Getting Started

Thank you for choosing Freenove products!

Get Support and Offer Input

Freenove provides free, 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:

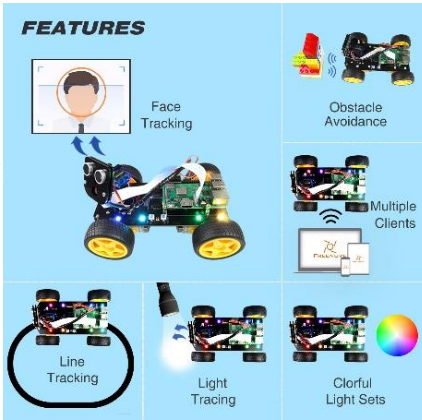
- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. 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.

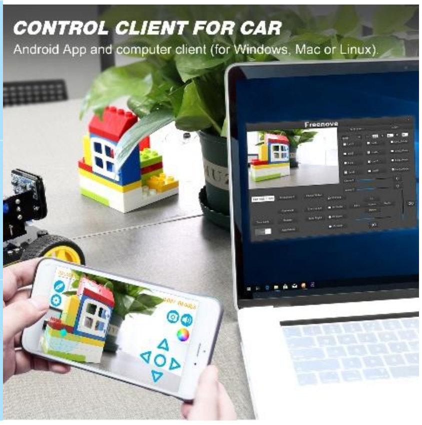
Car and Robot for Raspberry Pi

We also have cars and robot kit for Raspberry Pi. If you are interested in them, please visit our website for details.

http://www.freenove.com/store.html

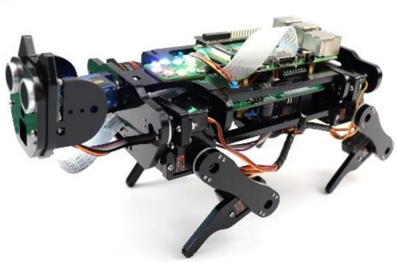
FNK0043 Freenove 4WD Smart Car Kit for Raspberry Pi

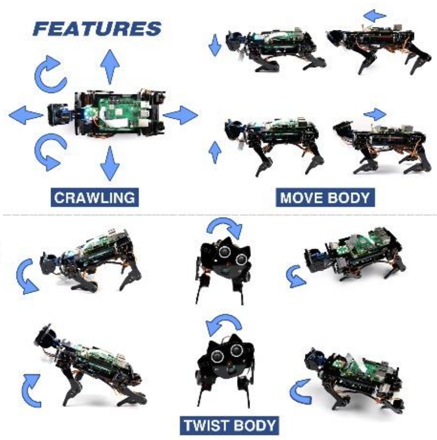


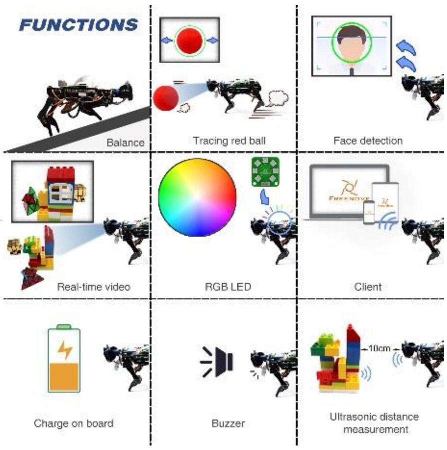


https://www.youtube.com/watch?v=4Zv0GZUQjZc

FNK0050 Freenove Robot Dog Kit for Raspberry Pi







https://www.youtube.com/watch?v=7BmIZ8_R9d4&t=35s

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** (**C** code and **Python** code). We provide both C and Python code for each project in this tutorial. After completing this tutorial, you can learn Java by reading Processing.pdf.

This kit contains all the accessory electronic components and modules needed to complete the projects described in the index. You can also use these components and modules to create projects of your own design.

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

support@freenove.com

Raspberry Pi

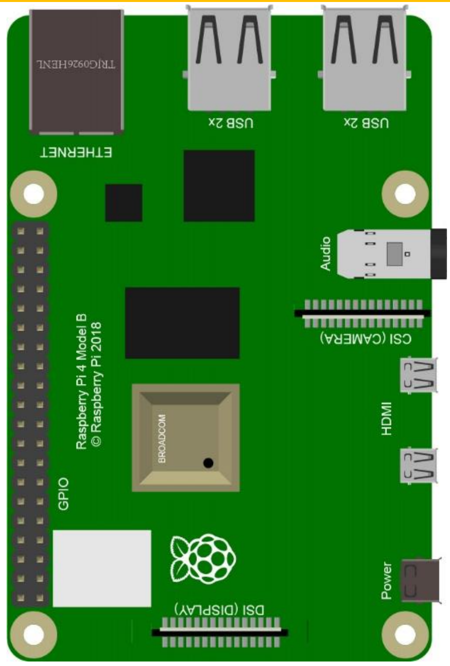
So far, at this writing, Raspberry Pi has advanced to its fourth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

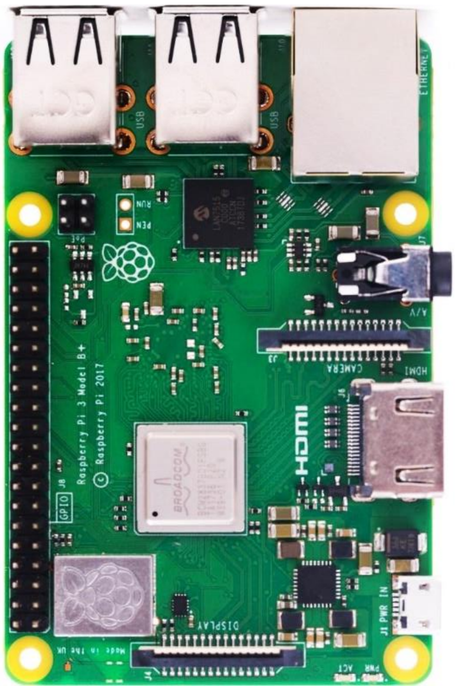
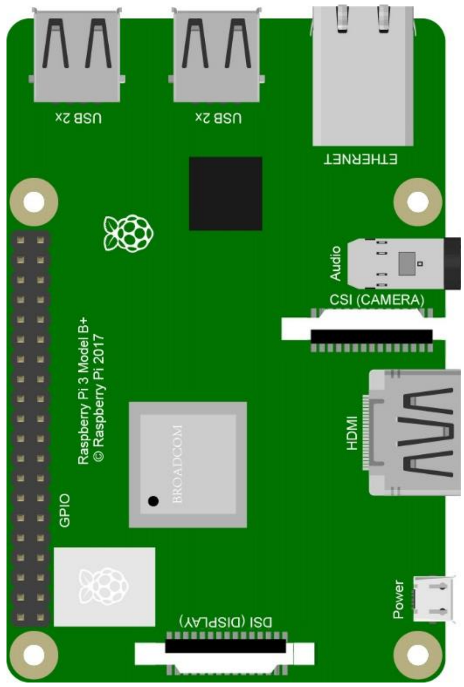
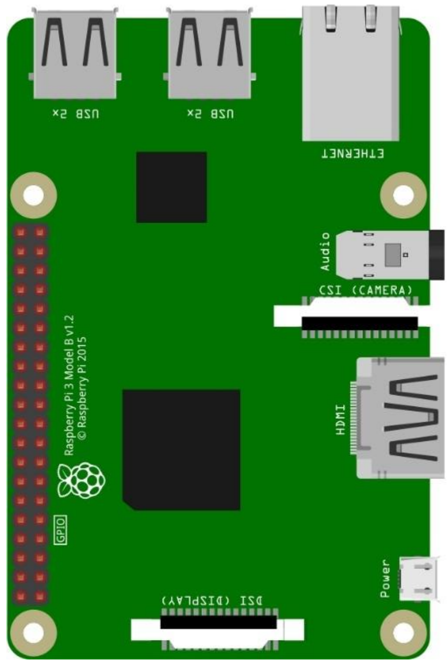
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility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

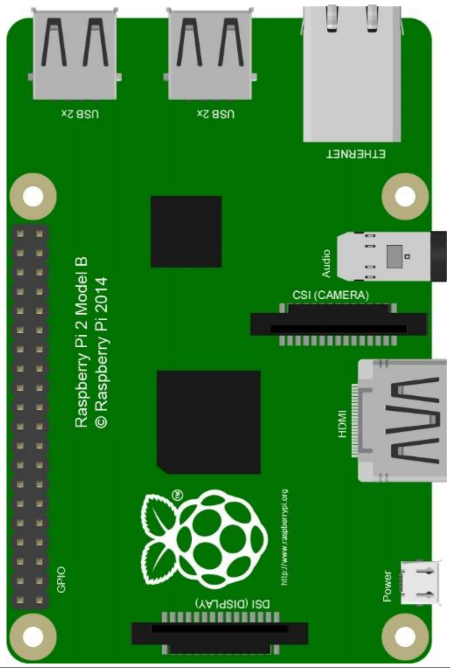
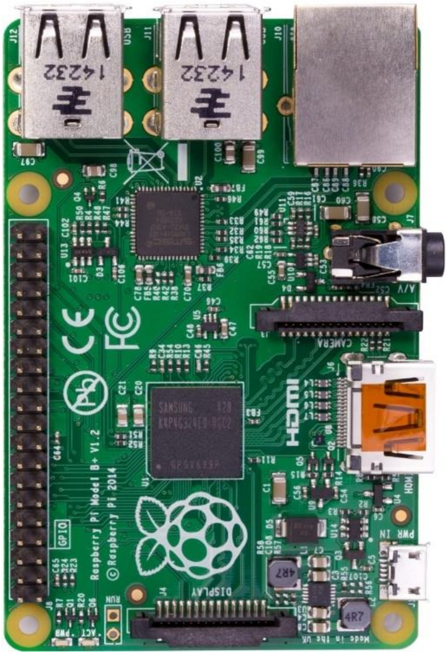
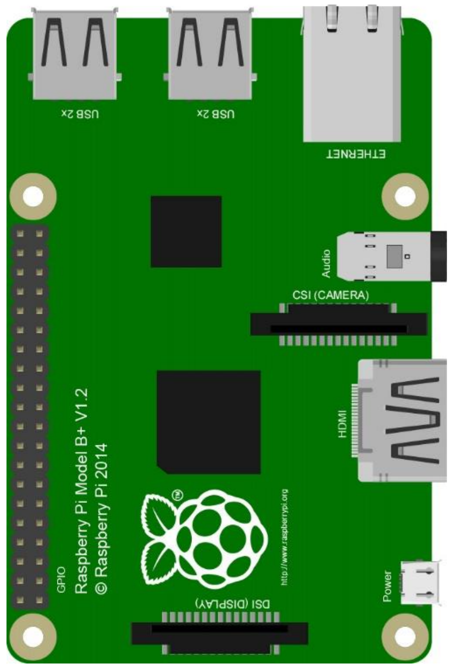
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

