Getting Started

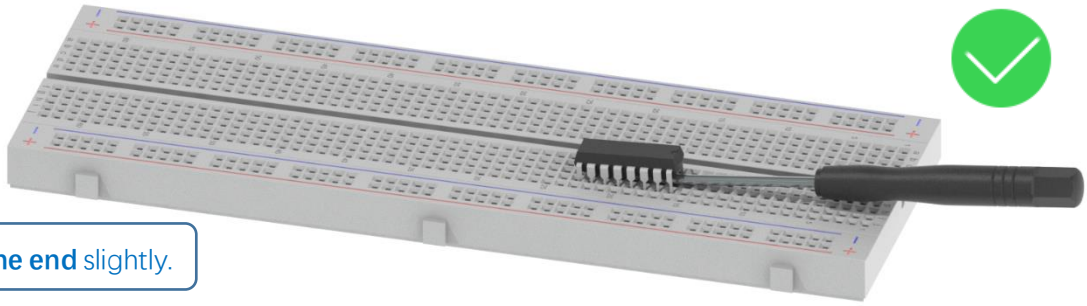
Thank you for choosing Freenove products!
After you download the ZIP file we provide. Unzip it and you will get a folder contains several files and folders. There are two PDF files:

- **Tutorial.pdf**
  It contains basic operations such as installing system for Raspberry Pi.
  The code in this PDF is in C and Python.
- **Processing.pdf** in Freenove_LCD1602_Starter_Kit_for_Raspberry_Pi\Processing
  The code in this PDF is in Java.

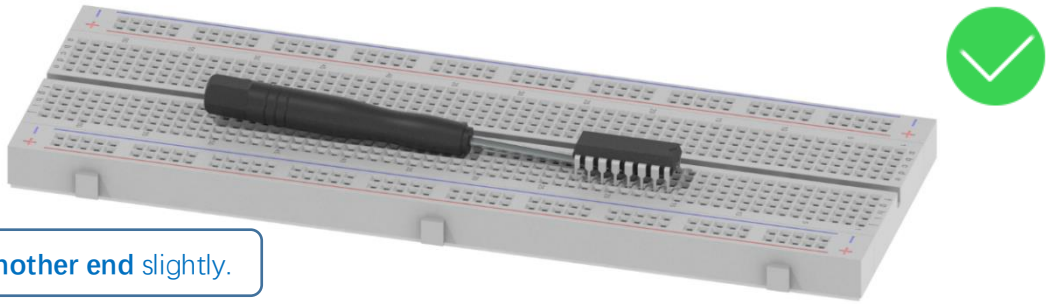
We recommend you to start with Tutorial.pdf first.
If you want to start with Processing.pdf or skip some chapters of Tutorial.pdf, you need to finish necessary steps in **Chapter 7 AD/DA** of **Tutorial.pdf** first.

Remove the Chips

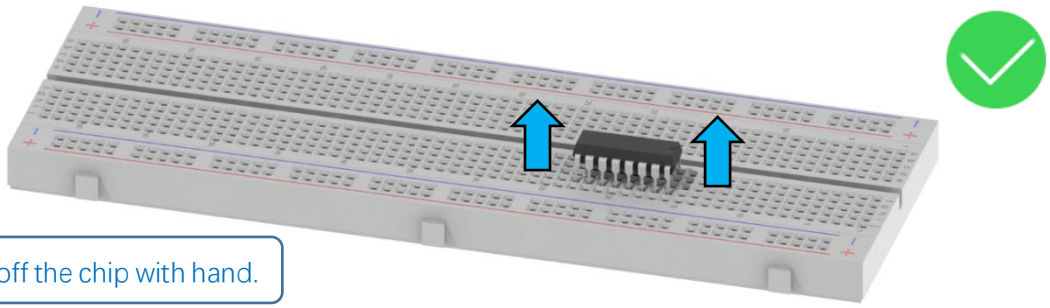
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



Step 1, lift **one end** slightly.

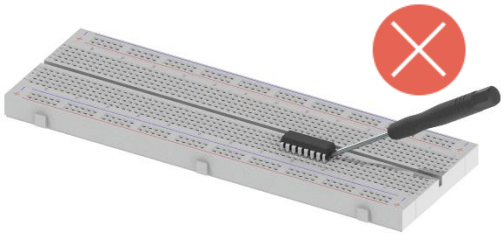


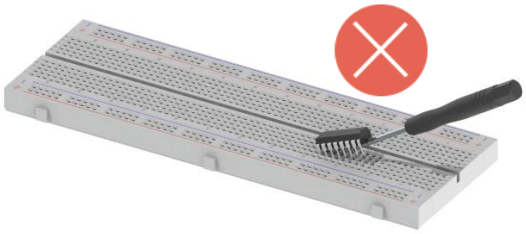
Step 2, lift **another end** slightly.



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

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** (**C** code and **Python** code). We provide both C and 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:

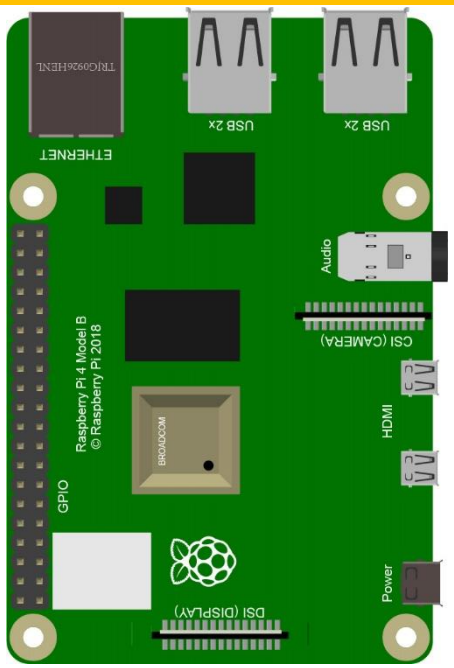
support@freenove.com

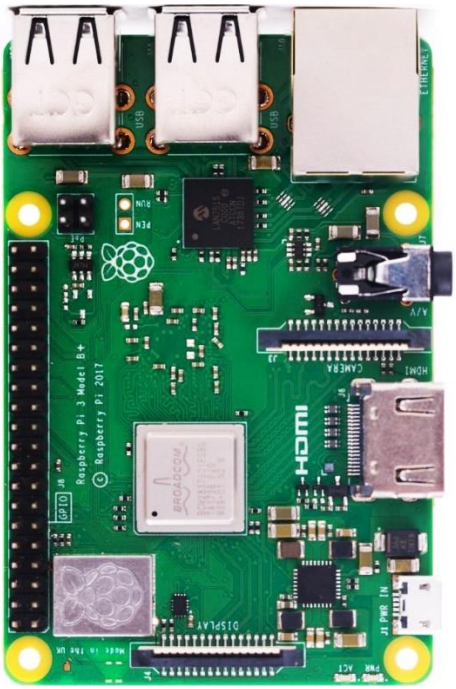
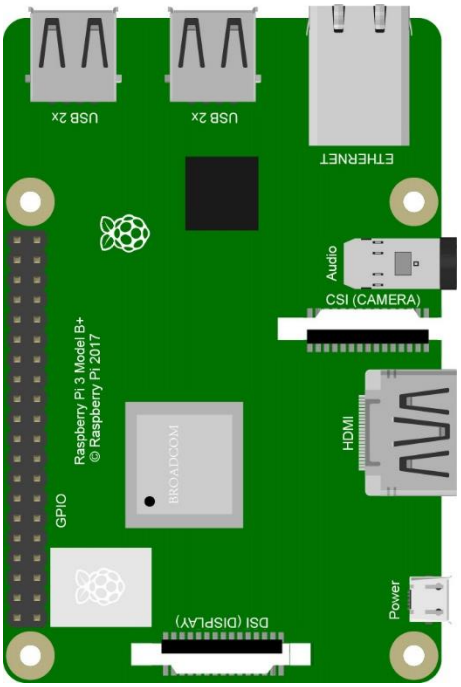
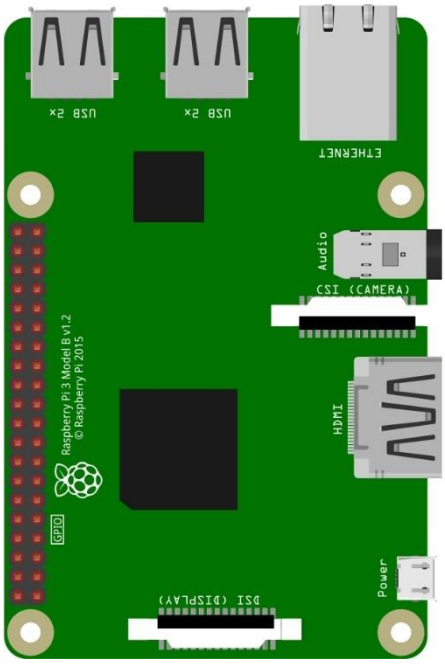
Raspberry Pi

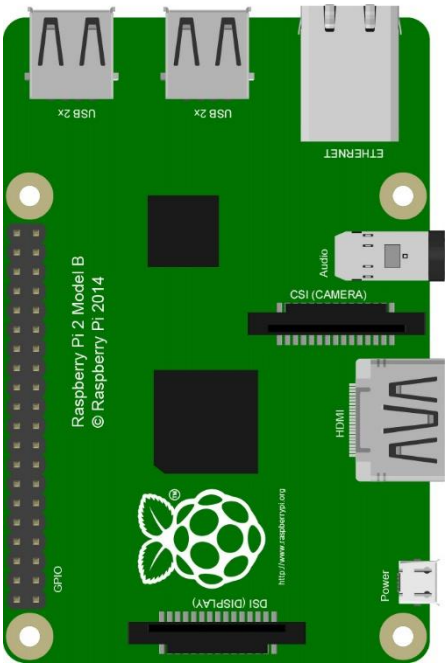
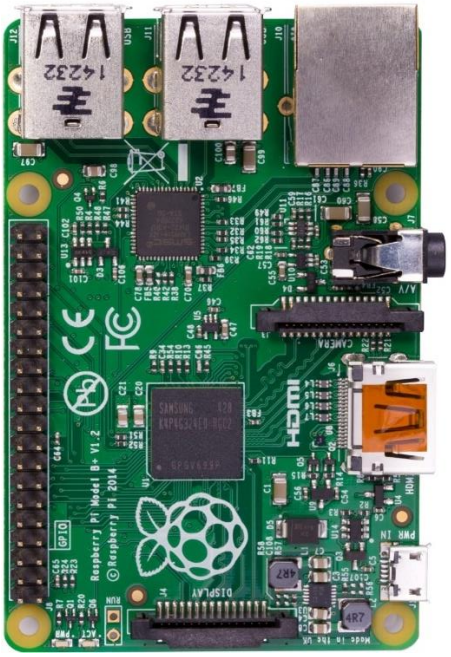
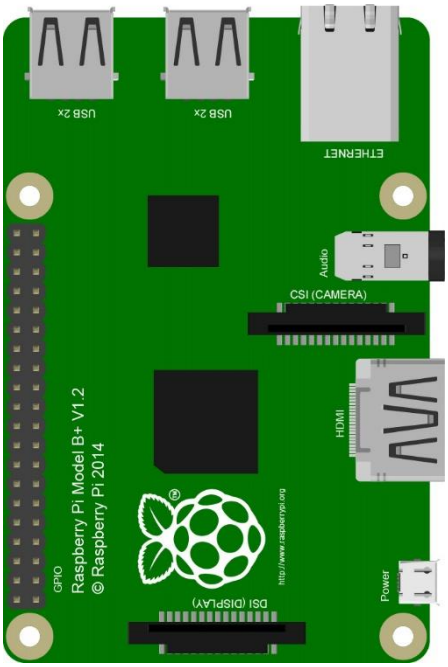
So far, at this writing, Raspberry Pi has advanced to its fourth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

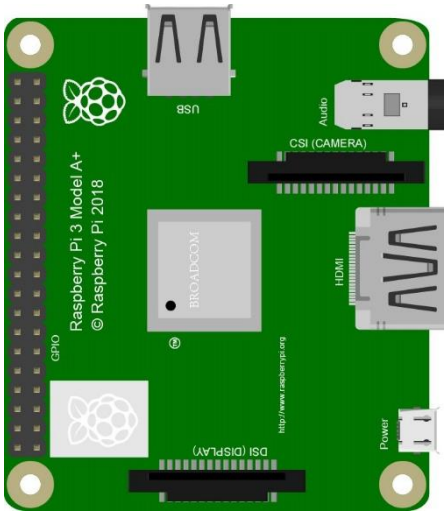
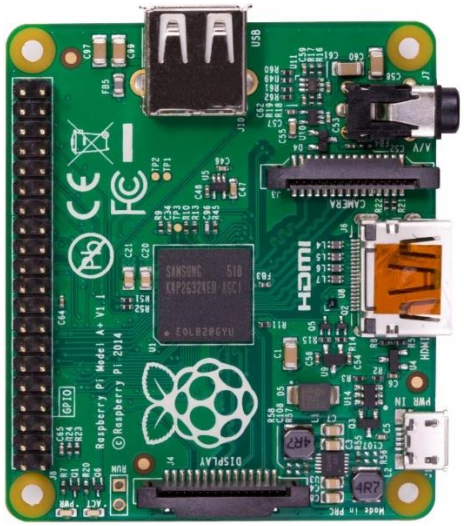
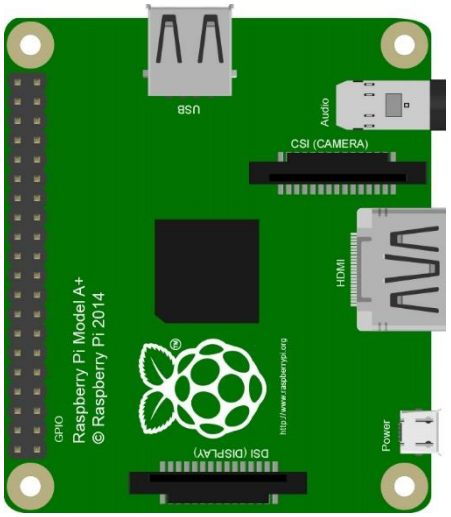
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility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

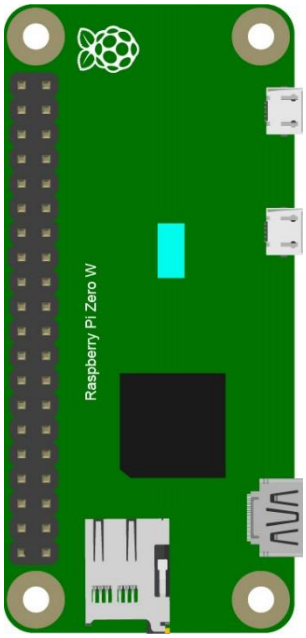
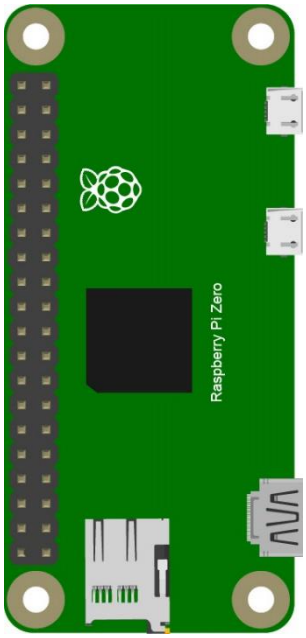
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

Actual image of Raspberry Pi 4 Model B:	CAD image of Raspberry Pi 4 Model B:
	

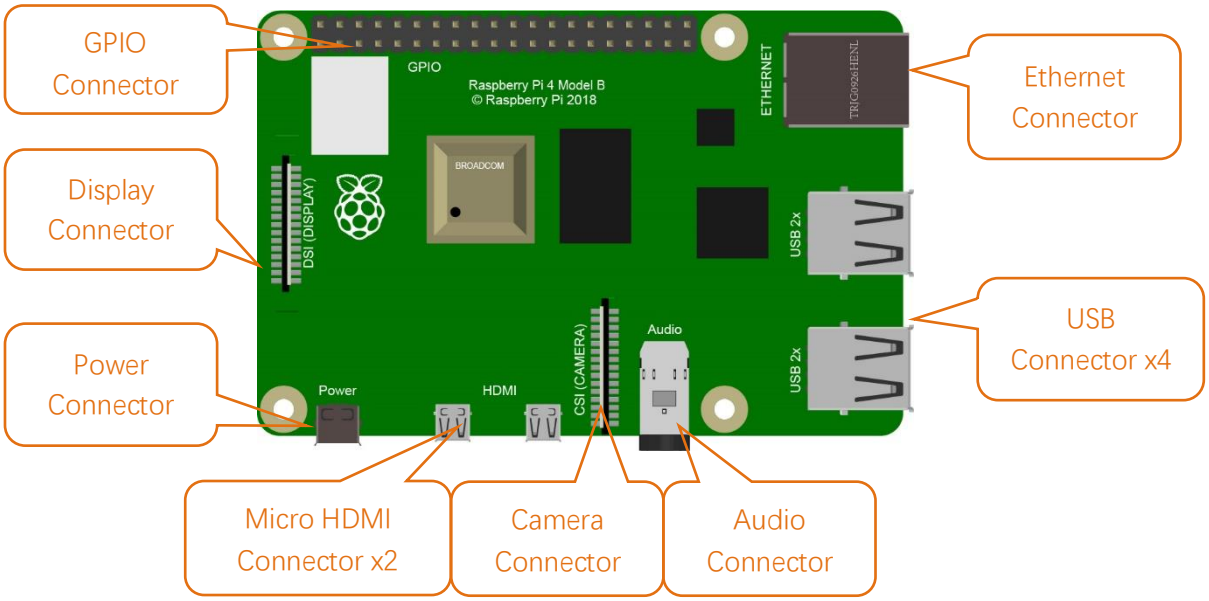
Actual image of Raspberry Pi 3 Model B+: 	CAD image of Raspberry Pi 3 Model B+: 
Actual image of Raspberry Pi 3 Model B: 	CAD image of Raspberry Pi 3 Model B: 

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+:
	

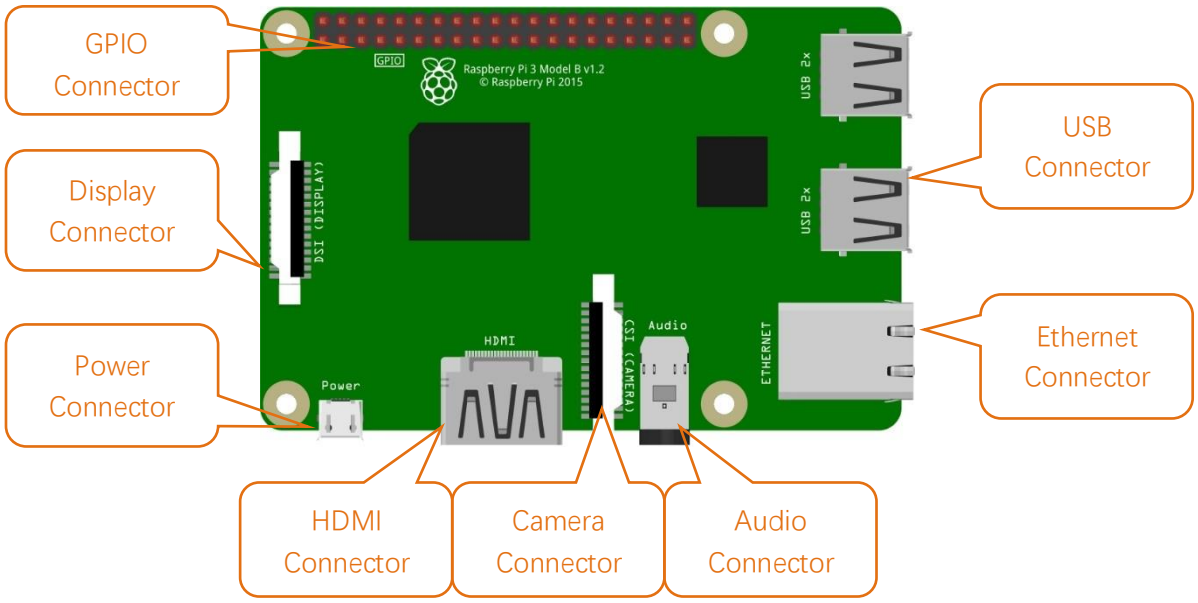
Actual image of Raspberry Pi 3 Model A+: 	CAD image of Raspberry Pi 3 Model A+: 
Actual image of Raspberry Pi 1 Model A+: 	CAD image of Raspberry Pi 1 Model A+: 

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: 

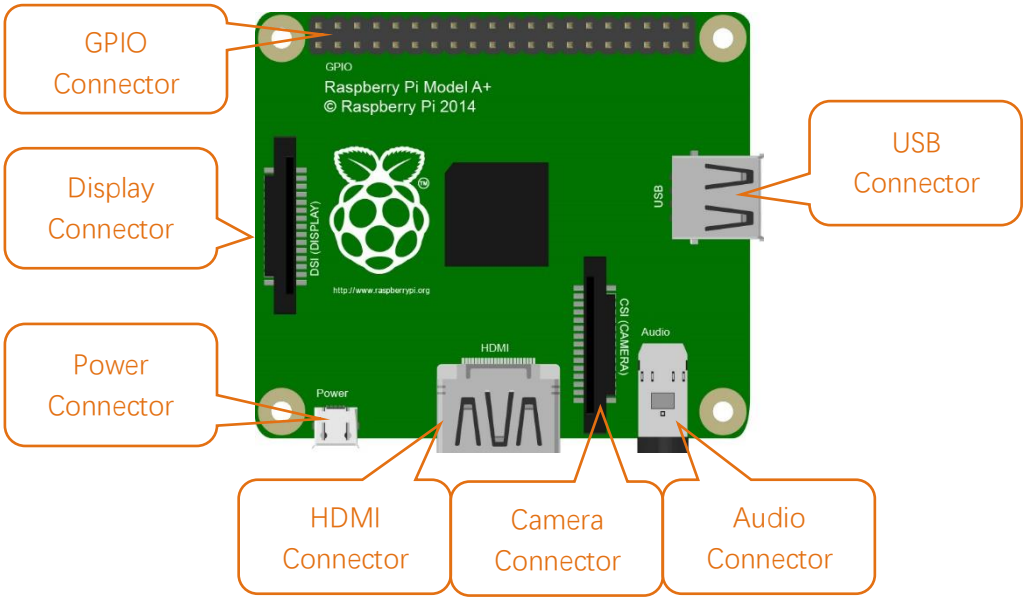
Hardware interface diagram of RPi 4B:



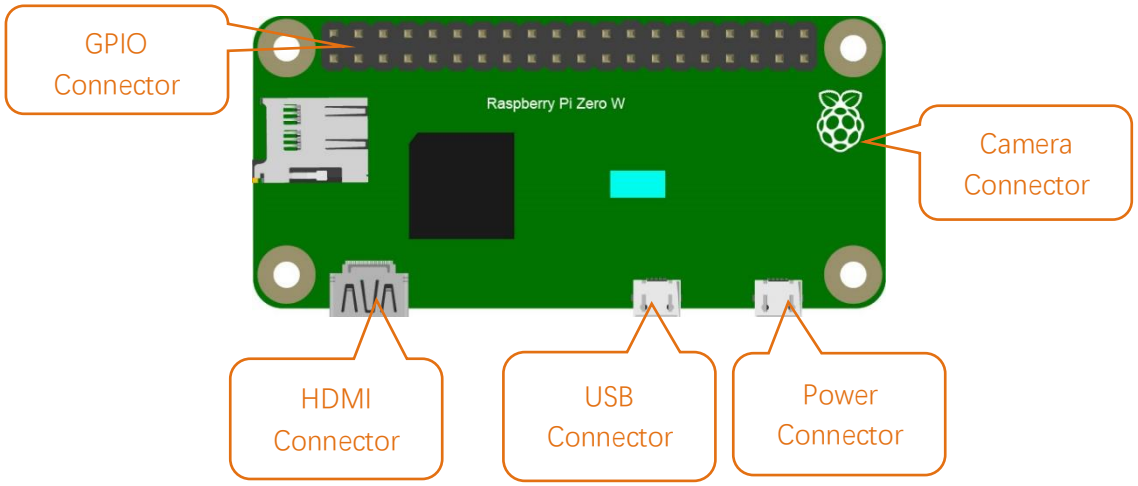
Hardware interface diagram of RPi 3B+/3B/2B/1B+:



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



Hardware interface diagram of RPi Zero/Zero W:

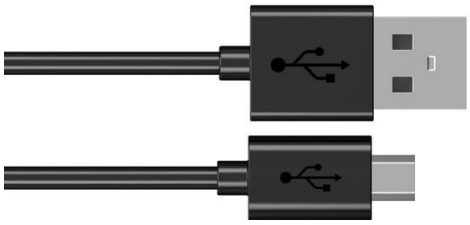


Installing an Operating System

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.

<table>
<tr><th></th><th>Pi Zero</th><th>Pi A+</th><th>Pi Zero W</th><th>Pi 3A+</th><th>Pi B+/2B</th><th>Pi 3B/3B+</th><th>Pi 4B</th></tr>
<tr><td>Monitor</td><td colspan="7">Yes (All)</td></tr>
<tr><td>Mouse</td><td colspan="7">Yes (All)</td></tr>
<tr><td>Keyboard</td><td colspan="7">Yes (All)</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td>Yes</td><td>No</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td colspan="6">No</td><td>Yes</td></tr>
<tr><td>Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)</td><td>Yes</td><td>No</td><td>Yes</td><td colspan="4">No</td></tr>
<tr><td>USB HUB</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td><td>No</td><td></td></tr>
<tr><td>USB to Ethernet Interface</td><td colspan="2" rowspan="2">select one from two or select two from two</td><td colspan="2" rowspan="2">optional</td><td>Internal Integration</td><td colspan="2" rowspan="2">Internal Integration</td></tr>
<tr><td>USB Wi-Fi Receiver</td><td>optional</td></tr>
</table>

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/HEywFsFrj3I

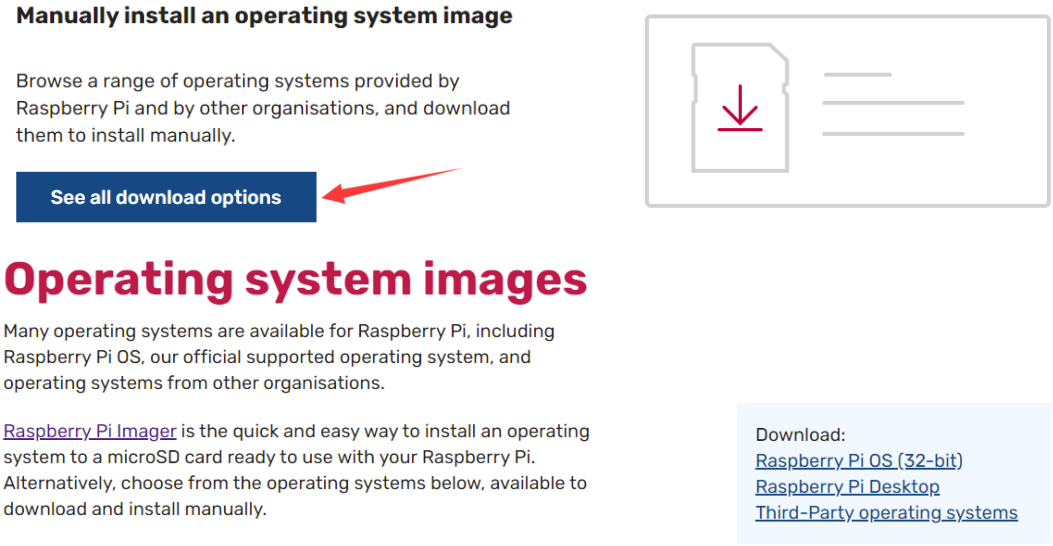
Automatically Method

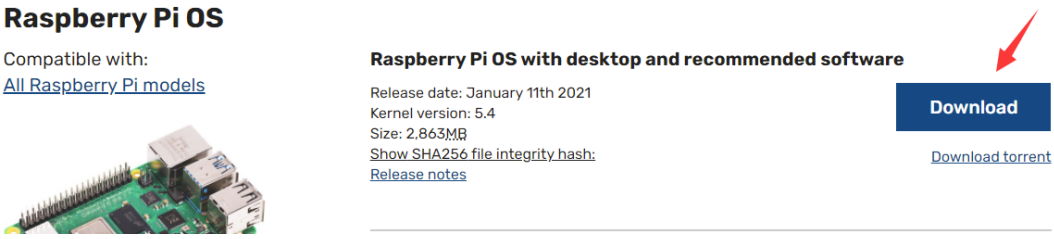
You can follow the official method to install the system for raspberry pi via visiting link below:
https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2
In this way, the system will be downloaded **automatically** via the application.

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/



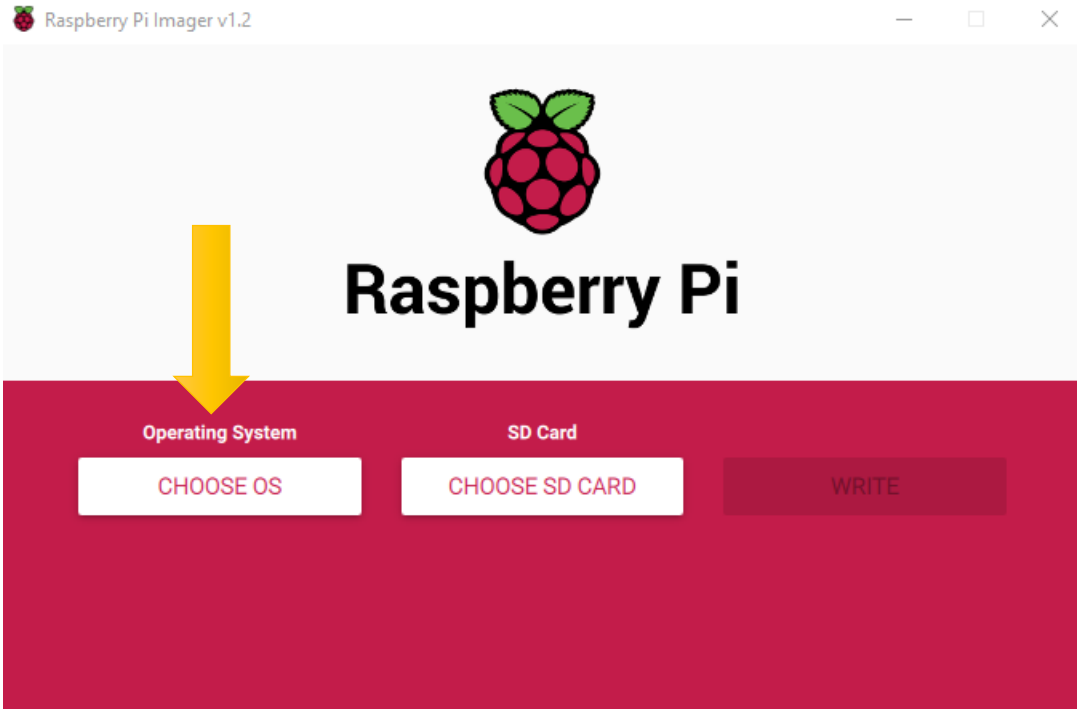


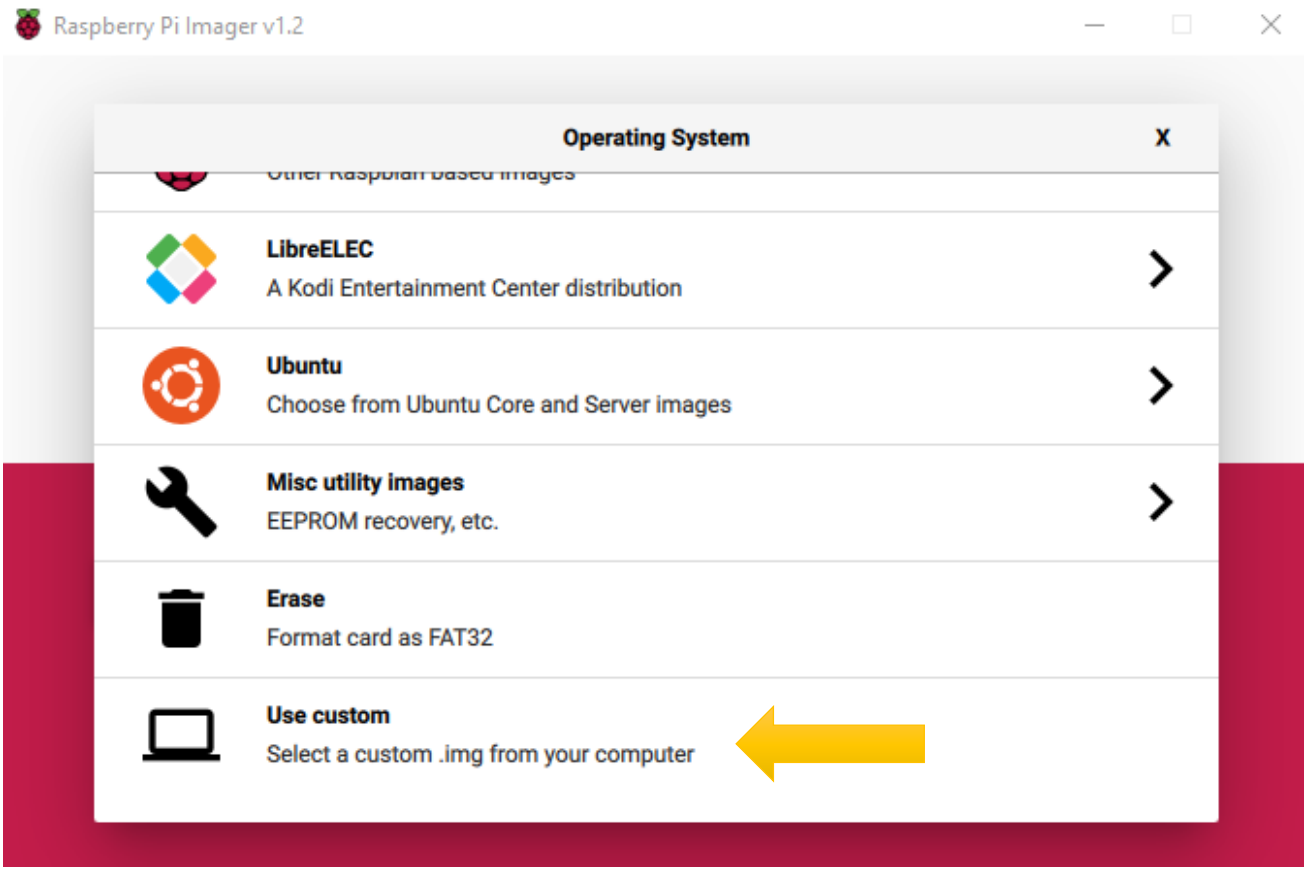
And then the zip file is downloaded.

Write System to Micro SD Card

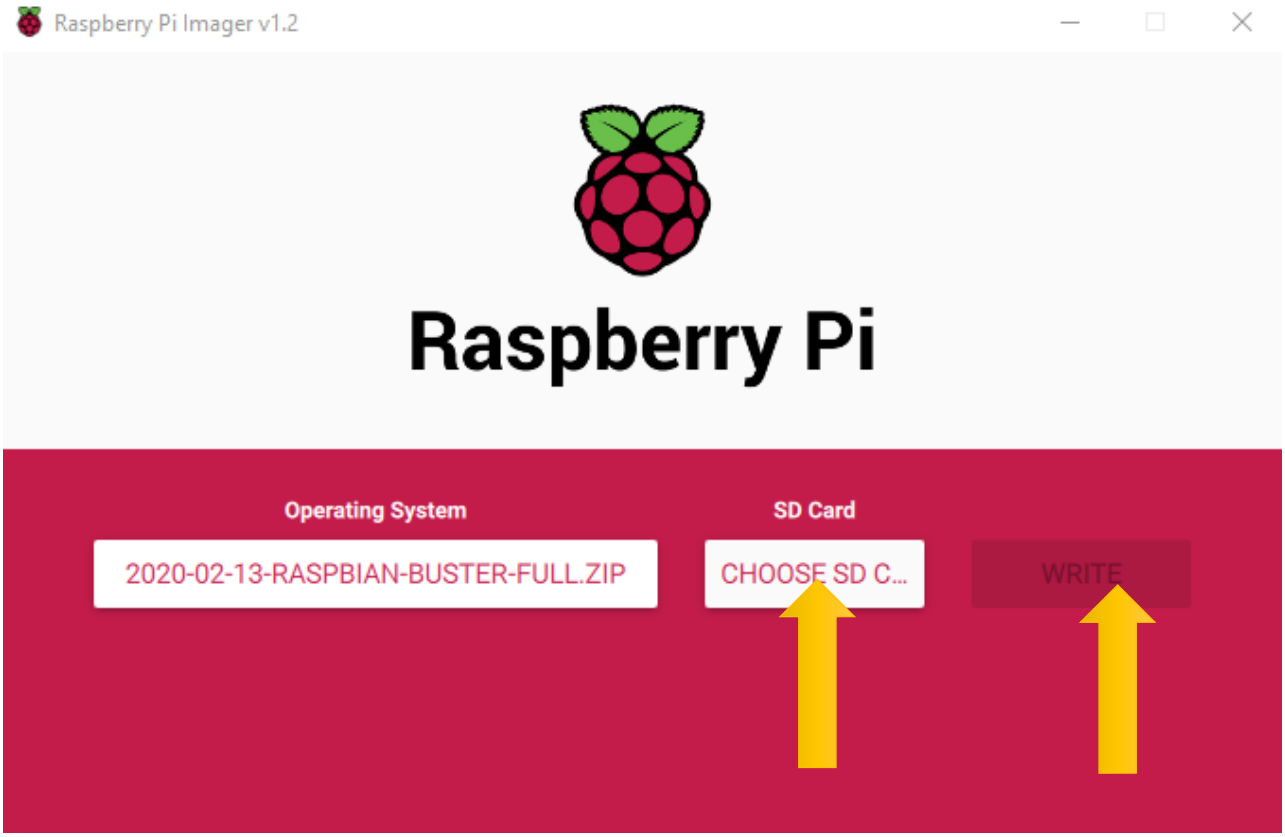
First, put your Micro **SD card** into card reader and connect it to USB port of PC.

