Free your innovation

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

Warning

When you purchase or use Freenove Ultimate 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 Ultimate Starter Kit for Raspberry Pi, which includes the following folders and files:

- Datasheet Datasheet of electronic components and modules
- Code Code for experimental
- 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

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.

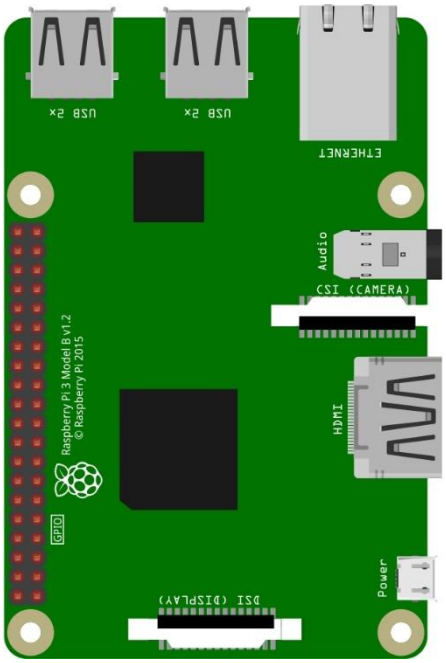
Raspberry Pi and Arduino have a lot of fans in the world. They are keen to exploration, innovation and DIY and they contributed a great number of high-quality open source code, circuit and rich knowledge base. So we can realize our own creativity more efficiently by using these free resource. Of course, 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 Ultimate 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.

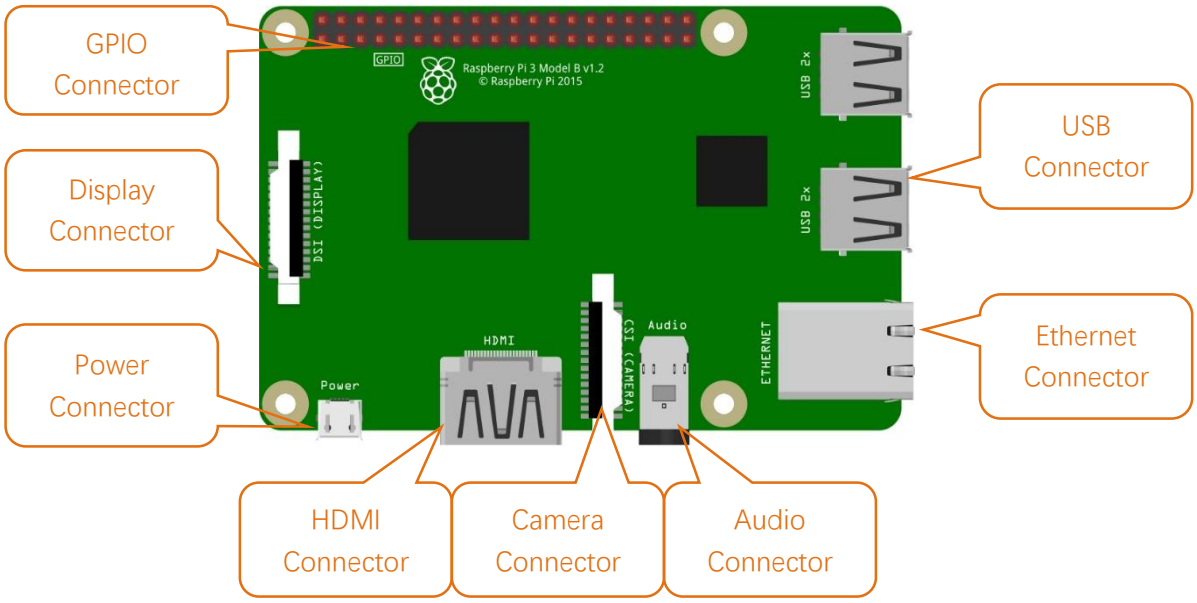
Raspberry Pi

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.

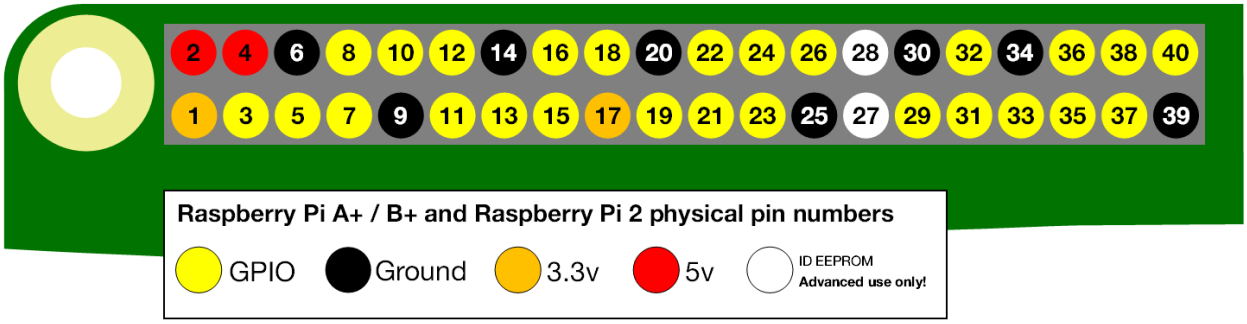
When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering.

GPIO NUMBERING

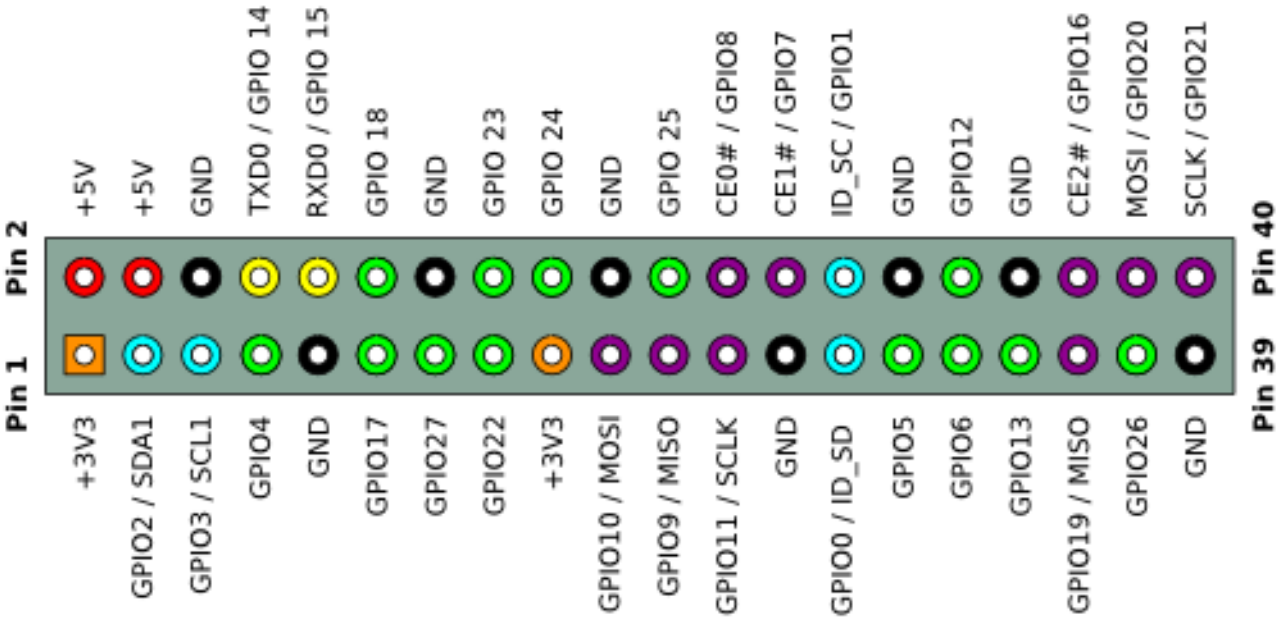
These are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. You will need a printed reference or a reference board that fits over the pins.

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:



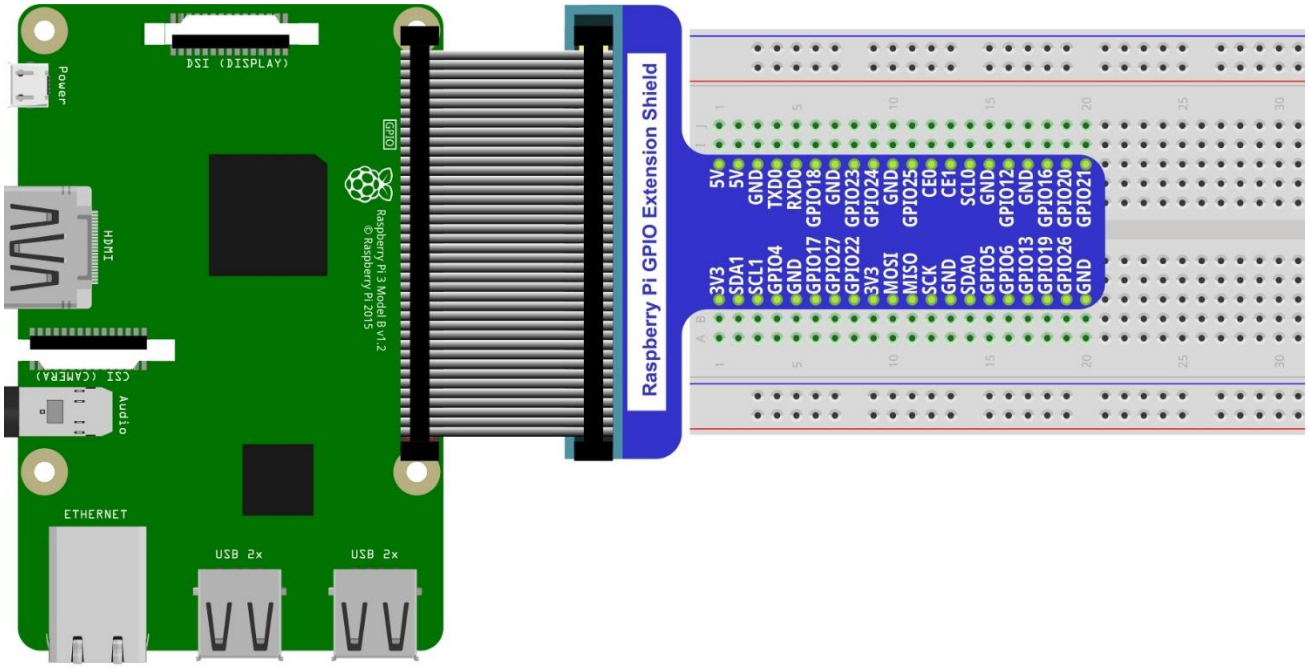
Each pin is defined as below:



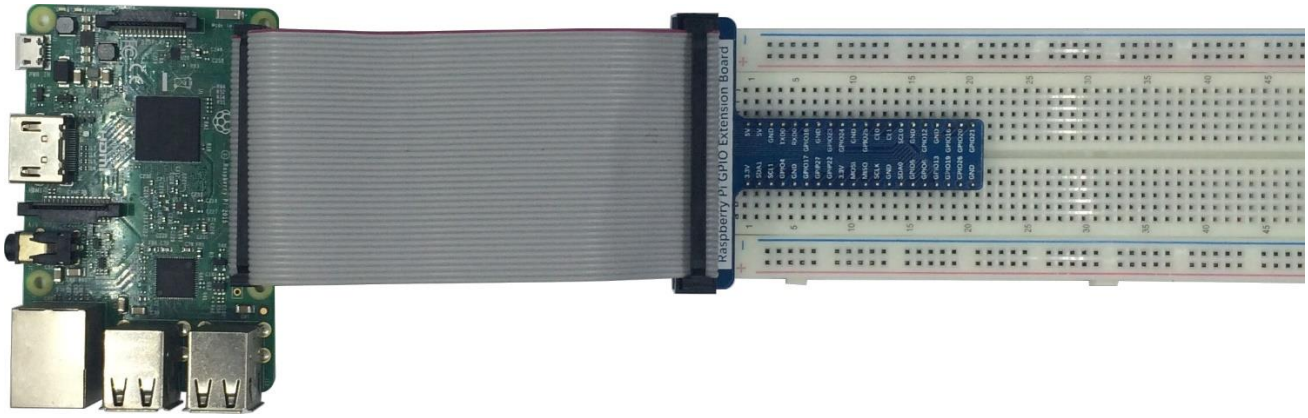
For more details about pin definition of GPIO, please refer to http://pinout.xyz/

GPIO Extension Board

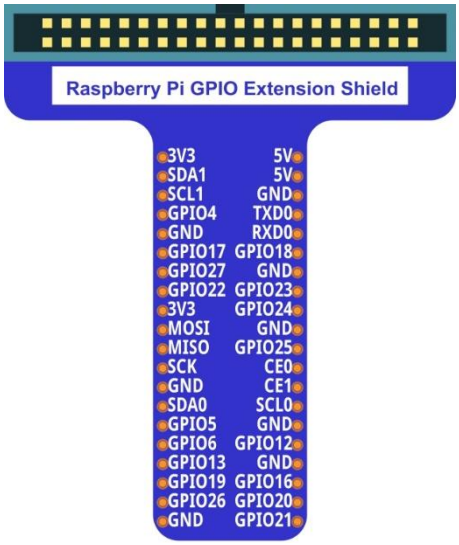
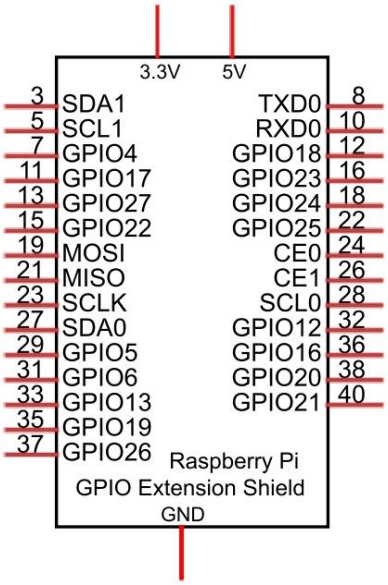
When we use RPi to do the experiment, we had better use GPIO, which is more convenient to extend all IO ports of RPi to the bread board directly. The GPIO sequence on Extension Board is identical to the GPIO sequence of RPi. Since the GPIO of different versions of RPi is different, the corresponding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



Practicality picture of connection:

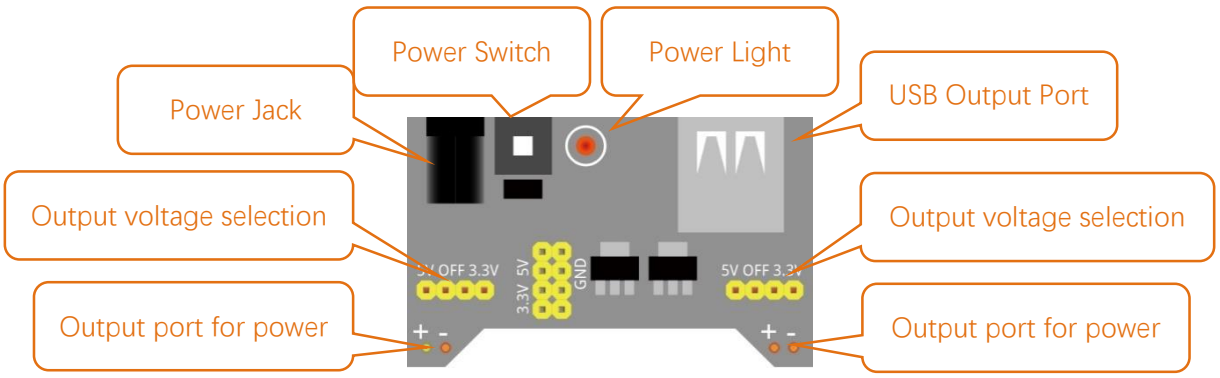


Among them, GPIO Extension Board and its schematic are shown below:

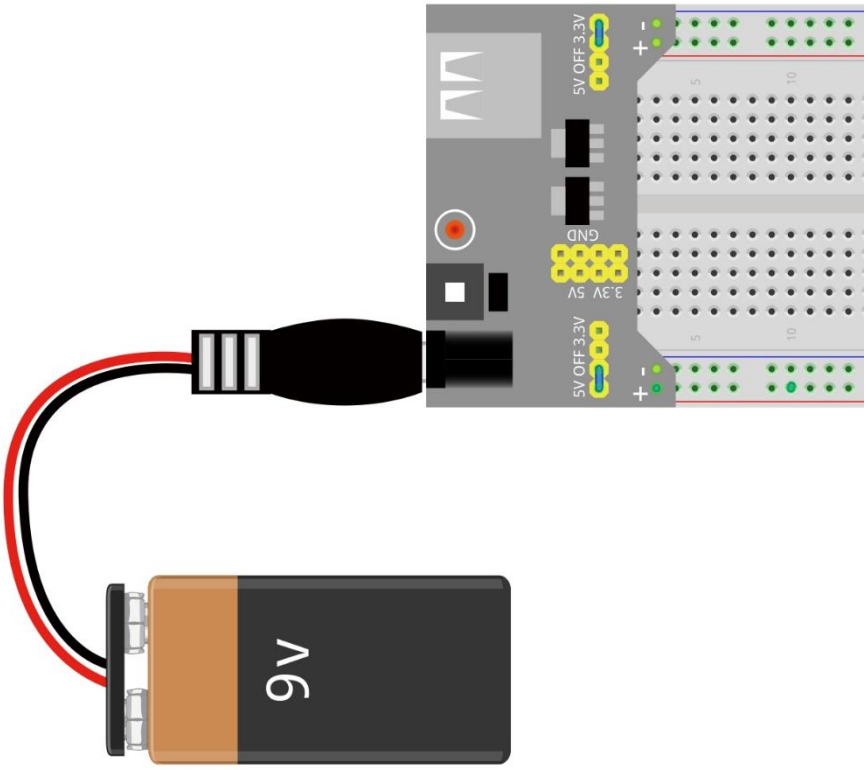
GPIO Extension Board	Definition of pins
	

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

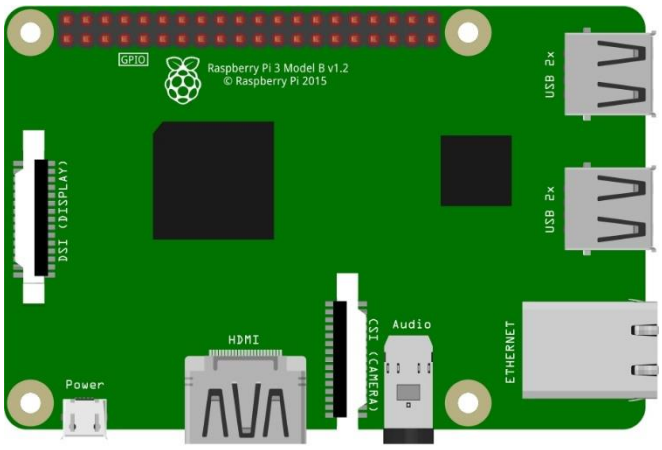
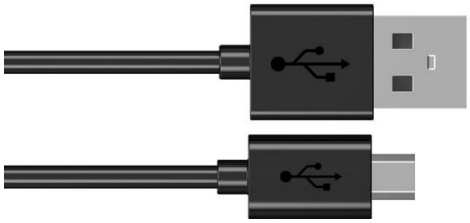
Why is “Chapter 0”? Because in the program code, all the counts are starting from 0. We choose to follow this rule (just a joke). In this chapter, we will do some necessary preparation work: start your Pi Raspberry and install some necessary libraries. If your Raspberry Pie can be started normally and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

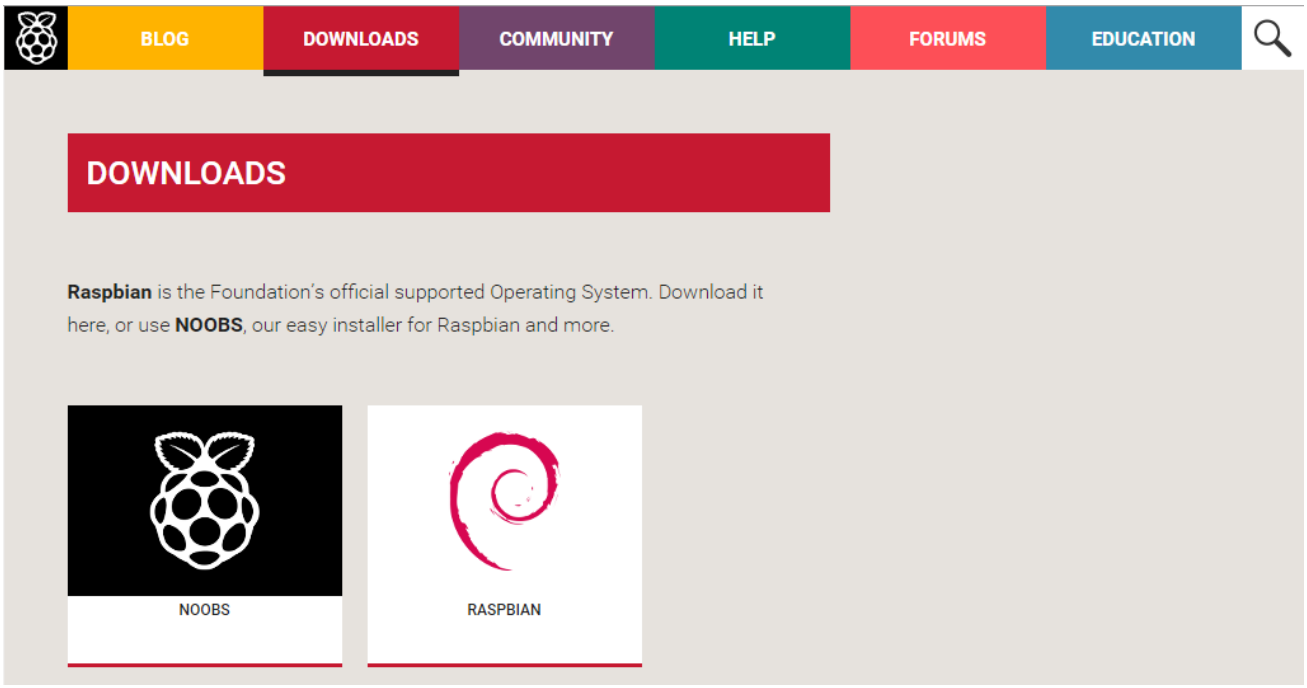
Among these optional components, the Display as a screen for RPi. If no, it does not matter, you can use a remote desktop of your personal PC to control your RPi. Mouse and keyboard are the same, if no, use remote desktop and share a set of keyboard and mouse personal with PC.

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

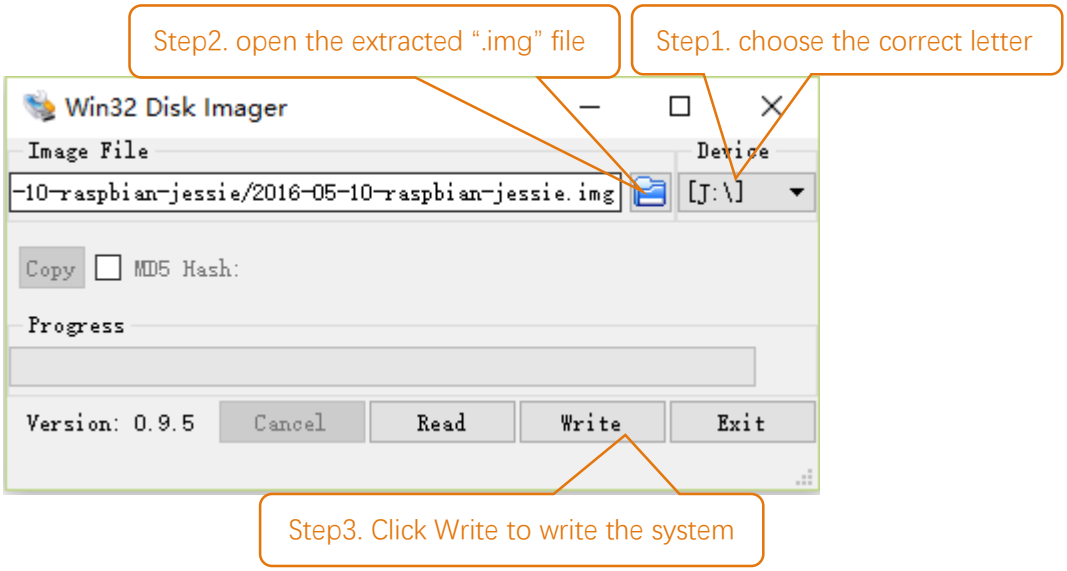
Visit RPi official website (https://www.Raspberry_Pi.org/), click "Downloads" and choose to download "RASPBIAN". RASPBIAN supported by RPI is an operating system based on Linux, which contains a number of contents required for RPi. We recommended RASPBIAN system to beginners. All experiments in this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



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

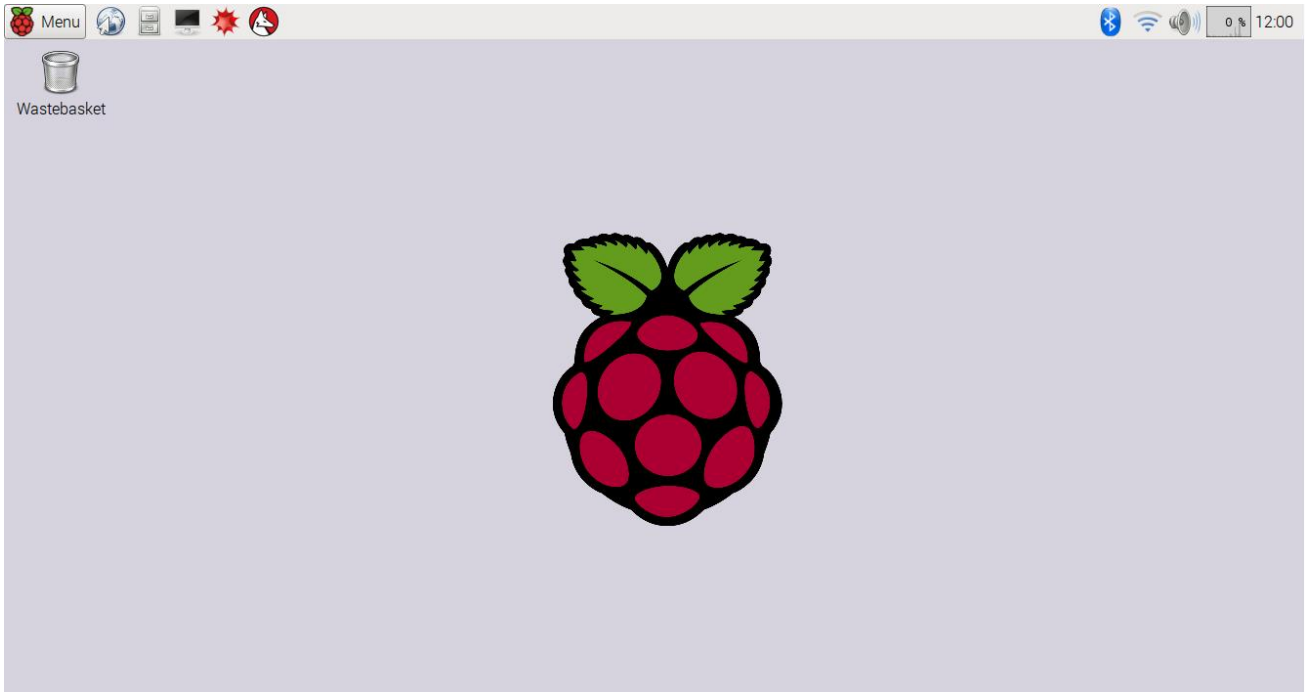
Write System to Micro SD Card

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

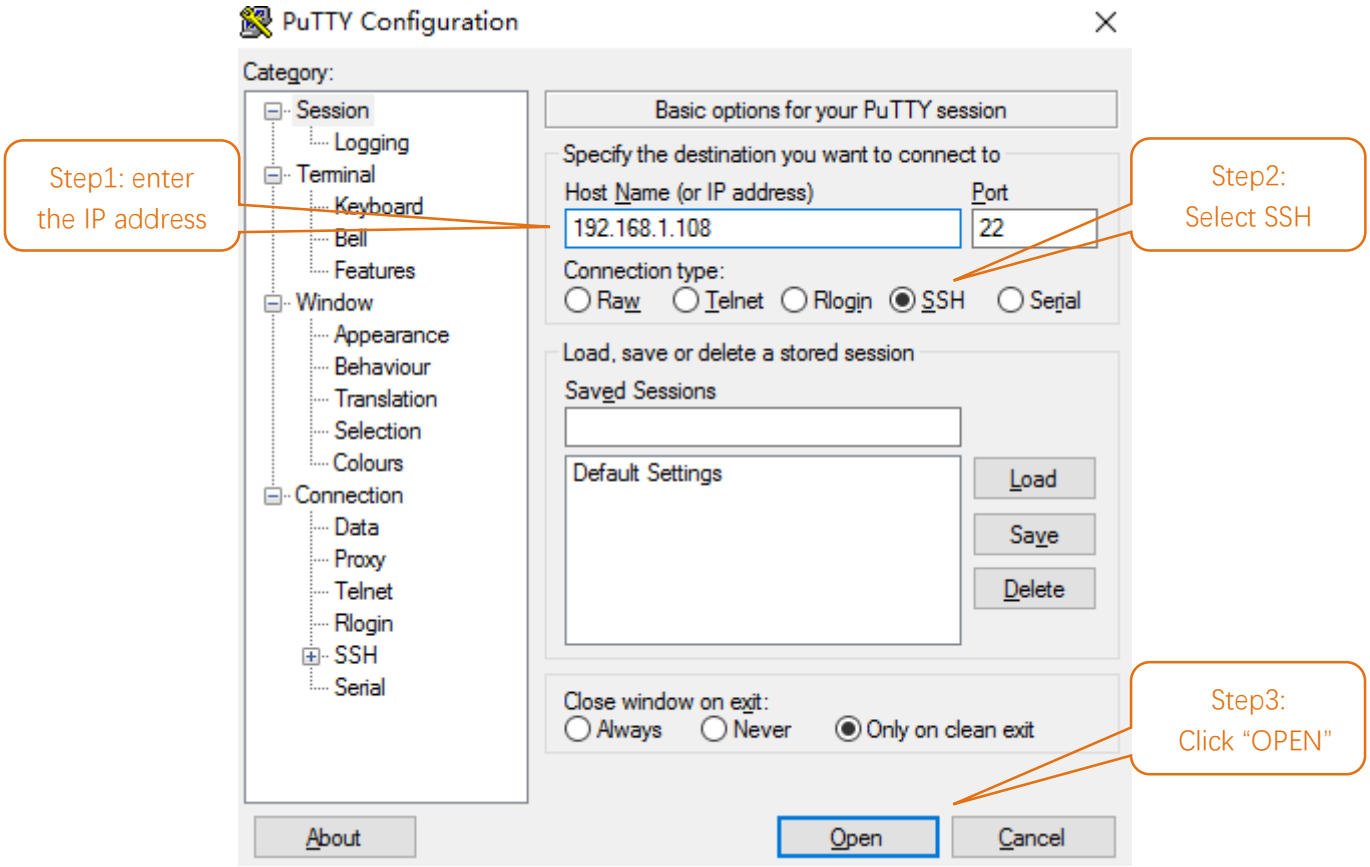
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use remote desktop to control RPi under the Windows operating system.

Install Xrdp Services

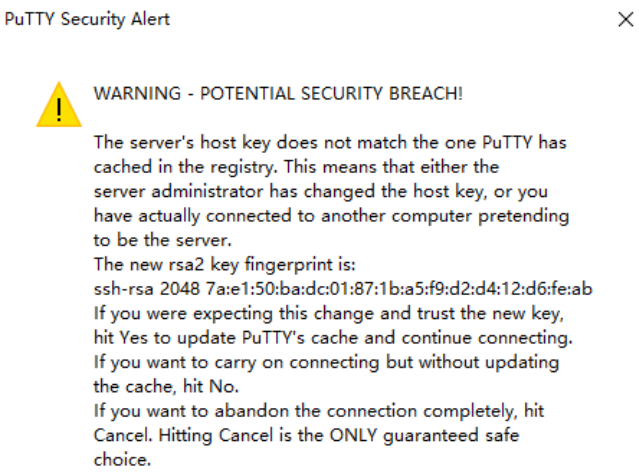
First, download the tool software Putty. Its official address: http://www.putty.org/

Or download it here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

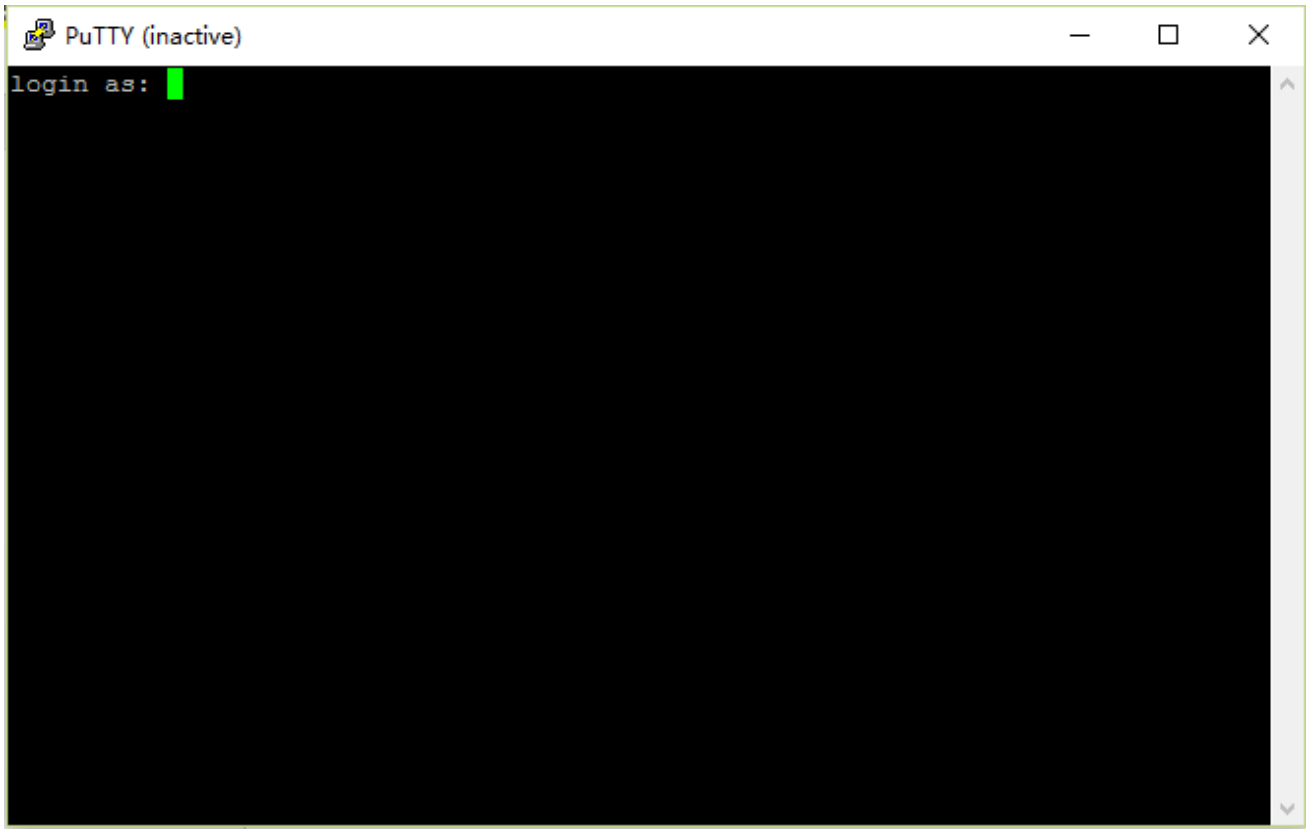
Then use cable to connect your RPi to the routers of your PC LAN to ensure your PC and your RPi in the same LAN. Then put the system TF card prepared before into the slot of the RPi and turn on the power supply waiting for the start RPi. Later, enter control terminal of the router to inquiry IP address 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:



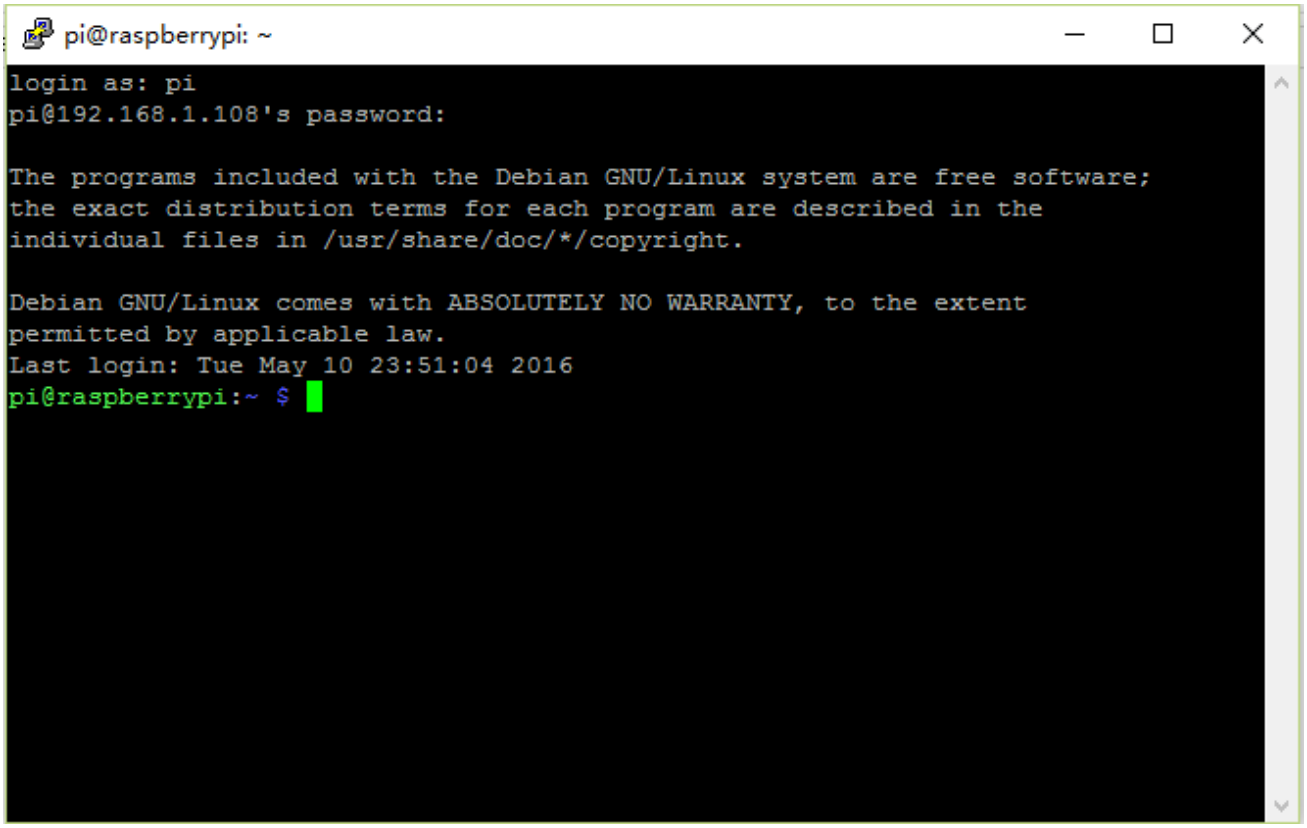
There will appear a security warning at first logining. Just click “YES”.



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.



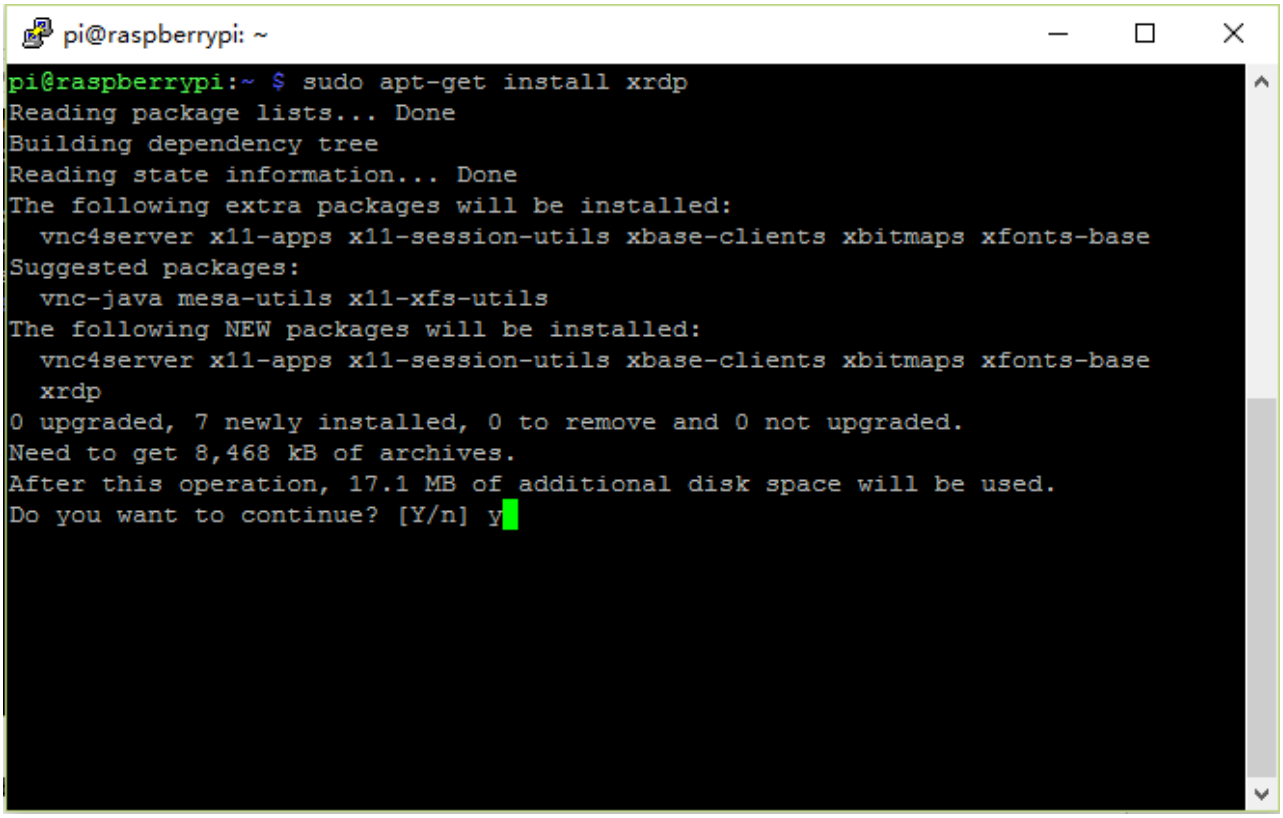
Then enter the command line of RPi, which means that you have successfully login to RPi command line mode.



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.

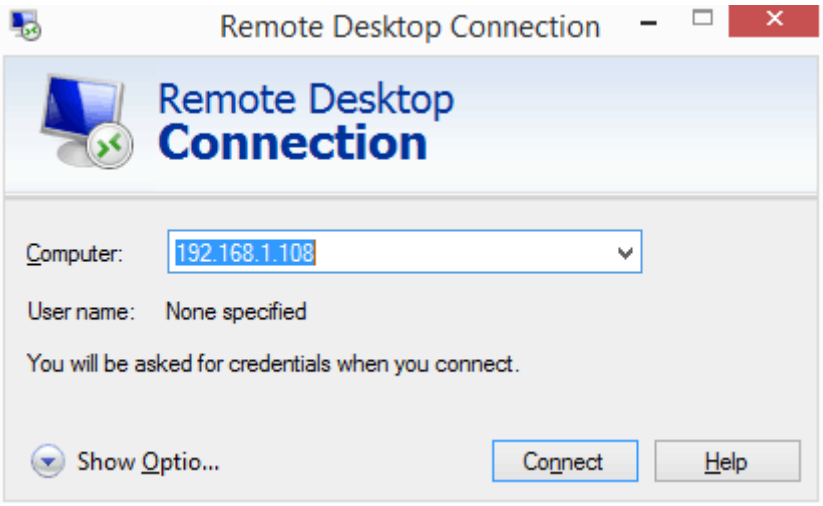


Enter "Y", press key “Enter” to confirm.

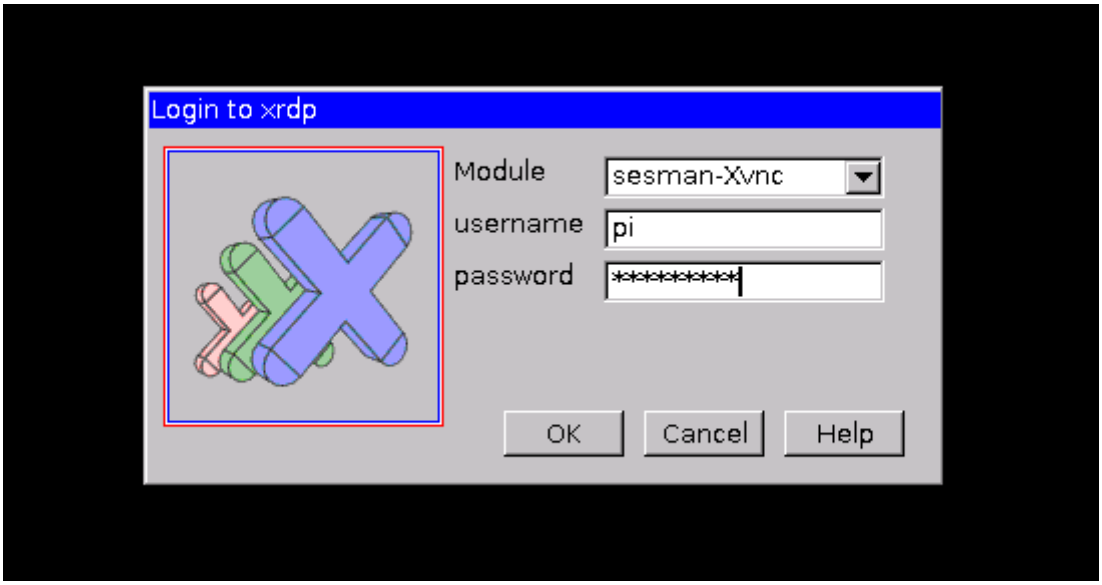
After the installation is completed, you can use Windows remote desktop applications to login to your RPi.

Login to Windows remote desktop

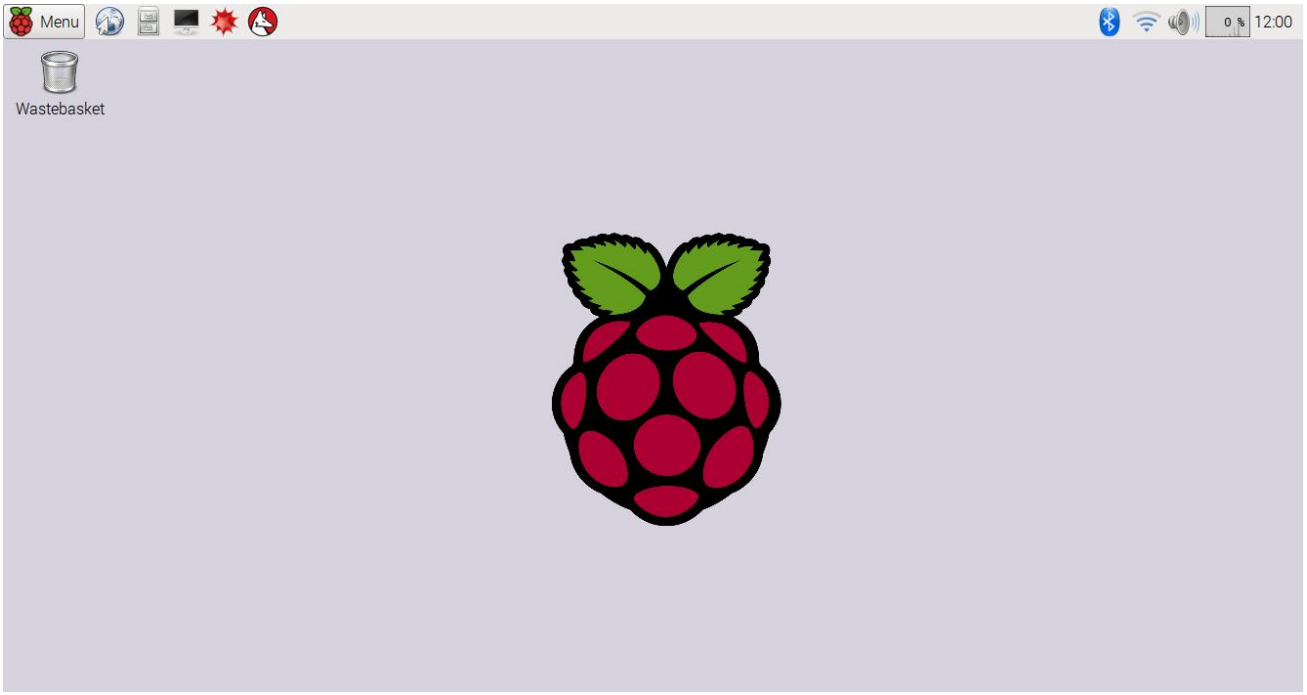
Use "WIN+R" or search function, open the remote desktop application "mstsc.exe" under Windows, enter the IP address of RPi and then click “Connect”.



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.



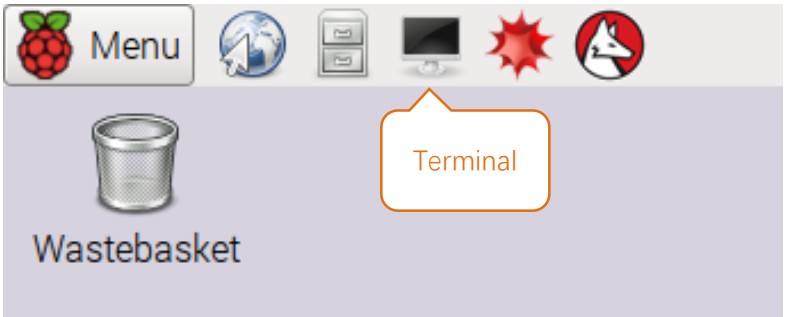
Here, you have successfully used the remote desktop login to RPi.
Then continue to do some preparation work: install a GPIO library wiringPi for your RPi.

