Free your innovation

Freenove is an open-source electronics platform.
www.freenove.com

Warning

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

- This product contains small parts. Swallowing or improper operation them 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
- 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

References

You can download the sketches and references used in this product in the following websites:
http://www.freenove.com
https://github.com/freenove
If you have any difficulties, you can send email to technical support for help.

The references for this product is named Freenove Basic Starter Kit for Raspberry Pi, which includes the following folders and files:

- Code Code for Raspberry Pi
- Readme.txt Instructions

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
On working day, we usually reply to you within 24 hours.

Copyright

Contents

Contents I
Preface 1
Raspberry Pi 1
GPIO Extension Board 4
Breadboard Power Module 5
C code & Python code 6
Chapter 0 Preparation 7
Step 0.1 Install the System 7
Step 0.2 Install WiringPi 14
Step 0.3 Obtain the Experiment Code 17
Step 0.4 Code Editor 18
Next 23
Chapter 1 LED 24
Project 1.1 Blink 24
Chapter 2 Button & LED 32
Project 2.1 Button & LED 32
Project 2.2 MINI table lamp 37
Chapter 3 LEDBar Graph 43
Project 3.1 Flowing Water Light 43
Chapter 4 Analog & PWM 48
Project 4.1 Breathing LED 48
Chapter 5 RGBLED 54
Project 5.1 Colorful LED 54
Chapter 6 Buzzer 60
Project 6.1 Doorbell 60
Project 6.2 Alertor 66
Chapter 7 WebIOPi & IOT 71
Project 7.1 Remote LED 71
What's next? 76

Preface

If you want to become a maker, you may have heard of Pi Raspberry or Arduino before. If not, it dosen't matter. Through referencing this tutorial, you can be relaxed in using raspberry Pi to create dozens of electronical interesting projects, and gradually realize the fun of using raspberry Pi to complete creative works.

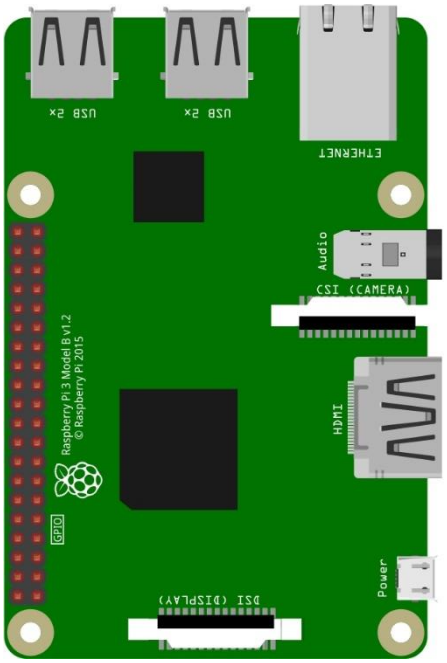
Raspberry Pi and Arduino have a lot of fans in the world. They are keen to exploration, innovation and DIY and they contributed a great number of high-quality open source code, circuit and rich knowledge base. So we can realize our own creativity more efficiently by using these free resource. Of course, you can also contribute your own strength to the resource. Raspberry Pi, different from Arduino, is more like a control center with a complete operating system, which can deasl with more tasks at the same time. Of course, you can also combine the advantages of them to make something creative.

Usually, a Raspberry Pi project consists of code and circuit. If you are familiar with computer language and very interested in the electronic module. Then this tutorial is very suitable for you. It will, from easy to difficult, explain the Raspberry Pi programming knowledge, the use of various types of electronic components and sensor modules and their operation principle. And we assign scene applications for most of the module. We provide code of both C and Python language versions for each project, so, whether you are a C language user or a Python language user, you are able to easily grasp the code in this tutorial. The supporting kit, Freenove Basic Starter Kit for Raspberry Pi, contains all the electronic components and modules needed to complete these projects. After completing all projects in this tutorial, you can also use these components and modules to achieve your own creativity, like smart home, smart car and robot. Additionally, if you have any difficulties or questions about this tutorial and the kit, you can always ask us for quick and free technical support.

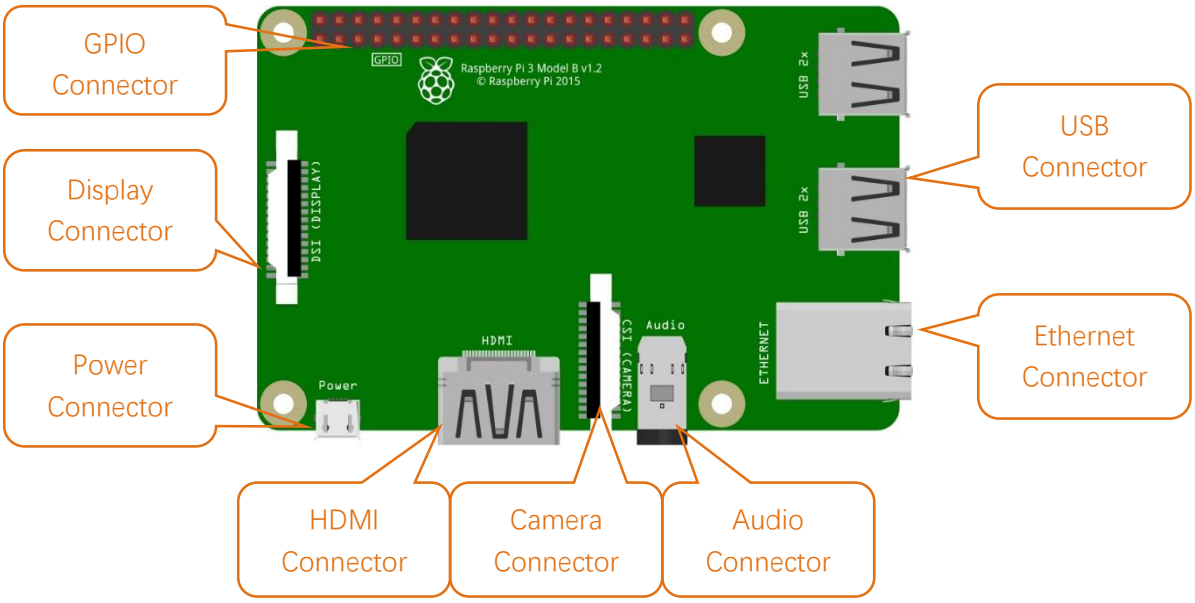
Raspberry Pi

Raspberry Pi (called RPi, RPI, RasPi, the text these words will be used alternately behind), a micro computer with size of a card, quickly swept the world since its debut. It is widely used in desktop workstation, media center, smart home, robots, and even the servers, etc. It can do almost anything, which continues to attract fans to explore it. Raspberry Pi used to be running in Linux system and along with the release of windows 10 IoT, we can also run it in Windows. Raspberry Pi (with interfaces for USB, network, HDMI, camera, audio, display and GPIO), as a microcomputer, can be running in command line mode and desktop system mode. Additionly, it is easy to operate just like Arduino, and you can even directly operate the GPIO of CPU. So far, Pi Raspberry has 7 versions: A type, A+ type, B type, B+ type, second-generation B type, third-generation B type and Zero version, respectively. 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. Orther 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. The projects in this tutorial, with no special note, are using Raspberry Pi 3 Model B (RPi3B), which is compatible with Raspberry Pi A+, B+, 2B and Zero.

Schematic diagram of RPi3B is shown below:

Practicality picture of Raspberry Pi 3 Model B:	Model diagram of Raspberry Pi 3 Model B:
	

Hardware interface diagram of RPi3B is shown below:



GPIO

General purpose input/output; in this specific case the pins on the Raspberry Pi and what you can do with them. So called because you can use them for all sorts of purposes; most can be used as either inputs or outputs, depending on your program.

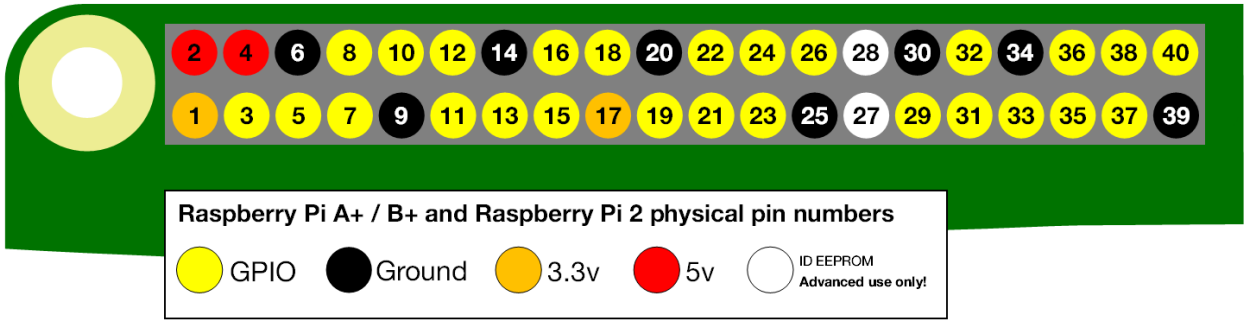
When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering.

GPIO NUMBERING

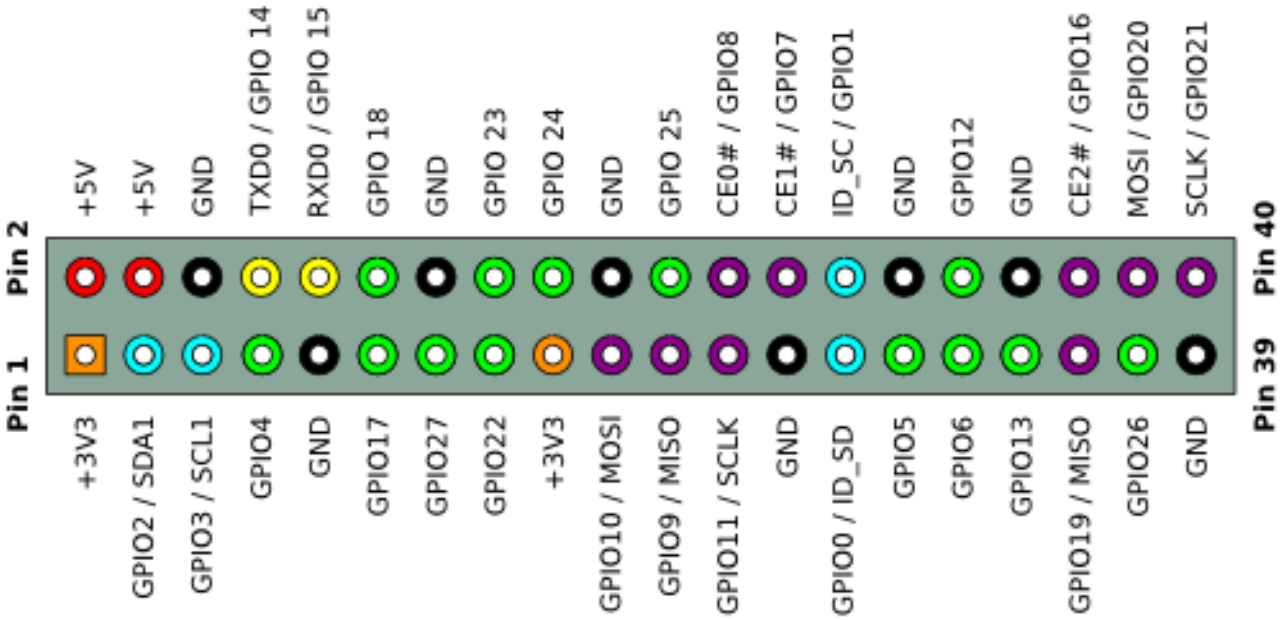
These are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. You will need a printed reference or a reference board that fits over the pins.

PHYSICAL NUMBERING

The other 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' and it looks like this:



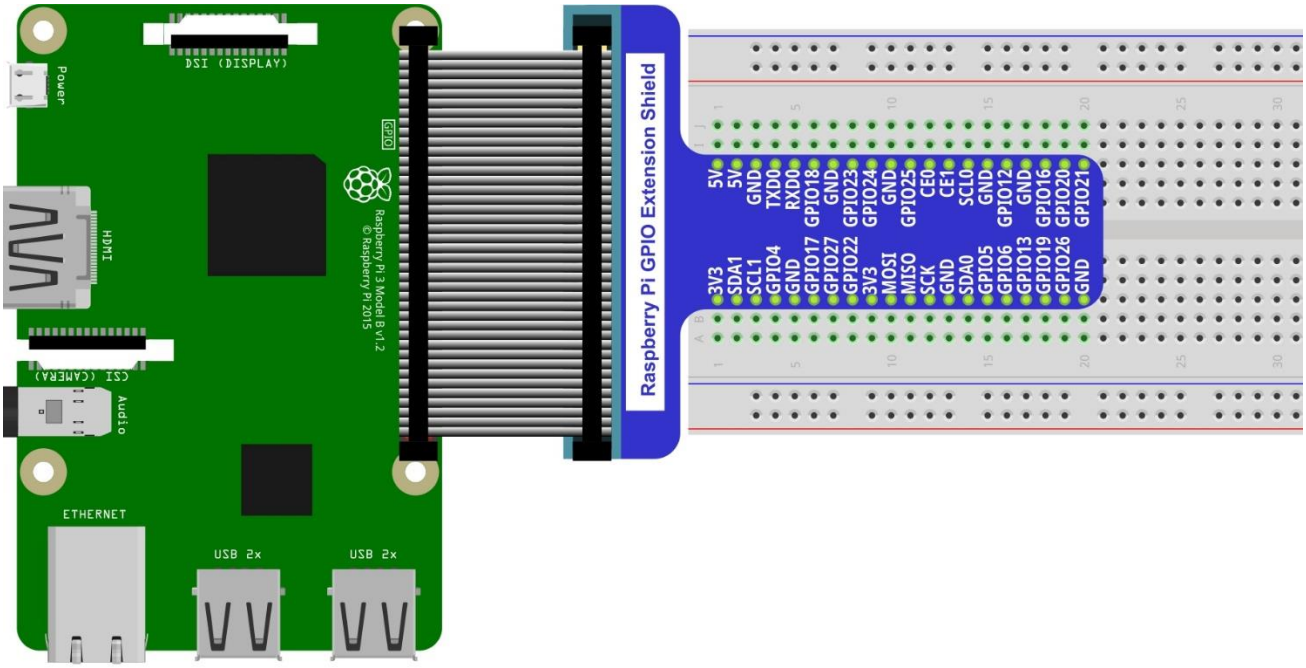
Each pin is defined as below:



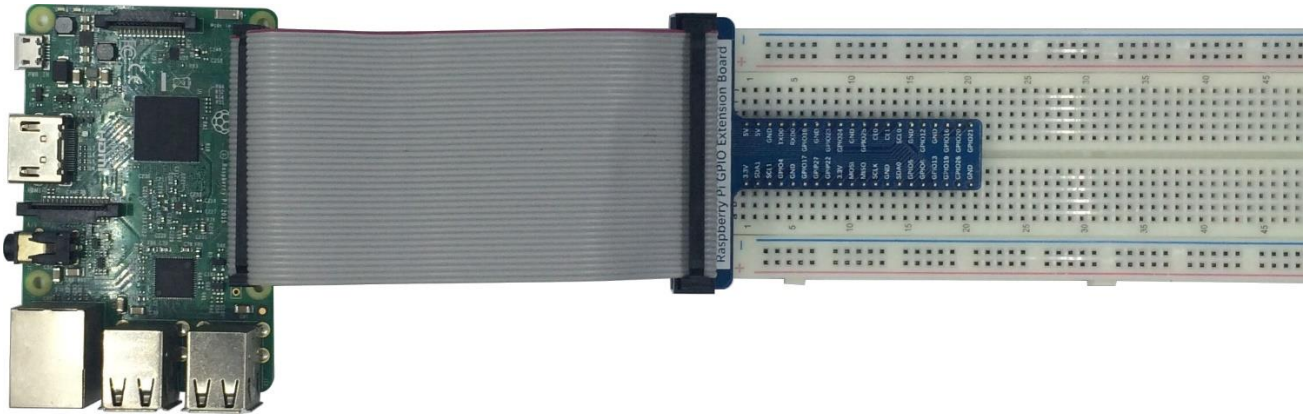
For more details about pin definition of GPIO, please refer to http://pinout.xyz/

GPIO Extension Board

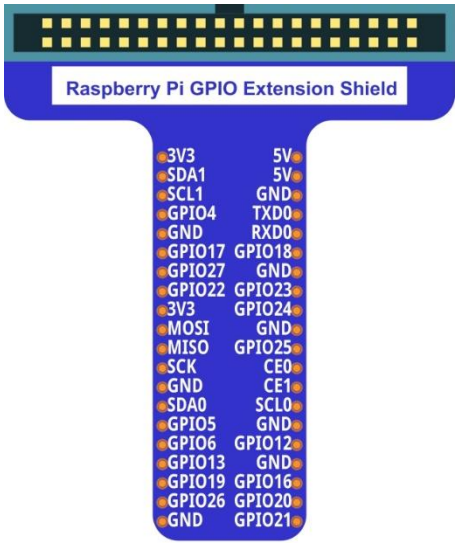
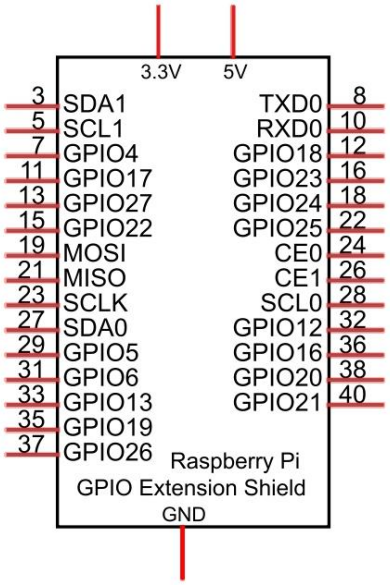
When we use RPi to do the experiment, we had better use GPIO, which is more convenient to extend all IO ports of RPi to the bread board directly. The GPIO sequence on Extension Board is identical to the GPIO sequence of RPi. Since the GPIO of different versions of RPi is different, the corresponding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



Practicality picture of connection:

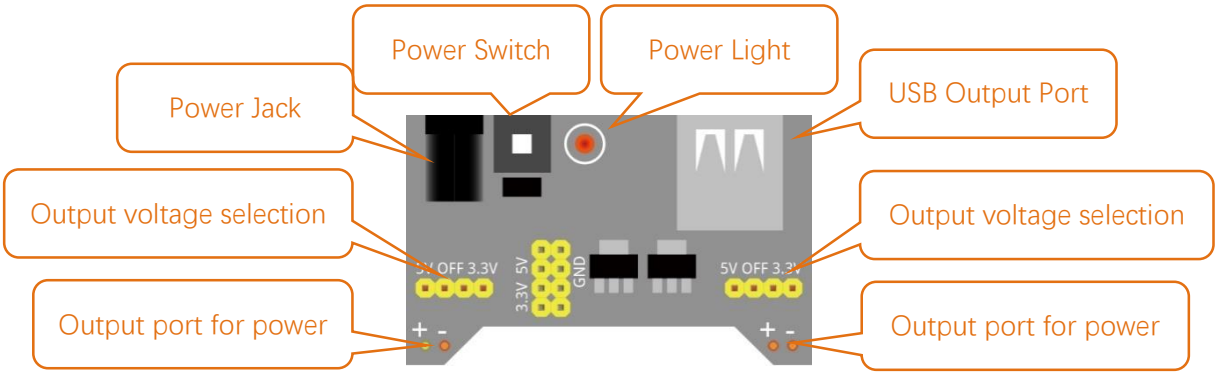


Among them, GPIO Extension Board and its schematic are shown below:

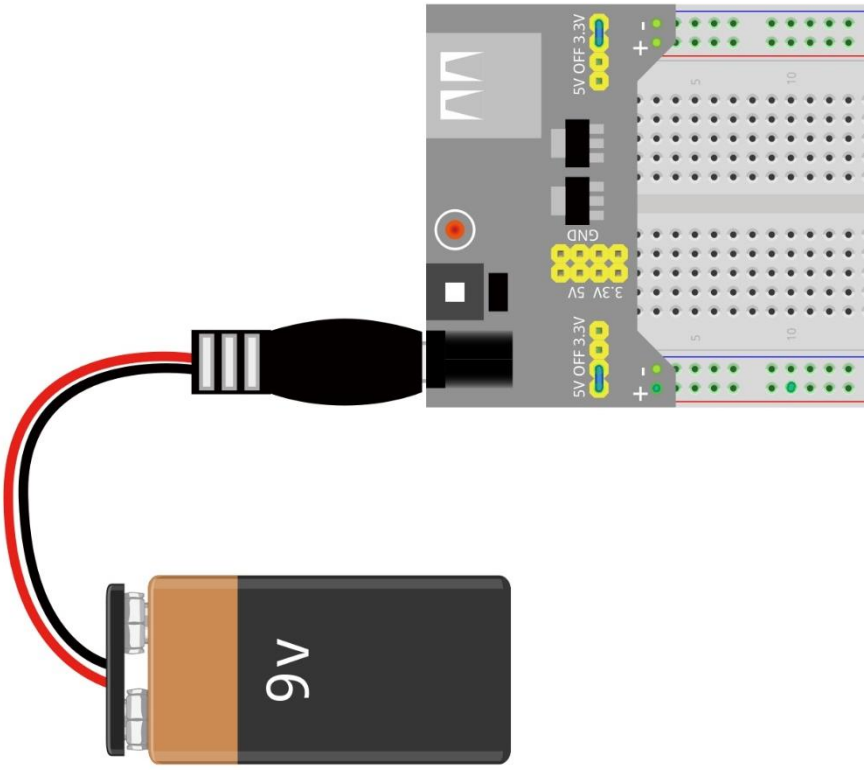
GPIO Extension Board	Definition of pins
	

Breadboard Power Module

Breadboard Power Module is an independent board, can provide independent 5V or 3.3V power for bread board when used to build the circuit, which 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:



C code & Python code

Experiments involving programming in this tutorial use both C and python languages. And every kind of code will be explained in details, and be followed by detailed comments to ensure that you can quickly master it. In addition, you can also directly contact us to obtain other help.

These codes is available in http://github.com/freenove.

Chapter 0 Preparation

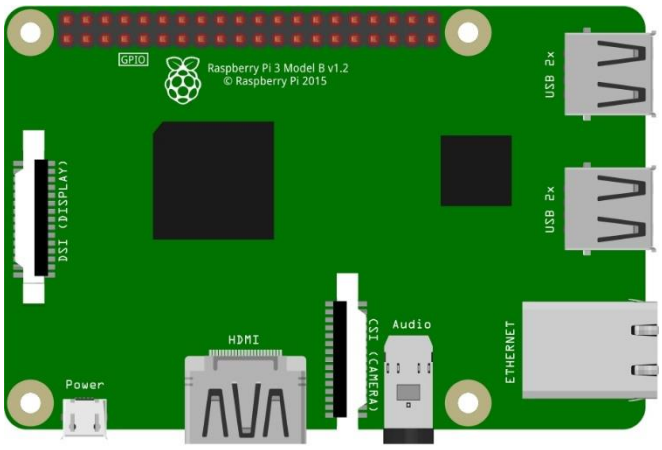
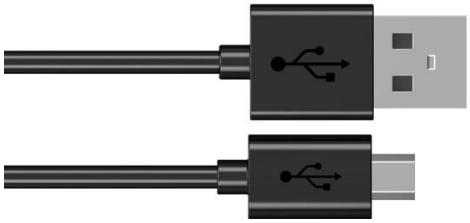
Why is “Chapter 0”? Because in the program code, all the counts are starting from 0. We choose to follow this rule (just a joke). In this chapter, we will do some necessary preparation work: start your Pi Raspberry and install some necessary libraries. If your Raspberry Pie can be started normally and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

Step 0.1 Install the System

Firstly, install a system for your RPi.

Component List

Required Components

Raspberry Pi 3B x1 	5V/2A Power Adapter 
Micro USB Cable x1 	Micro SD Card （TF Card）x1； Card Reader x1 

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/2A, because lack of power supply will lead to many abnormal problems, even damage to your RPI. So use of power supply with 5V/2A is highly recommend. SD Card Micro (recommended capacity 8GB or more) is a hard drive for RPi, which is used to store the system and personal files. In latter experiments, the components list with a RPi will contain these required components, using only RPi as a representative rather than presenting details.

Optional Components

1.Display with HDMI interface

2.Mouse and Keyboard with USB interface

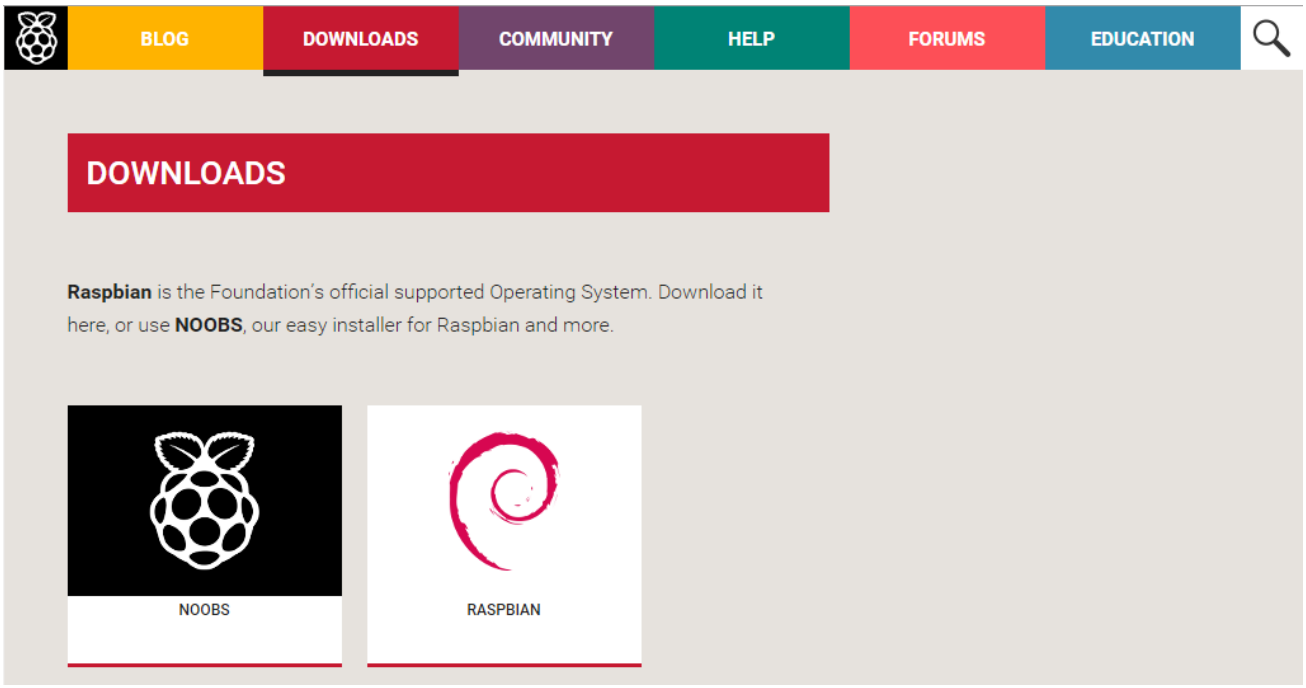
Among these optional components, the Display as a screen for RPi. If no, it does not matter, you can use a remote desktop of your personal PC to control your RPi. Mouse and keyboard are the same, if no, use remote desktop and share a set of keyboard and mouse personal with PC.

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

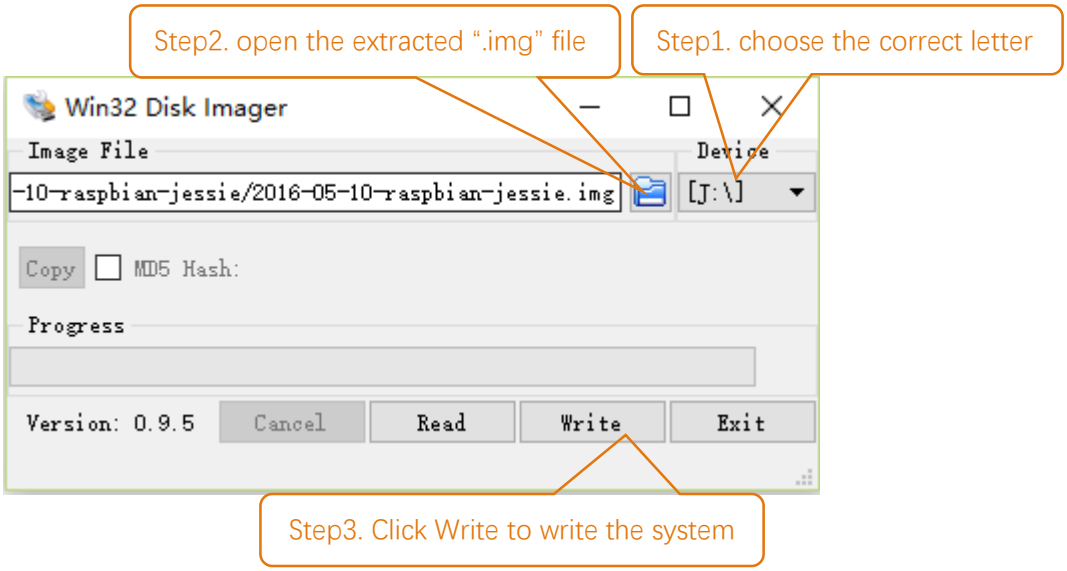
Visit RPi official website (https://www.Raspberry_Pi.org/), click "Downloads" and choose to download "RASPBIAN". RASPBIAN supported by RPI is an operating system based on Linux, which contains a number of contents required for RPi. We recommended RASPBIAN system to beginners. All experiments in this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



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

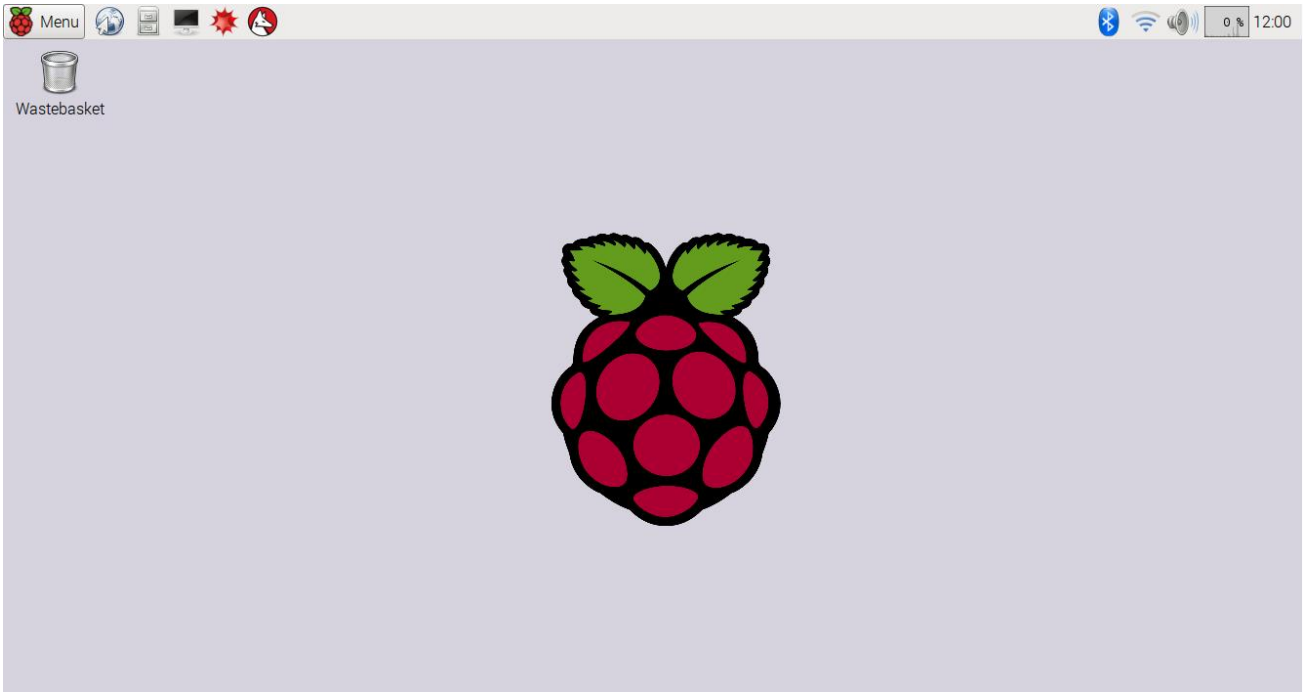
Write System to Micro SD Card

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 “J”), open the extracted “.img” file and then click the "Write".



Start Raspberry Pi

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. Latter, you need to enter the user name and password to login. The default user name: Pi; password: raspberry. Enter and login. After logining, you can enter the following interface.



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

Remote desktop

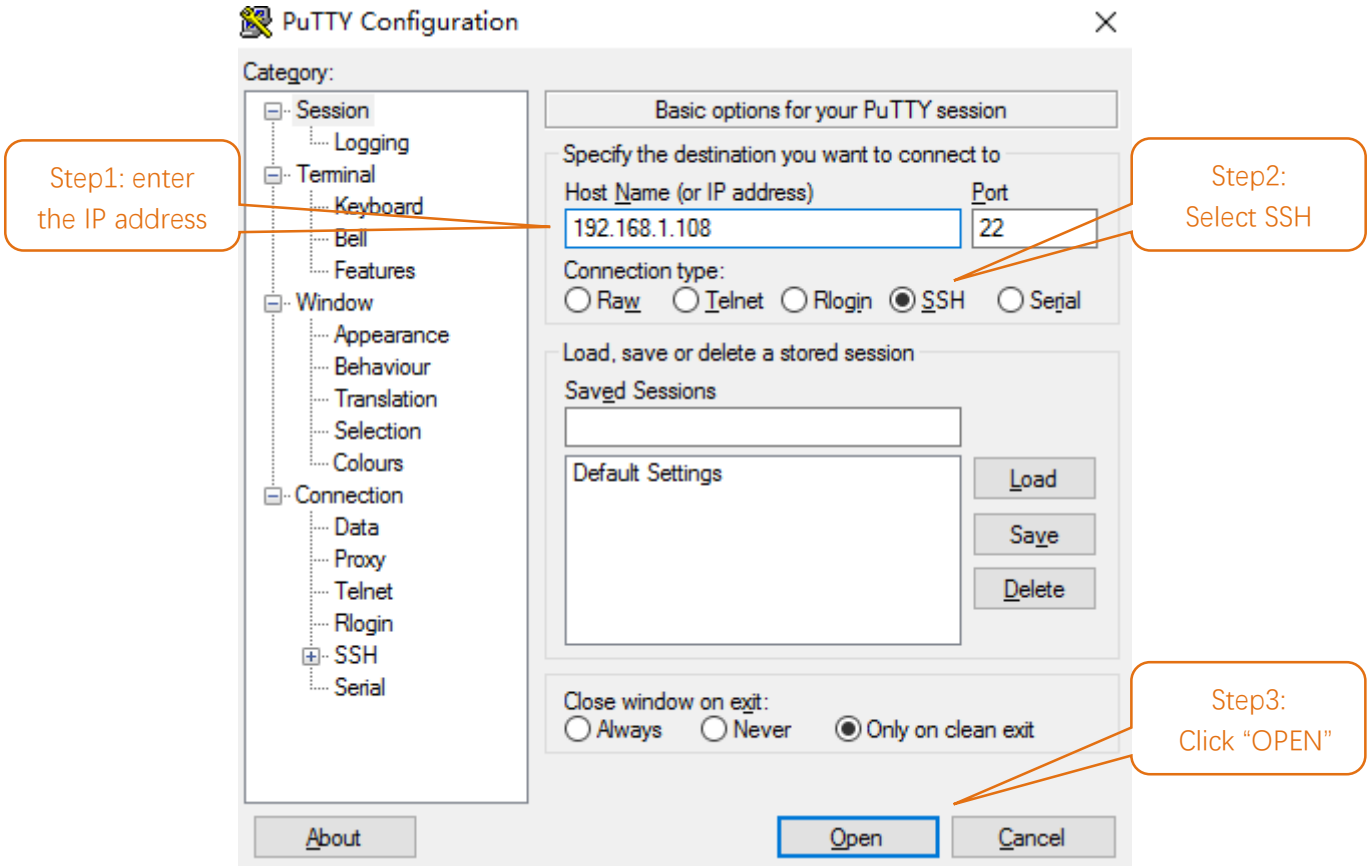
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use remote desktop to control RPi under the Windows operating system.

