Getting Started

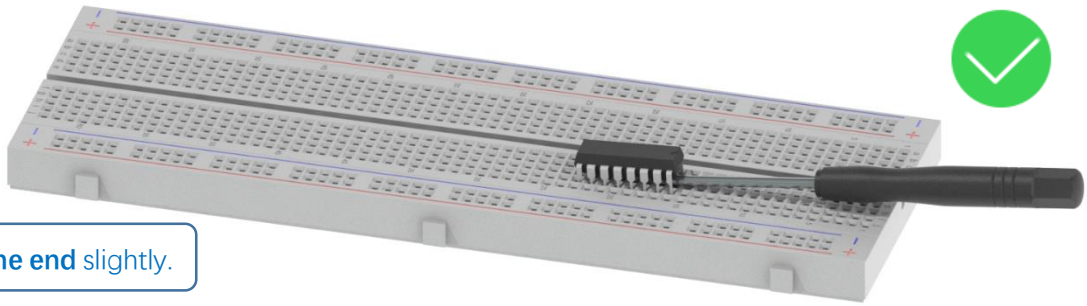
Thank you for choosing Freenove products!
After you download the ZIP file we provide. Unzip it and you will get a folder contains several files and folders. There are three PDF files:

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

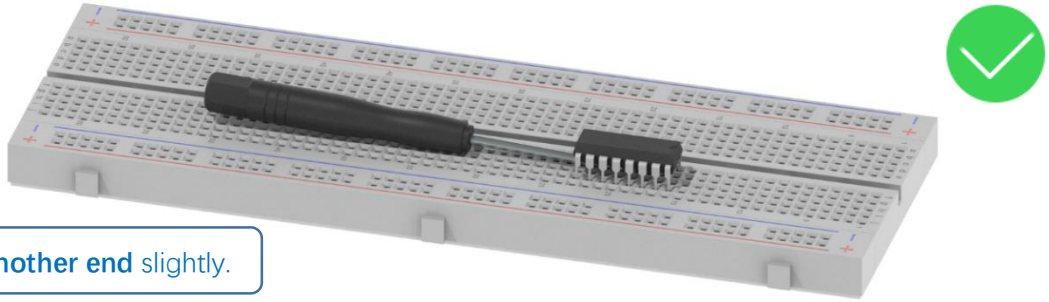
We recommend you to start with **Tutorial.pdf** or **Tutorial_GPIOZero.pdf** first.
If you want to start with Processing.pdf or skip some chapters of Tutorial.pdf, you need to finish necessary steps in **Chapter 7 AD/DA** of **Tutorial.pdf** first.

Remove the Chips

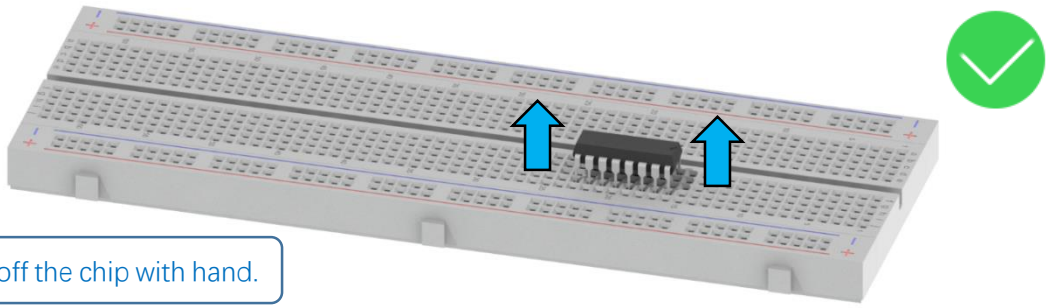
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



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

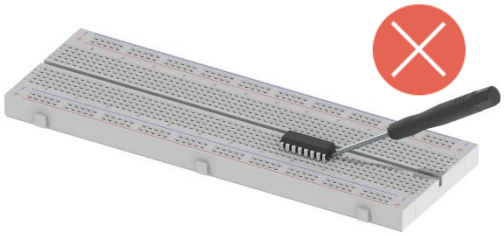


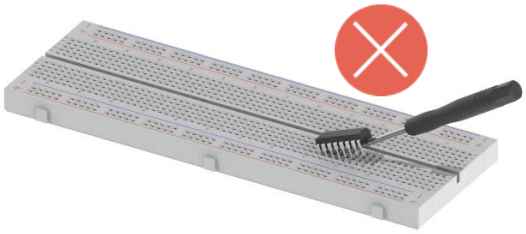
Step 2, lift **another end** slightly.



Step 3, take off the chip with hand.

Avoid lifting one end with big angle directly.





Get Support and Offer Input

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

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

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

support@freenove.com

Safety and Precautions

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

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

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

About Freenove

Freenove provides open source electronic products and services worldwide.

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

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

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

http://www.freenove.com

Copyright

Contents

Preface

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

In this tutorial, most chapters consist of **Components List**, **Component Knowledge**, **Circuit**, and **Code**. We provide C 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:

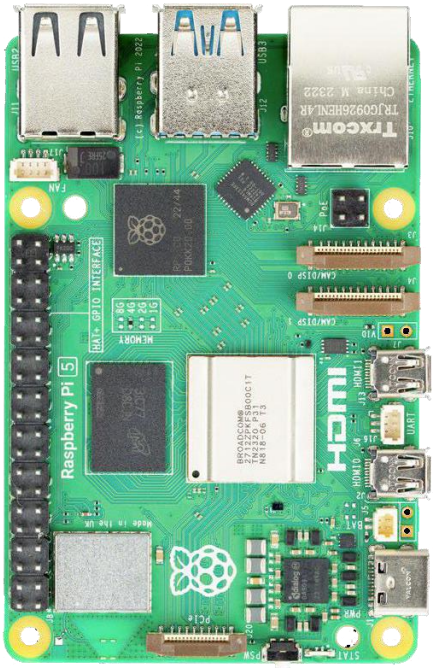
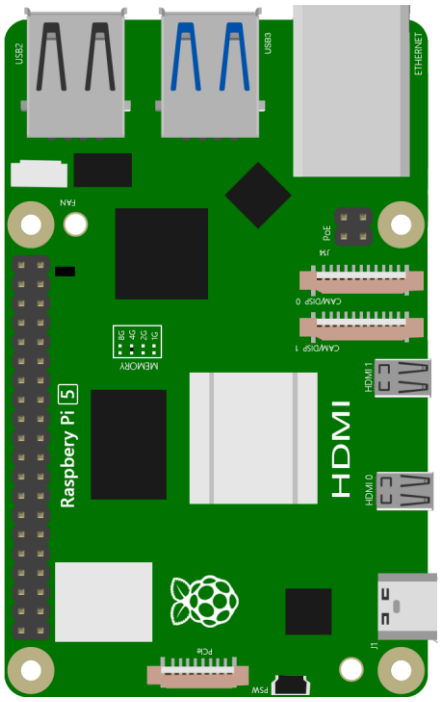
support@freenove.com

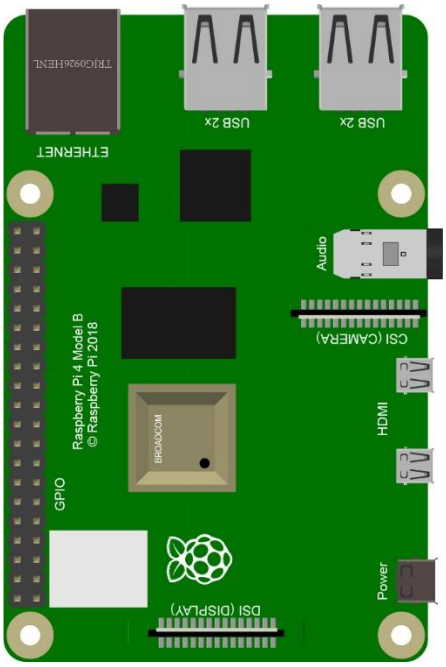
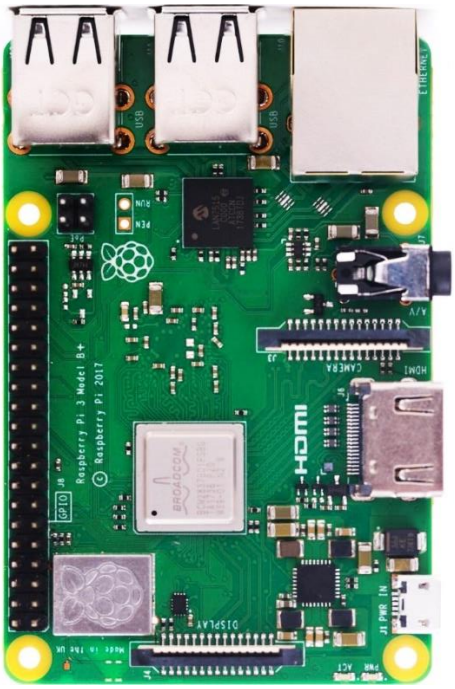
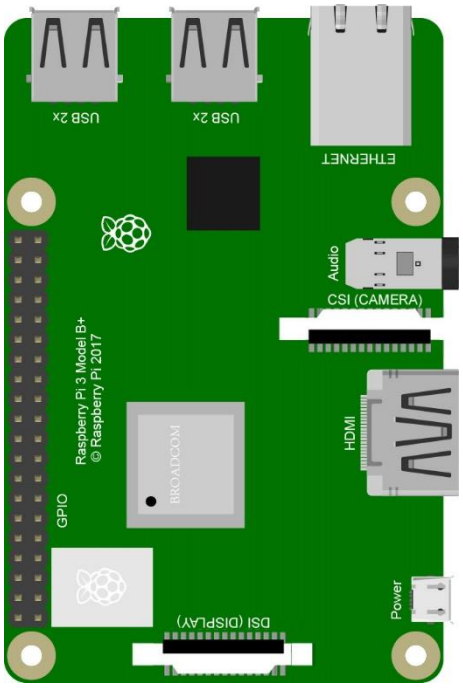
Raspberry Pi

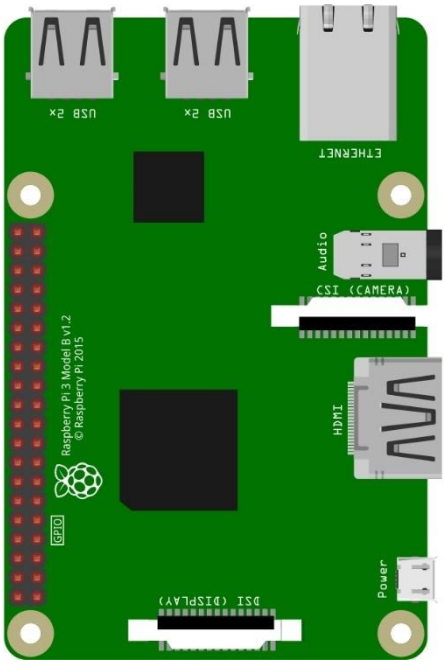
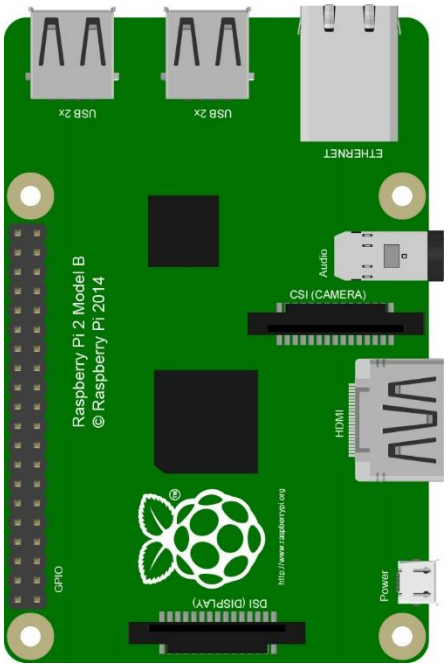
So far, at this writing, Raspberry Pi has advanced to its fifth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

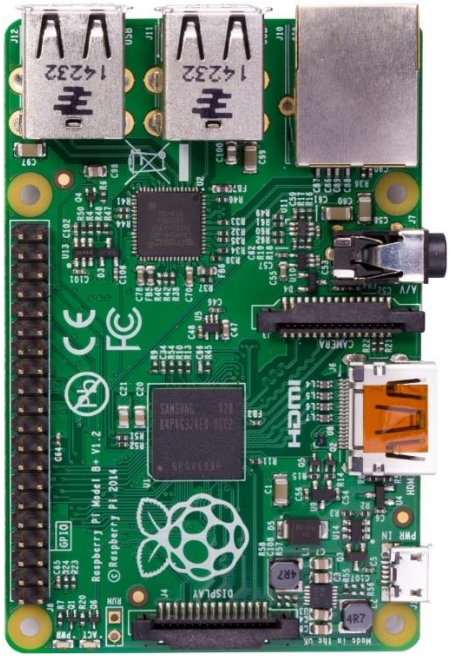
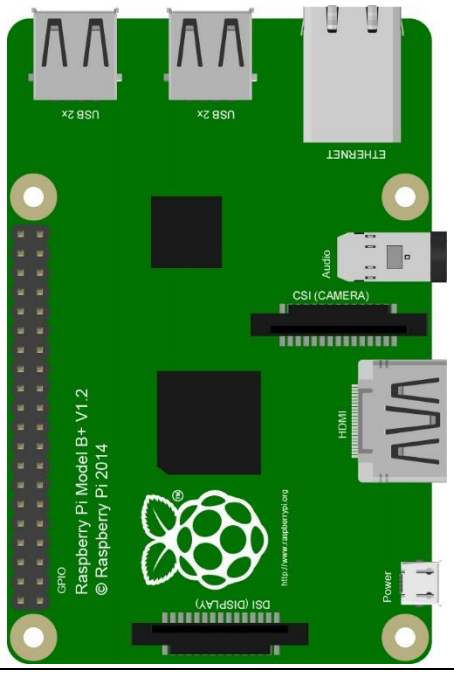
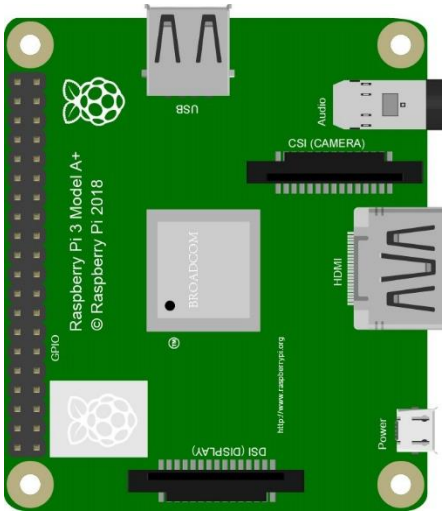
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility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

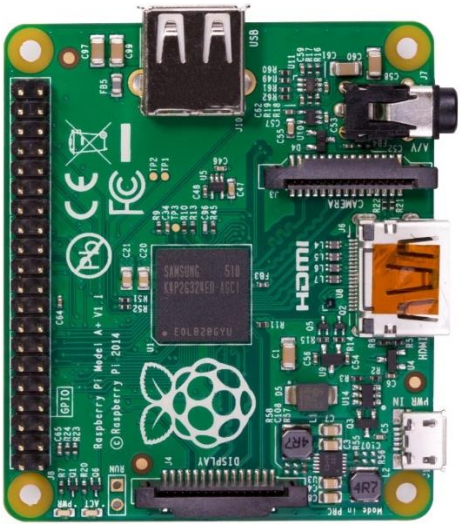
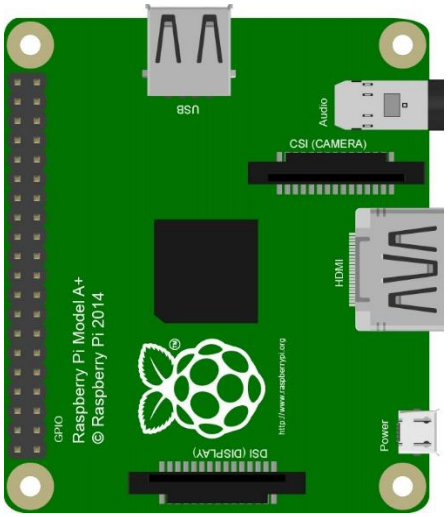
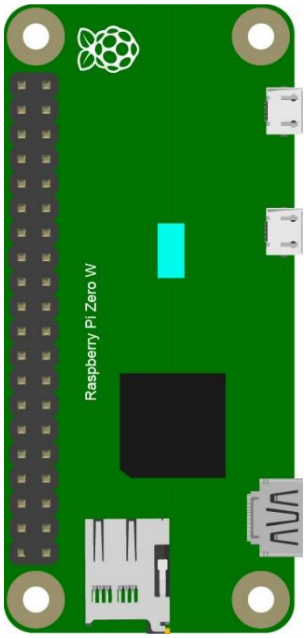
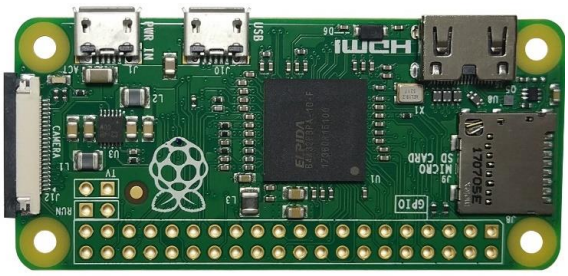
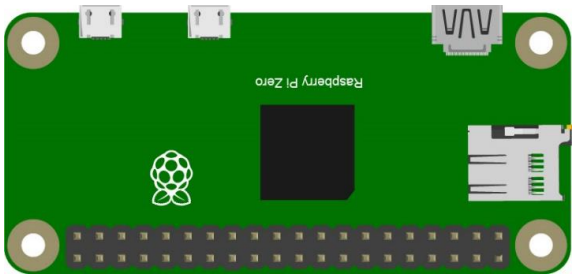
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

Practicality picture of Raspberry Pi 5:	Model diagram of Raspberry Pi 5:
	

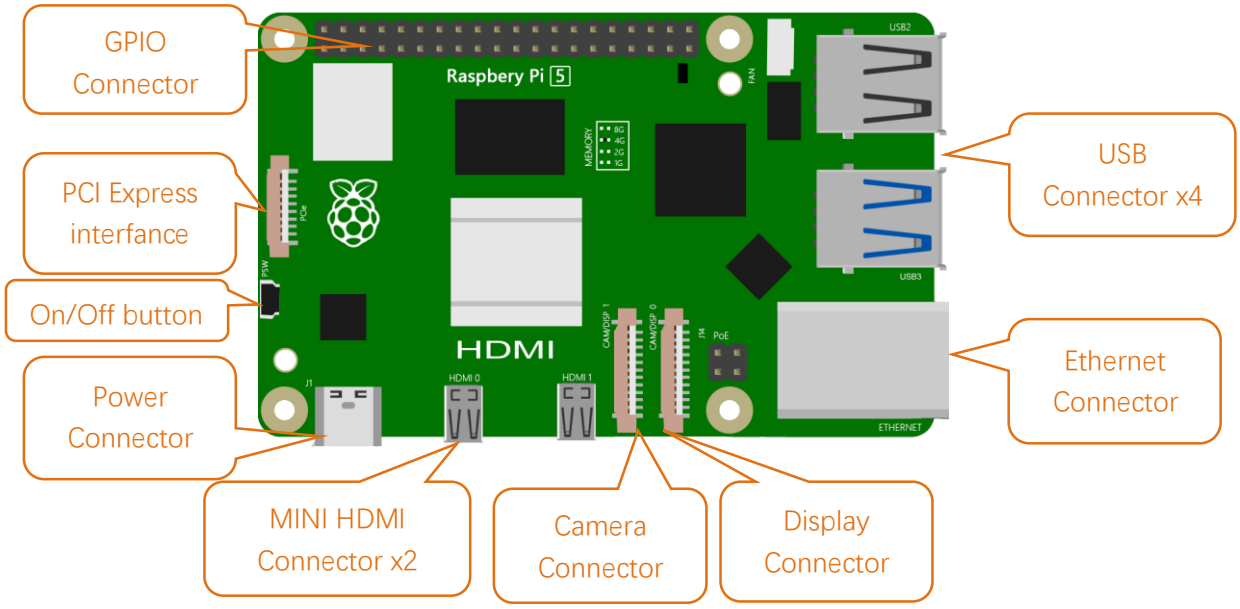
Actual image of Raspberry Pi 4 Model B: 	CAD image of Raspberry Pi 4 Model B: 
Actual image of Raspberry Pi 3 Model B+: 	CAD image of Raspberry Pi 3 Model B+: 

Actual image of Raspberry Pi 3 Model B: 	CAD image of Raspberry Pi 3 Model B: 
Actual image of Raspberry Pi 2 Model B: 	CAD image of Raspberry Pi 2 Model B: 

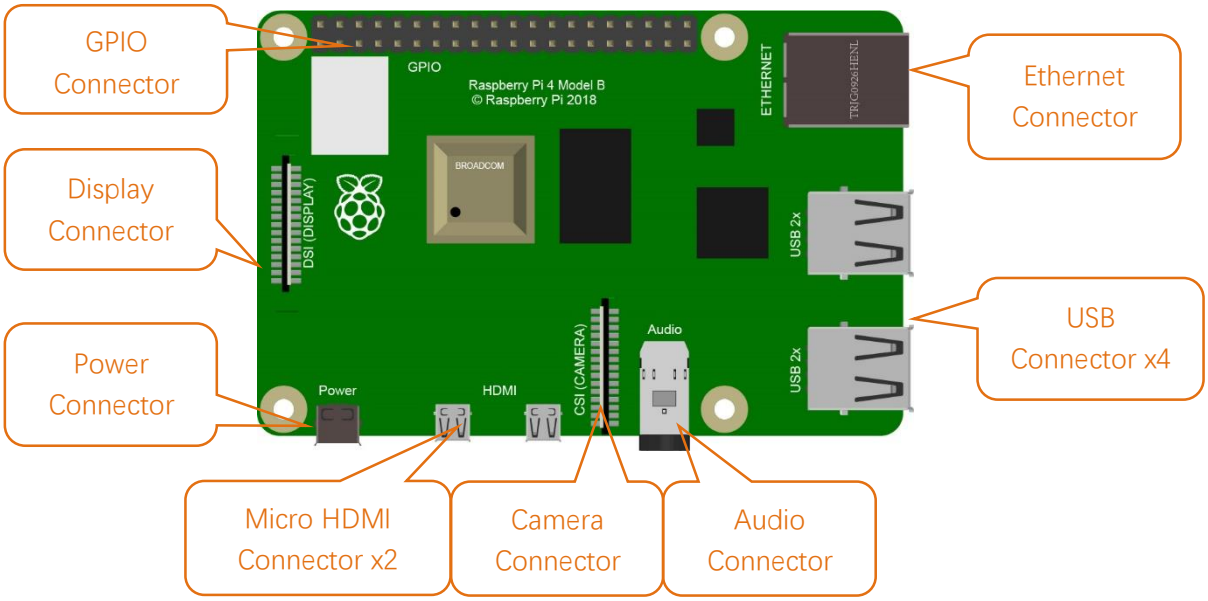
Actual image of Raspberry Pi 1 Model B+:	CAD image of Raspberry Pi 1 Model B+:
	
Actual image of Raspberry Pi 3 Model A+:	CAD image of Raspberry Pi 3 Model A+:
	

Actual image of Raspberry Pi 1 Model A+:	CAD image of Raspberry Pi 1 Model A+:
	
Actual image of Raspberry Pi Zero W:	CAD image of Raspberry Pi Zero W:
	
Actual image of Raspberry Pi Zero:	CAD image of Raspberry Pi Zero:
	

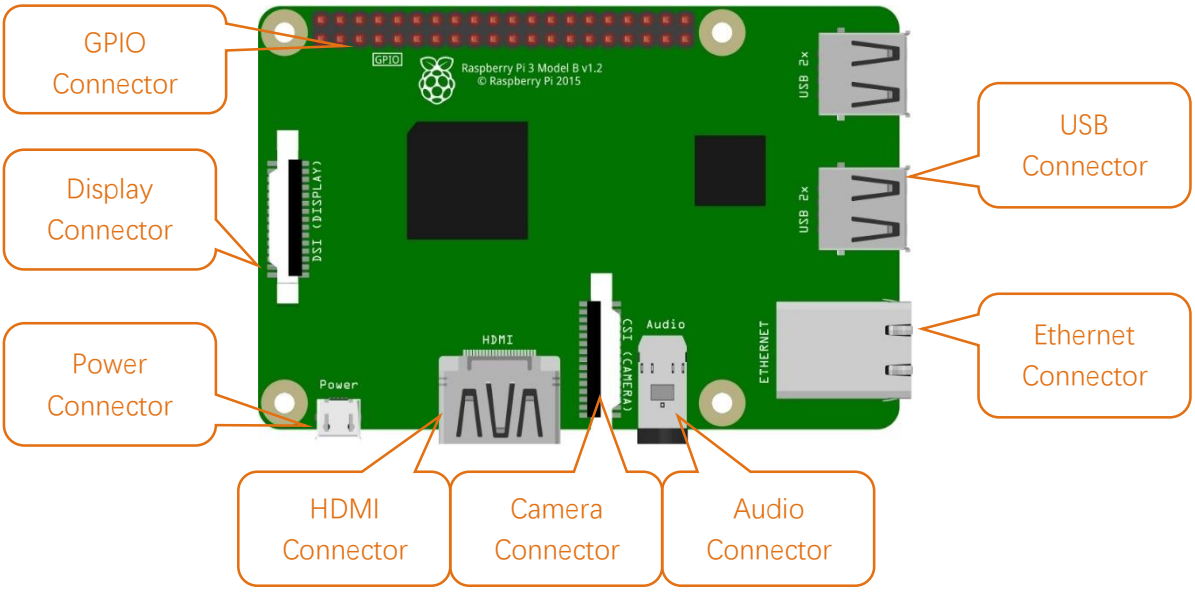
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.
Hardware interface diagram of RPi 5 is shown below:



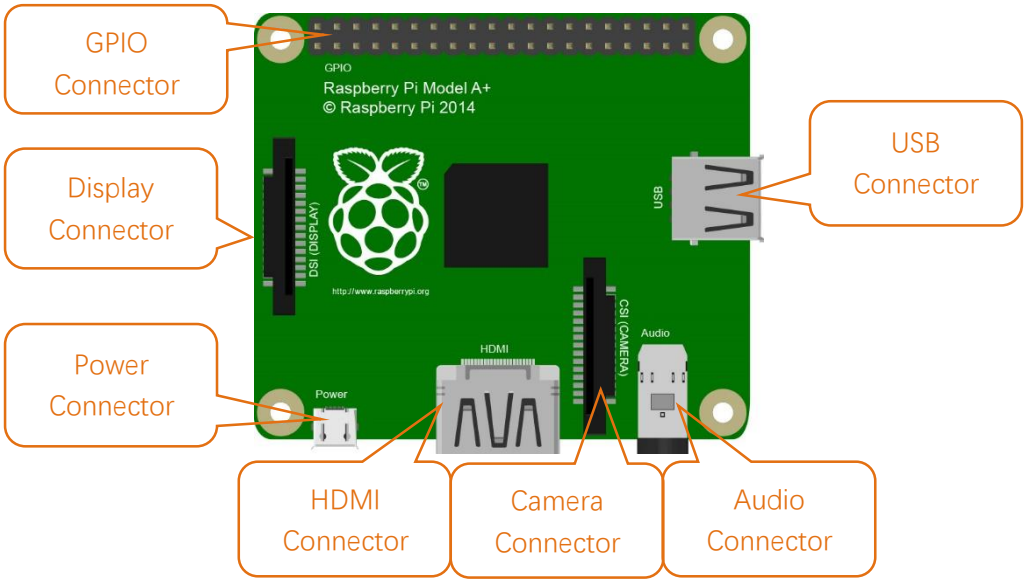
Hardware interface diagram of RPi 4B is shown below:



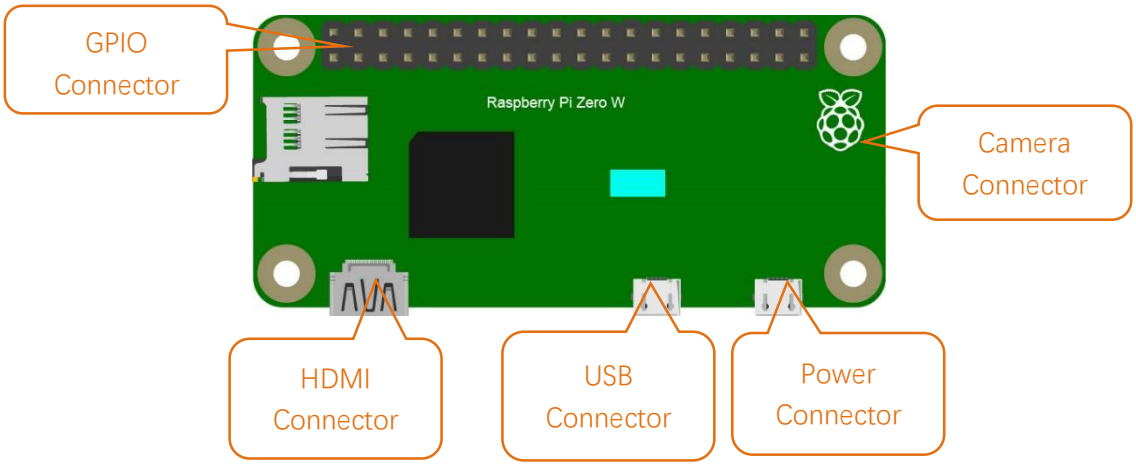
Hardware interface diagram of RPi 3B+/3B/2B/1B+:



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



Hardware interface diagram of RPi Zero/Zero W:

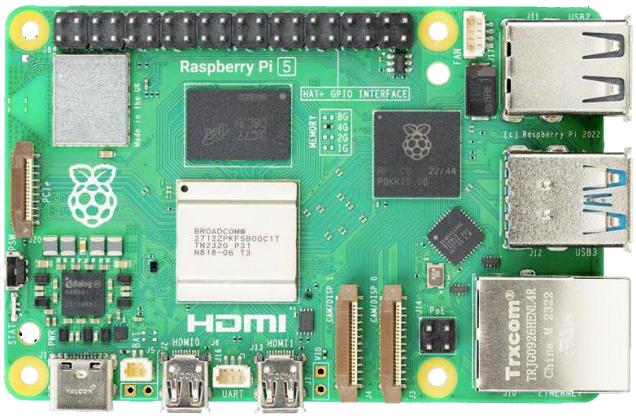
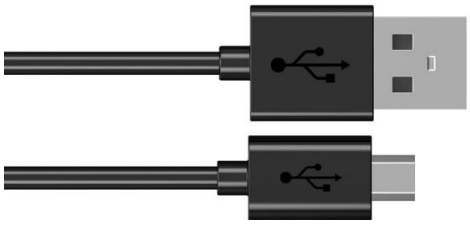


Installing an Operating System

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 1 Model A	700mA	500mA	200mA
Raspberry Pi 1 Model B	1.2A	500mA	500mA
Raspberry Pi 1 Model A+	700mA	500mA	180mA
Raspberry Pi 1 Model B+	1.8A	1.2A	330mA
Raspberry Pi 2 Model B	1.8A	1.2A	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 5	5.0A	1.6A (600mA if using a 3A power supply)	800mA
Raspberry Pi 400	3.0A	1.2A	800mA
Raspberry Pi Zero	1.2A	Limited by PSU, board, and connector ratings only	100mA
Raspberry Pi Zero W	1.2A	Limited by PSU, board, and connector ratings only.	150mA
Raspberry Pi Zero 2 W	2A	Limited by PSU, board, and connector ratings only.	350mA

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).

The Raspberry Pi 5 provides 1.6A of power to downstream USB peripherals when connected to a power supply capable of 5A at +5V (25W). When connected to any other compatible power supply, the Raspberry Pi 5 restricts downstream USB devices to 600mA of power.

Optional Components

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

Required Accessories for Monitor

If you choose to use an independent monitor, mouse and keyboard, you also need the following accessories:
1. A display with a HDMI interface
2. A Mouse and a Keyboard with an USB interface

As to Pi Zero and Pi Zero W, you also need the following accessories:
1. A Mini-HDMI to HDMI Adapter and Cable.
2. A Micro-USB to USB-A Adapter and Cable (Micro USB OTG Cable).
3. A USB HUB.
4. USB to Ethernet Interface or USB Wi-Fi receiver.

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

<table>
<tr><th></th><th>Pi Zero</th><th>Pi A+</th><th>Pi Zero W</th><th>Pi 3A+</th><th>Pi B+/2B</th><th>Pi 3B/3B+</th><th>Pi 4B</th><th>Pi 5</th></tr>
<tr><td>Monitor</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Mouse</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Keyboard</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td>Yes</td><td>No</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td colspan="6">No</td><td colspan="2">Yes</td></tr>
<tr><td>Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)</td><td>Yes</td><td>No</td><td>Yes</td><td colspan="5">No</td></tr>
<tr><td>USB HUB</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>USB to Ethernet Interface</td><td rowspan="2" colspan="2">select one from two or select two from two</td><td colspan="2">optional</td><td>Internal Integration</td><td rowspan="2" colspan="3">Internal Integration</td></tr>
<tr><td>USB Wi-Fi Receiver</td><td colspan="2">Internal Integration</td><td>optional</td></tr>
</table>

Required Accessories for Remote Desktop

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

	Pi Zero	Pi Zero W	Pi A+	Pi 3A+ Pi B+/2B Pi 3B/3B+/4B/5
Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)	Yes	Yes	No	NO
USB to Ethernet interface	Yes	Yes	Yes	

Raspberry Pi OS

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

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

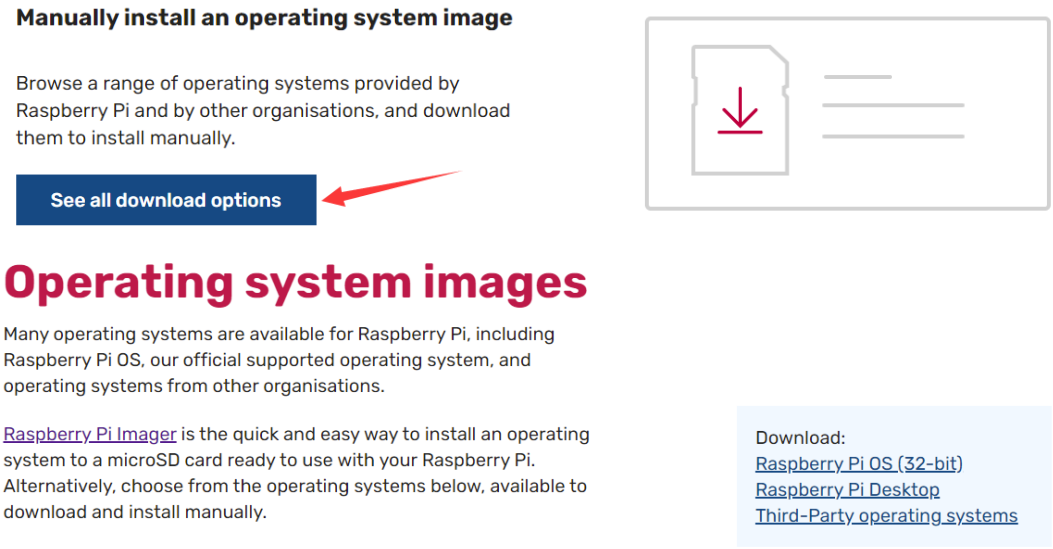
Automatically Method

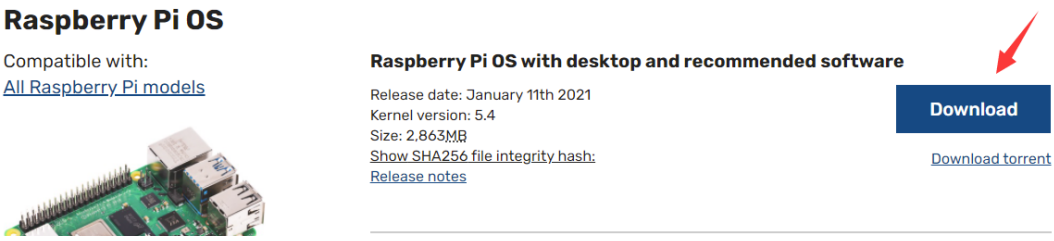
You can follow the official method to install the system for raspberry pi via visiting link below:
https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2
In this way, the system will be downloaded **automatically** via the application.

Manually Method

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

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



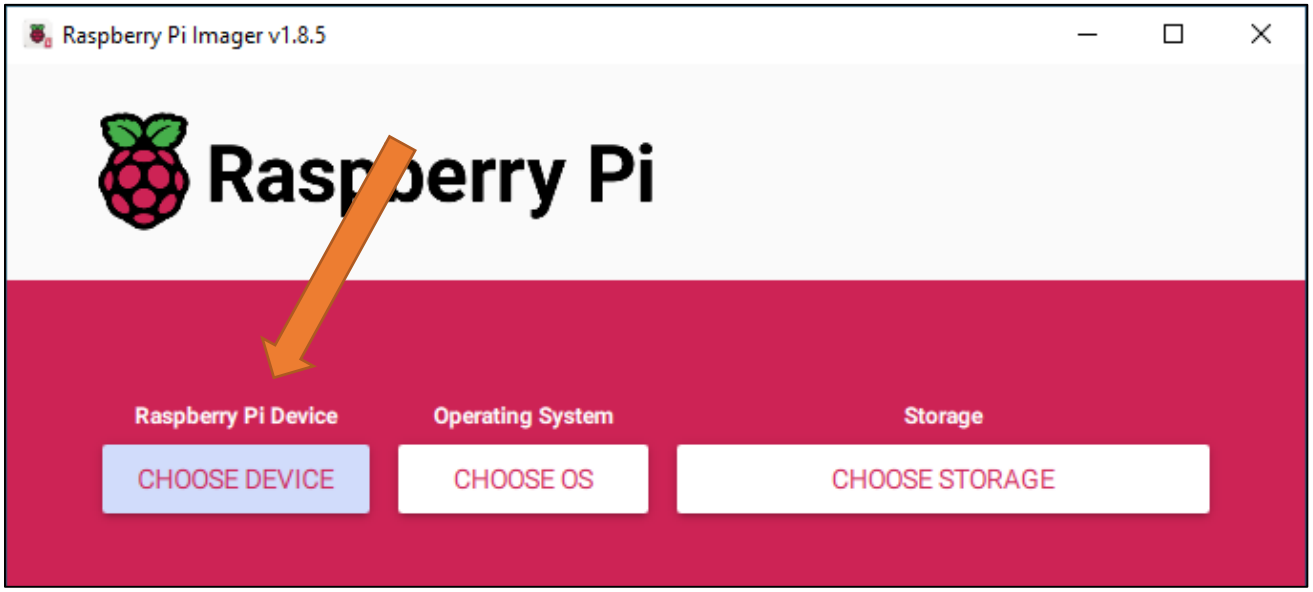


And then the zip file is downloaded.

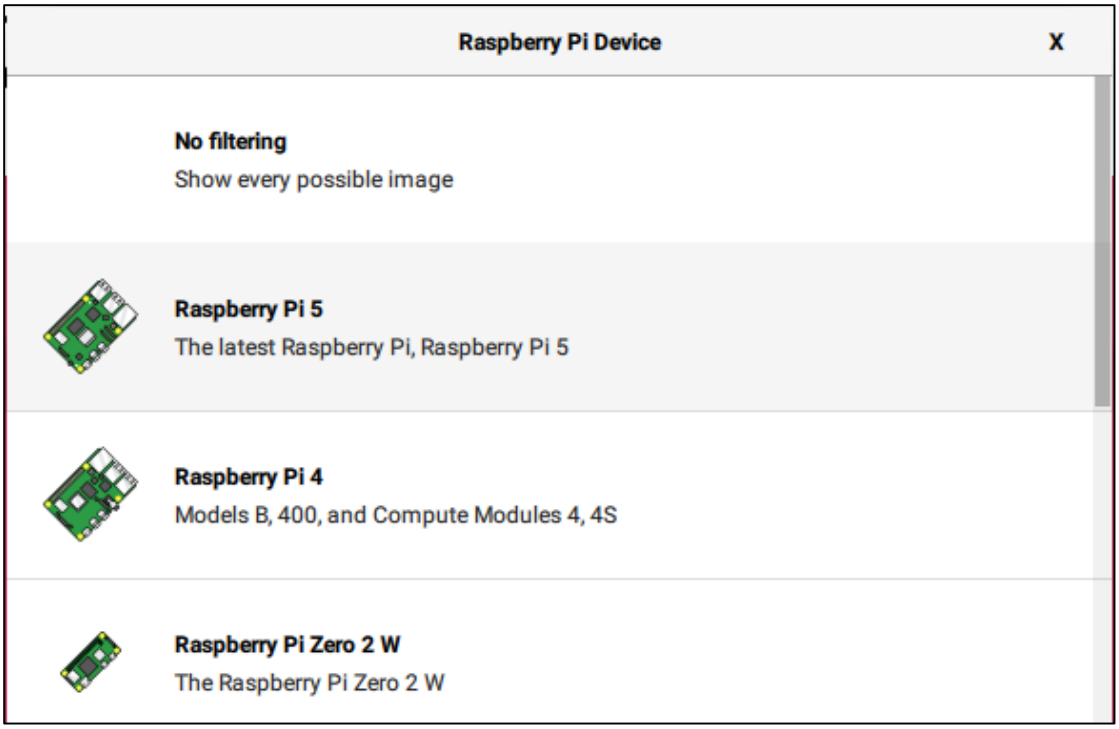
Write System to Micro SD Card

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

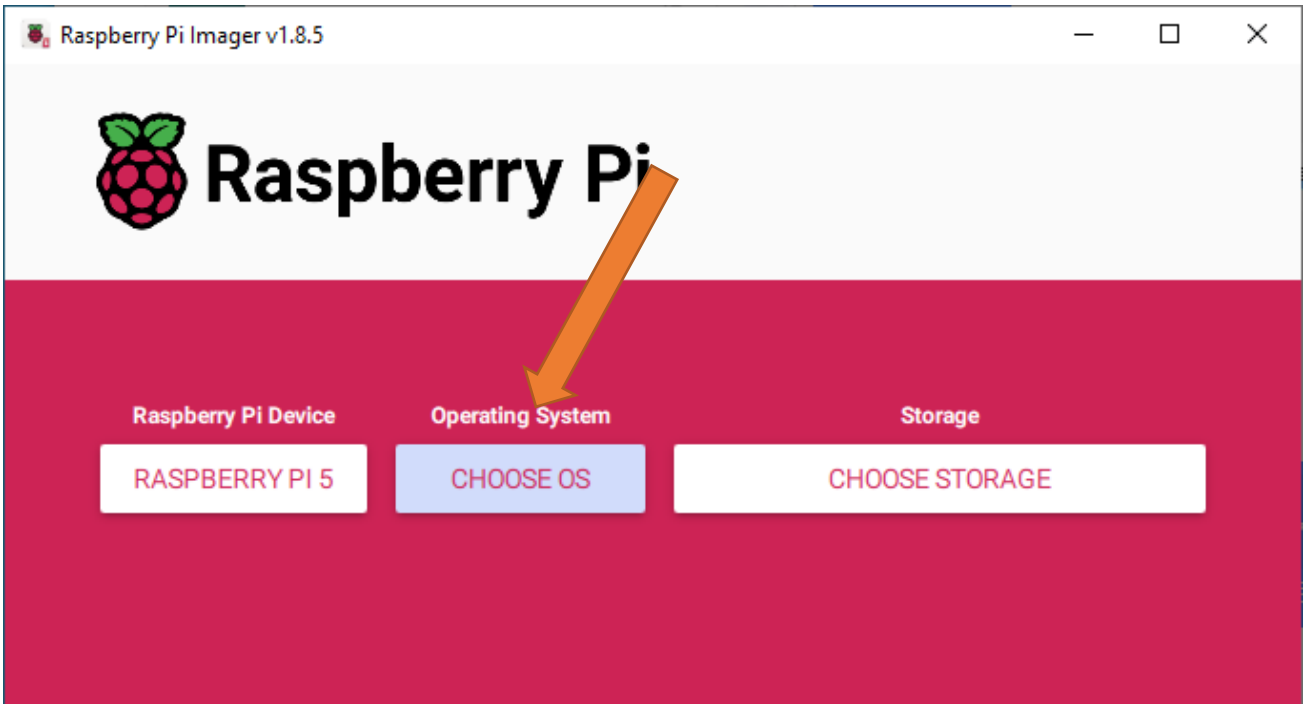
Then open imager toll. Clicked Choose Device.



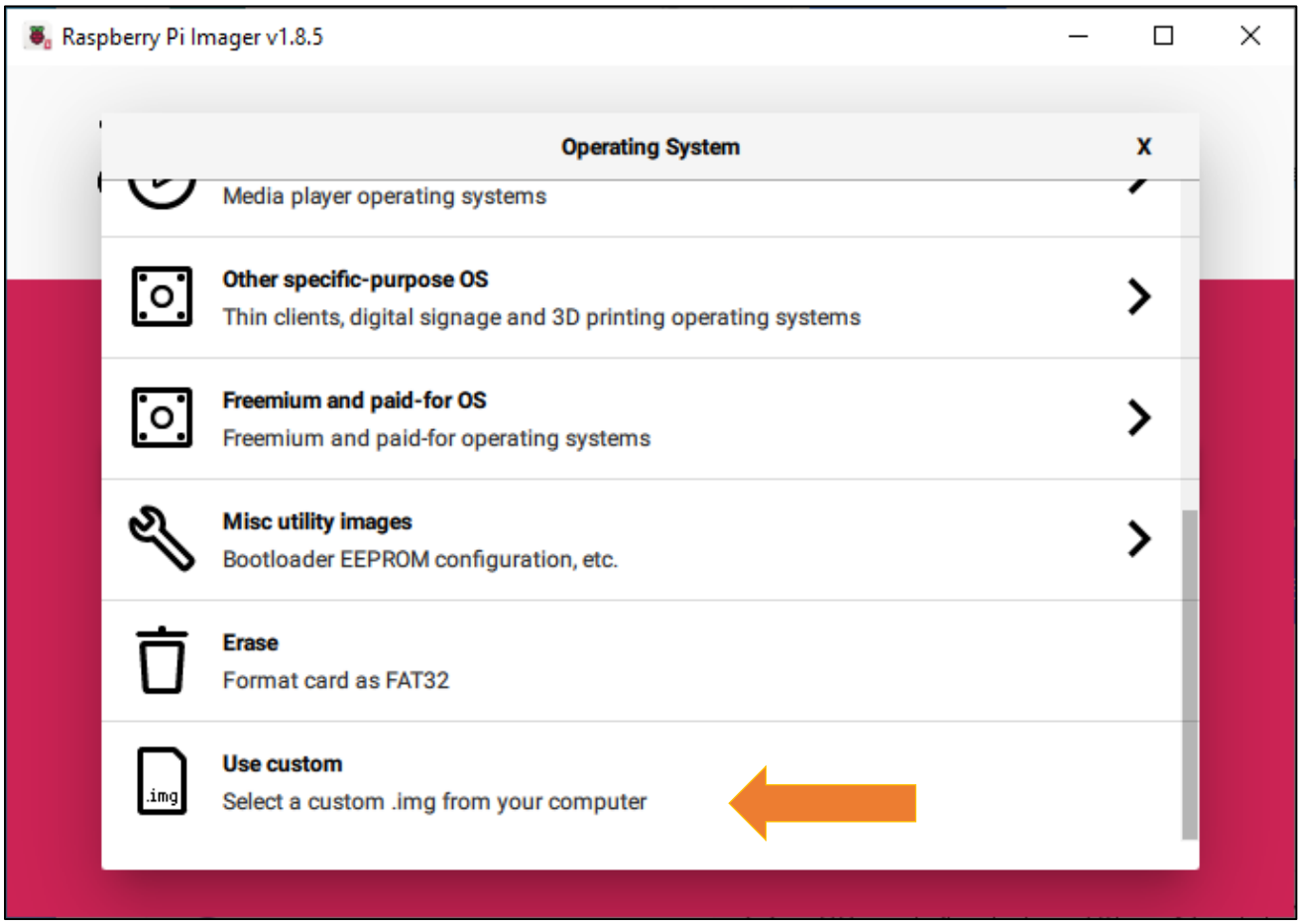
Select a Raspberry PI Device based on your Raspberry PI version. It will help us filter out the right version of the system for the Raspberry PI.



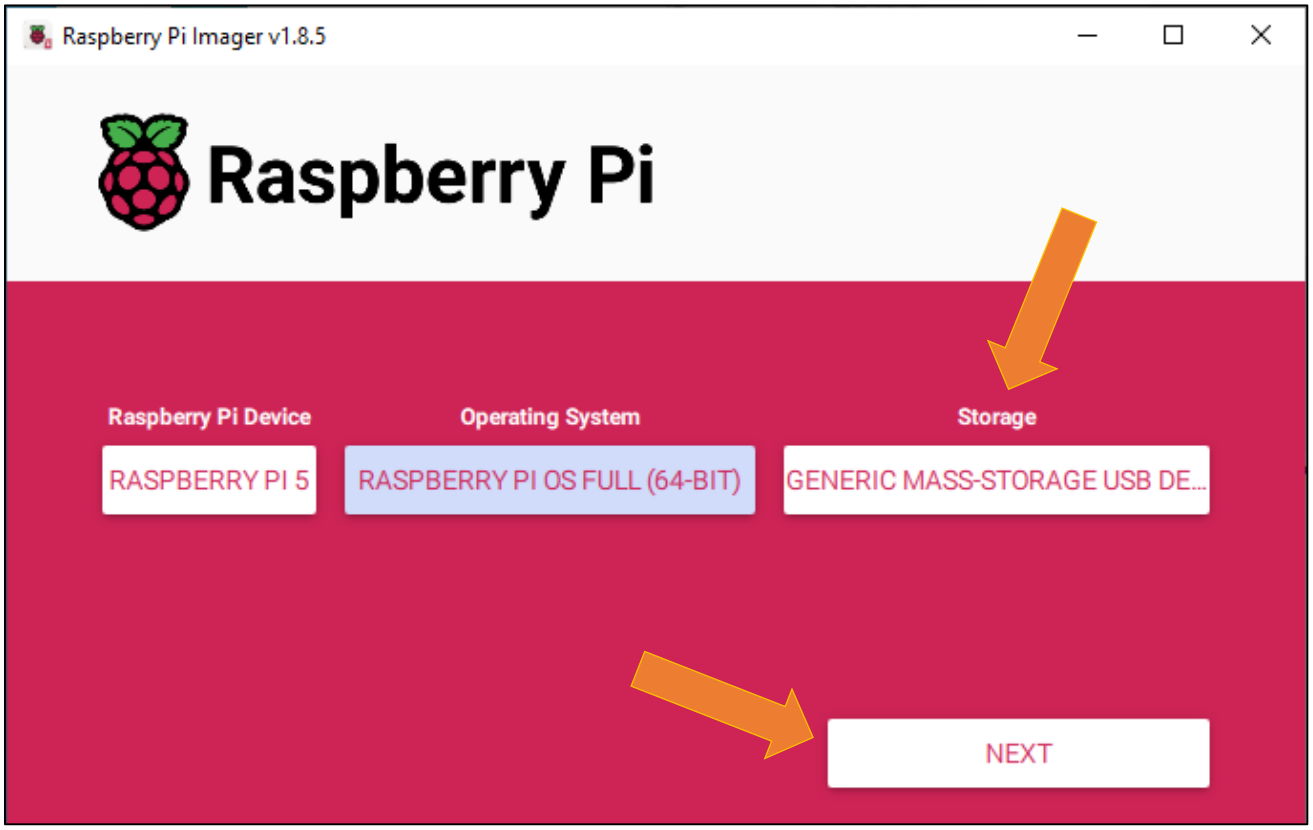
Clicked Operating System.



