Getting Started

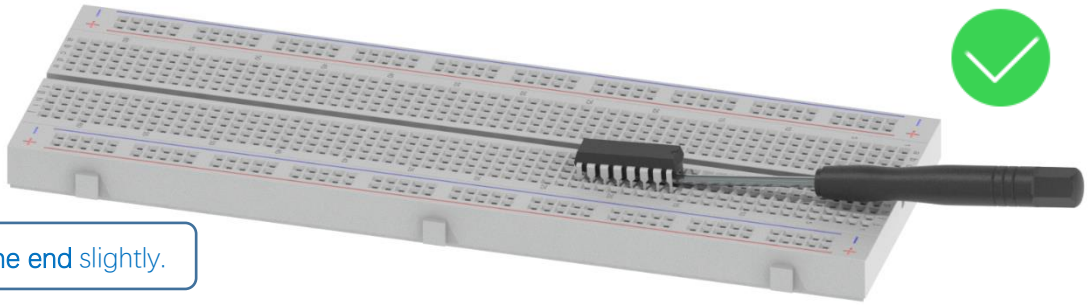
Thank you for choosing Freenove products!
After you download the ZIP file we provide. Unzip it and you will get a folder contains several files and folders. There are three PDF files:

- 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_Super_Starter_Kit_for_Raspberry_Pi\Processing
  The code in this PDF is in Java.
- Pi4j.pdf in Freenove_Super_Starter_Kit_for_Raspberry_Pi\Pi4j
  The code in this PDF is in Java.

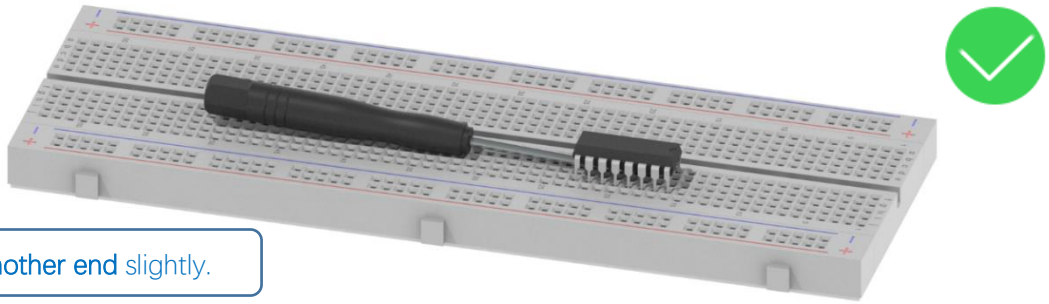
We recommend you to start with Tutorial.pdf or Tutorial_GPIOZero.pdf first.
If you want to start with Processing.pdf or skip some chapters of Tutorial.pdf, you need to finish necessary steps in Chapter 7 AD/DA of Tutorial.pdf first.

Remove the Chips

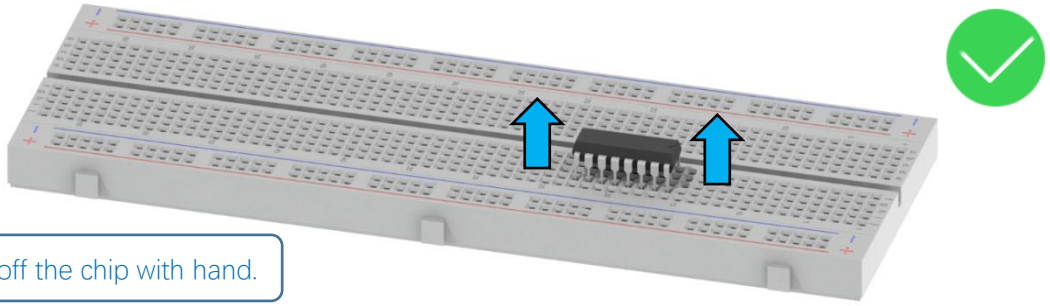
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



Step 1, lift one end slightly.

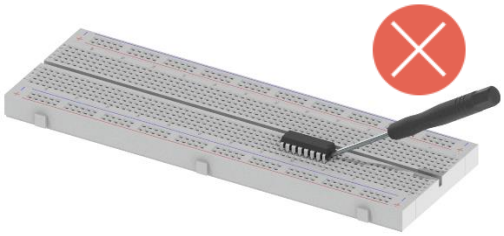


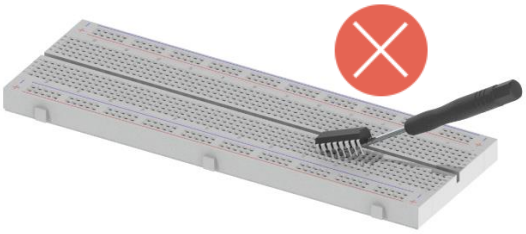
Step 2, lift another end slightly.



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:

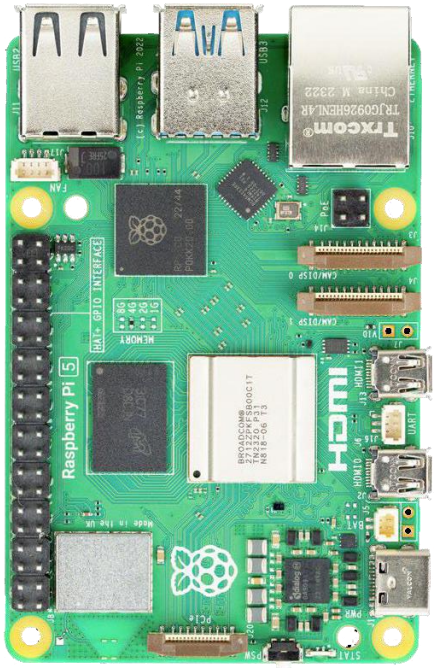
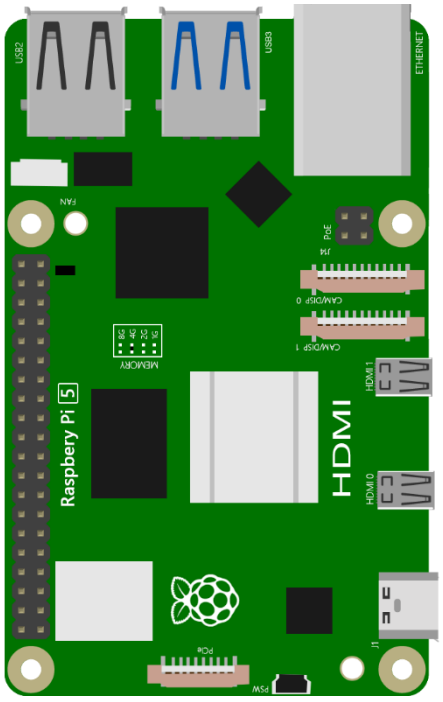
support@freenove.com

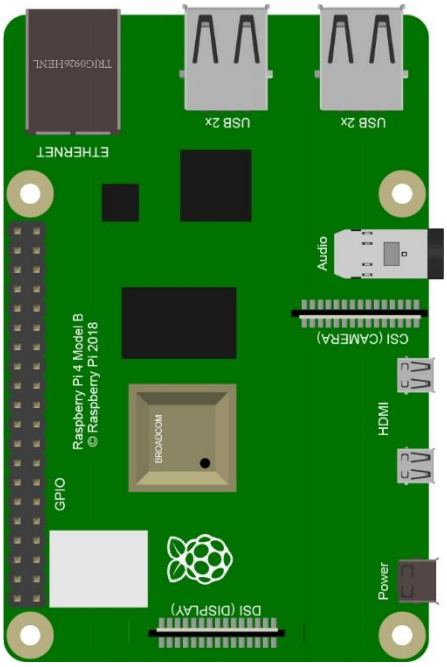
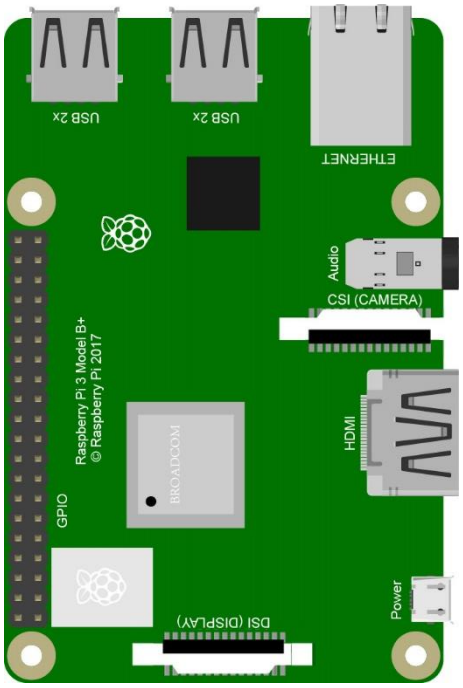
Raspberry Pi

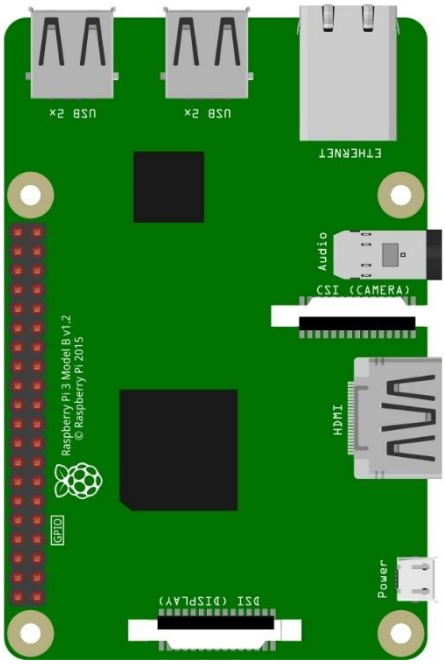
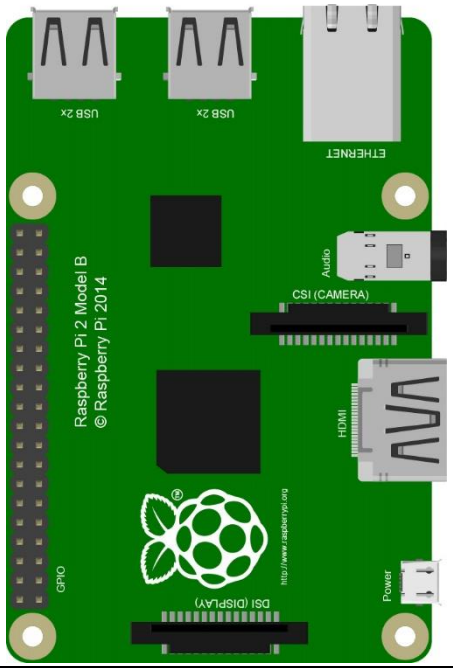
So far, at this writing, Raspberry Pi has advanced to its fifth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

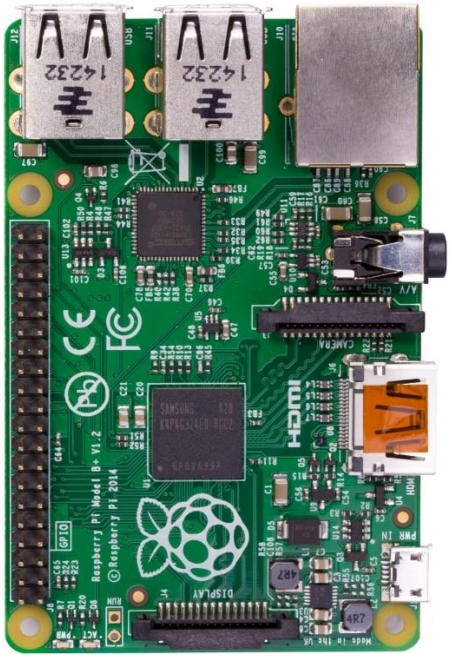
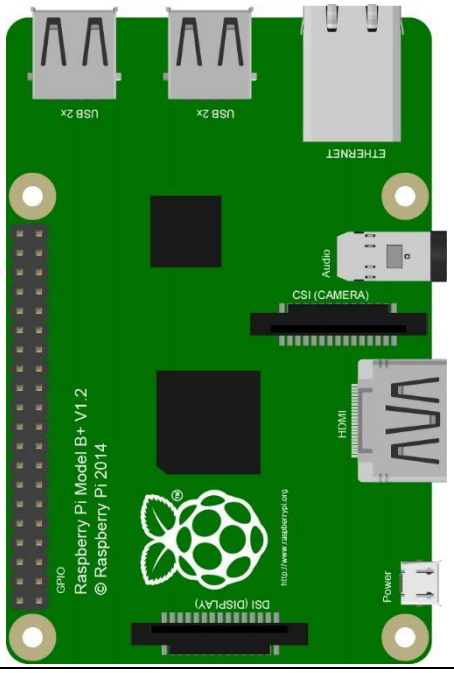
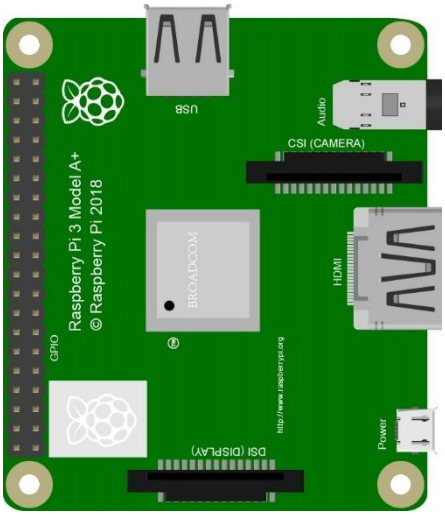
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility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

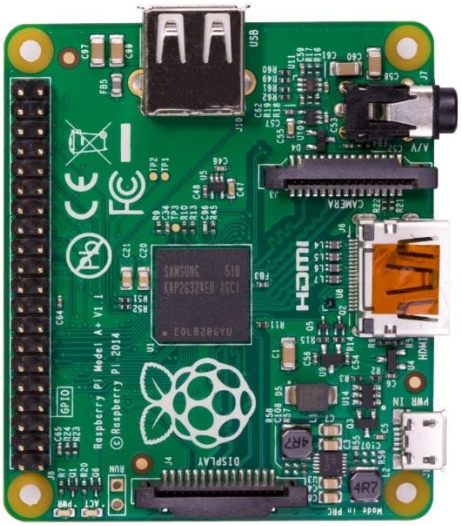
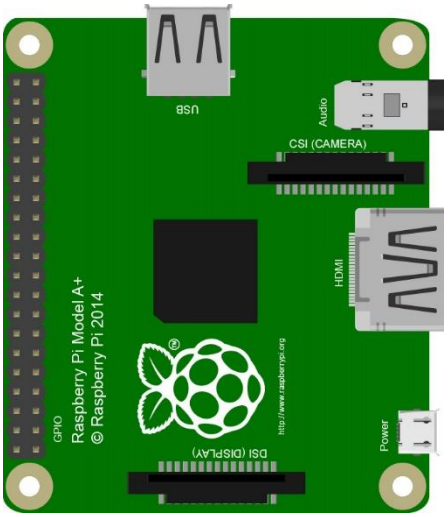
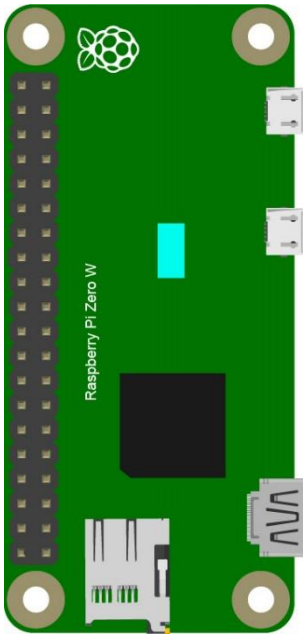
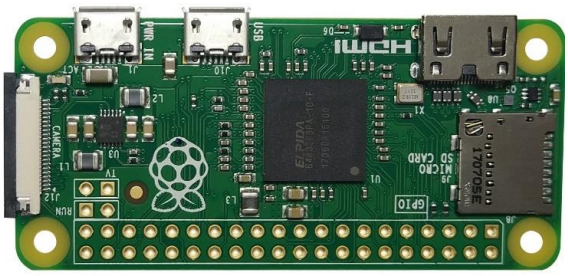
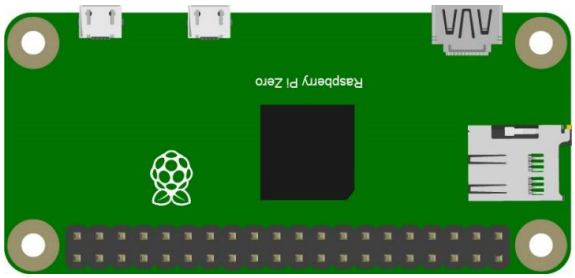
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

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

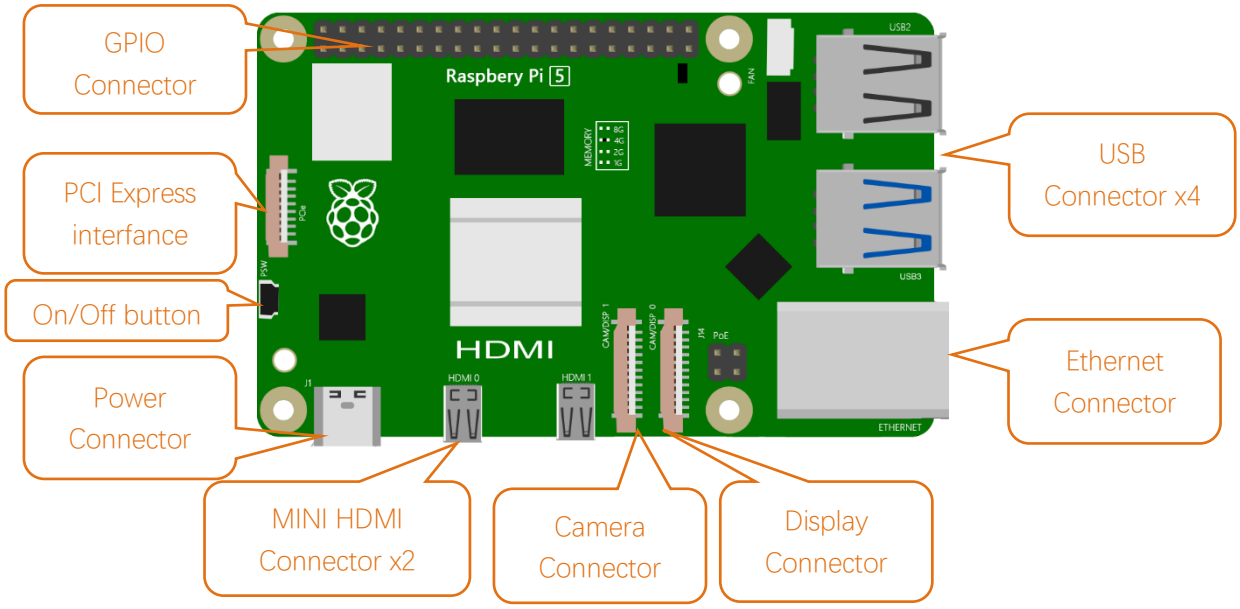
Actual image of Raspberry Pi 4 Model B: 	CAD image of Raspberry Pi 4 Model B: 
Actual image of Raspberry Pi 3 Model B+: 	CAD image of Raspberry Pi 3 Model B+: 

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: 

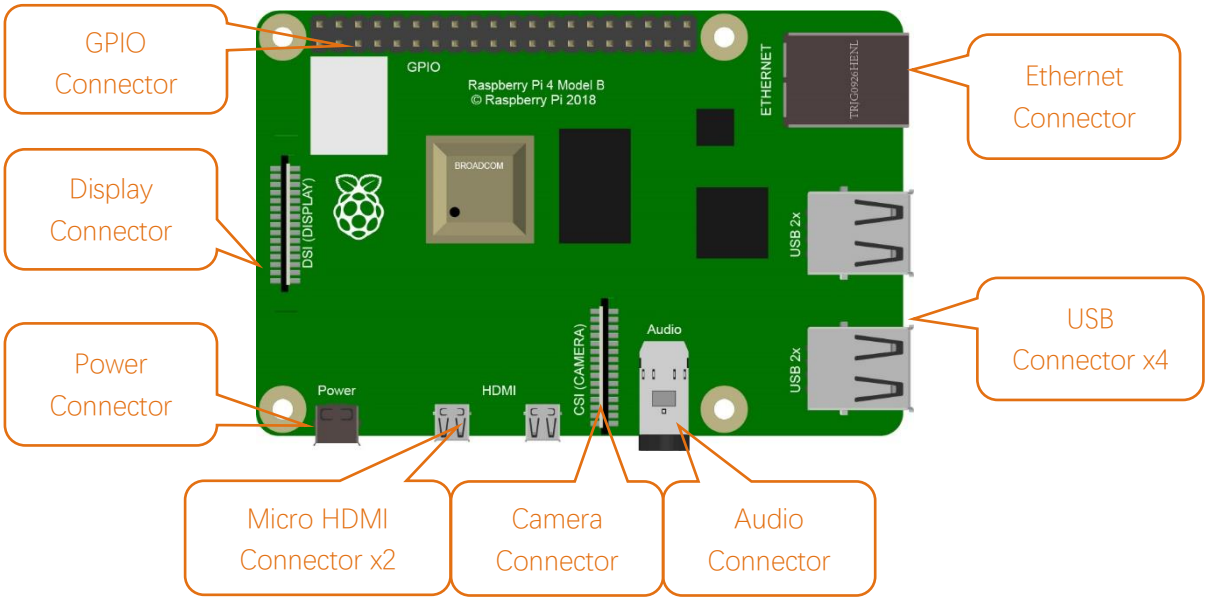
Actual image of Raspberry Pi 1 Model B+: 	CAD image of Raspberry Pi 1 Model B+: 
Actual image of Raspberry Pi 3 Model A+: 	CAD image of Raspberry Pi 3 Model A+: 

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: 

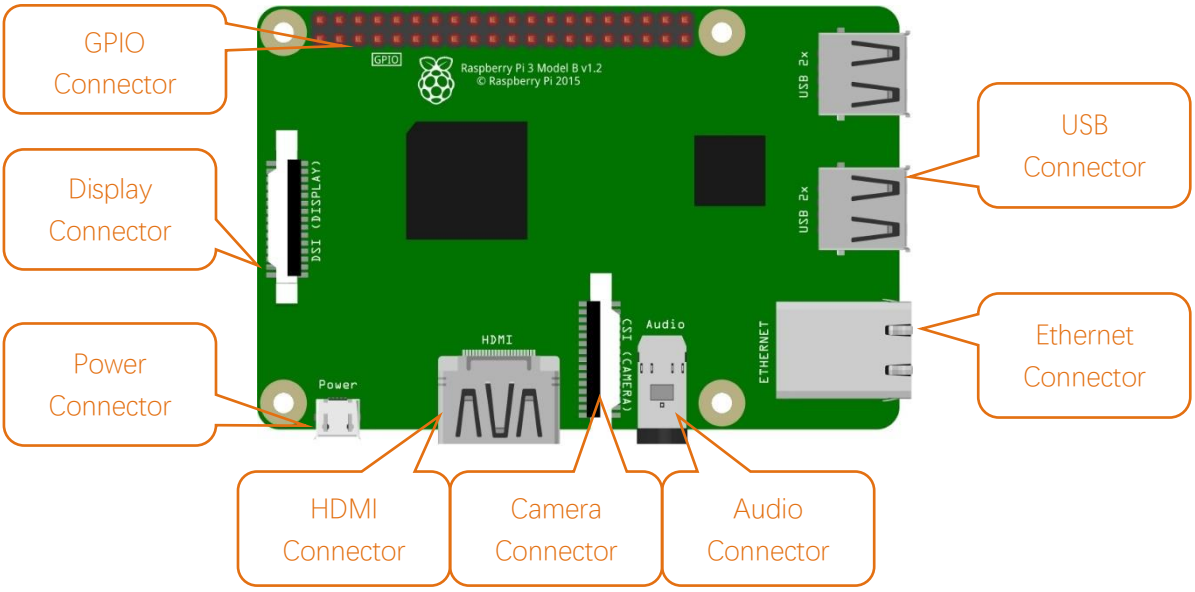
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.
Hardware interface diagram of RPi 5 is shown below:



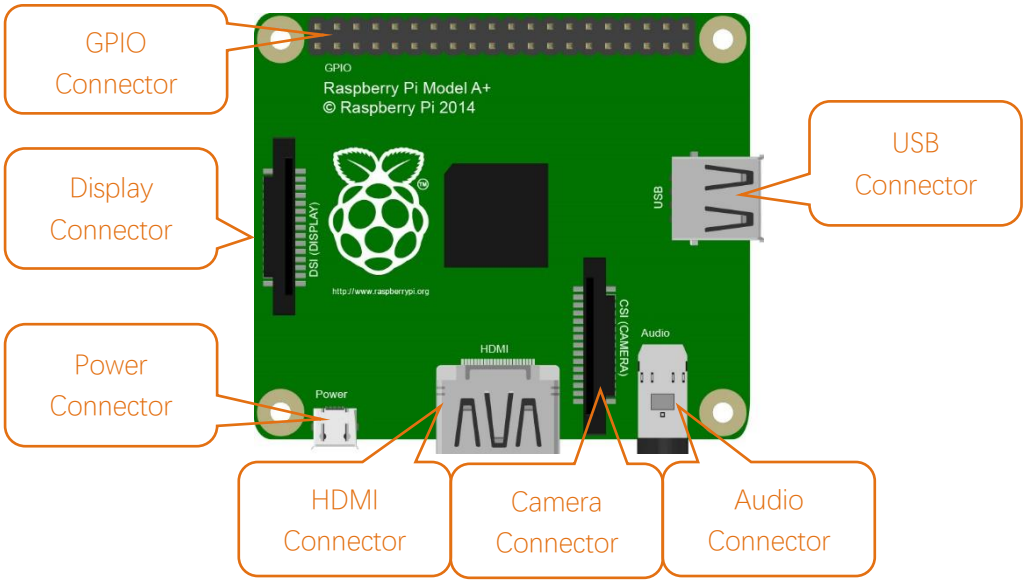
Hardware interface diagram of RPi 4B is shown below:



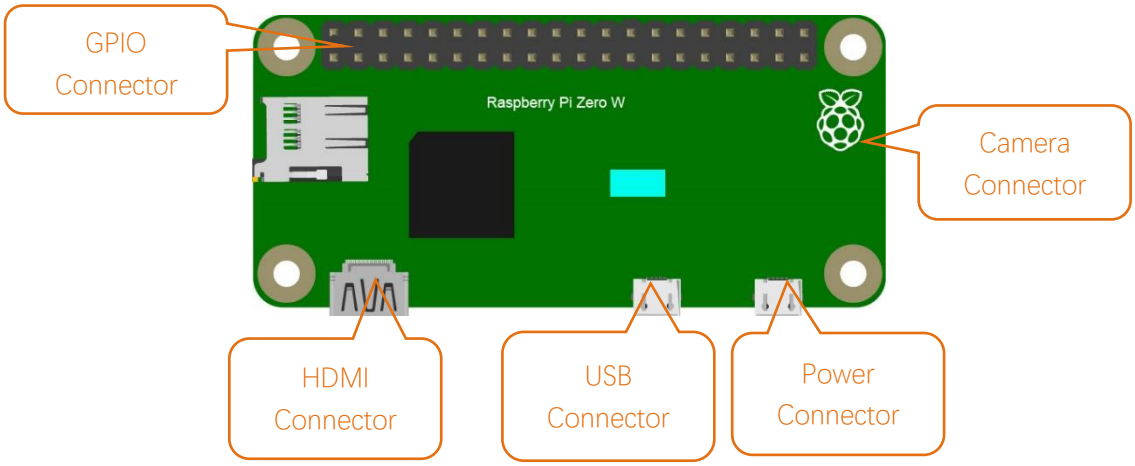
Hardware interface diagram of RPi 3B+/3B/2B/1B+:



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



Hardware interface diagram of RPi Zero/Zero W:

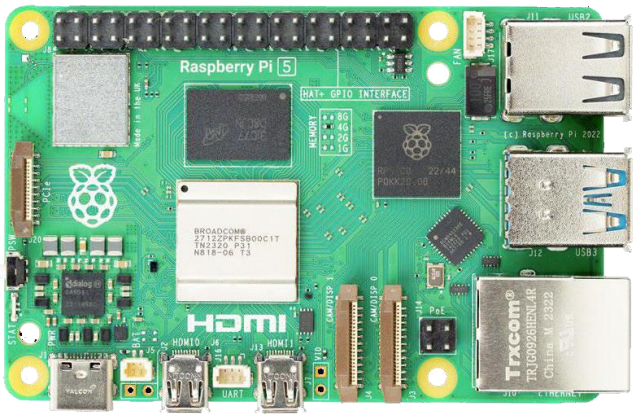
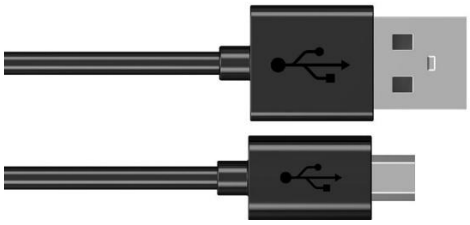


Installing an Operating System

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

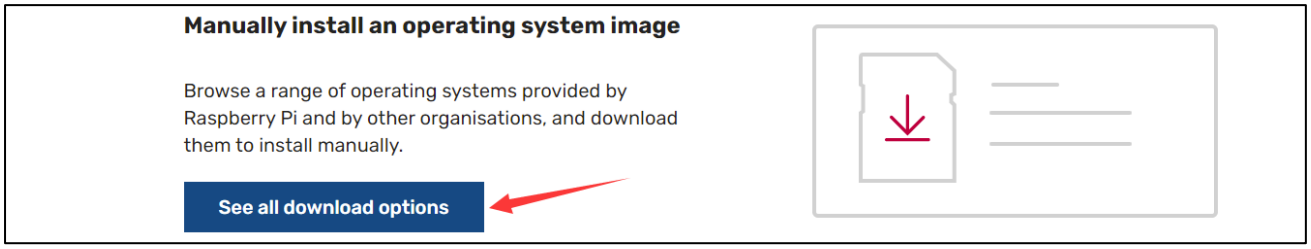
Automatically Method

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.

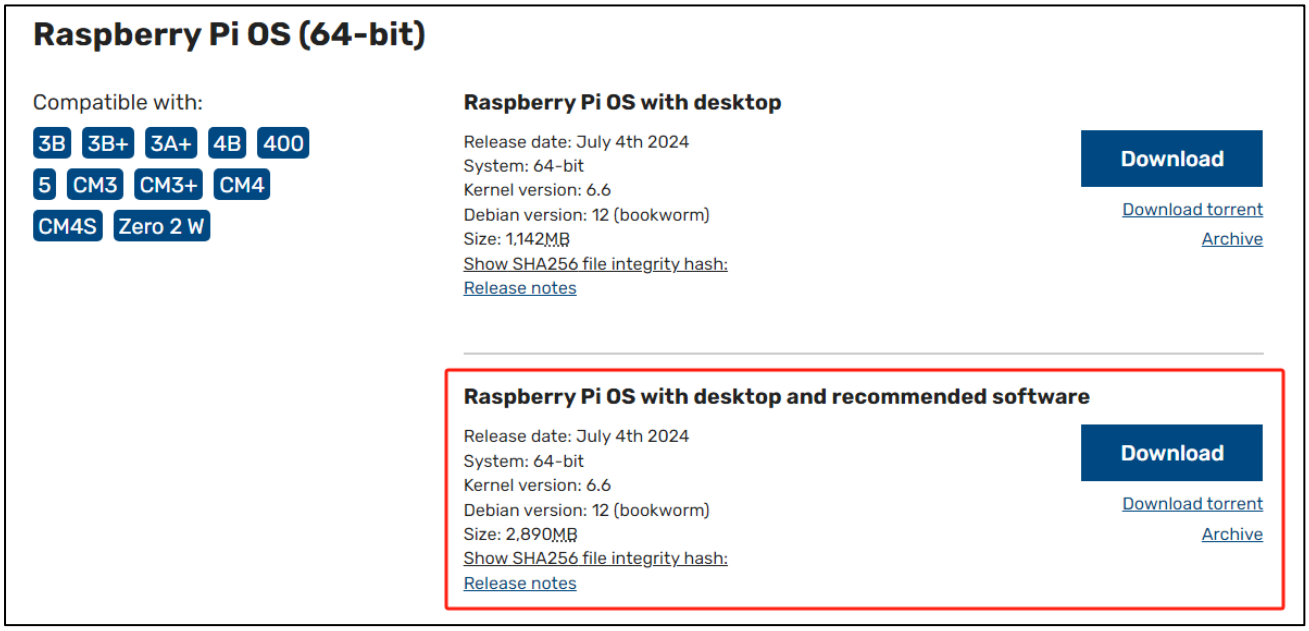
Manually Method

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

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



The image is for reference only, please select the latest version.

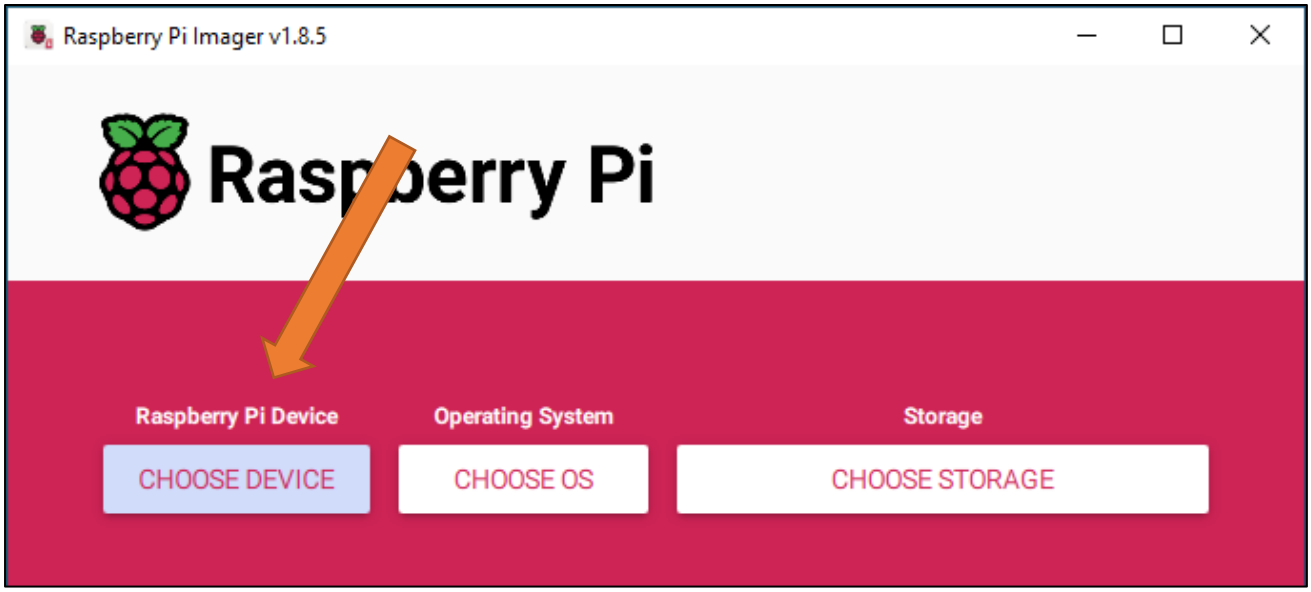


Then the zip file is downloaded.

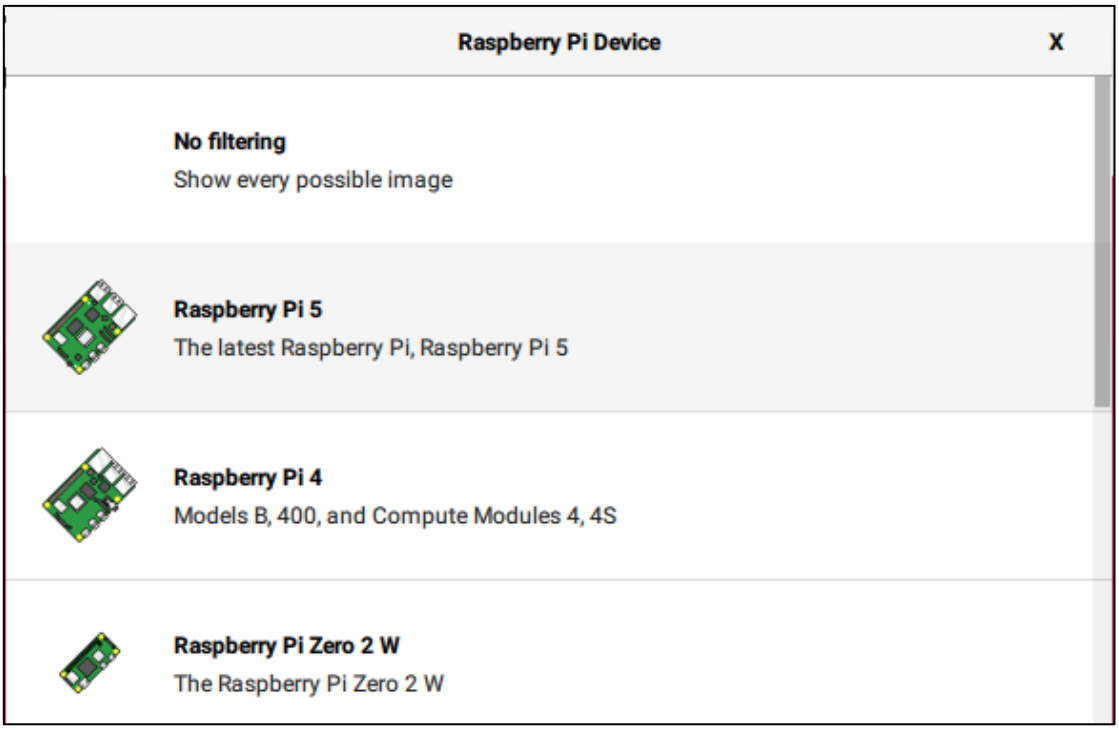
Write System to Micro SD Card

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

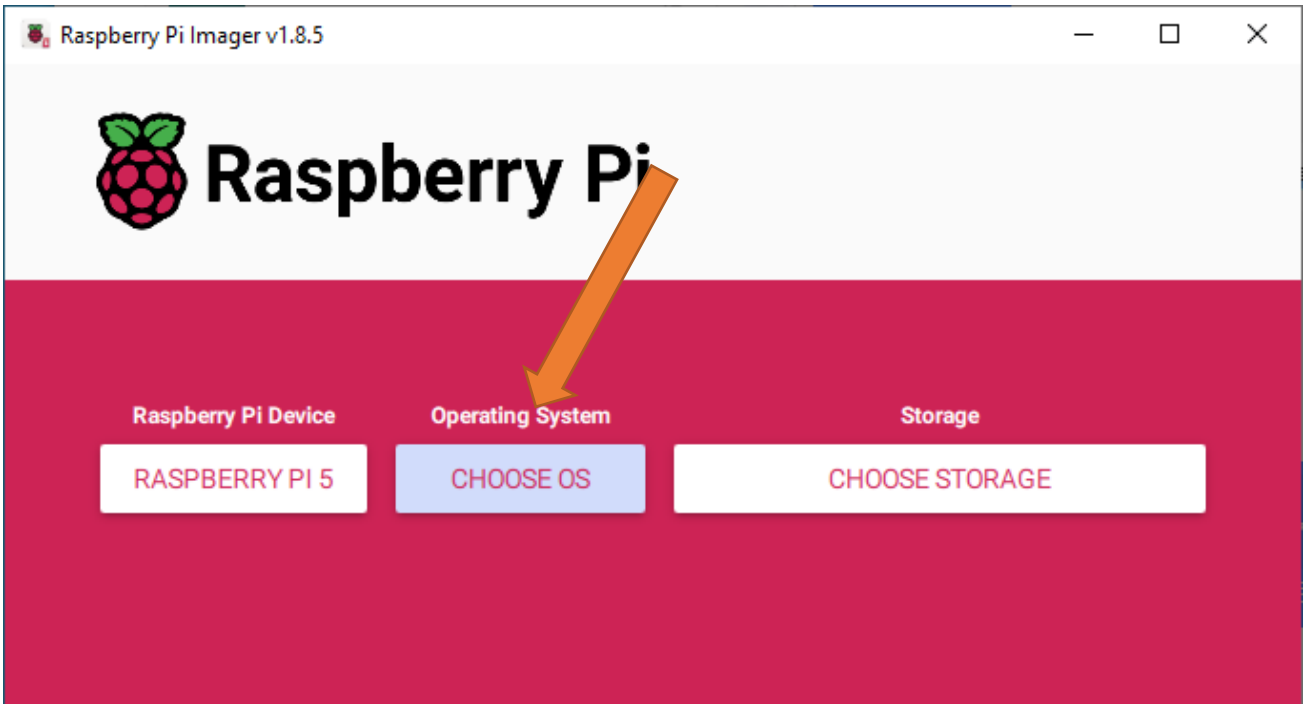
Then open imager toll. Clicked Choose Device.



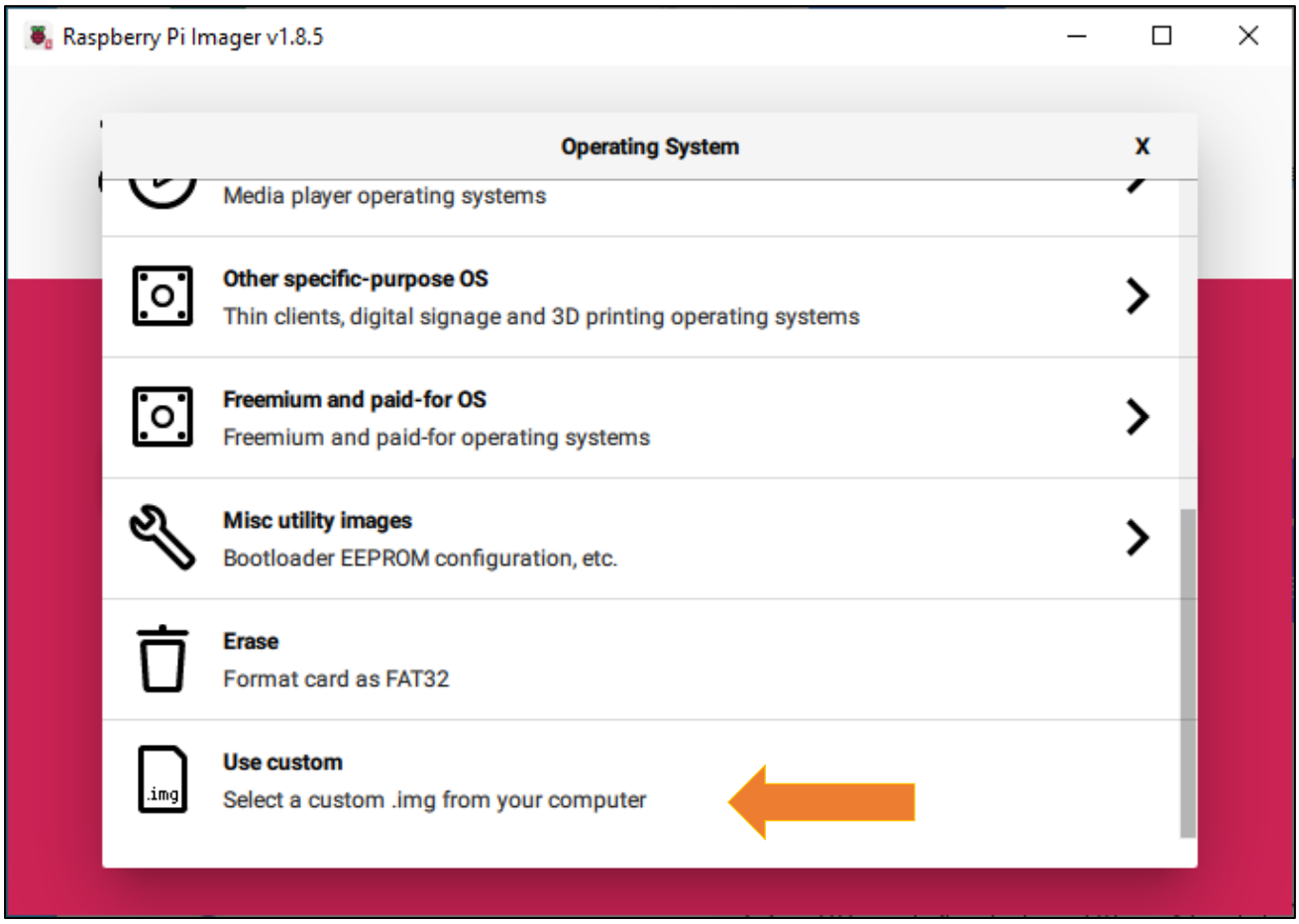
Select a Raspberry PI Device based on your Raspberry PI version. It will help us filter out the right version of the system for the Raspberry PI.



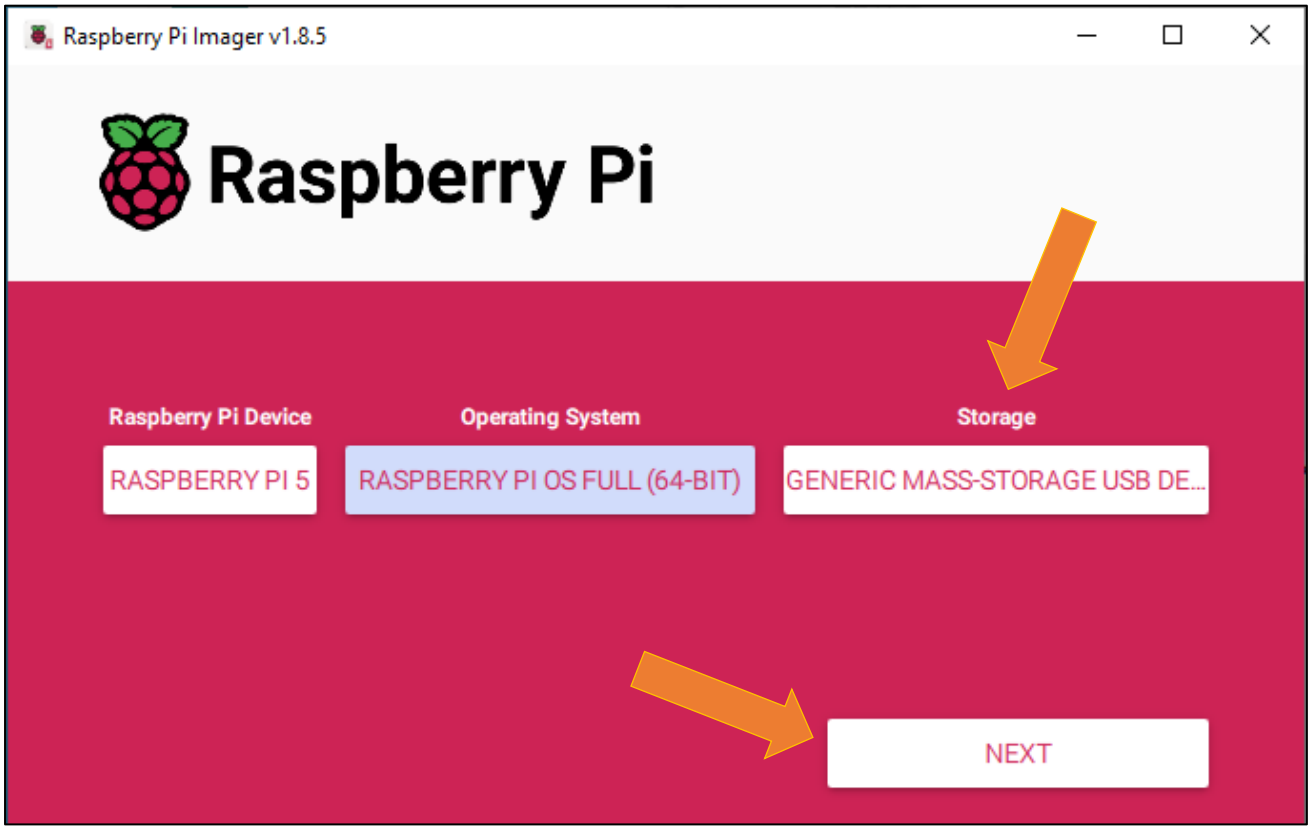
Clicked Operating System.