Install Xrdp Services

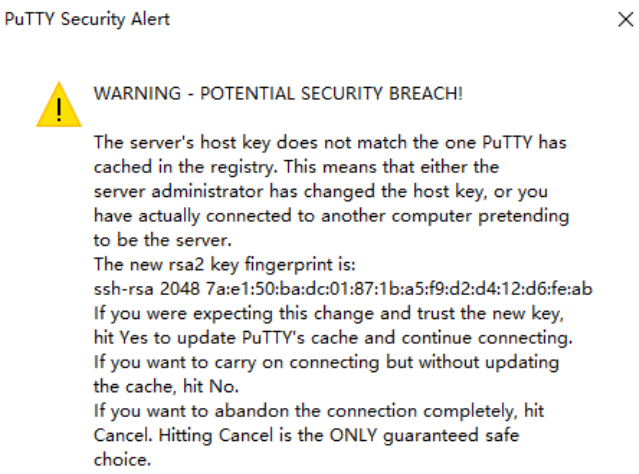
First, download the tool software Putty. Its official address: http://www.putty.org/

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 TF card prepared before into the slot of the RPi and turn on the power supply waiting for the start RPi. Later, enter control terminal of the router to inquiry IP address named “raspberrypi”. For example, I have inqueried to my RPi IP address is “192.168.1.108". Then open Putty, enter the address, select SSH, and then click "OPEN", as shown below:

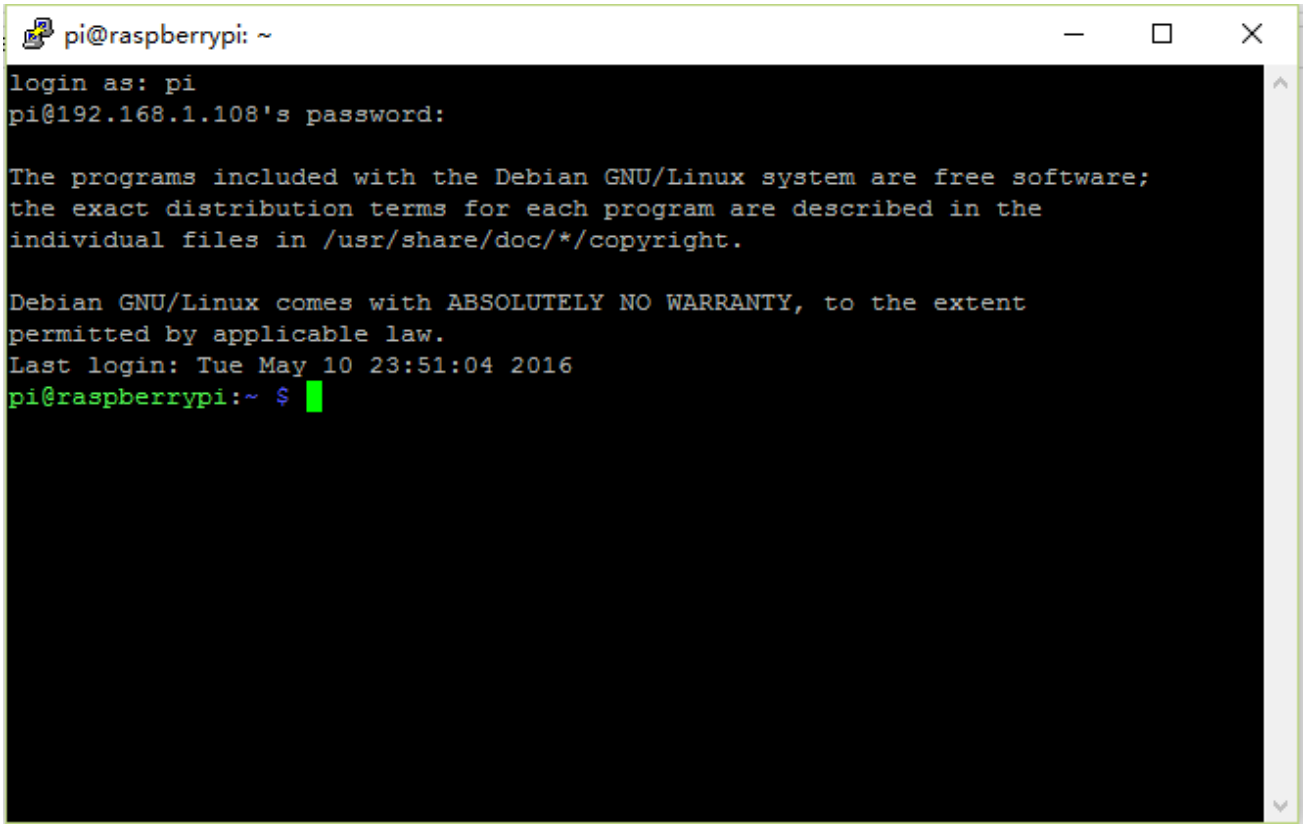


There will appear a security warning at first logining. Just click “YES”.



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, but this does not mean that you didn’t enter. After the correct output, press “Enter” to confirm.

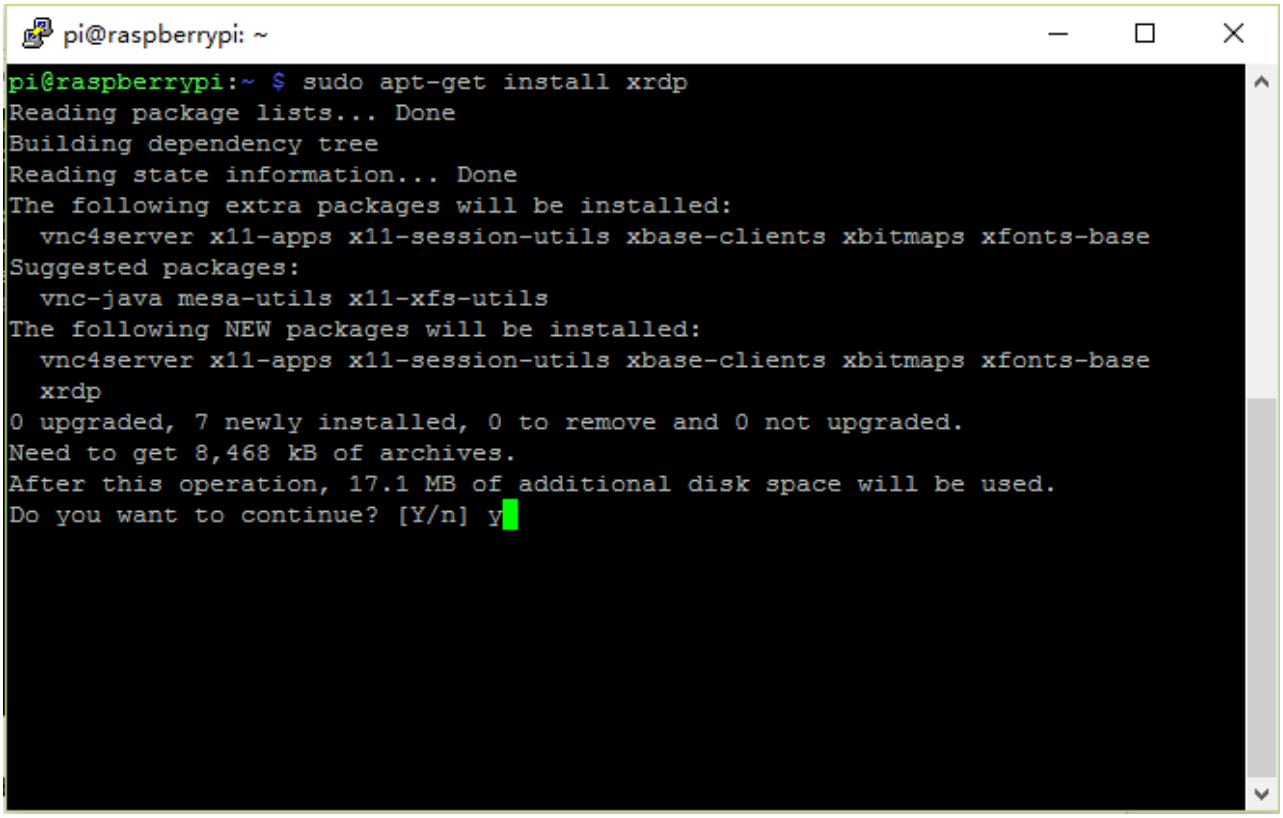
Then enter the command line of RPi, which means that you have successfully login to RPi command line mode.



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

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

Latter, the installation starts.

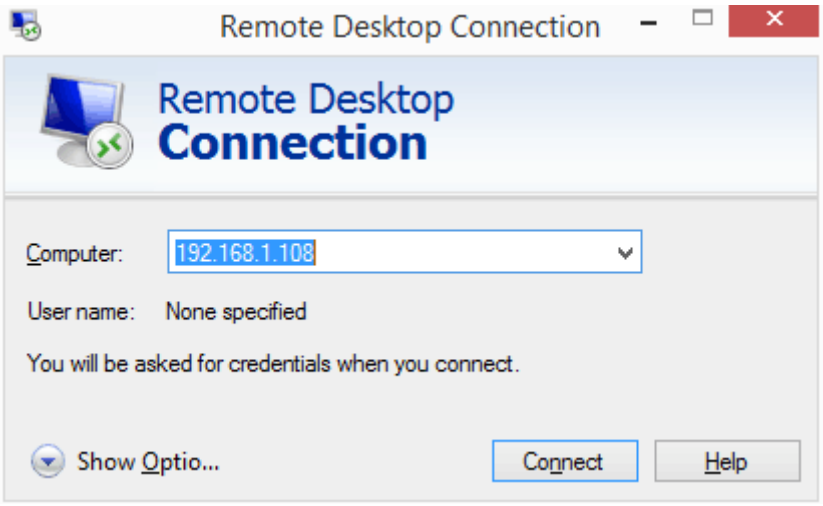


Enter "Y", press key “Enter” to confirm.

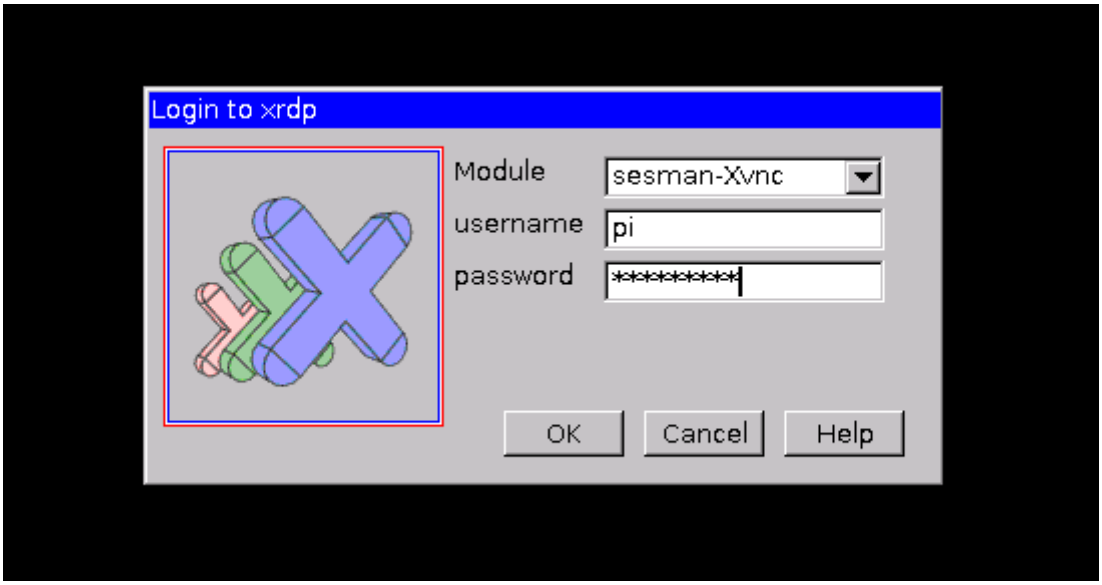
After the installation is completed, you can use Windows remote desktop applications to login to your RPi.

Login to Windows remote desktop

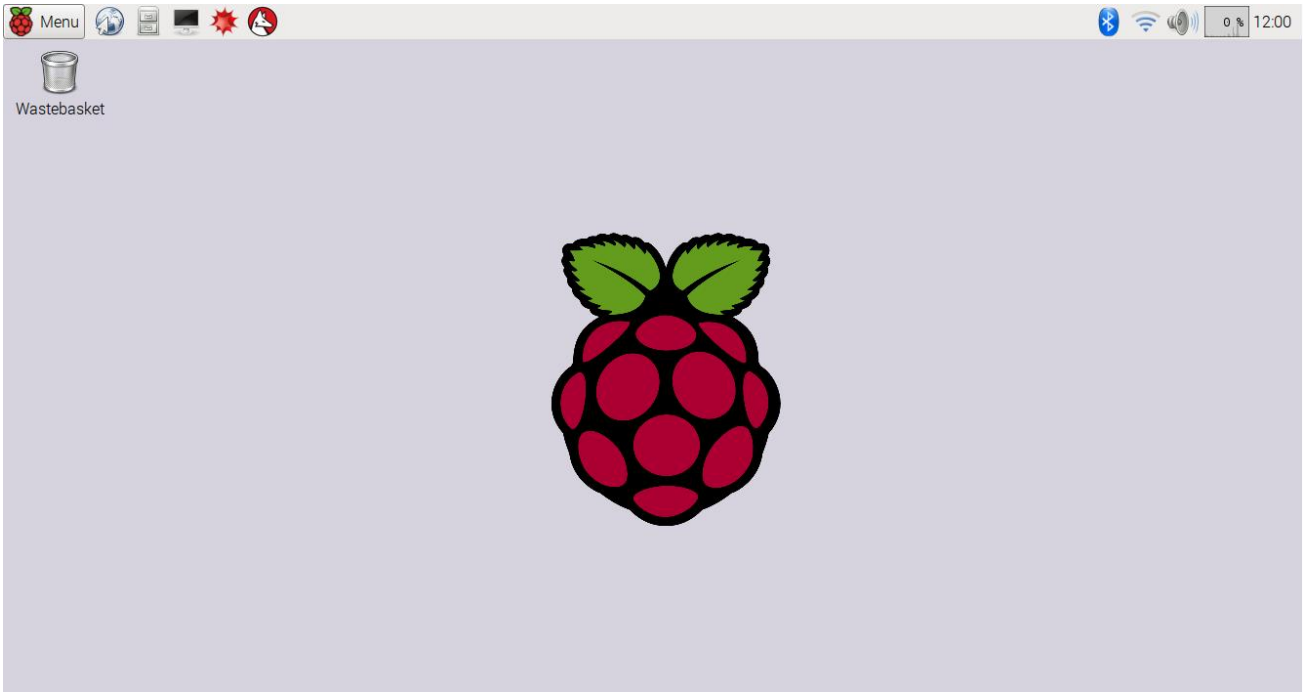
Use "WIN+R" or search function, open the remote desktop application "mstsc.exe" under Windows, enter the IP address of RPi and then click “Connect”.



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.



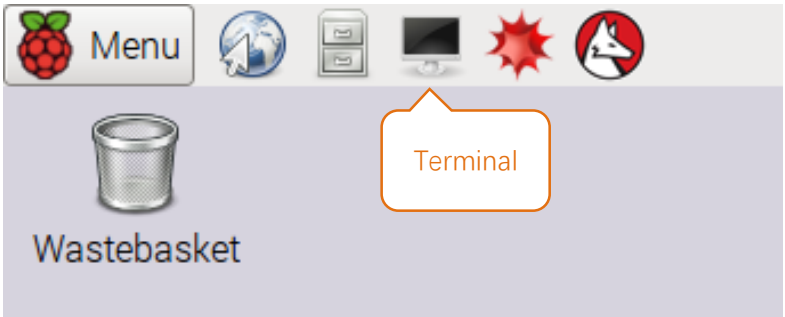
Here, you have successfully used the remote desktop login to RPi.
Then continue to do some preparation work: install a GPIO library wiringPi for your RPi.

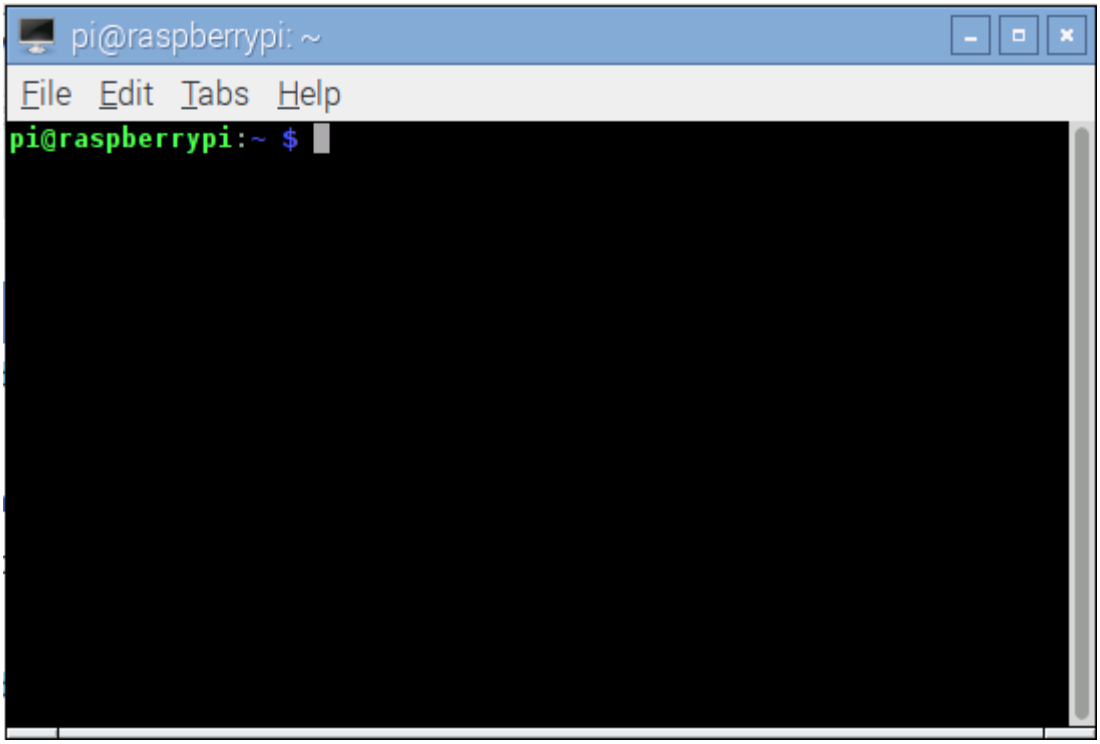
Step 0.2 Install WiringPi

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 and 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

open the terminal:





Follow these steps and commands to complete the installation.

