
SunFounder Super_Kit_V3.0_for_Raspberry_Pi

www.sunfounder.com

Sep 21, 2022

CONTENTS

| | | |
|----------|--|------------|
| 1 | Components List | 3 |
| 2 | Preparation | 11 |
| 2.1 | What Do We Need? | 11 |
| 2.2 | Installing the OS | 13 |
| 2.3 | Set up Your Raspberry Pi | 19 |
| 3 | Assemble the Raspberry Pi to RAB Holder | 27 |
| 3.1 | T_Extension Board | 27 |
| 3.2 | RAB Holder | 29 |
| 4 | Libraries | 33 |
| 4.1 | RPi.GPIO | 33 |
| 4.2 | WiringPi | 34 |
| 5 | Download the Code | 37 |
| 6 | Lessons | 39 |
| 6.1 | Lesson 1 Blinking LED | 39 |
| 6.2 | Lesson 2 Controlling an LED by a Button | 53 |
| 6.3 | Lesson 3 Flowing LED Lights | 61 |
| 6.4 | Lesson 4 Breathing LED | 69 |
| 6.5 | Lesson 5 RGB LED | 77 |
| 6.6 | Lesson 6 Buzzer | 85 |
| 6.7 | Lesson 7 Relay | 91 |
| 6.8 | Lesson 8 4N35 | 98 |
| 6.9 | Lesson 9 Ne555 | 104 |
| 6.10 | Lesson 10 Slide Switch | 110 |
| 6.11 | Lesson 11 How to Drive a DC Motor | 118 |
| 6.12 | Lesson 12 Rotary Encoder | 128 |
| 6.13 | Lesson 13 Driving LEDs by 74HC595 | 138 |
| 6.14 | Lesson 14 Driving 7-Segment Display by 74HC595 | 148 |
| 6.15 | Lesson 15 Driving Dot-Matrix by 74HC595 | 163 |
| 6.16 | Lesson 16 LCD1602 | 176 |
| 6.17 | Lesson 17 ADXL345 | 188 |
| 7 | Appendix | 199 |
| 7.1 | I2C Configuration | 199 |
| 7.2 | SPI Configuration | 201 |
| 7.3 | Remote Desktop | 204 |
| 7.4 | Components Introduction | 213 |

| | | |
|-----------|----------------------------------|------------|
| 8 | FAQ | 221 |
| 8.1 | C code is not working? | 221 |
| 9 | Thank You | 223 |
| 10 | Copyright Notice | 225 |

About the Super Kit 3.0

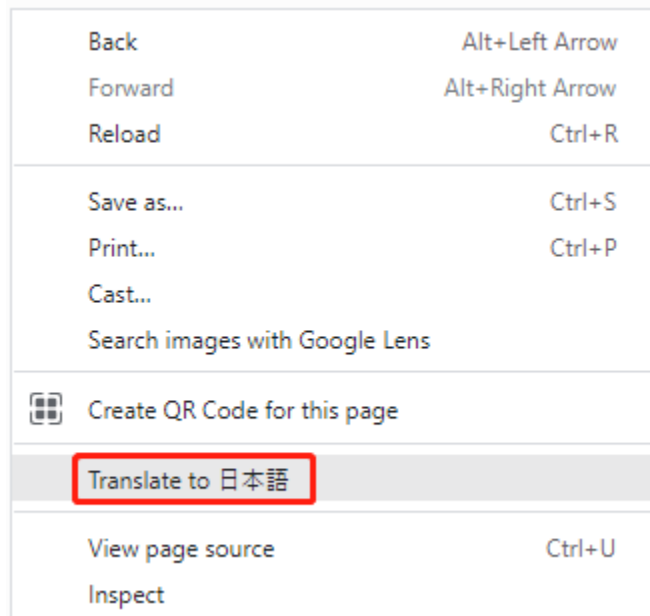
This super kit is suitable for the Raspberry Pi B, model B+ 2 model B3 Model B/B+ and 4 Model B. It includes various components and chips that can show different interesting phenomena. You can make it happen by following the experiment instructions, and learn basic knowledge about them. Also you can explore more application after mastering the principle and code. Now get on the road!

About the display language

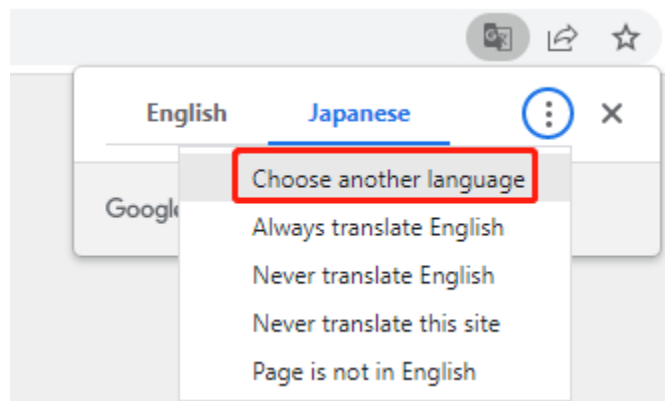
In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

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.



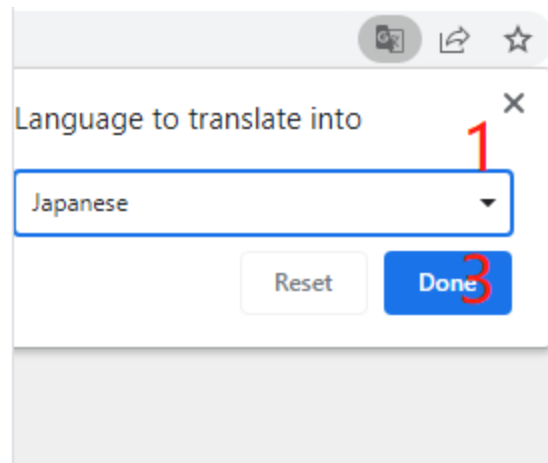
- There will be a language popup in the upper right corner. Click on the menu button to **choose another language**.



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

Arabic
Armenian
Azerbaijani
Bangla
Basque
Belarusian
Bosnian
Bulgarian
Burmese
Catalan

2



The screenshot shows a web browser window with a light gray header bar containing three icons: a document, a share icon, and a star. Below the header, a white dialog box is open. The dialog has a title bar with a close button (X) in the top right corner. The main text inside the dialog is "Language to translate into". Below this text is a dropdown menu with a blue border and a small downward arrow on the right. The word "Japanese" is displayed inside the dropdown. To the right of the dropdown, a large red number "1" is overlaid. Below the dropdown are two buttons: a white "Reset" button and a blue "Done" button. A large red number "3" is overlaid on the "Done" button.

COMPONENTS LIST

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




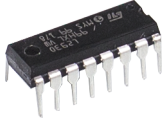

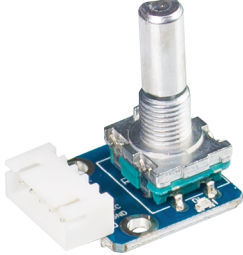
| No. | Name | Quantity | Component |
|-----|--------------------------|----------|--|
| 1 | 555 Timer IC | 1 |  |
| 2 | Optocoupler (4N35) | 2 |  |
| 3 | Shift Register (74HC595) | 2 |  |
| 4 | L293D | 1 |  |
| 5 | Accelerometer ADXL345 | 1 |  |
| 6 | Rotary Encoder | 1 |  |

Table 1 – continued from previous page











| | | | |
|----|-----------------|---|---|
| 7 | Potentiometer | 1 |  |
| 8 | Resistor (220) | 8 |  |
| 9 | Resistor (1k) | 8 |  |
| 10 | Resistor (10k) | 4 |  |
| 11 | Resistor | 4 |  |
| 12 | Resistor (1M) | 1 |  |
| 13 | Resistor | 1 |  |
| 14 | Diode Rectifier | 4 |  |
| 15 | Switch | 1 |  |
| 16 | Button | 5 |  |

Table 1 – continued from previous page

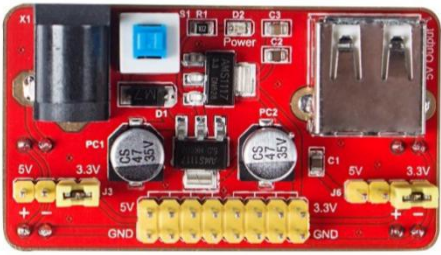

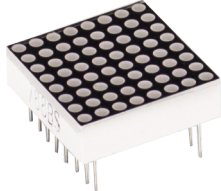
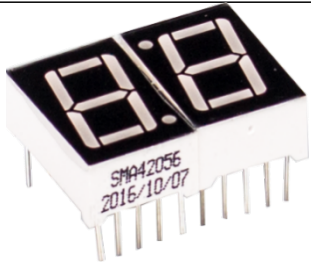
| | | | |
|----|--------------|---|--|
| 17 | Power Supply | 1 |  |
| 18 | LCD1602 | 1 |  |
| 19 | Dot Matrix | 1 |  |
| 20 | 7-Segment | 2 |  |

Table 1 – continued from previous page










| | | | |
|----|-------------------------|---|--|
| 21 | DC Motor | 1 |  |
| 22 | RGB LED | 1 |  |
| 23 | LED (red) | 8 |  |
| 24 | LED (white) | 4 |  |
| 25 | LED (green) | 4 |  |
| 26 | LED (yellow) | 4 |  |
| 27 | NPN Transistor (S8050) | 2 |  |
| 28 | PNP Transistor (S8550) | 2 |  |
| 29 | Capacitor Ceramic 100nF | 4 |  |

Table 1 – continued from previous page


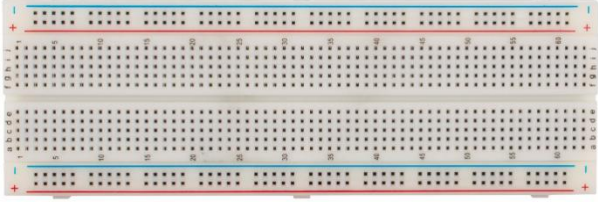



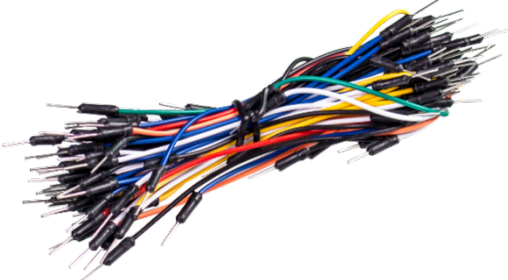
| | | | |
|----|--------------------------|----|--|
| 30 | Capacitor Ceramic 10nF | 4 |  |
| 31 | Breadboard | 1 |  |
| 32 | Active Buzzer | 1 |  |
| 33 | Relay | 1 |  |
| 34 | Fan | 1 |  |
| 35 | Male-to-Male Jumper Wire | 65 |  |

Table 1 – continued from previous page

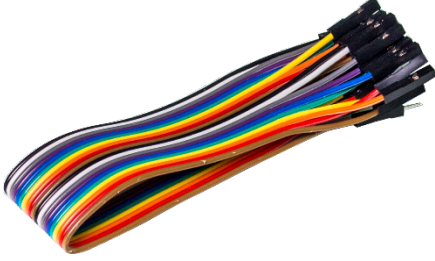





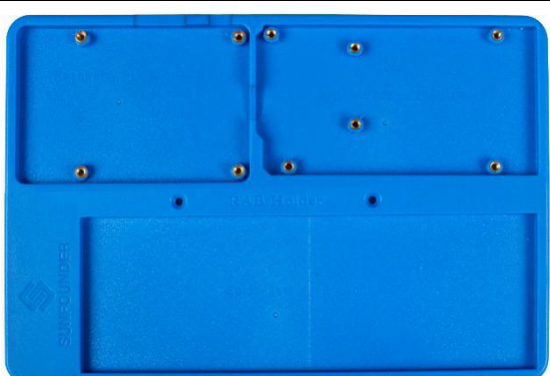
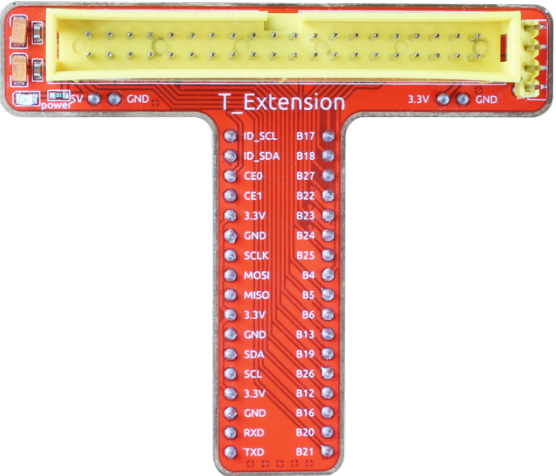

| | | | |
|----|----------------------------|----|--|
| 36 | Female-to-Male Dupont Wire | 20 |  |
| 37 | 5-Pin Anti-reverse Cable | 2 |  |
| 38 | 9V Battery Buckle | 1 |  |
| 39 | M3*10 Screw | 2 |  |
| 40 | M2.5*6 Screw | 4 |  |
| 41 | M3*6 Screw | 6 |  |
| 42 | RAB Holder | 1 |  |

Table 1 – continued from previous page

| | | | |
|----|-------------|---|---|
| 43 | T-Extension | 1 |  |
| 44 | 40-Pin GPIO | 1 |  |

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: [raspberrypi-setting-up](https://www.raspberrypi.org/documentation/hardware/raspberrypi/setting-up.md).

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

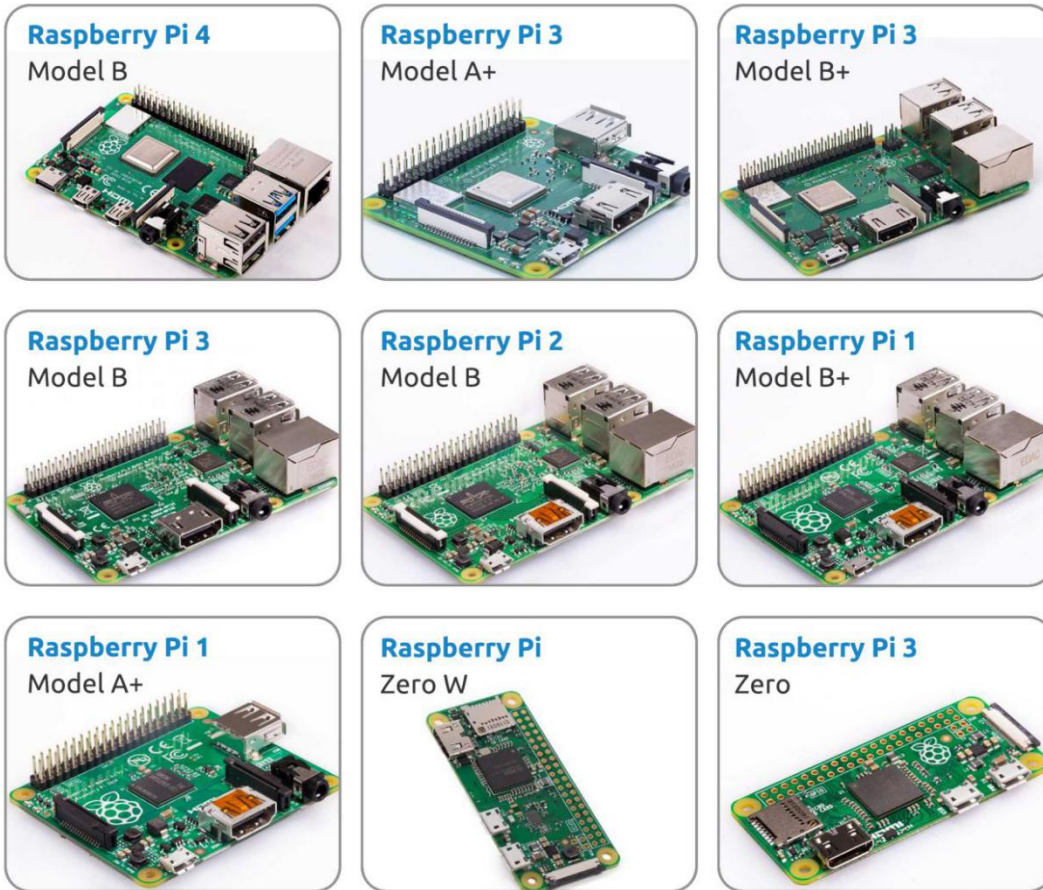
2.1 What Do We Need?

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

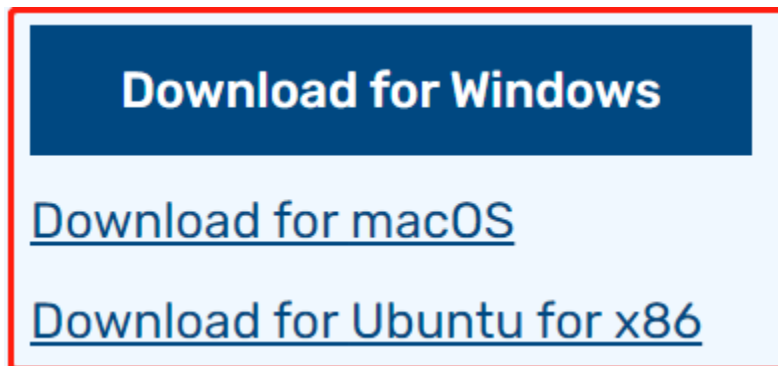
Required Components

| | |
|-------------------|-----------------------|
| 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.

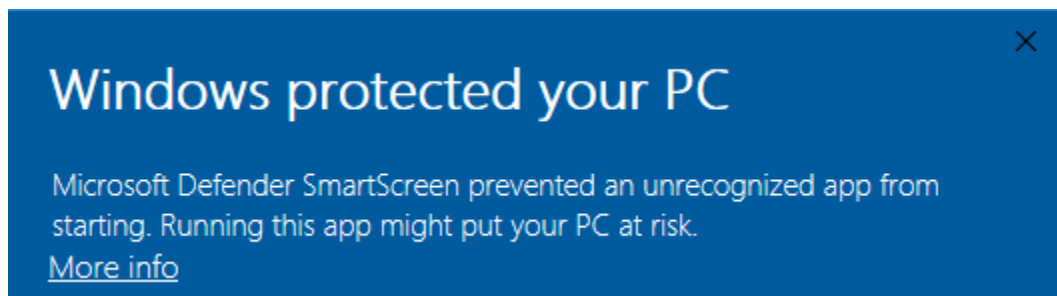
Visit the download page: <https://www.raspberrypi.org/software/>. Click on the link for the Raspberry Pi Imager that matches your operating system, when the download finishes, click it to launch the installer.



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.

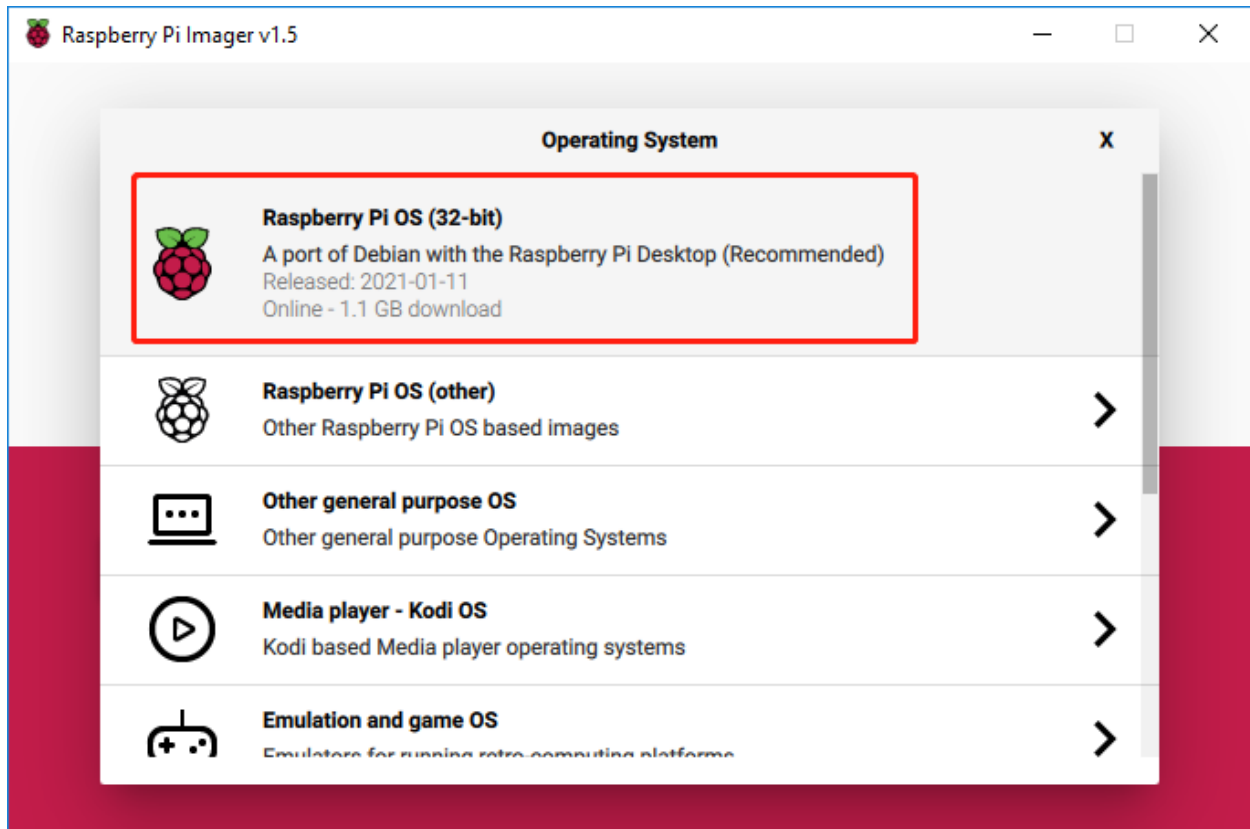


Step 3

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.

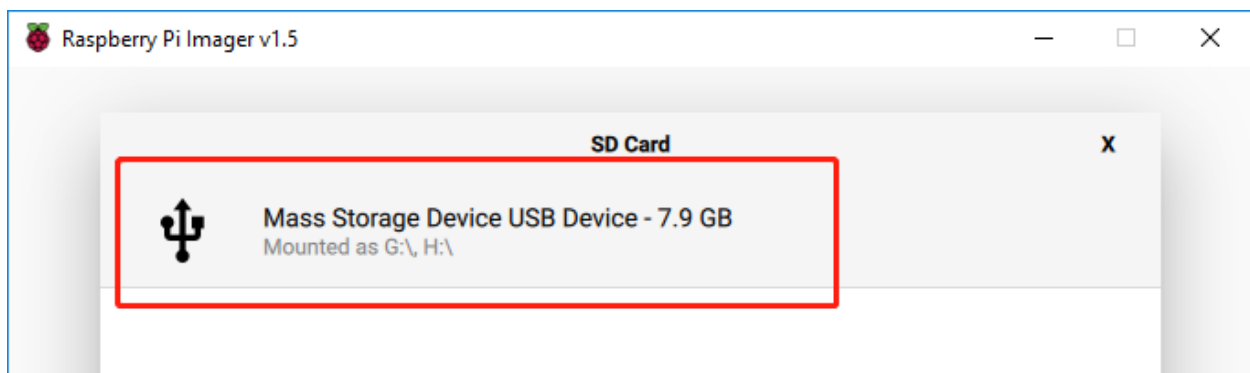


Note:

- 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”.
-

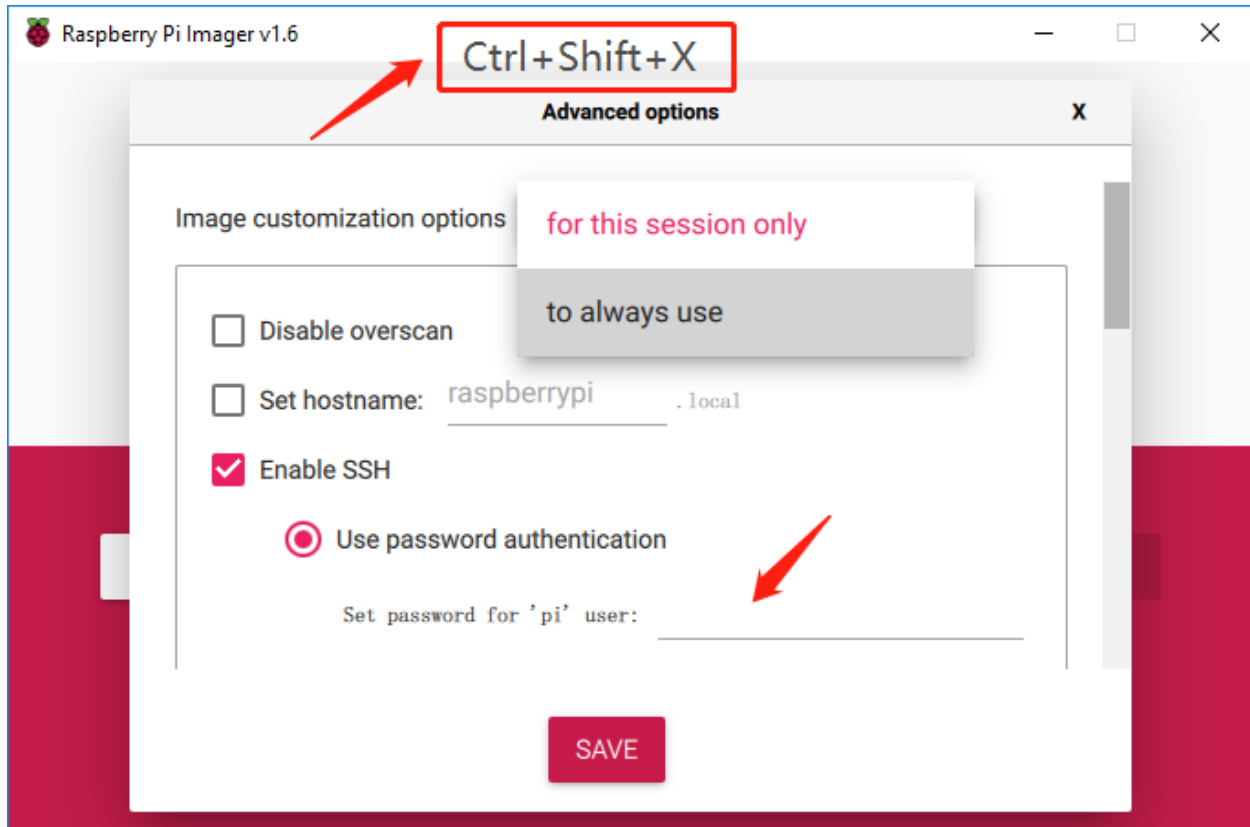
Step 5

Select the SD card you are using.



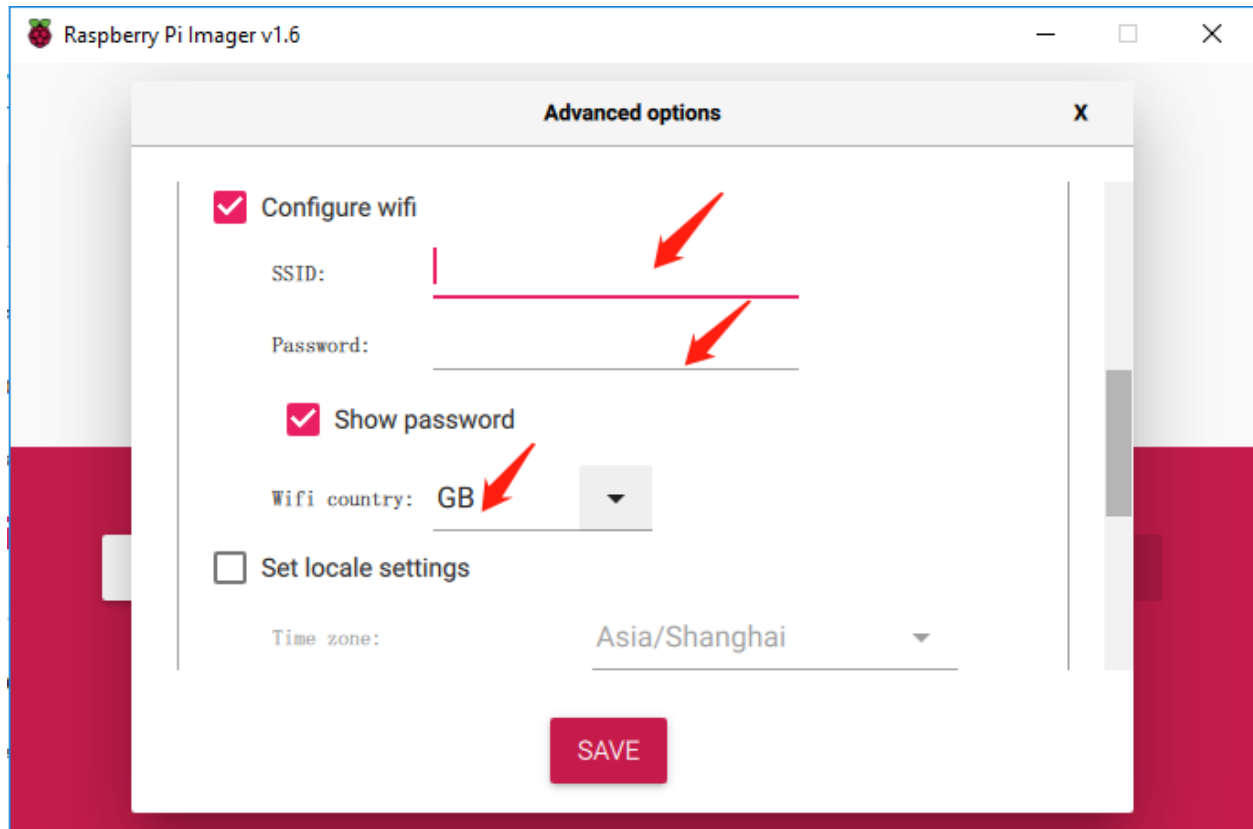
Step 6

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



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

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



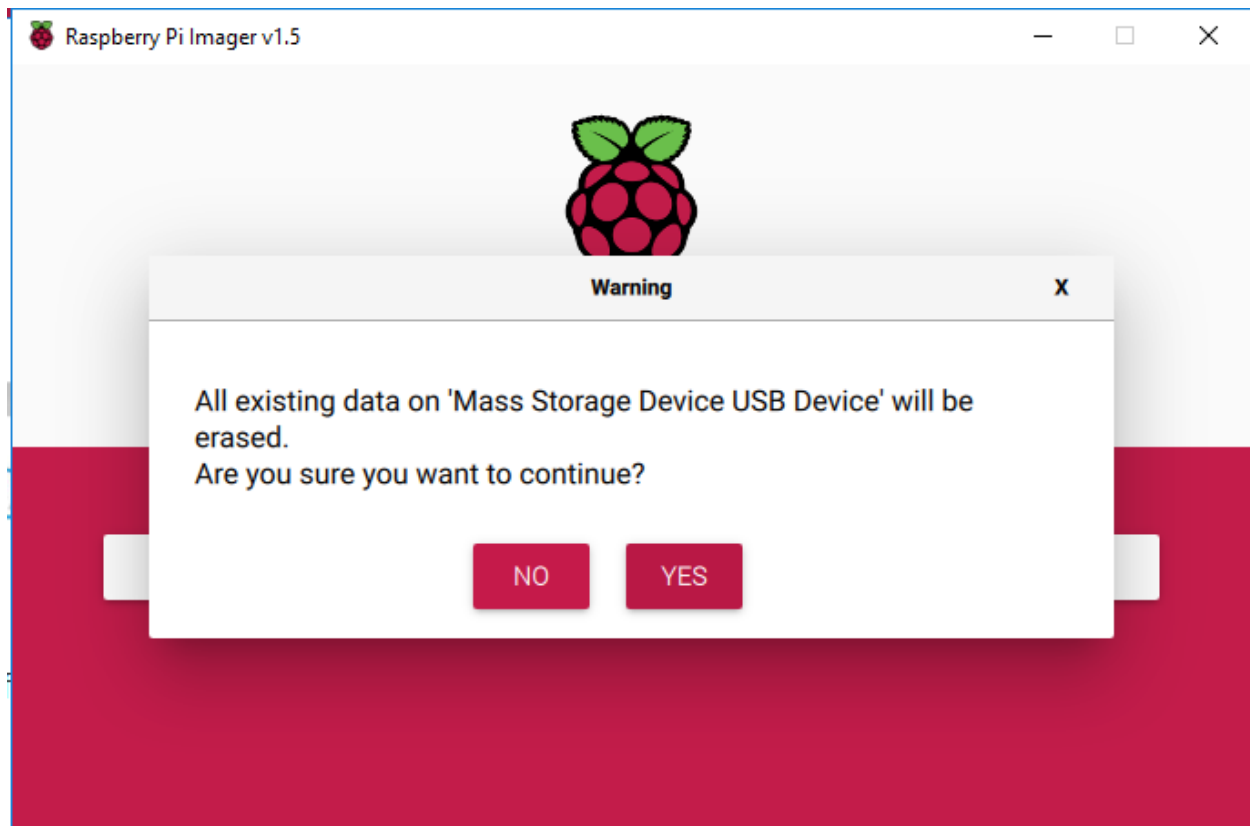
Step 7

Click the **WRITE** button.



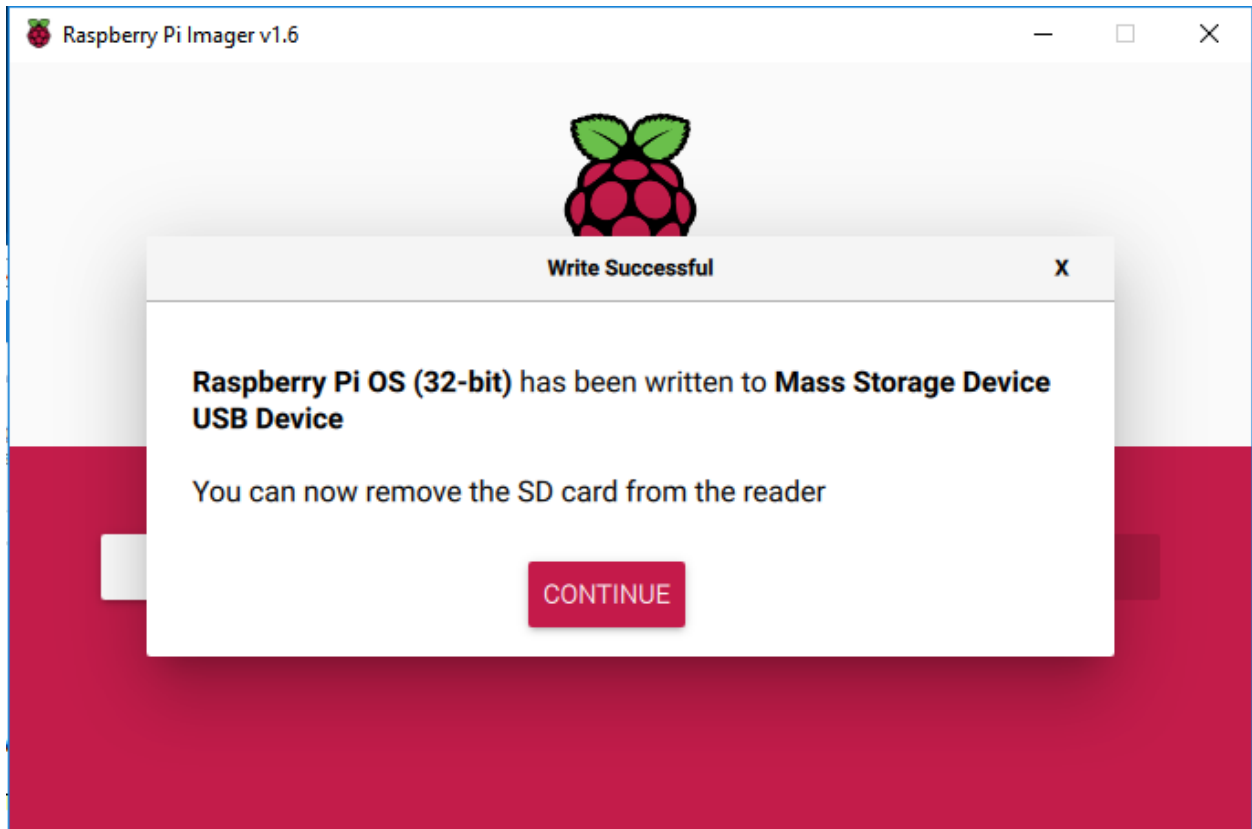
Step 8

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



Step 9

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



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.

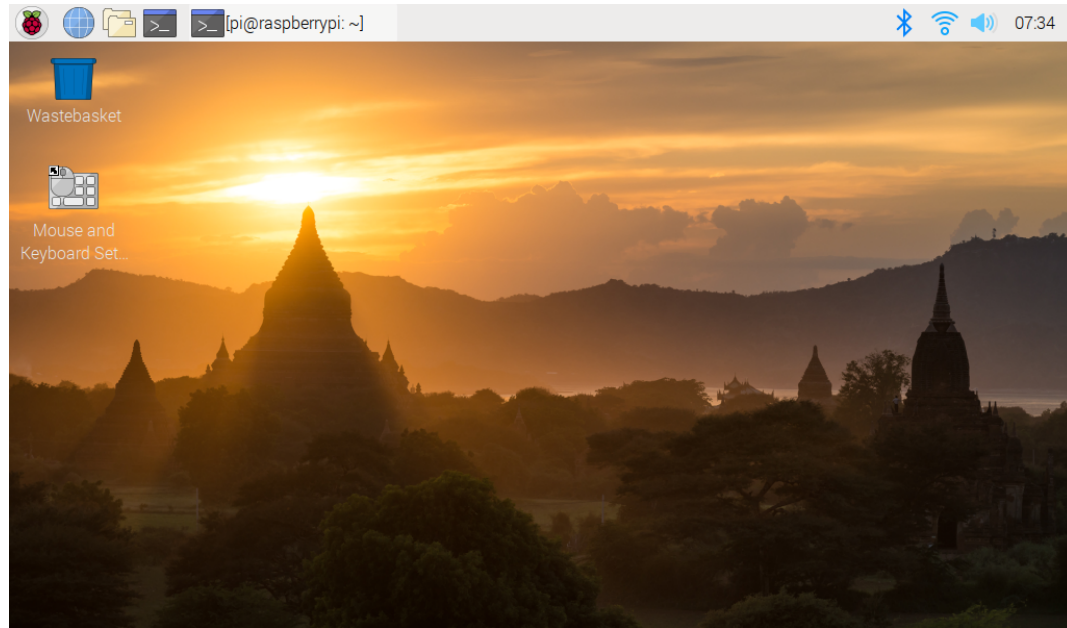
Required Components

| | |
|-------------------|--------------------------|
| 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.

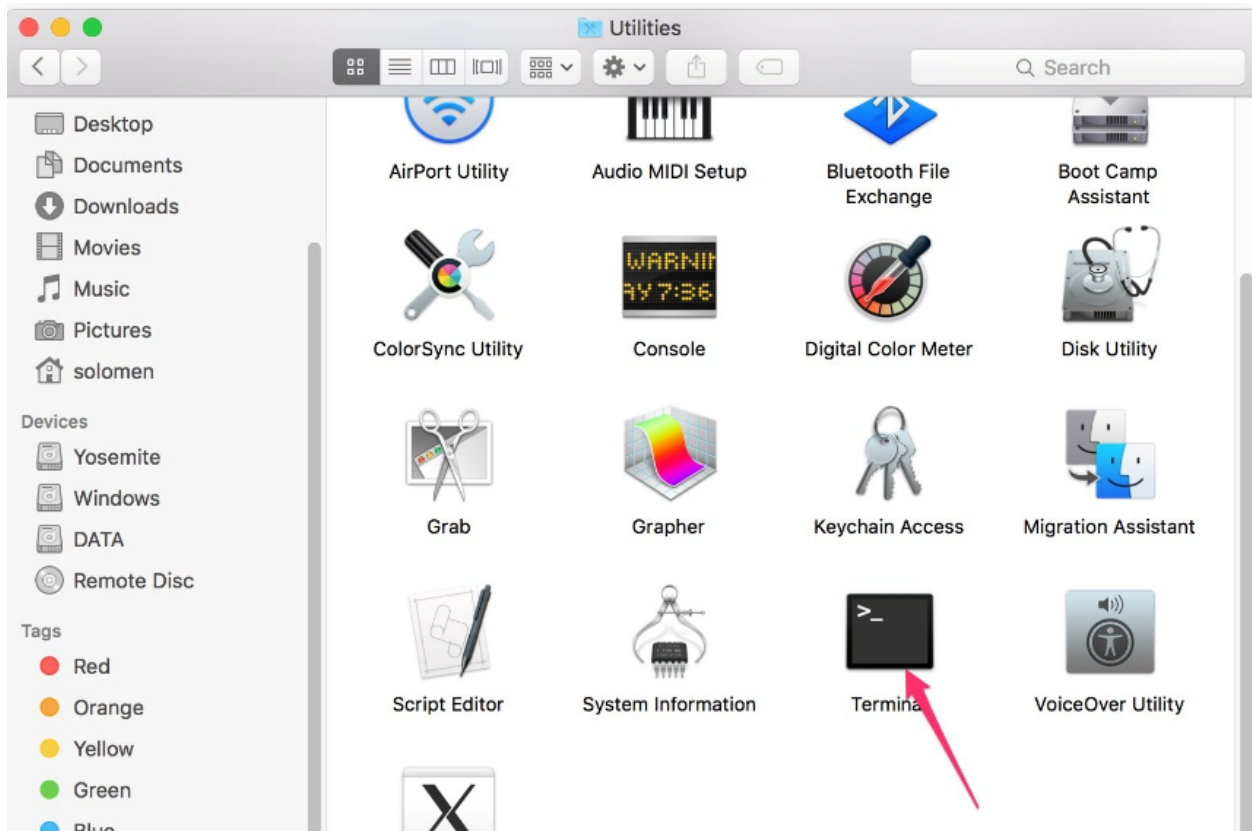
Use the SSH Remote Control

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”.

```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

Step 4

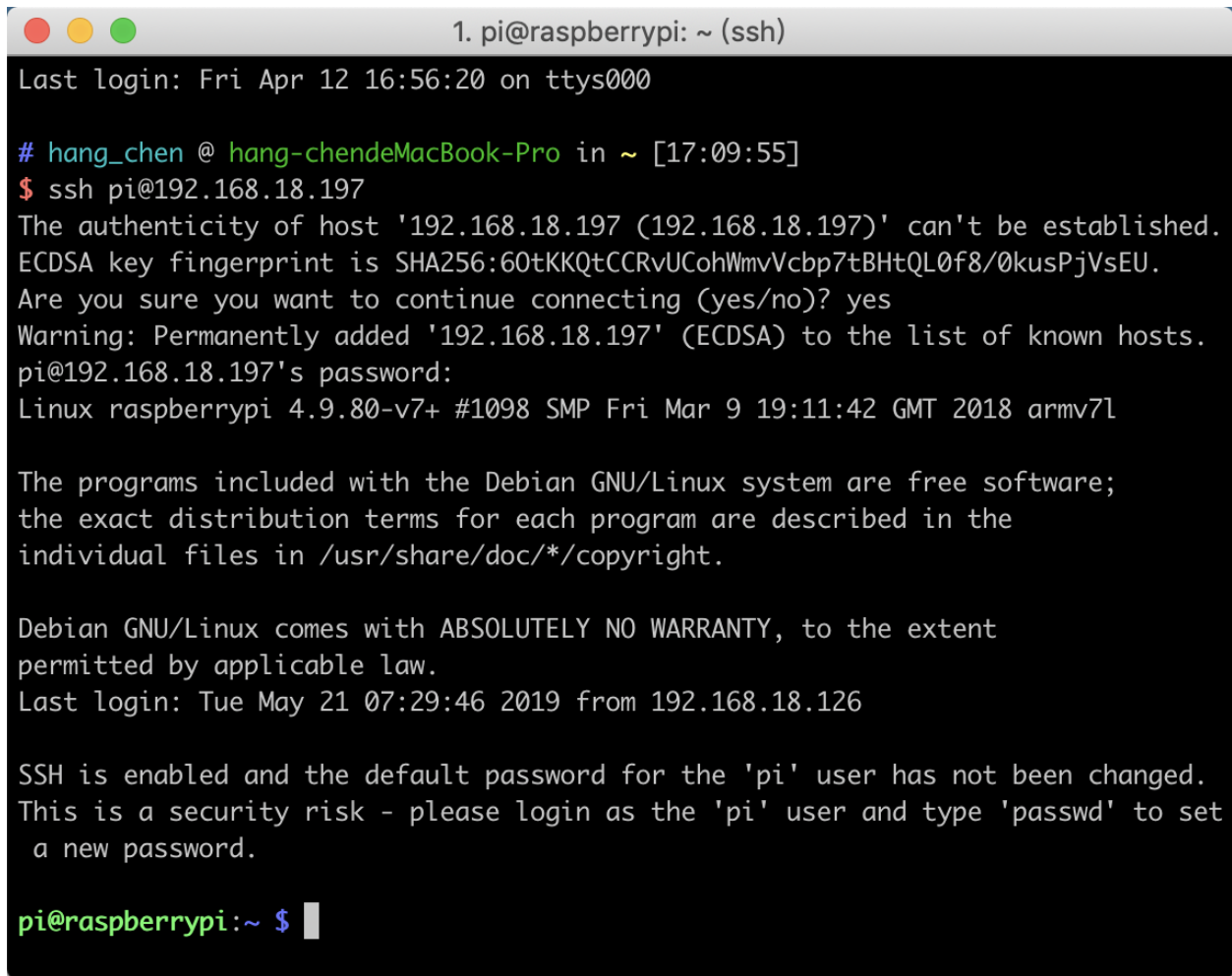
Input the passcode and the default password is **raspberrypi**.

```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password: 
```

Step 5

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



```
1. pi@raspberrypi: ~ (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRVUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
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

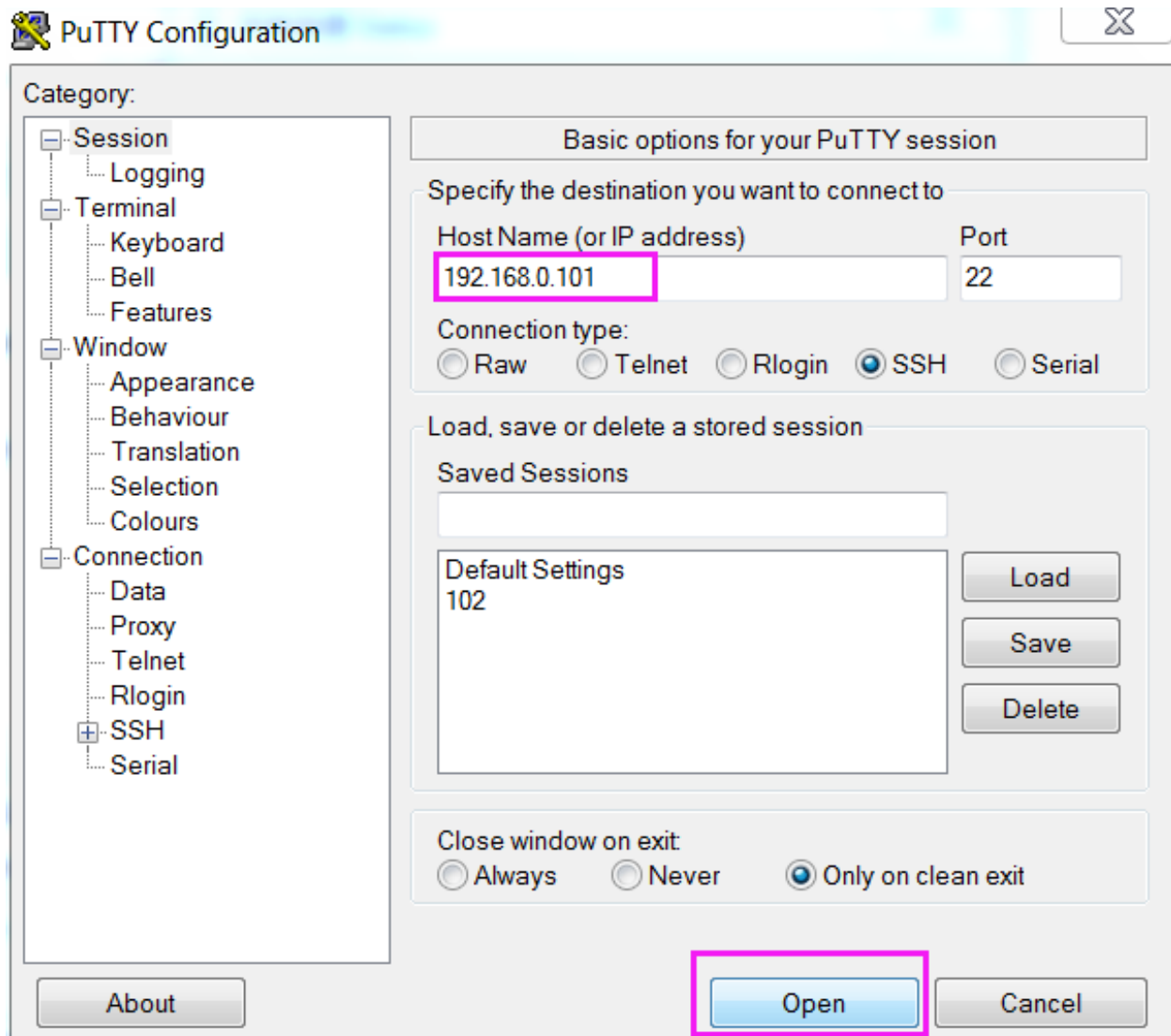
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

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).

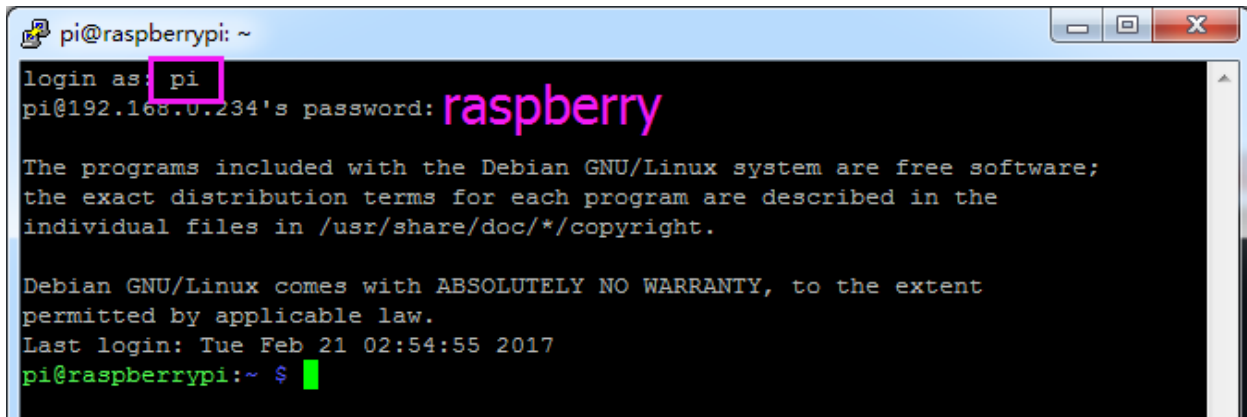


Step 3

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:** “**raspberrypi**” (the default one, if you haven’t changed it).