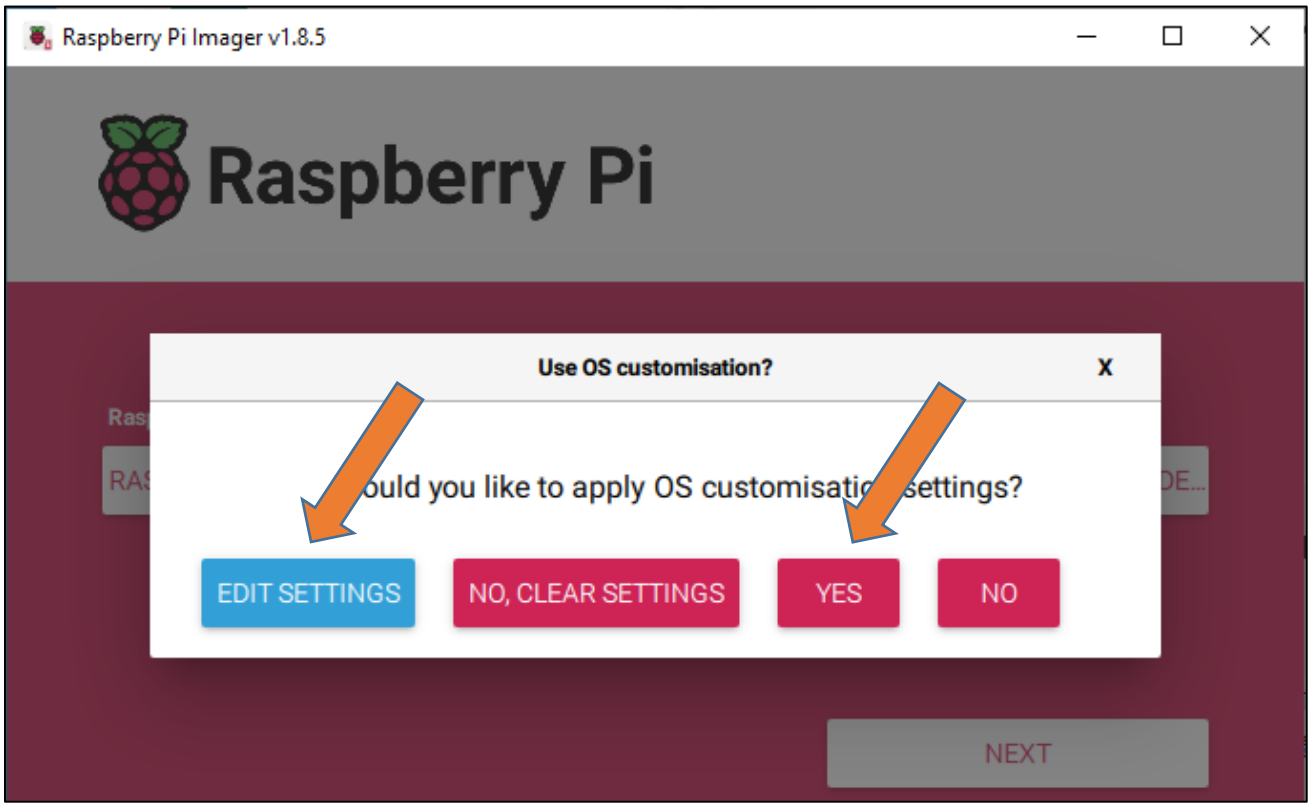
Choose system that you just downloaded in Use custom.



Choose the SD card. Then click "Next".

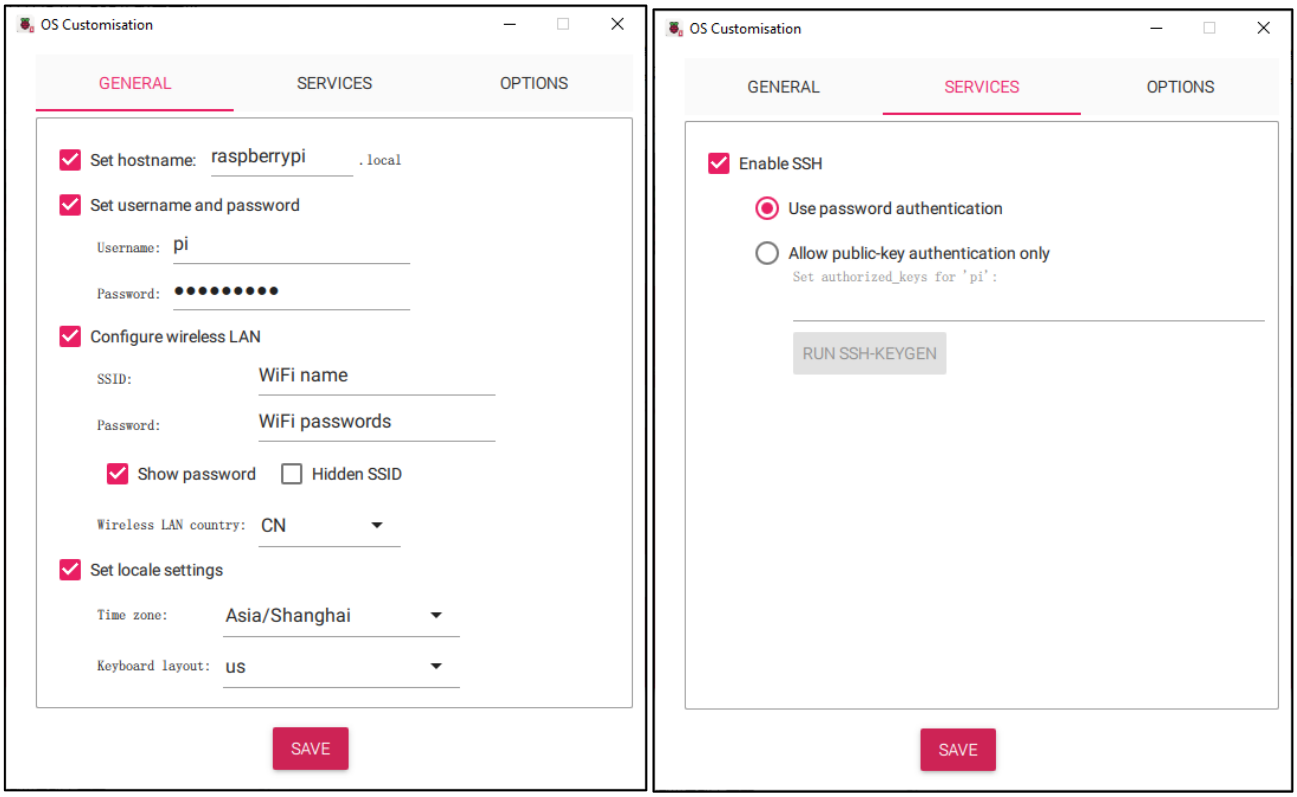


You can configure the Raspberry PI according to your needs.

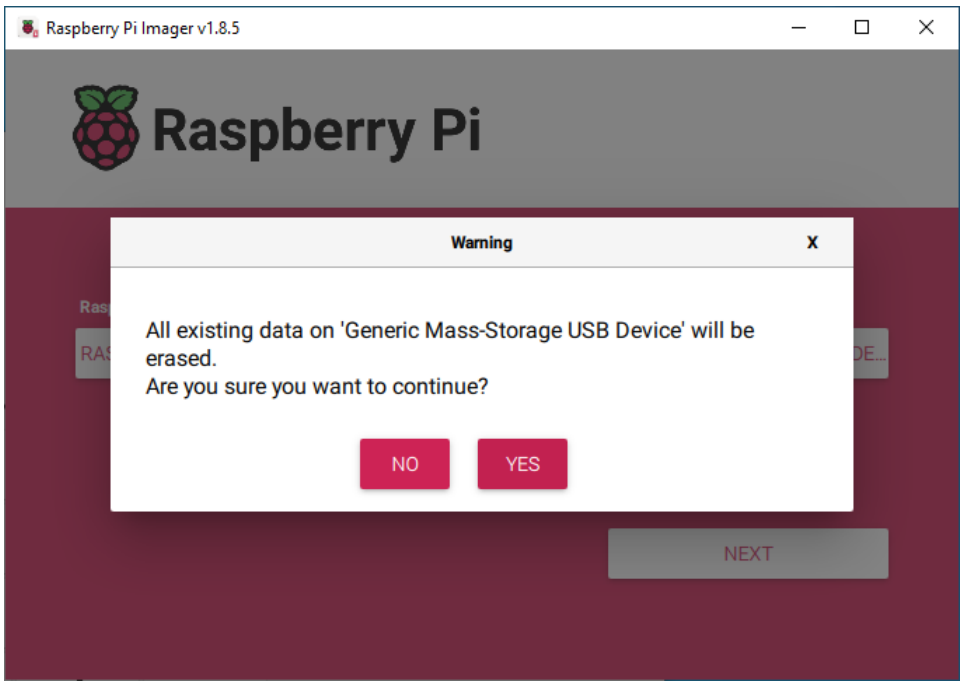


Enable ssh and configure WiFi

On the GENERAL screen, configure your information based on your actual situation.
Enable SSH on the SERVICES page.

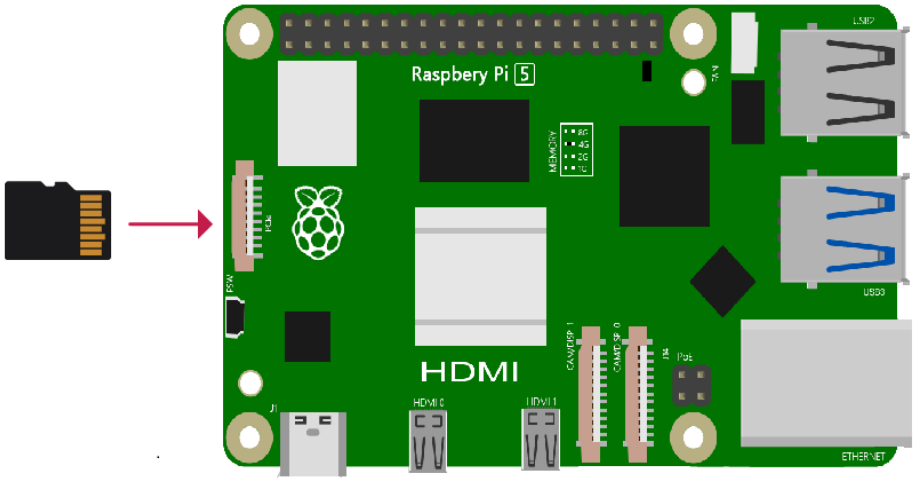


Click Save, in the new screen, click Yes, wait for SD to brush into the Raspberry system.



Insert SD card

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



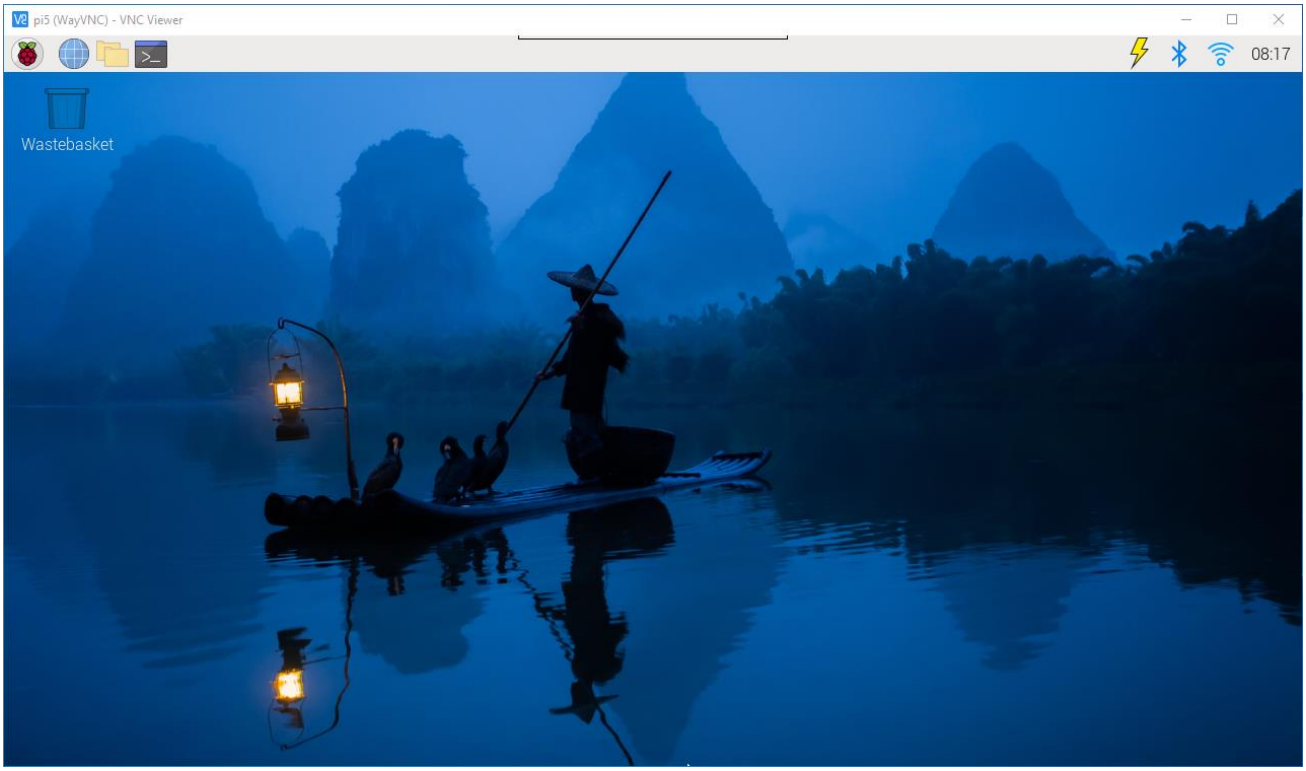
Connect to the power supply and wait for the Raspberry PI to turn on.

Getting Started with Raspberry Pi

Monitor desktop

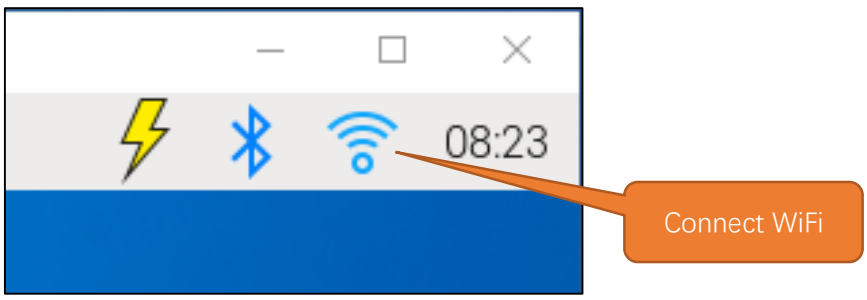
If you do not have a spare monitor, please skip to next section Remote desktop & VNC. If you have a spare monitor, please follow the steps in this section.

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 5, 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.

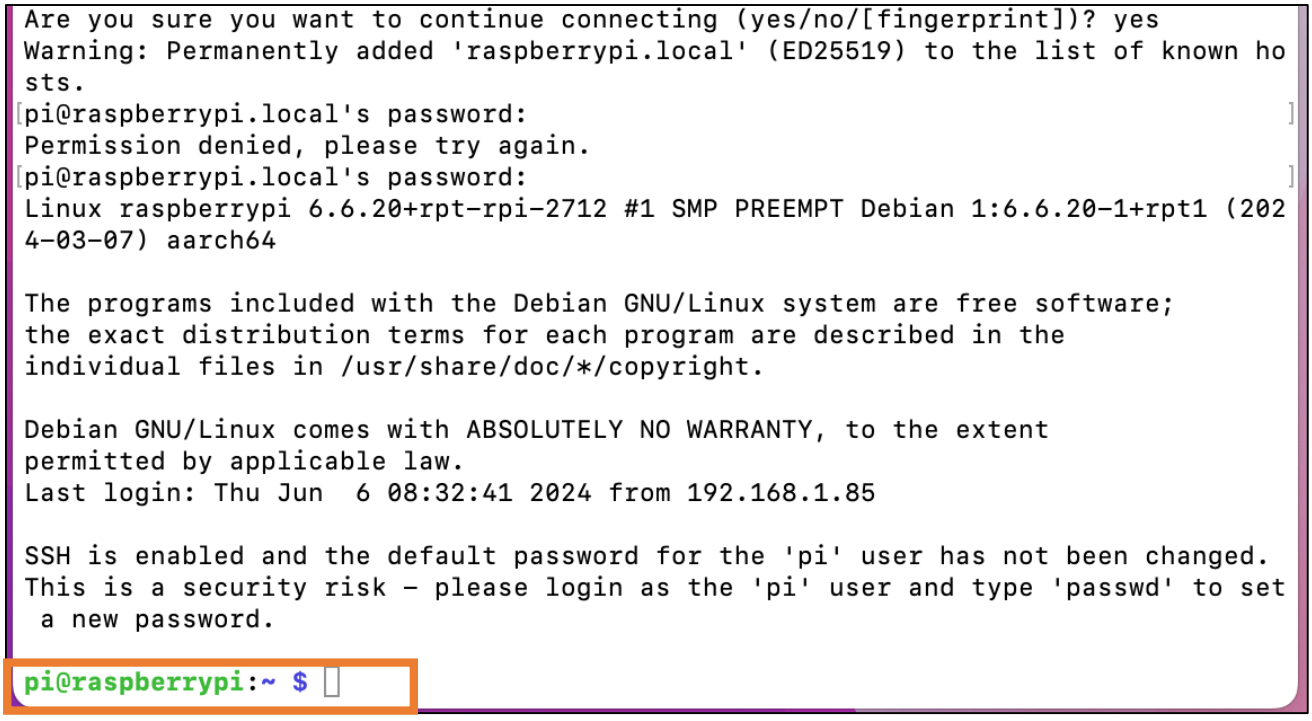
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

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.



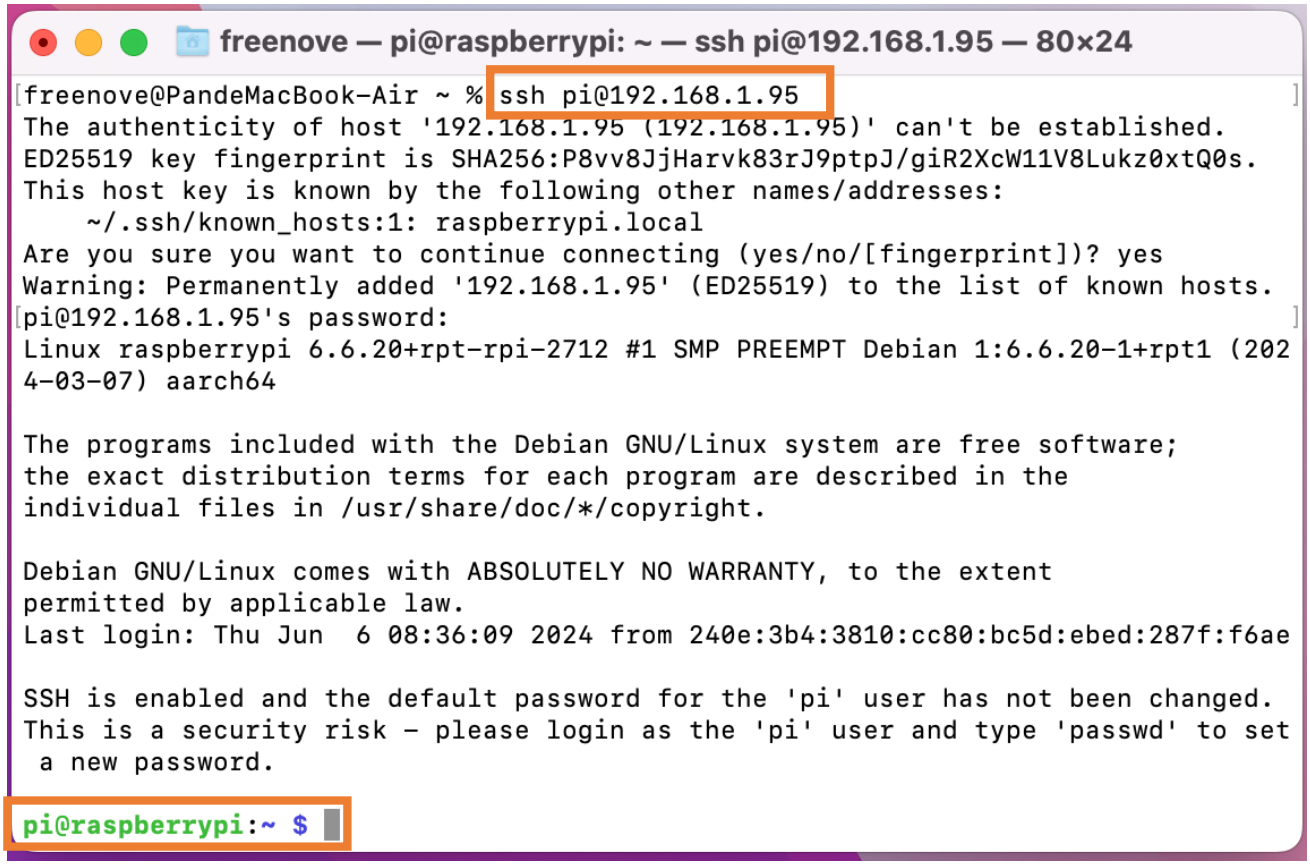
You can also use the IP address to log in Pi.

Enter **router** client to **inquiry IP address** named "raspberry pi". For example, I have inquired to **my RPi IP address, and it is "192.168.1.95".**

Open the terminal and type following command.

```
ssh pi@192.168.1.95
```

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



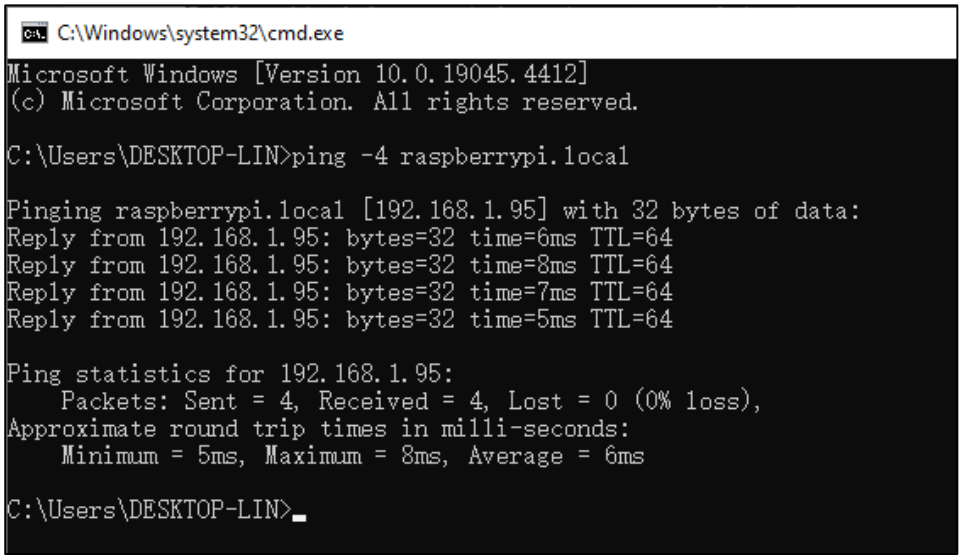
Then you can skip to VNC Viewer.

Windows OS Remote Desktop

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

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

```
ping -4 raspberrypi.local
```



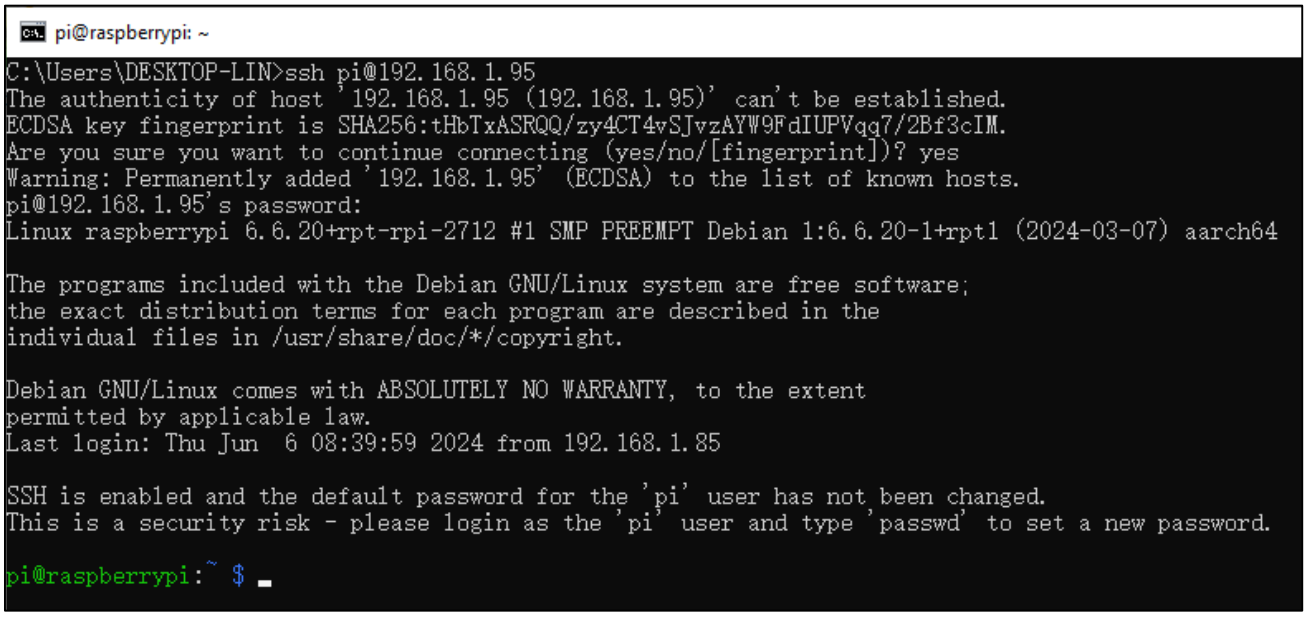
Then 192.168.1.147 is my Raspberry Pi IP.

Or enter **router** client to **inquiry IP address** named "**raspberrypi**". For example, I have inquired to **my RPi IP address, and it is "192.168.1.95".**

```
ssh pi@xxxxxxxxxxx(IP address)
```

Enter the following command:

```
ssh pi@192.168.1.95
```



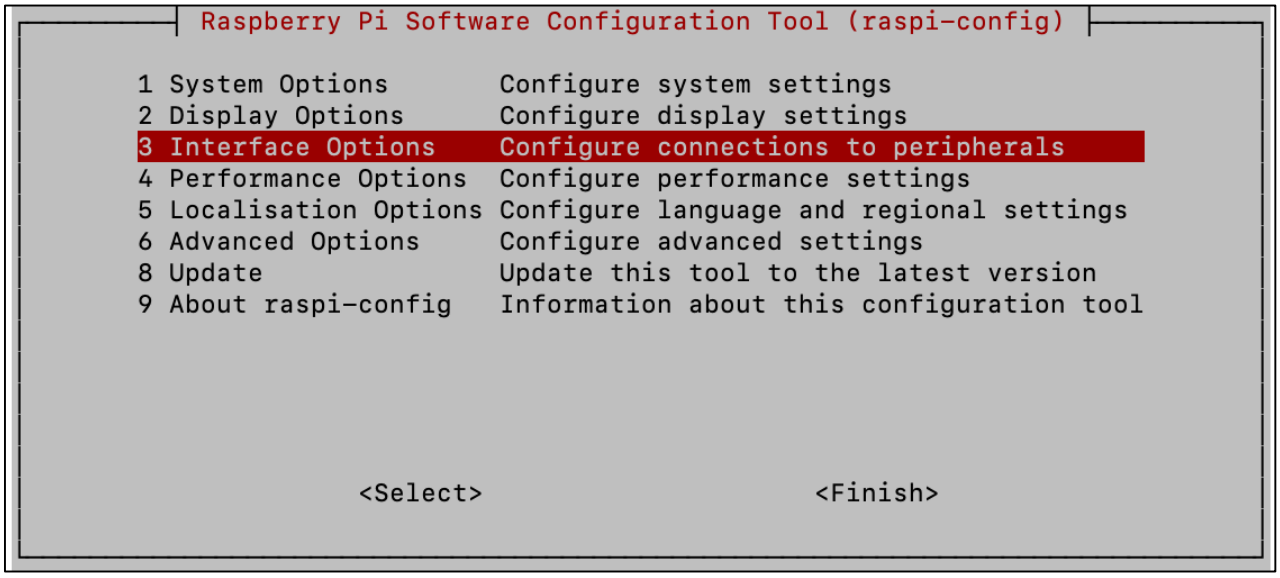
VNC Viewer & VNC

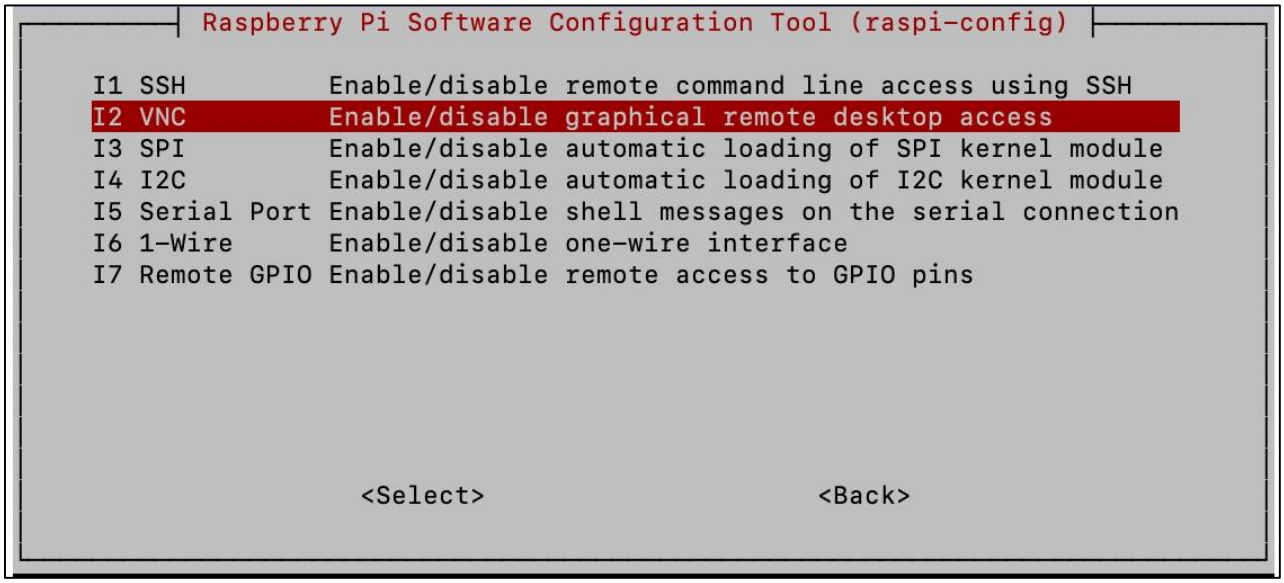
Enable VNC

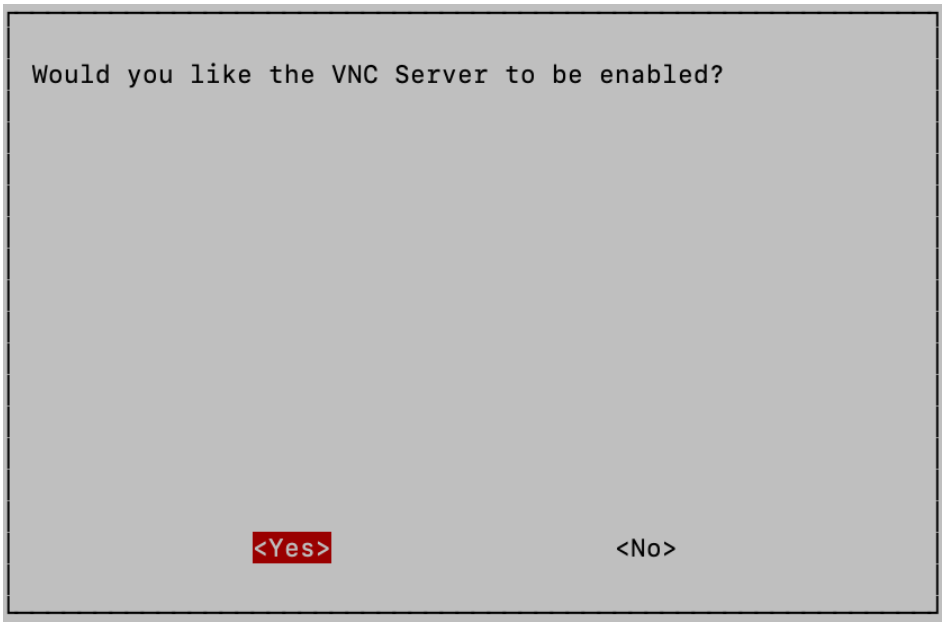
Type the following command. And select Interface Options→P5 VNC → Enter→Yes→OK. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

```
sudo raspi-config
```

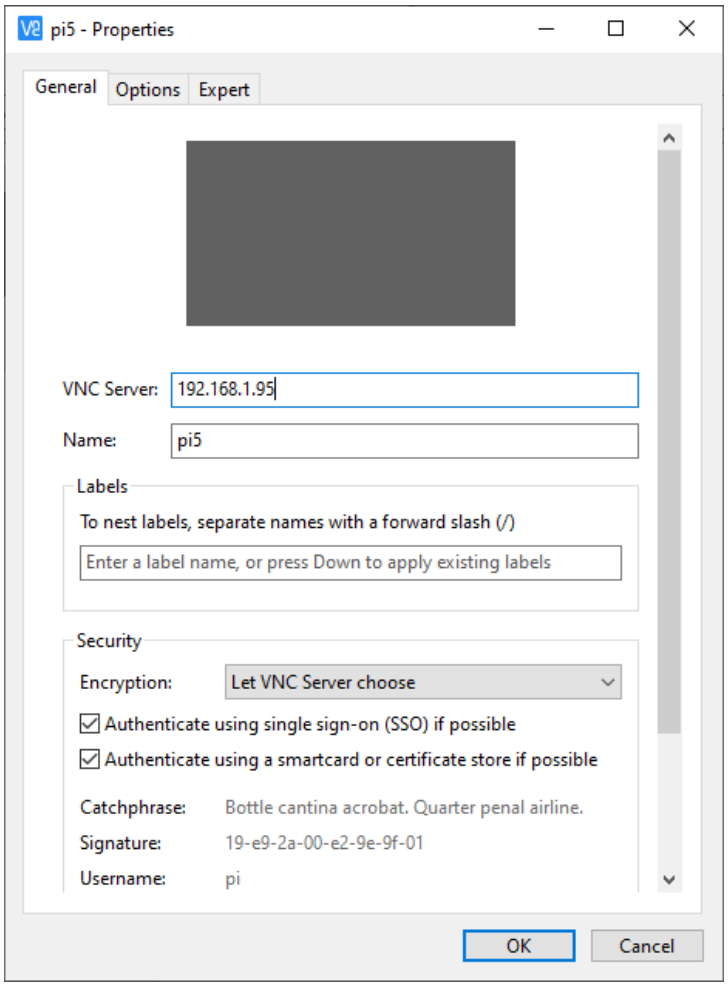
```
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ sudo raspi-config
```





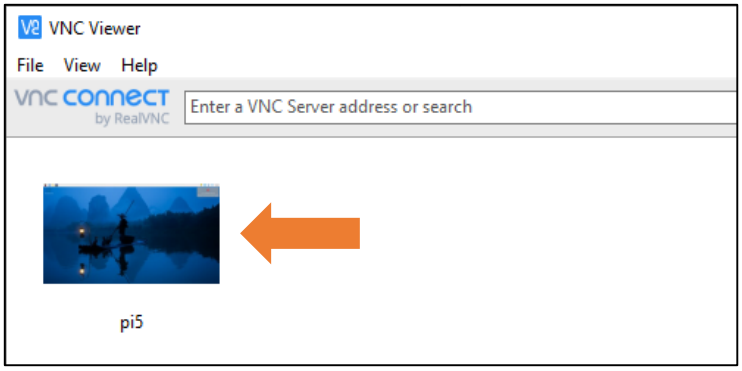


Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.

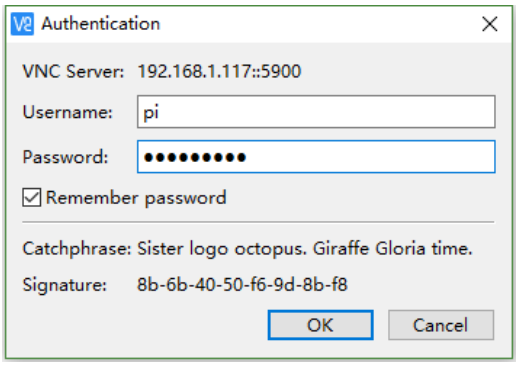


Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

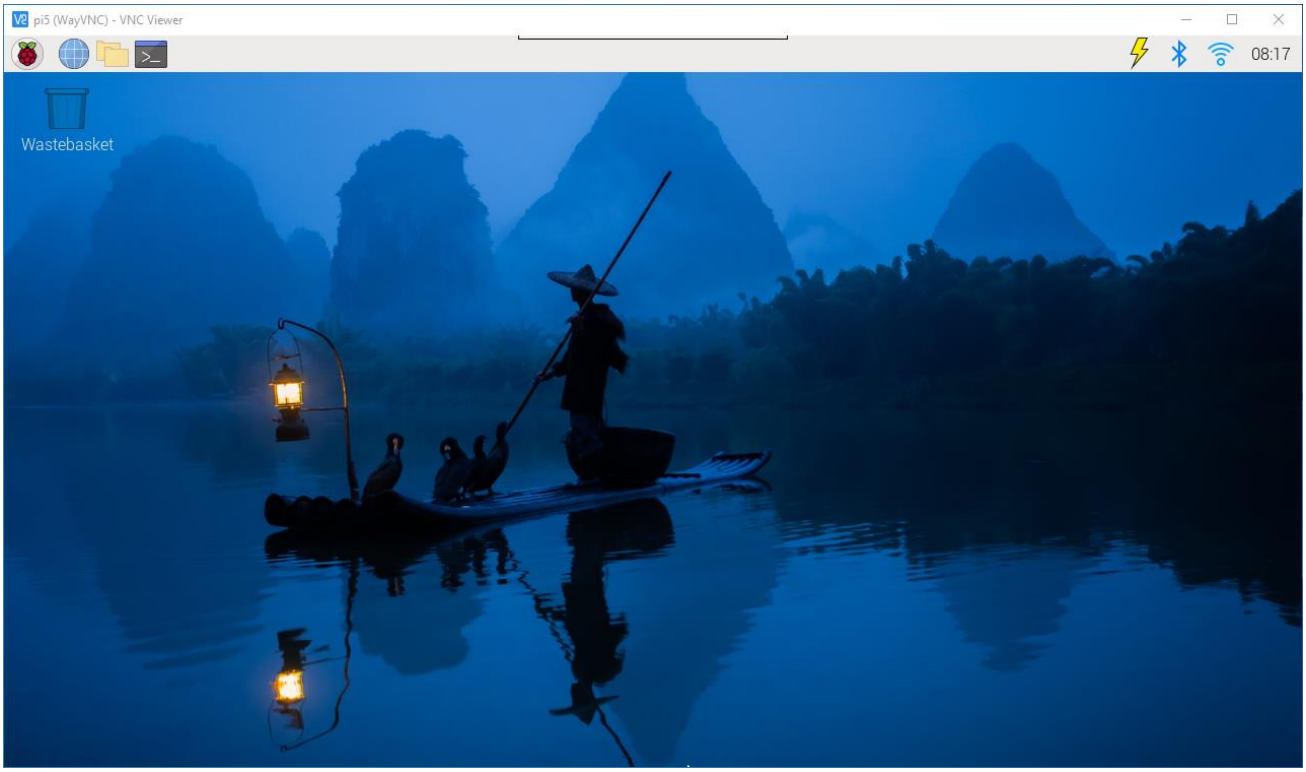
Then on the VNC Viewer panel, double-click new connection you just created,



and the following dialog box pops up.

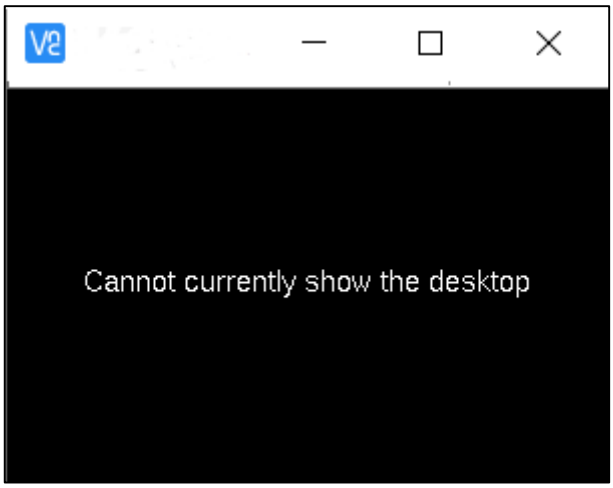


Enter username: **pi** and Password: **raspberry**. And click OK.



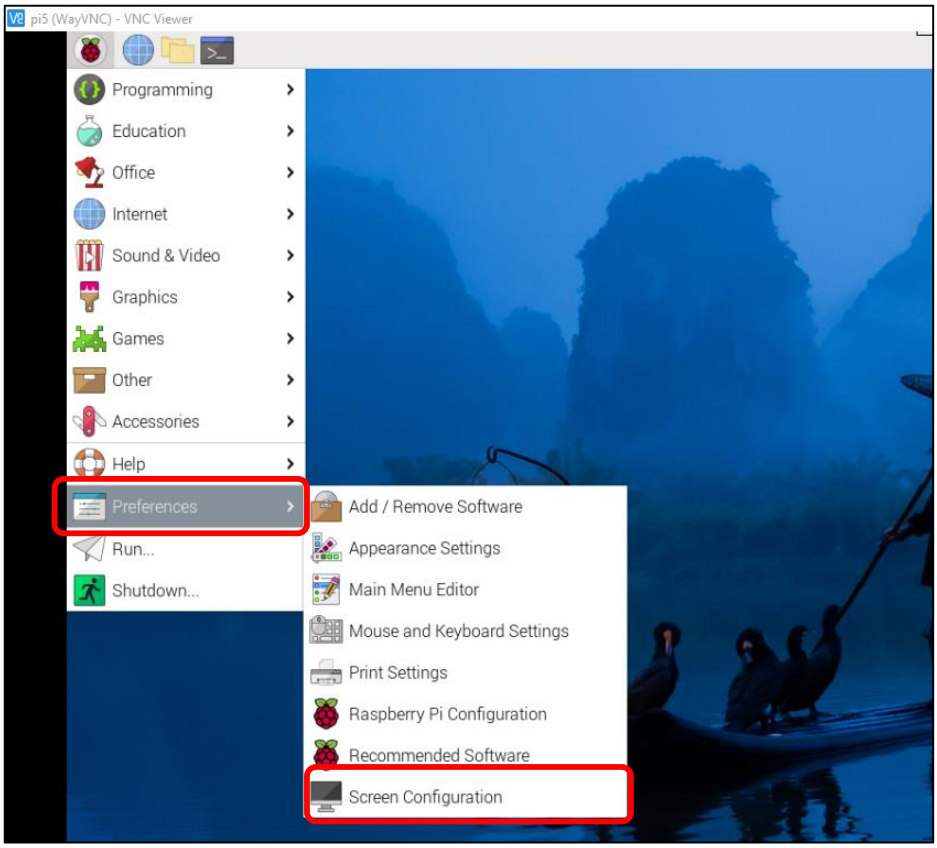
Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

If there is black window, please set resolution.

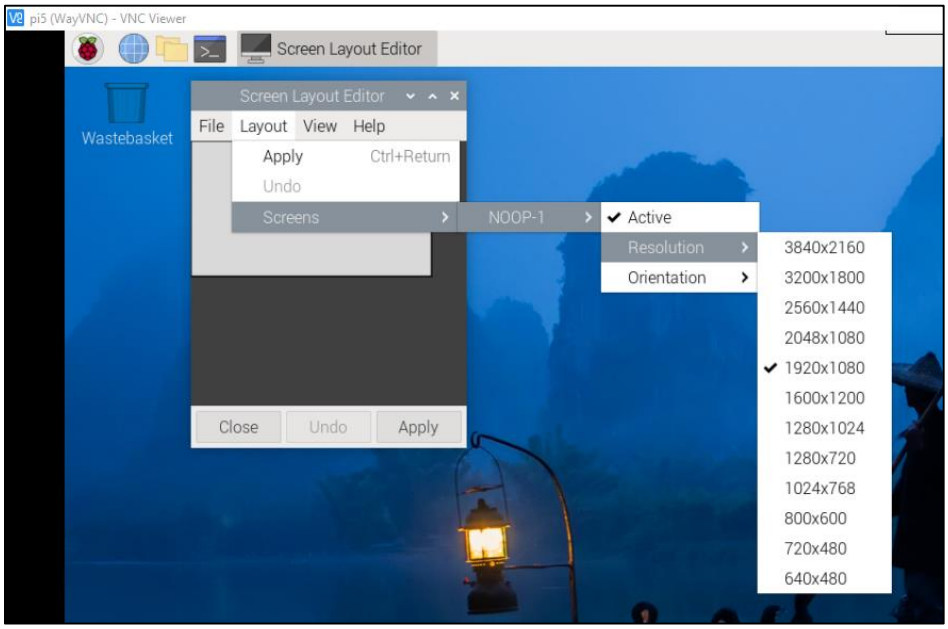


Set Resolution

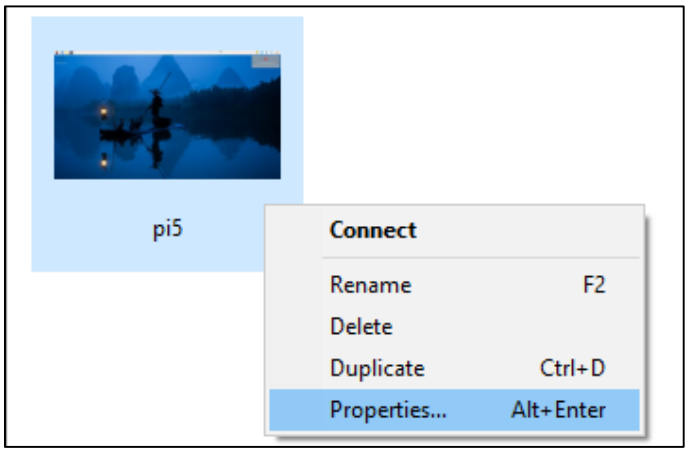
You can also set other resolutions.

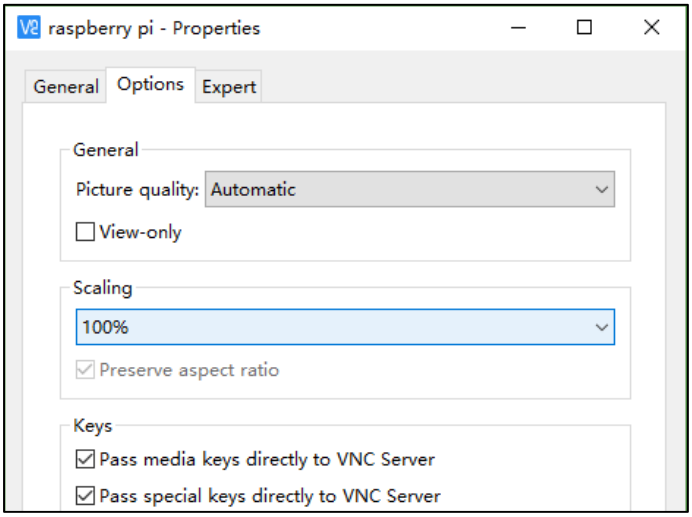


If you don't know what resolution to set properly, you can try 1920x1080.



