Getting Started

Thank you for choosing Freenove products!

After you download the ZIP file we provide. Unzip it and you will get a folder contains several files and folders. There are three PDF files:

- **Tutorial.pdf**

  It contains basic operations such as installing system for Raspberry Pi.

  The code in this PDF is in C and Python.
- **Tutorial_GPIOZero.pdf**

  It contains basic operations such as installing system for Raspberry Pi.

  The code in this PDF is in Python.
- **Processing.pdf** in Freenove_RFID_Starter_Kit_for_Raspberry_Pi\Processing

  The code in this PDF is in Java.

**We recommend you to start with Tutorial.pdf first.**

If you want to start with Processing.pdf or skip some chapters of Tutorial.pdf, you need to finish necessary steps in **Chapter 7 AD/DA** of **Tutorial.pdf** first.
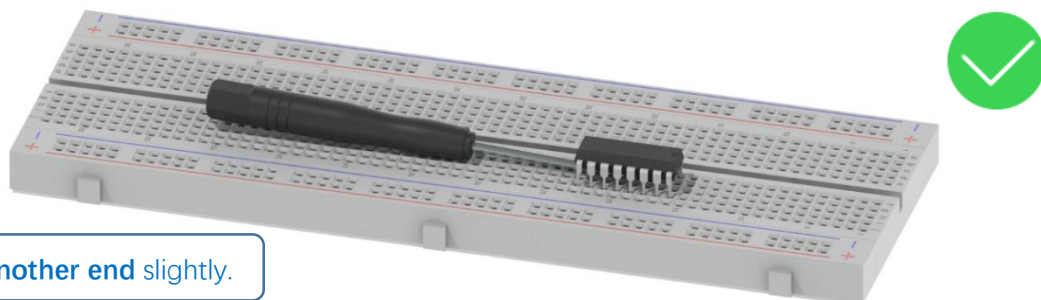
Remove the Chips

Some chips and modules are inserted into the breadboard to protect their pins.

You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
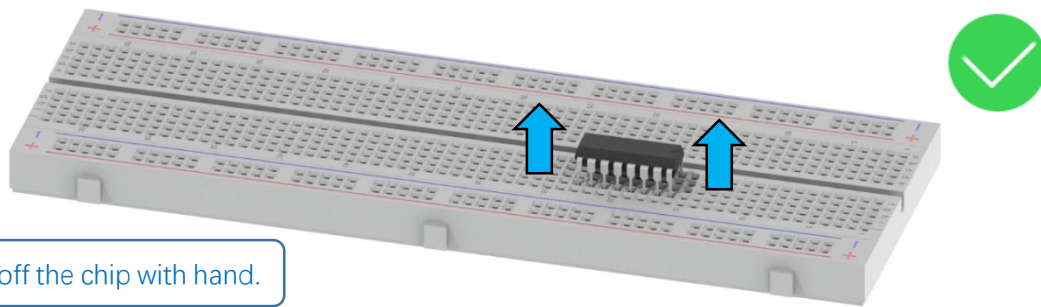
Please find a tool (like a little screw driver) to handle them like below:
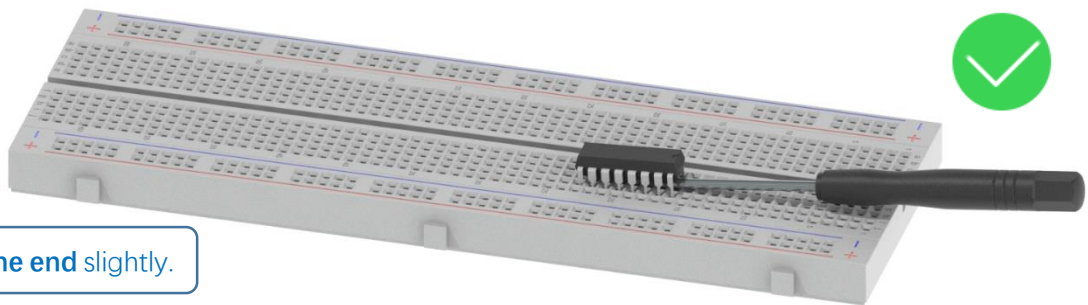
Step 1, lift **one end** slightly.

Step 2, lift **another end** slightly.

Step 3, take off the chip with hand.

Avoid lifting one end with big angle directly.

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

● Product quality issues
● Product use and build issues
● Questions regarding the technology employed in our products for learning and education
● Your input and opinions are always welcome
● We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

● Keep this product out of the reach of children under 6 years old.
● This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
● This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
● When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
● It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it

III
www.freenove.com
support@freenove.com

cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

Copyright

Contents

Preface

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 an incredibly capable little device that enables people of all ages to explore computing, and to learn how to program in a variety of computer languages like Scratch and Python. It is capable of doing everything you would expect from a desktop computer, such as browsing the internet, playing high-definition video content, creating spreadsheets, performing word-processing, and playing video games. For more information, you can refer to Raspberry Pi official website. For clarification, this tutorial will also reference Raspberry Pi as RPi, RPI and RasPi.

In this tutorial, most chapters consist of **Components List**, **Component Knowledge**, **Circuit**, and **Code** (**Python** code). We provide Python code for each project in this tutorial. After completing this tutorial, you can learn Java by reading Processing.pdf.

This kit does not contain **Raspberry and its accessories**. You can also use the components and modules in this kit to create projects of your own design.

Additionally, if you encounter any issues or have questions about this tutorial or the contents of kit, you can always contact us for free technical support at:

support@freenove.com

Raspberry Pi

So far, at this writing, Raspberry Pi has advanced to its fourth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility of peripheral devices greatly enhanced between versions.

Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

Actual image of Raspberry Pi 4 Model B:	CAD image of Raspberry Pi 4 Model B:
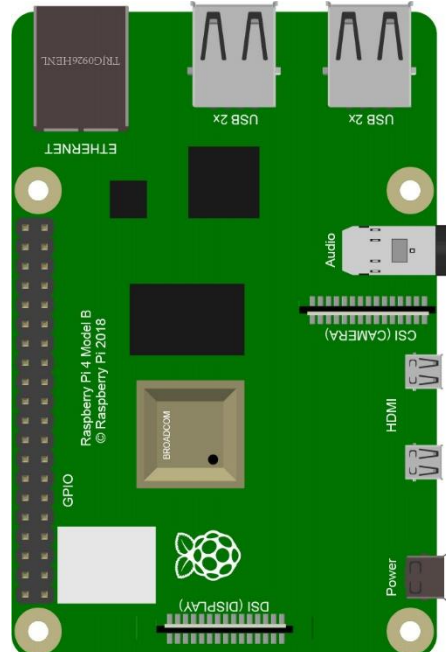
Actual image of Raspberry Pi 3 Model B+:	CAD image of Raspberry Pi 3 Model B+:
Actual image of Raspberry Pi 3 Model B:	CAD image of Raspberry Pi 3 Model B:
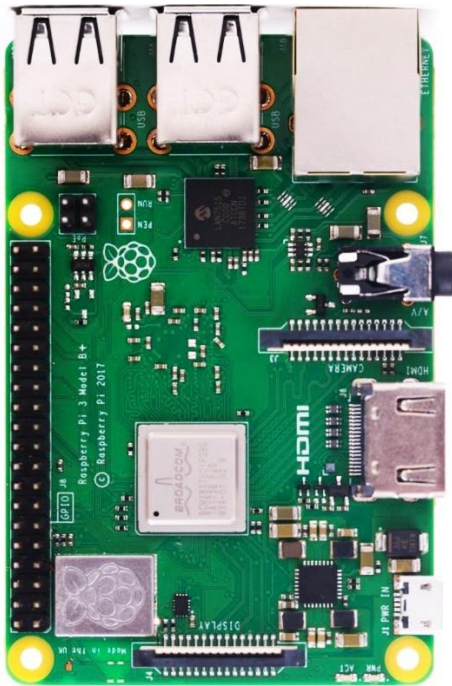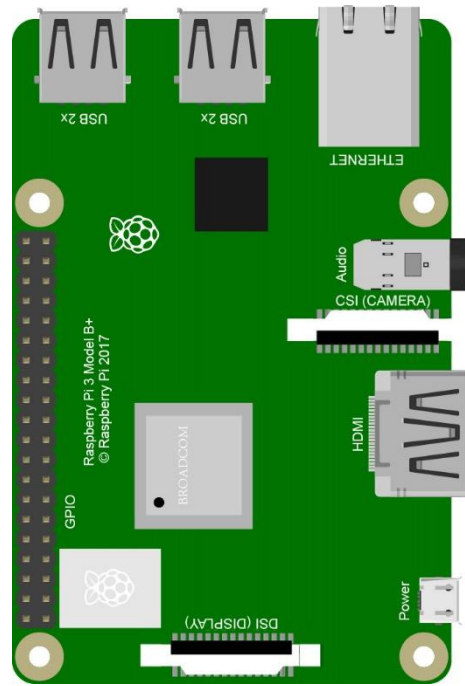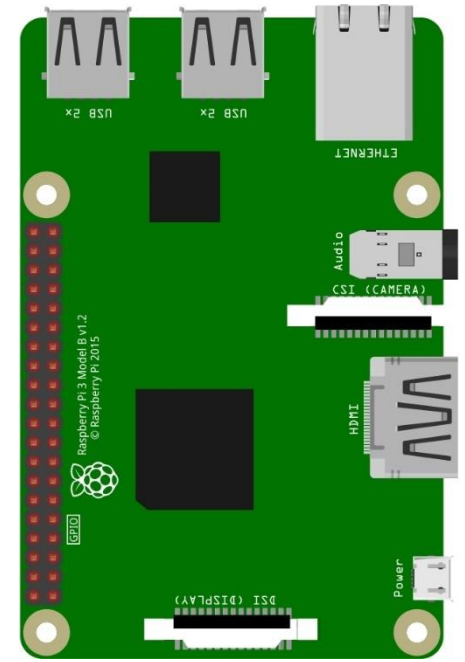
Actual image of Raspberry Pi 2 Model B:	CAD image of Raspberry Pi 2 Model B:
Actual image of Raspberry Pi 1 Model B+:	CAD image of Raspberry Pi 1 Model B+:
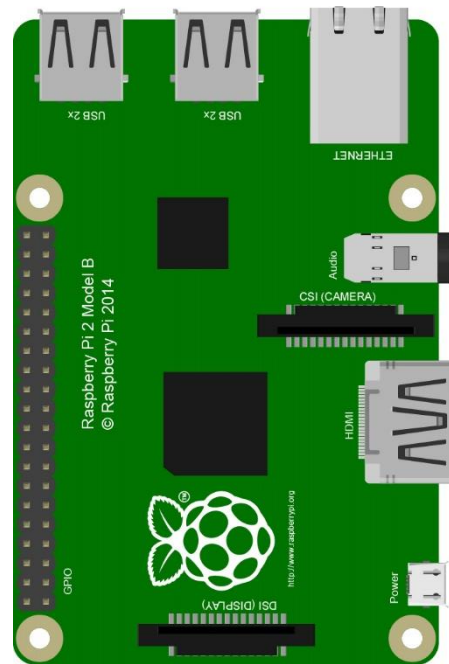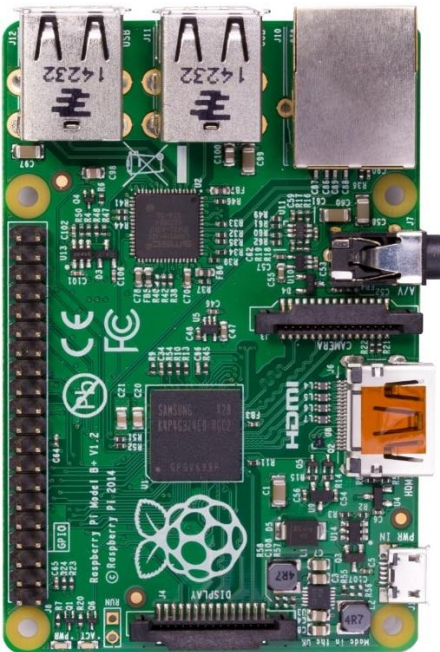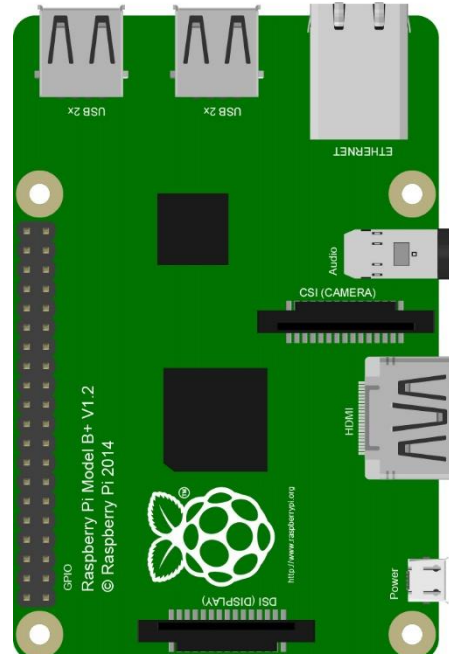
Actual image of Raspberry Pi 3 Model A+:	CAD image of Raspberry Pi 3 Model A+:
Actual image of Raspberry Pi 1 Model A+:	CAD image of Raspberry Pi 1 Model A+:
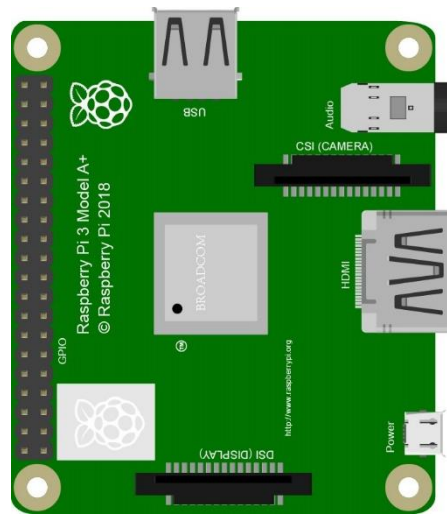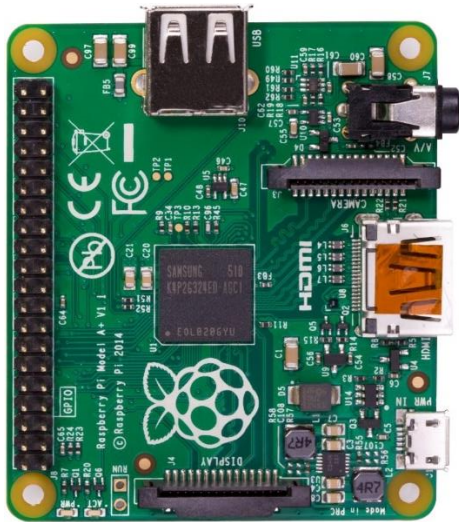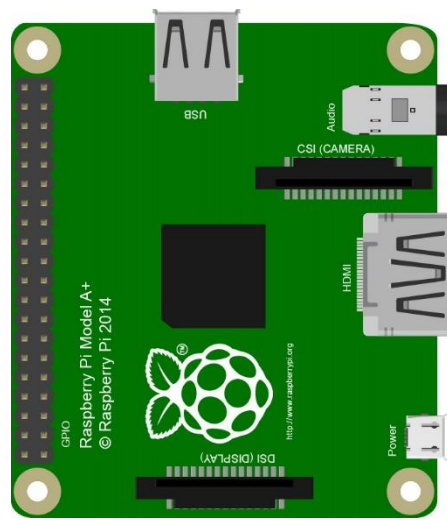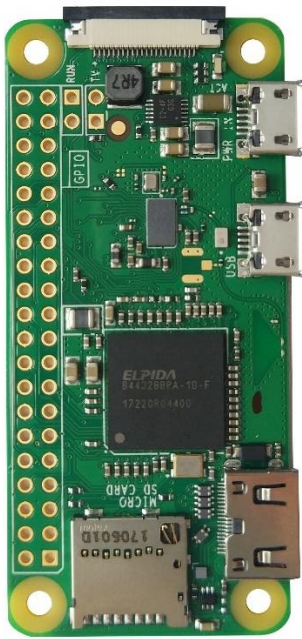
Actual image of Raspberry Pi Zero W:	CAD image of Raspberry Pi Zero W:
Actual image of Raspberry Pi Zero:	CAD image of Raspberry Pi Zero:
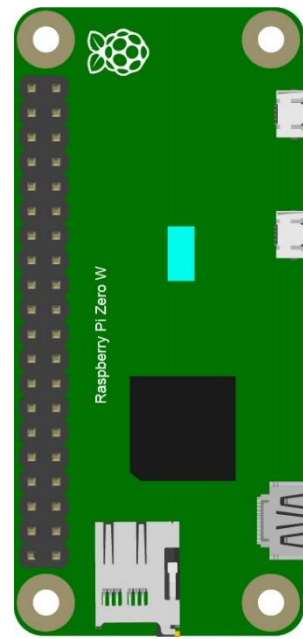
Hardware interface diagram of RPi 4B:

GPIO
Connector

Display
Connector

Power
Connector

Ethernet
Connector

USB
Connector x4

Micro HDMI
Connector x2

Camera
Connector

Audio
Connector
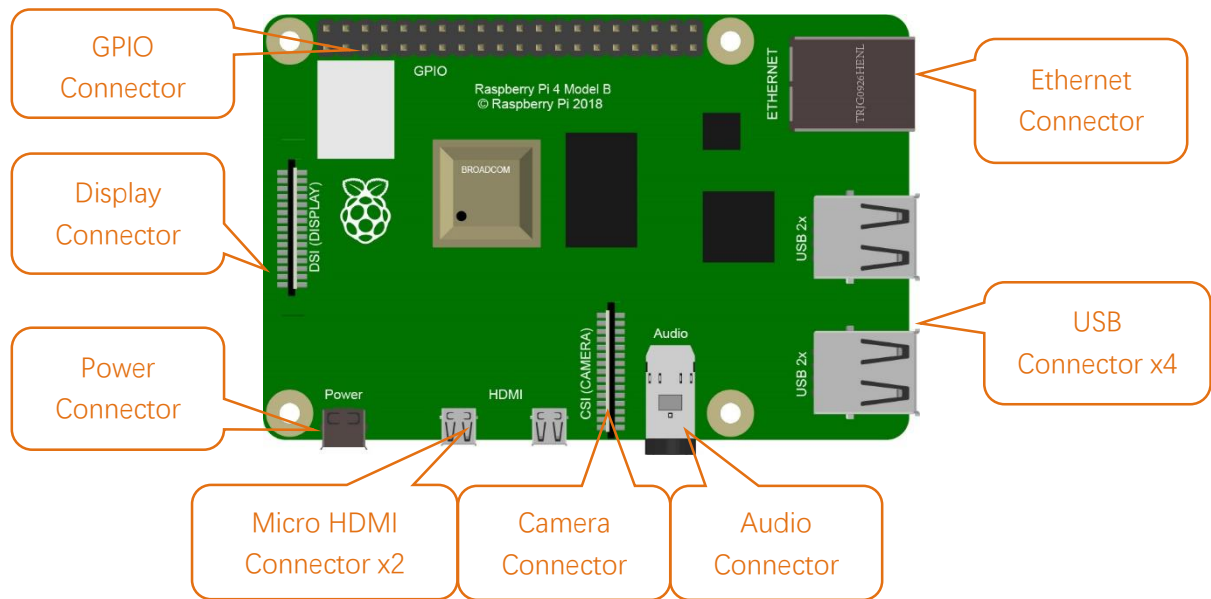
Hardware interface diagram of RPi 3B+/3B/2B/1B+:

GPIO
Connector

Display
Connector

Power
Connector

USB
Connector

Ethernet
Connector

HDMI
Connector

Camera
Connector

Audio
Connector

Hardware interface diagram of RPi 3A+/A+:

GPIO
Connector

USB
Connector

Display
Connector

Power
Connector

HDMI
Connector

Camera
Connector

Audio
Connector

Raspberry Pi Model A+
© Raspberry Pi 2014

GPIO

DSI (DISPLAY)

http://www.raspberrypi.org

Power

HDMI

CSI (CAMERA)

Audio

USB
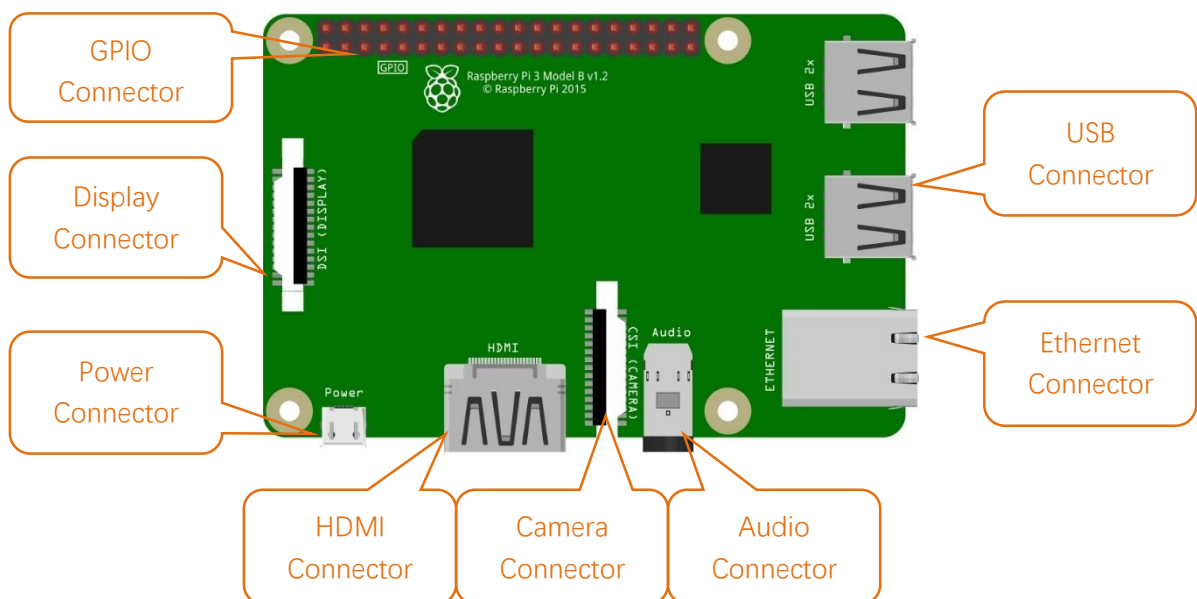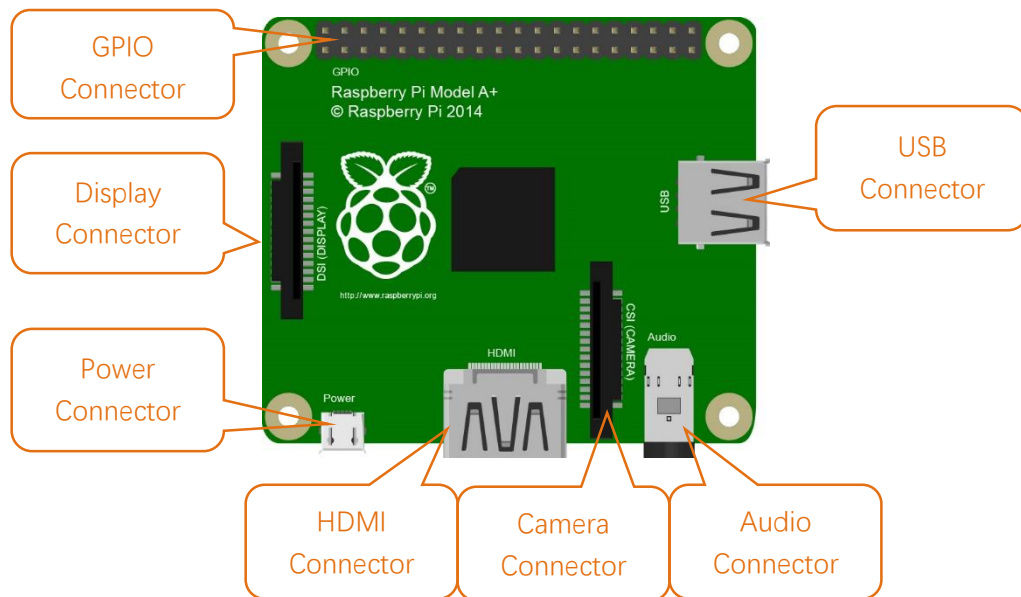
Hardware interface diagram of RPi Zero/Zero W:

GPIO
Connector

Camera
Connector

Raspberry Pi Zero W

HDMI
Connector

USB
Connector

Power
Connector
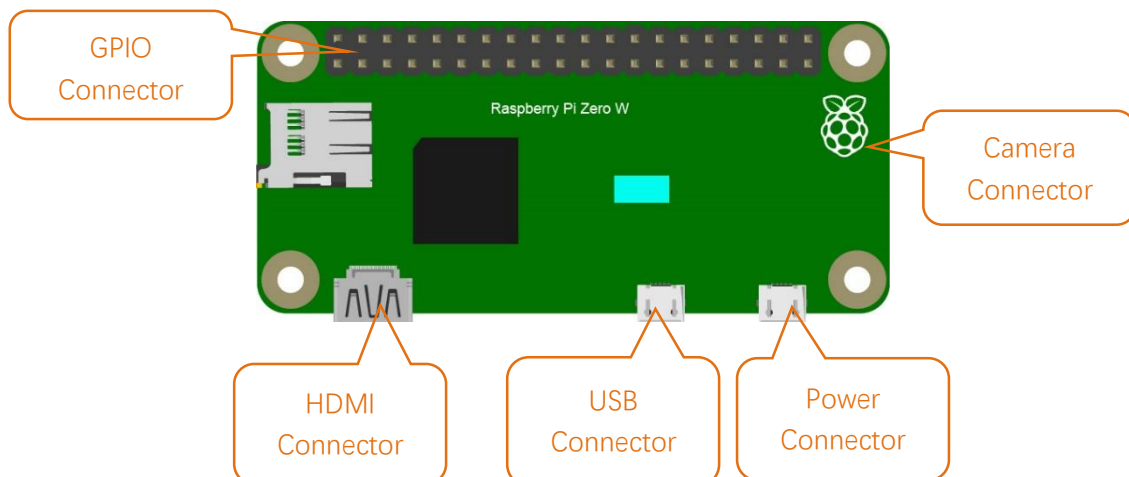
Installing an Operating System

The first step is to install an operating system on your RPi so that it can be programmed and function. If you have installed a system in your RPi, you can start from Chapter 0 Preparation.

Component List

Required Components

Any Raspberry Pi with 40 GPIO	5V/3A Power Adapter. Note: Different versions of Raspberry Pi have different power requirements (please check the power requirements for yours on the chart in the following page.)
Micro or Type-C USB Cable x1	Micro SD Card (TF Card) x1, Card Reader x1
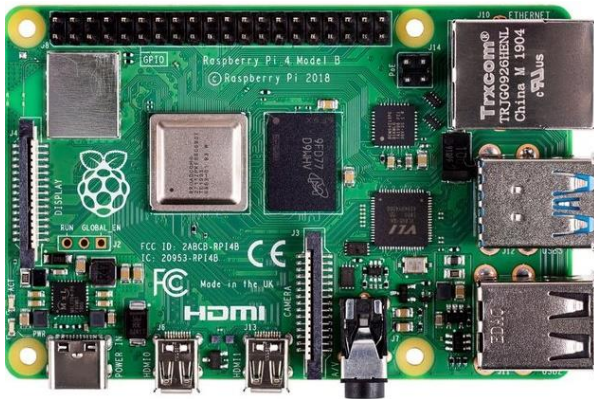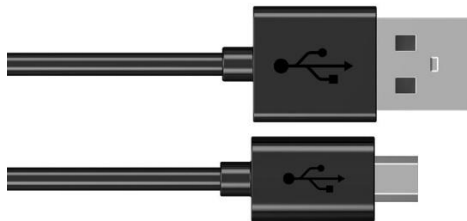
Power requirements of various versions of Raspberry Pi are shown in following table:

Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi Model A	700mA	500mA	200mA
Raspberry Pi Model B	1.2A	500mA	500mA
Raspberry Pi Model A+	700mA	500mA	180mA
Raspberry Pi Model B+	1.8A	600mA/1.2A (switchable)	330mA
Raspberry Pi 2 Model B	1.8A	600mA/1.2A (switchable)	350mA
Raspberry Pi 3 Model B	2.5A	1.2A	400mA
Raspberry Pi 3 Model A+	2.5A	Limited by PSU, board, and connector ratings only.	350mA
Raspberry Pi 3 Model B+	2.5A	1.2A	500mA
Raspberry Pi 4 Model B	3.0A	1.2A	600mA
Raspberry Pi Zero W	1.2A	Limited by PSU, board, and connector ratings only.	150mA
Raspberry Pi Zero	1.2A	Limited by PSU, board, and connector ratings only	100mA

For more details, please refer to https://www.raspberrypi.org/help/faqs/#powerReqs

In addition, RPi also needs an Ethernet network cable used to connect it to a WAN (Wide Area Network).

All these components are necessary for any of your projects to work. Among them, the power supply of at least 5V/2.5A, because a lack of a sufficient power supply may lead to many functional issues and even damage your RPi, we STRONGLY RECOMMEND a 5V/2.5A power supply. We also recommend using a SD Micro Card with a capacity of 16GB or more (which, functions as the RPI's "hard drive") and is used to store the operating system and necessary operational files.

Optional Components

Under normal circumstances, there are two ways to login to Raspberry Pi: 1) Using a stand-alone monitor. 2) Using a remote desktop or laptop computer monitor "sharing" the PC monitor with your RPi.

Required Accessories for Monitor

If you choose to use an independent monitor, mouse and keyboard, you also need the following accessories:
1. A display with a HDMI interface
2. A Mouse and a Keyboard with an USB interface

As to Pi Zero and Pi Zero W, you also need the following accessories:
1.   A Mini-HDMI to HDMI Adapter and Cable.
2.   A Micro-USB to USB-A Adapter and Cable (Micro USB OTG Cable).
3.   A USB HUB.
4.   USB to Ethernet Interface or USB Wi-Fi receiver.

For different Raspberry Pi Modules, the optional items may vary slightly but they all aim to convert the interfaces to Raspberry Pi standards.

	Pi Zero	Pi A+	Pi Zero W	Pi 3A+	Pi B+/2B	Pi 3B/3B+	Pi 4B
Monitor	Yes (All)						
Mouse	Yes (All)						
Keyboard	Yes (All)						
Micro-HDMI to HDMI Adapter & Cable	Yes	No	Yes	No	No	No	No
Micro-HDMI to HDMI Adapter & Cable	No						Yes
Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)	Yes	No	Yes	No			
USB HUB	Yes	Yes	Yes	Yes	No	No	
USB to Ethernet Interface	select one from two or select two from two		optional		Internal Integration	Internal Integration	
USB Wi-Fi Receiver			Internal Integration	optional			

Required Accessories for Remote Desktop

If you do not have an independent monitor, or if you want to use a remote desktop, you first need to login to Raspberry Pi through SSH, and then open the VNC or RDP service. This requires the following accessories.

	Pi Zero	Pi Zero W	Pi A+	Pi 3A+	Pi B+/2B	Pi 3B/3B+/4B
Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)	Yes	Yes	No		NO	
USB to Ethernet interface	Yes	Yes	Yes			

Raspberry Pi OS

Without Screen - Use Raspberry Pi - under Windows PC: https://youtu.be/YND0RUuP-to

With Screen - Use Raspberry Pi - under Windows PC: https://youtu.be/HEywFsFrj3l

Automatically Method

You can follow the official method to install the system for raspberry pi via visiting link below:
https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2
In this way, the system will be downloaded **automatically** via the application.

Manually Method

After installing the Imager Tool in the **link above**. You can **also** download the system **manually** first.

Visit https://www.raspberrypi.org/downloads/

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. Watch our 45-second video to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

Download for Windows

Download for macOS

Download for Ubuntu for x86

To install on **Raspberry Pi OS**, type
`sudo apt install rpi-imager`
in a Terminal window.

Visit following website to download the OS file.
https://www.raspberrypi.com/software/operating-systems/

And then the zip file is downloaded.

Write System to Micro SD Card

First, put your Micro **SD card** into card reader and connect it to USB port of PC.

Then open imager toll. Choose system that you just downloaded in Use custom.
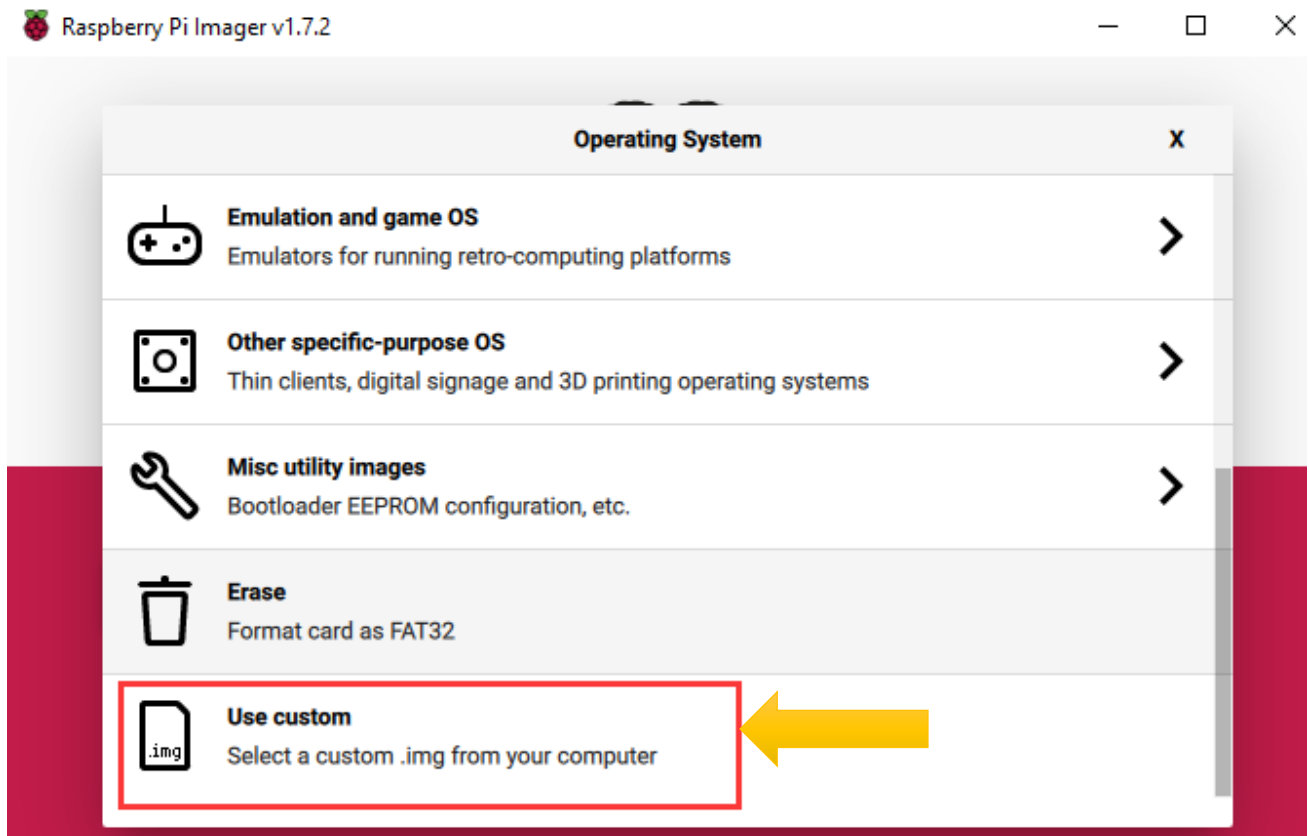
Choose the SD card.
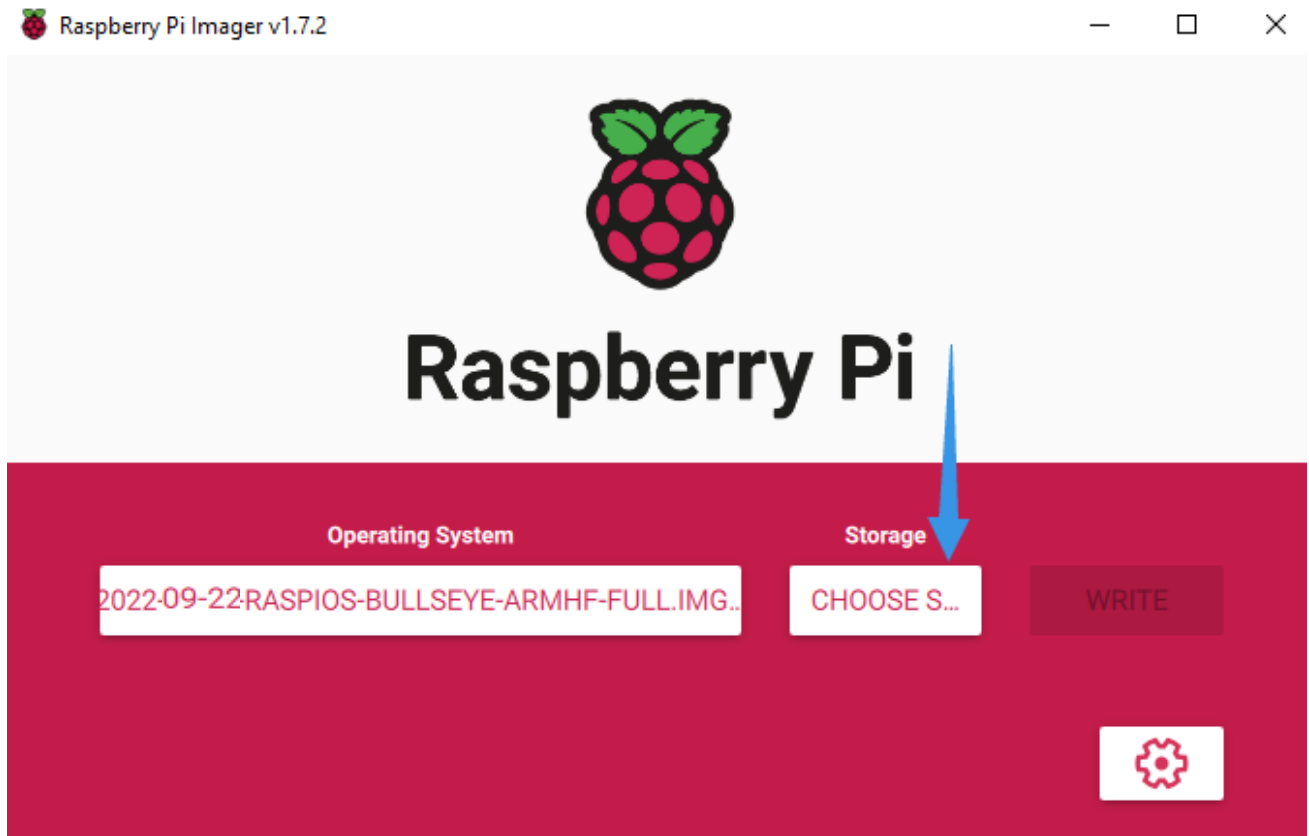
Enable ssh and configure WiFi

Image option.

Enable SSH.
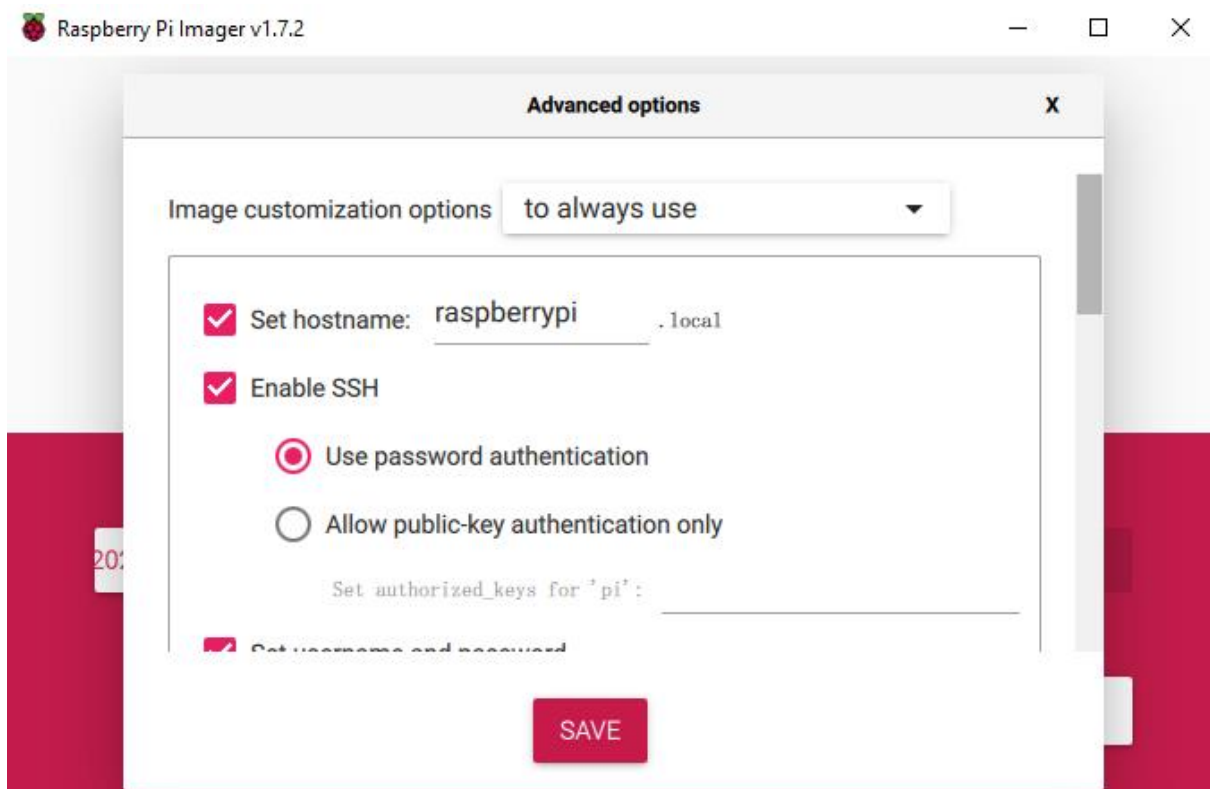
Configure WiFi and location. Here we set username as **pi**, password as **raspberry.** Click Save after setting
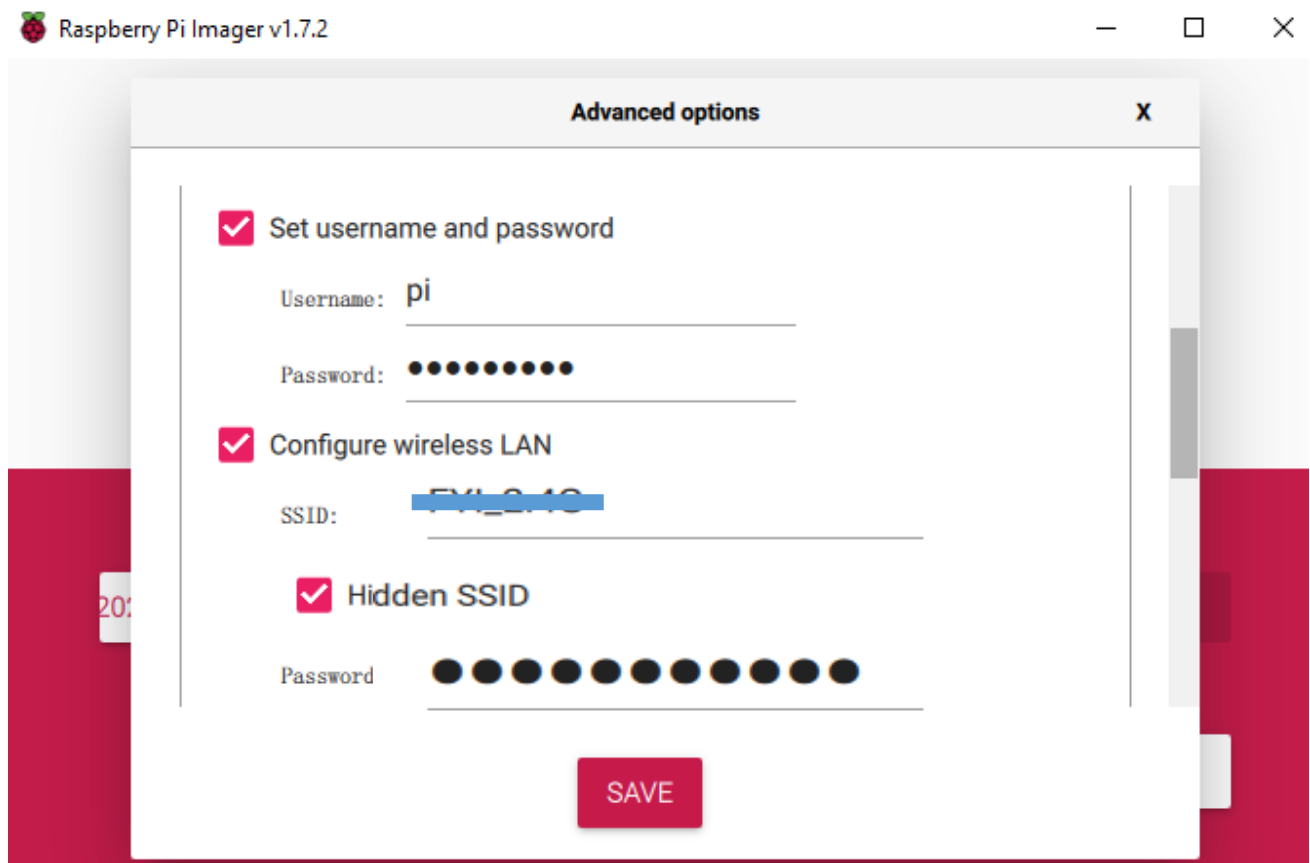
Finally WRITE.
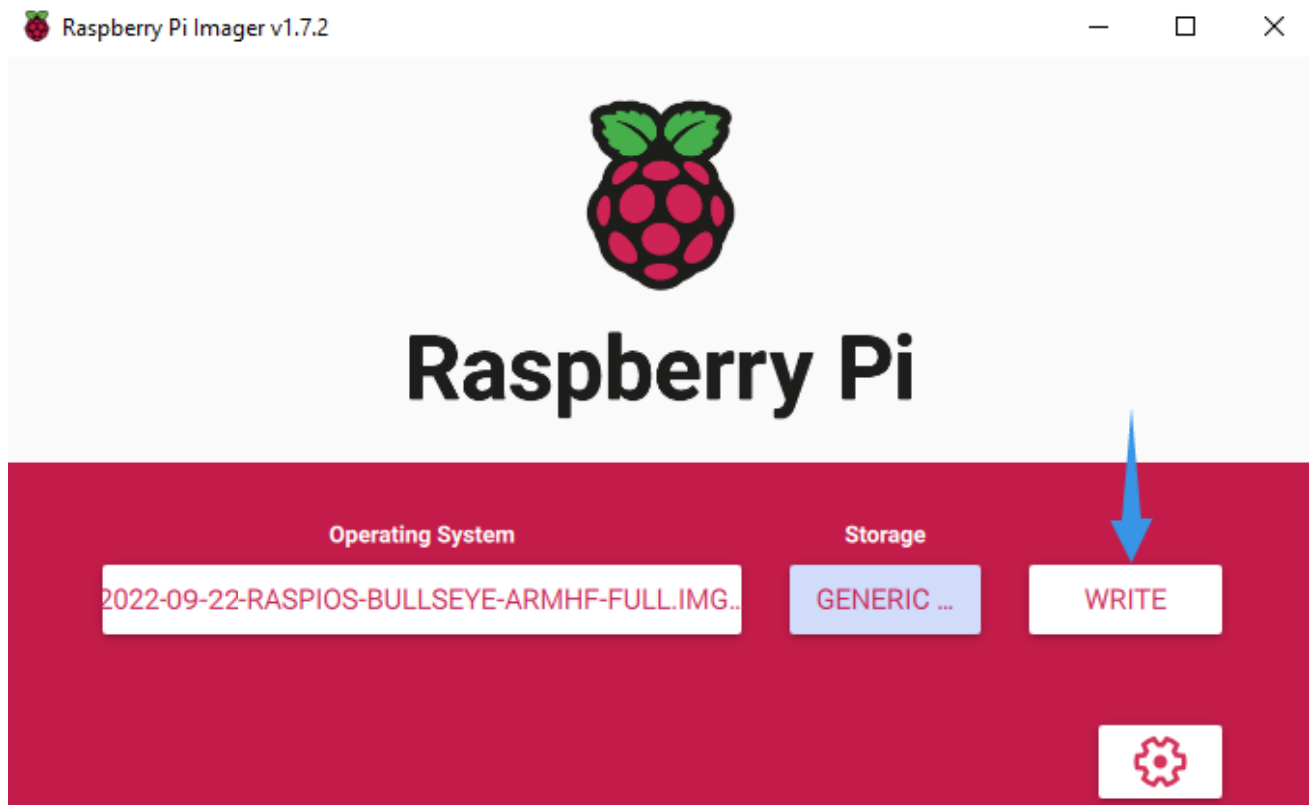
Insert SD card

Then remove SD card from card reader and insert it into Raspberry Pi.
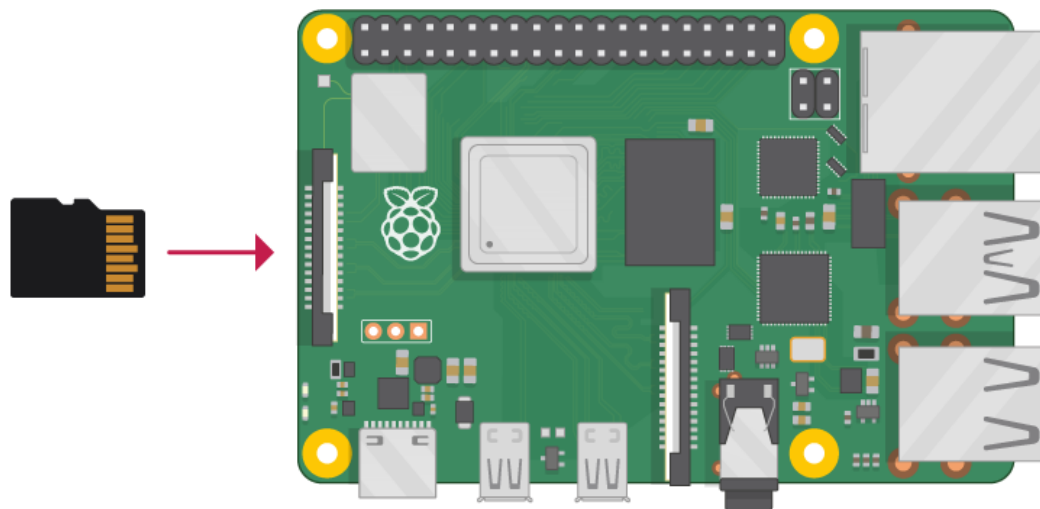
Getting Started with Raspberry Pi

Monitor desktop

If you do not have a spare monitor, please skip to next section Remote desktop & VNC. If you have a spare monitor, please follow the steps in this section.

After the system is written successfully, take out Micro SD Card and put it into the SD card slot of RPi. Then connect your RPi to the monitor through the HDMI port, attach your mouse and keyboard through the USB ports, attach a network cable to the network port and finally, connect your power supply (making sure that it meets the specifications required by your RPi Module Version. Your RPi should start (power up). Later, after setup, you will need to enter your user name and password to login. The default user name: pi; password: raspberry. After login, you should see the following screen.

Congratulations! You have successfully installed the RASPBERRY PI OS operating system on your RPi.

Raspberry Pi 4B, 3B+/3B integrates a Wi-Fi adaptor. You can use it to connect to your Wi-Fi. Then you can use the wireless remote desktop to control your RPi. This will be helpful for the following work. Raspberry Pi of other models can use wireless remote desktop through accessing an external USB wireless card.

Remote desktop & VNC

If you have logged in Raspberry Pi via display, you can skip to VNC Viewer.

If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

MAC OS Remote Desktop

Open the terminal and type following command. **If this command doesn't work, please move to next page.**
ssh pi@raspberrypi.local
The password is **raspberry** by default, case sensitive.
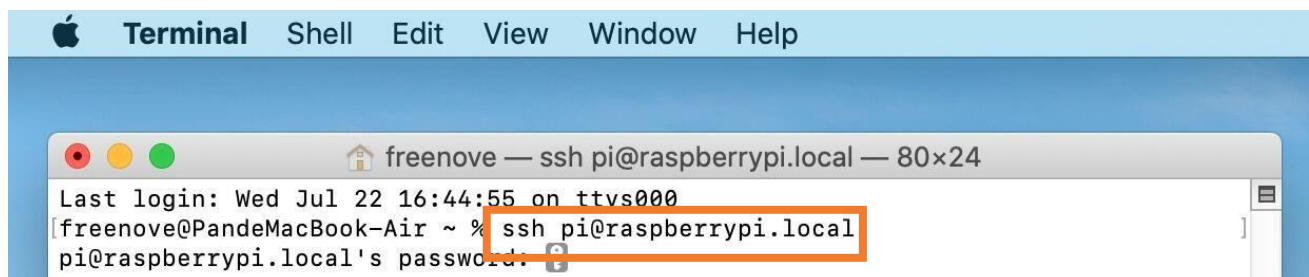
You may need to type **yes** during the process.
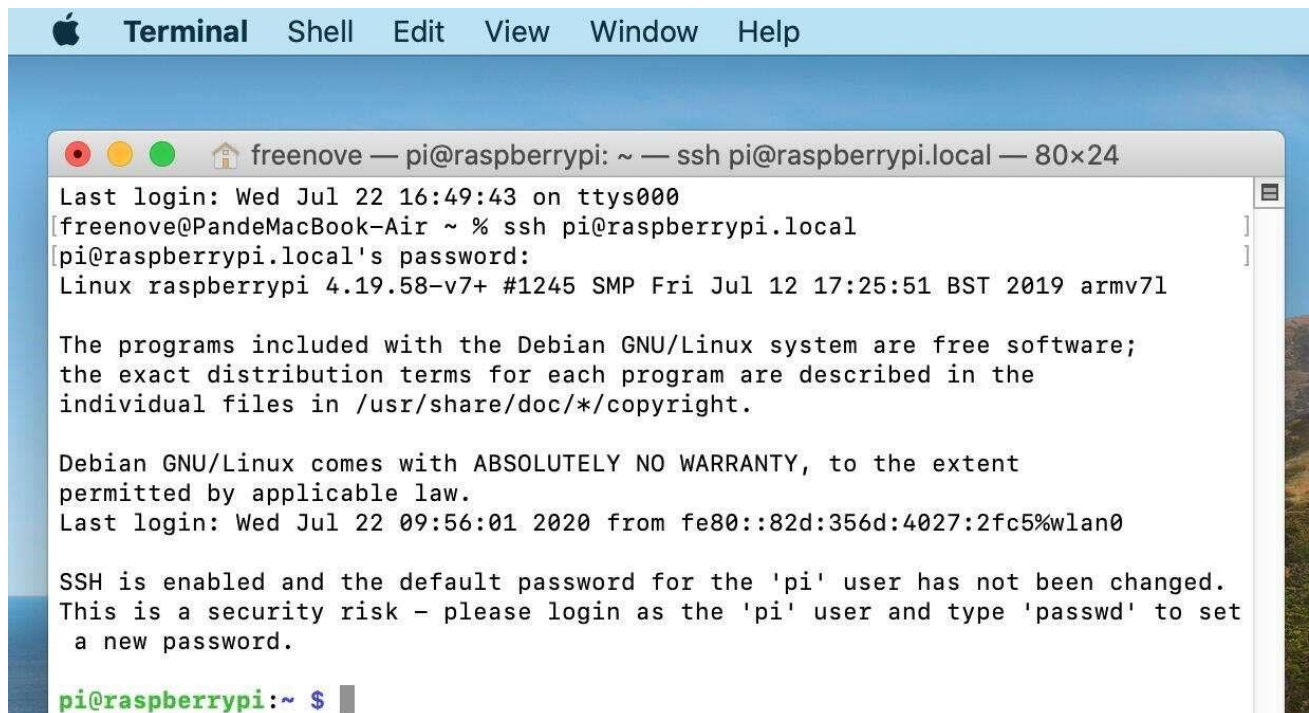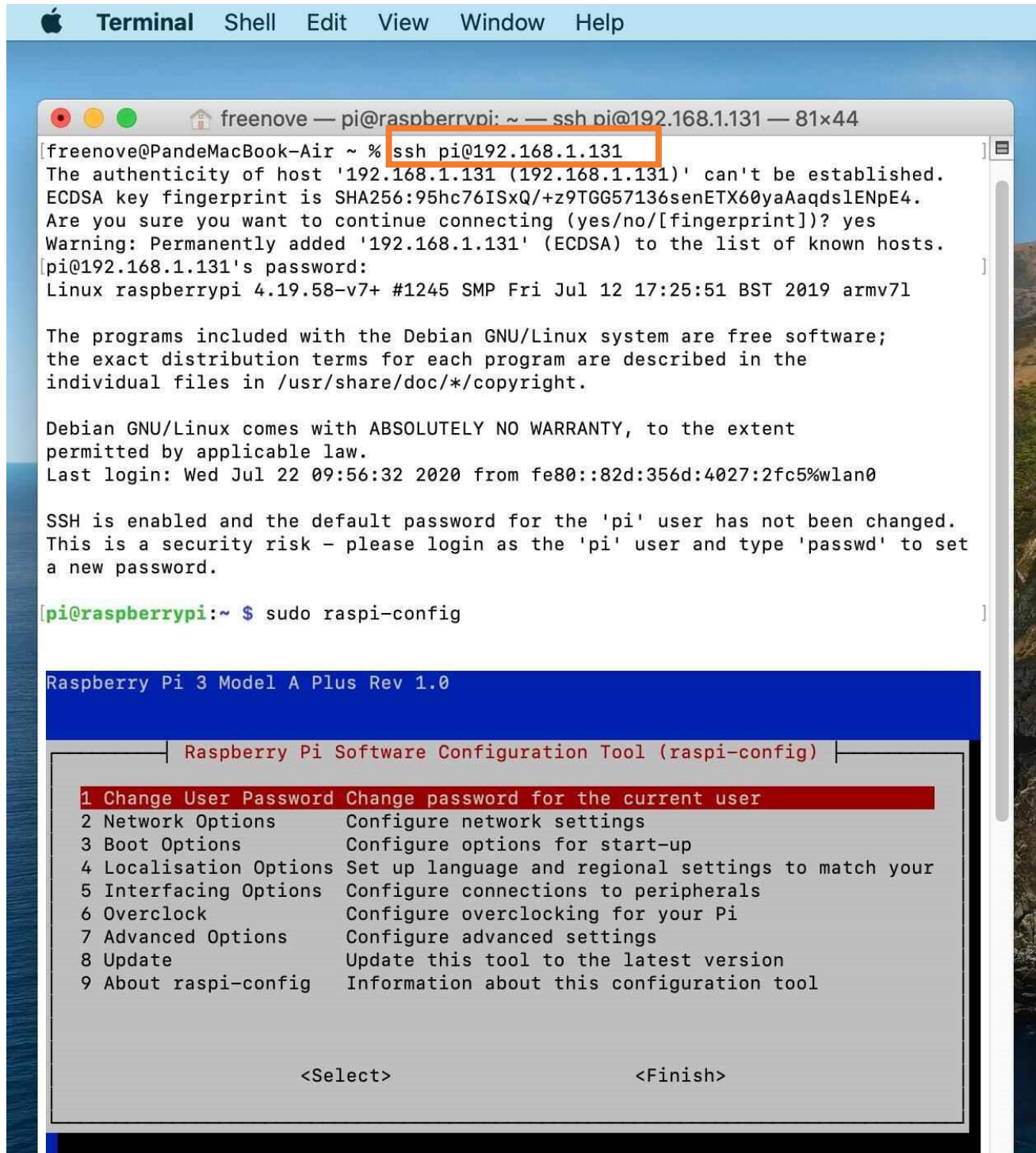
You can also use the IP address to log in Pi.

