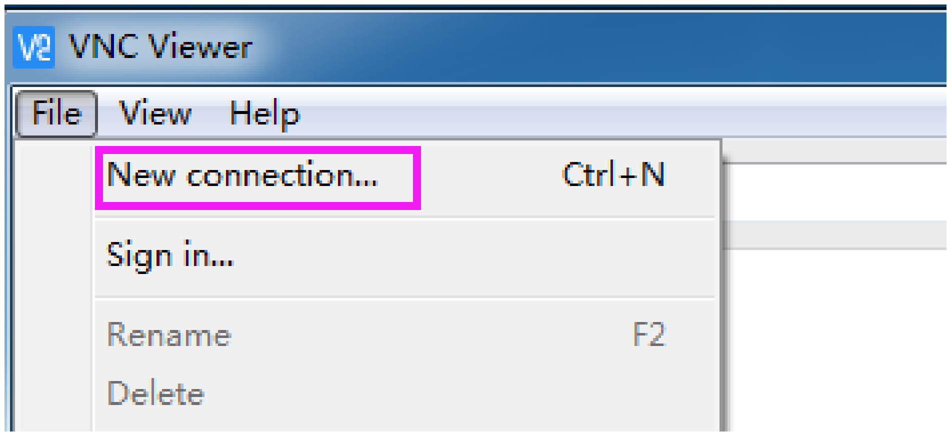
Login to VNC

Step 1

You need to download and install the VNC Viewer on personal computer. After the installation is done, open it.