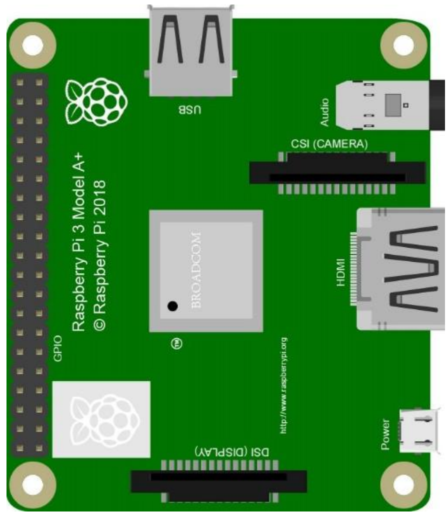
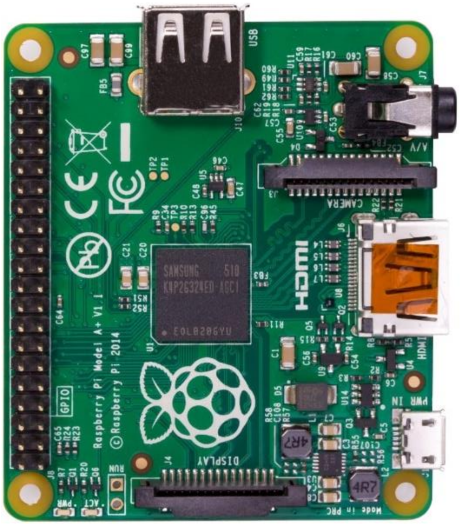
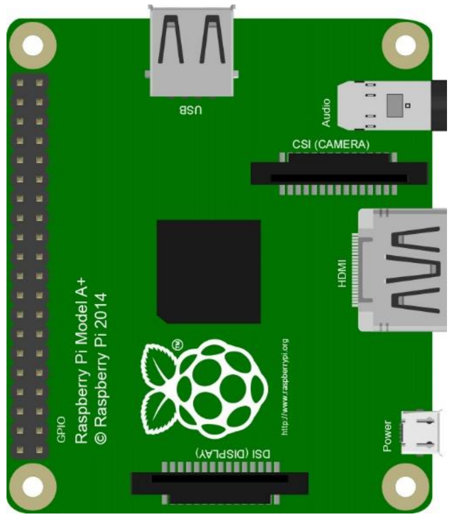
Actual image of Raspberry Pi 4 Model B:

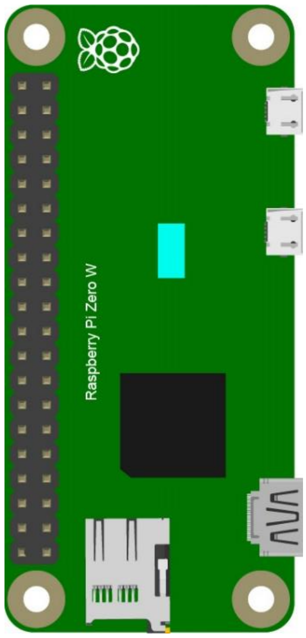
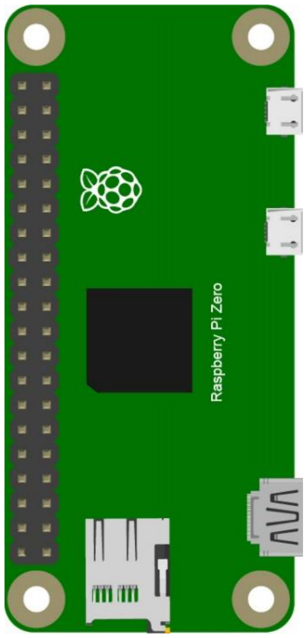
CAD image of Raspberry Pi 4 Model B:



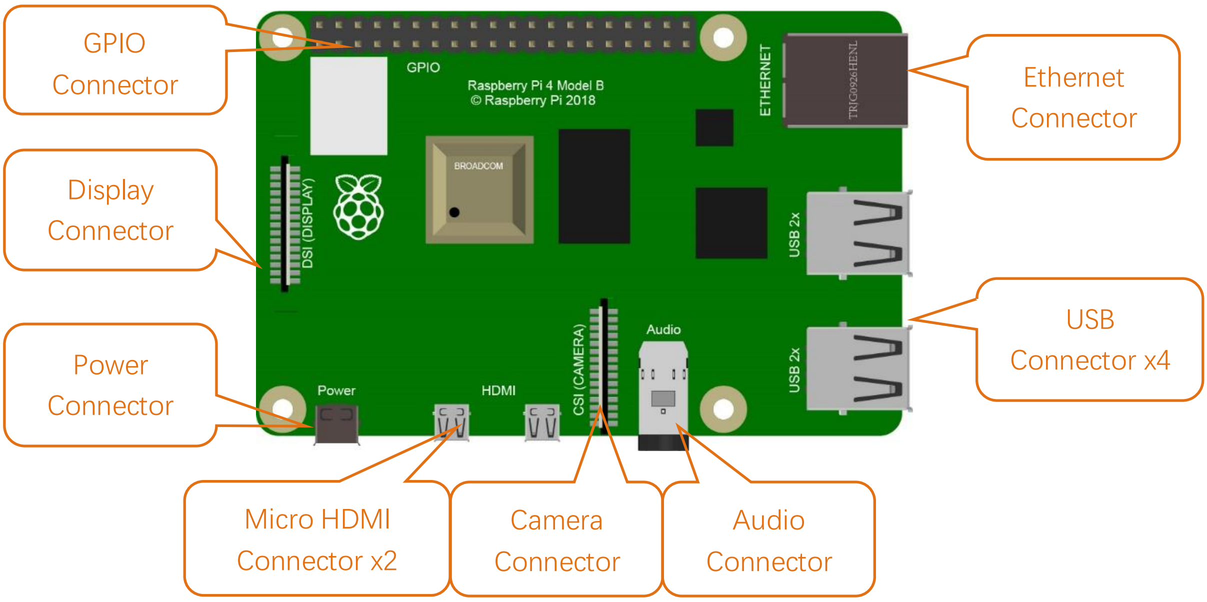
Actual image of Raspberry Pi 3 Model B+: 	CAD image of Raspberry Pi 3 Model B+: 
Actual image of Raspberry Pi 3 Model B: 	CAD image of Raspberry Pi 3 Model B: 

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+: 

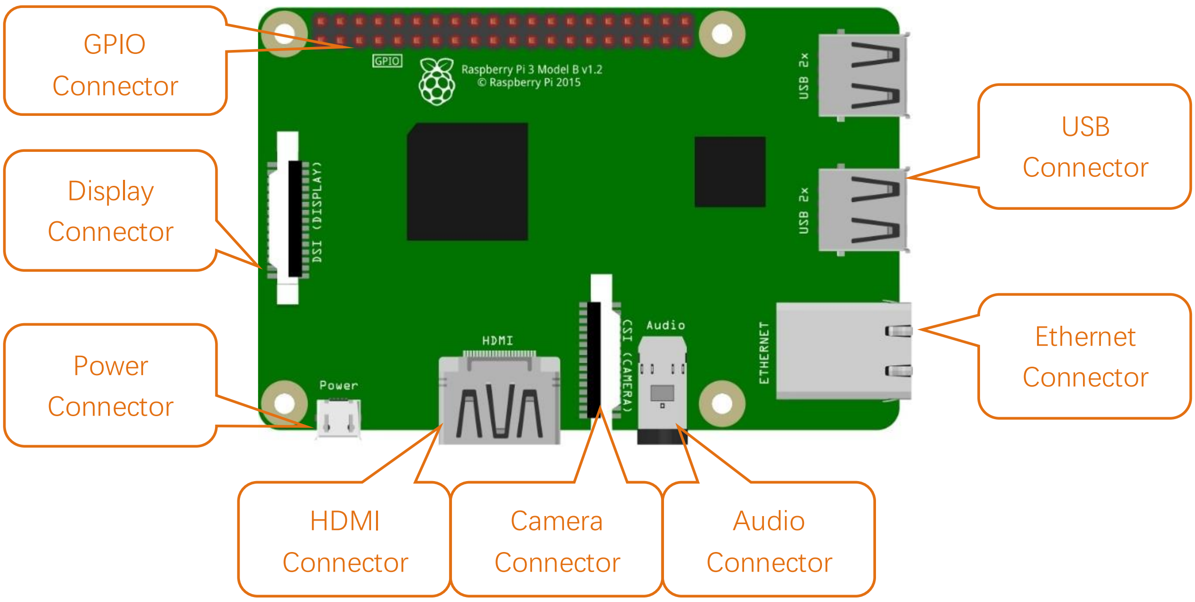
Actual image of Raspberry Pi 3 Model A+:  	CAD image of Raspberry Pi 3 Model A+: 
Actual image of Raspberry Pi 1 Model A+:  	CAD image of Raspberry Pi 1 Model A+: 