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

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

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

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.

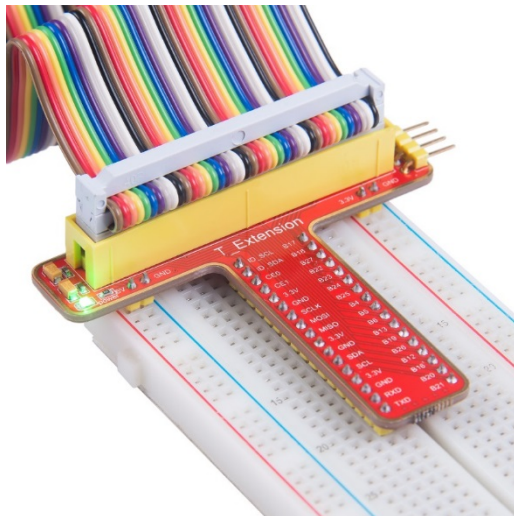
Note: If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily.

For details on how to do this, please refer to [Remote Desktop](#).

ASSEMBLE THE RASPBERRY PI TO RAB HOLDER

3.1 T_Extension Board

The function of the extension board is to lead out pins of the Raspberry Pi to breadboard by GPIO Extension Board to avoid GPIO damage caused by frequent plugging in or out. For plugging convenience, we designed it in T-shape and name it T-Shape Extension Board.



This is our 40-pin GPIO Extension Board and GPIO cable for Raspberry Pi model B+, 2 model B, 3 model B and 4 model B.



For your better understanding of every pins, we have drawn a table for you to know the Name, BCM and wiring pi of each pin.

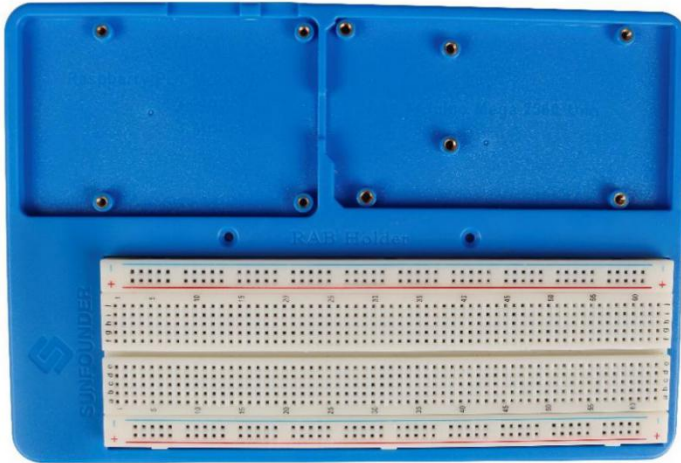
| Name | wiringPi | BCM | | BCM | wiringPi | Name | |
|-------------|----------|------|--------|-----|----------|------|--------|
| T Extension | | | | | | | |
| ID_SCL | 31 | 1 | ID_SCL | B17 | 17 | 0 | GPIO0 |
| ID_SDA | 30 | 0 | ID_SDA | B18 | 18 | 1 | GPIO1 |
| CEO | 30 | 8 | CEO | B27 | 27 | 2 | GPIO2 |
| CE1 | 11 | 7 | CE1 | B22 | 22 | 3 | GPIO3 |
| 3.3V | 3.3V | 3.3V | 3.3V | B23 | 23 | 4 | GPIO4 |
| 0V | GND | GND | GND | B24 | 24 | 5 | GPIO5 |
| SCLK | 14 | 11 | SCLK | B25 | 25 | 6 | GPIO6 |
| MOSI | 12 | 10 | MOSI | B4 | 4 | 7 | GPIO7 |
| MISO | 13 | 9 | MISO | B5 | 5 | 21 | GPIO21 |
| 3.3V | 3.3V | 3.3V | 3.3V | B6 | 6 | 22 | GPIO22 |
| 0V | GND | GND | GND | B13 | 13 | 23 | GPIO23 |
| SDA | 8 | 2 | SDA | B19 | 19 | 24 | GPIO24 |
| SCL | 9 | 3 | SCL | B26 | 26 | 25 | GPIO25 |
| 3.3V | 3.3V | 3.3V | 3.3V | B12 | 12 | 26 | GPIO26 |
| 0V | GND | GND | GND | B16 | 16 | 27 | GPIO27 |
| RXD | 16 | 15 | RXD | B20 | 20 | 28 | GPIO28 |
| TXD | 15 | 14 | TXD | B21 | 21 | 29 | GPIO29 |

3.2 RAB Holder

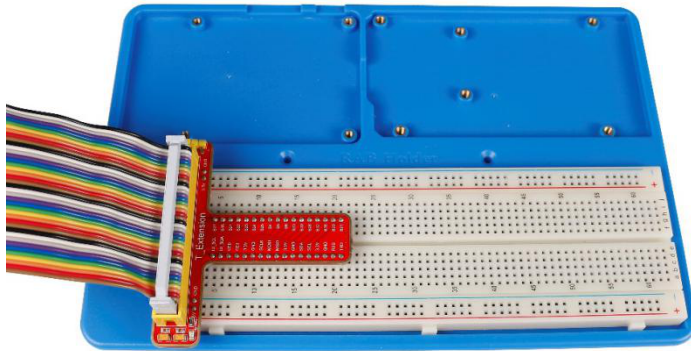
RAB Holder is a basic but indispensable component for your experiment. It makes your experiment easier and can be used for fixing a bread board, an Arduino board like Uno board or Mega2560, or a Raspberry Pi board.

Before starting the experiment, you need to assemble the Raspberry Pi and the breadboard on the RAB Holder first.

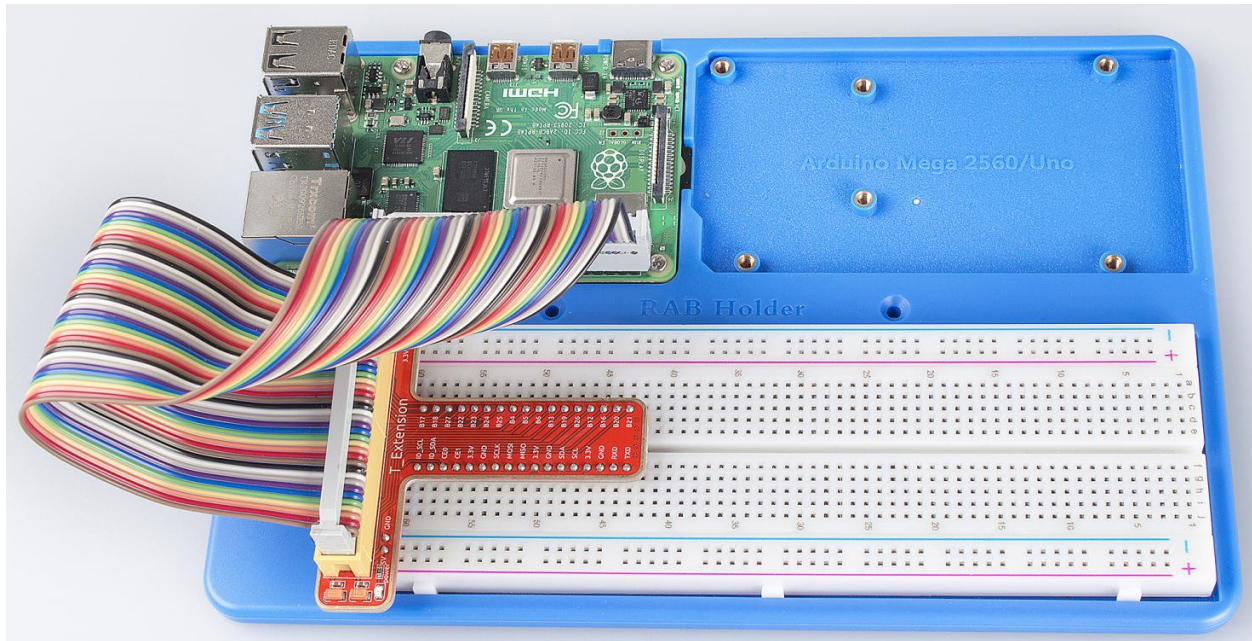
Remove the sticker from the back of the breadboard first, and fix the breadboard on the RAB Holder. Pay attention to place it in such a position as shown in the figure below, so that the 3.3V pin and 5V pin on the T-Extension Board align with the bus strips besides the two red lines when we insert the T-Extension Board later.



Then insert the T-Extension Board into the Breadboard, and insert the 40-pin GPIO Cable into the board.



Then place the Raspberry Pi in the holder, fasten it with M2.5x5 screws. Since it may be a little difficult to fasten it, be careful to operate.



Pay attention to the direction when plugging in the 40-pin GPIO Cable into the Raspberry Pi pins. The black wire at the edge should be close to the TF card slot.

Note: DO NOT connect them inversely, or the Raspberry Pi will be short cut!



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.

4.1 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:~ $ 
```

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

For more details about wiringPi, you can refer to [WiringPi](#).

DOWNLOAD THE CODE

We provide two methods for download:

Method 1: Use git clone (Recommended)

Log into Raspberry Pi's console, just as previously shown.

Change directory to */home/pi*.

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

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

Clone the repository from GitHub.

```
git clone https://github.com/sunfounder/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi.git
```

The advantage of this method is that, you can update the latest code any time you want, using git pull under the folder.

Method 2: Download the code.

Download the source code from github: https://github.com/sunfounder/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi

https://github.com/sunfounder/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi

SunFounder Supet Kit V3.0 for Raspberry Pi

| | | | | |
|------------|----------|------------|---------------|---------|
| 21 commits | 1 branch | 0 releases | 1 contributor | GPL-2.0 |
|------------|----------|------------|---------------|---------|

Branch: master New pull request Find file Clone or download

| File | Commit Message | Time |
|------------|------------------------|-------------|
| C | modify the code | |
| Python | Update 04_breathLed.py | |
| .gitignore | update | |
| LICENSE | create repository | 2 years ago |
| README.md | create repository | 2 years ago |
| show | create repository | 2 years ago |

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

https://github.com/sunfounder/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi

Open in Desktop Download ZIP

LESSONS

6.1 Lesson 1 Blinking LED

6.1.1 Introduction

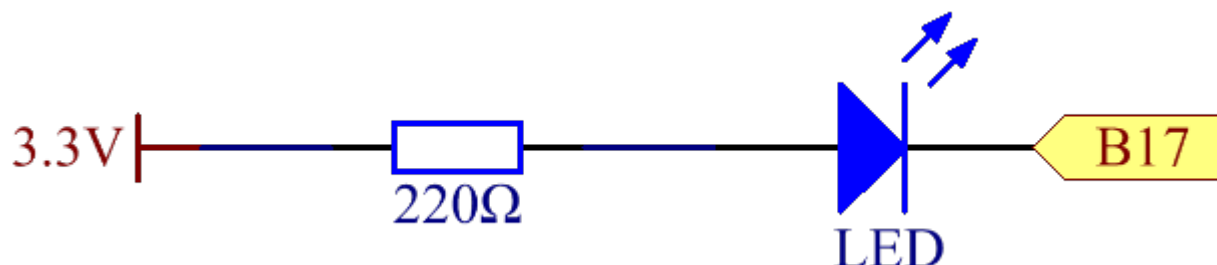
In this lesson, we will learn how to program Raspberry Pi to make an LED blink. You can play numerous tricks with an LED as you want. Now get to start and you will enjoy the fun of DIY at once!

6.1.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * T-Extension Board
- 1 * 40-Pin Cable
- 1 * LED
- 1 * Resistor (220)
- Jumper wires

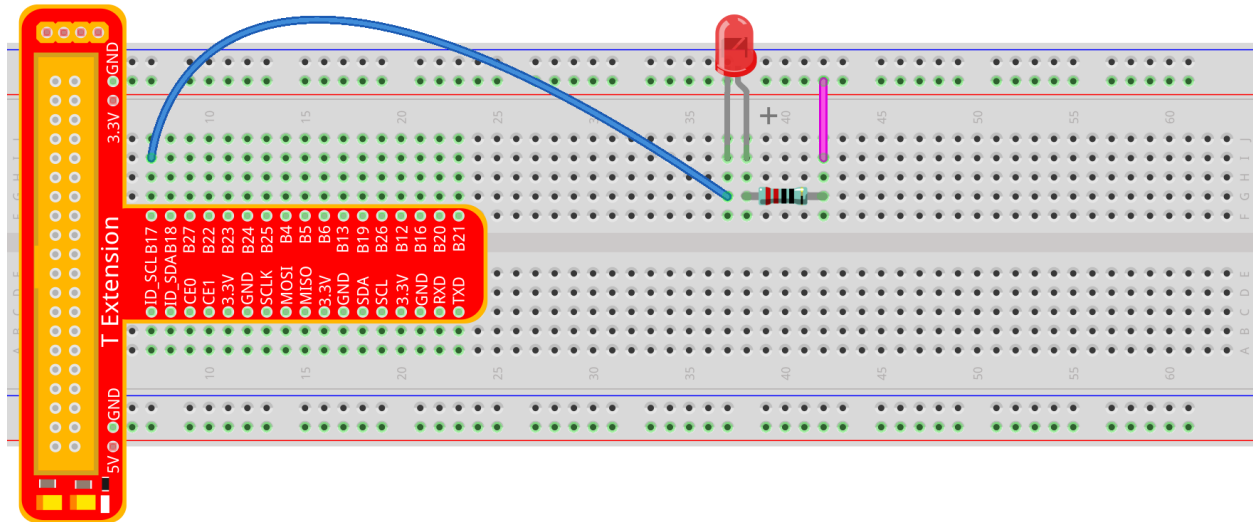
6.1.3 Principle

In this experiment, connect a 220 resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to B17 of Raspberry Pi. We can see from the schematic diagram that the anode of LED connects to a current-limiting resistor and then to 3.3V. **Therefore**, to turn on an LED, we need to make B17 low (0V) level. It can be realized by programming.



6.1.4 Experimental Procedures

Step 1: Build the circuit.



fritzing

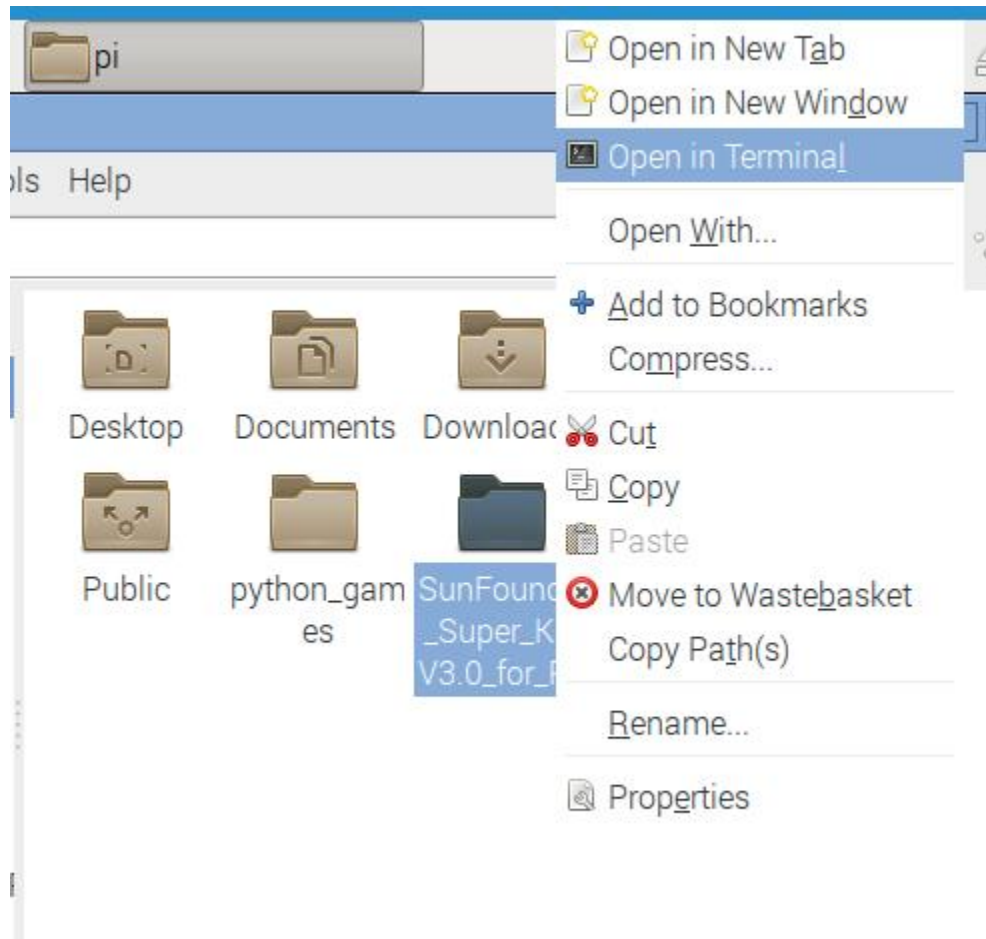
For C Language Users:

Step 2: Go to the folder of the code.

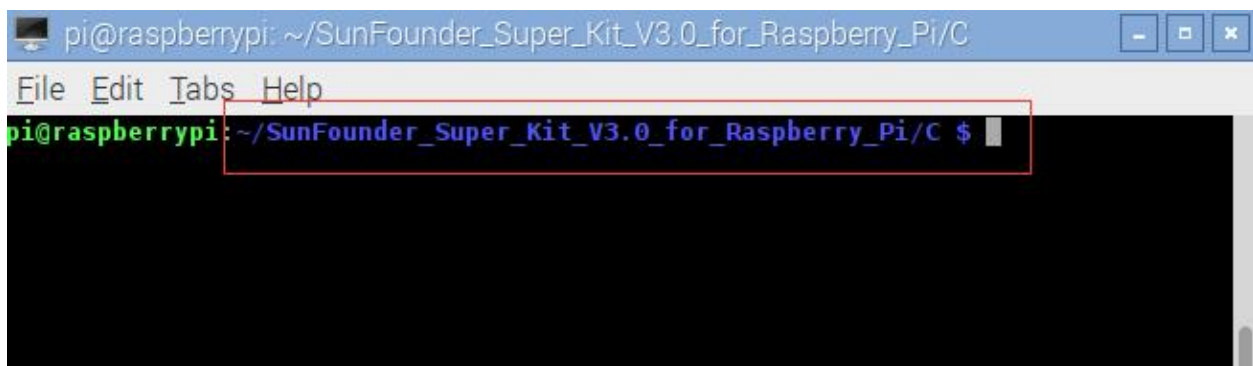
If you use a monitor, you're recommended to take the following steps.

Go to /home/pi/ and find the folder SunFounder_Super_Kit_V3.0_for_Raspberry_Pi .

Find C in the folder, right-click on it and select **Open in Terminal**.



Then a window will pop up as shown below. So now you've entered the path of the code **01_blinkLed.c**



In the lessons later, we will show how to get into the folder of the code in command way, not with the display. You only need to find out the code file and right click **Open in Terminal**. You can back to lesson 1 to check if you forgot. Certainly, the terminal can be opened if you're using display, and then use cd command directly to go to the code path.

If you log into the Raspberry Pi remotely, use “cd” to change directory:

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

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

```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
login as: pi
pi@192.168.0.131's password:
Server refused to set all environment variables

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jul 21 06:42:51 2016 from 192.168.0.130
pi@raspberrypi:~ $ cd ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C/
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $
```

In either way, you now are in the folder *C*. The subsequent procedures under the two methods are the same. Let's move on.

Step 3: Compile the Code.

```
gcc 01_blinkLed.c -o 01_blinkLed -lwiringPi
```

Note: gcc is GNU Compiler Collection. Here, it functions like compiling the C language file *01_blinkLed.c* and outputting an executable file *01_blinkLed*. In the command, *-o* means outputting and *-lwiringPi* is to load the library *wiringPi* (*l* is short for library). If you want to write your own C code and compile to run it, you need to master gcc.

Since the gcc command is too long, you can use make to test the experimental effect of the kit to make the process quicker and more convenient.

```
make 01_blinkLed
```

Note: The make command will compile according to the rules in the Makefile. Two files will be generated after compiling: “*.o” and an executable file.

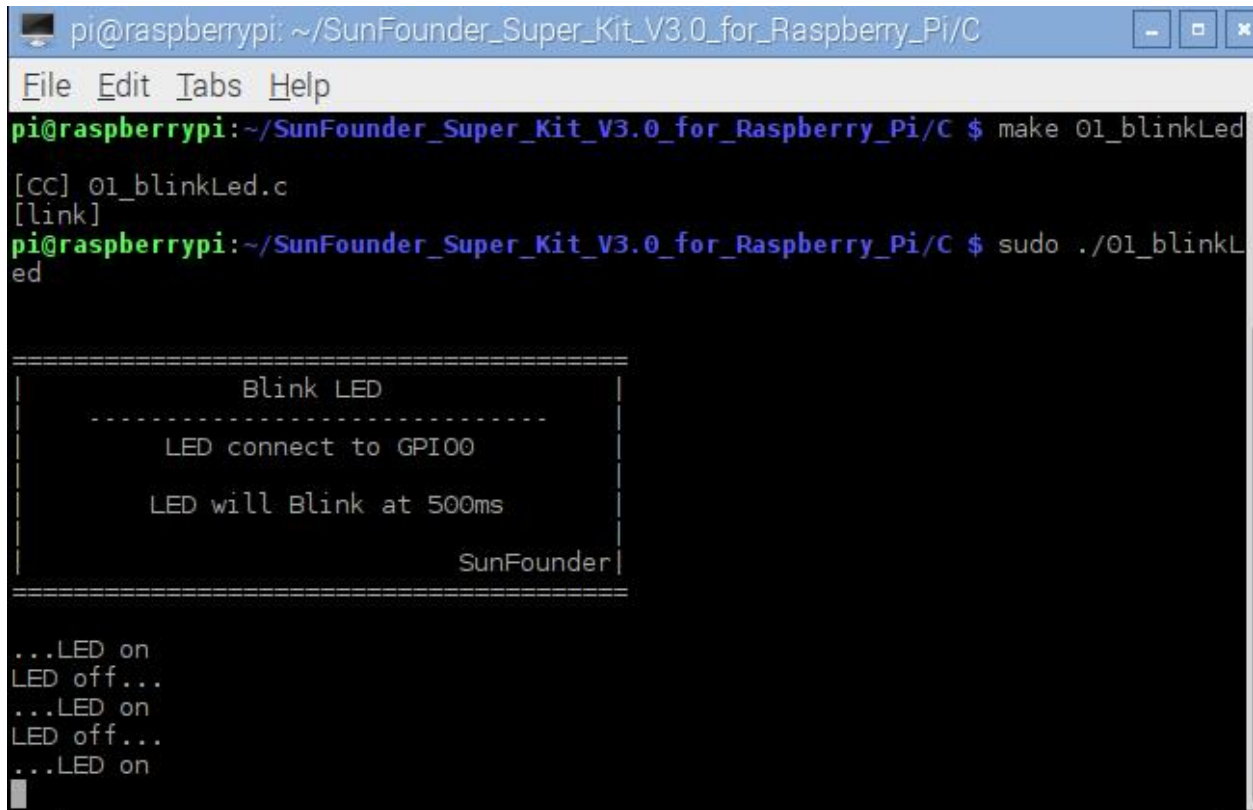
We use makefile, in essence, is to write the compilation method of gcc into the automated script. If you have written your own program in C language, you need to write and modify the makefile so as to use make command to compile your C code.

Step 4: Run the executable file output in the previous step:

```
sudo ./01_blinkLed
```

Note: To control the GPIO, you need to access to led with the permission of superuser (sudo is not needed to control the GPIO for the raspbian system after 2016-5-27), namely, by the command *sudo*. In the command “*./*” indicates the current directory. The whole command is to run the *01_blinkLed* in the current directory.

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?*.



```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
File Edit Tabs Help
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 01_blinkLed
[CC] 01_blinkLed.c
[link]
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./01_blinkLed

=====
                Blink LED
            -----
                LED connect to GPIO0
                LED will Blink at 500ms
                               SunFounder|
=====

...LED on
LED off...
...LED on
LED off...
...LED on
```

If you want to view the code *01_blinkLed.c*, press **Ctrl + C** to stop running the code. Then type the following command to open it:

```
nano 01_blinkLed.c
```

Note: nano is a text editor tool. The command is to open the code file *01_edblinkLed.c* by this tool.

```

pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
File Edit Tabs Help
GNU nano 2.2.6 File: 01_blinkLed.c
*****
* Filename   : blinkLed.c
* Description: Make an led blinking.
* Author    : Robot
* E-mail    : support@sunfounder.com
* website   : www.sunfounder.com
* Update    : Caven 2016/07/01
*****/
#include <wiringPi.h>
#include <stdio.h>

#define LedPin 0

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

    pinMode(LedPin, OUTPUT);
}
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```

Code

```

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

#define LedPin 0

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

    pinMode(LedPin, OUTPUT);

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          Blink LED          |\n");
    printf("|-----|\n");
    printf("|          LED connect to GPIO0          |\n");
    printf("|          |\n");
    printf("|          LED will Blink at 500ms          |\n");
    printf("|          |\n");

```

(continues on next page)

(continued from previous page)

```

printf("|                               SunFounder|\n");
printf("=====");
printf("\n");
printf("\n");

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

return 0;
}

```

Code Explanation

```

#include <wiringPi.h>
// The hardware drive library designed for the C language of Raspberry Pi.
// Adding this library is convenient for hardware initialization, I/O ports, PWM,
↳ outputs, etc.

#include <stdio.h>
/* Standard I/O library. The printf function used for
printing the data displayed on the screen is realized by this library.
There are many other performance functions for you to explore.*/

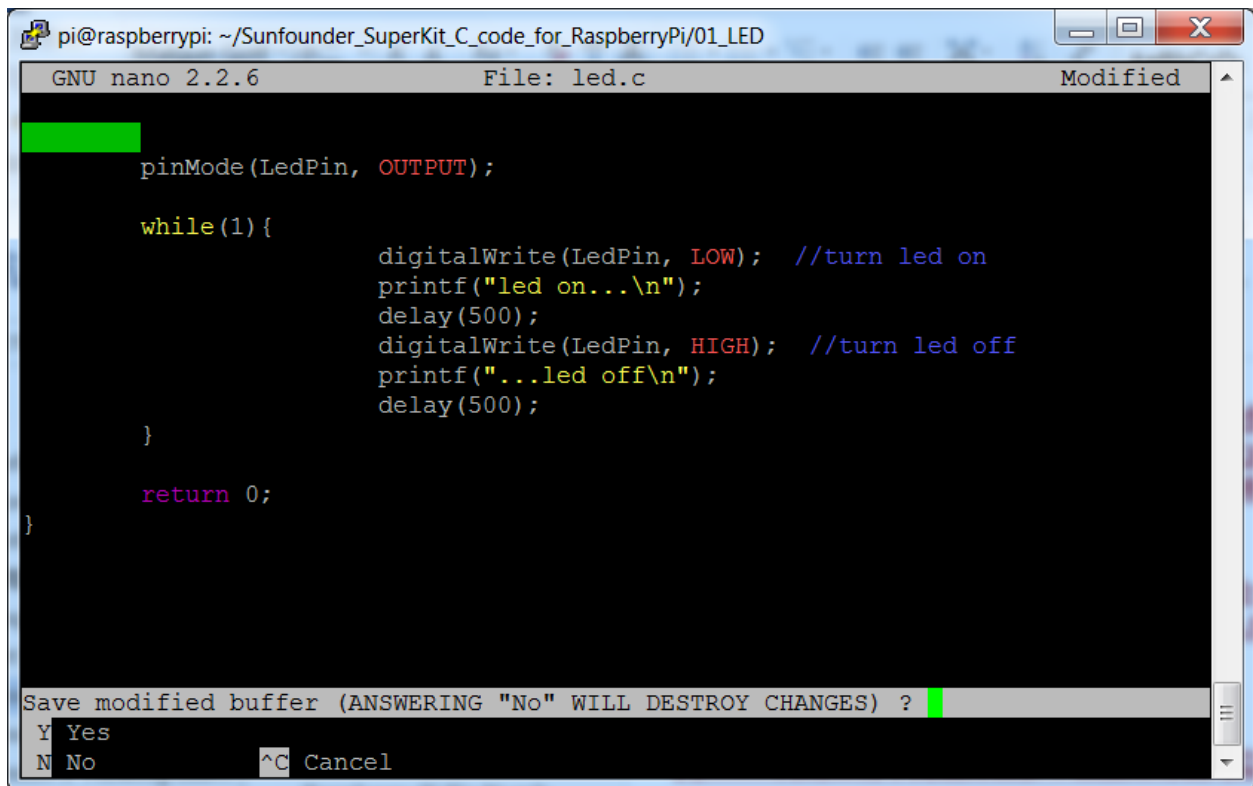
#define LedPin 0
/* Pin B17 of the T_Extension Board is corresponding to
the pin0 in wiringPi, namely, GPIO 0 of the raspberry Pi. Assign GPIO 0
to LedPin, LedPin represents GPIO 0 in the code later.*/

pinMode(LedPin, OUTPUT) // Set LedPin as output to write value to it.

digitalWrite(LedPin, LOW)
/* Set GPIO0 as 0V (low level). Since the
cathode of LED is connected to GPIO0, thus the LED will light up if
GPIO0 is set low. On the contrary, set GPIO0 as high level, digitalWrite
(LedPin, HIGH): LED will go out.*/

```

Press **Ctrl+X** to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in **Y** (save) or **N** (don't save). Then press **Enter** to exit. Repeat **Step 3** and **Step 4** to see the effect after modifying.



```
pi@raspberrypi: ~/Sunfounder_SuperKit_C_code_for_RaspberryPi/01_LED
GNU nano 2.2.6 File: led.c Modified
pinMode(LedPin, OUTPUT);

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

return 0;
}
```

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?

Y Yes

N No ^C Cancel

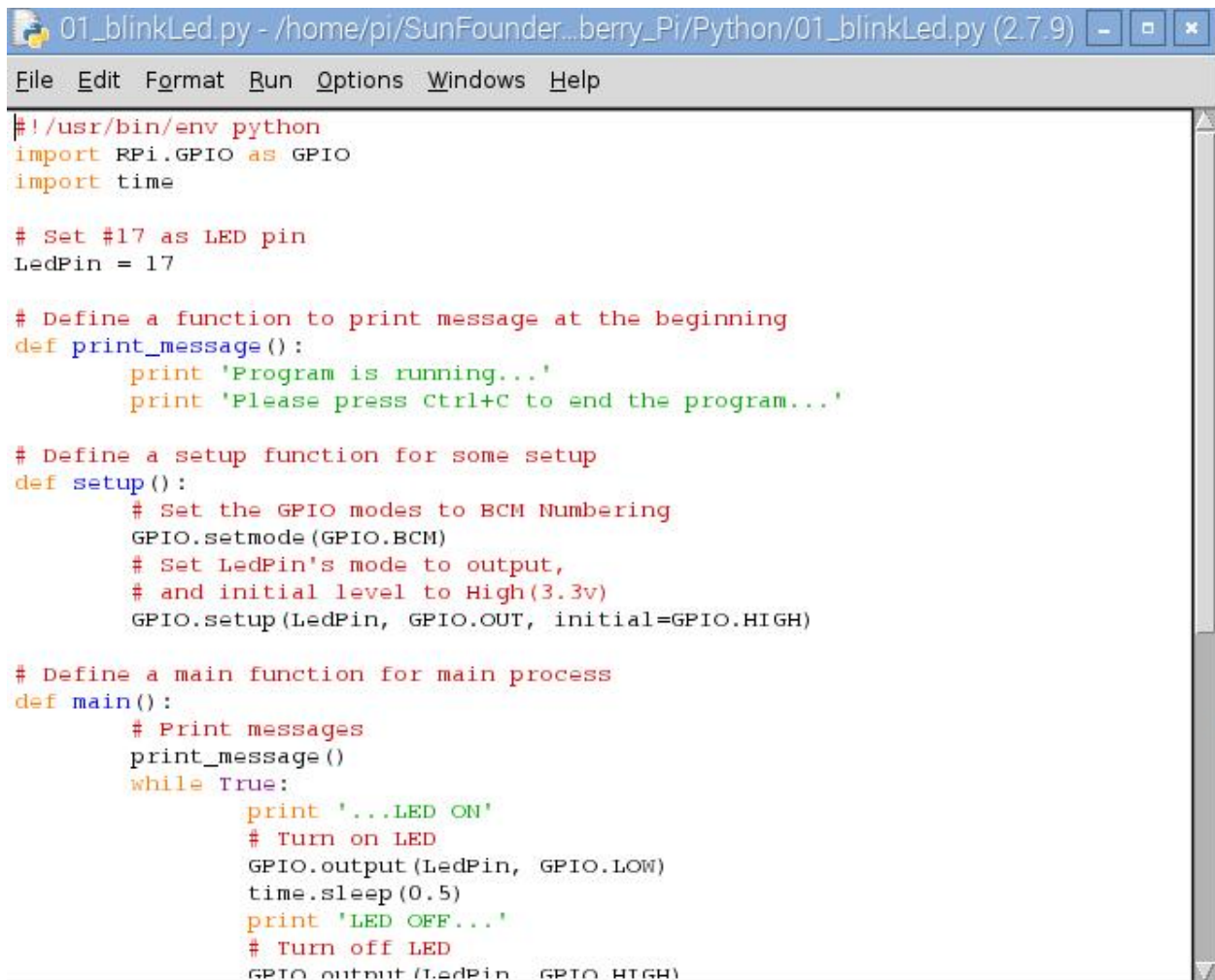
For Python Users:

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

Open the downloaded folder *SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python* and you can see them.

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

Find *01_blinkLed.py* and double click it to open. Now you're in the file.



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

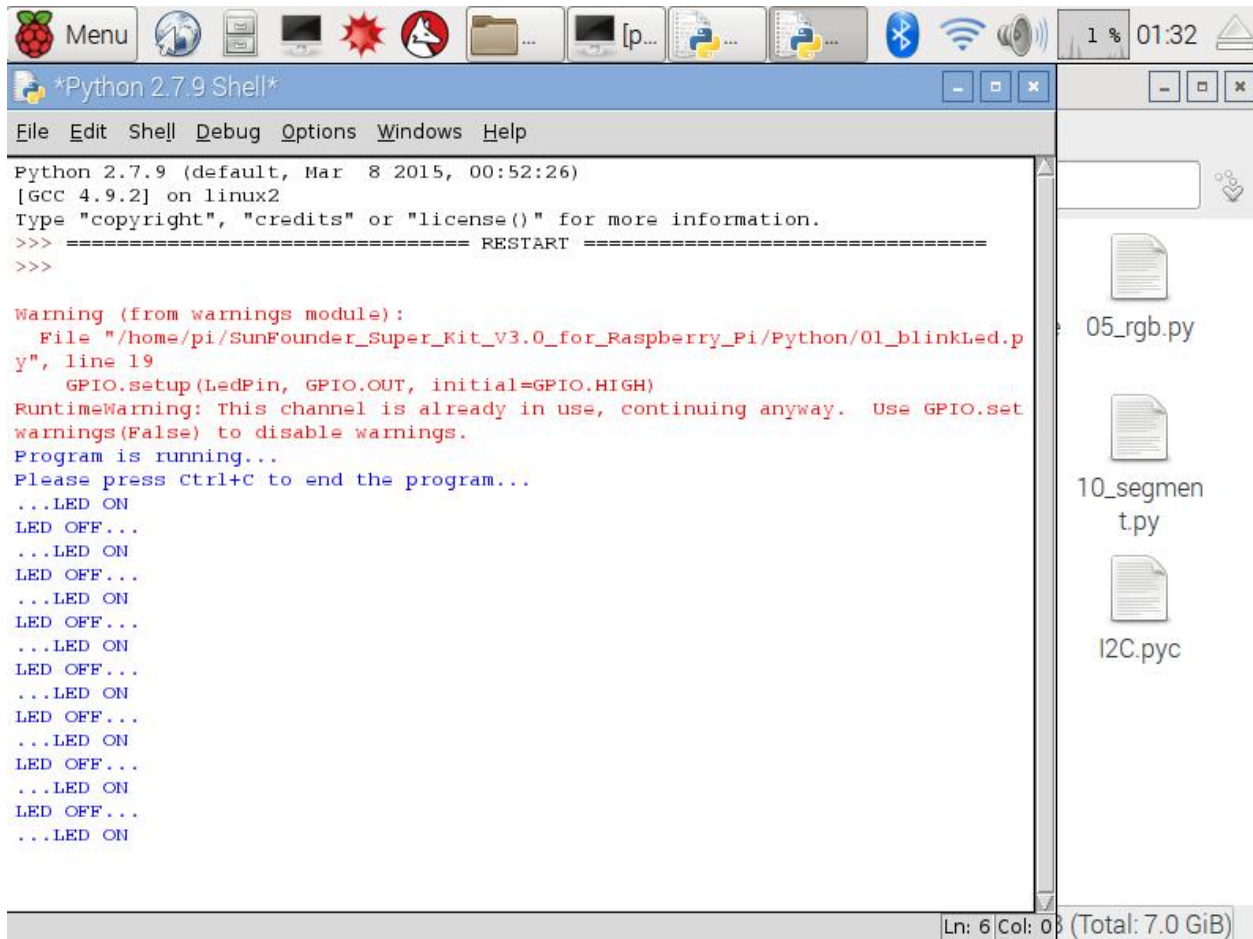
# Set #17 as LED pin
LedPin = 17

# Define a function to print message at the beginning
def print_message():
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'

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

# Define a main function for main process
def main():
    # Print messages
    print_message()
    while True:
        print '...LED ON'
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)
        print 'LED OFF...'
        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
```

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



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

If you want to log into the Raspberry Pi remotely, type in the command:

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Run the code:

```
sudo python3 01_blinkLed.py
```

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


```

pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
pi@raspberrypi:~ $ cd ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python/
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python $ sudo python 01_blinkLed.py
01_blinkLed.py:19: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(
s(False) to disable warnings.
  GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
Program is running...
Please press Ctrl+C to end the program...
...LED ON
LED OFF...
...LED ON
LED OFF...
...LED ON
LED OFF...

```

If you want to view the code `01_blinkLed.py`, press **Ctrl + C** to stop running the code. Then type the following command to open it:

```
nano 01_blinkLed.py
```

Note: nano is a text editor tool. The command is to open the code file `01_blinkLed.c` by this tool.

```

pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
GNU nano 2.2.6 File: 01_blinkLed.py

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

# Set #17 as LED pin
LedPin = 17

# Define a function to print message at the beginning
def print_message():
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'

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

[ Read 51 lines (Converted from DOS format) ]

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is     ^V Next Page     ^U UnCut Text    ^T To Spell

```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set #17 as LED pin

```

(continues on next page)

(continued from previous page)

```

LedPin = 17

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|                      Blink LED                      |")
    print ("|          -----          |")
    print ("|          LED connect to B17          |")
    print ("|                      |")
    print ("|          LED will Blink at 500ms          |")
    print ("|                      |")
    print ("|                               SunFounder |")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\\n")

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

# Define a main function for main process
def main():
    # Print messages
    print_message()
    while True:
        print ("...LED ON")
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)
        print ("LED OFF...")
        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)

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

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

```

Code Explanation

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

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

import RPi.GPIO as GPIO
# import RPI.GPIO package, thus python code control GPIO easily with it.

import time
# import time package, for time delay function in the following program.

LedPin = 17
# LED connects to the B17 of the T-shape extension board, namely, the GPIO 0 of the
↳ Raspberry Pi.

# Define a setup function for some setup
def setup():

    GPIO.setmode(GPIO.BCM) # Set the GPIO modes to BCM Numbering

    # Set LedPin's mode to output, and initial level to High (3.3v)

    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():

    # Print messages

    print_message()

    while True:

        print ("...LED ON")

        # Turn on LED

        GPIO.output(LedPin, GPIO.LOW)

        # delay 0.5 second, which is equals to the delay in C language, using
        second as the unit,

        time.sleep(0.5)

        print ("LED OFF...")

        # Turn off LED

        GPIO.output(LedPin, GPIO.HIGH)

        time.sleep(0.5)

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

(continues on next page)

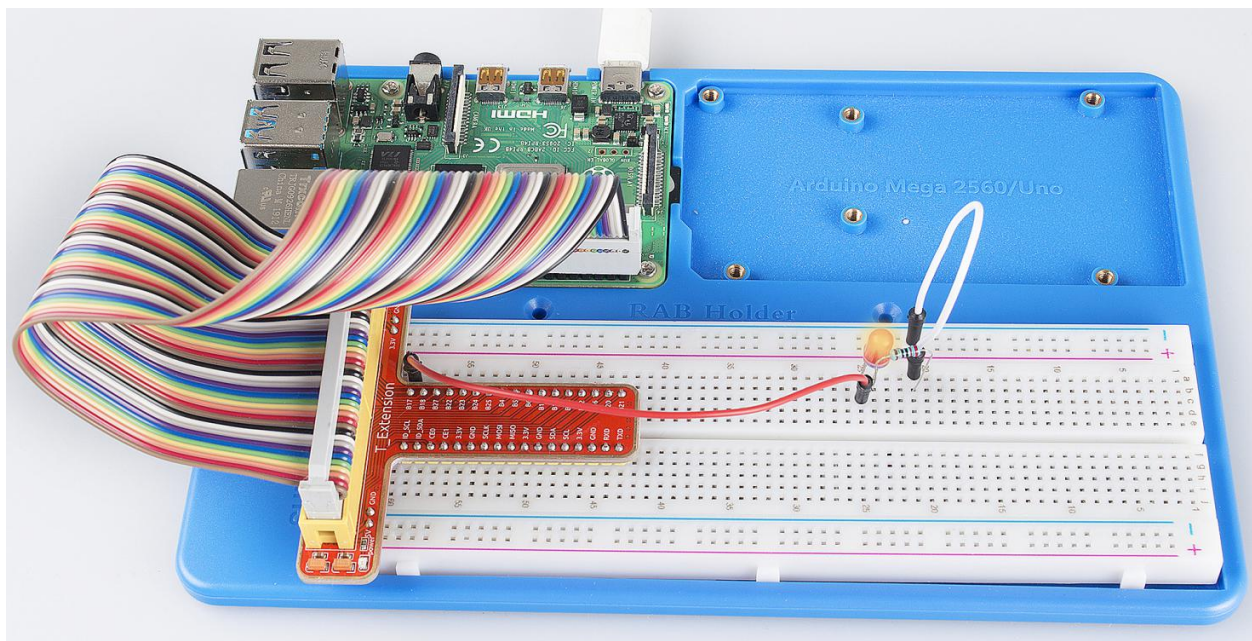
(continued from previous page)

```
def destroy():  
    # Turn off LED  
    GPIO.output(LedPin, GPIO.HIGH)  
    # Release resource  
    GPIO.cleanup()  
  
# If run this script directly, do:  
if __name__ == '__main__':  
    setup()  
  
    try:  
        main()  
  
        # When 'Ctrl+C' is pressed, the child program destroy () will be executed.  
  
    except KeyboardInterrupt:  
        destroy()
```

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

Then press **Enter** to exit. Type in `nano 01_blinkLed.py` again to see the effect after the change.

Run the code to make it work. It will be like below:



6.1.5 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)* (for C) or *time.sleep(0.2)* (for python) in the program, recompile and run, and then you will see the LED blink faster.

6.1.6 Summary

Raspberry Pi packages many low-level detail designs, which ease your way to explore your own apps. 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.

6.1.7 FAQ

If you haven't modified the code, you do not need to run make *01_blinkLed* again.

```
make 01_blinkLed
```

Or a message will appear: make: '01_blinkLed' is up to date.

```
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ gcc 01_blinkLed.c -o 01_blinkLed -lwiringPi
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 01_blinkLed
make: '01_blinkLed' is up to date.
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $
```

It will not appear only when you run the make command after having changed the code and saved it.

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.

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.

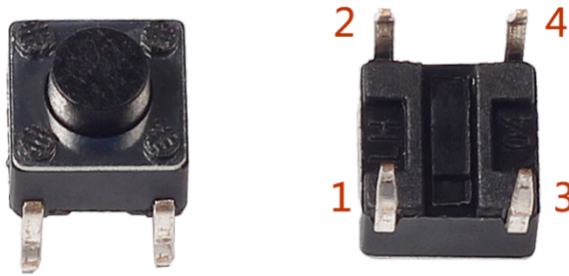
6.2.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * LED
- 1 * Button
- 2 * Resistor (220, 10K)
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin Cable

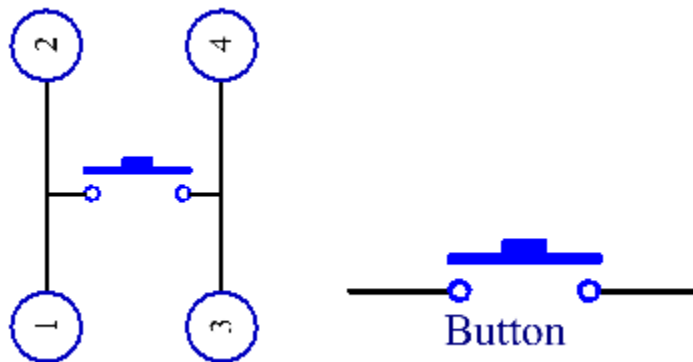
6.2.3 Principle

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. Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.



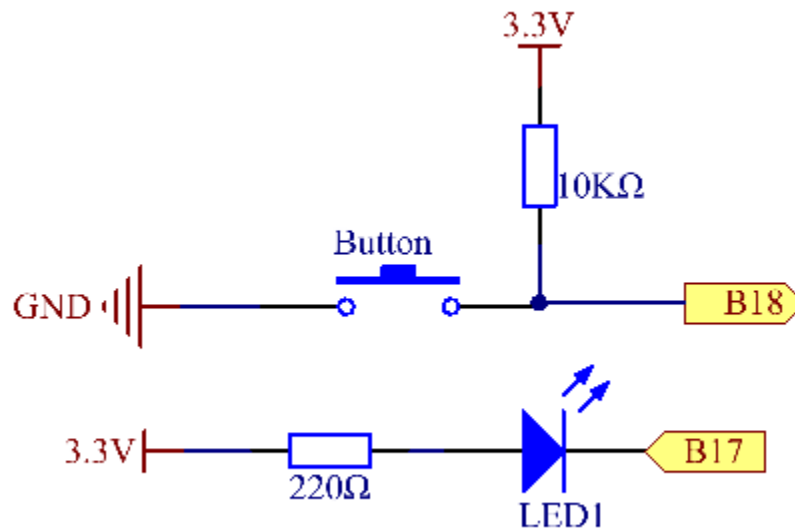
The following is the internal structure of a button. Since the pin 1 is connected to pin 2, and pin 3 to pin 4. The symbol on the right below is usually used to represent a button in circuits.



When the button is pressed, the 4 pins are connected, thus closing the circuit.

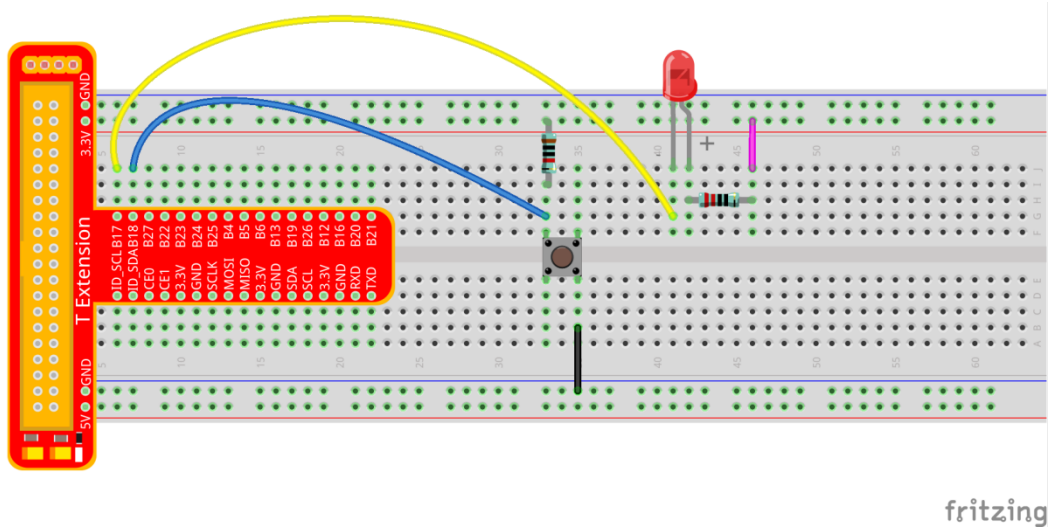
Use a normally open button as the input of Raspberry Pi, the detailed connection is as shown in the schematic diagram below. When the button is pressed, the B18 will turn into low level (0V). We can detect the state of the B18 through programming. That is, if the B18 turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

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



6.2.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Open the code file:

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

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

Step 3: Compile the Code.

```
gcc 02_buttonControlLed.c -o 02_buttonControlLed -lwiringPi
```

or

```
make 02_buttonControlLed
```

Step 4: Run the executable file above.

```
sudo ./02_buttonControlLed
```

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?*.

Step 5: Check the code.

```
nano 02_buttonControlLed.c
```

Code

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

#define LedPin          0
#define ButtonPin      1

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

    pinMode(LedPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    // Pull up to 3.3V,make GPIO1 a stable level
    pullUpDnControl(ButtonPin, PUD_UP);

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          Button control LED          |\n");
    printf("|-----|\n");
    printf("|          LED connect to GPIO0        |\n");
    printf("|          Button connect to GPIO1     |\n");
    printf("|                                       |\n");
    printf("|          Press button to turn on LED. |\n");
    printf("|                                       |\n");
    printf("|                                       |\n");
    printf("|                                       |\n");
    printf("|                                       |\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    digitalWrite(LedPin, HIGH);
    printf("LED off...\n");
```

(continues on next page)

(continued from previous page)

```

while(1){
    // Indicate that button has pressed down
    if(digitalRead(ButtonPin) == 0){
        // Led on
        digitalWrite(LedPin, LOW);
        printf("...LED on\n");
        delay(100);
    }
    else{
        // Led off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
        delay(100);
    }
}
return 0;
}

```

Code Explanation

```

#define LedPin 0
/* Pin B17 in the T_Extension Board connects to the GPIO0.
GPIO0 corresponds to pin0 in the wiringPi pin figure. So in C program,
LedPin is defined as 0. */

#define ButtonPin 1
/* Pin B18 in the T_Extension Board connects to the
GPIO8. GPIO8 corresponds to pin1 in the wiringPi pin figure. So in C
program, LedPin is defined as 1.*/

pinMode(LedPin, OUTPUT) // Set LedPin as output to assign value to it.

pinMode(ButtonPin, INPUT) // Set ButtonPin as input to read the value of ButtonPin.

pullUpDnControl(ButtonPin, PUD_UP)
/* Set the ButtonPin as pull-up input.
When the button is not pressed, the I/O port is 3.3V. When the button is
pressed, the I/O port connects to GND (0V). You can judge the button
status by reading the level value of the I/O port.*/

while(1){

    // indicate that button has pressed down

    if(digitalRead(ButtonPin) == 0)

    {

        // LED on

        digitalWrite(LedPin, LOW);

        printf("...LED on\n");
        delay(100);
    }

    else

```

(continues on next page)

(continued from previous page)

```

{

    // LED off

    digitalWrite(LedPin, HIGH);

    printf("LED off...\n");
    delay(100);
}

/* digitalWrite (LedPin, HIGH) in while: close the LED. if (digitalRead(ButtonPin) ==
↪ 0:
check whether the button has been pressed. Execute digitalWrite(LedPin, LOW)
when pressed to light up LED.*/
}

```

Press **Ctrl+X** to exit, if you have modified the code, there will be a prompt asking whether to save the changes or not. Type in **Y** (save) or **N** (don't save). Then press **Enter** to exit. Repeat **Step 3** and **Step 4** to see the effect after modifying.

For Python Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run the code.