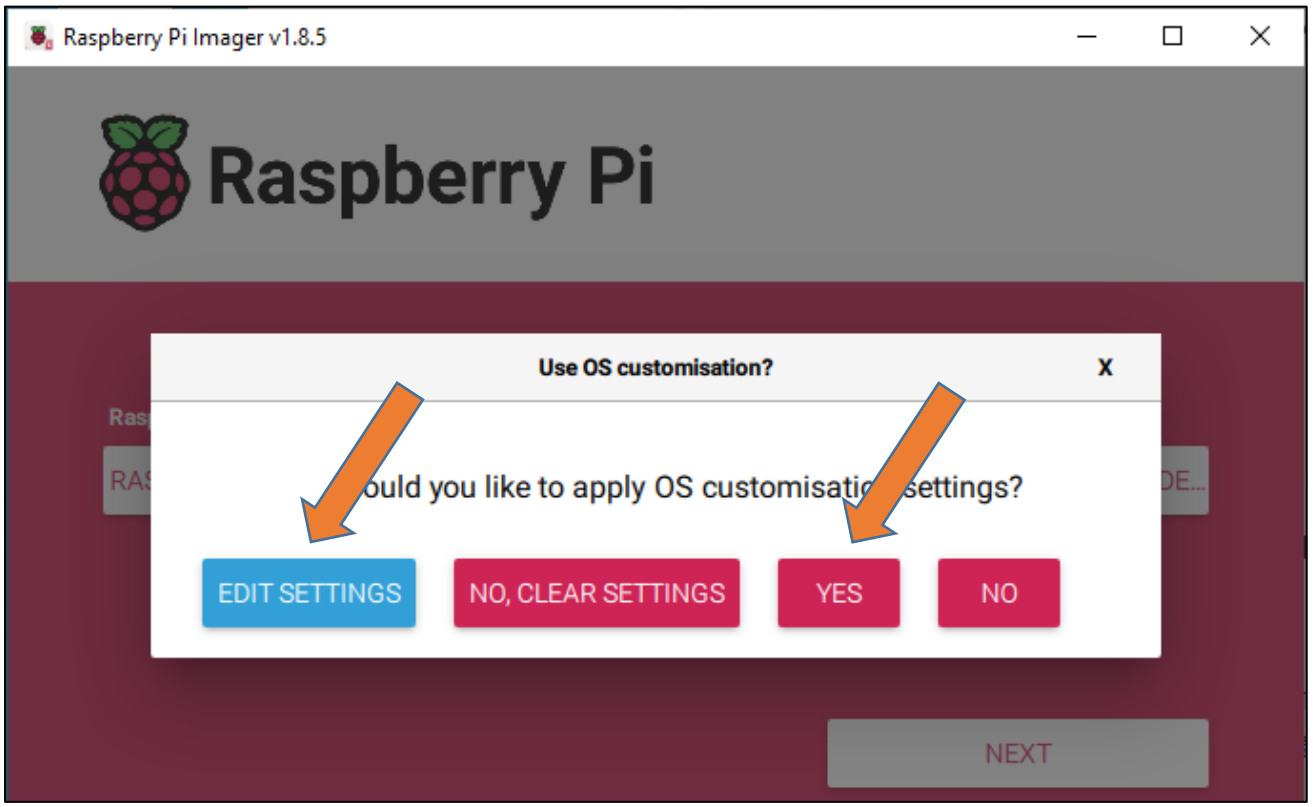
Choose system that you just downloaded in Use custom.



Choose the SD card. Then click "Next".

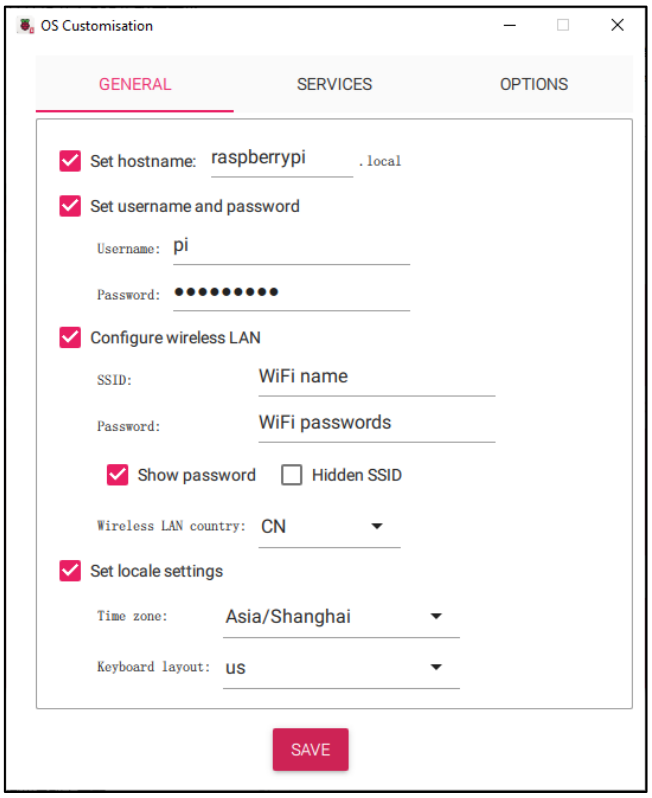


You can configure the Raspberry PI according to your needs.

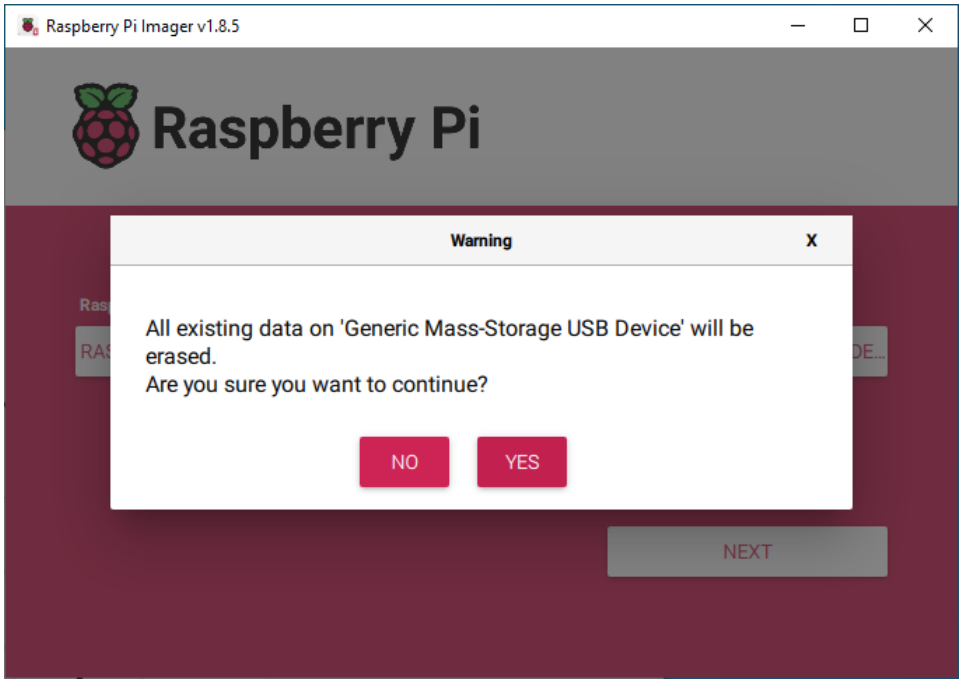


Enable ssh and configure WiFi

On the GENERAL screen, configure your information based on your actual situation.

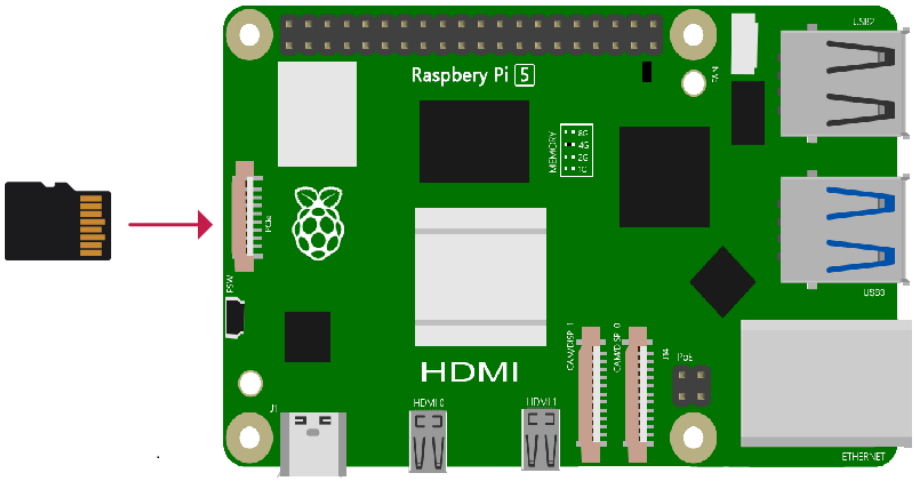


Click Save, in the new screen, click Yes, wait for SD to brush into the Raspberry system.



Insert SD card

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



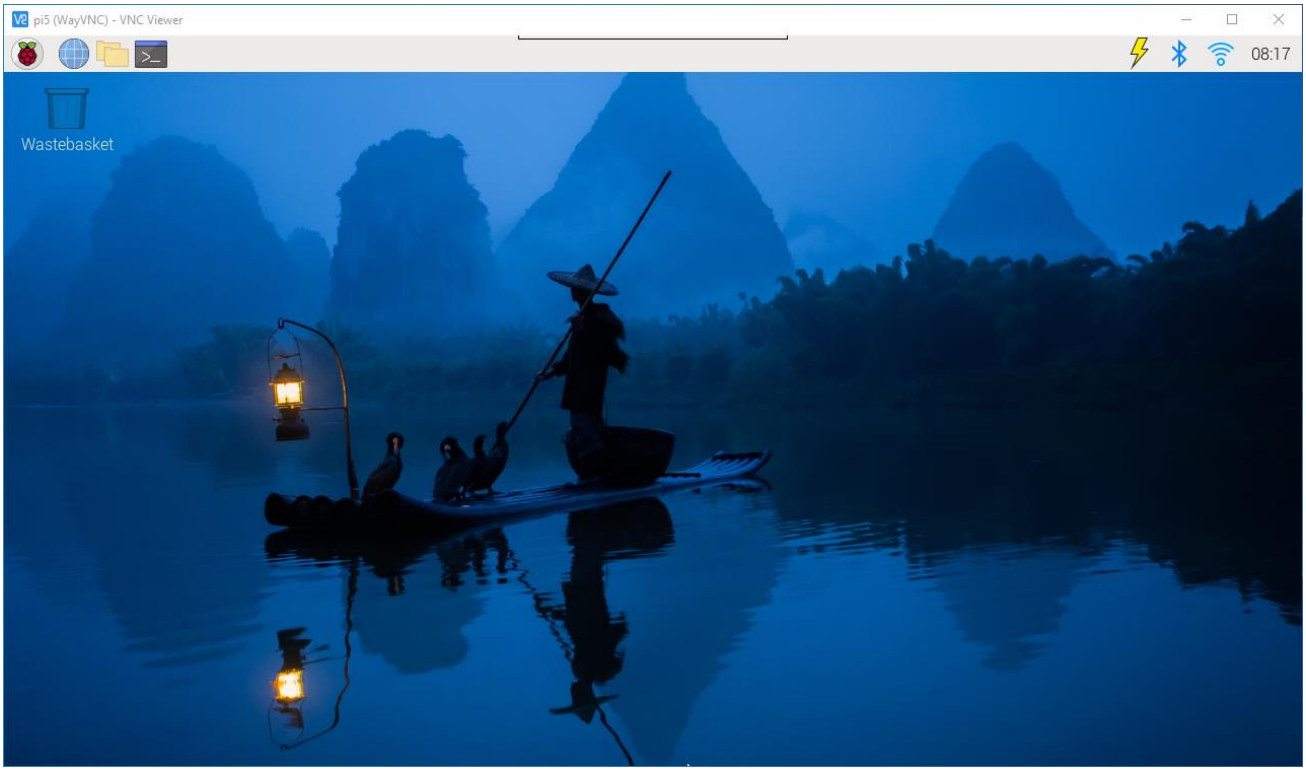
Connect to the power supply and wait for the Raspberry PI to turn on.

Getting Started with Raspberry Pi

Monitor desktop

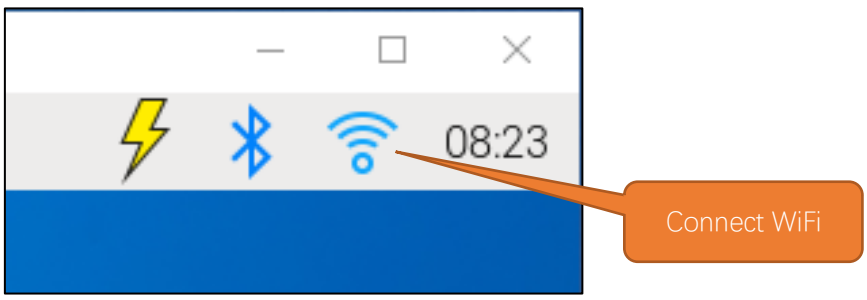
If you do not have a spare monitor, please skip to next section Remote desktop & VNC. If you have a spare monitor, please follow the steps in this section.

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.

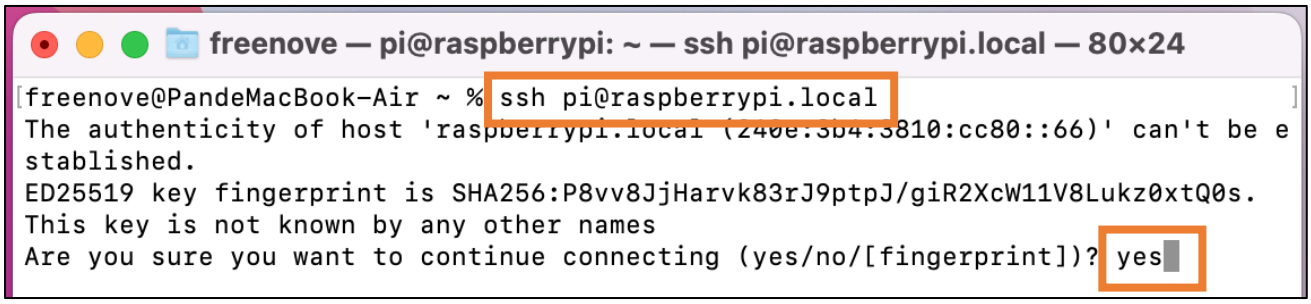
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

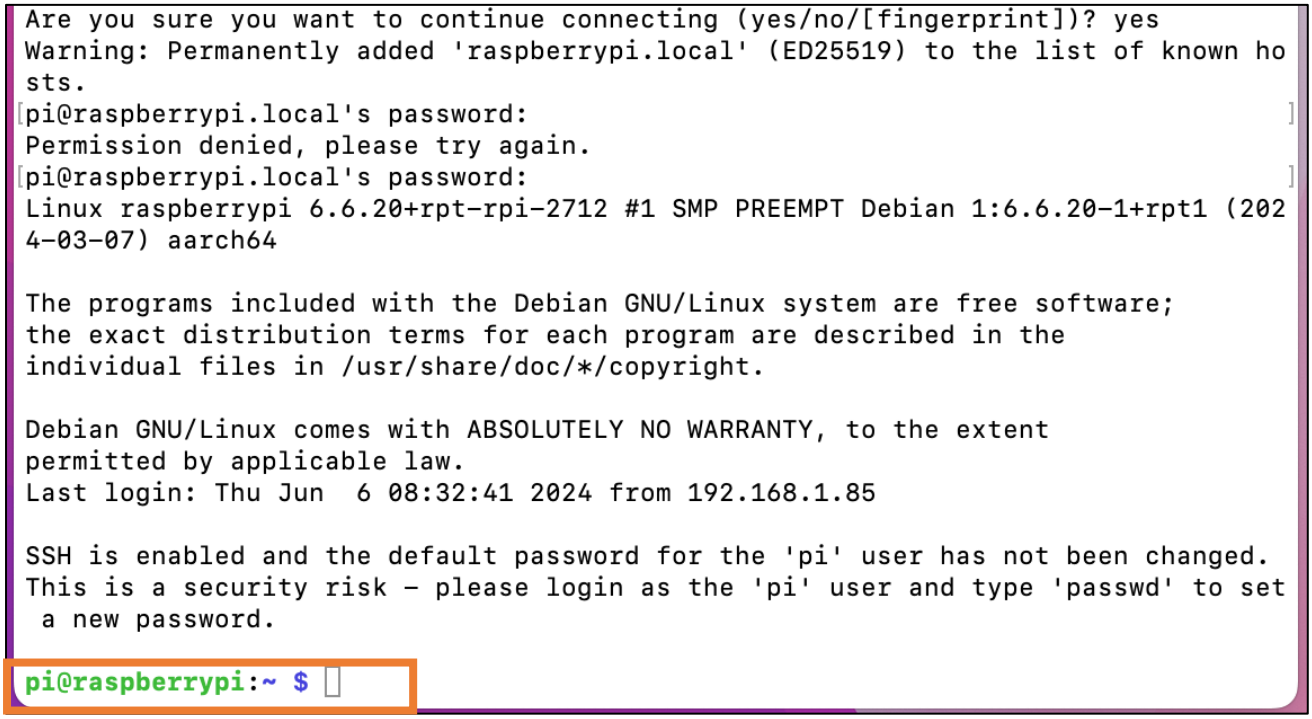
MAC OS Remote Desktop

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.





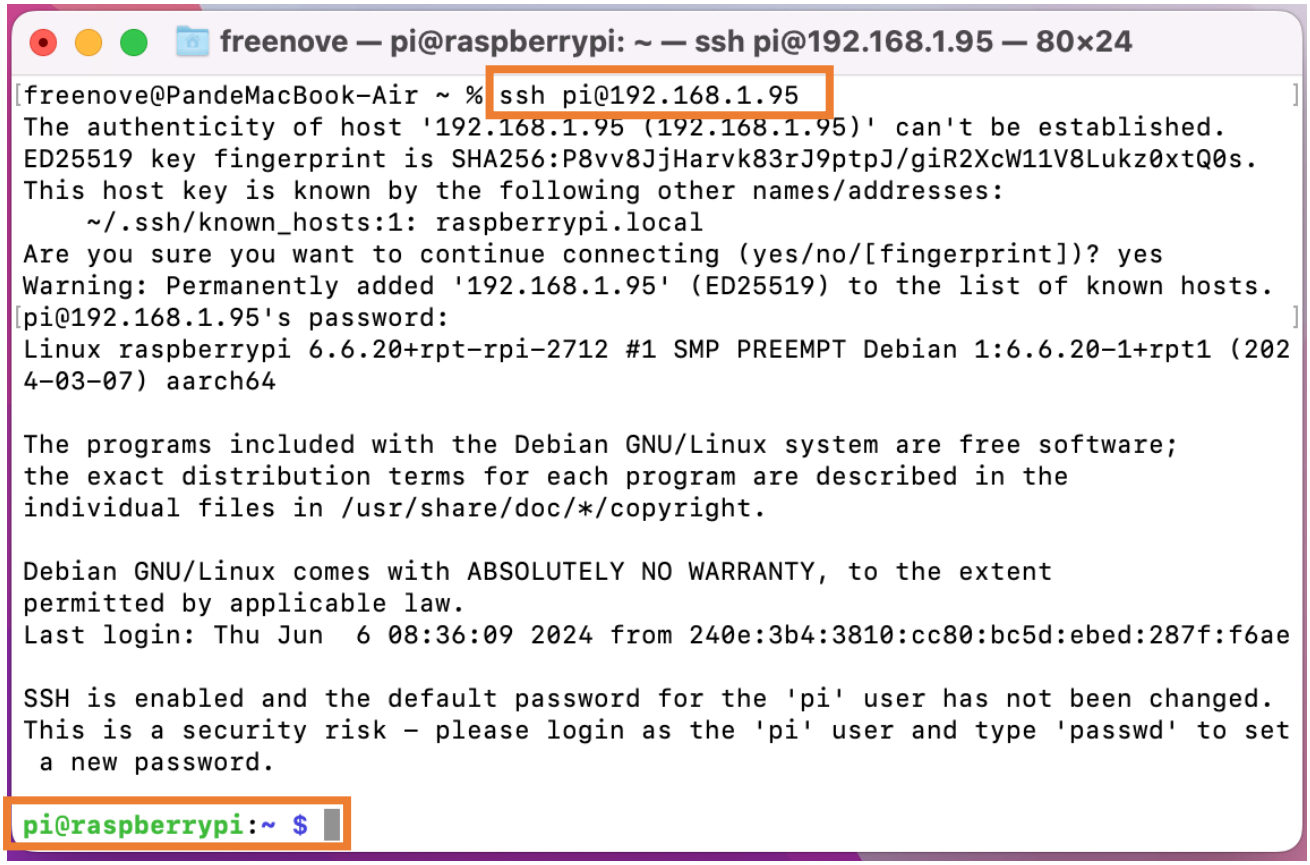
You can also use the IP address to log in Pi.

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.



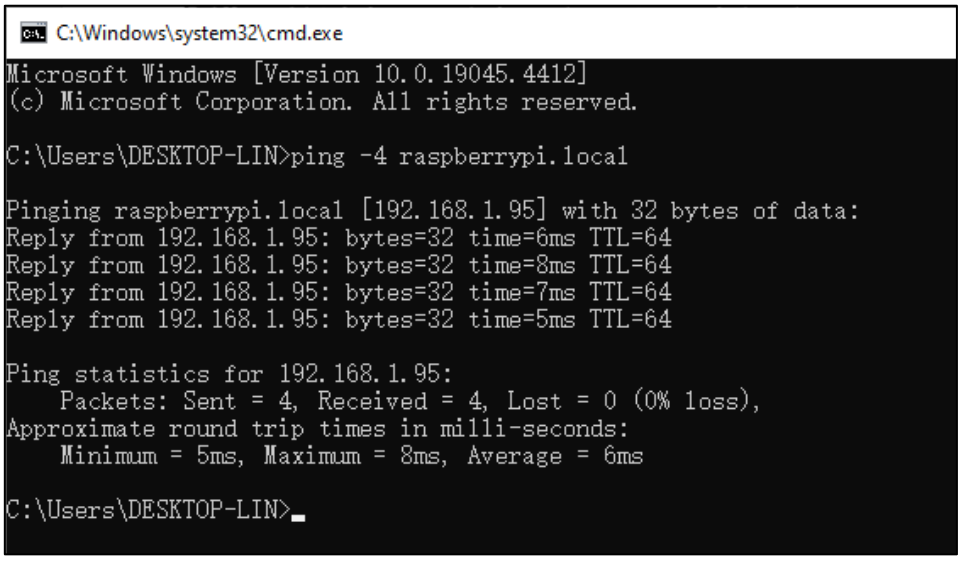
Then you can skip to VNC Viewer.

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
```



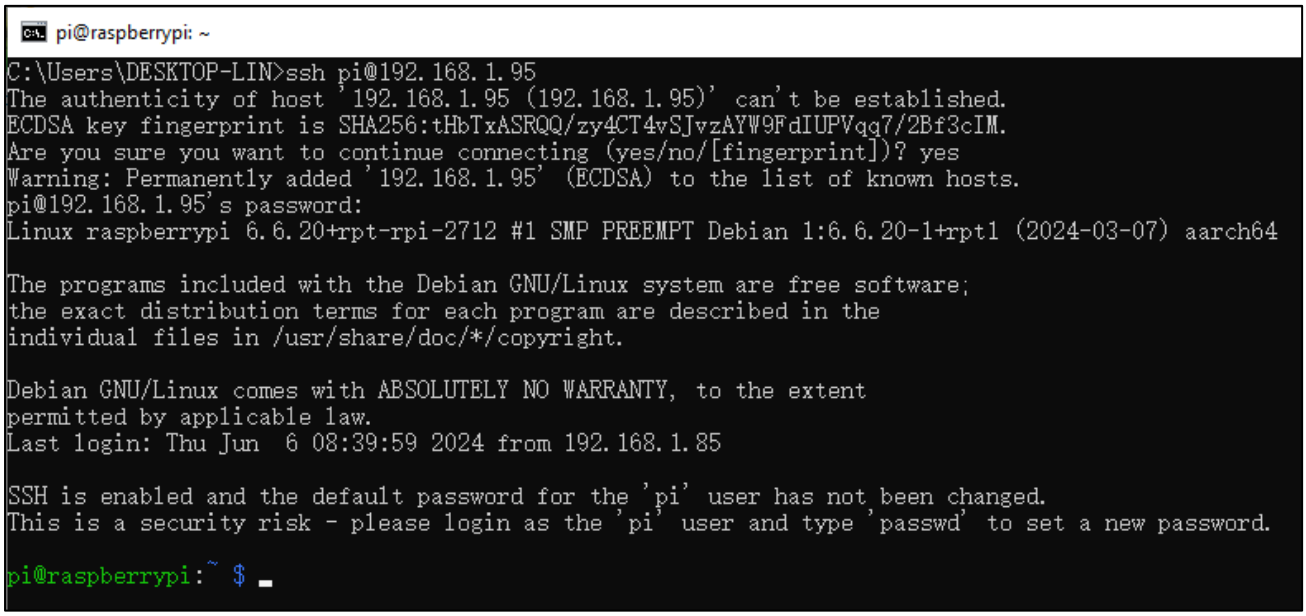
Then 192.168.1.147 is my Raspberry Pi IP.

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
```



Enable VNC

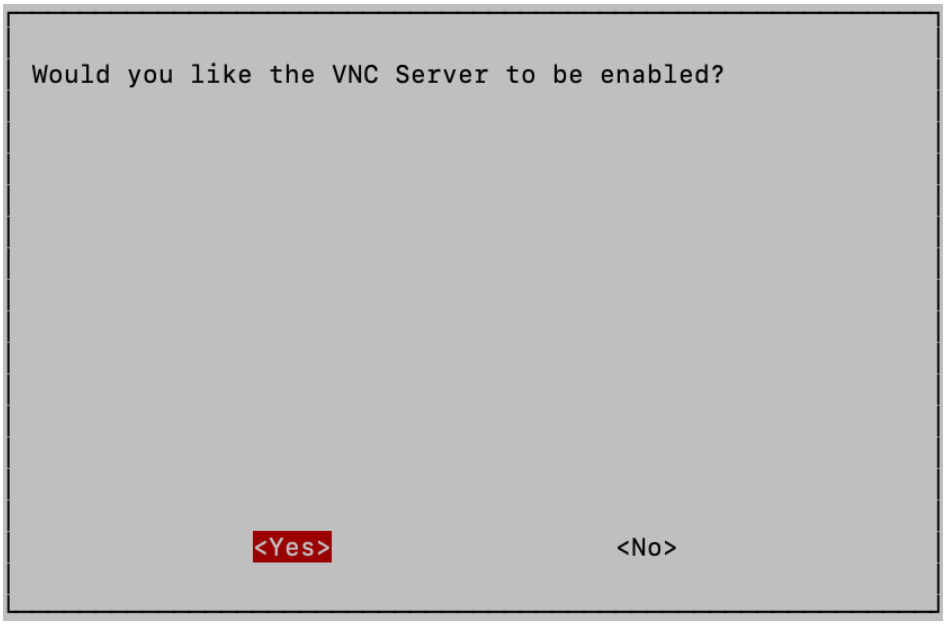
Type the following command. Select Interface Options→P5 VNC → Enter→Yes→OK. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

```
sudo raspi-config
```

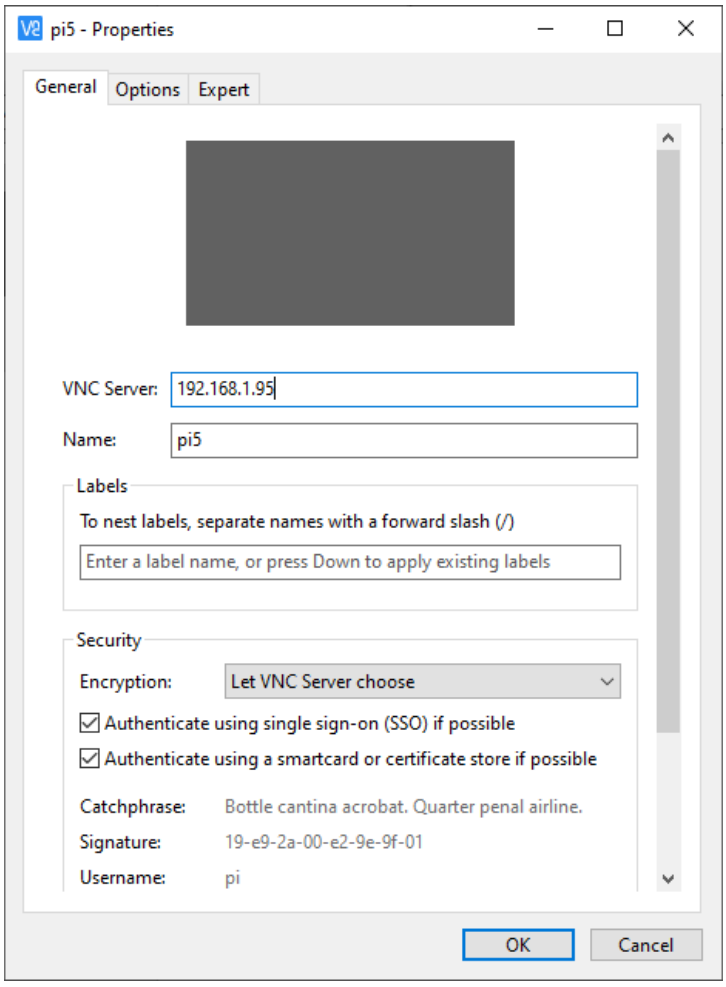
```
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

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

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

            <Select>                     <Finish>
```

```
Raspberry Pi Software Configuration Tool (raspi-config)

I1 SSH         Enable/disable remote command line access using SSH
I2 RPi Connect Enable/disable Raspberry Pi Connect
I3 VNC         Enable/disable graphical remote desktop access
I4 SPI         Enable/disable automatic loading of SPI kernel module
I5 I2C         Enable/disable automatic loading of I2C kernel module
I6 Serial Port Enable/disable shell messages on the serial connection
I7 1-Wire      Enable/disable one-wire interface
I8 Remote GPIO Enable/disable remote access to GPIO pins

            <Select>                     <Back>
```

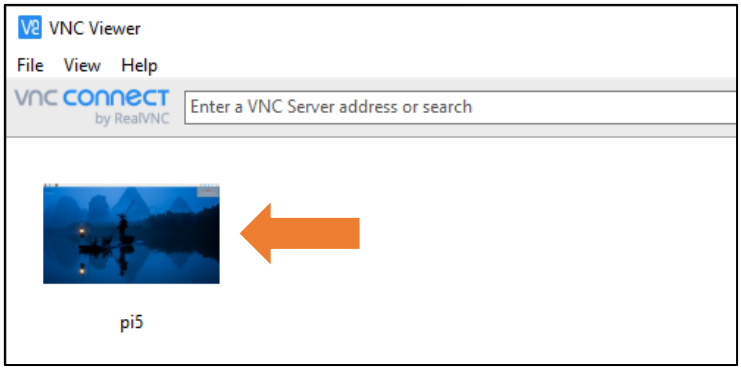


Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. Click File → New Connection. The interface is shown below.

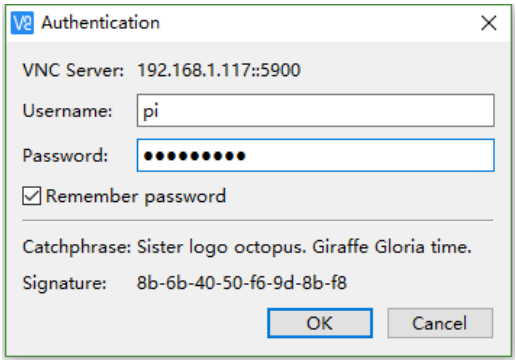


Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

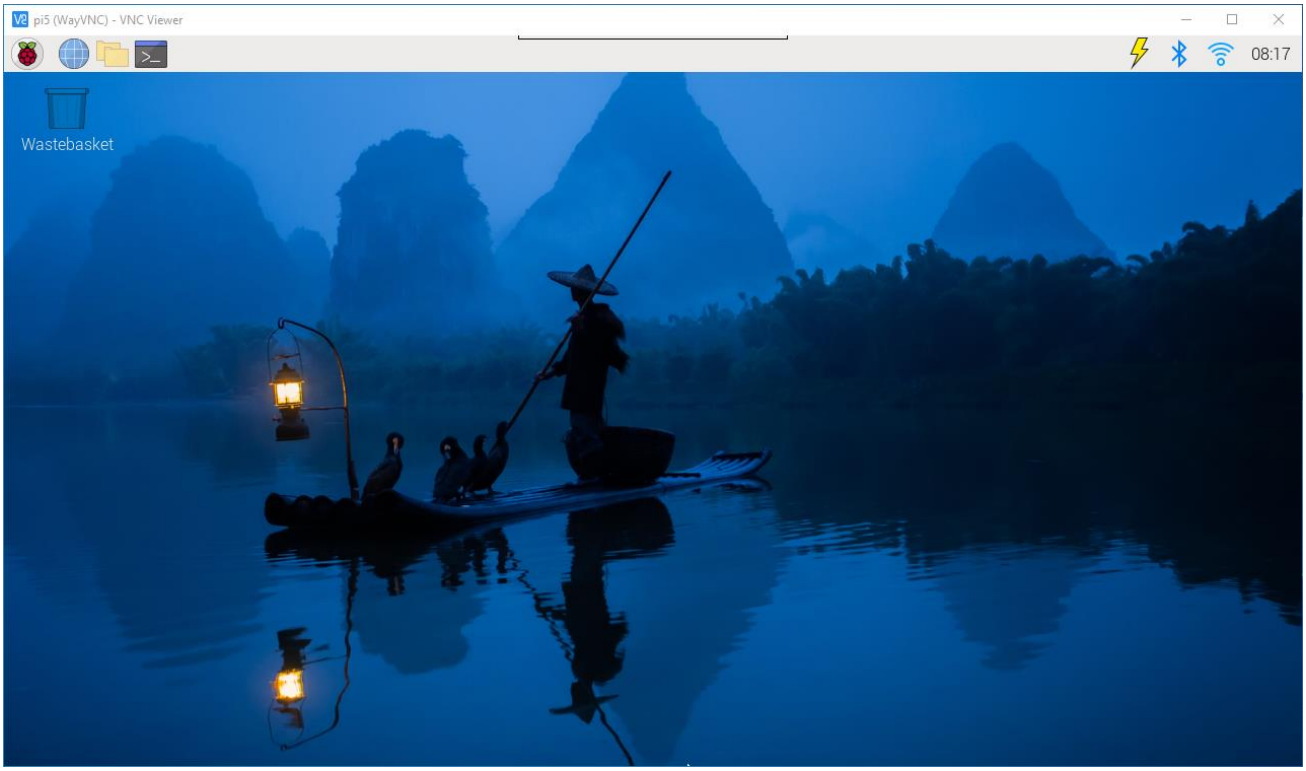
Then on the VNC Viewer panel, double-click new connection you just created,



and the following dialog box pops up.



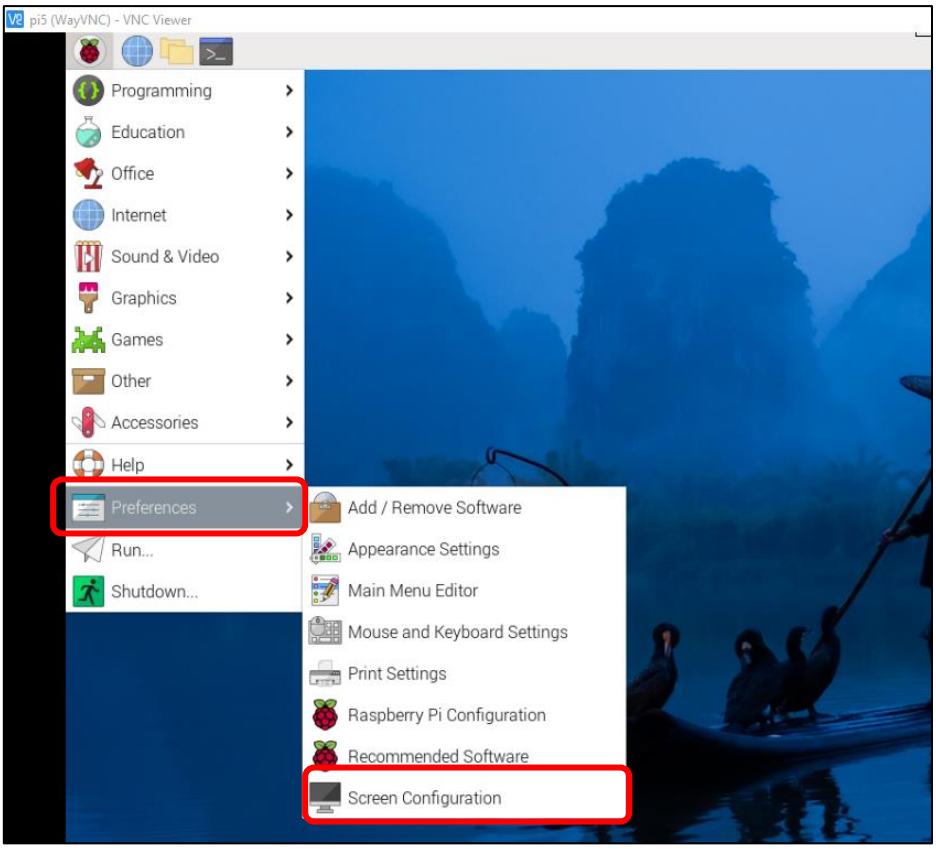
Enter username: **pi** and Password: **raspberry**. And click OK.



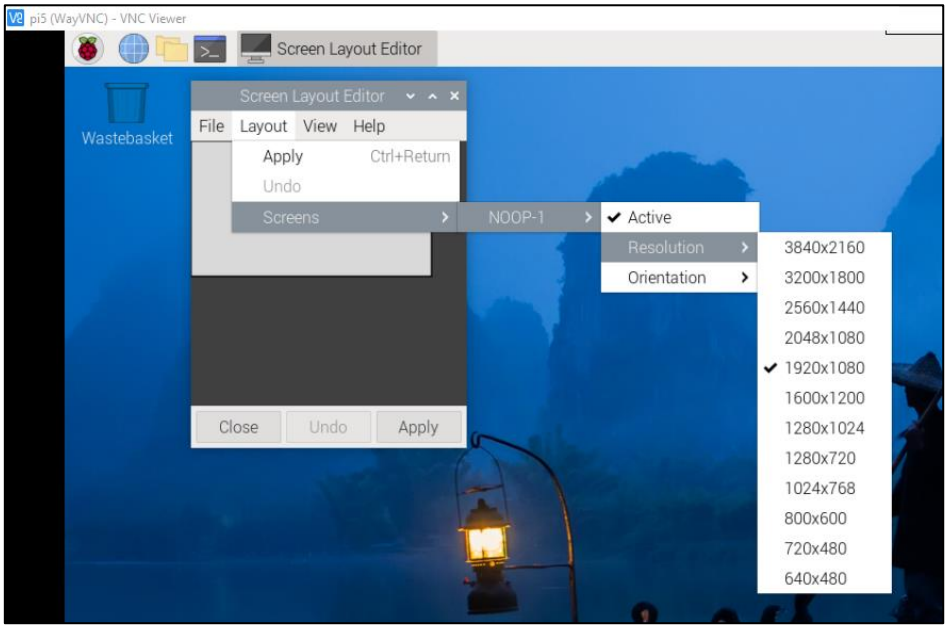
Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

Set Resolution

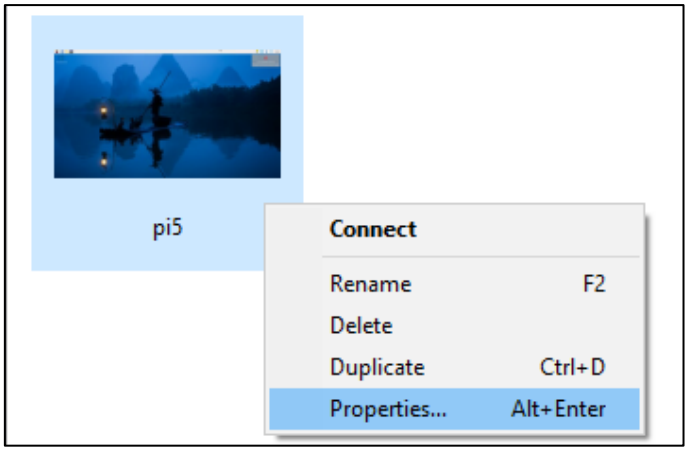
You can also set other resolutions.

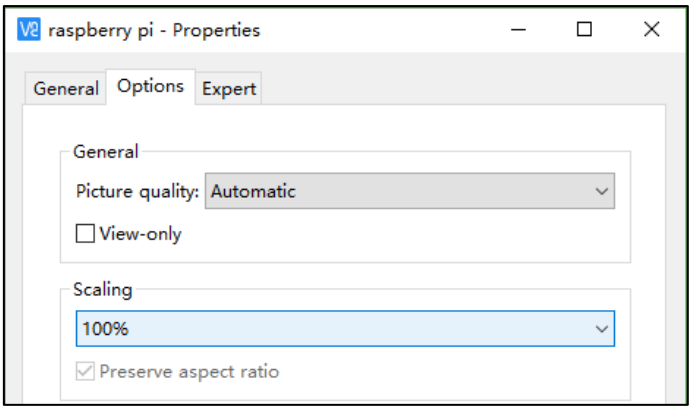


If you don't know what resolution to set properly, you can try 1920x1080.

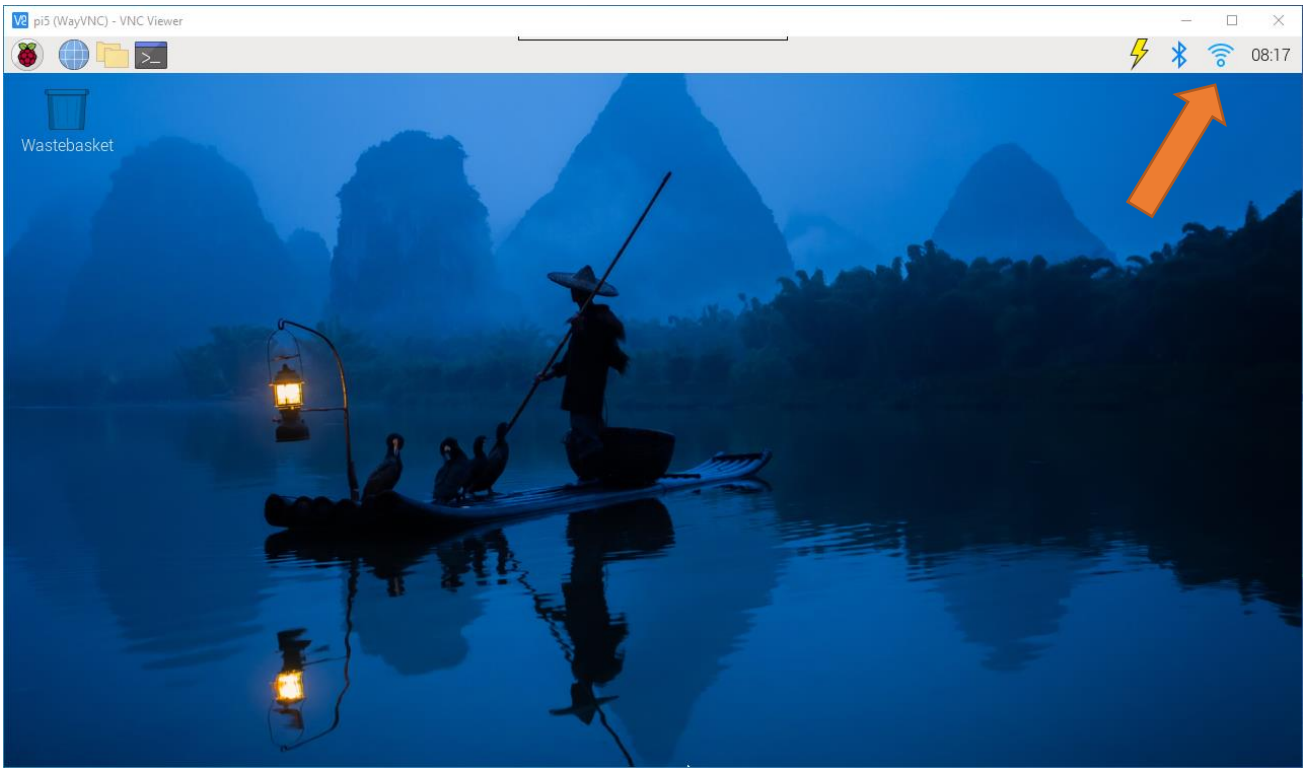


In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. And select Properties->Options label->Scaling. Then set proper scaling.





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.



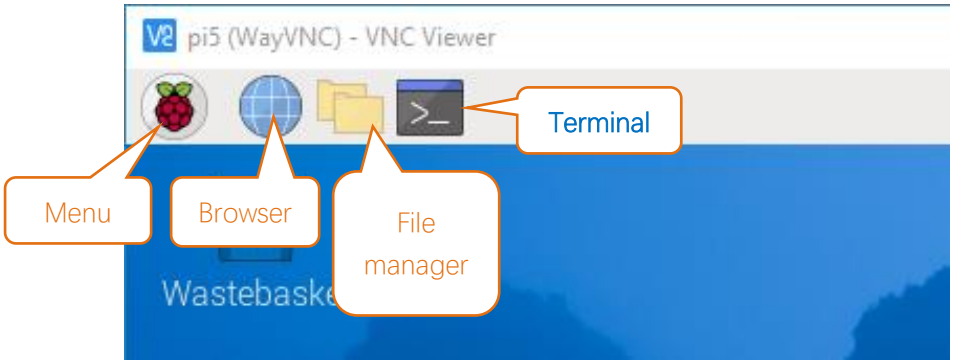
Chapter 0 Preparation

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.