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：
	

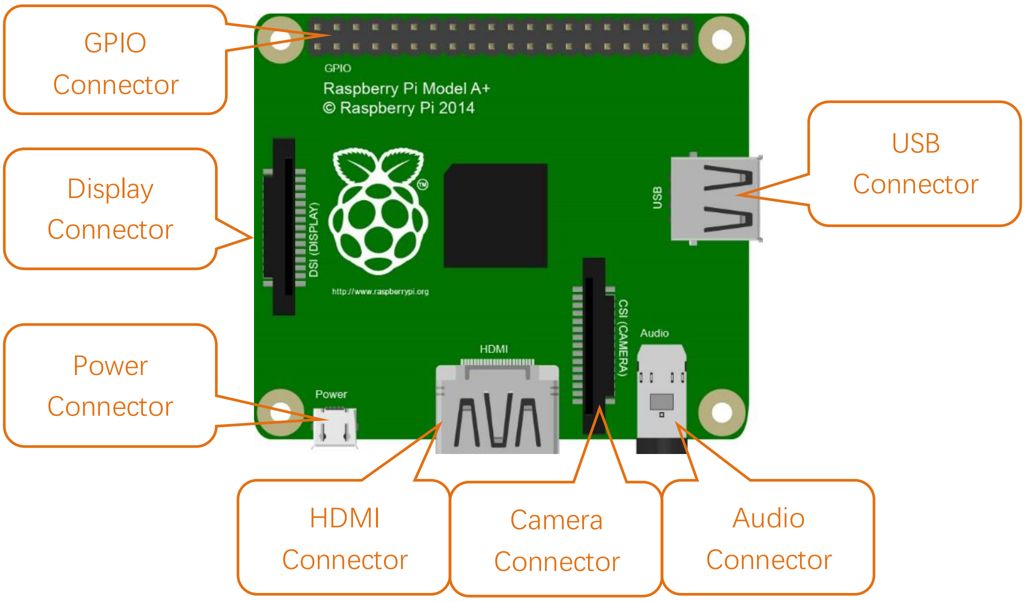
Hardware interface diagram of RPi 4B:



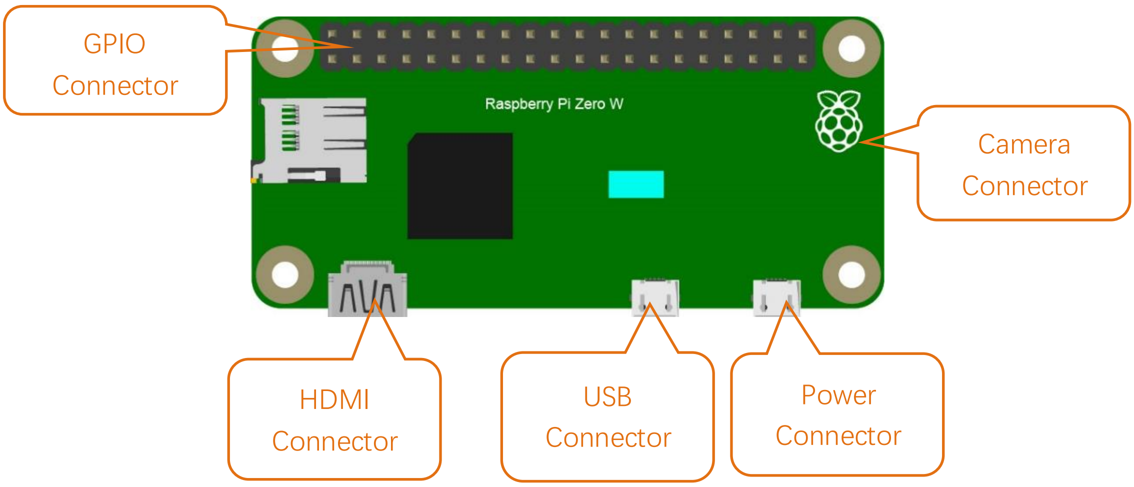
Hardware interface diagram of RPi 3B+/3B/2B/1B+:



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



Hardware interface diagram of RPi Zero/Zero W:



Installing an Operating System

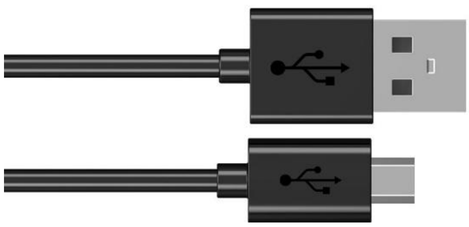
The first step is to install an operating system on your RPi so that it can be programmed and function. If you have installed a system in your RPi, you can start from Chapter 0 Preparation.

Component List

Required Components

<table>
<tr>
<td>Any Raspberry Pi

</td>
<td>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.)

</td>
</tr>
<tr>
<td>Micro or Type-C USB Cable x1
</td>
<td>Micro SD Card (TF Card) x1, Card Reader x1

</td>
</tr>
</table>

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

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

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

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

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

Optional Components

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

Required Accessories for Monitor

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

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

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

<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></tr>
<tr><td>Monitor</td><td colspan="7">Yes (All)</td></tr>
<tr><td>Mouse</td><td colspan="7">Yes (All)</td></tr>
<tr><td>Keyboard</td><td colspan="7">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></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td colspan="6">No</td><td>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="4">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></td></tr>
<tr><td>USB to Ethernet Interface</td><td colspan="2" rowspan="2">select one from two or select two from two</td><td colspan="2">optional</td><td>Internal Integration</td><td colspan="2" rowspan="2">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.

<table>
<tr><th></th><th>Pi Zero</th><th>Pi Zero W</th><th>Pi A+</th><th>Pi 3A+</th><th>Pi B+/2B</th><th>Pi 3B/3B+/4B</th></tr>
<tr><td>Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)</td><td>Yes</td><td>Yes</td><td>No</td><td rowspan="2" colspan="3">NO</td></tr>
<tr><td>USB to Ethernet interface</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
</table>

Raspberry Pi OS

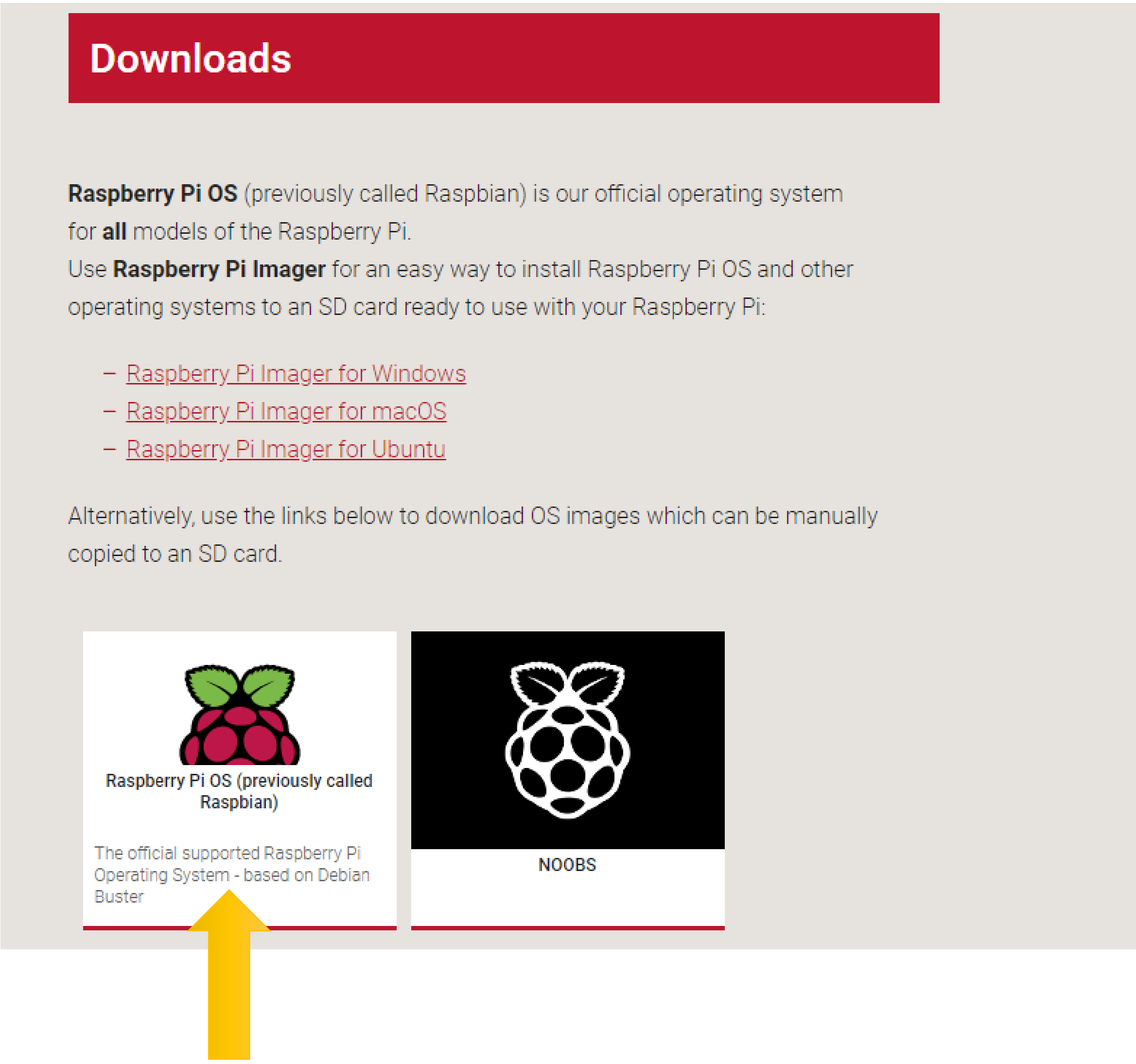
Automatically

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.