Enter **router** client to **inquiry IP address** named "raspberry pi". For example, I have inquired to **my RPi IP address, and it is "192.168.1.131".**

Open the terminal and type following command.

ssh pi@192.168.1.131

When you see **pi@raspberrypi**:~ **$,** you have logged in Pi successfully. Then you can skip to next section.

Then you can skip to VNC Viewer.

Windows OS Remote Desktop

If you are using win10, you can use follow way to login Raspberry Pi without desktop.

Press **Win+R**. Enter **cmd**. Then use this command to check IP：

```
ping -4 raspberrypi.local
```

```
Command Prompt

Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>ping -4 raspberrypi.local

Pinging raspberrypi.local [192.168.1.147] with 32 bytes of data:
Reply from 192.168.1.147: bytes=32 time=10ms TTL=64
Reply from 192.168.1.147: bytes=32 time=4ms TTL=64
Reply from 192.168.1.147: bytes=32 time=124ms TTL=64
Reply from 192.168.1.147  bytes=32 time=7ms TTL=64

Ping statistics for 192.168.1.147:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 4ms, Maximum = 124ms, Average = 36ms
```
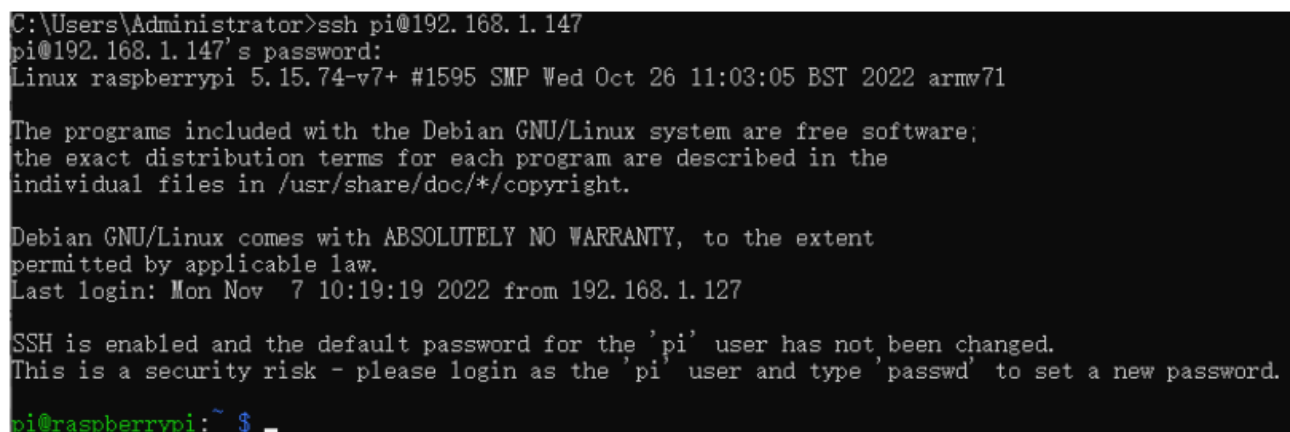
Then 192.168.1.147 is my Raspberry Pi IP.

Or enter **router** client to **inquiry IP address** named "**raspberrypi**". For example, I have inquired to **my RPi IP address**, and it is "192.168.1.147".

```
ssh pi@xxxxxxxxxxx(IP address)
```

Enter the following command：

```
ssh pi@192.168.1.147
```

```
C:\Users\Administrator>ssh pi@192.168.1.147
pi@192.168.1.147's password:
Linux raspberrypi 5.15.74-v7+ #1595 SMP Wed Oct 26 11:03:05 BST 2022 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: Mon Nov  7 10:19:19 2022 from 192.168.1.127

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:~ $
```

VNC Viewer & VNC

Enable VNC

Type the following command. And select Interface Options→P3 VNC → Enter→Yes→OK. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

sudo raspi-config
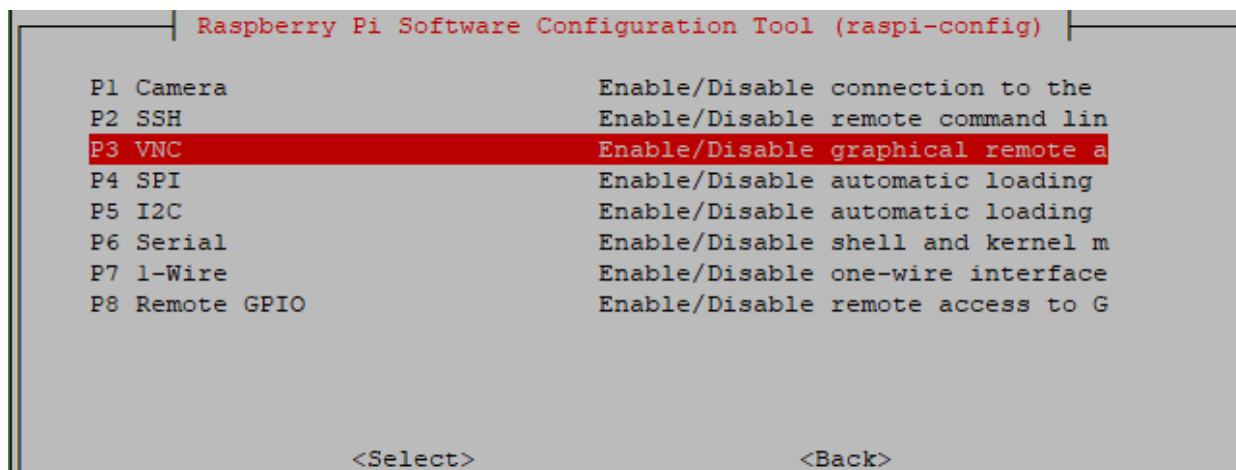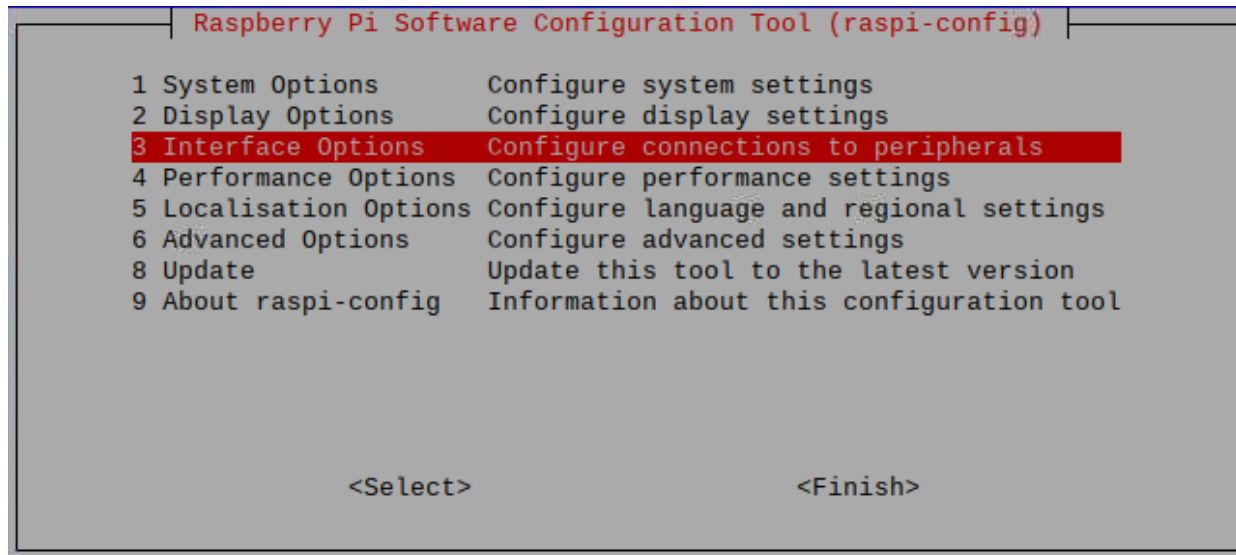
Set Resolution

You can also set other resolutions. If you don't know what to set, you can set it as 1280x720 first.

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

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

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

     D1 Resolution       Set a specific screen resolution
     D2 Underscan        Remove black border around screen
     D3 Pixel Doubling   Enable/disable 2x2 pixel mapping
     D4 Composite Video  Video output options for Raspberry Pi 4
     D5 Screen Blanking  Enable/disable screen blanking
```

```
Choose screen resolution

        Default     720x480
        DMT Mode 4   640x480 60Hz 4:3
        DMT Mode 9   800x600 60Hz 4:3
        DMT Mode 16 1024x768 60Hz 4:3
        DMT Mode 85 1280x720 60Hz 16:9
        DMT Mode 35 1280x1024 60Hz 5:4
        DMT Mode 51 1600x1200 60Hz 4:3
        DMT Mode 82 1920x1080 60Hz 16:9




        <Ok>                    <Cancel>
```

Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File ➔ New Connection. Then the interface is shown below.
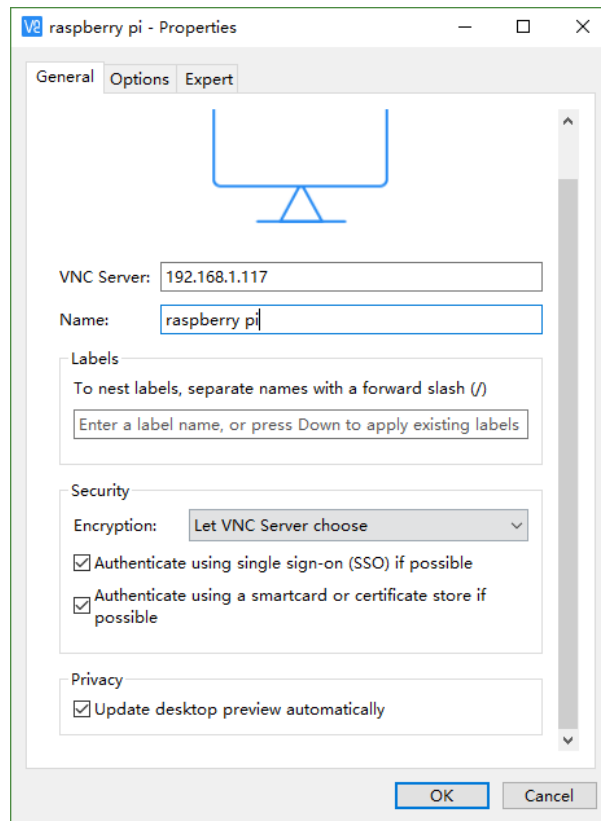
Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

Then on the VNC Viewer panel, double-click new connection you just created,

and the following dialog box pops up.
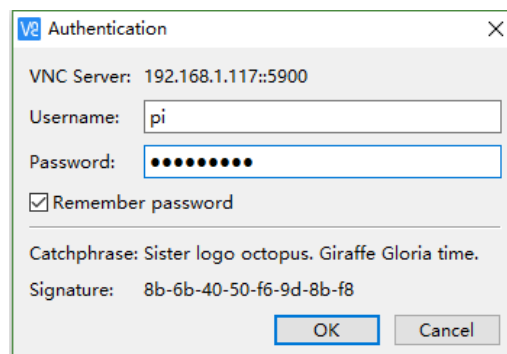
Enter username: **pi** and Password: **raspberry**. And click OK.

Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

If there is black window, please set another resolution.
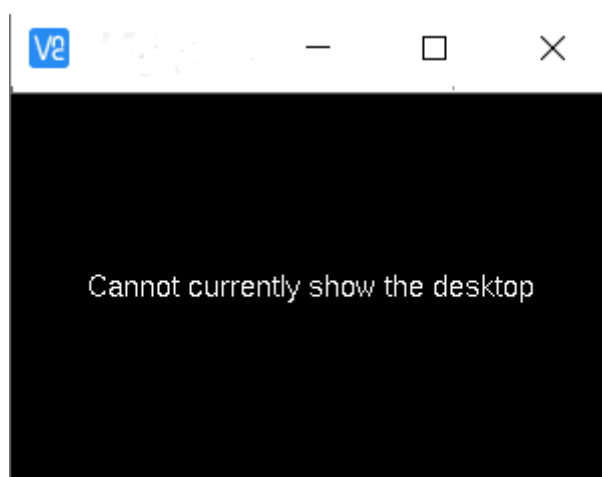
In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. And select Properties->Options label->Scaling. Then set proper scaling.
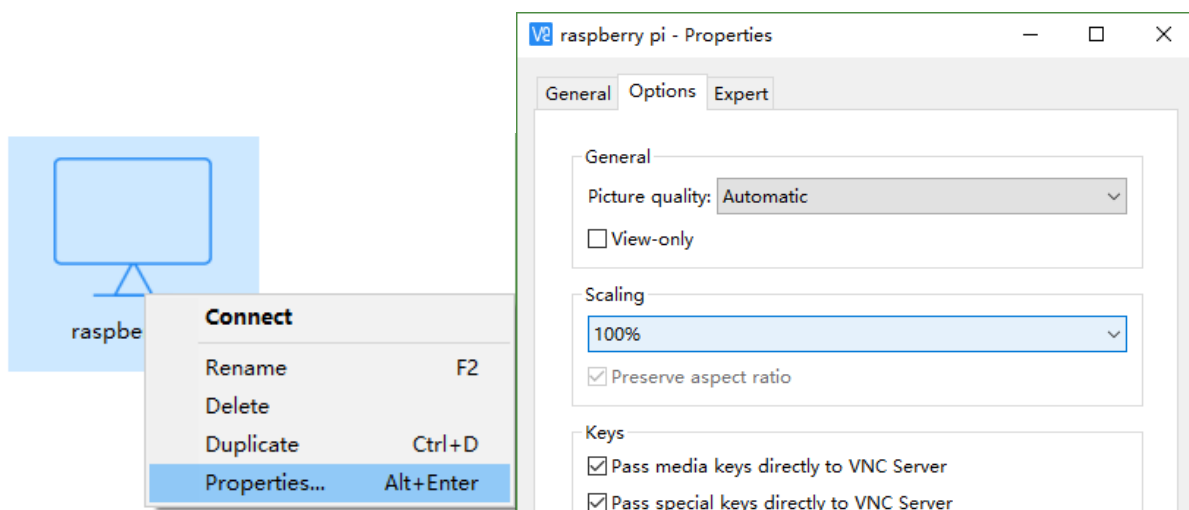
Here, you have logged in to Raspberry Pi successfully by using VNC Viewer and operated proper setting.

Raspberry Pi 4B/3B+/3B integrates a Wi-Fi adaptor.If you did not connect Pi to WiFi. You can connect it to wirelessly control the robot.

Chapter 0 Preparation

Why "Chapter 0"? Because in program code the first number is 0. We choose to follow this rule. In this chapter, we will do some necessary foundational preparation work: Start your Raspberry Pi and install some necessary libraries.

Linux Command

Raspberry Pi OS is based on the Linux Operation System. Now we will introduce you to some frequently used Linux commands and rules.
First, open the Terminal. All commands are executed in Terminal.
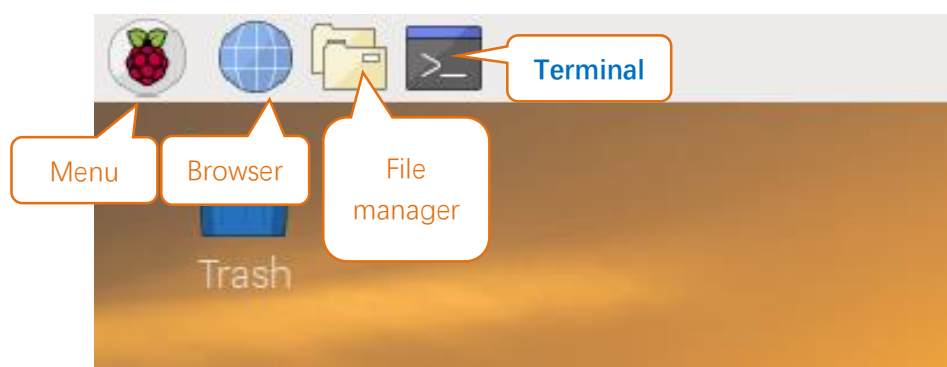
When you click the Terminal icon, following interface appears.
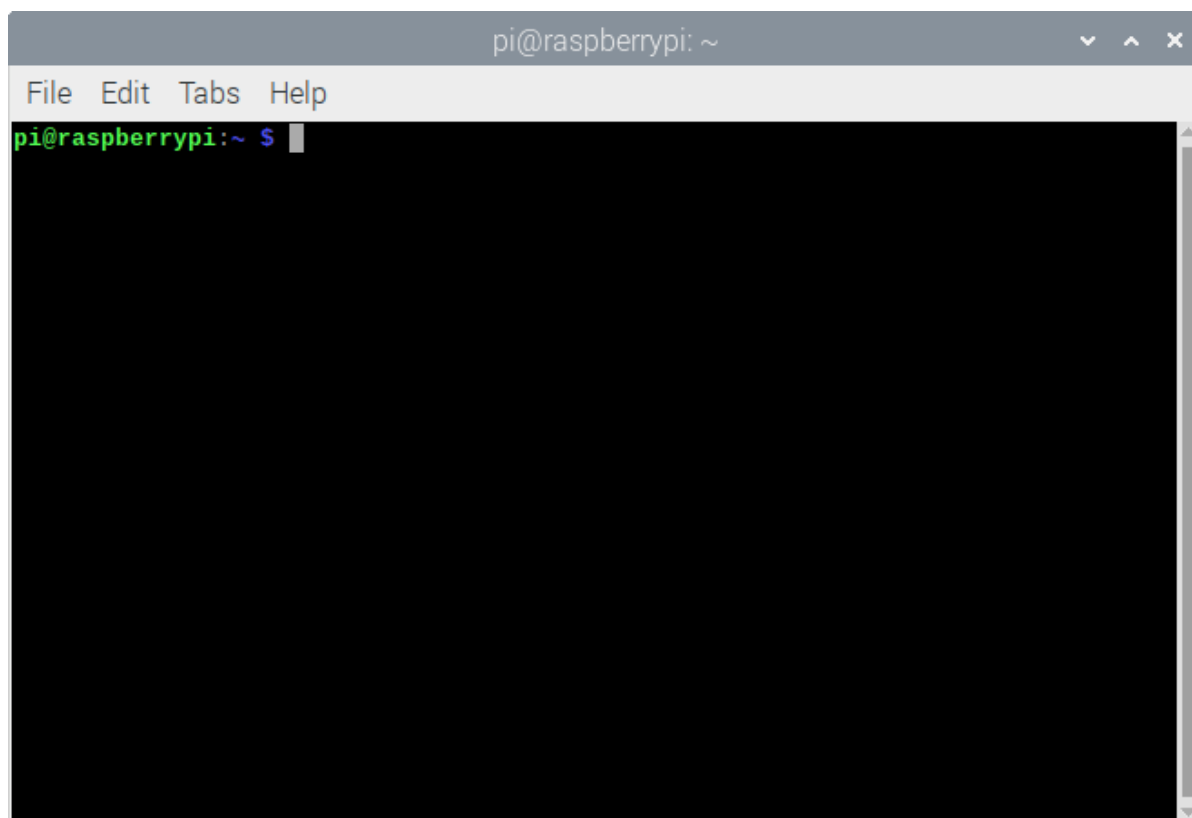
Note: The Linux is case sensitive.

First, type "ls" into the Terminal and press the "Enter" key. The result is shown below:

The "ls" command lists information about the files (the current directory by default).

Content between "$" and "pi@raspberrypi:" is the current working path. "~" represents the user directory, which refers to "/home/pi" here.

"cd" is used to change directory. "/" represents the root directory.

Later in this Tutorial, we will often change the working path. Typing commands under the wrong directory may cause errors and break the execution of further commands.

Many frequently used commands and instructions can be found in the following reference table.

Command	instruction
ls	Lists information about the FILEs (the current directory by default) and entries alphabetically.
cd	Changes directory
sudo + cmd	Executes cmd under root authority
./	Under current directory
gcc	GNU Compiler Collection
git clone URL	Use git tool to clone the contents of specified repository, and URL in the repository address.

There are many commands, which will come later. For more details about commands. You can refer to:
http://www.linux-commands-examples.com

Shortcut Key

Now, we will introduce several commonly used shortcuts that are very useful in Terminal.

1. **Up and Down Arrow Keys**: Pressing "↑" (the Up key) will go backwards through the command history and pressing "↓" (the Down Key) will go forwards through the command history.

2. **Tab Key**: The Tab key can automatically complete the command/path you want to type. When there is only one eligible option, the command/path will be completely typed as soon as you press the Tab key even you only type one character of the command/path.

As shown below, under the '~' directory, you enter the Documents directory with the "cd" command. After typing "cd D", pressing the Tab key (there is no response), pressing the Tab key again then all the files/folders that begin with "D" will be listed. Continue to type the letters "oc" and then pressing the Tab key, the "Documents" is typed automatically.

```
pi@raspberrypi:~ $ cd D
Desktop/    Documents/ Downloads/
pi@raspberrypi:~ $ cd Doc█
```

```
pi@raspberrypi:~ $ cd D
Desktop/    Documents/ Downloads/
pi@raspberrypi:~ $ cd Documents/
```

Install GPIO Zero Python library

GPIO Zero is a simple interface to GPIO devices with Raspberry Pi. GPIO Zero is installed by default in the Raspberry Pi OS desktop image, and the Raspberry Pi Desktop image for PC/Mac, both available from raspberrypi.org. Follow these guides to installing on Raspberry Pi OS Lite and other operating systems.
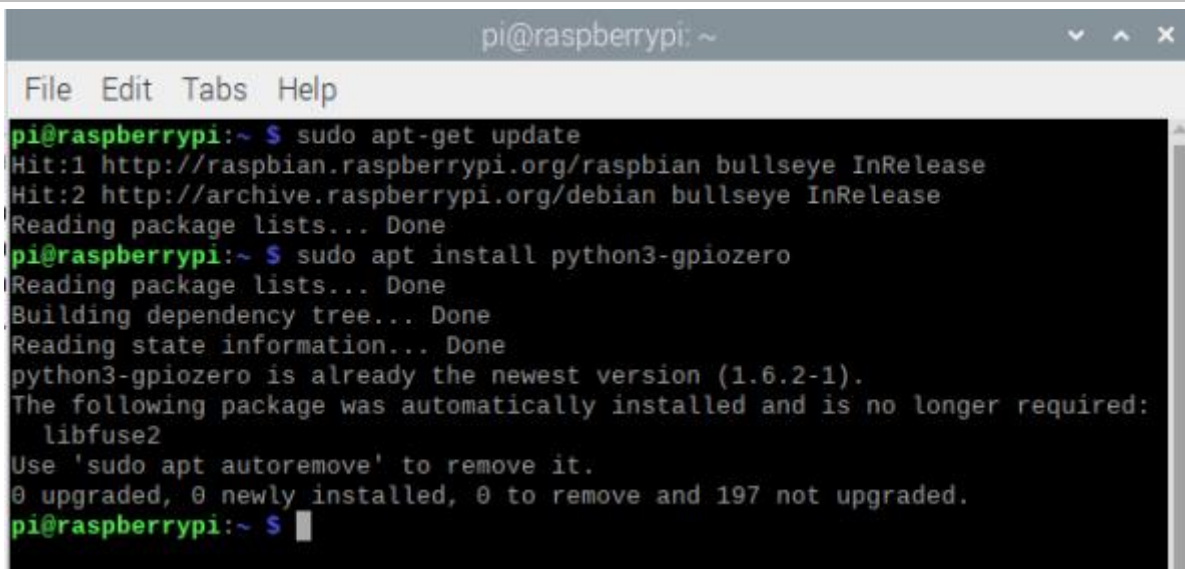
GPIO Zero Python library Installation Steps

To install the GPIO Zero Python library, please open the Terminal and then follow the steps and commands below.

Note: For a command containing many lines, execute them one line at a time.

Enter the following commands **one by one** in the **terminal** to install GPIO Zero:

```
sudo apt-get update
sudo apt install python3-gpiozero
```
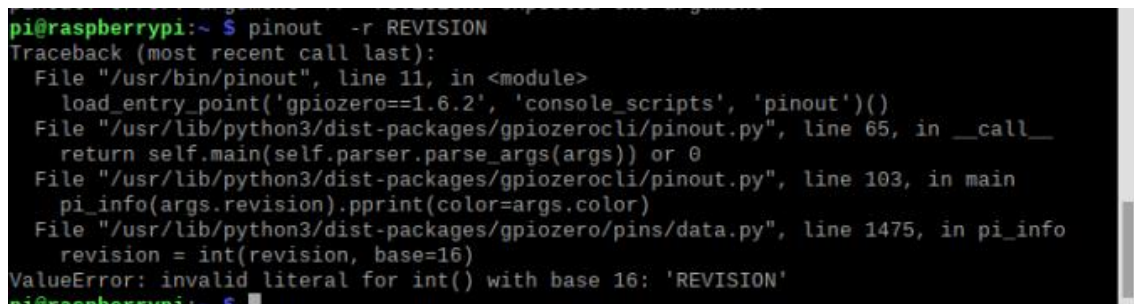

If you're using another operating system on your Raspberry Pi, you may need to use pip to install GPIO Zero instead. Install pip using get-pip and then type:

```
sudo pip3 install gpiozero
```

Run the gpiozero command to check the installation:

```
pinout -r REVISION
```

That should give you some confidence that the installation was a success.

Obtain the Project Code

After the above installation is completed, you can visit our official website (http://www.freenove.com) or our GitHub resources at (https://github.com/freenove) to download the latest available project code.
In this tutorial, we provide Python language code for each project.

This is the method for obtaining the code:
In the pi directory of the RPi terminal, enter the following command.

```
cd
git clone --depth 1 https://github.com/freenove/Freenove_RFID_Starter_Kit_for_Raspberry_Pi
```
(There is no need for a password. If you get some errors, please check your commands.)
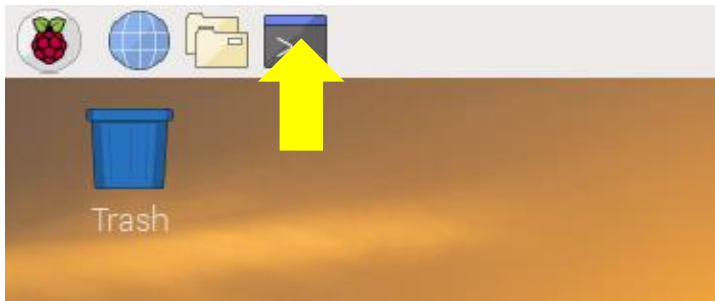
After the download is completed, a new folder "Freenove_RFID_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

This folder name seems a little too long. We can simply rename it by using the following command.

```
mv Freenove_RFID_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

"Freenove_Kit" is now the new and much shorter folder name.
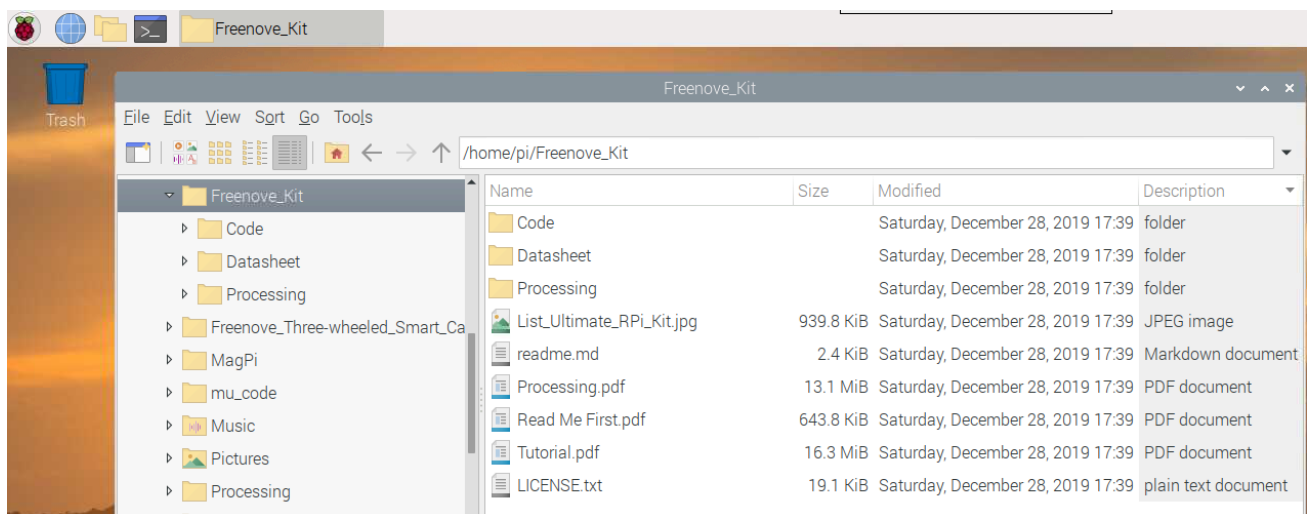
If you have no experience with Python, we suggest that you refer to this website for basic information and knowledge.
https://python.swaroopch.com/basics.html

Python2 & Python3

Python code, used in our kits, can now run on Python2 and Python3. **Python3 is recommend**. If you want to use Python2, please make sure your Python version is 2.7 or above. Python2 and Python3 are not fully compatible. However, Python2.6 and Python2.7 are transitional versions to python3, therefore you can also use Python2.6 and 2.7 to execute some Python3 code.

You can type "python2" or "python3" respectively into Terminal to check if python has been installed. Press Ctrl-Z to exit.

Type "python", and Terminal shows that it links to python3.

Set Python3 as default python

First, execute python to check the default python on your raspberry Pi. Press Ctrl-Z to exit.

If it is python3, you can skip this section.

If it is python2, you need execute the following commands to set default python to python3.

1. Enter directory /usr/bin

cd /usr/bin

2. Delete the originalpython link.

sudo rm python

3. Create new python links to python.

sudo ln -s python3 python

4. Check python. Press Ctrl-Z to exit.

python

```
pi@raspberrypi:/usr/bin $ sudo rm python
pi@raspberrypi:/usr/bin $ sudo ln -s python3 python
pi@raspberrypi:/usr/bin $ python
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you want to set python2 as default python in **other projects**, just repeat the commands above and change python3 to python2.

Shortcut Key

Now, we will introduce several shortcuts that are very **useful** and **commonly used** in terminal.

1. **up and down arrow keys**. History commands can be quickly brought back by using up and down arrow keys, which are very useful when you need to reuse certain commands.

When you need to type commands, pressing "↑" will go backwards through the history of typed commands, and pressing "↓" will go forwards through the history of typed command.

2. **Tab key**. The Tab key can automatically complete the command/path you want to type. When there are multiple commands/paths conforming to the already typed letter, pressing Tab key once won't have any result. And pressing Tab key again will list all the eligible options. This command/path will be completely typed as soon as you press the Tab key when there is only one eligible option.

As shown below, under the '~'directory, enter the Documents directory with the "cd" command. After typing "cd D", press Tab key, then there is no response. Press Tab key again, then all the files/folders that begin with "D" is listed. Continue to type the character "oc", then press the Tab key, and then "Documents" is completely typed automatically.

```
pi@raspberrypi:~ $ cd D
Desktop/    Documents/ Downloads/
pi@raspberrypi:~ $ cd Doc
```

```
pi@raspberrypi:~ $ cd D
Desktop/    Documents/ Downloads/
pi@raspberrypi:~ $ cd Documents/
```

Chapter 1 LED

This chapter is the Start Point in the journey to build and explore RPi electronic projects. We will start with simple "Blink" project.

Project 1.1 Blink

In this project, we will use RPi to control blinking a common LED.

Component List

Raspberry Pi (Recommended: Raspberry Pi 4B / 3B+ / 3B Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)	GPIO Extension Board & Ribbon Cable
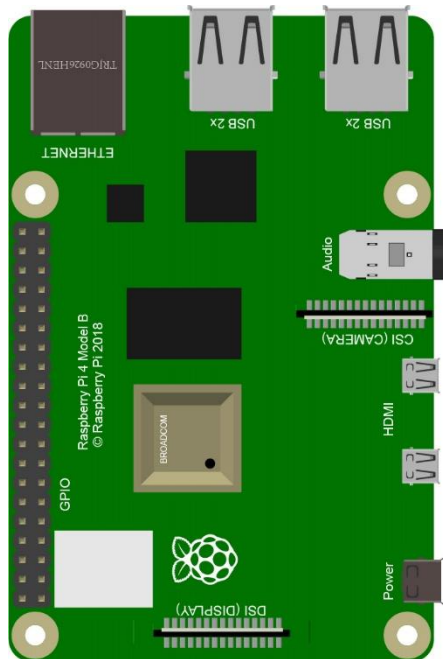
Breadboard x1

LED x1	Resistor 220Ω x1	Jumper
		Specific quantity depends on the circuit.

In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each project. Later, they will be reference by text only (no images as in above).

GPIO

GPIO: General Purpose Input/Output. Here we will introduce the specific function of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions.

When programming GPIO pins there are 3 different ways to reference them: GPIO Numbering, Physical Numbering and WiringPi GPIO Numbering.

BCM GPIO Numbering

The Raspberry Pi CPU uses Broadcom (BCM) processing chips BCM2835, BCM2836 or BCM2837. GPIO pin numbers are assigned by the processing chip manufacturer and are how the computer recognizes each pin. The pin numbers themselves do not make sense or have meaning as they are only a form of identification. Since their numeric values and physical locations have no specific order, there is no way to remember them so you will need to have a printed reference or a reference board that fits over the pins.

Each pin's functional assignment is defined in the image below:
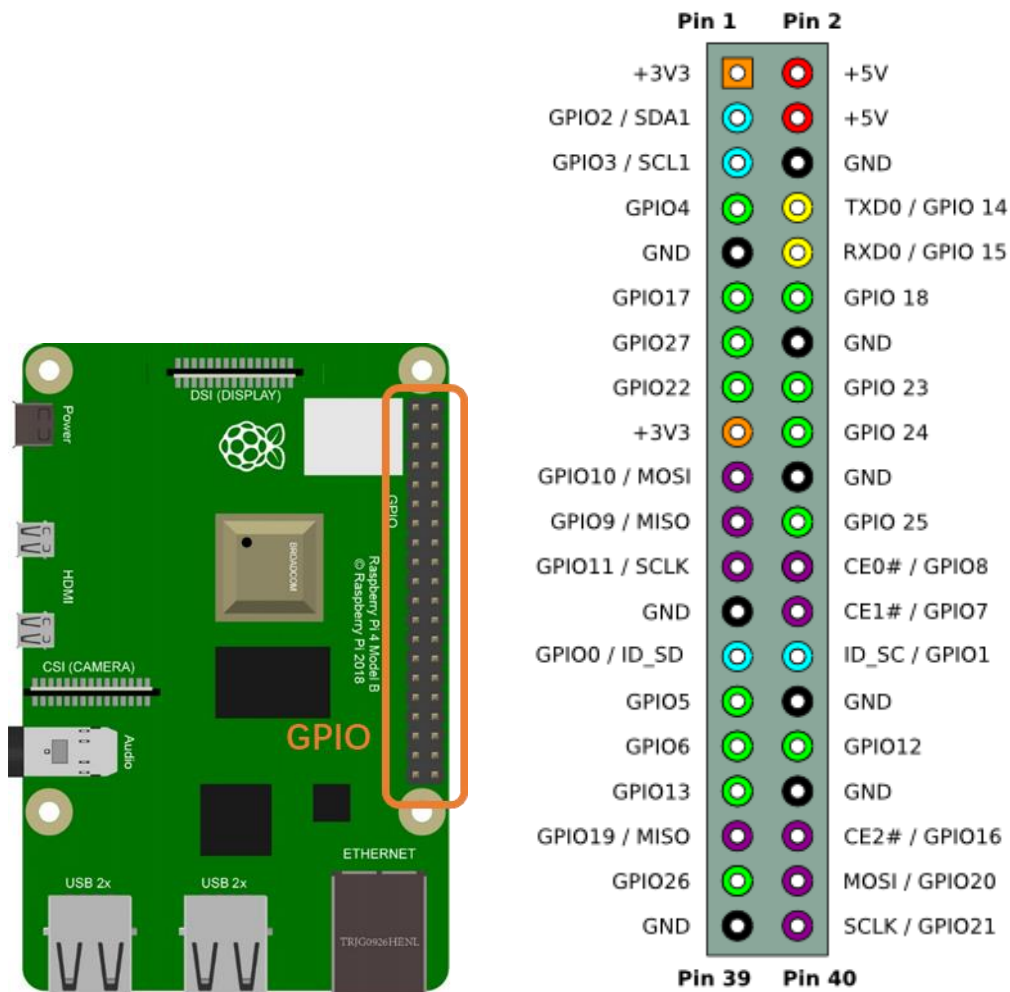
For more details about pin definition of GPIO, please refer to http://pinout.xyz/

PHYSICAL Numbering

Another way to refer to the pins is by simply counting across and down from pin 1 at the top left (nearest to the SD card). This is 'Physical Numbering', as shown below:
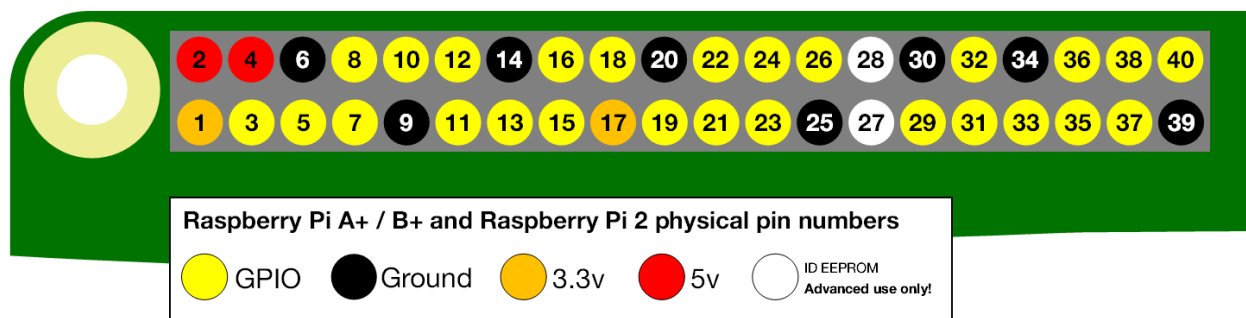
GPIO Numbering

You can use the following command to view their correlation.

`Pinout`

Circuit

First, disconnect your RPi from the GPIO Extension Shield. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield.
CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your RPi!

Schematic diagram
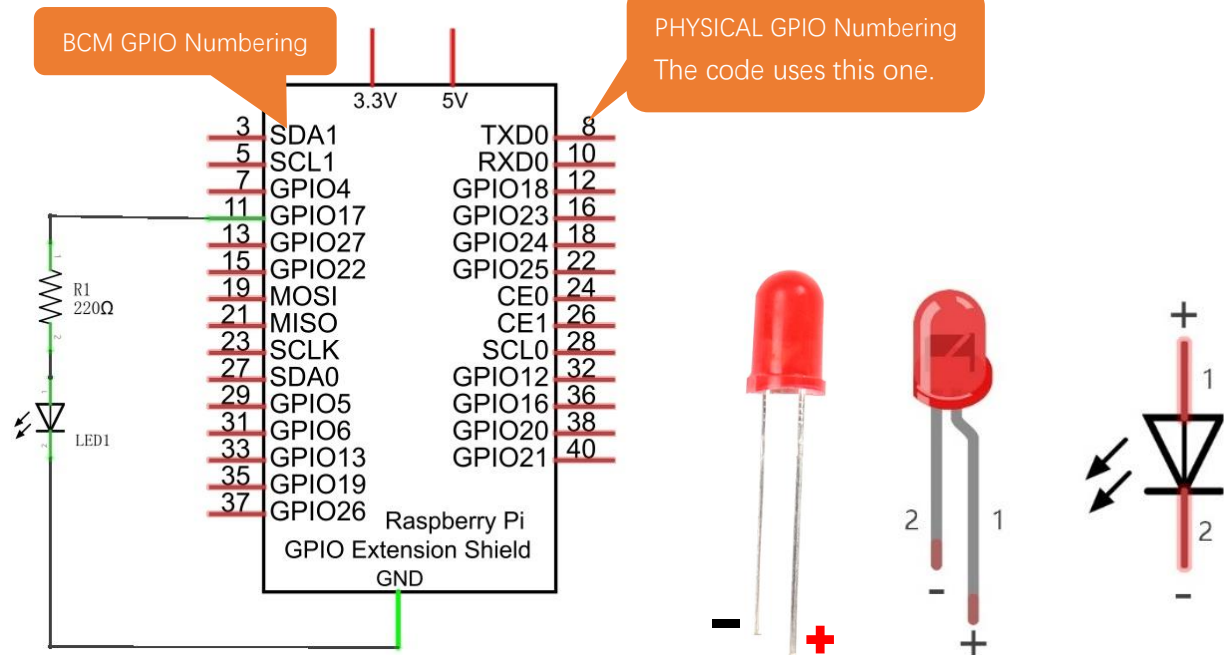
BCM GPIO Numbering

PHYSICAL GPIO Numbering
The code uses this one.

Hardware connection. **If you need any support, please contact us via:** support@freenove.com

Longer Pin
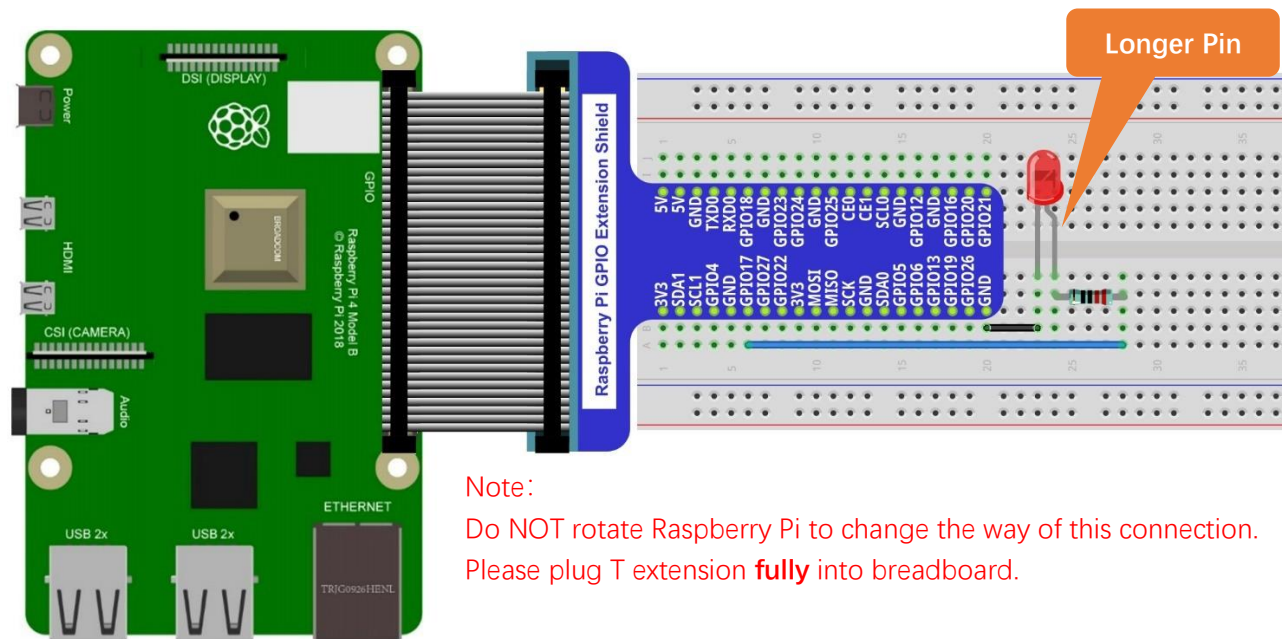
Note：
Do NOT rotate Raspberry Pi to change the way of this connection.
Please plug T extension **fully** into breadboard.

The connection of Raspberry Pi 400 **and T extension board is as below.** Don't reverse the ribbon.
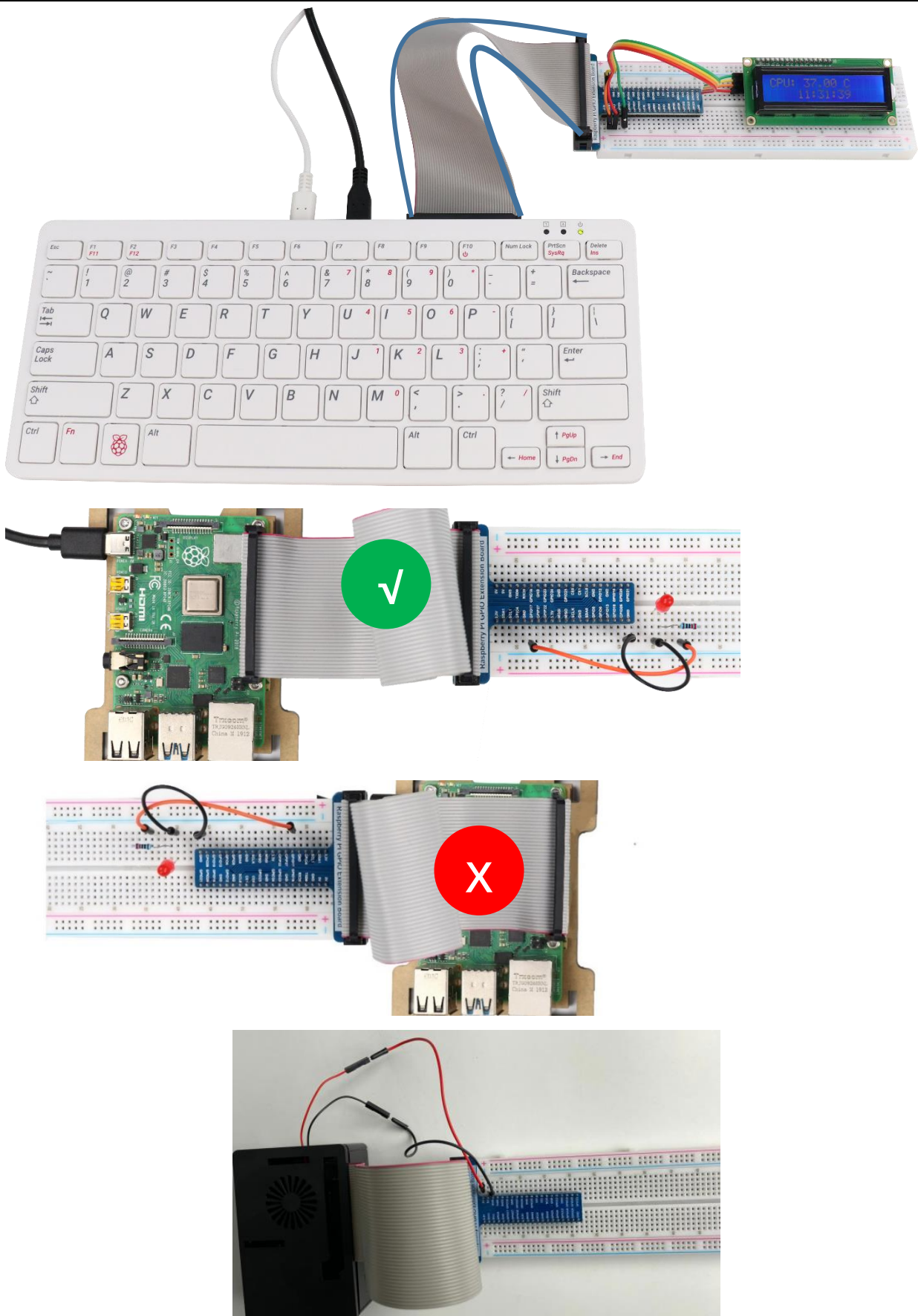
If you have a fan, you can connect it to 5V GND of breadboard via jumper wires.

<table>
<tr><td colspan="2">How to distinguish resistors?</td></tr>
<tr><td colspan="2">There are only three kind of resistors in this kit.</td></tr>
<tr><td>The one with 1 red ring is 10KΩ</td><td></td></tr>
<tr><td>The one with 2 red rings is 220Ω</td><td></td></tr>
<tr><td>The one with 0 red ring is 1KΩ</td><td></td></tr>
</table>

Future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

Component knowledge

LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-) output, which is also referred to as Ground (GND). This type of component is known as "Polar" (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burnt out.
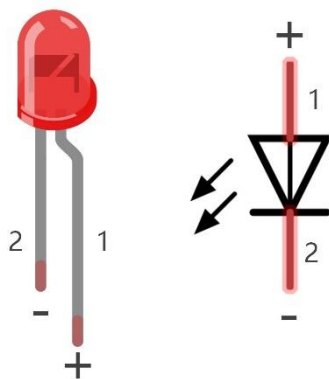
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.
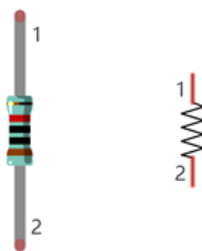
The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: I=V/R known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is: I=U/R=5V/10kΩ=0.0005A=0.5mA.
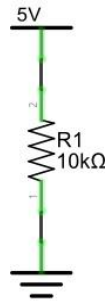
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

Breadboard

Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the ways the pins have shared electrical connection and the right picture shows the actual internal metal, which connect these rows electrically.
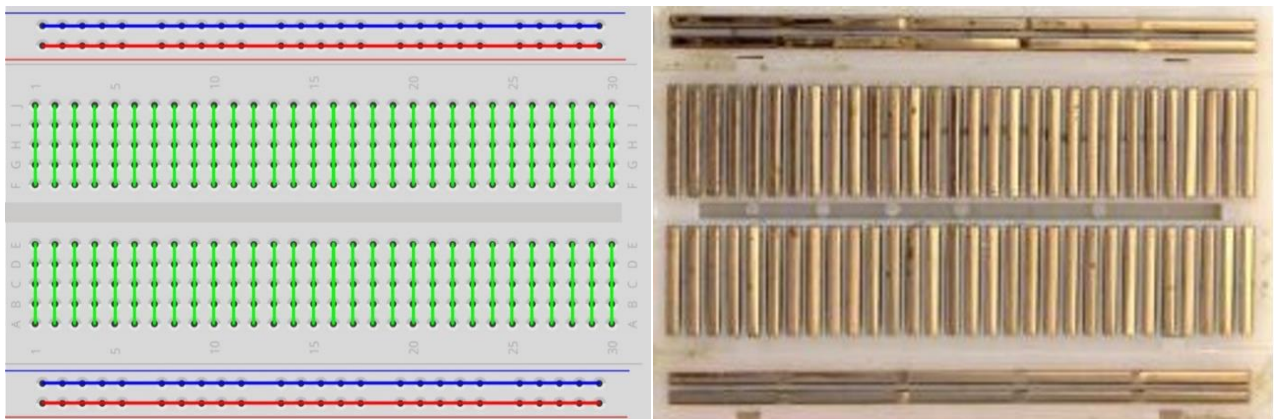
GPIO Extension Board

GPIO board is a convenient way to connect the RPi I/O ports to the breadboard directly. The GPIO pin sequence on Extension Board is identical to the GPIO pin sequence of RPi.
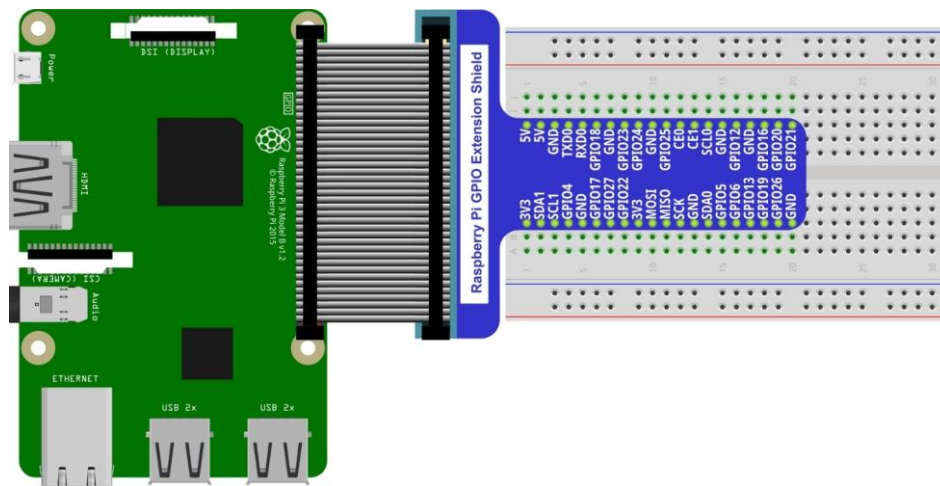
Code

According to the circuit, when the GPIO17 of RPi output level is high, the LED turns ON. Conversely, when the GPIO17 RPi output level is low, the LED turns OFF. Therefore, we can let GPIO17 cycle output high and output low level to make the LED blink. We will use Python code to achieve the target.

Python Code 1.1.1 Blink

Now, we will use Python language to make a LED blink.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 01.1.1_Blink directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/01.1.1_Blink
```

2. Use python command to execute python code blink.py.

```
python Blink.py
```

The LED starts blinking.

You can press "Ctrl+C" to end the program. The following is the program code:

```
1    from gpiozero import LED
2    from time import sleep
3
4    led = LED(17)              # define LED pin according to BCM Numbering
5    #led = LED("J8:11")        # BOARD Numbering
6    '''
7    # pins numbering, the following lines are all equivalent
8    led = LED(17)              # BCM
9    led = LED("GPIO17")        # BCM
10   led = LED("BCM17")         # BCM
11   led = LED("BOARD11")       # BOARD
12   led = LED("WPI0")          # WiringPi
13   led = LED("J8:11")         # BOARD
14   '''
15   def loop():
16       while True:
```

```
17          led.on()     # turn on LED
18          print ('led turned on >>>')  # print message on terminal
19          sleep(1)    # wait 1 second
20          led.off()   # turn off LED
21          print ('led turned off <<<') # print message on terminal
22          sleep(1)    # wait 1 second
23
24  if __name__ == '__main__':    # Program entrance
25      print ('Program is starting ... \n')
26      try:
27          loop()
28      except KeyboardInterrupt:  # Press ctrl-c to end the program.
29          print("Ending program")
```

Import the LED class from the gpiozero library.

```
from gpiozero import LED
```

Create an LED assembly for controlling the LED.

```
led = LED(17)          # define LED pin according to BCM Numbering
```

Turn on LED device.

```
led.on()     # turn on LED
```

Turn off LED devices.

```
led.off()   # turn off LED
```

The main function turns on the LED for one second and then turns it off for one second, which repeats endless.

```
def loop():
    while True:
        led.on()     # turn on LED
        print ('led turned on >>>')  # print message on terminal
        sleep(1)    # wait 1 second
        led.off()   # turn off LED
        print ('led turned off <<<') # print message on terminal
        sleep(1)    # wait 1 second
```

Reference

About GPIO Zero:

GPIO Zero

A simple interface to GPIO devices with Raspberry Pi, Using the GPIO Zero library makes it easy to get started with controlling GPIO devices with Python. The library is comprehensively documented at
https://gpiozero.readthedocs.io/en/stable/
https://github.com/gpiozero/gpiozero

For more information about the methods used by the LED class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#led

For more information about the methods used by the DigitalOutputDevice class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#digitaloutputdevice

"import time" time is a module of python.

https://docs.python.org/2/library/time.html?highlight=time%20time#module-time

In Python, libraries and functions used in a script must be imported by name at the top of the file, with the exception of the functions built into Python by default.

For example, to use the LED interface from GPIO Zero, it should be explicitly imported:

```python
from gpiozero import LED
```

Now LEDis available directly in your script:

```python
led = LED(17)            # define LED pin according to BCM Numbering
#led = LED("J8:11")      # BOARD Numbering
```

Alternatively, the whole GPIO Zero library can be imported:

```python
import gpiozero
```

In this case, all references to items within GPIO Zero must be prefixed:

```python
led = gpiozero.LED(17)           # define LED pin according to BCM Numbering
#led = gpiozero.LED("J8:11")     # BOARD Numbering
```

Pin Numbering

This library uses Broadcom (BCM) pin numbering for the GPIO pins, as opposed to physical (BOARD) numbering. Unlike in the RPi.GPIO library, this is not configurable. However, translation from other schemes can be used by providing prefixes to pin numbers (see below).

Any pin marked "GPIO" in the diagram below can be used as a pin number. For example, if an LED was attached to "GPIO17" you would specify the pin number as 17 rather than 11:
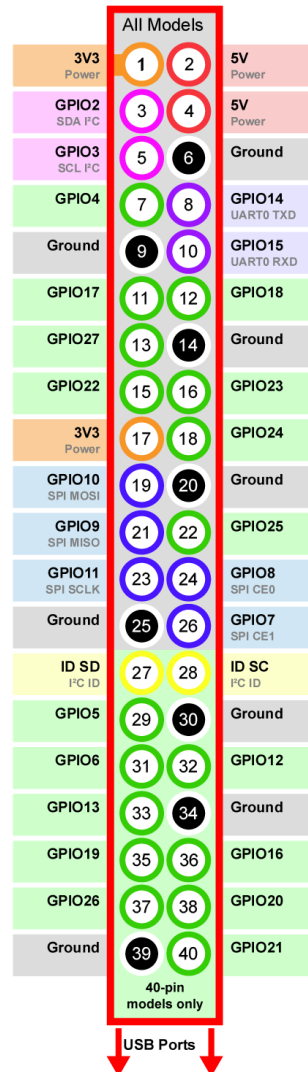
If you wish to use physical (BOARD) numbering you can specify the pin number as "BOARD11". If you are familiar with the wiringPi pin numbers (another physical layout) you could use "WPI0" instead. Finally, you can specify pins as "header:number", e.g. "J8:11" meaning physical pin 11 on header J8 (the GPIO header on modern Pis). Hence, the following lines are all equivalent:

```
led = LED(17)
led = LED("GPIO17")
led = LED("BCM17")
led = LED("BOARD11")
led = LED("WPI0")
led = LED("J8:11")
```

Note that these alternate schemes are merely translations. If you request the state of a device on the command line, the associated pin number will always be reported in the Broadcom (BCM) scheme:

```
led = LED("BOARD11")
led
<gpiozero.LED object on pin GPIO17, active_high=True, is_active=False>
```

In this tutorial, we will use the default integer pin number in the Broadcom (BCM) layout.