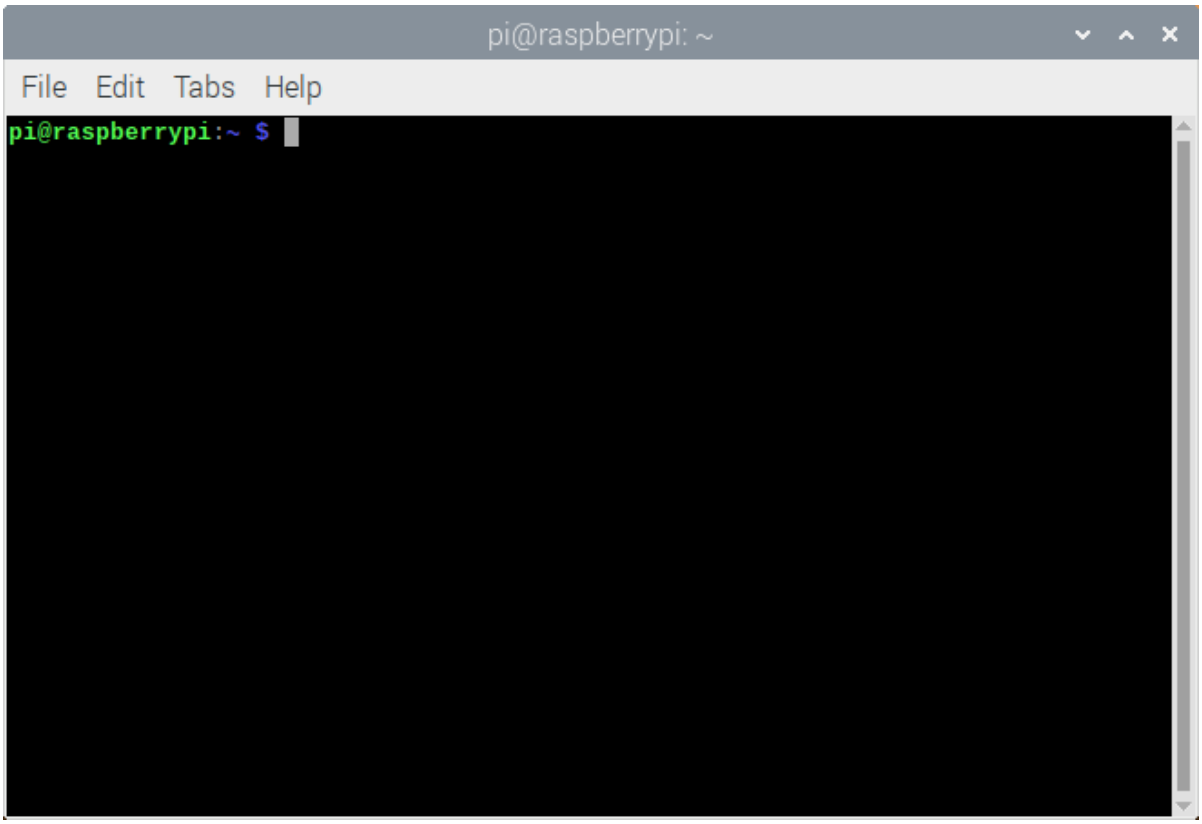
Linux Command

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.

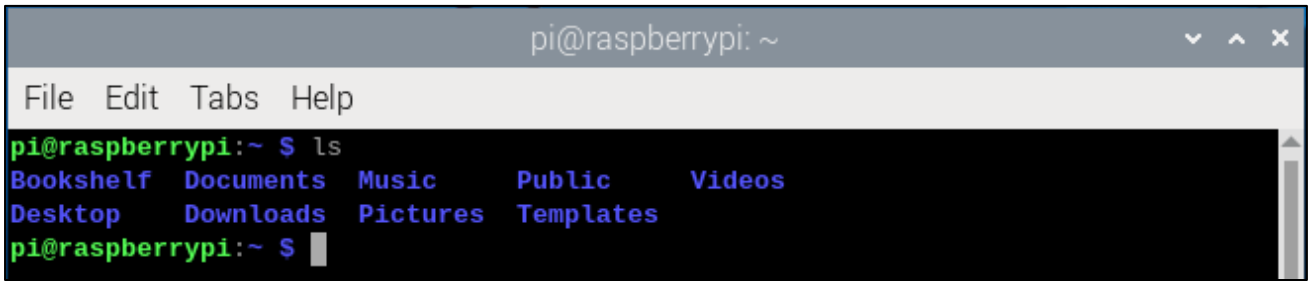


When you click the Terminal icon, following interface appears.



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/
```

Pi4j Introduction

Pi4J is a Java I/O library specially designed for Raspberry Pi platform.

The Pi4J project aims to provide Java programs with access, control and communication to the core I/O functions of the Raspberry Pi, enabling Java programmers to easily access and control the full I/O capabilities of the Raspberry Pi platform. It abstracts the low-level native integration and interrupt monitoring to enable Java programmers to focus on implementing their application business logic.

It is recommended to use JBang to run pi4j code. JBang allows you to execute Java code with dependencies as a single file without the need for a full Maven or Gradle project. You also don't need to compile your code. So it's a very easy way to get started with Java and Pi4J.

https://www.pi4j.com/examples/jbang/

To learn more about Pi4J, please refer to the documentation linked below:
https://www.pi4j.com/documentation/

Download Code

Run the following command to download the code to Raspberry Pi.

```
cd ~
git clone --depth 1 https://github.com/Freenove/Freenove_Projects_Kit_for_Raspberry_Pi.git
```

Run the command to rename the folder.

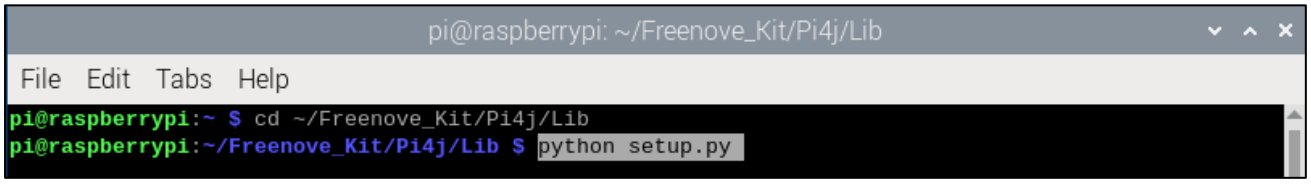
```
mv Freenove_Projects_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

Installation of JBang

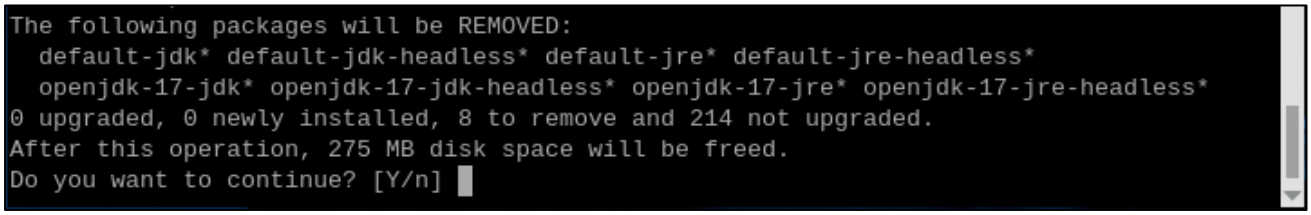
Run the following commands one by one to install jbang.

```
cd ~/Freenove_Kit/Pi4j/Lib
python setup.py
```

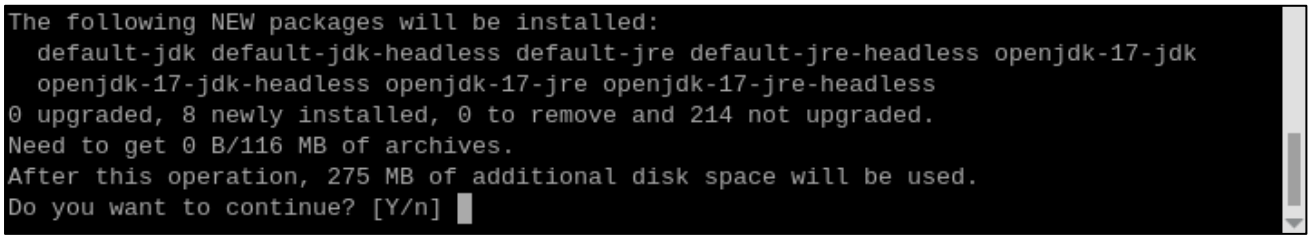
Please note that sudo is not applicable here.



The built-in default-jdk library is not complete, so we need to uninstall and reinstall it.
Enter 'Y' to uninstall it.



Enter 'Y' again to install the full default-jdk library.



When you see the messages below, it indicates that the installation is almost finished.
You can run the following commands one by one to check whether jbang is installed.

```
source ~/.bashrc
jbang --version
```

```
Successfully extracted and moved contents of jbang-0.119.0.tar to /home/pi/.jbang
Added export PATH="$HOME/.jbang/bin:$PATH" to /home/pi/.bashrc

Please run the following command in your terminal to update your PATH:
1, source ~/.bashrc
2, jbang --version
pi@raspberrypi:~/Freenove_Kit/Pi4j/Lib $ source ~/.bashrc
pi@raspberrypi:~/Freenove_Kit/Pi4j/Lib $ jbang --version
0.119.0
pi@raspberrypi:~/Freenove_Kit/Pi4j/Lib $
```

When you see the results as above, it means that jbang is already installed.

Installation of Geany

Geany is installed on Raspberry Pi OS by default.
You can run the following command to see if Geany is installed.

```
geany --version
```

If geany is not installed on your OS, please run the following command to install it.

```
sudo apt-get install geany
```

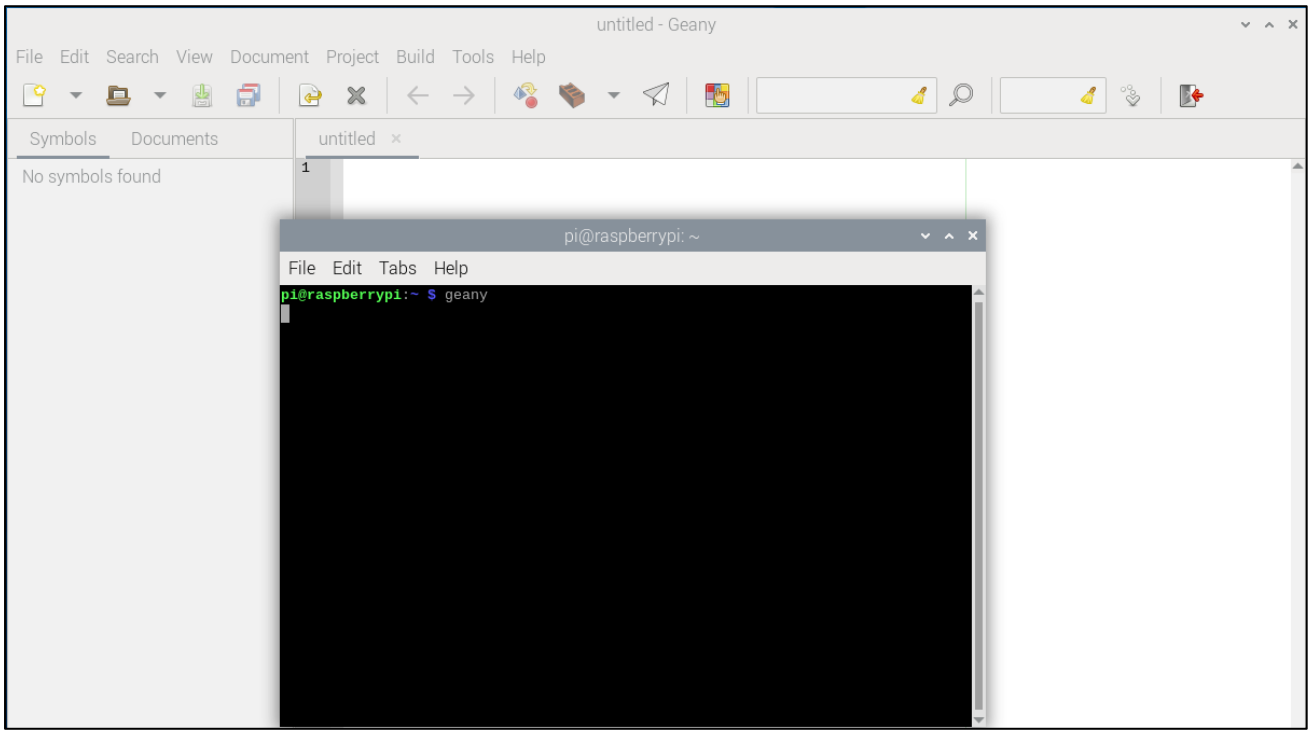
```
pi@raspberrypi:~ $ geany --version
bash: geany: command not found
pi@raspberrypi:~ $ sudo apt-get install geany
```

Geany Configuration

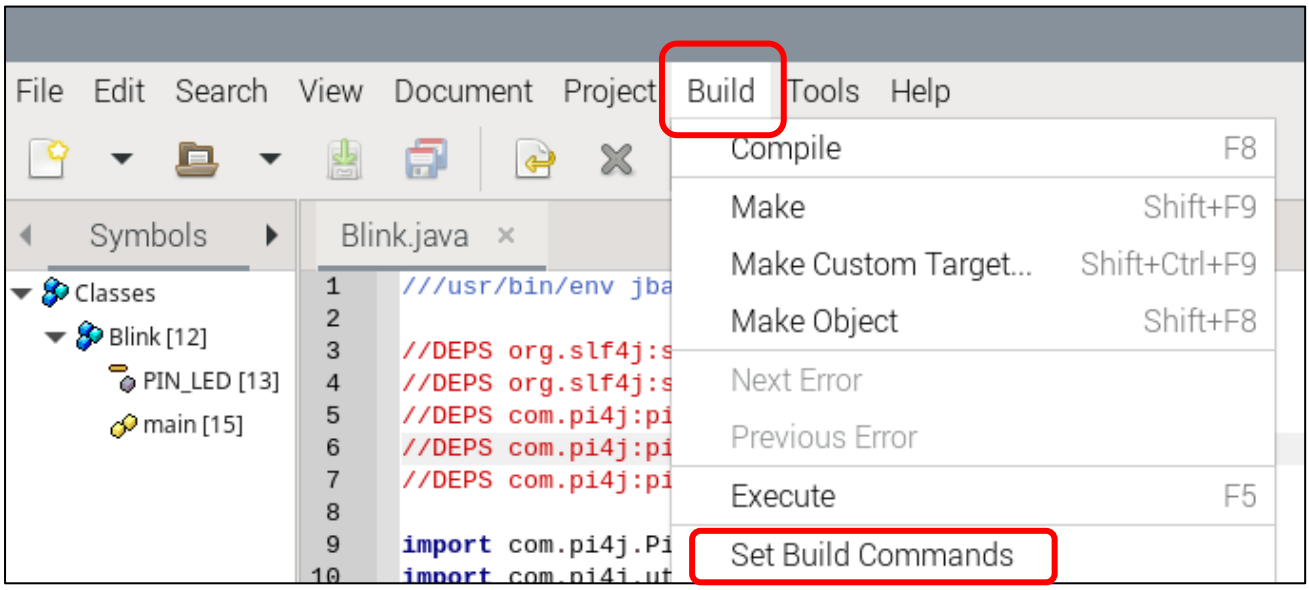
Run the command to open Geany software.

```
geany
```

As can be seen below, Geany is open after the command is run.



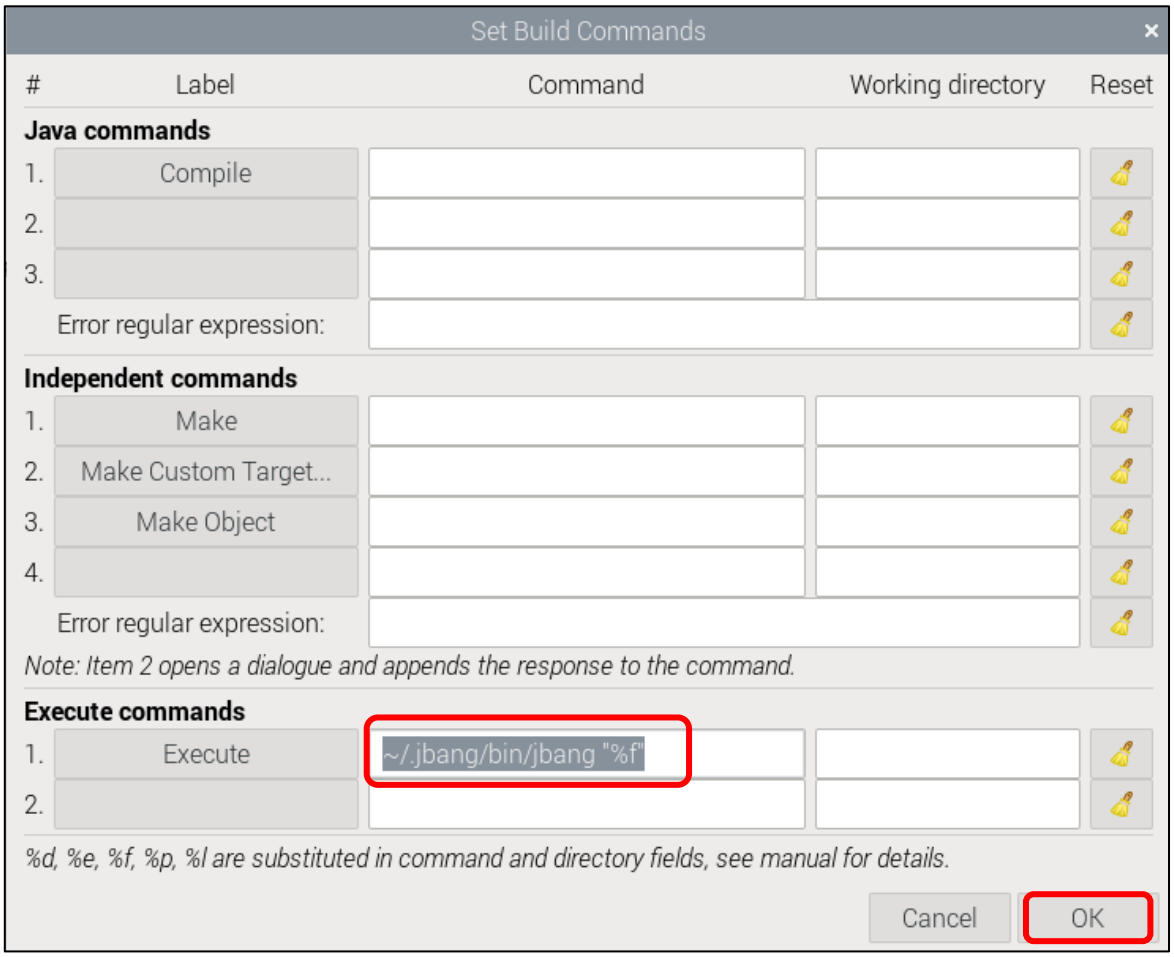
Click Build -> Set Build Commands on the menu bar.



In the pop-up window, enter the following command.

```
~/.jbang/bin/jbang "%f"
```

The detailed operation is as illustrated below:



So far, you can use Geany to open, edit, and run the code of the Pi4J tutorial.

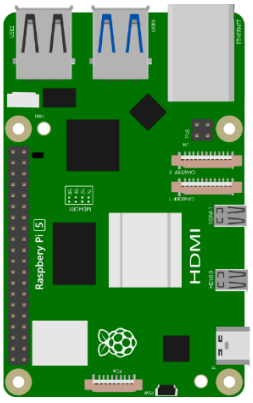
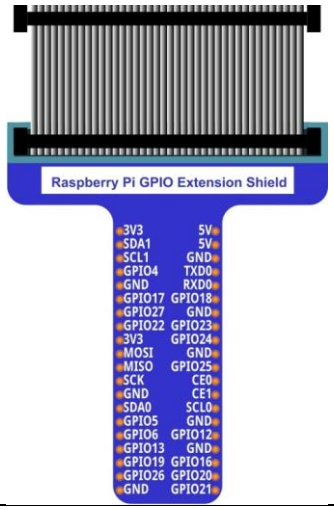
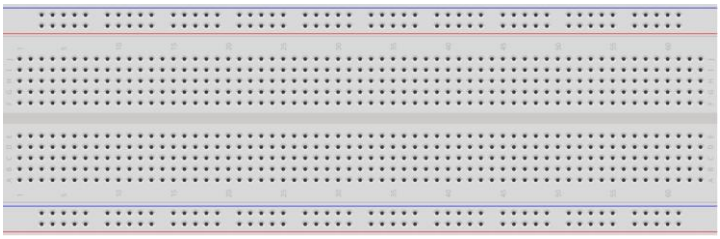
Chapter 1 LED

We will still start from Blink LED in this chapter, and learn the usage of some commonly used functions of Processing Software.

Project 1.1 Blink

In this project, we will Pi4J to control the Raspberry Pi's GPIO, so as to control the LED to blink.

Component List

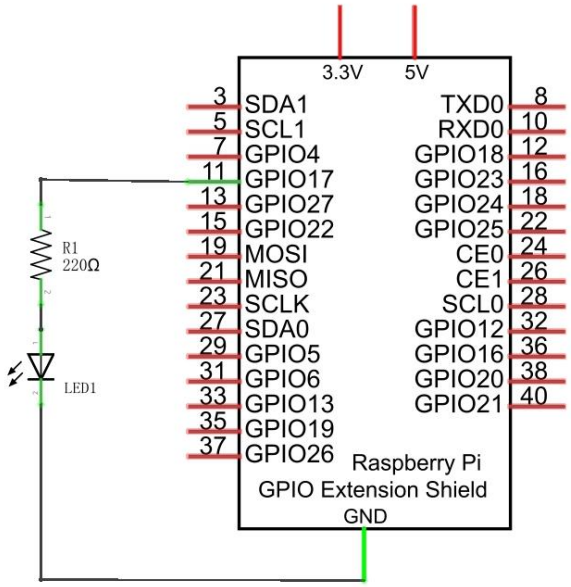
<table>
<tr><td>Raspberry Pi
(Recommended: Raspberry Pi 5 / 4B / 3B+ / 3B
Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)
  </td><td>GPIO Extension Board & Ribbon Cable
 
</td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper
Specific quantity depends on the circuit.
  </td></tr>
</table>

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).

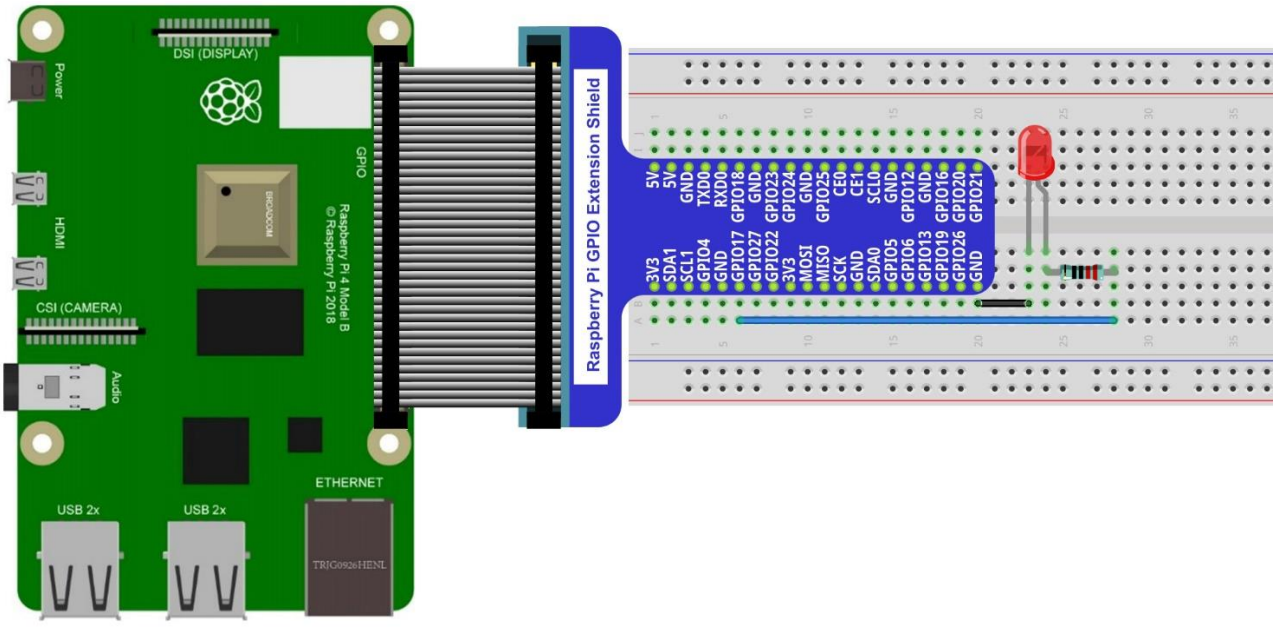
Build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield. CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!

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



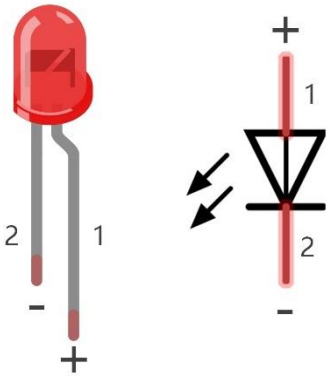
Because the numbering of the GPIO Extension Shield is the same as that of the RPi GPIO, future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

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			

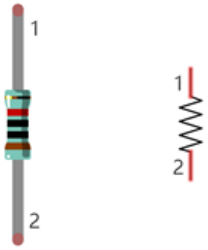
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

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.

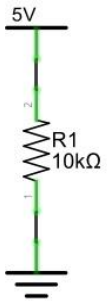
On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

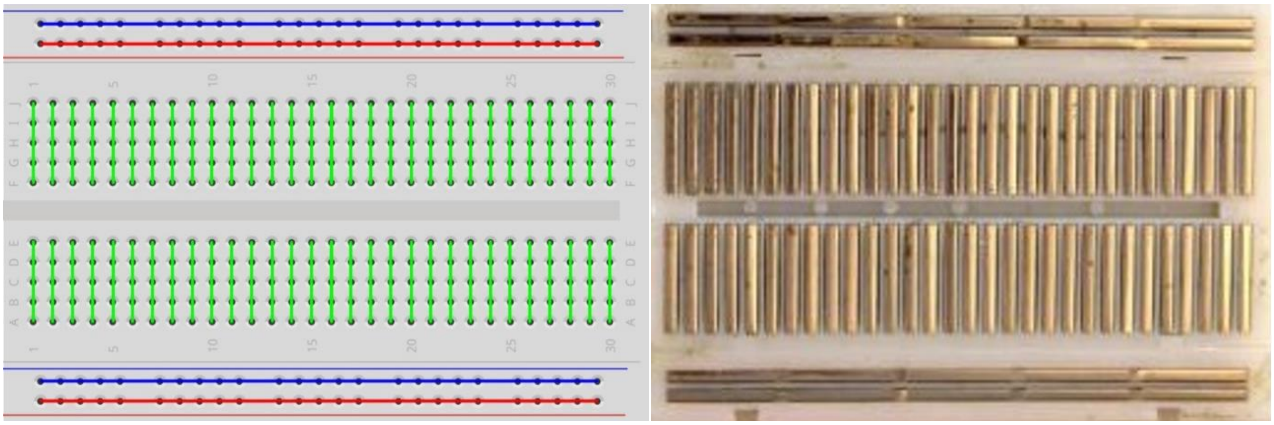


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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)

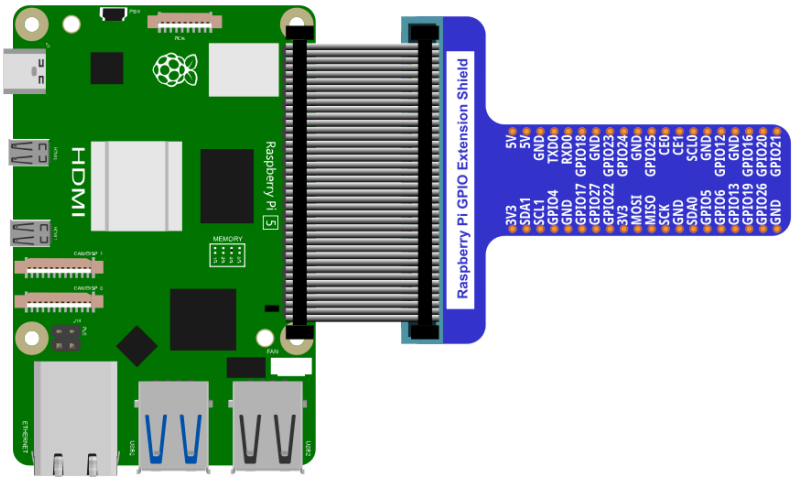
Breadboard

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.



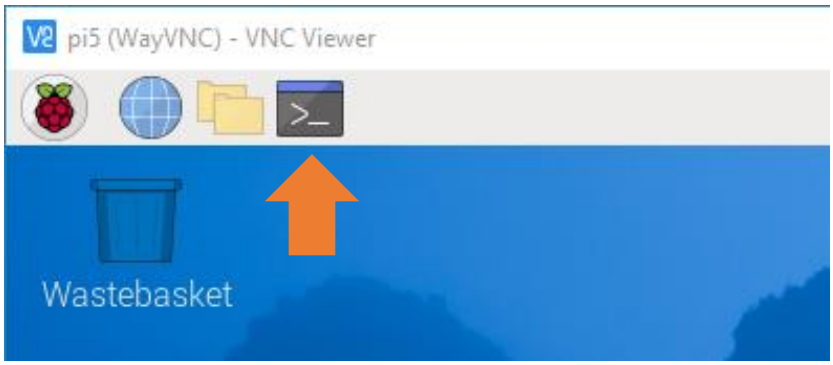
Sketch

According to the circuit, when the GPIO17 of Raspberry Pi output level is high, the LED turns ON. Conversely, when the GPIO17 Raspberry Pi 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.

Sketch_01_Blink

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_01_Blink
```



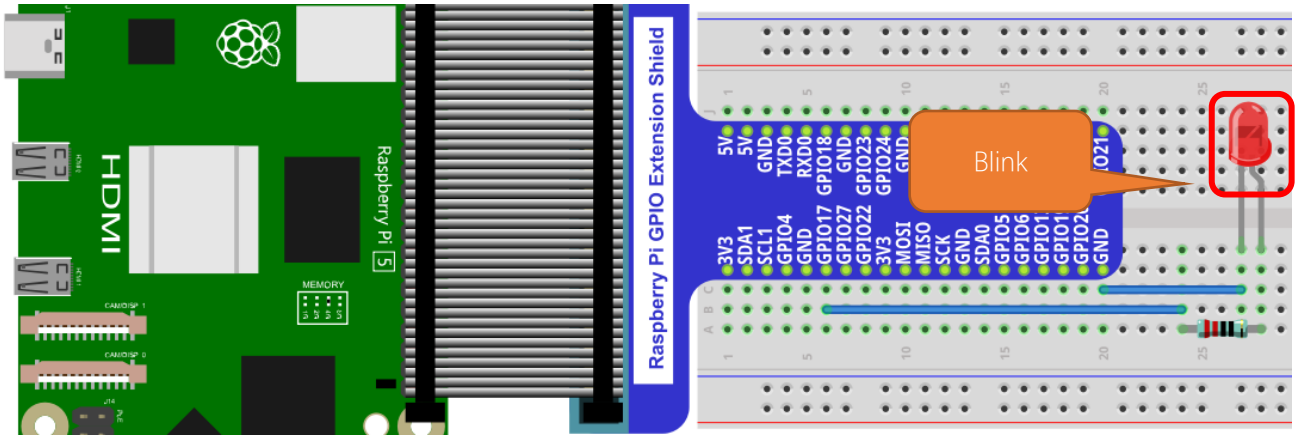

```
pi@raspberrypi:~ $ cd Freenove_Kit/Pi4j/Sketches/Sketch_01_Blink/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_01_Blink $
```

Enter the command to run the code.

```
jbang Blink.java
```

```
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_01_Blink $ jbang Blink.java
```

Upon the code runs, you can see the onboard blue LED blinks.



And on Raspberry Pi Terminal, you can see the corresponding messages printed.

```
[main] INFO com.pi4j.util.Console - LED low
[main] INFO com.pi4j.util.Console - LED high
[main] INFO com.pi4j.util.Console - LED low
[main] INFO com.pi4j.util.Console - LED high
```

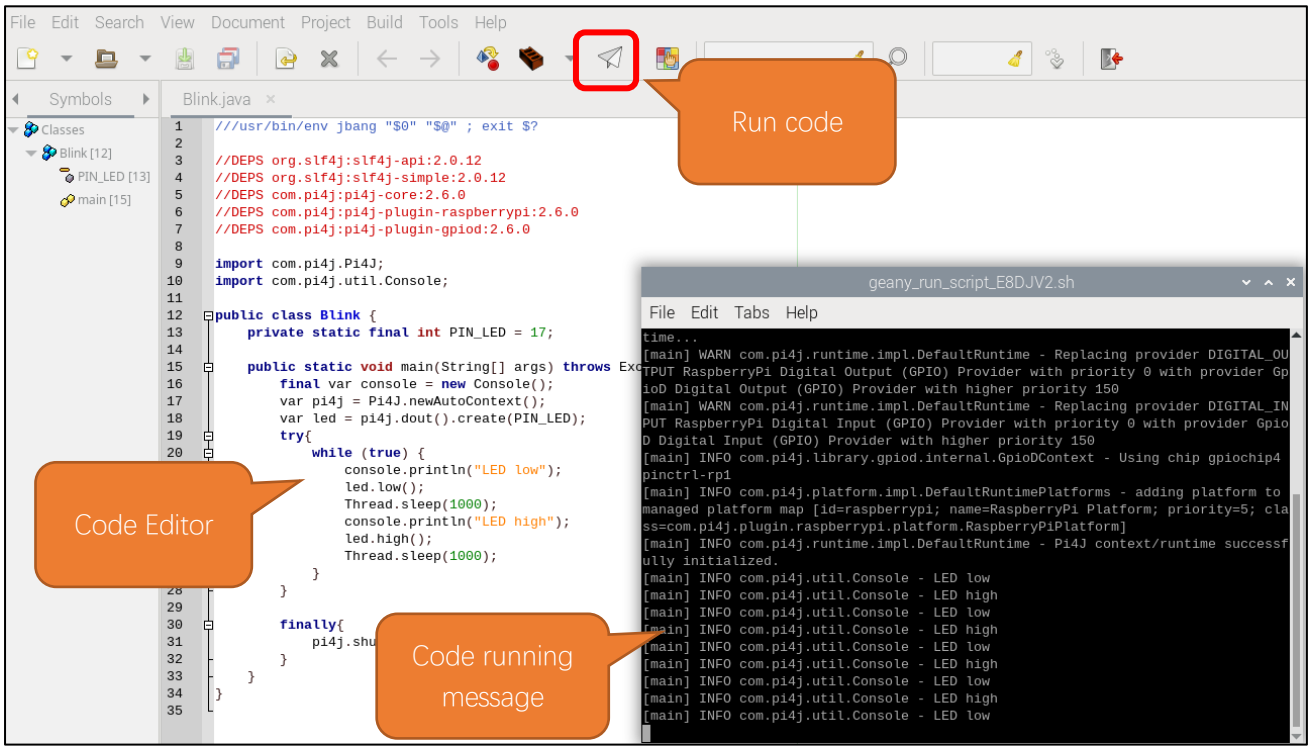
Press CTRL-C to exit the program.

```
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdow
n. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_01_Blink $
```

If you want to view or modify the code, you can open the code with Geany with the following command.

geany Blink.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
1   ///usr/bin/env jbang "$0" "$@" ; exit $?S
2
3   //DEPS org.slf4j:slf4j-api:2.0.12
4   //DEPS org.slf4j:slf4j-simple:2.0.12
5   //DEPS com.pi4j:pi4j-core:2.6.0
6   //DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
7   //DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
8
9   import com.pi4j.Pi4J;
10  import com.pi4j.util.Console;
11
12  public class Blink {
13      private static final int PIN_LED = 17;
14
15      public static void main(String[] args) throws Exception {
16          final var console = new Console();
17          var pi4j = Pi4J.newAutoContext();
18          var led = pi4j.dout().create(PIN_LED);
19          try{
20              while (true) {
```

```
                console.println("LED low");
                led.low();
                Thread.sleep(1000);
                console.println("LED high");
                led.high();
                Thread.sleep(1000);
            }
        }
        finally{
            pi4j.shutdown();
        }
    }
}
```

At the beginning of the code, we use shebang command to inform Raspberry Pi that we are going to use JBangto run Java script.

```
///usr/bin/env jbang "$0" "$@" ; exit $?
```

Likewise, we specify the dependencies required for the script to run. This includes the different components of the SLF4J logging library and the Pi4J library and their version numbers.

```
//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
```

Import Pi4J library and the Console class to facilitate creating and managing contexts and printing messages on the console.

```
import com.pi4j.Pi4J;
import com.pi4j.util.Console;
```

Assign the GPIO pin to the LED, and configure it as output mode.

```
   private static final int PIN_LED = 17;


    public static void main(String[] args) throws Exception {
        final var console = new Console();
        var pi4j = Pi4J.newAutoContext();
        var led = pi4j.dout().create(PIN_LED);
```

Use the try…finally structure to ensure the smooth running of the code.

```
        try{

        }
        finally{

        }
```

Make the LED turn on and off once every 1 second, repeat this process, and print prompt messages.

```
        while (true) {
            console.println("LED low"); // Print a prompt message on the terminal
            led.low();                  // turn off led
            Thread.sleep(1000);         // delay 1 second
            console.println("LED high"); // Print a prompt message on the terminal
            led.high();                  // turn on led
            Thread.sleep(1000);          // delay 1 second
        }
```

Chapter 2 Flowing Light

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

Project 2.1 Flowing Water Light

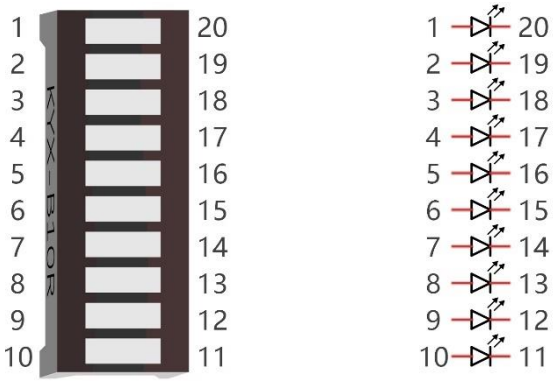
In this project, we use a number of LEDs to make a flowing water light.

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 11 		

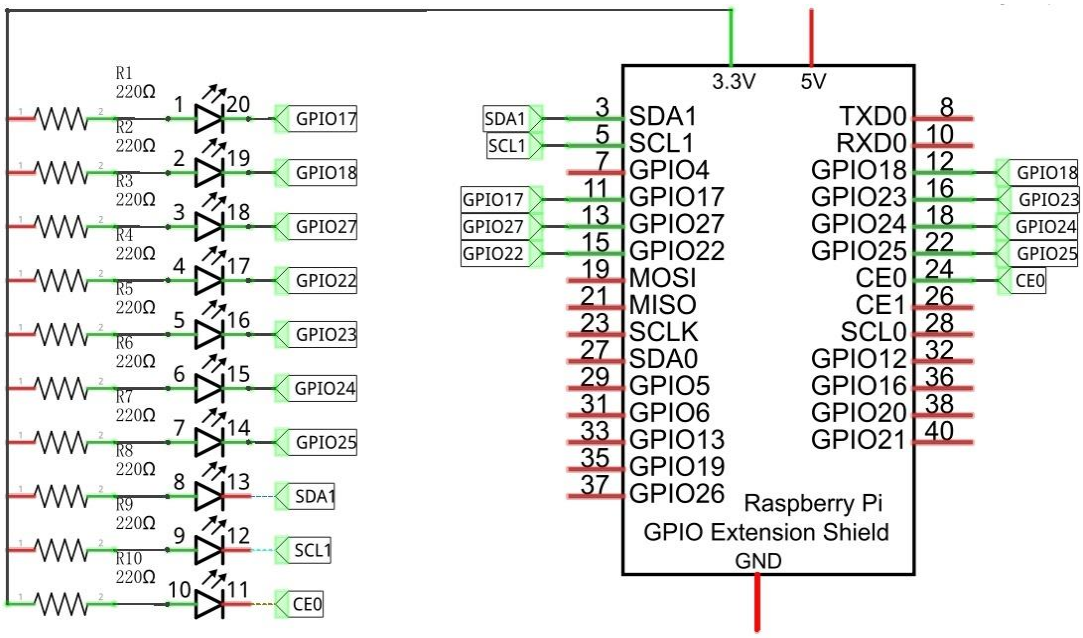
Bar Graph LED

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.

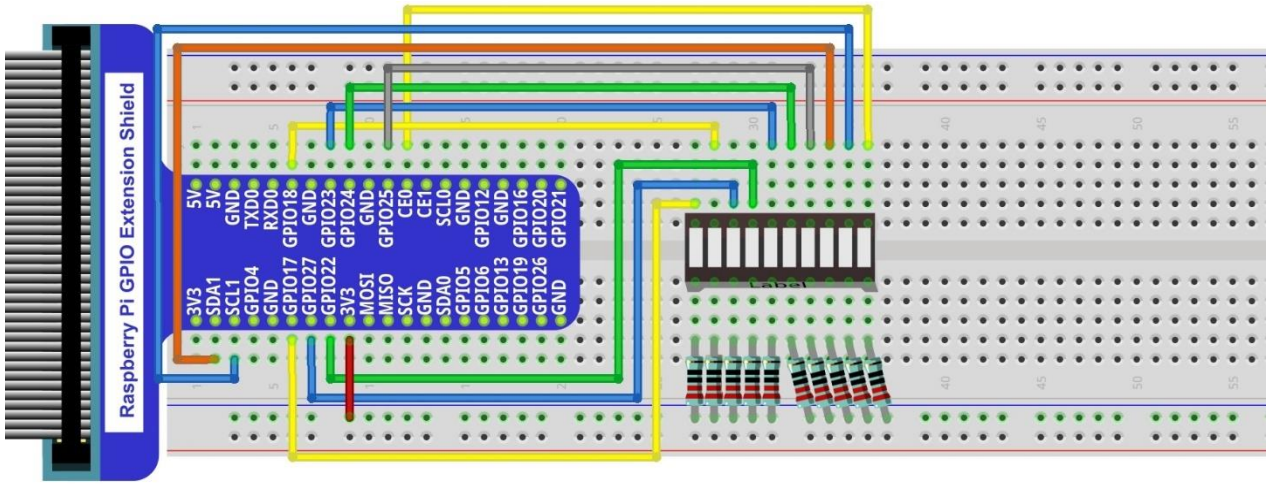


Circuit

Schematic diagram



Hardware connection.



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

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

If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we will introduce how to control multiple LEDs with various GPIOs, and make the LEDs present a flowing effect.

Sketch_02_FlowingLight

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_02_FlowingLight
```

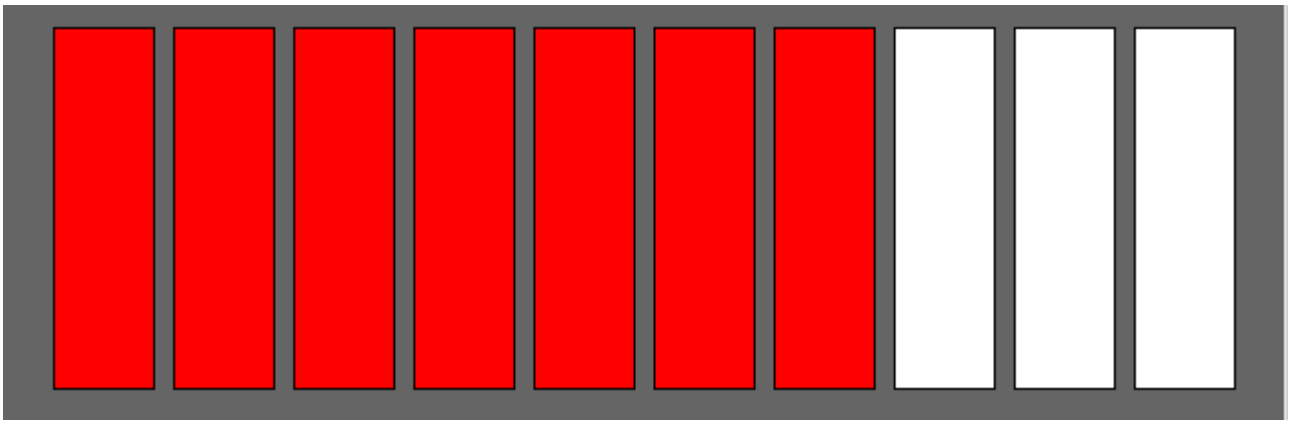
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_02_FlowingLight
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_02_FlowingLight $
```

Enter the command to run the code.

```
jbang FlowingLight.java
```

```
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_02_FlowingLight $ jbang FlowingLight.java
```

When the code is running, you can see the LedBar lights up in a flowing effect.



On the Raspberry Pi Terminal, you can see messages printed.

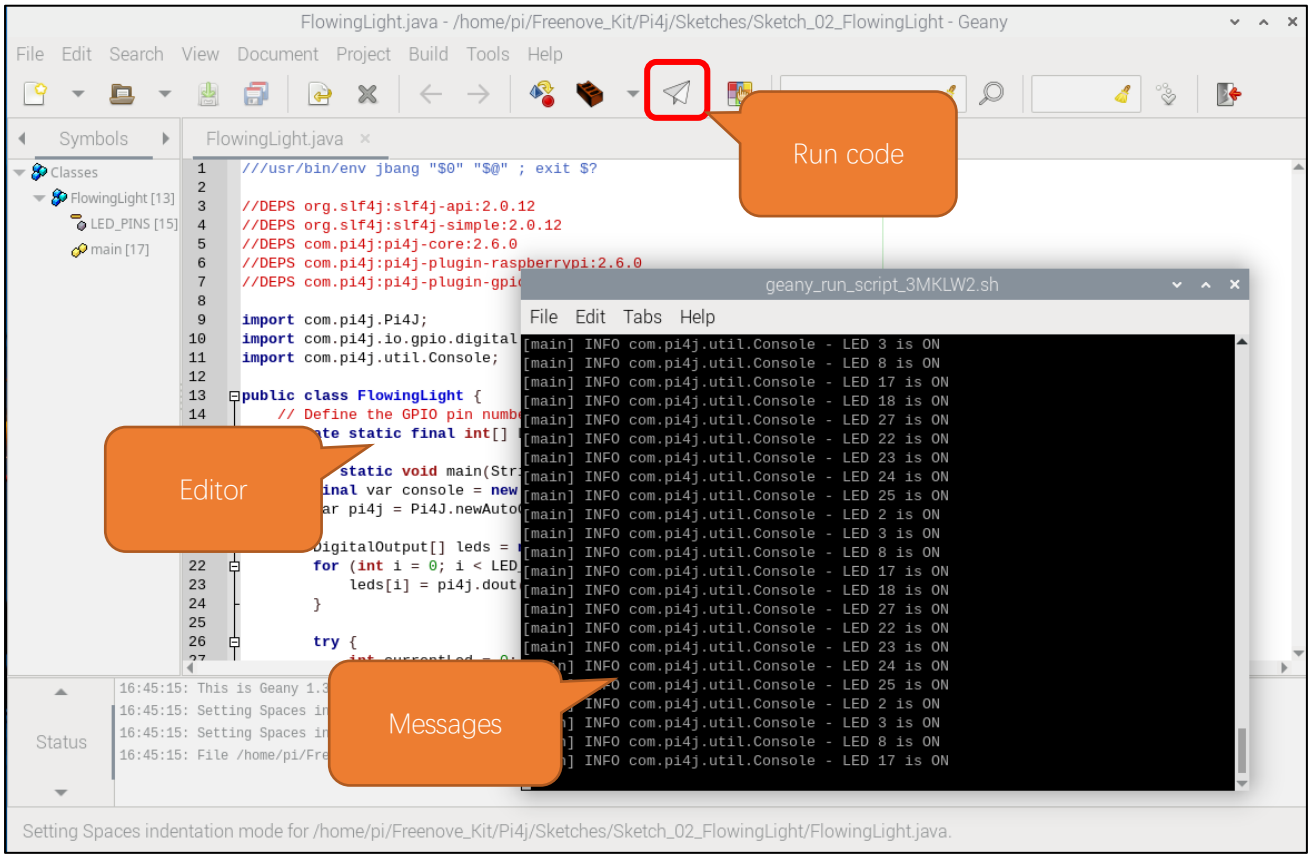
```
[main] INFO com.pi4j.util.Console - LED 17 is ON
[main] INFO com.pi4j.util.Console - LED 18 is ON
[main] INFO com.pi4j.util.Console - LED 27 is ON
[main] INFO com.pi4j.util.Console - LED 22 is ON
[main] INFO com.pi4j.util.Console - LED 23 is ON
[main] INFO com.pi4j.util.Console - LED 24 is ON
[main] INFO com.pi4j.util.Console - LED 25 is ON
[main] INFO com.pi4j.util.Console - LED 2 is ON
[main] INFO com.pi4j.util.Console - LED 3 is ON
[main] INFO com.pi4j.util.Console - LED 8 is ON
```

Press CTRL+C to exit the code.