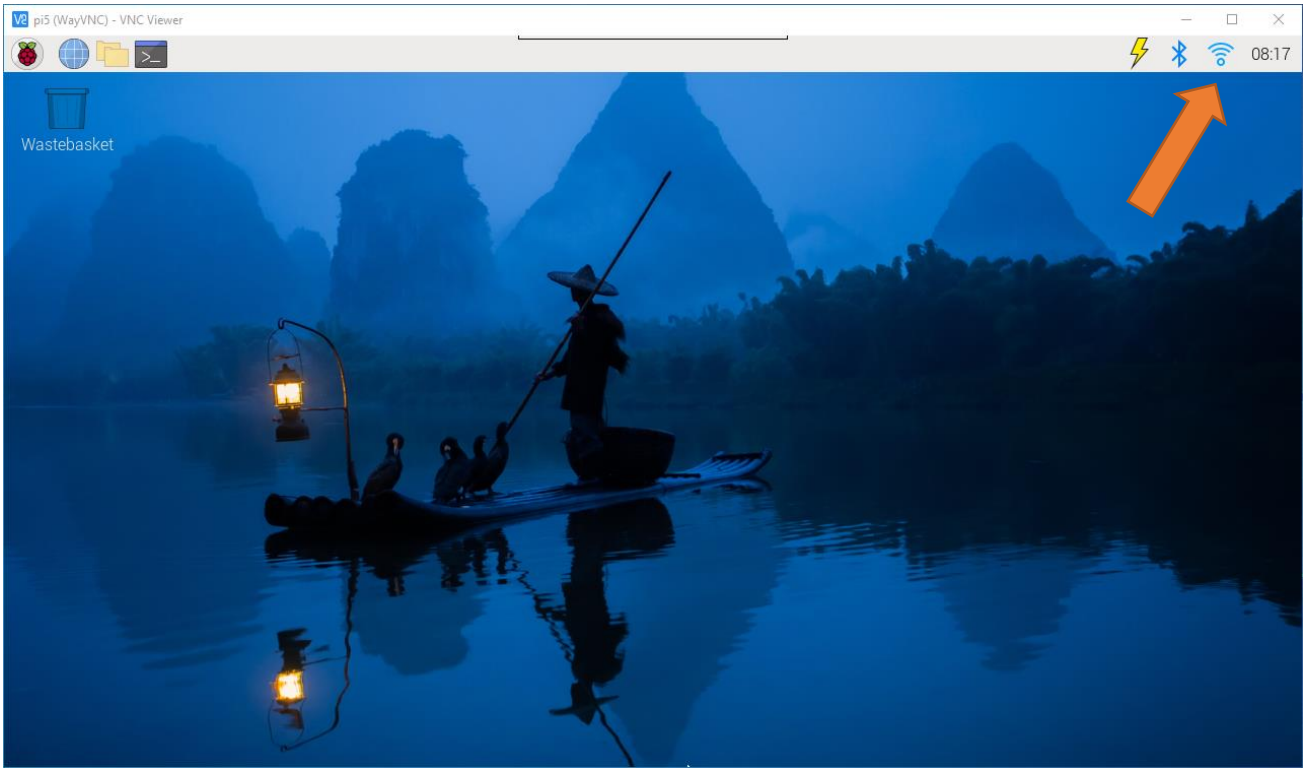
In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. And select Properties->Options label->Scaling. Then set proper scaling.





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

Raspberry Pi 5/4B/3B+/3B integrates a Wi-Fi adaptor.If you did not connect Pi to WiFi. You can connect it to wirelessly control the robot.

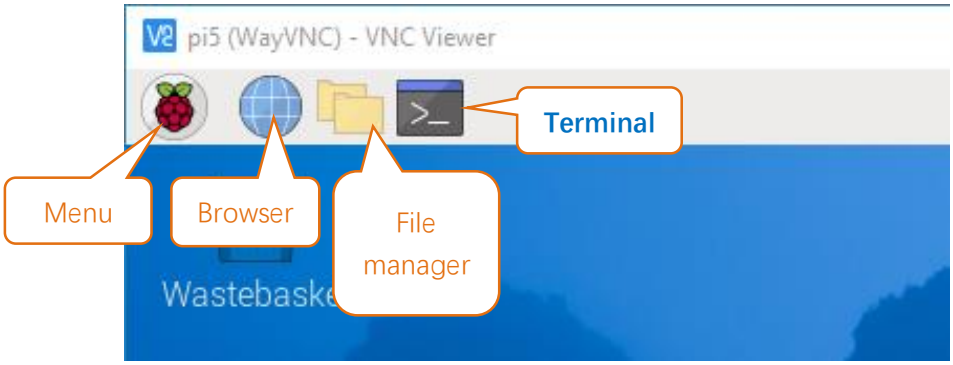


Chapter 0 Preparation

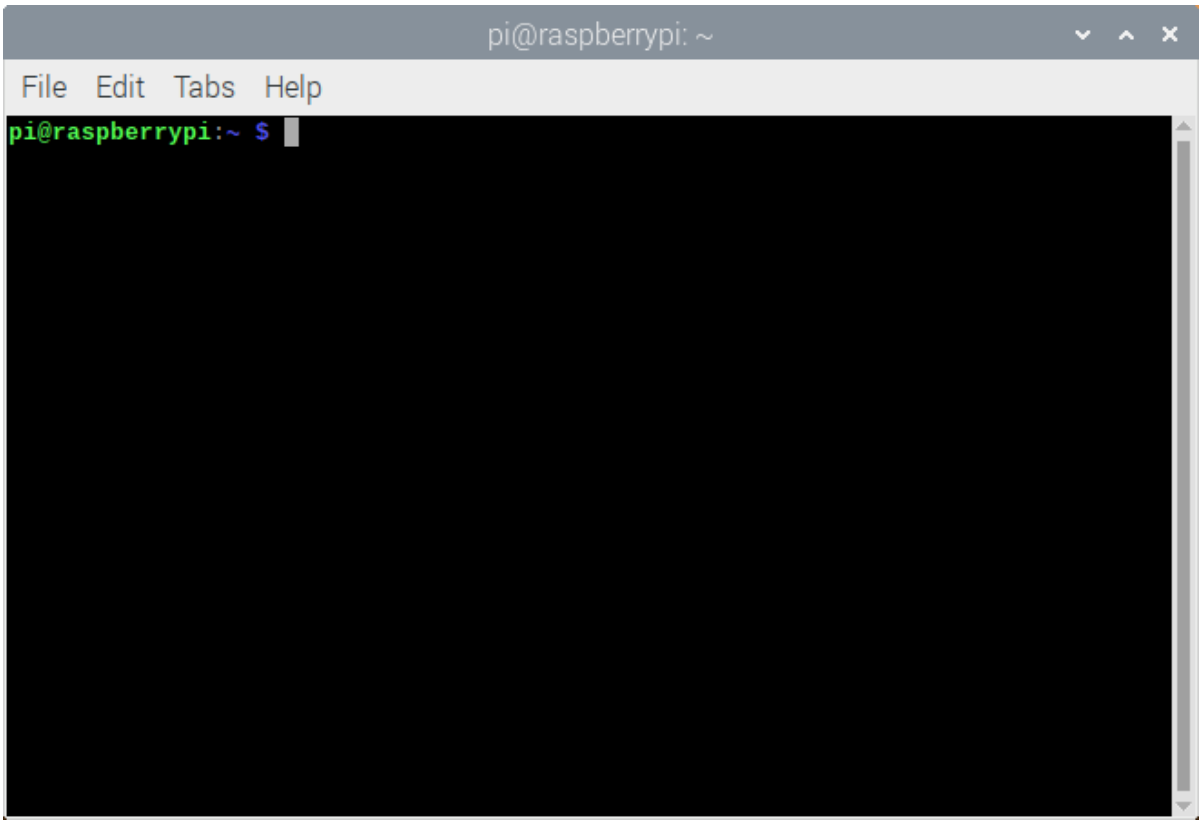
Why "Chapter 0"? Because in program code the first number is 0. We choose to follow this rule. In this chapter, we will do some necessary foundational preparation work: Start your Raspberry Pi and install some necessary libraries.

Linux Command

Raspberry Pi OS is based on the Linux Operation System. Now we will introduce you to some frequently used Linux commands and rules.
First, open the Terminal. All commands are executed in Terminal.

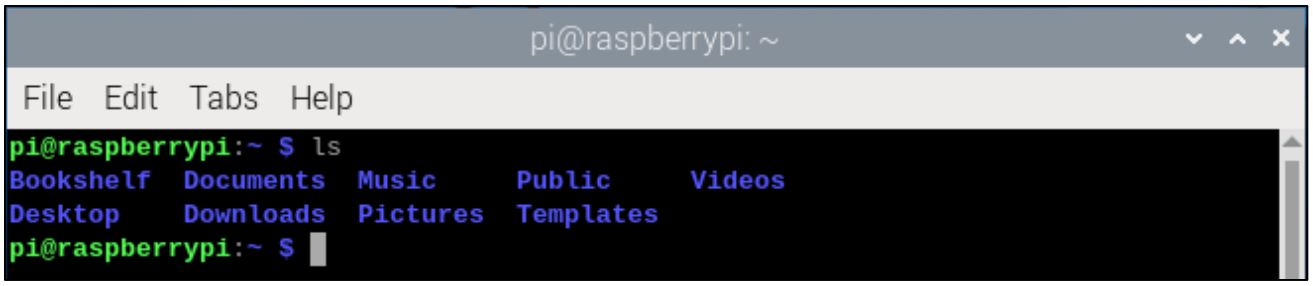


When you click the Terminal icon, following interface appears.



Note: The Linux is case sensitive.

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



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

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

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

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

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

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

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

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

There are many commands, which will come later. For more details about commands. You can refer to:
http://www.linux-commands-examples.com

Shortcut Key

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

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

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

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

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

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

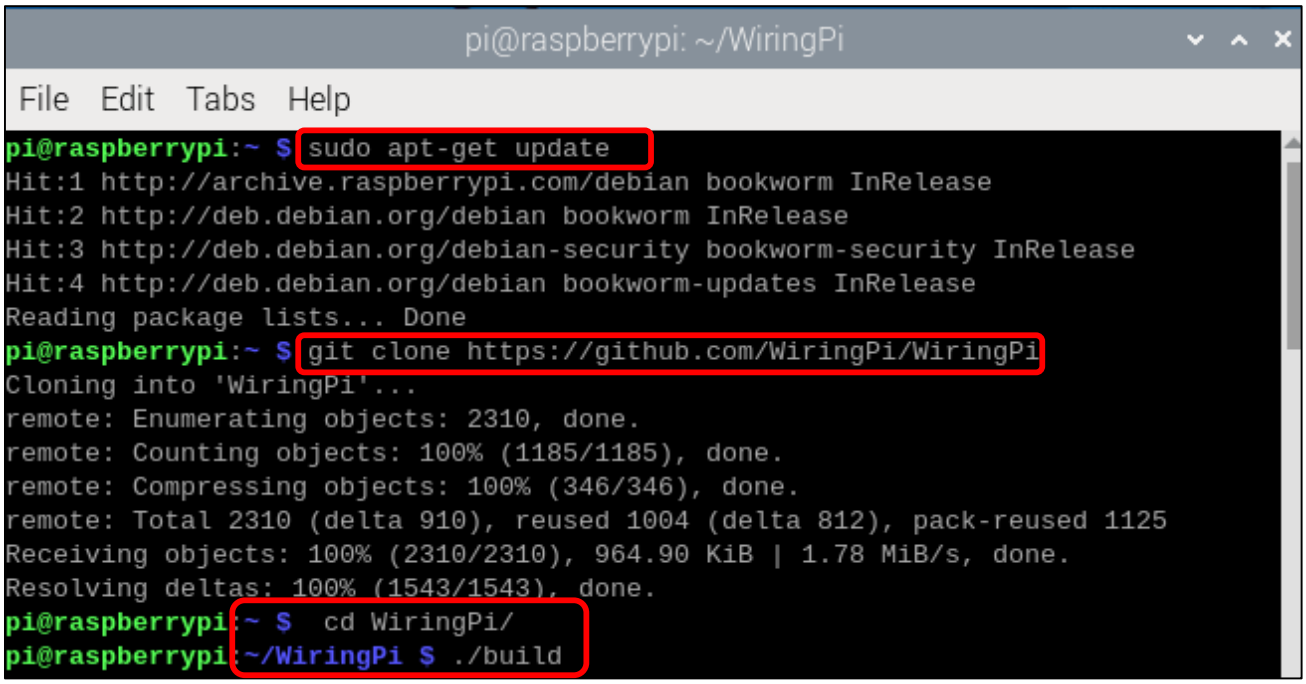
Install WiringPi

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

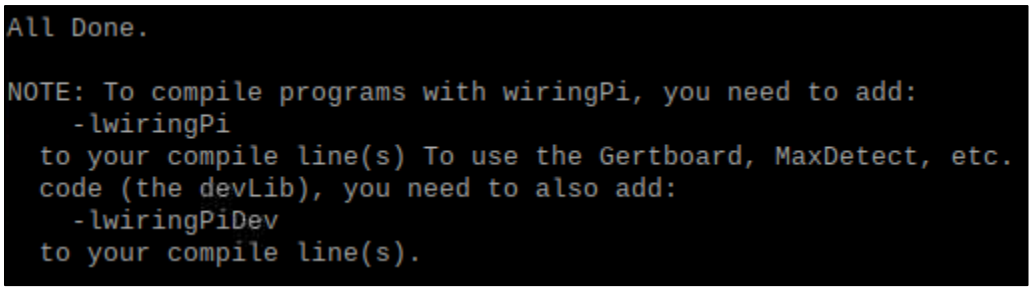
WiringPi Installation Steps

To install the WiringPi library, please open the Terminal and then follow the steps and commands below.
Note: For a command containing many lines, execute them one line at a time.
Enter the following commands **one by one** in the **terminal** to install WiringPi:

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



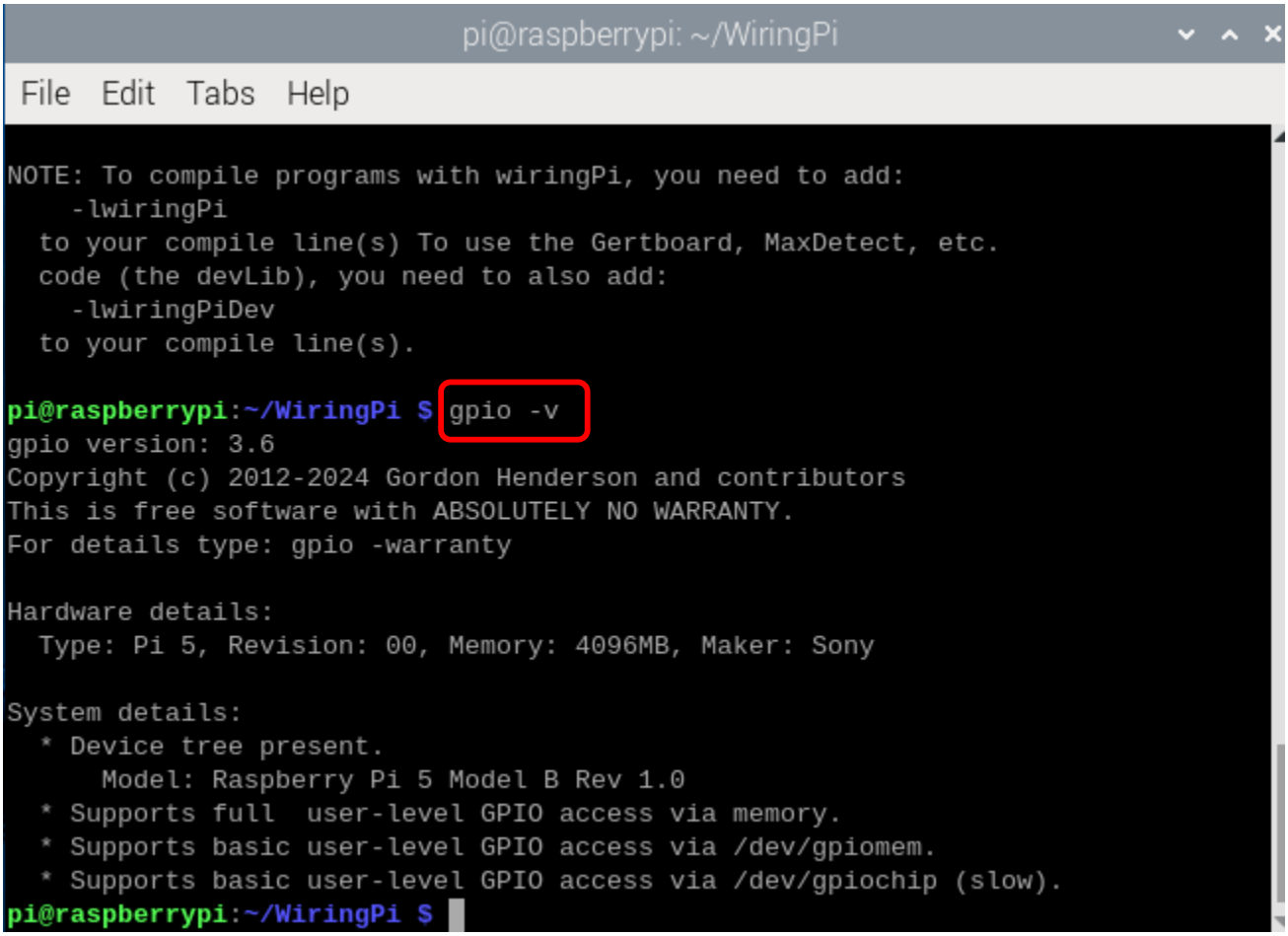
The following figure shows the successful installation.



Run the gpio command to check the installation:

```
gpio -v
```

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



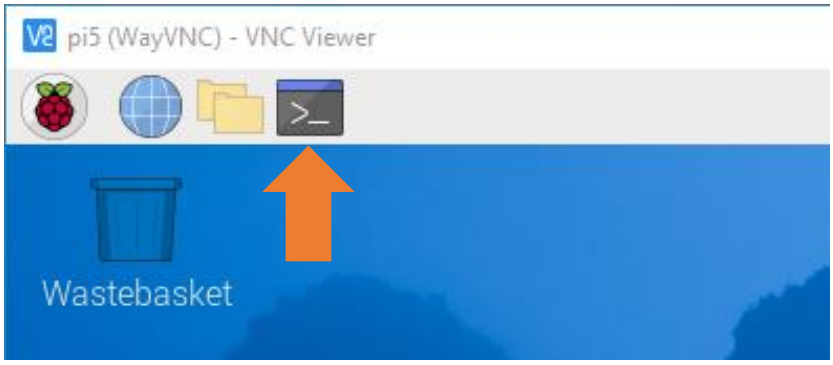
Obtain the Project Code

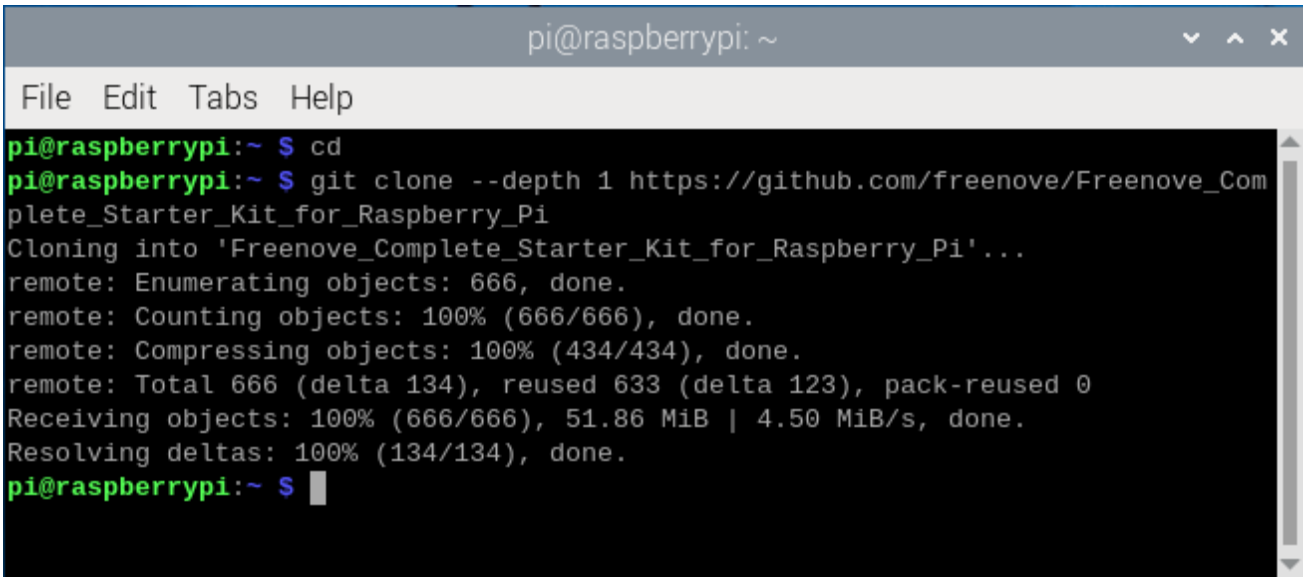
After the above installation is completed, you can visit our official website (http://www.freenove.com) or our GitHub resources at (https://github.com/freenove) to download the latest available project code. We provide both **C** language and **Python** language code for each project to allow ease of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

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



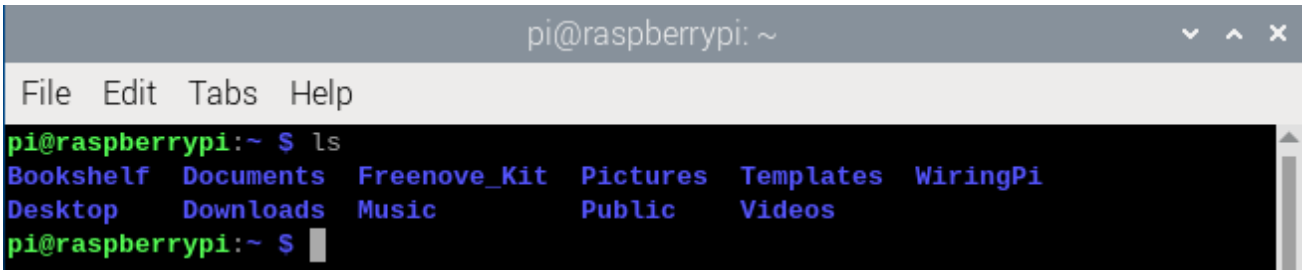


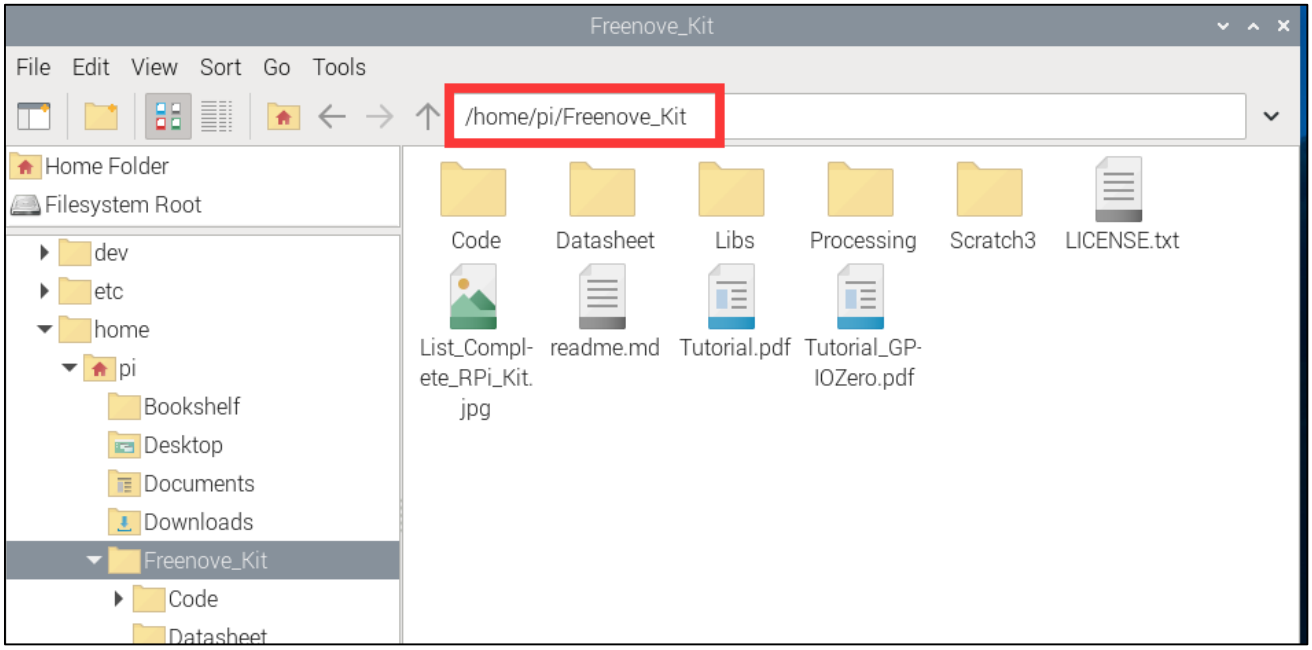
After the download is completed, a new folder "Freenove_Ultrasonic_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

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

```
mv Freenove_Ultrasonic_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

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





Chapter 1 LED

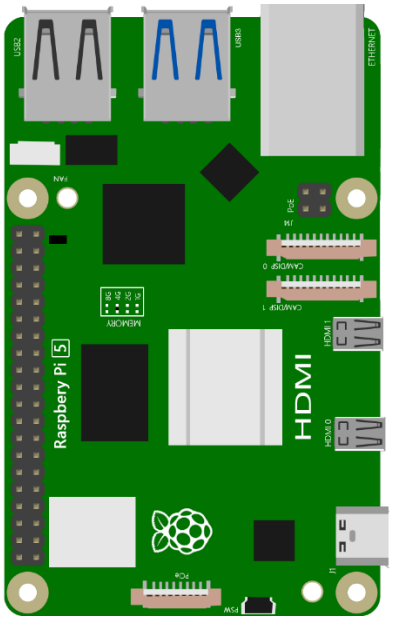
This chapter is the Start Point in the journey to build and explore RPi electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

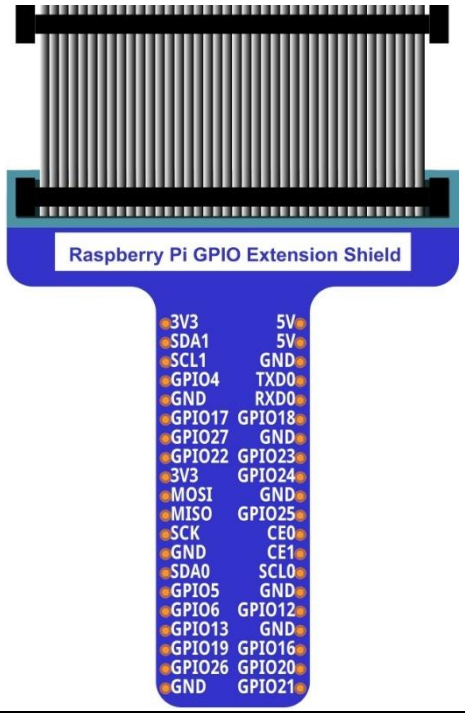
In this project, we will use RPi to control blinking a common LED.

Component List

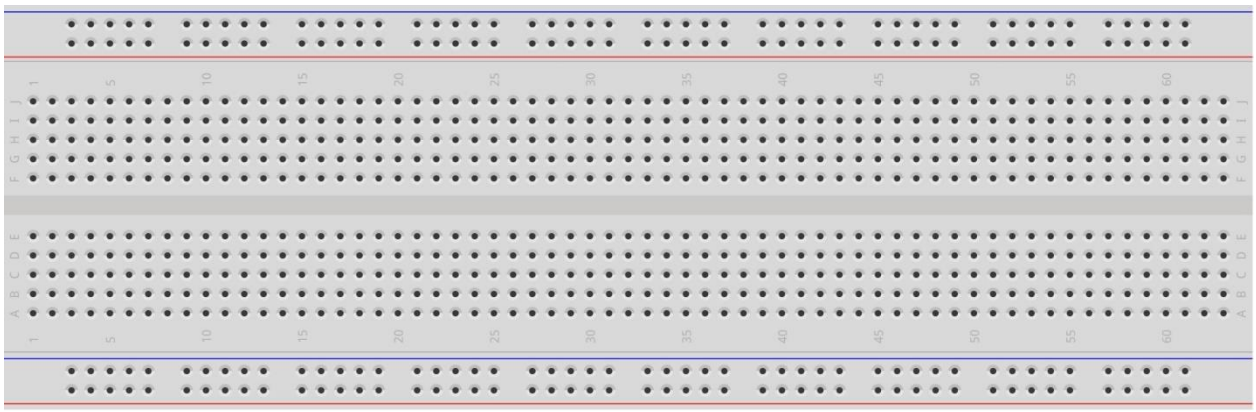
Raspberry Pi
(Recommended: Raspberry Pi 5 / 4B / 3B+ / 3B
Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)



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

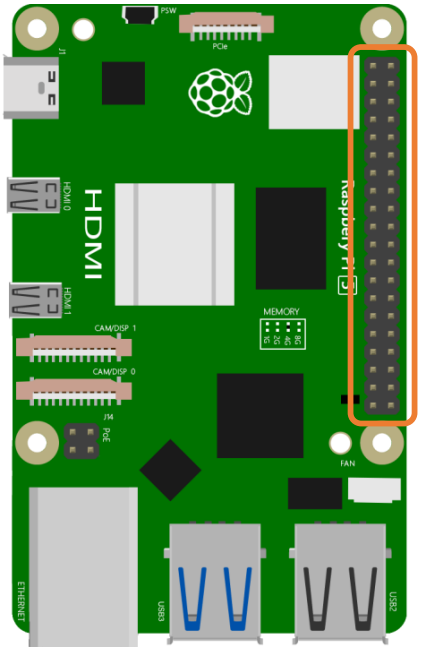
GPIO: General Purpose Input/Output. Here we will introduce the specific function of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions.

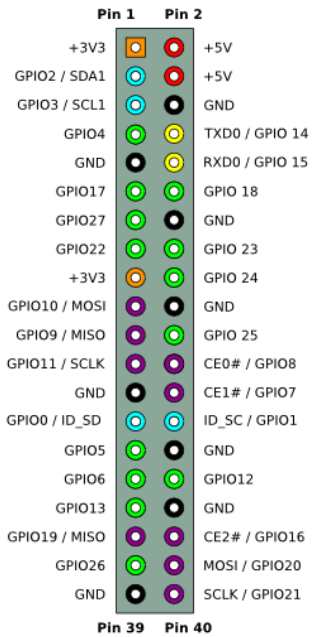
When programming GPIO pins there are 3 different ways to reference them: GPIO Numbering, Physical Numbering and WiringPi GPIO Numbering.

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:

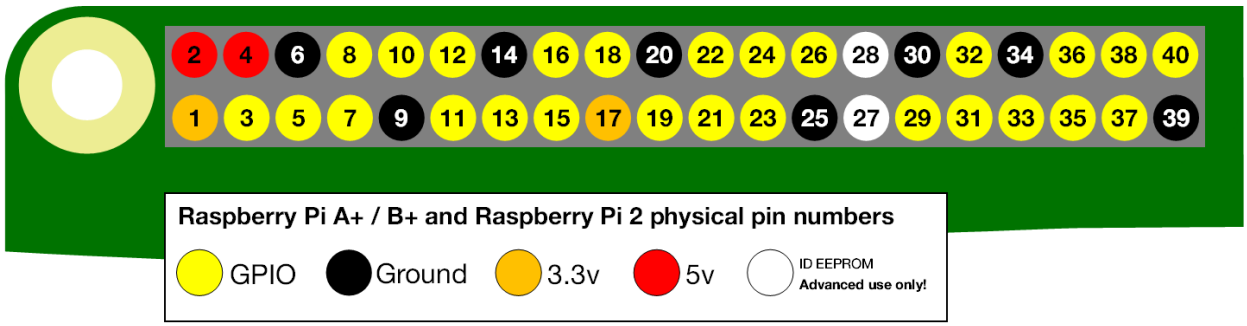




For more details about pin definition of GPIO, please refer to http://pinout.xyz/

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.

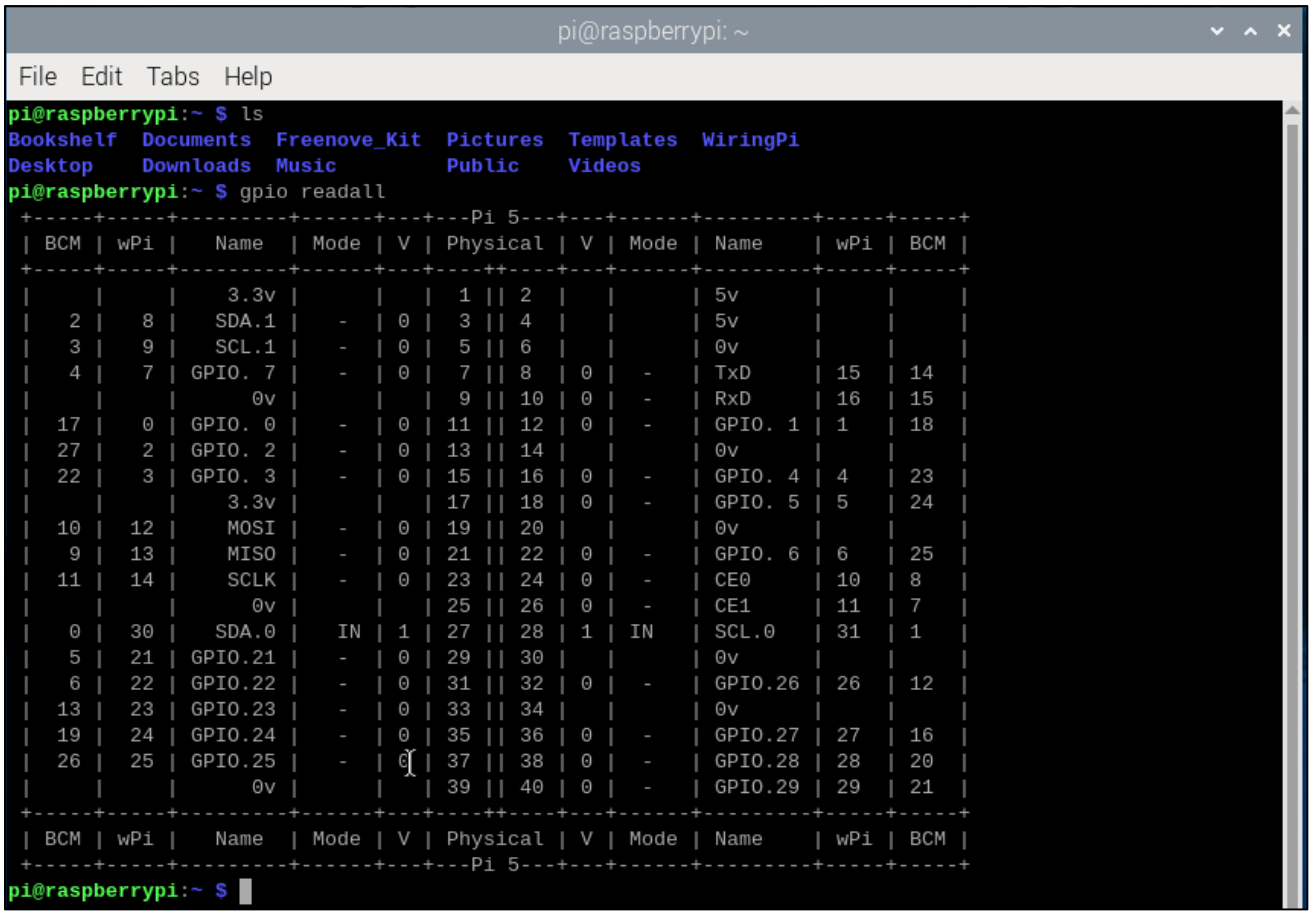
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin	
—	—	3.3v	1 | 2	5v	—	—	For Pi B
8	R1:0/R2:2	SDA	3 | 4	5v	—	—	
9	R1:1/R2:3	SCL	5 | 6	0v	—	—	
7	4	GPIO7	7 | 8	TxD	14	15	
—	—	0v	9 | 10	RxD	15	16	
0	17	GPIO0	11 | 12	GPIO1	18	1	
2	R1:21/R2:27	GPIO2	13 | 14	0v	—	—	
3	22	GPIO3	15 | 16	GPIO4	23	4	
—	—	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 more details, please refer to https://projects.drogon.net/raspberry-pi/wiringpi/pins/)

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

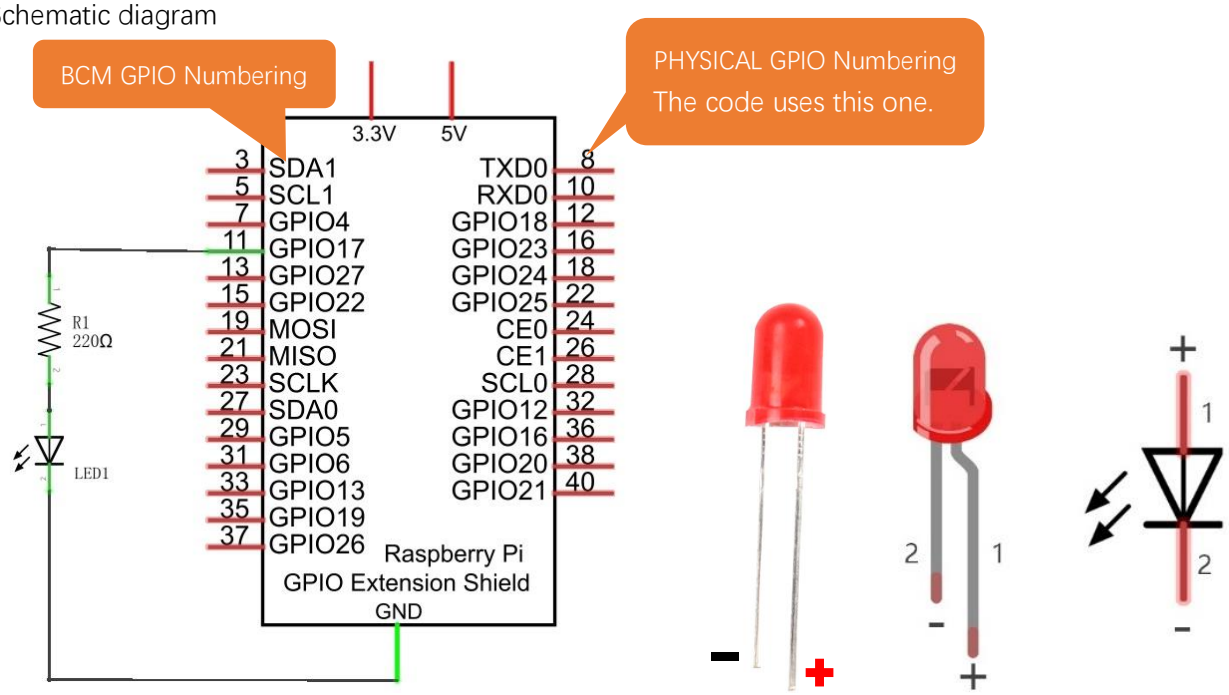
```
gpio readall
```



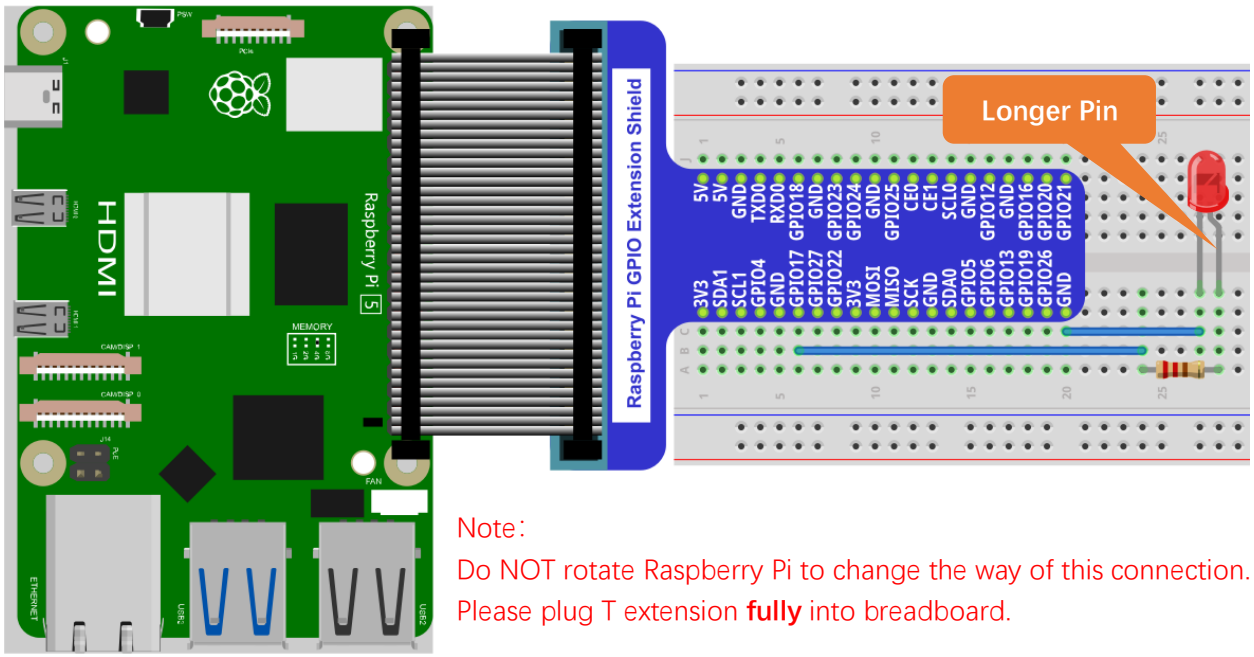
Circuit

First, disconnect your RPi from the GPIO Extension Shield. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield.
CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your RPi!

Schematic diagram



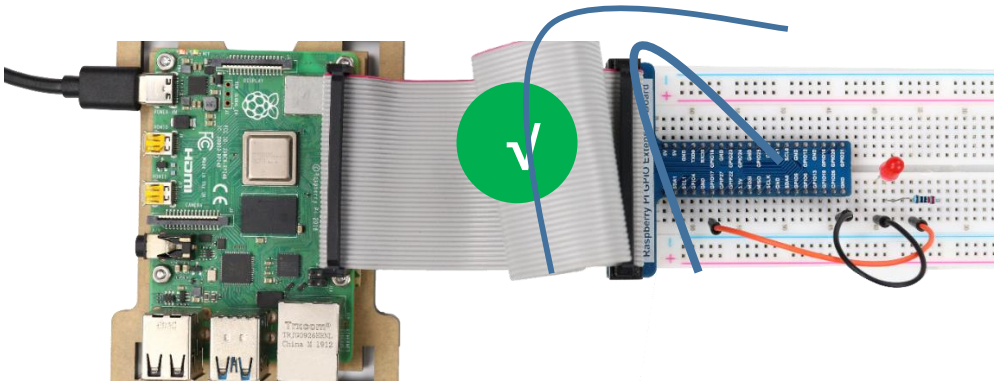
Hardware connection. **If you need any support, please contact us via:** support@freenove.com

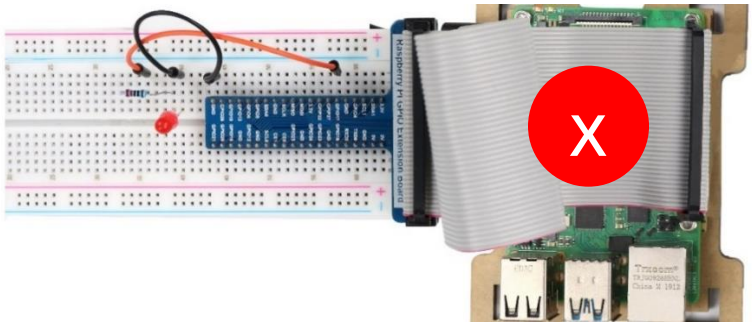


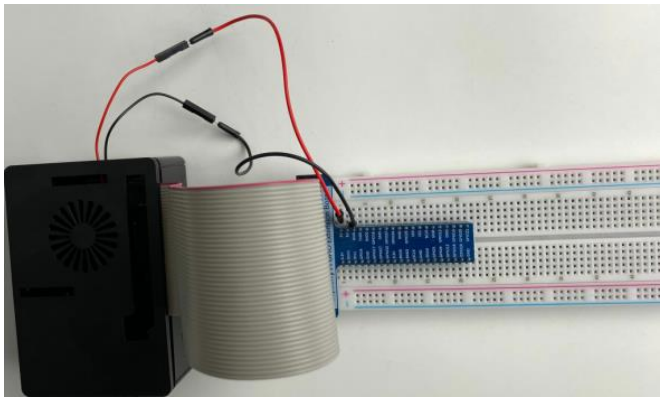
Note:
Do NOT rotate Raspberry Pi to change the way of this connection.
Please plug T extension **fully** into breadboard.

Youtube video https://youtu.be/hGQtnxsr1L4

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







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

How to distinguish resistors?

There are only three kind of resistors in this kit.

The one with 1 red ring is 10KΩ 

The one with 2 red rings is 220Ω 

The one with 0 red ring is 1KΩ 

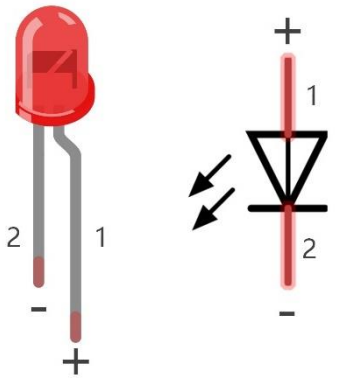
Future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

Component knowledge

LED

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

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



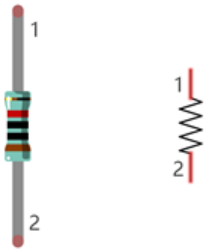
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

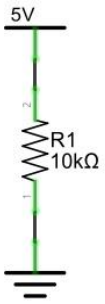
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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

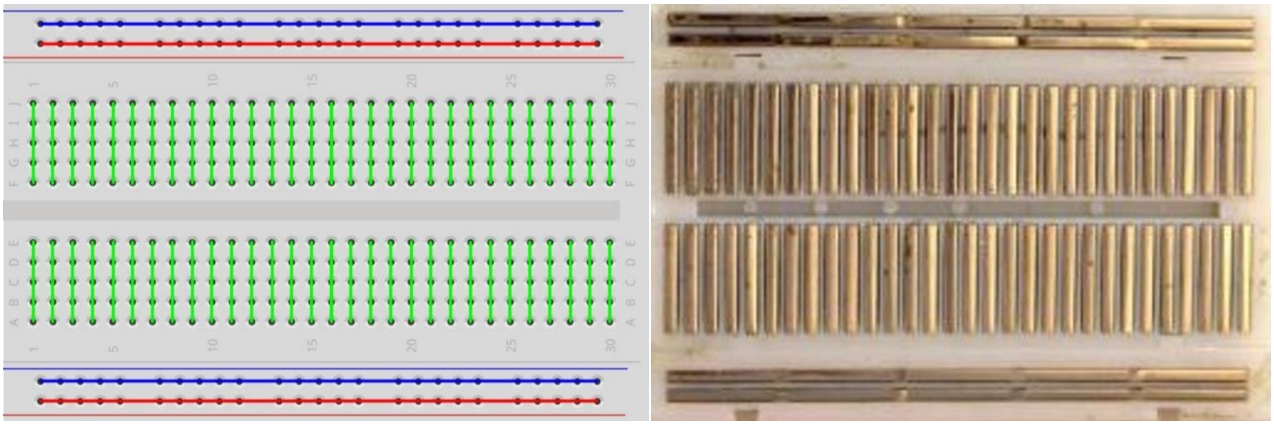


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

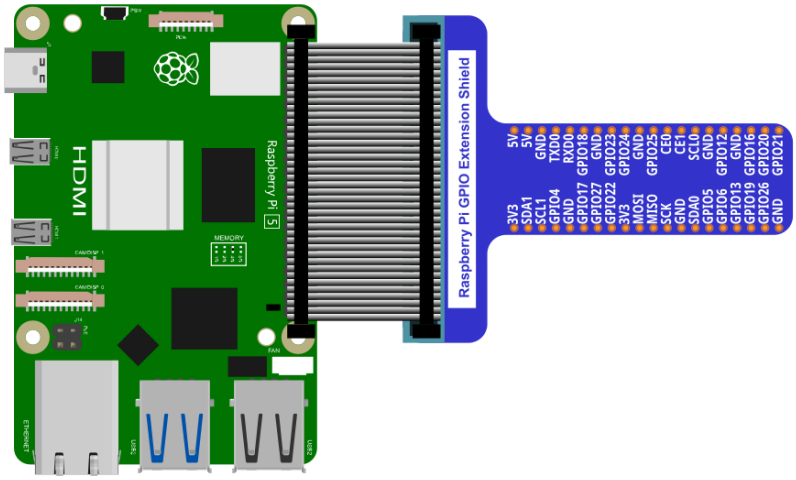
Breadboard

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



GPIO Extension Board

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



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 to achieve the target.

C Code 1.1.1 Blink

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

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

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

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

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

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

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

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

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

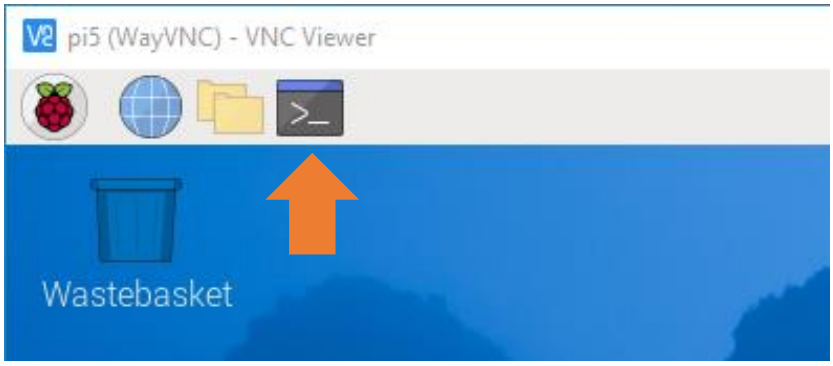
“l” of “lwiringPi” is low case of “L”.

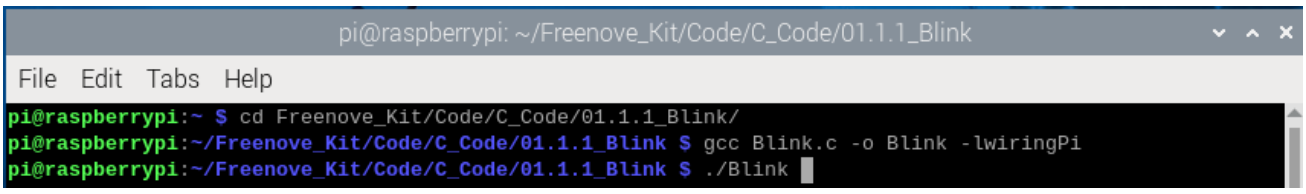
```
gcc Blink.c -o Blink -lwiringPi
```

4. Then run the generated file “blink”.

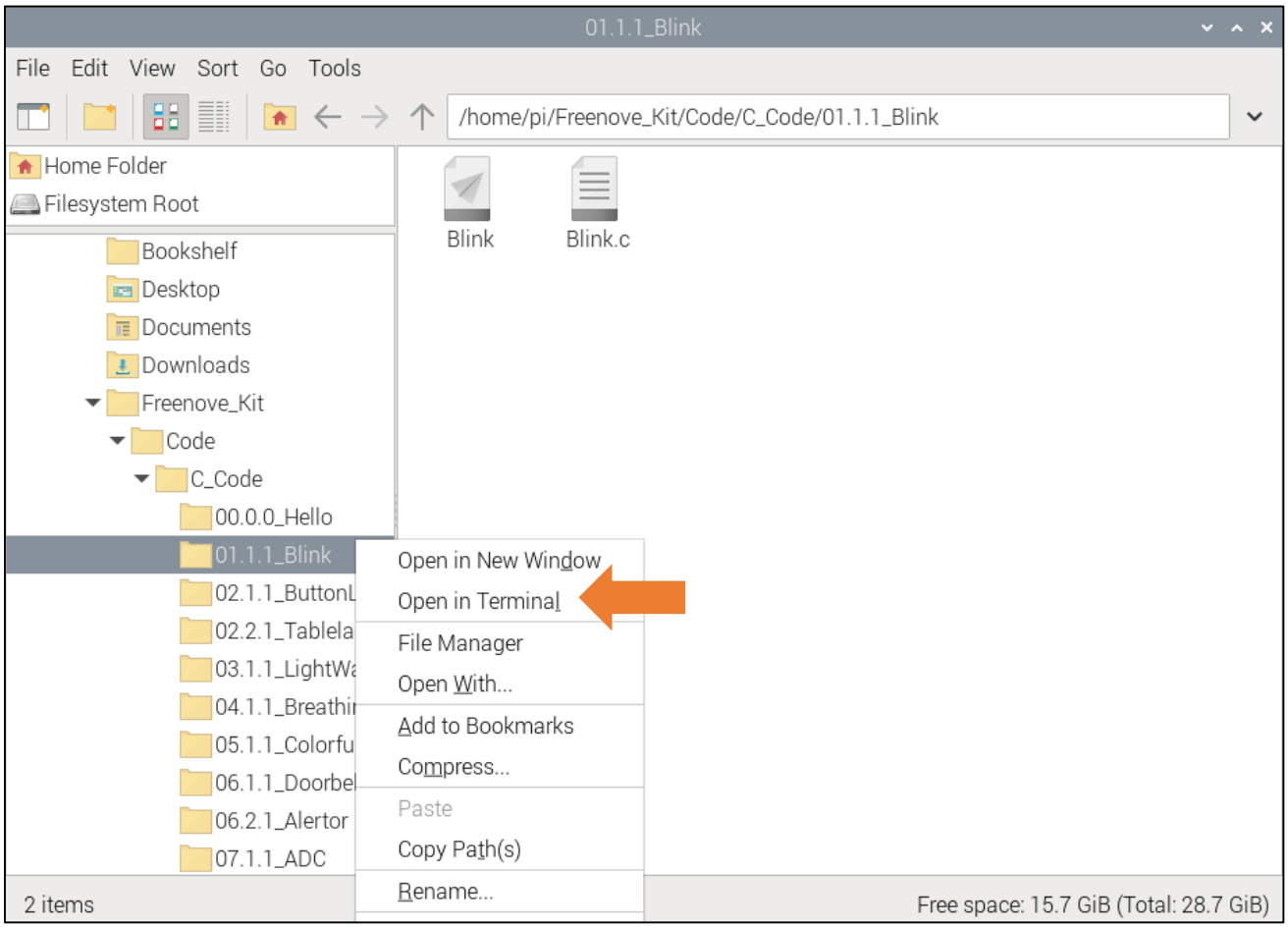
```
sudo ./Blink
```

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





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



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

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

#define  ledPin    0  //define the led pin number

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

    wiringPiSetup();  //Initialize wiringPi.

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