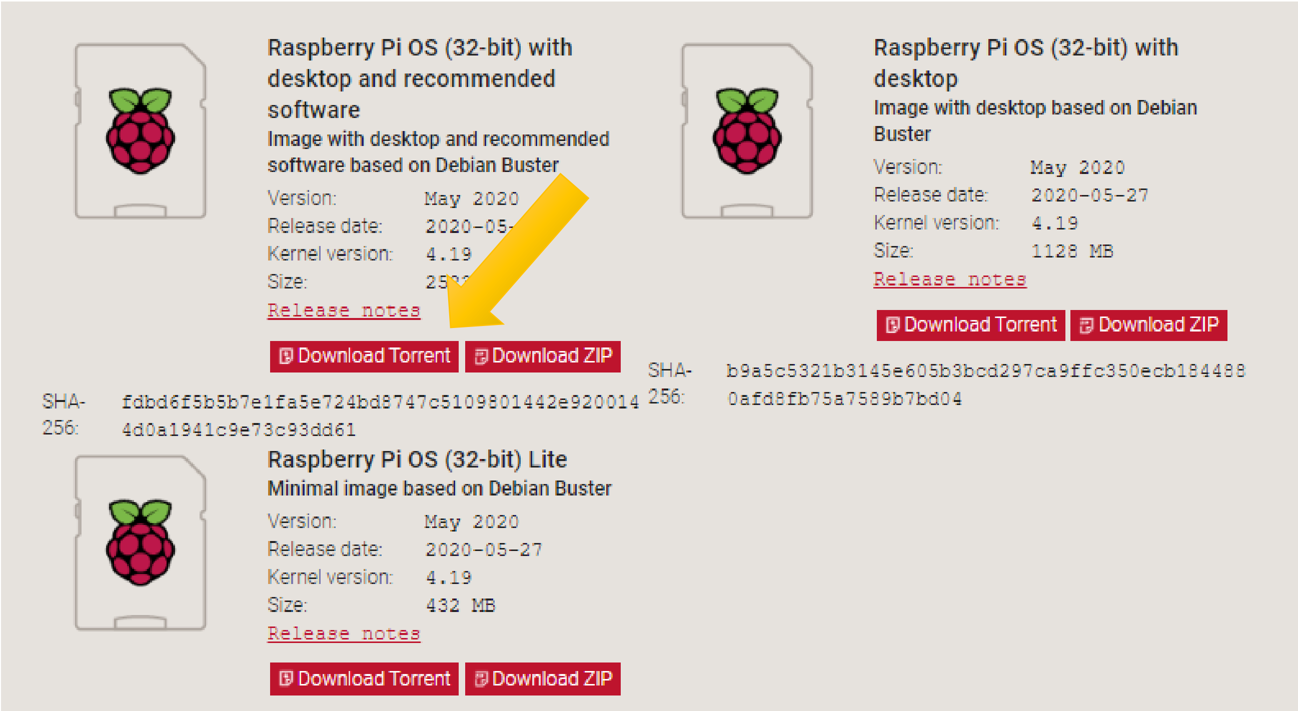
Manually

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

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



.

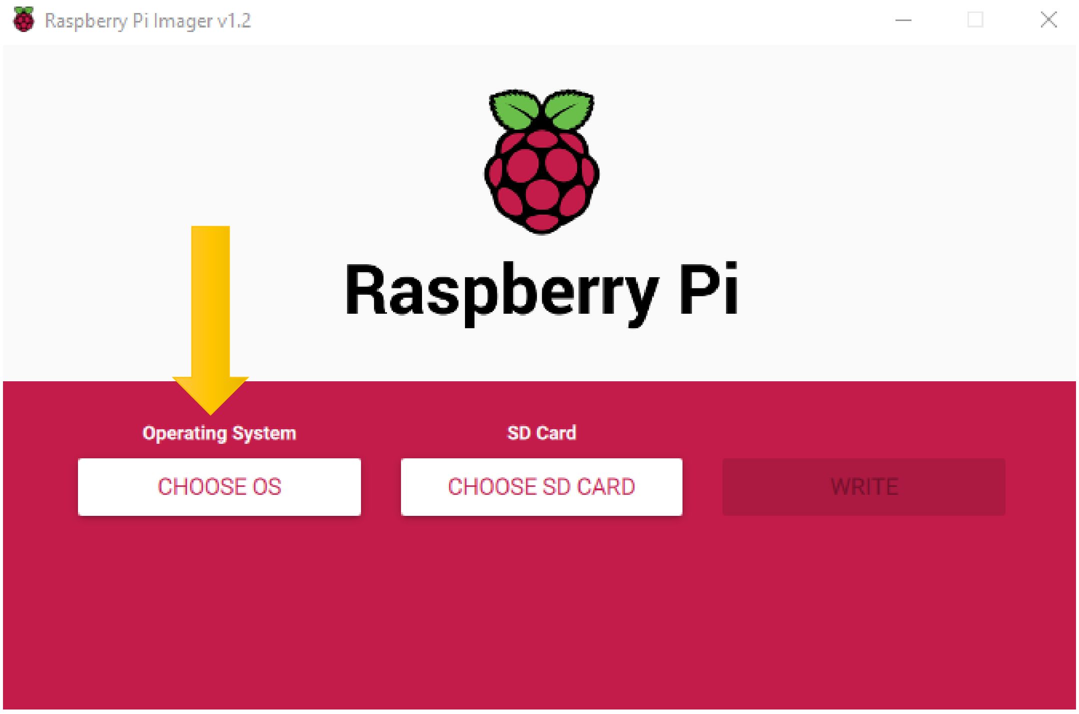


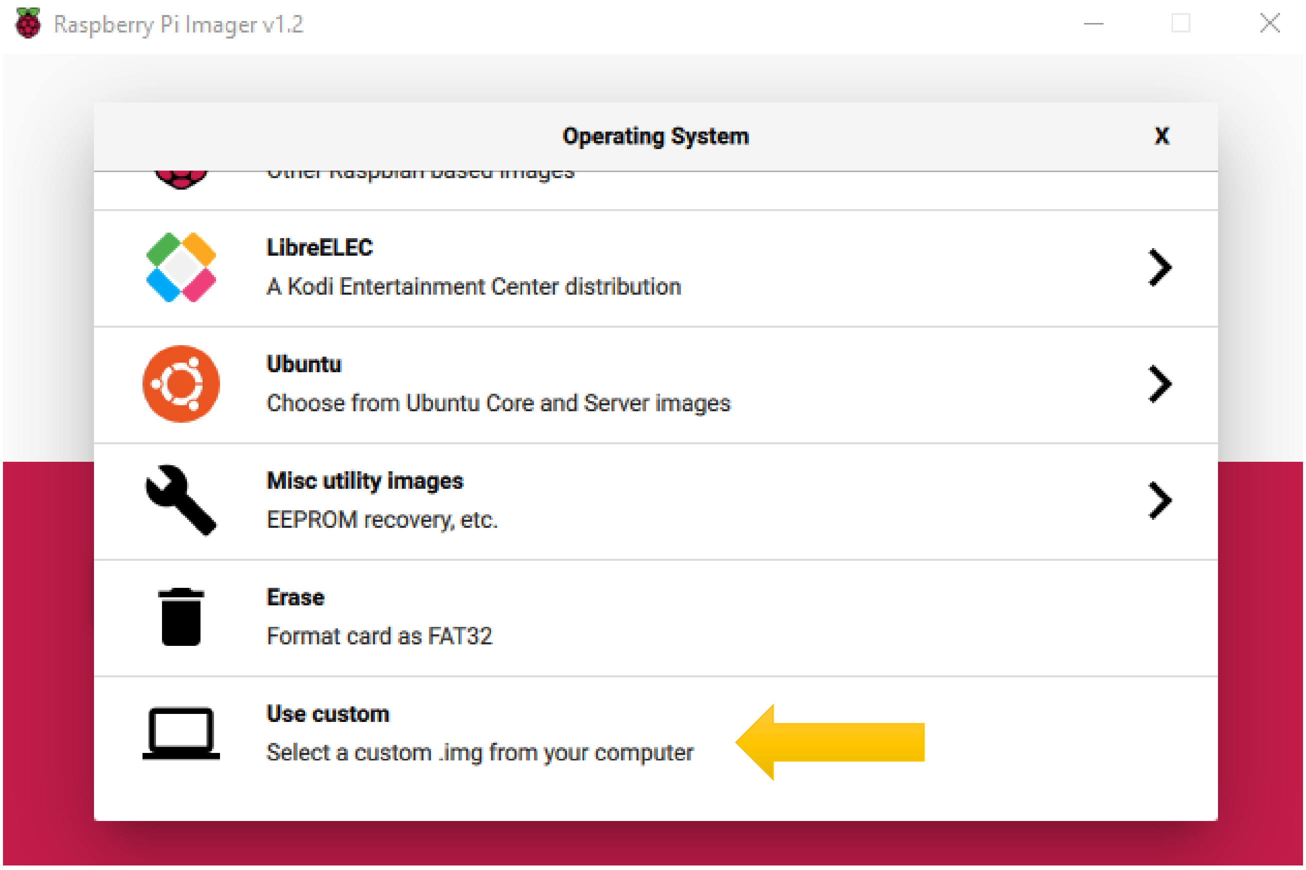
And then the zip file is downloaded.