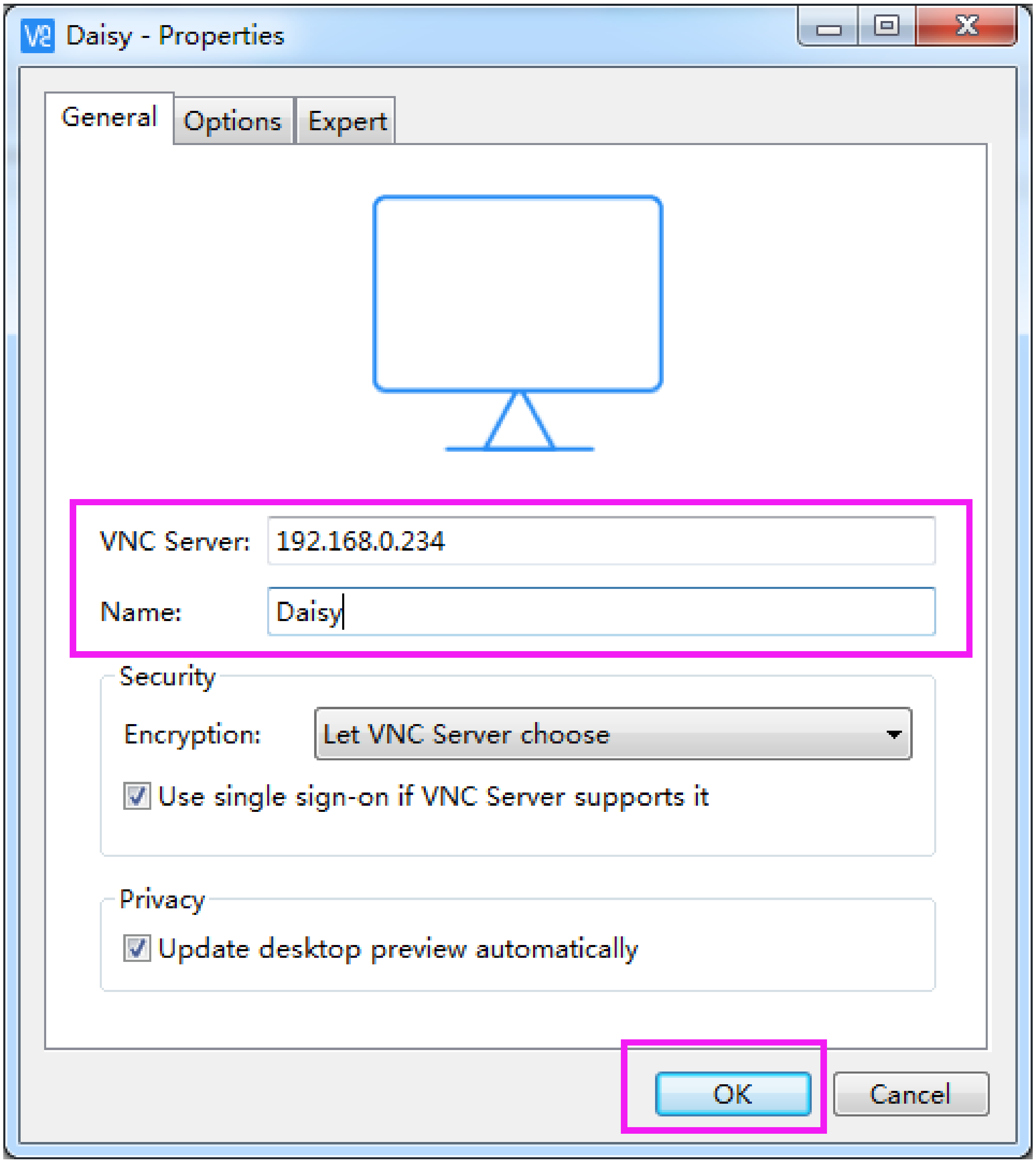
Step 2

Then select "**New connection**".



Step 3

Input IP address of Raspberry Pi and any **Name**.



VNC Server: 192.168.0.234

Name: Daisy

Security

Encryption: Let VNC Server choose

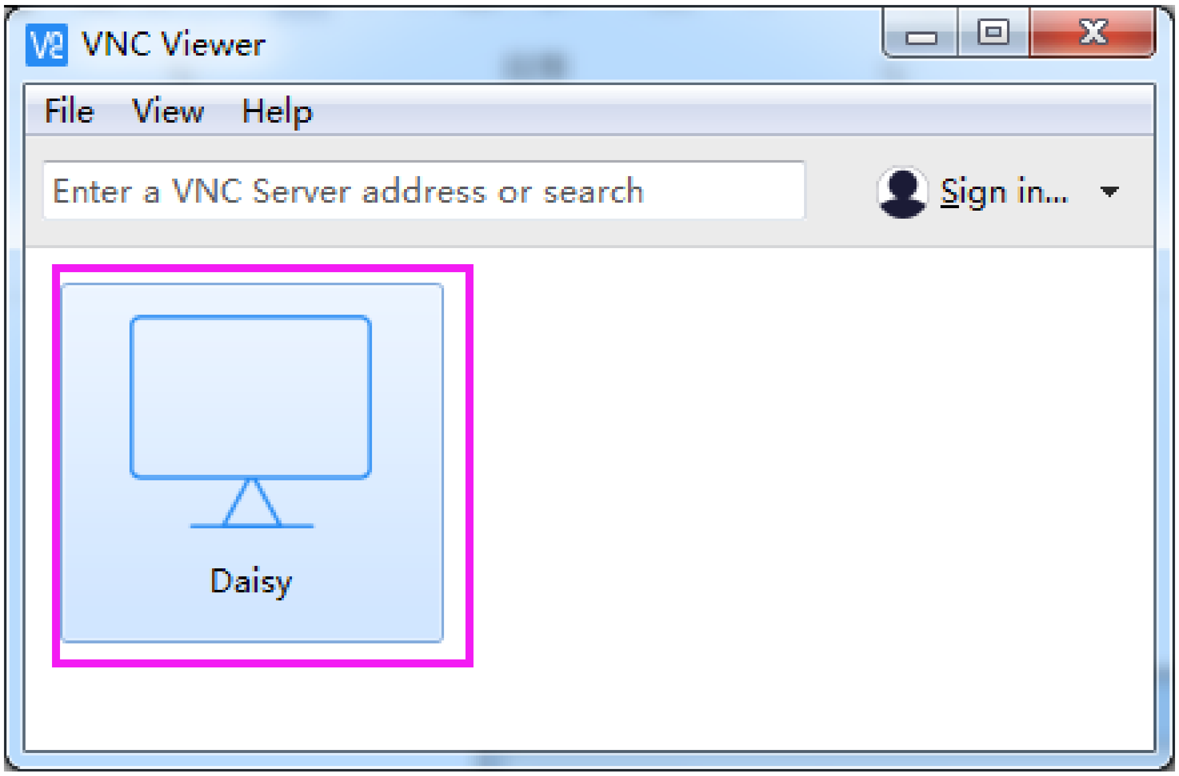
☑ Use single sign-on if VNC Server supports it