GPIO Numbering Relationship

WingPi	BCM(Extension)	Physical		BCM(Extension)	WingPi
3.3V	3.3V	1	2	5V	5V
8	GPIO2/SDA1	3	4	5V	5V
9	GPIO3/SCL1	5	6	GND	GND
7	GPIO4	7	8	GPIO14/TXD0	15
GND	GND	9	10	GPIO15/RXD0	16
0	GPIO17	11	12	GPIO18	1
2	GPIO27	13	14	GND	GND
3	GPIO22	15	16	GPIO23	4
3.3V	3.3V	17	18	GPIO24	5
12	GPIO10/MOSI	19	20	GND	GND
13	GPIO9/MOIS	21	22	GPIO25	6
14	GPIO11/SCLK	23	24	GPIO8/CE0	10
GND	GND	25	26	GPIO7/CE1	11
30	GPIO0/SDA0	27	28	GPIO1/SCL0	31
21	GPIO5	29	30	GND	GND
22	GPIO6	31	32	GPIO12	26
23	GPIO13	33	34	GND	GND
24	GPIO19	35	36	GPIO16	27
25	GPIO26	37	38	GPIO20	28
GND	GND	39	40	GPIO21	29

In loop(), there is a while loop, which is an endless loop (a while loop). That is, the program will always be executed in this loop, unless it is ended because of external factors. In this loop, set LED output high level, then the LED turns ON. After a period of time delay, set LED output low level, then the LED turns OFF, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
def loop():
    while True:
        led.on()      # turn on LED
        print ('led turned on >>>')  # print message on terminal
        sleep(1)      # wait 1 second
        led.off()     # turn off LED
        print ('led turned off <<<') # print message on terminal
        sleep(1)      # wait 1 second
```

In gpiozero, at the end of your script, cleanup is run automatically, restoring your GPIO pins to the state they were found.To explicitly close a connection to a pin, you can manually call the close() method on a device object:

```
led = LED(17)
```

```
    led.on()
    led
<gpiozero.LED object on pin GPIO17, active_high=True, is_active=True>
    led.close()
    led
<gpiozero.LED object closed>
```

This means that you can reuse the pin for another device, and that despite turning the LED on (and hence, the pin high), after calling close() it is restored to its previous state (LED off, pin low).

In this tutorial, most projects have added an active run cleanup program to restore the GPIO pin to the found default state.

Freenove Car, Robot and other products for Raspberry Pi

We also have car and robot kits for Raspberry Pi. You can visit our website for details.

https://www.amazon.com/freenove

FNK0043   **Freenove 4WD Smart Car Kit for Raspberry Pi**

https://www.youtube.com/watch?v=4Zv0GZUQjZc

FNK0050   **Freenove Robot Dog Kit for Raspberry Pi**

https://www.youtube.com/watch?v=7BmIZ8_R9d4

FNK0052   **Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi**
https://youtu.be/LvghnJ2DNZ0

Chapter 2 Buttons & LEDs

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and RPI was the control part. In practical applications, we not only make LEDs flash, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as turn on LEDs, make a buzzer beep and so on.

Control:
RPI, Arduino,
MCU and etc.

Input:
buttons, switches,
sensors and etc.

Output:
LED, buzzer,
motor and etc.
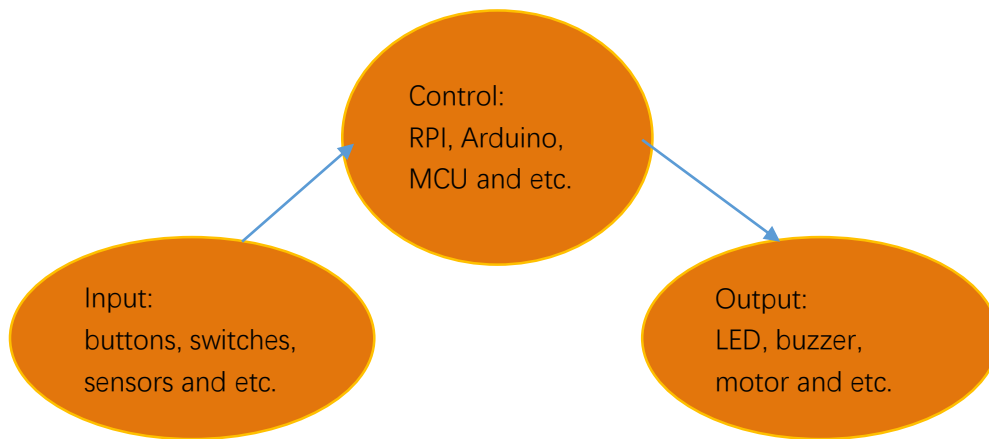
Next, we will build a simple control system to control an LED through a push button switch.

Project 2.1 Push Button Switch & LED

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1	Resistor 220Ω x1	Resistor 10kΩ x2	Push Button Switch x1
Jumper Wire				

Please Note: In the code "button" represents switch action.

Component knowledge

Push Button Switch

This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:
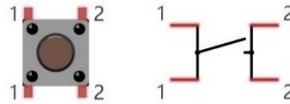
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram
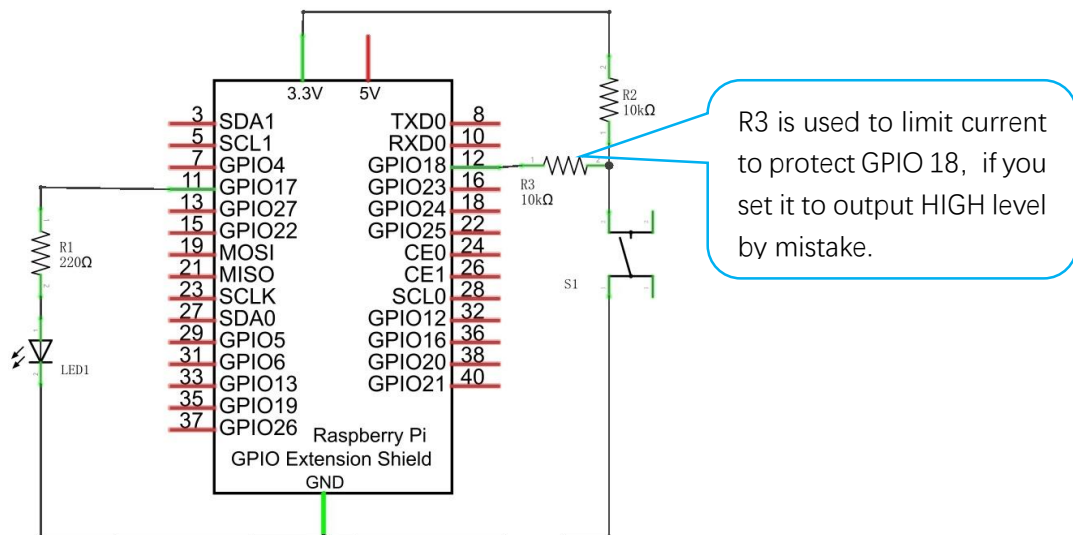
R3 is used to limit current to protect GPIO 18, if you set it to output HIGH level by mistake.

Hardware connection. If you need any support, please feel free to contact us via:**support@freenove.com**
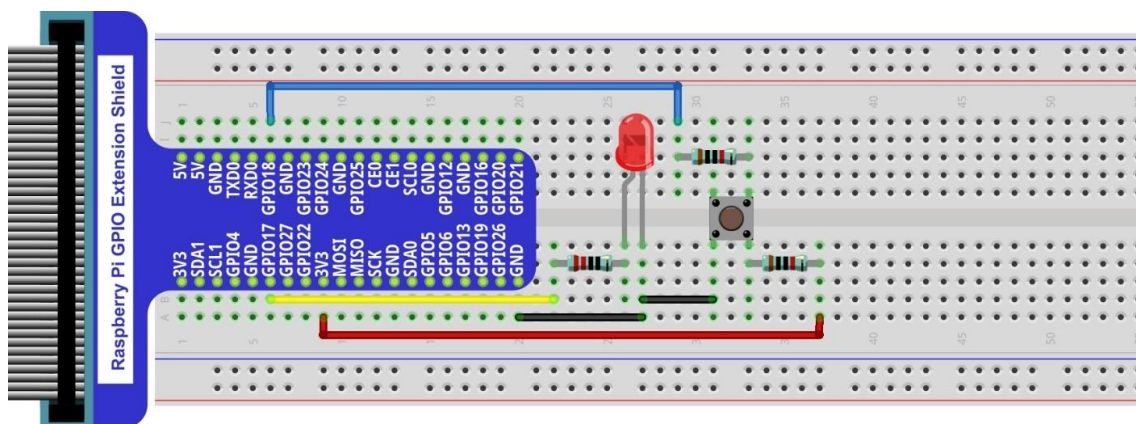
There are two kinds of push button switch in this kit.
The smaller push button switches are contained in a plastic bag.

This is how it works.
When button switch is released:
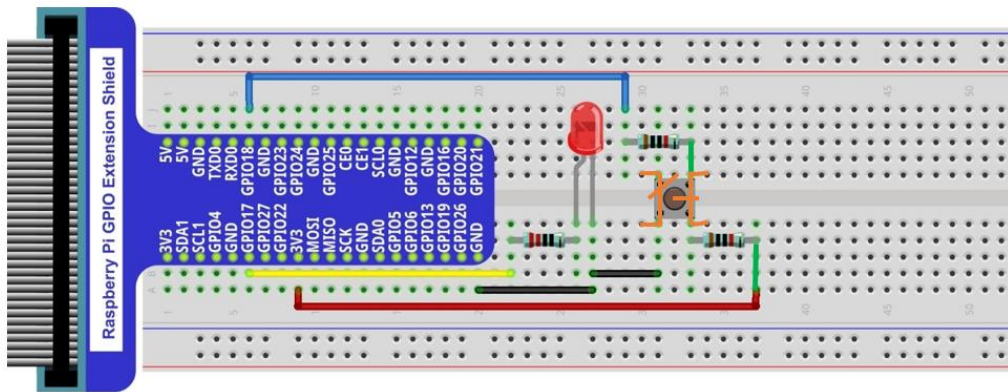
When button switch is pressed:
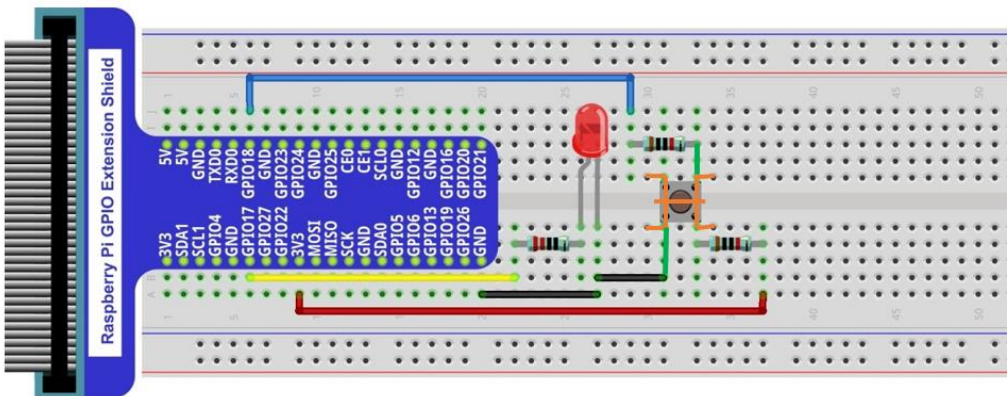
Code

This project is designed for learning how to use Push Button Switch to control an LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch.

Python Code 2.1.1 ButtonLED

First, observe the project result, then learn about the code in detail. Remember in code "button" = switch function

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 02.1.1_ButtonLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/02.1.1_ButtonLED
```

2. Use Python command to execute btnLED.py.

```
python ButtonLED.py
```

Then the Terminal window continues to show the characters "led off…", press the switch button and the LED turns ON and then Terminal window shows "led on…". Release the button, then LED turns OFF and then the terminal window text "led off…" appears. You can press "Ctrl+C" at any time to terminate the program.

The following is the program code:

```
1    from gpiozero import LED, Button
2
```

```
3   led = LED(17)      # define LED pin according to BCM Numbering
4   button = Button(18) # define Button pin according to BCM Numbering
5
6   def loop():
7       while True:
8           if button.is_pressed:  # if button is pressed
9               led.on()        # turn on led
10              print("Button is pressed, led turned on >>>") # print information on terminal
            else : # if button is relessed
11              led.off() # turn off led
12              print("Button is released, led turned off <<<")
13
14  if __name__ == '__main__':     # Program entrance
15      print ('Program is starting...')
16      try:
17          loop()
18      except KeyboardInterrupt:  # Press ctrl-c to end the program.
19          print("Ending program")
```

Import the Button class that controls Button from the gpiozero library.

```
from gpiozero import LED, Button
```

Define GPIO17 as the LED control pin and GPIO18 as the button control pin. The button is set to the input mode with a pull-up resistor by default.

```
led = LED(17)        # define LED pin according to BCM Numbering
button = Button(18) # define Button pin according to BCM Numbering
```

The loop continuously determines whether the key is pressed. When the button is pressed, the variable button.is_pressed has a value of 1 and the LED lights up. Otherwise, the LED will be off.

```
def loop():
    while True:
        if button.is_pressed:  # if button is pressed
            led.on()        # turn on led
            print("Button is pressed, led turned on >>>") # print information on terminal
        else : # if button is relessed
            led.off() # turn off led
            print("Button is released, led turned on <<<")
```

For more information about GPIOZero, please refer to the link below:

https://gpiozero.readthedocs.io/en/stable/

For more information about the methods used by the Button class in the GPIO Zero library,please refer to:

https://gpiozero.readthedocs.io/en/stable/api_input.html#button

Project 2.2 MINI Table Lamp

We will also use a Push Button Switch, LED and RPi to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

First, let us learn something about the push button switch.

Debounce a Push Button Switch

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it stabilizes in a new state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as "bounce".
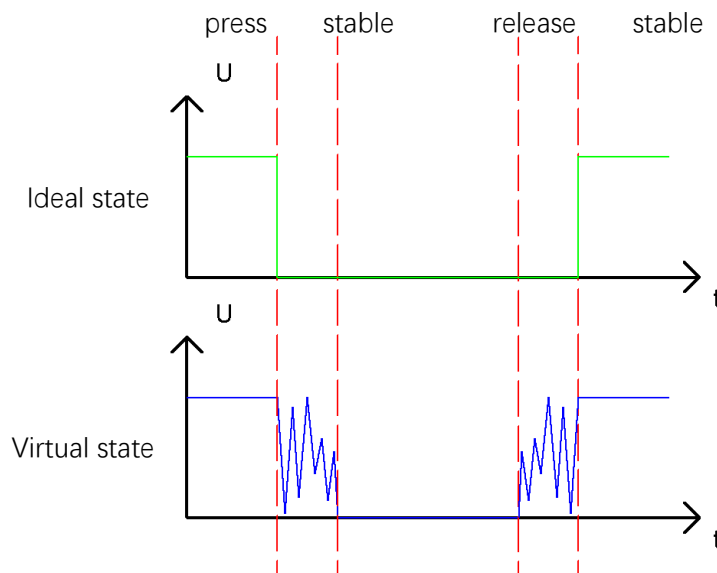
Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

This project needs the same components and circuits as we used in the previous section.

Code

In this project, we still detect the state of Push Button Switch to control an LED. Here we need to define a variable to define the state of LED. When the button switch is pressed once, the state of LED will be changed once. This will allow the circuit to act as a virtual table lamp.

Python Code 2.2.1 Tablelamp

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1.   Use cd command to enter 02.2.1_Tablelamp directory of Python code

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/02.2.1_Tablelamp
```

2.   Use python command to execute python code "Tablelamp.py".

```
python Tablelamp.py
```

When the program is executed, pressing the Button Switch once turns the LED ON. Pressing the Button Switch again turns the LED OFF.

```python
1  from gpiozero import LED, Button
2  import time
3
4  led = LED(17) # define LED pin according to BCM Numbering
5  button = Button(18) # define Button pin according to BCM Numbering
6
7  def onButtonPressed():  # When button is pressed, this function will be executed
8      led.toggle()
9      if led.is_lit :
10         print("Led turned on >>>")
11     else :
12         print("Led turned off <<<")
13 def loop():
14     #Button detect
15     button.when_pressed = onButtonPressed
16     while True:
17         time.sleep(1)
18 def destroy():
19     led.close()
20     button.close()
21 if __name__ == '__main__':     # Program entrance
22     print ('Program is starting...')
23     try:
24         loop()
25     except KeyboardInterrupt:  # Press ctrl-c to end the program.
26         destroy()
27         print("Ending program")
```

In GPIO Zero, you assign the when_pressed and when_released properties to set up callbacks on those actions.

Once it detects that the button is pressed, it executes the specified function onButtonPressed().In the onButtonPressed function, the led. toggle () function reverses the state of the LED device.If it's on, turn it off; If it's off, turn it on.Each time the key is pressed, the state of the LED will change once.

```python
def onButtonPressed():  # When button is pressed, this function will be executed
    led.toggle()
    if led.is_lit :
        print("Led turned on >>>")
    else :
        print("Led turned off <<<")


def loop():
    #Button detect
    button.when_pressed = onButtonPressed
    while True:
        time.sleep(1)
```

To explicitly close a connection to a pin, you can manually call the close() method on a device object:

```python
def destroy():
    led.close()
    button.close()


    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
        print("Ending program")
```

For more information about the methods used by the Button class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_input.html#button

Chapter 3 LED Bar Graph

We have learned how to control one LED to blink. Next, we will learn how to control a number of LEDs.

Project 3.1 Flowing Water Light

In this project, we use a number of LEDs to make a flowing water light.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Bar Graph LED x1	Resistor 220Ω x10
Jumper Wire x 1		

Component knowledge

Let us learn about the basic features of these components to use and understand them better.

Bar Graph LED

A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.
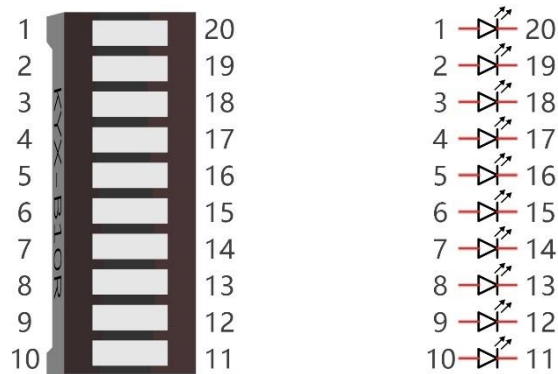
Circuit

A reference system of labels is used in the circuit diagram below. Pins with the same network label are connected together.

Schematic diagram
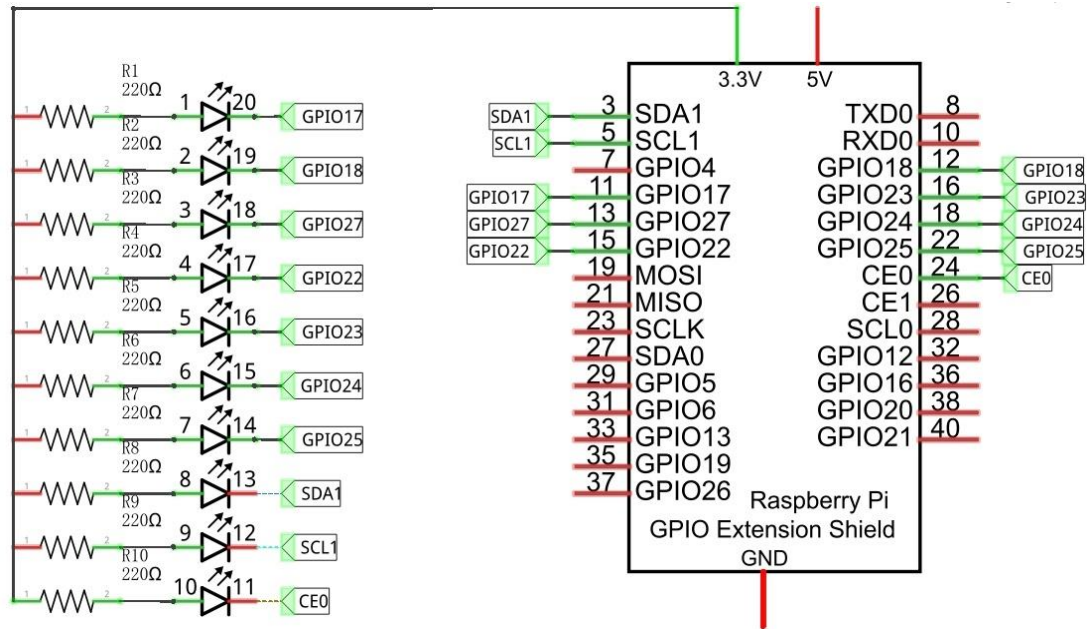
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
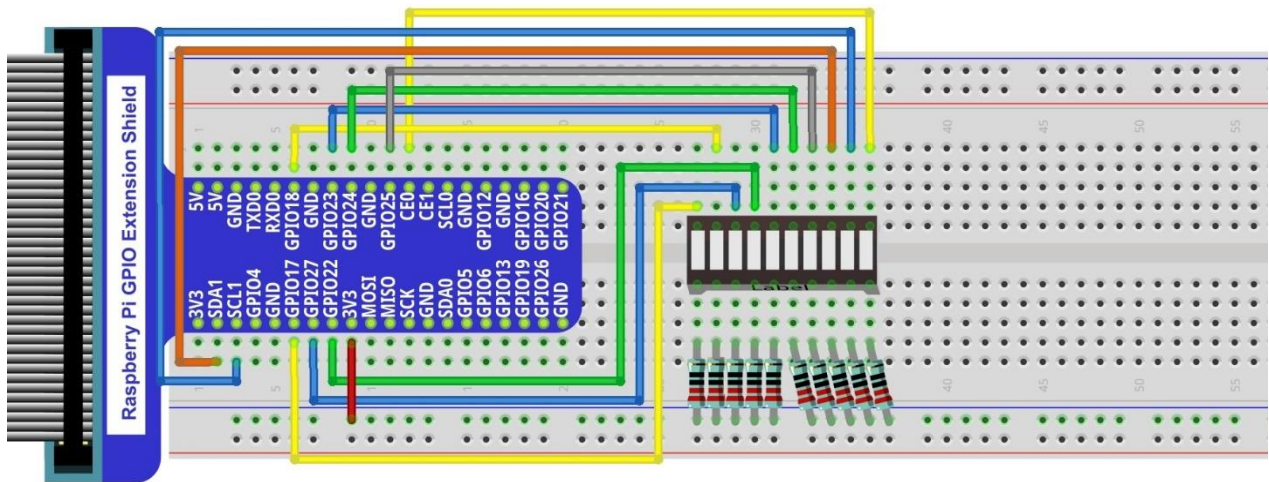
If LEDbar doesn't work, rotate LEDbar 180° to try. The label is random.

In this circuit, the cathodes of the LEDs are connected to the GPIO, which is different from the previous circuit. The LEDs turn ON when the GPIO output is low level in the program.

Code

This project is designed to make a flowing water lamp, which are these actions: First turn LED #1 ON, then

turn it OFF. Then turn LED #2 ON, and then turn it OFF… and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the "movements" of flowing water.

Python Code 3.1.1 LightWater

First observe the project result, and then view the code.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 03.1.1_LightWater directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/03.1.1_LightWater
```

2. Use Python command to execute Python code "LightWater.py".

```
python LightWater.py
```

After the program is executed, you will see that LED Bar Graph starts with the flowing water way to be turned on from left to right, and then from right to left.

The following is the program code:

```python
1    from gpiozero import LEDBarGraph
2    from time import sleep
3
4    #ledPins = ["J8:11", "J8:12","J8:13","J8:15","J8:16","J8:18","J8:22","J8:3","J8:5","J8:24"]
5    ledPins = [17, 18, 27, 22, 23, 24, 25, 2, 3, 8]
6
7    leds = LEDBarGraph(*ledPins, active_high=False)
8
9    def loop():
10       while True:
11           for index in range(0,len(ledPins),1):       # make led(on) move from left to right
12               leds.on(index)
13               sleep(0.1)
14               leds.off(index)
15           for index in range(len(ledPins)-1,-1,-1):   #move led(on) from right to left
16               leds.on(index)
17               sleep(0.1)
18               leds.off(index)
19
20   if __name__ == '__main__':      # Program entrance
21       print ('Program is starting...')
22       try:
23           loop()
24       except KeyboardInterrupt:  # Press ctrl-c to end the program.
25           print("Ending program")
```

Import the LEDBarGraph class that controls LED Bar Graph from the gpiozero library.

```python
from gpiozero import LEDBarGraph
```

Create the LEDBarGraph class for controlling the LEDBarGraph.

```python
ledPins = [17, 18, 27, 22, 23, 24, 25, 2, 3, 8]

leds = LEDBarGraph(*ledPins, active_high=False)
```

The LED is turned on or off by specifying the index of the LED, if no parameter is specified, the same Settings

are applied to all leds.   leds.off() means that all leds are turned off.

```
        for index in range(0,len(ledPins),1):        # make led(on) move from left to right
            leds.on(index)
            sleep(0.1)
            leds.off(index)
        for index in range(len(ledPins)-1,-1,-1):    #move led(on) from right to left
            leds.on(index)
            sleep(0.1)
            leds.off(index)
```

In the program, first define 10 pins connected to the LED and set them to output mode. In the loop() function, two for loops are used to make the lights flow from right to left and from left to right.

```
def loop():
    while True:
        for index in range(0,len(ledPins),1):        # make led(on) move from left to right
            leds.on(index)
            sleep(0.1)
            leds.off(index)
        for index in range(len(ledPins)-1,-1,-1):    #move led(on) from right to left
            leds.on(index)
            sleep(0.1)
            leds.off(index)
```

For more information about the methods used by the LEDBoard class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_boards.html#ledboard
For more information about the methods used by the LEDBarGraph class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_boards.html#ledbargraph

In this experiment you can use the LEDBoard and LEDBarGraph classes to control the LEDBarGraph

Chapter 4 Analog & PWM

In previous chapters, we learned that a Push Button Switch has two states: Pressed (ON) and Released (OFF), and an LED has a Light ON and OFF state. Is there a middle or intermediated state? We will next learn how to create an intermediate output state to achieve a partially bright (dim) LED.

First, let us learn how to control the brightness of an LED.

Project 4.1 Breathing LED

We describe this project as a Breathing Light. This means that an LED that is OFF will then turn ON gradually and then gradually turn OFF like "breathing". Okay, so how do we control the brightness of an LED to create a Breathing Light? We will use PWM to achieve this goal.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1	Resistor 220Ω x1
Jumper Wire		

Component Knowledge

Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0℃ to 10℃. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.
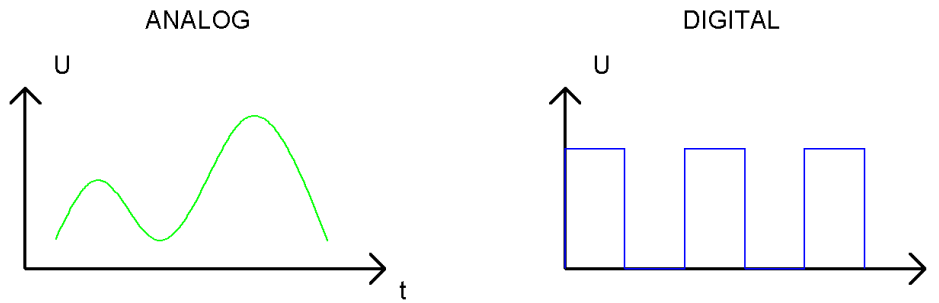
ANALOG   DIGITAL

Note that the Analog signals are curved waves and the Digital signals are "Square Waves".

In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Digital processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals).

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform. The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:
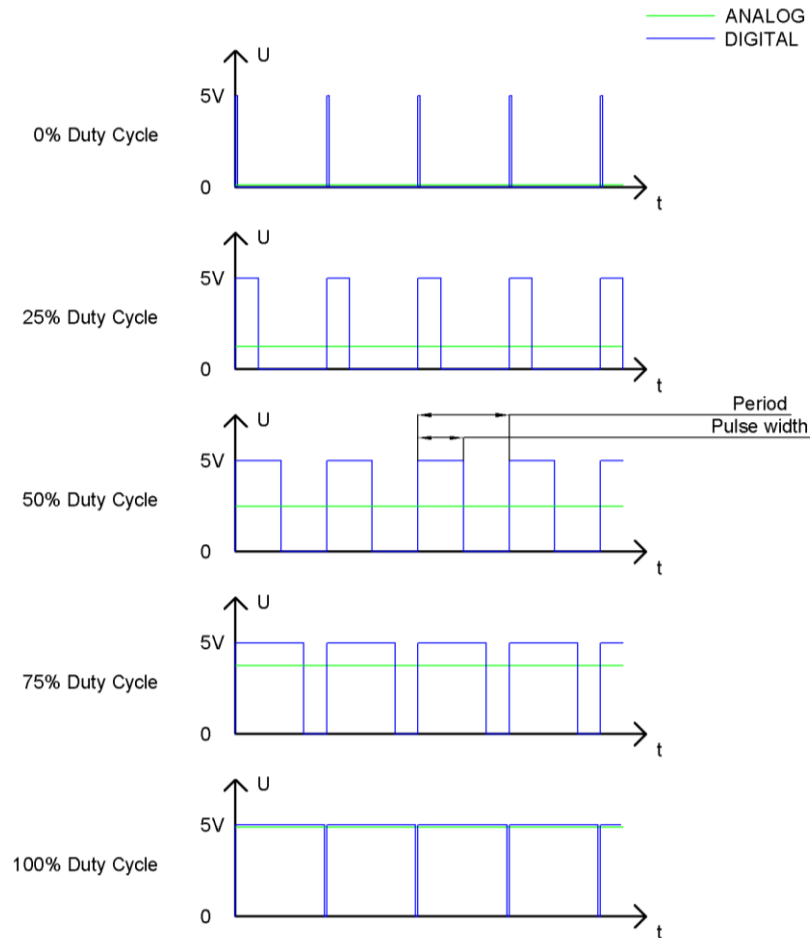
The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

It is evident, from the above, that PWM is not actually analog but the effective value of voltage is equivalent to the corresponding analog value. Therefore, by using PWM, we can control the output power of to an LED and control other devices and modules to achieve multiple effects and actions.

In RPi, GPIO18 pin has the ability to output to hardware via PWM with a 10-bit accuracy. This means that 100% of the pulse width can be divided into $2^{10}=1024$ equal parts.

The wiringPi library of C provides both a hardware PWM and a software PWM method, while the wiringPi library of Python does not provide a hardware PWM method. There is only a software PWM option for Python.

The hardware PWM only needs to be configured, does not require CPU resources and is more precise in time control. The software PWM requires the CPU to work continuously by using code to output high level and low level. This part of the code is carried out by multi-threading, and the accuracy is relatively not high enough.

In order to keep the results running consistently, we will use software PWM.

Circuit

Schematic diagram	Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
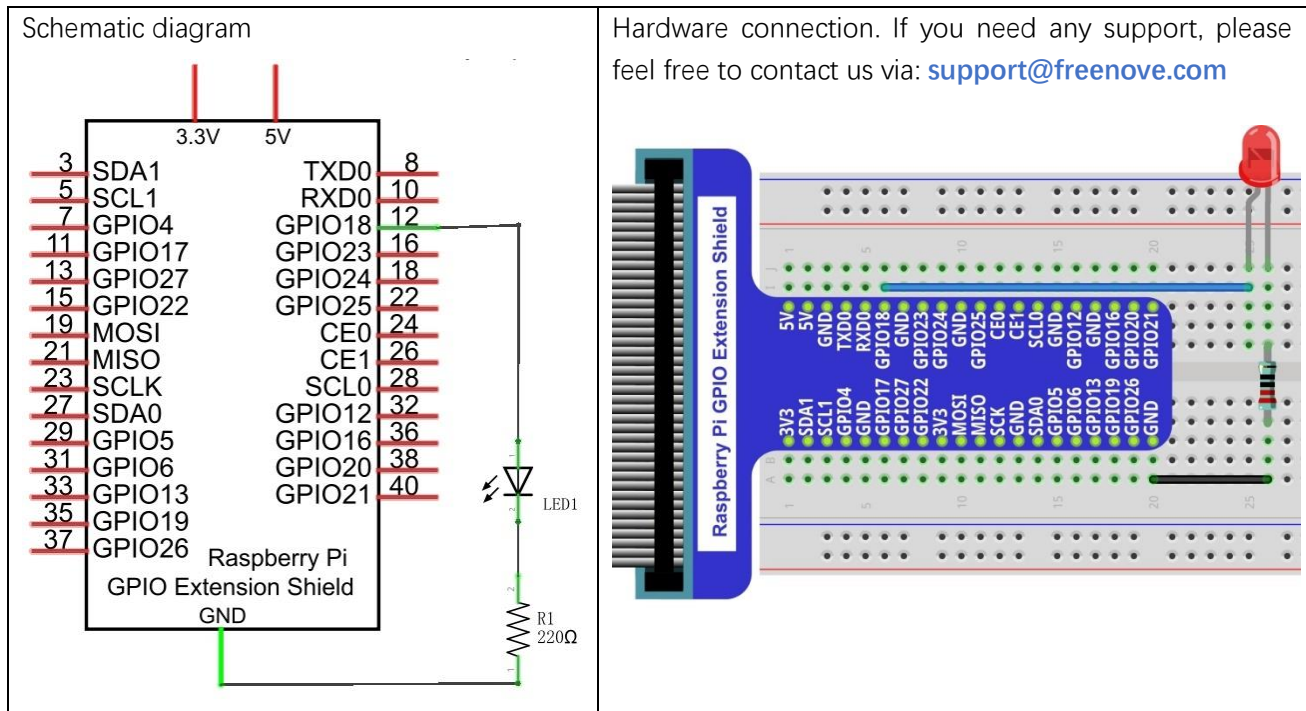
Code

This project uses the PWM output from the GPIO18 pin to make the pulse width gradually increase from 0% to 100% and then gradually decrease from 100% to 0% to make the LED glow brighter then dimmer.

Python Code 4.1.1 BreathingLED

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 04.1.1_BreathingLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/04.1.1_BreathingLED
```

2. Use the Python command to execute Python code "BreathingLED.py".

```
python BreathingLED.py
```

After the program is executed, you will see that the LED gradually turns ON and then gradually turns OFF similar to "breathing".

The following is the program code:

```python
1   from gpiozero import PWMLED
2   import time
3
4   led = PWMLED(18 ,initial_value=0 ,frequency=1000)
5   def loop():
6       while True:
7           for b in range(0, 101, 1):      # make the led brighter
8               led.value = b / 100.0       # set dc value as the duty cycle
```

```
9                 time.sleep(0.01)
10             time.sleep(1)
11             for b in range(100, -1, -1):  # make the led darker
12                 led.value = b / 100.0     # set dc value as the duty cycle
13                 time.sleep(0.01)
14             time.sleep(1)
15  def destroy():
16      led.close()
17  if __name__ == '__main__':      # Program entrance
18      print ('Program is starting ... ')
19      try:
20          loop()
21      except KeyboardInterrupt:  # Press ctrl-c to end the program.
22          destroy()
23          print("Ending program")
```

Import the PWMLED class that controls leds from the gpiozero library.

```
from gpiozero import PWMLED
```

Create the PWMLED class for controlling the LED.

```
led = PWMLED(18 , initial_value=0 , frequency=1000)
```

PWMLED is connected to GPIO18, and its PWM frequency is set to 1000HZ, and the initial duty cycle to 0%.

```
led = PWMLED(18 , initial_value=0 , frequency=1000) # Set the PWM frequency to 1000Hz and the
initial duty cycle to 0
```

There are two "for" loops used to control the breathing LED in the next endless "while" loop. The first loop outputs a power signal to the led PWM from 0% to 100% and the second loop outputs a power signal to the led PWM from 100% to 0%.

led.value represents:The duty cycle of the PWM device. 0.0 is off, 1.0 is fully on. led.value in between may be specified for varying levels of power in the device.

```
def loop():
    while True:
        for b in range(0, 101, 1):    # make the led brighter
            led.value = b / 100.0     # set dc value as the duty cycle
            time.sleep(0.01)
        time.sleep(1)
        for b in range(100, -1, -1):  # make the led darker
            led.value = b / 100.0     # set dc value as the duty cycle
            time.sleep(0.01)
        time.sleep(1)
```

For more information about the methods used by the PWMLED class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#pwmled
For more information about the methods used by the PWMOutputDevice class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_output.html#pwmoutputdevice

Chapter 5 RGB LED

In this chapter, we will learn how to control a RGB LED.

An RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Anode (+) or positive lead, the other 3 are the Cathodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Cathodes (2, 3 & 4) of the RGB LED
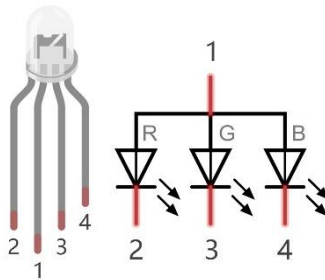
Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon.
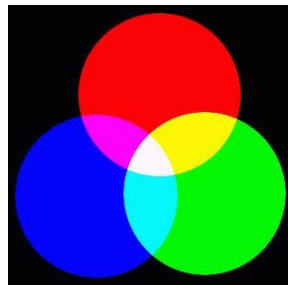
RGB

If we use a three 8 bit PWM to control the RGB LED, in theory, we can create $2^8 * 2^8 * 2^8 = 16777216$ (16 million) colors through different combinations of RGB light brightness.

Next, we will use RGB LED to make a multicolored LED.

Project 5.1 Multicolored LED

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	RGB LED x1	Resistor 220Ω x3
Jumper Wire		

Circuit
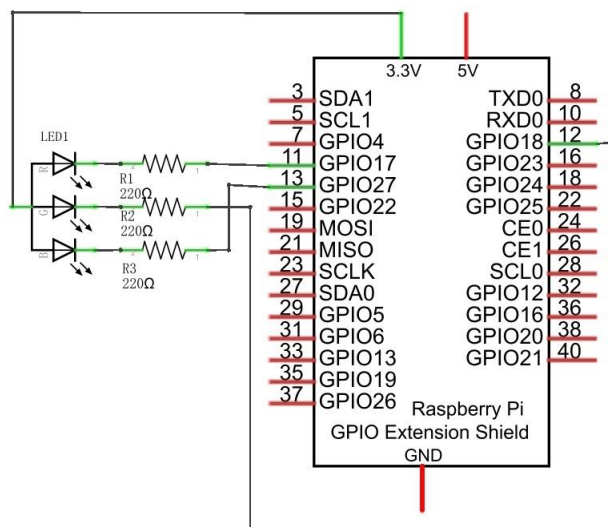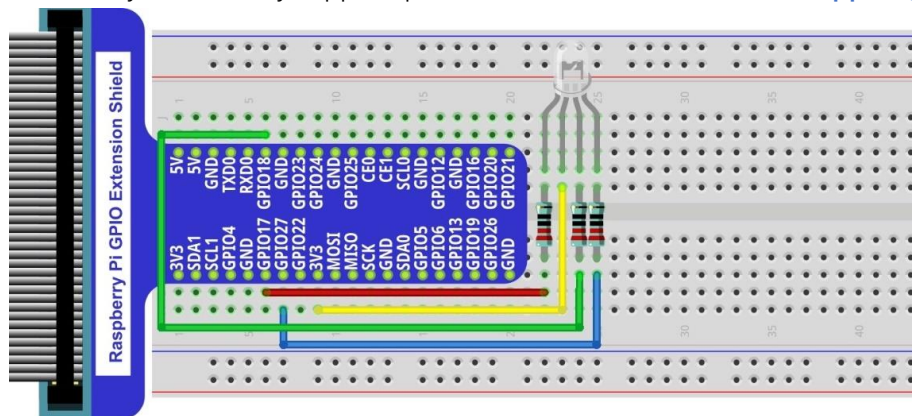
Schematic diagram
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**   In this kit, the RGB led is **Common anode**. The **voltage difference** between LED will make it work. There is no visible GND. The GPIO ports can also receive current while in output mode.

If circuit above doesn't work, the RGB LED may be common cathode. Please try following wiring.

There is no need to modify code for random color.
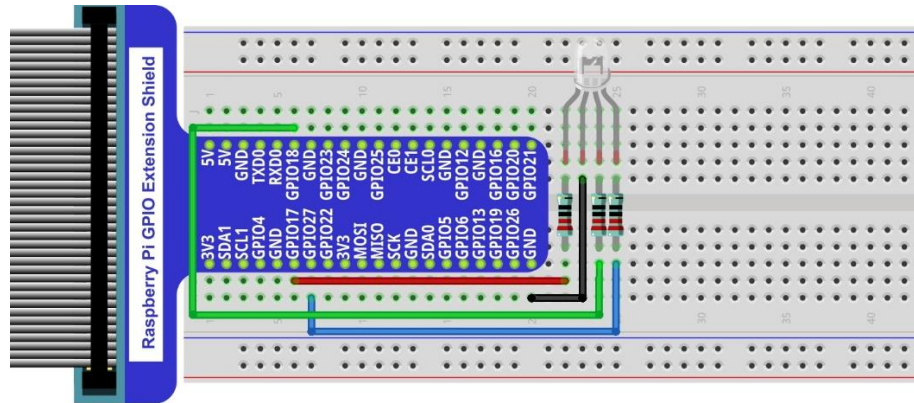
Code

We need to use RGBLED class to control RGBLED. The parameters for setting the RGBLED as common cathode or common anode are provided in the RGBLED class. You can set it according to the type of your RGB LED, and the default setting in our example code is based on common anode.

Python Code 5.1.1 ColorfulLED

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 05.1.1_ColorfulLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/05.1.1_ColorfulLED
```

2. Use python command to execute python code "ColorfulLED.py".

```
python ColorfulLED.py
```

After the program is executed, you will see that the RGB LED randomly lights up different colors.

The following is the program code:

```python
1   from gpiozero import RGBLED
2   import time
3   import random
4
5   #led = RGBLED(red="J8:11", green="J8:12", blue="J8:13" , active_high=False) # define the pins
    for R:11,G:12,B:13
6   led = RGBLED(red=17, green=18, blue=27, active_high=False) # define the pins for
    R:GPIO17,G:GPIO18,B:GPIO27
7   # If your RGBLED is a common cathode LED, set active_high to True
8
9   def setColor(r_val,g_val,b_val):        # change duty cycle for three pins to r_val,g_val,b_val
10      led.red=r_val/100                   # change pwmRed duty cycle to r_val
11      led.green = g_val/100               # change pwmRed duty cycle to r_val
12      led.blue = b_val/100                # change pwmRed duty cycle to r_val
13
14  def loop():
```

```
15      while True :
16          r=random.randint(0,100)   #get a random in (0,100)
17          g=random.randint(0,100)
18          b=random.randint(0,100)
19          setColor(r,g,b)           #set random as a duty cycle value
20          print ('r=%d, g=%d, b=%d ' %(r ,g, b))
21          time.sleep(1)
22
23  def destroy():
24      led.close()
25
26  if __name__ == '__main__':      # Program entrance
27      print ('Program is starting ... ')
28      try:
29          loop()
30      except KeyboardInterrupt:  # Press ctrl-c to end the program.
31          destroy()
32          print("Ending program")
```

Import the RGBLED class that controls RGBLED from the gpiozero library.

```
from gpiozero import RGBLED
```

Create the RGBLED class for controlling the RGBLED.

```
led = RGBLED(red=17, green=18, blue=27, active_high=False) # define the pins for
R:GPIO17,G:GPIO18,B:GPIO27
```

In the previous chapter, we learned how to make a pin output PWM using the Python language. In this project, we output to three pins via PWM. In the "while" loop of the "loop" function, we first generate three random numbers and then assign these three random numbers to the PWM values of the three pins, which will make the RGB LED randomly produce multiple colors.

```
def loop():
    while True :
        r=random.randint(0,100)   #get a random in (0,100)
        g=random.randint(0,100)
        b=random.randint(0,100)
        setColor(r,g,b)           #set random as a duty cycle value
        print ('r=%d, g=%d, b=%d ' %(r ,g, b))
        time.sleep(1)
```

For more information about the methods used by the RGBLED class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#rgbled

Chapter 6 Buzzer

In this chapter, we will learn about buzzers and the sounds they make. And in our next project, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

Project 6.1 Doorbell

We will make a doorbell with this functionality: when the Push Button Switch is pressed the buzzer sounds and when the button is released, the buzzer stops. This is a momentary switch function.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire		
NPN transistorx1 (S8050)	Active buzzer x1	Push Button Switch x1	Resistor 1kΩ x1	Resistor 10kΩ x2

Component knowledge

Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.
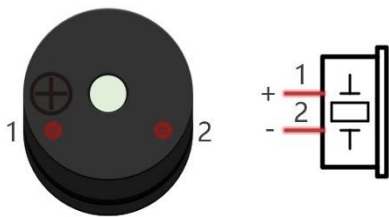
Active buzzer   Passive buzzer
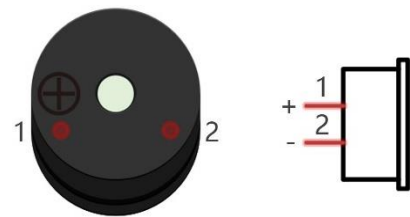
Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

How to identify active and passive buzzer?
1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes, view of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model.

Active buzzer bottom   Passive buzzer bottom

Transistors

A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to

amplify the current.

Transistors, full name: semiconductor transistor, is a semiconductor device that controls current (think of a transistor as an electronic "amplifying or switching device". Transistors can be used to amplify weak signals, or to work as a switch. Transistors have three electrodes (PINs): base (b), collector (c) and emitter (e). When there is current passing between "be" then "ce" will have a several-fold current increase (transistor magnification), in this configuration the transistor acts as an amplifier. When current produced by "be" exceeds a certain value, "ce" will limit the current output. at this point the transistor is working in its saturation region and acts like a switch. Transistors are available as two types as shown below: PNP and NPN,
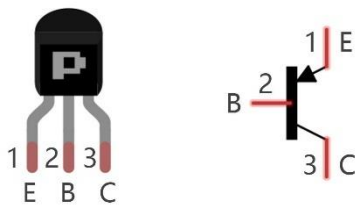
PNP transistor   NPN transistor
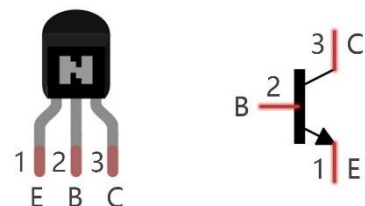
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Thanks to the transistor's characteristics, they are often used as switches in digital circuits. As micro-controllers output current capacity is very weak, we will use a transistor to amplify its current in order to drive components requiring higher current.

When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not conduct currentand buzzer will remain silent (no sounds).

When we use a PNP transistor to drive a buzzer, we often use the following method. If GPIO outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If GPIO outputs high level, no current flows through R1, the transistor will not conduct current and buzzer will remain silent (no sounds). Below are the circuit schematics for both a NPN and PNP transistor to power a buzzer.
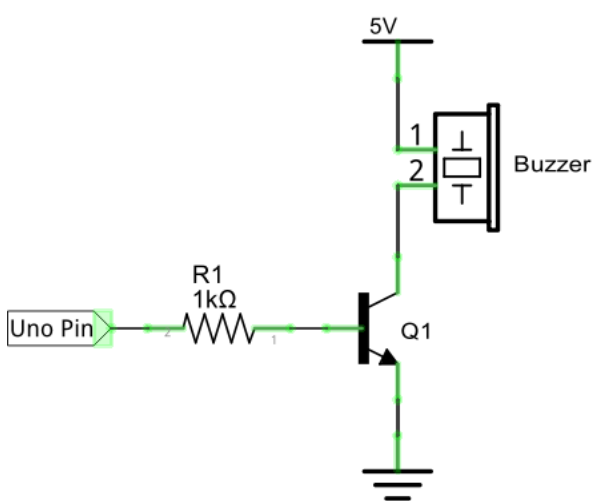
NPN transistor to drive buzzer   PNP transistor to drive buzzer
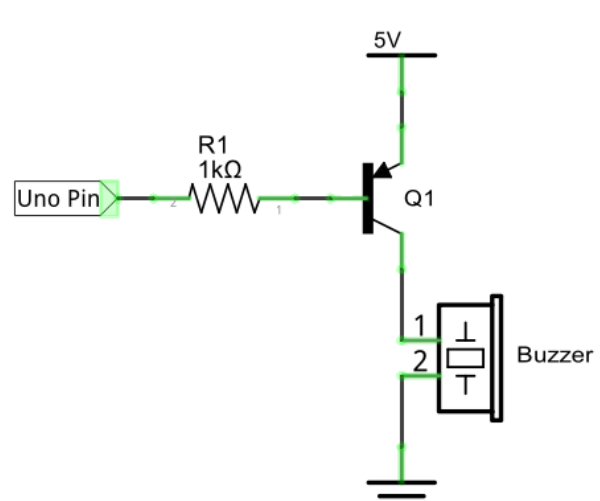
Circuit

Schematic diagram with RPi GPIO Extension Shield.
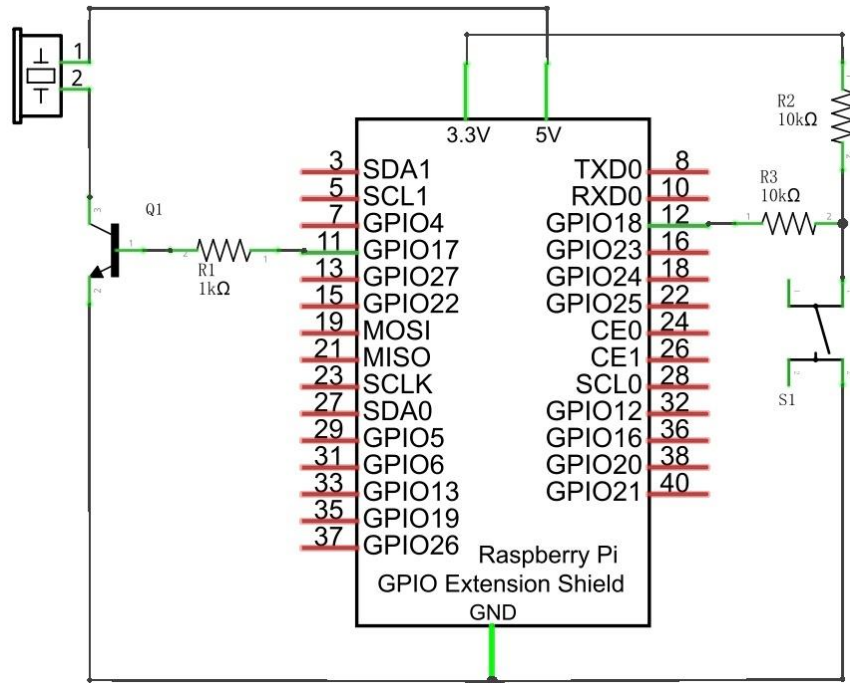
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
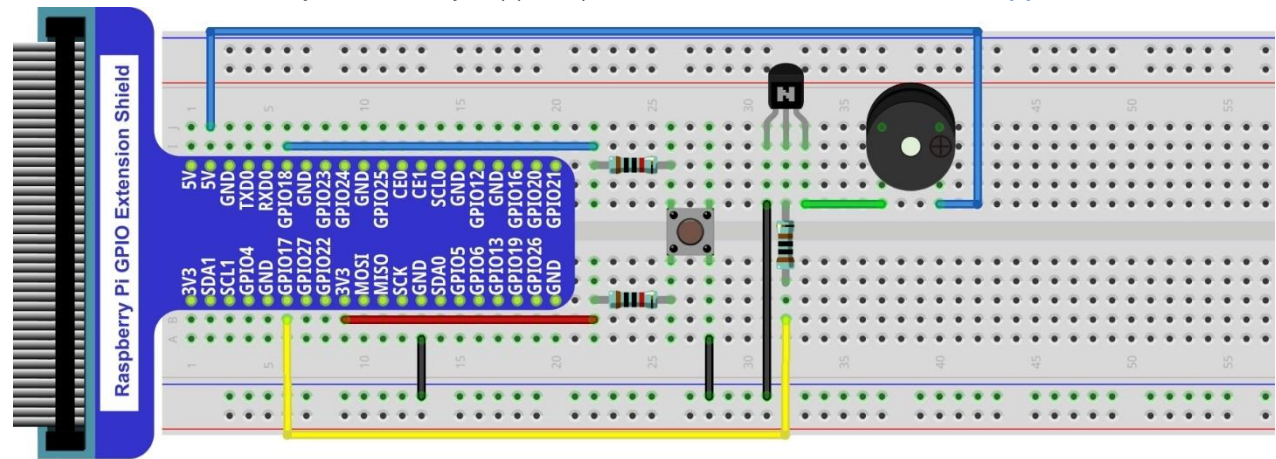
Note: in this circuit, the power supply for the buzzer is 5V, and pull-up resistor of the push button switch is connected to the 3.3V power feed. Actually, the buzzer can work when connected to the 3.3V power feed but this will produce a weak sound from the buzzer (not very loud).

Code

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

Python Code 6.1.1 Doorbell

First, observe the project result, then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 06.1.1_Doorbell directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/06.1.1_Doorbell
```

2. Use python command to execute python code "Doorbell.py".

```
python Doorbell.py
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.

The following is the program code:

```python
1   from gpiozero import Buzzer, Button
2   import time
3
4   buzzer = Buzzer(17)
5   button = Button(18)
6
7   def onButtonPressed():
8       buzzer.on()
9       print("Button is pressed, buzzer turned on >>>")
10
11  def onButtonReleased():
12      buzzer.off()
13      print("Button is released, buzzer turned on <<<")
14
15  def loop():
16      button.when_pressed = onButtonPressed
17      button.when_released = onButtonReleased
18      while True :
19          time.sleep(1)
20
21  def destroy():
22      buzzer.close()
23      button.close()
24
25  if __name__ == '__main__':      # Program entrance
26      print ('Program is starting ... ')
27      try:
28          loop()
29      except KeyboardInterrupt:  # Press ctrl-c to end the program.
30          destroy()
31          print("Ending program")
```

The code is exactly the same as when we used a push button switch to control an LED. You can also try using

the PNP transistor to achieve the same results.

Import the Buzzer class that controls Buzzer from the gpiozero library.

```
from gpiozero import Buzzer, Button
```

Create the Buzzer class for controlling the Buzzer.

```
buzzer = Buzzer(17)
```

In GPIO Zero, you assign the when_pressed and when_released properties to set up callbacks on those actions.

Once it detects that the button is pressed, it executes the specified function onButtonPressed(). Once it detects that the button is released, it executes the specified function onButtonReleased()

```
def loop():
    button.when_pressed = onButtonPressed
    button.when_released = onButtonReleased
```

For more information about the methods used by the Buzzer class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#buzzer

Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.

The list of components and the circuit is similar to the doorbell project. We only need to take the Doorbell circuit and replace the active buzzer with a passive buzzer.

Code

In this project, our buzzer alarm is controlled by the push button switch. Press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.

As stated before, it is analogous to our earlier project that controlled an LED ON and OFF.

To control a passive buzzer requires PWM of certain sound frequency.

Python Code 6.2.1 Alertor

First observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 06.2.1_Alertor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/06.2.1_Alertor
```

2. Use the python command to execute the Python code "Alertor.py".

```
python Alertor.py
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.

The following is the program code:

```python
1   from gpiozero import TonalBuzzer,Button
2   from gpiozero.tones import Tone
3   import time
4   import math
5
6   buzzer = TonalBuzzer(17)
7   button = Button(18) # define Button pin according to BCM Numbering
8
9   def loop():
10      while True:
11          if button.is_pressed:  # if button is pressed
12              alertor()
13              print ('alertor turned on >>> ')
14          else :
15              stopAlertor()
16              print ('alertor turned off <<<')
17  def alertor():
18          buzzer.play(Tone(220.0))
19          time.sleep(0.1)
```

```
20
21    def stopAlertor():
22        buzzer.stop()
23
24    def destroy():
25        buzzer.close()
26
27    if __name__ == '__main__':      # Program entrance
28        print ('Program is starting...')
29        try:
30            loop()
31        except KeyboardInterrupt:   # Press ctrl-c to end the program.
32            destroy()
33            print("Ending program")
```

Define GPIO17 as the buzzer control pin, and GPIO18 as the button control pin to control the passive buzzer. It requires a certain frequency of PWM to control a passive buzzer, so the TonalBuzzer class is needed.

```
buzzer = TonalBuzzer(17)
button = Button(18) # define Button pin according to BCM Numbering
```

In the while loop loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound.

```
def alertor():
        buzzer.play(Tone(220.0))
        time.sleep(0.1)
```

When the push button switch is released, the buzzer (in this case our Alarm) will stop.

```
def stopAlertor():
        buzzer.stop()
```

For more information about the methods used by the TonalBuzzer class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_output.html#tonalbuzzer
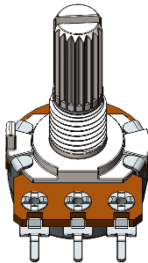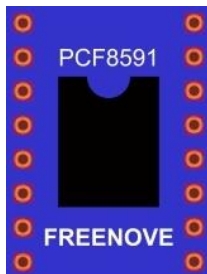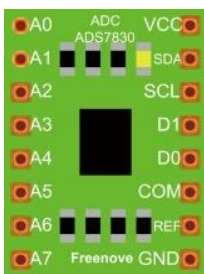
(**Important**) Chapter 7 ADC

We have learned how to control the brightness of an LED through PWM and that PWM is not a real analog signal. In this chapter, we will learn how to read analog values via an ADC Module and convert these analog values into digital.
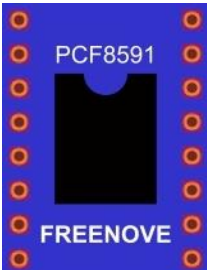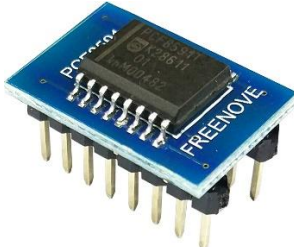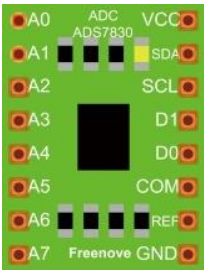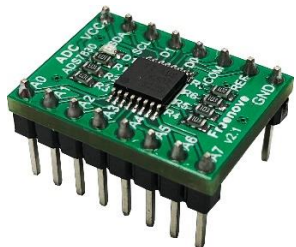
Project 7.1 Read the Voltage of Potentiometer

In this project, we will use the ADC function of an ADC Module to read the voltage value of a potentiometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire M/M x16	
Rotary potentiometer x1	ADC module x1		Resistor 10kΩ x2

This product contains **only one ADC module**, there are two types, PCF8591 and ADS7830. For the projects described in this tutorial, they function the same. Please build corresponding circuits according to the ADC module found in your Kit.

ADC module: PCF8591		ADC module: ADS7830	
Model diagram	Actual Picture	Model diagram	Actual Picture

Circuit knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC module is 8 bits, that means the resolution is 2^8=256, so that its range (at 3.3V) will be divided equally to 256 parts.

Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.
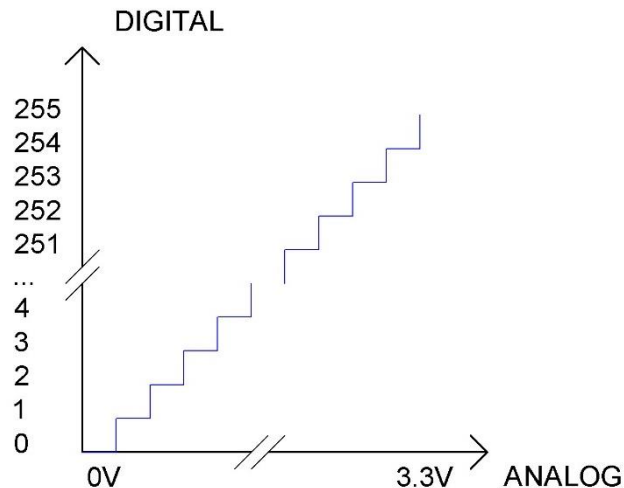
Subsection 1: the analog in range of 0V-3.3/256 V corresponds to digital 0;
Subsection 2: the analog in range of 3.3 /256 V-2*3.3 /256V corresponds to digital 1;
...

The resultant analog signal will be divided accordingly.

DAC

The reversing this process requires a DAC, Digital-to-Analog Converter. The digital I/O port can output high level and low level (0 or 1), but cannot output an intermediate voltage value. This is where a DAC is useful. The DAC module PCF8591 has a DAC output pin with 8-bit accuracy, which can divide VDD (here is 3.3V) into 2^8=256 parts. For example, when the digital quantity is 1, the output voltage value is 3.3/256 *1 V, and when the digital quantity is 128, the output voltage value is 3.3/256 *128=1.65V, the higher the accuracy of DAC, the higher the accuracy of output voltage value will be.

Component knowledge

Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.
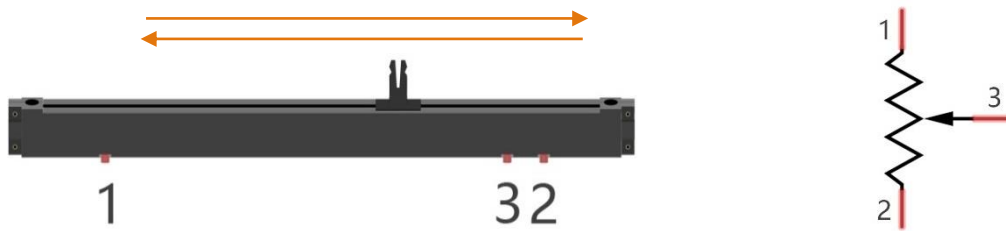
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.
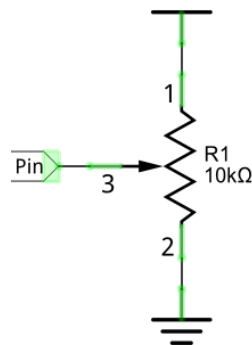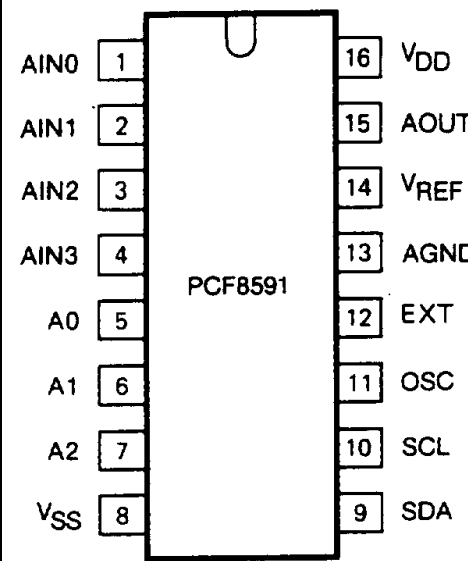
Rotary potentiometer

Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.
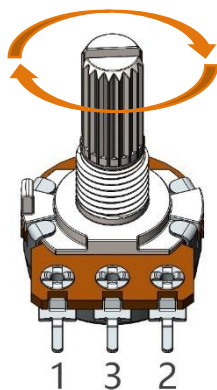
PCF8591
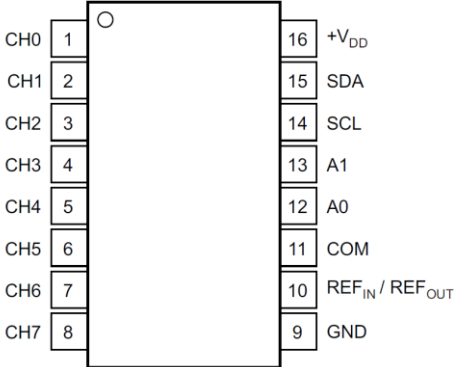
The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. The following table is the pin definition diagram of PCF8591.

SYMBOL	PIN	DESCRIPTION	TOP VIEW
AIN0	1	Analog inputs (A/D converter)	
AIN1	2		
AIN2	3		
AIN3	4		
A0	5	Hardware address	
A1	6		
A2	7		
Vss	8	Negative supply voltage	
SDA	9	I2C-bus data input/output	
SCL	10	I2C-bus clock input	
OSC	11	Oscillator input/output	
EXT	12	external/internal switch for oscillator input	
AGND	13	Analog ground	
Vref	14	Voltage reference input	
AOUT	15	Analog output(D/A converter)	
Vdd	16	Positive supply voltage	

For more details about PCF8591, please refer to the datasheet which can be found on the Internet.

ADS7830

The ADS7830 is a single-supply, low-power, 8-bit data acquisition device that features a serial I2C interface and an 8-channel multiplexer. The following table is the pin definition diagram of ADS7830.

SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		
CH5	6		
CH6	7		
CH7	8		
GND	9	Ground	
REF in/out	10	Internal +2.5V Reference, External Reference Input	
COM	11	Common to Analog Input Channel	
A0	12	Hardware address	
A1	13		
SCL	14	Serial Clock	
SDA	15	Serial Sata	
+VDD	16	Power Supply, 3.3V Nominal	

I2C communication

I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique address which can be used as a transmitter or receiver to communicate with devices connected via the bus.

Circuit with ADS7830
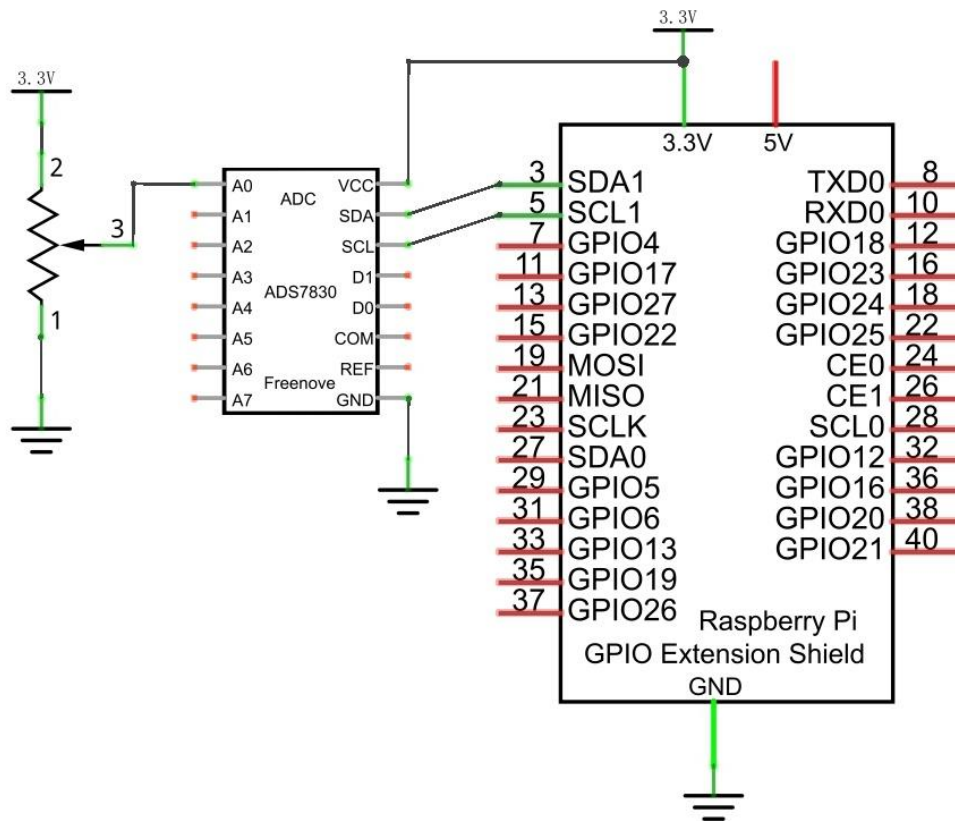
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
This product contains **only one ADC module.**
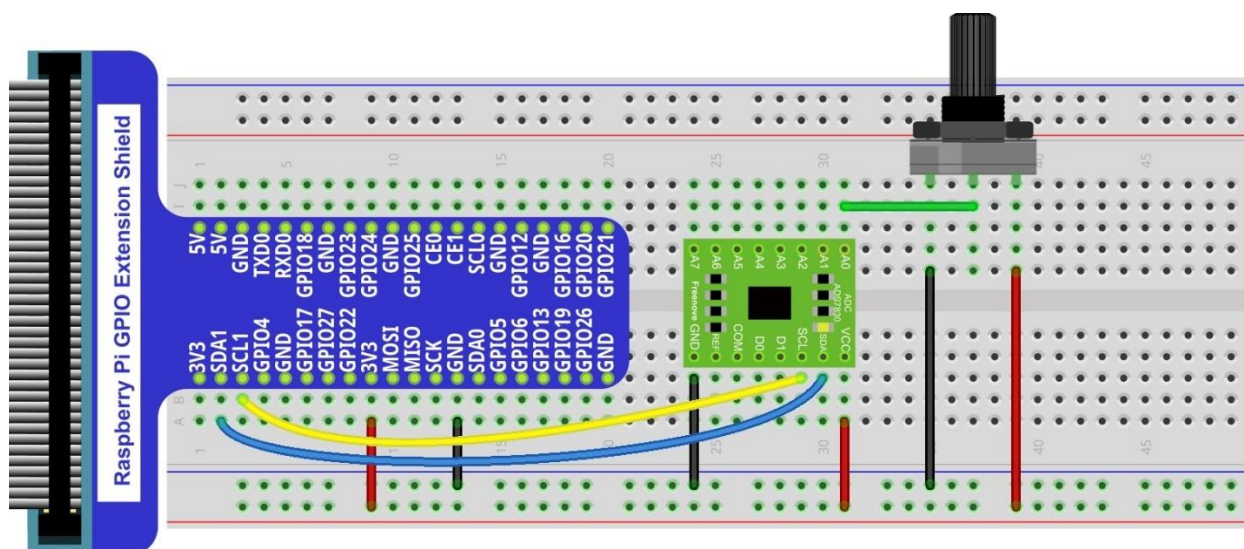
Circuit with PCF8591

Schematic diagram
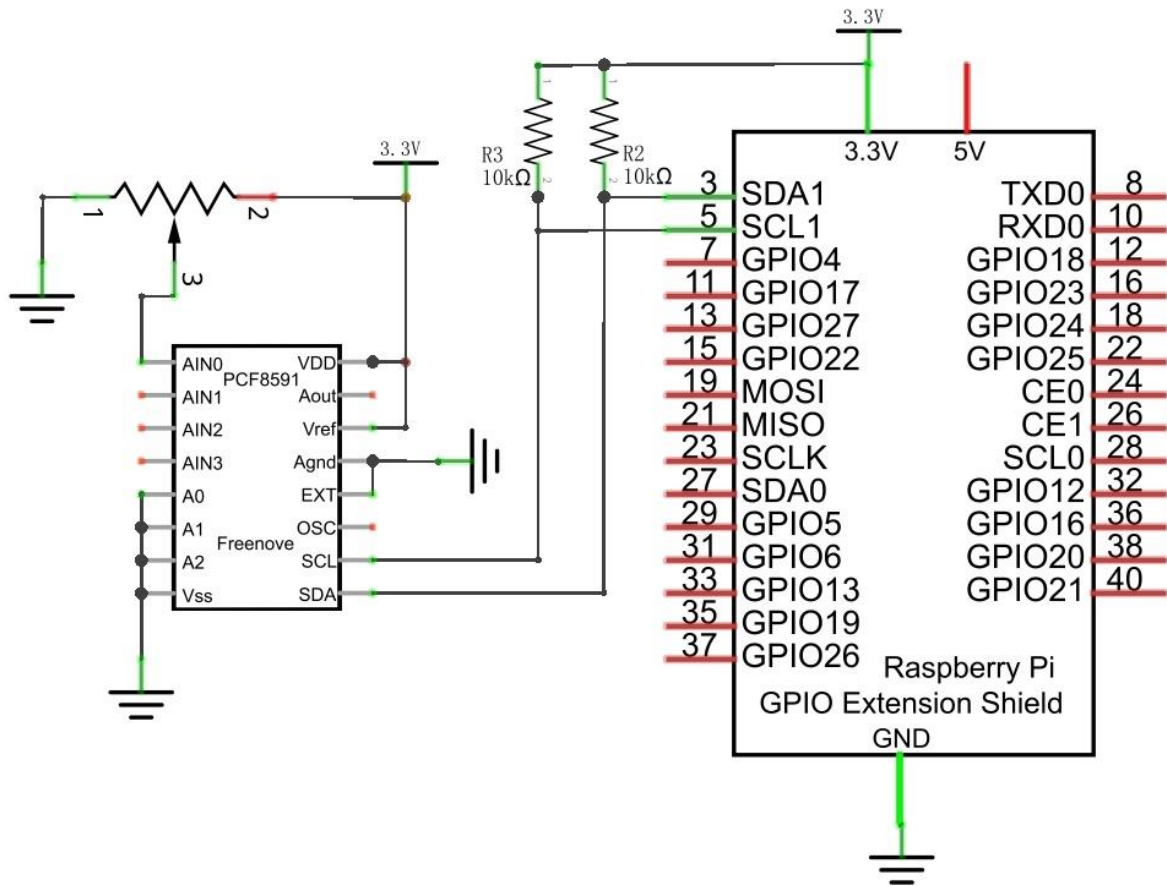
Hardware connection
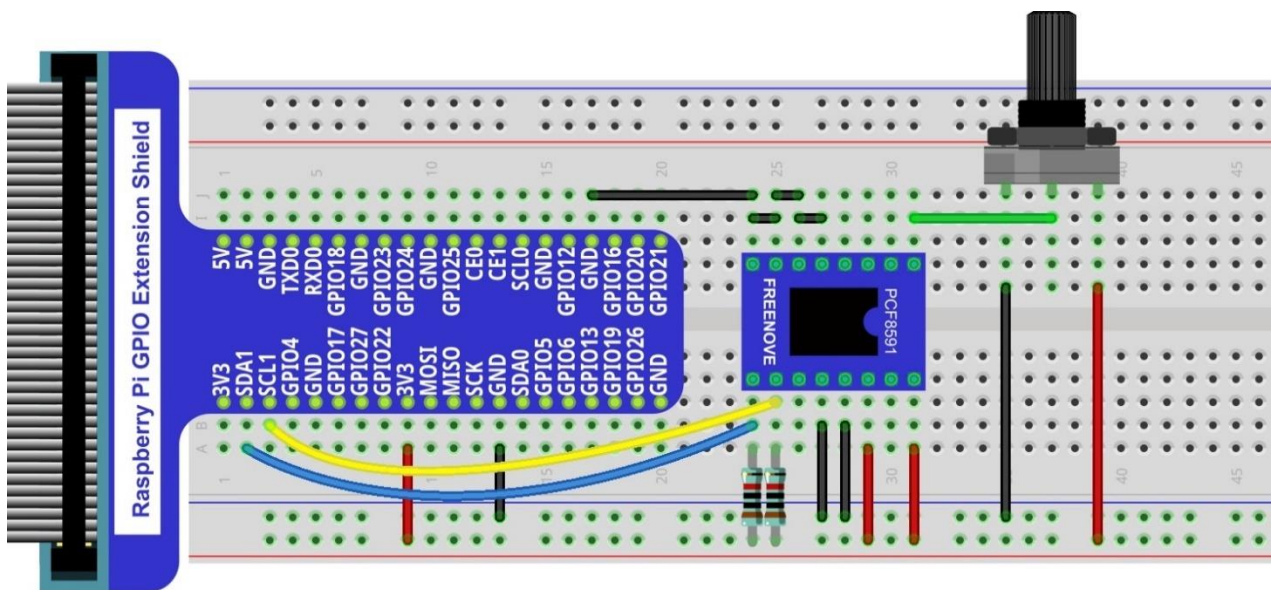
Please keep the **chip mark** consistent to make the chips under right direction and position.

Configure I2C and Install Smbus

Enable I2C

The I2C interface in Raspberry Pi is disabled by default. You will need to open it manually and enable the I2C interface as follows:

Type command in the Terminal:

```
sudo raspi-config
```

Then open the following dialog box:

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

  1 Change User Password          Change password for the current u
  2 Network Options               Configure network settings
  3 Boot Options                  Configure options for start-up
  4 Localisation Options          Set up language and regional sett
  5 Interfacing Options           Configure connections to peripher
  6 Overclock                     Configure overclocking for your P
  7 Advanced Options              Configure advanced settings
  8 Update                        Update this tool to the latest ve
  9 About raspi-config            Information about this configurat



                  <Select>                        <Finish>
```

Choose "5 Interfacing Options" then "P5 I2C" then "Yes" and then "Finish" in this order and restart your RPi. The I2C module will then be started.

Type a command to check whether the I2C module is started:

```
lsmod | grep i2c
```

If the I2C module has been started, the following content will be shown. "bcm2708" refers to the CPU model. Different models of Raspberry Pi display different contents depending on the CPU installed:

```
pi@raspberrypi:~ $ lsmod | grep i2c
i2c_bcm2708            4770  0
i2c_dev                5859  0
pi@raspberrypi:~ $
```

Install I2C-Tools

Next, type the command to install I2C-Tools. It is available with the Raspberry Pi OS by default.

```
sudo apt-get install i2c-tools
```

I2C device address detection:

```
i2cdetect -y 1
```

When you are using the PCF8591 Module, the result should look like this:

Here, 48 (HEX) is the I2C address of ADC Module (PCF8591).

When you are using ADS, the result should look like this:

Here, 4b (HEX) is the I2C address of ADC Module (ADS7830).

Install Smbus Module

```
sudo apt-get install python-smbus
sudo apt-get install python3-smbus
```

Code

Python Code 7.1.1 ADC

For Python code, ADCDevice requires a custom module which needs to be installed.

1. Use cd command to enter folder of ADCDevice.

```
cd ~/Freenove_Kit/Libs/Python-Libs/
```

2. Unzip the file.

```
tar zxvf ADCDevice-1.0.3.tar.gz
```

3. Open the unzipped folder.

```
cd ADCDevice-1.0.3
```

4.   Install library for python3 and python2.

```
sudo python3 setup.py install
sudo python2 setup.py install
```

A successful installation, without error prompts, is shown below:

```
Installed /usr/local/lib/python3.7/dist-packages/ADCDevice-1.0.2-py3.7.egg
Processing dependencies for ADCDevice==1.0.2
Finished processing dependencies for ADCDevice==1.0.2
```

Execute the following command. Observe the project result and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 07.1.1_ADC directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/07.1.1_ADC
```

2. Use the Python command to execute the Python code "ADC.py".

```
python ADC.py
```

After the program is executed, adjusting the potentiometer will produce a readout display of the potentiometer voltage values in the Terminal and the converted digital content.

```
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 169, Voltage : 2.19
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
```

The following is the code:

```python
1   import time
2   from ADCDevice import *
3
4   adc = ADCDevice() # Define an ADCDevice class object
5
6   def setup():
7       global adc
8       if(adc.detectI2C(0x48)): # Detect the pcf8591.
9           adc = PCF8591()
10      elif(adc.detectI2C(0x4b)): # Detect the ads7830
11          adc = ADS7830()
12      else:
13          print("No correct I2C address found, \n"
14          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
15          "Program Exit. \n");
```

```
16              exit(-1)
17
18  def loop():
19      while True:
20          value = adc.analogRead(0)    # read the ADC value of channel 0
21          voltage = value / 255.0 * 3.3  # calculate the voltage value
22          print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
23          time.sleep(0.1)
24
25  def destroy():
26      adc.close()
27
28  if __name__ == '__main__':   # Program entrance
29      print ('Program is starting ... ')
30      try:
31          setup()
32          loop()
33      except KeyboardInterrupt: # Press ctrl-c to end the program.
34          destroy()
```

In this code, a custom Python module "ADCDevice" is used. It contains the method of utilizing the ADC Module in this project, through which the ADC Module can easily and quickly be used. In the code, you need to first create an ADCDevice object adc.

```
adc = ADCDevice() # Define an ADCDevice class object
```

Then in setup(), use detecticIC(addr), the member function of ADCDevice, to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the address, we can determine which ADC Module is in the circuit. When the correct module is detected, a device specific class object is created and assigned to adc. The default address of PCF8591 is 0x48, and that of ADS7830 is 0x4b.

```
def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = PCF8591()
    elif(adc.detectI2C(0x4b)): # Detect the ads7830
        adc = ADS7830()
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
```

When you have a class object of a specific device, you can get the ADC value of the specified channel by calling the member function of this class, analogRead(chn). In loop(), get the ADC value of potentiometer.

```
value = adc.analogRead(0)    # read the ADC value of channel 0
```

Then according to the formula, the voltage value is calculated and displayed on the terminal monitor.

```
        voltage = value / 255.0 * 3.3  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.1)
```

Reference

About smbus Module:

smbus Module
The System Management Bus Module defines an object type that allows SMBus transactions on hosts running the Linux kernel. The host kernel must support I2C, I2C device interface support, and a bus adapter driver. All of these can be either built-in to the kernel, or loaded from modules. In Python, you can use help(smbus) to view the relevant functions and their descriptions. **bus=smbus.SMBus(1)**: Create an SMBus class object. **bus.read_byte_data(address,cmd+chn)**:   Read a byte of data from an address and return it. **bus.write_byte_data(address,cmd,value)**:   Write a byte of data to an address.
`class ADCDevice(object)`
This is a base class. `int detectI2C(int addr);` This is a member function, which is used to detect whether the device with the given I2C address exists. If it exists, it returns true. Otherwise, it returns false.
`class PCF8591(ADCDevice)` `class ADS7830(ADCDevice)`
These two classes are derived from the ADCDevice and the main function is analogRead(chn). `int analogRead(int chn);` This returns the value read on the supplied analog input pin. Parameter chn: For PCF8591, the range of chn is 0, 1, 2, 3. For ADS7830, the range is 0, 1, 2, 3, 4, 5, 6, 7.

You can find the source file of this library in the folder below:

`~/Freenove_Kit/Libs/Python-Libs/ADCDevice-1.0.3/src/ADCDevice/ADCdevice.py`
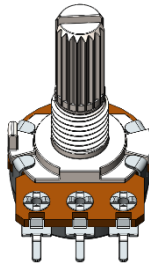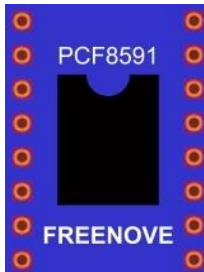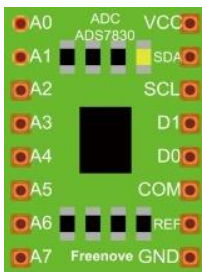
Chapter 8 Potentiometer & LED

Earlier we learned how to use ADC and PWM. In this chapter, we learn to control the brightness of an LED by using a potentiometer.

Project 8.1 Soft Light

In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle ratio of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire M/M x17			
Rotary Potentiometer x1	ADC Module x1 （**Only one**）		10kΩ x2	220Ω x1	LED x1

Circuit with ADS7830
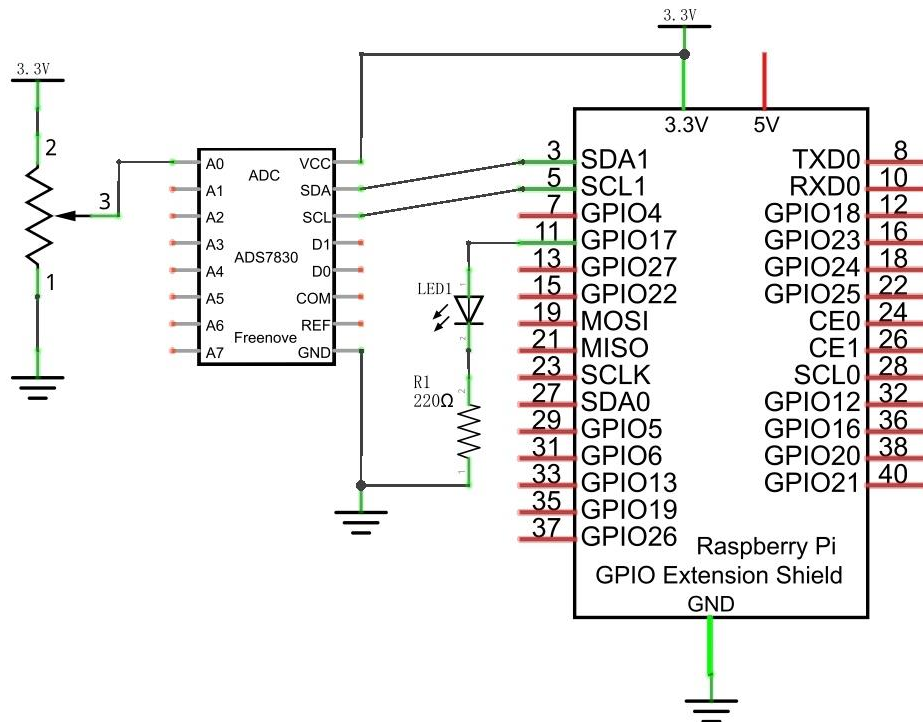
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
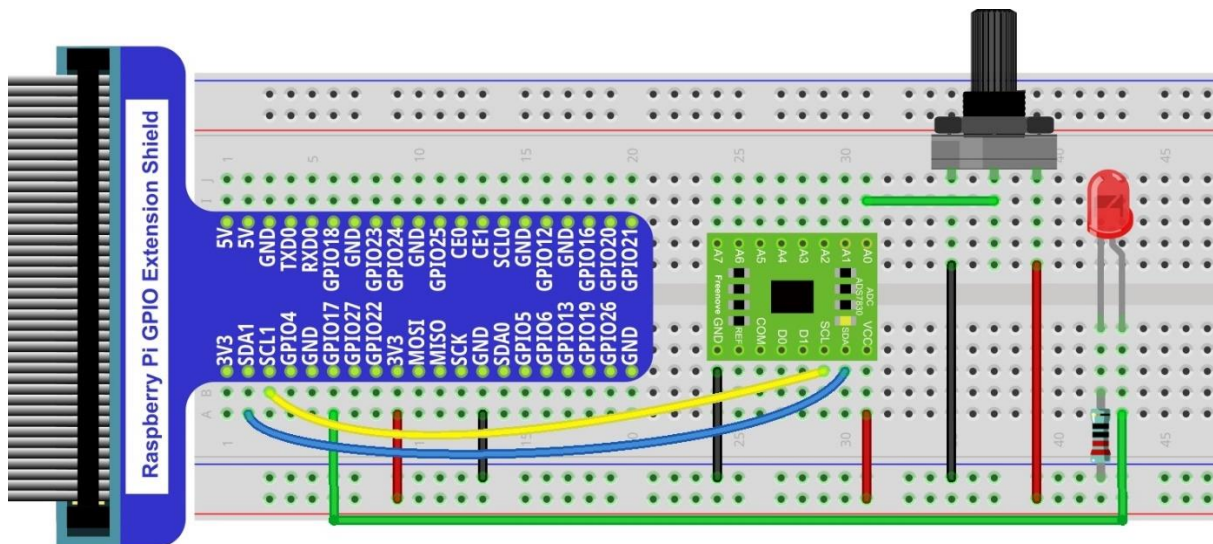
Circuit with PCF8591

Schematic diagram
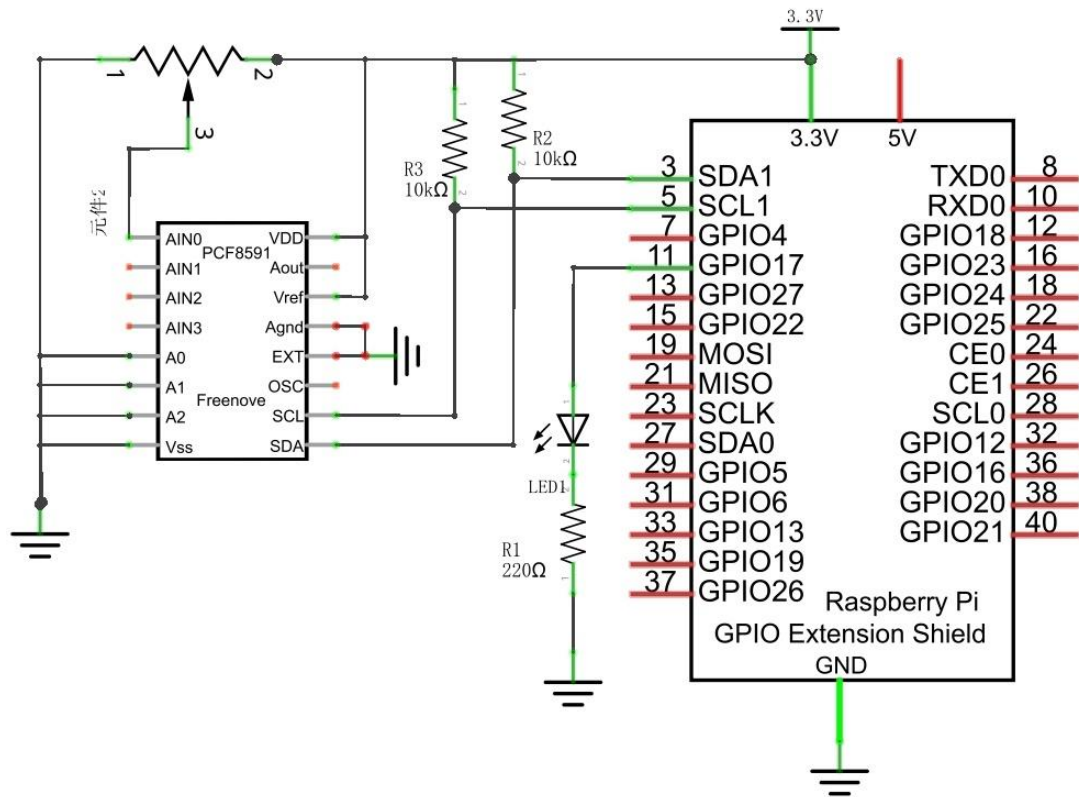
Hardware connection
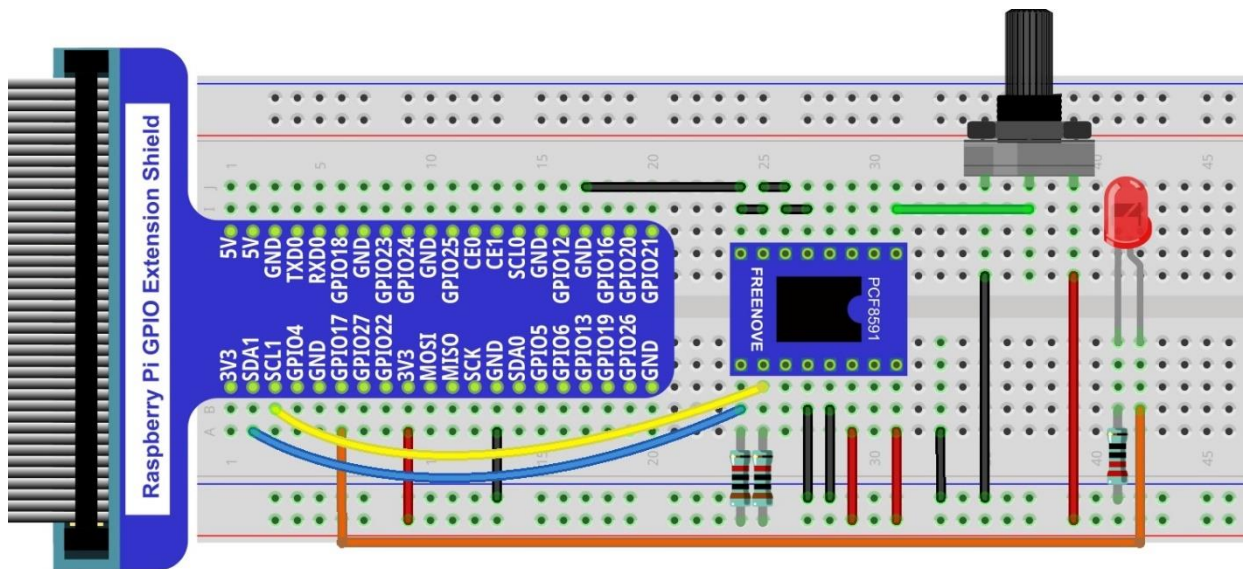
Code

Python Code 8.1.1 Softlight

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.
First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1.   Use cd command to enter 08.1.1_Softlight directory of Python code

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/08.1.1_Softlight
```

2.   Use the python command to execute the Python code "Softlight.py".

```
python Softlight.py
```

After the program is executed, adjusting the potentiometer will display the voltage values of the potentiometer in the Terminal window and the converted digital quantity. As a consequence, the brightness of LED will be changed.

The following is the code:

```
1    from gpiozero import PWMLED
2    import time
3    from ADCDevice import *
4
5    led = PWMLED(17,frequency=1000)        # define LED pin according to BCM Numbering
6    adc = ADCDevice() # Define an ADCDevice class object
7
8    def setup():
9        global adc
10       if(adc.detectI2C(0x48)): # Detect the pcf8591.
11           adc = PCF8591()
12       elif(adc.detectI2C(0x4b)): # Detect the ads7830
13           adc = ADS7830()
14       else:
15           print("No correct I2C address found, \n"
16           "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
17           "Program Exit. \n");
18           exit(-1)
19
20   def loop():
21       while True:
22           value = adc.analogRead(0)        # read the ADC value of channel 0
23           led.value = value / 255.0        # Mapping to PWM duty cycle
24           voltage = value / 255.0 * 3.3   # calculate the voltage value
25           print('ADC Value : %d, Voltage : %.2f'%(value,voltage))
26           time.sleep(0.03)
```

```
27
28   def destroy():
29       led.close()
30       adc.close()
31
32   if __name__ == '__main__':   # Program entrance
33       print ('Program is starting ... ')
34       try:
35           setup()
36           loop()
37       except KeyboardInterrupt: # Press ctrl-c to end the program.
38           destroy()
39           print("Ending program")
```

In the code, read ADC value of potentiometers and map it to the duty cycle of the PWM to control LED brightness.

```
value = adc.analogRead(0)       # read the ADC value of channel 0
led.value = value / 255.0       # Mapping to PWM duty cycle
```
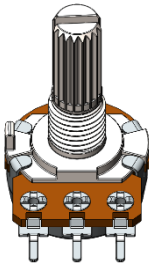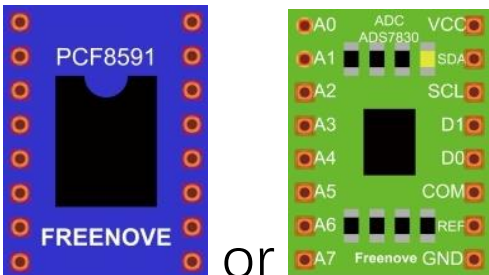
Chapter 9 Potentiometer & RGBLED

In this chapter, we will use 3 potentiometers to control the brightness of 3 LEDs of RGBLED to create multiple colors.

Project 9.1 Colorful Light

In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as with the Soft Light. project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the previous soft light project needed only one LED while this one required (1) RGB LEDs.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1			Jumper Wires M/M x17			
Rotary potentiometer x3	ADC module x1			10kΩ x2	220Ω x3	RGB LEDx1

Circuit with ADS7830
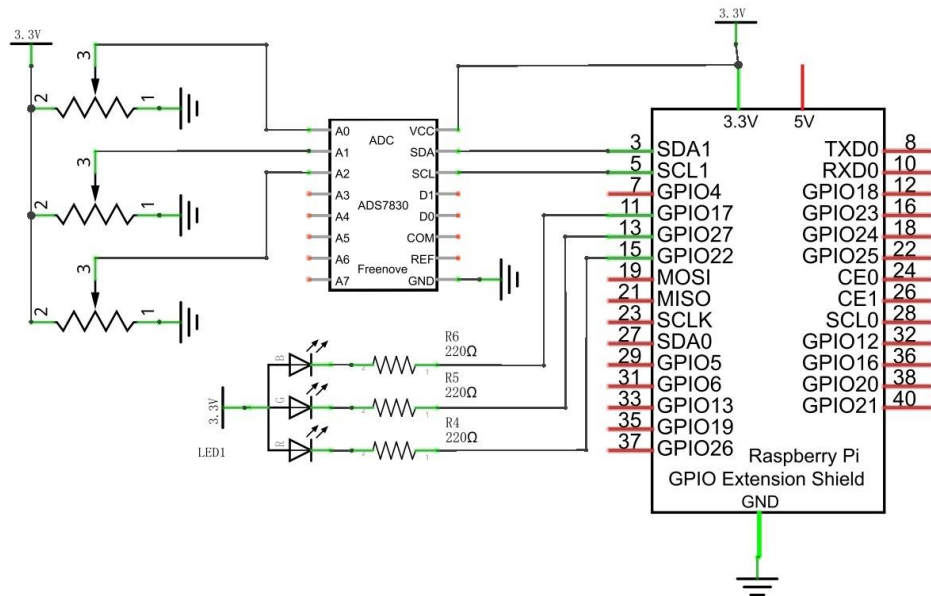
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
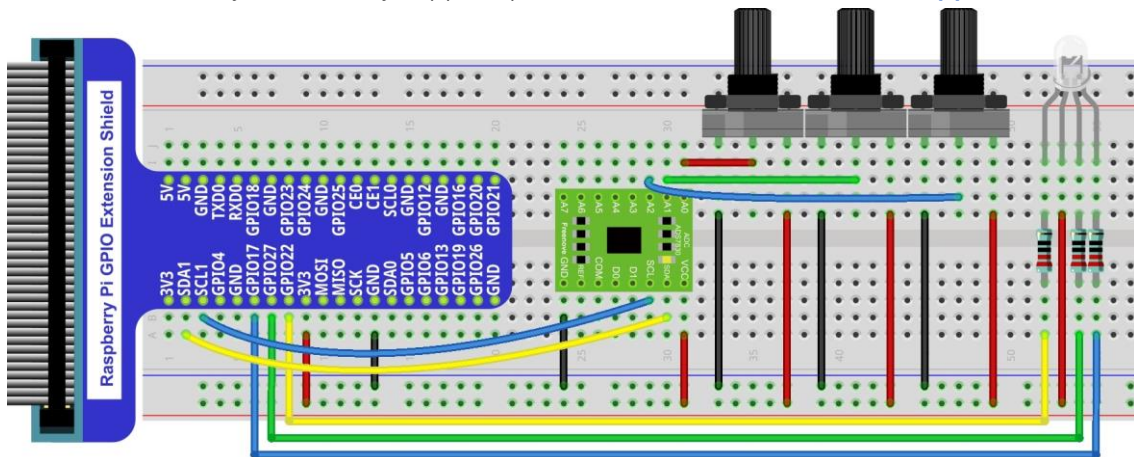
If circuit above doesn't work, please try following wiring.
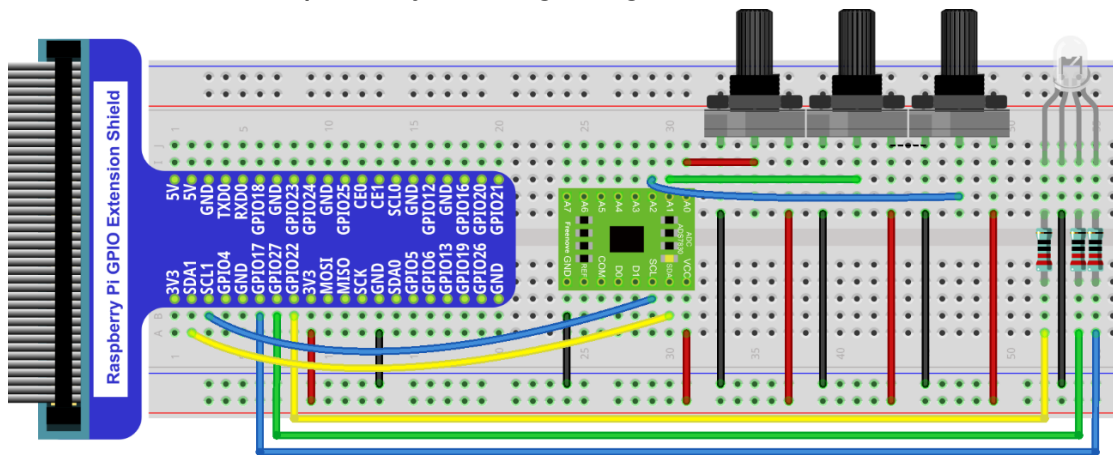
Circuit with PCF8591
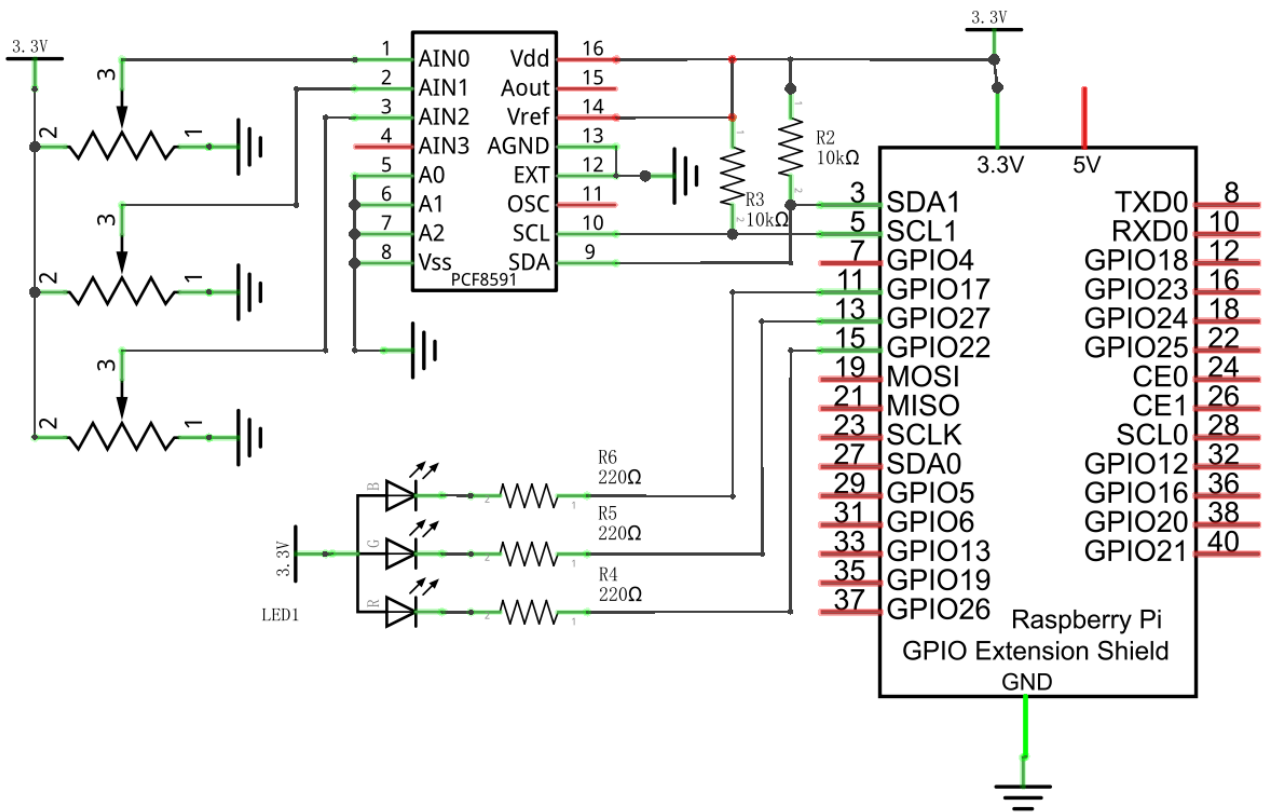
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
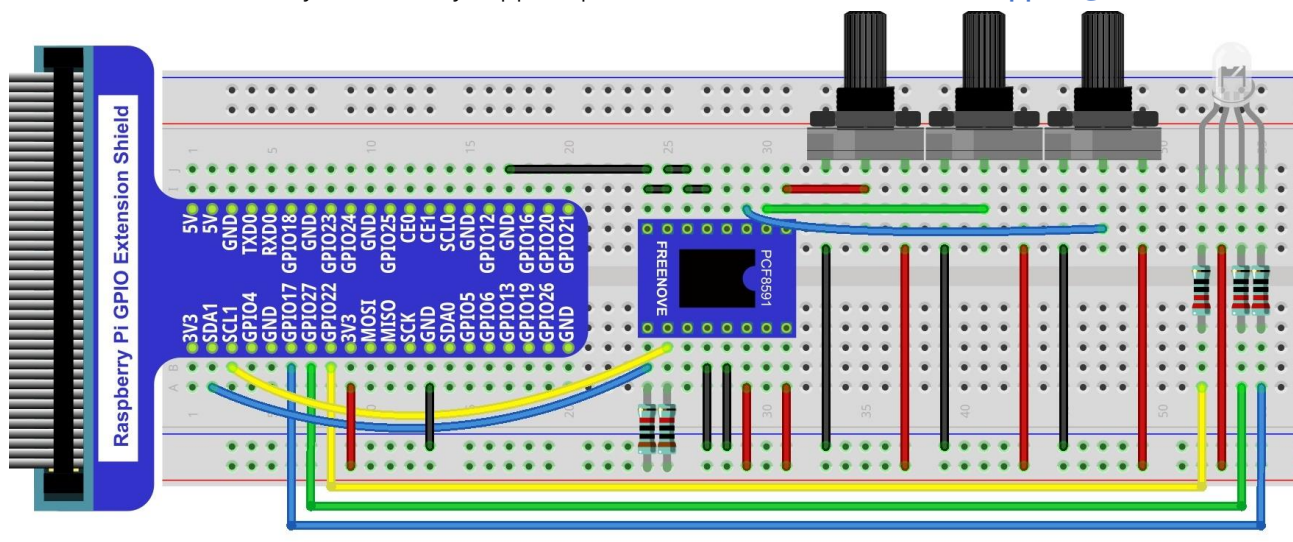
Code

Python Code 9.1.1 ColorfulSoftlight

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 09.1.1_ColorfulSoftlight directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/09.1.1_ColorfulSoftlight
```

2. Use python command to execute python code "ColorfulSoftlight.py".

```
python ColorfulSoftlight.py
```

After the program is executed, rotate one of the potentiometers, then the color of RGB LED will change. The Terminal window will display the ADC value of each potentiometer.

The following is the program code:

```python
1   from gpiozero import RGBLED
2   import time
3   from ADCDevice import *
4
5   led = RGBLED(red=22, green=27, blue=17, active_high=False) # define the pins for
    R:GPIO22,G:GPIO27,B:GPIO17
6   #led = RGBLED(red="J8:15", green="J8:13", blue="J8:11") # according to BOARD Numbering define
    the pins for R:11,G:12,B:13
7   adc = ADCDevice() # Define an ADCDevice class object
8
9   def setup():
10      global adc
11      if(adc.detectI2C(0x48)): # Detect the pcf8591.
12          adc = PCF8591()
13      elif(adc.detectI2C(0x4b)): # Detect the ads7830
14          adc = ADS7830()
15      else:
16          print("No correct I2C address found, \n"
17          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
18          "Program Exit. \n");
19          exit(-1)
20
21  def loop():
22      while True:
23          value_Red = adc.analogRead(0)        # read ADC value of 3 potentiometers
24          value_Green = adc.analogRead(1)
25          value_Blue = adc.analogRead(2)
26
```

```
27            led.red    =value_Red/255   # map the read value of potentiometers into PWM value and
       output it
28            led.green =value_Green/255
29            led.blue   =value_Blue/255
30            # print read ADC value
31            print ('ADC Value
       value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value_Blue))
32            time.sleep(0.01)
33
34    def destroy():
35        adc.close()
36        led.close()
37
38    if __name__ == '__main__': # Program entrance
39        print ('Program is starting ... ')
40        setup()
41        try:
42            loop()
43        except KeyboardInterrupt: # Press ctrl-c to end the program.
44            destroy()
45            print("Ending program")
```

In the code you can read the ADC values of the 3 potentiometers and map it into a PWM duty cycle to control the 3 LED elements to vary the color of their respective RGB LED.
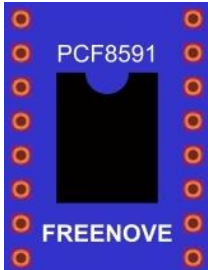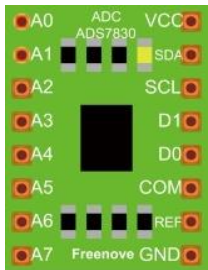
Chapter 10 Photoresistor & LED

In this chapter, we will learn how to use a photoresistor to make an automatic dimming nightlight.

Project 10.1 NightLamp

A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a nightlight with the following function. When the ambient light is less (darker environment), the LED will automatically become brighter to compensate and when the ambient light is greater (brighter environment) the LED will automatically dim to compensate.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wires M/M x15			
Photoresistor x1	ADC module x1		10kΩ x3	220Ω x1	LED x1

Component knowledge

Photoresistor

A Photoresistor is simply a light sensitive resistor. It is an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface. A Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.
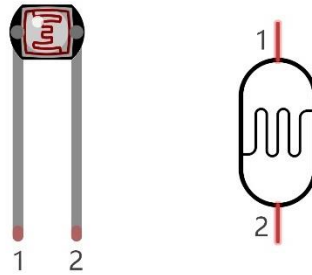
The circuit below is used to detect the change of a Photoresistor's resistance value:
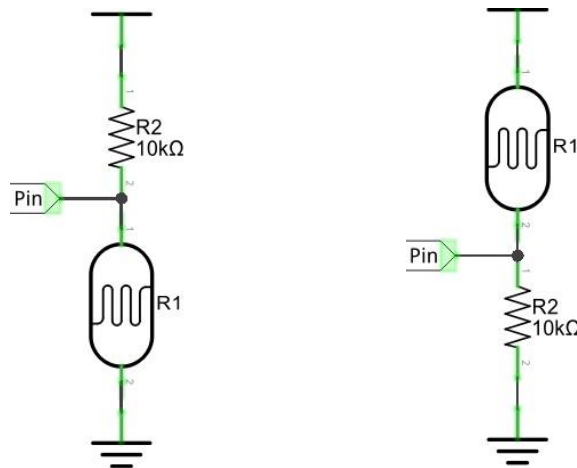
In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

Circuit with ADS7830

The circuit used is similar to the Soft light project. The only difference is that the input signal of the AIN0 pin of ADC changes from a Potentiometer to a combination of a Photoresistor and a Resistor.

Schematic diagram
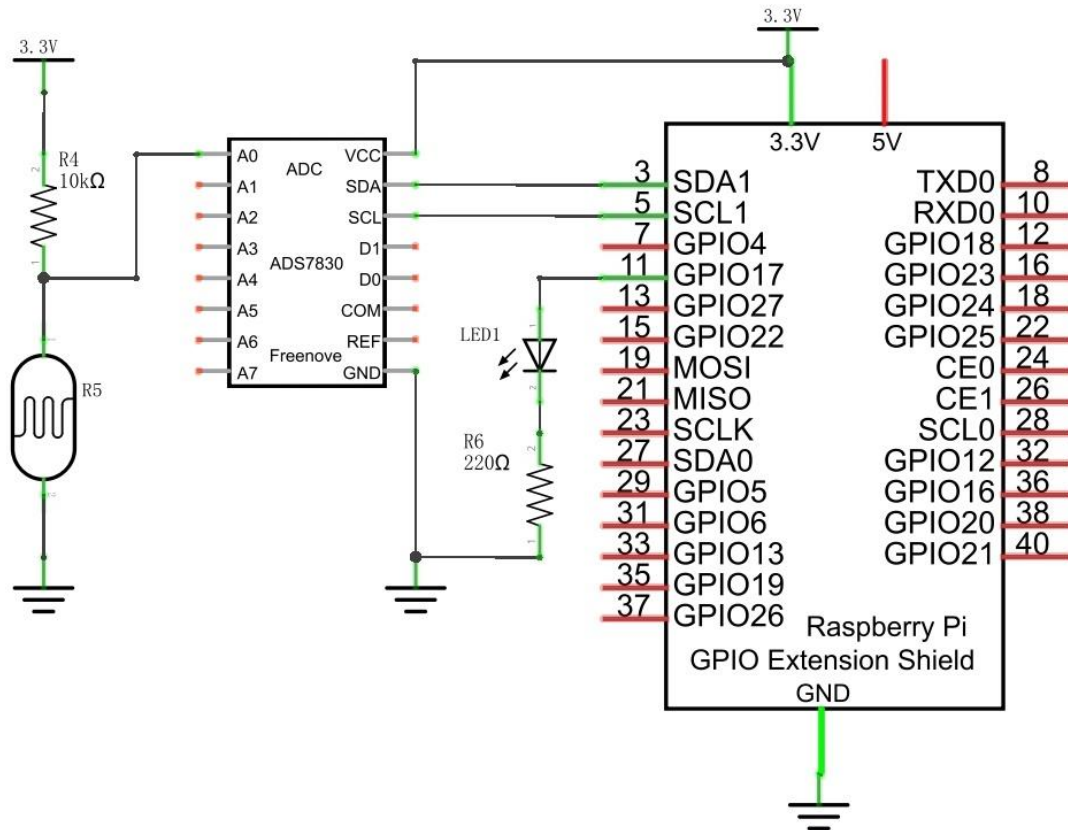
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
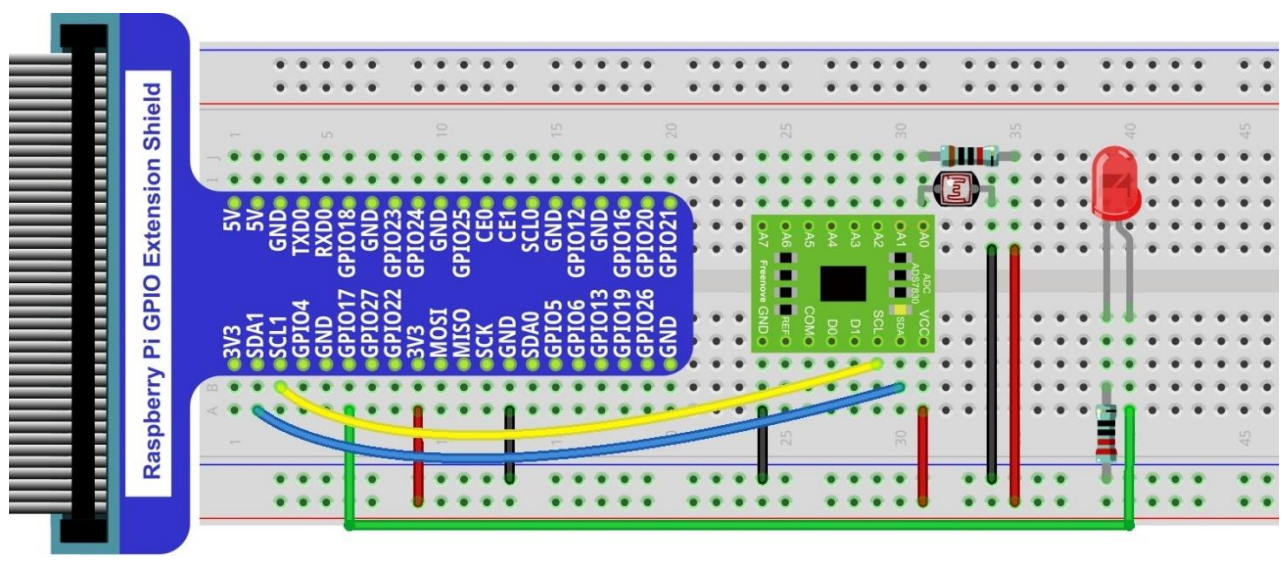
Circuit with PCF8591

The circuit used is similar to the Soft light project. The only difference is that the input signal of the AIN0 pin of ADC changes from a Potentiometer to a combination of a Photoresistor and a Resistor.

