

Getting Started

Thank you for choosing Freenove products!

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

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

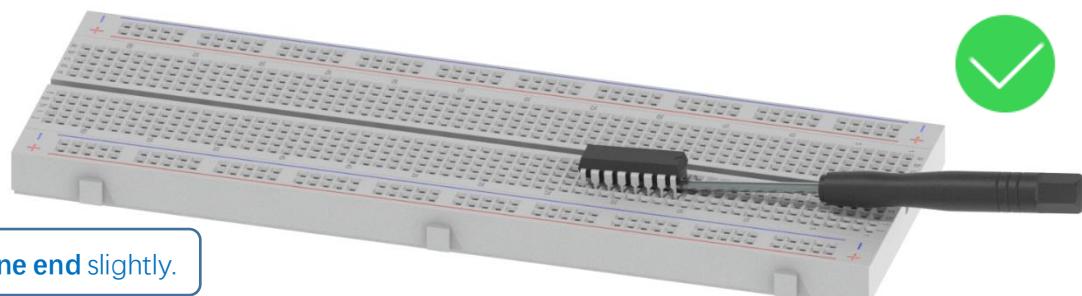
We recommend you to start with **Tutorial.pdf** first.

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

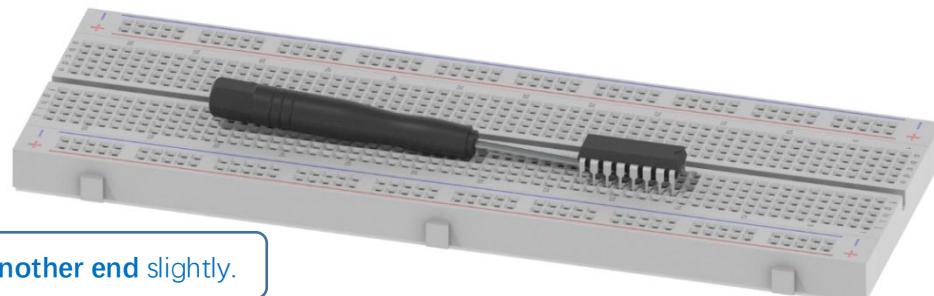
Remove the Chips

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

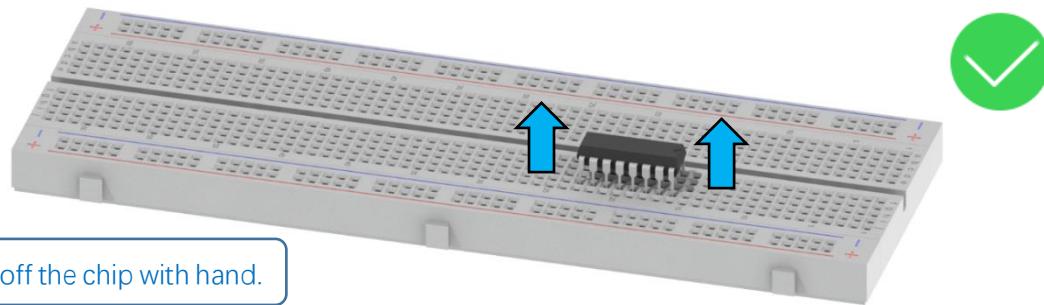
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.) Please find a tool (like a little screw driver) to handle them like below:



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



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



Avoid lifting one end with big angle directly.



Get Support and Offer Input

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

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

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

support@freenove.com

Safety and Precautions

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

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

cools down! When everything is safe and cool, review the product tutorial to identify the cause.

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

About Freenove

Freenove provides open source electronic products and services worldwide.

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

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

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

<http://www.freenove.com>

Copyright

All the files, materials and instructional guides provided are released under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). A copy of this license can be found in the folder containing the Tutorial and software files associated with this product.



This means you can use these resources in your own derived works, in part or completely, but **NOT for the intent or purpose of commercial use.**

Freenove brand and logo are copyright of Freenove Creative Technology Co., Ltd. and cannot be used without written permission.



Raspberry Pi® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

Contents

Getting Started	I
Remove the Chips.....	I
Safety and Precautions.....	II
About Freenove	III
Copyright	III
Contents.....	IV
Preface	1
Raspberry Pi.....	2
Installing an Operating System	9
Component List.....	9
Optional Components.....	11
Raspberry Pi OS	13
Getting Started with Raspberry Pi	18
Chapter 0 Preparation.....	27
Linux Command.....	27
Install WiringPi.....	30
Obtain the Project Code.....	32
Python2 & Python3.....	33
Chapter 1 LED	35
Project 1.1 Blink.....	35
Freenove Car, Robot and other products for Raspberry Pi	57
Chapter 2 Buttons & LEDs	58
Project 2.1 Push Button Switch & LED	58
Project 2.2 MINI Table Lamp	65
Chapter 3 LED Bar Graph	71
Project 3.1 Flowing Water Light.....	71
Chapter 4 Analog & PWM.....	77
Project 4.1 Breathing LED.....	77
Chapter 5 RGB LED.....	85
Project 5.1 Multicolored LED	86
Chapter 6 Buzzer.....	92
Project 6.1 Doorbell	92
Project 6.2 Alertor.....	99
(Important) Chapter 7 ADC	104
Project 7.1 Read the Voltage of Potentiometer.....	104
Chapter 8 Potentiometer & LED	120
Project 8.1 Soft Light.....	120
Chapter 9 Potentiometer & RGBLED	127
Project 9.1 Colorful Light.....	127
Chapter 10 Photoresistor & LED	135
Project 10.1 NightLamp.....	135

Chapter 11 Thermistor	143
Project 11.1 Thermometer	143
Chapter 12 Joystick	151
Project 12.1 Joystick	151
Chapter 13 Motor & Driver	159
Project 13.1 Control a DC Motor with a Potentiometer.....	159
Chapter 14 Relay & Motor	174
Project 14.1.1 Relay & Motor	174
Chapter 15 Servo	182
Project 15.1 Servo Sweep.....	182
Chapter 16 Stepper Motor	191
Project 16.1 Stepper Motor	191
Chapter 17 74HC595 & Bar Graph LED	202
Project 17.1 Flowing Water Light.....	202
Chapter 18 74HC595 & 7-Segment Display	211
Project 18.1 7-Segment Display	211
Project 18.2 4-Digit 7-Segment Display.....	218
Chapter 19 74HC595 & LED Matrix	231
Project 19.1 LED Matrix	231
Chapter 20 LCD1602	243
Project 20.1 I2C LCD1602	243
Chapter 21 Hygrothermograph DHT11	254
Project 21.1 Hygrothermograph.....	254
Chapter 22 Matrix Keypad	261
Project 22.1 Matrix Keypad	261
Chapter 23 Ultrasonic Ranging	271
Project 23.1 Ultrasonic Ranging.....	271
Chapter 24 RFID	280
Project 24.1 RFID.....	280
Chapter 25 Web IoT	299
Project 25.1 Remote LED.....	299
What's Next?	304

Preface

Raspberry Pi is a low cost, **credit card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is an incredibly capable little device that enables people of all ages to explore computing, and to learn how to program in a variety of computer languages like Scratch and Python. It is capable of doing everything you would expect from a desktop computer, such as browsing the internet, playing high-definition video content, creating spreadsheets, performing word-processing, and playing video games. For more information, you can refer to Raspberry Pi official [website](#). For clarification, this tutorial will also reference Raspberry Pi as RPi, RPI and RasPi.

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

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

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

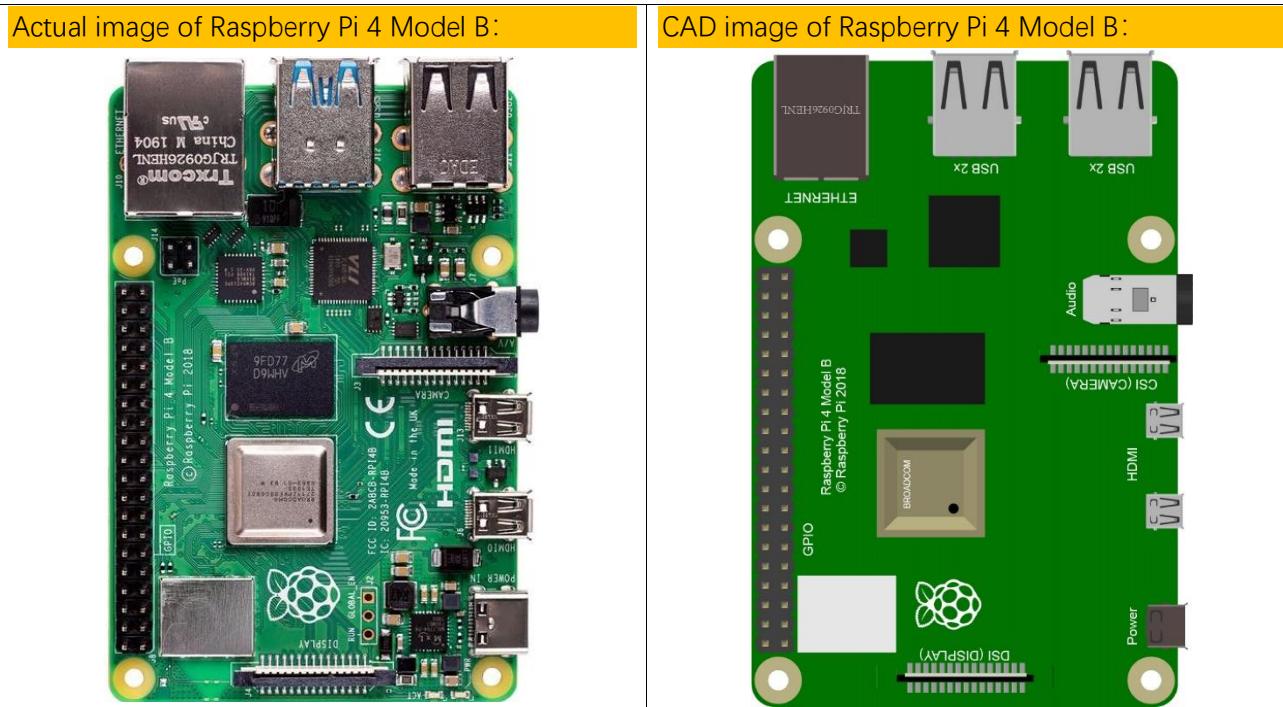
support@freenove.com

Raspberry Pi

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

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

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



Actual image of Raspberry Pi 3 Model B+:



CAD image of Raspberry Pi 3 Model B+:



Actual image of Raspberry Pi 3 Model B:



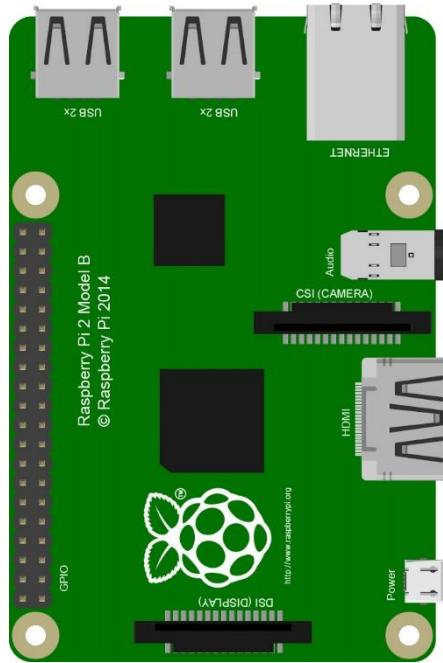
CAD image of Raspberry Pi 3 Model B:



Actual image of Raspberry Pi 2 Model B:



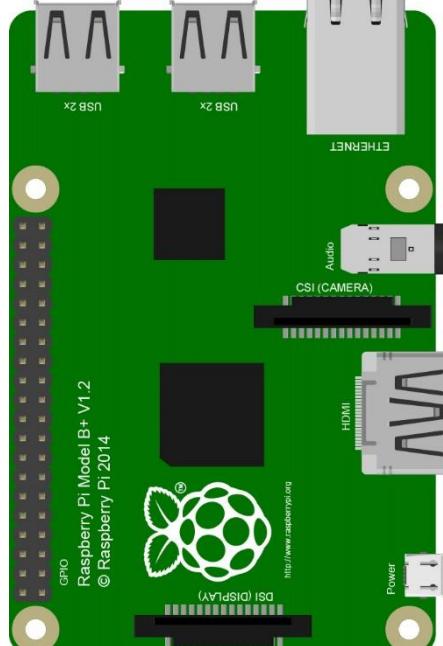
CAD image of Raspberry Pi 2 Model B:



Actual image of Raspberry Pi 1 Model B+:



CAD image of Raspberry Pi 1 Model B+:



Actual image of Raspberry Pi 3 Model A+:



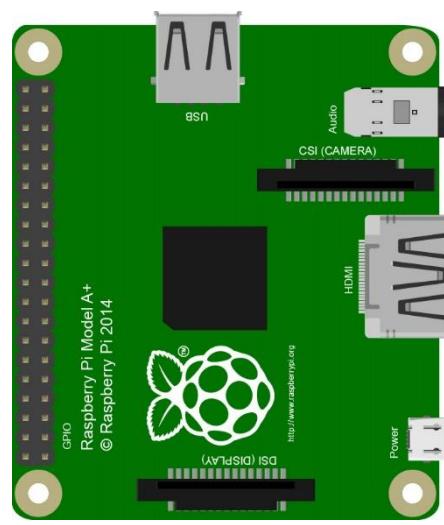
CAD image of Raspberry Pi 3 Model A+:



Actual image of Raspberry Pi 1 Model A+:



CAD image of Raspberry Pi 1 Model A+:



Actual image of Raspberry Pi Zero W:



CAD image of Raspberry Pi Zero W:



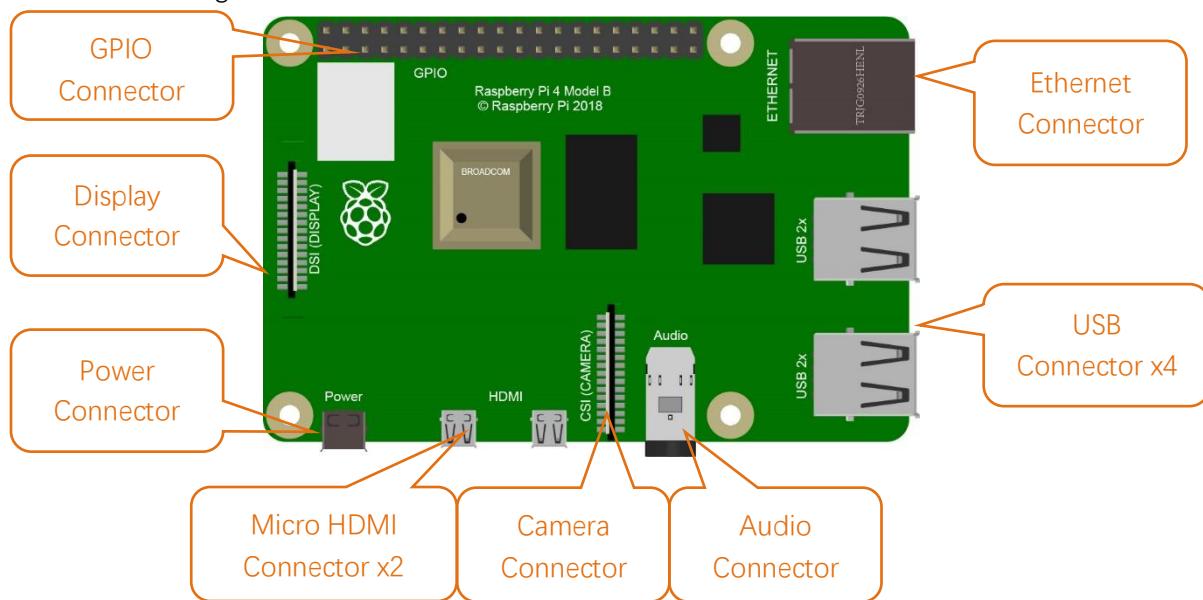
Actual image of Raspberry Pi Zero:



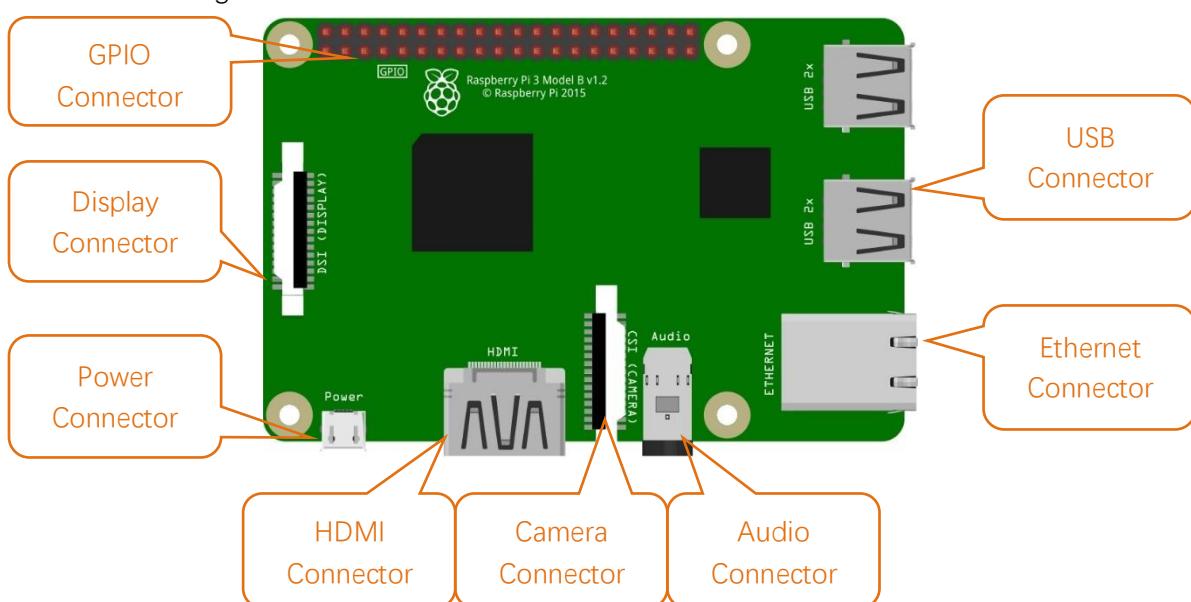
CAD image of Raspberry Pi Zero:



Hardware interface diagram of RPi 4B:



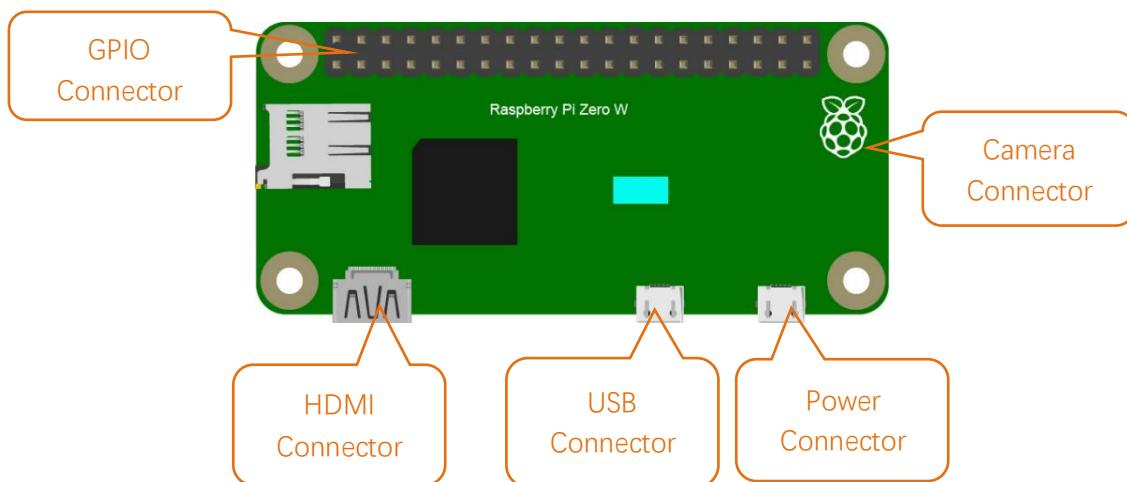
Hardware interface diagram of RPi 3B+/3B/2B/1B+:



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



Hardware interface diagram of RPi Zero/Zero W:



Installing an Operating System

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

Component List

Required Components

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



Power requirements of various versions of Raspberry Pi are shown in following table:

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

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

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

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

Optional Components

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

Required Accessories for Monitor

If you choose to use an independent monitor, mouse and keyboard, you also need the following accessories:

1. A display with a HDMI interface
2. A Mouse and a Keyboard with an USB interface

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

1. A Mini-HDMI to HDMI Adapter and Cable.
2. A Micro-USB to USB-A Adapter and Cable (Micro USB OTG Cable).
3. A USB HUB.
4. USB to Ethernet Interface or USB Wi-Fi receiver.

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

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

Required Accessories for Remote Desktop

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

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

Raspberry Pi OS

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

With Screen - Use Raspberry Pi - under Windows PC: <https://youtu.be/HEywFsFrj3I>

Automatically Method

You can follow the official method to install the system for raspberry pi via visiting link below:

<https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2>

In this way, the system will be downloaded **automatically** via the application.

Manually Method

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

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

Manually install an operating system image

Browse a range of operating systems provided by Raspberry Pi and by other organisations, and download them to install manually.

[See all download options](#)



Operating system images

Many operating systems are available for Raspberry Pi, including Raspberry Pi OS, our official supported operating system, and operating systems from other organisations.

[Raspberry Pi Imager](#) is the quick and easy way to install an operating system to a microSD card ready to use with your Raspberry Pi. Alternatively, choose from the operating systems below, available to download and install manually.

Download:
[Raspberry Pi OS \(32-bit\)](#)
[Raspberry Pi Desktop](#)
[Third-Party operating systems](#)

Raspberry Pi OS

Compatible with:

[All Raspberry Pi models](#)



Raspberry Pi OS with desktop and recommended software

Release date: January 11th 2021
 Kernel version: 5.4
 Size: 2.863MB
[Show SHA256 file integrity hash](#)
[Release notes](#)

[Download](#)

[Download torrent](#)



And then the zip file is downloaded.

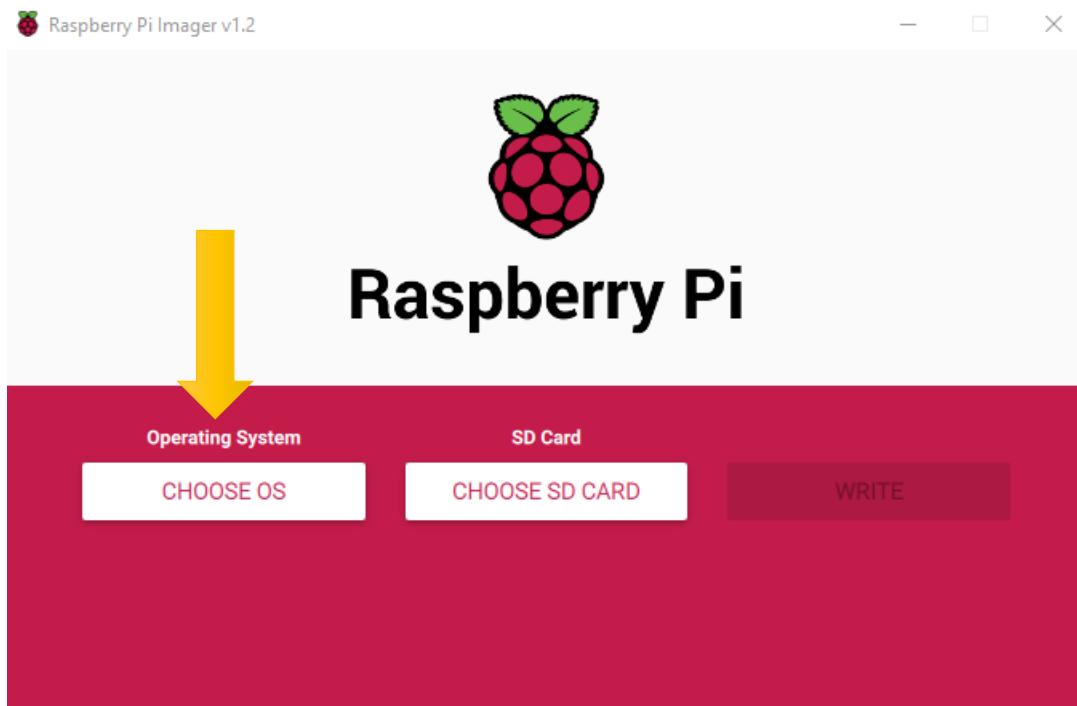


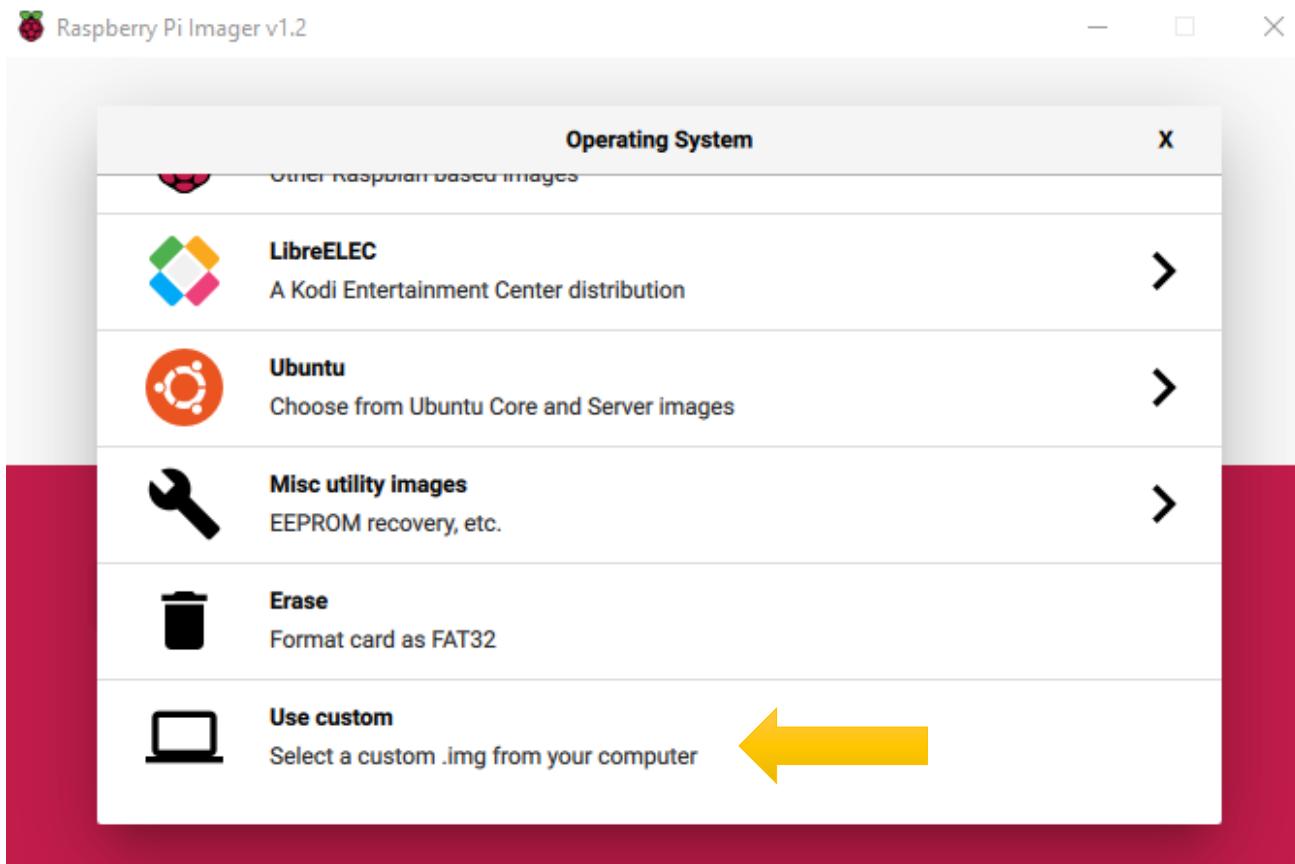
Write System to Micro SD Card

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

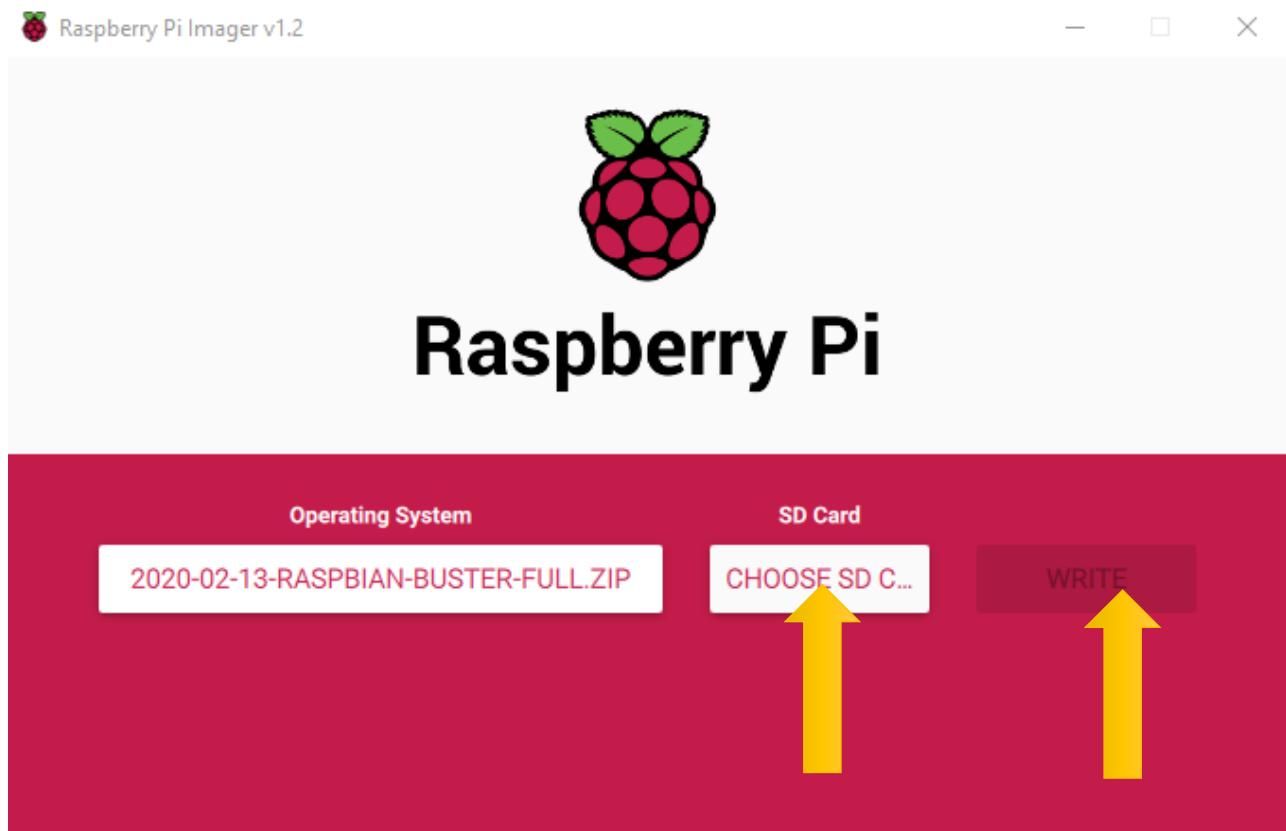


Then open imager toll. Choose system that you just downloaded in Use custom.





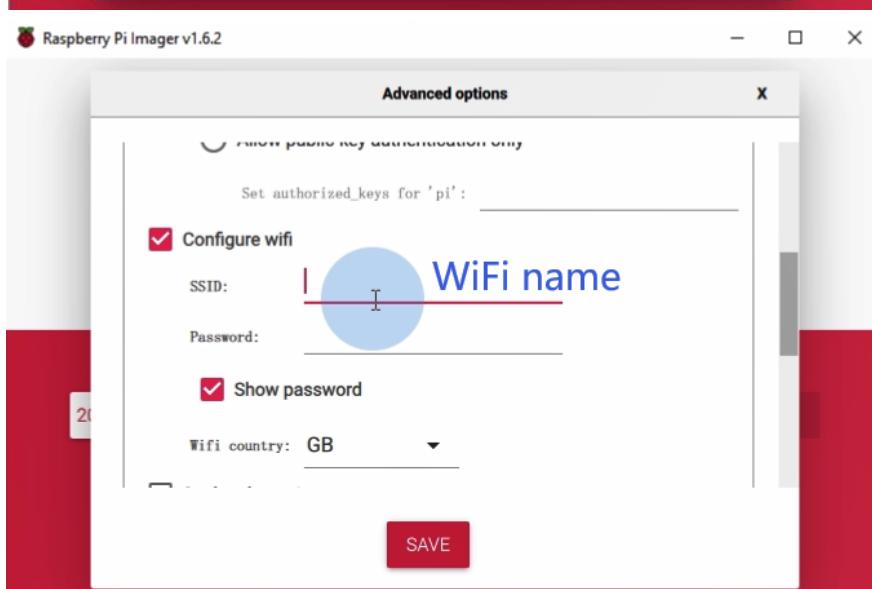
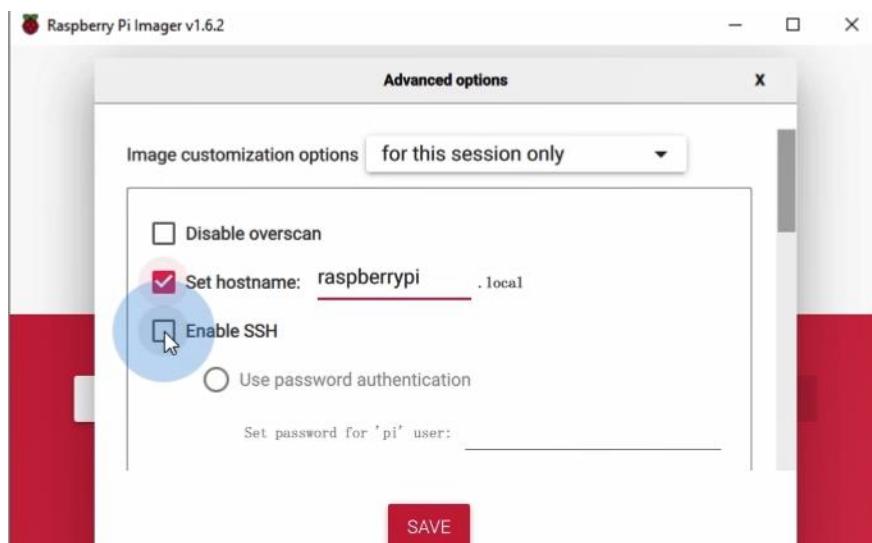
Choose the SD card. Then click "WRITE".

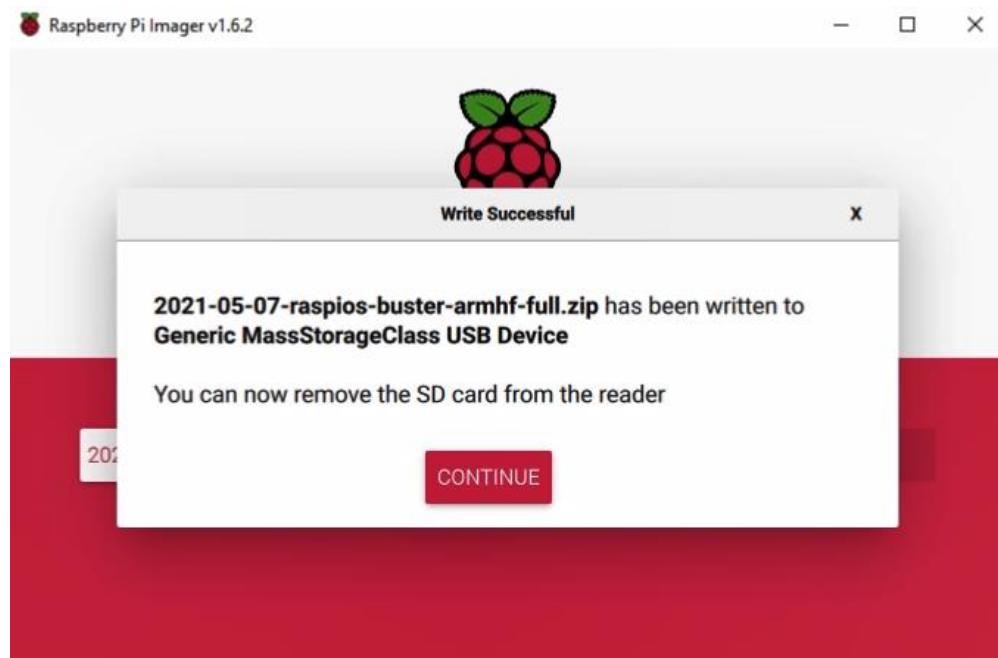


Enable ssh and configure WiFi



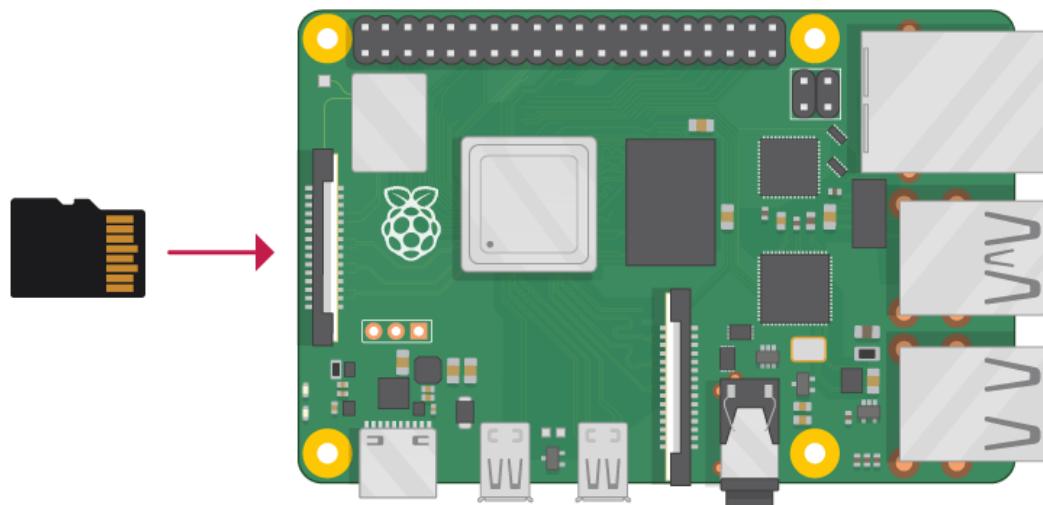
Press **Ctrl+Shift+x** to configure RPi.





Insert SD card

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

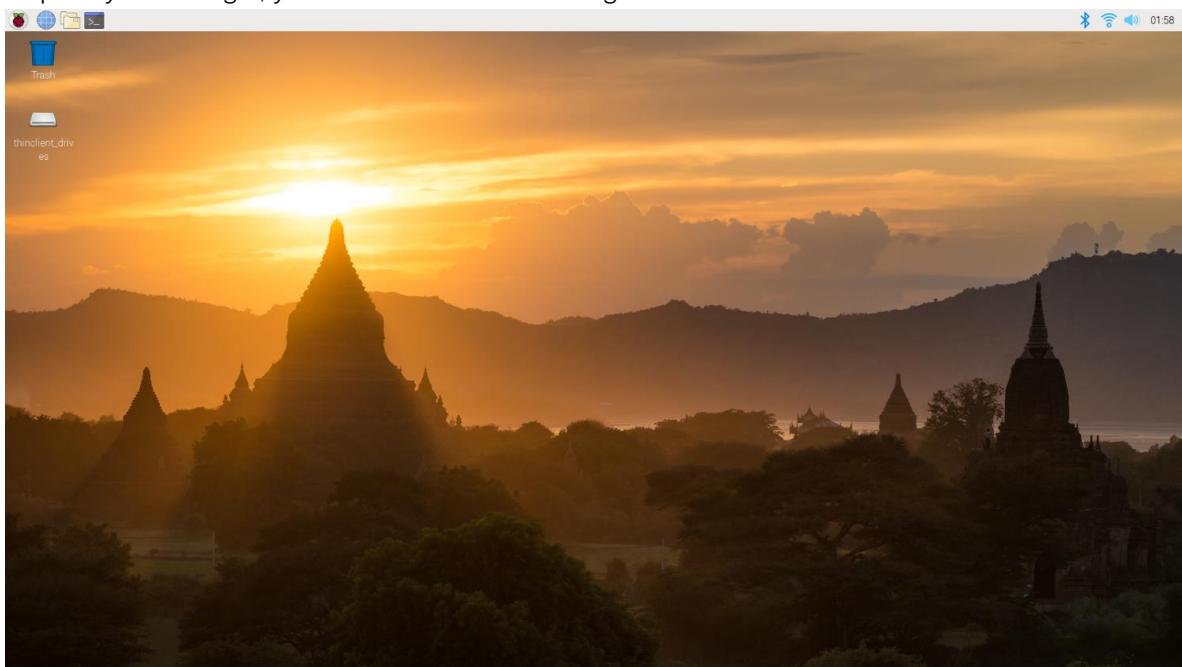


Getting Started with Raspberry Pi

Monitor desktop

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

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



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

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



Remote desktop & VNC

If you have logged in Raspberry Pi via display, you can skip to [VNC Viewer](#).

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

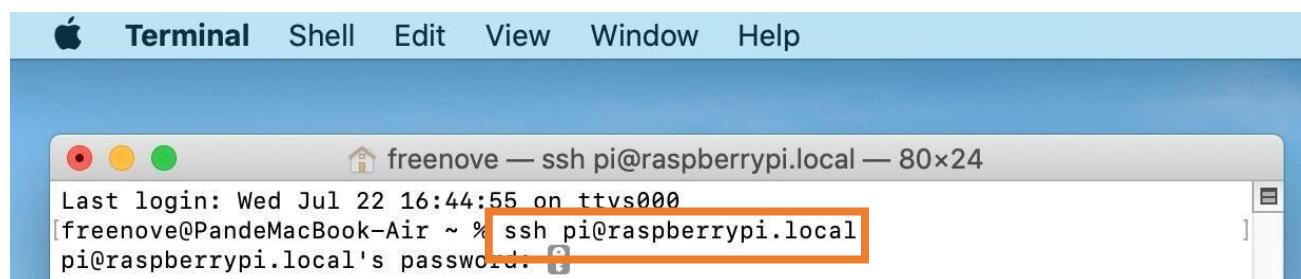
[MAC OS remote desktop](#) and [Windows OS remote desktop](#).

MAC OS Remote Desktop

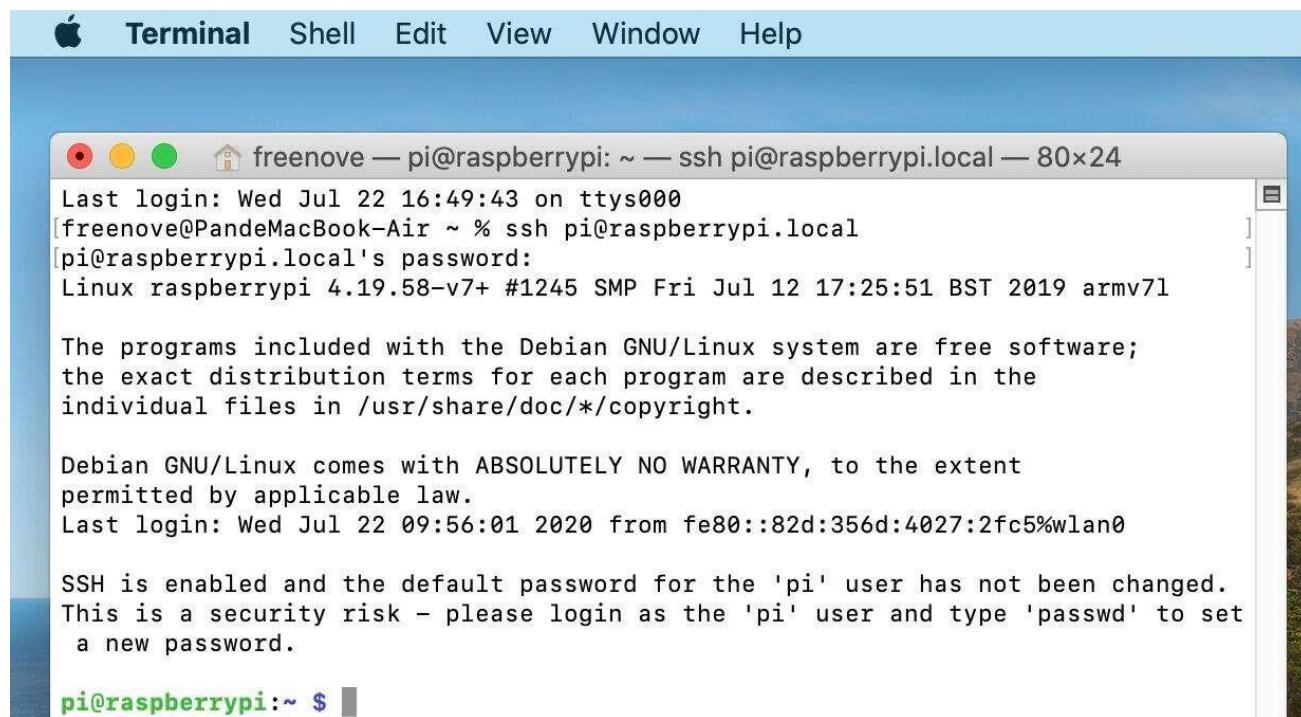
Open the terminal and type following command. **If this command doesn't work, please move to next page.**

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

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



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



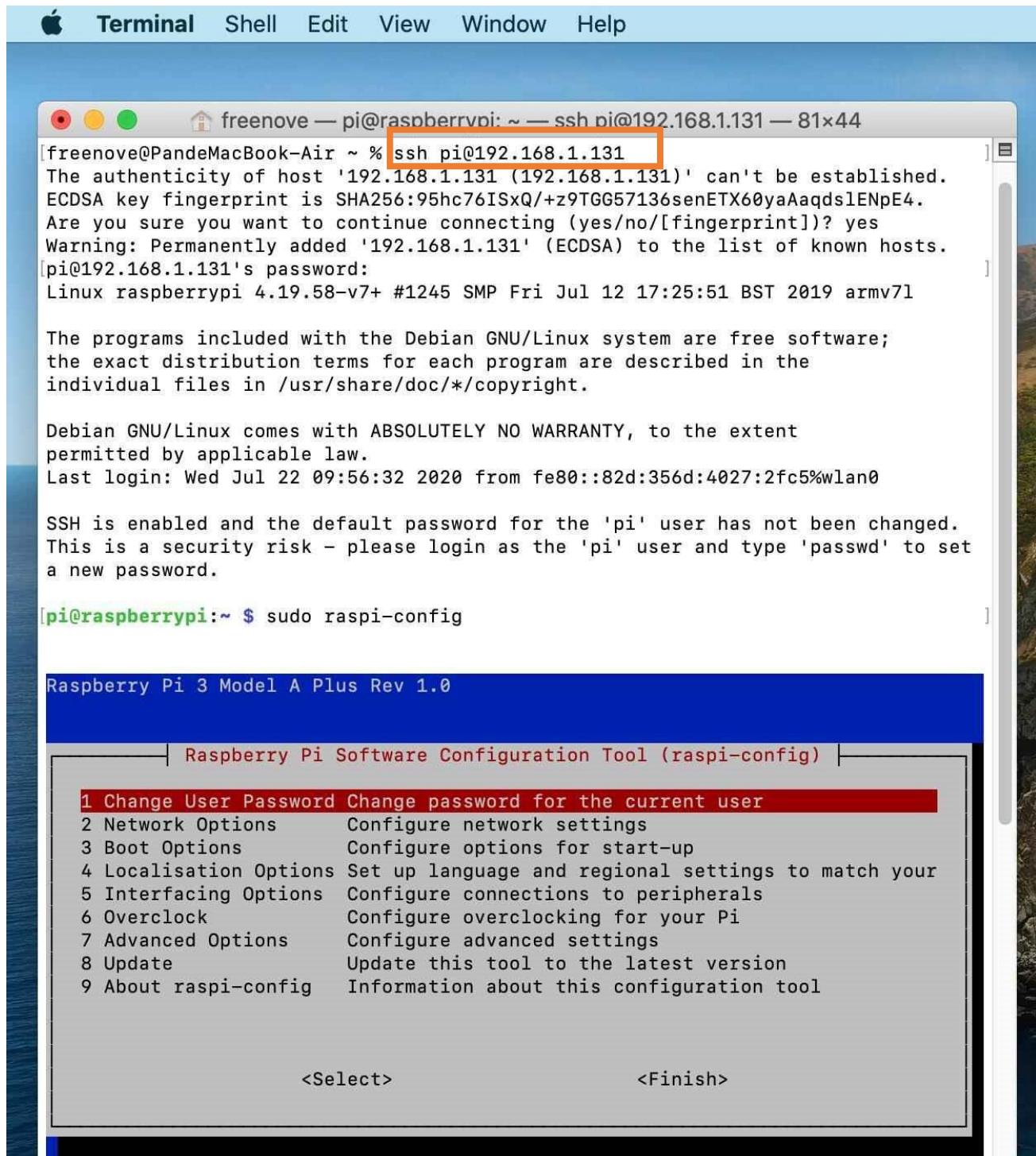
You can also use the IP address to log in Pi.

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

Open the terminal and type following command.

```
ssh pi@192.168.1.131
```

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



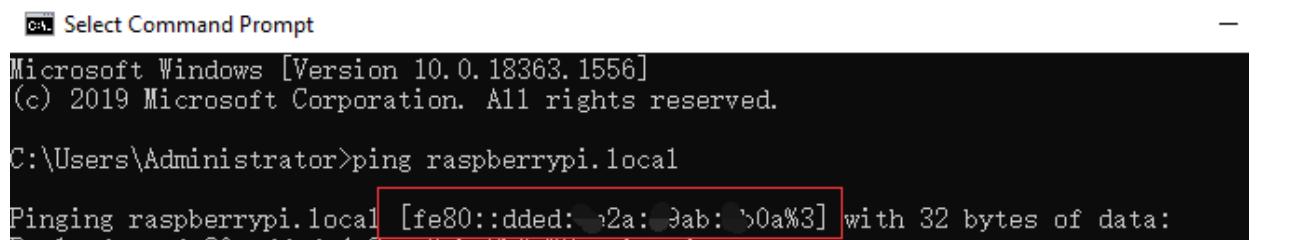
Then you can skip to [VNC Viewer](#).

Windows OS Remote Desktop

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

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

```
ping raspberrypi.local
```



```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>ping raspberrypi.local

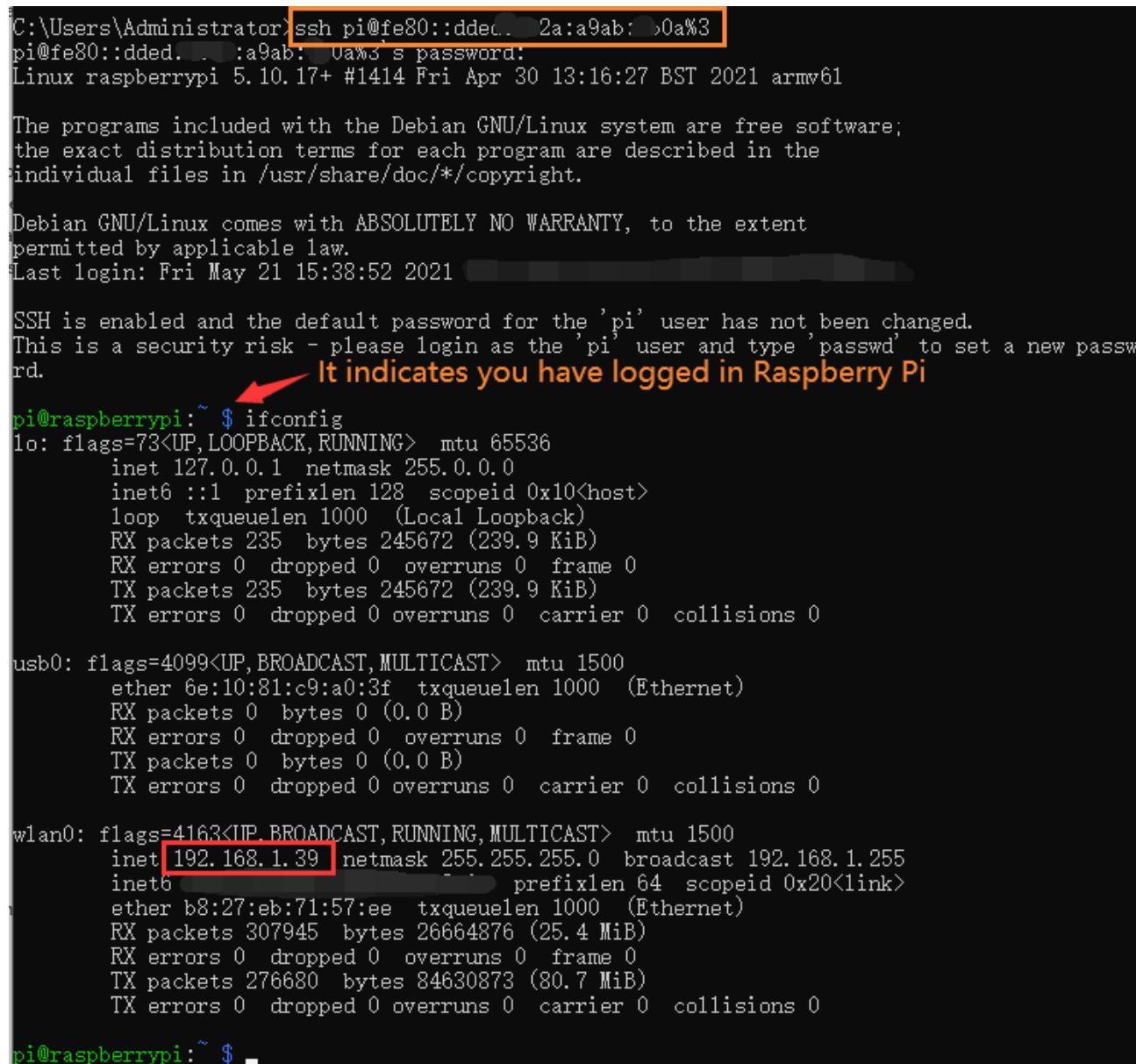
Pinging raspberrypi.local [fe80::dded:2a:a9ab:10a%3] with 32 bytes of data:
```

The one before raspberrypi.local is the IPv6 address of RaspberryPi

Use following command to login Raspberry Pi.

```
ssh pi@xxxxxxxxxxxxx(IPV6 address)
```

Enter yes not y if needed.



```
C:\Users\Administrator>ssh pi@fe80::dded:2a:a9ab:10a%3
pi@fe80::dded:2a:a9ab:10a%3's password:
Linux raspberrypi 5.10.17+ #1414 Fri Apr 30 13:16:27 BST 2021 armv6l

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Fri May 21 15:38:52 2021

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.
```

It indicates you have logged in Raspberry Pi

```
pi@raspberrypi: ~ $ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 235 bytes 245672 (239.9 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 235 bytes 245672 (239.9 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

usb0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether 6e:10:81:c9:a0:3f txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.39 netmask 255.255.255.0 broadcast 192.168.1.255
        inetc
            ether b8:27:eb:71:57:ee txqueuelen 1000 (Ethernet)
            RX packets 307945 bytes 26664876 (25.4 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 276680 bytes 84630873 (80.7 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

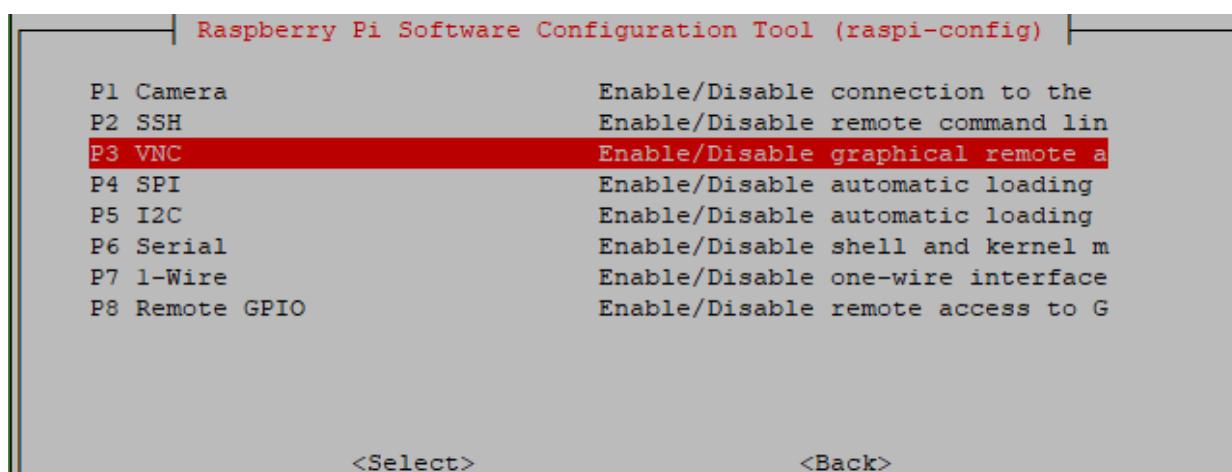
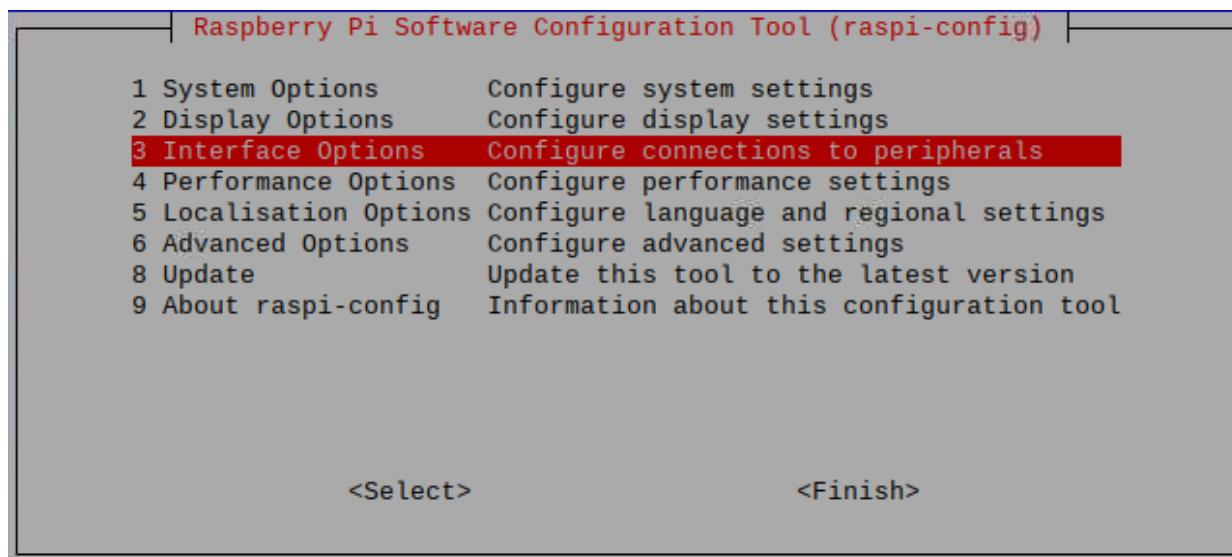
pi@raspberrypi: ~ $
```

VNC Viewer & VNC

Enable VNC

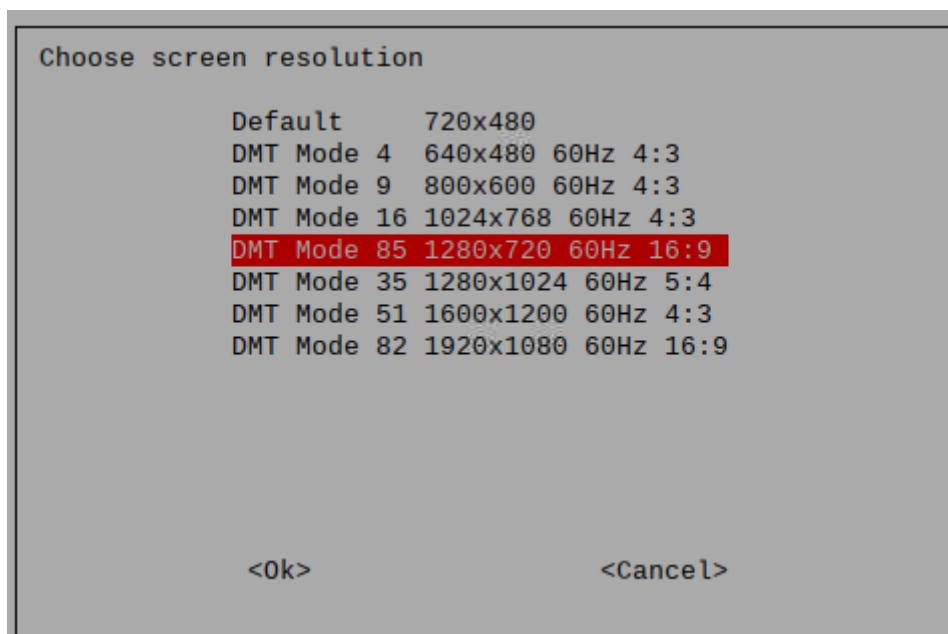
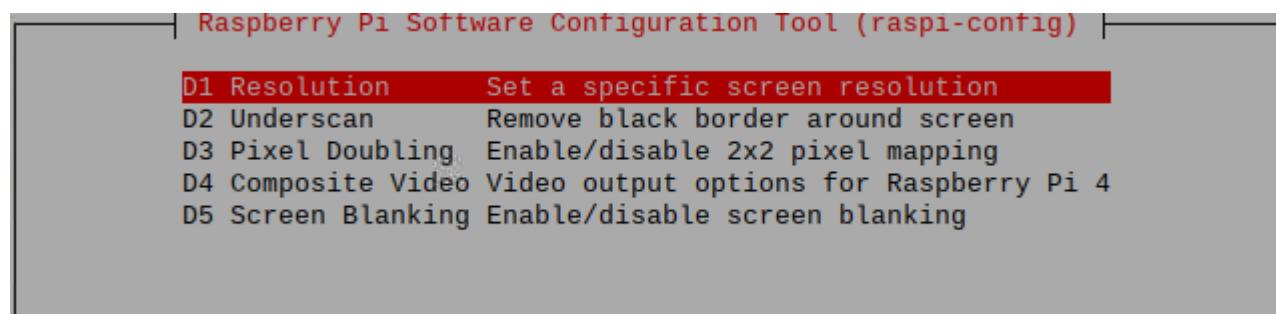
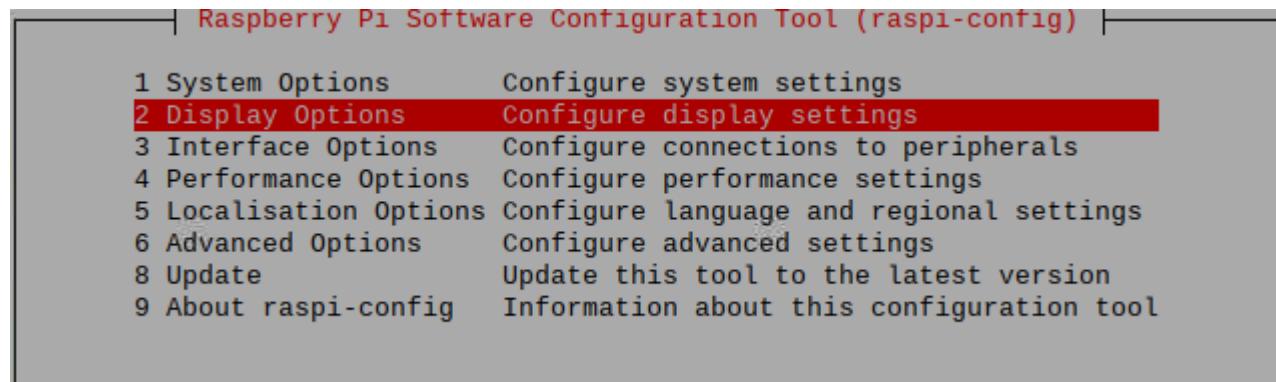
Type the following command. And select Interface Options → P3 VNC → Enter → Yes → OK. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

```
sudo raspi-config
```



Set Resolution

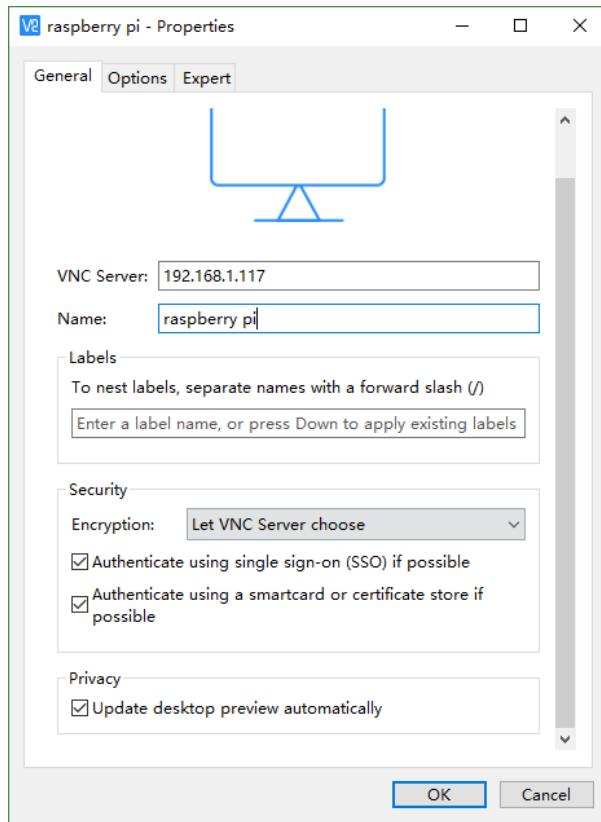
You can also set other resolutions. If you don't know what to set, you can set it as 1280x720 first.