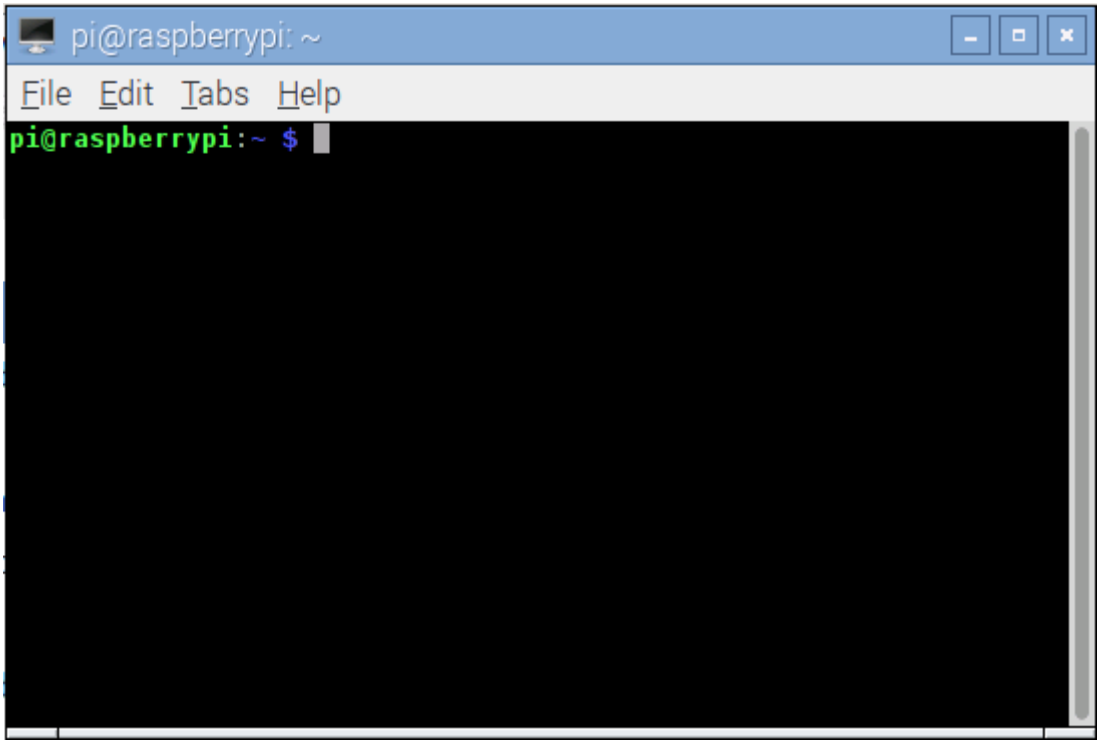
Step 0.2 Install WiringPi

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_Ultimate_Starter_Kit_for_Raspberry_Pi
```

Put the file into user directory pi/ and the code file into Freenove_Ultimate_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:

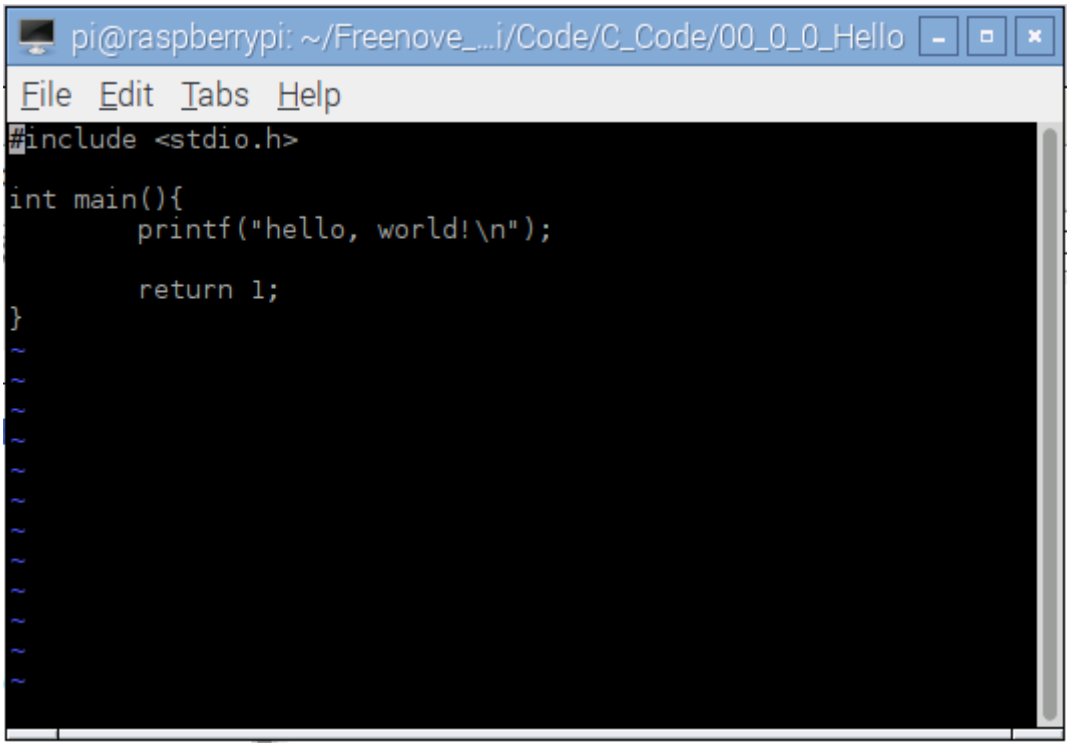
First, use the cd command to enter the sample code folder.

```
cd Freenove_Ultimate_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
```

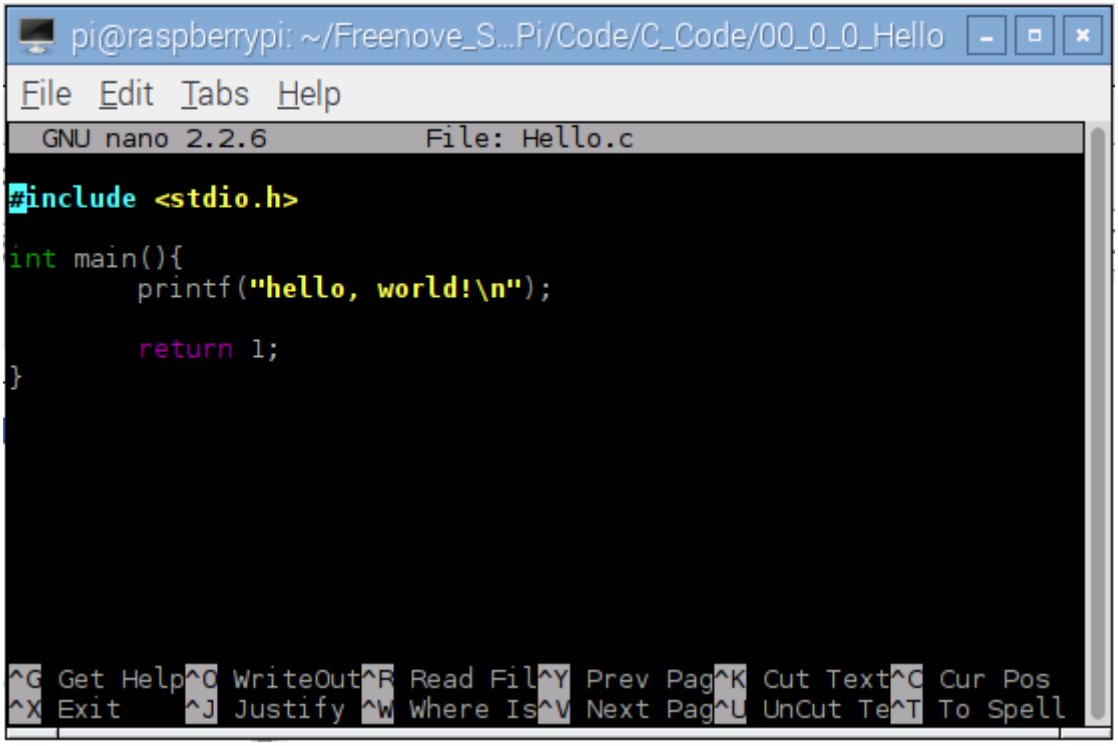
As is shown below:



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

```
nano Hello.c
```

As is hown below：



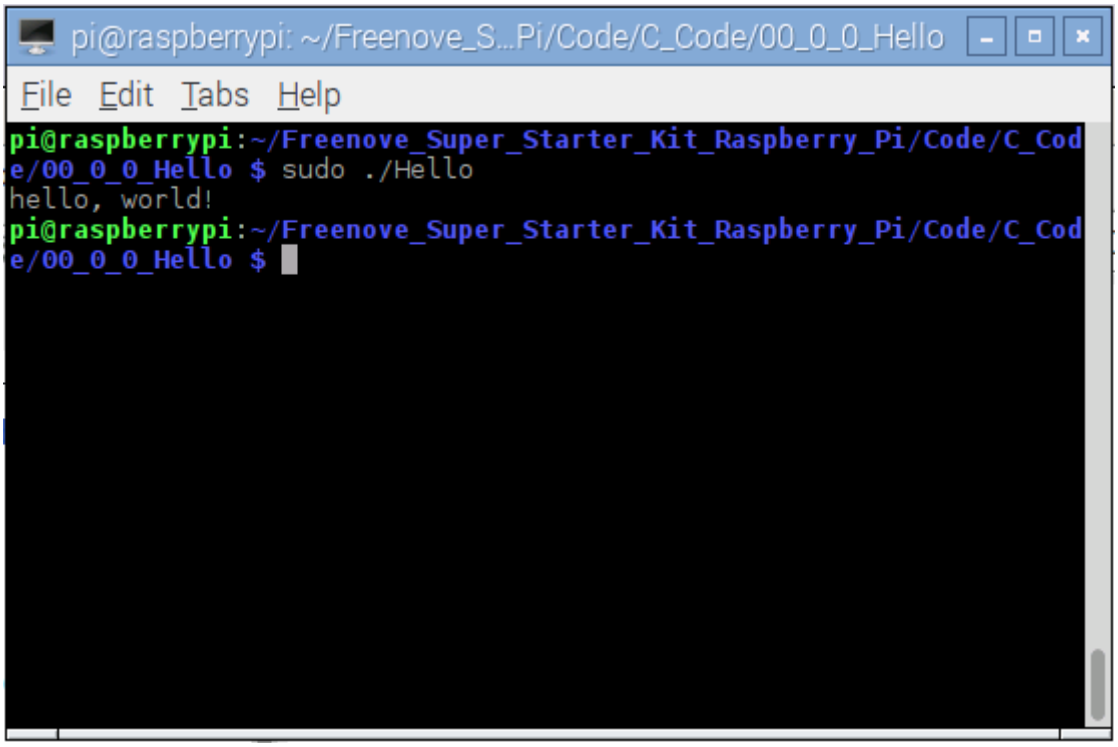
Use the following command to compile the code to generate the executable file “Hello”.

```
gcc Hello.c –o Hello
```

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

```
sudo ./Hello
```

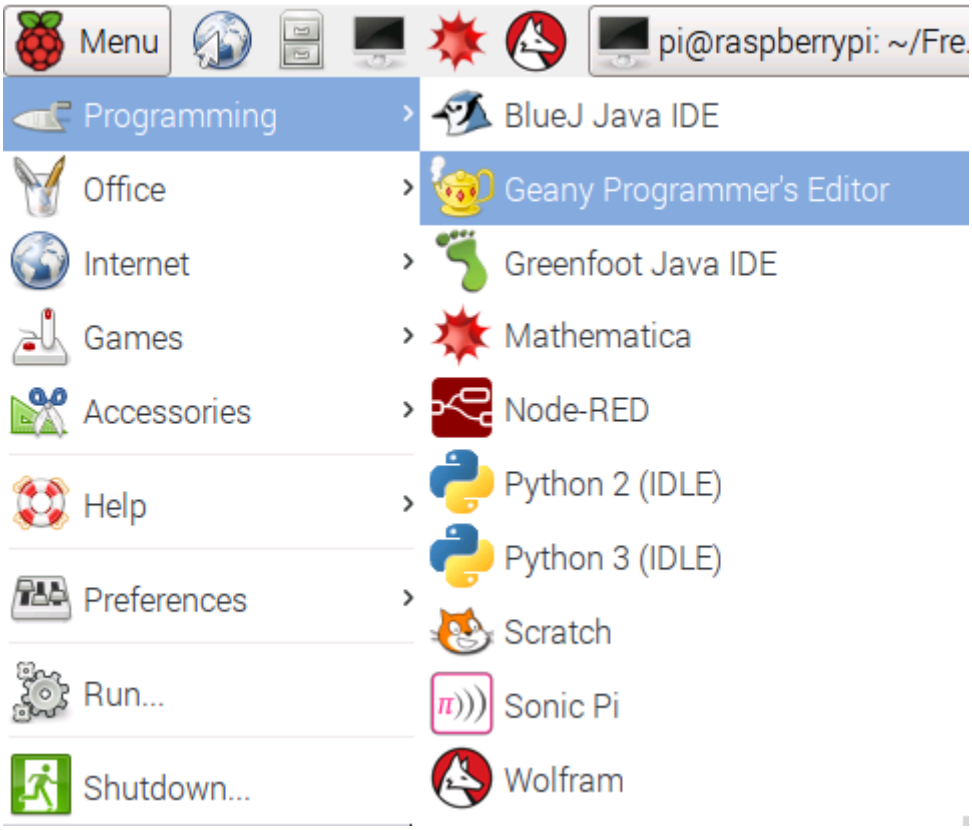
After the execution, "Hello, World!" is printed out in terminal.

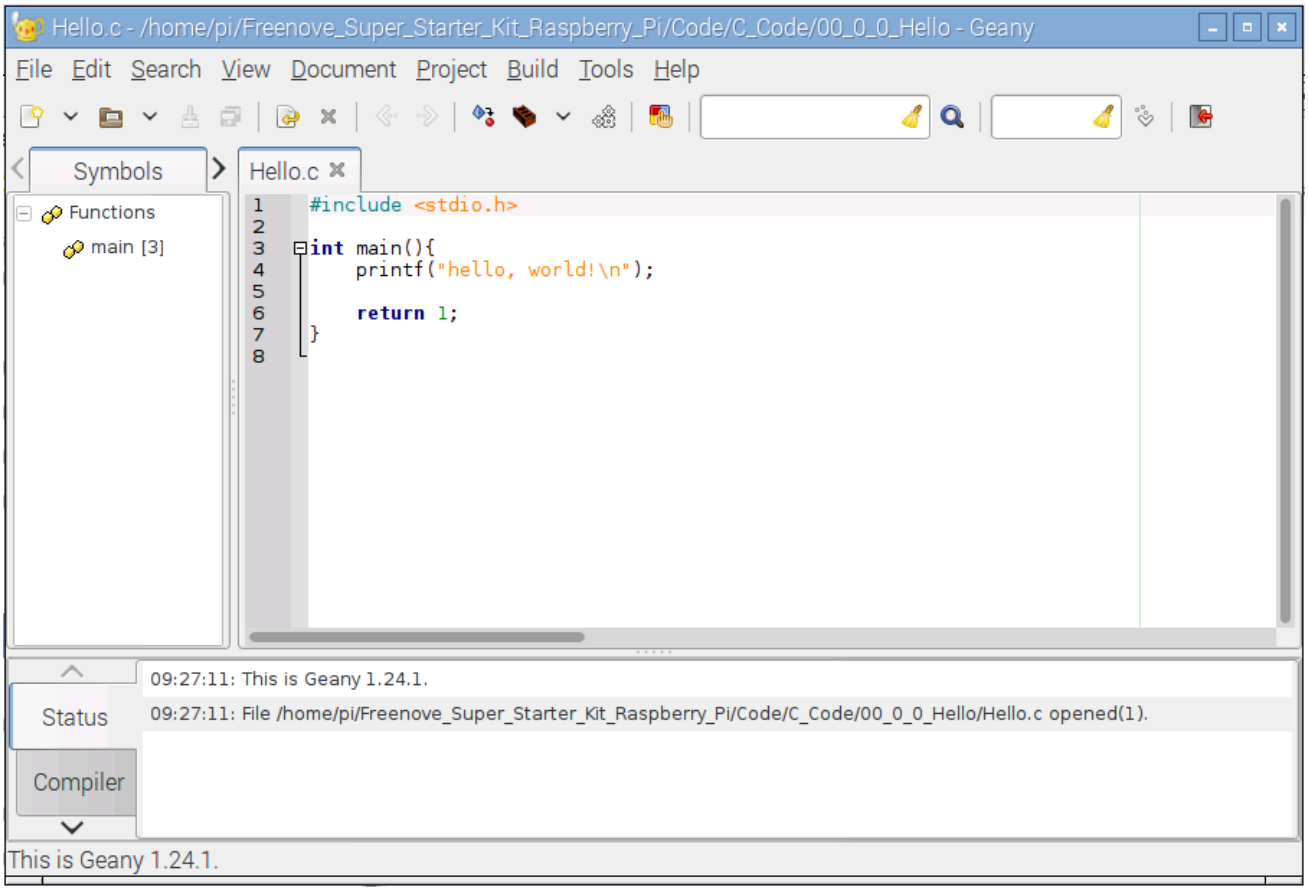


Next, learn to use the Geany editor. Use the following command to open the Geany in the sample file "Hello.c" file directory path.

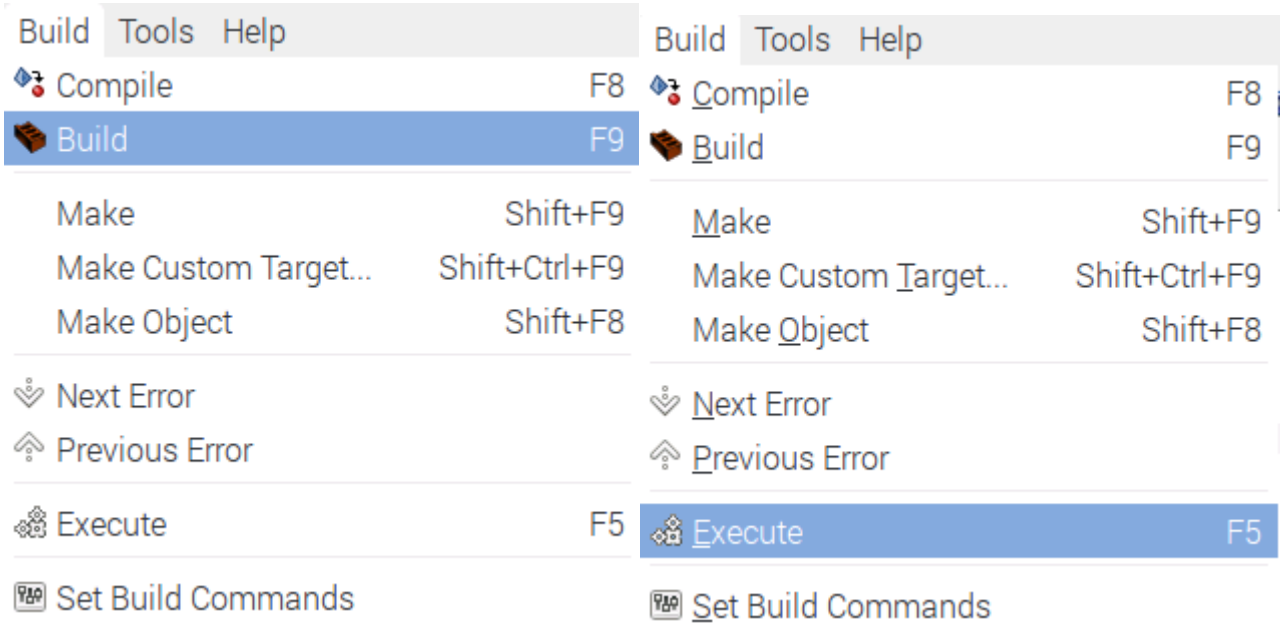
```
geany Hello.c
```

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.

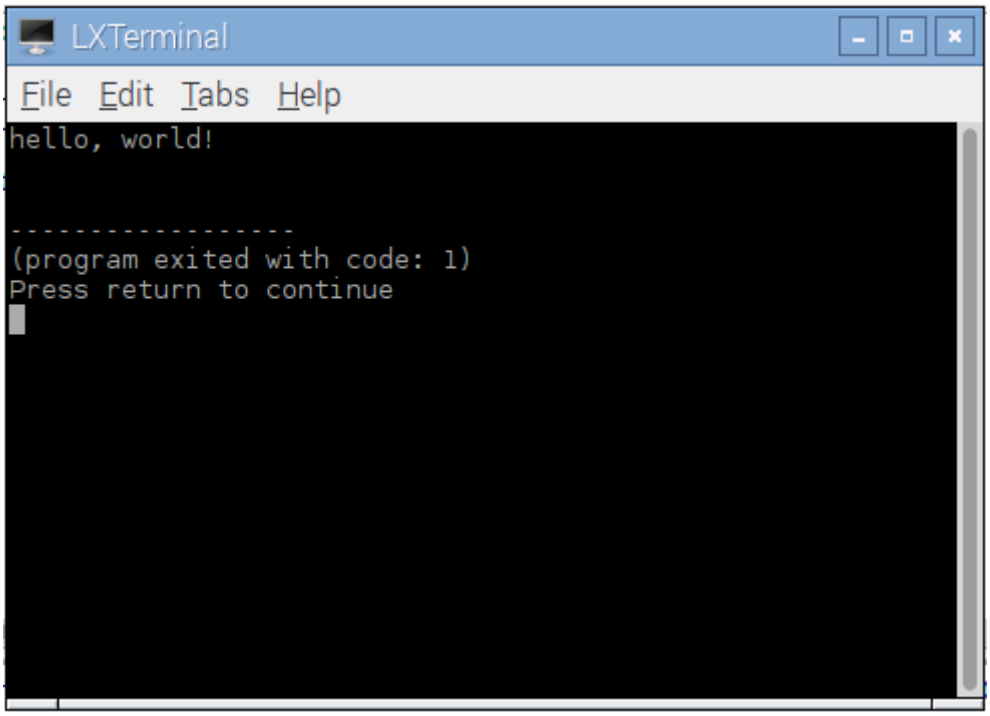




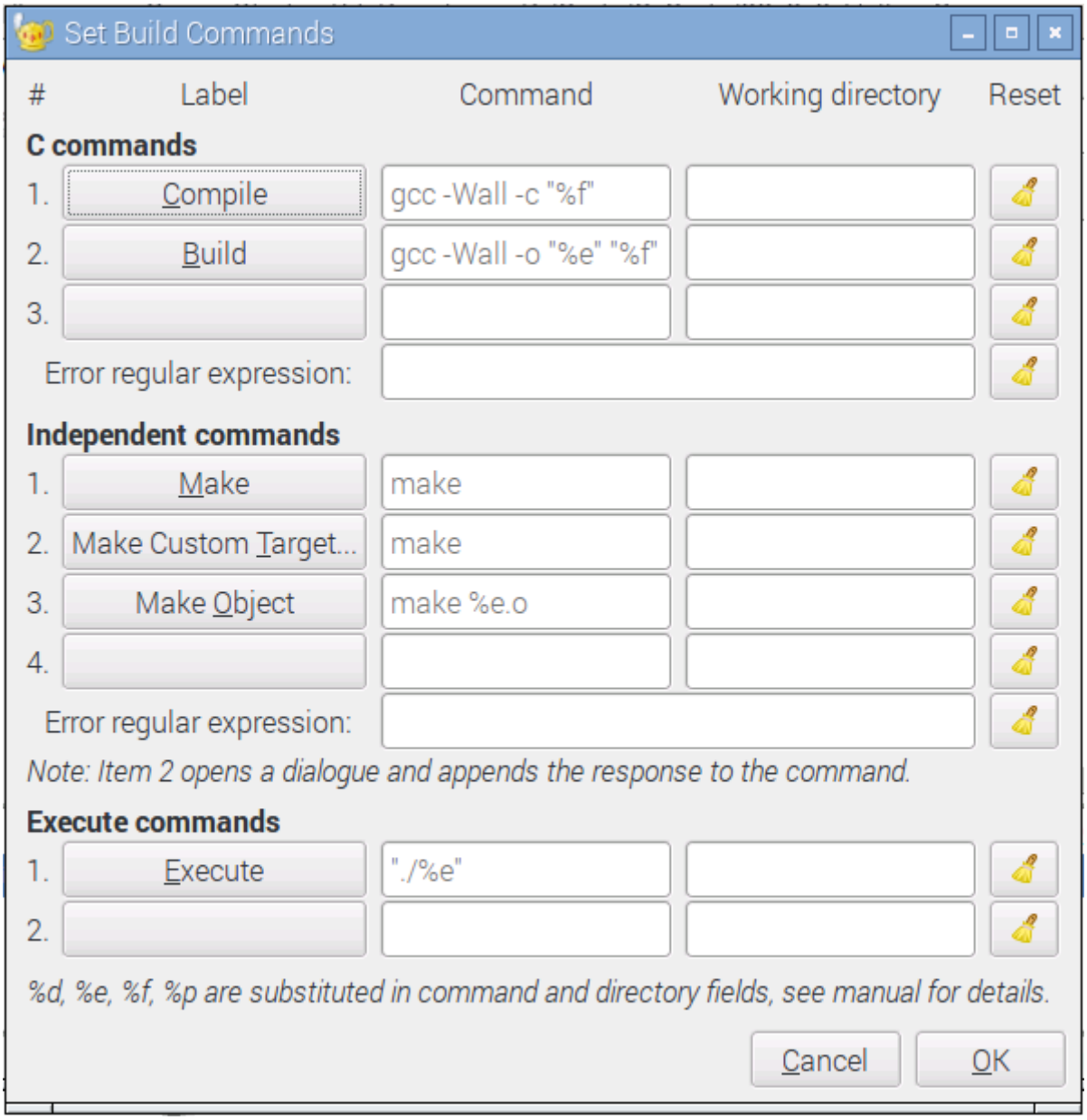
Generates an executable file by clicking menu bar Build->Build, then execute the generated file by clicking menu bar Build->Excute.



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.

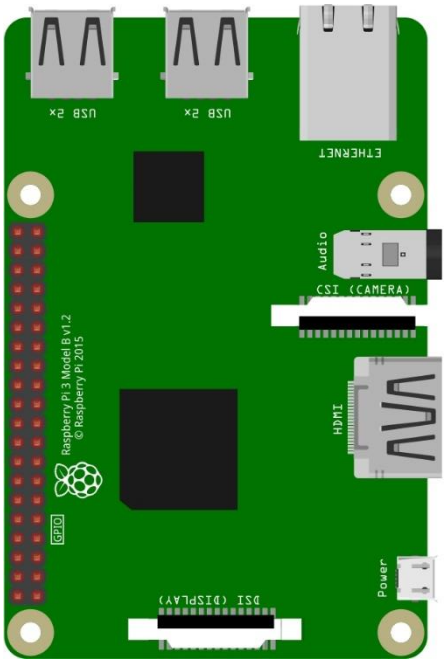
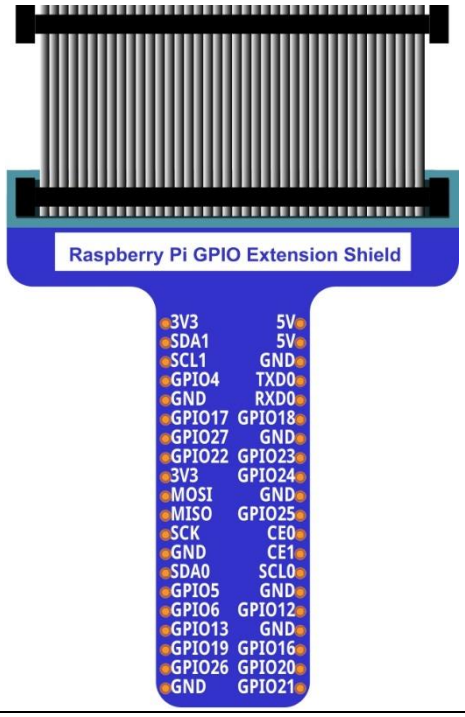
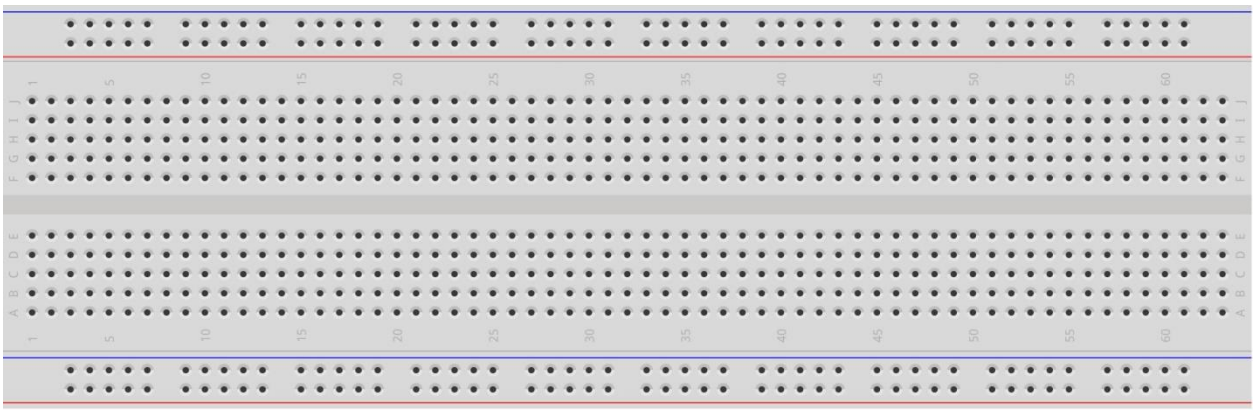
Chapter 1 LED

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
		

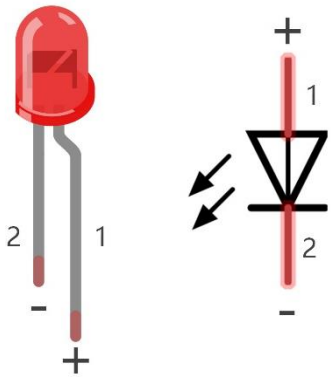
In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each experiment. They will be listed only in text form.

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.



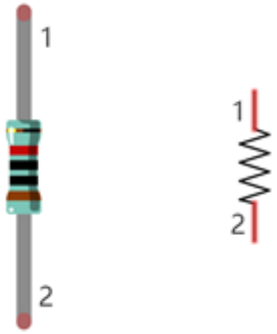
The LED can not be directly connected to power supply, which can damage component. A resistor with certain resisitance must be connected in series in the circuit of LED.

Resistor

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

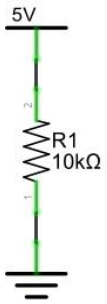
Resistor is an electrical component that limits or regulates the flow of current in an electronic circuit.

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.

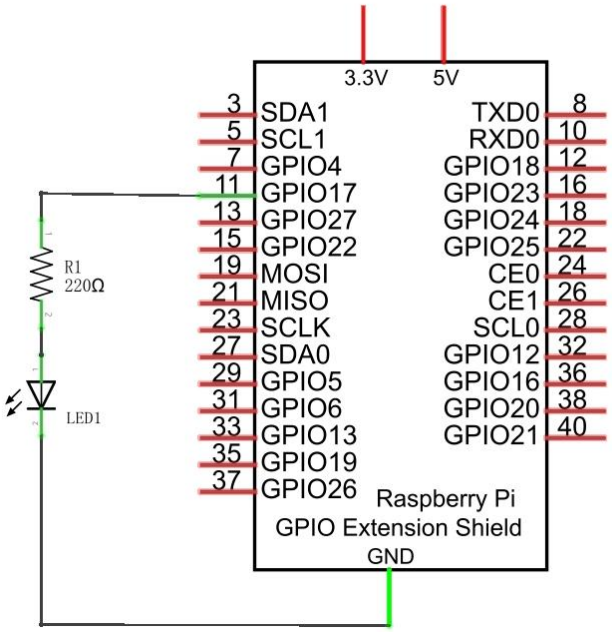


Do not connect the two poles of power supply with low resistance, which will make the current too high to damage electronic components.

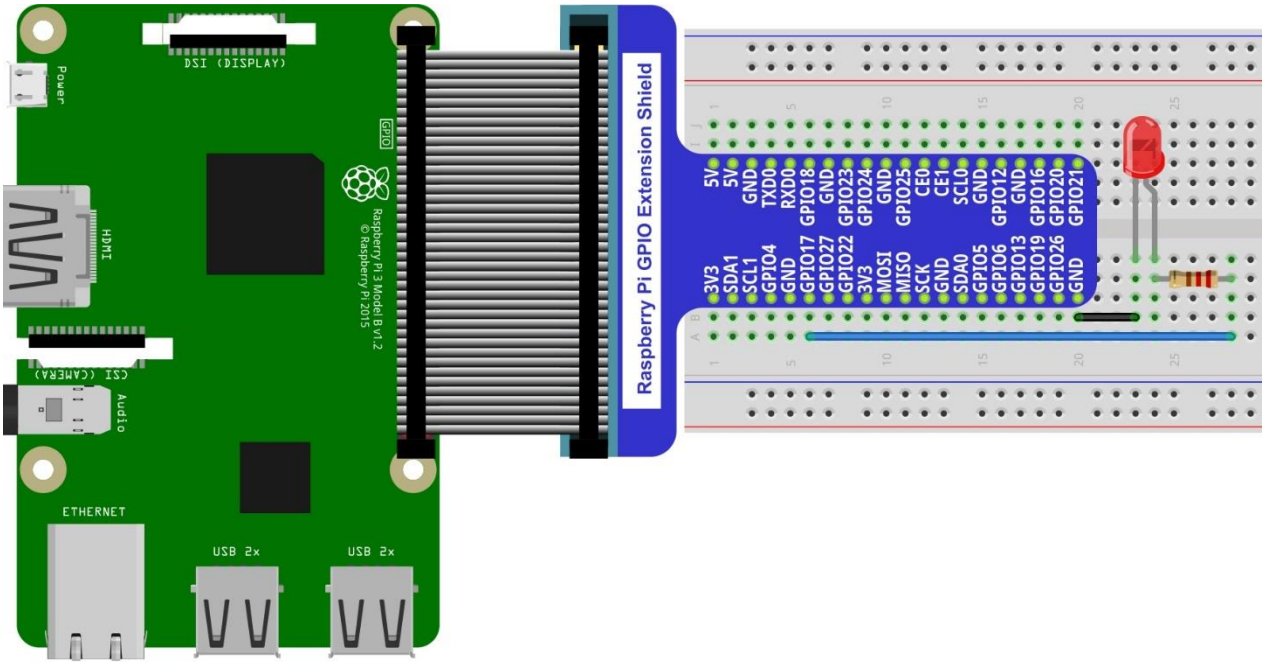
Circuit

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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
```

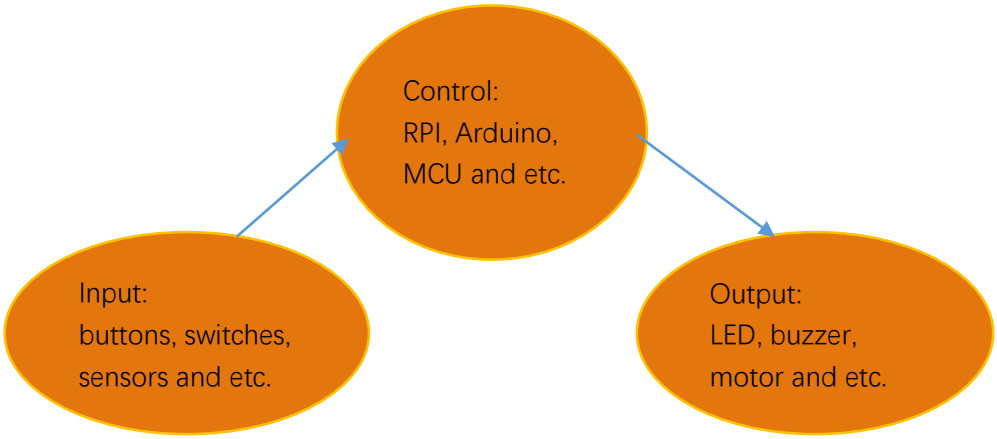
About RPi.GPIO：

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.

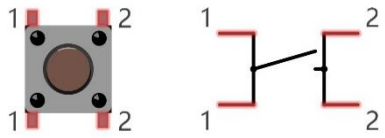
Component List

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

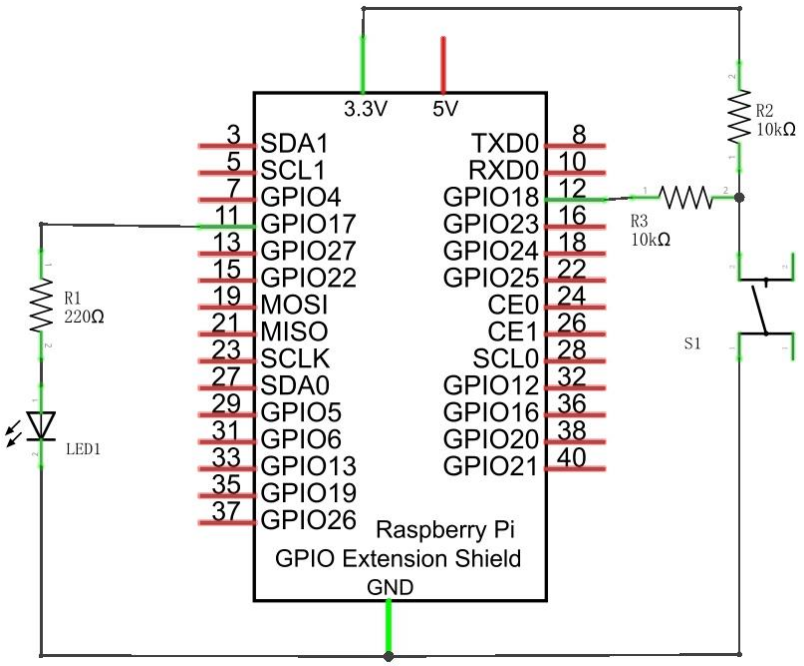
Push button has 4 pins. Two pins on the left is connected, and the right is similar as the left, which is shown in the below:



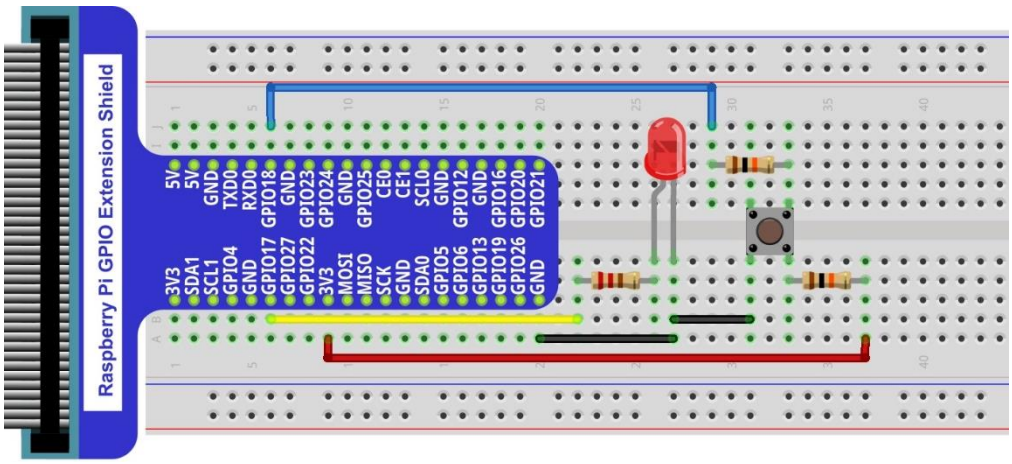
When the push button is pressed, the circuit is turned on.

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_Ultimate_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 01.1.1_btnLED directory of Python code.

```
cd Freenove_Ultimate_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
```

```
    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)

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 ...'

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()
```

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.

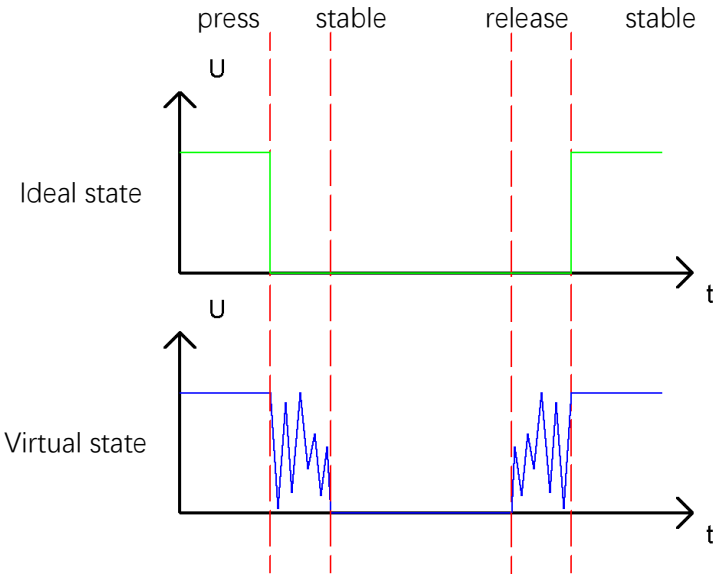
Project 2.2 MINI table lamp

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/02.1.1_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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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):
```

```
15      global ledState
16      print 'buttonEvent GPIO%d'%channel
17      ledState = not ledState
18      if ledState :
19          print 'Turn on LED ... '
20      else :
21          print 'Turn off LED ... '
22      GPIO.output(ledPin,ledState)
23
24  def loop():
25      #Button detect
26      GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent,bouncetime=300)
27      while True:
28          pass
29
30  def destroy():
31      GPIO.output(ledPin, GPIO.LOW)     # led off
32      GPIO.cleanup()                     # Release resource
33
34  if __name__ == '__main__':     # Program start from here
35      setup()
36      try:
37          loop()
38      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
39  will be  executed.
40          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 experiment, we use a number of LED to make a flowing water light.

Component List

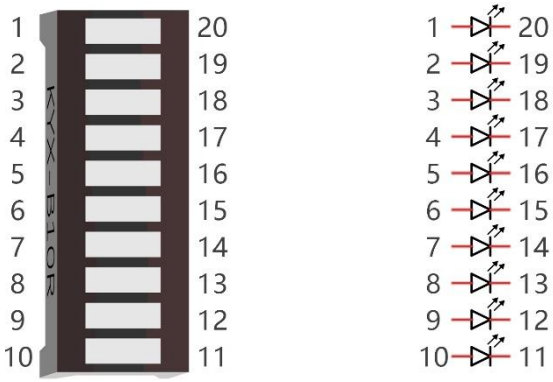
Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

Component knowledge

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

LED bar graph

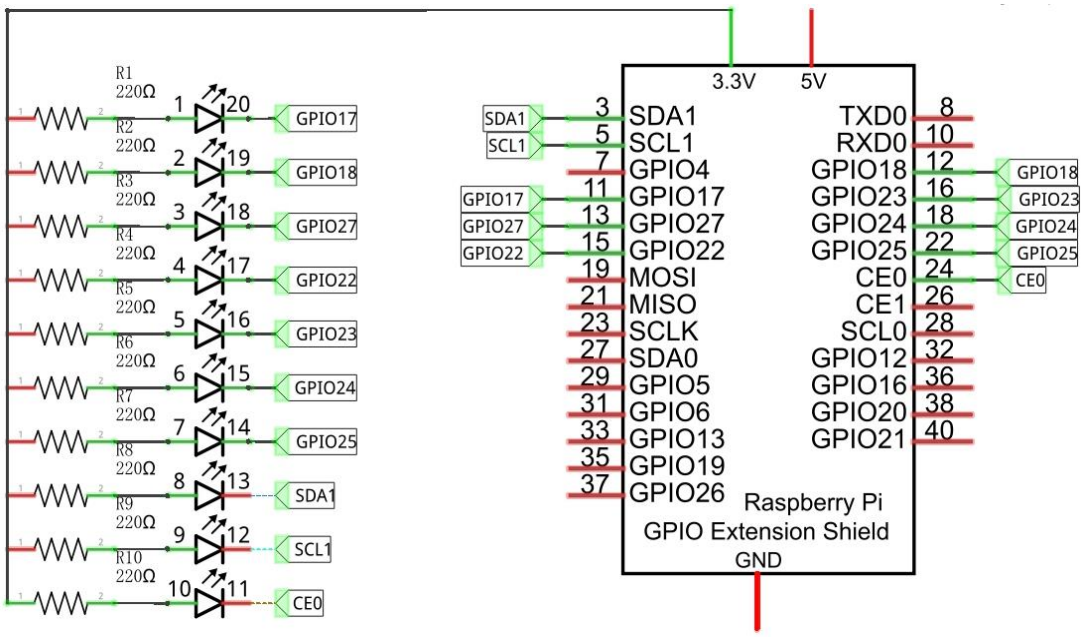
LED bar graph is a component Integration consist of 10 LEDs. There are two rows of pins at its bottom.



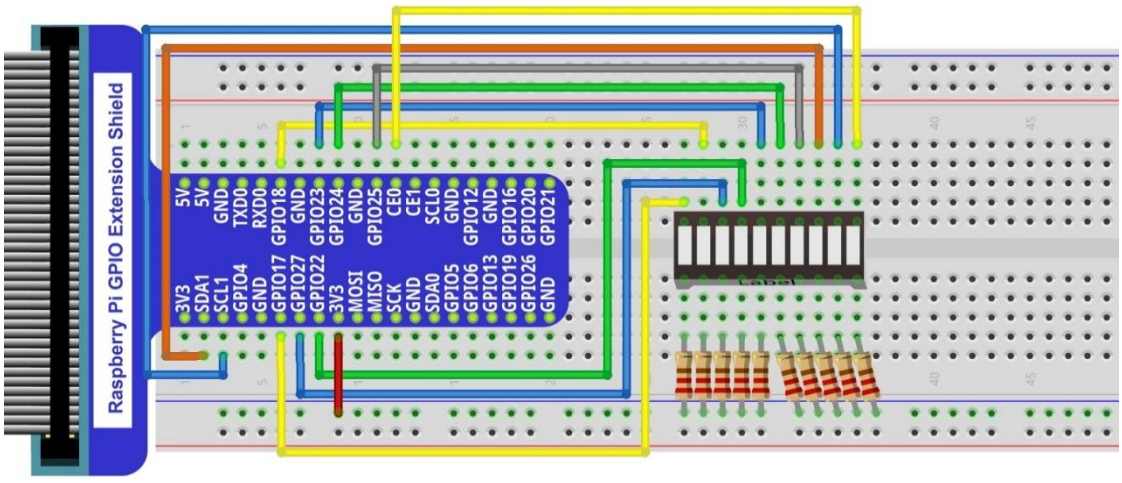
Circuit

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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
```

```
28              led_on(pins[i]);
29              delay(100);
30              led_off(pins[i]);
31          }
32          for(i=leds-1;i>-1;i--){    //make led on from right to left
33              led_on(pins[i]);
34              delay(100);
35              led_off(pins[i]);
36          }
37      }
38      return 0;
39  }
```

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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:

```
1  import RPi.GPIO as GPIO
2  import time
3  ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]
4  def setup():
5      print 'Program is starting...'
6      GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
7      for pin in ledPins:
8          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.

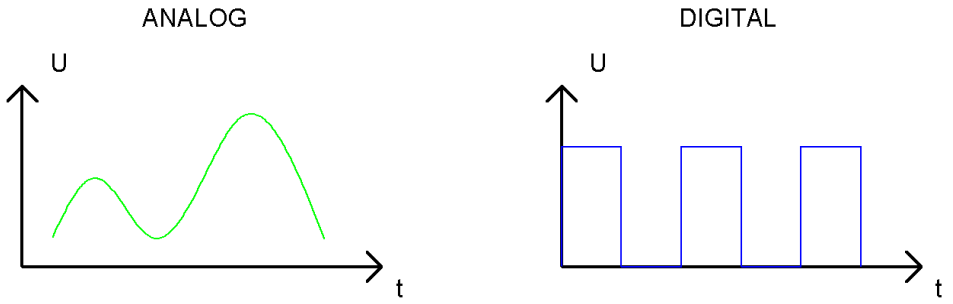
Component List

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

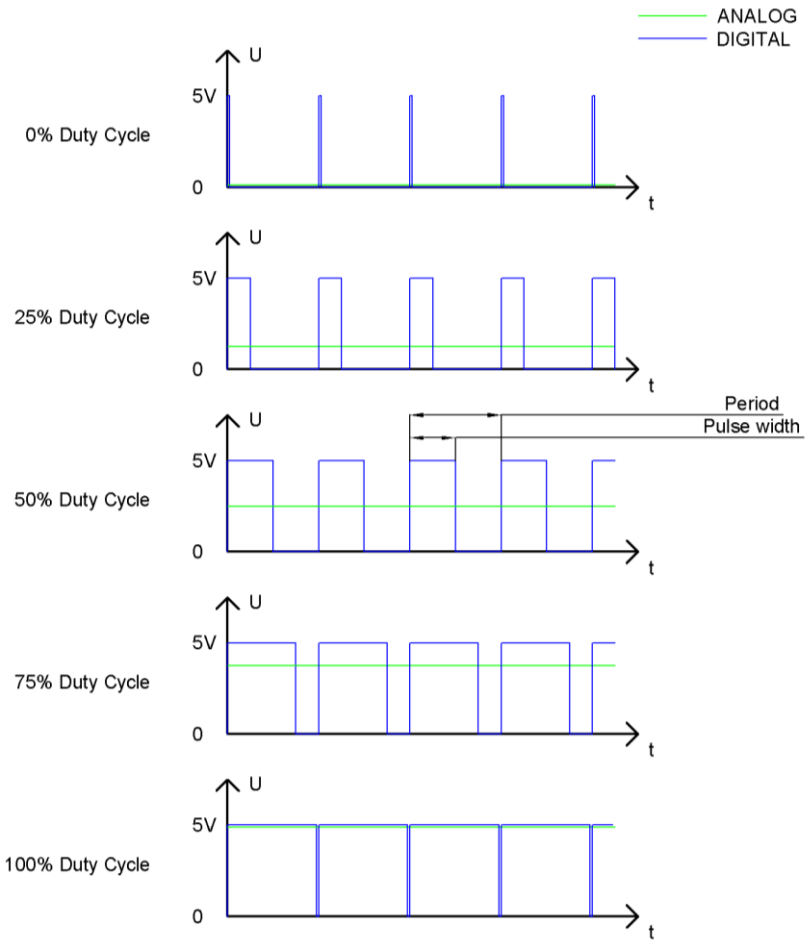
two forms (0 or 1), so it has strong stability. And digital signal and analog signal can be converted to each other.

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%:

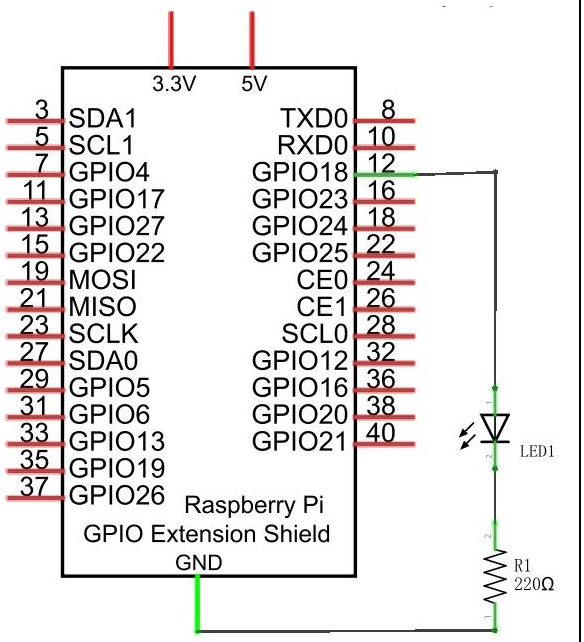
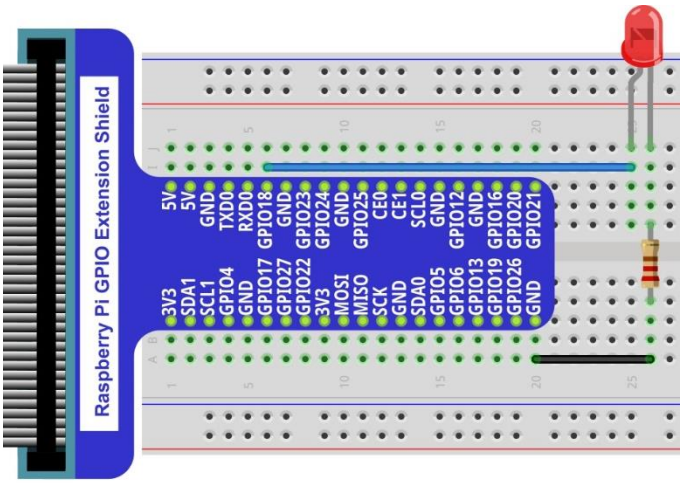


The larger PWM duty cycle is, the lager the output power will be. So we can use PWM to control the brightness of LED, the speed of DC motor and so on.