Enter the following command in the terminal to obtain WiringPi using GIT:

```
git clone git://git.drogon.net/wiringPi
```

After the cloning operation is completed, go to the wiring folder and update the latest WiringPi.

```
cd wiringPi
git pull origin
```

Run the build file to start the installation.

```
./build
```

The new build script will compile and install it all for you – it does use the sudo command at one point, so you may wish to inspect the script before running it.

run the gpio command to check the installation:

```
gpio -v
gpio readall
```

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

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

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

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

For Pi B

For Pi B+, 2B, 3B, Zero

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

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

```
gpio readall
```

```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 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 | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

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

Step 0.3 Obtain the Experiment 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 experiment code. We provide both C language and Python code for each experiment 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:

```
git clone https://github.com/freenove/Freenove_Basic_Starter_Kit_for_Raspberry_Pi
```

Put the file into user directory pi/ and the code file into Freenove_Basic_Starter_Kit_for_Raspberry_Pi/Code. There are two folders "C code" and "Python code“ used to store C code and Python code of each experiment separately.

Step 0.4 Code Editor

vi, nano, Geany

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. We will use the three editors to open an example code "Hello.c" respectively. First to demonstrate the use of the vi and nano editor:

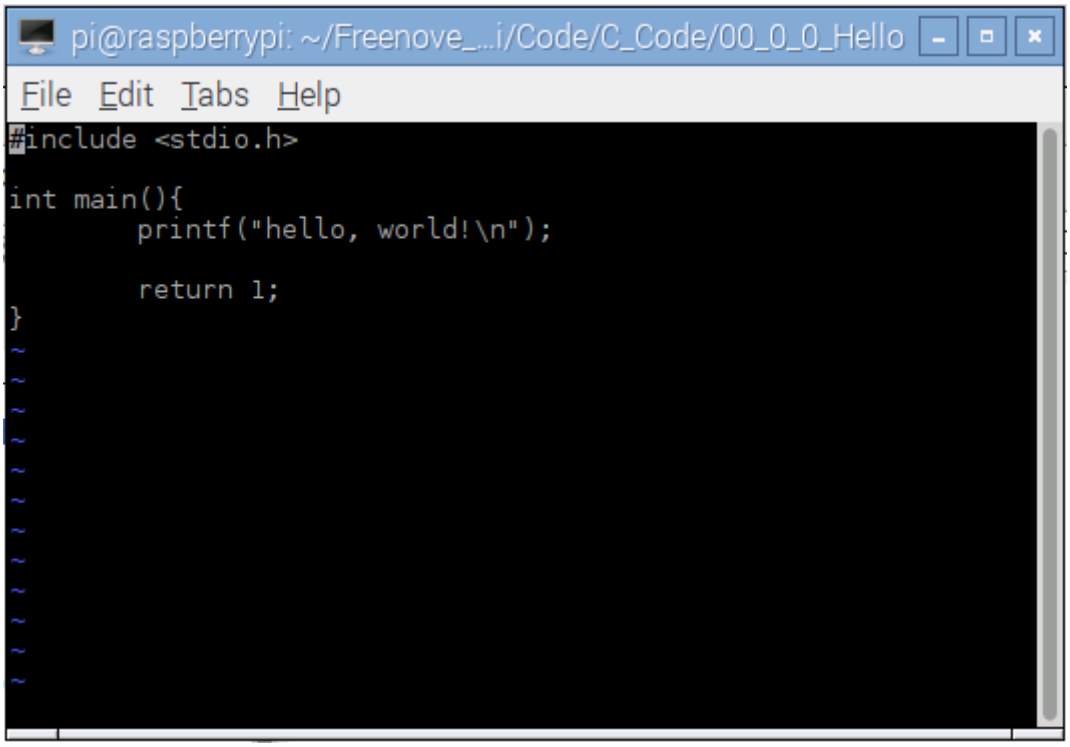
First, use the cd command to enter the sample code folder.

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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
```

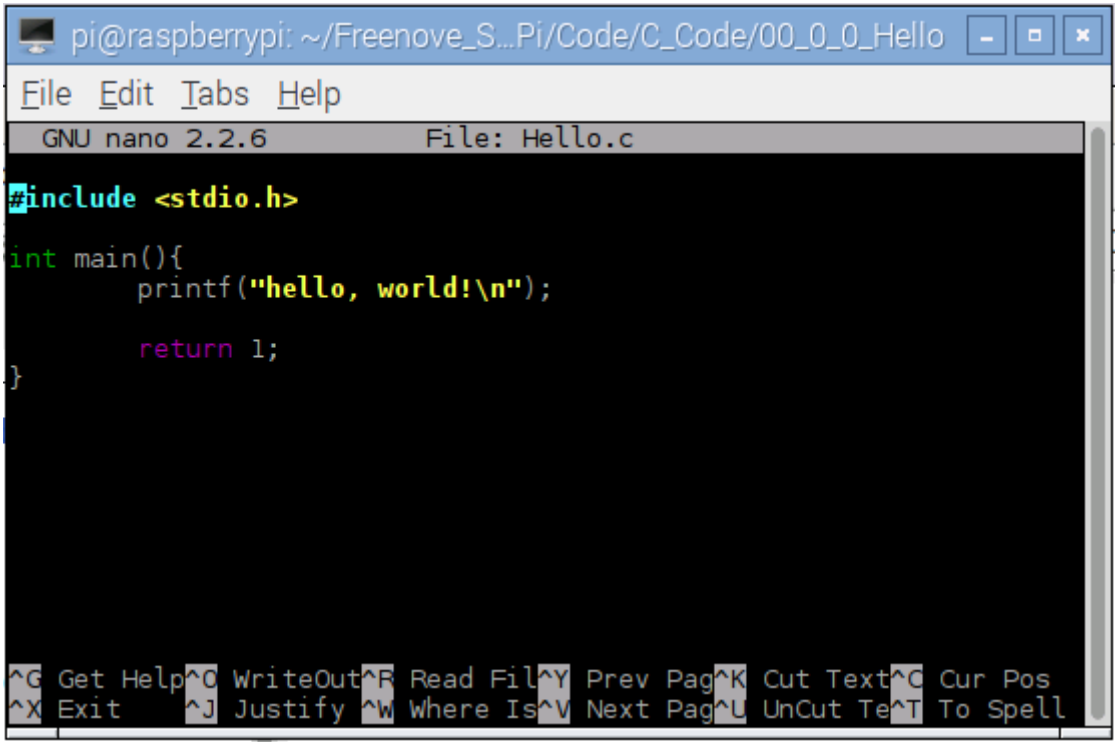
As is shown below:



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

```
nano Hello.c
```

As is hown below：



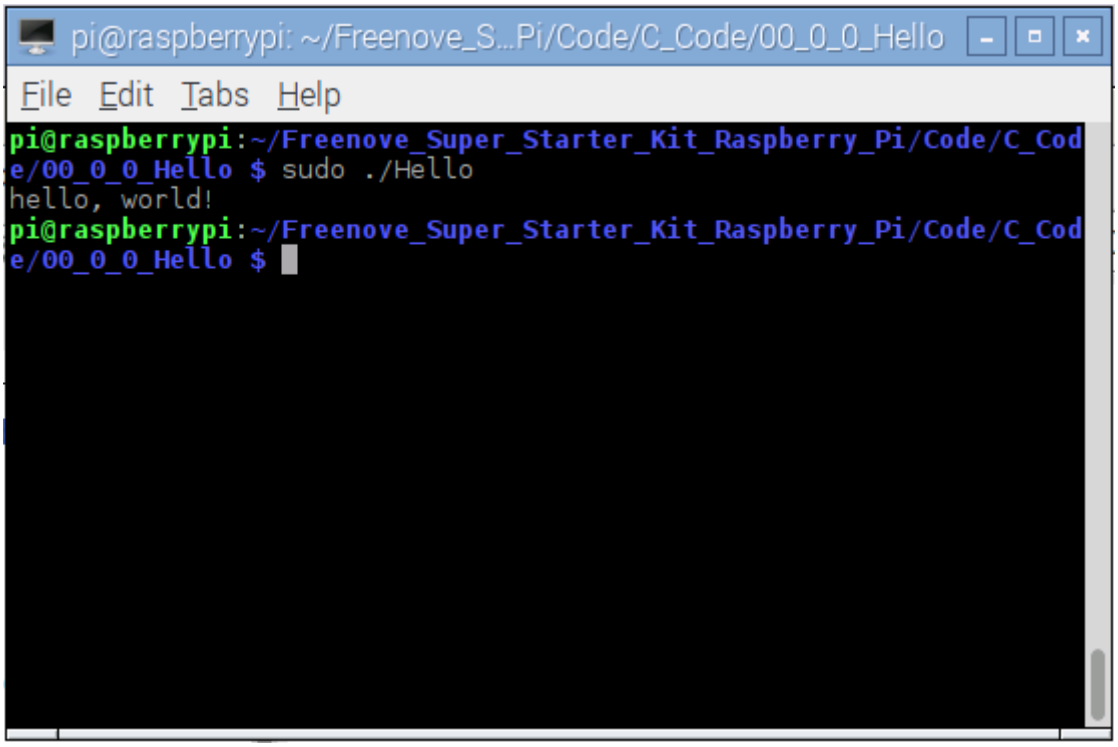
Use the following command to compile the code to generate the executable file “Hello”.

```
gcc Hello.c –o Hello
```

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

```
sudo ./Hello
```

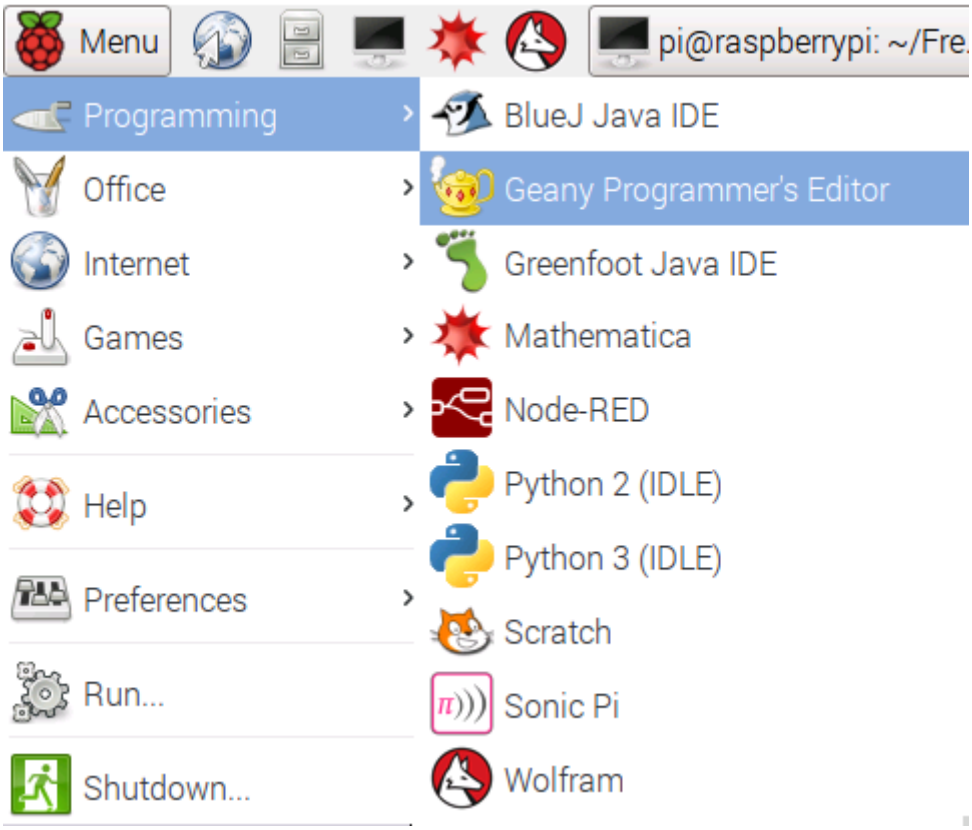
After the execution, "Hello, World!" is printed out in terminal.

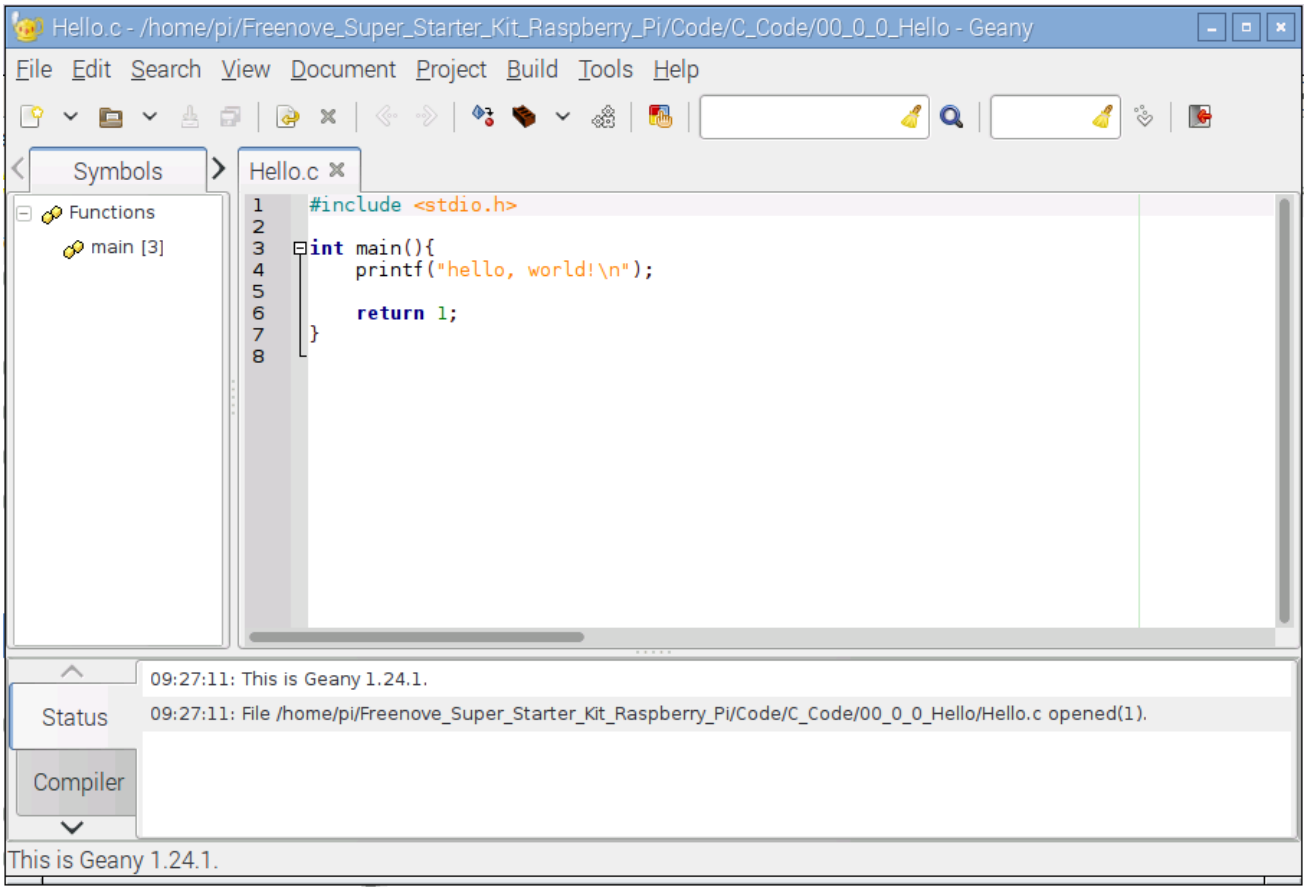


Next, learn to use the Geany editor. Use the following command to open the Geany in the sample file "Hello.c" file directory path.

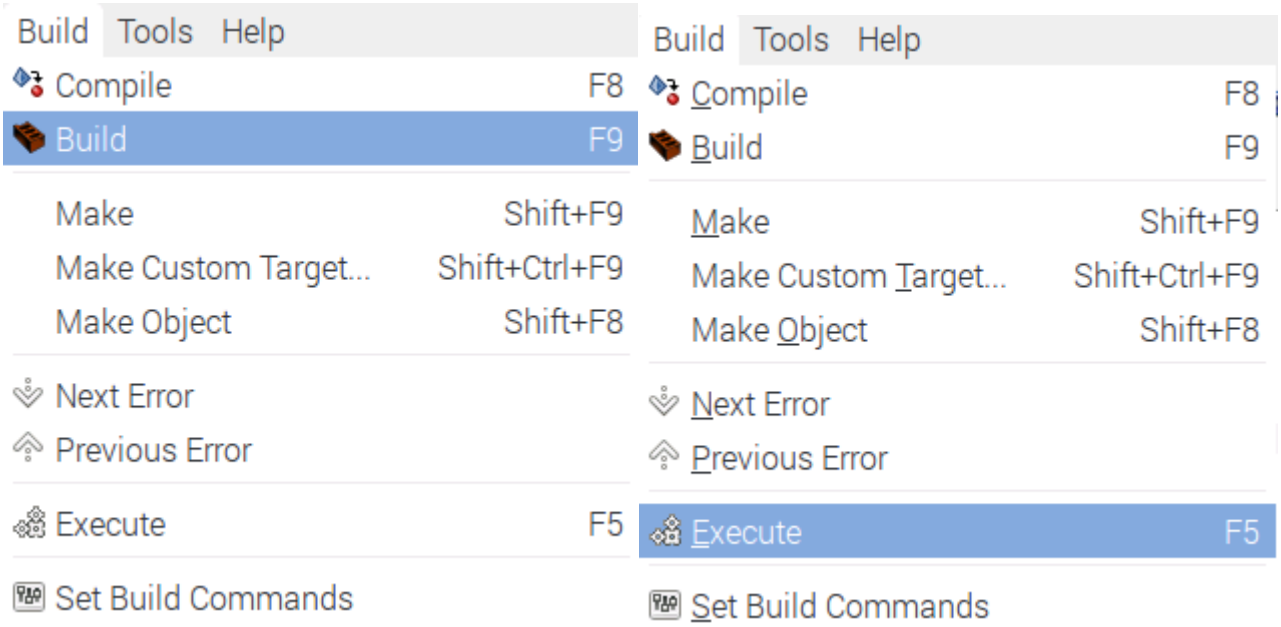
```
geany Hello.c
```

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.