Then open imager toll. Choose system that you just downloaded in Use custom.



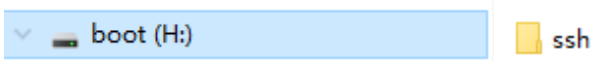


Choose the SD card. Then click "WRITE".



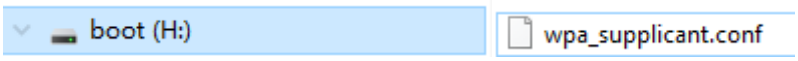
Enable ssh

If you don't have a separate monitor, after the system is written successfully, **create a folder named "ssh" under generated boot disk of Micro SD Card.**



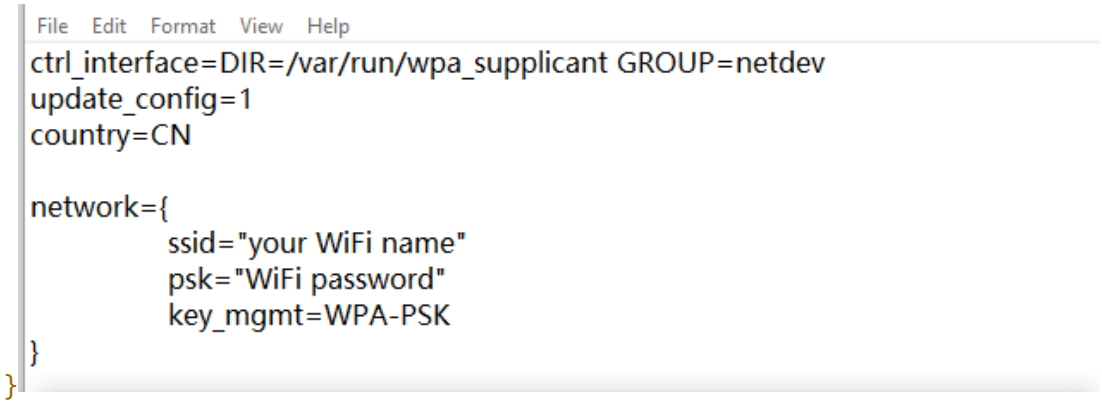
Configure WiFi

Create a file named **wpa_supplicant.conf** in **boot**.



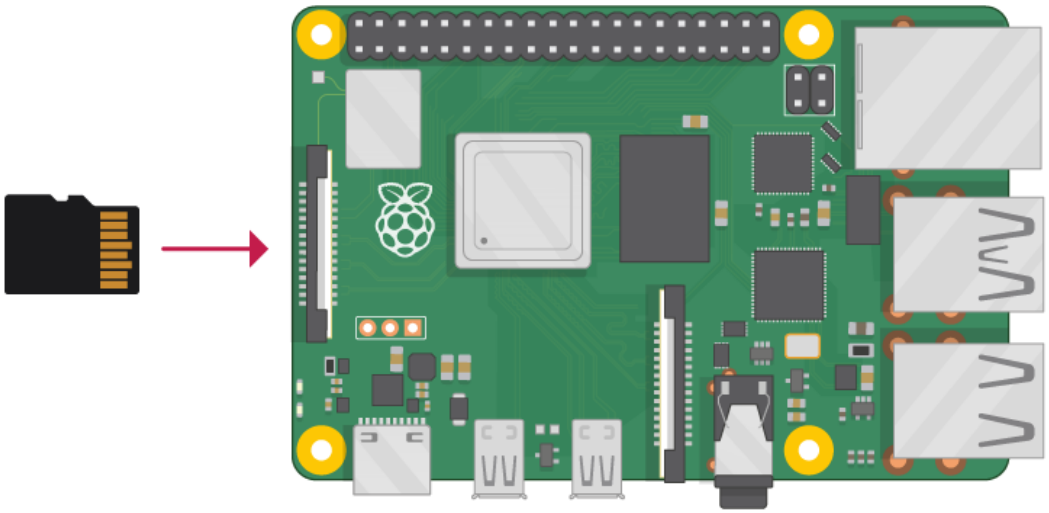
Open it with notepad. Add WiFi information. Input following content and save.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=CN
network={
    ssid="your WiFi name"
    psk="WiFi password"
    key_mgmt=WPA-PSK
}
```



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.

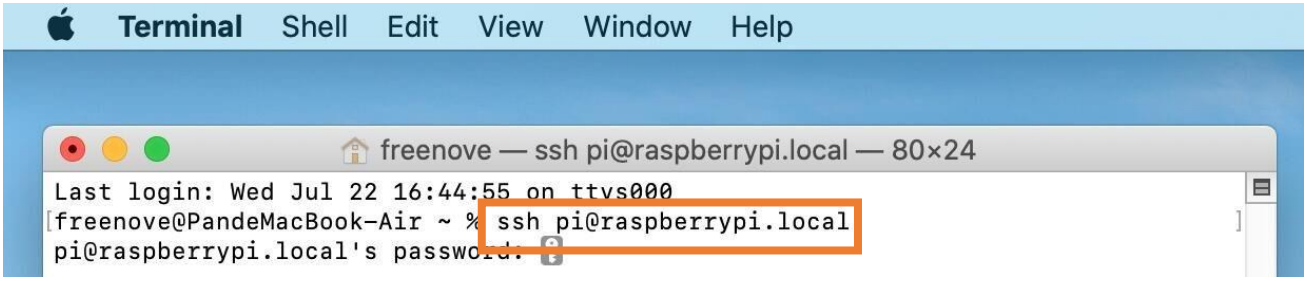
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

MAC OS Remote Desktop

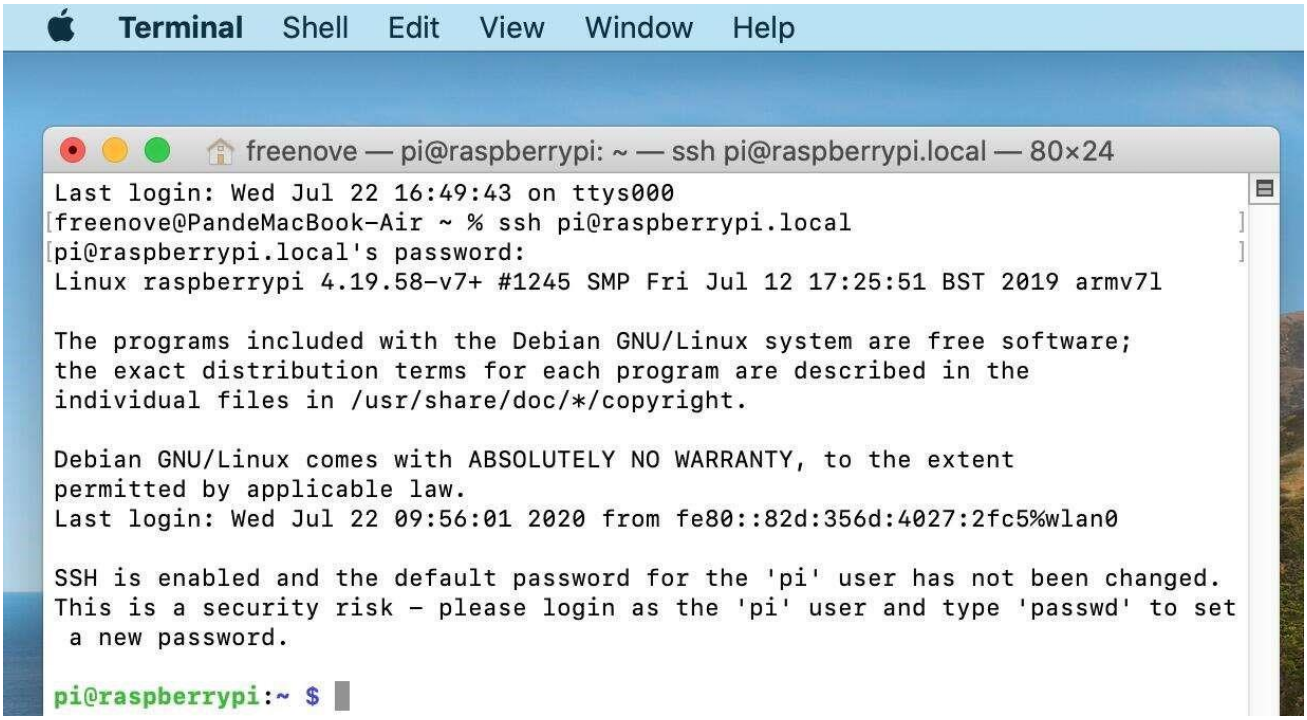
Open the terminal and type following command. **If this command doesn't work, please move to next page.**

```
ssh pi@raspberrypi.local
```

The password is **raspberry** by default, case sensitive.



You may need to type **yes** during the process.



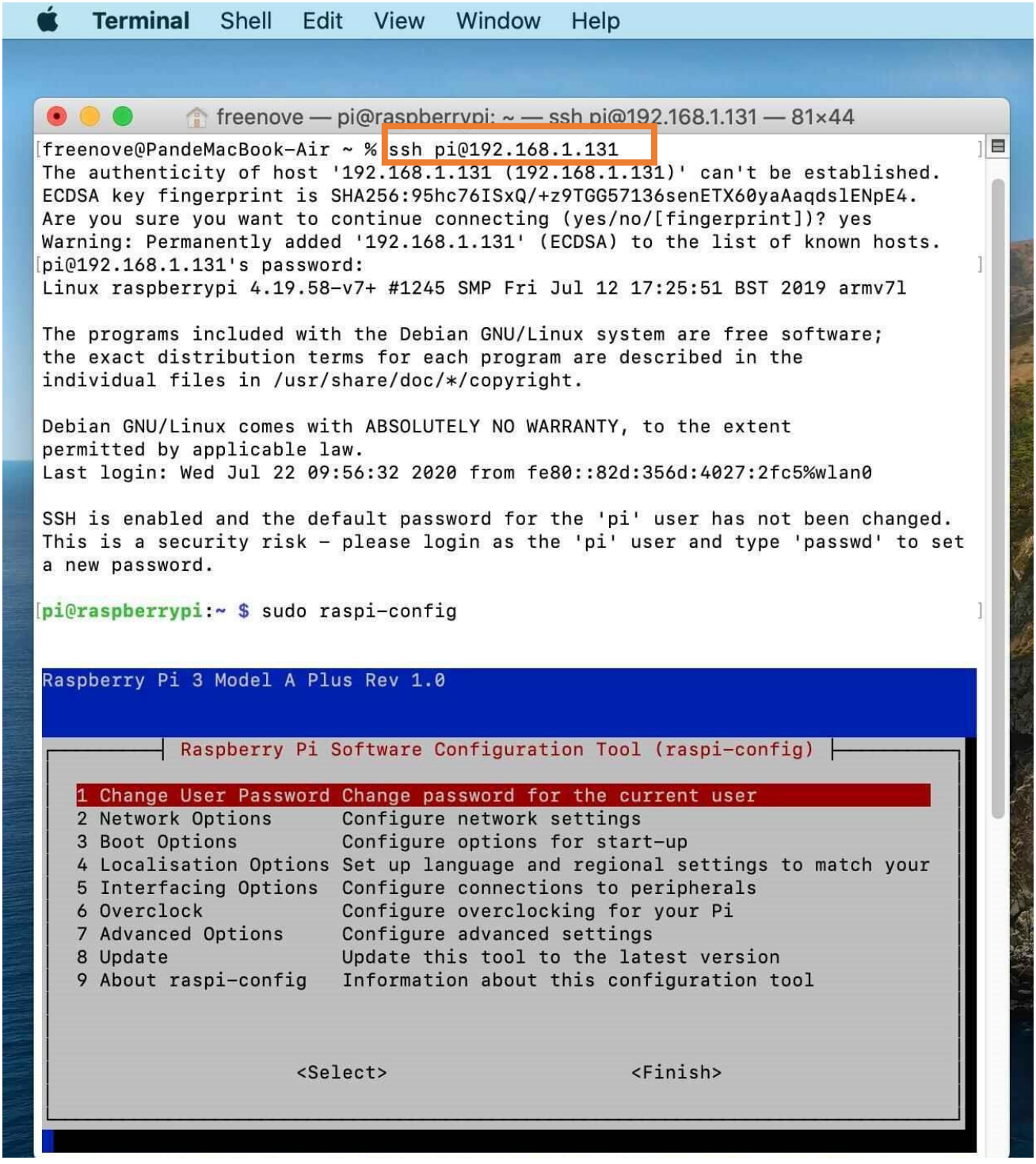
You can also use the IP address to log in Pi.

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.

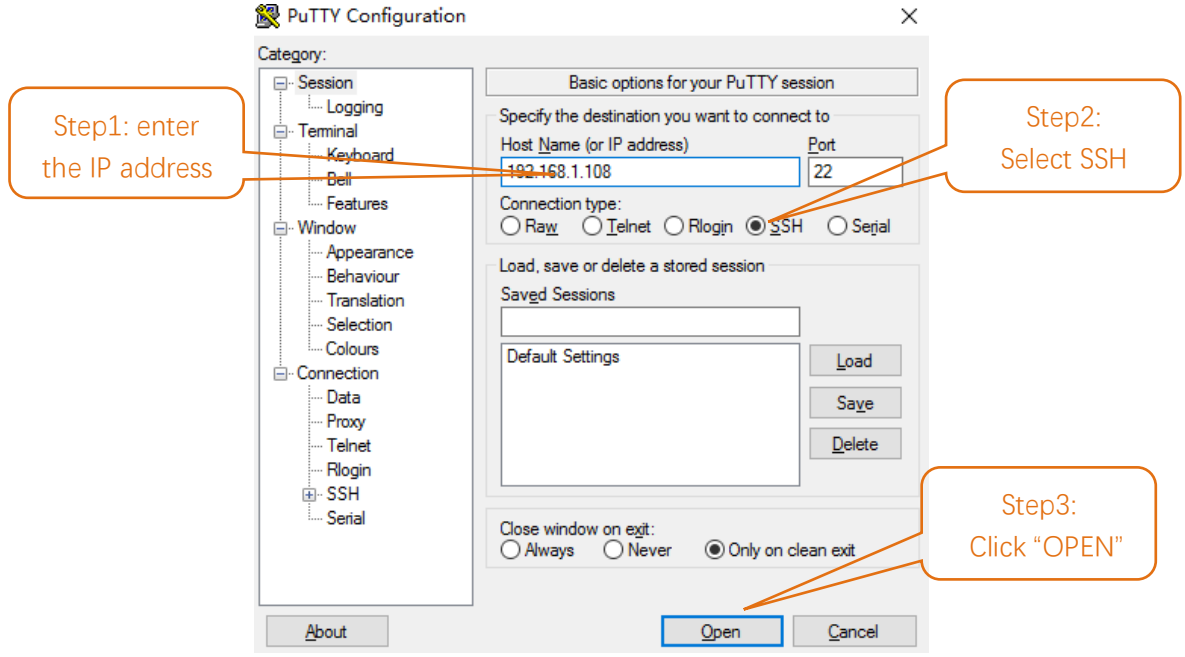


Then you can skip to VNC Viewer.

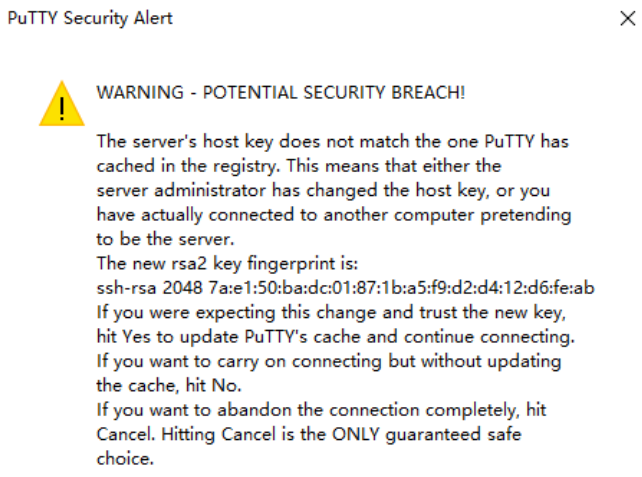
Windows OS Remote Desktop

The windows built-in application remote desktop corresponds to the Raspberry Pi xrdp service.
Download the tool software Putty. Its official address: http://www.putty.org/
Or download it here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Then use net cable to connect your RPi to the same router with your PC. Then put the system Micro SD Card prepared before into the slot of the RPi and turn on the power supply. Enter router client to inquiry IP address named “raspberry pi”. For example, my RPi IP address is “192.168.1.108".
Then open Putty, enter the address, select SSH, and then click "OPEN", as shown below:

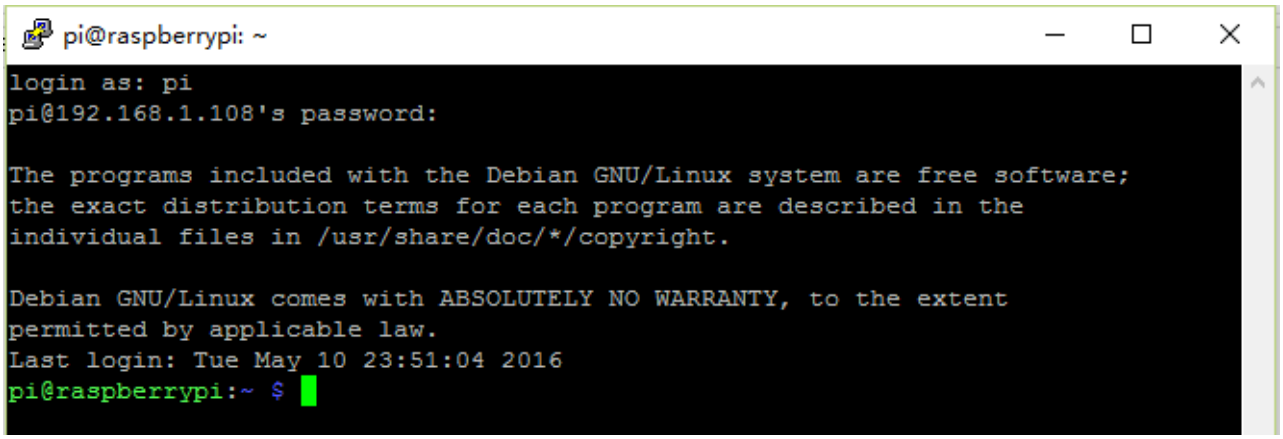


There will appear a security warning at first login. Just click “YES”.



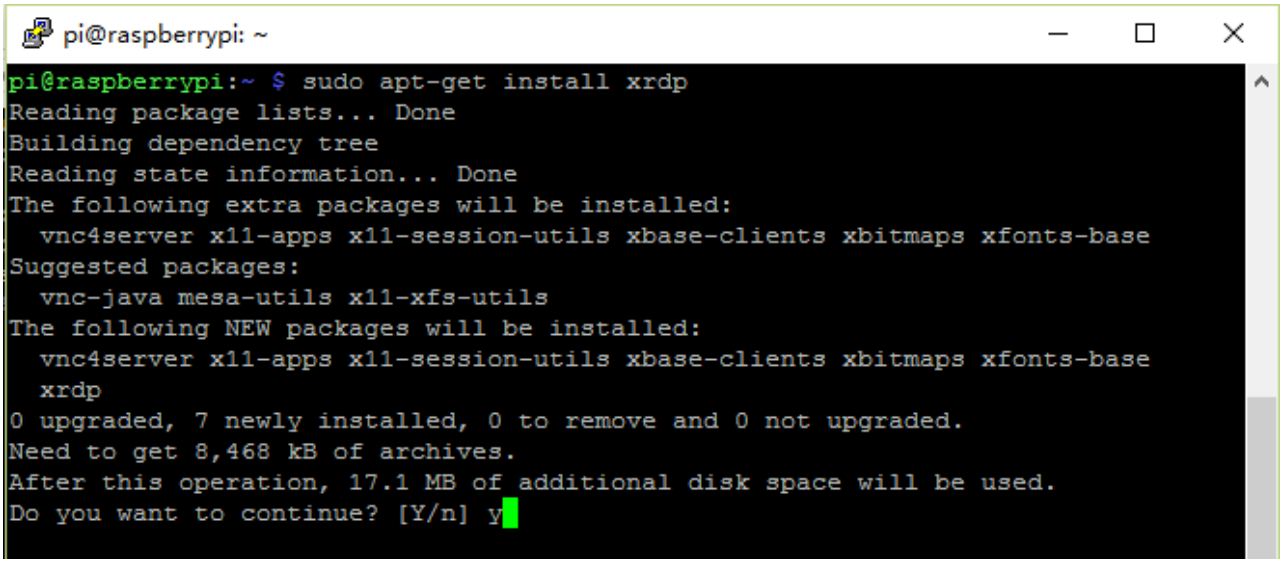
Then there will be a login interface. Login as: **pi**; password: **raspberry**. When you enter the password, there will be **no display** on the screen. This is normal. After the correct input, press "Enter" to confirm.

Then enter the command line of RPi, which means that you have successfully login to RPi command line mode.



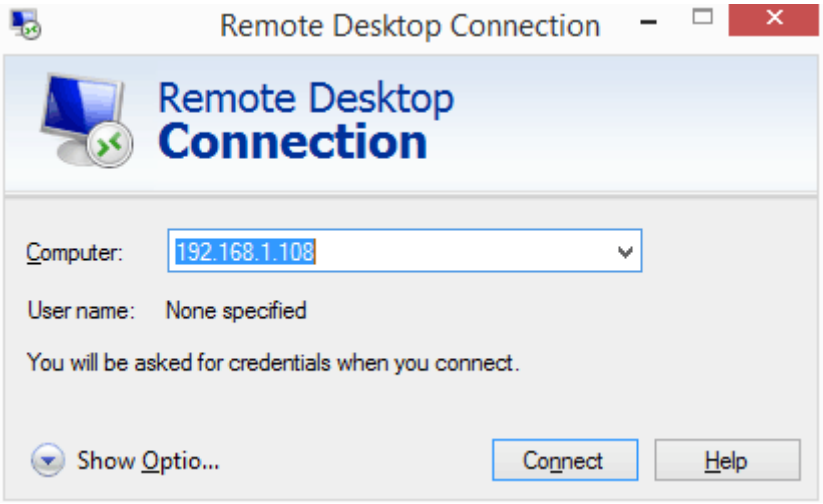
Next, install a xrdp service, an open source remote desktop protocol(xrdp) server, for RPi. Type the following command, then press enter to confirm:

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

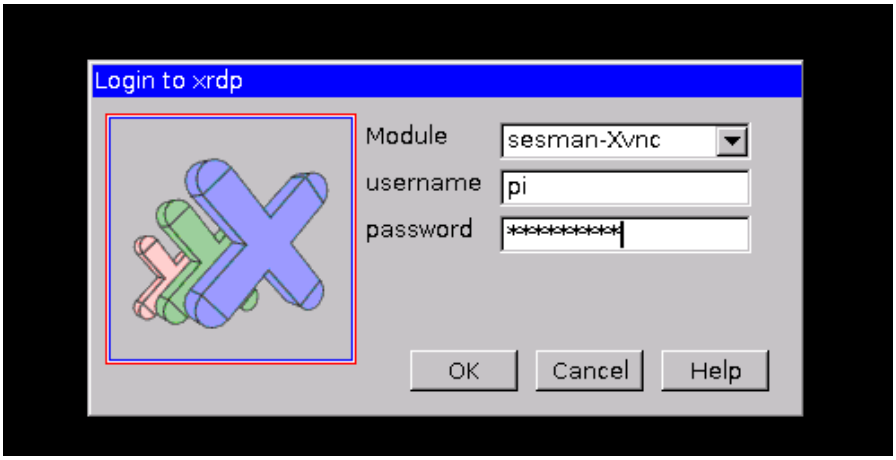


Enter "Y", press key "Enter" to confirm.

After the installation is completed, you can use Windows remote desktop applications to login to your RPi. Use "WIN+R" or search function, open the remote desktop application "mstsc.exe" under Windows, enter the IP address of RPi and then click "Connect".



Later, there will be xrdp login screen. Enter the user name and password of RPi (RPi default user name: pi; password: raspberry) and click "OK".



Later, you can enter the RPi desktop system.

Here, you have successfully used the remote desktop login to RPi.

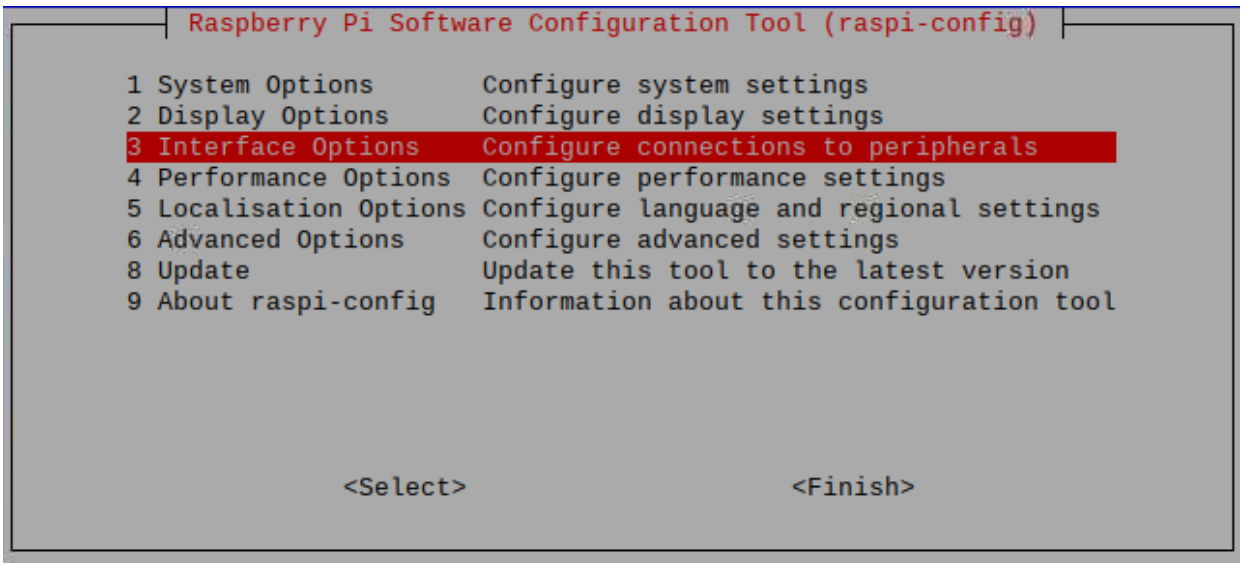
Raspberry Pi 4B/3B+/3B integrates a Wi-Fi adaptor. You can use it to connect to your Wi-Fi. Then you can use the wireless remote desktop to control your RPi.

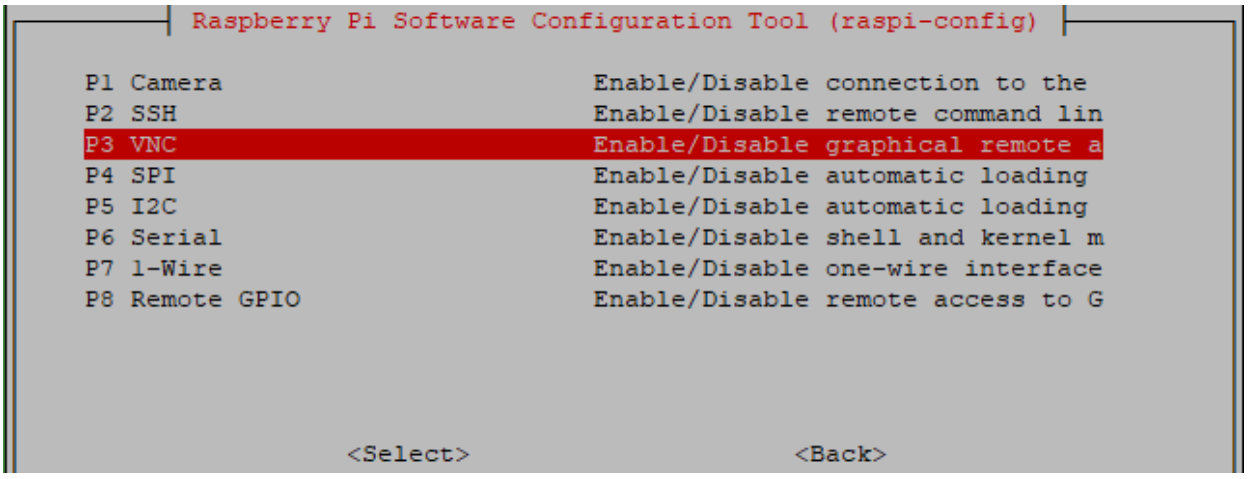
VNC Viewer & VNC

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.

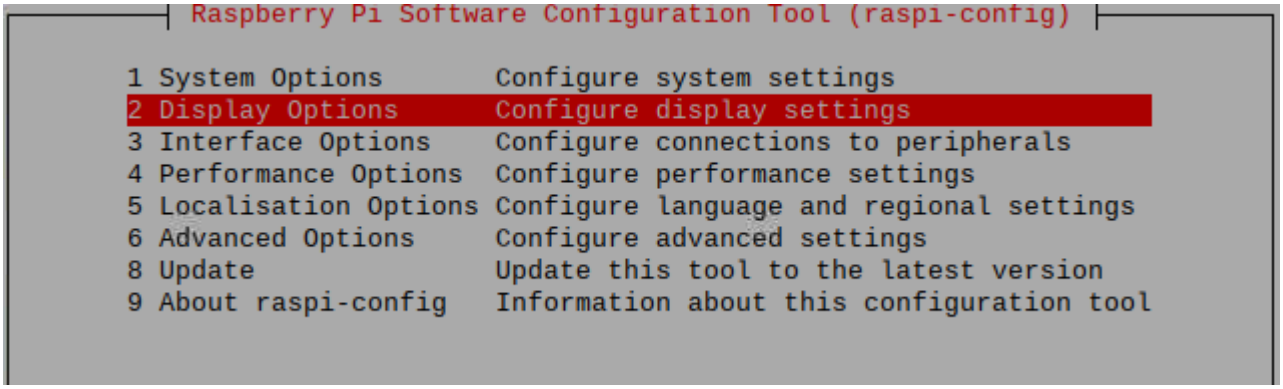
```
sudo raspi-config
```

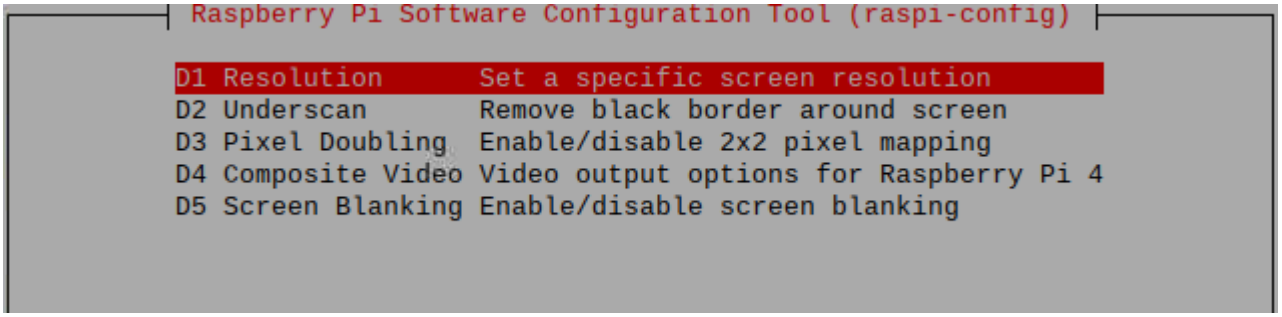


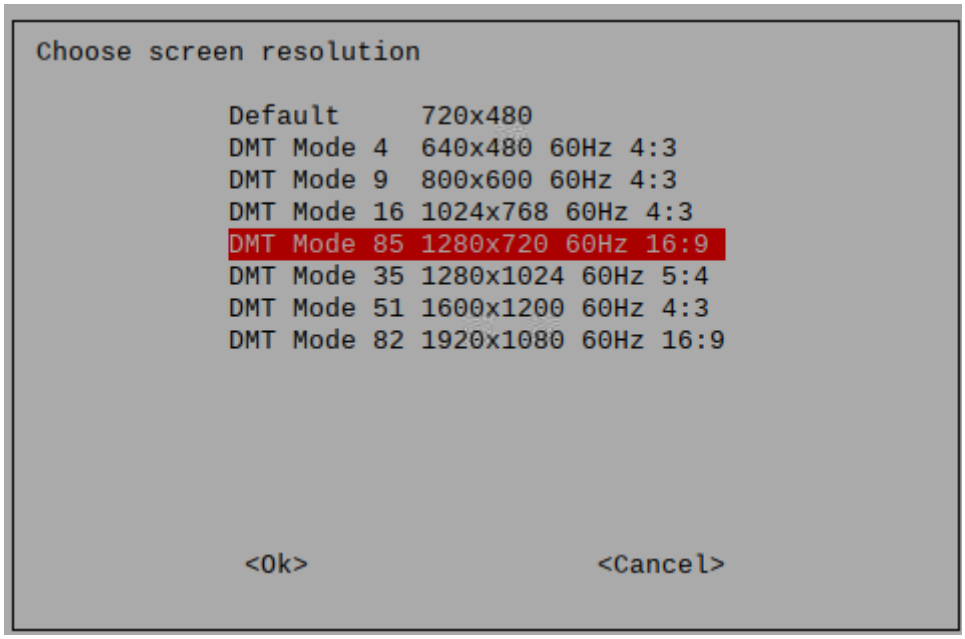


Set Resolution

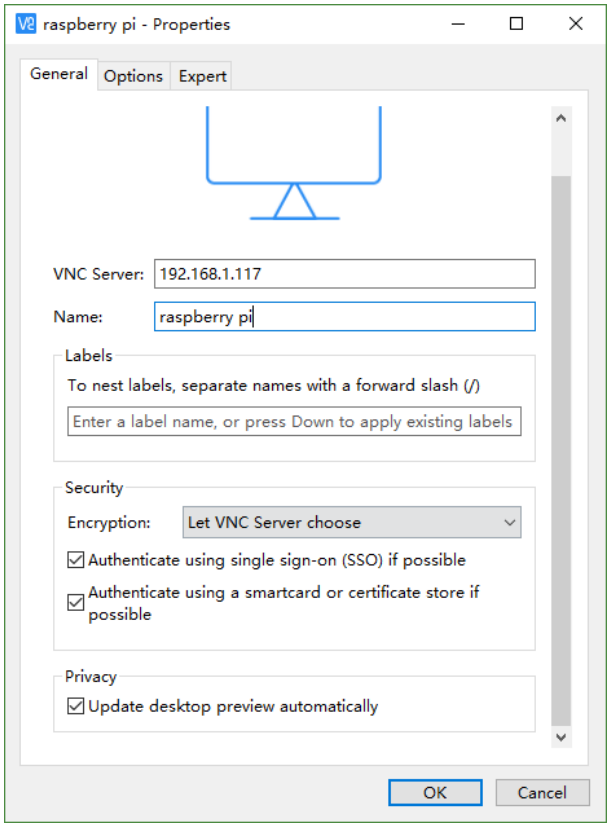
You can also set other resolutions. If you don't know what to set, you can set it as 1280x720 first.







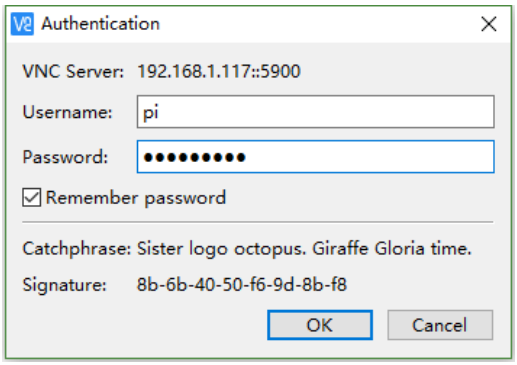
Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.



Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

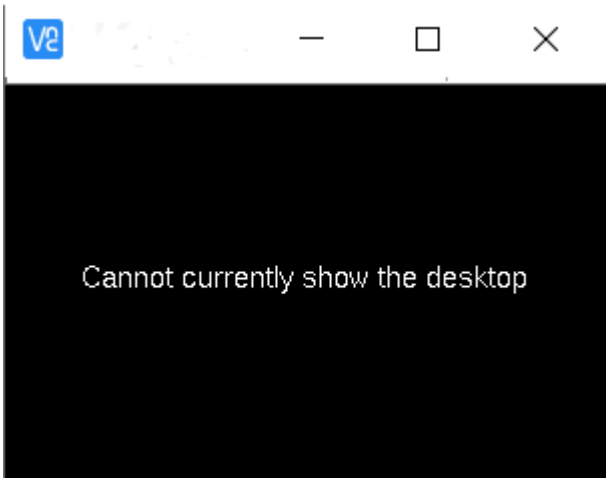
Then on the VNC Viewer panel, double-click new connection you just created,

and the following dialog box pops up.

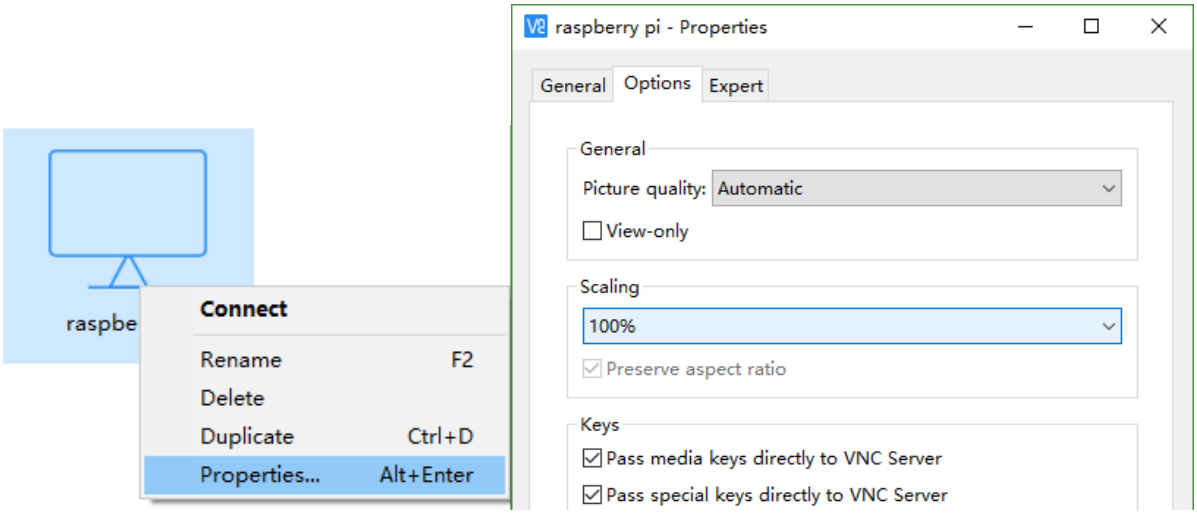


Enter username: **pi** and Password: **raspberry**. And click OK.

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.

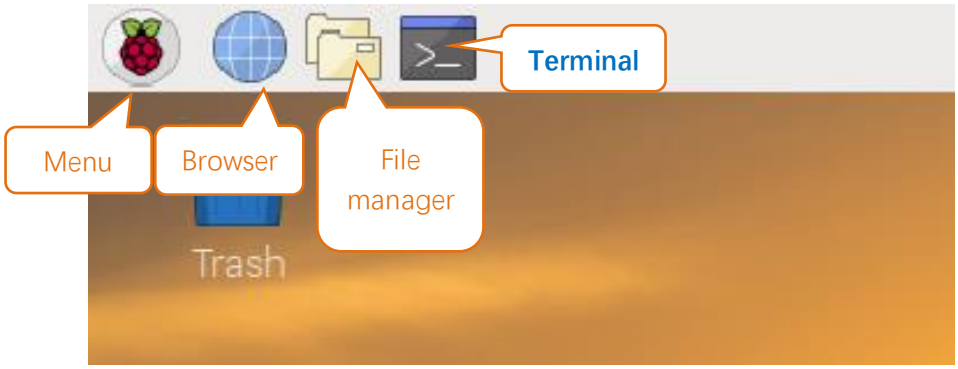
Chapter 0 Preparation

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.

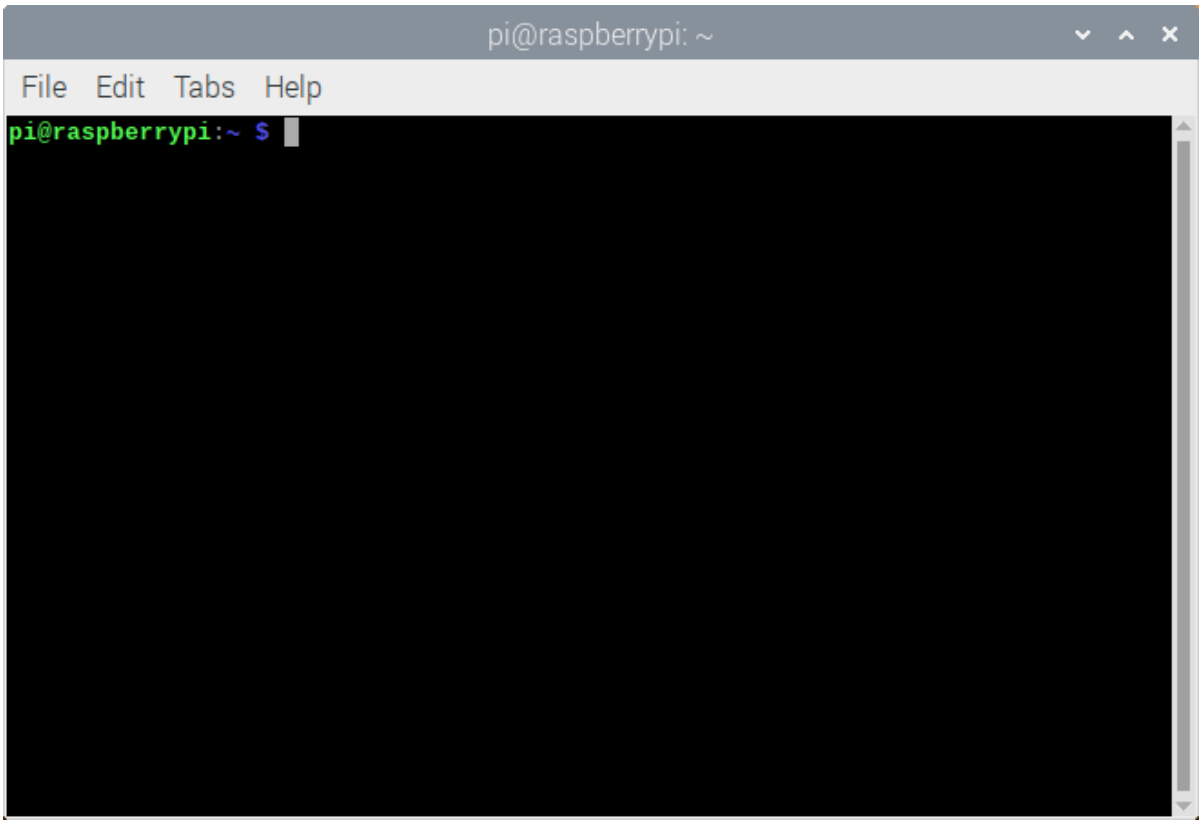
Linux Command

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.

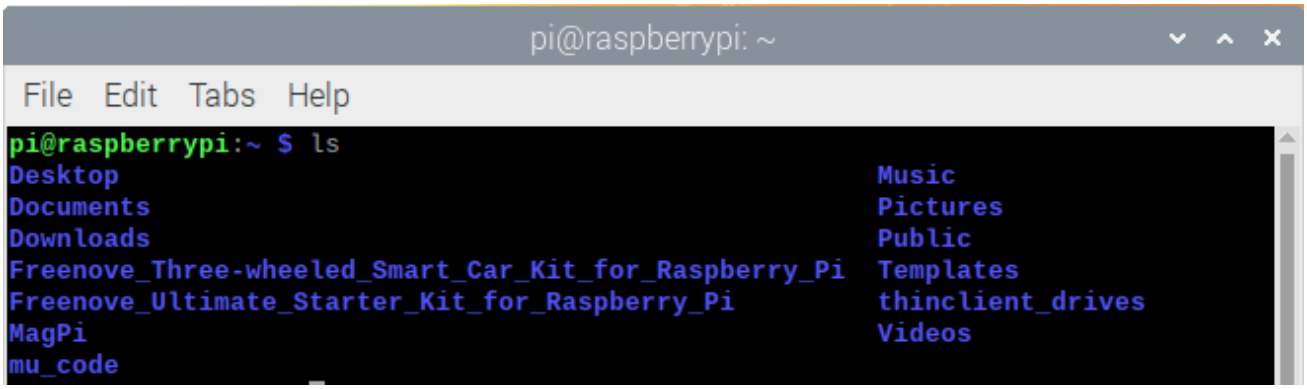


When you click the Terminal icon, following interface appears.



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.

