



FREENOVE

FREE YOUR INNOVATION

Freenove is an open-source electronics platform.

www.freenove.com

Freenove

Freenove is committed to provide high-quality products and services for customers, making it easy to get started with programing and electronics and launching innovative open source products. We value the user experience very much.

For more information, please refer to:

<http://www.freenove.com> and <http://www.freenove.com/store.html>

We offer product customization service. If you have any business matters, please feel free to contact us:
sale@freenove.com

References

You can download the sketches and references used in this product in the following websites:

<http://www.freenove.com> or <https://github.com/freenove>

Please pay more attention to two of the references:

- Read Me First.pdf It is recommended to read it first.
- Processing.pdf It is based on Java to create graphic interface to work with Raspberry Pi.

Support

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

- Quality problems of products
- Problems in using products
- Questions for learning and technology
- Opinions and suggestions
- Ideas and thoughts

Please send email to:

support@freenove.com

We will make things right for you. On working day, we usually reply to you within 24 hours.

Copyright

Freenove reserves all rights to this book. No copies or plagiarizations are allowed for the purpose of commercial use.

The code and circuit involved in this product are released as Creative Commons Attribution ShareAlike 3.0. This means you can use them on your own derived works, in part or completely, as long as you also adopt the same license. Freenove brand and Freenove logo are copyright of Freenove Creative Technology Co., Ltd and cannot be used without formal permission.

Contents

Contents	3
Preface	5
Raspberry Pi	6
Install the System.....	13
Component List	13
Optional Components.....	15
Raspbian System	17
Remote desktop & VNC	20
Chapter 0 Preparation	30
Linux Command	30
Install WiringPi	33
Obtain the Project Code.....	34
Python2 & Python3.....	35
Code Editor	37
GPIO	42
GPIO Extension Board	47
Breadboard Power Module	48
Next	49
Chapter 1 LED	50
Project 1.1 Blink	50
Chapter 2 Button & LED.....	60
Project 2.1 Button & LED.....	60
Project 2.2 MINI table lamp.....	66
Chapter 3 LEDBar Graph	72
Project 3.1 Flowing Water Light.....	72
Chapter 4 Analog & PWM.....	77
Project 4.1 Breathing LED	77
Chapter 5 RGBLED	84
Project 5.1 Colorful LED	84
Chapter 6 Buzzer	90
Project 6.1 Doorbell	90
Project 6.2 Alertor	96
(Important)Chapter 7 ADC	101
Project 7.1 Read the Voltage of Potentiometer.....	101
Chapter 8 Potentiometer & LED	116
Project 8.1 Soft Light.....	116
Chapter 9 Photoresistor & LED	123
Project 9.1 NightLamp.....	123
Chapter 10 Thermistor.....	131
Project 10.1 Thermometer	131
Chapter 11 LCD1602	139

Project 11.1 I2C LCD1602.....	139
Chapter 12 WebIOPi & IOT	149
Project 12.1 Remote LED.....	149
What's next?.....	154

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 a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games. For more information, you can refer to Raspberry Pi official [website](#). We will call it RPi, RPI, RasPi later in this tutorial.

In this tutorial, the content for most projects is 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 complete this tutorial, you can learn Java by reading Processing.pdf.

The supporting kit contains the accessory electronic components and modules needed to complete these projects. You can also use these components and modules to achieve your own creativity.

Additionally, if you have any difficulties or questions about this tutorial and the kit, you can always contact us for quick and free technical support: support@freenove.com

Raspberry Pi

So far, Raspberry Pi has developed to the fourth generation. Changes in versions are accompanied by increase and upgrades in hardware.

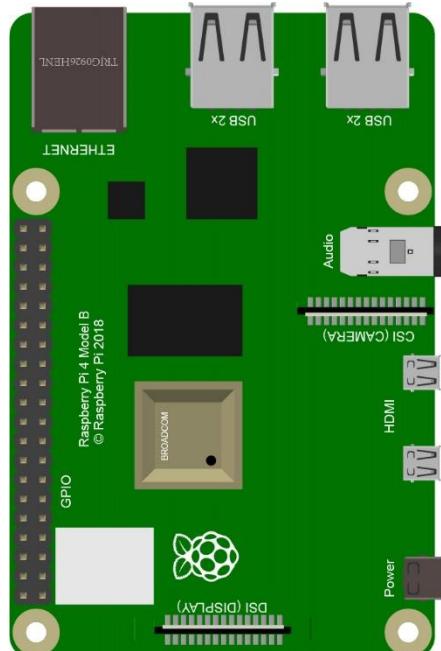
A type and B type, the first generation of products, have been stopped due to various reasons. Other versions are popular and active and the most important is that they are consistent in the order and number of pins, which makes the compatibility of peripheral devices greatly enhanced between different versions.

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

Practicality picture of Raspberry Pi 4 Model B:



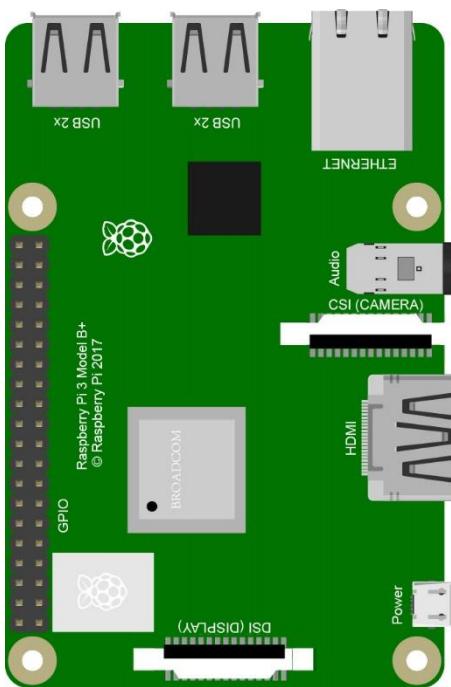
Model diagram of Raspberry Pi 4 Model B:



Practicality picture of Raspberry Pi 3 Model B+ :



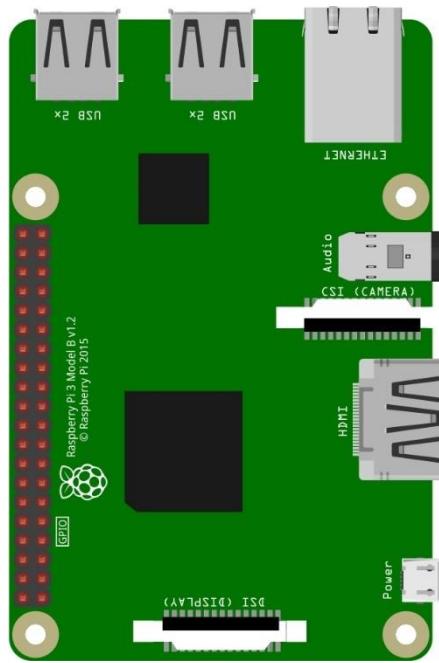
Model diagram of Raspberry Pi 3 Model B+ :



Practicality picture of Raspberry Pi 3 Model B :



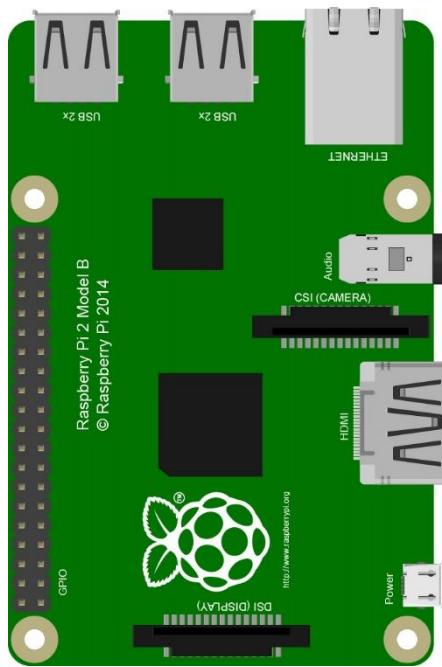
Model diagram of Raspberry Pi 3 Model B :



Practicality picture of Raspberry Pi 2 Model B:



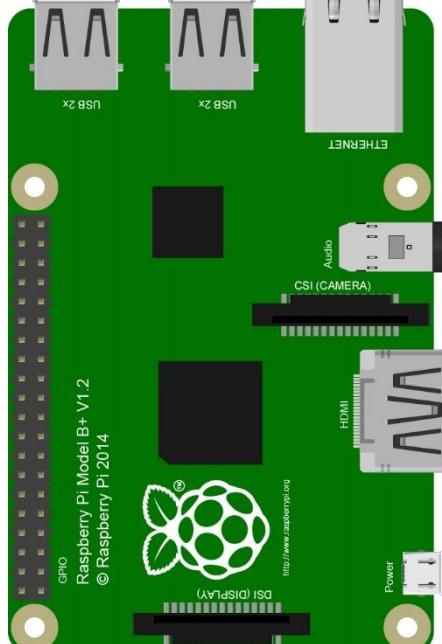
Model diagram of Raspberry Pi 2 Model B:



Practicality picture of Raspberry Pi 1 Model B+:



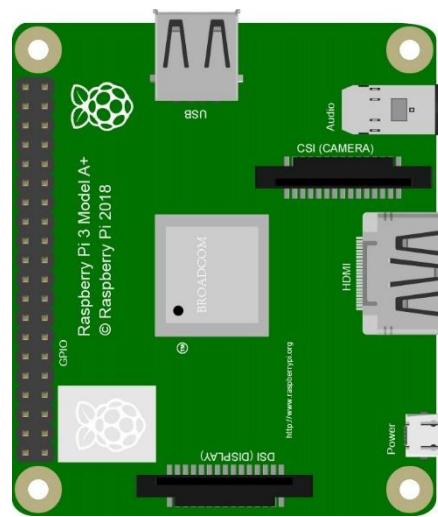
Model diagram of Raspberry Pi 1 Model B+:



Practicality picture of Raspberry Pi 3 Model A+:



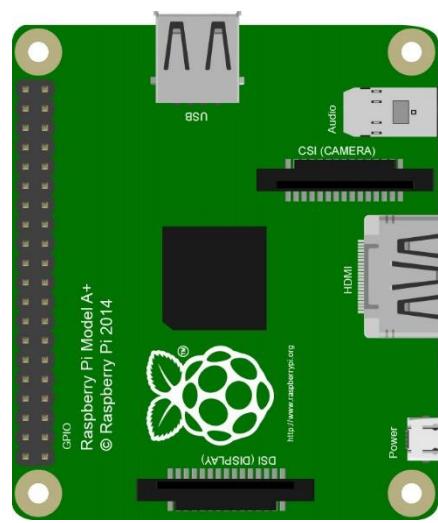
Model diagram of Raspberry Pi 3 Model A+:



Practicality picture of Raspberry Pi 1 Model A+:



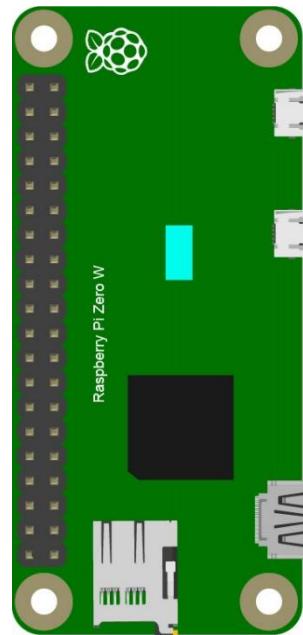
Model diagram of Raspberry Pi 1 Model A+:



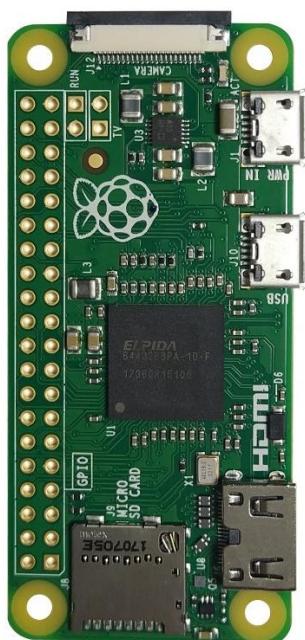
Practicality picture of Raspberry Pi Zero W:



Model diagram of Raspberry Pi Zero W:



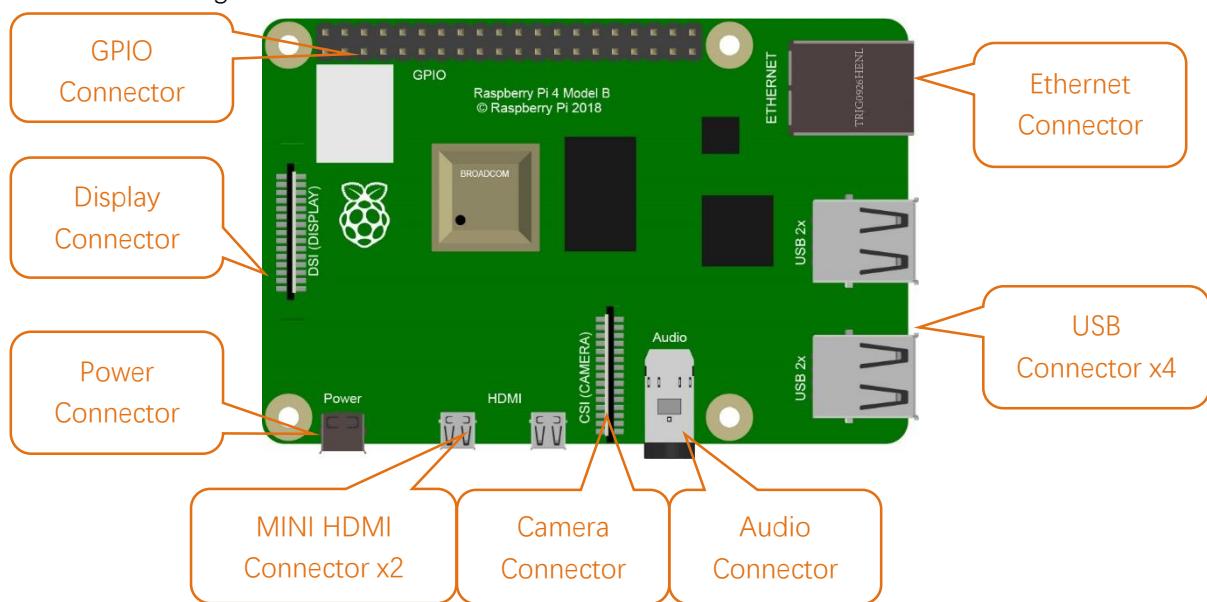
Practicality picture of Raspberry Pi Zero:



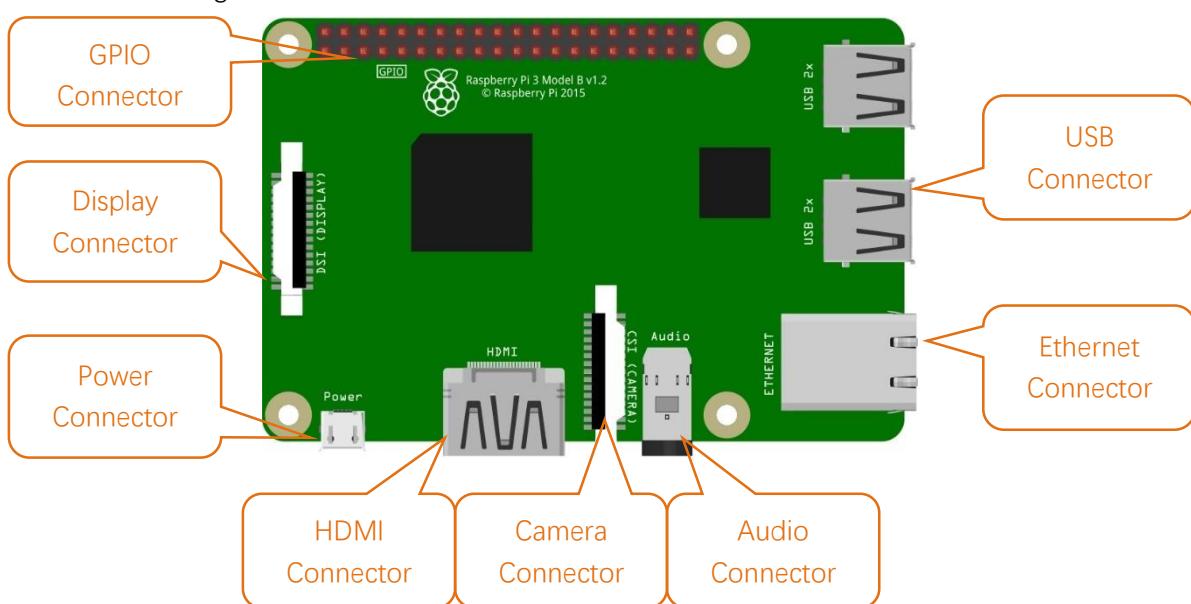
Model diagram of Raspberry Pi Zero:



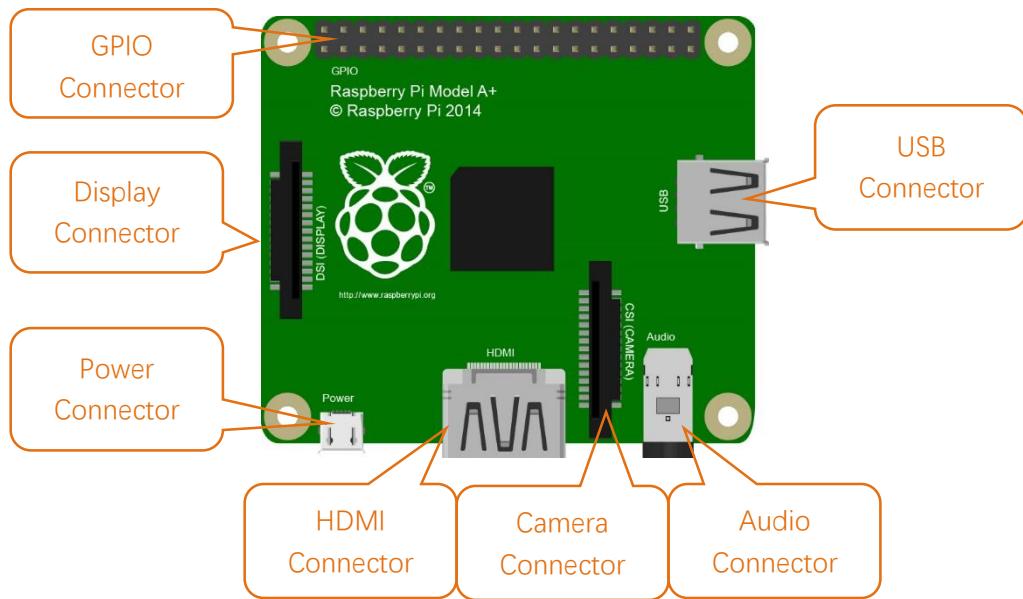
Hardware interface diagram of RPi 4B is shown below:



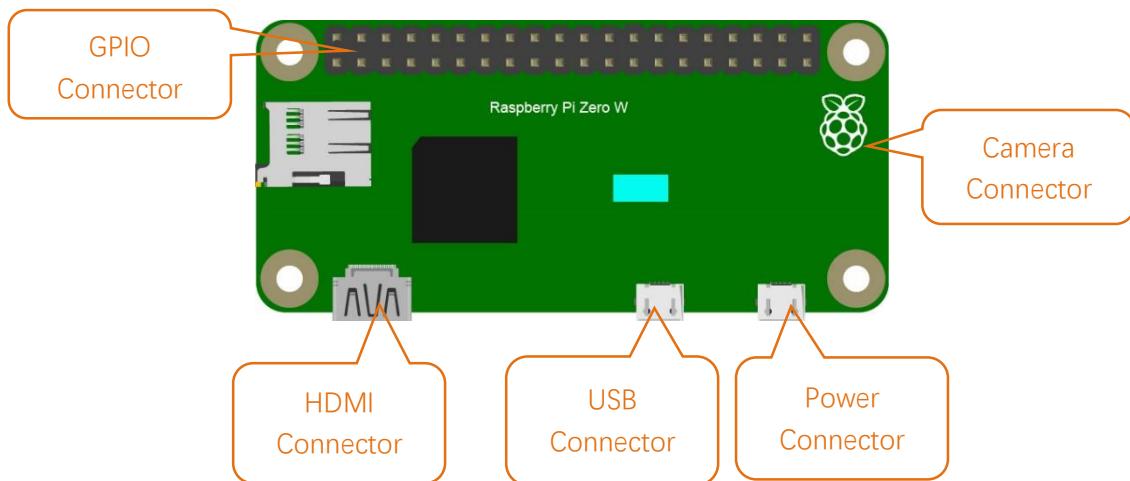
Hardware interface diagram of RPi 3B+/3B/2B/1B+ are shown below:



Hardware interface diagram of RPi 3A+/A+ is shown below:



Hardware interface diagram of RPi Zero/Zero W is shown below:

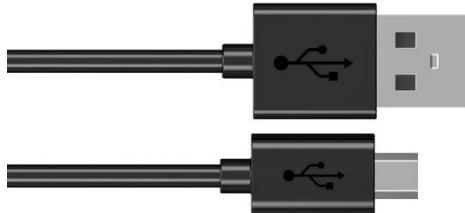


Install the System

Firstly, install a system for your RPi.

Component List

Required Components

Any Raspberry Pi	5V/3A Power Adapter. Different versions of Raspberry Pi have different power requirements. 
Micro or Type-C USB Cable x1	Micro SD Card(TF Card)x1, Card Reader x1  



Power requirement of different versions of Raspberry Pi is 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 a network cable used to connect it to wide area network.

All of these components are necessary. Among them, the power supply is required at least 5V/2.5A, because lack of power supply will lead to many abnormal problems, even damage to your RPi. So power supply with 5V/2.5A is highly recommend. SD Card Micro (recommended capacity 16GB or more) is a hard drive for RPi, which is used to store the system and personal files. In later projects, the components list with a RPi will contains these required components, using only RPi as a representative rather than presenting details.

Optional Components

Under normal circumstances, there are two ways to login to Raspberry Pi: using independent monitor, or remote desktop to share a monitor with your PC.

Required Accessories for Monitor

If you want to use independent monitor, mouse and keyboard, you also need the following accessories.

1. Display with HDMI interface
2. Mouse and Keyboard with USB interface

As to Pi Zero and Pi Zero W, you also need the following accessories.

1. Mini-HDMI to HDMI converter&wire.
2. Micro-USB to USB-A Receptacles converter&wire (Micro USB OTG wire).
3. USB HUB.
4. USB transferring to Ethernet interface or USB Wi-Fi receiver.

For different Raspberry Pi, the optional items are slightly different. But all of their aims are to convert the special interface to standard interface of standard Raspberry Pi.

	Pi Zero	Pi A+	Pi Zero W	Pi 3A+	Pi B+/2B	Pi 3B/3B+	Pi 4B
Monitor				Yes			
Mouse				Yes			
Keyboard				Yes			
Mini-HDMI to HDMI converter or wire	Yes	No	Yes	No	No	No	No
Micro-HDMI to HDMI converter & wire				No			Yes
Micro-USB to USB-A Receptacles converter & wire (Micro USB OTG wire)	Yes	No	Yes			No	
USB HUB	Yes	Yes	Yes	Yes	No	No	
USB transferring to Ethernet interface	select one from two or select two from two		optional		Internal Integration	Internal Integration	
USB Wi-Fi receiver			Internal Integration		optional		



Required Accessories for Remote Desktop

If you don't have an independent monitor, or you want to use a remote desktop, first you need to login to Raspberry Pi through SSH, then open the VNC or RDP service. So you need the following accessories.

	Pi Zero	Pi Zero W	Pi A+	Pi 3A+	Pi B+/2B	Pi 3B/3B+/4B
Micro-USB to USB-A Receptacles converter&wire (Micro USB OTG wire)	Yes	Yes	No			NO
USB transferring to Ethernet interface	Yes	Yes	Yes			

Raspbian System

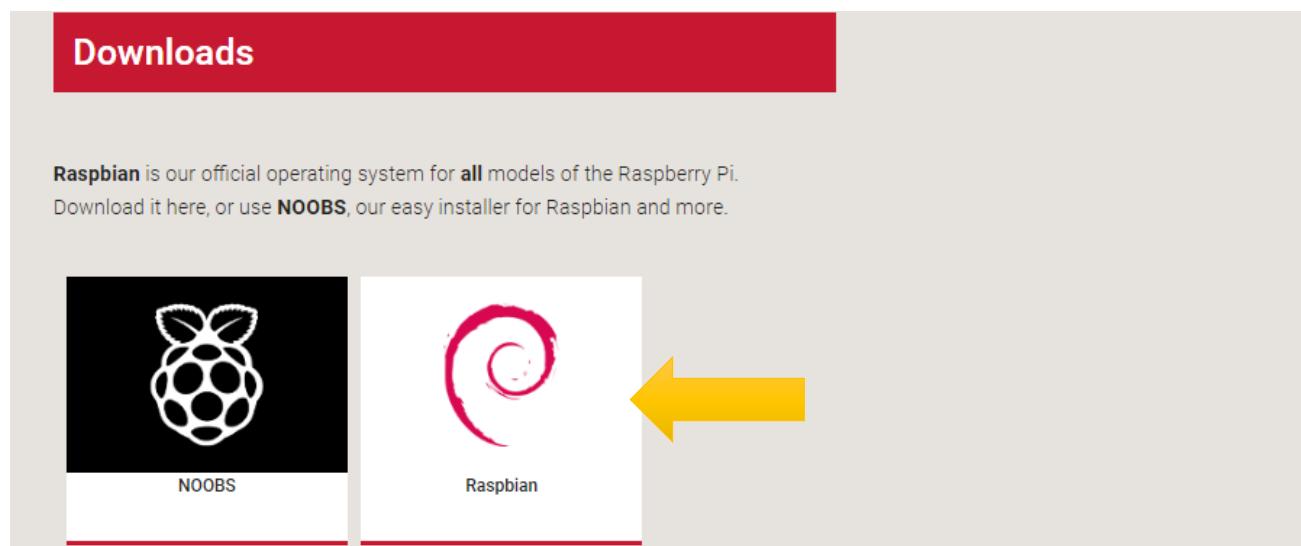
Tool and System image

Software Tool

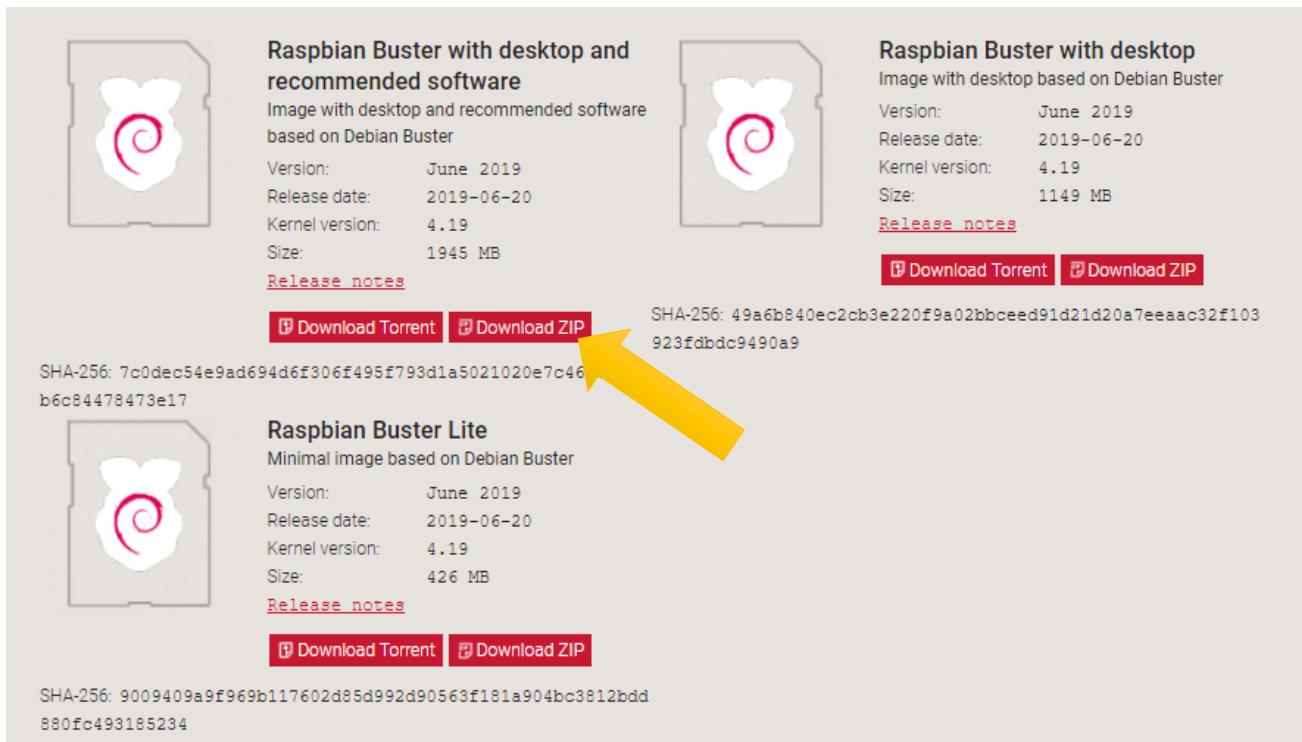
A tool Disk Imager Win32 is required to write system. You can download and install it through visiting the web site: <https://sourceforge.net/projects/win32diskimager/>

Selecting System

Visit RPi official website (<https://www.RaspberryPi.org/>), click “Downloads” and choose to download “RASPBIAN”. RASPBIAN supported by RPI is an operating system based on Linux, which contains a number of contents required for RPi. We recommended RASPBIAN system to beginners. All projects in this tutorial are operated under the RASPBIAN system.



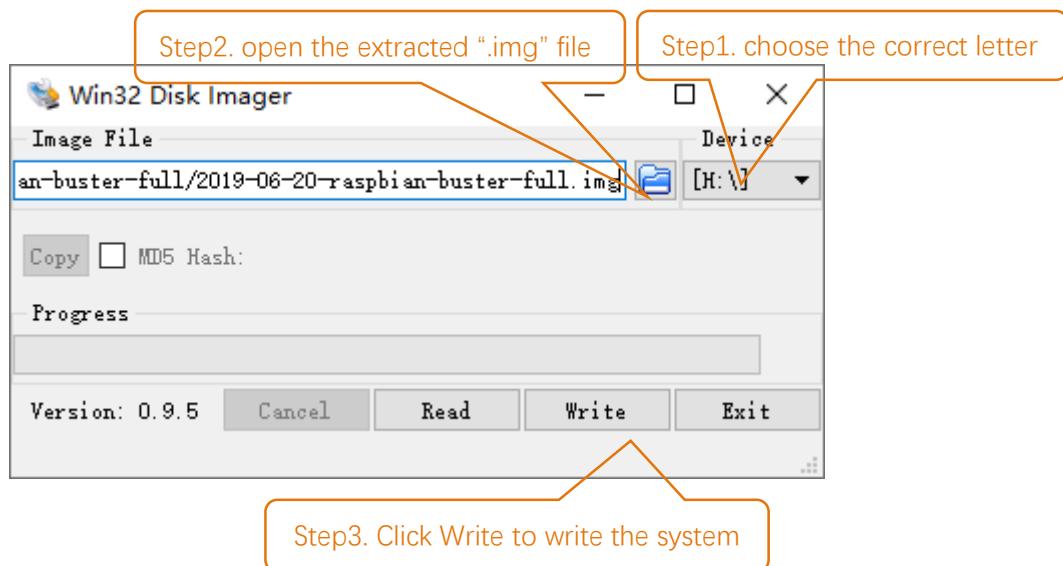
<https://www.raspberrypi.org/downloads/raspbian/>



After download, extract file with suffix (.img). Preparation is ready to start making the system.

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 Win32 disk imager, choose the correct letter of your Micro SD Card (here is "H"), open the extracted ".img" file and then click the "Write".



Start Raspberry Pi

If you don't have a spare monitor, please jumper to next section. If you have a spare monitor, please follow steps in this section.

After the system is written successfully, take out Micro SD Card and put it into the card slot of RPi. Then connect RPi to screen through the HDMI, to mouse and keyboard through the USB port, to network cable through the network card interface and to the power supply. Then your RPi starts initially. Later, you need to enter the user name and password to login. The default user name: pi; password: raspberry. Enter and login. After login, you can enter the following interface.



Now, you have successfully installed the RASPBIAN operating system for your RPi.



Remote desktop & VNC

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 remote desktop to control RPi under the Windows operating system.

Under windows, Raspberry Pi can be generally accessed remotely through two applications. The first one is the windows built-in application remote desktop, which corresponds to the Raspberry Pi xrdp service. The second one is the free application VNC Viewer, which corresponds to the VNC interface of Raspberry Pi. Each way has its own advantages. You can choose either one or two.

Windows	Raspberry Pi
Remote Desktop Connection	Xrdp
VNC Viewer	VNC

VNC Viewer can not only run under Windows, but also under system MAC, Linux, IOS, Android and so on.