Write System to Micro SD Card

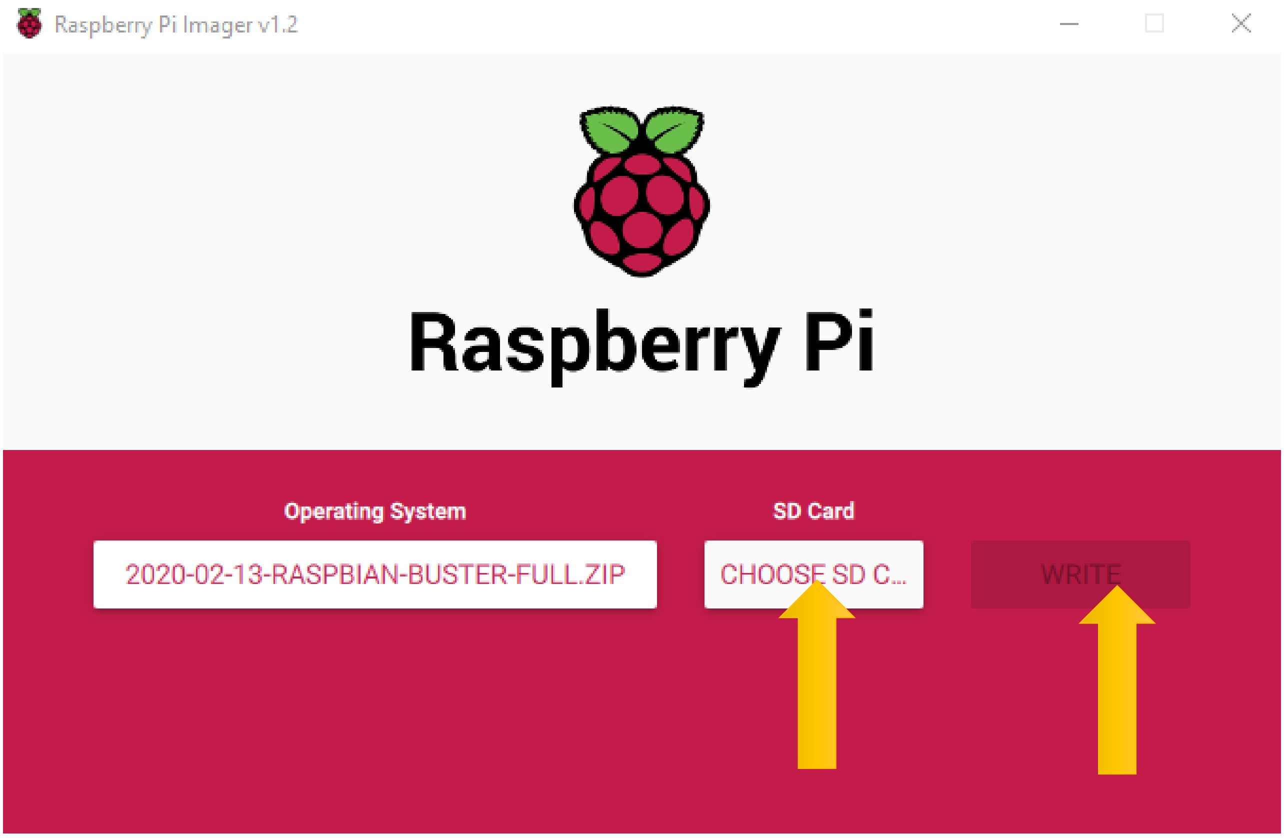
First, put your Micro **SD card** into card reader and connect it to USB port of PC.

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


Raspberry Pi Imager v1.2
Raspberry Pi
Operating System
CHOOSE OS
SD Card
CHOOSE SD CARD
WRITE

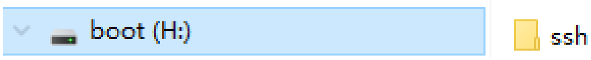

Raspberry Pi Imager v1.2
Operating System
X
LibreELEC
A Kodi Entertainment Center distribution
Ubuntu
Choose from Ubuntu Core and Server images
Misc utility images
EEPROM recovery, etc.
Erase
Format card as FAT32
Use custom
Select a custom .img from your computer

Choose the SD card. Then click "WRITE".

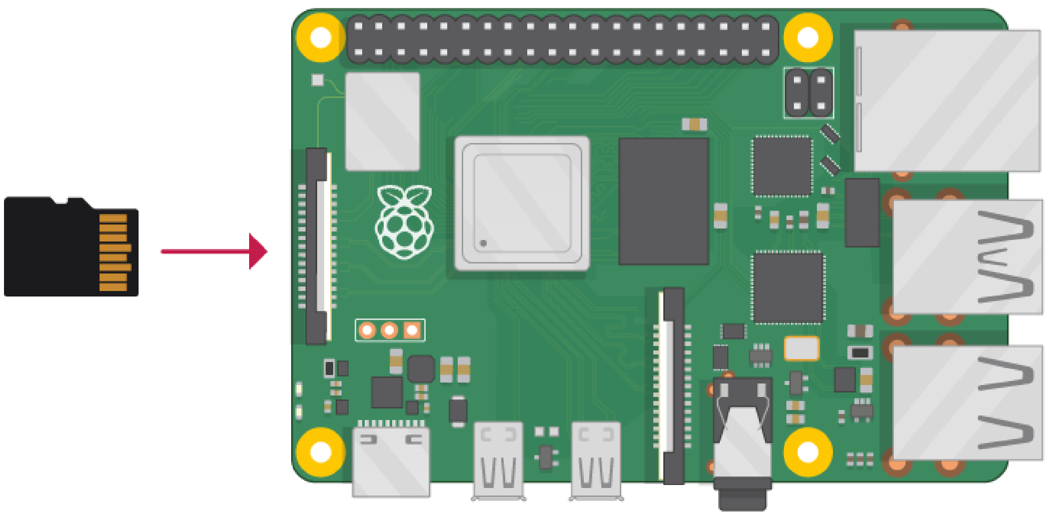


Enable ssh

If you don't have a separate monitor, after the system is written successfully, **create a folder named "ssh" under generated boot disk of Micro SD Card.**



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



Getting Started with Raspberry Pi

Monitor desktop

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

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

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

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

Remote desktop & VNC

If you have logged in Raspberry Pi via display, you can skip to VNC Viewer.

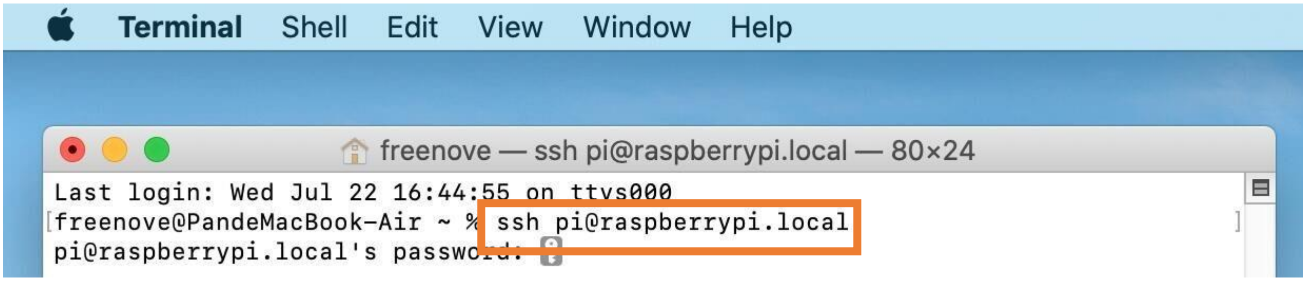
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.
Connect your pi and computer to the router via a network cable.

MAC OS Remote Desktop

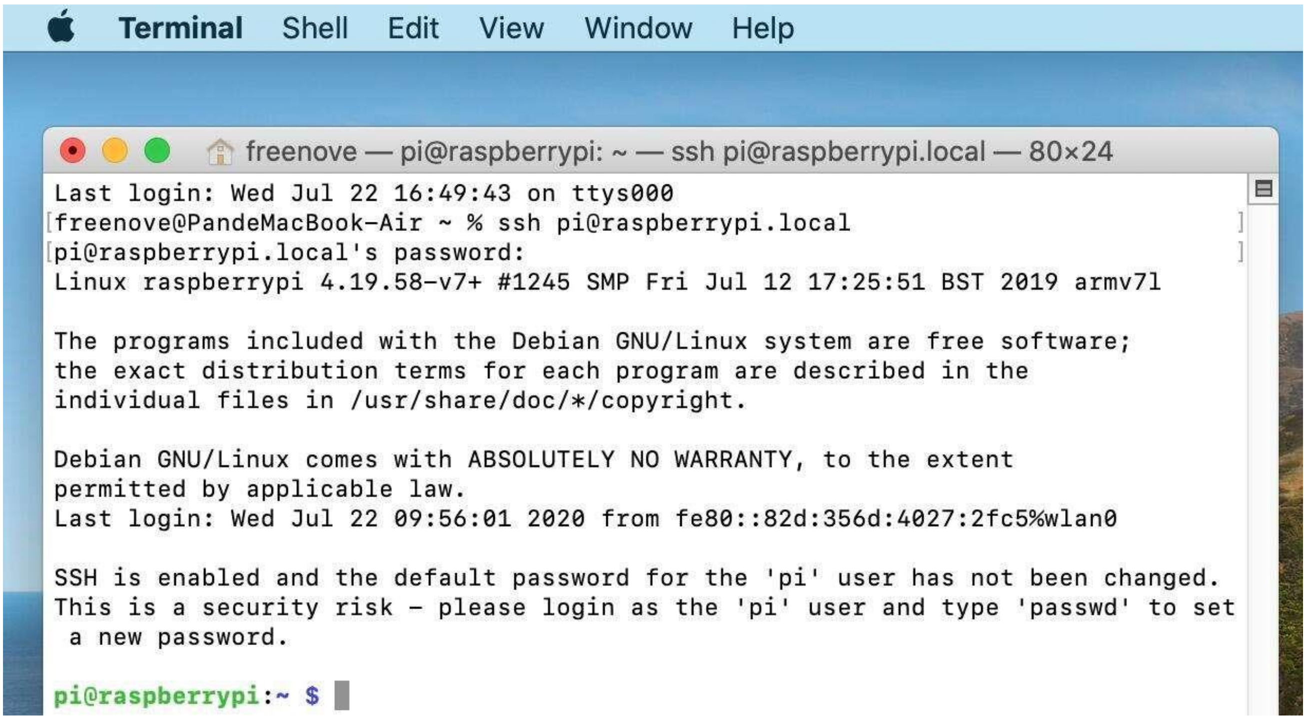
Open the terminal and type following command. If this command doesn't work, please move to next page.