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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。

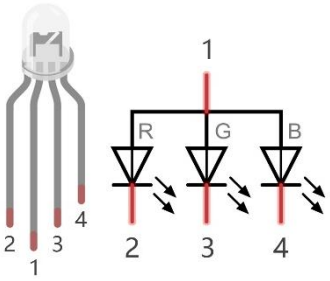
For more details about usage method for PMW of RPi.GPIO, please refer to:

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

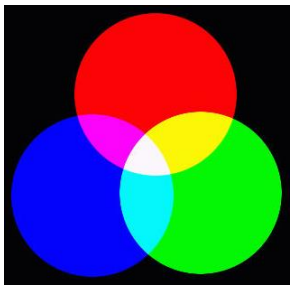
Chapter 5 RGBLED

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

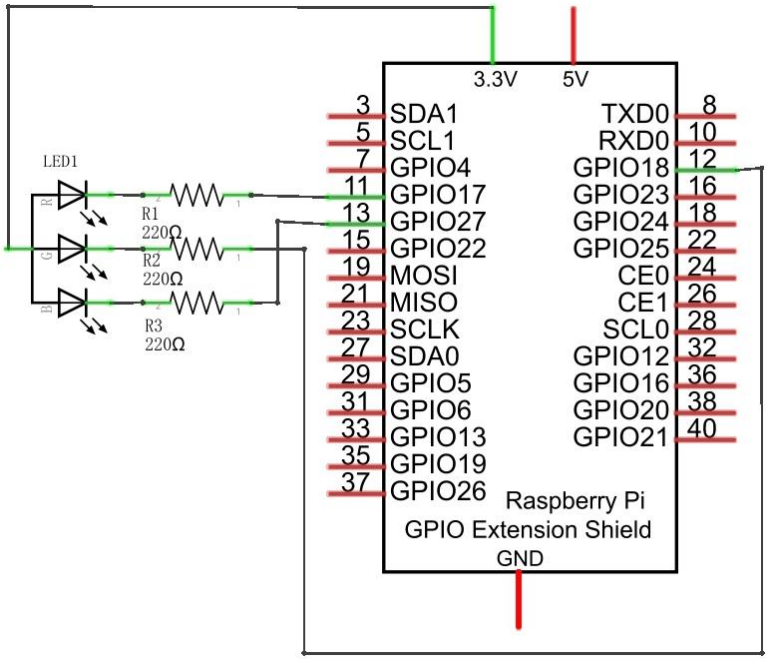
In this experiment, we will make a colorful LED. And we can control RGBLED to switch different colors automaticly.

Component List

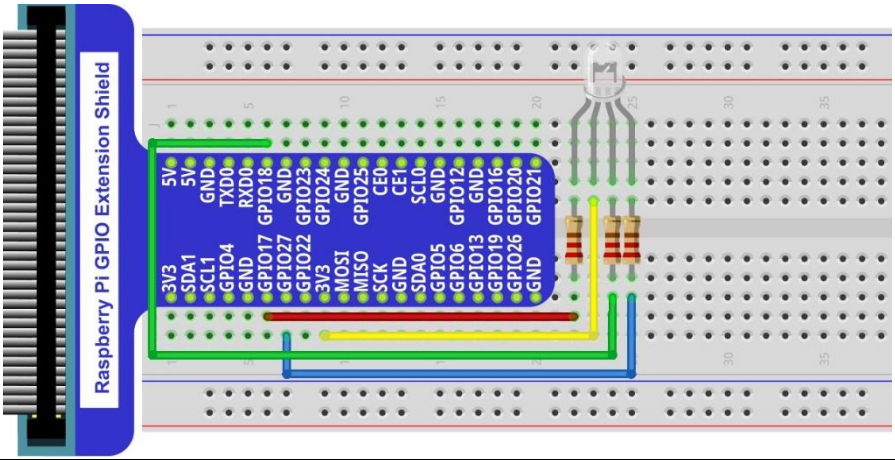
Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	RGBLED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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 experiment, 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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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 experiment, 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).

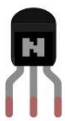
Chapter 6 Buzzer

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

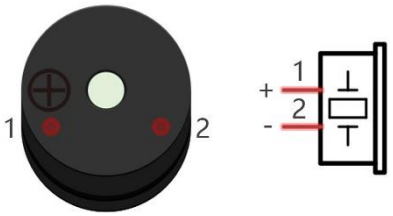
<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x7
  </td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

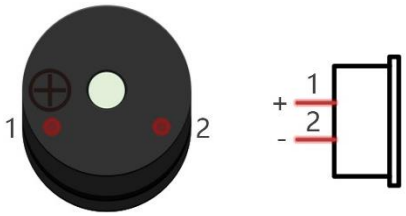
Buzzer

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



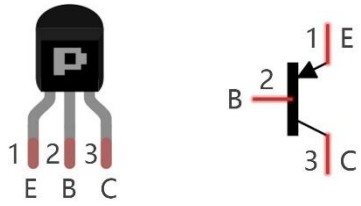
Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer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

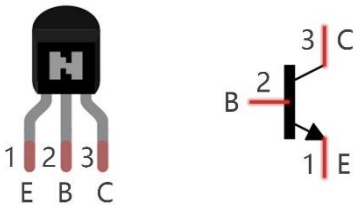
Due to the current operating of buzzer is so large that GPIO of RPi output capability can not be satisfied, a transistor of NPN type is needed here to amplify the current.

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

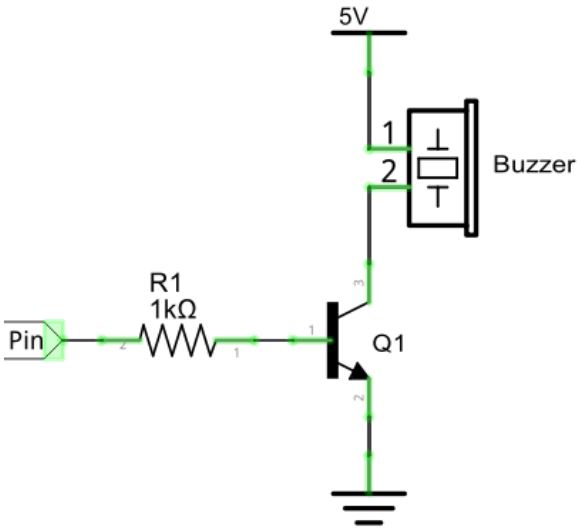
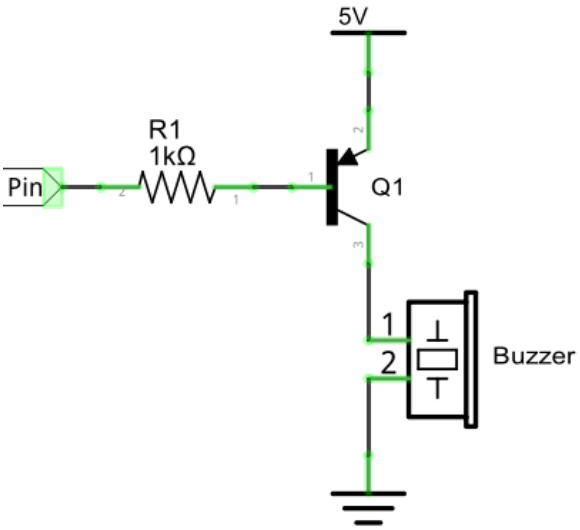


NPN transistor



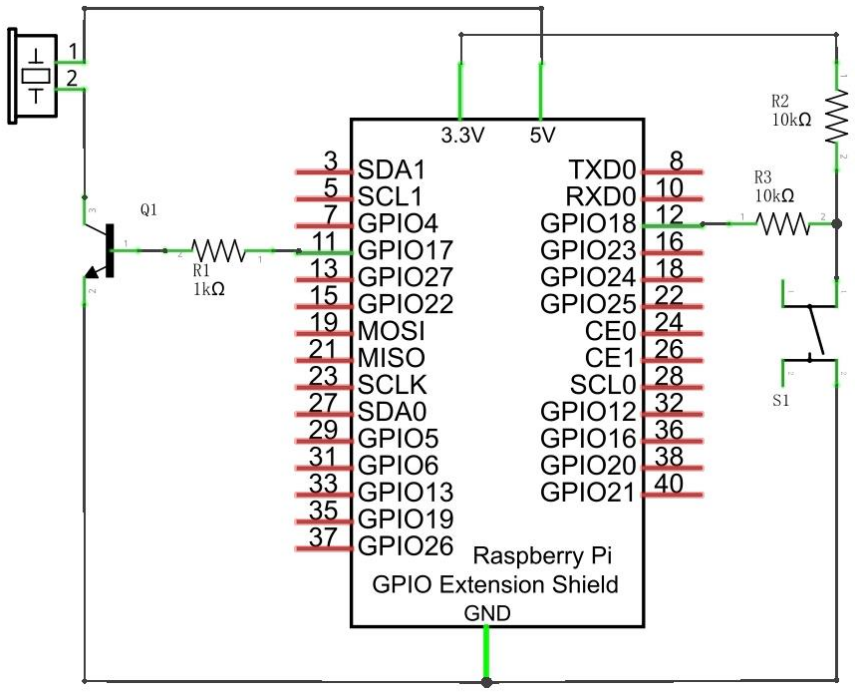
According to the transistor's characteristics, it is often used as a switch in digital circuits. For micro-controller's capacity of output current is very weak, we will use transistor to amplify current and drive large-current components.

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.

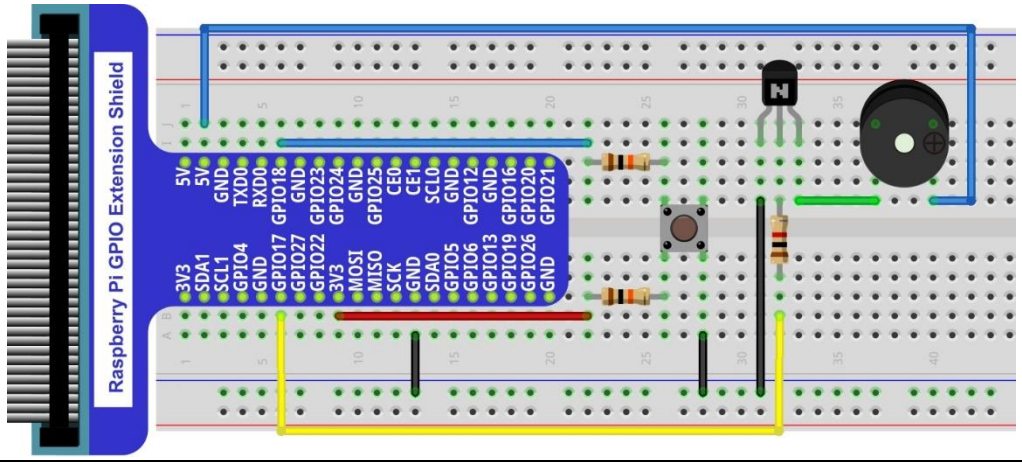
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

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 experiment, 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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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");
        }
```

```
28      }
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 experimental phenomenon, then analyze the code.

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

```
cd Freenove_Ultimate_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 experiment, 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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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 experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_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()
```

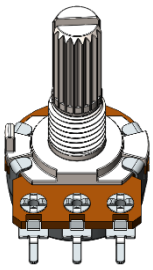
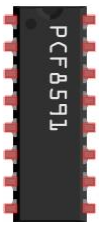
Chapter 7 PCF8591

We have learned how to control the brightness of LED through the output PWM and understood that PWM is not the real analog before. In this chapter, we will learn how to read analog quantities through PCF8591, convert it into digital quantity and convert the digital quantity into analog output. That is, ADC and DAC.

Project 7.1 Read the Voltage of Potentiometer

In this experiment, we will use the ADC function of PCF8591 to read the voltage value of potentiometer. And then output the voltage value through the DAC to control the brightness of LED.

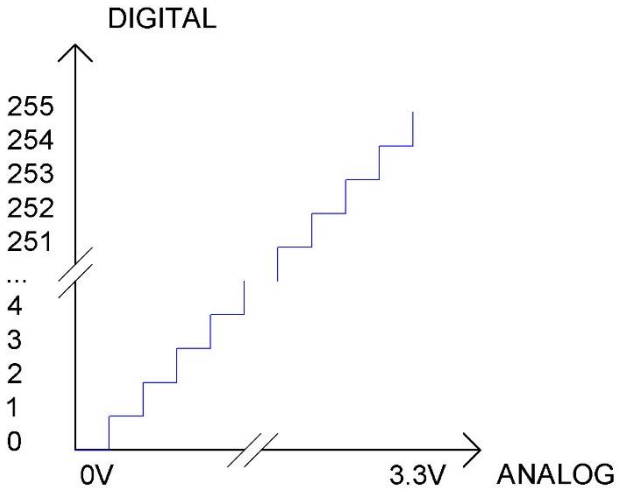
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x16
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>PCF8591 x1
  </td><td>Resistor 10kΩ x2
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

Circuit knowledge

ADC

ADC, Analog-to-Digital Converter, is a device used to convert analog to digital. The range of the ADC on PCF8591 is 8 bits, that means the resolution is 2^8=256, and it represents the range (here is 3.3V) will be divided equally to 256 parts. The analog of each range corresponds to one ADC values. So the more bits ADC has, the denser the partition of analog will be, also the higher precision of the conversion will be.



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

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

...

The following analog will be divided accordingly.

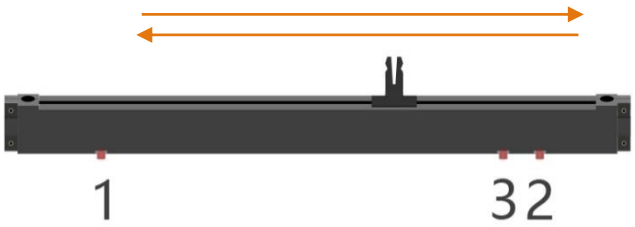
DAC

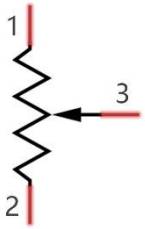
DAC, that is, Digital-to-Analog Converter, is the reverse process of ADC. The digital I/O port can output high level and low level, but can not output an intermediate voltage value, which can be solved by DAC. PCF8591 has a DAC output pin with 8 bit accuracy, which can divide VDD (here is 3.3V) into 2^8=256 parts. For example, when the digital quantity is 1, the output voltage value is 3.3/256 *1 V, and when the digital quantity is 128, the output voltage value is 3.3/256 *128=1.65V, the higher accuracy of PCF8591 is, the higher the accuracy of output voltage value is.

Component knowledge

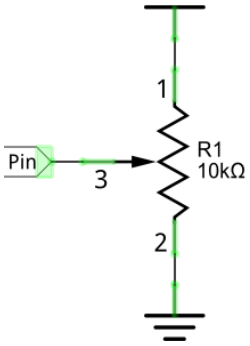
Potentiometer

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



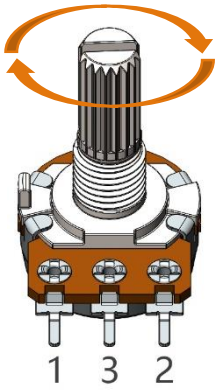


What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pins 1 to pin 2, the resistance between pin 1, and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.
In the circuit. The both sides of resistance body are often connected to the positive and negative electrode of the power. When you slide the brush pin 3, you can get a certain voltage in the range of the power supply.



Rotary potentiometer

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



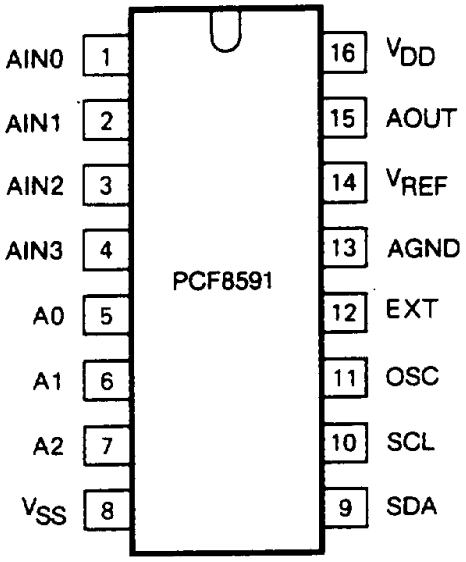
PCF8591

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface.

FEATURES

- Single power supply
- Operating supply voltage 2.5 V to 6 V
- Low standby current
- Serial input/output via I2C-bus
- Address by 3 hardware address pins
- Sampling rate given by I2C-bus speed
- differential inputs
- Auto-incremented channel selection
- Analog voltage range from VSS to VDD
- On-chip track and hold circuit
- 8-bit successive approximation A/D conversion
- Multiplying DAC with one analog output.
- 4 analog inputs programmable as single-ended or

PINNING

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

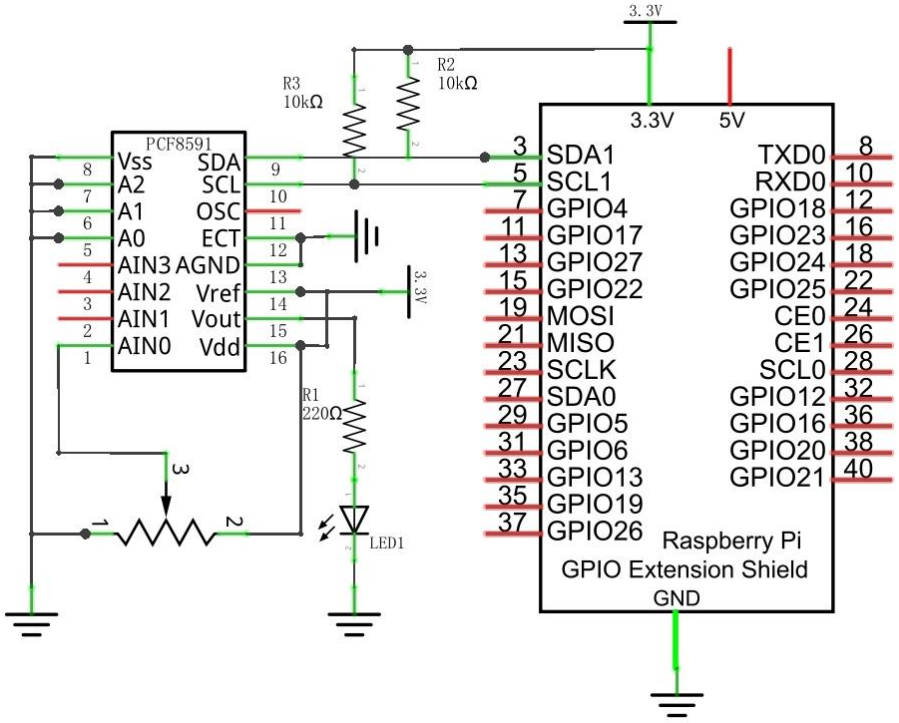
For more details about PCF8591, please refer to datasheet.

I2C communication

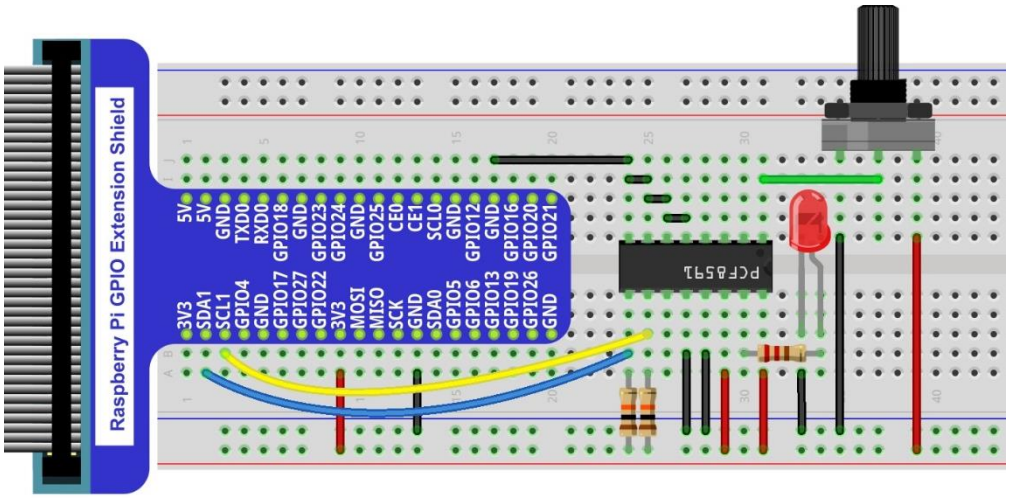
I2C(Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used to connection of micro controller and its peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique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

Circuit

Schematic diagram



Hardware connection



Configure I2C

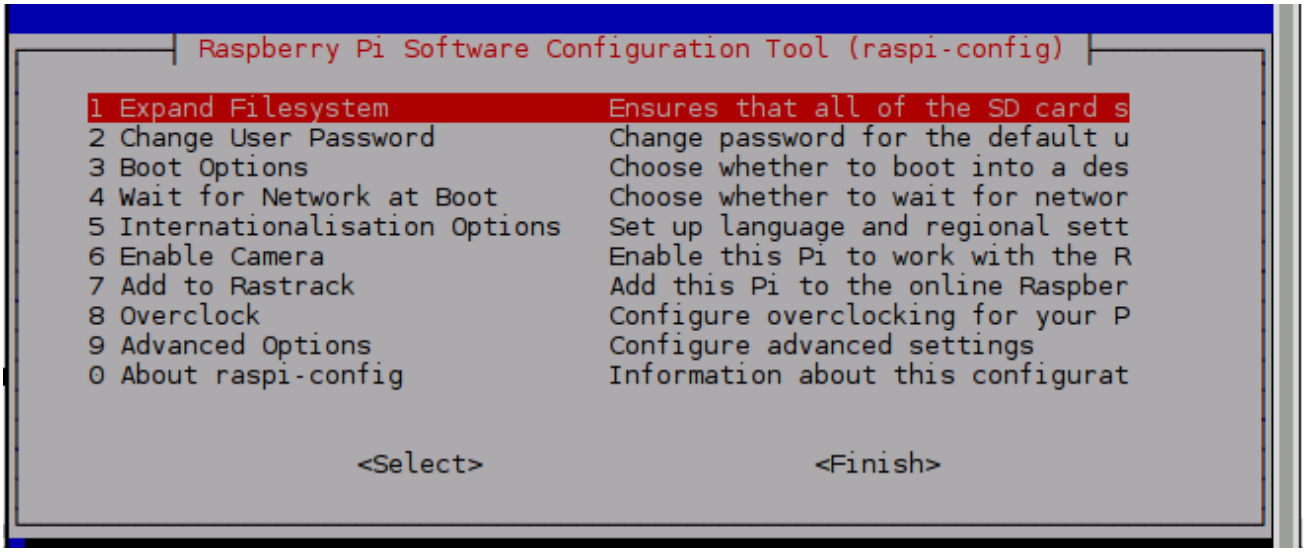
Enable I2C

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

Type command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “9 Advanced Options” “A6 I2C”→“Yes”→“Finish” in order and restart your RPi later. Then the I2C module is started.

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

```
lsmod | grep i2c
```

If the I2C module has been started, the following content will be shown:

Install I2C-Tools

Type the command to install I2C-Tools.

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

I2C device address detection:

```
i2cdetect –y 1
```

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

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

Code

C Code 7.1.1 pcf8591

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 07.1.1_ PCF8591 directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/07.1.1_PCF8591
```

2. Use following command to compile “PCF8591.c” and generate executable file “PCF8591”.

```
gcc PCF8591.c –o PCF8591 –lwiringPi
```

3. Then run the generated file “PCF8591”.

```
sudo ./PCF8591
```

After the program is executed, shift the potentiometer, then the terminal will print out the potentiometer voltage value and the converted digital content. When the voltage is greater than 1.6V (voltage need to turn on red LED), LED starts emitting light. If you continue to increase the output voltage, the LED will become more bright gradually.

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

The following is the code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

int main(void){
    int value;
    float voltage;
    wiringPiSetup();
    pcf8591Setup(pinbase,address);
    while(1){
        value = analogRead(A0);  //read A0 pin
        analogWrite(pinbase+0,value);
        voltage = (float)value / 255.0 * 3.3;  // calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
    }
}
```

The default I2C address of PCF8591 is 0x48. The pinbase is an any value greater than or equal to 64. And we have defined the ADC input channel A1, A2, A0, A3 of PCF8591.

```
#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3
```

In the main function, after PCF8591 is initialized by pcf8591Setup(pinbase, address), you can use the function analogRead() and analogWrite() to operate the ADC and DAC.

```
pcf8591Setup(pinbase,address);
```

In the “while” cycle, analogRead (A0) is used to read the ADC value of the A0 port (connected potentiometer), then the readed ADC value is output through analogWrite(). And then the corresponding actual voltage value will be calculated and displayed.

```
    while(1){
        value = analogRead(A0);  //read A0 pin
        analogWrite(pinbase+0,value);
        voltage = (float)value / 255.0 * 3.3;  // calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
```

```
        delay(100);
    }
```

Details about analogRead() and analogWrite():

`void analogWrite (int pin, int value) ;`
This writes the given value to the supplied analog pin. You will need to register additional analog modules to enable this function for devices.
`int analogRead (int pin) ;`
This returns the value read on the supplied analog input pin. You will need to register additional analog modules to enable this function for devices.

For more detailed instructions about PCF8591 of wiringPi, please refer to:
http://wiringpi.com/extensiones/i2c-pcf8591/

Python Code 7.1.1 pcf8591

First install a smbus module, and the command is as follows:

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

After the installation is completed, operate according to the following steps. Observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 07.1.1_pcf8591 directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/07.1.1_pcf8591
```

2. Use the python command to execute the Python code “pcf8591.py”..

```
python pcf8591.py
```

After the program is executed, shift the potentiometer, then the terminal will print out the potentiometer voltage value and the converted digital content. When the voltage is greater than 1.6V (voltage need to turn on red LED), LED starts emitting light. If you continue to increase the output voltage, the LED will become more bright gradually.

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

The following is the code:

```
1  import smbus
2  import time
3
4  address = 0x48  #default address of PCF8591
5  bus=smbus.SMBus(1)
6  cmd=0x40        #command
7
```

```
8  def analogRead(chn):# read ADC value, chn:0,1,2,3
9      value = bus.read_byte_data(address,cmd+chn)
10     return value
11
12 def analogWrite(value):#write DAC value
13     bus.write_byte_data(address,cmd,value)
14
15 def loop():
16     while True:
17         value = analogRead(0) # read the ADC value of channel 0
18         analogWrite(value)       # write the DAC value
19         voltage = value / 255.0 * 3.3  # calculate the voltage value
20         print 'ADC Value : %d, Voltage : %.2f'%(value,voltage)
21         time.sleep(0.01)
22
23 def destroy():
24     bus.close()
25
26 if __name__ == '__main__':
27     print 'Program is starting ... '
28     try:
29         loop()
30     except KeyboardInterrupt:
31         destroy()
```

First, define the I2C address and control word of PCF8591, and then instantiate object bus of SMBus, which can be used to operate ADC and DAC of PCF8591.

```
address = 0x48  # default address of PCF8591
bus=smbus.SMBus(1)
cmd=0x40        # command
```

This sub function is used to read the ADC. Its parameter “chn” represents the input channel number: 0, 1, 2, 3. Its return value is the readed ADC value.

```
def analogRead(chn):# read ADC value, chn:0,1,2,3
    value = bus.read_byte_data(address,cmd+chn)
    return value
```

This sub function is used to write DAC. Its parameter “value” represents the digital quality to be written, between 0-255.

```
def analogWrite(value):# write DAC value
    bus.write_byte_data(address,cmd,value)
```

In the “while” cycle, first read the ADC value of channel 0, and then wite the value as the DAC digital quality and output corresponding voltage in the Aout pin of PCF8591. Then calculate the corresponding voltage value and print it out.

```
def loop():
    while True:
        value = analogRead(0)    #read the ADC value of channel 0
```

```
        analogWrite(value)       # write ADC value
        voltage = value / 255.0 * 3.3  # calculate voltage value
        print 'ADC Value : %d, Voltage : %.2f'%(value,voltage)
        time.sleep(0.01)
```

About smbus module:

smbus Module

That is System Management Bus。This module defines an object type that allows SMBus transactions on hosts running the Linux kernel.The host kernel must have I2C support, I2C device interface support, and a bus adapter driver. All of these can be either built-in to the kernel, or loaded from modules.

In Python, you can use help(smbus) to view the relevant function and their descriptions.

bus=smbus.SMBus(1)：Create an SMBus class object.

bus.read_byte_data(address,cmd+chn)： Read a byte of data from an address and return it.

bus.write_byte_data(address,cmd,value)： Write a byte of data to an address.

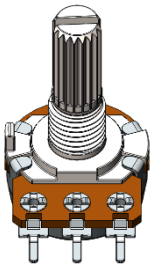
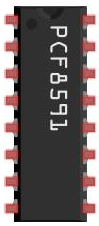
Chapter 8 Potentiometer & LED

We have learned how to use ADC and DAC before. When using DAC output analog to drive LED, we found that, when the output voltage is less than led turn-on voltage, the LED does not light, the output analog voltage is greater than the LED voltage, the LED will light. This leads to a certain degree of waste of resources. Therefore, in the control of LED brightness, we should choose a more reasonable way of PWM control. In this chapter, we learn to control the brightness of LED through a potentiometer.

Project 8.1 Soft Light

In this experiment, we will make a soft light. Use PCF8591 to read ADC value of potentiometers and map it to duty cycle ratio of PWM used to control the brightness of LED. Then you can make the LED brightness changed by shifting the potentiometer.

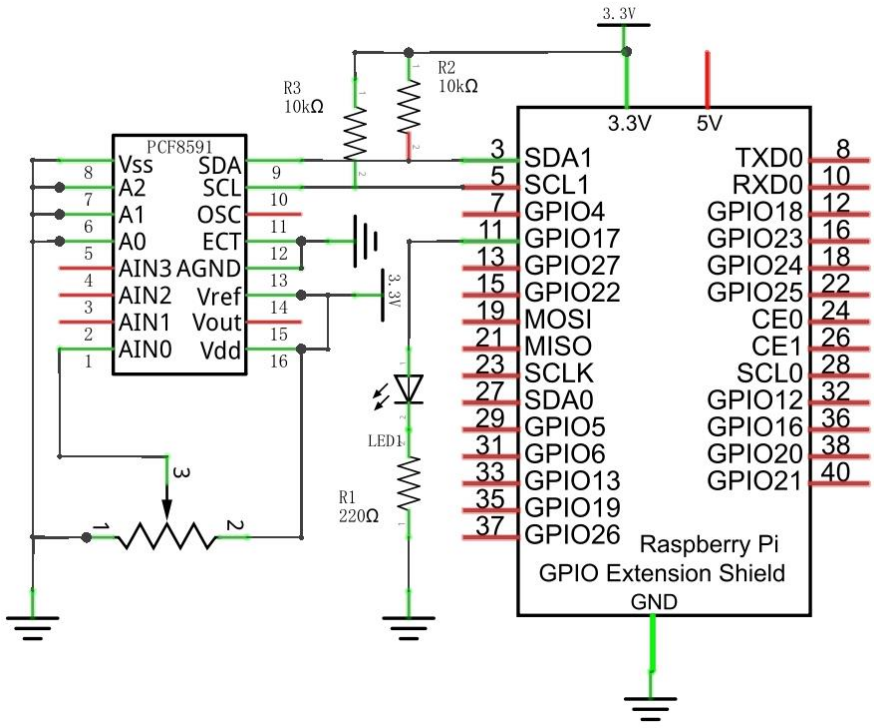
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x17
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>PCF8591 x1
  </td><td>Resistor 10kΩ x2
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

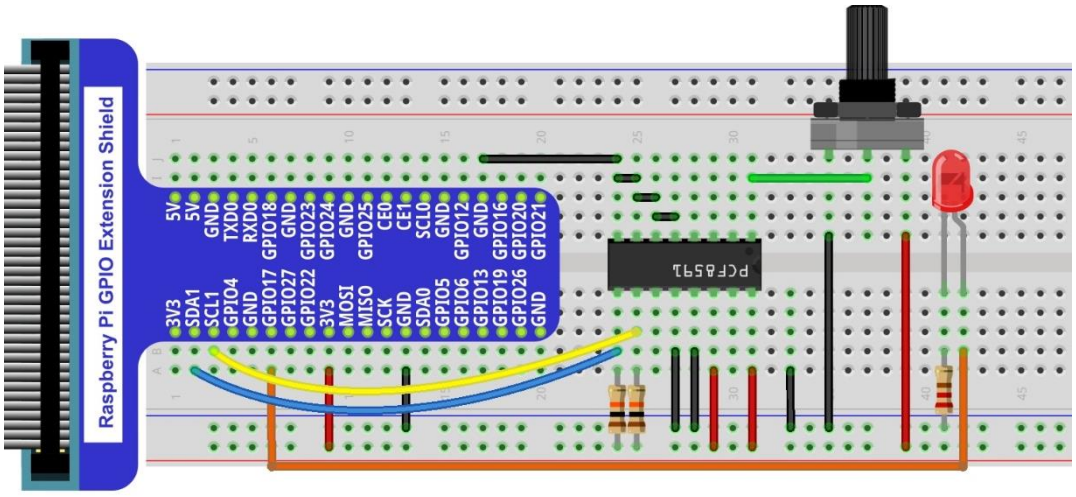
Circuit

The circuit of this experiment is similar to the one in the last chapter. The only difference is that the pin used to control LED is different.

Schematic diagram



Hardware connection



Code

C Code 8.1.1 Softlight

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 08.2.1_Softlight directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/08.1.1_Softlight
```

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

```
gcc Softlight.c –o Softlight–lwiringPi –lpthread
```

3. Then run the generated file "Softlight".

```
sudo ./Softlight
```

After the program is executed, shift the potentiometer, then the terminal window will print out the voltage value of the potentiometer and the converted digital quantity. And brightness of LED will be changed consequently.

The following is the code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define ledPin 0
int main(void){
    int value;
    float voltage;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledPin,0,100);
    pcf8591Setup(pinbase,address);

    while(1){
        value = analogRead(A0);  //read A0 pin
        softPwmWrite(ledPin,value*100/255);
        voltage = (float)value / 255.0 * 3.3;  // calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
```

```
30      }
31      return 0;
32  }
```

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

Python Code 8.1.1 Softlight

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 08.2.1_Softlight directory of Python code

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/08.1.1_Softlight
```

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

```
python Softlight.py
```

After the program is executed, shift the potentiometer, then the terminal window will print out the voltage value of the potentiometer and the converted digital quantity. And brightness of LED will be changed consequently.

The following is the code:

```
1   import RPi.GPIO as GPIO
2   import smbus
3   import time
4
5   address = 0x48
6   bus=smbus.SMBus(1)
7   cmd=0x40
8   ledPin = 11
9
10  def analogRead(chn):
11      value = bus.read_byte_data(address,cmd+chn)
12      return value
13
14  def analogWrite(value):
15      bus.write_byte_data(address,cmd,value)
16
17  def setup():
18      global p
19      GPIO.setmode(GPIO.BOARD)
20      GPIO.setup(ledPin,GPIO.OUT)
21      GPIO.output(ledPin,GPIO.LOW)
22
23      p = GPIO.PWM(ledPin,1000)
24      p.start(0)
25
26  def loop():
27      while True:
28          value = analogRead(0)
29          p.ChangeDutyCycle(value*100/255)
30          voltage = value / 255.0 * 3.3
```

```
31          print 'ADC Value : %d, Voltage : %.2f'%(value,voltage)
32          time.sleep(0.01)
33
34  def destroy():
35      bus.close()
36      GPIO.cleanup()
37
38  if __name__ == '__main__':
39      print 'Program is starting ... '
40      setup()
41      try:
42          loop()
43      except KeyboardInterrupt:
44          destroy()
```

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

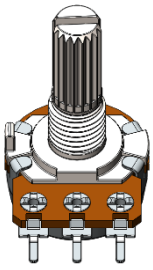
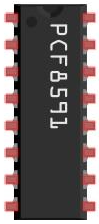
Chapter 9 Potentiometer & RGBLED

In this chapter, we will use 3 potentiometers to control the brightness of 3 LEDs of RGBLED to make it show different colors.

Project 9.1 Colorful Light

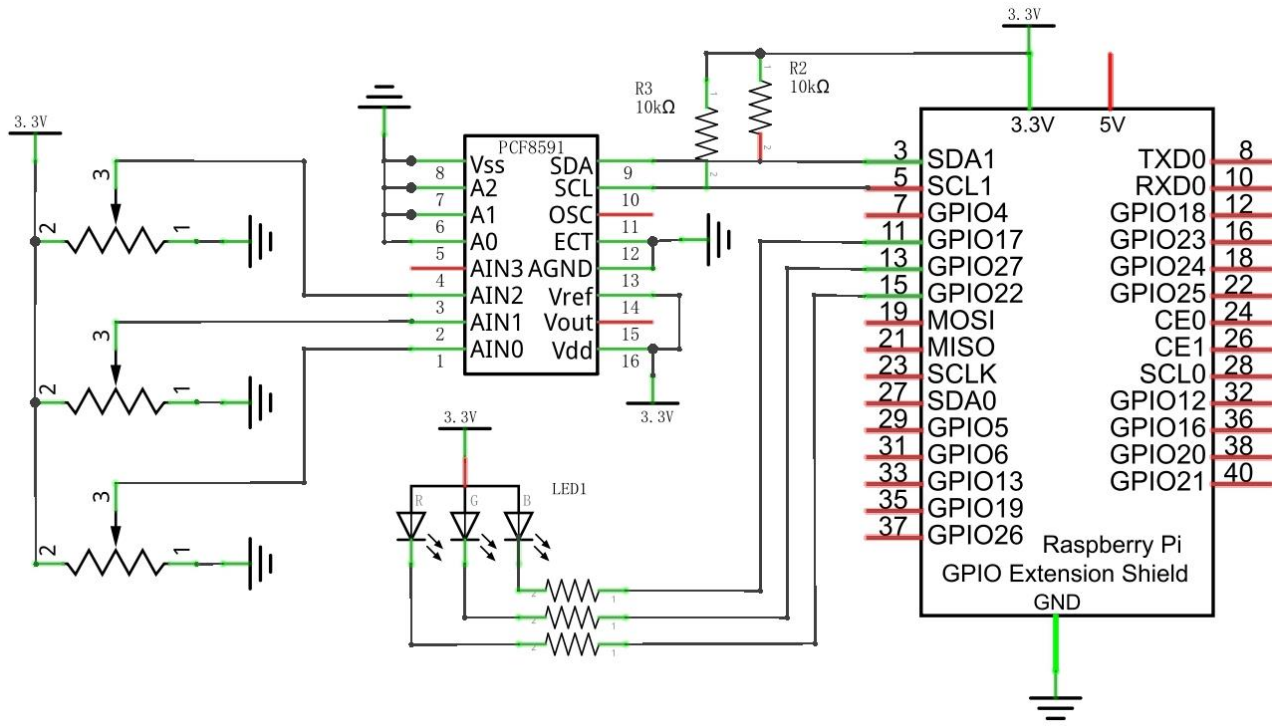
In this experiment, 3 potentiometers are used to control RGBLED and the principle is the same with the front soft light. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the front one need only one LED, but this experiment needs a RGBLED（3 LEDs）.

Component List

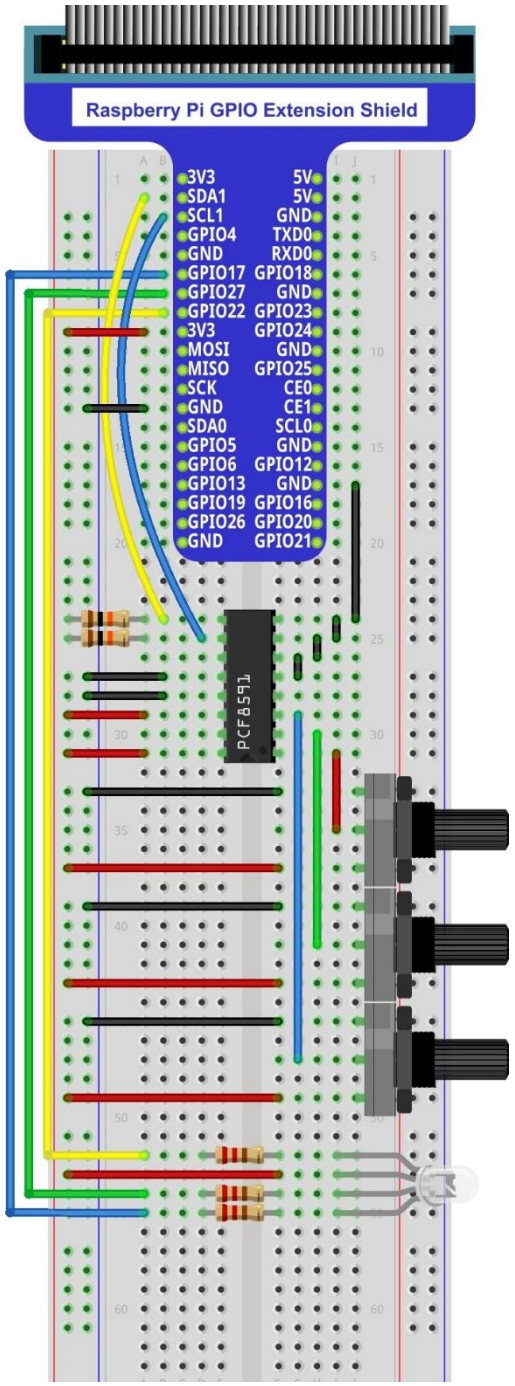
<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Expansion Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x25
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td>PCF8591 x1
  </td><td>Resistor 10kΩ x2
  </td><td>Resistor 220Ω x3
  </td><td>RGBLED x1
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection



Code

C Code 9.1.1 Colorful Softlight

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/09.1.1_ColorfulSoftlight
```

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

```
gcc ColorfulSoftlight.c –o ColorfulSoftlight –lwiringPi –lpthread
```

3. Then run the generated file "ColorfulSoftlight".

```
sudo ./ColorfulSoftlight
```

After the program is executed, rotate one of potentiometers, then the color of RGBLED will change consequently. And the terminal window will print out the ADC value of each potentiometer.

```
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 238
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 206
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 174
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 152
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 139
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
```

The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define ledRedPin 3         //define 3 pins of RGBLED
#define ledGreenPin 2
#define ledBluePin 0
int main(void){
    int val_Red,val_Green,val_Blue;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledRedPin,0,100);     //creat 3 PMW output pins for RGBLED
    softPwmCreate(ledGreenPin,0,100);
```

```
24      softPwmCreate(ledBluePin,0,100);
25      pcf8591Setup(pinbase,address);       //initialize PCF8591
26
27      while(1){
28          val_Red = analogRead(A0);  //read 3 potentiometers
29          val_Green = analogRead(A1);
30          val_Blue = analogRead(A2);
31          softPwmWrite(ledRedPin,val_Red*100/255);    //map the read value of
32  potentiometers into PWM value and output it
33          softPwmWrite(ledGreenPin,val_Green*100/255);
34          softPwmWrite(ledBluePin,val_Blue*100/255);
35          //print out the read ADC value
36          printf("ADC value val_Red: %d  ,\tval_Green: %d  ,\tval_Blue: %d
37  \n",val_Red,val_Green,val_Blue);
38          delay(100);
39      }
40      return 0;
41  }
```

In the code, read the ADC value of 3 potentiometers and map it into PWM duty cycle to control the control 3 LEDs with different color of RGBLED, respectively.

Python Code 9.1.1 ColorfulSoftlight

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/09.1.1_ ColorfulSoftlight
```

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

```
python ColorfulSoftlight.py
```

After the program is executed, rotate one of potentiometers, then the color of RGBLED will change consequently. And the terminal window will print out the ADC value of each potentiometer.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import smbus
3  import time
4
5  address = 0x48
6  bus=smbus.SMBus(1)
7  cmd=0x40
8
9  ledRedPin = 15      #define 3 pins of RGBLED
10 ledGreenPin = 13
11 ledBluePin = 11
12
13 def analogRead(chn):    #read ADC value
14     bus.write_byte(address,cmd+chn)
15     value = bus.read_byte(address)
16     value = bus.read_byte(address)
17     return value
18
19 def analogWrite(value):
20     bus.write_byte_data(address,cmd,value)
21
22 def setup():
23     global p_Red,p_Green,p_Blue
24     GPIO.setmode(GPIO.BOARD)
25     GPIO.setup(ledRedPin,GPIO.OUT)       #set 3 pins of RGBLED to output mode
26     GPIO.setup(ledGreenPin,GPIO.OUT)
27     GPIO.setup(ledBluePin,GPIO.OUT)
28
29     p_Red = GPIO.PWM(ledRedPin,1000)     #configure PMW to 3 pins of RGBLED
30     p_Red.start(0)
31     p_Green = GPIO.PWM(ledGreenPin,1000)
32     p_Green.start(0)
33     p_Blue = GPIO.PWM(ledBluePin,1000)
34     p_Blue.start(0)
35
```

```
def loop():
    while True:
        value_Red = analogRead(0)        #read ADC value of 3 potentiometers
        value_Green = analogRead(1)
        value_Blue = analogRead(2)
        p_Red.ChangeDutyCycle(value_Red*100/255)  #map the read value of potentiometers
into PWM value and output it
        p_Green.ChangeDutyCycle(value_Green*100/255)
        p_Blue.ChangeDutyCycle(value_Blue*100/255)
        #print read ADC value
        print 'ADC Value
value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value_Blue)
        time.sleep(0.01)

def destroy():
    bus.close()
    GPIO.cleanup()

if __name__ == '__main__':
    print 'Program is starting ... '
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

In the code, read the ADC value of 3 potentiometers and map it into PWM duty cycle to control the control 3 LEDs with different color of RGBLED, respectively.

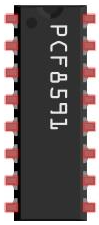
Chapter 10 Photoresistor & LED

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

Project 10.1 NightLamp

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

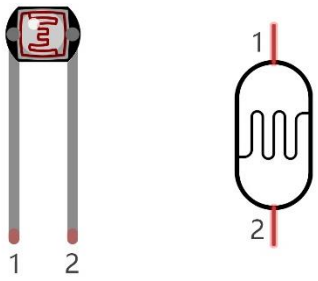
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x15
  </td></tr>
<tr><td>Photoresistor x1
  </td><td>PCF8591 x1
  </td><td>Resistor 10kΩ x3
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

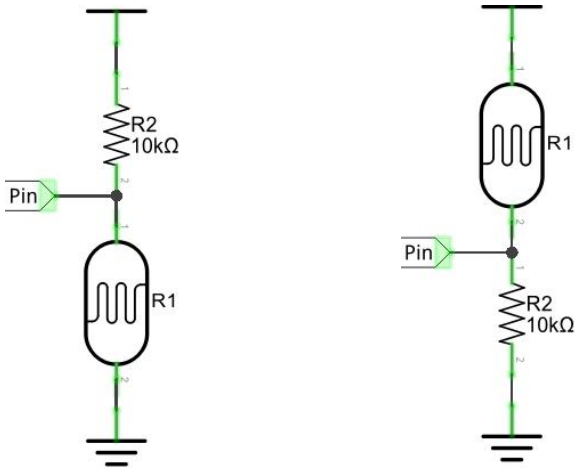
Component knowledge

Photoresistor

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



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

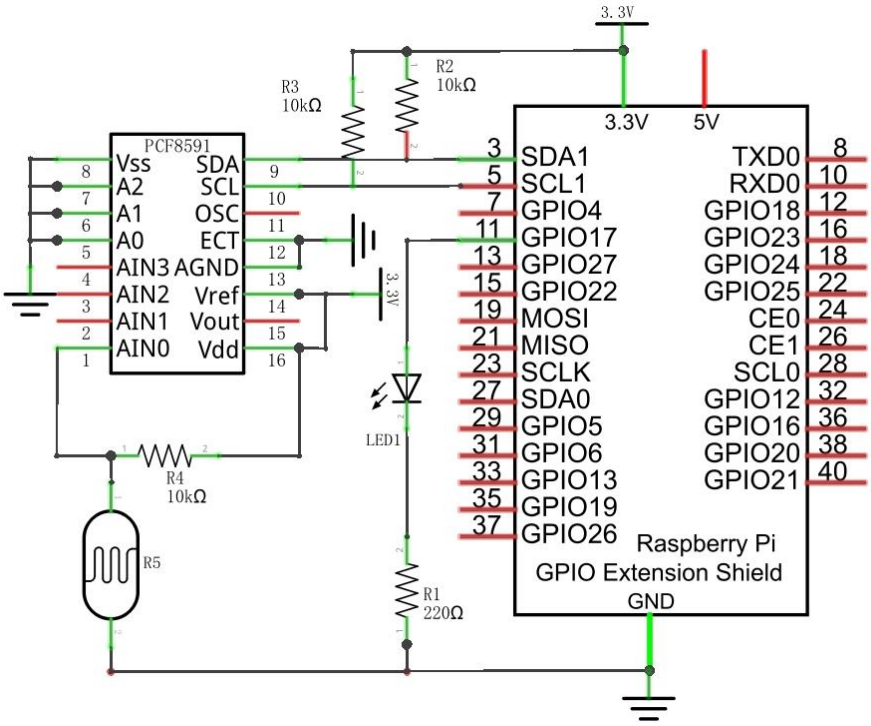


In the above circuit, when photoresistor resistance changes due to light intensity, voltage between photoresistor and resistor R1 will change, so light's intensity can be obtained by measuring the voltage.

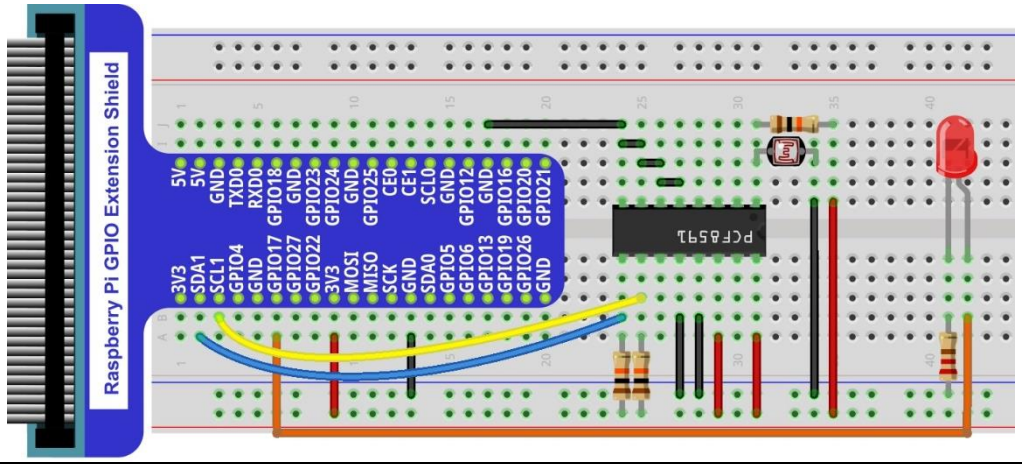
Circuit

The circuit of this experiment is similar to the one in last chapter. The only difference is that the input signal of the AIN0 pin of PCF8591 is changed from a potentiometer to combination of a photoresistor and a resistor.

Schematic diagram



Hardware connection



Code

The code of this experiment is identical with the one in last chapter logically.

C Code 10.1.1 Nightlamp

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 010.1.1_Nightlamp directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/10.1.1_Nightlamp
```

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

```
gcc Nightlamp.c –o Nightlamp –lwiringPi -lpthread
```

3. Then run the generated file "Nightlamp".

```
sudo ./Nightlamp
```

After the program is executed, when you cover the photosensitive resistance or make a flashlight toward the photoresistor, the brightness of LED will be enhanced or weakened. And the terminal window will print out the current input voltage value of PCF8591 AIN0 pin and the converted digital quantity.