SSH

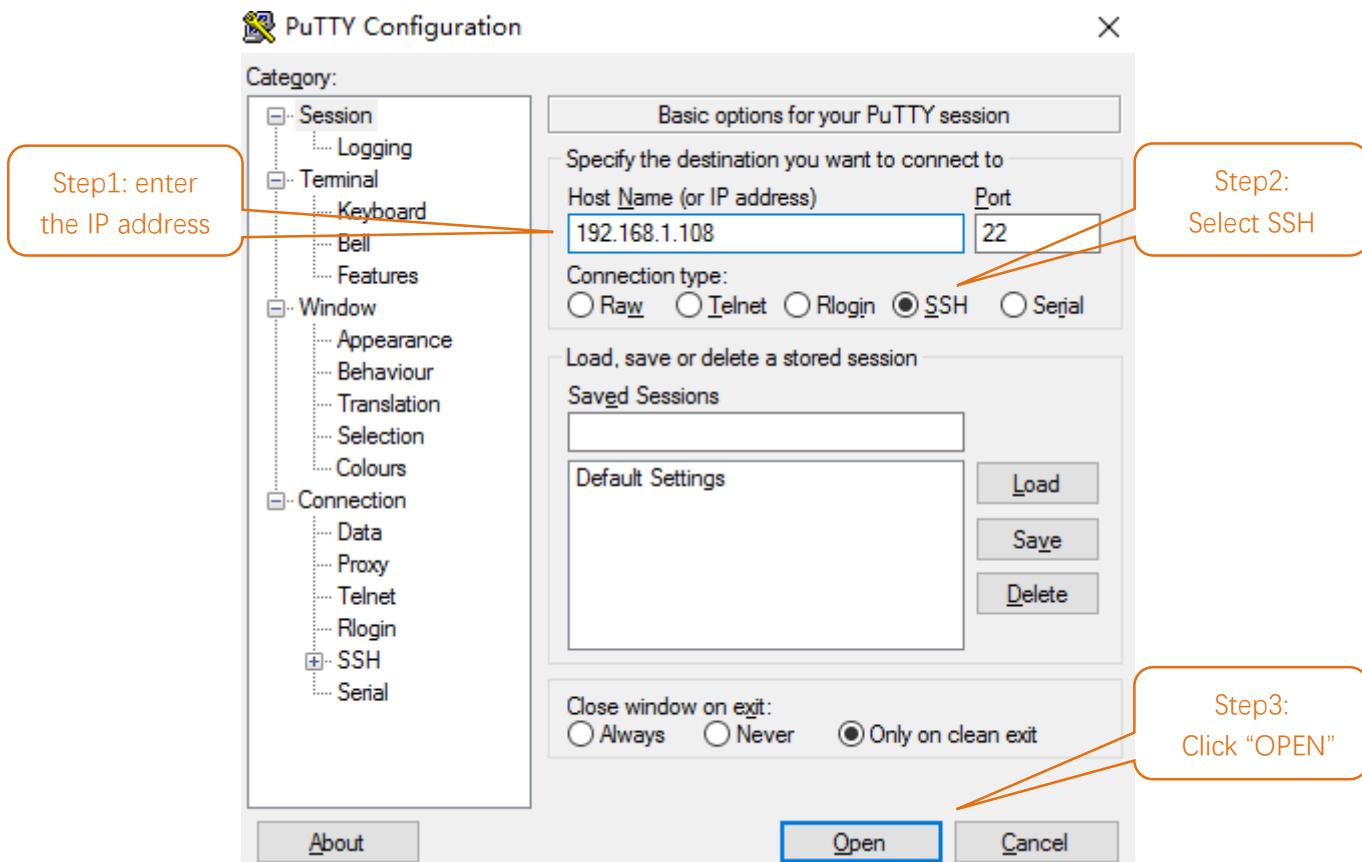
Under previous Raspbian system, SSH is opened by default. Under the latest version of Raspbian system, it is closed by default. So you need to open it first.

Method: after the system is written. Create a folder named “ssh” under generated boot disk, then the SSH connection will be opened.

And then, 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 cable to connect your RPi to the routers of your PC LAN, to ensure your PC and your RPi in the same LAN. Then put the system Micro SD Card prepared before into the slot of the RPi and turn on the power supply waiting for starting RPi. Later, enter control terminal of the router to inquiry IP address named "raspberry pi". For example, I have inquired to my RPi IP address, and it 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".

PuTTY Security Alert

WARNING - POTENTIAL SECURITY BREACH!

The server's host key does not match the one PuTTY has cached in the registry. This means that either the server administrator has changed the host key, or you have actually connected to another computer pretending to be the server.

The new rsa2 key fingerprint is:
`ssh-rsa 2048 7a:e1:50:ba:dc:01:87:1b:a5:f9:d2:d4:12:d6:fe:ab`

If you were expecting this change and trust the new key, hit Yes to update PuTTY's cache and continue connecting.

If you want to carry on connecting but without updating the cache, hit No.

If you want to abandon the connection completely, hit Cancel. Hitting Cancel is the ONLY guaranteed safe choice.



Then there will be a login interface (RPi default user name: pi; the password: raspberry). When you enter the password, there will be no display on the screen. This is normal. After the correct output, press “Enter” to confirm.

```
PutTY (inactive)
login as:
```

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

```
pi@raspberrypi: ~
login as: pi
pi@192.168.1.108's password:

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 10 23:51:04 2016
pi@raspberrypi:~ $
```

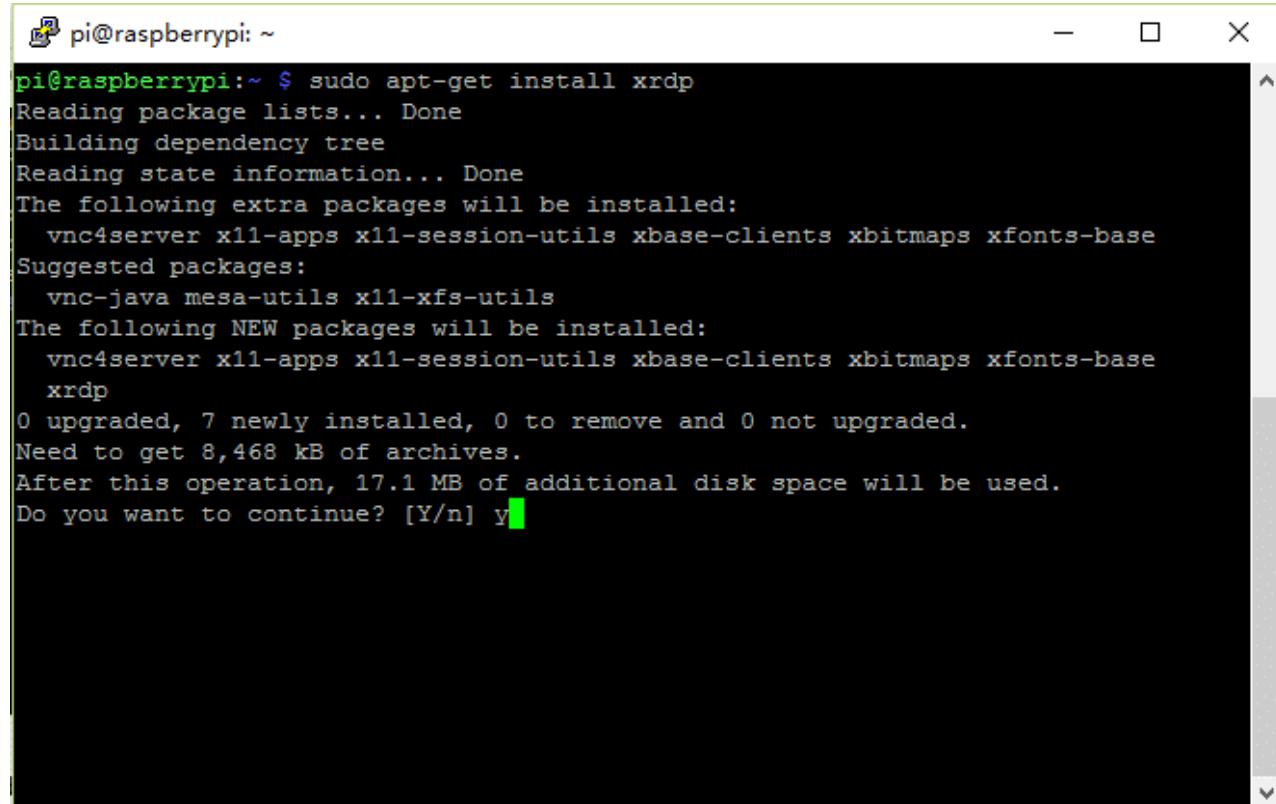
Remote Desktop Connection & xrdp

If you want to use built-in Remote Desktop Connection under Windows, you need install xrdp service on Raspberry Pi.

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

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

Later, the installation starts.



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the output of the command "sudo apt-get install xrdp". The text in the terminal includes package lists, dependency building, state information, extra packages to be installed (vnc4server, x11-apps, x11-session-utils, xbase-clients, xbitmaps, xfonts-base), suggested packages (vnc-java, mesa-utils, x11-xfs-utils), new packages to be installed (vnc4server, x11-apps, x11-session-utils, xbase-clients, xbitmaps, xfonts-base, xrdp), upgrade counts, archive sizes, disk space usage, and a prompt asking if the user wants to continue with "y".

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

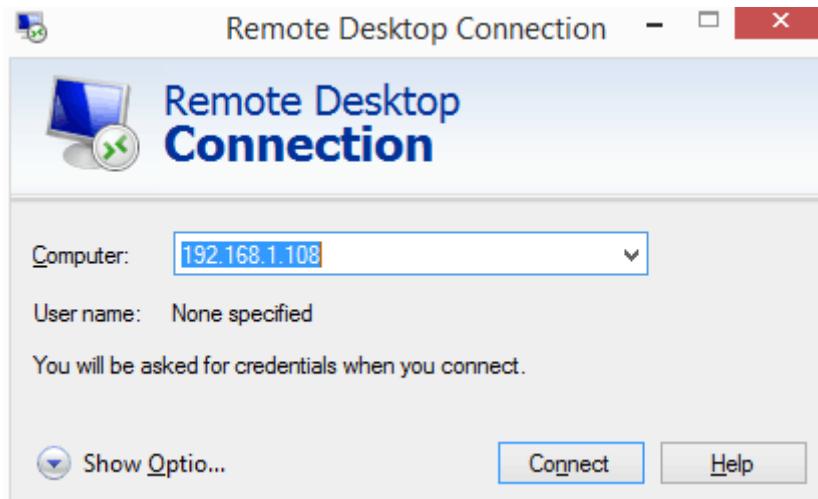
Enter "Y", press key "Enter" to confirm.

After the installation is completed, you can use Windows remote desktop applications to login to your RPi.

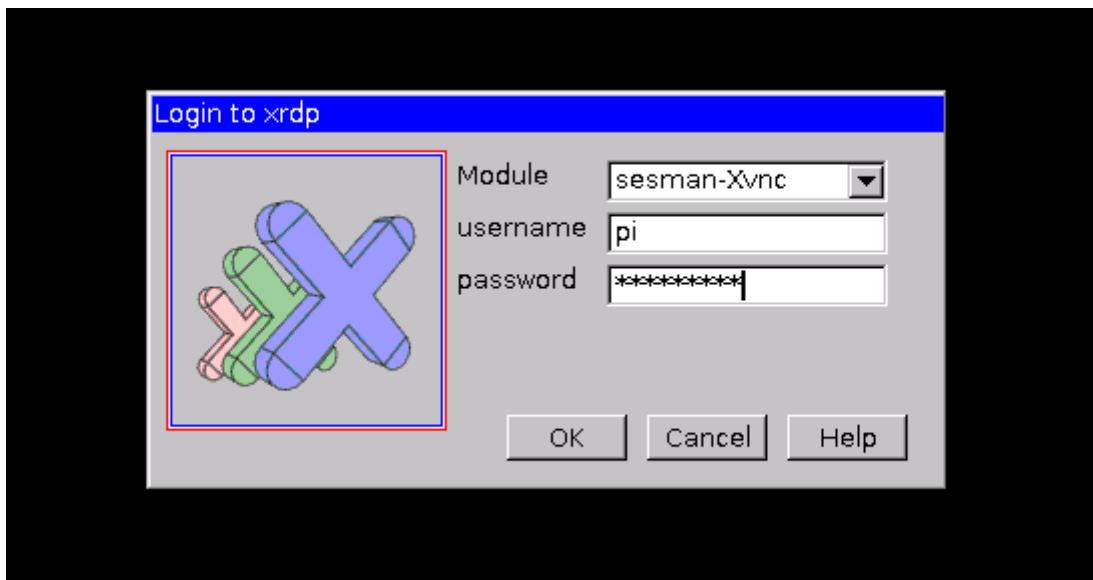


Login to Windows remote desktop

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.

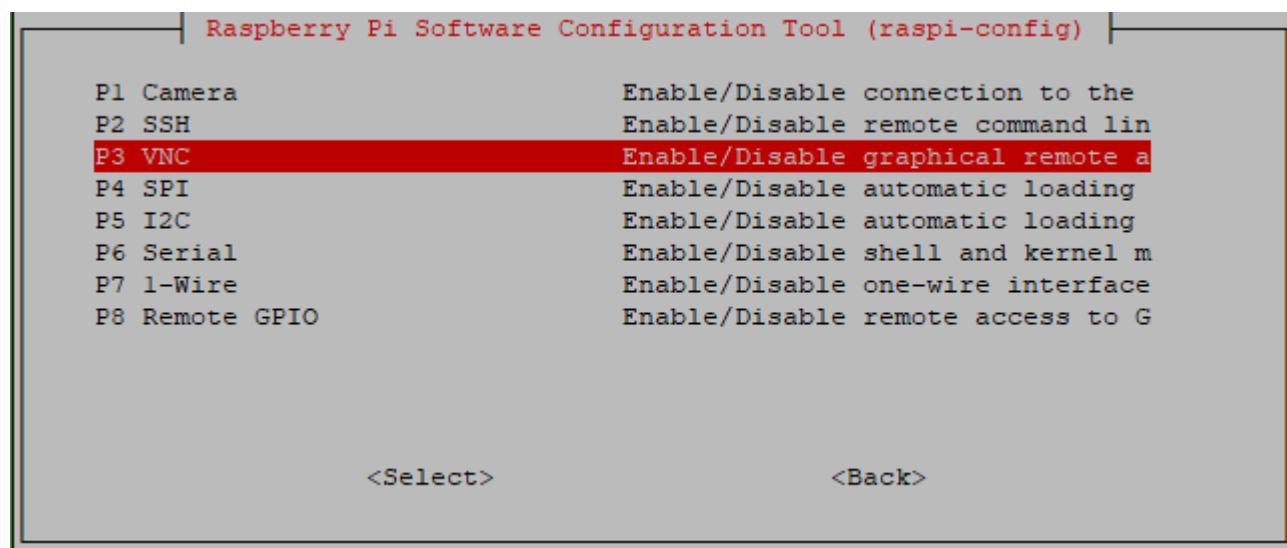
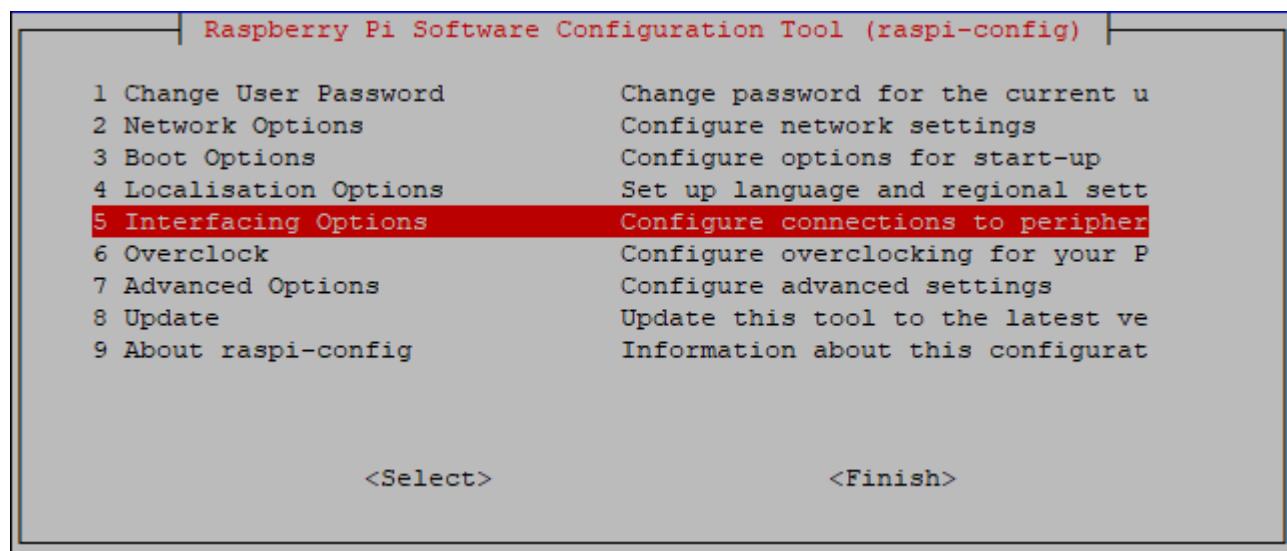


Here, you have successfully used the remote desktop login to RPi.

VNC Viewer & VNC

Type the following command. And select 5 Interfacing Options → P3 VNC → Yes → OK → Finish. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

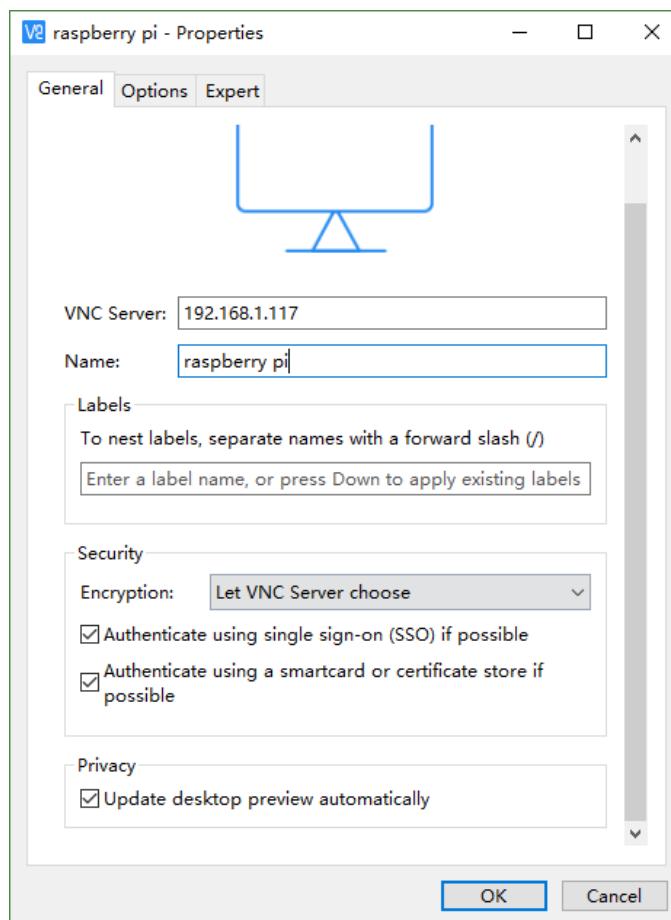
```
sudo raspi-config
```



Then download and install VNC Viewer by click following link:

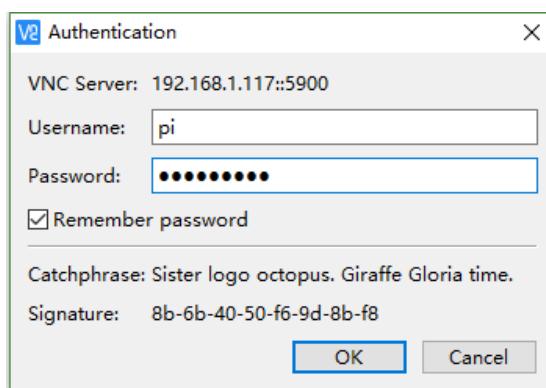
<https://www.realvnc.com/en/connect/download/viewer/windows/>

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. And 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.

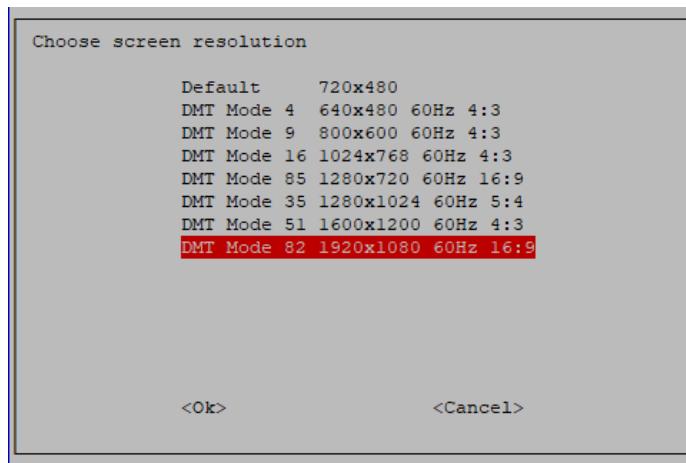


Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

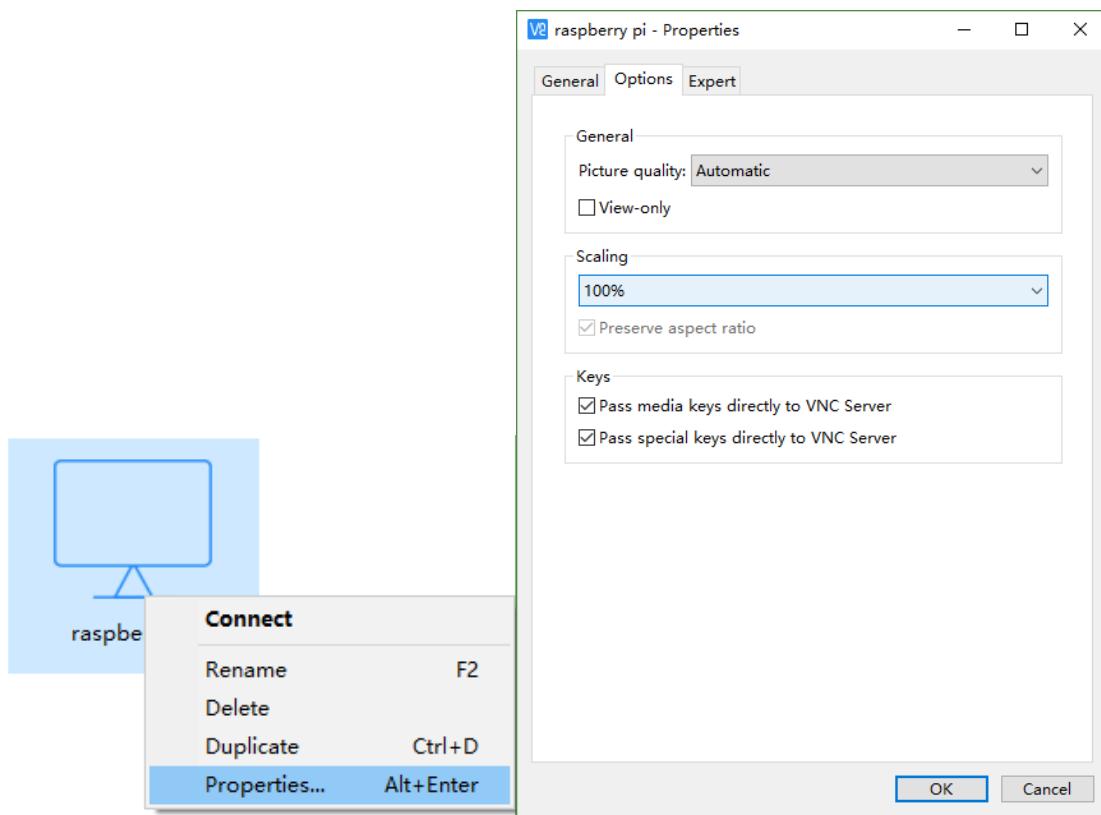
If you think resolution ratio is not OK, you can set a proper resolution ratio on set interface of Raspberry Pi.

`sudo raspi-config`

Select 7 Advanced Options → A5 Resolution → proper resolution ratio(set by yourself) → OK. If it needs restart, just restart.



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. Then continue to do some preparation work: install a GPIO library wiringPi for your RPi.

Wi-Fi

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.



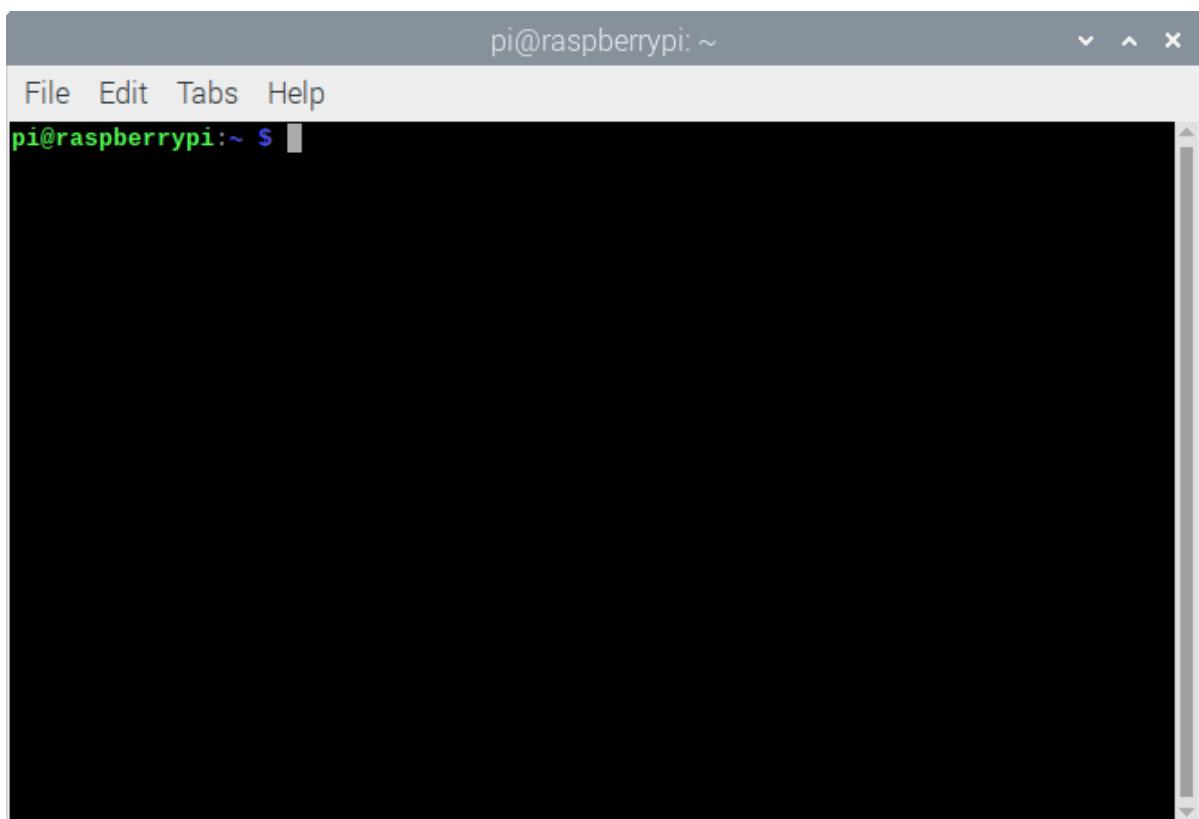
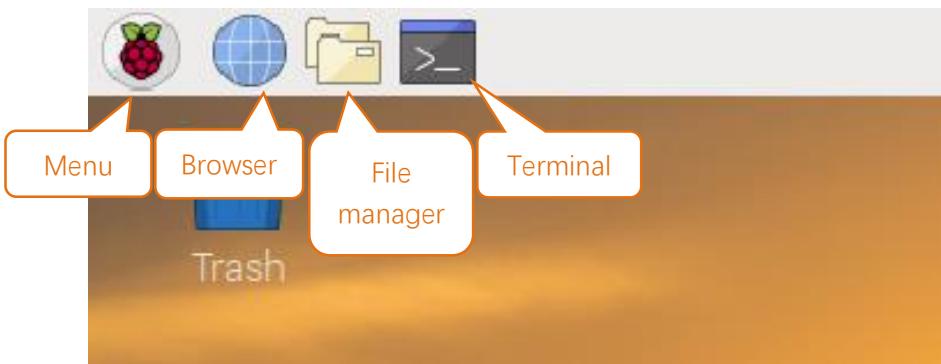
Chapter 0 Preparation

Why is “Chapter 0”? Because in the program code, all the counts are starting from 0. We choose to follow this rule (just a joke). In this chapter, we will do some necessary preparation work: start your Pi Raspberry and install some necessary libraries. If your Raspberry Pi can be started normally and used normally, you can skip this chapter.

Linux Command

Raspbian is based on Linux operation system. Now we will introduce some frequently-used Linux commands and usage methods.

First, open terminal. All commands are executed in terminal.



And the terminal is case sensitive.

Then type "ls" in terminal and press "Enter" key. The result is shown below:

```
pi@raspberrypi:~ $ ls
Desktop
Documents
Downloads
Freenove_Three-wheeled_Smart_Car_Kit_for_Raspberry_Pi
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi
MagPi
mu_code
Music
Pictures
Public
Templates
thinclient_drives
Videos
```

"ls", List information about the FILES (the current directory by default). Sort entries alphabetically.

Content between "\$" and "pi@raspberrypi:" is the current working path. "~" presents user directory. It is equal to "/home/pi" here. "pwd" can be used to view current working path.

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

"cd" is used to change directory. "/" presents 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:~ $
```

In later content, we will often change working path. Typing commands under wrong working path will cause error messages. And the commands can not continue to be executed.

There are some frequently used commands and instructions in following table.

Command	instruction
ls	List information about the FILES (the current directory by default). Sort entries alphabetically.
cd	Change directory
sudo + cmd	Excute 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 is the repository address.

There are many commands later. For more details about command. You can refer to:

<http://www.linux-commands-examples.com>



Shortcut Key

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

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

When you need to type command, pressing “↑” will bring back the previous command, and pressing “↓” will bring back the latter command.

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

As shown below, under the '~' directory, enter the Documents directory with the "cd" command. After typing "cd D", press Tab key, then there is no response. Press Tab key again, then all the files/folders that begin with "D" is listed. Continue to type the character "oc", then press the Tab key, and then "Documents" is completed 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 for the BCM2835/BMC2836/ BMC2837 used in the Raspberry Pi. It's released under the GNU LGPLv3 license and is usable from C, C++ and many other languages with suitable wrappers (See below) It's designed to be familiar to people who have used the Arduino "wiring" system. (for more details, please refer to <http://wiringpi.com/>)

WiringPi Installation Steps

New Raspbian system has integrated this library. So it may prompt that you have installed it. open the terminal. Follow these steps and commands to complete the installation.

For command with many lines, execute them line by line.

Enter the following command in the terminal to install WiringPi:

```
sudo apt-get update  
sudo apt-get upgrade  
sudo apt-get install wiringpi
```

As shown below, the installation will be completed soon.

```
pi@raspberrypi:~ $ sudo apt-get install wiringpi  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  freetype2-doc rpi.gpio-common  
Use 'sudo apt autoremove' to remove them.  
The following NEW packages will be installed:  
  wiringpi  
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.  
Need to get 0 B/52.9 kB of archives.  
After this operation, 0 B of additional disk space will be used.  
Selecting previously unselected package wiringpi.  
(Reading database ... 159071 files and directories currently installed.)  
Preparing to unpack .../wiringpi_2.50_armhf.deb ...  
Unpacking wiringpi (2.50) ...  
Setting up wiringpi (2.50) ...  
Processing triggers for man-db (2.8.5-2) ...
```

Run the gpio command to check the installation:

```
gpio -v
```

That should give you some confidence that it's working well.

```
pi@raspberrypi:~ $ gpio -v  
gpio version: 2.50  
Copyright (c) 2012-2018 Gordon Henderson  
This is free software with ABSOLUTELY NO WARRANTY.  
For details type: gpio -warranty  
  
Raspberry Pi Details:  
  Type: Unknown17, Revision: 01, Memory: 1024MB, Maker: Sony  
  * Device tree is enabled.  
  *--> Raspberry Pi 4 Model B Rev 1.1  
  * This Raspberry Pi supports user-level GPIO access.
```



Obtain the Project Code

After the above work is done, you can visit our official website (<http://www.freenove.com>) or our github (<https://github.com/freenove>) to download the latest project code. We provide both **C** language and **Python** language code for each project in order to apply to user skilled in different languages.

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_LCD_Starter_Kit_for_Raspberry_Pi
```

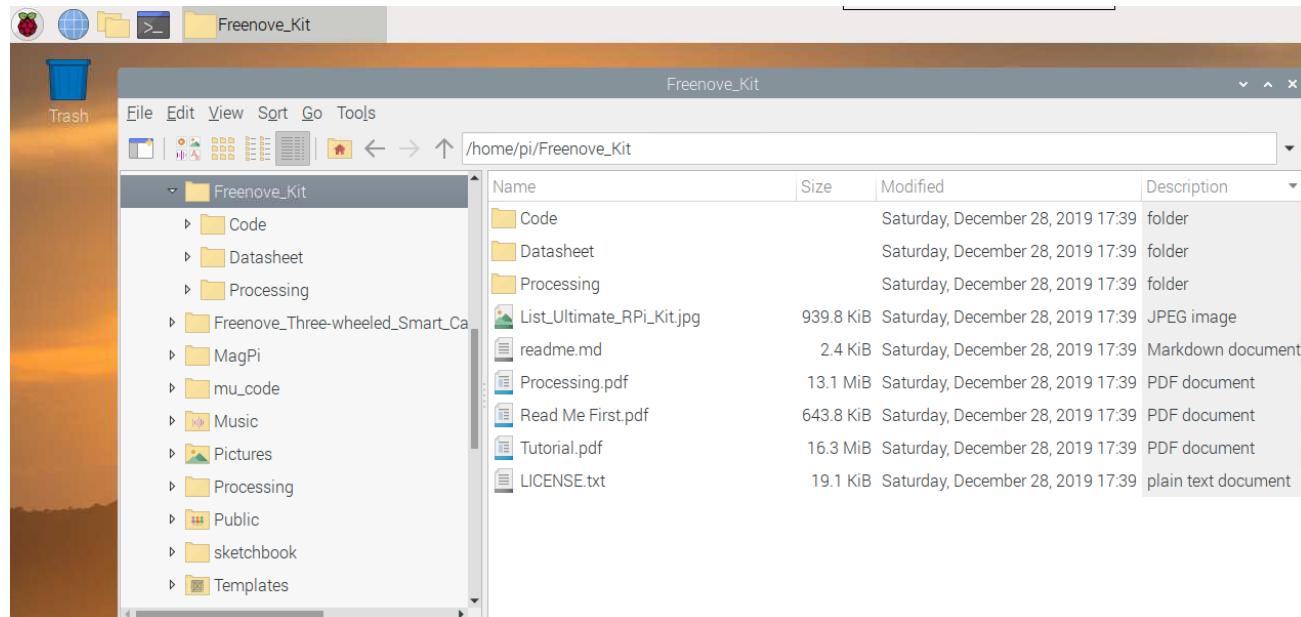
(There is no need for password. If you get some errors, please check if your commands is correct.)