```
pi@raspberrypi:~ $ pwd
/home/pi
```

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

```
pi@raspberrypi:~ $ cd /usr
pi@raspberrypi:/usr $ ls
bin  games  include  lib  local  man  sbin  share  src
pi@raspberrypi:/usr $ cd ~
pi@raspberrypi:~ $
```

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/
```

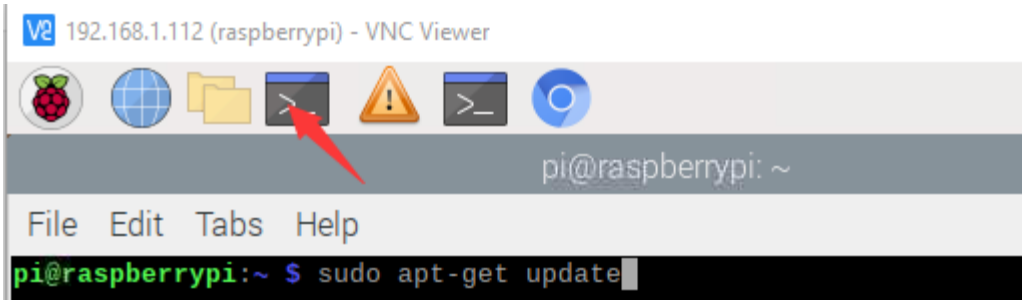
Install WiringPi

WiringPi is a GPIO access library written in C language for the used in the Raspberry Pi.

WiringPi Installation Steps

To install the WiringPi 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 WiringPi:

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




```
pi@raspberrypi:~ $ git clone https://github.com/WiringPi/WiringPi
Cloning into 'WiringPi'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 1483 (delta 15), reused 13 (delta 1), pack-reused 1442
Receiving objects: 100% (1483/1483), 793.91 KiB | 924.00 KiB/s, done.
Resolving deltas: 100% (918/918), done.
```

```
pi@raspberrypi:~ $ cd WiringPi
pi@raspberrypi:~/WiringPi $ ./build
wiringPi Build script
=====================
```

```
All Done.

NOTE: To compile programs with wiringPi, you need to add:
    -lwiringPi
  to your compile line(s) To use the Gertboard, MaxDetect, etc.
  code (the devLib), you need to also add:
    -lwiringPiDev
  to your compile line(s).
```

Run the gpio command to check the installation:

```
gpio -v
```

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

```
gpio version: 2.60
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
```

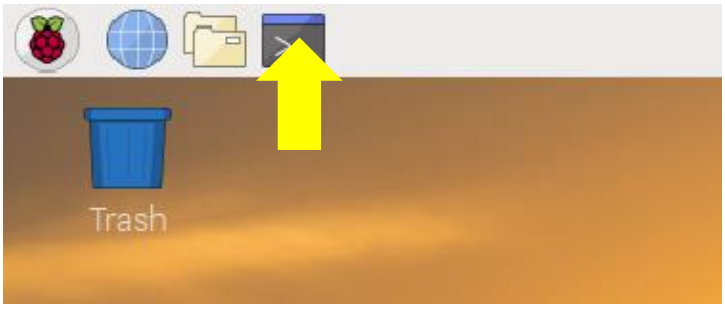
Obtain the Project Code

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. We provide both **C** language and **Python** language code for each project to allow ease of use for those who are skilled in either language.

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_LCD1602_Starter_Kit_for_Raspberry_Pi
```

(There is no need for a password. If you get some errors, please check your commands.)

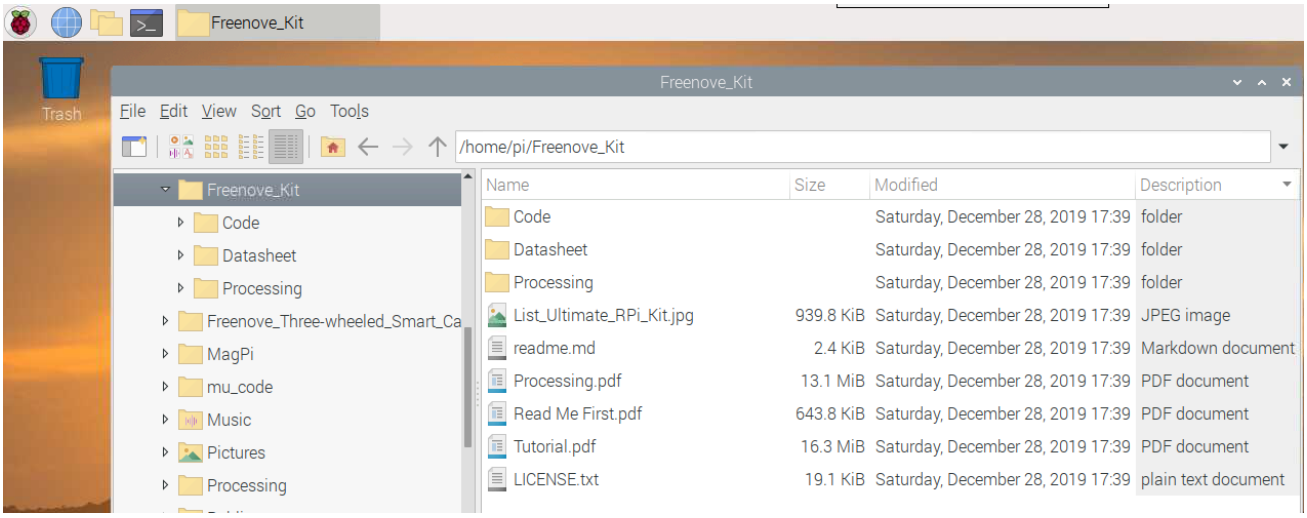


After the download is completed, a new folder "Freenove_LCD1602_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

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

```
mv Freenove_LCD1602_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

If you only use C/C++, you can skip this section.

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.

```
pi@raspberrypi:~ $ python2
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
[2]+  Stopped                 python2
pi@raspberrypi:~ $ python3
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.
>>>
```

Type "python", and Terminal shows that it links to python2.

```
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you want to use Python3 in Raspberry Pi, it is recommended to set python3 as default Python by following the steps below.

1. Enter directory /usr/bin

```
cd /usr/bin
```

2. Delete the old python link.

```
sudo rm python
```

3. Create new python links to python3.

```
sudo ln -s python3 python
```

4. Execute python to check whether the link succeeds.

```
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 use Python2, repeat the steps above and just change the third command to the following:

```
sudo ln -s python2 python
```

```
pi@raspberrypi:/usr/bin $ sudo rm python
pi@raspberrypi:/usr/bin $ sudo ln -s python2 python
pi@raspberrypi:/usr/bin $ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

We will only use the term “Python” without reference to Python2 or Python3. You can choose to use either. Finally, all the necessary preparations have been completed! Next, we will combine the RPi and electronic components to build a series of projects from easy to the more challenging and difficult as we focus on learning the associated knowledge of each electronic circuit.

Chapter 1 LED

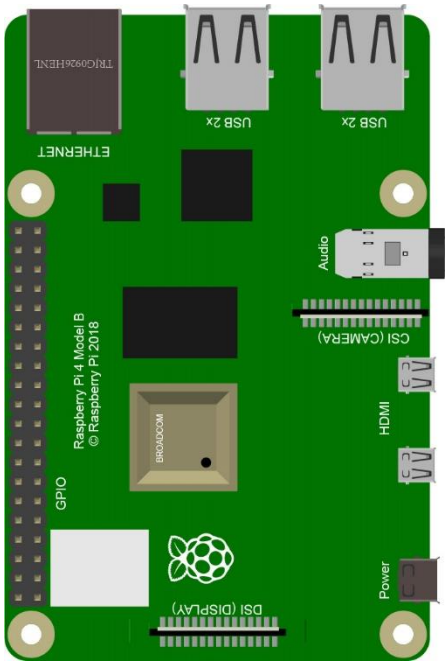
This chapter is the Start Point in the journey to build and explore RPi electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

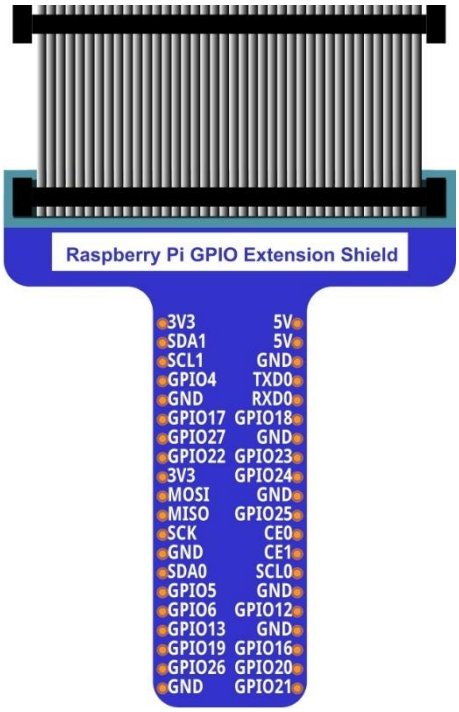
In this project, we will use RPi to control blinking a common LED.

Component List

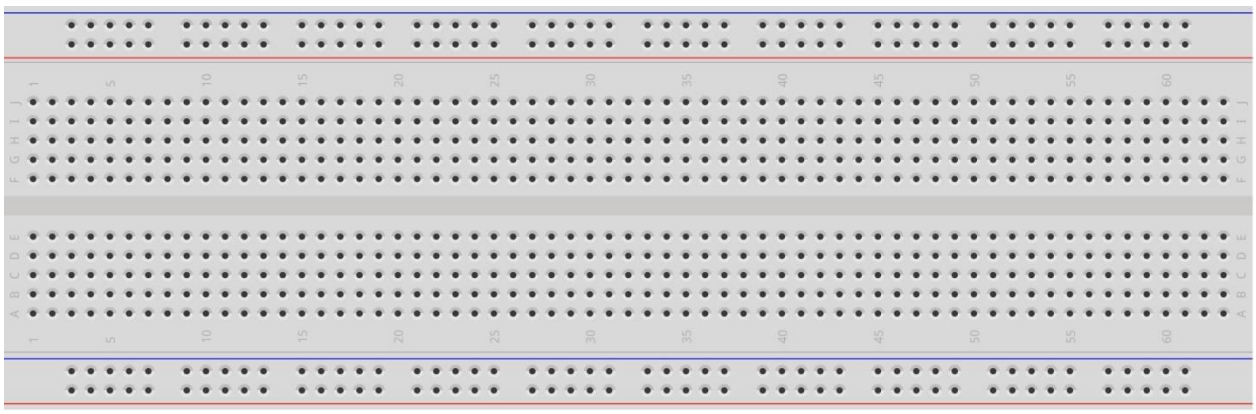
Raspberry Pi
(Recommended: Raspberry Pi 4B / 3B+ / 3B
Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)



GPIO Extension Board & Ribbon Cable



Breadboard x1



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

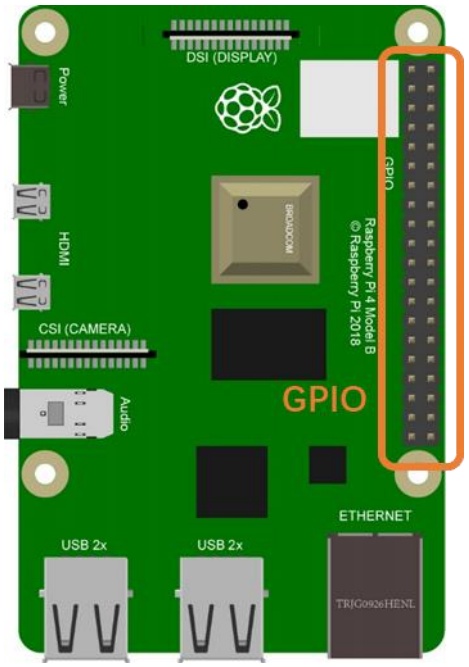
In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each project. Later, they will be reference by text only (no images as in above).

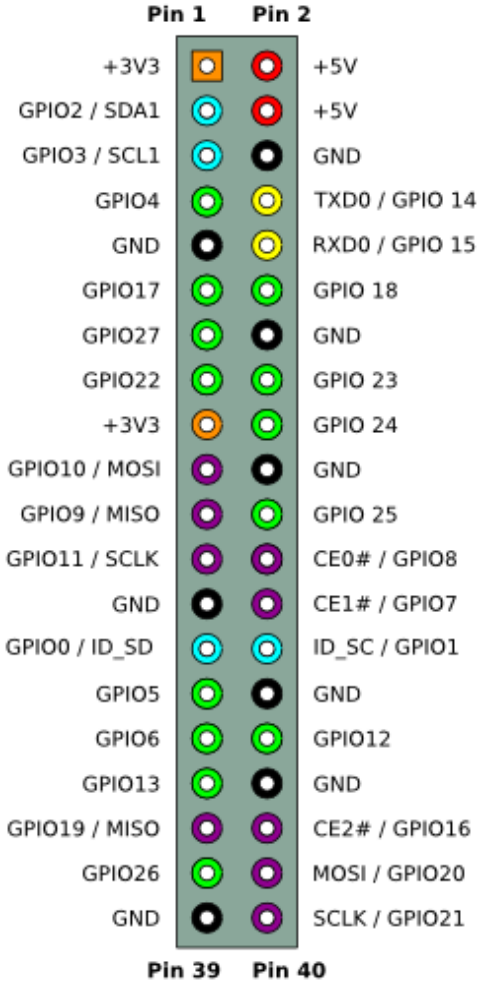
GPIO

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:

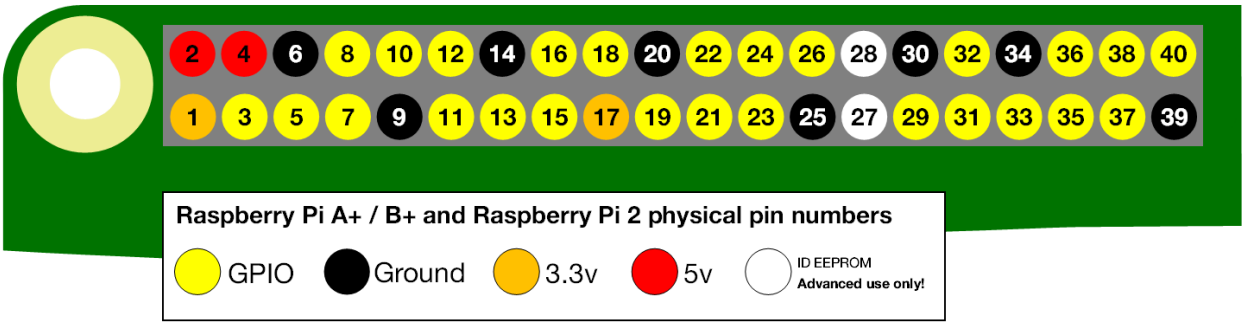




For more details about pin definition of GPIO, please refer to http://pinout.xyz/

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:



WiringPi GPIO Numbering

Different from the previous two types of GPIO serial numbers, RPi GPIO serial number of the WiringPi are numbered according to the BCM chip use in RPi.

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
—	—	3.3v	1 | 2	5v	—	—
8	R1:0/R2:2	SDA	3 | 4	5v	—	—
9	R1:1/R2:3	SCL	5 | 6	0v	—	—
7	4	GPIO7	7 | 8	TxD	14	15
—	—	0v	9 | 10	RxD	15	16
0	17	GPIO0	11 | 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 | 14	0v	—	—
3	22	GPIO3	15 | 16	GPIO4	23	4
—	—	3.3v	17 | 18	GPIO5	24	5
12	10	MOSI	19 | 20	0v	—	—
13	9	MISO	21 | 22	GPIO6	25	6
14	11	SCLK	23 | 24	CE0	8	10
—	—	0v	25 | 26	CE1	7	11
30	0	SDA.0	27 | 28	SCL.0	1	31
21	5	GPIO.21	29 | 30	0V		
22	6	GPIO.22	31 | 32	GPIO.26	12	26
23	13	GPIO.23	33 | 34	0V		
24	19	GPIO.24	35 | 36	GPIO.27	16	27
25	26	GPIO.25	37 | 38	GPIO.28	20	28
		0V	39 | 40	GPIO.29	21	29
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin

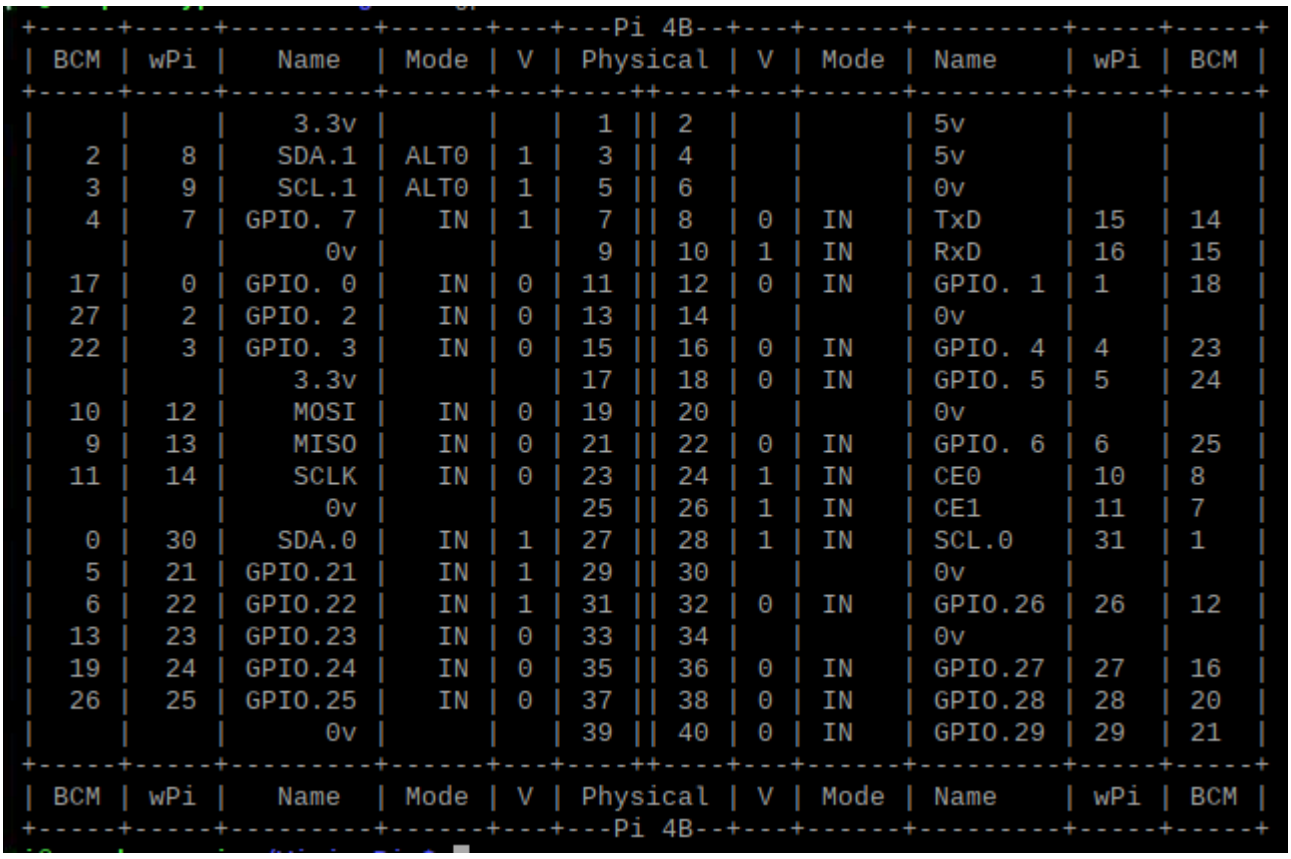
For Pi B

For A+, B+, 2B, 3B, 3B+, 4B, Zero

(For more details, please refer to https://projects.drogon.net/raspberry-pi/wiringpi/pins/)

You can also use the following command to view their correlation.

```
gpio readall
```

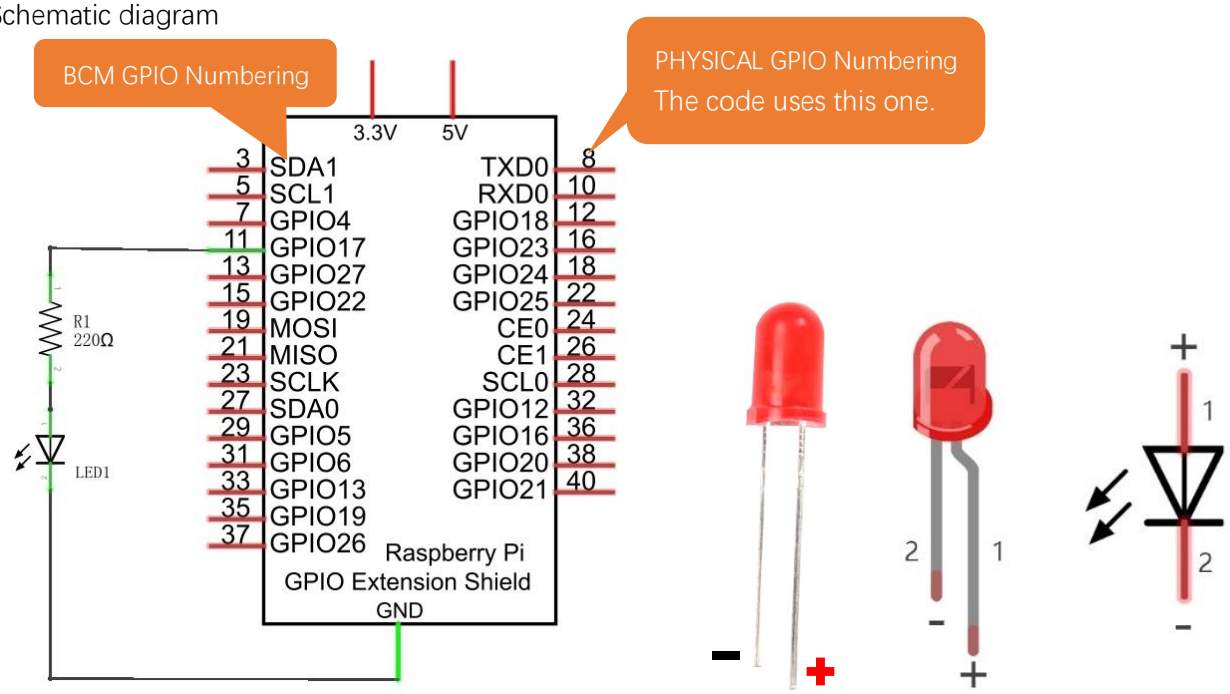


```
 +-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      |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

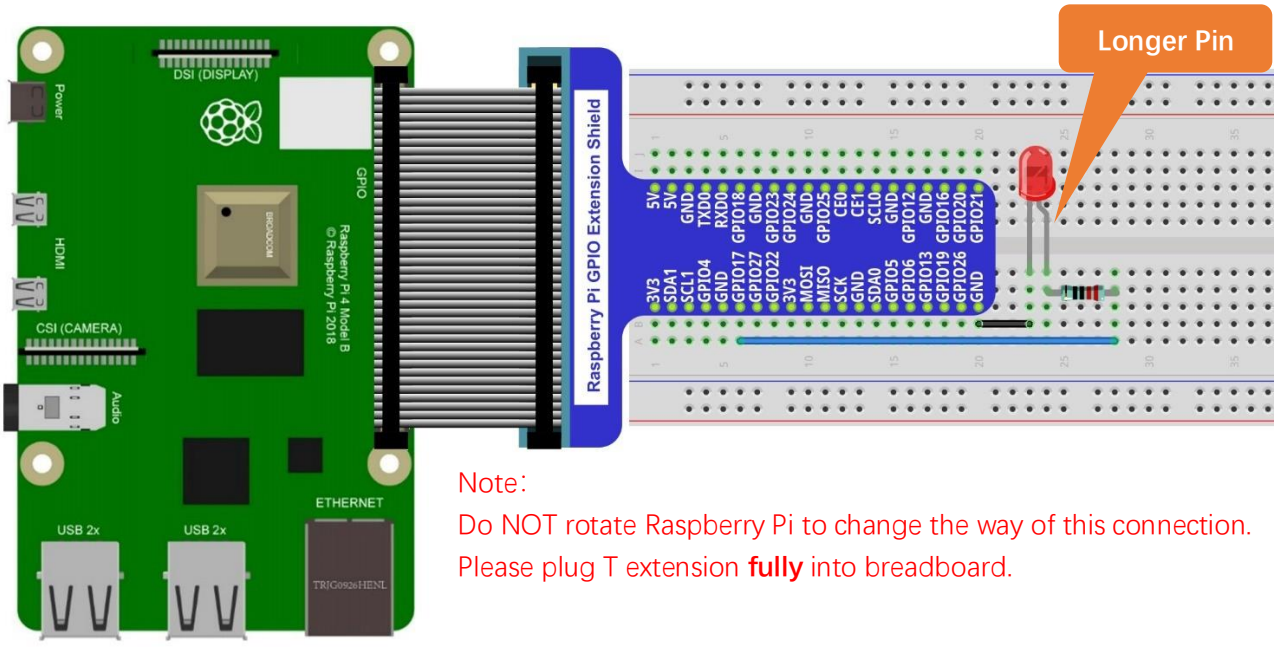
Circuit

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

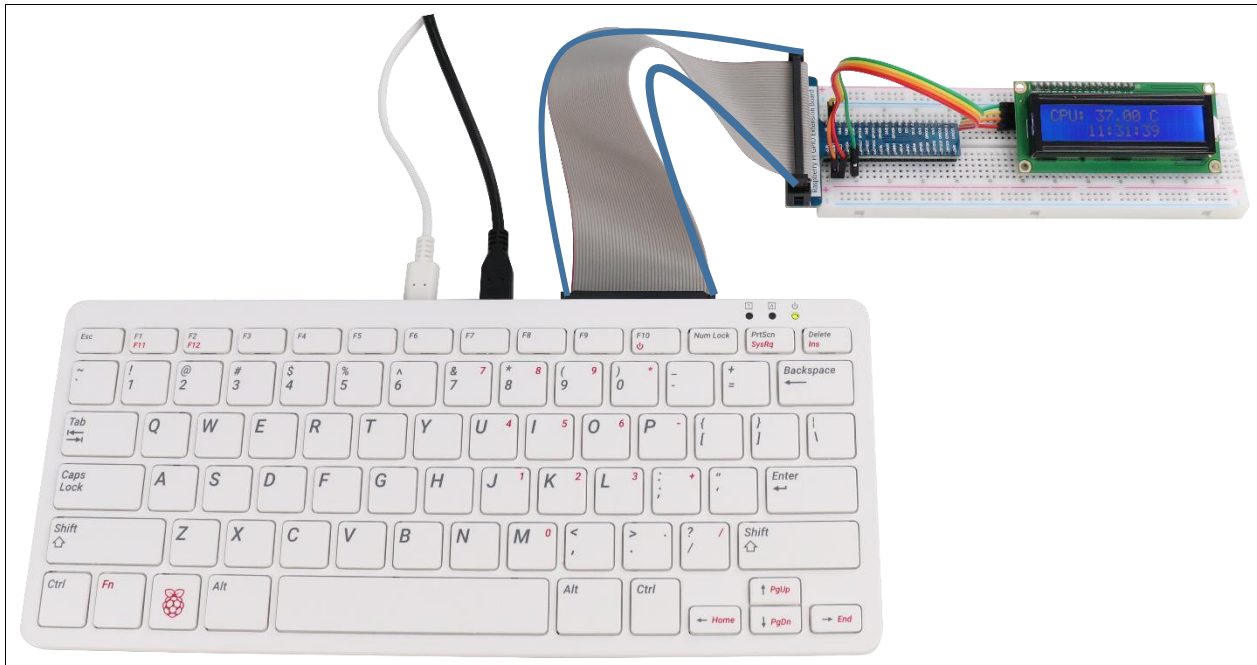


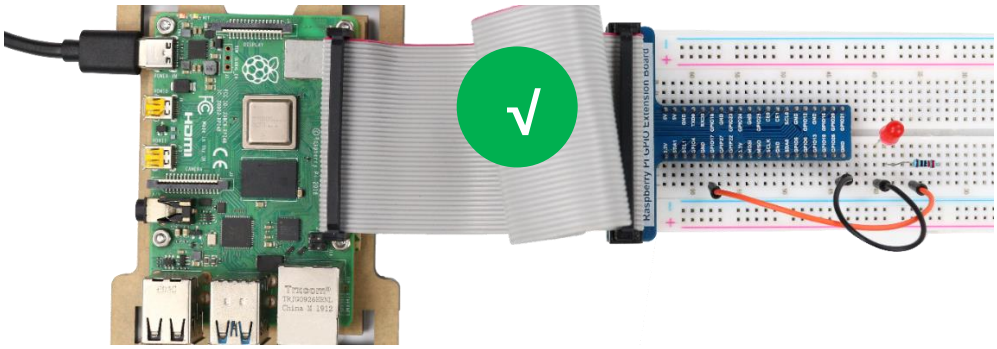
Hardware connection. **If you need any support, please contact us via: support@freenove.com**

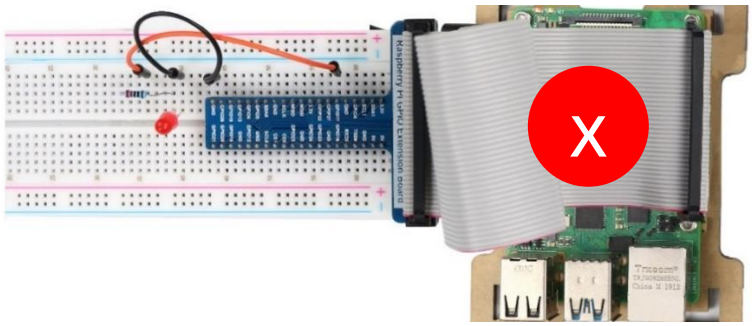


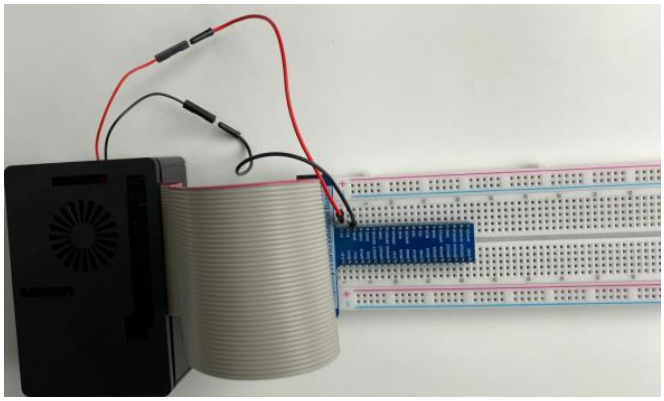
Note:
Do NOT rotate Raspberry Pi to change the way of this connection.
Please plug T extension **fully** into breadboard.

You can refer to this **youtube video**.(No video for later projects yet.) https://youtu.be/hGQtnxsr1L4
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.

How to distinguish resistors?

There are only three kind of resistors in this kit.