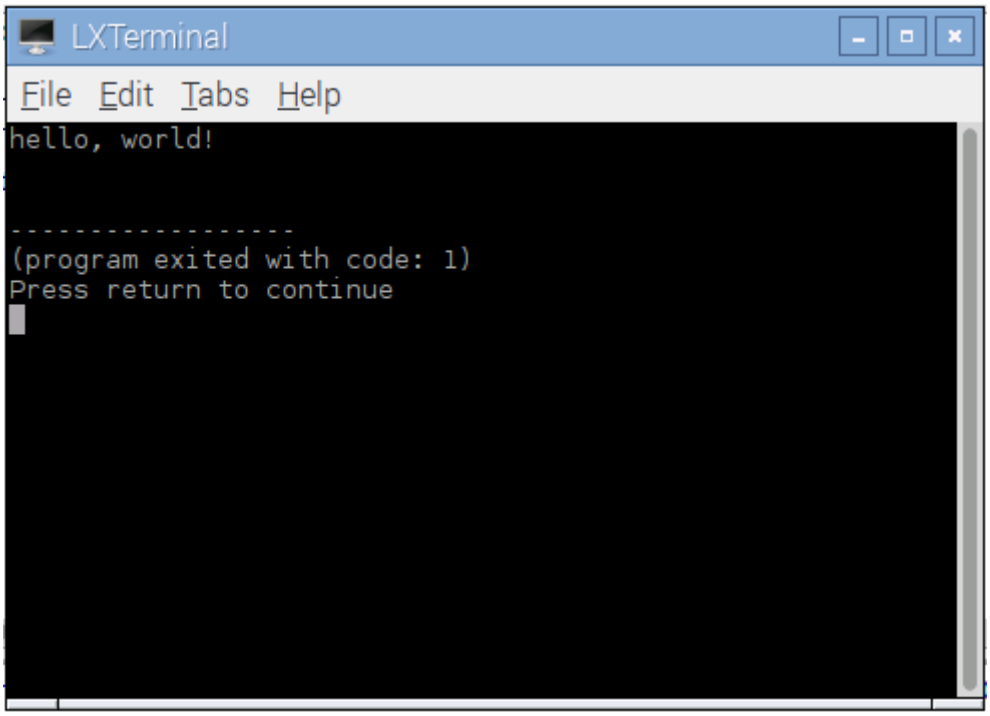




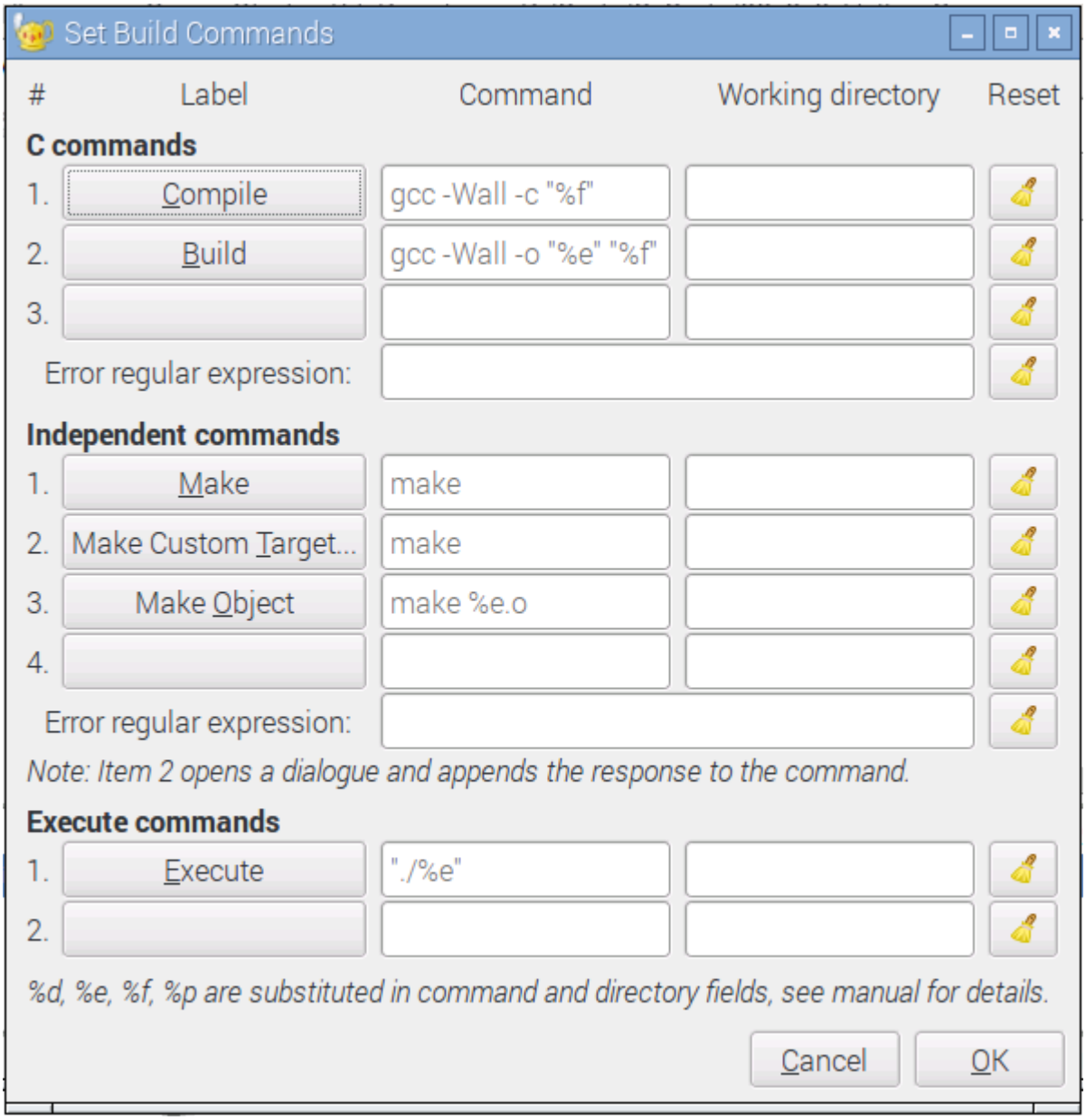
Generates an executable file by clicking menu bar Build->Build, then execute the generated file by clicking menu bar Build->Excute.



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



You can click Build->Set Build Commands to set compiler commands. In later experiments, 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 tlatter experiments, 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.

Next

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

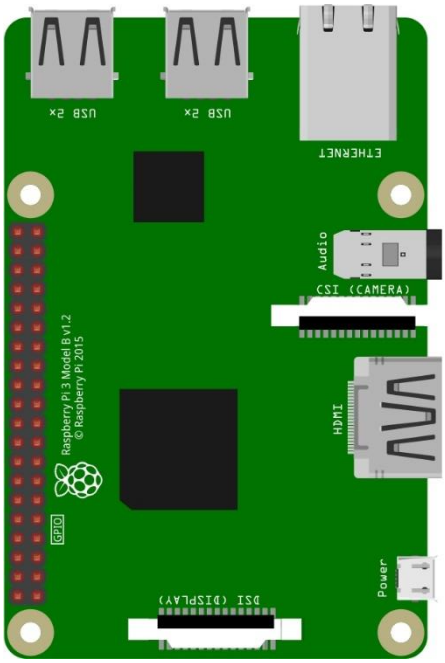
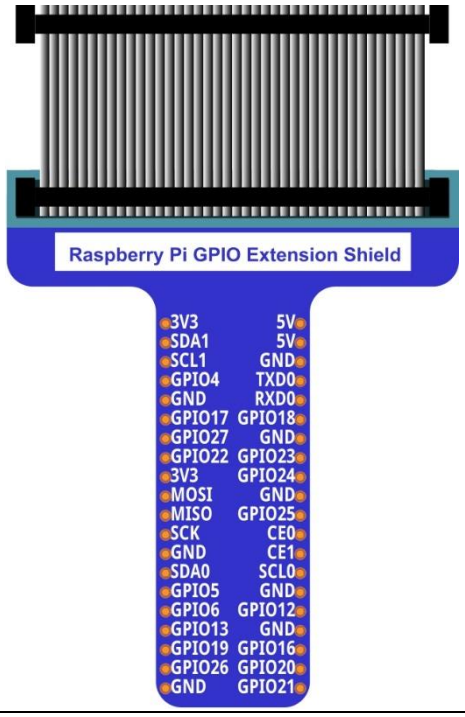
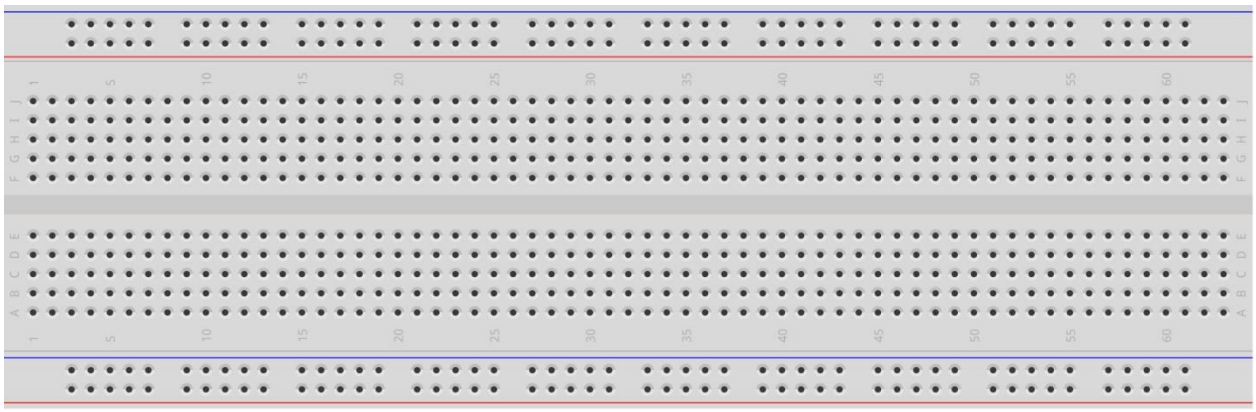
Chapter 1 LED

This chapter is the starting point of the journey to explore RPi electronic experiments. 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 3B x1 	GPIO Extension Board & Wire x1 
BreadBoard x1 	

LED x1	Resistor 220Ω x1	Jumper Wire M/M x2
		

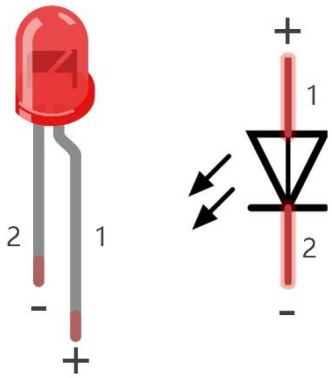
In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each experiment. They will be listed only in text form.

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.



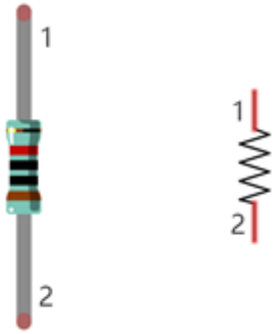
The LED can not be directly connected to power supply, which can damage component. A resistor with certain resisitance must be connected in series in the circuit of LED.

Resistor

The unit of resistance(R) is ohm(Ω). 1mΩ=1000kΩ, 1kΩ=1000Ω.

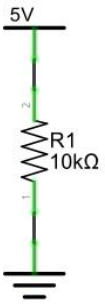
Resistor is an electrical component that limits or regulates the flow of current in an electronic circuit.

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/10kΩ=0.0005A=0.5mA.

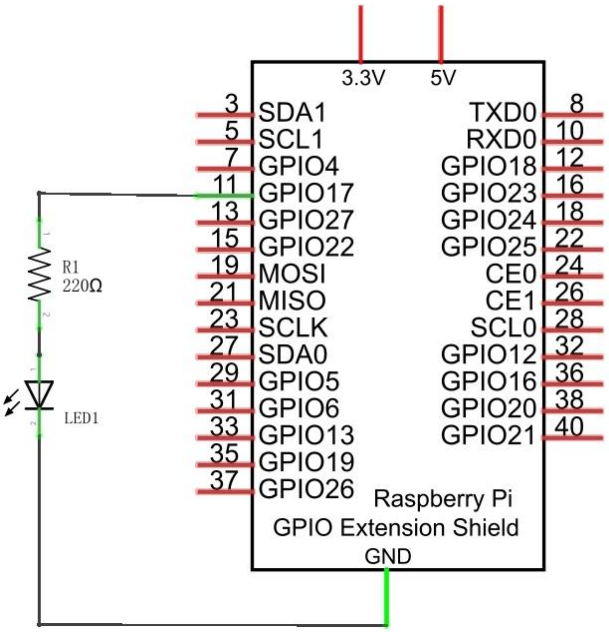


Do not connect the two poles of power supply with low resistance, which will make the current too high to damage electronic components.

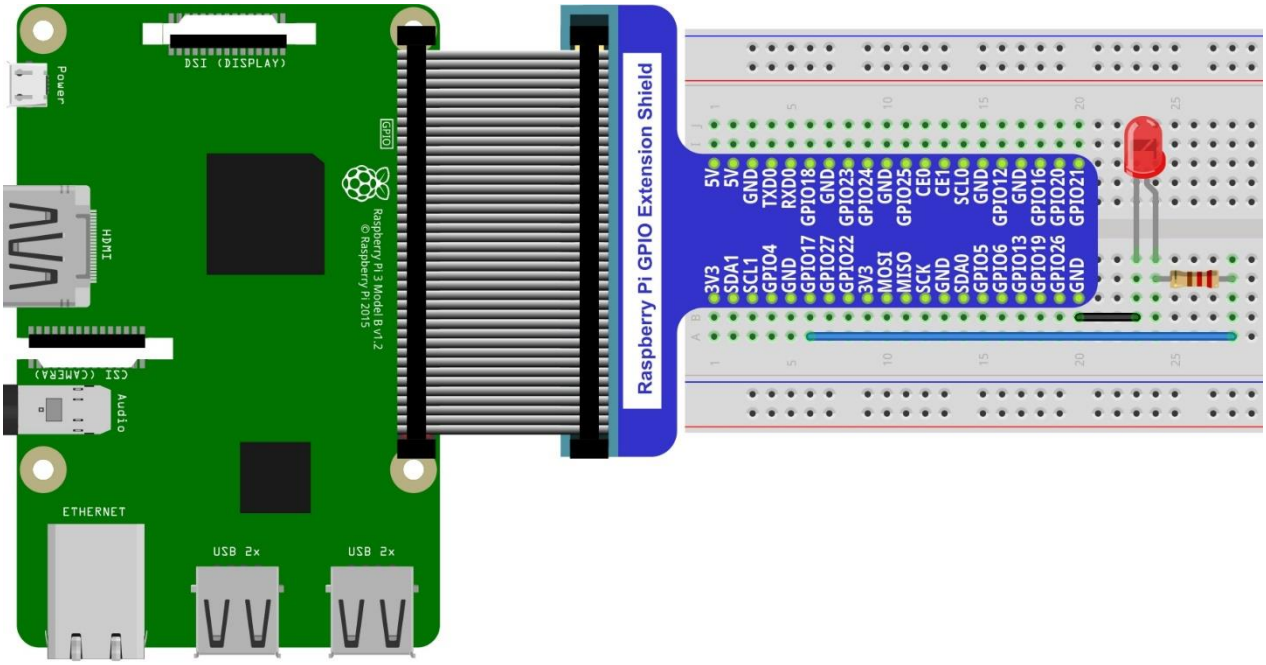
Circuit

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 anormal circuit work, or even damage to PRi.

Schematic diagram



Hardware connection



Because Numbering of GPIO Extension Shield is the same as RPi GPIO, latter 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 cyclely to make LED blink. We will use both C code and Python code to achieve the target.

C Code 1.1.1 Blink

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

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/Code/C_Code/01.1.1_Blink
```

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

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

3. Then run the generated file “blink”.

```
sudo ./Blink
```

Now, LED start blink. You can press “Ctrl+C” to end the program.

The following is the program code:

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

#define  ledPin    0

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

    pinMode(ledPin, OUTPUT);

    while(1){
            digitalWrite(ledPin, HIGH);  //led on
            printf("led on...\n");
            delay(1000);
            digitalWrite(ledPin, LOW);  //led off
            printf("...led off\n");
            delay(1000);
    }

    return 0;
}
```

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

```
#define  ledPin    0
```

In the main function main(), initialize wiringPi first, and then print out the initial results. Once the initialization fails, exit the program.

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

After the wiringPi is initialized successfully, set the ledPin to output mode. And then enter the while cycle, which is a 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);
    while(1){
            digitalWrite(ledPin, HIGH);  //led is turned on
            printf("led on...\n");
            delay(1000);
            digitalWrite(ledPin, LOW);  //led is turned off
            printf("...led off\n");
            delay(1000);
    }
```

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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/01.1.1_Blink
```

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

```
python Blink.py
```

Now, LED start blinking.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  ledPin = 11    # RPi Board pin11
5
6  def setup():
7      GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs based on physical location
8      GPIO.setup(ledPin, GPIO.OUT)   # Set ledPin to output mode
9      GPIO.output(ledPin, GPIO.LOW) # Set ledPin low to turn off led
10     print 'using pin%d'%ledPin
11
12 def loop():
13     while True:
14         GPIO.output(ledPin, GPIO.HIGH)  # led is turned on
15         print '...led on'
16         time.sleep(1)
17         GPIO.output(ledPin, GPIO.LOW) # led is turned off
18         print 'led off...'
19         time.sleep(1)
20
21 def destroy():
22     GPIO.output(ledPin, GPIO.LOW)     # led is turned off
23     GPIO.cleanup()                     # Release resource
24
25 if __name__ == '__main__':     # Program start from here
26     setup()
27     try:
28         loop()
29     except KeyboardInterrupt:  # When "Ctrl+C" is pressed, the subprogram destroy()
30 will be executed.
31         destroy()
```

In the subfunction setup (), GPIO.setmode (GPIO.BOARD) is used to set the serial number for GPIO based on physical location of the pin. The GPIO17 use the pin11 of the board, so define the ledPin as 11 and set the ledPin to output mode (output low level).