Privacy

☑ Update desktop preview automatically

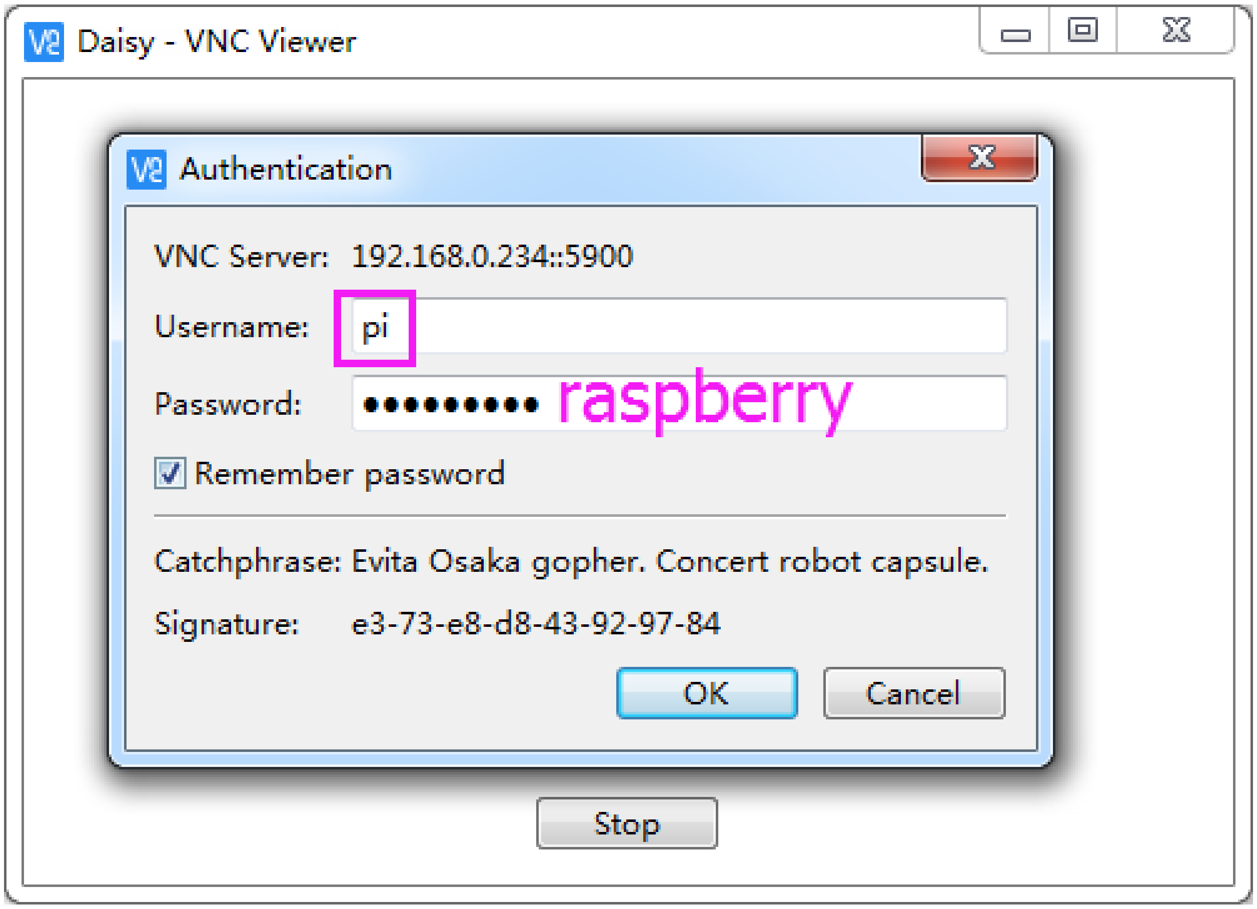
Step 4

Double click the **connection** just created:



Step 5

Enter Username (**pi**) and Password (**raspberry** by default).