The one with 1 red ring is 10KΩ 

The one with 2 red rings is 220Ω 

The one with 0 red ring is 1KΩ 

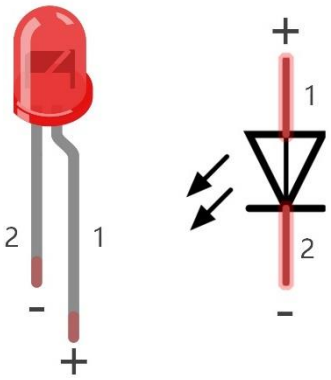
Future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

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.



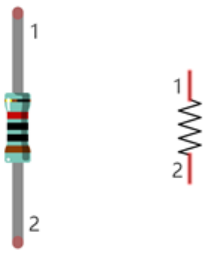
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

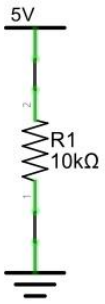
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

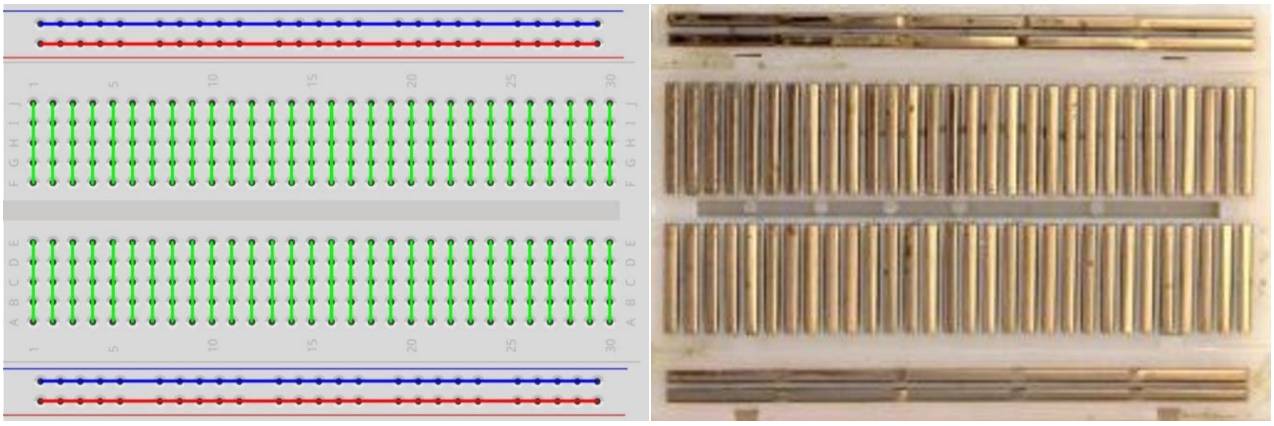


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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)

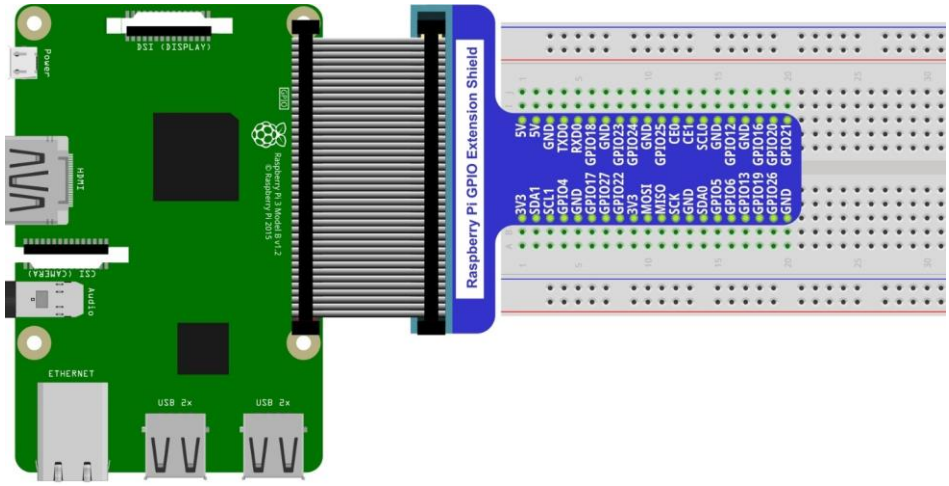
Breadboard

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.


DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
Raspberry Pi GPIO Extension Shield
5V
5V
GND
TXD0
RXD0
GPIO18
GND
GPIO23
GPIO24
GND
GPIO25
CE0
CE1
SCL0
GND
GPIO12
GND
GPIO16
GPIO20
GPIO21
3V3
SDA1
SCL1
GPIO4
GND
GPIO17
GPIO27
GPIO22
3V3
MOSI
MISO
SCK
GND
SDA0
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
GND

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 both C code and Python code to achieve the target.

C Code 1.1.1 Blink

First, enter this command into the Terminal one line at a time. Then observe the results it brings on your project, and learn about the code in detail.

If you want to execute it with editor, please refer to section Code Editor to configure.

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

It is recommended that to execute the code via command line.

1. If you did not update wiring pi, please execute following commands **one by one**.

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

2. Use cd command to enter 01.1.1_Blink directory of C code.

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

3. Use the following command to compile the code “Blink.c” and generate executable file “Blink”.

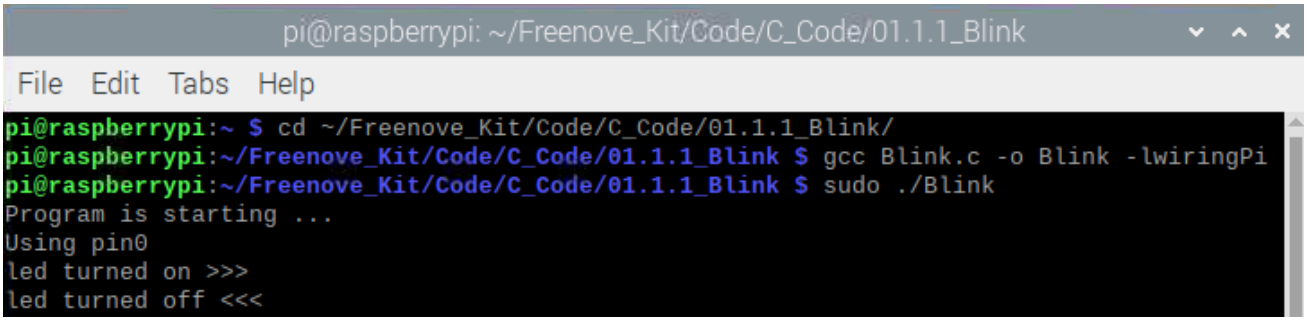
“l” of “lwiringPi” is low case of “L”.

```
gcc Blink.c -o Blink -lwiringPi
```

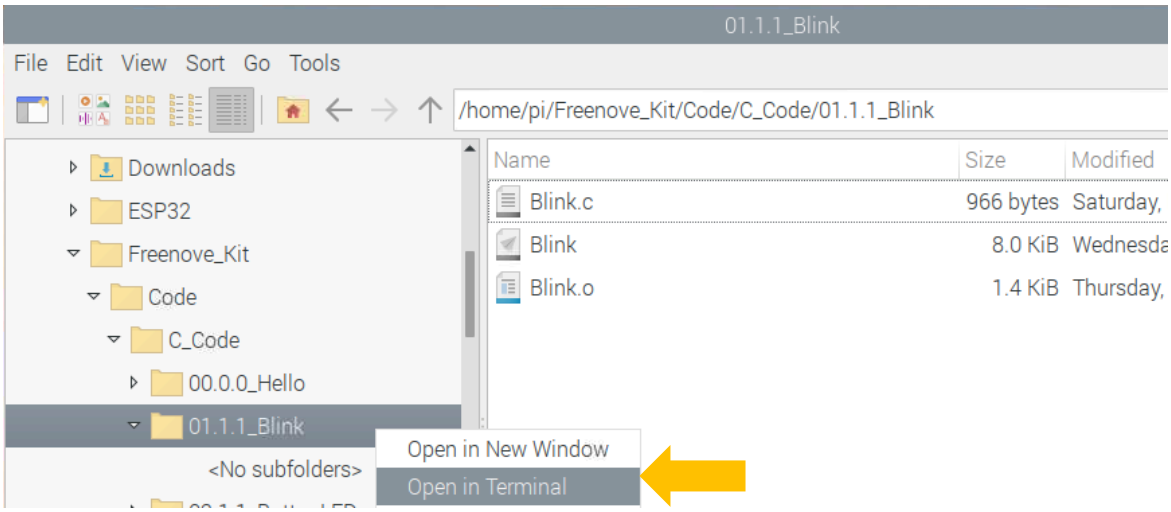
4. Then run the generated file “blink”.

```
sudo ./Blink
```

Now your LED should start blinking! CONGRATUALTIONS! You have successfully completed your first RPi circuit!



You can also use the file browser. On the left of folder tree, right-click the folder you want to enter, and click "Open in Terminal".



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

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

#define  ledPin    0  //define the led pin number

void main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();  //Initialize wiringPi.

    pinMode(ledPin, OUTPUT);//Set the pin mode
    printf("Using pin%d\n",%ledPin);    //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
        printf("led turned on >>>\n");      //Output information on terminal
        delay(1000);                        //Wait for 1 second
        digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
        printf("led turned off <<<\n");         //Output information on terminal
        delay(1000);                        //Wait for 1 second
    }
}
```

In the code above, the configuration function for GPIO is shown below as:

`void pinMode(int pin, int mode);`
This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes. This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program
`void digitalWrite (int pin, int value);`
Writes the value HIGH or LOW (1 or 0) to the given pin, which must have been previously set as an output.

For more related wiringpi functions, please refer to http://wiringpi.com/reference/

GPIO connected to ledPin in the circuit is GPIO17 and GPIO17 is defined as 0 in the wiringPi numbering. So ledPin should be defined as 0 pin. You can refer to the corresponding table in Chapter 0.

```
#define  ledPin    0 //define the led pin number
```

GPIO Numbering Relationship

WingPi	BCM(Extension)	Physical		BCM(Extension)	WingPi
3.3V	**3.3V**	1	2	5V	**5V**
8	SDA1	3	4	**5V**	**5V**
9	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 the main function main(), initialize wiringPi first.

```
    wiringPiSetup();  //Initialize wiringPi.
```

After the wiringPi is initialized successfully, you can set the ledPin to output mode and then enter the while loop, which is an endless loop (a while loop). That is, the program will always be executed in this cycle, unless it is ended because of external factors. In this loop, use digitalWrite (ledPin, HIGH) to make ledPin output high level, then LED turns ON. After a period of time delay, use digitalWrite(ledPin, LOW) to make ledPin output low

level, then LED turns OFF, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
    pinMode(ledPin, OUTPUT);//Set the pin mode
    printf("Using pin%d\n",%ledPin);    //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
        printf("led turned on >>>\n");      //Output information on terminal
        delay(1000);                        //Wait for 1 second
        digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
        printf("led turned off <<<\n");         //Output information on terminal
        delay(1000);                        //Wait for 1 second
    }
```

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

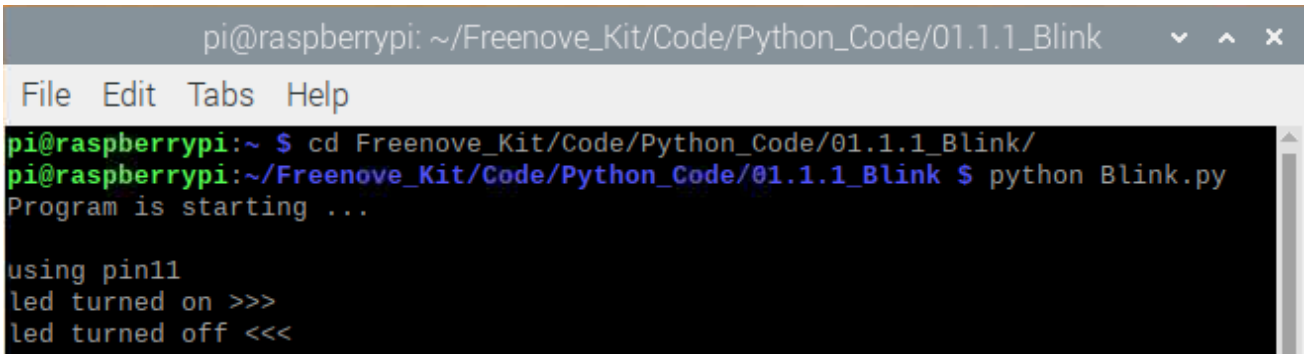
1. Use cd command to enter 01.1.1_Blink directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_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  import RPi.GPIO as GPIO
2  import time
3
4  ledPin = 11    # define ledPin
5
6  def setup():
7      GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
8      GPIO.setup(ledPin, GPIO.OUT)   # set the ledPin to OUTPUT mode
9      GPIO.output(ledPin, GPIO.LOW)  # make ledPin output LOW level
10     print ('using pin%d'%ledPin)
11
12 def loop():
13     while True:
14         GPIO.output(ledPin, GPIO.HIGH)  # make ledPin output HIGH level to turn on led
15         print ('led turned on >>>')        # print information on terminal
16         time.sleep(1)                   # Wait for 1 second
17         GPIO.output(ledPin, GPIO.LOW)   # make ledPin output LOW level to turn off led
18         print ('led turned off <<<')
19         time.sleep(1)                   # Wait for 1 second
20
21 def destroy():
22     GPIO.cleanup()                     # Release all GPIO
23
24 if __name__ == '__main__':    # Program entrance
```

```
25      print ('Program is starting ... \n')
26      setup()
27      try:
28          loop()
29      except KeyboardInterrupt:   # Press ctrl-c to end the program.
30          destroy()
```

About RPi.GPIO:

RPi.GPIO
This is a Python module to control the GPIO on a Raspberry Pi. It includes basic output function and input function of GPIO, and functions used to generate PWM.
GPIO.setmode(mode)
Sets the mode for pin serial number of GPIO. mode=GPIO.BOARD, which represents the GPIO pin serial number based on physical location of RPi. mode=GPIO.BCM, which represents the pin serial number based on CPU of BCM chip.
GPIO.setup(pin,mode)
Sets pin to input mode or output mode, "pin" for the GPIO pin, "mode" for INPUT or OUTPUT.
GPIO.output(pin,mode)
Sets pin to output mode, "pin" for the GPIO pin, "mode" for HIGH (high level) or LOW (low level).

For more functions related to RPi.GPIO, please refer to:
https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/
"import time" time is a module of python.
https://docs.python.org/2/library/time.html?highlight=time%20time#module-time

In subfunction setup(), GPIO.setmode (GPIO.BOARD) is used to set the serial number for GPIO based on physical location of the pin. GPIO17 uses pin 11 of the board, so define ledPin as 11 and set ledPin to output mode (output low level).

```
ledPin = 11    # define ledPin

def setup():
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)    # set the ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW)   # make ledPin output LOW level
    print ('using pin%d'%ledPin)
```

GPIO Numbering Relationship

WingPi	BCM(Extension)	Physical		BCM(Extension)	WingPi
3.3V	3.3V	1	2	5V	5V
8	SDA1	3	4	5V	5V
9	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 ledPin output high level, then the LED turns ON. After a period of time delay, set ledPin 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:
        GPIO.output(ledPin, GPIO.HIGH)  # make ledPin output HIGH level to turn on led
        print ('led turned on >>>')        # print information on terminal
        time.sleep(1)                   # Wait for 1 second
        GPIO.output(ledPin, GPIO.LOW)   # make ledPin output LOW level to turn off led
        print ('led turned off <<<')
        time.sleep(1)                   # Wait for 1 second
```

Finally, when the program is terminated, subfunction (a function within the file) will be executed, the LED will be turned off and then the IO port will be released. If you close the program Terminal directly, the program will also be terminated but the finish() function will not be executed. Therefore, the GPIO resources will not be released which may cause a warning message to appear the next time you use GPIO. Therefore, do not get into the habit of closing Terminal directly.

```
def finish():
    GPIO.cleanup()                  # Release all GPIO
```

Other Code Editors (Optional)

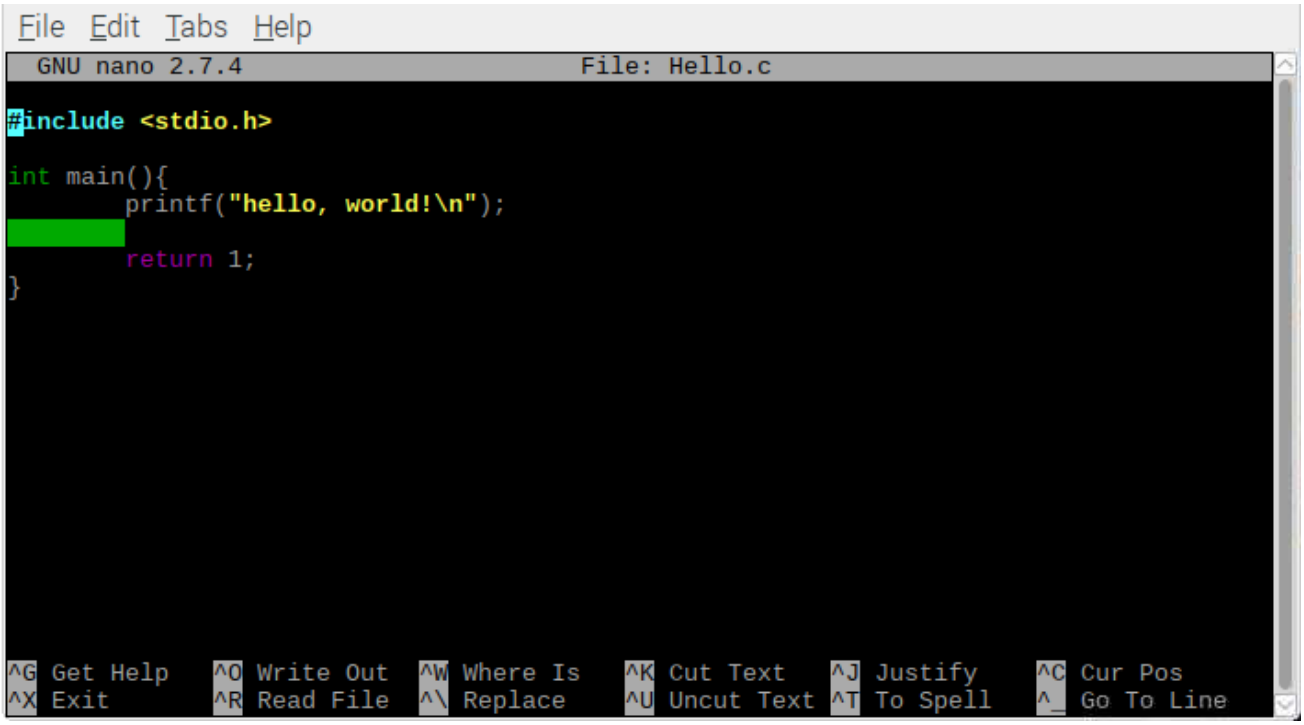
If you want to use other editor to edit and execute the code, you can learn them in this section.

nano

Use the nano editor to open the file "Hello.c", then press " Ctrl+X " to exit.

```
nano Hello.c
```

As is shown below:



Use the following command to compile the code to generate the executable file “Hello”.

```
gcc Hello.c  -o Hello
```

Use the following command to run the executable file “Hello”.

```
sudo ./Hello
```

After the execution, "Hello, World!" is printed out in terminal.

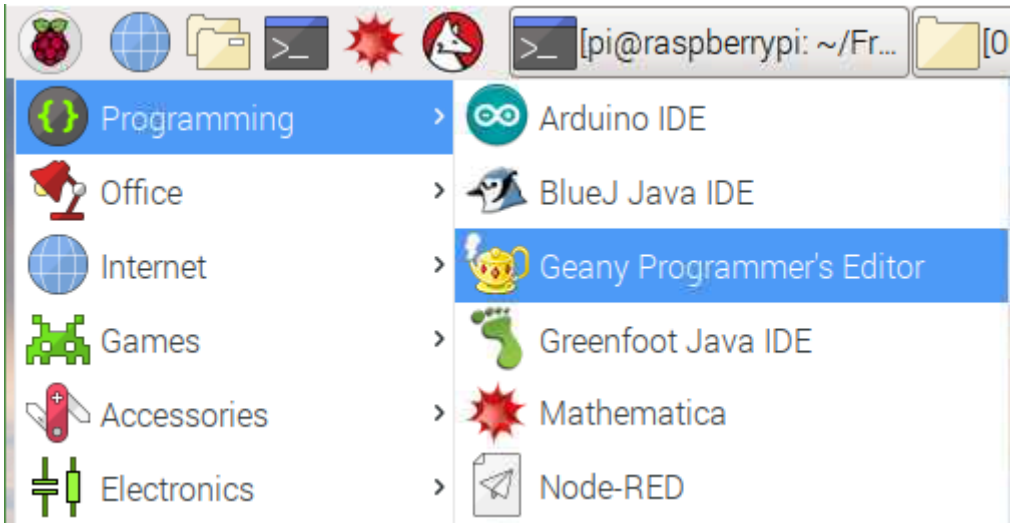
```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ gcc Hello.c -o Hello
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ sudo ./Hello
hello, world!
```

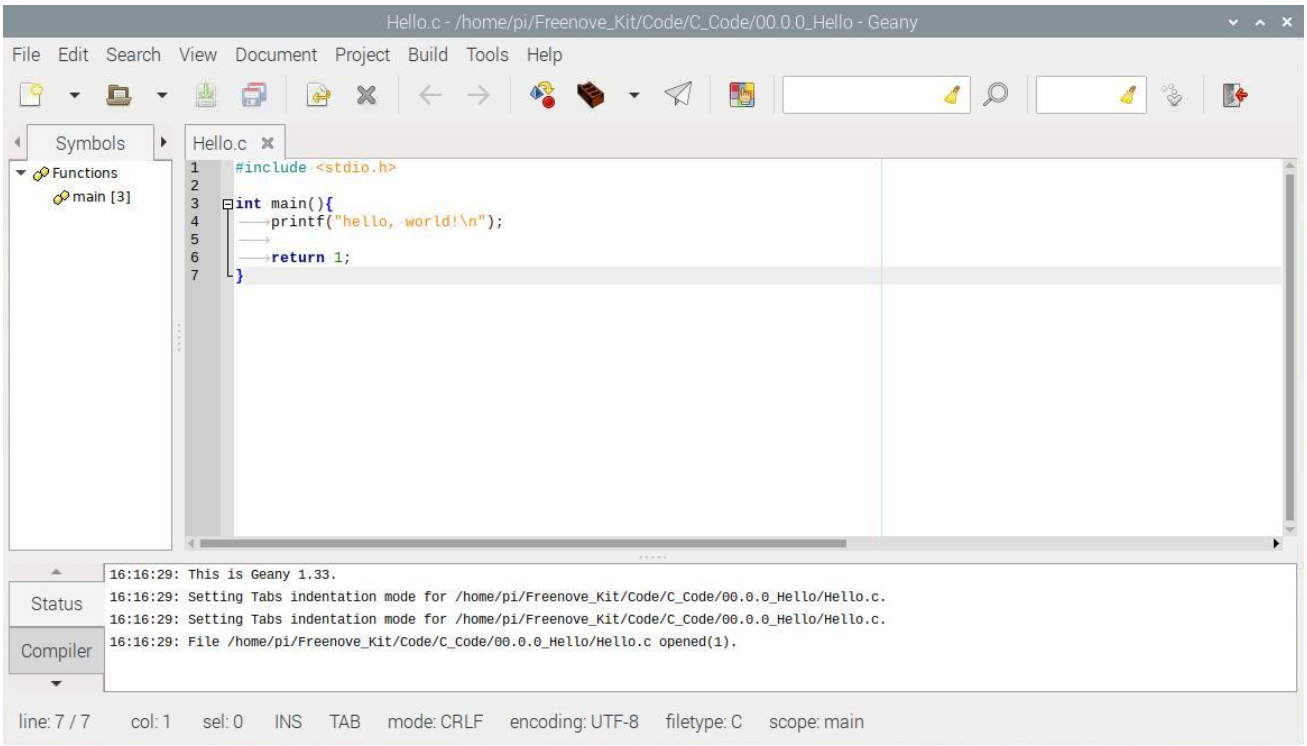
geany

Next, learn to use the Geany editor. Use the following command to open the Geany in the sample file "Hello.c" file directory path.

```
geany Hello.c
```

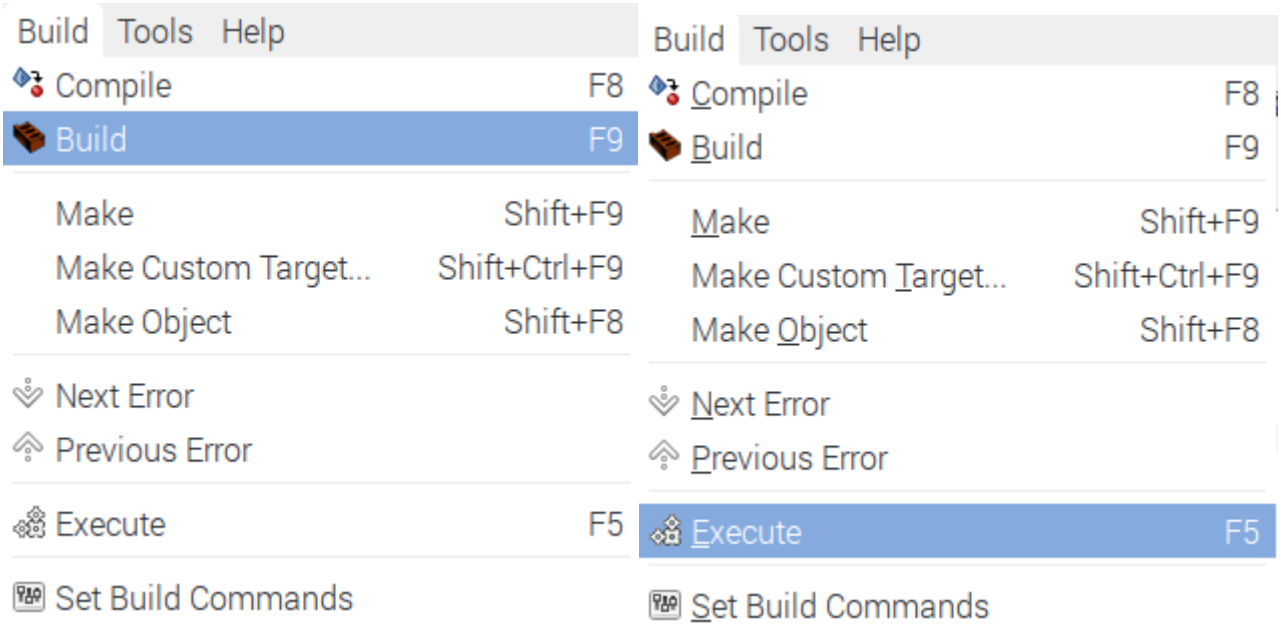
Or find and open Geany directly in the desktop main menu, and then click **File→Open** to open the "Hello.c", Or drag "Hello.c" to Geany directly.



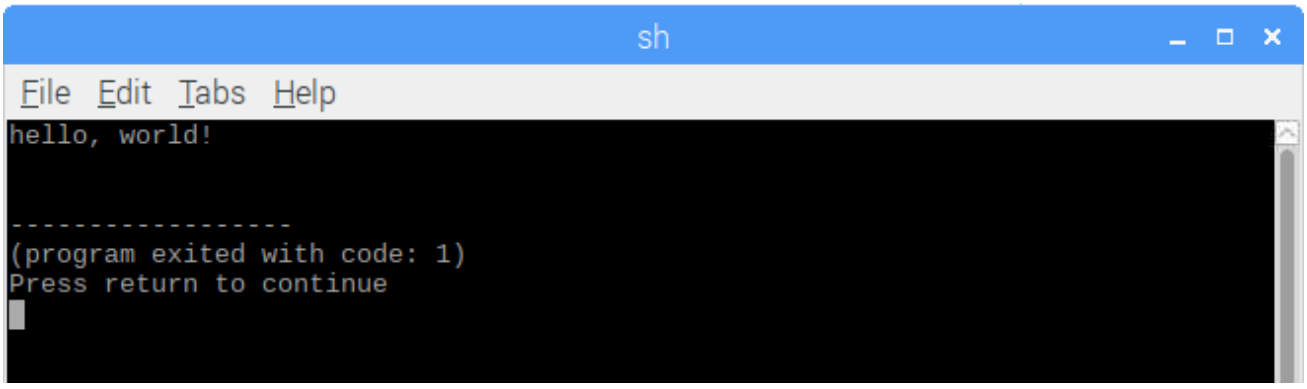


If you want to create a new code, click File→New→File→Save as (name.c or name.py). Then write the code.

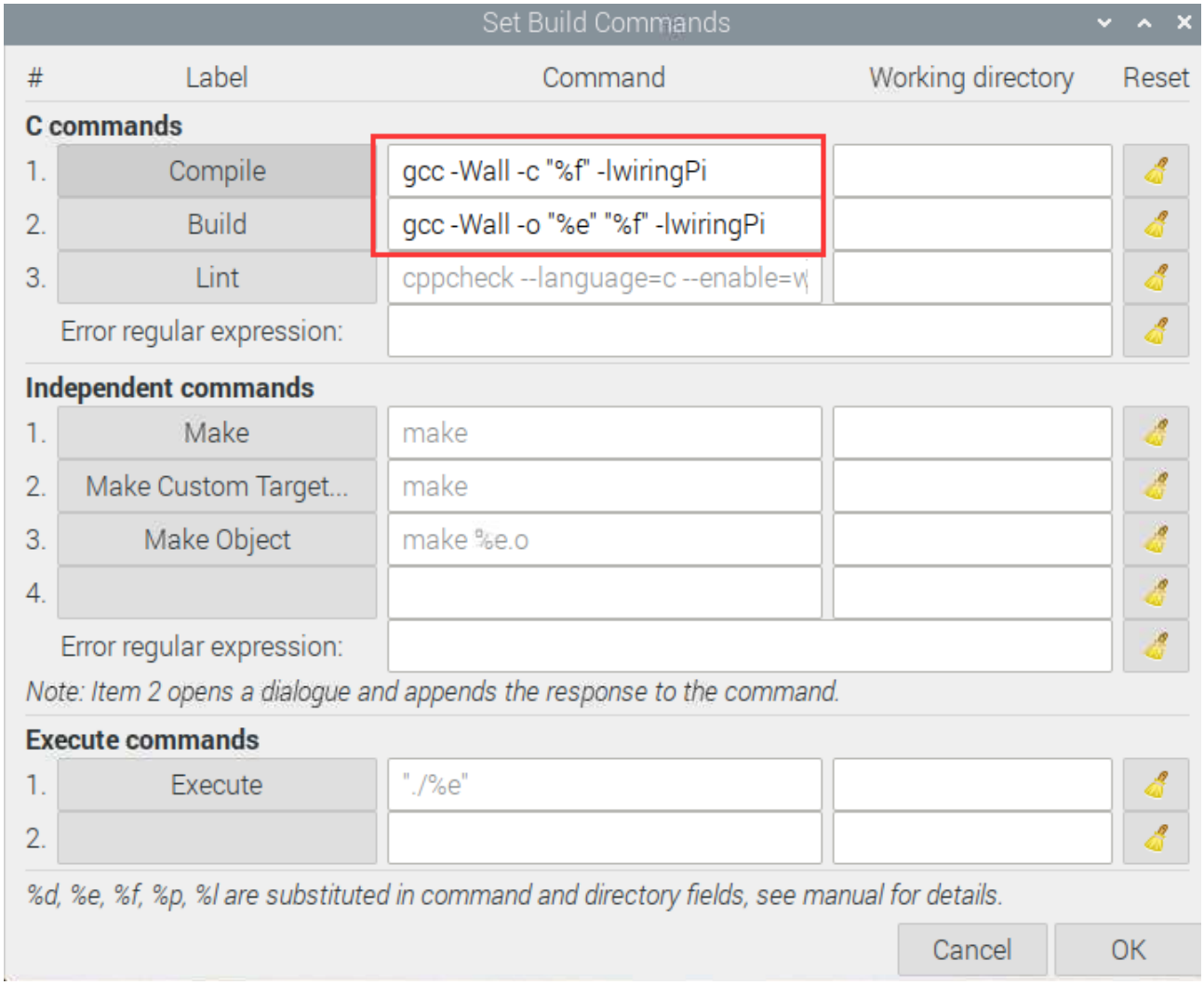
Generate an executable file by clicking menu bar Build->Build, then execute the generated file by clicking menu bar Build->Execute.



After the execution, a new terminal window will output the characters “Hello, World!”, as shown below:



You can click Build->Set Build Commands to set compiler commands. In later projects, we will use various compiler command options. If you choose to use Geany, you will need change the compiler command here. As is shown below:



Here we have identified three code editors: vi, nano and Geany. There are also many other good code editors available to you, and you can choose whichever you prefer to use.

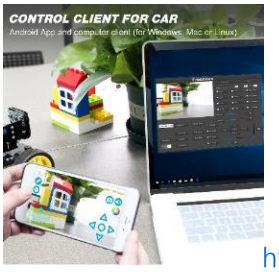
In later projects, we will only use terminal to execute the project code. This way will not modify the code by mistake.

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.

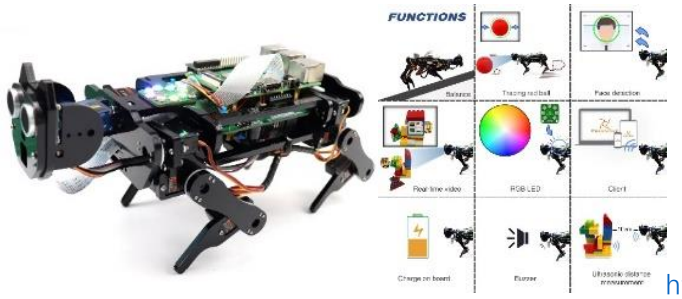
https://www.amazon.com/freenove

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

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

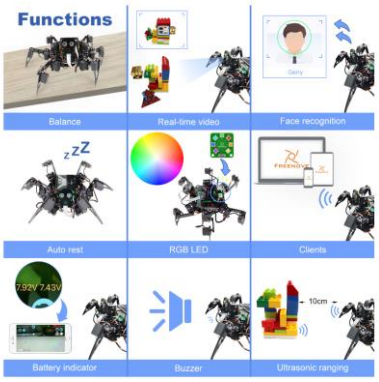
FNK0050 **Freenove Robot Dog Kit for Raspberry Pi**



https://www.youtube.com/watch?v=7BmlZ8_R9d4

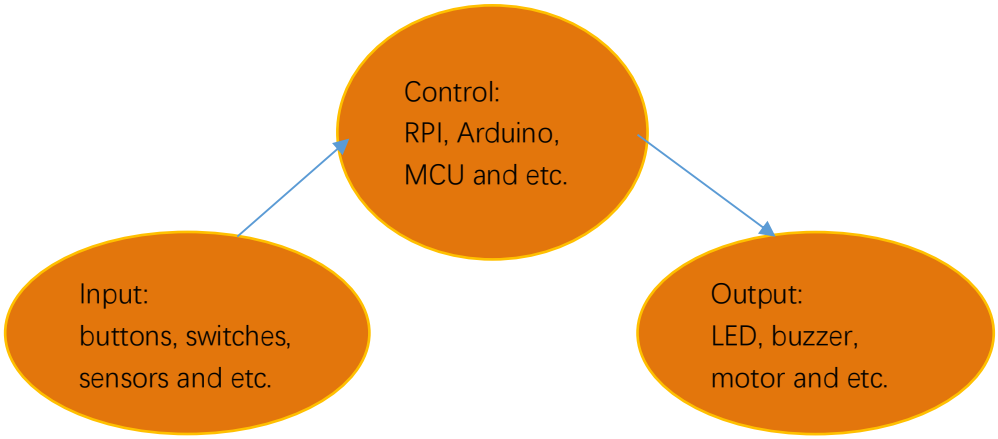
FNK0052 **Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi**

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.



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

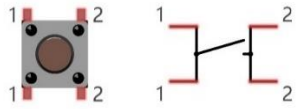
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push Button Switch x1 
Jumper Wire 				

Please Note: In the code “button” represents switch action.