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

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



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

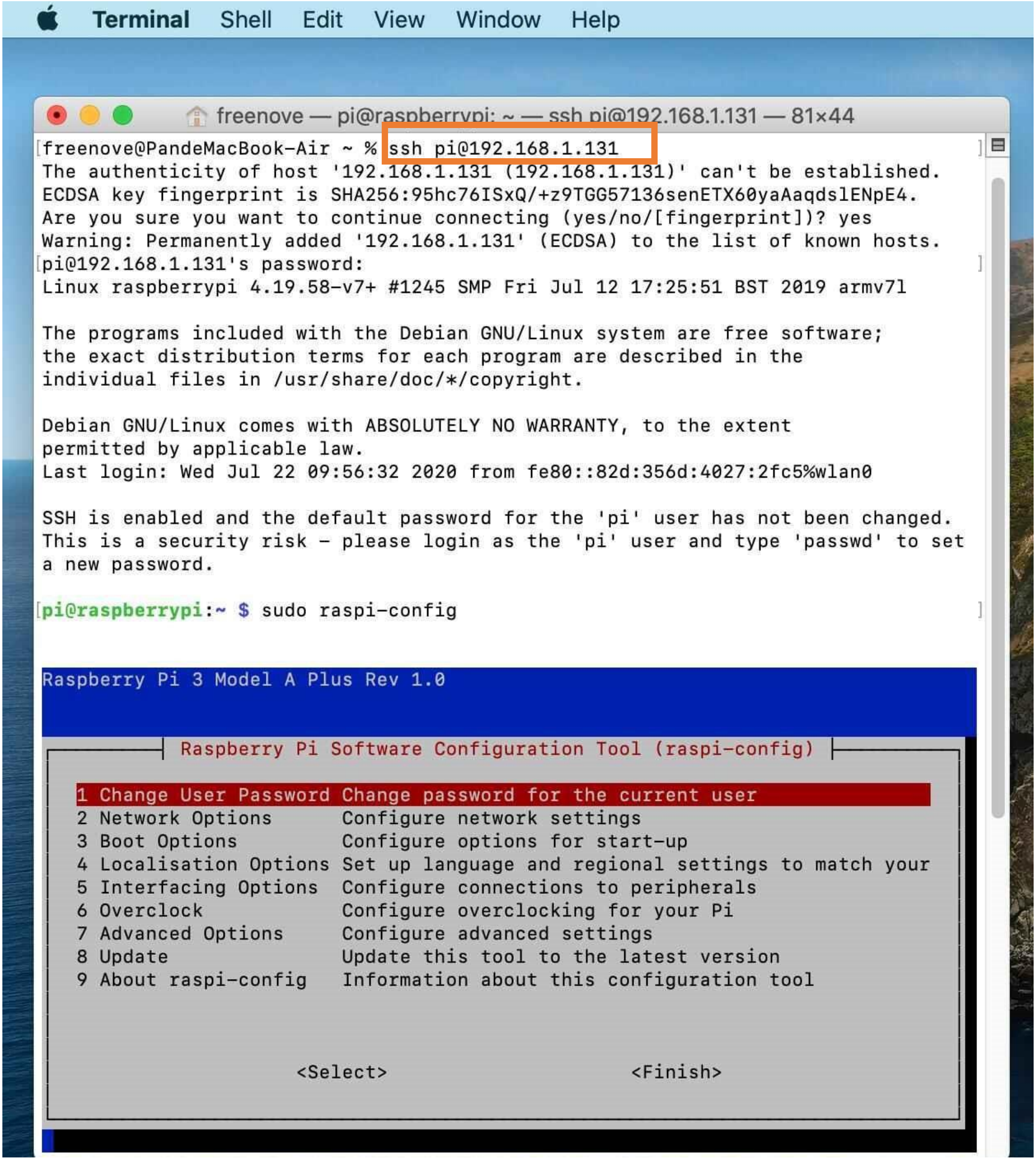


When you see pi@raspberrypi:~ $, you have logged in Pi successfully. Then you can skip to next section.

You can also use the IP address to log in Pi.
Enter router client to inquiry IP address named “raspberry pi”. For example, I have inquired to my RPi IP address, and it is “192.168.1.131".

Open the terminal and type following command.

```
ssh pi@192.168.1.131
```



Then you can skip to VNC Viewer.

Windows OS Remote Desktop

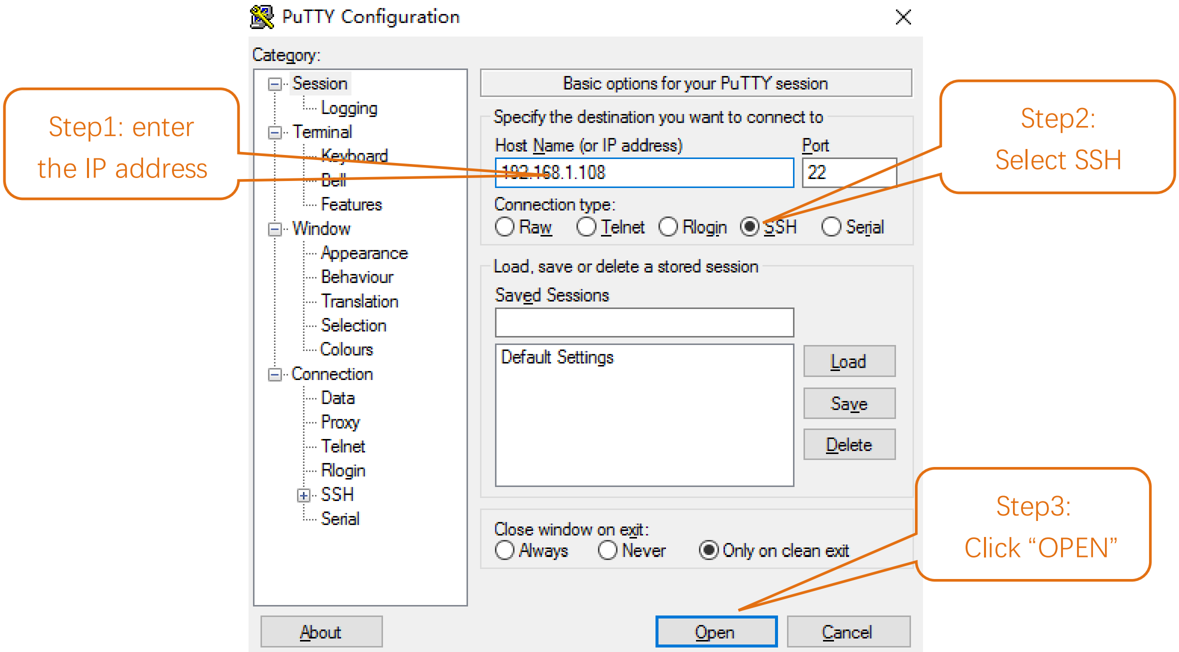
The windows built-in application remote desktop corresponds to the Raspberry Pi xrdp service.

Download the tool software Putty. Its official address: http://www.putty.org/

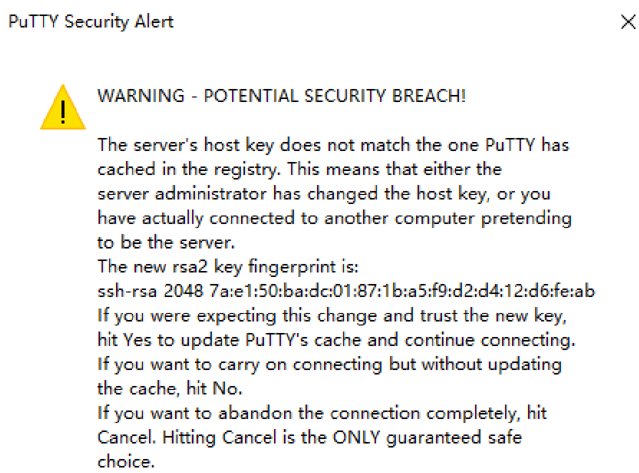
Or download it here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Then use net cable to connect your RPi to the same router with your PC. Then put the system Micro SD Card prepared before into the slot of the RPi and turn on the power supply. Enter router client to inquiry IP address named "raspberry pi". For example, my RPi IP address is "192.168.1.108".

Then open Putty, enter the address, select SSH, and then click "OPEN", as shown below:

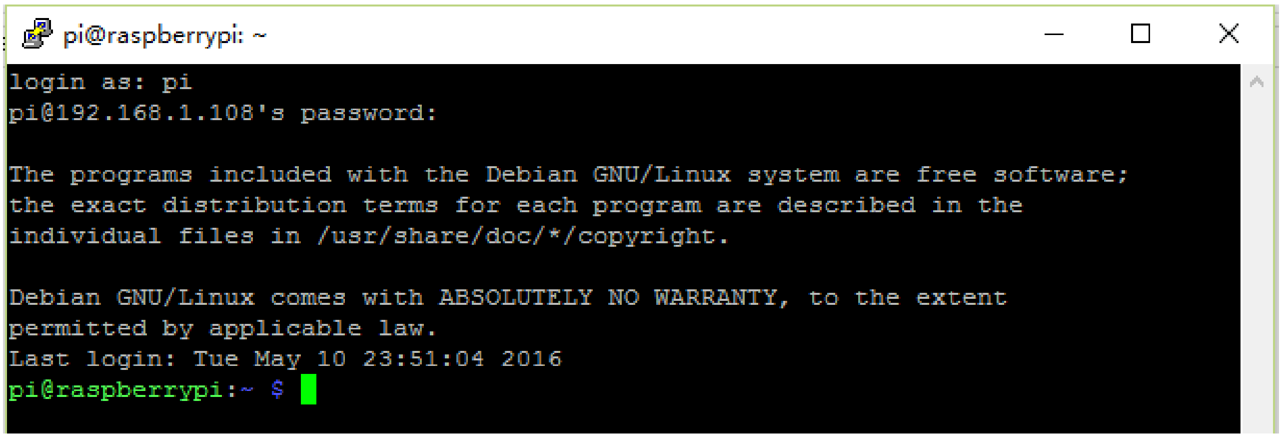


There will appear a security warning at first login. Just click "YES".