```
sudo python3 02_buttonControlLed.py
```

Step 4: Check the code.

```
nano 02_buttonControlLed.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

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

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

# Define a function to print message at the beginning
def print_message():
    print ("=====")

```

(continues on next page)

(continued from previous page)

```

print ("|          Button control LED          |")
print ("| -----|")
print ("|          LED connect to GPIO17          |")
print ("|          Button connect to GPIO18        |")
print ("|                                         |")
print ("|          Press button to turn on/off LED. |")
print ("|                                         |")
print ("|                                         |SunFounder|")
print ("=====\\n")
print ("Program is running...")
print ("Please press Ctrl+C to end the program...")
#raw_input ("Press Enter to begin\\n")

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

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

# Define a main function for main process
def main():
    # Print messages
    print_message()
    while True:
        # Don't do anything.
        time.sleep(1)

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

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

```

(continues on next page)

(continued from previous page)

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

Code Explanation

```
LedPin = 17 # Set #17 as LED pin

BtnPin = 18 # Set #18 as button pin

# Set up a falling detect on BtnPin, and callback function to swled

GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLED)

# Define a callback function for button callback, execute the function after the
↳callback of the interrupt.

def swLed(ev=None):

    global Led_status

    # Switch Led status (on-->off; off-->on)

    Led_status = not Led_status

    GPIO.output(LedPin, Led_status)

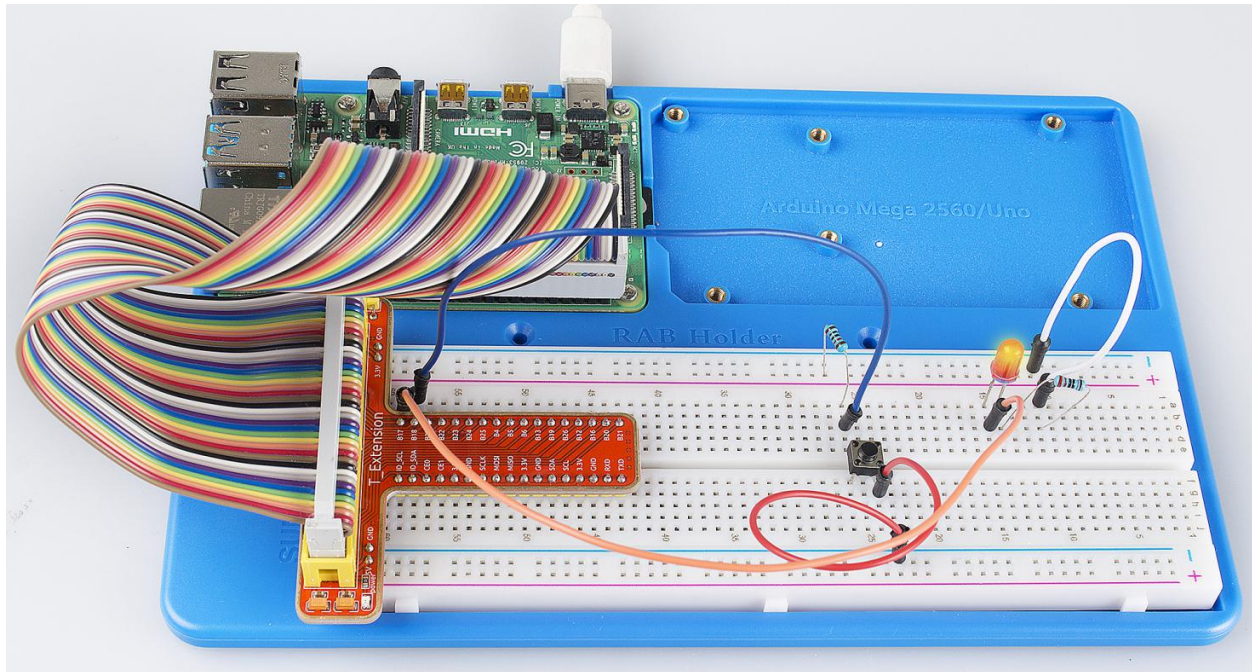
    if Led_status:

        print ("LED OFF...")

    else:

        print ("...LED ON")
```

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.



6.3 Lesson 3 Flowing LED Lights

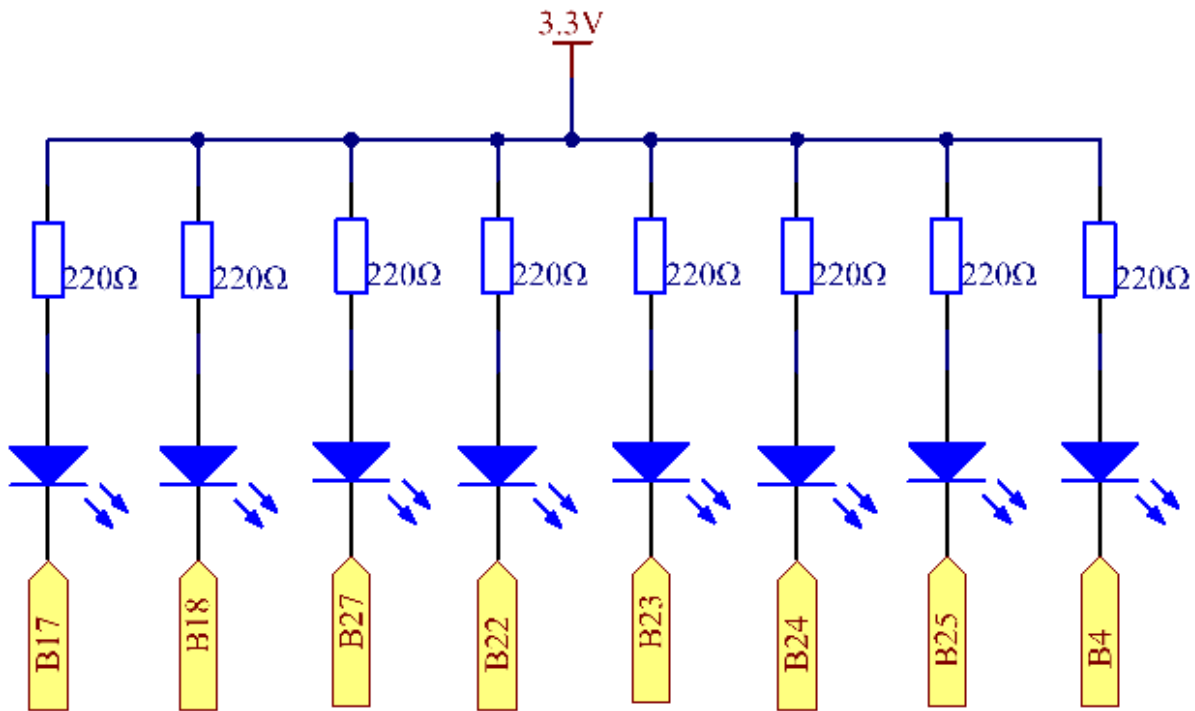
6.3.1 Introduction

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
- 1 * T-Extension Board
- 1 * 40-Pin Cable

6.3.3 Principle



Principle: Judging from the schematic diagram, we can know that a LED and a current-limiting resistor have been connected to B17, B18, B27, B22, B23, B24, B25, and B4 respectively. The current-limiting resistor has been connected to the 3.3V power supply on other side. Therefore, if we want to light up one LED, we only need to set the GPIO of the LED as low level. So in this experiment, set B17, B18, B27, B22, B23, B24, B25, and B4 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.



Commit the following line.

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

Step 3: Open the code file.

Note: Use the `cd` command to switch to the code path of this experiment.

or

Note: gcc is a linux command which can realize compiling and generating the C language program file **03_8Led.c** to the executable file **03_8Led**.

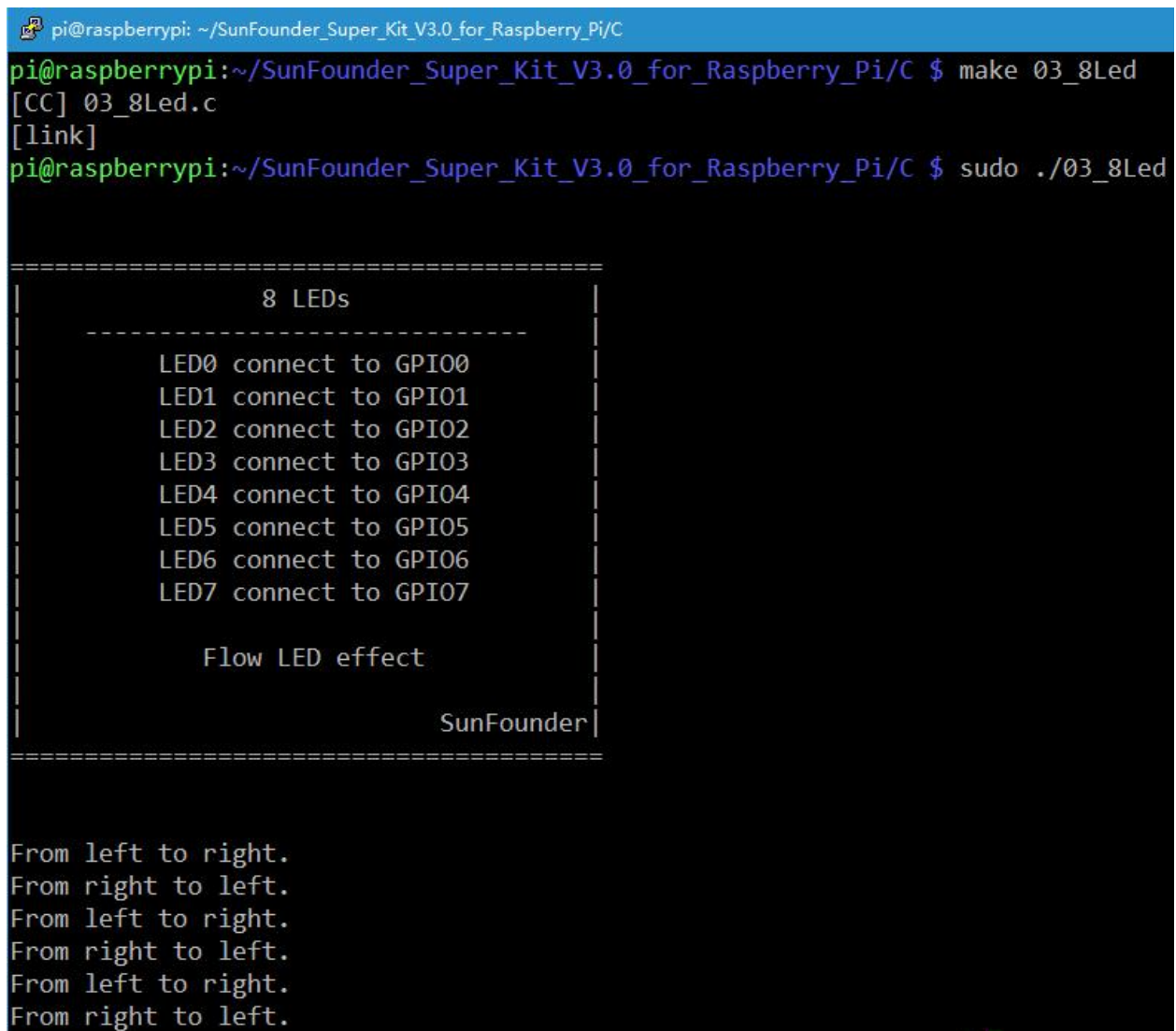
make is a linux command which can compiling and generating the executable file according to the rule inside the makefile.

Step 5: Run the executable file above.

```
sudo ./03_8Led
```

Note: Here the Raspberry Pi will run the executable file *03_8Led* compiled previously.

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?*.



```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 03_8Led
[CC] 03_8Led.c
[link]
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./03_8Led

=====
|               8 LEDs               |
|-----|
| LED0 connect to GPIO0 |
| LED1 connect to GPIO1 |
| LED2 connect to GPIO2 |
| LED3 connect to GPIO3 |
| LED4 connect to GPIO4 |
| LED5 connect to GPIO5 |
| LED6 connect to GPIO6 |
| LED7 connect to GPIO7 |
|                               |
|      Flow LED effect      |
|                               |
|               SunFounder |
|-----|
=====

From left to right.
From right to left.
From left to right.
From right to left.
From left to right.
From right to left.
```

Code

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

(continues on next page)

(continued from previous page)

```

// Turn LED(channel) on
void led_on(int channel){
    digitalWrite(channel, LOW);
}

// Turn LED(channel) off
void led_off(int channel){
    digitalWrite(channel, HIGH);
}

int main(void){
    int i;

    // When initialize wiring failed, print messageto screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    // Set 8 pins' modes to output
    for(i=0;i<8;i++){
        pinMode(i, OUTPUT);
    }

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          8 LEDs          |\n");
    printf("|-----|\n");
    printf("|    LED0 connect to GPIO0    |\n");
    printf("|    LED1 connect to GPIO1    |\n");
    printf("|    LED2 connect to GPIO2    |\n");
    printf("|    LED3 connect to GPIO3    |\n");
    printf("|    LED4 connect to GPIO4    |\n");
    printf("|    LED5 connect to GPIO5    |\n");
    printf("|    LED6 connect to GPIO6    |\n");
    printf("|    LED7 connect to GPIO7    |\n");
    printf("|                               |\n");
    printf("|    Flow LED effect          |\n");
    printf("|                               |\n");
    printf("|                               |\n");
    printf("|                               |\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    while(1){
        // Turn LED on from left to right
        printf("From left to right.\n");
        for(i=0;i<8;i++){
            led_on(i);
            delay(100);
            led_off(i);
        }
        // Turn LED off from right to left
        printf("From right to left.\n");
        for(i=8;i>=0;i--){
            led_on(i);
            delay(100);

```

(continues on next page)

(continued from previous page)

```
        led_off(i);
    }
}

return 0;
}
```

Code Explanation

```
void Led_on(int channel)
{ /* This is a subfunction with a formal parameter
   int channel for importing the numbers of the controlled pins. Its
   function body is digitalWrite(channel, LOW); Set the I/O port of channel
   as low level(0V), the LED on this port lights up. void led_off(int
   channel) is to turn off the LED. Setting function simplifies the input
   for the repeated content.*/

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

        pinMode(i, OUTPUT);

    }
    /*The cathodes of the 8 LEDs connect to B17, B18, B27, B22, B23, B24,
    B25, and B4 of the T-shape extension board respectively, corresponding
    to 0,1,2,3,4,5,6,7. It is to set the 8 LEDs as output separately. Use
    for loop to make it more concise and efficient.*/

    for(i=0;i<8;i++)
    { //make LED on from left to right

        Led_on(i); // turn the LED i on

        delay(100); // keep the LED i lighting for 100ms.

        Led_off(i); // Turn the LED i off

    }
    /* Light up and turn off the LEDs in GPIO0~7 successively. i increases
    progressively from 0 to 7, LED0 to LED7 changes accordingly, making it
    like a flowing LED light from left to right.*/

    for(i=7;i>=0;i--)
    { //make LED off from right to left

        led_on(i); // turn the LED i on

        delay(100); // keep the LED i lighting for 100ms

        led_off(i); //turn the LED i off

    }
}
/* In this for loop, light up and turn off the LED in GPIO7 to GPIO0 successively,
making a flowing LED light from left to right.*/
```

For Python Users:**Step 3:** Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 4: Run.

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

Code

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

if version_info.major == 3:
    raw_input = input

# Set 8 Pins for 8 LEDs.
LedPins = [17, 18, 27, 22, 23, 24, 25, 4]

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|                      8 LEDs                      |")
    print ("|  -----  |")
    print ("|      LED0 connect to GPIO17      |")
    print ("|      LED1 connect to GPIO18      |")
    print ("|      LED2 connect to GPIO27      |")
    print ("|      LED3 connect to GPIO22      |")
    print ("|      LED4 connect to GPIO23      |")
    print ("|      LED5 connect to GPIO24      |")
    print ("|      LED6 connect to GPIO25      |")
    print ("|      LED7 connect to GPIO4       |")
    print ("|                                |")
    print ("|      Flow LED effect            |")
    print ("|                                |")
    print ("|                                SunFounder |")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\\n")

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

# Define a main function for main process
def main():
    # Print messages
    print_message()
    leds = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ']
```

(continues on next page)

(continued from previous page)

```

while True:
    # Turn LED on from left to right
    print ("From left to right.")
    for pin in LedPins:
        #print pin
        GPIO.output(pin, GPIO.LOW)
        leds[LedPins.index(pin)] = 0    # Show which led is on
        print (leds)
        time.sleep(0.1)
        GPIO.output(pin, GPIO.HIGH)
        leds[LedPins.index(pin)] = '-'  # Show the led is off

    # Turn LED off from right to left
    print ("From right to left.")
    for pin in reversed(LedPins):
        #print pin
        GPIO.output(pin, GPIO.LOW)
        leds[LedPins.index(pin)] = 0    # Show which led is on
        print (leds)
        time.sleep(0.1)
        GPIO.output(pin, GPIO.HIGH)
        leds[LedPins.index(pin)] = '-'  # Show the led is off

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

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

```

Code Explanation

```

LedPins = [17, 18, 27, 22, 23, 24, 25, 4] '''The cathodes of the 8 LEDs
connect to B17, B18, B27, 22, 23, 24, 25, 4 of the T-shape extension
board. In BCM, these pins are corresponding to 17, 18, 27, 22, 23, 24,
25, and 4.'''

leds = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ']
# the array to print out the status of the 8 LEDs

for pin in LedPins:
    # Assign the element in pins list to pin variable one by one.
    # In GPIO.setup (pin, GPIO.OUT), set the pins in list as output one by one.

    GPIO.output(pin, GPIO.LOW)

```

(continues on next page)

(continued from previous page)

```
# Set each element in the pins list as low level to light up the LEDs

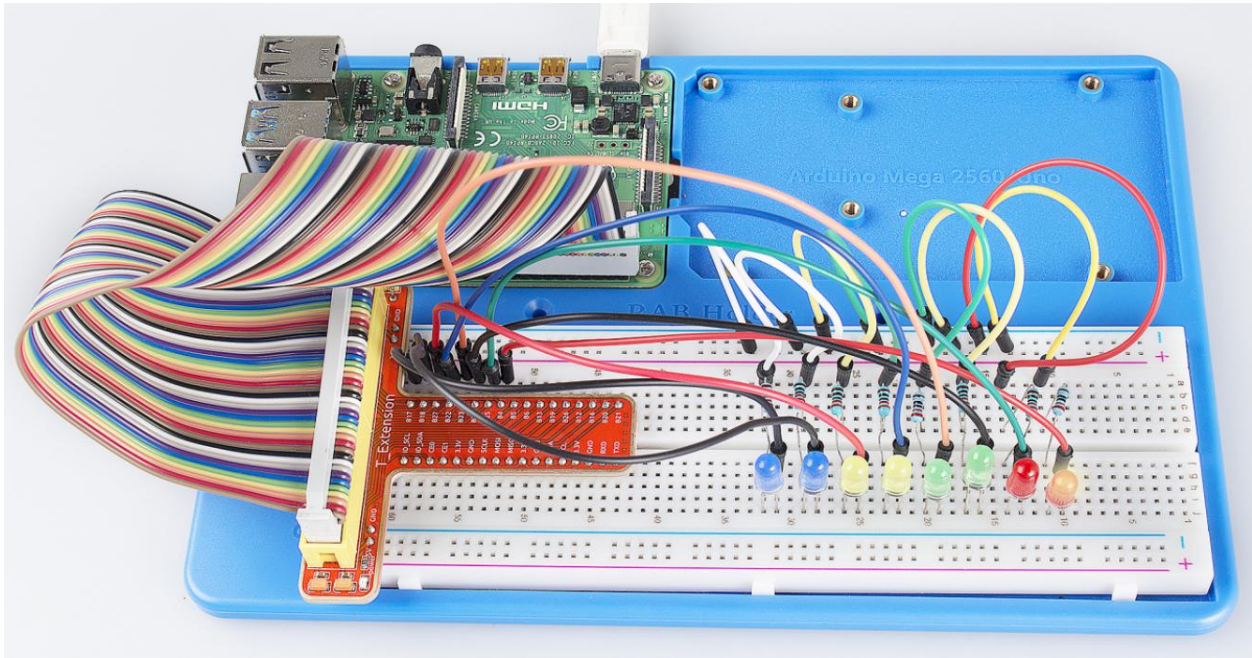
leds[LedPins.index(pin)] = 0 # Show which LED is on

time.sleep(0.1) # wait for 0.1s

GPIO.output(pin, GPIO.HIGH)
# After delaying, set it as high level to light up or turn off the LED.

leds[LedPins.index(pin)] = '-' # Show the led is off
```

You will see the eight LEDs lighten up one by one, and then dim in turn.



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
- 1 * T-Extension Board
- 1 * 40-Pin Cable

6.4.3 Principle

PWM

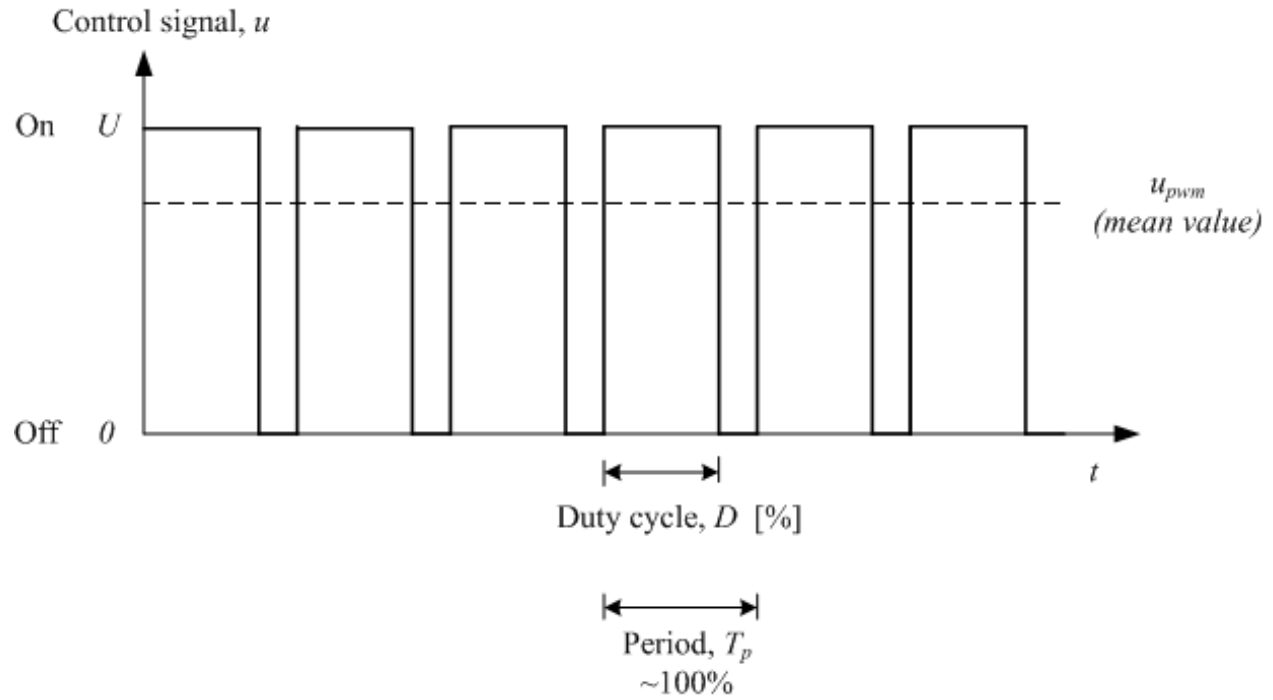
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (3.3 Volts) and off (0 Volts) by changing the portion of the 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.

Duty Cycle

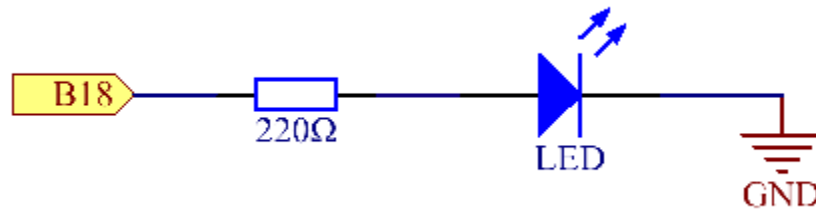
A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

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

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 Experimental Procedures

Step 1: Build the circuit.

fritzing

(continues on next page)

(continued from previous page)

```

}
softPwmCreate(LedPin, 0, 100);

printf("\n");
printf("\n");
printf("=====\n");
printf("|           Breath LED           |\n");
printf("| ----- |\n");
printf("|           LED connect to GPIO0   |\n");
printf("|           |\n");
printf("|           Make LED breath         |\n");
printf("|           |\n");
printf("|                               SunFounder |\n");
printf("=====\n");
printf("\n");
printf("\n");

while(1){
    printf("Breath on\n");
    for(i=0;i<=100;i++){
        softPwmWrite(LedPin, i);
        delay(20);
    }
    delay(1000);
    printf("Breath off\n");
    for(i=100;i>=0;i--){
        softPwmWrite(LedPin, i);
        delay(20);
    }
}

return 0;
}

```

Code Explanation

```

pinMode(LedPin, PWM_OUTPUT); // Set the I/O as pwm output

for(i=0;i<1024;i++)
{
    // i, as the value of pwm, increases progressively during 0-1024.

    pwmWrite(LedPin, i); // Write i into the LEDPin

    delay(2);
    // wait for 2ms, interval time between the changes indicates the speed of
    ↪breathing.
}
// the value of pwm add 1 every 2ms, when the value of pwm increases, the luminance
↪of the LED increases.

for(i=1023;i>=0;i--)
{

    pwmWrite(LedPin, i);
}

```

(continues on next page)

(continued from previous page)

```

    delay(2);
}
// the value of pwm minus 1 every 2ms, when the value of pwm decreases, the luminance
↳ of the LED decreases.

```

For Python Users:**Step 2:** Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 04_breathLed.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set #18 as LED pin
LedPin = 18

def print_message():
    print ("=====")
    print ("|                      Breath LED                      |")
    print ("|-----|")
    print ("|          LED connect to GPIO18          |")
    print ("|                                          |")
    print ("|          Make LED breath          |")
    print ("|                                          |")
    print ("|                                          |SunFounder|")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program..")
    #raw_input ("Press Enter to begin\\n")

def setup():
    global pLed
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,
    # and initial level to low (0v)

    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
    # Set pLed as pwm output and frequece to 1KHz
    pLed = GPIO.PWM(LedPin, 1000)
    # Set pLed begin with value 0

```

(continues on next page)

(continued from previous page)

```

pLed.start(0)

def main():
    print_message()
    # Set increase/decrease step
    step = 2
    # Set delay time.
    delay = 0.05
    while True:
        # Increase duty cycle from 0 to 100
        for dc in range(0, 101, step):
            # Change duty cycle to dc
            pLed.ChangeDutyCycle(dc)
            print (" ++ Duty cycle: %s" %dc)
            time.sleep(delay)
        time.sleep(1)
        # decrease duty cycle from 100 to 0
        for dc in range(100, -1, -step):
            # Change duty cycle to dc
            pLed.ChangeDutyCycle(dc)
            print (" -- Duty cycle: %s" %dc)
            time.sleep(delay)
        #time.sleep(1)

def destroy():
    # Stop pLed
    pLed.stop()
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

```

Code Explanation

```

GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
# Set LedPin as OUTPUT, initialize the pin as low level.

pLED = GPIO.PWM(LedPin, 1000) '''use PWM in the RPi.GPIO library. Set
LedPin as analog PWM output, the frequency as 1000Hz, assign these
configurations to pLed.'''

pLed.start(0) # Start pLed with 0% pulse width

time.sleep(0.05)

while True:

```

(continues on next page)

(continued from previous page)

```

# Increase duty cycle from 0 to 100

for dc in range(0, 101, step):
    # set dc from 0 to 100 in for loop. Set step to cycle.

    # Change duty cycle to dc

    pLed.ChangeDutyCycle(dc)
    # ChangeDutyCycle() function in pLED output pulse width 0~100% according to
    ↳the variable dc.

    print (" ++ Duty cycle: %s" %dc) # print information

    time.sleep(delay) '''it will delay after changing the pulse width for
    each time, this parameter can be modified to change the LED's lighting
    and dimming speed.'''

time.sleep(1)

# decrease duty cycle from 100 to 0

for dc in range(100, -1, -step):
    # the luminance of the LED decreases with each cycle.

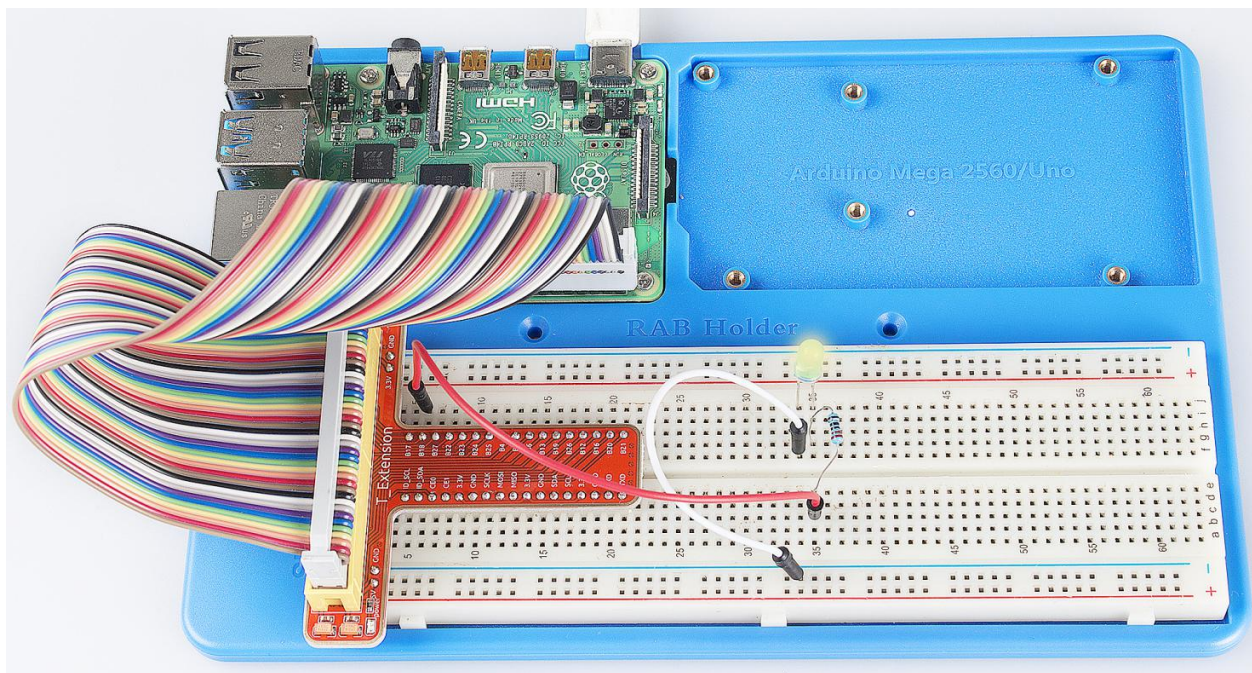
    # Change duty cycle to dc

    pLED.ChangeDutyCycle(dc) # same as the last for loop

    print (" -- Duty cycle: %s" %dc)

    time.sleep(delay)
    # Now you will see the gradual change of the LED luminance, between bright
    ↳and dim.

```



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

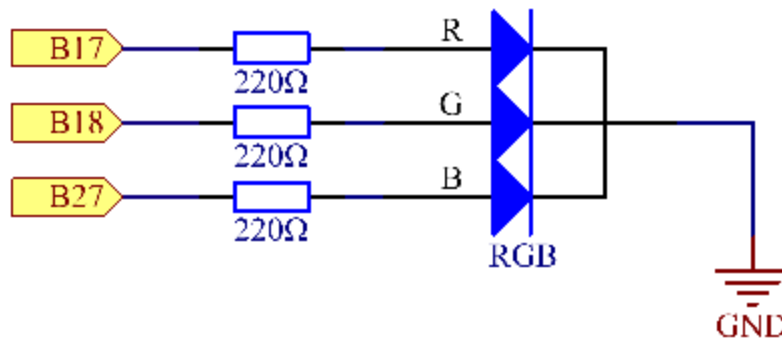
- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * RGB LED
- 3 * Resistor (220)
- Several jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.5.3 Principle

In this experiment, we will use a RGB. For details of RGB, please refer to the introduction of RGB LED in **Components Introduction**.

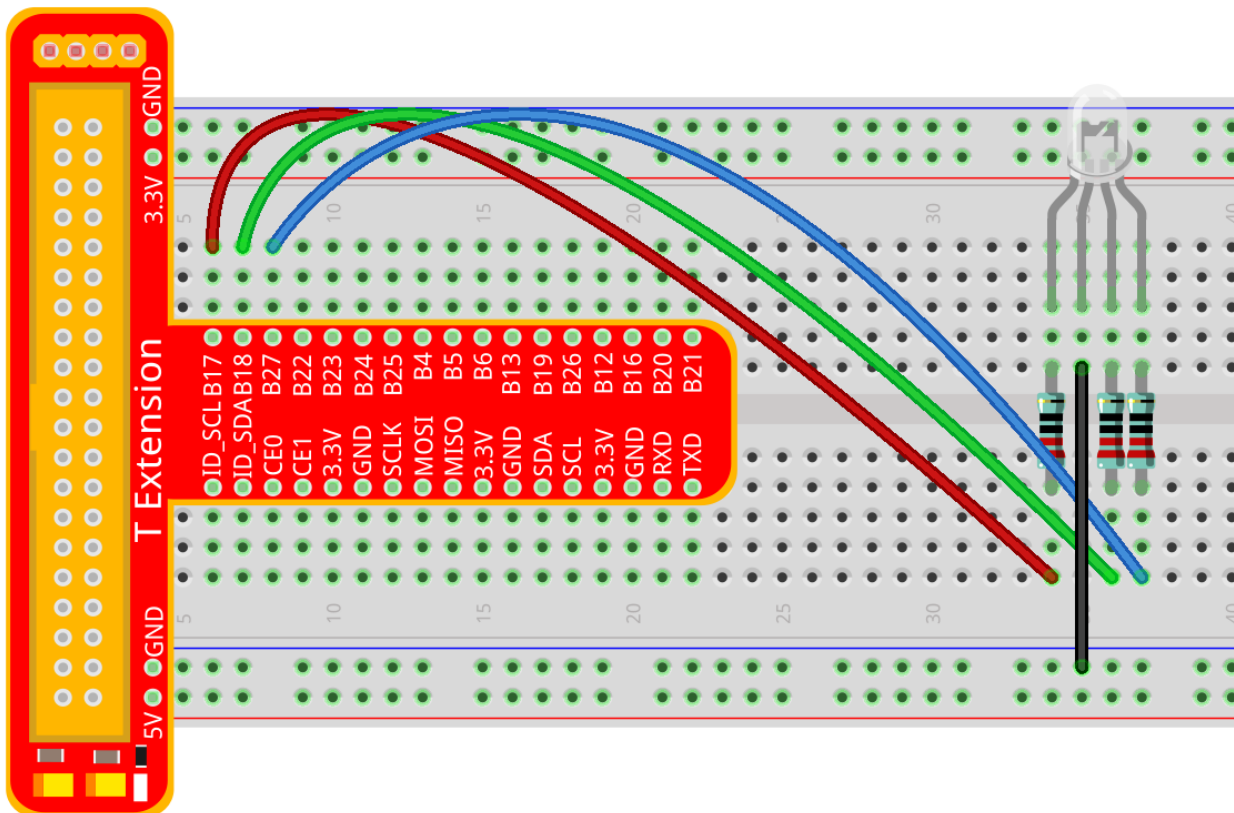


The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with 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.



6.5.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:**Step 2:** Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile the Code.

```
make 05_rgb
```

Step 4: Run the executable file above.

```
sudo ./05_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) {

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

    ledInit();

    printf("\n");
    printf("\n");
    printf("===== \n");
    printf("|           Breath LED           | \n");
    printf("|           -----           | \n");
```

(continues on next page)

(continued from previous page)

```

printf("|      Red Pin connect to GPIO0      |\n");
printf("|      Green Pin connect to GPIO1      |\n");
printf("|      Blue Pin connect to GPIO2       |\n");
printf("|                                     |\n");
printf("|  Make a RGB LED emits various color  |\n");
printf("|                                     |\n");
printf("|                                     |\n");
printf("|                                     |\n");
printf("=====|\n");
printf("\n");
printf("\n");

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

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

return 0;
}

```

Code Explanation

```

#include <softPwm.h>
// library used for realizing the pwm function of the software.

void ledInit(void)
{ // define function used for initializing I/O port to output for pwm.

    /* LedPinX refers to one pin. 0 is the minimum value and 100 is the
    maximum (as a percentage). The function is to use software to create a
    PWM pin, set its value between 0-100%.*/

    softPwmCreate(LedPinRed, 0, 100);

    softPwmCreate(LedPinGreen,0, 100);

    softPwmCreate(LedPinBlue, 0, 100);

    void ledColorSet(uchar r_val, uchar g_val, uchar b_val)
    { /* This function is to set the colors of the LED. Using RGB, the formal
    parameter r_val represents the luminance of the red one, g_val of the

```

(continues on next page)

(continued from previous page)

```

green one, b_val of the blue one. The three formal parameters' different
values corresponds to various colors. You can modify the 3 formal
parameters randomly to verify.*/

    softPwmWrite(LedPinRed, r_val);

    softPwmWrite(LedPinGreen, g_val);

    softPwmWrite(LedPinBlue, b_val);

}

ledColorSet(0xff,0x00,0x00); /* red calls the function defined before.
Write 0xff into LedPinRed and 0x00 into LedPinGreen and LedPinBlue. Only
the Red LED lights up after running this code. If you want to light up
LEDs in other colors, just modify the parameters.*/
}

```

For Python Users:**Step 2:** Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

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

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

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

def print_message():
    print ("=====")
    print ("|                      Breath LED                      |")
    print ("|-----|")
    print ("|      Red Pin connect to GPIO17      |")
    print ("|      Green Pin connect to GPIO18     |")
    print ("|      Blue Pin connect to GPIO27     |")
    print ("|")
    print ("| Make a RGB LED emits various color |")
    print ("|")
    print ("|                      SunFounder |")
    print ("=====\\n")

```

(continues on next page)

(continued from previous page)

```

print ("Program is running...")
print ("Please press Ctrl+C to end the program...")
#raw_input ("Press Enter to begin\n")

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

    # Set all led as pwm channel,
    # and frequece to 2KHz
    p_R = GPIO.PWM(pins['Red'], 2000)
    p_G = GPIO.PWM(pins['Green'], 2000)
    p_B = GPIO.PWM(pins['Blue'], 2000)

    # Set all begin with value 0
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

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

# Define a function to set up colors
# input color should be Hexadecimal with
# red value, blue value, green value.
def setColor(color):
    # Devide colors from 'color' variable
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0

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

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

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

def main():
    print_message()
    while True:
        for color in COLOR:
            setColor(color)
            time.sleep(0.5)

```

(continues on next page)

(continued from previous page)

```

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

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

```

Code Explanation

```

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

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

p_R = GPIO.PWM(pins['Red'], 2000)
# the same as the last lesson, here we configure the channels and frequencies of the
↪ 3 PWM.

p_G = GPIO.PWM(pins['Green'], 2000)

p_B = GPIO.PWM(pins['Blue'], 2000)

p_R.start(0)
# the same as the last lesson, the PWM of the 3 LEDs begin with 0.

p_G.start(0)

p_B.start(0)

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

def setColor(color): # configures the three LEDs' luminance with the inputted color
↪ value .

    R_val = (color & 0xFF0000) >> 16
    # these three lines are used for analyzing the col variables

```

(continues on next page)

(continued from previous page)

```
G_val = (color & 0x00FF00) >> 8
# assign the first two values of the hexadecimal to R, the middle two assigned to
↪G

B_val = (color & 0x0000FF) >> 0
# assign the last two values to B, please refer to the shift operation of the
↪hexadecimal for details.

R_val = MAP(R_val, 0, 255, 0, 100)
# use map function to map the R,G,B value among 0~255 into PWM value among 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)
# Assign the mapped duty cycle value to the corresponding PWM channel to change
↪the luminance.

p_G.ChangeDutyCycle(G_val)

p_B.ChangeDutyCycle(B_val)

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

    setColor(color) # change the color of the RGB LED

    time.sleep(0.5)
# set delay for 0.5s after each color changing. Modify
# this parameter will changed the LED's color changing rate.
```

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

6.6 Lesson 6 Buzzer

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

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * Buzzer (Active)
- 1 * PNP transistor (8550)
- 1 * Resistor (1K)
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.6.3 Principle

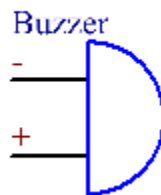
Buzzer

As a type of electronic buzzer with integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms 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.



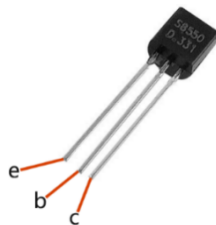
The difference between an active buzzer and a passive buzzer is: An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not beep if DC signals are used; instead, you need to use square waves whose frequency 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.

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



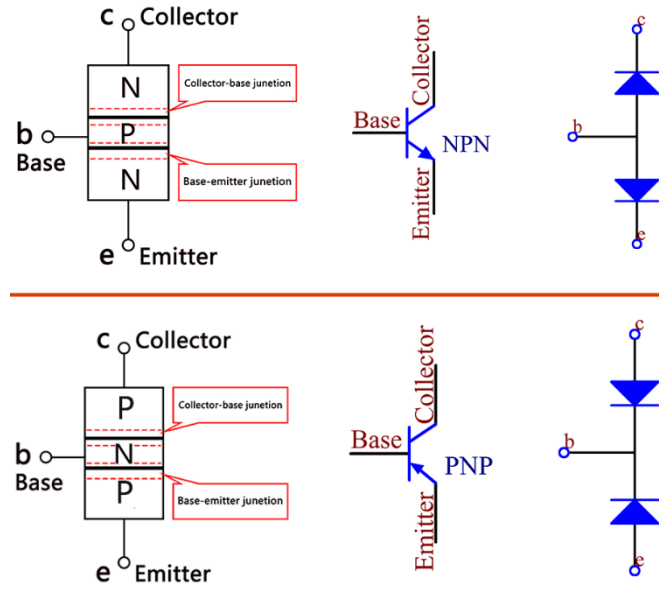
You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

Transistor



The transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used as a non-contact switch. A transistor is a three-layer structure composed of P-type or N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are all N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier.

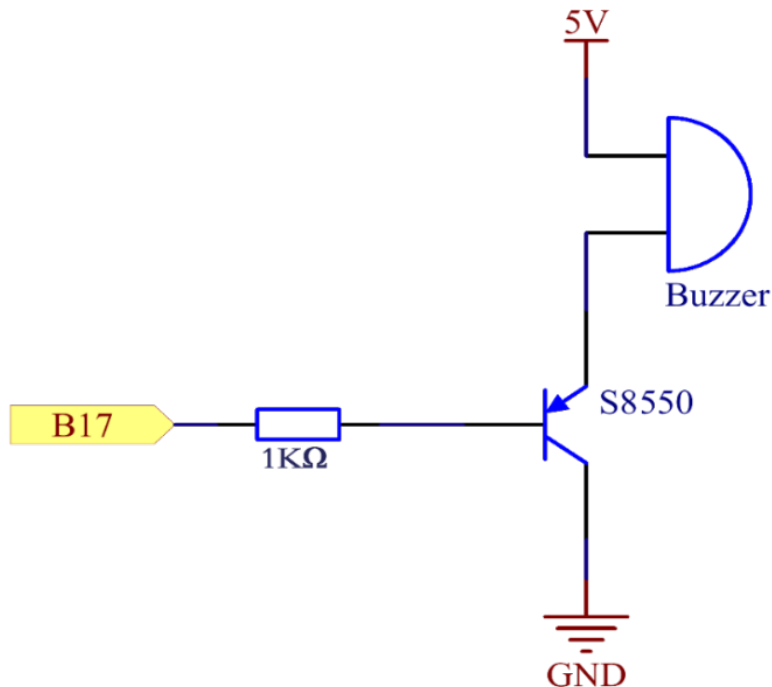
From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The arrow in the circuit symbol indicates the direction of emitter junction. Transistors can be divided into two kinds: the NPN and PNP one. The former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.



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

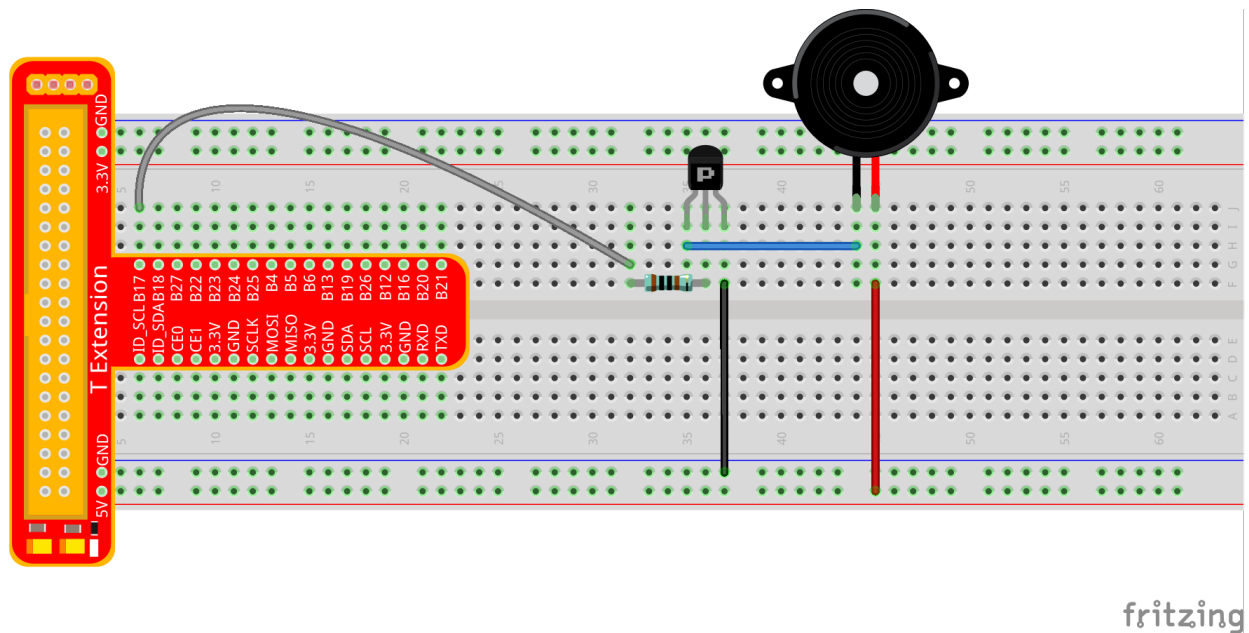
6.6.4 The Schematic Diagram

Principle: In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the B17 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 poles of the buzzer: The one with + label is the positive pole and the other is the negative.)



For C Language Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile the Code.

```
make 06_beep
```

Step 4: Run the executable file above.

```
sudo ./06_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 message to screen
        printf("setup wiringPi failed !");
    }
}
```

(continues on next page)

(continued from previous page)

```

    return 1;
}

pinMode(BeepPin, OUTPUT);    //set GPIO0 output

printf("\n");
printf("\n");
printf("=====\n");
printf("|                Beep                |\n");
printf("|-----|                |\n");
printf("|          Buzzer connect to GPIO0          |\n");
printf("|                |\n");
printf("|          Make Buzzer beep          |\n");
printf("|                |\n");
printf("|                SunFounder|\n");
printf("=====\n");
printf("\n");
printf("\n");

while(1){
    //beep on
    printf("Buzzer on\n");
    digitalWrite(BeepPin, LOW);
    delay(100);
    printf("Buzzer off\n");
    //beep off
    digitalWrite(BeepPin, HIGH);
    delay(100);
}

return 0;
}

```

Code Explanation

`digitalWrite(BeepPin, LOW);` /* We use an active buzzer in this experiment, so it will make sound automatically when connecting to the direct current. This sketch is to set the I/O port as low level (0V), thus to manage the transistor and make the buzzer beep.*/

`digitalWrite(BeepPin, HIGH);` /* To set the I/O port as high level(5V), thus the transistor is not energized and the buzzer doesn't beep.*/

For Python Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

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

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set #17 as buzzer pin
BeepPin = 17

def print_message():
    print ("=====")
    print ("|                               Beep                               |")
    print ("|-----|")
    print ("|          Buzzer connect to GPIO17          |")
    print ("|-----|")
    print ("|          Make Buzzer beep          |")
    print ("|-----|")
    print ("|                               SunFounder|")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\\n")

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

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

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

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

```

(continues on next page)

(continued from previous page)

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

Code Explanation

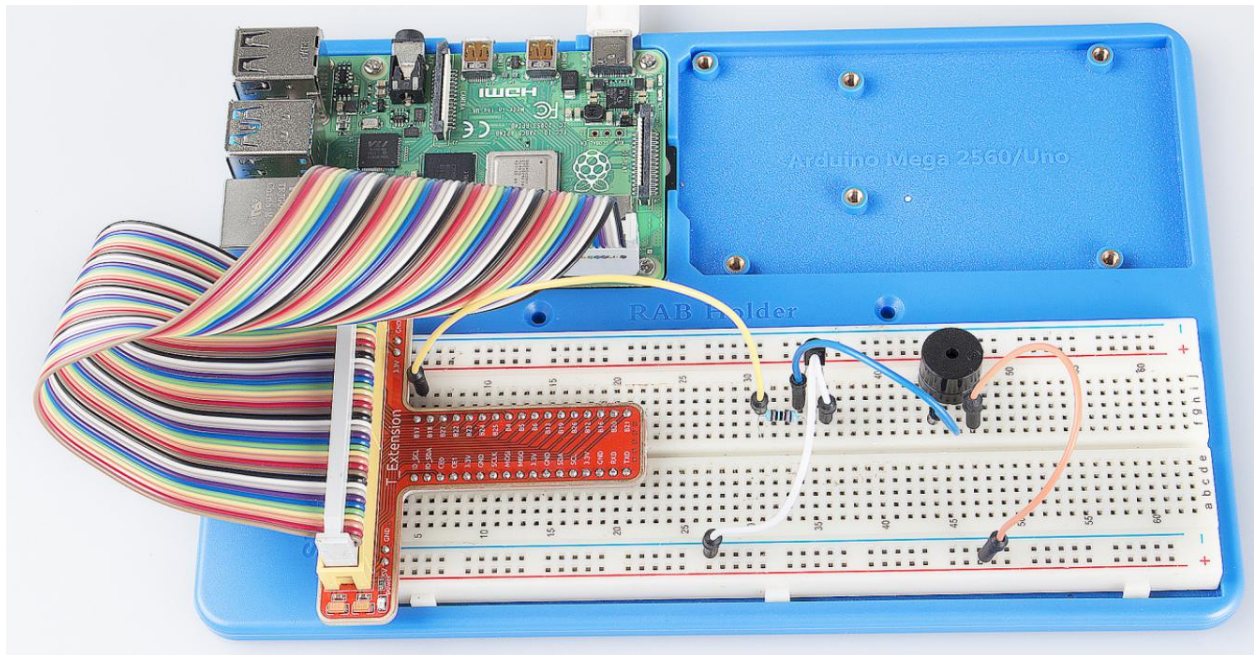
```
GPIO.output(BeepPin, GPIO.LOW) # Set the buzzer pin as low level.

time.sleep(0.1) # Wait for 0.1 second. Change the switching frequency by
#changing this parameter. Note: Not the sound frequency. Active Buzzer
#cannot change sound frequency.