VNC Server: 192.168.0.234::5900

Username: pi

Password: ●●●●●●●●● raspberry

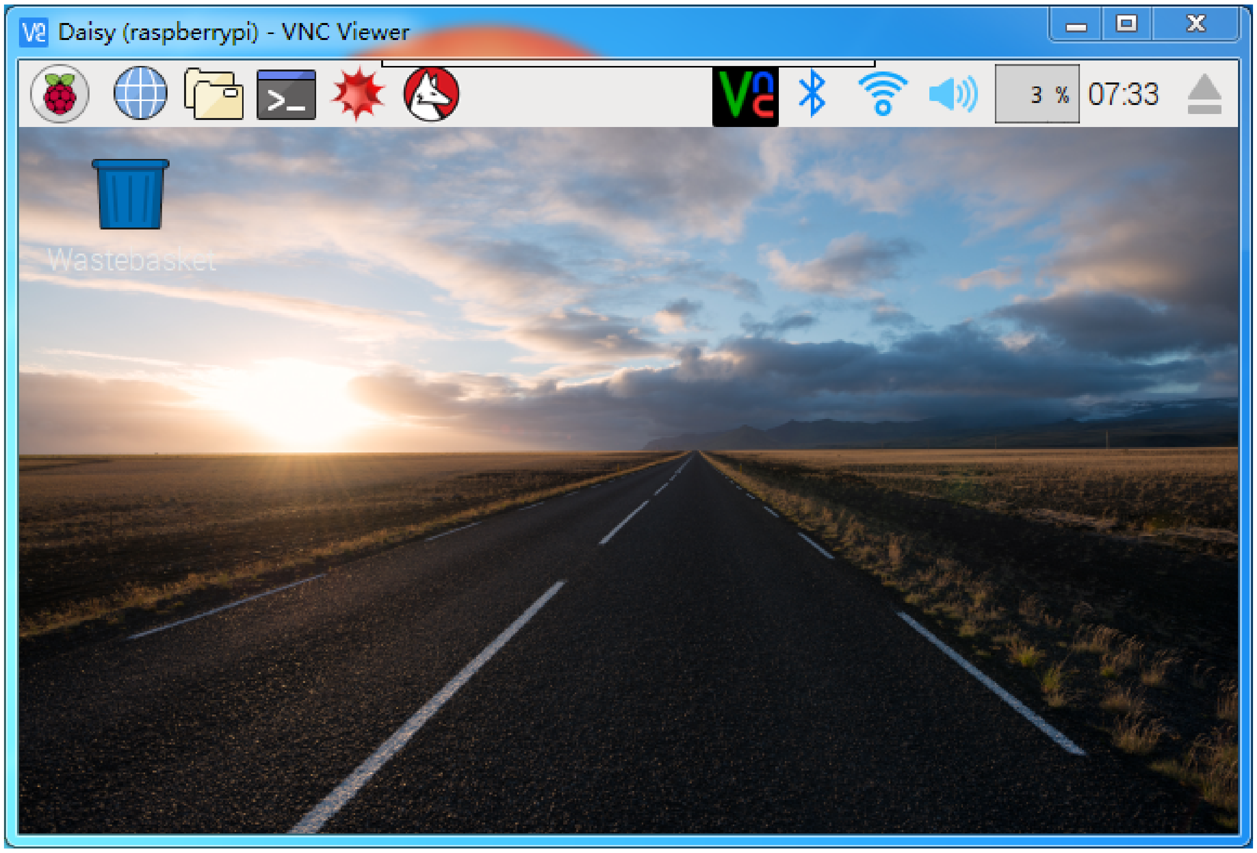
☑ Remember password

Catchphrase: Evita Osaka gopher. Concert robot capsule.

Signature: e3-73-e8-d8-43-92-97-84

Step 6

Now you can see the desktop of the Raspberry Pi:



That's the end of the VNC part.