Component knowledge

Push Button Switch

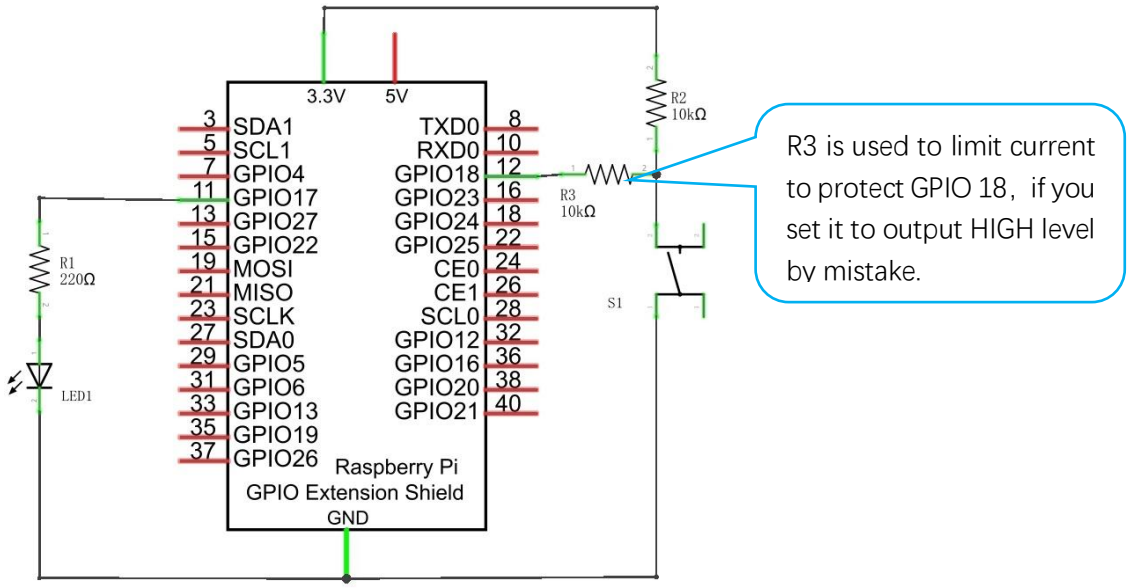
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



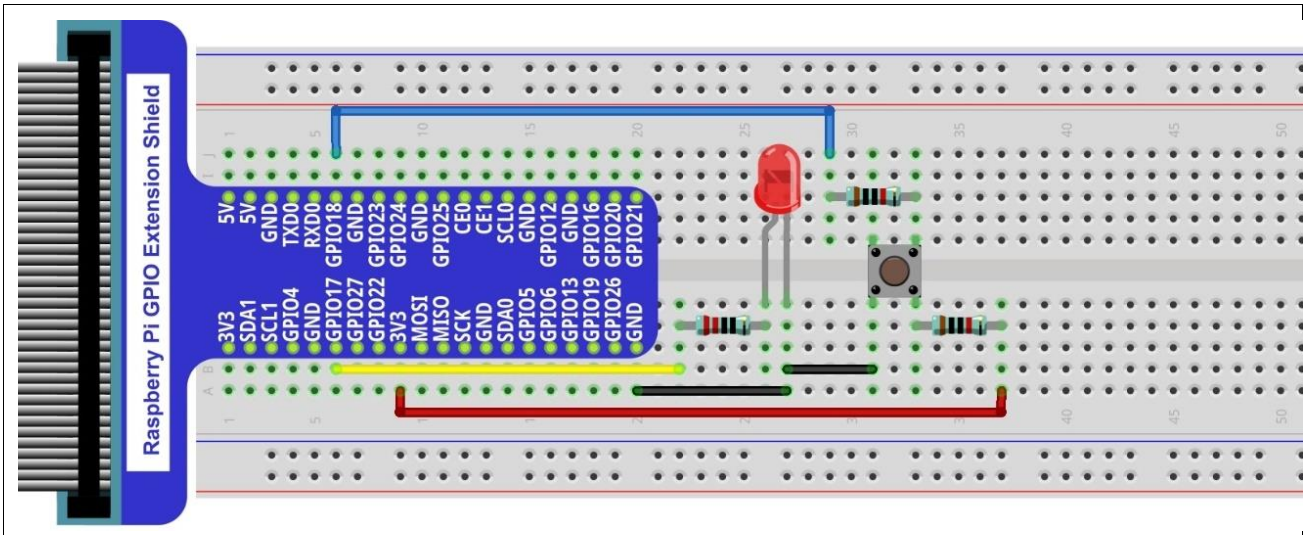
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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.

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.

C Code 2.1.1 ButtonLED

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 02.1.1_ButtonLED directory of C code.

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

2. Use the following command to compile the code "ButtonLED.c" and generate executable file "ButtonLED"

```
gcc ButtonLED.c -o ButtonLED -lwiringPi
```

3. Then run the generated file "ButtonLED".

```
sudo ./ButtonLED
```

Later, the terminal window continues to print out the characters "led off…". Press the button, then LED is turned on and then terminal window prints out the "led on…". Release the button, then LED is turned off and then terminal window prints out the "led off…". You can press "Ctrl+C" to terminate the program.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define ledPin    0   //define the ledPin
5  #define buttonPin 1       //define the buttonPin
6
7  void  main(void)
8  {
9      printf("Program is starting ... \n");
10
```

```
11      wiringPiSetup();  //Initialize wiringPi.
12
13      pinMode(ledPin, OUTPUT); //Set ledPin to output
14      pinMode(buttonPin, INPUT);//Set buttonPin to input
15
16      pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
17      while(1){
18          if(digitalRead(buttonPin) == LOW){ //button is pressed
19              digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
20              printf("Button is pressed, led turned on >>>\n");        //Output information on
21  terminal
22          }
23          else {                              //button is released
24              digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
25              printf("Button is released, led turned off <<<\n");      //Output information on
26  terminal
27          }
28      }
29  }
```

In the circuit connection, LED and Button are connected with GPIO17 and GPIO18 respectively, which correspond to 0 and 1 respectively in wiringPI. So define ledPin and buttonPin as 0 and 1 respectively.

```
#define ledPin    0     //define the ledPin
#define buttonPin 1     //define the buttonPin
```

In the while loop of main function, use digitalRead(buttonPin) to determine the state of Button. When the button is pressed, the function returns low level, the result of "if" is true, and then turn on LED. Or, turn off LED.

```
if(digitalRead(buttonPin) == LOW){ //button has pressed down
            digitalWrite(ledPin, HIGH);   //led on
            printf("led on...\n");
        }
        else {              //button has released
            digitalWrite(ledPin, LOW);   //led off
            printf("...led off\n");
        }
```

Reference:

int digitalRead (int pin);

This function returns the value read at the given pin. It will be "**HIGH**" or "**LOW**"(1 or 0) depending on the logic level at the pin.

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_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  import RPi.GPIO as GPIO
2
3  ledPin = 11    # define ledPin
4  buttonPin = 12    # define buttonPin
5
6  def setup():
7
8      GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
9      GPIO.setup(ledPin, GPIO.OUT)   # set ledPin to OUTPUT mode
10     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
11 INPUT mode
12
13 def loop():
14     while True:
15         if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
16             GPIO.output(ledPin,GPIO.HIGH)    # turn on led
17             print ('led turned on >>>')      # print information on terminal
18         else : # if button is relessed
19             GPIO.output(ledPin,GPIO.LOW) # turn off led
20             print ('led turned off <<<')
21
22 def destroy():
23     GPIO.cleanup()                     # Release GPIO resource
24
25 if __name__ == '__main__':     # Program entrance
26     print ('Program is starting...')
27     setup()
28     try:
29         loop()
30     except KeyboardInterrupt:  # Press ctrl-c to end the program.
31         destroy()
```

In subfunction setup (), GPIO.setmode (GPIO.BOARD) is used to set the serial number of the GPIO, which is based on physical location of the pin. Therefore, GPIO17 and GPIO18 correspond to pin11 and pin12 respectively in the circuit. Then set ledPin to output mode, buttonPin to input mode with a pull resistor.

```
ledPin = 11    # define ledPin
buttonPin = 12    # define buttonPin

def setup():

    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)    # set ledPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
INPUT mode
```

The loop continues endlessly to judge whether the key is pressed. When the button is pressed, the GPIO.input(buttonPin) will return low level, then the result of "if" is true, ledPin outputs high level, LED is turned on. Otherwise, LED will be turned off.

```
def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
            GPIO.output(ledPin,GPIO.HIGH)    # turn on led
            print ('led turned on >>>')      # print information on terminal
        else : # if button is released
            GPIO.output(ledPin,GPIO.LOW) # turn off led
            print ('led turned off <<<')
```

Execute the function destroy (), close the program and release the occupied GPIO pins.

About function GPIO.input ():

GPIO.input()

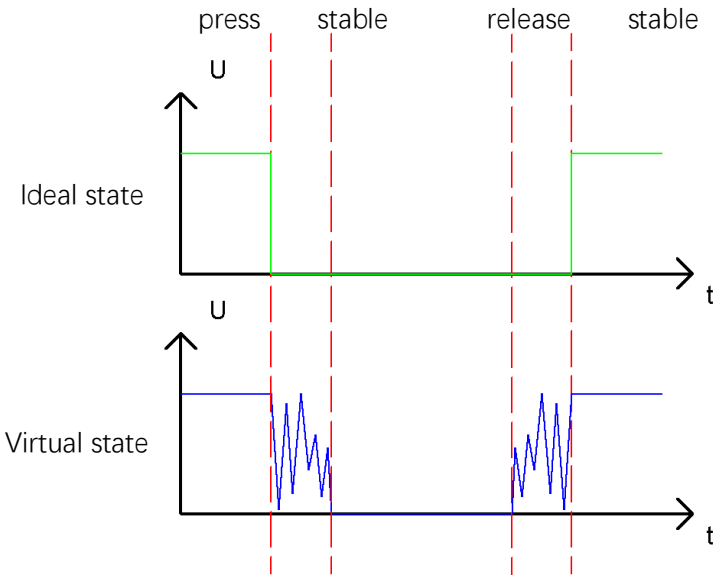
This function returns the value read at the given pin. It will be "**HIGH**" or "**LOW**"(1 or 0) depending on the logic level at the pin.

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.

C 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 C code.

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

2. Use the following command to compile "Tablelamp.c" and generate executable file "Tablelamp".

```
gcc Tablelamp.c -o Tablelamp -lwiringPi
```

3. Tablelamp: Then run the generated file "Tablelamp".

```
sudo ./Tablelamp
```

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

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

#define ledPin    0   //define the ledPin
#define buttonPin 1       //define the buttonPin
int ledState=LOW;     //store the State of led
int buttonState=HIGH; //store the State of button
int lastbuttonState=HIGH;//store the lastState of button
long lastChangeTime;  //store the change time of button state
long captureTime=50;  //set the stable time for button state
int reading;
int main(void)
{
    printf("Program is starting...\n");

    wiringPiSetup(); //Initialize wiringPi.

    pinMode(ledPin, OUTPUT);   //Set ledPin to output
    pinMode(buttonPin, INPUT); //Set buttonPin to input

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
    while(1){
        reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){  //if the button state has changed, record the time
point
            lastChangeTime = millis();
        }
```

```
28          //if changing-state of the button last beyond the time we set, we consider that
29          //the current button state is an effective change rather than a buffeting
30          if(millis() - lastChangeTime > captureTime){
31              //if button state is changed, update the data.
32              if(reading != buttonState){
33                  buttonState = reading;
34                  //if the state is low, it means the action is pressing
35                  if(buttonState == LOW){
36                      printf("Button is pressed!\n");
37                      ledState = !ledState; //Reverse the LED state
38                      if(ledState){
39                          printf("turn on LED ...\n");
40                      }
41                      else {
42                          printf("turn off LED ...\n");
43                      }
44                  }
45                  //if the state is high, it means the action is releasing
46                  else {
47                      printf("Button is released!\n");
48                  }
49              }
50          }
51          digitalWrite(ledPin,ledState);
52          lastbuttonState = reading;
53      }
54
55      return 0;
56  }
```

This code focuses on eliminating the buffeting (bounce) of the button switch. We define several variables to define the state of LED and button switch. Then read the button switch state constantly in while () to determine whether the state has changed. If it has, then this time point is recorded.

```
reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){
            lastChangeTime = millis();
        }
```

millis()

This returns a number representing the number of milliseconds since your program called one of the wiringPiSetup functions. It returns to an unsigned 32-bit number value after 49 days because it “wraps” around and restarts to value 0.

Then according to the recorded time point, evaluate the duration of the button switch state change. If the duration exceeds captureTime (buffeting time) we have set, it indicates that the state of the button switch has changed. During that time, the while () is still detecting the state of the button switch, so if there is a change, the time point of change will be updated. Then the duration will be evaluated again until the duration is determined to be a stable state because it exceeds the time value we set.

```
        if(millis() - lastChangeTime > captureTime){
            //if button state is changed,update the data.
            if(reading != buttonState){
                buttonState = reading;
```

Finally, we need to judge the state of Button Switch. If it is low level, the changing state indicates that the button Switch has been pressed, if the state is high level, then the button has been released. Here, we change the status of the LED variable, and then update the state of the LED.

```
                if(buttonState == LOW){
                    printf("Button is pressed!\n");
                    ledState = !ledState; //Reverse the LED state
                    if(ledState){
                        printf("turn on LED ...\n");
                    }
                    else {
                        printf("turn off LED ...\n");
                    }
                }
                //if the state is high, it means the action is releasing
                else {
                    printf("Button is released!\n");
                }
```

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_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.

```
1  import RPi.GPIO as GPIO
2
3  ledPin = 11       # define ledPin
4  buttonPin = 12    # define buttonPin
5  ledState = False
6
7  def setup():
8      GPIO.setmode(GPIO.BOARD)          # use PHYSICAL GPIO Numbering
9      GPIO.setup(ledPin, GPIO.OUT)      # set ledPin to OUTPUT mode
10     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # set buttonPin to PULL UP
11 INPUT mode
12
13 def buttonEvent(channel): # When button is pressed, this function will be executed
14     global ledState
15     print ('buttonEvent GPIO%d' %channel)
16     ledState = not ledState
17     if ledState :
18         print ('Led turned on >>>')
19     else :
20         print ('Led turned off <<<')
21     GPIO.output(ledPin,ledState)
22
23 def loop():
24     #Button detect
25     GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent,bouncetime=300)
26     while True:
27         pass
28
29 def destroy():
30     GPIO.cleanup()                     # Release GPIO resource
31
32 if __name__ == '__main__':     # Program entrance
33     print ('Program is starting...')
34     setup()
35     try:
```

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

RPi.GPIO provides us with a simple but effective function to eliminate "jitter", that is GPIO.add_event_detect(). It uses the callback function. Once it detects that the buttonPin has a specified action FALLING, it executes a specified function buttonEvent(). In the function buttonEvent, each time the ledState is reversed, the state of the LED will be updated.

```
def buttonEvent(channel): # When button is pressed, this function will be executed
    global ledState
    print ('buttonEvent GPIO%d' %channel)
    ledState = not ledState
    if ledState :
        print ('Led turned on >>>')
    else :
        print ('Led turned off <<<')
    GPIO.output(ledPin,ledState)

def loop():
    #Button detect
    GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent,bouncetime=300)
    while True:
        pass
```

Of course, you can also use the same programming idea in C code above to achieve this target.

GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)

This is an event detection function. The first parameter specifies the IO port to be detected. The second parameter specifies the action to be detected. The third parameter specifies a function name; the function will be executed when the specified action is detected. The fourth parameter is used to set the jitter time.

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

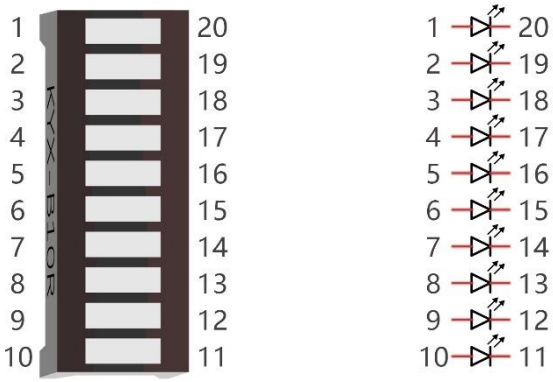
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Bar Graph LED x1 	Resistor 220Ω x10 
Jumper Wire x 1 		

Component knowledge

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

Bar Graph LED

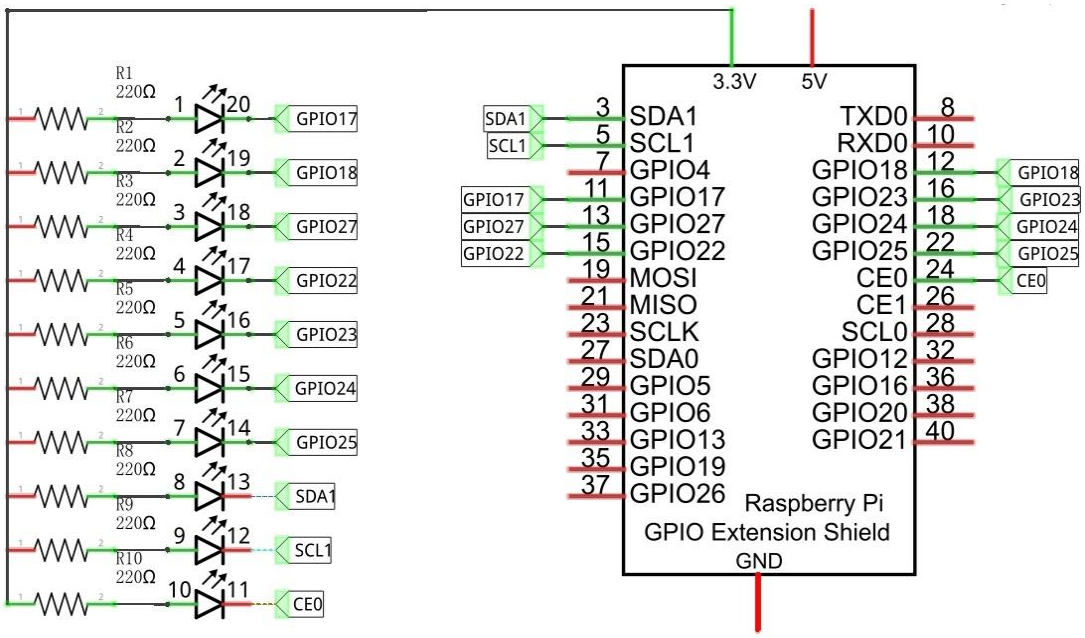
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



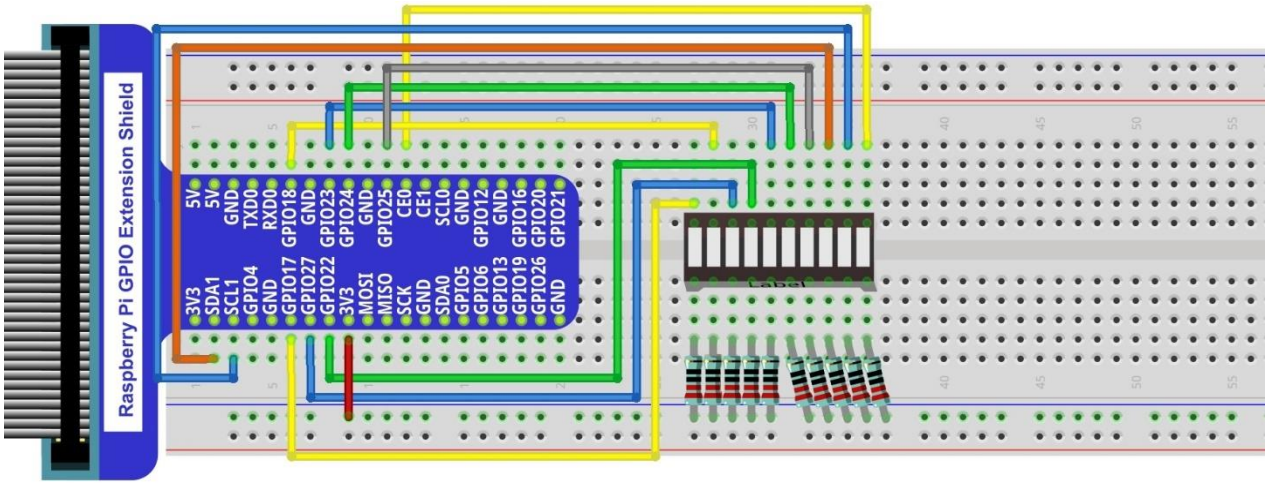
Circuit

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.

C Code 3.1.1 LightWater

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 03.1.1_LightWater directory of C code.

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

2. Use the following command to compile "LightWater.c" and generate executable file "LightWater".

```
gcc LightWater.c -o LightWater -lwiringPi
```

3. Then run the generated file "LightWater".

```
sudo ./LightWater
```

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:

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

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

void main(void)
{
    int i;
    printf("Program is starting ... \n");

    wiringPiSetup(); //Initialize wiringPi.

    for(i=0;i<ledCounts;i++){       //Set pinMode for all led pins to output
        pinMode(pins[i], OUTPUT);
    }
    while(1){
        for(i=0;i<ledCounts;i++){   // move led(on) from left to right
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
        for(i=ledCounts-1;i>-1;i--){   // move led(on) from right to left
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
    }
}
```

In the program, configure the GPIO0-GPIO9 to output mode. Then, in the endless "while" loop of main function, use two "for" loop to realize flowing water light from left to right and from right to left.

```
    while(1){
        for(i=0;i<ledCounts;i++){   // move led(on) from left to right
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
        for(i=ledCounts-1;i>-1;i--){   // move led(on) from right to left
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
    }
}
```

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_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:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]
5
6  def setup():
7      GPIO.setmode(GPIO.BOARD)        # use Physical GPIO Numbering
8      GPIO.setup(ledPins, GPIO.OUT)   # set all ledPins to OUTPUT mode
9      GPIO.output(ledPins, GPIO.HIGH) # make all ledPins output HIGH level, turn off all led
10
11 def loop():
12     while True:
13         for pin in ledPins:        # make led(on) move from left to right
14             GPIO.output(pin, GPIO.LOW)
15             time.sleep(0.1)
16             GPIO.output(pin, GPIO.HIGH)
17         for pin in ledPins[::-1]:      # make led(on) move from right to left
18             GPIO.output(pin, GPIO.LOW)
19             time.sleep(0.1)
20             GPIO.output(pin, GPIO.HIGH)
21
22 def destroy():
23     GPIO.cleanup()                     # Release all GPIO
24
25 if __name__ == '__main__':     # Program entrance
26     print ('Program is starting...')
27     setup()
28     try:
29         loop()
30     except KeyboardInterrupt:  # Press ctrl-c to end the program.
31         destroy()
```

In the program, first define 10 pins connected to LED, and set them to output mode in subfunction setup(). Then in the loop() function, use two "for" loops to realize flowing water light from right to left and from left to right. ledPins[::-1] is used to get elements of ledPins in reverse order.

```
def loop():
    while True:
        for pin in ledPins:     #make led on from left to right
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)
        for pin in ledPins[::-1]:        #make led on from right to left
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)
```

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.

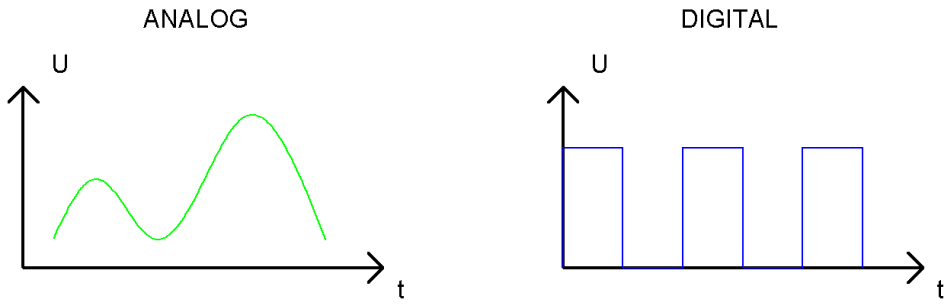
Component List

<table>
<tr><td>Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td rowspan="2">LED x1
  </td><td rowspan="2">Resistor 220Ω x1
  </td></tr>
<tr><td>Jumper Wire
  </td></tr>
</table>

Component Knowledge

Analog & Digital

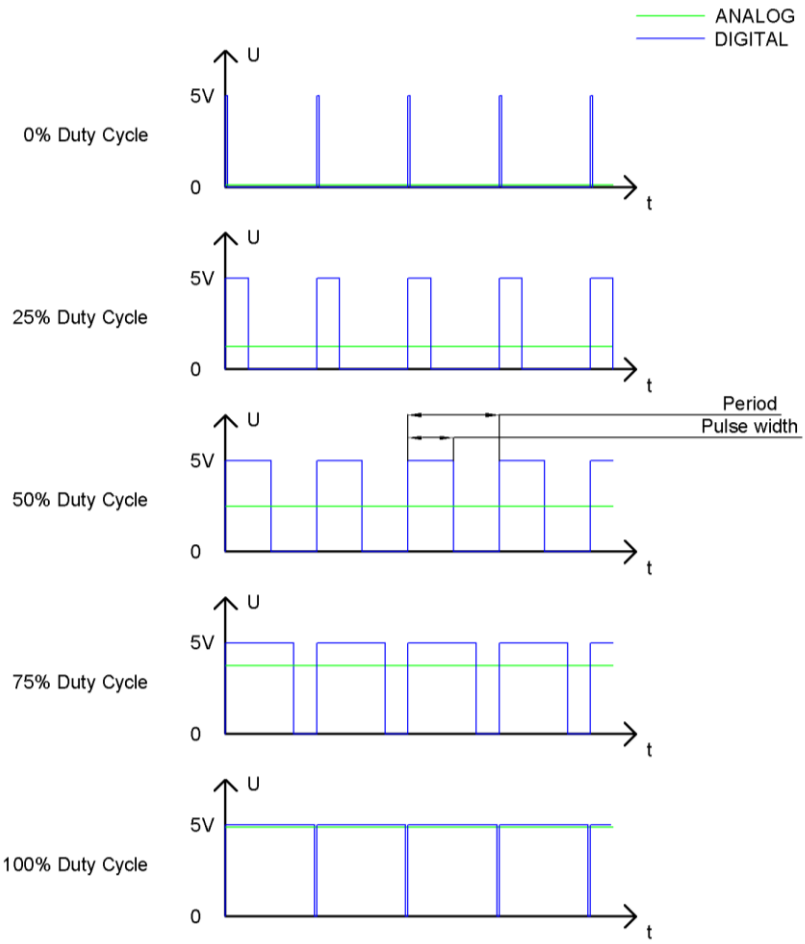
An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal **or discrete-time signal is a time series consisting of a sequence of quantities**. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout 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.



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.

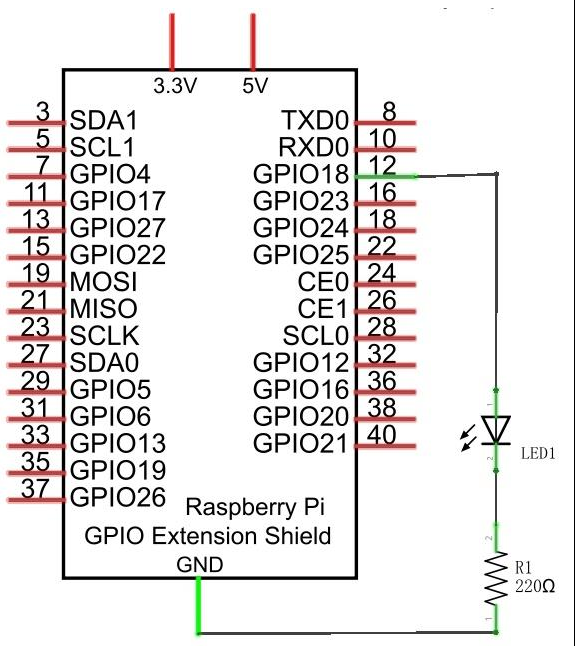
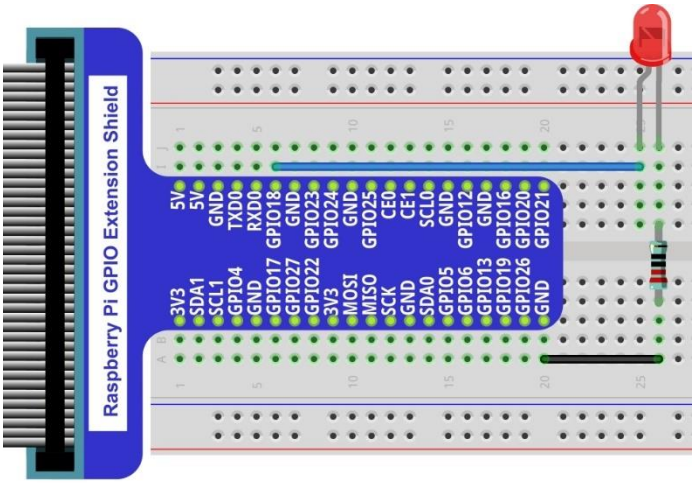
In RPi, GPIO18 pin has the ability to output to hardware via PWM with a 10-bit accuracy. This means that 100% of the pulse width can be divided into 2^{10}=1024 equal parts.

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 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.

C 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 C code.

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

2. Use following command to compile "BreathingLED.c" and generate executable file "BreathingLED".

```
gcc BreathingLED.c -o BreathingLED -lwiringPi
```

3. Then run the generated file "BreathingLED"

```
sudo ./BreathingLED
```

After the program is executed, you'll see that LED is turned from on to off and then from off to on gradually like breathing.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <softPwm.h>
4  #define ledPin    1
5  void main(void)
6  {
```

```
7       int i;
8
9       printf("Program is starting ... \n");
10
11      wiringPiSetup();  //Initialize wiringPi.
12
13      softPwmCreate(ledPin,  0, 100);//Creat SoftPWM pin
14
15      while(1){
16          for(i=0;i<100;i++){  //make the led brighter
17              softPwmWrite(ledPin, i);
18              delay(20);
19          }
20          delay(300);
21          for(i=100;i>=0;i--){  //make the led darker
22              softPwmWrite(ledPin, i);
23              delay(20);
24          }
25          delay(300);
26      }
27  }
```

First, create a software PWM pin.

```
    softPwmCreate(ledPin,  0, 100);//Creat SoftPWM pin
```

There are two “for” loops in the next endless “while” loop. The first loop outputs a power signal to the ledPin PWM from 0% to 100% and the second loop outputs a power signal to the ledPin PWM from 100% to 0%.

```
    while(1){
        for(i=0;i<100;i++){
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
        for(i=100;i>=0;i--){
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
    }
```

You can also adjust the rate of the state change of LED by changing the parameter of the delay() function in the “for” loop.

int softPwmCreate (int pin, int initialValue, int pwmRange) ;
This creates a software controlled PWM pin.
void softPwmWrite (int pin, int value) ;
This updates the PWM value on the given pin.

For more details, please refer http://wiringpi.com/reference/software-pwm-library/

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_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:

```
import RPi.GPIO as GPIO
import time

LedPin = 12       # define the LedPin

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(LedPin, GPIO.OUT)    # set LedPin to OUTPUT mode
    GPIO.output(LedPin, GPIO.LOW)   # make ledPin output LOW level to turn off LED

    p = GPIO.PWM(LedPin, 500)       # set PWM Frequence to 500Hz
    p.start(0)                      # set initial Duty Cycle to 0

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

def destroy():
    p.stop() # stop PWM
    GPIO.cleanup() # Release all GPIO

if __name__ == '__main__':     # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
```

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

The LED is connected to the IO port called GPIO18. The LedPin is defined as pin 12 and set to output mode according to the corresponding chart for pin designations. Then create a PWM instance and set the PWM frequency to 1000HZ and the initial duty cycle to 0%.

```
LedPin = 12       # define the LedPin

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(LedPin, GPIO.OUT)    # set LedPin to OUTPUT mode
    GPIO.output(LedPin, GPIO.LOW)   # make ledPin output LOW level to turn off LED

    p = GPIO.PWM(LedPin, 500)       # set PWM Frequence to 500Hz
    p.start(0)                      # set 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 ledPin PWM from 0% to 100% and the second loop outputs a power signal to the ledPin PWM from 100% to 0%.

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

The related functions of PWM are described as follows:

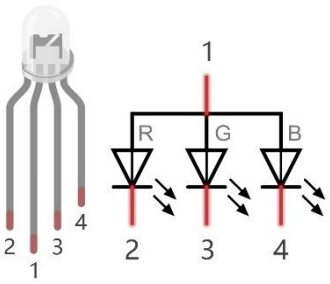
p = GPIO.PWM(channel, frequency)
To create a PWM instance:
p.start(dc)
To start PWM, where dc is the duty cycle (0.0 <= dc <= 100.0)
p.ChangeFrequency(freq)
To change the frequency, where freq is the new frequency in Hz
p.ChangeDutyCycle(dc)
To change the duty cyclewhere 0.0 <= dc <= 100.0
p.stop()
To stop PWM.

For more details regarding methods for using PWM with RPi.GPIO, please refer to:
https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/

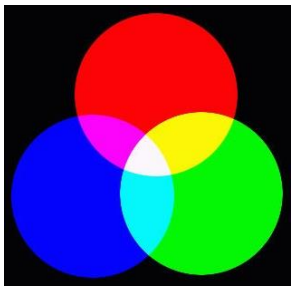
Chapter 5 RGB LED

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

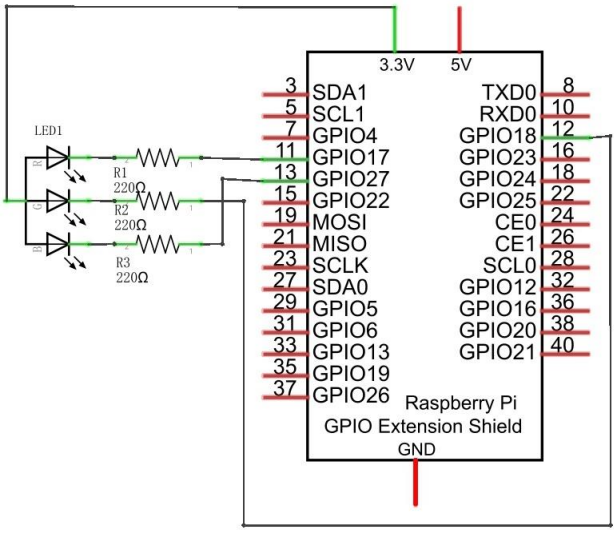
In this project, we will make a multicolored LED, which we can program the RGB LED to automatically change colors.

Component List

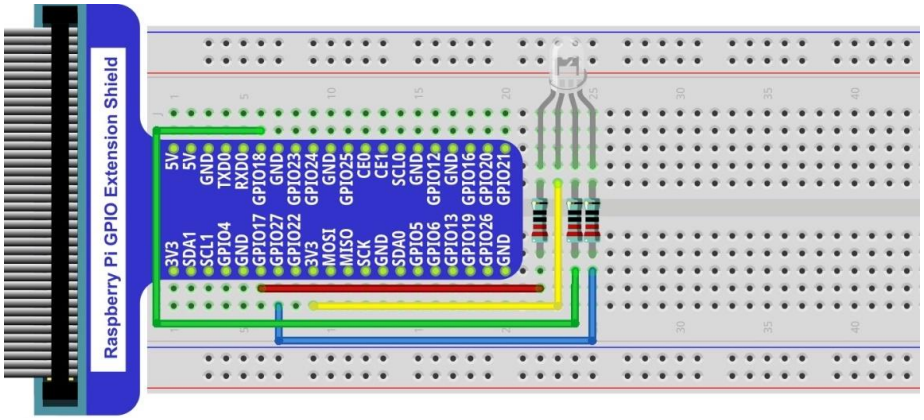
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	RGB LED x1 	Resistor 220Ω x3 
Jumper Wire 		

Circuit

Schematic diagram



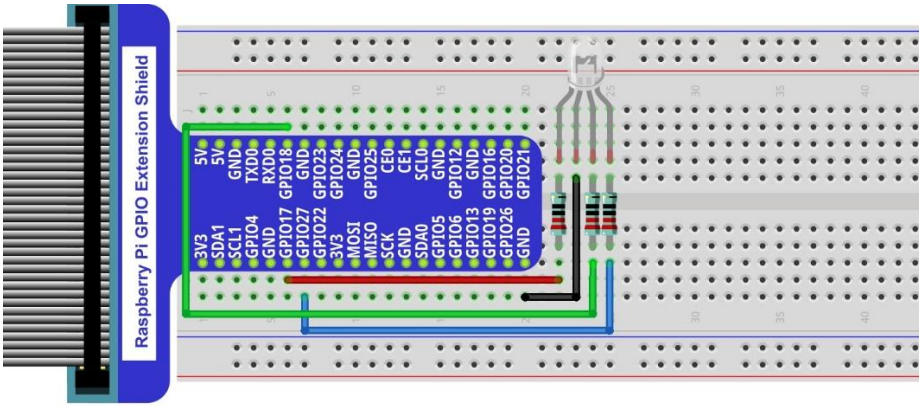
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



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 the software to make the ordinary GPIO output PWM, since this project requires 3 PWM and in RPi only one GPIO has the hardware capability to output PWM,

C Code 5.1.1 Colorful LED

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 C code.

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

2. Use following command to compile "ColorfulLED.c" and generate executable file "ColorfulLED".
   Note: in this project, the software PWM uses a multi-threading mechanism. So "-lpthread" option need to be add to the compiler.

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

3. And then run the generated file "ColorfulLED".

```
sudo ./ColorfulLED
```

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

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <softPwm.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define ledPinRed    0
7  #define ledPinGreen  1
8  #define ledPinBlue   2
9
10 void setupLedPin(void)
```

```
{
    softPwmCreate(ledPinRed,   0,  100);  //Creat SoftPWM pin for red
    softPwmCreate(ledPinGreen, 0,  100);  //Creat SoftPWM pin for green
    softPwmCreate(ledPinBlue,  0,  100);  //Creat SoftPWM pin for blue
}

void setLedColor(int r,  int g,  int b)
{
    softPwmWrite(ledPinRed,    r);  //Set the duty cycle
    softPwmWrite(ledPinGreen,  g);   //Set the duty cycle
    softPwmWrite(ledPinBlue,   b);   //Set the duty cycle
}

int main(void)
{
    int r,g,b;

    printf("Program is starting ...\n");

    wiringPiSetup();  //Initialize wiringPi.

    setupLedPin();
    while(1){
        r=random()%100;  //get a random in (0,100)
        g=random()%100;  //get a random in (0,100)
        b=random()%100;  //get a random in (0,100)
        setLedColor(r,g,b);//set random as the duty cycle value
        printf("r=%d,   g=%d,   b=%d \n",r,g,b);
        delay(1000);
    }
    return 0;
}
```

First, in subfunction of ledInit(), create the software PWM control pins used to control the R, G, B pin respectively.

```
void setupLedPin(void)
{
    softPwmCreate(ledPinRed,   0,  100);  //Creat SoftPWM pin for red
    softPwmCreate(ledPinGreen, 0,  100);  //Creat SoftPWM pin for green
    softPwmCreate(ledPinBlue,  0,  100);  //Creat SoftPWM pin for blue
}
```

Then create subfunction, and set the PWM of three pins.

```
void setLedColor(int r,  int g,  int b)
{
```

```
    softPwmWrite(ledPinRed,   r);  //Set the duty cycle
    softPwmWrite(ledPinGreen, g);   //Set the duty cycle
    softPwmWrite(ledPinBlue,  b);   //Set the duty cycle
}
```

Finally, in the "while" loop of main function, get three random numbers and specify them as the PWM duty cycle, which will be assigned to the corresponding pins. So RGB LED can switch the color randomly all the time.

```
    while(1){
        r=random()%100;  //get a random in (0,100)
        g=random()%100;  //get a random in (0,100)
        b=random()%100;  //get a random in (0,100)
        setLedColor(r,g,b);//set random as the duty cycle value
        printf("r=%d,  g=%d,  b=%d \n",r,g,b);
        delay(1000);
    }
```

The related function of PWM Software can be described as follows:

long random();

This function will return a random number.