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

The folder name seems a little too long. Here we rename it by following command.

```
mv Freenove_LCD_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

Among them, "Freenove_Kit" represents the new folder name.



If you never learned any things of python, it is recommended to refer to follow website for basic knowledge.
<https://python.swaroopch.com/basics.html>

Python2 & Python3

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

Now Python code of our kits can run on Python2 and Python3. **Python3 is recommend**. If you want to use python2, please make sure your Python version is above 2.7. Python2 and Python3 is not fully compatible. However, Python2.6 and Python2.7 are transition versions to python3. So you can also use Python2.6 and 2.7 to execute some Python3 code.

You can type python2 and python3 respectively 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 the 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 set Python3 as default Python actuators. please follow 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 set python2 as default python actuators, repeat above steps 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 execute a same python file Hello.py with Python2 and Python3.

First, use Python2 to execute the code.

1. Use cd command to enter 00.0.0_Hello directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/00.0.0_Hello
```

2. Use python2 command to execute python code Hello.py.

```
python2 Hello.py
```

```
pi@raspberrypi:~ $ cd Freenove_Kit/Code/Python_Code/00.0.0_Hello/
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/00.0.0_Hello $ python2 Hello.py
Hello World!
```

Use Python3 to execute the code under same directory.

3. Use python3 command to execute python code Hello.py.

```
python3 Hello.py
```

```
pi@raspberrypi:~ $ cd Freenove_Kit/Code/Python_Code/00.0.0_Hello/
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/00.0.0_Hello $ python3 Hello.py
Hello World!
```

As you can see, we get same results.

Because the code for our kit supports Python2 and Python3. We just say python later, not specific Python2 or Python3. You can choose python version according to your situation.

Code Editor

vi, nano, Geany

vi

Here we will introduce three kinds of code editor: vi, nano and Geany. Among them, nano and vi are used to edit files directly in the terminal. And Geany is an independent editing software, which is recommended for beginner. We will use the three editors to open an example code "Hello.c" respectively. First we will show how use vi and nano editor:

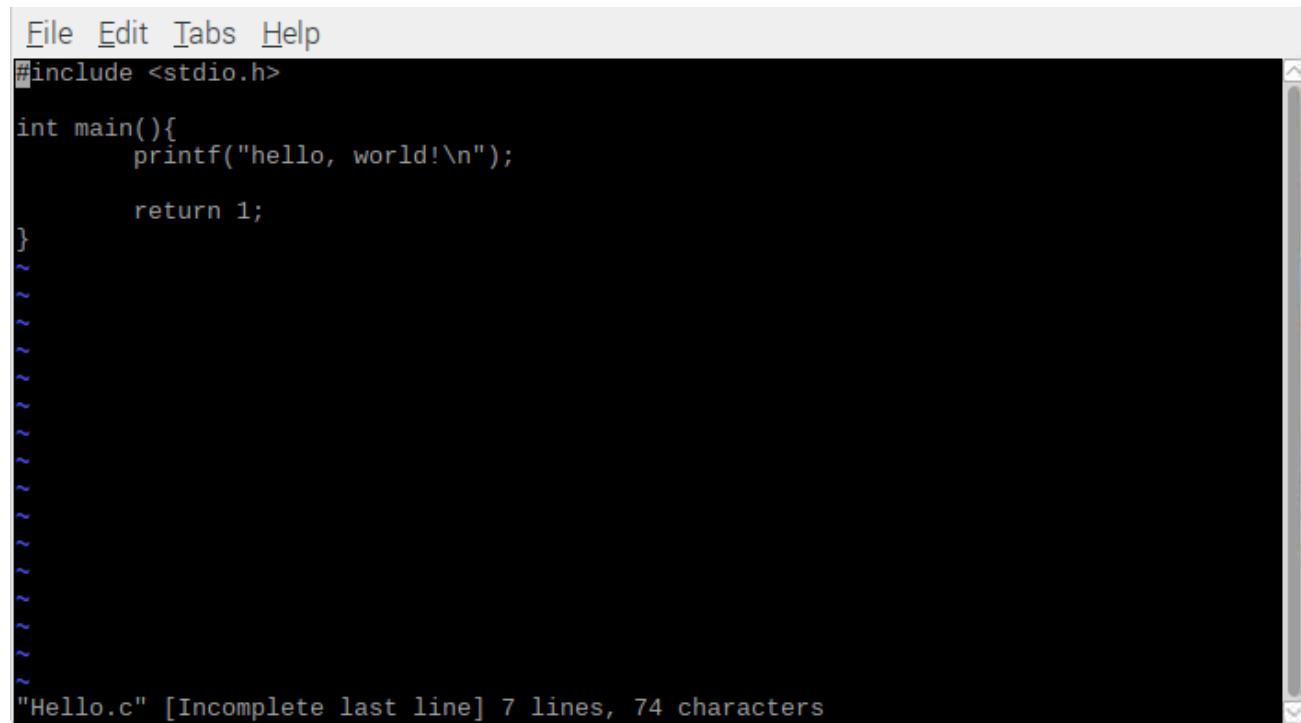
First, use cd command to enter the sample code folder.

```
cd ~  
cd ~/Freenove_Kit/Code/C_Code/00.0.0_Hello
```

Use the vi editor to open the file "Hello.c", then press ": q" and "Enter" to exit.

```
vi Hello.c
```

As is shown below:



The screenshot shows the vi editor interface with a black background. At the top, there is a menu bar with options: File, Edit, Tabs, and Help. Below the menu, the code for "Hello.c" is displayed. The code consists of a single main function that prints "hello, world!\n" and returns 1. The editor shows several cursor marks (~) along the left margin. At the bottom of the screen, the status bar displays the message: "'Hello.c' [Incomplete last line] 7 lines, 74 characters".

nano

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

```
nano Hello.c
```



As is shown below:

```
File Edit Tabs Help
GNU nano 2.7.4 File: Hello.c
#include <stdio.h>
int main(){
    printf("Hello, world!\n");
    return 1;
}

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit      ^R Read File ^V Replace ^U Uncut Text ^T To Spell ^L Go To Line
```

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.

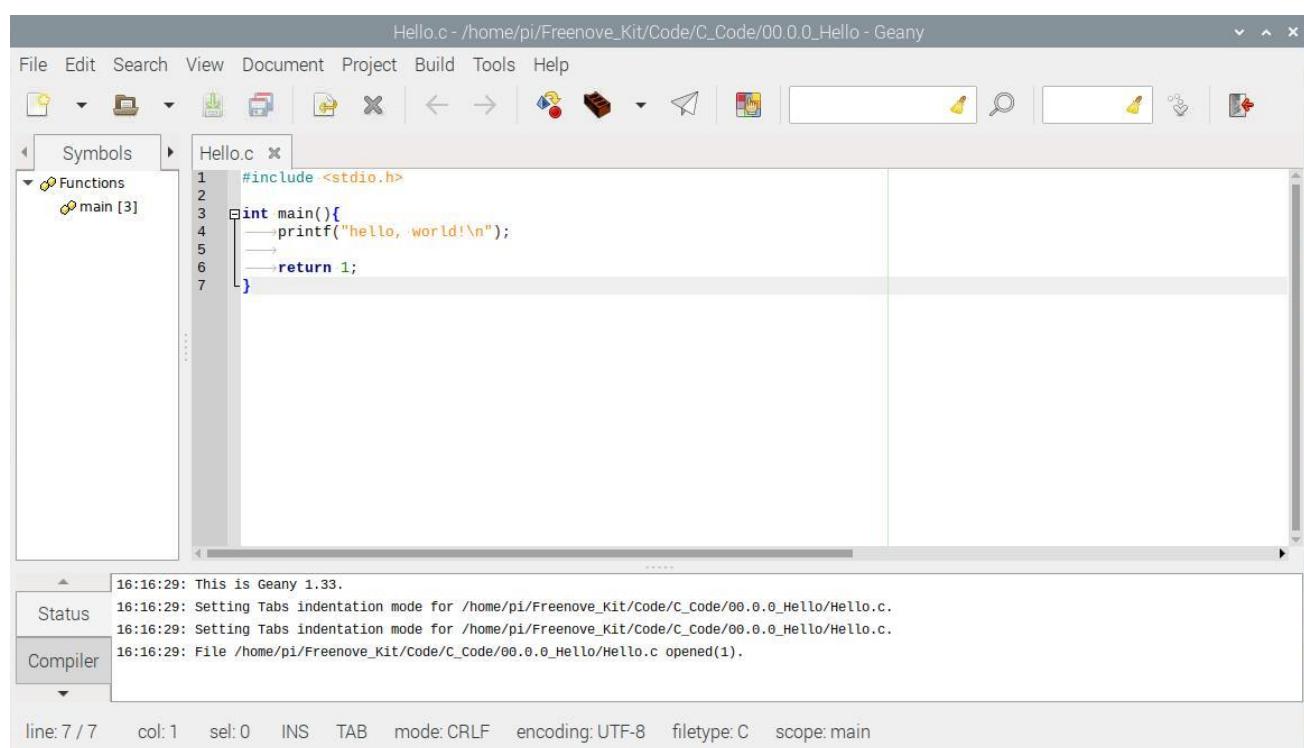
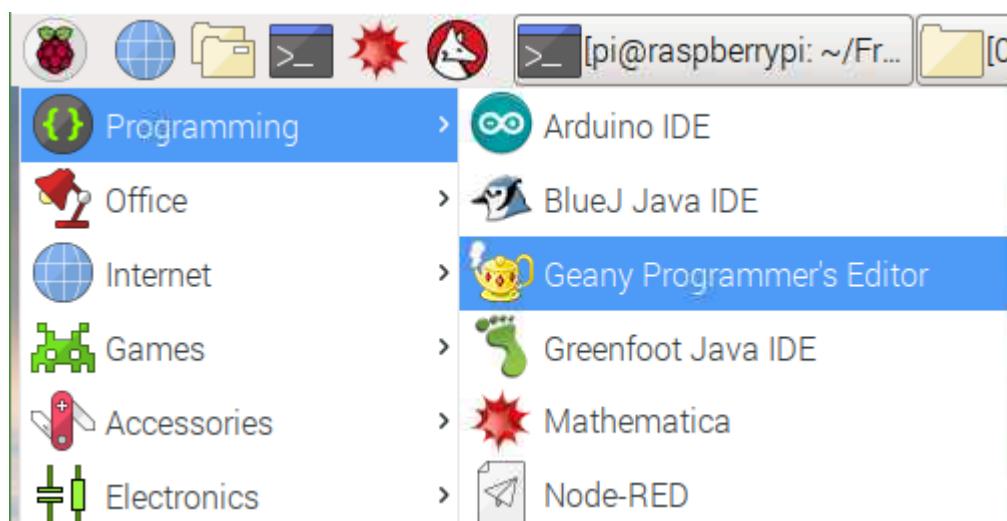
```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ gcc Hello.c -o Hello
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ sudo ./Hello
Hello, world!
```

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

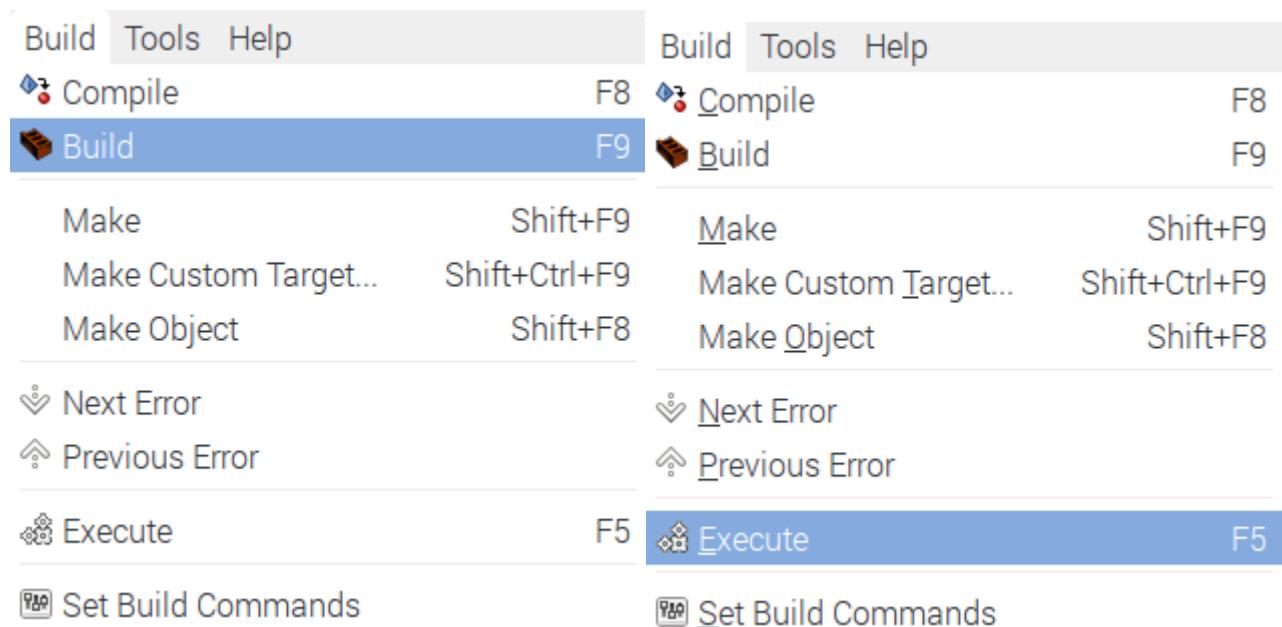
Or find and open Geany directly in the desktop main menu, and then click **File→Open** to open the "Hello.c", Or drag "Hello.c" to Geany directly.



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



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



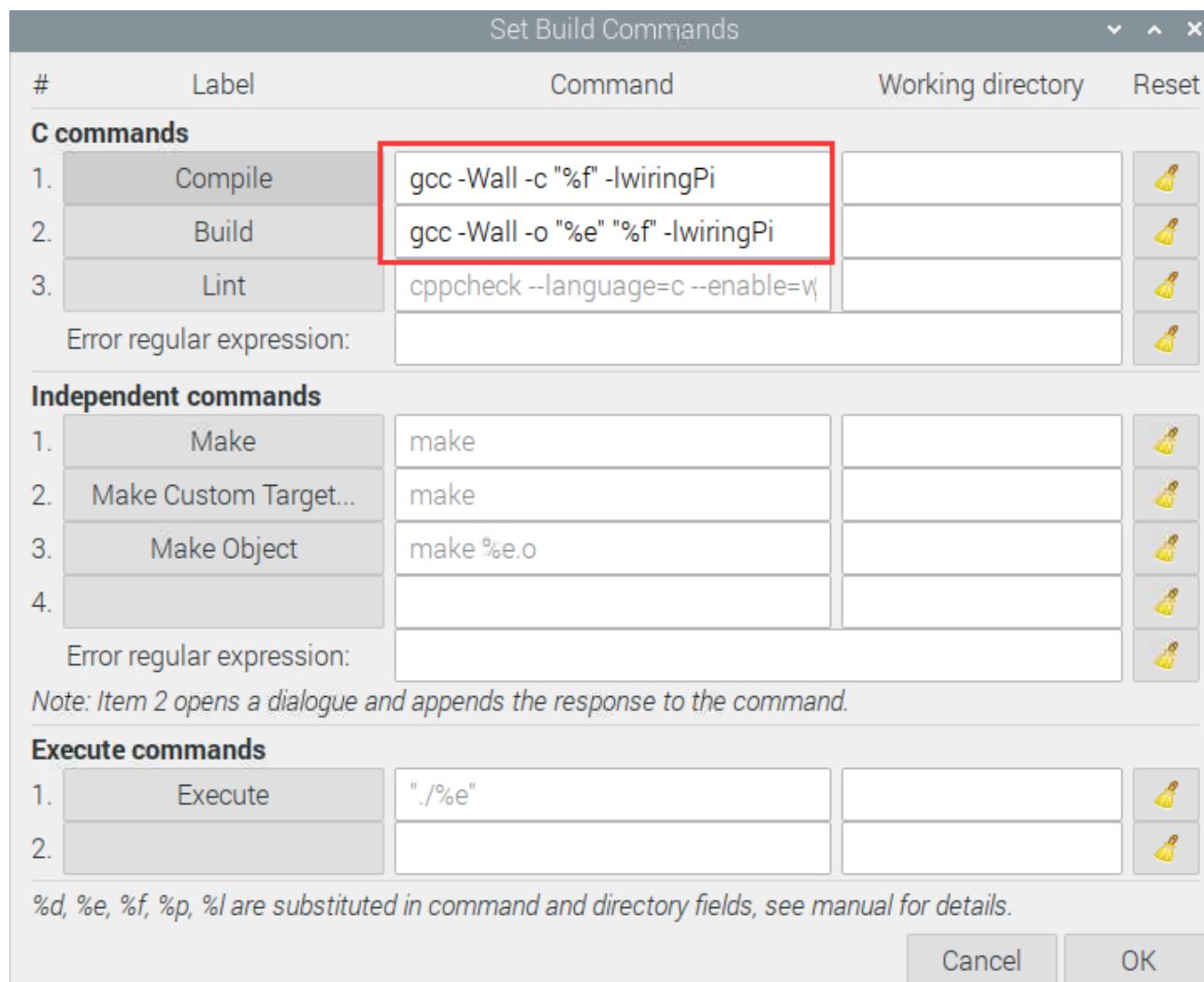
After the execution, there will be a terminal printing out the characters "Hello, World!", as shown below:

```
sh
File Edit Tabs Help
hello, world!

-----
(program exited with code: 1)
Press return to continue
```

A screenshot of a terminal window titled 'sh'. The window has a blue header bar with the title 'sh' and standard window control buttons (-, □, ×). The main area is black with white text. It displays the output of a program: 'hello, world!' followed by a dashed line, '(program exited with code: 1)', and the instruction 'Press return to continue'. A vertical scroll bar is visible on the right side of the terminal window.

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:



Summary

Here we have introduced three code editors. There also many other good code editors, and you can choose any one you like. In later projects, about the entry path and the compiler execute commands, we will operate the contents in the terminal as examples. We won't emphasize the code editing process, but will explain the contents of the code in details.



GPIO

GPIO: General purpose input/output. We will introduce the specific feature of the pins on the Raspberry Pi and what you can do with them. You can use them for all sorts of purposes. Most of them can be used as either inputs or outputs, depending on your program.

When programming the GPIO pins there are 3 different ways to refer to them: GPIO numbering, physical numbering, WiringPi GPIO Numbering.

BCM GPIO Numbering

Raspberry Pi CPU use BCM2835/BCM2836/BCM2837 of Broadcom. GPIO pin number is set by chip manufacturer. These are the GPIO pins as that computer recognizes. The numbers don't make any sense to humans. They jump all over the place, so there is no easy way to remember them. You will need a printed reference or a reference board that fits over the pins.

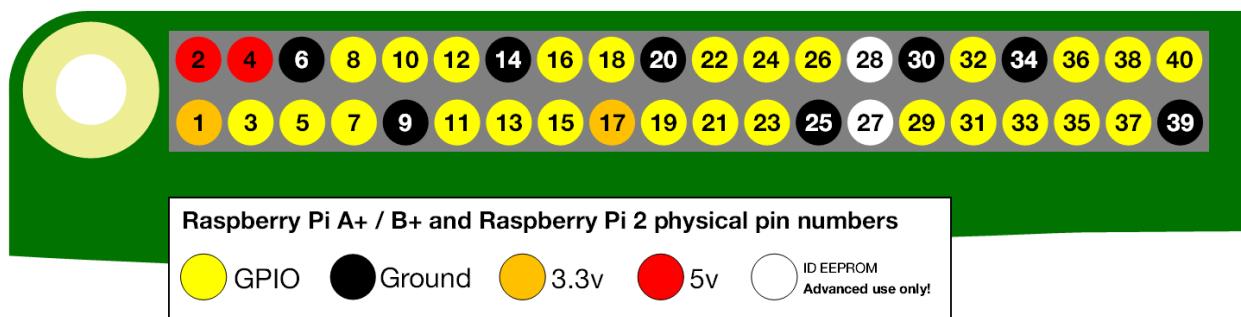
Each pin is defined as below:

Pin 1	Pin 2		Pin 39	Pin 40
+3V3		+5V		
GPIO2 / SDA1		+5V		
GPIO3 / SCL1		GND		
GPIO4		TXDO / GPIO 14		
		RXD0 / GPIO 15		
		GPIO 18		
		GND		
		GPIO 23		
		GPIO 24		
		GND		
		GPIO 25		
		CE0# / GPIO8		
		CE1# / GPIO7		
		ID_SD / GPIO1		
		GND		
		GPIO5		
		GPIO6		
		GPIO13		
		GND		
		CE2# / GPIO16		
		MOSI / GPIO20		
		SCLK / GPIO21		
		GND		

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 mentioned two kinds of GPIO serial numbers, RPi GPIO serial number of the WiringPi was renumbered. Here we have three kinds of GPIO number mode: based on the number of BCM chip, based on the physical sequence number and based on wiringPi. The correspondence between these three GPIO numbers is shown below:

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin	
—	—	3.3v	1 2	5v	—	—	For A+, B+, 2B, 3B, 3B+, 4B, Zero
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 more details, please refer to <https://projects.drogon.net/raspberry-pi/wiringpi/pins/>)

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

```
gpio readall
```

Pi 3 Model B											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1 2			5v			
2	8	SDA.1	ALTO	1	3 4			5V			
3	9	SCL.1	ALTO	1	5 6			0v			
4	7	GPIO. 7	IN	1	7 8	1	ALT5	TxD	15	14	
		0v			9 10	1	ALT5	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	ALTO	0	19 20			0v			
9	13	MISO	ALTO	0	21 22	0	IN	GPIO. 6	6	25	
11	14	SCLK	ALTO	0	23 24	1	OUT	CE0	10	8	
		0v			25 26	1	OUT	CE1	11	7	
0	30	SDA.0	IN	1	27 28	1	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	
Pi 3 Model B											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	

If you are using Raspberry Pi 4B, there will be errors when executing command “gpio readall”. As is below:

```
pi@raspberrypi:~ $ gpio readall
Oops - unable to determine board type... model: 17
```

This is because the official version of the library supporting 4B has not yet been released, resulting in some commands can not be used properly. But it won't affect the next project. For this problem, you can solve it by installing a patch. Just execute the commands below in the terminal.

```
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

After the installation is completed, Execute "gpio -v" and "gpio readall" commands again.

```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.52
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

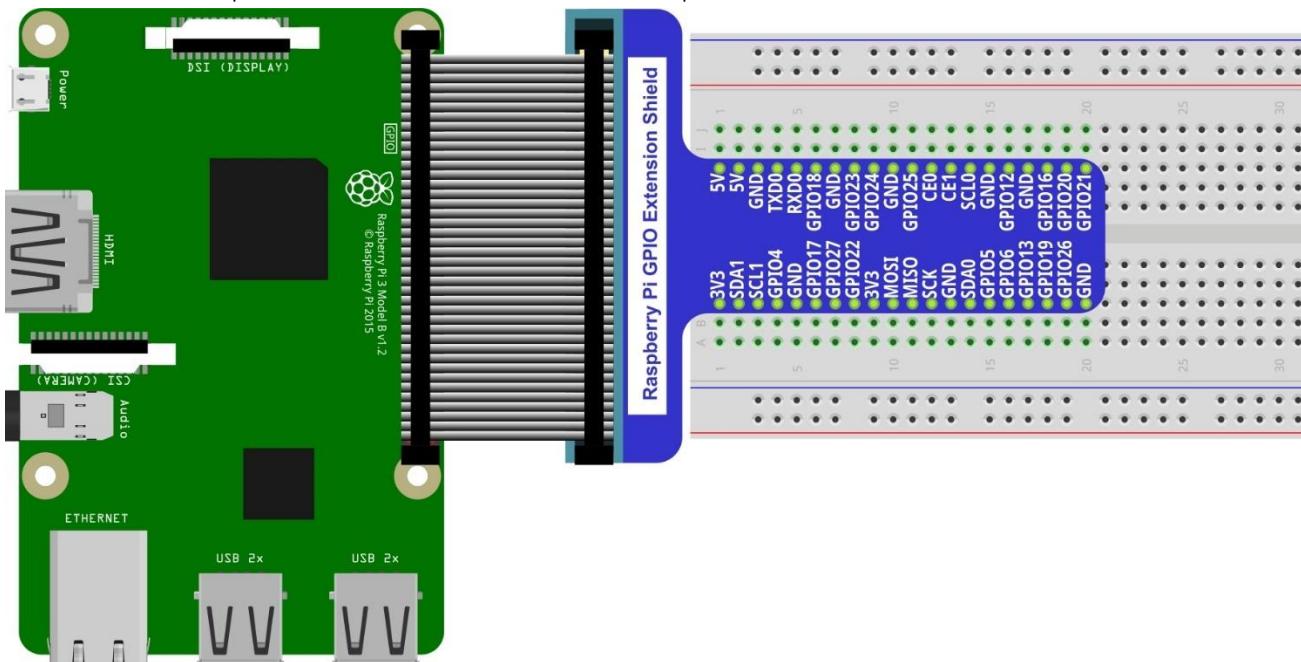
Raspberry Pi Details:
  Type: Pi 4B, Revision: 01, Memory: 1024MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 4 Model B Rev 1.1
  * This Raspberry Pi supports user-level GPIO access.

pi@raspberrypi:~ $ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 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 | IN   | TxD | 15  | 14  |
|      |      |          |      | 9 |    9 |   10 | 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  |
|      |      |          |      | 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   |
|      |      |          |      | 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 |
|      |      |          |      | 39 |   40 |   0  | IN  | GPIO.29 | 29 | 21 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |     Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

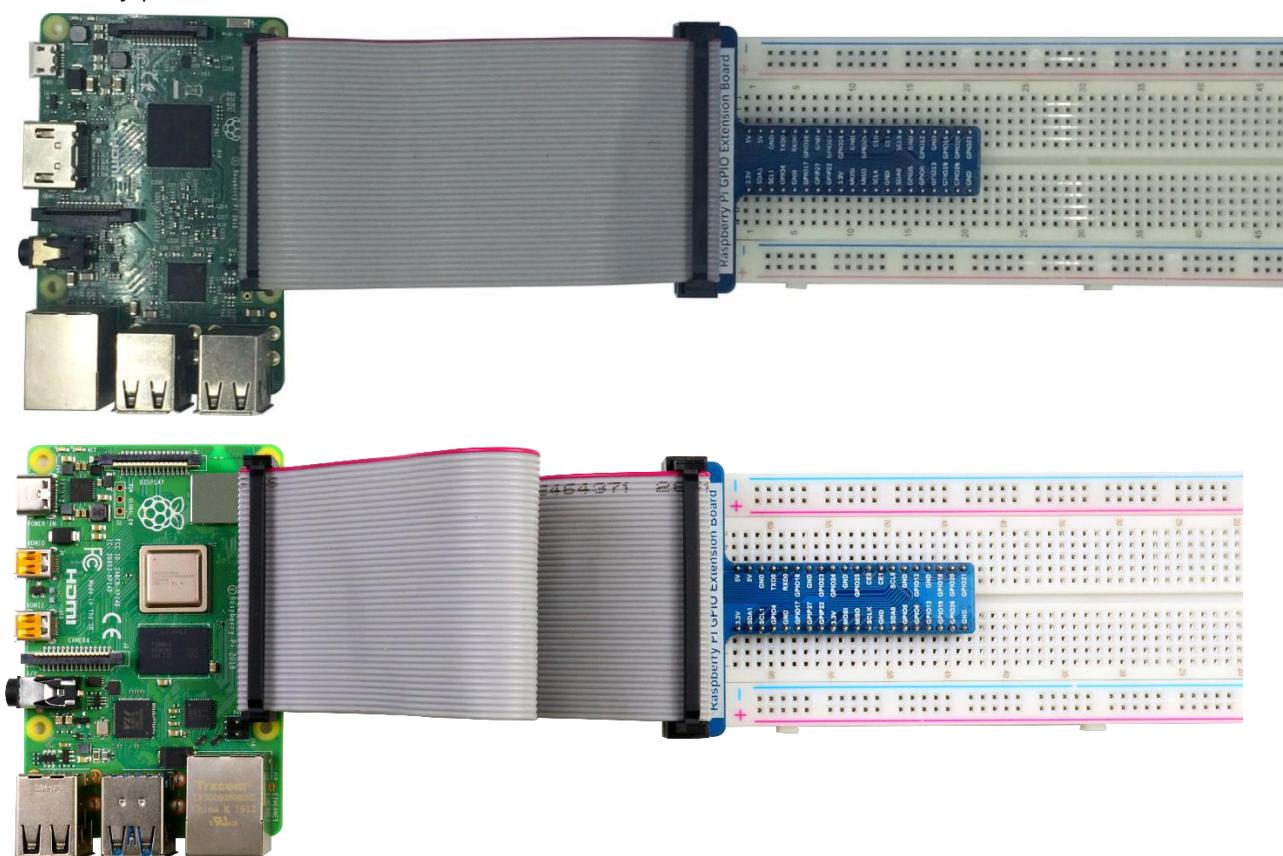
For more details about wiringPi, please refer to <http://wiringpi.com/>.

GPIO Extension Board

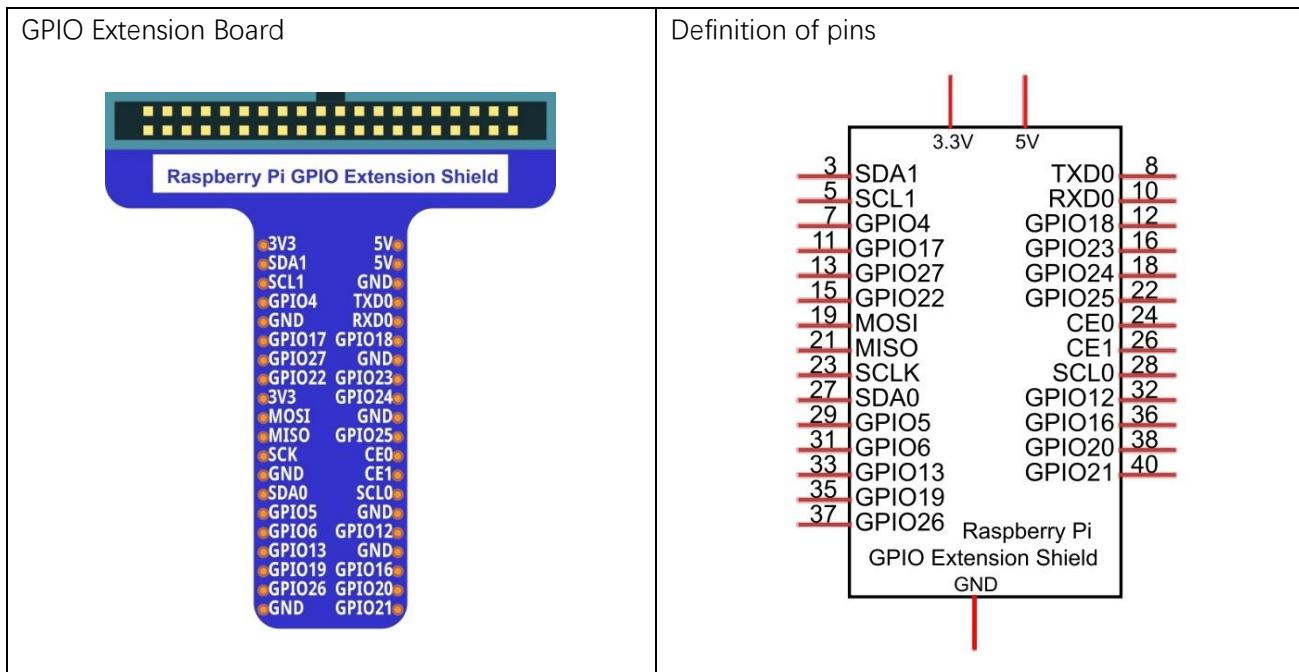
When we use RPi to do the project, we had better use GPIO, which is more convenient to extend all IO ports of RPi to the bread board directly. The GPIO sequence on Extension Board is identical to the GPIO sequence of RPi. Since the GPIO of different versions of RPi is different, the corresponding extensions board are also different. For example, a GPIO extensions board with 40 pins is connected to RPi as follows:



Practicality picture of connection:

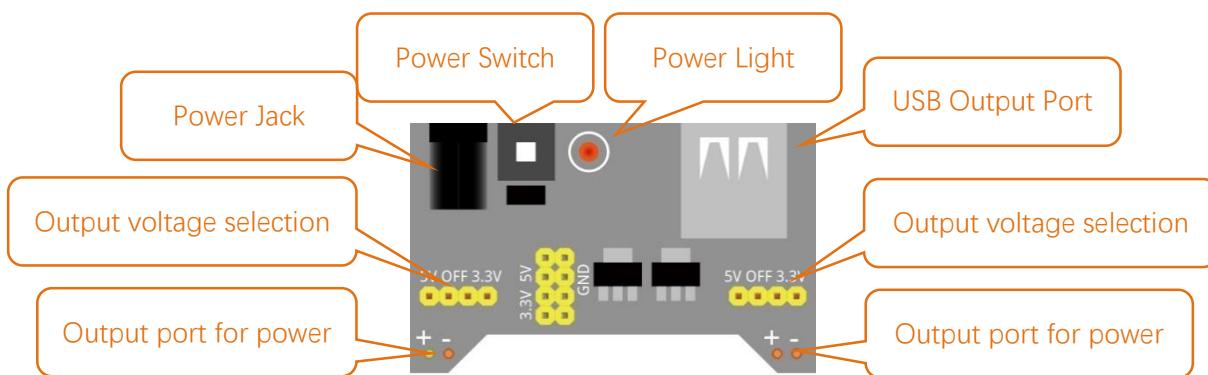


GPIO Extension Board and its schematic are shown below:

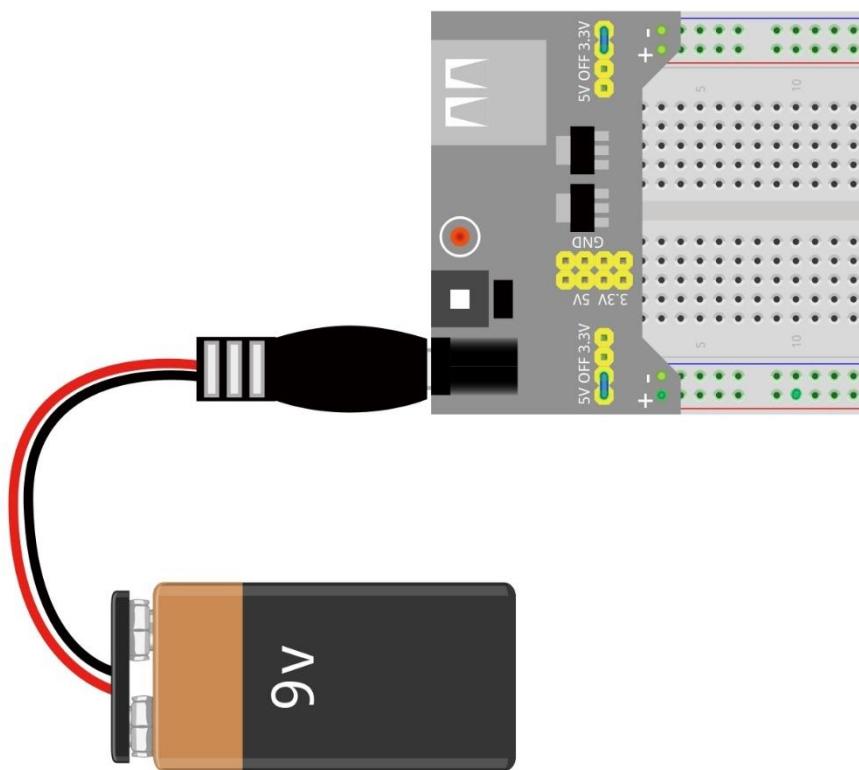


Breadboard Power Module

Breadboard Power Module is an independent board, which can provide independent 5V or 3.3V power for bread board when used to build the circuit, and it can avoid excessive load power damaging RPi power. The schematic diagram of the Breadboard Power Module is shown below:



The connection between Breadboard Power Module and Breadboard is shown below:



Next

Here, all preparations have been completed. Next, we will combine the RPi and electronic components to do a series of projects from easy to difficult and focus on explaining the relevant knowledge of electronic circuit.