```
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 https://github.com/WiringPi/WiringPi

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	GPIO2/SDA1	3	4	5V	5V
9	GPIO3/SCL1	5	6	GND	GND
7	GPIO4	7	8	GPIO14/TXD0	15
GND	GND	9	10	GPIO15/RXD0	16
0	GPIO17	11	12	GPIO18	1
2	GPIO27	13	14	GND	GND
3	GPIO22	15	16	GPIO23	4
3.3V	3.3V	17	18	GPIO24	5
12	GPIO10/MOSI)	19	20	GND	GND
13	GPIO9/MOIS	21	22	GPIO25	6
14	GPIO11/SCLK	23	24	GPIO8/CE0	10
GND	GND	25	26	GPIO7/CE1	11
30	GPIO0/SDA0	27	28	GPIO1/SCL0	31
21	GPIO5	29	30	GND	GND
22	GPIO6	31	32	GPIO12	26
23	GPIO13	33	34	GND	GND
24	GPIO19	35	36	GPIO16	27
25	GPIO26	37	38	GPIO20	28
GND	GND	39	40	GPIO21	29

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

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

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

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

Other Code Editors (Optional)

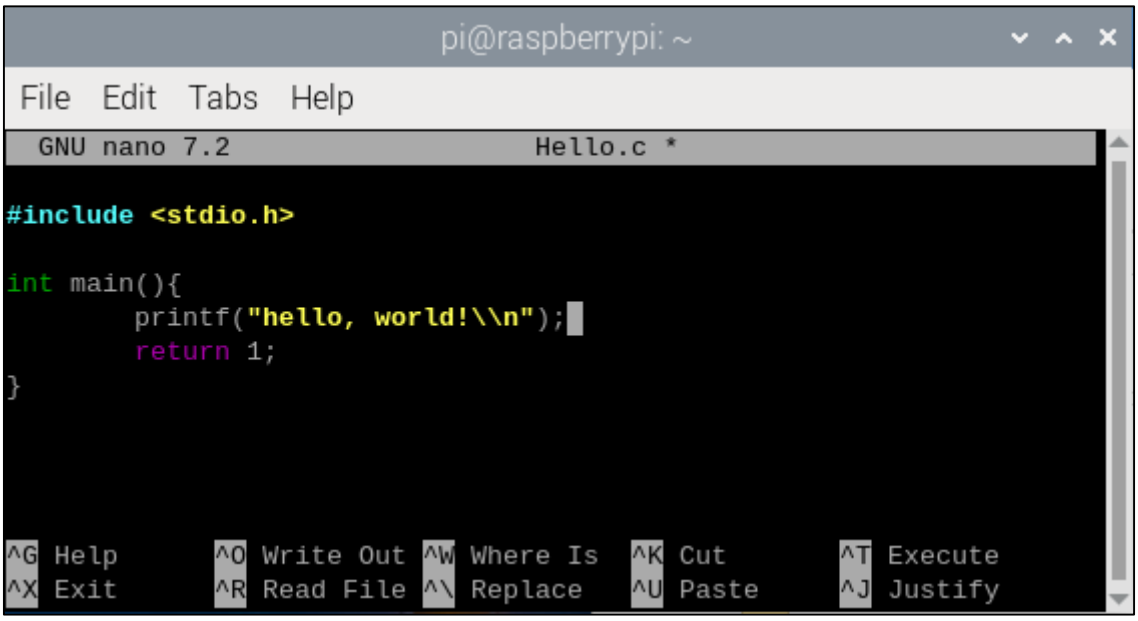
If you want to use other editor to edit and execute the code, you can learn them in this section.

nano

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

```
nano Hello.c
```

As is shown below:



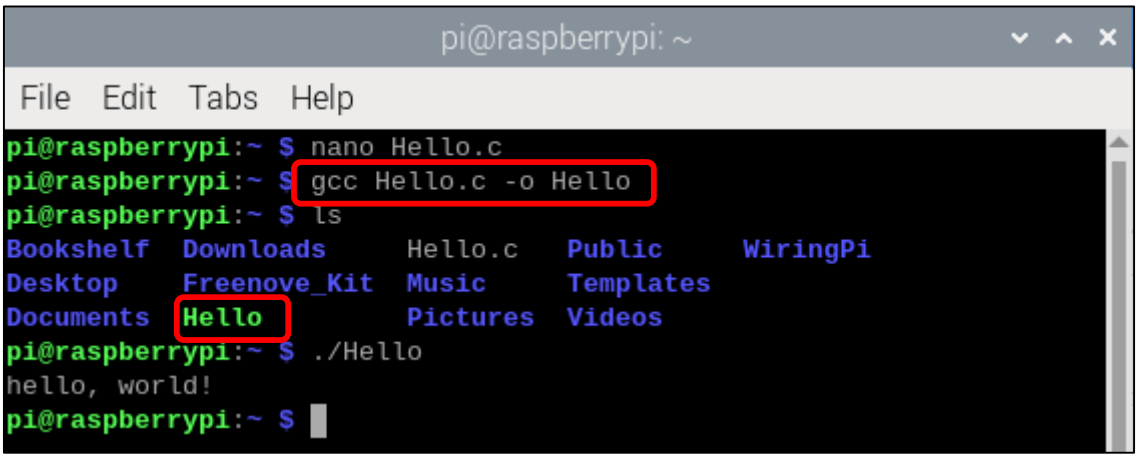
Use the following command to compile the code to generate the executable file “Hello”.

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

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

```
sudo ./Hello
```

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

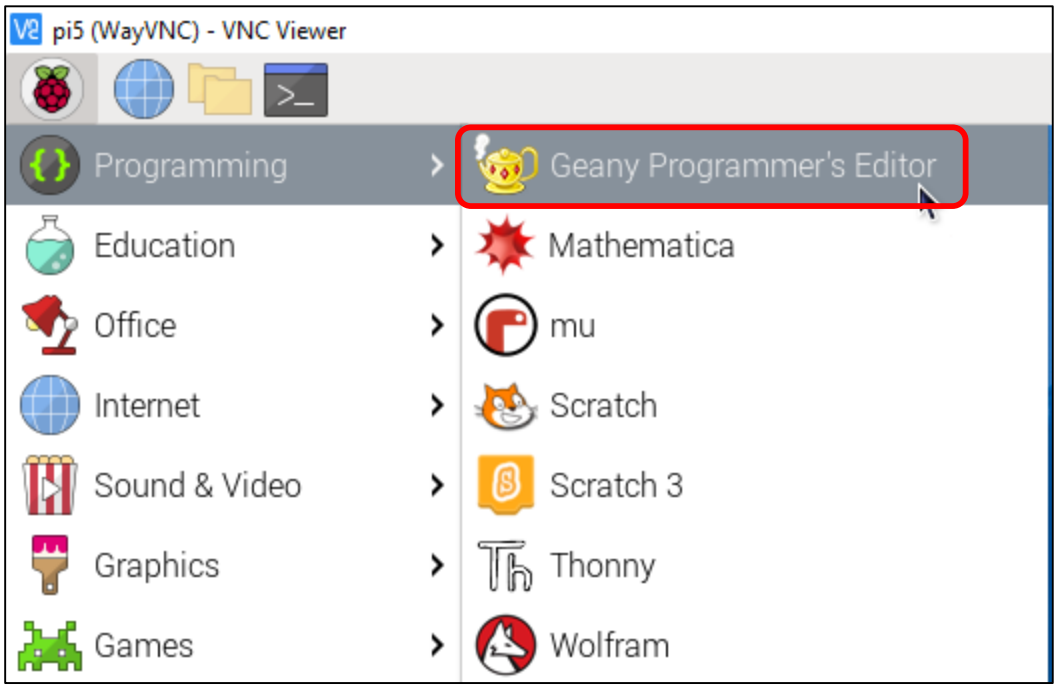


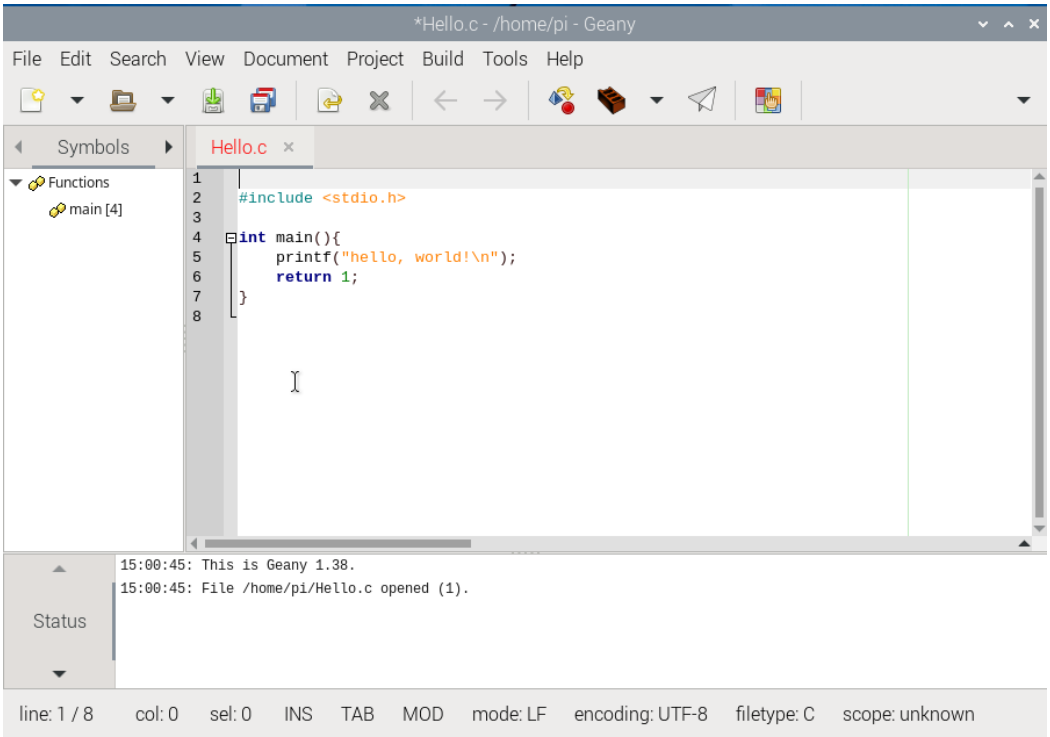
geany

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
```

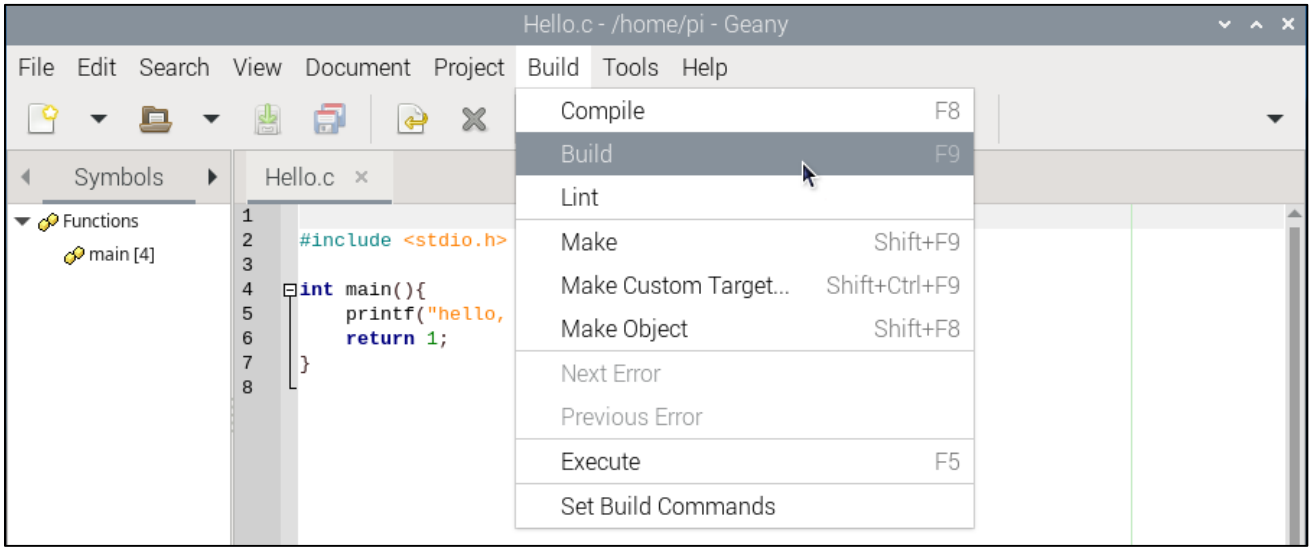
Or find and open Geany directly in the desktop main menu, and then click **File→Open** to open the "Hello.c", Or drag "Hello.c" to Geany directly.



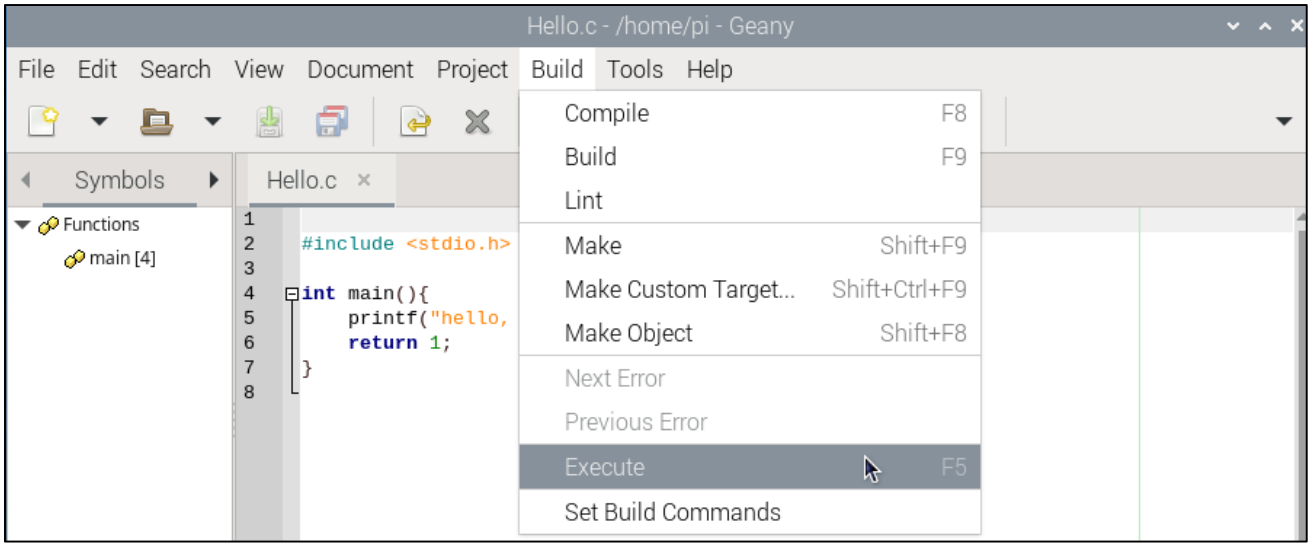


If you want to create a new code, click File→New→File→Save as (name.c or name.py). Then write the code.

Generate an executable file by clicking menu bar Build->Build.

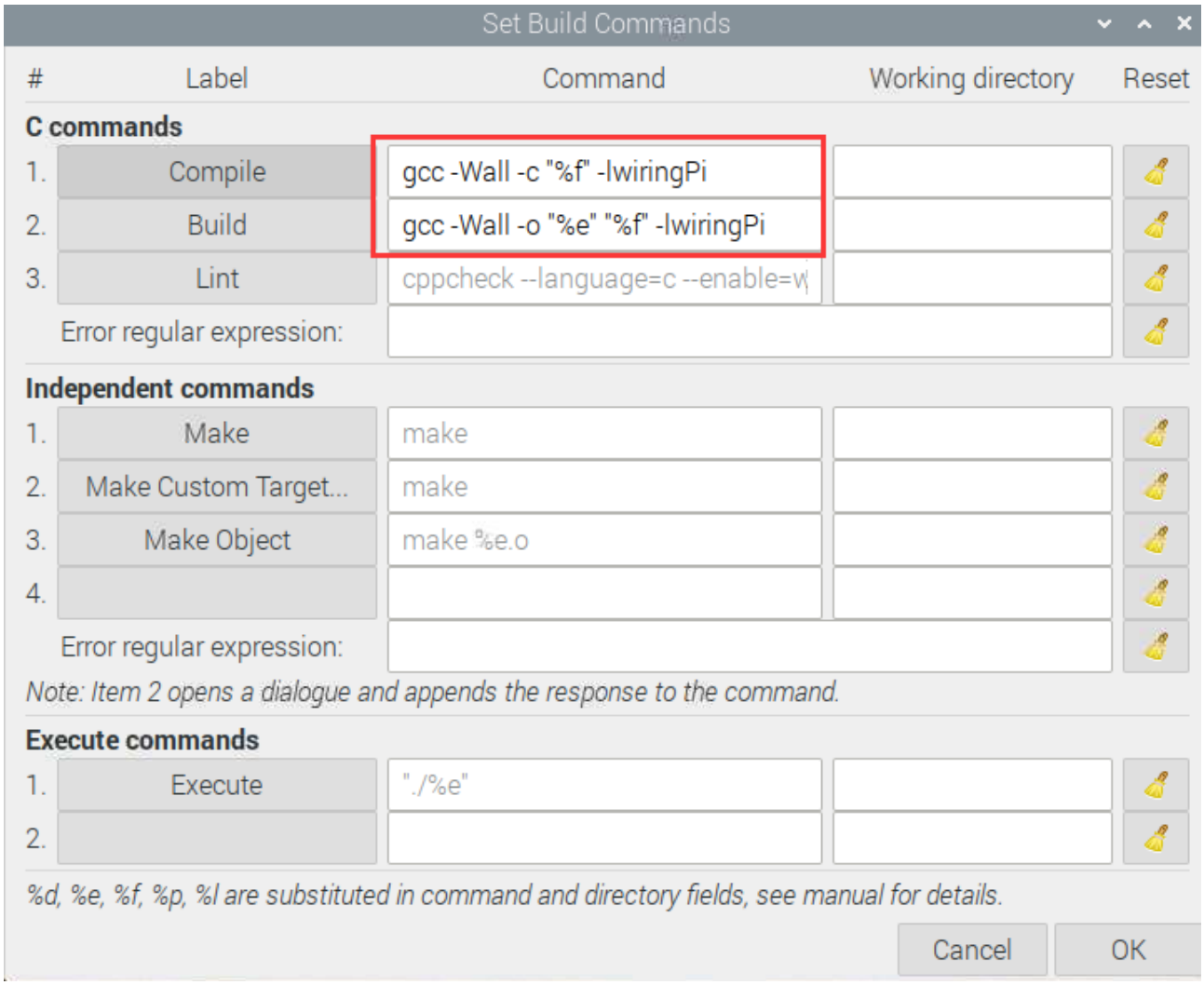


Then execute the generated file by clicking menu bar Build->Execute.



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

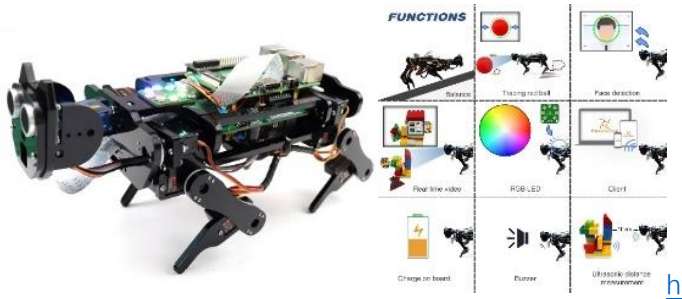
We also have car and robot kits for Raspberry Pi. You can visit our website for details.

https://www.amazon.com/freenove

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


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

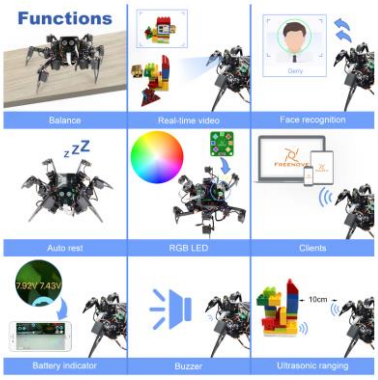
FNK0050 **Freenove Robot Dog Kit for Raspberry Pi**



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

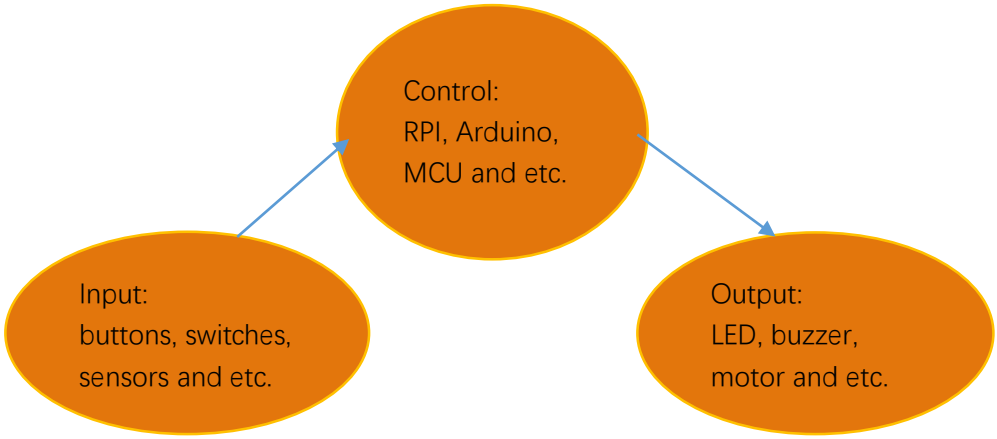
FNK0052 **Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi**

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

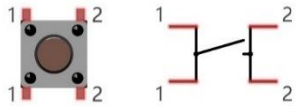
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push Button Switch x1 
Jumper Wire 				

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

Component knowledge

Push Button Switch

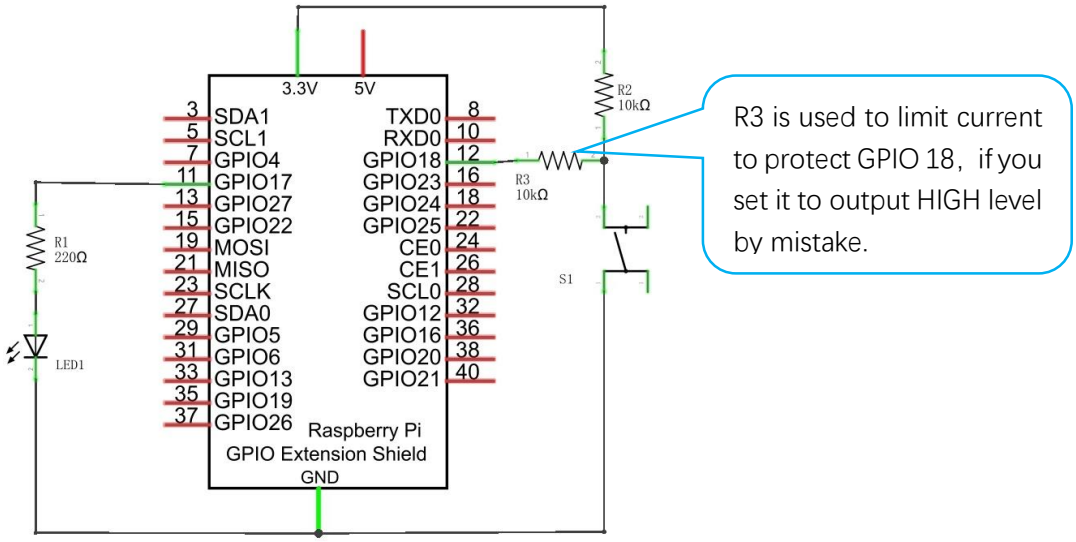
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



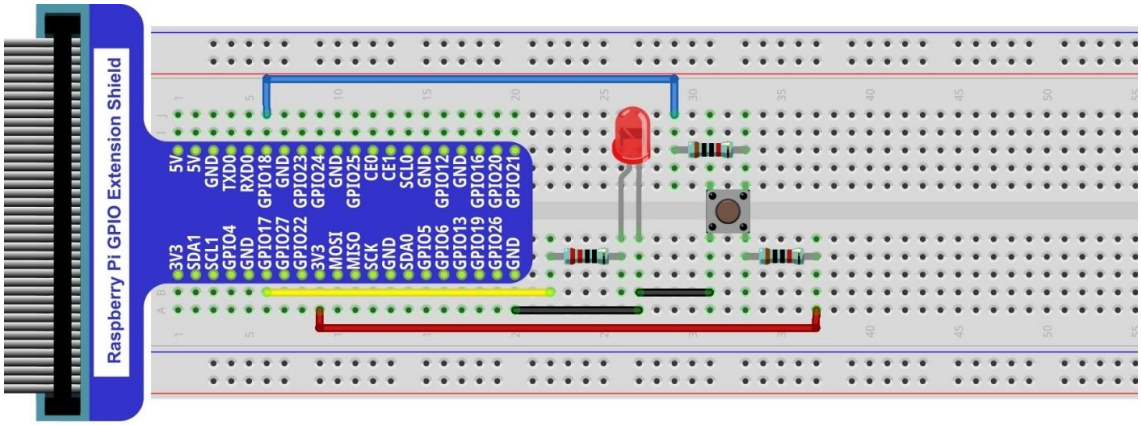
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



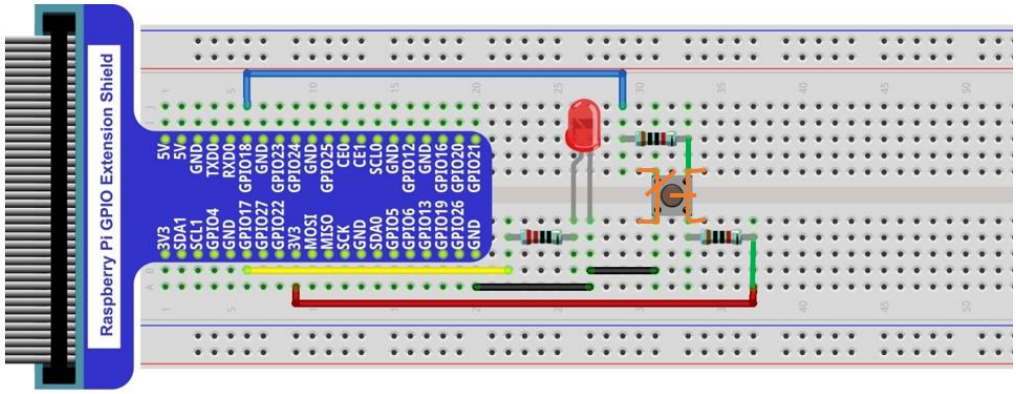
There are two kinds of push button switch in this kit.

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

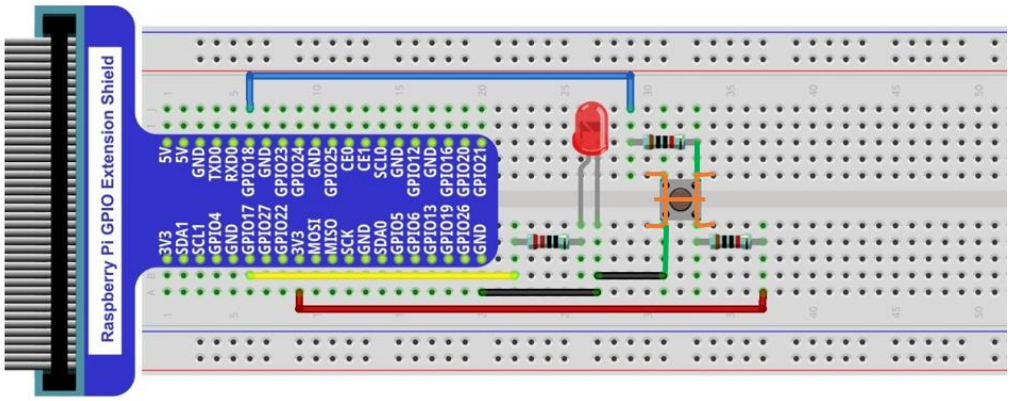
Youtube video: https://youtu.be/_5ge1d6f1nM

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>
```

```
#include <stdio.h>

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

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

    wiringPiSetup();  //Initialize wiringPi.

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

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){
        if(digitalRead(buttonPin) == LOW){ //button is pressed
            digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
            printf("Button is pressed, led turned on >>>\n");        //Output information on terminal
        }
        else {                              //button is released
            digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
            printf("Button is released, led turned off <<<\n");      //Output information on terminal
        }
    }
}
```

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:

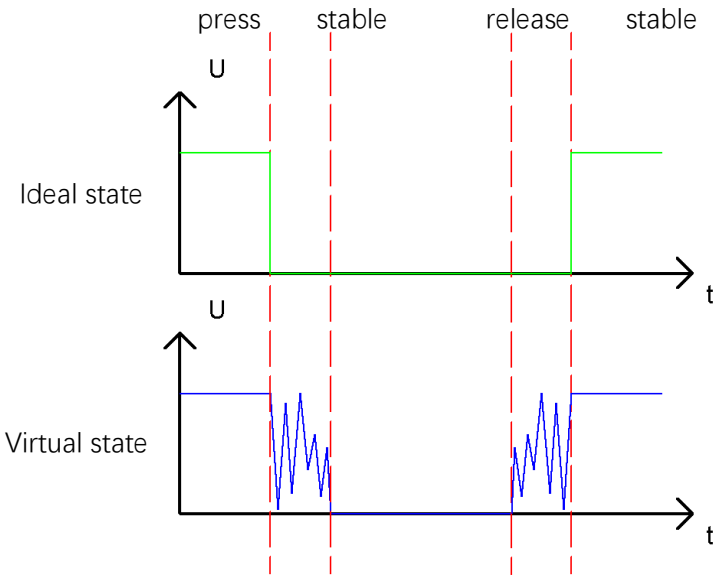
`int digitalRead (int pin);`
This function returns the value read at the given pin. It will be "**HIGH**" or "**LOW**"(1 or 0) depending on the logic level at the pin.

Project 2.2 MINI Table Lamp

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

Debounce a Push Button Switch

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



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

Code

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

C Code 2.2.1 Tablelamp

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

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

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

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

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

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

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

```
sudo ./Tablelamp
```

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

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

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

    wiringPiSetup(); //Initialize wiringPi.

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

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

```
27          //if changing-state of the button last beyond the time we set, we consider 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, it means the action is pressing
34                  if(buttonState == LOW){
35                      printf("Button is pressed!\n");
36                      ledState = !ledState; //Reverse the LED state
37                      if(ledState){
38                          printf("turn on LED ...\n");
39                      }
40                      else {
41                          printf("turn off LED ...\n");
42                      }
43                  }
44                  //if the state is high, it means the action is releasing
45                  else {
46                      printf("Button is released!\n");
47                  }
48              }
49          }
50          digitalWrite(ledPin,ledState);
51          lastbuttonState = reading;
52      }
53
54      return 0;
55  }
```

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");
                }
```

Chapter 3 LED Bar Graph

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

Project 3.1 Flowing Water Light

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

Component List

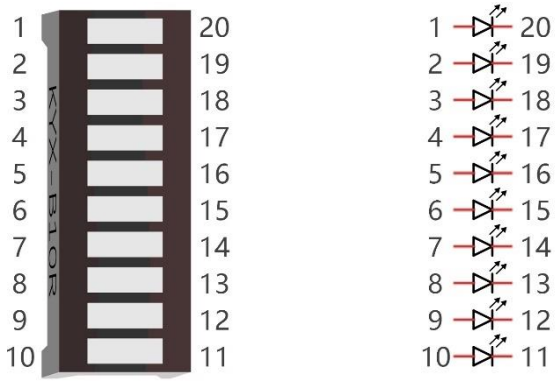
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Bar Graph LED x1 	Resistor 220Ω x10 
Jumper Wire x 1 		

Component knowledge

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

Bar Graph LED

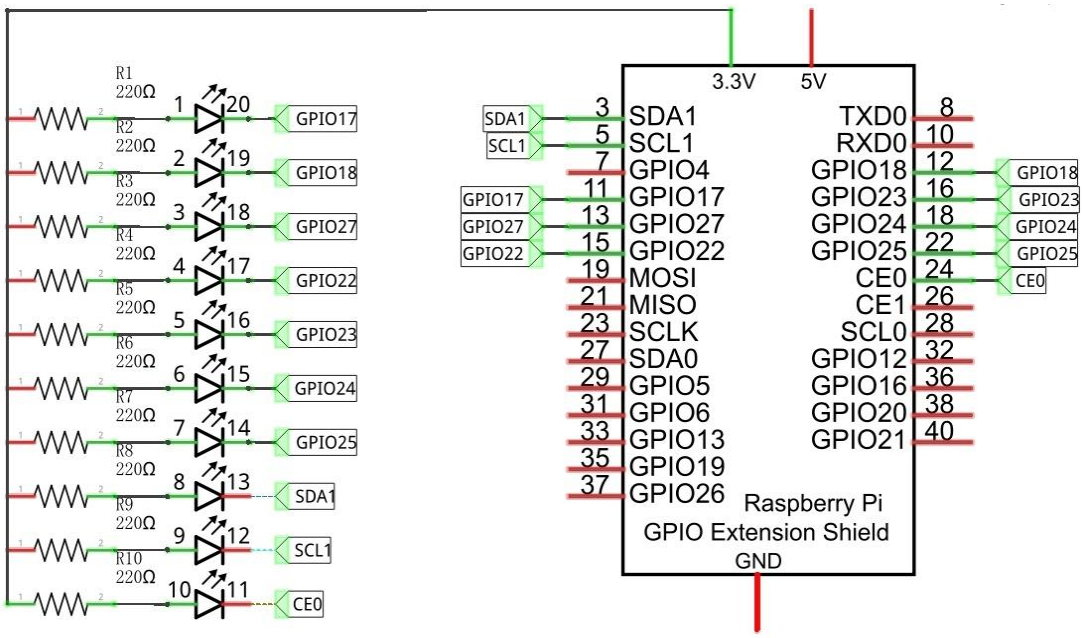
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



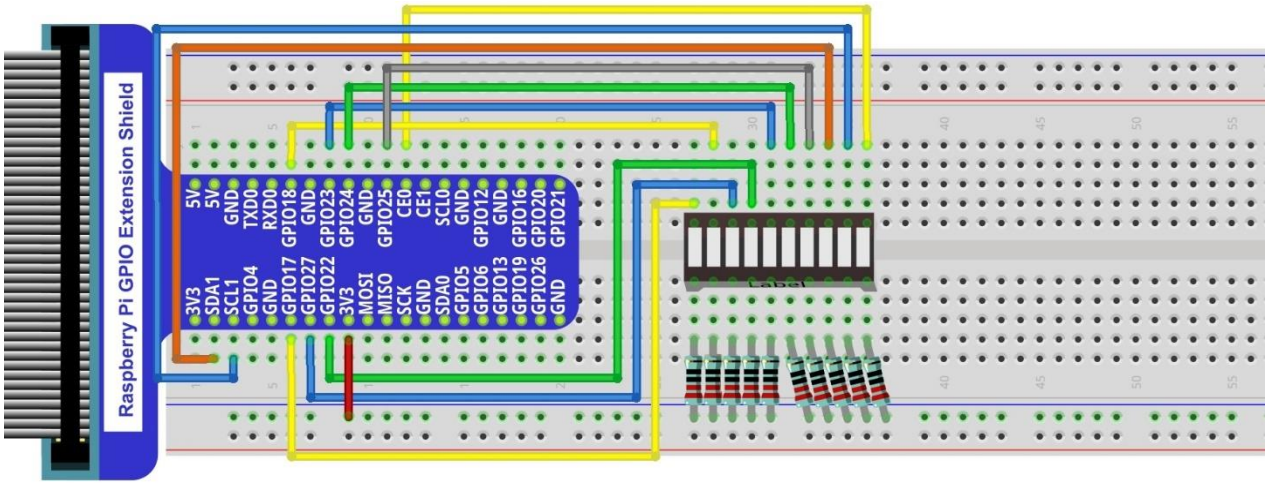
Circuit

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

Schematic diagram



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



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

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:

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

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

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

    wiringPiSetup(); //Initialize wiringPi.

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

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

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

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.

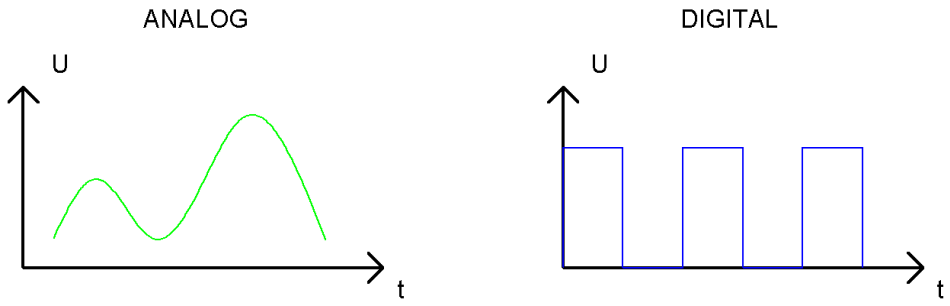
Component List

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

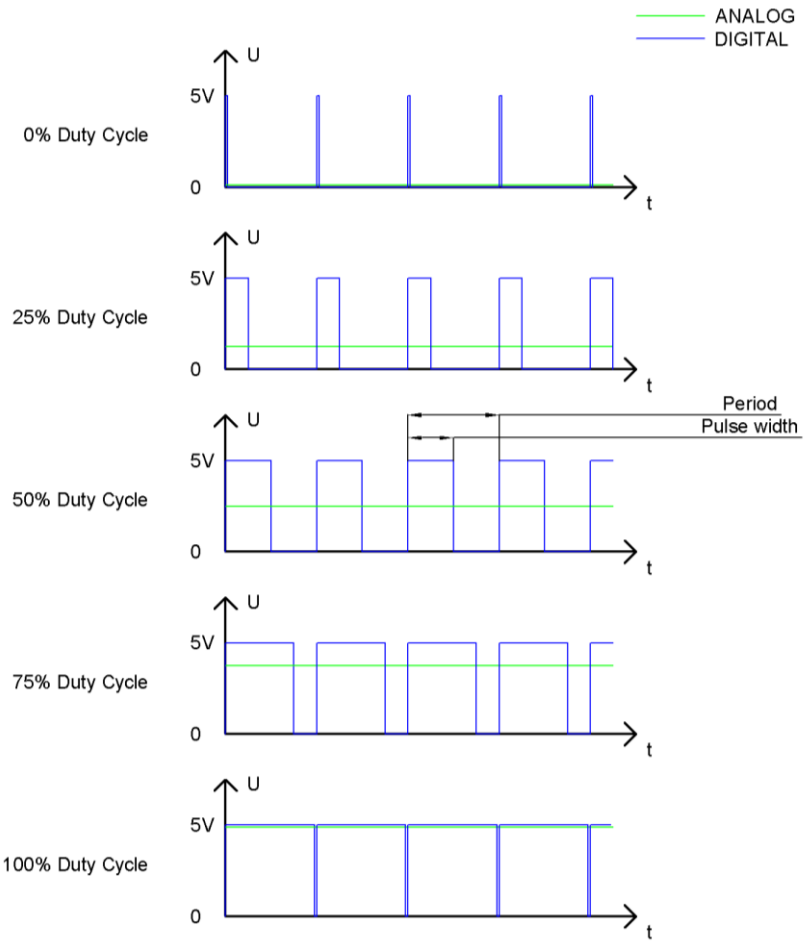
An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal **or discrete-time signal is a time series consisting of a sequence of quantities**. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout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



Note that the Analog signals are curved waves and the Digital signals are “Square Waves”.
In practical applications, we often use binary as the digital signal, that is a series of 0’s and 1’s. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Digital processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals).
PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called “pulse width”, and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform. The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



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

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

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

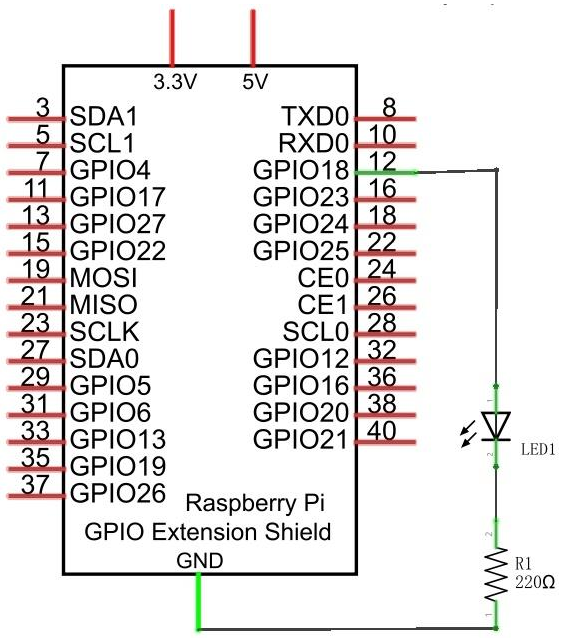
The wiringPi library of C provides both a hardware PWM and a software PWM method.

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.

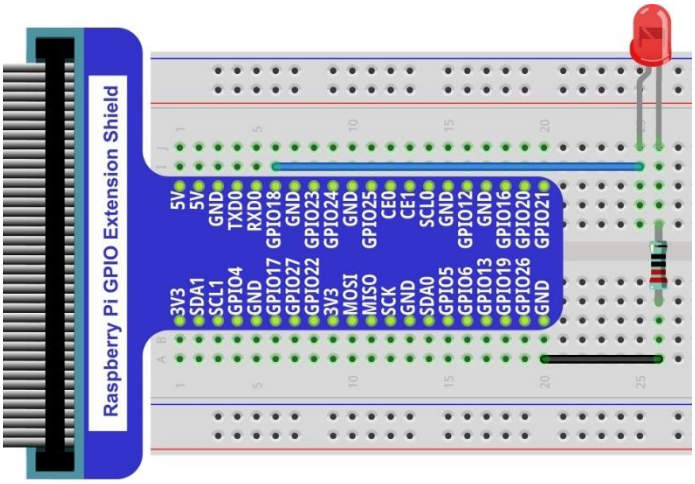
In order to keep the results running consistently, we will use PWM.

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);//Creat SoftPWM pin
14
15      while(1){
16          for(i=0;i<100;i++){  //make the led brighter
17              softPwmWrite(ledPin, i);
18              delay(20);
19          }
20          delay(300);
21          for(i=100;i>=0;i--){  //make the led darker
22              softPwmWrite(ledPin, i);
23              delay(20);
24          }
25          delay(300);
26      }
27  }
```

First, create a software PWM pin.

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

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

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

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

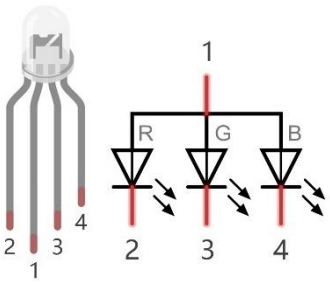
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
This creates a software controlled PWM pin.
void softPwmWrite (int pin, int value) ;
This updates the PWM value on the given pin.

For more details, please refer https://github.com/WiringPi/WiringPi

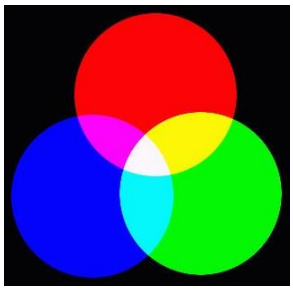
Chapter 5 RGB LED

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

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



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



RGB

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

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

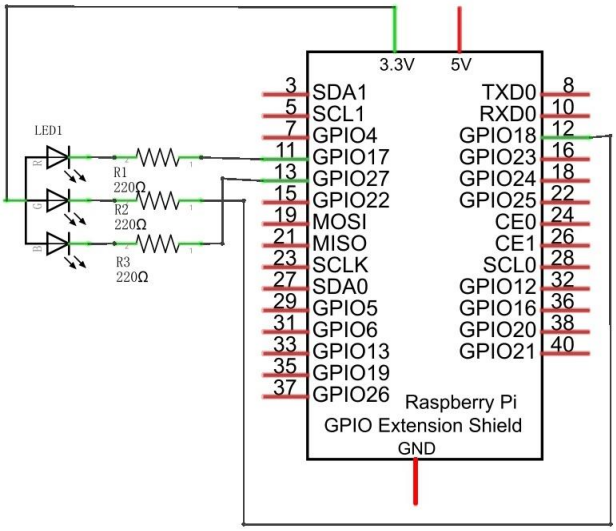
Project 5.1 Multicolored LED

Component List

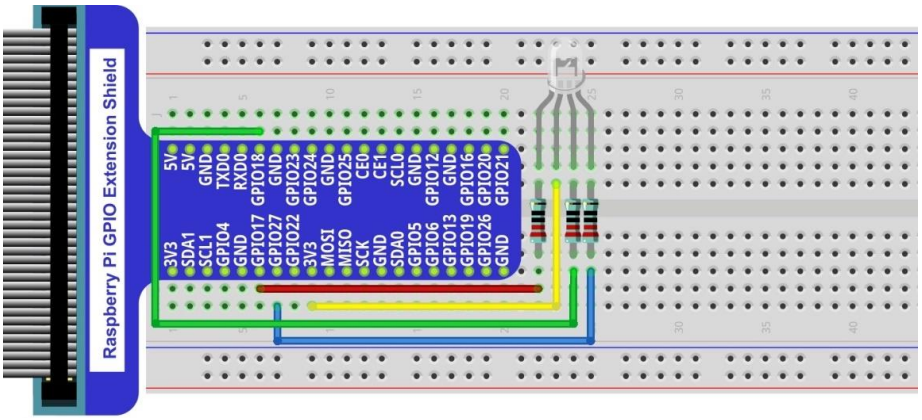
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	RGB LED x1 	Resistor 220Ω x3 
Jumper Wire 		

Circuit

Schematic diagram



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



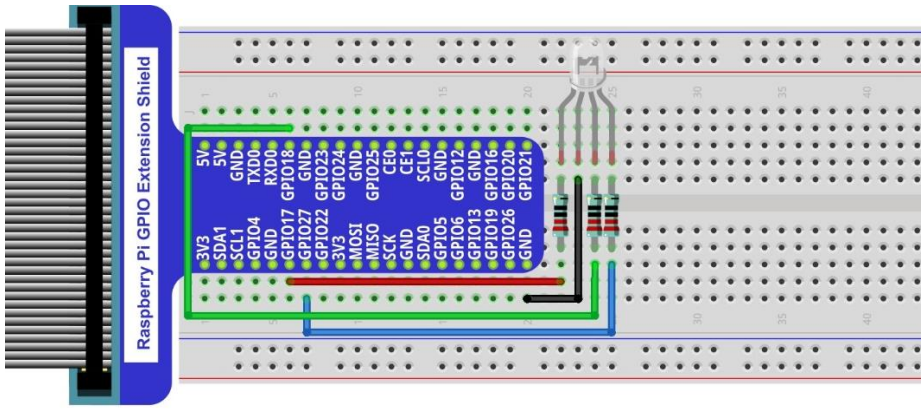
Video: https://youtu.be/tbnX2AsX2y4

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_ColorfulLED directory of C code.