For more details about Software PWM, please refer to: http://wiringpi.com/reference/software-pwm-library/

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_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:

```
import RPi.GPIO as GPIO
import time
import random

pins = [11, 12, 13]         # define the pins for R:11,G:12,B:13

def setup():
    global pwmRed,pwmGreen,pwmBlue
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(pins, GPIO.OUT)      # set RGBLED pins to OUTPUT mode
    GPIO.output(pins, GPIO.HIGH)    # make RGBLED pins output HIGH level
    pwmRed = GPIO.PWM(pins[0], 2000)       # set PWM Frequence to 2kHz
    pwmGreen = GPIO.PWM(pins[1], 2000)  # set PWM Frequence to 2kHz
    pwmBlue = GPIO.PWM(pins[2], 2000)     # set PWM Frequence to 2kHz
    pwmRed.start(0)       # set initial Duty Cycle to 0
    pwmGreen.start(0)
    pwmBlue.start(0)

def setColor(r_val,g_val,b_val):      # change duty cycle for three pins to r_val,g_val,b_val
    pwmRed.ChangeDutyCycle(r_val)     # change pwmRed duty cycle to r_val
    pwmGreen.ChangeDutyCycle(g_val)
    pwmBlue.ChangeDutyCycle(b_val)

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
# if you are using common anode RGBLED,it should be setColor(100-r,100-g,100-b)
        print ('r=%d, g=%d, b=%d ' %(r ,g, b))
        time.sleep(1)

def destroy():
    pwmRed.stop()
```

```
36      pwmGreen.stop()
37      pwmBlue.stop()
38      GPIO.cleanup()
39
40  if __name__ == '__main__':     # Program entrance
41      print ('Program is starting ... ')
42      setup()
43      try:
44          loop()
45      except KeyboardInterrupt:  # Press ctrl-c to end the program.
            destroy()
```

In last chapter, we learned how to use Python language to make a pin output PWM. In this project, we output to three pins via PWM and the method is exactly the same as we used in the last chapter. In the "while" loop of "loop" function, we first generate three random numbers, and then specify these three random numbers as the PWM values for the three pins, which will make the RGB LED produce multiple colors randomly.

```
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)
```

About the randint() function :

random.randint(a, b)

This function can return a random integer (a whole number value) within the specified range (a, b).

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

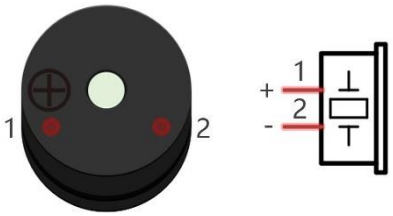
<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push Button Switch x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

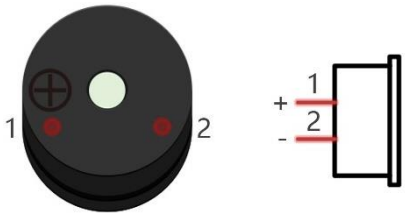
Buzzer

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

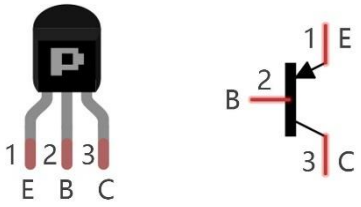
Transistors

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

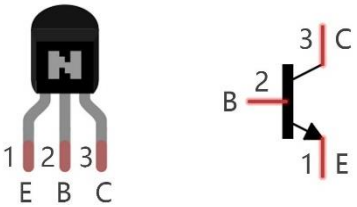
amplify the current.

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



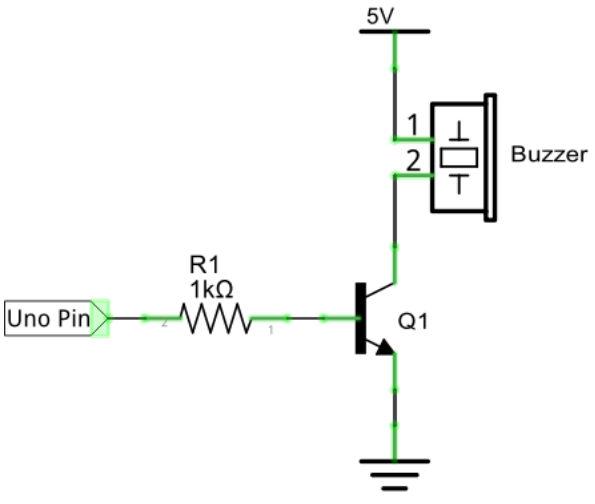
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

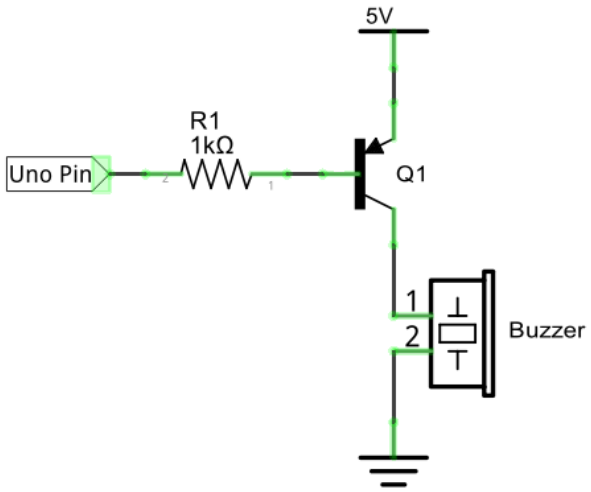
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not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

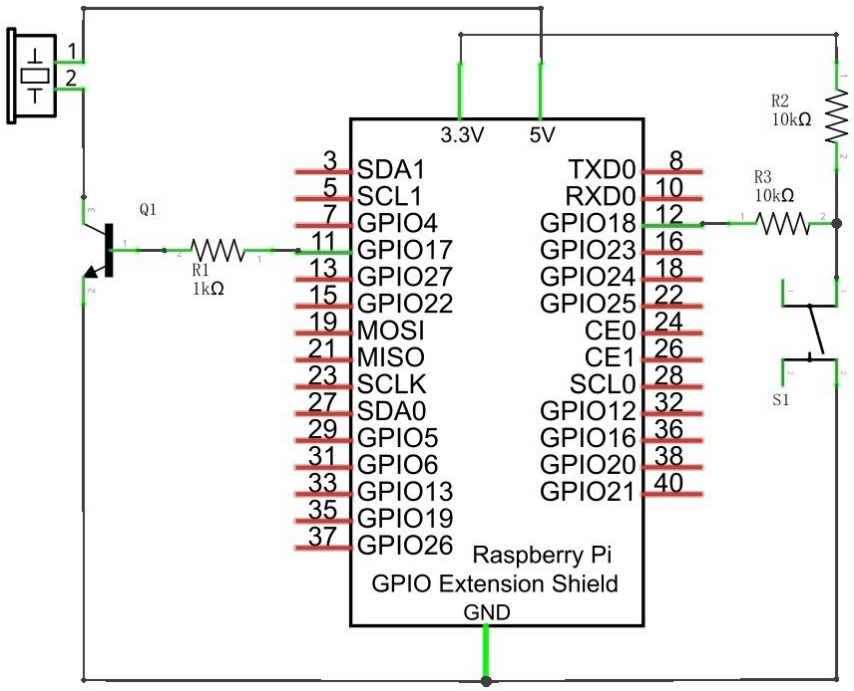


PNP transistor to drive buzzer

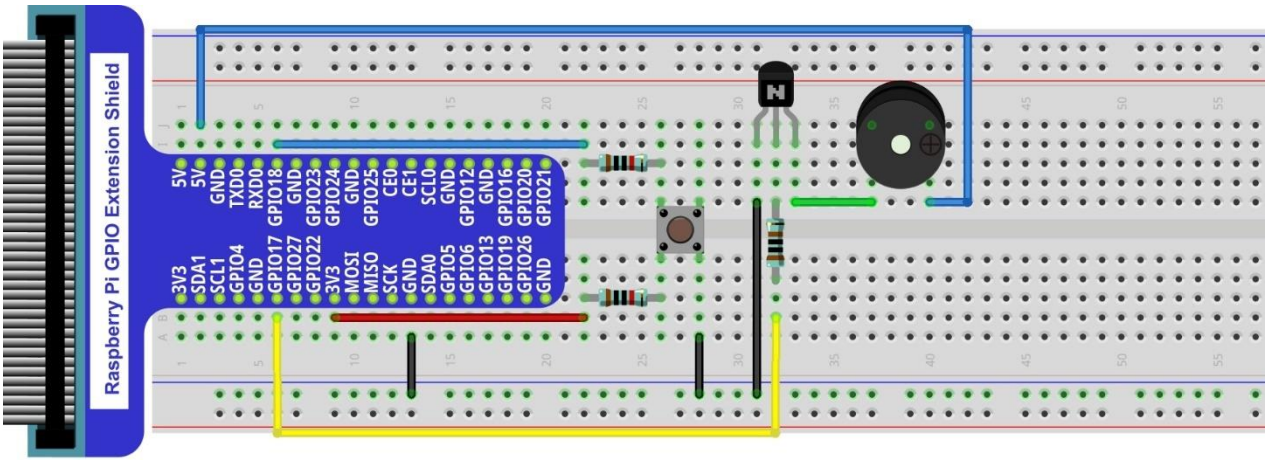


Circuit

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.

C Code 6.1.1 Doorbell

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 06.1.1_Doorbell directory of C code.

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

2. Use following command to compile "Doorbell.c" and generate executable file "Doorbell.c".

```
gcc Doorbell.c -o Doorbell -lwiringPi
```

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

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

The following is the program code:

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

#define buzzerPin 0   //define the buzzerPin
#define buttonPin 1       //define the buttonPin

void main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();

    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){

        if(digitalRead(buttonPin) == LOW){ //button is pressed
            digitalWrite(buzzerPin, HIGH);   //Turn on buzzer
            printf("buzzer turned on >>> \n");
        }
        else {                //button is released
            digitalWrite(buzzerPin, LOW);   //Turn off buzzer
            printf("buzzer turned off <<< \n");
        }
    }
}
```

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.

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_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:

```
import RPi.GPIO as GPIO

buzzerPin = 11    # define buzzerPin
buttonPin = 12    # define buttonPin

def setup():
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)   # set buzzerPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
INPUT mode

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
            GPIO.output(buzzerPin,GPIO.HIGH) # turn on buzzer
            print ('buzzer turned on >>>')
        else : # if button is relessed
            GPIO.output(buzzerPin,GPIO.LOW) # turn off buzzer
            print ('buzzer turned off <<<')

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

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

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.

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.

C 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 C code.

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

2. Use following command to compile "Alertor.c" and generate executable file "Alertor". "-lm" and "-lpthread" compiler options need to added here.

```
gcc Alertor.c -o Alertor -lwiringPi -lm -lpthread
```

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

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:

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

#define buzzerPin    0    //define the buzzerPin
#define buttonPin      1       //define the buttonPin

void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ // frequency of the alertor is consistent with the sine wave
        sinVal = sin(x * (M_PI / 180));         //Calculate the sine value
        toneVal = 2000 + sinVal * 500;      //Add the resonant frequency and weighted sine
value
        softToneWrite(pin,toneVal);             //output corresponding PWM
```

```
17          delay(1);
18      }
19  }
20  void stopAlertor(int pin){
21      softToneWrite(pin,0);
22  }
23  int main(void)
24  {
25      printf("Program is starting ... \n");
26
27      wiringPiSetup();
28
29      pinMode(buzzerPin, OUTPUT);
30      pinMode(buttonPin, INPUT);
31      softToneCreate(buzzerPin); //set buzzerPin
32      pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
33      while(1){
34          if(digitalRead(buttonPin) == LOW){ //button is pressed
35              alertor(buzzerPin);   // turn on buzzer
36              printf("alertor turned on >>> \n");
37          }
38          else {                  //button is released
39              stopAlertor(buzzerPin);   // turn off buzzer
40              printf("alertor turned off <<< \n");
41          }
42      }
43      return 0;
44  }
```

The code is the same to the active buzzer but the method is different. A passive buzzer requires PWM of a certain frequency, so you need to create a software PWM pin though softToneCreate (buzzeRPin). Here softTone is designed to generate square waves with variable frequency and a duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

```
    softToneCreate(buzzeRPin);
```

In the while 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. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```
    void alertor(int pin){
        int x;
        double sinVal, toneVal;
        for(x=0;x<360;x++){ //The frequency is based on the sine curve.
            sinVal = sin(x * (M_PI / 180));
```

```
            toneVal = 2000 + sinVal * 500;
            softToneWrite(pin,toneVal);
            delay(1);
        }
    }
```

If you want to stop the buzzer, just set PWM frequency of the buzzer pin to 0.

```
void stopAlertor(int pin){
    softToneWrite(pin,0);
}
```

The related functions of softTone are described as follows:

int softToneCreate (int pin) ;
This creates a software controlled tone pin.
void softToneWrite (int pin, int freq) ;
This updates the tone frequency value on the given pin.

For more details about softTone, please refer to :http://wiringpi.com/reference/software-tone-library/

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_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:

```
import RPi.GPIO as GPIO
import time
import math

buzzerPin = 11    # define the buzzerPin
buttonPin = 12    # define the buttonPin

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)   # set RGBLED pins to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin to INPUT
mode, and pull up to HIGH level, 3.3V
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            alertor()
            print ('alertor turned on >>> ')
        else :
            stopAlertor()
            print ('alertor turned off <<<')
def alertor():
    p.start(50)
    for x in range(0,361):    # Make frequency of the alertor consistent with the sine wave
        sinVal = math.sin(x * (math.pi / 180.0))        # calculate the sine value
        toneVal = 2000 + sinVal * 500  # Add to the resonant frequency with a Weighted
        p.ChangeFrequency(toneVal)     # Change Frequency of PWM to toneVal
        time.sleep(0.001)

def stopAlertor():
```

```
34      p.stop()
35
36  def destroy():
37      GPIO.output(buzzerPin, GPIO.LOW)      # Turn off buzzer
38      GPIO.cleanup()                        # Release GPIO resource
39
40  if __name__ == '__main__':     # Program entrance
41      print ('Program is starting...')
42      setup()
43      try:
44          loop()
45      except KeyboardInterrupt:  # Press ctrl-c to end the program.
46          destroy()
```

The code is the same to the active buzzer but the method is different. A passive buzzer requires PWM of a certain frequency, so you need to create a software PWM pin through softToneCreate (buzzeRPin). The way to create a PWM was introduced earlier in the BreathingLED and RGB LED projects.

```
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)    # set RGBLED pins to OUTPUT mode
    GPIO.setup(buzzerPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # Set buttonPin to INPUT
mode, and pull up to HIGH level, 3.3V
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);
```

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. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```
def alertor():
    p.start(50)
    for x in range(0,361):
        sinVal = math.sin(x * (math.pi / 180.0))
        toneVal = 2000 + sinVal * 500
        p.ChangeFrequency(toneVal)
        time.sleep(0.001)
```

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

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

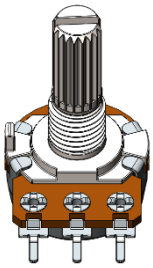
(Important) Chapter 7 ADC

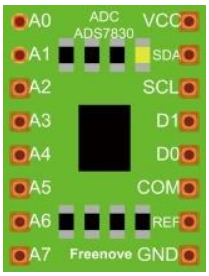
We have learned how to control the brightness of an LED through PWM and that PWM is not a real analog signal. In this chapter, we will learn how to read analog values via an ADC Module and convert these analog values into digital.

Project 7.1 Read the Voltage of Potentiometer

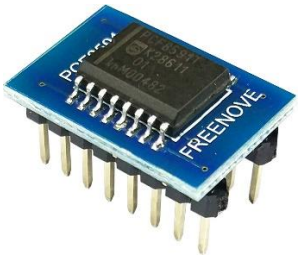
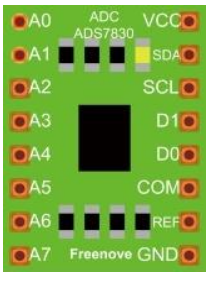
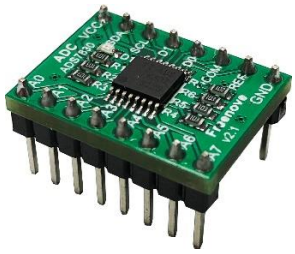
In this project, we will use the ADC function of an ADC Module to read the voltage value of a potentiometer.

Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper Wire M/M x16
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>ADC module x1
 

or

</td><td>Resistor 10kΩ x2
  </td></tr>
</table>

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.

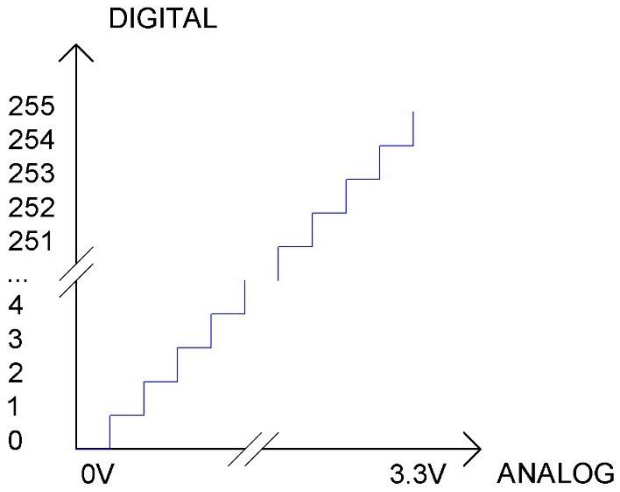
<table>
<tr><th colspan="2">ADC module: PCF8591</th><th colspan="2">ADC module: ADS7830</th></tr>
<tr><td>Model diagram
 
</td><td>Actual Picture
 
</td><td>Model diagram
 
</td><td>Actual Picture
  </td></tr>
</table>

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.

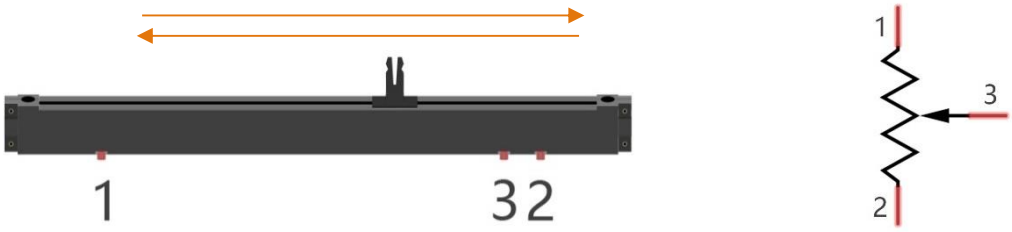
DAC

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

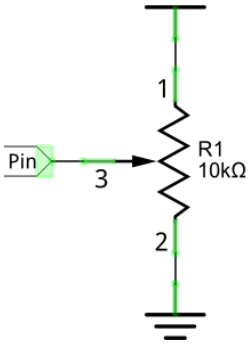
Potentiometer

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.



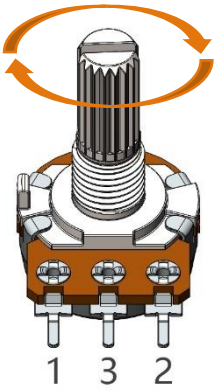
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (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.



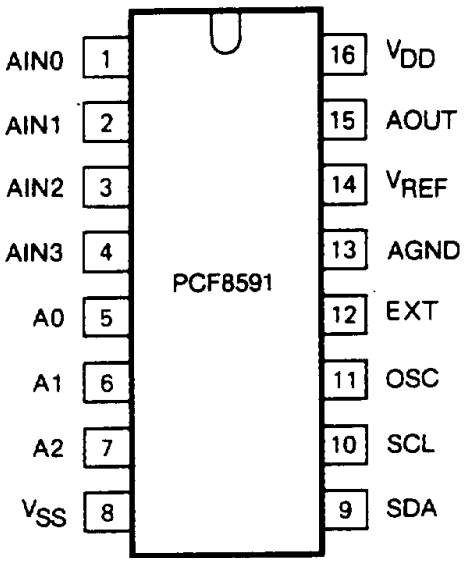
Rotary potentiometer

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	

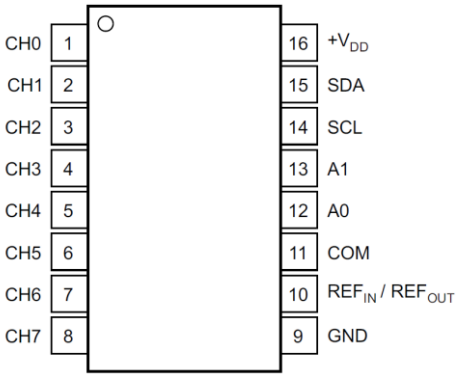
For more details about PCF8591, please refer to the datasheet which can be found on the Internet.

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

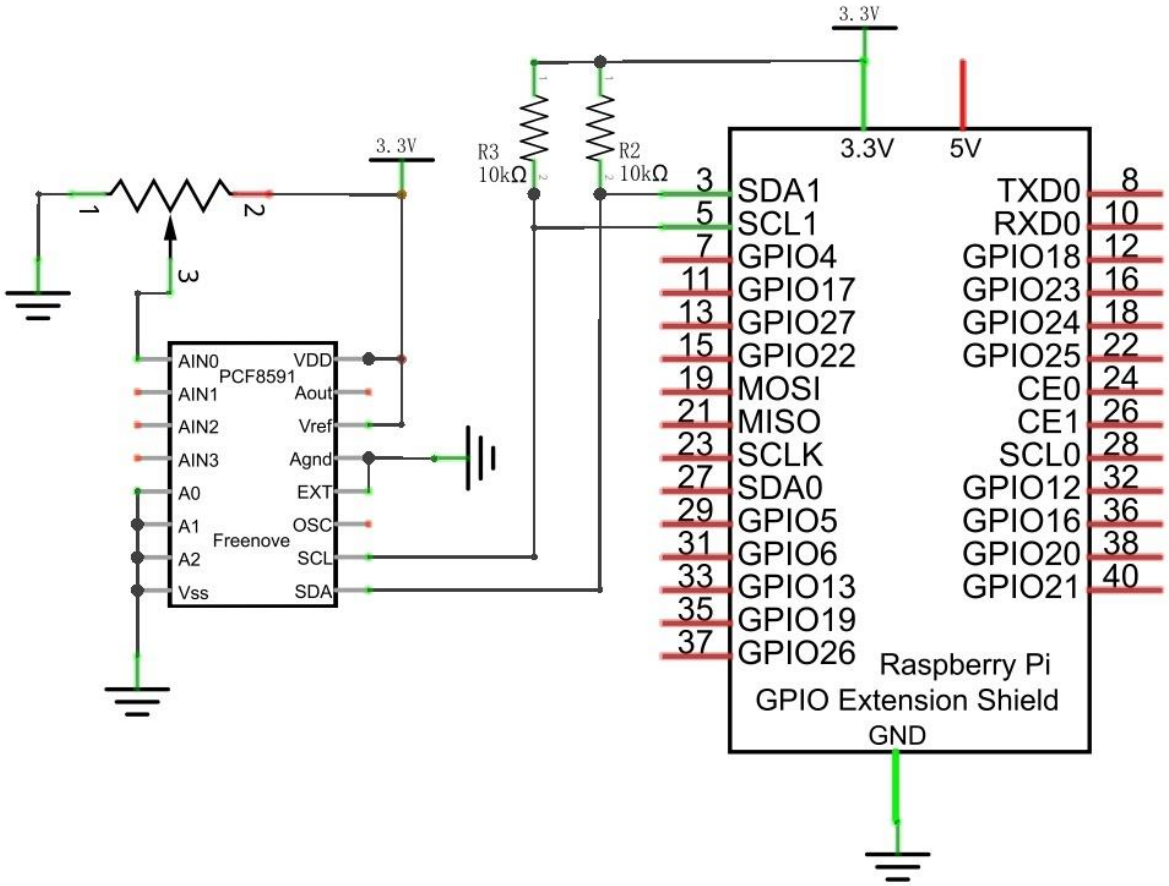


I2C communication

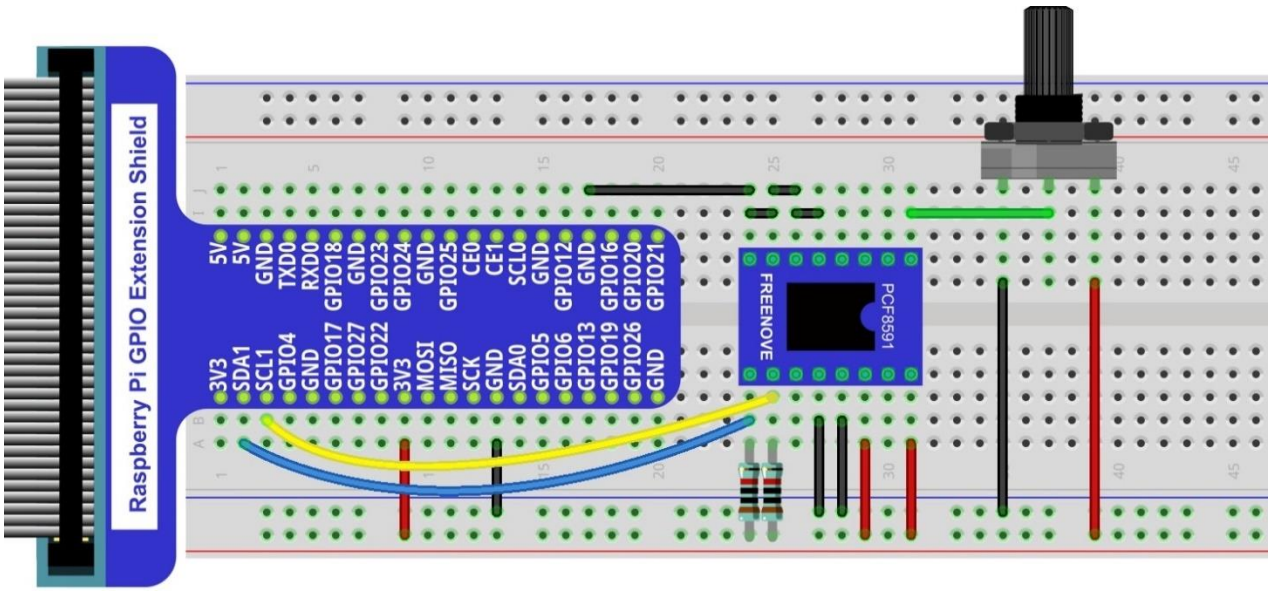
I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique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

Circuit with PCF8591

Schematic diagram



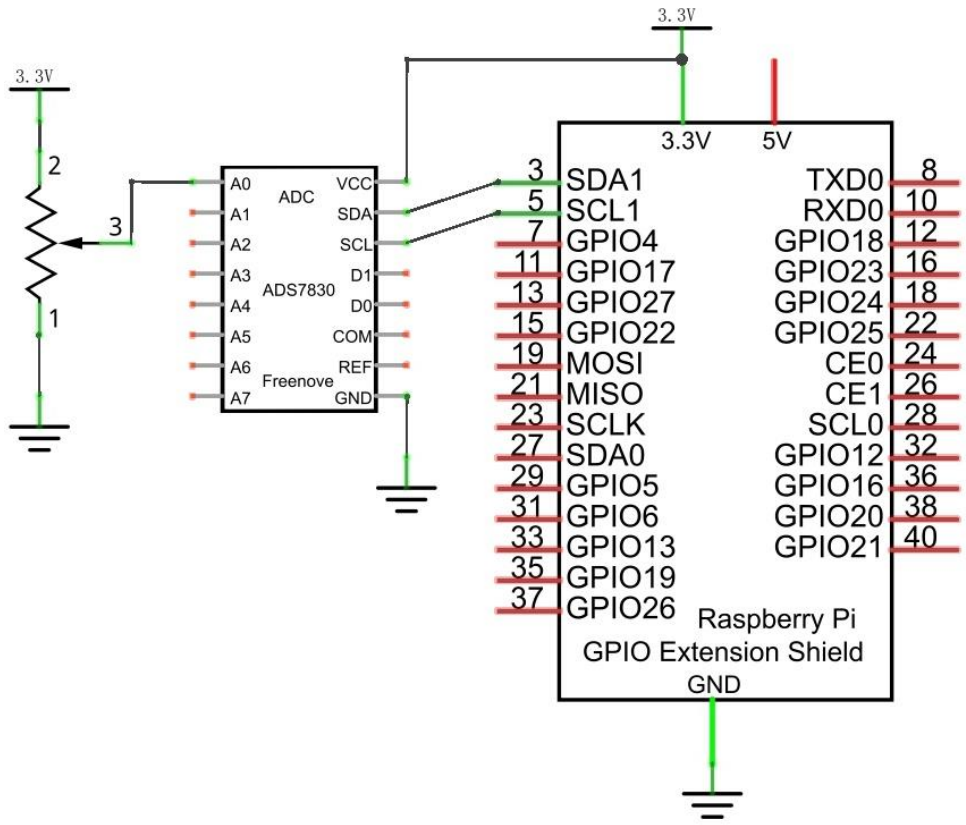
Hardware connection



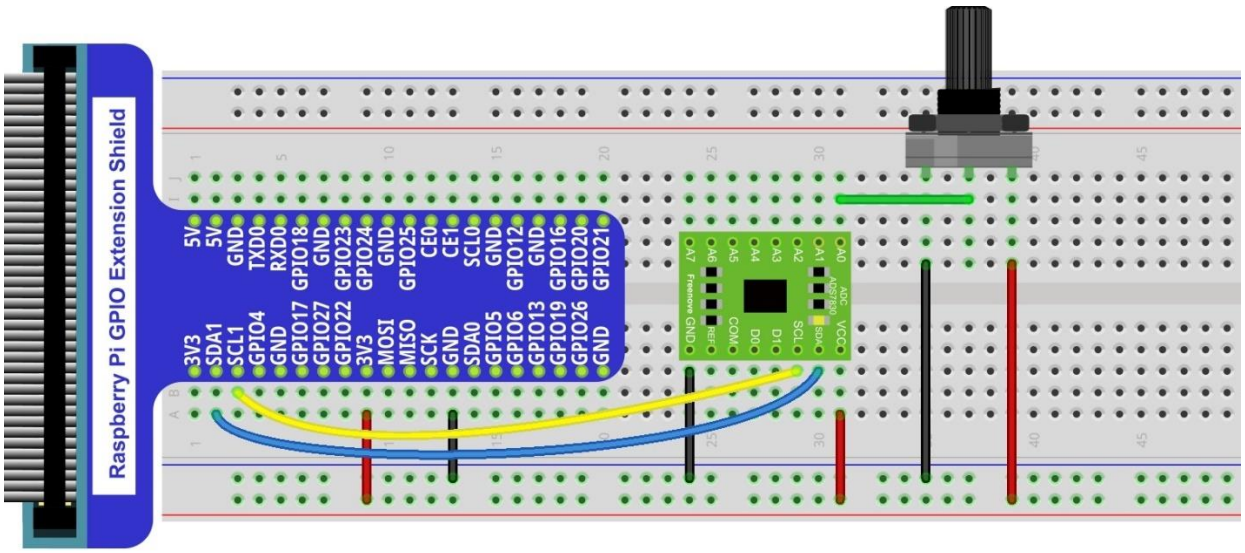
Please keep the **chip mark** consistent to make the chips under right direction and position.

Circuit with ADS7830

Schematic diagram



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



Configure I2C and Install Smbus

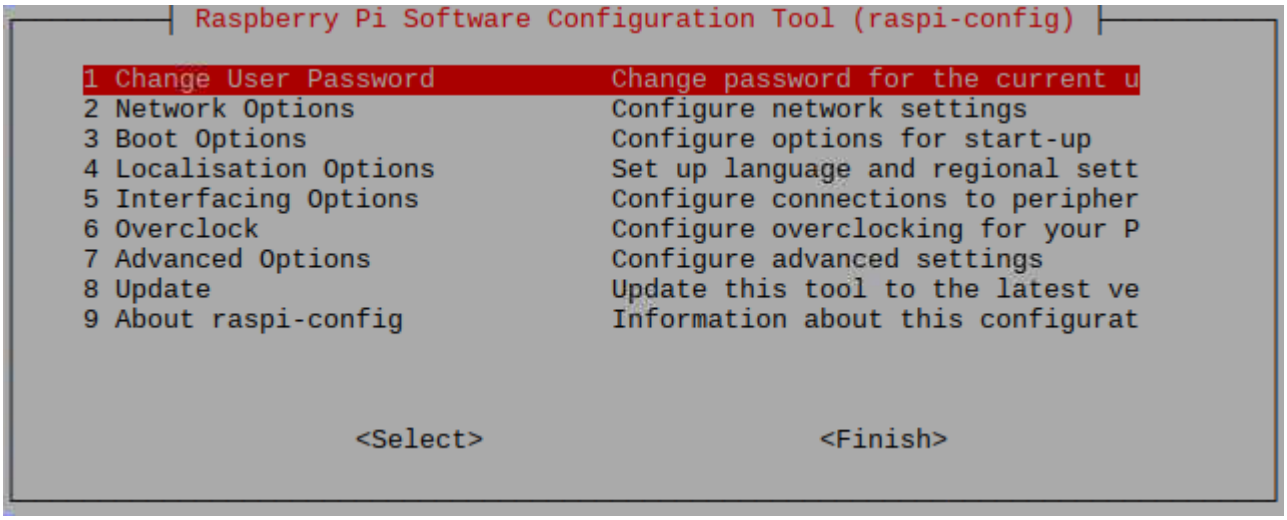
Enable I2C

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:



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:

Install I2C-Tools

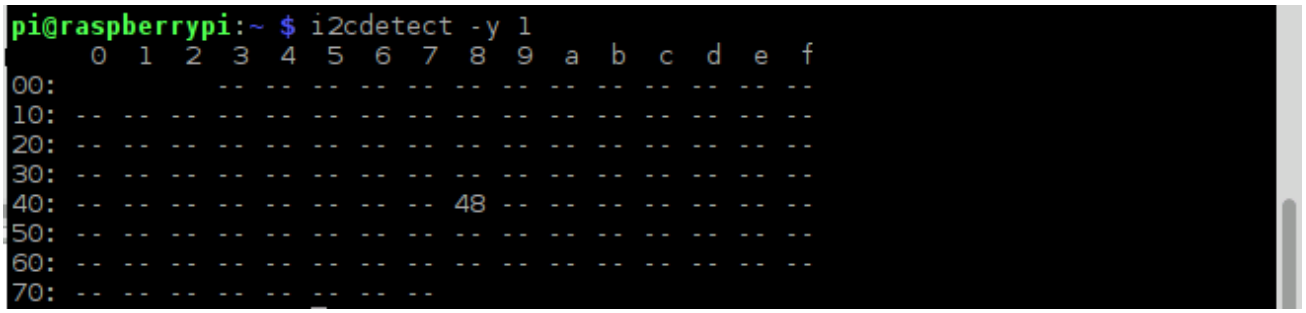
Next, type the command to install I2C-Tools. It is available with the Raspberry Pi OS by default.

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

I2C device address detection:

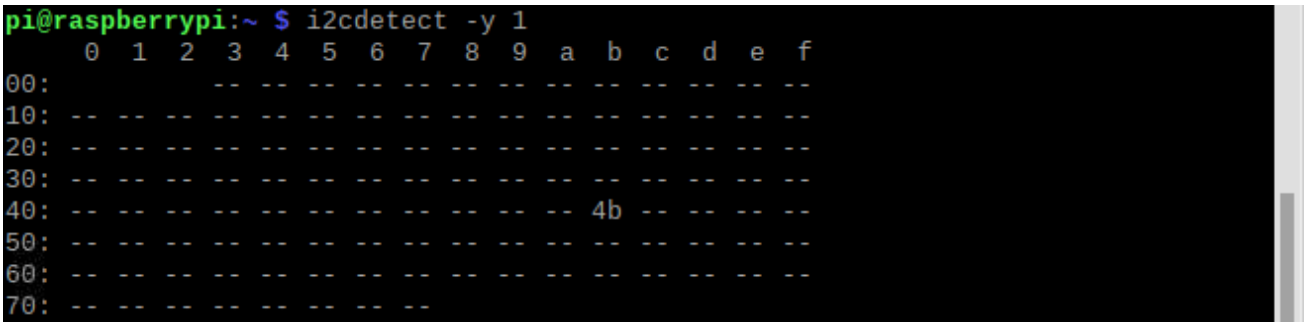
```
i2cdetect -y 1
```

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

C Code 7.1.1 ADC

For C code for the ADC Device, a custom library needs to be installed.

1. Use cd command to enter folder of the ADC Device library.

```
cd ~/Freenove_Kit/Libs/C-Libs/ADCDevice
```

2. Execute command below to install the library.