```
ledPin = 11    # RPi Board pin11

def setup():
    GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT)   # Set ledPin to output mode
    GPIO.output(ledPin, GPIO.LOW) # Set ledPin to low level to turn off led
    print 'using pin%d'%ledPin
```

In the subfunction of 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)  # led is turned on
        print '...led on'
        time.sleep(1)
        GPIO.output(ledPin, GPIO.LOW) # led is turned off
        print 'led off...'
        time.sleep(1)
```

Finally, when the program is terminated, the 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 destroy () 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 destroy():
    GPIO.output(ledPin, GPIO.LOW)     # led is turned off
    GPIO.cleanup()                    # Release resource
```

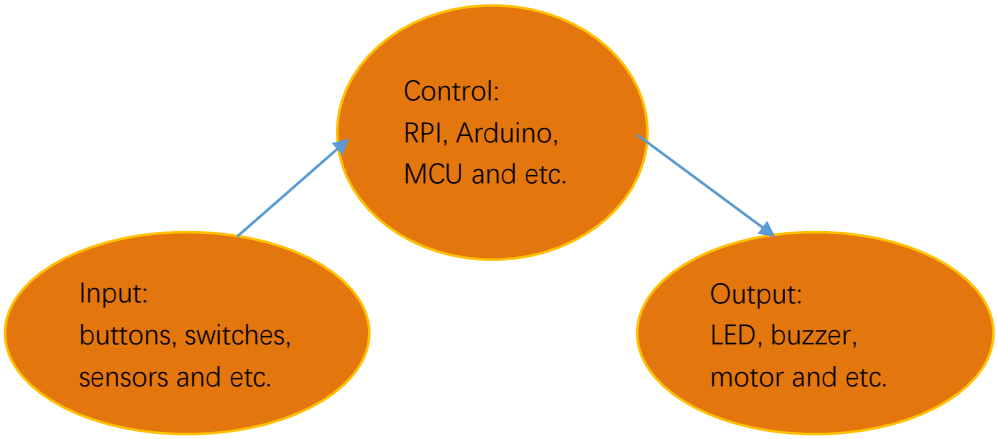
About RPi.GPIO：

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 funtions related to RPi.GPIO, please refer to:
https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In the 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 experiment, 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.

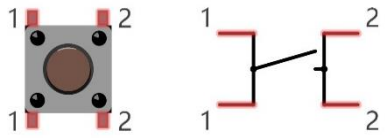
Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push button x1 
Jumper M/M x5 				

Component knowledge

Push button

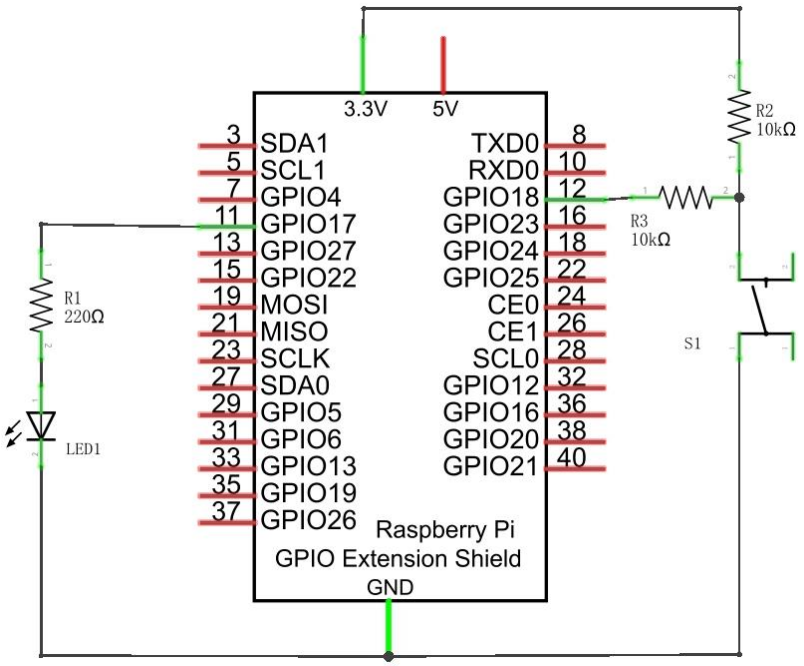
Push button has 4 pins. Two pins on the left is connected, and the right is similar as the left, which is shown in the below:



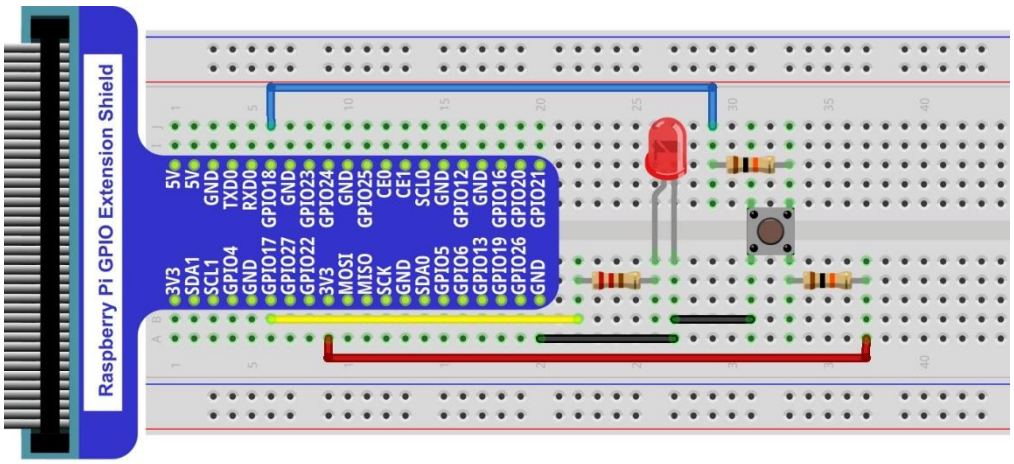
When the push button is pressed, the circuit is turned on.

Circuit

Schematic diagram



Hardware connection



Code

This experiment is designed for 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 experimental phenomena, then analyse the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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
```

Latter, 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:

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

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

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

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

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

        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");
```

```
28          }
29      }
30      return 0;
31  }
```

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

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

About digitalRead():

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.

The code of Python language is shown below.

Python Code 2.1.1 ButtonLED

First, observe the experimental phenomena, then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/02.1.1_ButtonLED
```

2. Use Python command to execute btnLED.py.

```
python ButtonLED.py
```

Latter, 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 the ledPin
4  buttonPin = 12    # define the buttonPin
5
6  def setup():
7      print 'Program is starting...'
8      GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
```

```
9       GPIO.setup(ledPin, GPIO.OUT)   # Set ledPin's mode is output
10      GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
11  input, and pull up to high level(3.3V)
12
13  def loop():
14      while True:
15          if GPIO.input(buttonPin)==GPIO.LOW:
16              GPIO.output(ledPin,GPIO.HIGH)
17              print 'led on ...'
18          else :
19              GPIO.output(ledPin,GPIO.LOW)
20              print 'led off ...'
21
22  def destroy():
23      GPIO.output(ledPin, GPIO.LOW)     # led off
24      GPIO.cleanup()                     # Release resource
25
26  if __name__ == '__main__':     # Program start from here
27      setup()
28      try:
29          loop()
30      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
31  will be executed.
32          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 the ledPin
buttonPin = 12    # define the buttonPin
def setup():
    print 'Program is starting...'
    GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT)   # Set ledPin's mode is output
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
input, and pull up to high level(3.3V)
```

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:
            GPIO.output(ledPin,GPIO.HIGH)
            print 'led on ...'
```

```
        else :
            GPIO.output(ledPin,GPIO.LOW)
            print 'led 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.

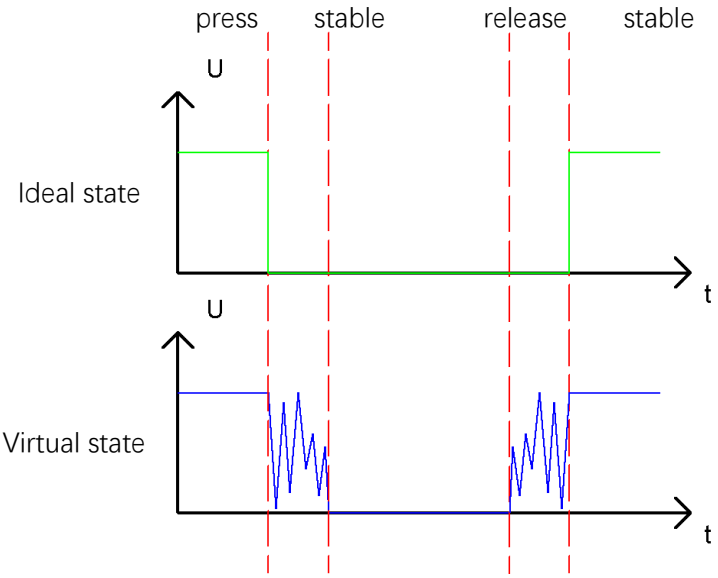
Project 2.2 MINI table lamp

We will also use a button, LED and UNO 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 chang 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 experiment, 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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 button state stable time
11 int reading;
12 int main(void)
13 {
14     if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
15         printf("setup wiringPi failed !");
16         return 1;
17     }
18     printf("Program is starting...\n");
19     pinMode(ledPin, OUTPUT);
20     pinMode(buttonPin, INPUT);
21
22     pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
23     while(1){
24         reading = digitalRead(buttonPin); //read the current state of button
25         if( reading != lastbuttonState){  //if the button state has changed ,record the
26 time point
27             lastChangeTime = millis();
28         }
```

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

Returns the number of milliseconds since the Arduino board began running the current program.

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

Python Code 2.2.1 Tablelamp

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

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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 the ledPin
4  buttonPin = 12    # define the buttonPin
5  ledState = False
6
7  def setup():
8      print 'Program is starting...'
9      GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
10     GPIO.setup(ledPin, GPIO.OUT)    # Set ledPin's mode is output
11     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
12 input, and pull up to high
13
14 def buttonEvent(channel):
```

```
    global ledState
    print 'buttonEvent GPIO%d'%channel
    ledState = not ledState
    if ledState :
        print 'Turn on LED ... '
    else :
        print 'Turn off LED ... '
    GPIO.output(ledPin,ledState)

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

def destroy():
    GPIO.output(ledPin, GPIO.LOW)     # led off
    GPIO.cleanup()                     # Release resource

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

RPi.GPIO provides us with a simple and effective funtion 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):
    global ledState
    print 'buttonEvent GPIO%d'%channel
    ledState = not ledState
    if ledState :
        print 'Turn on LED ... '
    else :
        print 'Turn off LED ... '
    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

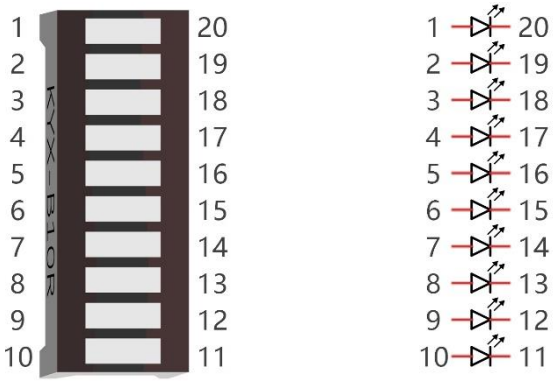
Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

Component knowledge

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

LED bar graph

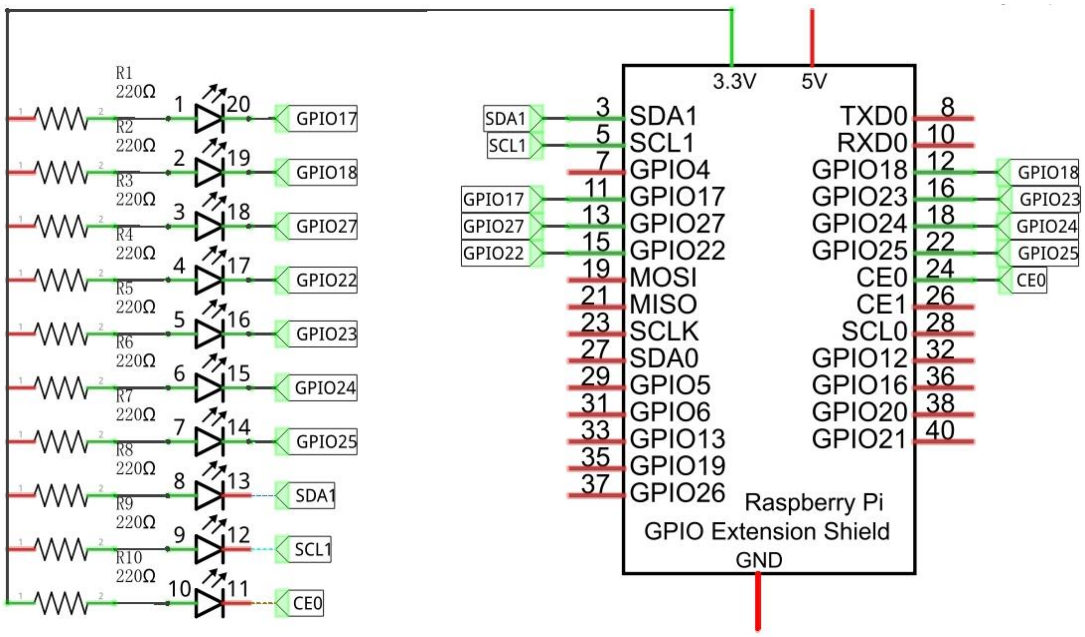
LED bar graph is a component Integration consist of 10 LEDs. There are two rows of pins at its bottom.



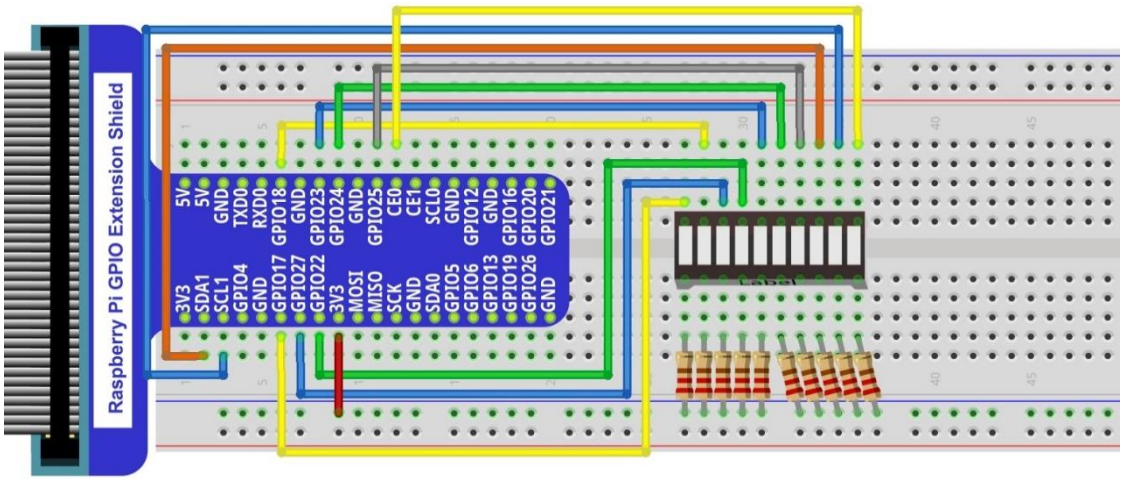
Circuit

The network label is used in the circuit diagram below, and the pins with the same network label are connected togather.

Schematic diagram



Hardware connection



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 experiment 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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:

```
#include <wiringPi.h>
#include <stdio.h>
#define leds 10
int pins[leds] = {0,1,2,3,4,5,6,8,9,10};
void led_on(int n)//make led_n on
{
    digitalWrite(n, LOW);
}

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

int main(void)
{
    int i;
    printf("Program is starting ... \n");
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(i=0;i<leds;i++){       //make leds pins' mode is output
        pinMode(pins[i], OUTPUT);
    }
    while(1){
        for(i=0;i<leds;i++){   //make led on from left to right
```

```
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
        for(i=leds-1;i>-1;i--){    //make led on from right to left
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
    }
    return 0;
}
```

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<leds;i++){    //make led on from left to right
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
        for(i=leds-1;i>-1;i--){    //make led on from right to left
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
    }
```

Python Code 3.1.1 LightWater

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

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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:

```
import RPi.GPIO as GPIO
import time
ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]
def setup():
    print 'Program is starting...'
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    for pin in ledPins:
        GPIO.setup(pin, GPIO.OUT)   # Set all ledPins' mode is output
```