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

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

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

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

```
sudo ./ColorfulLED
```

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

The following is the program code:

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

Chapter 6 Buzzer

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

Project 6.1 Doorbell

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

Component List

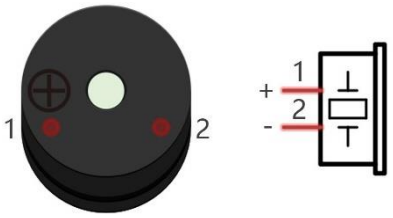
<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push Button Switch x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ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

Component knowledge

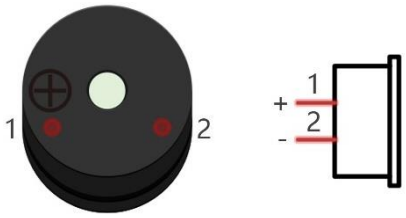
Buzzer

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

Active buzzer



Passive buzzer



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

How to identify active and passive buzzer?

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

Active buzzer bottom

Passive buzzer bottom

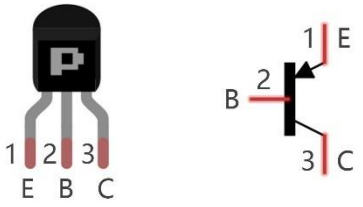
Transistors

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

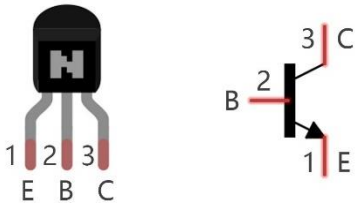
amplify the current.

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

PNP transistor



NPN transistor



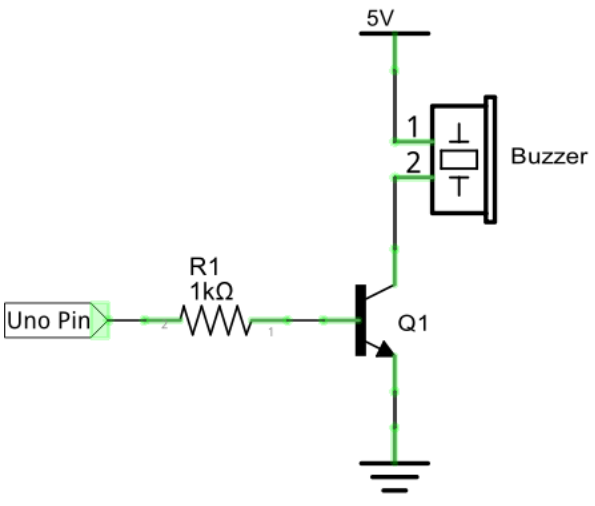
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

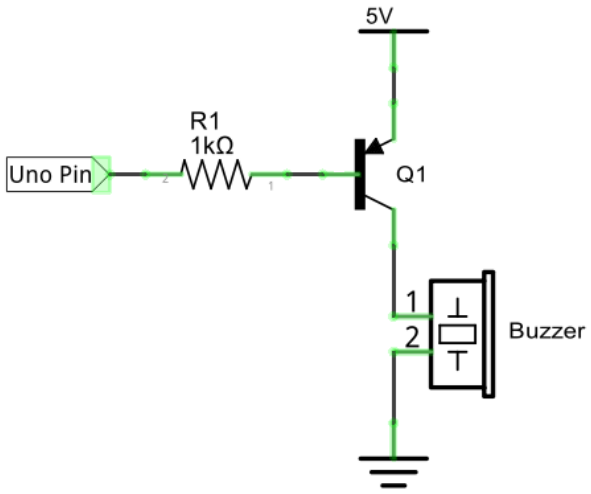
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not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

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

NPN transistor to drive buzzer

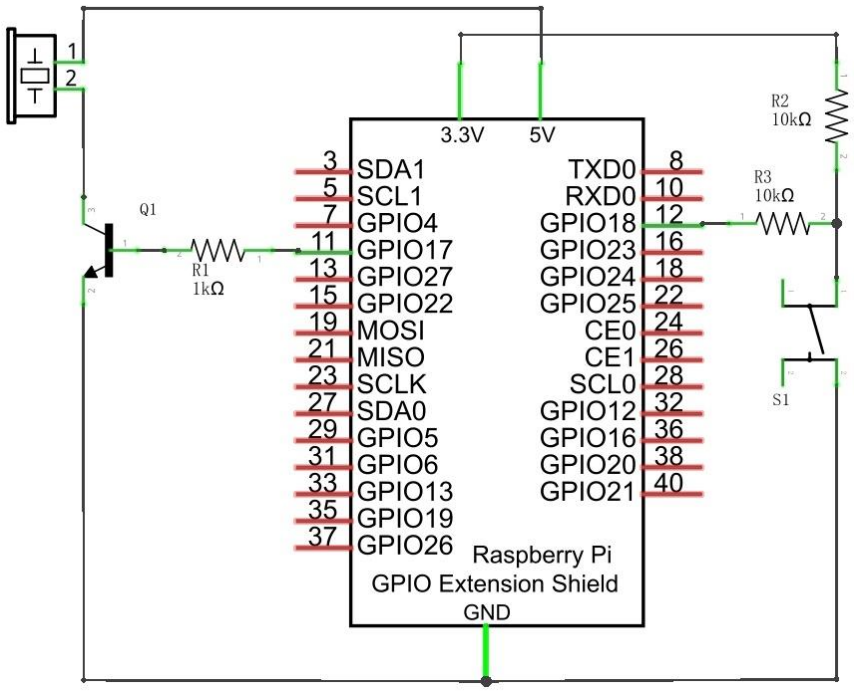


PNP transistor to drive buzzer

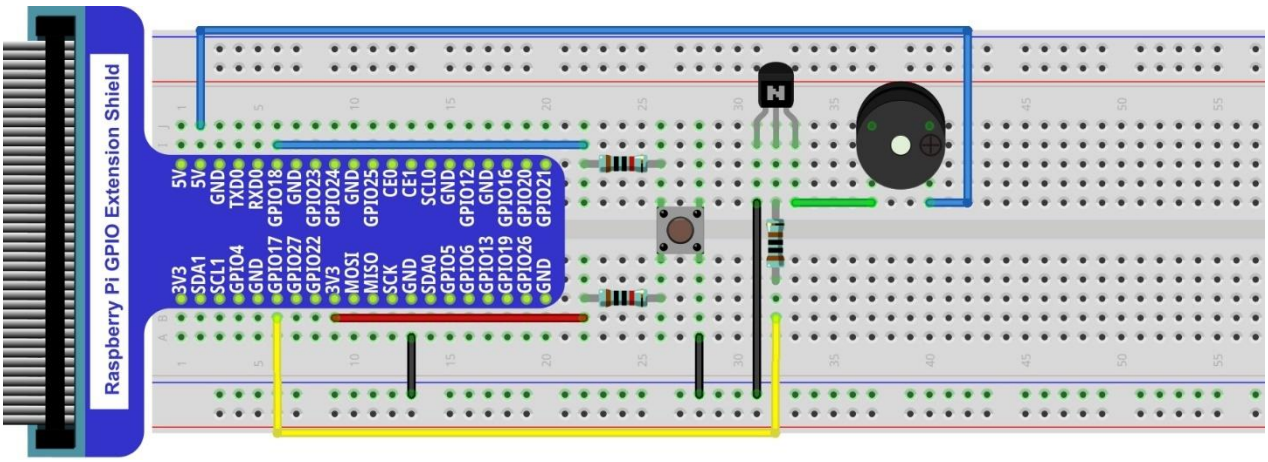


Circuit

Schematic diagram with RPi GPIO Extension Shield



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



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.c".

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

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

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

The following is the program code:

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

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

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

    wiringPiSetup();

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

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

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

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

Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.
The list of components and the circuit is similar to the doorbell project. We only need to take the Doorbell circuit and replace the active buzzer with a passive buzzer.

Code

In this project, our buzzer alarm is controlled by the push button switch. Press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.
As stated before, it is analogous to our earlier project that controlled an LED ON and OFF.
To control a passive buzzer requires PWM of certain sound frequency.

C Code 6.2.1 Alertor

First, observe the project result, and then learn about the code in detail.
If you have any concerns, please contact us via: support@freenove.com

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

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

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

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

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

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

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

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

void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ // frequency of the alertor is consistent with the sine wave
        sinVal = sin(x * (M_PI / 180));     //Calculate the sine value
        toneVal = 2000 + sinVal * 500; //Add the resonant frequency and weighted sine value
        softToneWrite(pin,toneVal);             //output corresponding PWM
        delay(1);
```

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

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

```
    softToneCreate(buzzeRPin);
```

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

```
    void alertor(int pin){
        int x;
        double sinVal, toneVal;
        for(x=0;x<360;x++){ //The frequency is based on the sine curve.
            sinVal = sin(x * (M_PI / 180));
            toneVal = 2000 + sinVal * 500;
```

```
            softToneWrite(pin,toneVal);
            delay(1);
        }
    }
```

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

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

The related functions of softTone are described as follows:

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

For more details about softTone, please refer to : https://github.com/WiringPi/WiringPi

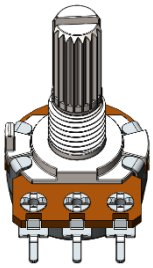
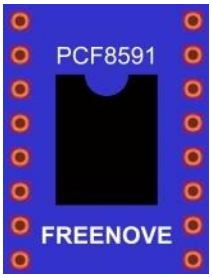
(Important) Chapter 7 ADC

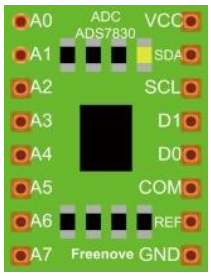
We have learned how to control the brightness of an LED through PWM and that PWM is not a real analog signal. In this chapter, we will learn how to read analog values via an ADC Module and convert these analog values into digital.

Project 7.1 Read the Voltage of Potentiometer

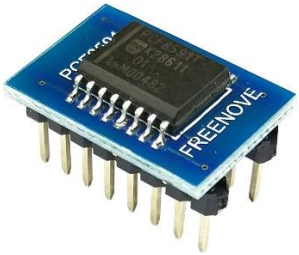
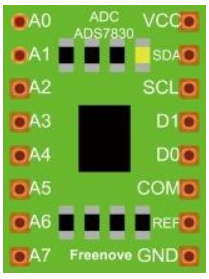
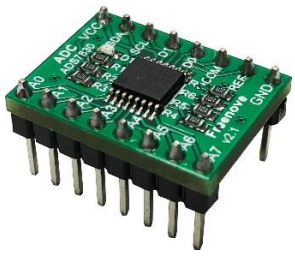
In this project, we will use the ADC function of an ADC Module to read the voltage value of a potentiometer.

Component List

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

or

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

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

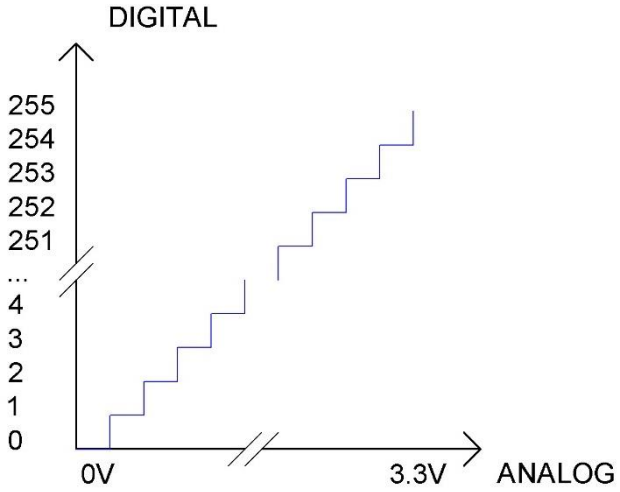
<table>
<tr><th colspan="2">ADC module: PCF8591</th><th colspan="2">ADC module: ADS7830</th></tr>
<tr><td>Model diagram
 
</td><td>Actual Picture
 
</td><td>Model diagram
 
</td><td>Actual Picture
  </td></tr>
</table>

Circuit knowledge

ADC

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

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



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

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

...

The resultant analog signal will be divided accordingly.

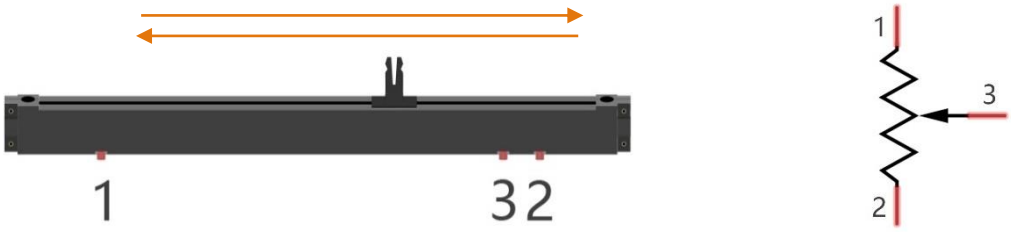
DAC

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

Component knowledge

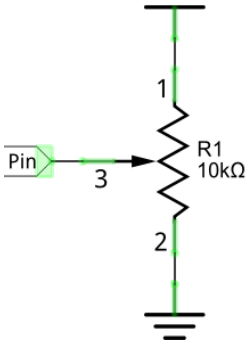
Potentiometer

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



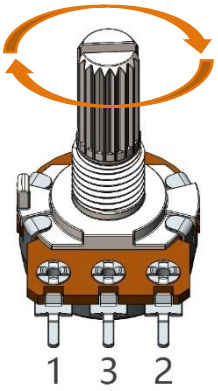
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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

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



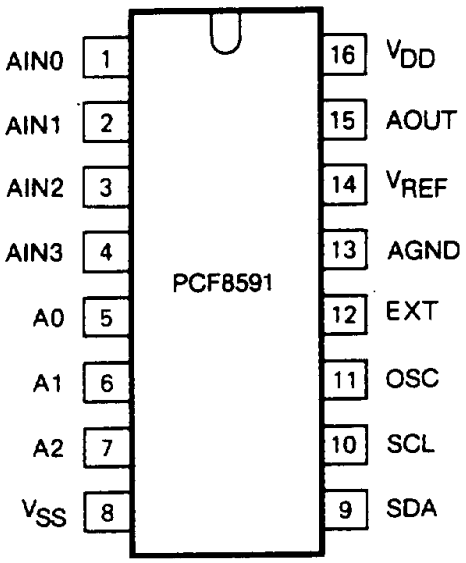
Rotary potentiometer

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



PCF8591

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

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

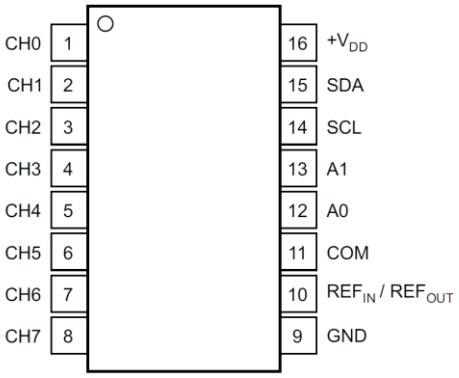
For more details about PCF8591, please refer to the datasheet which can be found on the Internet.

ADS7830

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

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

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

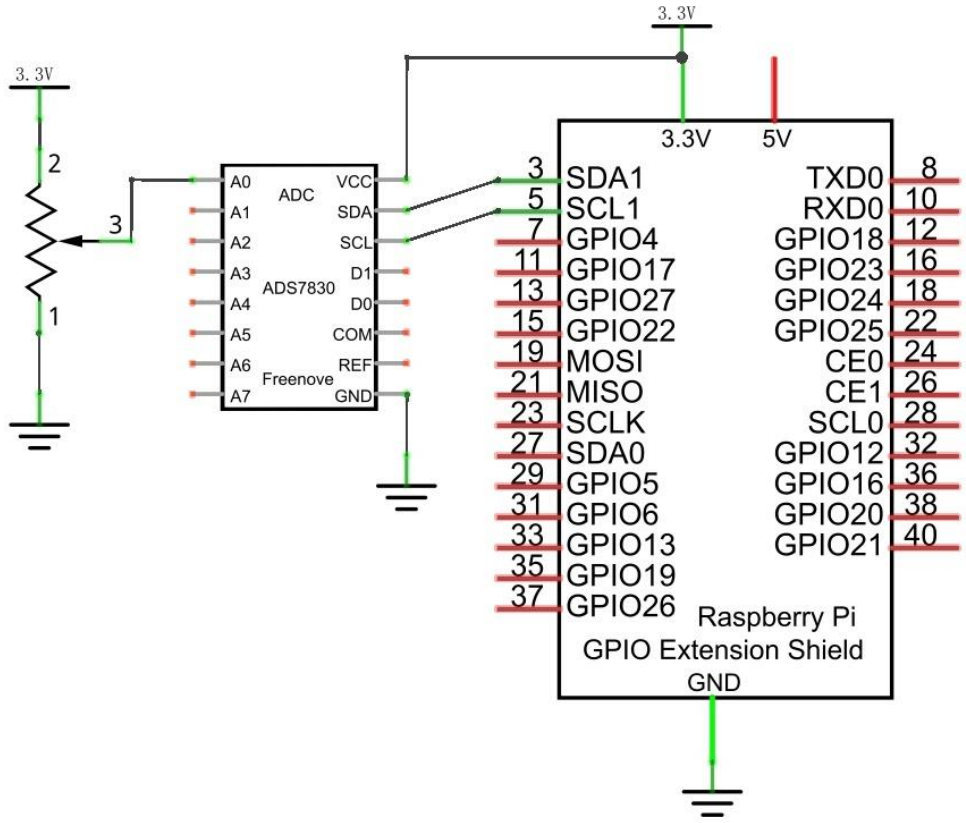


I2C communication

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

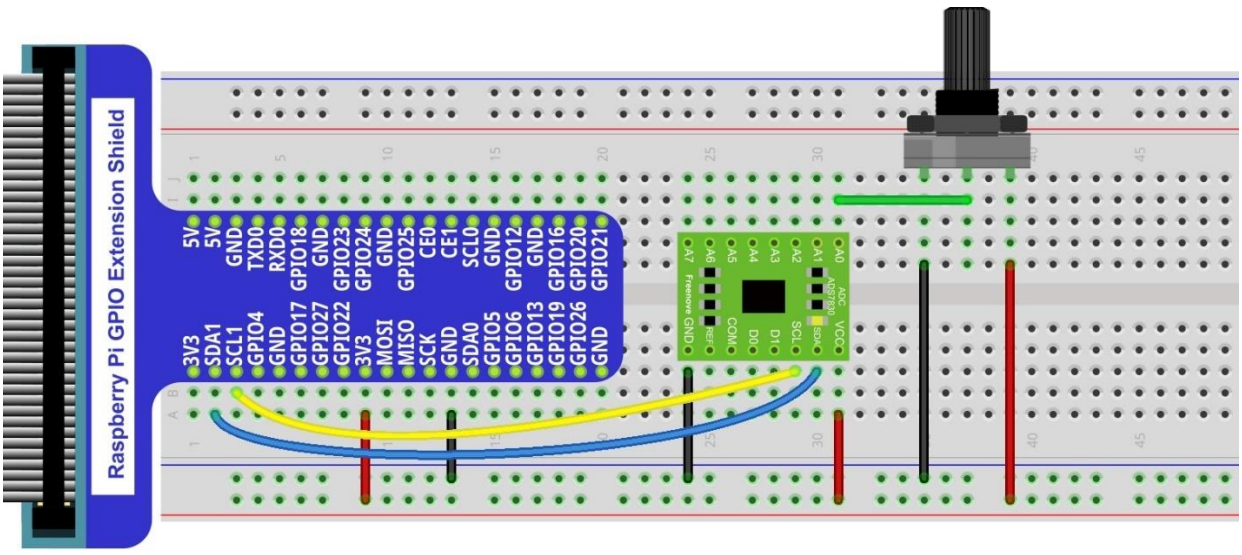
Circuit with ADS7830

Schematic diagram



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

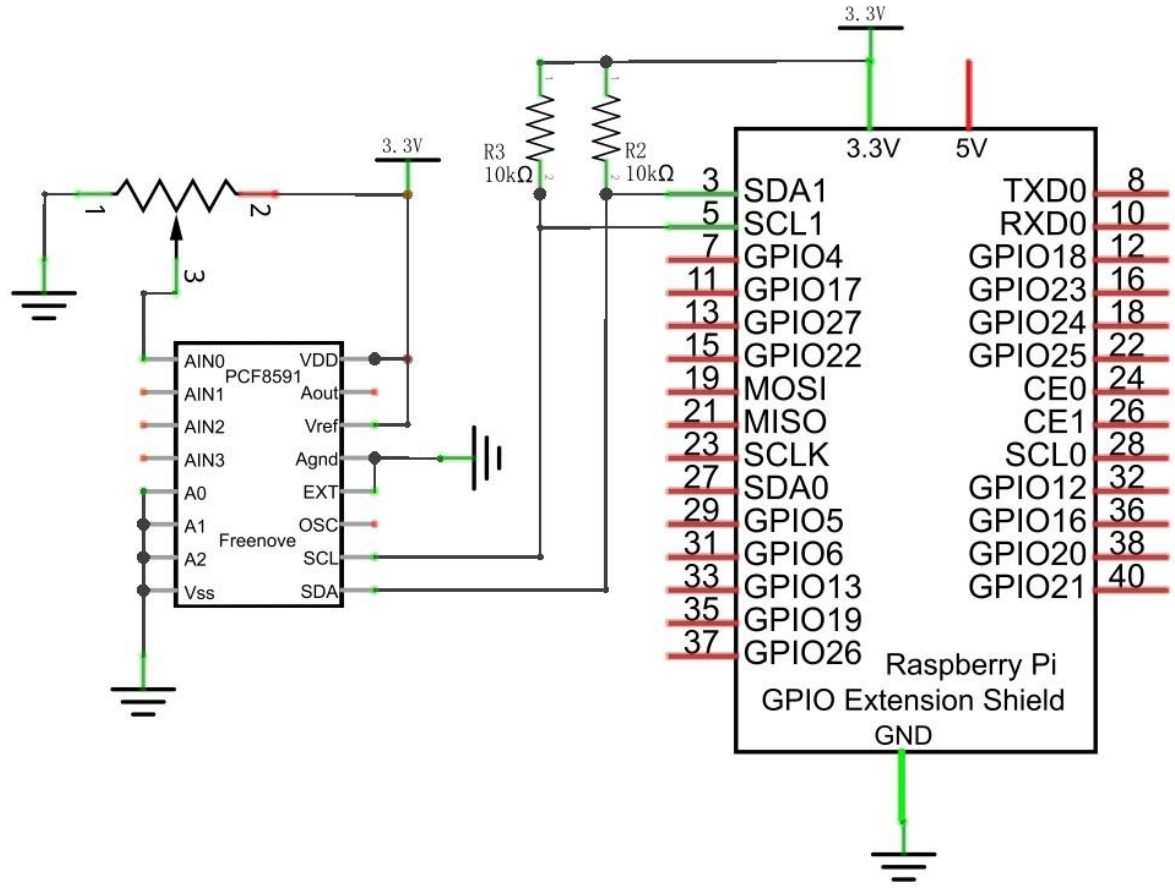
This product contains **only one ADC module.**



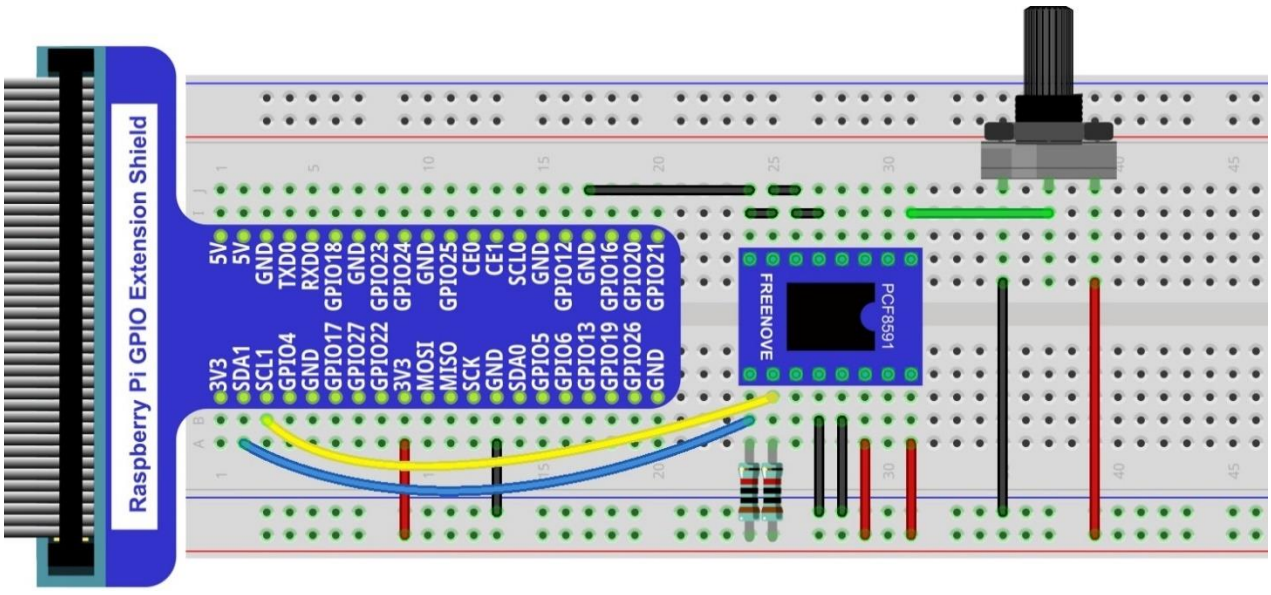
Video: https://youtu.be/PSUCctu_DqA

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

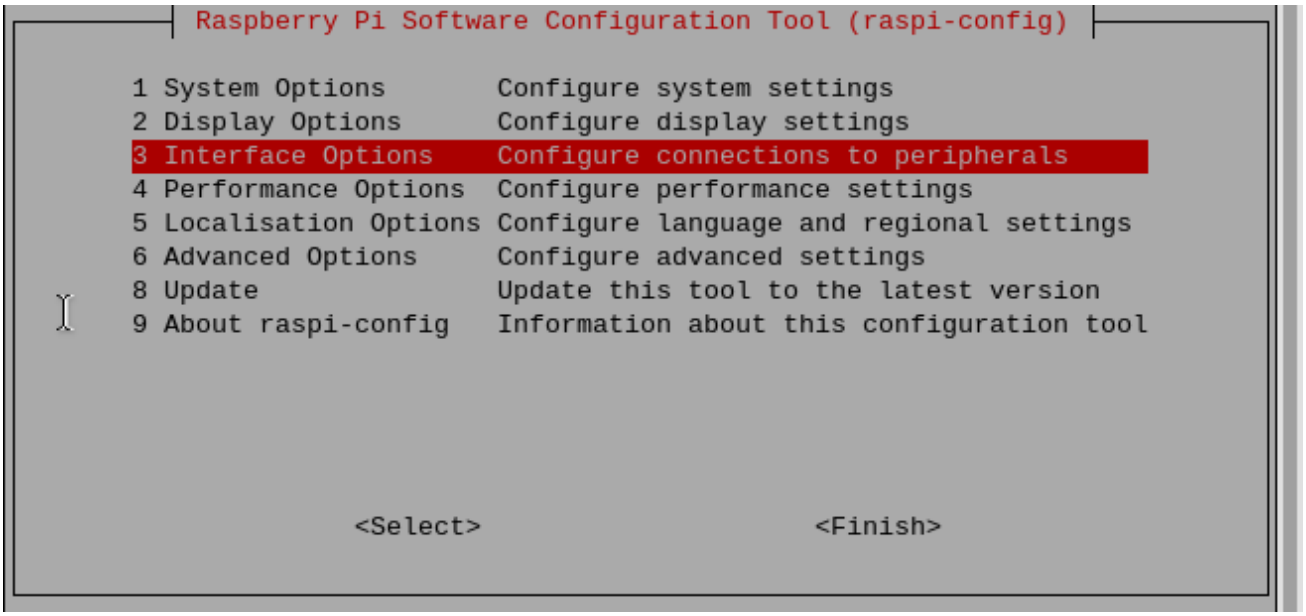
Enable I2C

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 “3 Interfacing Options” then “I4 I2C” then “Yes” and then “Finish” in this order and restart your RPi. The I2C module will then be started.

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

```
lsmod | grep i2c
```

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

Install I2C-Tools

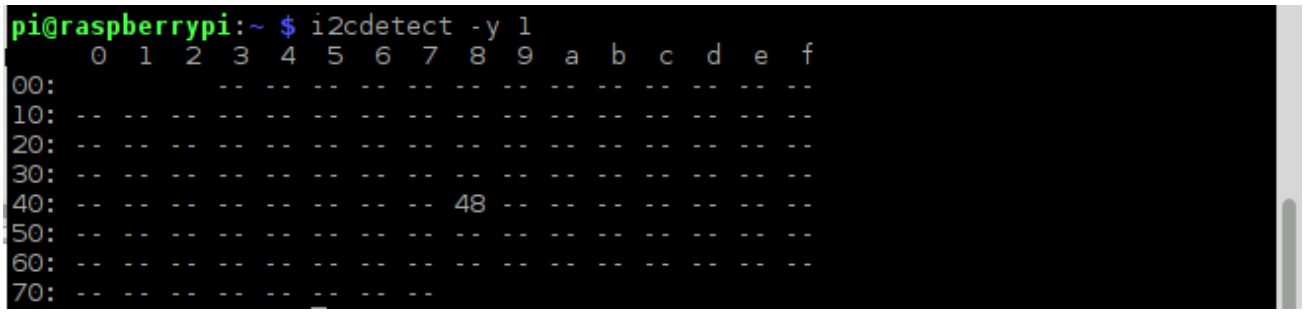
Next, type the command to install I2C-Tools. It is available with the Raspberry Pi OS by default.

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

I2C device address detection:

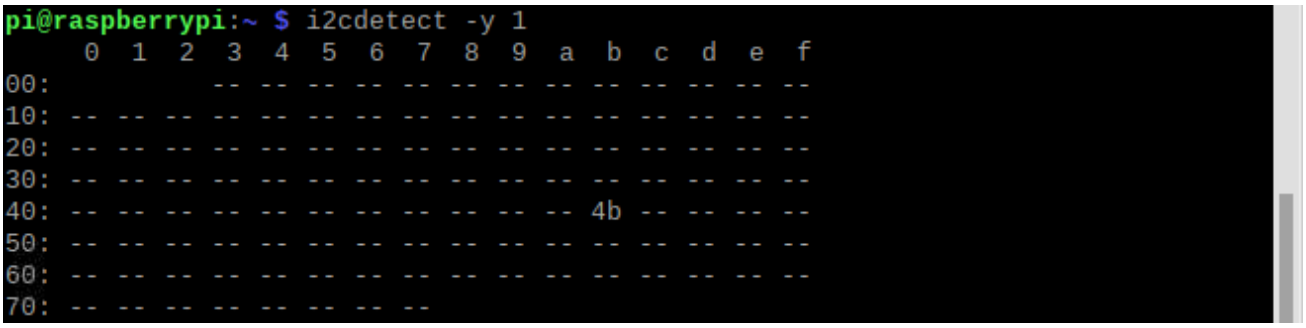
```
i2cdetect -y 1
```

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



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

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



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

Code

C Code 7.1.1 ADC

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

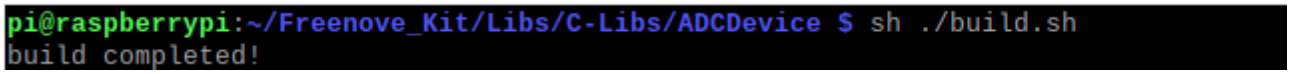
1. Use cd command to enter folder of the ADC Device library.

```
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:



Next, we will execute the code for this project.

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

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

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

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

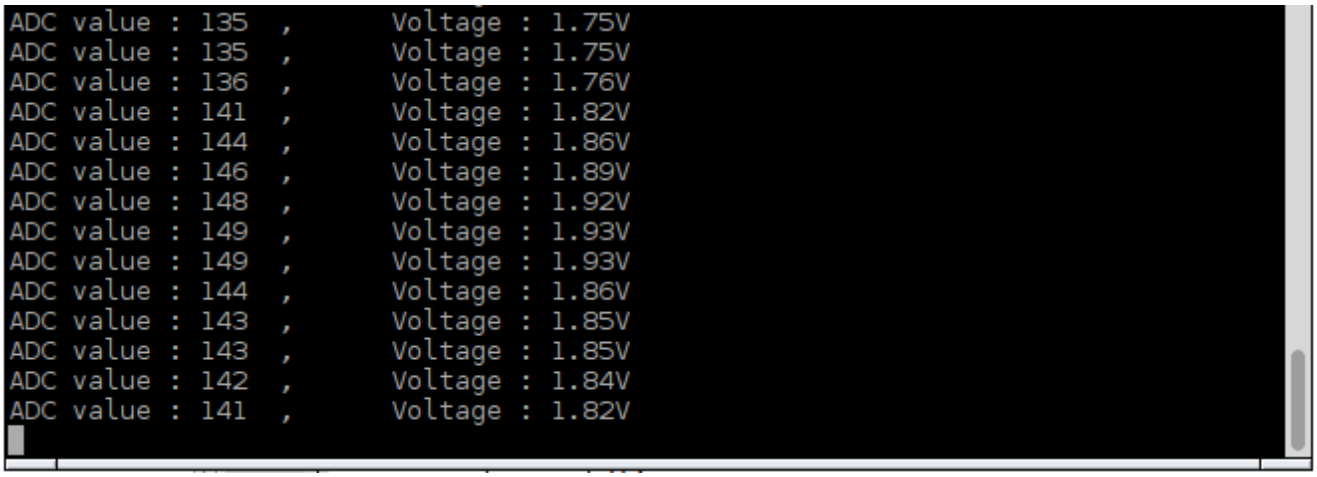
2. Use following command to compile "ADC.cpp" and generate the executable file "ADC".

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

3. Then run the generated file "ADC".

```
sudo ./ADC
```

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



The following is the code:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Reference

class ADCDevice

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

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

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

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

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

class PCF8591:public ADCDevice
class ADS7830:public ADCDevice

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

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

This returns the value read on the supplied analog input pin.
Parameter chn: For PCF8591, the range of chn is 0, 1, 2, 3. For ADS7830, the range of is 0, 1, 2, 3, 4, 5, 6, 7.

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

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

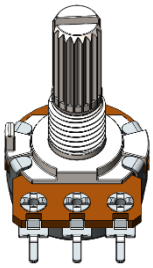
Chapter 8 Potentiometer & LED

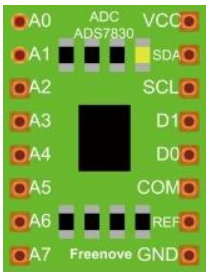
Earlier we learned how to use ADC and PWM. In this chapter, we learn to control the brightness of an LED by using a potentiometer.

Project 8.1 Soft Light

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

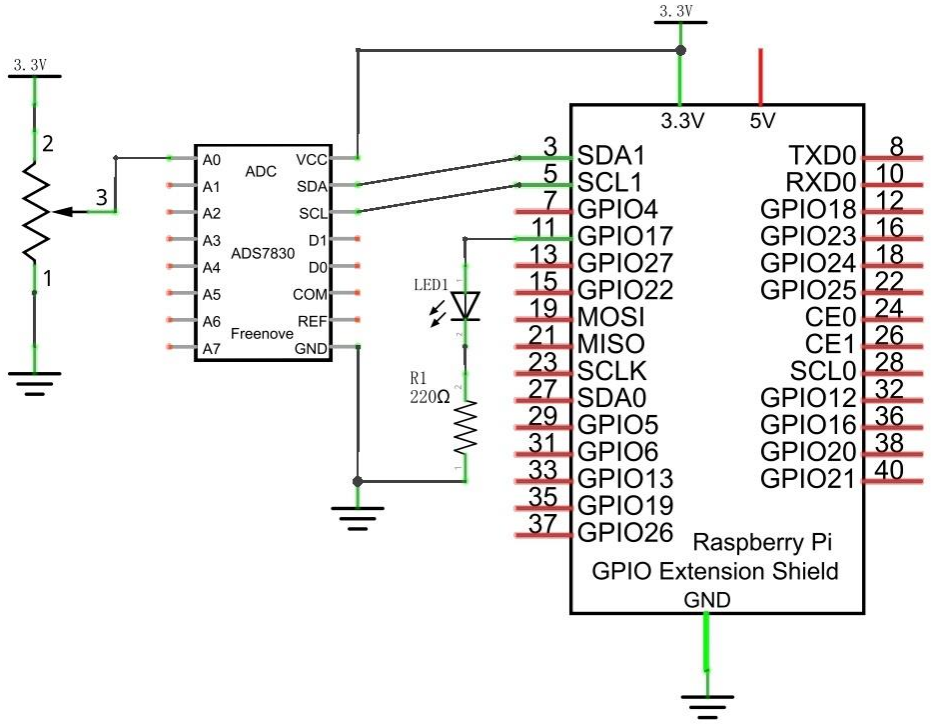
Component List

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

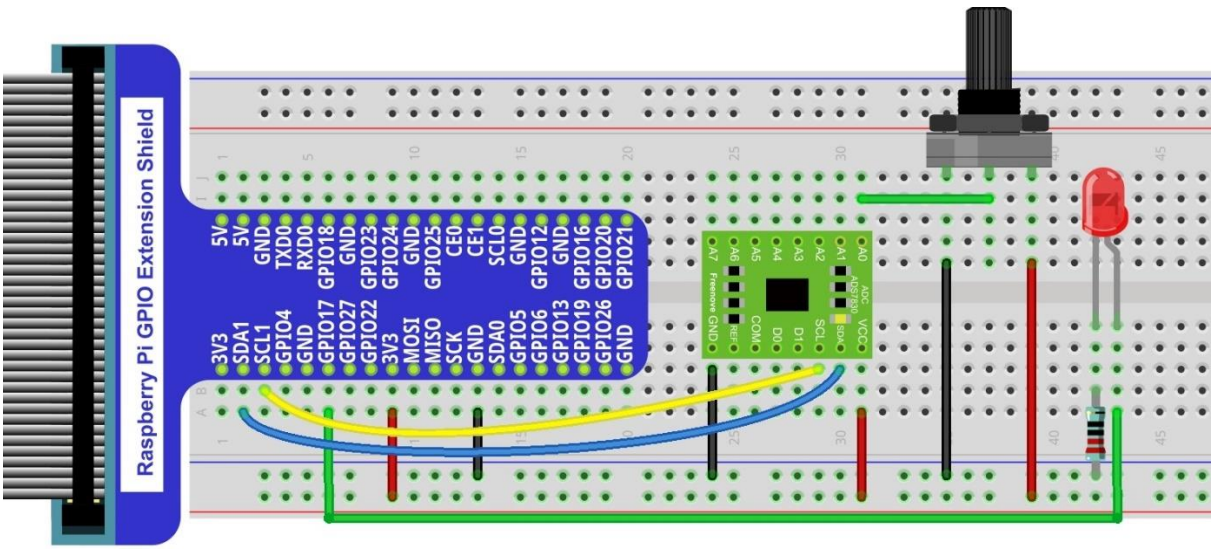
or

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

Circuit with ADS7830

Schematic diagram



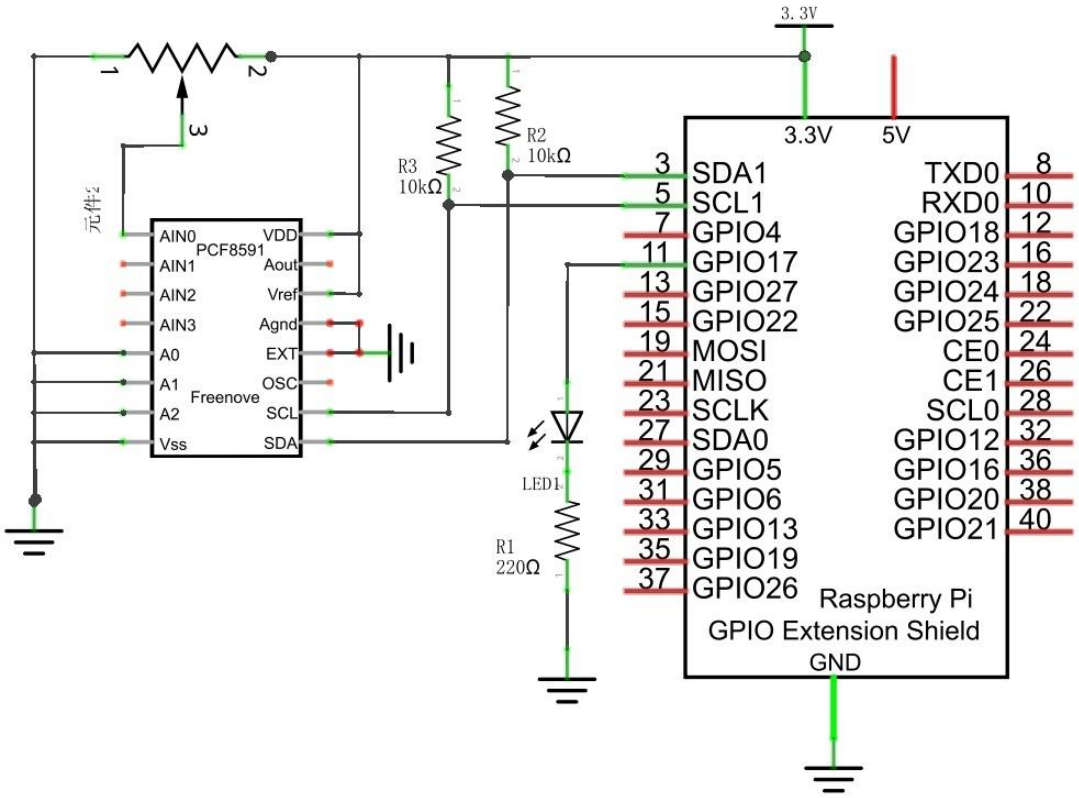
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



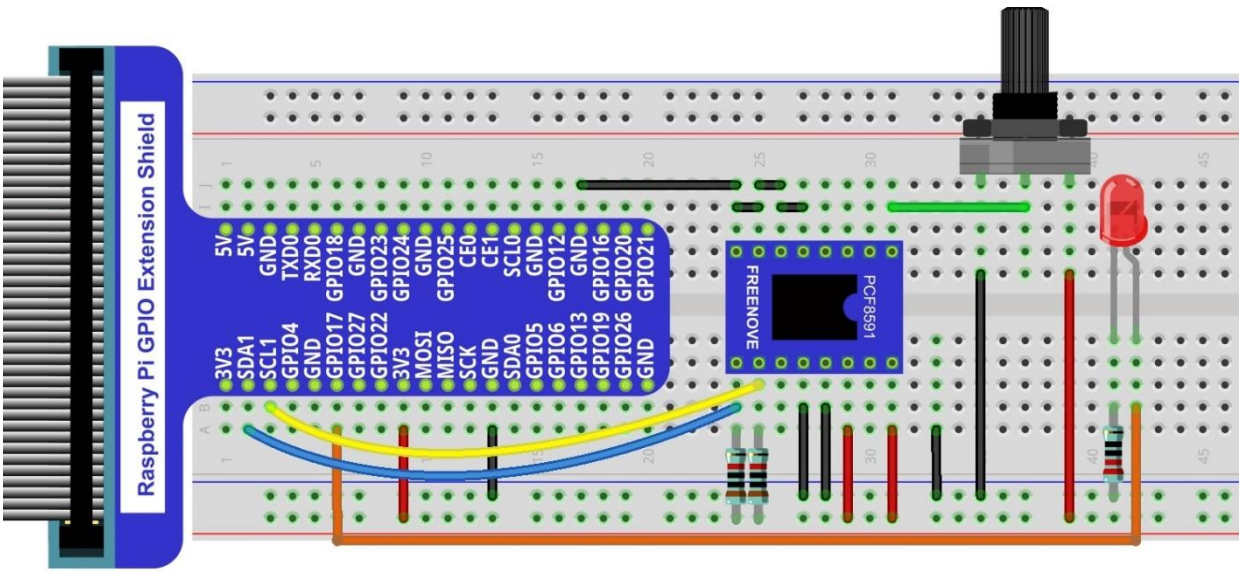
Video: https://youtu.be/YMEfe9IWU6I

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:

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

#define ledPin 0

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

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

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

```
26          return -1;
27      }
28      wiringPiSetup();
29      softPwmCreate(ledPin,0,100);
30      while(1){
31          int 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
```

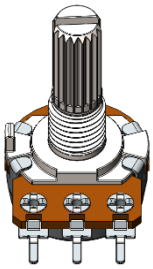
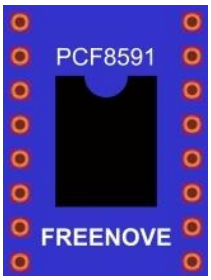
Chapter 9 Potentiometer & RGBLED

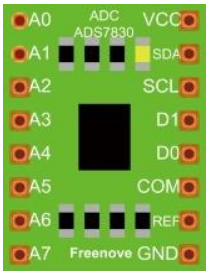
In this chapter, we will use 3 potentiometers to control the brightness of 3 LEDs of RGBLED to create multiple colors.

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.

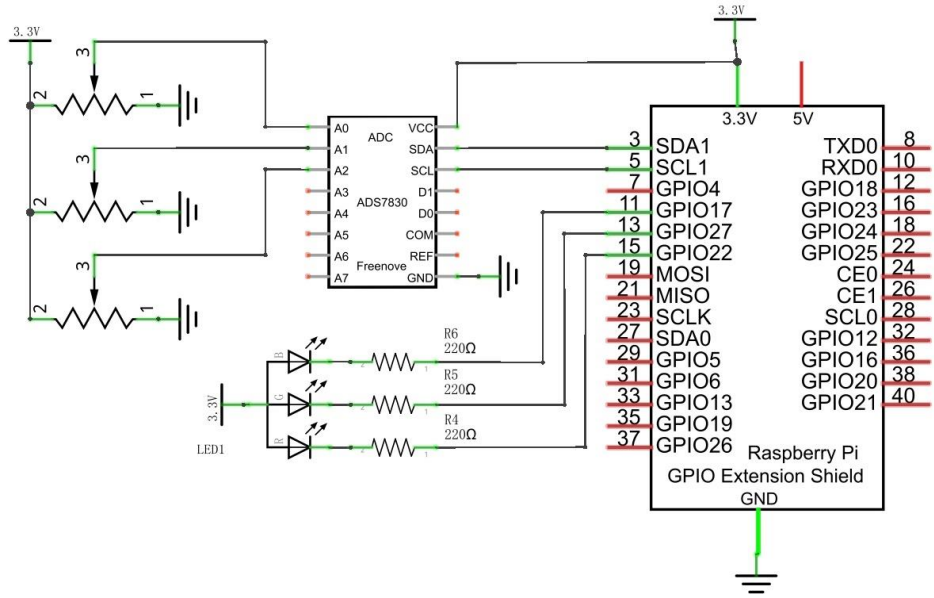
Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="4">Jumper Wires M/M x17
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td colspan="2">ADC module x1
 

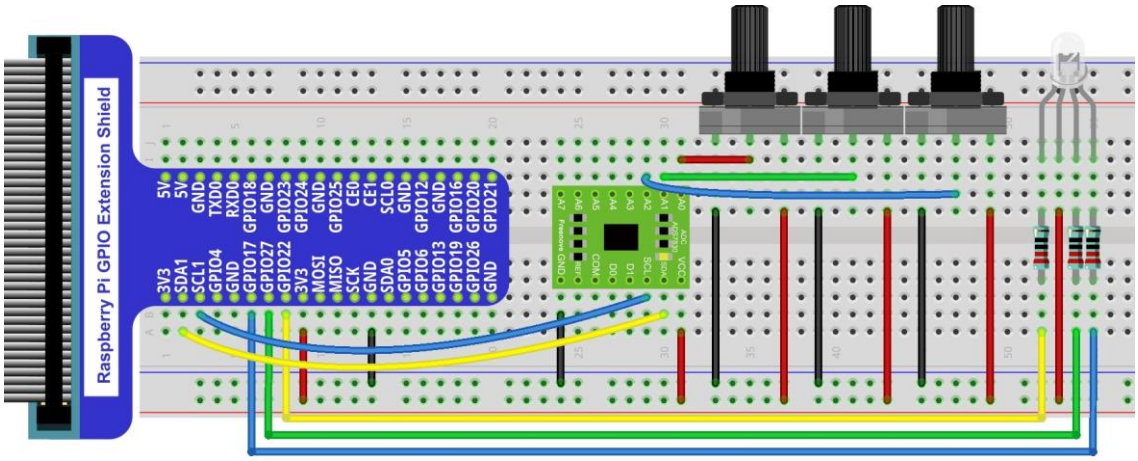
or

</td><td>10kΩ x2
  </td><td>220Ω x3
  </td><td>RGB LEDx1
  </td></tr>
</table>

Circuit with ADS7830

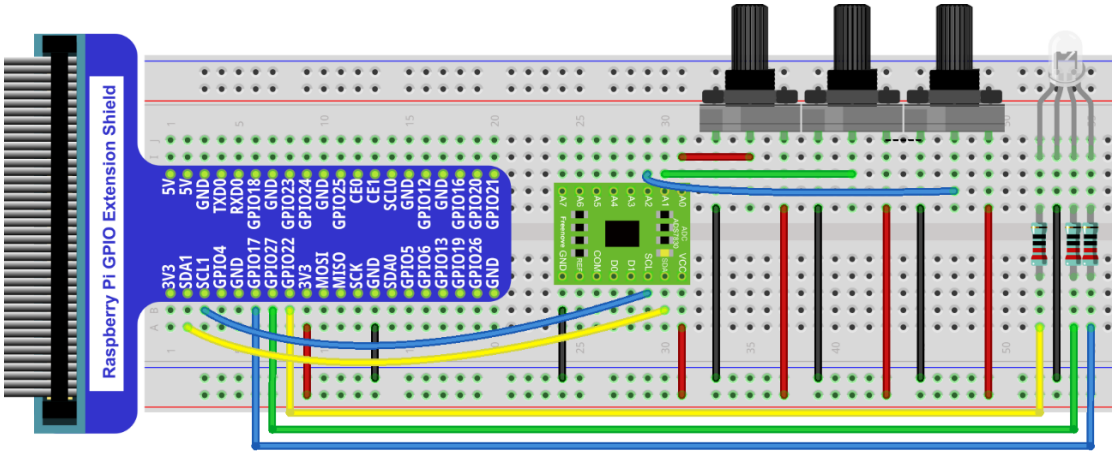
Schematic diagram



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

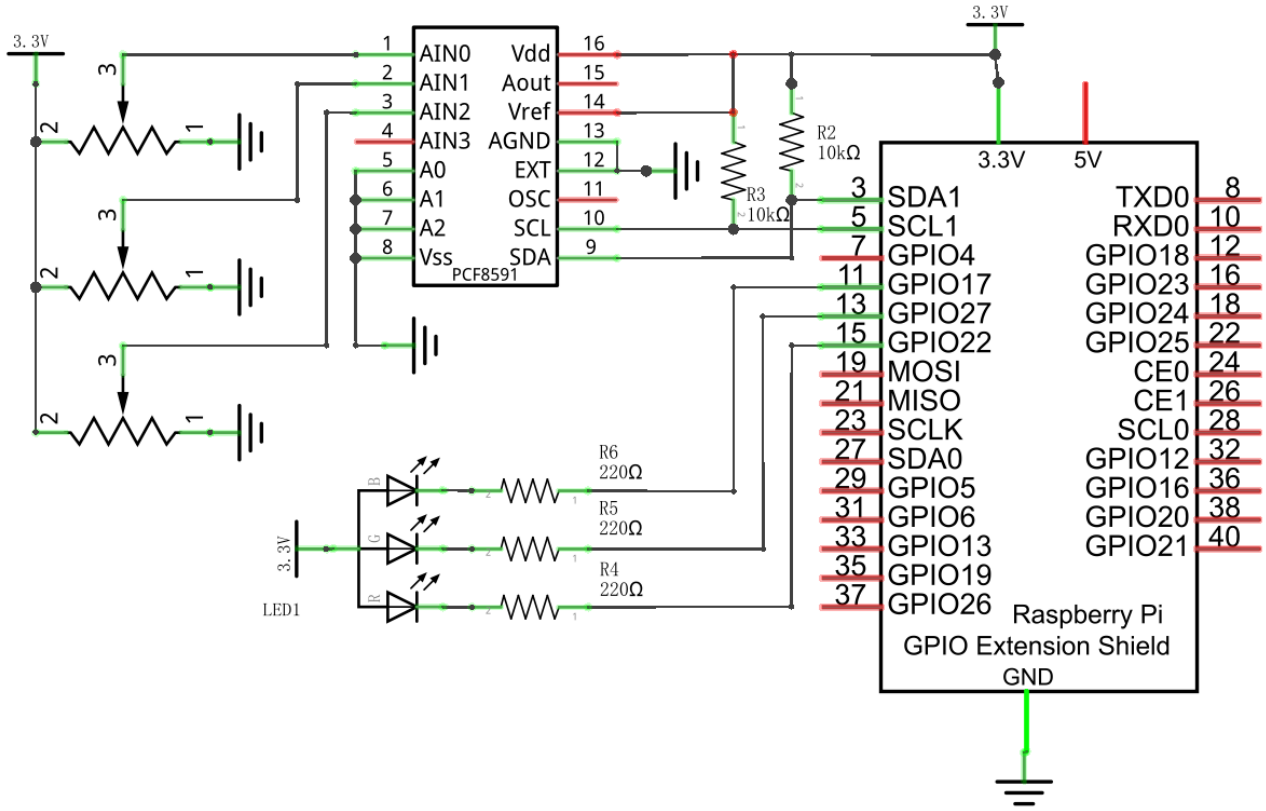


If circuit above doesn't work, please try following wiring.