Chapter 1 LED

This chapter is the starting point of the journey to explore RPi electronic projects. Let's start with simple "Blink".

Project 1.1 Blink

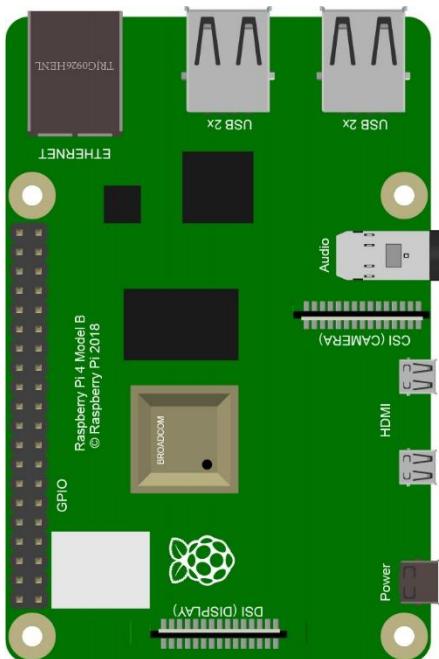
In this project, let's try to use RPi to control LED blinking.

Component List

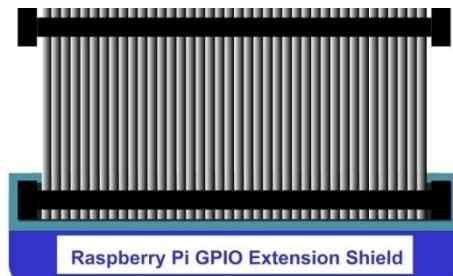
Raspberry Pi (

Recommended: Raspberry Pi 4B / 3B+ / 3B

Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)

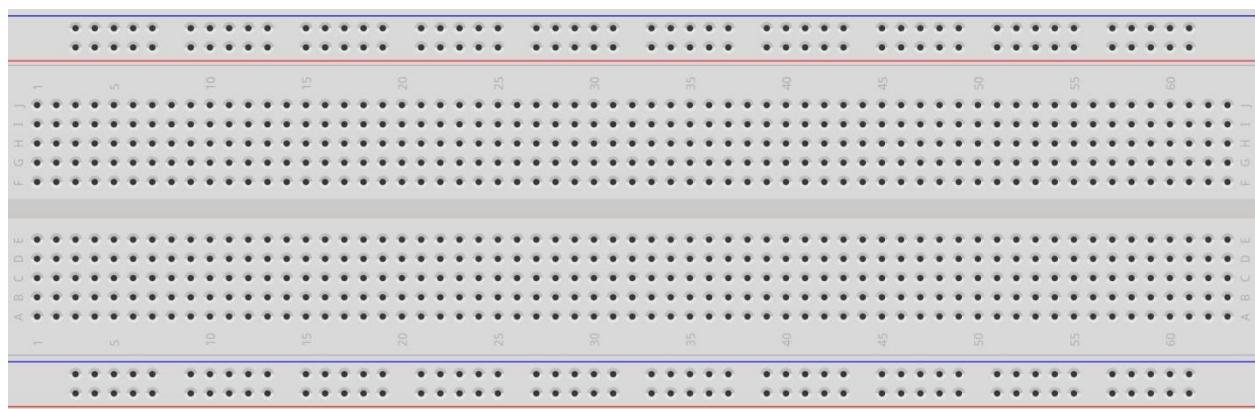


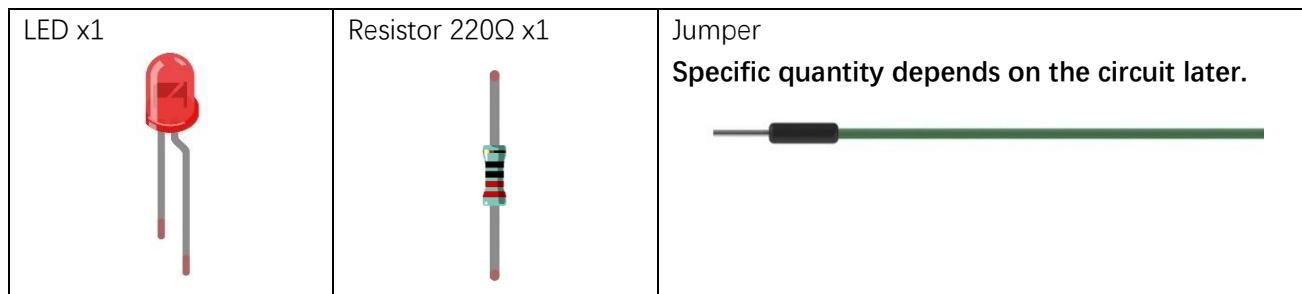
GPIO Extension Board & Wire x1



3V3	5V
SDA1	5V
SCL1	GND
GPIO4	TX0
GND	RX0
GPIO17	GPIO18
GPIO27	GND
GPIO22	GPIO23
3V3	GPIO24
MOSI	GND
MISO	GPIO25
SCK	CE0
GND	CE1
SDA0	SCL0
GPIO5	GND
GPIO6	GPIO12
GPIO13	GND
GPIO19	GPIO16
GPIO26	GPIO20
GND	GPIO21

BreadBoard x1





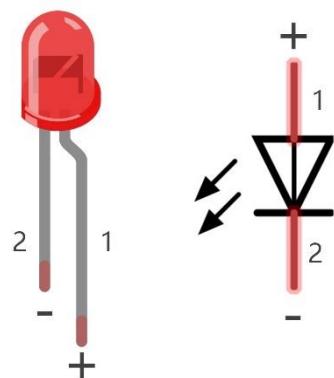
In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each project. They will be listed only in text form later.

Component knowledge

LED

LED is a kind of diode. LED will shine only if the long pin of LED is connected to the positive electrode and the short pin is connected to negative electrode.

This is also the features of the common diode. Diode works only if the voltage of its positive electrode is higher than its negative electrode.



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

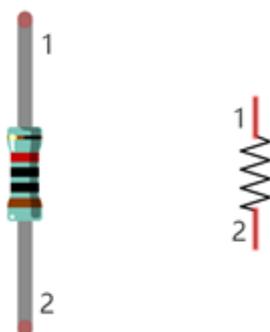
The LED can not be directly connected to power supply, which can damage component. A resistor with certain resistance must be connected in series in the circuit of LED.

Resistor

The unit of resistance(R) is ohm(Ω). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

Resistor is an electrical component that limits or regulates the flow of current in an electronic circuit.

The left is the appearance of resistor, and the right is the symbol of resistor represented in circuit.

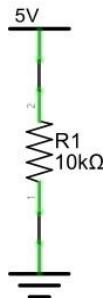


Color rings attached to the resistor is used to indicate its resistance. For more details of resistor color code, please refer to the appendix of this tutorial.



With the same voltage there will be less current with more resistance. And the links among current, voltage and resistance can be expressed by the formula below: $I=U/R$.

In the following diagram, the current through R1 is: $I=U/R=5V/10k\Omega=0.0005A=0.5mA$.



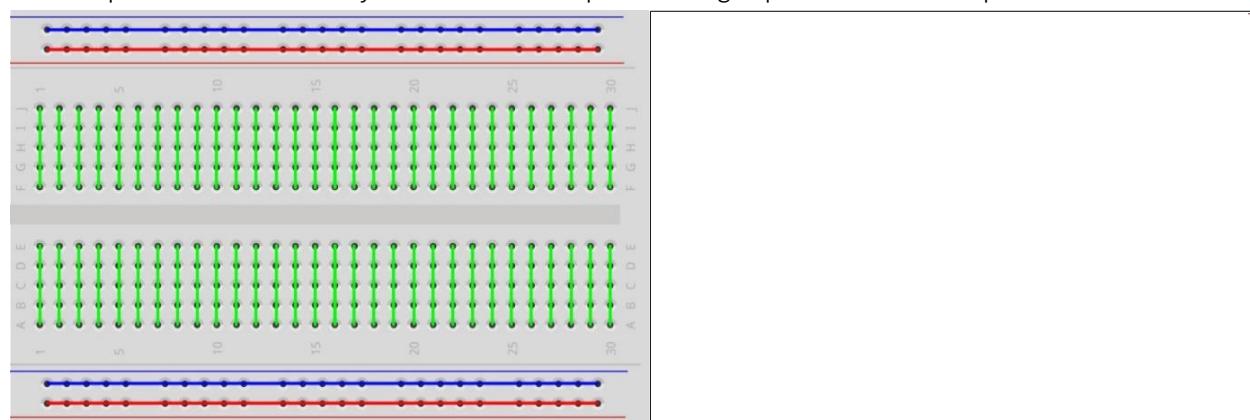
Do not connect the two poles of power supply with low resistance, which will make the current too high to damage electronic components.

And resistor has no poles.

Breadboard

Take a short breadboard as an example to introduce its feature, as below.

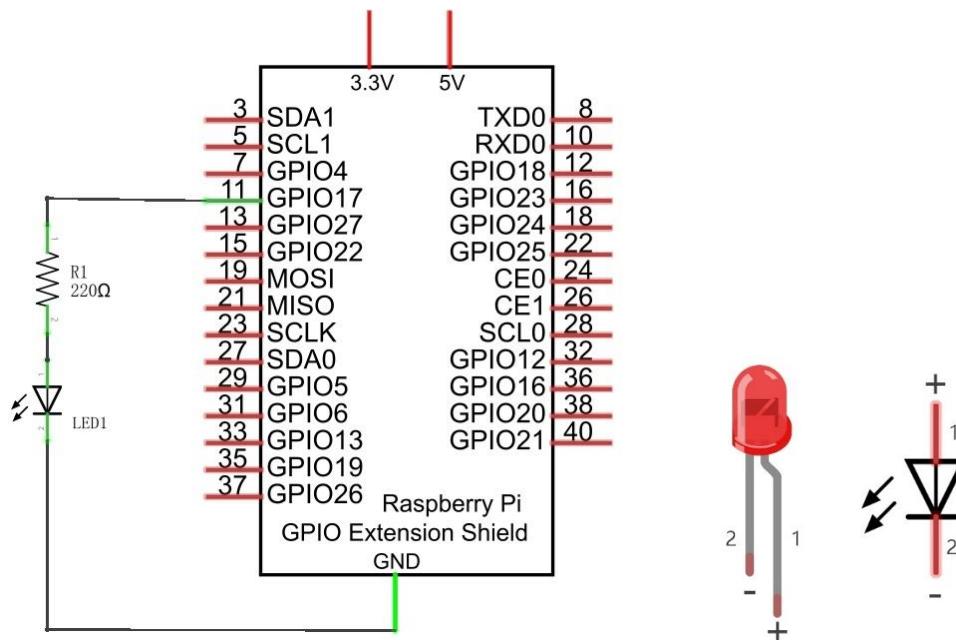
The left picture shows the way of connection of pins. The right picture shows the practical internal structure.



Circuit

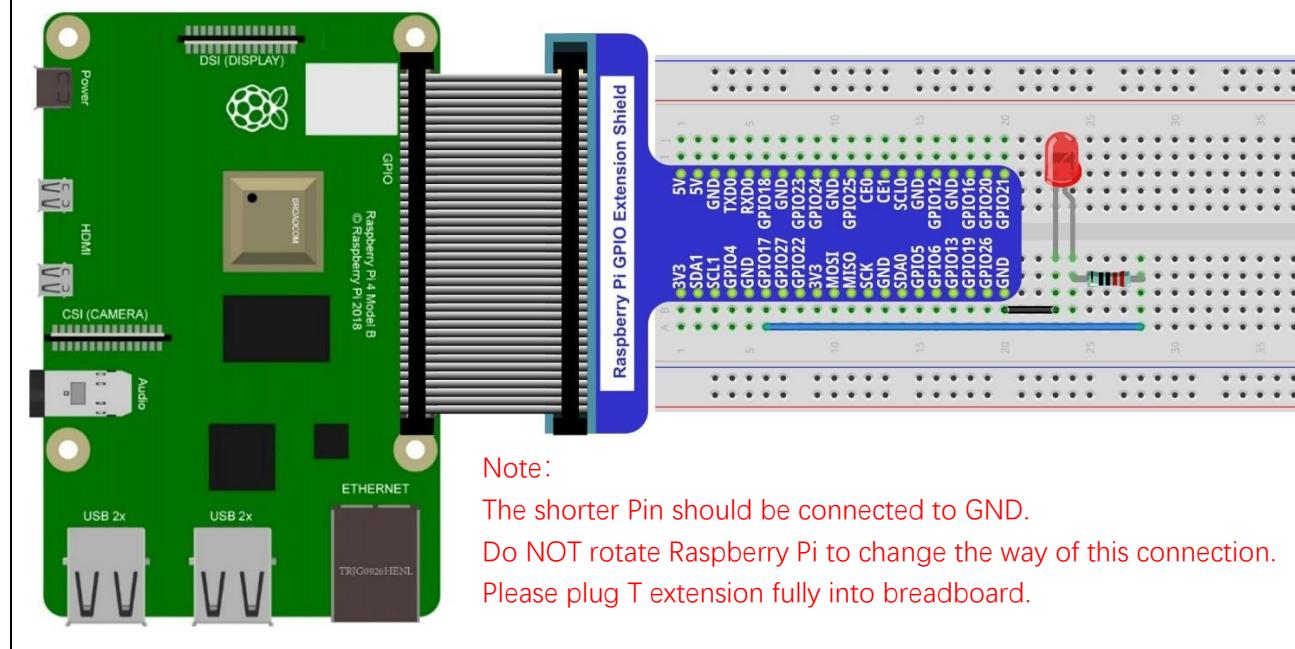
Disconnect RPi from GPIO Extension Shield first. Then build the circuit according to the circuit diagram and the hardware connection diagram. After the circuit is built and confirmed, connect RPi to GPIO Extension Shield. In addition, short circuit (especially 5V and GND, 3.3V and GND) should be avoid, because short circuit may cause abnormal circuit work, or even damage to RPi.

Schematic diagram



Hardware connection.

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



Because Numbering of GPIO Extension Shield is the same as RPi GPIO, later Hardware connection diagram will only show the part of breadboard and GPIO Extension Shield.

Code

According to the circuit, when the GPIO17 of RPi output high level, LED is turned on. Conversely, when the GPIO17 RPi output low level, LED is turned off. Therefore, we can let GPIO17 output high and low level in cycle to make LED blink. We will use both C code and Python code to achieve the target.

C Code 1.1.1 Blink

First, execute command into the terminal one by one. Then observe the project result, and analyze the code. If you want to execute it with editor, please refer to section [Code Editor](#) to configure.

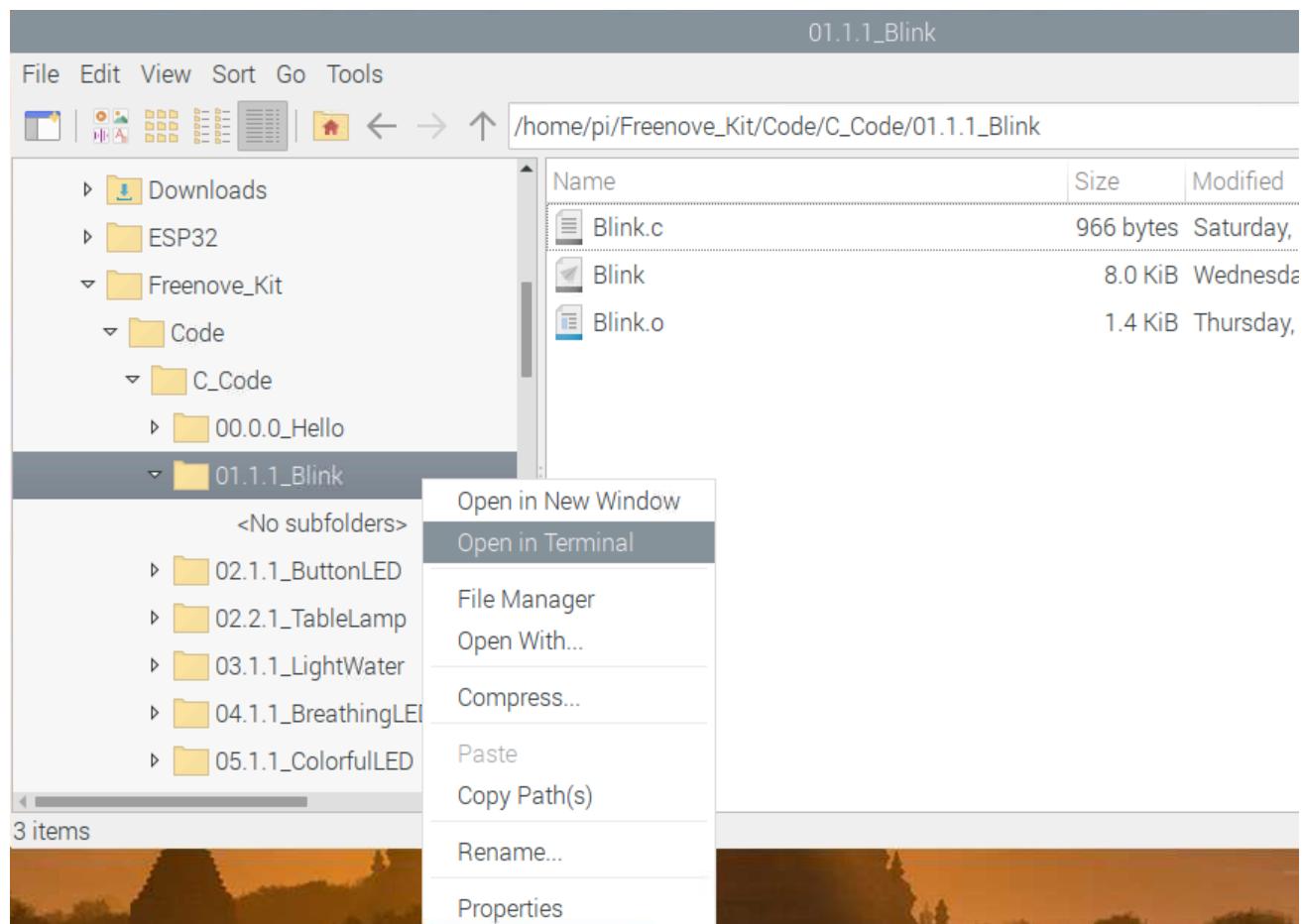
If you have any concerns, please contact us via: support@freenove.com

It is recommended to execute code as below.

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

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

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



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

"I" of "lwiringPi" is low case of "L".

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

3. Then run the generated file "blink".

```
sudo ./Blink
```

Now, LED start blink.

```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/01.1.1_Blink$ cd ~/Freenove_Kit/Code/C_Code/01.1.1_Blink
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/01.1.1_Blink$ gcc Blink.c -o Blink -lwiringPi
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/01.1.1_Blink$ sudo ./Blink
Program is starting ...
Using pin0
led turned on >>>
led turned off <<<
```

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

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define ledPin    0 //define the led pin number
5
6 void main(void)
7 {
8     printf("Program is starting ... \n");
9
10    wiringPiSetup(); //Initialize wiringPi.
11
12    pinMode(ledPin, OUTPUT); //Set the pin mode
13    printf("Using pin%d\n", ledPin); //Output information on terminal
14    while(1) {
15        digitalWrite(ledPin, HIGH); //Make GPIO output HIGH level
16        printf("led turned on >>>\n"); //Output information on terminal
17        delay(1000); //Wait for 1 second
18        digitalWrite(ledPin, LOW); //Make GPIO output LOW level
19        printf("led turned off <<<\n"); //Output information on terminal
20        delay(1000); //Wait for 1 second
21    }
22 }
```

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
```

In the main function main(), initialize wiringPi first.

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

After the wiringPi is initialized successfully, set the ledPin to output mode. And then enter the while cycle, which is an endless loop. That is, the program will always be executed in this cycle, unless it is ended outside. In this cycle, use digitalWrite (ledPin, HIGH) to make ledPin output high level, then LED is turned on. After a period of time delay, use digitalWrite(ledPin, LOW) to make ledPin output low level, then LED is turned off, which is followed by a delay. Repeat the cycle, 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
}
```

Among them, 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 functions, please refer to <http://wiringpi.com/reference/>

Python Code 1.1.1 Blink

Net, we will use Python language to make LED blink.

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

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
```

Now, LED start blinking.

```
pi@raspberrypi:~ $ cd Freenove_Kit/Code/Python_Code/01.1.1_Blink/
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/01.1.1_Blink $ python Blink.py
Program is starting ...

using pin11
led turned on >>>
led turned off <<<
```

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.BRD)      # 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()

```

In subfunction `setup()`, `GPIO.setmode(GPIO.BOARD)` is used to set the serial number for GPIO based on physical location of the pin. `GPIO17` use 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)

```

In `loop()`, there is a while cycle, which is an endless loop. That is, the program will always be executed in this cycle, unless it is ended outside. In this cycle, set `ledPin` output high level, then LED is turned on. After a period of time delay, set `ledPin` output low level, then LED is turned off, which is followed by a delay. Repeat the cycle, 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 will be executed, the LED will be turned off and then the IO port will be released. If close the program terminal directly, the program will be terminated too, but `finish()` function will not be executed. So, GPIO resources won't be released, in the warning message may appear next time you use GPIO. So, it is not a good habit to close the program terminal directly.

```

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

```

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 function used to generate PWM.

GPIO.setmode(mode)

Set the mode for pin serial number of GPIO.

mode=GPIO.BOARD, which represents the GPIO pin serial number is based on physical location of RPi.

mode=GPIO.BCM, which represents the pin serial number is based on CPU of BCM chip.

GPIO.setup(pin, mode)

Set pin to input mode or output mode. "pin" for the GPIO pin, "mode" for INPUT or OUTPUT.

GPIO.output(pin, mode)

Set 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:

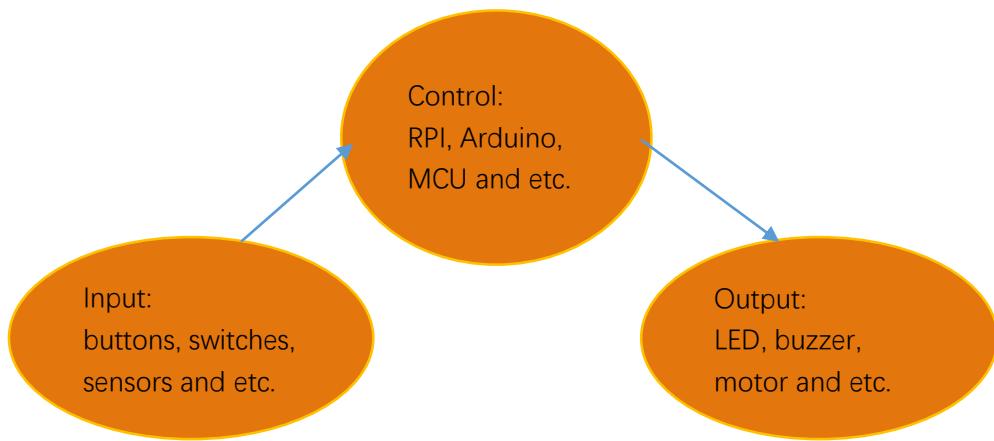
<https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>

"import time" time is a module of python.

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

Chapter 2 Button & LED

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



Next, we will build a simple control system to control LED through a button.

Project 2.1 Button & LED

In the project, we will control the LED state through a button. When the button is pressed, LED will be turn on, and when it is released, LED will be turn off.

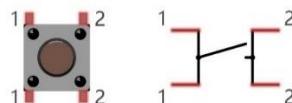
Component List

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

Component knowledge

Push button

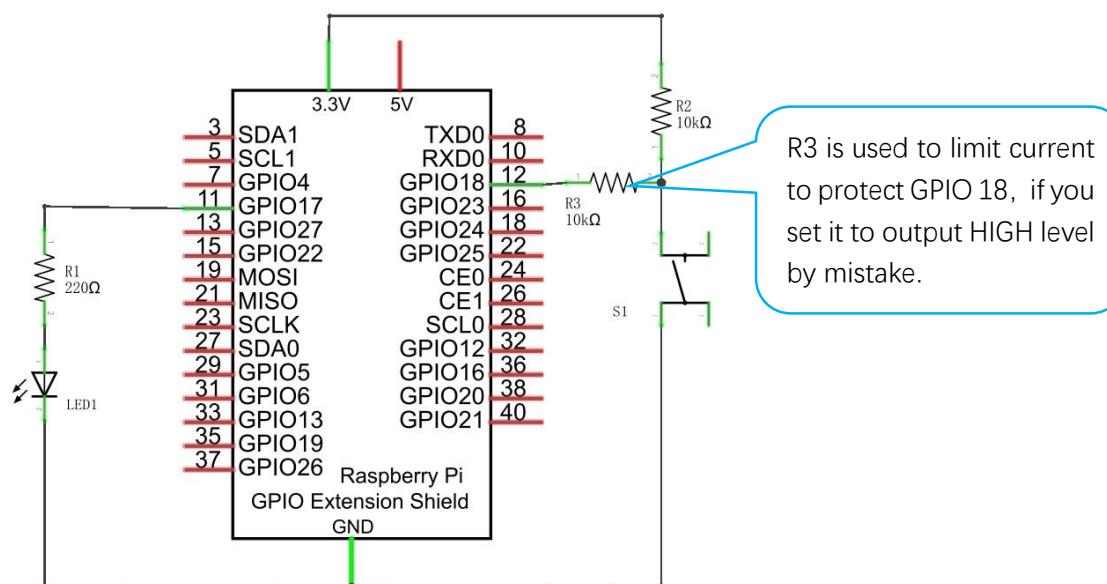
Push button has 4 pins. Two pins on the left is connected, and the right is similar as the left, which is shown in the below:



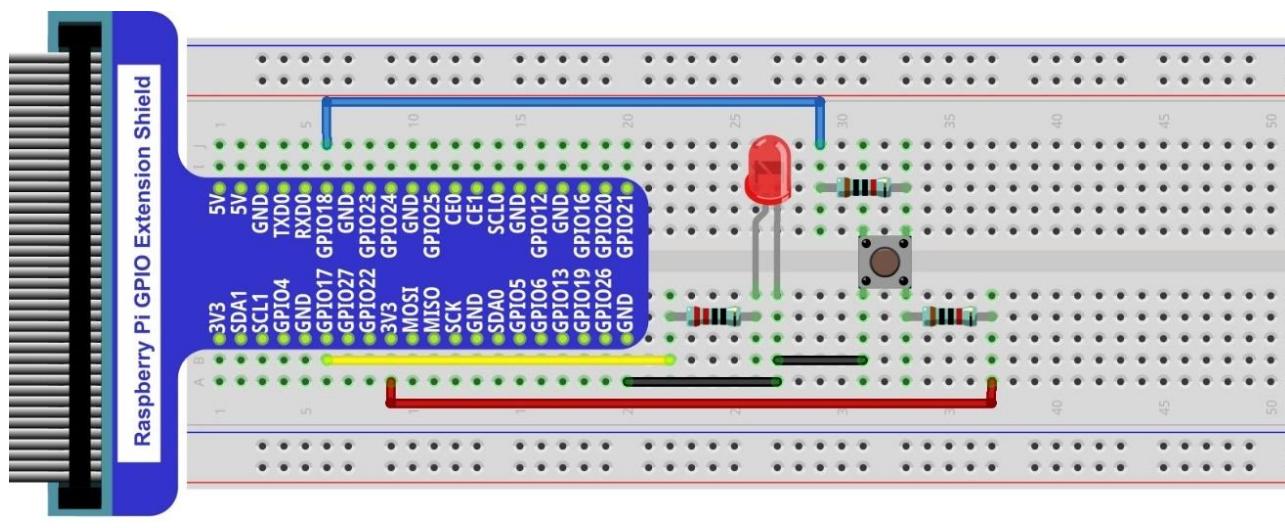
When the push button is pressed, the circuit is turned on.

Circuit

Schematic diagram



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



Code

This project is designed for learning how to use button to control LED. We first need to read the state of button, and then determine whether turn on LED according to the state of the button.

C Code 2.1.1 ButtonLED

First, observe the project result, then analyze the code.

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 cycle of main function, use digitalWrite(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 analyze the code.

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
```

Later, the terminal window continue to print out the characters "led off...", press the button, then LED is turned on and then terminal window print out the "led on...". Release the button, then LED is turned off and then terminal window print out the "led off...". You can press "Ctrl+C" 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 released
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. So, 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
```

In the loop function while dead circulation, continue 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. Or, 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 resource.

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.

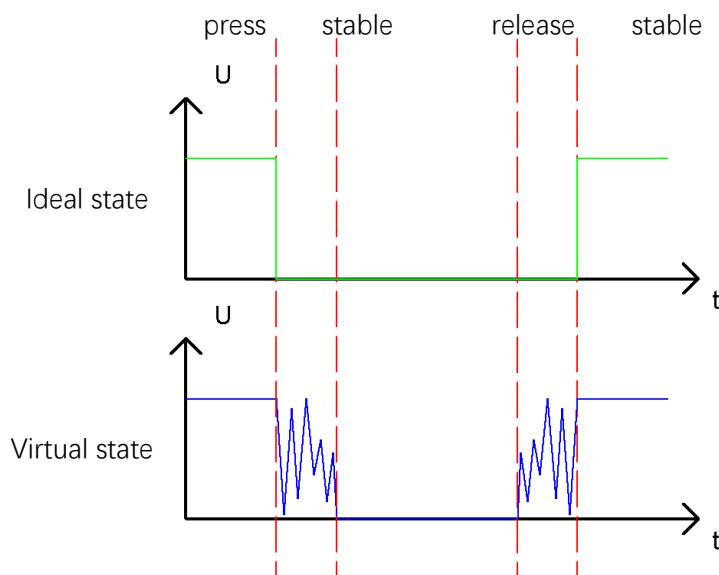
Project 2.2 MINI table lamp

We will also use a button, LED and RPi to make a MINI table lamp. But the function is different: Press the button, the LED will be turned on, and press the button again, the LED goes out.

First, let us learn some knowledge about the button.

Debounce for Push Button

When a Push Button is pressed, it will not change from one state to another state immediately. Due to mechanical vibration, there will be a continuous buffeting before it becomes another state. And the releasing-situation is similar with that process.



Therefore, if we directly detect the state of Push Button, there may be multiple pressing and releasing action in one pressing process. The buffeting will mislead the high-speed operation of the microcontroller to cause a lot of false judgments. So we need to eliminate the impact of buffeting. Our solution is: to judge the state of the button several times. Only when the button state is stable after a period of time, can it indicate that the button is pressed down.

This project needs the same components and circuits with the last section.

Code

In the project, we still detect the state of Button to control LED. Here we need to define a variable to save the state of LED. And when the button is pressed once, the state of LED will be changed once. This has achieved the function of the table lamp.

C Code 2.2.1 Tablelamp

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

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 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 once, then LED is turned on. Press the Button another time, then LED is turned off.