```
9           GPIO.output(pin, GPIO.HIGH) # Set all ledPins to high(+3.3V) to off led
10  def loop():
11      while True:
12          for pin in ledPins:     #make led on from left to right
13              GPIO.output(pin, GPIO.LOW)
14              time.sleep(0.1)
15              GPIO.output(pin, GPIO.HIGH)
16          for pin in ledPins[10:0:-1]:        #make led on from right to left
17              GPIO.output(pin, GPIO.LOW)
18              time.sleep(0.1)
19              GPIO.output(pin, GPIO.HIGH)
20  def destroy():
21      for pin in ledPins:
22          GPIO.output(pin, GPIO.HIGH)     # turn off all leds
23      GPIO.cleanup()                      # Release resource
24  if __name__ == '__main__':     # Program start from here
25      setup()
26      try:
27          loop()
28      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
29  will be  executed.
30          destroy()
```

In the program, first define 10 pins connected to LED, and set them to output mode in the sub function setup(). Then in the loop() function, use two “for” cycles to relize flowing water light from right to left and from left to right. Among them, ledPins[10:0:-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[10:0:-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 the previous study, we have known that the 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.

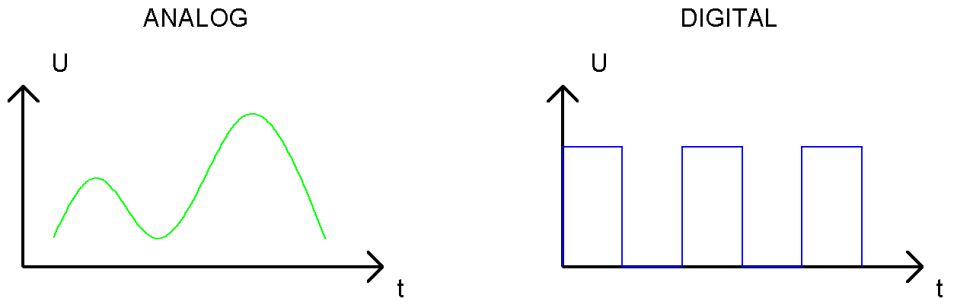
Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED x1 	Resistor 220Ω x1 
Jumper M/M x2 		

Circiut 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 appea a sudden change directly from 0℃ to 10℃, 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

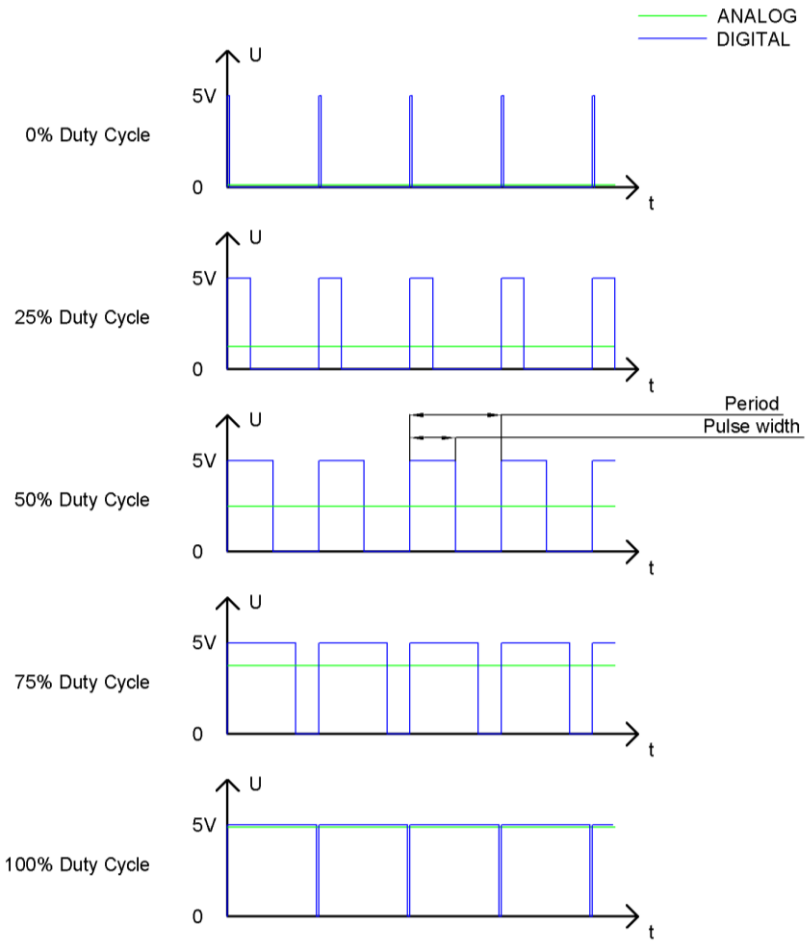
two forms (0 or 1), so it has strong stability. And digital signal and analog signal can be converted to each other.

PWM

PWM, namely Width Modulation Pulse, 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%:

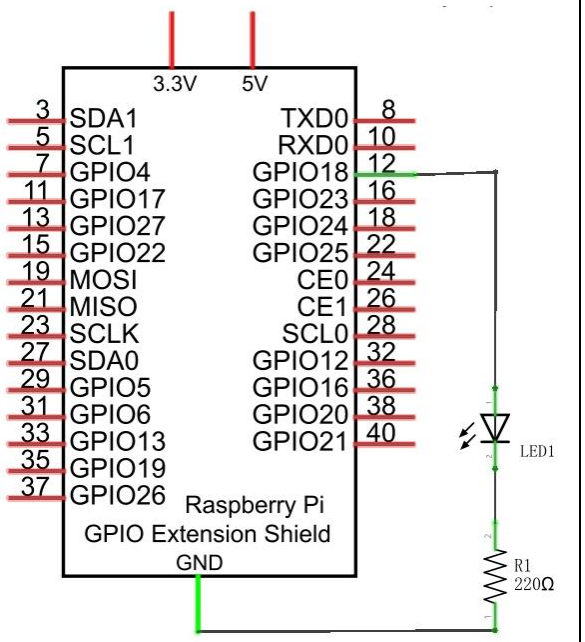
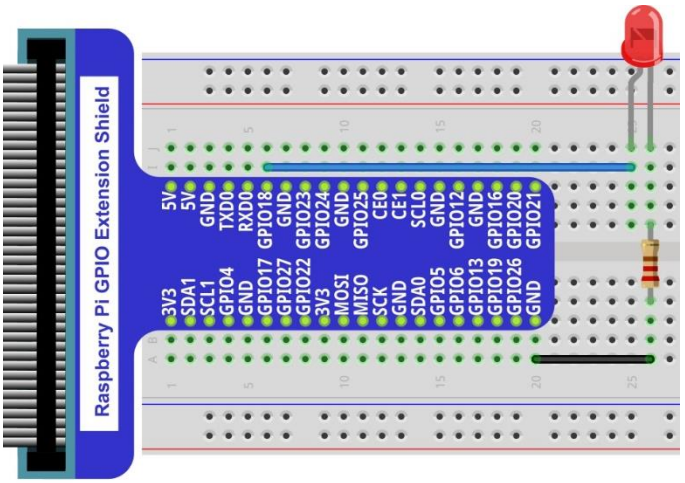


The larger PWM duty cycle is, the lager the output power will be. So we can use PWM to control the brightness of LED, the speed of DC motor and so on.

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 PWM with a 10 bit accuracy, that is, 100% of the pulse width can be divided into 2^{10}=1024 equal parts.

Circuit

Schematic diagram	Hardware connection
	

Code

This experiment 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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 turn 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  #define ledPin    1 //Only GPIO18 can output PWM
4  int main(void)
5  {
6      int i;
7      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
8          printf("setup wiringPi failed !");
```

```
9           return 1;
10      }
11
12      pinMode(ledPin, PWM_OUTPUT);//pwm output mode
13      while(1){
14          for(i=0;i<1024;i++){
15              pwmWrite(ledPin, i);
16              delay(2);
17          }
18          delay(300);
19          for(i=1023;i>=0;i--){
20              pwmWrite(ledPin, i);
21              delay(2);
22          }
23          delay(300);
24      }
25      return 0;
26  }
```

Since only GPIO18 of RPi has hardware capability to output PWM, the ledPin should be defined as 1 and set its output mode to PWM_OUTPUT based on the corresponding chart for pins.

```
    pinMode(ledPin, PWM_OUTPUT);//pwm output mode
```

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<1024;i++){
            pwmWrite(ledPin, i);
            delay(2);
        }
        delay(300);
        for(i=1023;i>=0;i--){
            pwmWrite(ledPin, i);
            delay(2);
        }
        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.

void pwmWrite (int pin, int value) ;

Writes the value to the PWM register for the given pin. The Raspberry Pi has one on-board PWM pin, pin 1 (BCM_GPIO 18, Phys 12) and the range is 0-1024. .

Python Code 4.1.1 BreathingLED

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

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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 turn from on to off and then from off to on gradually like breathing.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
LedPin = 12
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)    # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.LOW)   # Set LedPin to low
    p = GPIO.PWM(LedPin, 1000)      # set Frequece to 1KHz
    p.start(0)                      # Duty Cycle = 0
def loop():
    while True:
        for dc in range(0, 101, 1):   # Increase duty cycle: 0~100
            p.ChangeDutyCycle(dc)     # Change duty cycle
            time.sleep(0.01)
        time.sleep(1)
        for dc in range(100, -1, -1): # Decrease duty cycle: 100~0
            p.ChangeDutyCycle(dc)
            time.sleep(0.01)
        time.sleep(1)
def destroy():
    p.stop()
    GPIO.output(LedPin, GPIO.LOW)    # turn off led
    GPIO.cleanup()
if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
will be  executed.
        destroy()
```

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
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)    # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.LOW)   # Set LedPin to low
    p = GPIO.PWM(LedPin, 1000)      # set Frequece to 1KHz
    p.start(0)                      # Duty Cycle = 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):  # Increase duty cycle: 0~100
            p.ChangeDutyCycle(dc)     # Change duty cycle
            time.sleep(0.01)
        time.sleep(1)
        for dc in range(100, -1, -1): # Decrease duty cycle: 100~0
            p.ChangeDutyCycle(dc)
            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。

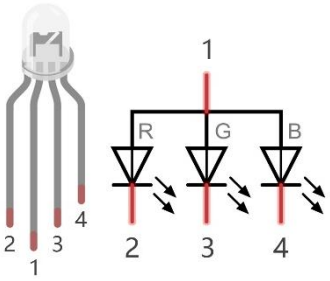
For more details about usage method for PMW of RPi.GPIO, please refer to:

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

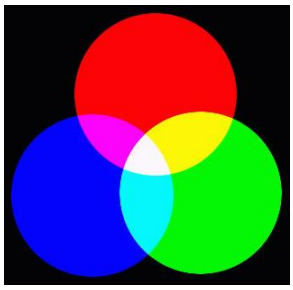
Chapter 5 RGBLED

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

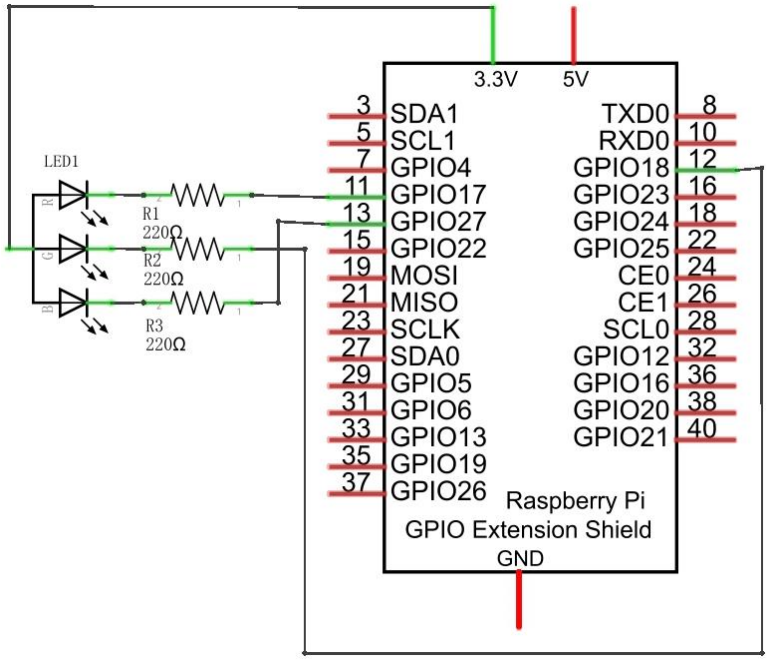
In this project, we will make a colorful LED. And we can control RGBLED to switch different colors automaticly.

Component List

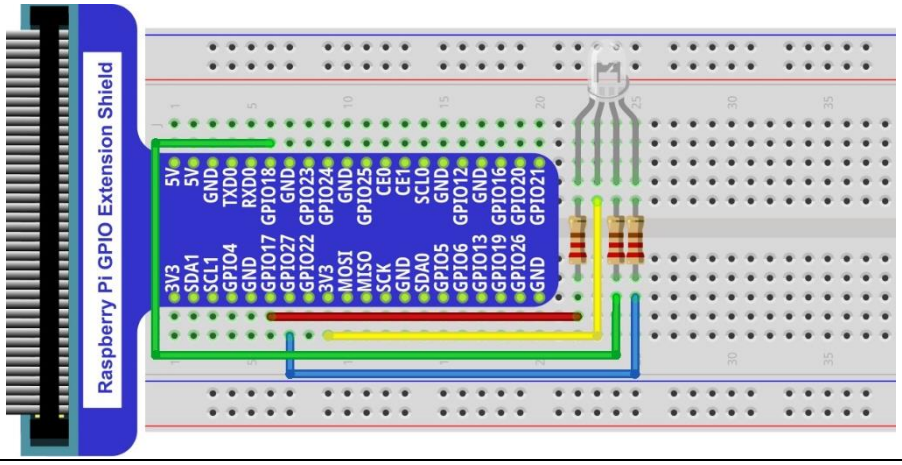
Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	RGBLED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

Circuit

Schematic diagram



Hardware connection



Code

Since this test 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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:

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

#define ledPinRed    0
#define ledPinGreen  1
#define ledPinBlue   2

void ledInit(void)
{
    softPwmCreate(ledPinRed,   0, 100);
    softPwmCreate(ledPinGreen, 0, 100);
    softPwmCreate(ledPinBlue,  0, 100);
}

void ledColorSet(int r_val, int g_val, int b_val)
{
    softPwmWrite(ledPinRed,   r_val);
    softPwmWrite(ledPinGreen, g_val);
    softPwmWrite(ledPinBlue,  b_val);
}

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

```
28          return 1;
29      }
30      printf("Program is starting ...\n");
31      ledInit();
32
33      while(1){
34          r=random()%100;
35          g=random()%100;
36          b=random()%100;
37          ledColorSet(r,g,b);
38          printf("r=%d,   g=%d,   b=%d \n",r,g,b);
39          delay(300);
40      }
41      return 0;
42  }
```

First, in the sub function of ledInitl(), create the software PWM control pins used to control the R G, RGBLED, B pin respectively.

```
void ledInit(void)
{
    softPwmCreate(ledPinRed,   0,  100);
    softPwmCreate(ledPinGreen, 0,  100);
    softPwmCreate(ledPinBlue,  0,  100);
}
```

Then create the sub function, and set the PWM of three pins.

```
void ledColorSet(int r_val, int g_val, int b_val)
{
    softPwmWrite(ledPinRed,    r_val);
    softPwmWrite(ledPinGreen,  g_val);
    softPwmWrite(ledPinBlue,   b_val);
}
```

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;
        g=random()%100;
        b=random()%100;
        ledColorSet(r,g,b);
        printf("r=%d,   g=%d,   b=%d \n",r,g,b);
        delay(300);
    }
```

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

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.
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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/05.1.1_ColorfulLED
```

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

```
python ColorfulLED.py
```

After the program is executed, you will see that the 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  pins = {'pin_R':11, 'pin_G':12, 'pin_B':13}  # pins is a dict
5  def setup():
6      global p_R,p_G,p_B
7      print 'Program is starting ... '
8      GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
9      for i in pins:
10         GPIO.setup(pins[i], GPIO.OUT)   # Set pins' mode is output
11         GPIO.output(pins[i], GPIO.HIGH) #Set pins to high(+3.3V) to off led
12     p_R = GPIO.PWM(pins['pin_R'], 2000)  # set Frequece to 2KHz
13     p_G = GPIO.PWM(pins['pin_G'], 2000)
14     p_B = GPIO.PWM(pins['pin_B'], 2000)
15     p_R.start(0)       # Initial duty Cycle = 0
16     p_G.start(0)
17     p_B.start(0)
18 def setColor(r_val,g_val,b_val):
19     p_R.ChangeDutyCycle(r_val)      # Change duty cycle
20     p_G.ChangeDutyCycle(g_val)
21     p_B.ChangeDutyCycle(b_val)
22 def loop():
23     while True :
24         r=random.randint(0,100)
25         g=random.randint(0,100)
26         b=random.randint(0,100)
```

```
27          setColor(r,g,b)
28          print 'r=%d, g=%d, b=%d '%(r ,g, b)
29          time.sleep(0.3)
30  def destroy():
31      p_R.stop()
32      p_G.stop()
33      p_B.stop()
34      GPIO.cleanup()
35  if __name__ == '__main__':     # Program start from here
36      setup()
37      try:
38          loop()
39      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
40  will be  executed.
41          destroy()
```