GPIO.output(BeepPin, GPIO.HIGH) # close the buzzer

time.sleep(0.1)
```

Now, you should hear the buzzer make sounds.



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 Relay

6.7.1 Introduction

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

6.7.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * Relay
- 1 * LED
- 1 * Resistor (220)
- 1 * Resistor (1K)
- 1 * NPN Transistor
- 1 * Diode (Rectifier)
- Several jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.7.3 Principle

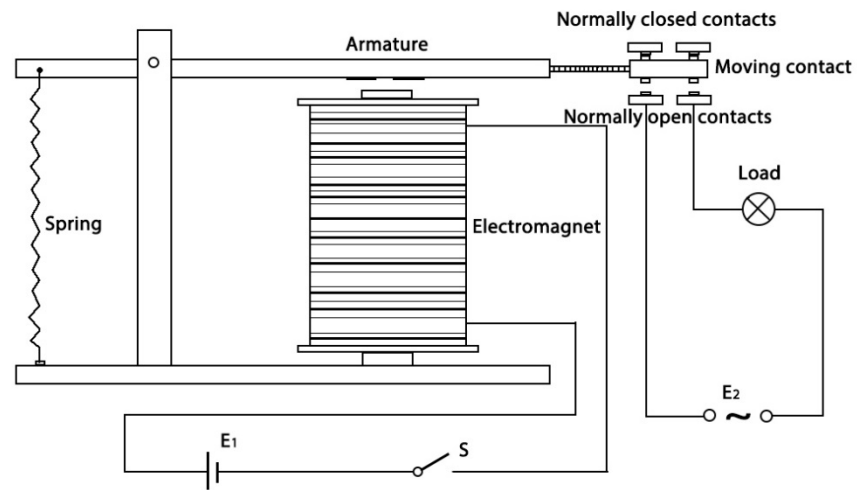
Relay

There are 5 parts in every relay:

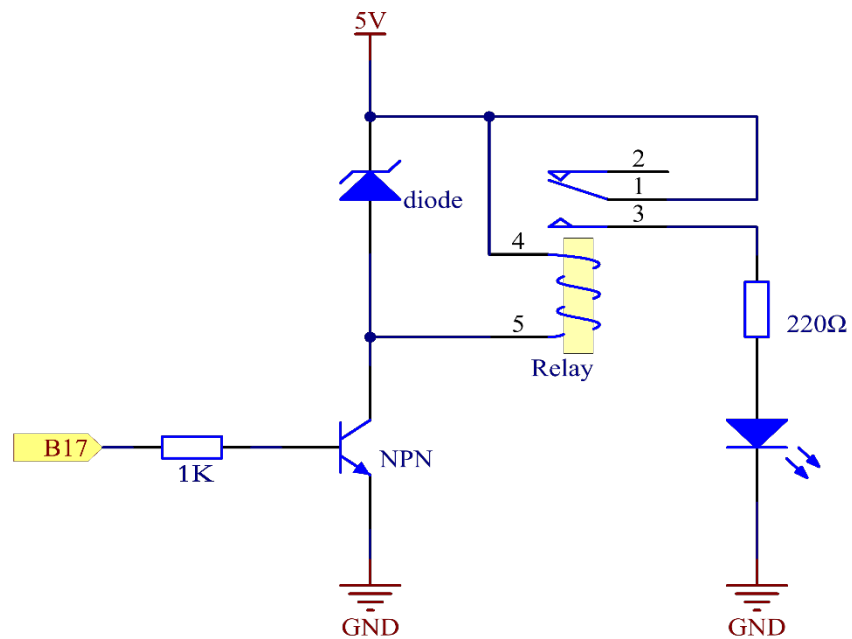
1. **Electromagnet** – It consists of an iron core wound by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.
2. **Armature** – The movable magnetic strip is known as armature. When current flows through them, the coil is energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).
3. **Spring** – When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.
4. Set of electrical **contacts** – There are two contact points:
 - Normally open - connected when the relay is activated, and disconnected when it is inactive.
 - Normally close – not connected when the relay is activated, and connected when it is inactive.
5. Molded frame – Relays are covered with plastic for protection.

6.7.4 Working of Relay

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

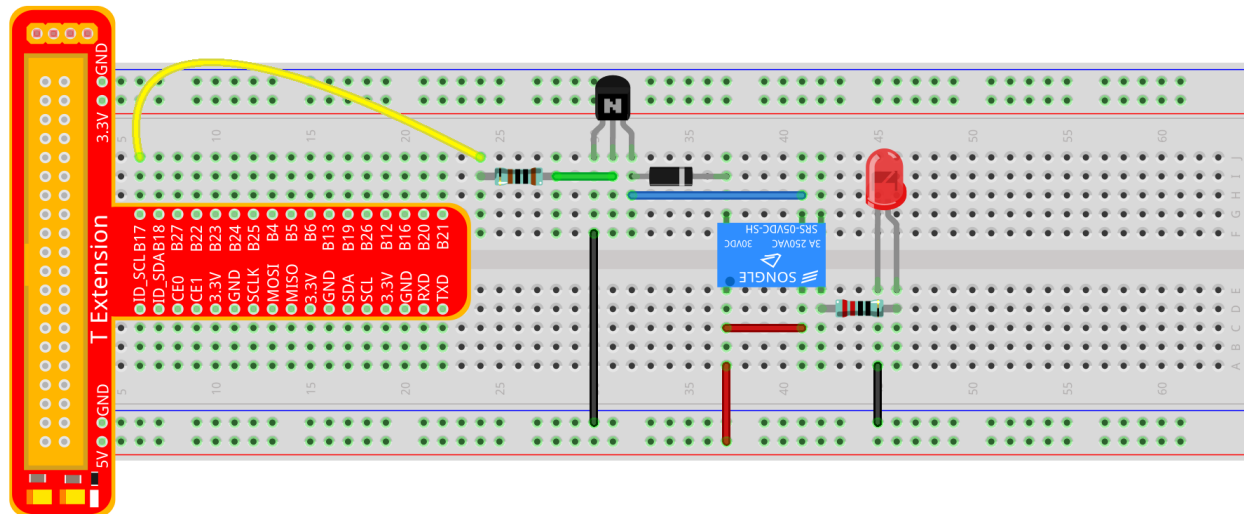


6.7.5 Schematic Diagram:



6.7.6 Experimental Procedures

Step 1: Build the circuit.



fritzing

For C Language Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile the Code.

```
make 07_relay
```

Step 4: Run the executable file above.

```
sudo ./07_relay
```

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

#define RelayPin 0

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

(continues on next page)

(continued from previous page)

```

pinMode(RelayPin, OUTPUT);    //set GPIO0 output

printf("\n");
printf("\n");
printf("=====\n");
printf("|               Relay               |\n");
printf("| ----- |\n");
printf("| GPIO0 connect to relay's control pin |\n");
printf("| led connect to relay's NormalOpen pin|\n");
printf("| 5v connect to relay's com pin      |\n");
printf("| |\n");
printf("|      Make relay to contral a led    |\n");
printf("| |\n");
printf("|                               SunFounder|\n");
printf("=====\n");
printf("\n");
printf("\n");

while(1){
    // Tick
    printf(".....Relay Close\n");
    digitalWrite(RelayPin, LOW);
    delay(1000);
    // Tock
    printf("Relay Open.....\n");
    digitalWrite(RelayPin, HIGH);
    delay(1000);
}

return 0;
}

```

Code Explanation

`digitalWrite(relayPin, LOW);` /* Set the I/O port as LOW level (5V), thus the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens.*/

`digitalWrite(relayPin, HIGH);` /* set the I/O port as HIGH level (0V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes.*/

For Python Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 07_relay.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# GPIO00 connect to relay's control pin
# led connect to relay's NormalOpen pin
# 5v connect to relay's com pin
# Set #17 as contral pin
relayPin = 17

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|                      Relay                      |")
    print ("| ----- |")
    print ("| GPIO17 connect to relay's control pin |")
    print ("| led connect to relay's NormalOpen pin|")
    print ("| 5v connect to relay's com pin      |")
    print ("|                                     |")
    print ("|      Make relay to contral a led    |")
    print ("|                                     |")
    print ("|                                     SunFounder|")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program..")
    #raw_input ("Press Enter to begin\\n")

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

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

# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED

```

(continues on next page)

(continued from previous page)

```

GPIO.output(relayPin, GPIO.HIGH)
# Release resource
GPIO.cleanup()

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

```

Code Explanation

```

GPIO.output(relayPin, GPIO.LOW)
# Set the pins of the transistor as low level to let the relay open.

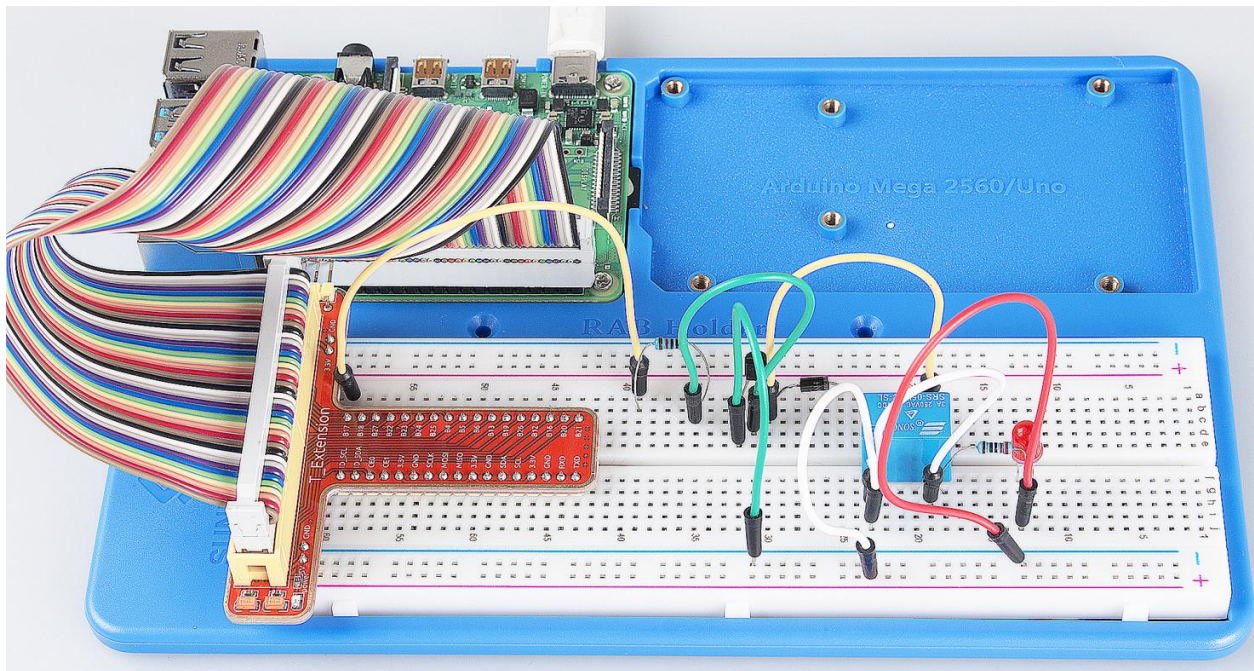
time.sleep(1) # wait for 1 second. Change the switching frequency of the
#relay by changing this parameter. Note: Relay is a kind of metal dome
#formed in mechanical structure. So its lifespan will be shortened under
#high-frequency using.

GPIO.output(relayPin, GPIO.HIGH)
# Set the pins of transistor as HIGH level to actuate the relay.

time.sleep(1)

```

Now, connect a device of high voltage, and the relay will close and the LED will light up; connect one of low voltage, and it will open and the LED will go out. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.



6.8 Lesson 8 4N35

6.8.1 Introduction

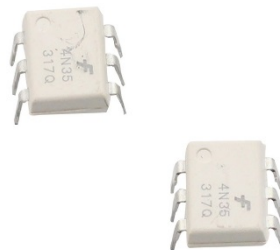
The 4N35 is an optocoupler for general purpose application. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor. When the input signal is applied to the LED in the input terminal, the LED lights up. After receiving the light signal, the light receiver then converts it into electrical signal and outputs the signal directly or after amplifying it into a standard digital level. Thus, the transition and transmission of electricity-light-electricity is completed. Since light is the media of the transmission, meaning the input terminal and the output one are isolated electrically, this process is also be known as electrical isolation.

6.8.2 Components

- 1 * Raspberry Pi
- 1 * 4N35
- 1 * LED
- 1 * 220 Ohm Resistor
- 1* 1k Ohm Resistor
- Some jump wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

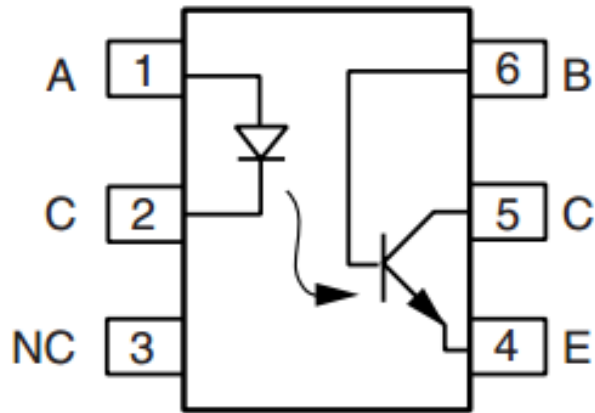
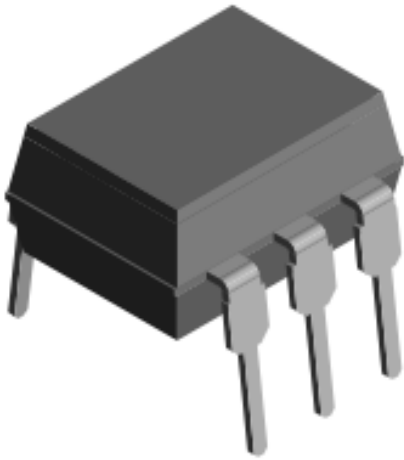
6.8.3 Principle

4N35



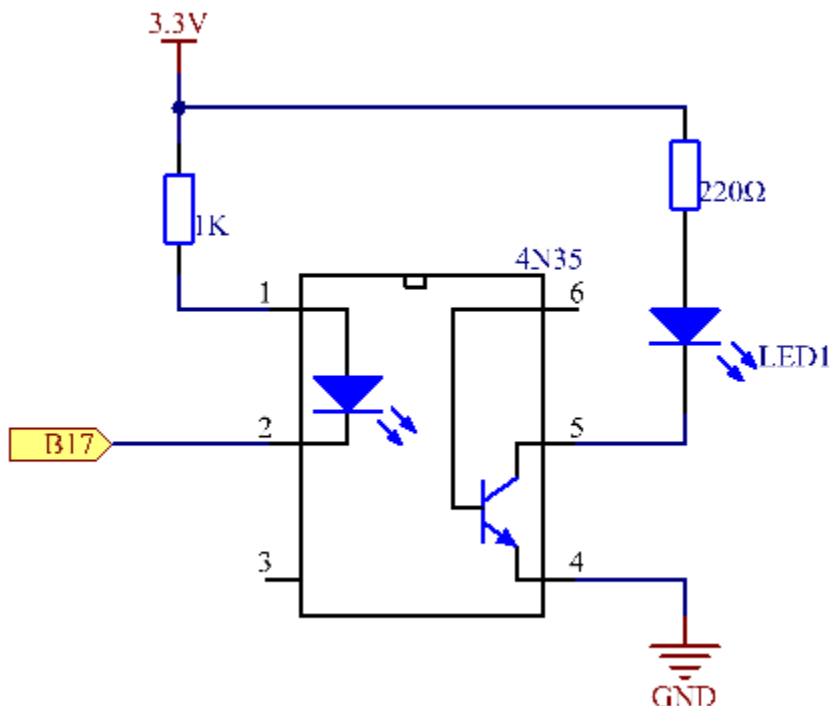
The 4N35 is an optocoupler for general purpose application. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor.

What an optocoupler does is to break the connection between signal source and signal receiver, so as to stop electrical interference. In other words, it is used to prevent interference from external electrical signals. 4N35 can be used in AV conversion audio circuits. Broadly it is widely used in electrical insulation for a general optocoupler.



See the internal structure of 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays. This can be done to control the load connected to the phototransistor. Even when the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

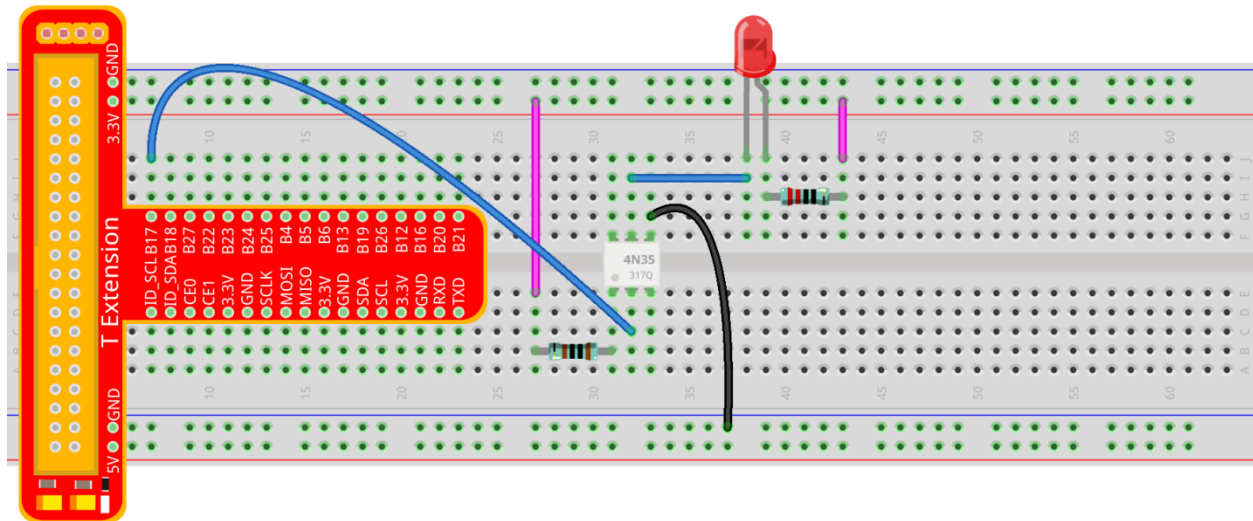
6.8.4 The Schematic Diagram:



Principle: In this experiment, use an LED as the load connected to the NPN phototransistor. Connect pin 2 of 4N35 to pin B17, pin 1 connects a 1K current-limiting resistor and then a 3.3V. Connect pin 4 to GND, and pin 5 to the cathode

of the LED. Then hook the anode of the LED to 3.3V after connecting with a 220 Ohm resistor. When in program, a LOW level is given to pin B17, the infrared LED will emit infrared rays. Then the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus turning on the LED. Also you can control the LED by circuits only – connect pin 2 to ground and it will brighten.

Step 1: Build the circuit.



fritzing

For C Language Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile the Code.

```
make 08_4N35
```

Step 4: Run the executable file above.

```
sudo ./08_4N35
```

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 _4N35Pin          0

int main(void)
{
```

(continues on next page)

(continued from previous page)

```

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

pinMode(_4N35Pin, OUTPUT);

printf("\n");
printf("\n");
printf("=====\n");
printf("|           4N35           |\n");
printf("| ----- |\n");
printf("| LED connect to 4N35 pin5; |\n");
printf("| gpio0 connect to 4N35 pin2 |\n");
printf("| |\n");
printf("| 4N35 to contral led blinking. |\n");
printf("| |\n");
printf("| SunFounder |\n");
printf("=====");
printf("\n");
printf("\n");

while(1){
    // LED on
    digitalWrite(_4N35Pin, LOW);
    printf("...LED on\n");
    delay(500);
    // LED off
    digitalWrite(_4N35Pin, HIGH);
    printf("LED off...\n");
    delay(500);
}

return 0;
}

```

Code Explanation

`digitalWrite(_4N35Pin, LOW);` /* set the I/O port as low level (0V), thus the optocoupler is energized, and the pin connected to LED conducts to the 0V. Then the LED lights up.*/

`delay(500);`
 // optocoupler is a kind of electronic device and there is no limitation on its on-off frequency.

`digitalWrite(_4N35Pin, HIGH);` /* set I/O port as high level (3.3V), thus the optocoupler is not energized, and the pin connected to LED cannot conduct to the 0V. Then the LED goes out.*/

For Python Users:

Step 2: Open the code file.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 08_4N35.py
```

Code

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

if version_info.major == 3:
    raw_input = input

# Set #17 as 4N35 pin
Pin_4N35 = 17

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|                4N35                |")
    print ("| ----- |")
    print ("|      LED connect to 4N35 pin5;      |")
    print ("|      gpio17 connect to 4N35 pin2;    |")
    print ("|                                     |")
    print ("|      4N35 to contral led blinking.   |")
    print ("|                                     |")
    print ("|                                     SunFounder |")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program..")
    #raw_input ("Press Enter to begin\\n")

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set Pin_4N35's mode to output,
    # and initial level to High(3.3v)
    GPIO.setup(Pin_4N35, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():
    # Print messages
    print_message()
    while True:
        print ("...LED ON")
        # Turn on LED
        GPIO.output(Pin_4N35, GPIO.LOW)
        time.sleep(0.5)
        print ("LED OFF...")
        # Turn off LED
```

(continues on next page)

(continued from previous page)

```
GPIO.output(Pin_4N35, GPIO.HIGH)
time.sleep(0.5)

# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(Pin_4N35, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

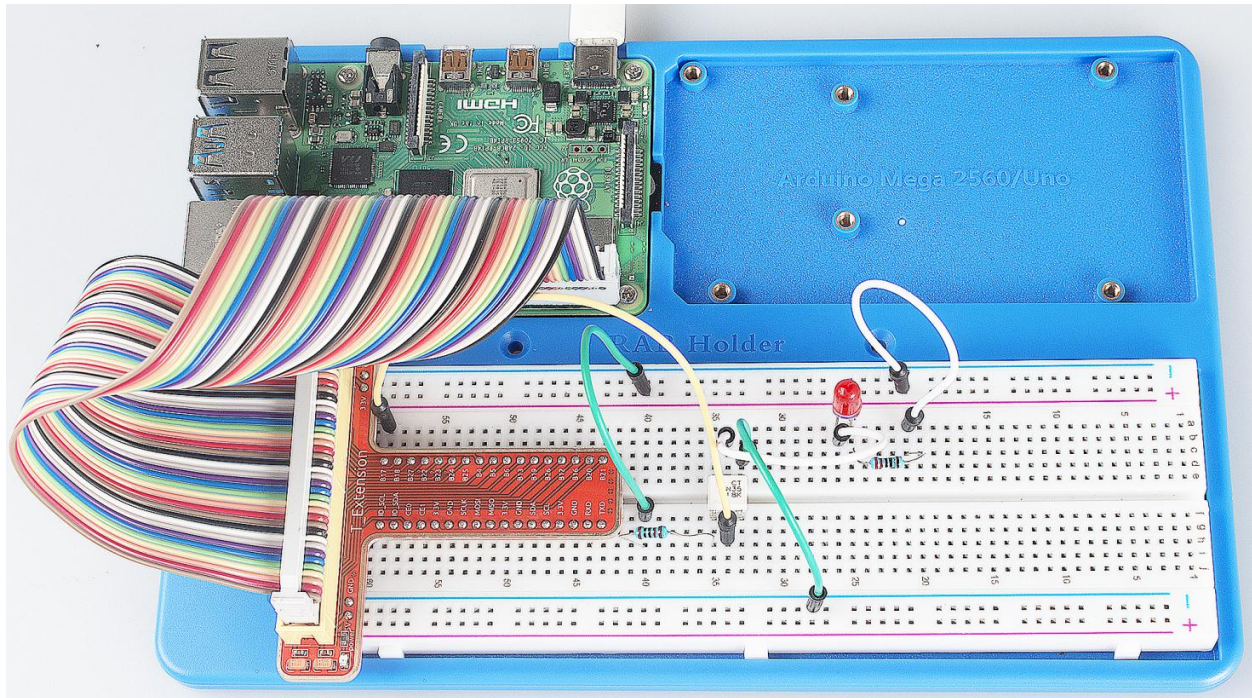
```
GPIO.output(Pin_4N35, GPIO.LOW) # set the pins of optocoupler as low
#level, thus the optocoupler is energized, and the pin connected to LED
#conducts to the 0V. Then the LED lights up.

time.sleep(0.5) #wait for 0.5 second. The on-off frequency of the
#optocoupler can be changed by modifying this parameter.

GPIO.output(Pin_4N35, GPIO.HIGH) # set the pins of optocoupler as high
#level, thus the optocoupler is disconnected, and the pin connected to
#LED break the connection to the 0V. Then the LED goes out.

time.sleep(0.5)
```

You will see the LED blinks.



6.9 Lesson 9 Ne555

6.9.1 Introduction

If you ask anyone in the know to rank the most commonly and widely used IC, the famous 555 time base IC would certainly be at the top of the list. The 555 – a mixed circuit composed of analog and digital circuits – integrates analogue and logical functions into an independent IC, and hence tremendously expands the application range of analog integrated circuits. The 555 is widely used in various timers, pulse generators, and oscillators.

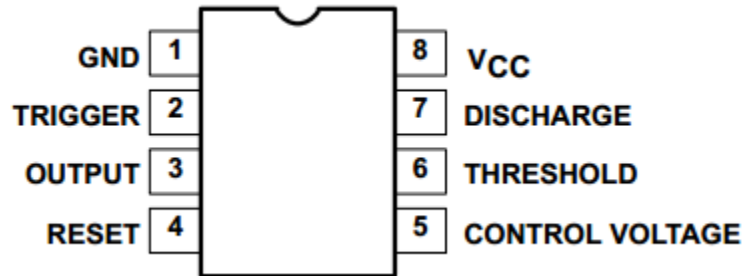
6.9.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * NE555
- 2 * 104 ceramic capacitor
- 1 * Potentiometer (50K)
- 1 * Resistor (10K)
- 1 * USB cable
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.9.3 Principle

The 555 IC was originally used as a timer, hence the name 555 time base circuit. It is now widely used in various electronic products because of its reliability, convenience, and low price. The 555 is a complex hybrid circuit with dozens of components such as a divider, comparator, basic R-S trigger, discharge tube, and buffer.

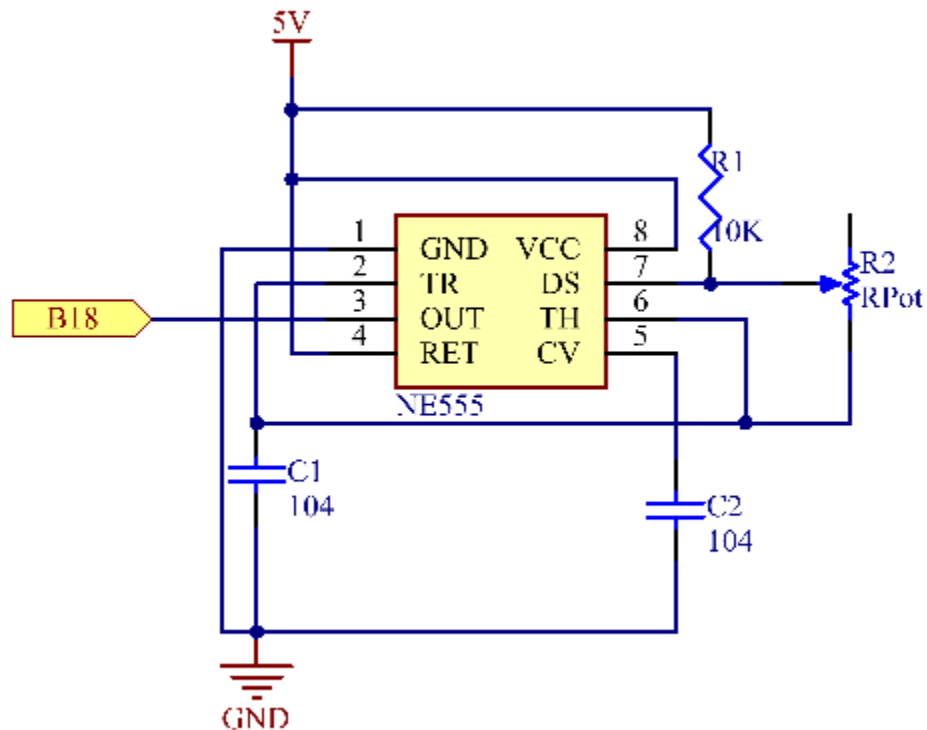
555 chip pins are introduced as follows:



As shown in the picture, the 555 IC is dual in-line with the 8-pin package. Thus:

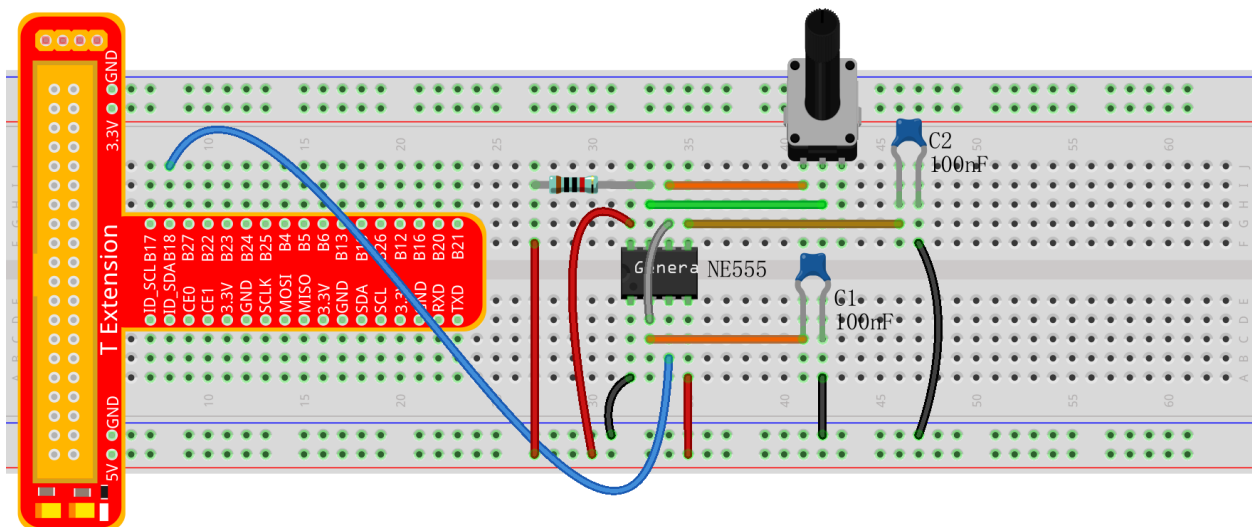
- Pin 1 (GND): the ground;
- Pin 2 (TRIGGER): the input of lower comparator;
- Pin 3 (OUTPUT): having two states of 0 and 1 decided by the input electrical level;
- Pin 4 (RESET): output low level when supplied a low one;
- Pin 5 (CONTROL VOLTAGE): changing the upper and lower level trigger values;
- Pin 6 (THRESHOLD): the input of upper comparator;
- Pin 7 (DISCHARGE): having two states of suspension and ground connection also decided by input, and the output of the internal discharge tube;
- Pin 8 (VCC): the power supply;

6.9.4 The Schematic Diagram



6.9.5 Experimental Procedures

Step 1: Build the circuit.



fritzing

For C Language Users:**Step 2:** Go to the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 09_ne555
```

Step 4: Run the executable file above.

```
sudo ./09_ne555
```

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 1

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;
    }

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|           Ne555           |\n");
    printf("| ----- |\n");
    printf("| Output pin of ne555 connect to gpio1; |\n");
    printf("| |\n");
    printf("| Count the pulses procude by NE555. |\n");
    printf("| |\n");
    printf("|                               SunFounder|\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    delay(2000);
```

(continues on next page)

(continued from previous page)

```
pinMode(Pin0, INPUT);
pullUpDnControl(Pin0, PUD_UP);
wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR);

while(1){
    printf("Current pluse number is : %d, %d\n", globalCounter, digitalRead(Pin0));
    dealy(100);
}

return 0;
}
```

Code Explanation

```
static volatile int globalCounter = 0 ;
// a static integer variable to store the pulse count

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

wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR); /* set an interrupt
here and the signal is falling edge for Pin 0. When the interrupt happens,
execute the function exInt0_ISR(), and the pulse count will add 1.*/

while(1)
{
    // if no interrupt happens, the pulse count will stay and just print it.

    printf("Current pulse number is : %d\n", globalCounter);
    delay(100);
}
```

For Python Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 09_ne555.py
```

Code

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

if version_info.major == 3:
    raw_input = input
```

(continues on next page)

(continued from previous page)

```

# ne555 pin3 connect to BCM GPIO18
SigPin = 18      # BCM 18

g_count = 0

def print_msg():
    print ("=====");
    print ("|                               Ne555                               |");
    print ("| -----|-----|");
    print ("| Output pin of ne555 connect to gpio18;|");
    print ("|                               |");
    print ("| Count the pulses procude by NE555. |");
    print ("|                               |");
    print ("|                               SunFounder|");
    print ("=====\\n");
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\\n")

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

def setup():
    GPIO.setmode(GPIO.BCM)      # Numbers GPIOs by physical location
    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 main():
    print_msg()
    while True:
        print ("g_count = %d" % g_count)
        time.sleep(0.001)

def destroy():
    GPIO.cleanup()      # Release resource

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

```

Code Explanation

```

g_count = 0 # a global variable used to store the pulse count

def count(ev=None): # define a function to be run when an interrupt happens

    global g_count # this function will change the value of the global
    # variable g_count, thus here we add global before it.

```

(continues on next page)

(continued from previous page)

```
g_count += 1

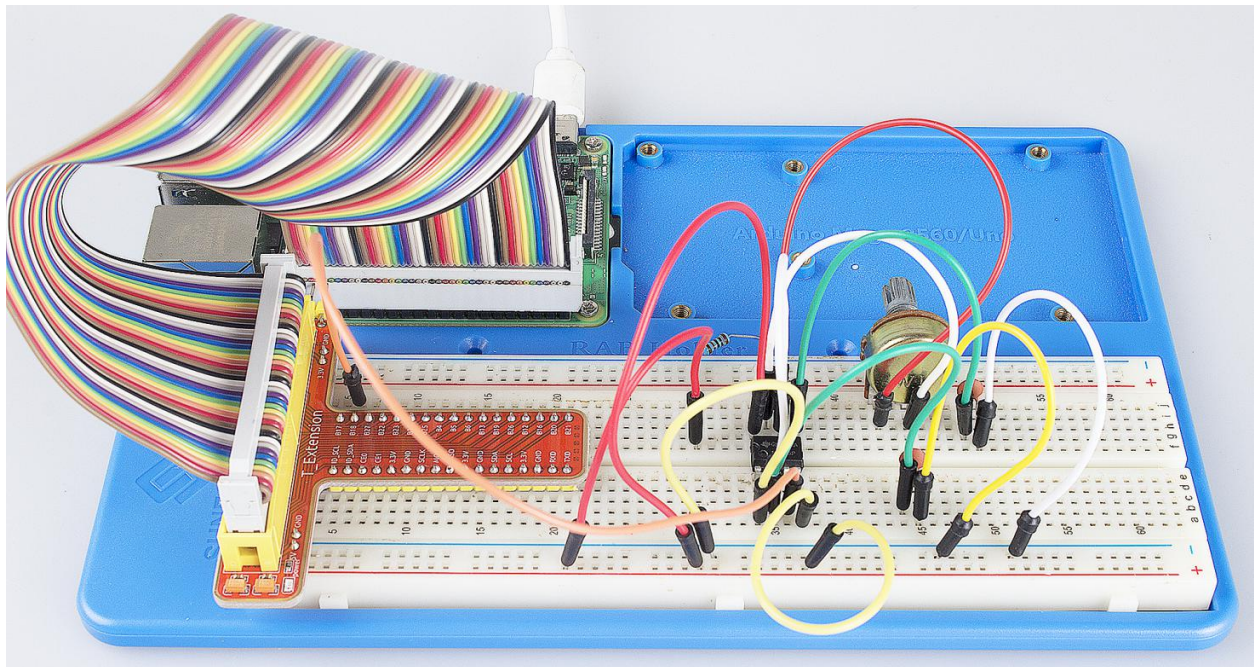
GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # set an
# interrupt here and the interrupt signal is a rising edge for Pin Sig. It
# will run the function count() accordingly

while True: # wait for the interrupt

    print ("g_count = %d" % g_count) # print the information

    time.sleep(0.001)
```

Now you can see the number of square waves printed. Spin the potentiometer and the value will decrease or increase.



6.10 Lesson 10 Slide Switch

6.10.1 Introduction

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

6.10.2 Components

- 1 * Raspberry Pi
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable
- 1 * Breadboard
- 1 * Slide Switch
- 2 * LED
- 3 * Resistors (220,10k)
- 1 * USB cable
- Jumper wires
- 1 * 104 Capacitor Ceramic

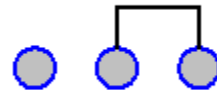
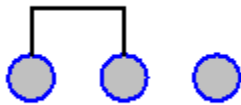
6.10.3 Principle

Slide Switch

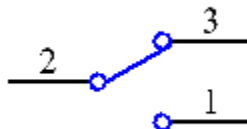


A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The Slide Switch is commonly used in low-voltage circuit. It features flexibility and stability, and widely applies in electric instruments and electric toys.

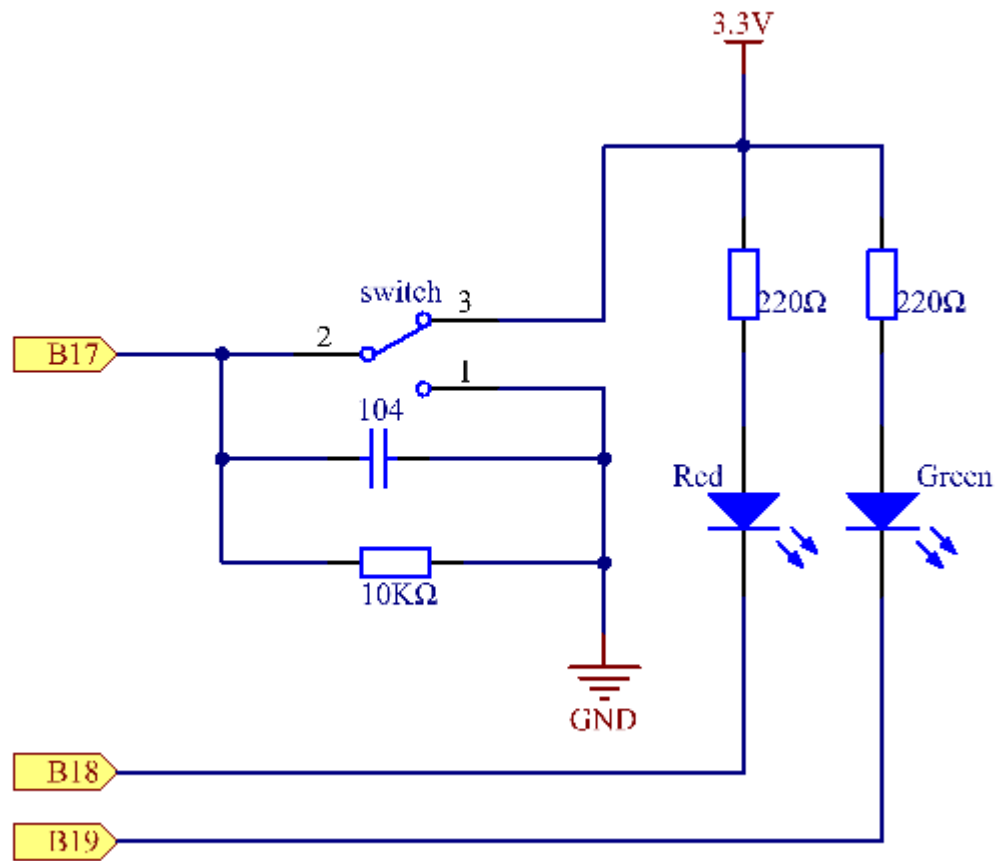
How it works: Use the middle pin as the fixed one. When you pull the slide to the left, the left two pins are connected; to the right, the right two pins connected. Thus, it connects and disconnects circuits as a switch. See the figure below:



The circuit symbol of the slide switch is as shown below. 2 in the figure means the middle pin.

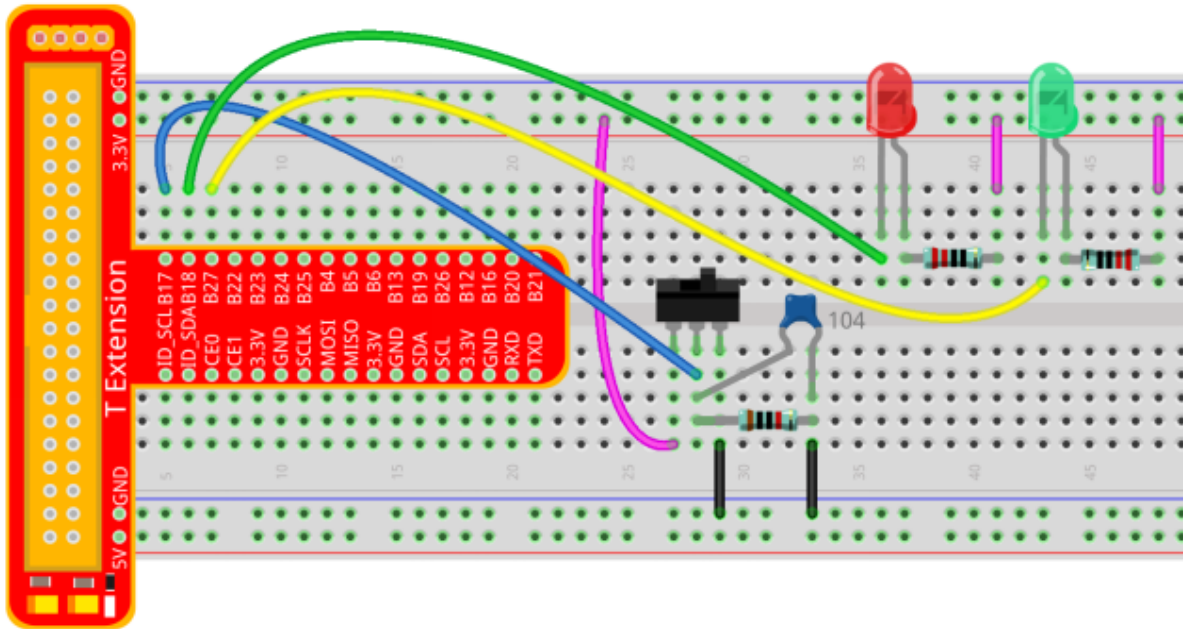


Principle: Connect the middle pin of the Slide Switch to B17, and two LEDs to pin B18 and B27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.



6.10.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Go to the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 10_slideSwitch
```

Step 4: Run the executable file above.

```
sudo ./10_slideSwitch
```

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to [C code is not working?](#).

Code

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

#define slidePin      0
#define led1          1
#define led2          2

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

(continues on next page)

(continued from previous page)

```

    return 1;
}

pinMode(slidePin, INPUT);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);

printf("\n");
printf("\n");
printf("=====\\n");
printf("|                Slide Switch                |\\n");
printf("| -----|-----|\\n");
printf("|      Middle pin of slide switch      |\\n");
printf("|      connect to gpio0                |\\n");
printf("|                |\\n");
printf("|slide switch to contral which led on. |\\n");
printf("|                |\\n");
printf("|                SunFounder|\\n");
printf("=====");
printf("\\n");
printf("\\n");

while(1){
    // slide switch high, led1 on
    if(digitalRead(slidePin) == 1){
        digitalWrite(led1, LOW);
        digitalWrite(led2, HIGH);
        printf("LED1 on\\n");
        delay(100);
    }
    // slide switch low, led2 on
    if(digitalRead(slidePin) == 0){
        digitalWrite(led2, LOW);
        digitalWrite(led1, HIGH);
        printf(".....LED2 on\\n");
        delay(100);
    }
}

return 0;
}

```

Code Explanation

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

```

if(digitalRead(slidePin) == 1)
{
    digitalWrite(led1, LOW);

    digitalWrite(led2, HIGH);

    printf("LED1 on\\n");
    delay(100);
}

```

(continues on next page)

(continued from previous page)

```

}

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

if(digitalRead(slidePin) == 0)
{
    digitalWrite(led2, LOW);

    digitalWrite(led1, HIGH);

    printf(".....LED2 on\n");
    delay(100);
}

```

For Python Users:**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 10_slideSwitch.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set #17 as slide switch pin, #18 as led1 pin, #27 as led2 pin
slidePin = 17
led1Pin = 18
led2Pin = 27

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|                Slide Switch                |")
    print ("|-----|")
    print ("|      Middle pin of slide switch      |")
    print ("|      connect to gpio17;              |")
    print ("|")
    print ("|slide switch to contral which led on. |")
    print ("|")
    print ("|                SunFounder|")
    print ("=====\\n")

```

(continues on next page)

(continued from previous page)

```

print ("Program is running...")
print ("Please press Ctrl+C to end the program...")
#raw_input ("Press Enter to begin\n")

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

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

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

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

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

```

Code Explanation

```

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

```

(continues on next page)

(continued from previous page)

```
if GPIO.input(slidePin) == 1:

    print (" LED1 ON ")

    GPIO.output(led1Pin, GPIO.LOW)

    GPIO.output(led2Pin, GPIO.HIGH)

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

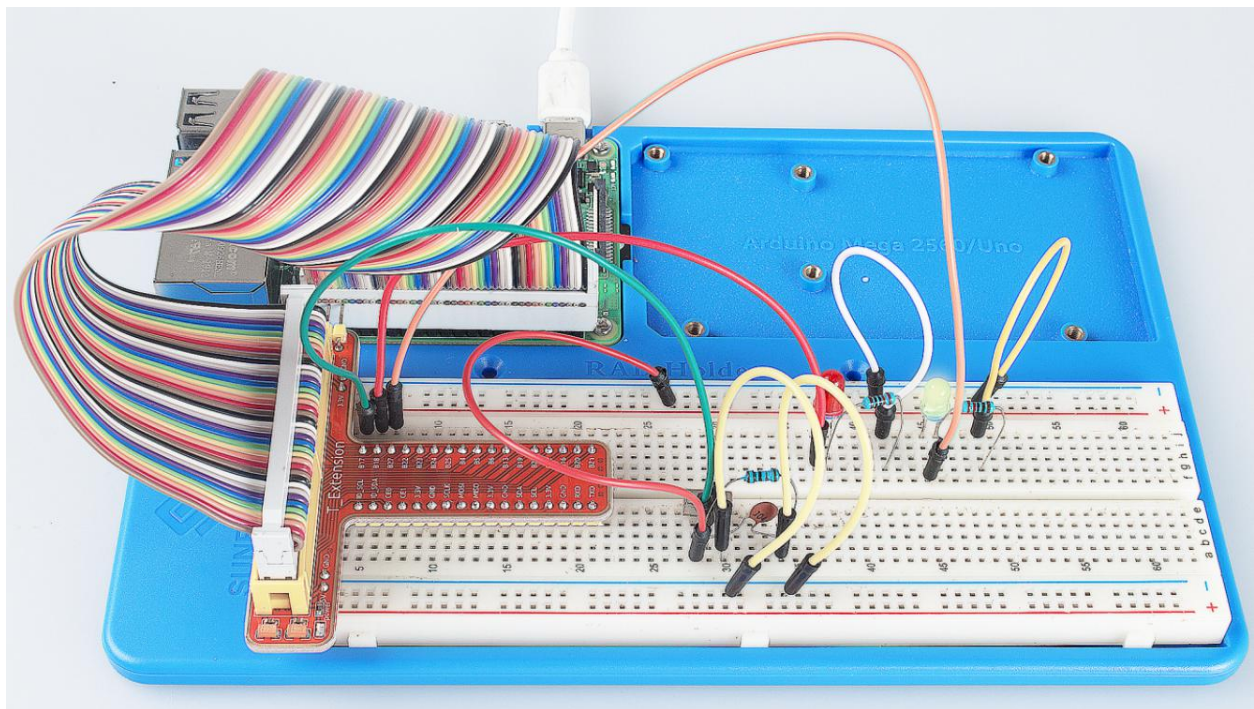
if GPIO.input(slidePin) == 0:

    print (" LED2 ON ")

    GPIO.output(led2Pin, GPIO.LOW)

    GPIO.output(led1Pin, GPIO.HIGH)
```

Now pull the slide, and you can see the two LEDs light up alternately.