You can view and edit the code with Geany by running the following command.

geany FlowingLight.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.io.gpio.digital.DigitalOutput;
import com.pi4j.util.Console;

public class FlowingLight {
    // Define the GPIO pin number for the LED.
    private static final int[] LED_PINS = {17, 18, 27, 22, 23, 24, 25, 2, 3, 8};

    public static void main(String[] args) throws Exception {
```

```
18          final var console = new Console();
19          var pi4j = Pi4J.newAutoContext();
20
21          DigitalOutput[] leds = new DigitalOutput[LED_PINS.length];
22          for (int i = 0; i < LED_PINS.length; i++) {
23              leds[i] = pi4j.dout().create(LED_PINS[i]);
24          }
25
26          try {
27              int currentLed = 0;
28              while (true) {
29                  console.println("LED " + LED_PINS[currentLed] + " is ON");
30                  for (DigitalOutput led : leds) {
31                      led.low();
32                  }
33                  leds[currentLed].high();
34                  Thread.sleep(100);
35                  currentLed = (currentLed + 1) % LED_PINS.length;
36              }
37          } finally {
38              pi4j.shutdown();
39          }
40      }
41  }
```

Import the classes of Pi4J library for GPIO control and simple console output.

```
import com.pi4j.Pi4J;
import com.pi4j.io.gpio.digital.DigitalOutput;
import com.pi4j.util.Console;
```

Define an array that includes the GPIO numbers connecting to LEDs.

```
    // Define the GPIO pin number for the LED.
    private static final int[] LED_PINS = {17, 18, 27, 22, 23, 24, 25, 2, 3, 8};
```

Create a DigitalOutput array based on the GPIO array that controls the LEDs, and create a DigitalOutput instance for each pin.

```
DigitalOutput[] leds = new DigitalOutput[LED_PINS.length];
        for (int i = 0; i < LED_PINS.length; i++) {
            leds[i] = pi4j.dout().create(LED_PINS[i]);
        }
```

Iterate through all LEDs and turn them off (set to low level).

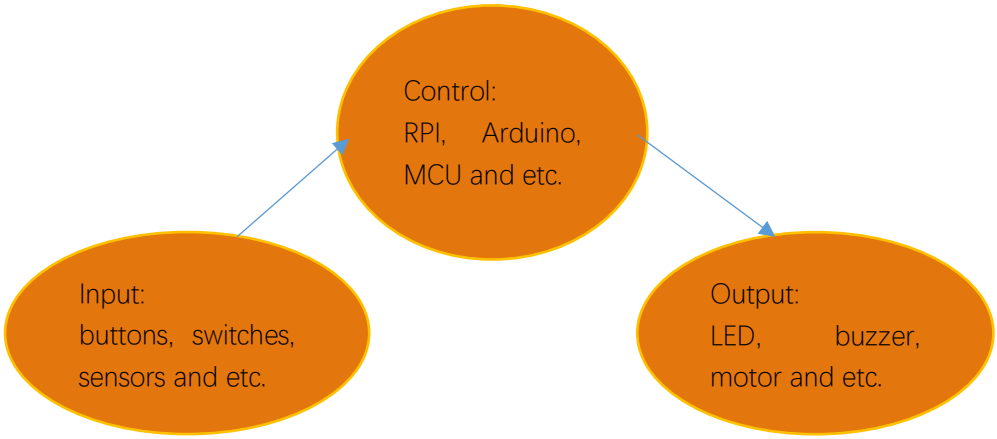
```
                for (DigitalOutput led : leds) {
                    led.low();
                }
```

Use ‘currentLed’ to record the position of the LED that is lit, recalculate the position of the lit LED every 100 milliseconds, and print a prompt message to the console. At the same time, turn off all LEDs except the LED at the position recorded by ‘currentLed’.

```
            int currentLed = 0;
            while (true) {
                console.println("LED " + LED_PINS[currentLed] + " is ON");
                for (DigitalOutput led : leds) {
                    led.low();
                }
                leds[currentLed].high();
                Thread.sleep(100);
                currentLed = (currentLed + 1) % LED_PINS.length;
            }
```

Chapter 3 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 3.1 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

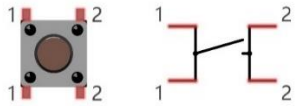
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push Button Switch x1 
Jumper Wire 				

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

Component knowledge

Push Button Switch

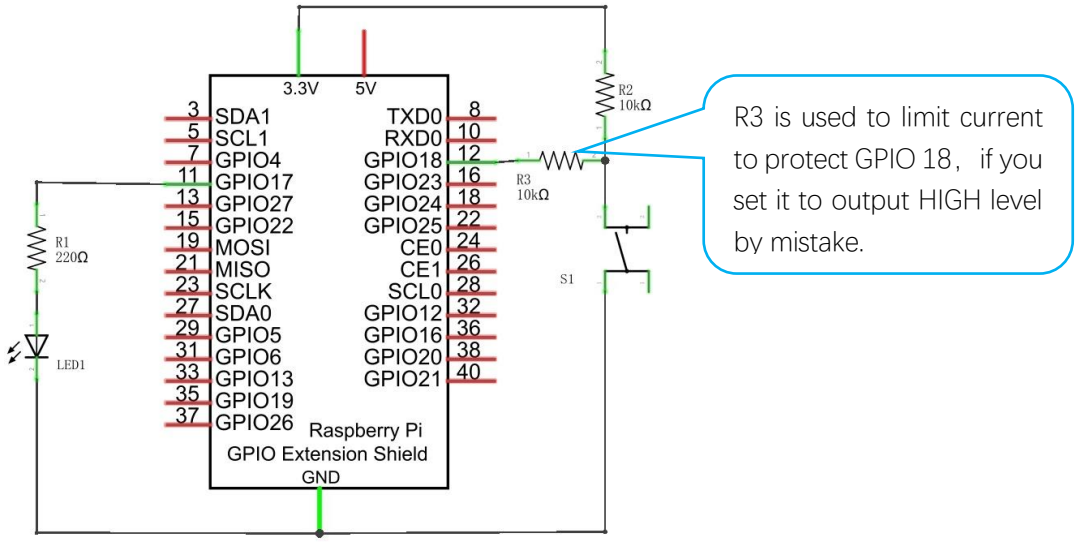
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



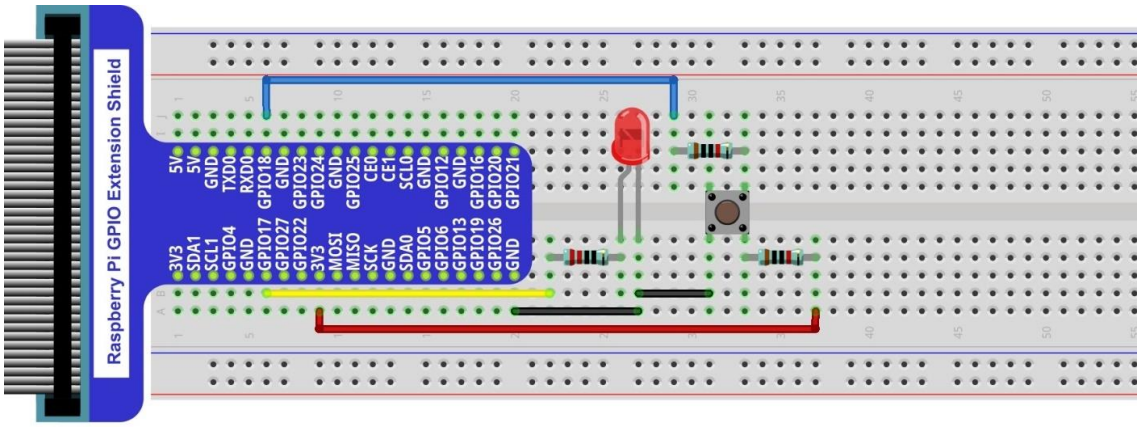
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we will introduce how to use the button to control the LED.

Sketch_03_ButtonLED

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_03_ButtonLED
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_03_ButtonLED
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_03_ButtonLED $
```

Enter the command to run the code.

```
jbang ButtonLED.java
```

```
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_03_ButtonLED $ jbang ButtonLED.java
```

When the code is running, press the button and you can see the LED is lit; release it and the LED goes off.

On the Raspberry Pi terminal, you will see the corresponding messages printed.

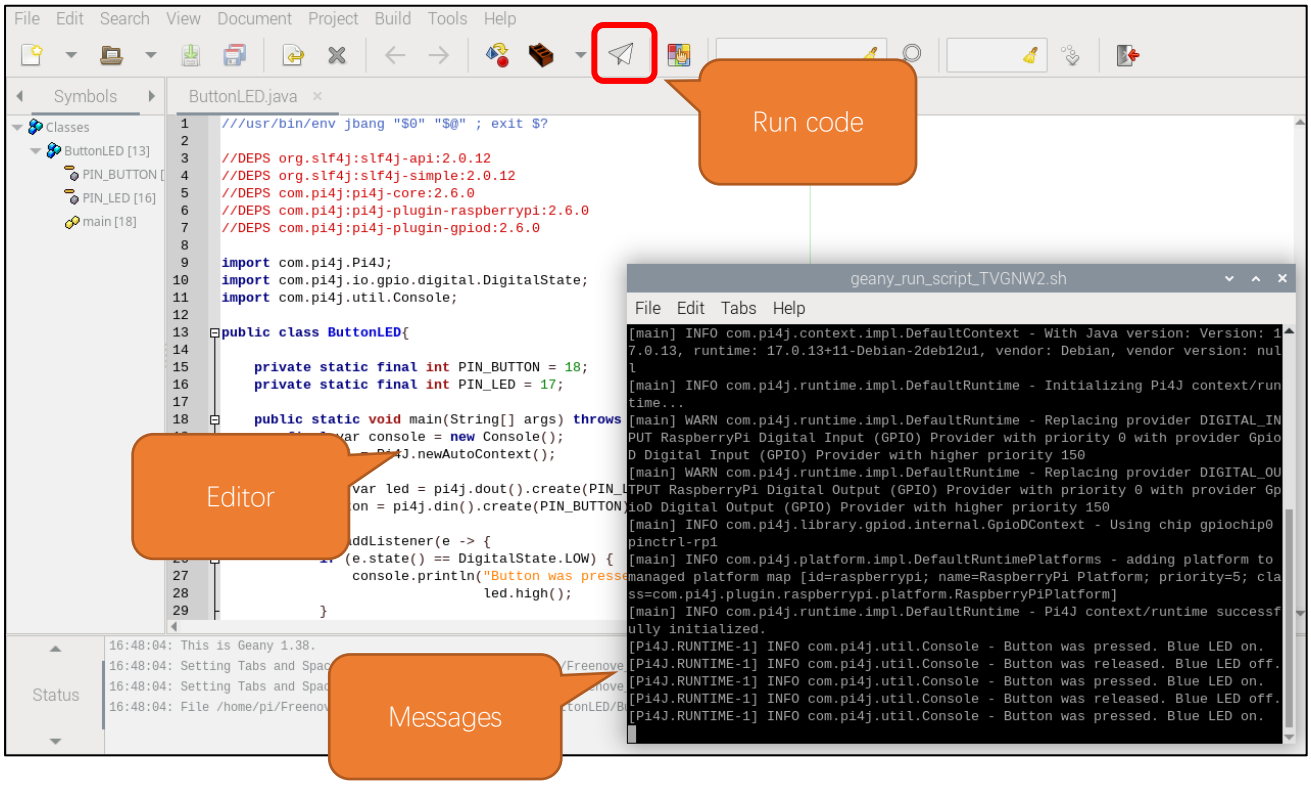
```
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was pressed. Blue LED on.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was released. Blue LED off.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was pressed. Blue LED on.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was released. Blue LED off.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was pressed. Blue LED on.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was released. Blue LED off.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was pressed. Blue LED on.
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - Button was released. Blue LED off.
```

Press Ctrl+C to exit the code.

You can open the code with geany to view and edit it.

geany ButtonLED.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.io.gpio.digital.DigitalState;
import com.pi4j.util.Console;

public class ButtonLED{

    private static final int PIN_BUTTON = 18;
    private static final int PIN_LED = 17;

    public static void main(String[] args) throws Exception {
```

```
19          final var console = new Console();
20          var pi4j = Pi4J.newAutoContext();
21
22          var led = pi4j.dout().create(PIN_LED);
23          var button = pi4j.din().create(PIN_BUTTON);
24
25          button.addListener(e -> {
26              if (e.state() == DigitalState.LOW) {
27                  console.println("Button was pressed. Blue LED on.");
28                  led.high();
29              }
30              else if(e.state() == DigitalState.HIGH){
31                  console.println("Button was released. Blue LED off.");
32                  led.low();
33              }
34          });
35
36          try{
37              while (true) {
38                  Thread.sleep(500);
39              }
40          }
41          finally{
42              pi4j.shutdown();
43          }
44      }
45  }
```

Import the classes of Pi4J library for GPIO control and simple console output.

```
import com.pi4j.Pi4J;
import com.pi4j.io.gpio.digital.DigitalState;
import com.pi4j.util.Console;
```

Define the GPIO numbers for the button and LED.

```
    private static final int PIN_BUTTON = 18;
    private static final int PIN_LED = 17;
```

Create a Console instance for printing logs or messages.
Create a Pi4J context to manage the GPIO interface.
Create an LED output pin object, connected to the pin specified by PIN_LED.
Create a button input pin object, connected to the pin specified by PIN_BUTTON.

```
final var console = new Console();
var pi4j = Pi4J.newAutoContext();
var led = pi4j.dout().create(PIN_LED);
var button = pi4j.din().create(PIN_BUTTON);
```

Add a listener event to the button, which is triggered when the button's state changes.
When the button is pressed, its state is low, and we control the LED to light up.
When the button is released, its state is high, and we control the LED to turn off.

```
button.addListener(e -> {
    if (e.state() == DigitalState.LOW) {
        console.println("Button was pressed. Blue LED on.");
        led.high();
    }
    else if(e.state() == DigitalState.HIGH){
        console.println("Button was released. Blue LED off.");
        led.low();
    }
});
```

Nothing needs to be done in the main loop. Just set it in an infinite loop and ensure that the Pi4J context is closed when the program ends.

```
try{
    while (true) {
        Thread.sleep(500);
    }
}
finally{
    pi4j.shutdown();
}
```

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.

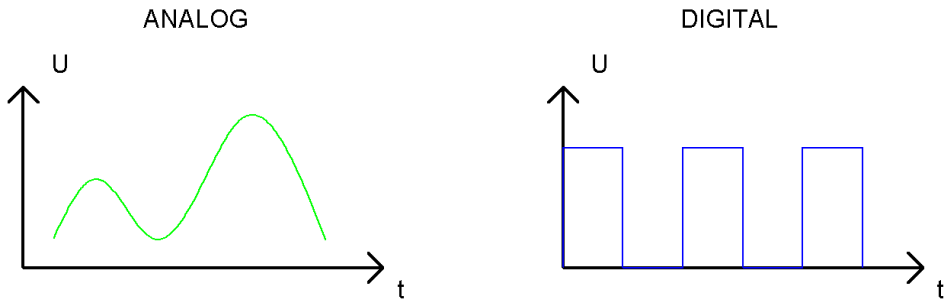
Component List

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

Component Knowledge

Analog & Digital

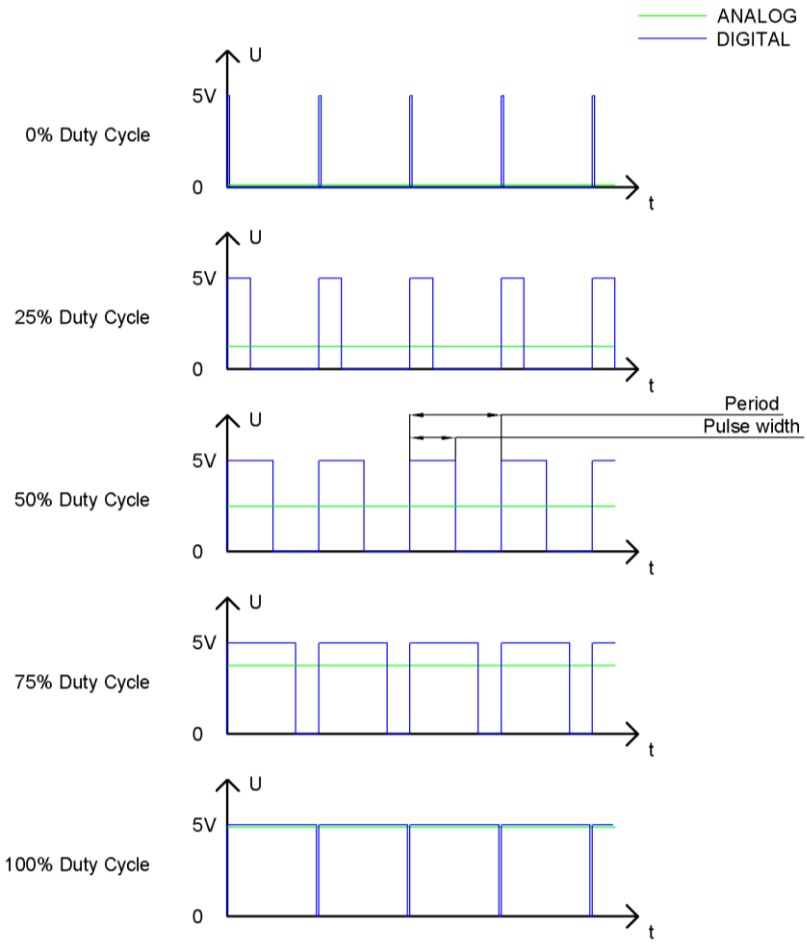
An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal **or discrete-time signal is a time series consisting of a sequence of quantities**. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout 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.

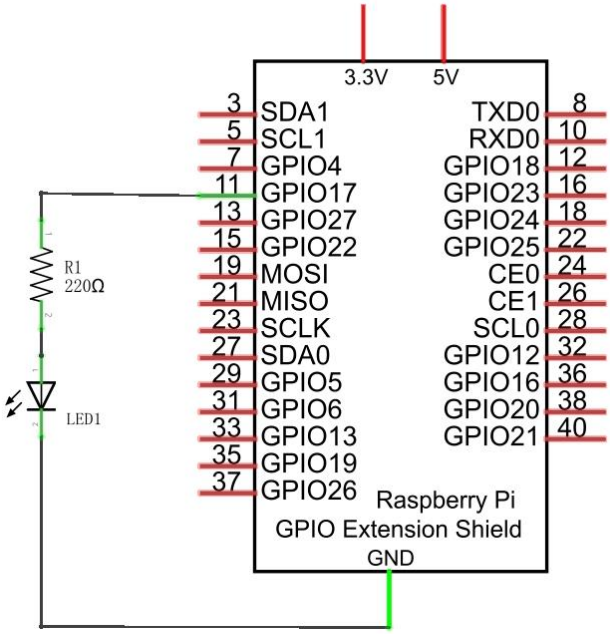
The wiringPi library of C provides both a hardware PWM and a software PWM method.

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.

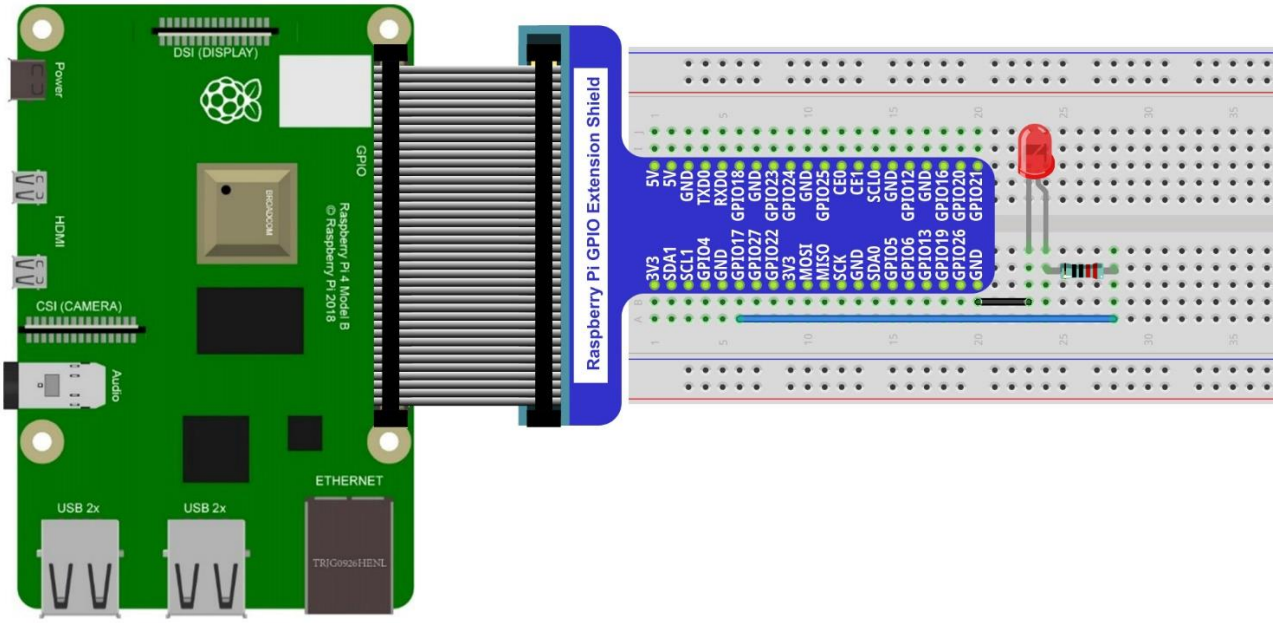
In order to keep the results running consistently, we will use PWM.

Circuit

Schematic diagram



Hardware connection



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we will learn how to make the LED to present a breathing effect.

Sketch_04_BreathingLED

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED

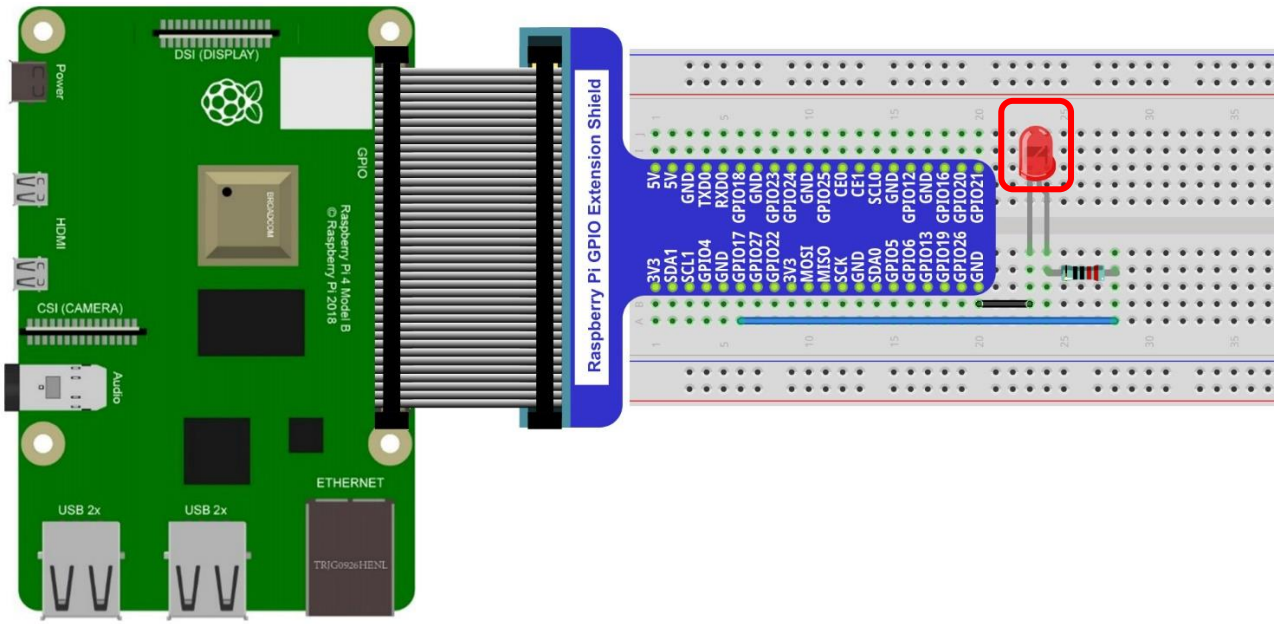
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED $
```

You can enter the command to run the code.

jbang BreathingLED.java

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED $ jbang BreathingLED.java
```

When the code is running, you can see the LED lights up from dim to bright and then from bright to dim, and the process repeats.



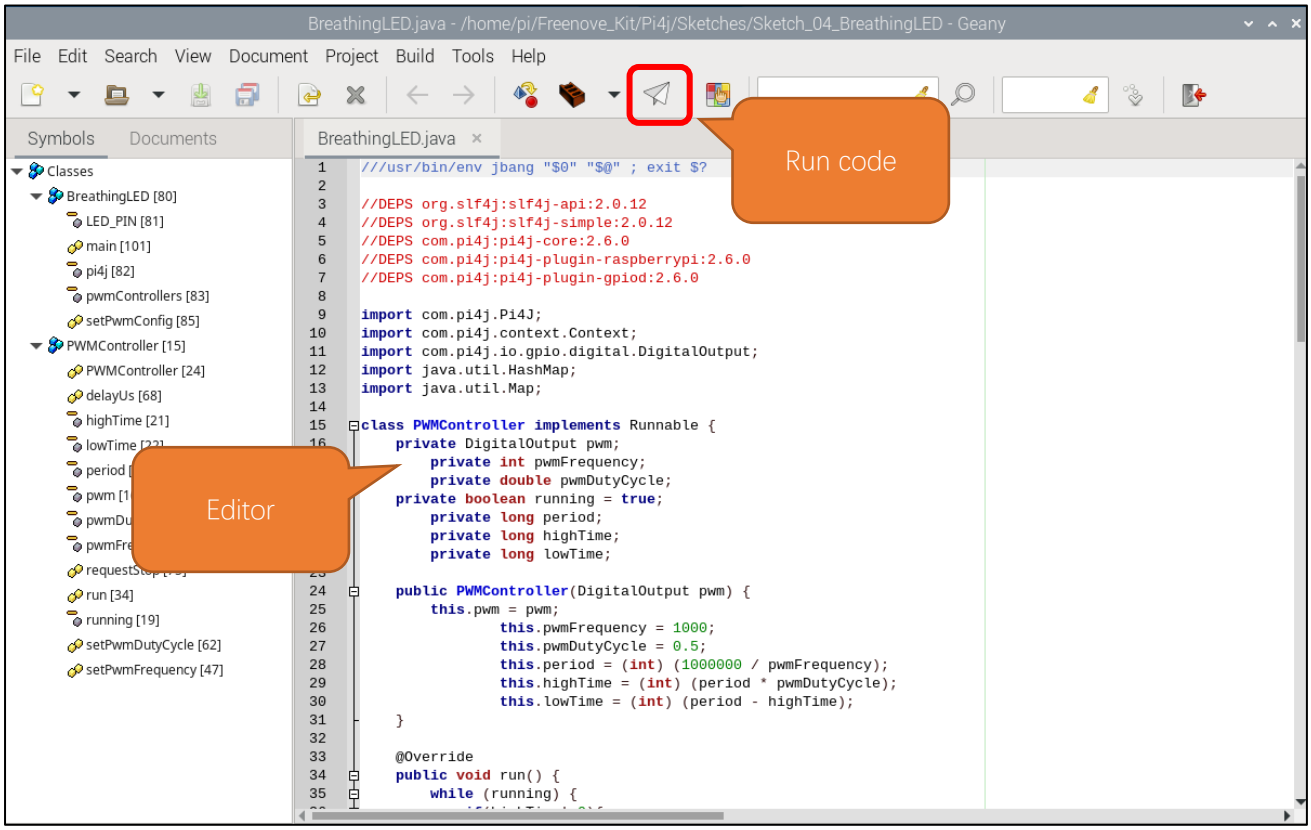
Press Ctrl-C to exit the code.

```
[main] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully initialized.
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime.
..
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shu
tdown. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_04_BreathingLED $
```

You can open the code with Geany to view and edit it.

geany BreathingLED.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalOutput;
import java.util.HashMap;
import java.util.Map;

class PWMController implements Runnable {
    private DigitalOutput pwm;
    private int pwmFrequency;
```

```
18      private double pwmDutyCycle;
19      private boolean running = true;
20      private long period;
21      private long highTime;
22      private long lowTime;
23
24      public PWMController(DigitalOutput pwm) {
25          this.pwm = pwm;
26          this.pwmFrequency = 1000;
27          this.pwmDutyCycle = 0.5;
28          this.period = (int) (1000000 / pwmFrequency);
29          this.highTime = (int) (period * pwmDutyCycle);
30          this.lowTime = (int) (period - highTime);
31      }
32
33      @Override
34      public void run() {
35          while (running) {
36              if(highTime!=0){
37                  pwm.high();
38                  delayUs(highTime);
39              }
40              if(lowTime!=0){
41                  pwm.low();
42                  delayUs(lowTime);
43              }
44          }
45      }
46
47      public void setPwmFrequency(int frequency) {
48          if(frequency!=0){
49              this.pwmFrequency = frequency;
50              this.period = (int) (1000000 / pwmFrequency);
51              this.highTime = (int) (period * pwmDutyCycle);
52              this.lowTime = (int) (period - highTime);
53          }
54          else{
55              this.pwmFrequency = 0;
56              this.period = (int) (1000);
57              this.highTime = (int) (0);
58              this.lowTime = (int) (period - highTime);
59          }
60      }
61
```

```
62      public void setPwmDutyCycle(double dutyCycle) {
63          this.pwmDutyCycle = dutyCycle;
64          this.highTime = (int) (period * pwmDutyCycle);
65          this.lowTime = (int) (period - highTime);
66      }
67
68      private void delayUs(long us) {
69          long startTime = System.nanoTime();
70          long endTime = startTime + (us * 1000);
71          while (System.nanoTime() < endTime) {
72          }
73      }
74
75      public void requestStop() {
76          running = false;
77      }
78  }
79
80  public class BreathingLED {
81      private static int   LED_PIN = 17;
82      private static final Context pi4j = Pi4J.newAutoContext();
83      private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
84
85      public static void setPwmConfig(int pin) throws Exception {
86          DigitalOutput led = pi4j.dout().create(pin);
87          PWMController = new PWMController(led);
88          Thread pwmThread = new Thread(pwmController, "PWM LED Controller " + pin);
89          pwmControllers.put(pin, pwmController);
90          pwmThread.start();
91          Runtime.getRuntime().addShutdownHook(new Thread(() -> {
92              pwmController.requestStop();
93              try {
94                  pwmThread.join();
95              } catch (InterruptedException e) {
96                  Thread.currentThread().interrupt();
97              }
98          }));
99      }
100
101     public static void main(String[] args) throws Exception {
102         setPwmConfig(LED_PIN);
103         PWMController BreathingLed = pwmControllers.get(LED_PIN);
104         try {
105                 while (true) {
```

```
106                     double dutyCycle = 0.01;
107                     while (dutyCycle < 1) {
108                         BreathingLed.setPwmDutyCycle(dutyCycle);
109                         dutyCycle += 0.01;
110                         Thread.sleep(10);
111                     }
112                     while (dutyCycle > 0) {
113                         BreathingLed.setPwmDutyCycle(dutyCycle);
114                         dutyCycle -= 0.01;
115                         Thread.sleep(10);
116                     }
117                 }
118             }
119         finally {
120             for (PWMController controller : pwmControllers.values()) {
121                 controller.requestStop();
122             }
123             pi4j.shutdown();
124         }
125     }
126 }
```

Use JBange to run the script and automatically process the declared dependencies.

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
```

Import Pi4J library, context management, digital output interface, HashMap class, and Map interface.

```
import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalOutput;
import java.util.HashMap;
import java.util.Map;
```

Pi4j only has 4 hardware PWM pins, and the use of these pins is greatly limited, so we use software PWM to control the LED. Although software PWM is not as precise as hardware PWM, it can be applied to any GPIO.
To implement software PWM, we have written a PWMController class to control the GPIO output of PWM. Since PWMController implements the Runnable interface, it can be used to create a Thread object, passing an instance of PWMController as the target to the Thread constructor, and then starting this thread. In this way, the run method of PWMController will be executed in a new thread, allowing it to perform PWM control tasks concurrently.

```
class PWMController implements Runnable {
.....
}
```

Define variables to configure PWM's parameters.

```
    private DigitalOutput pwm;
    private int pwmFrequency;
    private double pwmDutyCycle;
    private boolean running = true;
    private long period;
    private long highTime;
    private long lowTime;
```

Constructor, to initialize various parameters of the PWM controller.

```
    public PWMController(DigitalOutput pwm) {
        this.pwm = pwm;
        this.pwmFrequency = 1000;
        this.pwmDutyCycle = 0.5;
        this.period = (int) (1000000 / pwmFrequency);
        this.highTime = (int) (period * pwmDutyCycle);
        this.lowTime = (int) (period - highTime);
    }
```

This is a simple delay method in microsecond.

```
    private void delayUs(long us) {
        long startTime = System.nanoTime();
        long endTime = startTime + (us * 1000);
        while (System.nanoTime() < endTime) {
        }
    }
```

The functions to set PWM frenqunency.

```
    public void setPwmFrequency(int frequency) {
        if(frequency!=0){
            this.pwmFrequency = frequency;
            this.period = (int) (1000000 / pwmFrequency);
            this.highTime = (int) (period * pwmDutyCycle);
            this.lowTime = (int) (period - highTime);
        }
        else{
            this.pwmFrequency = 0;
            this.period = (int) (1000);
            this.highTime = (int) (0);
            this.lowTime = (int) (period - highTime);
        }
    }
```

The functions to set PWM duty cycle.

```
    public void setPwmDutyCycle(double dutyCycle) {
        this.pwmDutyCycle = dutyCycle;
        this.highTime = (int) (period * pwmDutyCycle);
        this.lowTime = (int) (period - highTime);
    }
```

@Override means that the run method overrides the method in the parent class or implements the interface. This method allows the pin to output PWM signals.

```
    @Override
    public void run() {
        while (running) {
            if(highTime!=0){
                pwm.high();
                delayUs(highTime);
            }
            if(lowTime!=0){
                pwm.low();
                delayUs(lowTime);
            }
        }
    }
```

Request to stop the PWM controller. The PWMController class is used in the thread. When running is false, exit the thread.

```
    public void requestStop() {
        running = false;
    }
```

Define the pins that control PWM and create an integer Map variable pwmControllers to store the mapping of PWMController objects.

```
    private static int   LED_PIN = 17;
    private static final Context pi4j = Pi4J.newAutoContext();
    private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
```

The PWM configuration function only needs to fill in the GPIO number in the parameter. The code will automatically apply for a PWMController object and create a corresponding thread to run it. At the same time, the PWMController is stored in pwmControllers and then the thread is started.

```
    public static void setPwmConfig(int pin) throws Exception {
        DigitalOutput led = pi4j.dout().create(pin);
        PWMController pwmController = new PWMController(led);
        Thread pwmThread = new Thread(pwmController, "PWM LED Controller " + pin);
        pwmControllers.put(pin, pwmController);
        pwmThread.start();
        ......
    }
```

Add JVM shutdown hook to ensure PWM controller is stopped on JVM shutdown.

```
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            pwmController.requestStop();
            try {
                pwmThread.join();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }));
```

The main program, which is used to test the effect of making the LED breathe.

```
    public static void main(String[] args) throws Exception {
        ......
    }
```

Configure PWM and associate it with pin.

```
        setPwmConfig(LED_PIN);
```

Obtain PWM controller corresponding to the pin.

```
        PWMController BreathingLed = pwmControllers.get(LED_PIN);
```

The duty cycle value is changed every 10 milliseconds, so that the LED cycles from dark to bright and then from bright to dark.

```
        try {
                while (true) {
                    double dutyCycle = 0.01;
                    while (dutyCycle < 1) {
                        BreathingLed.setPwmDutyCycle(dutyCycle);
                        dutyCycle += 0.01;
                        Thread.sleep(10);
                    }
                    while (dutyCycle > 0) {
                        BreathingLed.setPwmDutyCycle(dutyCycle);
                        dutyCycle -= 0.01;
                        Thread.sleep(10);
                    }
                }
            }
```

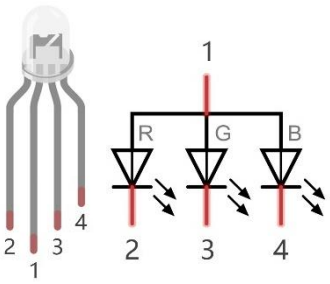
When exiting the main loop, first close all PWM controller threads and then close the pi4j context.

```
        finally {
            for (PWMController controller : pwmControllers.values()) {
                controller.requestStop();
            }
            pi4j.shutdown();
        }
```

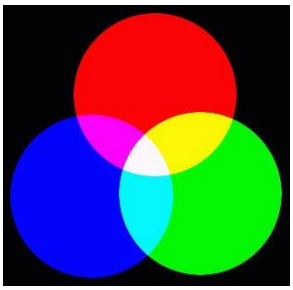
Chapter 5 RGB LED

In this chapter, we will learn how to control an 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 an 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 RainbowLED

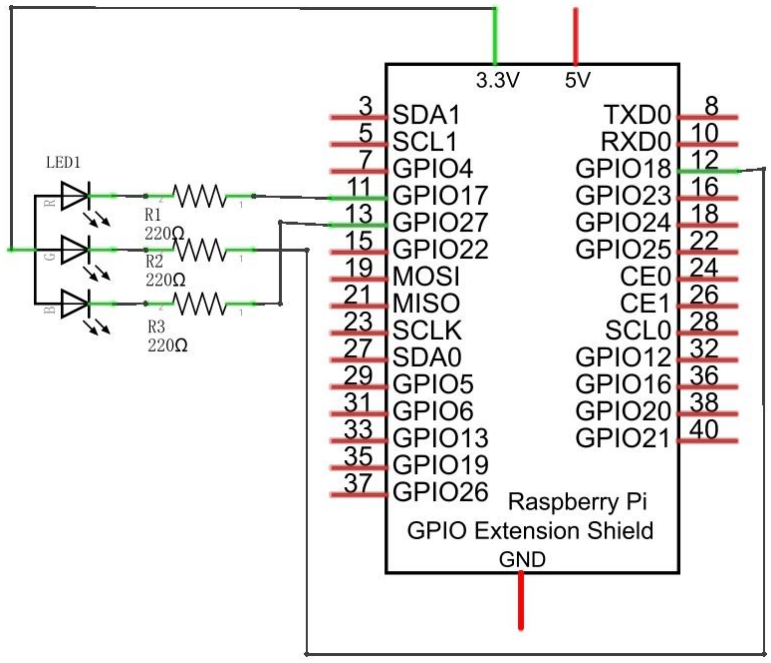
In this project, we will make a multicolored LED, which we can program the RGB LED to automatically change colors.

Component List

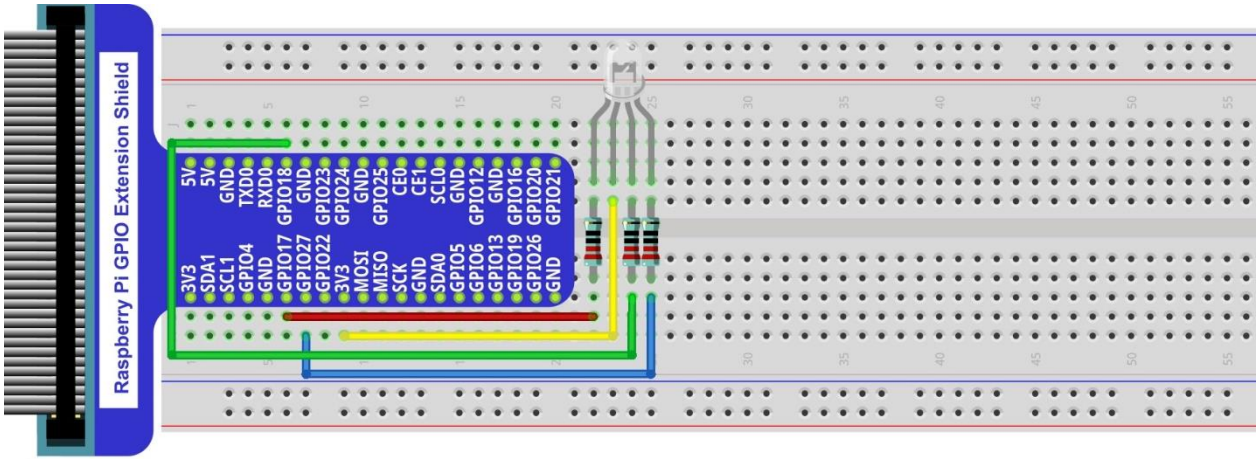
Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	RGBLED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

Circuit

Schematic diagram



Hardware connection



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we will control the RGB LED with 3 PWMs.

Sketch_05_RainbowLED

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED

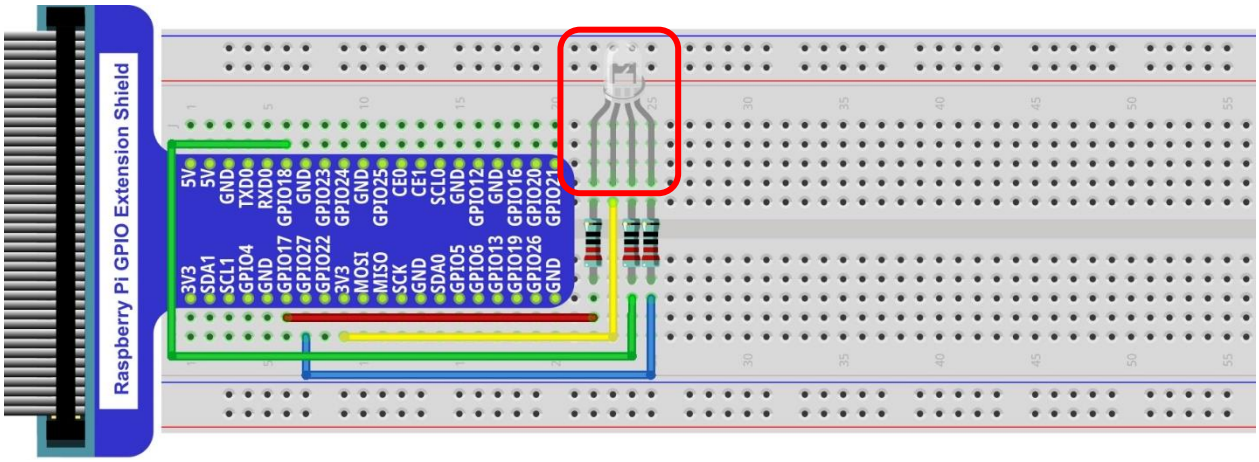
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED $
```

Enter the command to run the code.

jbang RainbowLED.java

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED $ jbang RainbowLED.java
```

When the code is running, the RGB LED randomly emits various colors.




```
[main] INFO com.pi4j.util.Console - R:69  G:68, B:10
[main] INFO com.pi4j.util.Console - R:80  G:33, B:96
[main] INFO com.pi4j.util.Console - R:43  G:33, B:87
[main] INFO com.pi4j.util.Console - R:28  G:17, B:98
[main] INFO com.pi4j.util.Console - R:16  G:33, B:55
[main] INFO com.pi4j.util.Console - R:81  G:88, B:53
[main] INFO com.pi4j.util.Console - R:45  G:98, B:82
[main] INFO com.pi4j.util.Console - R:52  G:73, B:45
```

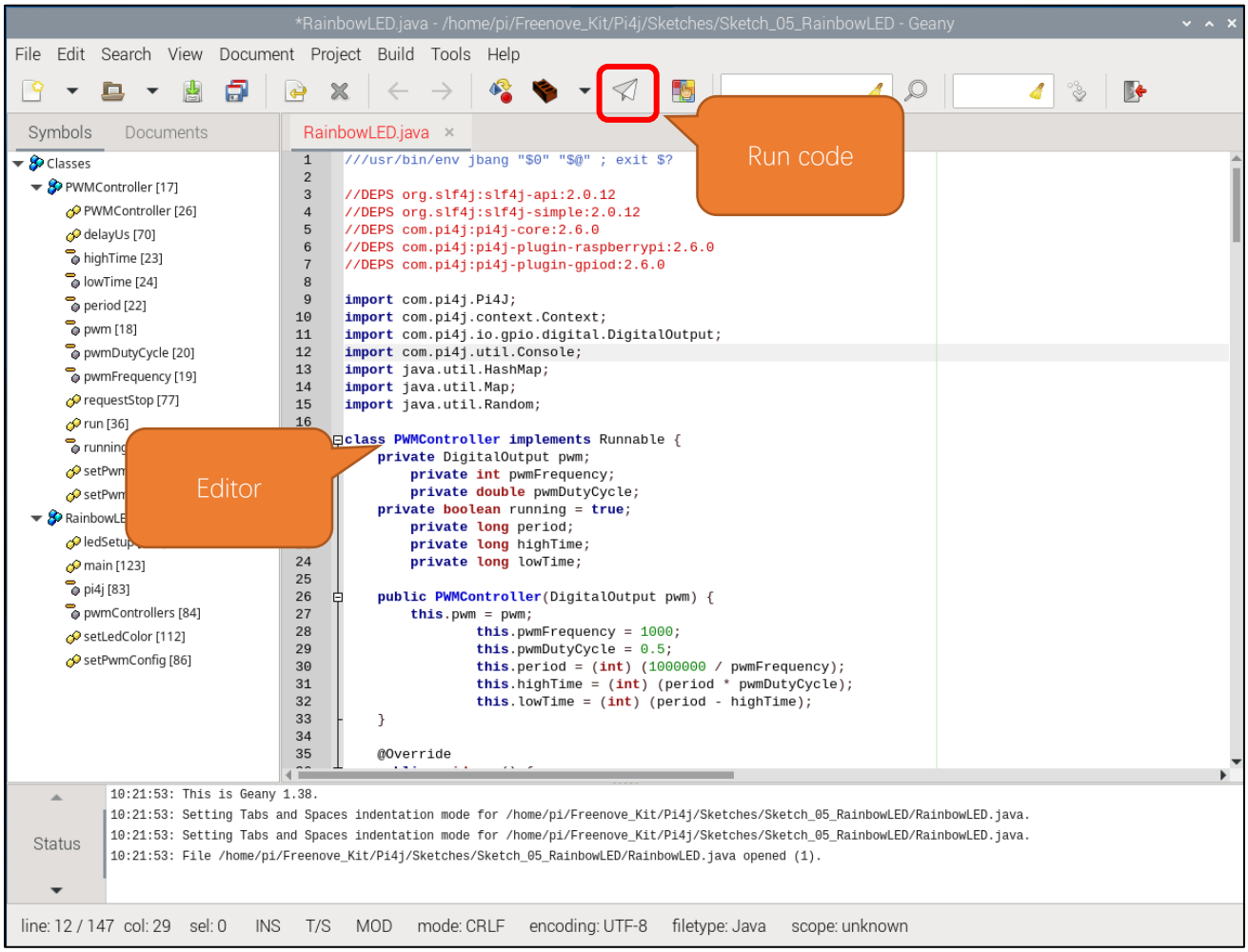
Press Ctr+C to exit the program.