Then download and install VNC Viewer according to your computer system by click following link:

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

After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.

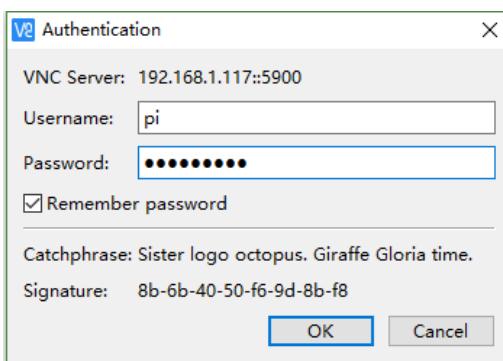


Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

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



and the following dialog box pops up.

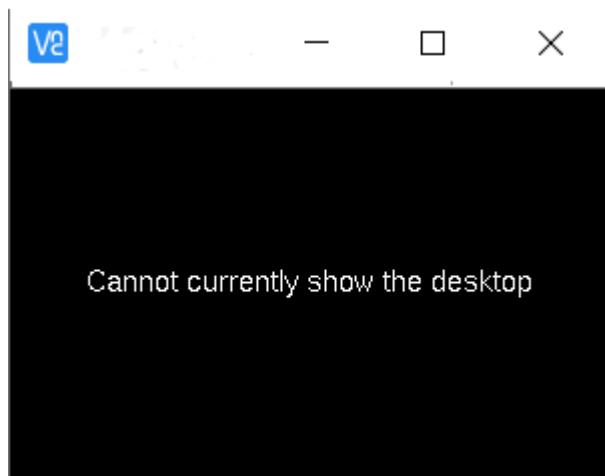


Enter username: **pi** and Password: **raspberry**. And click OK.

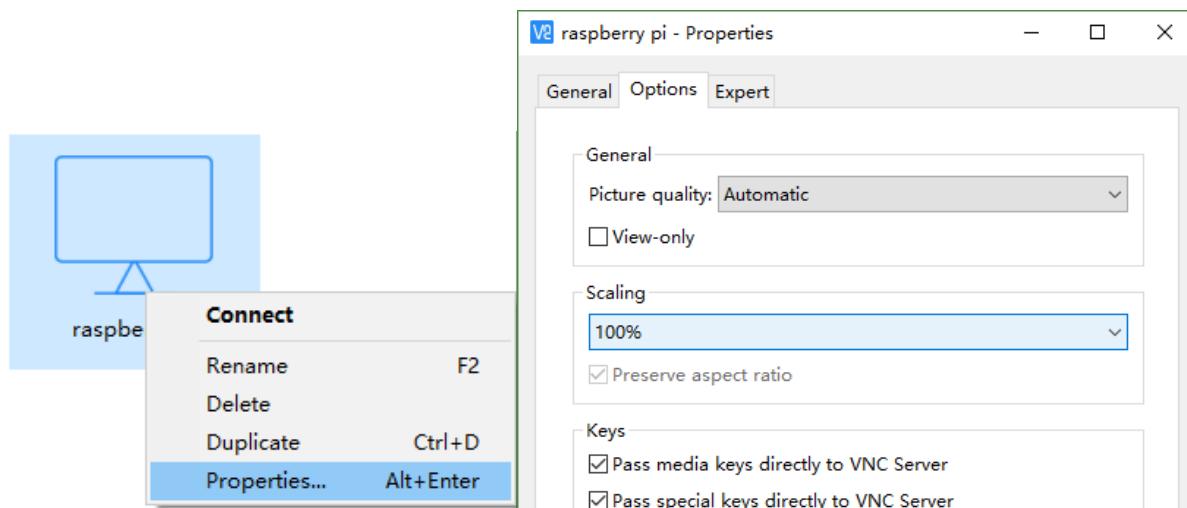


Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

If there is black window, please [set another resolution](#).



In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. And select Properties->Options label->Scaling. Then set proper scaling.





Here, you have logged in to Raspberry Pi successfully by using VNC Viewer and operated proper setting.

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



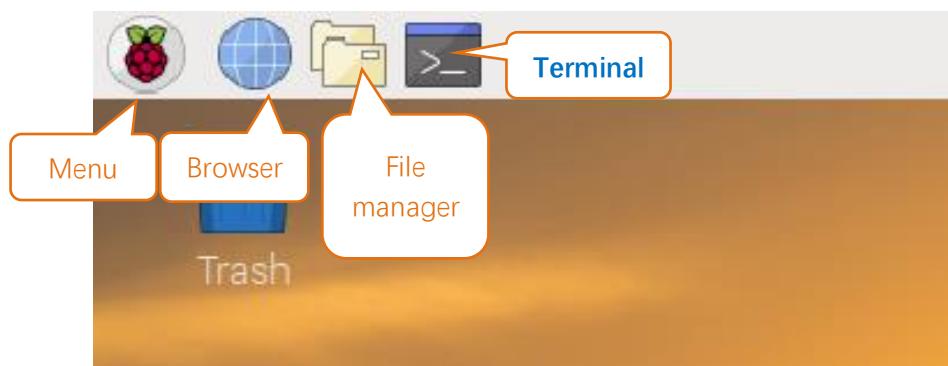
Chapter 0 Preparation

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

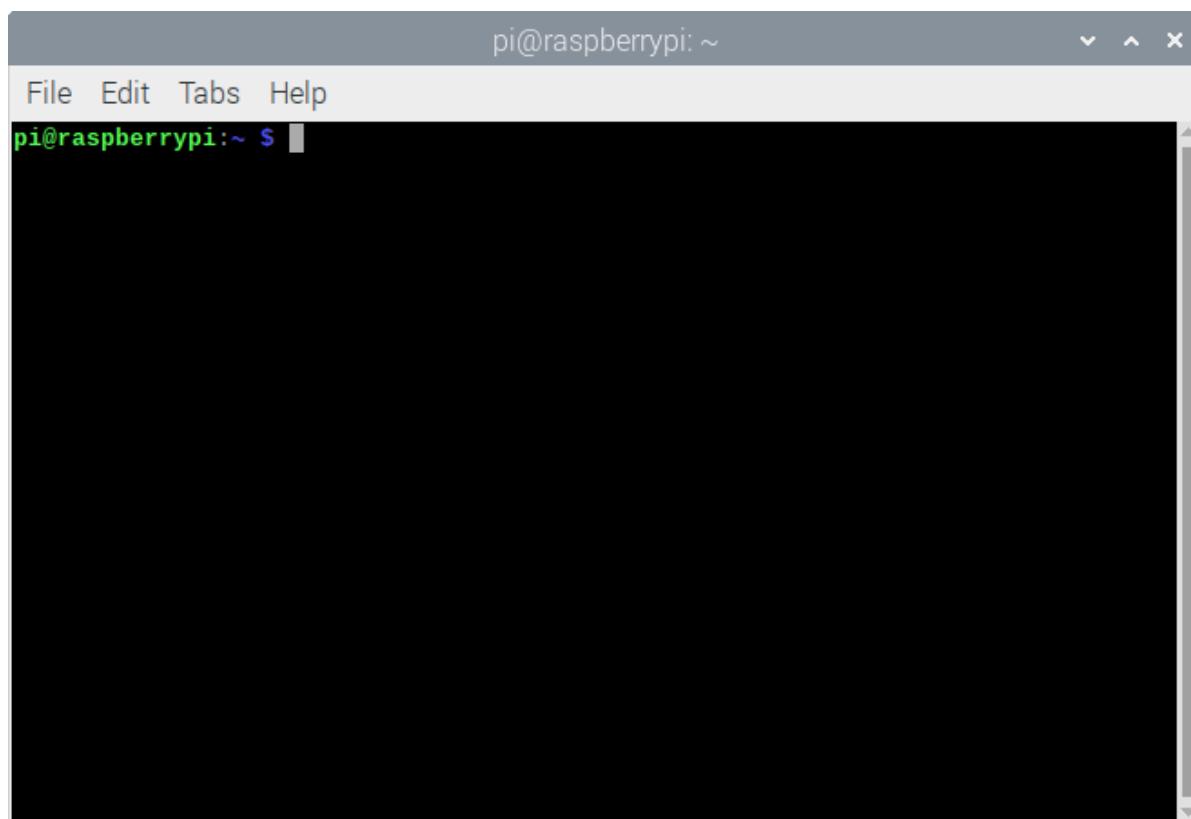
Linux Command

Raspberry Pi OS is based on the Linux Operation System. Now we will introduce you to some frequently used Linux commands and rules.

First, open the Terminal. All commands are executed in Terminal.



When you click the Terminal icon, following interface appears.



Note: The Linux is case sensitive.

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



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

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

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

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

“cd” is used to change directory. “/” represents the root directory.

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

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

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

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

There are many commands, which will come later. For more details about commands. You can refer to:

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

Shortcut Key

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

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

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

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

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

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

Install WiringPi

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

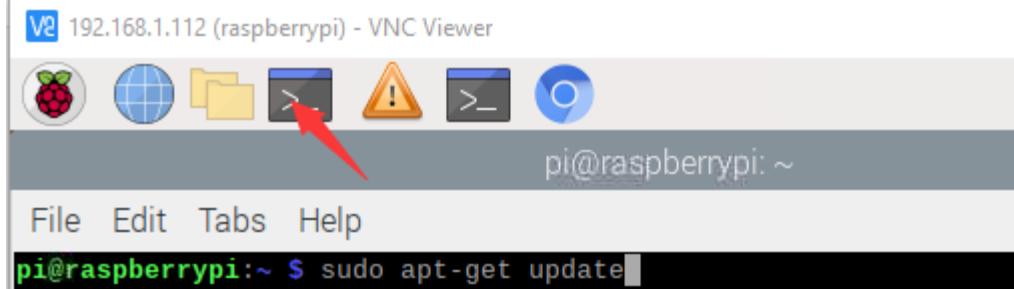
WiringPi Installation Steps

To install the WiringPi library, please open the Terminal and then follow the steps and commands below.

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

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

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



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

```
pi@raspberrypi:~ $ git clone https://github.com/WiringPi/WiringPi
pi@raspberrypi:~/WiringPi $ ./build
wiringPi Build script
=====
```

All Done.

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

Run the gpio command to check the installation:

```
gpio -v
```

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

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

Raspberry Pi Details:
```

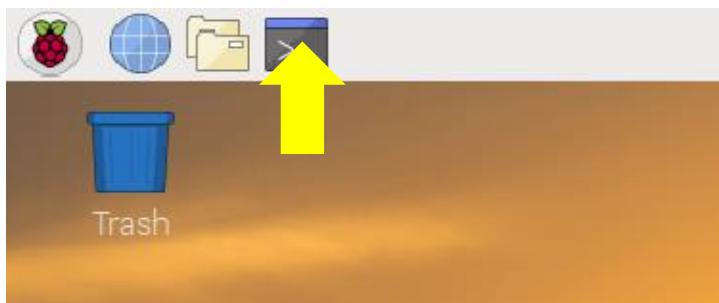
Obtain the Project Code

After the above installation is completed, you can visit our official website (<http://www.freenove.com>) or our GitHub resources at (<https://github.com/freenove>) to download the latest available project code. We provide both **C** language and **Python** language code for each project to allow ease of use for those who are skilled in either language.

This is the method for obtaining the code:

In the pi directory of the RPi terminal, enter the following command.

```
cd  
git clone --depth 1 https://github.com/freenove/Freenove_RFID_Starter_Kit_for_Raspberry_Pi  
(There is no need for a password. If you get some errors, please check your commands.)
```

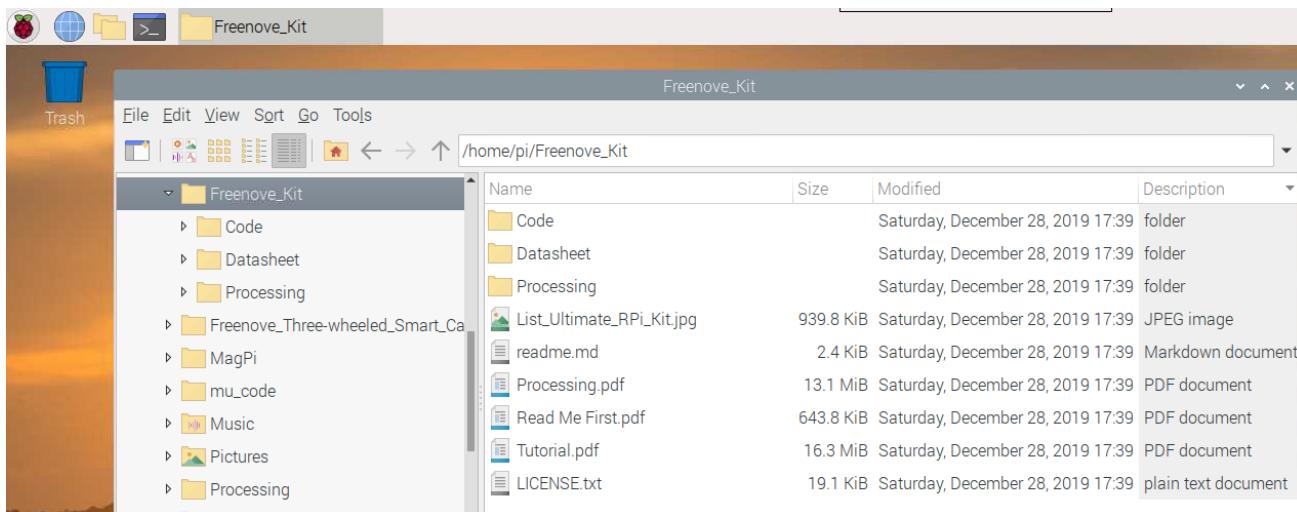


After the download is completed, a new folder "Freenove_RFID_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

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

```
mv Freenove_RFID_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

"Freenove_Kit" is now the new and much shorter folder name.



If you have no experience with Python, we suggest that you refer to this website for basic information and knowledge.

<https://python.swaroopch.com/basics.html>

Python2 & Python3

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

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

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

```
pi@raspberrypi:~ $ python2
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
[2]+ Stopped                  python2
pi@raspberrypi:~ $ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

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

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

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

1. Enter directory /usr/bin

```
cd /usr/bin
```

2. Delete the old python link.

```
sudo rm python
```

3. Create new python links to python3.

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

4. Execute python to check whether the link succeeds.

```
python
```

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

If you want to use Python2, repeat the steps above and just change the third command to the following:

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

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

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

Chapter 1 LED

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

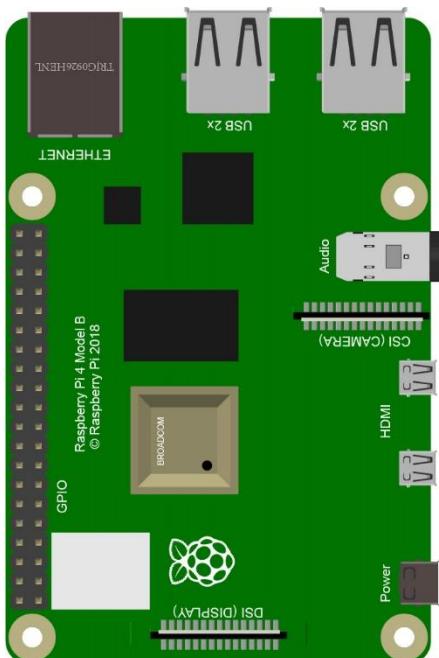
Project 1.1 Blink

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

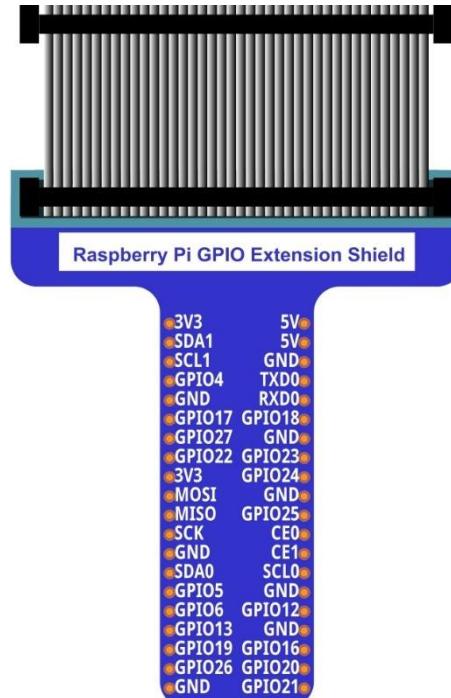
Component List

Raspberry Pi

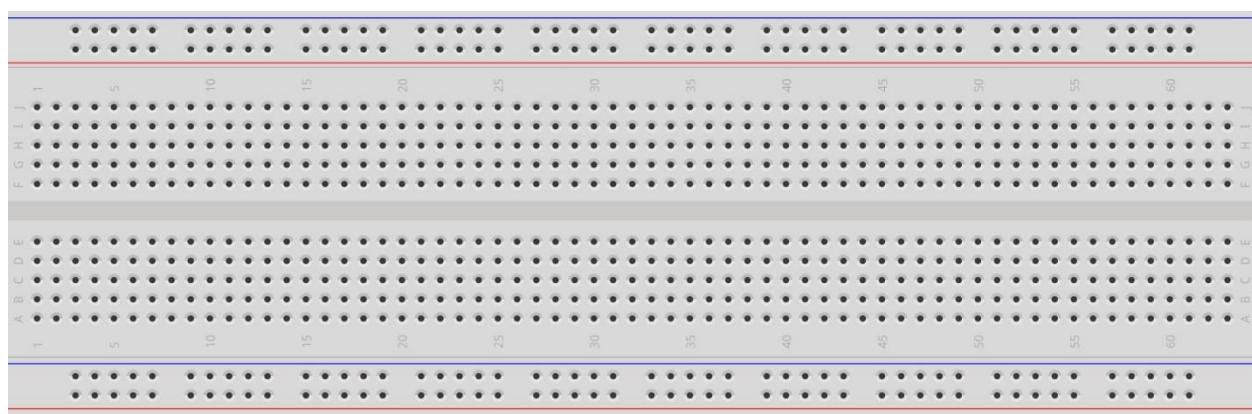
(Recommended: Raspberry Pi 4B / 3B+ / 3B
Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)



GPIO Extension Board & Ribbon Cable



Breadboard x1



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

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

GPIO

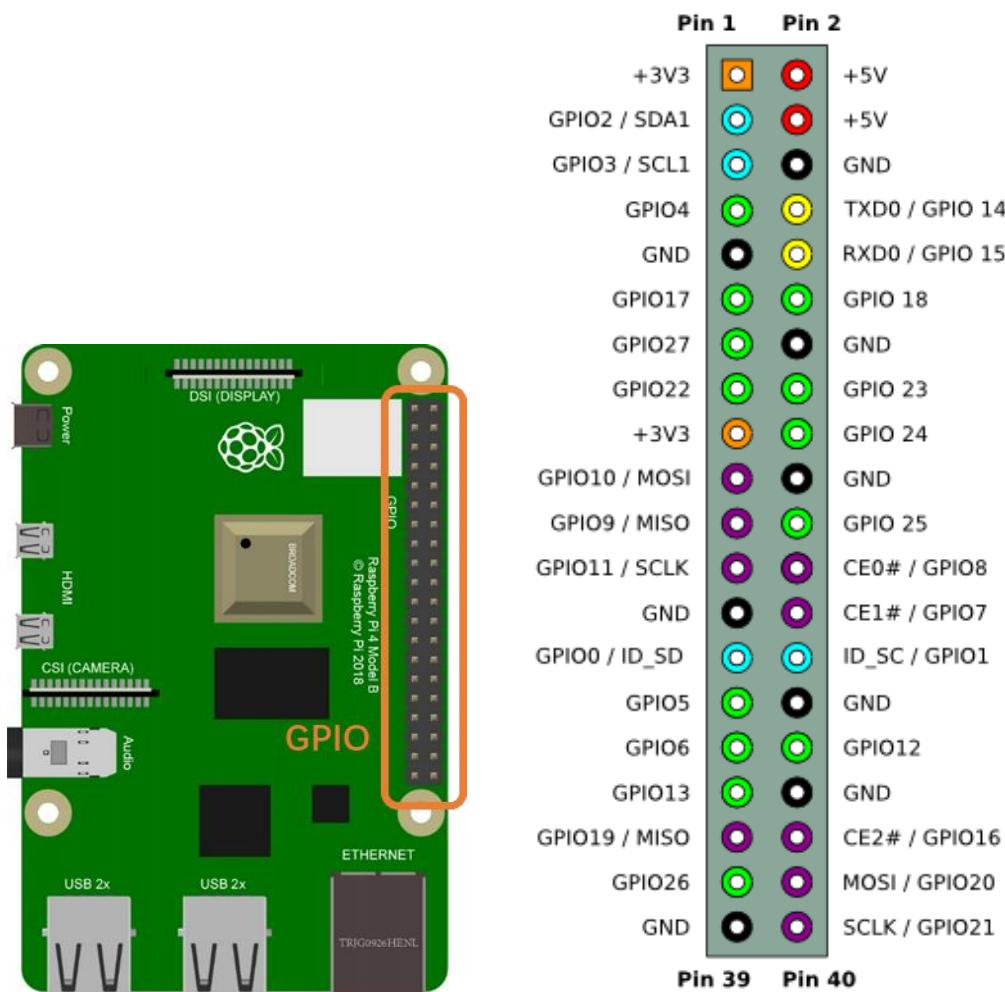
GPIO: General Purpose Input/Output. Here we will introduce the specific function of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions.

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

BCM GPIO Numbering

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

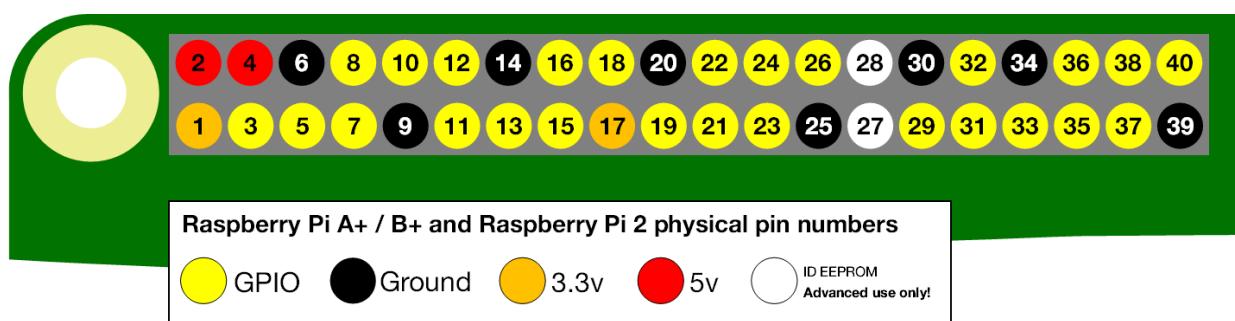
Each pin's functional assignment is defined in the image below:



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

PHYSICAL Numbering

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



WiringPi GPIO Numbering

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

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

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

For Pi B

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

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

```
gpio readall
```

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

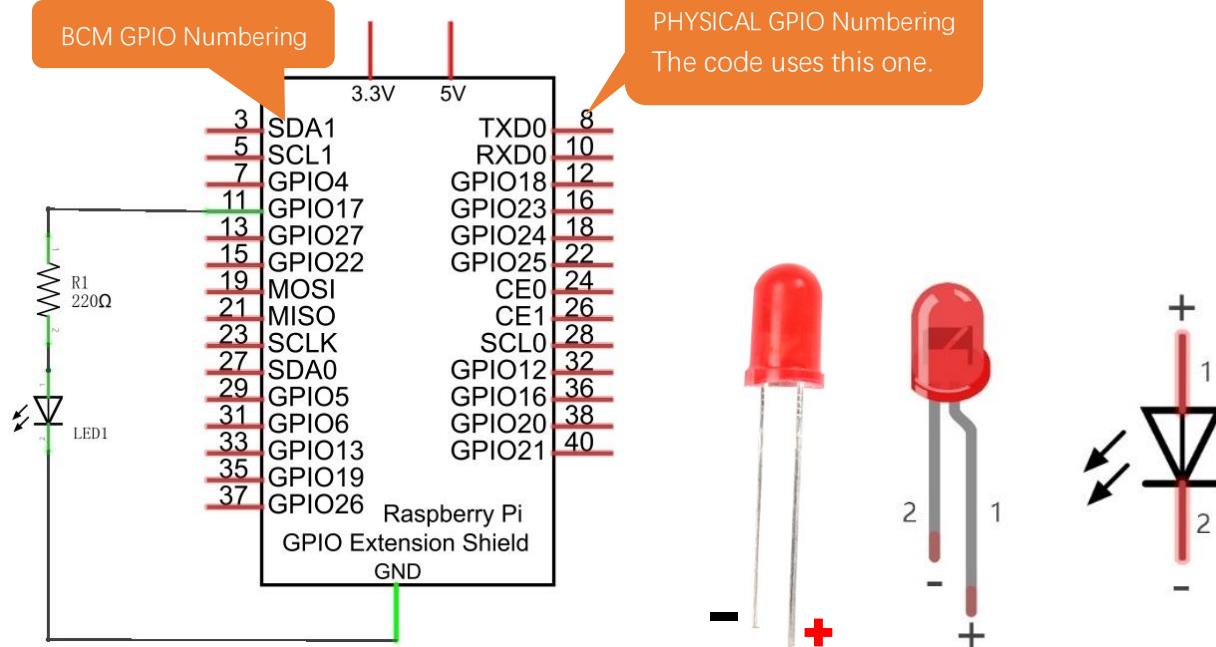
Circuit

First, disconnect your RPi from the GPIO Extension Shield. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield.

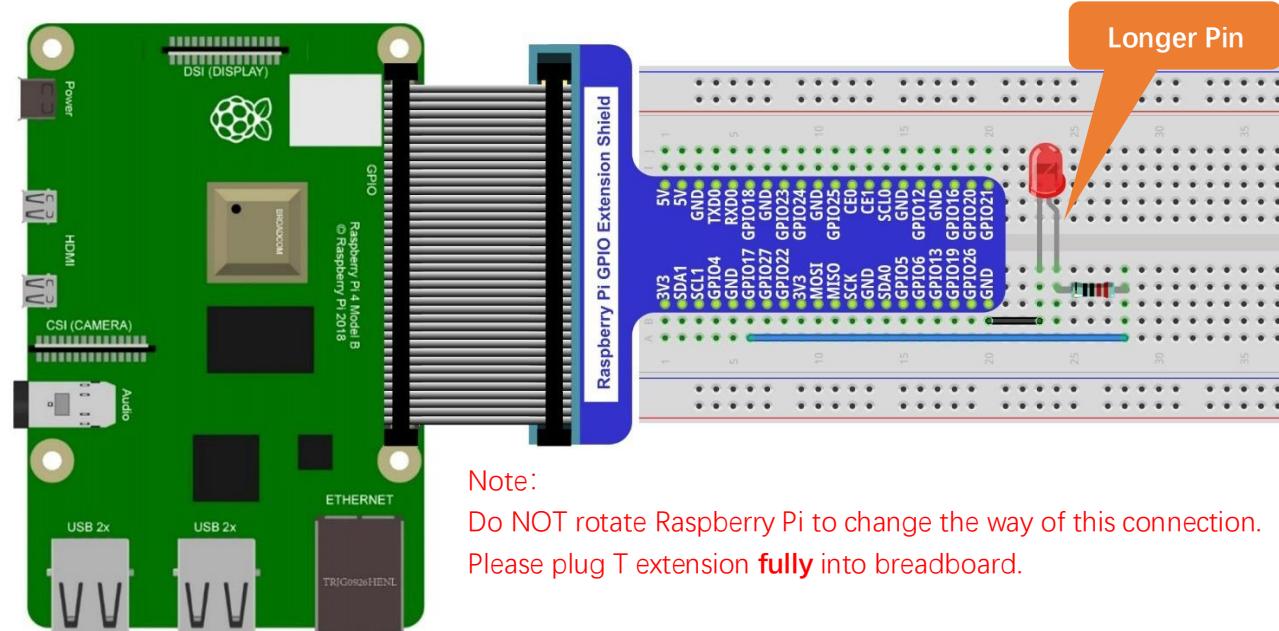
CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!

WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your RPi!

Schematic diagram

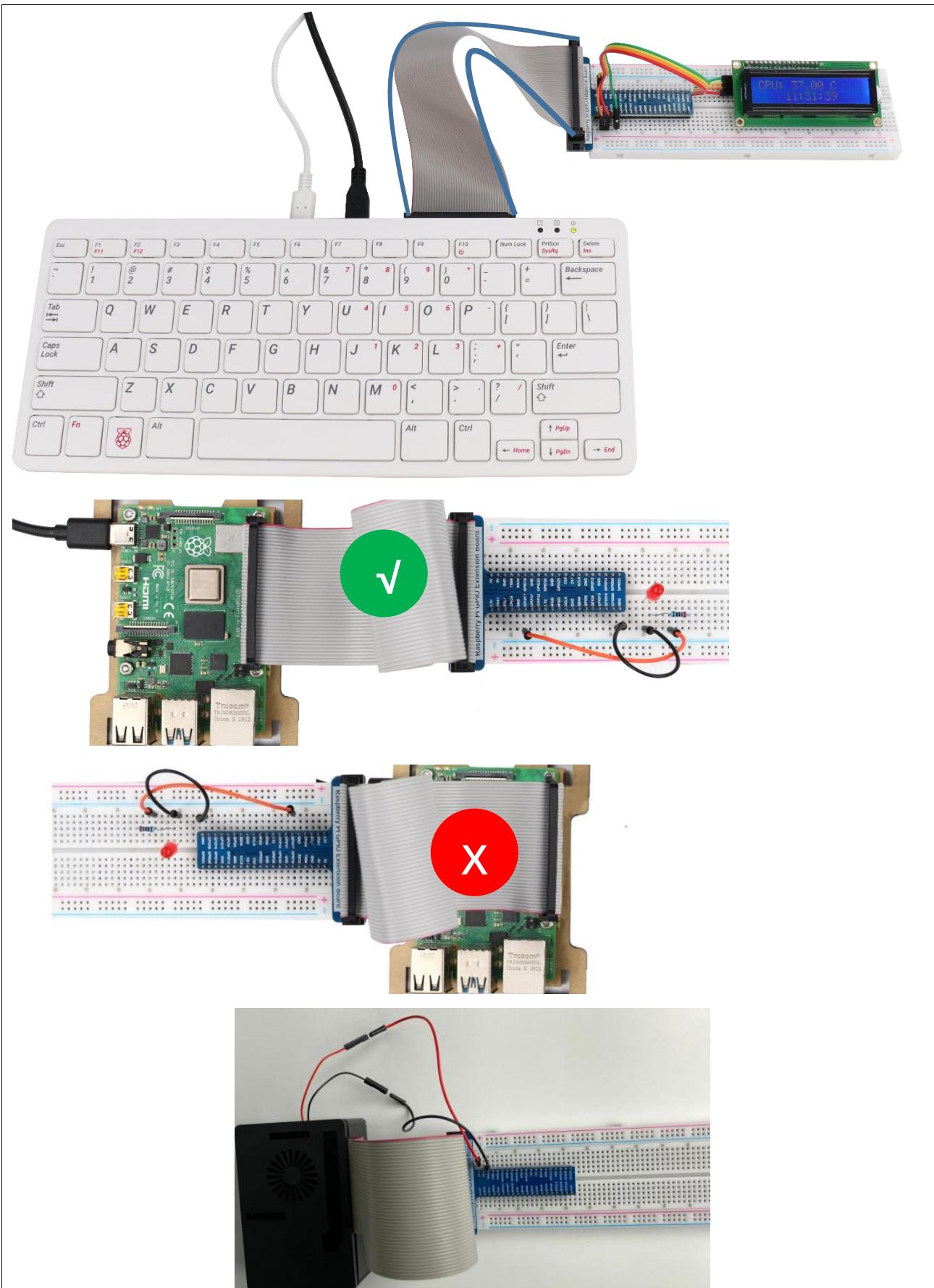


Hardware connection. **If you need any support, please contact us via: support@freenove.com**



Youtube video <https://youtu.be/hGQtnxsr1L4>

The connection of **Raspberry Pi 400** and **T extension board** is as below. **Don't reverse the ribbon.**



If you have a fan, you can connect it to 5V GND of breadboard via jumper wires.

How to distinguish resistors?

There are only three kind of resistors in this kit.

The one with 1 red ring is $10\text{K}\Omega$ 

The one with 2 red rings is 220Ω 

The one with 0 red ring is $1\text{K}\Omega$ 

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

Component knowledge

LED

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

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burnt out.



LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA

Volt ampere characteristics conform to diode

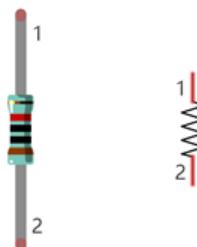
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit.

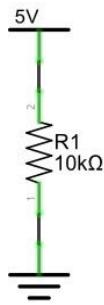
On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.

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

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

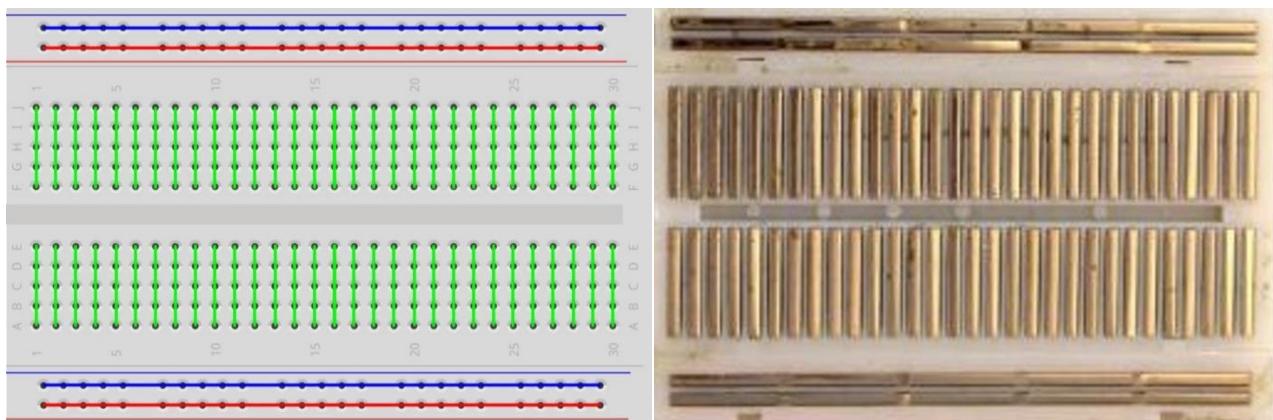


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

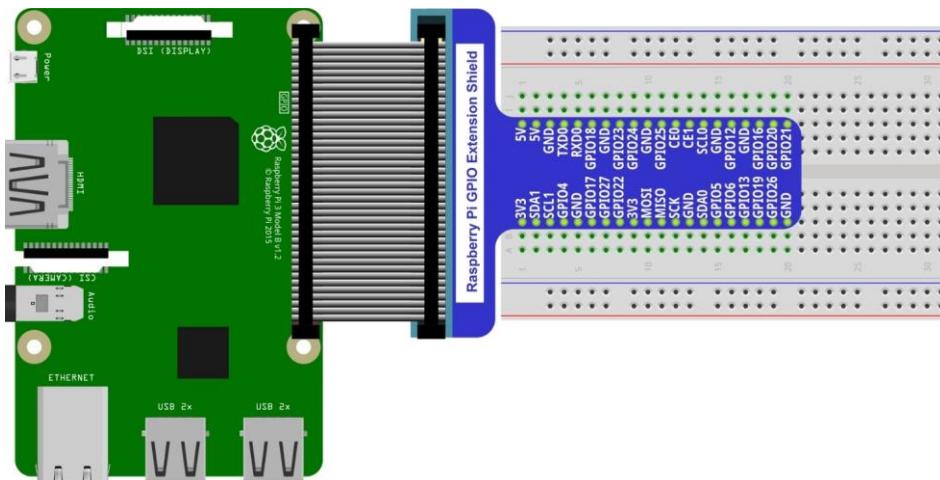
Breadboard

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



GPIO Extension Board

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



Code

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

C Code 1.1.1 Blink

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

If you want to execute it with editor, please refer to section [Code Editor](#) to configure.

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

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

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

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

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

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

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

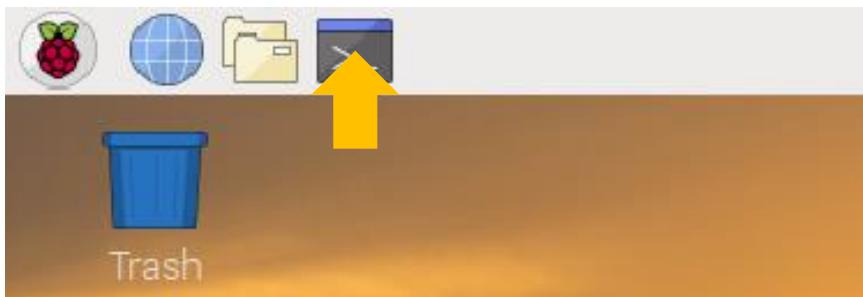
"l" of "lwiringPi" is low case of "L".

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

4. Then run the generated file "blink".

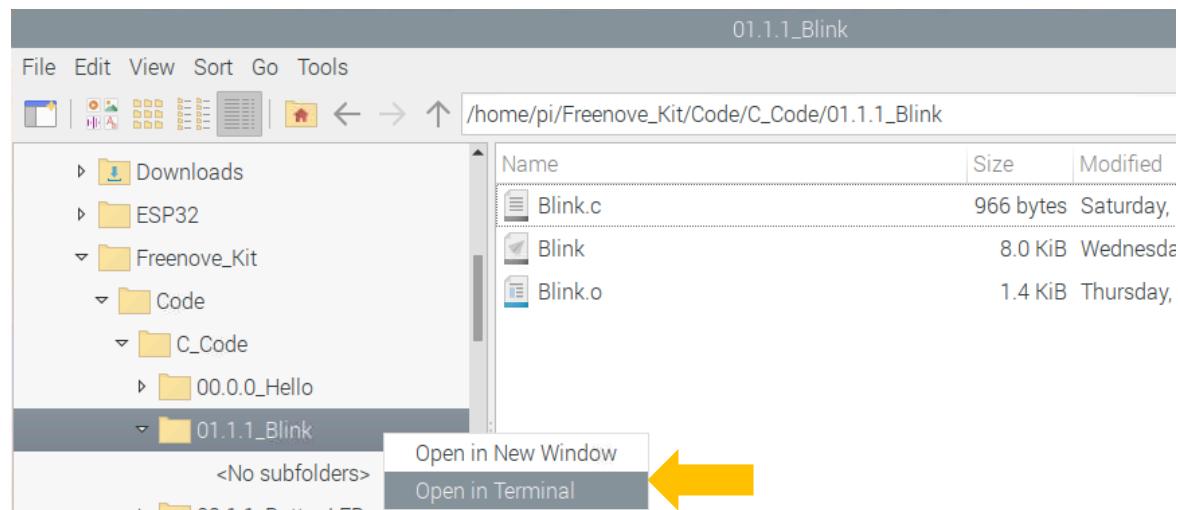
```
sudo ./Blink
```

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



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

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



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

```

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



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

```
void pinMode(int pin, int mode);
```

This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program

```
void digitalWrite (int pin, int value);
```

Writes the value HIGH or LOW (1 or 0) to the given pin, which must have been previously set as an output.

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

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

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

GPIO Numbering Relationship

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

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

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

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

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

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



Python Code 1.1.1 Blink

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

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

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

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

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

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

```
python Blink.py
```

The LED starts blinking.

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

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

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

```

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



```

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

```

About RPi.GPIO:

RPi.GPIO

This is a Python module to control the GPIO on a Raspberry Pi. It includes basic output function and input function of GPIO, and functions used to generate PWM.

GPIO.setmode(mode)

Sets the mode for pin serial number of GPIO.

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

GPIO.setup(pin, mode)

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

GPIO.output(pin, mode)

Sets pin to output mode, "pin" for the GPIO pin, "mode" for HIGH (high level) or LOW (low level).

For more functions related to RPi.GPIO, please refer to:

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

"import time" time is a module of python.

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

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

```

ledPin = 11      # define ledPin

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

```



GPIO Numbering Relationship

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

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

```
def loop():
    while True:
        GPIO.output(ledPin, GPIO.HIGH) # make ledPin output HIGH level to turn on led
        print ('led turned on >>>') # print information on terminal
        time.sleep(1) # Wait for 1 second
        GPIO.output(ledPin, GPIO.LOW) # make ledPin output LOW level to turn off led
        print ('led turned off <<<')
        time.sleep(1) # Wait for 1 second
```

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

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

Other Code Editors (Optional)

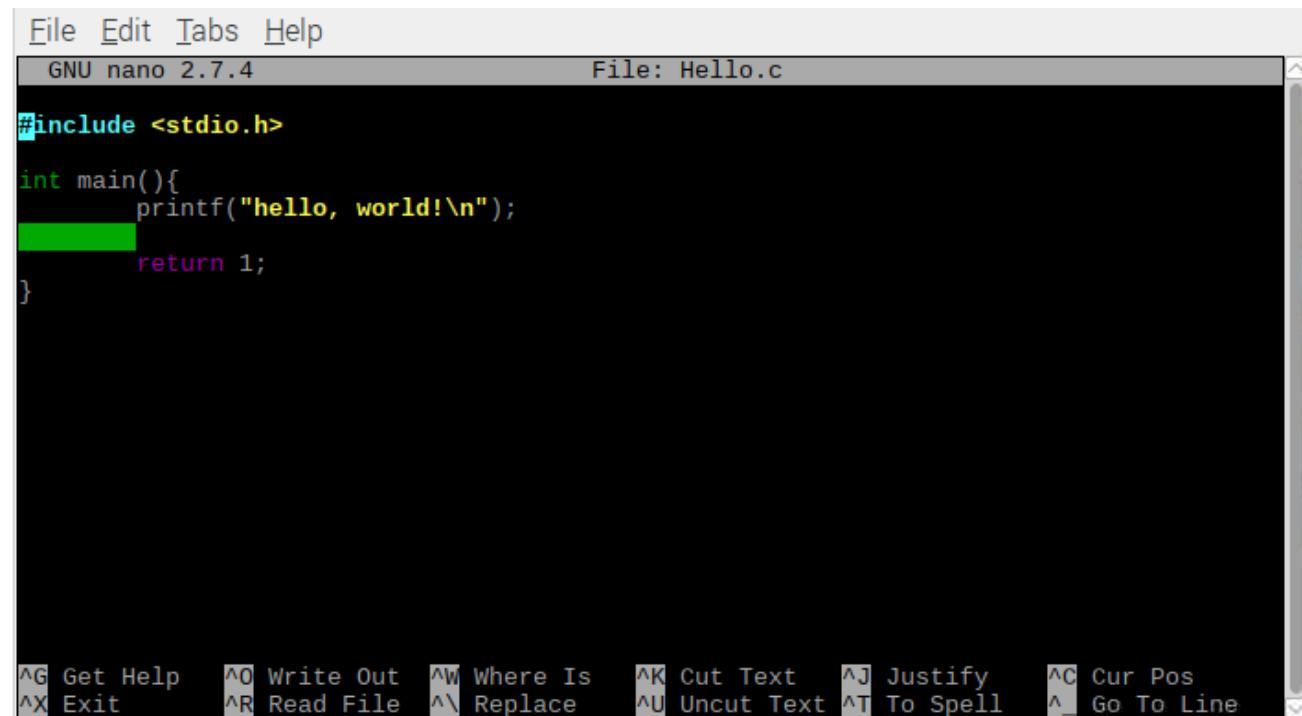
If you want to use other editor to edit and execute the code, you can learn them in this section.

nano

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

```
nano Hello.c
```

As is shown below:



The screenshot shows the nano text editor interface. The menu bar includes File, Edit, Tabs, and Help. The title bar displays "GNU nano 2.7.4" and "File: Hello.c". The main editing area contains the following C code:

```
#include <stdio.h>

int main(){
    printf("hello, world!\n");
    return 1;
}
```

The bottom of the window shows a series of keyboard shortcuts:

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

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

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

Use the following command to run the executable file "Hello".

```
sudo ./Hello
```

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

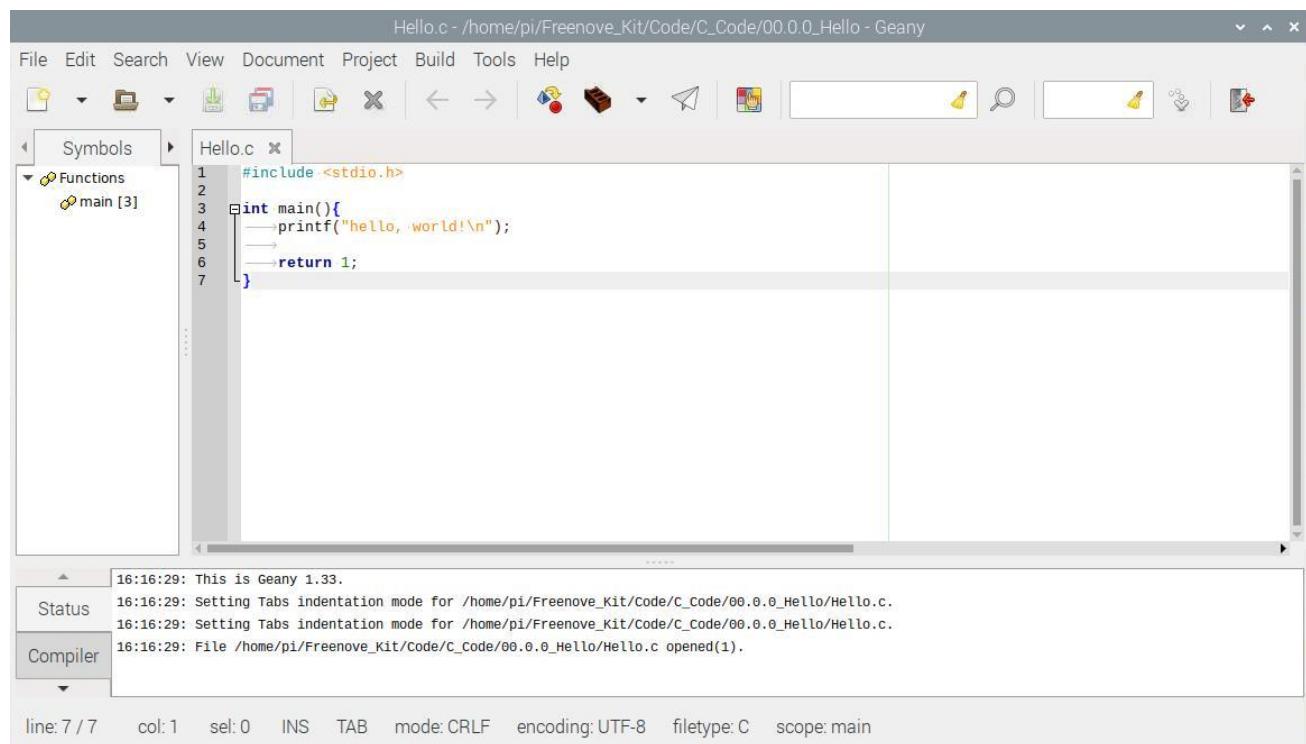
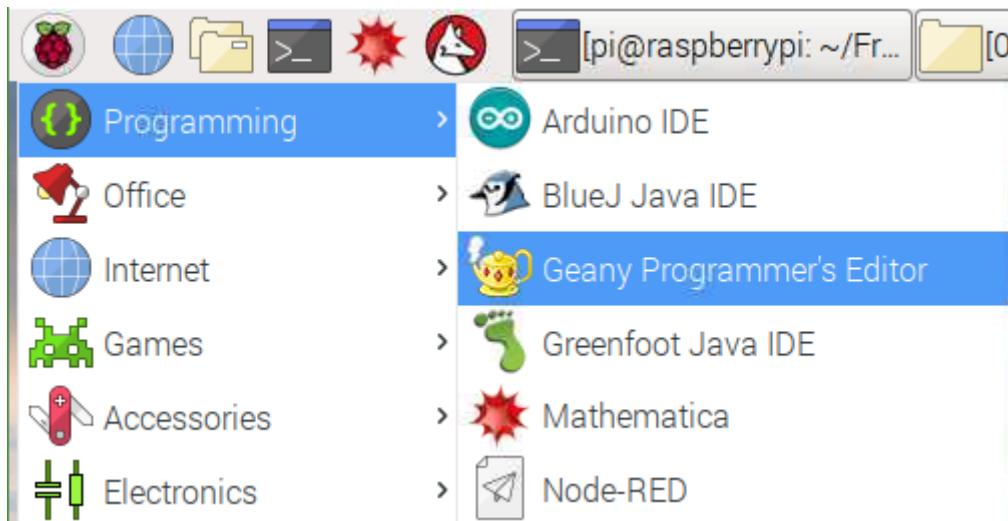
```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ gcc Hello.c -o Hello
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ sudo ./Hello
hello, world!
```

geany

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

geany Hello.c

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

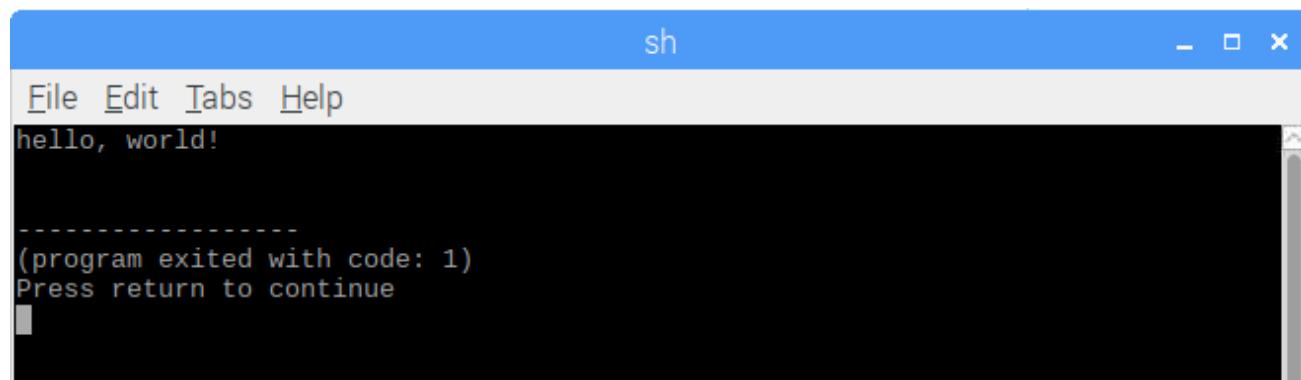


If you want to create a new code, click **File→New→File→Save as** (name.c or name.py). Then write the code.

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

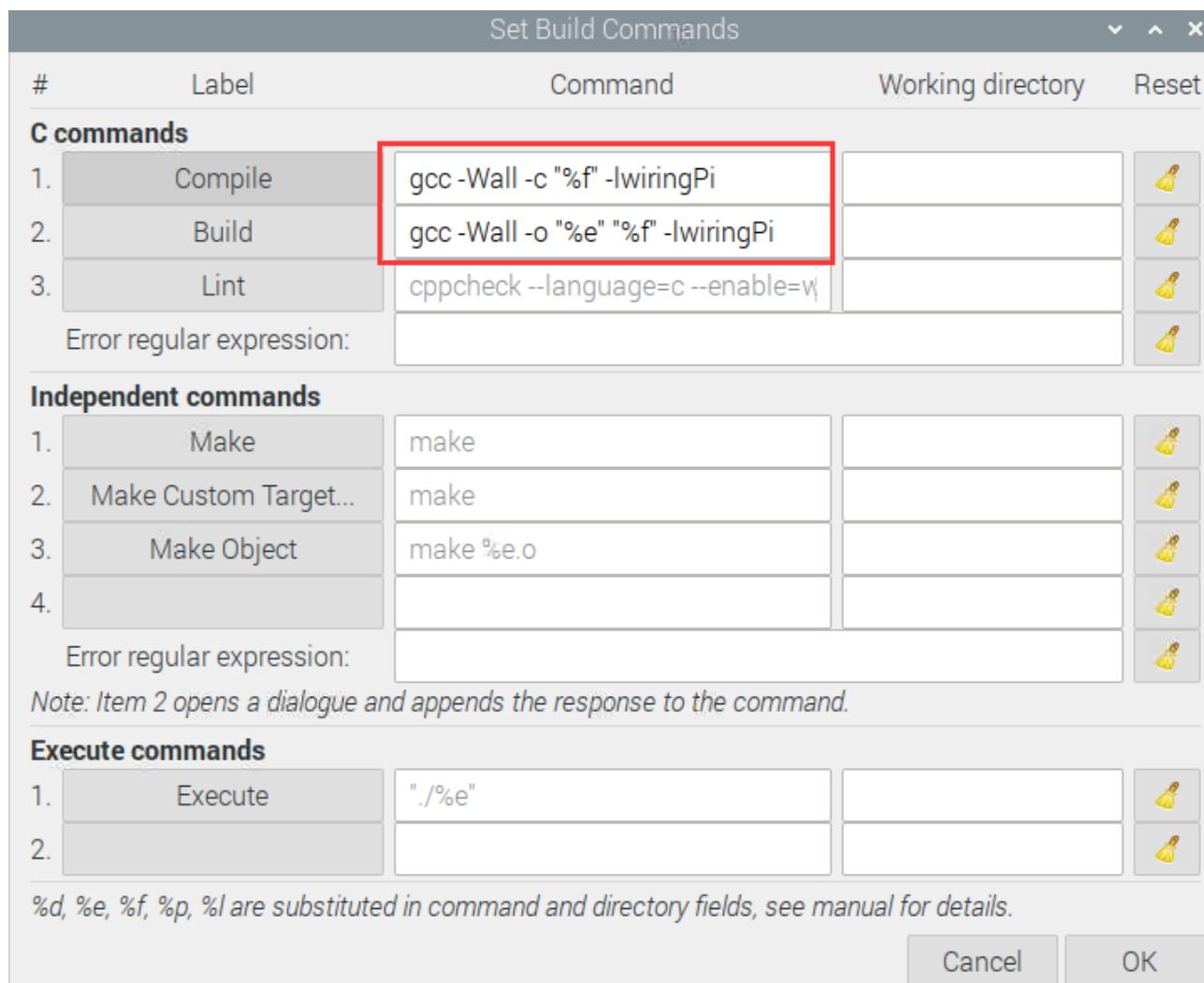


After the execution, a new terminal window will output the characters “Hello, World!”, as shown below:





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



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

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

Freenove Car, Robot and other products for Raspberry Pi

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

<https://www.amazon.com/freenove>

FNK0043 Freenove 4WD Smart Car Kit for Raspberry Pi



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

FNK0050 Freenove Robot Dog Kit for Raspberry Pi



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

FNK0052 Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi

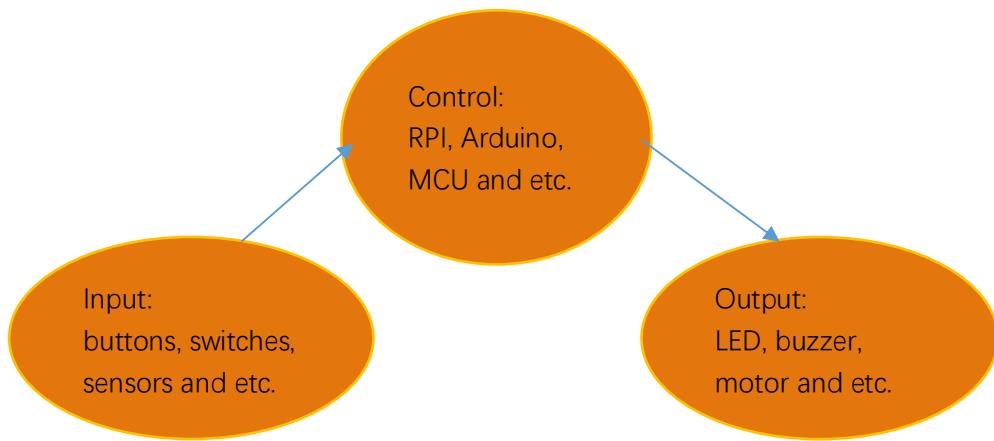
<https://youtu.be/LvghnJ2DNZ0>





Chapter 2 Buttons & LEDs

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



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

Project 2.1 Push Button Switch & LED

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

Component List

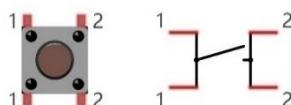
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1	Resistor 220Ω x1	Resistor 10kΩ x2	Push Button Switch x1
Jumper Wire				

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

Component knowledge

Push Button Switch

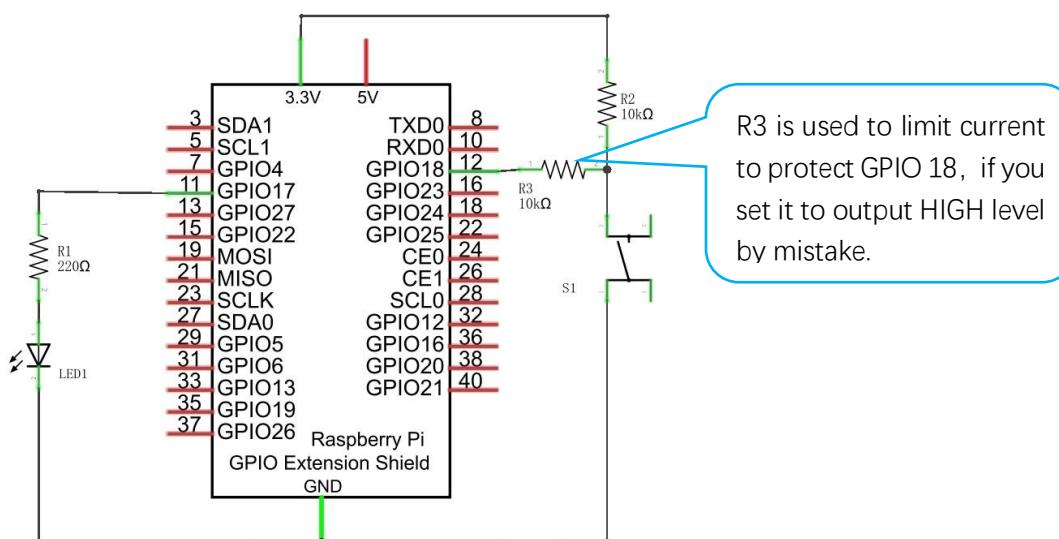
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



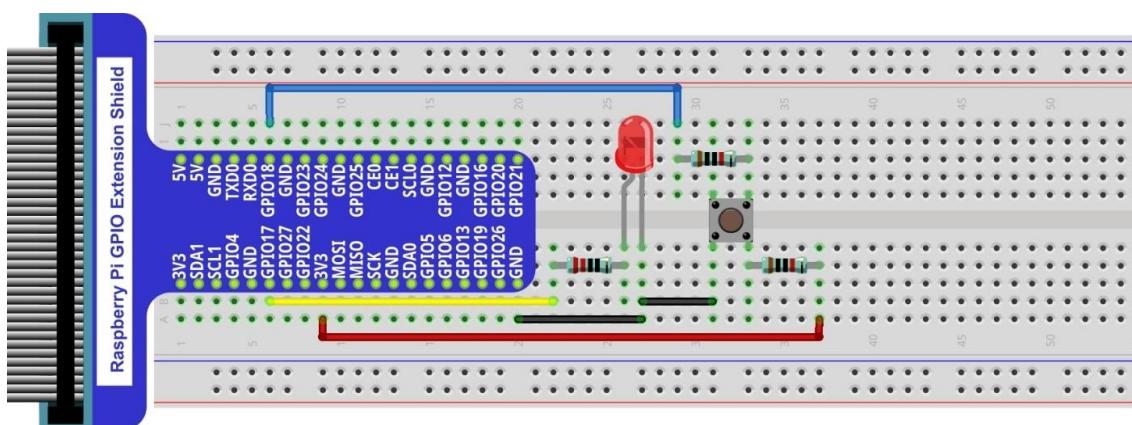
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



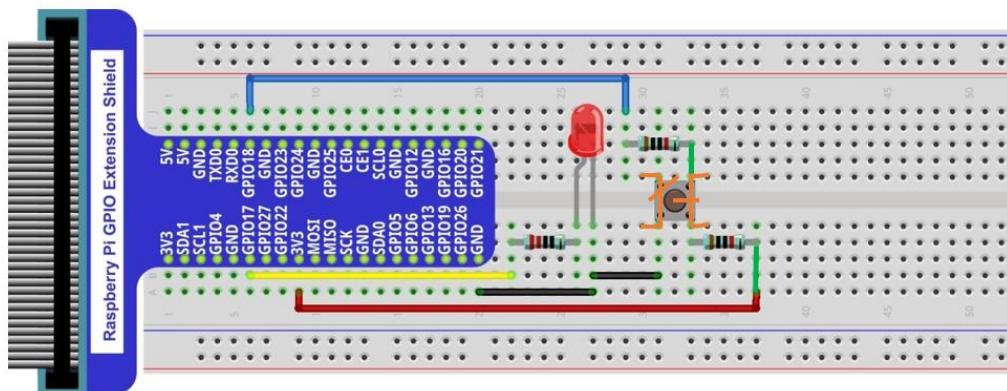
There are two kinds of push button switch in this kit.

The smaller push button switches are contained in a plastic bag.

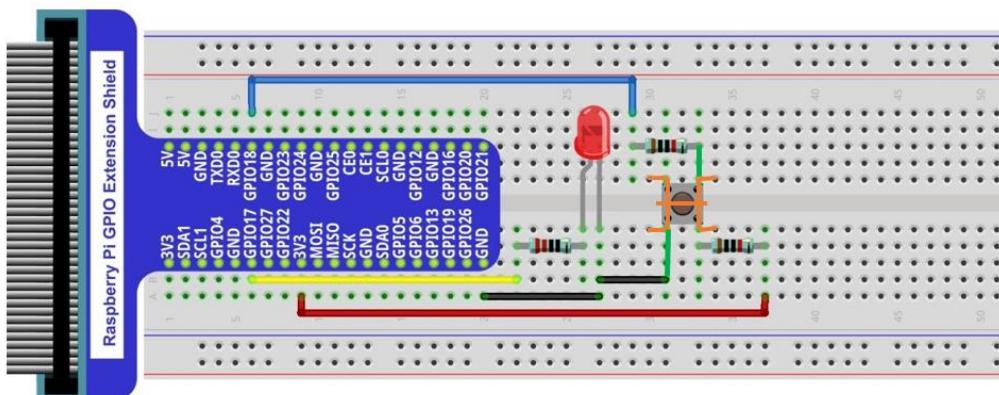
Youtube video: https://youtu.be/_5ge1d6f1nM

This is how it works.

When button switch is released:



When button switch is pressed:



Code

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

C Code 2.1.1 ButtonLED

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

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

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

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

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

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

3. Then run the generated file "ButtonLED".