The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define ledPin 0
int main(void){
    int value;
    float voltage;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledPin,0,100);
    pcf8591Setup(pinbase,address);

    while(1){
        value = analogRead(A0);  //read A0 pin
        softPwmWrite(ledPin,value*100/255);
        voltage = (float)value / 255.0 * 3.3;  // calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
```

```
29          delay(100);
30      }
31      return 0;
32  }
```

Python Code 10.1.1 Nightlamp

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/10.1.1_Nightlamp
```

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

```
python Nightlamp.py
```

After the program is executed, when you cover the photosensitive resistance or make a flashlight toward the photoresistor, the brightness of LED will be enhanced or weakened. And the terminal window will print out the current input voltage value of PCF8591 AIN0 pin and the converted digital quantity.

The following is the program code:

```
1   import RPi.GPIO as GPIO
2   import smbus
3   import time
4
5   address = 0x48
6   bus=smbus.SMBus(1)
7   cmd=0x40
8   ledPin = 11
9
10  def analogRead(chn):
11      value = bus.read_byte_data(address,cmd+chn)
12      return value
13
14  def analogWrite(value):
15      bus.write_byte_data(address,cmd,value)
16
17  def setup():
18      global p
19      GPIO.setmode(GPIO.BOARD)
20      GPIO.setup(ledPin,GPIO.OUT)
21      GPIO.output(ledPin,GPIO.LOW)
22
23      p = GPIO.PWM(ledPin,1000)
24      p.start(0)
25
26  def loop():
27      while True:
28          value = analogRead(0)
29          p.ChangeDutyCycle(value*100/255)
30          voltage = value / 255.0 * 3.3
```

```
31          print 'ADC Value : %d, Voltage : %.2f'%(value,voltage)
32          time.sleep(0.01)
33
34  def destroy():
35      bus.close()
36      GPIO.cleanup()
37
38  if __name__ == '__main__':
39      print 'Program is starting ... '
40      setup()
41      try:
42          loop()
43      except KeyboardInterrupt:
44          destroy()
```

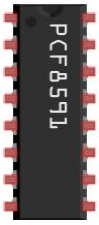
Chapter 11 Thermistor

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

Project 11.1 Thermometer

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

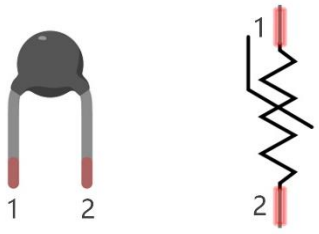
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x14
  </td></tr>
<tr><td>Thermistor x1
  </td><td>PCF8591 x1
  </td><td>Resistor 10kΩ x3
  </td></tr>
</table>

Component knowledge

Thermistor

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



The relationship between resistance value and temperature of thermistor is:

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

Where:

Rt is the thermistor resistance under T2 temperature;

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

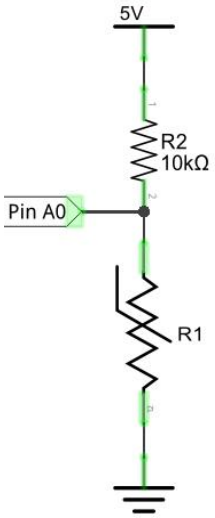
EXP[n] is nth power of E;

B is for thermal index;

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

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

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



We can use the value measured by the analog pin of UNO to obtain resistance value of thermistor, and then can use the formula to obtain the temperature value.

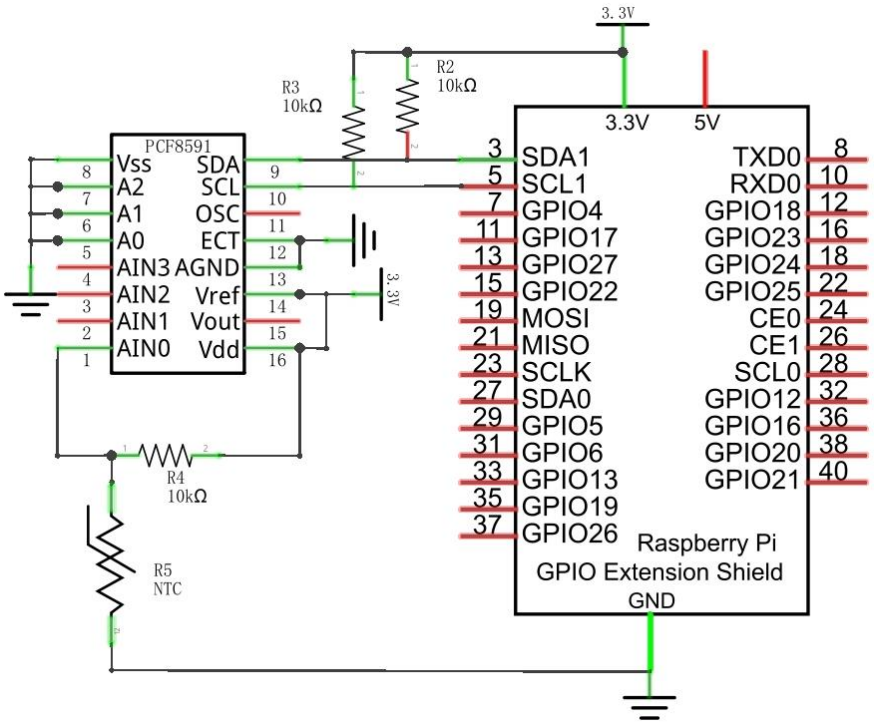
Cconsequently, the temperature formula can be concluded:

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

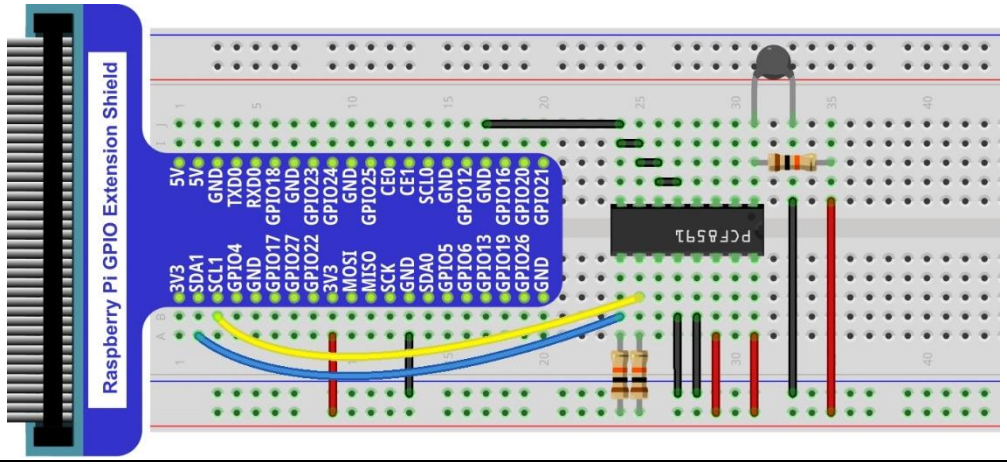
Circuit

The circuit of this experiment is similar to the one in the last chapter. The only difference is that the photoresistor is replaced by the thermistor.

Schematic diagram



Hardware connection



Code

In this experimental code, the ADC value is still needed to be read, and the difference is that a specific formula is used to calculate the temperature value.

C Code 11.1.1 Thermometer

First observe the experimental phenomenon, and then analyze the code.

Use the cd command to enter 11.1.1_Thermometer directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/11.1.1_Thermometer
```

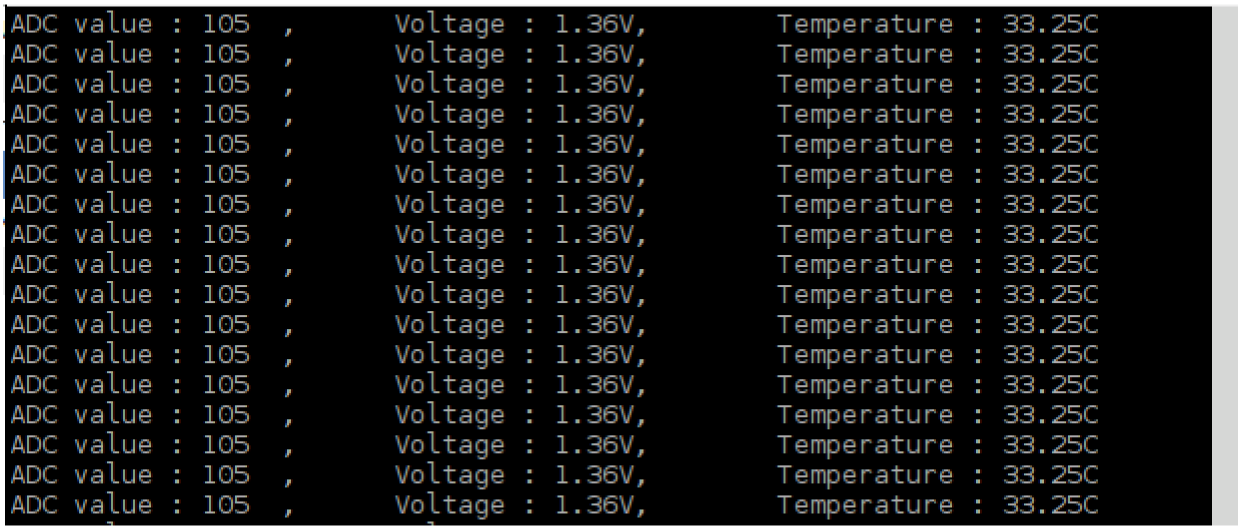
1. Use following command to compile “Thermometer.c” and generate executable file “Thermometer”. “-lm” option is needed.

```
gcc Thermometer.c –o Thermometer –lwiringPi –lm
```

2. Then run the generated file “Thermometer”.

```
sudo ./Thermometer
```

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



```
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
ADC value : 105  ,      Voltage : 1.36V,        Temperature : 33.25C
```

The following is the code:

```c
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <math.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

int main(void){
```

```
14      int adcValue;
15      float tempK,tempC;
16      float voltage,Rt;
17      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
18          printf("setup wiringPi failed !");
19          return 1;
20      }
21      pcf8591Setup(pinbase,address);
22      while(1){
23          adcValue = analogRead(A0);  //read A0 pin
24          voltage = (float)adcValue / 255.0 * 3.3;     // calculate voltage
25          Rt = 10 * voltage / (3.3 - voltage);  //calculate resistance value of thermistor
26          tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0); //calculate temperature (Kelvin)
27          tempC = tempK -273.15;       //calculate temperature (Celsius)
28          printf("ADC value : %d  ,\tVoltage : %.2fV,
29  \tTemperature : %.2fC\n",adcValue,voltage,tempC);
30          delay(100);
31      }
32      return 0;
33  }
```

In the code, read the ADC value of PCF8591 A0 port, and then calculate the voltage and the resistance of thermistor according to Ohms law. Finally, calculate the current temperature. according to the front formula.

Python Code 11.1.1 Thermometer

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/11.1.1_Thermometer
```

2. Use python command to execute python code “Thermometer.py”.

```
python Thermometer.py
```

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

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

The following is the code:

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

address = 0x48
bus=smbus.SMBus(1)
cmd=0x40

def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value

def analogWrite(value):
    bus.write_byte_data(address,cmd,value)

def setup():
    GPIO.setmode(GPIO.BOARD)

def loop():
    while True:
        value = analogRead(0)        #read A0 pin
        voltage = value / 255.0 * 3.3        #calculate voltage
        Rt = 10 * voltage / (3.3 - voltage) #calculate resistance value of thermistor
        tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) #calculate temperature
(Kelvin)
        tempC = tempK -273.15        #calculate temperature (Celsius)
        print 'ADC Value : %d, Voltage : %.2f, Temperature : %.2f'%(value,voltage,tempC)
        time.sleep(0.01)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':
    print 'Program is starting ... '
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

In the code, read the ADC value of PCF8591 A0 port, and then calculate the voltage and the resistance of thermistor according to Ohms law. Finally, calculate the current temperature. according to the front formula.

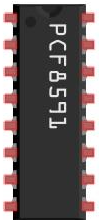
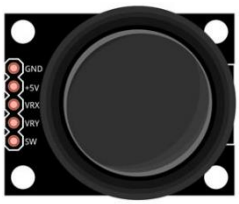
Chapter 12 Joystick

In the previous chapter, we have learned how to use rotary potentiometer. Now, let's learn a new electronic module Joystick which working on the same principle as rotary potentiometer.

Project 12.1 Joystick

In this experiment, we will read the output data of Joystick and print it to the screen.

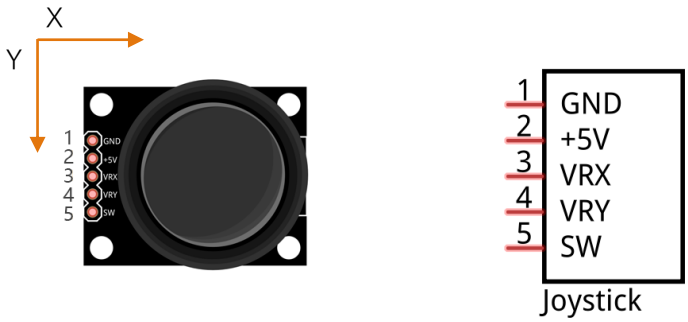
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Expansion Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x12 M/F x5
  </td></tr>
<tr><td>PCF8591 x1
  </td><td>Resistor 10kΩ x2
  </td><td>Joystick x1
  </td></tr>
</table>

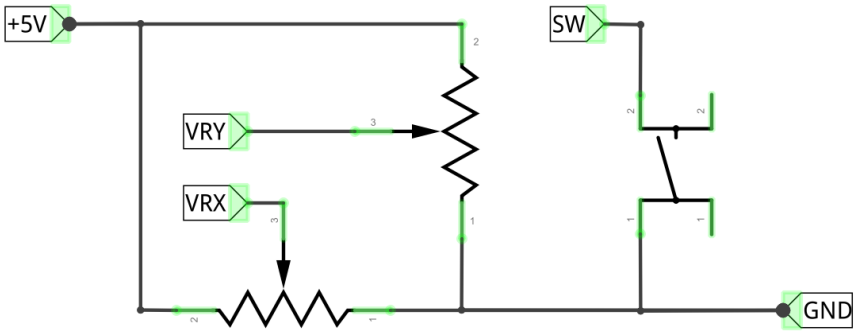
Component knowledge

Joystick

Joystick is a kind of sensor used with your fingers, which is widely used in gamepad and remote controller. It can shift in direction Y or direction X at the same time. And it can also be pressed in direction Z.



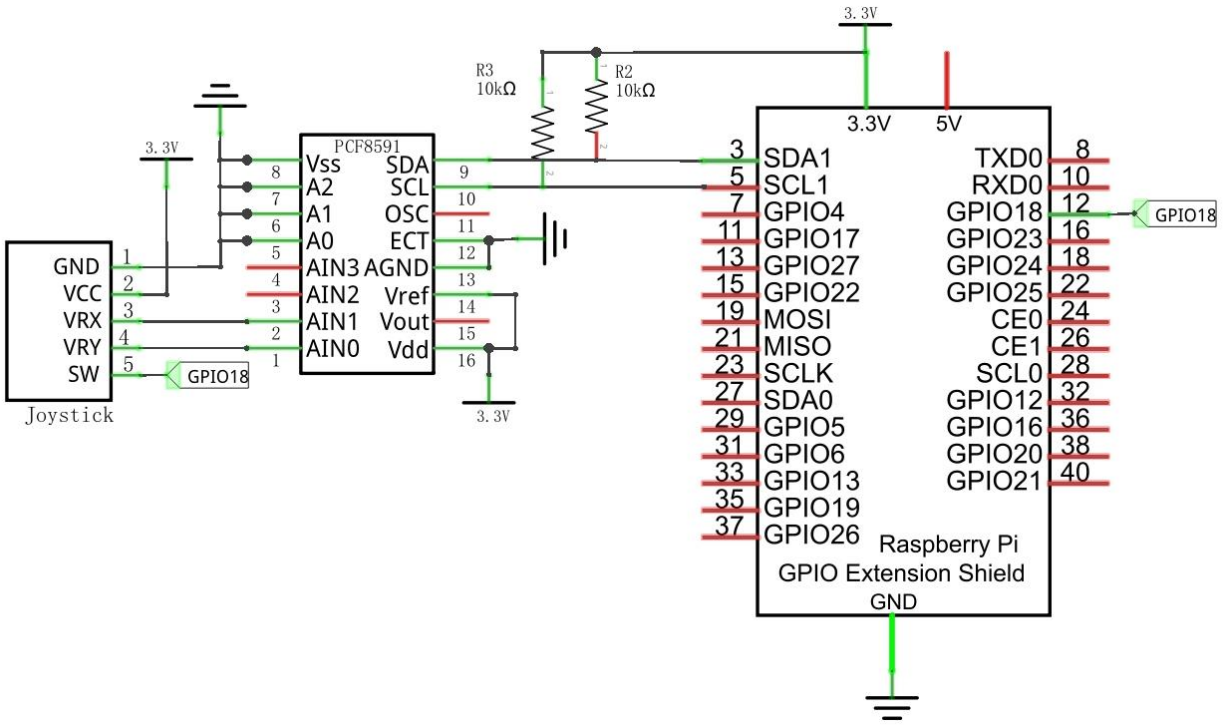
Two rotary potentiometers inside the joystick are set to detect the shift direction of finger, and a push button in vertical direction is set to detect the action of pressing.



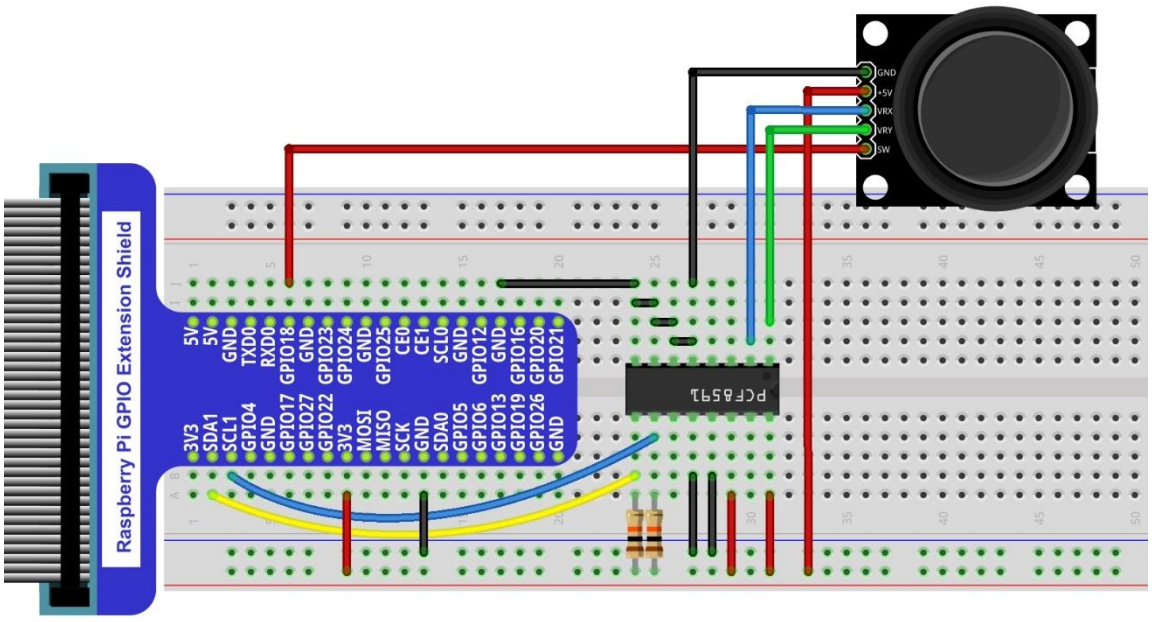
When read the data of joystick, there are some different between axis: data of X and Y axis is analog, wich need to use ADC. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

Circuit

Schematic diagram



Hardware connection



Code

In this experimental code, we will read ADC value of X and Y axis of Joystick, and read digital quality of Z axis, then print these data out.

C Code 12.1.1 Joystick

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/12.1.1_ Joystick
```

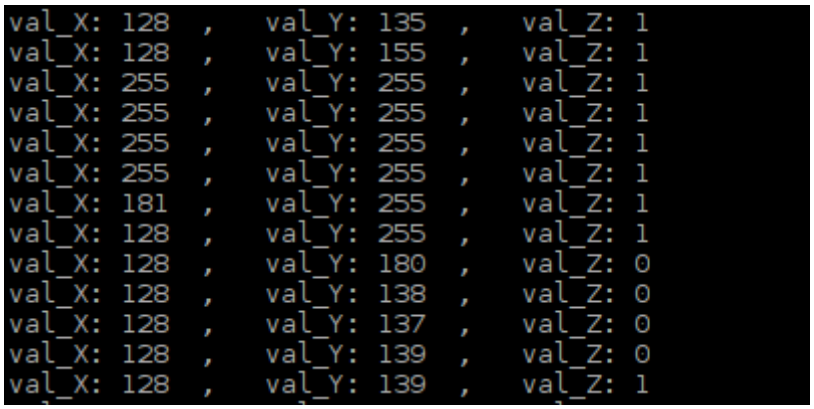
2. Use following command to compile "Joystick.c" and generate executable file "Joystick.c". "-lm" option is needed.

```
gcc Joystick.c –o Joystick–lwiringPi –lm
```

3. Then run the generated file "Joystick".

```
sudo ./Joystick
```

After Program is executed, the terminal window will print out the data of 3 axes X, Y, Z. And shifting the Joystick or pressing it will make those data change.



The flowing is the code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48        //pcf8591 default address
#define pinbase 64          //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define Z_Pin 1     //define pin for axis Z

int main(void){
    int val_X,val_Y,val_Z;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
```

```
18          printf("setup wiringPi failed !");
19          return 1;
20      }
21      pinMode(Z_Pin,INPUT);        //set Z_Pin as input pin and pull-up mode
22      pullUpDnControl(Z_Pin,PUD_UP);
23      pcf8591Setup(pinbase,address);      //initialize PCF8591
24
25      while(1){
26          val_Z = digitalRead(Z_Pin);  //read digital quality of axis Z
27          val_Y = analogRead(A1);      //read analog quality of axis X and Y
28          val_X = analogRead(A2);
29          printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
30          delay(100);
31      }
32      return 0;
33  }
```

In the code, configure Z_Pin to pull-up input mode. In while cycle of main function, use **analogRead** () to read the value of axis X and Y and use **digitalRead** () to read the value of axis Z, then print them out.

```
    while(1){
        val_Z = digitalRead(Z_Pin);  //read digital quality of axis Z
        val_Y = analogRead(A1);      //read analog quality of axis X and Y

        val_X = analogRead(A2);
        printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
        delay(100);
    }
```

Python Code 12.1.1 Joystick

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/12.1.1_ Joystick
```

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

```
python Joystick.py
```

After Program is executed, the terminal window will print out the data of 3 axes X, Y, Z. And shifting the Joystick or pressing it will make those data change.

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

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import smbus
3  import time
4
5  address = 0x48
6  bus=smbus.SMBus(1)
7  cmd=0x40
8  Z_Pin = 12      #define pin for Z_Pin
9  def analogRead(chn):        #read ADC value
10     bus.write_byte(address,cmd+chn)
11     value = bus.read_byte(address)
12     value = bus.read_byte(address)
13     #value = bus.read_byte_data(address,cmd+chn)
14     return value
15
16 def analogWrite(value):
17     bus.write_byte_data(address,cmd,value)
18
19 def setup():
20     global p_Red,p_Green,p_Blue
21     GPIO.setmode(GPIO.BOARD)
22     GPIO.setup(Z_Pin,GPIO.IN,GPIO.PUD_UP)    #set Z_Pin to pull-up mode
23 def loop():
24     while True:
25         val_Z = GPIO.input(Z_Pin)        #read digital quality of axis Z
26         val_Y = analogRead(1)            #read analog quality of axis X and Y
27         val_X = analogRead(2)
```

```
28          print 'value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z)
29          time.sleep(0.01)
30
31  def destroy():
32      bus.close()
33      GPIO.cleanup()
34
35  if __name__ == '__main__':
36      print 'Program is starting ... '
37      setup()
38      try:
39          loop()
40      except KeyboardInterrupt:
41          destroy()
```

In the code, configure Z_Pin to pull-up input mode. In while cycle of loop, use **analogRead** () to read the value of axis X and Y and use **GPIO.input** () to read the value of axis Z, then print them out.

```
    while True:
        val_Z = GPIO.input(Z_Pin)        #read digital quality of axis Z
        val_Y = analogRead(1)            #read analog quality of axis X and Y
        val_X = analogRead(2)
        print 'value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z)
        time.sleep(0.01)
```

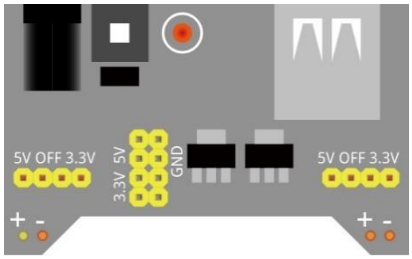
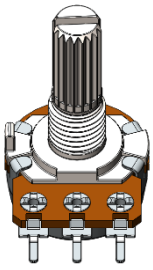
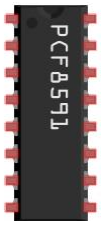
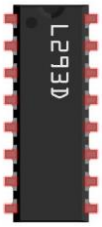
Chapter 13 Motor & Driver

In this chapter, we will learn some knowledge about DC motor and DC motor drive, and how to control the speed and direction of motor.

Project 13.1 Control Motor with Potentiometer

In this experiment, a potentiometer is used to control motor. When the potentiometer is in the midpoint position, the motor will stops rotating, and when away from the middle position, the motor speed increases. When potentiometer is shifted to limited ends, the motor speed reaches maximum. When the potentiometer position is at different side of middle position, the direction of motor is different.

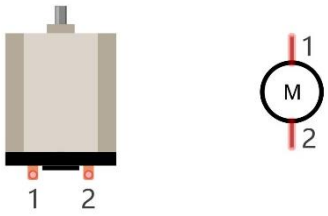
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x22
  </td></tr>
<tr><td colspan="2">Breadboard power module x1
 
</td><td colspan="3">9V Battery (provided by yourself) & battery cable
 
</td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Motor x1
  </td><td>Resistor 10kΩ x2
  </td><td>PCF8591 x1
 
</td><td>L293D
 
</td></tr>
</table>

Component knowledge

Motor

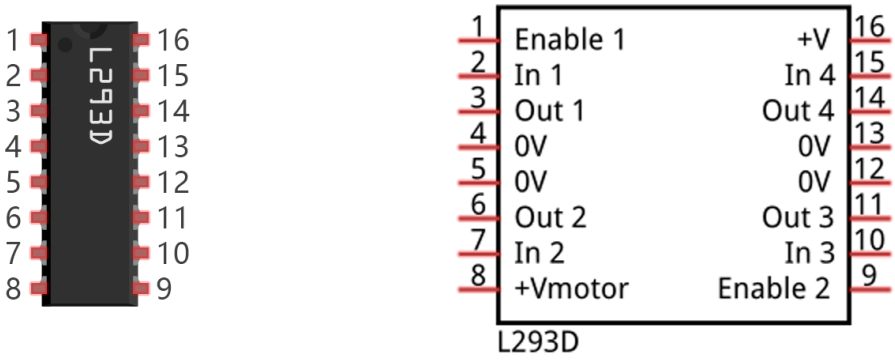
Motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive other mechanical devices to run. Diagram below is a small DC motor with two pins.



When motor get connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then motor rotates in opposite direction.

L293D

L293D is a chip integrated with 4-channel motor drive. You can drive a unidirectional motor with 4 ports or a bi-directional motor with 2 port or a stepper motor.



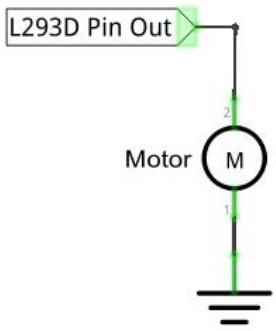
Port description of L293D module is as follows:

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

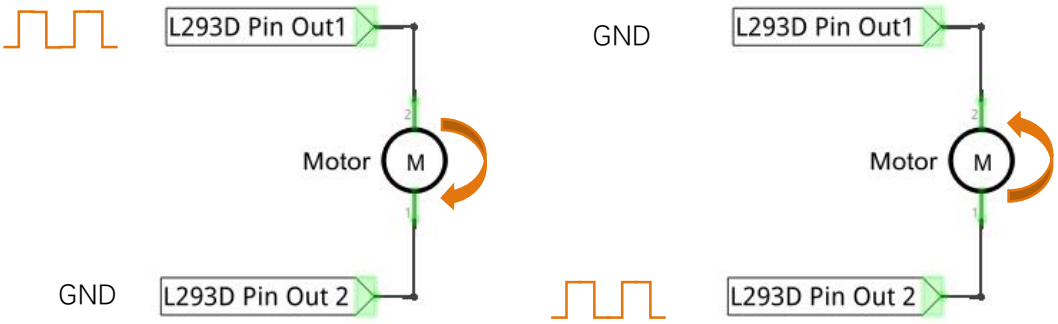
For more details, please see datasheet.

When using L293D to drive DC motor, there are usually two kinds of connection.

Following connection uses one channel, and it can control motor speed through PWM, but the motor can only rotate in one direction.



Following connection uses two channels: one channel outputs PWM wave, and another channel connects GND, so you can control the speed of motor. Whe n these two channel signals are exchanged, the current direction of the motor can be reversed, and the motor will rotate in reverse direction. This can not only control the speed of motor, but also can control the steering of motor.

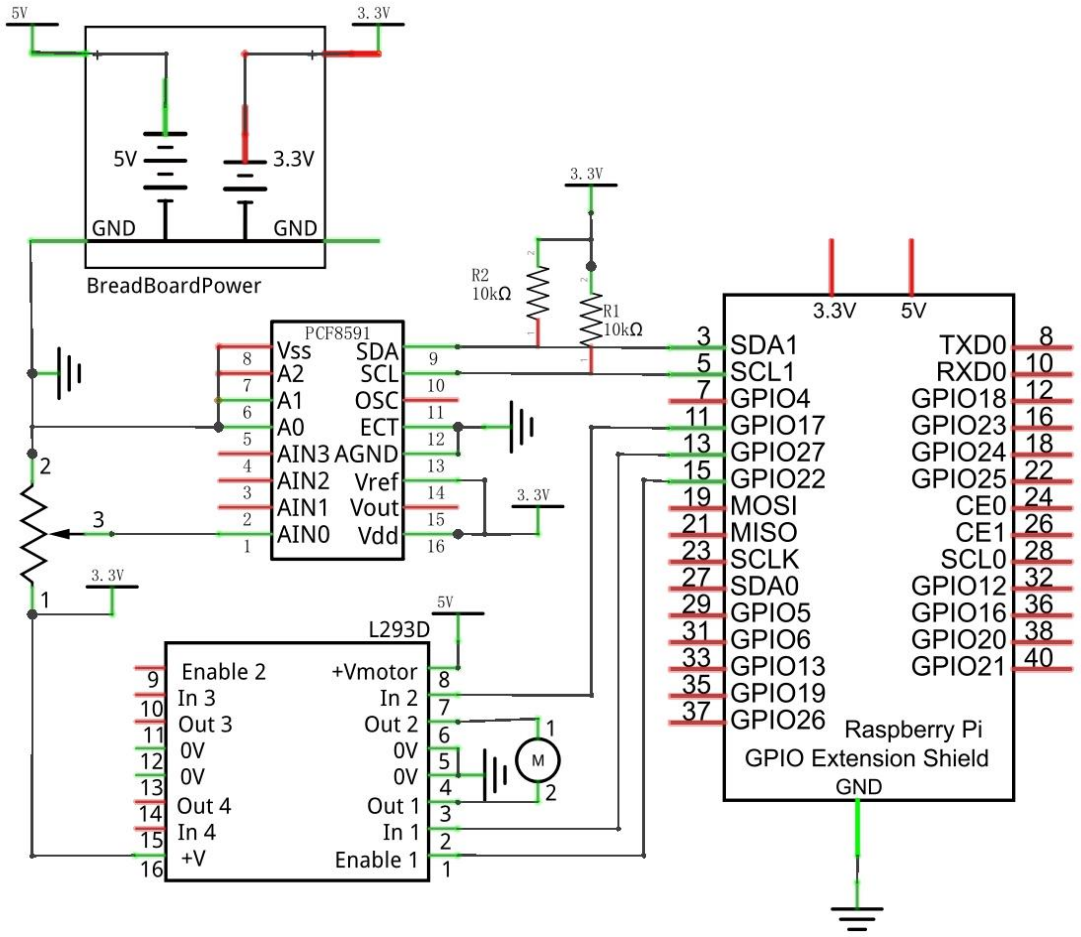


In actual use, motor is usually connected to the channel 1 and 2, output different level to in1 and in2 to control the rotation direction of the motor, and output PWM wave to Enable1 port to control the motor rotation speed. Or, get motor connected to the channel 3 and 4, output different level to in3 and in4 to control the motor's rotation direction, and output PWM wave to Enable2 pin to control the motor rotation speed.

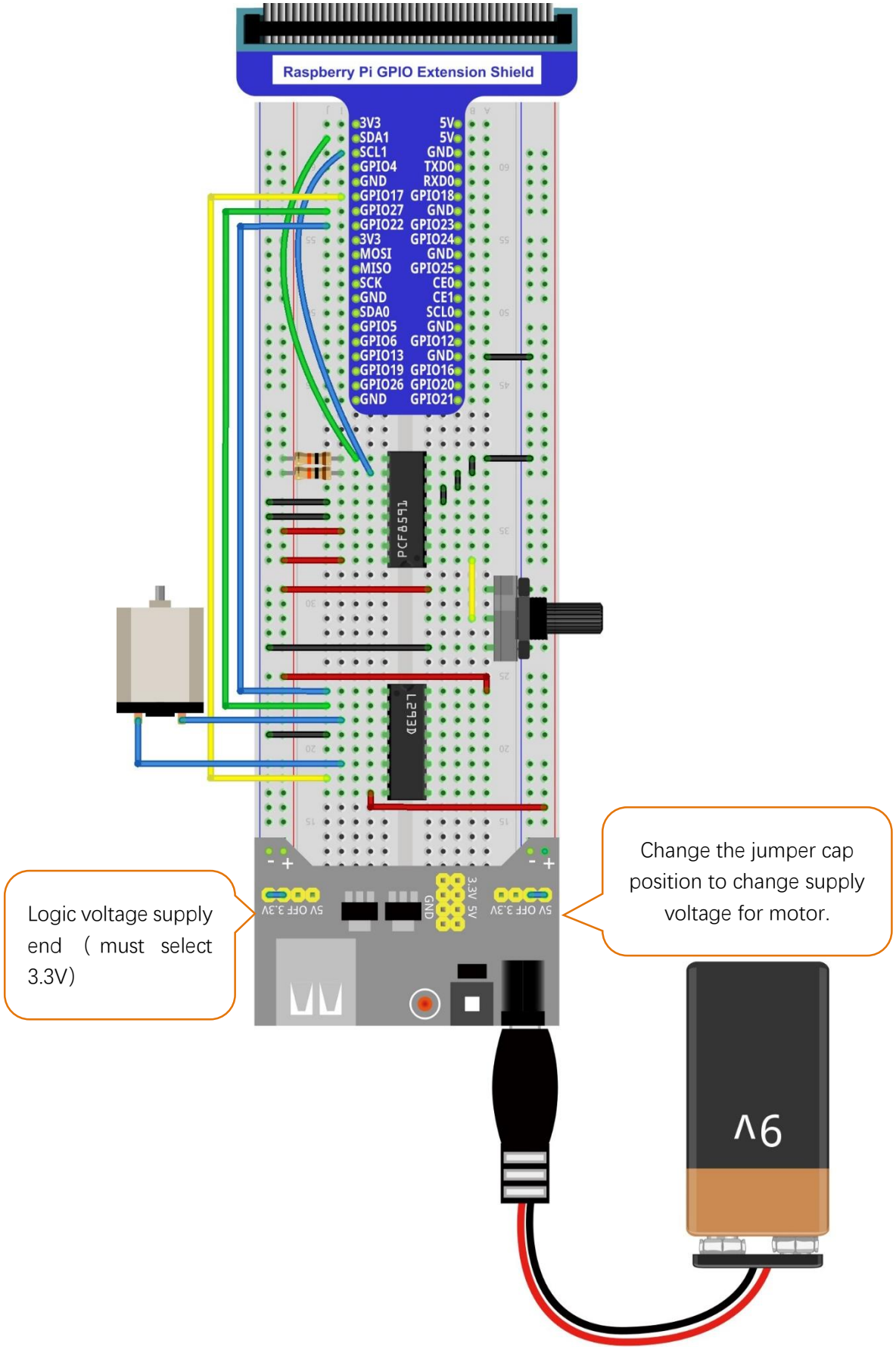
Circuit

When connecting the circuit, pay attention to that beacause the motor is a high-power component, do not use the power provided by the RPi, which may do damage to your RPi. the logic circuit can be powered by RPi power or external power supply which should have the common ground with RPi.

Schematic diagram



Hardware connection



Code

In this experimental code, first read the ADC value, and then control the rotation direction and speed of the motor according to the value of the ADC.

C Code 13.1.1 Motor

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/13.1.1_Motor
```

2. Use following command to compile "Motor.c" and generate executable file "Motor". "-lm" and "-lpthread" option is needed.

```
gcc Motor.c –o Motor–lwiringPi –lm -lpthread
```

3. Then tun the generated file "Motor".

```
sudo ./Motor
```

After the program is executed, shift the potentiometer, then the rotation speed and direction of the motor will change with it. And when the potentiometer is turned to midpoint position, the motor stops running. When away from the middle position, the motor speed will increase. When to both ends, motor speed reach to maximum. When the potentiometer is turned to different side of the middle position, the motor will run with different direction. Meanwhile, the terminal will print out ADC value of the potentiometer, the motor direction and the PWM duty cycle used to control motor speed.

```
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
```

The following is the code:

```
1  #include <wiringPi.h>
2  #include <pcf8591.h>
3  #include <stdio.h>
4  #include <softPwm.h>
5  #include <math.h>
6  #include <stdlib.h>
7
8  #define address 0x48        //pcf8591 default address
9  #define pinbase 64          //any number above 64
```

```
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define motoRPin1   2       // define the pin connected to L293D
#define motoRPin2   0
#define enablePin   3
// Map function: map the value from a range of mapping to another range.
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
//motor function: determine the direction and speed of the motor according to the ADC
value to be input.
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        digitalWrite(motoRPin1,HIGH);
        digitalWrite(motoRPin2,LOW);
        printf("turn Forward...\n");
    }
    else if (value<0){
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,HIGH);
        printf("turn Back...\n");
    }
    else {
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,LOW);
        printf("Motor Stop...\n");
    }
    softPwmWrite(enablePin,map(abs(value),0,128,0,255));
    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);//print the PMW duty cycle.
}
int main(void){
    int value;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(enablePin,OUTPUT);// set mode for the pin
    pinMode(motoRPin1,OUTPUT);
    pinMode(motoRPin2,OUTPUT);
    softPwmCreate(enablePin,0,100);// define PMW pin
```

```
54      pcf8591Setup(pinbase,address);//initialize PCF8591
55
56      while(1){
57          value = analogRead(A0);   //read A0 pin
58          printf("ADC value : %d \n",value);
59          motor(value);         // start the motor
60          delay(100);
61      }
62      return 0;
63  }
```

We have been familiar with reading ADC value. So, let's learn directly the subfunction void motor(int ADC): first, compare ADC value with 128 (value corresponding to midpoint). When the current ADC value is greater, make motoRPin1 output high level and motoRPin2 output low level to control motor to run with corotation direction. When the current ADC value, make motoRPin1 output low level and motoRPin2 output high level to control run with reversed direction. When the ADC value is equal to 128, make motoRPin1 and motoRPin2 output low level, then the motor stops. And then determine PWM duty cycle according to the difference between ADC value and 128. Because the absolute difference value stays within 0-128. We need to use the map () sub function mapping the difference value to range of 0-255. Finally print out the duty cycle.

```
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        digitalWrite(motoRPin1,HIGH);
        digitalWrite(motoRPin2,LOW);
        printf("turn Forward...\n");
    }
    else if (value<0){
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,HIGH);
        printf("turn Backward...\n");
    }
    else {
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,LOW);
        printf("Motor Stop...\n");
    }
    softPwmWrite(enablePin,map(abs(value),0,128,0,255));
    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);// print out PMW duty
cycle.
}
```

Pyhton Code 13.1.1 Motor

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/13.1.1_Motor
```

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

```
python Motor.py
```

After the program is executed, shift the potentiometer, then the rotation speed and direction of the motor will change with it. And when the potentiometer is turned to midpoint position, the motor stops running. When away from the middle position, the motor speed will increase. When to both ends, motor speed reach to maximum. When the potentiometer is turned to different side of the middle position, the motor will run with different direction. Meanwhile, the terminal will print out ADC value of the potentiometer, the motor direction and the PWM duty cycle used to control motor speed.

```
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%
```

The following is the code:

```
1  import RPi.GPIO as GPIO
2  import smbus
3  import time
4
5  address = 0x48
6  bus=smbus.SMBus(1)
7  cmd=0x40
8  # define the pin connected to L293D
9  motoRPin1 = 13
10 motoRPin2 = 11
11 enablePin = 15
12
13 def analogRead(chn):
14     value = bus.read_byte_data(address,cmd+chn)
15     return value
16
17 def analogWrite(value):
18     bus.write_byte_data(address,cmd,value)
19
```

```
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)     # set mode for pin
    GPIO.setup(motoRPin1,GPIO.OUT)
    GPIO.setup(motoRPin2,GPIO.OUT)
    GPIO.setup(enablePin,GPIO.OUT)

    p = GPIO.PWM(enablePin,1000)# creat PWM
    p.start(0)
#mapNUM function: map the value from a range of mapping to another range.

def mapNUM(value,fromLow,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
#motor function: determine the direction and speed of the motor according to the ADC
value to be input.
def motor(ADC):
    value = ADC -128
    if (value > 0):
        GPIO.output(motoRPin1,GPIO.HIGH)
        GPIO.output(motoRPin2,GPIO.LOW)
        print 'Turn Forward...'
    elif (value < 0):
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.HIGH)
        print 'Turn Backward...'
    else :
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.LOW)
        print 'Motor Stop...'
    p.start(mapNUM(abs(value),0,128,0,100))
    print 'The PWM duty cycle is %d%%\n'%(abs(value)*100/127)    #print PMW duty cycle.

def loop():
    while True:
        value = analogRead(0)
        print 'ADC Value : %d'%(value)
        motor(value)
        time.sleep(0.01)

def destroy():
    bus.close()
    GPIO.cleanup()

if __name__ == '__main__':
```

```
64      print 'Program is starting ... '
65      setup()
66      try:
67          loop()
68      except KeyboardInterrupt:
69          destroy()
```

We have been familiar with reading ADC value. So, let's learn directly the subfunction `def motor(ADC)`: first, compare ADC value with 128 (value corresponding to midpoint). When the current ADC value is greater, make motoRPin1 output high level and motoRPin2 output low level to control motor to run with corotation direction. When the current ADC value, make motoRPin1 output low level and motoRPin2 output high level to control run with reversed direction. When the ADC value is equal to 128, make motoRPin1 and motoRPin2 output low level, then the motor stops. And then determine PWM duty cycle according to the difference between ADC value and 128. Because the absolute difference value stays within 0-128. We need to use the map () sub function mapping the difference value to range of 0-255. Finally print out the duty cycle.

```
def motor(ADC):
    value = ADC -128
    if (value > 0):
        GPIO.output(motoRPin1,GPIO.HIGH)
        GPIO.output(motoRPin2,GPIO.LOW)
        print 'Turn Forward...'
    elif (value < 0):
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.HIGH)
        print 'Turn Backward...'
    else :
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.LOW)
        print 'Motor Stop...'
    p.start(mapNUM(abs(value),0,128,0,100))
    print 'The PWM duty cycle is %d%%\n'%(abs(value)*100/127)    #print PMW duty cycle.
```

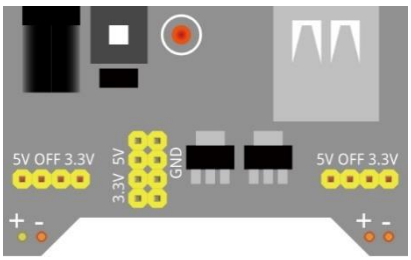
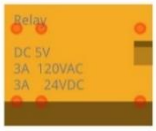
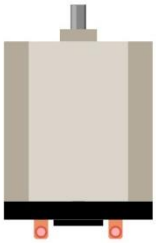
Chapter 14 Relay & Motor

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

Project 14.1.1 Relay & Motor

In this experiment, we will use a push button to control a relay and drive the motor.

Component List

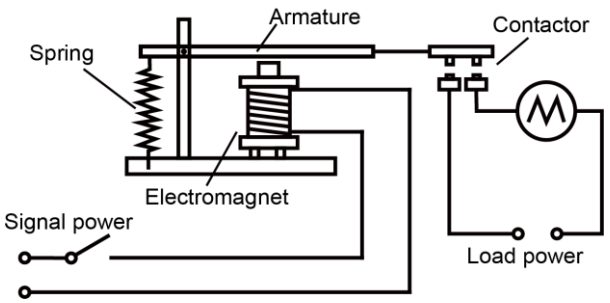
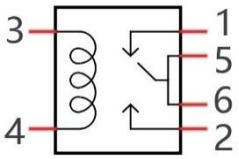
<table>
<tr><td colspan="3">Raspberry Pi 3B x1
GPIO Expansion Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x8
  </td></tr>
<tr><td colspan="6">9V battery (prepared by yourself) & battery line
  </td></tr>
<tr><td colspan="3">Breadboard extension x1
  </td><td>Resistor 10kΩ x2
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 220Ω x1
  </td></tr>
<tr><td>NPN transistor x1
  </td><td>Relay x1
  </td><td>Motor x1
  </td><td>Push button x1
  </td><td>LED x1
  </td><td>Diode x1
  </td></tr>
</table>

Component knowledge

Relay

Relay is a safe switch which can use low power circuit to controll high power circuit. It consists of electromagnet and contacts. The electromagnet is controlled by low power circuit and contacts is used in high power circuit. When the electromagnet is energized, it will attract contacts.

The following is a principle diagram of common relay and the feature and circuit symbol of 5V relay used in this experiment:

Diagram	Feature：	Symbol
		

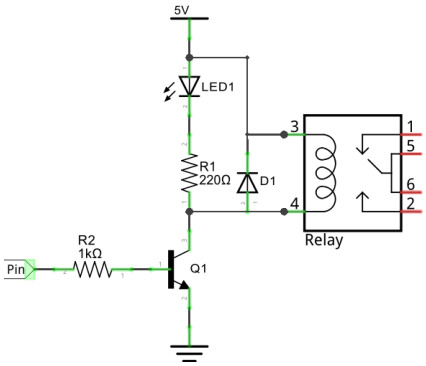
Pin 5 and pin 6 are connected to each other inside. When the coil pin3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

Inductor

The unit of inductance(L) is the henry (H). 1H=1000mH, 1mH=1000μH.

Inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductor will hinder the changing current passing through the inductor. When the current passing through inductor increases, it will attempt to hinder the increasing trend of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing trend of current. So the current passing through inductor is not transient.

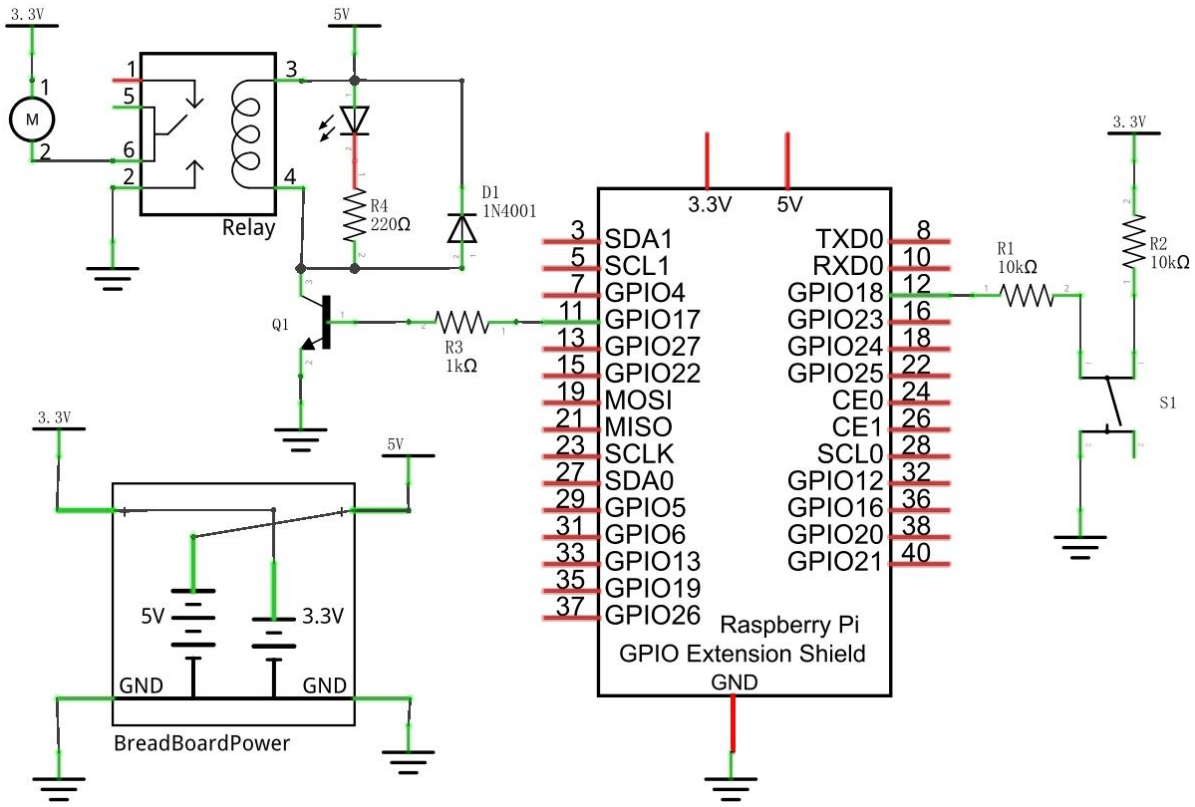
The reference circuit for relay is as follows. The coil of relay can be equivalent to inductor, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to both ends of relay coil pin in reversing direction, then the current will pass through diode, avoiding the impact on power supply.