Schematic diagram
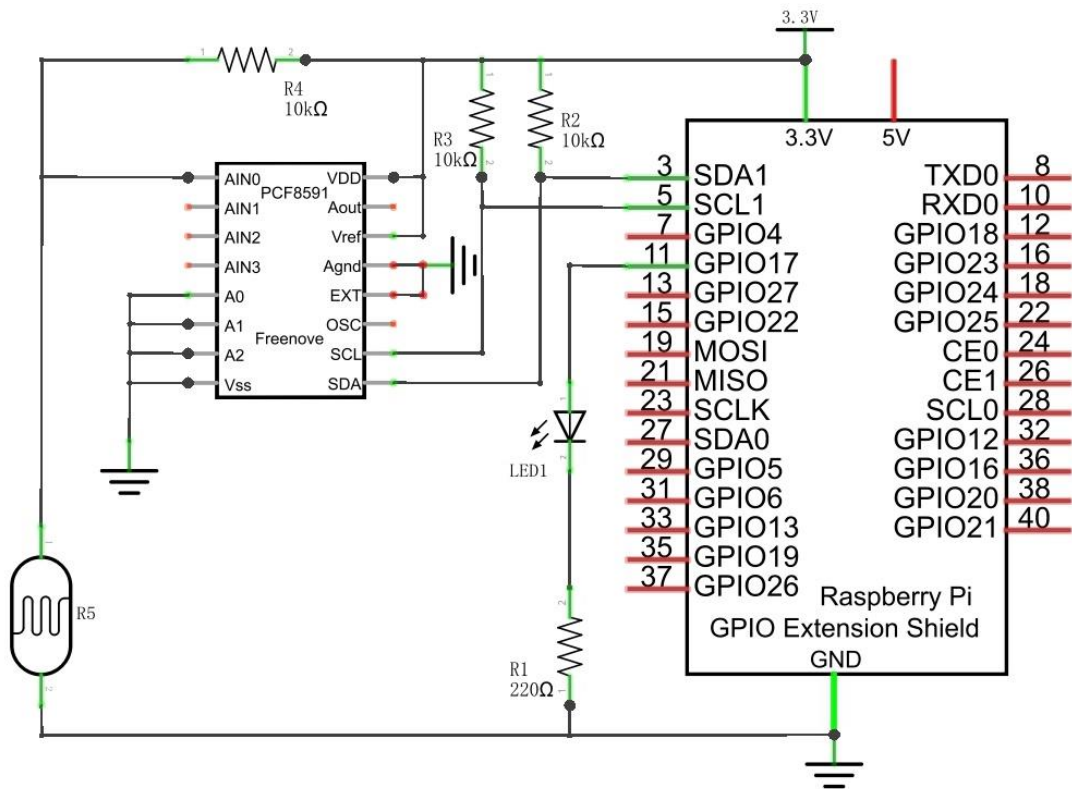
Hardware connection
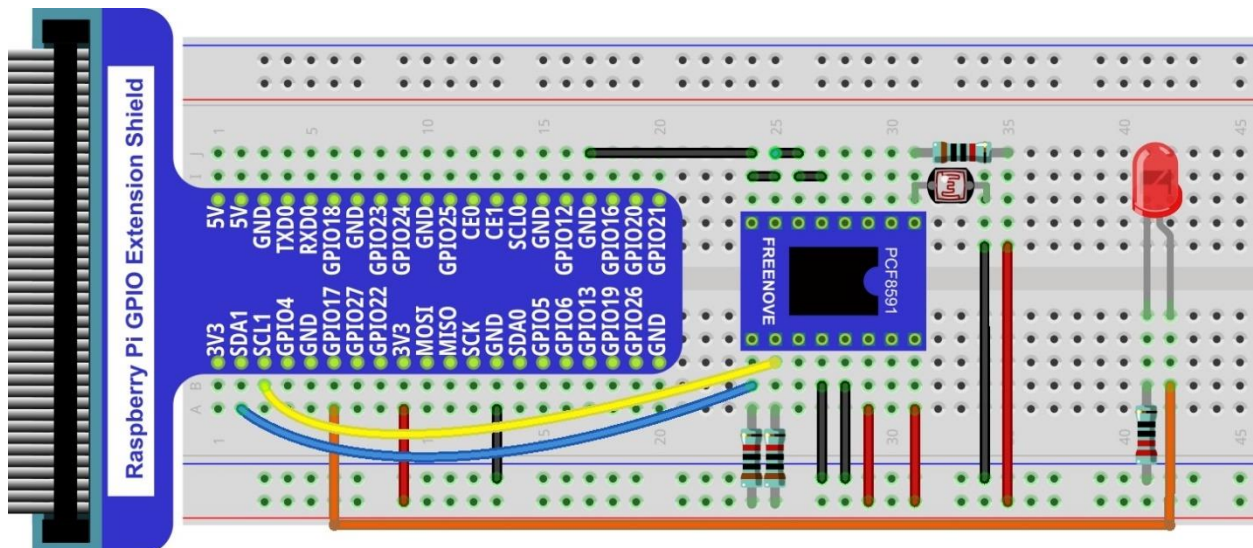
Code

The code used in this project is identical with what was used in the last chapter.

Python Code 10.1.1 Nightlamp

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1.   Use cd command to enter 10.1_Nightlamp directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/10.1.1_Nightlamp
```

2.   Use the python command to execute the Python code "Nightlamp.py".

```
python Nightlamp.py
```

After the program is executed, if you cover the Photoresistor or increase the light shining on it, the brightness of the LED changes accordingly. As in previous projects the Terminal window will display the current input voltage value of ADC module A0 pin and the converted digital quantity.

The following is the program code:

```python
1    from gpiozero import PWMLED
2    import time
3    from ADCDevice import *
4
5    ledPin = 17 # define ledPin
6    led = PWMLED(ledPin)
7    adc = ADCDevice() # Define an ADCDevice class object
8
9    def setup():
10       global adc
11       if(adc.detectI2C(0x48)): # Detect the pcf8591.
12           adc = PCF8591()
13       elif(adc.detectI2C(0x4b)): # Detect the ads7830
14           adc = ADS7830()
15       else:
16           print("No correct I2C address found, \n"
17           "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
18           "Program Exit. \n");
19           exit(-1)
20
21   def loop():
22       while True:
23           value = adc.analogRead(0)      # read the ADC value of channel 0
24           led.value = value / 255.0      # Mapping to PWM duty cycle
25           voltage = value / 255.0 * 3.3
26           print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
27           time.sleep(0.01)
```

```
28
29    def destroy():
30        led.close()
31        adc.close()
32
33    if __name__ == '__main__':    # Program entrance
34        print ('Program is starting ... ')
35        setup()
36        try:
37            loop()
38        except KeyboardInterrupt:   # Press ctrl-c to end the program.
39            destroy()
40            print("Ending program")
```
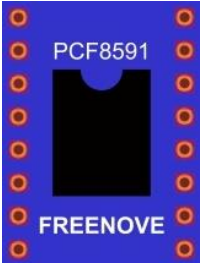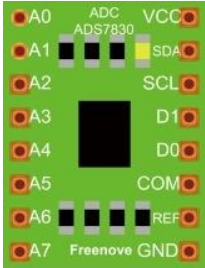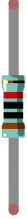
Chapter 11 Thermistor

In this chapter, we will learn about Thermistors which are another kind of Resistor.

Project 11.1 Thermometer

A Thermistoris a type of Resistor whose resistance value is dependent on temperature and changes in temperature. Therefore, we can take advantage of this characteristic to make a Thermometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire M/M x14	
Thermistor x1	ADC module x1	or	Resistor 10kΩ x3

Component knowledge

Thermistor

Thermistor is a temperature sensitive resistor. When it senses a change in temperature, the resistance of the Thermistor will change. We can take advantage of this characteristic by using a Thermistor to detect temperature intensity. A Thermistor and its electronic symbol are shown below.
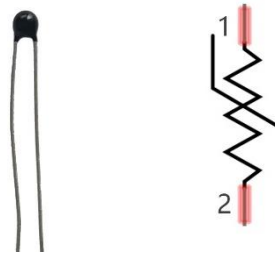
The relationship between resistance value and temperature of a thermistor is:

$$Rt=R*EXP [B*(1/T2-1/T1)]$$

Where:

   Rt is the thermistor resistance under T2 temperature;

   R is in the nominal resistance of thermistor under T1 temperature;

   EXP[n] is nth power of e;

   B is for thermal index;

   T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: B=3950, R=10k, T1=25.

The circuit connection method of the Thermistor is similar to photoresistor, as the following:
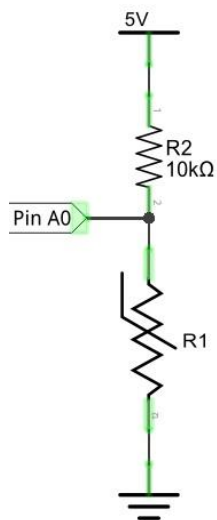
We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

Therefore, the temperature formula can be derived as:

$$T2 = 1/(1/T1 + ln(Rt/R)/B)$$

Circuit with ADS7830

The circuit of this project is similar to the one in last chapter. The only difference is that the Photoresistor is replaced by the Thermistor.

Schematic diagram
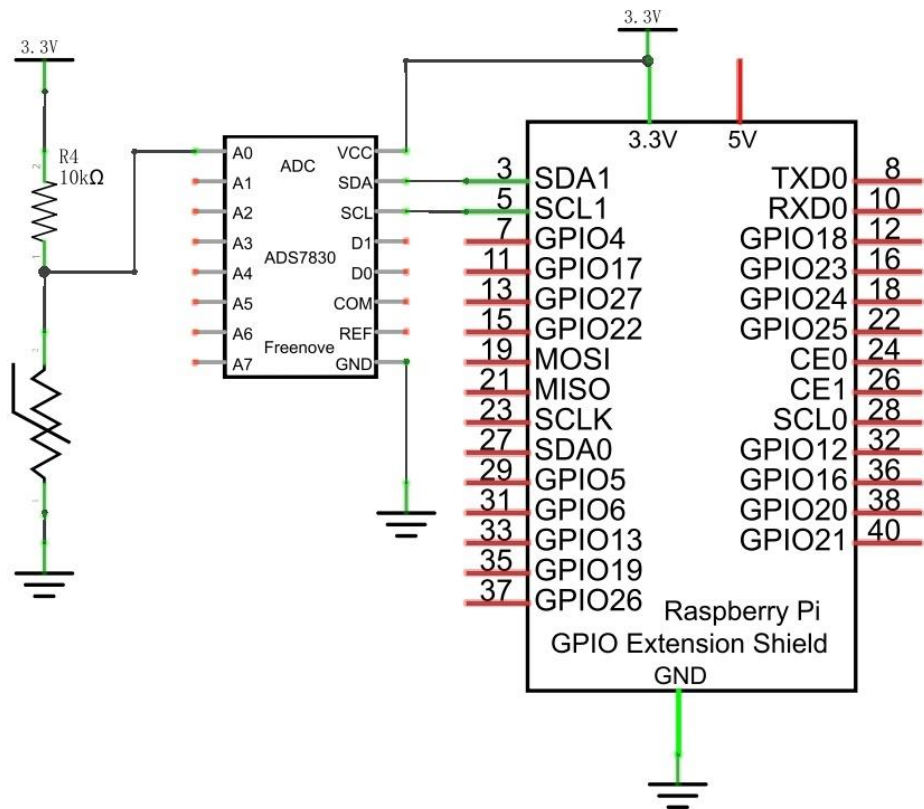
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
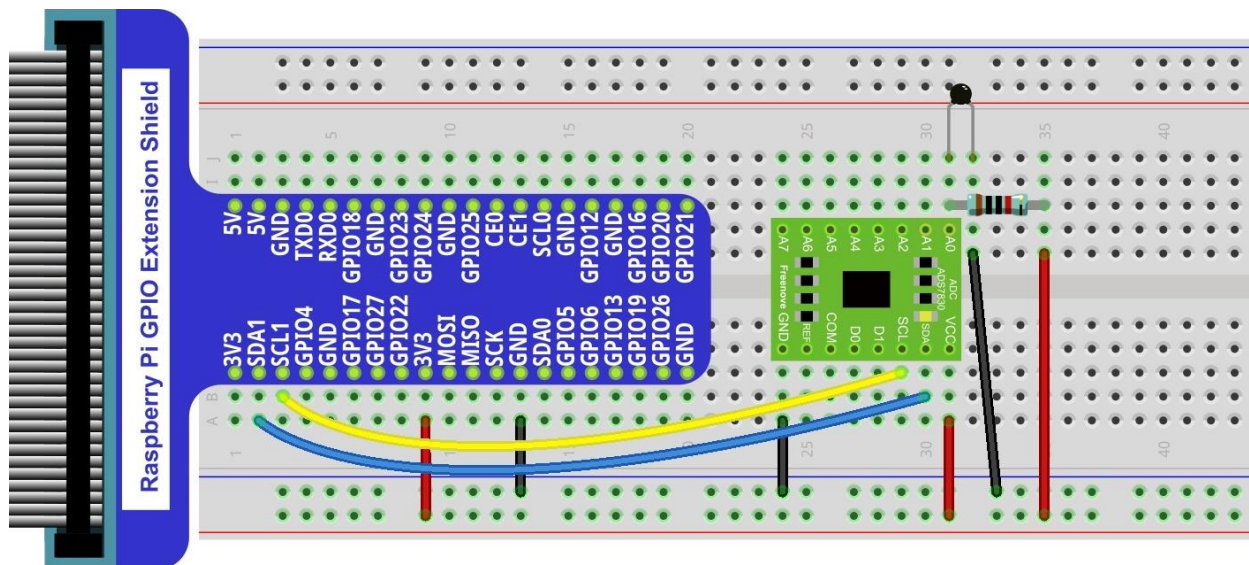
Thermistor has longer pins than the one shown in circuit.

Circuit with PCF8591

The circuit of this project is similar to the one in the last chapter. The only difference is that the Photoresistor is replaced by the Thermistor.

Schematic diagram
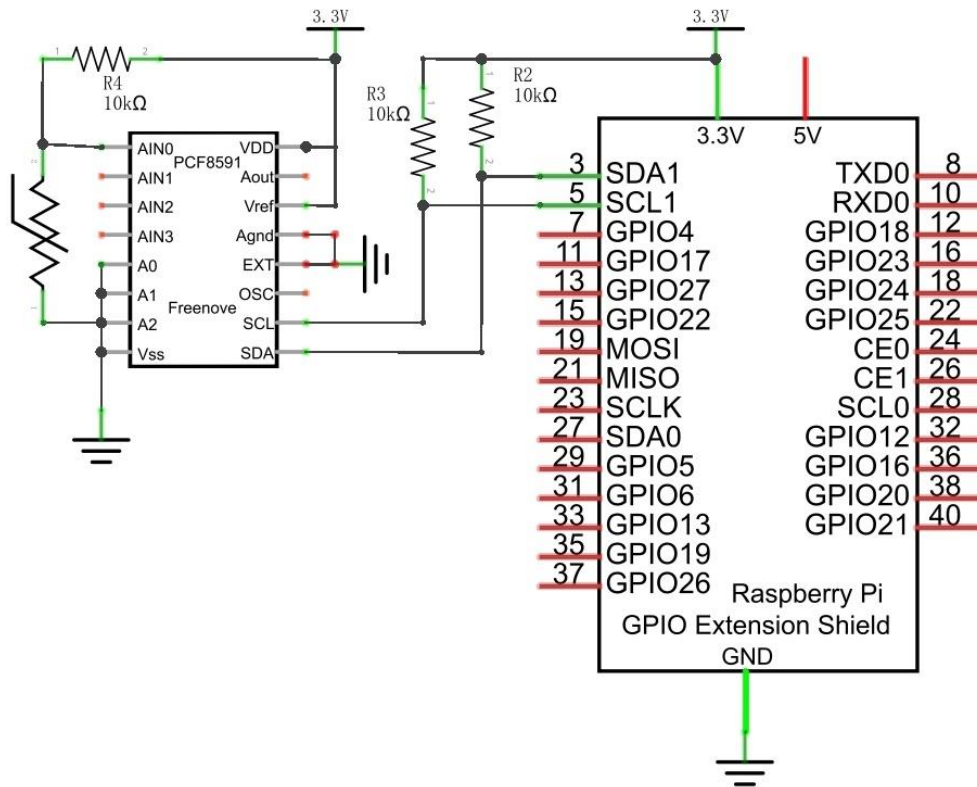
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
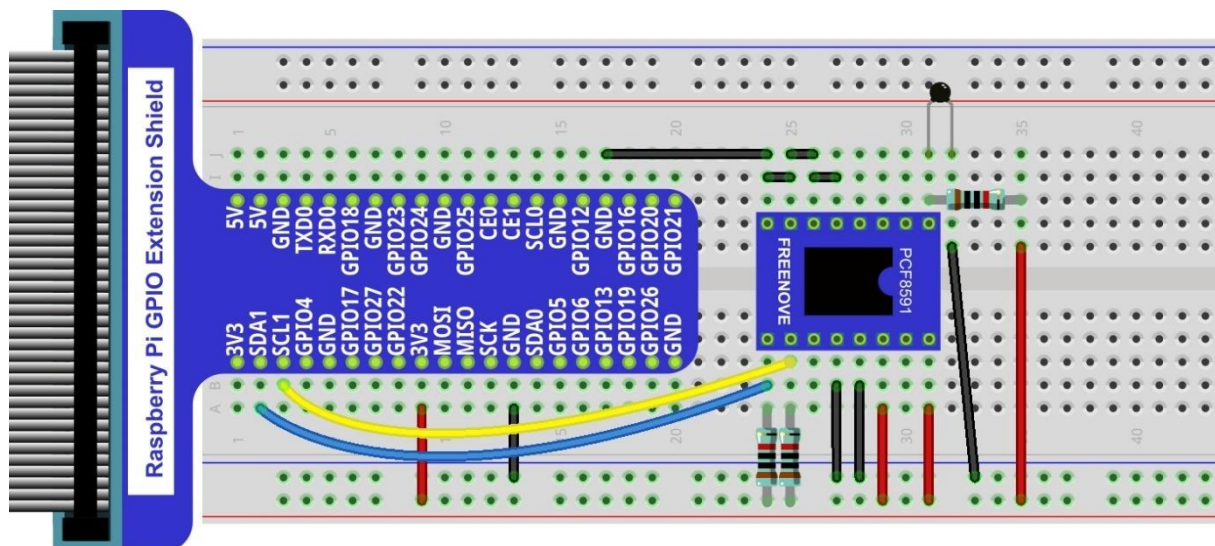
Thermistor has longer pins than the one shown in circuit.

Code

In this project code, the ADC value still needs to be read, but the difference here is that a specific formula is used to calculate the temperature value.

Python Code 11.1.1 Thermometer

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.
First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 11.1.1_Thermometer directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/11.1.1_Thermometer
```

2. Use python command to execute Python code "Thermometer.py".

```
python Thermometer.py
```

After the program is executed, the Terminal window will display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

```
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
```

The following is the code:

```python
1    import time
2    import math
3    from ADCDevice import *
4
5    adc = ADCDevice() # Define an ADCDevice class object
6
7    def setup():
8        global adc
9        if(adc.detectI2C(0x48)): # Detect the pcf8591.
10           adc = PCF8591()
11       elif(adc.detectI2C(0x4b)): # Detect the ads7830
12           adc = ADS7830()
13       else:
```

```
14              print("No correct I2C address found, \n"
15              "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
16              "Program Exit. \n");
17              exit(-1)
18
19   def loop():
20       while True:
21           value = adc.analogRead(0)        # read ADC value A0 pin
22           voltage = value / 255.0 * 3.3       # calculate voltage
23           Rt = 10 * voltage / (3.3 - voltage)    # calculate resistance value of thermistor
24           tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) # calculate temperature (Kelvin)
25           tempC = tempK -273.15        # calculate temperature (Celsius)
26           print ('ADC Value : %d, Voltage : %.2f, Temperature : %.2f'%(value,voltage,tempC))
27           time.sleep(0.01)
28
29   def destroy():
30       adc.close()
31
32   if __name__ == '__main__':  # Program entrance
33       print ('Program is starting ... ')
34       setup()
35       try:
36           loop()
37       except KeyboardInterrupt: # Press ctrl-c to end the program.
38           destroy()
39           print("Ending program")
```

In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of Thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the Thermistor, according to the formula.
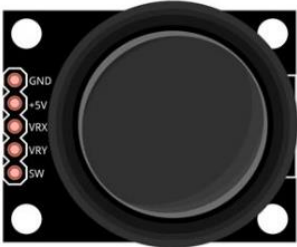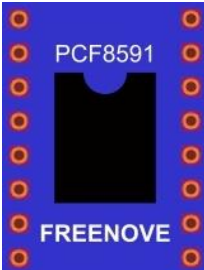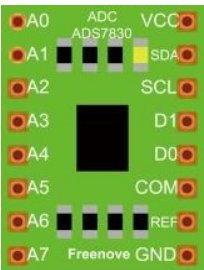
Chapter 12 Joystick

In an earlier chapter, we learned how to use Rotary Potentiometer. We will now learn about joysticks, which are electronic modules that work on the same principle as the Rotary Potentiometer.

Project 12.1 Joystick

In this project, we will read the output data of a joystick and display it to the Terminal screen.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper x18	
Joystick x1	ADC module x1		Resistor 10kΩ x3

Component knowledge

Joystick

A Joystick is a kind of input sensor used with your fingers. You should be familiar with this concept already as they are widely used in gamepads and remote controls. It can receive input on two axes (Y and or X) at the same time (usually used to control direction on a two dimensional plane). And it also has a third direction capability by **pressing down (Z axis/direction)**.
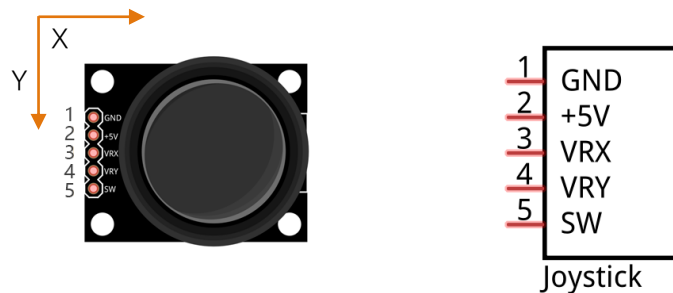
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the "vertical" axis, which can detect when a User presses on the Joystick.
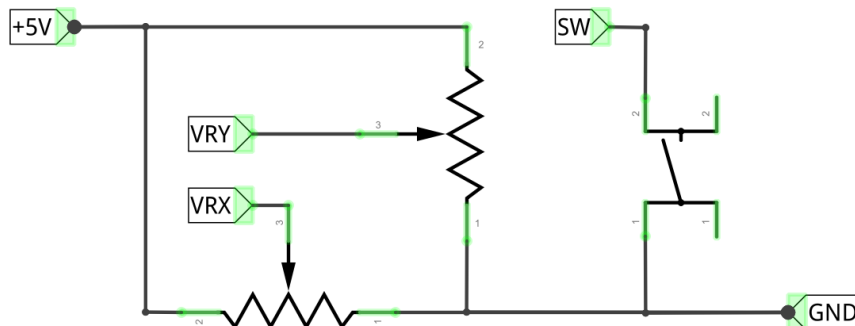
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit with ADS7830
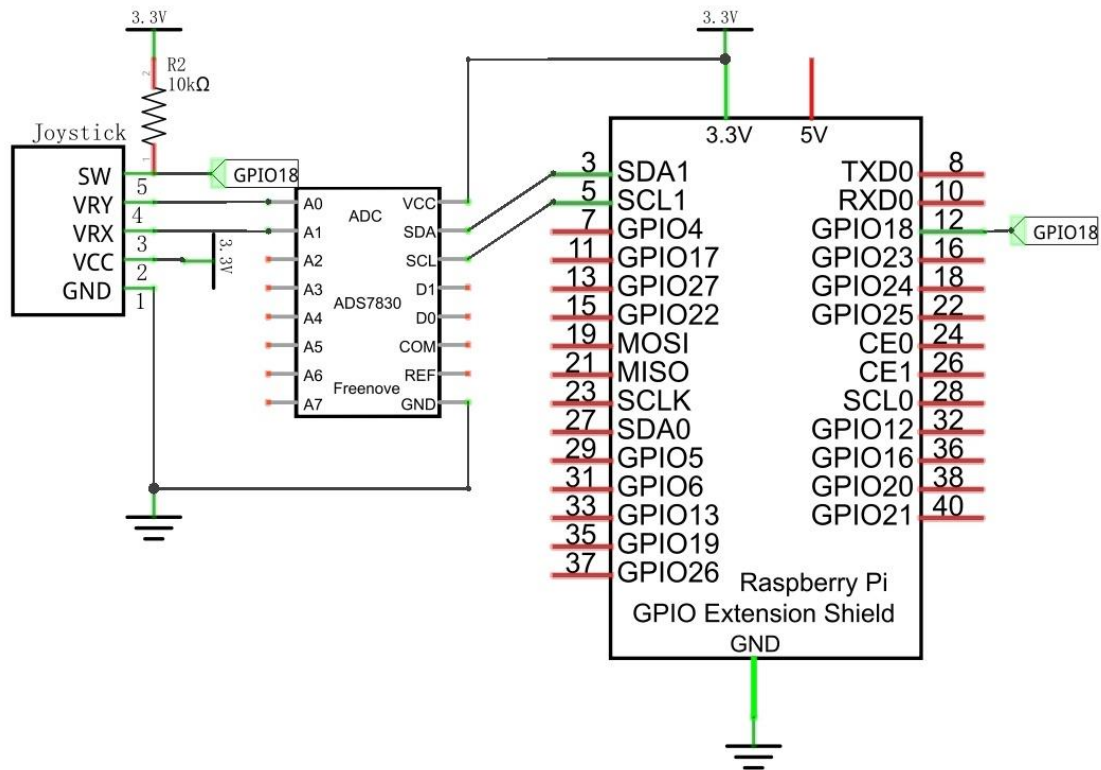
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

Circuit with PCF8591

Schematic diagram
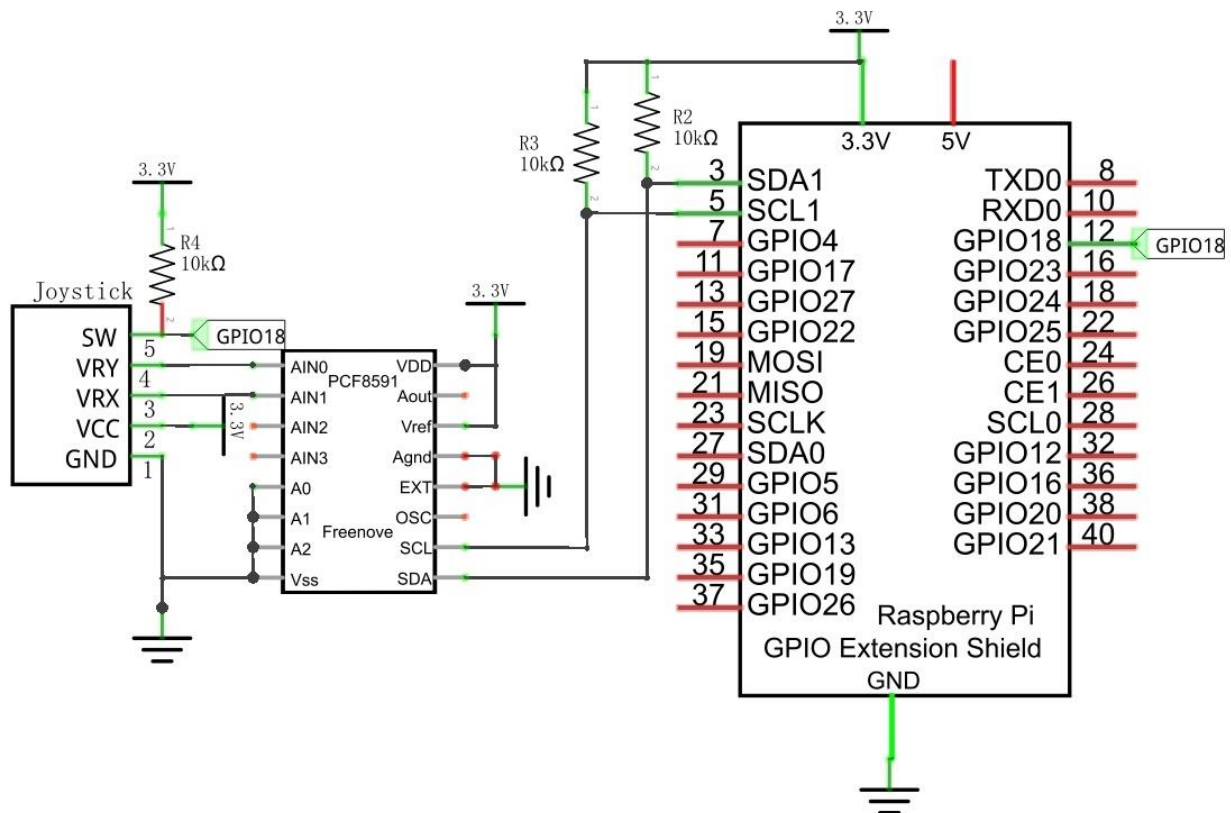
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
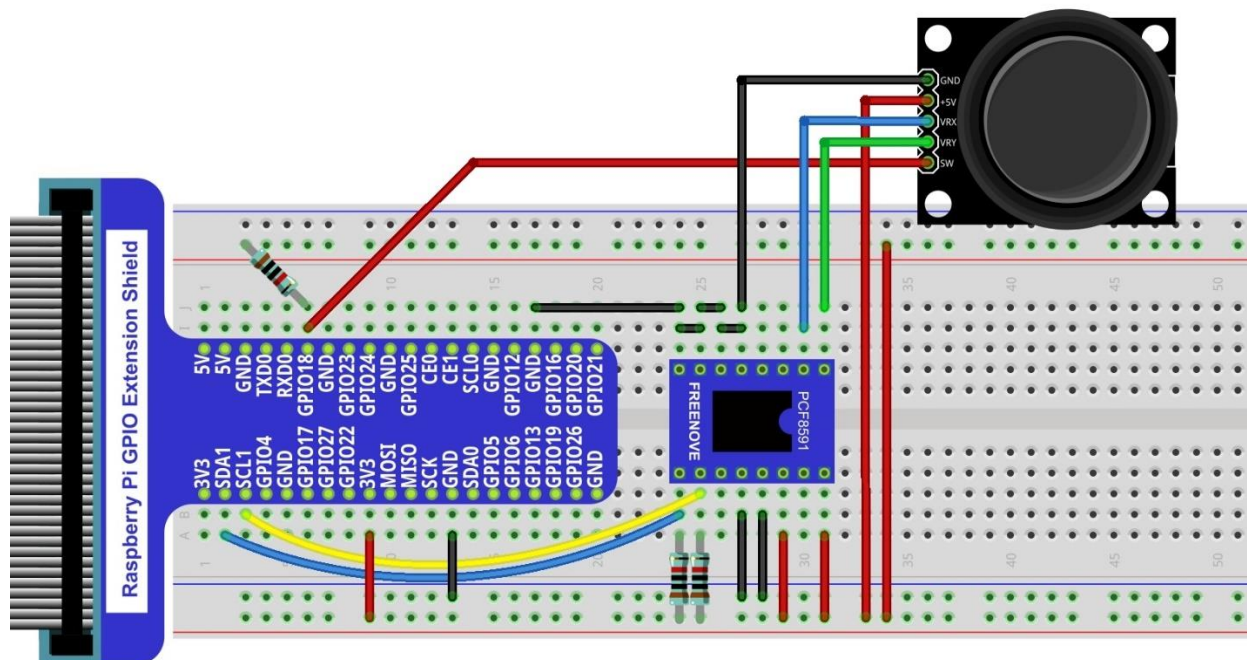
Code

In this project's code, we will read the ADC values of X and Y axes of the Joystick, and read digital quality of the Z axis, then display these out in Terminal.

Python Code 12.1.1 Joystick

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 12.1.1_Joystick directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/12.1.1_Joystick
```

2. Use Python command to execute Python code "Joystick.py".

```
python Joystick.py
```

After the program is executed, the Terminal window will display the data of 3 axes X, Y and Z. Shifting (moving) the joystick or pressing it down will make the data change.

```
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,   vlue_Y: 135 ,   value_Z: 1
```

The following is the program code:

```python
1   from gpiozero import Button
2   import time
3   from ADCDevice import *
4
5   Z_Pin = 18      # define Z_Pin
6   button = Button(Z_Pin) # define Button pin according to BCM Numbering
7   adc = ADCDevice() # Define an ADCDevice class object
8
9   def setup():
10      global adc
11      if(adc.detectI2C(0x48)): # Detect the pcf8591.
12          adc = PCF8591()
13      elif(adc.detectI2C(0x4b)): # Detect the ads7830
14          adc = ADS7830()
15      else:
16          print("No correct I2C address found, \n"
17          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
18          "Program Exit. \n");
19          exit(-1)
20
```

```
21   def loop():
22       while True:
23           val_Z = not button.value            # read digital value of axis Z
24           val_Y = adc.analogRead(0)            # read analog value of axis X and Y
25           val_X = adc.analogRead(1)
26           print('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
27           time.sleep(0.01)
28
29   def destroy():
30       adc.close()
31       button.close()
32
33   if __name__ == '__main__':
34       print('Program is starting ... ') # Program entrance
35       setup()
36       try:
37           loop()
38       except KeyboardInterrupt: # Press ctrl-c to end the program.
39           destroy()
40           print("Ending program")
```

In the code, configure Z_Pin as pull-up input mode. In the while loop, use analogRead() to read the values of the axes X and Y and use the variable val_Z to save the value of the button.value variable for the Z axis, and then display them. When the button is pressed, the value of the variable button.value is 1, otherwise the value is 0.

```
def loop():
    while True:
        val_Z = not button.value            # read digital value of axis Z
        val_Y = adc.analogRead(0)            # read analog value of axis X and Y
        val_X = adc.analogRead(1)
        print('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
        time.sleep(0.01)
```

For more information about the methods used by the Button class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_input.html#button
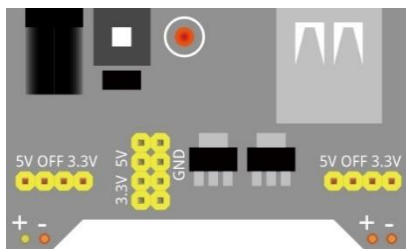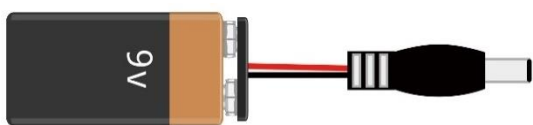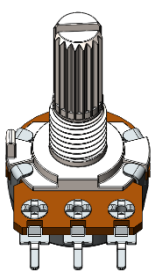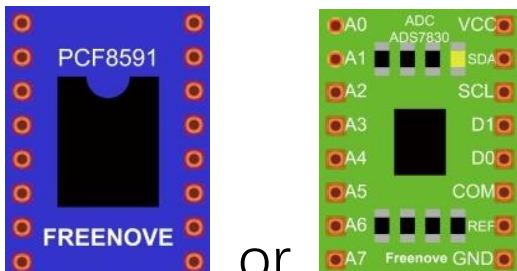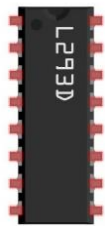
Chapter 13 Motor & Driver

In this chapter, we will learn about DC Motors and DC Motor Drivers and how to control the speed and direction of a DC Motor.

Project 13.1 Control a DC Motor with a Potentiometer

In this project, a potentiometer will be used to control a DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reached the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1			Jumper Wires x23	
Breadboard Power Module x1			9V Battery (you provide) & 9V Battery Cable	
Rotary Potentiometer x1	DC Motor x1	10kΩ x2	ADC Module x1 or	L293D IC Chip

Component knowledge

Breadboard Power Module

Breadboard Power Module is an independent circuit board, which can provide independent 5V or 3.3V power to the breadboard when building circuits. It also has built-in power protection to avoid damaging your RPi module. The schematic diagram below identifies the important features of this Power Module:
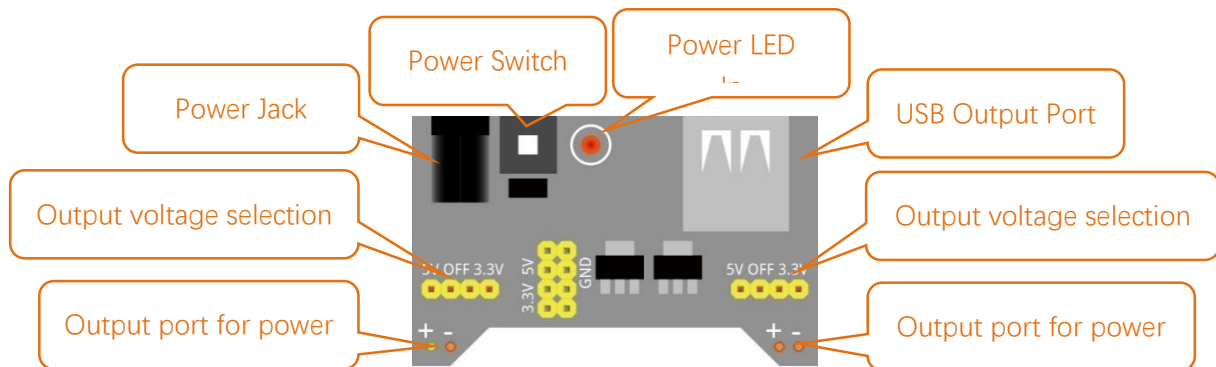
Here is an acceptable connection between Breadboard Power Module and Breadboard using a 9V battery and the provided power harness:
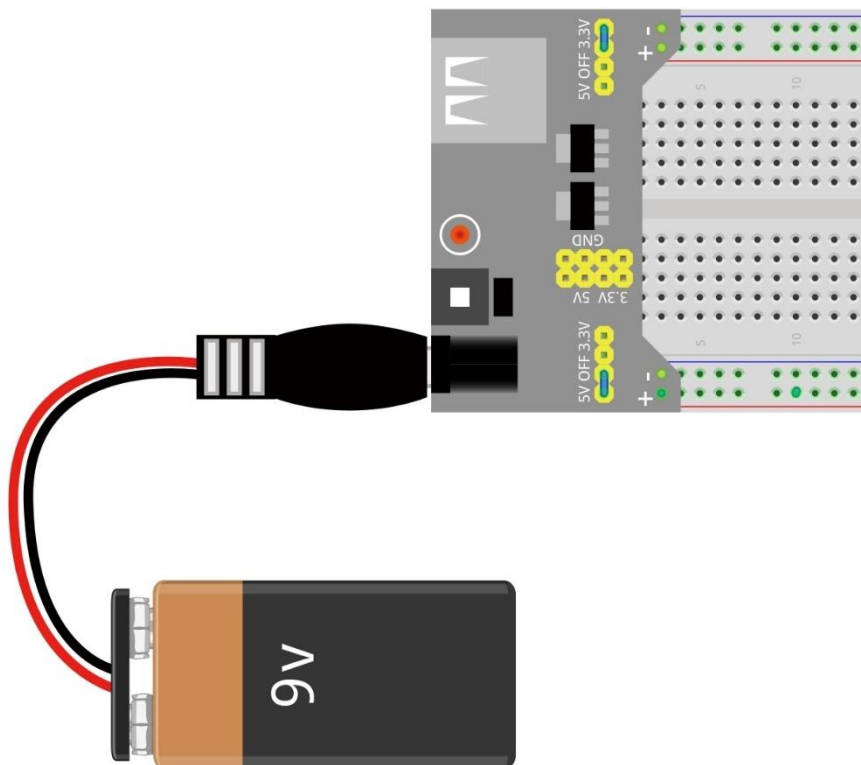
DC Motor

DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair of permanent magnets), and it has terminals to connect to the power if it is made up of electromagnet coils. Most Hobby DC Motors only use Permanent Magnets for the Stator Field. The Rotor is usually the shaft of motor with 3 or more electromagnets connected to a commutator where the brushes (via the terminals 1 & 2 below) supply electrical power, which can drive other mechanical devices. The diagram below shows a small DC Motor with two terminal pins.
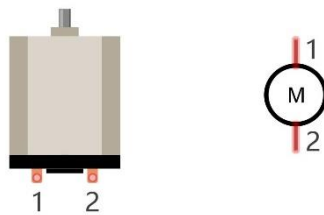
When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.

L293D

L293D is an IC Chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a Unidirectional DC Motor with 4 ports or a Bi-Directional DC Motor with 2 ports or a Stepper Motor (Stepper Motors are covered later in this Tutorial).
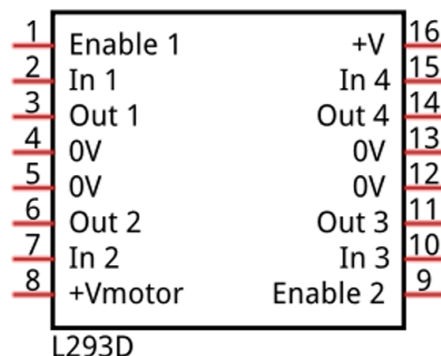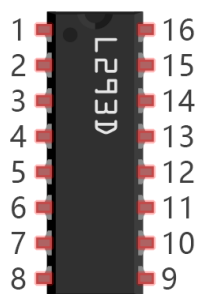
Port description of L293D module is as follows:

Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, gets connected to +Vmotor or 0V
Enable1	1	Channel 1 and Channel 2 enable pin, high level enable
Enable2	9	Channel 3 and Channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power Cathode (GND)
+V	16	Positive Electrode (VCC) of power supply, supply voltage 4.5~36V
+Vmotor	8	Positive Electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

For more details, please see the datasheet for this IC Chip.

When using the L293D to drive a DC Motor, there are usually two connection options.

The following connection option uses one channel of the L239D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.
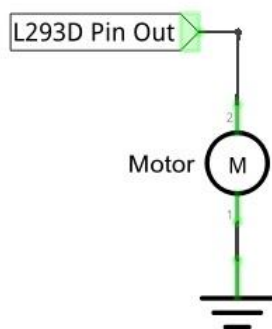
The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, not only controls the speed of motor, but also can control the direction of the motor.
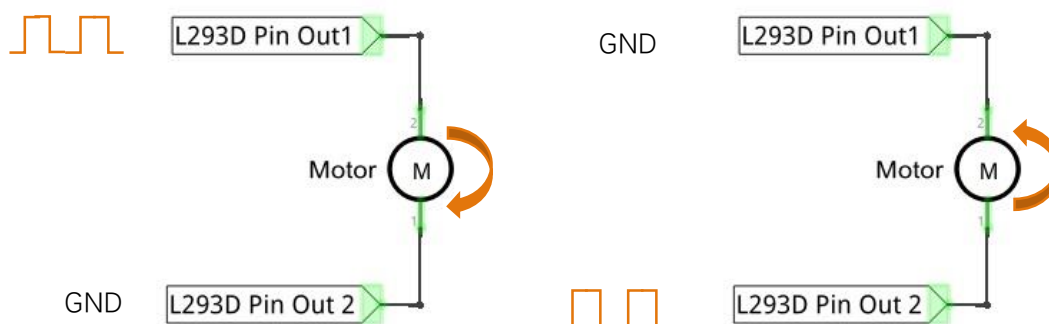
In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

Circuit with ADS7830

Use caution when connecting this circuit because the DC Motor is a high-power component. **Do not use the power provided by the RPi to power the motor directly, as this may cause permanent damage to your RPi!** The logic circuit can be powered by the RPi's power or an external power supply, which should share a common ground with RPi.
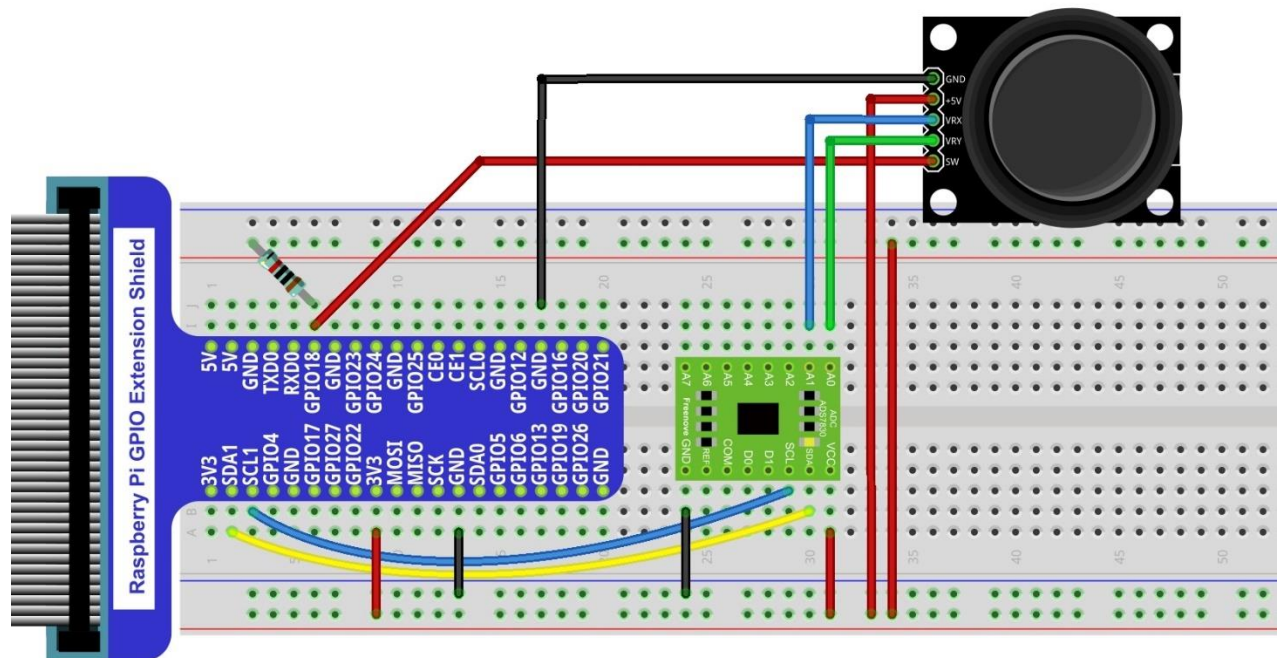
Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**

F/M Jumper Wire x2

Change the Jumper Position to Change the Motor's Supply Voltage (3.3V or 5V)

Select OFF

Press power switch when using.
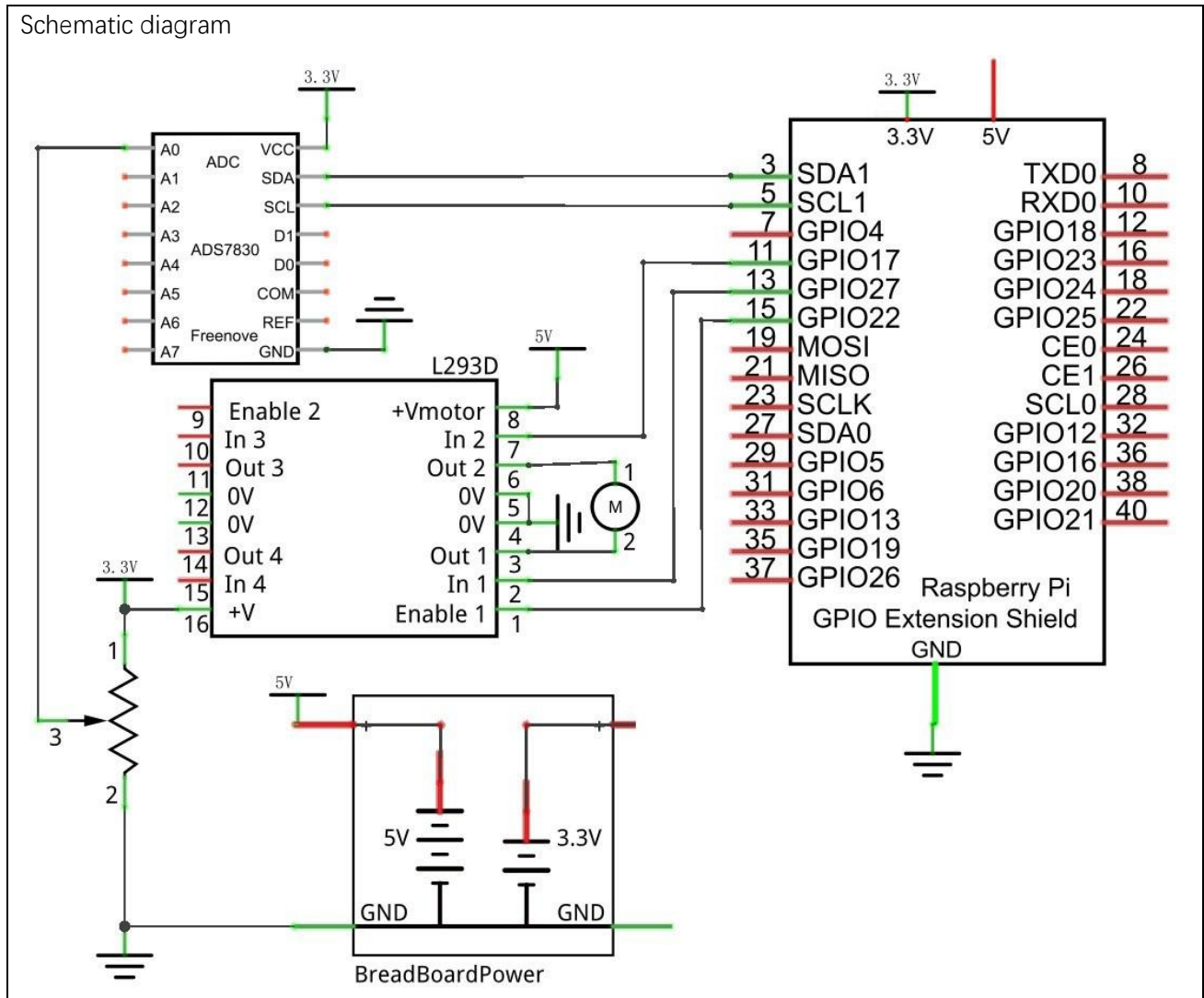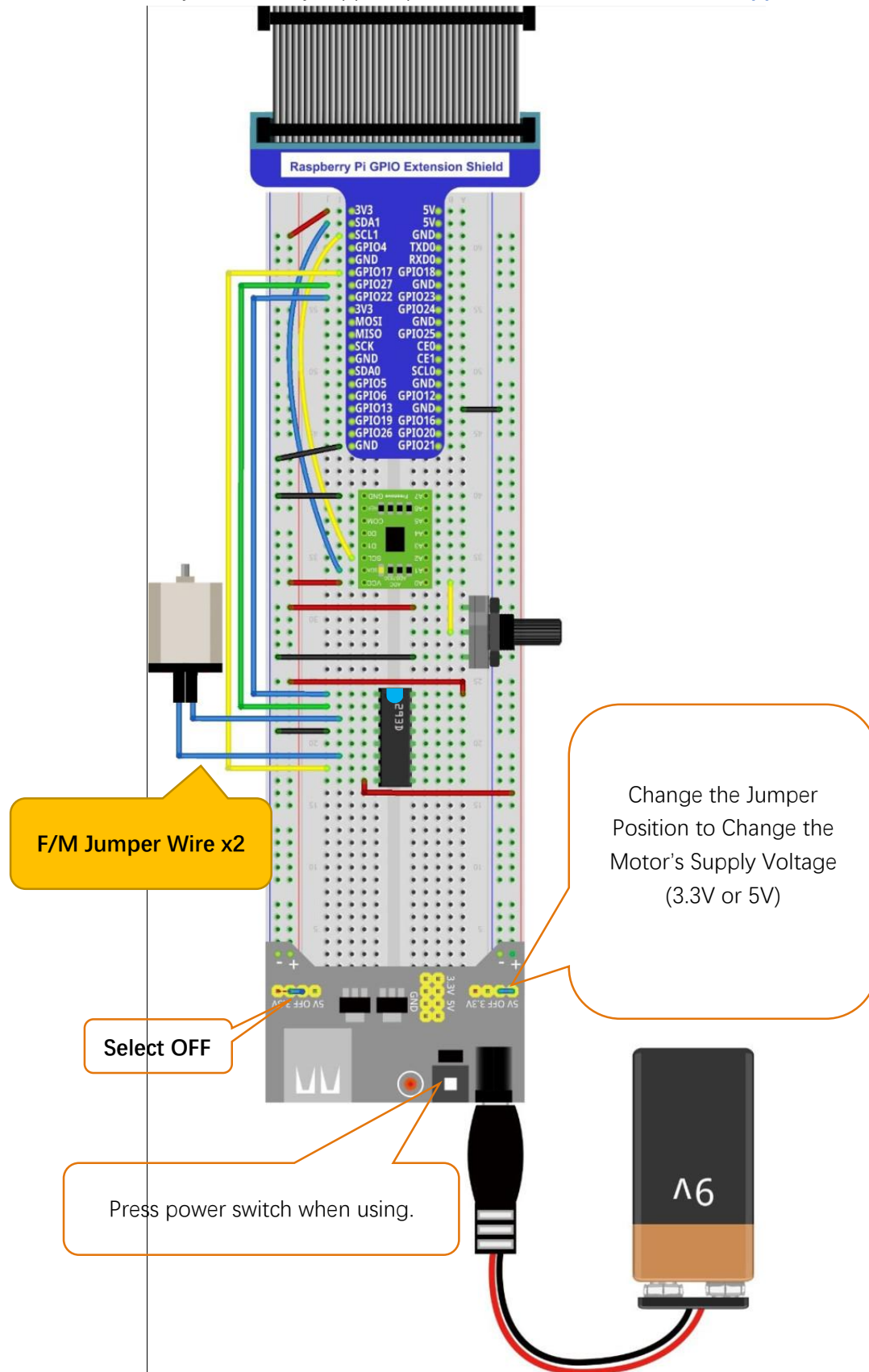
Circuit with PCF8591

Use caution when connecting this circuit because the DC Motor is a high-power component. **Do not use the power provided by the RPi to power the motor directly, as this may cause permanent damage to your RPi!** The logic circuit can be powered by the RPi's power or an external power supply, which should share a common ground with RPi.

Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**

F/M Jumper Wire x2

Change the Jumper Position to Change the Motor's Supply Voltage (3.3V or 5V)

Select OFF

The Power Switch
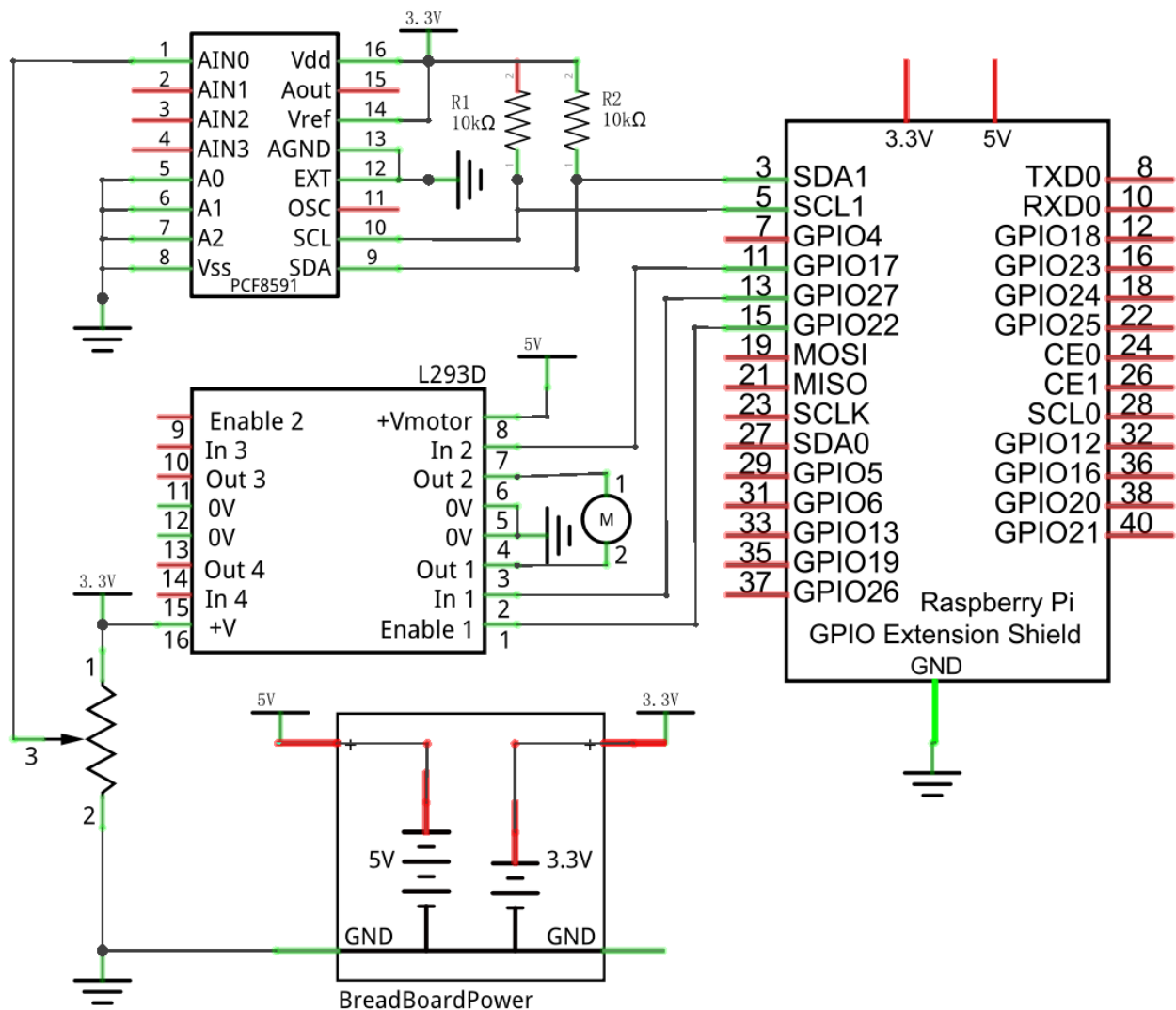
Code

In code for this project, first read the ADC value and then control the rotation direction and speed of the DC Motor according to the value of the ADC.

Python Code 13.1.1 Motor

If you did not configure I2C and install Smbus, please refer to Chapter 7. If you did, please Continue.
First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 13.1.1_Motor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/13.1.1_Motor
```

2. Use python command to execute the Python code "Motor.py".

```
python Motor.py
```

After the program is executed, you can use the Potentiometer to control the DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reaches the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction. You will also see the ADC value of the potentiometer displayed in the Terminal with the motor direction and the PWM duty cycle used to control the DC Motor's speed.

The following is the code:

```python
1   from gpiozero import DigitalOutputDevice,PWMOutputDevice
2   import time
3   from ADCDevice import *
4
5   # define the pins connected to L293D
6   motoRPin1 = DigitalOutputDevice(27)          # define L293D pin according to BCM Numbering
7   motoRPin2 = DigitalOutputDevice(17)          # define L293D pin according to BCM Numbering
8   enablePin = PWMOutputDevice(22,frequency=1000)
9   adc = ADCDevice() # Define an ADCDevice class object
```

```python
10
11   def setup():
12       global adc
13       if(adc.detectI2C(0x48)): # Detect the pcf8591.
14           adc = PCF8591()
15       elif(adc.detectI2C(0x4b)): # Detect the ads7830
16           adc = ADS7830()
17       else:
18           print("No correct I2C address found, \n"
19               "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
20               "Program Exit. \n");
21           exit(-1)
22   # mapNUM function: map the value from a range of mapping to another range.
23   def mapNUM(value,fromLow,fromHigh,toLow,toHigh):
24       return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
25
26   # motor function: determine the direction and speed of the motor according to the input ADC
     value input
27   def motor(ADC):
28       value = ADC -128
29       if (value > 0):   # make motor turn forward
30           motoRPin1.on()          # motoRPin1 output HIHG level
31           motoRPin2.off()         # motoRPin2 output LOW level
32           print ('Turn Forward...')
33       elif (value < 0): # make motor turn backward
34           motoRPin1.off()
35           motoRPin2.on()
             print ('Turn Backward...')
36       else :
37           motoRPin1.off()
38           motoRPin2.off()
39           print ('Motor Stop...')
40       b=mapNUM(abs(value),0,128,0,100)
41       enablePin.value = b / 100.0      # set dc value as the duty cycle
42       print ('The PWM duty cycle is %d%%\n'%(abs(value)*100/127))   # print PMW duty cycle.
43
44   def loop():
45       while True:
46           value = adc.analogRead(0) # read ADC value of channel 0
47           print ('ADC Value : %d'%(value))
48           motor(value)
49           time.sleep(0.2)
50
51   def destroy():
```

```
52          motoRPin1.close()
53          motoRPin2.close()
54          enablePin.close()
55          adc.close()
56
57  if __name__ == '__main__':  # Program entrance
58      print ('Program is starting ... ')
59      setup()
60      try:
61          loop()
62      except KeyboardInterrupt: # Press ctrl-c to end the program.
63          destroy()
64          print("Ending program")
```

Now that we have familiarity with reading ADC values, let's learn the subfunction void motor (int ADC): first, compare the ADC value with 128 (value corresponding to midpoint). When the current ADC value is higher, motoRPin1 outputs high level and motoRPin2 outputs low level to control the DC Motor to run in the "Forward" Rotational Direction. When the current ADC value is lower, motoRPin1 outputs low level and motoRPin2 outputs high level to control the DC Motor to run in the "Reverse" Rotational Direction. When the ADC value is equal to 128, motoRPin1 and motoRPin2 output low level, the motor STOPS. Then determine the PWM duty cycle according to the difference (delta) between ADC value and 128. Because the absolute delta value stays within 0-128. We need to use the map() subfunction mapping the delta value to a range of 0-255. Finally, we see a display of the duty cycle in Terminal.

```
def motor(ADC):
    value = ADC -128
    if (value > 0):  # make motor turn forward
        motoRPin1.on()          # motoRPin1 output HIHG level
        motoRPin2.off()         # motoRPin2 output LOW level
        print ('Turn Forward...')
    elif (value < 0): # make motor turn backward
        motoRPin1.off()
        motoRPin2.on()
        print ('Turn Backward...')
    else :
        motoRPin1.off()
        motoRPin2.off()
        print ('Motor Stop...')
    b=mapNUM(abs(value), 0, 128, 0, 100)
    enablePin.value = b / 100.0      # set dc value as the duty cycle
    print ('The PWM duty cycle is %d%%\n'%(abs(value)*100/127))    # print PMW duty cycle.
```
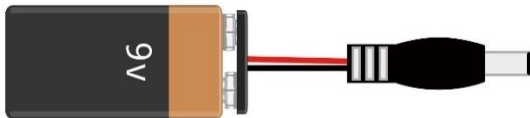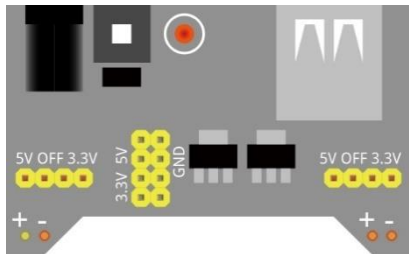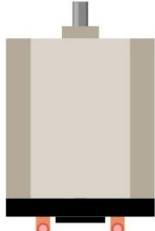
Chapter 14 Relay & Motor

In this chapter, we will learn a kind of special switch module, Relay Module.