```
[main] INFO com.pi4j.util.Console - R:60  G:6, B:66
[main] INFO com.pi4j.util.Console - R:69  G:87, B:89
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime
...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully sh
utdown. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_05_RainbowLED $
```

You can open the code with Geany to view and edit it, with the following command.

```
geany RainbowLED.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalOutput;
import com.pi4j.util.Console;
import java.util.HashMap;
```

```
14  import java.util.Map;
15  import java.util.Random;
16
17  class PWMController implements Runnable {
…       ......
80  }
81
82  public class RainbowLED {
83      private static final Context pi4j = Pi4J.newAutoContext();
84      private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
85
86      public static void setPwmConfig(int pin) throws Exception {
…           ......
100     }
101
102     public static void ledSetup(int[] pins) {
103         for (int pin : pins) {
104             try {
105                 setPwmConfig(pin);
106             } catch (Exception e) {
107                 e.printStackTrace();
108             }
109         }
110     }
111
112     public static void setLedColor(int[] pins, int r, int g, int b) {
113         if (pins.length >= 3) {
114             PWMController red = pwmControllers.get(pins[0]);
115             PWMController green = pwmControllers.get(pins[1]);
116             PWMController blue = pwmControllers.get(pins[2]);
117             if (red != null) red.setPwmDutyCycle((double) r / 100.0);
118             if (green != null) green.setPwmDutyCycle((double) g / 100.0);
119             if (blue != null) blue.setPwmDutyCycle((double) b / 100.0);
120         }
121     }
122
123     public static void main(String[] args) throws Exception {
124         final var console = new Console();
125         int[] LED_PINS = {17, 18, 27};
126         Random = new Random();
127
128         try {
129             ledSetup(LED_PINS);
130             while (true) {
```

```
131                 int red = random.nextInt(101);
132                 int green = random.nextInt(101);
133                 int blue = random.nextInt(101);
134                 console.println("R:%d  G:%d, B:%d", red, green, blue);
134                 setLedColor(LED_PINS, red, green, blue);
135                 Thread.sleep(500);
136             }
137         }
138         finally {
139             for (PWMController controller : pwmControllers.values()) {
140                 controller.requestStop();
141             }
142             pi4j.shutdown();
143         }
144     }
145 }
```

Import Pi4j library, context management, digital output interface, HashMap class and Map interface, and random function library.

```
import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalOutput;
import com.pi4j.util.Console;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
```

Initialize the pins corresponding to the RGB lights and print a stack trace if an exception occurs.

```
    public static void ledSetup(int[] pins) {
        for (int pin : pins) {
            try {
                setPwmConfig(pin);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Configure the RGB colored lights and adjust the brightness of the LED by adjusting the duty cycle of PWM.

```
    public static void setLedColor(int[] pins, int r, int g, int b) {
        if (pins.length >= 3) {
            PWMController red = pwmControllers.get(pins[0]);
            PWMController green = pwmControllers.get(pins[1]);
            PWMController blue = pwmControllers.get(pins[2]);
            if (red != null) red.setPwmDutyCycle((double) r / 100.0);
            if (green != null) green.setPwmDutyCycle((double) g / 100.0);
            if (blue != null) blue.setPwmDutyCycle((double) b / 100.0);
        }
    }
```

Create a console instance, define an array of GPIO pins for LED connections, and create a random number generator instance.

```
        final var console = new Console();
        int[] LED_PINS = {17, 18, 27};
        Random = new Random();
```

Initialize the array of GPIO pins connected to the LEDs, generate 3 new random numbers every 500 milliseconds as brightness values for the RGB lights, and print prompt messages on the console.

```
        try {
            ledSetup(LED_PINS);
            while (true) {
                int red = random.nextInt(101);
                int green = random.nextInt(101);
                int blue = random.nextInt(101);
                console.println("R:%d  G:%d, B:%d", red, green, blue);
                setLedColor(LED_PINS, red, green, blue);
                Thread.sleep(500);
            }
        }
```

At the end of the program, stop all PWM controllers and close the Pi4J context.

```
        finally {
            for (PWMController controller : pwmControllers.values()) {
                controller.requestStop();
            }
            pi4j.shutdown();
        }
```

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

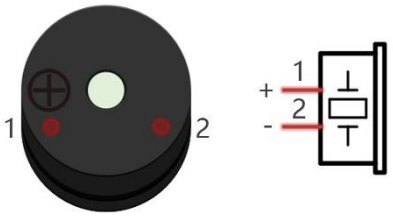
<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push Button Switch x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ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

Component knowledge

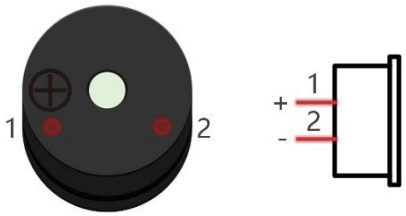
Buzzer

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

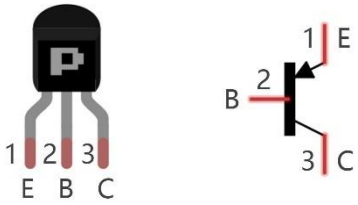
Passive buzzer bottom

Transistors

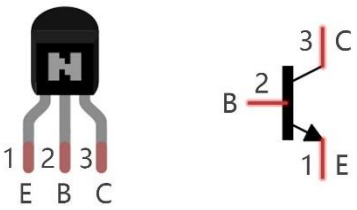
A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to amplify the current.

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



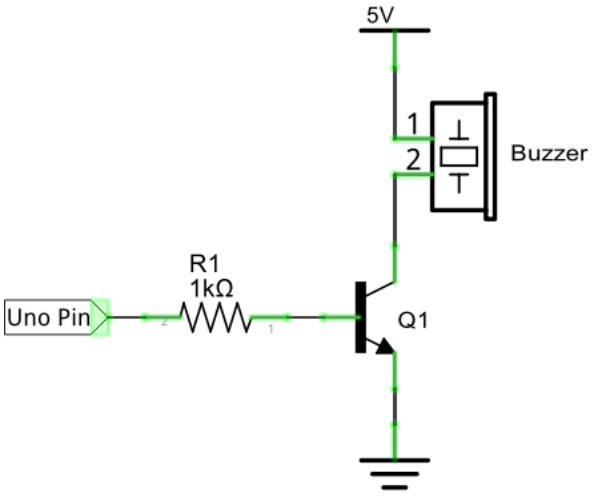
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

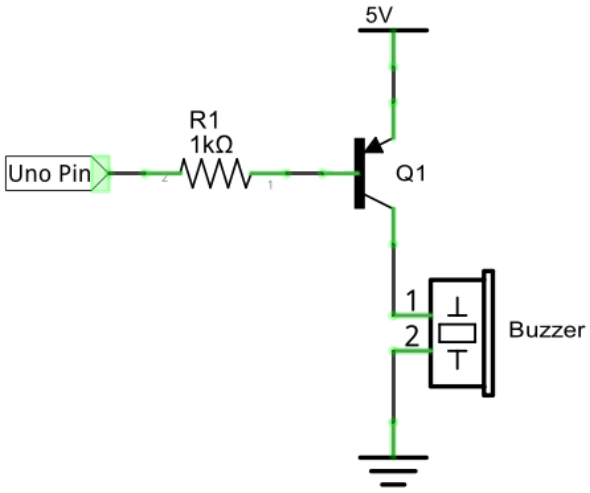
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not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

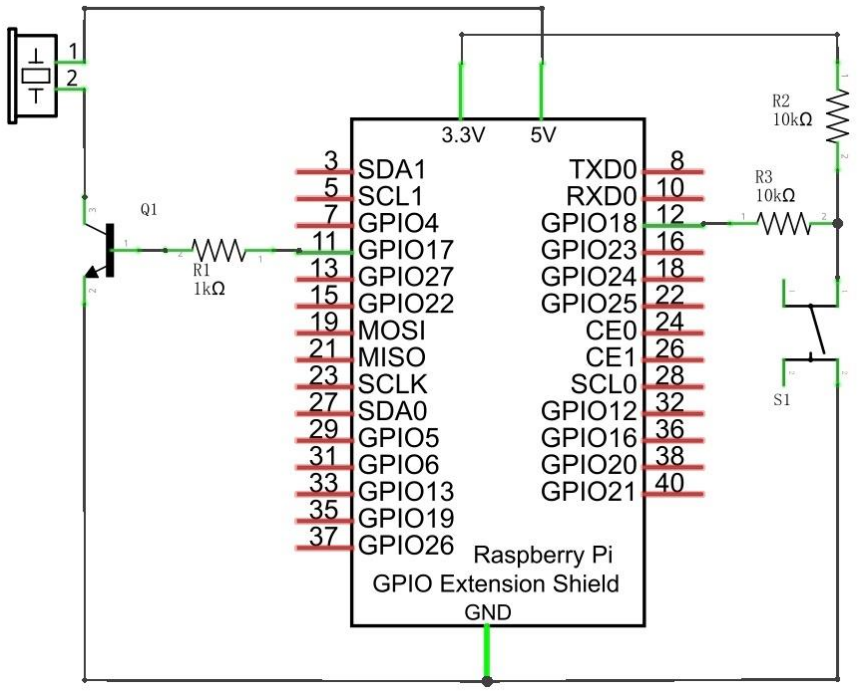


PNP transistor to drive buzzer

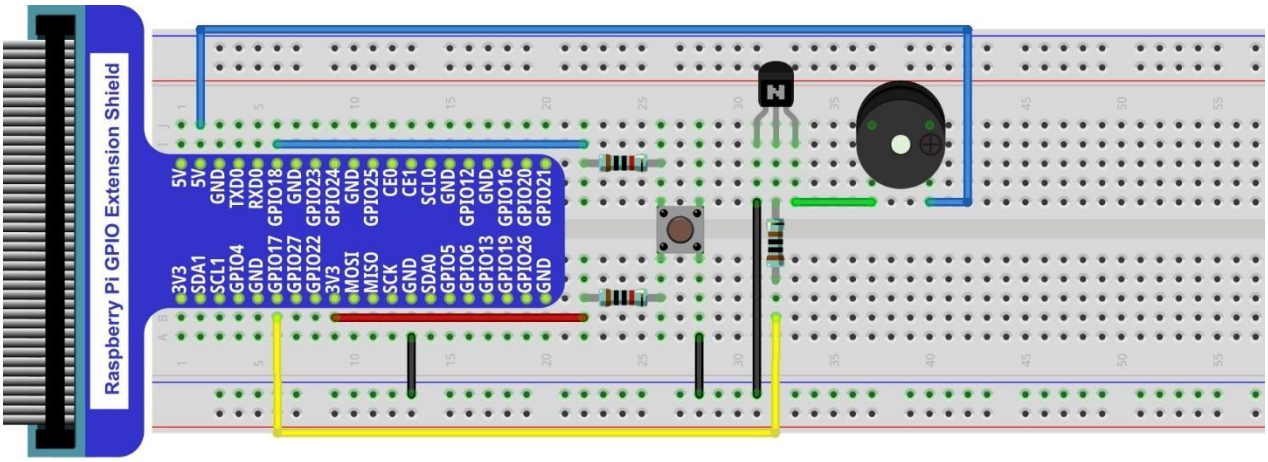


Circuit

Schematic diagram



Hardware connection



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).

If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we use a button to control the active buzzer. The code is similar to that in Chapter 3. Sketch_06_1_Doorbell

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell

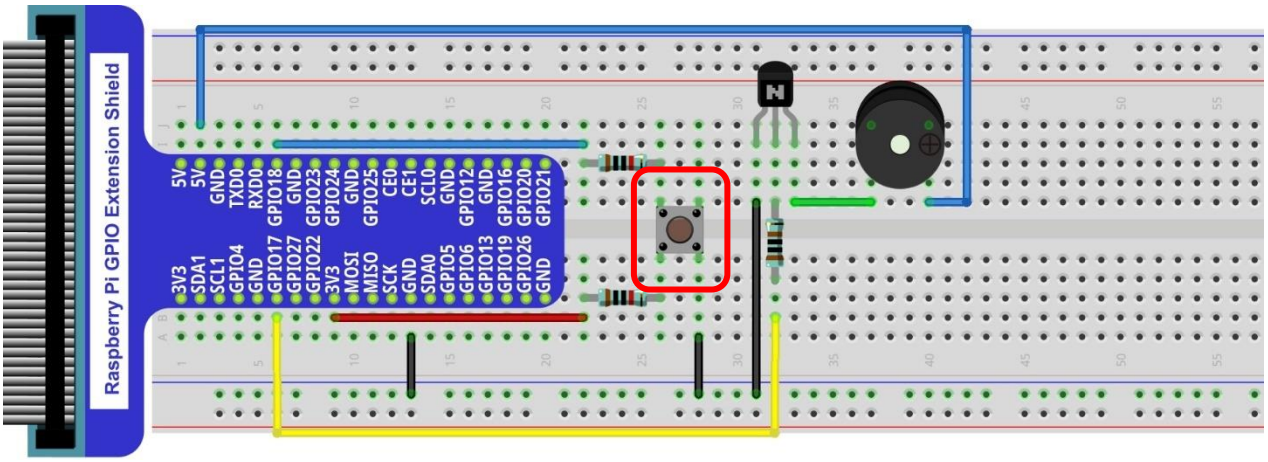
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell $
```

Enter the command to run the code.

jbang Doorbell.java

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell $ jbang Doorbell.java
```

During the code running, press the button to sound the buzzer and release the button to stop it.



You can see messages printed on the terminal.

```
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - buzzer turned on.  >>>
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - buzzer turned off. >>>
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - buzzer turned on.  >>>
[Pi4J.RUNTIME-1] INFO com.pi4j.util.Console - buzzer turned off. >>>
```

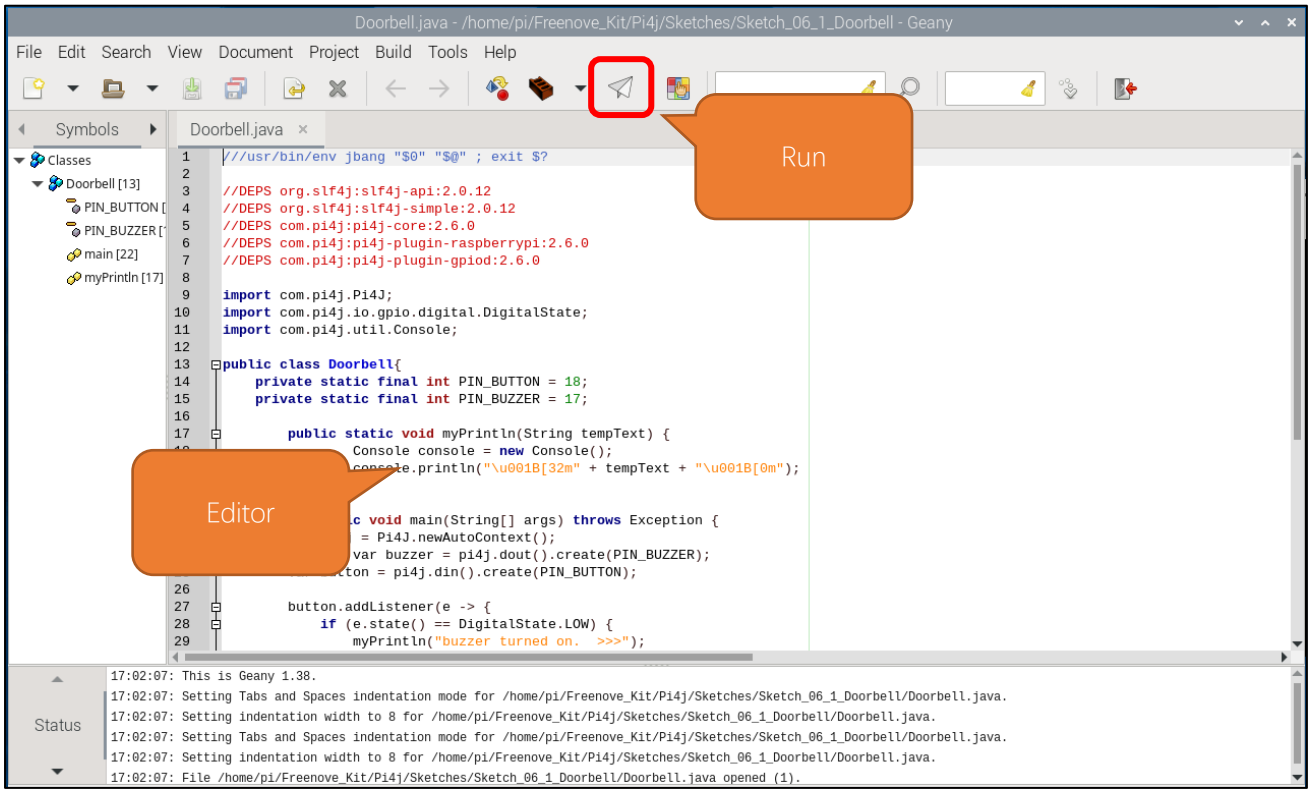
Press Ctrl+C to exit the code.

```
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.plugin.gpiod.provider.gpio.digital.GpioDDigitalInput - Shutdown input listener for DIN-21
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_06_1_Doorbell $
```

You can open the code with Geany to view and edit it, with the following command.

```
geany Doorbell.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
1  ///usr/bin/env jbang "$0" "$@" ; exit $?
2
3  //DEPS org.slf4j:slf4j-api:2.0.12
4  //DEPS org.slf4j:slf4j-simple:2.0.12
5  //DEPS com.pi4j:pi4j-core:2.6.0
6  //DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
7  //DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
8
9  import com.pi4j.Pi4J;
10 import com.pi4j.io.gpio.digital.DigitalState;
11 import com.pi4j.util.Console;
12
13 public class Doorbell{
14
15     private static final int PIN_BUTTON = 18;
16     private static final int PIN_BUZZER = 17;
17
18     public static void myPrintln(String tempText) {
```

```
19          Console console = new Console();
20          console.println("\u001B[32m" + tempText + "\u001B[0m");
21      }
22
23      public static void main(String[] args) throws Exception {
24          var pi4j = Pi4J.newAutoContext();
25          var buzzer = pi4j.dout().create(PIN_BUZZER);
26          var button = pi4j.din().create(PIN_BUTTON);
27
28          button.addListener(e -> {
29              if (e.state() == DigitalState.LOW) {
30                  myPrintln("buzzer turned on.  >>>");
31                  buzzer.high();
32              }
33              else if(e.state() == DigitalState.HIGH){
34                  myPrintln("buzzer turned off. >>>");
35                  buzzer.low();
36              }
37          });
38
39          try{
40              while (true) {
41                  Thread.sleep(500);
42              }
43          }
44          finally{
45              pi4j.shutdown();
46          }
47      }
48  }
```

Please note that as the code is similar to that in chapter 3, here we do not explain it again.

In order to make the prompt messages printed by the terminal more conspicuous, we wrote the myprintln function to wrap the console.println function.

```
    public static void myPrintln(String tempText) {
        Console console = new Console();
        console.println("\u001B[32m" + tempText + "\u001B[0m");
    }
```

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.

Sketch

In this chapter, we use button to control the passive buzzer to make sound.

Sketch_06_2_Alertor

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_2_Alertor
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_2_Alertor/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_06_2_Alertor $
```

Enter the command to run the code.

```
jbang Alertor.java
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_06_2_Alertor/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_06_2_Alertor $ jbang Alertor.java
```

When the code is running, press the button to sound the buzzer and release to stop it.

You can see messages printed on the terminal.

```
[main] INFO com.pi4j.util.Console - buzzer turned off. >>>
[main] INFO com.pi4j.util.Console - buzzer turned on.  >>>
[main] INFO com.pi4j.util.Console - buzzer turned on.  >>>
[main] INFO com.pi4j.util.Console - buzzer turned off. >>>
[main] INFO com.pi4j.util.Console - buzzer turned off. >>>
[main] INFO com.pi4j.util.Console - buzzer turned off. >>>
```

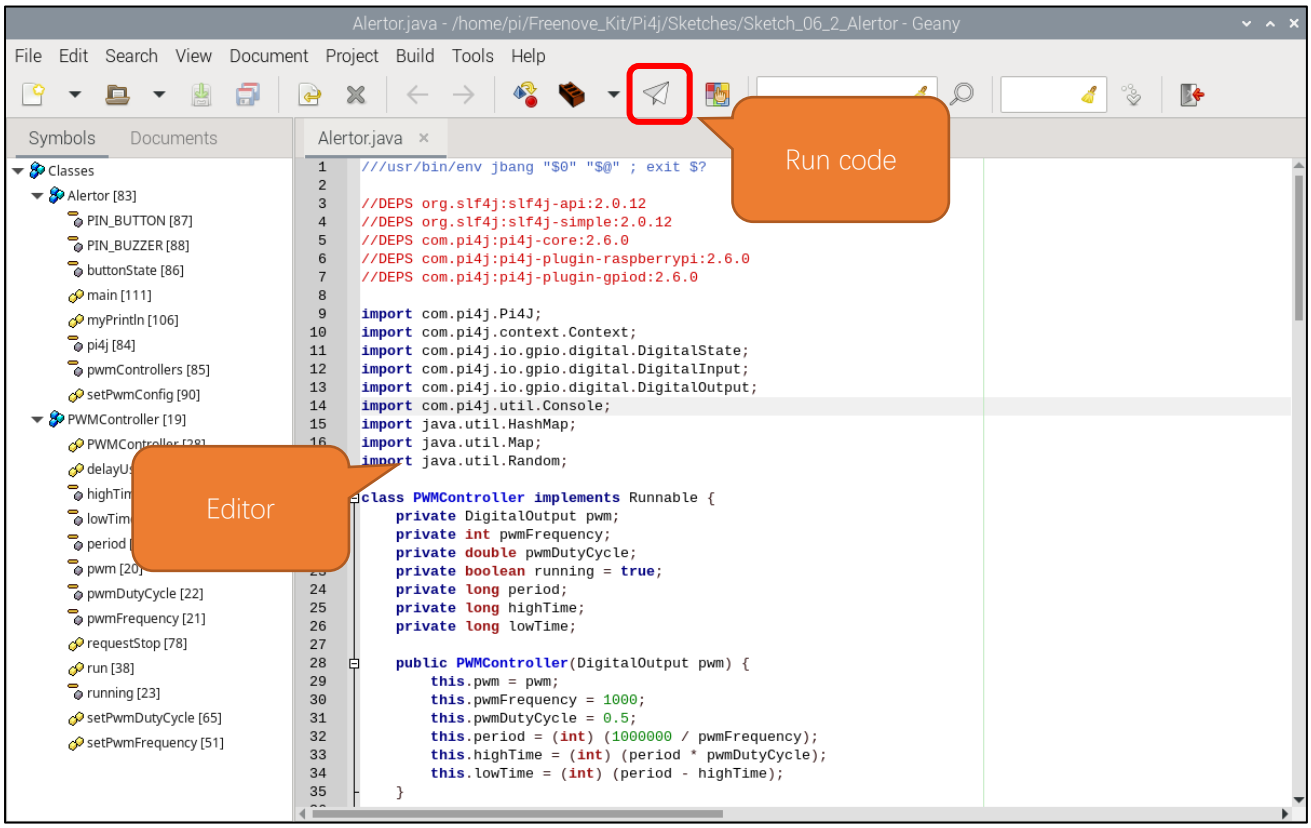
Press ctrl+C to exit the code.

```
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.plugin.gpiod.provider.gpio.digital.GpioDDigitalInput - Shutdown input listener for DIN-20
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shut
down event.
```

You can open the code with Geany to view and edit it, with the following command.

```
geany Alertor.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalState;
import com.pi4j.io.gpio.digital.DigitalInput;
import com.pi4j.io.gpio.digital.DigitalOutput;
import com.pi4j.util.Console;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
```

```
18
19  class PWMController implements Runnable {
...     ......
81  }
82
83  public class Alertor {
84      private static final Context pi4j = Pi4J.newAutoContext();
85      private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
86      private static volatile int buttonState = 0;
87      private static int PIN_BUTTON = 18;
88      private static int PIN_BUZZER = 17;
89
90      public static void setPwmConfig(int pin) throws Exception {
91          DigitalOutput pwm = pi4j.dout().create(pin);
92          PWMController pwmController = new PWMController(pwm);
93          Thread pwmThread = new Thread(pwmController, "PWM Controller " + pin);
94          pwmControllers.put(pin, pwmController);
95          pwmThread.start();
96          Runtime.getRuntime().addShutdownHook(new Thread(() -> {
97              pwmController.requestStop();
98              try {
99                  pwmThread.join();
100             } catch (InterruptedException e) {
101                 Thread.currentThread().interrupt();
102             }
103         }));
104     }
105
106     public static void myPrintln(String tempText) {
107         Console console = new Console();
108         console.println("\u001B[32m" + tempText + "\u001B[0m");
109     }
110
111     public static void main(String[] args) throws Exception {
112         DigitalInput button = pi4j.din().create(PIN_BUTTON);
113         button.addListener(e -> {
114             buttonState = e.state() == DigitalState.LOW ? 1 : 0;
115         });
116         setPwmConfig(PIN_BUZZER);
117         PWMController buzzer = pwmControllers.get(PIN_BUZZER);
118         buzzer.setPwmFrequency(0);
119         buzzer.setPwmDutyCycle(0.5);
120         try {
121             while (true) {
```

```
122                 if (buttonState == 1) {
123                     myPrintln("buzzer turned on.  >>>");
124                     buzzer.setPwmFrequency(1000);
125                 } else {
126                     myPrintln("buzzer turned off. >>>");
127                     buzzer.setPwmFrequency(0);
128                 }
129                 Thread.sleep(100);
130             }
131         }
132         finally {
133             for (PWMController controller : pwmControllers.values()) {
134                 controller.requestStop();
135             }
136             pi4j.shutdown();
137         }
138     }
139 }
```

Create a pi4j context for manipulating the GPIOs, and define the GPIO numbers that control the keys and buzzer.

```
    private static final Context pi4j = Pi4J.newAutoContext();
    private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
    private static volatile int buttonState = 0;
    private static int PIN_BUTTON = 18;
    private static int PIN_BUZZER = 17;
```

Initialize the button input pin, create a monitoring event, and assign the key value to buttonState when the button state changes.

```
        DigitalInput button = pi4j.din().create(PIN_BUTTON);
        button.addListener(e -> {
            buttonState = e.state() == DigitalState.LOW ? 1 : 0;
        });
```

Initialize the PWM pin that controls the passive buzzer.

```
        setPwmConfig(PIN_BUZZER);
        PWMController buzzer = pwmControllers.get(PIN_BUZZER);
        buzzer.setPwmFrequency(0);
        buzzer.setPwmDutyCycle(0.5);
```

When buttonState is 1, the passive buzzer sounds, and when buttonState is 0, the buzzer sound is turned off. Print the information in the console.

```
                if (buttonState == 1) {
                    myPrintln("buzzer turned on.  >>>");
                    buzzer.setPwmFrequency(1000);
                } else {
                    myPrintln("buzzer turned off. >>>");
                    buzzer.setPwmFrequency(0);
                }
```

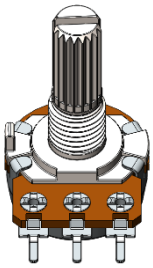
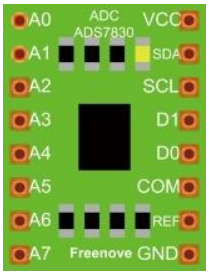
(Important) Chapter 7 ADC

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.

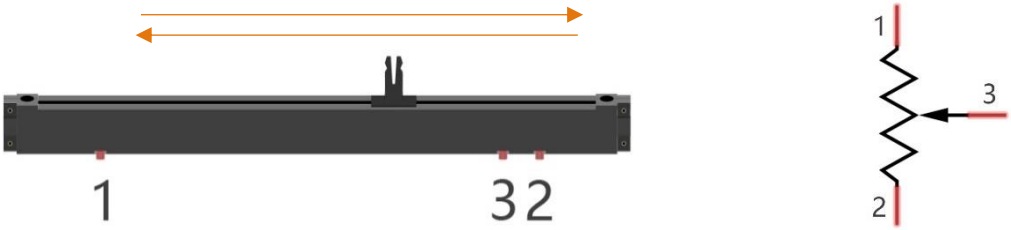
Component List

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

Component knowledge

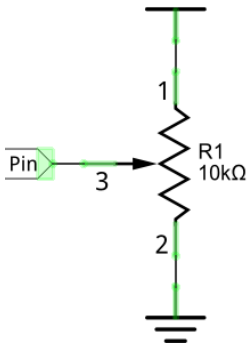
Potentiometer

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.



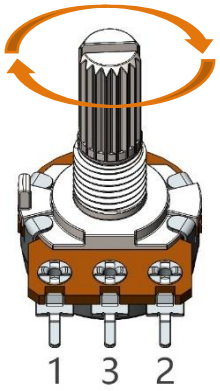
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (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.



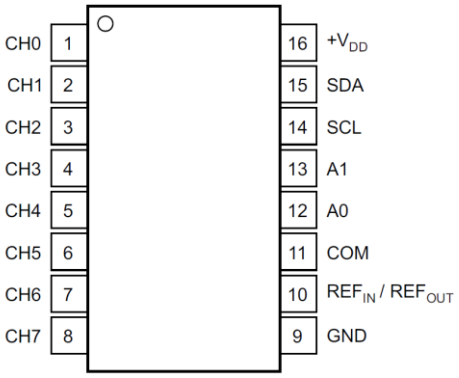
Rotary potentiometer

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



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.

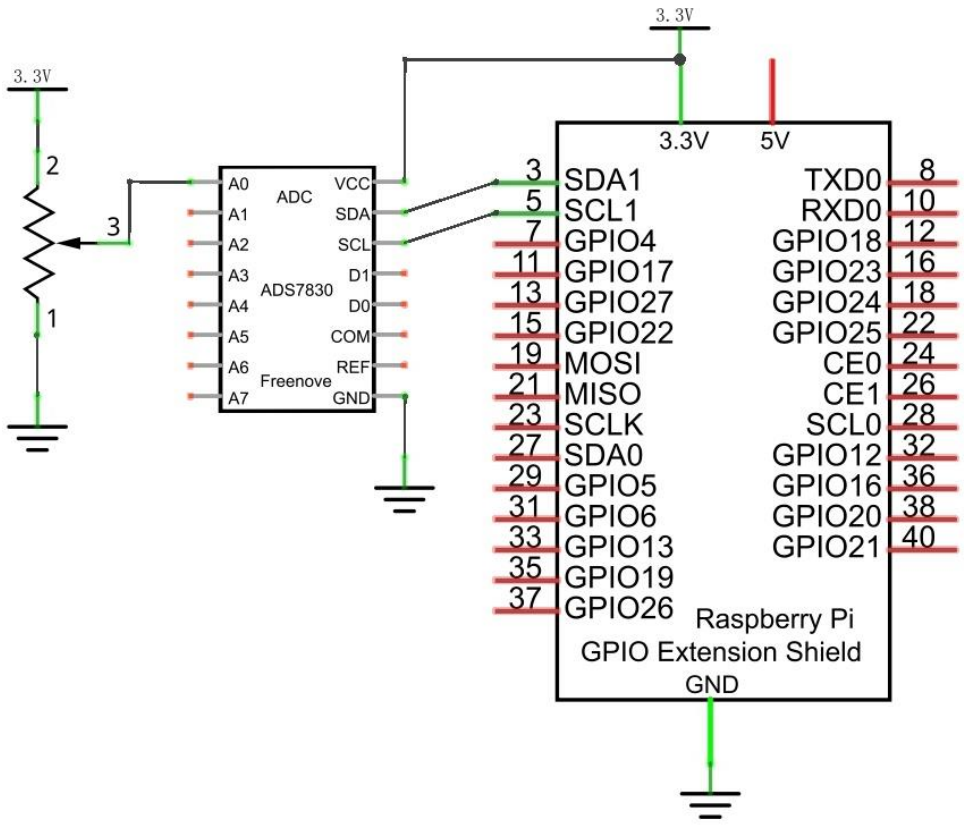
SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		
CH5	6		
CH6	7		
CH7	8		
GND	9	Ground	
REF in/out	10	Internal +2.5V Reference, External Reference Input	
COM	11	Common to Analog Input Channel	
A0	12	Hardware address	
A1	13		
SCL	14	Serial Clock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

I2C communication

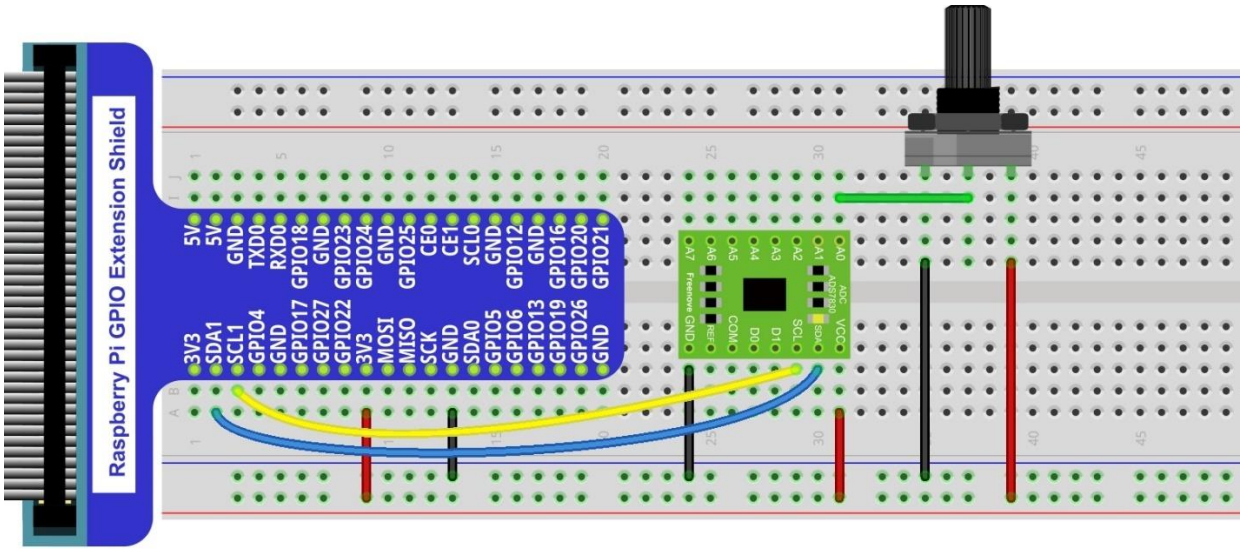
I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique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

Circuit

Schematic diagram



Hardware connection.



If you have any concerns, please send an email to: support@freenove.com

Configure I2C and Install Smbus

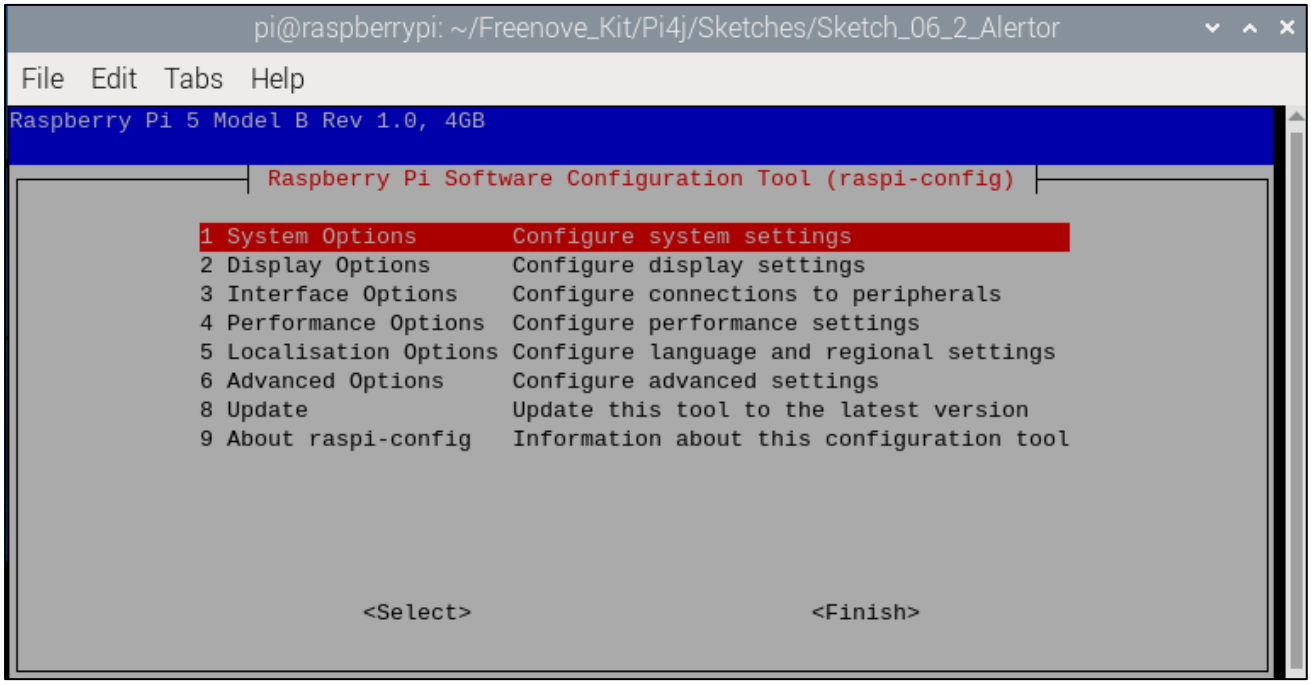
Enable I2C

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

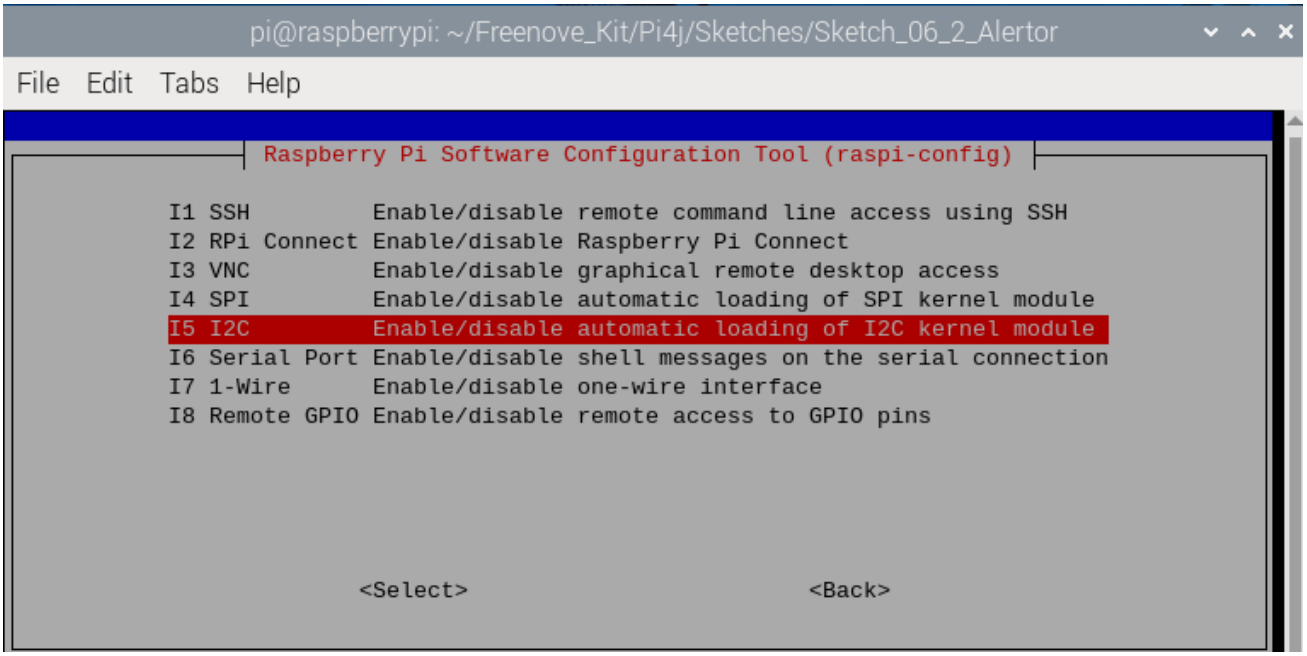
Type command in the Terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose "Interfacing Options" then "I5 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
```

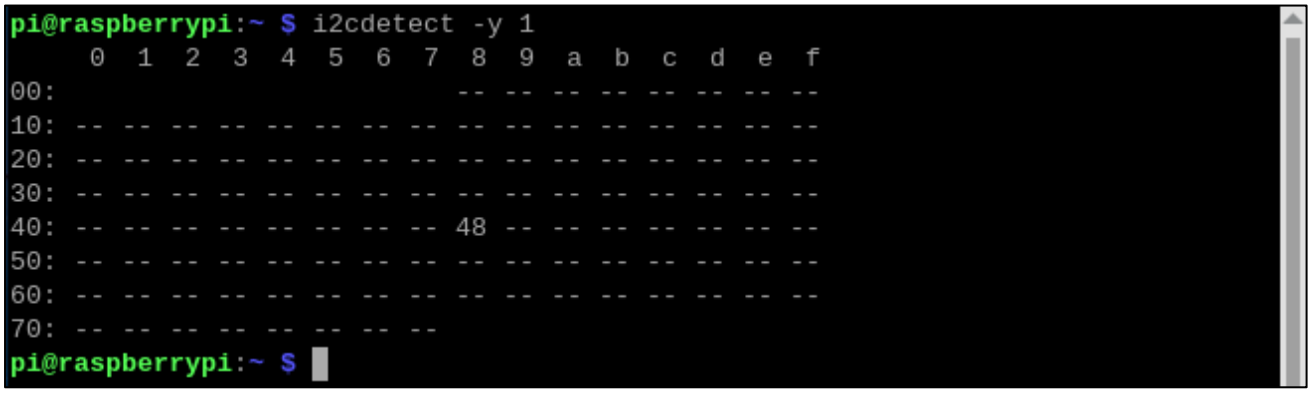
If the I2C module has been started, the following content will be shown.

Different models of Raspberry Pi display different contents depending on the CPU installed:

I2C device address detection:

```
i2cdetect -y 1
```

When you are using the ADS7830 Module, the result should look like this:



Here, 48 (HEX) is the I2C address of ADC Module (ADS7830).

Sketch

In this chapter, we will learn the combined usage of ADC and potentiometer.

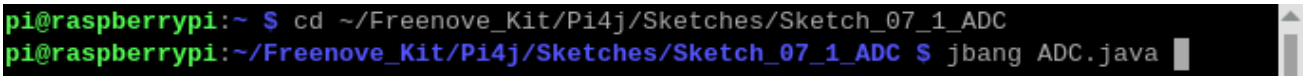
Sketch_07_1_ADC

First, enter where the project is located:

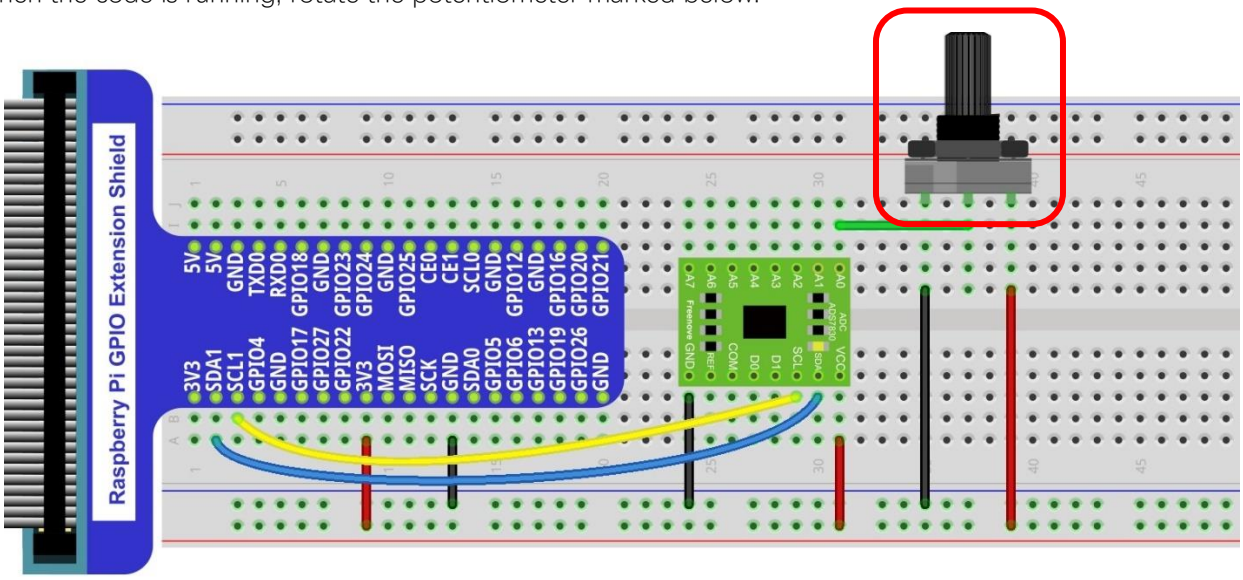
```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_1_ADC
```

Enter the command to run the code.

```
jbang ADC.java
```



When the code is running, rotate the potentiometer marked below.