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define ledPin 0 //define the ledPin
5 #define buttonPin 1 //define the buttonPin
6 int ledState=LOW; //store the State of led
7 int buttonState=HIGH; //store the State of button
8 int lastbuttonState=HIGH; //store the lastState of button
9 long lastChangeTime; //store the change time of button state
10 long captureTime=50; //set the stable time for button state
11 int reading;
12 int main(void)
13 {
14     printf("Program is starting... \n");
15
16     wiringPiSetup(); //Initialize wiringPi.
17
18     pinMode(ledPin, OUTPUT); Set ledPin to output
19     pinMode(buttonPin, INPUT); Set buttonPin to input
20
21     pullUpDnControl(buttonPin, PUD_UP); //pull up to high level
22     while(1){
23         reading = digitalRead(buttonPin); //read the current state of button
24         if( reading != lastbuttonState){ //if the button state has changed, record the time
25             point
26             lastChangeTime = millis();
27         }
28     }
29 }
```

```

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 of button. We define several variables to save the state of LED and button. Then read the button state in while () constantly, and determine whether the state has changed. If it is, record this time point.

```

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 an unsigned 32-bit number which wraps after 49 days.

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

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

Finally, judge the state of Button. And if it is low level, the changing state indicates that the button is pressed, if the state is high level, then the button is released. Here, we change the status of the LED variable, and then update the state of 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 analyze the code.

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, press the Button once, then LED is turned on. Press the Button another time, then LED is turned 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:

```

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

RPi.GPIO provides us with a simple and effective function to eliminate the jitter, that is GPIO.add_event_detect(). It uses callback function. Once it detect that the buttonPin has a specified action FALLING, execute the 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 of 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 specified a function name, the function will be executed when the specified action is detected. And the fourth parameter is used to set the jitter time.

Chapter 3 LEDBar Graph

We have learned how to control a LED blinking, and next we will learn how to control a number of LED.

Project 3.1 Flowing Water Light

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

Component List

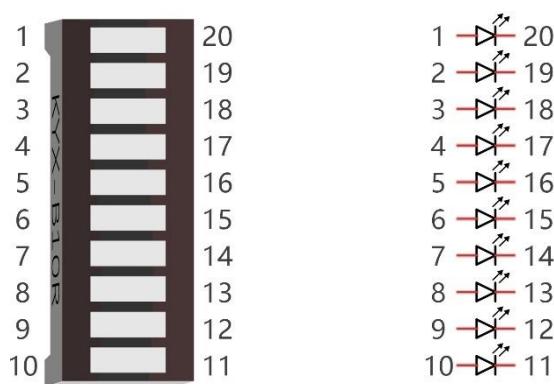
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED bar graph x1	Resistor 220Ω x10
Jumper		

Component knowledge

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

LED bar graph

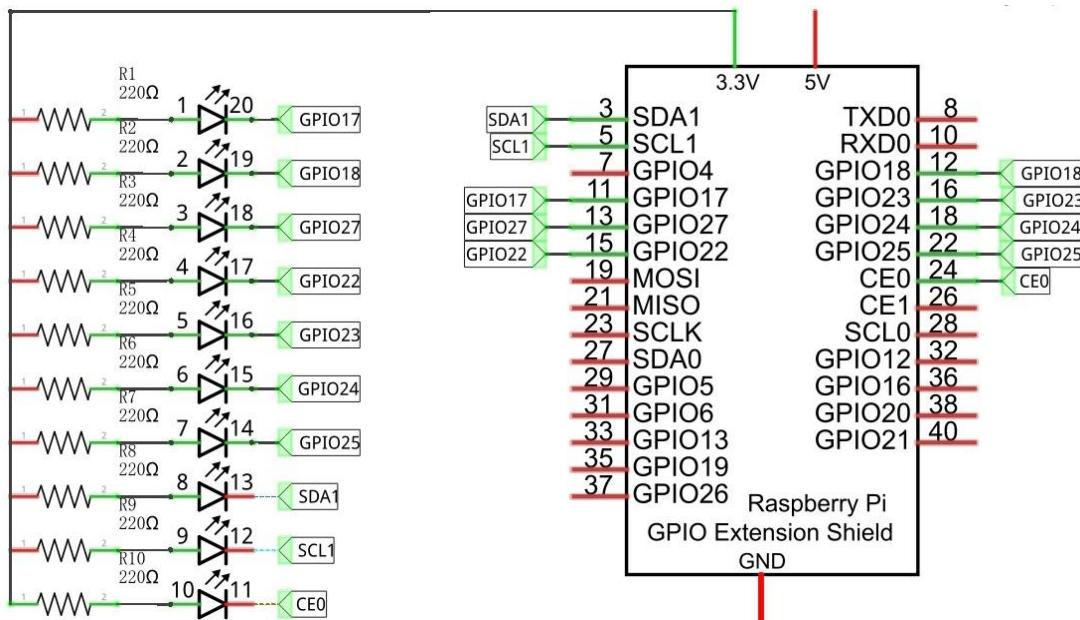
LED bar graph is a component Integration consist of 10 LEDs. There are two rows of pins at its bottom.



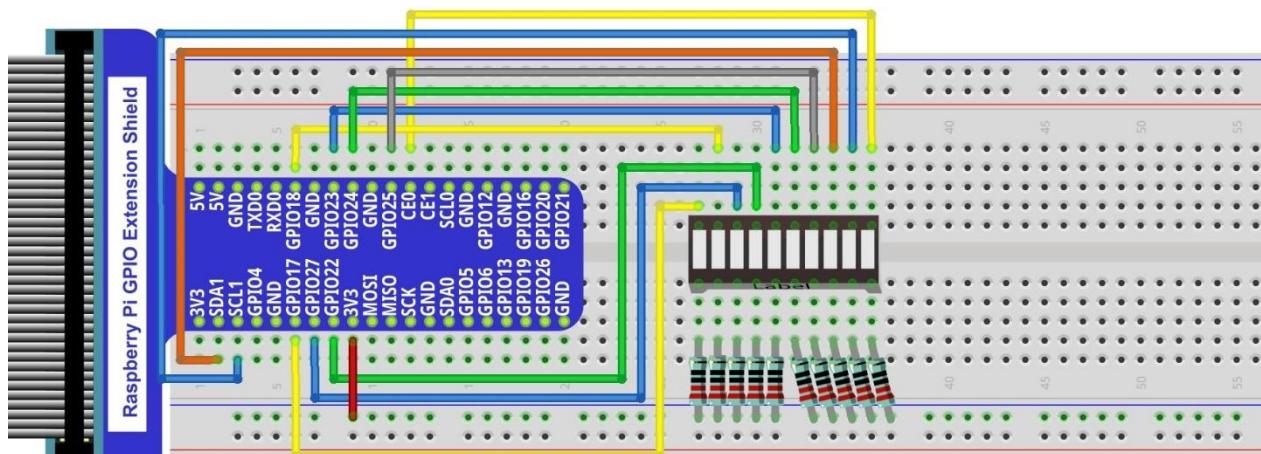
Circuit

The network label is used in the circuit diagram below, and the 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



In this circuit, the cathode of LED is connected to GPIO, which is the different from the front circuit. So, LED will be turned on when GPIO output low level in the program.

Code

This project is designed to make a water lamp. First turn on the first LED, then turn off it. Then turn on the second LED, and then turn off it..... Until the last LED is turned on, then is turned off. And repeats the process to achieve the effect of flowing water light.

C Code 3.1.1 LightWater

First observe the project result, and then analyze 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 C code.

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

2. Use 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 LEDBar 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 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define ledCounts 10
5 int pins[ledCounts] = {0, 1, 2, 3, 4, 5, 6, 8, 9, 10};
6
7 void main(void)
8 {
9     int i;
10    printf("Program is starting ... \n");
11
12    wiringPiSetup(); //Initialize wiringPi.
13
14    for(i=0;i<ledCounts;i++) {      //Set pinMode for all led pins to output
15        pinMode(pins[i], OUTPUT);
16    }
17
18    while(1) {
19        for(i=0;i<ledCounts;i++) { // move led(on) from left to right
20            digitalWrite(pins[i], LOW);
21            delay(100);
22            digitalWrite(pins[i], HIGH);
23        }
24        for(i=ledCounts-1;i>-1;i--) { // move led(on) from right to left
25            digitalWrite(pins[i], LOW);
26            delay(100);
27            digitalWrite(pins[i], HIGH);
28        }
29    }
30}
```

```

27     }
28 }
29 }
```

In the program, configure the GPIO0-GPIO9 to output mode. Then, in the endless “while” cycle of main function, use two “for” cycle 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 analyze 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 LEDBar 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” cycles to realize flowing water light from right to left and from left to right. Among them, `ledPins[::-1]` is used to traverse 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 study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That's what we're going to learn.

First, let's learn how to control the brightness of a LED.

Project 4.1 Breathing LED

Breathing light, that is, LED is turned from off to on gradually, gradually from on to off, just like "breathing". So, how to control the brightness of a LED? We will use PWM to achieve this target.

Component List

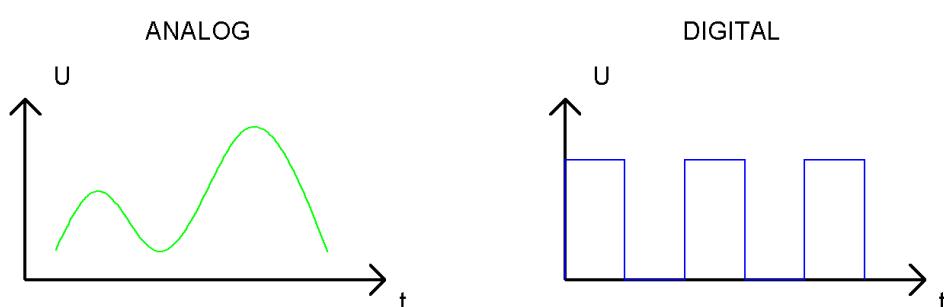
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED x1	Resistor 220Ω x1
Jumper		

Circuit knowledge

Analog & Digital

The analog signal is a continuous signal in time and value. On the contrary, digital signal is a discrete signal in time and value. Most signals in life are analog signals, for example, the temperature in one day is continuously changing, and will not appear a sudden change directly from 0°C to 10°C, while the digital signal is a jump change, which can be directly from 1 to 0.

Their difference can be illustrated by the following figure.



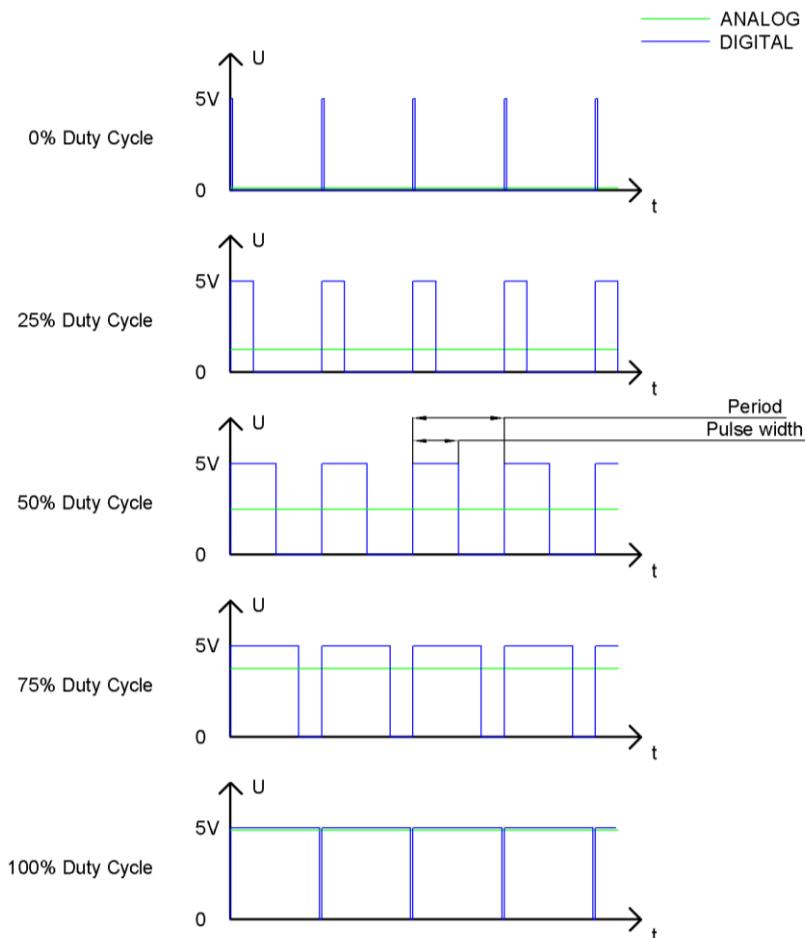
In practical application, we often use binary signal as digital signal, that is 0 and 1. The binary signal only has two forms (0 or 1), so it has strong stability. And digital signal and analog signal can be converted to each other.

PWM

PWM, namely Pulse-Width Modulation, is a very effective technique for using digital signals to control analog circuits. The common processors can not directly output analog signals. PWM technology make it very convenient to achieve this purpose.

PWM technology uses digital pins to send certain frequency of square waves, that is, the output of high level and low level that last for a while alternately. The total time for each set of high level and low level is generally fixed, which is called period (the reciprocal of the period is frequency). The time of high level outputting is generally called pulse width, and the percentage of pulse width is called duty cycle.

The longer the output of high level last, the larger the duty cycle and the larger the corresponding voltage in analog signal will be. The following figures show how the analog signals voltage vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The larger PWM duty cycle is, the larger the output power will be. So we can use PWM to control the brightness of LED, the speed of DC motor and so on.

It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent to the corresponding analog. so, we can control the output power of the LED and other output modules to achieve different effects.

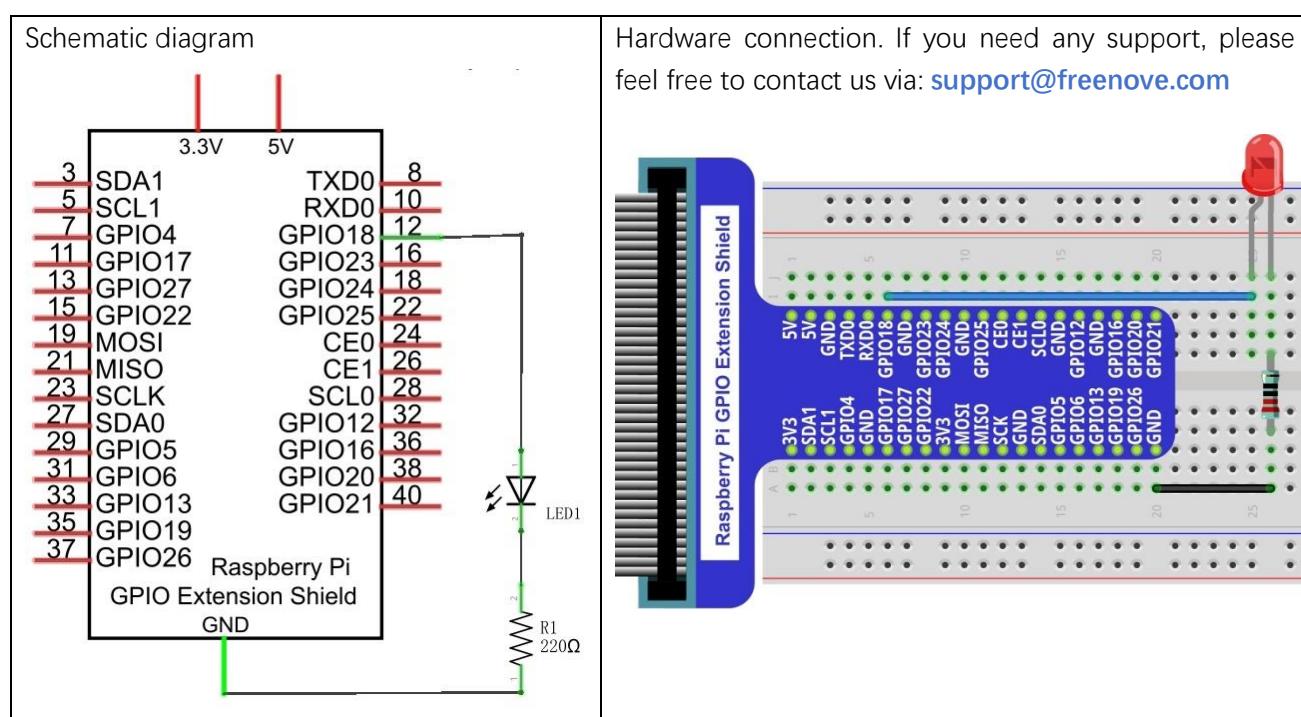
In RPi, only GPIO18 has the ability to output hardware PWM with a 10-bit accuracy, that is, 100% of the pulse width can be divided into $2^{10}=1024$ equal parts.

The wiringPi library of C provides a hardware PWM method and a software PWM method, while wiringPi library of Python doesn't provide hardware PWM method. There is only software PWM for Python.

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

In order to keep the running results consistent, we adopt the way of software PWM.

Circuit



Code

This project is designed to make PWM output GPIO18 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

C Code 4.1.1 BreathingLED

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

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
5 #define ledPin 1
6
7 void main(void)
8 {
9     int i;
10
11     printf("Program is starting ... \n");
12
13     wiringPiSetup(); //Initialize wiringPi.
14
15     softPwmCreate(ledPin, 0, 100); //Create SoftPWM pin
16
17     while(1) {
18         for(i=0;i<100;i++) { //make the led brighter
19             softPwmWrite(ledPin, i);
20             delay(20);
21         }
22         delay(300);
23         for(i=100;i>=0;i--) { //make the led darker
24             softPwmWrite(ledPin, i);
25             delay(20);
26         }
27         delay(300);
28     }
29 }
```

First, create a software PWM pin.

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

There are two “for” cycles in the next endless “while” cycle. The first makes the ledPin output PWM from 0% to 100% and the second makes the ledPin output 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 parameters of the `delay()` function in the “for” cycle.

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 analyze the code.

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 `python` command to execute python code “BreathingLED.py”.

python BreathingLED.py

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 import RPi.GPIO as GPIO
2 import time
3
4 LedPin = 12      # define the LedPin
5
6 def setup():
7     global p
8     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
9     GPIO.setup(LedPin, GPIO.OUT)   # set LedPin to OUTPUT mode
10    GPIO.output(LedPin, GPIO.LOW)  # make ledPin output LOW level to turn off LED
11
12    p = GPIO.PWM(LedPin, 500)    # set PWM Frequency to 500Hz
13    p.start(0)                  # set initial Duty Cycle to 0
14
15 def loop():
16     while True:
17         for dc in range(0, 101, 1): # make the led brighter
18             p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
19             time.sleep(0.01)
20             time.sleep(1)
21         for dc in range(100, -1, -1): # make the led darker
22             p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
23             time.sleep(0.01)

```

```

24         time.sleep(1)
25
26     def destroy():
27         p.stop() # stop PWM
28         GPIO.cleanup() # Release all GPIO
29
30     if __name__ == '__main__':    # Program entrance
31         print ('Program is starting ... ')
32         setup()
33         try:
34             loop()
35         except KeyboardInterrupt: # Press ctrl-c to end the program.
36             destroy()

```

LED is connected to the IO port called GPIO18. And LedPin is defined as 12 and set to output mode according to the corresponding chart for pins. Then create a PWM instance and set the PWM frequency to 1000HZ, 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 Frequency to 500Hz
    p.start(0)                   # set initial Duty Cycle to 0

```

There are two “for” cycles used to realize breathing LED in the next endless “while” cycle. The first makes the ledPin output PWM from 0% to 100% and the second makes the ledPin output 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 cycle, where 0.0 <= dc <= 100.0

p.stop()

To stop PWM.

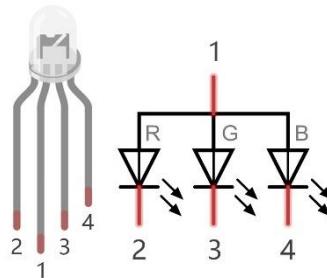
For more details about usage method for PWM of RPi.GPIO, please refer to:

<https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/>

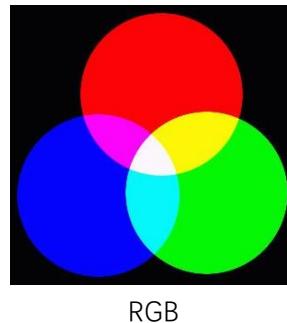
Chapter 5 RGBLED

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

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light. And it has 4 pins. The long pin (1) is the common port, that is, 3 LED's positive or negative port. The RGB LED with common positive port and its symbol are shown below. We can make RGB LED emit various colors of light by controlling these 3 LEDs to emit light with different brightness,



Red, green, and blue light are called 3 primary colors. When you combine these three primary-color light with different brightness, it can produce almost all kinds of visible lights. Computer screens, single pixel of cell phone screen, neon, and etc. are working under this principle.



RGB

If we use three 8 bit PWM to control the RGBLED, in theory, we can create $2^8 * 2^8 * 2^8 = 16777216$ (16 million) color through different combinations.

Next, we will use RGBLED to make a colorful LED.

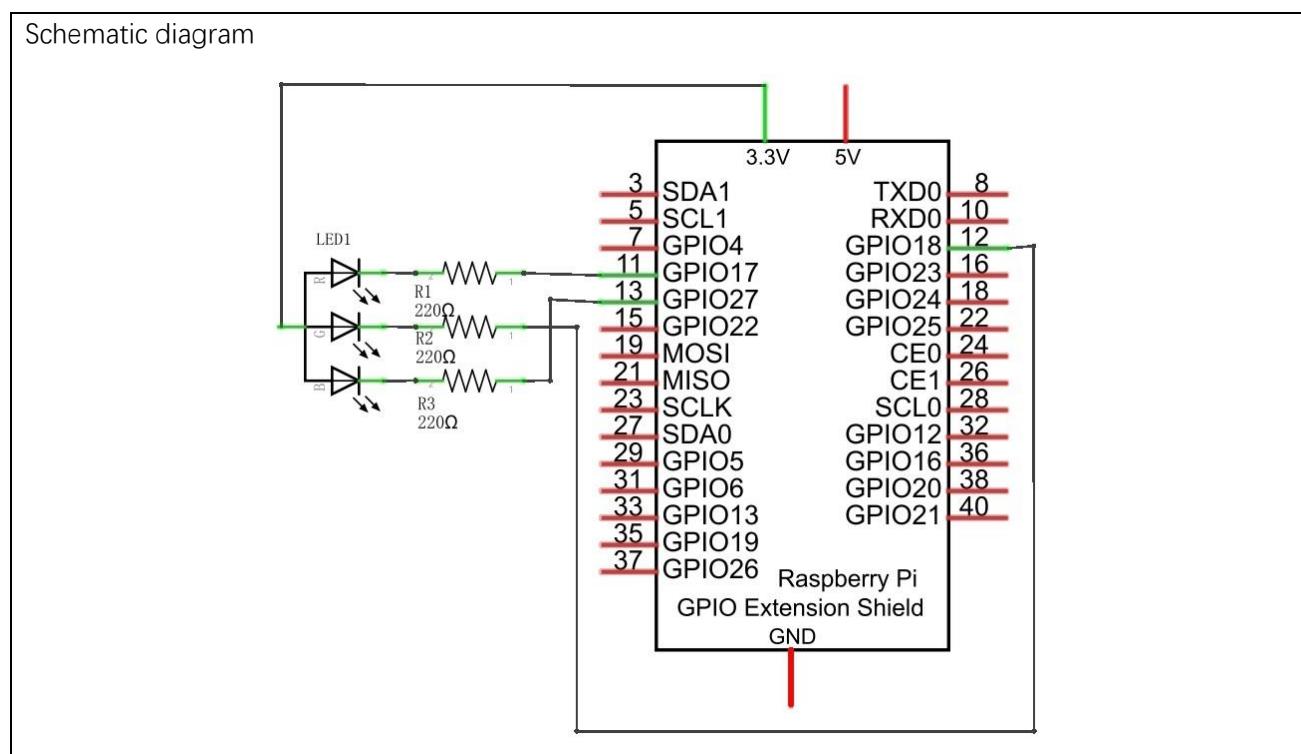
Project 5.1 Colorful LED

In this project, we will make a colorful LED. And we can control RGBLED to switch different colors automatically.

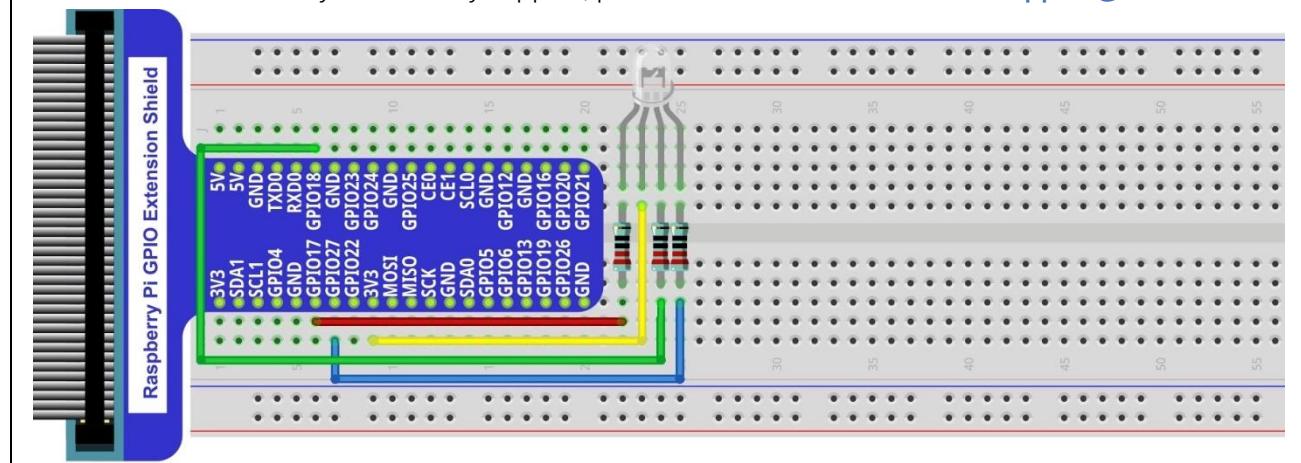
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 BreadBoard x1	RGBLED x1	Resistor 220Ω x3
Jumper		

Circuit



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



Code

Since this project requires 3 PWM, but in RPi, only one GPIO has the hardware capability to output PWM, we need to use the software to make the ordinary GPIO output PWM.

C Code 5.1.1 ColorfulLED

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

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 the compiler.

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

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

```
sudo ./ColorfulLED
```

After the program is executed, you will see that the RGBLED shows light of different color 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)
11 {
12     softPwmCreate(ledPinRed, 0, 100); //Creat SoftPWM pin for red
13     softPwmCreate(ledPinGreen, 0, 100); //Creat SoftPWM pin for green
14     softPwmCreate(ledPinBlue, 0, 100); //Creat SoftPWM pin for blue
15 }
16
17 void setLedColor(int r, int g, int b)
18 {
19     softPwmWrite(ledPinRed, r); //Set the duty cycle
20     softPwmWrite(ledPinGreen, g); //Set the duty cycle
21     softPwmWrite(ledPinBlue, b); //Set the duty cycle
22 }
23
24 int main(void)
25 {
26     int r,g,b;
```

```

27
28     printf("Program is starting ... \n");
29
30     wiringPiSetup(); //Initialize wiringPi.
31
32     setupLedPin();
33     while(1) {
34         r=random()%100; //get a random in (0, 100)
35         g=random()%100; //get a random in (0, 100)
36         b=random()%100; //get a random in (0, 100)
37         setLedColor(r, g, b); //set random as the duty cycle value
38         printf("r=%d, g=%d, b=%d \n", r, g, b);
39         delay(300);
40     }
41     return 0;
42 }
```

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” cycle of main function, get three random numbers and specify them as the PWM duty cycle, which will be assigned to the corresponding pins. So RGBLED 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(300);
}
```

The related function of Software PWM 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 analyze the code.

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 "ColorfullLED.py".

python ColorfulLED.py

After the program is executed, you will see that the RGBLED shows light of different color randomly.

The following is the program code:

```

1 import RPi.GPIO as GPIO
2 import time
3 import random
4
5 pins = {'pinRed':11, 'pinGreen':12, 'pinBlue':13} # define the pins for RGBLED
6
7 def setup():
8     global pwmRed, pwmGreen, pwmBlue
9     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
10    GPIO.setup(pins, GPIO.OUT)    # set RGBLED pins to OUTPUT mode
11    GPIO.output(pins, GPIO.HIGH)  # make RGBLED pins output HIGH level
12    pwmRed = GPIO.PWM(pins['pinRed'], 2000)    # set PWM Frequnce to 2kHz
13    pwmGreen = GPIO.PWM(pins['pinGreen'], 2000)  # set PWM Frequnce to 2kHz
14    pwmBlue = GPIO.PWM(pins['pinBlue'], 2000)    # set PWM Frequnce to 2kHz
15    pwmRed.start(0)      # set initial Duty Cycle to 0
16    pwmGreen.start(0)
17    pwmBlue.start(0)
18
19 def setColor(r_val,g_val,b_val):      # change duty cycle for three pins to r_val,g_val,b_val
20     pwmRed.ChangeDutyCycle(r_val)      # change pwmRed duty cycle to r_val
21     pwmGreen.ChangeDutyCycle(g_val)
22     pwmBlue.ChangeDutyCycle(b_val)
23
24 def loop():
25     while True :
26         r=random.randint(0,100)  #get a random in (0,100)
27         g=random.randint(0,100)
28         b=random.randint(0,100)
29         setColor(r,g,b)        #set random as a duty cycle value

```

```

30         print (' r=%d, g=%d, b=%d ' %(r ,g, b))
31         time.sleep(0.3)
32
33     def destroy():
34         pwmRed.stop()
35         pwmGreen.stop()
36         pwmBlue.stop()
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()