Project 14.1.1 Relay & Motor

In this project, we will use a Push Button Switch indirectly to control the DC Motor via a Relay.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x11		
9V battery (prepared by yourself) & battery line			
Breadboard Power module x1	Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1
NPN transistor x1	Relay x1	Motor x1	Push button x1
LED x1	Diode x1		

Component knowledge

Relay

Relays are a type of Switch that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit using an electromagnet to initiate the Switch action. When the electromagnet is energized (powered), it will attract internal contacts completing a circuit, which act as a Switch. Many times Relays are used to allow a low powered circuit (and a small low amperage switch) to safely turn ON a larger more powerful circuit. They are commonly found in automobiles, especially from the ignition to the starter motor.

The following is a basic diagram of a common Relay and the image and circuit symbol diagram of the 5V relay used in this project:

Diagram	Feature:	Symbol
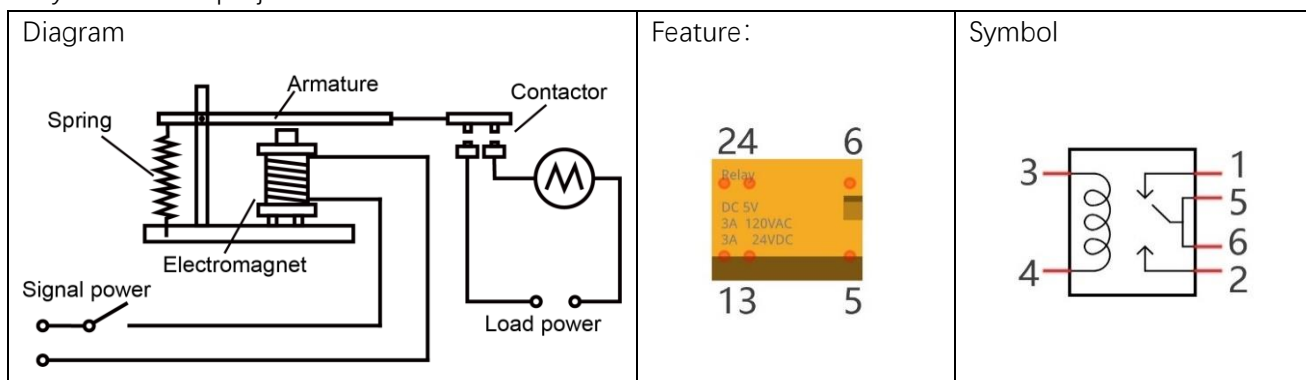
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

Inductor

The symbol of Inductance is "L" and the unit of inductance is the "Henry" (H). Here is an example of how this can be encountered: 1H=1000mH, 1mH=1000μH.

An Inductor is a passive device that stores energy in its Magnetic Field and returns energy to the circuit whenever required. An Inductor is formed by a Cylindrical Core with many Turns of conducting wire (usually copper wire). Inductors will hinder the changing current passing through it. When the current passing through the Inductor increases, it will attempt to hinder the increasing movement of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing movement of current. So the current passing through an Inductor is not transient.

The circuit for a Relay is as follows: The coil of Relay can be equivalent to an Inductor, when a Transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes in parallel are placed on both ends of the Relay coil pins in opposite polar direction. Having the current pass through the diodes will avoid any adverse effect on the power supply.
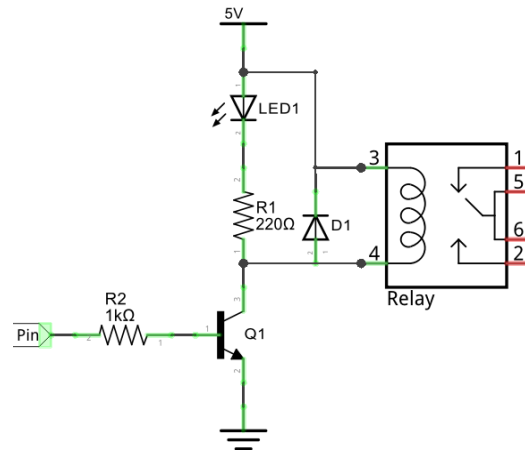
Circuit

Use caution with the power supply voltage needed for the components in this circuit. The Relay requires a power supply voltage of 5V, and the DC Motor only requires 3.3V. Additionally, there is an LED present, which acts as an indicator (ON or OFF) for the status of the Relay's active status.

Schematic diagram
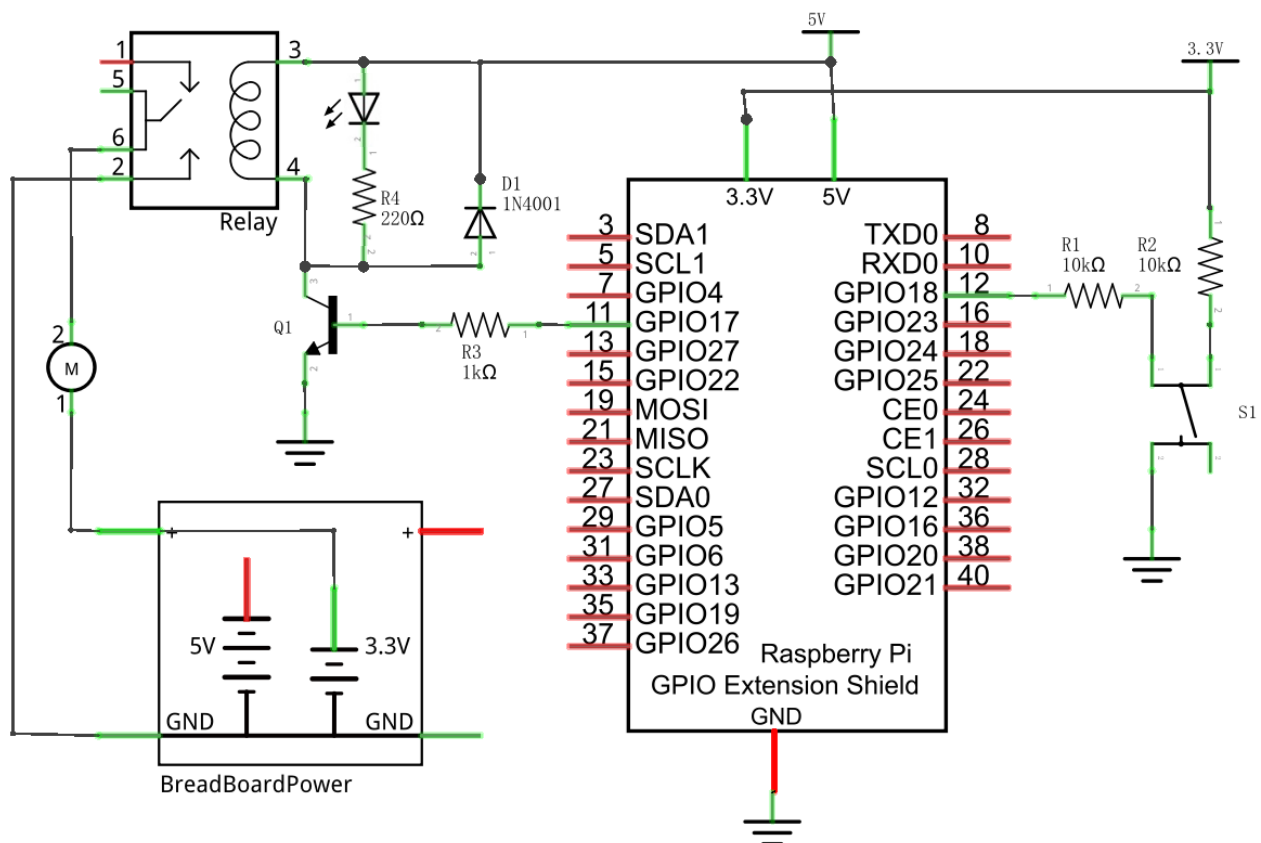
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
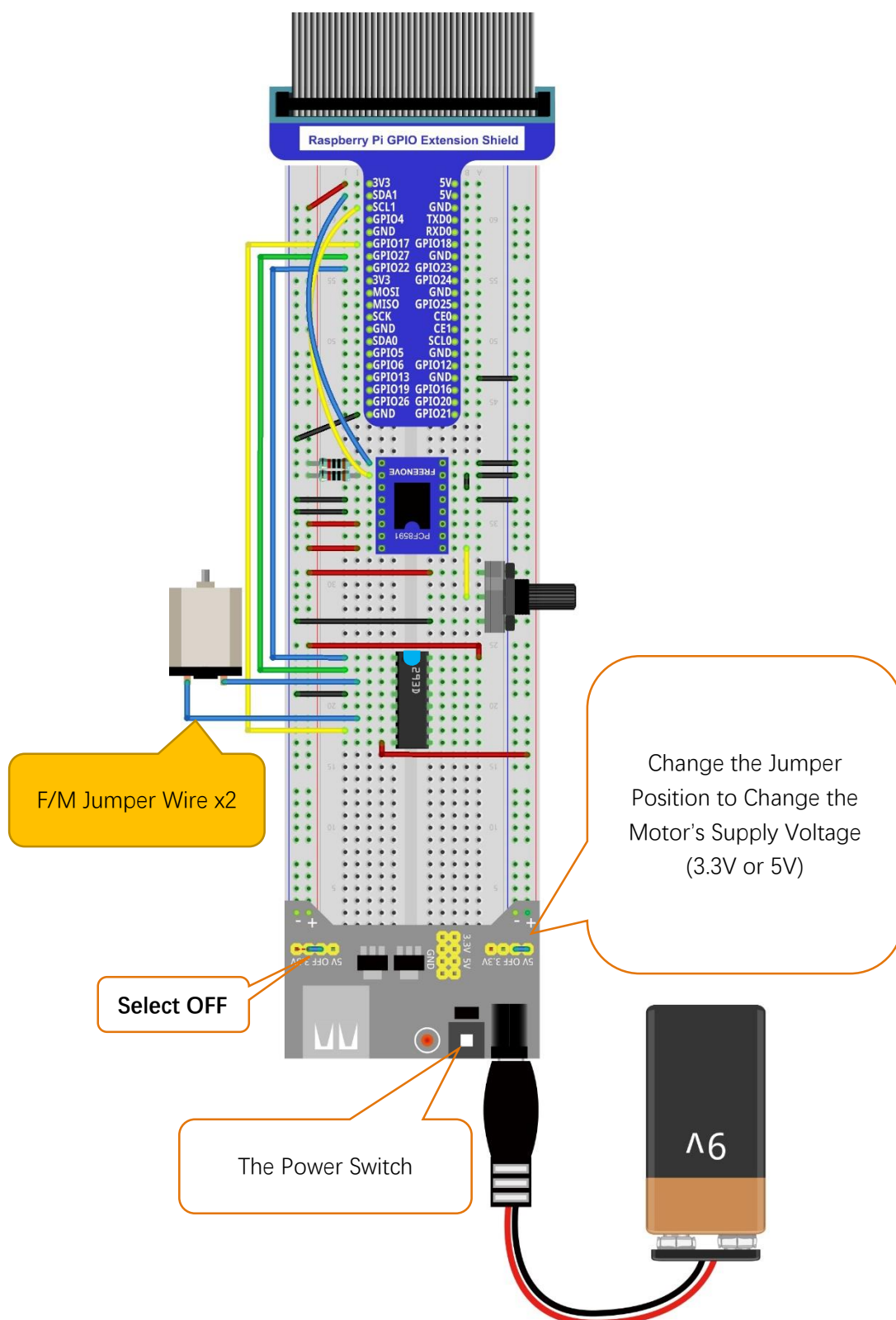

Press replay

for connection.

3.3V

OFF

9V
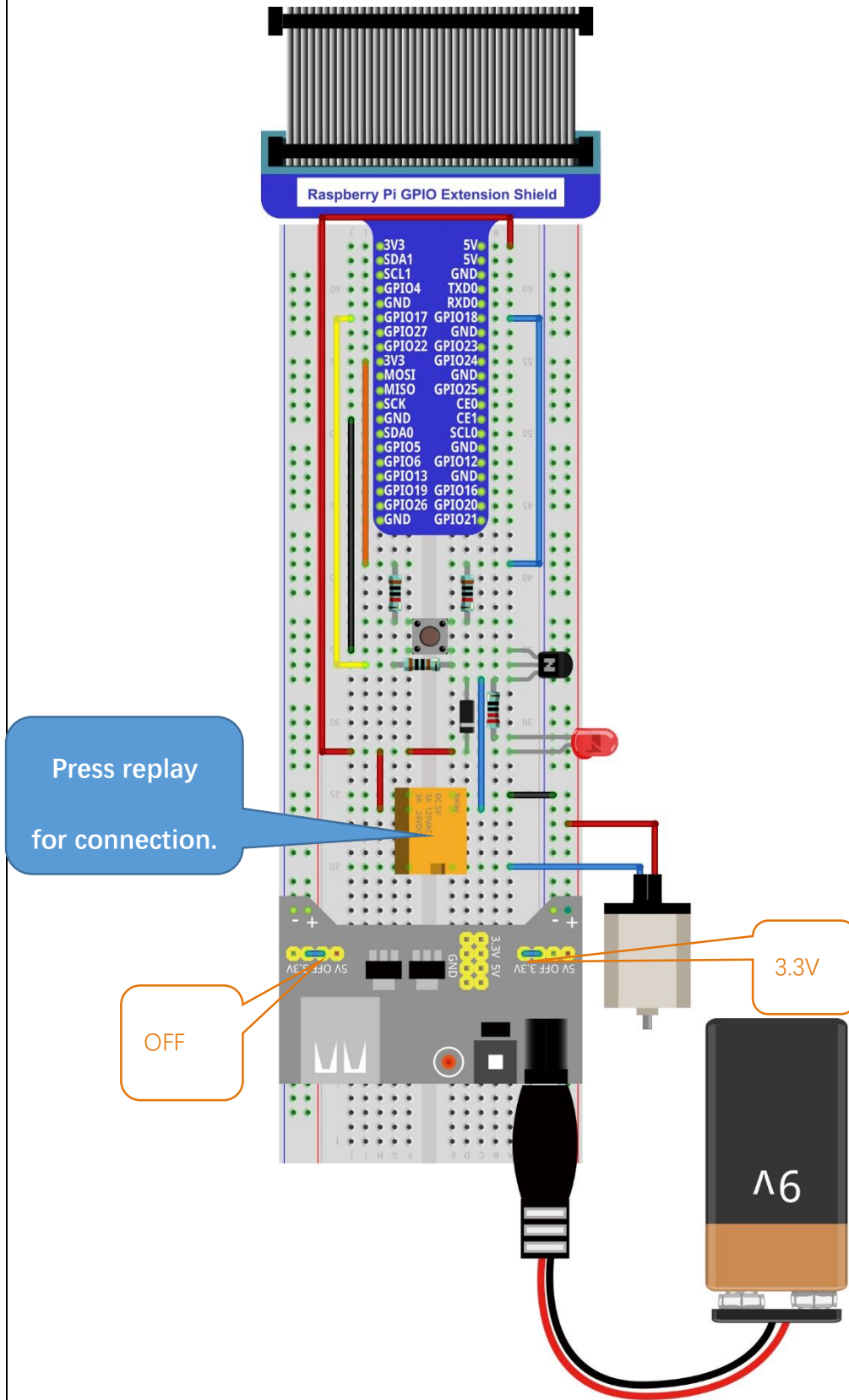
Code

The project code is in the same as we used earlier in the Table Lamp project. Pressing the Push Button Switch activates the transistor. Because the Relay and the LED are connected in parallel, they will be powered ON at the same time. Press the Push Button Switch again will turn them both OFF.

Python Code 14.1.1 Relay

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 14.1.1_Relay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/14.1.1_Relay
```

2. Use Python command to execute code "Relay.py".

```
python Relay.py
```

After the program is executed, pressing the Push Button Switch activates the Relay (the internal switch is closed), which powers the DC Motor to rotate and simultaneously powers the LED to turn ON. If you press the Push Button Switch again, the Relay is deactivated (the internal switch opens), the Motor STOPS and the LED turns OFF.

The following is the program code:

```python
1   from gpiozero import DigitalOutputDevice, Button
2   import time
3
4   relayPin = 17     # define the relayPin
5   buttonPin = 18    # define the buttonPin
6   relay = DigitalOutputDevice(relayPin)     # define LED pin according to BCM Numbering
7   button = Button(buttonPin) # define Button pin according to BCM Numbering
8
9   def onButtonPressed():  # When button is pressed, this function will be executed
10      relay.toggle()
11      if relay.value :
12          print("Turn on relay ...")
13      else :
14          print("Turn off relay ... ")
15
16  def loop():
17      button.when_pressed = onButtonPressed
18      while True:
19          time.sleep(1)
20
21  def destroy():
22      relay.close()
23      button.close()
24
25  if __name__ == '__main__':     # Program entrance
26      print ('Program is starting...')
```

```
27        try:
28            loop()
29        except KeyboardInterrupt:    # Press ctrl-c to end the program.
30            destroy()
31            print("Ending program")
```

The project code is in the same as we used earlier in the Table Lamp project.
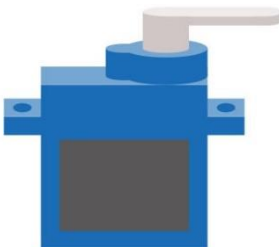
Chapter 15 Servo

Previously, we learned how to control the speed and rotational direction of a DC Motor. In this chapter, we will learn about Servos which are a rotary actuator type motor that can be controlled rotate to specific angles.

Project 15.1 Servo Sweep

First, we need to learn how to make a Servo rotate.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x3
Servo x1	

Component knowledge

Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.
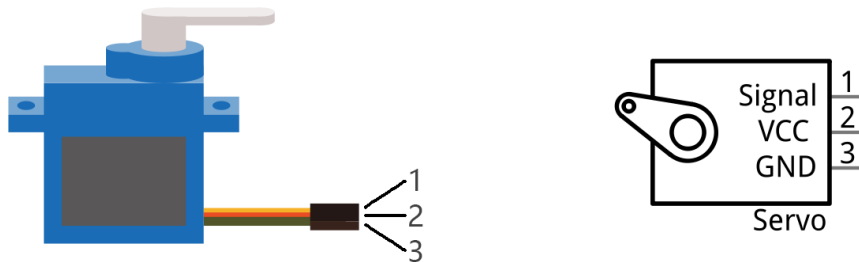
We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

Note: the lasting time of high level corresponding to the servo angle is absolute instead of accumulating. For example, the high level time lasting for 0.5ms correspond to the 0 degree of the servo. If the high level time lasts for another 1ms, the servo rotates to 45 degrees.

High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

When you change the Servo signal value, the Servo will rotate to the designated angle.

Circuit

Use caution when supplying power to the Servo it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram
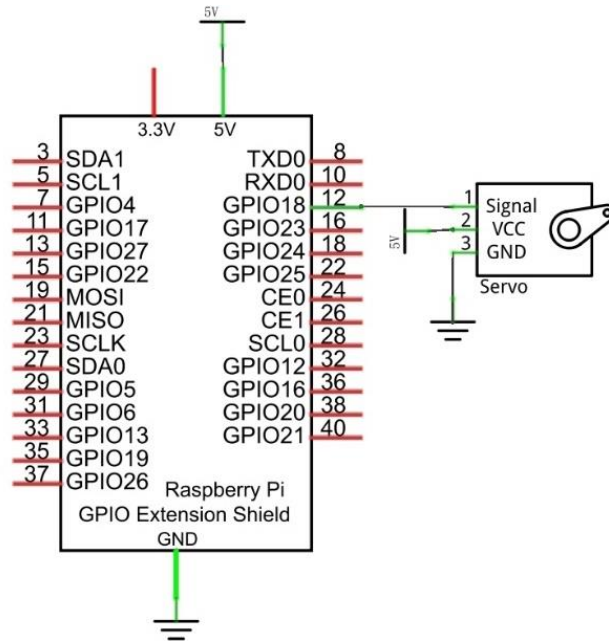
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
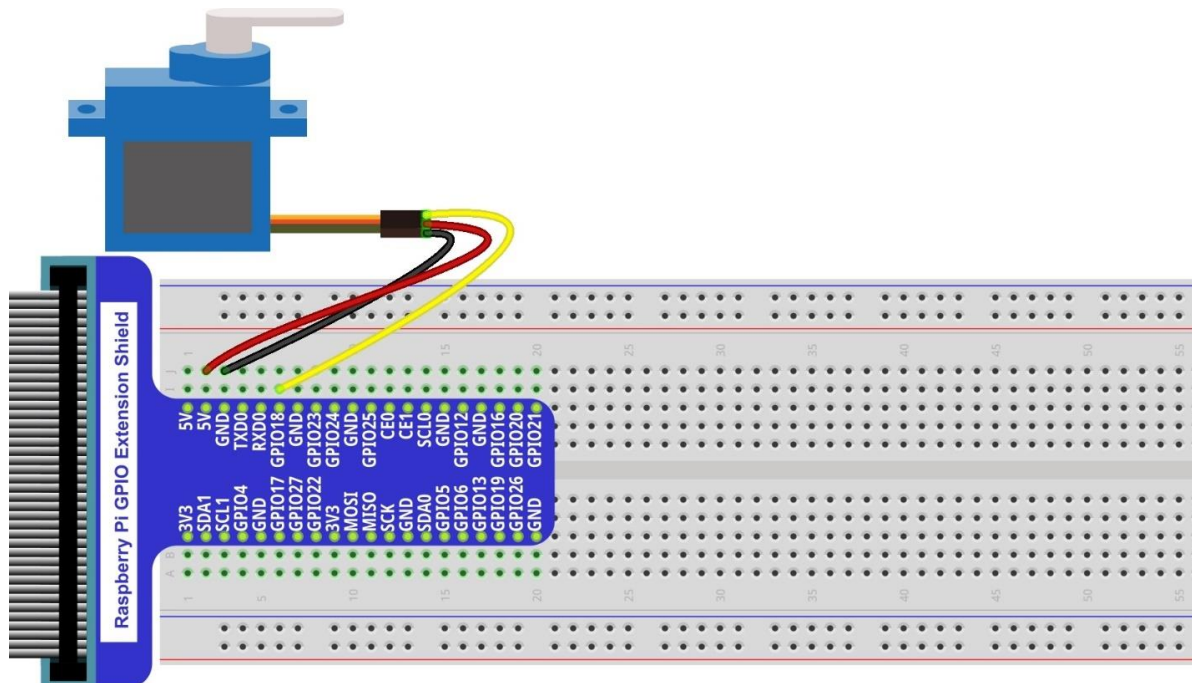
Code

In this project, we will make a Servo rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

Python Code 15.1.1 Sweep

First observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 15.1.1_Sweep directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/15.1.1_Sweep
```

2. Use python command to execute code "Sweep.py".

```
python Sweep.py
```

After the program is executed, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

```python
1   from gpiozero import AngularServo
2   import time
3
4   myGPIO=18
5   SERVO_DELAY_SEC = 0.001
6   myCorrection=0.0
7   maxPW=(2.5+myCorrection)/1000
8   minPW=(0.5-myCorrection)/1000
9   servo =  AngularServo(myGPIO,initial_angle=0,min_angle=0,
    max_angle=180,min_pulse_width=minPW,max_pulse_width=maxPW)
10
11  def loop():
12      while True:
13          for angle in range(0, 181, 1):    # make servo rotate from 0 to 180 deg
14              servo.angle = angle
15              time.sleep(SERVO_DELAY_SEC)
16          time.sleep(0.5)
17          for angle in range(180, -1, -1): # make servo rotate from 180 to 0 deg
18              servo.angle = angle
19              time.sleep(SERVO_DELAY_SEC)
20          time.sleep(0.5)
21
22  if __name__ == '__main__':     # Program entrance
23      print ('Program is starting...')
24      try:
25          loop()
26      except KeyboardInterrupt: # Press ctrl-c to end the program.
27          print("Ending program")
```

Import the AngularServo class that controls Servo from the gpiozero library.

```
from gpiozero import AngularServo
```

A 50 Hz pulse for a 20ms cycle is required to control the Servo. By default, the AngularServo class has set the control period to 20 milliseconds.

```
servo = AngularServo(myGPIO, initial_angle=0, min_angle=0,
max_angle=180, min_pulse_width=minPW, max_pulse_width=maxPW)
```

The 0-180 degree rotation of the servo corresponds to a PWM pulse width of 0.5-2.5ms at a period of 20ms and a duty cycle of 2.5%-12.5%. After setting the AngularServo class and passing in the corresponding angle parameters, the servo will turn to the corresponding position. However, in actual operation, as there is a deviation in the width of the servo pulse, we need to define minimum and maximum pulse width and error offset (this is essential in robotics).

```
myCorrection=0.0 #define pulse offset of servo
maxPW=(2.5+myCorrection)/1000     #define pulse duty cycle for minimum angle of servo
minPW=(0.5-myCorrection)/1000     #define pulse duty cycle for maximum angle of s
servo = AngularServo(myGPIO, initial_angle=0, min_angle=0,
max_angle=180, min_pulse_width=minPW, max_pulse_width=maxPW)
......
OFFSE_DUTY = 0.5         #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of s ervo
......
        for angle in range(0, 181, 1):   # make servo rotate from 0 to 180 deg
            servo.angle = angle
            time.sleep(SERVO_DELAY_SEC)
        time.sleep(0.5)
```

Finally, in the "while" cycle of main function, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

```
def loop():
    while True:
        for angle in range(0, 181, 1):   # make servo rotate from 0 to 180 deg
            servo.angle = angle
            time.sleep(SERVO_DELAY_SEC)
        time.sleep(0.5)
        for angle in range(180, -1, -1): # make servo rotate from 180 to 0 deg
            servo.angle = angle
            time.sleep(SERVO_DELAY_SEC)
        time.sleep(0.5)
```

For more information about the methods used by the AngularServo class in the GPIO Zero library,please refer to:https://gpiozero.readthedocs.io/en/stable/api_output.html#angularservo

In the above experiment, you can see that the servo jitter obviously when working.

Note: To reduce servo jitter, use the pigpio pin driver rather than the default RPi.GPIO driver (pigpio uses DMA sampling for much more precise edge timing).

You can refer to the code sweep2.py for more details.

1. Use cd command to enter 15.1.1_Sweep directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/15.1.1_Sweep
```

2. Use python command to execute code "Sweep.py".

```
python Sweep2.py
```

After the program executes, the servo will rotate from 0 degrees to 180 degrees, then reverse the direction so it rotates from 180 degrees to 0 degrees, and repeat these actions in an infinite loop. At this time, the steering gear can hardly feel the vibration.

This code is based on pigpio. In the latest Raspberry Pi OS, "pigpio" library has been installed. You only need to run the command to enable it.

```
sudo pigpiod
```

```
pi@raspberrypi:~ $ sudo pigpiod
pi@raspberrypi:~ $
```

If the "pigpio" library has not yet been installed, please follow the steps to install it.

Run the command to install "pigpio" library.

```
sudo apt-get update
sudo apt-get install pigpio python-pigpio python3-pigpio
```

The following is the program code:

```
1   import os
2   os.system("sudo pigpiod")
3   from gpiozero import AngularServo
4   from gpiozero.pins.pigpio import PiGPIOFactory
5   import time
6
7   my_factory = PiGPIOFactory()
8   myGPIO=18
9   SERVO_DELAY_SEC = 0.001
    myCorrection=0.0
10  maxPW=(2.5+myCorrection)/1000
11  minPW=(0.5-myCorrection)/1000
12  servo =  AngularServo(myGPIO,initial_angle=0,min_angle=0,
13  max_angle=180,min_pulse_width=minPW,max_pulse_width=maxPW,pin_factory=my_factory)
14
15  def loop():
16      while True:
17          for angle in range(0, 181, 1):    # make servo rotate from 0 to 180 deg
18              servo.angle = angle
19              time.sleep(SERVO_DELAY_SEC)
```

```
20              time.sleep(0.5)
21              for angle in range(180, -1, -1): # make servo rotate from 180 to 0 deg
22                  servo.angle = angle
23                  time.sleep(SERVO_DELAY_SEC)
24              time.sleep(0.5)
25
26  if __name__ == '__main__':      # Program entrance
27      print ('Program is starting...')
28      try:
29          loop()
30      except KeyboardInterrupt:  # Press ctrl-c to end the program.
31          servo.close()
32          os.system("sudo killall pigpiod")
33          print("Ending program")
```

The following values, and the corresponding Factory and Pin classes are listed in the table below. Factories are listed in the order that they are tried by default.

Name	Factory class	Pin class
rpigpio	gpiozero.pins.rpigpio.RPiGPIOFactory	gpiozero.pins.rpigpio.RPiGPIOPin
lgpio	gpiozero.pins.lgpio.LGPIOFactory	gpiozero.pins.lgpio.LGPIOPin
rpio	gpiozero.pins.rpio.RPIOFactory	gpiozero.pins.rpio.RPIOPin
pigpio	gpiozero.pins.pigpio.PiGPIOFactory	gpiozero.pins.pigpio.PiGPIOPin
native	gpiozero.pins.native.NativeFactory	gpiozero.pins.native.NativePin

See Changing the pin factory for further information：
https://gpiozero.readthedocs.io/en/stable/api_pins.html#changing-pin-factory
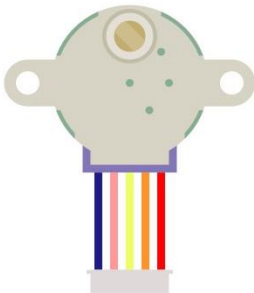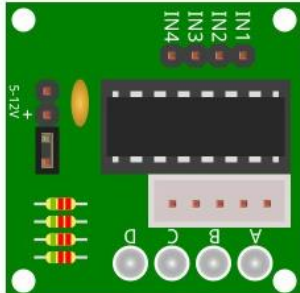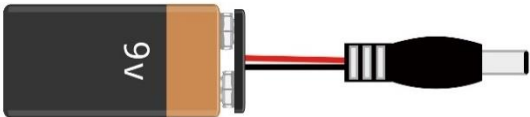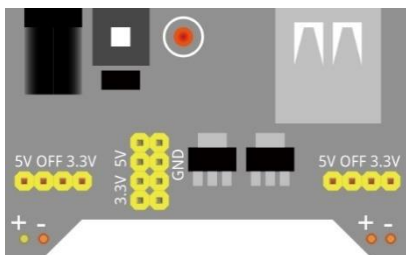
Chapter 16 Stepper Motor

Thus far, we have learned about DC Motors and Servos. A DC motor can rotate constantly in on direction but we cannot control the rotation to a specific angle. On the contrary, a Servo can rotate to a specific angle but cannot rotate constantly in one direction. In this chapter, we will learn about a Stepper Motor which is also a type of motor. A Stepper Motor can rotate constantly and also to a specific angle. Using a Stepper Motor can easily achieve higher accuracies in mechanical motion.

Project 16.1 Stepper Motor

In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x12
Stepper Motor x1	ULN2003 Stepper Motor Driver x1
9V battery (prepared by yourself) & battery line	Breadboard Power module x1

Component knowledge

Stepper Motor

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:
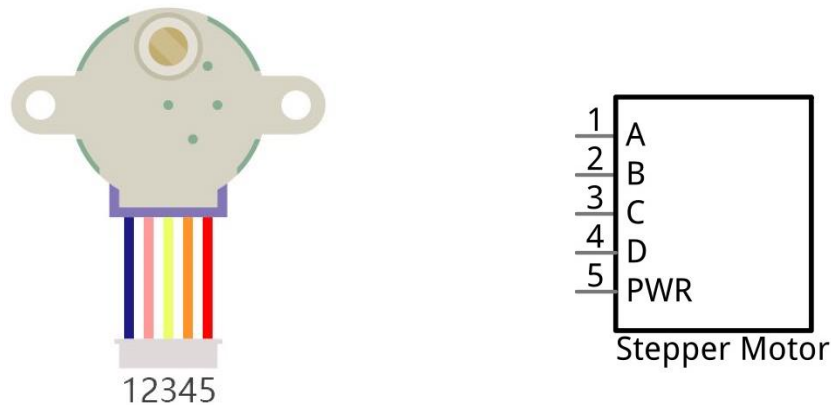
The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:
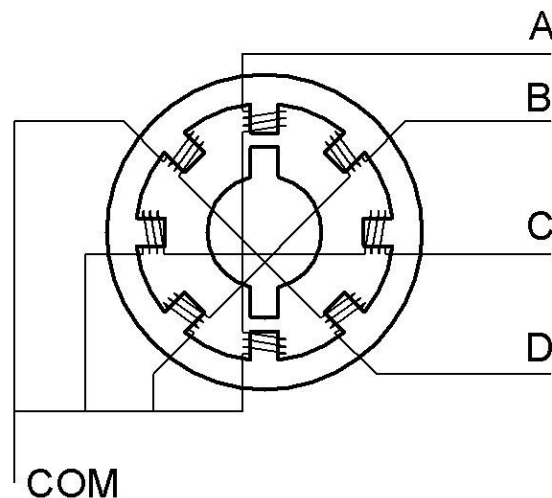
The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There is a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor's surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving sequence is shown here:
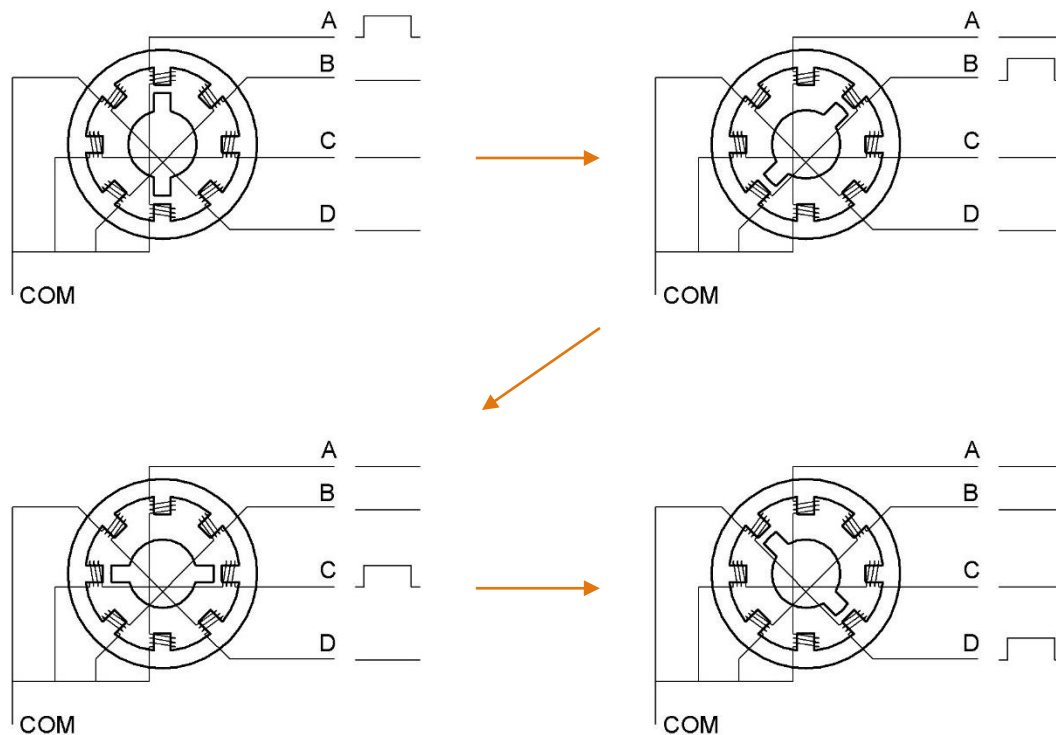
In the sequence above, the Stepper Motor rotates by a certain angle at once, which is called a "step". By controlling the number of rotational steps, you can then control the Stepper Motor's rotation angle. By defining the time between two steps, you can control the Stepper Motor's rotation speed. When rotating clockwise, the order of coil powered on is: A → B → C → D → A →······ . And the rotor will rotate in accordance with this order, step by step, called four-steps, four-part. If the coils is powered ON in the reverse order, D → C → B → A → D →··· , the rotor will rotate in counter-clockwise direction.

There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. Tise sequence of powering the coils looks like this: A → AB → B → BC → C → CD → D → DA → A →······, the rotor will rotate in accordance to this sequence ar, a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the Stepper Motor will rotate in the opposite direction.

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor's housing) requires 32 X 64 = 2048 steps to make one full revolution.

ULN2003 Stepper Motor driver

A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.
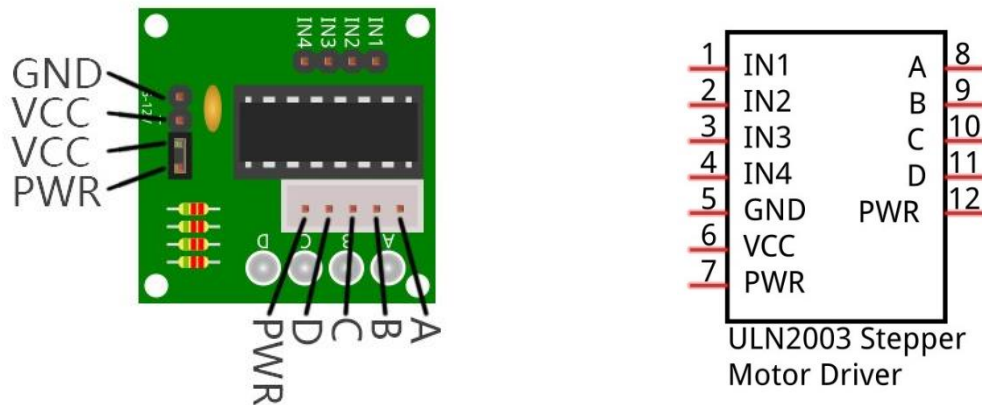
Circuit

When building the circuit, note that rated voltage of the Stepper Motor is 5V, and we need to use the breadboard power supply independently, (**Caution do not use the RPi power supply**). Additionally, the breadboard power supply needs to share Ground with Rpi.

Schematic diagram
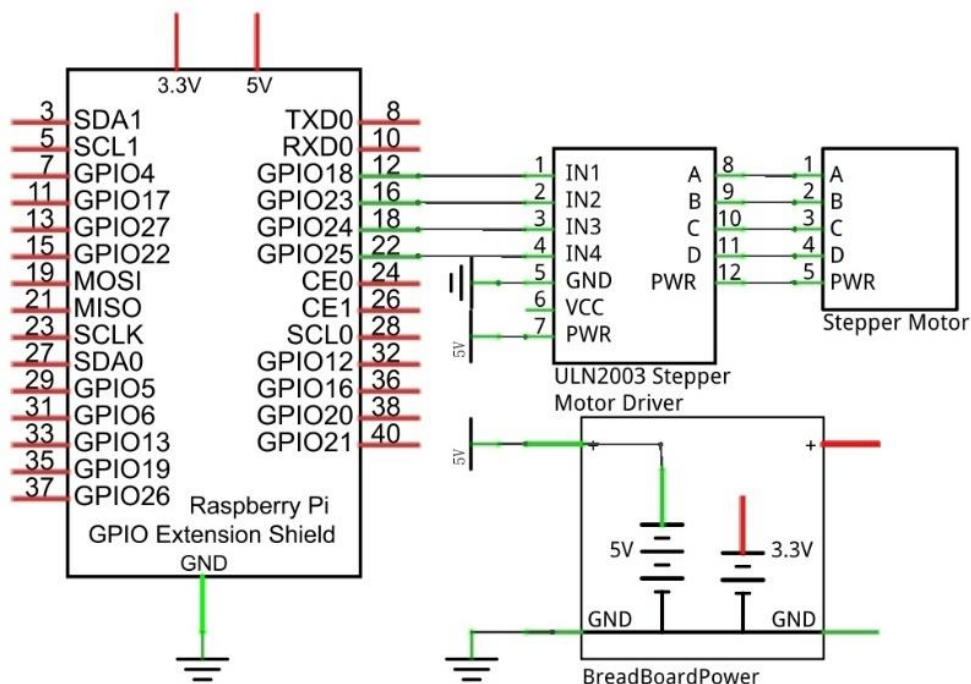
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
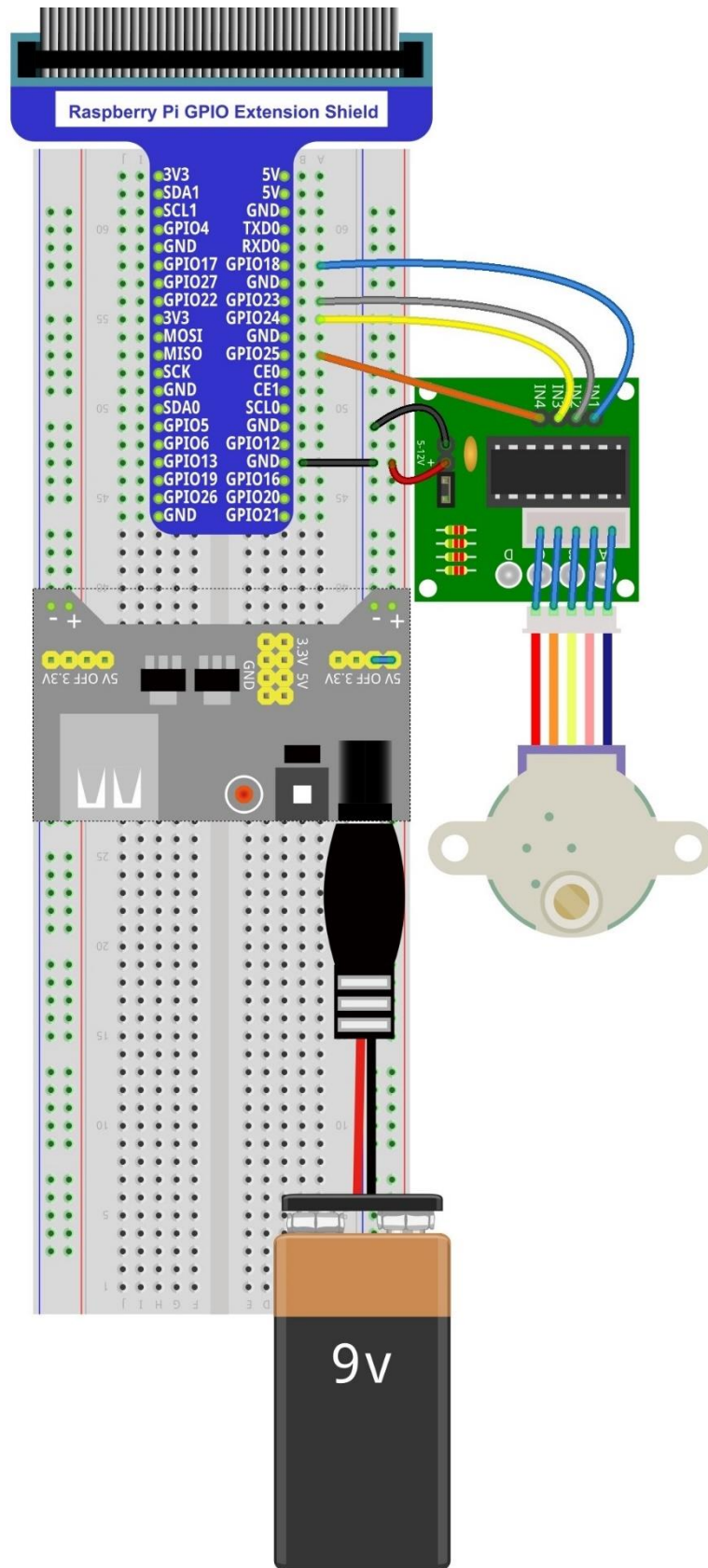
Code

Python Code 16.1.1 SteppingMotor

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 16.1.1_SteppingMotor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/16.1.1_SteppingMotor
```

2. Use Python command to execute code "SteppingMotor.py".

```
python SteppingMotor.py
```

After the program is executed, the Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

```python
1   from gpiozero import OutputDevice
2   import time
3
4   motorPins = (18, 23, 24, 25) # define pins connected to four phase ABCD of stepper motor
5   # motorPins = ("J8:12", "J8:16", "J8:18", "J8:22") # define pins connected to four phase ABCD
    of stepper motor
6   motors = list(map(lambda pin: OutputDevice(pin), motorPins))
7   CCWStep = (0x01,0x02,0x04,0x08) # define power supply order for rotating anticlockwise
8   CWStep = (0x08,0x04,0x02,0x01)  # define power supply order for rotating clockwise
9
10  # as for four phase stepping motor, four steps is a cycle. the function is used to drive the
    stepping motor clockwise or anticlockwise to take four steps
11  def moveOnePeriod(direction,ms):
12      for j in range(0,4,1):      # cycle for power supply order
13          for i in range(0,4,1):  # assign to each pin
14              if (direction == 1):# power supply order clockwise
15                  motors[i].on() if (CCWStep[j] == 1<<i) else motors[i].off()
16              else :              # power supply order anticlockwise
17                  motors[i].on() if CWStep[j] == 1<<i else motors[i].off()
18          if(ms<3):        # the delay can not be less than 3ms, otherwise it will exceed speed
    limit of the motor
19              ms = 3
20          time.sleep(ms*0.001)
21
22  # continuous rotation function, the parameter steps specifies the rotation cycles, every four
    steps is a cycle
23  def moveSteps(direction, ms, steps):
24      for i in range(steps):
25          moveOnePeriod(direction, ms)
26
27  # function used to stop motor
```

```
28  def motorStop():
29      for i in range(0,4,1):
30          motors.off()
31
32  def loop():
33      while True:
34          moveSteps(0,3,512)   # rotating 360 deg clockwise, a total of 2048 steps in a circle,
512 cycles
35          time.sleep(0.5)
36          moveSteps(1,3,512)   # rotating 360 deg anticlockwise
37          time.sleep(0.5)
38
39  if __name__ == '__main__':     # Program entrance
40      print ('Program is starting...')
41      try:
42          loop()
43      except KeyboardInterrupt:  # Press ctrl-c to end the program.
44          print("Ending program")
```

In the code we define the four pins of the Stepper Motor and the order to supply power to the coils for a four-step rotation mode.

```
motorPins = (18, 23, 24, 25) # define pins connected to four phase ABCD of stepper motor
# motorPins = ("J8:12", "J8:16", "J8:18", "J8:22") # define pins connected to four phase ABCD
of stepper motor
motors = list(map(lambda pin: OutputDevice(pin), motorPins))
CCWStep = (0x01,0x02,0x04,0x08) # define power supply order for rotating anticlockwise
CWStep = (0x08,0x04,0x02,0x01)   # define power supply order for rotating clockwise
```

Subfunction **moveOnePeriod** ((int dir, int ms) will drive the Stepper Motor rotating four-step clockwise or anticlockwise, four-step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

```
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):       # cycle for power supply order
        for i in range(0,4,1):   # assign to each pin
            if (direction == 1):# power supply order clockwise
                motors[i].on() if (CCWStep[j] == 1<<i) else motors[i].off()
            else :               # power supply order anticlockwise
                motors[i].on() if CWStep[j] == 1<<i else motors[i].off()
        if(ms<3):        # the delay can not be less than 3ms, otherwise it will exceed speed
limit of the motor
            ms = 3
        time.sleep(ms*0.001)
```

Subfunction **moveSteps** (direction, ms, steps) is used to specify the cycle number of Stepper Motor.

```
def moveSteps(direction, ms, steps):
```

```
        for i in range(steps):
            moveOnePeriod(direction, ms)
```

Subfunction **motorStop** () is used to stop the Stepper Motor.

```
    def motorStop():
        for i in range(0, 4, 1):
            motors.off()
```

Finally, in the while loop of main function, rotate one revolution clockwise, and then one revolution anticlockwise. According to the previous material covered, the Stepper Motor one revolution requires 2048 steps, that is, 2048/4=512 cycle.

```
    while True:
        moveSteps(0, 3, 512)   # rotating 360 deg clockwise, a total of 2048 steps in a circle,
512 cycles
        time.sleep(0.5)
        moveSteps(1, 3, 512)   # rotating 360 deg anticlockwise
        time.sleep(0.5)
```

For more information about the methods used by the OutputDevice class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_output.html#outputdevice
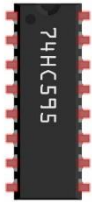
Chapter 17 74HC595 & Bar Graph LED

We have used LED Bar Graph to make a flowing water light, in which 10 GPIO ports of RPi are occupied. More GPIO ports mean that more peripherals can be connected to RPi, so GPIO resource is very precious. Can we make flowing water light with less GPIO ports? In this chapter, we will learn a component, 74HC595, which can achieve the target.

Project 17.1 Flowing Water Light

Now let us learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper x17	
74HC595 x1	Bar Graph LED x1		Resistor 220Ω x8

Component knowledge

74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of a Raspberry Pi. At least 3 ports on the RPI board are required to control the 8 ports of the 74HC595 chip.
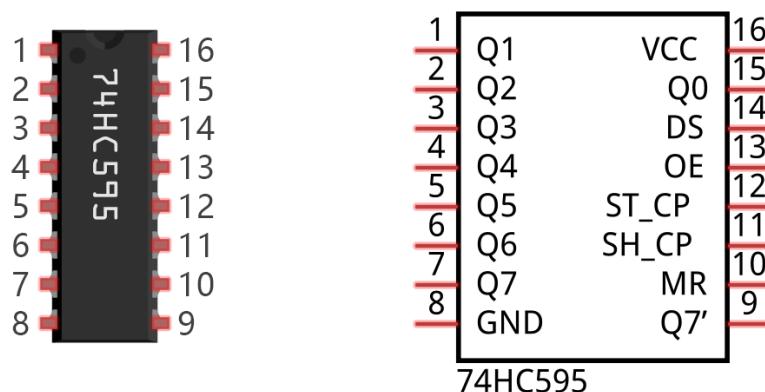
The ports of the 74HC595 chip are described as follows:

Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel Data Output
VCC	16	The Positive Electrode of the Power Supply, the Voltage is 2~6V
GND	8	The Negative Electrode of Power Supply
DS	14	Serial Data Input
OE	13	Enable Output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel Update Output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial Shift Clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove Shift Register: When this pin is in low level, the content in shift register will be cleared.
Q7'	9	Serial Data Output: it can be connected to more 74HC595 chips in series.

For more details, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram
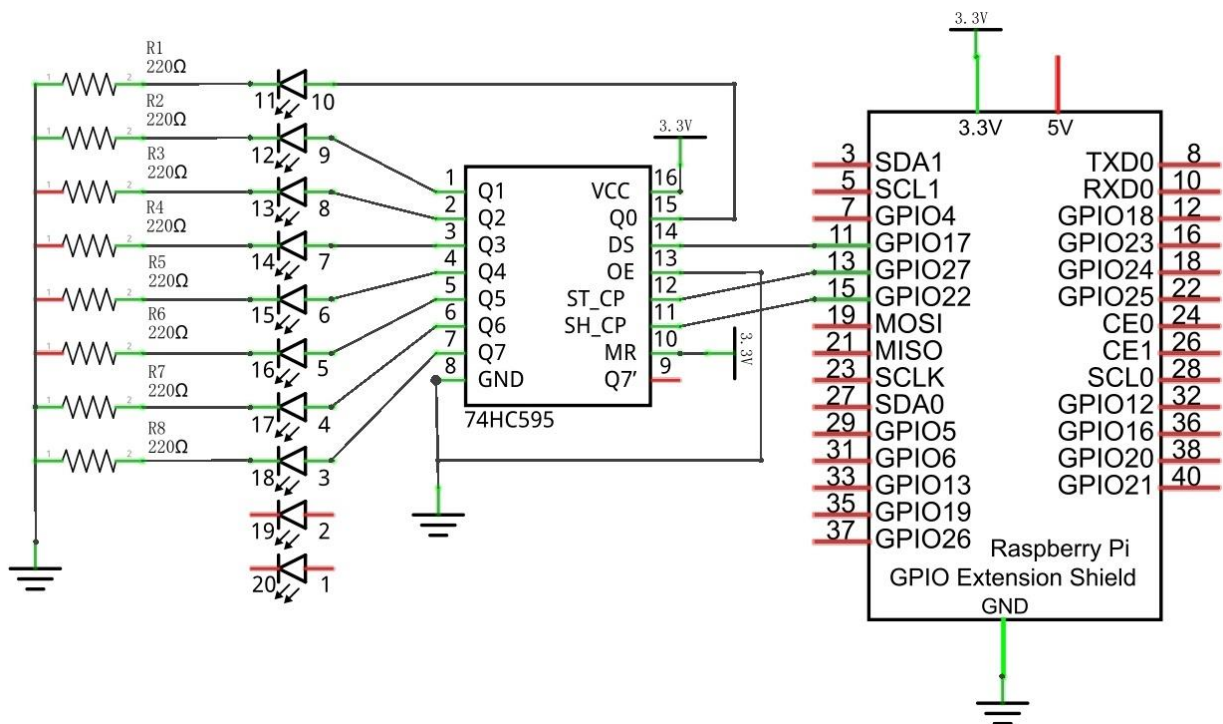
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
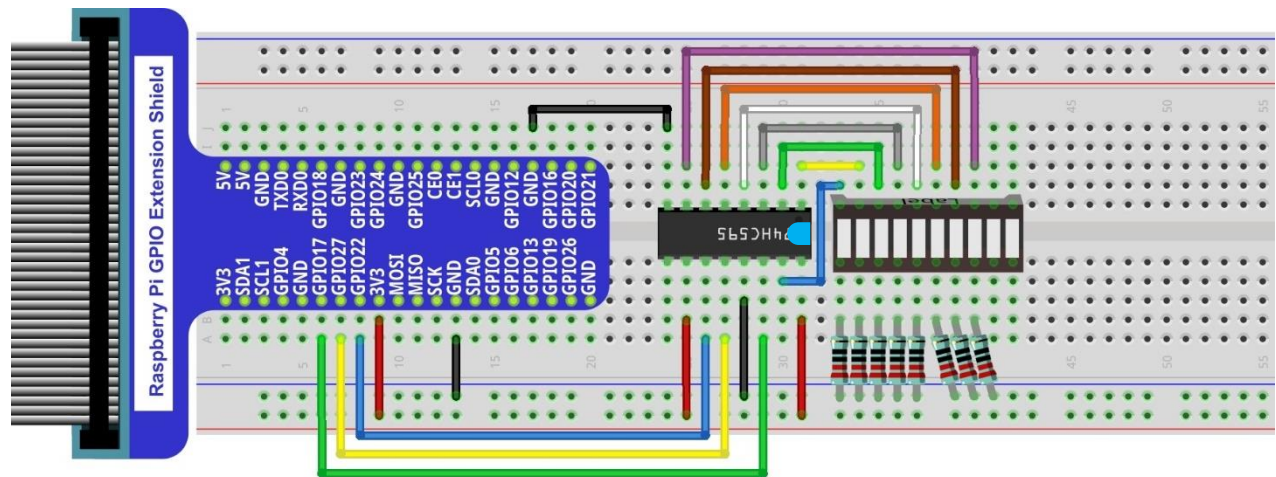
Code

In this project we will make a flowing water light with a 74HC595 chip to learn about its functions.

Python Code 17.1.1 LightWater02

First, observe the project result, and then learn about the code in detail.
If you have any concerns, please contact us via: support@freenove.com
1. Use cd command to enter 17.1.1_LightWater02 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/17.1.1_LightWater02
```

2. Use python command to execute Python code "LightWater02.py".

```
python LightWater02.py
```

After the program is executed, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.
The following is the program code:

```python
1   from gpiozero import OutputDevice
2   import time
3   # Defines the data bit that is transmitted preferentially in the shiftOut function.
4   LSBFIRST = 1
5   MSBFIRST = 2
6   # define the pins for 74HC595
7   dataPin   = OutputDevice(17)      # DS Pin of 74HC595(Pin14)
8   latchPin  = OutputDevice(27)      # ST_CP Pin of 74HC595(Pin12)
9   clockPin  = OutputDevice(22)      # CH_CP Pin of 74HC595(Pin11)
10
11  # shiftOut function, use bit serial transmission.
12  def shiftOut(order,val):
13      for i in range(0,8):
14          clockPin.off()
15          if(order == LSBFIRST):
16              dataPin.on() if (0x01&(val>>i)==0x01) else dataPin.off()
17          elif(order == MSBFIRST):
18              dataPin.on() if (0x80&(val<<i)==0x80) else dataPin.off()
19          clockPin.on()
20
21  def loop():
22      while True:
23          x=0x01
24          for i in range(0,8):
25              latchPin.off()# Output low level to latchPin
26              shiftOut(LSBFIRST,x) # Send serial data to 74HC595
27              latchPin.on()    # Output high level to latchPin, and 74HC595 will update the data
28  to the parallel output port.
29              x<<=1 # make the variable move one bit to left once, then the bright LED move one
30  step to the left once.
```

```
31              time.sleep(0.1)
32         x=0x80
33         for i in range(0,8):
               latchPin.off()
34             shiftOut(LSBFIRST,x)
               latchPin.on()
35             x>>=1
36             time.sleep(0.1)
37
38 def destroy():
39     dataPin.close()
40     latchPin.close()
41     clockPin.close()
42
43 if __name__ == '__main__': # Program entrance
44     print ('Program is starting...' )
45     try:
46         loop()
47     except KeyboardInterrupt:   # Press ctrl-c to end the program.
48         destroy()
49         print("Ending program")
```

Import the OutputDevice class that controls the 74HC595 chip from the gpiozero library.