6.11 Lesson 11 How to Drive a DC Motor

6.11.1 Introduction

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

6.11.2 Components

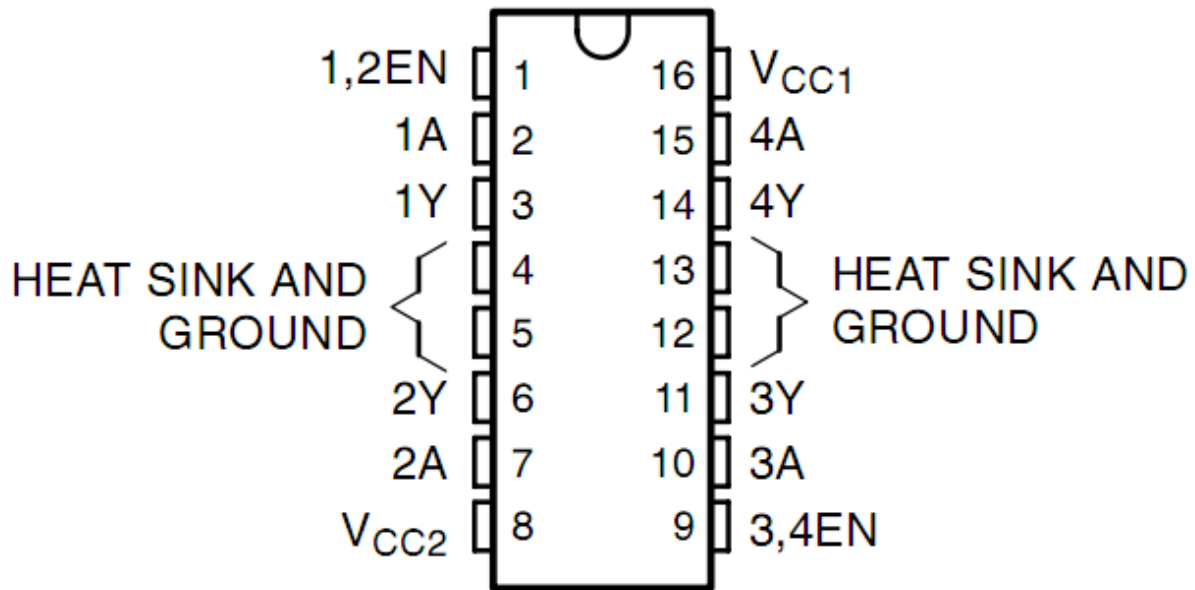
- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * L293D
- 1 * DC motor
- 1 * Power Module
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.11.3 Principle

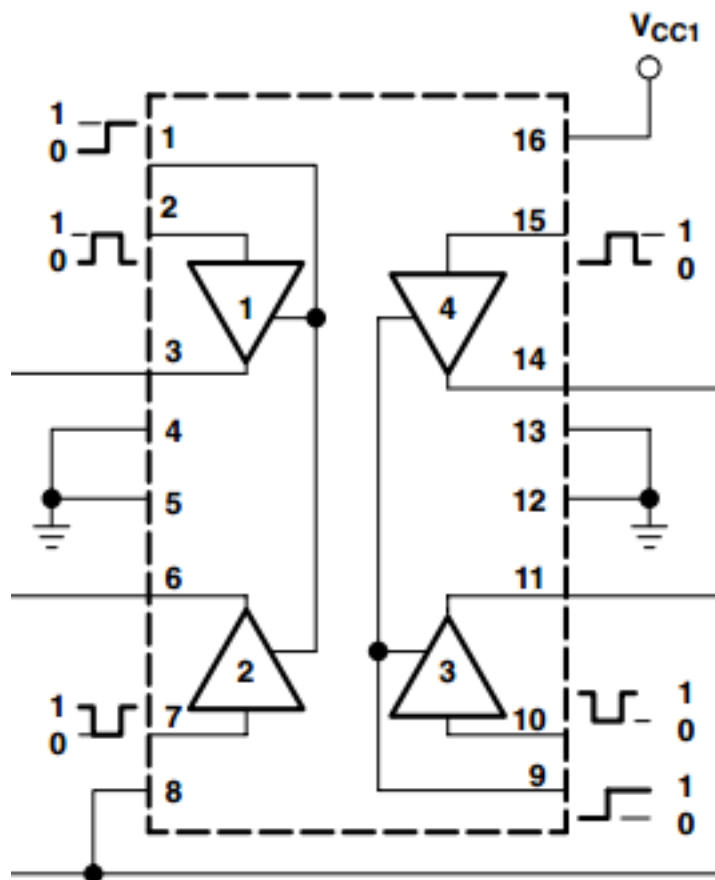
L293D

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, stepping motors) and power switching transistors etc. DC Motors are devices that turn DC electrical energy into mechanical energy. They are widely used in electrical drive for their superior speed regulation performance.

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



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



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

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

In this experiment, it just needs to drive one motor, so here only half of the L293D will be used.

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): 70m

A Free-run speed (3V): 13000RPM

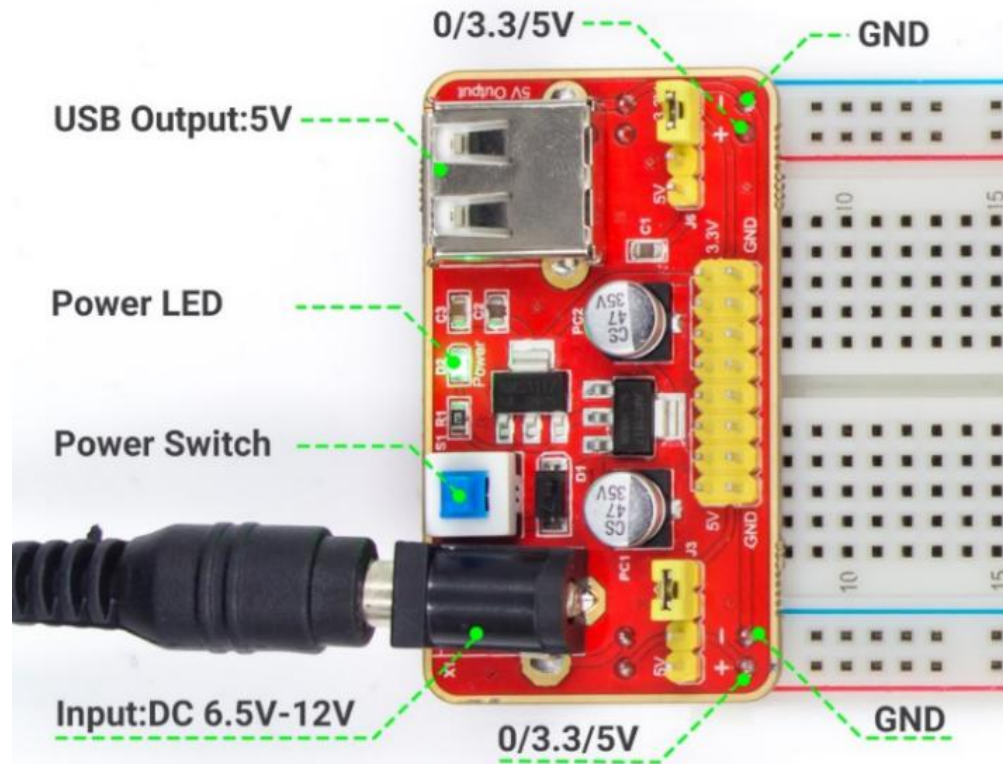
Stall current (3V): 800mA

Shaft diameter: 2mm

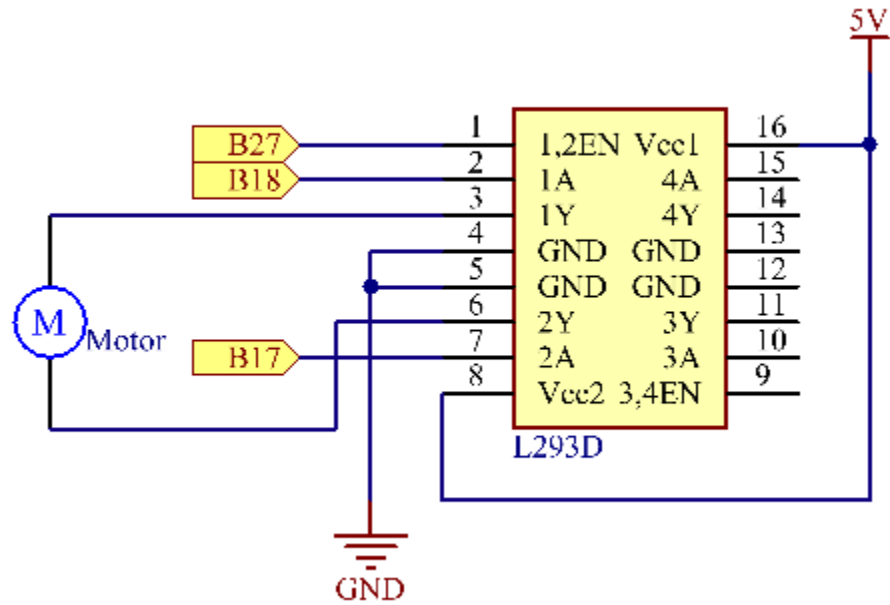
Power Supply Module

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:**

Principle: 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 B27, and set it as high level. Connect pin2 to B18, and pin7 to B27, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

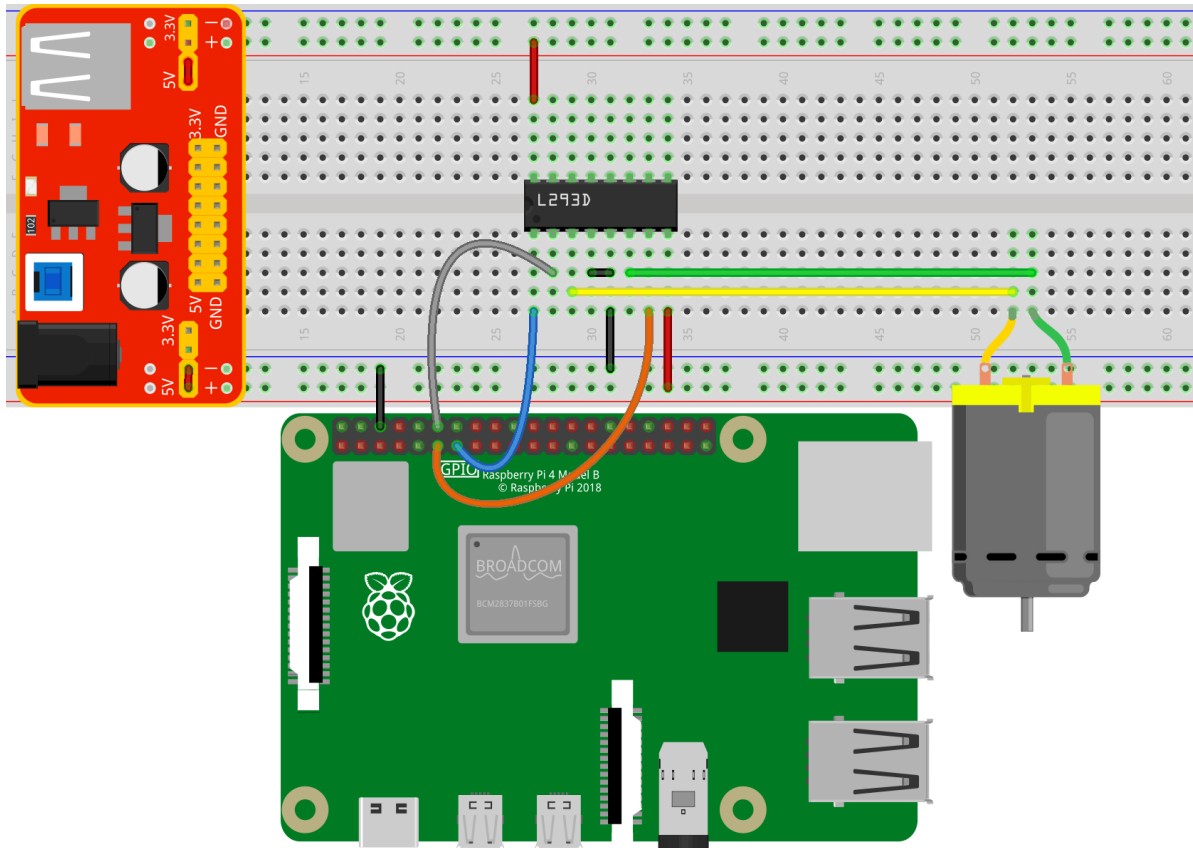


Experimental Procedures

Step 1: Build the circuit. Since the power supply module and T-cable are incompatible, we will not use the T-Cable in this experiment.

Note: The power module can apply a 9V battery with the 9V Battery Buckle in the kit. Insert the jumper cap of the power module into the 5V bus strips of the breadboard.





For C Language Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 11_motor
```

Step 4: Run the executable file above.

```
sudo ./11_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
```

(continues on next page)

(continued from previous page)

```

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

    pinMode(MotorPin1, OUTPUT);
    pinMode(MotorPin2, OUTPUT);
    pinMode(MotorEnable, OUTPUT);

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|                Motor                |\n");
    printf("|-----|\n");
    printf("|    Motor pin 1 connect to GPIO0    |\n");
    printf("|    Motor pin 2 connect to GPIO1    |\n");
    printf("|    Motor enable connect to GPIO3    |\n");
    printf("|                                     |\n");
    printf("|          Controlling a motor          |\n");
    printf("|                                     |\n");
    printf("|                               SunFounder |\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

return 0;
}

```

Code Explanation

```

digitalWrite(MotorEnable, HIGH) // Enable the L239D

digitalWrite(MotorPin1, HIGH);
// Set a high level for 2A(pin 7); since 1,2EN(pin 1) is in high level, 2Y will
↳ output high level
digitalWrite(MotorPin2, LOW) /* Set a low level for 1A, then 1Y
will output low level, and the motor will rotate.*/

for(i=0;i<3;i++)
{

    delay(1000);

} // this loop is to delay for 3*1000ms

digitalWrite(MotorEnable, LOW)
// If 1,2EN (pin1) is in low level, L293D does not work. Motor stops rotating.

digitalWrite(MotorPin1, LOW)

digitalWrite(MotorPin2, HIGH)
// Reverse the current flow of the motor, then the motor will rotate reversely.

```

For Python Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 11_motor.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set up pins
MotorPin1 = 17
MotorPin2 = 18
MotorEnable = 27

def print_message():
    print ("=====")
    print ("|                               Motor                               |")

```

(continues on next page)

(continued from previous page)

```

print ("| ----- |")
print ("|      Motor pin 1 connect to GPIO17      |")
print ("|      Motor pin 2 connect to GPIO18      |")
print ("|      Motor enable connect to GPIO27      |")
print ("| ----- |")
print ("|      Controlling a motor                  |")
print ("| ----- |")
print ("|                                     SunFounder |")
print ("=====\\n")
print ("Program is running...")
print ("Please press Ctrl+C to end the program...")
#raw_input ("Press Enter to begin\\n")

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

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

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

```

(continues on next page)

(continued from previous page)

```

        motor(directions['STOP'])
        time.sleep(5)
        # Anticlockwise
        motor(directions['CCW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)

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

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

```

Code Explanation

```

GPIO.setup(MotorPin1, GPIO.OUT)
# Set pin1 and pin2 for motor's rotation direction as output pin

GPIO.setup(MotorPin2, GPIO.OUT)

GPIO.setup(MotorEnable, GPIO.OUT)
# Set pins for motor's working condition as output pin

GPIO.output(MotorEnable, GPIO.LOW)
# Set the motor low level for initial state

GPIO.output(MotorEnable, GPIO.HIGH) # Set the motor in high level

GPIO.output(MotorPin1, GPIO.HIGH)
# Set pin1 in high level and pin2 in low level

GPIO.output(MotorPin2, GPIO.LOW) # Make the motor rotate clockwise

time.sleep(5) # rotate for 5 seconds

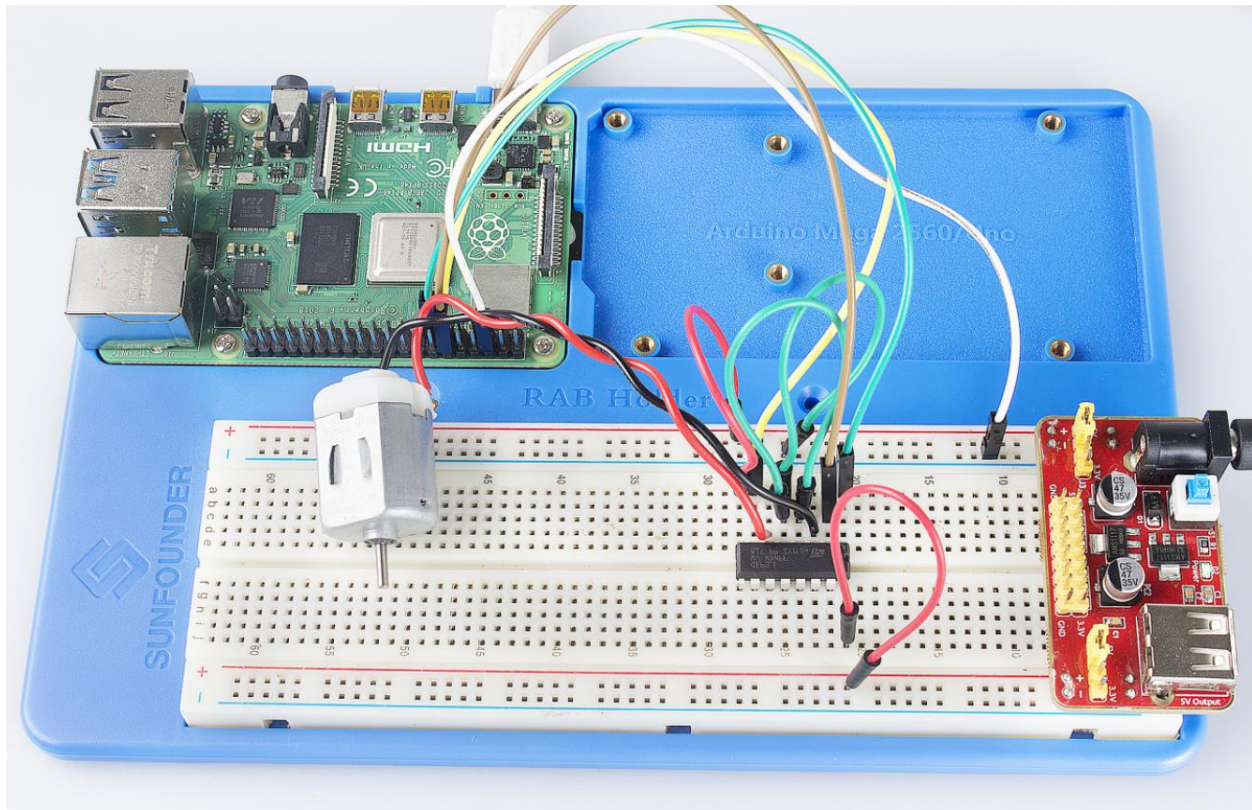
GPIO.output(MotorEnable, GPIO.LOW) # Stop the motor

time.sleep(5) #wait for 5 seconds

#Code for motor counter-clockwise rotation is similar to sketch above

```

Now, you should see the motor blade rotating.



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.

6.12 Lesson 12 Rotary Encoder

6.12.1 Introduction

A rotary encoder is an electro-mechanical device that converts the angular position or motion of a shaft or axle to analog or digital code. Rotary encoders are usually placed at the side which is perpendicular to the shaft. They act as sensors for detecting angle, speed, length, position, and acceleration in automation field.

6.12.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 4 * Jumper wires (Male to Male, 2 red and 2 black)
- 1 * Network cable (or USB wireless network adapter)
- 1 * Rotary Encoder module
- 1 * 5-Pin anti-reverse cable
- 1 * T-Extension Board

- 1 * 40-Pin GPIO Cable

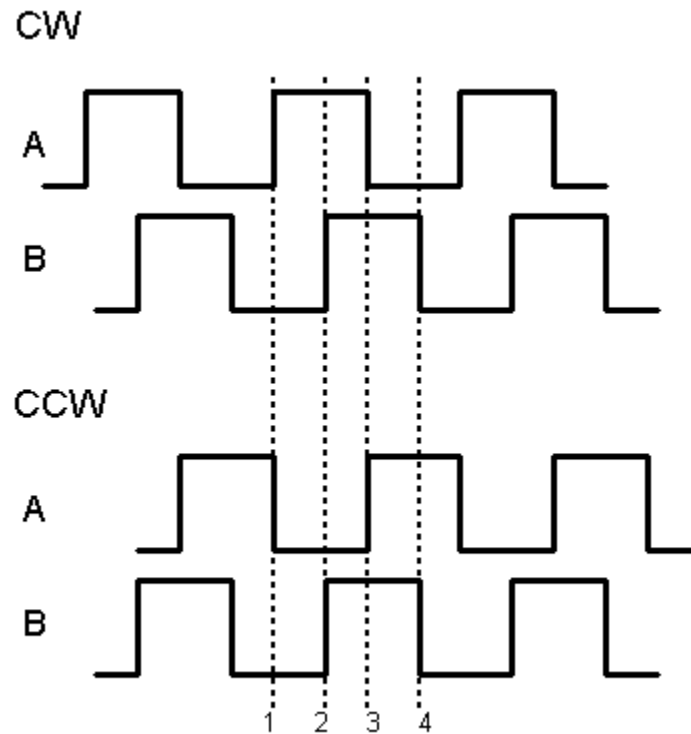
6.12.3 Experimental Principle



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, and so on.

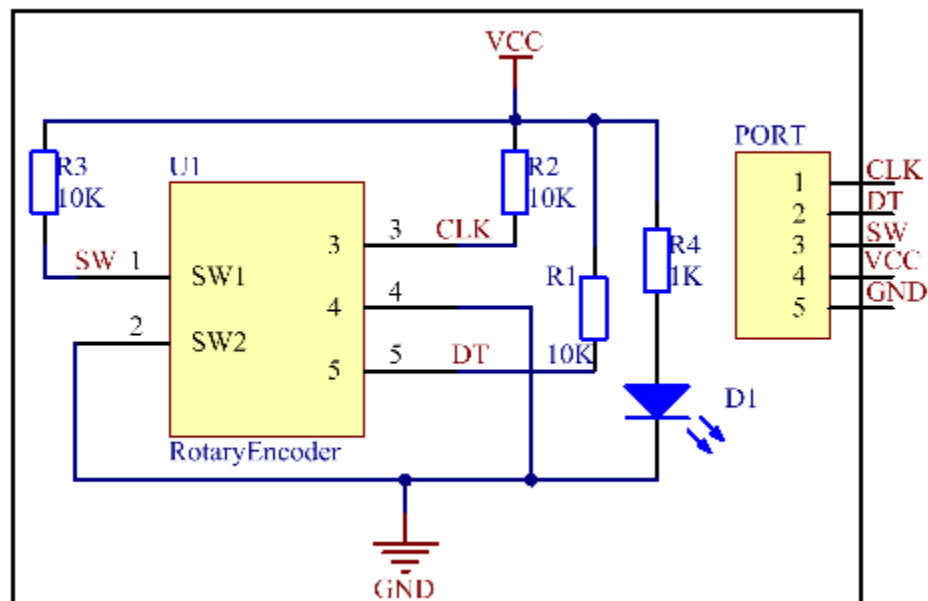
There are mainly two types of rotary encoders: absolute and incremental (relative) encoders. Here we use an incremental (relative) encoders.

Most rotary encoders have 5 pins with three functions of turning left & right and pressing down. Pin 1 and pin 2 are switch wiring terminals used to press. Pin 4 is generally connected to ground. Pin 3 and pin 5 are first connected to pull-up resistor and connect to VCC. Pin 3 and pin 5 generate two-phase square waves whose phase difference is 90°. Usually the two-phase square waves are called channel A and channel B as shown below:



We can see from the figure above: If channel A is in low level, and channel B converts from high level to low, it indicates the Rotary Encoder has spun clockwise (CW). If channel A is in low level, and channel B converts from low level to high, it indicates the Rotary Encoder has spun counter-clockwise (CCW). Thus when channel A is in low level, we can know the direction that Rotary Encoder spun by channel B.

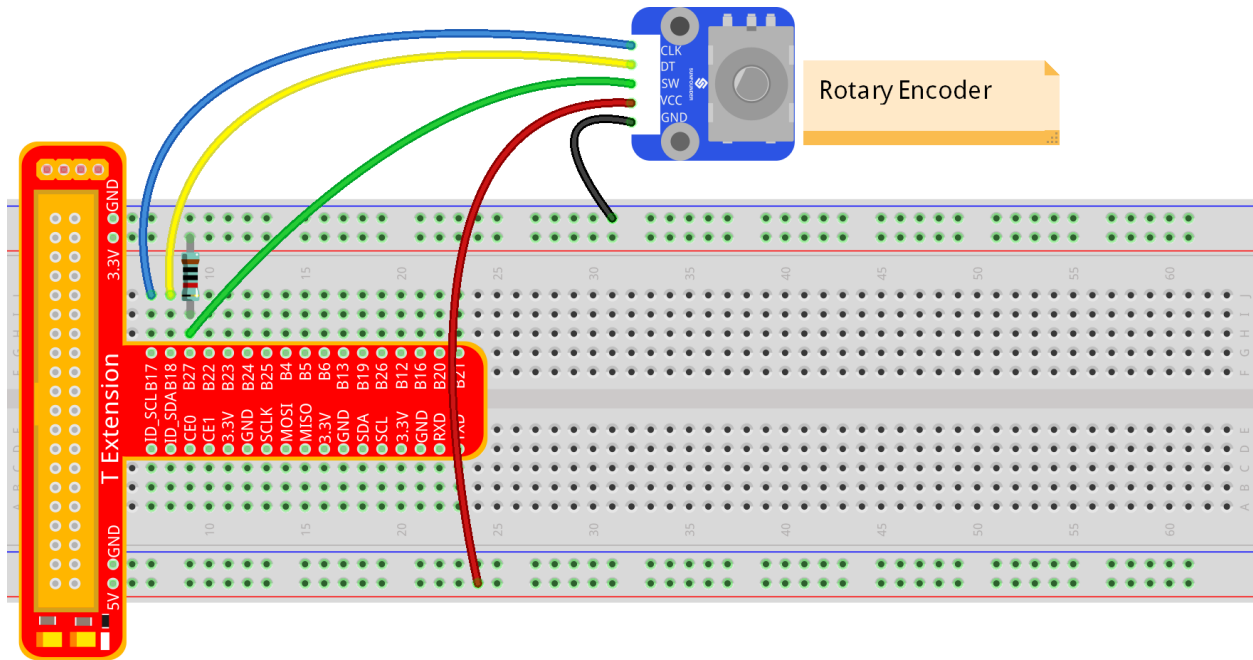
The schematic diagram of the Rotary Encoder is shown as below. We can see that pin 3 on the Rotary Encoder is CLK of the module, while pin 5 is DT. Then we can know the Rotary's rotating direction by the value of CLK and DT.



It is summarized by using oscilloscope to observe the output waveform of CLK and DT and operating the rotary encoder. You can try yourself.

6.12.4 Experimental Procedures

Step 1: Build the circuit.



fritzing

For C Language Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 12_rotaryEncoder
```

Step 4: Run the executable file above.

```
sudo ./12_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>
```

(continues on next page)

(continued from previous page)

```

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

#define RoAPin 0
#define RoBPin 1
#define SWPin 2

static volatile int globalCounter = 0 ;

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

void btnISR(void){
    globalCounter = 0;
}

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 ++;
        }
        if((Last_RoB_Status == 1)&&(Current_RoB_Status == 0)){
            globalCounter --;
        }
    }
}

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

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

    pullUpDnControl(SWPin, PUD_UP);

    if(wiringPiISR(SWPin, INT_EDGE_FALLING, &btnISR) < 0){
        printf("Unable to init ISR:%s\n",strerror(errno));
        return 1;
    }

    printf("\n");
    printf("\n");
    printf("=====\n");

```

(continues on next page)

(continued from previous page)

```

printf("|           Rotary Encoder           |\n");
printf("| ----- |\n");
printf("|           Pin A connect to GPIO0      |\n");
printf("|           Pin B connect to GPIO1      |\n");
printf("|           Button Pin connect to GPIO 2 |\n");
printf("| ----- |\n");
printf("|           Use a Rotary Encoder        |\n");
printf("|           Rotary to add/minus counter  |\n");
printf("|           Press to set counter to 0    |\n");
printf("| ----- |\n");
printf("|                                     SunFounder |\n");
printf("=====|\n");
printf("\n");
printf("\n");

int tmp = 0;
while(1){
    rotaryDeal();
    if (tmp != globalCounter){
        printf("Counter : %d\n",globalCounter);
        tmp = globalCounter;
    }
}
return 0;
}

```

Code Explanation

```

#define RoAPin 0 // CLK connects to B17, define B17 as 0 in wiring Pi.

#define RoBPin 1 // DT connects to GPIO1, define B18 as 1 in wiring Pi.

#define SWPin 2 // SW connects to GPIO2

void rotaryDeal(void)
/* Pi detects the pulse when spinning the rotary
encoder, and judge the spinning direction, then increase or decrease the
value of globalCounter to record the angular displacement. */
{
    Last_RoB_Status = digitalRead(RoBPin); // Read the value of DT

    while(!digitalRead(RoAPin)) // If CLK is low, run the program below.
    {
        Current_RoB_Status = digitalRead(RoBPin);
        // Read the value of DT, and store it in Current_RoB_Status.
        flag = 1;
    }

    if(flag == 1) // If CLK outputs low level, then flag=1
    {
        flag = 0;
        if((Last_RoB_Status == 0)&&(Current_RoB_Status == 1))
            // If DT value converts from low to high, the globalCounter adds 1.
            {
                globalCounter ++;
            }
        if((Last_RoB_Status == 1)&&(Current_RoB_Status == 0))

```

(continues on next page)

(continued from previous page)

```

        //If DT value converts from high to low
        {
            globalCounter --; // the globalCounter decreases 1.
        }
    }
}

printf("globalCounter : %d\n",globalCounter); // Print the value of globalCounter.

void btnISR(void): // If the rotary encoder is pressed down, reset the value.

```

For Python Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 12_rotaryEncoder.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

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

def print_message():
    print ("=====")
    print ("|          Rotary Encoder          |")
    print ("|-----|")
    print ("|      Pin A connect to GPIO17      |")
    print ("|      Pin B connect to GPIO18      |")
    print ("|      Button Pin connect to GPIO27  |")
    print ("|-----|")
    print ("|      Use a Rotary Encoder         |")
    print ("|      Rotary to add/minus counter   |")
    print ("|      Press to set counter to 0     |")
    print ("|-----|")
    print ("|                               SunFounder |")
    print ("=====\\n")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\\n")

```

(continues on next page)

(continued from previous page)

```

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
    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

def main():
    print_message()
    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.

```

(continues on next page)

(continued from previous page)

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

Code Explanation

```
globalCounter = 0 # Set a global variable to count

flag = 0 # Set a flag for reverse spinning.

Last_RoB_Status = 0 # Set a variable to store the previous state of pinB

Current_RoB_Status = 0 # Set a variable to store the present state of pinB

# 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) # Store channel B state

    # When RoAPin level changes

    while(not GPIO.input(RoAPin)): # When channel A is not in low, exit the while loop

        Current_RoB_Status = GPIO.input(RoBPin)

        flag = 1

    if flag == 1: # If flag value is 1, the rotary encoder is CW rotating

        # 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):

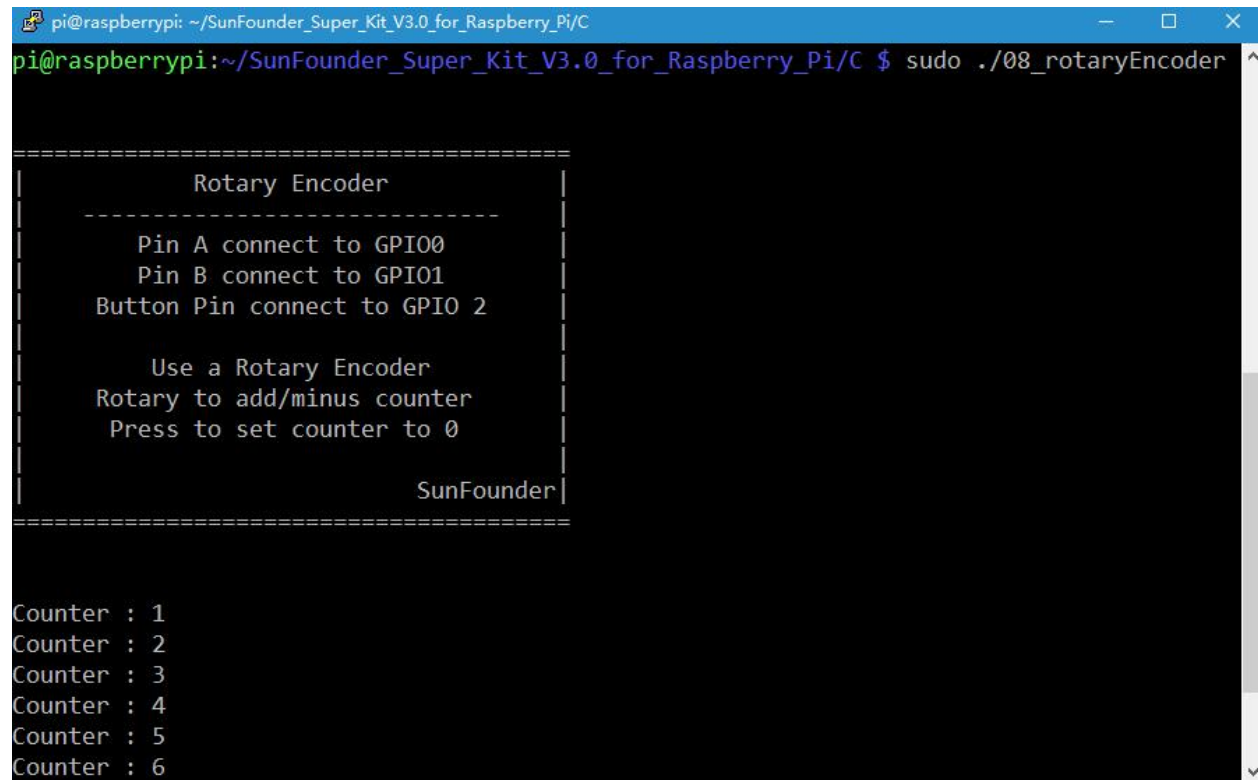
    global counter
```

(continues on next page)

(continued from previous page)

```
counter = 0
```

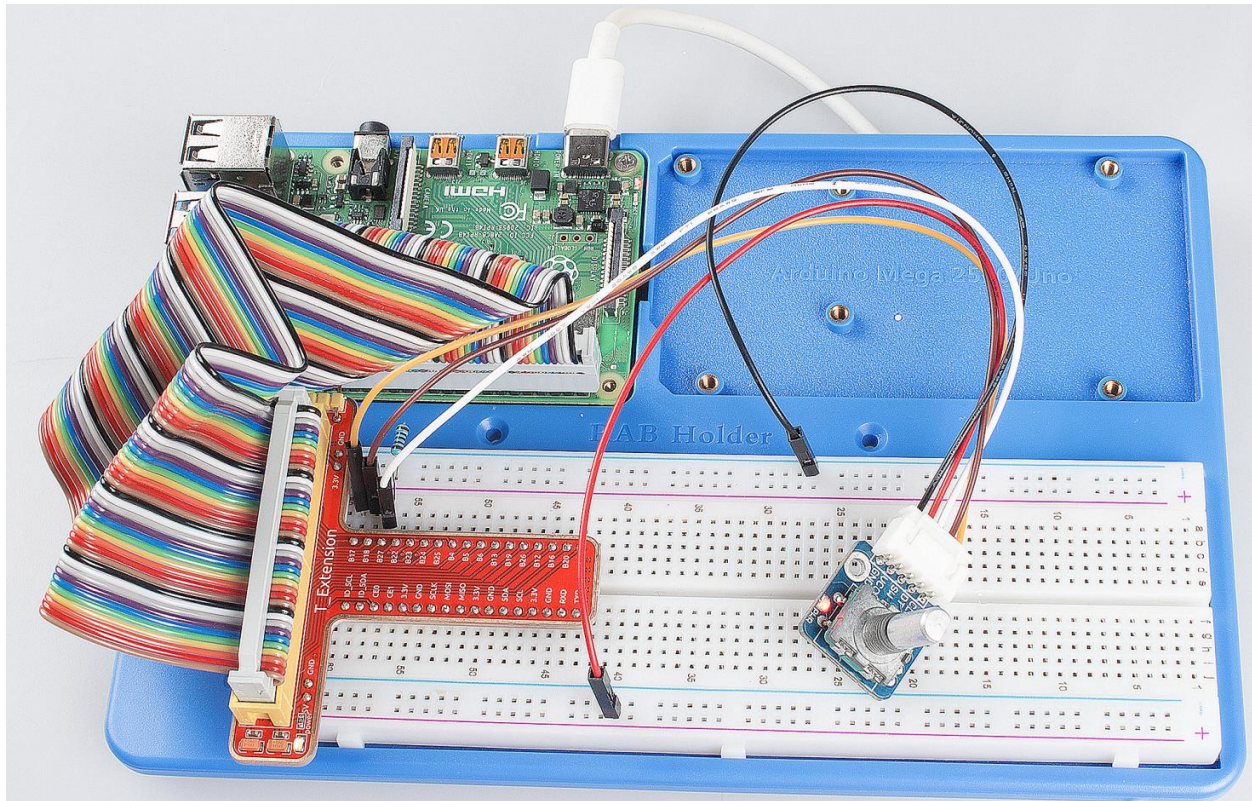
Now, gently rotate the encoder to change the value of the variable in the above program and you will see the value printed on the screen. Rotate the encoder clockwise, the value will increase; or rotate it counterclockwise, the value will decrease.



```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./08_rotaryEncoder

=====
|               Rotary Encoder               |
|-----|
|   Pin A connect to GPIO0   |
|   Pin B connect to GPIO1   |
| Button Pin connect to GPIO 2 |
|                               |
|   Use a Rotary Encoder     |
| Rotary to add/minus counter |
| Press to set counter to 0   |
|                               |
|                               SunFounder |
|-----|
=====

Counter : 1
Counter : 2
Counter : 3
Counter : 4
Counter : 5
Counter : 6
```



6.13 Lesson 13 Driving LEDs by 74HC595

6.13.1 Introduction

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

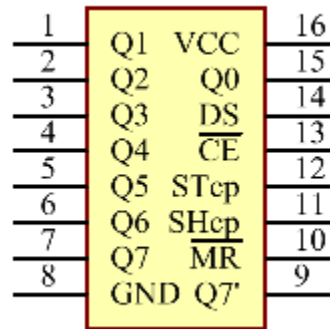
6.13.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * 74HC595
- 8 * LED
- 8 * Resistor (220)
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.13.3 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 to keep the chip from resetting.

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.

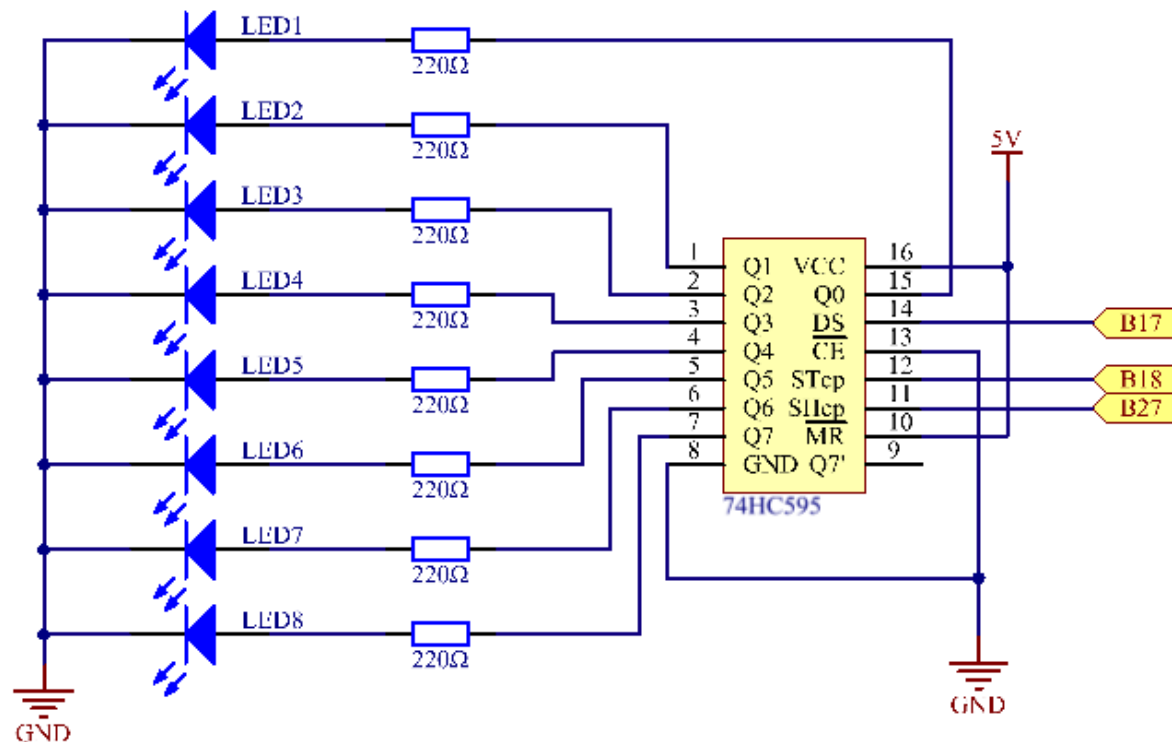
OE: Output enable pin, active at low level; here connected to GND to keep 74HC595 in output enable state.

DS: Serial data input pin

VCC: Positive supply voltage

GND: Ground

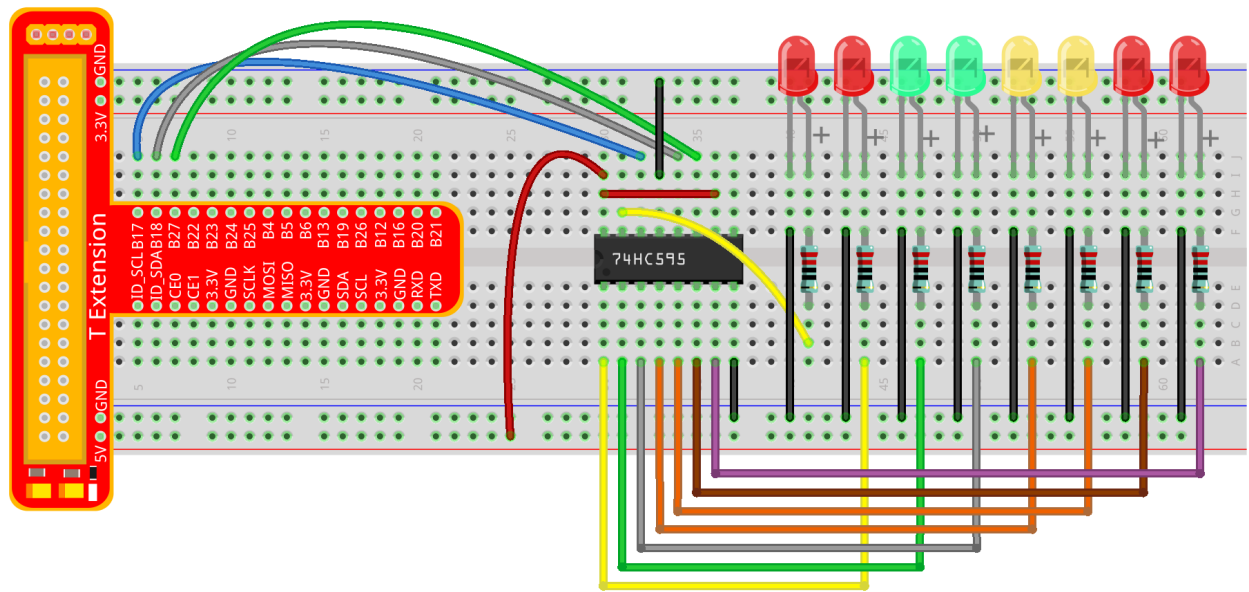
The schematic diagram is shown as below:



Principle: In this experiment, connect 74HC595's ST_CP to Raspberry Pi's B18, SH_CP to B27, and DS to B17; connect a current-limit resistor and then a LED to Q0-Q7 respectively; connect MR and VCC to 5V, CE and GND to GND. 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, and output to Q0-Q7. 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.

Experimental Procedures

Step 1: Build the circuit. If you want to take out the chip from the breadboard, DO NOT pull it in one direction forcefully, for fear that the pins on it may be bent and you may get hurt. Try to use a sharp tool to cross the notch of the breadboard to remove the chip.



fritzing

For C Language Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 13_74HC595_LED
```

Step 4: Run the executable file above.

```
sudo ./13_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);
```

(continues on next page)

(continued from previous page)

```

    digitalWrite(pin, 1);
}

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 message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    init();

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          LEDs with 74HC595          |\n");
    printf("| ----- |\n");
    printf("|          SDI connect to GPIO0          |\n");
    printf("|          RCLK connect to GPIO1          |\n");
    printf("|          SRCLK connect to GPIO 2          |\n");
    printf("|          |\n");
    printf("|          Control LEDs with 74HC595          |\n");
    printf("|          |\n");
    printf("|          SunFounder          |\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    while(1) {
        for(i=0; i<8; i++) {
            SIPO(LED[i]);
            pulse(RCLK);
            delay(150);
            //printf("i = %d\n", i);
        }
        delay(500);
    }
}

```

(continues on next page)

(continued from previous page)

```

    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);
        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;
}

```

Code Explanation

```

unsigned char LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
/* This array is to store the output values of Q0-Q7. For example, 0x01 in
binary format is 0000 0001, thus Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 are 0 0 0 0 0 0
0 1 respectively, that is Q0=1, and the LED connected to Q0 will light
up. Thus we can light up the eight LEDs separately in this way. */

void pulse(int pin){ // generate a rising edge

    digitalWrite(pin, 0);

    digitalWrite(pin, 1);

}

void SIPO(unsigned char byte){
    // Assign the char byte to the SDI bit by bit

    int i;

    for(i=0;i<8;i++){

        digitalWrite(SDI, ((byte & (0x80 >> i)) > 0)); /* Use the for loop to

```

(continues on next page)

(continued from previous page)

```

        count 8 times in cycle, and write a 1-bit data to the SDI each time. The
        data is a result of the AND operation. (0x80 >> i) is to implement the
        operation from left to right by bit, so each time one of the eight bits
        in byte (0000 0001). */

        pulse(SRCLK); /* the shift register generates a rising edge pulse, and
        data in DS will shift to the shift register. */

    } /* This part is to assign the data in byte to SDI(DS) by bits, thus
    when the shift register generates a rising edge pulse, data in SDI(DS)
    will transfer to it by bits. */

}

void init(void){ // Set DS, ST_CP, SH_CP as output, and low level as the initial state

    for(i=0;i<8;i++){

        SIPO(LED[i]); /* Assign the value in the LED[i] array to SDI(DS). When
        i=1, LED[0]=0x01 shifts to the shift register. */

        pulse(RCLK); /* RCLK (ST_CP) generates a rising edge pulse, and the data
        of the shift register is stored in the RCLK (ST_CP) storage register,
        and output at Q0-Q7. */

        delay(150);

    } /* After 8 cycles, Q0-Q7 will output 0x01 to 0x10 in sequence, that is to
    light up the LEDs connected to Q0-Q7 in turn. */

}

```

Sketch in later part not explained here is to light up 8 LEDs together, and dim them; then light up LEDs connected to Q7-Q0 one by one, and all 8 LEDs light up, dim in the end. Thus, a cycle completes. You can observe the LEDs' state.

For Python Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 13_74HC595_LED.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

if version_info.major == 3:
    raw_input = input

SDI    = 17

```

(continues on next page)

(continued from previous page)

```

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
BLINK = [0xff,0x00,0xff,0x00,0xff,0x00] #blink
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_message():
    print ("=====")
    print ("|          LEDs with 74HC595          |")
    print ("| ----- |")
    print ("|          SDI connect to GPIO17        |")
    print ("|          RCLK connect to GPIO18       |")
    print ("|          SRCLK connect to GPIO27      |")
    print ("|          |")
    print ("|          Control LEDs with 74HC595    |")
    print ("|          |")
    print ("|          SunFounder |")
    print ("=====")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program..")
    #raw_input ("Press Enter to begin\n")

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

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

def main():
    print_message()
    mode = LED0 # Change Mode, modes from LED0 to LED3
    sleeptime = 0.15 # Change speed, lower value, faster speed
    blink_sleeptime = 0.3
    leds = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ']
    while True:
        # Change LED status from mode

```

(continues on next page)

(continued from previous page)

```

print (" mode ")
for onoff in mode:
    hc595_shift(onoff)
    leds[mode.index(onoff)] = 1    # Show which led is on
    print (leds)
    time.sleep(sleeptime)
    leds[mode.index(onoff)] = '-'  # Show the led is off

print (" blink ")
for onoff in BLINK:
    hc595_shift(onoff)
    if (onoff == 0x00):
        leds = ['-'] * 8
    elif (onoff == 0xff):
        leds = [1] * 8
    print (leds)
    time.sleep(blink_sleeptime)

# Change LED status from mode reverse
print (" reversed mode ")
for onoff in reversed(mode):
    hc595_shift(onoff)
    leds[mode.index(onoff)] = 1    # Show which led is on
    print (leds)
    time.sleep(sleeptime)
    leds[mode.index(onoff)] = '-'  # Show the led is off

print (" blink ")
for onoff in BLINK:
    hc595_shift(onoff)
    if (onoff == 0x00):
        leds = ['-'] * 8
    elif (onoff == 0xff):
        leds = [1] * 8
    print (leds)
    time.sleep(blink_sleeptime)

def destroy():
    GPIO.cleanup()

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

```

Code Explanation

LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80] ''' Define some LED blinking modes. Convert hexadecimal value to binary value will be more intuitionistic. For instance, 0x01 is binary 00000001, meaning the last LED lighting up; 0x80 is binary 10000000, representing the first LED lighting up. '''

LED1 = [0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff] # blink mode 1

(continues on next page)

(continued from previous page)

```

LED2 = [0x01,0x05,0x15,0x55,0xb5,0xf5,0xfb,0xff] # blink mode 2
LED3 = [0x02,0x03,0x0b,0x0f,0x2f,0x3f,0xbf,0xff] # blink mode 3

# Shift the data to 74HC595

def hc595_shift(dat): # Shift the data to 74HC595
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit)) # Assign dat data to SDI pins of HC595_
        ↪by bits

        GPIO.output(SRCLK, GPIO.HIGH) # Every SRCLK adds one, the shift register_
        ↪moves one bit.

        time.sleep(0.001)

        GPIO.output(SRCLK, GPIO.LOW)

        GPIO.output(RCLK, GPIO.HIGH) # Everytime RCLK adds one, the HC595 updates output.

        time.sleep(0.001)

        GPIO.output(RCLK, GPIO.LOW)

leds = ['-','-','-','-','-','-','-','-']
# the array storing the LED state, used for command line printing.

while True:
    # Change LED status from mode

    print (" mode")

    for onoff in mode: # Assign value to variable onoff by mode[] list

        hc595_shift(onoff)

        leds[mode.index(onoff)] = 1 # Show which led is on

        print (leds)

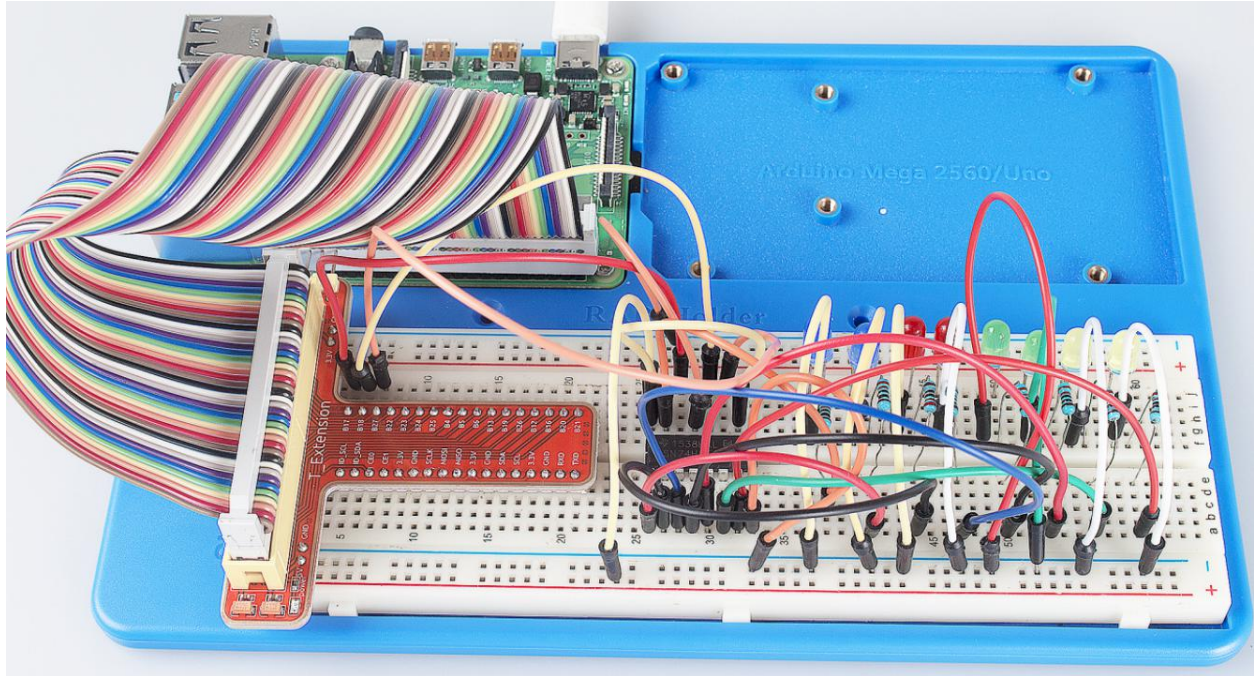
        time.sleep(sleeptime)

        leds[mode.index(onoff)] = '-' # Show the led is off

        # for loops in later part work similarly, lighting up LED by list.