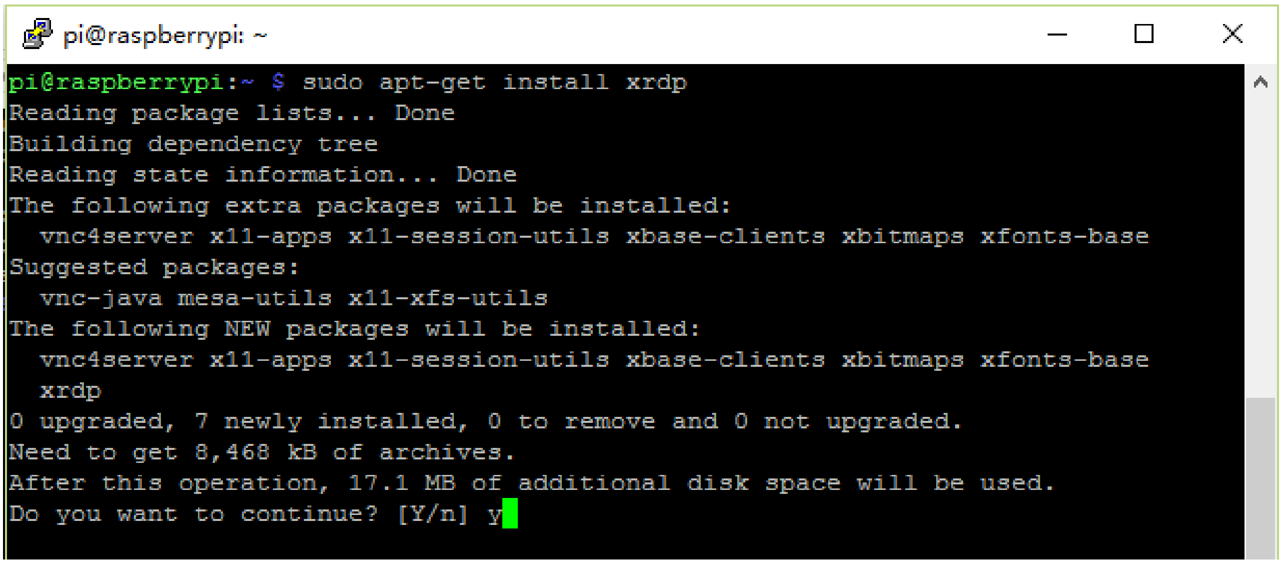
Then there will be a login interface. Login as: **pi**; password: **raspberry**. When you enter the password, there will be **no display** on the screen. This is normal. After the correct input, press "Enter" to confirm.

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



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

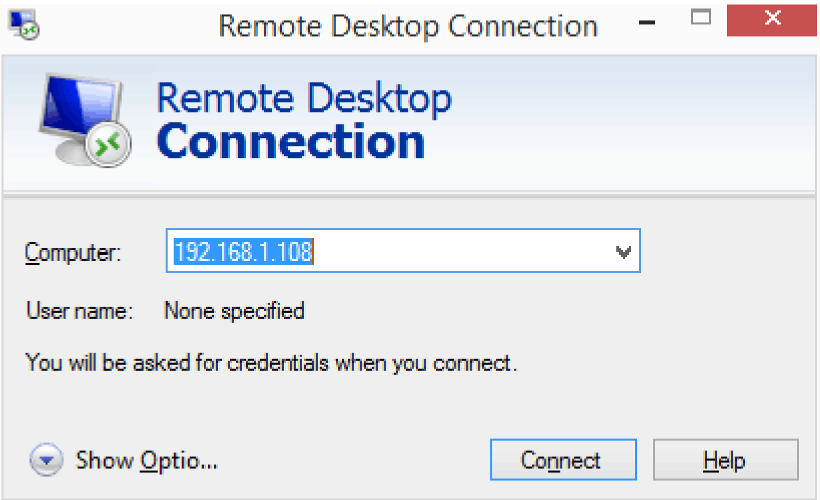
```
sudo apt-get install xrdp
```



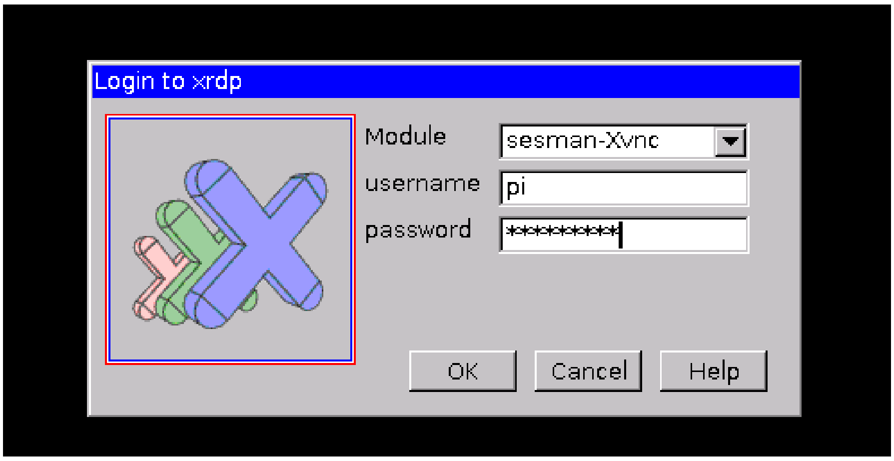
Enter "Y", press key "Enter" to confirm.

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

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



Later, there will be xrdp login screen. Enter the user name and password of RPi (RPi default user name: pi; password: raspberry) and click "OK".



Later, you can enter the RPi desktop system.

Here, you have successfully used the remote desktop login to RPi.

Raspberry Pi 4B/3B+/3B integrates a Wi-Fi adaptor. You can use it to connect to your Wi-Fi. Then you can use the wireless remote desktop to control your RPi.

VNC Viewer & VNC

Type the following command. And select 5 Interfacing Options→P3 VNC →Yes→OK→Finish. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

```
sudo raspi-config
```

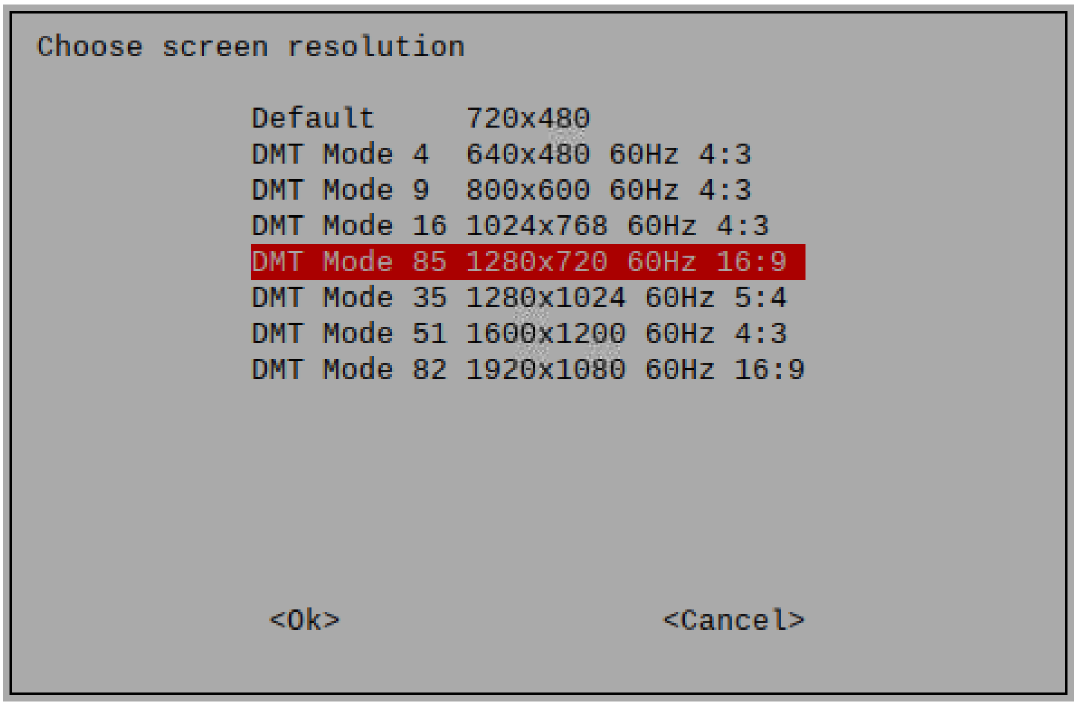
```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password          Change password for the current u
2 Network Options               Configure network settings
3 Boot Options                  Configure options for start-up
4 Localisation Options          Set up language and regional sett
5 Interfacing Options           Configure connections to peripher
6 Overclock                     Configure overclocking for your P
7 Advanced Options              Configure advanced settings
8 Update                        Update this tool to the latest ve
9 About raspi-config            Information about this configurat

              <Select>                     <Finish>
```

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

P1 Camera                       Enable/Disable connection to the
P2 SSH                          Enable/Disable remote command lin
P3 VNC                          Enable/Disable graphical remote a
P4 SPI                          Enable/Disable automatic loading
P5 I2C                          Enable/Disable automatic loading
P6 Serial                       Enable/Disable shell and kernel m
P7 1-Wire                       Enable/Disable one-wire interface
P8 Remote GPIO                  Enable/Disable remote access to G