7.3.2 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

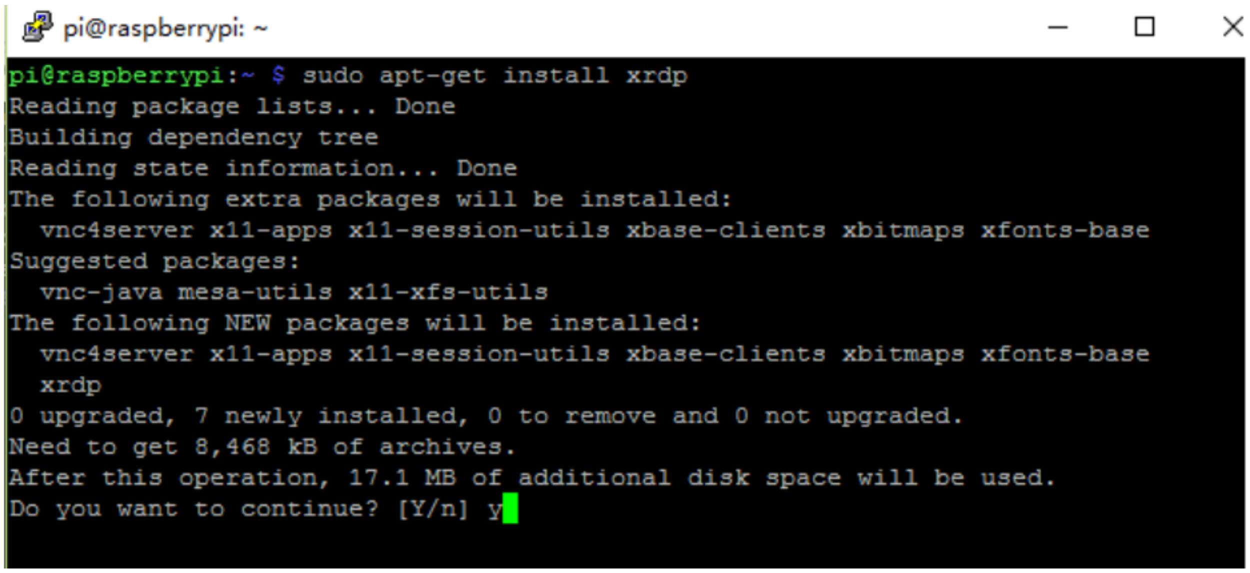
Input the following instructions to install XRDP.

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

Step 3

Later, the installation starts.

Enter "Y", press key "Enter" to confirm.



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

Step 4

Finished the installation, you should login to your Raspberry Pi by using Windows remote desktop applications.