```

Input a 2-bit hexadecimal parameter dat via hc595_in(dat) to control 8 LEDs state, and hc595_out() will output state to 8 LEDs. In While True, the for loop will shift the LED blinking list to the hc595_in(dat) function, thus we can see the LED light flowing. Here you should see eight LEDs light up one by one, and then all light up and dim after a while; then eight LEDs will light up from reverse direction one by one, and then all light up and then dim after a while. This cycle will keep running.



6.14 Lesson 14 Driving 7-Segment Display by 74HC595

6.14.1 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 and A to F.

6.14.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * 74HC595
- 1 * 7-segment display
- 2 * Resistor (220, 10k)
- 1 * Button
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

6.14.3 Principle

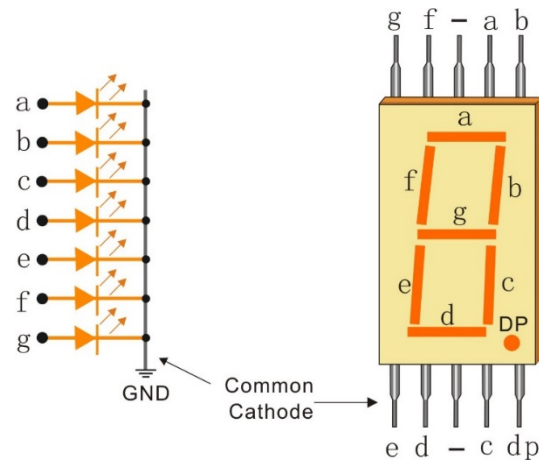
7-Segment Display

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.

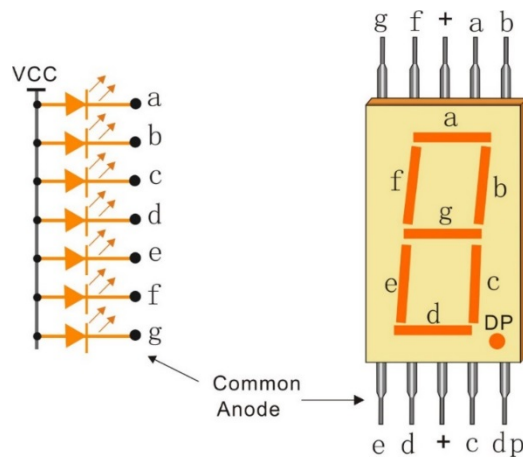


Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from “a” through to “g” representing each individual LED. The other LED pins are connected together forming a common pin. So by forward 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.



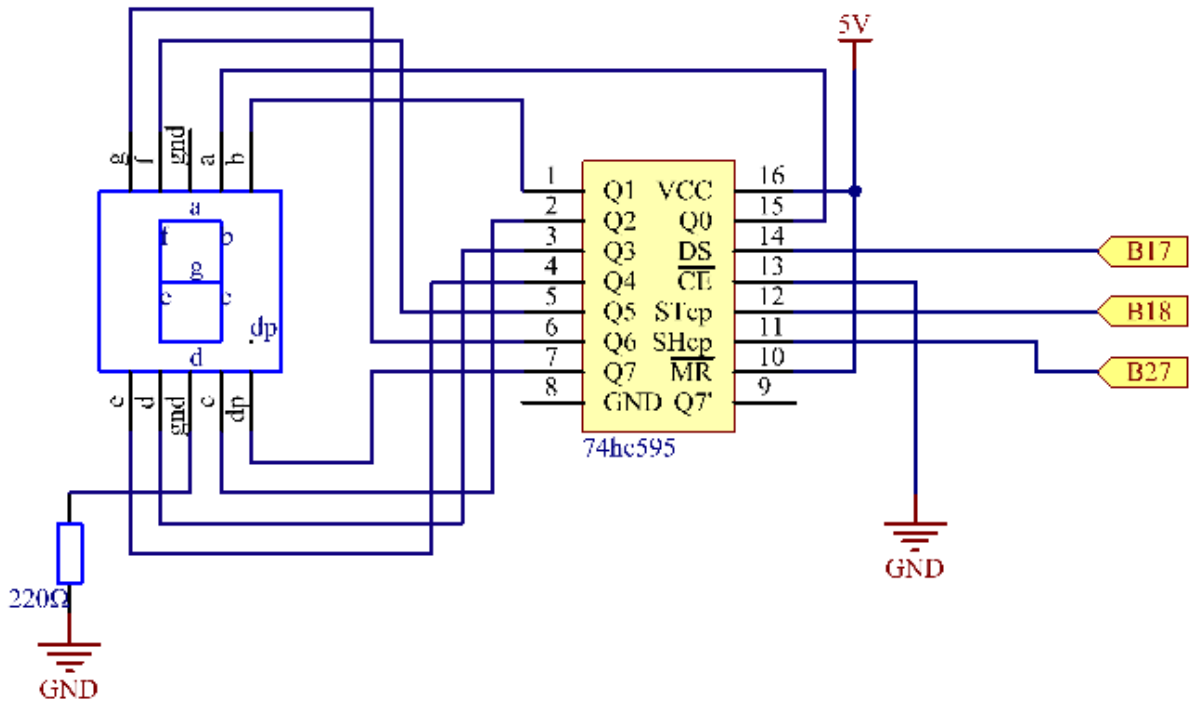
In a common cathode display, the cathodes of all the LED segments are connected to the logic “0” or ground. Then an individual segment (a-g) is energized by a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the anode of the segment.



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.

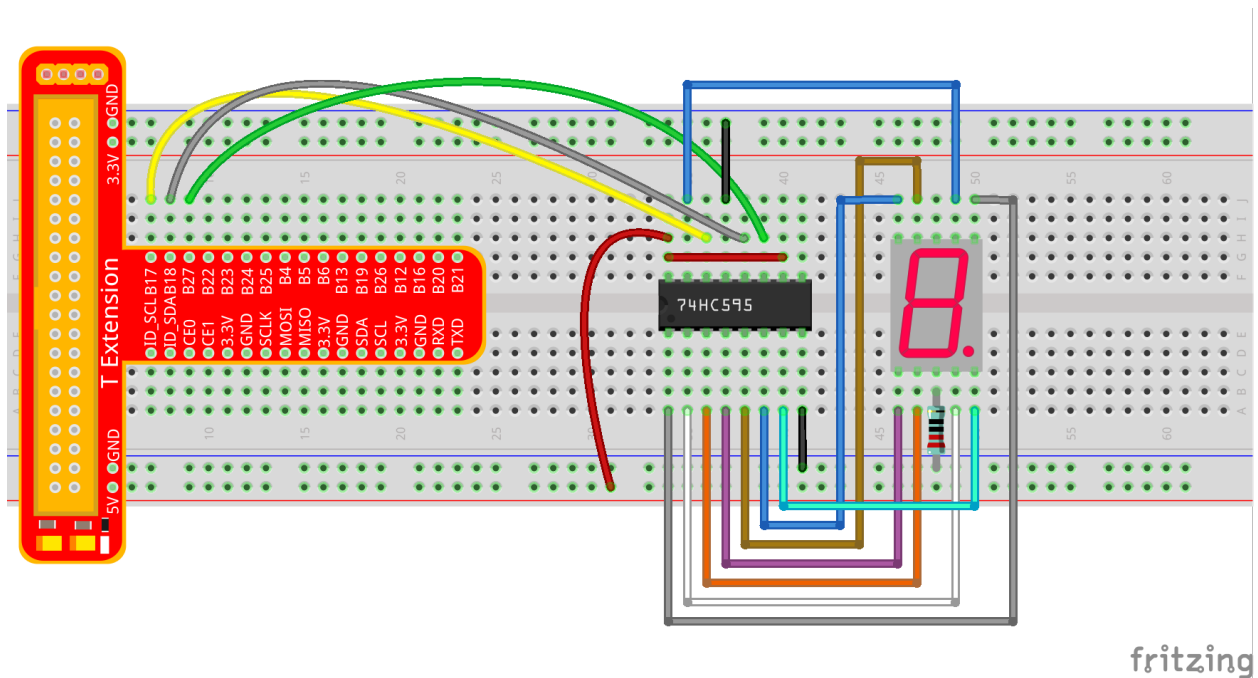
The schematic diagram is shown as below:



Principle: Connect pin ST_CP of 74HC595 to Raspberry Pi B18, SH_CP to B27, DS to B17, 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.

6.14.4 Experimental Procedures

Step 1: Build the circuit.



For C Language Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 14_segment
```

Step 4: Run the executable file above.

```
sudo ./14_segment
```

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 message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    init();

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          Segment with 74HC595          |\n");
    printf("|-----|\n");
    printf("|          SDI connect to GPIO0          |\n");
    printf("|          RCLK connect to GPIO1         |\n");
    printf("|          SRCLK connect to GPIO 2       |\n");
    printf("|          |\n");
    printf("|          Control segment with 74HC595   |\n");
    printf("|          |\n");
    printf("|          SunFounder                     |\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    while(1){
        for(i=0;i<17;i++){
            printf("Print %1X on Segment\n", i);
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

return 0;
}

```

Code Explanation

```

unsigned char SegCode[17] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80}
↪;
// display array from 0 to F.

void init(void){}
// Initialize the function, set ds, st_cp, sh_cp three pins to low level, and the_
↪initial state as 0.

void hc595_shift(unsigned char dat){

    int i;

    for(i=0;i<8;i++){

        digitalWrite(SDI, 0x80 & (dat << i)); /* Assign the dat data to SDI(DS)
        by bits. Here we assume dat=0x3f(0011 1111, when i=0, 0x3f will shift
        right(<<) 0 bits, 0x3f & 0x80 = 1000 0000 */

        digitalWrite(SRCLK, 1); /* SH_CP will convert from low to high, and
        generate a rising edge pulse, then shift the DS date to shift register. */

        delay(1);

        digitalWrite(SRCLK, 0);

    } // to assign 8 bit value to 74HC595's shift register

    digitalWrite(RCLK, 1); /* ST_CP converts from low to high and generate a
    rising edge, then shift data from shift register to storage register. */

    delay(1);

    digitalWrite(RCLK, 0);

} // Transfer data in shift register to data register to update the displayed data.

```

For Python Users:

Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 14_segment.py
```

Code

```

import RPi.GPIO as GPIO
import time
from sys import version_info

```

(continues on next page)

(continued from previous page)

```

if version_info.major == 3:
    raw_input = input

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

# Define a segment code from 0 to F in Hexadecimal
# Common cathode
segCode = [0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79,
↪ 0x71]
# Common anode
# segCode = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1,
↪ 0x86, 0x8e]

def print_msg():
    print ("=====")
    print ("|          Segment with 74HC595          |")
    print ("|-----|")
    print ("|          SDI connect to GPIO17          |")
    print ("|          RCLK connect to GPIO18          |")
    print ("|          SRCLK connect to GPIO27         |")
    print ("|-----|")
    print ("|          Control segment with 74HC595     |")
    print ("|-----|")
    print ("|          SunFounder|")
    print ("=====")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program..")
    #raw_input ("Press Enter to begin\n")

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

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

def main():
    print_msg()
    while True:
        # Shift the code one by one from segCode list
        for code in segCode:
            hc595_shift(code)
            print ("segCode[%s]: 0x%02X"%(segCode.index(code), code)) # double digit_
↪to print

```

(continues on next page)

(continued from previous page)

```

        time.sleep(0.5)

def destroy():
    GPIO.cleanup()

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

```

Code Explanation

```

# Define a segment code from 0 to F in Hexadecimal

# Common cathode

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

# Common anode

# segCode = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88,
            0x83, 0xc6, 0xa1, 0x86, 0x8e]

# Shift the data to 74HC595

def hc595_shift(dat):

    for bit in range(0, 8):

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

        GPIO.output(SRCLK, GPIO.HIGH)

        time.sleep(0.001)

        GPIO.output(SRCLK, GPIO.LOW)

        GPIO.output(RCLK, GPIO.HIGH)

        time.sleep(0.001)

        GPIO.output(RCLK, GPIO.LOW)

    for code in segCode: # Input item in segCode list to hc595_shift() function, to
        ↪ display the character.

        hc595_shift(code)

```

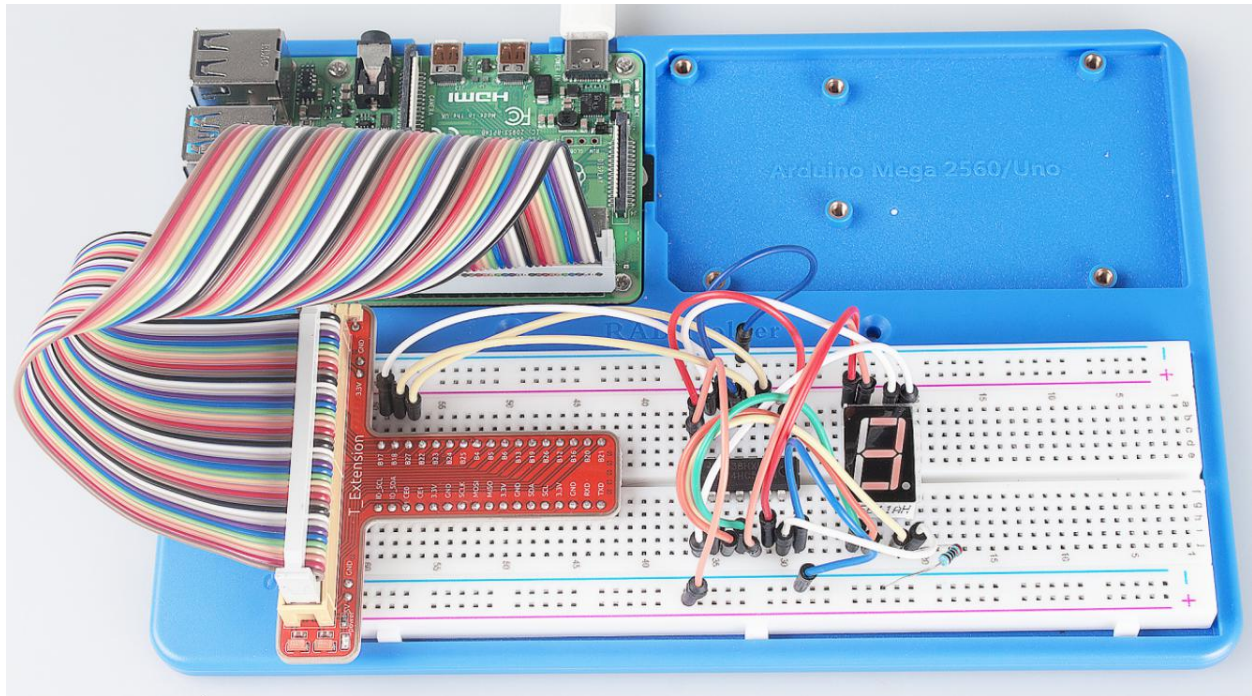
If you want to display a number, use the `hc595_shift()` function, `segCode` list and decimal value `x` in the sketch:

```
hc595_shift(segCode[x])
```

''' `x` is a number needs to be displayed ranging from 0~15, and it will be converted and displayed by 0~F in hexadecimal. '''

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

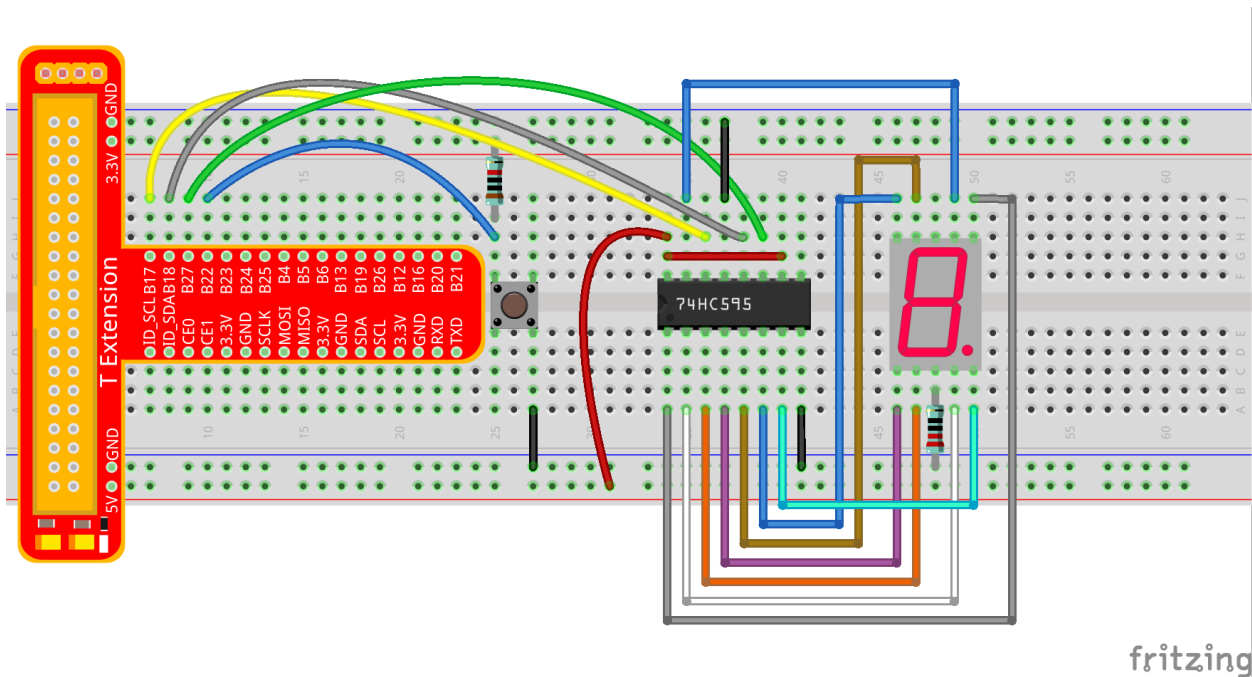
You should see the 7-segment display from 0 to 9 and A to F.



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.

Build the circuit:



Get into the folder of the code.

```
cd/home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Next, Compile the Code*.*

```
make 14_dice
```

Run.

```
sudo ./14_dice
```

Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>

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

#define TouchPin 3

unsigned char SegCode[6] = {0x06,0x5b,0x4f,0x66,0x6d,0x7d};

unsigned char flag = 0;

void init(void)
{
    pinMode(SDI, OUTPUT); //make P0 output
    pinMode(RCLK, OUTPUT); //make P1 output
    pinMode(SRCLK, OUTPUT); //make P2 output
    pinMode(TouchPin, INPUT);
    pullUpDnControl(TouchPin, PUD_UP);

    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);
}
```

(continues on next page)

(continued from previous page)

```

void randomISR(void)
{
    flag = 1;
}

int main(void)
{
    int num;

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

    init();

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("          Dice          |\n");
    printf("-----|\n");
    printf("      SDI connect to GPIO0    |\n");
    printf("      RCLK connect to GPIO1   |\n");
    printf("      SRCLK connect to GPIO 2 |\n");
    printf("      Button Pin connect to GPIO 3 |\n");
    printf("      |\n");
    printf("      Control segment with 74HC595 |\n");
    printf("      random number 0~6          |\n");
    printf("      Press to supend segment 2 second |\n");
    printf("      |\n");
    printf("      SunFounder|\n");
    printf("=====\n");
    printf("\n");
    printf("\n");

    if(wiringPiISR(TouchPin, INT_EDGE_FALLING, &randomISR)){
        printf("Unable to setup ISR : %s\n", strerror(errno));
        return 1;
    }

    srand(time(NULL));

    while(1){
        num = rand() % 6;
        hc595_shift(SegCode[num]);
        if(flag == 1){
            printf("flag = %d, ", flag);
            printf("Pressed when %d on Segment\n", (num+1));
            delay(2000);
            flag = 0;
        }
        else{
            delay(60);
        }
    }
}

```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

Code Explanation

```
void randomISR(void){ // An interrupt function, run when the interrupt happens

    flag = 1; // flag represents the state of the button

}

if(wiringPiISR(TouchPin, INT_EDGE_FALLING, &randomISR)){ /* Set an
interrupt here as the falling edge for TouchPin. When the interrupt
happens, execute the function randomISR(). */

    printf("Unable to setup ISR : %s\n", strerror(errno));

    return 1;

}

srand(time(NULL));

num = rand() % 6;

/* Two functions here: One is the srand function, which is used before
calling function rand() and used as seed for the random number
generator; while the other is rand(), which is a function to generate
the random number. Usually, these two functions are used together to
generate the random number. Thus a random number of 0-6 will be
displayed on the 7-segment display. */
```

For Python Users:

Step 2: Get into the folder of the code.

```
cd/home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 14_dice.py
```

Code

```
import RPi.GPIO as GPIO
import time
import random
from sys import version_info

if version_info.major == 3:
    raw_input = input

# Set up pins
SDI    = 17
```

(continues on next page)

(continued from previous page)

```

RCLK = 18
SRCLK = 27

TouchPin = 22

# Define a segment code from 1 to 6 in Hexadecimal
SegCode = [0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d]

# Used to record button press
flag = 0

def print_msg():
    print ("=====")
    print ("|                Dice                |")
    print ("|-----|")
    print ("|      SDI connect to GPIO17      |")
    print ("|      RCLK connect to GPIO18      |")
    print ("|      SRCLK connect to GPIO27     |")
    print ("|      Button Pin connect to GPIO22 |")
    print ("|-----|")
    print ("|      Control segment with 74HC595 |")
    print ("|      random number 1~6            |")
    print ("|      Press to supend segment 2 second |")
    print ("|-----|")
    print ("|                SunFounder |")
    print ("=====")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    raw_input ("Press Enter to begin\n")

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(TouchPin, GPIO.IN, pull_up_down = GPIO.PUD_UP)
    GPIO.add_event_detect(TouchPin, GPIO.RISING, callback = randomISR, bouncetime = 20)

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

def randomISR(channel):
    global flag
    flag = 1

def destroy():
    GPIO.cleanup()

```

(continues on next page)

(continued from previous page)

```
def main():
    global flag
    print_msg()
    while True:
        num = random.randint(1,6)
        hc595_shift(SegCode[num-1])
        print (num, hex(SegCode[num-1]))
        if flag == 1:
            print ("Num: ", num)
            time.sleep(2)
            flag = 0
        else:
            time.sleep(0.01)

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

Code Explanation

```
import random # use this function to generate the random number

SegCode = [0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d]
# Define a segment code from 1 to 6 in Hexadecimal

GPIO.add_event_detect(TouchPin, GPIO.RISING, callback = randomISR, bouncetime = 20)
''' Set an interrupt, and the rising edge for TouchPin.
When the interrupt happens, execute the function randomISR().
Set bouncetime for button to 20ms. '''

def randomISR(channel): # Interrupt calling the function

    global flag

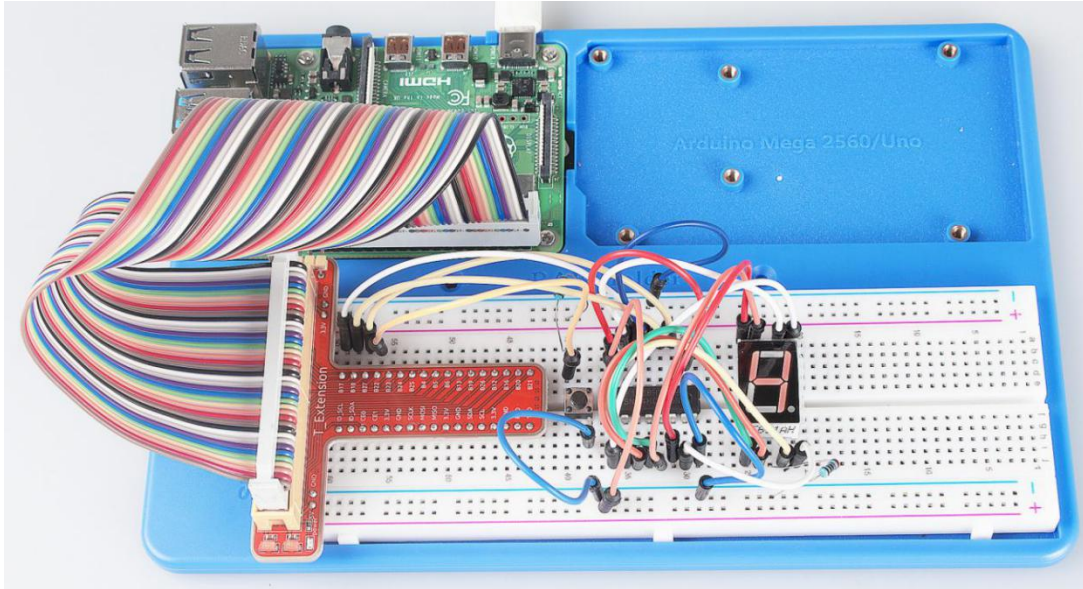
    flag = 1

    num = random.randint(1,6)

    # Generate a random number from 1~6.

    hc595_shift(SegCode[num-1]) # Output the hexadecimal values in list by 74HC595.
```

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.



6.15 Lesson 15 Driving Dot-Matrix by 74HC595

6.15.1 Introduction

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

6.15.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 2 * 74HC595
- 1 * Dot-Matrix
- Jumper wires
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable

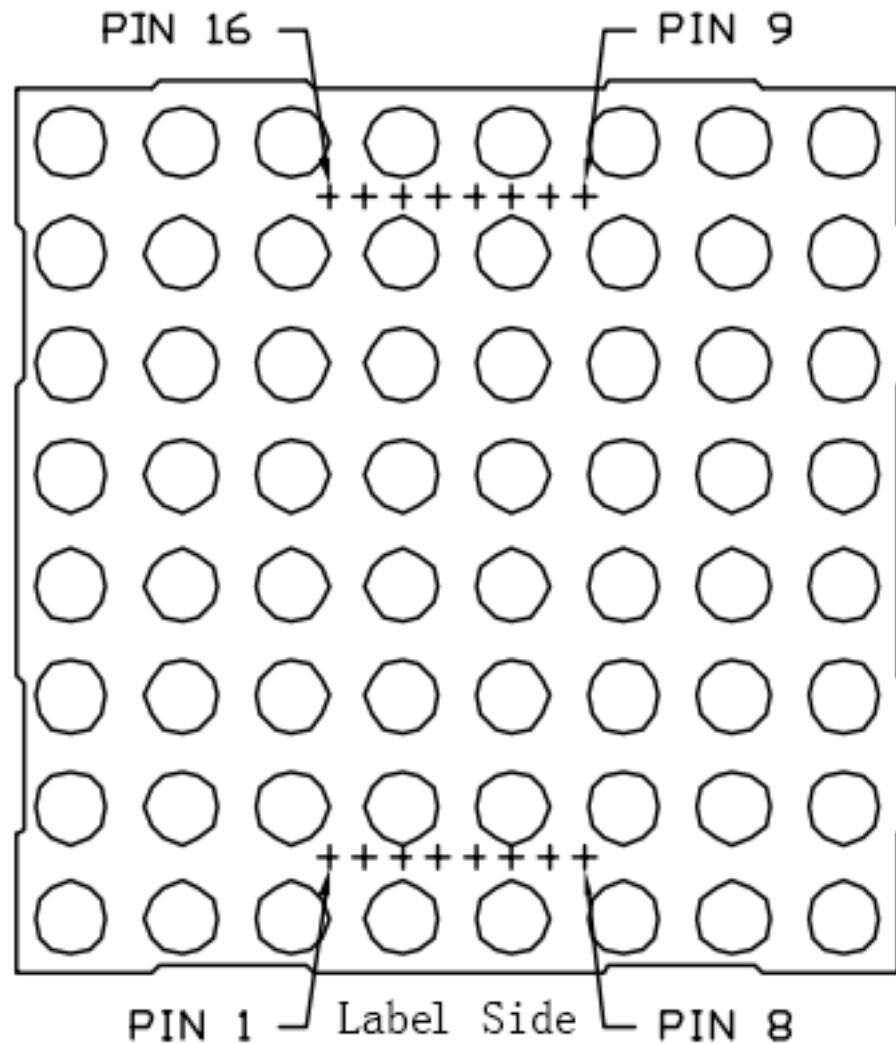
6.15.3 Principle

Dot Matrix

Generally, dot matrix can be categorized into two types: common cathode (CC) and common anode (CA). They look much alike, but internally the difference lies. You can tell by test. A CA one is used in this kit. You can see **788BS** labeled at the side.

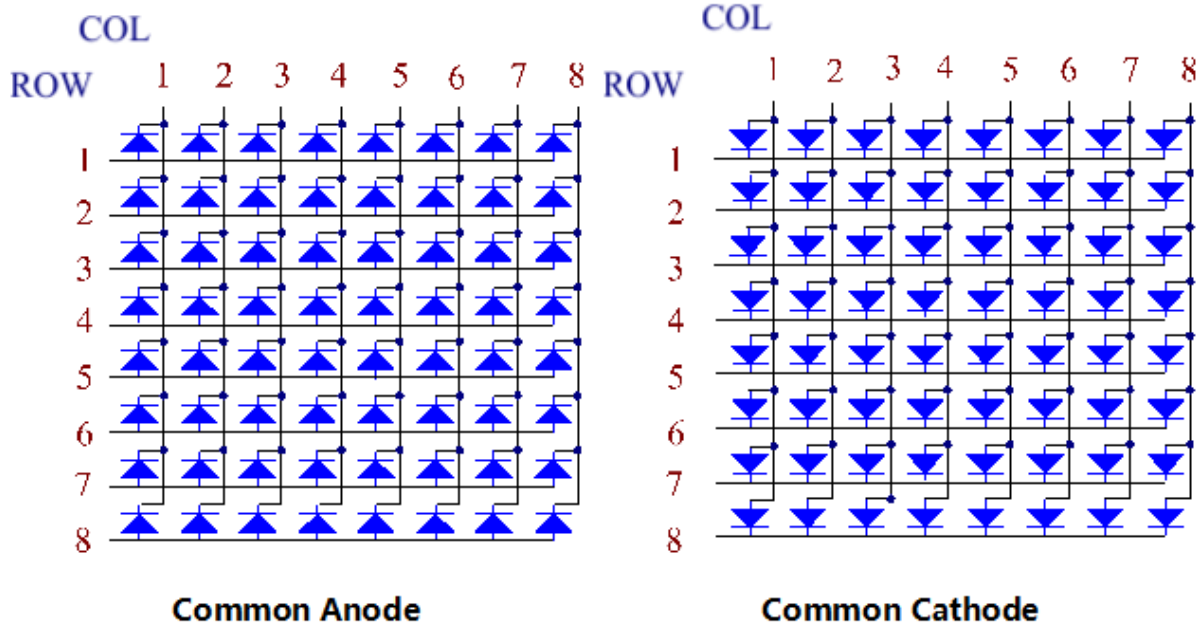
See the figure below. The pins are arranged at the two ends at the back. Take the label side for reference: pins on this end are pin 1-8, and oh the other are pin 9-16.

The external view:



Below the figures show their internal structure. You can see in a CA matrix, ROW represents the anode of the LED, and COL is cathode; it's contrary for a CC one. One thing in common: for both types, pin 13, 3, 4, 10, 6, 11, 15, and 16 are all COL, when pin 9, 14, 8, 12, 1, 7, 2, and 5 are all ROW. If you want to turn on the first LED at the top left corner, for a CA matrix, just set pin 9 as High and pin 13 as Low, and for a CC one, set pin 13 as High and pin 9 as Low. If you want to light up the whole first column, for CA, set pin 13 as Low and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as High, when for CC, set pin 13 as High and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as Low. Consider the following figures for better understanding.

The internal view:

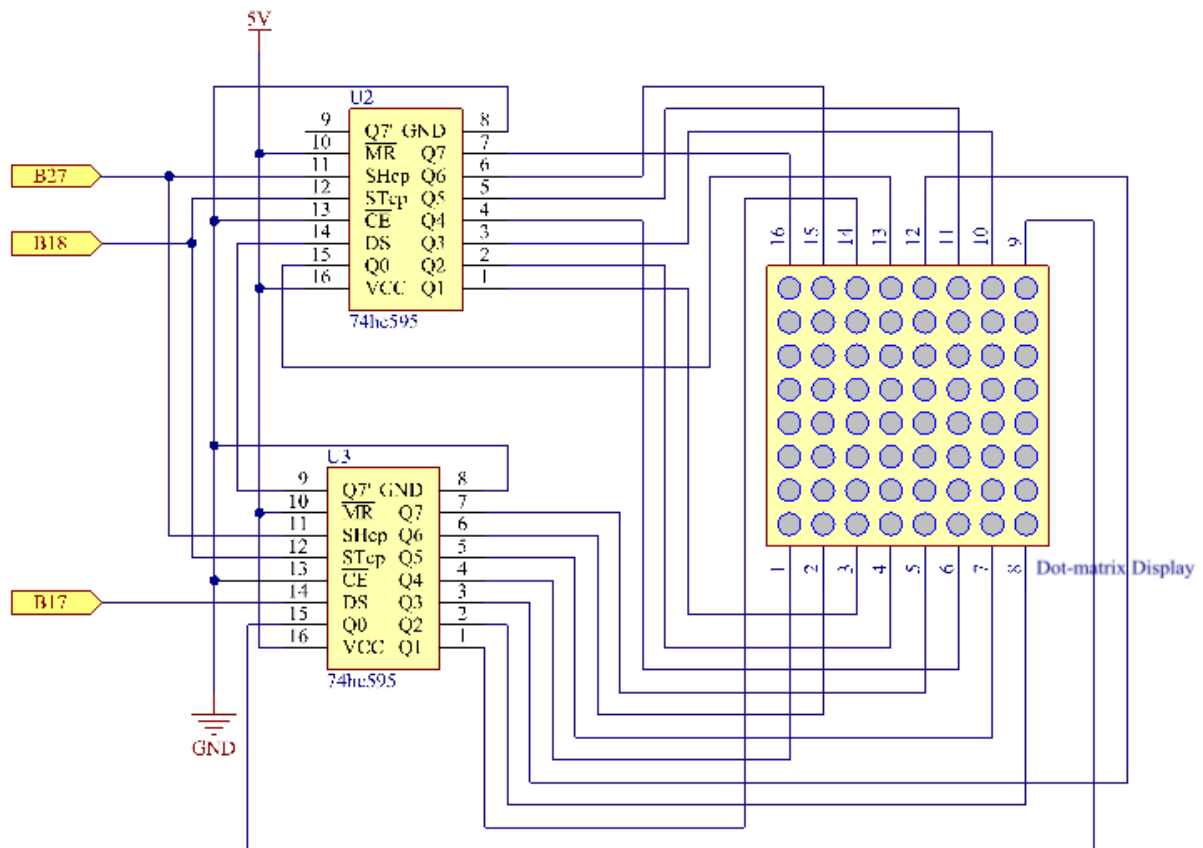


Pin numbering corresponding to the above rows and columns:

| 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 |

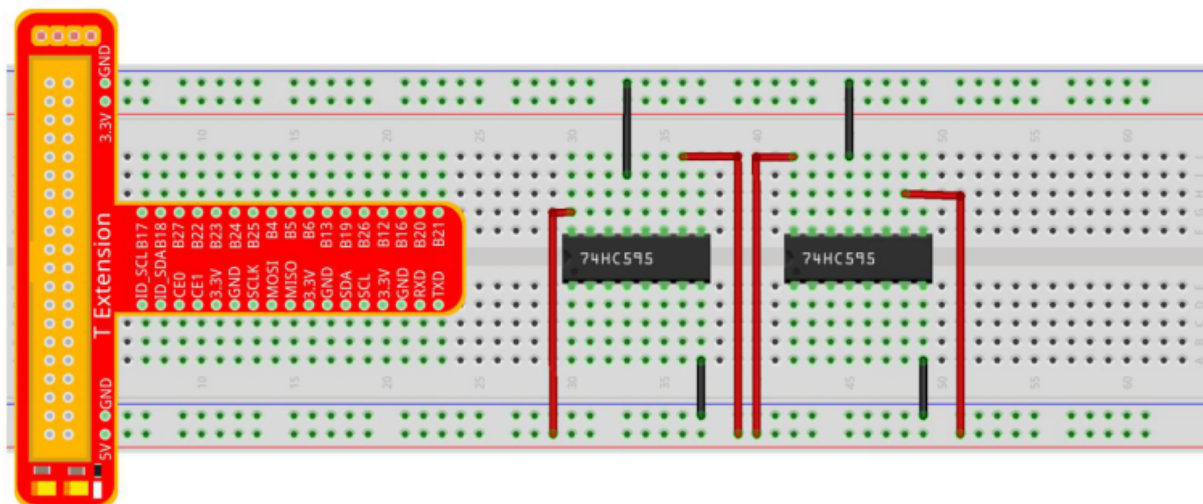
In this experiment, a CA dot matrix is used. You can see the label ends with “BS”. The wiring and code are done for the CA matrix. Therefore, if you happen to have a CC matrix, you need to change the wiring and code. In addition, two 74HC595 chips are used here. One is to control the rows of the dot matrix while the other, the columns.

6.15.4 The Schematic Diagram

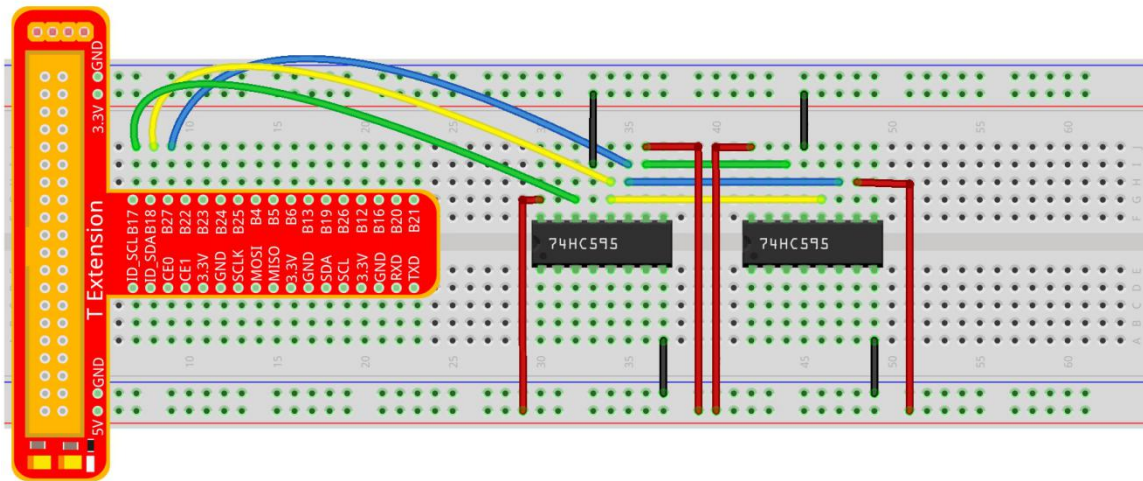


Experimental Procedures

Step 1: Build the circuit. Since the wiring is complicated, let's make it step by step. First, inset the T-Cobbler and two 74HC595 chips into breadboard. Connect the 5V and GND of the T-Cobbler to holes on the two sides of the board, then hook up pin16 and 10 of the two 74HC595 chips to VCC and pin 13 respectively, and pin 8 to GND.



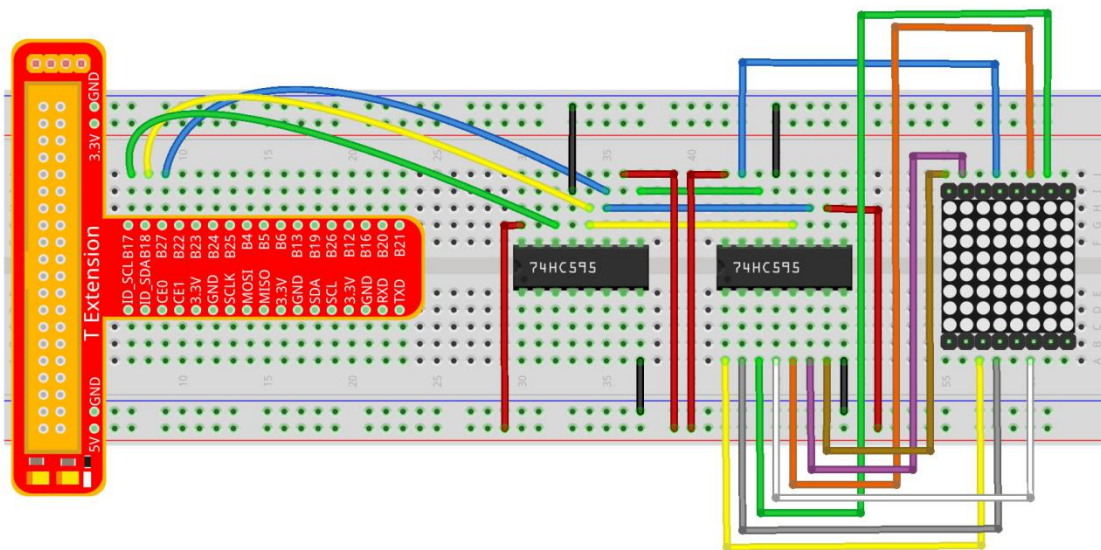
Step 2: Connect pin 11 of the two 74HC595 together, and then to GPIO27; then pin 12 of the two chips, and to GPIO18; next, pin 14 of the 74HC595 on the left side to GPIO17 and pin 9 to pin 14 of the other 74HC595.



fritzing

Step 3: Insert the dot matrix onto the breadboard. The 74HC595 on the right side is to control columns of the matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

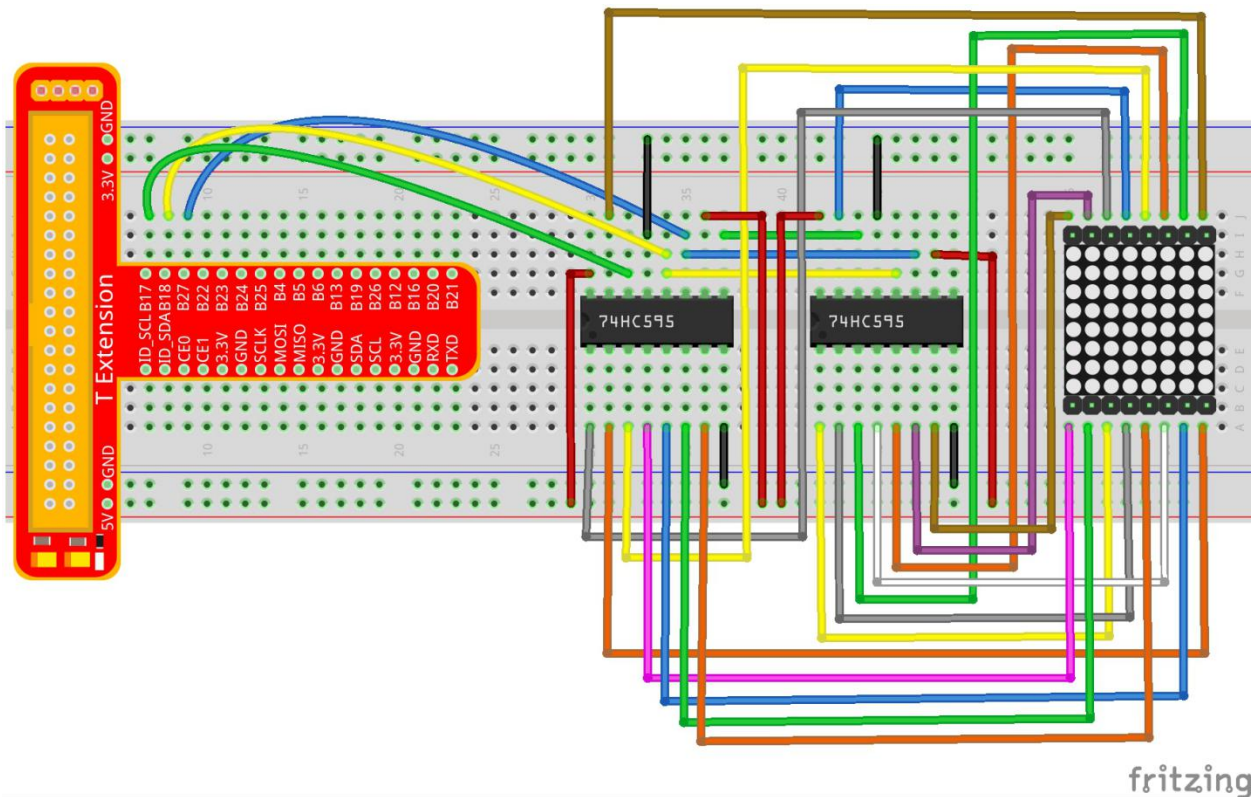
| COL | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------|----|---|---|----|---|----|----|----|
| Pin No. | 13 | 3 | 4 | 10 | 6 | 11 | 15 | 16 |



fritzing

Step 4: Now connect the ROWs of the dot matrix. The 74HC595 on the left controls ROW of the matrix. See the table below for the mapping. We can see, Q0-Q7 of the 74HC595 on the left are mapped with pin 9, 14, 8, 12, 1, 7, 2, and 5 respectively.

| ROW | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------|---|----|---|----|---|---|---|---|
| Pin No. | 9 | 14 | 8 | 12 | 1 | 7 | 2 | 5 |



Note: PLEASE connect devices correctly. DO NOT wire up insufficiently. DO NOT connect to the wrong side of the dot matrix. In the Fritzing image above, the side with label is at the bottom.

For C Language Users:

Step 2: Get into the folder of code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 15_dotMatrix
```

Step 4: Run.