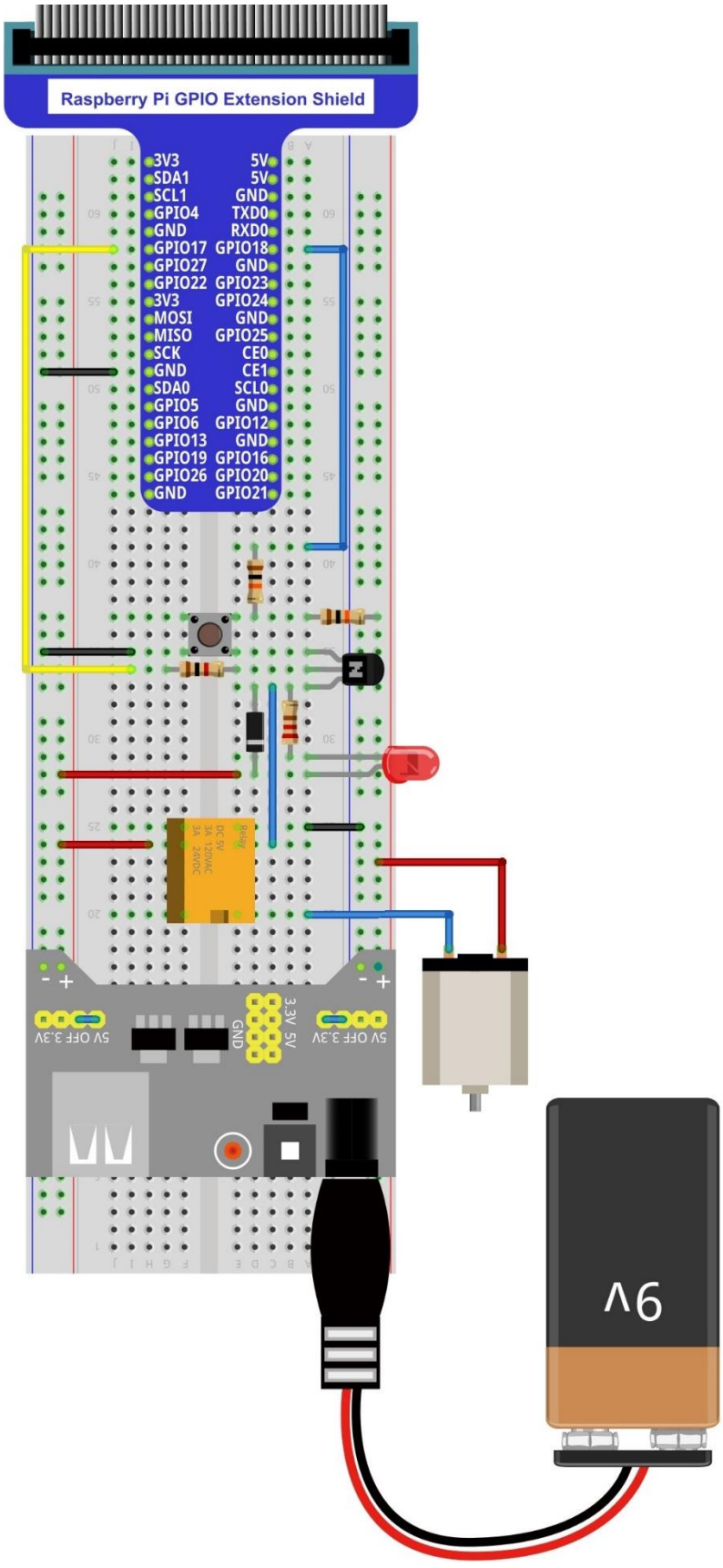
Circuit

Pay attention to the power supply voltage needed for the components in circuit, in which the relay needs power supply voltage 5V, and the motor needs 3.3V. Additionally, a LED is used as an indicator for the relay (turned on or turned off).

Schematic diagram



Hardware connection



Code

The experimental code is in the same logic as TableLamp. Press the button to driver the transistor conducted. Because the relay and LED are connected in parallel, they will be opened at the same time. And if you press the button again, they will be closed.

C Code 14.1.1 Relay

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/14.1.1_Relay
```

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

```
gcc Relay.c –o Relay –lwiringPi
```

3. Run the generated file "Relay".

```
sudo ./Relay
```

After the program is executed, press the button, then the relay is opened, the Motor starts to rotate and LED is turned on. If you press the button again, the relay is closed, the Motor stops running, and the LED is turned off.

The following is the program code:

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

#define relayPin    0   //define the relayPin
#define buttonPin 1     //define the buttonPin
int relayState=LOW;     //store the State of relay
int buttonState=HIGH;   //store the State of button
int lastbuttonState=HIGH;//store the lastState of button
long lastChangeTime;    //store the change time of button state
long captureTime=50;    //set the button state stable time
int reading;
int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring fairelay,print messageto screen
        printf("setup wiringPi fairelay !");
        return 1;
    }
    printf("Program is starting...\n");
    pinMode(relayPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
    while(1){
        reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){  //if the button state has changed ,record the
time point
            lastChangeTime = millis();
```

```
27          }
28          //if changing-state of the button last beyond the time we set,we considered that
29          //the current button state is an effective change rather than a buffeting
30          if(millis() - lastChangeTime > captureTime){
31              //if button state is changed ,update the data.
32              if(reading != buttonState){
33                  buttonState = reading;
34                  //if the state is low ,the action is pressing
35                  if(buttonState == LOW){
36                      printf("Button is pressed!\n");
37                      relayState = !relayState;
38                      if(relayState){
39                          printf("turn on relay ...\n");
40                      }
41                      else {
42                          printf("turn off relay ...\n");
43                      }
44                  }
45                  //if the state is high ,the action is releasing
46                  else {
47                      printf("Button is released!\n");
48                  }
49              }
50          }
51          digitalWrite(relayPin,relayState);
52          lastbuttonState = reading;
53      }
54
55      return 0;
56  }
```

The code is in the same logic as TableLamp code above.

Pyhton Code 14.1.1 Relay

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/14.1.1_ Relay
```

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

```
python Relay.py
```

After the program is executed, press the button, then the relay is opened, the Motor starts to rotate and LED is turned on. If you press the button again, the relay is closed, the Motor stops running, and the LED is turned off.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2
```

```
relayPin = 11    # define the relayPin
buttonPin = 12    # define the buttonPin
relayState = False

def setup():
    print 'Program is starting...'
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(relayPin, GPIO.OUT)   # Set relayPin'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

def buttonEvent(channel):
    global relayState
    print 'buttonEvent GPIO%d'%channel
    relayState = not relayState
    if relayState :
        print 'Turn on relay ... '
    else :
        print 'Turn off relay ... '
    GPIO.output(relayPin,relayState)

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

def destroy():
    GPIO.output(relayPin, GPIO.LOW)     # relay 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()
```

The code is in the same logic as TableLamp code above.

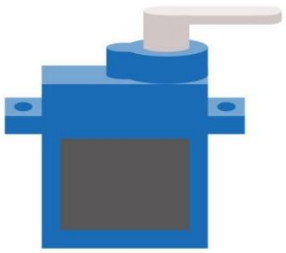
Chapter 15 Servo

We have learned how to control the speed and steering of the motor before. In this chapter, we will learn a kind of motor that can rotate to a specific angle, servo.

Project 15.1 Servo Sweep

First, let's learn how to make the servo rotate.

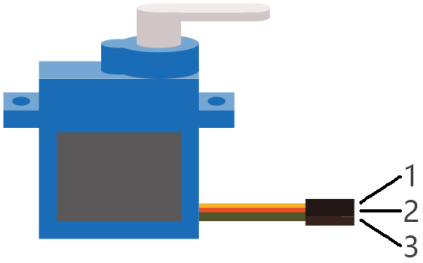
Component List

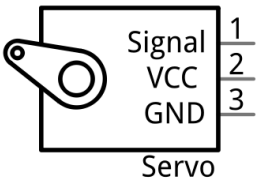
Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	Jumper M/M x3 
Servo x1 	

Component knowledge

Servo

Servo is an auto-control system, consisting of DC motor, reduction gear, sensor and control circuit. Usually, it can rotate in the range of 180 degrees. Servo can output larger torque and is widely used in model airplane, robot and so on. It has three lines, including two for electric power line positive (2-VCC, red), negative (3-GND, brown), and the signal line (1-Signal, orange).





We use 50Hz PWM signal with a duty cycle in a certain range to drive the servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

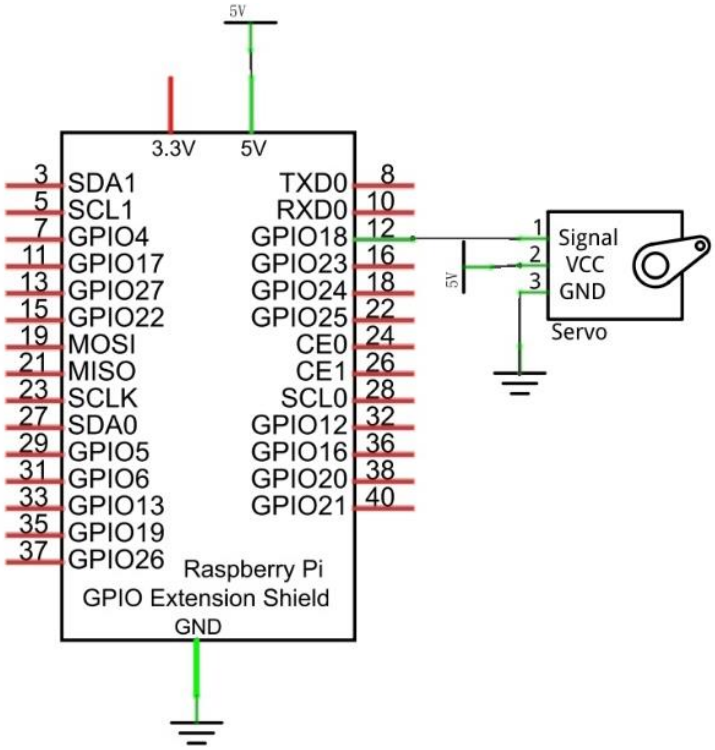
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

When you change the servo signal, servo will rotate to the designated position.

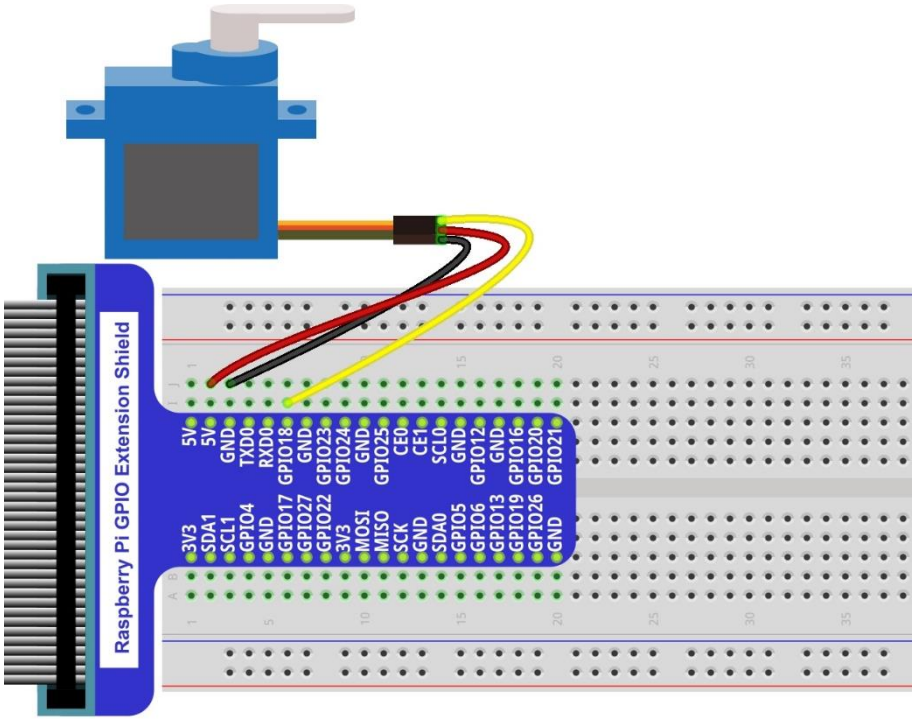
Circuit

Pay attention to the power supply for stepping motor is 5v, and don't confuse the line sequence.

Schematic diagram



Hardware connection



Code

In this experiment, we make the servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

C Code 15.1.1 Sweep

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/15.1.1_Sweep
```

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

```
gcc Sweep.c –o Sweep –lwiringPi
```

3. Run the generated file "Sweep".

```
sudo ./Sweep
```

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

The following is the program code:

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define OFFSET_MS 3     //Define the unit of servo pulse offset: 0.1ms
#define SERVO_MIN_MS 5+OFFSET_MS        //define the pulse duration for minimum angle of servo
#define SERVO_MAX_MS 25+OFFSET_MS       //define the pulse duration for maximum angle of servo

#define servoPin    1       //define the GPIO number connected to servo
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void servoInit(int pin){        //initialization function for servo PMW pin
    softPwmCreate(pin,  0,  200);
}
void servoWrite(int pin, int angle){    //Specif a certain rotation angle (0-180) for the servo
    if(angle > 180)
        angle = 180;
    if(angle < 0)
        angle = 0;
    softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
}
void servoWriteMS(int pin, int ms){     //specific the unit for pulse(5-25ms) with specific duration output by servo pin: 0.1ms
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
```

```
29      if(ms < SERVO_MIN_MS)
30          ms = SERVO_MIN_MS;
31      softPwmWrite(pin,ms);
32  }
33
34  int main(void)
35  {
36      int i;
37      if(wiringPiSetup() == -1){ //when initialize wiring faiservo,print messageto screen
38          printf("setup wiringPi faiservo !");
39          return 1;
40      }
41      printf("Program is starting ...\n");
42      servoInit(servoPin);         //initialize PMW pin of servo
43      while(1){
44          for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle
45  to maximum angle
46              servoWriteMS(servoPin,i);
47              delay(10);
48          }
49          delay(500);
50          for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle
51  to minimum angle
52
53              servoWriteMS(servoPin,i);
54              delay(10);
55          }
56          delay(500);
57      }
58      return 0;
59  }
```

50 Hz pulse, namely cycle for 20ms, is required to control Servo. In function **softPwmCreate** (int pin, int initialValue, int pwmRange), the unit of third parameter pwmRange is 100US, namely 0.1ms. In order to get the PWM with cycle of 20ms, the pwmRange shoulde be set to 200. So in sub function of servoInit (), we create a PWM pin with pwmRange 200.

```
    void servoInit(int pin){         //initialization function for servo PMW pin
        softPwmCreate(pin,  0,  200);
    }
```

As 0-180 degrees of servo corresponds to PWM pulse width 0.5-2.5ms, with PwmRange 200 and unit 0.1ms. So, in function **softPwmWrite** (int pin, int value), the scope 5-25 of parameter value corresponds to 0-180 degrees of servo. What's more, the number writen in subfunction **servoWriteMS** () should be within the range of 5-25. However, in practice, due to the manufacture error of each servo, pulse width will also have deviation. So we define a minimum pulse width and a maximum one and an error offset.

```
#define OFFSET_MS 3     //Define the unit of servo pulse offset: 0.1ms
#define SERVO_MIN_MS 5+OFFSET_MS        //define the pulse duration for minimum angle of
servo
#define SERVO_MAX_MS 25+OFFSET_MS       //define the pulse duration for maximum angle of
servo
......
void servoWriteMS(int pin, int ms){
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}
```

In subfunction **servoWrite** (), input directly angle (0-180 degrees), and map the angle to the pulse width and then output it.

```
void servoWrite(int pin, int angle){    //Specif a certain rotation angle (0-180) for the
servo
    if(angle > 180)
        angle = 180;
    if(angle < 0)
        angle = 0;
    softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
}
```

Finally, in the "while" cycle of main function, use two "for" cycle to make servo ratate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
    while(1){
        for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle
to maximum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
        for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle
to minimum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
    }
```

Pyhton Code 15.1.1 Sweep

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/15.1.1_Sweep
```

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

```
python Sweep.py
```

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
OFFSE_DUTY = 0.5        #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
servoPin = 12

def map( value, fromLow, fromHigh, toLow, toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(servoPin, GPIO.OUT)   # Set servoPin's mode is output
    GPIO.output(servoPin, GPIO.LOW)  # Set servoPin to low

    p = GPIO.PWM(servoPin, 50)      # set Frequece to 50Hz
    p.start(0)                      # Duty Cycle = 0

def servoWrite(angle):      # make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY))#map the angle to
duty cycle and output it

def loop():
    while True:
        for dc in range(0, 181, 1):   #make servo rotate from 0°  to 180°
            servoWrite(dc)      # Write to servo
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): #make servo rotate from 180° to 0°
```

```
35
36              servoWrite(dc)
37              time.sleep(0.001)
38          time.sleep(0.5)
39
40  def destroy():
41      p.stop()
42      GPIO.cleanup()
43
44  if __name__ == '__main__':     #Program start from here
45      print 'Program is starting...'
46      setup()
47      try:
48          loop()
49      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
50  will be  executed.
51          destroy()
```

50 Hz pulse, namely cycle for 20ms, is required to control Servo. So we need set set PMW frequece of servoPin to 50Hz.

```
    p = GPIO.PWM(servoPin, 50)      # set Frequece to 50Hz
```

As 0-180 degrees of servo corresponds to PWM pulse width 0.5-2.5ms within cycle 20ms and to duty cycle 2.5%-12.5% . In subfunction **servoWrite** (angle), map the angle to duty cycle to ouput the PWM, then the servo will rotate a specific angle. However, in practice, due to the manufacture error of each servo, pulse width will also have deviation. So we define a minimum pulse width and a maximum one and an error offset.

```
    OFFSE_DUTY = 0.5        #define pulse offset of servo
    SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
    SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
    ……
    def servoWrite(angle):      #make the servo rotate to specific angle (0-180 degrees)
        if(angle<0):
            angle = 0
        elif(angle > 180):
            angle = 180
        p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY))
```

Finally, in the "while" cycle of main function, use two "for" cycle to make servo ratate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
def loop():
    while True:
        for dc in range(0, 181, 1):   #make servo rotate from 0° to 180°
            servoWrite(dc)      # Write to servo
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): #make servo rotate from 180° to 0°
            servoWrite(dc)
            time.sleep(0.001)
        time.sleep(0.5)
```

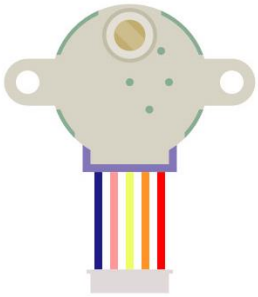
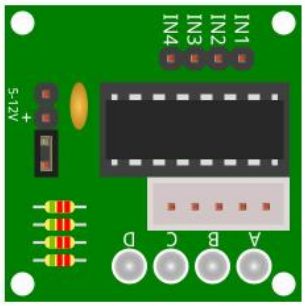
Chapter 16 Stepping Motor

We have learned DC motor and servo before: the DC motor can rotate onstantly but we can not make it rotate to a specific angle. On the contrary, the ordinary servo can rotate to a certain angle but can not rotate constantly. In thischapter, we will learn a motor which can rotate not only constanly, but also to a specific angle, stepping motor. Using stepping motor can achieve higher accuracy of mechanical motion easily.

Project 16.1 Stepping Motor

In this experiment, we will learn how to drive stepping motor, and understand its working principle.

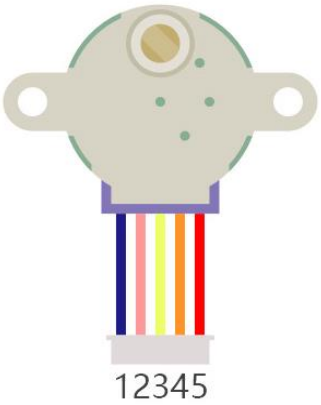
Component List

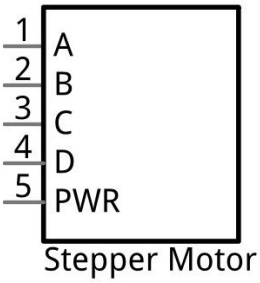
Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	Jumper M/M x1 M/F x6 
Stepping Motor x1 	ULN2003 Stepper Motor Driver x1 

Component knowledge

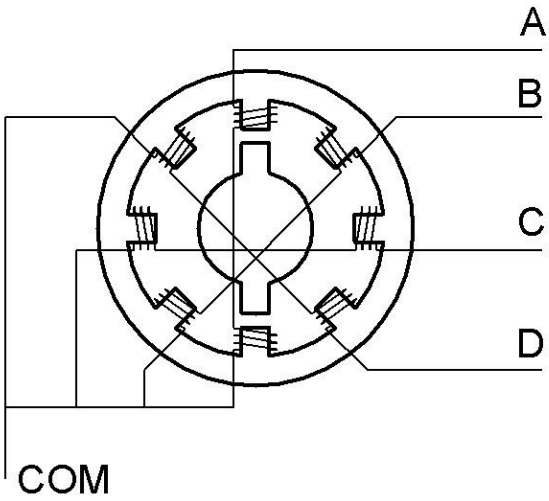
Stepping Motor

Step motor is a open-loop control device which converts the electric pulse signal into angular displacement or linear displacement. In non overload condition, the speed of the motor and the location of the stop depends only on the pulse signal frequency and pulse number, and not affected by the load changes. A small four-phase deceleration stepping motor is shown as follows:



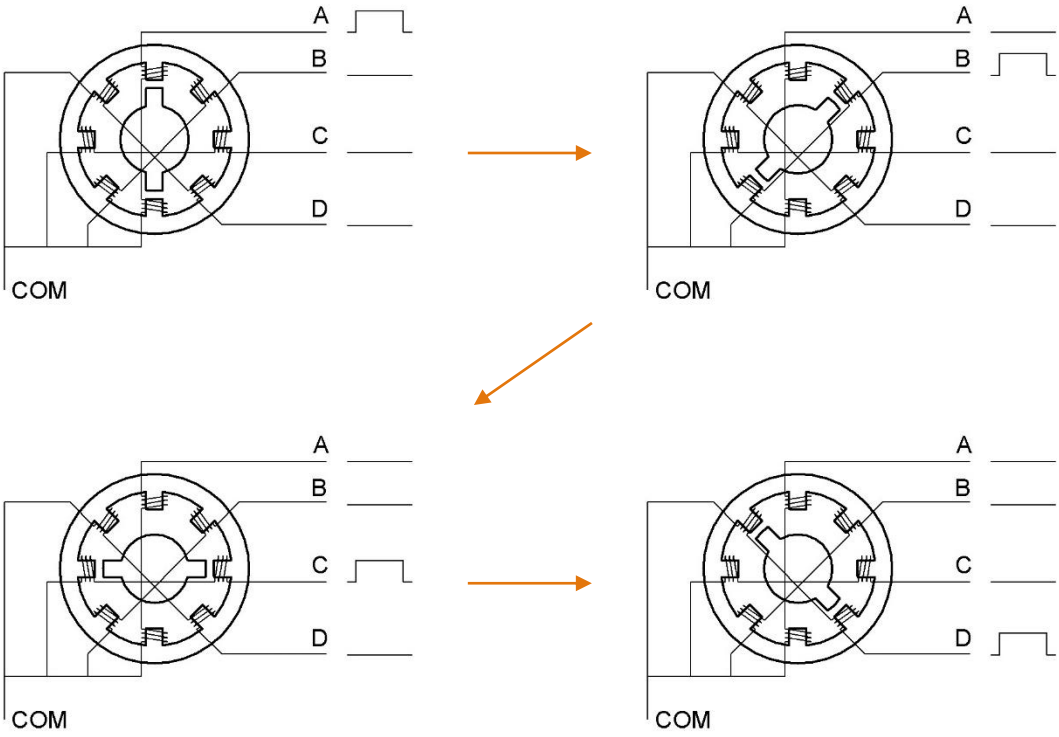


The schematic diagram of four-phase stepping motor is shown below:



The outside piece is the stator and the inside is the rotor of the motor. There are a certain number of coils, usually integer multiple of phases number, in the stator and when powered on, an electromagnet will be formed to attract a convex part (usually iron or permanent magnet) of the rotor. Therefore, the electric motor can be drived by conducting the coils on stator orderly.

A common driving process is as follows:



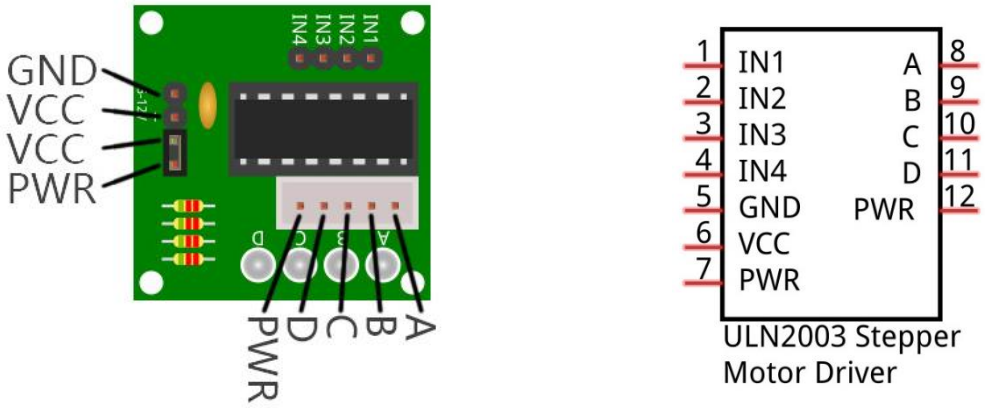
In the course above, the stepping motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise, the order of coil powered on is: A→B→C→D→A→…… . And the rotor will rotate in accordance with the order, step by step down, called four steps four pats. If the coils is powered on in the reverse order, D→C→B→A→D→… , the rotor will rotate in anti clockwise direction.

Stepping motor has other control methods, such as onnect A phase, then connect A B phase, the stator will be located in the middle of the A B, only a half step. This way can improve the stability of stepping motor, and reduce noise, the sequence of coil powered on is: A→AB→B→BC→C→CD→D→DA→A→……, the rotor will rotate in accordance with the order, a half step by a half step, called four step eight pat. Equally, if the coil is powered on in reverse order, the stepping motor will rotate in reverse rotation.

The stator of stepping motor we use has 32 magnetic poles, so a circle needs 32 steps. The output shaft of the stepping motor is connected with a reduction gear set, and the reduction ratio is 1/64. So the final output shaft rotates a circle requiring a 32*64=2048 step.

ULN2003 Stepping motor driver

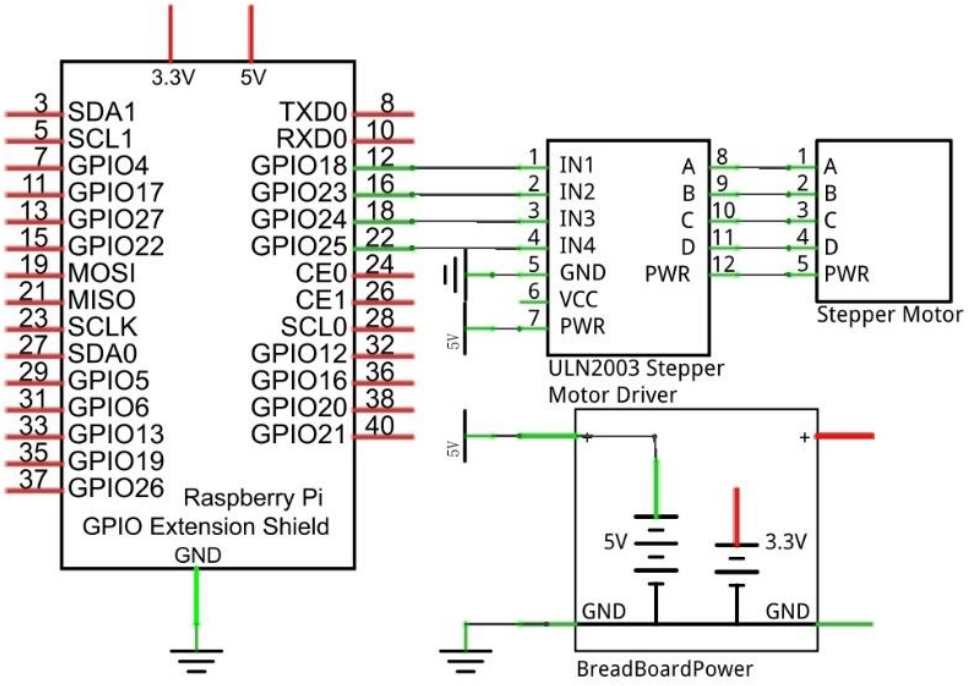
ULN2003 stepper motor driver is used to convert the weak signal into powerful control signal to drive the stepping motor. The input signal IN1-IN4 corresponds to the output signal A-D, and4 LED is integrated in the board to indicate the state of signals. The PWR interface can be used as apower supply for step motor. By default, PWR and VCC are connected by a short circuit.



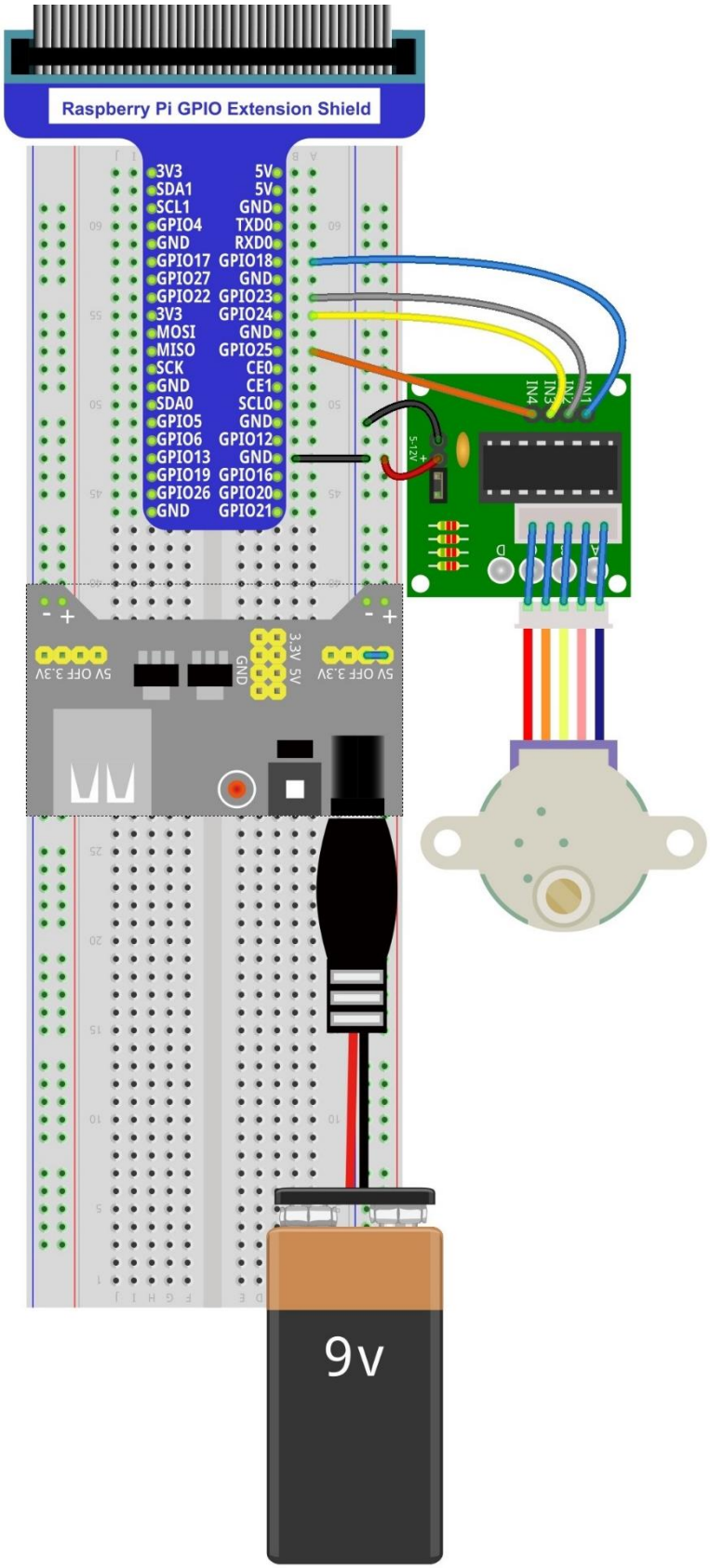
Circuit

When building the circuit, the rated voltage of the stepper motor is 5V, and use the breadboard power supply independently, and do not use the RPi power supply. Additionally, breadboard power supply needs to share Ground with RPi.

Schematic diagram



Hardware connection



Code

This code use four step four pat mode to drive the stepping motor forward and reverse direction.

C Code 16.1.1 SteppingMotor

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/16.1.1_SteppingMotor
```

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

```
gcc SteppingMotor.c –o SteppingMotor–lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./SteppingMotor
```

After the program is executed, the stepping motor will rotate 360° clockwise and then 360° anticlockwise, circularly.

The following is the program code:

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

const int motorPins[]={1,4,5,6};    //define pins connected to four phase ABCD of stepper motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for rotating anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};   //define power supply order for coil for rotating clockwise
//as for four phase stepping motor, four steps is a cycle. the function is used to drive the stepping motor clockwise or anticlockwise to take four steps
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order
        for (i=0;i<4;i++){  //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH : LOW);
            else        //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH : LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        printf("Step cycle!\n");
        if(ms<3)        //the delay can not be less than 3ms, otherwise it will exceed speed limit of the motor
            ms=3;
        delay(ms);
    }
}

```

```
//continuous rotation function, the parameter steps specifies the rotation cycles, every four steps is a cycle
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
void motorStop(){   //function used to stop rotating
    int i;
    for(i=0;i<4;i++){
        digitalWrite(motorPins[i],LOW);
    }
}
int main(void){
    int i;

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(i=0;i<4;i++){
        pinMode(motorPins[i],OUTPUT);
    }

    while(1){
        moveSteps(1,3,512);     //rotating  360°  clockwise, a total of 2048 steps in a circle, namely, 512 cycles.
        delay(500);
        moveSteps(0,3,512);     //rotating 360°  anticlockwise
        delay(500);
    }
    return 0;
}
```

In the code, define four pins of stepping motor and coil power supply order of four steps rotation mode.

```
const int motorPins[]={1,4,5,6};    //define pins connected to four phase ABCD of stepper motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for rotating anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};   //define power supply order for coil for rotating clockwise
```

Subfunction **moveOnePeriod** ((int dir,int ms) will drive the stepper motor rotating four step clockwise or anticlockwise, four step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will totate foreward, otherwise it rotates to reverse direction. Parameter "ms" indicates the time between each

two steps. The "ms" of steping motor used in this experiment is 3ms (the shortest time), less than 3ms will exceed the speed limit of stepper motor resulting in that motor can not rotate.

```
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order
        for (i=0;i<4;i++){  //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH : LOW);
            else        //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH : LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        printf("Step cycle!\n");
        if(ms<3)        //the delay can not be less than 3ms, otherwise it will exceed
speed limit of the motor
            ms=3;
        delay(ms);
    }
}
```

Subfunction **moveSteps** (int dir, int ms, int steps) is used to specific cycle number of stepping motor.

```
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
```

Sunbfunction **motorStop** () is used to stop the stepping motor.

```
void motorStop(){   //function used to stop rotating
    int i;
    for(i=0;i<4;i++){
        digitalWrite(motorPins[i],LOW);
    }
}
```

Finally, in the while cycle of main function, rotate one circle clockwise, and then one circle anticlockwise. According to the previous knowledge of the stepping motor, it can be known that the stepping motor rotation for one circle requires 2048 steps, that is, 2048/4=512 cycle.

```
    while(1){
        moveSteps(1,3,512);     //rotating   360°  clockwise, a total of 2048 steps in a
circle, namely, this function(four steps) will be called 512 times.
        delay(500);
        moveSteps(0,3,512);     //rotating 360°  anticlockwise
        delay(500);
    }
```

Pyhton Code 16.1.1 SteppingMotor

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/16.1.1_ SteppingMotor
```

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

```
python SteppingMotor.py
```

After the program is executed, the stepping motor will rotate 360° clockwise and then 360° anticlockwise, circularly.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

motorPins = (12, 16, 18, 22)    #define pins connected to four phase ABCD of stepper
motor
CCWStep = (0x01,0x02,0x04,0x08) #define power supply order for coil for rotating
anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  #define power supply order for coil for rotating
clockwise

def setup():
    print 'Program is starting...'
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    for pin in motorPins:
        GPIO.setup(pin,GPIO.OUT)
#as for four phase stepping motor, four steps is a cycle. the function is used to drive
the stepping motor clockwise or anticlockwise to take four steps
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      #cycle for power supply order
        for i in range(0,4,1):  #assign to each pin, a total of 4 pins
            if (direction == 1):#power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH orGPIO.LOW))
            else :              #power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        if(ms<3):       #the delay can not be less than 3ms, otherwise it will exceed
speed limit of the motor
            ms = 3
        time.sleep(ms*0.001)
#continuous rotation function, the parameter steps specifies the rotation cycles, every
four steps is a cycle
def moveSteps(direction, ms, steps):
    for i in range(steps):
        moveOnePeriod(direction, ms)
#function used to stop rotating
def motorStop():
```

```
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)

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

def destroy():
    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()
```

In the code, define four pins of stepping motor and coil power supply order of four steps rotation mode.

```
motorPins = (12, 16, 18, 22)    #define pins connected to four phase ABCD of stepper
motor

CCWStep = (0x01,0x02,0x04,0x08) #define power supply order for coil for rotating
anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  #define power supply order for coil for rotating
clockwise
```

Subfunction **moveOnePeriod** (direction, ms) will drive the stepper motor rotating four step clockwise or anticlockwise, four step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will totate foreward, otherwise it rotates to reverse direction. Parameter "ms" indicates the time between each two steps. The "ms" of steping motor used in this experiment is 3ms (the shortest time), less than 3ms will exceed the speed limit of stepper motor resulting in that motor can not rotate.

```
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      #cycle for power supply order
        for i in range(0,4,1):  #assign to each pin, a total of 4 pins
            if (direction == 1):#power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH orGPIO.LOW))
            else :              #power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        if(ms<3):       #the delay can not be less than 3ms, otherwise it will exceed
speed limit of the motor
            ms = 3
```

```
        time.sleep(ms*0.001)
```

Subfunction **moveSteps** (direction, ms, steps) is used to specific cycle number of stepping motor.

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

Sunbfunction **motorStop** () is used to stop the stepping motor.

```
def motorStop():
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)
```

Finally, in the while cycle of main function, rotate one circle clockwise, and then one circle anticlockwise. According to the previous knowledge of the stepping motor, it can be known that the stepping motor rotation for one circle requires 2048 steps, that is, 2048/4=512 cycle.

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

Chapter 17 74HC595 & LEDBar Graph

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

Project 17.1 Flowing Water Light

Now let's learn how to use 74HC595 to make a flowing water light with less GPIO.

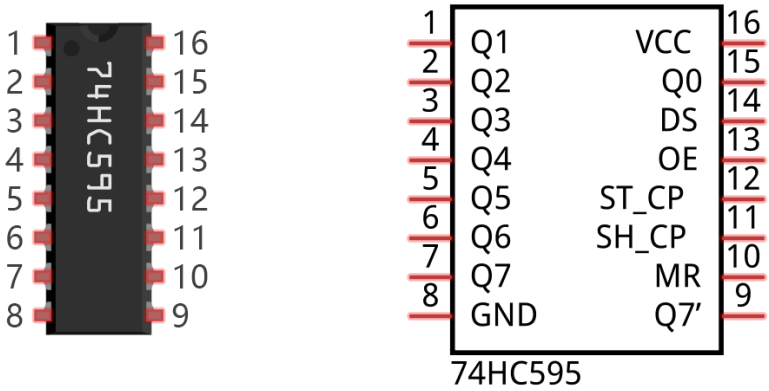
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x17
  </td></tr>
<tr><td>74HC595 x1
  </td><td>LEDBar Graph x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Component knowledge

74HC595

74HC595 chip is used to convert serial data into parallel data. 74HC595 can convert the serial data of one byte to 8 bits, and send its corresponding level to the corresponding 8 ports. With this feature, 74HC595 can be used to expand the IO port of Arduino board. At least 3 ports on the RPI board are need to control 8 ports of 74HC595.



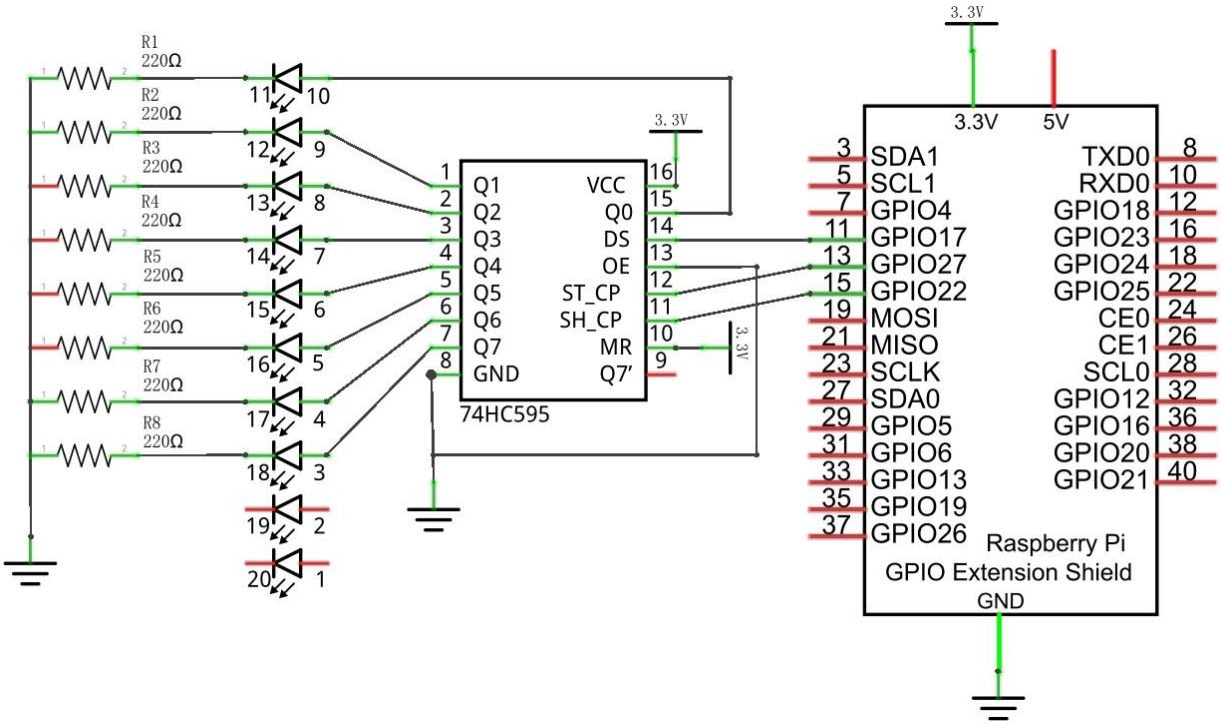
The ports of 74HC595 are described as follows:

Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel update output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial shift clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove shift register:When this pin is in low level, the content in shift register will be cleared .
Q7'	9	Serial data output: it can be connected to more 74HC595 in series.

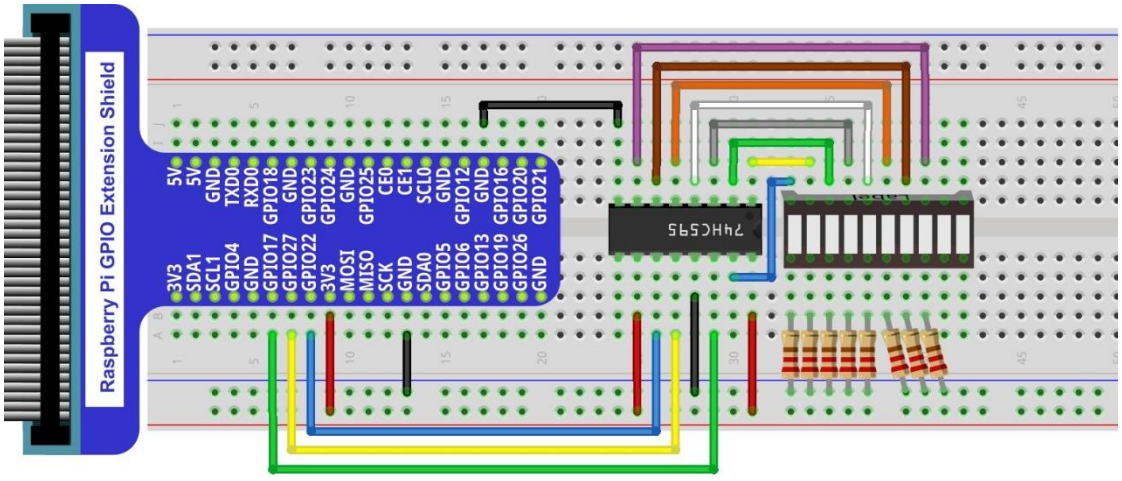
For more detail, please refer to the dataheet.

Circuit

Schematic diagram



Hardware connection



Code

In this experiment, make a flowing water light with 74HC595 to learn its usage.

C Code 17.1.1 LightWater02

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 17.1.1_LightWater02 directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/17.1.1_LightWater02
```

2. Use following command to compile "LightWater02.c" and generate executable file "LightWater02".

```
gcc LightWater02.c –o LightWater02 –lwiringPi
```

3. Then run the generated file "LightWater02".

```
sudo ./LightWater02
```

After the program is executed, LEDBar Graph begin to display flowing water light from left to right, then from right to left.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)

int main(void)
{
    int i;
    unsigned char x;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        x=0x01;
        for(i=0;i<8;i++){
            digitalWrite(latchPin,LOW);     // Output low level to latchPin
            shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
            digitalWrite(latchPin,HIGH); // Output high level to latchPin, and 74HC595
will update the data to the parallel output port.
            x<<=1; // make the variable move one bit to left once, then the bright LED
move one step to the left once.
            delay(100);
```

```
30          }
31          x=0x80;
32          for(i=0;i<8;i++){
33              digitalWrite(latchPin,LOW);
34              shiftOut(dataPin,clockPin,LSBFIRST,x);
35              digitalWrite(latchPin,HIGH);
36              x>>=1;
37              delay(100);
38          }
39      }
40      return 0;
41  }
42
```

In the code, we configure three pins to control the 74HC595. And define a one-byte variable to control the state of 8 LEDs through the 8 bits of the variable. The LED lights on when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED on.

```
x=0x01;
```

In the “while” cycle of main function, use “for” cycle to send x to 74HC595 output pin to control the LED. In “for” cycle, x will be shift one bit to left in one cycle, then in the next round when data of x is sent to 74HC595, the LED turned on will move one bit to left once.

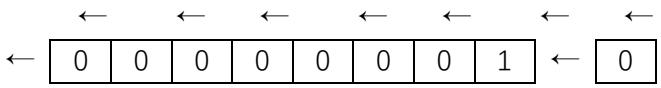
```
        for(i=0;i<8;i++){
            digitalWrite(latchPin,LOW);     // Output low level to latchPin
            shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
            digitalWrite(latchPin,HIGH); // Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
            x<<=1; // make the variable move one bit to left once, then the bright LED move one step to the left once.
            delay(100);
        }
```

In second “for” cycle, the situation is the same. The difference is that x is shift from 0x80 to right in order.

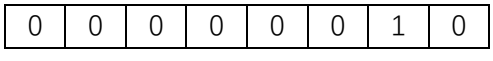
<< operator

"<<" is the left shift operator, which can make all bits of 1 byte shift by several bits to the left (high) direction and add 0 on the right (low). For example, shift binary 00000001 by 1 bit to left:

byte x = 1 << 1;

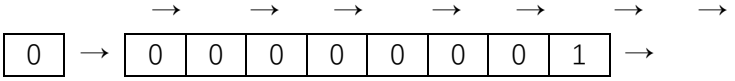


The result of x is 2 (binary 00000010)。



Thre is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

byte x = 1 >> 1;



The result of x is 0（00000000）。

0	0	0	0	0	0	0	0

X <<= 1 is equivalent to x = x << 1 and x >>= 1 is equivalent to x = x >> 1

About shift function：

uint8_t shiftIn (uint8_t dPin, uint8_t cPin, uint8_t order) ;
This shifts an 8-bit data value in with the data appearing on the dPin and the clock being sent out on the cPin. Order is either LSBFIRST or MSBFIRST. The data is sampled after the cPin goes high. (So cPin high, sample data, cPin low, repeat for 8 bits) The 8-bit value is returned by the function.
void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val) ;
The shifts an 8-bit data value val out with the data being sent out on dPin and the clock being sent out on the cPin. order is as above. Data is clocked out on the rising or falling edge – ie. dPin is set, then cPin is taken high then low – repeated for the 8 bits.

For more details about shift function, please refer to:http://wiringpi.com/reference/shift-library/

Pyhton Code 17.1.1 LightWater02

First observe the experimental phenomenon, and then analyze the code.

1. Use the cd command to enter 17.1.1_LightWater02 directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/17.1.1_LightWater02
```

2. Use python command to execute python code “LightWater02.py”.

```
python LightWater02.py
```

After the program is executed, LEDBar Graph begin to display flowing water light from left to right, then from right to left.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3  # Defines the data bit that is transmitted preferentially in the shiftOut function.
4  LSBFIRST = 1
5  MSBFIRST = 2
6  #define the pins connect to 74HC595
7  dataPin   = 11      #DS Pin of 74HC595(Pin14)
8  latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
9  clockPin = 15       #SH_CP Pin of 74HC595(Pin11)
10
11 def setup():
12     GPIO.setmode(GPIO.BOARD)     # Number GPIOs by its physical location
13     GPIO.setup(dataPin, GPIO.OUT)
14     GPIO.setup(latchPin, GPIO.OUT)
15     GPIO.setup(clockPin, GPIO.OUT)
16 # shiftOut function, use bit serial transmission.
17 def shiftOut(dPin,cPin,order,val):
18     for i in range(0,8):
19         GPIO.output(cPin,GPIO.LOW);
20         if(order == LSBFIRST):
21             GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
```

```
        elif(order == MSBFIRST):
            GPIO.output(dPin, (0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

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

def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
    GPIO.cleanup()

if __name__ == '__main__': # Program starting from here
    print 'Program is starting...'
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

In the code, we define a shiftOut() function, which is used to output val with bit in order. And where the dPin for the data pin, cPin for the clock amd order for the priority bit flag (high or low). This function conforms to the operation mode of 74HC595.