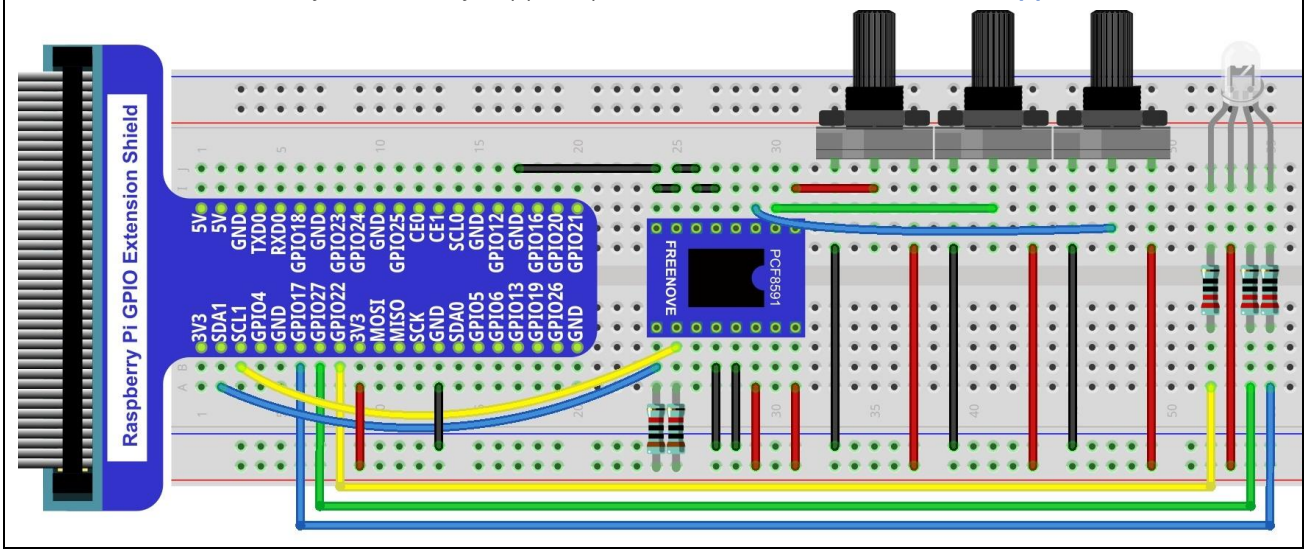


Circuit with PCF8591

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 potentiometers
    into PWM value and output it
39          softPwmWrite(ledGreenPin, val_Green*100/255);
40          softPwmWrite(ledBluePin, val_Blue*100/255);
41          //print out the read ADC value
42          printf("ADC value val_Red: %d  ,\tval_Green: %d  ,\tval_Blue: %d
43  \n", val_Red, val_Green, val_Blue);
44          delay(100);
45      }
46      return 0;
47  }
```

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

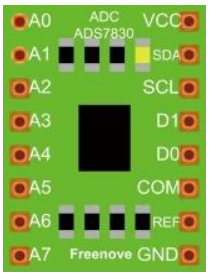
In this chapter, we will learn how to use a photoresistor to make an automatic dimming nightlight.

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

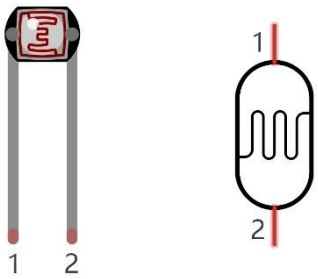
<table>
<tr><td colspan="3">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x15
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">ADC module x1
 

or

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

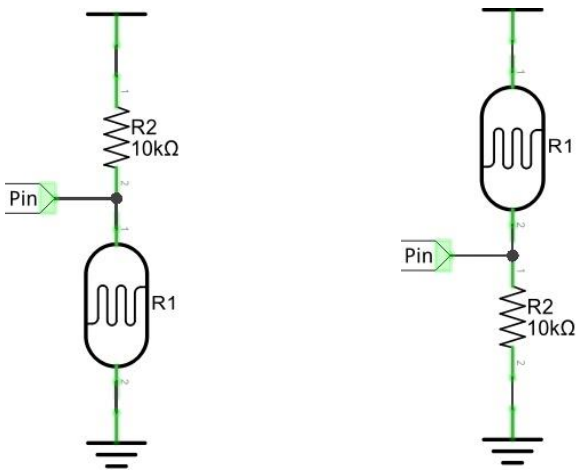
Component knowledge

Photoresistor

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



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

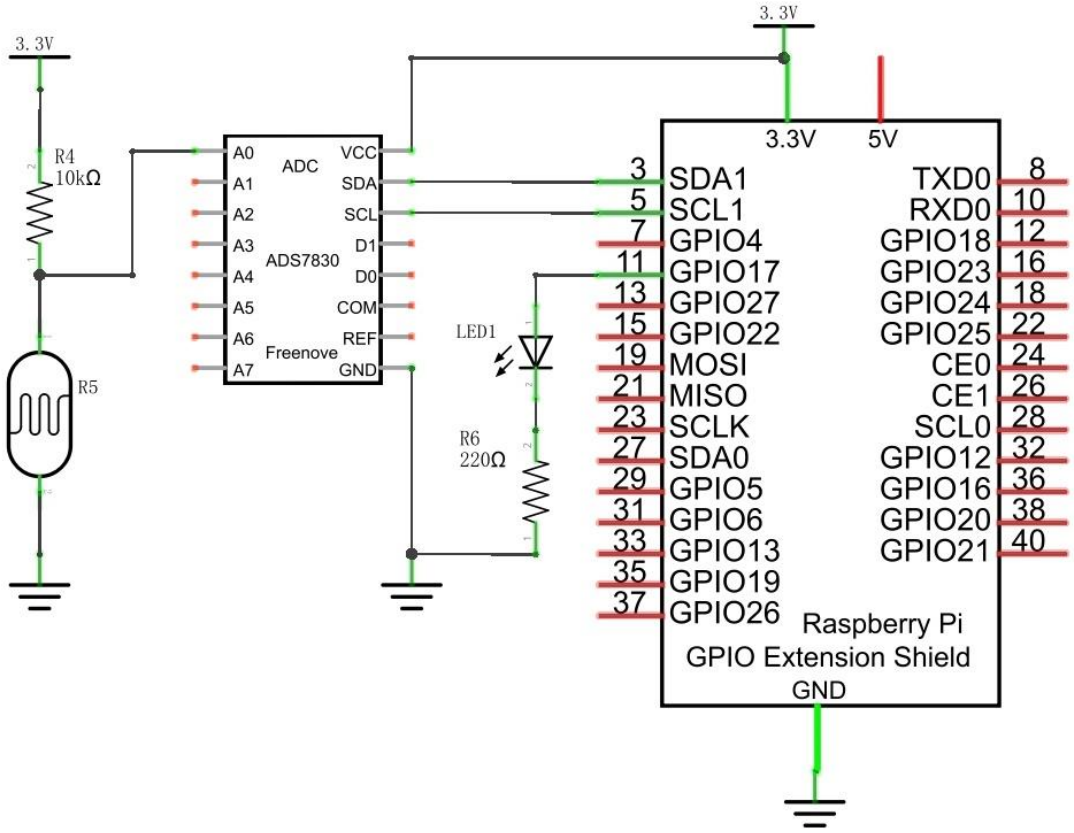


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

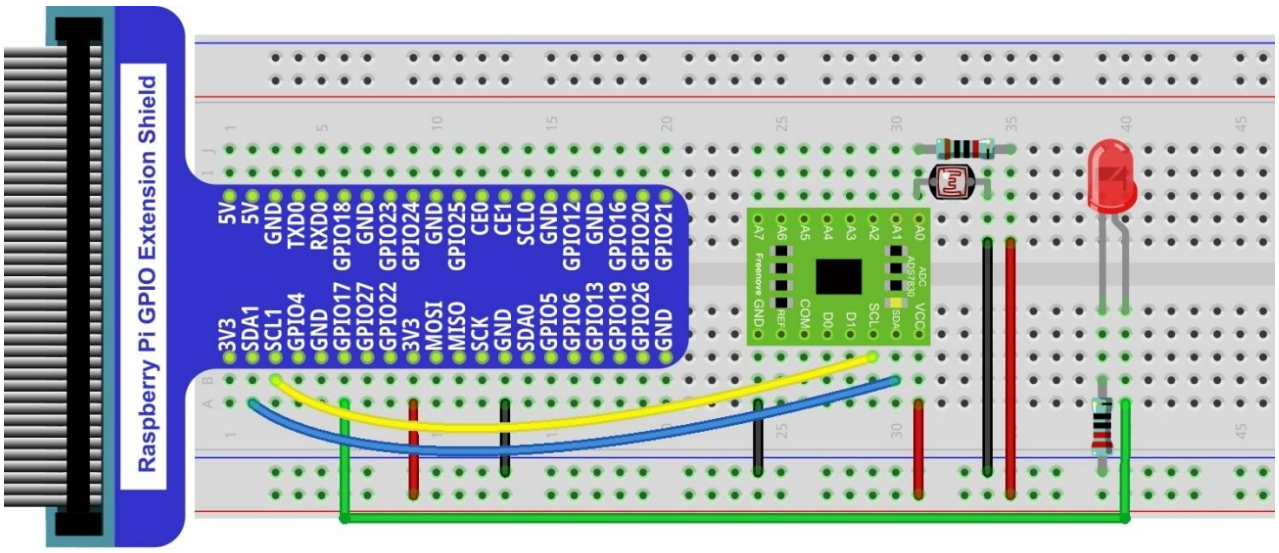
Circuit with ADS7830

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

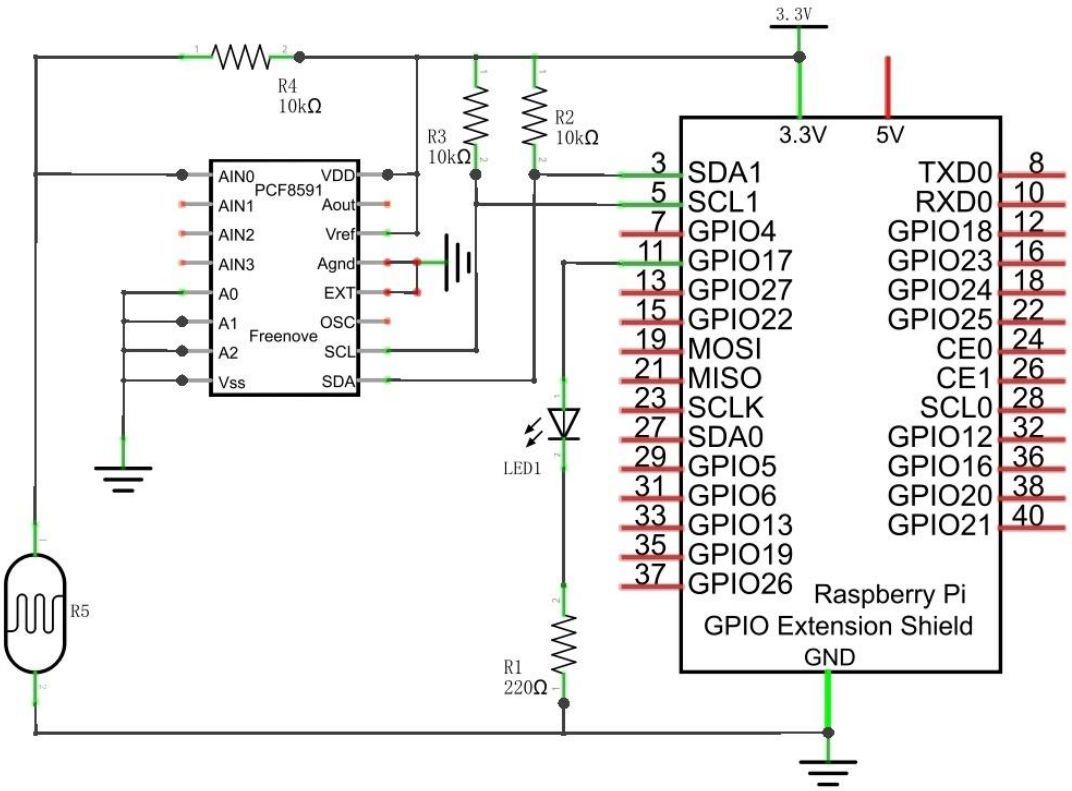


Video: https://youtu.be/r6p3zhXsyko

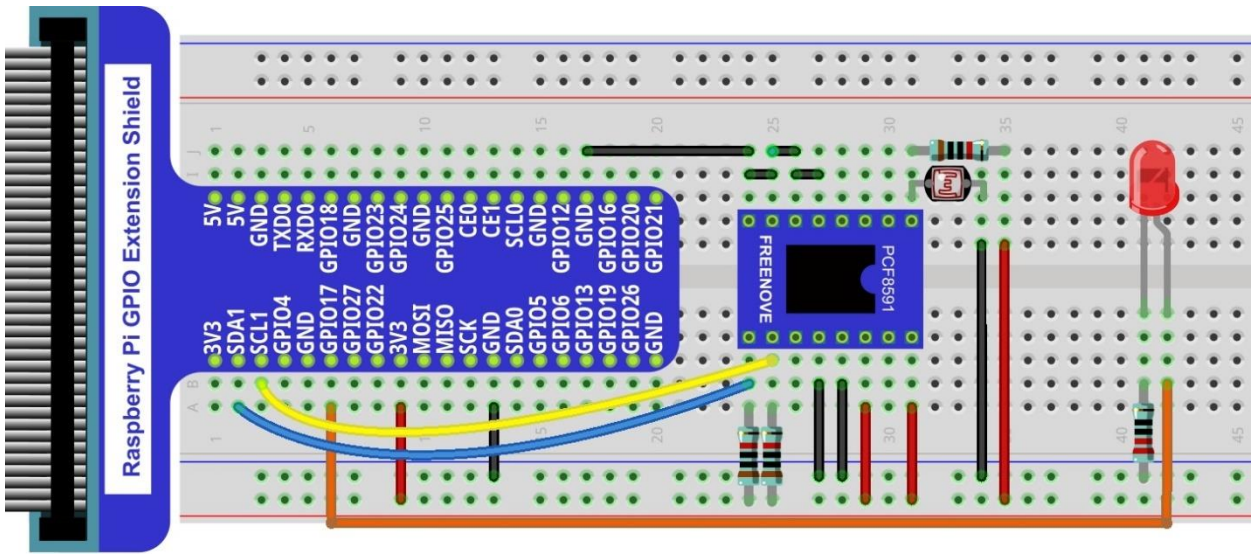
Circuit with PCF8591

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:

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

#define ledPin 0

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

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

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

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

Chapter 11 Thermistor

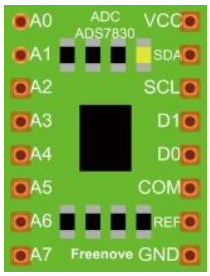
In this chapter, we will learn about Thermistors which are another kind of Resistor.

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

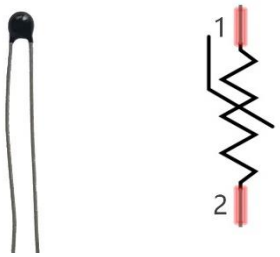
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper Wire M/M x14
  </td></tr>
<tr><td>Thermistor x1
  </td><td colspan="2">ADC module x1
 

or

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

Component knowledge

Thermistor

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



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

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

Where:

Rt is the thermistor resistance under T2 temperature;

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

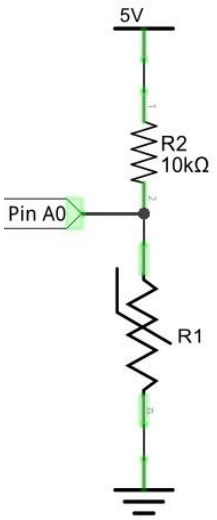
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



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

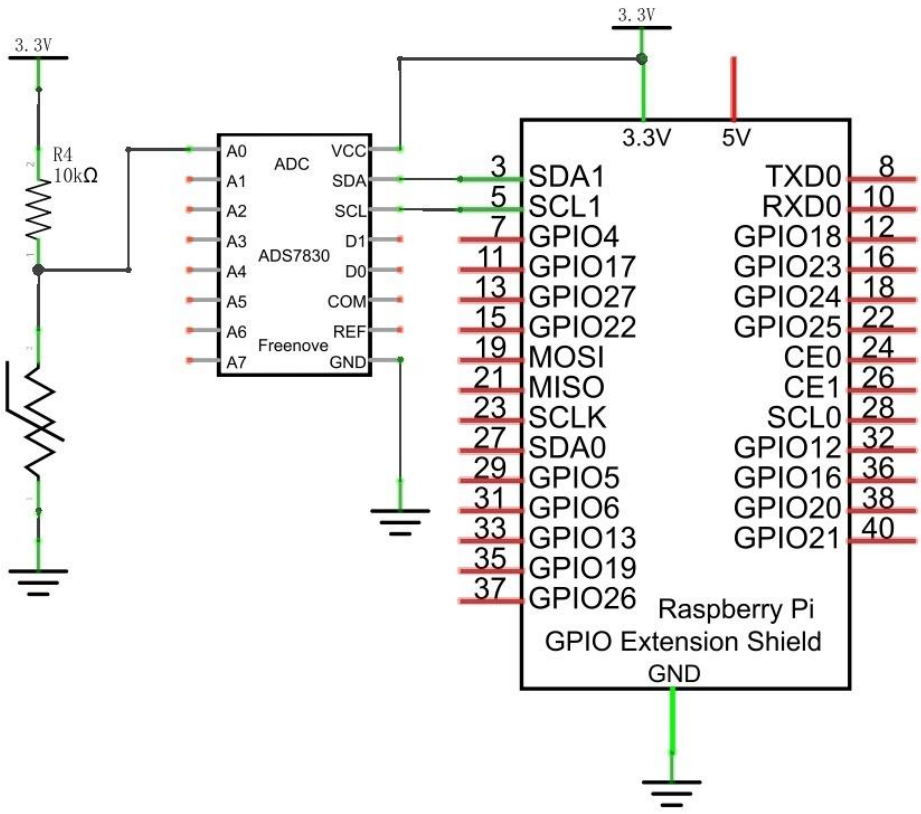
Therefore, the temperature formula can be derived as:

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

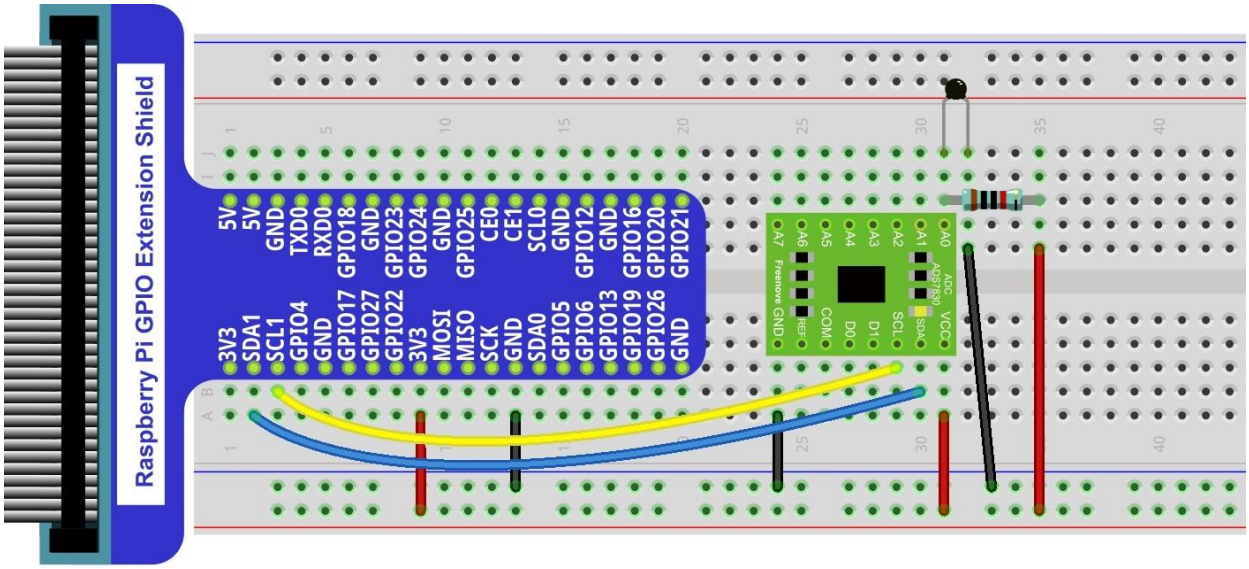
Circuit with ADS7830

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 feel free to contact us via: support@freenove.com



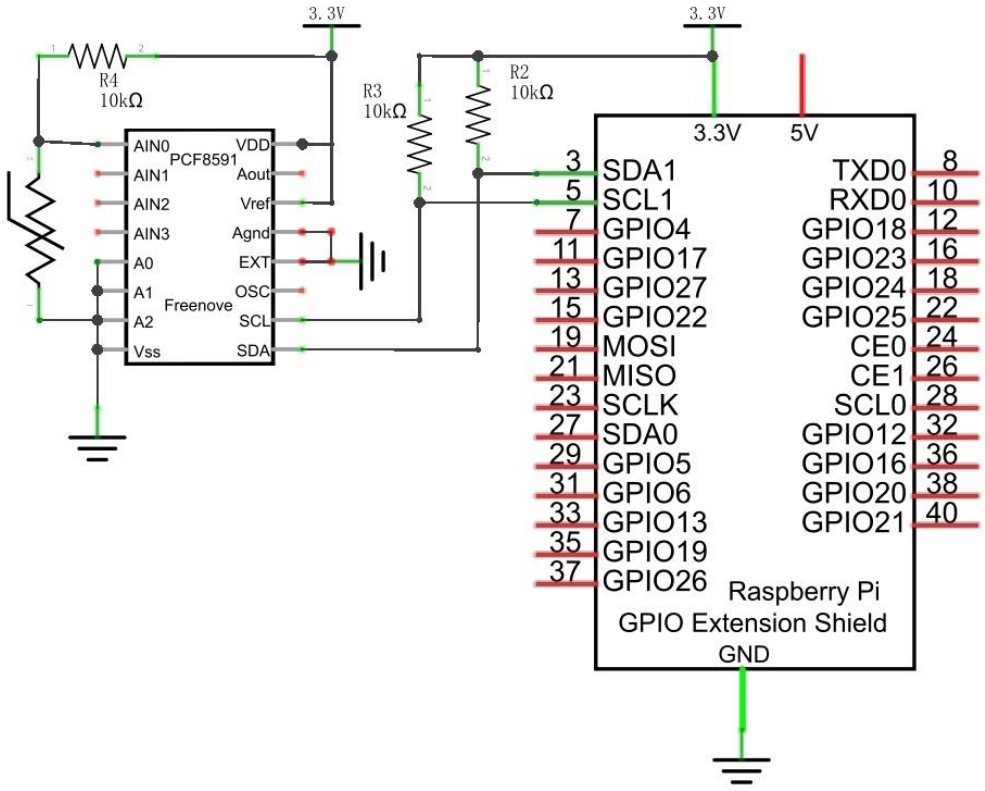
Thermistor has longer pins than the one shown in circuit.

Video: https://youtu.be/spOaIxanNMc

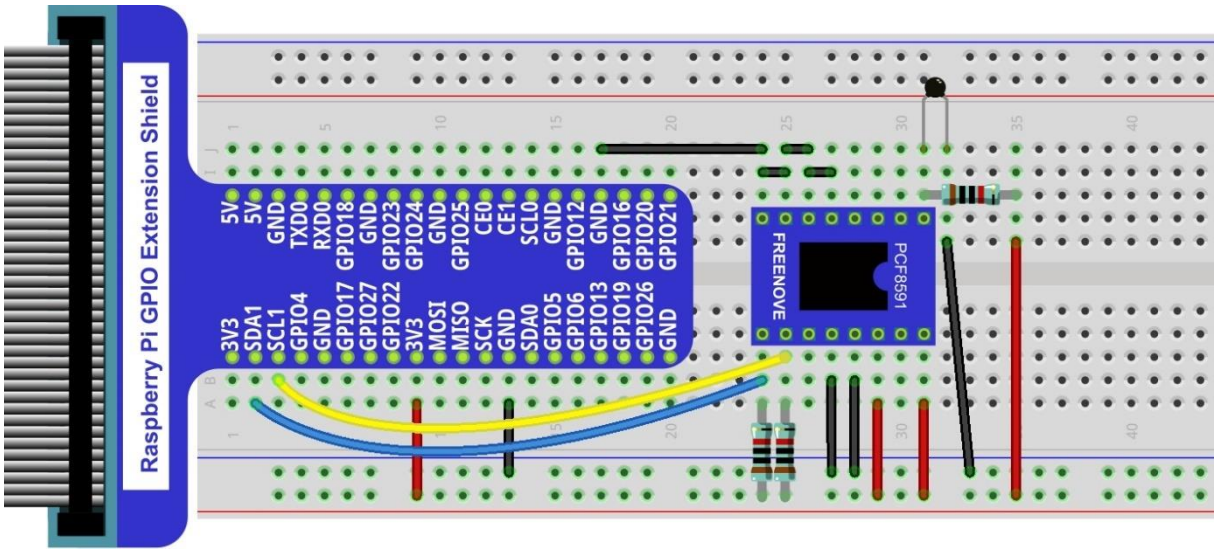
Circuit with PCF8591

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_Thermometer directory of C code.

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

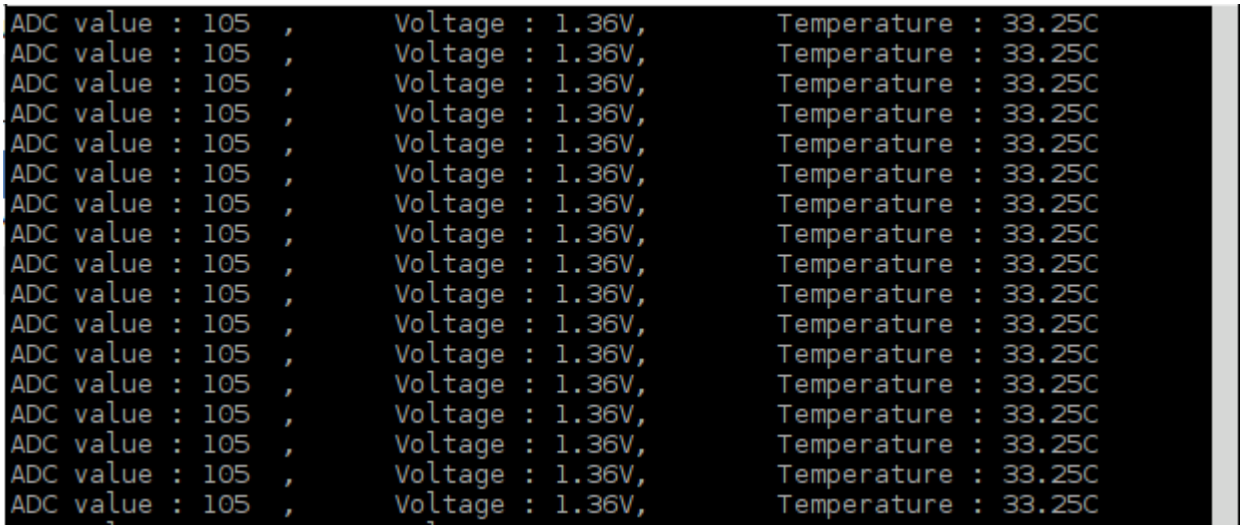
2 Use following command to compile "Thermometer.cpp" and generate executable file "Thermometer".

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

3 Then run the generated file "Thermometer".

```
sudo ./Thermometer
```

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



The following is the code:

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

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

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

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

In the code, 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.

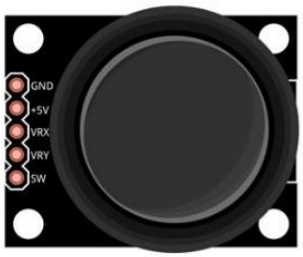
Chapter 12 Joystick

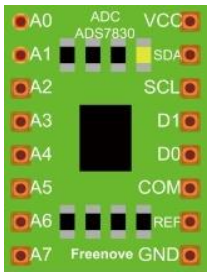
In an earlier chapter, we learned how to use Rotary Potentiometer. We will now learn about joysticks, which are electronic modules that work on the same principle as the Rotary Potentiometer.

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

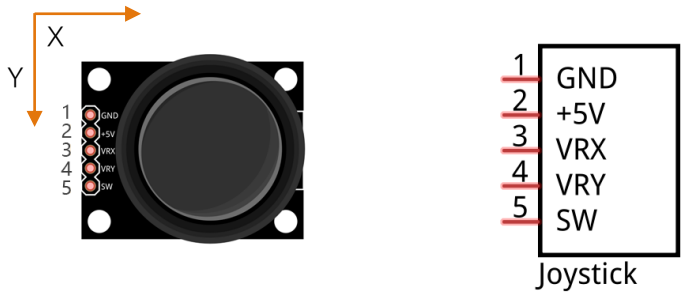
<table>
<tr><td>Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="2">Jumper x18
  </td></tr>
<tr><td>Joystick x1
 
</td><td>ADC module x1
 

or

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

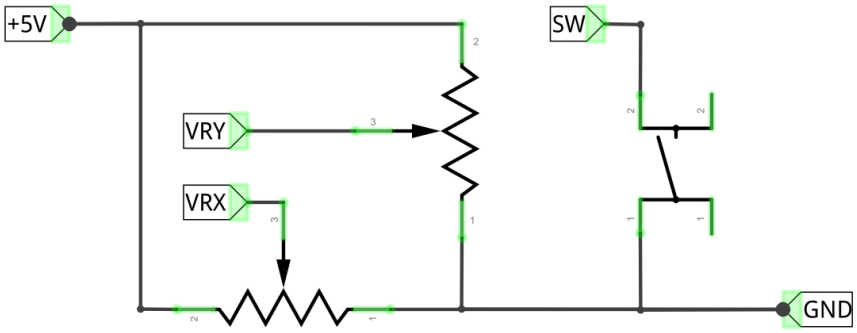
Component knowledge

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)**.



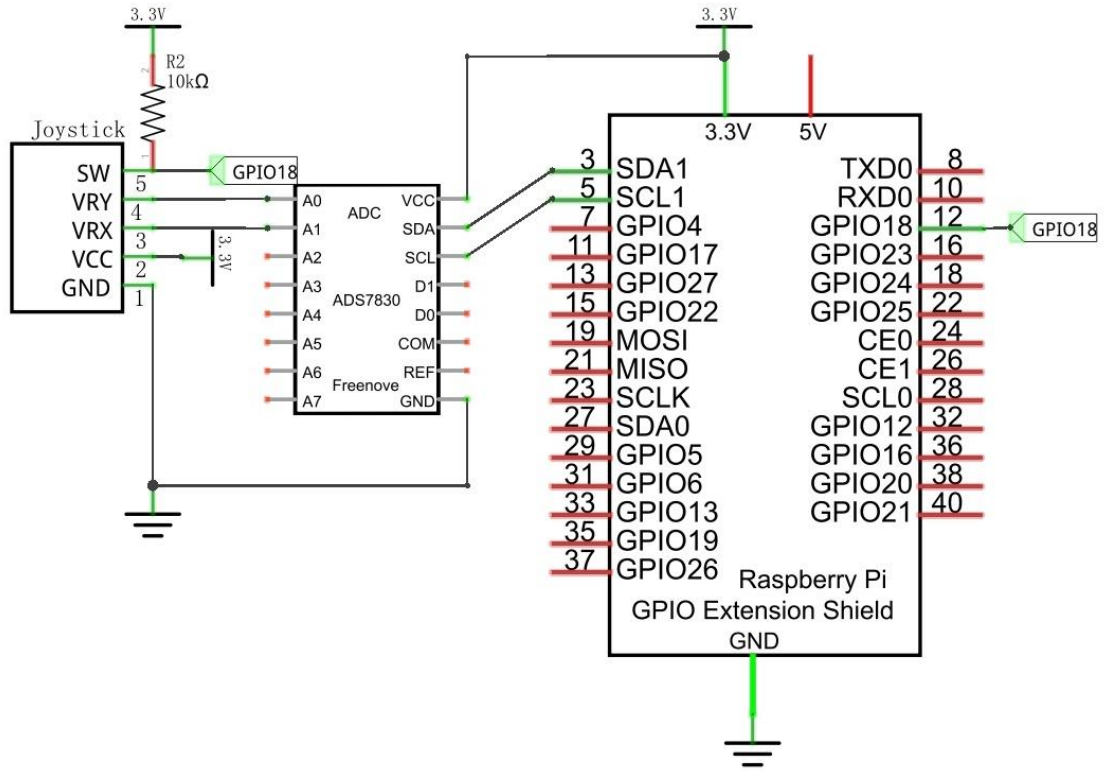
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



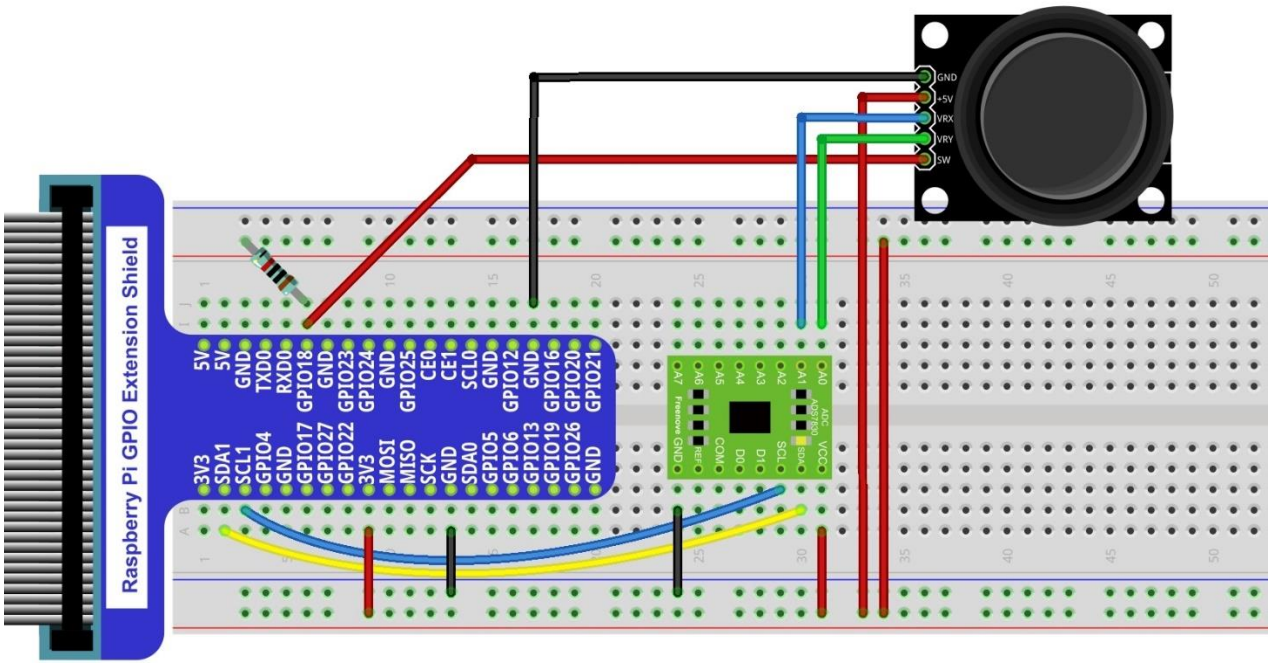
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit with ADS7830

Schematic diagram



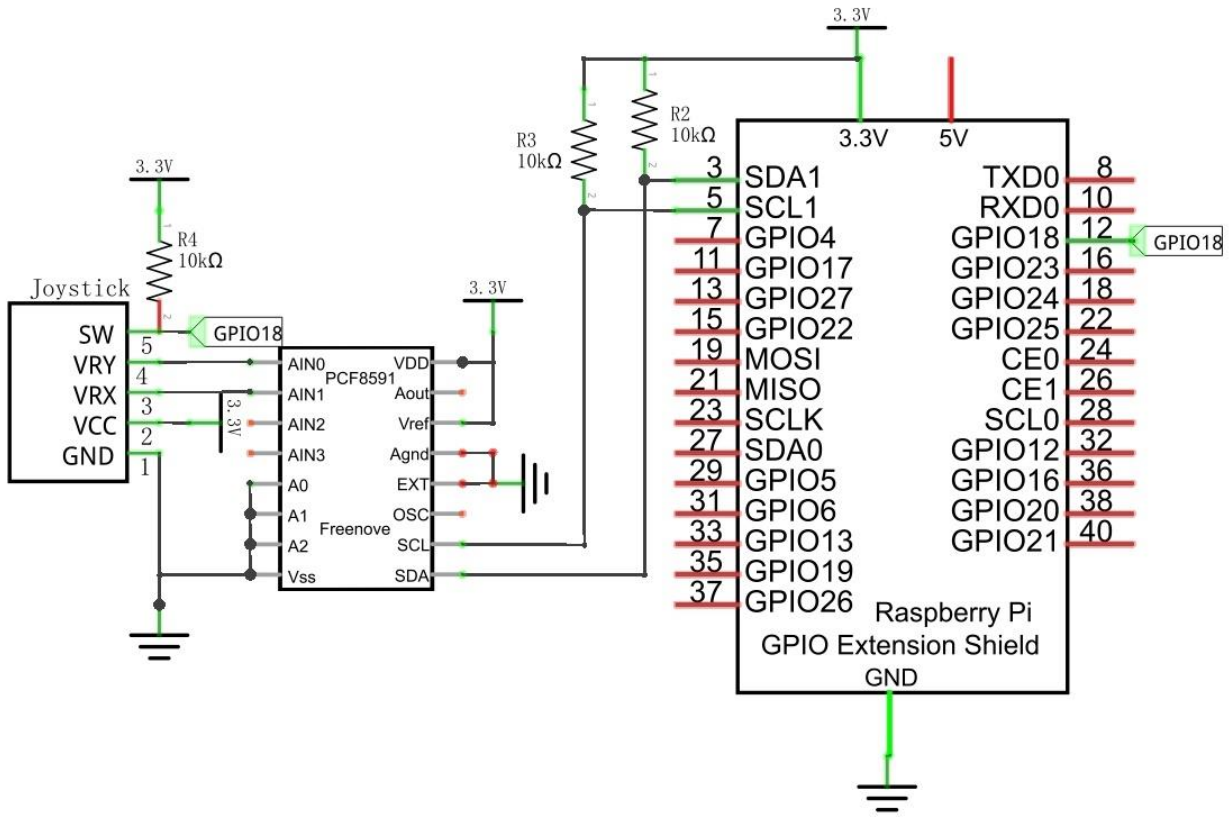
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



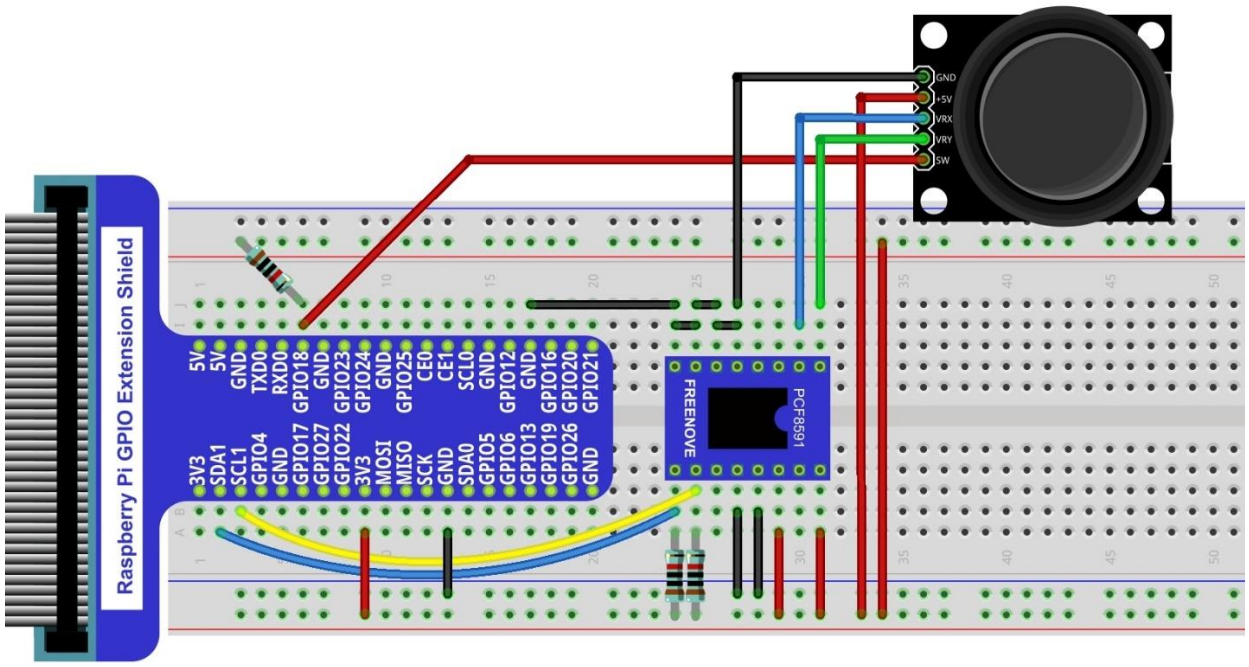
Video: https://youtu.be/qjP3HpbPJTM

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
```

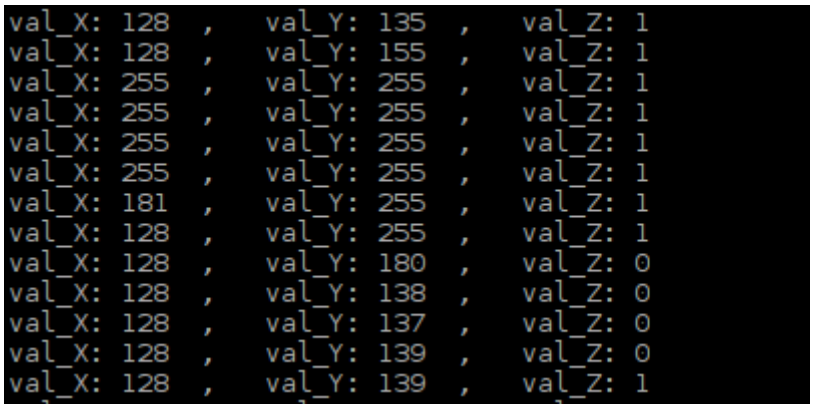
2. Use following command to compile "Joystick.cpp" and generate executable file "Joystick".

```
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.



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
```

```
    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;                 // Free previously pointed memory
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;                 // Free previously pointed memory
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    pinMode(Z_Pin,INPUT);        //set Z_Pin as input pin and pull-up mode
    pullUpDnControl(Z_Pin,PUD_UP);
    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);
    }
    return 0;
}
```