```

In last chapter, we have learned how to use python language to make a pin output PWM. In this project, we let three pins output PWM, and the usage is exactly the same as last chapter. In the “while” cycle of “loop” function, we first obtain three random numbers, and then specify these three random numbers as the PWM value of the three pins so that the RGBLED switching of different 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(0.3)

```

About function randint():

random.randint(a, b)

The function can returns a random integer within the specified range (a, b).

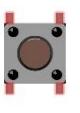
Chapter 6 Buzzer

In this chapter, we will learn a component that can sound, buzzer.

Project 6.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

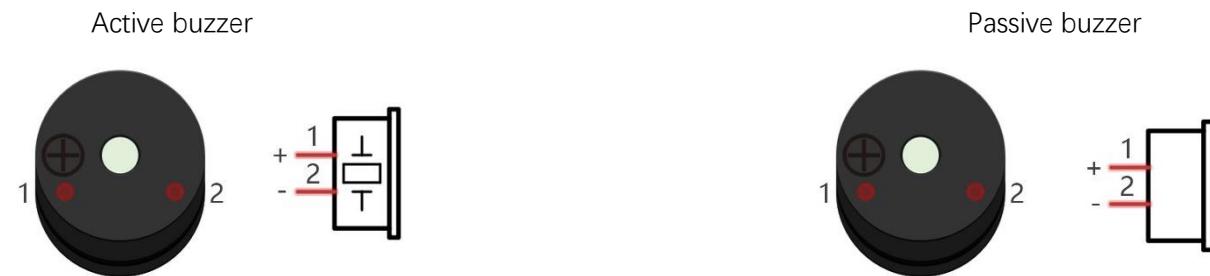
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper 			
NPN transistor x1 (S8050) 	Active buzzer x1 	Push button x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component knowledge

Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock, alarm. Buzzer has active and passive type. Active buzzer has oscillator inside, and it will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.



Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer is loudest when its resonant frequency is 2kHz.

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

How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzer is more complex than passive buzzer in manufacture. There are a lot of circuits and crystal oscillator elements inside active buzzer, which are covered with a waterproof coating, around its pins. Passive buzzer doesn't have these coatings. From the pins of passive buzzer, you can even see the circuit board, coils, and permanent magnets directly.

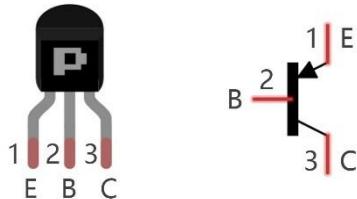
Transistor

Due to the current operating of buzzer is so large that GPIO of RPi output capability can not meet the requirement. A transistor of NPN type is needed here to amplify the current.

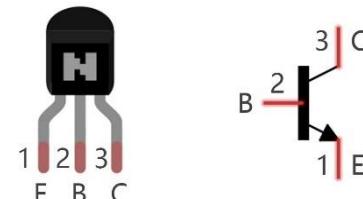


Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types shown below: PNP and NPN,

PNP transistor



NPN transistor

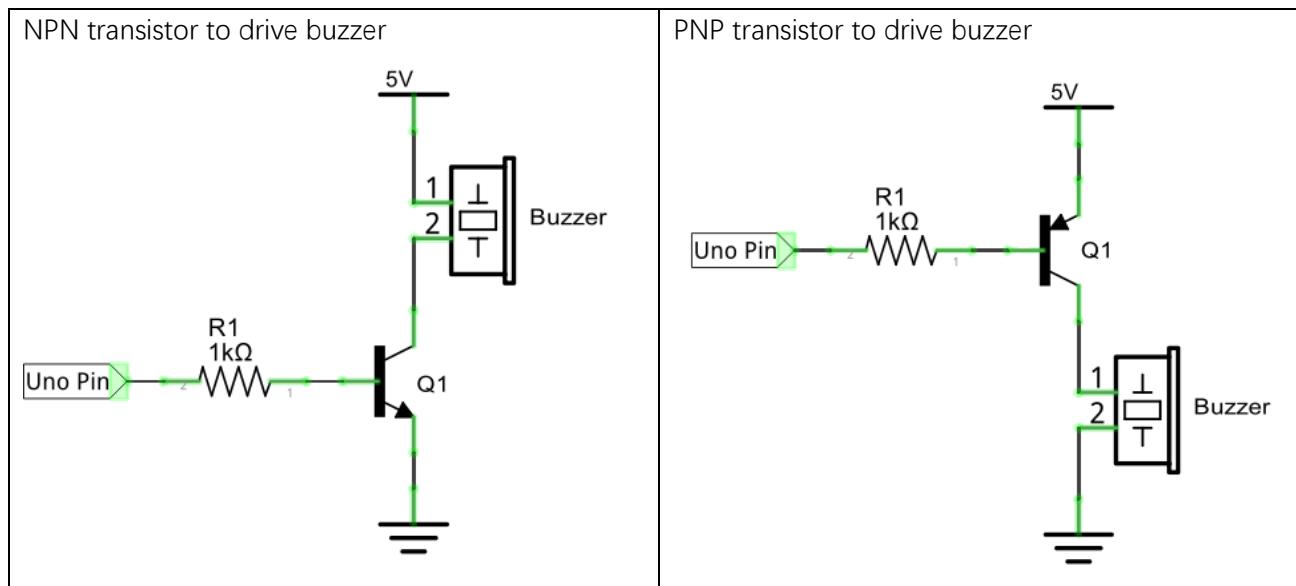


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

According to the transistor's characteristics, it is often used as a switch in digital circuits. For micro-controller's capacity of output current is very weak, we will use transistor to amplify current and drive large-current components.

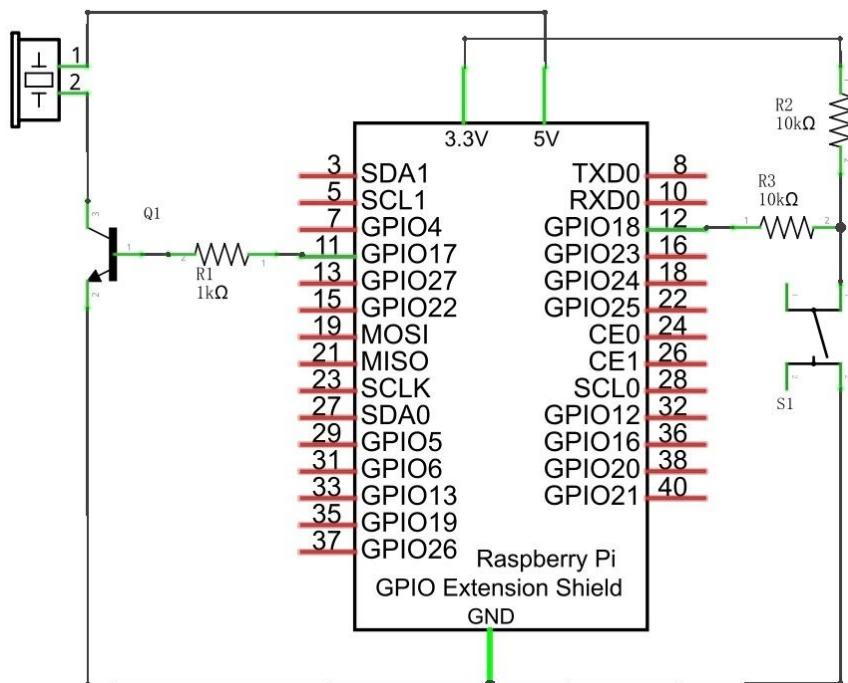
When use NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor gets conducted, and the buzzer make a sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When use PNP transistor to drive buzzer, we often adopt the following method. If GPIO outputs low level, current will flow through R1, the transistor gets conducted, buzzer make a sound. If GPIO outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

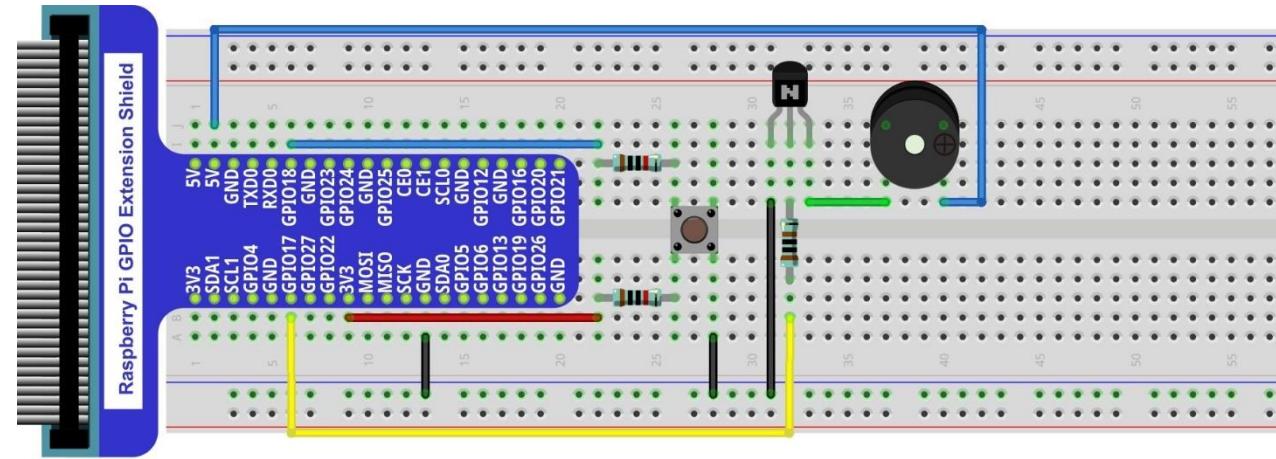


Circuit

Schematic diagram



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 buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.

Code

In this project, buzzer is controlled by the button. When the button is pressed, the buzzer sounds. And when the button is released, the buzzer stops sounding. In the logic, it is the same to using button to control LED.

C Code 6.1.1 Doorbell

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

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 button, then buzzer sounds. And when the button is release, the buzzer will stop sounding.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define buzzerPin 0 //define the buzzerPin
5 #define buttonPin 1 //define the buttonPin
6
7 void main(void)
8 {
9     printf("Program is starting ... \n");
10
11     wiringPiSetup();
12
13     pinMode(buzzerPin, OUTPUT);
14     pinMode(buttonPin, INPUT);
15
16     pullUpDnControl(buttonPin, PUD_UP); //pull up to HIGH level
17     while(1) {
18
19         if(digitalRead(buttonPin) == LOW){ //button is pressed
20             digitalWrite(buzzerPin, HIGH); //Turn on buzzer
21             printf("buzzer turned on >> \n");
22         }
23         else { //button is released
24             digitalWrite(buzzerPin, LOW); //Turn off buzzer
25             printf("buzzer turned off << \n");
26         }
}
```

```

27     }
28 }
```

The code is exactly the same to using button to control LED logically. You can try to use the PNP transistor to achieve the function of his circuit once again.

Python Code 6.1.1 Doorbell

First observe the project result, then analyze the code.

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 button, then buzzer sounds. And when the button is released, the buzzer will stop sounding.

The following is the program code:

```

1 import RPi.GPIO as GPIO
2
3 buzzerPin = 11      # define buzzerPin
4 buttonPin = 12      # define buttonPin
5
6 def setup():
7     GPIO.setmode(GPIO.BCM)          # use PHYSICAL GPIO Numbering
8     GPIO.setup(buzzerPin, GPIO.OUT)  # set buzzerPin to OUTPUT mode
9     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
10    INPUT mode
11
12 def loop():
13     while True:
14         if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
15             GPIO.output(buzzerPin,GPIO.HIGH) # turn on buzzer
16             print ('buzzer turned on >>>')
17         else : # if button is released
18             GPIO.output(buzzerPin,GPIO.LOW) # turn off buzzer
19             print ('buzzer turned off <<<')
20
21 def destroy():
22     GPIO.cleanup()                  # Release all GPIO
23
24 if __name__ == '__main__':      # Program entrance
25     print ('Program is starting...')
26     setup()
27     try:
```

```

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

```

The code is exactly the same to using button to control LED logically. You can try to use the PNP transistor to achieve the function of his circuit once again.

Project 6.2 Alertor

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

Component list and the circuit part is the similar to last section. In the Doorbell circuit only the active buzzer needs to be replaced with a passive buzzer.

Code

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. In the logic, it is the same to using button to control LED. In the control method, passive buzzer requires PWM of certain frequency to sound.

C Code 6.2.1 Alertor

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

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 are needed to add 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 button, then buzzer sounds. And when the button is release, the buzzer will stop sounding.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softTone.h>
4 #include <math.h>
5
6 #define buzzerPin    0 //define the buzzerPin
7 #define buttonPin    1 //define the buttonPin
8
9 void alertor(int pin) {
10     int x;
11     double sinVal, toneVal;

```

```
12     for(x=0;x<360;x++) { // frequency of the alertor is consistent with the sine wave
13         sinVal = sin(x * (M_PI / 180));           //Calculate the sine value
14         toneVal = 2000 + sinVal * 500;           //Add the resonant frequency and weighted sine
15         value
16         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 logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so you need to create a software PWM pin though softToneCreate (buzzeRPin). Here softTone is dedicated to generate square wave with variable frequency and duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

	softToneCreate (buzzeRPin) ;
--	------------------------------

In the while cycle of main function, when the button is pressed, subfunction alertor() will be called and the alertor will issue a warning sound. The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (here is 500) and 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 close the buzzer, just set PWM frequency of the buzzer pin to 0.

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

The related functions of softTone is 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 analyze the code.

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 button, then the buzzer sounds. When the button is released, the buzzer will stop sounding.

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 import math
4
5 buzzerPin = 11      # define the buzzerPin
6 buttonPin = 12      # define the buttonPin
```

```
7
8 def setup():
9     global p
10    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
11    GPIO.setup(buzzerPin, GPIO.OUT)   # set RGBLED pins to OUTPUT mode
12    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set buttonPin to INPUT
13 mode, and pull up to HIGH level, 3.3V
14    p = GPIO.PWM(buzzerPin, 1)
15    p.start(0);
16
17 def loop():
18     while True:
19         if GPIO.input(buttonPin)==GPIO.LOW:
20             alertor()
21             print ('alertor turned on >>> ')
22         else :
23             stopAlertor()
24             print ('alertor turned off <<< ')
25 def alertor():
26     p.start(50)
27     for x in range(0,361):      # Make frequency of the alertor consistent with the sine wave
28         sinVal = math.sin(x * (math.pi / 180.0))          # calculate the sine value
29         toneVal = 2000 + sinVal * 500 # Add to the resonant frequency with a Weighted
30         p.ChangeFrequency(toneVal)      # Change Frequency of PWM to toneVal
31         time.sleep(0.001)
32
33 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 logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so you need to create a software PWM pin through softToneCreate (buzzerPin). The way to create PWM is also introduced before in the sections about BreathingLED and RGBLED.

```
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);
```

In the while cycle of main function, when the button is pressed, subfunction alertor() will be called and the alertor will issue a warning sound. The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (here is 500) and plus the resonant frequency of buzzer. We can set the PWM frequency through p.ChangeFrequency(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 button is released, the buzzer will be closed.

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

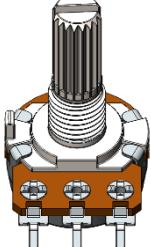
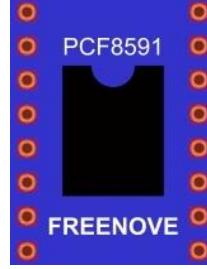
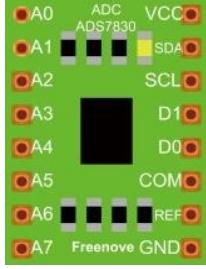
(Important)Chapter 7 ADC

We have learned how to control the brightness of LED through PWM and understood that PWM is not the real analog before. In this chapter, we will learn how to read analog quantities through ADC Module, convert it into digital.

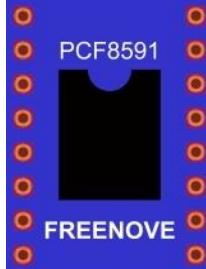
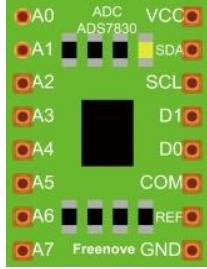
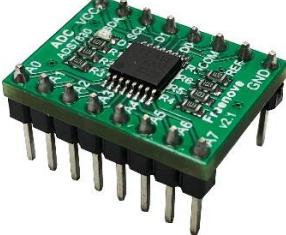
Project 7.1 Read the Voltage of Potentiometer

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

Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M x16 
Rotary potentiometer x1 	ADC module x1 Or   

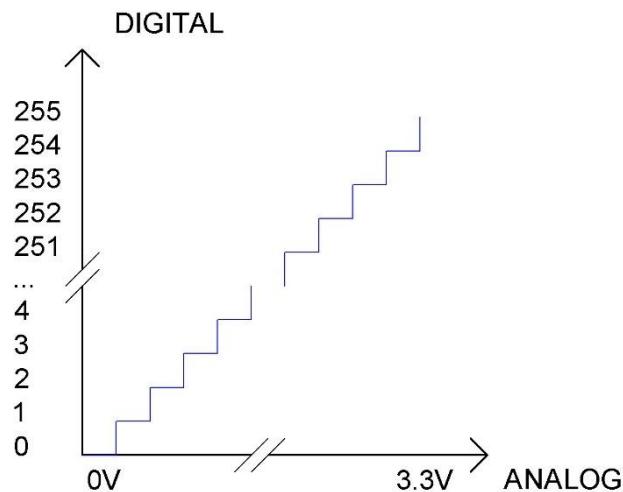
This product contains **only one ADC module**, which has two different types, PCF8591 and ADS7830. The functions involved in this tutorial are the same. Please build corresponding circuits according to different ADC module in your kit.

ADC module: PCF8591	ADC module: ADS7830
Model diagram  	Model diagram  

Circuit knowledge

ADC

ADC, Analog-to-Digital Converter, is a device used to convert analog to digital. The range of the ADC module is 8 bits, that means the resolution is $2^8=256$, and it represents the range (here is 3.3V) will be divided equally to 256 parts. The analog of each range corresponds to one ADC values. So the more bits ADC has, the denser the partition of analog will be, also the higher precision of the conversion will be.



Subsection 1: the analog in rang of 0V-3.3/256 V corresponds to digital 0;

Subsection 2: the analog in rang of 3.3 /256 V-2*3.3 /256V corresponds to digital 1;

...

The following analog will be divided accordingly.

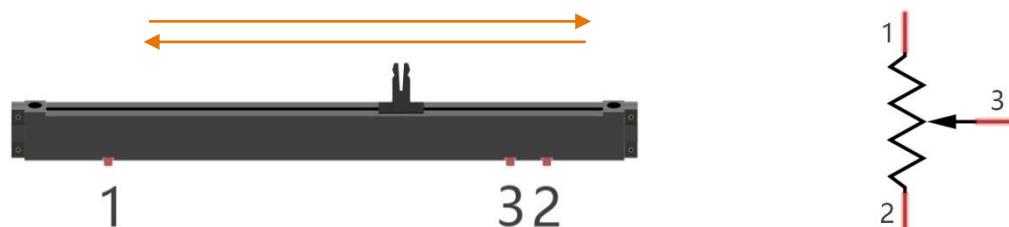
DAC

DAC, that is, Digital-to-Analog Converter, is the reverse process of ADC. The digital I/O port can output high level and low level, but cannot output an intermediate voltage value, which can be solved by DAC. 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 $3.3/256 *128=1.65$ V, the higher accuracy of DAC is, the higher the accuracy of output voltage value is.

Component knowledge

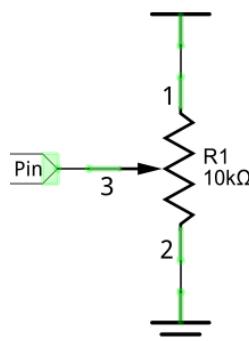
Potentiometer

Potentiometer is a resistive element with three Terminal part and the resistance can be adjusted according to a certain variation. Potentiometer is often made up by resistance and removable brush. When the brush moves along the resistor body, there will be resistance or voltage that has a certain relationship with displacement on the output side (3). Figure shown below is the linear sliding potentiometer and its symbol.



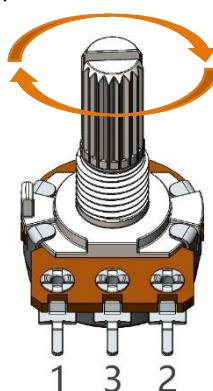
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pins 1 to pin 2, the resistance between pin 1, and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

In the circuit. The both sides of resistance body are often connected to the positive and negative electrode of the power. When you slide the brush pin 3, you can get a certain voltage in the range of the power supply.



Rotary potentiometer

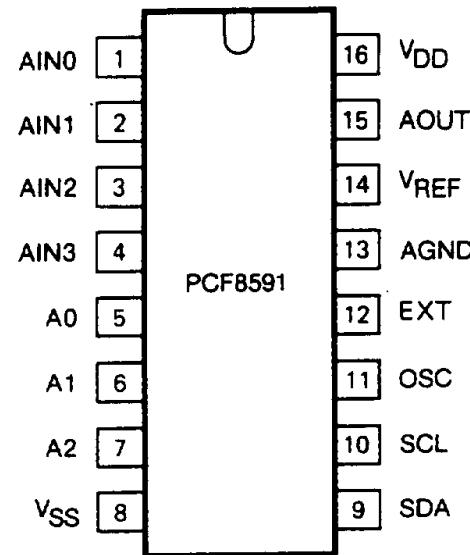
Rotary potentiometer and linear potentiometer have similar function; the only difference is: the resistance is adjusted through rotating the potentiometer.



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	I ² C-bus data input/output	
SCL	10	I ² C-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 datasheet.

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.

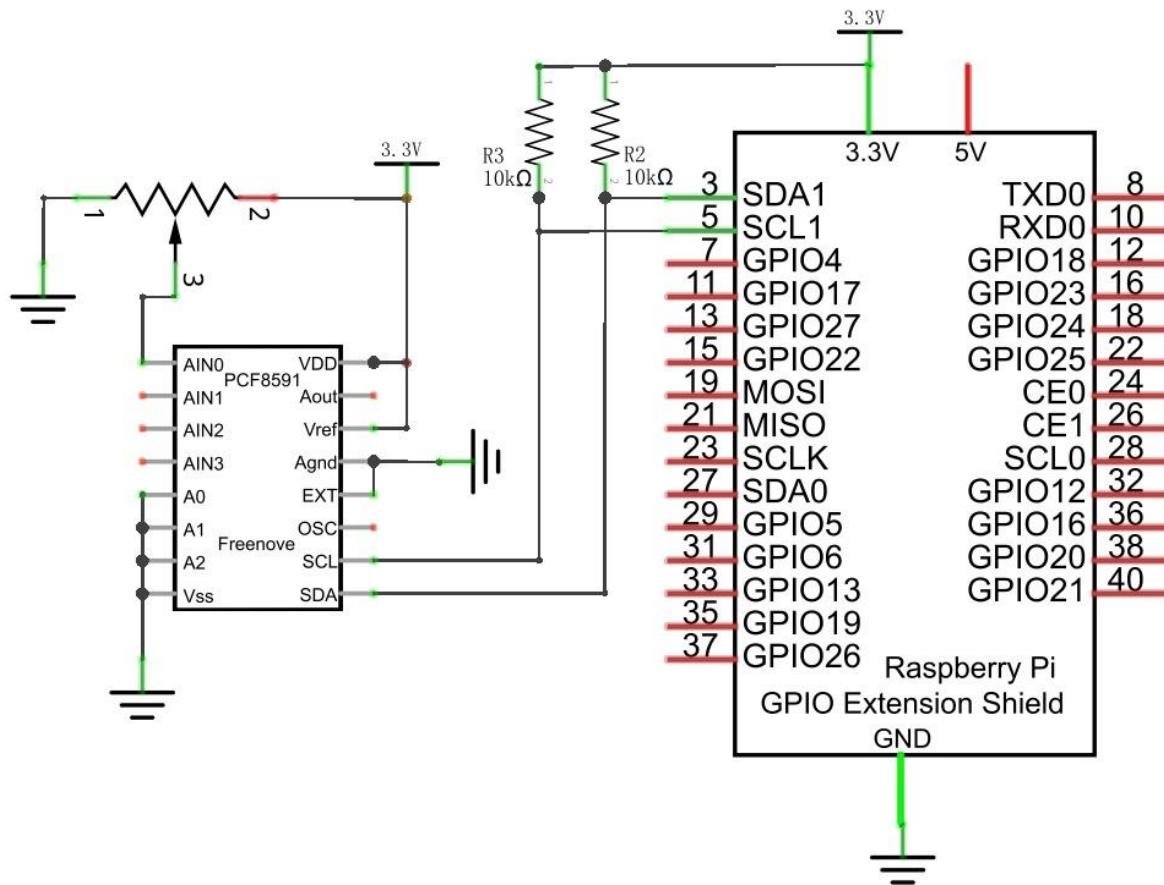
SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		
CH5	6		
CH6	7		
CH7	8		
GND	9	Ground	
REF in/out	10	Internal +2.5V Reference, External Reference Input	
COM	11	Common to Analog Input Channel	
A0	12	Hardware address	
A1	13		
SCL	14	Serial Clock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

I2C communication

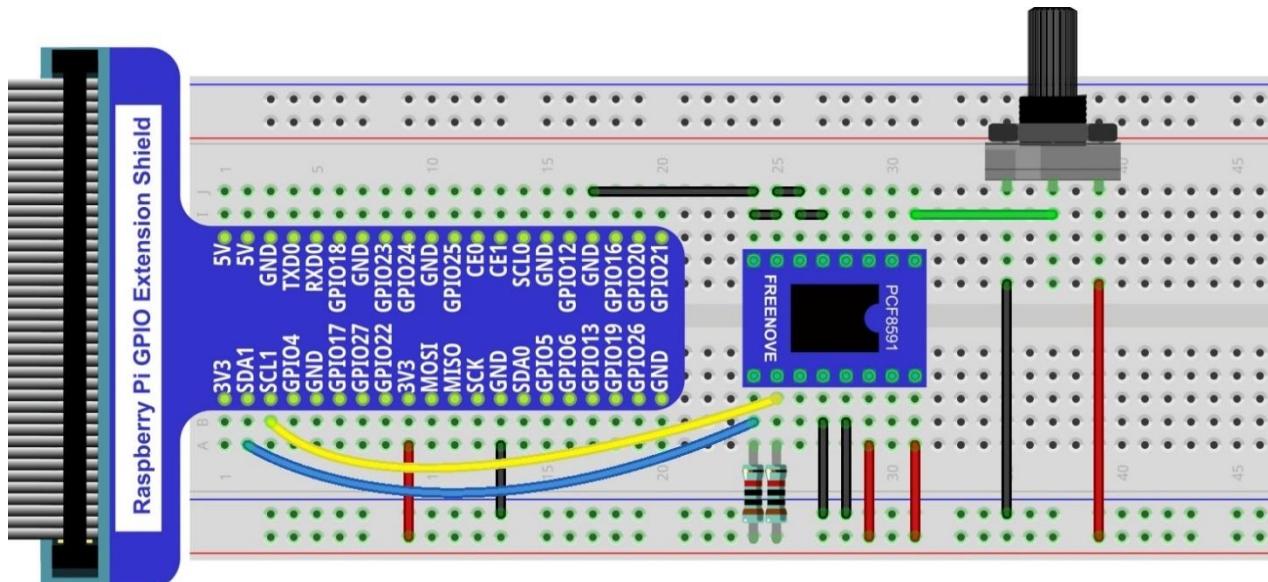
I2C(Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used to connection of micro controller and its peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique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

Circuit with PCF8591

Schematic diagram



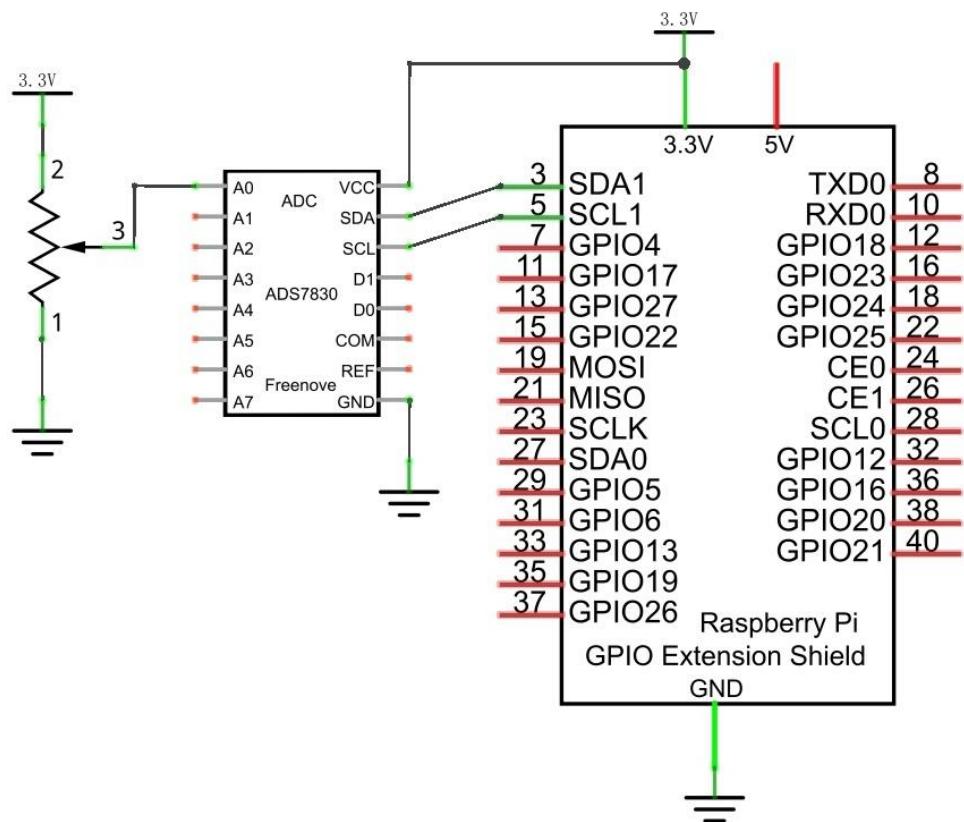
Hardware connection



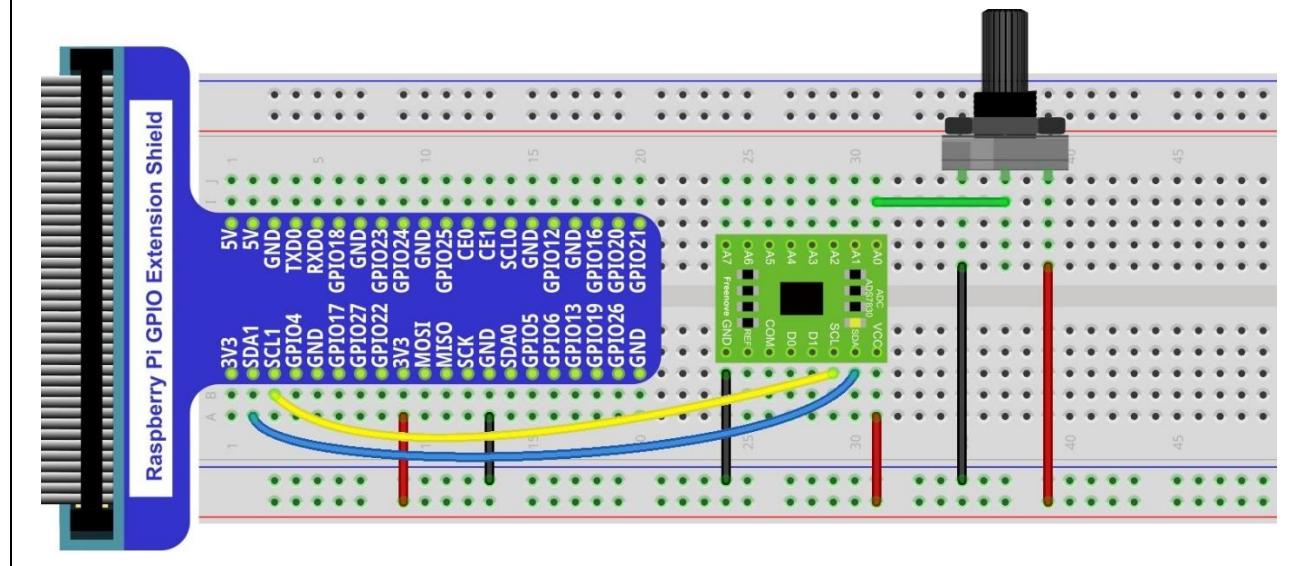
Please keep the **chip mark** consistent to make the chips under right direction and position.

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

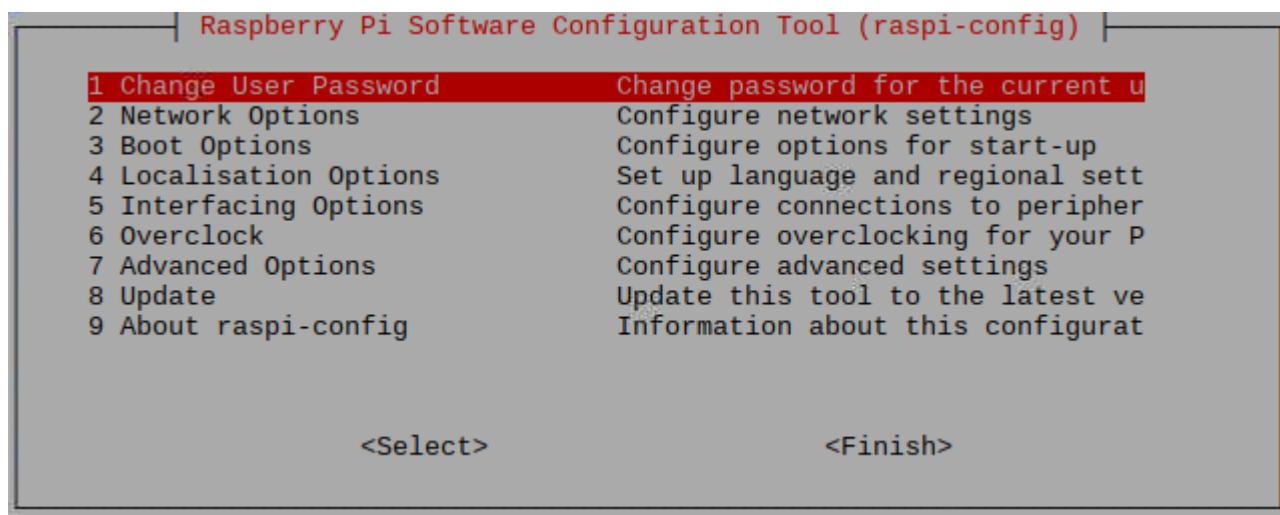
Enable I2C

The I2C interface raspberry pie is closed in default. You need to open it manually. You can enable the I2C interface in the following way.

Type command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options” → “P5 I2C” → “Yes” → “Finish” in order and restart your RPi later. Then the I2C module is 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:

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

Install I2C-Tools

Type the command to install I2C-Tools. It is installed in Raspbian system by default.

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

I2C device address detection:

```
i2cdetect -y 1
```

When you are using PCF8591, the result is below:

```
pi@raspberrypi:~ $ i2cdetect -y 1
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: ...
10: ...
20: ...
30: ...
40: ...          48 ...
50: ...
60: ...
70: ...
```

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

When you are using ADS, the result is below:

```
pi@raspberrypi:~ $ i2cdetect -y 1
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: ...
10: ...
20: ...
30: ...
40: ...          -- 4b ...
50: ...
60: ...
70: ...
```

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

Install Smbus Module

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

Code

C Code 7.1.1 ADC

For C code, ADCDevice, a custom library, needs to be installed.

1. Use cd command to enter folder of ADCDevice library.

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

2. Execute command below to install the library.

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

Successful installation, without error prompts, is shown below:

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



Next, execute the code of this project.

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

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 executable file "ADC".

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

3. Then run the generated file "ADC".

```
sudo ./ADC
```

After the program is executed, adjust the potentiometer, then the terminal will print out the potentiometer voltage value and the converted digital content.

```
ADC value : 135 , Voltage : 1.75V
ADC value : 135 , Voltage : 1.75V
ADC value : 136 , Voltage : 1.76V
ADC value : 141 , Voltage : 1.82V
ADC value : 144 , Voltage : 1.86V
ADC value : 146 , Voltage : 1.89V
ADC value : 148 , Voltage : 1.92V
ADC value : 149 , Voltage : 1.93V
ADC value : 149 , Voltage : 1.93V
ADC value : 144 , Voltage : 1.86V
ADC value : 143 , Voltage : 1.85V
ADC value : 143 , Voltage : 1.85V
ADC value : 142 , Voltage : 1.84V
ADC value : 141 , Voltage : 1.82V
```

The following is the code:

```
1 #include <wiringPi.h>
2 #include <wiringPiI2C.h>
3 #include <stdio.h>
4 #include <ADCDevice.hpp>
5
6 ADCDevice *adc; // Define an ADC Device class object
7
8 int main(void) {
9     adc = new ADCDevice();
10    printf("Program is starting ... \n");
11
12    if(adc->detectI2C(0x48)){ // Detect the pcf8591.
13        delete adc;
14        adc = new PCF8591(); // If detected, create an instance of PCF8591.
15    }
16    else if(adc->detectI2C(0x4b)){// Detect the ads7830
17        delete adc;
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
27 while(1){
28     int adcValue = adc->analogRead(0); //read analog value of A0 pin
29     float voltage = (float)adcValue / 255.0 * 3.3; // Calculate voltage
30     printf("ADC value : %d , \tVoltage : %.2fV\n", adcValue, voltage);
31     delay(100);
32 }
33 }
```

In this code, a custom class library "ADCDevice" is used. It contains the method of using the ADC module in this product, through which the ADC module can be used easily and quickly. In the code, first create a class pointer adc, and then point the pointer to an instantiated object.

```

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

Then use the member function detectI2C(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 in the circuit is. When the correct module is detected, the pointer adc will point to the address of the object, and the previously pointed content is 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 pointer 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 printed 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, a custom module, needs to be installed.

1. Use cd command to enter folder of ADCDevice.

```
cd ~/Freenove_Kit/Libs/Python-Libs/ADCDevice-1.0.2
```

2. Unzip the file.

```
tar zxvf ADCDevice-1.0.2.tar.gz
```

3. Enter the unzipped folder.

```
cd ADCDevice-1.0.2
```

4. Install.

```
sudo python setup.py install
```

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 analyze the code.

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, adjust the potentiometer, then the terminal will print out the potentiometer voltage value and the converted digital content.