You can see that the ADC values change with the rotation of the potentiometer. The value 0 means that the potentiometer's voltage read by ADC is 0V, 255 indicates that the voltage is 5V.

```
pi@raspberrypi: ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_1_ADC
File  Edit  Tabs  Help
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:53
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:112
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:162
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:206
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:255
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:255
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:255
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:190
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:89
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:16
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
[main] INFO com.pi4j.util.Console - ADC Channel 2 Value:0
```

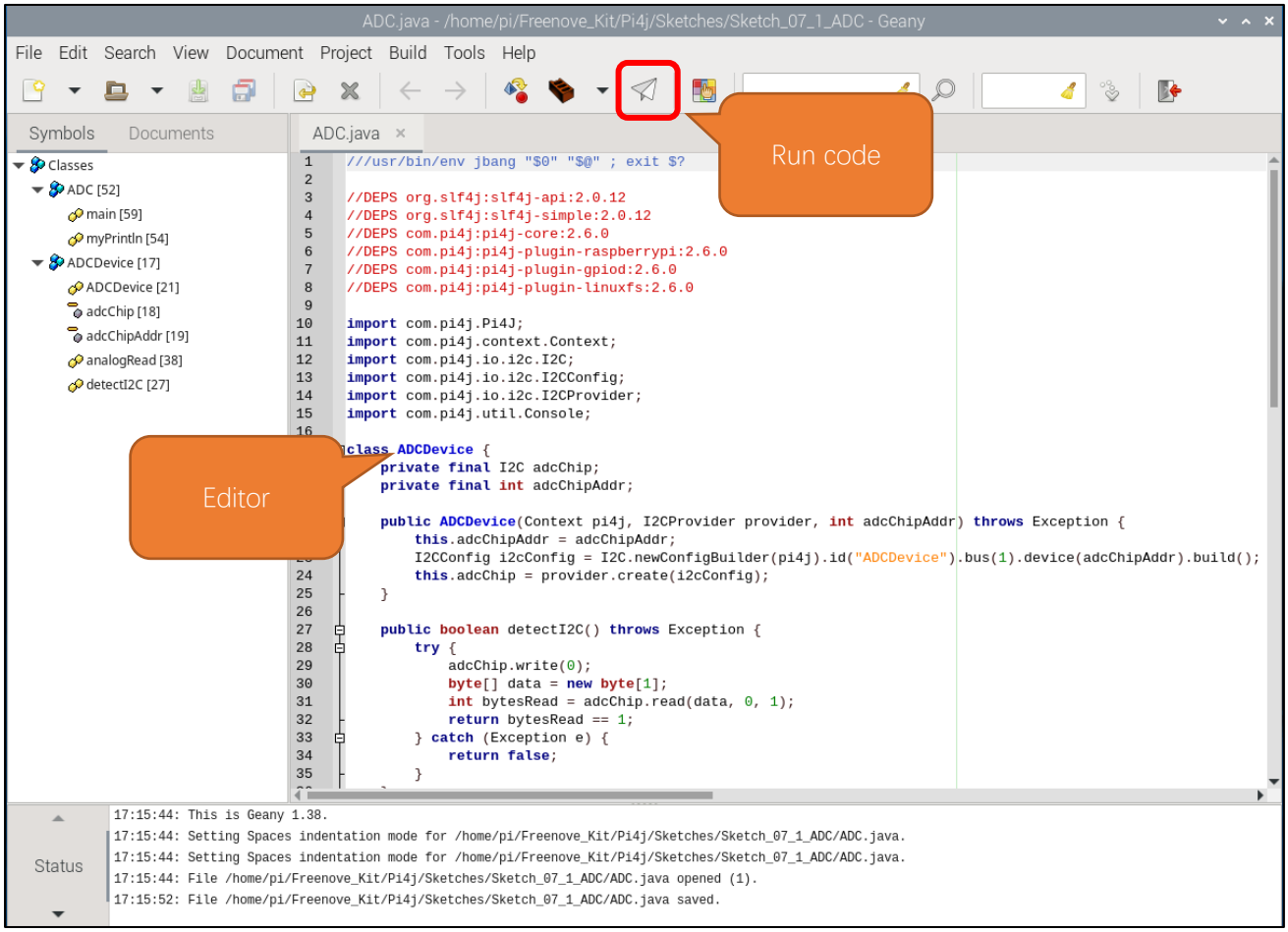
Press Ctrl+C to exit the code.

```
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdow
n. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_1_ADC $
```

You can open the code with Geany with the following command to view and edit it.

geany ADC.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
1  ///usr/bin/env jbang "$0" "$@" ; exit $?
2
3  //DEPS org.slf4j:slf4j-api:2.0.12
4  //DEPS org.slf4j:slf4j-simple:2.0.12
5  //DEPS com.pi4j:pi4j-core:2.6.0
6  //DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
7  //DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
8  //DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0
9
10 import com.pi4j.Pi4J;
11 import com.pi4j.context.Context;
12 import com.pi4j.io.i2c.I2C;
13 import com.pi4j.io.i2c.I2CConfig;
14 import com.pi4j.io.i2c.I2CProvider;
```

```
15  import com.pi4j.util.Console;
16
17  class ADCDevice {
18      private final I2C adcChip;
19      private final int adcChipAddr;
20
21      public ADCDevice(Context pi4j, I2CProvider provider, int adcChipAddr) throws Exception {
22          this.adcChipAddr = adcChipAddr;
23          I2CConfig i2cConfig =
            I2C.newConfigBuilder(pi4j).id("ADCDevice").bus(1).device(adcChipAddr).build();
24          this.adcChip = provider.create(i2cConfig);
25      }
26
27      public boolean detectI2C() throws Exception {
28          try {
29              adcChip.write(0);
30              byte[] data = new byte[1];
31              int bytesRead = adcChip.read(data, 0, 1);
32              return bytesRead == 1;
33          } catch (Exception e) {
34              return false;
35          }
36      }
37
38      public int analogRead(int chn) {
39          byte command = (byte) (0x84 | (((chn << 2 | chn >> 1) & 0x07) << 4));
40          adcChip.write(command);
41          byte[] data = new byte[1];
42          int bytesRead = adcChip.read(data, 0, 1);
43          if (bytesRead == 1) {
44              int adcValue = data[0] & 0xFF;
45              return adcValue;
46          } else {
47              return -1;
48          }
49      }
50  }
51
52  public class ADC{
53
54      public static void myPrintln(String format, Object... args) {
55          Console console = new Console();
56          console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
57      }
```

```
58
59      public static void main(String[] args) throws Exception {
60          Context pi4j = Pi4J.newAutoContext();
61          I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
62          try {
63              int ADC_CHIP_ADDR = 0x48;
64              ADCDevice adcDevice = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
65              if (adcDevice.detectI2C()) {
66                  int ADC_CHANNEL = 0;
67                  while (true) {
68                      int adcValue = adcDevice.analogRead(ADC_CHANNEL);
69                      if (adcValue != -1) {
70                          myPrintln("ADC Channel %d Value:%d", ADC_CHANNEL, adcValue);
71                      } else {
72                          myPrintln("Failed to read data from ADC.");
73                      }
74                      Thread.sleep(100);
75                  }
76              } else {
77                  myPrintln("ADS7830 device not detected at address 0x" +
78                  Integer.toHexString(ADC_CHIP_ADDR));
79              }
80          } finally {
81              pi4j.shutdown();
82          }
83      }
84  }
```

Dependency declaration, these libraries will be automatically downloaded by jbang at runtime and added to the classpath.

```
//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0
```

Import I2C library. In this project, we use I2C to read the channel value of ADS7830.

```
import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
import com.pi4j.util.Console;
```

Constructor of ADCDevice class, which is used to initialize I2C bus to facilitate later reading and writing ADS7830 chip.

```
    public ADCDevice(Context pi4j, I2CProvider provider, int adcChipAddr) throws Exception {
        this.adcChipAddr = adcChipAddr;
        I2CConfig i2cConfig =
        I2C.newConfigBuilder(pi4j).id("ADCDevice").bus(1).device(adcChipAddr).build();
        this.adcChip = provider.create(i2cConfig);
    }
```

Write a byte to the target chip, and then read the data. If the data can be read, it means the target chip exists and communication is successful. If an I2C exception is detected, it means the target chip does not exist.

```
    public boolean detectI2C() throws Exception {
        try {
            adcChip.write(0);
            byte[] data = new byte[1];
            int bytesRead = adcChip.read(data, 0, 1);
            return bytesRead == 1;
        } catch (Exception e) {
            return false;
        }
    }
```

Write the read command to the ADS7830 and read the corresponding ADC value. It is returned by the return value.

```
    public int analogRead(int chn) {
        byte command = (byte) (0x84 | (((chn << 2 | chn >> 1) & 0x07) << 4));
        adcChip.write(command);
        byte[] data = new byte[1];
        int bytesRead = adcChip.read(data, 0, 1);
        if (bytesRead == 1) {
            int adcValue = data[0] & 0xFF;
            return adcValue;
        } else {
            return -1;
        }
    }
```

Create a pi4j context to get the Raspberry PI i2c interface.

```
        Context pi4j = Pi4J.newAutoContext();
        I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
```

The I2C address of the ADS7830 is 0x48.

Create an ADCDevice class, associate it with the Raspberry PI I2C interface, and assign a value to the adcDevice.

```
int ADC_CHIP_ADDR = 0x48;
ADCDevice adcDevice = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
```

Check whether the chip can communicate normally. If the communication is successful, read channel 0 of the ADS7830 chip and print it out in the terminal.

```
if (adcDevice.detectI2C()) {
    int ADC_CHANNEL = 0;
    while (true) {
        int adcValue = adcDevice.analogRead(ADC_CHANNEL);
        if (adcValue != -1) {
            myPrintln("ADC Channel %d Value:%d", ADC_CHANNEL, adcValue);
        } else {
            myPrintln("Failed to read data from ADC.");
        }
        Thread.sleep(100);
    }
}
```

If communication with the chip fails, a prompt message is printed on the terminal.

```
else {
    myPrintln("ADS7830 device not detected at address 0x" +
    Integer.toHexString(ADC_CHIP_ADDR));
}
```

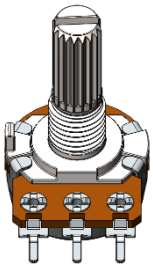
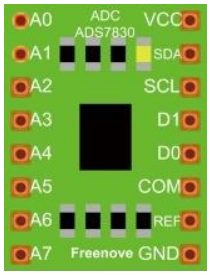
When the code finishes running, close the Pi4J context.

```
finally {
    pi4j.shutdown();
}
```

Project 7.2 Soft Light

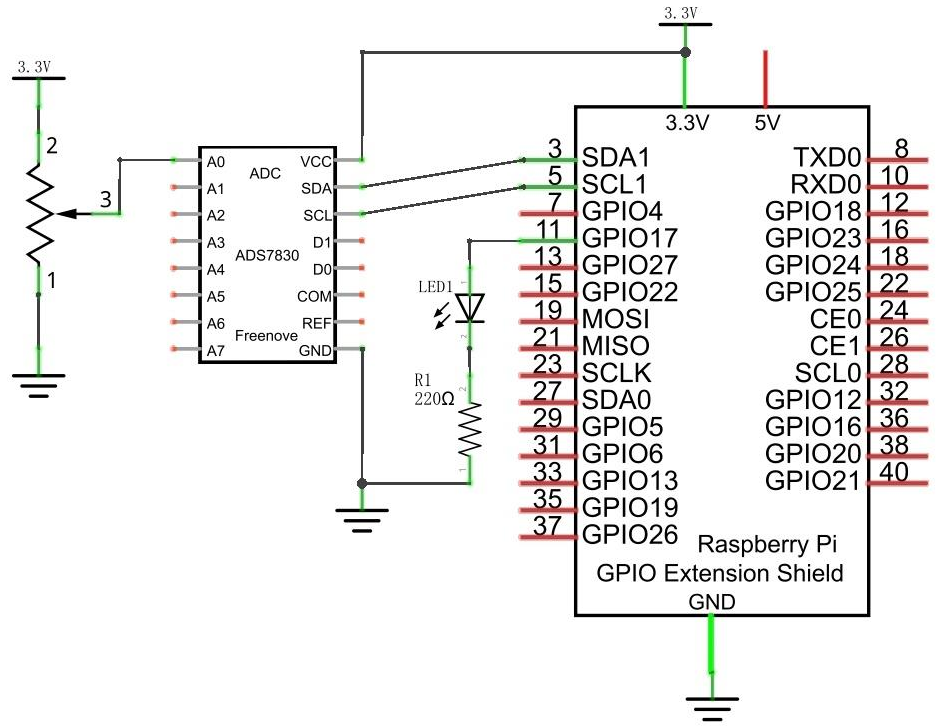
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle ratio of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

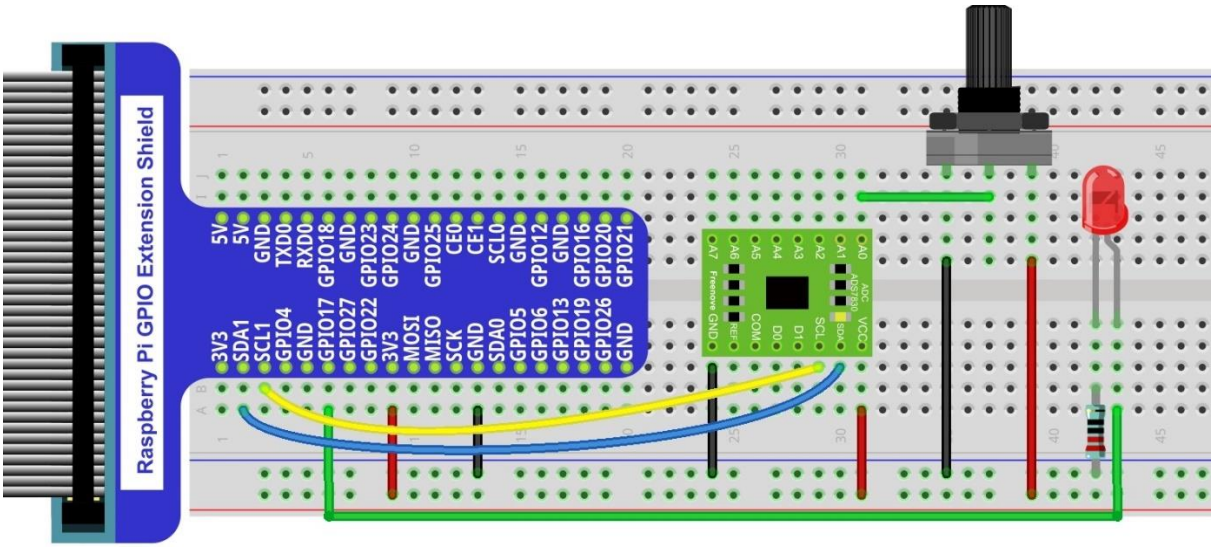
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wire M/M x17
  </td></tr>
<tr><td>Rotary Potentiometer x1
  </td><td>ADC Module x1 （Only one）
 
</td><td>10kΩ x2
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

Circuit

Schematic diagram



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



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this project, we learn how to control the brightness of LED with the potentiometer.

Sketch_07_2_Softlight

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_2_Softlight
```

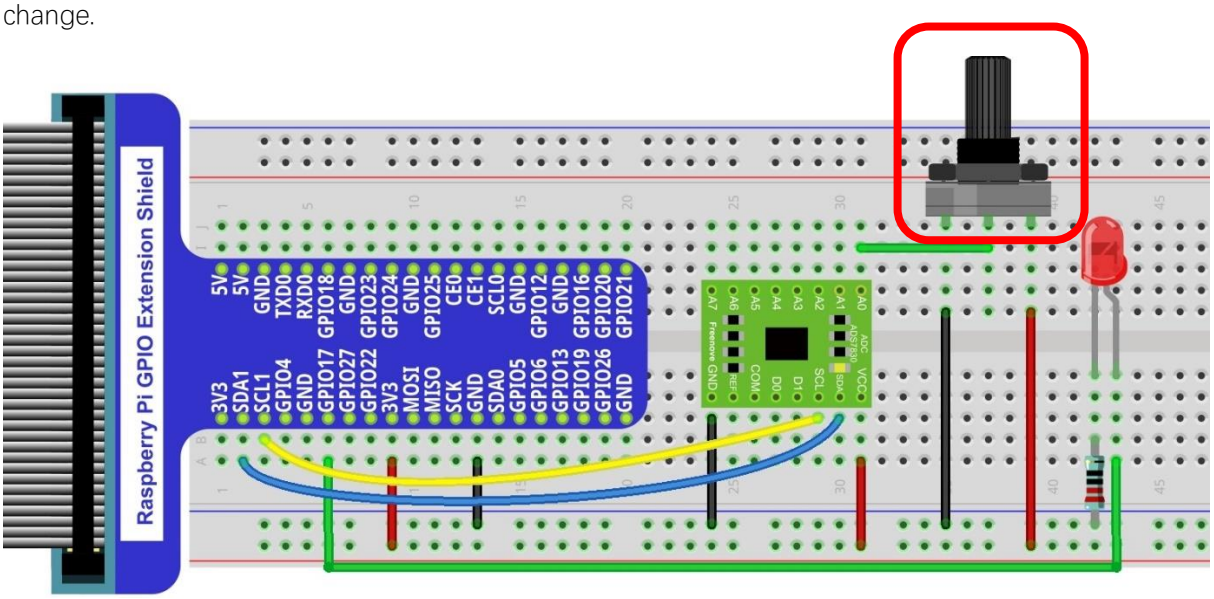
```
pi@raspberrypi:~ $ cd Freenove_Kit/Pi4j/Sketches/Sketch_07_2_Softlight/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_2_Softlight $
```

Enter the command to run code.

```
jbang Softlight.java
```

```
pi@raspberrypi:~ $ cd Freenove_Kit/Pi4j/Sketches/Sketch_07_2_Softlight/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_2_Softlight $ jbang Softlight.java
```

When the code is running, turn the potentiometer marked below and you can see the brightness of the LED change.



On the Terminal, you can see the printed ADC values and the calculated voltage values.

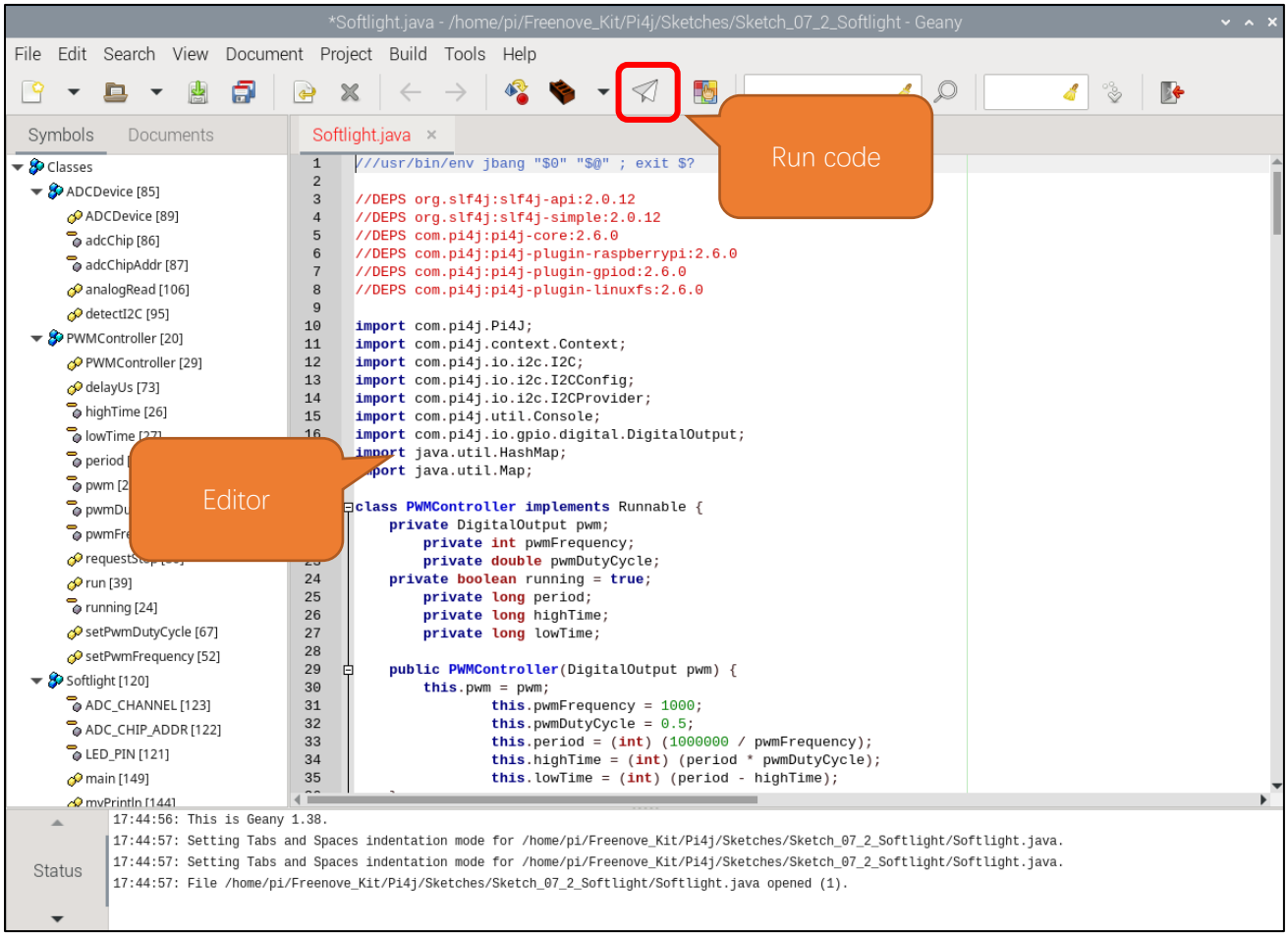
```
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
[main] INFO com.pi4j.util.Console - ADC value:92, Voltage:1.80V
```

Press Ctrl+C to exit the program.

You can open the code with Geany with the following command to view and edit it.

```
geany Softlight.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
```

```
15  import com.pi4j.util.Console;
16  import com.pi4j.io.gpio.digital.DigitalOutput;
17  import java.util.HashMap;
18  import java.util.Map;
19
20  class PWMController implements Runnable {
...     ......
83  }
84
85  class ADCDevice {
...     ......
118 }
119
120 public class Softlight{
121     private static int   LED_PIN = 17;
122     private static int   ADC_CHIP_ADDR = 0x48;
123     private static int   ADC_CHANNEL = 0;
124
125     private static final Context pi4j = Pi4J.newAutoContext();
126     private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
127
128     public static void setPwmConfig(int pin) throws Exception {
129         DigitalOutput pwm = pi4j.dout().create(pin);
130         PWMController pwmController = new PWMController(pwm);
131         Thread pwmThread = new Thread(pwmController, "PWM Controller " + pin);
132         pwmControllers.put(pin, pwmController);
133         pwmThread.start();
134         Runtime.getRuntime().addShutdownHook(new Thread(() -> {
135             pwmController.requestStop();
136             try {
137                 pwmThread.join();
138             } catch (InterruptedException e) {
139                 Thread.currentThread().interrupt();
140             }
141         }));
142     }
143
144     public static void myPrintln(String format, Object... args) {
145         Console console = new Console();
146         console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
147     }
148
149     public static void main(String[] args) throws Exception {
150         Context pi4j = Pi4J.newAutoContext();
```

```
151         I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
152         setPwmConfig(LED_PIN);
153         PWMController led = pwmControllers.get(LED_PIN);
154 
155         try {
156             ADCDevice adc = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
157             if (adc.detectI2C()) {
158                 while (true) {
159                     int adcValue = adc.analogRead(ADC_CHANNEL);
160                     if (adcValue != -1) {
161                         led.setPwmDutyCycle(((double)adcValue/255.0));
162                         double voltage = (double)adcValue / 255.0 * 5.0;
163                         myPrintln("ADC value:%d, Voltage:%.2fV", adcValue, voltage);
164                     } else {
165                         myPrintln("Failed to read data from ADC.");
166                     }
167                     Thread.sleep(100);
168                 }
169             } else {
170                 myPrintln("ADS7830 device not detected at address 0x" +
                    Integer.toHexString(ADC_CHIP_ADDR));
171             }
172         } finally {
173             pi4j.shutdown();
174         }
175     }
176 }
```

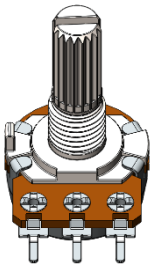
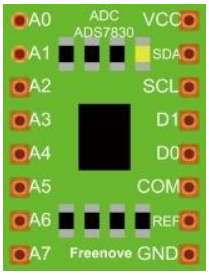
The ADC value of the potentiometer is obtained every 100 milliseconds and printed on the terminal. Meanwhile, the ADC value is converted into the duty cycle value of the LED to control the brightness of the LED.

```
                while (true) {
                    int adcValue = adc.analogRead(ADC_CHANNEL);
                    if (adcValue != -1) {
                        led.setPwmDutyCycle(((double)adcValue/255.0));
                        double voltage = (double)adcValue / 255.0 * 5.0;
                        myPrintln("ADC value:%d, Voltage:%.2fV", adcValue, voltage);
                    } else {
                        myPrintln("Failed to read data from ADC.");
                    }
                    Thread.sleep(100);
                }
```

Project 7.3 Colorful Light

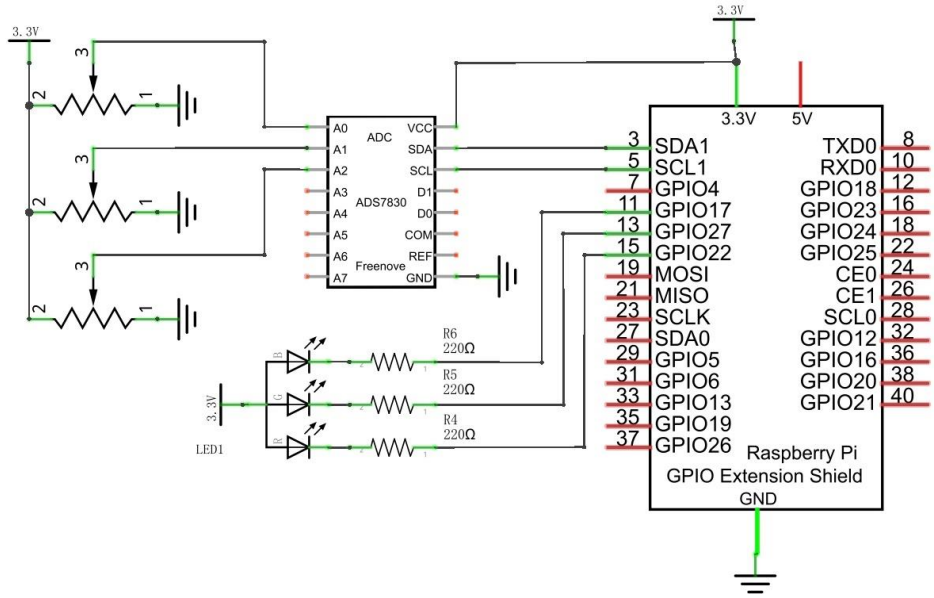
In this project, 3 potentiometers are used to control the RGB LED and in principle, it is the same as with the 'Soft Light' project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the previous soft light project needed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

Component List

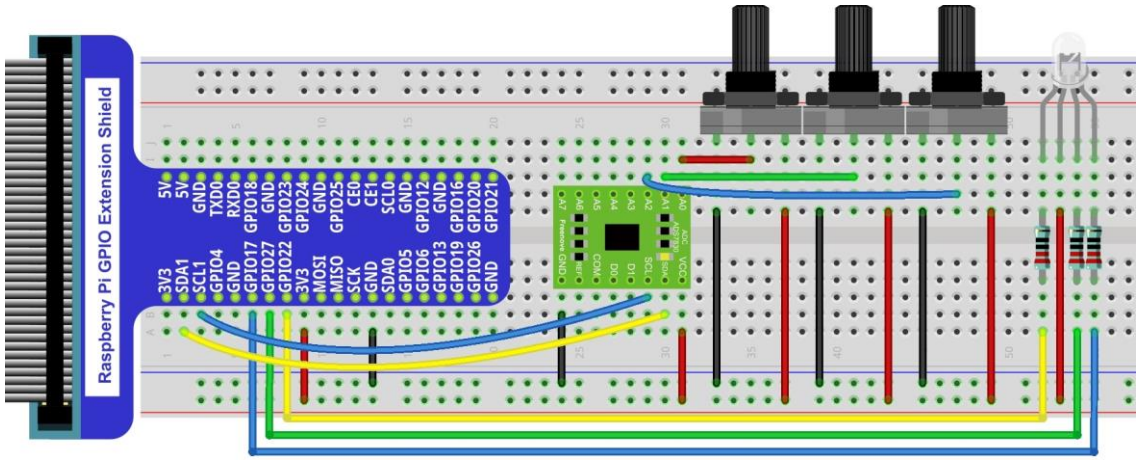
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x17
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td>ADC module x1
 
</td><td>10kΩ x2
  </td><td>220Ω x3
  </td><td>RGB LEDx1
  </td></tr>
</table>

Circuit

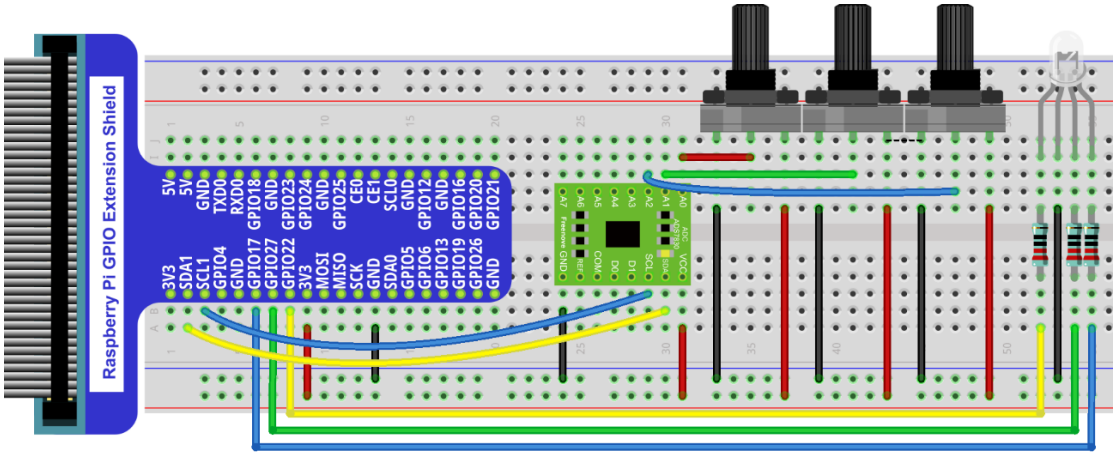
Schematic diagram



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.



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this project, we learn to use the potentiometer to control the color and brightness of the RGB LED.

Sketch_07_3_ColorfulSoftlight

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_3_ColorfulSoftlight
```

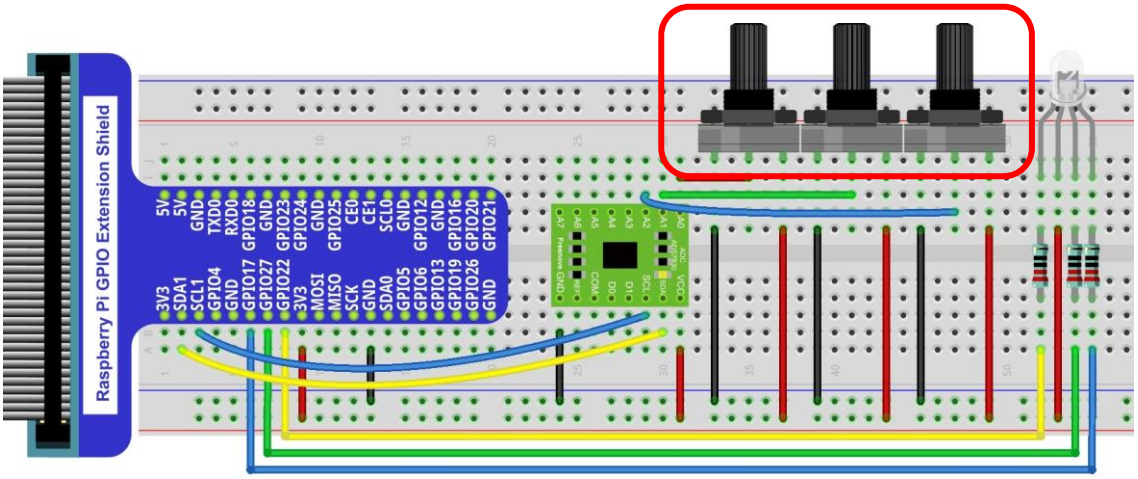
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_3_ColorfulSoftlight
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_3_ColorfulSoftlight $
```

Enter the command to run the code.

```
jbang ColorfulSoftlight.java
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_07_3_ColorfulSoftlight
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_3_ColorfulSoftlight $ jbang ColorfulSoftlight.java
```

When the code is running, rotate the three potentiometers marked below, you will see the RGB LED's color and brightness change.



The ADC value is printed on the terminal.

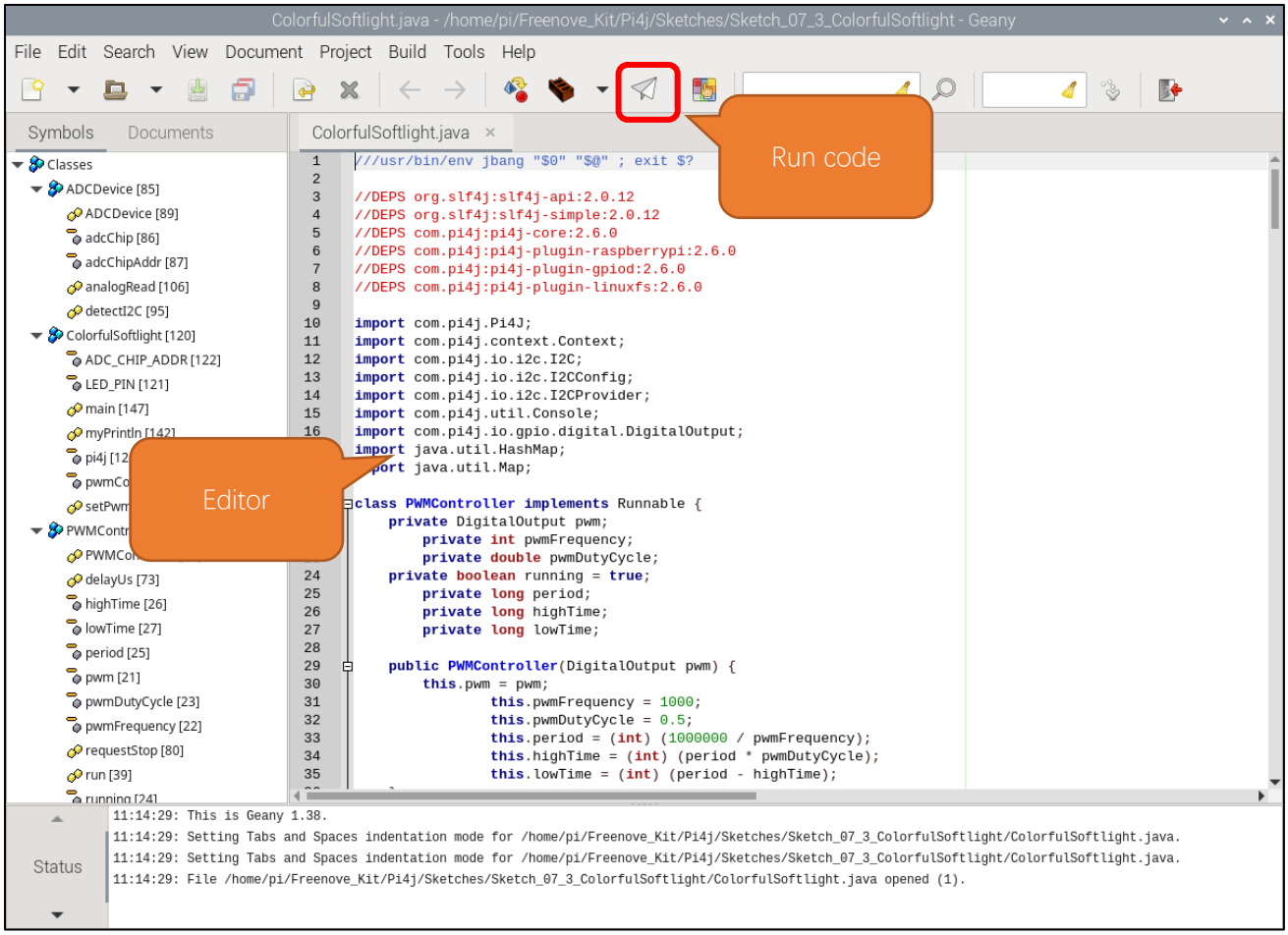
```
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:144
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:144
[main] INFO com.pi4j.util.Console - ADC value val_Red:131, val_Green:127, val_Blue:144
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:143
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:144
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:143
[main] INFO com.pi4j.util.Console - ADC value val_Red:132, val_Green:127, val_Blue:144
```

Press Ctrl+C to exit the program.

You can open the code with Geany with the following command to view and edit it.

```
geany ColorfulSoftlight.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
```

```
15  import com.pi4j.util.Console;
16  import com.pi4j.io.gpio.digital.DigitalOutput;
17  import java.util.HashMap;
18  import java.util.Map;
19
20  class PWMController implements Runnable {
...     ......
83  }
84
85  class ADCDevice {
...     ......
118 }
119
120 public class ColorfulSoftlight{
121     private static int   LED_PIN = 17;
122     private static int   ADC_CHIP_ADDR = 0x48;
123     private static final Context pi4j = Pi4J.newAutoContext();
124     private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
125
126     public static void setPwmConfig(int pin) throws Exception {
127         DigitalOutput pwm = pi4j.dout().create(pin);
128         PWMController pwmController = new PWMController(pwm);
129         Thread pwmThread = new Thread(pwmController, "PWM Controller " + pin);
130         pwmControllers.put(pin, pwmController);
131         pwmThread.start();
132         Runtime.getRuntime().addShutdownHook(new Thread(() -> {
133             pwmController.requestStop();
134             try {
135                 pwmThread.join();
136             } catch (InterruptedException e) {
137                 Thread.currentThread().interrupt();
138             }
139         }));
140     }
141
142     public static void myPrintln(String format, Object... args) {
143         Console console = new Console();
144         console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
145     }
146
147     public static void main(String[] args) throws Exception {
148         Context pi4j = Pi4J.newAutoContext();
149         I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
150
```

```
151         int[] ADC_CHN =  {0, 1, 2};
152         int[] LED_PINS = {17, 27, 22};
153         for (int pin : LED_PINS) {
154             setPwmConfig(pin);
155         }
156         PWMController red_led = pwmControllers.get(LED_PINS[0]);
157         PWMController green_led = pwmControllers.get(LED_PINS[1]);
158         PWMController blue_led = pwmControllers.get(LED_PINS[2]);
159 
160         try {
161             ADCDevice adc = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
162             if (adc.detectI2C()) {
163                 while (true) {
164                     int val_Red = adc.analogRead(ADC_CHN[0]);
165                     int val_Green = adc.analogRead(ADC_CHN[1]);
166                     int val_Blue = adc.analogRead(ADC_CHN[2]);
167 
168                     red_led.setPwmDutyCycle(1-(double)(val_Red/255.0));
169                     green_led.setPwmDutyCycle(1-(double)(val_Green/255.0));
170                     blue_led.setPwmDutyCycle(1-(double)(val_Blue/255.0));
171 
172                     myPrintln("ADC value val_Red:%d, val_Green:%d, val_Blue:%d", val_Red,
                        val_Green, val_Blue);
173                     Thread.sleep(100);
174                 }
175             } else {
176                 myPrintln("ADS7830 device not detected at address 0x" +
                    Integer.toHexString(ADC_CHIP_ADDR));
177             }
178         } finally {
179             pi4j.shutdown();
180         }
181     }
182 }
```

Initialize the pins that control the RGB LED.

```
        int[] LED_PINS = {17, 27, 22};
        for (int pin : LED_PINS) {
            setPwmConfig(pin);
        }
        PWMController red_led = pwmControllers.get(LED_PINS[0]);
        PWMController green_led = pwmControllers.get(LED_PINS[1]);
        PWMController blue_led = pwmControllers.get(LED_PINS[2]);
```

Get the ADC values corresponding to the 3 rotentiometers every 100 milliseconds; convert the values into duty cycle values corresponding to PWM, and print prompt information on the terminal.

```
while (true) {
    int val_Red = adc.analogRead(ADC_CHN[0]);
    int val_Green = adc.analogRead(ADC_CHN[1]);
    int val_Blue = adc.analogRead(ADC_CHN[2]);

    red_led.setPwmDutyCycle(1-(double)(val_Red/255.0));
    green_led.setPwmDutyCycle(1-(double)(val_Green/255.0));
    blue_led.setPwmDutyCycle(1-(double)(val_Blue/255.0));

    myPrintln("ADC value val_Red:%d, val_Green:%d, val_Blue:%d", val_Red,
    val_Green, val_Blue);
    Thread.sleep(100);
}
```

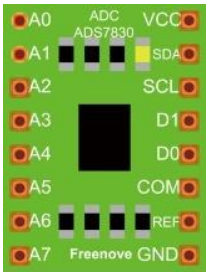
Chapter 8 Photoresistor & LED

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

Project 8.1 NightLamp

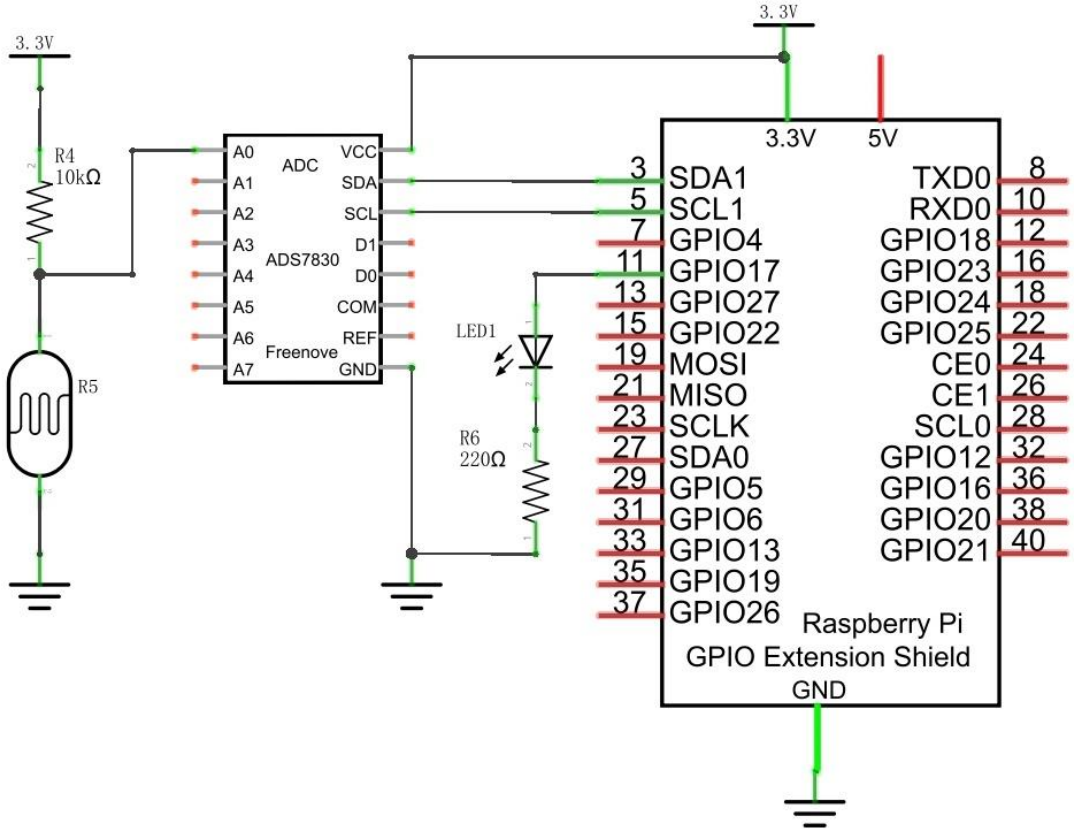
A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a nightlight with the following function. When the ambient light is less (darker environment), the LED will automatically become brighter to compensate and when the ambient light is greater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

Component List

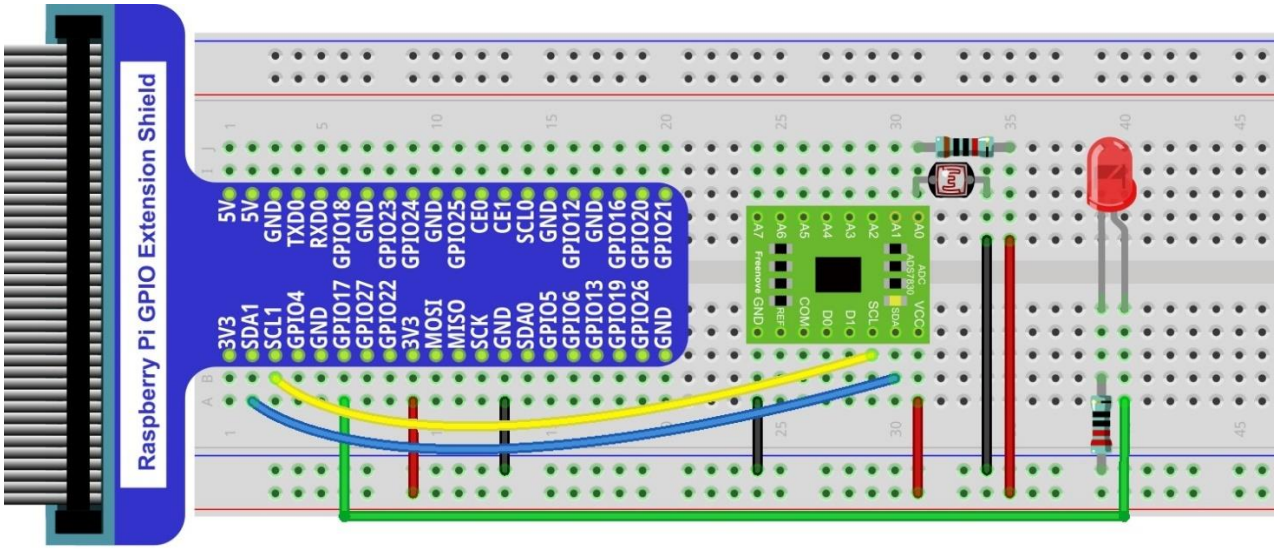
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x15
  </td></tr>
<tr><td>Photoresistor x1
  </td><td>ADC module x1
 
</td><td>10kΩ x3
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this project, we will change the brightness of the LED based on the light intensity received by the photoresistor.

Sketch_08_Nightlamp

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_08_Nightlamp
```

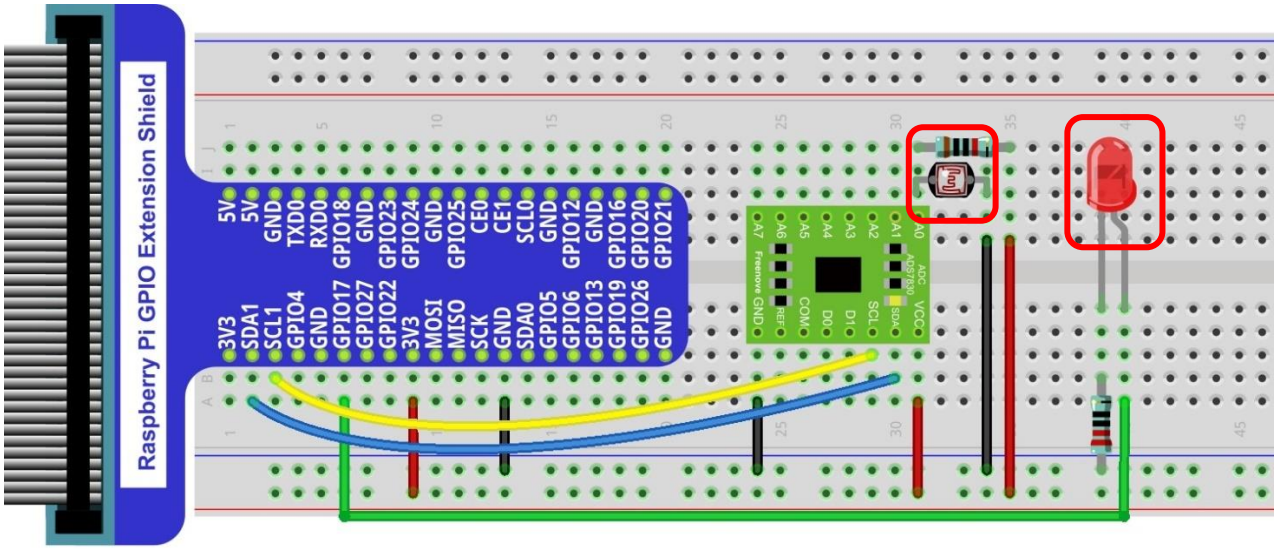
```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_08_Nightlamp
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_08_Nightlamp $
```

Enter the command to run the code.

```
jbang Nightlamp.java
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_08_Nightlamp
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_08_Nightlamp $ jbang Nightlamp.java
```

When the code is running, use light to illuminate the photosensitive module, or cover the photosensitive module with your hand, and you can observe that the brightness of the LED on the board changes accordingly.



On the terminal, you can see the ADC and voltage values of the photoresistor are printed.