In the 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.o that the RGBLED switching of different colors randomly.

```
def loop():
    while True :
        r=random.randint(0,100)
        g=random.randint(0,100)
        b=random.randint(0,100)
        setColor(r,g,b)
        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

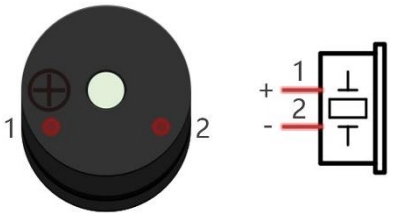
<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x7
  </td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

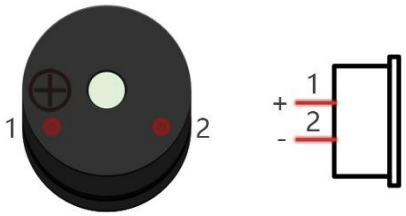
Buzzer

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.

Activebuzzer



Passive buzzer



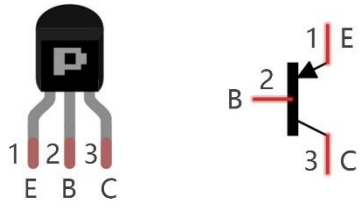
Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer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

Transistor

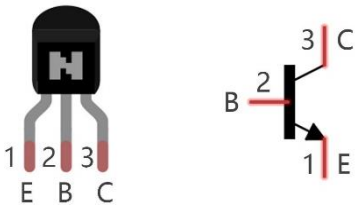
Due to the current operating of buzzer is so large that GPIO of RPi output capability can not be satisfied, a transistor of NPN type is needed here to amplify the current.

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

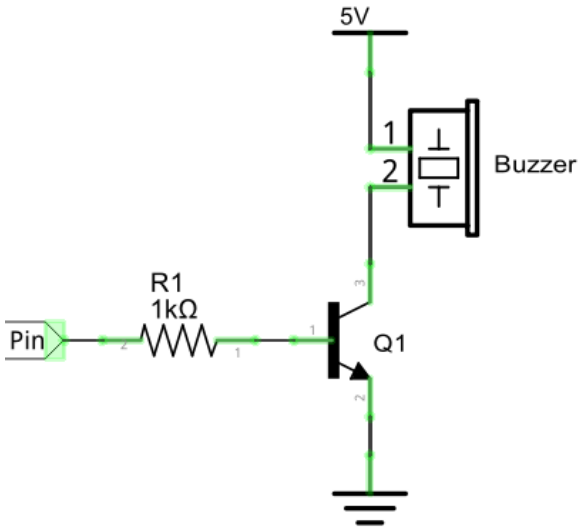
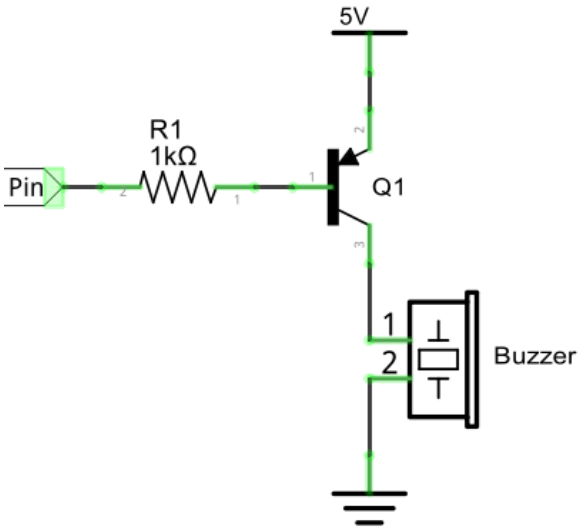


NPN transistor



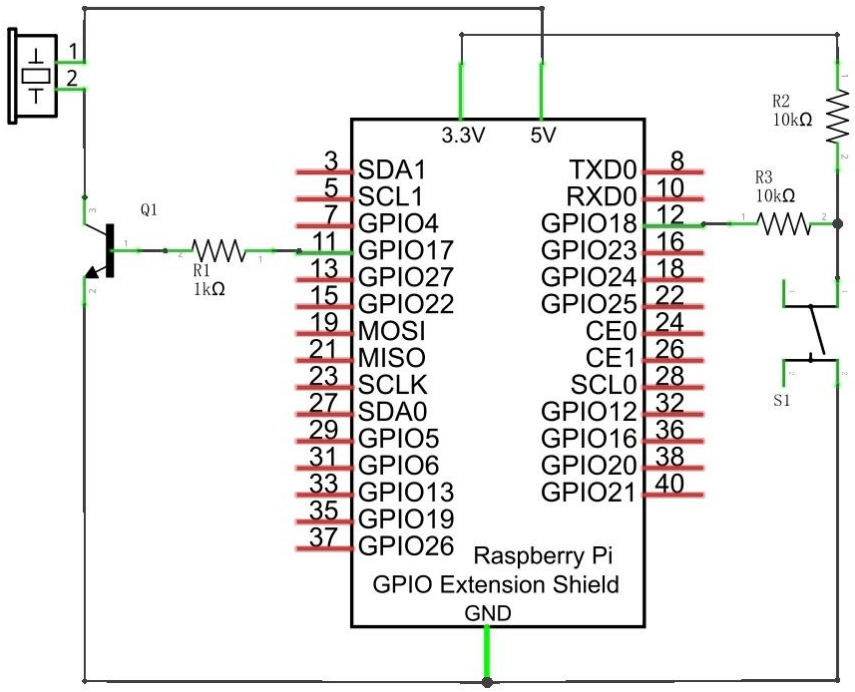
According to the transistor's characteristics, it is often used as a switch in digital circuits. For micro-controller's capacity of output current is very weak, we will use transistor to amplify current and drive large-current components.

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.

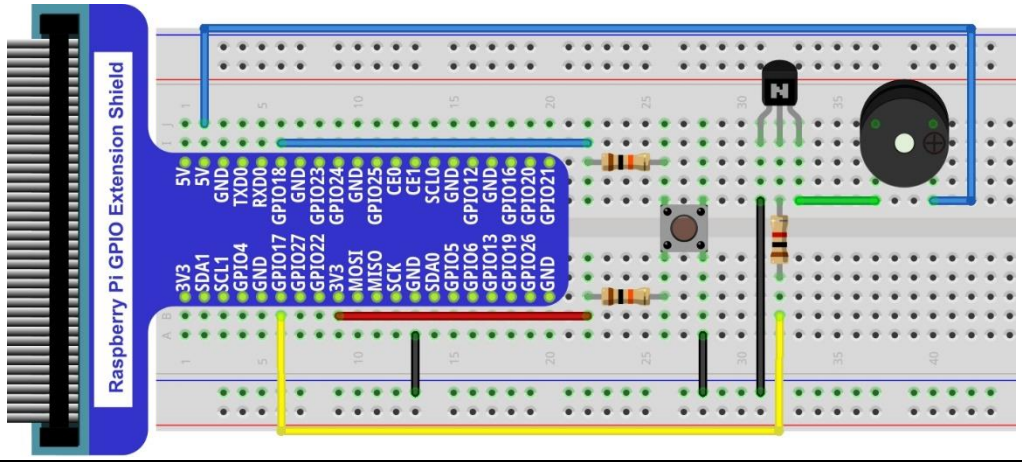
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

Circuit

Schematic diagram



Hardware connection



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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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:

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

#define buzzeRPin    0      //define the buzzeRPin
#define buttonPin 1     //define the buttonPin

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

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

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

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

```
29
30      return 0;
31  }
```

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

Python Code 6.1.1 Doorbell

First observe the running result, then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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 the buzzeRPin
4   buttonPin = 12    # define the buttonPin
5
6   def setup():
7       print 'Program is starting...'
8       GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
9       GPIO.setup(buzzeRPin, GPIO.OUT)    # Set buzzeRPin's mode is output
10      GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
11  input, and pull up to high level(3.3V)
12
13  def loop():
14      while True:
15          if GPIO.input(buttonPin)==GPIO.LOW:
16              GPIO.output(buzzeRPin,GPIO.HIGH)
17              print 'buzzer on ...'
18          else :
19              GPIO.output(buzzeRPin,GPIO.LOW)
20              print 'buzzer off ...'
21
22  def destroy():
23      GPIO.output(buzzeRPin, GPIO.LOW)      # buzzer off
24      GPIO.cleanup()                     # Release resource
25
26  if __name__ == '__main__':     # Program start from here
27      setup()
28      try:
29          loop()
```

```
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
will be  executed.
        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 functiont 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softTone.h>
#include <math.h>
#define buzzeRPin    0      //define the buzzeRPin
#define buttonPin    1      //define the buttonPin
void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ // The frequency is based on the sine curve.
        sinVal = sin(x * (M_PI / 180));
        toneVal = 2000 + sinVal * 500;
        softToneWrite(pin,toneVal);
        delay(1);
```

```
15      }
16  }
17  void stopAlertor(int pin){
18      softToneWrite(pin,0);
19  }
20  int main(void)
21  {
22      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
23          printf("setup wiringPi failed !");
24          return 1;
25      }
26      pinMode(buzzeRPin, OUTPUT);
27      pinMode(buttonPin, INPUT);
28      softToneCreate(buzzeRPin);
29      pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
30      while(1){
31          if(digitalRead(buttonPin) == LOW){ //button has pressed down
32              alertor(buzzeRPin);   //buzzer on
33              printf("alertor on...\n");
34          }
35          else {              //button has released
36              stopAlertor(buzzeRPin);   //buzzer off
37              printf("...buzzer off\n");
38          }
39      }
40      return 0;
41  }
```

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, the 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 running result, and then analyze the code.

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

```
cd Freenove_Basic_Starter_Kit_for_Raspberry_Pi/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     print 'Program is starting...'
11     GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
12     GPIO.setup(buzzeRPin, GPIO.OUT)   # Set buzzeRPin's mode is output
13     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
14 input, and pull up to high level(3.3V)
15     p = GPIO.PWM(buzzeRPin, 1)
16     p.start(0);
17
18 def loop():
19     while True:
20         if GPIO.input(buttonPin)==GPIO.LOW:
21             alertor()
22             print 'buzzer on ...'
23         else :
```

```
24              stopAlertor()
25              print 'buzzer off ...'
26  def alertor():
27      p.start(50)
28      for x in range(0,361):
29          sinVal = math.sin(x * (math.pi / 180.0))
30          toneVal = 2000 + sinVal * 500
31          p.ChangeFrequency(toneVal)
32          time.sleep(0.001)
33
34  def stopAlertor():
35      p.stop()
36  def destroy():
37      GPIO.output(buzzeRPin, GPIO.LOW)      # buzzer off
38      GPIO.cleanup()                        # Release resource
39  if __name__ == '__main__':     # Program start from here
40      setup()
41      try:
42          loop()
43      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
44  will be  executed.
45          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 creat PMW is also introduced before in the sections about BreathingLED and RGBLED.

```
def setup():
    global p
    print 'Program is starting...'
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(buzzeRPin, GPIO.OUT)    # Set buzzeRPin's mode is output
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
input, 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, the 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()
```

Chapter 7 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 7.1 Remote LED

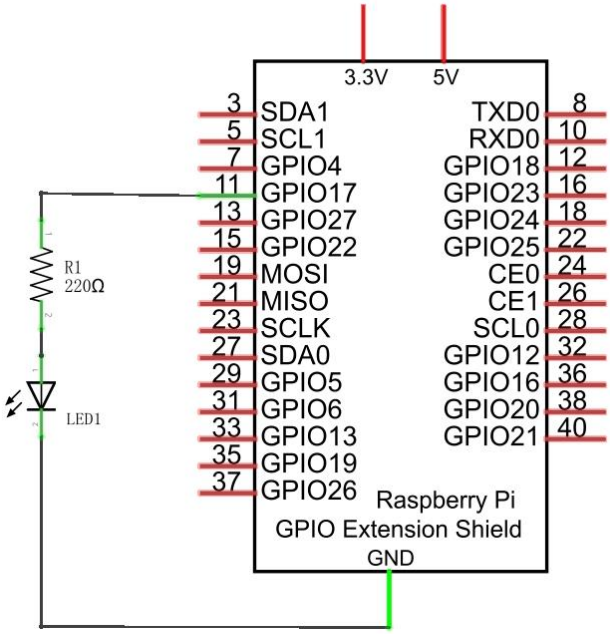
In this project, we need build a WebIOPi service, and then control the RPi GPIO to control a LED through Web browser of phone or PC.

Component List

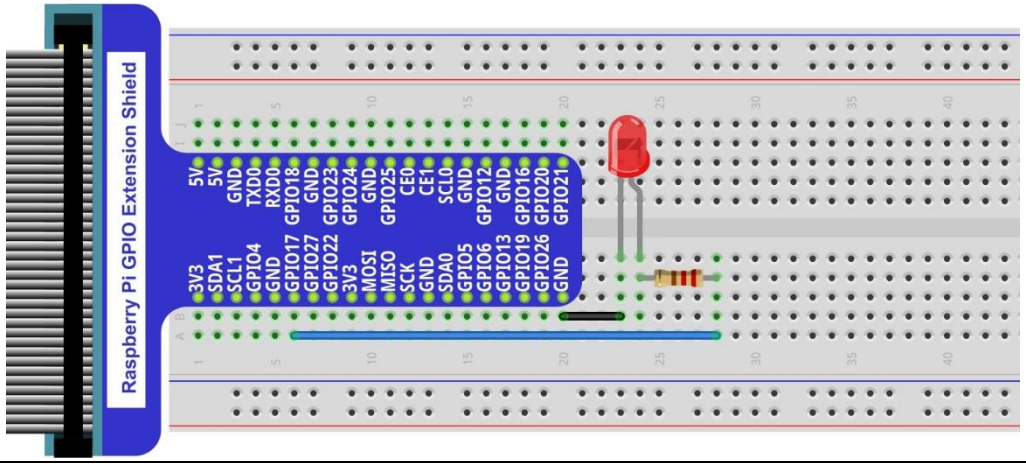
<table>
<tr><td>Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td rowspan="2">LED x1
  </td><td rowspan="2">Resistor 220Ω x1
  </td></tr>
<tr><td>Jumper M/M x2
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection



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 installation steps. The latest version（until 2016-6-27）WebIOPi is 0.7.1. So, if your RPi model is 2B or 3B, 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. visit http://webiopi.trouch.com/DOWNLOADS.html to get the latest installation package. You can use the following command to obtain.

```
wget http://sourceforge.net/projects/webiopi/files/WebIOPi-0.7.1.tar.gz/download
```

2. Rename the package toW ebIOPi.tar.gz。

```
mv download WebIOPi.tar.gz
```

3. Extract the package and generate a folder named "WebIOPi-0.7.1". Then enter the folder.

```
tar xvzf WebIOPi.tar.gz
cd WebIOPi-0.7.1
```

4. If your RPI version is 2B or 3B, then you need to modify some files to make it work properly. If your RPi is the other version, you don't need to modify and you can just skip this step.

   1)Edit the file "python/native/cpuinfo.c". Find "BCM2708" and change it to "BCM2709".

```
while(!feof(fp)) {
   fgets(buffer, sizeof(buffer) , fp);
   sscanf(buffer, "Hardware   : %s", hardware);
   if (strcmp(hardware, "BCM2708") == 0)
      rpi_found = 1;
   sscanf(buffer, "Revision   : %s", revision);
}
```

```
while(!feof(fp)) {
   fgets(buffer, sizeof(buffer) , fp);
   sscanf(buffer, "Hardware   : %s", hardware);
   if (strcmp(hardware, "BCM2709") == 0)
      rpi_found = 1;
   sscanf(buffer, "Revision   : %s", revision);
}
```

   2）Edit the file "python/native/gpio.c". Change “#define BCM2708_PERI_BASE 0x20000000” to “#define BCM2708_PERI_BASE 0x3f000000”.

```
#define BCM2708_PERI_BASE    0x20000000
#define BCM2708_PERI_BASE    0x3f000000
```

5. Run setup.sh to start the installation, and the process need a period of time to wait.

```
sudo ./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.

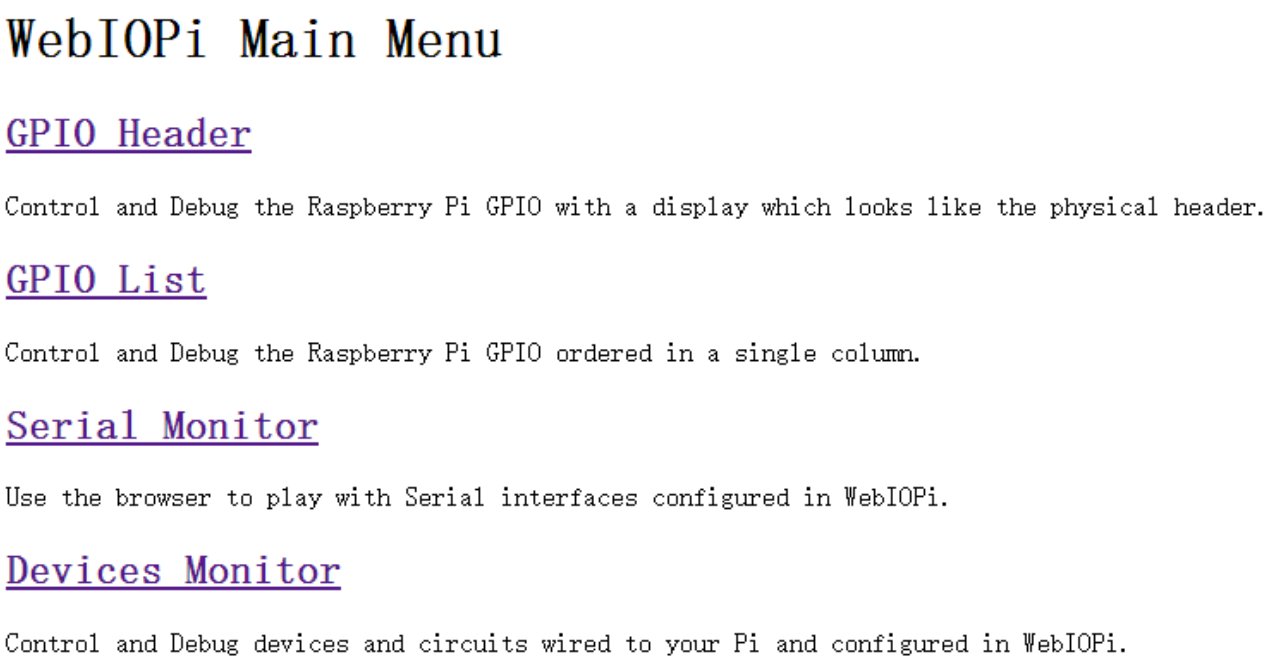
Until now, WebIOPi has been launched, and you can press "Ctrl+C" to terminate service.

Access WebIOPi over local network

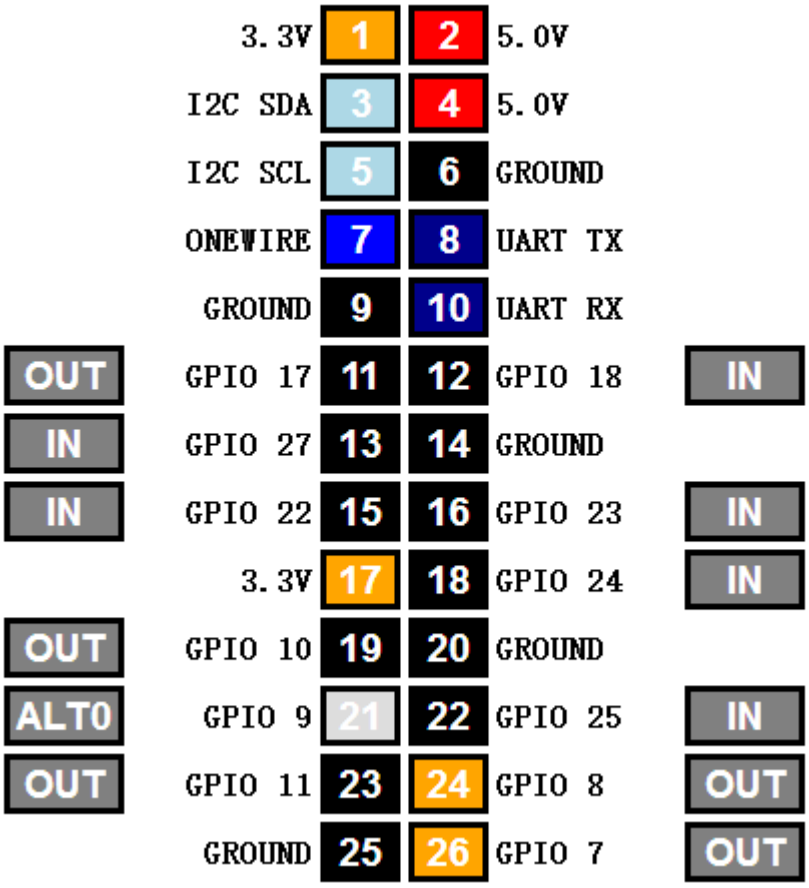
Under the same network, use mobile phone or PC browser to open your RPi IP address, and add port number like 8000. For example, my raspberry pie IP address is 192.168.1.109. Then, in the browser, should input: http://192.168.1.109:8000/

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

Then, enter the main control interface:



Click on GPIO Header to enter the GPIO control interface.



Control methods：

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

Completed

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

About WebIOPi

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

WebIOPi has not provide corresponding installation package for RPi 2B and 3B 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, missions or you have other ideas and questions about contents of this tutorial or the kit and etc, please feel free to contact us, and we will check and correct it as soon as possible.

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

Thank you again for choosing Freenove products.