```
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
8     if(adc.detectI2C(0x48)): # Detect the pcf8591.
9         adc = PCF8591()
```

```

10 elif(adc.detectI2C(0x4b)): # Detect the ads7830
11     adc = ADS7830()
12 else:
13     print("No correct I2C address found, \n"
14         "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
15         "Program Exit. \n");
16     exit(-1)
17
18 def loop():
19     while True:
20         value = adc.analogRead(0)    # read the ADC value of channel 0
21         voltage = value / 255.0 * 3.3 # calculate the voltage value
22         print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
23         time.sleep(0.1)
24
25 def destroy():
26     adc.close()
27
28 if __name__ == '__main__': # Program entrance
29     print ('Program is starting ... ')
30     try:
31         setup()
32         loop()
33     except KeyboardInterrupt: # Press ctrl-c to end the program.
34         destroy()

```

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

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

Then in setup(), use detectI2C(addr), the member function of ADCDevice, 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 in the circuit is. 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.

	<pre> 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" </pre>
--	--

```
"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 printed 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

That is System Management Bus. This module defines an object type that allows SMBus transactions on hosts running the Linux kernel. The host kernel must have I2C support, 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 function 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. All ADC module classes are its derived classes.

```
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 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
```

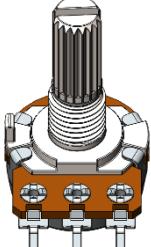
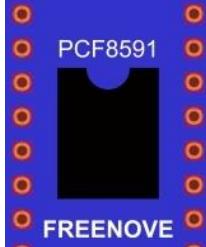
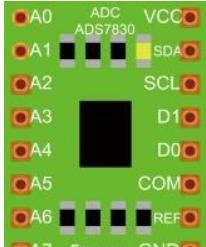
Chapter 8 Potentiometer & LED

We have learned how to use ADC and PWM before. In this chapter, we learn to control the brightness of LED through a potentiometer.

Project 8.1 Soft Light

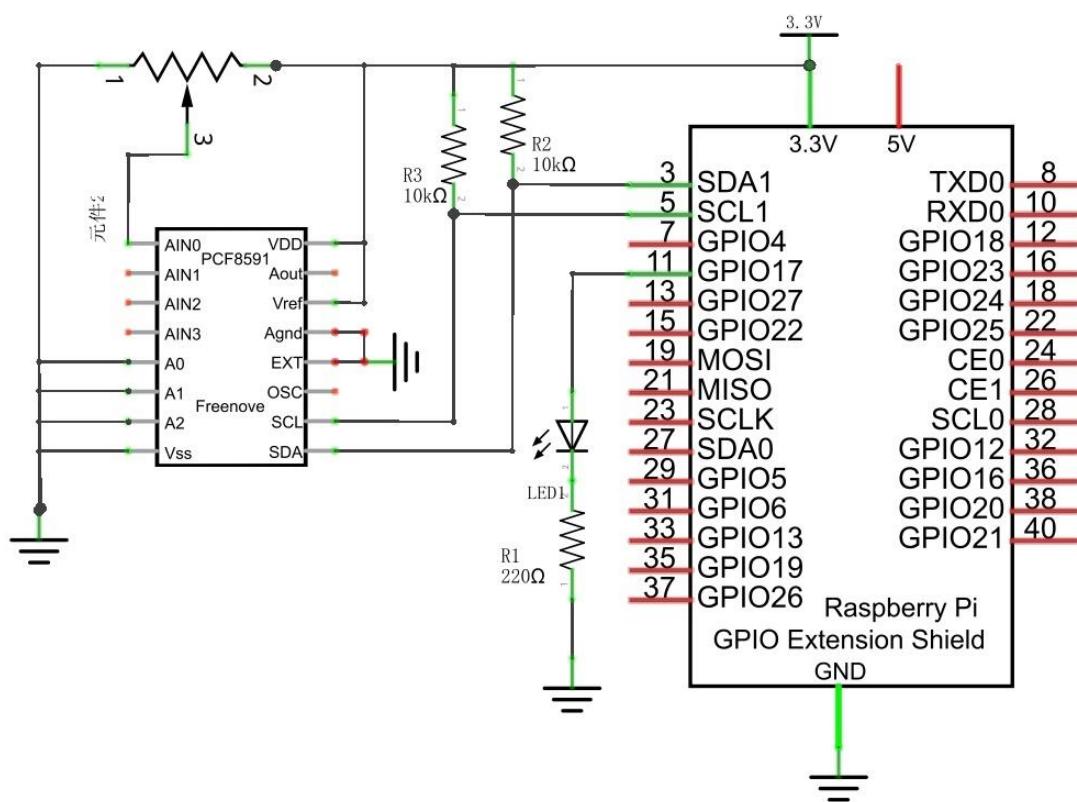
In this project, we will make a soft light. Use ADC Module to read ADC value of potentiometers and map it to duty cycle ratio of PWM used to control the brightness of LED. Then you can make the LED brightness changed by adjusting the potentiometer.

Component List

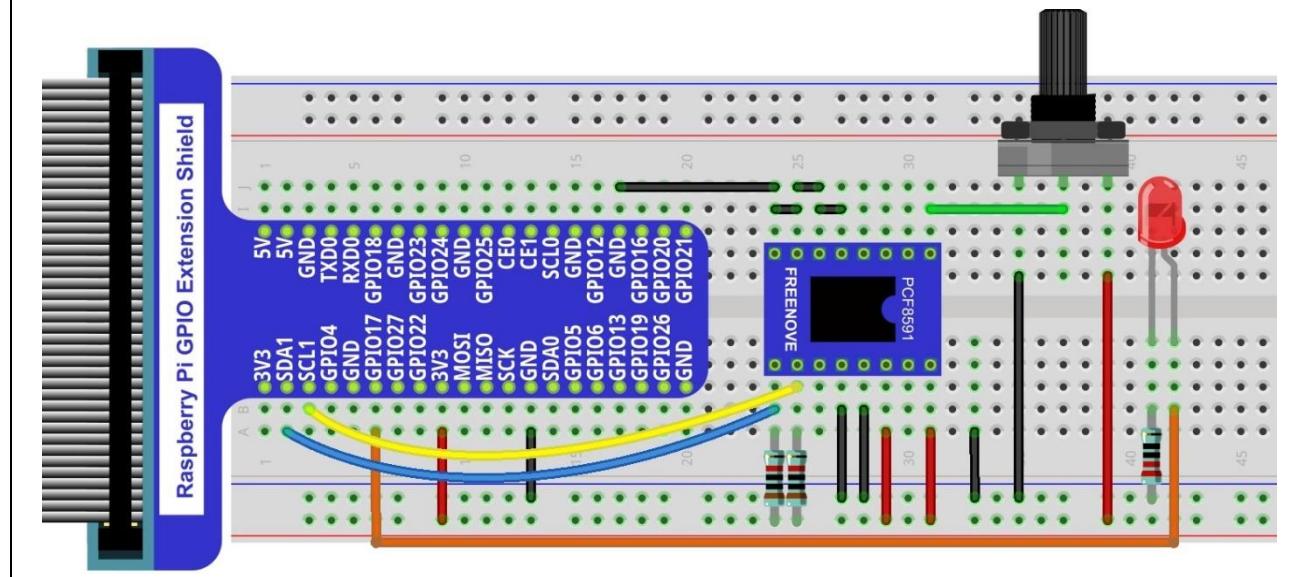
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M x17
Rotary potentiometer x1 	ADC module x1  Or 

Circuit with PCF8591

Schematic diagram

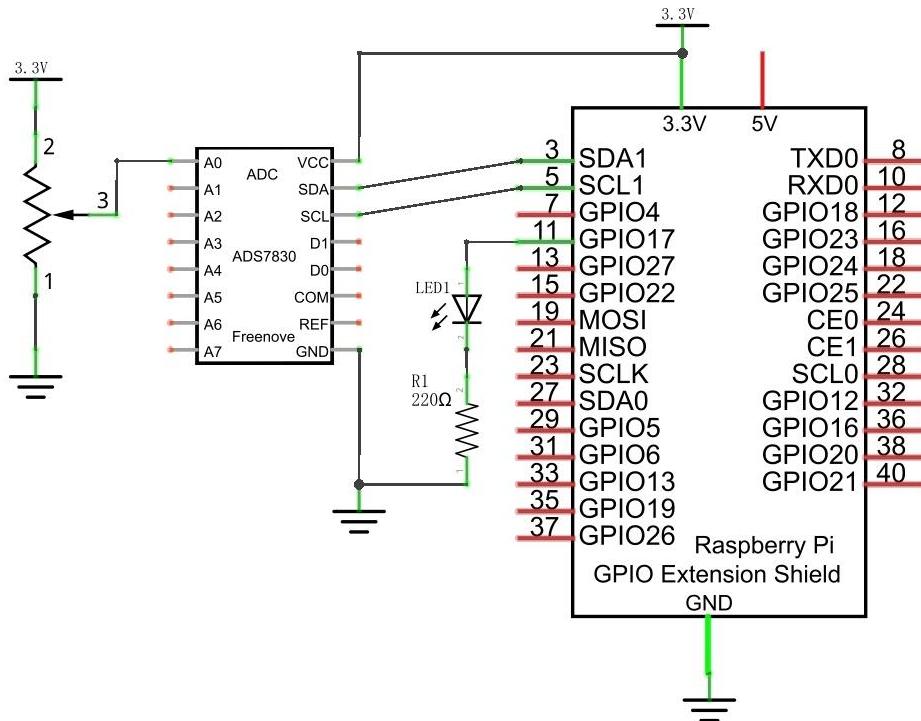


Hardware connection

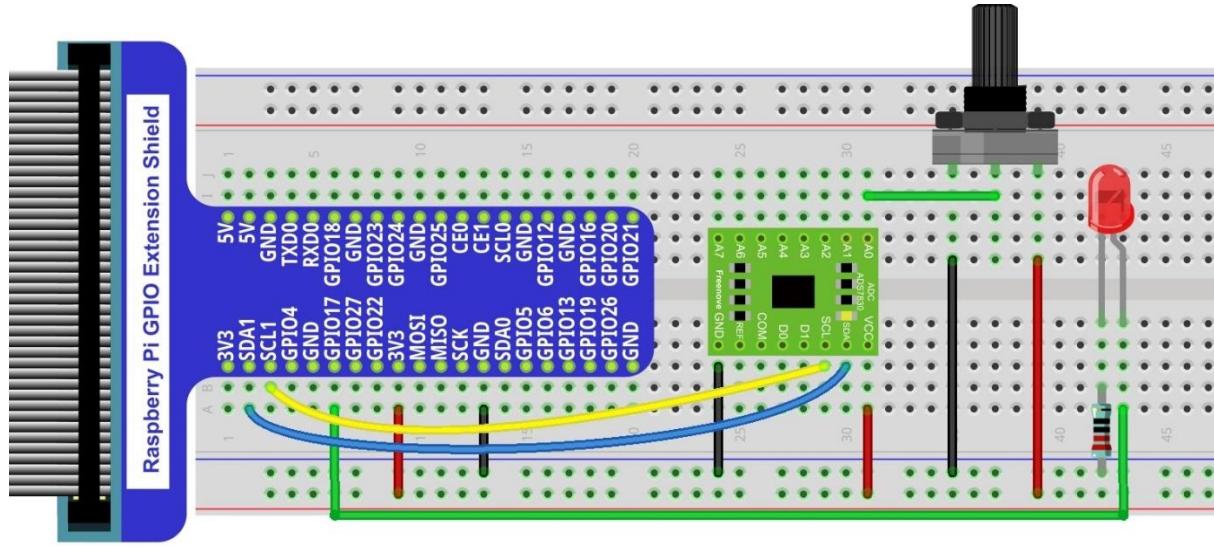


Circuit with ADS7830

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 analyze the code.

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, adjust the potentiometer, then the terminal window will print out the voltage value of the potentiometer and the converted digital quantity. And brightness of LED will be changed consequently.

The following is the code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define ledPin 0
7
8 ADCDevice *adc; // Define an ADC Device class object
9
10 int main(void) {
11     adc = new ADCDevice();
12     printf("Program is starting ... \n");
13
14     if(adc->detectI2C(0x48)){ // Detect the pcf8591.
15         delete adc;           // Free previously pointed memory
16         adc = new PCF8591(); // If detected, create an instance of PCF8591.
17     }
18     else if(adc->detectI2C(0x4b)){// Detect the ads7830
19         delete adc;           // Free previously pointed memory
20         adc = new ADS7830(); // If detected, create an instance of ADS7830.
21     }
22     else{
23         printf("No correct I2C address found, \n"
24             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
25     }
```

```
25     "Program Exit. \n");
26     return -1;
27 }
28 wiringPiSetup();
29 softPwmCreate(ledPin, 0, 100);
30 while(1){
31     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 ADC value of potentiometers and map it to 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 move on.

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

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, adjust the potentiometer, then the terminal window will print out the voltage value of the potentiometer and the converted digital quantity. And brightness of LED will be changed consequently.

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.BCM)
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__':
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 duty cycle of 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
```

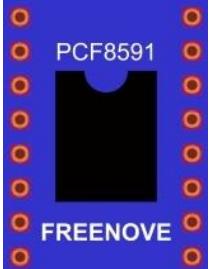
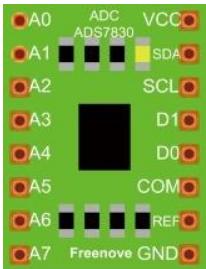
Chapter 9 Photoresistor & LED

In this chapter, we will learn how to use photoresistor.

Project 9.1 NightLamp

Photoresistor is very sensitive to illumination strength. So we can use this feature to make a nightlamp, when ambient light gets darker, LED will become brighter automatically, and when the ambient light gets brighter, LED will become darker automatically.

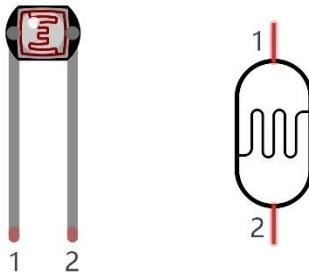
Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M			
Photoresistor x1 	ADC module x1  Or 	10kΩ x3 	220Ω x1 	LED x1 

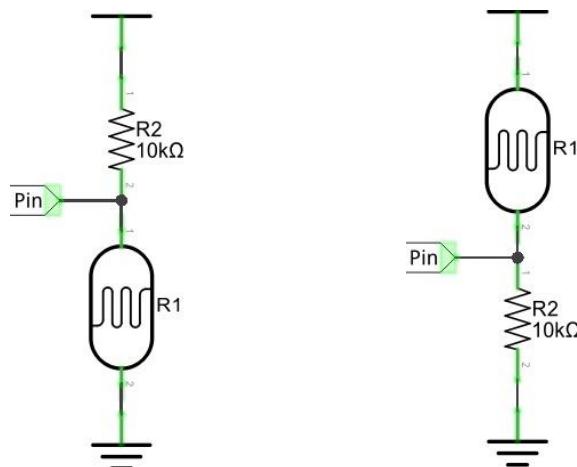
Component knowledge

Photoresistor

Photoresistor is a light sensitive resistor. When the strength that light casts onto the photoresistor surface is not the same, resistance of photoresistor will change. With this feature, we can use photoresistor to detect light intensity. Photoresistor and symbol are as follows.



The circuit below is often used to detect the change of photoresistor resistance:

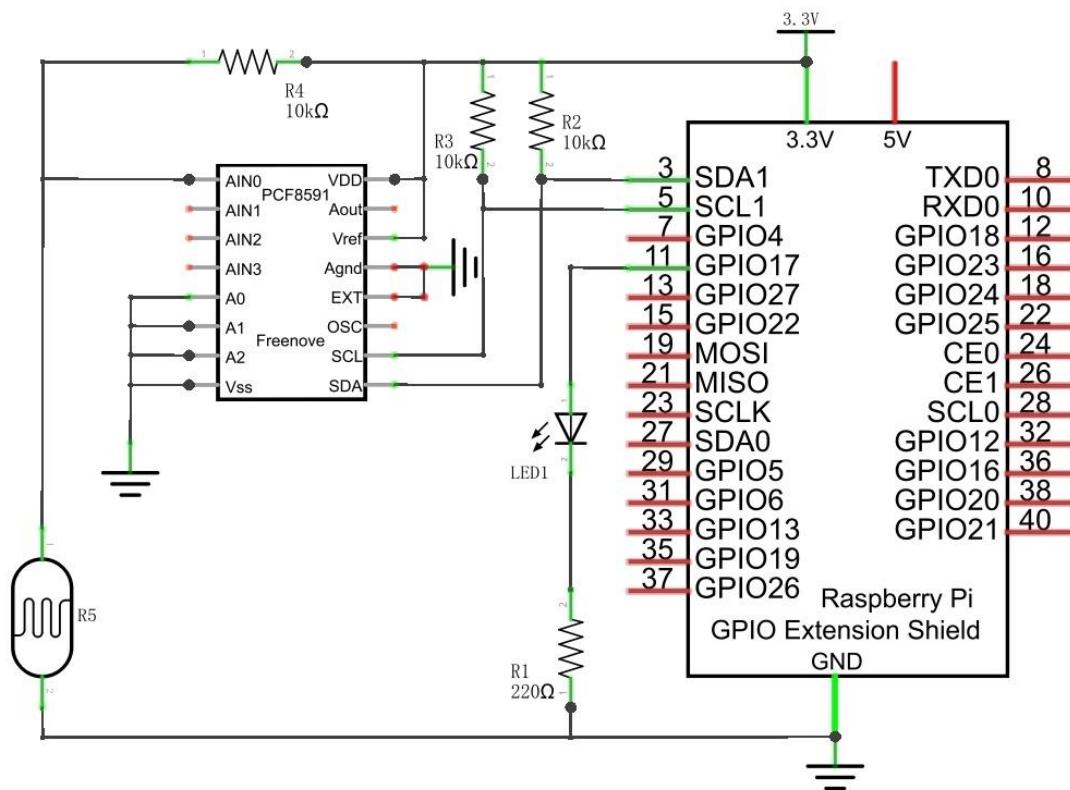


In the above circuit, when photoresistor resistance changes due to light intensity, voltage between photoresistor and resistor R1 will change, so light's intensity can be obtained by measuring the voltage.

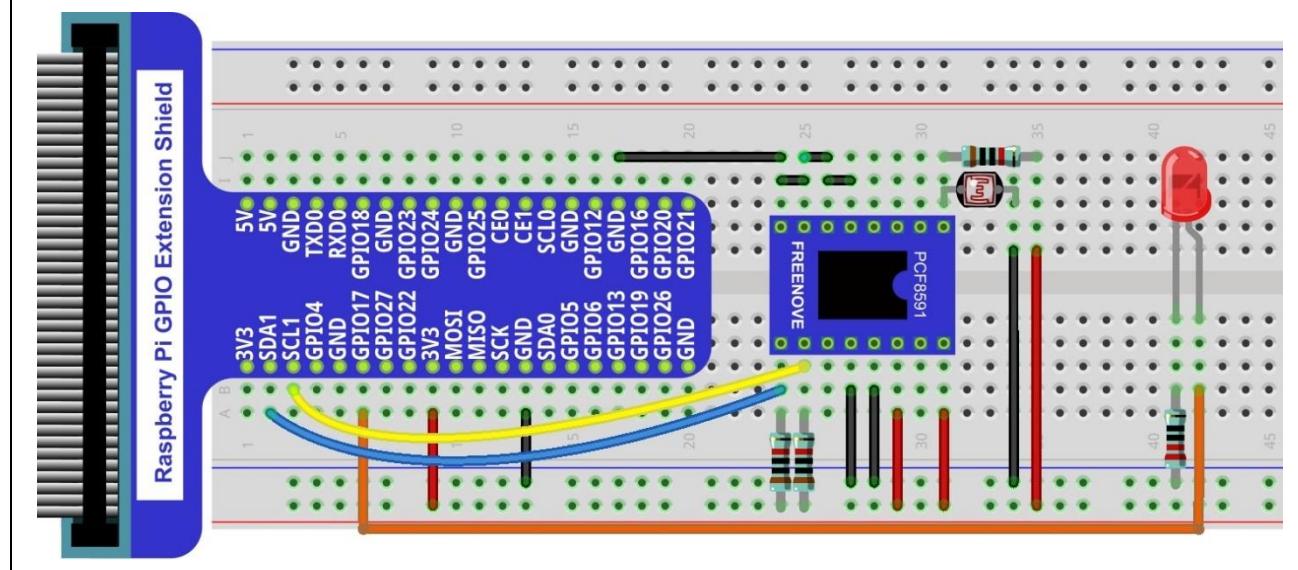
Circuit with PCF8591

The circuit of this experiment is similar to the project Soft light. The only difference is that the input signal of the AIN0 pin of ADC is changed from a potentiometer to combination of a photoresistor and a resistor.

Schematic diagram



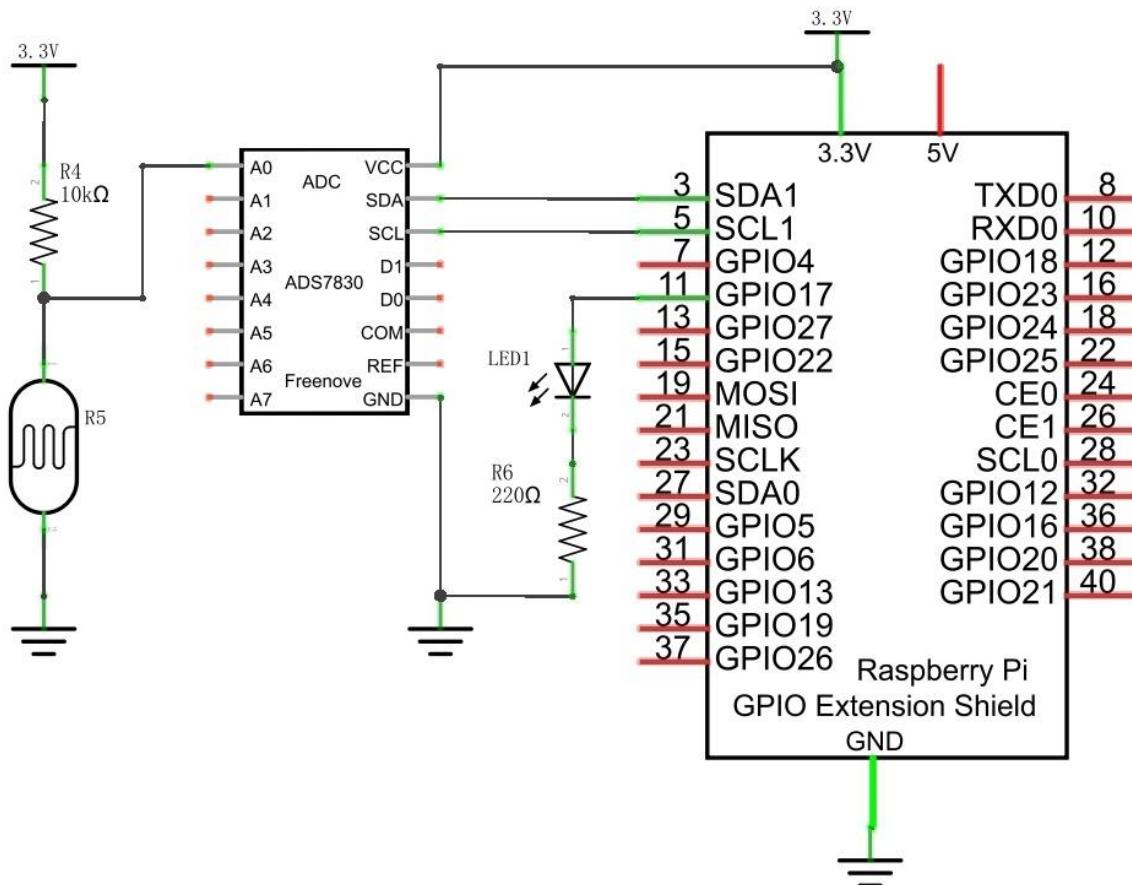
Hardware connection



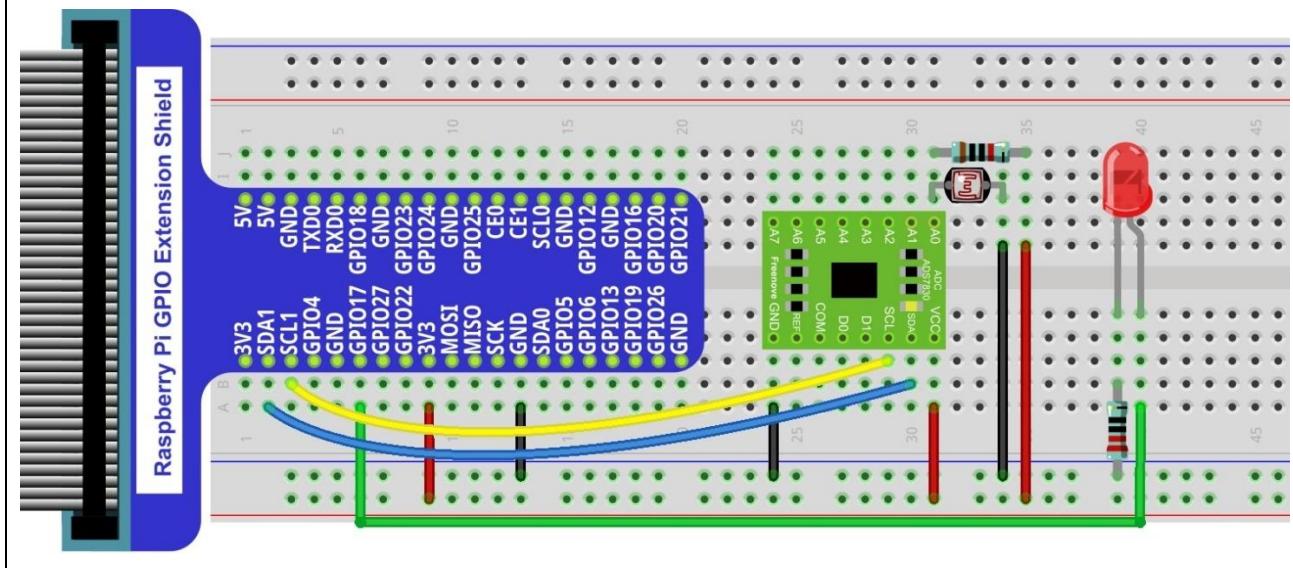
Circuit with ADS7830

The circuit of this project is similar to the project Soft light. The only difference is that the input signal of the AIN0 pin of ADC Module is changed from a potentiometer to 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 of this project is identical with the one in last chapter logically.

C Code 9.1.1 Nightlamp

If you did not [configure I2C](#), please refer to [Chapter 7](#). If you did, please move on.

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

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, when you cover the photosensitive resistance or make a flashlight toward the photoresistor, the brightness of LED will be enhanced or weakened. And the terminal window will print out the current input voltage value of ADC module A0 pin and the converted digital quantity.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define ledPin 0
7
8 ADCDevice *adc; // Define an ADC Device class object
9
10 int main(void) {
11     adc = new ADCDevice();
12     printf("Program is starting ... \n");
13
14     if(adc->detectI2C(0x48)){ // Detect the pcf8591.
15         delete adc;           // Free previously pointed memory
16         adc = new PCF8591(); // If detected, create an instance of PCF8591.
17     }
18     else if(adc->detectI2C(0x4b)){// Detect the ads7830
19         delete adc;           // Free previously pointed memory
20         adc = new ADS7830(); // If detected, create an instance of ADS7830.
21     }
22     else{
23         printf("No correct I2C address found, \n"
24             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
25             "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 move on.

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

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, when you cover the photosensitive resistance or make a flashlight toward the photoresistor, the brightness of LED will be enhanced or weakened. And the terminal window will print out the current input voltage value of ADC module A0 pin and the converted digital quantity.

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 from ADCDevice import *
4
5 ledPin = 11 # define ledPin
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)    # set ledPin to OUTPUT mode
22    GPIO.output(ledPin,GPIO.LOW)
23
24    p = GPIO.PWM(ledPin,1000) # set PWM Frequnce to 1kHz
25    p.start(0)
26
27 def loop():
28     while True:
29         value = adc.analogRead(0)      # read the ADC value of channel 0
30         p.ChangeDutyCycle(value*100/255)
31         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()
```

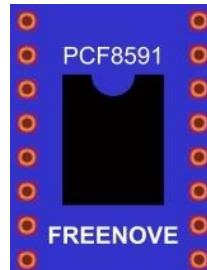
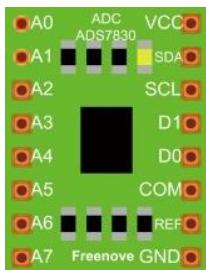
Chapter 10 Thermistor

In this chapter, we will learn another new kind of resistor, thermistor.

Project 10.1 Thermometer

The resistance of thermistor will be changed with temperature change. So we can make a thermometer according to this feature.

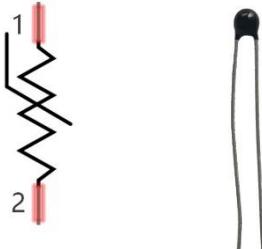
Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M
Thermistor x1	ADC module x1
	 or 

Component knowledge

Thermistor

Thermistor is a temperature sensitive resistor. When the temperature changes, resistance of thermistor will change. With this feature, we can use thermistor to detect temperature intensity. Thermistor and symbol are as follows.



The relationship between resistance value and temperature of thermistor is:

$$R_t = R \cdot \exp[B \cdot (1/T_2 - 1/T_1)]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

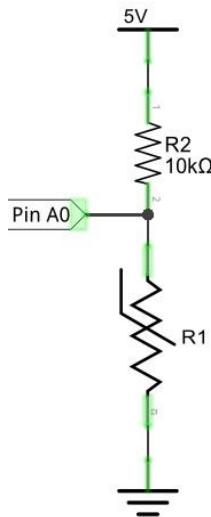
EXP[n] is nth power of E;

B is for thermal index;

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

Parameters of the thermistor we use is: B=3950, R=10k, T1=25.

The circuit connection method of the thermistor is similar to photoresistor, like the following method:



We can use the value measured by ADC converter to obtain resistance value of thermistor, and then can use the formula to obtain the temperature value.