```
sudo ./ButtonLED
```

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

The following is the program code:

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

```

2 #include <stdio.h>
3
4 #define ledPin    0 //define the ledPin
5 #define buttonPin 1 //define the buttonPin
6
7 void main(void)
8 {
9     printf("Program is starting ... \n");
10
11     wiringPiSetup(); //Initialize wiringPi.
12
13     pinMode(ledPin, OUTPUT); //Set ledPin to output
14     pinMode(buttonPin, INPUT); //Set buttonPin to input
15
16     pullUpDnControl(buttonPin, PUD_UP); //pull up to HIGH level
17     while(1) {
18         if(digitalRead(buttonPin) == LOW){ //button is pressed
19             digitalWrite(ledPin, HIGH); //Make GPIO output HIGH level
20             printf("Button is pressed, led turned on >>>\n"); //Output information on
21 terminal
22         }
23         else { //button is released
24             digitalWrite(ledPin, LOW); //Make GPIO output LOW level
25             printf("Button is released, led turned off <<<\n"); //Output information on
26 terminal
27         }
28     }
29 }
```

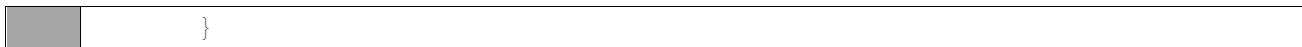
In the circuit connection, LED and Button are connected with GPIO17 and GPIO18 respectively, which correspond to 0 and 1 respectively in wiringPi. So define ledPin and buttonPin as 0 and 1 respectively.

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

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

```

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



Reference:

```
int digitalRead (int pin);
```

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

Python Code 2.1.1 ButtonLED

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

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

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

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

2. Use Python command to execute btnLED.py.

```
python ButtonLED.py
```

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

The following is the program code:

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

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

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

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

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

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

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

About function GPIO.input ():

GPIO.input()

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

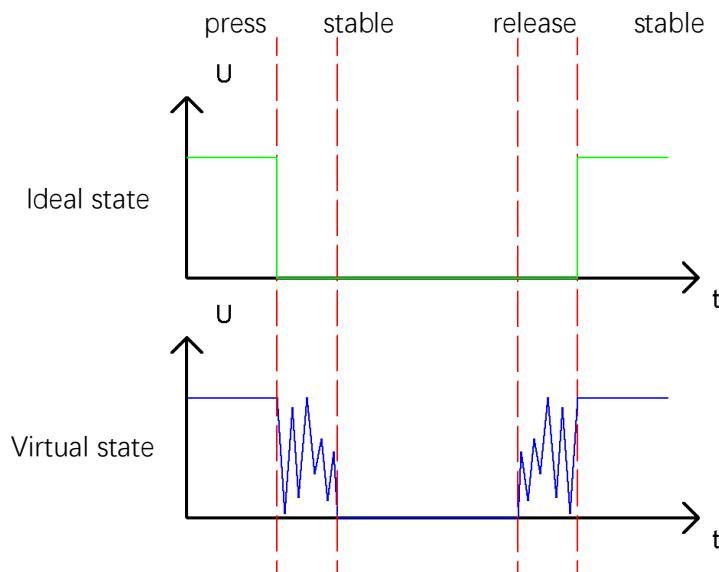
Project 2.2 MINI Table Lamp

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

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

Debounce a Push Button Switch

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it stabilizes in a new state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as "bounce".



Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

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

Code

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

C Code 2.2.1 Tablelamp

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

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

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

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

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

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

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

```
sudo ./Tablelamp
```

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

```

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

```

```

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

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

```

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

millis()

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

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

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

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

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

Python Code 2.2.1 Tablelamp

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

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

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

cd ~/Freenove_Kit/Code/Python_Code/02.2.1_Tablelamp

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

python Tablelamp.py

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

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

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

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

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

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

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

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

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

Chapter 3 LED Bar Graph

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

Project 3.1 Flowing Water Light

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

Component List

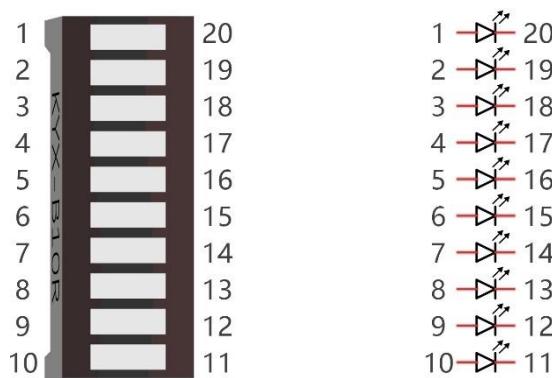
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Bar Graph LED x1	Resistor 220Ω x10
Jumper Wire x 1		

Component knowledge

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

Bar Graph LED

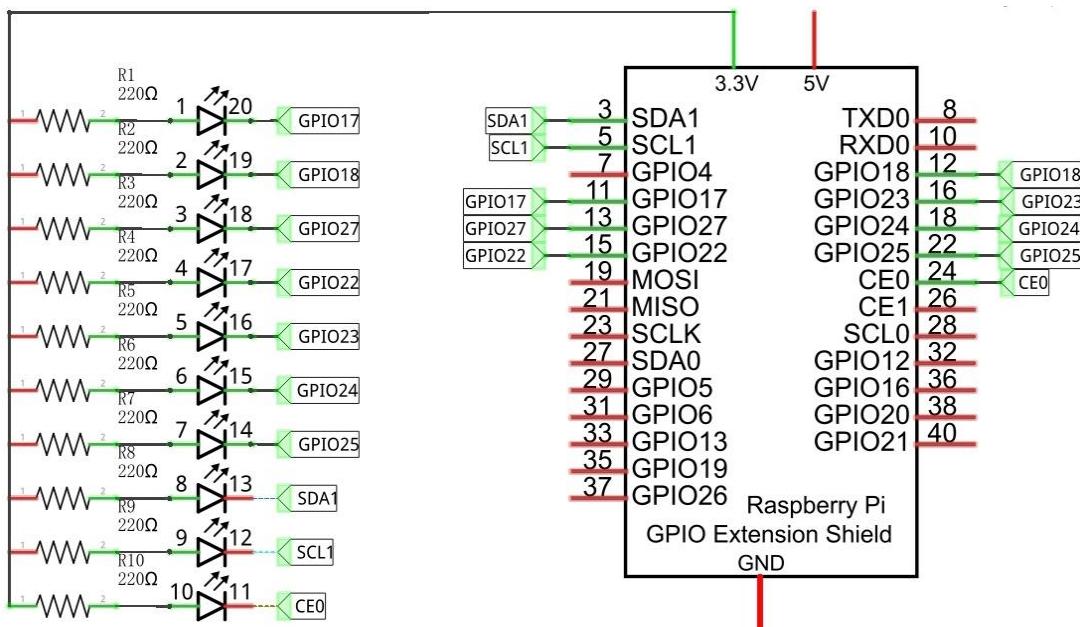
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



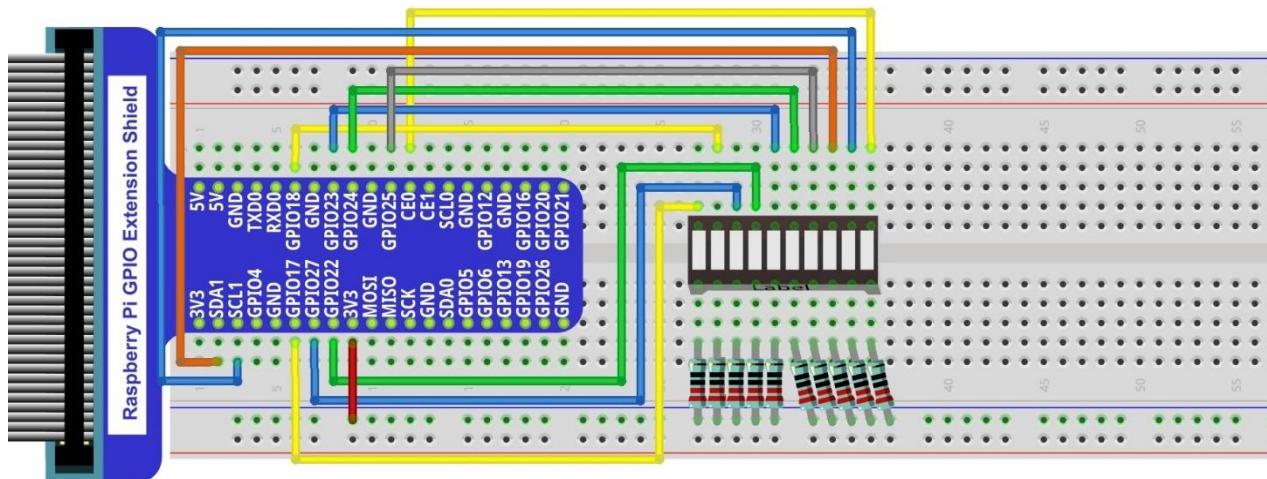
Circuit

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

Schematic diagram



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



If LEDbar doesn't work, rotate LEDbar 180° to try. The label is random.

Youtube video: <https://youtu.be/3rh-b05VoiU>

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

Code

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

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

C Code 3.1.1 LightWater

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

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

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

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

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

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

3. Then run the generated file “LightWater”.

```
sudo ./LightWater
```

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

The following is the program code:

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



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

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

Python Code 3.1.1 LightWater

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

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

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

cd ~/Freenove_Kit/Code/Python_Code/03.1.1_LightWater

2. Use Python command to execute Python code “LightWater.py”.

python LightWater.py

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

The following is the program code:

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

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

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

Chapter 4 Analog & PWM

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

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

Project 4.1 Breathing LED

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

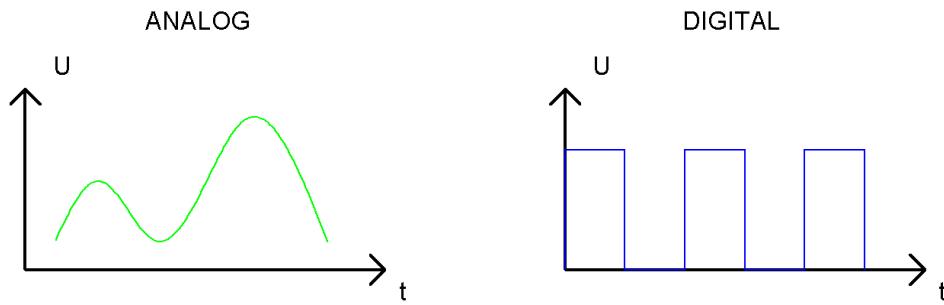
Component List

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

Component Knowledge

Analog & Digital

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



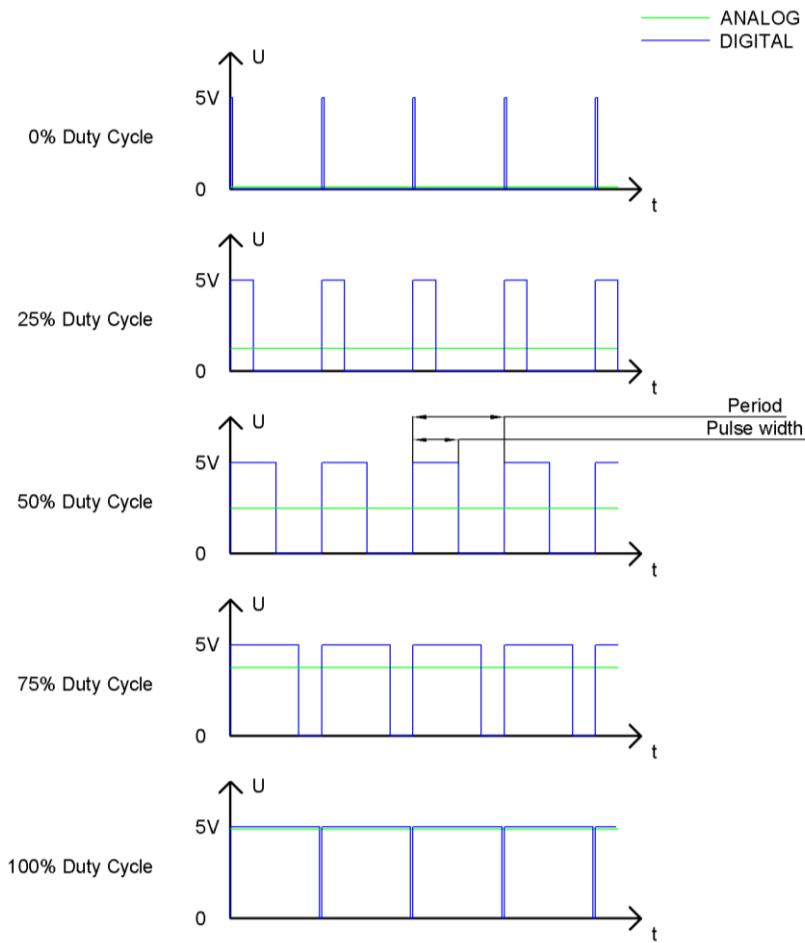
Note that the Analog signals are curved waves and the Digital signals are "Square Waves".

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

PWM

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

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



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

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

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

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

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

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

Circuit

Schematic diagram

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

Youtube video: <https://youtu.be/rYxykuVgYtA>

Code

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

C Code 4.1.1 BreathingLED

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

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

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

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

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

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

3. Then run the generated file "BreathingLED"

```
sudo ./BreathingLED
```

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

The following is the program code:

```

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

```

```

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

First, create a software PWM pin.

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

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

```

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

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

```
int softPwmCreate (int pin, int initialValue, int pwmRange);
```

This creates a software controlled PWM pin.

```
void softPwmWrite (int pin, int value);
```

This updates the PWM value on the given pin.

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

Python Code 4.1.1 BreathingLED

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

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

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

cd ~/Freenove_Kit/Code/Python_Code/04.1.1_BreathingLED

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

python BreathingLED.py

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2
3
4 LedPin = 12      # define the LedPin
5
6 def setup():
7     global p
8     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
9     GPIO.setup(LedPin, GPIO.OUT)   # set LedPin to OUTPUT mode
10    GPIO.output(LedPin, GPIO.LOW)  # make ledPin output LOW level to turn off LED
11
12    p = GPIO.PWM(LedPin, 500)     # set PWM Frequency to 500Hz
13    p.start(0)                  # set initial Duty Cycle to 0
14
15 def loop():
16     while True:
17         for dc in range(0, 101, 1):  # make the led brighter
18             p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
19             time.sleep(0.01)
20             time.sleep(1)
21         for dc in range(100, -1, -1): # make the led darker
22             p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
23             time.sleep(0.01)
24             time.sleep(1)
25
26 def destroy():
27     p.stop() # stop PWM
28     GPIO.cleanup() # Release all GPIO
29
30 if __name__ == '__main__':      # Program entrance
31     print ('Program is starting ... ')
32     setup()
33     try:
34         loop()
```

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

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

```
LedPin = 12      # define the LedPin

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

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

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

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

The related functions of PWM are described as follows:

p = GPIO.PWM(channel, frequency)

To create a PWM instance:

p.start(dc)

To start PWM, where dc is the duty cycle (0.0 <= dc <= 100.0)

p.ChangeFrequency(freq)

To change the frequency, where freq is the new frequency in Hz

p.ChangeDutyCycle(dc)

To change the duty cycle where 0.0 <= dc <= 100.0

p.stop()

To stop PWM.

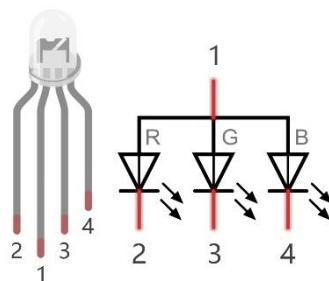
For more details regarding methods for using PWM with RPi.GPIO, please refer to:

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

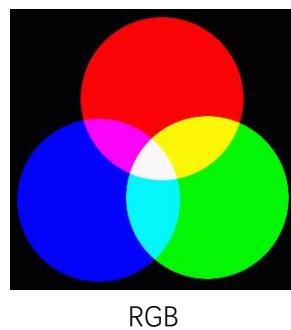
Chapter 5 RGB LED

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

An RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Anode (+) or positive lead, the other 3 are the Cathodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Cathodes (2, 3 & 4) of the RGB LED



Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon.



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

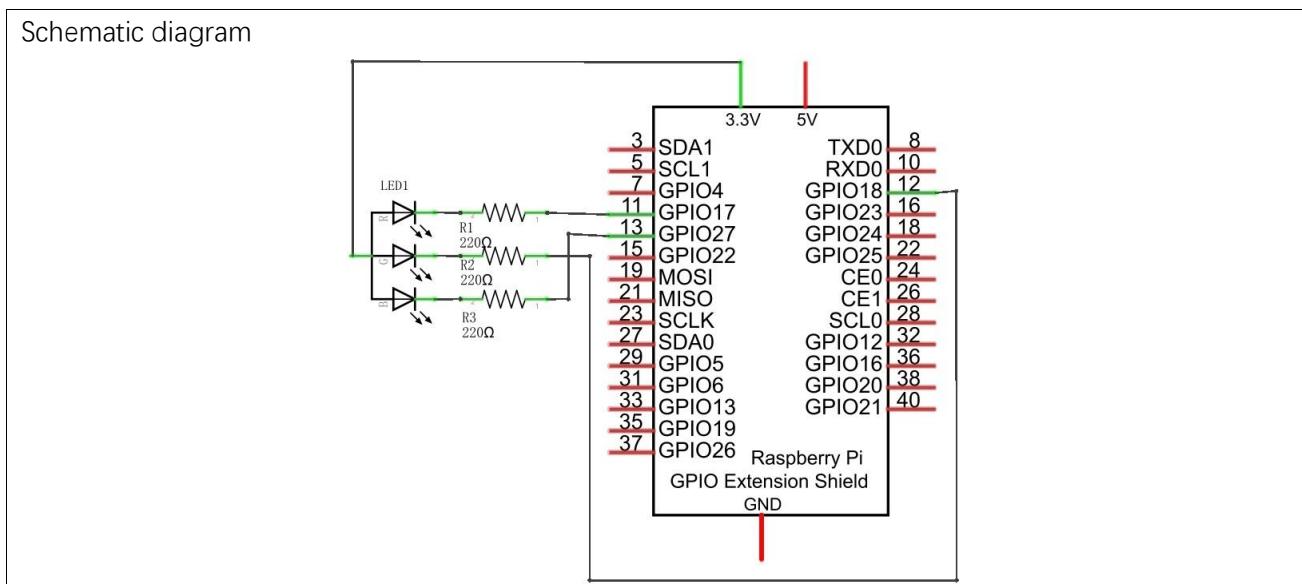
Next, we will use RGB LED to make a multicolored LED.

Project 5.1 Multicolored LED

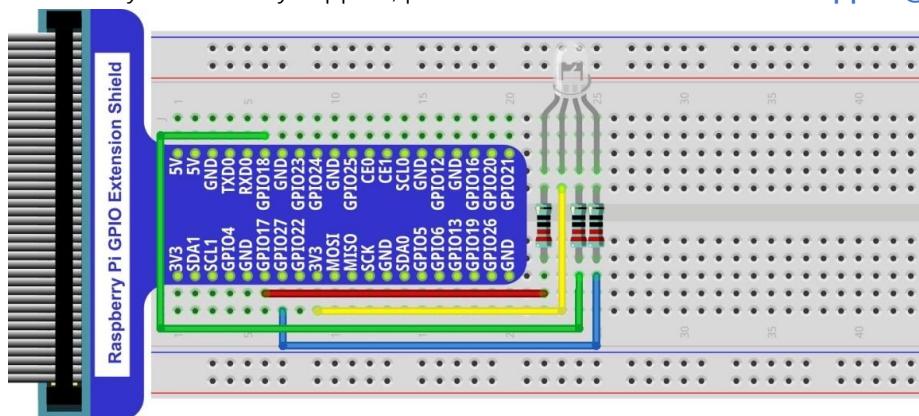
Component List

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

Circuit



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



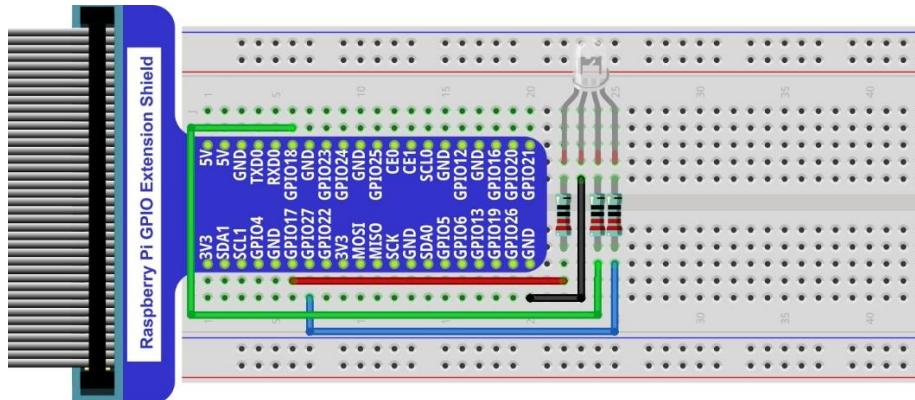
Video: <https://youtu.be/tbnX2AsX2y4>

In this kit, the RGB led is **Common anode**. The **voltage difference** between LED will make it work. There is

no visible GND. The GPIO ports can also receive current while in output mode.

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

There is no need to modify code for random color.



Code

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

C Code 5.1.1 Colorful LED

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

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

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

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

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

Note: in this project, the software PWM uses a multi-threading mechanism. So "-lpthread" option need to be add to the compiler.

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

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

```
sudo ./ColorfullLED
```

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

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <softPwm.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #define ledPinRed    0
7 #define ledPinGreen   1
8 #define ledPinBlue    2
9
10 void setupLedPin(void)
11 {

```



```

12     softPwmCreate(ledPinRed, 0, 100); //Creat SoftPWM pin for red
13     softPwmCreate(ledPinGreen, 0, 100); //Creat SoftPWM pin for green
14     softPwmCreate(ledPinBlue, 0, 100); //Creat SoftPWM pin for blue
15 }
16
17 void setLedColor(int r, int g, int b)
18 {
19     softPwmWrite(ledPinRed, r); //Set the duty cycle
20     softPwmWrite(ledPinGreen, g); //Set the duty cycle
21     softPwmWrite(ledPinBlue, b); //Set the duty cycle
22 }
23
24 int main(void)
25 {
26     int r,g,b;
27
28     printf("Program is starting ... \n");
29
30     wiringPiSetup(); //Initialize wiringPi.
31
32     setupLedPin();
33     while(1) {
34         r=random()%100; //get a random in (0,100)
35         g=random()%100; //get a random in (0,100)
36         b=random()%100; //get a random in (0,100)
37         setLedColor(r,g,b); //set random as the duty cycle value
38 // If you are using common anode RGBLED, it should be setLedColor(100-r, 100-g, 100-b)
39 // If you want show red, it should be setLedColor(0,100,100)
40         printf("r=%d, g=%d, b=%d \n",r,g,b);
41         delay(1000);
42     }
43     return 0;
44 }
```

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

```

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

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

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

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

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

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

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

long random();

This function will return a random number.

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

Python Code 5.1.1 ColorfulLED

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

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

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

cd ~/Freenove_Kit/Code/Python_Code/05.1.1_ColorfulLED

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

python ColorfulLED.py

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

The following is the program code:

```

1 import RPi.GPIO as GPIO
2 import time
3 import random
4
5 pins = [11, 12, 13]      # define the pins for R:11,G:12,B:13
6
7 def setup():
8     global pwmRed, pwmGreen, pwmBlue
9     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
10    GPIO.setup(pins, GPIO.OUT)    # set RGBLED pins to OUTPUT mode
11    GPIO.output(pins, GPIO.HIGH)  # make RGBLED pins output HIGH level
12    pwmRed = GPIO.PWM(pins[0], 2000)    # set PWM Frequency to 2kHz
13    pwmGreen = GPIO.PWM(pins[1], 2000)   # set PWM Frequency to 2kHz
14    pwmBlue = GPIO.PWM(pins[2], 2000)    # set PWM Frequency to 2kHz
15    pwmRed.start(0)      # set initial Duty Cycle to 0
16    pwmGreen.start(0)
17    pwmBlue.start(0)
18
19 def setColor(r_val, g_val, b_val):      # change duty cycle for three pins to r_val, g_val, b_val
20     pwmRed.ChangeDutyCycle(r_val)        # change pwmRed duty cycle to r_val
21     pwmGreen.ChangeDutyCycle(g_val)
22     pwmBlue.ChangeDutyCycle(b_val)
23
24 def loop():
25     while True :
26         r=random.randint(0,100)  #get a random in (0,100)
27         g=random.randint(0,100)
28         b=random.randint(0,100)
29         setColor(r,g,b)          #set random as a duty cycle value
30     # If you are using common anode RGBLED, it should be setColor(100-r, 100-g, 100-b)
31     # If you want show red, it should be setColor(0, 100, 100)
32         print (' r=%d, g=%d, b=%d ' %(r ,g, b))
33         time.sleep(1)
34
35 def destroy():

```

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

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

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

About the randint() function :

random.randint(a, b)

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

Chapter 6 Buzzer

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

Project 6.1 Doorbell

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

Component List

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

Component knowledge

Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.



Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

How to identify active and passive buzzer?

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



Active buzzer bottom



Passive buzzer bottom

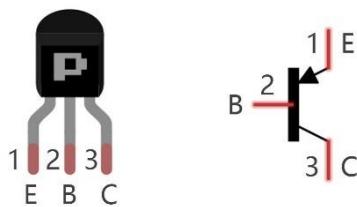
Transistors

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

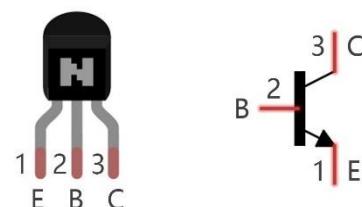
amplify the current.

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

PNP transistor



NPN transistor



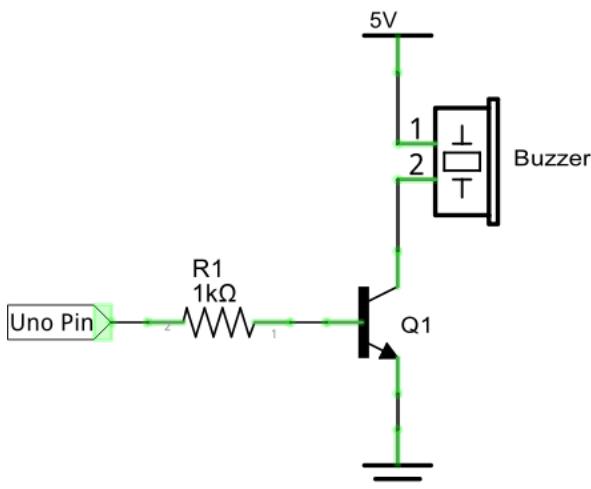
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

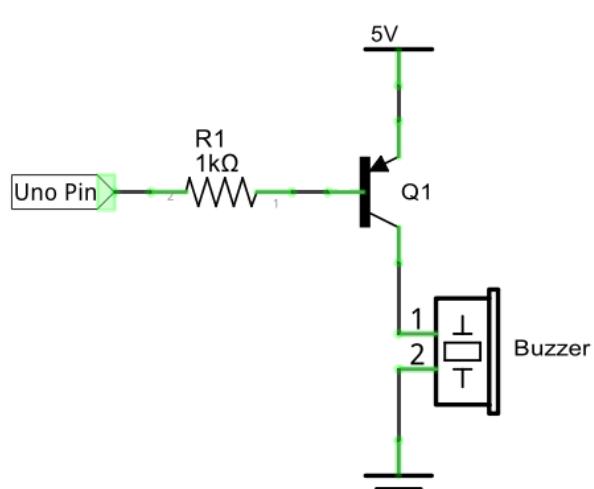
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not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

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

NPN transistor to drive buzzer

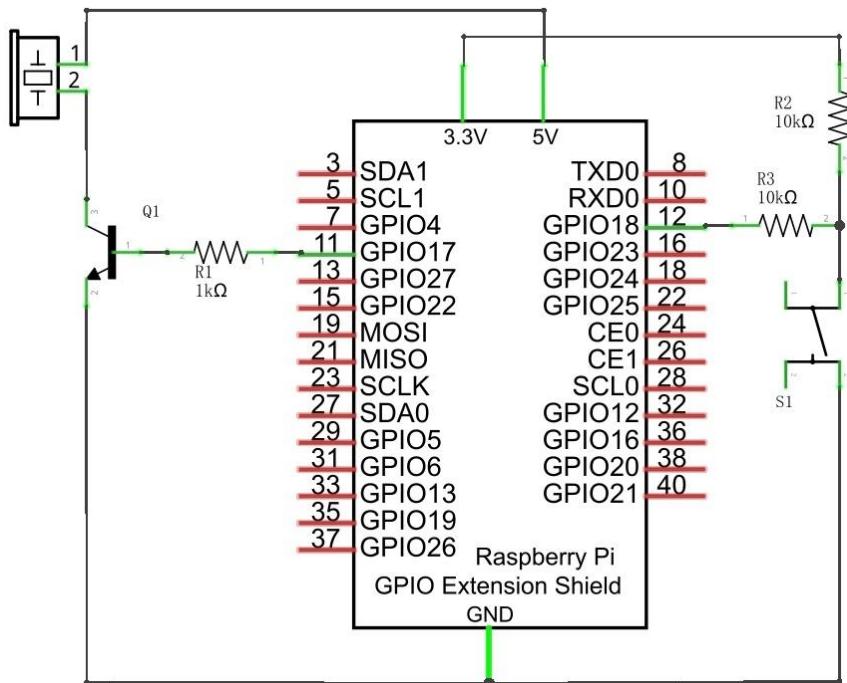


PNP transistor to drive buzzer

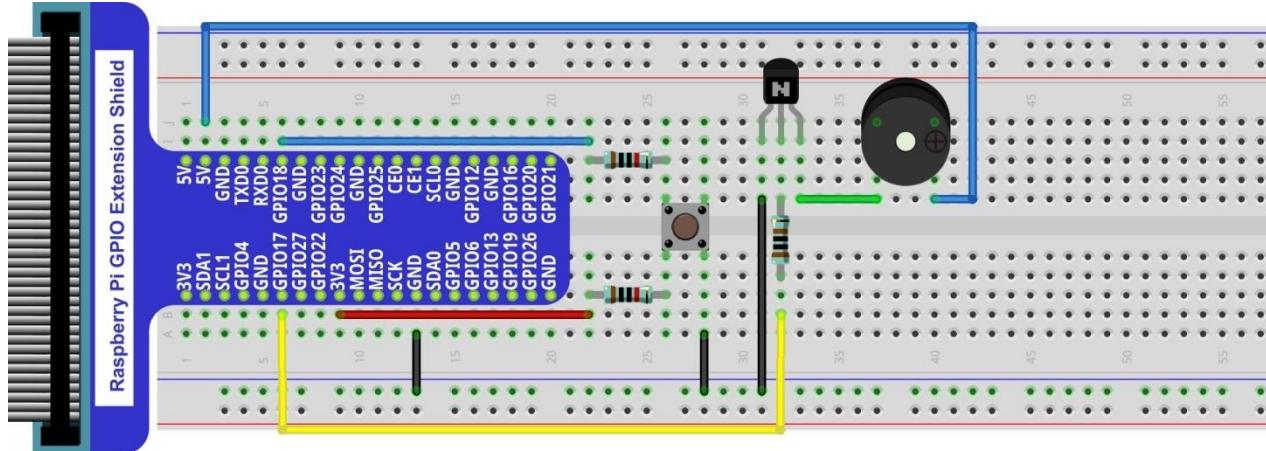


Circuit

Schematic diagram with RPi GPIO Extension Shield



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



Video: https://youtu.be/R_dmi3YwY-U

Note: in this circuit, the power supply for the buzzer is 5V, and pull-up resistor of the push button switch is connected to the 3.3V power feed. Actually, the buzzer can work when connected to the 3.3V power feed but this will produce a weak sound from the buzzer (not very loud).

Code

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

C Code 6.1.1 Doorbell

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

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

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

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

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

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

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

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

The following is the program code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define buzzerPin 0 //define the buzzerPin
5 #define buttonPin 1 //define the buttonPin
6
7 void main(void)
8 {
9     printf("Program is starting ... \n");
10
11     wiringPiSetup();
12
13     pinMode(buzzerPin, OUTPUT);
14     pinMode(buttonPin, INPUT);
15
16     pullUpDnControl(buttonPin, PUD_UP); //pull up to HIGH level
17     while(1) {
18
19         if(digitalRead(buttonPin) == LOW){ //button is pressed
20             digitalWrite(buzzerPin, HIGH); //Turn on buzzer
21             printf("buzzer turned on >>> \n");
22         }
23         else { //button is released
24             digitalWrite(buzzerPin, LOW); //Turn off buzzer
25             printf("buzzer turned off <<< \n");
26         }
27     }
28 }
```

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

Python Code 6.1.1 Doorbell

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

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

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

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

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

```
python Doorbell.py
```

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

The following is the program code:

```

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

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

Project 6.2 Alertor

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

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

Code

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

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

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

C Code 6.2.1 Alertor

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

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

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

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

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

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

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

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

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softTone.h>
4 #include <math.h>
5
6 #define buzzerPin 0 //define the buzzerPin
7 #define buttonPin 1 //define the buttonPin
8
9 void alertor(int pin) {
10     int x;
11     double sinVal, toneVal;
12     for(x=0;x<360;x++) { // frequency of the alertor is consistent with the sine wave
13         sinVal = sin(x * (M_PI / 180)); //Calculate the sine value
14         toneVal = 2000 + sinVal * 500; //Add the resonant frequency and weighted sine
15         value
16         softToneWrite(pin, toneVal); //output corresponding PWM

```



```

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

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

	softToneCreate (buzzerPin) ;
--	------------------------------

In the while loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

	void alertor(int pin) { int x; double sinVal, toneVal; for(x=0;x<360;x++){ //The frequency is based on the sine curve. sinVal = sin(x * (M_PI / 180)); softToneWrite(pin, sinVal * 500 + 500); delay(1); } }
--	--

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

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

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

The related functions of softTone are described as follows:

int softToneCreate (int pin) ;

This creates a software controlled tone pin.

void softToneWrite (int pin, int freq) ;

This updates the tone frequency value on the given pin.

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

Python Code 6.2.1 Alertor

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

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

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

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

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

```
python Alertor.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 import math
4
5 buzzerPin = 11      # define the buzzerPin
6 buttonPin = 12      # define the buttonPin
7
8 def setup():
9     global p
10    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
11    GPIO.setup(buzzerPin, GPIO.OUT)   # set RGBLED pins to OUTPUT mode
12    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set buttonPin to INPUT
13 mode, and pull up to HIGH level, 3.3V
14 p = GPIO.PWM(buzzerPin, 1)
15 p.start(0);
16
17 def loop():
18     while True:
19         if GPIO.input(buttonPin)==GPIO.LOW:
20             alertor()
21             print (' alertor turned on >>> ')
22         else :
23             stopAlertor()
24             print (' alertor turned off <<< ')
25 def alertor():
26     p.start(50)
27     for x in range(0,361):      # Make frequency of the alertor consistent with the sine wave
28         sinVal = math.sin(x * (math.pi / 180.0))          # calculate the sine value
29         toneVal = 2000 + sinVal * 500 # Add to the resonant frequency with a Weighted
30         p.ChangeFrequency(toneVal)      # Change Frequency of PWM to toneVal
31         time.sleep(0.001)
32
33 def stopAlertor():
```

```

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

```

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

```

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

```

In the while loop loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```

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

```

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

```

def stopAlertor():
    p.stop()

```



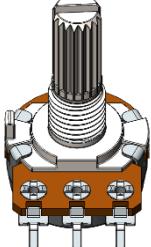
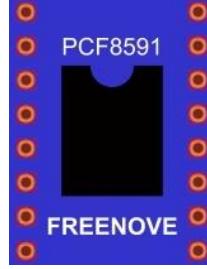
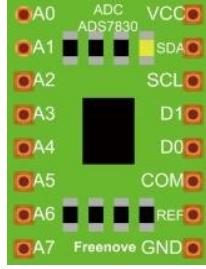
(Important) Chapter 7 ADC

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

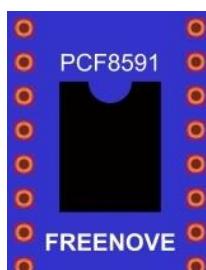
Project 7.1 Read the Voltage of Potentiometer

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

Component List

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

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

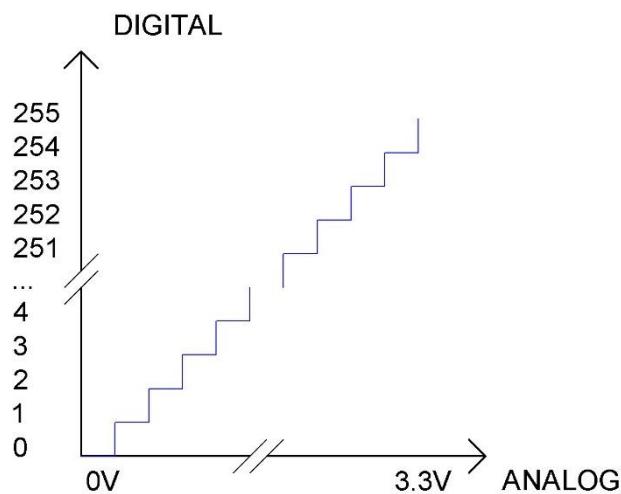
ADC module: PCF8591	ADC module: ADS7830
Model diagram 	Actual Picture 

Circuit knowledge

ADC

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

Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



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

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

...

The resultant analog signal will be divided accordingly.

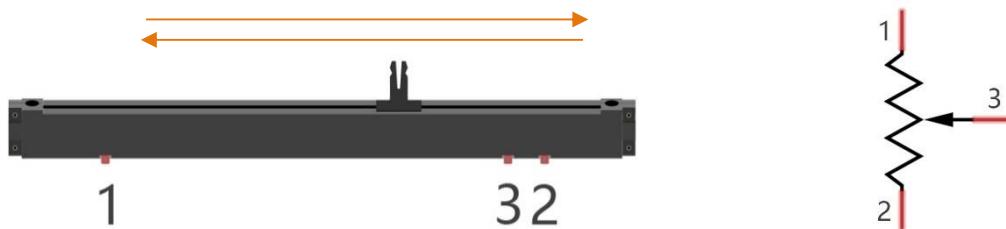
DAC

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

Component knowledge

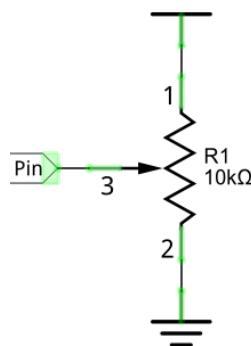
Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



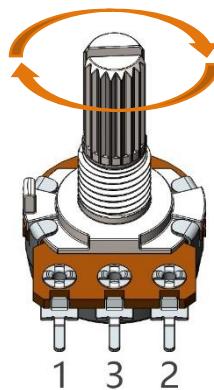
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

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



Rotary potentiometer

Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



PCF8591

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

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

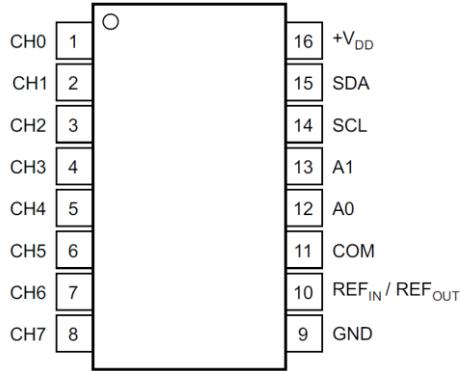
For more details about PCF8591, please refer to the datasheet which can be found on the Internet.

ADS7830

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

SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		

CH5	6		
CH6	7		
CH7	8		
GND	9	Ground	
REF in/out	10	Internal +2.5V Reference, External Reference Input	
COM	11	Common to Analog Input Channel	
A0	12	Hardware address	
A1	13		
SCL	14	Serial Clock	
SDA	15	Serial Sata	
+VDD	16	Power Supply, 3.3V Nominal	



The pinout diagram shows the MCP3208 chip with its 16 pins numbered 1 to 16. Pin 1 is labeled with an open circle. The pins and their functions are as follows:

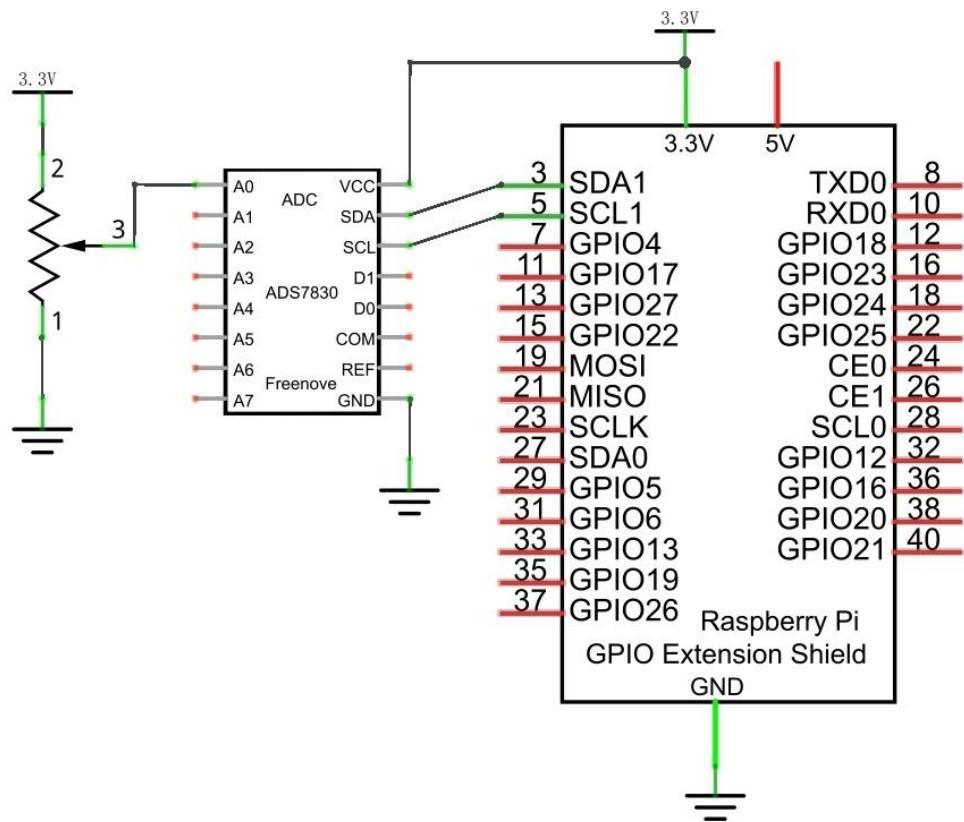
- Pin 1: GND
- Pin 2: CH1
- Pin 3: CH2
- Pin 4: CH3
- Pin 5: CH4
- Pin 6: CH5
- Pin 7: CH6
- Pin 8: CH7
- Pin 9: GND
- Pin 10: REF_{IN} / REF_{OUT}
- Pin 11: COM
- Pin 12: A0
- Pin 13: A1
- Pin 14: SCL
- Pin 15: SDA
- Pin 16: +V_{DD}

I2C communication

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

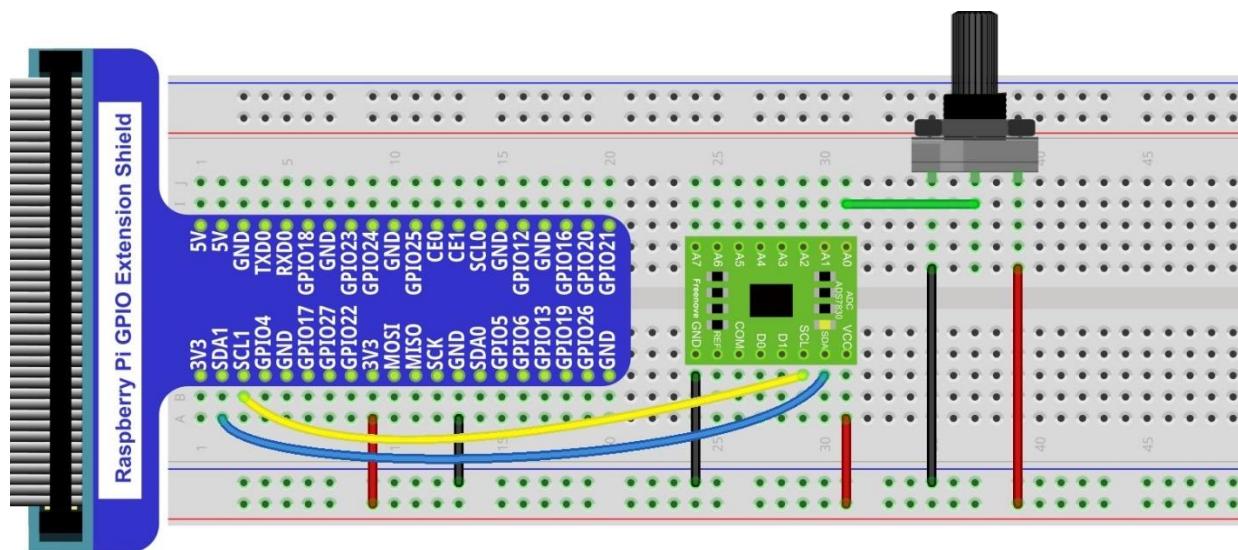
Circuit with ADS7830

Schematic diagram



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

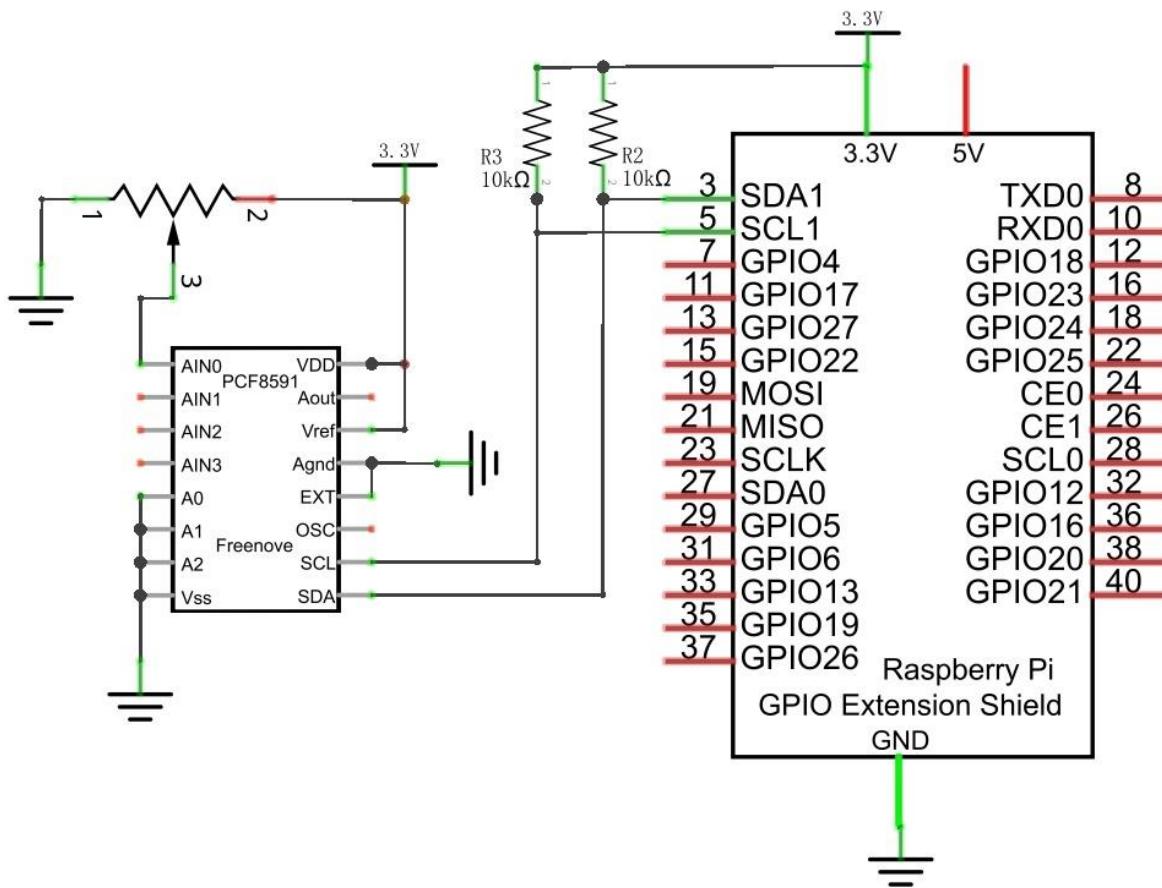
This product contains **only one ADC module**.



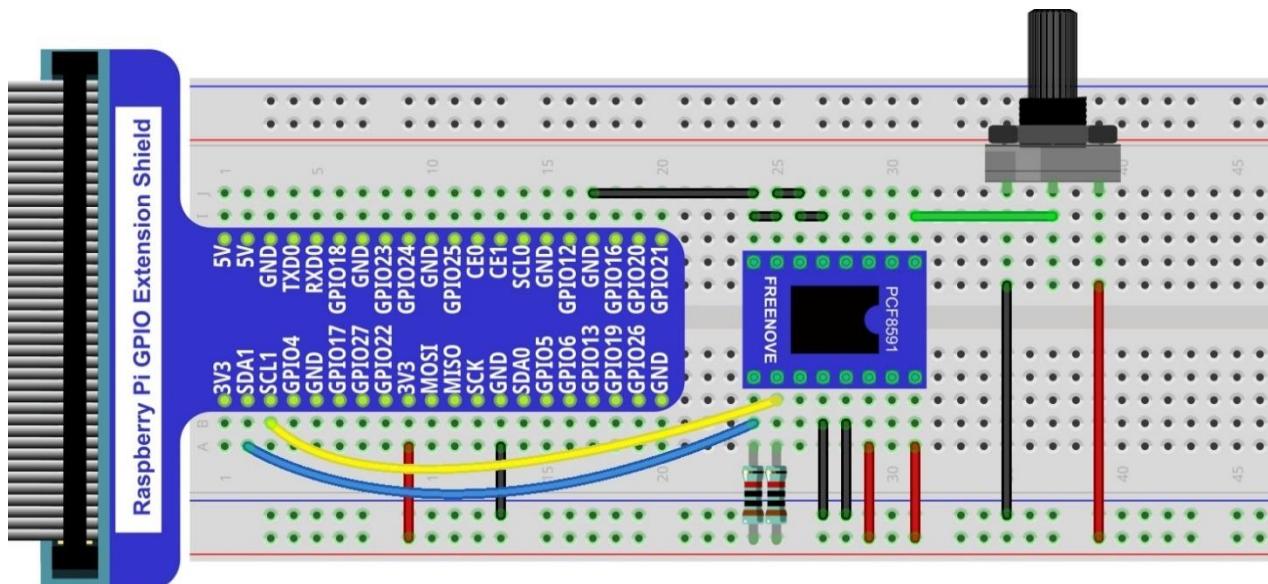
[Video: https://youtu.be/PSUCctu_DqA](https://youtu.be/PSUCctu_DqA)

Circuit with PCF8591

Schematic diagram



Hardware connection



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

Configure I2C and Install Smbus

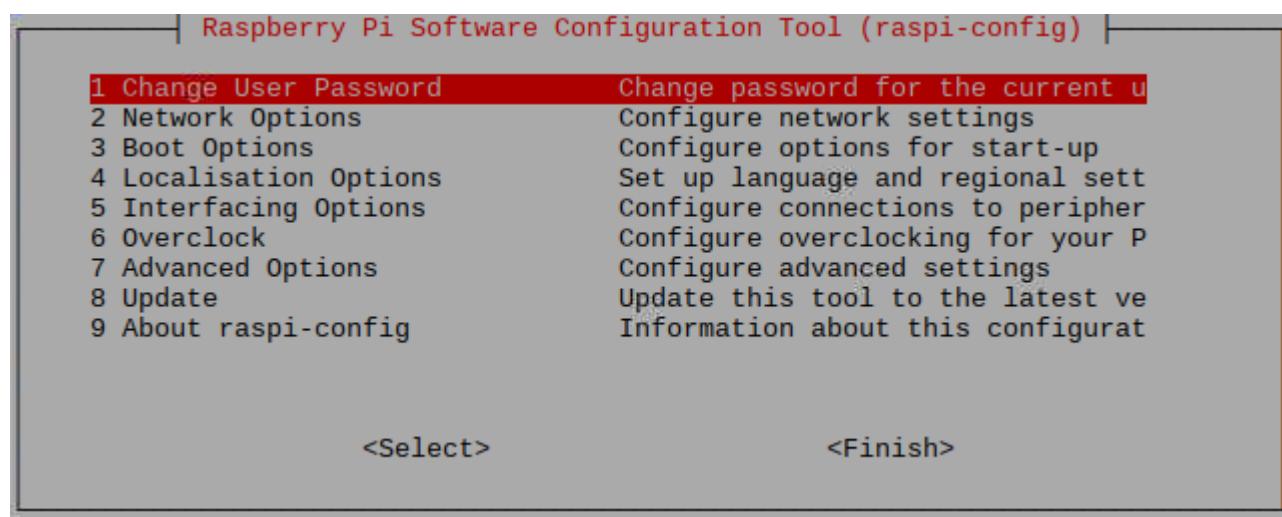
Enable I2C

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

Type command in the Terminal:

```
sudo raspi-config
```

Then open the following dialog box:



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

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

```
lsmod | grep i2c
```

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

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

Install I2C-Tools

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

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

I2C device address detection:

```
i2cdetect -y 1
```

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

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

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

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

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

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

Install Smbus Module

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

Code

C Code 7.1.1 ADC

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

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

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

2. Execute command below to install the library.

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

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

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

Next, we will execute the code for this project.

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

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

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

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

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

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

3. Then run the generated file "ADC".