```
[main] INFO com.pi4j.util.Console - ADC value:186, Voltage:3.65V
[main] INFO com.pi4j.util.Console - ADC value:183, Voltage:3.59V
[main] INFO com.pi4j.util.Console - ADC value:177, Voltage:3.47V
[main] INFO com.pi4j.util.Console - ADC value:168, Voltage:3.29V
```

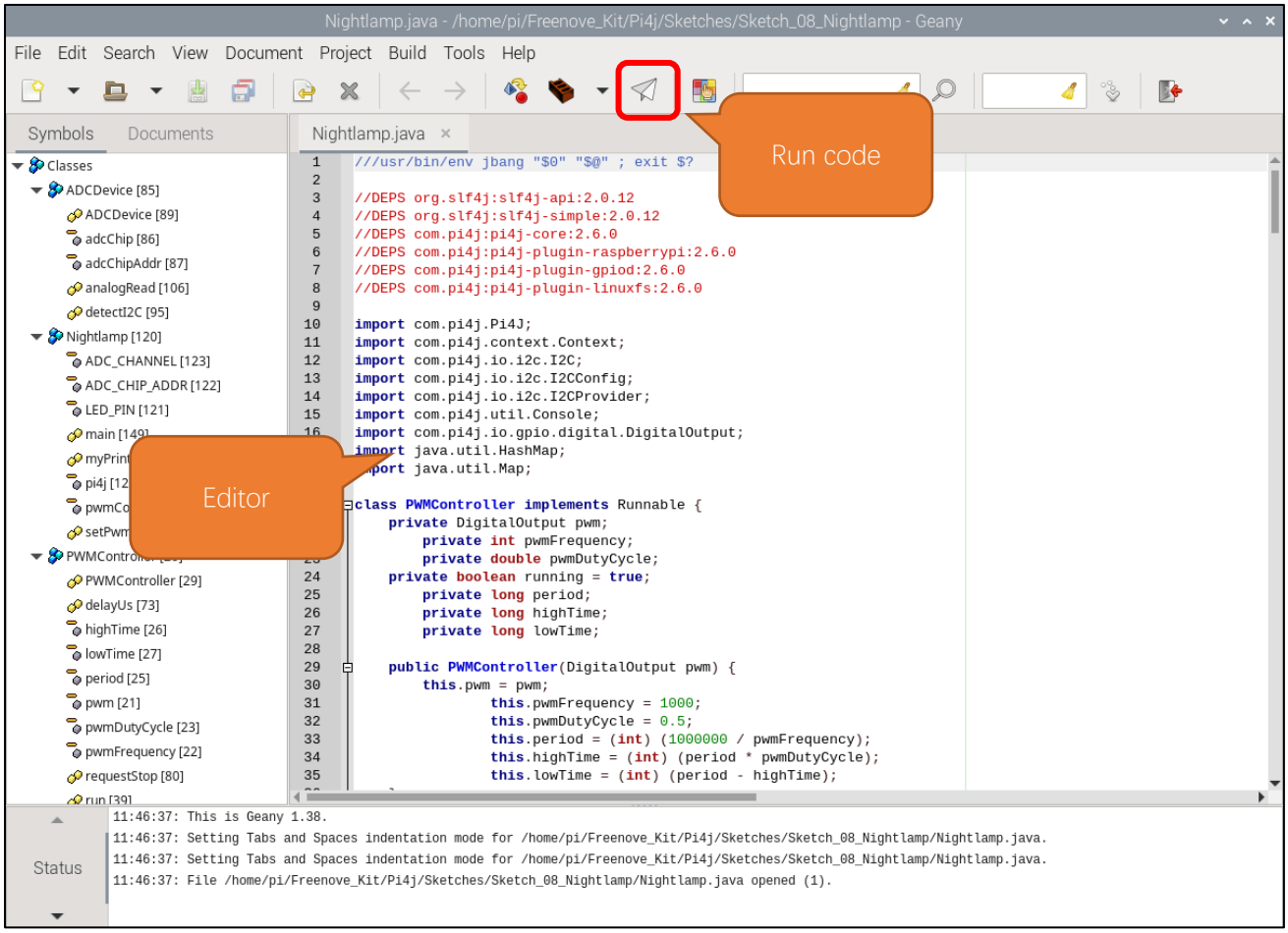
Press Ctrl+C to exit the program.

```
[main] INFO com.pi4j.util.Console - ADC value:176, Voltage:3.45V
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shutdown event.
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shutdown event.
```

You can open the code with Geany with the following command to view and edit it.

```
geany Nightlamp.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
```

```
15  import com.pi4j.util.Console;
16  import com.pi4j.io.gpio.digital.DigitalOutput;
17  import java.util.HashMap;
18  import java.util.Map;
19
20  class PWMController implements Runnable {
...     ......
83  }
84
85  class ADCDevice {
...     ......
118 }
119
120 public class Nightlamp{
121     private static int   LED_PIN = 17;
122     private static int   ADC_CHIP_ADDR = 0x48;
123     private static int   ADC_CHANNEL = 0;
124
125     private static final Context pi4j = Pi4J.newAutoContext();
126     private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
127
128     public static void setPwmConfig(int pin) throws Exception {
129         DigitalOutput pwm = pi4j.dout().create(pin);
130         PWMController pwmController = new PWMController(pwm);
131         Thread pwmThread = new Thread(pwmController, "PWM Controller " + pin);
132         pwmControllers.put(pin, pwmController);
133         pwmThread.start();
134         Runtime.getRuntime().addShutdownHook(new Thread(() -> {
135             pwmController.requestStop();
136             try {
137                 pwmThread.join();
138             } catch (InterruptedException e) {
139                 Thread.currentThread().interrupt();
140             }
141         }));
142     }
143
144     public static void myPrintln(String format, Object... args) {
145         Console console = new Console();
146         console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
147     }
148
149     public static void main(String[] args) throws Exception {
150         Context pi4j = Pi4J.newAutoContext();
```

```
151         I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
152         setPwmConfig(LED_PIN);
153         PWMController led = pwmControllers.get(LED_PIN);
154
155         try {
156             ADCDevice adc = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
157             if (adc.detectI2C()) {
158                 while (true) {
159                     int adcValue = adc.analogRead(ADC_CHANNEL);
160                     if (adcValue != -1) {
161                         led.setPwmDutyCycle(((double)adcValue/255.0));
162                         double voltage = (double)adcValue / 255.0 * 5.0;
163                         myPrintln("ADC value:%d, Voltage:%.2fV", adcValue, voltage);
164                     } else {
165                         myPrintln("Failed to read data from ADC.");
166                     }
167                     Thread.sleep(100);
168                 }
169             } else {
170                 myPrintln("ADS7830 device not detected at address 0x" +
                    Integer.toHexString(ADC_CHIP_ADDR));
171             }
172         } finally {
173             pi4j.shutdown();
174         }
175     }
176 }
```

The ADC value at the photosensor is obtained every 100 milliseconds, converted into a PWM duty cycle value for controlling the LED, and then a prompt message is printed on the terminal.

```
                while (true) {
                    int adcValue = adc.analogRead(ADC_CHANNEL);
                    if (adcValue != -1) {
                        led.setPwmDutyCycle(((double)adcValue/255.0));
                        double voltage = (double)adcValue / 255.0 * 5.0;
                        myPrintln("ADC value:%d, Voltage:%.2fV", adcValue, voltage);
                    } else {
                        myPrintln("Failed to read data from ADC.");
                    }
                    Thread.sleep(100);
                }
```

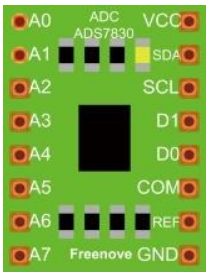
Chapter 9 Thermistor

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

Project 9.1 Thermometer

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

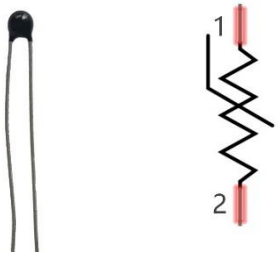
Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire M/M x14 	
Thermistor x1 	ADC module x1 	Resistor 10kΩ x3 

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;

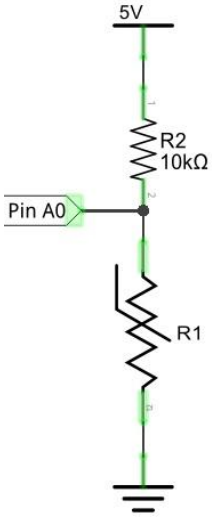
EXP[n] is nth power of e;

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:



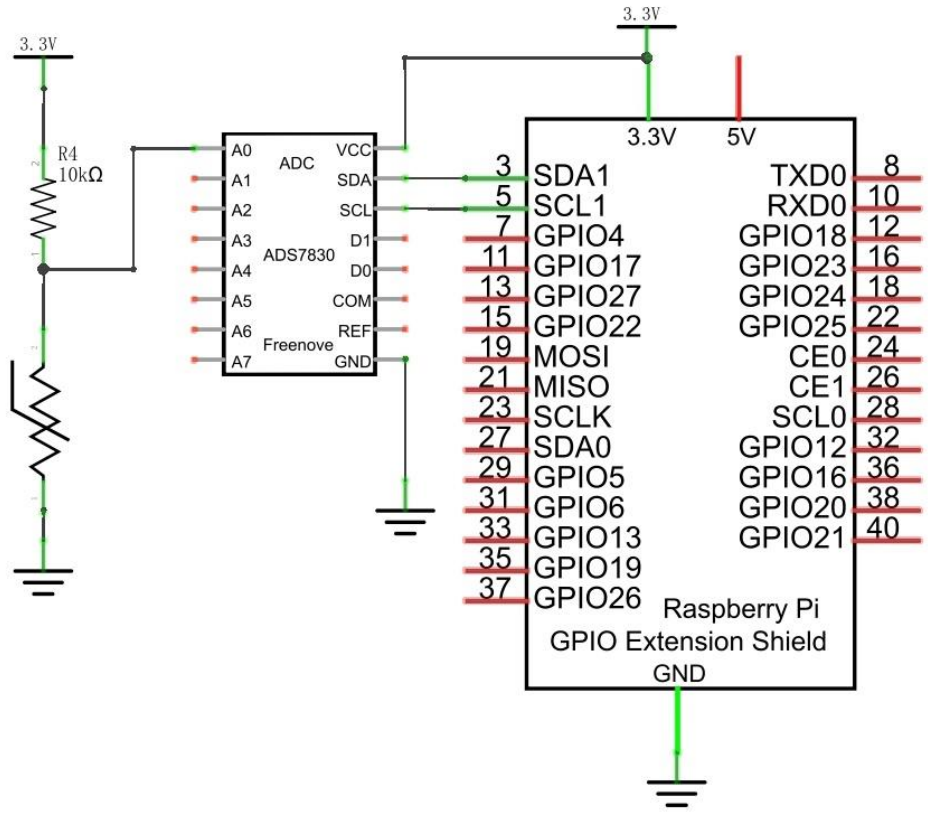
We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

Therefore, the temperature formula can be derived as:

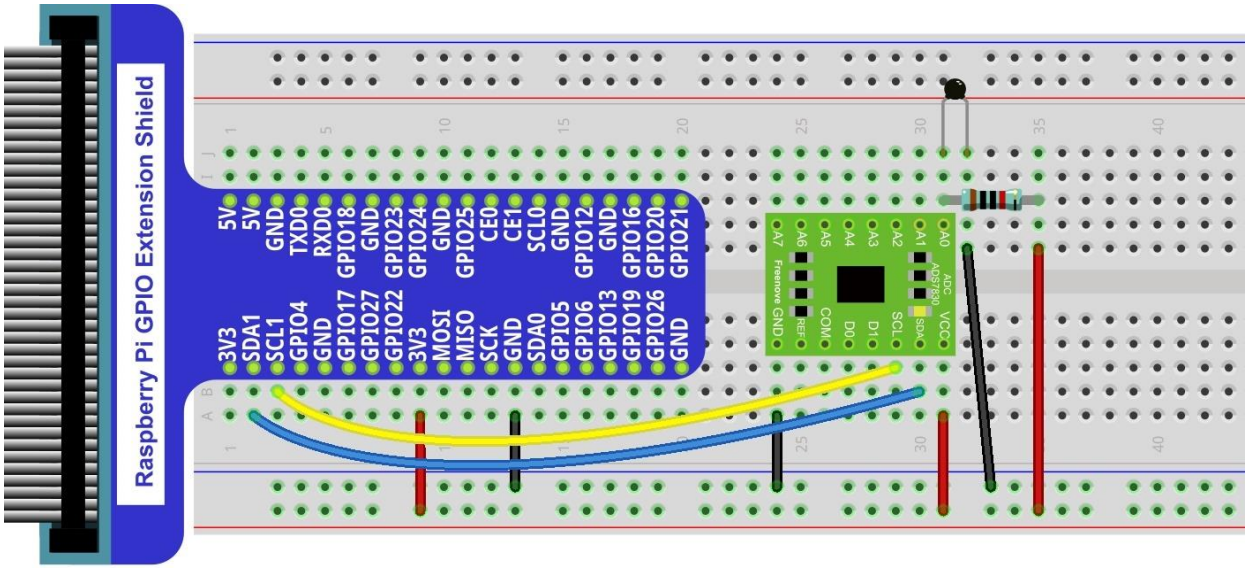
$$T2 = 1/(1/T1 + \ln(Rt/R)/B)$$

Circuit

Schematic diagram



Hardware connection.



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this project, we will collect the ADC value of the thermistor and calculate its temperature.

Sketch_09_Thermometer

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer

```
pi@raspberrypi:~ $ cd Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer $
```

Enter the command to run the code.

jbang Thermometer.java

```
pi@raspberrypi:~ $ cd Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer/
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer $ jbang Thermometer.java
```

After running the code, the Raspberry Pi will obtain the ADC value of the thermistor and convert it into voltage and temperature values, as shown in the figure below.

```
[main] INFO com.pi4j.util.Console - ADC value:117, Voltage:2.29V, Temperature:28.76C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[main] INFO com.pi4j.util.Console - ADC value:117, Voltage:2.29V, Temperature:28.76C
```

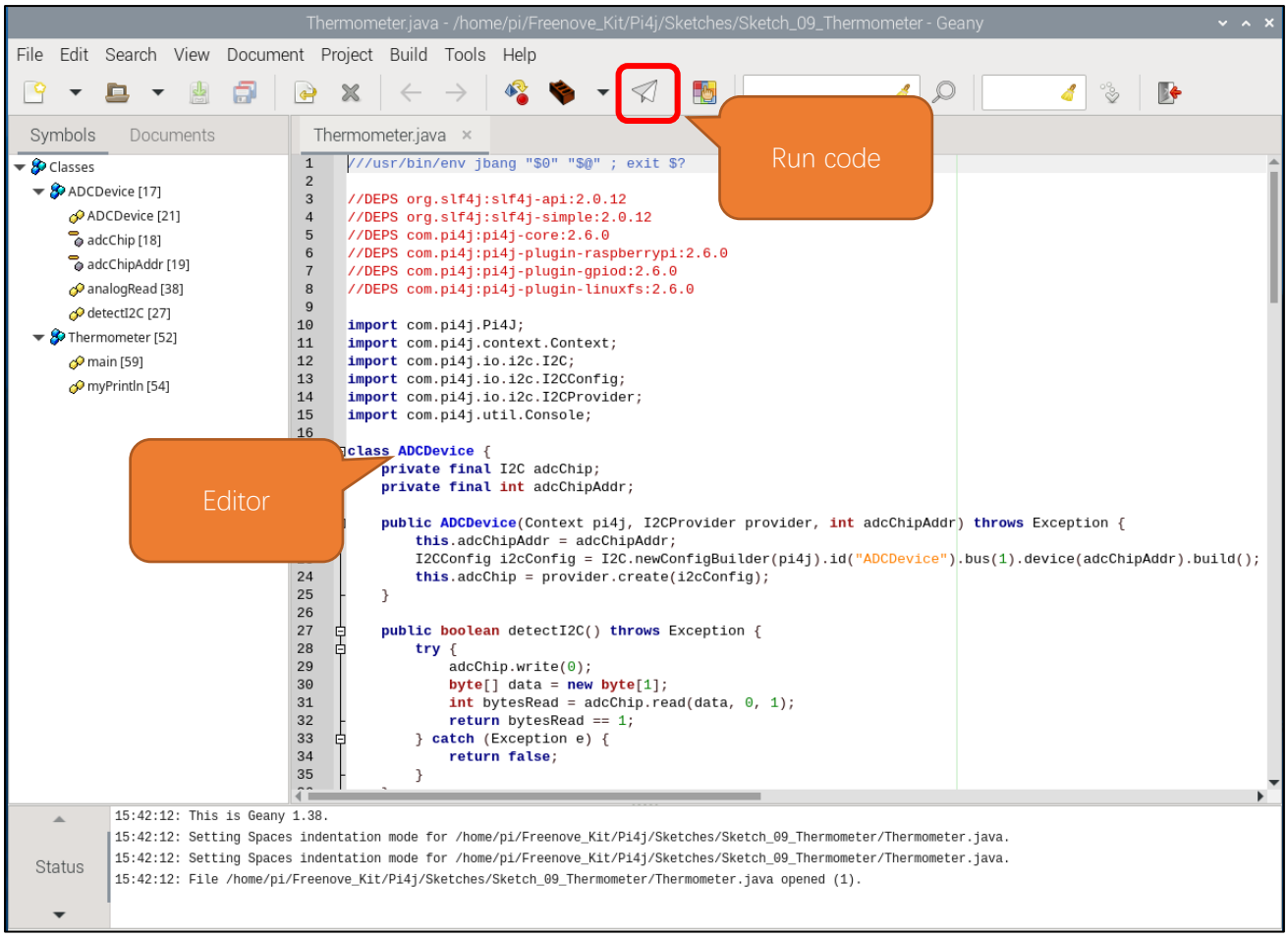
Press Ctrl+C to exit the program.

```
pi@raspberrypi: ~/Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer
File  Edit  Tabs  Help
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:115, Voltage:2.25V, Temperature:29.49C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[main] INFO com.pi4j.util.Console - ADC value:116, Voltage:2.27V, Temperature:29.13C
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdo
wn. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_09_Thermometer $
```

You can open the code with Geany to view and edit it.

geany Thermometer.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
import com.pi4j.util.Console;

class ADCDevice {
```

```
...     ......
50  }
51
52  public class Thermometer{
53      public static void myPrintln(String format, Object... args) {
54          Console console = new Console();
55          console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
56      }
57
58      public static void main(String[] args) throws Exception {
59          Context pi4j = Pi4J.newAutoContext();
60          I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
61          try {
62              int ADC_CHIP_ADDR = 0x48;
63              ADCDevice adcDevice = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
64              if (adcDevice.detectI2C()) {
65                  int ADC_CHANNEL = 0;
66                  while (true) {
67                      int adcValue = adcDevice.analogRead(ADC_CHANNEL);
68                      if (adcValue != -1) {
69                          double voltage = (double)adcValue / 255.0 * 5.0; // calculate voltage
70                          double Rt = 10 * voltage / (5.0 - voltage);      //calculate
    resistance value of thermistor
71                          double tempK = 1/(1/(273.15 + 25) + Math.log(Rt/10)/3950.0);
    //calculate temperature (Kelvin)
72                          double tempC = tempK -273.15;        //calculate temperature (Celsius)
73                          myPrintln("ADC value:%d, Voltage:%.2fV, Temperature:%.2fC", adcValue,
    voltage, tempC);
74                      } else {
75                          myPrintln("Failed to read data from ADC.");
76                      }
77                      Thread.sleep(100);
78                  }
79              } else {
80                  myPrintln("ADS7830 device not detected at address 0x" +
    Integer.toHexString(ADC_CHIP_ADDR));
81              }
82          } finally {
83              pi4j.shutdown();
84          }
85      }
86  }
```

Get the ADC value of the thermistor every 100 milliseconds, convert the ADC value into a temperature value according to the formula introduced at component knowledge section, and print the relevant information on the terminal.

```
int ADC_CHANNEL = 0;
while (true) {
    int adcValue = adcDevice.analogRead(ADC_CHANNEL);
    if (adcValue != -1) {
        double voltage = (double)adcValue / 255.0 * 5.0; // calculate voltage
        double Rt = 10 * voltage / (5.0 - voltage);      //calculate resistance value of
thermistor
        double tempK = 1/(1/(273.15 + 25) + Math.log(Rt/10)/3950.0); //calculate temperature
(Kelvin)
        double tempC = tempK -273.15;         //calculate temperature (Celsius)
        myPrintln("ADC value:%d, Voltage:%.2fV, Temperature:%.2fC", adcValue, voltage, tempC);
    } else {
        myPrintln("Failed to read data from ADC.");
    }
    Thread.sleep(100);
}
```

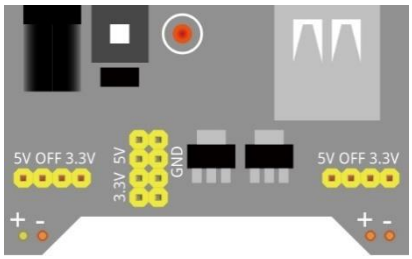
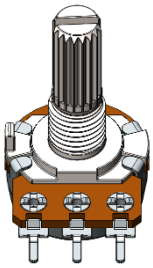
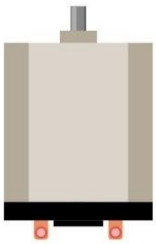
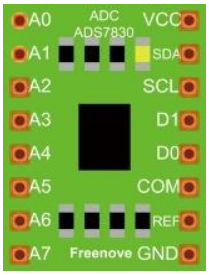
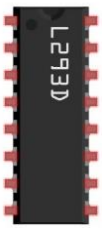
Chapter 10 Motor & Driver

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

Project 10.1 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.

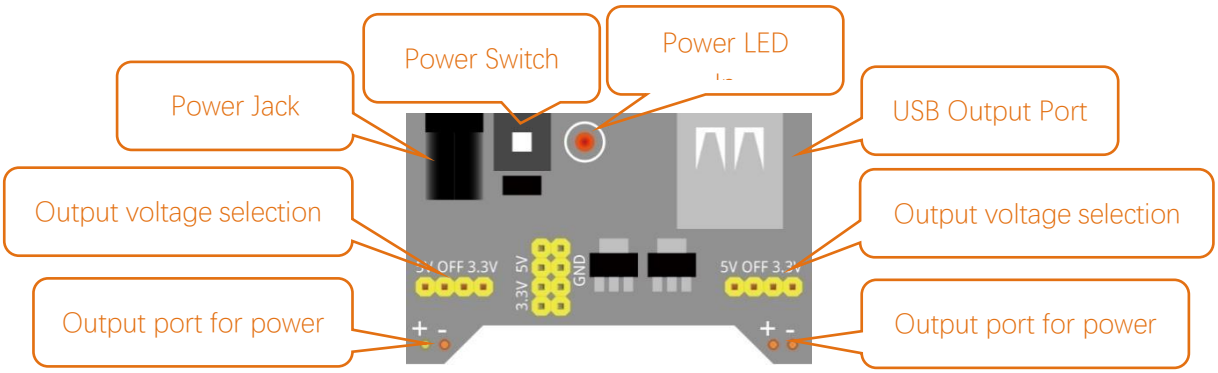
Component List

<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>ADC Module x1
 
</td><td>L293D IC Chip
 
</td></tr>
</table>

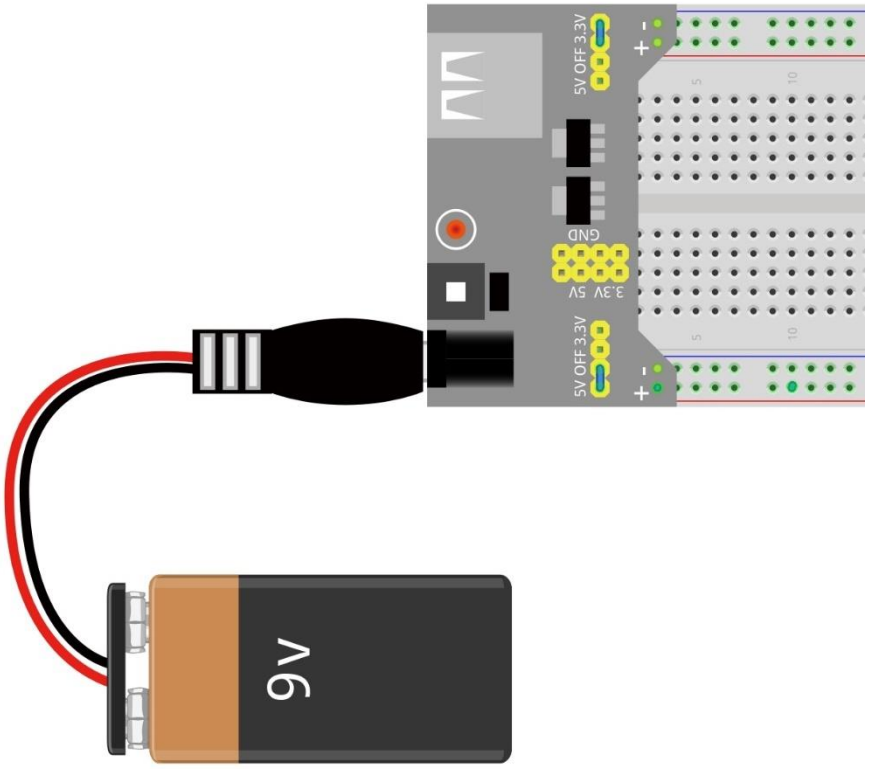
Component knowledge

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:

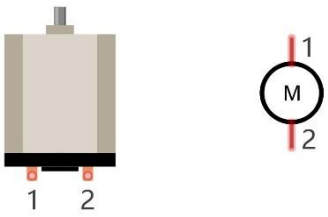


Here is an acceptable connection between Breadboard Power Module and Breadboard using a 9V battery and the provided power harness:



DC Motor

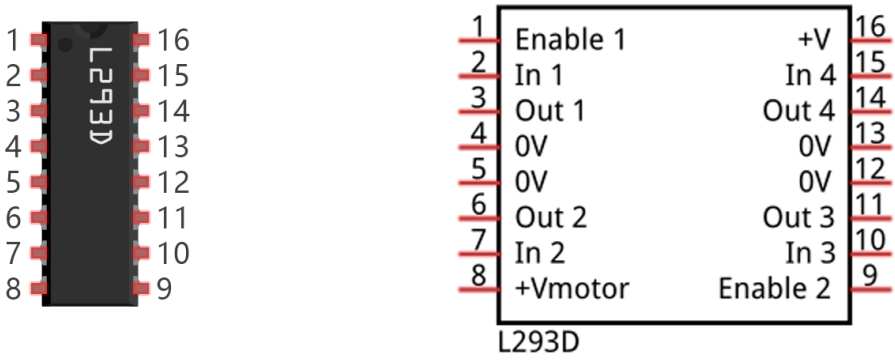
DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair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



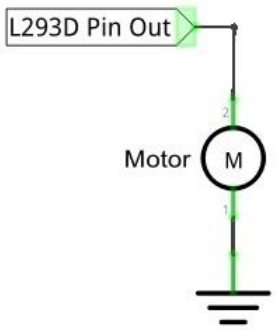
Port description of L293D module is as follows:

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

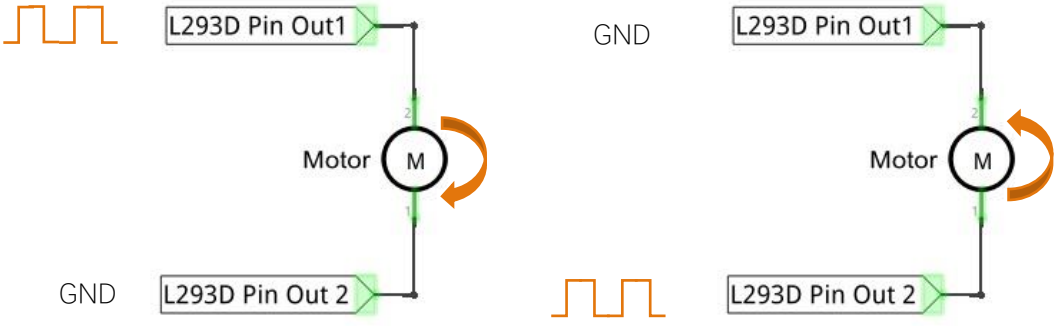
For more details, please see the datasheet for this IC Chip.

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.

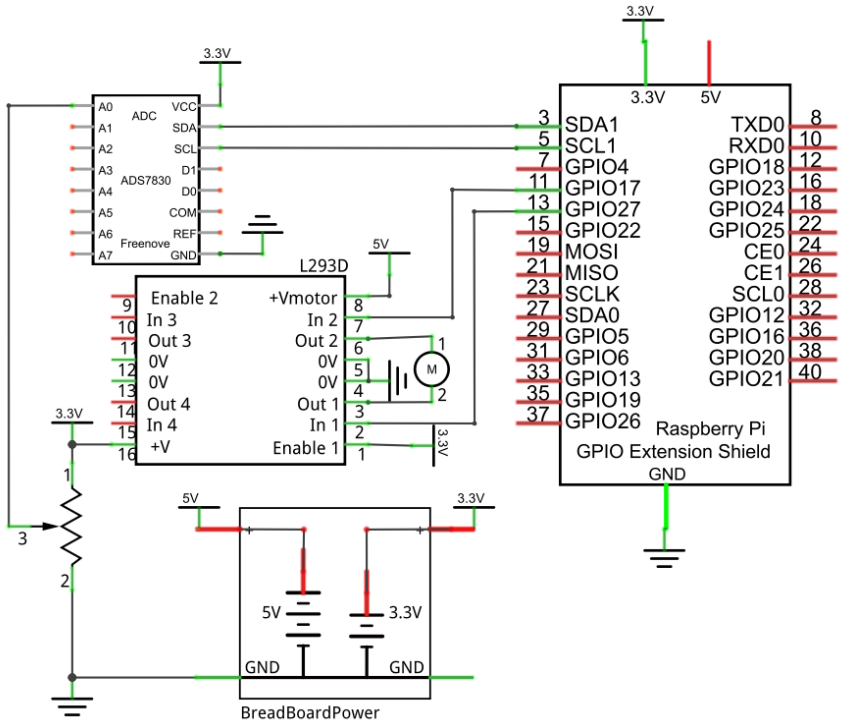


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

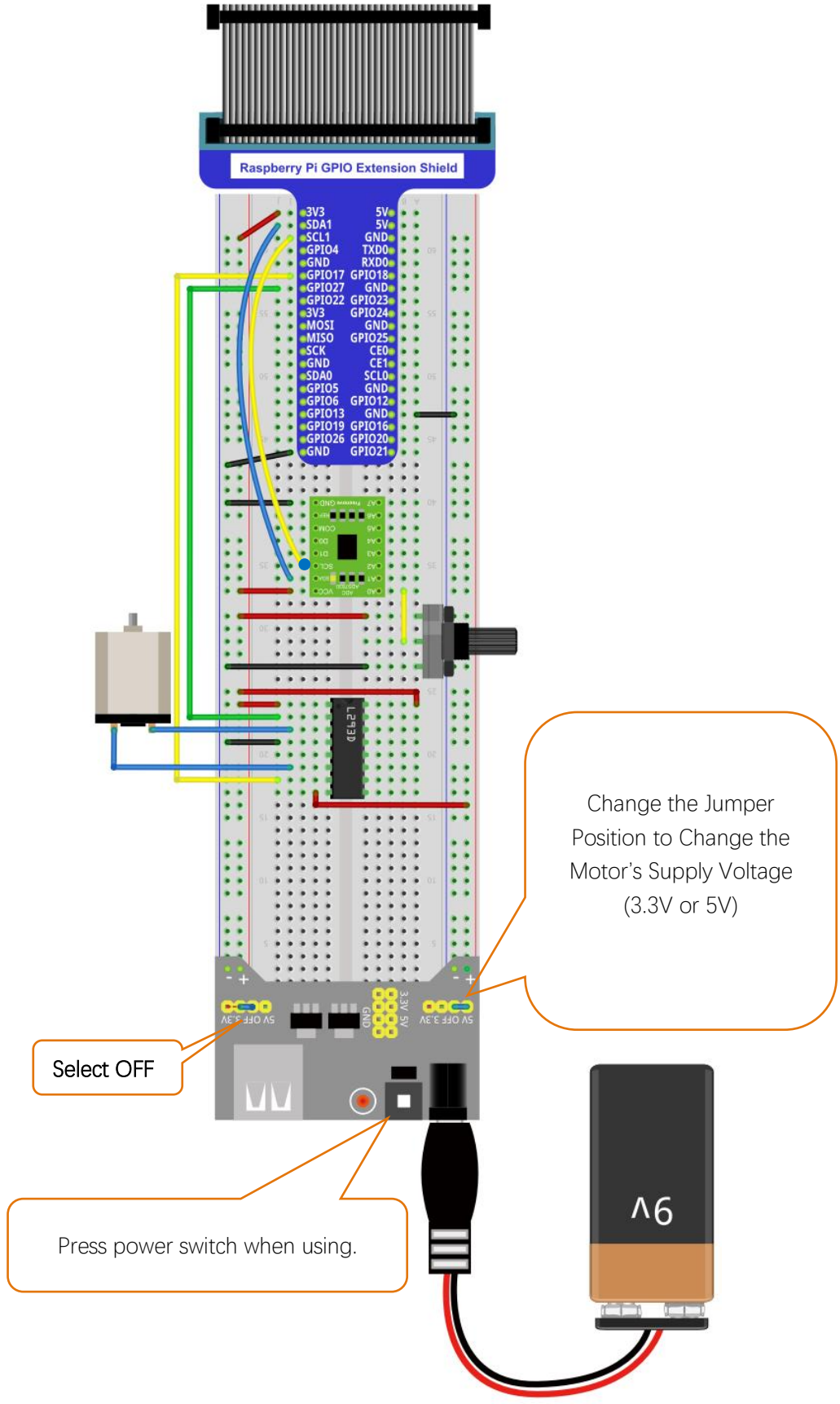
Circuit

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



Sketch

In this chapter, we will learn how to use the potentiometer to control the rotation speed and direction of the DC motor.

Sketch_10_Motor

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_10_Motor

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_11_Motor
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_11_Motor $
```

Enter the command to run the code.

jbang Motor.java

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_11_Motor
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_11_Motor $ jbang Motor.java
```

When the code is running, Raspberry Pi will obtain the value of the potentiometer and control the direct of the motor accordingly.

```
[main] INFO com.pi4j.util.Console - ADC value : 129
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 0%

[main] INFO com.pi4j.util.Console - ADC value : 133
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 3%

[main] INFO com.pi4j.util.Console - ADC value : 134
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 4%

[main] INFO com.pi4j.util.Console - ADC value : 133
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 3%

[main] INFO com.pi4j.util.Console - ADC value : 105
[main] INFO com.pi4j.util.Console - turn Back...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 18%
```

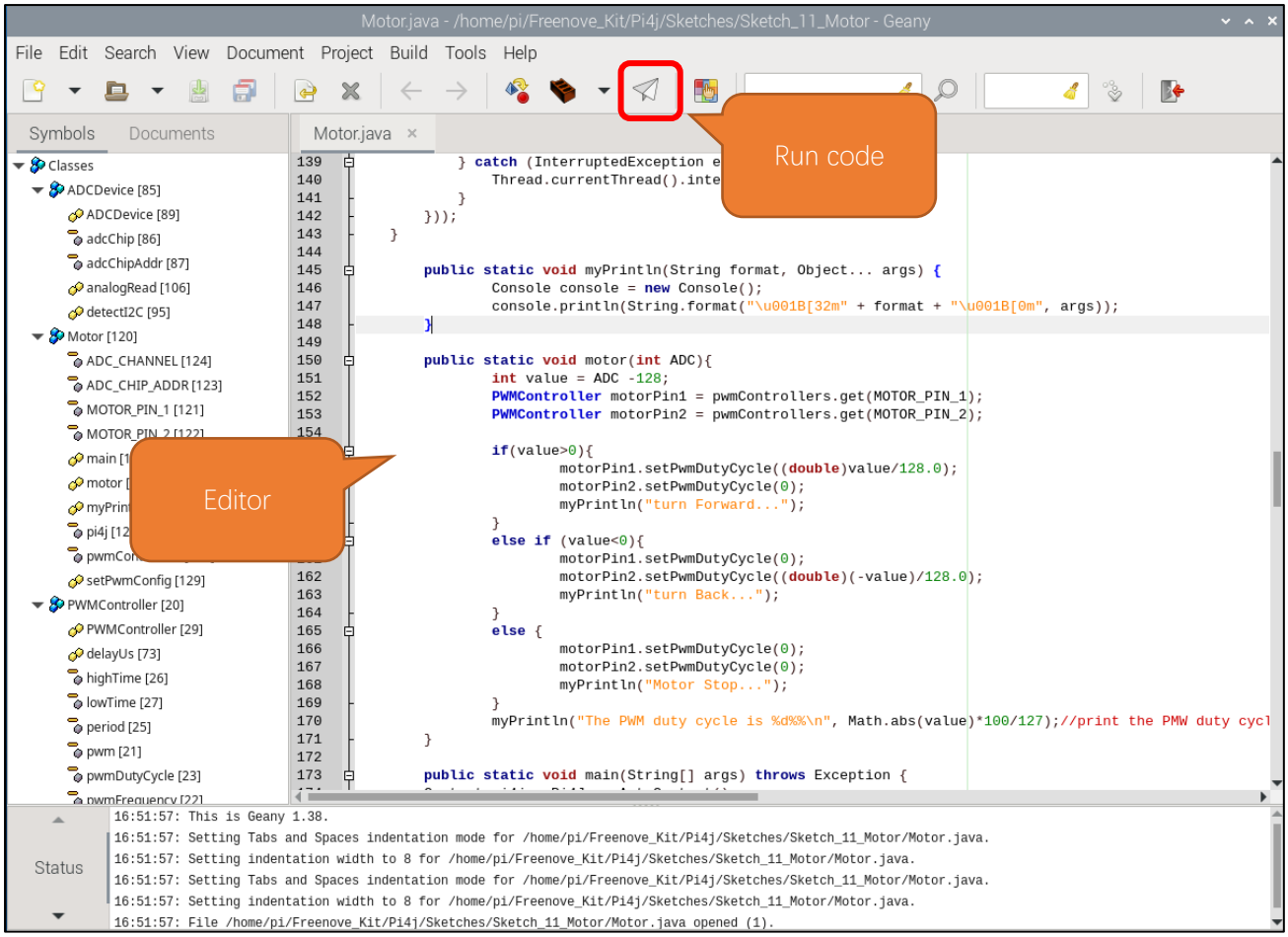
Press Ctrl+C to exit the code.

```
[main] INFO com.pi4j.util.Console - ADC value : 136
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 6%

[main] INFO com.pi4j.util.Console - ADC value : 136
[main] INFO com.pi4j.util.Console - turn Forward...
[main] INFO com.pi4j.util.Console - The PWM duty cycle is 6%

^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shutdown event.
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdown. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_11_Motor $
```

You can run the following command to open the code with Geany to view and edit it.

geany Motor.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CProvider;
import com.pi4j.util.Console;
import com.pi4j.io.gpio.digital.DigitalOutput;
import java.util.HashMap;
```

```
51  import java.util.Map;
52
53  class PWMController implements Runnable {
...     ......
83  }
84
85  class ADCDevice {
...     ......
118 }
119
120 public class Motor{
121     private static int MOTOR_PIN_1 = 27;
122     private static int MOTOR_PIN_2 = 17;
123     private static int ADC_CHIP_ADDR = 0x48;
124     private static int ADC_CHANNEL = 0;
125     private static final Context pi4j = Pi4J.newAutoContext();
126     private static final Map<Integer, PWMController> pwmControllers = new HashMap<>();
127
128     public static void setPwmConfig(int pin) throws Exception {
129         DigitalOutput pwm = pi4j.dout().create(pin);
130         PWMController pwmController = new PWMController(pwm);
131         Thread pwmThread = new Thread(pwmController, "PWM Controller " + pin);
132         pwmControllers.put(pin, pwmController);
133         pwmThread.start();
134         Runtime.getRuntime().addShutdownHook(new Thread(() -> {
135             pwmController.requestStop();
136             try {
137                 pwmThread.join();
138             } catch (InterruptedException e) {
139                 Thread.currentThread().interrupt();
140             }
141         }));
142     }
143
144     public static void myPrintln(String format, Object... args) {
145         Console console = new Console();
146         console.println(String.format("\u001B[32m" + format + "\u001B[0m", args));
147     }
148
149     public static void motor(int ADC){
150         int value = ADC -128;
151         PWMController motorPin1 = pwmControllers.get(MOTOR_PIN_1);
152         PWMController motorPin2 = pwmControllers.get(MOTOR_PIN_2);
153
```

```
154             if(value>0){
155                 motorPin1.setPwmDutyCycle((double)value/128.0);
156                 motorPin2.setPwmDutyCycle(0);
157                 myPrintln("turn Forward...");
158             }
159             else if (value<0){
160                 motorPin1.setPwmDutyCycle(0);
161                 motorPin2.setPwmDutyCycle((double)(-value)/128.0);
162                 myPrintln("turn Back...");
163             }
164             else {
165                 motorPin1.setPwmDutyCycle(0);
166                 motorPin2.setPwmDutyCycle(0);
167                 myPrintln("Motor Stop...");
168             }
169             myPrintln("The PWM duty cycle is %d%%\n", Math.abs(value)*100/127);
170         }
171 
172         public static void main(String[] args) throws Exception {
173             Context pi4j = Pi4J.newAutoContext();
174             I2CProvider i2CProvider = pi4j.provider("linuxfs-i2c");
175             setPwmConfig(MOTOR_PIN_1);
176             setPwmConfig(MOTOR_PIN_2);
177             try {
178                 ADCDevice adc = new ADCDevice(pi4j, i2CProvider, ADC_CHIP_ADDR);
179                 if (adc.detectI2C()) {
180                     while (true) {
181                         int adcValue = adc.analogRead(ADC_CHANNEL);
182                         myPrintln("ADC value : %d", adcValue);
183                         motor(adcValue);
184                         Thread.sleep(100);
185                     }
186                 } else {
187                     myPrintln("ADS7830 device not detected at address 0x" +
188                     Integer.toHexString(ADC_CHIP_ADDR));
189                 }
190             } finally {
191                 pi4j.shutdown();
192             }
193         }
194 }
```

The range of ADC value is 0-255, with 128 as the middle value. If the ADC value is greater than 128, the Raspberry Pi controls the motor to rotate forward. The larger the ADC value, the faster the motor speed. Conversely, if the ADC value is less than 128, the Raspberry Pi controls the motor to rotate backward. The smaller the ADC value, the faster the motor speed.

```
    public static void motor(int ADC){
        int value = ADC -128;
        PWMController motorPin1 = pwmControllers.get(MOTOR_PIN_1);
        PWMController motorPin2 = pwmControllers.get(MOTOR_PIN_2);

        if(value>0){
            motorPin1.setPwmDutyCycle((double)value/128.0);
            motorPin2.setPwmDutyCycle(0);
            myPrintln("turn Forward...");
        }
        else if (value<0){
            motorPin1.setPwmDutyCycle(0);
            motorPin2.setPwmDutyCycle((double)(-value)/128.0);
            myPrintln("turn Back...");
        }
        else {
            motorPin1.setPwmDutyCycle(0);
            motorPin2.setPwmDutyCycle(0);
            myPrintln("Motor Stop...");
        }
        myPrintln("The PWM duty cycle is %d%%\n", Math.abs(value)*100/127);
    }
```

The ADC value at the potentiometer is obtained every 100 milliseconds, and the ADC value is sent as a parameter to the motor function to control the direction and speed of the motor, and the ADC value is printed out in the terminal.