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);
    }
```

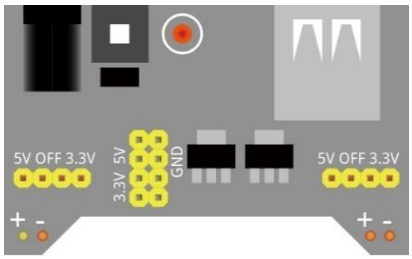
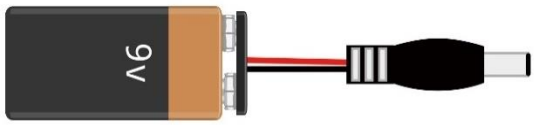
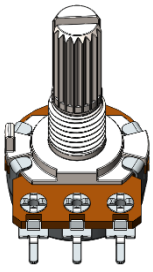
Chapter 13 Motor & Driver

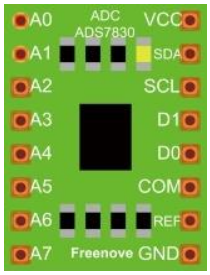
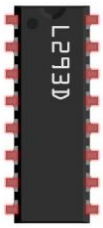
In this chapter, we will learn about DC Motors and DC Motor Drivers and how to control the speed and direction of a DC Motor.

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

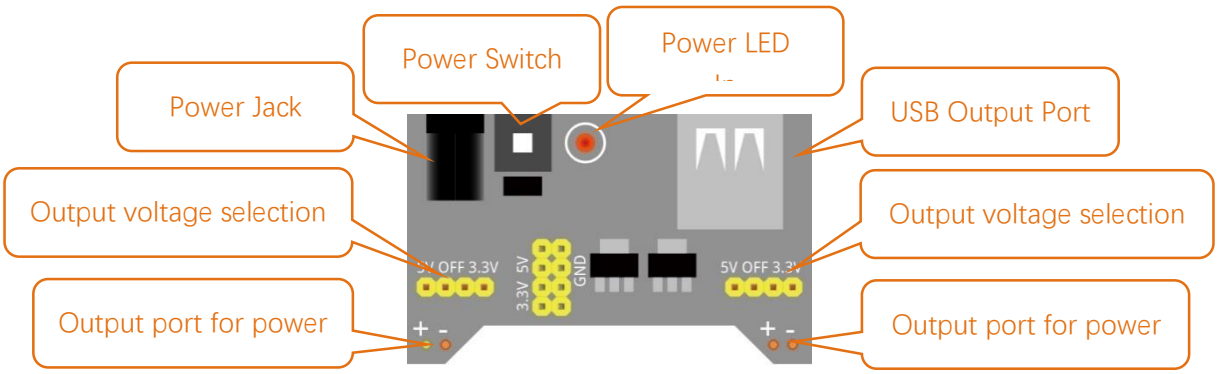
<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires x23
  </td></tr>
<tr><td colspan="2">Breadboard Power Module x1
 
</td><td colspan="3">9V Battery (you provide) & 9V Battery Cable
 
</td></tr>
<tr><td>Rotary Potentiometer x1
  </td><td>DC Motor x1
  </td><td>10kΩ x2
  </td><td colspan="2">ADC Module x1
 

or

</td><td>L293D
IC Chip
 
</td></tr>
</table>

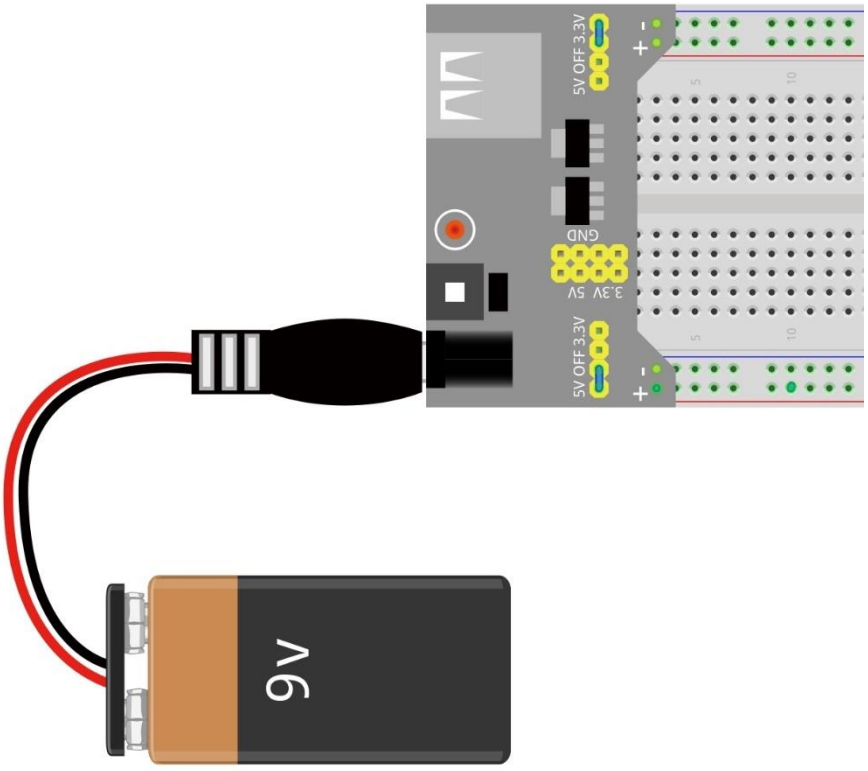
Component knowledge

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:

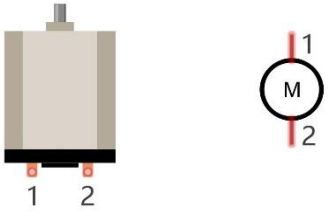


Here is an acceptable connection between Breadboard Power Module and Breadboard using a 9V battery and the provided power harness:



DC Motor

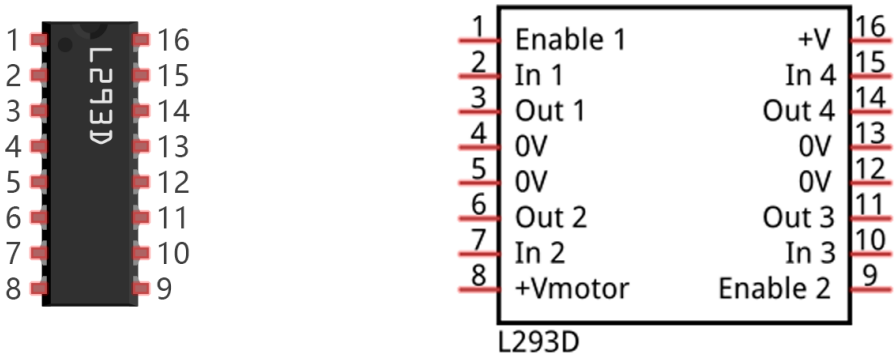
DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair 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).



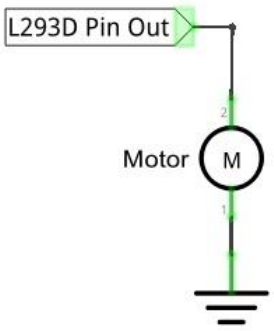
Port description of L293D module is as follows:

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

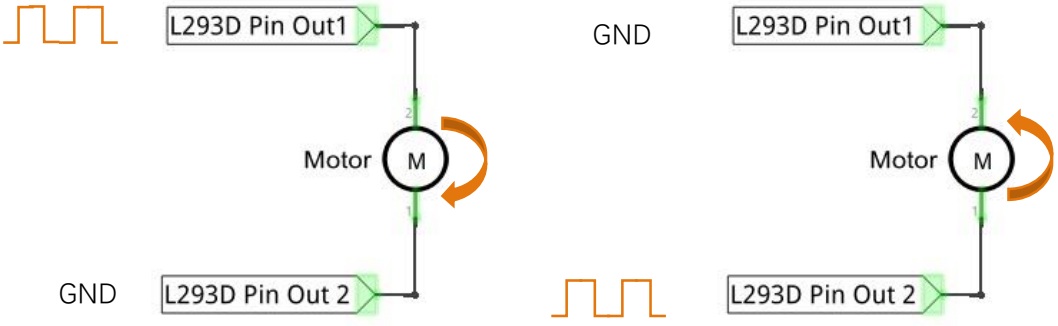
For more details, please see the datasheet for this IC Chip.

When using the L293D to drive a DC Motor, there are usually two connection options.

The following connection option uses one channel of the L239D, 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 L239D: 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.

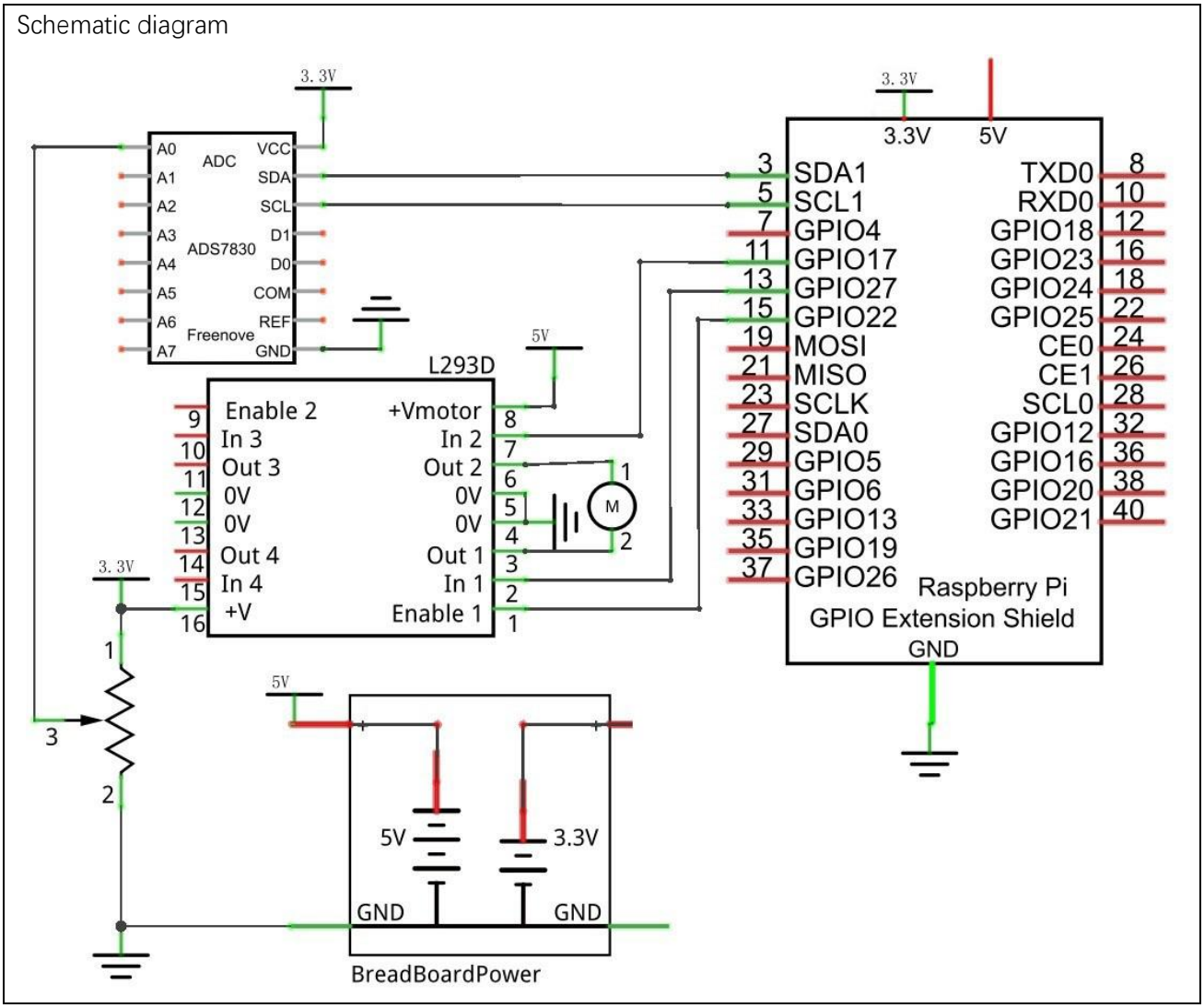


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting 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.

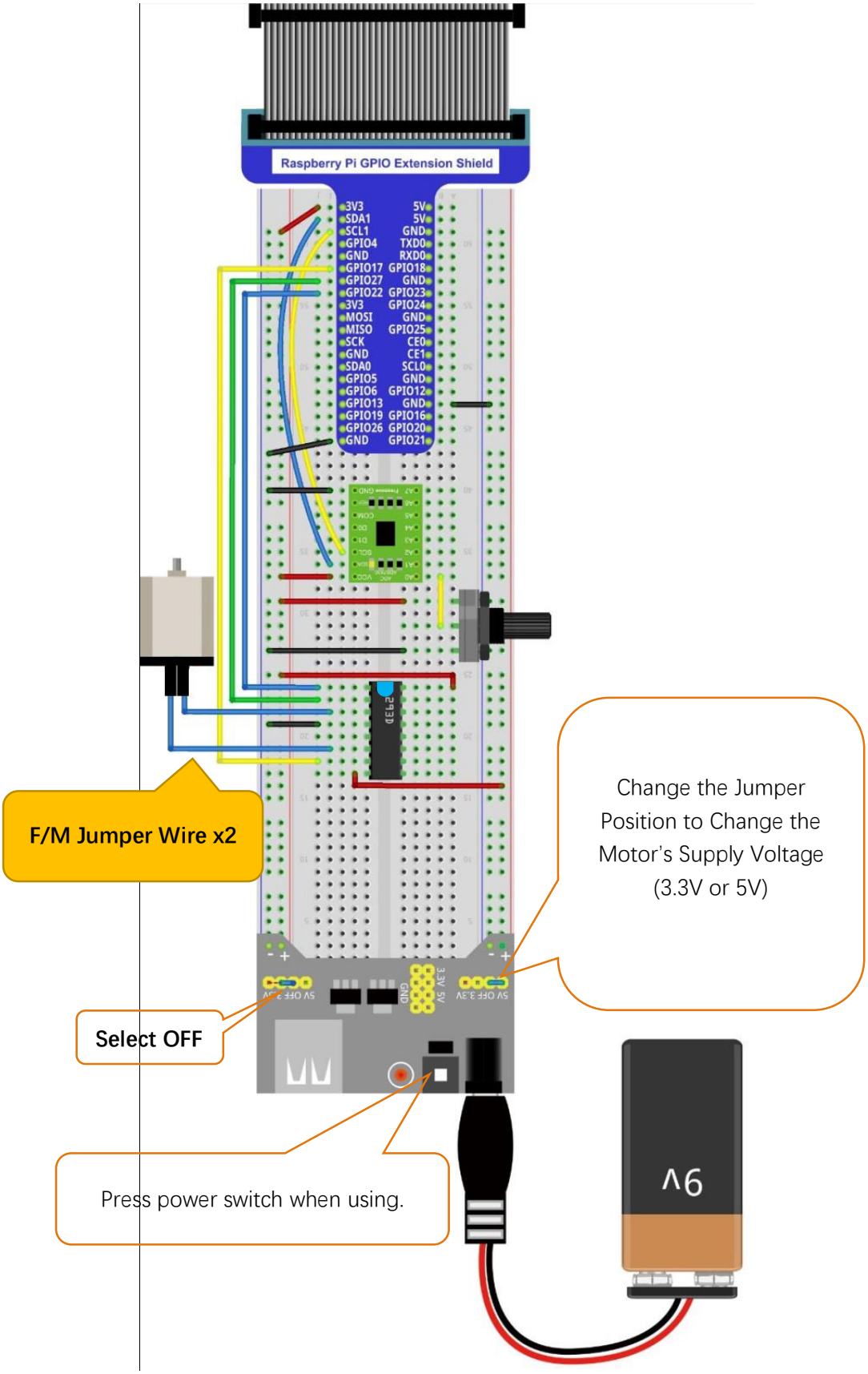
Circuit with ADS7830

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 feel free to contact us via: support@freenove.com

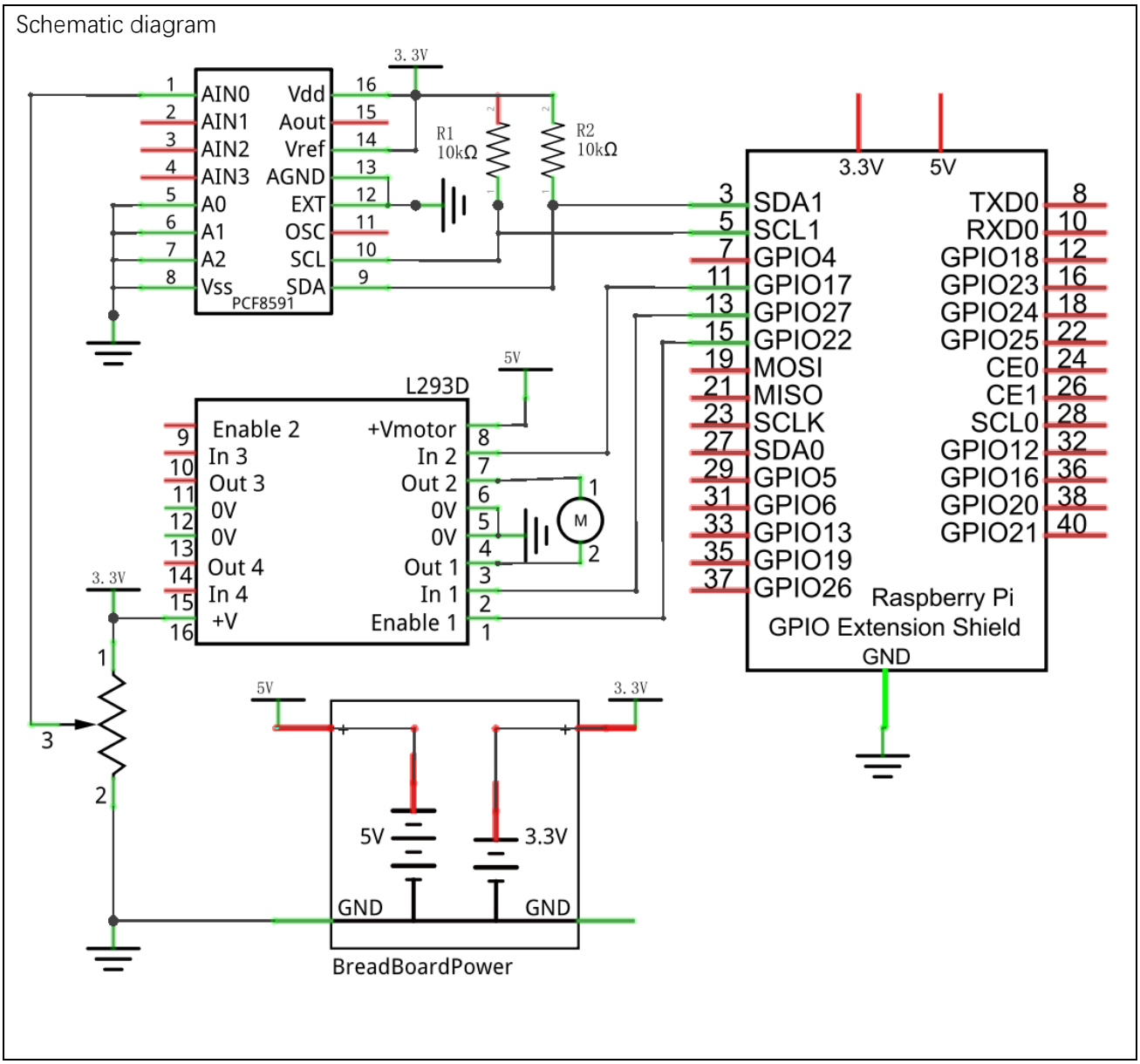


Video: https://youtu.be/d5lRMTDK-wg

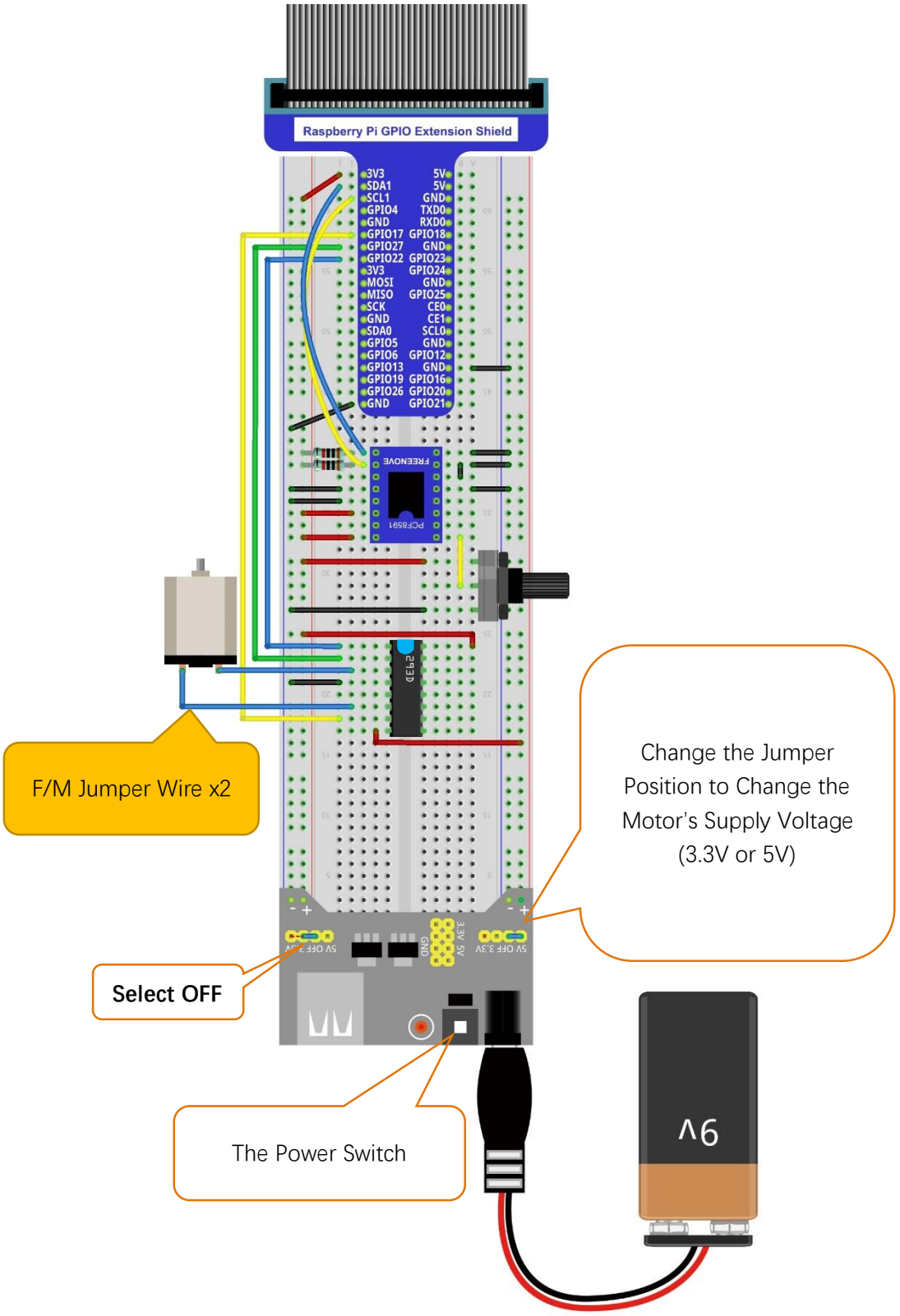
Circuit with PCF8591

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 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
```

```
#define motorPin1    2        //define the pin connected to L293D
#define motorPin2    0
#define enablePin    3

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

//Map function: map the value from a range to another range.
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
//motor function: determine the direction and speed of the motor according to the ADC
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        digitalWrite(motorPin1,HIGH);
        digitalWrite(motorPin2,LOW);
        printf("turn Forward...\n");
    }
    else if (value<0){
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,HIGH);
        printf("turn Back...\n");
    }
    else {
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,LOW);
        printf("Motor Stop...\n");
    }
    softPwmWrite(enablePin,map(abs(value),0,128,0,100));
    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);//print the PMW duty cycle
}
int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

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

```
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 PMW 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.
}
```

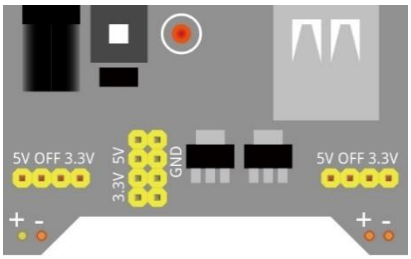
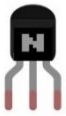
Chapter 14 Relay & Motor

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

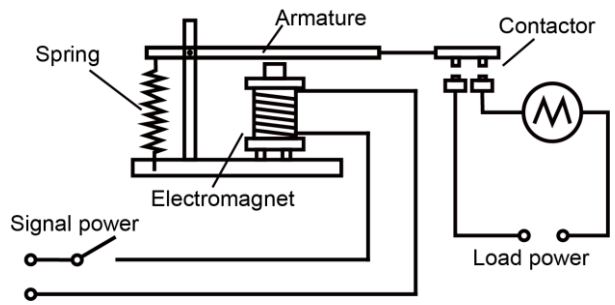
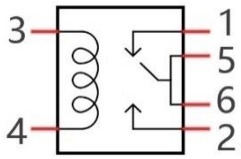
<table>
<tr><td colspan="3">Raspberry Pi (with 40 GPIO) x1
GPIO Expansion Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire x11
  </td></tr>
<tr><td colspan="6">9V battery (prepared by yourself) & battery line
  </td></tr>
<tr><td colspan="3">Breadboard Power module x1
  </td><td>Resistor 10kΩ x2
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 220Ω x1
  </td></tr>
<tr><td>NPN transistor x1
  </td><td>Relay x1
  </td><td>Motor x1
  </td><td>Push button x1
  </td><td>LED x1
  </td><td>Diode x1
  </td></tr>
</table>

Component knowledge

Relay

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
		

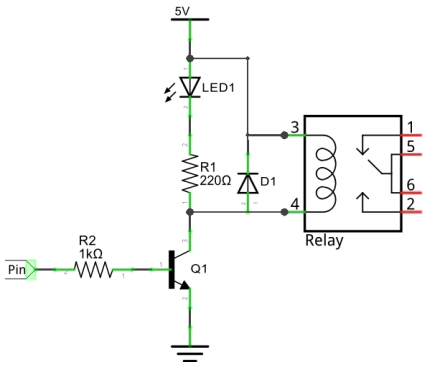
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

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: 1H=1000mH, 1mH=1000μH.

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.

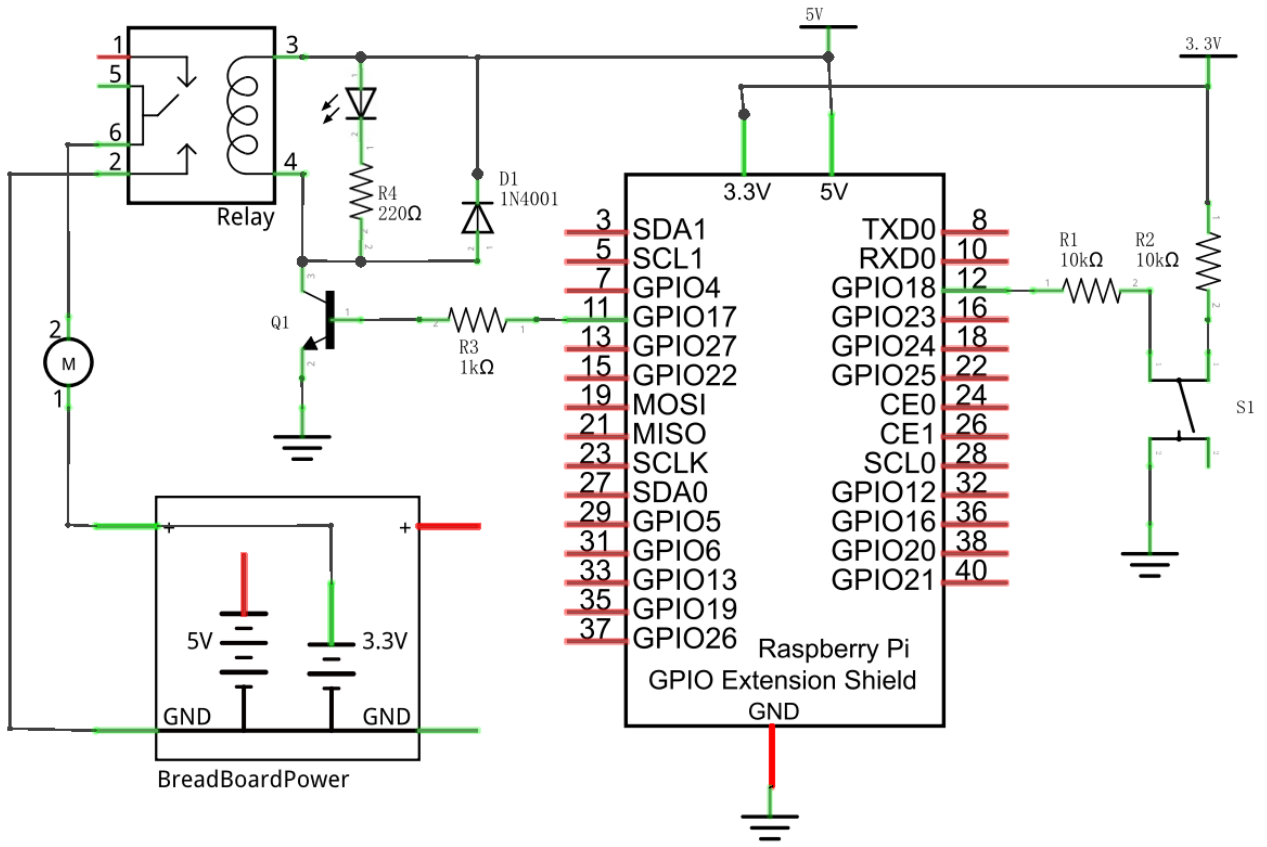
The circuit for a Relay is as follows: The coil of Relay can be equivalent to an Inductor, when a Transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay’s coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes 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.



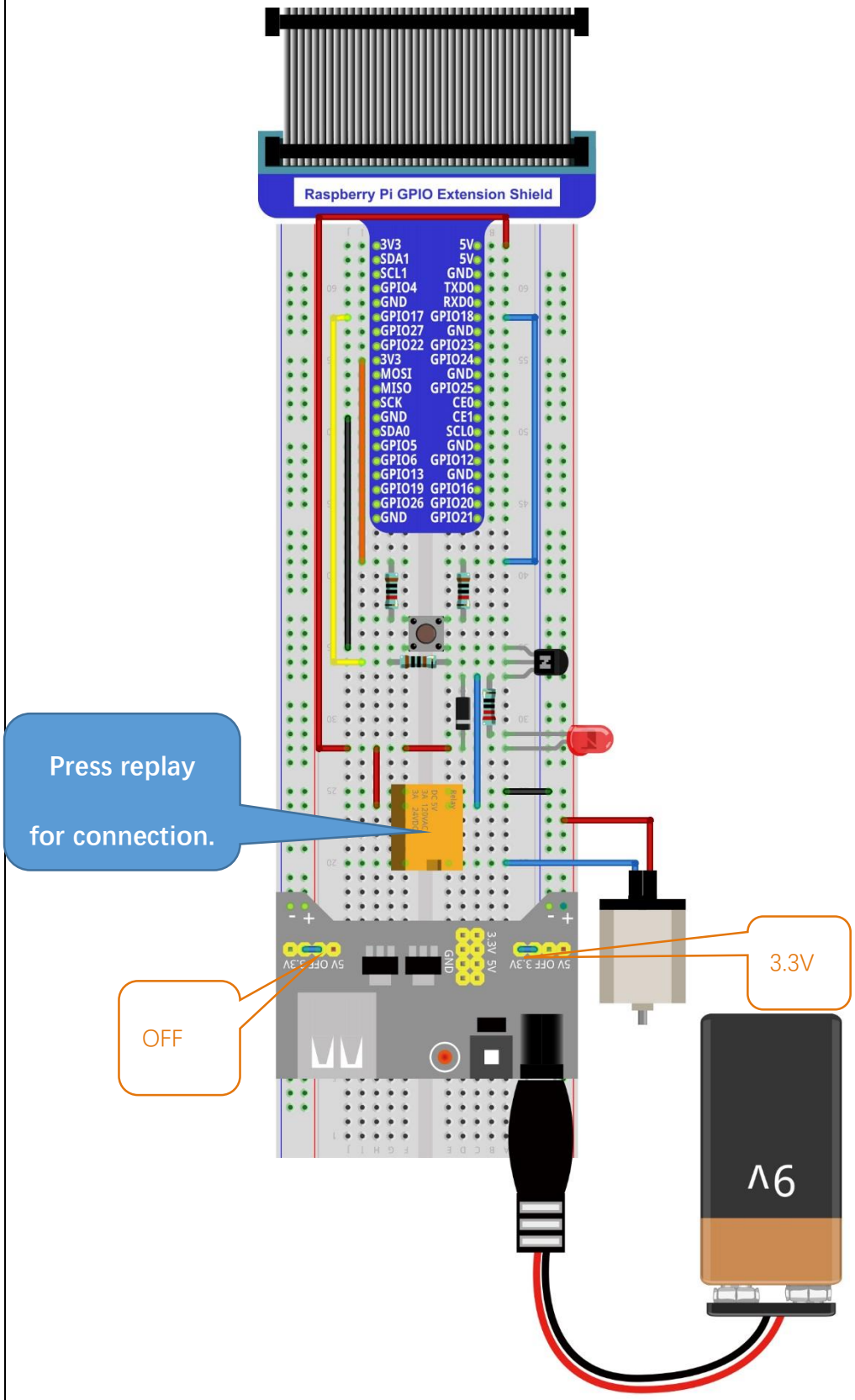
Circuit

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.

Schematic diagram



Hardware connection. If you need any support, please feel 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
point
```

```
24                  lastChangeTime = millis();
25              }
26              //if changing-state of the button last beyond the time we set, we considered that
27              //the current button state is an effective change rather than a buffeting
28              if(millis() - lastChangeTime > captureTime){
29                  //if button state is changed, update the data.
30                  if(reading != buttonState){
31                      buttonState = reading;
32                      //if the state is low, the action is pressing.
33                      if(buttonState == LOW){
34                          printf("Button is pressed!\n");
35                          relayState = !relayState;
36                          if(relayState){
37                              printf("turn on relay ...\n");
38                          }
39                          else {
40                              printf("turn off relay ...\n");
41                          }
42                      }
43                      //if the state is high, the action is releasing.
44                      else {
45                          printf("Button is released!\n");
46                      }
47                  }
48              }
49              digitalWrite(relayPin,relayState);
50              lastbuttonState = reading;
51          }
52
53          return 0;
54      }
```

The project code is in the same as we used earlier in the Table Lamp project.

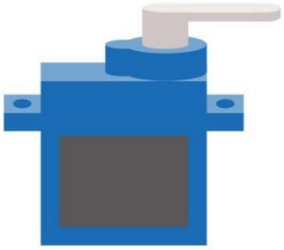
Chapter 15 Servo

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.

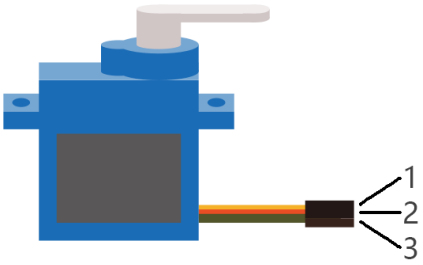
Component List

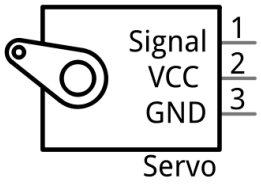
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x3 
Servo x1 	

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.

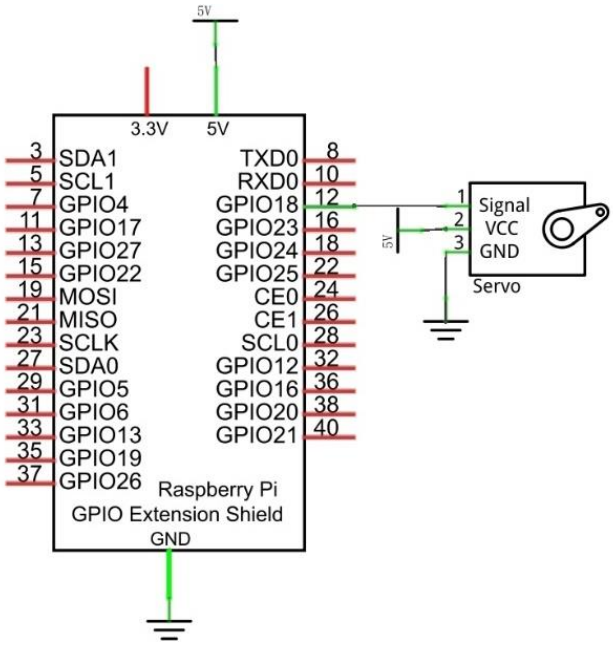
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

When you change the Servo signal value, the Servo will rotate to the designated angle.

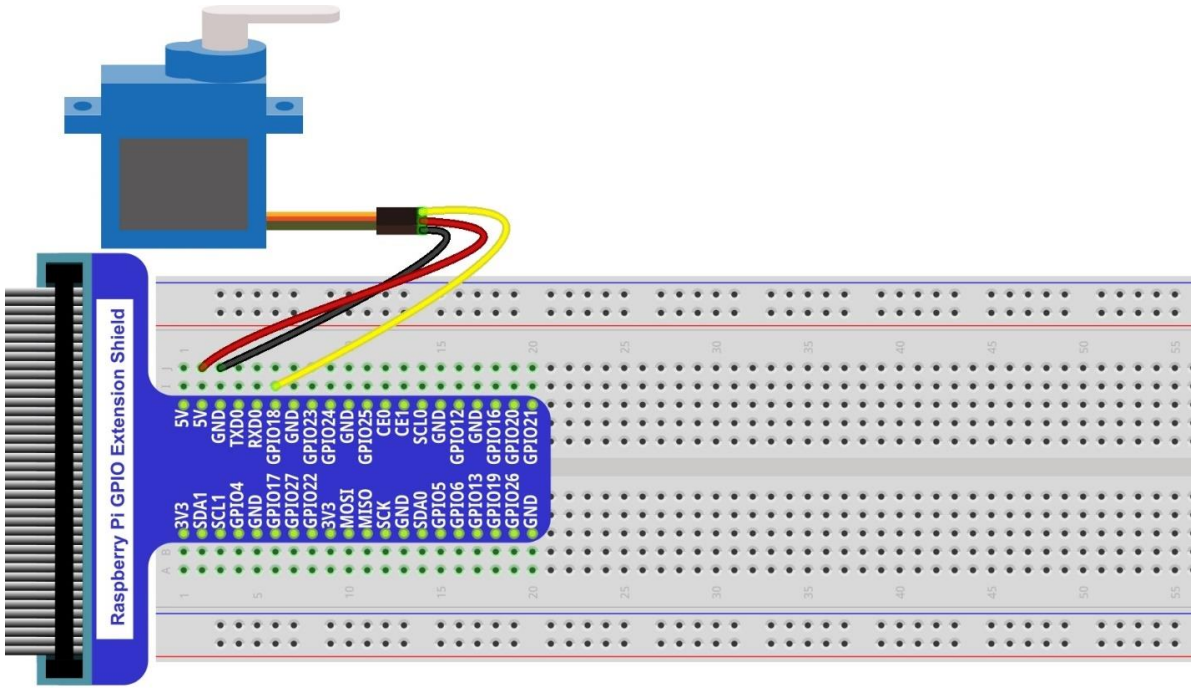
Circuit

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

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:

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

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

```
26          ms = SERVO_MIN_MS;
27      softPwmWrite(pin,ms);
28  }
29
30  int main(void)
31  {
32      int i;
33
34      printf("Program is starting ...\n");
35
36      wiringPiSetup();
37      servoInit(servoPin);        //initialize PMW pin of servo
38      while(1){
39          for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle to
    maximum angle
40              servoWriteMS(servoPin,i);
41              delay(10);
42          }
43          delay(500);
44          for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle to
    minimum angle
45              servoWriteMS(servoPin,i);
46              delay(10);
47          }
48          delay(500);
49      }
50      return 0;
51  }
```

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

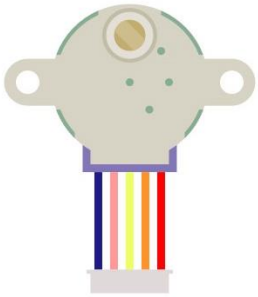
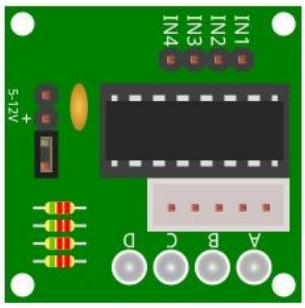
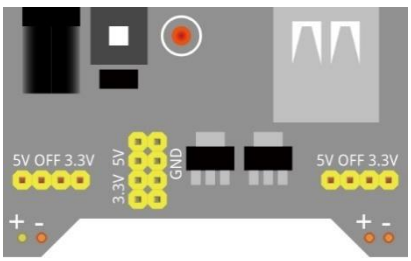
Chapter 16 Stepper Motor

Thus far, we have learned about DC Motors and Servos. A DC motor can rotate constantly in on 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.

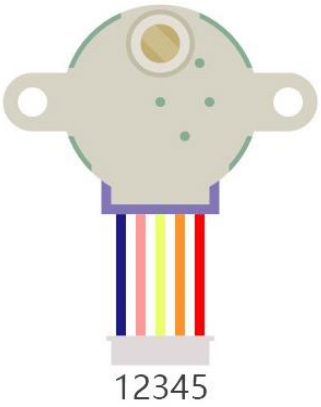
Component List

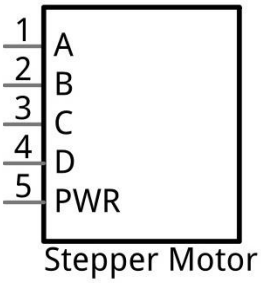
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Jumper Wire x12 
Stepper Motor x1 	ULN2003 Stepper Motor Driver x1 
9V battery (prepared by yourself) & battery line 	Breadboard Power module x1 

Component knowledge

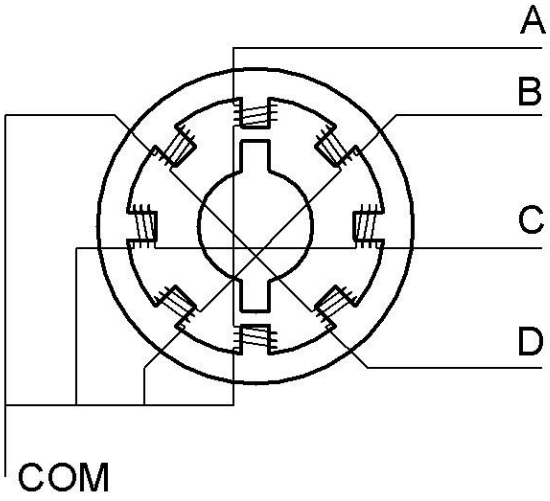
Stepper Motor

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:



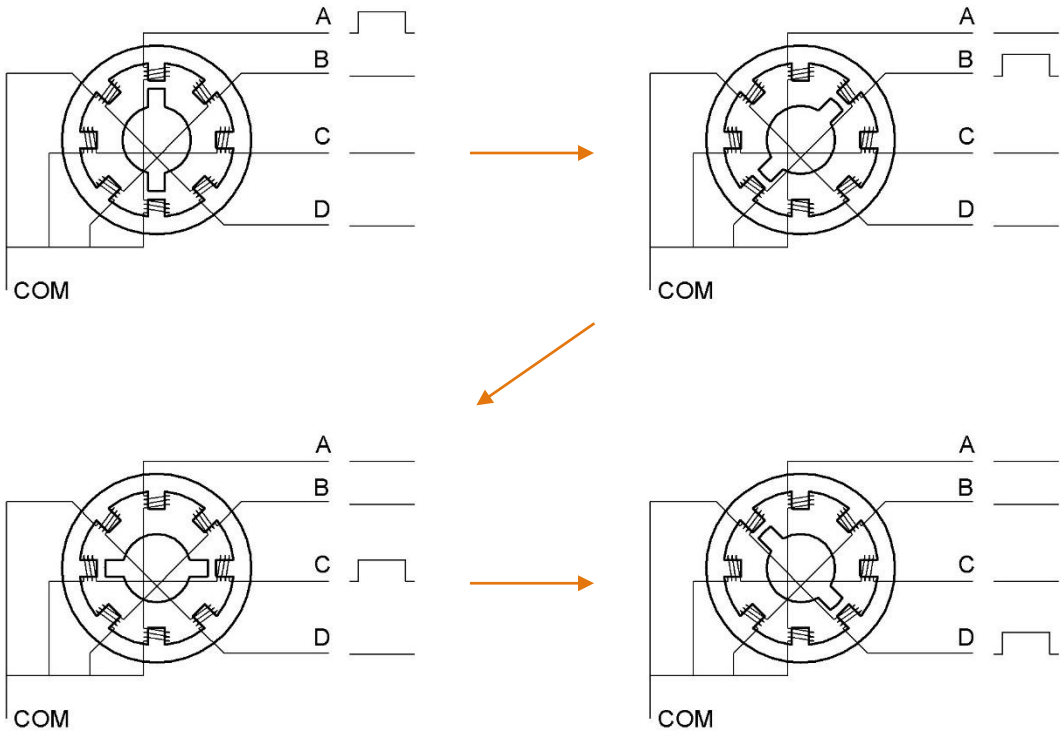


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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:



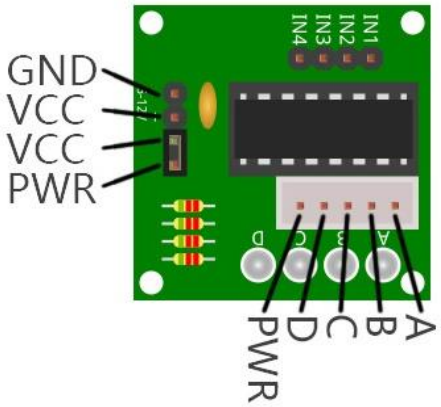
In the sequence above, the Stepper Motor rotates by a certain angle at once, which is called a “step”. By controlling the number of rotational steps, you can then control the Stepper Motor’s rotation angle. By defining the time between two steps, you can control the Stepper Motor’s rotation speed. When 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 is powered ON in the reverse order, D → C → B → A → D →··· , the rotor will rotate in counter-clockwise direction.

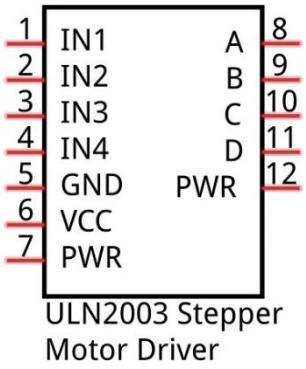
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. Tise sequence of powering the coils looks like this: A → AB → B → BC → C → CD → D → DA → A →·····, the rotor will rotate in accordance to this sequence ar, 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 X 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.