```
sudo ./15_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>
```

(continues on next page)

(continued from previous page)

```

#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};
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();

    printf("\n");
    printf("\n");
    printf("=====\n");
    printf("|          Dot matrix with two 74HC595          |\n");
    printf("|          -----                               |\n");

```

(continues on next page)

(continued from previous page)

```

printf("|          SDI connect to GPIO0          |\n");
printf("|          RCLK connect to GPIO1          |\n");
printf("|          SRCLK connect to GPIO 2         |\n");
printf("|          |\n");
printf("|          Control Dot matrix with 74HC595 |\n");
printf("|          |\n");
printf("|          SunFounder |\n");
printf("=====|\n");
printf("\n");
printf("\n");

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

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

return 0;
}

```

Code Explanation

```

void hc595_in(unsigned char dat)
{
    // Write an 8-bit data to the shift register of the 74HC595

    int i;

    for(i=0;i<8;i++){
        {

            digitalWrite(SDI, 0x80 & (dat << i));
            // Write the value of dat to pin SDI of the HC595 bit by bit

            digitalWrite(SRCLK, 1); // Everytime SRCLK adds one, the shift register moves_
↪ 1 bit

            delay(1);

            digitalWrite(SRCLK, 0);

        }
    }
}

void hc595_out()
{ // Update the output data of the 74HC596

```

(continues on next page)

(continued from previous page)

```

digitalWrite(RCLK, 1); // Everytime RCLK adds 1, the HC595 updates the output.

delay(1);

digitalWrite(RCLK, 0);
}

while(1)
{
    for(i=0;i<sizeof(code_H);i++){
        // The data of ROW and COL table for the matrix adds 1 each time.

        hc595_in(code_L[i]); // Write to the first data of the Row table

        hc595_in(code_H[i]);
        // Write to the first data of the COL table, and the ROW data previously goes
        ↪to the other HC595.

        hc595_out(); /* Update the output of the 74HC595; output the data
        controlled by both two HC595, and the dot matrix will show the pattern. */

        delay(100);

    }

    for(i=sizeof(code_H);i>=0;i--){
        // The data of ROW and COL table for the matrix decreases by 1 each time.

        hc595_in(code_L[i]); // Write to the first data of the Row table

        hc595_in(code_H[i]); /* Write to the first data of the COL table, and
        the ROW data previously goes to the other HC595. */

        hc595_out(); /* Update the output of the 74HC595; output the data
        controlled by both two HC595, and the dot matrix will show the pattern. */

        delay(100);

    }
}

```

For Python Users:

Step 2: Get into the folder of code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 15_dotMatrix.py
```

Code

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

if version_info.major == 3:
    raw_input = input

SDI    = 17
RCLK   = 18
SRCLK  = 27

# we use BX matrix, ROW for anode, and COL for cathode
# ROW  ++++
code_H = [0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,
↪0xff,0xff,0xff,0xff,0xff]
# COL  ----
code_L = [0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,
↪0xf7,0xef,0xdf,0xbf,0x7f]

def print_msg():
    print ("=====")
    print ("|          Dot matrix with two 74HC595          |")
    print ("|-----|")
    print ("|          SDI connect to GPIO17                    |")
    print ("|          RCLK connect to GPIO18                    |")
    print ("|          SRCLK connect to GPIO27                   |")
    print ("|          |")
    print ("|          Control Dot matrix with 74HC595          |")
    print ("|          |")
    print ("|          SunFounder|")
    print ("=====")
    print ("Program is running...")
    print ("Please press Ctrl+C to end the program...")
    #raw_input ("Press Enter to begin\n")

def print_matrix(matrix):
    for i in range(0,len(matrix)):
        print (matrix[i])

def get_matrix(row_buffer, col_buffer, max_row=8, max_col=8):
    matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)]

    print("row_buffer = 0x%02x , col_buffer = 0x%02x"%(row_buffer, col_buffer))
    for row_num in range(0,8):
        for col_num in range(0,8):
            #print (row_num, col_num), '-->', (((row_buffer >> row_num) & 0x01),
↪(((col_buffer >> col_num) & 0x01))
```

(continues on next page)

(continued from previous page)

```

        if ((row_buffer >> row_num) & 0x01) - ((col_buffer >> col_num) & 0x01)):
            matrix_msg[row_num][col_num] = 1
    print_matrix(matrix_msg)
    matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)]

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

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

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

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

def destroy():
    GPIO.cleanup()

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

```

Code Explanation

```

# We use a Common Anode matrix, so ROW pins are the common anode, and COL, the common_
↪cathode.

''' row and column lists. When characters are displayed, an element in row
and one in column are acquired and assigned to the two HC595 chips

```

(continues on next page)

(continued from previous page)

```

respectively. Thus a pattern is shown on the matrix. '''

# ROW ++++

code_H = [0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
          0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff]

# COL ----

code_L = [0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
          0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f]

def get_matrix(row_buffer, col_buffer, max_row=8, max_col=8):
    # The functions is to print the pattern on the matrix by the 2D array on the command_
    ↪line interface (CLI).

    matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)] # Initialize a_
    ↪2D array

    print ("row_buffer = 0x%02x , col_buffer = 0x%02x"%(row_buffer, col_buffer))

    for row_num in range(0,8):

        for col_num in range(0,8):

            if (((row_buffer >> row_num) & 0x01) - ((col_buffer >> col_num) & 0x01)):
                # for Common Anode type matrix, when row is High and column is low, the_
                ↪LED will light up.

                matrix_msg[row_num][col_num] = 1 ''' To turn on an LED at a certain_
                ↪row
                and column, assign 1 to the corresponding elements in the 2D array'''

    print_matrix(matrix_msg) # Print the 2D array on the CLI

    matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)]
    # Reset the array after one print

def hc595_shift(dat): # Shift the data to 74HC595

    for bit in range(0, 8):

        GPIO.output(SDI, 0x80 & (dat << bit)) # Write the value of dat bit by bit to_
        ↪pin SDI of the HC595

        GPIO.output(SRCLK, GPIO.HIGH) # Everytime SRCLK is High, the shift register_
        ↪shifts one bit

        time.sleep(0.001)

        GPIO.output(SRCLK, GPIO.LOW)

        GPIO.output(RCLK, GPIO.HIGH) # Everytime RCLK is high, HC595 updates its output.

        time.sleep(0.001)

        GPIO.output(RCLK, GPIO.LOW)

```

(continues on next page)

(continued from previous page)

```

def main():

    print_msg()

    while True:

        for i in range(0, len(code_H)): #Assign elements of the column table in_
            ↪sequence

            hc595_shift(code_L[i])# Write to the first data of the Row table

            hc595_shift(code_H[i])
            # Write to the first data of the COL table, and the ROW data previously_
            ↪goes to the other HC595.

            get_matrix(code_L[i], code_H[i]) # Print the 2D array on the CLI

            time.sleep(0.1)

        for i in range(len(code_H)-1, -1, -1): #Assign elements of the column table_
            ↪in inverse order

            hc595_shift(code_L[i])

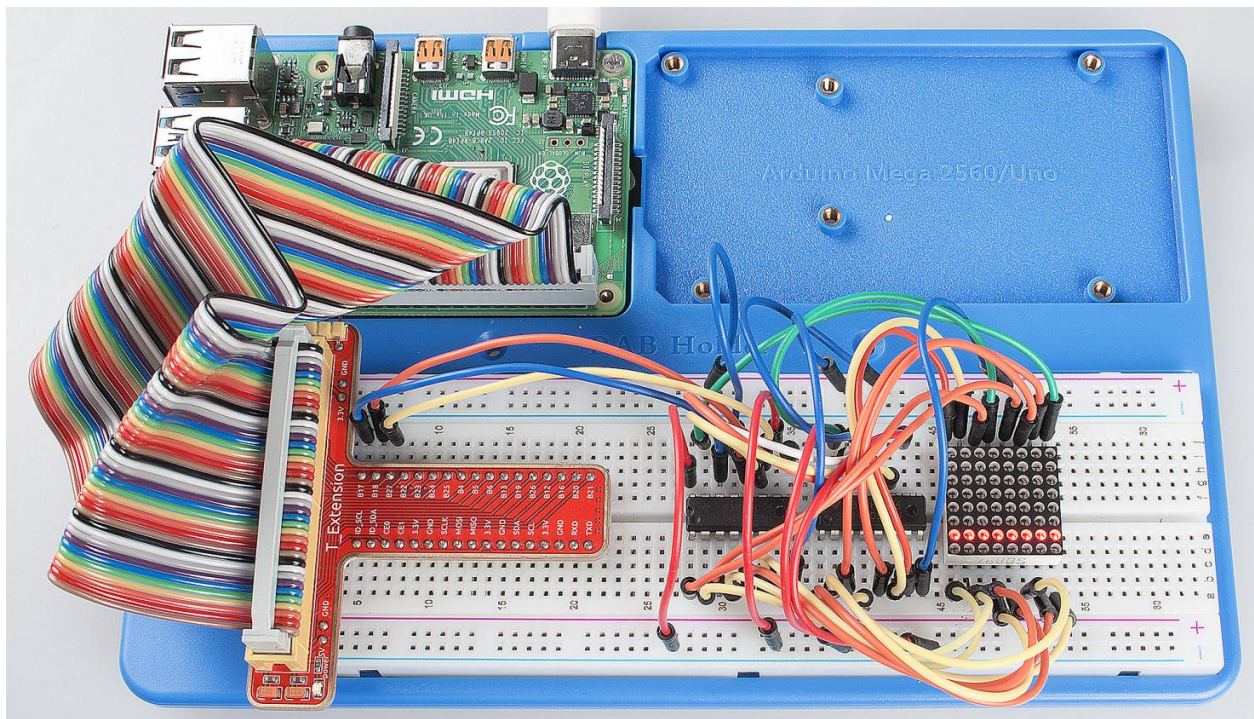
            hc595_shift(code_H[i])

            get_matrix(code_L[i], code_H[i])

            time.sleep(0.1)

```

You should see LEDs light up as you control.



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!

Further Exploration

If you want to display characters on the matrix, please refer to a python code: https://github.com/sunfounder/SunFounder_Dot_Matrix.

6.16 Lesson 16 LCD1602

6.16.1 Introduction

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



6.16.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * LCD1602
- 1 * Potentiometer
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable
- Jumper wires

6.16.3 Principle

LCD1602

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the SunFounder Uno board are almost completely occupied. If you want to connect more sensors, there will be no ports available. Therefore, the four-port connection is used here for better application.

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

Pin 1 (GND): connected to Ground

Pin 2 (V_{ce}): connected to 5V power supply

Pin 3 (V_o): 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, controlling 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, where the LCD's controller looks for instructions on what to do next.

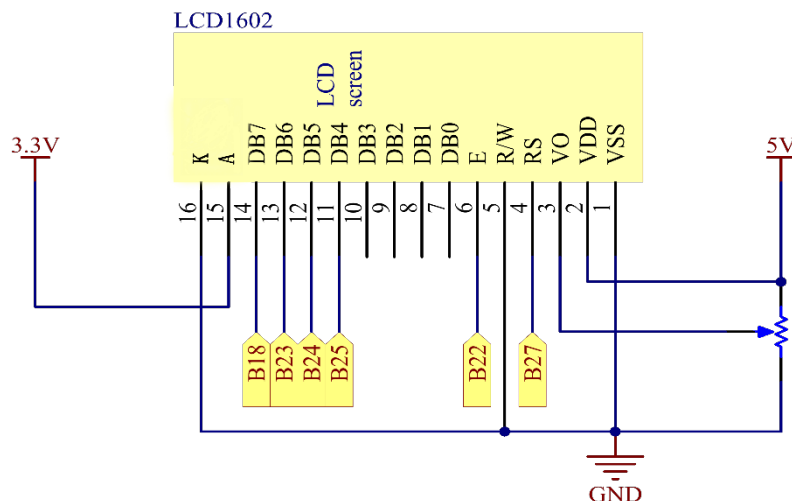
Pin 5 (R/W): to read/write signals; it reads signals when supplied with high level (1), and writes when low level (0) (in this experiment, you only need to write data to LCD1602, so just connect this pin to ground)

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; K connects to GND, and A to 3.3V. Turn the backlight on and you can see the characters displayed clear in a dim environment

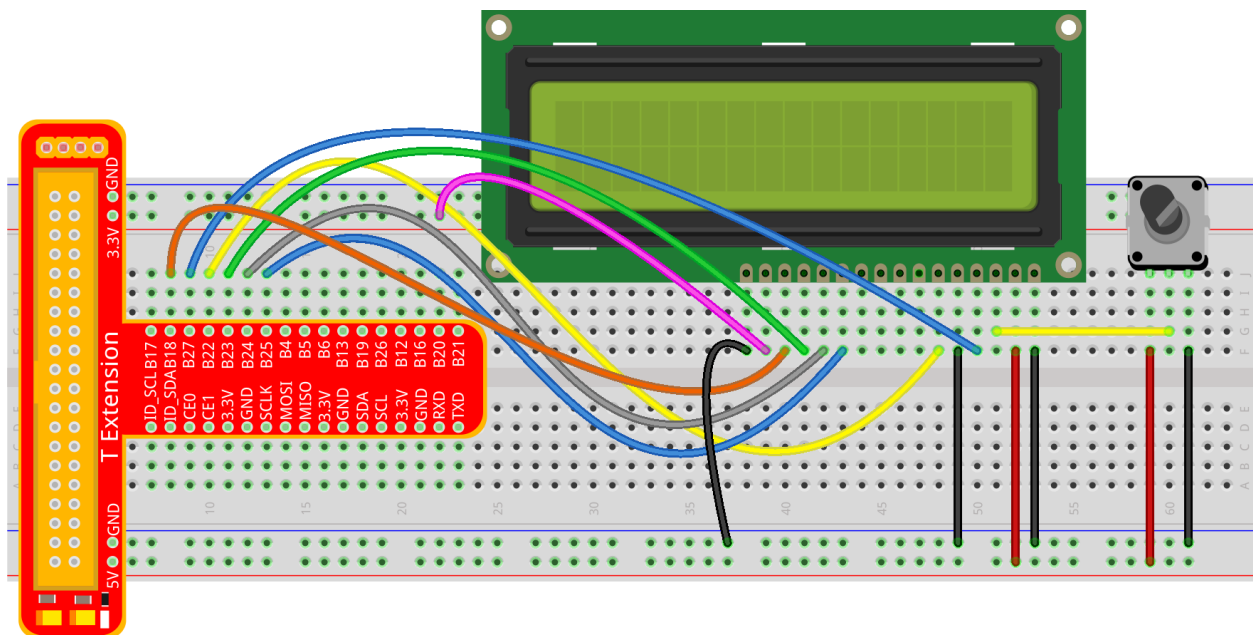
LCD1602 has two operation modes: 4-bit and 8-bit. When the IOs of the MCU are insufficient, you can choose the 4-bit mode, under which only pins D4~D7 are used. After connecting the circuit, you can operate LCD1602 by the Raspberry Pi.



6.16.4 Experimental Procedures

Step 1: Build the circuit (please be sure the pins are connected correctly. Otherwise, characters will not be displayed properly):

| LCD1602 | T-Extension Board |
|---------|--|
| VDD | 5V |
| VSS | GND |
| OV | Connect to the middle pin of potentiometer |
| RS | B27 |
| R/W | GND |
| E | B22 |
| D0-D3 | Not connected |
| D4 | B25 |
| D5 | B24 |
| D6 | B23 |
| D7 | B18 |
| A | 3.3V |
| K | GND |



fritzing

Note: After you run the code, characters may not appear on the LCD1602. You need to adjust the contrast of the screen (the gradual change from black to white) by spinning the potentiometer clockwise or anticlockwise, until the screen displays characters clearly.

For C Language Users:**Step 2:** Get into the folder of code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile.

```
make 16_lcd1602
```

Step 4: Run.

```
sudo ./16_lcd1602
```

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)
{
    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;
    }
    sleep(1);
    lcdClear(fd);
    lcdPosition(fd, 0, 0);
    lcdPuts(fd, "Welcome To-->");

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

    sleep(1);
    lcdClear(fd);

    printf("\n");
    printf("\n");
    printf("=====\n");
```

(continues on next page)

(continued from previous page)

```

printf("|                LCD1602                |\n");
printf("| ----- |\n");
printf("|      D4 connect to GPIO6      |\n");
printf("|      D5 connect to GPIO5      |\n");
printf("|      D6 connect to GPIO4      |\n");
printf("|      D7 connect to GPIO1      |\n");
printf("|      RS connect to GPIO2      |\n");
printf("|      RW connect to GND        |\n");
printf("|      CE connect to GPIO3      |\n");
printf("|                Control LCD1602                |\n");
printf("|                SunFounder                |\n");
printf("=====|\n");
printf("\n");
printf("\n");

while(1){
    lcdClear(fd);
    for(i=0; i<16; i++){
        lcdPosition(fd, i, 0);
        lcdPutchar(fd, *(myBuf+i));
        delay(100);
    }
    for(i=0; i<sizeof(Buf)-1; i++){
        lcdPosition(fd, i, 1);
        lcdPutchar(fd, *(Buf+i));
        delay(200);
    }
    sleep(0.5);
}
return 0;
}

```

Code Explanation

```

#include <lcd.h>
// includes the lcd library, containing some functions for the LCD1602 display for
↳convenient use

const unsigned char Buf[] = "---SUNFOUNDER---";
// An array to store the characters to be displayed on the LCD1602

const unsigned char myBuf[] = " sunfounder.com";
// Another array to store the characters

fd = lcdInit(2,16,4, 2,3, 6,5,4,1,0,0,0,0);
// Initialize the LCD display, see /usr/local/include/lcd.h

/* lcdInit(rows, cols, bits, rs, strb, d0, d1, d2, d3, d4, d5, d6, d7) -
LCD1602 shows 2 rows and 16 columns. If the initialization succeeds, it
will return True. */

lcdClear(fd); // Clear the screen

lcdPosition(fd, 0, 0);
// Locate the position of the cursor at Row 0 and Col 0 (in fact it's the first line
↳and first column)

```

(continues on next page)

(continued from previous page)

```

lcdPuts(fd, "Welcom To--->");
// Display the character "Welcom To--->"on the LCD1602

lcdPosition(fd, 0, 1); // Place the cursor at Col 0, Row 0.

lcdPuts(fd, " sunfounder.com");

while(1)
{
    lcdClear(fd);

    for(i=0; i<16; i++)
    { // i adds one in the loop. i means the number of columns, so i adds to 16 at
      ↪most.

        lcdPosition(fd, i, 0);
        // Place the cursor at the first row, and moves left to right from the first
      ↪character

        lcdPuchar(fd, *(myBuf+i));
        // *(myBuf+i) is a pointer that points to contents in the myBuf[] array, and
      ↪output the pointed data to lcd

        delay(100);

    }

    for(i=0; i<sizeof(Buf)-1; i++)
    {

        lcdPosition(fd, i, 1); // Place the cursor at the second row, moves from the
      ↪first character

        lcdPuchar(fd, *(Buf+i)); // A pointer that points to data in the Buf[] array;
      ↪ output it to lcd

        delay(200);

    }

    sleep(0.5);
}

```

For Python Users:**Step 2:** Get into the folder of code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 16_lcd1602.py
```

Code

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

if version_info.major == 3:
    raw_input = input

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
```

(continues on next page)

(continued from previous page)

```

        self.pin_e = pin_e
        self.pins_db = pins_db

        self.used_gpio = self.pins_db[:]
        self.used_gpio.append(pin_e)
        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_SETDRAMADDR | (col + self.row_offsets[row]))

```

(continues on next page)

(continued from previous page)

```

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

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 DisplayLeft(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

```

(continues on next page)

(continued from previous page)

```

self.displaymode &= ~self.LCD_ENTRYSHIFTINCREMENT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

def write4bits(self, bits, char_mode=False):
    # Send command to LCD
    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 print_msg():
    print ("=====")
    print ("| LCD1602 |")
    print ("| ----- |")
    print ("| D4 connect to GPIO25 |")
    print ("| D5 connect to GPIO24 |")
    print ("| D6 connect to GPIO23 |")
    print ("| D7 connect to GPIO18 |")

```

(continues on next page)

(continued from previous page)

```

print ("|          RS connect to GPIO27          |")
print ("|          CE connect to GPIO22          |")
print ("|          RW connect to GND          |")
print ("|          |          |          |")
print ("|          Control LCD1602          |")
print ("|          |          |          |")
print ("|          SunFounder          |")
print ("=====\\n")
print ("Program is running...")
print ("Please press Ctrl+C to end the program...")
#raw_input ("Press Enter to begin\\n")

def main():
    global lcd
    print_msg()
    lcd = LCD()
    line0 = "  sunfounder.com"
    line1 = "---SUNFOUNDER---"

    lcd.clear()
    lcd.message("Welcome to --->\\n  sunfounder.com")
    sleep(3)

    msg = "%s\\n%s" % (line0, line1)
    while True:
        lcd.begin(0, 2)
        lcd.clear()
        for i in range(0, len(line0)):
            lcd.setCursor(i, 0)
            lcd.message(line0[i])
            sleep(0.1)
        for i in range(0, len(line1)):
            lcd.setCursor(i, 1)
            lcd.message(line1[i])
            sleep(0.1)
        sleep(1)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        lcd.clear()
        lcd.destroy()

```

Code Explanation

```

class LCD: # Write an LCD class

def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO = None):
    ''' Initialization function for the class, run when an object is created of the class.
    A parameter needs to be transferred to the object when it's created; otherwise,
    the default value in __init__ will be assigned. '''

    self.used_gpio = self.pins_db[:] ''' Note down the used gpio to easily
    clear IO setting after the stop. pins_db[:] writes all in the pins_db
    list to the used_gpio list; if here use used_gpio = self.pins_db, it
    means used_gpio call pins_db, in other words, any change of pins_db will

```

(continues on next page)

(continued from previous page)

```

affect used_gpio. '''

self.used_gpio.append(pin_e)

self.used_gpio.append(pin_rs)

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

""" 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

def begin(self, cols, lines): # Start the LCD

def setCursor(self, col, row): # Set the cursor location

def message(self, text): # Send strings to the LCD. The new line wraps to the second_
↪line

def destroy(self): # Clean up the used gpio

lcd = LCD(0, 2) # Create an lcd object

lcd.clear() # Clear the LCD display

for i in range(0, len(line0)): # i adds 1 each time within the length of the_
↪character line0

    lcd.setCursor(i, 0) # Locate the cursor at character No. i, Row 0

    lcd.message(line0[i]) # Display the character on the screen

    sleep(0.1)

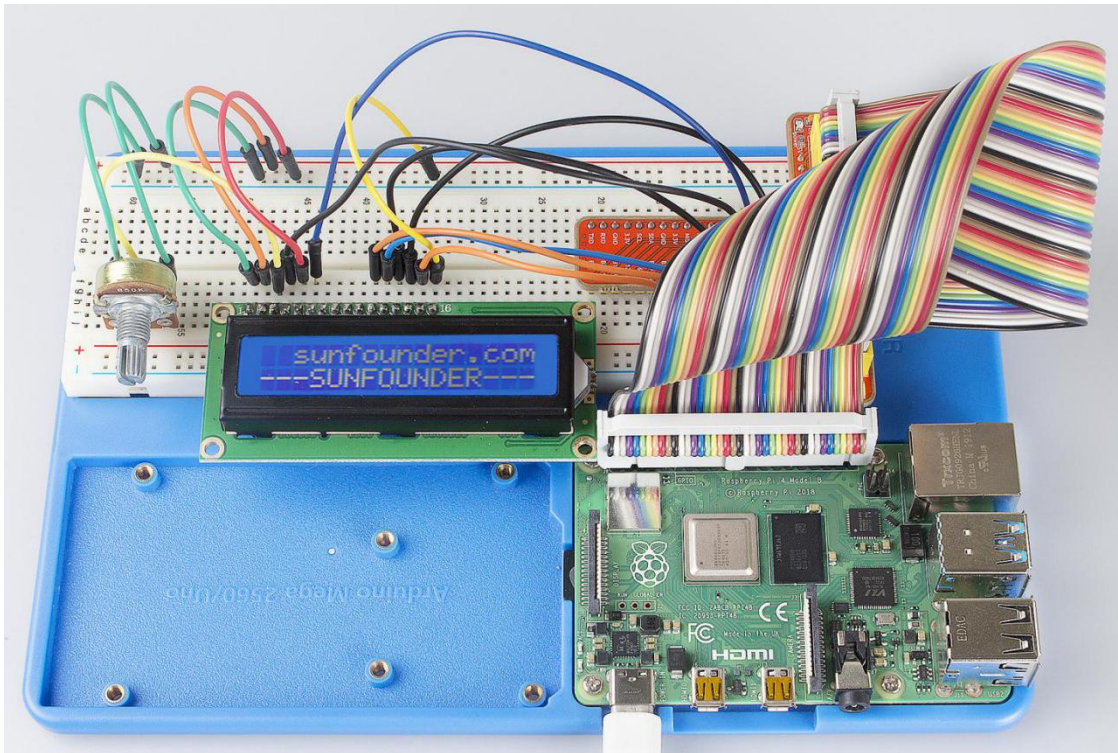
for i in range(0, len(line1)): # i adds 1 each time within the length of the_
↪character line0

    lcd.setCursor(i, 1) # Locate the cursor at character No. i, Row 1

    lcd.message(line1[i]) # Display the character on the LCD

```

You should see two lines of characters displayed on the LCD1602: " **Welcome to —>** ", " **sunfounder.com** " and " **—SUNFOUNDER—** ".



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.

6.17 Lesson 17 ADXL345

6.17.1 Introduction

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

6.17.2 Components

- 1 * Raspberry Pi
- 1 * Breadboard
- 1 * ADXL345 module
- 1 * T-Extension Board
- 1 * 40-Pin GPIO Cable
- Jumper wires

6.17.3 Principle

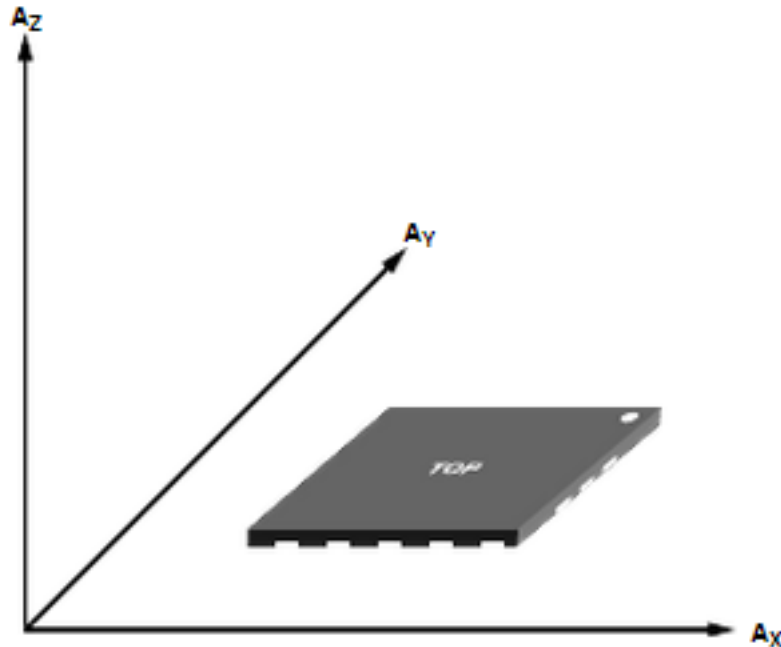
ADXL345

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 ± 16 g.

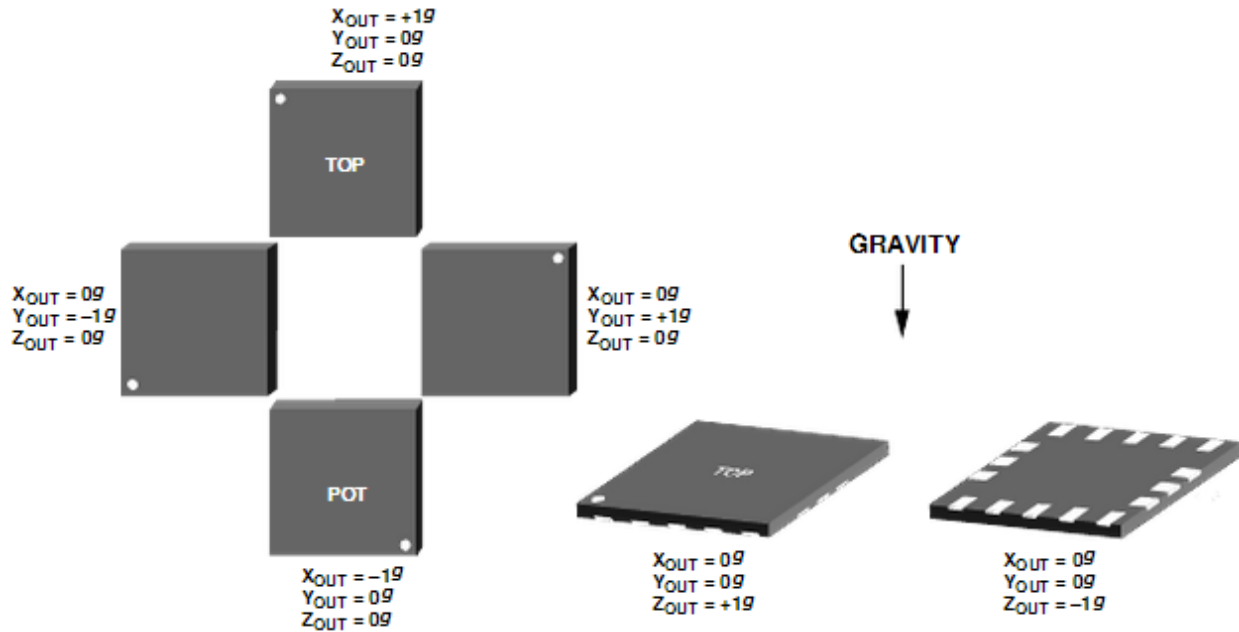
In this experiment, I2C digital interface is used.

ADXL345 works like this:



Axes of detection by ADXL345

When you place the module face up, **Z_OUT** is at the maximum which is +1g; face down, **Z_OUT** is at the minimum. No matter of face, as long as it's placed on a level surface, **X_OUT** increases along the **Ax** axis direction, so does **Y_OUT** along the **Ay** axis. See the picture below. Thus, when you rotate the module, you can see the changes of **X_OUT**, **Y_OUT**, and **Z_OUT**.



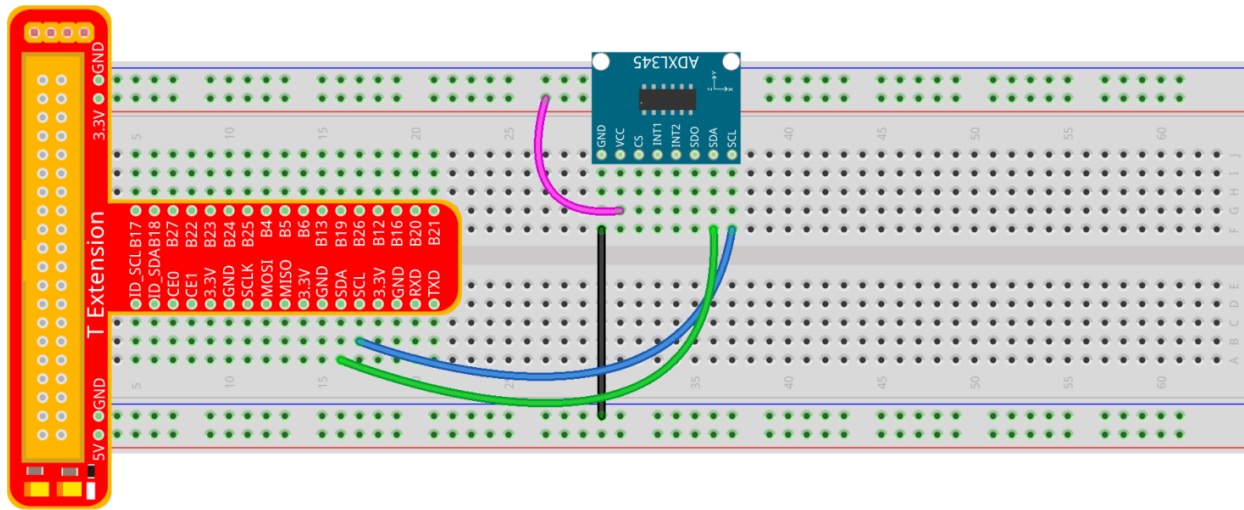
Relationship between output and gravity direction

Pin Function of ADXL345 Module:

| Name | Description |
|------|---|
| VS | Supply Voltage |
| CS | Chip Select; I2C mode is enabled if it's tie-high to VDD I/O (VDD I/O = 1.8V). |
| SDO | Serial Data Out, alternate I2C address select |
| INT1 | Interrupt 1 Output |
| INT2 | Interrupt 2 Output |
| 3.3V | 3.3V |
| SDA | Serial Data (I2C), Serial Data In (SPI 4-Wire), Serial Data In/Out (SPI 3-Wire) |
| SCL | Serial Communications Clock |
| GND | GND |

6.17.4 Experimental Procedures

Step 1: Build the circuit.



fritzing

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.

For C Language Users:

Step 2: Get into the folder of code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Step 3: Compile the Code.

```
make 17_adxl345
```

Step 4: Run.

```
sudo ./17_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 <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define DevAddr 0x53 //device address
#define SENSITIVITY 256.00 //the sensitivity at each axis is 256LSB/g in +-2g,10bit
//or full resolution

struct acc_dat{
```

(continues on next page)

(continued from previous page)

```

    int x;
    int y;
    int z;
};

void adxl345_init(int fd)
{
    wiringPiI2CWriteReg8(fd, 0x31, 0x0b); //set the range as +/-16g & full resolution
    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);
        float x = acc_xyz.x/SENSITIVITY;
        float y = acc_xyz.y/SENSITIVITY;
        float z = acc_xyz.z/SENSITIVITY;
        printf("x: %.2f y: %.2f z: %.2f\n", x,y,z);

        sleep(1);
    }

    return 0;
}

```

Code Explanation

```

#include <wiringPiI2C.h> // Include functions and method for the IIC protocol

#define DevAddr 0x53 // device address

struct acc_dat
{ // a struct variable to store the value of xyand z

    int x;

    int y;

    int z;

};

fd = wiringPiI2CSetup(DevAddr); // This initialises the I2C system with your given_
↪device identifier

void adxl345_init(int fd)
{ // Initialize the device by i2c

    wiringPiI2CWriteReg8(fd, 0x31, 0x0b);

```

(continues on next page)

(continued from previous page)

```

    // These write an 8-bit data value into the device register indicated.

    wiringPiI2CWriteReg8(fd, 0x2d, 0x08);
    // Write 0x08 to the address(0x21) of the i2c device
}

struct acc_dat adxl345_read_xyz(int fd)
{
    // a struct function, returning a struct value

    char x0, y0, z0, x1, y1, z1;

    struct acc_dat acc_xyz;

    x0 = 0xff - wiringPiI2CReadReg8(fd, 0x32);
    // These read an 8- or 16-bit value from the device register indicated.

    x1 = 0xff - wiringPiI2CReadReg8(fd, 0x33);
    // Read an 8-bit data from the 0x33 register of the I2C device fd, assign to x1

    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;
    // Assign values to members of the struct; the value of x consists of x1 (high 8_
    ↪bits) and x0 (low 8 bits).

    acc_xyz.y = (int) (y1 << 8) + (int)y0;

    acc_xyz.z = (int) (z1 << 8) + (int)z0;

    if(acc_xyz.x > 32767)
    { // Set the value of x as no more than 0x7FFF

        acc_xyz.x -= 65536;

    }

    if(acc_xyz.y > 32767)
    { // Set the value of y as no more than 0x7FFF

        acc_xyz.y -= 65536;

    }
}

```

(continues on next page)

(continued from previous page)

```

    if(acc_xyz.z > 32767)
    {

        acc_xyz.z -= 65536;

    }

    return acc_xyz; // The function ends, return to the acc_xyz struct
}

acc_xyz = adxl345_read_xyz(fd);
// Call the function to read the data collected by the accelerometer module

printf("x: %05d y: %05d z: %05d\n", acc_xyz.x, acc_xyz.y, acc_xyz.z);
// Print the data collected by the accelerometer; %05d means the printed
// data is a 5-bit one, and the empty bit will be replaced by 0.

```

For Python Users:**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Step 3: Run.

```
sudo python3 17_adxl345.py
```

Code

```

from I2C import I2C
from time import sleep

class ADXL345(I2C):

    ADXL345_ADDRESS          = 0x53
    ADXL345_REG_DATA_FORMAT = 0x31
    ADXL345_REG_DEVID        = 0x00 # Device ID
    ADXL345_REG_DATA_X0      = 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_25_HZ = 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

```

(continues on next page)

(continued from previous page)

```

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
ADXL345_SENSITIVITY     = 256.00 # 256LSB/g in full resolution

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
    self.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 = []
    for i in range(0, 6, 2):
        g = raw[i] | (raw[i+1] << 8)
        if g > 32767:
            g -= 65535
        res.append(g/self.ADXL345_SENSITIVITY)
    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()

```

(continues on next page)

(continued from previous page)

```

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

```

Code Explanation

```

class ADXL345(I2C): # Define a class ADXL345 and the class inheritance is I2C

def __init__(self, busnum=1, debug=False):
    # The initialize function of the class, which is run when an instance is created of
    # the class

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:

def getRange(self): # Read an 8-bit data from the device register

def setDataRate(self, dataRate):
    # Note: The LOW_POWER bits are currently ignored; we always keep the device in 'normal
    # mode

def getDataRate(self): # get the rate from the register

def read(self): # Read data from the accelerometer

    raw = self.accel.readList(self.ADXL345_REG_DATA0, 6)
    # Read 6 values from the register, respectively equal to the high and low bits of
    # the x, y, and z value

    print ( raw)

    res = []

    for i in range(0, 6, 2):

        g = raw[i] | (raw[i+1] << 8)
        # Combine the high 8 bits and low 8 bits and obtain a measurement value g =
        # 65535-g

        if g > 32767:

            g -= 65535

        res.append(g)

    return res

accel = ADXL345() # Create an instance accel of class ADXL345

```

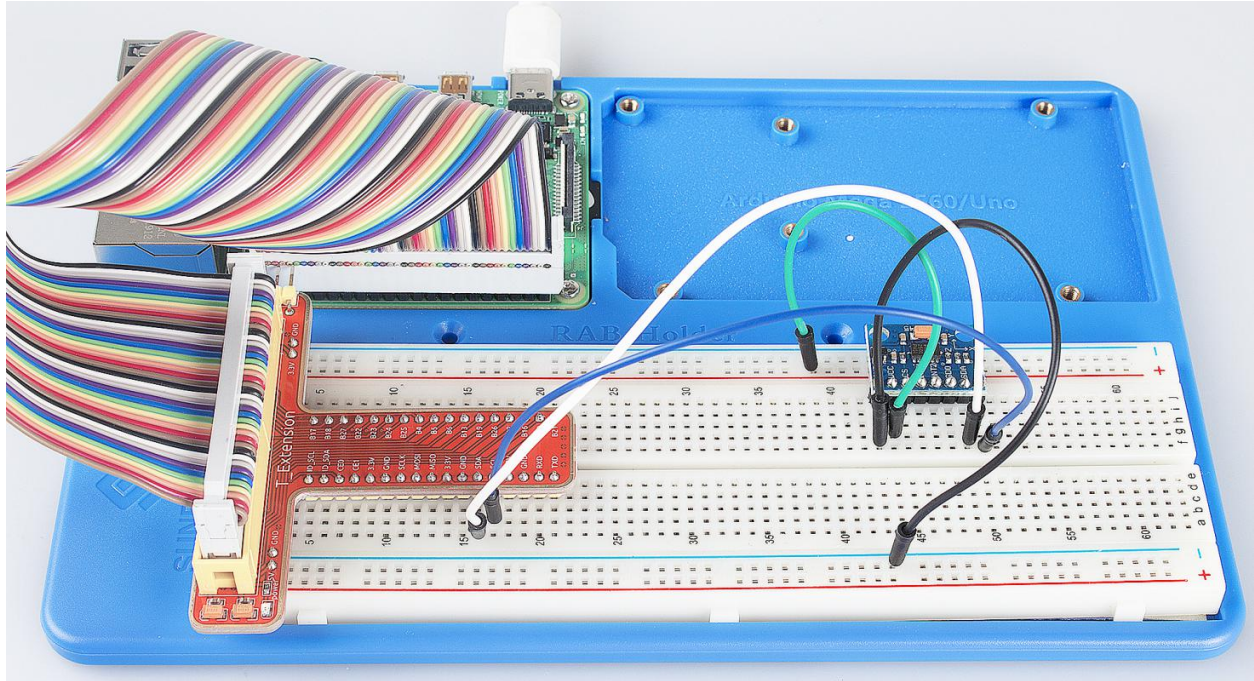
(continues on next page)

(continued from previous page)

```
x, y, z = accel.read() # accel calls itself to measure x, y, and z and store them in_
↳ a list.

# Then assign the values measured to x, y, and z.
```

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

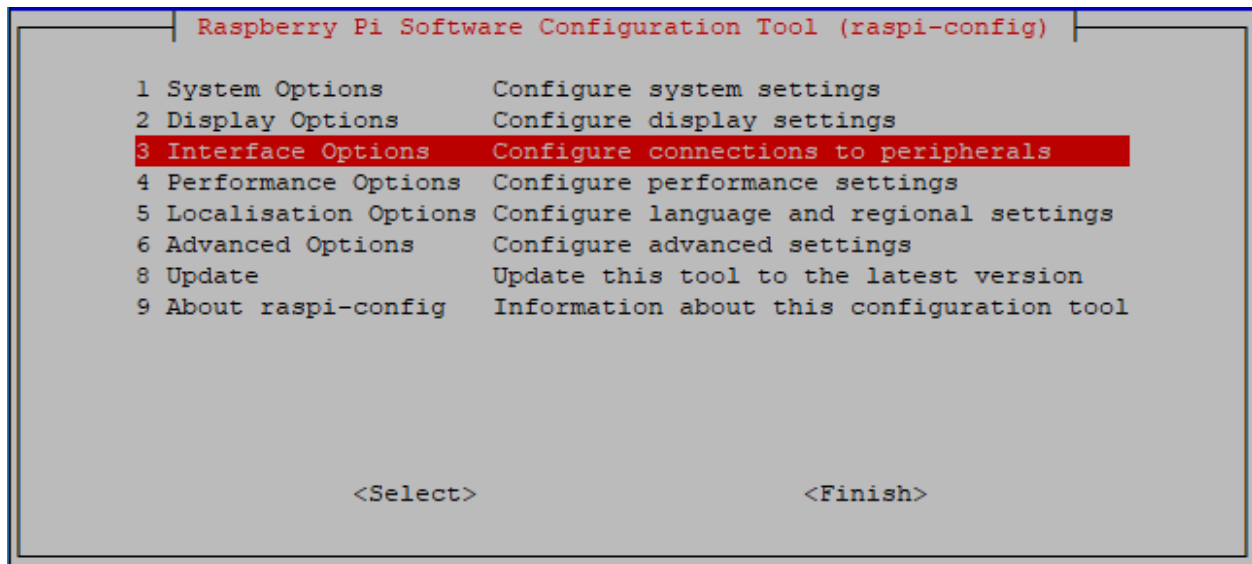


7.1 I2C Configuration

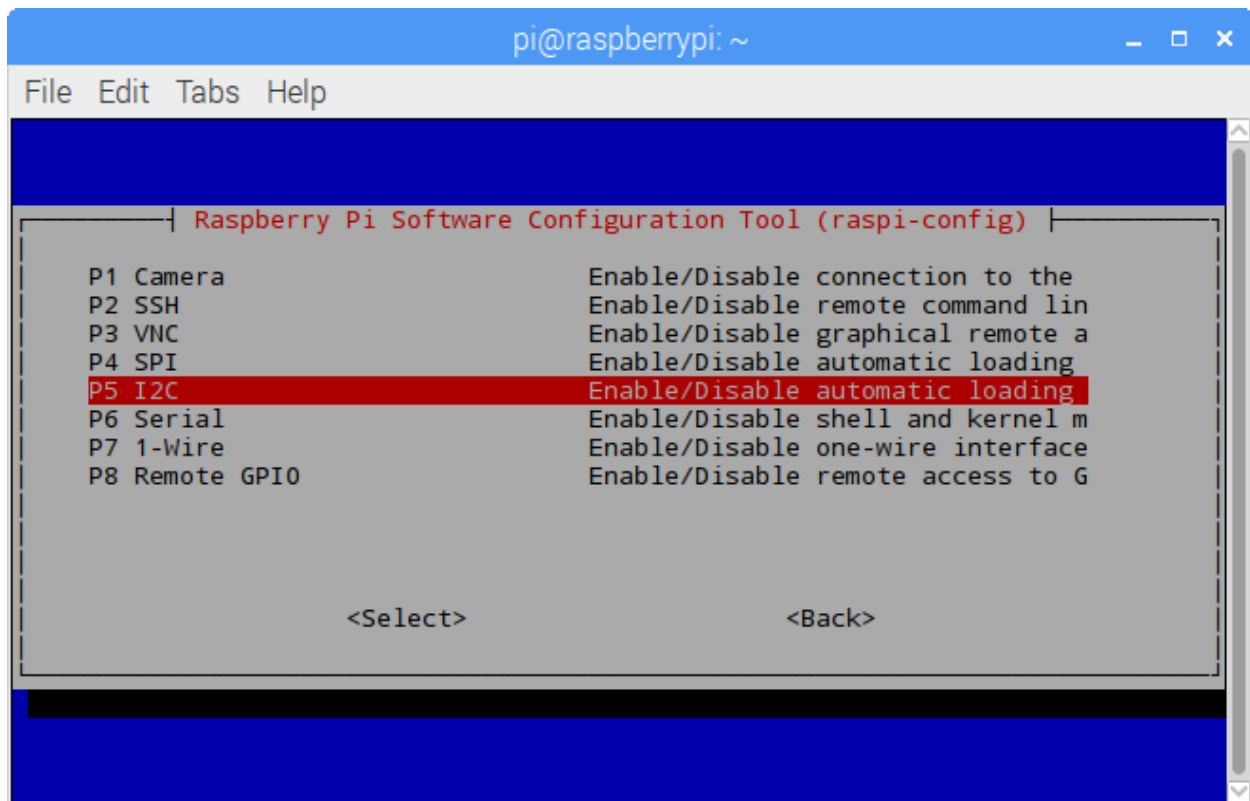
Step 1: Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

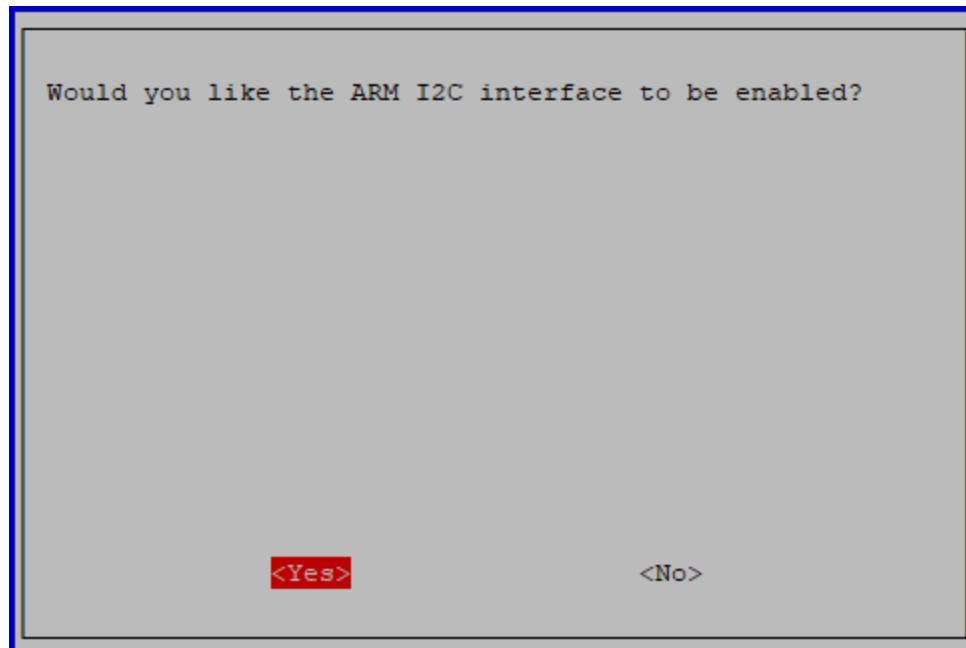
3 Interfacing options



P5 I2C



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



Step 2: Check whether the i2c modules are loaded and active.

```
lsmod | grep i2c
```