```
sh ./build.sh
```

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

```
pi@raspberrypi:~/Freenove_Kit/Libs/C-Libs/ADCDevice $ sh ./build.sh
build completed!
```

Next, we will execute the code for this project.

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 07.1.1_ADC directory of C code.

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

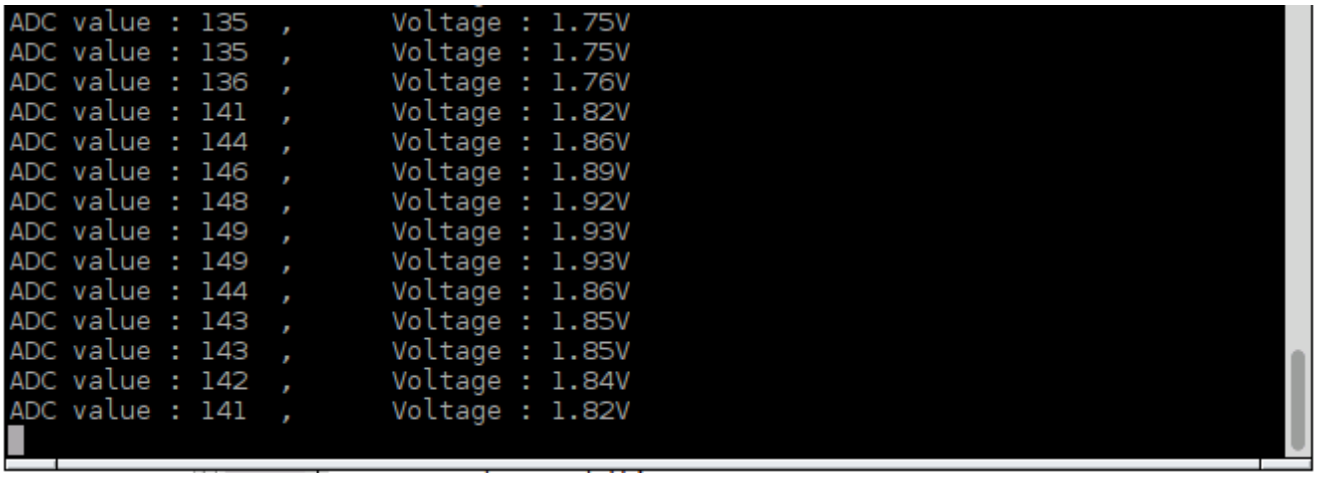
2. Use following command to compile "ADC.cpp" and generate the executable file "ADC".

```
g++ ADC.cpp -o ADC -lwiringPi -lADCDevice
```

3. Then run the generated file "ADC".

```
sudo ./ADC
```

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.



The following is the code:

```
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <stdio.h>
#include <ADCDevice.hpp>

ADCDevice *adc;  // Define an ADC Device class object

int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

    if(adc->detectI2C(0x48)){   // Detect the pcf8591.
        delete adc;
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
```

```
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }

    while(1){
        int adcValue = adc->analogRead(0);  //read analog value of A0 pin
        float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
        delay(100);
    }
}
```

In this code, a custom class library "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 a class pointer adc, and then point to an instantiated object. (Note: An instantiated object is given a name and created in memory or on disk using the structure described within a class declaration.)

```
ADCDevice *adc;  // Define an ADC Device class object
......
adc = new ADCDevice();
```

Then use the member function detectIC(addr) in the class to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the different addresses, we can determine what the ADC module is in the circuit. When the correct module is detected, the pointer adc will point to the address of the object, and then the previously pointed content will be deleted to free memory. The default address of ADC module PCF8591 is 0x48, and that of ADC module ADS7830 is 0x4b.

```
    if(adc->detectI2C(0x48)){   // Detect the pcf8591.
        delete adc;
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
```

When you have a class object pointed to a specific device, you can get the ADC value of the specific channel by calling the member function analogRead (chn) in this class

```
int adcValue = adc->analogRead(0);  //read analog value of A0 pin
```

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

```
        float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
```

Reference

class ADCDevice

This is a base class. All ADC module classes are its derived classes. It has a real function and a virtual function.

`int detectI2C(int addr);`

This is a real function, which is used to detect whether the device with given I2C address exists. If it exists, return 1, otherwise return 0.

`virtual int analogRead(int chn);`

This is a virtual function that reads the ADC value of the specified channel. It is implemented in a derived class.

class PCF8591:public ADCDevice

class ADS7830:public ADCDevice

These two classes are derived from the ADCDevice class and mainly implement the function 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 of 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/C-Libs/ADCDevice/

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 python2 and python3.

```
sudo python2 setup.py install
sudo python3 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_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 : 169, Voltage : 2.19
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
```

The following is the code:

```
1  import time
2  from ADCDevice import *
3
4  adc = ADCDevice() # Define an ADCDevice class object
5
6  def setup():
7      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)

def loop():
    while True:
        value = adc.analogRead(0)    # read the ADC value of channel 0
        voltage = value / 255.0 * 3.3  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.1)

def destroy():
    adc.close()

if __name__ == '__main__':   # Program entrance
    print ('Program is starting ... ')
    try:
        setup()
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        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-1.0.2/src/ADCDevice/ADCdevice.py

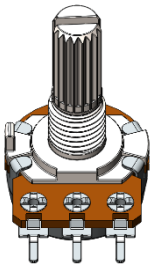
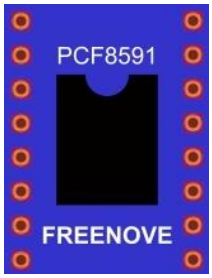
Chapter 8 Potentiometer & LED

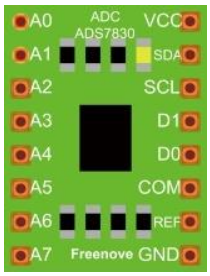
Earlier we learned how to use ADC and PWM. In this chapter, we learn to control the brightness of an LED by using a potentiometer.

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.

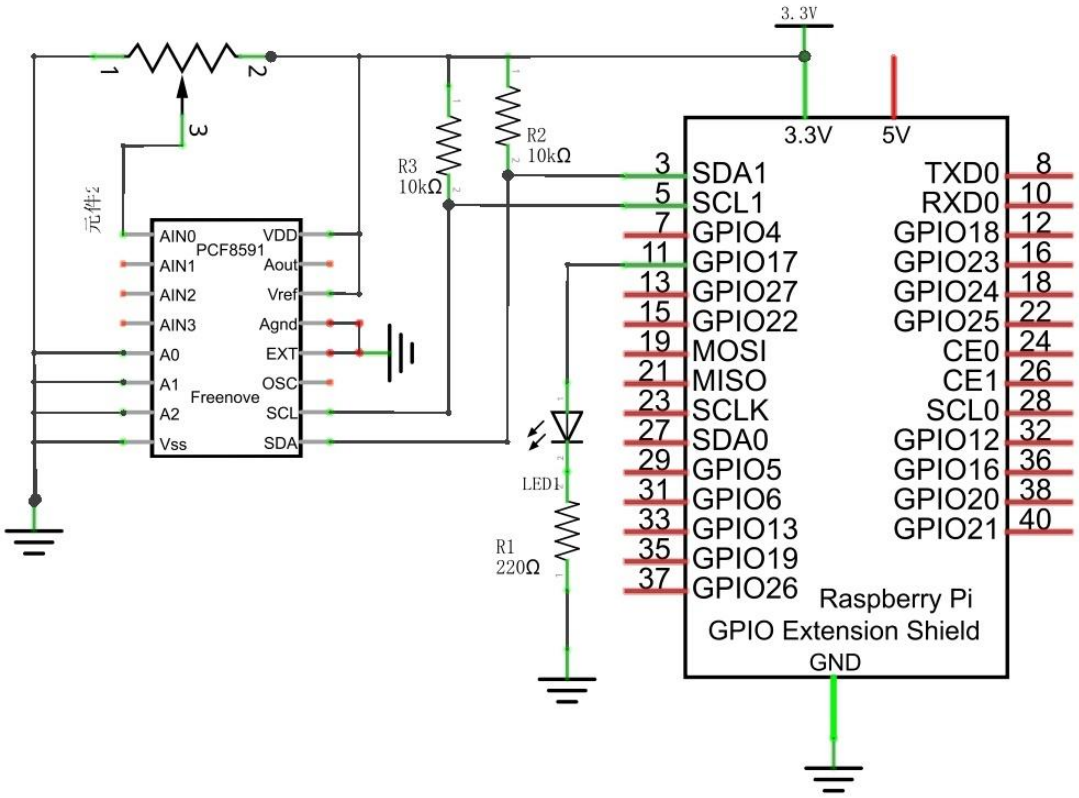
Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire M/M x17
  </td></tr>
<tr><td>Rotary Potentiometer x1
  </td><td colspan="2">ADC Module x1
 

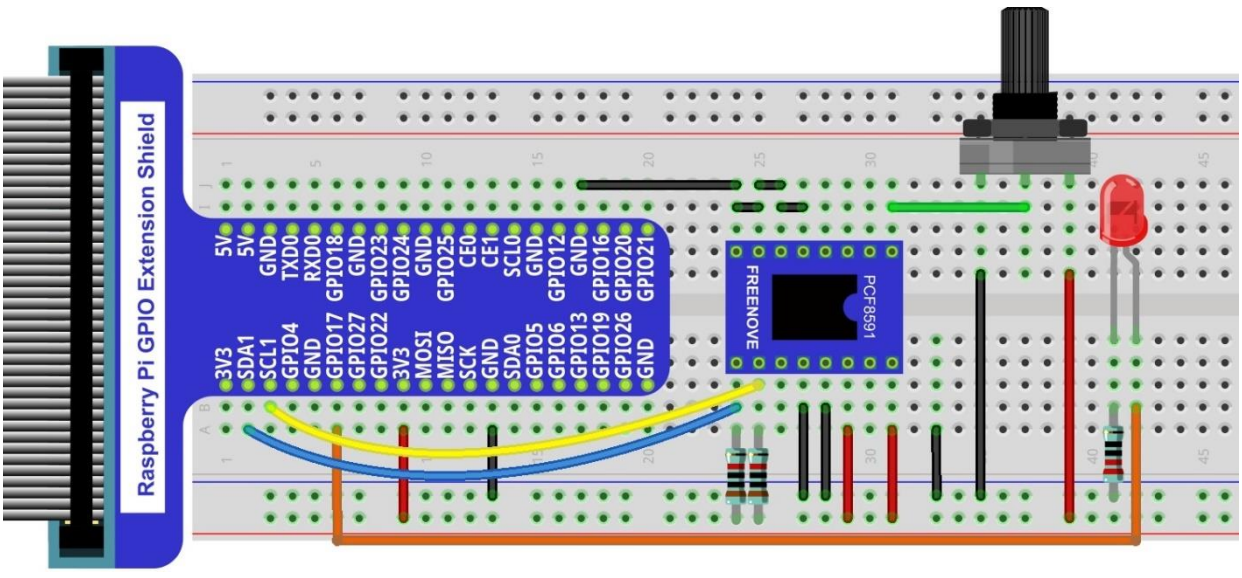
or

</td><td>10kΩ x2
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

Circuit with PCF8591

Schematic diagram

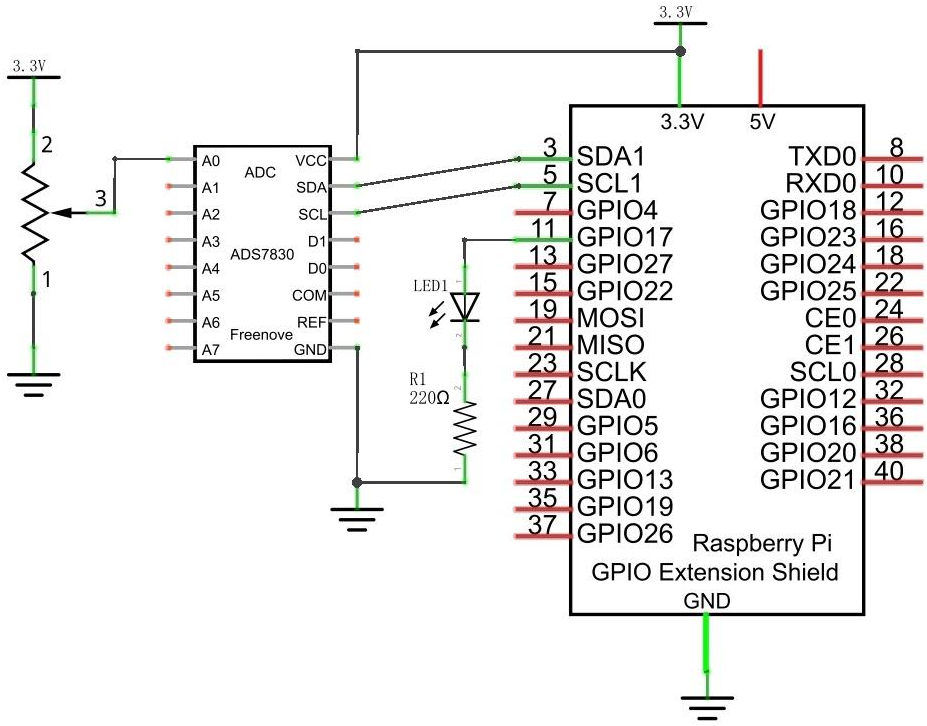


Hardware connection

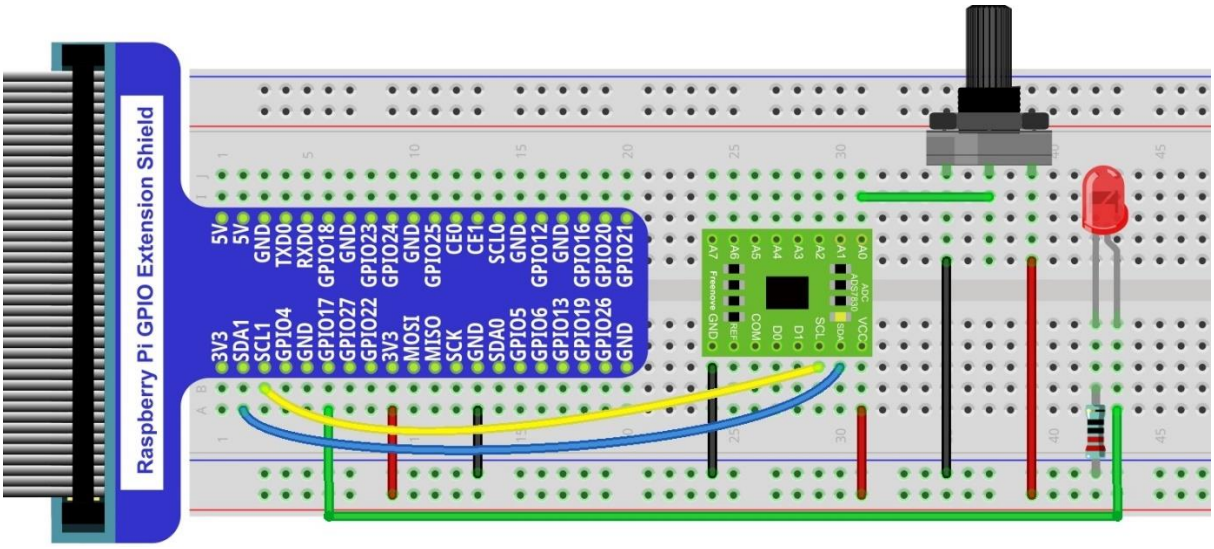


Circuit with ADS7830

Schematic diagram



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



Code

C Code 8.1.1 Softlight

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

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 C code.

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

2. Use following command to compile "Softlight.cpp" and generate executable file "Softlight".

```
g++ Softlight.cpp -o Softlight -lwiringPi -lADCDevice
```

3. Then run the generated file "Softlight".

```
sudo ./Softlight
```

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:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>
#include <ADCDevice.hpp>

#define ledPin 0

ADCDevice *adc;  // Define an ADC Device class object

int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;                 // Free previously pointed memory
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;                // Free previously pointed memory
        adc = new ADS7830();       // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    softPwmCreate(ledPin,0,100);
    while(1){
        int adcValue = adc->analogRead(0);     //read analog value of A0 pin
```

```
32          softPwmWrite(ledPin,adcValue*100/255);     // Mapping to PWM duty cycle
33          float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
34          printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
35          delay(30);
36      }
37      return 0;
38  }
```

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

```
            int adcValue = adc->analogRead(0);     //read analog value of A0 pin
            softPwmWrite(ledPin,adcValue*100/255);     // Mapping to PWM duty cycle
```

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_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  import RPi.GPIO as GPIO
2  import time
3  from ADCDevice import *
4
5  ledPin = 11
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     global p
20     GPIO.setmode(GPIO.BOARD)
21     GPIO.setup(ledPin, GPIO.OUT)
22     p = GPIO.PWM(ledPin, 1000)
23     p.start(0)
24
25 def loop():
26     while True:
27         value = adc.analogRead(0)     # read the ADC value of channel 0
28         p.ChangeDutyCycle(value*100/255)         # Mapping to PWM duty cycle
29         voltage = value / 255.0 * 3.3  # calculate the voltage value
```

```
30          print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
31          time.sleep(0.03)
32
33  def destroy():
34      adc.close()
35
36  if __name__ == '__main__':   # Program entrance
37      print ('Program is starting ... ')
38      try:
39          setup()
40          loop()
41      except KeyboardInterrupt: # Press ctrl-c to end the program.
42          destroy()
```

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
            p.ChangeDutyCycle(value*100/255)        # Mapping to PWM duty cycle
```

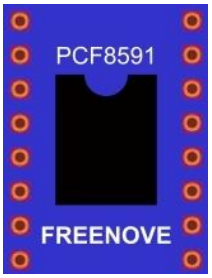
Chapter 9 Photoresistor & LED

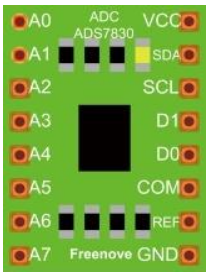
In this chapter, we will learn how to use a photoresistor to make an automatic dimming nightlight.

Project 9.1 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

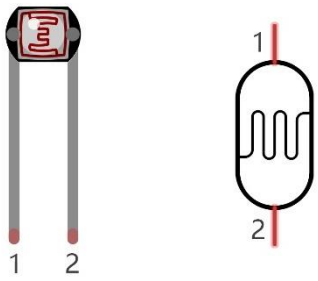
<table>
<tr><td colspan="3">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x15
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">ADC module x1
 

or

</td><td>10kΩ x3
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

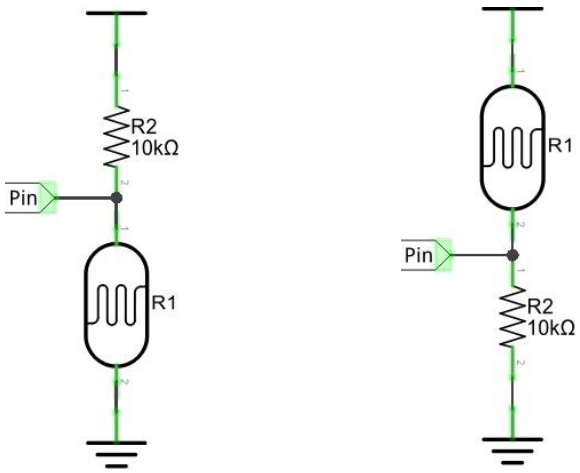
Component knowledge

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:

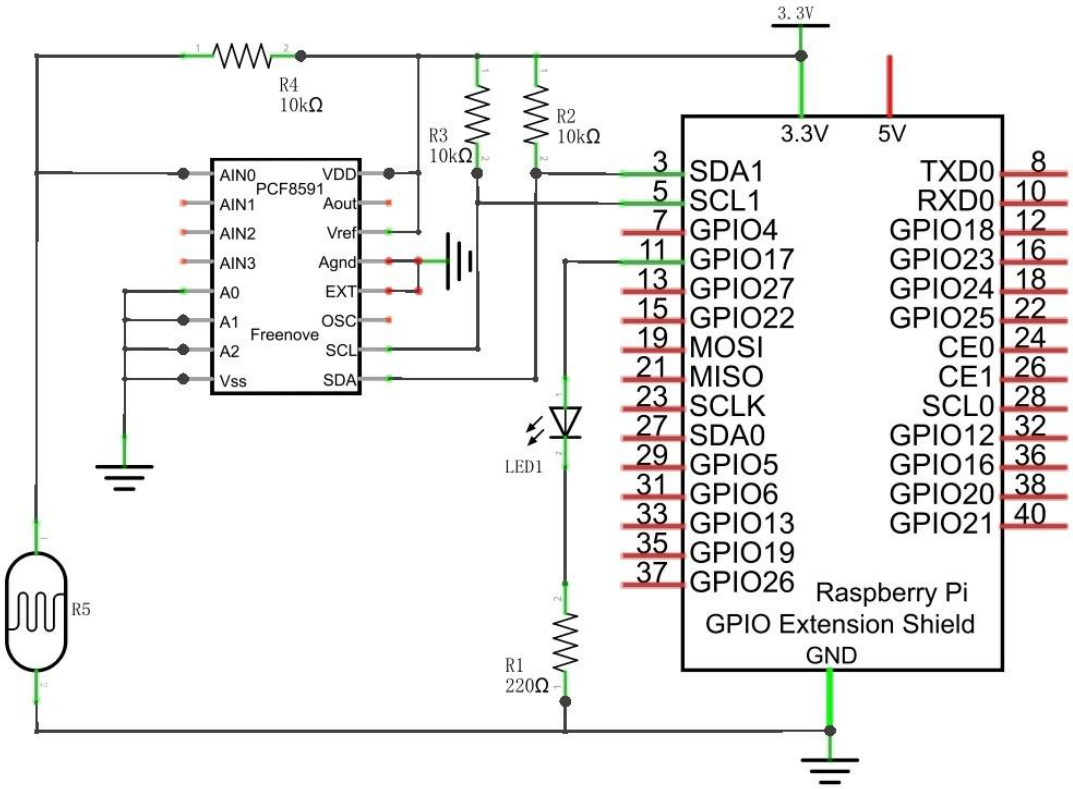


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

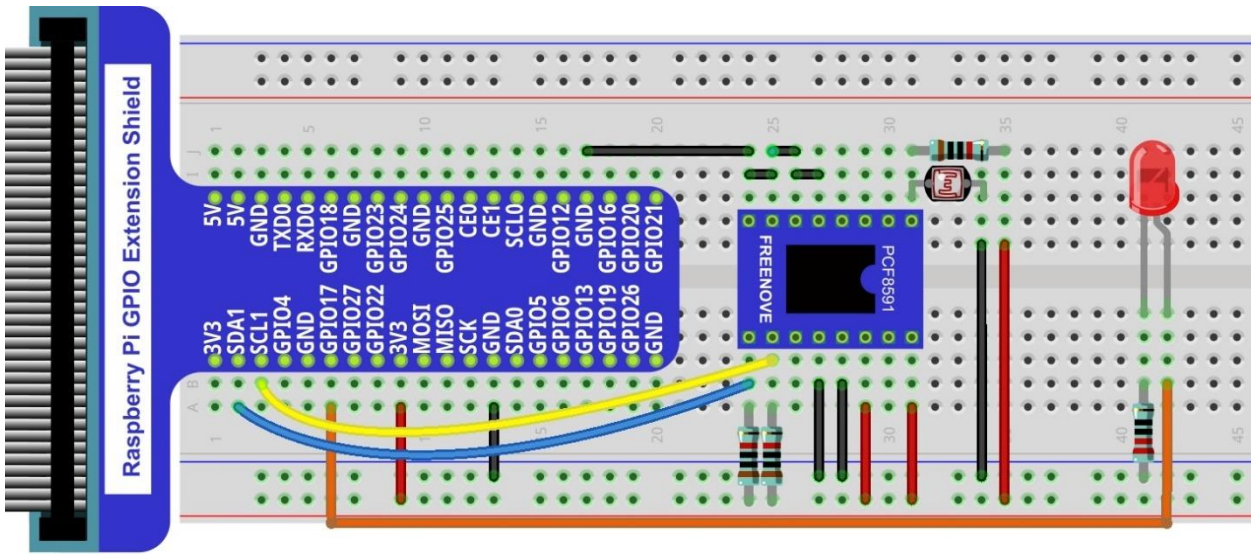
Circuit with PCF8591

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



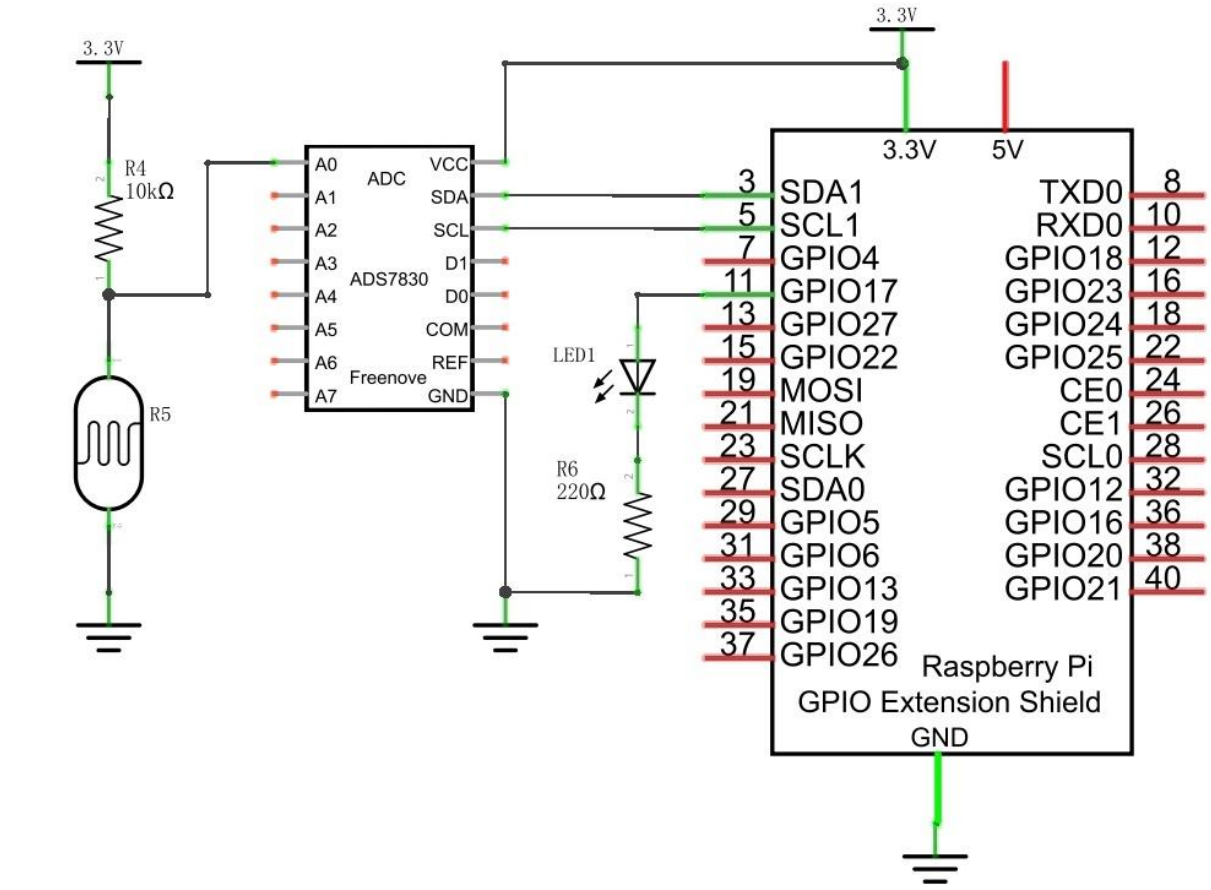
Hardware connection



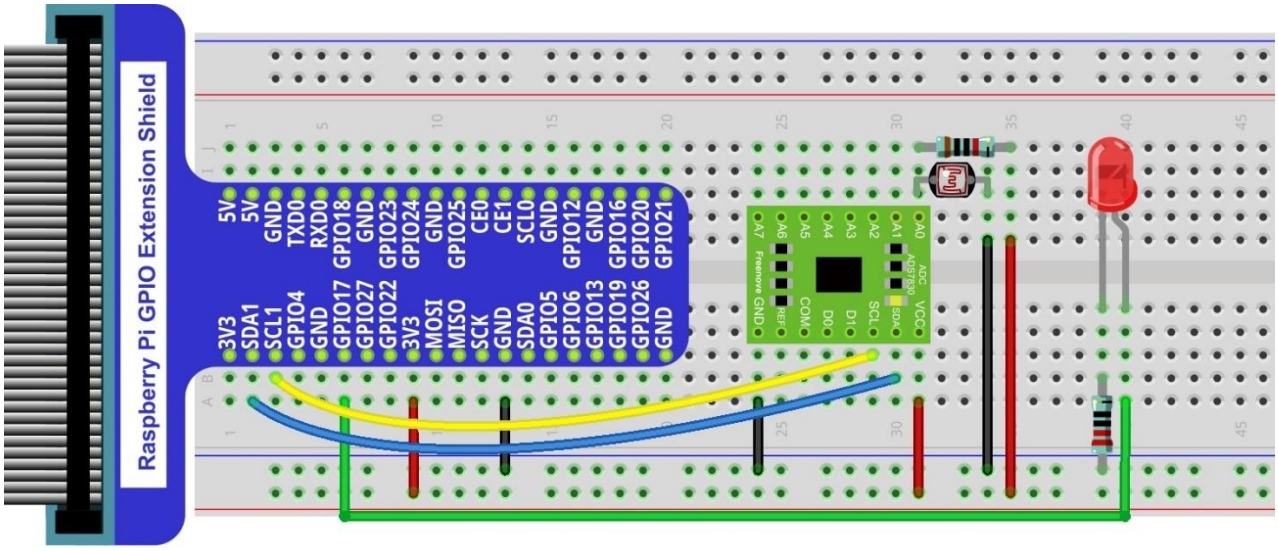
Circuit with ADS7830

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



Code

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

C Code 9.1.1 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 9.1.1_Nightlamp directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/9.1.1_Nightlamp
```

2. Use following command to compile "Nightlamp.cpp" and generate executable file "Nightlamp".

```
g++ Nightlamp.cpp -o Nightlamp -lwiringPi -lADCDevice
```

3. Then run the generated file "Nightlamp".

```
sudo ./Nightlamp
```

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:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>
#include <ADCDevice.hpp>

#define ledPin 0

ADCDevice *adc;  // Define an ADC Device class object

int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;              // Free previously pointed memory
        adc = new PCF8591();     // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;               // Free previously pointed memory
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
```

```
26          return -1;
27      }
28      wiringPiSetup();
29      softPwmCreate(ledPin,0,100);
30      while(1){
31          int value = adc->analogRead(0);  //read analog value of A0 pin
32          softPwmWrite(ledPin,value*100/255);
33          float voltage = (float)value / 255.0 * 3.3;  // calculate voltage
34          printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
35          delay(100);
36      }
37      return 0;
38  }
```

Python Code 9.1.1 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_Code/9.1.1_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:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

ledPin = 11 # define ledPin
adc = ADCDevice() # Define an ADCDevice class object

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)
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)   # set ledPin to OUTPUT mode
    GPIO.output(ledPin,GPIO.LOW)

    p = GPIO.PWM(ledPin,1000) # set PWM Frequence to 1kHz
    p.start(0)

def loop():
    while True:
        value = adc.analogRead(0)    # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 3.3
```

```
32          print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
33          time.sleep(0.01)
34
35  def destroy():
36      adc.close()
37      GPIO.cleanup()
38
39  if __name__ == '__main__':   # Program entrance
40      print ('Program is starting ... ')
41      setup()
42      try:
43          loop()
44      except KeyboardInterrupt:  # Press ctrl-c to end the program.
45          destroy()
```

Chapter 10 Thermistor

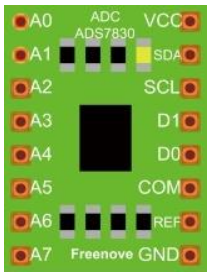
In this chapter, we will learn about Thermistors which are another kind of Resistor.

Project 10.1 Thermometer

A Thermistor is 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

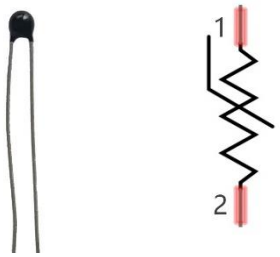
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper Wire M/M x14
  </td></tr>
<tr><td>Thermistor x1
  </td><td colspan="2">ADC module x1
 

or

</td><td>Resistor 10kΩ x3
  </td></tr>
</table>

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;

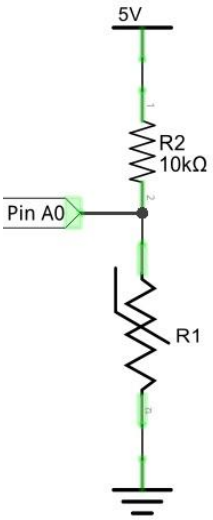
EXP[n] is nth power of e;

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.

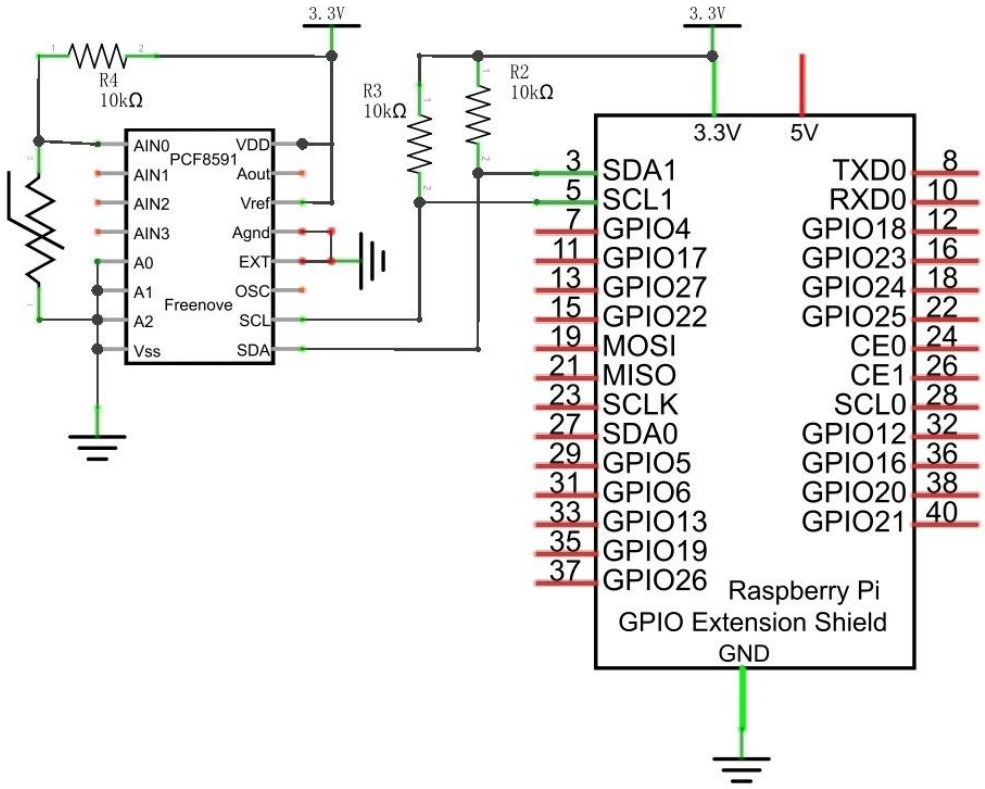
Therefore, the temperature formula can be derived as:

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

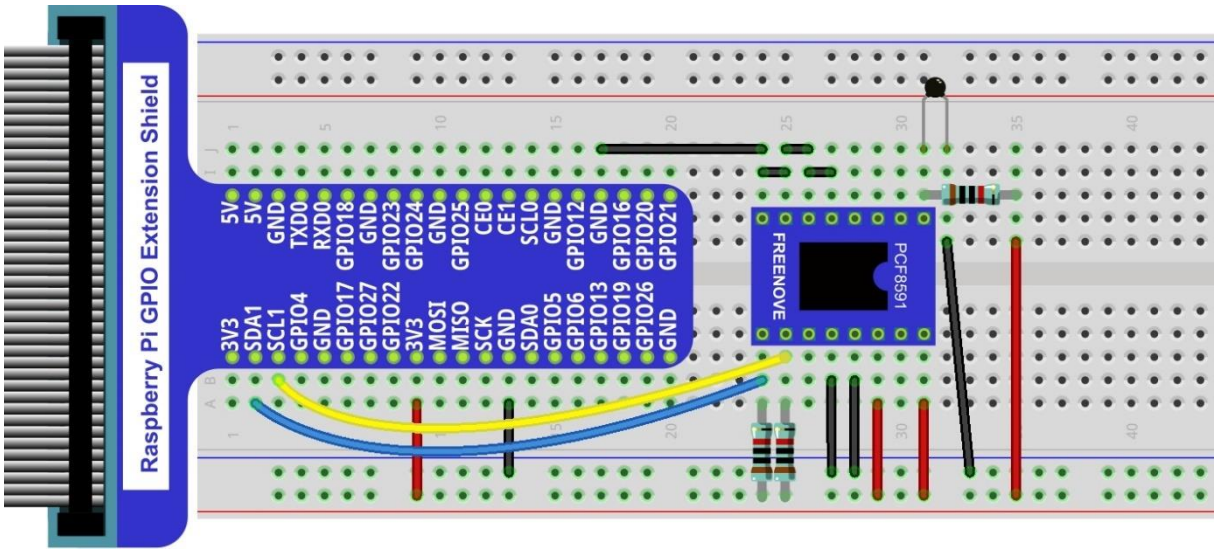
Circuit with PCF8591

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

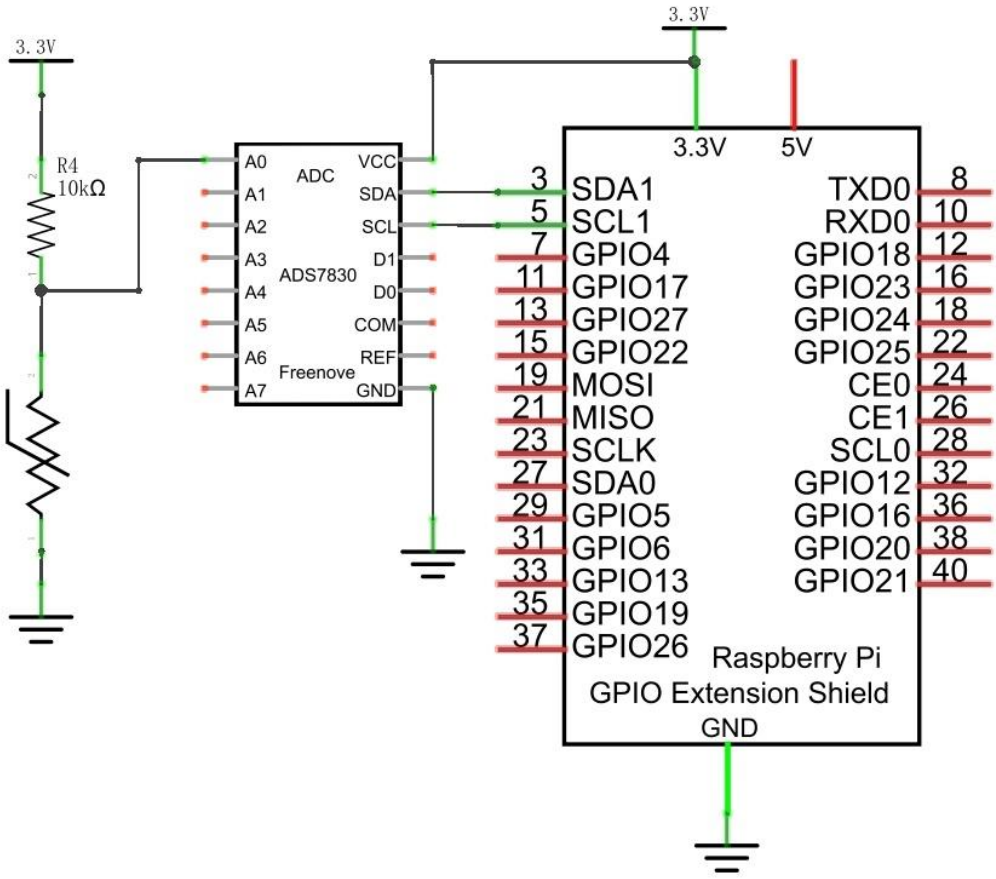


Thermistor has longer pins than the one shown in circuit.