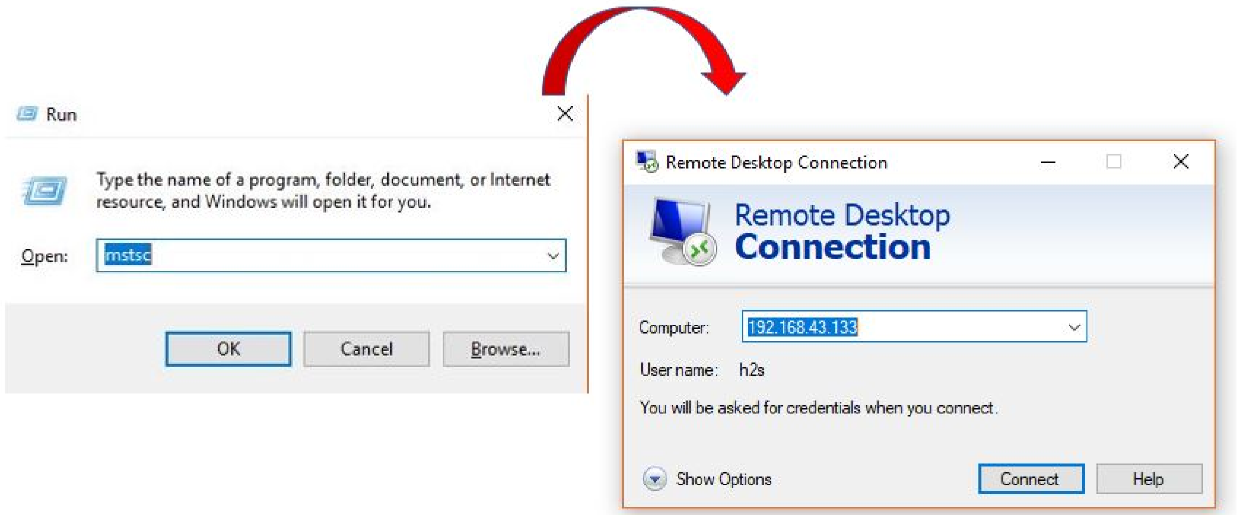
Login to XRDP

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".



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: mstsc

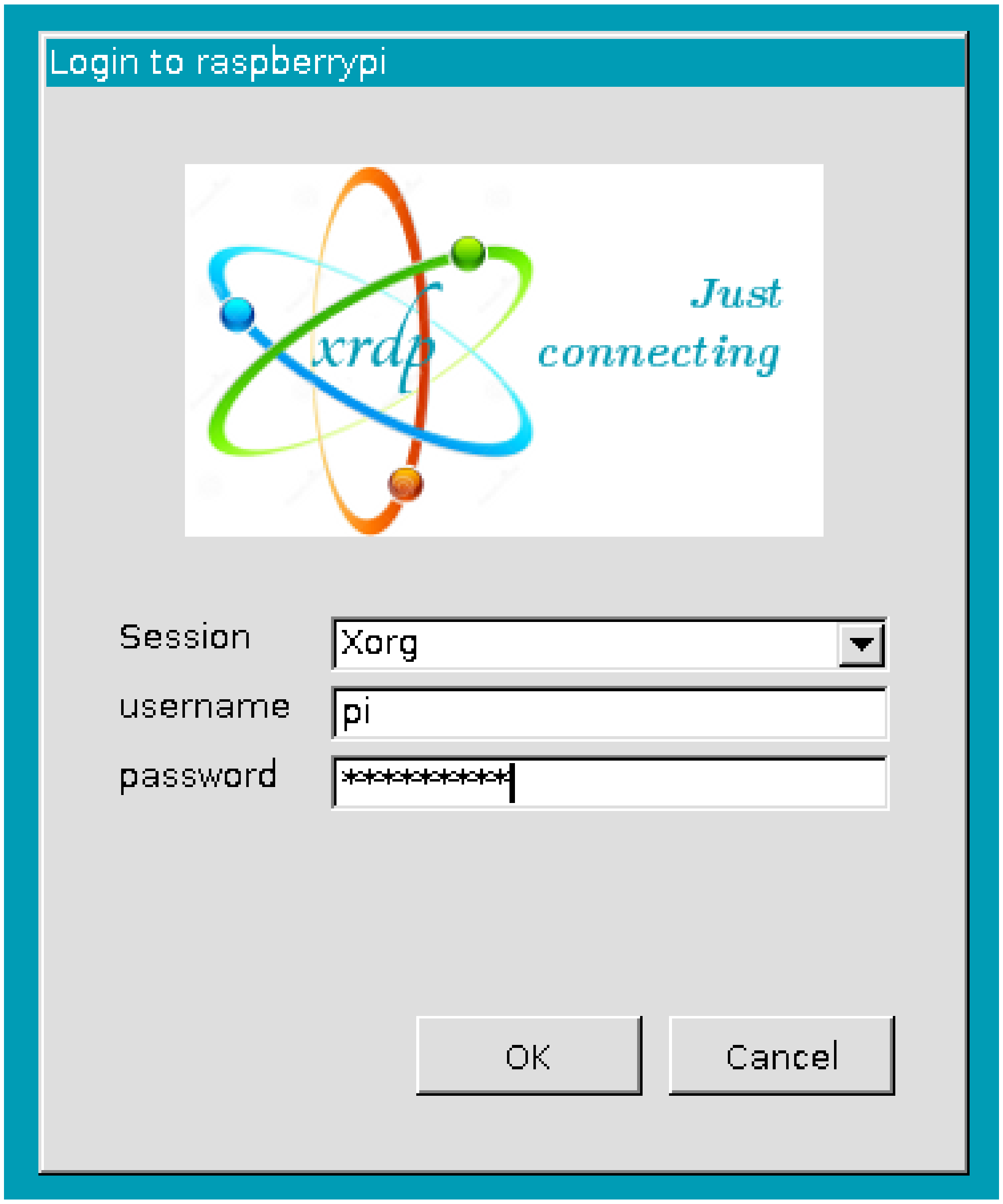
Computer: 192.168.43.133

User name: h2s

You will be asked for credentials when you 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".