Then the following codes will appear (the number may be different).


```
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 smbus2 for I2C.

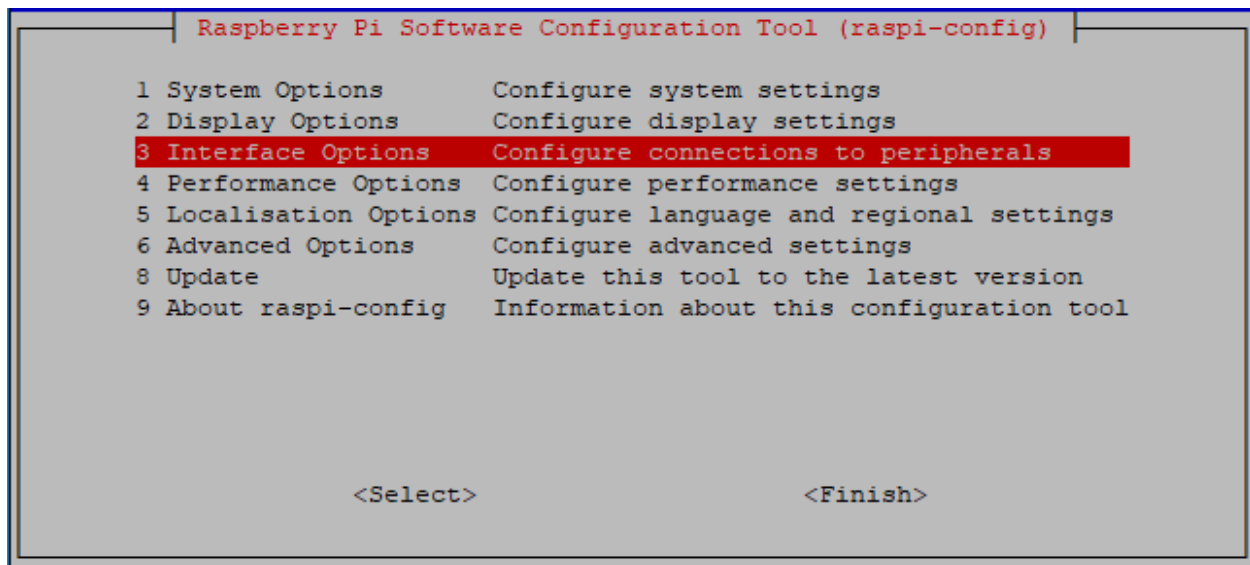
```
sudo pip3 install smbus2
```

7.2 SPI Configuration

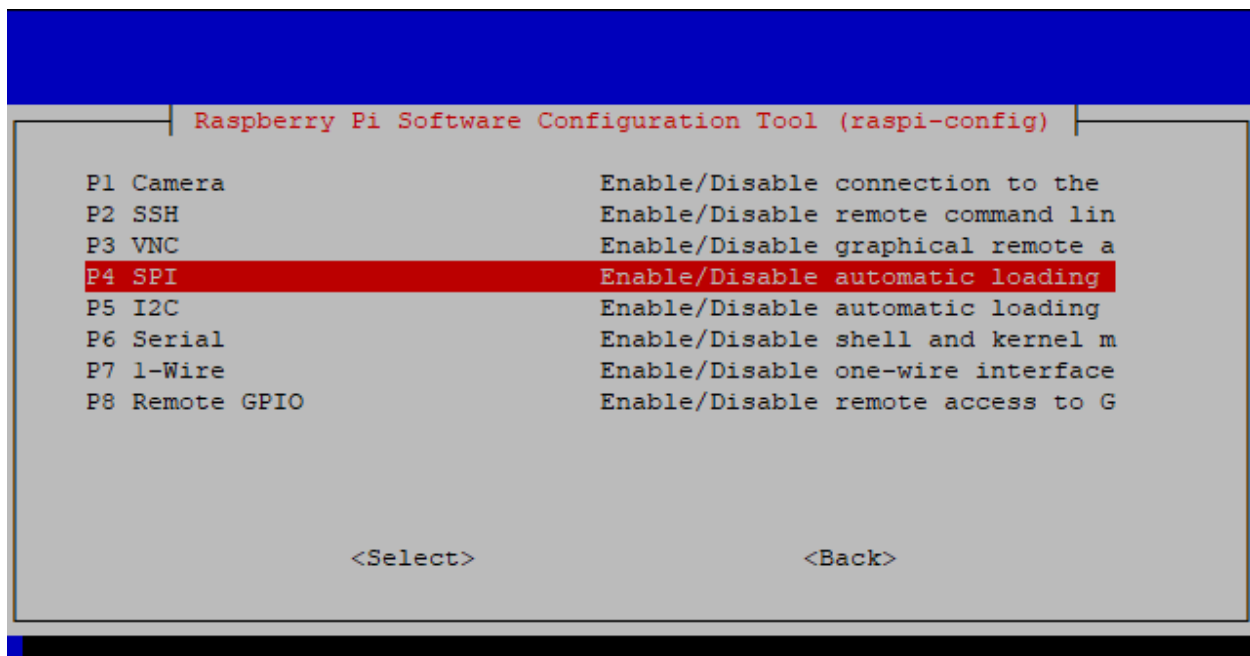
Step 1: Enable the SPI port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

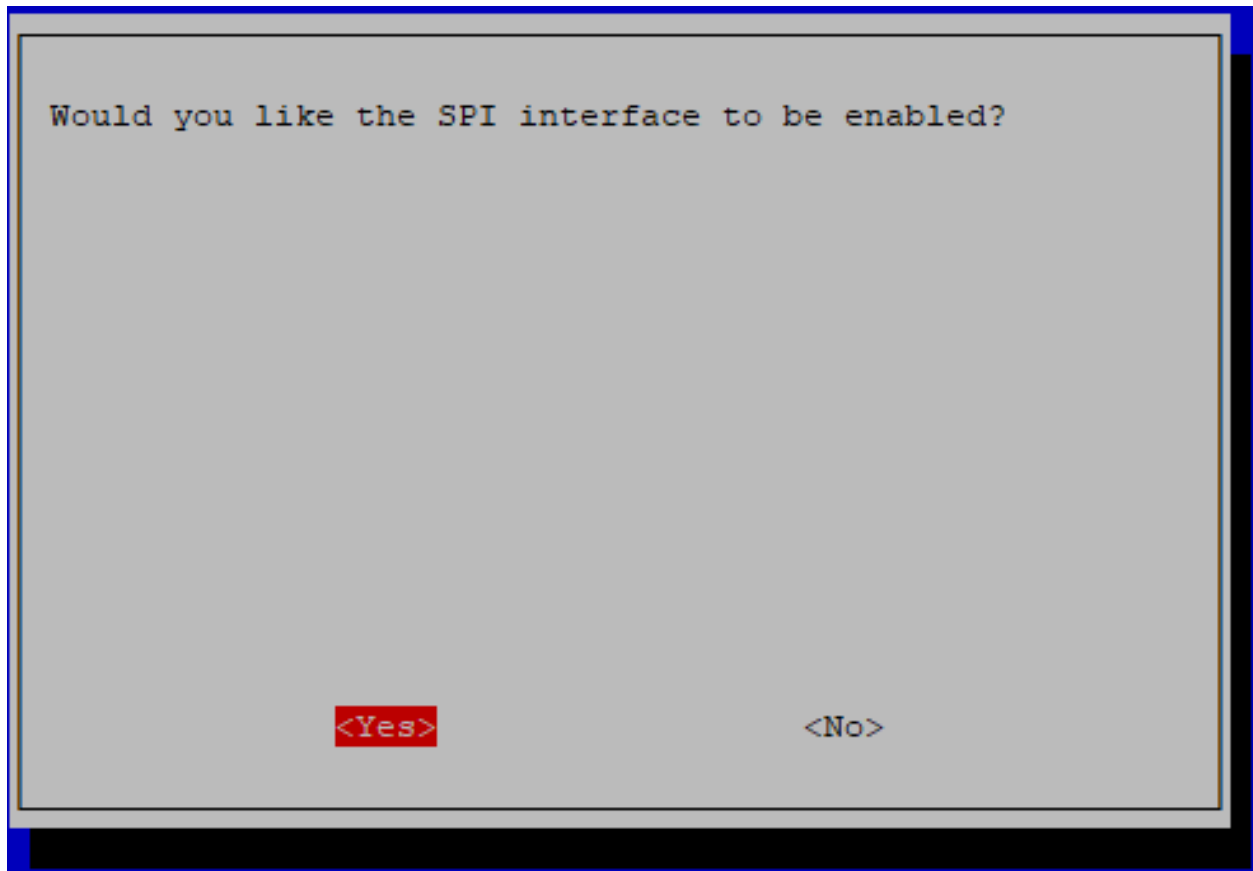
3 Interfacing options



P4 SPI



<YES>, then click <OK> and <Finish>. Now you can use the `sudo reboot` command to reboot the Raspberry Pi.



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.

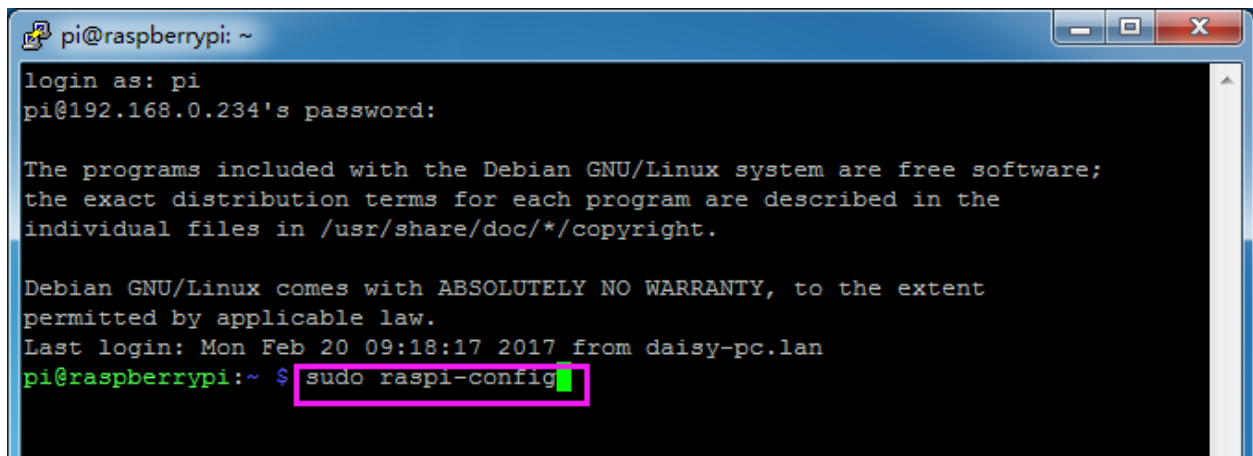
Enable VNC service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

Step 1

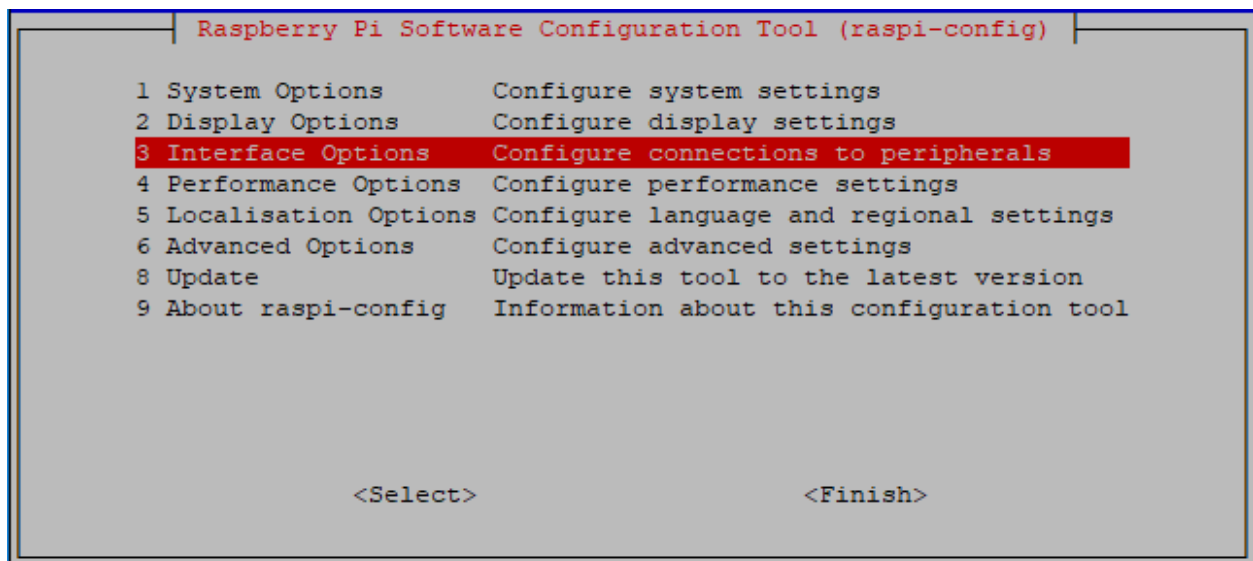
Input the following command:

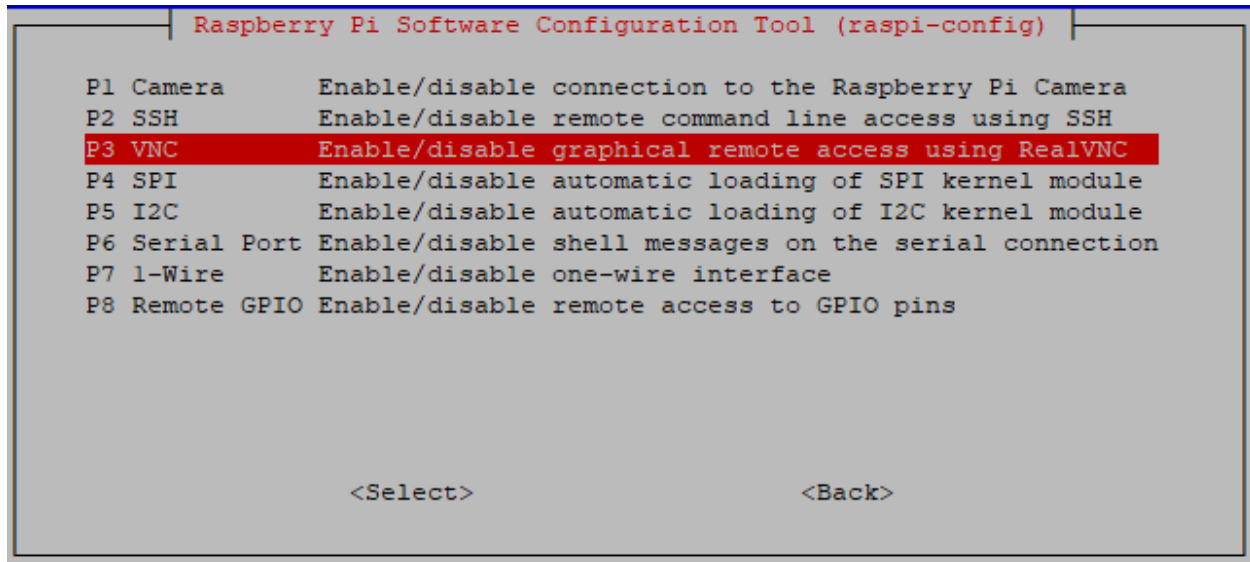
```
sudo raspi-config
```



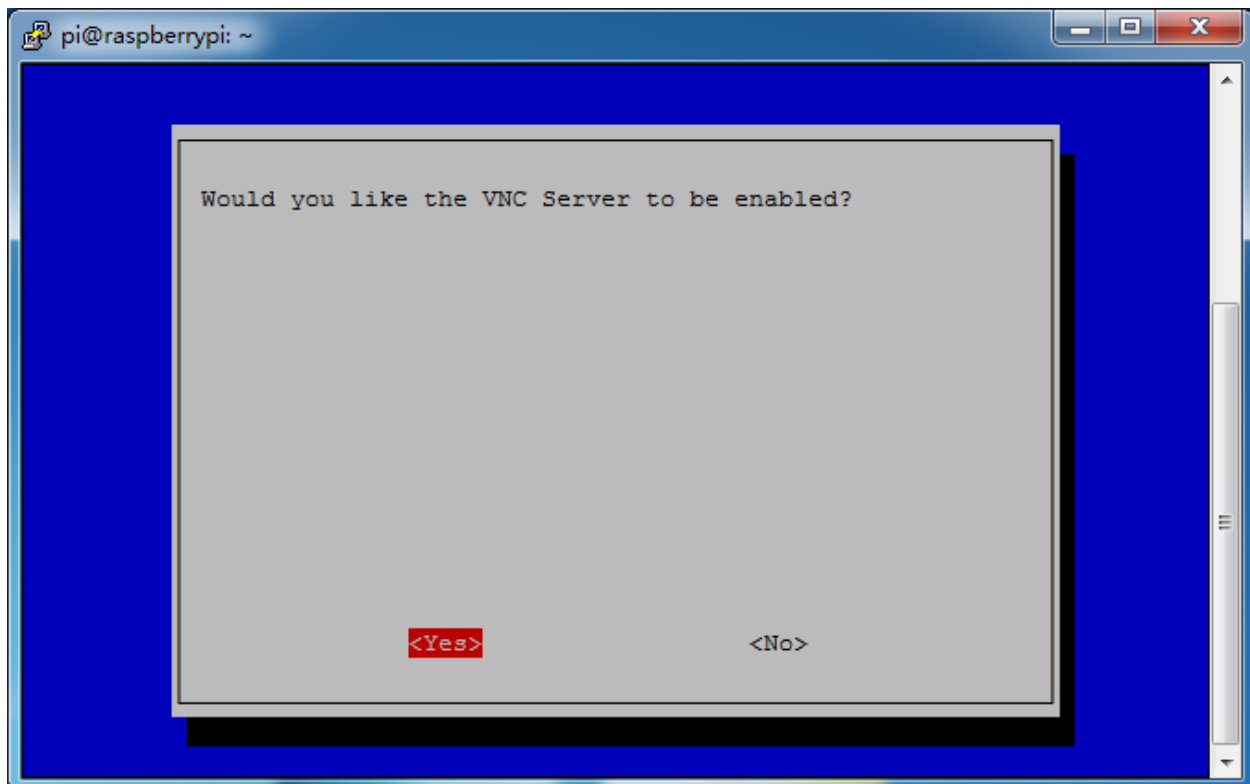
Step 2

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



Step 3**P3 VNC****Step 4**

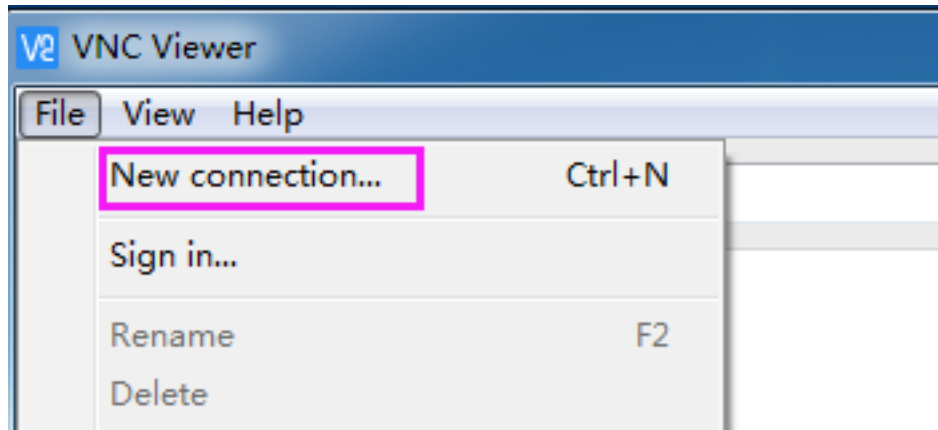
Select **Yes** -> **OK** -> **Finish** to exit the configuration.

**Login to VNC****Step 1**

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

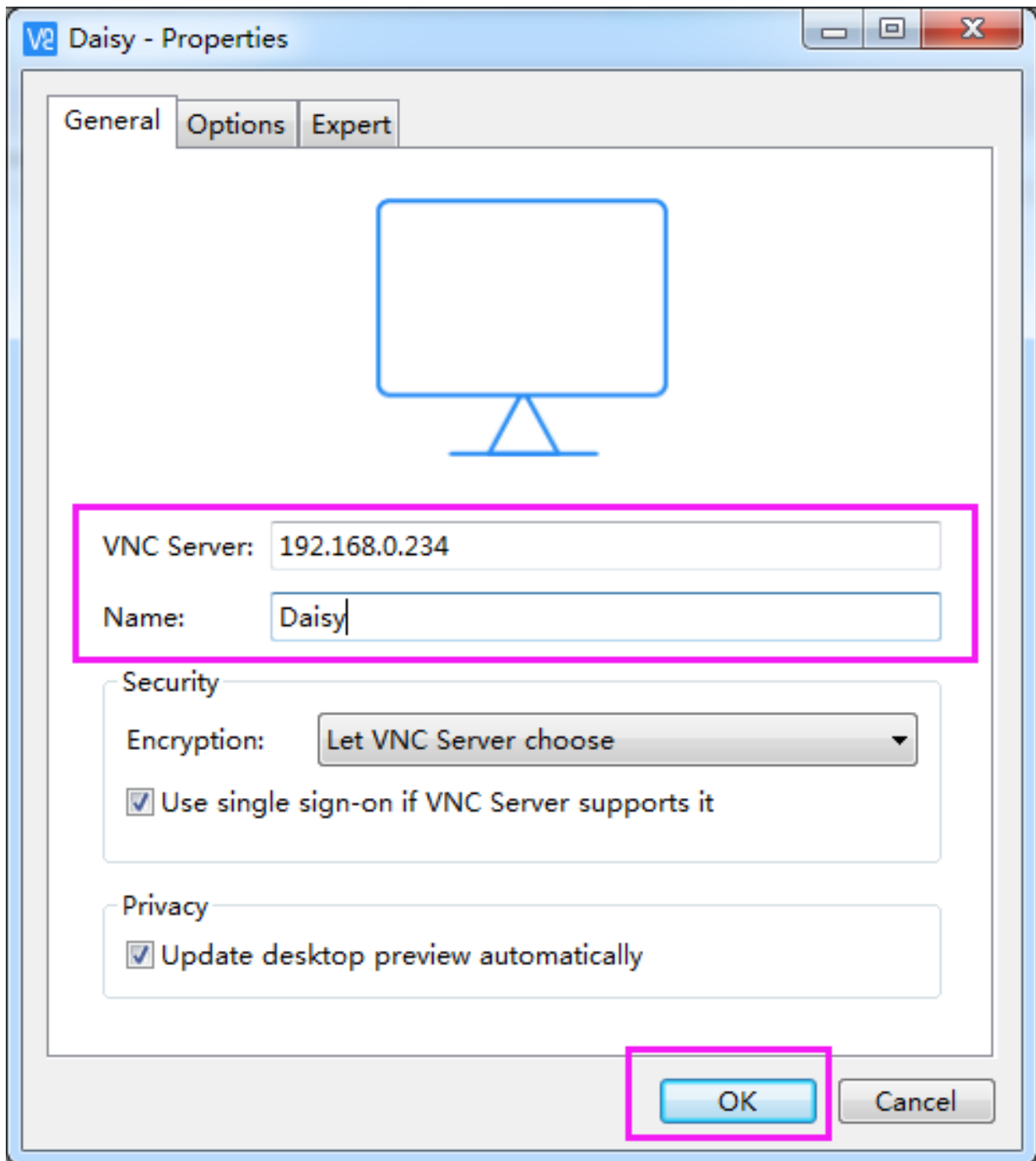
Step 2

Then select “New connection”.



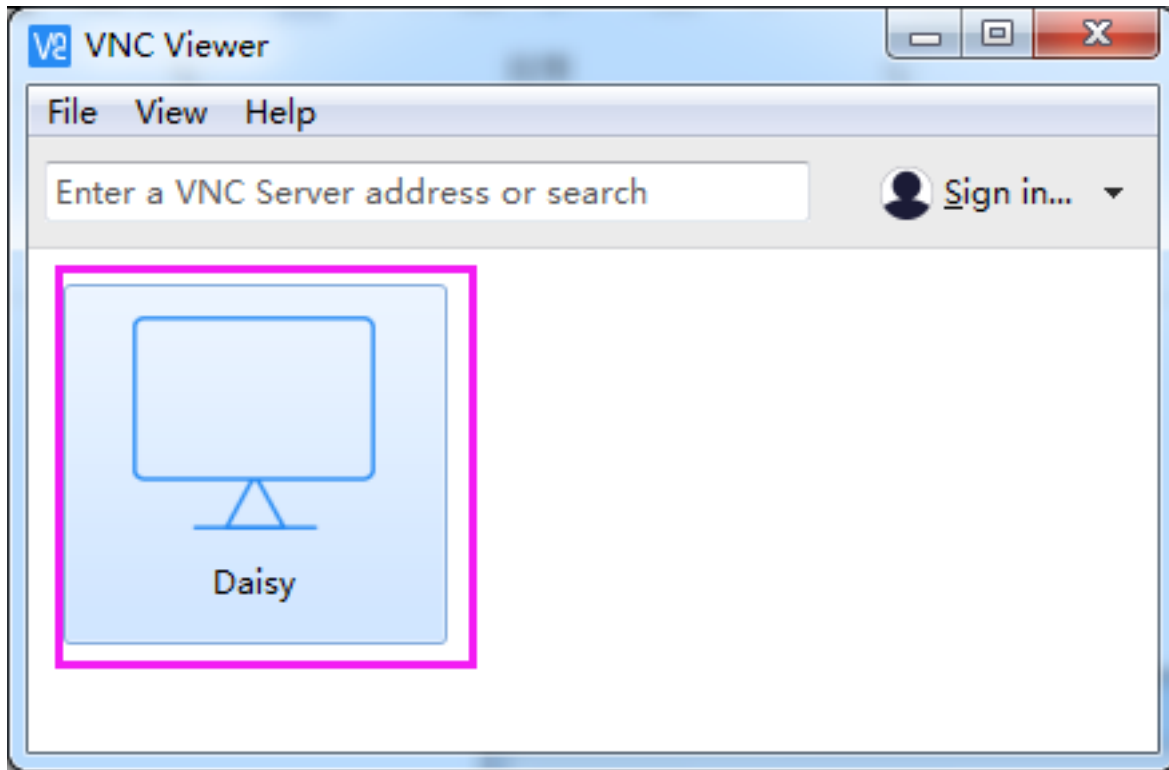
Step 3

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



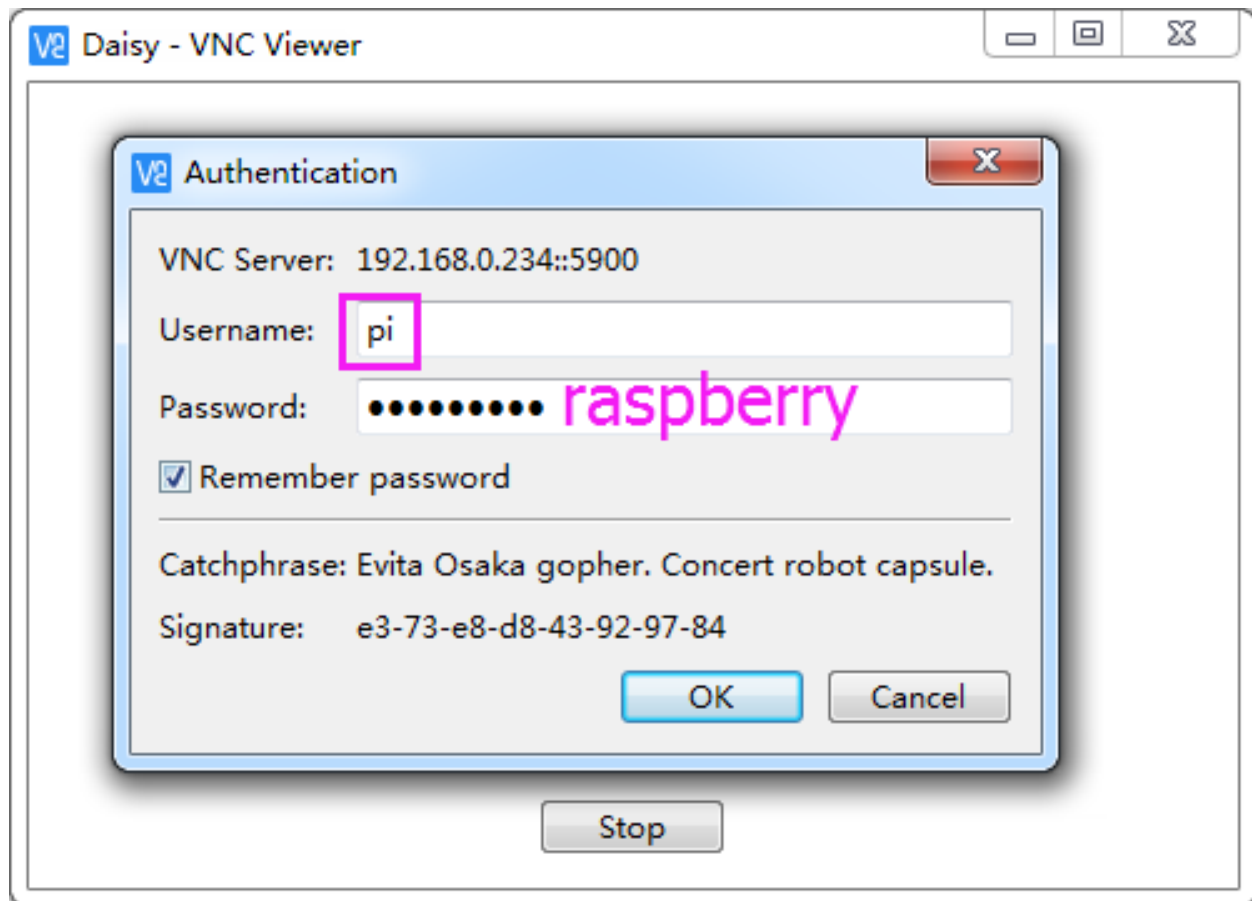
Step 4

Double click the **connection** just created:

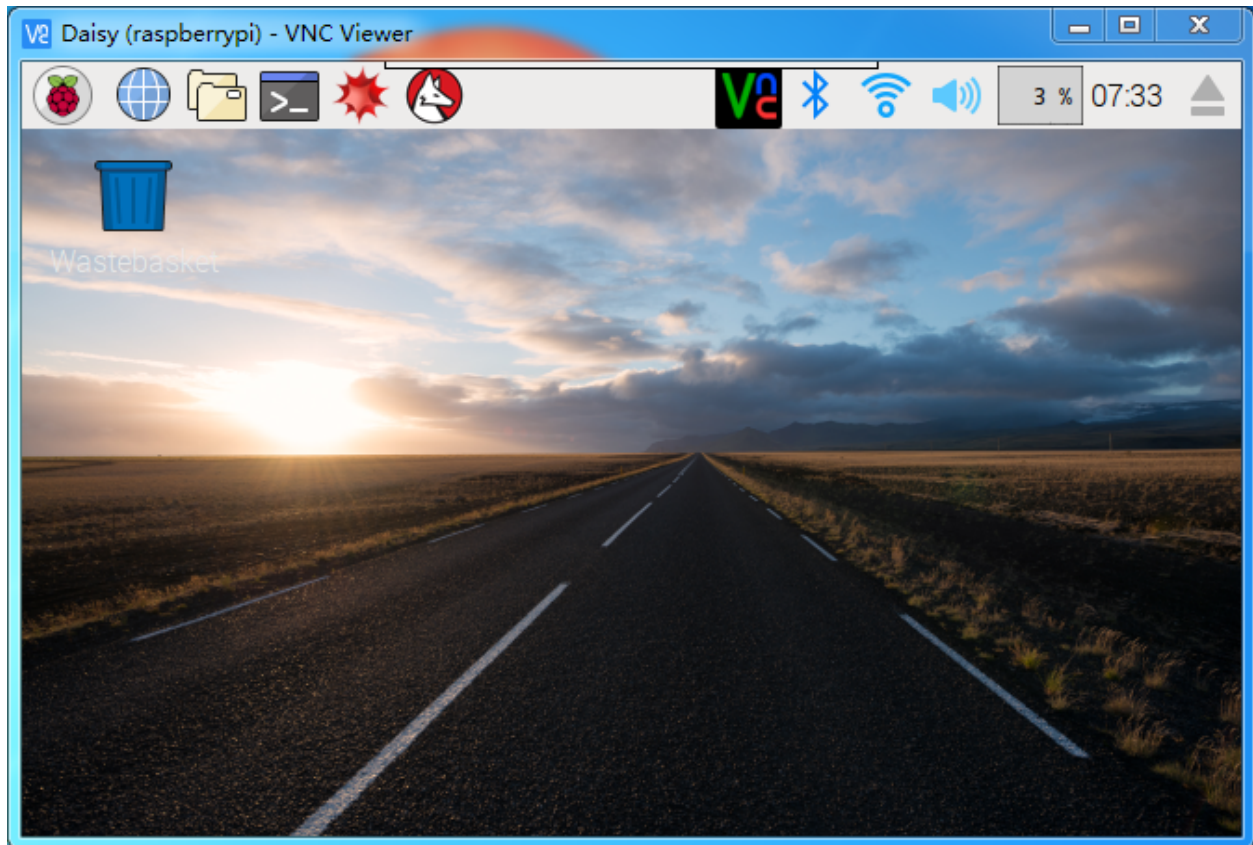


Step 5

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

**Step 6**

Now you can see the desktop of the Raspberry Pi:



That's the end of the VNC part.

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: ~
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.

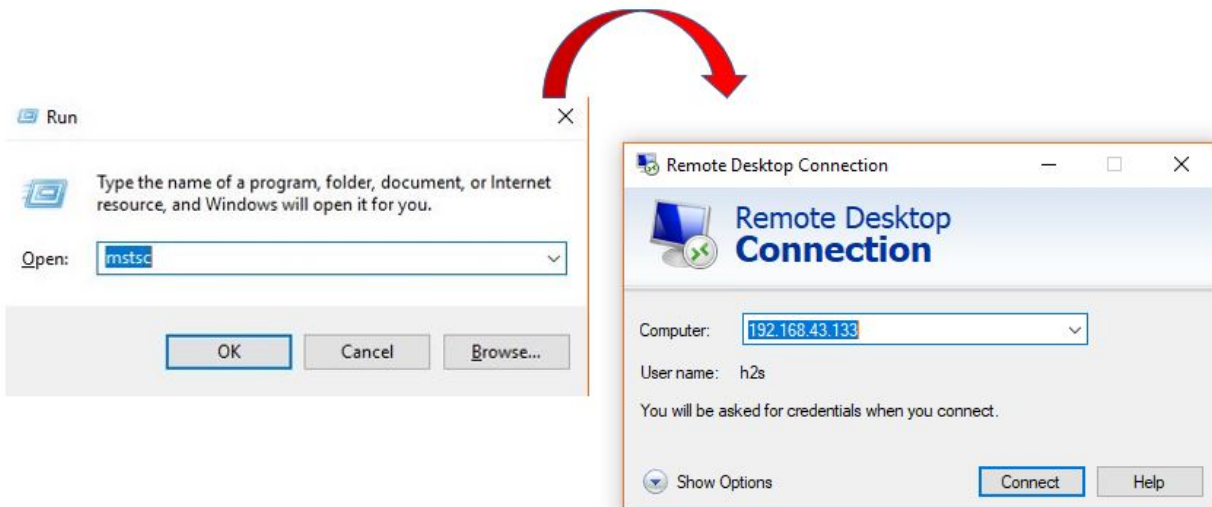
Login to XRDP

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.

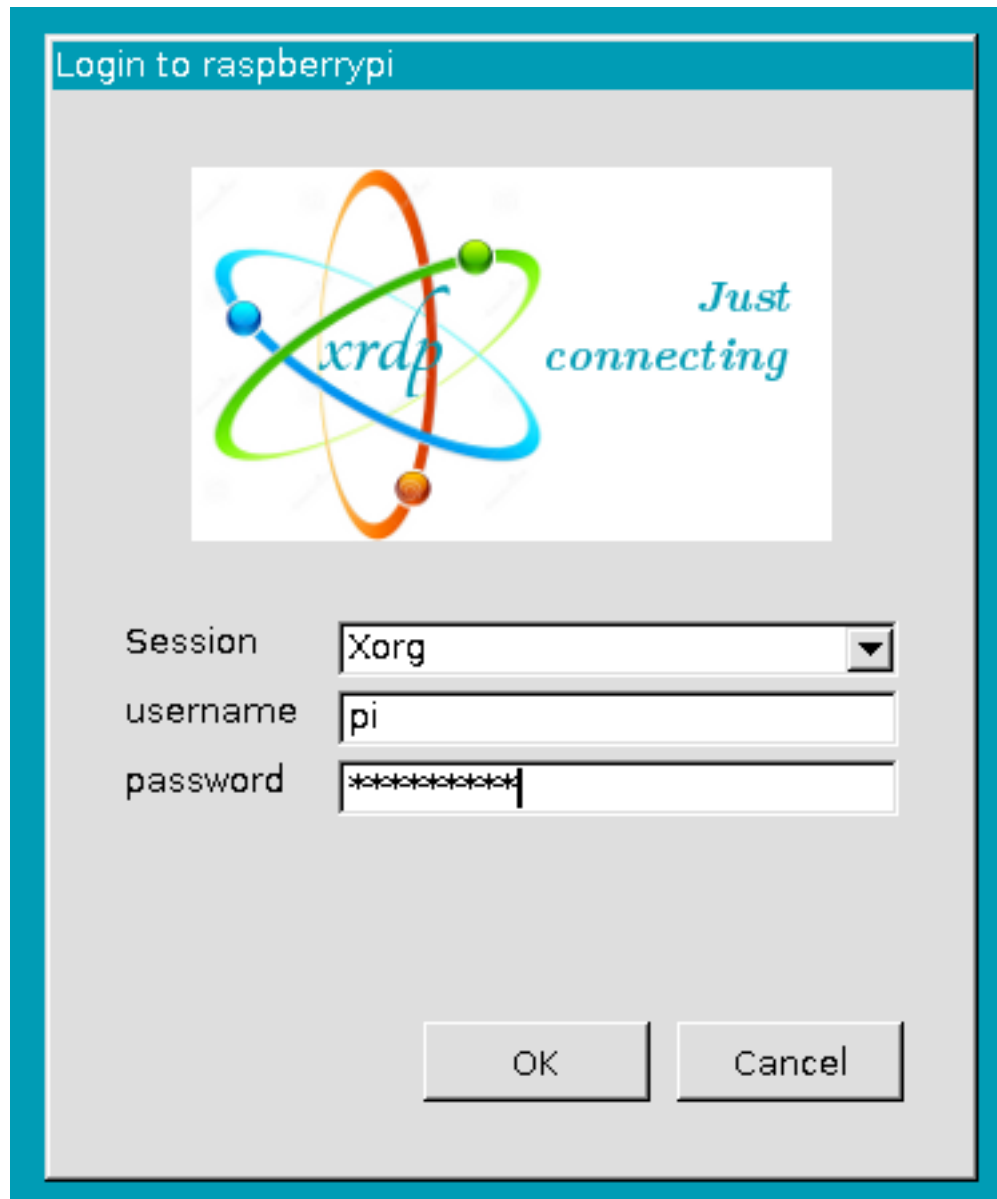
Step 2

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”.



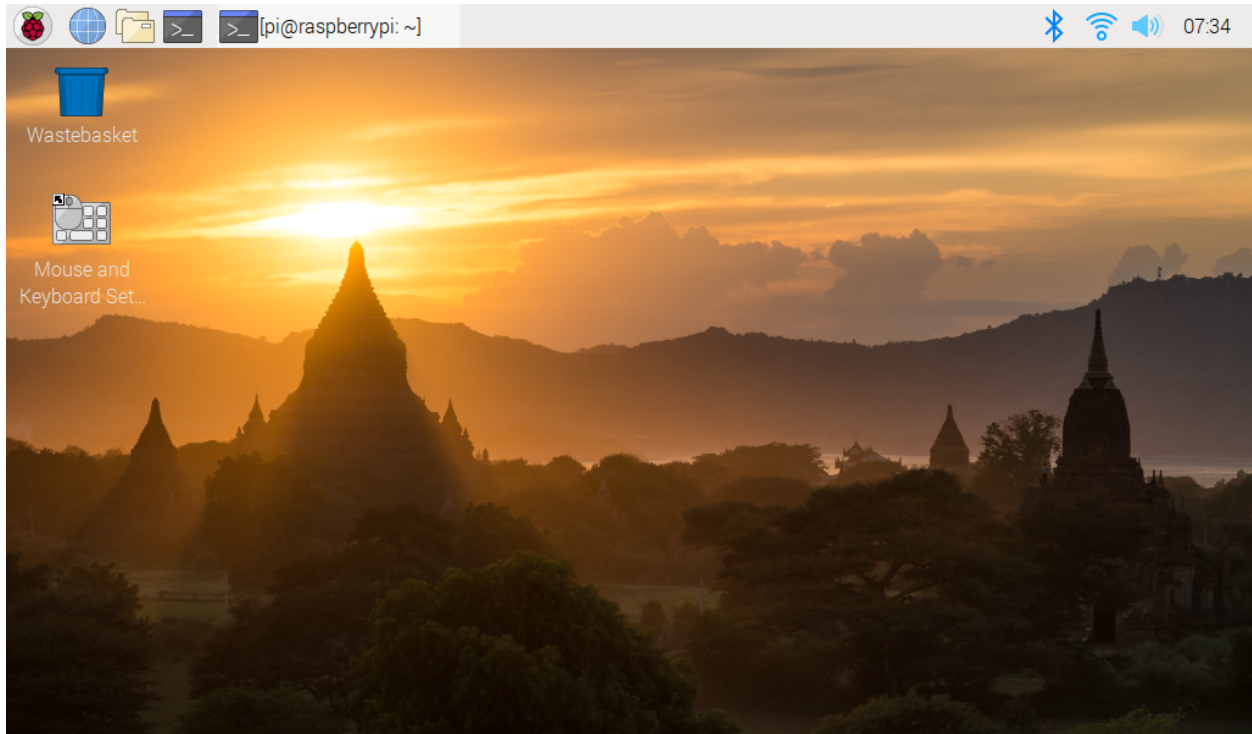
Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click “OK”. At the first time you log in, your username is “pi” and the password is “raspberry”.



Step 4

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



Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.

7.4 Components Introduction

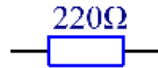
Resistor

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

The resistors in this kit are fixed ones. It is essential in the circuit to protect the connected components. Figure (a) below shows a 220 resistor. is the unit of resistance and the larger includes K, M, etc. Their relationship can be shown as follows: $1\text{ M}=1000\text{ K}$ $1\text{ K}=1000$, which means $1\text{ M}=1000,000=10^6$. Figure (b) and (c) show two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.



(a)



(b)



(c)

The resistance can be marked directly, in color code, and by character. The resistors offered in this kit are marked by different colors. Namely, the bands on the resistor indicate the resistance.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster. If you are not sure about the value, use the multimeter.

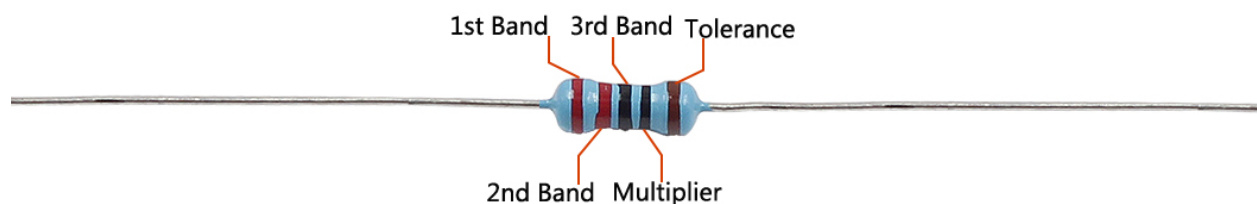
In the kit, a Resistor Color Code Calculator card is provided as shown below:

| Color | 1st Band | 2nd Band | 3rd Band | Multiplier | Tolerance |
|--------|----------|----------|----------|------------|-------------|
| Black | 0 | 0 | 0 | 1Ω | |
| Brown | 1 | 1 | 1 | 10Ω | ± 1% (F) |
| Red | 2 | 2 | 2 | 100Ω | ± 2% (G) |
| Orange | 3 | 3 | 3 | 1KΩ | |
| Yellow | 4 | 4 | 4 | 10KΩ | |
| Green | 5 | 5 | 5 | 100KΩ | ± 0.5% (D) |
| Blue | 6 | 6 | 6 | 1MΩ | ± 0.25% (C) |
| Violet | 7 | 7 | 7 | 10MΩ | ± 0.10% (B) |
| Grey | 8 | 8 | 8 | | ± 0.05% |
| White | 9 | 9 | 9 | | |
| Gold | | | | 0.1 | ± 5% (J) |
| Silver | | | | 0.01 | ± 10% (K) |

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

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

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands. Let's see how to read the resistance value of a 5-band resistor as shown below. Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4th and 5th band will be comparatively larger. Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.



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

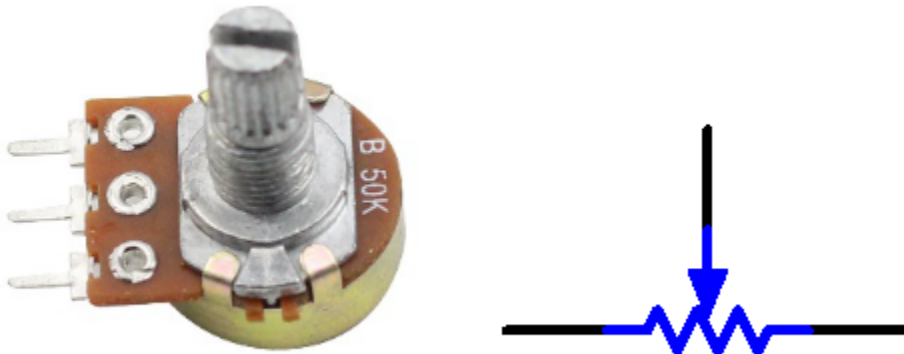
One more example. The resistance of the resistor below should be 1(brown) 0(black) 0(black) $\times 10^1(\text{brown}) = 100 \times 10 = 1000 = 1\text{K}$, and the permissible error is $\pm 1\%$ (brown). Now try it by yourself!



You can also use a multimeter to measure the resistance value of these resistors to double check whether you've read it correctly or not.

Potentiometer

Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation. Potentiometer usually consists of resistor and movable brush. When the brush is moving along the resistor, there is a certain resistance or voltage output depending on the displacement. Figure left is the potentiometer and figure right is the corresponding circuit symbol. The middle pin in left figure, represented by the arrow in right figure is the movable brush.



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

1. Serving as a voltage divider

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

2. Serving as a rheostat

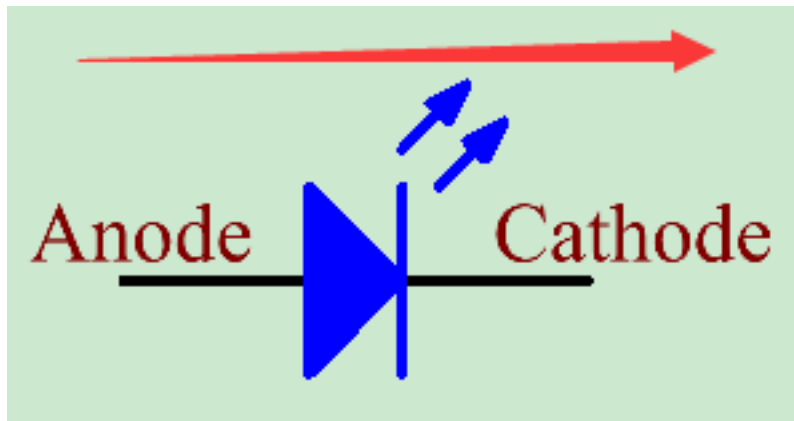
3. When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

4. Serving as a current controller

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

LED

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



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

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

The formula of the resistance value is as follows:

$$R = (V_{\text{supply}} - V_D) / I$$

R stands for the resistance value of the current limiting resistor, V_{supply} for voltage supply, V_D for voltage drop and I for the working current of the LED.

If we provide 5 voltage for the red LED, the minimum resistance of the current limiting resistor should be: $(5V - 1.8V) / 20\text{mA} = 160$. Therefore, you need a 160 or larger resistor to protect the LED. You are recommended to use the 220 resistor offered in the kit.

RGB LED

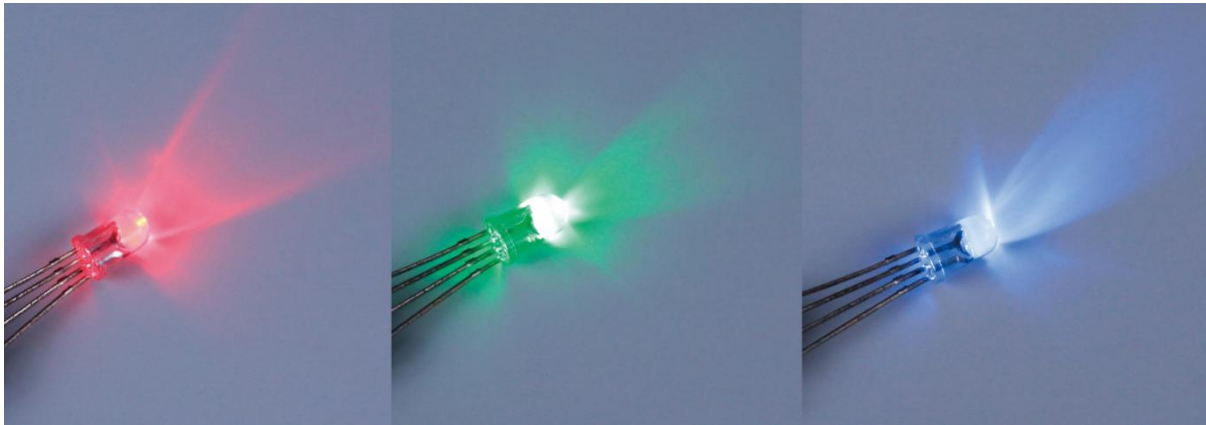
An RGB LED is provided in this kit. RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to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

RGB LEDs can be categorized into common anode and common cathode ones. In this experiment, the latter is used. The common cathode, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and

plug in the three pins, the LED will flash the corresponding color.



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



Or you can distinguish them in another way. As GND is the longest one and can be defined directly, you just need to confirm the other three pins. You can test it by giving them a small voltage. The forward voltage drop from the three pins to the GND are respectively 1.8V (red), 2.5V (blue), and 2.3V (green). Thus, when you connect the same current limiting resistor with the three pins and supply them with the same voltage, the red one is the brightest, and then comes the green and the blue one. Therefore, you may need to add a current limiting resistor with different resistances to the three pins for these colors.

Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Among others, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jumper wires are fitted by inserting their “end connectors” into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The “end connectors” are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female. The reason we call it Male-to-Female is because it has the outstanding tip in one end as well as a sunk female end. Male-to-Male means both side are male and Female-to-Female means both ends are female.



Male-to-Female



Male-to-Male



Female-to-Female

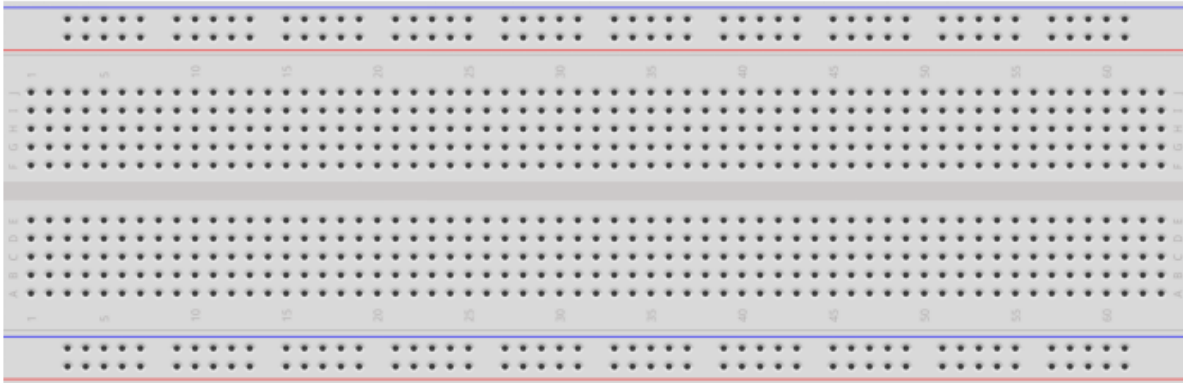
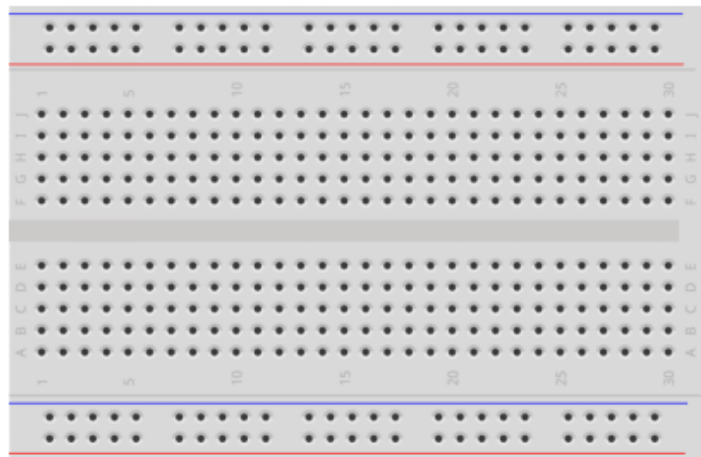
Male-to-Female Male-to-Male Female-to-Female

More than one type of them may be used in a project. The colors of the jumper wires are different but it doesn't mean their functions are different accordingly; it's just designed so to better identify the connection between each circuit. The Male-to-Male and Male-to-Female jumper wires are included in the kit. But actually only some Male-to-Male ones will be used in the experiments. You can use the Male-to-Female wires in other experiments.

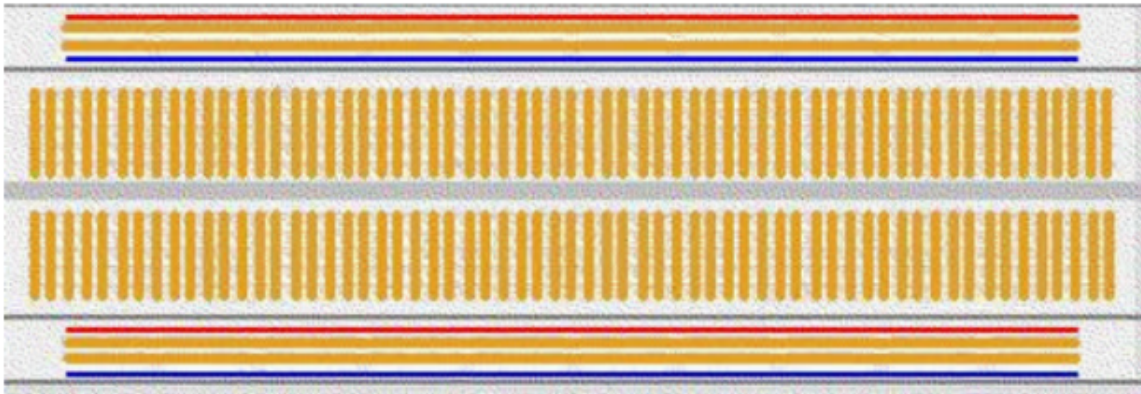
Breadboard

A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finalizing any circuit design. And it has many holes into which components like ICs and resistors as well as jumper wires mentioned above can be inserted. The breadboard allows you to easily plug in and remove components. If there is going to be many changes or if you just want to make a circuit quickly, it will be much quicker than soldering up your circuit. Therefore, in lots of experiments, it is often used as a hub to connect two or more devices.

Normally, there are two types of breadboard: full+ and half+. You can tell their difference from the names. A half+ breadboard is half the size of a full+ one and their functions are the same. Here take the full+ breadboard.

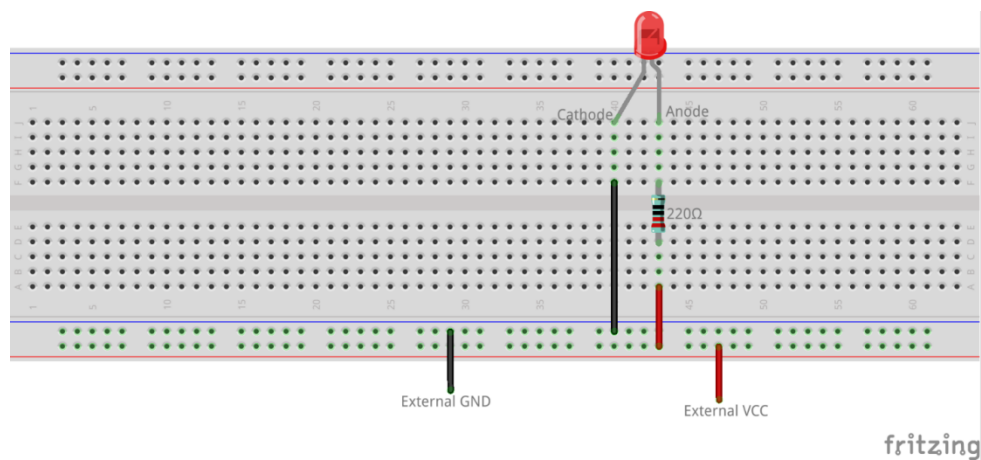
**Full+****Half+**

This is the internal structure of a full+ breadboard. Although there are holes on the breadboard, internally some of them are connected with metal strips. Those holes are to insert pins of devices or wires. As shown in the fig. (t) below, there are four long metal strips on the long sides; the blue and red lines are marked just for clear observation. But you can take the blue line as the GND and red one as VCC for convenience. Every five holes in the middle are vertically connected with metal strips internally which don't connect with each other. You can connect them horizontally with wires or components. A groove is made in the middle on the breadboard for IC chips.



Internal structure of the full+

Now let's make some simple experiment with the breadboard. Turn on an LED as shown in the figure below. You can have a try and the LED will light up. The breadboard makes it possible for you to plug and pull components at any time without welding, which is very convenient for tests.



Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.

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.

THANK YOU

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

COPYRIGHT NOTICE

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.