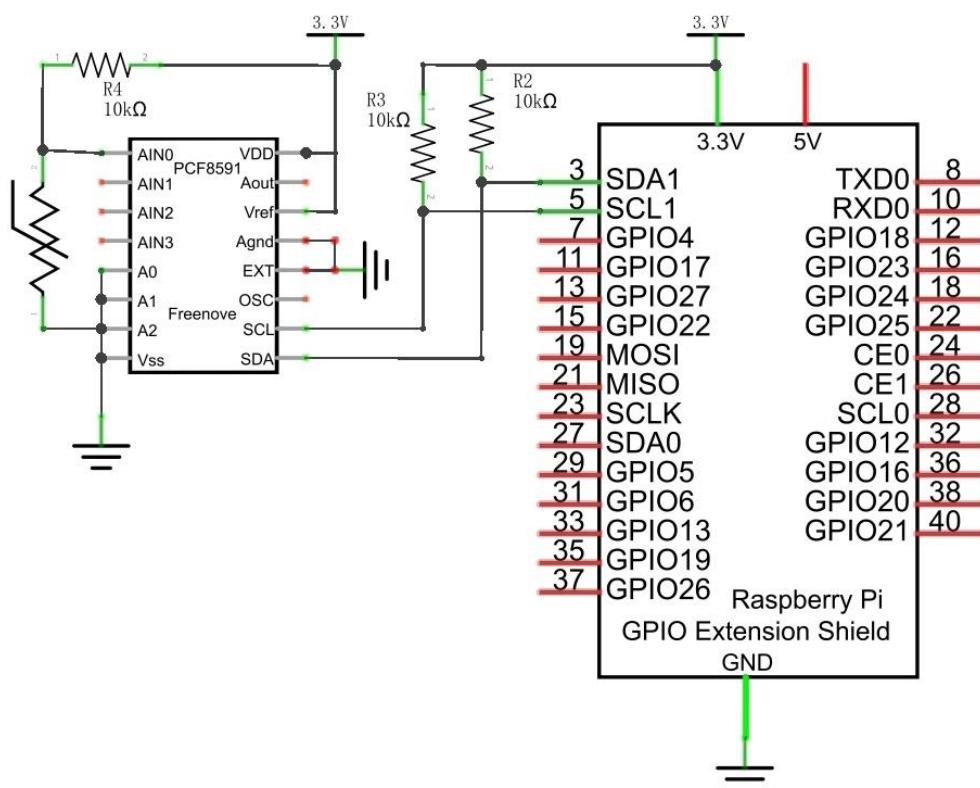
Consequently, the temperature formula can be concluded:

$$T_2 = 1 / (1/T_1 + \ln(R_t/R)/B)$$

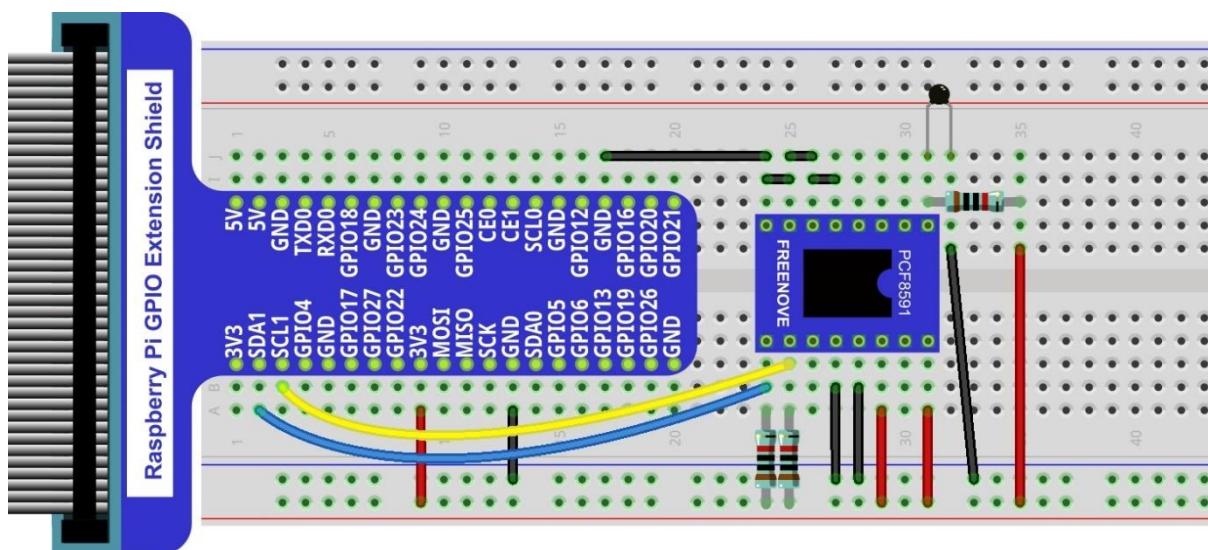
Circuit with PCF8591

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

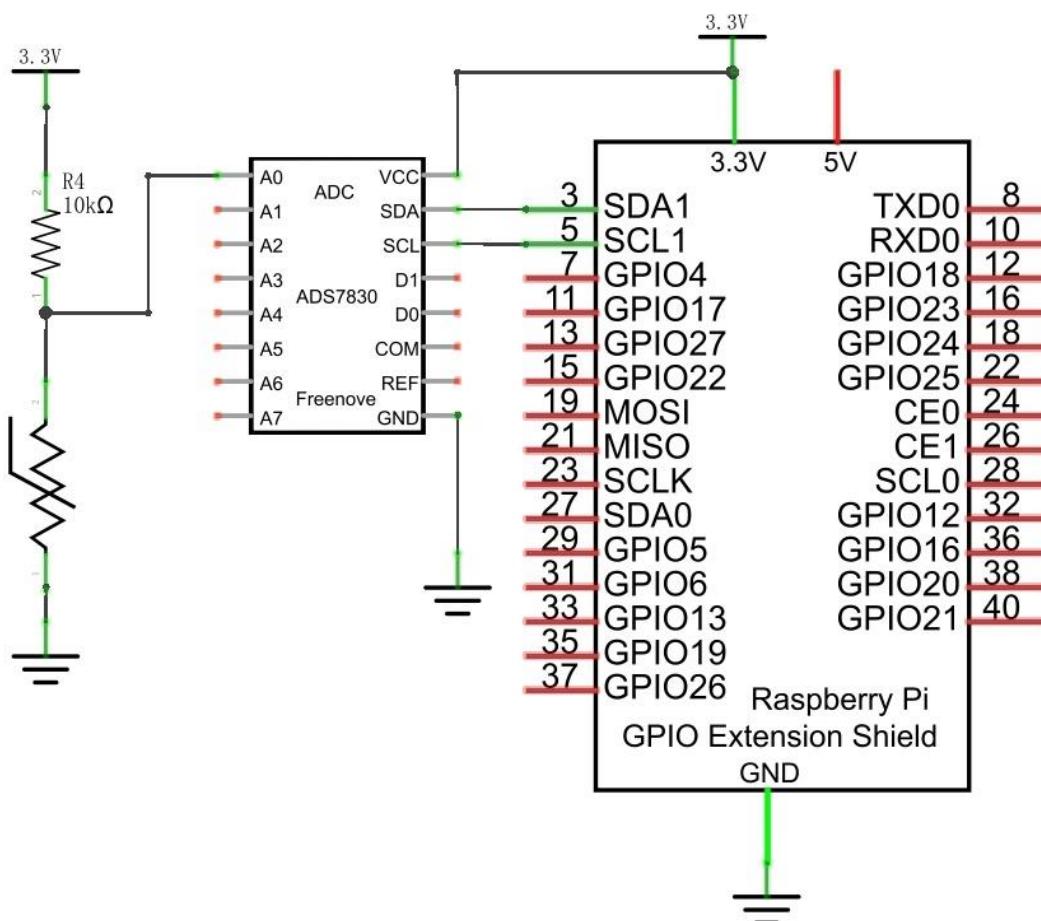




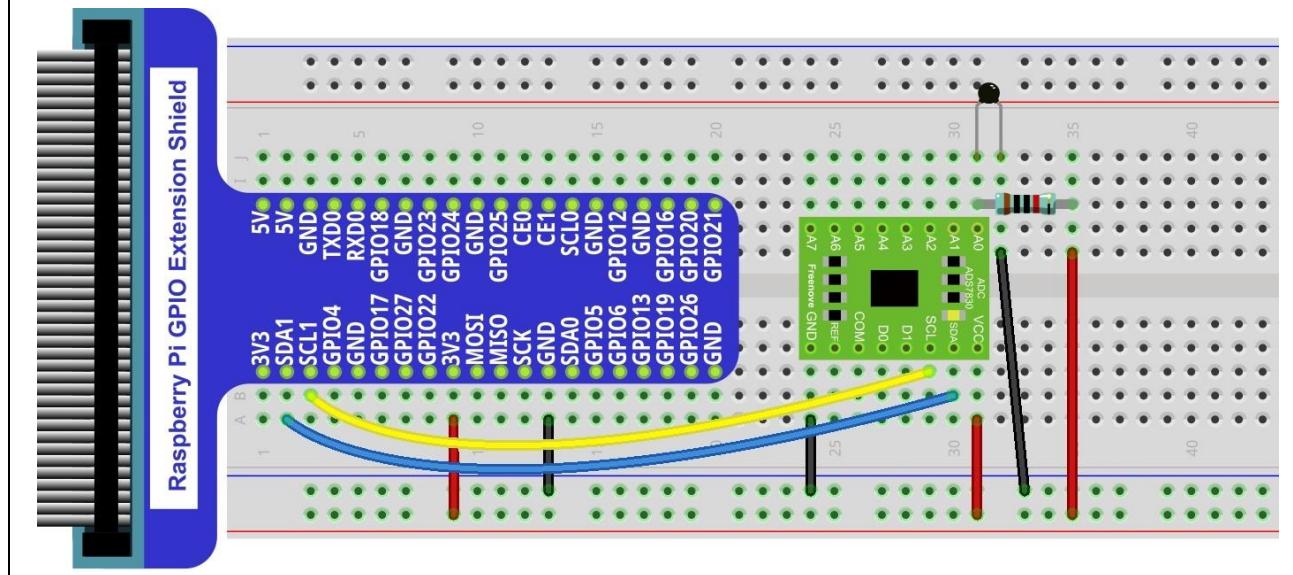
Circuit with ADS

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



Code

In this project code, the ADC value is still needed to be read, and the difference 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 move on.

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

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_Termometer
```

- 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 print out the current ADC value, voltage value and temperature value. Try to pinch the thermistor (do not touch pin) with hand lasting for a while, then the temperature value will be increased.

The following is the code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <math.h>
4 #include <ADCDevice.hpp>
5
6 ADCDevice *adc; // Define an ADC Device class object
7
8 int main(void) {
9     adc = new ADCDevice();
10    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
thermistor
31         float tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0); //calculate temperature
(Kelvin)
32         float tempC = tempK -273.15; //calculate temperature (Celsius)
33         printf("ADC value : %d ,\tVoltage : %.2fV,
34 \tTemperature : %.2fC\n",adcValue,voltage,tempC);
35         delay(100);
36     }
37     return 0;
38 }
```

In the code, read the ADC value of ADC module A0 port, and then calculate the voltage and the resistance of thermistor according to Ohms law. Finally, calculate the current temperature. according to the front formula.

Python Code 10.1.1 Thermometer

If you did not [configure I2C](#), please refer to [Chapter 7](#). If you did, please move on.

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

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

1. Use cd command to enter 10.1.1_Termometer directory of Python code.

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

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

```
python Thermometer.py
```

After the program is executed, the terminal window will print out the current ADC value, voltage value and temperature value. Try to pinch the thermistor (do not touch pin) with hand lasting for a while, then the temperature value will be increased.

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

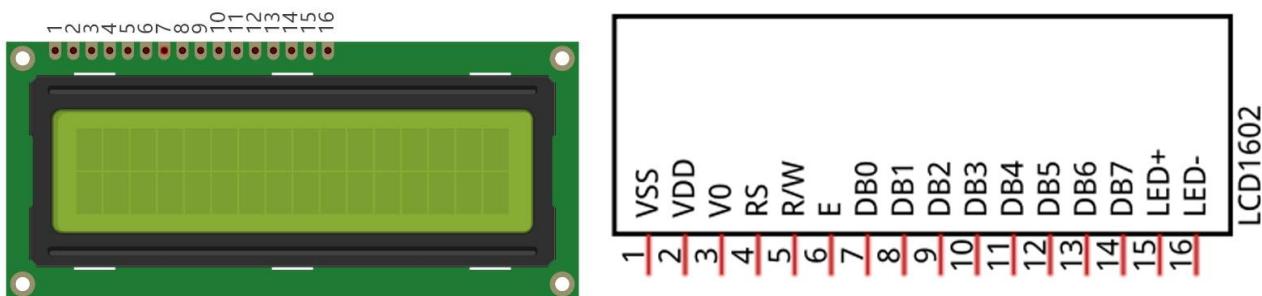
In the code, read the ADC value of ADC module A0 port, and then calculate the voltage and the resistance of thermistor according to Ohms law. Finally, calculate the current temperature. according to the front formula.

Chapter 11 LCD1602

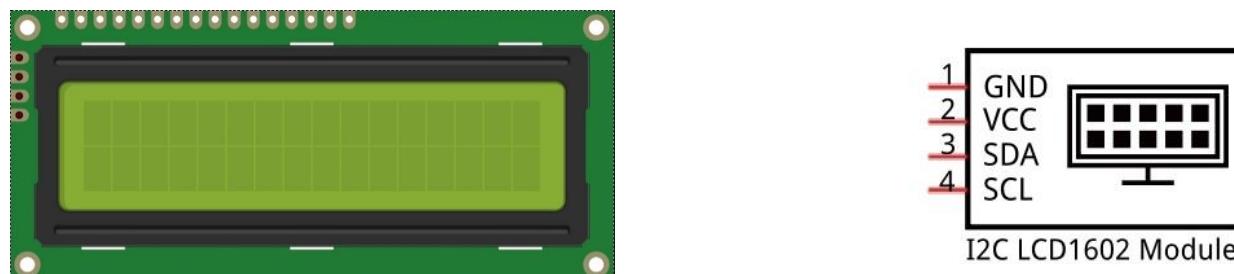
In this chapter, we will learn a display screen, LCD1602.

Project 11.1 I2C LCD1602

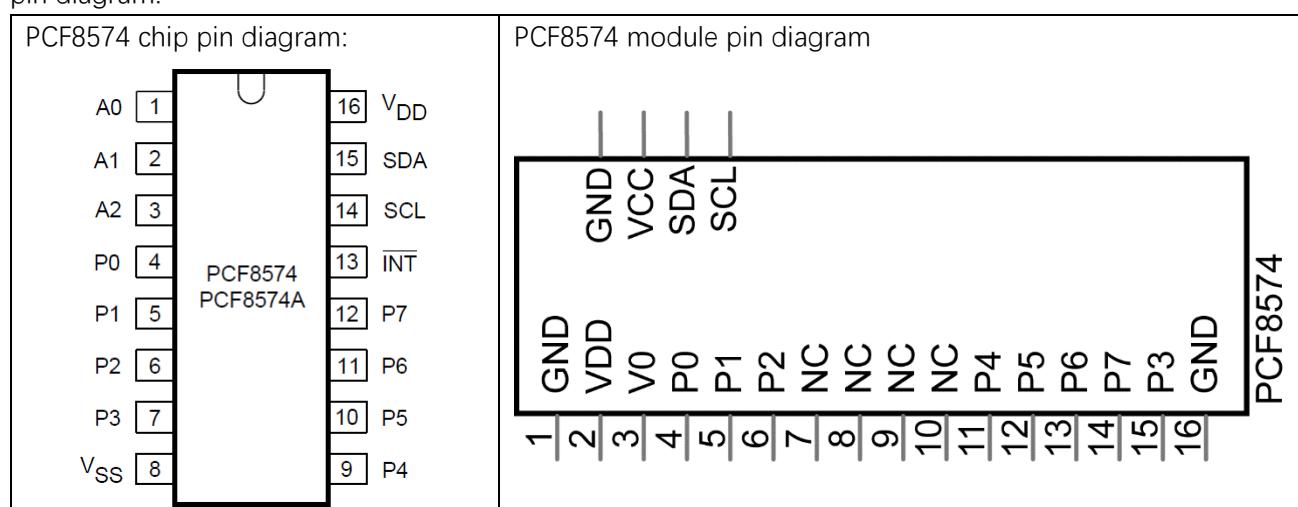
LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 display screen, and its circuit pin diagram:



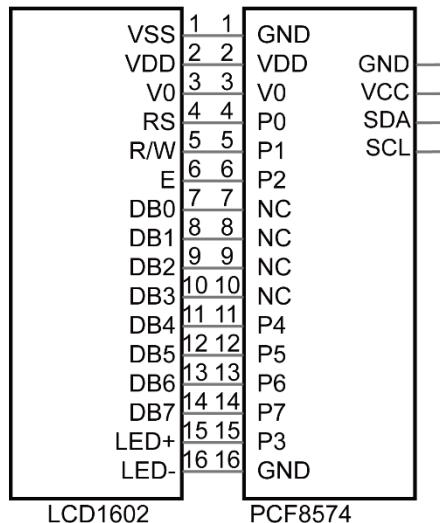
I2C LCD1602 integrates a I2C interface, which connects the serial-input ¶llel-output module to LCD1602. We just use 4 lines to operate LCD1602 easily.



The serial-to-parallel chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F), and you can view all the RPI bus on your I2C device address through command "i2cdetect -y 1" to. (refer to the "configuration I2C" section below) below is the PCF8574 pin schematic diagram and the block pin diagram:



PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



So, we can use just 4 pins to control LCD1602 with 16 pins easily through I²C interface.

In this project, we will use I²CLCD1602 to display some static characters and dynamic variables.

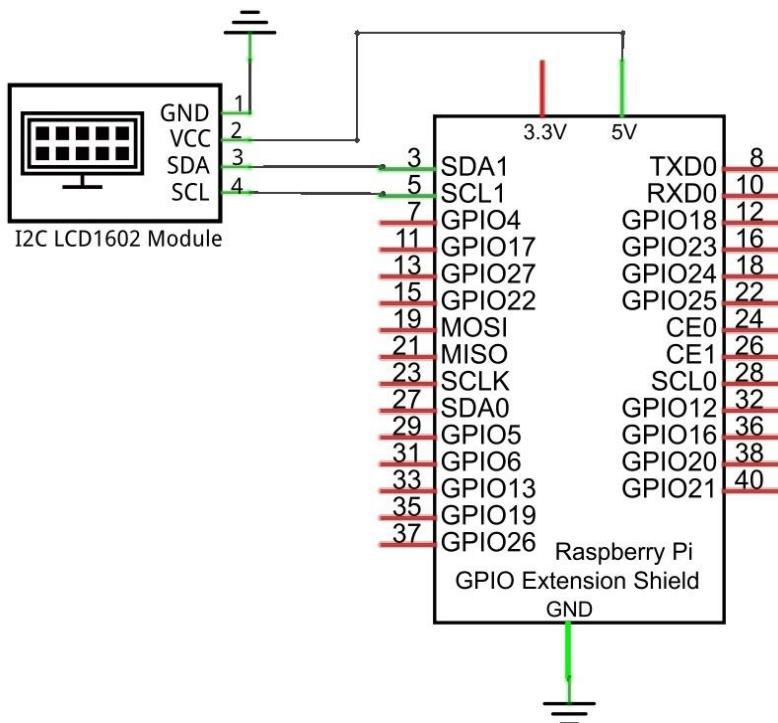
Component List

Raspberry Pi (with 40 GPIO) x1	Jumper
GPIO Extension Board & Wire x1	
BreadBoard x1	
I ² C LCD1602 Module x1	

Circuit

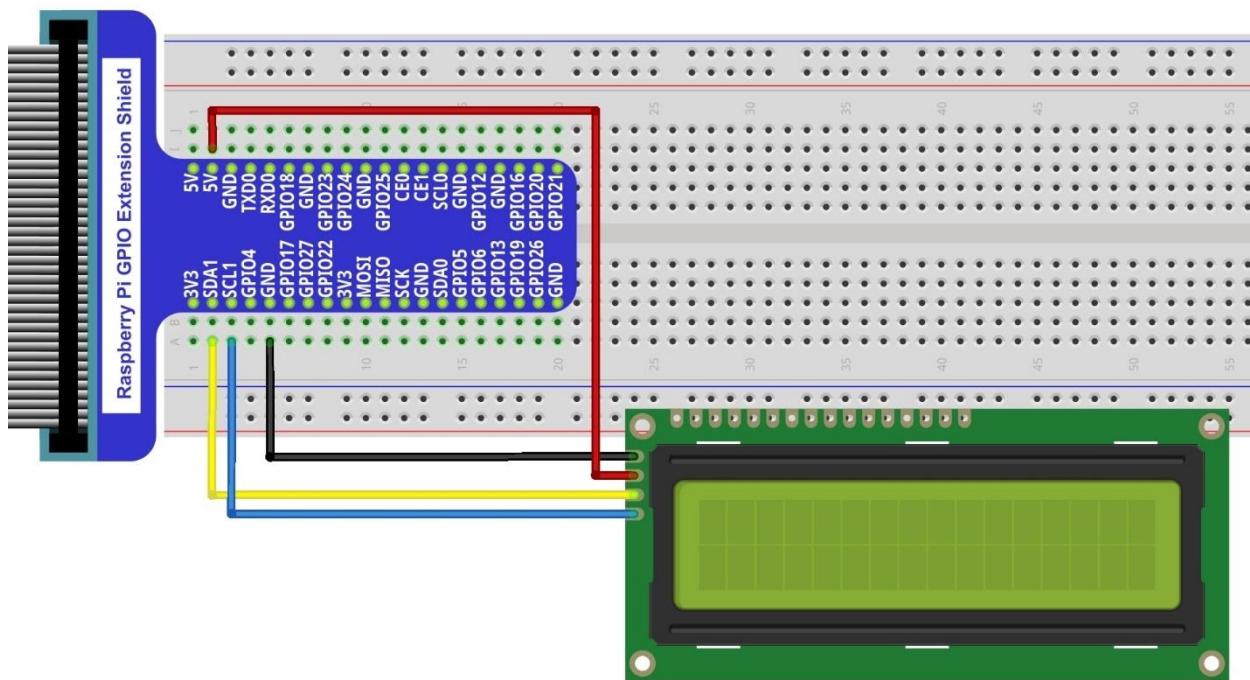
Note that the power supply for I2CLCD1602 in this circuit is 5V.

Schematic diagram



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

It is necessary to configure i2c and install Sumbus first on [chapter 7](#).



Code

This code will get the CPU temperature and system time of RPi, display them on LCD1602.

C Code 11.1.1 I2CLCD1602

If you did not [configure I2C and install Sumbus](#), please refer to [Chapter 7](#). If you did, please move on.

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

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. Open the file I2CLCD1602.c, and find the macro definition "pcf8574_address". If your serial-to-parallel module uses chip PCF8574T, set the macro "pcf8574_address" value to 0x27. If your serial-to-parallel module uses chip PCF8574AT, set the macro "pcf8574_address" value to 0x3F.

```
14 // #define pcf8574_address 0x27      // default I2C address of Pcf8574
15 #define pcf8574_address 0x3F      // default I2C address of Pcf8574A
```

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

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

4. Then run the generated file "I2CLCD1602".

```
sudo ./ I2CLCD1602
```

After the program is executed, LCD1602 screen will display current CPU temperature and system time.

If there is no display or the display is not clear, rotate white knob on back of LCD to adjust the contrast of LCD1602 until the screen can display clearly.



The following is the program code:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <wiringPi.h>
4 #include <pcf8574.h>
5 #include <lcd.h>
6 #include <time.h>
7
8 // #define pcf8574_address 0x27      // default I2C address of Pcf8574
9 #define pcf8574_address 0x3F      // default I2C address of Pcf8574A
10 #define BASE 64      // BASE any number above 64
11 // Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
```

```
12 #define RS      BASE+0
13 #define RW      BASE+1
14 #define EN      BASE+2
15 #define LED     BASE+3
16 #define D4      BASE+4
17 #define D5      BASE+5
18 #define D6      BASE+6
19 #define D7      BASE+7
20
21 int lcdhd;// used to handle LCD
22 void printCPUtemperature() {// sub function used to print CPU temperature
23     FILE *fp;
24     char str_temp[15];
25     float CPU_temp;
26     // CPU temperature data is stored in this directory.
27     fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
28     fgets(str_temp,15,fp);      // read file temp
29     CPU_temp = atof(str_temp)/1000.0; // convert to Celsius degrees
30     printf("CPU's temperature : %.2f \n",CPU_temp);
31     lcdPosition(lcdhd,0,0);      // set the LCD cursor position to (0,0)
32     lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp); // Display CPU temperature on LCD
33     fclose(fp);
34 }
35 void printDataTime() //used to print system time
36     time_t rawtime;
37     struct tm *timeinfo;
38     time(&rawtime); // get system time
39     timeinfo = localtime(&rawtime); //convert to local time
40     printf("%s \n",asctime(timeinfo));
41     lcdPosition(lcdhd,0,1); // set the LCD cursor position to (0,1)
42     lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
43 //Display system time on LCD
44 }
45 int main(void) {
46     int i;
47
48     printf("Program is starting ... \n");
49
50     wiringPiSetup();
51
52     pcf8574Setup(BASE,pcf8574_address); //initialize PCF8574
53     for(i=0;i<8;i++) {
54         pinMode(BASE+i,OUTPUT); //set PCF8574 port to output mode
55     }
```

```

56     digitalWrite(LED, HIGH);      //turn on LCD backlight
57     digitalWrite(RW, LOW);       //allow writing to LCD
58     lcdhd = lcdInit(2, 16, 4, RS, EN, D4, D5, D6, D7, 0, 0, 0, 0); // initialize LCD and return "handle"
59 used to handle LCD
60     if(lcdhd == -1) {
61         printf("lcdInit failed !");
62         return 1;
63     }
64     while(1) {
65         printCPUtemperature(); //print CPU temperature
66         printDataTime();      // print system time
67         delay(1000);
68     }
69     return 0;
70 }
```

It can be seen from the code that PCF8591 and PCF8574 have a lot of similarities, they are through the I2C interface to expand the GPIO RPI. First defines the I2C address of the PCF8574 and the Extension of the GPIO pin, which is connected to the GPIO pin of the LCD1602.

```

//#define pcf8574_address 0x27      // default I2C address of Pcf8574
#define pcf8574_address 0x3F      // default I2C address of Pcf8574A
#define BASE 64          // BASE is not less than 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.

```

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 (namely, can be write) according to requirements of this function. The return value of the function called "Handle" is used to handle LCD1602 next.

```

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 called before you use any other LCD functions.

Rows and **cols** are the rows and columns on the display (e.g. 2, 16 or 4,20). **Bits** is the number of bits wide on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the displays RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the Pi 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 parameters)

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 CPU temperature and the time. First look at subfunction `printCPUTemperature()`. The CPU temperature data is stored in the `"/sys/class/thermal/thermal_zone0/temp"` file. We need read contents of the file, and converts it to temperature value stored in variable `CPU_temp`, and use `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 `printf` formatting commands.

Next is subfunction `printDataTime()` used to print system time. First, got the standard time and store it into variable `rawtime`, and then converted it to the local time and tore it into `timeinfo`, and finally display the time information on LCD1602.

```
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 Sumbus](#), please refer to [Chapter 7](#). If you did, please move on.

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

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, LCD1602 screen will display current CPU temperature and system time.

If there is no display or the display is not clear, rotate white knob on back of LCD to adjust the contrast of LCD1602 until the screen can display clearly.



The following is the program code:

```

1  from PCF8574 import PCF8574_GPIO
2  from Adafruit_LCD1602 import Adafruit_CharLCD
3
4  from time import sleep, strftime
5  from datetime import datetime
6
7  def get_cpu_temp():      # get CPU temperature and store it into file
8      "/sys/class/thermal/thermal_zone0/temp"
9      tmp = open('/sys/class/thermal/thermal_zone0/temp')
10     cpu = tmp.read()
11     tmp.close()
12     return '{:.2f}'.format(float(cpu)/1000) + ' C'
13
14 def get_time_now():      # get system time

```

```

15     return datetime.now().strftime('%H:%M:%S')
16
17 def loop():
18     mcp.output(3, 1)      # turn on LCD backlight
19     lcd.begin(16, 2)      # set number of LCD lines and columns
20     while(True):
21         #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 file are stored in the same directory, and neither of them is dispensable. Please do not delete. PCF8574.py is used to provide I2C communication mode and operation method of some port for RPi and PCF8574 chip. Adafruit module Adafruit_LCD1602.py is used to provide some function operation method for LCD1602.

In the code, first get the object used to operate PCF8574 port, then get the object used to operate 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 positive pole of LCD1602 backlight. Then in the loop () function, use of mcp.output(3,1) to turn on LCD1602 backlight, and set 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 cycle, 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 WebIOPi & IOT

In this chapter, we will learn how to use GPIO to control RPi through remote network and how to build a WebIOPi service on the RPi.

“IOT” is Internet of Things. The development of IOT will greatly change our habits and make our lives more convenient and efficient.

“WebIOPi” is the Raspberry Pi Internet of Things Framework. After configuration for WebIOPi on your RPi is completed, you can use web browser on mobile phones, computers and other equipments to control, debug and use RPi GPIO conveniently. It also supports many commonly used communication protocol, such as serial, I2C, SPI, etc., and a lot of equipments, like AD/DA converter pcf8591 used before and so on. Then on this basis, through adding some peripheral circuits, you can create your own smart home.

For more details about WebIOPi, please refer to: <http://webiopi.trouch.com/>

Project 12.1 Remote LED

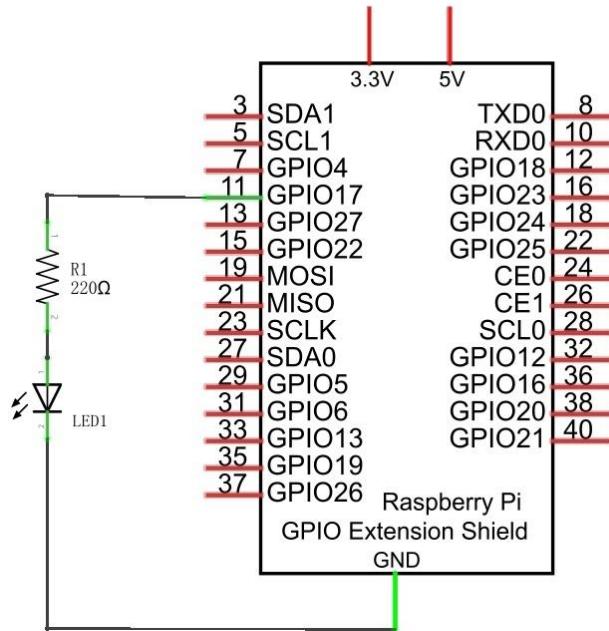
In this experiment, we need build a WebIOPi service, and then control the RPi GPIO to control a LED through Web browser of phone or PC.

Component List

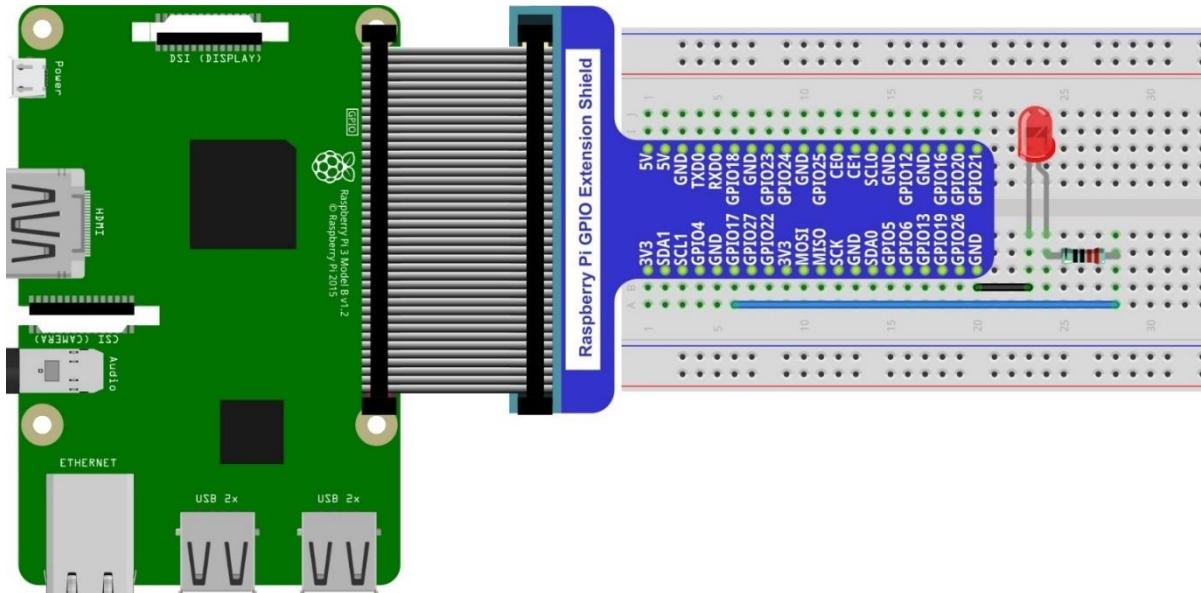
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED x1	Resistor 220Ω x1
Jumper M/M x2 		

Circuit

Schematic diagram



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



Build WebIOPi Service Framework

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 (until 2016-6-27) WebIOPi is 0.7.1. So, you may have some problems in use. We will explain these problems and provide the solution in the following installation steps.

Here are the steps to build 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, and the process need a period of time to wait.

```
 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
------	------------------------------

For instance, to start with verbose output and the default config file :

```
 sudo webiopi -d -c /etc/webiopi/config
```

The Port is 8000 in default.

Until now, WebIOPi has been launched, and you can press "Ctrl+C" to terminate service.

Access WebIOPi over local network

Under the same network, use mobile phone or PC browser to open your RPi IP address, and add port number like 8000. For example, my raspberry pi IP address is 192.168.1.109. Then, in the browser, should input:

<http://192.168.1.109:8000/>

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

Then, enter the main control interface:

WebIOPi Main Menu

GPIO Header

Control and Debug the Raspberry Pi GPIO with a display which looks like the physical header.

GPIO List

Control and Debug the Raspberry Pi GPIO ordered in a single column.

Serial Monitor

Use the browser to play with Serial interfaces configured in WebIOPi.

Devices Monitor

Control and Debug devices and circuits wired to your Pi and configured in WebIOPi.

Click on GPIO Header to enter the GPIO control interface.

3.3V	1	2	5.0V	
I2C SDA	3	4	5.0V	
I2C SCL	5	6	GROUND	
ONEWIRE	7	8	UART TX	
GROUND	9	10	UART RX	
OUT	GPIO 17	11	12	GPIO 18 IN
IN	GPIO 27	13	14	GROUND
IN	GPIO 22	15	16	GPIO 23 IN
	3.3V	17	18	GPIO 24 IN
ALTO	GPIO 10	19	20	GROUND
ALTO	GPIO 9	21	22	GPIO 25 IN
ALTO	GPIO 11	23	24	GPIO 8 OUT
GROUND	25	26	GPIO 7	OUT
--	27	28	--	
IN	GPIO 5	29	30	GROUND
IN	GPIO 6	31	32	GPIO 12 IN
IN	GPIO 13	33	34	GROUND
IN	GPIO 19	35	36	GPIO 16 IN
IN	GPIO 26	37	38	GPIO 20 IN
GROUND	39	40	GPIO 21	IN

Control methods:

- Click/Tap the OUT/IN button to change GPIO direction.
- Click/Tap pins to change the GPIO output state.

Completed

According to the circuit we build, set GPIO17 to OUT, then click Header11 to control the LED.

About WebIOPi

(If you are using raspberry pie 4B, you may have some trouble. We will also finish the WebIOPi adaptation to raspberry 4B as soon as possible, and update it at the first time.)

The reason for changing file in the configuration process is that the model of new generation of RPi CPU is different from old one, which result in some of the issues during using.

WebIOPi has not provide corresponding installation package for latest RPi timely. Therefore, there are two changes in the configuration, and some BUG may exist to cause some problems to WebIOPi function. We look forward to that the author of WebIOPi to provide a complete set of the latest version of installation package to fit with RPi. WebIOPi can achieve far more than this, so we also look forward to learning and exploring with the funs.



What's next?

Thanks for your reading.

This tutorial is all over here. If you find any mistakes, omissions or you have other ideas and questions about contents of this tutorial or the kit and etc, please feel free to contact us: support@freenove.com
We will check and correct it as soon as possible.

If you are interesting in processing, you can learn the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and other interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

<http://www.freenove.com/>

Thank you again for choosing Freenove products.

Warning

When you purchase or use Freenove LCD Starter Kit for Raspberry Pi, please note the following:

- This product contains small parts. Swallowing or improper operation can cause serious infections and death. Seek immediate medical attention when the accident happened.
- Do not allow children under 3 years old to play with or near this product. Please place this product in where children under 3 years of age cannot reach.
- Do not allow children lack of ability of safe to use this product alone without parental care.
- Never use this product and its parts near any AC electrical outlet or other circuits to avoid the potential risk of electric shock.
- Never use this product near any liquid and fire.
- Keep conductive materials away from this product.
- Never store or use this product in any extreme environments such as extreme hot or cold, high humidity and etc.
- Remember to turn off circuits when not in use this product or when left.
- Do not touch any moving and rotating parts of this product while they are operating.
- Some parts of this product may become warm to touch when used in certain circuit designs. This is normal. Improper operation may cause excessively overheating.
- Using this product not in accordance with the specification may cause damage to the product.

About

Freenove is an open-source electronics platform. Freenove is committed to helping customer quickly realize the creative idea and product prototypes, making it easy to get started for enthusiasts of programing and electronics and launching innovative open source products. Our services include:

- Electronic components and modules
- Learning kits for Arduino
- Learning kits for Raspberry Pi
- Learning kits for Technology
- Product customization service
- Robot kits
- Auxiliary tools for creations

Our code and circuit are open source. You can obtain the details and the latest information through visiting the following web sites:

<http://www.freenove.com>

<https://github.com/freenove>

Your comments and suggestions are warmly welcomed, and please send them to the following email address:
support@freenove.com

If you have any business matters, please feel free to contact us:
sale@freenove.com