```
sudo ./ADC
```

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

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

The following is the code:

```
1 #include <wiringPi.h>
2 #include <wiringPiI2C.h>
3 #include <stdio.h>
4 #include <ADCDevice.hpp>
5
6 ADCDevice *adc; // Define an ADC Device class object
7
8 int main(void) {
9     adc = new ADCDevice();
10    printf("Program is starting ... \n");
11
12    if(adc->detectI2C(0x48)){ // Detect the pcf8591.
13        delete adc;
14        adc = new PCF8591(); // If detected, create an instance of PCF8591.
15    }
16    else if(adc->detectI2C(0x4b)){// Detect the ads7830
17        delete adc;
18        adc = new ADS7830(); // If detected, create an instance of ADS7830.
```

```

19 }
20 else{
21     printf("No correct I2C address found, \n"
22         "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
23         "Program Exit. \n");
24     return -1;
25 }
26
27 while(1){
28     int adcValue = adc->analogRead(0); //read analog value of A0 pin
29     float voltage = (float)adcValue / 255.0 * 3.3; // Calculate voltage
30     printf("ADC value : %d , \tVoltage : %.2fV\n", adcValue, voltage);
31     delay(100);
32 }
33 }
```

In this code, a custom class library "ADCDevice" is used. It contains the method of utilizing the ADC Module in this project, through which the ADC Module can easily and quickly be used. In the code, you need to first create a class pointer adc, and then point to an instantiated object. (Note: An instantiated object is given a name and created in memory or on disk using the structure described within a class declaration.)

```

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

Then use the member function detectI2C(addr) in the class to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the different addresses, we can determine what the ADC module is in the circuit. When the correct module is detected, the pointer adc will point to the address of the object, and then the previously pointed content will be deleted to free memory. The default address of ADC module PCF8591 is 0x48, and that of ADC module ADS7830 is 0x4b.

```

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

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

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

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

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

Reference

```
class ADCDevice
```

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

```
int detectI2C(int addr);
```

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

```
virtual int analogRead(int chn);
```

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

```
class PCF8591:public ADCDevice
class ADS7830:public ADCDevice
```

These two classes are derived from the `ADCDevice` class and mainly implement the function `analogRead(chn)`.

```
int analogRead(int chn);
```

This returns the value read on the supplied analog input pin.

Parameter `chn`: For `PCF8591`, the range of `chn` is 0, 1, 2, 3. For `ADS7830`, the range of is 0, 1, 2, 3, 4, 5, 6, 7.

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

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

Python Code 7.1.1 ADC

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

1. Use cd command to enter folder of ADCDevice.

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

2. Unzip the file.

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

3. Open the unzipped folder.

```
cd ADCDevice-1.0.3
```

4. Install library for python2 and python3.

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

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

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

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

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

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

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

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

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

```
python ADC.py
```

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

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

The following is the code:

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

```

8     if(adc.detectI2C(0x48)): # Detect the pcf8591.
9         adc = PCF8591()
10    elif(adc.detectI2C(0x4b)): # Detect the ads7830
11        adc = ADS7830()
12    else:
13        print("No correct I2C address found, \n"
14            "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
15            "Program Exit. \n");
16        exit(-1)
17
18    def loop():
19        while True:
20            value = adc.analogRead(0)    # read the ADC value of channel 0
21            voltage = value / 255.0 * 3.3 # calculate the voltage value
22            print (' ADC Value : %d, Voltage : %.2f' %(value,voltage))
23            time.sleep(0.1)
24
25    def destroy():
26        adc.close()
27
28 if __name__ == '__main__': # Program entrance
29     print (' Program is starting ... ')
30     try:
31         setup()
32         loop()
33     except KeyboardInterrupt: # Press ctrl-c to end the program.
34         destroy()

```

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

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

Then in setup(), use detectI2C(addr), the member function of ADCDevice, to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the address, we can determine which ADC Module is in the circuit. When the correct module is detected, a device specific class object is created and assigned to adc. The default address of PCF8591 is 0x48, and that of ADS7830 is 0x4b.

```

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = PCF8591()
    elif(adc.detectI2C(0x4b)): # Detect the ads7830
        adc = ADS7830()
    else:
        print("No correct I2C address found, \n"

```

```
"Please use command 'i2cdetect -y 1' to check the I2C address! \n"
"Program Exit. \n");
exit(-1)
```

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

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

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

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

Reference

About smbus Module:

smbus Module

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

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

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

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

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

class ADCDevice(object)

This is a base class.

```
int detectI2C(int addr);
```

This is a member function, which is used to detect whether the device with the given I2C address exists. If it exists, it returns true. Otherwise, it returns false.

class PCF8591(ADCDevice) class ADS7830(ADCDevice)

These two classes are derived from the ADCDevice and the main function is analogRead(chn).

```
int analogRead(int chn);
```

This returns the value read on the supplied analog input pin.

Parameter chn: For PCF8591, the range of chn is 0, 1, 2, 3. For ADS7830, the range is 0, 1, 2, 3, 4, 5, 6, 7.

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

```
~/Freenove_Kit/Libs/Python-Libs/ADCDevice-1.0.2/src/ADCDevice/ADCdevice.py
```

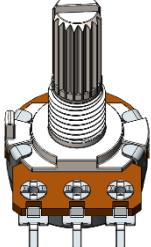
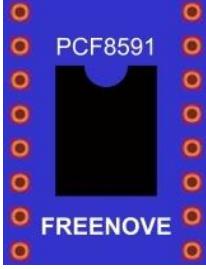
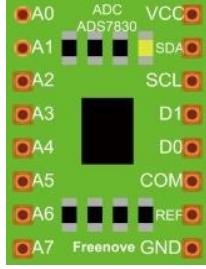
Chapter 8 Potentiometer & LED

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

Project 8.1 Soft Light

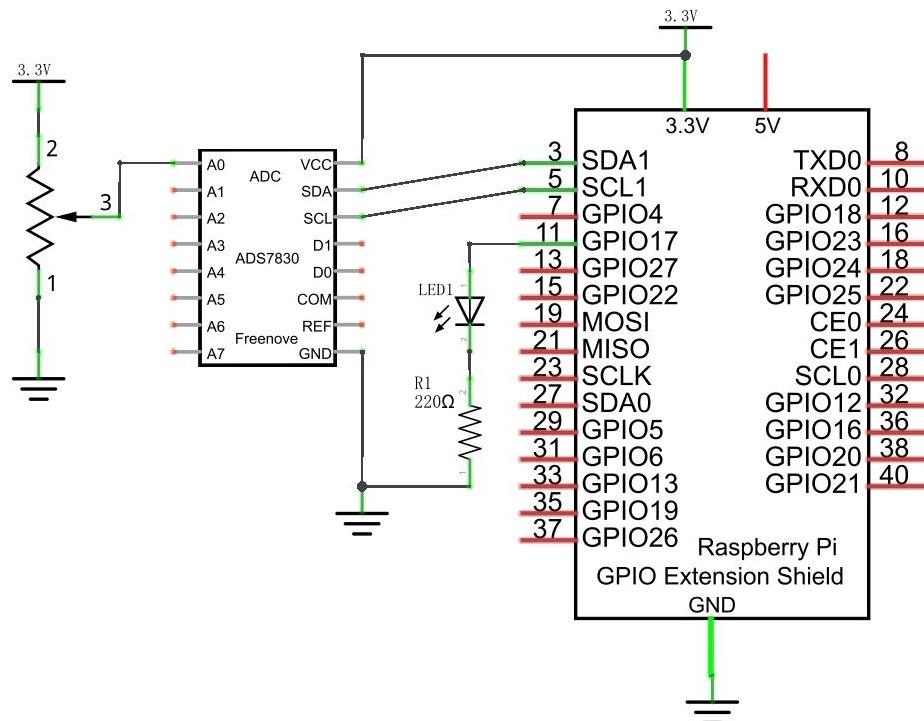
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle ratio of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

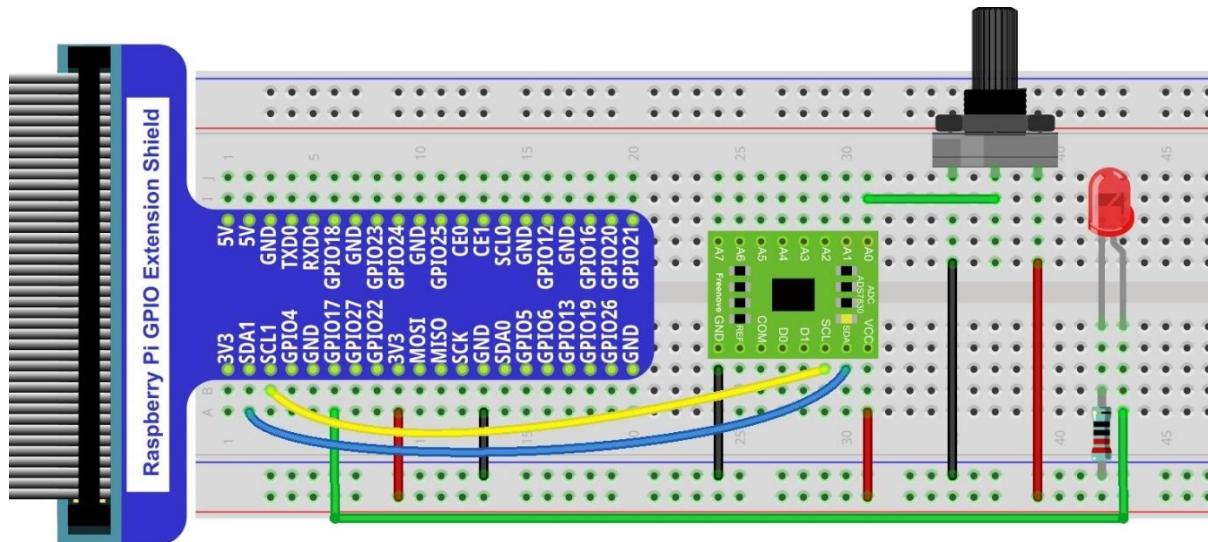
Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire M/M x17
Rotary Potentiometer x1 	ADC Module x1 (Only one)  Or 

Circuit with ADS7830

Schematic diagram



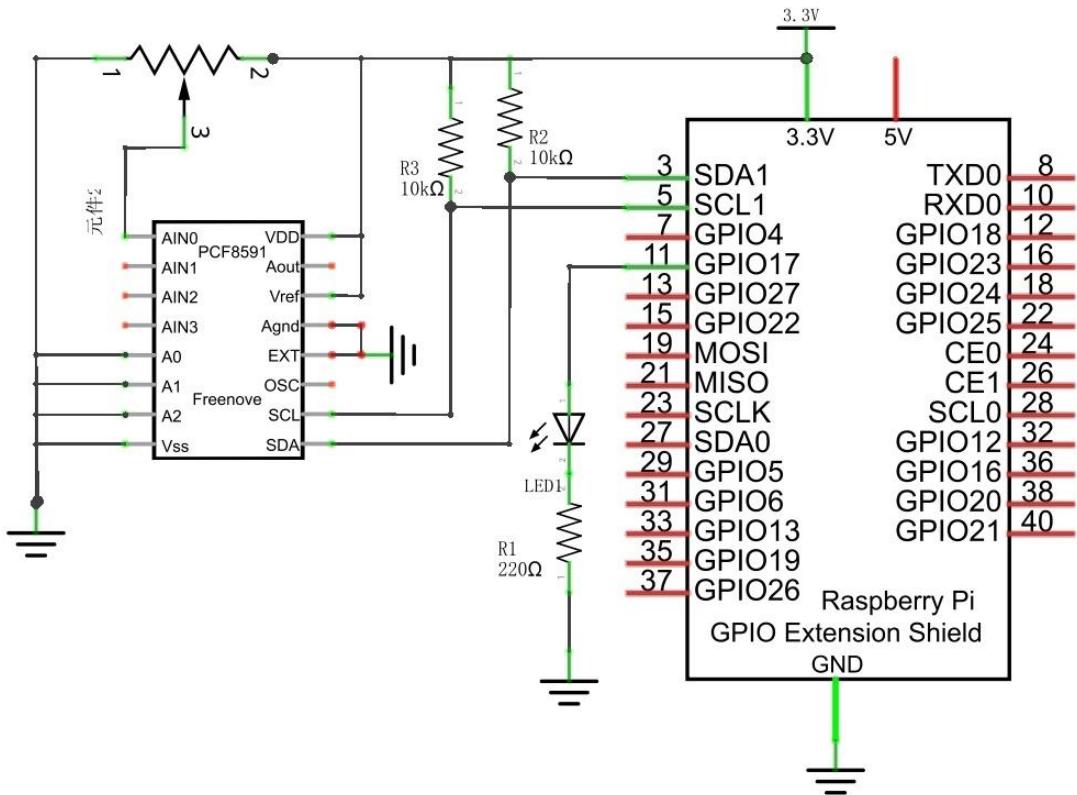
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



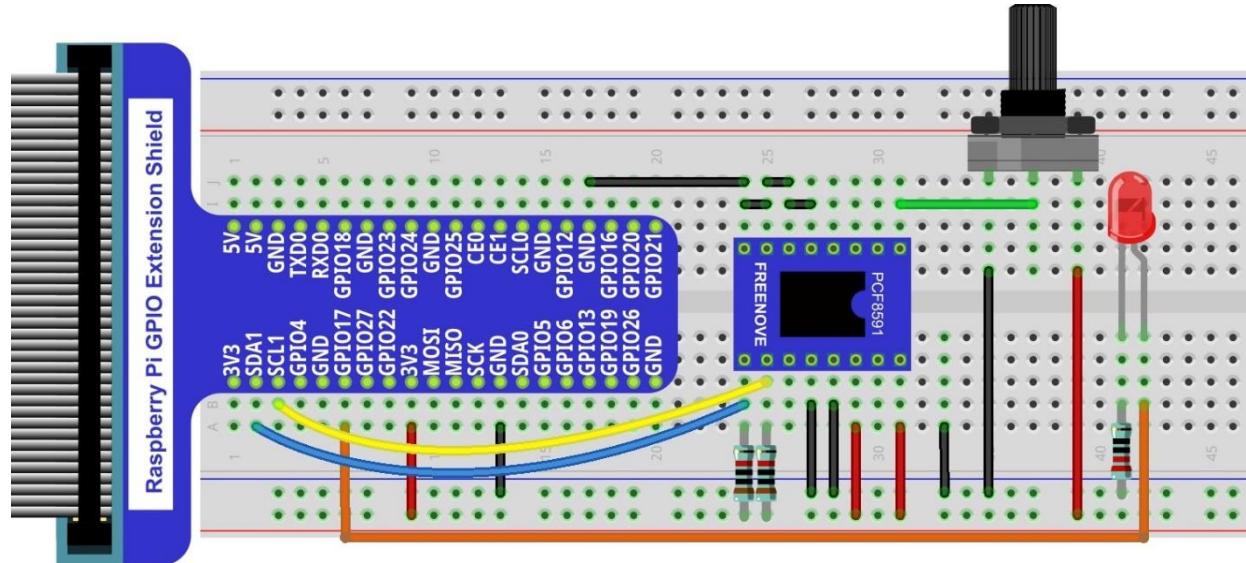
Video: <https://youtu.be/YMEfe9IWU6I>

Circuit with PCF8591

Schematic diagram



Hardware connection



Code

C Code 8.1.1 Softlight

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

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

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

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

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

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

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

3. Then run the generated file "Softlight".

```
sudo ./Softlight
```

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

The following is the code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define ledPin 0
7
8 ADCDevice *adc; // Define an ADC Device class object
9
10 int main(void) {
11     adc = new ADCDevice();
12     printf("Program is starting ... \n");
13
14     if(adc->detectI2C(0x48)){ // Detect the pcf8591.
15         delete adc;           // Free previously pointed memory
16         adc = new PCF8591(); // If detected, create an instance of PCF8591.
17     }
18     else if(adc->detectI2C(0x4b)){// Detect the ads7830
19         delete adc;           // Free previously pointed memory
20         adc = new ADS7830(); // If detected, create an instance of ADS7830.
21     }
22     else{
23         printf("No correct I2C address found, \n"
24             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
25             "Program Exit. \n");
26     }
27 }
```

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

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

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

Python Code 8.1.1 Softlight

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

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

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

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

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

```
python Softlight.py
```

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

The following is the code:

```

1 import RPi.GPIO as GPIO
2 import time
3 from ADCDevice import *
4
5 ledPin = 11
6 adc = ADCDevice() # Define an ADCDevice class object
7
8 def setup():
9     global adc
10    if(adc.detectI2C(0x48)): # Detect the pcf8591.
11        adc = PCF8591()
12    elif(adc.detectI2C(0x4b)): # Detect the ads7830
13        adc = ADS7830()
14    else:
15        print("No correct I2C address found, \n"
16              "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
17              "Program Exit. \n");
18        exit(-1)
19    global p
20    GPIO.setmode(GPIO.BCM)
21    GPIO.setup(ledPin,GPIO.OUT)
22    p = GPIO.PWM(ledPin,1000)
23    p.start(0)
24
25 def loop():
26    while True:
27        value = adc.analogRead(0)      # read the ADC value of channel 0
28        p.ChangeDutyCycle(value*100/255)          # Mapping to PWM duty cycle
29        voltage = value / 255.0 * 3.3 # calculate the voltage value

```

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

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

```
value = adc.analogRead(0)      # read the ADC value of channel 0
p.ChangeDutyCycle(value*100/255)      # Mapping to PWM duty cycle
```

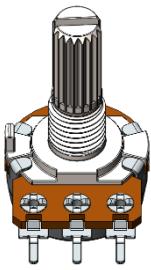
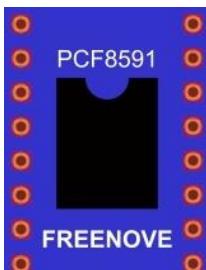
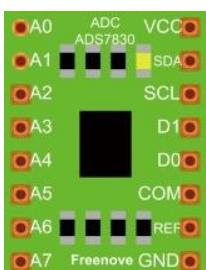
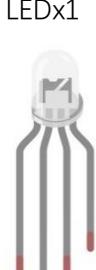
Chapter 9 Potentiometer & RGBLED

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

Project 9.1 Colorful Light

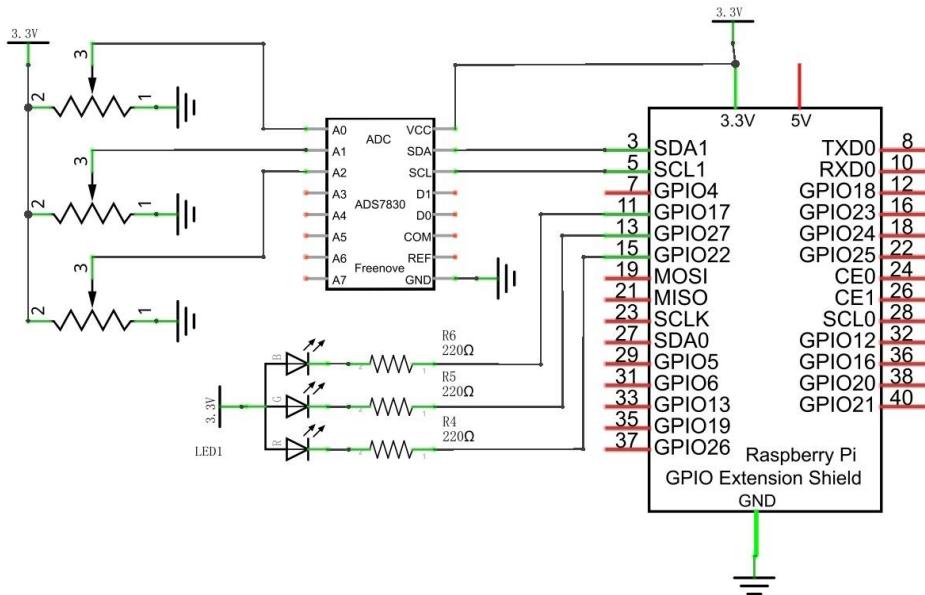
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as with the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the previous soft light project needed only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

Component List

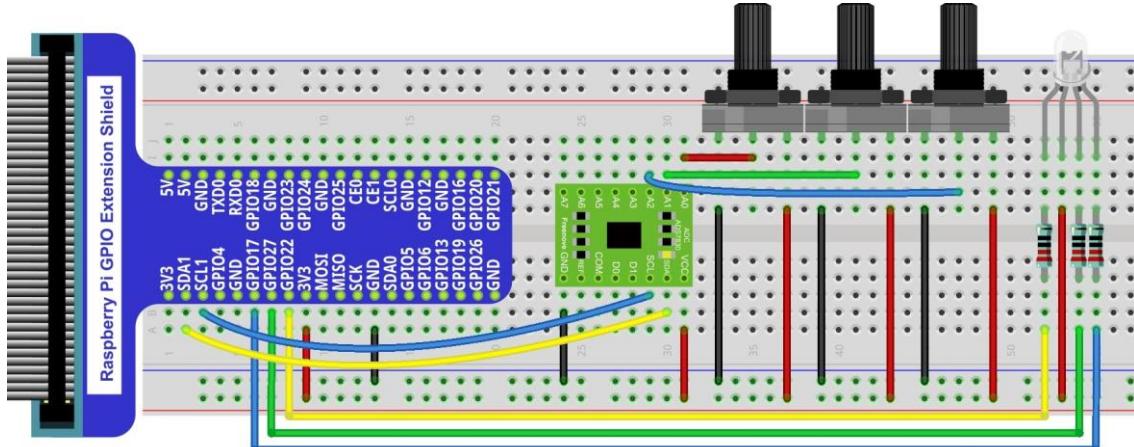
Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wires M/M x17			
Rotary potentiometer x3 	ADC module x1  or 	10kΩ x2 	220Ω x3 	RGB LEDx1 

Circuit with ADS7830

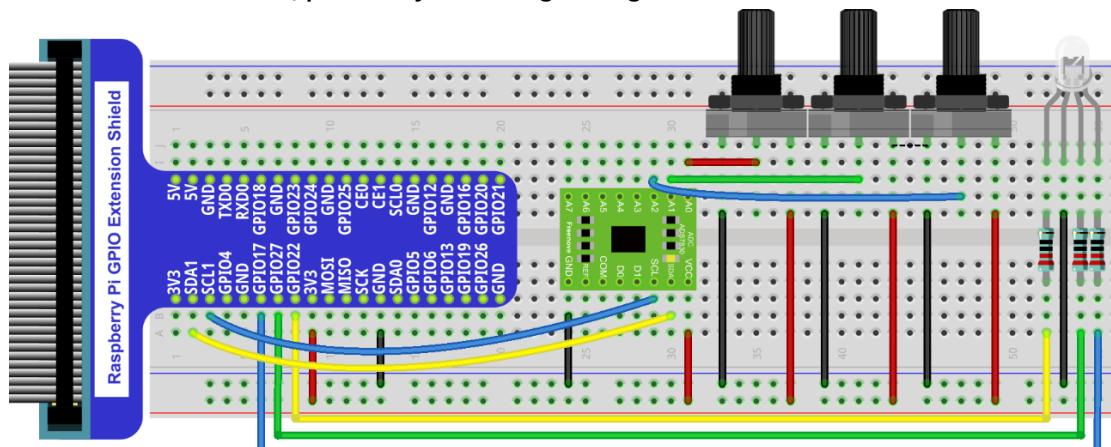
Schematic diagram



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



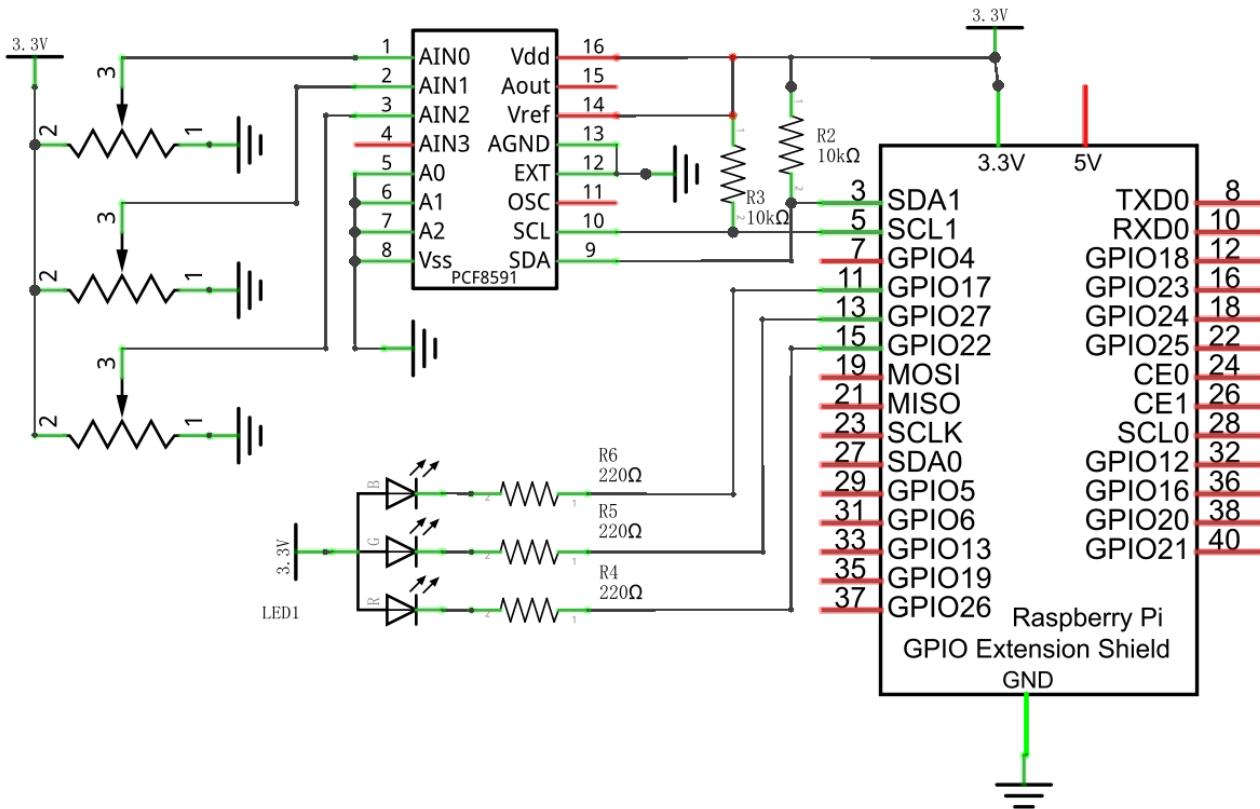
If circuit above doesn't work, please try following wiring.



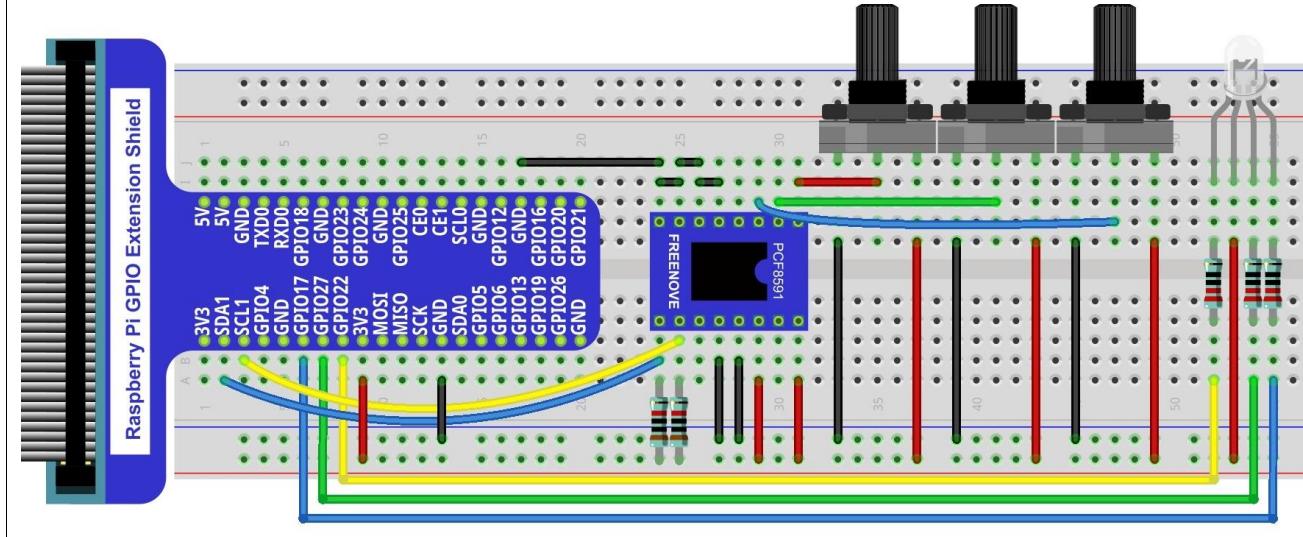
Video:

Circuit with PCF8591

Schematic diagram



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



Code

C Code 9.1.1 Colorful Softlight

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

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

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

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

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

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

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

3. Then run the generated file "ColorfulSoftlight".

```
sudo ./ColorfulSoftlight
```

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

```
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:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define ledRedPin 3           //define 3 pins for RGBLED
7 #define ledGreenPin 2
8 #define ledBluePin 0
9
10 ADCDevice *adc; // Define an ADC Device class object
11
12 int main(void) {
13     adc = new ADCDevice();
14     printf("Program is starting ... \n");
15
16     if(adc->detectI2C(0x48)){ // Detect the pcf8591.
17         delete adc;          // Free previously pointed memory
```

```

18         adc = new PCF8591();    // If detected, create an instance of PCF8591.
19     }
20     else if(adc->detectI2C(0x4b)){// Detect the ads7830
21         delete adc;           // Free previously pointed memory
22         adc = new ADS7830();   // If detected, create an instance of ADS7830.
23     }
24     else{
25         printf("No correct I2C address found, \n"
26             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
27             "Program Exit. \n");
28         return -1;
29     }
30     wiringPiSetup();
31     softPwmCreate(ledRedPin, 0, 100);    //creat 3 PMW output pins for RGBLED
32     softPwmCreate(ledGreenPin, 0, 100);
33     softPwmCreate(ledBluePin, 0, 100);
34     while(1){
35         int val_Red = adc->analogRead(0); //read analog value of 3 potentiometers
36         int val_Green = adc->analogRead(1);
37         int val_Blue = adc->analogRead(2);
38         softPwmWrite(ledRedPin, val_Red*100/255); //map the read value of
39         potentiometers into PWM value and output it
40         softPwmWrite(ledGreenPin, val_Green*100/255);
41         softPwmWrite(ledBluePin, val_Blue*100/255);
42         //print out the read ADC value
43         printf("ADC value val_Red: %d ,\tval_Green: %d ,\tval_Blue: %d
44 \n", val_Red, val_Green, val_Blue);
45         delay(100);
46     }
47     return 0;
48 }
```

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

Python Code 9.1.1 ColorfulSoftlight

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

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

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

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

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

```
python ColorfulSoftlight.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 from ADCDevice import *
4
5 ledRedPin = 15      # define 3 pins for RGBLED
6 ledGreenPin = 13
7 ledBluePin = 11
8 adc = ADCDevice() # Define an ADCDevice class object
9
10 def setup():
11     global adc
12     if(adc.detectI2C(0x48)): # Detect the pcf8591.
13         adc = PCF8591()
14     elif(adc.detectI2C(0x4b)): # Detect the ads7830
15         adc = ADS7830()
16     else:
17         print("No correct I2C address found, \n"
18             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
19             "Program Exit. \n");
20         exit(-1)
21
22     global p_Red, p_Green, p_Blue
23     GPIO.setmode(GPIO.BCM)
24     GPIO.setup(ledRedPin,GPIO.OUT)      # set RGBLED pins to OUTPUT mode
25     GPIO.setup(ledGreenPin,GPIO.OUT)
26     GPIO.setup(ledBluePin,GPIO.OUT)
27
28     p_Red = GPIO.PWM(ledRedPin,1000)    # configure PMW for RGBLED pins, set PWM
29     Frequency to 1kHz
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
36 def loop():
37     while True:
38         value_Red = adc.analogRead(0)          # read ADC value of 3 potentiometers
39         value_Green = adc.analogRead(1)
40         value_Blue = adc.analogRead(2)
41     #if you are using common anode RGBLED, it should be
42     #    value_Red = 255-adc.analogRead(0)
43     #    value_Green = 255-adc.analogRead(1)
44     #    value_Blue = 255-adc.analogRead(2)
45
46         p_Red.ChangeDutyCycle(value_Red*100/255)
47     # map the read value of potentiometers into PWM value and output it
48         p_Green.ChangeDutyCycle(value_Green*100/255)
49         p_Blue.ChangeDutyCycle(value_Blue*100/255)
50     # print read ADC value
51
52
53     print (' ADC Value
54 value_Red: %d , \tvalue_Green: %d , \tvalue_Blue: %d' %(value_Red, value_Green, value_Blue))
55     time.sleep(0.01)
56
57 def destroy():
58     adc.close()
59     GPIO.cleanup()
60
61 if __name__ == '__main__': # Program entrance
62     print (' Program is starting ... ')
63     setup()
64     try:
65         loop()
66     except KeyboardInterrupt: # Press ctrl-c to end the program.
67         destroy()
```

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

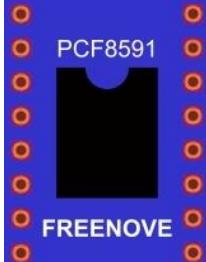
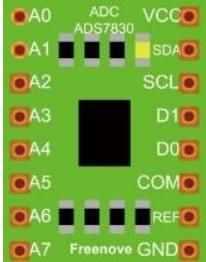
Chapter 10 Photoresistor & LED

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

Project 10.1 NightLamp

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

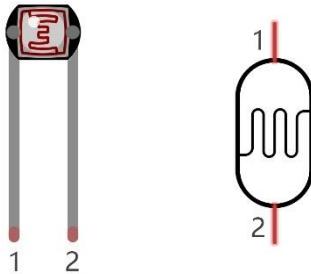
Component List

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

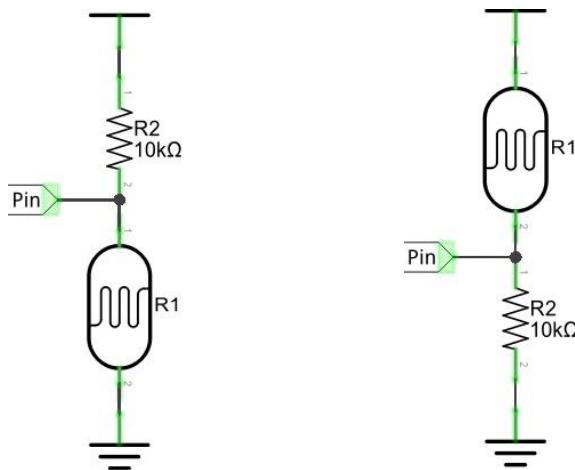
Component knowledge

Photoresistor

A Photoresistor is simply a light sensitive resistor. It is an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface. A Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.



The circuit below is used to detect the change of a Photoresistor's resistance value:

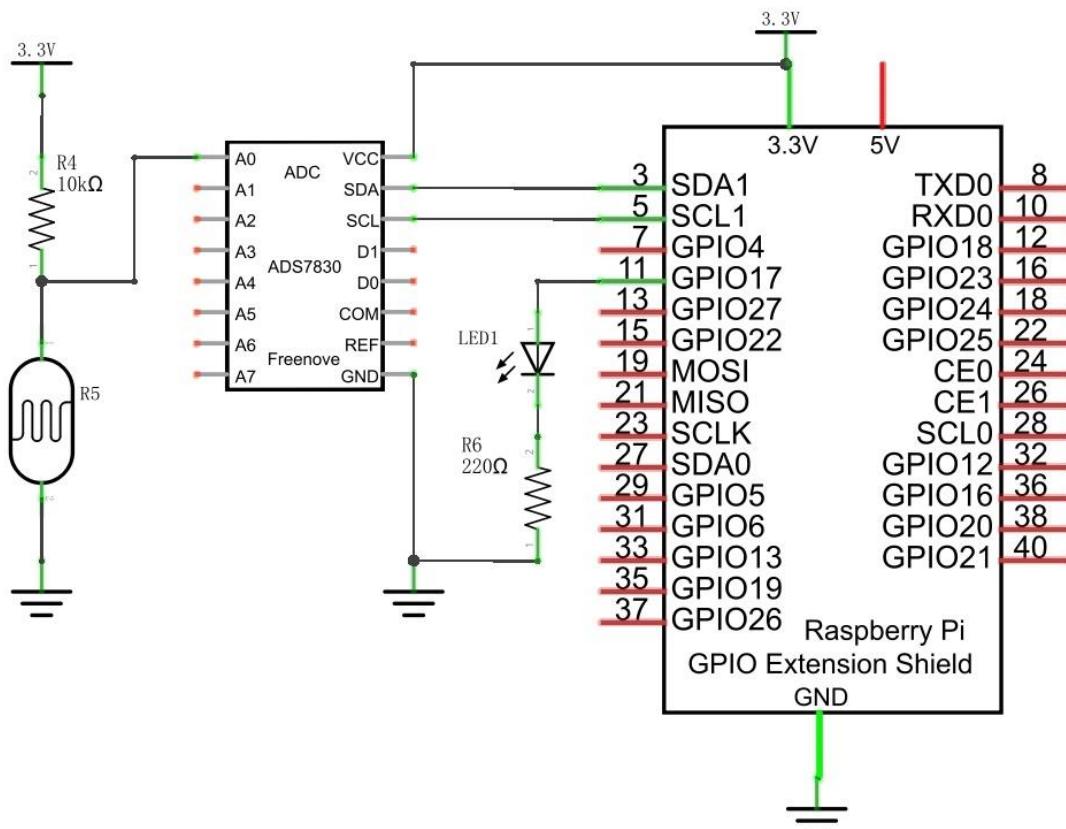


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

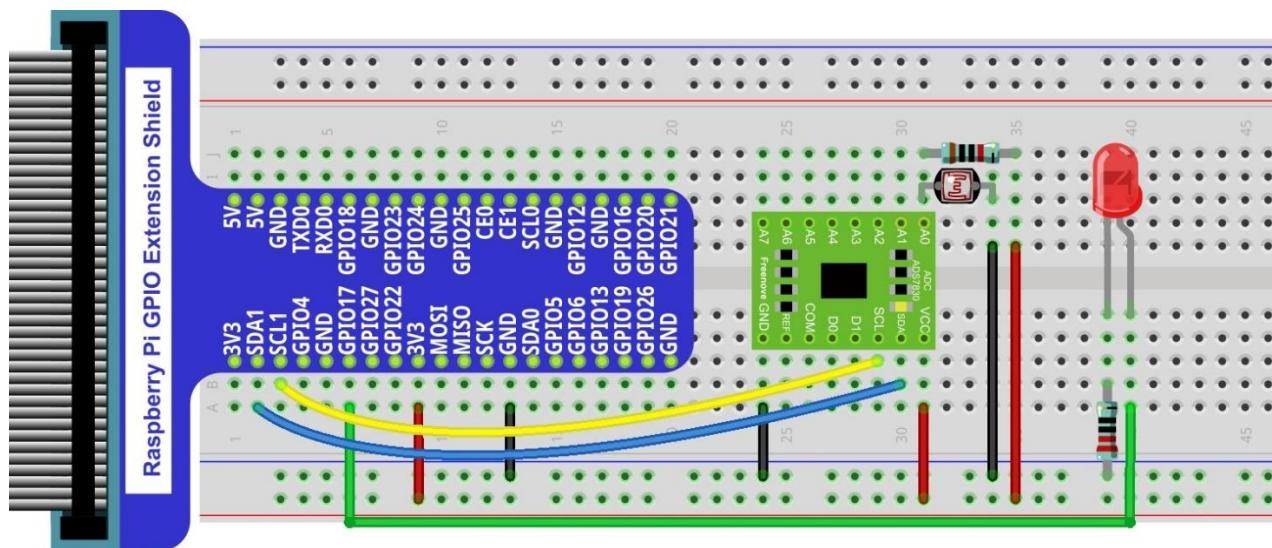
Circuit with ADS7830

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

Schematic diagram



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

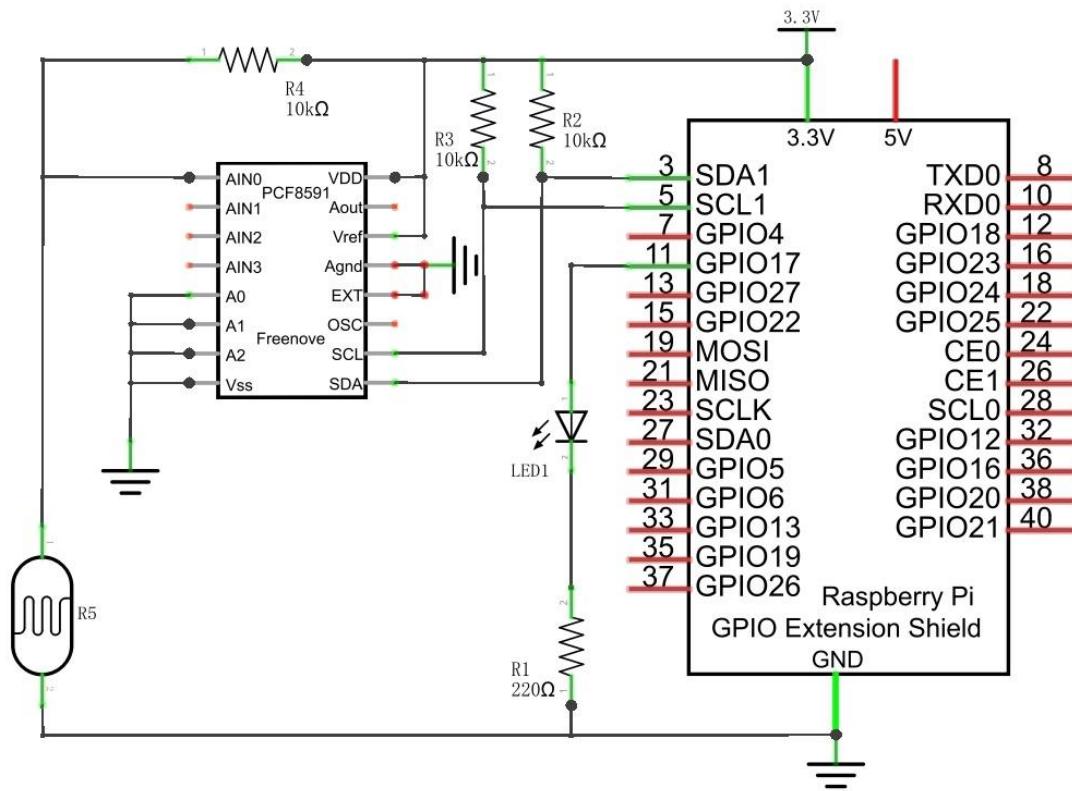


Video: <https://youtu.be/r6p3zhXsyko>

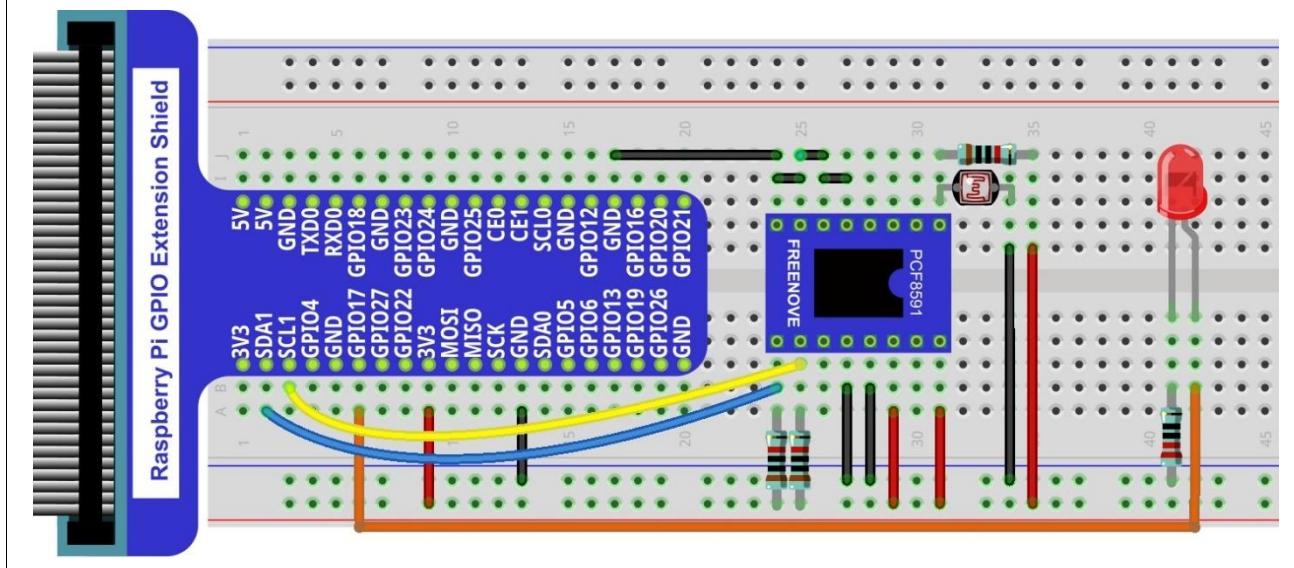
Circuit with PCF8591

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

Schematic diagram



Hardware connection



Code

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

C Code 10.1.1 Nightlamp

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

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

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

1. Use cd command to enter 10.1.1_Nightlamp directory of C code.

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

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

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

3. Then run the generated file "Nightlamp".

```
sudo ./Nightlamp
```

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

The following is the program code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define ledPin 0
7
8 ADCDevice *adc; // Define an ADC Device class object
9
10 int main(void) {
11     adc = new ADCDevice();
12     printf("Program is starting ... \n");
13
14     if(adc->detectI2C(0x48)){ // Detect the pcf8591.
15         delete adc;           // Free previously pointed memory
16         adc = new PCF8591(); // If detected, create an instance of PCF8591.
17     }
18     else if(adc->detectI2C(0x4b)){// Detect the ads7830
19         delete adc;           // Free previously pointed memory
20         adc = new ADS7830(); // If detected, create an instance of ADS7830.
21     }
22     else{
23         printf("No correct I2C address found, \n"
24             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
25             "Program Exit. \n");
26     }
27 }
```

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

Python Code 10.1.1 Nightlamp

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

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

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

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

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

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

```
python Nightlamp.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 from ADCDevice import *
4
5 ledPin = 11 # define ledPin
6 adc = ADCDevice() # Define an ADCDevice class object
7
8 def setup():
9     global adc
10    if(adc.detectI2C(0x48)): # Detect the pcf8591.
11        adc = PCF8591()
12    elif(adc.detectI2C(0x4b)): # Detect the ads7830
13        adc = ADS7830()
14    else:
15        print("No correct I2C address found, \n"
16              "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
17              "Program Exit. \n");
18        exit(-1)
19    global p
20    GPIO.setmode(GPIO.BOARD)
21    GPIO.setup(ledPin,GPIO.OUT)    # set ledPin to OUTPUT mode
22    GPIO.output(ledPin,GPIO.LOW)
23
24    p = GPIO.PWM(ledPin,1000) # set PWM Frenquenc to 1kHz
25    p.start(0)
26
27 def loop():
28    while True:
29        value = adc.analogRead(0)      # read the ADC value of channel 0
30        p.ChangeDutyCycle(value*100/255)
31        voltage = value / 255.0 * 3.3
```

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

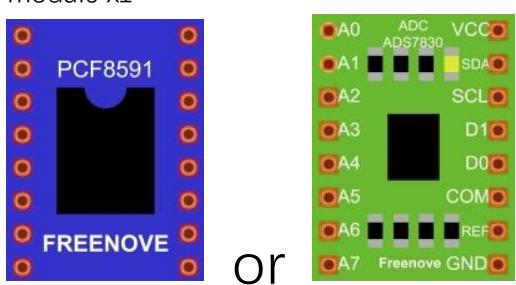
Chapter 11 Thermistor

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

Project 11.1 Thermometer

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

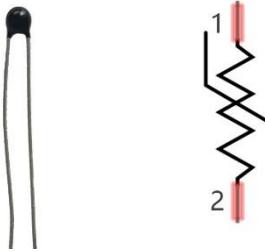
Component List

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

Component knowledge

Thermistor

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



The relationship between resistance value and temperature of a thermistor is:

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

Where:

R_t is the thermistor resistance under T₂ temperature;

R is the nominal resistance of thermistor under T₁ temperature;

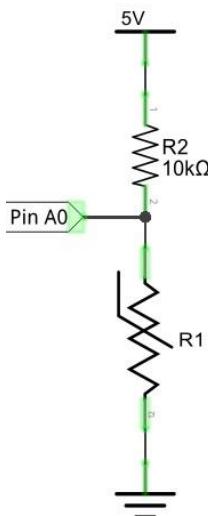
EXP[n] is nth power of e;

B is for thermal index;

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

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

The circuit connection method of the Thermistor is similar to photoresistor, as the following:



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

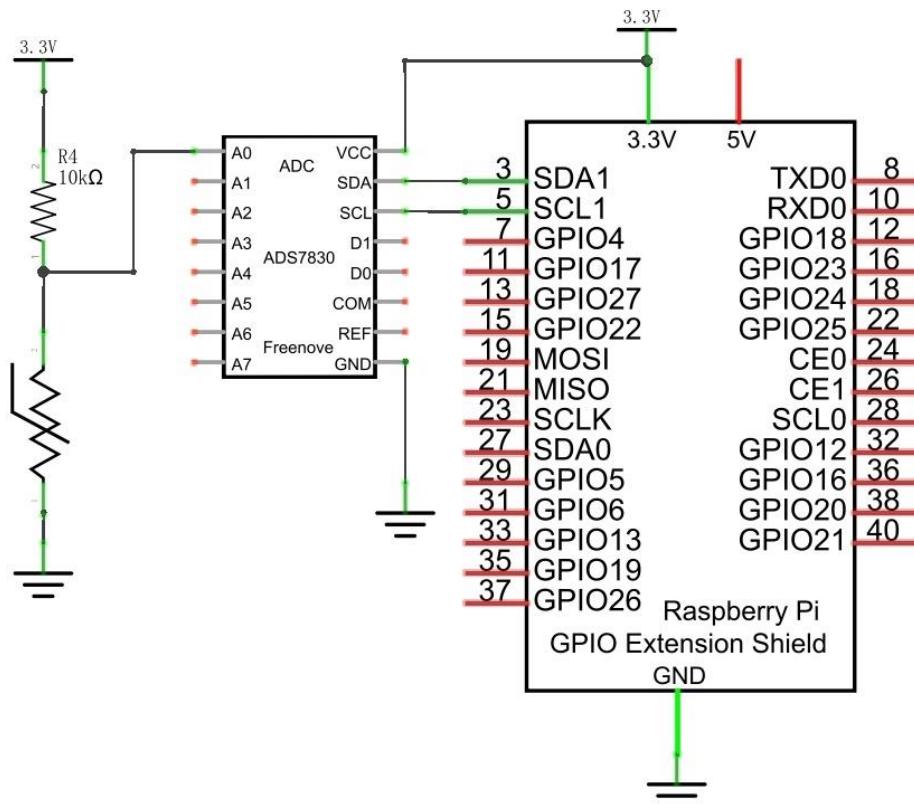
Therefore, the temperature formula can be derived as:

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

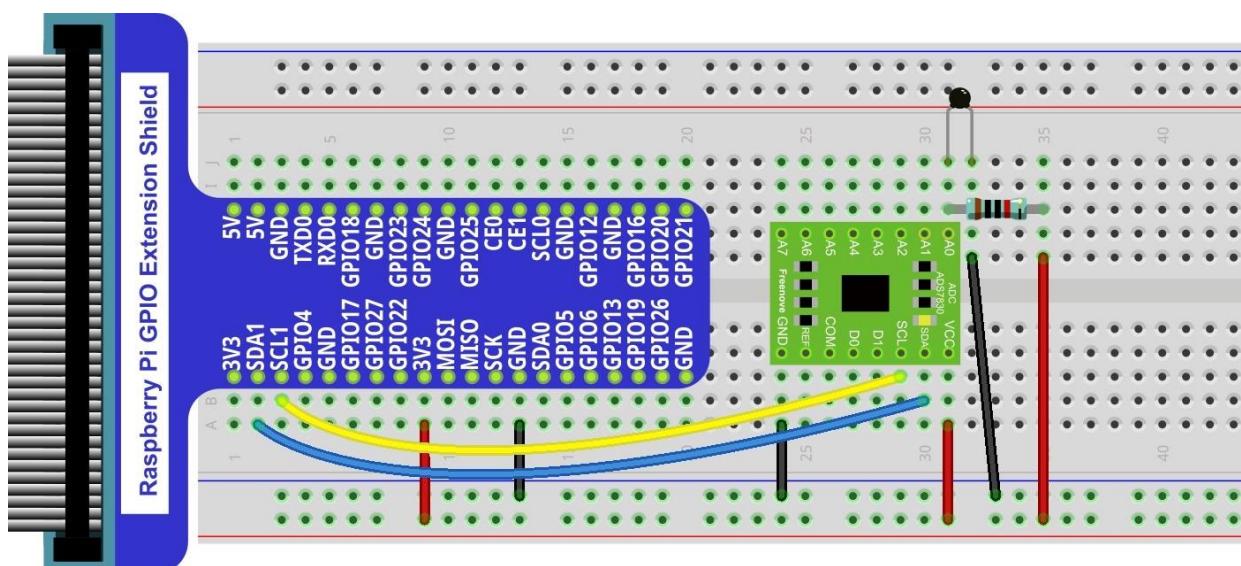
Circuit with ADS7830

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

Schematic diagram



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



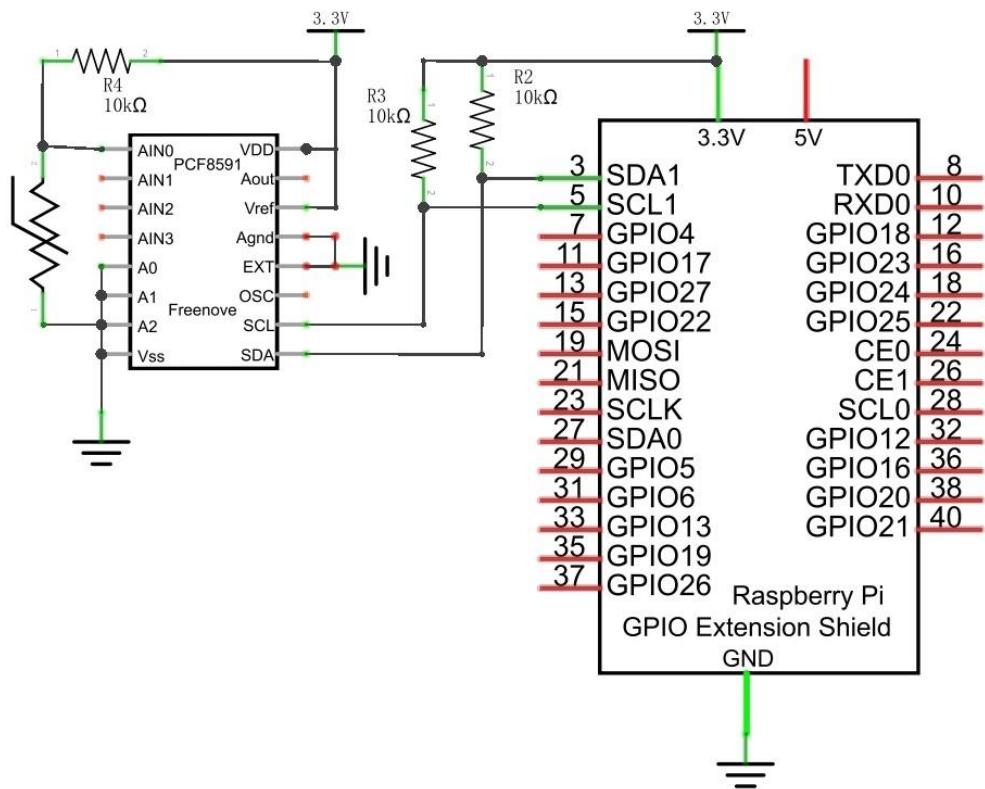
Thermistor has **longer pins** than the one shown in circuit.

Video: <https://youtu.be/spOalxanNMc>

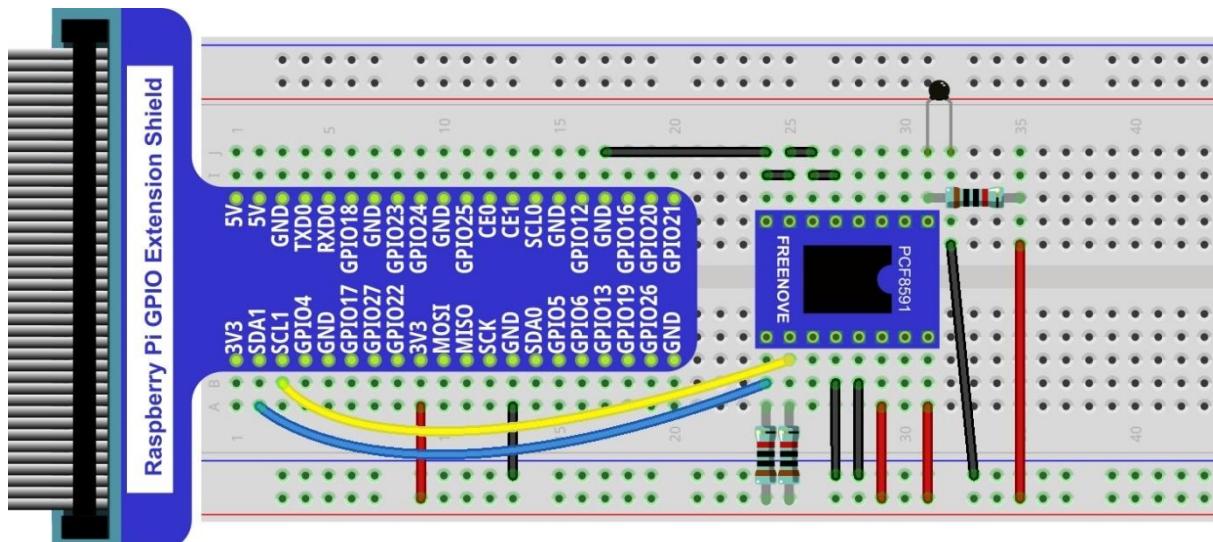
Circuit with PCF8591

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

Schematic diagram



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



Thermistor has longer pins than the one shown in circuit.

Code

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

C Code 11.1.1 Thermometer

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

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

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

1. Use cd command to enter 11.1.1_Termometer directory of C code.

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

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

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

3 Then run the generated file "Thermometer".