```
def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin, (0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin, (0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);
```

In the loop () function, we use two "for" cycle to achieve the target. First, define a variable x=0x01, binary 00000001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then a LED is turned on. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED turned on will be shifted. Repeat the operation, the effect of flowing water light will be formed. If the direction of the shift operation for x is different, the flowing direction is different.

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

Chapter 18 74HC595 & 7-segment display.

In this chapter, we will learn a new component, 7-segment display.

Project 18.1 7-segment display.

We will use 74HC595 to control 7-segment display. and make it display sixteen decimal character "0-F".

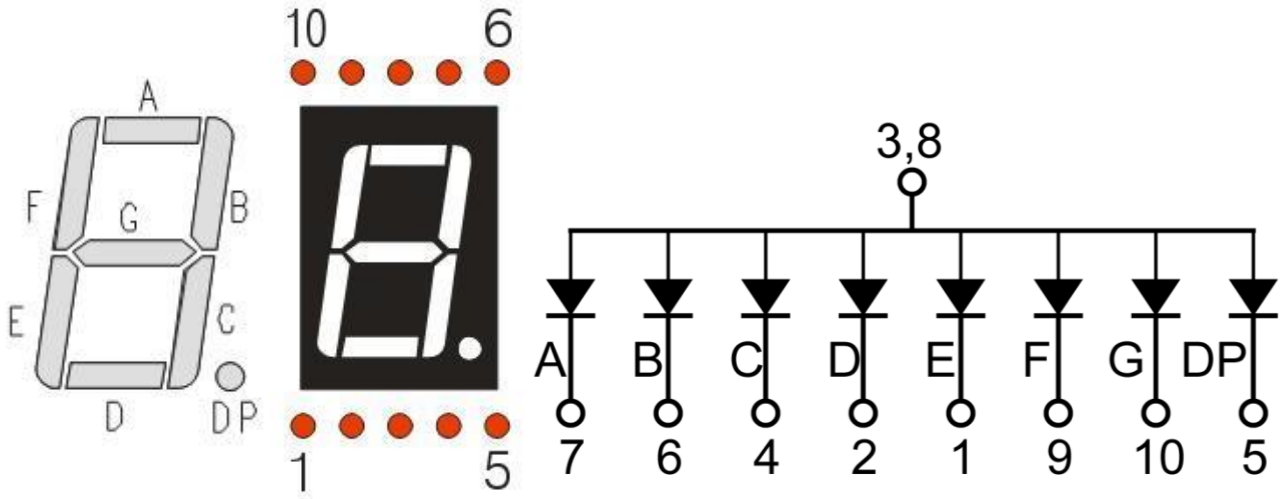
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x18
  </td></tr>
<tr><td>74HC595 x1
  </td><td>7-segment display x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Component knowledge

7-segment display

7-segment display is a digital electronic display device. There is a figure of "8" and a decimal point, which consist of 8 LED. According to the difference about common cathode and anode. its internal structure and pins diagram is shown below:

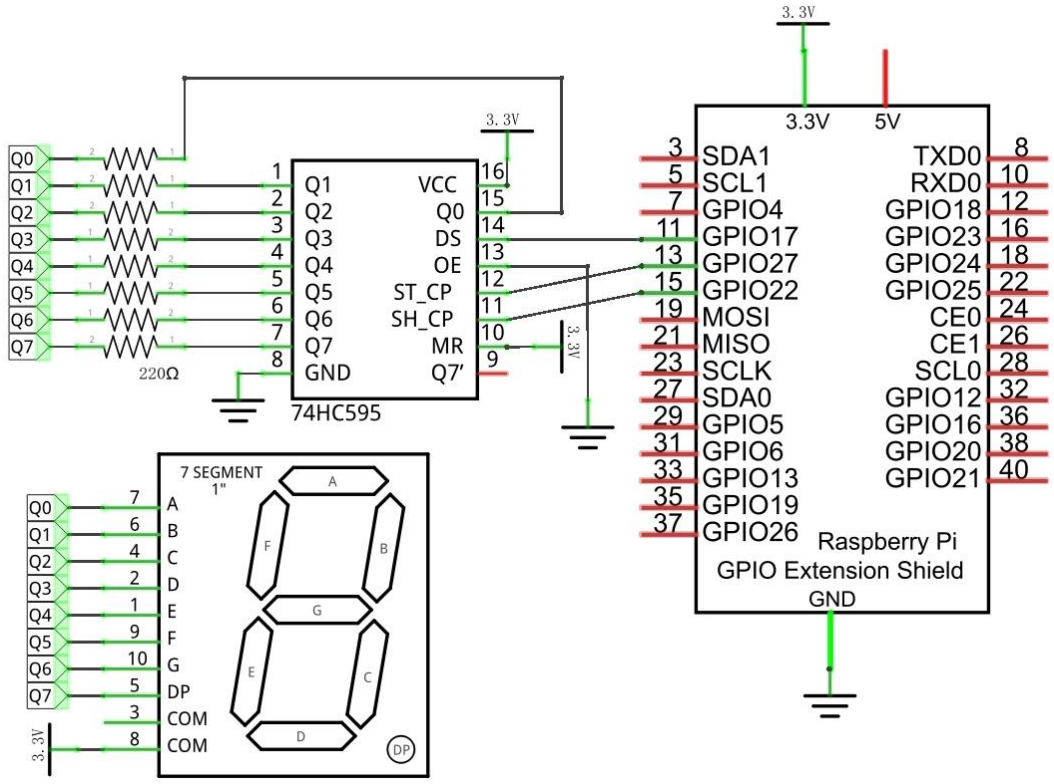


As is known from the above circuit diagram that we can control the state of each LED separately. So, through combinating LED with different state, we can display different numbers. For example, display figure 0: we need to turn on LED segment A, B, C, D, E, F, and turn off LED segment G and DP.

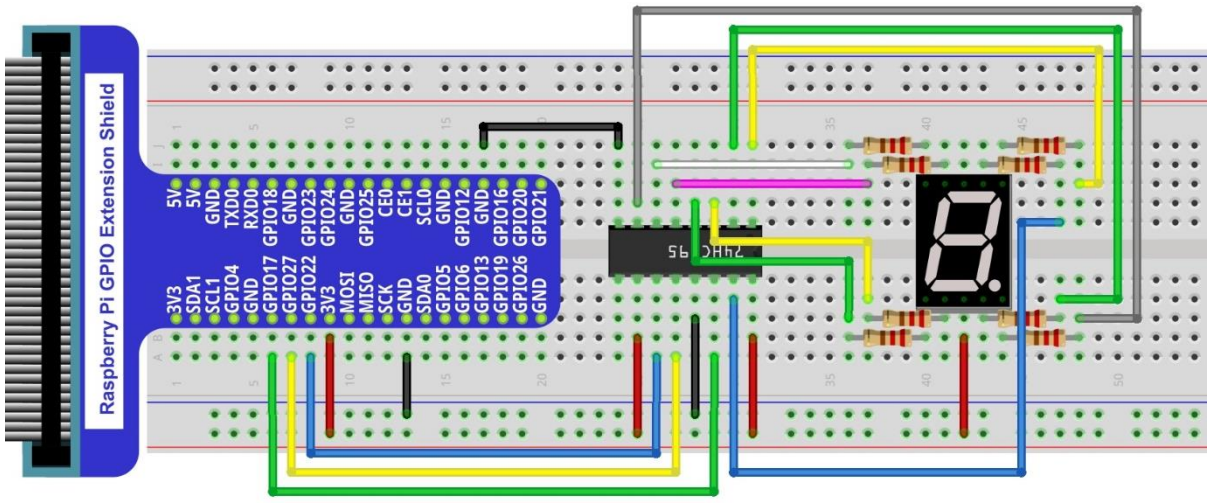
In this experiment, we use a display 7-segment (common anode). Therefore, when the input low level to a LED segment, the LED will be turned on. Define segment “A” as the lowest level, the segment “DP” as the highest level, that is, from high to low: “DP”, “G”, “F”, “E”, “D”, “C”, “B”, “A”. And character "0" corresponds to the code: 1100 0000b=0xc0.

Circuit

Schematic diagram



Hardware connection



Code

In this code, uses 74HC595 to control the 7-segment display. The usage of 74HC595 is generally the same to last section. The content 74HC595 outputs is different. We need code character “0”- “F” one by one, and then output them with 74HC595.

C Code 18.1.1 SevenSegmentDisplay

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/18.1.1_SevenSegmentDisplay
```

2. Use following command to compile “SevenSegmentDisplay.c” and generate executable file “SevenSegmentDisplay”.

```
gcc SevenSegmentDisplay.c –o SevenSegmentDisplay –lwiringPi
```

3. Then run the generated file “SevenSegmentDisplay”.

```
sudo ./SevenSegmentDisplay
```

After the program is executed, SevenSegmentDisplay starts to display the character “0”- “F” successively.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)
// encoding for character 0-F of common anode SevenSegmentDisplay.
unsigned char
num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e};

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the
highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
```

```
28              delay(500);
29          }
30          for(i=0;i<sizeof(num);i++){
31              digitalWrite(latchPin,LOW);
32              shiftOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);// Use the "&0x7f" to
33  display the decimal point.
34              digitalWrite(latchPin,HIGH);
35              delay(500);
36          }
37      }
38      return 0;
39  }
```

First, put encoding of "0"-"F" into the array.

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

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

```
for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the
highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
```

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

```
shiftOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);
```

Pyhton Code 18.1.1 SevenSegmentDisplay

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/18.1.1_SevenSegmentDisplay
```

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

```
python SevenSegmentDisplay.py
```

After the program is executed, SevenSegmentDisplay starts to display the character "0"- "F" successively.

The following is the program code:

```
1   import RPi.GPIO as GPIO
2   import time
3
4   LSBFIRST = 1
5   MSBFIRST = 2
6   #define the pins connect to 74HC595
7   dataPin   = 11      #DS Pin of 74HC595(Pin14)
```

```
latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
clockPin = 15       #SH_CP Pin of 74HC595(Pin11)
#SevenSegmentDisplay display the character “0”- “F” successively
 num = [0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e]
def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])# Output the figures and the
highest level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i]&0x7f)#Use “&0x7f” to display the
decimal point.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)

def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
    GPIO.cleanup()

if __name__ == '__main__': # Program starting from here
    print 'Program is starting...'
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

First, put encoding of “0”-“F” into the array.

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

In the “for” cycle of loop() function, use 74HC595 to output contents of array “num” successively. SevenSegmentDisplay can correctly display the corresponding characters. Pay attention to that in shiftOut function, the transmission bit, flag bit highest bit will be transmitted preferentially.

```
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])#Output the figures and the highest level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
```

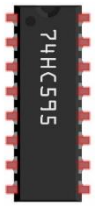
If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

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

Project 18.2 4-Digit 7-segment display

Now, let’s try to control more Digit 7-segment display

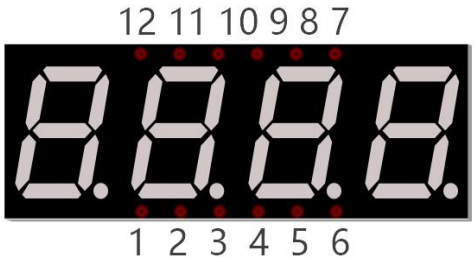
Component List

Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1		Jumper M/M x27 		
74HC595 x1 	PNP transistor x4 	4-Digit 7-segment display x1 	Resistor 220Ω x8 	Resistor 1KΩ x4 

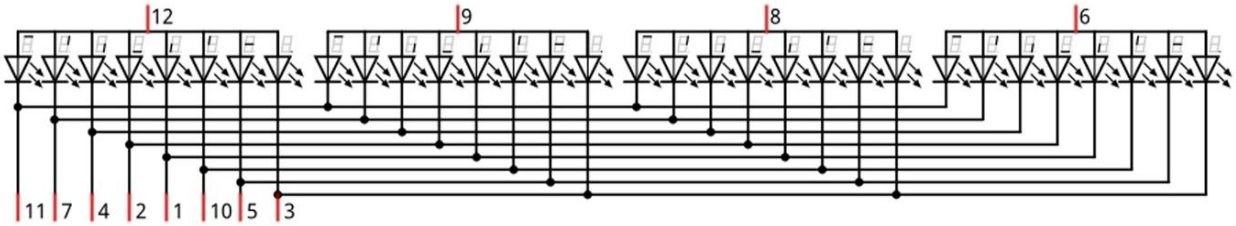
Component knowledge

4 Digit 7-Segment Display

4 Digit 7-segment display integrates four 7-segment display, so it can display more numbers. According to the difference about common cathode and anode. its internal structure and pins diagram is shown below:



The internal circuit is shown below, and all 8 LED cathode pins of each 7-segment display are connected together.

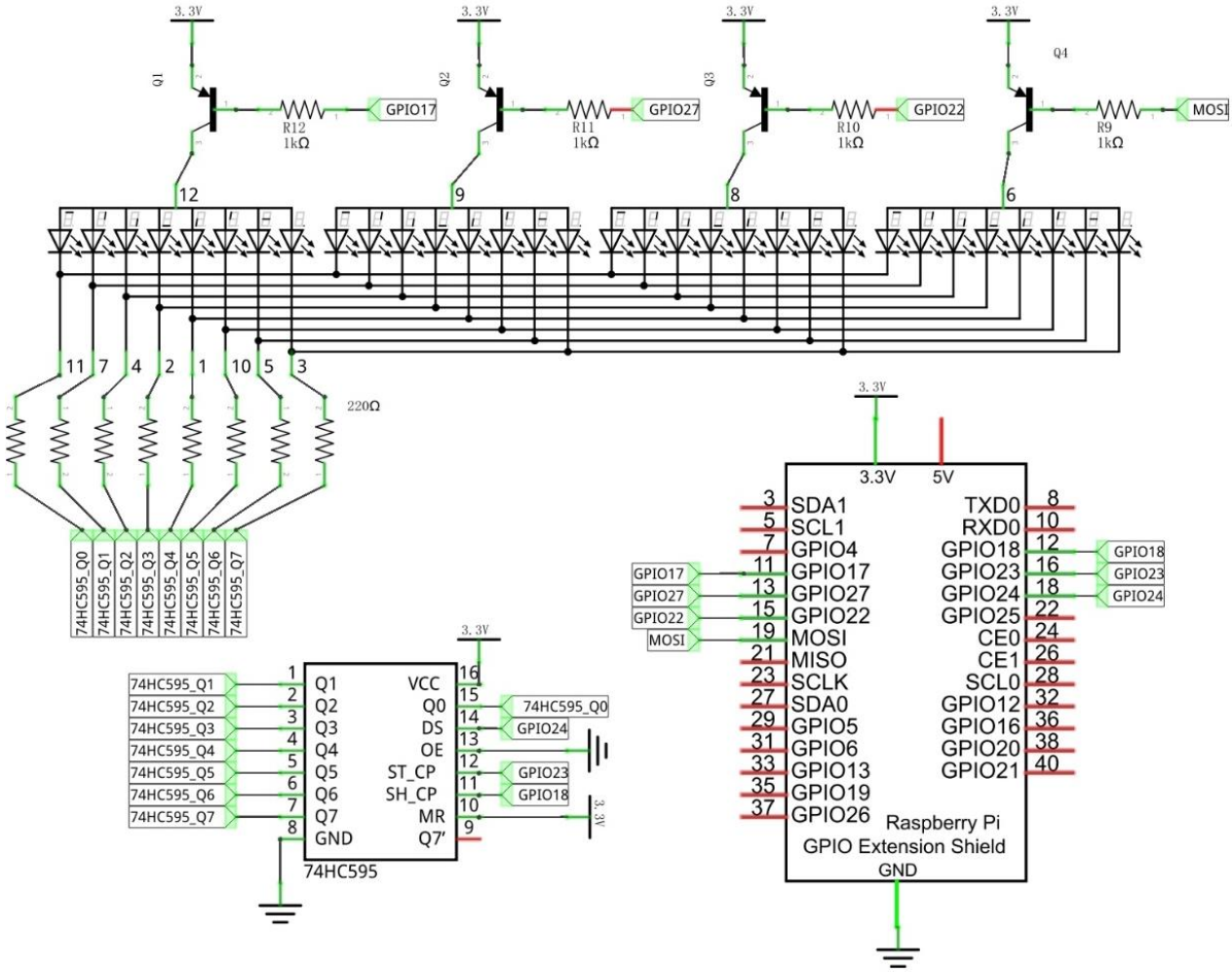


Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that 4-Digit display in turn, one by one, not together. First send high level to common end of the first tube, and send low level to the rest of the three common end, and then send content to 8 LED cathode pins of the first tube. At this time, the first 7-segment display will display content and the rest three one in closed state.

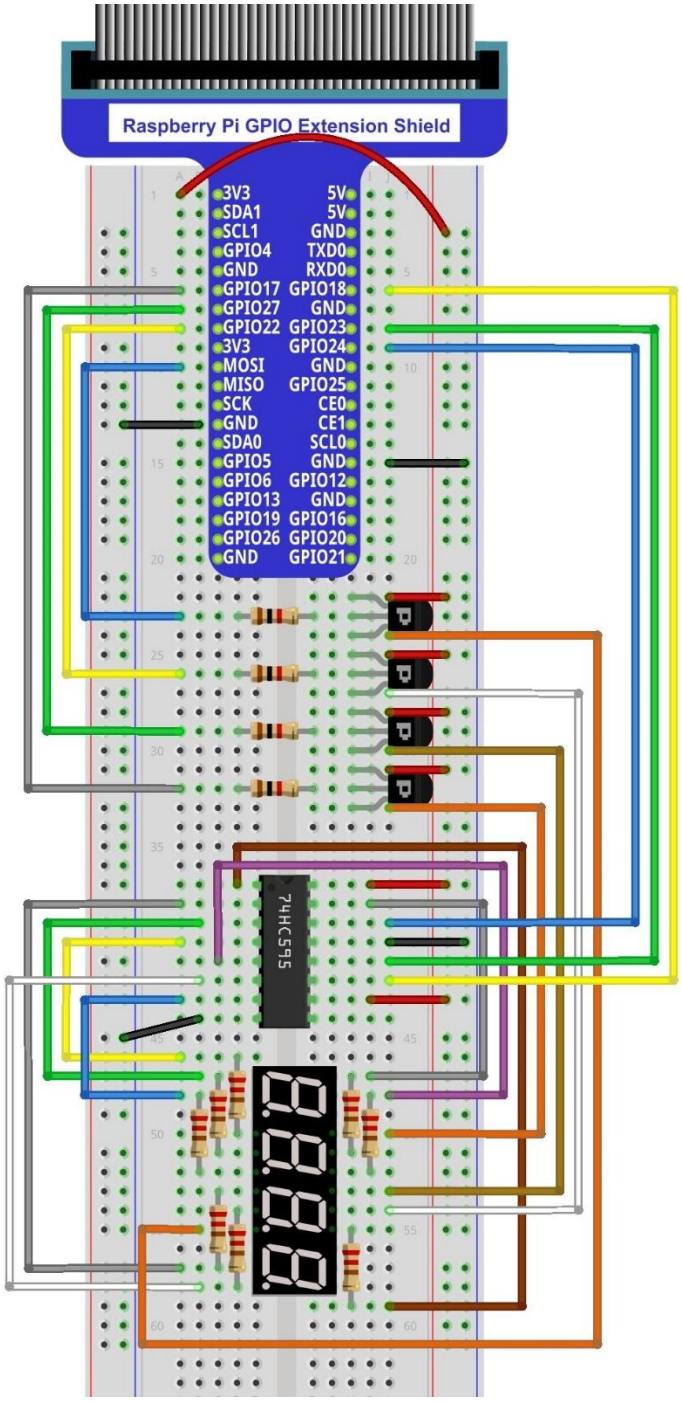
Similarly, the second, third, fourth 7-segment display display the content in turn, namely, scan display. Although the four numbers are displayed in turn separately, but this process is very fast, and due to the optical afterglow effect and people in vision persistence effect, we can see all 4 numbers at the same time. On the contrary, if each figure is displayed for a long time, you can see that the numbers are displayed separately.

Circuit

Schematic diagram



Hardware connection



Code

In this code, we use 74HC595 to control 4-Digit 7-segment display, and use dynamic scanning way to show the changing numbers.

C Code 18.2.1 StopWatch

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/18.2.1_StopWatch
```

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

```
gcc StopWatch.c –o StopWatch –lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./StopWatch
```

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

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define     dataPin     5   //DS Pin of 74HC595(Pin14)
#define     latchPin    4   //ST_CP Pin of 74HC595(Pin12)
#define     clockPin    1    //CH_CP Pin of 74HC595(Pin11)
const int digitPin[]={0,2,3,12};        // Define 7-segment display common pin
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;    //variable counter,the number will be displayed by 7-segment display
//Open one of the 7-segment display and close the remaining three, the parameter digit is
optional for 1,2,4,8
void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) == 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) == 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) == 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) == 0x01) ? LOW : HIGH);
}
void outData(int8_t data){      //function used to output data for 74HC595 输出数据函数
    digitalWrite(latchPin,LOW);
    shiftOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
void display(int dec){  //display function for 7-segment display
    selectDigit(0x01);      //select the first, and display the single digit
```

```
    outData(num[dec%10]);
    delay(1);               //display duration
    selectDigit(0x02);      //select the second, and display the tens digit
    outData(num[dec%100/10]);
    delay(1);
    selectDigit(0x04);      //select the third, and display the hundreds digit
    outData(num[dec%1000/100]);
    delay(1);
    selectDigit(0x08);      //select the fourth, and display the thousands digit
    outData(num[dec%10000/1000]);
    delay(1);
}
void timer(int  sig){       //Timer function
    if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus 1, and
update the number displayed by 7-segment display
        counter ++;
        alarm(1);           //set the next timer time
        printf("counter : %d \n",counter);
    }
}
int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);        //set the pin connected to74HC595 for output mode
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    //set the pin connected to 7-segment display common end to output mode

    for(i=0;i<4;i++){
        pinMode(digitPin[i],OUTPUT);
        digitalWrite(digitPin[i],LOW);
    }
    signal(SIGALRM,timer);  //configure the timer
    alarm(1);               //set the time of timer to 1s
    while(1){
        display(counter);   //display the number counter
    }
    return 0;
}
```

First, define the pin of 74HC595 and 7-segment display common end, character encoding and a variable

"counter" to be displayed counter.

```
#define     dataPin     5   //DS Pin of 74HC595(Pin14)
#define     latchPin    4   //ST_CP Pin of 74HC595(Pin12)
#define     clockPin    1    //CH_CP Pin of 74HC595(Pin11)
const int digitPin[]={12,3,2,0};  //Define the pin of 7-segment display common end
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;    //variable counter, the number will be dislayed by 7-segment display
```

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

```
void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) == 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) == 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) == 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) == 0x01) ? LOW : HIGH);
}
```

The subfunction **outData** (int8_t data) is used to make the 74HC595 output a 8-bit data immediately.

```
void outData(int8_t data){      // function used to output data for 74HC595 输出数据函数
    digitalWrite(latchPin,LOW);
    shiftOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
```

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

```
void display(int dec){  //display function for 7-segment display
    selectDigit(0x01);      //select the first, and display the single digit
    outData(num[dec%10]);
    delay(1);               //display duration
    selectDigit(0x02);      //Select the second, and display the tens digit
    outData(num[dec%100/10]);
    delay(1);
    selectDigit(0x04);      //Select the third, and display the hundreds digit
    outData(num[dec%1000/100]);
    delay(1);
    selectDigit(0x08);      //Select the fourth, and display the thousands digit
    outData(num[dec%10000/1000]);
    delay(1);
}
```

Subfunction **timer** (int sig) is the timer function, wich will set a alarm signal. This function wil be ececuted once

at set intervals. Accompanied by the execution, the variable counter will be added 1, and then reset the time of timer to 1s.

```
void timer(int  sig){       //timer function
    if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus 1, and update the number displayed by 7-segment display
        counter ++;
        alarm(1);           //set the next timer time
    }
}
```

Finally, in the main function, configure all the GPIO, and set the timer function.

```
    pinMode(dataPin,OUTPUT);        //set the pin connected to74HC595 for output mode
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    //set the pin connected to 7-segment display common end to output mode
    for(i=0;i<4;i++){
        pinMode(digitPin[i],OUTPUT);
        digitalWrite(digitPin[i],LOW);
    }
    signal(SIGALRM,timer);  //configure the timer
    alarm(1);               //set the time of timer to 1s
```

In the while cycle, make the digital display variable counter value. The value will change in function timer (), so the content displaied by 7-segment display will change accordingly.

```
    while(1){
        display(counter);   //display number counter
    }
```

Pyhton Code 18.2.1 StopWatch

This code use four step four pat mode to drive the stepping motor forward and reverse direction.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/18.2.1_StopWatch
```

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

```
python StopWatch.py
```

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

The following is the program code:

```
import RPi.GPIO as GPIO
import time
import threading

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 18      #DS Pin of 74HC595(Pin14)
latchPin  = 16      #ST_CP Pin of 74HC595(Pin12)
clockPin = 12       #SH_CP Pin of 74HC595(Pin11)
num = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
digitPin = (19,15,13,11)    # Define the pin of 7-segment display common end
counter = 0         # Variable counter, the number will be dislayed by 7-segment display
t = 0               # defien the Timer object
def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)       # Set pin mode to output
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)
    for pin in digitPin:
        GPIO.setup(pin,GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH)

def outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
```

```
    shiftOut(dataPin,clockPin,MSBFIRST,data)
    GPIO.output(latchPin,GPIO.HIGH)

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

def display(dec):   #display function for 7-segment display
    outData(0xff)   #eliminate residual display
    selectDigit(0x01)   #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.003)   #display duration
    outData(0xff)
    selectDigit(0x02)   # Select the second, and display the tens digit
    outData(num[dec%100/10])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x04)   # Select the third, and display the hundreds digit
    outData(num[dec%1000/100])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x08)   # Select the fourth, and display the thousands digit
    outData(num[dec%10000/1000])
    time.sleep(0.003)
def timer():        #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                           #Start timing
    counter+=1
    print "counter : %d"%counter

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

def destroy():   # When "Ctrl+C" is pressed, the function is executed.
```

```
78      global t
79      GPIO.cleanup()
80      t.cancel()      #cancel the timer
81
82  if __name__ == '__main__': # Program starting from here
83      print 'Program is starting...'
84      setup()
85      try:
86          loop()
87      except KeyboardInterrupt:
89          destroy()
```

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

```
dataPin   = 18      #DS Pin of 74HC595(Pin14)
latchPin  = 16      #ST_CP Pin of 74HC595(Pin12)
clockPin = 12       #CH_CP Pin of 74HC595(Pin11)
num = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
digitPin = (19,15,13,11)    # Define the pin of 7-segment display common end
counter = 0         # Variable counter, the number will be dislayed by 7-segment display
```

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

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

The subfunction **outData** (data) is used to make the 74HC595 output a 8-bit data immediately.

```
def outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
    shiftOut(dataPin,clockPin,MSBFIRST,data)
    GPIO.output(latchPin,GPIO.HIGH)
```

Subfunction **display** (dec) is used to make 4-Digit 7-segment display display a 4-bit integer. First open the common end of first 7-segment display and close to the other three, at this time, it can be used as 1-Digit 7-segment display. The first is used for displaying single digit of "dec", the second for tens digit, third for hundreds digit and fourth for thousands digit respectively. Each digit will be displaied for a period of time through using **delay** (). The time in this code is set very short, so you will see different digit is in a mess. If the time is set long enough, you will see that every digit is display independent.

```
def display(dec):   #display function for 7-segment display
    outData(0xff)   #eliminate residual display
    selectDigit(0x01)   #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.003)   #display duration
```

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

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

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

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

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

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

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

```
def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
    global t
    GPIO.cleanup()
    t.cancel()      # cancel the timer
```

Chapter 19 74HC595 & LED Matrix

We have learned how to use 74HC595 to control LEDBar Graph and Seven-SegmentDisplay. And we will continue to use the 74HC595 to control more LED, LEDMatrix.

Project 19.1 LED Matrix

In this experiment, we will use two 74HC595 to control a monochrome LEDMatrix (8*8) to make it display some graphics and characters.

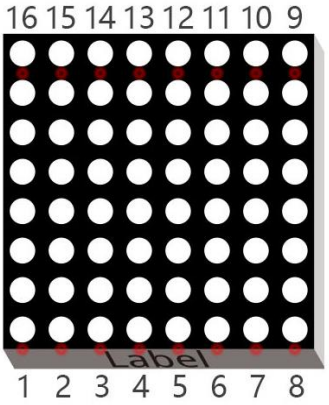
Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x41
  </td></tr>
<tr><td>74HC595 x2
  </td><td>8*8 LEDMatrix x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Componet knowledge

LED matrix

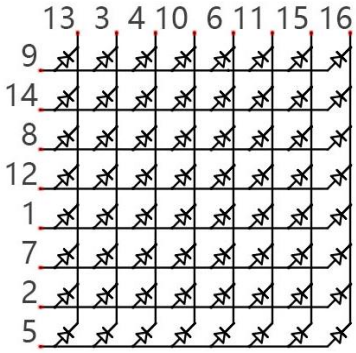
LED matrix is a rectangular display module taht consists of several LEDs. The following is an 8*8 monochrome LED matrix with 64 LEDs (8 rows and 8 columns).



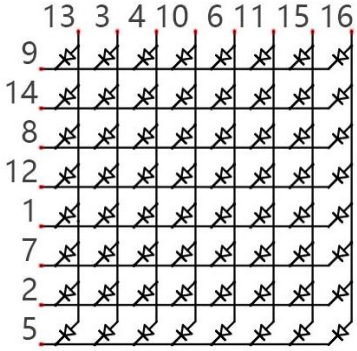
In order to facilitate the operation and save the ports, positive pole of LEDs in each row and negative pole of LEDs in each column are respectively connected together inside LED matrix module, which is called Common Anode. There is another form. Negative pole of LEDs in each row and positive pole of LEDs in each column are respectively connected together, which is called Common Cathode.

The one we use in this experiment is a common anode LEDMatrix.

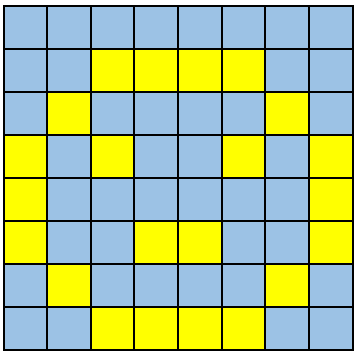
Connection mode of common anode



Connection mode of common cathode



Let us learn how connection mode of common anode works. Choose 16 ports on RPI board to connect to the 16 ports of LED Matrix. Configured one port in columns for low level, which make the column of the port selected. Then configure the eight ports in row to display content in the selected column. Delay for a moment. And then select the next column and outputs the corresponding content. This kind of operation to column is called scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.



1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

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

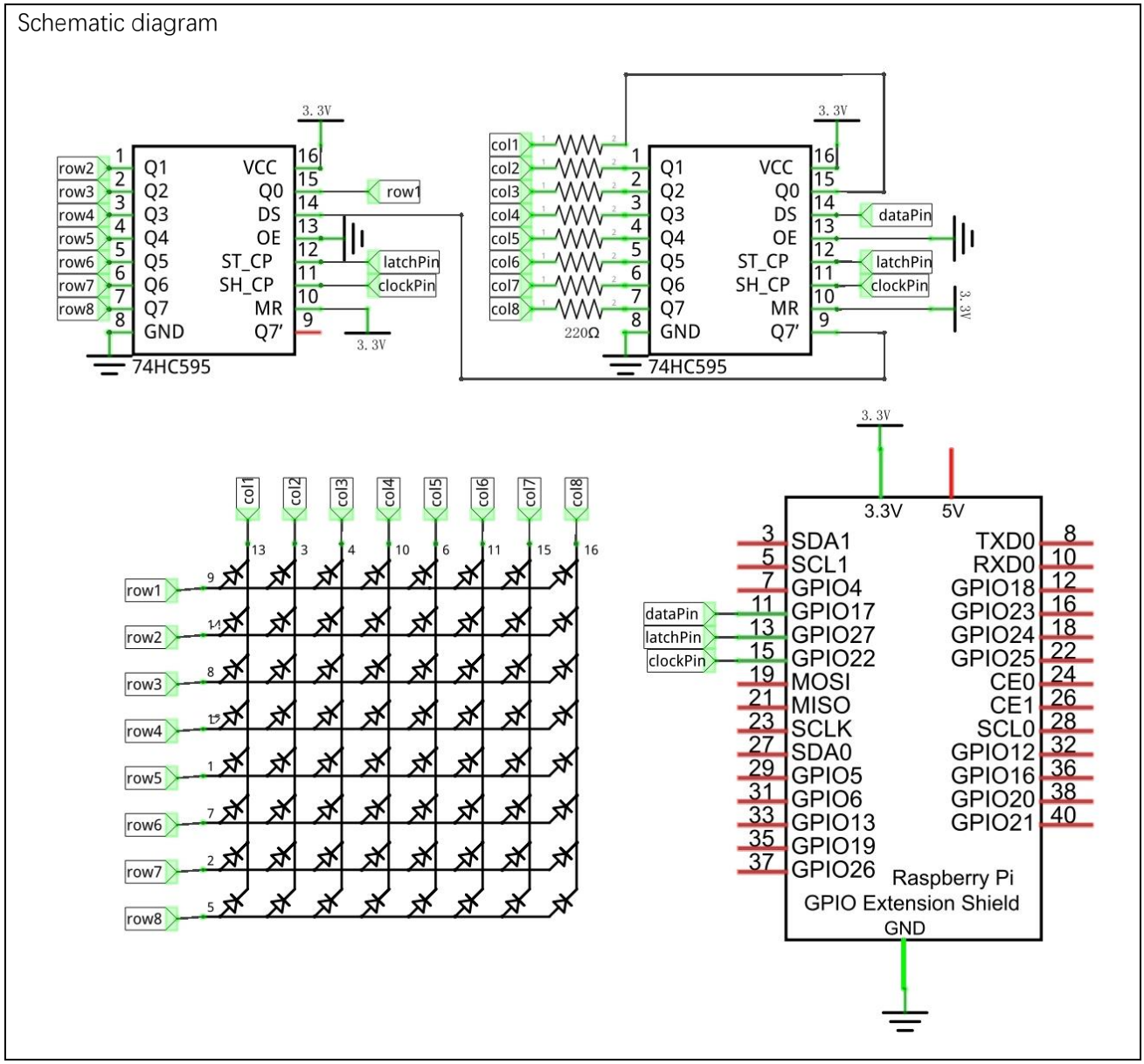
First, display the first column, then turn off the first column and display the second column..... turn off the seventh column and display the 8th column, and then start from the first column again like the control of Graph LEDBar. The whole prograss will be repeated rapidly and circularly. Due to afterglow effect of LED and visual residual effect of human eyes, we will see a picture of a smiling face directly rather than LED are turned on one column by one column (although in fact it is the real situation).

Scaning rows is another display way of dot matrix. Whether scanning line or column, 16 GPIO are required. In order to save GPIO of control board, two 74HC595 is used. Every piece of 74HC595 has eight parallel output ports, so two pieces has 16 ports in total, just enough. The control line and data line of two 74HC595 are not all connected to the RPi, but connect Q7 pin of first stage 74HC595 to data pin of second one, namely, two 74HC595 are connected in series. It is the same to using one "74HC595" with 16 parallel output ports.

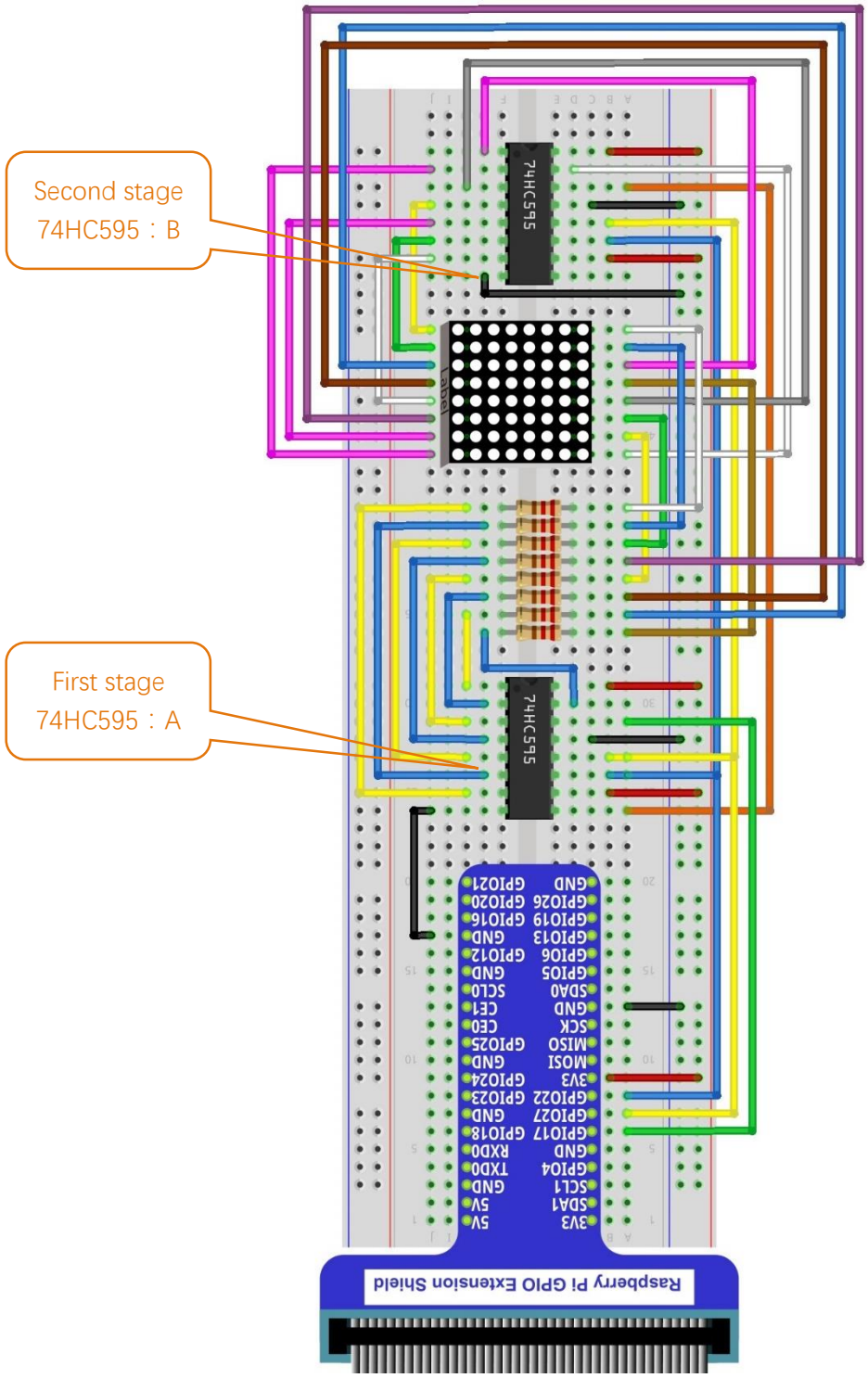
Circuit

In this experimental circuit, the power pin of 74HC595 is connected to 3.3V. It can also be connected to 5V to make LEDMatrix brighter.

Schematic diagram



Hardware connection



Code

Two 74HC595 are used in this experiment used, one for controlling columns of LEDMatrix, another for lines. And two 74HC595 are connected in cascade way (series) and has 16 output port. Beacause shiftOut () function output 8-bit data once, twice shiftOut () function are required and data of second stage 74HC595 should be transmitted preferentially. There are two 74HC595 in this experimental circuit, A (first stage) and B (second stage). When the RPi uses shiftOut() function to send data "data1", data of A port will be "data1", and data of B will be 0. Next, use shiftOut() to send "data2", then data "data1" of A will be moved to B and new data "data2" will be moved to A. According to the circuit connection, line data should be sent first, then send column data. The following code will make LEDMatrix display a smiling face, and then display scrolling character "0-F".

C Code 19.1.1 LEDMatrix

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/19.1.1_LEDMatrix
```

2. Use following command to compile “LEDMatrix.c” and generate executable file “LEDMatrix”.

```
gcc LEDMatrix.c –o LEDMatrix –lwiringPi
```

3. Then run the generated file “LEDMatrix”.

```
sudo ./LEDMatrix
```

After the program is executed, LEDMatrix will display a smiling face, and then the display scrolling character "0-F", circularly.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)
// data of smiling face
unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
unsigned char data[]={  // data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
```

```
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
};
int main(void)
{
    int i,j,k;
    unsigned char x;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(j=0;j<500;j++){// Repeat enough times to display the smiling face a period of
time
            x=0x80;
            for(i=0;i<8;i++){
                digitalWrite(latchPin,LOW);
                shiftOut(dataPin,clockPin,LSBFIRST,pic[i]);// first shift data of line
information to the first stage 74HC959
                shiftOut(dataPin,clockPin,LSBFIRST,~x);//then shift data of column
information to the second stage 74HC959

                digitalWrite(latchPin,HIGH);//Output data of two stage 74HC595 at the
same time
                x>>=1;// display the next column
                delay(1);
            }
        }
        for(k=0;k<sizeof(data)-8;k++){  //sizeof(data) total number of "0-F" columns
            for(j=0;j<20;j++){// times of repeated displaying LEDMatrix in every frame,
the bigger the “j”, the longer the display time
               x=0x80;      // Set the column information to start from the first column
                for(i=k;i<8+k;i++){
                    digitalWrite(latchPin,LOW);
                    shiftOut(dataPin,clockPin,LSBFIRST,data[i]);
                    shiftOut(dataPin,clockPin,LSBFIRST,~x);
```

```
66                      digitalWrite(latchPin,HIGH);
67                      x>>=1;
68                      delay(1);
69                  }
70              }
71          }
72      }
73      return 0;
74  }
```

The first “for” cycle in the “while” cycle is used to display a static smile. Display column information from left to right, one column by one column, totally 8 columns. Repeat 500 times to ensure display time enough.

```
          for(j=0;j<500;j++){// Repeat enough times to display the smiling face a period
of time
              x=0x80;
              for(i=0;i<8;i++){
                  digitalWrite(latchPin,LOW);
                  shiftOut(dataPin,clockPin,LSBFIRST,pic[i]);
                  shiftOut(dataPin,clockPin,LSBFIRST,~x);
                  digitalWrite(latchPin,HIGH);
                  x>>=1;
                  delay(1);
              }
          }
```

The second “for” cycle is used to display scrolling characters "0-F", totally 18*8=144 columns. Display the 0-8 column, 1-9 column, 2-10 column..... 138-144 column in turn to achievew scrolling effect. The display of each frame is repeated a certain number of times, and the more times the number of repetitions, the longer the single frame display, the slower the rolling.

```
          for(k=0;k<sizeof(data)-8;k++){  //sizeof(data) total number of "0-F" columns
              for(j=0;j<20;j++){// times of repeated displaying LEDMatrix in every frame,
the bigger the “j”, the longer the display time
                  x=0x80;      // Set the column information to start from the first column
                  for(i=k;i<8+k;i++){
                      digitalWrite(latchPin,LOW);
                      shiftOut(dataPin,clockPin,LSBFIRST,data[i]);
                      shiftOut(dataPin,clockPin,LSBFIRST,~x);
                      digitalWrite(latchPin,HIGH);
                      x>>=1;
                      delay(1);
                  }
              }
          }
```

Pyhton Code 19.1.1 LEDMatrix

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/19.1.1_LEDMatrix
```

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

```
python LEDMatrix.py
```

After the program is executed, LEDMatrix will display a smiling face, and then the display scrolling character "0-F", circularly.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 11      #DS Pin of 74HC595(Pin14)
latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
clockPin = 15       #SH_CP Pin of 74HC595(Pin11)
pic = [0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c]# data of smiling face
data = [#data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
]
def setup():
    GPIO.setmode(GPIO.BOARD)     # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)
```

```
36
37  def shiftOut(dPin,cPin,order,val):
38      for i in range(0,8):
39          GPIO.output(cPin,GPIO.LOW);
40          if(order == LSBFIRST):
41              GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
42          elif(order == MSBFIRST):
43              GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
44          GPIO.output(cPin,GPIO.HIGH);
45
46  def loop():
47      while True:
48          for j in range(0,500):# Repeat enough times to display the smiling face a period
49  of time
50              x=0x80
51              for i in range(0,8):
52                  GPIO.output(latchPin,GPIO.LOW)
53                  shiftOut(dataPin,clockPin,LSBFIRST,pic[i])  #first  shift  data  of  line
54  information to first stage 74HC959
55
56                  shiftOut(dataPin,clockPin,LSBFIRST,~x) #then shift data of column
57  information to second stage 74HC959
58                  GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage 74HC595 at the
59  same time
60                  time.sleep(0.001)# display the next column
61                  x>>=1
62          for k in range(0,len(data)-8):#len(data) total number of "0-F" columns
63              for j in range(0,20):# times of repeated displaying LEDMatrix in every frame,
64  the bigger the “j”, the longer the display time.
65                  x=0x80      # Set the column information to start from the first column
66                  for i in range(k,k+8):
67                      GPIO.output(latchPin,GPIO.LOW)
68                      shiftOut(dataPin,clockPin,LSBFIRST,data[i])
69                      shiftOut(dataPin,clockPin,LSBFIRST,~x)
70                      GPIO.output(latchPin,GPIO.HIGH)
71                      time.sleep(0.001)
72                      x>>=1
73  def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
74      GPIO.cleanup()
75  if __name__ == '__main__': # Program starting from here
76      print 'Program is starting...'
77      setup()
78      try:
79          loop()
```

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

The first “for” cycle in the “while” cycle is used to display a static smile. Display column information from left to right, one column by one column, totally 8 columns. Repeat 500 times to ensure display time enough.

```
        for j in range(0,500):# Repeat enough times to display the smiling face a period of time
            x=0x80
            for i in range(0,8):
                GPIO.output(latchPin,GPIO.LOW)
                shiftOut(dataPin,clockPin,LSBFIRST,pic[i])#first shift data of line information to first stage 74HC959
                shiftOut(dataPin,clockPin,LSBFIRST,~x)#then shift data of column information to first stage 74HC959

                GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage 74HC595 at the same time.
                time.sleep(0.001)# display the next column
                x>>=1
```

The second “for” cycle is used to display scrolling characters "0-F", totally 18*8=144 columns. Display the 0-8 column, 1-9 column, 2-10 column..... 138-144 column in turn to achievew scrolling effect. The display of each frame is repeated a certain number of times, and the more times the number of repetitions, the longer the single frame display, the slower the rolling.

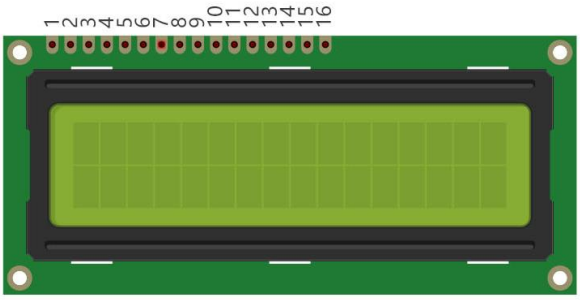
```
        for k in range(0,len(data)-8):#len(data) total number of “0-F” columns.
            for j in range(0,20):# times of repeated displaying LEDMatrix in every frame, the bigger the “j”, the longer the display time
                x=0x80      # Set the column information to start from the first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

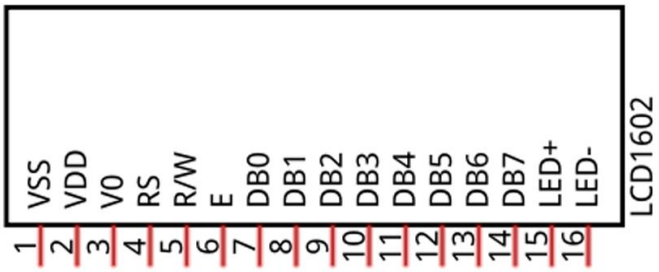
Chapter 20 LCD1602

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

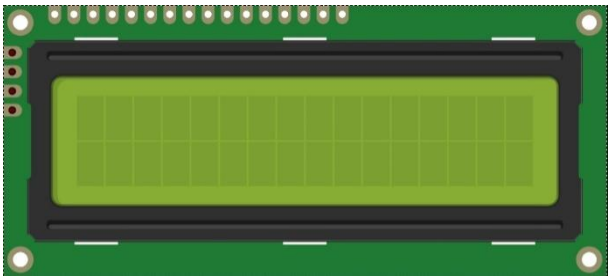
Project 20.1 I2C LCD1602

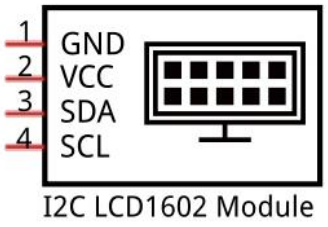
LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. As shownbelow is a monochrome LCD1602 display screen， and its circuit pin diagram：





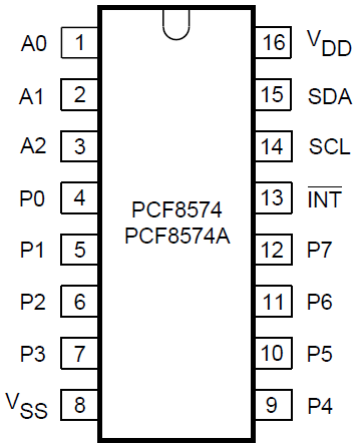
I2C LCD1602 integrates a I2C interface, which connects the serial-input &parallel-output module to LCD1602. We just use 4 lines to the operate LCD1602 easily.



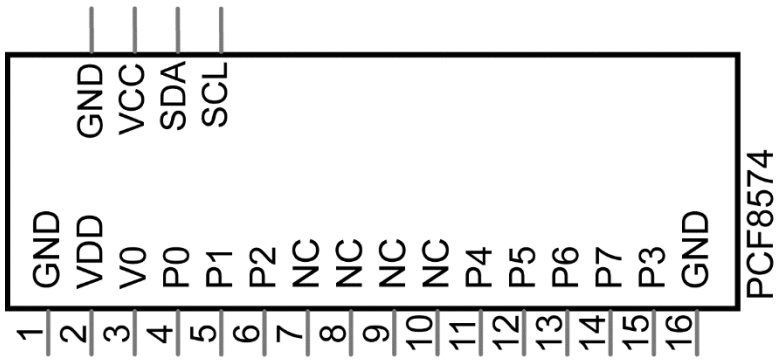


The serial-to-parallel chip used in this module is PCF8574(PCF8574A),and its default I2C address is 0x27(0x3F), and you can view all the RPI bus on your I2C device address through command "i2cdetect –y 1" to. (refer to the "configuration I2C" section below) below is the PCF8574 pin schematic diagram and the block pin diagram:

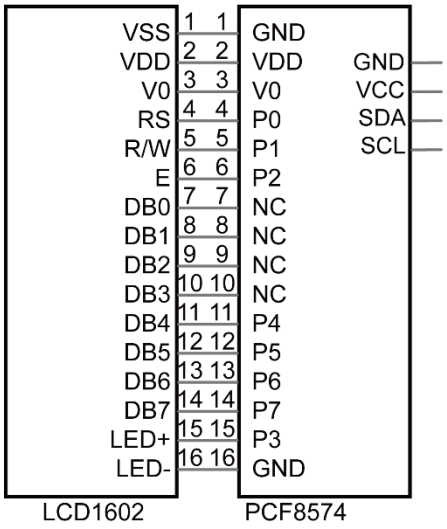
PCF8574 chip pin diagram:



PCF8574 module pin diagram

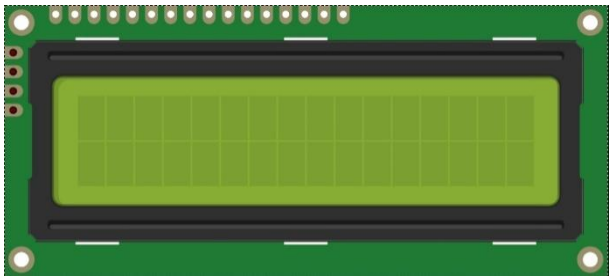


PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



So, we can use just 4 pins to control LCD1602 with 16 pins easilyt hrough I2C interface.
In this experiment, we will use I2CLCD1602 to display some static characters and dynamic variables.

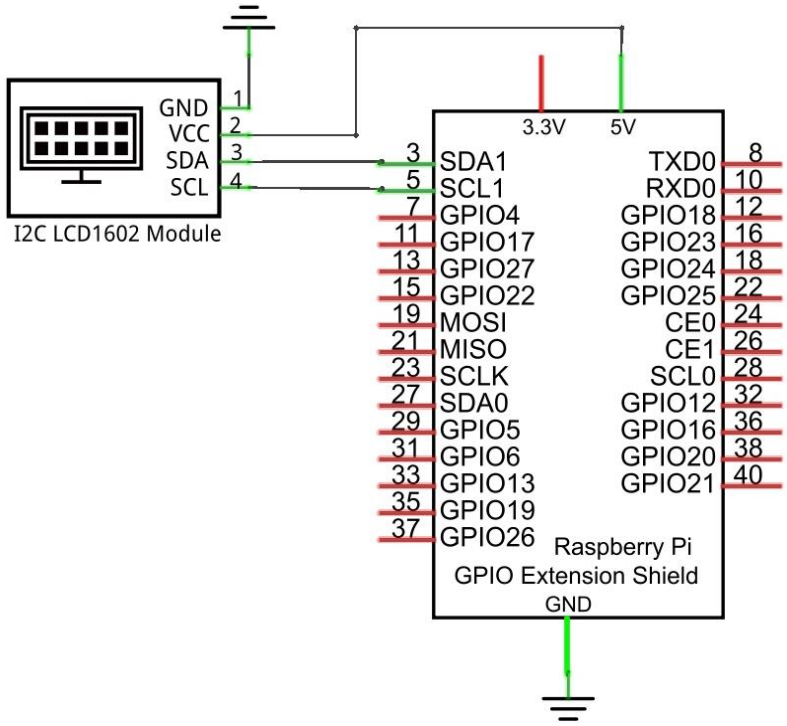
Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M x4 
I2C LCD1602 Module x1 	

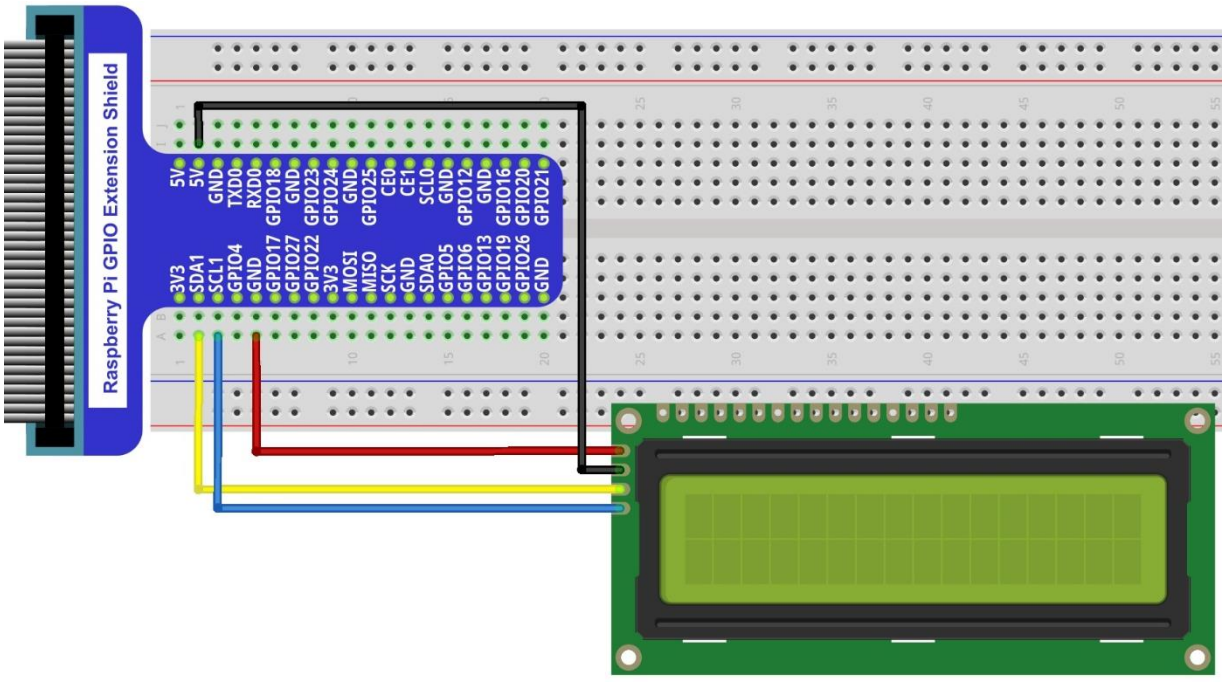
Circuit

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

Schematic diagram



Hardware connection



Code

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

C Code 20.1.1 I2CLCD1602

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Super_Starter_Kit_for_Raspberry_Pi/Code/C_Code/20.1.1_I2CLCD1602
```

2. Open the file I2CLCD1602.c, and find the macro definition "pcf8574_address". If your serial-to-parallel module uses chip PCF8574, set the macro "pcf8574_address" value to 0x27.If your serial-to-parallel module uses chip PCF8574A, set the macro "pcf8574_address" value to 0x3F.