```
from gpiozero import OutputDevice
```

Create the OutputDevice class for controlling the 74HC595 chip.

```
dataPin   = OutputDevice(17)        # DS Pin of 74HC595(Pin14)
latchPin  = OutputDevice(27)        # ST_CP Pin of 74HC595(Pin12)
clockPin  = OutputDevice(22)        # CH_CP Pin of 74HC595(Pin11)
```

In the code, we define a shiftOut() function, which is used to output values with bits in order, where the dPin for the data pin, cPin for the clock and order for the priority bit flag (high or low). This function conforms to the operational modes of the 74HC595. LSBFIRST and MSBFIRST are two different flow directions.

```
# shiftOut function, use bit serial transmission.
def shiftOut(order,val):
    for i in range(0,8):
        clockPin.off()
        if(order == LSBFIRST):
            dataPin.on() if (0x01&(val>>i)==0x01) else dataPin.off()
        elif(order == MSBFIRST):
            dataPin.on() if (0x80&(val<<i)==0x80) else dataPin.off()
        clockPin.on()
```

In the loop() function, we use two cycles to achieve the action goal. First, define a variable x=0x01, binary 00000001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then an LED turns ON. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED

that turns ON will be shifted. Repeat the operation, over and over and the effect of a flowing water light will
be visible. If the direction of the shift operation for x is different, the flowing direction is different.

```python
def loop():
    while True:
        x=0x01
        for i in range(0,8):
            latchPin.off()# Output low level to latchPin
            shiftOut(LSBFIRST,x) # Send serial data to 74HC595
            latchPin.on()   # Output high level to latchPin, and 74HC595 will update the data
to the parallel output port.
            x<<=1 # make the variable move one bit to left once, then the bright LED move one
step to the left once.
            time.sleep(0.1)
        x=0x80
        for i in range(0,8):
            latchPin.off()
            shiftOut(LSBFIRST,x)
            latchPin.on()
            x>>=1
            time.sleep(0.1)
```

For more information about the methods used by the OutputDevice class in the GPIO Zero library,please refer
to: https://gpiozero.readthedocs.io/en/stable/api_output.html#outputdevice
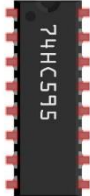
Chapter 18 74HC595 & 7-Segment Display

In this chapter, we will introduce the 7-Segment Display.

Project 18.1 7-Segment Display

We will use a 74HC595 IC Chip to control a 7-Segment Display and make it display sixteen decimal characters "0" to "F".

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x18	
74HC595 x1	7-Segment Display x1	Resistor 220Ω x8

Component knowledge

7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:
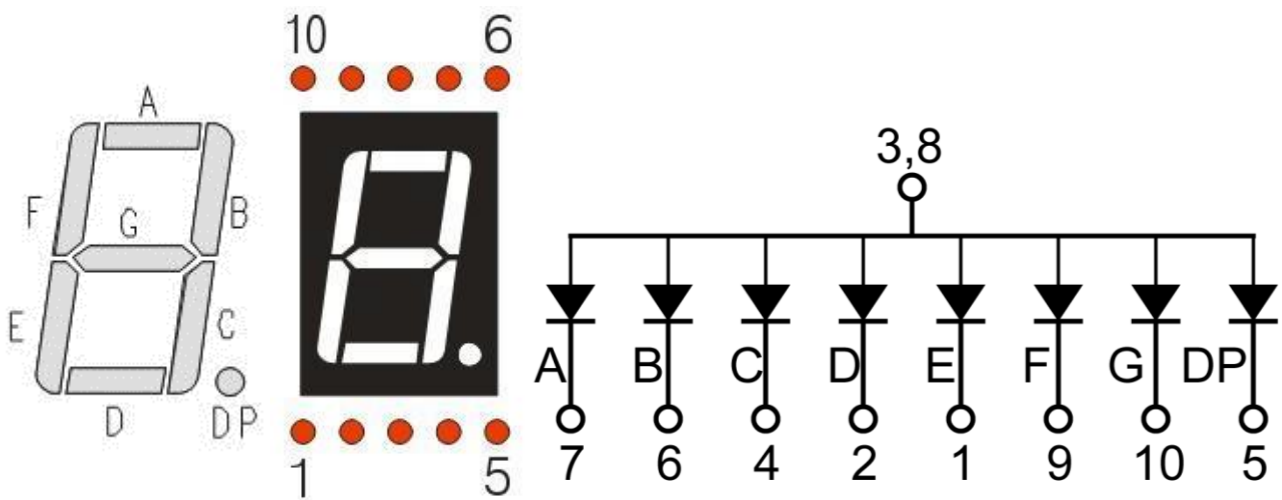
As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a "0": we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.

In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment "A" as the lowest level and segment "DP" as the highest level, from high to low would look like this: "DP", "G", "F", "E", "D", "C", "B", "A". Character "0" corresponds to the code: 1100 0000b=0xc0.

Circuit

Schematic diagram
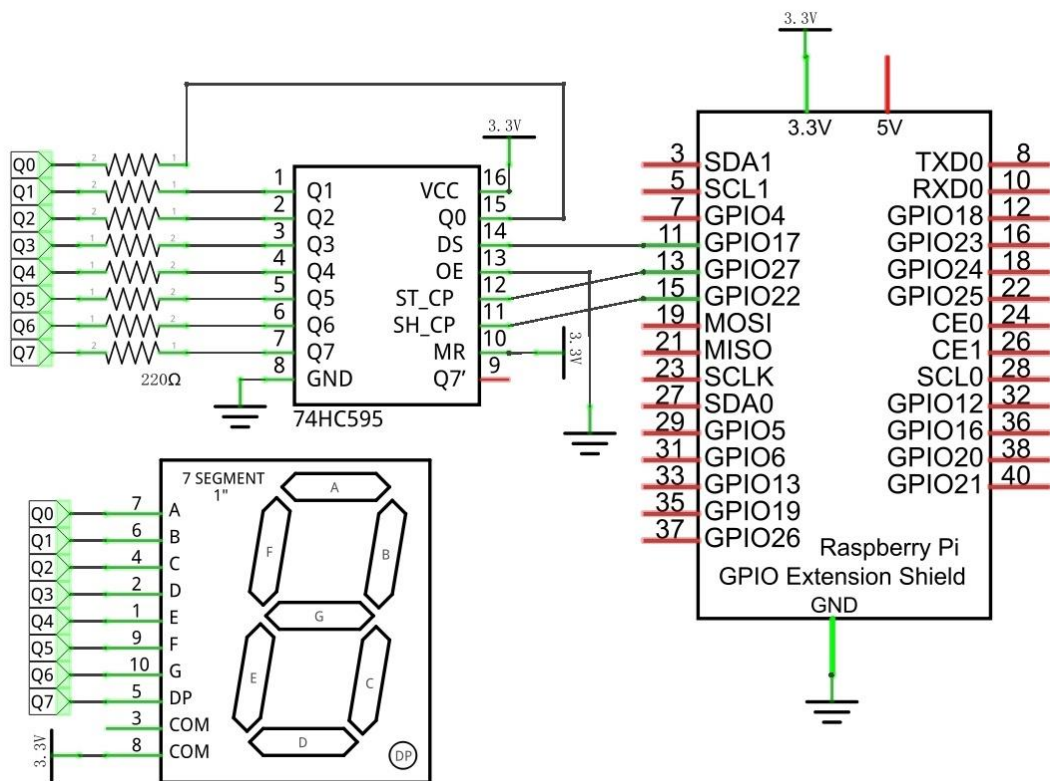
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
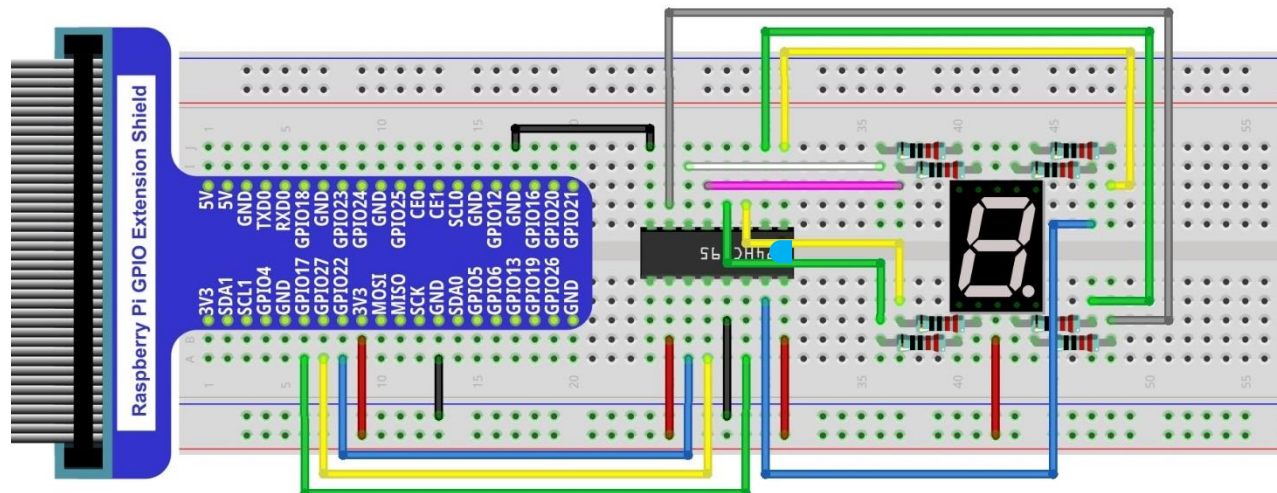
Code

This code uses a 74HC595 IC Chip to control the 7-Segment Display. The use of the 74HC595 IC Chip is

generally the same throughout this Tutorial. We need code to display the characters "0" to "F" one character at a time, and then output to display them with the 74HC595 IC Chip.

Python Code 18.1.1 SevenSegmentDisplay

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 18.1.1_SevenSegmentDisplay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/18.1.1_SevenSegmentDisplay
```

2. Use Python command to execute Python code "SevenSegmentDisplay.py".

```
python SevenSegmentDisplay.py
```

After the program is executed, the 7-Segment Display starts to display the characters "0" to "F" in succession. The following is the program code:

```python
1    from gpiozero import OutputDevice
2    import time
3
4    LSBFIRST = 1
5    MSBFIRST = 2
6    # define the pins for 74HC595
7    dataPin   = OutputDevice(17)      # DS Pin of 74HC595(Pin14)
8    latchPin  = OutputDevice(27)      # ST_CP Pin of 74HC595(Pin12)
9    clockPin  = OutputDevice(22)      # CH_CP Pin of 74HC595(Pin11)
10   # SevenSegmentDisplay display the character "0"- "F" successively
11   num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
12
13   def shiftOut(order, val):
14       for i in range(0, 8):
15           clockPin.off()
16           if(order == LSBFIRST):
17               dataPin.on() if (0x01&(val>>i)==0x01) else dataPin.off()
18           elif(order == MSBFIRST):
19               dataPin.on() if (0x80&(val<<i)==0x80) else dataPin.off()
20           clockPin.on()
21
22   def loop():
23       while True:
24           for i in range(0, len(num)):
25               latchPin.off()
26               shiftOut(MSBFIRST, num[i])   # Send serial data to 74HC595
27               latchPin.on()
28               time.sleep(0.5)
29           for i in range(0, len(num)):
30               latchPin.off()
31               shiftOut(MSBFIRST, num[i]&0x7f) # Use "&0x7f" to display the decimal point.
32               latchPin.on()
33               time.sleep(0.5)
```

```
34
35   def destroy():
36       dataPin.close()
37       latchPin.close()
38       clockPin.close()
39
40   if __name__ == '__main__': # Program entrance
41       print ('Program is starting...' )
42       try:
43           loop()
44       except KeyboardInterrupt:   # Press ctrl-c to end the program.
45           destroy()
46           print("Ending program")
```

First, we need to create encoding for characters "0" to "F" in the array.

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

In the "for" loop of loop() function, use the 74HC595 IC Chip to output contents of array "num" successively. SevenSegmentDisplay can then correctly display the corresponding characters. Pay attention to this in regard to shiftOut function, the transmission bit, flag bit amd highest bit will be transmitted preferentially.

```
for i in range(0, len(num)):
    latchPin.off()
    shiftOut(MSBFIRST, num[i]) #Output the figures and the highest level is transfered
preferentially.
    latchPin.on()
    time.sleep(0.5)
```

If you want to display the decimal point, make the highest bit of each array "0", which can be implemented easily by num[i]&0x7f.

```
shiftOut(MSBFIRST, num[i]&0x7f) # Use "&0x7f" to display the decimal point.
```

For more information about the methods used by the OutputDevice class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_output.html#outputdevice

Project 18.2 4-Digit 7-Segment Display

Now, let's try to control more-than-one digit displays by using a Four 7-Segment Display in one project.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Wire x1 Breadboard x1		Jumper Wire x30		
74HC595 x1	PNP transistor x4	4-Digit 7-Segment Display x1	Resistor 220Ω x8	Resistor 1KΩ x4

Component knowledge

4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-Segment Displays into one module, therefore it can display more characters. All of the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:
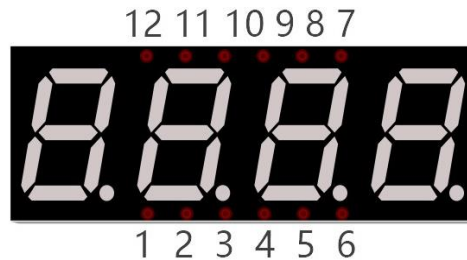
The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-Segment Display are connected together.
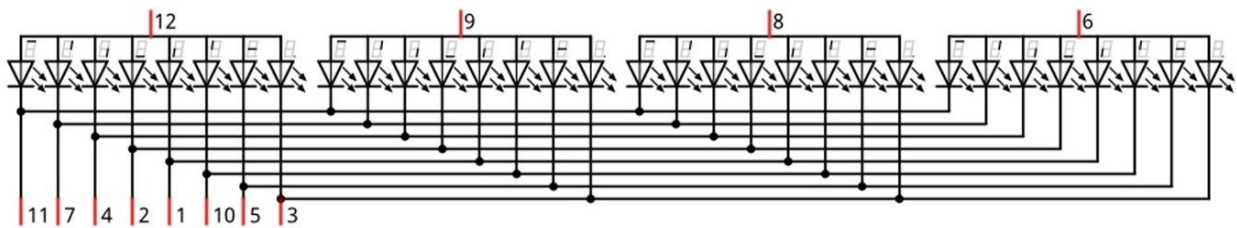
Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so very fast that it is unperceivable to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

Circuit

Schematic diagram
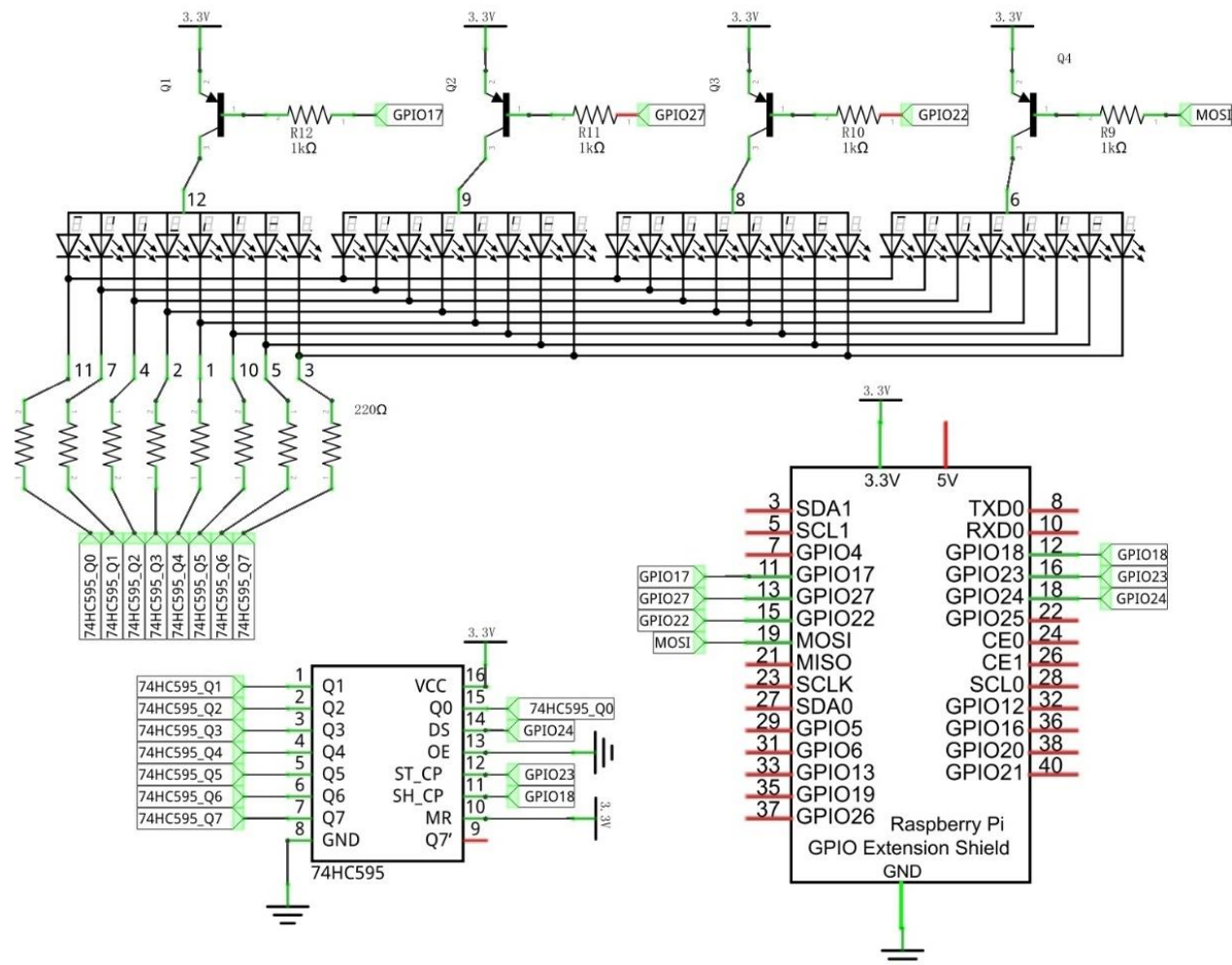
Hardware connection
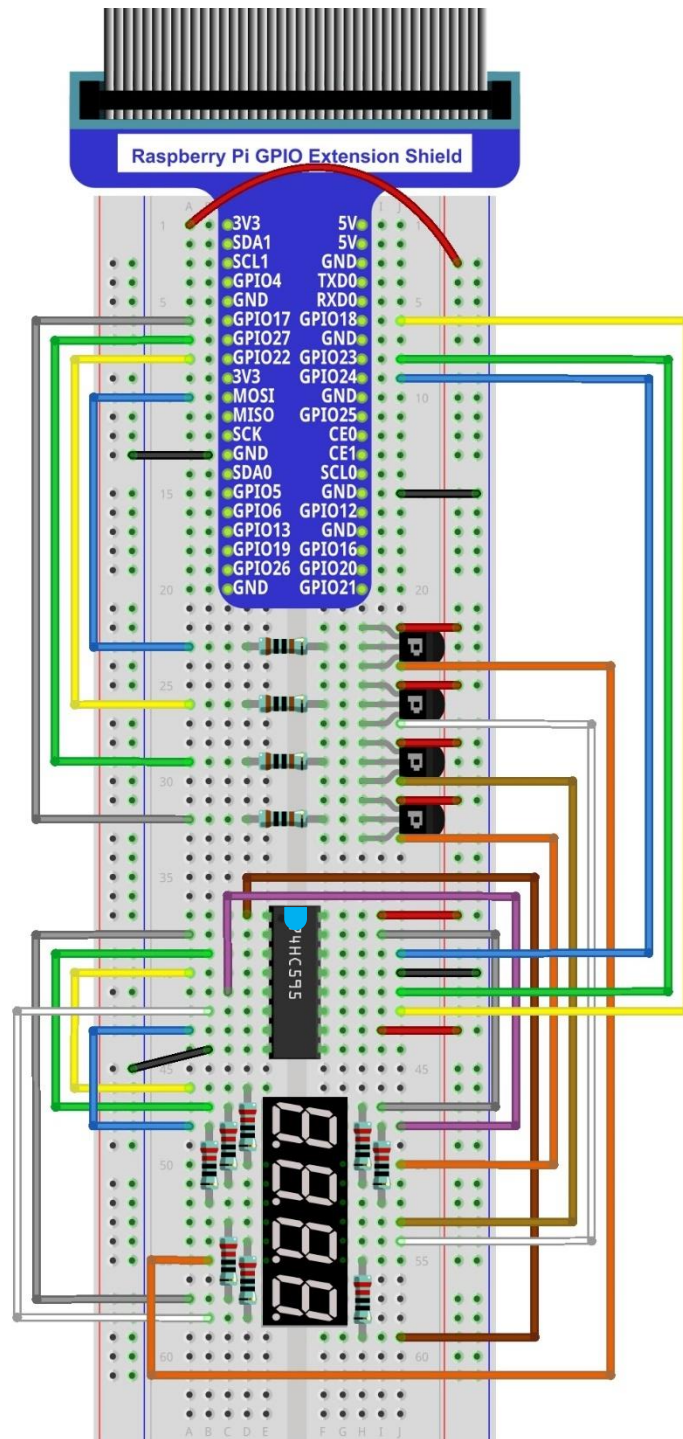
Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

Python Code 18.2.1 StopWatch

This code uses the four step four pat mode to drive the Stepper Motor clockwise and reverse direction.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 18.2.1_StopWatch directory of Python code.

`cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/18.2.1_StopWatch`

2. Use python command to execute code "StopWatch.py".

`python StopWatch.py`

After the program is executed, 4-Digit 7-segment start displaying a four-digit number dynamically, and the will plus 1 in each successive second.

The following is the program code:

```python
from gpiozero import OutputDevice
import time
import threading

LSBFIRST = 1
MSBFIRST = 2
# define the pins connect to 74HC595
dataPin   = OutputDevice(24)      # DS Pin of 74HC595
latchPin  = OutputDevice(23)      # ST_CP Pin of 74HC595
clockPin  = OutputDevice(18)      # SH_CP Pin of 74HC595
num = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
digitPin = (17, 27, 22, 10)     # Define the pin of 7-segment display common end
outputs = list(map(lambda pin: OutputDevice(pin), digitPin))
counter = 0           # Variable counter, the number will be dislayed by 7-segment display
t = 0                 # define the Timer object

def shiftOut(order, val):
    for i in range(0, 8):
        clockPin.off()
        if(order == LSBFIRST):
            dataPin.on() if (0x01&(val>>i)==0x01) else dataPin.off()
        elif(order == MSBFIRST):
            dataPin.on() if (0x80&(val<<i)==0x80) else dataPin.off()
        clockPin.on()

def outData(data):      # function used to output data for 74HC595
    latchPin.off()
    shiftOut(MSBFIRST, data)
    latchPin.on()
```

```
30
31   def selectDigit(digit): # Open one of the 7-segment display and close the remaining three, the
32   parameter digit is optional for 1,2,4,8
33       outputs[0].off() if ((digit&0x08) == 0x08) else outputs[0].on()
34       outputs[1].off() if ((digit&0x04) == 0x04) else outputs[1].on()
35       outputs[2].off() if ((digit&0x02) == 0x02) else outputs[2].on()
36       outputs[3].off() if ((digit&0x01) == 0x01) else outputs[3].on()
37
38   def display(dec):   # display function for 7-segment display
39       outData(0xff)   # eliminate residual display
40       selectDigit(0x01)   # Select the first, and display the single digit
41       outData(num[dec%10])
42       time.sleep(0.003)   # display duration
43       outData(0xff)
44       selectDigit(0x02)   # Select the second, and display the tens digit
45       outData(num[dec%100//10])
46       time.sleep(0.003)
47       outData(0xff)
48       selectDigit(0x04)   # Select the third, and display the hundreds digit
49       outData(num[dec%1000//100])
50       time.sleep(0.003)
51       outData(0xff)
52       selectDigit(0x08)   # Select the fourth, and display the thousands digit
53       outData(num[dec%10000//1000])
54       time.sleep(0.003)
55   def timer():
56       global counter
57       global t
58       t = threading.Timer(1.0,timer)      # reset time of timer to 1s
59       t.start()                           # Start timing
60       counter+=1
61       print ("counter : %d"%counter)
62
63   def loop():
64       global t
65       global counter
66       t = threading.Timer(1.0,timer)      # set the timer
67       t.start()                           # Start timing
68       while True:
69           display(counter)                # display the number counter
70
71   def destroy():
72       global t
73       dataPin.close()
```

```
74        latchPin.close()
75        clockPin.close()
76        t.cancel()
77
78   if __name__ == '__main__': # Program entrance
79        print ('Program is starting...' )
80        try:
81            loop()
82        except KeyboardInterrupt:    # Press ctrl-c to end the program.
83            destroy()
84            print("Ending program")
```

First, define the pin of 74HC595 and 7-segment display common end, character encoding and a variable "counter" to be displayed counter.

```
# define the pins connect to 74HC595
dataPin   = OutputDevice (24)      # DS Pin of 74HC595
latchPin  = OutputDevice (23)      # ST_CP Pin of 74HC595
clockPin  = OutputDevice (18)      # SH_CP Pin of 74HC595
num = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
digitPin = (17,27,22,10)     # Define the pin of 7-segment display common end
outputs = list(map(lambda pin: OutputDevice(pin), digitPin))
counter = 0          # Variable counter, the number will be dislayed by 7-segment display
```

Subfunction **selectDigit** (digit) function is used to open one of the 7-segment display and close the other 7-segment display, where the parameter digit value can be 1,2,4,8. Using "|" can open a number of 7-segment display.

```
def selectDigit(digit): # Open one of the 7-segment display and close the remaining three, the
parameter digit is optional for 1,2,4,8
    outputs[0].off() if ((digit&0x08) == 0x08) else outputs[0].on()
    outputs[1].off() if ((digit&0x04) == 0x04) else outputs[1].on()
    outputs[2].off() if ((digit&0x02) == 0x02) else outputs[2].on()
    outputs[3].off() if ((digit&0x01) == 0x01) else outputs[3].on()
```

Subfunction **outData** (data) is used to make the 74HC595 output an 8-bit data immediately.

```
def outData(data):       # function used to output data for 74HC595
    latchPin.off()
    shiftOut(MSBFIRST,data)
    latchPin.on()
```

Subfunction **display** (int dec) is used to make a 4-Digit 7-Segment Display a 4-bit integer. First open the common end of first 7-Segment Display Digit and turn OFF the other three Digits, now it can be used as 1-Digit 7-Segment Display. The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively. Each digit will be displayed for a period by using **delay** (). The time in this code is very brief, so you will a mess of Digits. If the time is set long enough, you will see that every digit is displayed independently.

```
def display(dec):    #display function for 7-segment display
    outData(0xff)    #eliminate residual display
    selectDigit(0x01)   #Select the first, and display the single digit
```

```
    outData(num[dec%10])
    time.sleep(0.003)    #display duration
    outData(0xff)
    selectDigit(0x02)    #Select the second, and display the tens digit
    outData(num[dec%100/10])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x04)    #Select the third, and display the hundreds digit
    outData(num[dec%1000/100])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x08)    #Select the fourth, and display the thousands digit
    outData(num[dec%10000/1000])
    time.sleep(0.003)
```

Subfunction **timer** () is the timer callback function. When the time is up, this function will be executed. Accompanied by the execution, the variable counter will be added 1, and then reset the time of timer to 1s. 1s later, the function will be executed again.

```
def timer():         #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                           #Start timing
    counter+=1
    print ("counter : %d"%counter)
```

Subfunction **setup**(), configure all input output modes for the GPIO pin used.

Finally, in loop function, make the digital tube display variable counter value in the while loop. The value will change in function **timer** (), so the content displayed by 7-segment display will change accordingly.

```
def loop():
    global t
    global counter
    t = threading.Timer(1.0,timer)      # set the timer
    t.start()                           #Start timing
    while True:
        display(counter)               #display the number counter
```

After the program is executed, press "Ctrl+C", then subfunction destroy() will be executed, and GPIO resources and timers will be released in this subfunction.

```
def destroy():
    global t
    dataPin.close()
    latchPin.close()
    clockPin.close()
    t.cancel()
```

Chapter 19 74HC595 & LED Matrix

Thus far we have learned how to use the 74HC595 IC Chip to control the Bar Graph LED and the 7-Segment Display. We will now use 74HC595 IC Chips to control an LED Matrix.

Project 19.1 LED Matrix

In this project, we will use two 74HC595 IC chips to control a monochrome (one color) (8X8) LED Matrix to make it display both simple graphics and characters.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper x36	
74HC595 x2		8X8 LEDMatrix x1	Resistor 220Ω x8

Component knowledge

LED matrix

An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).
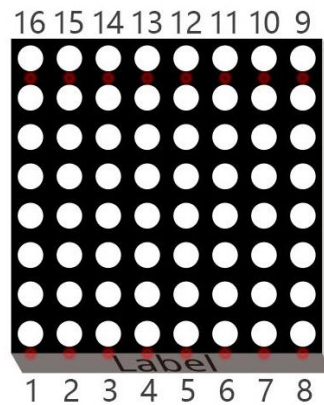
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another arrangement type. Negative Poles of the LEDs in each row and the Positive Poles of the LEDs in each column are respectively connected together, which is called a Common Cathode.

The LED Matrix that we use in this project is a Common Anode LED Matrix.

Connection mode of Common Anode   Connection mode of Common Cathode
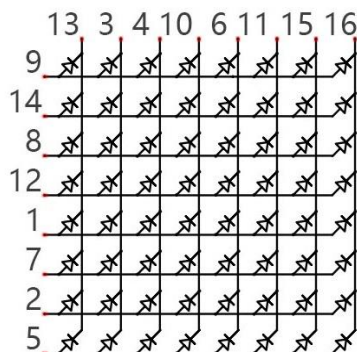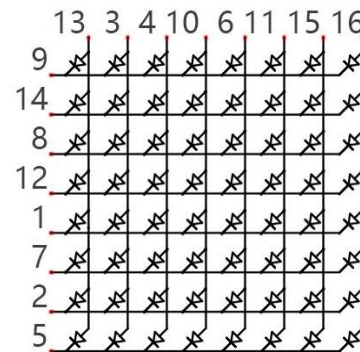
Here is how a Common Anode LED Matrix works. First, choose 16 ports on RPI board to connect to the 16 ports of LED Matrix. Configure one port in columns for low level, which makes that column the selected port. Then configure the eight port in the row to display content in the selected column. Add a delay value and then select the next column that outputs the corresponding content. This kind of operation by column is called Scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.
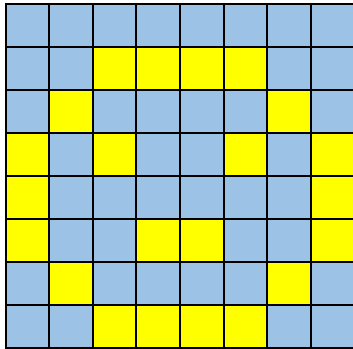
1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

To begin, display the first column, then turn off the first column and display the second column. (and so on) …. turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated rapidly in a loop. Due to the principle of optical afterglow effect and the vision persistence effect in human sight, we will see a picture of a smiling face directly rather than individual columns of LEDs turned ON one column at a time (although in fact this is the reality we cannot perceive).

Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit. Every 74HC595 IC Chip has eight parallel output ports, so two of these have a combined total of 16 ports, which is just enough for our project. The control lines and data lines of the two 74HC595 IC Chips are not all connected to the RPi, but connect to the Q7 pin of first stage 74HC595 IC Chip and to the data pin of second IC Chip. The two 74HC595 IC Chips are connected in series, which is the same as using one "74HC595 IC Chip" with 16 parallel output ports.

Circuit

In circuit of this project, the power pin of the 74HC595 IC Chip is connected to 3.3V. It can also be connected to 5V to make LED Matrix brighter.

Schematic diagram
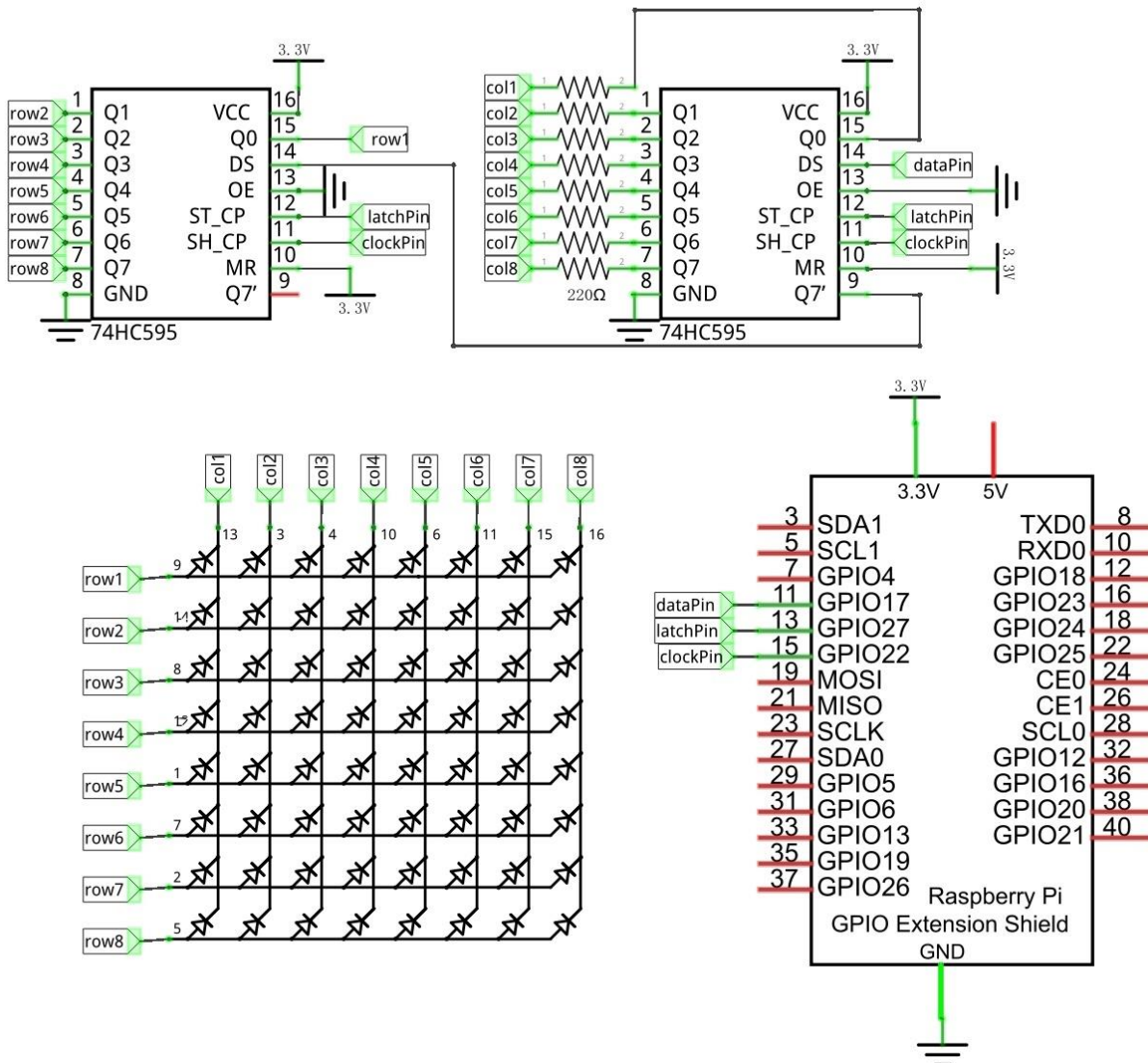
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**

Second stage
74HC595: B

First stage
74HC595: A
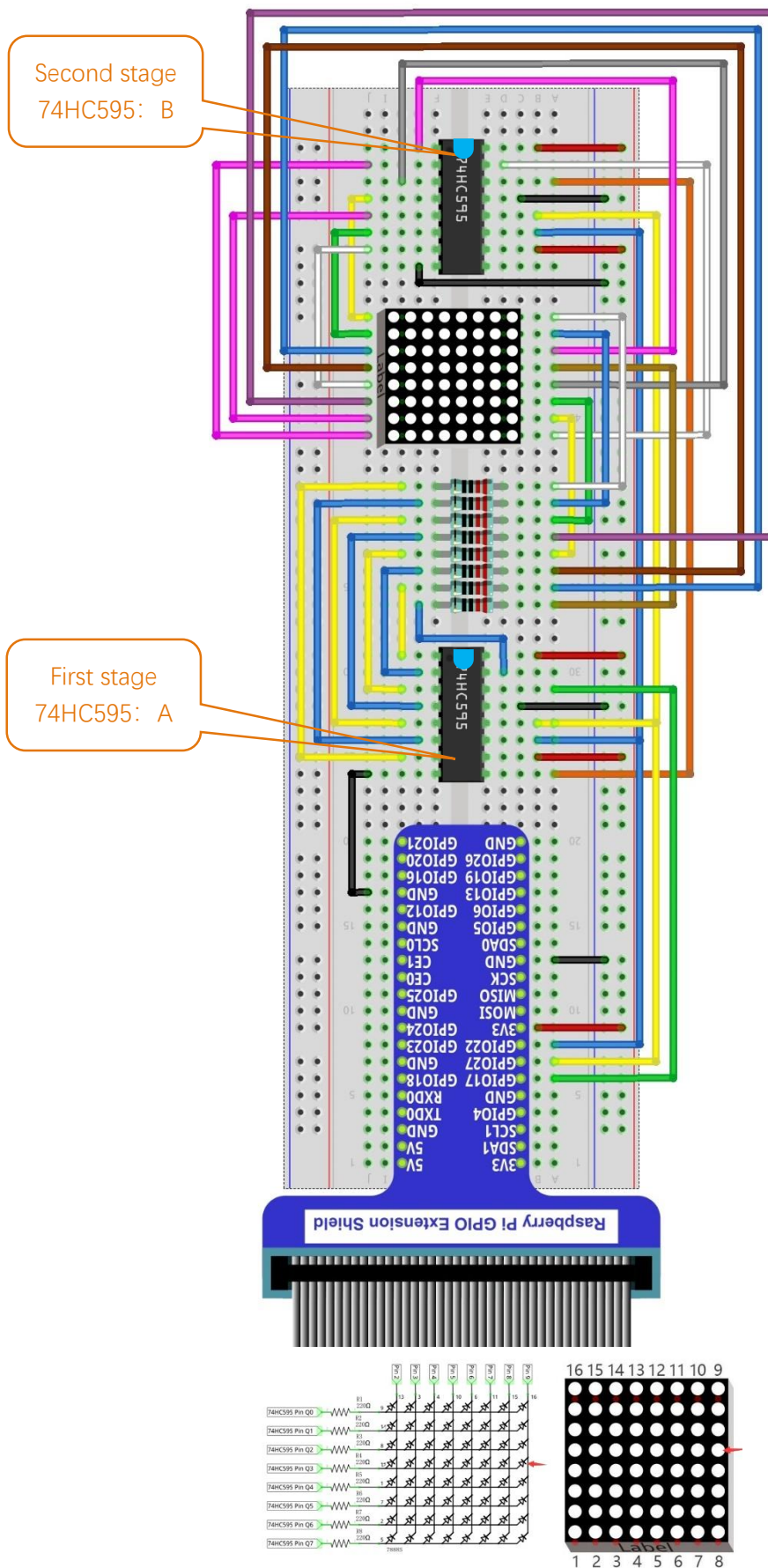
Code

Two 74HC595 IC Chips are used in this project, one for controlling the LED Matrix's columns and the other for controlling the rows. According to the circuit connection, row data should be sent first, then column data. The following code will make the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

Python Code 19.1.1 LEDMatrix

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 19.1.1_LEDMatrix directory of Python language.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/19.1.1_LEDMatrix
```

2. Use Python command to execute Python code "LEDMatrix.py".

```
python LEDMatrix.py
```

After the program is executed, the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```python
1   from gpiozero import OutputDevice
2   import time
3
4   LSBFIRST = 1
5   MSBFIRST = 2
6   dataPin   = OutputDevice(17)      # DS Pin of 74HC595(Pin14)
7   latchPin  = OutputDevice(27)      # ST_CP Pin of 74HC595(Pin12)
8   clockPin  = OutputDevice(22)      # CH_CP Pin of 74HC595(Pin11)
9   pic = [0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c]  # data of smiling face
10  data = [     # data of "0-F"
11      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
12      0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
13      0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
14      0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
15      0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
16      0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
17      0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
18      0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
19      0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
20      0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
21      0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
22      0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
23      0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
24      0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
25      0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
26      0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
27      0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
```

```
28        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
29    ]
30
31    def shiftOut(order,val):
32        for i in range(0,8):
33            clockPin.off()
34            if(order == LSBFIRST):
35                dataPin.on() if (0x01&(val>>i)==0x01) else dataPin.off()
36            elif(order == MSBFIRST):
37                dataPin.on() if (0x80&(val<<i)==0x80) else dataPin.off()
38            clockPin.on()
39
40    def loop():
41        while True:
42            for j in range(0,500): # Repeat enough times to display the smiling face a period of
    time
43                x=0x80
44                for i in range(0,8):
45                    latchPin.off()
46                    shiftOut(MSBFIRST,pic[i]) #first shift data of line information to first stage
    74HC959
47                    shiftOut(MSBFIRST,~x) #then shift data of column information to second stage
    74HC959
48                    latchPin.on()# Output data of two stage 74HC595 at the same time
                    time.sleep(0.001) # display the next column
49                    x>>=1
50            for k in range(0,len(data)-8): #len(data) total number of "0-F" columns
51                for j in range(0,20): # times of repeated displaying LEDMatrix in every frame, the
    bigger the "j", the longer the display time.
52                    x=0x80       # Set the column information to start from the first column
                    for i in range(k,k+8):
53                        latchPin.off()
                        shiftOut(MSBFIRST,data[i])
54                        shiftOut(MSBFIRST,~x)
55                        latchPin.on()
56                        time.sleep(0.001)
57                        x>>=1
58    def destroy():
59        dataPin.close()
60        latchPin.close()
61        clockPin.close()
62    if __name__ == '__main__': # Program entrance
63        print ('Program is starting...' )
64        try:
```

65	loop()
66	except KeyboardInterrupt: # Press ctrl-c to end the program.
67	destroy()
68	print("Ending program")

The first "for" loop in the "while" loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```
for j in range(0,500): # Repeat enough times to display the smiling face a period of
time
        x=0x80
        for i in range(0,8):
            latchPin.off()
            shiftOut(MSBFIRST,pic[i]) #first shift data of line information to first stage
74HC959
            shiftOut(MSBFIRST,~x) #then shift data of column information to second stage
74HC959
            latchPin.on()# Output data of two stage 74HC595 at the same time
            time.sleep(0.001) # display the next column
            x>>=1
```

The second "for" loop is used to display scrolling characters "0 to F", for a total of 18 X 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column…… and so on…138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

```
for k in range(0,len(data)-8): #len(data) total number of "0-F" columns
        for j in range(0,20): # times of repeated displaying LEDMatrix in every frame, the
bigger the "j", the longer the display time.
            x=0x80      # Set the column information to start from the first column
            for i in range(k,k+8):
                latchPin.off()
                shiftOut(MSBFIRST,data[i])
                shiftOut(MSBFIRST,~x)
                latchPin.on()
                time.sleep(0.001)
                x>>=1
```

Chapter 20 LCD1602

In this chapter, we will learn about the LCD1602 Display Screen,

Project 20.1 I2C LCD1602

There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram
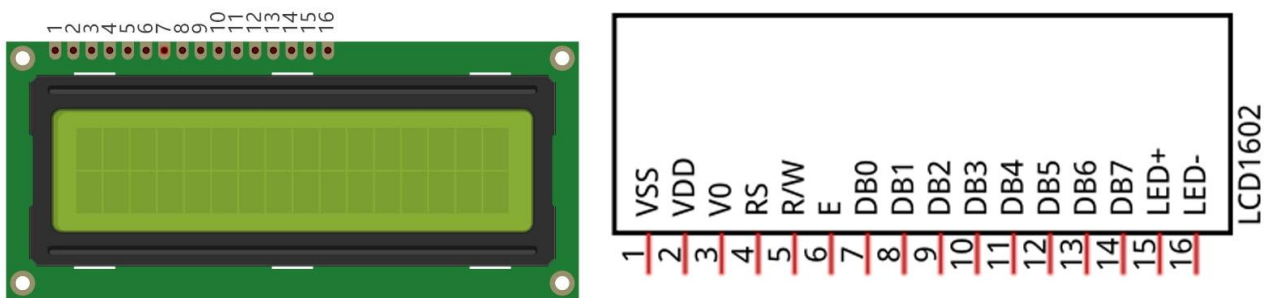
I2C LCD1602 Display Screen integrates a I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to only use 4 lines to operate the LCD1602.
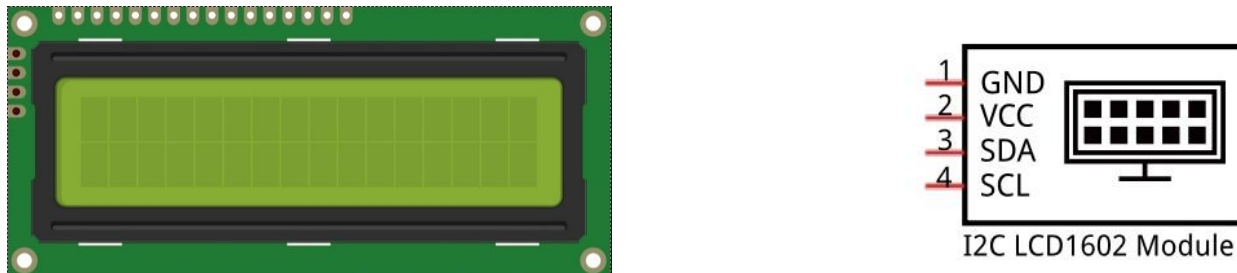
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the RPI bus on your I2C device address through command "i2cdetect -y 1" (refer to the "configuration I2C" section below).

Below is the PCF8574 chip pin diagram and its module pin diagram:

PCF8574 chip pin diagram:	PCF8574 module pin diagram
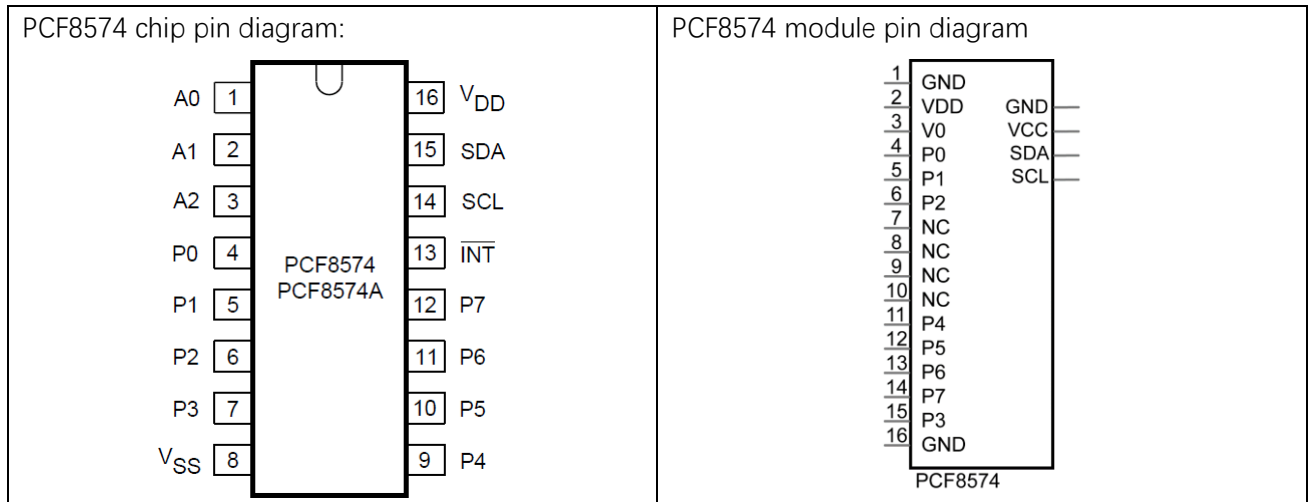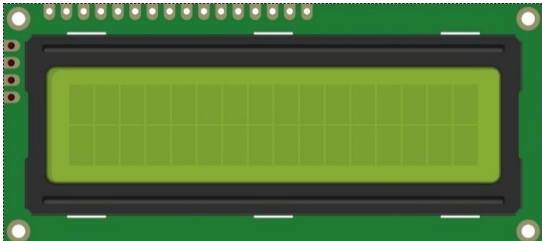
PCF8574 module pins and LCD1602 pins correspond to each other and connected to each other:
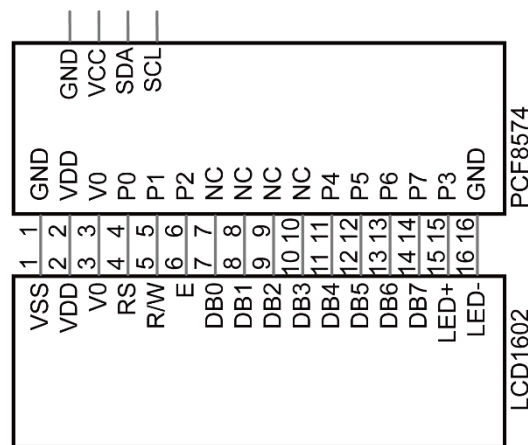
Because of this, as stated earlier, we only need 4 pins to control the16 pins of the LCD1602 Display Screen through the I2C interface.

In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x4
I2C LCD1602 Module x1	

Circuit

Note that the power supply for I2C LCD1602 in this circuit is 5V.

Schematic diagram
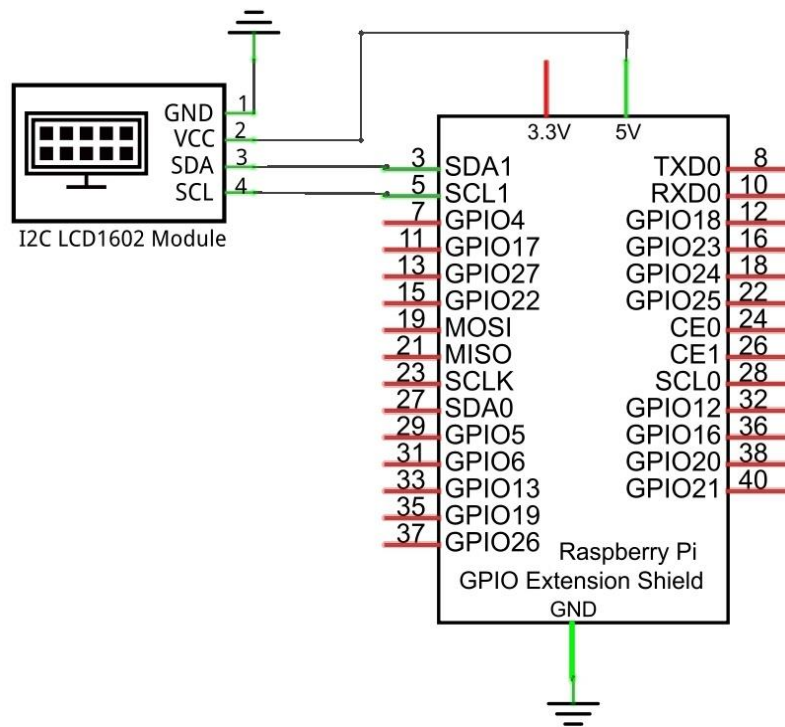
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**

NOTE: It is necessary to configure 12C and install Smbus first (see chapter 7 for details)
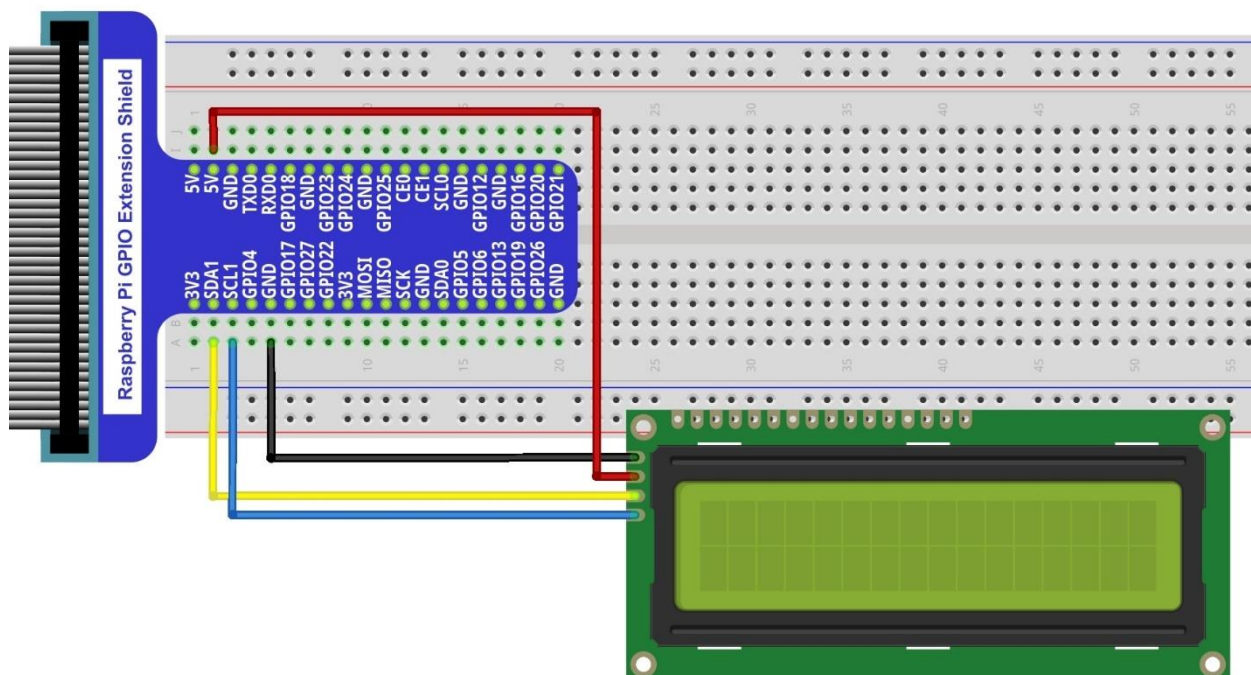
Code

This code will have your RPi's CPU temperature and System Time Displayed on the LCD1602.

Python Code 20.1.1 I2CLCD1602

If you did not configure I2C and install Smbus, please refer to Chapter 7. If you did, continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 20.1.1_ I2CLCD1602 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/20.1.1_I2CLCD1602
```

2. Use Python command to execute Python code "I2CLCD1602.py".

```
python I2CLCD1602.py
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time. So far, at this writing, we have two types of LCD1602 on sale. One needs to adjust the backlight, and the other does not.

The LCD1602 that does not need to adjust the backlight is shown in the figure below.

If the LCD1602 you received is the following one, and you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display clearly.

The following is the program code:

1	`import smbus`
2	`from time import sleep, strftime`
3	`from datetime import datetime`
4	`from LCD1602 import CharLCD1602`
5	
6	`lcd1602 = CharLCD1602()`

```
7    def get_cpu_temp():      # get CPU temperature from file
8    "/sys/class/thermal/thermal_zone0/temp"
9        tmp = open('/sys/class/thermal/thermal_zone0/temp')
10       cpu = tmp.read()
11       tmp.close()
12       return '{:.2f}'.format( float(cpu)/1000 ) + ' C '
13
14   def get_time_now():      # get system time
15       return datetime.now().strftime('    %H:%M:%S')
16
17   def loop():
18       lcd1602.init_lcd()
19       count = 0
20       while(True):
21           lcd1602.clear()
22           lcd1602.write(0, 0, 'CPU: ' + get_cpu_temp() )# display CPU temperature
23           lcd1602.write(0, 1, get_time_now() )   # display the time
24           sleep(1)
25   def destroy():
26       lcd1602.clear()
27   if __name__ == '__main__':
28       print ('Program is starting ... ')
29       try:
30           loop()
31       except KeyboardInterrupt:
32           destroy()
```

In a while loop, set the cursor position, and display the CPU temperature and time.