```
        while (true) {
            int adcValue = adc.analogRead(ADC_CHANNEL);
            myPrintln("ADC value : %d", adcValue);
            motor(adcValue);
            Thread.sleep(100);
        }
```

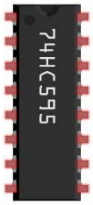
Chapter 11 74HC595 & Bar Graph LED

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 11.1 Flowing Water Light

Now let us learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

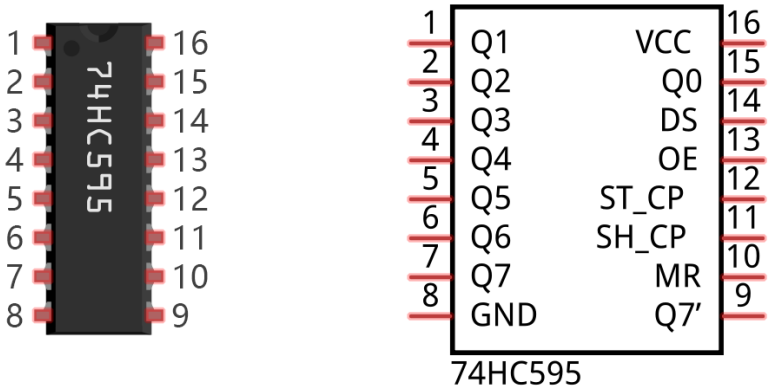
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper x17 	
74HC595 x2 	Bar Graph LED x1 	Resistor 220Ω x8 

Component knowledge

74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of a Raspberry Pi. At least 3 ports on the RPI board are required to control the 8 ports of the 74HC595 chip.



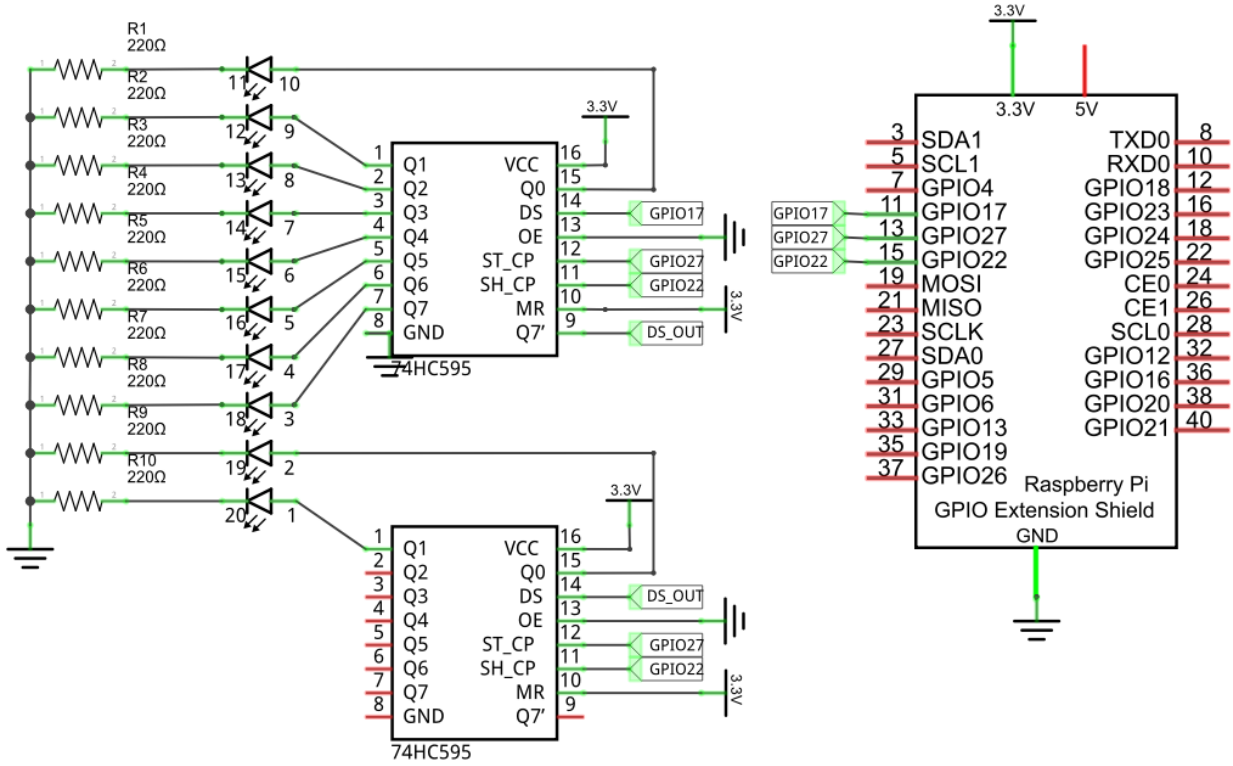
The ports of the 74HC595 chip are described as follows:

Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel Data Output
VCC	16	The Positive Electrode of the Power Supply, the Voltage is 2~6V
GND	8	The Negative Electrode of Power Supply
DS	14	Serial Data Input
OE	13	Enable Output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel Update Output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial Shift Clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove Shift Register: When this pin is in low level, the content in shift register will be cleared.
Q7'	9	Serial Data Output: it can be connected to more 74HC595 chips in series.

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

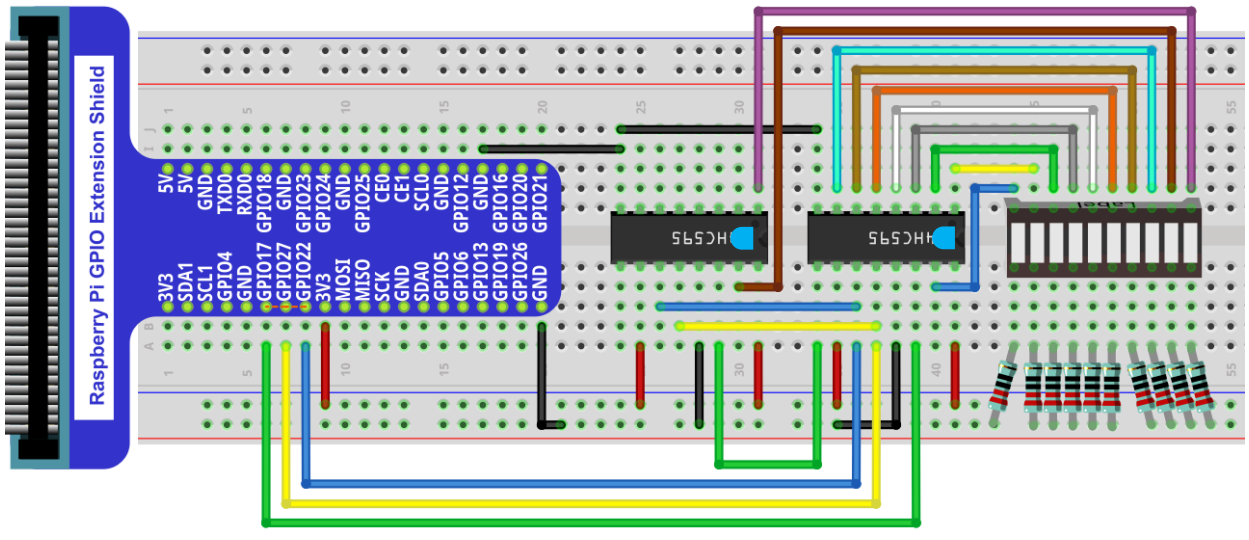
Circuit

Schematic diagram



Hardware connection.

If it dosen't work, rotate the LED bar graph for 180°.



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this chapter, we will learn how to drive the LED Bar by expanding the chip.

Sketch_11_FlowingLight02

First, enter where the project is located:

cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_11_FlowingLight02

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_15_FlowingLight02
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_15_FlowingLight02 $
```

Enter the command to run the code.

jbang FlowingLight02.java

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_15_FlowingLight02
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_15_FlowingLight02 $ jbang FlowingLight02.java
```

When the code is running, you can see the LEDs of the LED bar light up in a flowing water pattern.

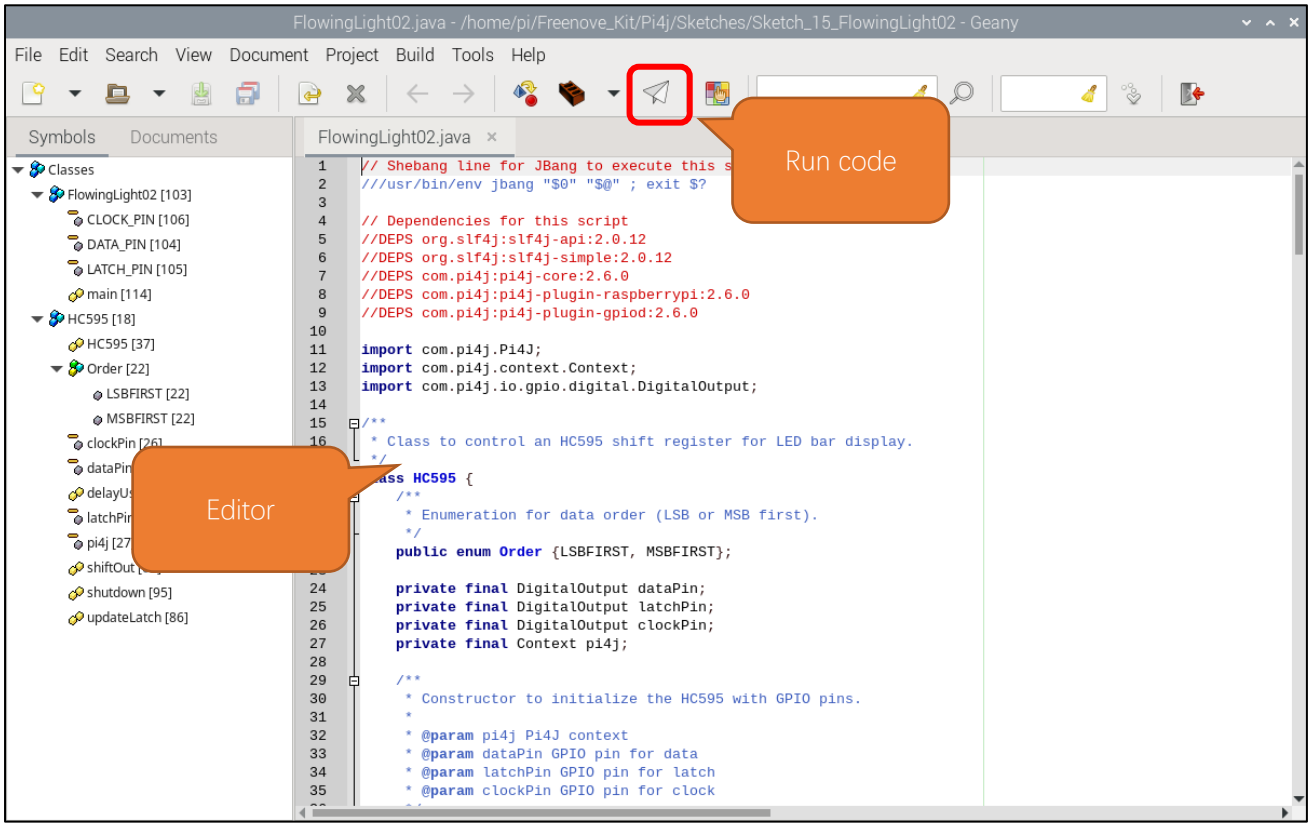
Press Ctrl+C to exit the program.

```
[main] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully initialized.
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdo
wn. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_15_FlowingLight02 $
```

You can run the following command to open the code with Geany to view and edit it.

geany FlowingLight02.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.gpio.digital.DigitalOutput;

class HC595 {
    public enum Order {LSBFIRST, MSBFIRST};
    private final DigitalOutput dataPin;
    private final DigitalOutput latchPin;
    private final DigitalOutput clockPin;
    private final Context pi4j;

    public HC595(Context pi4j, int dataPin, int latchPin, int clockPin) {
        this.pi4j = pi4j;
        this.dataPin = pi4j.dout().create(dataPin);
        this.latchPin = pi4j.dout().create(latchPin);
        this.clockPin = pi4j.dout().create(clockPin);
    }

    private static void delayUs(long us) {
        long startTime = System.nanoTime();
        long endTime = startTime + (us * 1000);
        while (System.nanoTime() < endTime) {
        }
    }

    public void shiftOut(Order order, int val) {
        int i;
        for (i = 0; i < 8; i++) {
            clockPin.low();
            if (order == Order.LSBFIRST) {
                if ((0x01 & (val >> i)) == 0x01) {
                    dataPin.high();
                } else {
                    dataPin.low();
                }
```

```
            } else {
                if ((0x80 & (val << i)) == 0x80) {
                    dataPin.high();
                } else {
                    dataPin.low();
                }
            }
            clockPin.high();
        }
    }

    public void updateLatch() {
        latchPin.low();
        delayUs(1);
        latchPin.high();
    }

    public void shutdown() {
        pi4j.shutdown();
    }
}

public class FlowingLight02 {
    private static final int DATA_PIN = 17;
    private static final int LATCH_PIN = 27;
    private static final int CLOCK_PIN = 22;
    public static void main(String[] args) throws Exception {
        var pi4j = Pi4J.newAutoContext();
        HC595 ledbar = new HC595(pi4j, DATA_PIN, LATCH_PIN, CLOCK_PIN);
        try {
            int x;
            while (true) {
                x = 0x0001;
                for (int i = 0; i < 10; i++) {
                    ledbar.shiftOut(HC595.Order.MSBFIRST, (x >> 8) & 0xff);
                    ledbar.shiftOut(HC595.Order.MSBFIRST, x & 0xff);
                    ledbar.updateLatch();
                    x <<= 1;
                    Thread.sleep(100);
                }
                x = 0x0200;
                for (int i = 0; i < 10; i++) {
                    ledbar.shiftOut(HC595.Order.MSBFIRST, (x >> 8) & 0xff);
                    ledbar.shiftOut(HC595.Order.MSBFIRST, x & 0xff);
```

```
88                          ledbar.updateLatch();
89                          x >>= 1;
90                          Thread.sleep(100);
91                      }
92                  }
93              } finally {
94                  ledbar.shutdown();
95              }
96          }
97      }
```

Define the data transfer order for enumeration types.

```
    public enum Order {LSBFIRST, MSBFIRST};
```

Define the data pin, latch pin, clock pin, and Pi4j context.

```
    private final DigitalOutput dataPin;
    private final DigitalOutput latchPin;
    private final DigitalOutput clockPin;
    private final Context pi4j;
```

Constructor, initialize pins and context.

```
    public HC595(Context pi4j, int dataPin, int latchPin, int clockPin) {
        this.pi4j = pi4j;
        this.dataPin = pi4j.dout().create(dataPin);
        this.latchPin = pi4j.dout().create(latchPin);
        this.clockPin = pi4j.dout().create(clockPin);
    }
```

Delay function in microsecond.

```
    private static void delayUs(long us) {
        long startTime = System.nanoTime();
        long endTime = startTime + (us * 1000);
        while (System.nanoTime() < endTime) {
        }
    }
```

Shift function for expansion chip. The Raspberry Pi sends data to the extended chip through GPIO.

```
    public void shiftOut(Order order, int val) {
        int i;
        for (i = 0; i < 8; i++) {
            clockPin.low();
            if (order == Order.LSBFIRST) {
                if ((0x01 & (val >> i)) == 0x01) {
                    dataPin.high();
                } else {
                    dataPin.low();
                }
            } else {
                if ((0x80 & (val << i)) == 0x80) {
                    dataPin.high();
                } else {
                    dataPin.low();
                }
            }
            clockPin.high();
        }
    }
```

Update the expansion chip latch to let the expansion chip output the signal. Usually you need to first call the shiftOut function to input data to the expansion chip, and then call the updateLatch function to have the expansion chip output the signal level corresponding to the data.

```
    public void updateLatch() {
        latchPin.low();
        delayUs(1);
        latchPin.high();
    }
```

When Pi4j context is not used, shut it down to release resources.

```
    public void shutdown() {
        pi4j.shutdown();
    }
```

Define the pin number of the driver expansion chip.

```
    private static final int DATA_PIN = 17;
    private static final int LATCH_PIN = 27;
    private static final int CLOCK_PIN = 22;
```

Create a pi4j automatic context and create an HC595 instance.

```
        var pi4j = Pi4J.newAutoContext();
        HC595 ledbar = new HC595(pi4j, DATA_PIN, LATCH_PIN, CLOCK_PIN);
```

The Raspberry Pi controls the LED bar to flow from left to right and then from right to left.

```
            while (true) {
                x = 0x0001;
                for (int i = 0; i < 10; i++) {
                    ledbar.shiftOut(HC595.Order.MSBFIRST, (x >> 8) & 0xff);
                    ledbar.shiftOut(HC595.Order.MSBFIRST, x & 0xff);
                    ledbar.updateLatch();
                    x <<= 1;
                    Thread.sleep(100);
                }
                x = 0x0200;
                for (int i = 0; i < 10; i++) {
                    ledbar.shiftOut(HC595.Order.MSBFIRST, (x >> 8) & 0xff);
                    ledbar.shiftOut(HC595.Order.MSBFIRST, x & 0xff);
                    ledbar.updateLatch();
                    x >>= 1;
                    Thread.sleep(100);
                }
            }
```

Shutdown HC595 instance resources.

```
        finally {
            ledbar.shutdown();
        }
```

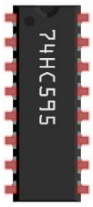
Chapter 12 74HC595 & LED Matrix

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 12.1 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.

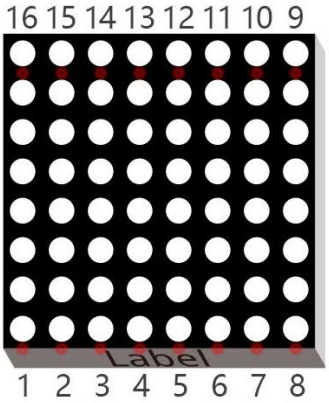
Component List

<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td>Jumper x36
  </td></tr>
<tr><td>74HC595 x2
  </td><td>8X8 LEDMatrix x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Component knowledge

LED matrix

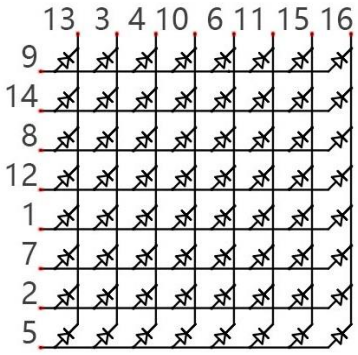
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



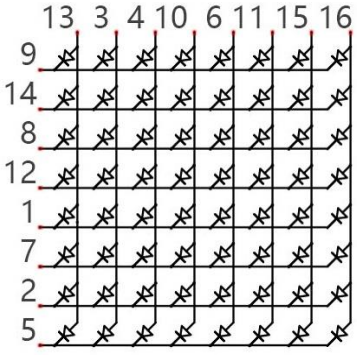
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

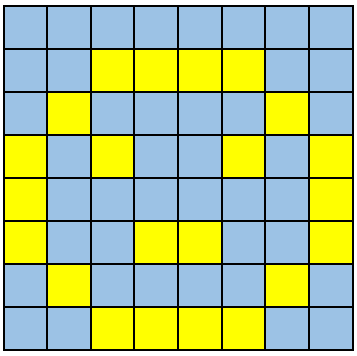
Connection mode of Common Anode



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

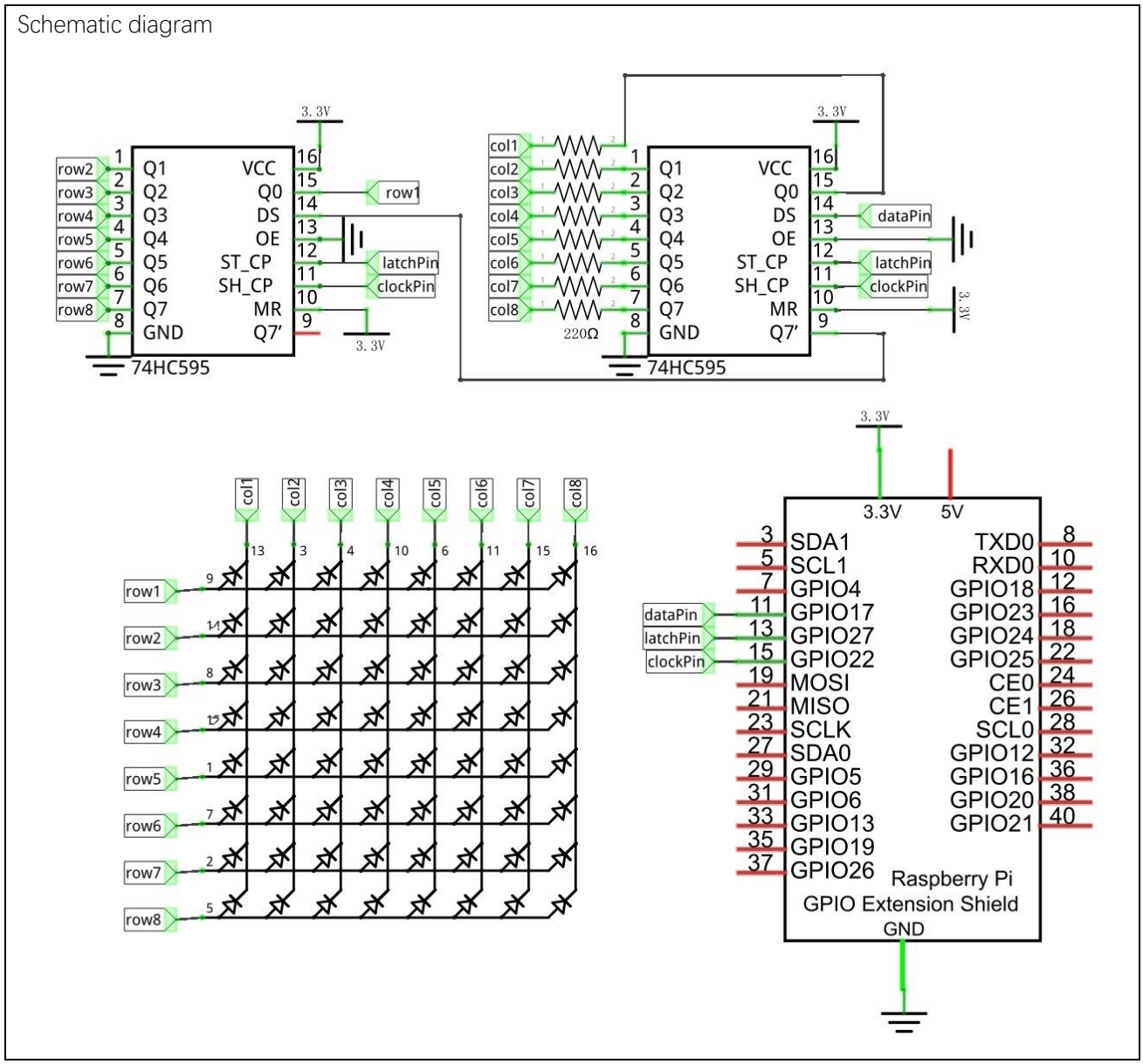
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated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

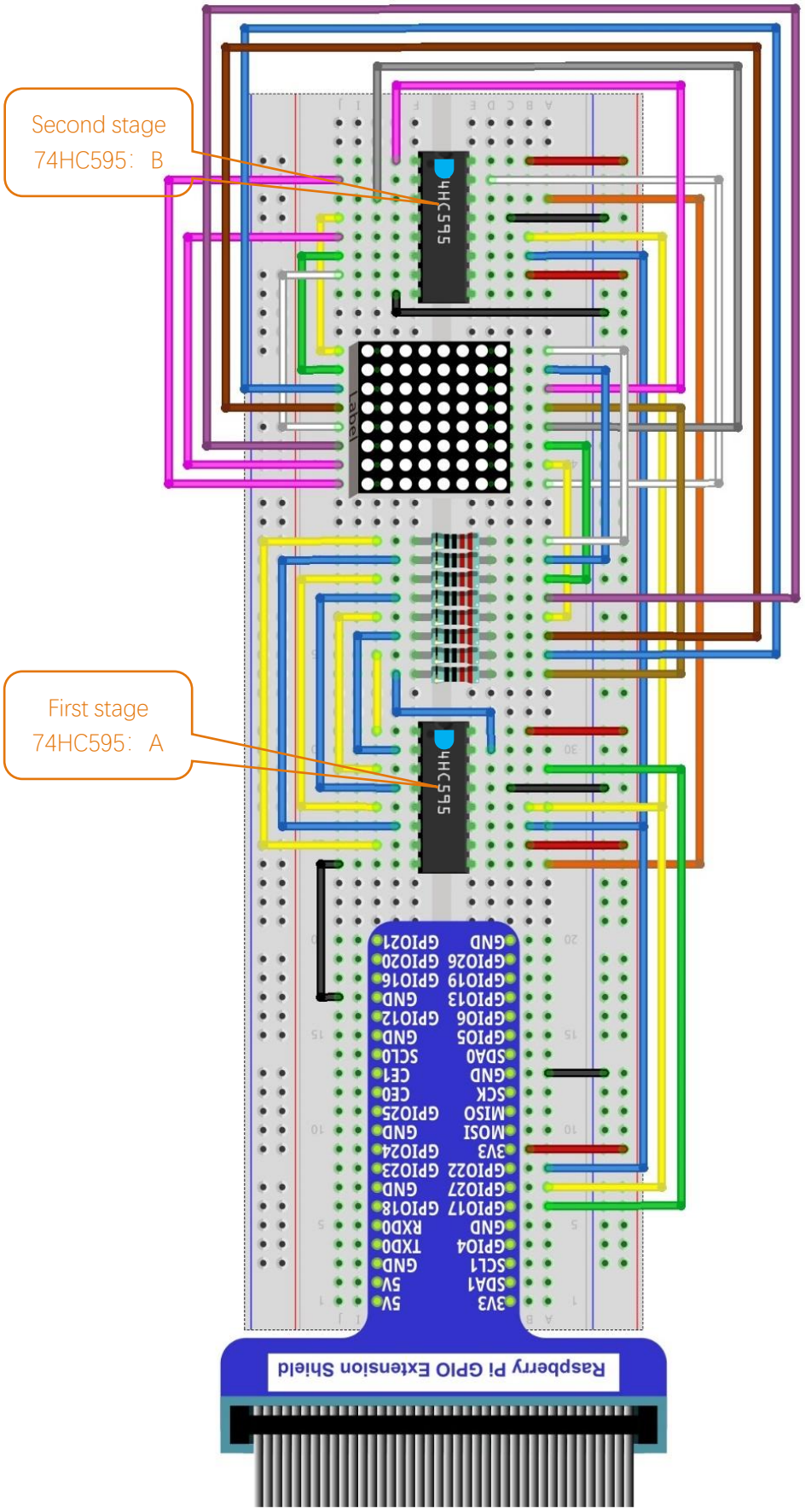
Circuit

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



Sketch

In this project, we will drive the LED matrix to display different contents.

Sketch_12_LEDMatrix

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_12_LEDMatrix
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_17_LEDMatrix
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_17_LEDMatrix $
```

Enter the command to run the code.

```
jbang LEDMatrix.java
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_17_LEDMatrix
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_17_LEDMatrix $ jbang LEDMatrix.java
```

When the code is running, you can see that the LED matrix first displays a smiley face, then scrolls through the characters 0-F, and repeats this process in a continuous loop.

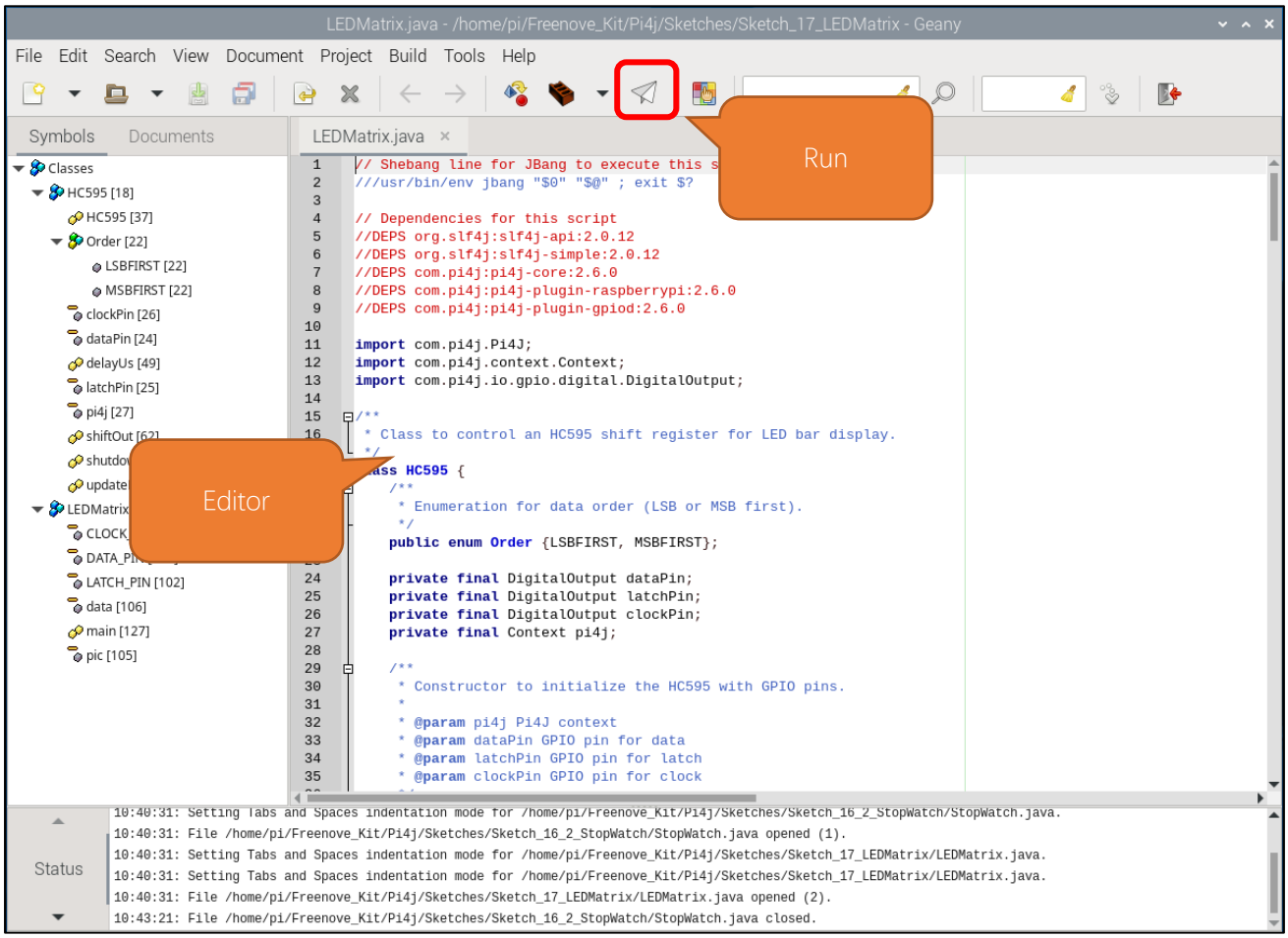
Press Ctrl+C to exit the program.

```
[main] INFO com.pi4j.library.gpiod.internal.GpioDContext - Using chip gpiochip4 pinctrl-rp1
[main] INFO com.pi4j.platform.impl.DefaultRuntimePlatforms - adding platform to managed platform ma
p [id=raspberrypi; name=RaspberryPi Platform; priority=5; class=com.pi4j.plugin.raspberrypi.platfor
m.RaspberryPiPlatform]
[main] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully initialized.
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutd
own. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_17_LEDMatrix $
```

You can run the following command to open the code with Geany to view and edit it.

geany LEDMatrix.java

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
1   ///usr/bin/env jbang "$0" "$@" ; exit $?
2
3   //DEPS org.slf4j:slf4j-api:2.0.12
4   //DEPS org.slf4j:slf4j-simple:2.0.12
5   //DEPS com.pi4j:pi4j-core:2.6.0
6   //DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
7   //DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
8
9   import com.pi4j.Pi4J;
10  import com.pi4j.context.Context;
11  import com.pi4j.io.gpio.digital.DigitalOutput;
12
13  class HC595 {
…       ......
64  }
```

```
65
66  public class LEDMatrix {
67      private static final int DATA_PIN = 22;
68      private static final int LATCH_PIN = 27;
69      private static final int CLOCK_PIN = 17;
70
        private static final int[] pic  = {0x1c, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1c};
71      private static final int[] data = { // data of "0-F"
72          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
73          0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
74          0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
75          0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
76          0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
77          0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
78          0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
79          0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
80          0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
81          0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
82          0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
83          0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
84          0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
85          0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
86          0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
87          0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
88          0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
89          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
90      };
91
92      public static void main(String[] args) throws Exception {
93          var pi4j = Pi4J.newAutoContext();
94          HC595 ledMatrix = new HC595(pi4j, DATA_PIN, LATCH_PIN, CLOCK_PIN);
95
96          try {
97              while(true){
98                  for(int j=0; j<500; j++){  //Repeat enough times to display the smiling face a
    period of time
99                      int x = 0x80;
100                     for(int i=0; i<8; i++){
101                         ledMatrix.shiftOut(HC595.Order.MSBFIRST, pic[i] & 0xff);
102                         ledMatrix.shiftOut(HC595.Order.MSBFIRST, ~x);
103                         ledMatrix.updateLatch();
104                         x >>= 1;            //display the next column
105                         Thread.sleep(1);
106                     }
```

```
                }
                for(int k=0; k<data.length-8; k++){  //sizeof(data) total number of "0-F" columns
                    for(int j=0; j<10; j++){
                        int x=0x80;
                        for(int i=k; i<8+k; i++){
                            ledMatrix.shiftOut(HC595.Order.MSBFIRST, data[i] & 0xff);
                            ledMatrix.shiftOut(HC595.Order.MSBFIRST, ~x);
                            ledMatrix.updateLatch();
                            x >>= 1;
                            Thread.sleep(1);
                        }
                    }
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        } finally {
            ledMatrix.shutdown();
        }
    }
}
```

Define the contents displayed on the LED matrix.

```
    private static final int[] pic  = {0x1c, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1c};
    private static final int[] 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, // " "
    };
```

Use 2 nested for functions to make the LED dot matrix display a smiley face and loop it 500 times.

```
for(int j=0; j<500; j++){  //Repeat enough times to display the smiling face a period of time
    int x = 0x80;
    for(int i=0; i<8; i++){
        ledMatrix.shiftOut(HC595.Order.MSBFIRST, pic[i] & 0xff);
        ledMatrix.shiftOut(HC595.Order.MSBFIRST, ~x);
        ledMatrix.updateLatch();
        x >>= 1;           //display the next column
        Thread.sleep(1);
    }
}
```

Display the array from character 0 and loop 10 times to ensure that it is visible to human eyes then move the array elements one by one to make the characters scroll.

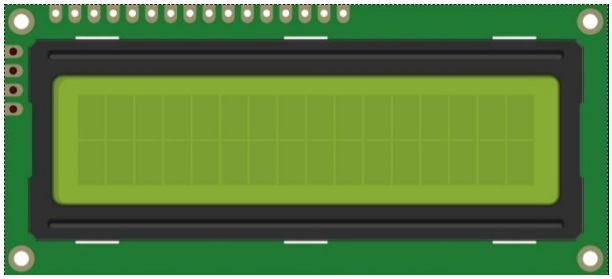
```
for(int k=0; k<data.length-8; k++){  //sizeof(data) total number of "0-F" columns
    for(int j=0; j<10; j++){
        int x=0x80;
        for(int i=k; i<8+k; i++){
            ledMatrix.shiftOut(HC595.Order.MSBFIRST, data[i] & 0xff);
            ledMatrix.shiftOut(HC595.Order.MSBFIRST, ~x);
            ledMatrix.updateLatch();
            x >>= 1;
            Thread.sleep(1);
        }
    }
}
```

Chapter 13 LCD1602

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

Project 13.1 I2C LCD1602

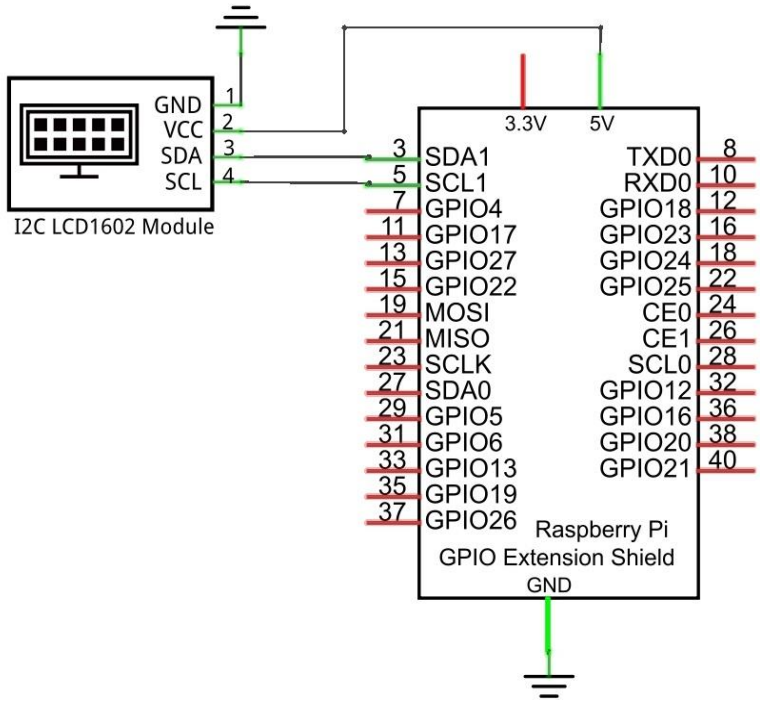
Component List

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.

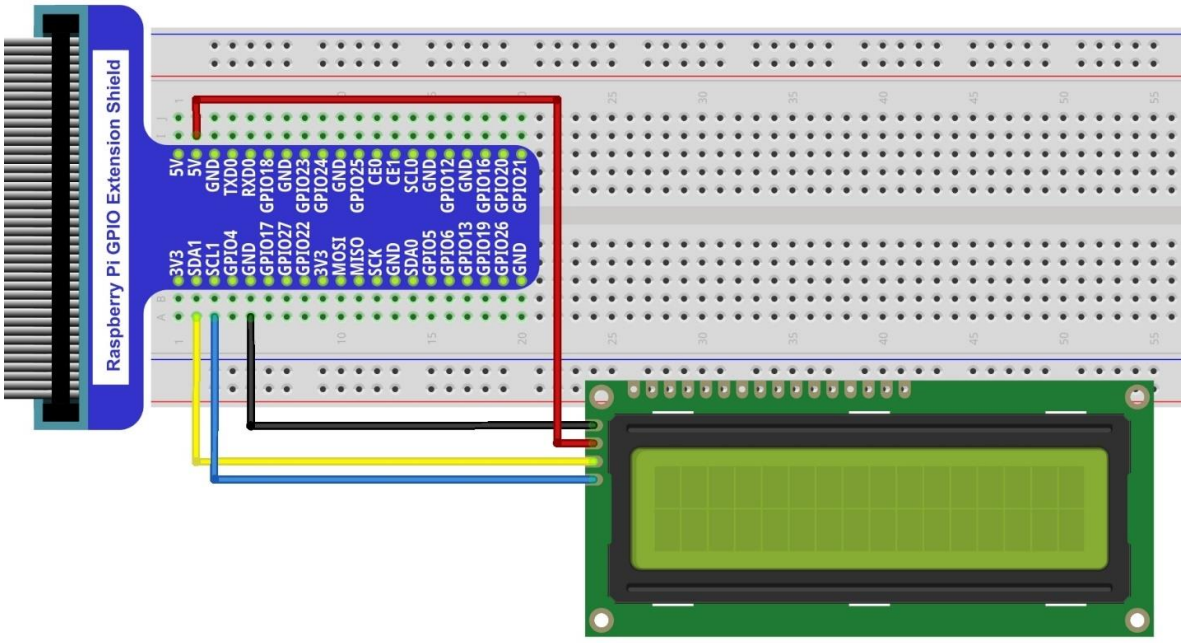
Schematic diagram



Hardware connection.

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)



If you have any concerns, please send an email to: support@freenove.com

Sketch

In this project, we will drive the LCD1602 display with I2C.

Sketch_13_I2CLCD1602

First, enter where the project is located:

```
cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_13_I2CLCD1602
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_18_I2CLCD1602
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_18_I2CLCD1602 $
```

Enter the command to run the code.

```
jbang I2CLCD1602.java
```

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Pi4j/Sketches/Sketch_18_I2CLCD1602
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_18_I2CLCD1602 $ jbang I2CLCD1602.java
```

When the code is running, you can see the first line of the display shows “Hello World”, and the second line displays the running time in second.

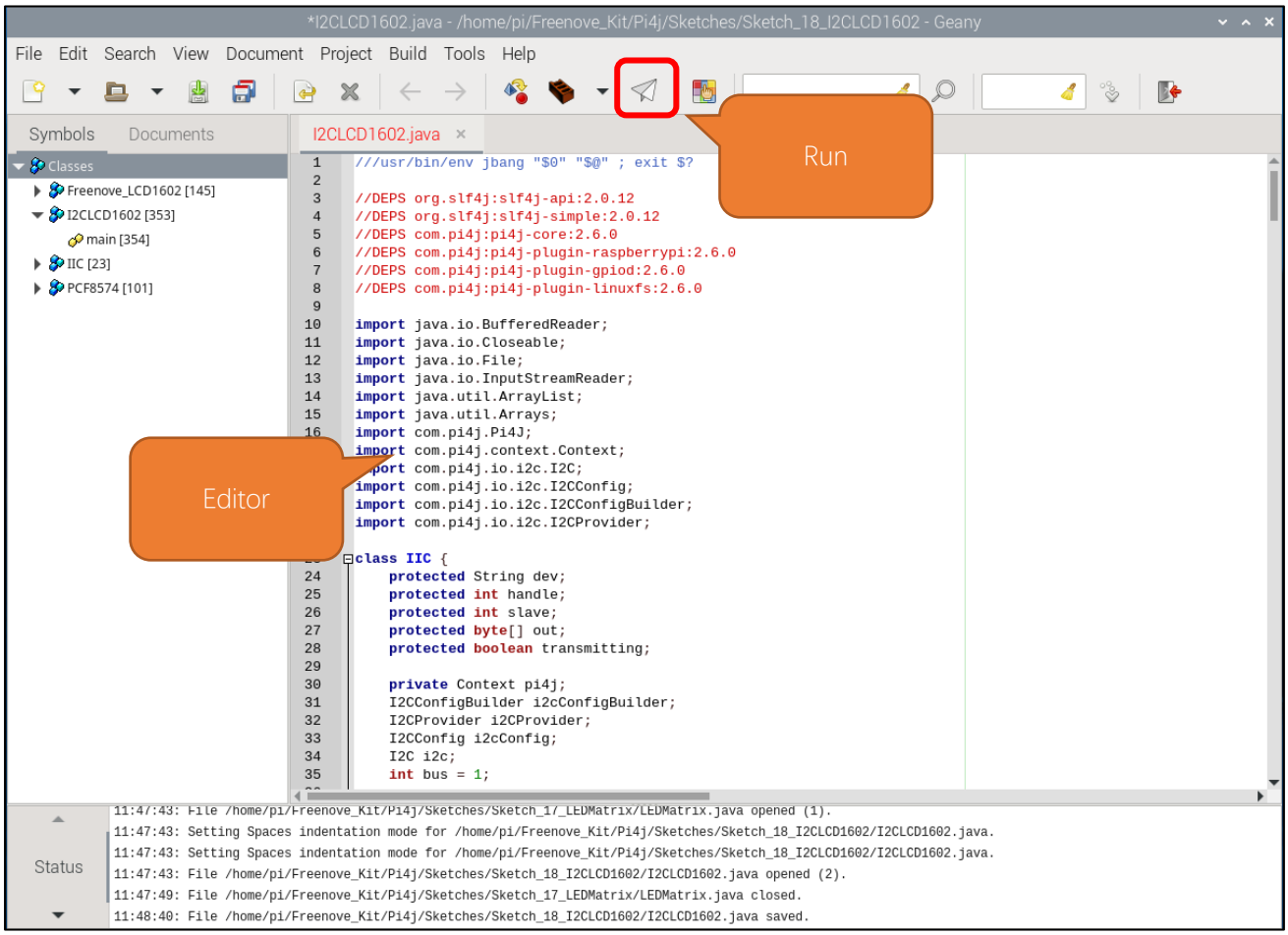
Press Ctrl+C to exit the program.

```
^C[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Shutting down Pi4J context/runtime...
[pi4j-shutdown] INFO com.pi4j.util.ExecutorPool - Shutting down executor pool Pi4J.RUNTIME
[pi4j-shutdown] INFO com.pi4j.runtime.impl.DefaultRuntime - Pi4J context/runtime successfully shutdow
n. Dispatching shutdown event.
pi@raspberrypi:~/Freenove_Kit/Pi4j/Sketches/Sketch_07_1_ADC $
```

You can run the following command to open the code with Geany to view and edit it.

```
geany I2CLCD1602.java
```

Click the icon to run the code.



If the code fails to run, please check Geany Configuration.

The following is program code:

```
// Shebang line for JBang to execute this script directly
///usr/bin/env jbang "$0" "$@" ; exit $?

// Dependencies for this script
//DEPS org.slf4j:slf4j-api:2.0.12
//DEPS org.slf4j:slf4j-simple:2.0.12
//DEPS com.pi4j:pi4j-core:2.6.0
//DEPS com.pi4j:pi4j-plugin-raspberrypi:2.6.0
//DEPS com.pi4j:pi4j-plugin-gpiod:2.6.0
//DEPS com.pi4j:pi4j-plugin-linuxfs:2.6.0

import java.io.BufferedReader;
import java.io.Closeable;
import java.io.File;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;

import com.pi4j.Pi4J;
import com.pi4j.context.Context;
import com.pi4j.io.i2c.I2C;
import com.pi4j.io.i2c.I2CConfig;
import com.pi4j.io.i2c.I2CConfigBuilder;
import com.pi4j.io.i2c.I2CProvider;

class IIC {
    protected String dev;
    protected int handle;
    protected int slave;
    protected byte[] out;
    protected boolean transmitting;
    private Context pi4j;
    I2CConfigBuilder i2cConfigBuilder;
    I2CProvider i2CProvider;
    I2CConfig i2cConfig;
    I2C i2c;
    int bus = 1;

    public IIC(int bus) {
        this.bus = bus;
        constructor();
    }
    public IIC(String s) {
```

```
44          String b = s.split("i2c-")[1];
45          try {
46              this.bus = Integer.parseInt(b);
47          } catch (Exception e) {
48              this.bus = 1;
49          }
50          constructor();
51      }
52
53      private void constructor() {
54          pi4j = Pi4J.newAutoContext();
55          i2CProvider = pi4j.provider("linuxfs-i2c");
56          i2cConfigBuilder = I2C.newConfigBuilder(pi4j).bus(bus);
57      }
58      public void beginTransmission(int address) {
59          if (i2cConfig == null) {
60              i2cConfig = i2cConfigBuilder.device(address).build();
61          }
62          if (i2c == null) {
63              i2c = i2CProvider.create(i2cConfig);
64          }
65      }
66      public void write(int b) {
67          i2c.write(b);
68      }
69      public byte read() {
70          return i2c.readByte();
71      }
72      public byte[] read(int size) {
73          return i2c.readByteBuffer(size).array();
74      }
75      public void endTransmission() {
76          i2c.close();
77      }
78      public static String[] list() {
79          ArrayList<String> devs = new ArrayList<String>();
80          File dir = new File("/dev");
81          File[] files = dir.listFiles();
82          if (files != null) {
83              for (File file : files) {
84                  if (file.getName().startsWith("i2c-")) {
85                      devs.add(file.getName());
86                  }
87              }
```

```
88          }
89          String[] tmp = devs.toArray(new String[devs.size()]);
90          Arrays.sort(tmp);
91          return tmp;
92      }
93  }
94
95  class PCF8574 {
...     ......
146 }
147
148 class Freenove_LCD1602 {
...     ......
372 }
373
374 public class I2CLCD1602{
375     public static void main(String[] args) throws Exception {
376         PCF8574 pcf = new PCF8574(0x27);// or 0x3F
377         Freenove_LCD1602 lcd;
378         lcd = new Freenove_LCD1602(pcf);
379         lcd.position(0, 0);             //show time on the lcd display
380         lcd.puts("Hello World.");
381         try{
382             int count = 0;
383             while(true){
384                 String buf = "Count: " + count;
385                 lcd.position(0, 1);   //show time on the lcd display
386                 lcd.puts(buf);
387                 Thread.sleep(1000);
388                 count++;
389             }
390         }
391         catch (InterruptedException e) {
392             Thread.currentThread().interrupt();
393         }
394     }
395 }
```

Constructor, assigns the I2C bus to bus, and calls the constructor function to initialize the I2C bus.

```
    public IIC(int bus) {
        this.bus = bus;
        constructor();
    }
    public IIC(String s) {
        String b = s.split("i2c-")[1];
        try {
            this.bus = Integer.parseInt(b);
        } catch (Exception e) {
            this.bus = 1;
        }
        constructor();
    }
```

I2C constructor, initialize the I2C bus.

```
    private void constructor() {
        pi4j = Pi4J.newAutoContext();
        i2CProvider = pi4j.provider("linuxfs-i2c");
        i2cConfigBuilder = I2C.newConfigBuilder(pi4j).bus(bus);
    }
```

I2C bus acquisition function. Calling the list() function can obtain the names of all I2C buses currently existing on the Raspberry Pi.

```
    public static String[] list() {
        ArrayList<String> devs = new ArrayList<String>();
        File dir = new File("/dev");
        File[] files = dir.listFiles();
        if (files != null) {
            for (File file : files) {
                if (file.getName().startsWith("i2c-")) {
                    devs.add(file.getName());
                }
            }
        }
        String[] tmp = devs.toArray(new String[devs.size()]);
        Arrays.sort(tmp);
        return tmp;
    }
```

Use the pi4j library to repackage the I2C functions. These functions refer to the classic usage of Arduino. This is to be compatible with the subsequent PCF8574 class and Freenove_LCD1602 class, making it easier to drive LCD1602.

```
    public void beginTransmission(int address) {
        if (i2cConfig == null) {
            i2cConfig = i2cConfigBuilder.device(address).build();
        }
        if (i2c == null) {
            i2c = i2CProvider.create(i2cConfig);
        }
    }
    public void write(int b) {
        i2c.write(b);
    }
    public byte read() {
        return i2c.readByte();
    }
    public byte[] read(int size) {
        return i2c.readByteBuffer(size).array();
    }
    public void endTransmission() {
        i2c.close();
    }
```

The following are the two classes we wrote for LCD1602 in reference to Arduino. Here, we do not go into too much detail here. If you are interested in this code, you can use Geany to view the code.

```
class PCF8574 {
    ......
}

class Freenove_LCD1602 {
    ......
}
```

Initialize PCF8574 and Freenove_LCD1602 classes, and assign the value to LCD.

```
        PCF8574 pcf = new PCF8574(0x27); // or 0x3F
        Freenove_LCD1602 lcd;
        lcd = new Freenove_LCD1602(pcf);
```

At the first lin of LCD1602, print the character “Hello World”.

```
        lcd.position(0, 0);           //show time on the lcd display
        lcd.puts("Hello World.");
```

Print the character “Count” and the counts number at the second line every one second.

```
int count = 0;
while(true){
    String buf = "Count: " + count;
    lcd.position(0, 1);   //show time on the lcd display
    lcd.puts(buf);
    Thread.sleep(1000);
    count++;
}
```

In Java, an `InterruptedException` is thrown when a thread's waiting, sleeping (e.g., using ‘Thread.sleep()’), or other blocking operations are interrupted. When a thread is interrupted, its interrupted status is set to true, and any of these blocking operations will result in an ‘InterruptedException’ being thrown.

When ‘InterruptedException’ is capured, ‘Thread.currentThread().interrupt()’ is called keep the interrupt status, so that the thread can respond to the interruption request appropriately.

```
catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions, or if 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 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.