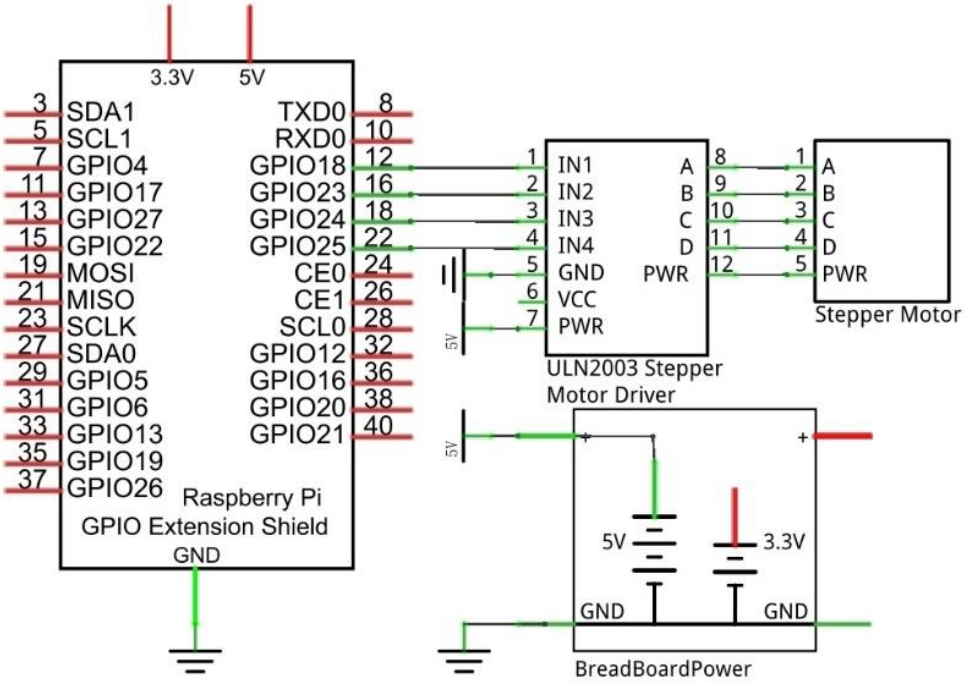




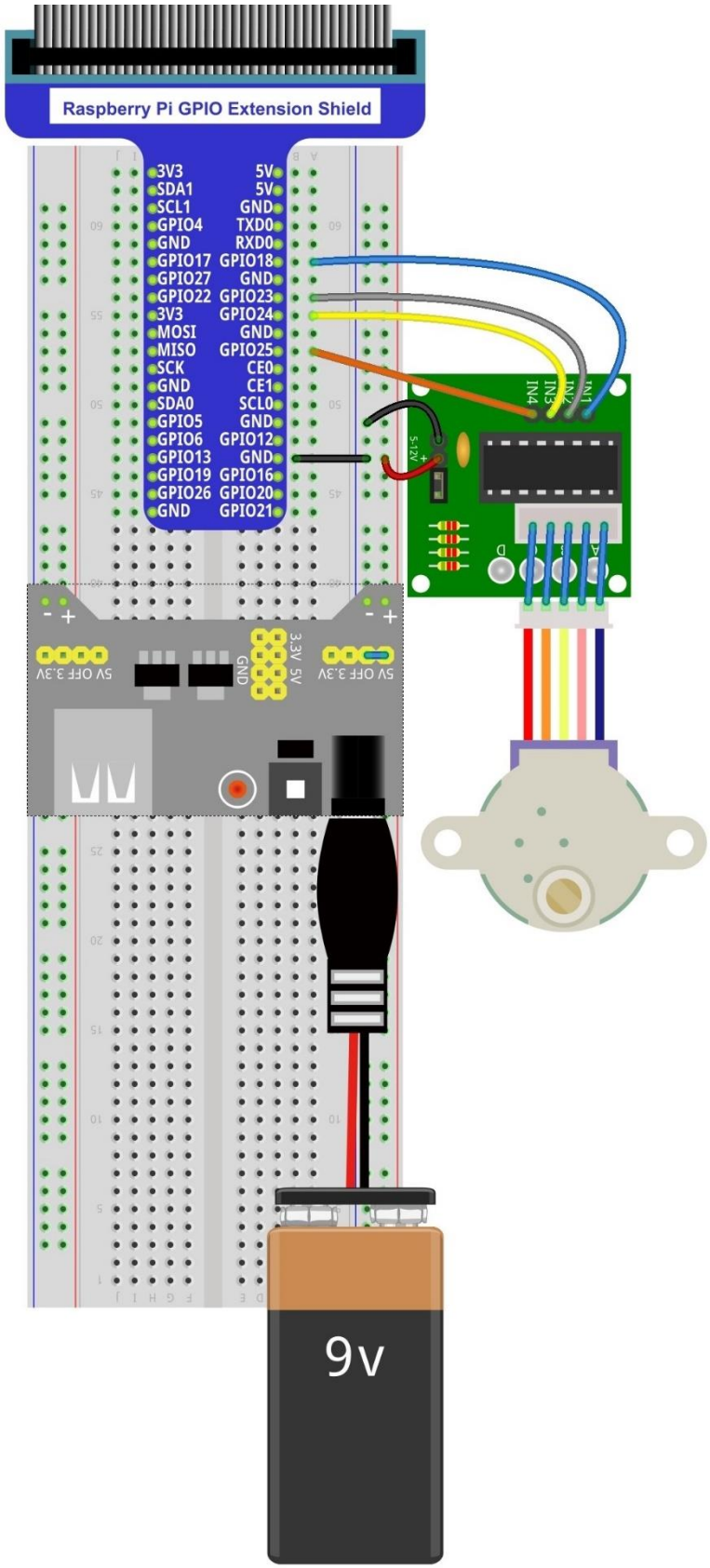
Circuit

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:

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

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

```
    steps is a cycle
25  void moveSteps(int dir, int ms, int steps){
26      int i;
27      for(i=0;i<steps;i++){
28          moveOnePeriod(dir,ms);
29      }
30  }
31  void motorStop(){   //function used to stop rotating
32      int i;
33      for(i=0;i<4;i++){
34          digitalWrite(motorPins[i],LOW);
35      }
36  }
37  int main(void){
38      int i;
39
40      printf("Program is starting ...\n");
41
42      wiringPiSetup();
43
44      for(i=0;i<4;i++){
45          pinMode(motorPins[i],OUTPUT);
46      }
47
48      while(1){
49          moveSteps(1,3,512);     //rotating 360°  clockwise, a total of 2048 steps in a circle,
    namely, 512 cycles.
50          delay(500);
51          moveSteps(0,3,512);     //rotating 360°  anticlockwise
52          delay(500);
53      }
54      return 0;
55  }
```

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);
    }
```

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.

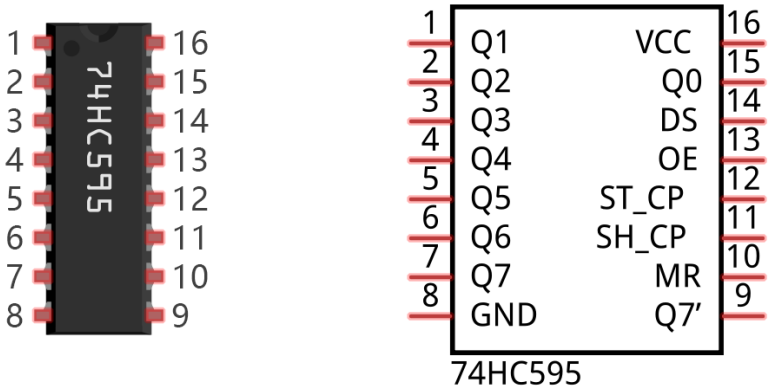
Component List

<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper x17
  </td></tr>
<tr><td>74HC595 x1
  </td><td>Bar Graph LED x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

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.



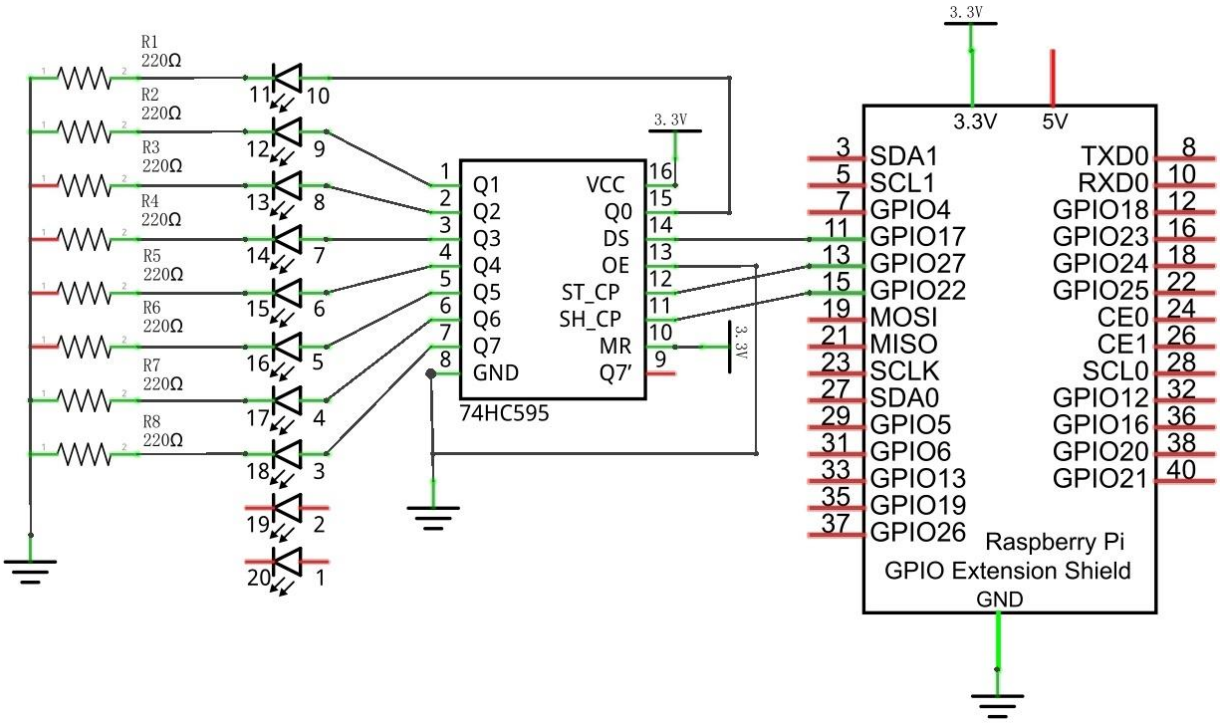
The ports of the 74HC595 chip are described as follows:

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.

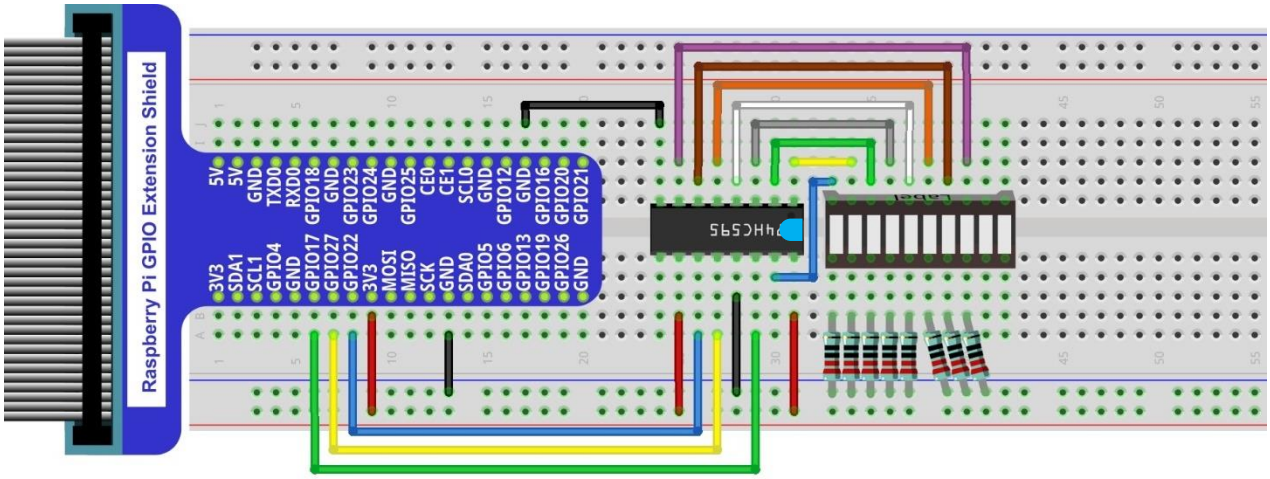
For more details, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel 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:

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

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

void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}

int main(void)
{
    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
    update the data to the parallel output port.
44              x<<=1;      //make the variable move one bit to left once, then the bright LED
    move one step to the left once.
45              delay(100);
46          }
47          x=0x80;
48          for(i=0;i<8;i++){
49              digitalWrite(latchPin,LOW);
50              _shiftOut(dataPin,clockPin,LSBFIRST,x);
51              digitalWrite(latchPin,HIGH);
52              x>>=1;
53              delay(100);
54          }
55      }
56      return 0;
57  }
```

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.
```

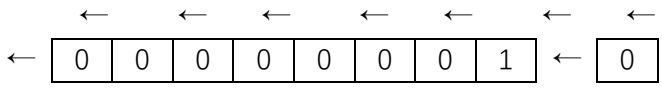
```
            x<<=1; // make the variable move one bit to left once, then the bright LED move
one step to the left once.
            delay(100);
        }
```

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

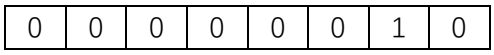
<< operator

"<<" 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;

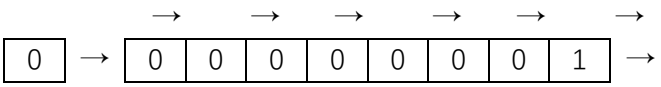


The result of x is 2（binary 00000010）.

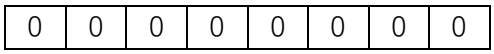


There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

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: https://github.com/WiringPi/WiringPi

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".

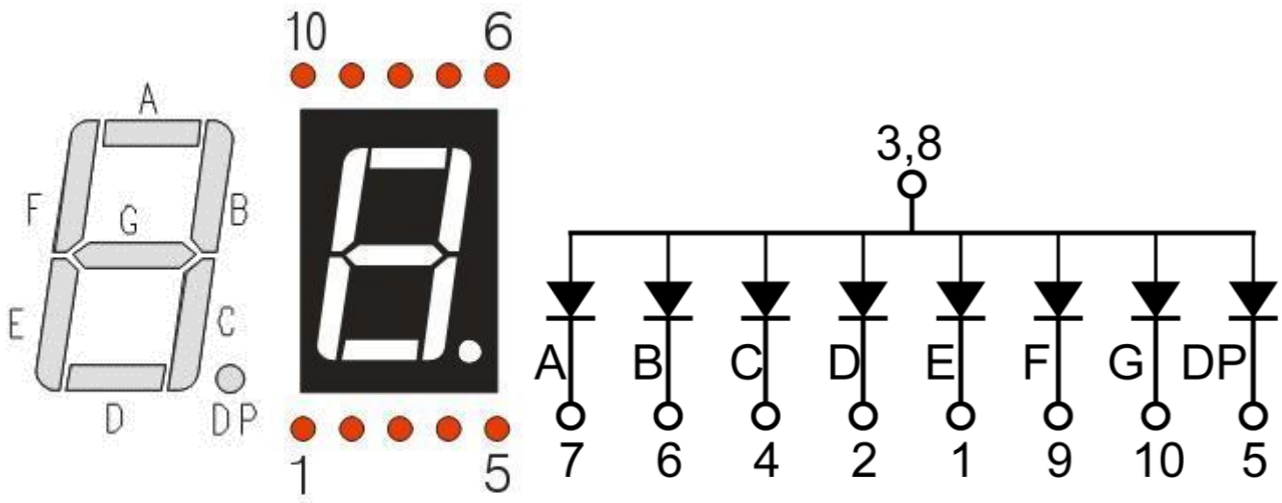
Component List

<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper Wire x18
  </td></tr>
<tr><td>74HC595 x1
  </td><td>7-Segment Display x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Component knowledge

7-segment display

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:

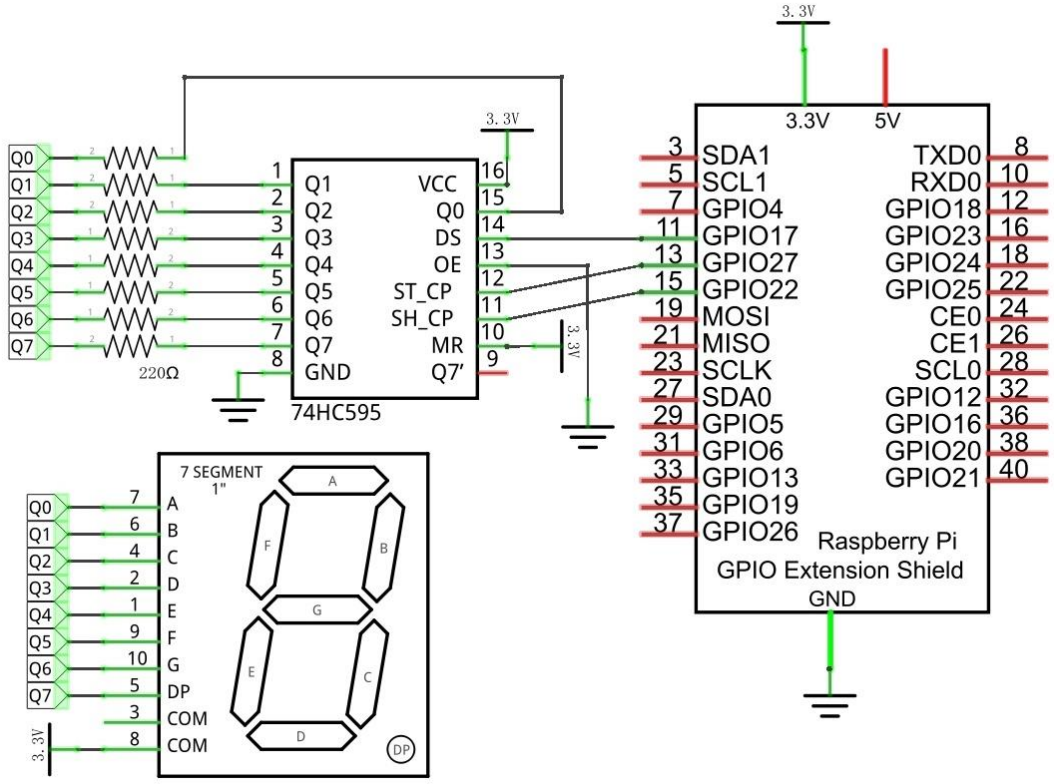


As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a “0”: we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.

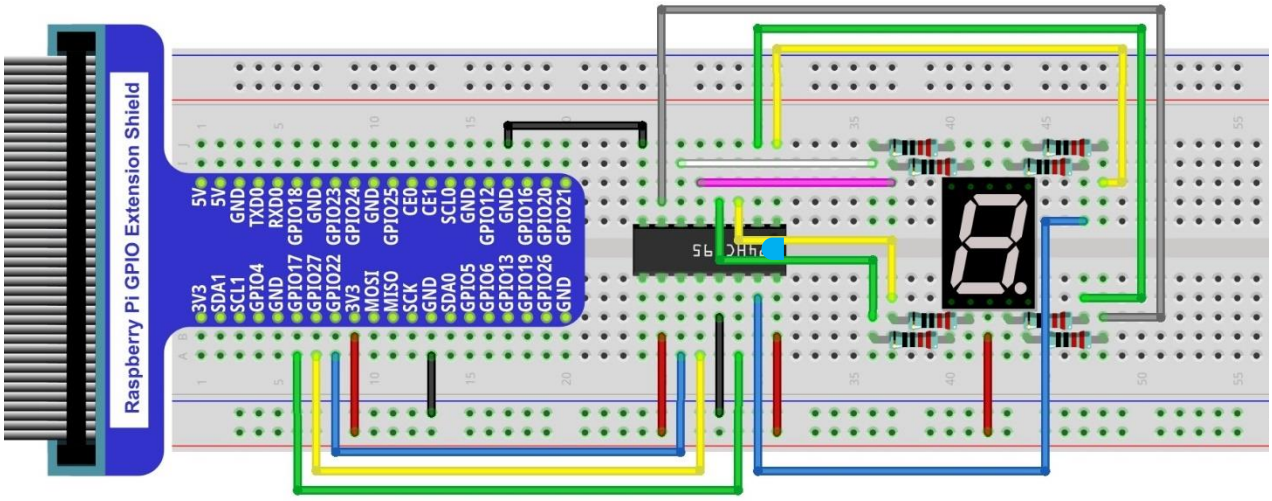
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”, “B”, “A”. Character "0" corresponds to the code: 1100 0000b=0xc0.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel 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:

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

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

void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
```

```
}

int main(void)
{
    int i;

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

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            _shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            _shiftOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);//Use the "&0x7f" to display the decimal point.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
    }
    return 0;
}
```

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 transfered 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);
```

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.

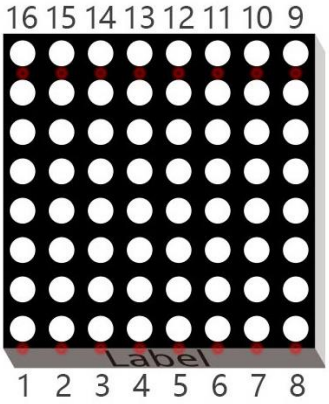
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper x36 	
74HC595 x2 	8X8 LEDMatrix x1 	Resistor 220Ω x8 

Component knowledge

LED matrix

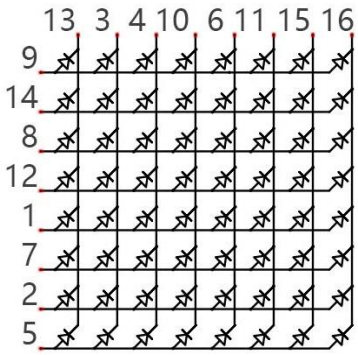
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



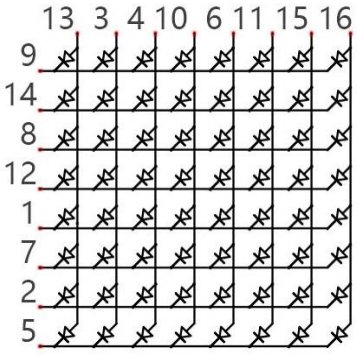
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another 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.

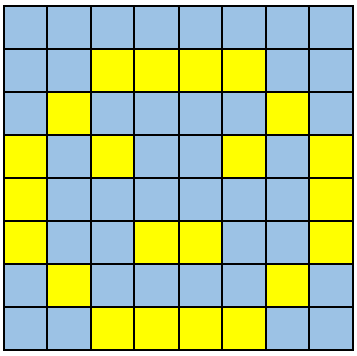
Connection mode of Common Anode



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

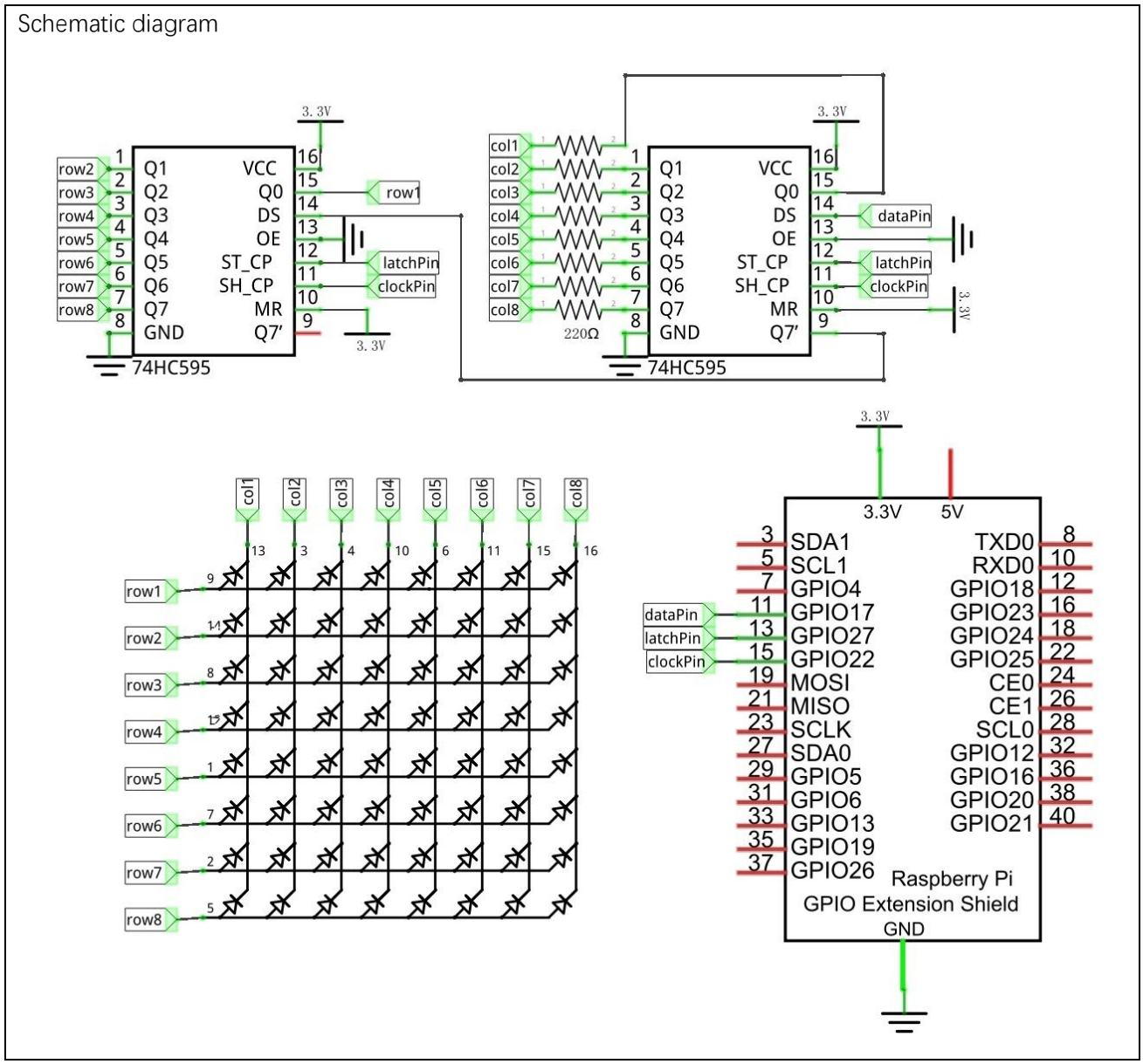
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated 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.

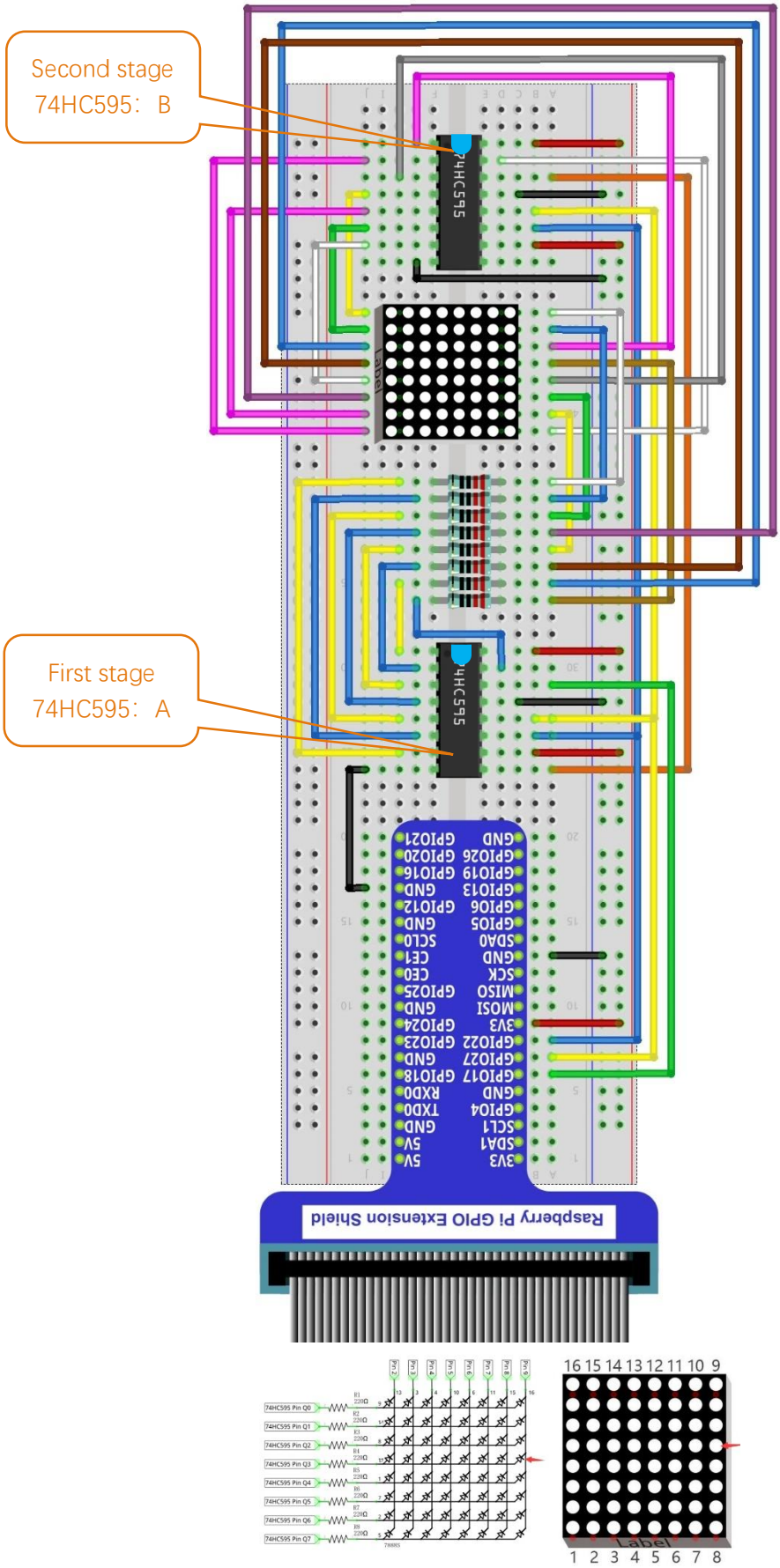
Circuit

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:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)
// data of smile face
unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
unsigned char data[]={  // data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
```

```
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
};
void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}
int main(void)
{
    int i,j,k;
    unsigned char x;

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

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(j=0;j<500;j++){  //Repeat enough times to display the smiling face a period of time
            x=0x80;
            for(i=0;i<8;i++){
                digitalWrite(latchPin,LOW);
                _shiftOut(dataPin,clockPin,MSBFIRST,pic[i]);// first shift data of line information to the first stage 74HC959
                _shiftOut(dataPin,clockPin,MSBFIRST,~x);//then shift data of column information to the second stage 74HC959

                digitalWrite(latchPin,HIGH);//Output data of two stage 74HC595 at the same
```

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

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=0;j<500;j++){// Repeat enough times to display the smiling face a period of time
              x=0x80;
              for(i=0;i<8;i++){
                  digitalWrite(latchPin,LOW);
                  shiftOut(dataPin,clockPin,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 X 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.

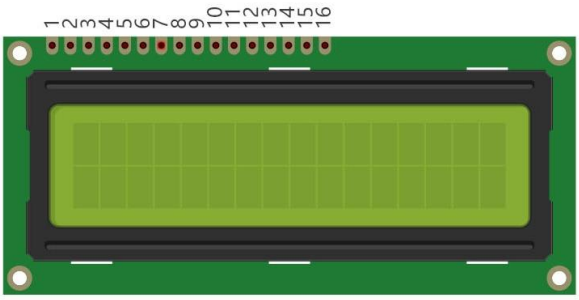
```
        for(k=0;k<sizeof(data)-8;k++){  //sizeof(data) total number of "0-F" columns
            for(j=0;j<20;j++){// times of repeated displaying LEDMatrix in every frame, the
bigger the “j”, the longer the display time
                x=0x80;      // Set the column information to start from the 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);
                }
            }
        }
```

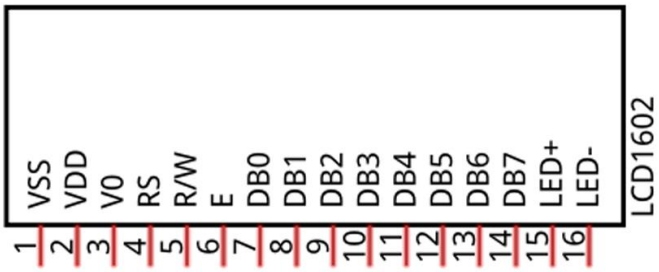
Chapter 20 LCD1602

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

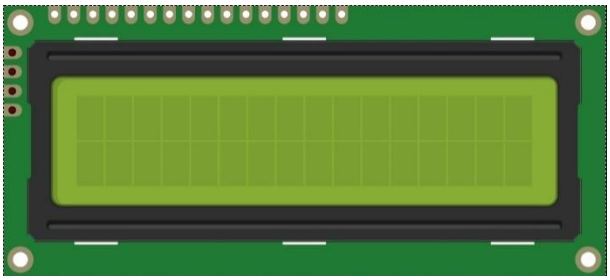
Project 20.1 I2C LCD1602

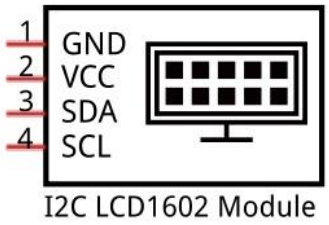
There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII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

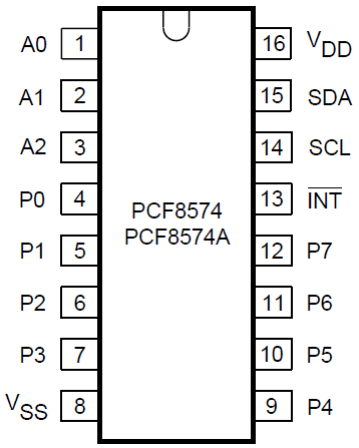




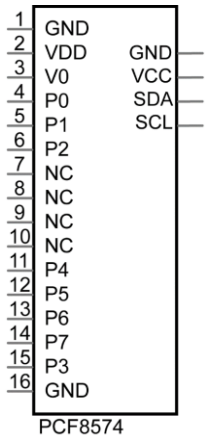
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the RPI bus on your I2C device address through command "i2cdetect -y 1" (refer to the "configuration I2C" section below).

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

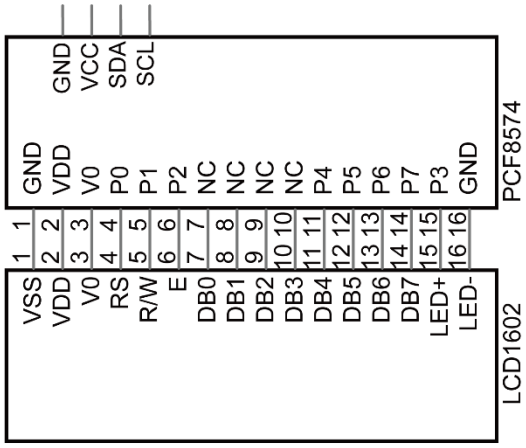
PCF8574 chip pin diagram:



PCF8574 module pin diagram



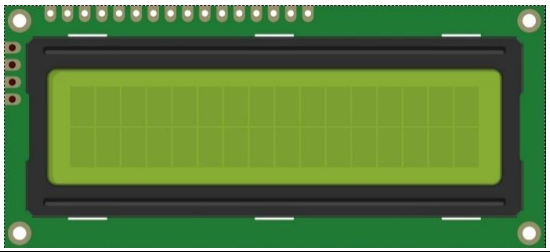
PCF8574 module pins and LCD1602 pins correspond to each other and connected to each other:



Because of this, as stated earlier, we only need 4 pins to control the16 pins of the LCD1602 Display Screen through the I2C interface.

In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

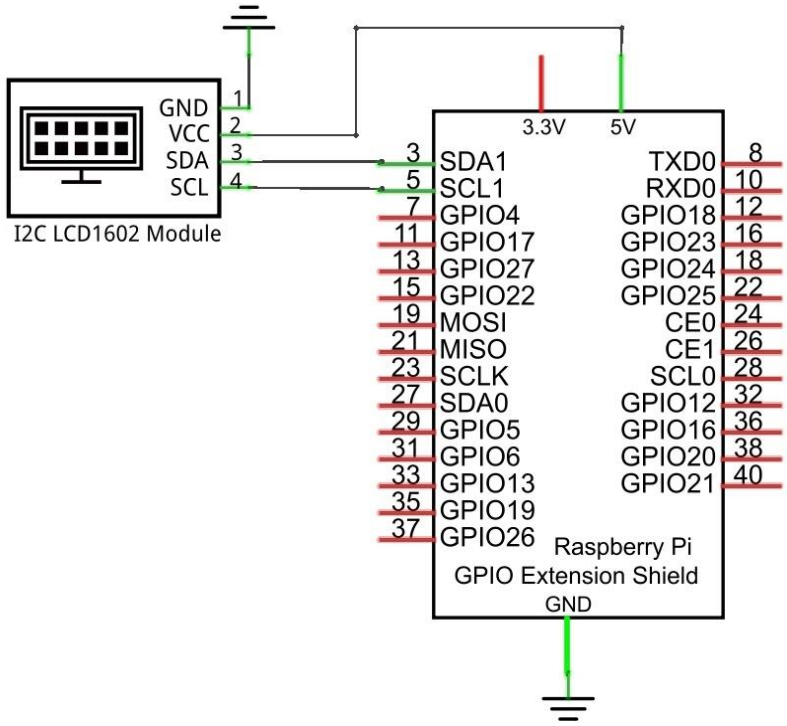
Component List

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

Circuit

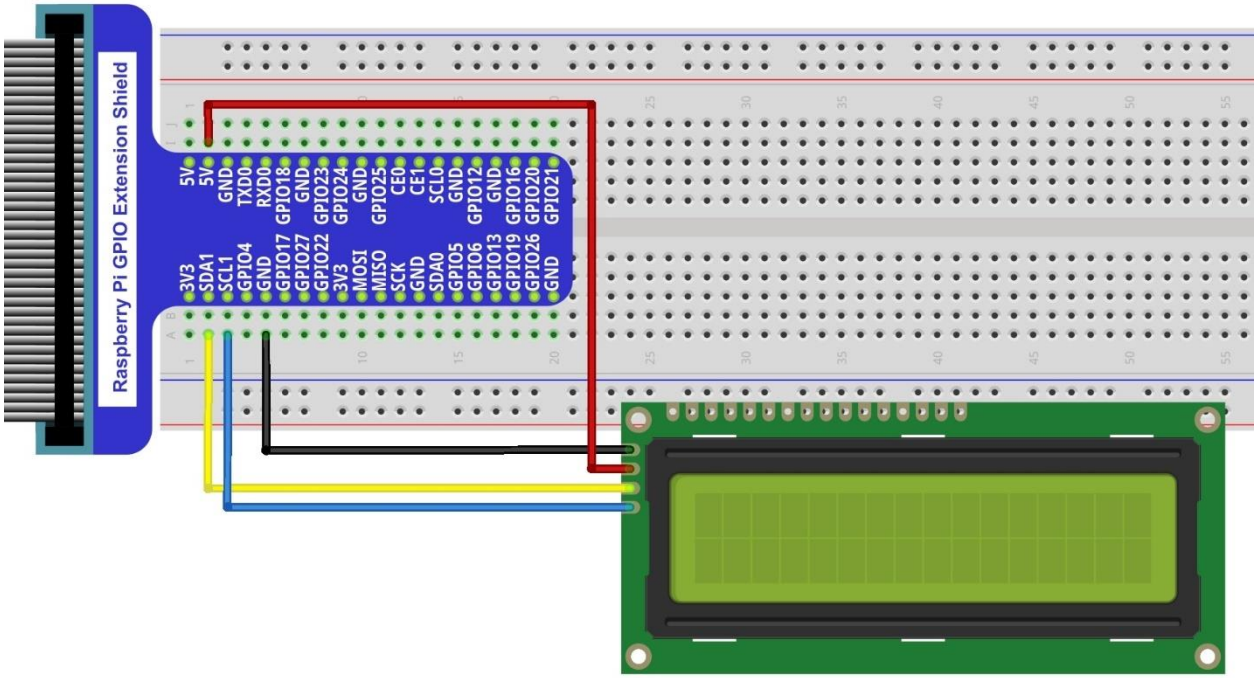
Note that the power supply for I2C LCD1602 in this circuit is 5V.

Schematic diagram



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

NOTE: It is necessary to configure 12C and install Smbus first (see chapter 7 for details)



Code

This code will have your RPi's CPU temperature and System Time Displayed on the LCD1602.

C Code 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
```

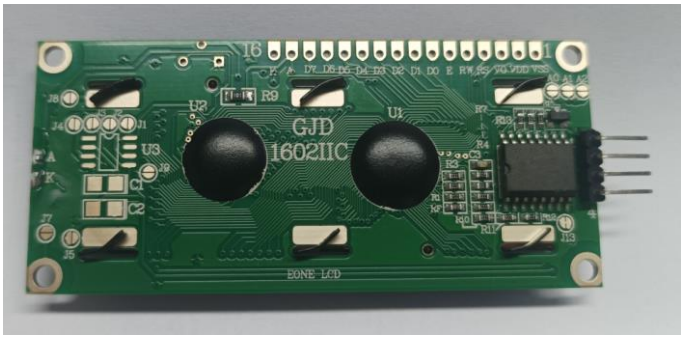
2. Use following command to compile "I2CLCD1602.c" and generate executable file "I2CLCD1602".

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

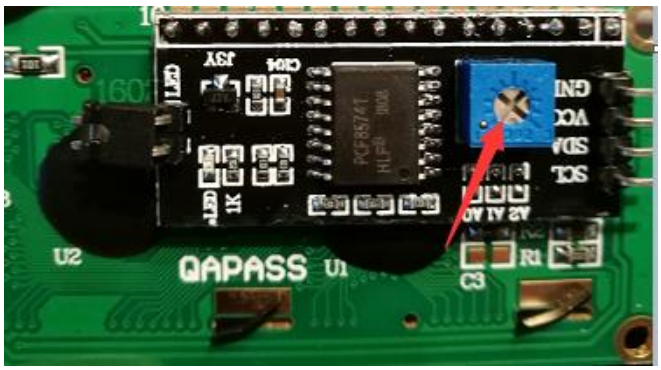
3. Then run the generated file "I2CLCD1602".

```
sudo ./I2CLCD1602
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.
So far, at this writing, we have two types of LCD1602 on sale. One needs to adjust the backlight, and the other does not.
The LCD1602 that does not need to adjust the backlight is shown in the figure below.



If the LCD1602 you received is the following one, and 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 clearly.



The following is the program code:

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

```
#include <lcd.h>
#include <time.h>

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

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

lcdPrintf(lcdhd,"Time:%02d:%02d:%02d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
int detectI2C(int addr){  //Used to detect i2c address of LCD
    int _fd = wiringPiI2CSetup (addr);
    if (_fd < 0){
        printf("Error address : 0x%x \n",addr);
```

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

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

First, define the I2C address of the PCF8574 and the Extension of the GPIO pin, which is connected to the GPIO pin of the LCD1602. LCD1602 has two different i2c addresses. Set 0x27 as default.

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

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

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

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

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

Details about lcdInit():

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

This is the main initialization function and must be executd first before you use any other LCD functions. **Rows** and **cols** are the rows and columns of the Display (e.g. 2, 16 or 4, 20). **Bits** is the number of how wide the number of bits is on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the Display's RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the RPi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault (usually incorrect parameter)

For more details about LCD Library, please refer to: https://projects.drogon.net/raspberry-pi/wiringpi/lcd-library/

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

temperature value stored in variable CPU_temp and uses lcdPrintf() to display it on LCD.

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


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

Details about lcdPosition() and lcdPrintf():

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

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

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

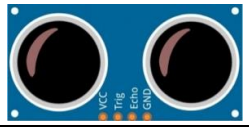
Chapter 21 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance.

Project 21.1 Ultrasonic Ranging

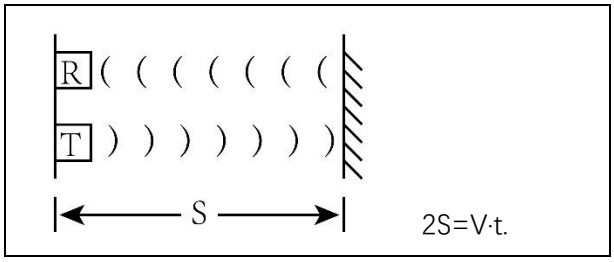
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

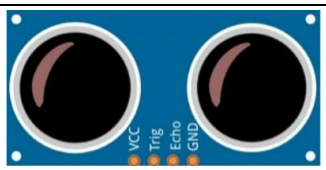
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	Ultrasonic Module x1 
Jumper Wire x4 	Resistor 1kΩ x3 

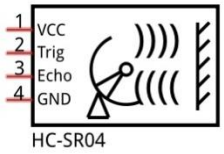
Component Knowledge

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=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



The Ultrasonic Ranging Module integrates a 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 Ultrasonic Ranging Module are shown below:





Pin description:

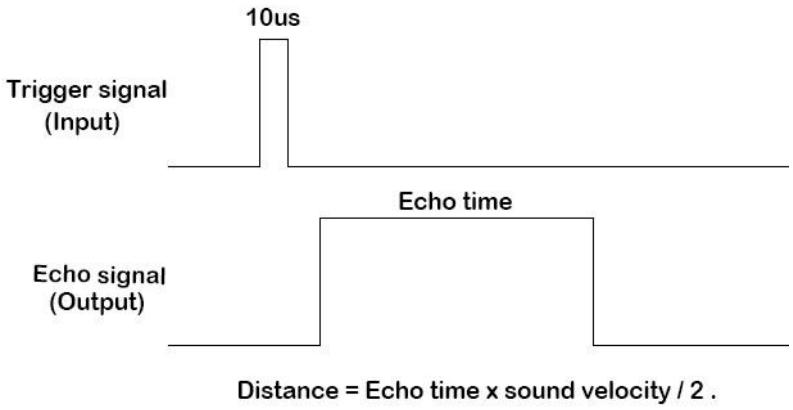
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

Minimum measured distance: 2cm Maximum measured distance: 200cm

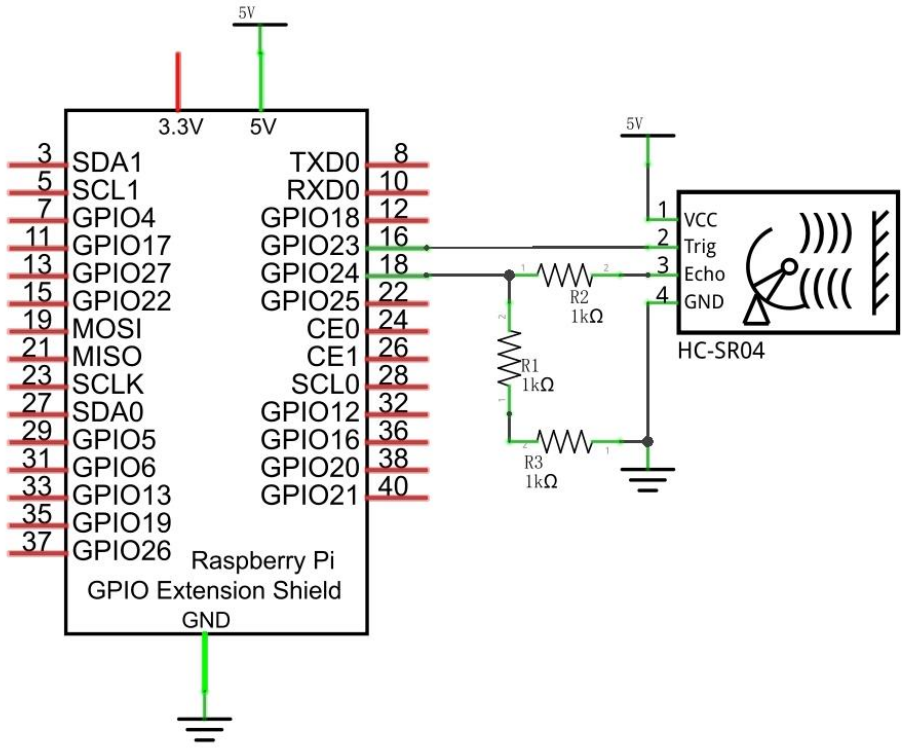
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The 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.



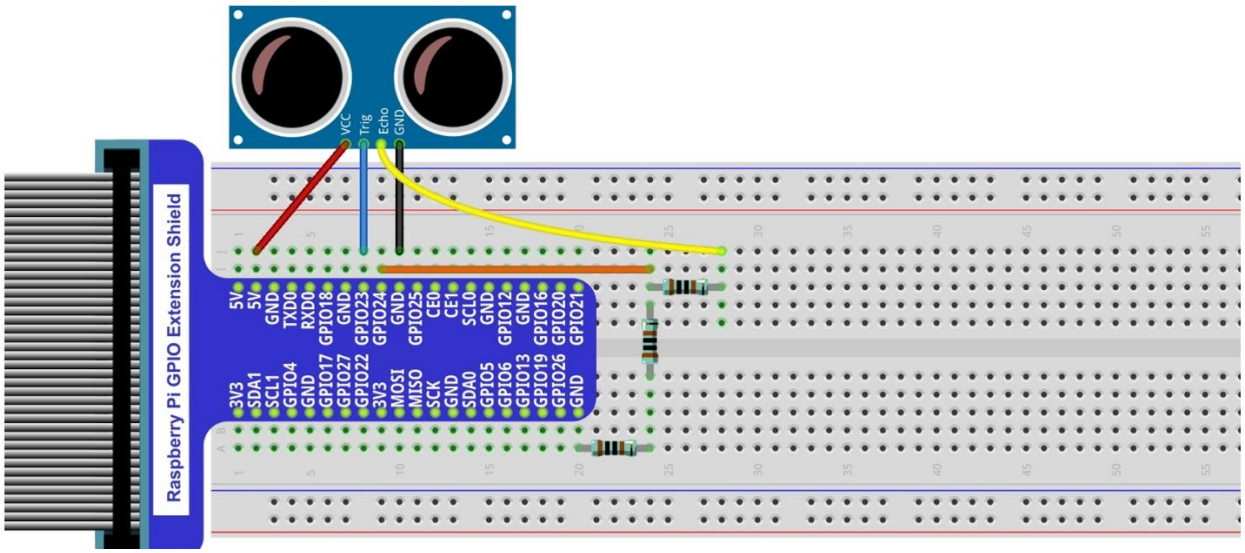
Circuit

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 21.1.1 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 21.1.1_UltrasonicRanging directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/21.1.1_UltrasonicRanging
```

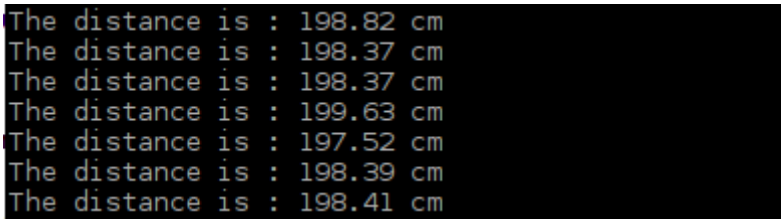
2. Use following command to compile "UltrasonicRanging.c" and generate executable file "UltrasonicRanging".

```
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 following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220        // define the maximum measured distance
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured distance
//function pulseIn: obtain pulse time of a pin
int pulseIn(int pin, int level, int timeout);
float getSonar(){   //get the measurement result of ultrasonic module with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //send 10us high level to trigPin
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; //calculate distance with sound speed 340m/s
    return distance;
```

```
}

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

    wiringPiSetup();

    float distance = 0;
    pinMode(trigPin,OUTPUT);
    pinMode(echoPin,INPUT);
    while(1){
        distance = getSonar();
        printf("The distance is : %.2f cm\n",distance);
        delay(1000);
    }
    return 1;
}
```

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).

Other Components

This kit also includes other common components that can help your ideas come true. Now we will introduce components not mentioned in the previous section.

Component Knowledge

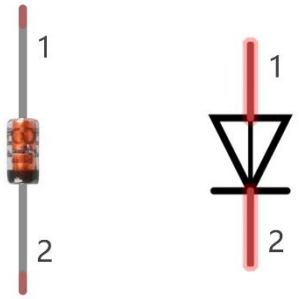
Toggle switch

Like push button switch, toggle switch is also a kind of switching devices. The difference is that toggle switch is suitable for long-time open or close circuits.

When the lever is moved to the left, pin 1, 2 get conducted, and pin 2, 3 are disconnected from each other;
When the lever is moved to the right, pin 2,3 get conducted, and pin 1,2 are disconnected from each other;

Switch diode

There are several types of diodes. We have used 1N4001 before, which is a common rectifier diode and commonly used in ac rectifier.
Here is 1N4148, which is a kind of high-speed switching diodes and is characterized by a relatively rapid switching.



For the switching diode, the changing time from conduction to cut off or from cut off to conduction is shorter than general diode and it is mainly used in electronic computer, pulse and switching circuits.

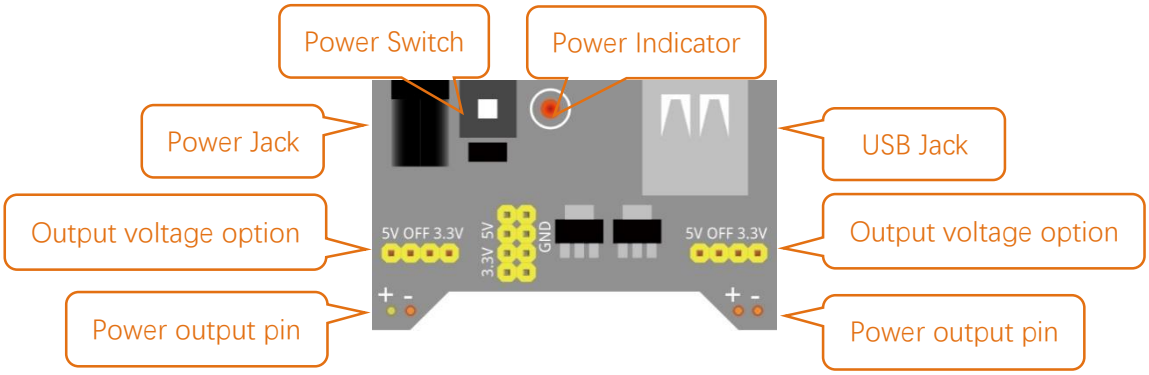
9V battery cable

A 9V battery cable can connect a 9 V battery, which can supply power for control board.

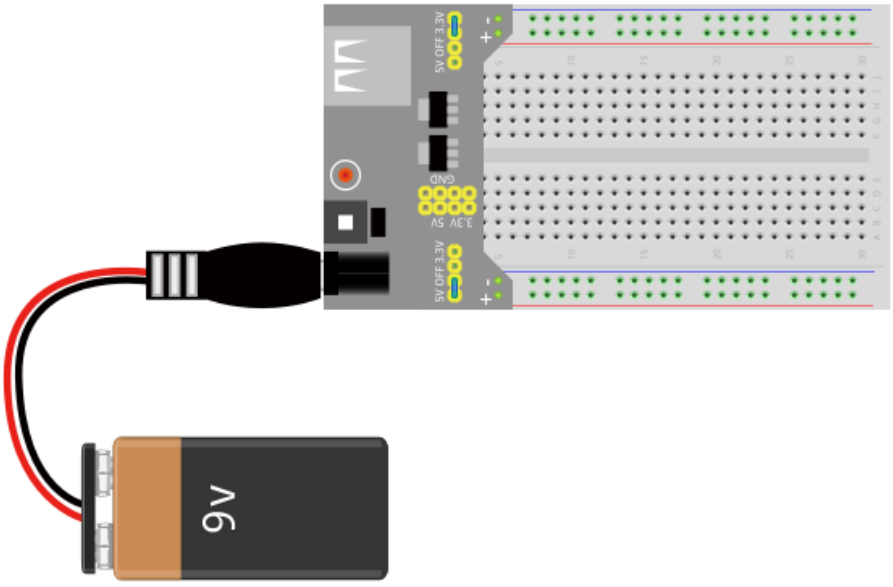
The installation of 9V battery cable is as follows:

Power supply module for breadboard

The following is the power supply module for breadboard. This module can provide the breadboard with two-channel power supply separately, and each can be configured to 3.3 V or 5 V separately through a jumper.



We can build a circuit conveniently by using this module. You only need to provide power supply for this module, and then insert it on the breadboard.



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.