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

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

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

4. Then run the generated file “I2CLCD1602”.

```
sudo ./ I2CLCD1602
```

After the program is executed, LCD1602 screen will display current CPU temperature and system time. If there is no display or the display is not clear, adjust potentiometer of PCF8574 module to adjust the contrast of LCD1602 until the screen can display clearly.

The following is the program code:

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <wiringPi.h>
4  #include <pcf8574.h>
5  #include <lcd.h>
6  #include <time.h>
7
8  //#define pcf8574_address 0x27        // default I2C address of Pcf8574
9  #define pcf8574_address 0x3F        // default I2C address of Pcf8574A
10 #define BASE 64         // BASE is not less than 64
11 //////// Define the output pins of the PCF8574, which are directly connected to the
12 LCD1602 pin.
13 #define RS      BASE+0
14 #define RW      BASE+1
15 #define EN      BASE+2
16 #define LED     BASE+3
17 #define D4      BASE+4
18 #define D5      BASE+5
19 #define D6      BASE+6
20 #define D7      BASE+7
21
22 int lcdhd;// used to handle LCD
23 void printCPUTemperature(){// sub function used to print CPU temperature
```

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

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pcf8574Setup(BASE,pcf8574_address);// initialize PCF8574
    for(i=0;i<8;i++){
        pinMode(BASE+i,OUTPUT);     // set PCF8574 port to output mode
    }
    digitalWrite(LED,HIGH);     // turn on LCD backlight
    digitalWrite(RW,LOW);       // allow writing to LCD
lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);// initialize LCD and return  “handle”
used to handle LCD
    if(lcdhd == -1){
        printf("lcdInit failed !");
        return 1;
    }
    while(1){
        printCPUTemperature();// print CPU temperature
        printDataTime();        // print system time
```

```
68          delay(1000);
69      }
70      return 0;
}
```

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

```
//#define pcf8574_address 0x27        // default I2C address of Pcf8574
#define pcf8574_address 0x3F      // default I2C address of Pcf8574A
#define BASE 64         // BASE is not less than 64
//////// Define the output pins of the PCF8574, which are directly connected to the
LCD1602 pin.
#define RS      BASE+0
#define RW      BASE+1
#define EN      BASE+2
#define LED     BASE+3
#define D4      BASE+4
#define D5      BASE+5
#define D6      BASE+6
#define D7      BASE+7
```

Then, in main function, initialize the PCF8574, set all the pins to output mode, and turn on the LCD1602 backlight.

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

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

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

Details about lcdInit() :

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

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

In the next "while", two sub functions are called to display the CPU temperature and the time. First look at the sub function printCPUTemperature(). The CPU temperature data is stored in the "/sys/class/thermal/thermal_zone0/temp " file. We need read contents of the file, and converts it to temperature value stored in variable CPU_temp, and use lcdPrintf() to display it on LCD.

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

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

Details about lcdPosition() and lcdPrintf():

lcdPosition (int handle, int x, int y);

Set the position of the cursor for subsequent text entry.

lcdPutchar (int handle, uint8_t data)
lcdPuts (int handle, char *string)
lcdPrintf (int handle, char *message, …)

These output a single ASCII character, a string or a formatted string using the usual printf formatting commands.

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

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

Pyhton Code 20.1.1 I2CLCD1602

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Super_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/20.1.1_I2CLCD1602
```

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

```
python I2CLCD1602.py
```

After the program is executed, LCD1602 screen will display current CPU temperature and system time. If there is no display or the display is not clear, adjust potentiometer of PCF8574 module to adjust the contrast of LCD1602 until the screen can display clearly.

The following is the program code:

```
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

from time import sleep, strftime
from datetime import datetime

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

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

def loop():
    mcp.output(3,1)     # turn on LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)  # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( get_time_now() )   # display the time
        sleep(1)

def destroy():
    lcd.clear()

PCF8574_address = 0x27  # I2C address of the PCF8574 chip.
PCF8574A_address = 0x3F  # I2C address of the PCF8574A chip.
# Create PCF8574 GPIO adapter.
try:
    mcp = PCF8574_GPIO(PCF8574_address)
```

```
except:
    try:
        mcp = PCF8574_GPIO(PCF8574A_address)
    except:
        print 'I2C Address Error !'
        exit(1)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)

if __name__ == '__main__':
    print 'Program is starting ... '
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

Two modules are used in the code, PCF8574.py and Adafruit_LCD1602.py. These two documents and the code file are stored in the same directory, and neither of them is dispensable. Please do not delete. PCF8574.py is used to provide I2C communication mode and operation method of some port for RPi and PCF8574 chip. Adafruit module Adafruit_LCD1602.py is used to provide some function operation method for LCD1602.

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

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

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

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

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

```
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)  # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( get_time_now() )   # display the time
        sleep(1)
```

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

```
def get_cpu_temp():     # get CPU temperature and store it into file
  "/sys/class/thermal/thermal_zone0/temp"
    tmp = open('/sys/class/thermal/thermal_zone0/temp')
```

```
    cpu = tmp.read()
    tmp.close()
    return '{:.2f}'.format( float(cpu)/1000 ) + ' C'
```

Sub function used to get time:

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

Details about PCF8574.py and Adafruit_LCD1602.py:

Module PCF8574

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

Mofule Adafruit_LCD1602

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

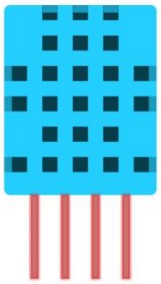
Chapter 21 Hygrothermograph DHT11

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

Project 21.1 Hygrothermograph

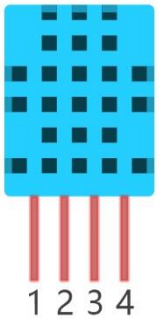
Hygrothermograph is an important tool in our life to remind us of keeping warm and replenishing moisture in time. In this experiment, we will use RPi to read temperature and humidity data of DHT11.

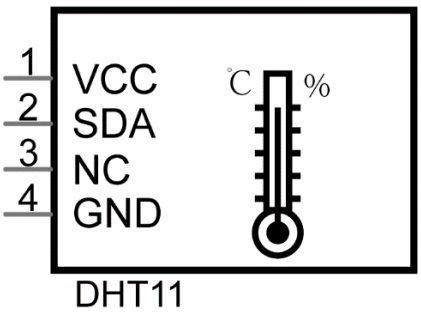
Component List

Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	DHT11 x1 	Resistor 10kΩ x1 
Jumper M/M x4 		

Component knowledge

Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated inside.

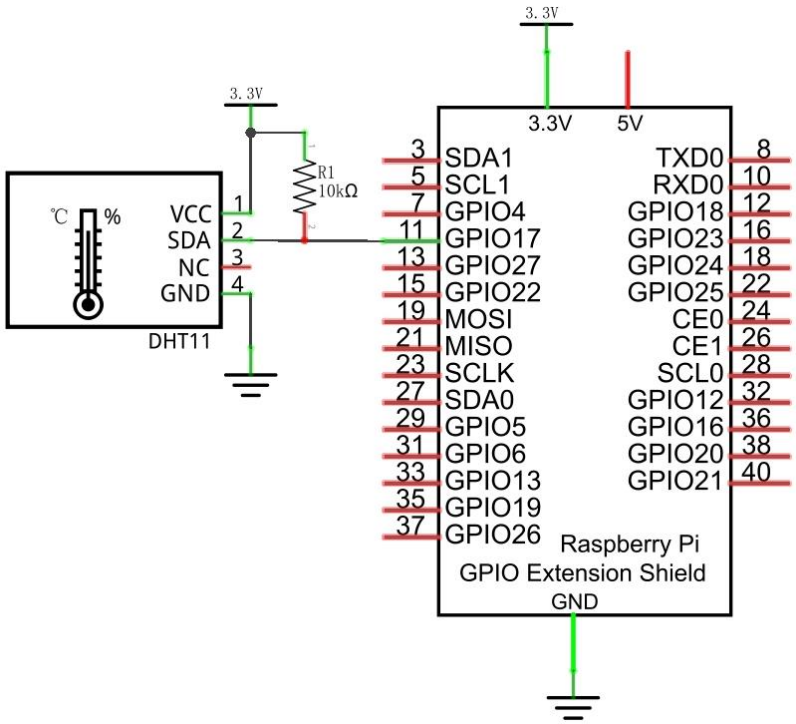




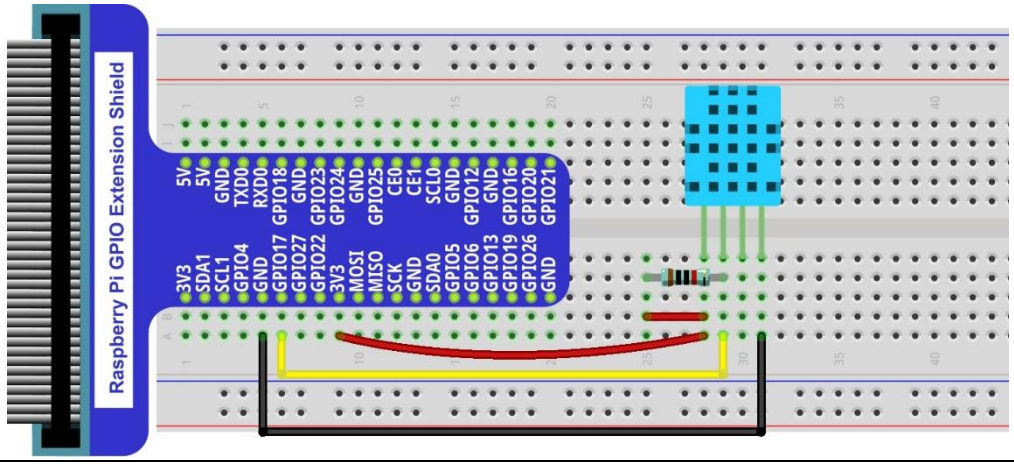
It has 1S's initialization time after powered up. The operating voltage is within the range of 3.3V-5.5V。

Circuit

Schematic diagram



Hardware connection



Code

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

C Code 21.1.1 DHT11

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/21.1.1_DHT11
```

2. Code of this experiment contains a custom header file. Use the following command to compile the code DHT11.cpp and DHT.cpp and generate executable file DHT11. And the custom header file will be compiled at the same time.

```
gcc DHT.cpp DHT11.cpp –o DHT11 –lwiringPi
```

3. Run the generated file "DHT11".

```
sudo ./DHT11
```

After the program is executed, the terminal window will display the current total number of reading times, the read state, as well as the temperature and humidity value. As is shown below:

```
The sumCnt is : 1
DHT11,OK!
Humidity is 57.00 %,     Temperature is 30.00 *C

The sumCnt is : 2
DHT11,OK!
Humidity is 57.00 %,     Temperature is 30.00 *C

The sumCnt is : 3
DHT11,OK!
Humidity is 57.00 %,     Temperature is 30.00 *C

The sumCnt is : 4
DHT11,OK!
Humidity is 57.00 %,     Temperature is 29.00 *C
```

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include "DHT.hpp"
5
6  #define DHT11_Pin  0    //define the pin of sensor
7
8  int main(){
9      DHT dht;        //create a DHT class object
10     int chk,sumCnt;//chk:read the return value of sensor; sumCnt:times of reading sensor
11     if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
12         printf("setup wiringPi failed !");
13         return 1;
14     }
15     while(1){
```

```
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
        sumCnt++;       //counting number of reading times
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:     //if the return value is DHTLIB_OK, the data is normal.
                printf("DHT11,OK! \n");
                break;
            case DHTLIB_ERROR_CHECKSUM:     //data check has errors
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:      //reading DHT times out
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:      //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f
*C\n\n",dht.humidity,dht.temperature);
        delay(1000);
    }
    return 1;
}
```

In this experimental code, we use a custom library file "DHT.hpp". It is located in the same directory with program files "DHT11.cpp" and "DHT.cpp", and methods for reading DHT sensor are provided in the library file. By using this library, we can easily read the DHT sensor. First create a DHT class object in the code.

```
    DHT dht;
```

And then in the "while" cycle, use chk = dht.**readDHT11** (DHT11_Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk". And then use variable sumCnt to record number of reading times.

```
while(1){
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
        sumCnt++;       //读取次数计数 count number of times of reading
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:     //if the return value is DHTLIB_OK, the data is normal.
                printf("DHT11,OK! \n");
                break;
            case DHTLIB_ERROR_CHECKSUM:     //data check has errors
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:      //reading DHT times out
```

```
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:      //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
```

Finally print the results:

```
printf("Humidity is %.2f %%, \t Temperature is %.2f *C\n\n",dht.humidity,dht.temperature);
```

Library file "DHT.hpp" contains a DHT class and his public member functions int **readDHT11** (int pin) is used to read sensor DHT11 and store the temperature and humidity data read to member variables double humidity and temperature. The implementation method of the function is included in the file "DHT.cpp".

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <stdint.h>
4
5  ////read return flag of sensor
6  #define DHTLIB_OK               0
7  #define DHTLIB_ERROR_CHECKSUM   -1
8  #define DHTLIB_ERROR_TIMEOUT    -2
9  #define DHTLIB_INVALID_VALUE    -999
10
11 #define DHTLIB_DHT11_WAKEUP     18
12 #define DHTLIB_DHT_WAKEUP       1
13
14 #define DHTLIB_TIMEOUT          100
15
16 class DHT{
17     public:
18         double humidity,temperature;    //use to store temperature and humidity data read
19         int readDHT11(int pin);     //read DHT11
20     private:
21         int bits[5];    //Buffer to receiver data
22         int readSensor(int pin,int wakeupDelay);    //
23
24 };
```

Pyhton Code 21.1.1 DHT11

First observe the experimental phenomenon, and then analyze the code.

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

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/21.1.1_DHT11
```

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

```
python DHT11.py
```

After the program is executed, the terminal window will display the current total number of read, the read state, as well as the temperature and humidity value. As is shown below:

```
Program is starting ...
The sumCnt is : 1,        chk : 0
DHT11,OK!
Humidity : 62.00,         Temperature : 30.00

The sumCnt is : 2,        chk : 0
DHT11,OK!
Humidity : 63.00,         Temperature : 30.00

The sumCnt is : 3,        chk : 0
DHT11,OK!
Humidity : 62.00,         Temperature : 30.00

The sumCnt is : 4,        chk : 0
DHT11,OK!
Humidity : 62.00,         Temperature : 30.00
```

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3  import Freenove_DHT as DHT
4  DHTPin = 11     #define the pin of DHT11
5
6  def loop():
7      dht = DHT.DHT(DHTPin)   #create a DHT class object
8      sumCnt = 0              #number of reading times
9      while(True):
10         sumCnt += 1         #counting number of reading times
11         chk = dht.readDHT11(DHTPin)     #read DHT11 and get a return value. Then
12 determine whether data read is normal according to the return value.
13         print"The sumCnt is : %d, \t chk    : %d"%(sumCnt,chk)
14         if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then
15 determine whether data read is normal according to the return value.
16             print"DHT11,OK!"
17         elif(chk is dht.DHTLIB_ERROR_CHECKSUM): #data check has errors
18             print"DHTLIB_ERROR_CHECKSUM!!"
19         elif(chk is dht.DHTLIB_ERROR_TIMEOUT):  #reading DHT times out
20             print"DHTLIB_ERROR_TIMEOUT!"
21         else:               #other errors
22             print"Other error!"
23
24         print"Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature)
25         time.sleep(1)
26
27 if __name__ == '__main__':
28     print 'Program is starting ... '
29     try:
30         loop()
```

```
31      except KeyboardInterrupt:
32          GPIO.cleanup()
33          exit()
```

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

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

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

```
while(True):
        sumCnt += 1         #counting number of reading times
        chk = dht.readDHT11(DHTPin)     #read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
        print"The sumCnt is : %d, \t chk    : %d"%(sumCnt,chk)
        if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
            print"DHT11,OK!"
        elif(chk is dht.DHTLIB_ERROR_CHECKSUM): #data check has errors
            print"DHTLIB_ERROR_CHECKSUM!!"
        elif(chk is dht.DHTLIB_ERROR_TIMEOUT):  #reading DHT times out
            print"DHTLIB_ERROR_TIMEOUT!"

        else:               #other errors
            print"Other error!"
```

Finally print the results:

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

Module "**Freenove_DHT.py**" contains a DHT class. And class functions def **readDHT11** (pin) is used to read sensor DHT11 and store the temperature and humidity data read to member variables humidity and temperature.

Freenove_DHT Module

This is a Python module for reading the temperature and humidity data of the DHT sensor. Partial functions and variables are described as follows:

Variable **humidity**: store humidity data read from sensor

Variable **temperature**: store temperature data read from sensor

def readDHT11 **(pin)**: read the temperature and humidity of sensor DHT11, and return values used to determine whether the data is normal.

Chapter 22 Matrix Keypad

We have learned usage of a single button before. In this chapter, we will learn a device which integrates a number of key, matrix keyboard.

Project 22.1 Matrix Keypad

In this experiment, we will try to get every key code on the Keypad.

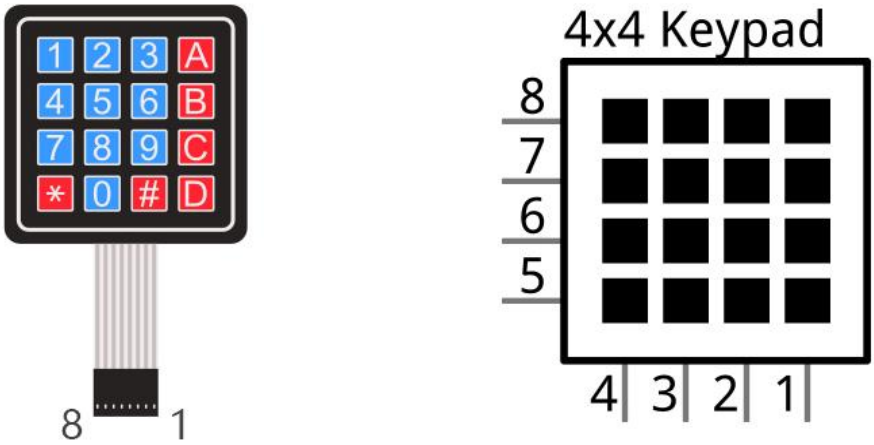
Component List

Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	4x4 Matrix Keypad x1 
Jumper M/M x8 	

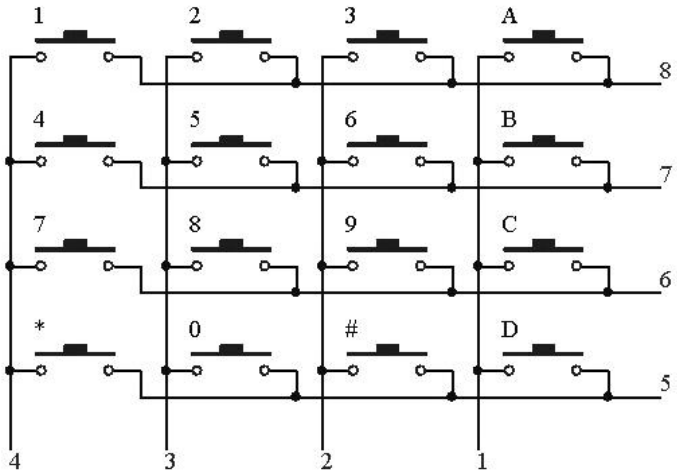
Component knowledge

4x4 Matrix Keypad

Keypad is a device that integrates a number of keys. As is shown bellow, a 4x4 Keypad integrates 16 keys:



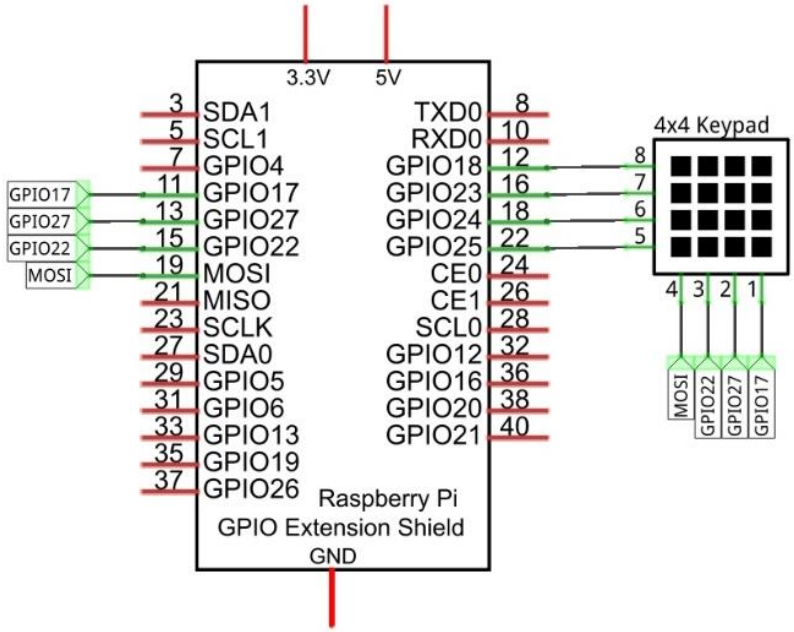
Like the integration of LED matrix, in 4x4 Keypad each row of keys are connected in with one pin and it is the same as each column.: such connection can reduce the occupation of processor port. Internal circuit is shown below.



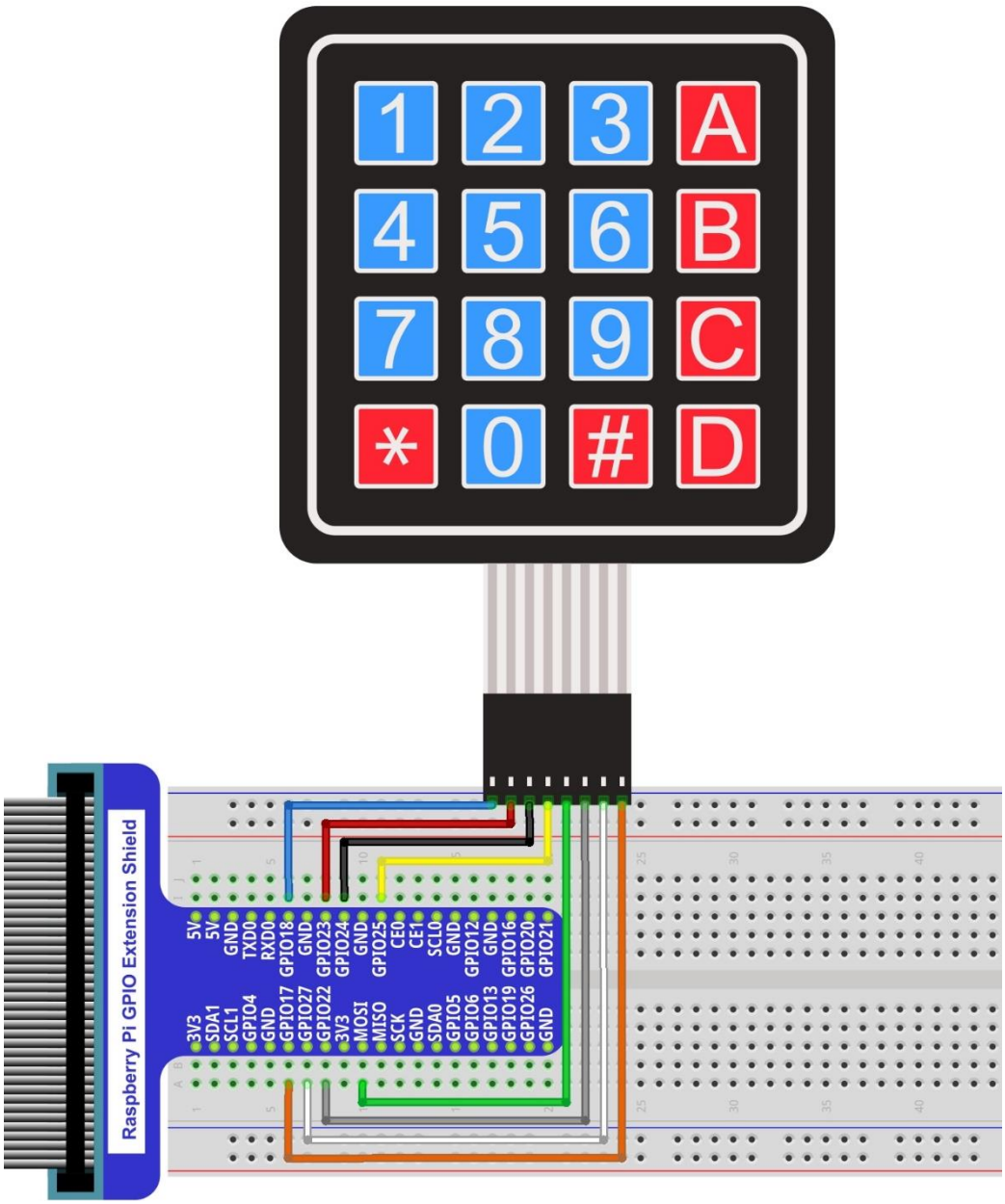
The usage method is similar to the Matrix LED, namely, uses a row scan or column scanning method to detect the state of key on each column or row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. And then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Then, you can get the state of all keys.

Circuit

Schematic diagram



Hardware connection



Code

This code is used to obtain all key code of 4x4 Matrix Keypad, when one of keys is pressed, the key code will be printed out in he terminal window.

C Code 22.1.1 MatrixKeypad

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 22.1.1_MatrixKeypad directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/22.1.1_MatrixKeypad
```

2. Code of this experiment contains a custom header file. Use the following command to compile the code MatrixKeypad.cpp, Keypad.cpp amd Key.cpp and generate executable file MatrixKeypad. And the custom header file will be compiled at the same time.

```
gcc MatrixKeypad.cpp Keypad.cpp Key.cpp –o MatrixKeypad –lwiringPi
```

3. Run the generated file "MatrixKeypad".

```
sudo ./MatrixKeypad
```

After the program is executed, press any key on the MatrixKeypad, the terminal will print out the corresponding key code. As is shown bellow:

```
Program is starting ...
You Pressed key :  1
You Pressed key :  2
You Pressed key :  3
You Pressed key :  4
You Pressed key :  5
You Pressed key :  6
You Pressed key :  7
You Pressed key :  8
You Pressed key :  9
You Pressed key :  0
You Pressed key :  A
You Pressed key :  B
You Pressed key :  C
You Pressed key :  D
You Pressed key :  *
You Pressed key :  #
```

The following is the program code:

```
#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {  //key code
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
//create Keypad object
```

```
14  Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
15
16  int main(){
17      printf("Program is starting ... \n");
18      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
19          printf("setup wiringPi failed !");
20          return 1;
21      }
22      char key = 0;
23      keypad.setDebounceTime(50);
24      while(1){
25          key = keypad.getKey();  //get the state of keys
26          if (key){       //if a key is pressed, print out its key code
27              printf("You Pressed key :  %c \n",key);
28          }
29      }
30      return 1;
31  }
```

In this experimental code, we use two custom library file "**Keypad**.hpp" and "**Key**.hpp". They are located in the same directory with program files "**MatrixKeypad**.cpp", "**Keypad**.cpp" and "**Key**.cpp". Library Keypad is transplanted from the Arduino library Keypad. And this library file provides a method to read the keyboard. By using this library, we can easily read the matrix keyboard.

First, define the information of the matrix keyboard used in this experiment: the number of rows and columns, code of each key and GPIO pin connected to each column and each row. It is necessary to include the header file "**Keypad**.hpp".

```
#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {  //key code
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
```

And then, based on the above information, instantiate a Keypad class object to operate the matrix keyboard.

```
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

Set the debounce time to 50ms, and this value can be set based on the actual use of the keyboard flexiblely, with default time 10ms.

```
keypad.setDebounceTime(50);
```

In the "while" cycle, use the function key= keypad.**getKey** () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", then be printed out.

```
    while(1){
        key = keypad.getKey();  //get the state of keys
        if (key){       // if a key is pressed, print out its key code
            printf("You Pressed key :  %c \n",key);
        }
    }
```

The library Keypad used for RPi is transplanted from the Arduino library Keypad. And the source files can be obtained by visiting http://playground.arduino.cc/Code/Keypad. As for transplanted function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad library are described below:

class Keypad

Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols);

Constructor, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.

char getKey();

Get the key code of the pressed key. If no key is pressed, the return value is NULL.

void setDebounceTime(uint);

Set the debounce time. And the default time is 10ms.

void setHoldTime(uint);

Set the time when the key holds stable state after pressed.

bool isPressed(char keyChar);

Judge wether the key with code "keyChar" is pressed.

char waitForKey();

Wait for a key to be pressed, and return key code of the pressed key.

KeyState getState();

Get state of the keys.

bool keyStateChanged();

Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad or through the opening file "Keypad.hpp".

Pyhton Code 22.1.1 MatrixKeypad

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 22.1.1_MatrixKeypad directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/22.1.1_MatrixKeypad
```

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

```
python MatrixKeypad.py
```

After the program is executed, press any key on the MatrixKeypad, the terminal will print out the corresponding key code. As is shown bellow:

```
Program is starting ...
You Pressed Key : 1
You Pressed Key : 2
You Pressed Key : 3
You Pressed Key : 4
You Pressed Key : 5
You Pressed Key : 6
You Pressed Key : 7
You Pressed Key : 8
You Pressed Key : 9
You Pressed Key : *
You Pressed Key : 0
You Pressed Key : #
You Pressed Key : A
You Pressed Key : B
You Pressed Key : C
You Pressed Key : D
```

The following is the program code：

```
import Keypad        #import module Keypad
ROWS = 4         # number of rows of the Keypad
COLS = 4         #number of columns of the Keypad
keys =  [   '1','2','3','A',     #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'      ]
rowsPins = [12,16,18,22]         #connect to the row pinouts of the keypad
colsPins = [19,15,13,11]         #connect to the column pinouts of the keypad

def loop():
    keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)     #creat Keypad object
    keypad.setDebounceTime(50)       #set the debounce time
    while(True):
        key = keypad.getKey()        #obtain the state of keys
        if(key != keypad.NULL):      #if there is key pressed, print its key code.
            print "You Pressed Key : %c "%(key)

if __name__ == '__main__':     #Program start from here
    print "Program is starting ... "
    try:
        loop()
```

```
23      except KeyboardInterrupt:  #When 'Ctrl+C' is pressed, exit the program.
24          GPIO.cleanup()
```

In this experimental code, we use two custom module "**Keypad**.py", which is located in the same directory with program file "**MatrixKeypad**.py". And this library file, which is transplanted from Arduino function library Keypad, provides a method to read the keyboard. By using this library, we can easily read the matrix keyboard. First, import module Keypad. Then define the information of the matrix keyboard used in this experiment: the number of rows and columns, code of each key and GPIO pin connected to each column and each row.

```
import Keypad       #import module Keypad
ROWS = 4        #number of rows of the Keypad
COLS = 4        #number of columns of the Keypad
keys =  [   '1','2','3','A',    #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'     ]
rowsPins = [12,16,18,22]        #connect to the row pinouts of the keypad
colsPins = [19,15,13,11]        #connect to the column pinouts of the keypad
```

And then, based on the above information, instantiate a Keypad class object to operate the matrix keyboard.

```
keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)
```

Set the debounce time to 50ms, and this value can be set based on the actual use of the keyboard flexiblely, with default time 10ms.

```
keypad.setDebounceTime(50)
```

In the "while" cycle, use the function key= keypad.**getKey** () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", and then be printed out.

```
    while(True):
        key = keypad.getKey()       #get the state of keys
        if(key != keypad.NULL):     # if a key is pressed, print out its key code
            print "You Pressed Key : %c "%(key)
```

The library Keypad used for RPi is transplanted from the Arduino library Keypad. The source files is written by language C++ and translated to Python can be btained by visiting http://playground.arduino.cc/Code/Keypad. As for transplanted function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad library are described below:

class Keypad

```
def __init__(self,usrKeyMap,row_Pins,col_Pins,num_Rows,num_Cols):
```
Constructed function, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.

```
def getKey(self):
```
Get a pressed key. If no key is pressed, the return value is keypad NULL.

```
def setDebounceTime(self,ms):
```
Set the debounce time. And the default time is 10ms.

```
def setHoldTime(self,ms):
```
Set the time when the key holds stable state after pressed.

```
def isPressed(keyChar):
```
Judge wether the key with code "keyChar" is pressed.

```
def waitForKey():
```
Wait for a key to be pressed, and return key code of the pressed key.

```
def getState():
```
Get state of the keys.

```
def keyStateChanged():
```
Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad or through the opening file "Keypad.py".

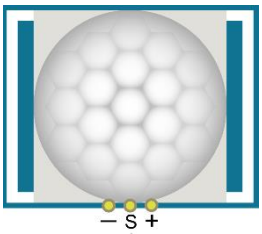
Chapter 23 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 23.1 Sense LED

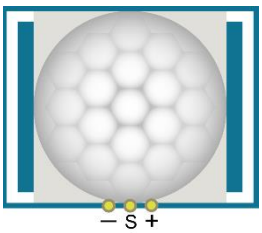
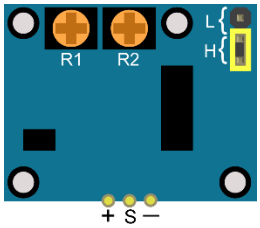
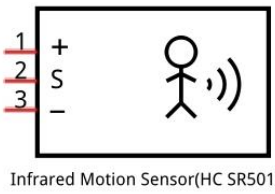
In this project, we will make a sense LED, with the human body infrared pyroelectric sensors.
When someone get close to the LED, it will light automatically. On the contrary, it will be out.
This infrared motion sensor is a kind of sensor which can detect the infrared emitted by human and animals.

Component List

<table>
<tr><td colspan="2">Raspberry Pi 3B x1
GPIO Expansion Board & Wire x1
BreadBoard x1</td><td>Jumper M/M x2 M/F x3
  </td></tr>
<tr><td>HC SR501 x1
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td></tr>
</table>

Component knowledge

The following is the diagram of infrared Motion sensor（HC SR-501）:

Top	Bottom	Schematic
		 Infrared Motion Sensor(HC SR501)

Description:

1. Working voltage: 5v-20v(DC)　Static current: 65uA.
2. Automatic trigger. When the body enter into the active area of sensor, the module will output high level (3.3V). When body leave out, it will output high level lasting for time T, then output low level(0V). Delay time T can be adjusted by potentiometer R1.
3. According to the position of jumper cap, you can choose non-repeatable trigger mode or repeatable mode.

L: non-repeatable trigger mode. The module output high level after sensing body, then when delay time is over, the module will output low level. And during high level time, the sensor don't sense body anymore.

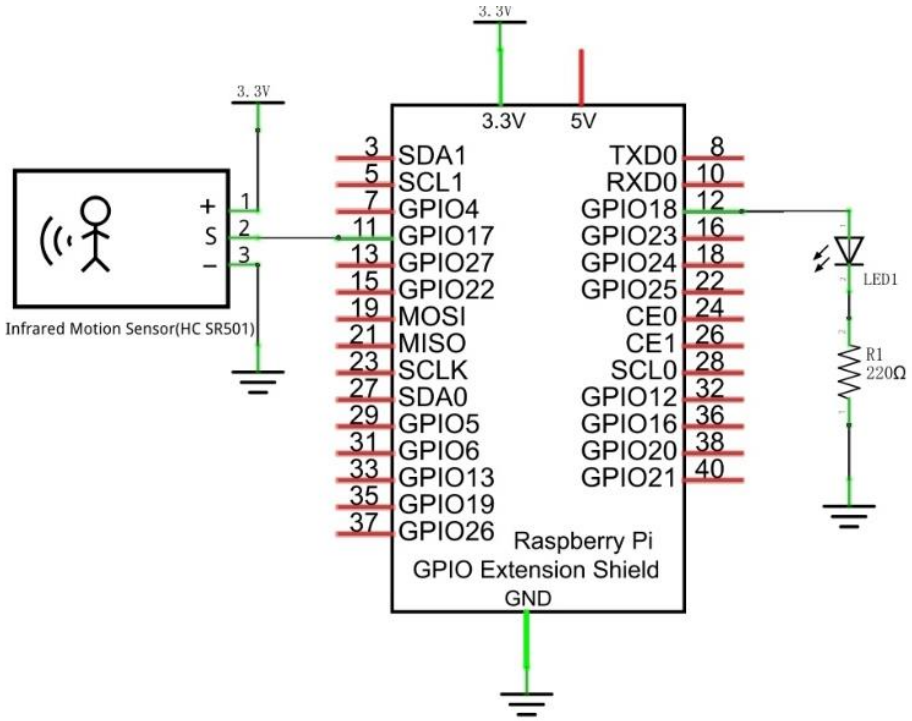
H: repeatable trigger mode. The distinction from L is that it can sense body until body leaves during the period of outputting high level. And then it starts to time and output low level after delaying T time.

4. Induction block time: the induction will stay in block condition and do not induce external signal at a little time (less than delay time) after outputting high level or low level
5. Initialization time: the module needs about 1 minute to initialize after powered on. During this period, it will output high or low level alternately.
6. In consideration of feature of this sensor, when body get close or away from edgewise side, the sensor will work with high sensitively. When body get close or away in vertical direction, the sensor can't work well, which should get your attention. Sensing distance is adjusted by potentiometer.

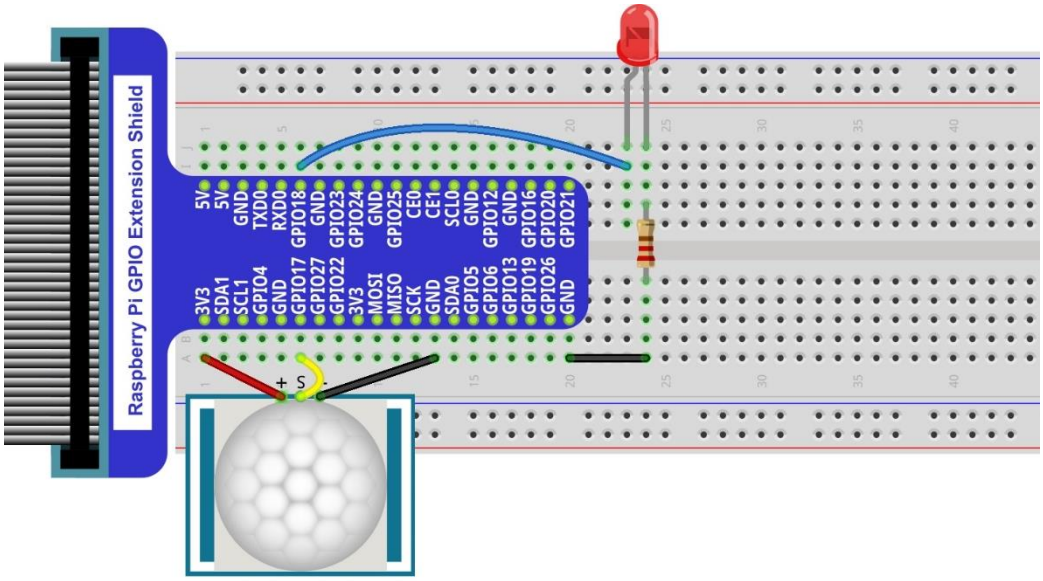
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



Hardware connection



Code

In this experiment, we use infrared motion sensor to control LED, and take infrared motion sensor as a switch, so the code very similar to front experiment "Button&LED" in logic. The difference is that, when infrared motion sensor detects change, it will output high level; when button is pressed, it will output low level. When the sensor output high level, the LED will be turned on, or it will be turned off.

C Code 23.1.1 SenseLED

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 23.1.1_SenseLED directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/23.1.1_SenseLED
```

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

```
gcc SenseLED.c –o SenseLED –lwiringPi
```

3. Run the generated file "SenseLED".

```
sudo ./SenseLED
```

After the program is executed, try to leave away from or get closed to the Motion Sensor Infrared and observe whether the LED will be turned on or off. The terminal window will print out the state of LED constantly. As is shown below:

The following is the program code:

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

#define ledPin    1     //define the ledPin
#define sensorPin 0     //define the sensorPin

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

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

    while(1){

        if(digitalRead(sensorPin) == HIGH){
```

```
            digitalWrite(ledPin, HIGH);   //led on
            printf("led on...\n");
        }
        else {
            digitalWrite(ledPin, LOW);   //led off
            printf("...led off\n");
        }
    }

    return 0;
}
```

It can be seen that the code is based on the same logic with the "ButtonLED" code in addition to determining the level of the input signal.

Python Code 23.1.1 SenseLED

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 22.1.1_MatrixKeypad directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/23.1.1_SenseLED
```

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

```
python SenseLED.py
```

After the program is executed, try to leave away from or get closed to the Motion Sensor Infrared and observe whether the LED will be turned on or off. The terminal window will print out the state of LED constantly. As is shown below:

The following is the program code:

```
import RPi.GPIO as GPIO

ledPin = 12    # define the ledPin
sensorPin = 11    # define the sensorPin

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(sensorPin, GPIO.IN)    # Set sensorPin's mode is input
def loop():
    while True:
        if GPIO.input(sensorPin)==GPIO.HIGH:
            GPIO.output(ledPin,GPIO.HIGH)
```

```
15              print 'led on ...'
16          else :
17              GPIO.output(ledPin,GPIO.LOW)
18              print 'led off ...'
19
20  def destroy():
21      GPIO.cleanup()                     # Release resource
22
23  if __name__ == '__main__':     # Program start from here
24      setup()
25      try:
26          loop()
27      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
28  will be  executed.
29          destroy()
```

It can be seen that the code is based on the same logic with the "ButtonLED" code in addition to determining the level of the input signal.

Chapter 24 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 24.1 Ultrasonic Ranging

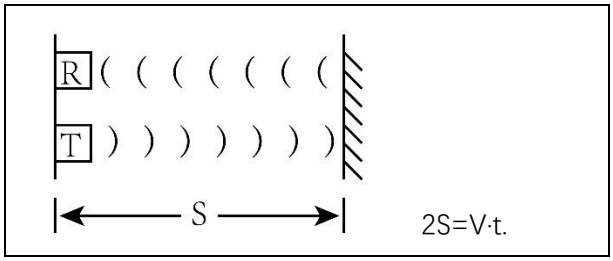
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

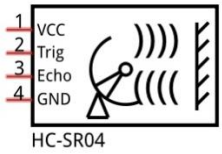
Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	HC SR501 x1 
Jumper M/F x4 	

Component Knowledge

Ultrasonic ranging module use the principle that ultrasonic will reflect when it encounters obstacles. Start counting the time when ultrasonic is transmitted. And when ultrasonic encounters an obstacle, it will reflect back. The counting will end after ultrasonic is received, and the time difference is the total time of ultrasonic from transmitting to receiving. Because the speed of sound in air is constant, and is about v=340m/s. So we can calculate the distance between the modle and the obstacle: s=vt/2.



Ultrasonic module integrates a transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into sound waves (mechanical energy) and the function of the receiver is opposite. The object picture and the diagram of HC SR04 ultrasonic module are shown below:



Pin description:

VCC	power supply pin
Trig	triger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

Maximum measured distance: 200cm

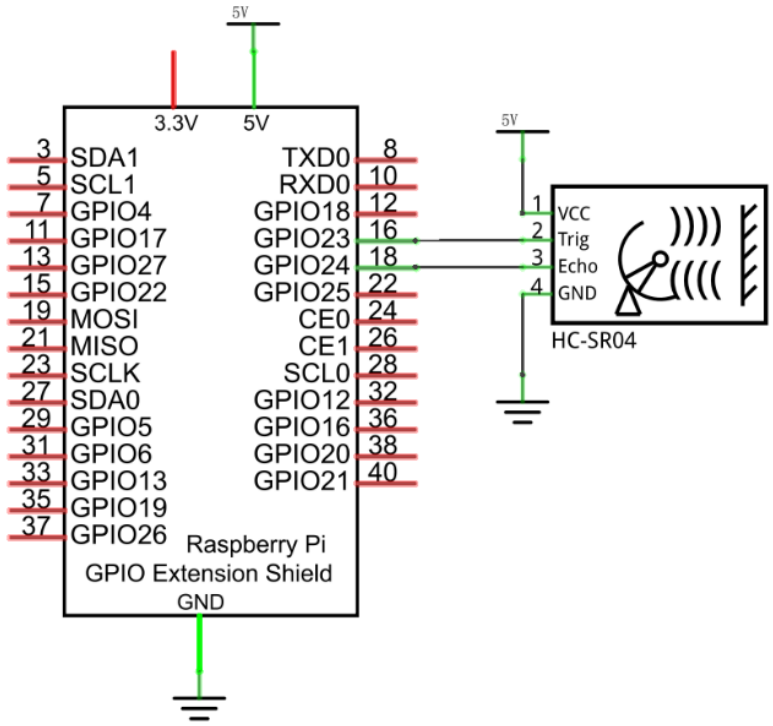
Size: 45mm*20mm*15mm

Instructions for use: output a high-level pulse in Trig pin lasting for least 10uS. Then the the module begins to transmit ultrasonic. At the same time, the Echo pin will be pulled up. When the module receives the returned ultrasonic, the Echo pin will be pulled down. The duration of high level in Echo pin is the total time of the ultrasonic from transmitting to receiving, s=vt/2.

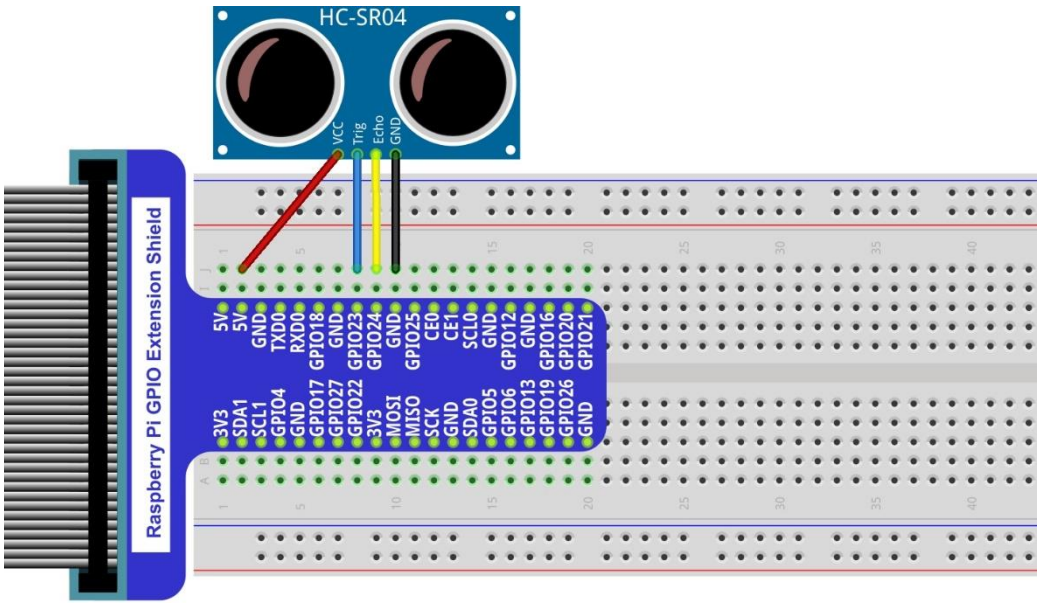
Circuit

Note that the voltage of ultrasonic module is 5V in the circuit.

Schematic diagram



Hardware connection



Code

C Code 24.1.1 UltrasonicRanging

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 24.1.1_UltrasonicRanging directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/24.1.1_UltrasonicRanging
```

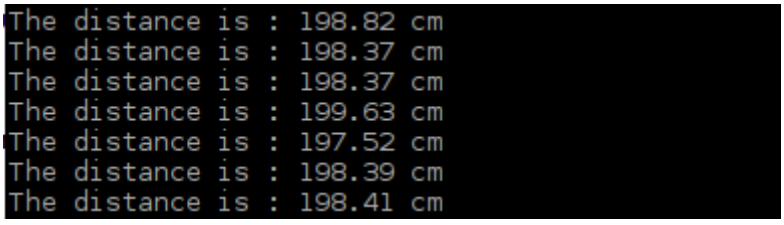
2. Use following command to compile "UltrasonicRanging.c" and generate executable file "UltrasonicRanging".

```
gcc UltrasonicRanging.c –o UltrasonicRanging –lwiringPi
```

3. Then run the generated file "UltrasonicRanging".

```
sudo ./UltrasonicRanging
```

After the program is executed, make the detector of ultrasonic ranging module aim at the plane of an object, then the distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:



The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220        // define the maximum measured distance
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured
distance
//function pulseIn: obtain pulse time of a pin
int pulseIn(int pin, int level, int timeout);
float getSonar(){   // get the measurement results of ultrasonic module,with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //trigPin send 10us high level
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; // the sound speed is 340m/s,and
calculate distance
    return distance;
}
```

```
23
24  int main(){
25      printf("Program is starting ... \n");
26      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
27          printf("setup wiringPi failed !");
28          return 1;
29      }
30      float distance = 0;
31      pinMode(trigPin,OUTPUT);
32      pinMode(echoPin,INPUT);
33      while(1){
34          distance = getSonar();
35          printf("The distance is : %.2f cm\n",distance);
36          delay(1000);
37      }
38      return 1;
39  }
```

First, define the pins and the maximum measurement distance.