```
sudo ./Thermometer
```

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

```
ADC value : 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
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:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <math.h>
4 #include <ADCDevice.hpp>
5
6 ADCDevice *adc; // Define an ADC Device class object
7
8 int main(void) {
9     adc = new ADCDevice();
10    printf("Program is starting ... \n");
```

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

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

Python Code 11.1.1 Thermometer

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

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

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

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

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

```
python Thermometer.py
```

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

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

The following is the code:

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

```
19
20 def loop():
21     while True:
22         value = adc.analogRead(0)          # read ADC value A0 pin
23         voltage = value / 255.0 * 3.3      # calculate voltage
24         Rt = 10 * voltage / (3.3 - voltage)    # calculate resistance value of thermistor
25         tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) # calculate temperature
26         (Kelvin)
27         tempC = tempK -273.15           # calculate temperature (Celsius)
28         print ('ADC Value : %d, Voltage : %.2f,
29 Temperature : %.2f' %(value, voltage, tempC))
30         time.sleep(0.01)
31
32 def destroy():
33     adc.close()
34     GPIO.cleanup()
35
36 if __name__ == '__main__': # Program entrance
37     print ('Program is starting ... ')
38     setup()
39     try:
40         loop()
41     except KeyboardInterrupt: # Press ctrl-c to end the program.
42         destroy()
```

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

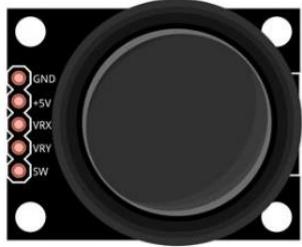
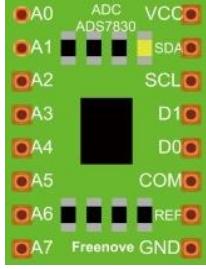
Chapter 12 Joystick

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

Project 12.1 Joystick

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

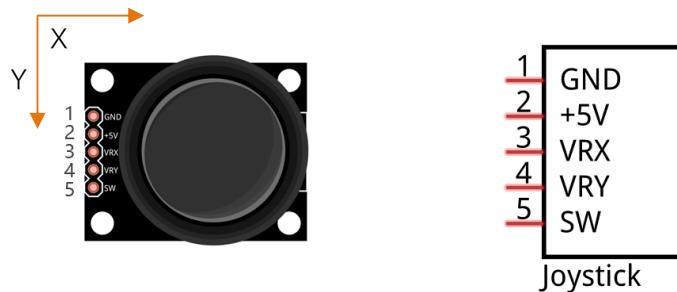
Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper x18 
Joystick x1 	ADC module x1 PCF8591  or 

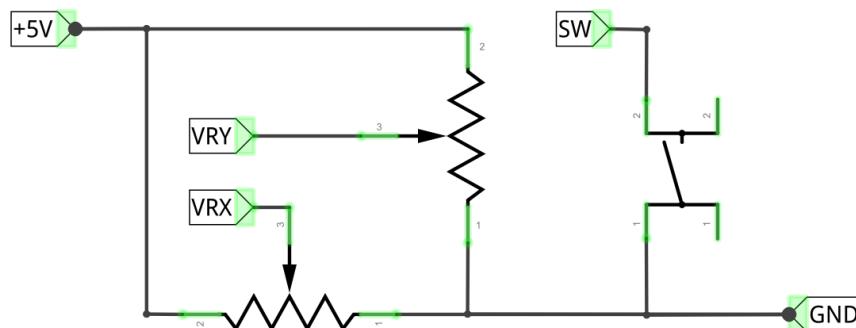
Component knowledge

Joystick

A Joystick is a kind of input sensor used with your fingers. You should be familiar with this concept already as they are widely used in gamepads and remote controls. It can receive input on two axes (Y and or X) at the same time (usually used to control direction on a two dimensional plane). And it also has a third direction capability by **pressing down (Z axis/direction)**.



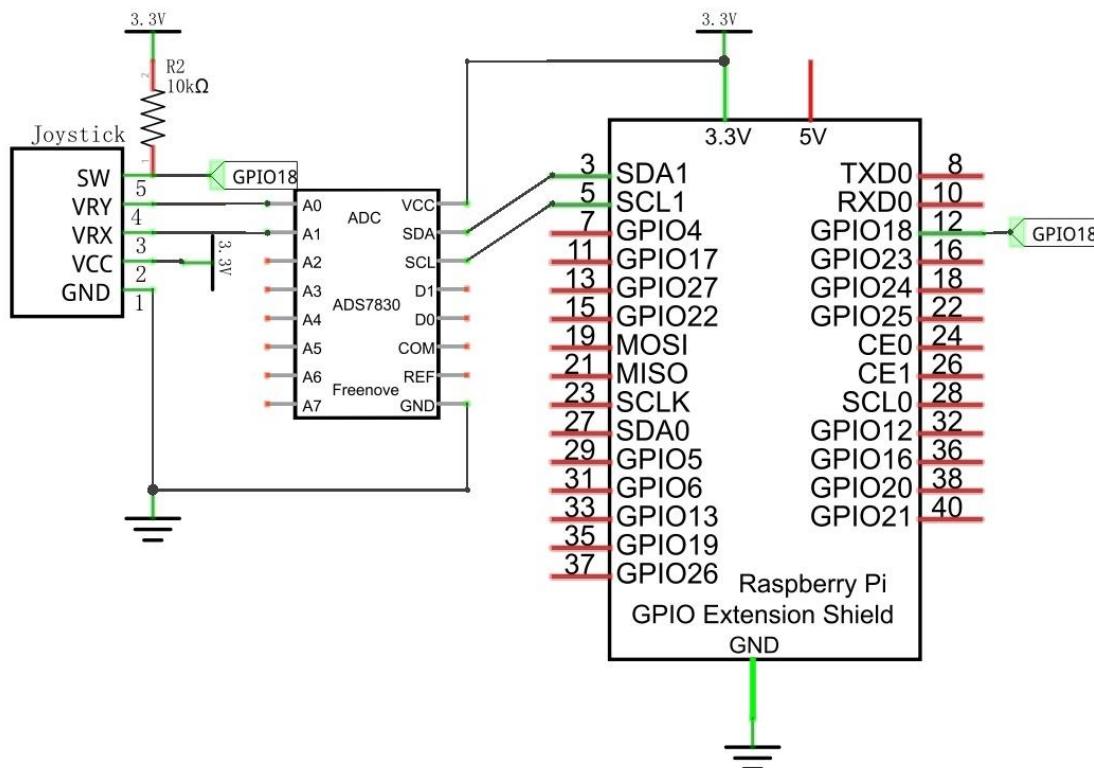
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses on the Joystick.



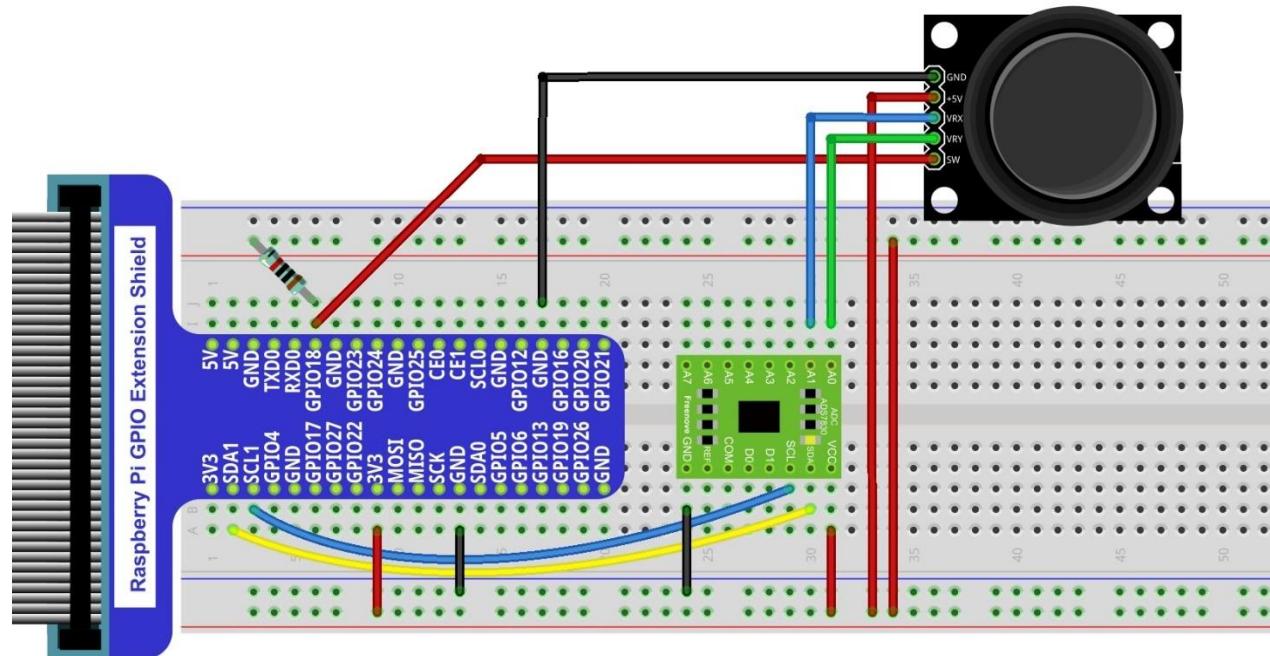
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit with ADS7830

Schematic diagram



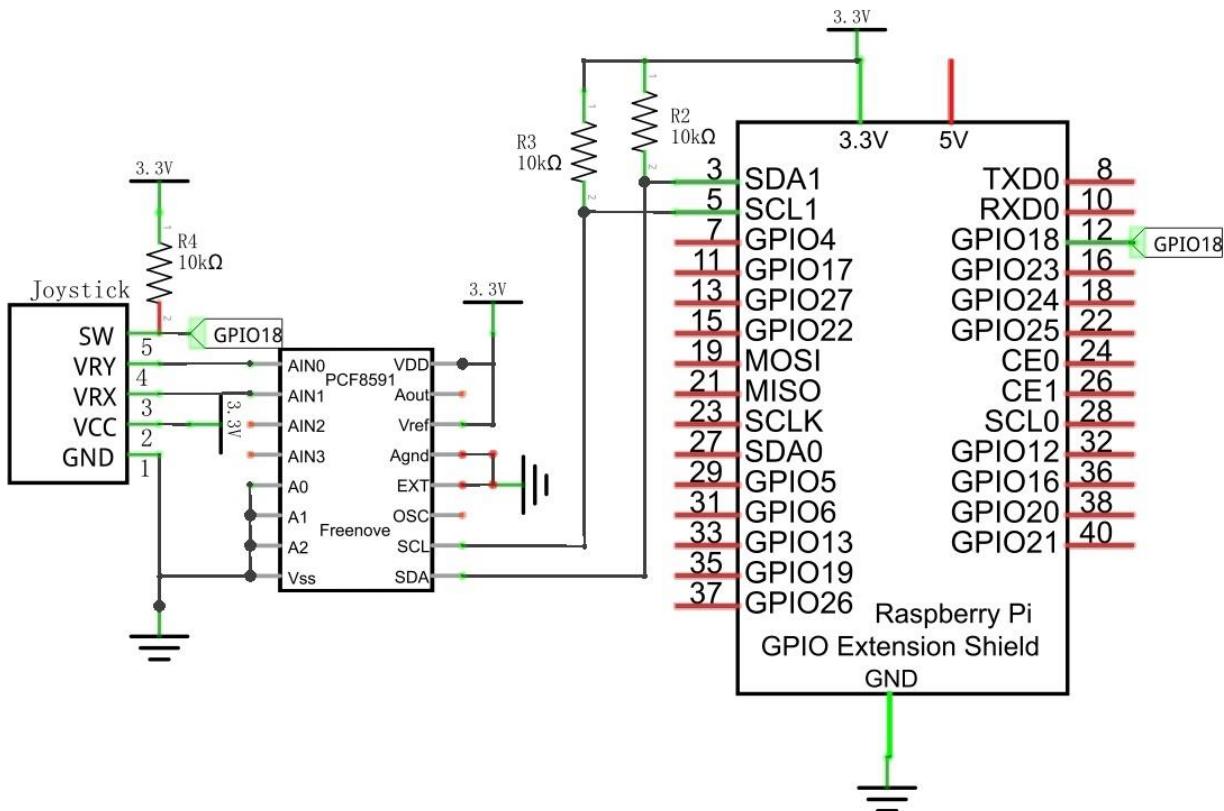
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



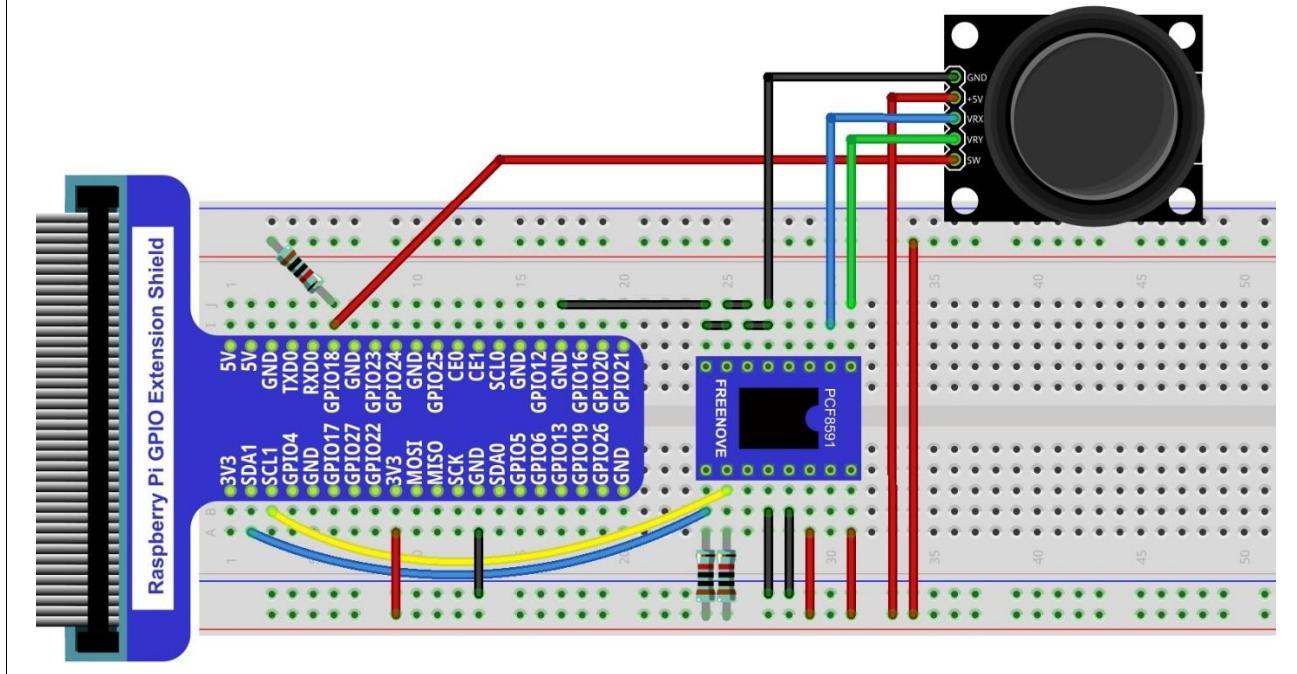
Video: <https://youtu.be/qjP3HpbPJTM>

Circuit with PCF8591

Schematic diagram



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



Code

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

C Code 12.1.1 Joystick

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

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

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

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

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

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

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

3. Then run the generated file "Joystick".

```
sudo ./Joystick
```

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

```
val_X: 128 , val_Y: 135 , val_Z: 1
val_X: 128 , val_Y: 155 , val_Z: 1
val_X: 255 , val_Y: 255 , val_Z: 1
val_X: 181 , val_Y: 255 , val_Z: 1
val_X: 128 , val_Y: 255 , val_Z: 1
val_X: 128 , val_Y: 180 , val_Z: 0
val_X: 128 , val_Y: 138 , val_Z: 0
val_X: 128 , val_Y: 137 , val_Z: 0
val_X: 128 , val_Y: 139 , val_Z: 0
val_X: 128 , val_Y: 139 , val_Z: 1
```

The flowing is the code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <softPwm.h>
4 #include <ADCDevice.hpp>
5
6 #define Z_Pin 1      //define pin for axis Z
7
8 ADCDevice *adc; // Define an ADC Device class object
9
10 int main(void) {
11     adc = new ADCDevice();
12     printf("Program is starting ... \n");
13 }
```

```

14     if(adc->detectI2C(0x48)){    // Detect the pcf8591.
15         delete adc;           // Free previously pointed memory
16         adc = new PCF8591();   // If detected, create an instance of PCF8591.
17     }
18     else if(adc->detectI2C(0x4b)){// Detect the ads7830
19         delete adc;           // Free previously pointed memory
20         adc = new ADS7830();   // If detected, create an instance of ADS7830.
21     }
22     else{
23         printf("No correct I2C address found, \n"
24             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
25             "Program Exit. \n");
26         return -1;
27     }
28     wiringPiSetup();
29     pinMode(Z_Pin, INPUT);      //set Z_Pin as input pin and pull-up mode
30     pullUpDnControl(Z_Pin, PUD_UP);
31     while(1){
32         int val_Z = digitalRead(Z_Pin); //read digital value of axis Z
33         int val_Y = adc->analogRead(0); //read analog value of axis X and Y
34         int val_X = adc->analogRead(1);
35         printf("val_X: %d ,\tval_Y: %d ,\tval_Z: %d \n", val_X, val_Y, val_Z);
36         delay(100);
37     }
38     return 0;
39 }
```

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

```

while(1){
    int val_Z = digitalRead(Z_Pin); //read digital value of axis Z
    int val_Y = adc->analogRead(0); //read analog value of axis X and Y
    int val_X = adc->analogRead(1);
    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

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

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

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

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

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

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

```
python Joystick.py
```

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

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3 from ADCDevice import *
4
5 Z_Pin = 12      # define Z_Pin
6 adc = ADCDevice() # Define an ADCDevice class object
7
8 def setup():
9     global adc
10    if(adc.detectI2C(0x48)): # Detect the pcf8591.
11        adc = PCF8591()
12    elif(adc.detectI2C(0x4b)): # Detect the ads7830
13        adc = ADS7830()
14    else:
15        print("No correct I2C address found, \n"
16              "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
17              "Program Exit. \n");
18        exit(-1)
19    GPIO.setmode(GPIO.BCM)
20    GPIO.setup(Z_Pin,GPIO.IN,GPIO.PUD_UP)    # set Z_Pin to pull-up mode
21
22 def loop():
23     while True:
24         val_Z = GPIO.input(Z_Pin)          # read digital value of axis Z
25         val_Y = adc.analogRead(0)          # read analog value of axis X and Y
26         val_X = adc.analogRead(1)
```

```
26     print (' value_X: %d ,\tvalue_Y: %d ,\tvalue_Z: %d' %(val_X, val_Y, val_Z))
27     time.sleep(0.01)
28
29 def destroy():
30     adc.close()
31     GPIO.cleanup()
32
33 if __name__ == '__main__':
34     print (' Program is starting ... ') # Program entrance
35     setup()
36     try:
37         loop()
38     except KeyboardInterrupt: # Press ctrl-c to end the program.
39         destroy()
```

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

```
while True:
    val_Z = GPIO.input(Z_Pin)      #read digital quality of axis Z
    val_Y = analogRead(0)          #read analog quality of axis X and Y
    val_X = analogRead(1)
    print (' value_X: %d ,\tvalue_Y: %d ,\tvalue_Z: %d' %(val_X, val_Y, val_Z))
    time.sleep(0.01)
```

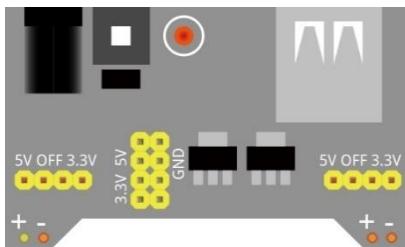
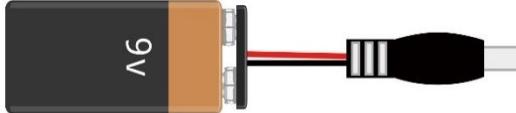
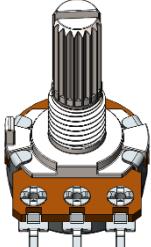
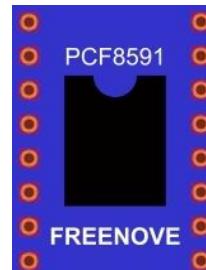
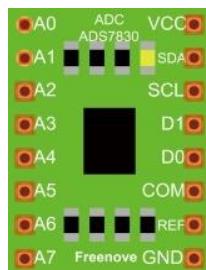
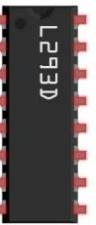
Chapter 13 Motor & Driver

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

Project 13.1 Control a DC Motor with a Potentiometer

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

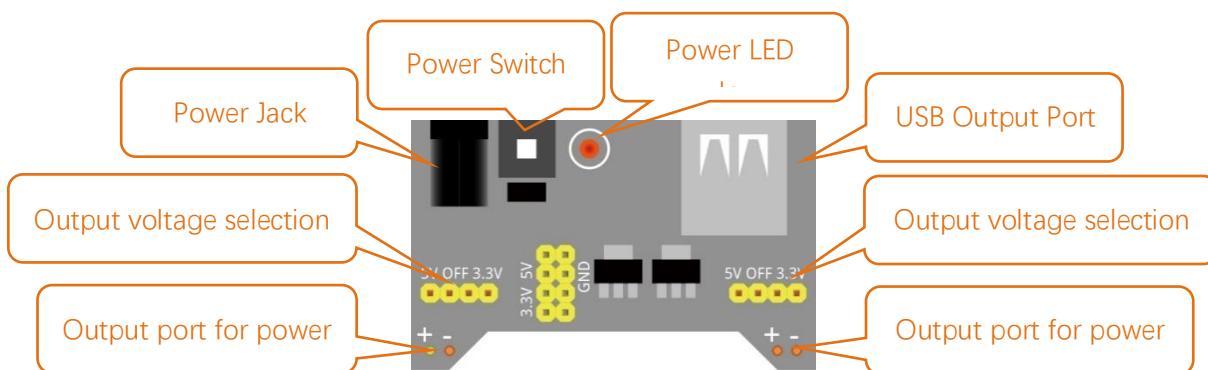
Component List

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

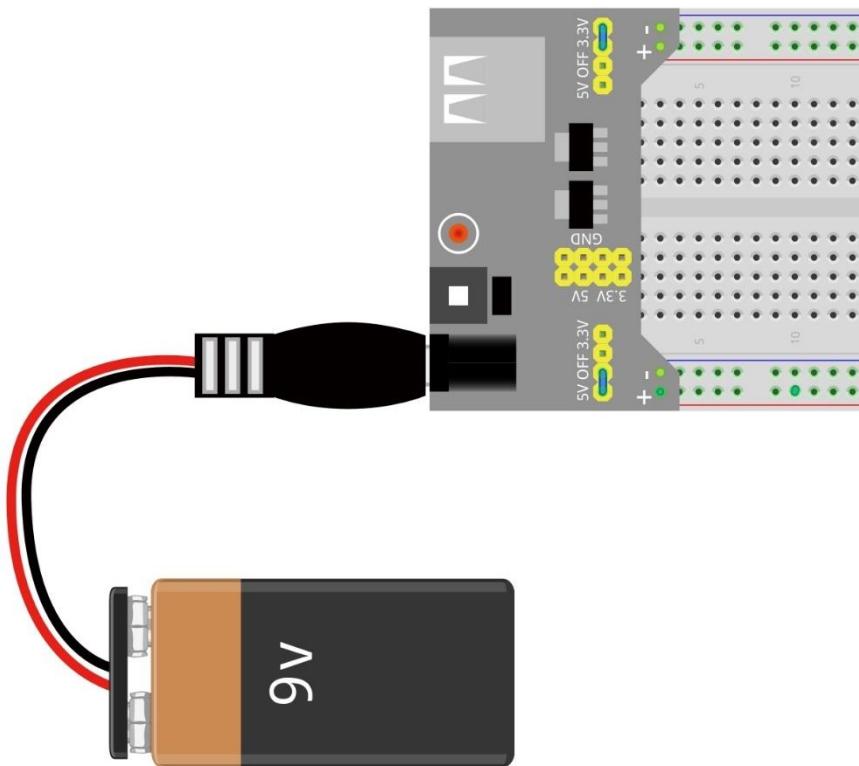
Component knowledge

Breadboard Power Module

Breadboard Power Module is an independent circuit board, which can provide independent 5V or 3.3V power to the breadboard when building circuits. It also has built-in power protection to avoid damaging your RPi module. The schematic diagram below identifies the important features of this Power Module:

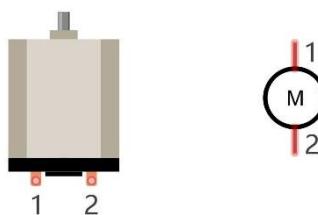


Here is an acceptable connection between Breadboard Power Module and Breadboard using a 9V battery and the provided power harness:



DC Motor

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

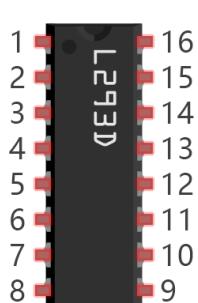


When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.



L293D

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



1	Enable 1	+V	16
2	In 1	In 4	15
3	Out 1	Out 4	14
4	0V	0V	13
5	0V	0V	12
6	Out 2	Out 3	11
7	In 2	In 3	10
8	+Vmotor	Enable 2	9

L293D



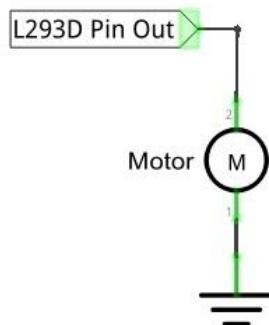
Port description of L293D module is as follows:

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

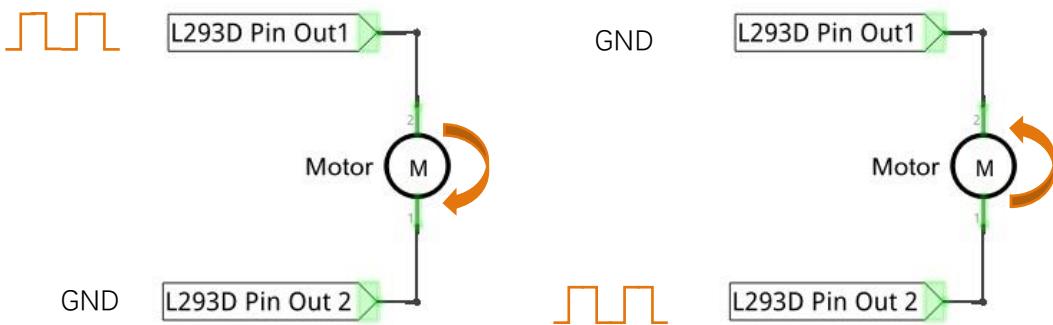
For more details, please see the datasheet for this IC Chip.

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

The following connection option uses one channel of the L293D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.



The following connection uses two channels of the L293D: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, not only controls the speed of motor, but also can control the direction of the motor.

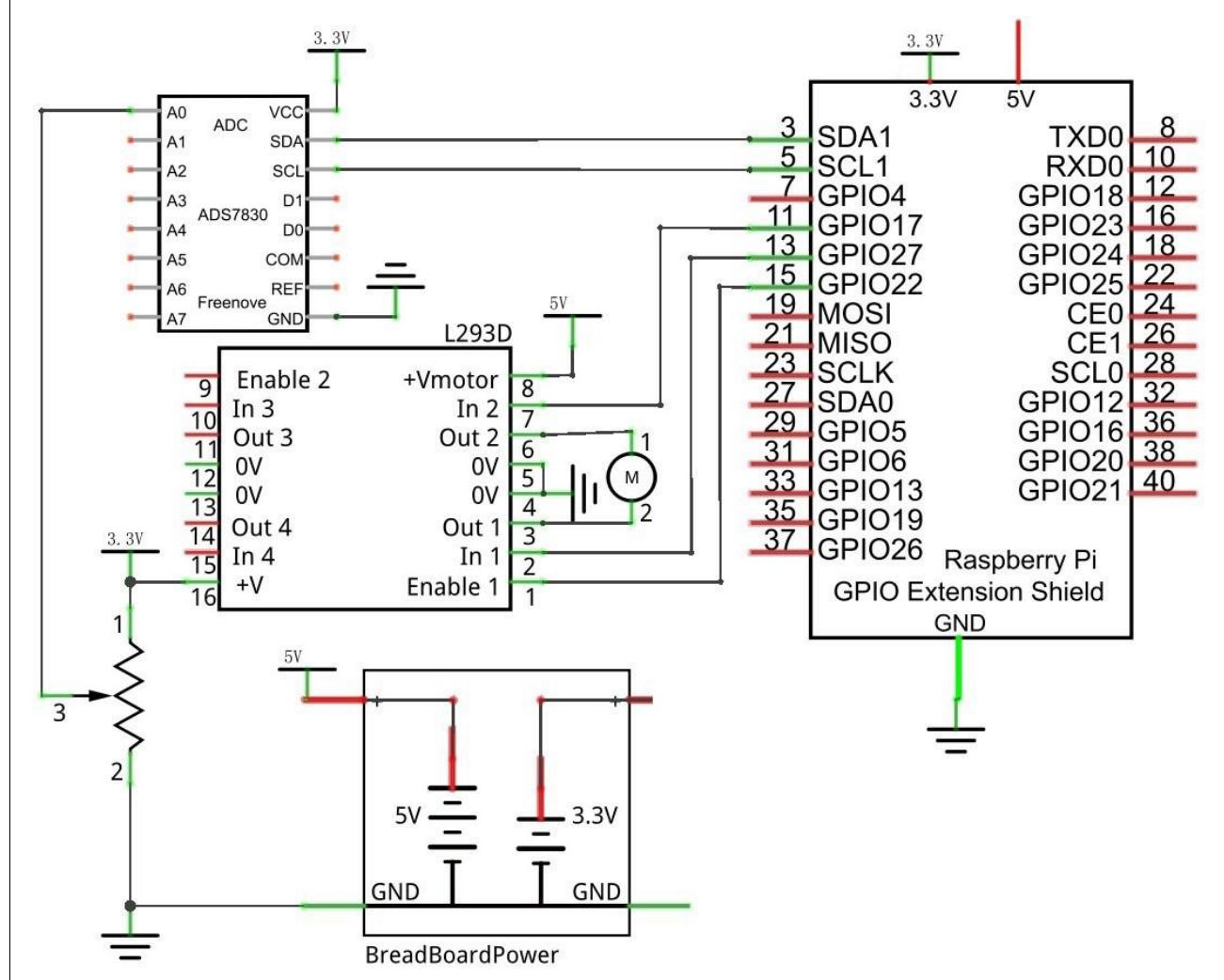


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

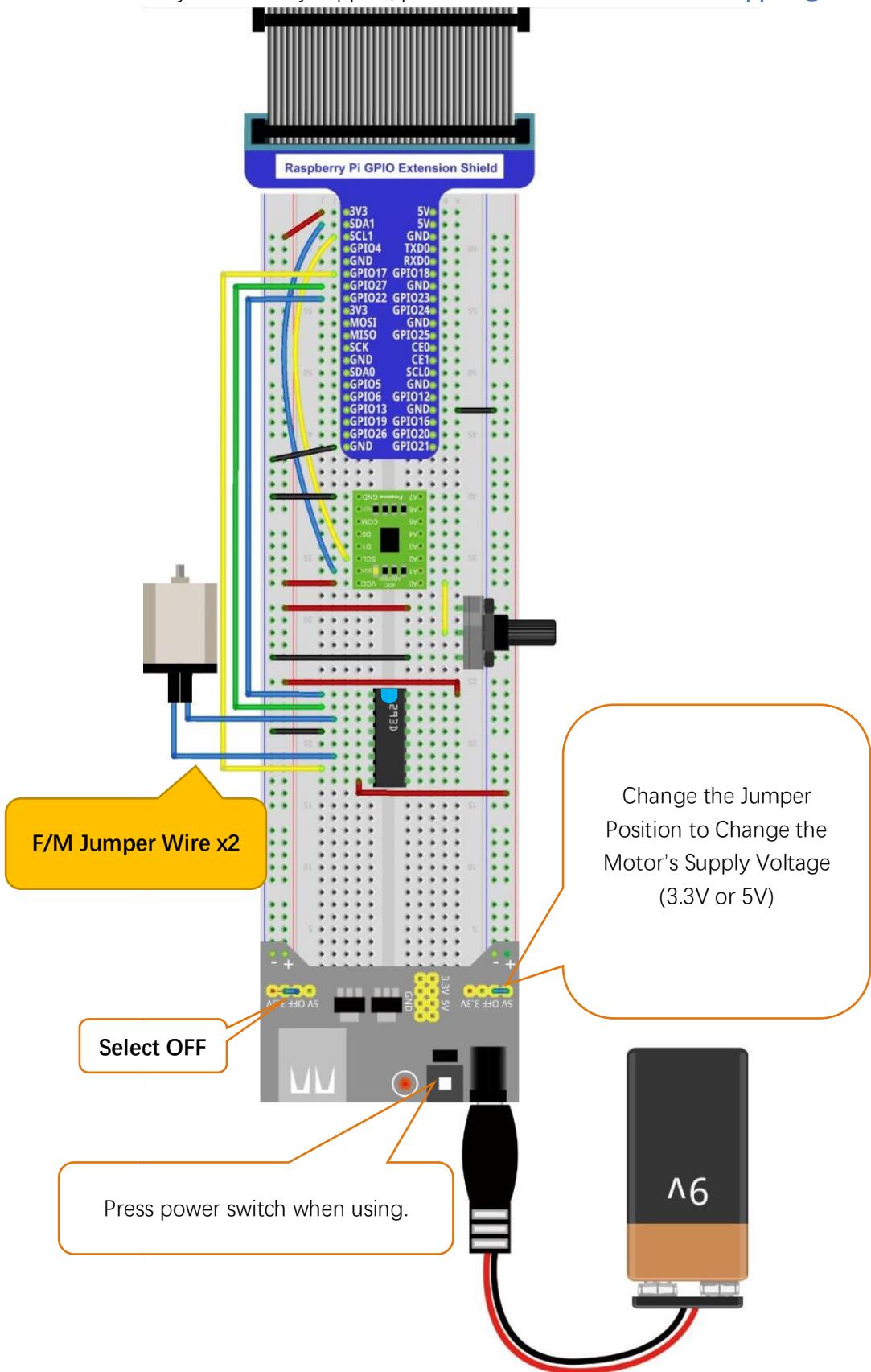
Circuit with ADS7830

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

Schematic diagram



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

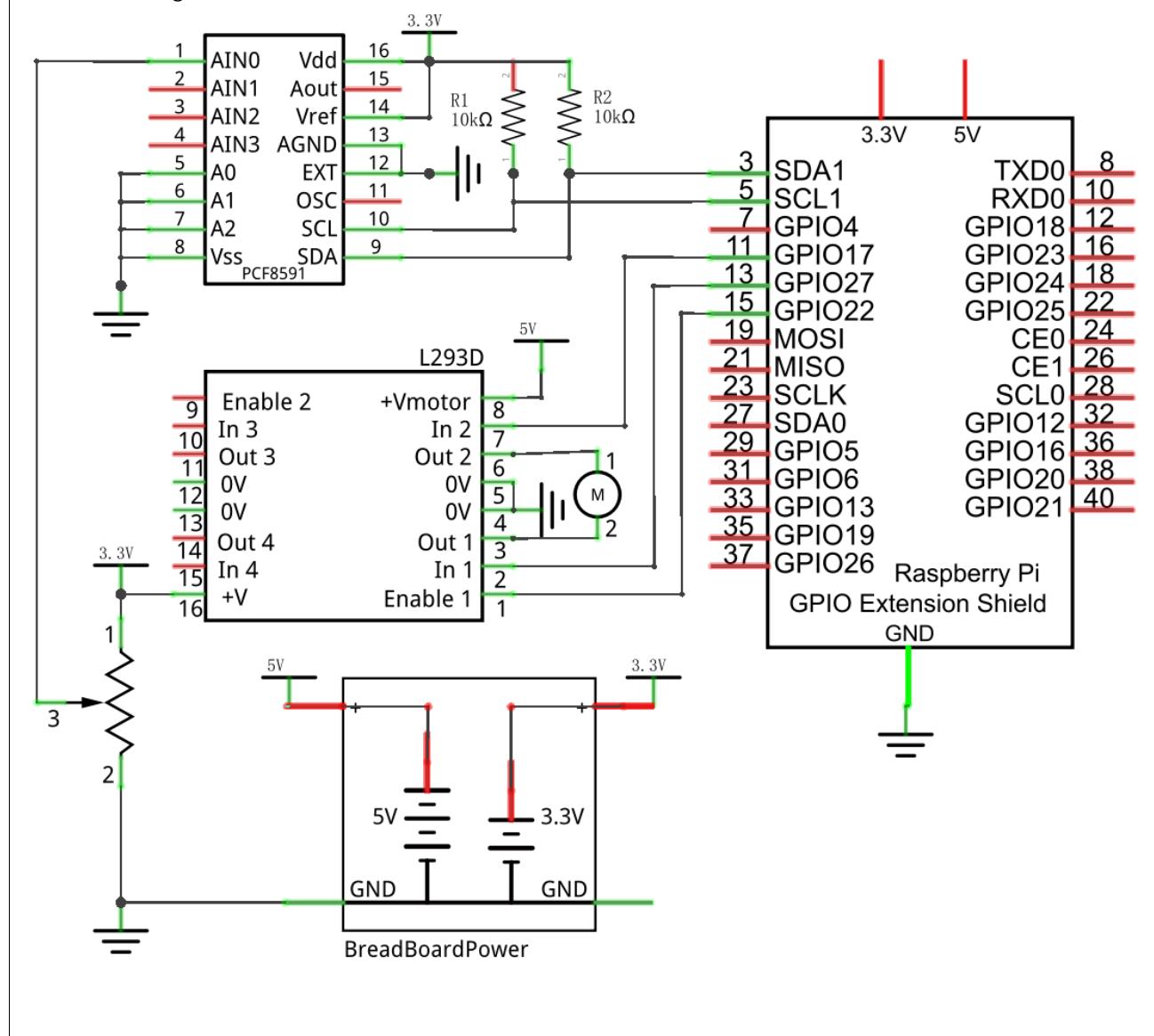


Video: <https://youtu.be/d5IRMTDK-wg>

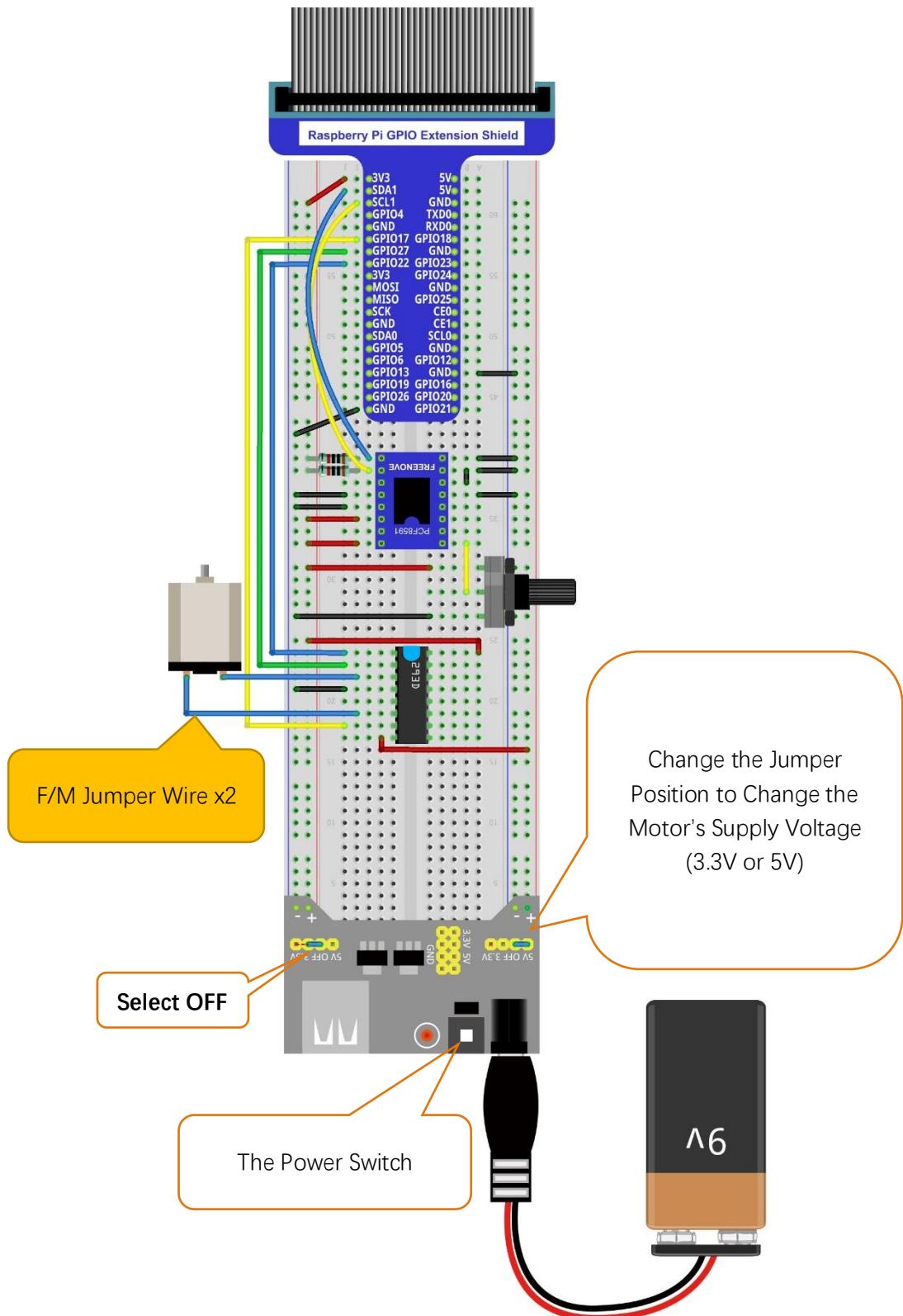
Circuit with PCF8591

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

Schematic diagram



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



Code

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

C Code 13.1.1 Motor

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

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

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

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

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

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

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

3. Then run the generated file "Motor".

```
sudo ./Motor
```

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

```
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 <stdio.h>
3 #include <softPwm.h>
4 #include <math.h>
5 #include <stdlib.h>
6 #include <ADCDevice.hpp>
7
```

```
8 #define motorPin1    2          //define the pin connected to L293D
9 #define motorPin2    0
10#define enablePin     3
11
12ADCDevice *adc; // Define an ADC Device class object
13
14//Map function: map the value from a range to another range.
15long map(long value, long fromLow, long fromHigh, long toLow, long toHigh) {
16    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
17}
18//motor function: determine the direction and speed of the motor according to the ADC
19void motor(int ADC) {
20    int value = ADC -128;
21    if(value>0) {
22        digitalWrite(motorPin1, HIGH);
23        digitalWrite(motorPin2, LOW);
24        printf("turn Forward... \n");
25    }
26    else if (value<0) {
27        digitalWrite(motorPin1, LOW);
28        digitalWrite(motorPin2, HIGH);
29        printf("turn Back... \n");
30    }
31    else {
32        digitalWrite(motorPin1, LOW);
33        digitalWrite(motorPin2, LOW);
34        printf("Motor Stop... \n");
35    }
36    softPwmWrite(enablePin, map(abs(value), 0, 128, 0, 100));
37    printf("The PWM duty cycle is %d%\n", abs(value)*100/127); //print the PWM duty cycle
38}
39int main(void) {
40    adc = new ADCDevice();
41    printf("Program is starting ... \n");
42
43    if(adc->detectI2C(0x48)){ // Detect the pcf8591.
44        delete adc;           // Free previously pointed memory
45        adc = new PCF8591(); // If detected, create an instance of PCF8591.
46    }
47    else if(adc->detectI2C(0x4b)){// Detect the ads7830
48        delete adc;           // Free previously pointed memory
49        adc = new ADS7830(); // If detected, create an instance of ADS7830.
50    }
51    else{
```

```

52     printf("No correct I2C address found, \n"
53     "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
54     "Program Exit. \n");
55     return -1;
56 }
57 wiringPiSetup();
58 pinMode(enablePin, OUTPUT); //set mode for the pin
59 pinMode(motorPin1, OUTPUT);
60 pinMode(motorPin2, OUTPUT);
61 softPwmCreate(enablePin, 0, 100); //define PWM pin
62 while(1){
63     int value = adc->analogRead(0); //read analog value of A0 pin
64     printf("ADC value : %d \n", value);
65     motor(value); //make the motor rotate with speed(analog value of A0 pin)
66     delay(100);
67 }
68 return 0;
69 }
```

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

```

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, 100));
    printf("The PWM duty cycle is %d%%\n", abs(value)*100/127); // print out PWM duty
cycle.
}
```

Python Code 13.1.1 Motor

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

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

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

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

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

```
python Motor.py
```

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

```
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 time
3 from ADCDevice import *
4
5 # define the pins connected to L293D
6 motoRPin1 = 13
7 motoRPin2 = 11
8 enablePin = 15
9 adc = ADCDevice() # Define an ADCDevice class object
10
11 def setup():
12     global adc
13     if(adc.detectI2C(0x48)): # Detect the pcf8591.
14         adc = PCF8591()
```

```
15 elif(adc.detectI2C(0x4b)): # Detect the ads7830
16     adc = ADS7830()
17 else:
18     print("No correct I2C address found, \n"
19         "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
20         "Program Exit. \n");
21     exit(-1)
22 global p
23 GPIO.setmode(GPIO.BCM)
24 GPIO.setup(motorRPin1,GPIO.OUT)    # set pins to OUTPUT mode
25 GPIO.setup(motorRPin2,GPIO.OUT)
26 GPIO.setup(enablePin,GPIO.OUT)
27
28 p = GPIO.PWM(enablePin,1000) # creat PWM and set Frequency to 1KHz
29 p.start(0)
30
31 # mapNUM function: map the value from a range of mapping to another range.
32 def mapNUM(value,fromLow,fromHigh,toLow,toHigh):
33     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
34
35 # motor function: determine the direction and speed of the motor according to the input
36 ADC value input
37 def motor(ADC):
38     value = ADC -128
39     if (value > 0): # make motor turn forward
40         GPIO.output(motorRPin1,GPIO.HIGH) # motorRPin1 output HIGH level
41         GPIO.output(motorRPin2,GPIO.LOW) # motorRPin2 output LOW level
42         print (' Turn Forward... ')
43     elif (value < 0): # make motor turn backward
44         GPIO.output(motorRPin1,GPIO.LOW)
45         GPIO.output(motorRPin2,GPIO.HIGH)
46         print (' Turn Backward... ')
47     else :
48         GPIO.output(motorRPin1,GPIO.LOW)
49         GPIO.output(motorRPin2,GPIO.LOW)
50         print (' Motor Stop... ')
51     p.start(mapNUM(abs(value),0,128,0,100))
52     print (' The PWM duty cycle is %d%%\n' %(abs(value)*100/127)) # print PMW duty cycle.
53
54 def loop():
55     while True:
56         value = adc.analogRead(0) # read ADC value of channel 0
57         print (' ADC Value : %d' %(value))
58         motor(value)
```

```

59         time.sleep(0.01)

60
61     def destroy():
62         GPIO.cleanup()

63
64     if __name__ == '__main__': # Program entrance
65         print (' Program is starting ... ')
66         setup()
67         try:
68             loop()
69         except KeyboardInterrupt: # Press ctrl-c to end the program.
70             destroy()

```

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

```

def motor(ADC):
    value = ADC -128
    if (value > 0):
        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 PWM duty cycle.

```

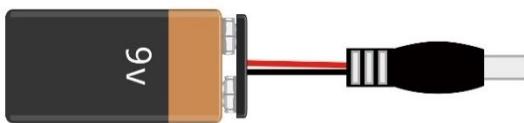
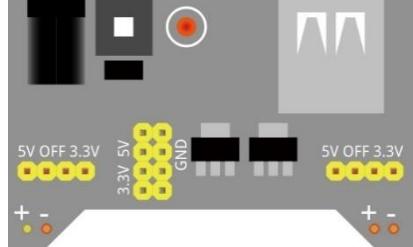
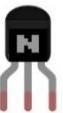
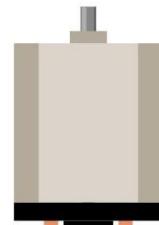
Chapter 14 Relay & Motor

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

Project 14.1.1 Relay & Motor

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

Component List

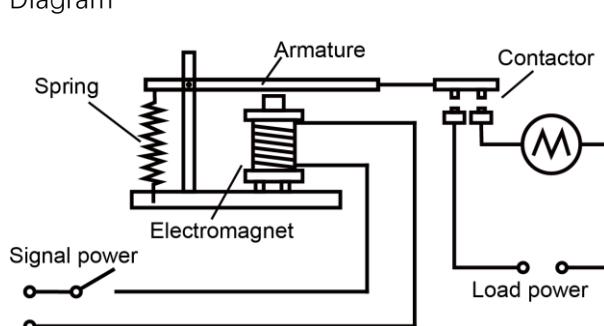
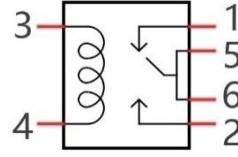
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x11 
9V battery (prepared by yourself) & battery line 	
Breadboard Power module x1 	Resistor 10kΩ x2 
Resistor 1kΩ x1 	Resistor 220Ω x1 
NPN transistor x1 	Relay x1 
Motor x1 	Push button x1 
LED x1 	Diode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

Component knowledge

Relay

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

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

Diagram	Feature:	Symbol
		

Pin 5 and pin 6 are internally connected to each other. When the coil pin 3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

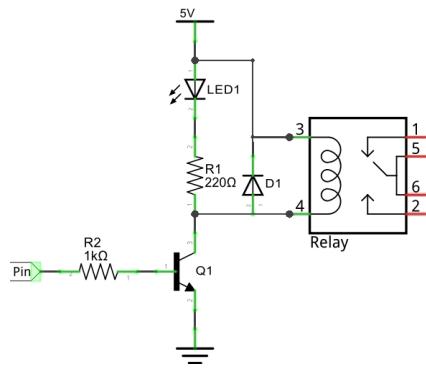
Inductor

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

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

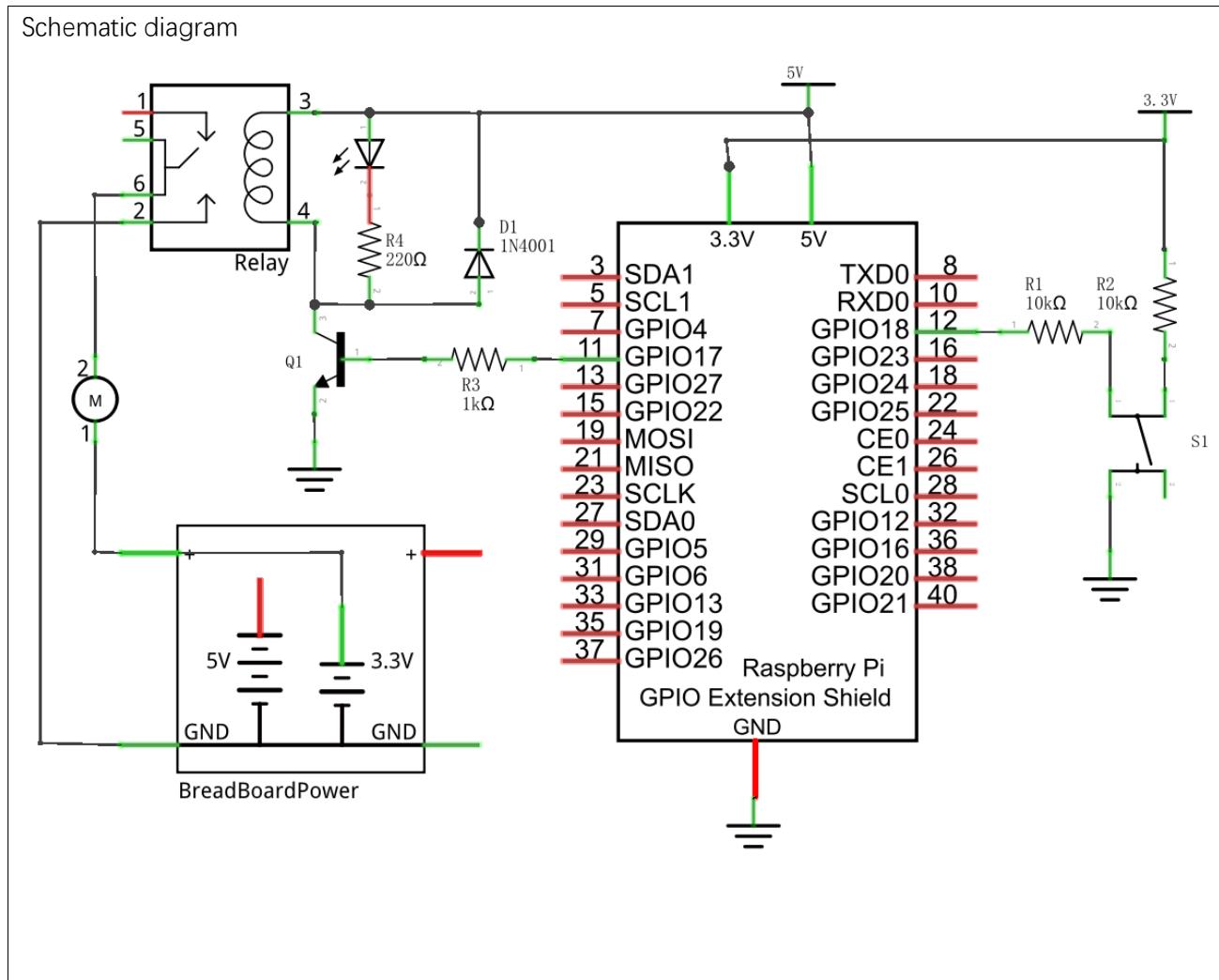


The circuit for a Relay is as follows: The coil of Relay can be equivalent to an Inductor, when a Transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes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

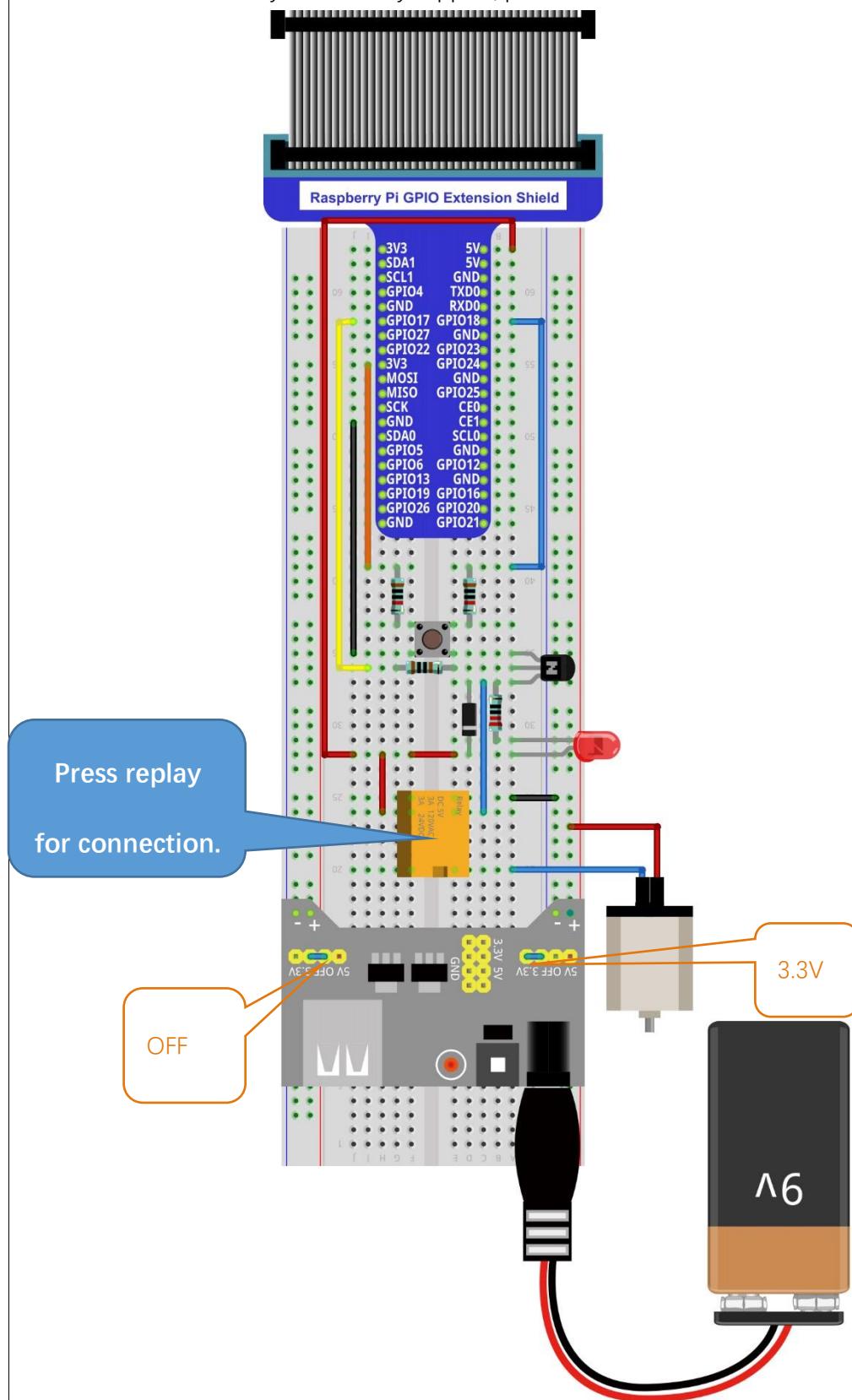


Circuit

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



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



Video: <https://youtu.be/CUpPpWq8YI8>

Code

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

C Code 14.1.1 Relay

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

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

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

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

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3
4 #define relayPin 0 //define the relayPin
5 #define buttonPin 1 //define the buttonPin
6 int relayState=LOW; //store the State of relay
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     printf("Program is starting...\n");
15
16     wiringPiSetup();
17
18     pinMode(relayPin, OUTPUT);
19     pinMode(buttonPin, INPUT);
20     pullUpDnControl(buttonPin, PUD_UP); //pull up to high level
21     while(1) {
22         reading = digitalRead(buttonPin); //read the current state of button
23         if( reading != lastbuttonState){ //if the button state changed ,record the time
24             point

```

```
25         lastChangeTime = millis();
26     }
27     //if changing-state of the button last beyond the time we set, we considered that
28     //the current button state is an effective change rather than a buffeting
29     if(millis() - lastChangeTime > captureTime){
30         //if button state is changed, update the data.
31         if(reading != buttonState){
32             buttonState = reading;
33             //if the state is low, the action is pressing.
34             if(buttonState == LOW){
35                 printf("Button is pressed!\n");
36                 relayState = !relayState;
37                 if(relayState){
38                     printf("turn on relay ... \n");
39                 }
40                 else {
41                     printf("turn off relay ... \n");
42                 }
43             }
44             //if the state is high, the action is releasing.
45             else {
46                 printf("Button is released!\n");
47             }
48         }
49     }
50     digitalWrite(relayPin, relayState);
51     lastbuttonState = reading;
52 }
53
54     return 0;
55 }
```

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

Python Code 14.1.1 Relay

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

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

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

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

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

```
python Relay.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3
4 relayPin = 11      # define the relayPin
5 buttonPin = 12     # define the buttonPin
6 debounceTime = 50
7
8 def setup():
9     GPIO.setmode(GPIO.BCM)
10    GPIO.setup(relayPin, GPIO.OUT)   # set relayPin to OUTPUT mode
11    GPIO.setup(buttonPin, GPIO.IN)  # set buttonPin to INPUT mode
12
13 def loop():
14     relayState = False
15     lastChangeTime = round(time.time()*1000)
16     buttonState = GPIO.HIGH
17     lastButtonState = GPIO.HIGH
18     reading = GPIO.HIGH
19     while True:
20         reading = GPIO.input(buttonPin)
21         if reading != lastButtonState :
22             lastChangeTime = round(time.time()*1000)
23             if ((round(time.time()*1000) - lastChangeTime) > debounceTime):
24                 if reading != buttonState :
25                     buttonState = reading;
26                     if buttonState == GPIO.LOW:
27                         print("Button is pressed!")
28                         relayState = not relayState
29                         if relayState:
30                             print("Turn on relay ...")
31                         else :
```

```
32             print("Turn off relay ... ")
33     else :
34         print("Button is released!")
35     GPIO.output(relayPin, relayState)
36     lastButtonState = reading # lastButtonState store latest state
37
38 def destroy():
39     GPIO.cleanup()
40
41 if __name__ == '__main__':      # Program entrance
42     print ('Program is starting...')
43     setup()
44     try:
45         loop()
46     except KeyboardInterrupt:    # Press ctrl-c to end the program.
47         destroy()
```

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



Chapter 15 Servo

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

Project 15.1 Servo Sweep

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

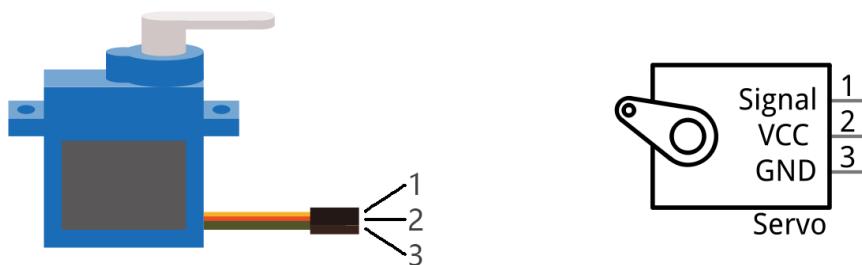
Component List

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

Component knowledge

Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.



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

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

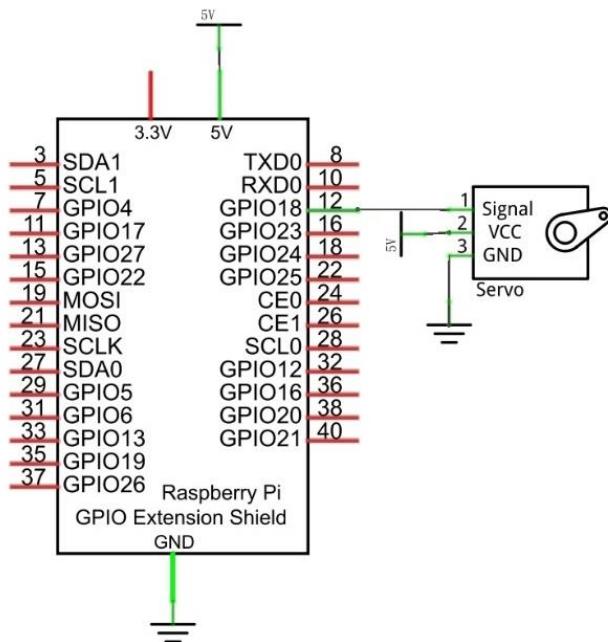
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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

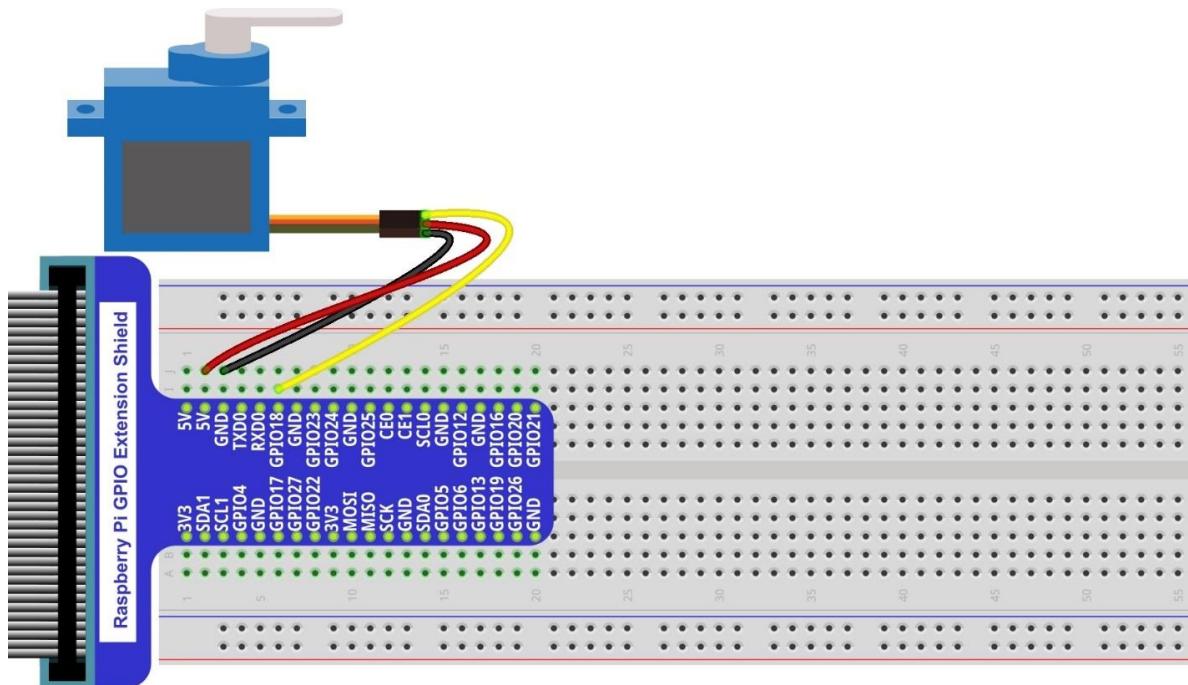
Circuit

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

Schematic diagram



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



[Video: <https://youtu.be/leptbJh32ZI>](https://youtu.be/leptbJh32ZI)

[Sorry latter chapters don't have videos yet.](#)

Code

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

C Code 15.1.1 Sweep

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

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

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

```
cd ~/Freenove_Kit/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 reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

```
1 #include <wiringPi.h>
2 #include <softPwm.h>
3 #include <stdio.h>
4 #define OFFSET_MS 3      //Define the unit of servo pulse offset: 0.1ms
5 #define SERVO_MIN_MS 5+OFFSET_MS          //define the pulse duration for minimum angle of servo
6 #define SERVO_MAX_MS 25+OFFSET_MS        //define the pulse duration for maximum angle of servo
7
8 #define servoPin    1      //define the GPIO number connected to servo
9 long map(long value, long fromLow, long fromHigh, long toLow, long toHigh) {
10     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
11 }
12 void servoInit(int pin){           //initialization function for servo PMW pin
13     softPwmCreate(pin, 0, 200);
14 }
15 void servoWrite(int pin, int angle){ //Specify a certain rotation angle (0-180) for the
16 servo
17     if(angle > 180)
18         angle = 180;
19     if(angle < 0)
20         angle = 0;
21     softPwmWrite(pin, map(angle, 0, 180, SERVO_MIN_MS, SERVO_MAX_MS));
22 }
23 void servoWriteMS(int pin, int ms){ //specific the unit for pulse(5-25ms) with specific
24 duration output by servo pin: 0.1ms
25     if(ms > SERVO_MAX_MS)
26         ms = SERVO_MAX_MS;
27     if(ms < SERVO_MIN_MS)
```

```

28         ms = SERVO_MIN_MS;
29         softPwmWrite(pin, ms);
30     }
31
32     int main(void)
33     {
34         int i;
35
36         printf("Program is starting ... \n");
37
38         wiringPiSetup();
39         servoInit(servoPin);           //initialize PWM pin of servo
40         while(1){
41             for(i=SEROV_MIN_MS;i<SERVO_MAX_MS;i++) { //make servo rotate from minimum angle to
42             maximum angle
43                 servoWriteMS(servoPin, i);
44                 delay(10);
45             }
46             delay(500);
47             for(i=SEROV_MAX_MS;i>SERVO_MIN_MS;i--) { //make servo rotate from maximum angle to
48             minimum angle
49                 servoWriteMS(servoPin, i);
50                 delay(10);
51             }
52             delay(500);
53         }
54         return 0;
55     }

```

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

```

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

```

Since 0-180 degrees of the Servo's motion corresponds to the PWM pulse width of 0.5-2.5ms, with a PwmRange of 200 ms. We then need the function **softPwmWrite** (int pin, int value) and the scope 5-25 of the parameter values to correspond to 0-180 degrees' motion of the Servo. What's more, the number written in subfunction **servoWriteMS()** should be within the range of 5-25. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

```
#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()**, directly input an angle value (0-180 degrees), 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" loop of the main function, use two "for" cycle to make servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

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

Python Code 15.1.1 Sweep

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

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

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

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

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

```
python Sweep.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2
3 OFFSE_DUTY = 0.5      #define pulse offset of servo
4 SERVO_MIN_DUTY = 2.5+OFFSE_DUTY    #define pulse duty cycle for minimum angle of servo
5 SERVO_MAX_DUTY = 12.5+OFFSE_DUTY  #define pulse duty cycle for maximum angle of servo
6 servoPin = 12
7
8 def map( value, fromLow, fromHigh, toLow, toHigh): # map a value from one range to another
9     range
10    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
11
12 def setup():
13     global p
14     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
15     GPIO.setup(servoPin, GPIO.OUT) # Set servoPin to OUTPUT mode
16     GPIO.output(servoPin, GPIO.LOW) # Make servoPin output LOW level
17
18     p = GPIO.PWM(servoPin, 50)    # set Freqeuce to 50Hz
19     p.start(0)                  # Set initial Duty Cycle to 0
20
21 def servoWrite(angle):      # make the servo rotate to specific angle, 0-180
22     if(angle<0):
23         angle = 0
24     elif(angle > 180):
25         angle = 180
26     p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY)) # map the angle to duty
27     cycle and output it
28
29 def loop():
30     while True:
31         for dc in range(0, 181, 1): # make servo rotate from 0 to 180 deg
32             servoWrite(dc)      # Write dc value to servo
33             time.sleep(0.001)
34             time.sleep(0.5)
```

```

35     for dc in range(180, -1, -1): # make servo rotate from 180 to 0 deg
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 entrance
45     print ('Program is starting...')
46     setup()
47     try:
48         loop()
49     except KeyboardInterrupt: # Press ctrl-c to end the program.
50         destroy()

```

A 50 Hz pulse for a 20ms cycle is required to control the Servo. So we need to set the PWM frequency of servoPin to 50Hz.