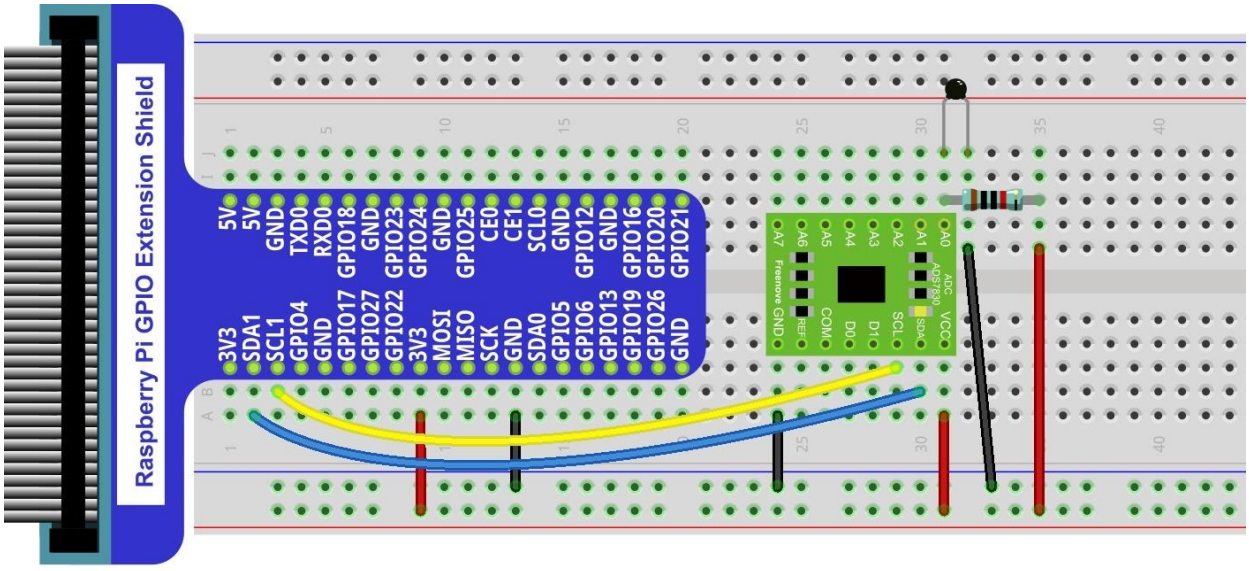
Circuit with ADS

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.

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.

C Code 10.1.1 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 10.1.1_Thermometer directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/10.1.1_Thermometer
```

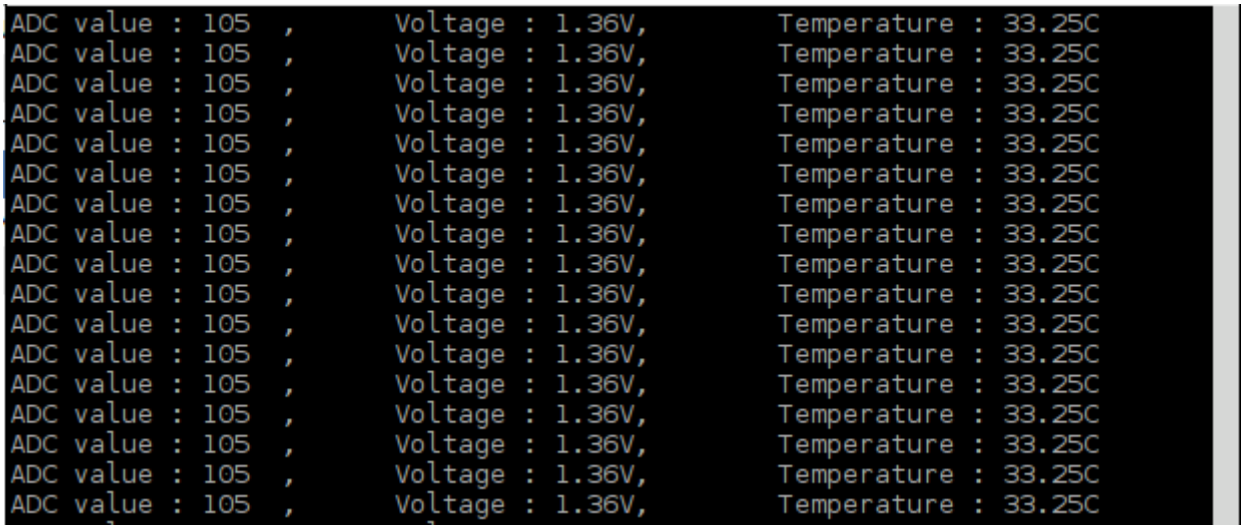
2 Use following command to compile "Thermometer.cpp" and generate executable file "Thermometer".

```
g++ Thermometer.cpp -o Thermometer -lwiringPi -lADCDevice
```

3 Then run the generated file "Thermometer".

```
sudo ./Thermometer
```

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.



The following is the code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <math.h>
#include <ADCDevice.hpp>

ADCDevice *adc;  // Define an ADC Device class object

int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");
```

```
11
12      if(adc->detectI2C(0x48)){    // Detect the pcf8591.
13          delete adc;                 // Free previously pointed memory
14          adc = new PCF8591();    // If detected, create an instance of PCF8591.
15      }
16      else if(adc->detectI2C(0x4b)){// Detect the ads7830
17          delete adc;                // Free previously pointed memory
18          adc = new ADS7830();       // If detected, create an instance of ADS7830.
19      }
20      else{
21          printf("No correct I2C address found, \n"
22          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
23          "Program Exit. \n");
24          return -1;
25      }
26      printf("Program is starting ... \n");
27      while(1){
28          int adcValue = adc->analogRead(0);  //read analog value A0 pin
29          float voltage = (float)adcValue / 255.0 * 3.3;     // calculate voltage
30          float Rt = 10 * voltage / (3.3 - voltage);         //calculate resistance value of
31  thermistor
32          float tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0); //calculate temperature
33  (Kelvin)
34          float tempC = tempK -273.15;         //calculate temperature (Celsius)
35          printf("ADC value : %d  ,\tVoltage : %.2fV,
36  \tTemperature : %.2fC\n",adcValue,voltage,tempC);
37          delay(100);
38      }
39      return 0;
40  }
```

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.

Python Code 10.1.1 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 10.1.1_Thermometer directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/10.1.1_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
```

The following is the code:

```
1  import RPi.GPIO as GPIO
2  import time
3  import math
4  from ADCDevice import *
5
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 ADC value A0 pin
23          voltage = value / 255.0 * 3.3        # calculate voltage
24          Rt = 10 * voltage / (3.3 - voltage)    # calculate resistance value of thermistor
25          tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) # calculate temperature
26  (Kelvin)
27          tempC = tempK -273.15         # calculate temperature (Celsius)
28          print ('ADC Value : %d, Voltage : %.2f,
29  Temperature : %.2f'%(value,voltage,tempC))
30          time.sleep(0.01)
31
32  def destroy():
33      adc.close()
34      GPIO.cleanup()
35
36  if __name__ == '__main__':  # Program entrance
37      print ('Program is starting ... ')
38      setup()
39      try:
40          loop()
41      except KeyboardInterrupt: # Press ctrl-c to end the program.
42          destroy()
```

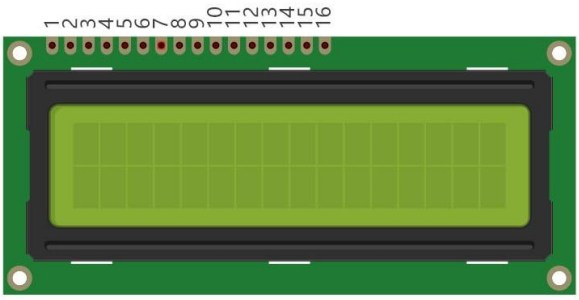
In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of Thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the Thermistor, according to the formula.

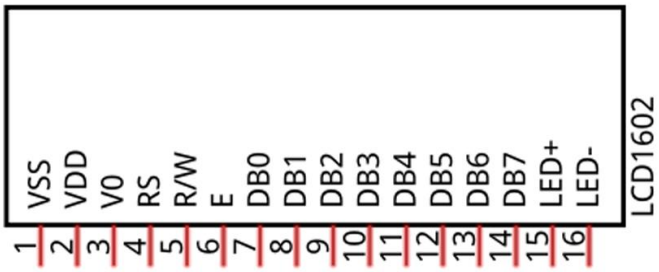
Chapter 11 LCD1602

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

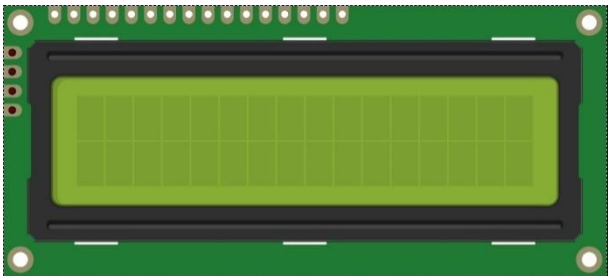
Project 11.1 I2C LCD1602

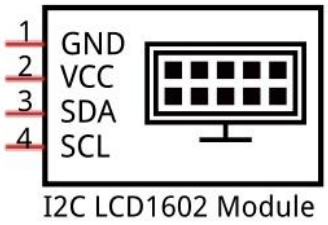
There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII 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.

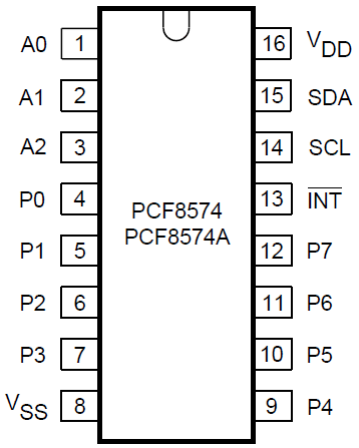




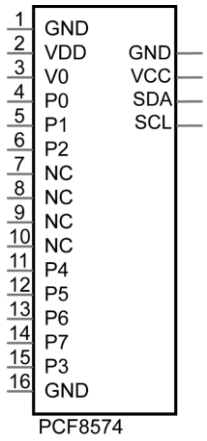
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the RPI bus on your I2C device address through command "i2cdetect -y 1" (refer to the "configuration I2C" section below).

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

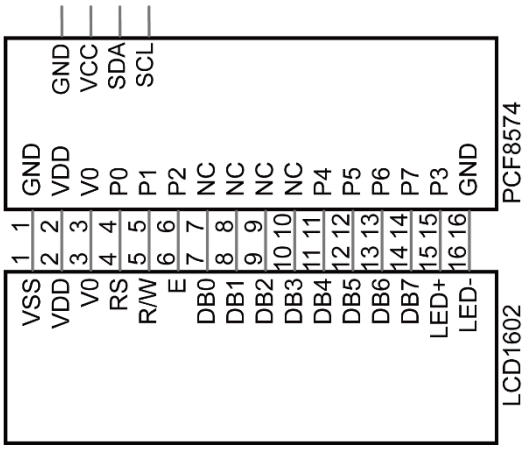
PCF8574 chip pin diagram:



PCF8574 module pin diagram



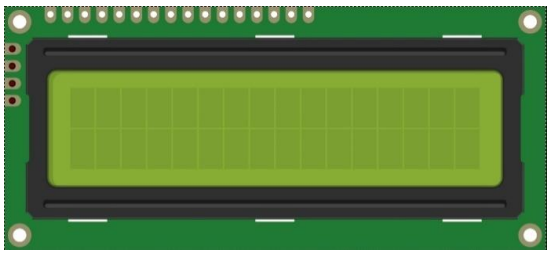
PCF8574 module pins and LCD1602 pins correspond to each other and connected to each other:



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.

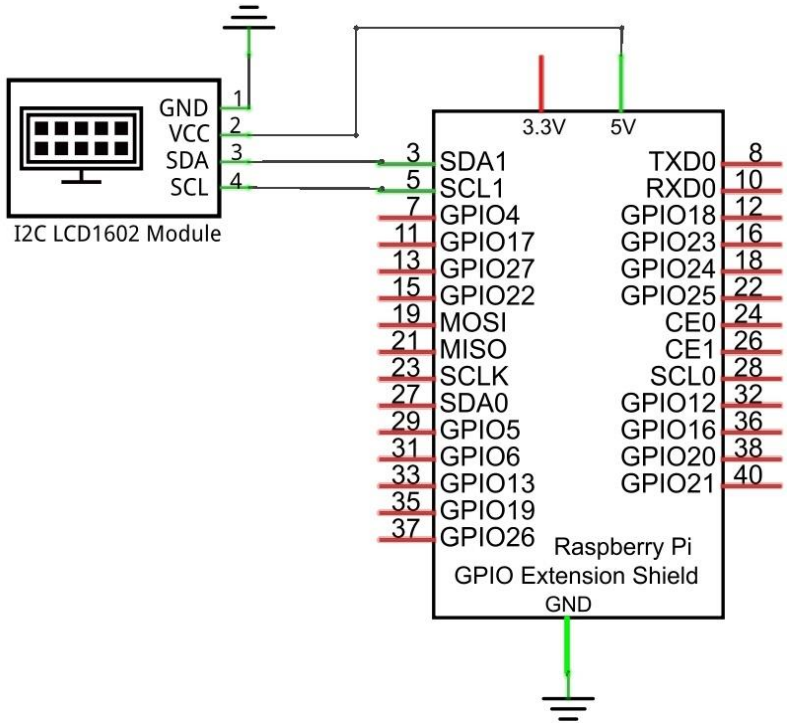
Component List

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

Circuit

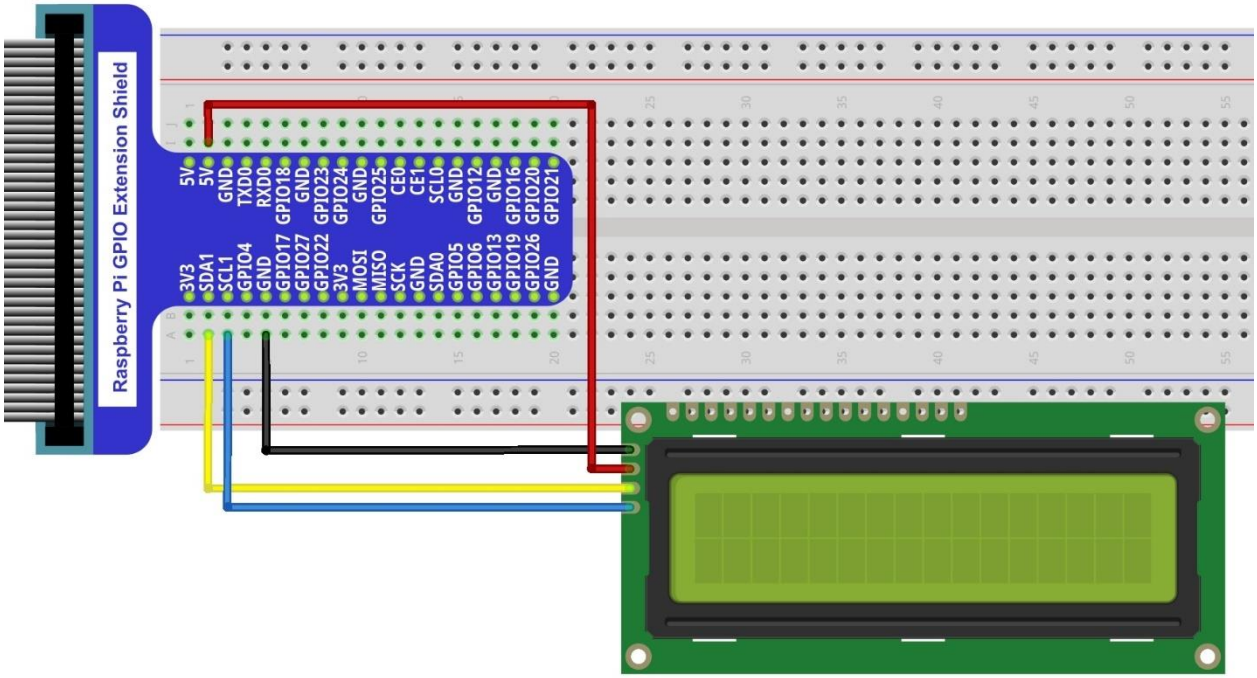
Note that the power supply for I2C LCD1602 in this circuit is 5V.

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.

C Code 11.1.1 I2CLCD1602

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 11.1.1_ I2CLCD1602 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/11.1.1_I2CLCD1602
```

2. Use following command to compile "I2CLCD1602.c" and generate executable file "I2CLCD1602".

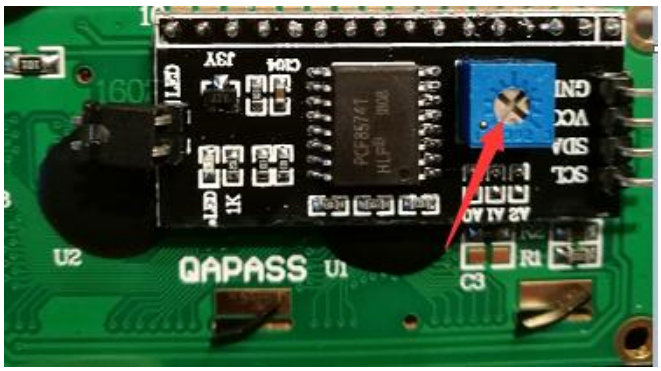
```
gcc I2CLCD1602.c -o I2CLCD1602 -lwiringPi -lwiringPiDev
```

3. Then run the generated file "I2CLCD1602".

```
sudo ./I2CLCD1602
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program is executed, if 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 the Time and Temperature clearly.



The following is the program code:

```c
#include <stdlib.h>
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <pcf8574.h>
#include <lcd.h>
#include <time.h>

int pcf8574_address = 0x27;        // PCF8574T:0x27, PCF8574AT:0x3F
#define BASE 64         // BASE any number above 64
//Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
#define RS      BASE+0
#define RW      BASE+1
#define EN      BASE+2
#define LED     BASE+3
#define D4      BASE+4
```

```
#define D5      BASE+5
#define D6      BASE+6
#define D7      BASE+7

int lcdhd;// used to handle LCD
void printCPUTemperature(){// sub function used to print CPU temperature
    FILE *fp;
    char str_temp[15];
    float CPU_temp;
    // CPU temperature data is stored in this directory.
    fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
    fgets(str_temp,15,fp);      // read file temp
    CPU_temp = atof(str_temp)/1000.0;   // convert to Celsius degrees
    printf("CPU's temperature : %.2f \n",CPU_temp);
    lcdPosition(lcdhd,0,0);     // set the LCD cursor position to (0,0)
    lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp);// Display CPU temperature on LCD
    fclose(fp);
}
void printDataTime(){//used to print system time
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);// get system time
    timeinfo = localtime(&rawtime);//convert to local time
    printf("%s \n",asctime(timeinfo));
    lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)

lcdPrintf(lcdhd,"Time:%02d:%02d:%02d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
int detectI2C(int addr){  //Used to detect i2c address of LCD
    int _fd = wiringPiI2CSetup (addr);
    if (_fd < 0){
        printf("Error address : 0x%x \n",addr);
        return 0 ;
    }
    else{
        if(wiringPiI2CWrite(_fd,0) < 0){
            printf("Not found device in address 0x%x \n",addr);
            return 0;
        }
        else{
            printf("Found device in address 0x%x \n",addr);
            return 1 ;
        }
```

```
61      }
62  }
63  int main(void){
64      int i;
65      printf("Program is starting ...\n");
66      wiringPiSetup();
67      if(detectI2C(0x27)){
68          pcf8574_address = 0x27;
69      }else if(detectI2C(0x3F)){
70          pcf8574_address = 0x3F;
71      }else{
72          printf("No correct I2C address found, \n"
73          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
74          "Program Exit. \n");
75          return -1;
76      }
77      pcf8574Setup(BASE,pcf8574_address);//initialize PCF8574
78      for(i=0;i<8;i++){
79          pinMode(BASE+i,OUTPUT);      //set PCF8574 port to output mode
80      }
81      digitalWrite(LED,HIGH);      //turn on LCD backlight
82      digitalWrite(RW,LOW);        //allow writing to LCD
83      lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);// initialize LCD and return  “handle”
84  used to handle LCD
85      if(lcdhd == -1){
86          printf("lcdInit failed !");
87          return 1;
88      }
89      while(1){
90          printCPUTemperature();//print CPU temperature
91          printDataTime();        // print system time
92          delay(1000);
93      }
94      return 0;
95  }
```

From the code, we can see that the PCF8591 and the PCF8574 have many similarities in using the I2C interface to expand the GPIO RPI.

First, define the I2C address of the PCF8574 and the Extension of the GPIO pin, which is connected to the

GPIO pin of the LCD1602. LCD1602 has two different i2c addresses. Set 0x27 as default.

```
int pcf8574_address = 0x27;        // PCF8574T:0x27, PCF8574AT:0x3F
#define BASE 64         // BASE any number above 64
//Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
#define RS      BASE+0
#define RW      BASE+1
#define EN      BASE+2
#define LED     BASE+3
#define D4      BASE+4
#define D5      BASE+5
#define D6      BASE+6
#define D7      BASE+7
```

Then, in main function, initialize the PCF8574, set all the pins to output mode, and turn ON the LCD1602 backlight (without the backlight the Display is difficult to read).

```
    pcf8574Setup(BASE, pcf8574_address);// initialize PCF8574
    for(i=0;i<8;i++){
        pinMode(BASE+i,OUTPUT);      // set PCF8574 port to output mode
    }
    digitalWrite(LED,HIGH);      // turn on LCD backlight
```

Then use lcdInit() to initialize LCD1602 and set the RW pin of LCD1602 to 0 (can be written) according to requirements of this function. The return value of the function called "Handle" is used to handle LCD1602".

```
        lcdhd = lcdInit(2, 16, 4, RS, EN, D4, D5, D6, D7, 0, 0, 0, 0);// initialize LCD and return
“handle” used to handle LCD
```

Details about lcdInit():

int lcdInit (int rows, int cols, int bits, int rs, int strb,
int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7) ;

This is the main initialization function and must be executd first before you use any other LCD functions. **Rows** and **cols** are the rows and columns of the Display (e.g. 2, 16 or 4, 20). **Bits** is the number of how wide the number of bits is on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the Display's RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the RPi to the display. Only the first 4 are used if you are running the display in 4-bit mode.
The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault (usually incorrect parameter)
For more details about LCD Library, please refer to: https://projects.drogon.net/raspberry-pi/wiringpi/lcd-library/

In the next “while”, two subfunctions are called to display the RPi's CPU Temperature and the SystemTime. First look at subfunction printCPUTemperature(). The CPU temperature data is stored in the "/sys/class/thermal/thermal_zone0/temp" file. We need to read the contents of this file, which converts it to temperature value stored in variable CPU_temp and uses lcdPrintf() to display it on LCD.

```
void printCPUTemperature(){//subfunction used to print CPU temperature

    FILE *fp;
    char str_temp[15];
```

```
    float CPU_temp;
    // CPU temperature data is stored in this directory.
    fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
    fgets(str_temp,15,fp);      // read file temp
    CPU_temp = atof(str_temp)/1000.0;   // convert to Celsius degrees
    printf("CPU's temperature : %.2f \n",CPU_temp);
    lcdPosition(lcdhd,0,0);     // set the LCD cursor position to (0,0)
    lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp);// Display CPU temperature on LCD
    fclose(fp);
}
```

Details about lcdPosition() and lcdPrintf():

lcdPosition (int handle, int x, int y);
Set the position of the cursor for subsequent text entry.
lcdPutchar (int handle, uint8_t data) **lcdPuts (int handle, char *string)** **lcdPrintf (int handle, char *message, …)**
These output a single ASCII character, a string or a formatted string using the usual print formatting commands to display individual characters (it is how you are able to see characters on your computer monitor).

Next is subfunction printDataTime() used to display System Time. First, it gets the Standard Time and stores it into variable Rawtime, and then converts it to the Local Time and stores it into timeinfo, and finally displays the Time information on the LCD1602 Display.

```
void printDataTime(){//used to print system time
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);// get system time
    timeinfo = localtime(&rawtime);// convert to local time
    printf("%s \n",asctime(timeinfo));
    lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)
    lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
```

Python Code 11.1.1 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 11.1.1_ I2CLCD1602 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/11.1.1_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.

NOTE: After the program is executed, if 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 the Time and Temperature clearly.

The following is the program code:

```
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

from time import sleep, strftime
from datetime import datetime

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'

def get_time_now():     # get system time
    return datetime.now().strftime('    %H:%M:%S')

def loop():
    mcp.output(3,1)     # turn on LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
    while(True):
        #lcd.clear()
```

```
22          lcd.setCursor(0,0)  # set cursor position
23          lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
24          lcd.message( get_time_now() )   # display the time
25          sleep(1)
26
27  def destroy():
28      lcd.clear()
29
30  PCF8574_address = 0x27  # I2C address of the PCF8574 chip.
31  PCF8574A_address = 0x3F  # I2C address of the PCF8574A chip.
32  # Create PCF8574 GPIO adapter.
33  try:
34      mcp = PCF8574_GPIO(PCF8574_address)
35  except:
36      try:
37          mcp = PCF8574_GPIO(PCF8574A_address)
38      except:
39          print ('I2C Address Error !')
40          exit(1)
41  # Create LCD, passing in MCP GPIO adapter.
42  lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
43
44  if __name__ == '__main__':
45      print ('Program is starting ... ')
46      try:
47          loop()
48      except KeyboardInterrupt:
49          destroy()
```

Two modules are used in the code, PCF8574.py and Adafruit_LCD1602.py. These two documents and the code files are stored in the same directory, and neither of them is dispensable. Please DO NOT DELETE THEM! PCF8574.py is used to provide I2C communication mode and operation method of some of the ports for the RPi and PCF8574 IC Chip. Adafruit module Adafruit_LCD1602.py is used to provide some functional operation method for the LCD1602 Display.

In the code, first get the object used to operate the PCF8574's port, then get the object used to operate the LCD1602.

```
address = 0x27  # I2C address of the PCF8574 chip.
# Create PCF8574 GPIO adapter.
mcp = PCF8574_GPIO(address)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
```

According to the circuit connection, port 3 of PCF8574 is connected to the positive pole of the LCD1602 Display's backlight. Then in the loop () function, use of mcp.output (3,1) to turn the LCD1602 Display's backlight ON and then set the number of LCD lines and columns.

```
def loop():
    mcp.output(3,1)     # turn on the LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
```

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

```
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)  # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( 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 PCF8574.py and Adafruit_LCD1602.py:

Module PCF8574

This module provides two classes **PCF8574_I2C** and **PCF8574_GPIO**.
Class **PCF8574_I2C**: provides reading and writing method for PCF8574.
Class **PCF8574_GPIO**: provides a standardized set of GPIO functions.
More information can be viewed through opening PCF8574.py.
Adafruit_LCD1602 Module

Module Adafruit_LCD1602

This module provides the basic operation method of LCD1602, including class Adafruit_CharLCD. Some member functions are described as follows:
def begin(self, cols, lines): set the number of lines and columns of the screen.
def clear(self): clear the screen
def setCursor(self, col, row): set the cursor position
def message(self, text): display contents
More information can be viewed through opening Adafruit_CharLCD.py.

Chapter 12 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 12.1 Remote LED

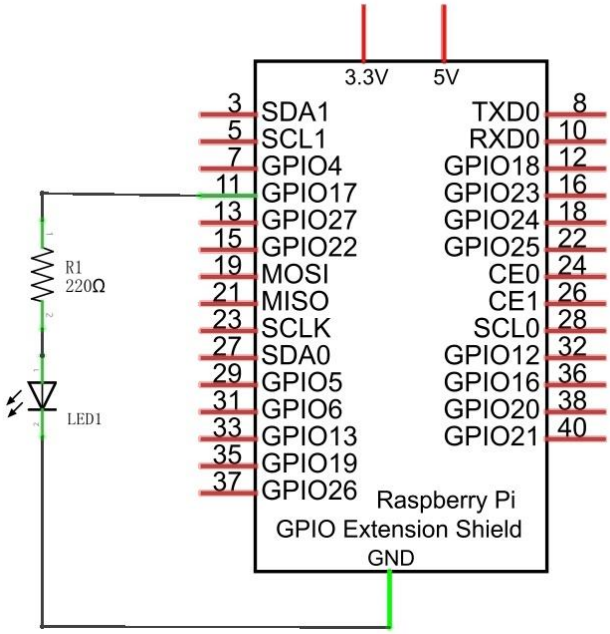
In this project, we need to build a WebIOPi service, and then use the RPi GPIO to control an LED through the web browser of phone or PC.

Component List

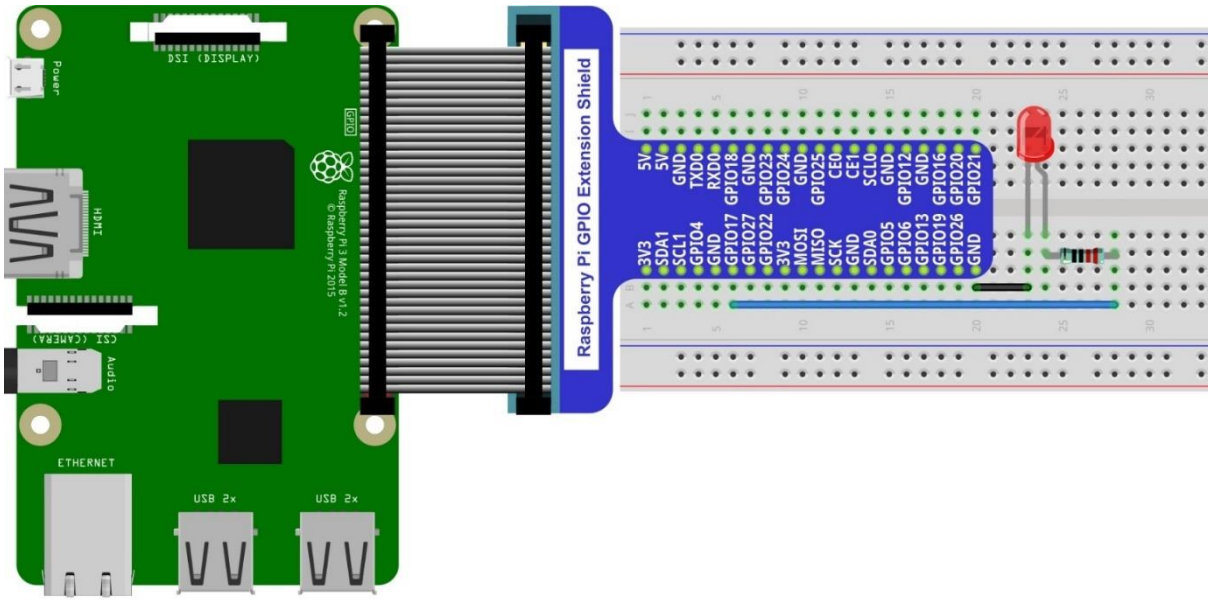
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1 	Resistor 220Ω x1 
Jumper M/M x2 		

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
```

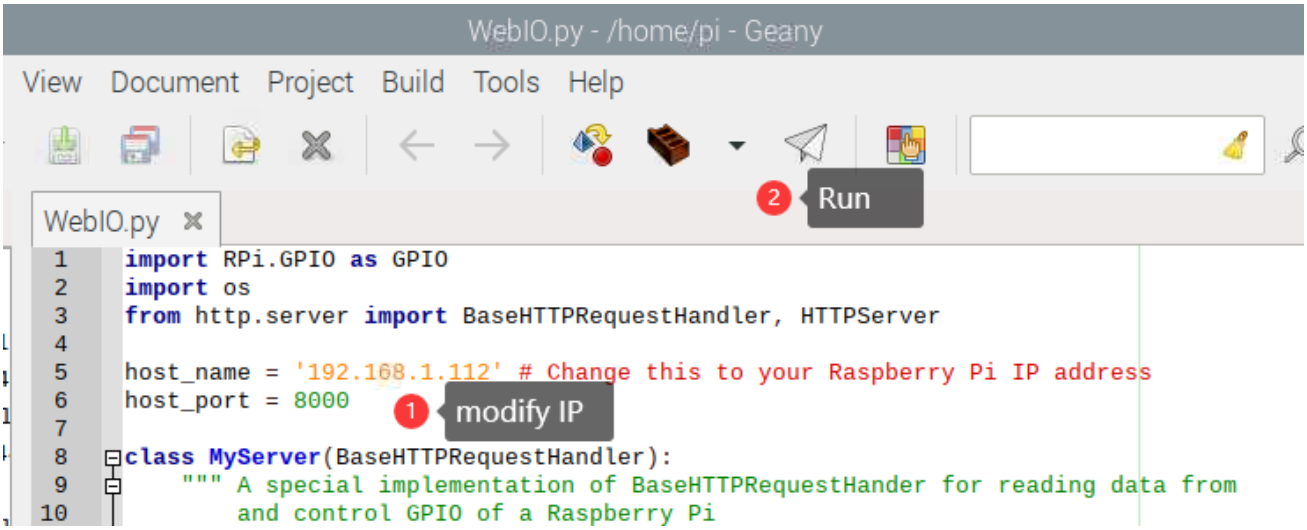
2, Open WebIO.py

```
cd ~/Freenove_Kit/Code/Python_Code/12.1.1_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 the 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
```

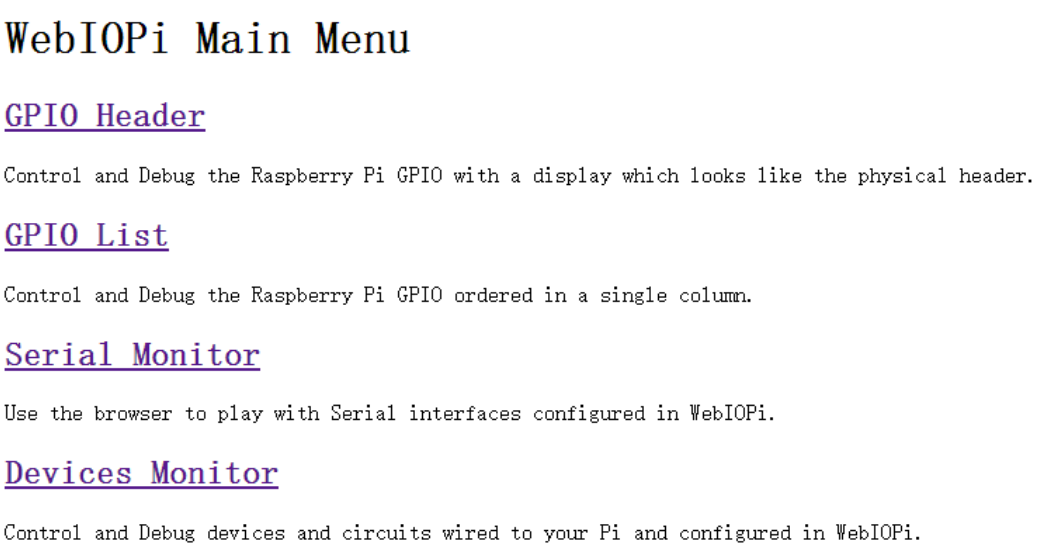
The Port is 8000 in default. Now WebIOPi has been launched. Keep it running.

Access WebIOPi over local network

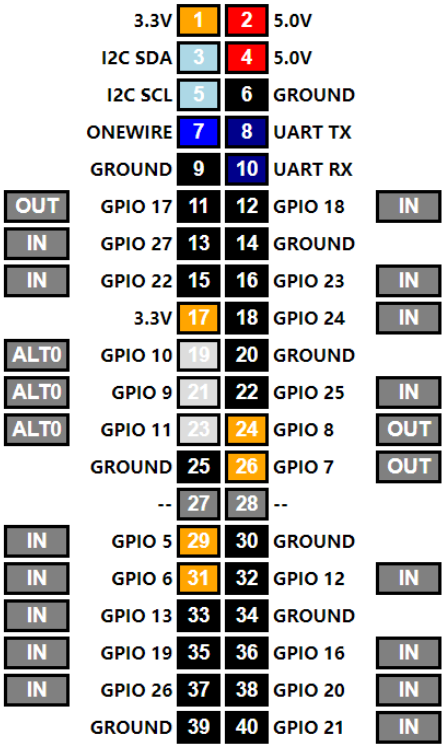
Under the same network, use a mobile phone or PC browser to open your RPi IP address, and add a port number like 8000. For example, my personal Raspberry Pi IP address is 192.168.1.109. Then, in the browser, I then should input: http://192.168.1.109:8000/

Default user is "webiopi" and password is "raspberry"。

Then, enter the main control interface:



Click on GPIO Header to enter the GPIO control interface.



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.