Login to raspberrypi

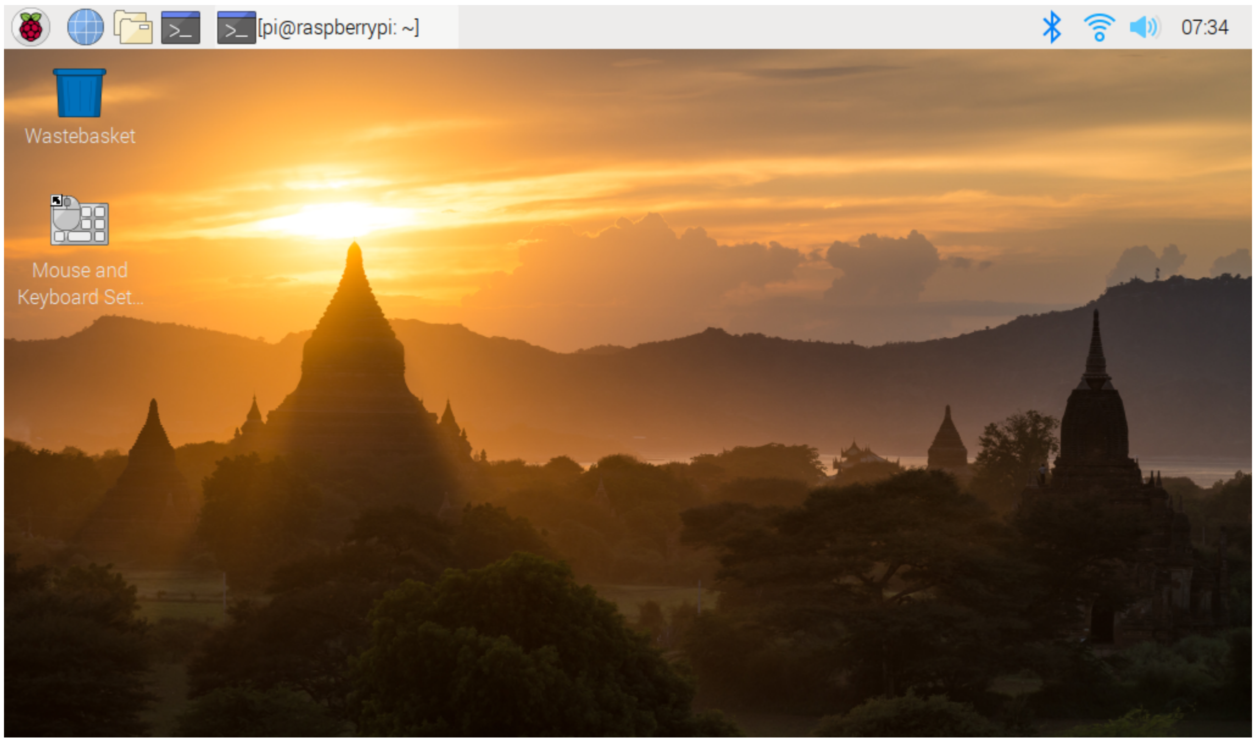
Session Xorg

username pi

password **********

Step 4

Here, you successfully login to RPi by using the remote desktop.


[pi@raspberrypi: ~]
07:34
Wastebasket
Mouse and Keyboard Set...

CHAPTER

EIGHT

FAQ

8.1 C code is not working?

- Check your wiring for problems.
- Check if the code is reporting errors, if so, refer to: *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 *WiringPi* to manually upgrade it to version 2.52.

CHAPTER

NINE

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.

CHAPTER

TEN

COPYRIGHT NOTICE