```
p = GPIO.PWM(servoPin, 50)      # Set Frequency to 50Hz
```

As 0-180 degrees of the Servo's rotation corresponds to the 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 output the PWM, then the Servo will rotate to specifically determined angle. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

```

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, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

```
def loop():
    while True:
        for 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)
```

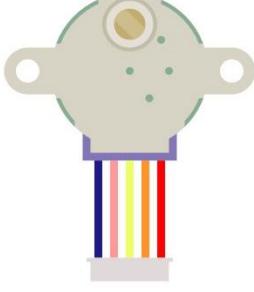
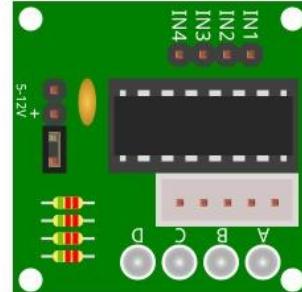
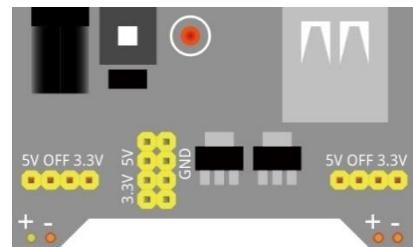
Chapter 16 Stepper Motor

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

Project 16.1 Stepper Motor

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

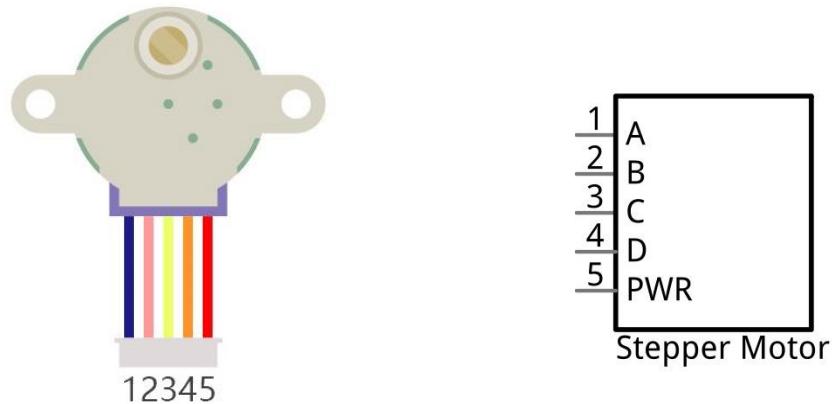
Component List

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

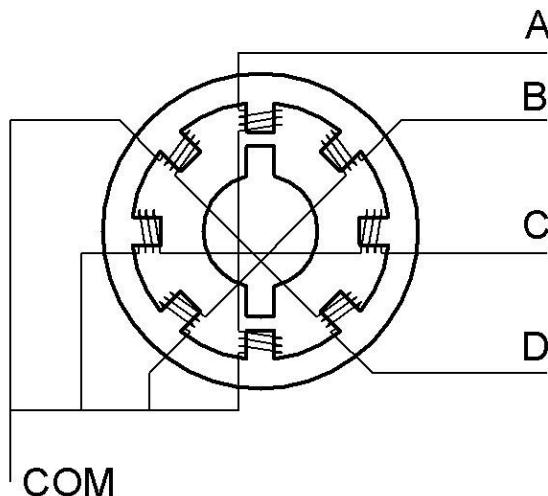
Component knowledge

Stepper Motor

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:

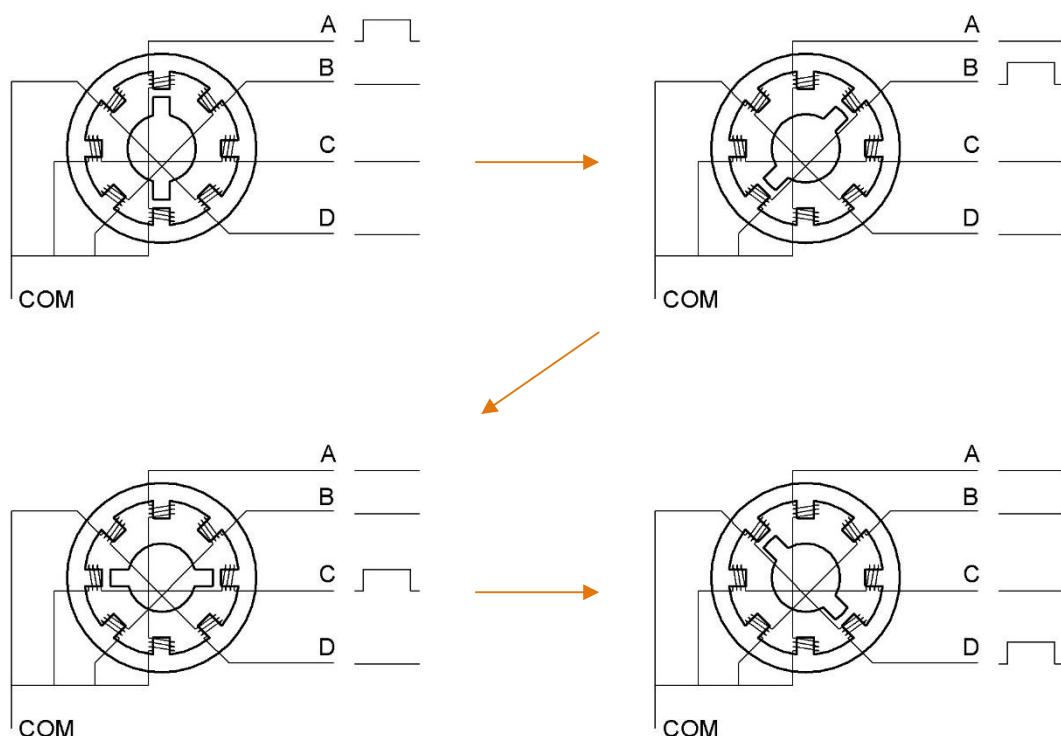


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There is a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor's surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving sequence is shown here:



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

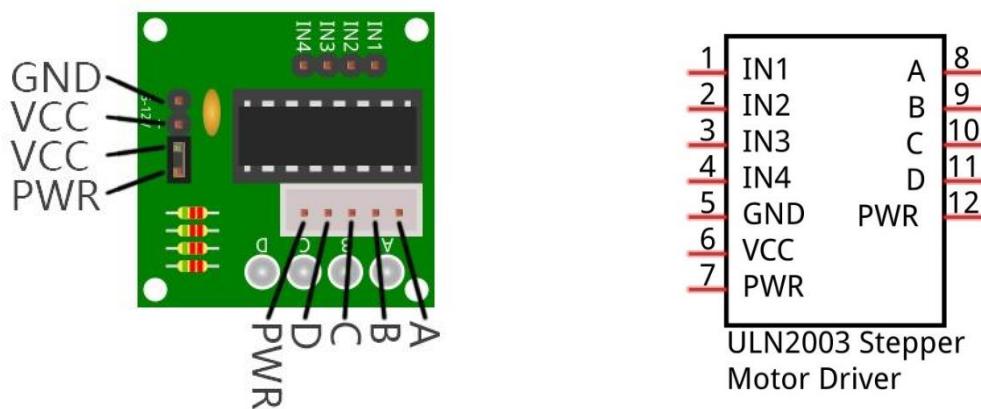
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. This sequence of powering the coils looks like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

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



ULN2003 Stepper Motor driver

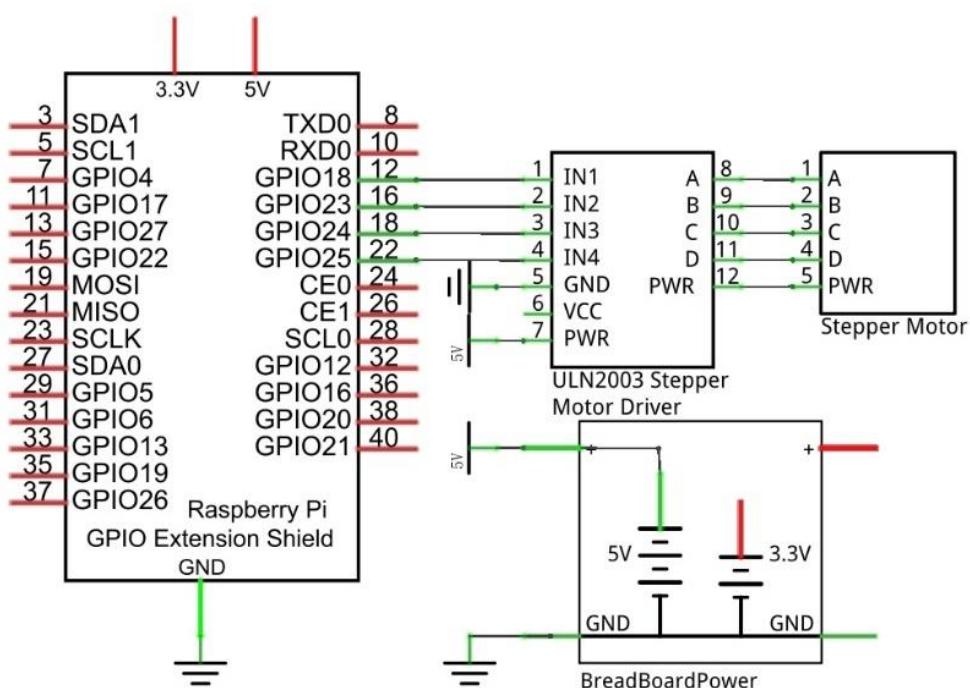
A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals.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



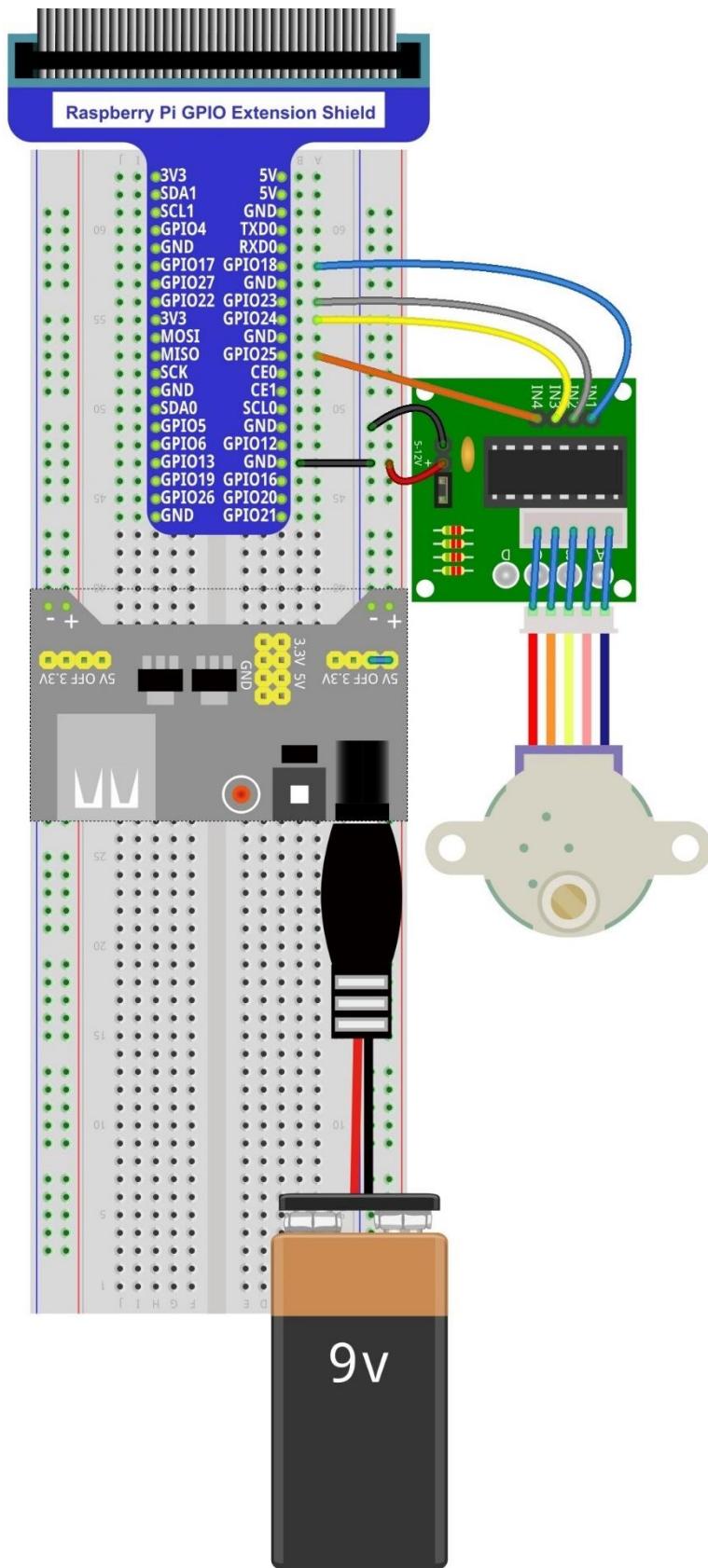
Circuit

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

Schematic diagram



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



Code

C Code 16.1.1 SteppingMotor

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

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

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

```
cd ~/Freenove_Kit/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 Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

```

1 #include <stdio.h>
2 #include <wiringPi.h>
3
4 const int motorPins[]={1, 4, 5, 6}; //define pins connected to four phase ABCD of stepper
5 motor
6 const int CCWStep[]={0x01, 0x02, 0x04, 0x08}; //define power supply order for coil for rotating
7 anticlockwise
8 const int CWStep[]={0x08, 0x04, 0x02, 0x01}; //define power supply order for coil for rotating
9 clockwise
10 //as for four phase Stepper Motor, four steps is a cycle. the function is used to drive the
11 Stepper Motor clockwise or anticlockwise to take four steps
12 void moveOnePeriod(int dir,int ms){
13     int i=0, j=0;
14     for (j=0;j<4;j++) { //cycle according to power supply order
15         for (i=0;i<4;i++) { //assign to each pin, a total of 4 pins
16             if(dir == 1) //power supply order clockwise
17                 digitalWrite(motorPins[i], (CCWStep[j] == (1<<i)) ? HIGH : LOW);
18             else //power supply order anticlockwise
19                 digitalWrite(motorPins[i], (CWStep[j] == (1<<i)) ? HIGH : LOW);
20             printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
21         }
22         printf("Step cycle!\n");
23         if(ms<3) //the delay can not be less than 3ms, otherwise it will exceed speed
24         limit of the motor
25             ms=3;
26             delay(ms);
27         }
28     } //continuous rotation function, the parameter steps specifies the rotation cycles, every four

```

```

30 steps is a cycle
31 void moveSteps(int dir, int ms, int steps){
32     int i;
33     for(i=0;i<steps;i++) {
34         moveOnePeriod(dir, ms);
35     }
36 }
37 void motorStop() { //function used to stop rotating
38     int i;
39     for(i=0;i<4;i++) {
40         digitalWrite(motorPins[i], LOW);
41     }
42 }
43 int main(void) {
44     int i;
45
46     printf("Program is starting ... \n");
47
48     wiringPiSetup();
49
50     for(i=0;i<4;i++) {
51         pinMode(motorPins[i], OUTPUT);
52     }
53
54     while(1) {
55         moveSteps(1, 3, 512); //rotating 360° clockwise, a total of 2048 steps in a circle,
56         namely, 512 cycles.
57         delay(500);
58         moveSteps(0, 3, 512); //rotating 360° anticlockwise
59         delay(500);
60     }
61     return 0;
62 }
```

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

```

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 rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between

each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

```
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 Stepper Motor.

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

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

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

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

```
while(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);
}
```

Python Code 16.1.1 SteppingMotor

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

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

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

cd ~/Freenove_Kit/Code/Python_Code/16.1.1_SteppingMotor

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

python SteppingMotor.py

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2 import time
3
4 motorPins = (12, 16, 18, 22)      # define pins connected to four phase ABCD of stepper motor
5 CCWStep = (0x01, 0x02, 0x04, 0x08) # define power supply order for rotating anticlockwise
6 CWStep = (0x08, 0x04, 0x02, 0x01)  # define power supply order for rotating clockwise
7
8 def setup():
9     GPIO.setmode(GPIO.BOARD)          # use PHYSICAL GPIO Numbering
10    for pin in motorPins:
11        GPIO.setup(pin, GPIO.OUT)
12
13    # as for four phase Stepper Motor, four steps is a cycle. the function is used to drive the
14    Stepper Motor clockwise or anticlockwise to take four steps
15    def moveOnePeriod(direction, ms):
16        for j in range(0, 4, 1):       # cycle for power supply order
17            for i in range(0, 4, 1):  # assign to each pin
18                if (direction == 1):# power supply order clockwise
19                    GPIO.output(motorPins[i], ((CCWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
20                else :                  # power supply order anticlockwise
21                    GPIO.output(motorPins[i], ((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
22                if(ms<3):           # the delay can not be less than 3ms, otherwise it will exceed speed
23                    limit of the motor
24                ms = 3
25                time.sleep(ms*0.001)
26
27    # continuous rotation function, the parameter steps specify the rotation cycles, every four
28    steps is a cycle
29    def moveSteps(direction, ms, steps):
30        for i in range(steps):
31            moveOnePeriod(direction, ms)
32
33    # function used to stop motor
34    def motorStop():
```

```

35     for i in range(0, 4, 1):
36         GPIO.output(motorPins[i], GPIO.LOW)
37
38     def loop():
39         while True:
40             moveSteps(1, 3, 512) # rotating 360 deg clockwise, a total of 2048 steps in a circle,
41             512 cycles
42             time.sleep(0.5)
43             moveSteps(0, 3, 512) # rotating 360 deg anticlockwise
44             time.sleep(0.5)
45
46     def destroy():
47         GPIO.cleanup()          # Release resource
48
49     if __name__ == '__main__':    # Program entrance
50         print ('Program is starting...')
51         setup()
52         try:
53             loop()
54         except KeyboardInterrupt: # Press ctrl-c to end the program.
55             destroy()

```

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

```

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

```

def moveOnePeriod(direction, ms):
    for j in range(0, 4, 1):      #cycle for power supply order
        for i in range(0, 4, 1):  #assign to each pin, a total of 4 pins
            if (direction == 1):#power supply order clockwise
                GPIO.output(motorPins[i], ((CCWStep[j] == 1<<i) and GPIO.HIGH or GPIO.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 specify the cycle number of Stepper Motor.

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

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

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

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

```
while True:  
    moveSteps(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 & Bar Graph LED

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

Project 17.1 Flowing Water Light

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

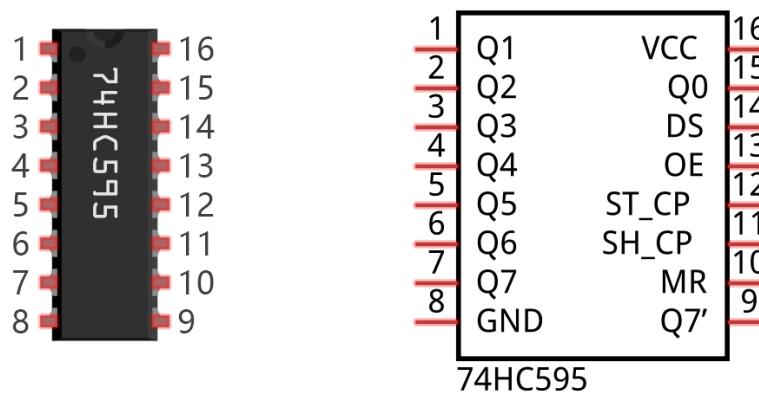
Component List

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

Component knowledge

74HC595

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



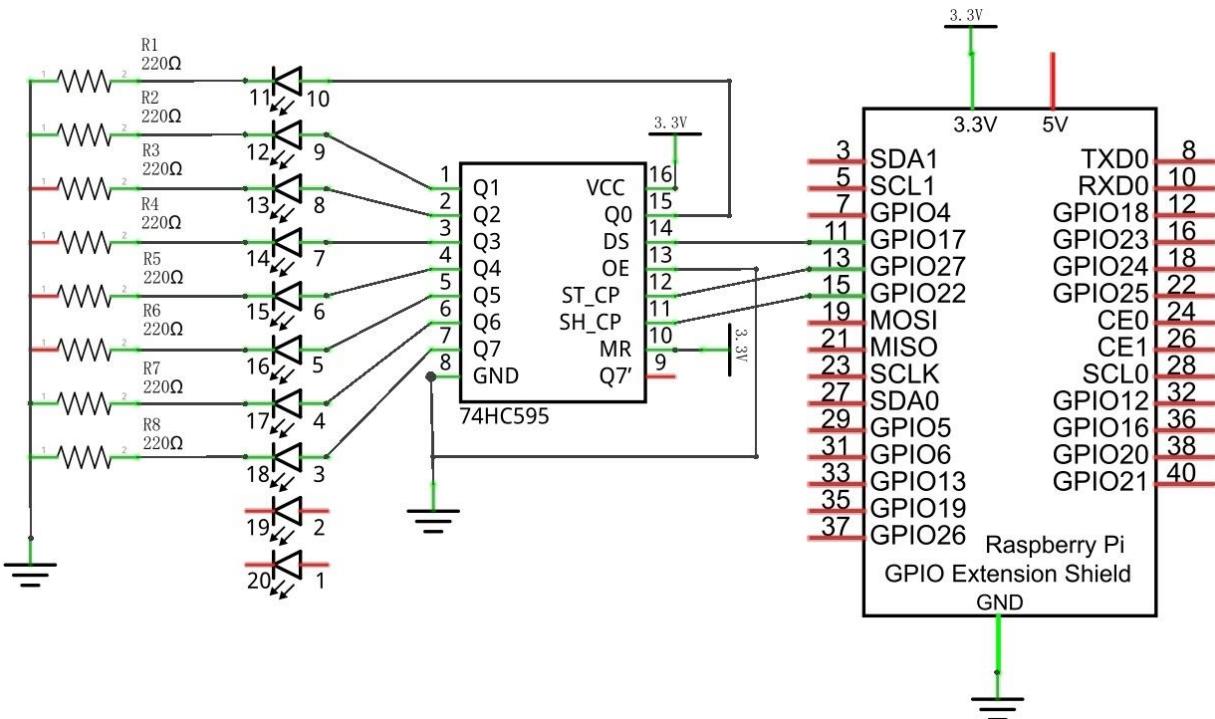
The ports of the 74HC595 chip are described as follows:

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

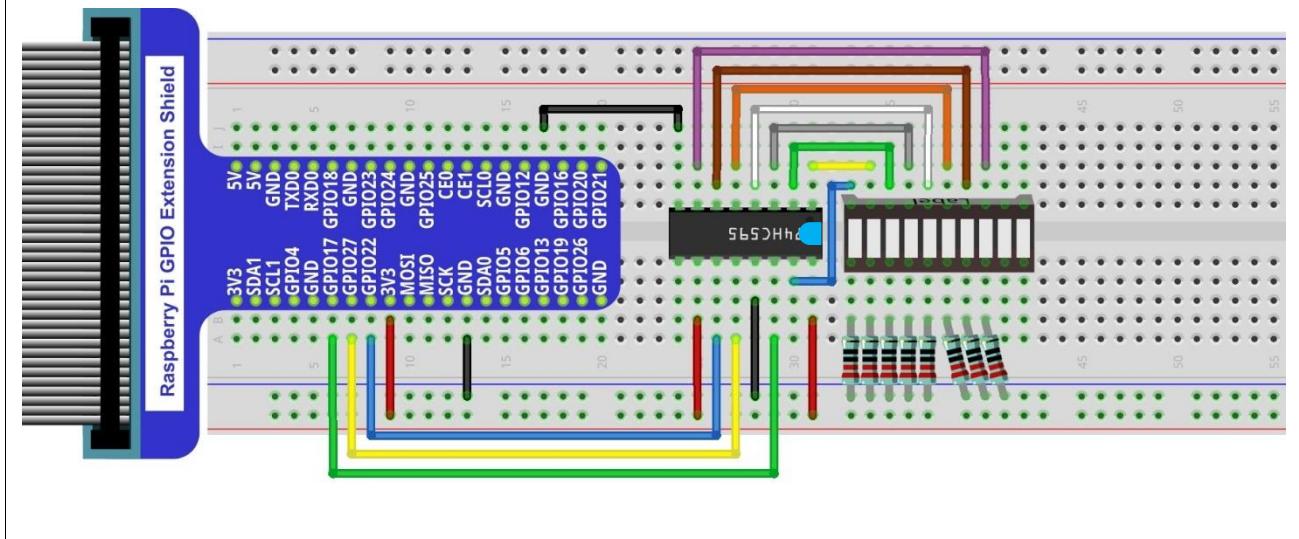
For more details, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



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



Code

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

C Code 17.1.1 LightWater02

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

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

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

```
cd ~/Freenove_Kit/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, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.

The following is the program code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <wiringShift.h>
4
5 #define  dataPin 0 //DS Pin of 74HC595(Pin14)
6 #define  latchPin 2 //ST_CP Pin of 74HC595(Pin12)
7 #define  clockPin 3 //CH_CP Pin of 74HC595(Pin11)
8
9 void _shiftOut(int dPin, int cPin, int order, int val) {
10     int i;
11     for(i = 0; i < 8; i++) {
12         digitalWrite(cPin, LOW);
13         if(order == LSBFIRST) {
14             digitalWrite(dPin, ((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
15             delayMicroseconds(10);
16         }
17         else {//if(order == MSBFIRST) {
18             digitalWrite(dPin, ((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
19             delayMicroseconds(10);
20         }
21         digitalWrite(cPin, HIGH);
22         delayMicroseconds(10);
23     }
24 }
25
26 int main(void)
27 {
28     int i;
```

```

29     unsigned char x;
30
31     printf("Program is starting ... \n");
32
33     wiringPiSetup();
34
35     pinMode(dataPin, OUTPUT);
36     pinMode(latchPin, OUTPUT);
37     pinMode(clockPin, OUTPUT);
38     while(1) {
39         x=0x01;
40         for(i=0;i<8;i++) {
41             digitalWrite(latchPin,LOW);           // Output low level to latchPin
42             _shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
43             digitalWrite(latchPin,HIGH);        //Output high level to latchPin, and 74HC595 will
44             update the data to the parallel output port.
45             x<<=1;           //make the variable move one bit to left once, then the bright LED
46             move one step to the left once.
47             delay(100);
48         }
49         x=0x80;
50         for(i=0;i<8;i++) {
51             digitalWrite(latchPin,LOW);
52             _shiftOut(dataPin,clockPin,LSBFIRST,x);
53             digitalWrite(latchPin,HIGH);
54             x>>=1;
55             delay(100);
56         }
57     }
58     return 0;
59 }
```

In the code, we configure three pins to control the 74HC595 chip and define a one-byte variable to control the state of the 8 LEDs (in the Bar Graph LED Module) through the 8 bits of the variable. The LEDs light 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 two cycles to send x to 74HC595 output pin to control the LED. In one cycle, x will shift one bit to the LEFT in one cycle, then when data of x is sent to 74HC595, the LED that is turned ON will move one bit to the 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);

}

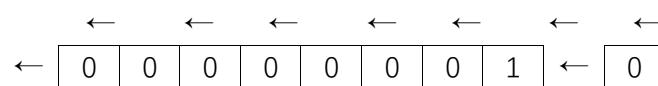
```

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

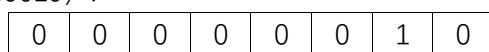
<< operator

"<<" 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;

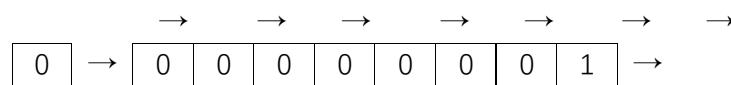


The result of x is 2 (binary 00000010) .



There 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) .



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 is used to shift 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);`

`void _shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val);`

This is used to shift an 8-bit data value 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/>



Python Code 17.1.1 LightWater02

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

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

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

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

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

```
python LightWater02.py
```

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

The following is the program code:

```
1 import RPi.GPIO as GPIO
2
3 # Defines the data bit that is transmitted preferentially in the shiftOut function.
4 LSBFIRST = 1
5 MSBFIRST = 2
6
7 # define the pins for 74HC595
8 dataPin    = 11      # DS Pin of 74HC595(Pin14)
9 latchPin   = 13      # ST_CP Pin of 74HC595(Pin12)
10 clockPin  = 15      # CH_CP Pin of 74HC595(Pin11)
11
12 def setup():
13     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
14     GPIO.setup(dataPin, GPIO.OUT) # set pin to OUTPUT mode
15     GPIO.setup(latchPin, GPIO.OUT)
16     GPIO.setup(clockPin, GPIO.OUT)
17
18 # shiftOut function, use bit serial transmission.
19 def shiftOut(dPin, cPin, order, val):
20     for i in range(0,8):
21         GPIO.output(cPin,GPIO.LOW);
22         if(order == LSBFIRST):
23             GPIO.output(dPin, (0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
24         elif(order == MSBFIRST):
25             GPIO.output(dPin, (0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
26         GPIO.output(cPin,GPIO.HIGH);
27
28 def loop():
29     while True:
30         x=0x01
31         for i in range(0,8):
32             GPIO.output(latchPin,GPIO.LOW) # Output low level to latchPin
33             shiftOut(dataPin,clockPin,LSBFIRST,x) # Send serial data to 74HC595
34             GPIO.output(latchPin,GPIO.HIGH) # Output high level to latchPin, and 74HC595 will
35             update the data to the parallel output port.
```

```

35         x<<=1 # make the variable move one bit to left once, then the bright LED move one
36         step to the left once.
37             time.sleep(0.1)
38             x=0x80
39             for i in range(0,8):
40                 GPIO.output(latchPin,GPIO.LOW)
41                 shiftOut(dataPin,clockPin,LSBFIRST,x)
42                 GPIO.output(latchPin,GPIO.HIGH)
43                 x>>=1
44                 time.sleep(0.1)
45
46     def destroy():
47         GPIO.cleanup()
48
49     if __name__ == '__main__': # Program entrance
50         print ('Program is starting... ')
51         setup()
52         try:
53             loop()
54         except KeyboardInterrupt: # Press ctrl-c to end the program.
55             destroy()

```

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

```

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 cycles to achieve the action goal. First, define a variable x=0x01, binary 00000001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then an LED turns ON. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED that turns ON will be shifted. Repeat the operation, over and over and the effect of a flowing water light will be visible. If the direction of the shift operation for x is different, the flowing direction is different.

```

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 introduce the 7-Segment Display.

Project 18.1 7-Segment Display

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

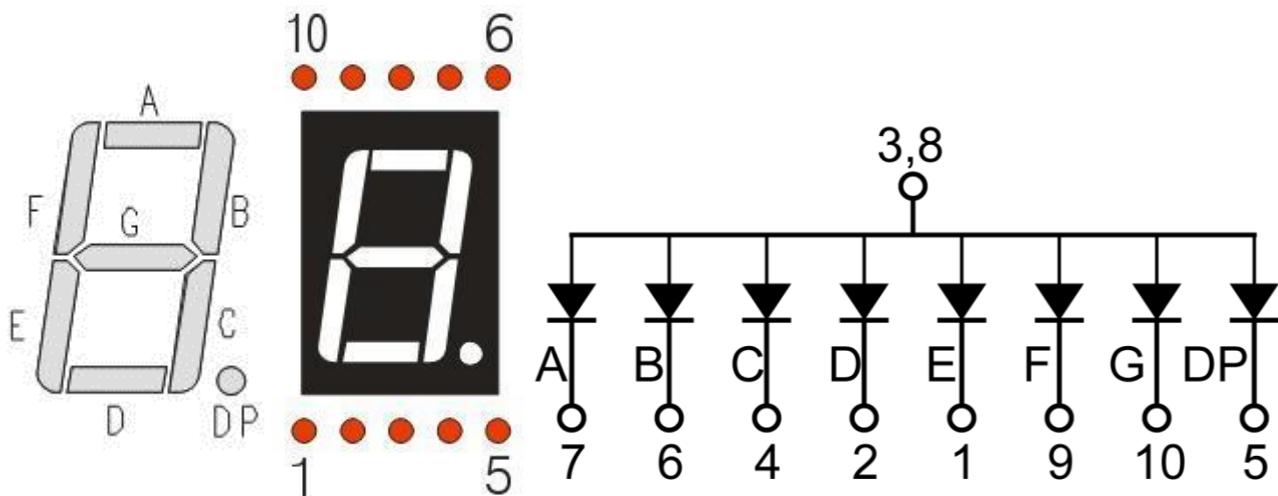
Component List

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

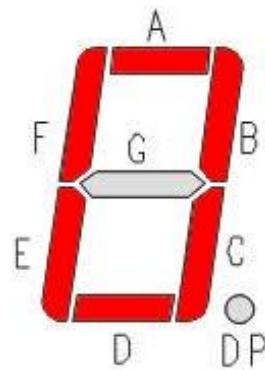
Component knowledge

7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



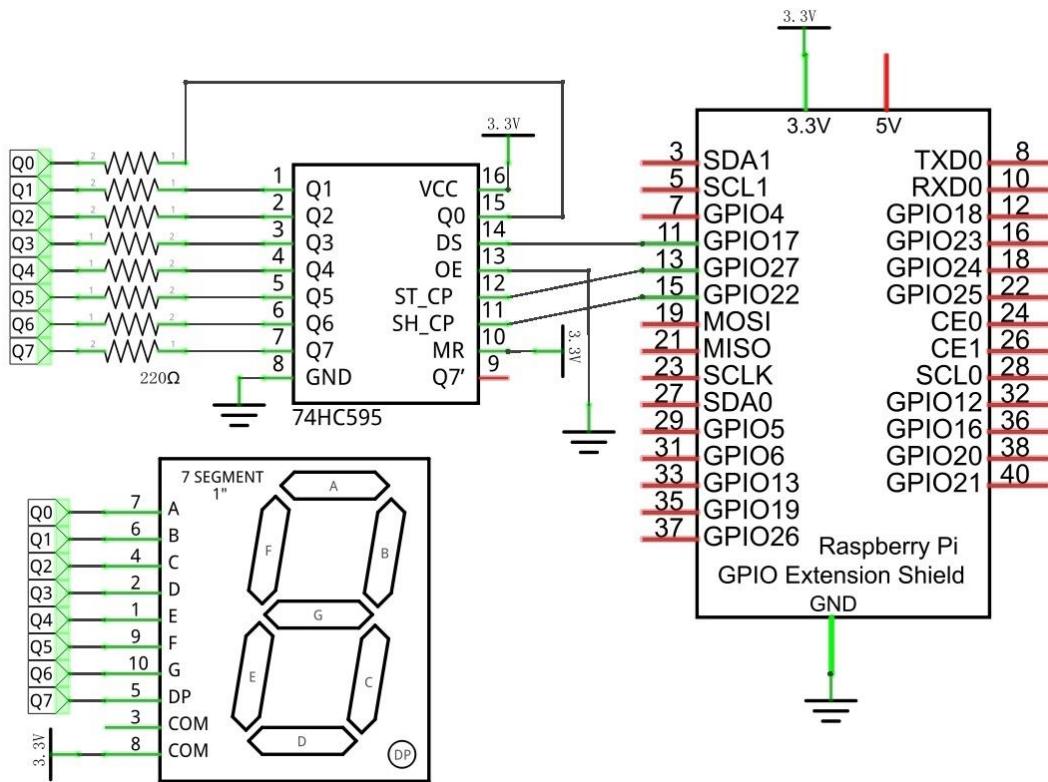
As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a "0": we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.



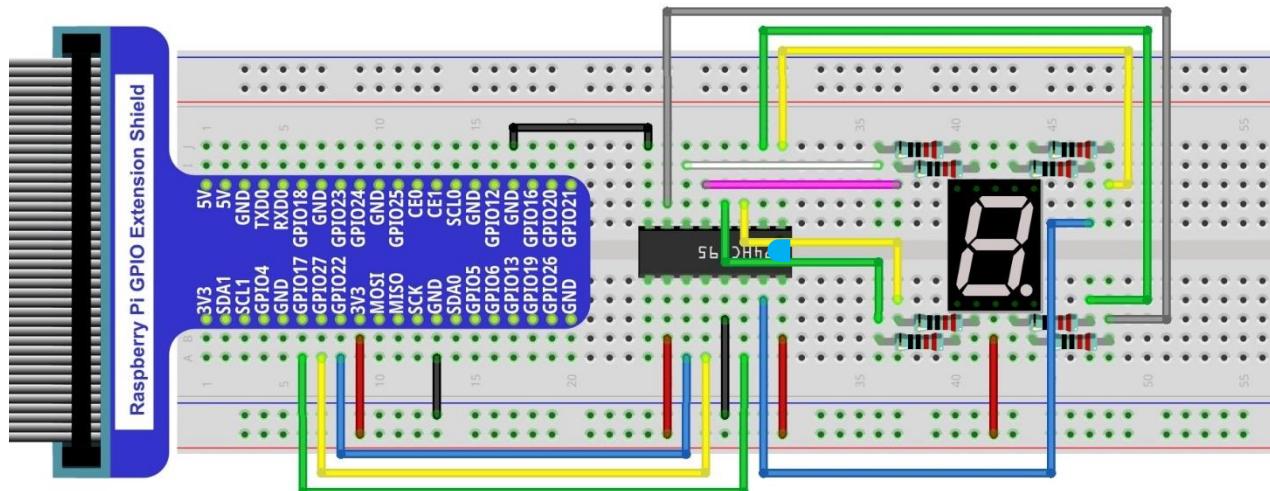
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment "A" as the lowest level and segment "DP" as the highest level, from high to low would look like this: "DP", "G", "F", "E", "D", "C", "B", "A". Character "0" corresponds to the code: $1100\ 0000_2 = 0xc0$.

Circuit

Schematic diagram



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



Video: <https://youtu.be/KSE0LdyuOFM>

Code

This code uses a 74HC595 IC Chip to control the 7-Segment Display. The use of the 74HC595 IC Chip is generally the same throughout this Tutorial. We need code to display the characters “0” to “F” one character at a time, and then output to display them with the 74HC595 IC Chip.

C Code 18.1.1 SevenSegmentDisplay

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

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

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

```
cd ~/Freenove_Kit/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, the 7-Segment Display starts to display the characters “0” to “F” in succession.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <wiringShift.h>
4
5 #define dataPin 0 //DS Pin of 74HC595(Pin14)
6 #define latchPin 2 //ST_CP Pin of 74HC595(Pin12)
7 #define clockPin 3 //CH_CP Pin of 74HC595(Pin11)
8 //encoding for character 0-F of common anode SevenSegmentDisplay.
9 unsigned char
10 num[]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e} ;
11
12 void _shiftOut(int dPin, int cPin, int order, int val) {
13     int i;
14     for(i = 0; i < 8; i++) {
15         digitalWrite(cPin, LOW);
16         if(order == LSBFIRST) {
17             digitalWrite(dPin, ((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
18             delayMicroseconds(10);
19         }
20         else {//if(order == MSBFIRST) {
21             digitalWrite(dPin, ((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
22             delayMicroseconds(10);
23         }
24         digitalWrite(cPin, HIGH);
25         delayMicroseconds(10);
26     }
}
```

```

27 }
28
29 int main(void)
30 {
31     int i;
32
33     printf("Program is starting ... \n");
34
35     wiringPiSetup();
36
37     pinMode(dataPin, OUTPUT);
38     pinMode(latchPin, OUTPUT);
39     pinMode(clockPin, OUTPUT);
40     while(1) {
41         for(i=0;i<sizeof(num);i++) {
42             digitalWrite(latchPin, LOW);
43             _shiftOut(dataPin, clockPin, MSBFIRST, num[i]); //Output the figures and the highest
44             level is transferred preferentially.
45             digitalWrite(latchPin, HIGH);
46             delay(500);
47         }
48         for(i=0;i<sizeof(num);i++) {
49             digitalWrite(latchPin, LOW);
50             _shiftOut(dataPin, clockPin, MSBFIRST, num[i] & 0x7f); //Use the "&0x7f" to display
51             the decimal point.
52             digitalWrite(latchPin, HIGH);
53             delay(500);
54         }
55     }
56     return 0;
57 }
```

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

```

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

In the “for” loop of loop() function, use the 74HC595 IC Chip to output contents of array “num” successively. SevenSegmentDisplay can then correctly display the corresponding characters. Pay attention to this in regard to shiftOut function, the transmission bit, flag bit and 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 transferred preferentially.
    digitalWrite(latchPin, HIGH);
    delay(500);
}
```

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

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

Python Code 18.1.1 SevenSegmentDisplay

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

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

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

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

2. Use Python command to execute Python code “SevenSegmentDisplay.py”.

```
python SevenSegmentDisplay.py
```

After the program is executed, the 7-Segment Display starts to display the characters “0” to “F” in succession.

The following is the program code:

```

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)
8 latchPin = 13     #ST_CP Pin of 74HC595(Pin12)
9 clockPin = 15      #CH_CP Pin of 74HC595(Pin11)
10 #SevenSegmentDisplay display the character "0"- "F" successively
11 num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
12 def setup():
13     GPIO.setmode(GPIO.BOARD)      # Number GPIOs by its physical location
14     GPIO.setup(dataPin, GPIO.OUT)
15     GPIO.setup(latchPin, GPIO.OUT)
16     GPIO.setup(clockPin, GPIO.OUT)
17
18 def shiftOut(dPin, cPin, order, val):
19     for i in range(0, 8):
20         GPIO.output(cPin, GPIO.LOW);
21         if(order == LSBFIRST):
22             GPIO.output(dPin, (0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
23         elif(order == MSBFIRST):
24             GPIO.output(dPin, (0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
25         GPIO.output(cPin, GPIO.HIGH);
26
27 def loop():
28     while True:
29         for i in range(0, len(num)):
30             GPIO.output(latchPin, GPIO.LOW)
31

```

```

32         shiftOut(dataPin, clockPin, MSBFIRST, num[i])#Output the figures and the highest
33 level is transferred preferentially.
34         GPIO.output(latchPin,GPIO.HIGH)
35         time.sleep(0.5)
36     for i in range(0, len(num)):
37         GPIO.output(latchPin,GPIO.LOW)
38         shiftOut(dataPin, clockPin, MSBFIRST, num[i]&0x7f)#Use "&0x7f" to display the
39 decimal point.
40         GPIO.output(latchPin,GPIO.HIGH)
41         time.sleep(0.5)
42
43 def destroy():
44     GPIO.cleanup()
45
46 if __name__ == '__main__': # Program starting from here
47     print ('Program is starting... ')
48     setup()
49     try:
50         loop()
51     except KeyboardInterrupt:
52         destroy()

```

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

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

In the “for” loop of loop() function, use the 74HC595 IC Chip to output contents of array “num” successively. SevenSegmentDisplay can then correctly display the corresponding characters. Pay attention to this in regard to shiftOut function, the transmission bit, flag bit and 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 transferred 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 “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-than-one digit displays by using a Four 7-Segment Display in one project.

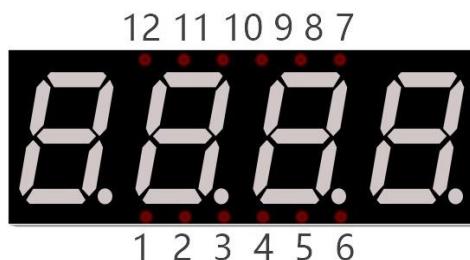
Component List

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

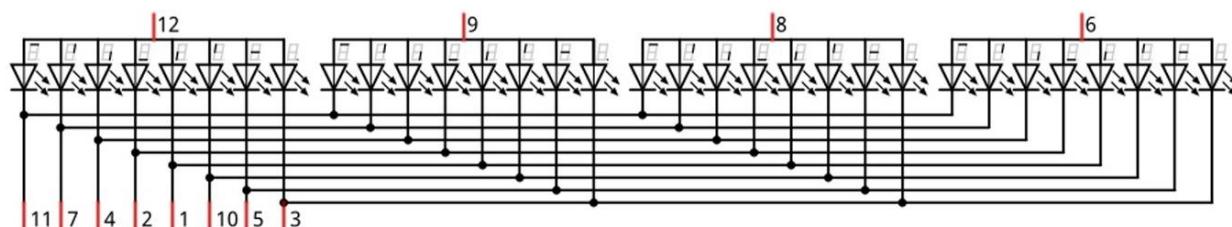
Component knowledge

4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-Segment Displays into one module, therefore it can display more characters. All of the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



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

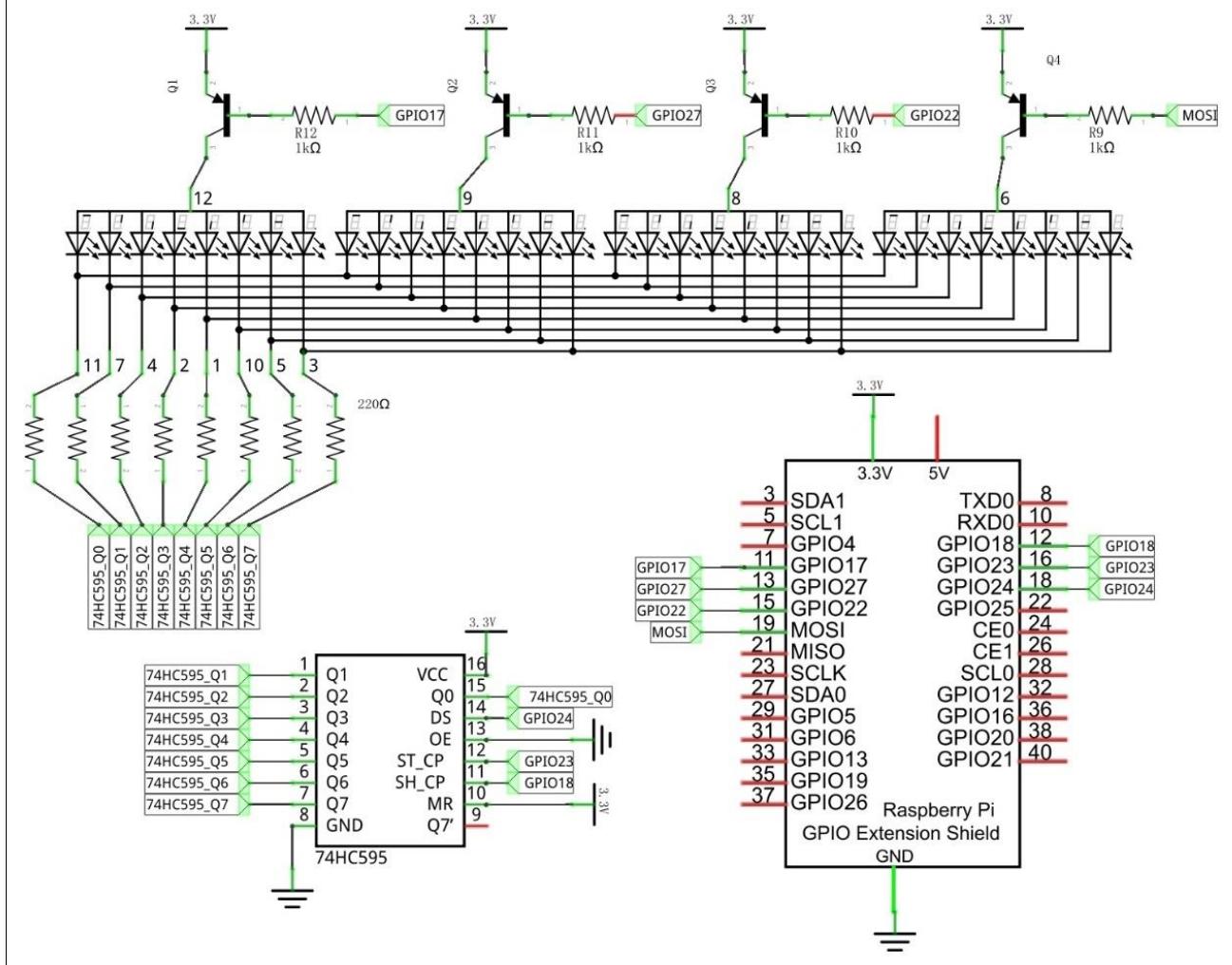


Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

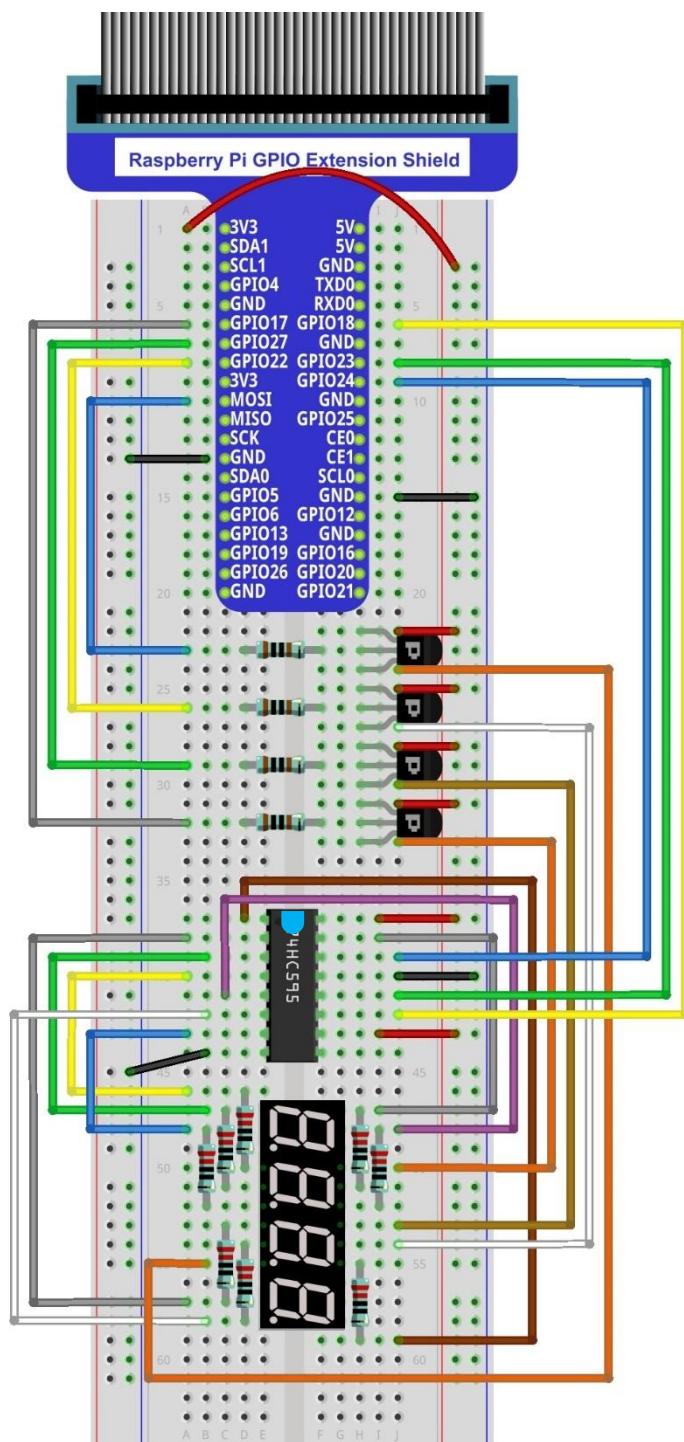
Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so very fast that it is unperceivable to the naked eye. This is due to the principle of optical afterglow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

Circuit

Schematic diagram



Hardware connection



Code

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

C Code 18.2.1 StopWatch

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

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

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

```
cd ~/Freenove_Kit/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 "StopWatch".