```
        while(True):
            lcd1602.clear()
            lcd1602.write(0, 0, 'CPU: ' + get_cpu_temp() )# display CPU temperature
            lcd1602.write(0, 1, get_time_now() )   # display the time
            sleep(1)
```

CPU temperature is stored in file "/sys/class/thermal/thermal_zone0/temp". Open the file and read content of the file, and then convert it to Celsius degrees and return. Subfunction used to get CPU temperature is shown below:

```
def get_cpu_temp():      # get CPU temperature and store it into file
  "/sys/class/thermal/thermal_zone0/temp"
    tmp = open('/sys/class/thermal/thermal_zone0/temp')
    cpu = tmp.read()
    tmp.close()
    return '{:.2f}'.format( float(cpu)/1000 ) + ' C'
```

Subfunction used to get time:

```
def get_time_now():      # get the time
```

```
    return datetime.now().strftime('    %H:%M:%S')
```

Details about LCD1602.py:

Module LCD1602
This module provides the basic operation method of LCD1602, including class CharLCD1602. Some member functions are described as follows: **def init_lcd(self,addr=None, bl=1)** : LDC1602 initializes the setting.   When the addr is None, the I2C address of the device will be automatically scanned.   You can also specify the I2C address, bl=1 to enable the backlight setting. **def clear(self)**: clear the screen **def send_command(self,comm)**: set the cursor position **def i2c_scan(self):** scan the device I2C address **def write(self,x, y, str)**: display contents More information can be viewed through opening LCD1602.py.
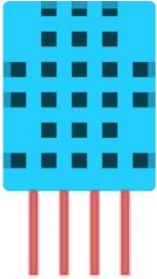
Chapter 21 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

Project 21.1 Hygrothermograph

Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the RPi to read Temperature and Humidity data of the DHT11 Module.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	DHT11 x1	Resistor 10kΩ x1
Jumper Wire x4		

Component knowledge

The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.
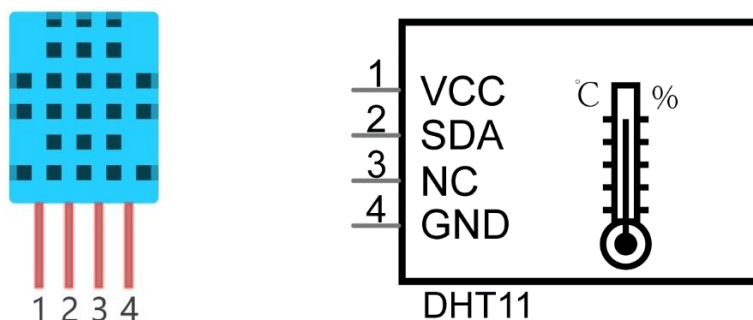
After being powered up, it will initialize in 1 second. Its operating voltage is within the range of 3.3V-5.5V. The SDA pin is a data pin, which is used to communicate with other devices.

The NC pin (Not Connected Pin) are a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins **should not be connected** to any of the circuit connections.

Circuit

Schematic diagram
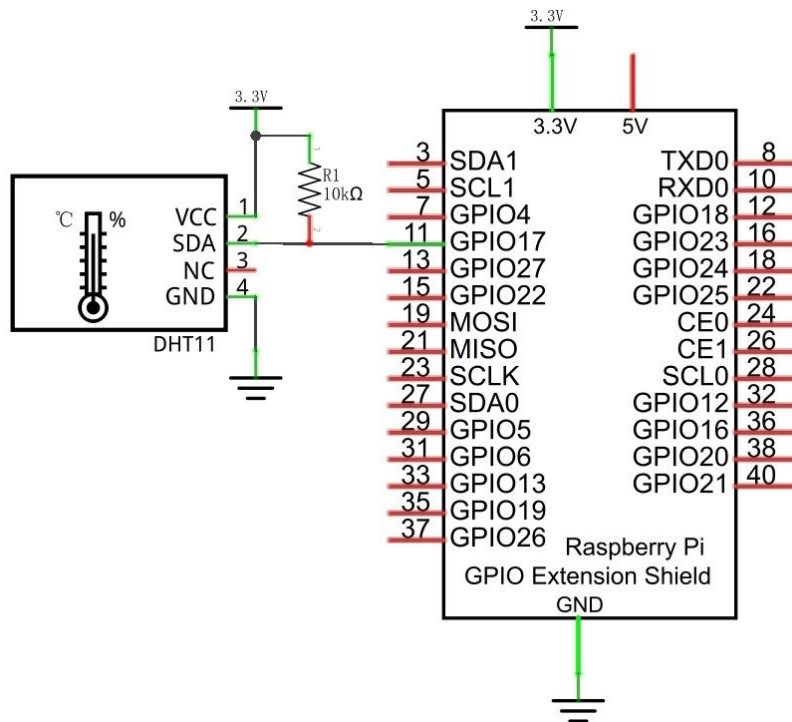
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
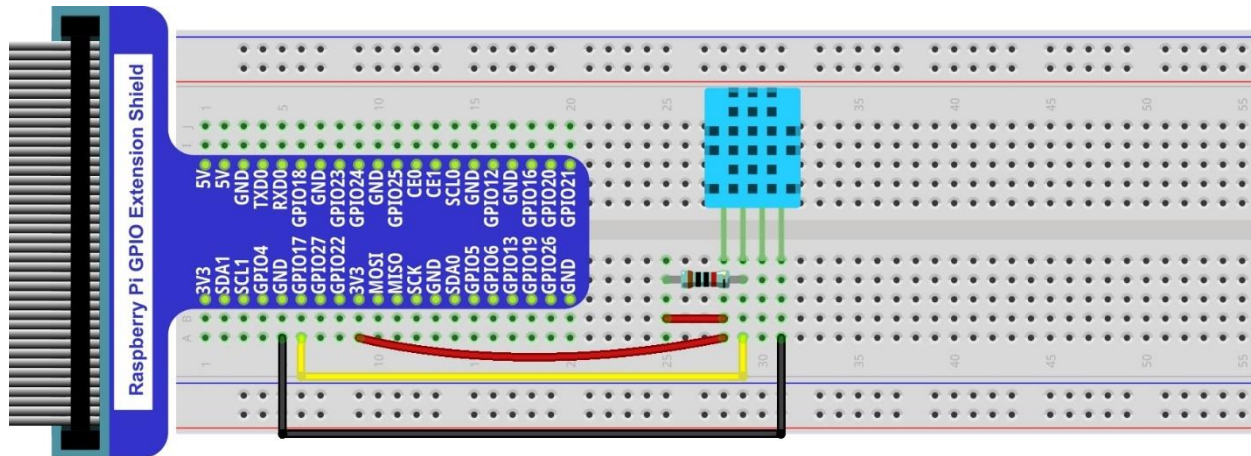
Code

The code is used to read the temperature and humidity data of DHT11, and display them.

Python Code 21.1.1 DHT11

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1.   Use cd command to enter 21.1.1_DHT11 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/21.1.1_DHT11
```

2.   Use Python command to execute code "DHT11.py".

```
python DHT11.py
```

After the program is executed, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts:  2
DHT11,OK!
Humidity : 53.00,        Temperature : 27.60

Measurement counts:  3
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  4
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  5
DHT11,OK!
Humidity : 52.00,        Temperature : 27.50
```

Since gpiozero does not support DHT11 sensors, RPi.GPIO is used here for control.

The following is the program code:

```
1   import time
2   import Freenove_DHT as DHT
3   DHTPin = 11      #define the pin of DHT11
4
5   def loop():
6       dht = DHT.DHT(DHTPin)   #create a DHT class object
7       counts = 0 # Measurement counts
8       while(True):
9           counts += 1
10          print("Measurement counts: ", counts)
11          for i in range(0,15):
12              chk = dht.readDHT11()    #read DHT11 and get a return value. Then determine
    whether data read is normal according to the return value.
13              if (chk is dht.DHTLIB_OK):     #read DHT11 and get a return value. Then determine
    whether data read is normal according to the return value.
14                  print("DHT11,OK!")
15                  break
```

```
16              time.sleep(0.1)
17         print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
18         time.sleep(2)
19
20  if __name__ == '__main__':
21     print ('Program is starting ... ')
22     try:
23         loop()
24     except KeyboardInterrupt:
25         exit()
```

In this project code, we use a module "**Freenove_DHT**.py", which provides the method of reading the DHT Sensor. It is located in the same directory with program files "**DHT11**.py". By using this library, we can easily read the DHT Sensor. First, we create a DHT class object in the code.

```
dht = DHT.DHT(DHTPin)    #create a DHT class object
```

Then in the "while" loop, use chk = dht.**readDHT11** (DHT11Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk". Then use variable sumCnt to record the number of times read.

```
while(True):
    counts += 1
    print("Measurement counts: ", counts)
    for i in range(0,15):
        chk = dht.readDHT11()     #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
        if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
            print("DHT11,OK!")
            break
        time.sleep(0.1)
    print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
    time.sleep(2)
```

Finally display the results:

```
print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
```

Module "**Freenove_DHT.py**" contains a DHT class. The class function of the def **readDHT11** (pin) is used to read the DHT11 Sensor and store the temperature and humidity data read to member variables humidity and temperature.

Freenove_DHT Module
This is a Python module for reading the temperature and humidity data of the DHT Sensor. Partial functions and variables are described as follows: Variable **humidity**: store humidity data read from sensor Variable **temperature**: store temperature data read from sensor **def** readDHT11 **(pin)**: read the temperature and humidity of sensor DHT11, and return values used to determine whether the data is normal.
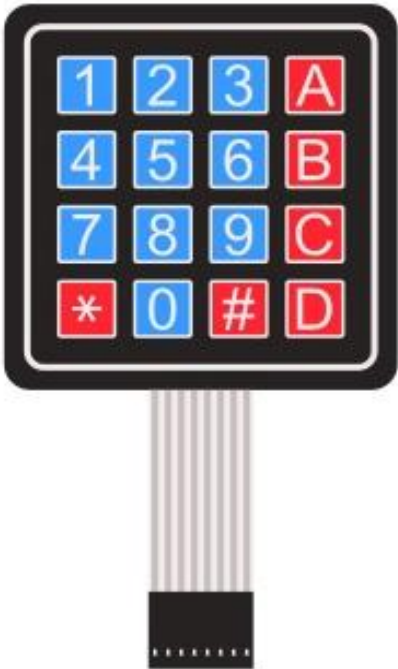
Chapter 22 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 22.1 Matrix Keypad

In this project, we will attempt to get every key code on the Matrix Keypad to work.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Wire x1 Breadboard x1	4x4 Matrix Keypad x1
Jumper wire	
Resistor 10kΩ x4	

Component knowledge

4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad

Matrix integrates 16 keys (think of this as 16 Push Button Switches in one module):
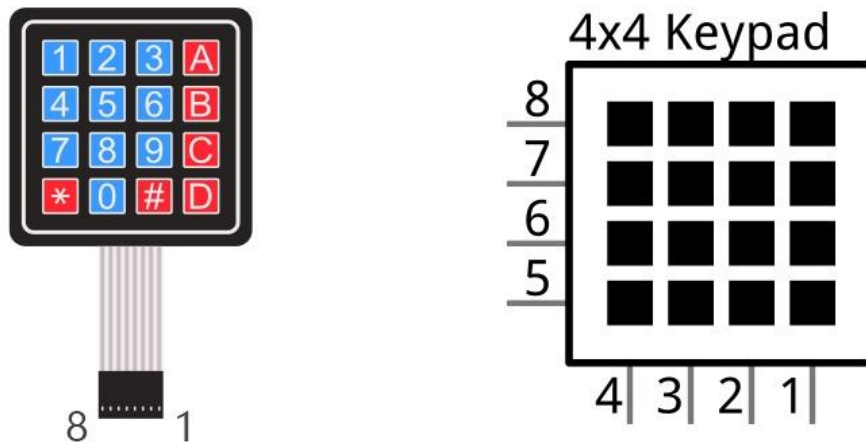
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.
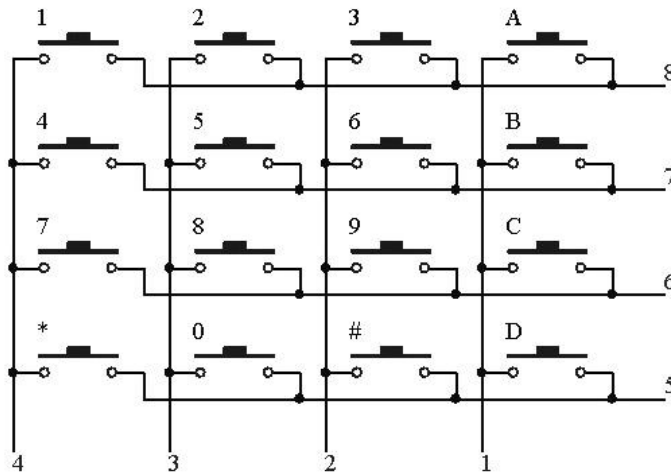
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Therefore, you can get the state of all of the keys.

Circuit

Schematic diagram
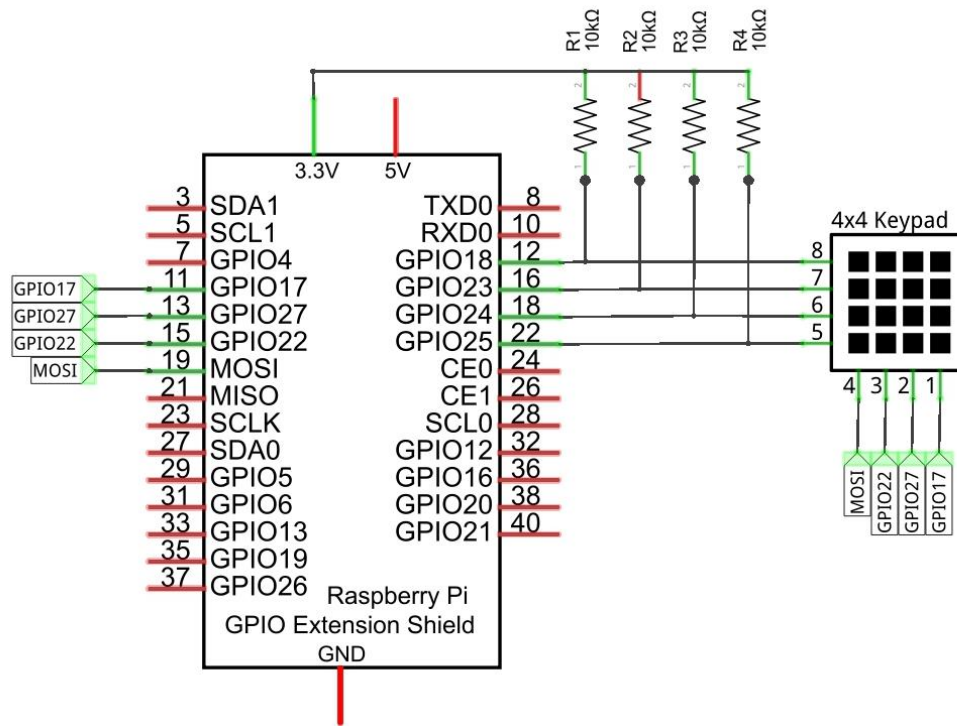
Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
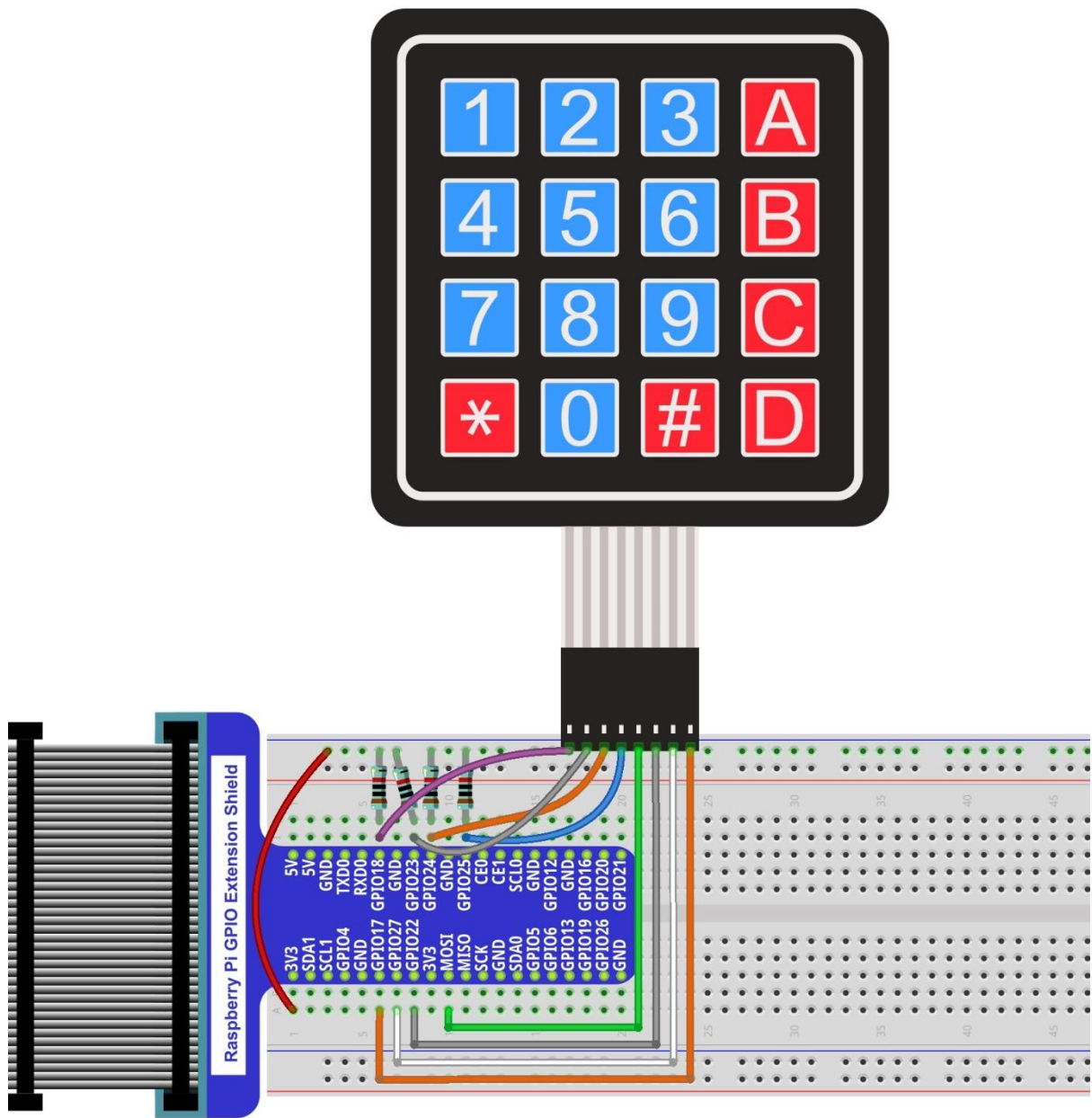
Code

This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be displayed in the terminal window.

Python Code 22.1.1 MatrixKeypad

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com
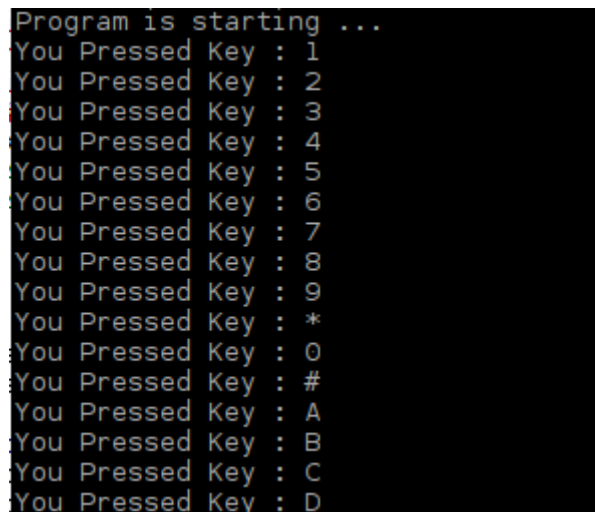
1. Use cd command to enter 22.1.1_MatrixKeypad directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/22.1.1_MatrixKeypad
```

2. Use Python command to execute code "MatrixKeypad.py".

```
python MatrixKeypad.py
```

After the program is executed, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:

The following is the program code:

```python
1    import Keypad    #import module Keypad
2    ROWS = 4         # number of rows of the Keypad
3    COLS = 4         #number of columns of the Keypad
4    keys =  [   '1','2','3','A',     #key code
5                '4','5','6','B',
6                '7','8','9','C',
7                '*','0','#','D'     ]
8    rowsPins = [18, 23, 24, 25]      #connect to the row pinouts of the keypad
9    colsPins = [10, 22, 27, 17]      #connect to the column pinouts of the keypad
10   def loop():
11       keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)    #creat Keypad object
12       keypad.setDebounceTime(50)        #set the debounce time
13       while(True):
14           key = keypad.getKey()         #obtain the state of keys
15           if(key != keypad.NULL):       #if there is key pressed, print its key code.
16               print ("You Pressed Key : %c "%(key))
```

```
17
18   if __name__ == '__main__':       #Program start from here
19       print ("Program is starting ... ")
20       try:
21           loop()
22       except KeyboardInterrupt:  #When 'Ctrl+C' is pressed, exit the program.
23           print("Ending program")
```

In this project code, we use a custom module "**Keypad**.py", which is located in the same directory with program file "**MatrixKeypad**.py". And this library file, which is transplanted from Arduino function library Keypad, provides a method to read the keyboard. By using this library, we can easily read the matrix keyboard. First, import module Keypad. Then define the information of the matrix keyboard used in this project: the number of rows and columns, code of each key and GPIO pin connected to each column and each row.

```
import Keypad    #import module Keypad
ROWS = 4         # number of rows of the Keypad
COLS = 4         #number of columns of the Keypad
keys = [   '1','2','3','A',    #key code
           '4','5','6','B',
           '7','8','9','C',
           '*','0','#','D'     ]
rowsPins = [18, 23, 24, 25]      #connect to the row pinouts of the keypad
colsPins = [26, 22, 27, 17]      #connect to the column pinouts of the keypad
```

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

```
keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)
```

Set the debounce time to 50ms, and this value can be set based on the actual characteristics of the keyboard's flexibly, with a default time of 10ms.

```
keypad.setDebounceTime(50)
```

In the "while" loop, use the function key= keypad.**getKey** () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", and then be displayed.

```
while(True):
    key = keypad.getKey()        #get the state of keys
    if(key != keypad.NULL):      # if a key is pressed, print out its key code
        print ("You Pressed Key : %c "%(key))
```

The Keypad Library used for the RPi is "transplanted" from the Arduino Keypad Library. The source files is written by language C++ and translated into Python can be obtained by visiting http://playground.arduino.cc/Code/Keypad. As for the "transplanted" function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad Library are described below:

```
class Keypad
def __init__(self, usrKeyMap, row_Pins, col_Pins, num_Rows, num_Cols):
```
Constructed function, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.
```
def getKey(self):
```
Get a pressed key. If no key is pressed, the return value is keypad NULL.
```
def setDebounceTime(self, ms):
```
Set the debounce time. And the default time is 10ms.
```
def setHoldTime(self, ms):
```
Set the time when the key holds stable state after pressed.
```
def isPressed(keyChar):
```
Judge whether the key with code "keyChar" is pressed.
```
def waitForKey():
```
Wait for a key to be pressed, and return key code of the pressed key.
```
def getState():
```
Get state of the keys.
```
def keyStateChanged():
```
Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad or through the opening file "Keypad.py".

For more information about the methods used by the InputDevice class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_input.html#inputdevice

For more information about the methods used by the OutputDevice class in the GPIO Zero library,please refer to: https://gpiozero.readthedocs.io/en/stable/api_output.html#outputdevice

Chapter 23 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance.

Project 23.1 Ultrasonic Ranging

In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Ultrasonic Module x1
Jumper Wire x4	Resistor 1kΩ x3

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will be reflected when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.
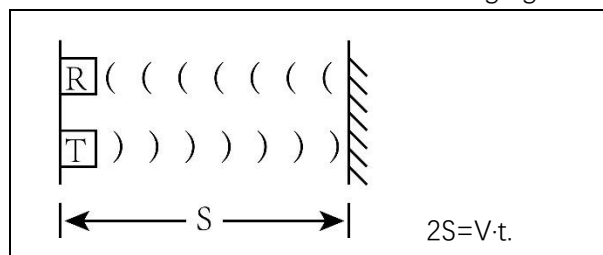
The Ultrasonic Ranging Module integrates a both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the Ultrasonic Ranging Module are shown below:
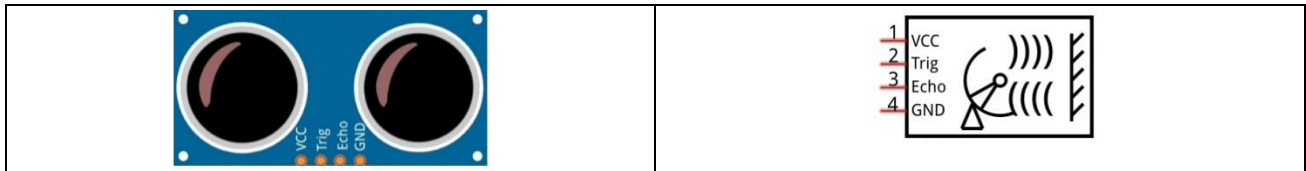
Pin description:

VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V   Working current: 12mA

Minimum measured distance: 2cm   Maximum measured distance: 200cm

Instructions for Use: output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2. This is done constantly.
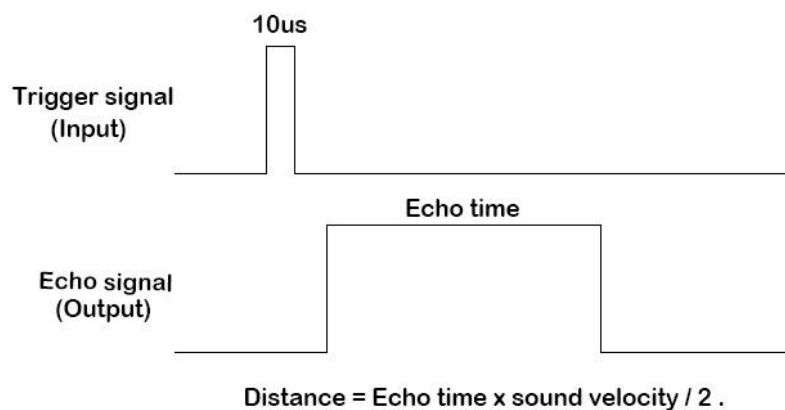
Circuit

Note that the voltage of ultrasonic module is 5V in this circuit.

Schematic diagram
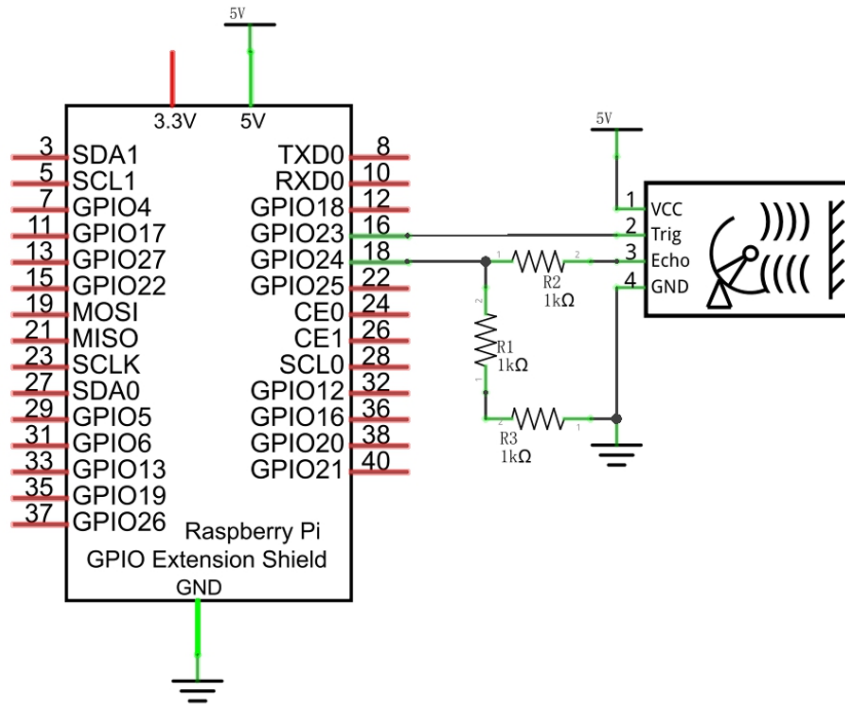
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
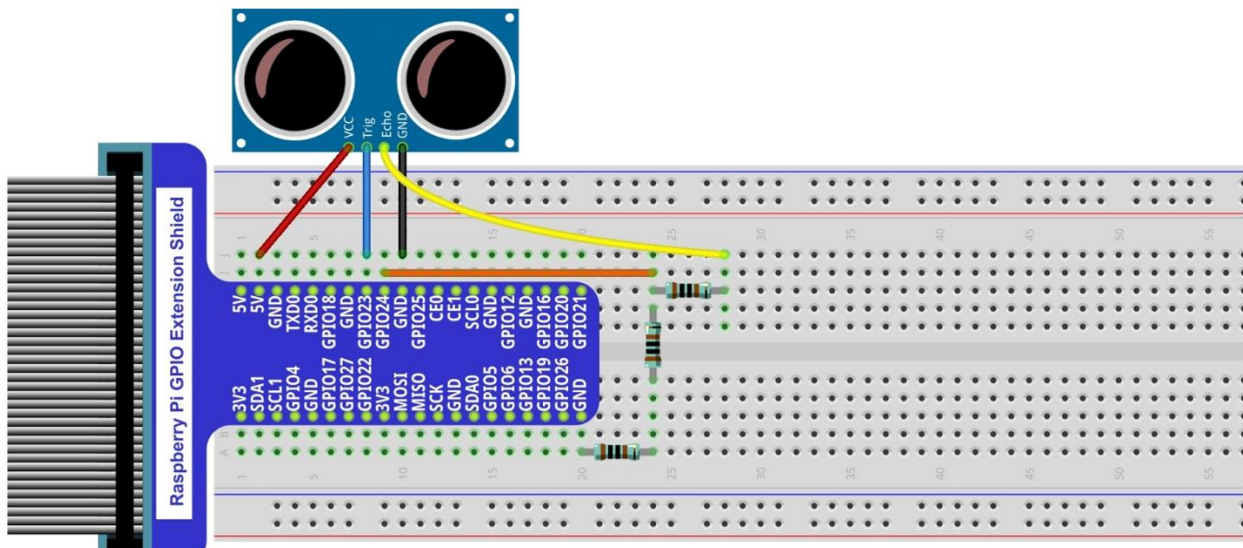
Code

Python Code 23.1.1 UltrasonicRanging

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 23.1.1_UltrasonicRanging directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/23.1.1_UltrasonicRanging
```

2. Use Python command to execute code "UltrasonicRanging.py".

```
python UltrasonicRanging.py
```

After the program is executed, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.75 cm
The distance is : 199.22 cm
The distance is : 198.42 cm
The distance is : 198.74 cm
The distance is : 198.37 cm
The distance is : 198.47 cm
The distance is : 198.41 cm
```

The following is the program code:

```python
1   from gpiozero import DistanceSensor
2   from time import sleep
3
4   trigPin = 23
5   echoPin = 24
6   sensor = DistanceSensor(echo=echoPin, trigger=trigPin ,max_distance=3)
7
8   def loop():
9       while True:
10          print('Distance: ', sensor.distance * 100,'cm')
11          sleep(1)
12
13  if __name__ == '__main__':      # Program entrance
14      print ('Program is starting...')
15      try:
16          loop()
17      except KeyboardInterrupt:  # Press ctrl-c to end the program.
18          sensor.close()
19          print("Ending program")
```

First, define the pins and the maximum measurement distance.

```python
trigPin = 23
echoPin = 24
```

```python
sensor = DistanceSensor(echo=echoPin, trigger=trigPin ,max_distance=3) # define the maximum
measured distance 300cm
```

Finally, in the while loop of main function, get the measurement distance and display it continually.

```python
def loop():
    while True:
        print('Distance: ', sensor.distance * 100,'cm')
        sleep(1)
```

For more information about the methods used by the DistanceSensor class in the GPIO Zero library,please
refer to: https://gpiozero.readthedocs.io/en/stable/api_input.html#distancesensor-hc-sr04

In the above experiments, you can see that the measurement data is unstable.
Note: For improved accuracy, use the pigpio pin driver rather than the default RPi.GPIO driver (pigpio uses
DMA sampling for much more precise edge timing). This is particularly relevant if you're using Pi 1 or Pi
Zero.
You can refer to UltrasonicRanging2.py for detailed code.
1.   Use cd command to enter 15.1.1_Sweep directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/23.1.1_UltrasonicRanging
```

2.   Use python command to execute code "Sweep.py".

```
python UltrasonicRanging2.py
```

After the program is executed, the distance between the ultrasonic module and the measured object will
be displayed on the terminal. At this point, the data is more accurate and stable.
This code is based on pigpio libarary. In the latest Raspberry Pi OS, "pigpio" library has been installed. You
only need to run the command to enable it.

```
sudo pigpiod
```

```
pi@raspberrypi:~ $ sudo pigpiod
pi@raspberrypi:~ $ ▮
```

If the "pigpio" library has not yet been installed, please follow the steps to install it.
Run the command to install "pigpio" library.

```
sudo apt-get update
sudo apt-get install pigpio python-pigpio python3-pigpio
```

The following is the program code:

```python
1   import os
2   os.system("sudo pigpiod")
3   from gpiozero import DistanceSensor
4   from gpiozero.pins.pigpio import PiGPIOFactory
5   from time import sleep
6
7   trigPin = 23
8   echoPin = 24
9   my_factory = PiGPIOFactory()
```

```
       sensor = DistanceSensor(echo=echoPin, trigger=trigPin ,max_distance=3,pin_factory=my_factory)
10
11     def loop():
12         while True:
13             print('Distance: ', sensor.distance * 100,'cm')
14             sleep(1)
15
16     if __name__ == '__main__':      # Program entrance
17         print ('Program is starting...')
18         try:
19             loop()
20         except KeyboardInterrupt:   # Press ctrl-c to end the program.
21             sensor.close()
22             os.system("sudo killall pigpiod")
23             print("Ending program")
```

See Changing the pin factory for further information:

https://gpiozero.readthedocs.io/en/stable/api_pins.html#changing-pin-factory
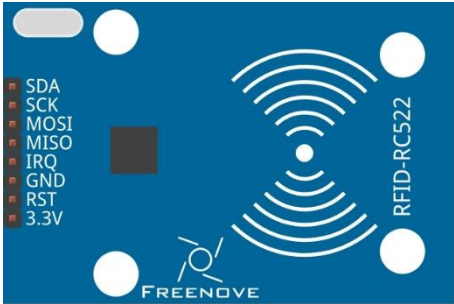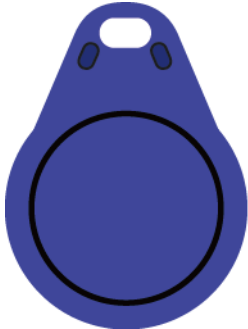
Chapter 24 RFID

In this chapter, we will learn how to use RFID.

Project 24.1 RFID

In this project, we will use RC522 RFID card reader to read and write the M1-S50 card.

Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 Breadboard x1	RC522 module x1
Jumper M/F x7 	
Mifare1 S50 Standard card x1 	Mifare1 S50 Non-standard card x1

Component Knowledge

RFID

RFID（Radio Frequency Identification）is a form of wireless communication technology. A complete RFID system is generally composed of a transponder and a reader. Generally, the transponder may be known as a tag, and each tag has a unique code, which is attached to an object to identify the target object. The reader is a device that reads (or writes) information in the tag.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products, among which, Passive RFID products are the earliest, the most mature and most widely used products in the market. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of them are classified as close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13-26mA(DC)\3.3V
Idle current	10-13mA(DC)\3.3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13.56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm
Operation temperature	20-80 degrees(Celsius)
Storage temperature	40-85 degrees (Celsius)
Operation humidity	5%-95%(Relative humidity)

Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years.

The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63).

And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control put in its last block, that is, Block 3, which is also known as sector trailer. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the card serial number and vendor code, which has been solidified and can't be changed. Except the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

(1) used as general data storage and can be operated for reading and writing data.

(2) used as data value, and can be operated for initializing, adding, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, a 4-byte access control and a 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally 0A1A2A3A4A5 for password A and B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read, so if you choose to verify password A but forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

For Mifare1 S50 card equipped in Freenove RFID Kit, the default password A and B are both FFFFFFFFFFFF.

Circuit

Schematic diagram:
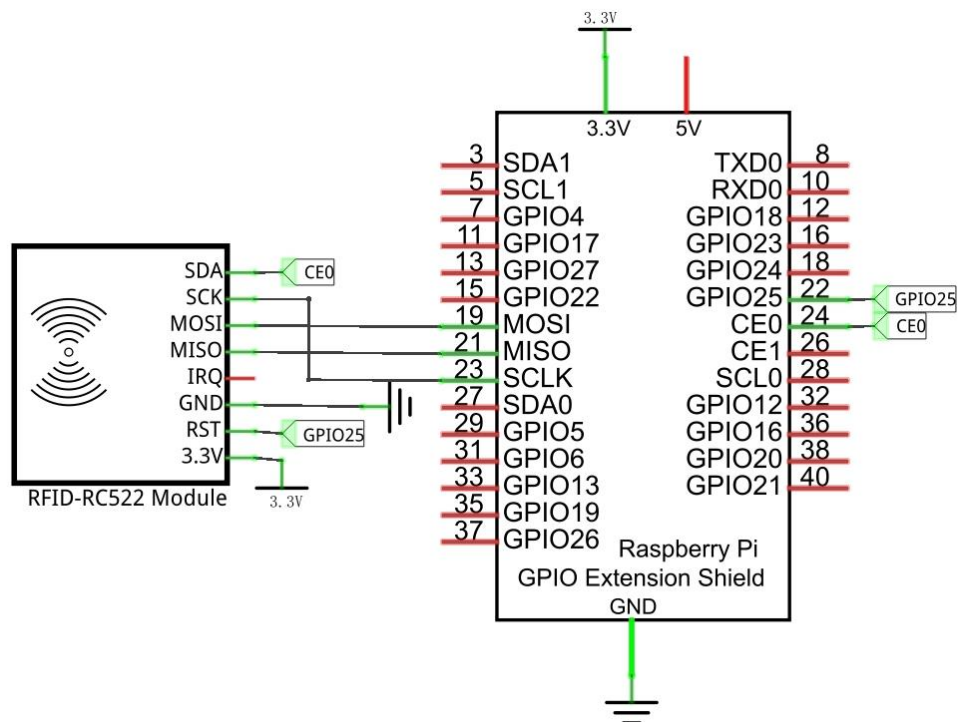
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
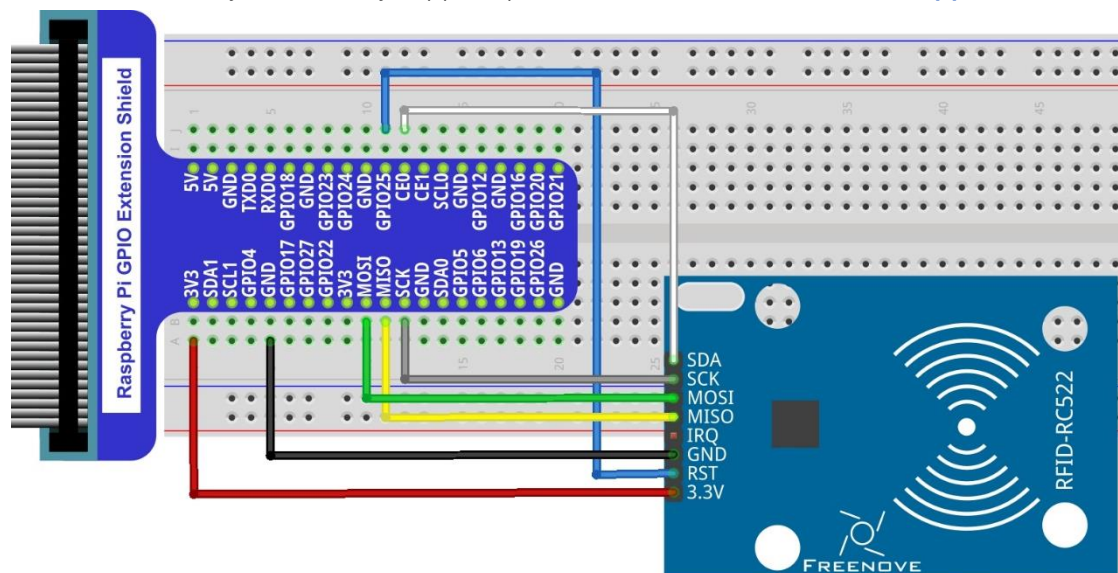
Configure SPI

Enable SPI

The SPI interface of raspberry pi is closed by default. You need to open it manually. You can enable the SPI interface in the following way.

Type the following command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:

```
┌─────┤ Raspberry Pi Software Configuration Tool (raspi-config) ├─────┐

   1 Change User Password        Change password for the current u
   2 Network Options             Configure network settings
   3 Boot Options                Configure options for start-up
   4 Localisation Options        Set up language and regional sett
   5 Interfacing Options         Configure connections to peripher
   6 Overclock                   Configure overclocking for your P
   7 Advanced Options            Configure advanced settings
   8 Update                      Update this tool to the latest ve
   9 About raspi-config          Information about this configurat


              <Select>                        <Finish>
```

Choose "5 Interfacing Options"➔"P4 SPI"➔"Yes"➔"Finish" in order and then restart your RPi. Then the SPI module is started.

Type the following command to check whether the module SPI is loaded successfully:

```
ls /dev/sp*
```

The following result indicates that the module SPI has been loaded successfully:

```
pi@raspberrypi:~ $ ls /dev/sp*
/dev/spidev0.0   /dev/spidev0.1
```

Install Python module SPI-Py

If you use Python language to write the code, please follow the steps below to install the module SPI-Py. If you use C/C++ language, you can skip this step.

Open the terminal and type the following command to install:

```
git clone https://github.com/Freenove/SPI-Py
cd SPI-Py
sudo python3 setup.py install
```

Code

The project code uses human-computer interaction command line mode to read and write the M1-S50 card.

Python Code 24.1.1 RFID

There are two code files for this project. They are respectively under Python2 folder and Python3 folder. **Their functions are the same, but they are not compatible.** Code under Python2 folder can only run on Python2. And code under Python3 folder can only run on Python3.

First observe the project result, and then learn about the code in detail.

If you need any support, please contact us via: support@freenove.com

1. Use cd command to enter RFID directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/24.1.1_RFID
```

2. Use python command to execute code "RFID.py".

```
python RFID.py
```

After the program is executed, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3 $ python RFID.py
Program is starting ...
Press Ctrl-C to exit.
RC522>
```

Here, type the command "quit" to exit the program.

Type command "scan", then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522> scan
scan
Scanning ...
Card detected
Card UID: ['0xe6', '0xcf', '0x5c', '0x8e', '0xfb']
Size: 8
RC522> E6CF5C8EFB>
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
        read <blockstart>
        dump
        halt
        clean <blockaddr>
        write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. As is shown below:

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to read the data of data block with address "blockstart". For example, using command "read 0" can display the content of data block 0. Using the command "read 1" can display the content of data block 1. As is shown below:

```
RC522> E6CF5C8EFB> read 0
['read', '0']
Sector 0 :  e6 cf 5c 8e fb 8 4 0 62 63 64 65 66 67 68 69  | ██▒▒cdefghi
RC522> E6CF5C8EFB> read 1
['read', '1']
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
RC522> E6CF5C8EFB> █
```

Command "dump" is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write "data" to data block with address "address", where the address range is 0-63 and the data length is 0-16. In the process of writing data to the data block, both the contents of data block before written and after written will be displayed. For example, if you want to write the string "Freenove" to the data block with address "1", you can type the following command.

write 1 Freenove

```
RC522> E6CF5C8EFB> write 1 Freenove
['write', '1', 'Freenove']
Before writing , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
4 backdata &0x0F == 0x0A 10
Data written
After written , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
```

Command "clean <address>" is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

clean 1

```
RC522> E6CF5C8EFB> clean 1
['clean', '1']
Before cleaning , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
```

Command "halt" is used to quit the selection state of the card.

```
RC522> E6CF5C8EFB> halt
['halt']
RC522> █
```

The following is the program code :

```python
1    from gpiozero import OutputDevice
2    import MFRC522
3    import sys
4    import os
5
6    # Create an object of the class MFRC522
7    mfrc = MFRC522.MFRC522()
8
9
10   def dis_ConmandLine():
11       print ("RC522>",end="")
```

```python
12    def dis_CardID(cardID):
13        print ("%2X%2X%2X%2X%2X>"%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]),end="")
14    def setup():
15        print ("Program is starting ... "  )
16        print ("Press Ctrl-C to exit.")
17        pass
18
19    def loop():
20        global mfrc3s
21        while(True):
22            dis_ConmandLine()
23            inCmd = input()
24            print (inCmd)
25            if (inCmd == "scan"):
26                print ("Scanning ... ")
27                mfrc = MFRC522.MFRC522()
28                isScan = True
29                while isScan:
30                    # Scan for cards
31                    (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
32                    # If a card is found
33                    if status == mfrc.MI_OK:
34                        print ("Card detected")
35                    # Get the UID of the card
36                    (status,uid) = mfrc.MFRC522_Anticoll()
37                    # If we have the UID, continue
38                    if status == mfrc.MI_OK:
39                        print ("Card UID: "+ str(map(hex,uid)))
40                        # Select the scanned tag
41                        if mfrc.MFRC522_SelectTag(uid) == 0:
42                            print ("MFRC522_SelectTag Failed!")
43                        if cmdloop(uid) < 1 :
44                            isScan = False
45
46            elif inCmd == "quit":
47                destroy()
48                exit(0)
49            else :
50                print ("\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
      program\n")
51
52    def cmdloop(cardID):
53        pass
54        while(True):
```

```
55          dis_ConmandLine()
56          dis_CardID(cardID)
57          inCmd = input()
58          cmd = inCmd.split(" ")
59          print (cmd)
60          if(cmd[0] == "read"):
61              blockAddr = int(cmd[1])
62              if((blockAddr<0) or (blockAddr>63)):
63                  print ("Invalid Address!")
64              # This is the default key for authentication
65              key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
66              # Authenticate
67              status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
68              # Check if authenticated
69              if status == mfrc.MI_OK:
70                  mfrc.MFRC522_Readstr(blockAddr)
71              else:
72                  print ("Authentication error")
73                  return 0
74
75          elif cmd[0] == "dump":
76              # This is the default key for authentication
77              key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
78              mfrc.MFRC522_Dump_Str(key,cardID)
79
80          elif cmd[0] == "write":
81              blockAddr = int(cmd[1])
82              if((blockAddr<0) or (blockAddr>63)):
83                  print ("Invalid Address!")
84              data = [0]*16
85              if(len(cmd)<2):
86                  data = [0]*16
87              else:
88                  data = cmd[2][0:17]
89                  data = map(ord,data)
90                  data = list(data)
91                  lenData = len(list(data))
92              if lenData<16:
93                      data+=[0]*(16-lenData)
94              # This is the default key for authentication
95              key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
96              # Authenticate
97              status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
98              # Check if authenticated
```

```python
 99                 if status == mfrc.MI_OK:
100                     print ("Before writing , The data in block %d  is: "%(blockAddr))
101                     mfrc.MFRC522_Readstr(blockAddr)
102                     mfrc.MFRC522_Write(blockAddr, data)
103                     print ("After written , The data in block %d  is: "%(blockAddr))
104                     mfrc.MFRC522_Readstr(blockAddr)
105                 else:
106                     print ("Authentication error")
107                     return 0
108
109            elif cmd[0] == "clean":
110                 blockAddr = int(cmd[1])
111                 if((blockAddr<0) or (blockAddr>63)):
112                     print ("Invalid Address!")
113                 data = [0]*16
114                 # This is the default key for authentication
115                 key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
116                 # Authenticate
117                 status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
118                 # Check if authenticated
119                 if status == mfrc.MI_OK:
120                     print ("Before cleaning , The data in block %d  is: "%(blockAddr))
121                     mfrc.MFRC522_Readstr(blockAddr)
122                     mfrc.MFRC522_Write(blockAddr, data)
123                     print ("After cleaned , The data in block %d  is: "%(blockAddr))
124                     mfrc.MFRC522_Readstr(blockAddr)
125                 else:
126                     print ("Authentication error")
127                     return 0
127            elif cmd[0] == "halt":
128                 return 0
129            else :
130                 print ("Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n" "\tclean
       <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n")
131
132 def destroy():
133     print("Ending program")
134
135 if __name__ == "__main__":
136     setup()
137     try:
138         loop()
139     except KeyboardInterrupt:  # Ctrl+C captured, exit
140         destroy()
```

In the code, first create an MFRC522 class object.

```
mfrc = MFRC522.MFRC522()
```

In the function loop, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```
        if (inCmd == "scan"):
            print "Scanning ... "
            isScan = True
            while isScan:
                ............
                    if cmdloop(uid) < 1 :
                        isScan = False
        elif inCmd == "quit":
            destroy()
            exit(0)
        else :
            print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit program\n"
```

The function cmdloop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```
def cmdloop(cardID):
    pass
    while(True):
        dis_ConmandLine()
        dis_CardID(cardID)
        inCmd = raw_input()
        cmd = inCmd.split(" ")
        print cmd
        if(cmd[0] == "read"):
            ............
        elif cmd[0] == "dump":
            ............
        elif cmd[0] == "write":
            ............
        elif cmd[0] == "clean":
            ............
        elif cmd[0] == "halt":
            return 0
        else :
            print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n" "\tclean
<blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"
```

The file "MFRC522.py" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Chapter 25 Web IoT

In this chapter, we will learn how to use GPIO to control the RPi remotely via a network and how to build a WebIO service on the RPi.
This concept is known as "IoT" or Internet of Things. The development of IoT will greatly change our habits and make our lives more convenient and efficient

Project 25.1 Remote LED

In this project, we need to build a WebIOPi service, and then use the RPi GPIO to control an LED through the web browser of phone or PC.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1	Resistor 220Ω x1
Jumper M/M x2		

Circuit

Schematic diagram
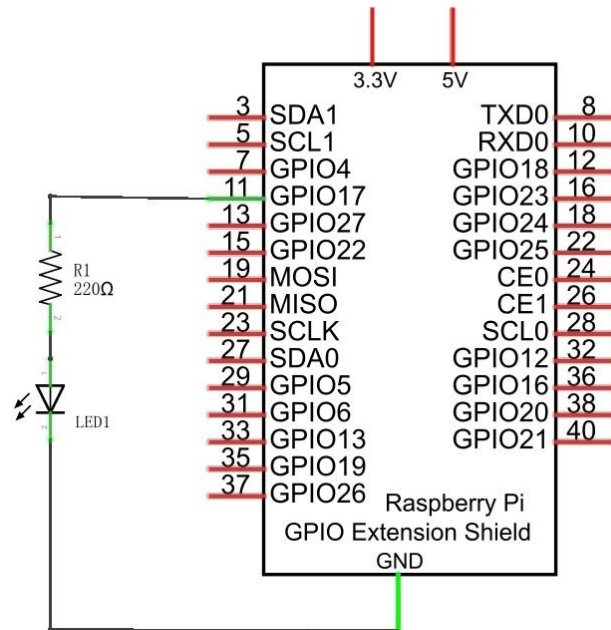
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
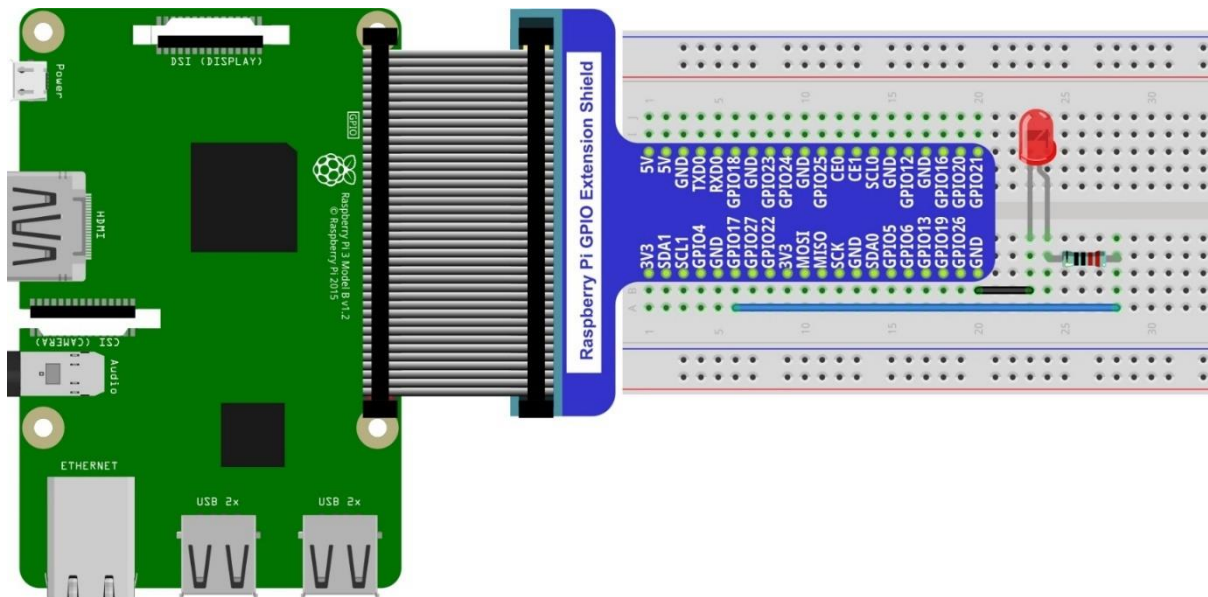
Solution from E-Tinkers

Here is a solution fromblog E-Tinkers, author Henry Cheung. For more details, please refer to link below:
https://www.e-tinkers.com/2018/04/how-to-control-raspberry-pi-gpio-via-http-web-server/

1, Make sure you have set python3 as default python. Then run following command in terminal to install http.server in your Raspberry Pi.

```
sudo apt-get install http.server
```

2, Open WebIO.py

```
cd ~/Freenove_Kit/Code/Python_GPIOZero_Code/25.1.1_WebIO
geany WebIO.py
```

3, Change the host_name into your Raspberry Pi IP address.

```
host_name = '192.168.1.112'    # Change this to your Raspberry Pi IP address
```

Then run the code WebIO.py

3, Visit http://192.168.1.112:8000/ in web brower on compter under local area networks. **Change IP to your Raspberry Pi IP address.**

WebIOPi Service Framework

Note: **If you have a Raspberry Pi 4B, you may have some trouble. The reason for changing the file in the configuration process is that the newer generation models of the RPi CPUs are different form the older ones and you may not be able to access the GPIO Header at t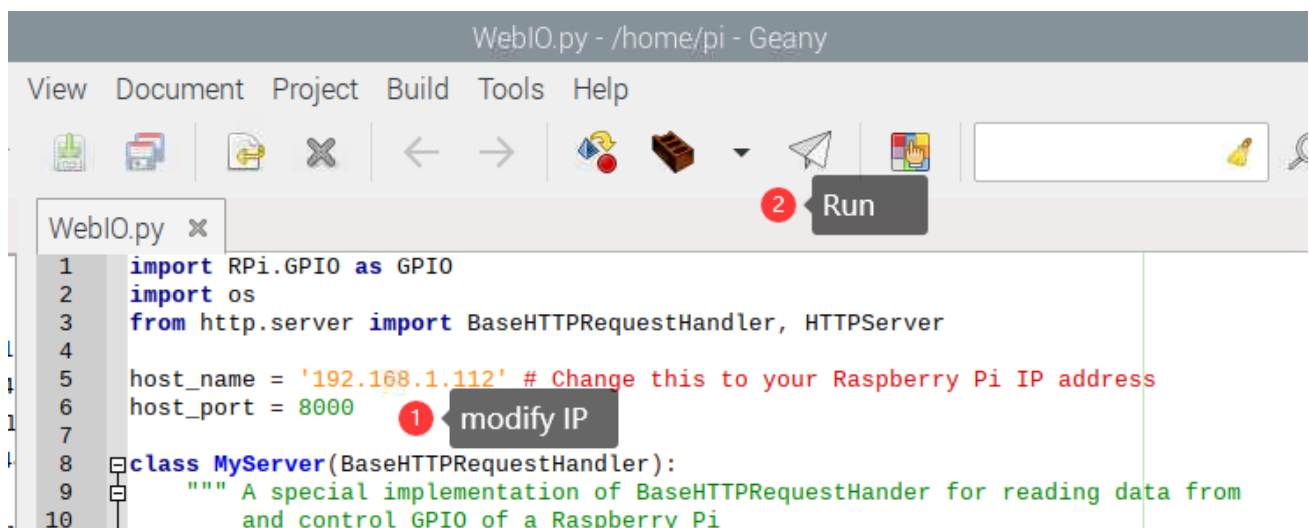he end of this tutorial.** A solution to this is given in an online tutorial by from E-Tinkers blogger Henry Cheung. For more details, please refer to previouse section.

The following is the key part of this chapter. The installation steps refer to WebIOPi official. And you also can directly refer to the official installation steps. The latest version (in 2016-6-27) of WebIOPi is 0.7.1. So, you may encounter some issues in using it. We will explain these issues and provide the solution in the following installation steps.

Here are the steps to build a WebIOPi:

Installation

1. Get the installation package. You can use the following command to obtain.

```
wget https://github.com/Freenove/WebIOPi/archive/master.zip -O WebIOPi.zip
```

2. Extract the package and generate a folder named "WebIOPi-master". Then enter the folder.

```
unzip WebIOPi.zip
cd WebIOPi-master/WebIOPi-0.7.1
```

3. Patch for Raspberry Pi B+, 2B, 3B, 3B+.

```
patch -p1 -i webiopi-pi2bplus.patch
```

4. Run setup.sh to start the installation, the process takes a while and you will need to be patient.

```
sudo ./setup.sh
```

5. If setup.sh does not have permission to execute, execute the following command

```
sudo sh ./setup.sh
```

Run

After the installation is completed, you can use the webiopi command to start running.

```
$ sudo webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]


    Options:
        -h, --help              Display this help
        -c, --config    file    Load config from file
        -l, --log       file    Log to file
        -s, --script    file    Load script from file
        -d, --debug             Enable DEBUG


    Arguments:
        port                    Port to bind the HTTP Server
```

Run webiopi with verbose output and the default config file:

```
sudo webiopi -d -c /etc/webiopi/config
```

The Port is 8000 in default. Now WebIOPi has been launched. Keep it running.

Access WebIOPi over local network

Under the same network, use a mobile phone or PC browser to open your RPi IP address, and add a port number like 8000. For example, my personal Raspberry Pi IP address is 192.168.1.109. Then, in the browser, I then should input: http://192.168.1.109:8000/

Default user is "webiopi" and password is "raspberry"。

Then, enter the main control interface:

WebIOPi Main Menu

GPIO Header

Control and Debug the Raspberry Pi GPIO with a display which looks like the physical header.

GPIO List

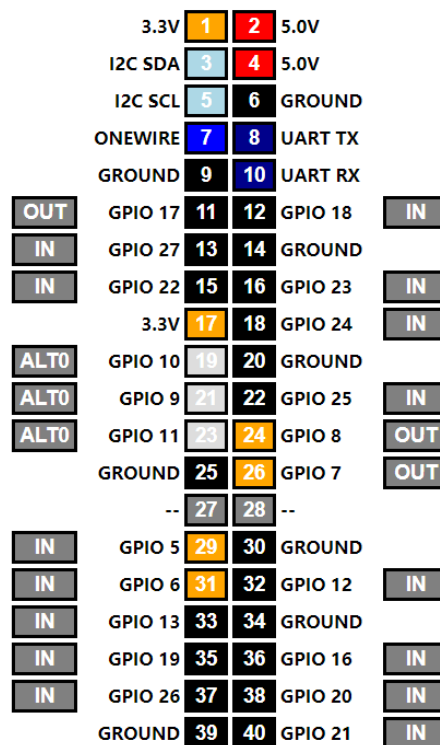Control and Debug the Raspberry Pi GPIO ordered in a single column.

Serial Monitor

Use the browser to play with Serial interfaces configured in WebIOPi.

Devices Monitor

Control and Debug devices and circuits wired to your Pi and configured in WebIOPi.

Click on GPIO Header to enter the GPIO control interface.

	3.3V	1 2	5.0V
	I2C SDA	3 4	5.0V
	I2C SCL	5 6	GROUND
	ONEWIRE	7 8	UART TX
	GROUND	9 10	UART RX
OUT	GPIO 17	11 12	GPIO 18 IN
IN	GPIO 27	13 14	GROUND
IN	GPIO 22	15 16	GPIO 23 IN
	3.3V	17 18	GPIO 24 IN
ALT0	GPIO 10	19 20	GROUND
ALT0	GPIO 9	21 22	GPIO 25 IN
ALT0	GPIO 11	23 24	GPIO 8 OUT
	GROUND	25 26	GPIO 7 OUT
	--	27 28	--
IN	GPIO 5	29 30	GROUND
IN	GPIO 6	31 32	GPIO 12 IN
IN	GPIO 13	33 34	GROUND
IN	GPIO 19	35 36	GPIO 16 IN
IN	GPIO 26	37 38	GPIO 20 IN
	GROUND	39 40	GPIO 21 IN

Control methods：
- Click/Tap the OUT/IN button to change GPIO direction.
- Click/Tap pins to change the GPIO output state.

Completed

According to the circuit we build, set GPIO17 to OUT, then click Header11 to control the LED.
You can end the webioPi in the terminal by "Ctr+C".

What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself a Raspberry Pi Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.