```
#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220        //define the maximum measured distance
```

If the module does not return high level, we can not wait forever. So we need to calculate the lasting time over maximum distance, that is, time Out. **timOut= 2*MAX_DISTANCE/100/340*1000000**. The constant part behind is approximately equal to 58.8.

```
#define timeOut MAX_DISTANCE*60
```

Subfunction **getSonar** () function is used to start the ultrasonic module for a measurement, and return the measured distance with unit cm. In this function, first let trigPin send 10us high level to start the ultrasonic module. Then use **pulseIn** () to read ultrasonic module and return the duration of high level. Finally calculate the measured distance according to the time.

```
float getSonar(){   // get the measurement results of ultrasonic module,with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //trigPin send 10us high level
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; // the sound speed is 340m/s,and
calculate distance
    return distance;
}
```

Finally, in the while loop of main function, get the measurement distance and print it out constantly.

```
    while(1){
        distance = getSonar();
        printf("The distance is : %.2f cm\n",distance);
        delay(1000);
    }
```

About function **pulseIn()**:

int pulseIn(int pin, int level, int timeout);

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

Python Code 24.1.1 UltrasonicRanging

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 24.1.1_UltrasonicRanging directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/24.1.1_UltrasonicRanging
```

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

```
python UltrasonicRanging.py
```

After the program is executed, make the detector of ultrasonic ranging module aim at the plane of an object, then the distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.75 cm
The distance is : 199.22 cm
The distance is : 198.42 cm
The distance is : 198.74 cm
The distance is : 198.37 cm
The distance is : 198.47 cm
The distance is : 198.41 cm
```

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  trigPin = 16
5  echoPin = 18
6  MAX_DISTANCE = 220          #define the maximum measured distance
7  timeOut = MAX_DISTANCE*60   #calculate timeout according to the maximum measured distance
8  def pulseIn(pin,level,timeOut): # function pulseIn: obtain pulse time of a pin
9      t0 = time.time()
10     while(GPIO.input(pin) != level):
11         if((time.time() - t0) > timeOut*0.000001):
12             return 0;
13     t0 = time.time()
14     while(GPIO.input(pin) == level):
15         if((time.time() - t0) > timeOut*0.000001):
16             return 0;
17     pulseTime = (time.time() - t0)*1000000
```

```
18      return pulseTime
19
20  def getSonar():     #get the measurement results of ultrasonic module,with unit: cm
21      GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
22      time.sleep(0.00001)     #10us
23      GPIO.output(trigPin,GPIO.LOW)
24      pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
25      distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and
26  calculate distance
27      return distance
28
29  def setup():
30      print 'Program is starting...'
31      GPIO.setmode(GPIO.BOARD)        #numbers GPIOs by physical location
32      GPIO.setup(trigPin, GPIO.OUT)   #
33      GPIO.setup(echoPin, GPIO.IN)    #
34
35  def loop():
36      GPIO.setup(11,GPIO.IN)
37      while(True):
38          distance = getSonar()
39          print "The distance is : %.2f cm"%(distance)
40          time.sleep(1)
41
42  if __name__ == '__main__':     #program start from here
43      setup()
44      try:
45          loop()
46      except KeyboardInterrupt:  #when 'Ctrl+C' is pressed, the program will exit
47          GPIO.cleanup()         #release resource
```

First, define the pins and the maximum measurement distance.

```
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          # define the maximum measured distance
```

If the module does not return high level, we can not wait forever. So we need to calculate the lasting time over maximum distance, that is, time Out. **timOut= 2*MAX_DISTANCE/100/340*1000000**. The constant part behind is approximately equal to 58.8.

```
timeOut = MAX_DISTANCE*60
```

Subfunction **getSonar** () function is used to start the ultrasonic module for a measurement, and return the measured distance with unit cm. In this function, first let trigPin send 10us high level to start the ultrasonic module. Then use **pulseIn** () to read ultrasonic module and return the duration of high level. Finally calculate

the measured distance according to the time.

```
def getSonar():     #get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
    time.sleep(0.00001)     #10us
    GPIO.output(trigPin,GPIO.LOW)
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and
calculate distance
    return distance
```

Finally, in the while loop of main function, get the measurement distance and print it out constantly.

```
    while(True):
        distance = getSonar()
        print "The distance is : %.2f cm"%(distance)
        time.sleep(1)
```

About function `def pulseIn(pin,level,timeOut)` :

def pulseIn(pin,level,timeOut):

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

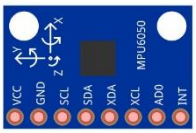
Chapter 25 Attitude Sensor MPU6050

In this chapter, we will learn an attitude sensor which integrates accelerometer and gyroscope, MPU6050.

Project 25.1 Read MPU6050

We will read acceleration data and gyroscope data of MPU6050 in this experiment.

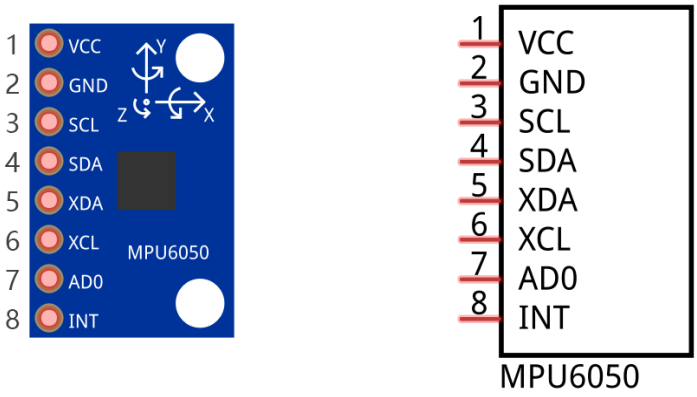
Component List

Raspberry Pi 3B x1 GPIO Expansion Board & Wire x1 BreadBoard x1	HC SR501 x1 
Jumper M/M x4 	

Component knowledge

MPU6050

MPU6050 is a sensor which integrates 3 axis accelerometer, 3 axes angular accelerometer (called gyroscope) and 1 digital attitude processor (DMP). The range of accelerometer and gyroscope of MPU6050 can be changed. A digital temperature sensor with wide range and high precision is integrated within it for temperature compensation, and the temperature value can be also read out. The MPU6050 module follows I2C communication protocol and the default address is 0x68.



The port description of the MPU6050 module is as follows:

Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication data pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69
INT	8	Output interrupt pin

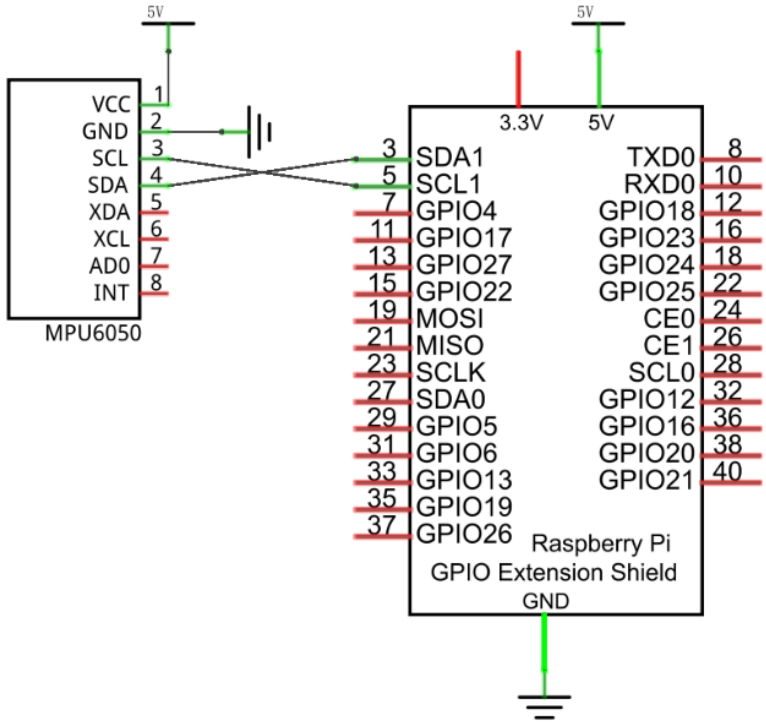
For more detail, please refer to dataheet.

MPU6050 is widely used in the field of balancing vehicles, aircraft and others which need to control the attitude.

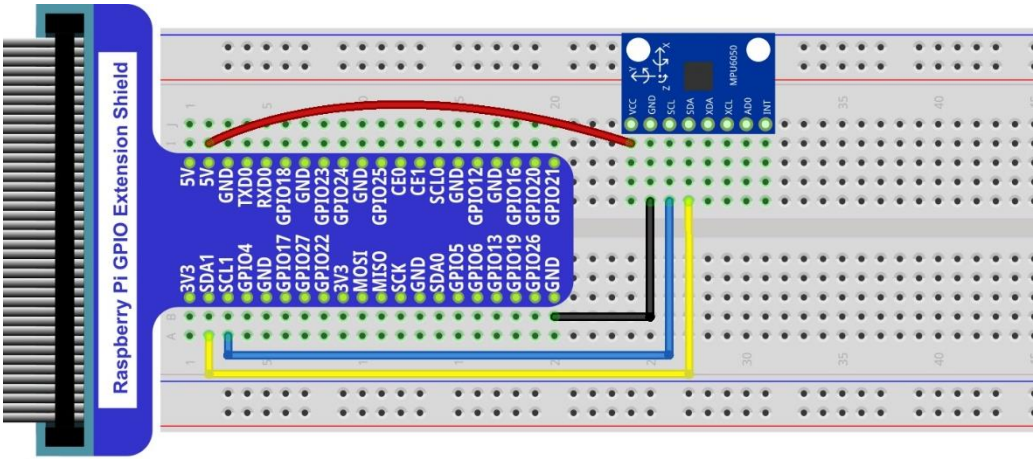
Circuit

Note that the power supply voltage for MPU6050 module is 5V in the circuit.

Schematic diagram



Hardware connection



Code

In this experiment, we will read the acceleration data and gyroscope data of MPU6050, and print them out.

C Code 25.1.1 MPU6050RAW

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 25.1.1_MPU6050RAW directory of C code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/24.1.1_MPU6050RAW
```

2. Use following command to compile "MPU6050RAW.c", "MPU6050.cpp" and "I2Cdev.cpp", and generate executable file "MPU6050RAW".

```
gcc MPU6050RAW.cpp MPU6050.cpp I2Cdev.cpp –o MPU6050RAW
```

3. Then run the generated file "MPU6050RAW".

```
sudo ./MPU6050RAW
```

After the program is excuted, the terminal will display the original acceleration and gyroscope data of MPU6050, as well as the conversion to gravity acceleration and angular velocity as the unit of data. As shown in the following figure:

```
a/g:   1360    120  15840     -320   -193   -114
a/g: 0.08 g 0.01 g 0.97 g   -2.44 d/s -1.47 d/s -0.87 d/s
a/g:   1108    -88  15476     -354   -252   -115
a/g: 0.07 g -0.01 g 0.94 g   -2.70 d/s -1.92 d/s -0.88 d/s
a/g:   1344   -264  15764     -396   -236   -121
a/g: 0.08 g -0.02 g 0.96 g   -3.02 d/s -1.80 d/s -0.92 d/s
a/g:   1440   -180  15720     -375   -162   -114
a/g: 0.09 g -0.01 g 0.96 g   -2.86 d/s -1.24 d/s -0.87 d/s
a/g:   1436     56  16608     -400   -154   -136
a/g: 0.09 g 0.00 g 1.01 g   -3.05 d/s -1.18 d/s -1.04 d/s
a/g:   1008    144  14940     -345   -142   -129
a/g: 0.06 g 0.01 g 0.91 g   -2.63 d/s -1.08 d/s -0.98 d/s
```

The following is the program code:

```
1  #include <stdio.h>
2  #include <stdint.h>
3  #include <unistd.h>
4  #include "I2Cdev.h"
5  #include "MPU6050.h"
6
7  MPU6050 accelgyro;      //instantiate a MPU6050 class object
8
9  int16_t ax, ay, az;     //store acceleration data
10 int16_t gx, gy, gz;     //store gyroscope data
11
12 void setup() {
13     // initialize device
14     printf("Initializing I2C devices...\n");
15     accelgyro.initialize();     //initialize MPU6050
16
17     // verify connection
18     printf("Testing device connections...\n");
```

```
19      printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" : "MPU6050
20  connection failed\n");
21  }
22
23  void loop() {
24      // read raw accel/gyro measurements from device
25      accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
26      // display accel/gyro x/y/z values
27      printf("a/g: %6hd %6hd %6hd   %6hd %6hd %6hd\n",ax,ay,az,gx,gy,gz);
28      printf("a/g: %.2f g %.2f g %.2f g   %.2f d/s %.2f d/s %.2f d/s
29  \n",(float)ax/16384,(float)ay/16384,(float)az/16384,
30          (float)gx/131,(float)gy/131,(float)gz/131);
31  }
32
33  int main()
34  {
35      setup();
36      while(1){
37          loop();
38      }
39      return 0;
40  }
```

Two library files "**MPU6050**.h" and "**I2Cdev**.h" are used in the code. They will be compiled as others. Class MPU6050 is used to operate the MPU6050. When used, first instantiate an object.

```
MPU6050 accelgyro;
```

In the setup function, the MPU6050 is initialized and the result of the initialization will be judged.

```
void setup() {
    // initialize device
    printf("Initializing I2C devices...\n");
    accelgyro.initialize();     //initialize MPU6050

    // verify connection
    printf("Testing device connections...\n");
    printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" : "MPU6050
connection failed\n");
}
```

In the loop function, read the original data of MPU6050 and print them out, and then convert the original data into the corresponding acceleration and angular velocity, then print the converted data out.

```
void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    // display accel/gyro x/y/z values
    printf("a/g: %6hd %6hd %6hd   %6hd %6hd %6hd\n",ax,ay,az,gx,gy,gz);
```

```
    printf("a/g: %.2f %.2f %.2f %.2f %.2f %.2f d/s %.2f d/s %.2f d/s
\n",(float)ax/16384,(float)ay/16384,(float)az/16384,
        (float)gx/131,(float)gy/131,(float)gz/131);
}
```

Finally, the main functions, call setup function and loop function respectively.

```
int main()
{
    setup();
    while(1){
        loop();
    }
    return 0;
}
```

About class MPU6050：

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

MPU6050()/MPU6050(uint8_t address):

Constructor. The parameter is I2C address, and the default I2C address is 0x68.

void initialize();

Initialization function, used to wake up MPU6050. Range of accelerometer is ±2g and range of gyroscope is ±250 degrees/sec.

void getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t* gz);

Get the original data of accelerometer and gyroscope.

int16_t getTemperature();

Get the original temperature data of MPU6050.

For details about more relevant member functions, pleases refer to MPU6050.h or visit:

https://github.com/jrowberg/i2cdevlib

Python Code 25.1.1 MPU6050RAW

First observe the experimental phenomenon, and then analyze the code.

1. Use cd command to enter 25.1.1_MPU6050RAW directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/25.1.1_MPU6050RAW
```

2. Use python command to execute code "MPU6050RAW.py".

```
python MPU6050RAW.py
```

After the program is excuted, the terminal will display the original acceleration and gyroscope data of MPU6050, as well as the conversion to gravity acceleration and angular velocity as the unit of data. As shown in the following figure:

```
a/g:1326        -160    16548   -48     -25     -16
a/g:0.08 g      -0.01 g 1.01 g  -0.37 d/s       -0.19 d/s       -0.12 d/s
a/g:1174        -116    15972   -44     -25     -17
a/g:0.07 g      -0.01 g 0.97 g  -0.34 d/s       -0.19 d/s       -0.13 d/s
a/g:1134        -130    16066   -45     -21     -17
a/g:0.07 g      -0.01 g 0.98 g  -0.34 d/s       -0.16 d/s       -0.13 d/s
a/g:1234        -76     15976   -45     -30     -16
a/g:0.08 g      -0.00 g 0.98 g  -0.34 d/s       -0.23 d/s       -0.12 d/s
a/g:996 -88     15748   -45     -22     -16
a/g:0.06 g      -0.01 g 0.96 g  -0.34 d/s       -0.17 d/s       -0.12 d/s
a/g:1196        -174    16182   -46     -25     -15
a/g:0.07 g      -0.01 g 0.99 g  -0.35 d/s       -0.19 d/s       -0.11 d/s
```

The following is the program code:

```
1  import MPU6050
2  import time
3
4  mpu = MPU6050.MPU6050()      #instantiate a MPU6050 class object
5  accel = [0]*3                #store accelerometer data
6  gyro = [0]*3                 #store gyroscope data
7  def setup():
8      mpu.dmp_initialize()     #initialize MPU6050
9
10 def loop():
11     while(True):
12         accel = mpu.get_acceleration()      #get accelerometer data
13         gyro = mpu.get_rotation()           #get gyroscope data
14         print"a/g:%d\t%d\t%d\t%d\t%d\t%d
15 "%(accel[0],accel[1],accel[2],gyro[0],gyro[1],gyro[2])
16         print"a/g:%.2f g\t%.2f g\t%.2f g\t%.2f d/s\t%.2f d/s\t%.2f
17 d/s"%(accel[0]/16384.0,accel[1]/16384.0,
18             accel[2]/16384.0,gyro[0]/131.0,gyro[1]/131.0,gyro[2]/131.0)
19         time.sleep(0.1)
20
21 if __name__ == '__main__':     # Program start from here
22     print"Program is starting ... "
23     setup()
24     try:
25         loop()
```

```
26      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the program will exit.
27          pass
```

A module "**MPU6050**.py" is used in the code. The module include a class used to operate MPU6050. When used, first instantiate an object.

```
mpu = MPU6050.MPU6050()
```

In the setup function, the MPU6050 is initialized.

```
def setup():
    mpu.dmp_initialize()
```

In the loop function, read the original data of MPU6050 and print them out, and then convert the original data into the corresponding acceleration and angular velocity, then print the converted data out.

```
def loop():
    while(True):
        accel = mpu.get_acceleration()      #get accelerometer data
        gyro = mpu.get_rotation()           #get gyroscope data
        print"a/g:%d\t%d\t%d\t%d\t%d\t%d
"%(accel[0],accel[1],accel[2],gyro[0],gyro[1],gyro[2])
        print"a/g:%.2f g\t%.2f g\t%.2f g\t%.2f d/s\t%.2f d/s\t%.2f
d/s"%(accel[0]/16384.0,accel[1]/16384.0,
            accel[2]/16384.0,gyro[0]/131.0,gyro[1]/131.0,gyro[2]/131.0)
        time.sleep(0.1)
```

About class MPU6050：

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

```
def __init__(self, a_bus=1, a_address=C.MPU6050_DEFAULT_ADDRESS,
                a_xAOff=None, a_yAOff=None, a_zAOff=None, a_xGOff=None,
                a_yGOff=None, a_zGOff=None, a_debug=False):
```

Constructor

```
def dmp_initialize(self):
```

Initialization function, used to wake up MPU6050. Range of accelerometer is ±2g and range of gyroscope is ±250 degrees/sec.

```
def get_acceleration(self):   &   def get_rotation(self):
```

Get the original data of accelerometer and gyroscope.

For details of more relevant member functions, please refer to MPU6050.py.

Chapter 26 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 26.1 Remote LED

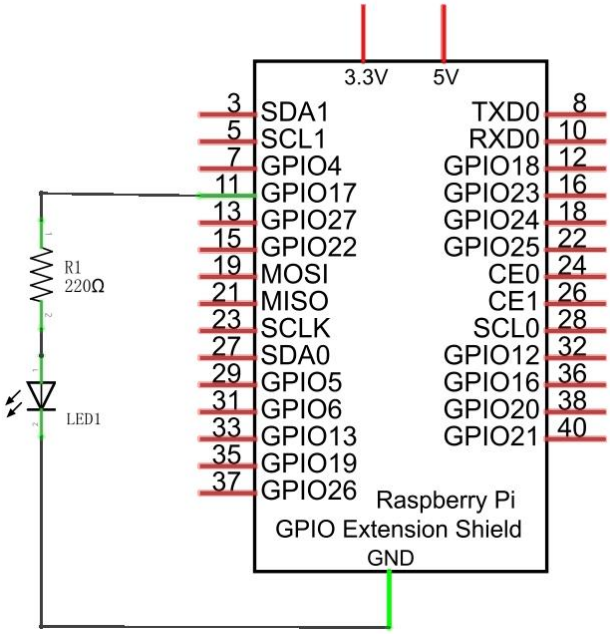
In this experiment, we need build a WebIOPi service, and then control the RPi GPIO to control a LED through Web browser of phone or PC.

Component List

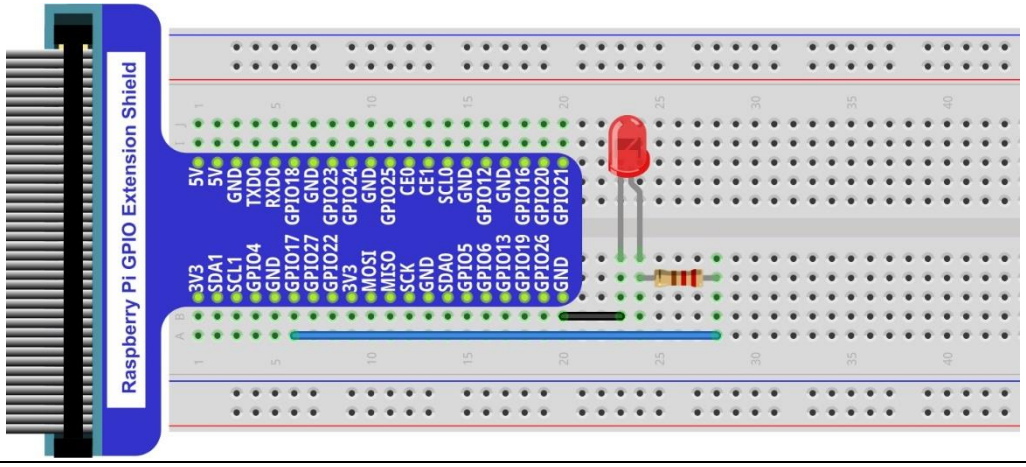
<table>
<tr><td>Raspberry Pi 3B x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td rowspan="2">LED x1
  </td><td rowspan="2">Resistor 220Ω x1
  </td></tr>
<tr><td>Jumper M/M x2
  </td></tr>
</table>

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
```

2. 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
```

3. Patch for Raspberry Pi B+, 2B and 3B.

```
wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch
patch -p1 -i webiopi-pi2bplus.patch
```

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

```
sudo ./setup.sh
```

Run

After the installation is completed, you can use the webiopi command to start running.

```
$ sudo webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]

    Options:
      -h, --help            Display this help
      -c, --config  file    Load config from file
      -l, --log     file    Log to file
      -s, --script  file    Load script from file
      -d, --debug           Enable DEBUG

    Arguments:
      port                  Port to bind the HTTP Server
```

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

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

The Port is 8000 in default.

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

Access WebIOPi over local network

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

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

Then, enter the main control interface:

WebIOPi Main Menu

GPIO Header

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

GPIO List

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

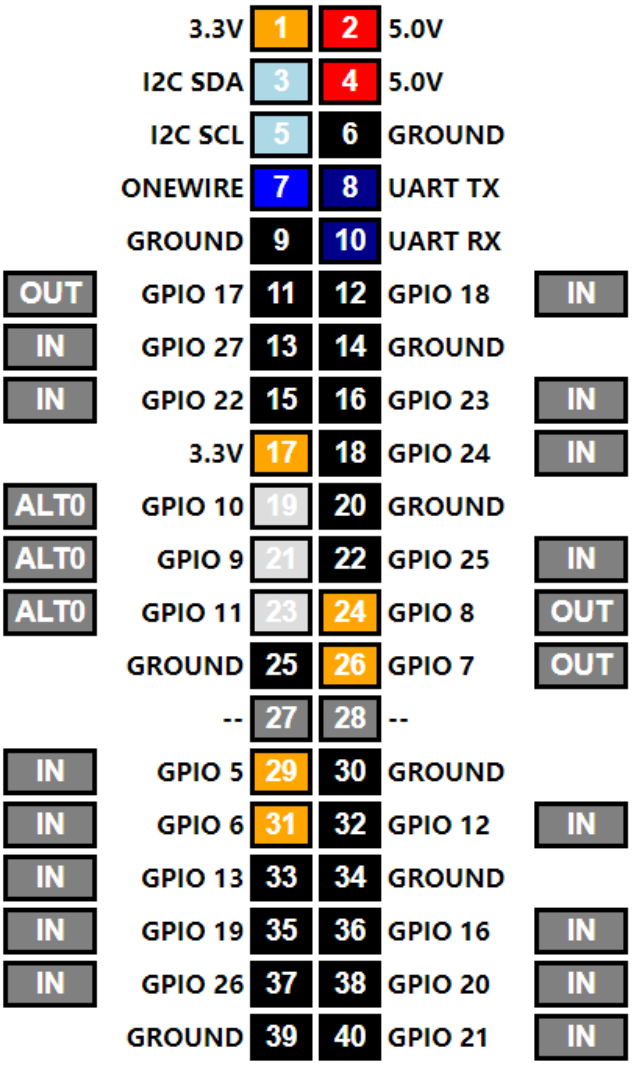
Serial Monitor

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

Devices Monitor

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

Click on GPIO Header to enter the GPIO control interface.



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.

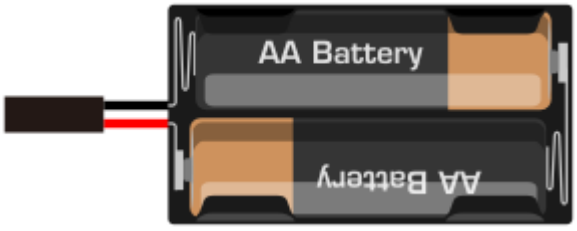
Chapter 27 Solder Circuit Board

From previous chapters, we have learned the knowledge of electronic circuit and component, and build a variety of circuits. Now, we will take a further step, making a piece of circuit board on your own.
We will use the general board to solder the circuit and components. And when this chapter is over, it's our hope to help you master the idea of how to design your own circuit, build and print circuit boards.
To finish this chapter, you need to prepare the necessary soldering equipments, including electric iron and solder. We have prepared the general board for you, please pay attention to safety when you operate these experiments.

Project 27.1 Solder a Buzzer

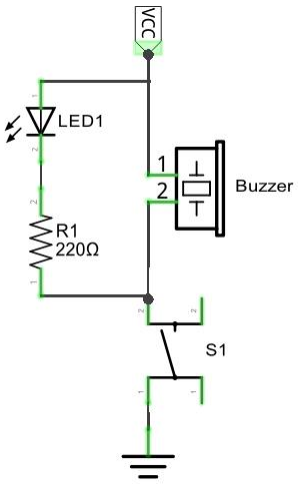
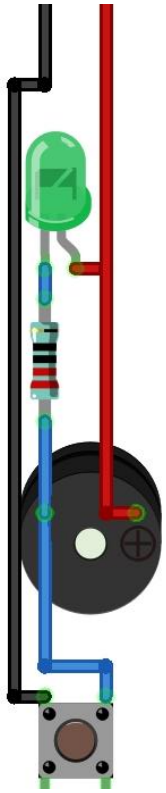
We have tried to use the buzzer from previous chapter, and now we will solder a circuit that when the button is pressed, the buzzer sounds
This circuit doesn't need programming and can work when it is powered on. And when the button is not pressed, there is no power consumption.
You can install it on your bike, bedroom door or any other places where it is needed.

Component list

Pin header x2 	LED x1 	Resistor 220Ω x1 	Active buzzer x1 	Push button x1 
AA Battery Holder x1 				

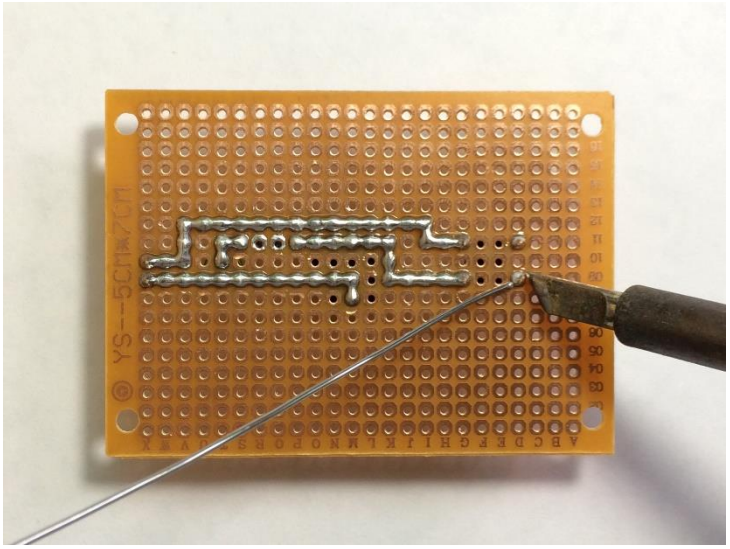
Circuit

We will solder the following circuit on the genneral board.

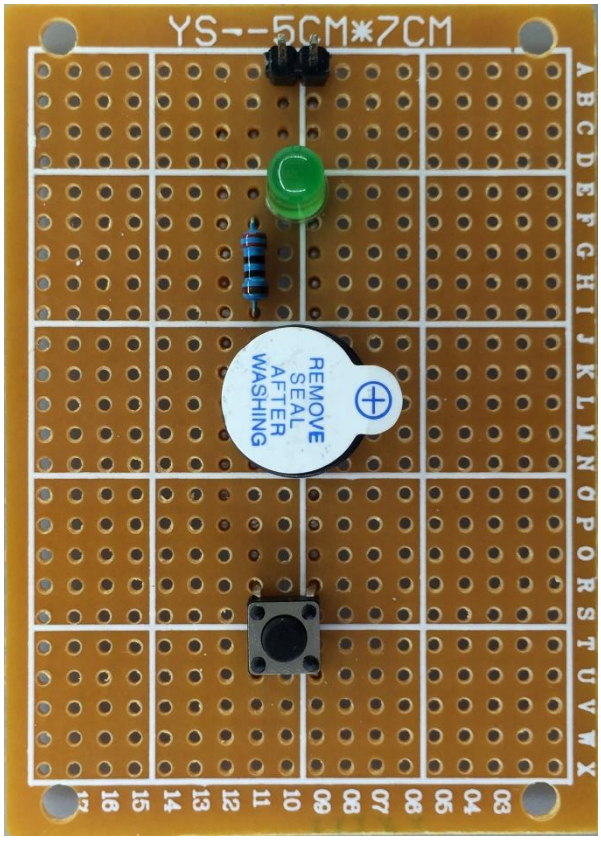
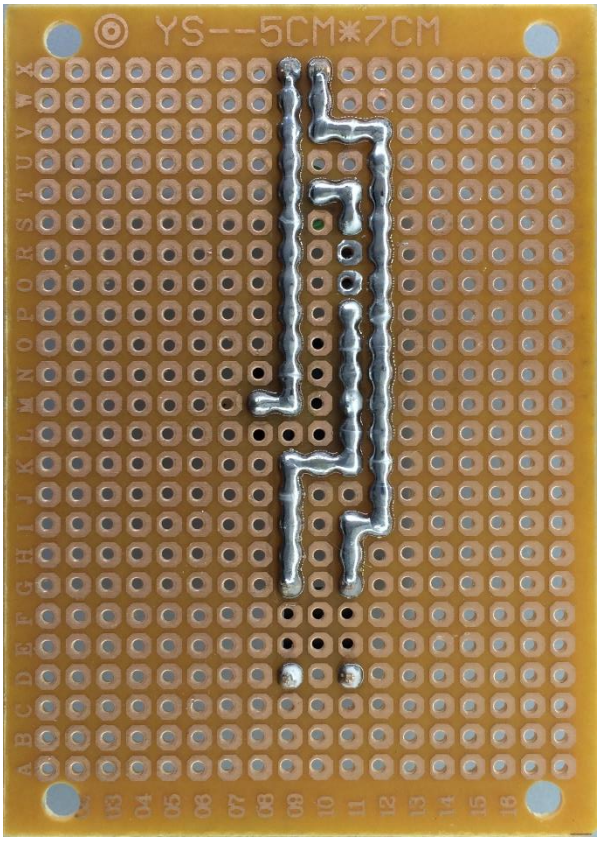
Schematic diagram	Hardware connection
	

Solder the Circuit

Insert the components in the general board and solder the circuit on the back.

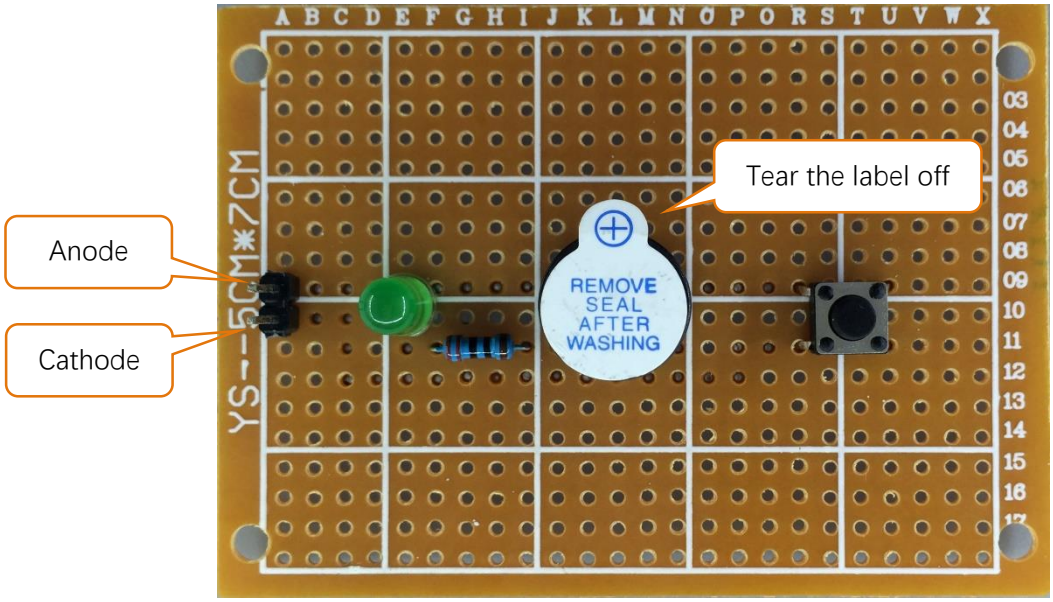


Effect diagram after soldering:

Front	Back
	

Test circuit

Connect the circuit board to power supply (3~5V). You can use Raspberry Pi board or battery box as the power supply.

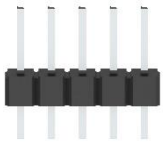
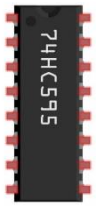


Press the push button after connecting the power, and then the buzzer will make a sound.

Project 27.2 Solder a Flowing Water Light

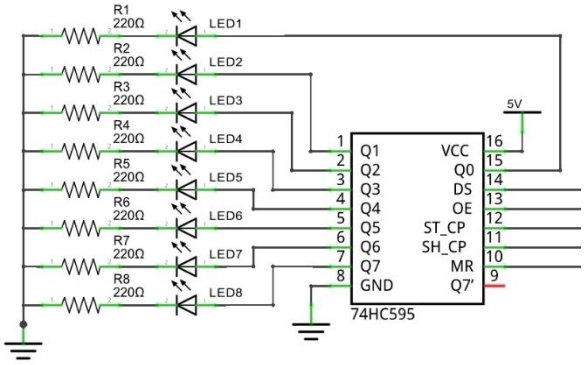
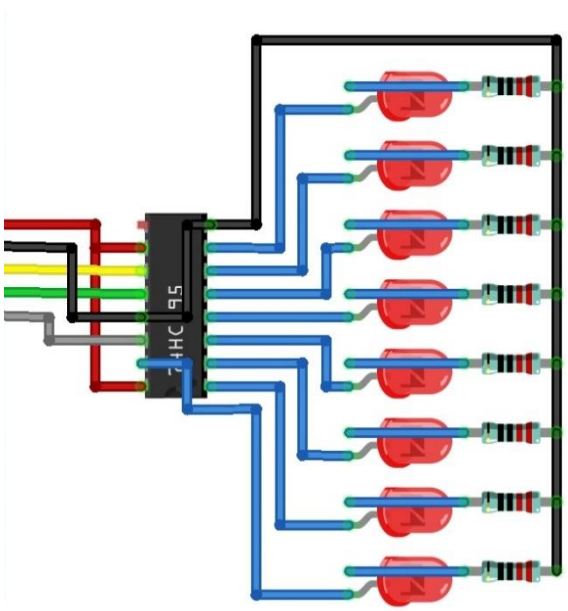
From previous chapter, we have learned to make a flowing water light with LED. Now, we will solder a circuit board, and use the improved code to make a more interesting flowing water light.

Component list

Pin header x5	Resistor 220Ω x8	LED x8	74HC595 x1
			

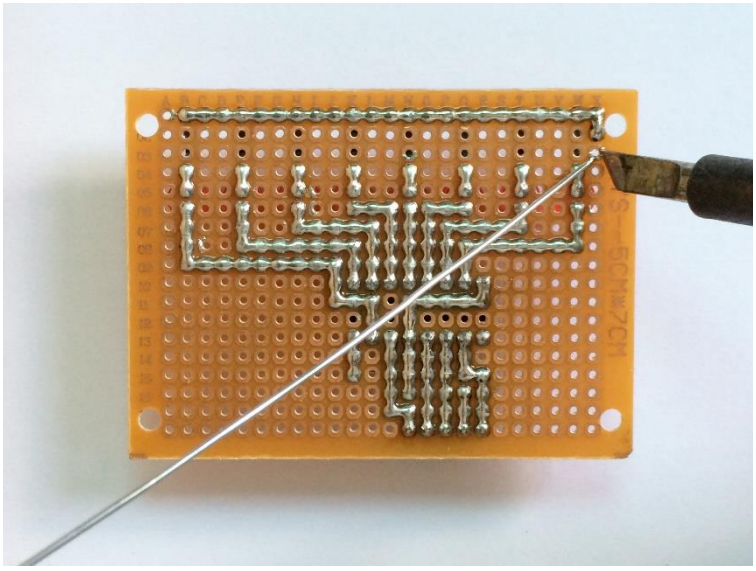
Circuit

Solder the following circuit on the general board.

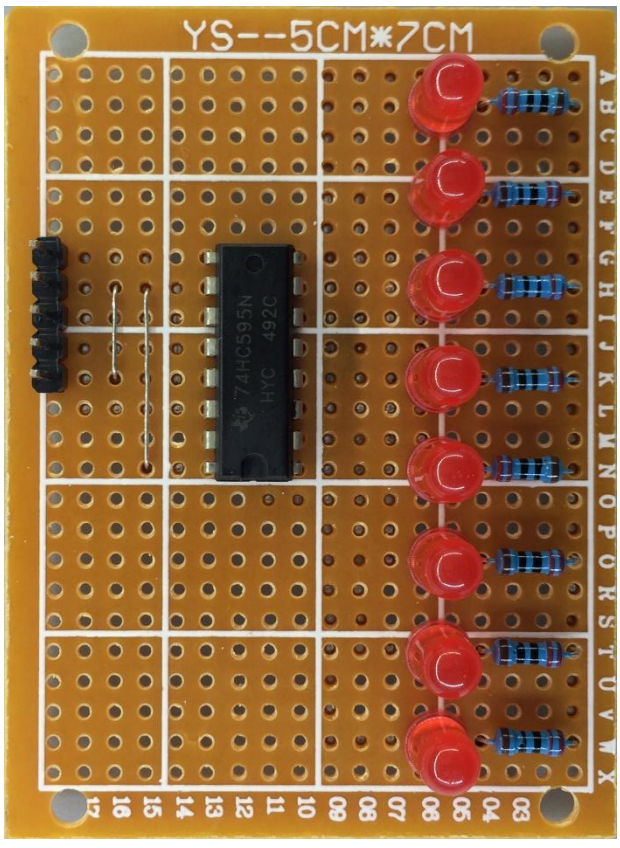
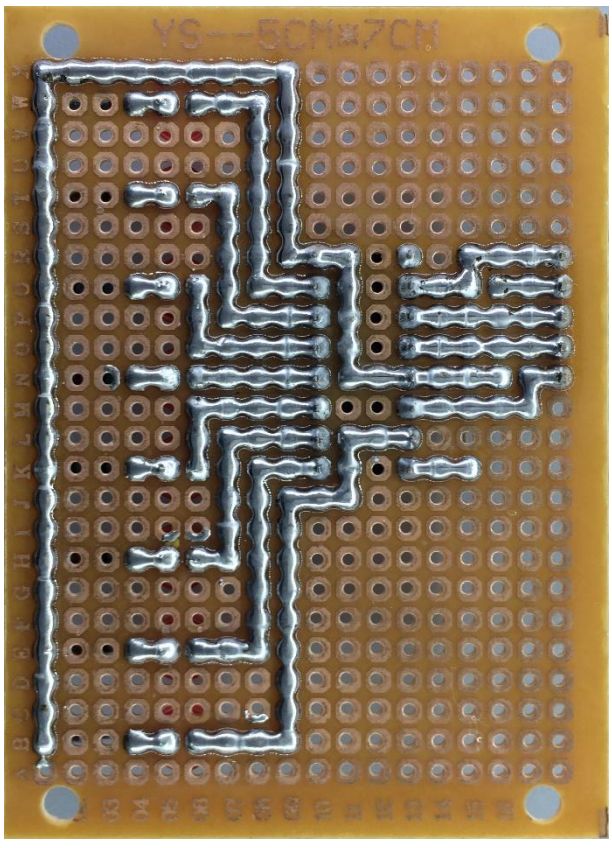
Schematic diagram	Hardware connection
	

Solder the Circuit

Insert the components in the general board, and solder the circuit on the back.

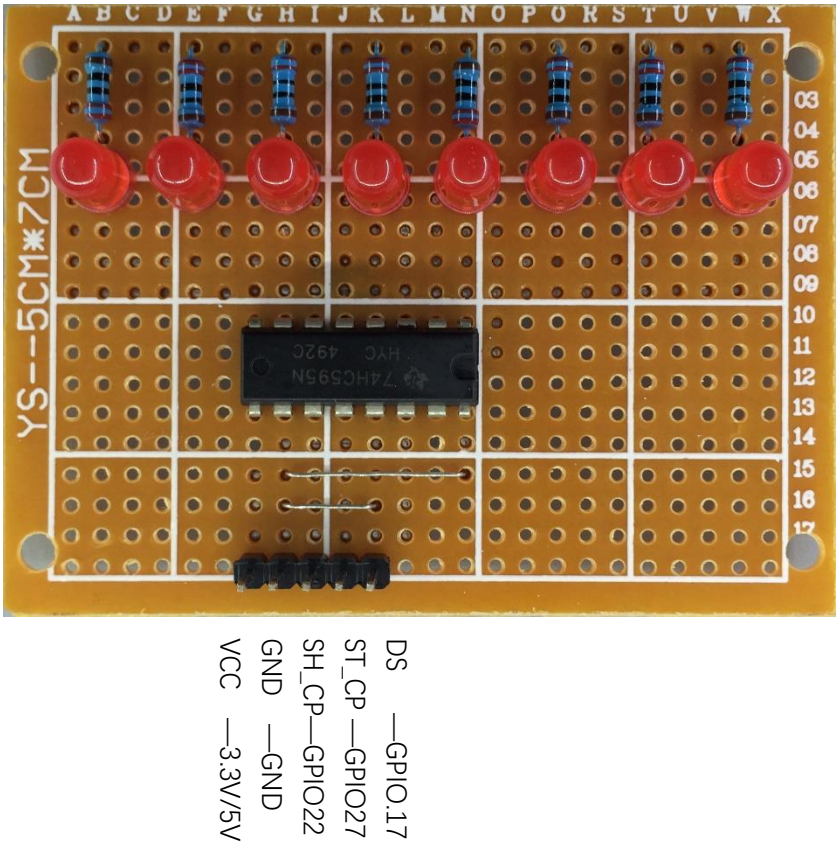


Effect diagram after soldering:

Front	Back
	

Connect the Circuit

Connenct the board to Raspberry Pi with jumper wire in the following way.



Code

This is the third time we have maked flowing water light. In this experiment, we solder a completely new new circuit for flowing water light. Additionally, the program is also different from previous ones. When this light flows, it will bring a long tail.

C Code 27.2.1 LightWater03

First observe the experimental phenomenon, and then analyze the code.

4. Use the cd command to enter 27.2.1_LightWater03 directory of C code.

```
cd Freenove_Super_Ultimate_Kit_for_Raspberry_Pi/Code/C_Code/27.2.1_LightWater03
```

5. Use following command to compile “LightWater03.c” and generate executable file “LightWater03”.

```
gcc LightWater03.c –o LightWater03 –lwiringPi
```

6. Then run the generated file “LightWater03”.

```
sudo ./LightWater03
```

After the program is executed, the LEDs will light up in the form of flowing water and carry a long tail.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)
//Define an array to save the pulse width of LED. Output the signal to the 8 adjacent
LEDs in order.
const int pluseWidth[]={0,0,0,0,0,0,0,0,64,32,16,8,4,2,1,0,0,0,0,0,0,0,0,0};
void outData(int8_t data){
    digitalWrite(latchPin,LOW);
    shiftOut(dataPin,clockPin,LSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
int main(void)
{
    int i,j,index;  //index:current position in array pluseWidth
    int moveSpeed = 100;    //move speed delay, the greater, the slower
    long lastMove;          //Record the last time point of the move
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    index = 0;      //Starting from the array index 0
    lastMove = millis();    //the start time
    while(1){
        if(millis() - lastMove > moveSpeed) { //speed control
            lastMove = millis();    //Record the time point of the move
            index++;        //move to next
            if(index > 15) index = 0;   //index to 0
        }
        for(i=0;i<64;i++){      //The cycle of PWM is 64 cycles
            int8_t data = 0;    //This loop of output data
            for(j=0;j<8;j++){   //Calculate the output state of this loop
                if(i < pluseWidth[index+j]){    //Calculate the LED state according to
the pulse width
                    data |= 0x01<<j ;   //Calculate the data
                }
```

```
            }
            outData(data);      //Send the data to 74HC595
        }
    }
    return 0;
}
```

We can see that this program is different from the previous one. We define an array to modulate different PWM pulse width for LEDs, so that different LEDs can emit different brightness. Starting from the array index 0, take an array of 8 adjacent numbers as the LED duty cycle and output it at a time. Increases the starting index number in turn, then it will creat a flowing effect.

```
const int pluseWidth[]={0,0,0,0,0,0,0,0,64,32,16,8,4,2,1,0,0,0,0,0,0,0,0,0};
```

By recording the moving time point to control the speed of the movement of index number, namely, control the flowing speed of flowing water light. Variable moveSpeed saves the time interval of each move, and the greater the value, the slower the flowing rate. On the contrary, the faster the flowing.

```
        if(millis() - lastMove > moveSpeed) { //speed control
            lastMove = millis();    //Record the time point of the move
            index++;        //move to next
            if(index > 15) index = 0;   //index to 0
        }
```

Finally, in a “for” cycle with i=64, modulate output pulse width of PWM square wave. And the progress, from the beginning of implementing the for cycle to the end, is a PWM cycle. In the cycle, there is anthoer for cycle with j=8. And in this cycle, compare the cycle number “i” to the value of the array to determine output high or low level. At last, the data will be sent to 74HC595.

```
        for(i=0;i<64;i++){      //The cycle of PWM is 64 cycles
            int8_t data = 0;    //This loop of output data
            for(j=0;j<8;j++){   //Calculate the output state of this loop
                if(i < pluseWidth[index+j]){    //Calculate the LED state according to
the pulse width
                    data |= 0x01<<j ;   //Calculate the data
                }
            }
            outData(data);      //Send the data to 74HC595
        }
```

Pyhton Code 27.2.1 LightWater03

First observe the experimental phenomenon, and then analyze the code.

3. Use the cd command to enter 27.2.1_LightWater03 directory of Python code.

```
cd Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/27.2.1_LightWater03
```

4. Use python command to execute python code "LightWater03.py".

```
python LightWater03.py
```

After the program is executed, the LEDs will light up in the form of flowing water and carry a long tail.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 11      #DS Pin of 74HC595(Pin14)
latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
clockPin = 15       #SH_CP Pin of 74HC595(Pin11)
#Define an array to save the pulse width of LED. Output the signal to the 8 adjacent LEDs
in order.
pluseWidth = [0,0,0,0,0,0,0,0,64,32,16,8,4,2,1,0,0,0,0,0,0,0,0,0]

def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def outData(data):
    GPIO.output(latchPin,GPIO.LOW)
    shiftOut(dataPin,clockPin,LSBFIRST,data)
    GPIO.output(latchPin,GPIO.HIGH)

def loop():
    moveSpeed = 0.1 #move speed delay, the greater, the slower
    index = 0       #Starting from the array index 0
```

```
    lastMove = time.time()      #the start time
    while True:
        if(time.time() - lastMove > moveSpeed): #speed control
            lastMove = time.time()      #Record the time point of the move
            index +=1           #move to next
            if(index > 15):     #index to 0
                index = 0

        for i in range(0,64):   #The cycle of PWM is 64 cycles
            data = 0            #This loop of output data
            for j in range(0,8):    #Calculate the output state of this loop
                if(i < pluseWidth[j+index]):    #Calculate the LED state according to the
pulse width
                    data |= 1<<j    #Calculate the data
            outData(data)           #Send the data to 74HC595

def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
    GPIO.cleanup()

if __name__ == '__main__': # Program starting from here
    print 'Program is starting...'
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

We can see that this procedure is different from the previous running water lamp, we define an array for the modulation of different PWM LED pulse width, so that different LED have different brightness. Starting from the array index 0, each take an array of 8 adjacent numbers, as the LED duty cycle output, which in turn increases the index number, it will produce a flow effect.

```
pluseWidth = [0,0,0,0,0,0,0,0,64,32,16,8,4,2,1,0,0,0,0,0,0,0,0,0]
```

By recording the moving time point to control the speed of the movement of index number, namely, control the flowing speed of flowing water light. Variable moveSpeed saves the time interval of each move, and the greater the value, the slower the flowing rate. On the contrary, the faster the flowing.

```
        if(time.time() - lastMove > moveSpeed): #speed control
            lastMove = time.time()      #Record the time point of the move
            index +=1           #move to next
            if(index > 15):     #index to 0
                index = 0
```

Finally, in a “for” cycle with i=64, modulate output pulse width of PWM square wave. And the progress, from the beginning of implementing the for cycle to the end, is a PWM cycle. In the cycle, there is anthoer for cycle with j=8. And in this cycle, compare the cycle number “i” to the value of the array to determine output high or low level. At last, the data will be sent to 74HC595.

```
        for i in range(0,64):   #The cycle of PWM is 64 cycles
            data = 0            #This loop of output data
            for j in range(0,8):    #Calculate the output state of this loop
                if(i < pluseWidth[j+index]):    #Calculate the LED state according to the
pulse width
                    data |= 1<<j    #Calculate the data
            outData(data)           #Send the data to 74HC595
```

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.