```
sudo ./StopWatch
```

After the program is executed, the 4-Digit 7-Segment Display starts displaying a four-digit number dynamically, and the numeric value of this number will increase by plus 1 each second thereafter.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <wiringShift.h>
4 #include <signal.h>
5 #include <unistd.h>
6 #define    dataPin      5 //DS Pin of 74HC595(Pin14)
7 #define    latchPin     4 //ST_CP Pin of 74HC595(Pin12)
8 #define    clockPin     1 //CH_CP Pin of 74HC595(Pin11)
9 const int digitPin[]={0,2,3,12};           // Define 7-segment display common pin
10 // character 0~9 code of common anode 7-segment display
11 unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
12 int counter = 0; //variable counter, the number will be displayed by 7-segment display
13 //Open one of the 7-segment display and close the remaining three, the parameter digit is
14 optional for 1,2,4,8
15 void selectDigit(int digit){
16     digitalWrite(digitPin[0], ((digit&0x08) == 0x08) ? LOW : HIGH);
17     digitalWrite(digitPin[1], ((digit&0x04) == 0x04) ? LOW : HIGH);
18     digitalWrite(digitPin[2], ((digit&0x02) == 0x02) ? LOW : HIGH);
19     digitalWrite(digitPin[3], ((digit&0x01) == 0x01) ? LOW : HIGH);
20 }
21 void _shiftOut(int dPin, int cPin, int order, int val) {
22     int i;
23     for(i = 0; i < 8; i++){
24         digitalWrite(cPin, LOW);
25         if(order == LSBFIRST) {
26             digitalWrite(dPin, ((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
27             delayMicroseconds(1);

```

```
28     }
29     else {//if(order == MSBFIRST) {
30         digitalWrite(dPin, ((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
31         delayMicroseconds(1);
32     }
33     digitalWrite(cPin, HIGH);
34     delayMicroseconds(1);
35 }
36 }
37 void outData(int8_t data){      //function used to output data for 74HC595
38     digitalWrite(latchPin,LOW);
39     _shiftOut(dataPin,clockPin,MSBFIRST,data);
40     digitalWrite(latchPin,HIGH);
41 }
42 void display(int dec){ //display function for 7-segment display
43     int delays = 1;
44     outData(0xff);
45     selectDigit(0x01);      //select the first, and display the single digit
46     outData(num[dec%10]);
47     delay(delays);        //display duration
48
49     outData(0xff);
50     selectDigit(0x02);      //select the second, and display the tens digit
51     outData(num[dec%100/10]);
52     delay(delays);
53
54     outData(0xff);
55     selectDigit(0x04);      //select the third, and display the hundreds digit
56     outData(num[dec%1000/100]);
57     delay(delays);
58
59     outData(0xff);
60     selectDigit(0x08);      //select the fourth, and display the thousands digit
61     outData(num[dec%10000/1000]);
62     delay(delays);
63 }
64 void timer(int sig){ //Timer function
65     if(sig == SIGALRM){ //If the signal is SIGALRM, the value of counter plus 1, and update
66         the number displayed by 7-segment display
67         counter++;
68         alarm(1);           //set the next timer time
69         printf("counter : %d \n",counter);
70     }
71 }
```

```

72 int main(void)
73 {
74     int i;
75
76     printf("Program is starting ... \n");
77
78     wiringPiSetup();
79
80     pinMode(dataPin, OUTPUT);           //set the pin connected to 74HC595 for output mode
81     pinMode(latchPin, OUTPUT);
82     pinMode(clockPin, OUTPUT);
83     //set the pin connected to 7-segment display common end to output mode
84     for(i=0;i<4;i++) {
85         pinMode(digitPin[i], OUTPUT);
86         digitalWrite(digitPin[i], HIGH);
87     }
88     signal(SIGALRM, timer); //configure the timer
89     alarm(1);             //set the time of timer to 1s
90     while(1) {
91         display(counter); //display the number counter
92     }
93     return 0;
94 }
```

First, we define the pin of the 74HC595 IC Chip and the 7-Segment Display Common Anode, use character encoding and a variable "counter" to enable the counter to be visible on the 7-Segment Display.

```

#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 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 displayed by 7-segment display
```

Subfunction **selectDigit** (int digit) function is used to open one of the 7-Segment Displays while closing the other 7-Segment Displays, where the parameter digit value can be 1,2,4,8. Using "|" can open a number of a 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);
}
```

Subfunction **outData** (int8_t data) is used to make the 74HC595 IC Chip output an 8-bit data immediately.

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

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

```
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, which will set an alarm to signal. This function will be executed once at set time intervals. Accompanied by the execution, "1" will be added as the variable counter 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 the GPIO, and set the timer function.

```
pinMode(dataPin, OUTPUT);           //set the pin connected to 74HC595 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 loop, make the digital display variable counter value “1”. The value will change in function timer (), so the content displayed by the 7-Segment Display will change accordingly.

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

Python Code 18.2.1 StopWatch

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

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

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

```
cd ~/Freenove_Kit/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:

```

1 import RPi.GPIO as GPIO
2 import time
3 import threading
4
5 LSBFIRST = 1
6 MSBFIRST = 2
7 #define the pins connect to 74HC595
8 dataPin = 18      #DS Pin of 74HC595(Pin14)
9 latchPin = 16     #ST_CP Pin of 74HC595(Pin12)
10 clockPin = 12    #SH_CP Pin of 74HC595(Pin11)
11 num = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
12 digitPin = (11, 13, 15, 19)   # Define the pin of 7-segment display common end
13 counter = 0        # Variable counter, the number will be displayed by 7-segment display
14 t = 0              # define the Timer object
15 def setup():
16     GPIO.setmode(GPIO.BOARD)      # Number GPIOs by its physical location
17     GPIO.setup(dataPin, GPIO.OUT)      # Set pin mode to output
18     GPIO.setup(latchPin, GPIO.OUT)
19     GPIO.setup(clockPin, GPIO.OUT)
20     for pin in digitPin:
21         GPIO.setup(pin, GPIO.OUT)
22
23 def shiftOut(dPin, cPin, order, val):
24     for i in range(0, 8):
25         GPIO.output
26         (cPin, GPIO.LOW);
27         if(order == LSBFIRST):
28             GPIO.output(dPin, (0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
29         elif(order == MSBFIRST):
30             GPIO.output(dPin, (0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
31         GPIO.output(cPin, GPIO.HIGH)
32
33 def outData(data):      #function used to output data for 74HC595
34     GPIO.output(latchPin, GPIO.LOW)
```

```
35     shiftOut(dataPin, clockPin, MSBFIRST, data)
36     GPIO.output(latchPin, GPIO.HIGH)
37
38 def selectDigit(digit): # Open one of the 7-segment display and close the remaining
39 three, the parameter digit is optional for 1,2,4,8
40     GPIO.output(digitPin[0],GPIO.LOW if ((digit&0x08) == 0x08) else GPIO.HIGH)
41     GPIO.output(digitPin[1],GPIO.LOW if ((digit&0x04) == 0x04) else GPIO.HIGH)
42     GPIO.output(digitPin[2],GPIO.LOW if ((digit&0x02) == 0x02) else GPIO.HIGH)
43     GPIO.output(digitPin[3],GPIO.LOW if ((digit&0x01) == 0x01) else GPIO.HIGH)
44
45 def display(dec): #display function for 7-segment display
46     outData(0xff) #eliminate residual display
47     selectDigit(0x01) #Select the first, and display the single digit
48     outData(num[dec%10])
49     time.sleep(0.003) #display duration
50     outData(0xff)
51     selectDigit(0x02) # Select the second, and display the tens digit
52     outData(num[dec%100//10])
53     time.sleep(0.003)
54     outData(0xff)
55     selectDigit(0x04) # Select the third, and display the hundreds digit
56     outData(num[dec%1000//100])
57     time.sleep(0.003)
58     outData(0xff)
59     selectDigit(0x08) # Select the fourth, and display the thousands digit
60     outData(num[dec%10000//1000])
61     time.sleep(0.003)
62 def timer(): #timer function
63     global counter
64     global t
65     t = threading.Timer(1.0,timer) #reset time of timer to 1s
66     t.start() #Start timing
67     counter+=1
68     print ("counter : %d"%counter)
69
70 def loop():
71     global t
72     global counter
73     t = threading.Timer(1.0,timer) #set the timer
74     t.start() # Start timing
75     while True:
76         display(counter) # display the number counter
77
78 def destroy():
```

```

79     global t
80     GPIO.cleanup()
81     t.cancel()      #cancel the timer
82
83 if __name__ == '__main__':
84     print ('Program is starting... ')
85     setup()
86     try:
87         loop()
88     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 = (11, 13, 15, 19)    # Define the pin of 7-segment display common end
counter = 0          # Variable counter, the number will be displayed 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)

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

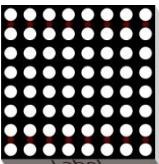
Chapter 19 74HC595 & LED Matrix

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

Project 19.1 LED Matrix

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

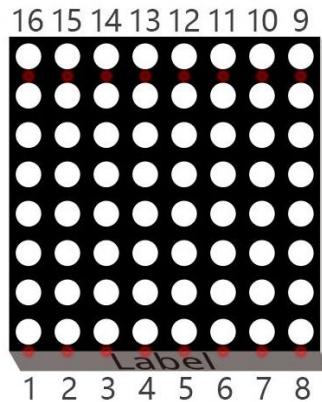
Component List

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

Component knowledge

LED matrix

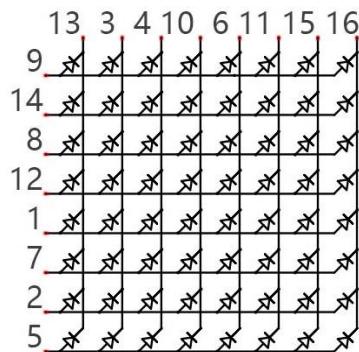
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



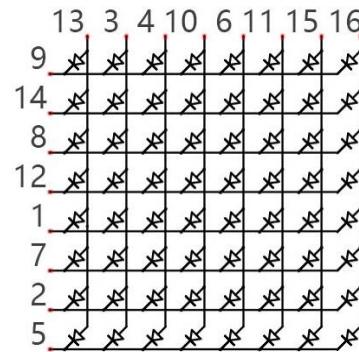
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another arrangement type. Negative Poles of the LEDs in each row and the Positive Poles of the LEDs in each column are respectively connected together, which is called a Common Cathode.

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

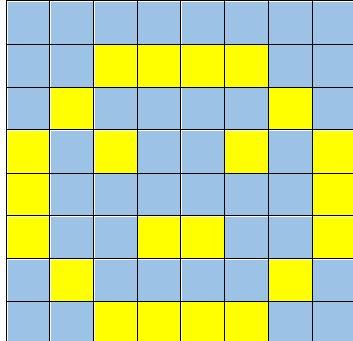
Connection mode of Common Anode



Connection mode of Common Cathode



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



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

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

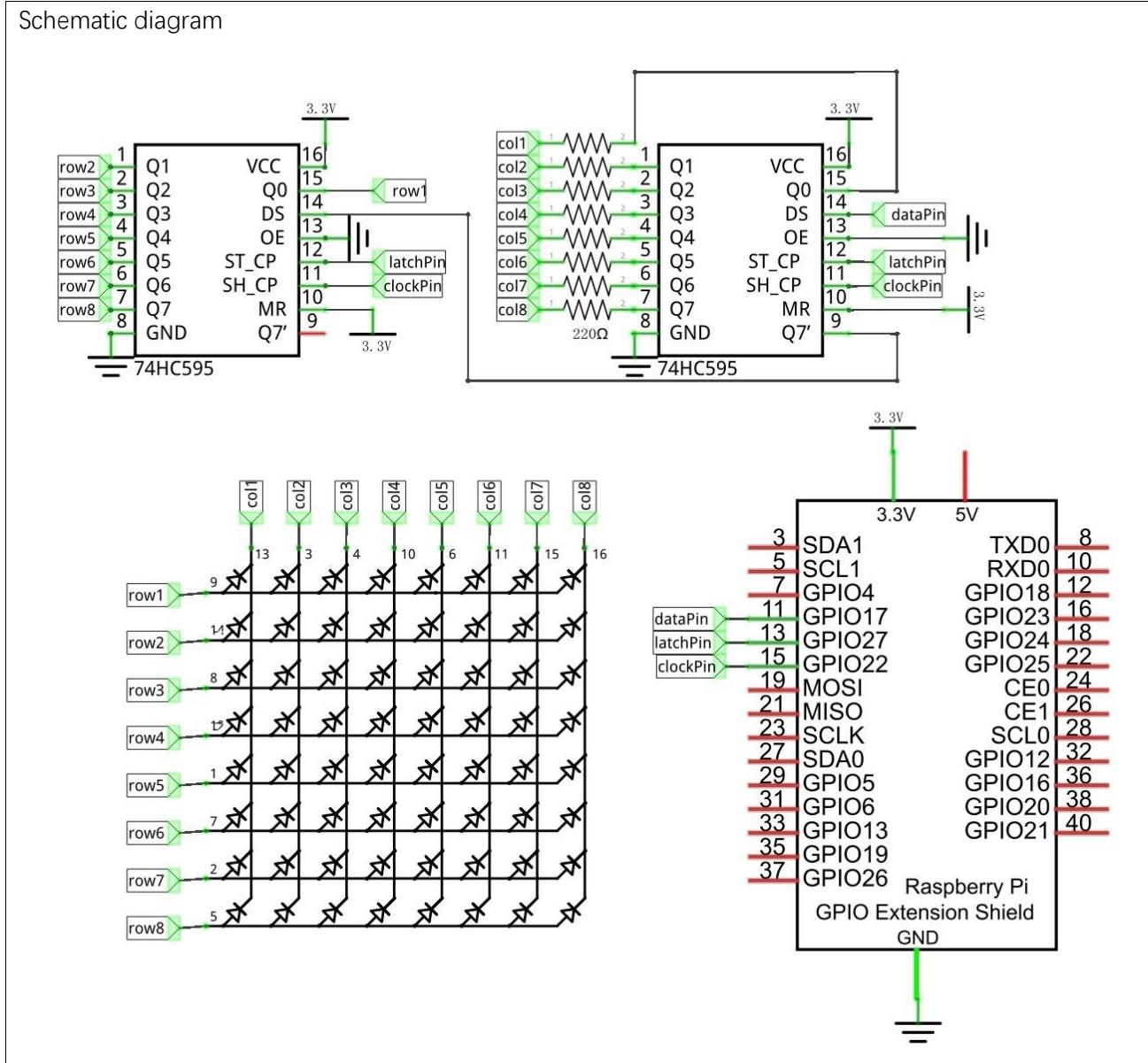
To begin, display the first column, then turn off the first column and display the second column. (and so on) turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated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

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

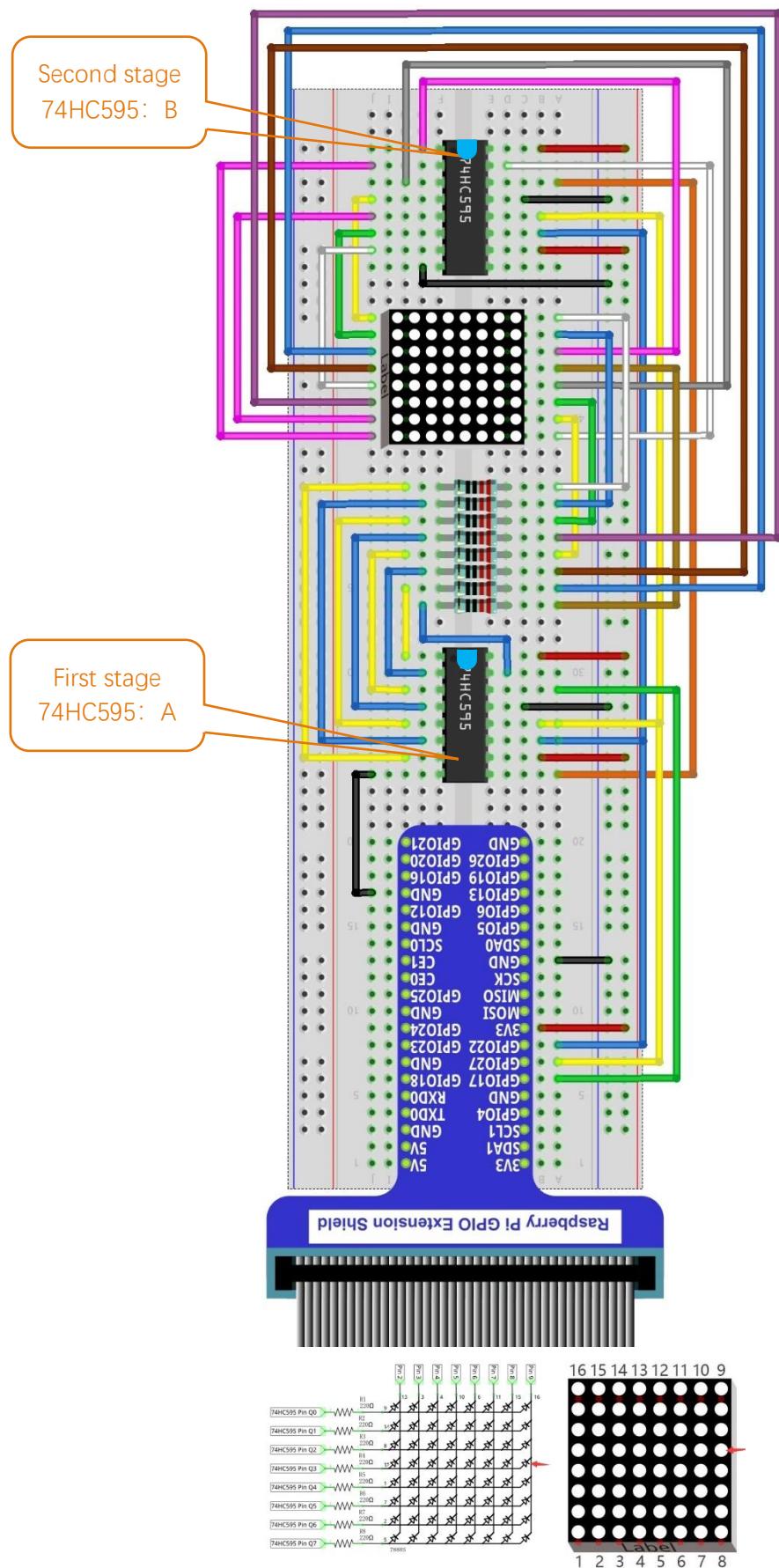
Circuit

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

Schematic diagram



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



Code

Two 74HC595 IC Chips are used in this project, one for controlling the LED Matrix's columns and the other for controlling the rows. According to the circuit connection, row data should be sent first, then column data. The following code will make the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

C Code 19.1.1 LEDMatrix

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 19.1.1_LEDMatrix directory of C language.

```
cd ~/Freenove_Kit/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, the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```

1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <wiringShift.h>
4
5 #define  dataPin  0 //DS Pin of 74HC595(Pin14)
6 #define  latchPin 2 //ST_CP Pin of 74HC595(Pin12)
7 #define  clockPin 3 //SH_CP Pin of 74HC595(Pin11)
8 // data of smile face
9 unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
10 unsigned char data[]={ // data of "0-F"
11     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // "
12     0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
13     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
14     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
15     0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
16     0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
17     0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
18     0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
19     0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
20     0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
21     0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
22     0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
23     0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
24     0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
25     0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
```

```
26     0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
27     0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
28     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // "
29 };
30 void _shiftOut(int dPin,int cPin,int order,int val) {
31     int i;
32     for(i = 0; i < 8; i++) {
33         digitalWrite(cPin,LOW);
34         if(order == LSBFIRST) {
35             digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
36             delayMicroseconds(10);
37         }
38         else {//if(order == MSBFIRST) {
39             digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
40             delayMicroseconds(10);
41         }
42         digitalWrite(cPin,HIGH);
43         delayMicroseconds(10);
44     }
45 }
46 int main(void)
47 {
48     int i,j,k;
49     unsigned char x;
50
51     printf("Program is starting ... \n");
52
53     wiringPiSetup();
54
55     pinMode(dataPin,OUTPUT);
56     pinMode(latchPin,OUTPUT);
57     pinMode(clockPin,OUTPUT);
58     while(1) {
59         for(j=0;j<500;j++) { //Repeat enough times to display the smiling face a period of
60             time
61             x=0x80;
62             for(i=0;i<8;i++) {
63                 digitalWrite(latchPin,LOW);
64                 _shiftOut(dataPin,clockPin,MSBFIRST,pic[i]);// first shift data of line
65             information to the first stage 74HC959
66                 _shiftOut(dataPin,clockPin,MSBFIRST,~x); //then shift data of column
67             information to the second stage 74HC959
68
69             digitalWrite(latchPin,HIGH); //Output data of two stage 74HC959 at the same
```

```

70     time
71             x>>=1;    //display the next column
72             delay(1);
73         }
74     }
75     for(k=0;k<sizeof(data)-8;k++) { //sizeof(data) total number of "0-F" columns
76         for(j=0;j<20;j++) { //times of repeated displaying LEDMatrix in every frame, the
77             bigger the "j" , the longer the display time
78                 x=0x80;           //Set the column information to start from the first column
79                 for(i=k;i<8+k;i++) {
80                     digitalWrite(latchPin,LOW);
81                     _shiftOut(dataPin,clockPin,MSBFIRST,data[i]);
82                     _shiftOut(dataPin,clockPin,MSBFIRST,~x);
83                     digitalWrite(latchPin,HIGH);
84                     x>>=1;
85                     delay(1);
86                 }
87             }
88         }
89     }
90 }
91     return 0;
92 }
```

The first “for” loop in the “while” loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```

of time
for(j=0;j<500;j++) { // Repeat enough times to display the smiling face a period
    x=0x80;
    for(i=0;i<8;i++) {
        digitalWrite(latchPin,LOW);
        shiftOut(dataPin,clockPin,MSBFIRST,pic[i]);
        shiftOut(dataPin,clockPin,MSBFIRST,~x);
        digitalWrite(latchPin,HIGH);
        x>>=1;
        delay(1);
    }
}
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of $18 \times 8 = 144$ columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column..... and so on...138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

```
for(k=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, MSBFIRST, data[i]);
            shiftOut(dataPin, clockPin, MSBFIRST, ~x);
            digitalWrite(latchPin, HIGH);
            x>>=1;
            delay(1);
        }
    }
}
```

Python Code 19.1.1 LEDMatrix

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 19.1.1_LEDMatrix directory of Python language.

```
cd ~/Freenove_Kit/Code/Python_Code/19.1.1_LEDMatrix
```

2. Use Python command to execute Python code "LEDMatrix.py".

```
python LEDMatrix.py
```

After the program is executed, the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```

1 import RPi.GPIO as GPIO
2
3
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 pic = [0x1c, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1c]# data of smiling face
11 data = [#data of "0-F"
12     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # ""
13     0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
14     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
15     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
16     0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
17     0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
18     0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
19     0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
20     0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
21     0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
22     0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
23     0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
24     0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
25     0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
26     0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
27     0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
28     0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
29     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # ""
30 ]
31 def setup():
32     GPIO.setmode(GPIO.BOARD)      # Number GPIOs by its physical location
33     GPIO.setup(dataPin, GPIO.OUT)
34     GPIO.setup(latchPin, GPIO.OUT)
```

```
35     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,MSBFIRST,pic[i]) #first shift data of line
54                 information to first stage 74HC959
55
56                     shiftOut(dataPin,clockPin,MSBFIRST,~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,MSBFIRST,data[i])
69                         shiftOut(dataPin,clockPin,MSBFIRST,~x)
70                         GPIO.output(latchPin,GPIO.HIGH)
71                         time.sleep(0.001)
72                         x>>=1
73     def destroy():
74         GPIO.cleanup()
75     if __name__ == '__main__':
76         print ('Program is starting... ')
77         setup()
78         try:
```

```

79     loop()
80     except KeyboardInterrupt:
81         destroy()

```

The first "for" loop in the "while" loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```

    for j in range(0,500):# Repeat enough times to display the smiling face a period
    of time
        x=0x80
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,pic[i])#first shift data of line
            information to first stage 74HC959
            shiftOut(dataPin,clockPin,MSBFIRST,~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" loop is used to display scrolling characters "0 to F", for a total of $18 \times 8 = 144$ columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column..... and so on...138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

```

    for k in range(0,len(data)-8):#len(data) total number of “0-F” columns.
        for j in range(0,20):# times of repeated displaying LEDMatrix in every frame,
        the bigger the “j”, the longer the display time
            x=0x80      # Set the column information to start from the first column
            for i in range(k,k+8):
                GPIO.output(latchPin,GPIO.LOW)
                shiftOut(dataPin,clockPin,MSBFIRST,data[i])
                shiftOut(dataPin,clockPin,MSBFIRST,~x)
                GPIO.output(latchPin,GPIO.HIGH)
                time.sleep(0.001)
            x>>=1

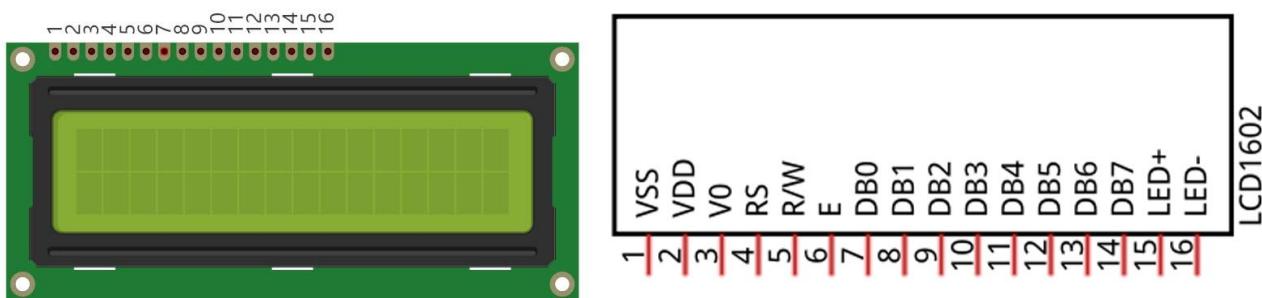
```

Chapter 20 LCD1602

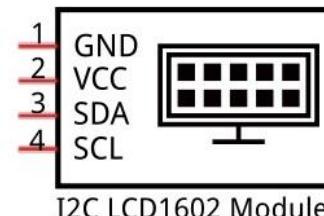
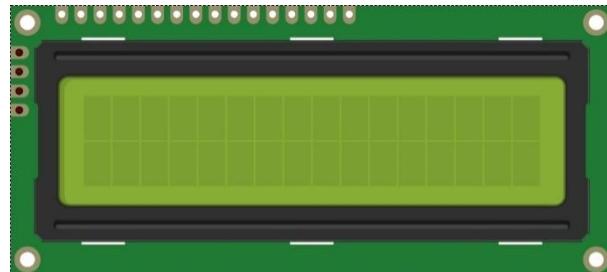
In this chapter, we will learn about the LCD1602 Display Screen,

Project 20.1 I2C LCD1602

There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram

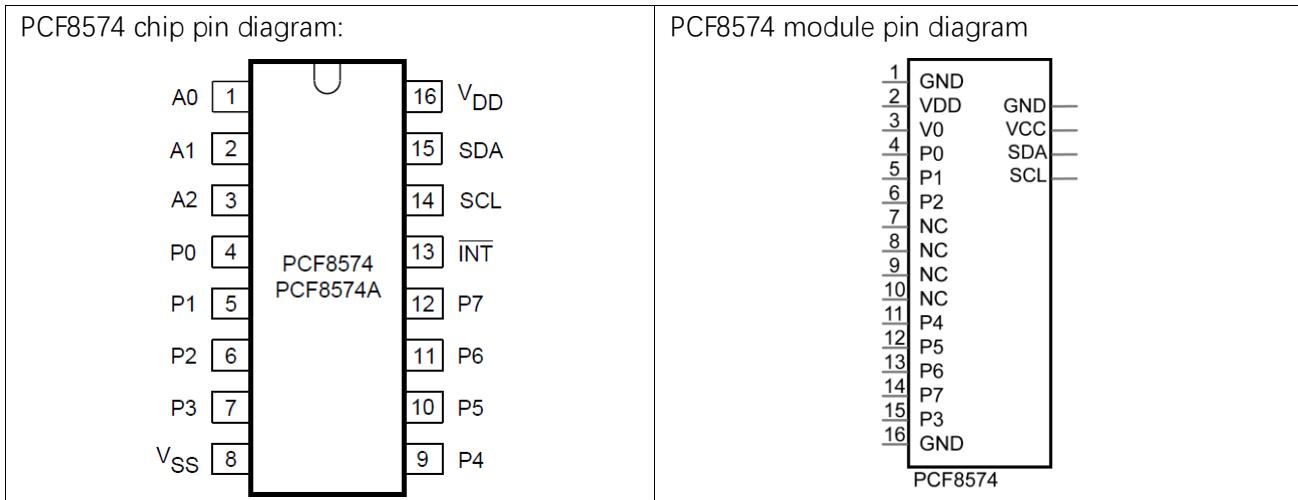


I2C LCD1602 Display Screen integrates a I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to only use 4 lines to operate the LCD1602.

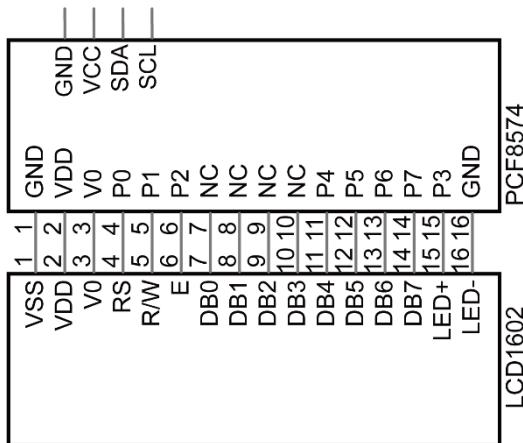


The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the RPI bus on your I2C device address through command "i2cdetect -y 1" (refer to the "configuration I2C" section below).

Below is the PCF8574 chip pin diagram and its module pin diagram:



PCF8574 module pins and LCD1602 pins correspond to each other and connected to each other:



Because of this, as stated earlier, we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface.

In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

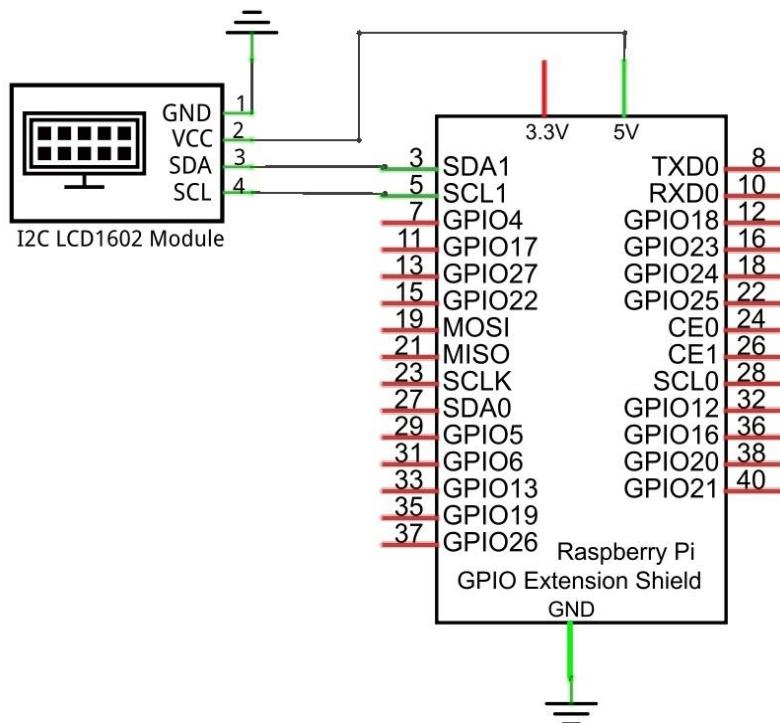
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x4
I2C LCD1602 Module x1	

Circuit

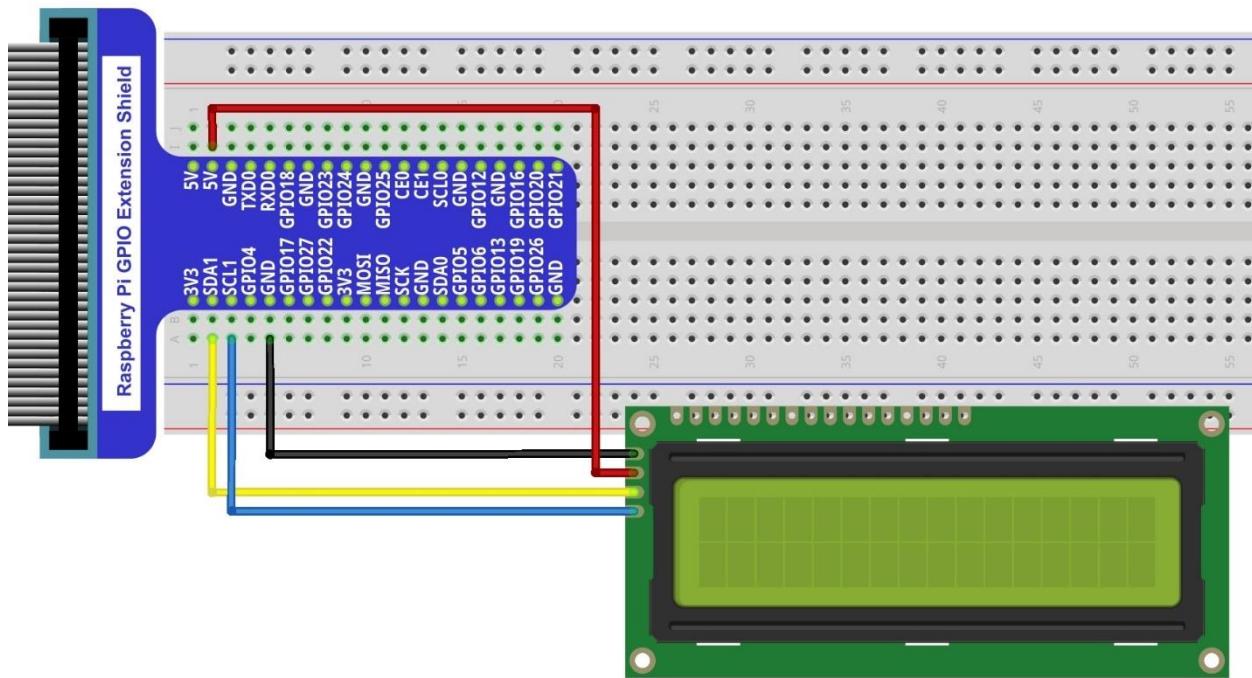
Note that the power supply for I2C LCD1602 in this circuit is 5V.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

NOTE: It is necessary to configure I2C and install Smbus first (see [chapter 7](#) for details)



Code

This code will have your RPi's CPU temperature and System Time Displayed on the LCD1602.

C Code 20.1.1 I2CLCD1602

If you did not [configure I2C and install Smbus](#), please refer to [Chapter 7](#). If you did, please continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 20.1.1_I2CLCD1602 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/20.1.1_I2CLCD1602
```

2. Use following command to compile "I2CLCD1602.c" and generate executable file "I2CLCD1602".

```
gcc I2CLCD1602.c -o I2CLCD1602 -lwiringPi -lwiringPiDev
```

3. Then run the generated file "I2CLCD1602".

```
sudo ./I2CLCD1602
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program is executed, if you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display the Time and Temperature clearly.



The following is the program code:

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <wiringPi.h>
4 #include <wiringPiI2C.h>
5 #include <pcf8574.h>
6 #include <lcd.h>
7 #include <time.h>
8
9 int pcf8574_address = 0x27;           // PCF8574T:0x27, PCF8574AT:0x3F
10 #define BASE 64                  // BASE any number above 64
11 //Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
12 #define RS      BASE+0
13 #define RW      BASE+1
14 #define EN      BASE+2
15 #define LED     BASE+3
16 #define D4      BASE+4

```

```
17 #define D5      BASE+5
18 #define D6      BASE+6
19 #define D7      BASE+7
20
21 int lcdhd;// used to handle LCD
22 void printCPUtemperature() {// sub function used to print CPU temperature
23     FILE *fp;
24     char str_temp[15];
25     float CPU_temp;
26     // CPU temperature data is stored in this directory.
27     fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
28     fgets(str_temp,15,fp);      // read file temp
29     CPU_temp = atof(str_temp)/1000.0;    // convert to Celsius degrees
30     printf("CPU's temperature : %.2f \n",CPU_temp);
31     lcdPosition(lcdhd,0,0);      // set the LCD cursor position to (0,0)
32     lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp); // Display CPU temperature on LCD
33     fclose(fp);
34 }
35 void printDataTime() //used to print system time
36 {
37     time_t rawtime;
38     struct tm *timeinfo;
39     time(&rawtime); // get system time
40     timeinfo = localtime(&rawtime); //convert to local time
41     printf("%s \n",asctime(timeinfo));
42     lcdPosition(lcdhd,0,1); // set the LCD cursor position to (0,1)
43     lcdPrintf(lcdhd,"Time:%02d:%02d:%02d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
44     //Display system time on LCD
45 }
46 int detectI2C(int addr){ //Used to detect i2c address of LCD
47     int _fd = wiringPiI2CSetup (addr);
48     if (_fd < 0){
49         printf("Error address : 0x%x \n",addr);
50         return 0 ;
51     }
52     else{
53         if(wiringPiI2CWrite(_fd,0) < 0){
54             printf("Not found device in address 0x%x \n",addr);
55             return 0;
56         }
57         else{
58             printf("Found device in address 0x%x \n",addr);
59             return 1 ;
60         }
61 }
```

```
61     }
62 }
63 int main(void) {
64     int i;
65     printf("Program is starting ... \n");
66     wiringPiSetup();
67     if(detectI2C(0x27)) {
68         pcf8574_address = 0x27;
69     } else if(detectI2C(0x3F)) {
70         pcf8574_address = 0x3F;
71     } else{
72         printf("No correct I2C address found, \n"
73             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
74             "Program Exit. \n");
75     }
76     pcf8574Setup(BASE, pcf8574_address); //initialize PCF8574
77     for(i=0;i<8;i++) {
78         pinMode(BASE+i, OUTPUT);      //set PCF8574 port to output mode
79     }
80     digitalWrite(LED, HIGH);       //turn on LCD backlight
81     digitalWrite(RW, LOW);        //allow writing to LCD
82     lcdhd = lcdInit(2, 16, 4, RS, EN, D4, D5, D6, D7, 0, 0, 0, 0); // initialize LCD and return "handle"
83 used to handle LCD
84     if(lcdhd == -1) {
85         printf("lcdInit failed !");
86         return 1;
87     }
88     while(1) {
89         printCPUTemperature(); //print CPU temperature
90         printDataTime();      // print system time
91         delay(1000);
92     }
93     return 0;
94 }
95 }
```

From the code, we can see that the PCF8591 and the PCF8574 have many similarities in using the I2C interface to expand the GPIO RPI.

First, define the I2C address of the PCF8574 and the Extension of the GPIO pin, which is connected to the

GPIO pin of the LCD1602. LCD1602 has two different i2c addresses. Set 0x27 as default.

```
int pcf8574_address = 0x27;      // PCF8574T:0x27, PCF8574AT:0x3F
#define BASE 64          // BASE any number above 64
//Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
#define RS    BASE+0
#define RW    BASE+1
#define EN    BASE+2
#define LED   BASE+3
#define D4    BASE+4
#define D5    BASE+5
#define D6    BASE+6
#define D7    BASE+7
```

Then, in main function, initialize the PCF8574, set all the pins to output mode, and turn ON the LCD1602 backlight (without the backlight the Display is difficult to read).

```
pcf8574Setup(BASE, pcf8574_address); // initialize PCF8574
for(i=0;i<8;i++) {
    pinMode(BASE+i, OUTPUT); // set PCF8574 port to output mode
}
digitalWrite(LED, HIGH); // turn on LCD backlight
```

Then use lcdInit() to initialize LCD1602 and set the RW pin of LCD1602 to 0 (can be written) according to requirements of this function. The return value of the function called "Handle" is used to handle LCD1602".

```
lcdhd = lcdInit(2, 16, 4, RS, EN, D4, D5, D6, D7, 0, 0, 0, 0); // initialize LCD and return
"handle" used to handle LCD
```

Details about lcdInit():

```
int lcdInit (int rows, int cols, int bits, int rs, int strb,
            int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7);
```

This is the main initialization function and must be executed first before you use any other LCD functions.

Rows and **cols** are the rows and columns of the Display (e.g. 2, 16 or 4, 20). **Bits** is the number of how wide the number of bits is on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the Display's RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the RPi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault (usually incorrect parameter)

For more details about LCD Library, please refer to: <https://projects.drogon.net/raspberry-pi/wiringpi/lcd-library/>

In the next "while", two subfunctions are called to display the RPi's CPU Temperature and the SystemTime. First look at subfunction printCPUtemperature(). The CPU temperature data is stored in the "/sys/class/thermal/thermal_zone0/temp" file. We need to read the contents of this file, which converts it to temperature value stored in variable CPU_temp and uses lcdPrintf() to display it on LCD.

```
void printCPUtemperature() { //subfunction used to print CPU temperature
    FILE *fp;
    char str_temp[15];
```

```

float CPU_temp;
// CPU temperature data is stored in this directory.
fp=fopen("/sys/class/thermal/thermal_zone0/temp", "r");
fgets(str_temp, 15, fp); // read file temp
CPU_temp = atof(str_temp)/1000.0; // convert to Celsius degrees
printf("CPU's temperature : %.2f \n",CPU_temp);
lcdPosition(lcdhd, 0, 0); // set the LCD cursor position to (0,0)
lcdPrintf(lcdhd, "CPU:%.2fC", CPU_temp); // Display CPU temperature on LCD
fclose(fp);
}

```

Details about `lcdPosition()` and `lcdPrintf()`:

lcdPosition (int handle, int x, int y);

Set the position of the cursor for subsequent text entry.

lcdPutchar (int handle, uint8_t data)

lcdPuts (int handle, char *string)

lcdPrintf (int handle, char *message, …)

These output a single ASCII character, a string or a formatted string using the usual print formatting commands to display individual characters (it is how you are able to see characters on your computer monitor).

Next is subfunction `printDataTime()` used to display System Time. First, it gets the Standard Time and stores it into variable `Rawtime`, and then converts it to the Local Time and stores it into `timeinfo`, and finally displays the Time information on the LCD1602 Display.

```

void printDataTime() //used to print system time
{
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime); // get system time
    timeinfo = localtime(&rawtime); // convert to local time
    printf("%s \n", asctime(timeinfo));
    lcdPosition(lcdhd, 0, 1); // set the LCD cursor position to (0,1)
    lcdPrintf(lcdhd, "Time:%d:%d:%d", timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec);
    //Display system time on LCD
}

```

Python Code 20.1.1 I2CLCD1602

If you did not [configure I2C and install Smbus](#), please refer to [Chapter 7](#). If you did, continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 20.1.1_I2CLCD1602 directory of Python code.

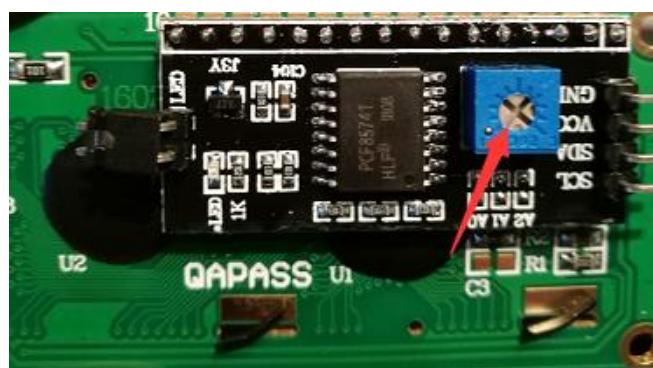
```
cd ~/Freenove_Kit/Code/Python_Code/20.1.1_I2CLCD1602
```

2. Use Python command to execute Python code "I2CLCD1602.py".

```
python I2CLCD1602.py
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program is executed, if you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display the Time and Temperature clearly.



The following is the program code:

```

1  from PCF8574 import PCF8574_GPIO
2  from Adafruit_LCD1602 import Adafruit_CharLCD
3
4  from time import sleep, strftime
5  from datetime import datetime
6
7  def get_cpu_temp():      # get CPU temperature and store it into file
8      "/sys/class/thermal/thermal_zone0/temp"
9      tmp = open('/sys/class/thermal/thermal_zone0/temp')
10     cpu = tmp.read()
11     tmp.close()
12     return '{:.2f}'.format(float(cpu)/1000) + ' C'
13
14 def get_time_now():      # get system time
15     return datetime.now().strftime('%H:%M:%S')
16
17 def loop():
18     mcp.output(3, 1)      # turn on LCD backlight
19     lcd.begin(16, 2)      # set number of LCD lines and columns
20     while(True):
21         lcd.clear()
```

```

22         lcd.setCursor(0, 0) # set cursor position
23         lcd.message( 'CPU: ' + get_cpu_temp()+' \n' )# display CPU temperature
24         lcd.message( get_time_now() ) # display the time
25         sleep(1)
26
27     def destroy():
28         lcd.clear()
29
30     PCF8574_address = 0x27 # I2C address of the PCF8574 chip.
31     PCF8574A_address = 0x3F # I2C address of the PCF8574A chip.
32     # Create PCF8574 GPIO adapter.
33     try:
34         mcp = PCF8574_GPIO(PCF8574_address)
35     except:
36         try:
37             mcp = PCF8574_GPIO(PCF8574A_address)
38         except:
39             print ('I2C Address Error !')
40             exit(1)
41     # Create LCD, passing in MCP GPIO adapter.
42     lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4, 5, 6, 7], GPIO=mcp)
43
44     if __name__ == '__main__':
45         print ('Program is starting ... ')
46         try:
47             loop()
48         except KeyboardInterrupt:
49             destroy()

```

Two modules are used in the code, PCF8574.py and Adafruit_LCD1602.py. These two documents and the code files are stored in the same directory, and neither of them is dispensable. Please DO NOT DELETE THEM! PCF8574.py is used to provide I2C communication mode and operation method of some of the ports for the RPi and PCF8574 IC Chip. Adafruit module Adafruit_LCD1602.py is used to provide some functional operation method for the LCD1602 Display.

In the code, first get the object used to operate the PCF8574's port, then get the object used to operate the LCD1602.

```

address = 0x27 # I2C address of the PCF8574 chip.
# Create PCF8574 GPIO adapter.
mcp = PCF8574_GPIO(address)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4, 5, 6, 7], GPIO=mcp)

```

According to the circuit connection, port 3 of PCF8574 is connected to the positive pole of the LCD1602 Display's backlight. Then in the loop () function, use of mcp.output (3,1) to turn the LCD1602 Display's backlight ON and then set the number of LCD lines and columns.

```
def loop():
    mcp.output(3, 1)      # turn on the LCD backlight
    lcd.begin(16, 2)      # set number of LCD lines and columns
```

In the next while loop, set the cursor position, and display the CPU temperature and time.

```
while(True):
    lcd.clear()
    lcd.setCursor(0, 0)  # set cursor position
    lcd.message('CPU: ' + get_cpu_temp()+'\n')# display CPU temperature
    lcd.message(get_time_now())  # display the time
    sleep(1)
```

CPU temperature is stored in file “/sys/class/thermal/thermal_zone0/temp”. Open the file and read content of the file, and then convert it to Celsius degrees and return. Subfunction used to get CPU temperature is shown below:

```
def get_cpu_temp():      # get CPU temperature and store it into file
    "/sys/class/thermal/thermal_zone0/temp"
    tmp = open('/sys/class/thermal/thermal_zone0/temp')
    cpu = tmp.read()
    tmp.close()
    return '{:.2f}'.format(float(cpu)/1000) + ' C'
```

Subfunction used to get time:

```
def get_time_now():      # get the time
    return datetime.now().strftime('%H:%M:%S')
```

Details about PCF8574.py and Adafruit_LCD1602.py:

Module PCF8574

This module provides two classes **PCF8574_I2C** and **PCF8574_GPIO**.

Class **PCF8574_I2C**: provides reading and writing method for PCF8574.

Class **PCF8574_GPIO**: provides a standardized set of GPIO functions.

More information can be viewed through opening PCF8574.py.

Adafruit_LCD1602 Module

Module Adafruit_LCD1602

This module provides the basic operation method of LCD1602, including class Adafruit_CharLCD. Some member functions are described as follows:

def begin(self, cols, lines): set the number of lines and columns of the screen.

def clear(self): clear the screen

def setCursor(self, col, row): set the cursor position

def message(self, text): display contents

More information can be viewed through opening Adafruit_CharLCD.py.

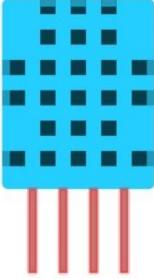
Chapter 21 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

Project 21.1 Hygrothermograph

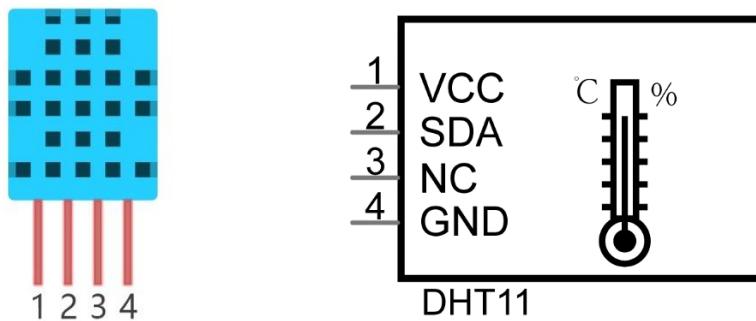
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the RPi to read Temperature and Humidity data of the DHT11 Module.

Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	DHT11 x1	Resistor 10kΩ x1
Jumper Wire x4		

Component knowledge

The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.

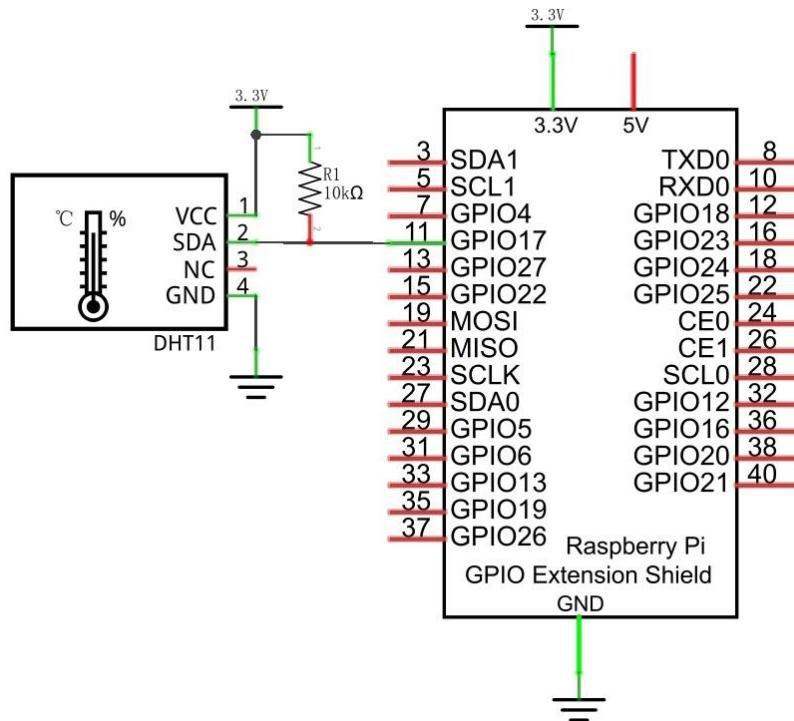


After being powered up, it will initialize in 1 second. Its operating voltage is within the range of 3.3V-5.5V. The SDA pin is a data pin, which is used to communicate with other devices.

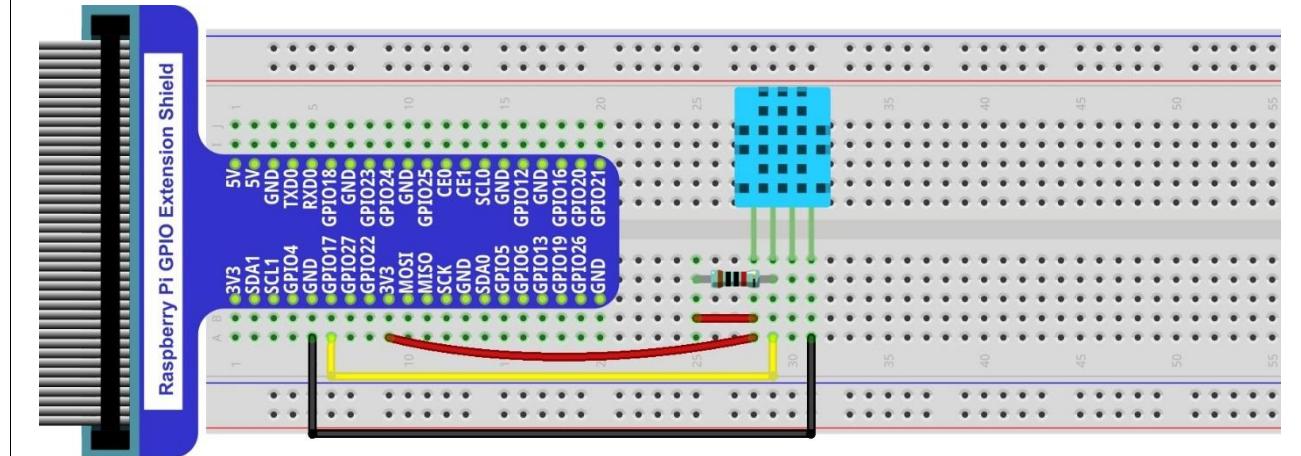
The NC pin (Not Connected Pin) are a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins **should not be connected** to any of the circuit connections.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

The code is used to read the temperature and humidity data of DHT11, and display them.

C Code 21.1.1 DHT11

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 21.1.1_DHT11 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/21.1.1_DHT11
```

2. The code used in this project contains a custom header file. Use the following command to compile the code DHT11.cpp and DHT.cpp and generate executable file DHT11. 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 read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts : 1
DHT11,OK!
Humidity is 50.00 %,      Temperature is 27.50 *C

Measurement counts : 2
DHT11,OK!
Humidity is 53.00 %,      Temperature is 27.50 *C

Measurement counts : 3
DHT11,OK!
Humidity is 54.00 %,      Temperature is 27.50 *C

Measurement counts : 4
DHT11,OK!
Humidity is 54.00 %,      Temperature is 27.50 *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, counts;   //chk:read the return value of sensor; sumCnt:times of reading
11    sensor
12
13    printf("Program is starting ... \n");
14}
```

```

15     while (1) {
16         counts++; //counting number of reading times
17         printf("Measurement counts : %d \n", counts);
18         for (int i = 0; i < 15; i++) {
19             chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
20             if(chk == DHTLIB_OK) {
21                 printf("DHT11,OK! \n");
22                 break;
23             }
24             delay(100);
25         }
26         printf("Humidity is %.2f %%,\t Temperature is %.2f *C\n\n", dht.humidity,
dht.temperature);
27         delay(2000);
28     }
29     return 1;
30 }
31 }
32 }
```

In this project code, we use a custom library file "DHT.hpp". It is located in the same directory with the 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, we create a DHT class object in the code.

```
DHT dht;
```

Then in the "while" loop, use `chk = dht.readDHT11(DHT11_Pin)` to read the DHT11, and determine whether the data read is normal according to the return value "chk". If the value is OK, end for loop and move on. Otherwise, try 15 times in total. Then use variable counts to record number of times to read.

```

while (1) {
    counts++; //counting number of reading times
    printf("Measurement counts : %d \n", counts);
    for (int i = 0; i < 15; i++) {
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
        if(chk == DHTLIB_OK) {
            printf("DHT11,OK! \n");
            break;
        }
        delay(100);
    }
    printf("Humidity is %.2f %%,\t Temperature is %.2f *C\n\n", dht.humidity,
dht.temperature);
    delay(2000);
}
```

Finally display 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 this public member function 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 #define _DHT_H_
2
3 #include <wiringPi.h>
4 #include <stdio.h>
5 #include <stdint.h>
6
7 //read return flag of sensor
8 #define DHTLIB_OK          0
9 #define DHTLIB_ERROR_CHECKSUM -1
10 #define DHTLIB_ERROR_TIMEOUT -2
11 #define DHTLIB_INVALID_VALUE -999
12
13 #define DHTLIB_DHT11_WAKEUP 20
14 #define DHTLIB_DHT_WAKEUP   1
15
16 #define DHTLIB_TIMEOUT      100
17
18 class DHT{
19     public:
20         DHT();
21         double humidity,temperature; //use to store temperature and humidity data read
22         int readDHT11Once(int pin); //read DHT11
23         int readDHT11(int pin); //read DHT11
24     private:
25         uint8_t bits[5]; //Buffer to receiver data
26         int readSensor(int pin,int wakeupDelay); //
27 };
```

Python Code 21.1.1 DHT11

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 21.1.1_DHT11 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/21.1.1_DHT11
```

2. Use Python command to execute code "DHT11.py".

```
python DHT11.py
```

After the program is executed, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts: 2
DHT11,OK!
Humidity : 53.00,           Temperature : 27.60

Measurement counts: 3
DHT11,OK!
Humidity : 53.00,           Temperature : 27.50

Measurement counts: 4
DHT11,OK!
Humidity : 53.00,           Temperature : 27.50

Measurement counts: 5
DHT11,OK!
Humidity : 52.00,           Temperature : 27.50
```

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     counts = 0 # Measurement counts
9     while(True):
10         counts += 1
11         print("Measurement counts: ", counts)
12         for i in range(0,15):
13             chk = dht.readDHT11()      #read DHT11 and get a return value. Then determine
14             whether data read is normal according to the return value.
15             if (chk is dht.DHTLIB_OK):    #read DHT11 and get a return value. Then determine
16             whether data read is normal according to the return value.
17                 print("DHT11,OK!")
18                 break
19             time.sleep(0.1)
20             print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
21             time.sleep(2)
22
```

```

23 if __name__ == '__main__':
24     print ('Program is starting ... ')
25     try:
26         loop()
27     except KeyboardInterrupt:
28         GPIO.cleanup()
29         exit()

```

In this project code, we use a module "**Freenove_DHT.py**", which provides the method of reading the DHT Sensor. It is located in the same directory with program files "**DHT11.py**". By using this library, we can easily read the DHT Sensor. First, we create a DHT class object in the code.

```
dht = DHT.DHT(DHTPin)    #create a DHT class object
```

Then in the "while" loop, use `chk = dht.readDHT11(DHT11Pin)` to read the DHT11, and determine whether the data read is normal according to the return value "chk". Then use variable sumCnt to record the number of times read.

```

while(True):
    counts += 1
    print("Measurement counts: ", counts)
    for i in range(0,15):
        chk = dht.readDHT11()      #read DHT11 and get a return value. Then determine
        whether data read is normal according to the return value.
        if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
        whether data read is normal according to the return value.
            print("DHT11, OK!")
            break
        time.sleep(0.1)
    print("Humidity : %.2f, \t Temperature : %.2f \n%(dht.humidity, dht.temperature))
    time.sleep(2)

```

Finally display the results:

```
print("Humidity : %.2f, \t Temperature : %.2f \n%(dht.humidity, dht.temperature))
```

Module "**Freenove_DHT.py**" contains a DHT class. The class function of the def **readDHT11** (pin) is used to read the DHT11 Sensor and store the temperature and humidity data read to member variables humidity and temperature.

Freenove_DHT Module

This is a Python module for reading the temperature and humidity data of the DHT Sensor. Partial functions and variables are described as follows:

Variable **humidity**: store humidity data read from sensor

Variable **temperature**: store temperature data read from sensor

def readDHT11 (pin): read the temperature and humidity of sensor DHT11, and return values used to determine whether the data is normal.

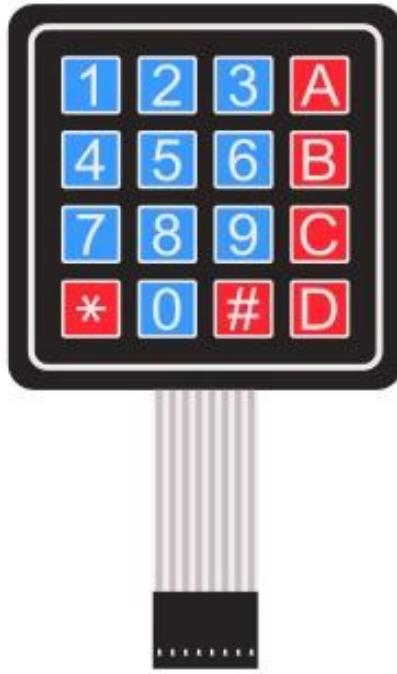
Chapter 22 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 22.1 Matrix Keypad

In this project, we will attempt to get every key code on the Matrix Keypad to work.

Component List

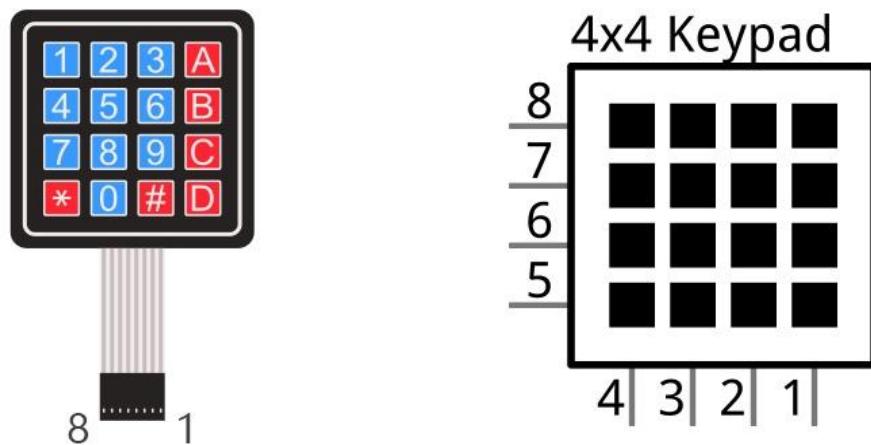
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Wire x1 Breadboard x1	4x4 Matrix Keypad x1
Jumper wire	
Resistor 10kΩ x4	

Component knowledge

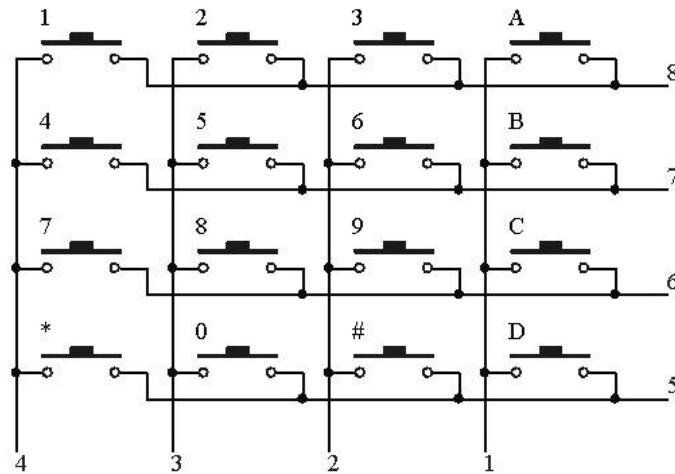
4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad

Matrix integrates 16 keys (think of this as 16 Push Button Switches in one module):

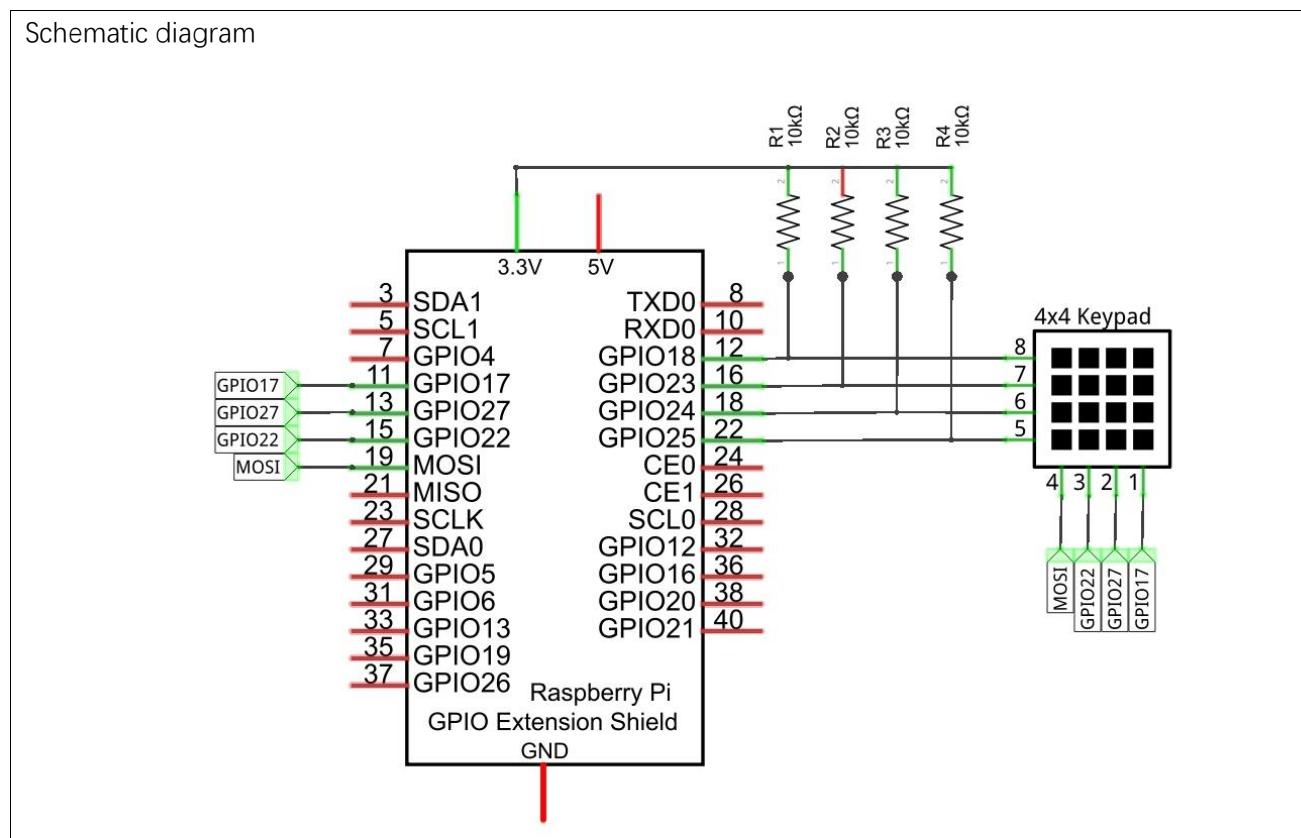


Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.

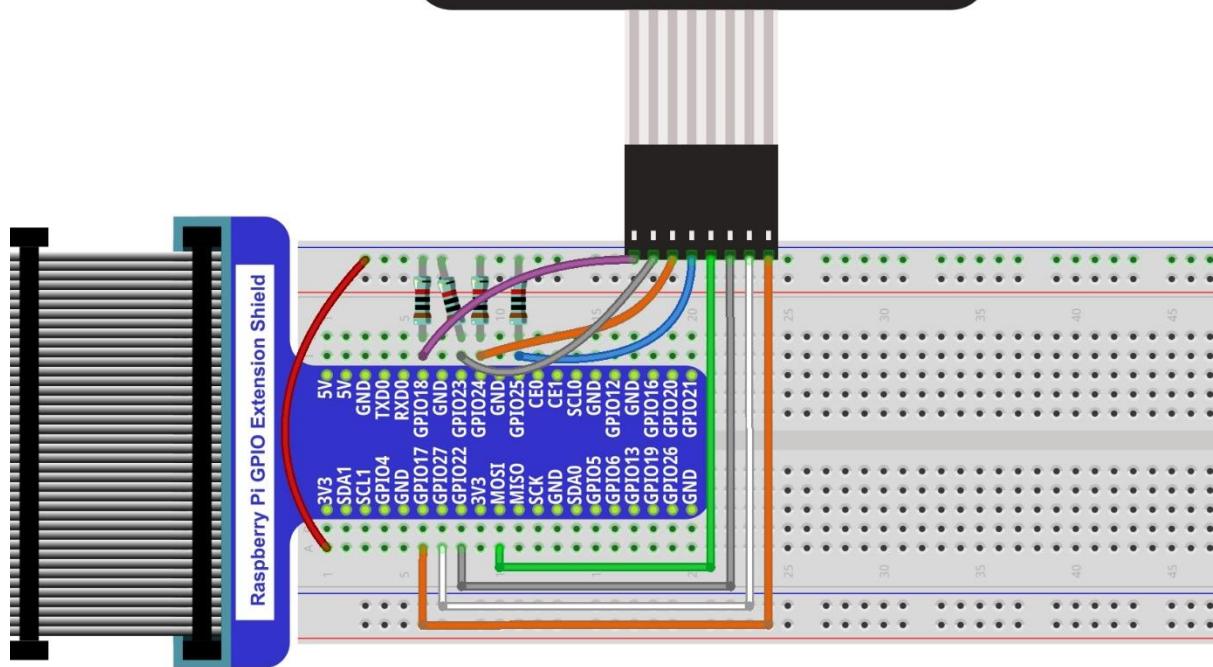
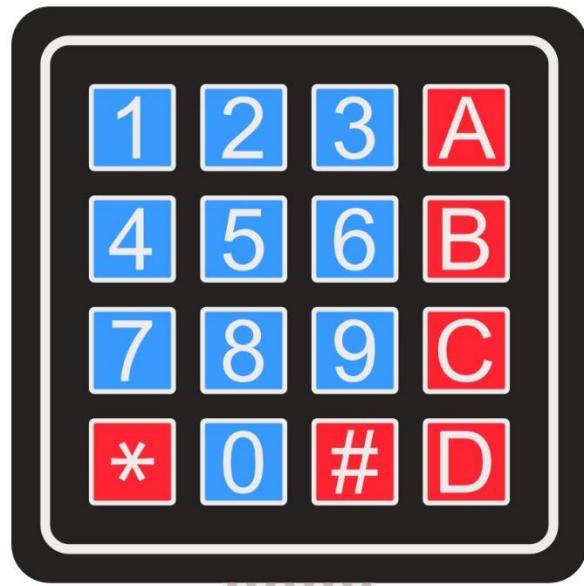


The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Therefore, you can get the state of all of the keys.

Circuit



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be displayed in the terminal window.

C Code 22.1.1 MatrixKeypad

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 22.1.1_MatrixKeypad directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/22.1.1_MatrixKeypad
```

2. Code of this project contains a custom header file. Use the following command to compile the code MatrixKeypad.cpp, Keypad.cpp and Key.cpp generate executable file MatrixKeypad. 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, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:

```
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:

```
1 #include "Keypad.hpp"
2 #include <stdio.h>
3 const byte ROWS = 4; //four rows
4 const byte COLS = 4; //four columns
5 char keys[ROWS][COLS] = { //key code
6     {'1','2','3','A'},
7     {'4','5','6','B'},
8     {'7','8','9','C'},
9     {'*','0','#','D'}
10 };
11 byte rowPins[ROWS] = {1, 4, 5, 6 }; //define the row pins for the keypad
12 byte colPins[COLS] = {12,3, 2, 0 }; //define the column pins for the keypad
```

```

13 //create Keypad object
14 Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
15
16 int main() {
17     printf("Program is starting ... \n");
18
19     wiringPiSetup();
20
21     char key = 0;
22     keypad.setDebounceTime(50);
23     while(1) {
24         key = keypad.getKey(); //get the state of keys
25         if (key){ //if a key is pressed, print out its key code
26             printf("You Pressed key : %c \n",key);
27         }
28     }
29     return 1;
30 }
```

In this project 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**". The Library Keypad is "transplanted" from the Arduino Library Keypad. This library file provides a method to read the Matrix Keyboard's input. By using this library, we can easily read the pressed keys of the Matrix Keyboard.

First, we define the information of the Matrix Keyboard used in this project: the number of rows and columns, code designation of each key and GPIO pin connected to each column and 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
```

Then, based on the above information, initiates 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 characteristics of the keyboard's flexibility, with a default time of 10ms.

```
keypad.setDebounceTime(50);
```

In the "while" loop, use the function key= keypad.getKey () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", then be displayed.

```
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 Keypad Library used for the RPi is transplanted from the Arduino Keypad Library. 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 whether 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".

Python Code 22.1.1 MatrixKeypad

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 22.1.1_MatrixKeypad directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/22.1.1_MatrixKeypad
```

2. Use Python command to execute code "MatrixKeypad.py".

```
python MatrixKeypad.py
```

After the program is executed, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:

```
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:

```
1 import RPi.GPIO as GPIO
2 import Keypad      #import module Keypad
3 ROWS = 4          # number of rows of the Keypad
4 COLS = 4          #number of columns of the Keypad
5 keys = [    '1','2','3','A',
6                 '4','5','6','B',
7                 '7','8','9','C',
8                 '*', '0', '#', 'D'   ]
9 rowsPins = [12, 16, 18, 22]      #connect to the row pinouts of the keypad
10 colsPins = [19, 15, 13, 11]      #connect to the column pinouts of the keypad
11
12 def loop():
13     keypad = Keypad.Keypad(keys, rowsPins, colsPins, ROWS, COLS)      #creat Keypad object
14     keypad.setDebounceTime(50)      #set the debounce time
15     while(True):
16         key = keypad.getKey()      #obtain the state of keys
17         if(key != keypad.NULL):    #if there is key pressed, print its key code.
18             print ("You Pressed Key : %c "%(key))
19
20     if __name__ == '__main__':
21         print ("Program is starting ... ")
```

```

22     try:
23         loop()
24     except KeyboardInterrupt:
25         GPIO.cleanup()

```

In this project code, we use a custom module "**Keypad.py**", which is located in the same directory with program file "**MatrixKeypad.py**". And this library file, which is transplanted from Arduino function library Keypad, provides a method to read the keyboard. By using this library, we can easily read the matrix keyboard. First, import module Keypad. Then define the information of the matrix keyboard used in this project: the number of rows and columns, code of each key and GPIO pin connected to each column and each row.

```

import Keypad      #import module Keypad
ROWS = 4          #number of rows of the Keypad
COLS = 4          #number of columns of the Keypad
keys = [ '1', '2', '3', 'A',
         '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

```

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

```
keypad = Keypad.Keypad(keys, rowsPins, colsPins, ROWS, COLS)
```

Set the debounce time to 50ms, and this value can be set based on the actual characteristics of the keyboard's flexibility, with a default time of 10ms.

```
keypad.setDebounceTime(50)
```

In the "while" loop, use the function `key= keypad.getKey ()` to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", and then be displayed.

```

while(True):
    key = keypad.getKey()      #get the state of keys
    if(key != keypad.NULL):   # if a key is pressed, print out its key code
        print ("You Pressed Key : %c "%(key))

```

The Keypad Library used for the RPi is “transplanted” from the Arduino Keypad Library. The source files is written by language C++ and translated into Python can be obtained by visiting <http://playground.arduino.cc/Code/Keypad>. As for the “transplanted” function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad Library are described below:

```
class Keypad  
def __init__(self, usrKeyMap, row_Pins, col_Pins, num_Rows, num_Cols):  
    Constructed function, the parameters are: key code of keyboard, row pin, column pin, the number of rows,  
    the number of columns.  
def getKey(self):  
    Get a pressed key. If no key is pressed, the return value is keypad NULL.  
def setDebounceTime(self, ms):  
    Set the debounce time. And the default time is 10ms.  
def setHoldTime(self, ms):  
    Set the time when the key holds stable state after pressed.  
def isPressed(keyChar):  
    Judge whether the key with code "keyChar" is pressed.  
def waitForKey():  
    Wait for a key to be pressed, and return key code of the pressed key.  
def getState():  
    Get state of the keys.  
def keyStateChanged():  
    Judge whether there is a change of key state, then return True or False.
```

For More information about Keypad, please visit: <http://playground.arduino.cc/Code/Keypad> or through the opening file "Keypad.py".

Chapter 23 Ultrasonic Ranging

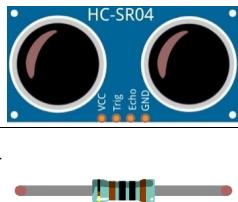
In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 23.1 Ultrasonic Ranging

In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

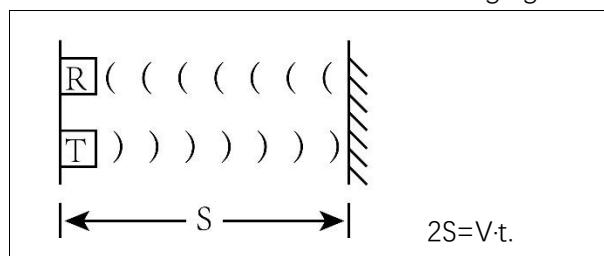
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	HC SR04 x1
Jumper Wire x4	Resistor 1kΩ x1

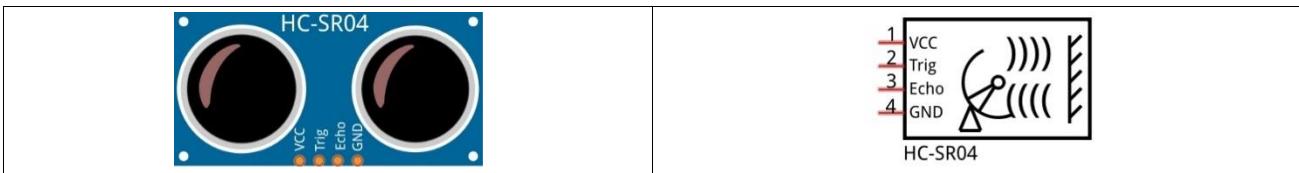


Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will be reflected when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about $v=340\text{m/s}$, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: $s=vt/2$.



The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

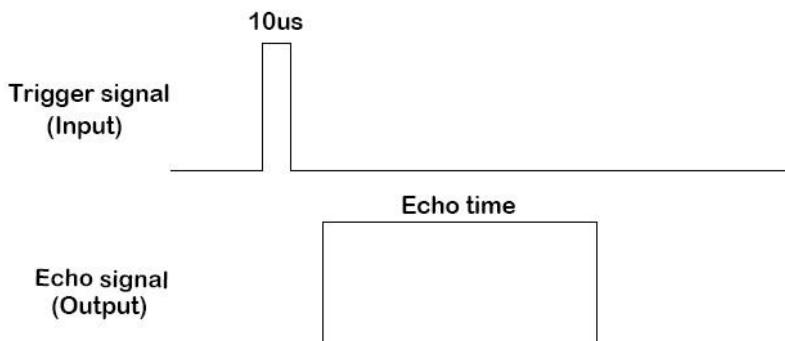
Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

Maximum measured distance: 200cm

Instructions for Use: output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, $s=vt/2$. This is done constantly.

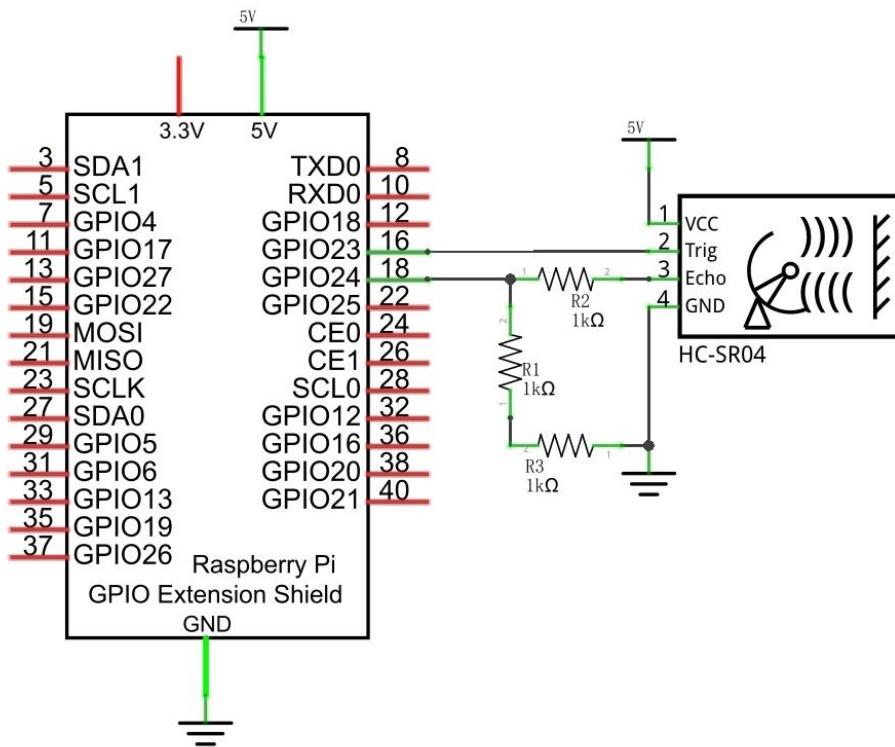


$$\text{Distance} = \text{Echo time} \times \text{sound velocity} / 2 .$$

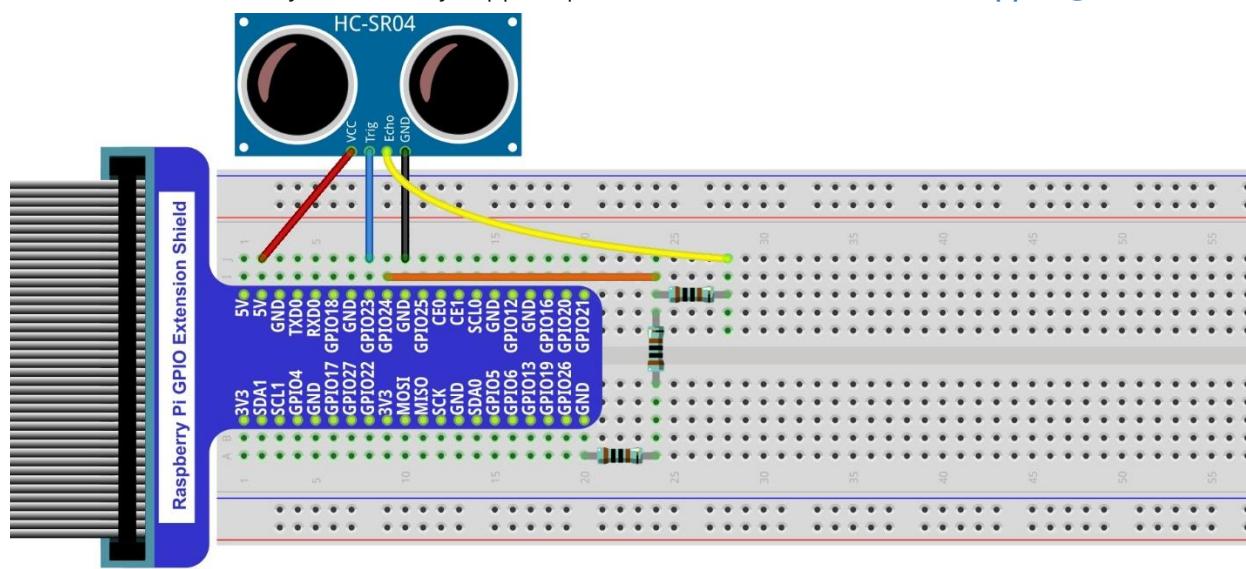
Circuit

Note that the voltage of ultrasonic module is 5V in this circuit.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

C Code 23.1.1 UltrasonicRanging

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 23.1.1_UltrasonicRanging directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/23.1.1_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, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.82 cm
The distance is : 198.37 cm
The distance is : 198.37 cm
The distance is : 199.63 cm
The distance is : 197.52 cm
The distance is : 198.39 cm
The distance is : 198.41 cm
```

The following is the program code:

```
1 #include <wiringPi.h>
2 #include <stdio.h>
3 #include <sys/time.h>
4
5 #define trigPin 4
6 #define echoPin 5
7 #define MAX_DISTANCE 220      // define the maximum measured distance
8 #define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured
9 distance
10 //function pulseIn: obtain pulse time of a pin
11 int pulseIn(int pin, int level, int timeout);
12 float getSonar() { //get the measurement result of ultrasonic module with unit: cm
13     long pingTime;
14     float distance;
15     digitalWrite(trigPin,HIGH); //send 10us high level to trigPin
16     delayMicroseconds(10);
17     digitalWrite(trigPin,LOW);
18     pingTime = pulseIn(echoPin,HIGH,timeOut); //read plus time of echoPin
19     distance = (float)pingTime * 340.0 / 2.0 / 10000.0; //calculate distance with sound speed
20     340m/s
21     return distance;
```

```

22 }
23
24 int main() {
25     printf("Program is starting ... \n");
26
27     wiringPiSetup();
28
29     float distance = 0;
30     pinMode(trigPin, OUTPUT);
31     pinMode(echoPin, INPUT);
32     while(1){
33         distance = getSonar();
34         printf("The distance is : %.2f cm\n",distance);
35         delay(1000);
36     }
37     return 1;
38 }
```

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 cannot wait for this forever, so we need to calculate the time period for the maximum distance, that is, time Out. **timeOut= 2*MAX_DISTANCE/100/340*1000000**. The result of the constant part in this formula is approximately 58.8.

```
#define timeOut MAX_DISTANCE*60
```

Subfunction **getSonar ()** function is used to start the Ultrasonic Module to begin measurements and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn ()** to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```

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;
}
```

Lastly, in the while loop of main function, get the measurement distance and display it continually.

```
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 23.1.1 UltrasonicRanging

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 23.1.1_UltrasonicRanging directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/23.1.1_UltrasonicRanging
```

2. Use Python command to execute code "UltrasonicRanging.py".

```
python UltrasonicRanging.py
```

After the program is executed, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.75 cm
The distance is : 199.22 cm
The distance is : 198.42 cm
The distance is : 198.74 cm
The distance is : 198.37 cm
The distance is : 198.47 cm
The distance is : 198.41 cm
```

The following is the program code:

```
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(cm)
7 timeOut = MAX_DISTANCE*60   #calculate timeout(μs) according to the maximum measured
8 distance
9
10 def pulseIn(pin, level, timeOut): # function pulseIn: obtain pulse time of a pin
11     t0 = time.time()
12     while(GPIO.input(pin) != level):
13         if((time.time() - t0) > timeOut*0.000001):
14             return 0;
15     t0 = time.time()
16     while(GPIO.input(pin) == level):
17         if((time.time() - t0) > timeOut*0.000001):
18             return 0;
19     pulseTime = (time.time() - t0)*1000000
20     return pulseTime
21
22 def getSonar():      #get the measurement results of ultrasonic module,with unit: cm
23     GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
24     time.sleep(0.00001)    #10us
25     GPIO.output(trigPin,GPIO.LOW)
26     pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)  #read plus time of echoPin
27
```

```

28     distance = pingTime * 340.0 / 2.0 / 10000.0      # the sound speed is 340m/s, and
29     calculate distance (cm)
30
31     return distance
32
33 def setup():
34     print ('Program is starting...')
35     GPIO.setmode(GPIO.BOARD)      #numbers GPIOs by physical location
36     GPIO.setup(trigPin, GPIO.OUT)  # set trigPin to output mode
37     GPIO.setup(echoPin, GPIO.IN)   # set echoPin to input mode
38
39 def loop():
40     while(True):
41         distance = getSonar()
42         print ("The distance is : %.2f cm"%(distance))
43         time.sleep(1)
44
45 if __name__ == '__main__':      #program start from here
46     setup()
47     try:
48         loop()
49     except KeyboardInterrupt:
50         GPIO.cleanup()

```

First, define the pins and the maximum measurement distance.

```

trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          # define the maximum measured distance 220cm

```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance (200cm). Then $\text{timOut} = 2 \times \text{MAX_DISTANCE} / 100 / 340 \times 1000000$. The result of the constant part in this formula is approximately 58.8.

```
timeOut = MAX_DISTANCE*60
```

Subfunction **getSonar ()** function is used to start the Ultrasonic Module to begin measurements, and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn ()** to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
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 display it continually.

```
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).

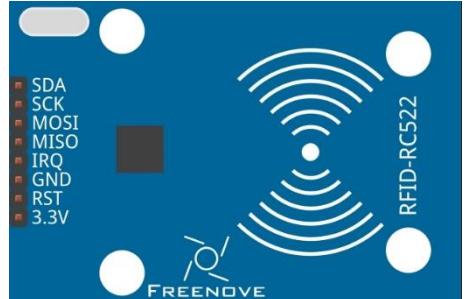
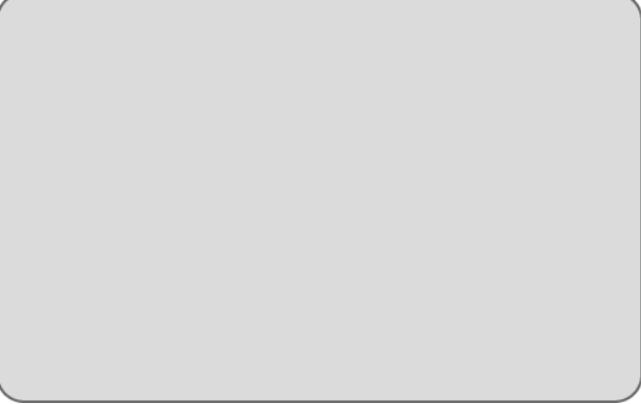
Chapter 24 RFID

In this chapter, we will learn how to use RFID.

Project 24.1 RFID

In this project, we will use RC522 RFID card reader to read and write the M1-S50 card.

Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 Breadboard x1 Jumper M/F x7	 A blue printed circuit board (PCB) labeled "RFID-RC522". It features several pins on the left labeled SDA, SCK, MOSI, MISO, IRQ, GND, RST, and 3.3V. On the right, there is a circular antenna coil and two white circular pads. The FREENOVE logo is visible at the bottom left.
Mifare1 S50 Standard card x1	 A light gray rectangular card, representing a standard Mifare1 S50 card.

Component Knowledge

RFID

RFID (Radio Frequency Identification) is a form of wireless communication technology. A complete RFID system is generally composed of a transponder and a reader. Generally, the transponder may be known as a tag, and each tag has a unique code, which is attached to an object to identify the target object. The reader is a device that reads (or writes) information in the tag.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products, among which, Passive RFID products are the earliest, the most mature and most widely used products in the market. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of them are classified as close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13~26mA (DC) \3. 3V
Idle current	10~13mA (DC) \3. 3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13. 56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm
Operation temperature	20~80 degrees (Celsius)
Storage temperature	40~85 degrees (Celsius)
Operation humidity	5%~95% (Relative humidity)

Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years.

The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63).

And each block contains 16 bytes (Byte0-Byte15), $64 \times 16 = 1024$. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
.....
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control put in its last block, that is, Block 3, which is also known as sector trailer. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the card serial number and vendor code, which has been solidified and can't be changed. Except the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

(1) used as general data storage and can be operated for reading and writing data.

(2) used as data value, and can be operated for initializing, adding, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, a 4-byte access control and a 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally 0A1A2A3A4A5 for password A and B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

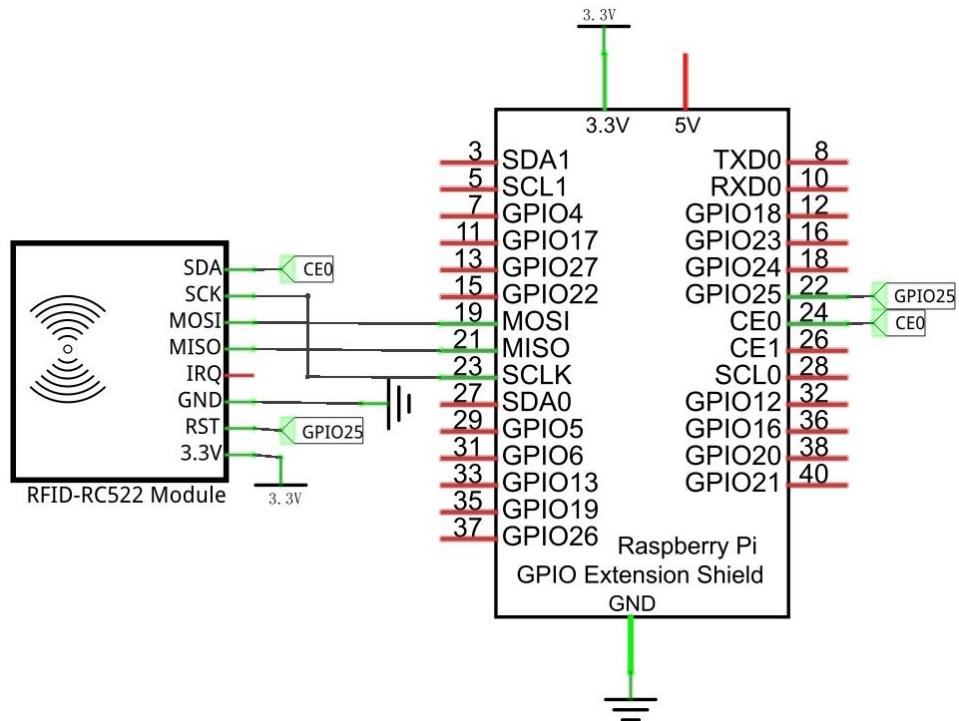
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read, so if you choose to verify password A but forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

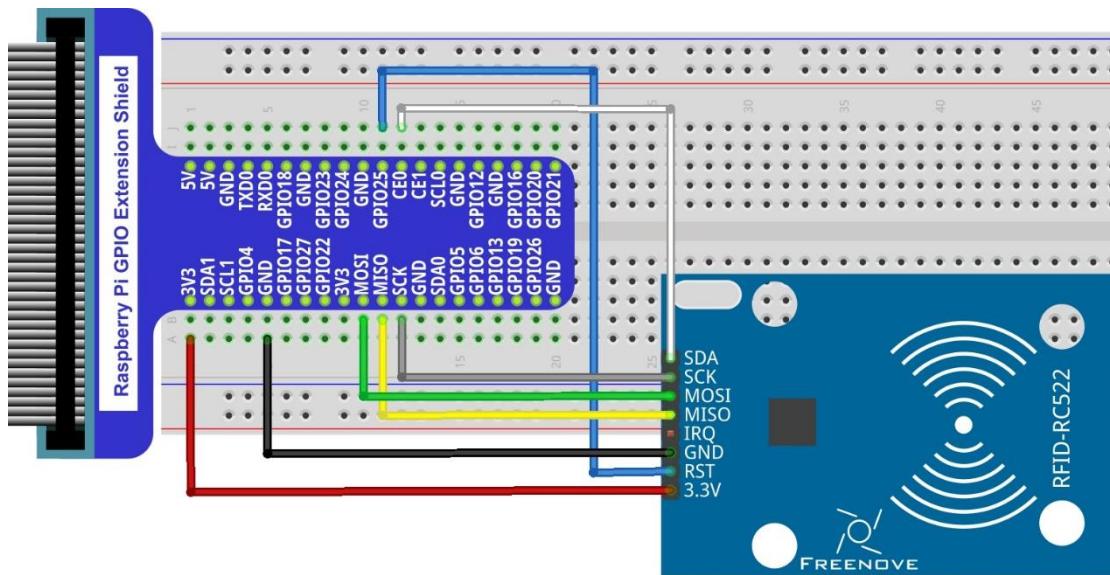
For Mifare1 S50 card equipped in Freenove RFID Kit, the default password A and B are both FFFFFFFFFF.

Circuit

Schematic diagram:



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Configure SPI

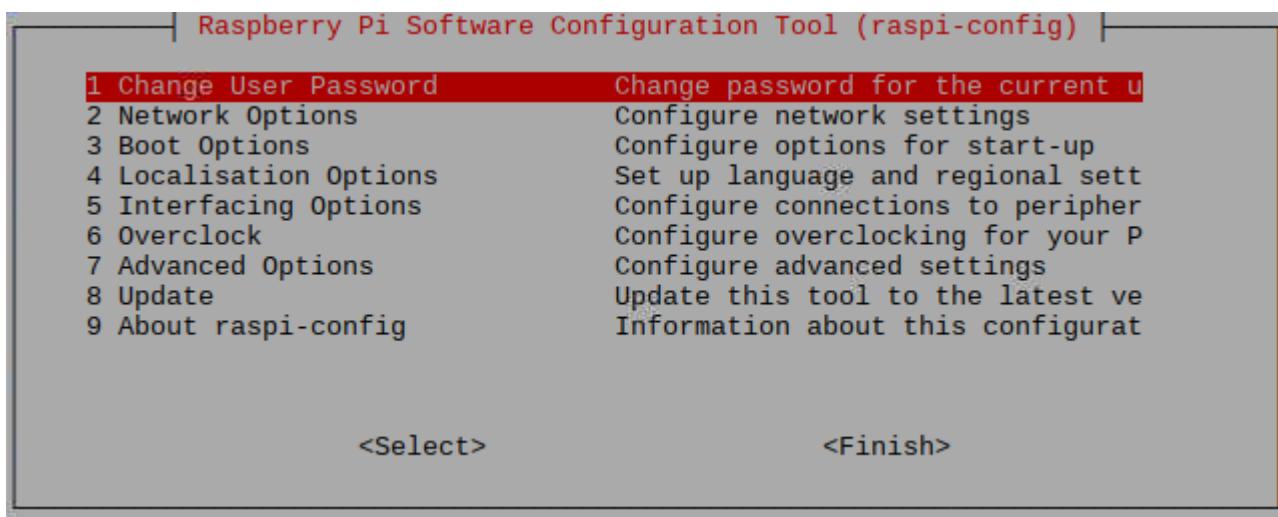
Enable SPI

The SPI interface of raspberry pi is closed by default. You need to open it manually. You can enable the SPI interface in the following way.

Type the following command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options” → “P4 SPI” → “Yes” → “Finish” in order and then restart your RPi. Then the SPI module is started.

Type the following command to check whether the module SPI is loaded successfully:

```
ls /dev/sp*
```

The following result indicates that the module SPI has been loaded successfully:

```
pi@raspberrypi:~ $ ls /dev/sp*
/dev/spidev0.0  /dev/spidev0.1
```

Install Python module SPI-Py

If you use Python language to write the code, please follow the steps below to install the module SPI-Py. If you use C/C++ language, you can skip this step.

Open the terminal and type the following command to install:

```
git clone https://github.com/Freenove/SPI-Py
cd SPI-Py
sudo python3 setup.py install
sudo python2 setup.py install
```

Code

The project code uses human-computer interaction command line mode to read and write the M1-S50 card.

C Code 24.1.1 RFID

First observe the running result, and then learn about the code in detail.

If you need any support, please contact us via: support@freenove.com

1. Use cd command to enter RFID directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/24.1.1_RFID
```

2. Use the following command to compile and generate executable file "RFID".

```
sudo sh ./build.sh
```

3. Then run the generated file "RFID".

```
sudo ./RFID
```

After the program is executed, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo sh ./build.sh
Build finished!
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo ./RFID
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>■
```

Here, type the command "quit" to exit the program.

Type command "scan", and then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522>scan
Scanning
.....
.....Card detected 0xE6 0xCF 0x5C 0x8E, Check Sum = 0xFB
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>E6CF5C8E>■
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
    read <blockstart>
    dump
    halt
    clean <blockaddr>
    write <blockaddr> <data>
```

In the command `read<blockstart>`, the parameter `blockstart` is the address of the data block, and the range is 0-63. This command is used to display all the data from `blockstart` address to the end of the sector. For example, sector 0 contains data block 0,1,2,3. Using the command “`read 0`” can display all contents of data block 0,1,2,3. Using the command “`read 1`” can display all contents of data block 1,2,3. As is shown below:

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  48: 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
RC522>E6CF5C8E>read 1
read
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  32: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff : .....i.....
RC522>E6CF5C8E>■
```

Command “`dump`” is used to display the content of all data blocks in all sectors.

Command `<address> <data>` is used to write “`data`” to data block with address “`address`”, where the address range is 0-63 and the data length is 0-16. For example, if you want to write the string “Freenove” to the data block with address “1”, you can type the following command.

```
write 1 Freenove
RC522>E6CF5C8E>write 1 Freenove
write
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to write block 1 with 8 byte data...OK
```

Read the contents of this sector and check the data just written.

read 0

The following results indicate that the string “Freenove” has been written successfully into the data block 1.

```
Read block address 0x03 ....OK read 144 bits
  0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
  16: 46 72 65 65 6e 6f 76 65 00 00 00 00 00 00 00 00 : Freenove.....
  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  48: 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
```

Command “`clean <address>`” is used to remove the contents of the data block with address “`address`”. For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

```
clean 1
RC522>E6CF5C8E>clean 1
clean
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to clean block 1...OK
```

Read the contents of data blocks in this sector again to check whether the data is erased. The following results indicate that the contents of data block 1 have been erased.

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  48: 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
```

Command "halt" is used to quit the selection state of the card.

```
RC522>E6CF5C8E>halt
halt
Halt
RC522>
```

The following is the program code:

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <unistd.h>
4 #include <string.h>
5 #include <getopt.h>
6 #include <stdlib.h>
7 #include "mfrc522.h"
8 #define DISP_COMMANDLINE() printf("RC522>")
9
10 int scan_loop(uint8_t *CardID);
11 int tag_select(uint8_t *CardID);
12 int main(int argc, char **argv) {
13     MFRC522_Status_t ret;
14     //Recognized card ID
15     uint8_t CardID[5] = { 0x00, };
16     static char command_buffer[1024];
17
18     ret = MFRC522_Init('A');
19     if (ret < 0) {
20         perror("Failed to initialize");
21         exit(-1);
22     }
23
24     printf("User Space RC522 Application\r\n");
25
26     while (1) {
27         /*Main Loop Start*/
28         DISP_COMMANDLINE();
29     }
```

```
30     scanf("%s", command_buffer);
31     if (strcmp(command_buffer, "scan") == 0) {
32         puts("Scanning");
33         while (1) {
34             ret = MFRC522_Check(CardID);
35             if (ret != MI_OK) {
36                 printf(".");
37                 fflush(stdout);
38                 continue;
39             }
40             ret |= tag_select(CardID);
41             if (ret == MI_OK) {
42                 ret = scan_loop(CardID);
43                 if (ret < 0) {
44                     goto END_SCAN;
45                 } else if (ret == 1) {
46                     goto HALT;
47                 }
48             }
49         }
50         END_SCAN: printf("Card error... ");
51         HALT: puts("Halt");
52     } else if (strcmp(command_buffer, "quit") == 0 ||
53                strcmp(command_buffer, "exit") == 0) {
54         return 0;
55     } else {
56         puts("Unknown command");
57         puts("scan:scan card and dump");
58         puts("quit:exit program");
59     }
60     /*Main Loop End*/
61 }
62 }
63 int scan_loop(uint8_t *CardID) {
64
65     while (1) {
66
67         char input[32];
68         int block_start;
69         DISP_COMMANDLINE();
70         printf("%02X%02X%02X%02X>, CardID[0], CardID[1], CardID[2], CardID[3]);
71         scanf("%s", input);
72         puts((char*)input);
73         if (strcmp(input, "halt") == 0) {
```

```
74         return 1;
75     } else if (strcmp(input, "dump") == 0) {
76         if (MFRC522_Debug_CardDump(CardID) < 0)
77             return -1;
78     } else if (strcmp(input, "read") == 0) {
79         scanf("%d", &block_start);
80         if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
81             return -1;
82         }
83     } else if(strcmp(input, "clean") == 0) {
84         char c;
85         scanf("%d", &block_start);
86         while ((c = getchar()) != '\n' && c != EOF)
87             ;
88         if (MFRC522_Debug_Clean(CardID, block_start)) {
89             return -1;
90         }
91
92     } else if (strcmp(input, "write") == 0) {
93         char write_buffer[256];
94         size_t len = 0;
95         scanf("%d", &block_start);
96         scanf("%s", write_buffer);
97         if (len >= 0) {
98             if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
99                 strlen(write_buffer)) < 0) {
100                 return -1;
101             }
102         }
103     } else {
104
105         printf(
106             "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n"
107             "\thalt\r\n" "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
108         return 0;
109     }
110 }
111 return 0;
112 }
113 int tag_select(uint8_t *CardID) {
114     int ret_int;
115     printf(
116         "Card detected    0x%02X 0x%02X 0x%02X 0x%02X, Check Sum = 0x%02X\r\n",
117 }
```

```

118     CardID[0], CardID[1], CardID[2], CardID[3], CardID[4]);
119     ret_int = MFRC522_SelectTag(CardID);
120     if (ret_int == 0) {
121         printf("Card Select Failed\r\n");
122         return -1;
123     } else {
124         printf("Card Selected, Type:%s\r\n",
125                MFRC522_TypeToString(MFRC522_ParseType(ret_int)));
126     }
127     ret_int = 0;
128     return ret_int;
129 }
```

In the code, first initialize the MFRC522. If the initialization fails, the program will exit.

```

ret = MFRC522_Init('A');
if (ret < 0) {
    perror("Failed to initialize");
    exit(-1);
}
```

In the main function, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```

scanf("%s", command_buffer);
if (strcmp(command_buffer, "scan") == 0) {
    puts("Scanning");
    while (1) {
        ret = MFRC522_Check(CardID);
        if (ret != MI_OK) {
            printf(".");
            fflush(stdout);
            continue;
        }
        ret |= tag_select(CardID);
        if (ret == MI_OK) {
            ret = scan_loop(CardID);
            if (ret < 0) {
                goto END_SCAN;
            } else if (ret == 1) {
                goto HALT;
            }
        }
    }
}
END_SCAN: printf("Card error... ");
HALT: puts("Halt");
```

```

} else if (strcmp(command_buffer, "quit") == 0
           || strcmp(command_buffer, "exit") == 0) {
    return 0;
} else {
    puts("Unknown command");
    puts("scan:scan card and dump");
    puts("quit:exit program");
}
/*Main Loop End*/
}

```

The function `scan_loop()` will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```

int scan_loop(uint8_t *CardID) {

    while (1) {

        char input[32];
        int block_start;
        DISP_COMMANDLINE();
        printf("%02X%02X%02X%02X>, CardID[0], CardID[1], CardID[2], CardID[3]);
        scanf("%s", input);
        puts((char*)input);
        if (strcmp(input, "halt") == 0) {
            return 1;
        } else if (strcmp(input, "dump") == 0) {
            if (MFRC522_Debug_CardDump(CardID) < 0)
                return -1;
        } else if (strcmp(input, "read") == 0) {
            scanf("%d", &block_start);
            if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
                return -1;
            }
        } else if (strcmp(input, "clean") == 0) {
            char c;
            scanf("%d", &block_start);
            while ((c = getchar()) != '\n' && c != EOF)
                ;
            if (MFRC522_Debug_Clean(CardID, block_start)) {
                return -1;
            }
        } else if (strcmp(input, "write") == 0) {
            char write_buffer[256];
            size_t len = 0;

```

```
scanf("%d", &block_start);
scanf("%s", write_buffer);
if (len >= 0) {
    if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
        strlen(write_buffer)) < 0) {
        return -1;
    }
}
} else {

printf(
    "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
    "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
return 0;
}
}
return 0;
}
```

The header file "mfrc522.h" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Python Code 24.1.1 RFID

There are two code files for this project. They are respectively under Python2 folder and Python3 folder. **Their functions are the same, but they are not compatible.** Code under Python2 folder can only run on Python2. And code under Python3 folder can only run on Python3.

First observe the project result, and then learn about the code in detail.

If you need any support, please contact us via: support@freenove.com

1. Use cd command to enter RFID directory of Python code.

If you use Python2, it is needed to enter Python2 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python2
```

If you use Python3, it is needed to enter Python3 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
```

2. Use python command to execute code "RFID.py".

```
python RFID.py
```

After the program is executed, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3 $ python RFID.py
Program is starting ...
Press Ctrl-C to exit.
RC522>■
```

Here, type the command "quit" to exit the program.

Type command "scan", then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522> scan
scan
Scanning ...
Card detected
Card UID: ['0xe6', '0xcf', '0x5c', '0x8e', '0xfb']
Size: 8
RC522> E6CF5C8EFB> ■
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
    read <blockstart>
    dump
    halt
    clean <blockaddr>
    write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. As is shown below:

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to read the data of data block with address "blockstart". For example, using command "read 0" can display the content of data block 0. Using the command "read 1" can display the content of data block 1. As is shown below:

Command “dump” is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write "data" to data block with address "address", where the address range is 0-63 and the data length is 0-16. In the process of writing data to the data block, both the contents of data block before written and after written will be displayed. For example, if you want to write the string "Freenove" to the data block with address "1", you can type the following command.

write 1 Freenove

Command "clean <address>" is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

clean 1

```
RC522> E6CF5C8EFB> clean 1
['clean', '1']
Before cleaning , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0 | Freenove
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
```

Command “halt” is used to quit the selection state of the card.

```
RC522> E6CF5C8EFB> halt  
['halt']  
RC522> █
```

The following is the program code (python2 code):

```
1 import RPi.GPIO as GPIO
2
3
4 # Create an object of the class MFRC522
5 mfrc = MFRC522.MFRC522()
6
7 def dis_CommandLine():
8     print "RC522>",
9 def dis_CardID(cardID):
10    print "%2X%2X%2X%2X%2X>%"(cardID[0], cardID[1], cardID[2], cardID[3], cardID[4]),
11 def setup():
```

```
12     print "Program is starting ... "
13     print "Press Ctrl-C to exit."
14     pass
15
16 def loop():
17     while(True):
18         dis_CommandLine()
19         inCmd = raw_input()
20         print inCmd
21         if (inCmd == "scan"):
22             print "Scanning ... "
23             isScan = True
24             while isScan:
25                 # Scan for cards
26                 (status, TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
27                 # If a card is found
28                 if status == mfrc.MI_OK:
29                     print "Card detected"
30                     # Get the UID of the card
31                     (status,uid) = mfrc.MFRC522_Anticoll()
32                     # If we have the UID, continue
33                     if status == mfrc.MI_OK:
34                         print "Card UID: "+ str(map(hex,uid))
35                         # Select the scanned tag
36                         if mfrc.MFRC522_SelectTag(uid) == 0:
37                             print "MFRC522_SelectTag Failed!"
38                             if cmdloop(uid) < 1 :
39                                 isScan = False
40                         elif inCmd == "quit":
41                             destroy()
42                             exit(0)
43                         else :
44                             print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
45 program\n"
46
47 def cmdloop(cardID):
48     pass
49     while(True):
50         dis_CommandLine()
51         dis_CardID(cardID)
52         inCmd = raw_input()
53         cmd = inCmd.split(" ")
54         print cmd
55         if(cmd[0] == "read"):
```

```
56     blockAddr = int(cmd[1])
57     if((blockAddr<0) or (blockAddr>63)):
58         print "Invalid Address!"
59     # This is the default key for authentication
60     key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
61     # Authenticate
62     status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
63     # Check if authenticated
64     if status == mfrc.MI_OK:
65         mfrc.MFRC522_Readstr(blockAddr)
66     else:
67         print "Authentication error"
68         return 0
69
70 elif cmd[0] == "dump":
71     # This is the default key for authentication
72     key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
73     mfrc.MFRC522_Dump_Str(key, cardID)
74
75 elif cmd[0] == "write":
76     blockAddr = int(cmd[1])
77     if((blockAddr<0) or (blockAddr>63)):
78         print "Invalid Address!"
79     data = [0]*16
80     if(len(cmd)<2):
81         data = [0]*16
82     else:
83         data = cmd[2][0:17]
84         data = map(ord, data)
85         if len(data)<16:
86             data+=[0]*(16-len(data))
87     # This is the default key for authentication
88     key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
89     # Authenticate
90     status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
91     # Check if authenticated
92     if status == mfrc.MI_OK:
93         print "Before writing , The data in block %d  is: "%(blockAddr)
94         mfrc.MFRC522_Readstr(blockAddr)
95         mfrc.MFRC522_Write(blockAddr, data)
96         print "After written , The data in block %d  is: "%(blockAddr)
97         mfrc.MFRC522_Readstr(blockAddr)
98     else:
99         print "Authentication error"
```

```

100         return 0
101
102     elif cmd[0] == "clean":
103         blockAddr = int(cmd[1])
104         if((blockAddr<0) or (blockAddr>63)):
105             print "Invalid Address!"
106         data = [0]*16
107         # This is the default key for authentication
108         key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
109         # Authenticate
110         status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
111         # Check if authenticated
112         if status == mfrc.MI_OK:
113             print "Before cleaning , The data in block %d is: %(blockAddr)
114             mfrc.MFRC522_Readstr(blockAddr)
115             mfrc.MFRC522_Write(blockAddr, data)
116             print "After cleaned , The data in block %d is: %(blockAddr)
117             mfrc.MFRC522_Readstr(blockAddr)
118         else:
119             print "Authentication error"
120             return 0
121         elif cmd[0] == "halt":
122             return 0
123         else :
124             print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
125             "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"
126
127     def destroy():
128         GPIO.cleanup()
129
130     if __name__ == "__main__":
131         setup()
132         try:
133             loop()
134         except KeyboardInterrupt: # Ctrl+C captured, exit
135             destroy()

```

In the code, first create an MFRC522 class object.

```
mfrc = MFRC522.MFRC522()
```

In the function loop, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```

if (inCmd == "scan"):
    print "Scanning ... "

```

```

isScan = True
while isScan:
    .....
    if cmdloop(uid) < 1 :
        isScan = False
    elif inCmd == "quit":
        destroy()
        exit(0)
    else :
        print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
program\n"

```

The function cmdloop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```

def cmdloop(cardID):
    pass
    while(True):
        dis_CommandLine()
        dis_CardID(cardID)
        inCmd = raw_input()
        cmd = inCmd.split(" ")
        print cmd
        if(cmd[0] == "read"):
            .....
        elif cmd[0] == "dump":
            .....
        elif cmd[0] == "write":
            .....
        elif cmd[0] == "clean":
            .....
        elif cmd[0] == "halt":
            return 0
        else :
            print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
            "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"

```

The file "MFRC522.py" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Chapter 25 Web IoT

In this chapter, we will learn how to use GPIO to control the RPi remotely via a network and how to build a WebIO service on the RPi.

This concept is known as “IoT” or Internet of Things. The development of IoT will greatly change our habits and make our lives more convenient and efficient

Project 25.1 Remote LED

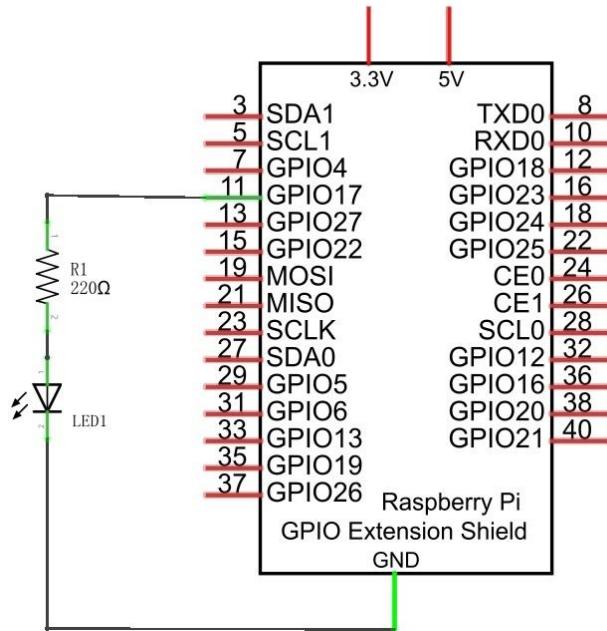
In this project, we need to build a WebIOPi service, and then use the RPi GPIO to control an LED through the web browser of phone or PC.

Component List

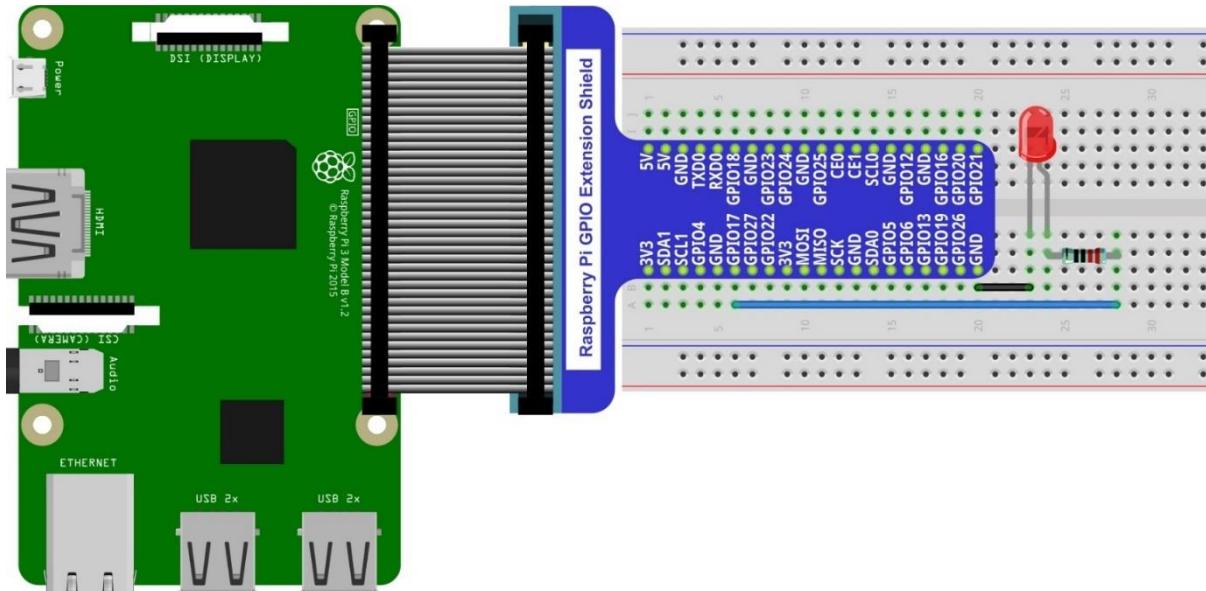
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1	Resistor 220Ω x1
Jumper M/M x2		

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Solution from E-Tinkers

Here is a solution from blog E-Tinkers, author Henry Cheung. For more details, please refer to link below:
<https://www.e-tinkers.com/2018/04/how-to-control-raspberry-pi-gpio-via-http-web-server/>

1, Make sure you have set python3 as default python. Then run following command in terminal to install http.server in your Raspberry Pi.

```
sudo apt-get install http.server
```

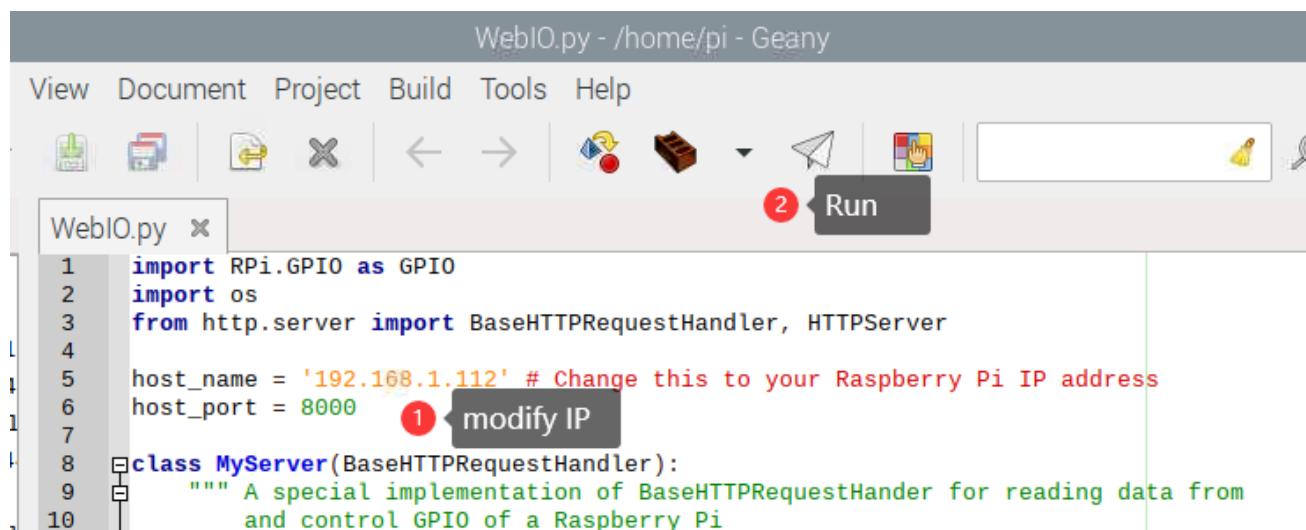
2, Open WebIO.py

```
cd ~/Freenove_Kit/Code/Python_Code/25.1.1_WebIO
geany WebIO.py
```

3, Change the host_name into your Raspberry Pi IP address.

```
host_name = '192.168.1.112' # Change this to your Raspberry Pi IP address
```

Then run the code WebIO.py



3, Visit <http://192.168.1.112:8000/> in web brower on computer under local area networks. **Change IP to your Raspberry Pi IP address.**



Welcome to my Raspberry Pi

Current GPU temperature is 53.0'C

WebIOPi Service Framework

Note: If you have a Raspberry Pi 4B, you may have some trouble. The reason for changing the file in the configuration process is that the newer generation models of the RPi CPUs are different from the older ones and you may not be able to access the GPIO Header at the end of this tutorial. A solution to this is given in an online tutorial by from E-Tinkers blogger Henry Cheung. For more details, please refer to previous section.

The following is the key part of this chapter. The installation steps refer to WebIOPi official. And you also can directly refer to the official installation steps. The latest version (in 2016-6-27) of WebIOPi is 0.7.1. So, you may encounter some issues in using it. We will explain these issues and provide the solution in the following installation steps.

Here are the steps to build a WebIOPi:

Installation

1. Get the installation package. You can use the following command to obtain.

```
 wget https://github.com/Freenove/WebIOPi/archive/master.zip -O WebIOPi.zip
```

2. Extract the package and generate a folder named "WebIOPi-master". Then enter the folder.

```
 unzip WebIOPi.zip
```

```
 cd WebIOPi-master/WebIOPi-0.7.1
```

3. Patch for Raspberry Pi B+, 2B, 3B, 3B+.

```
 patch -p1 -i webiopi-pi2bplus.patch
```

4. Run setup.sh to start the installation, the process takes a while and you will need to be patient.

```
 sudo ./setup.sh
```

5. If setup.sh does not have permission to execute, execute the following command

```
 sudo sh ./setup.sh
```

Run

After the installation is completed, you can use the webiopi command to start running.

```
$ sudo webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]
```

Options:

-h, --help	Display this help
-c, --config file	Load config from file
-l, --log file	Log to file
-s, --script file	Load script from file
-d, --debug	Enable DEBUG

Arguments:

port	Port to bind the HTTP Server
-------------	------------------------------

Run webiopi with verbose output and the default config file:

```
 sudo webiopi -d -c /etc/webiopi/config
```

The Port is 8000 in default. Now WebIOPi has been launched. Keep it running.

Access WebIOPi over local network

Under the same network, use a mobile phone or PC browser to open your RPi IP address, and add a port number like 8000. For example, my personal Raspberry Pi IP address is 192.168.1.109. Then, in the browser, I then should input: <http://192.168.1.109:8000/>

Default user is "webiopi" and password is "raspberry".

Then, enter the main control interface:

WebIOPi Main Menu

GPIO Header

Control and Debug the Raspberry Pi GPIO with a display which looks like the physical header.

GPIO List

Control and Debug the Raspberry Pi GPIO ordered in a single column.

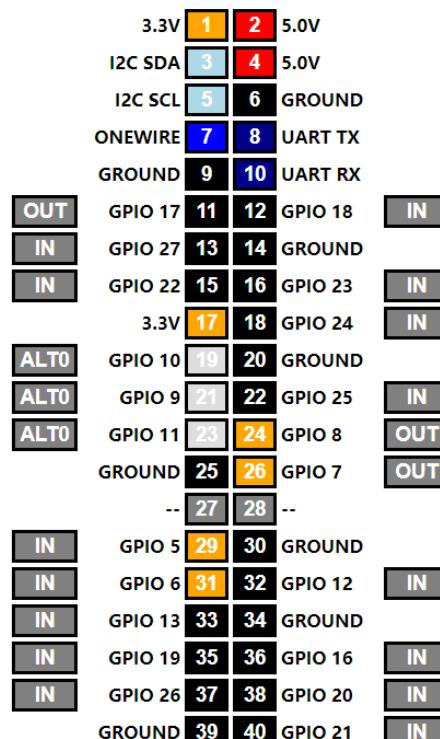
Serial Monitor

Use the browser to play with Serial interfaces configured in WebIOPi.

Devices Monitor

Control and Debug devices and circuits wired to your Pi and configured in WebIOPi.

Click on GPIO Header to enter the GPIO control interface.



Control methods:

- Click/Tap the OUT/IN button to change GPIO direction.
- Click/Tap pins to change the GPIO output state.



Completed

According to the circuit we build, set GPIO17 to OUT, then click Header11 to control the LED.
You can end the webioPi in the terminal by "Ctr+C".

What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself a Raspberry Pi Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:
support@freenove.com

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.

<http://www.freenove.com/>

Thank you again for choosing Freenove products.