              <Select>                     <Back>
```

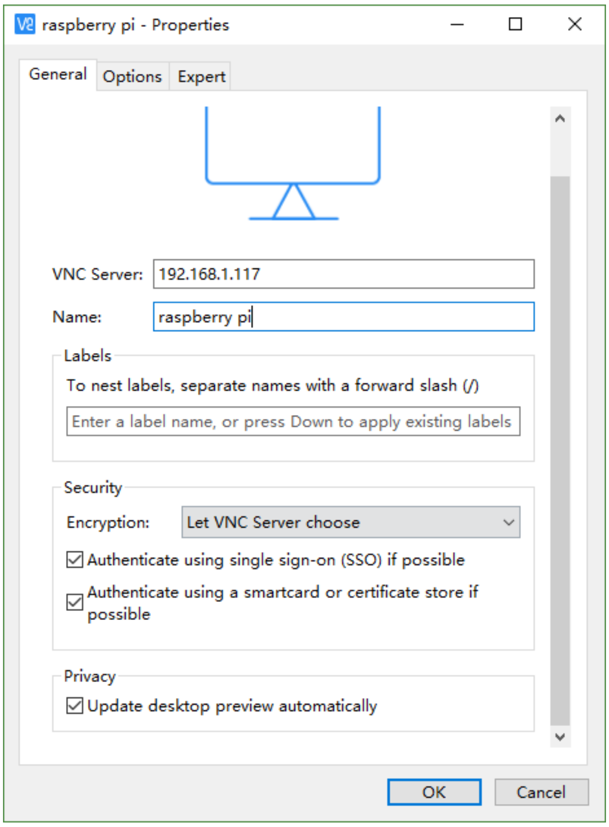
Then set resolution.

<Back>→ 7 Advanced Options→A5 Resolution→1280x720→OK→Finish.

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

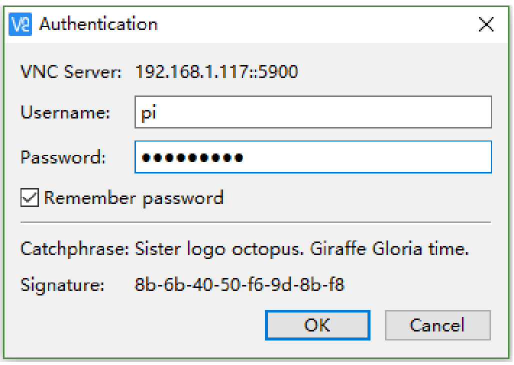


Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.



Enter IP address of your Raspberry Pi and fill in a name. Then click OK.
Then on the VNC Viewer panel, double-click new connection you just created,

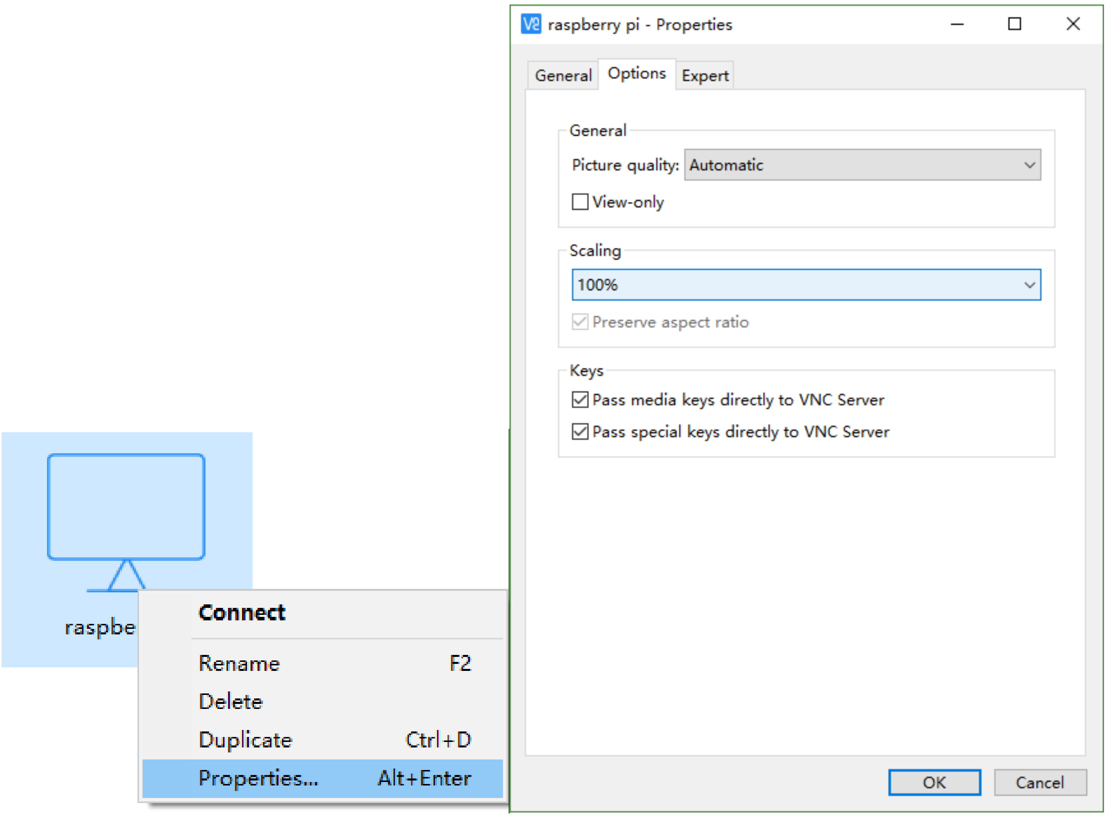
and the following dialog box pops up.



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

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

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



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

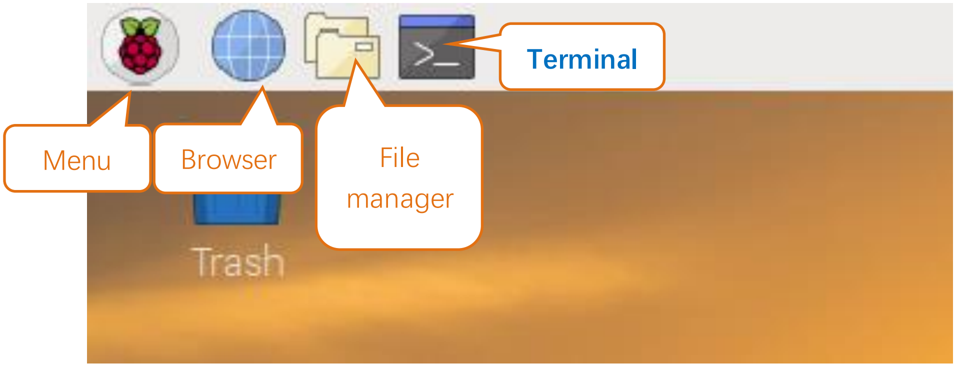
Raspberry Pi 4B/3B+/3B integrates a Wi-Fi adaptor.If you haven't connected Pi to WiFi, you can connect it to wirelessly control the robot.

Chapter 0 Preparation

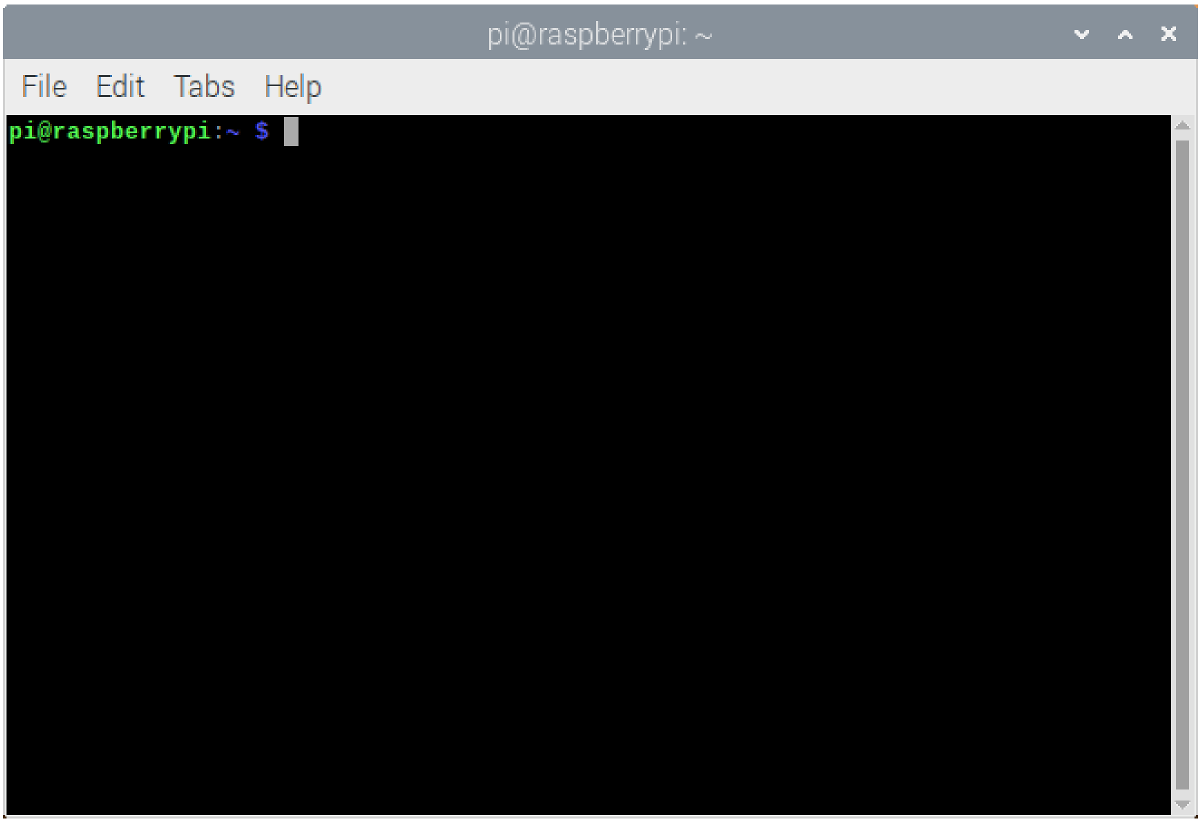
Why "Chapter 0"? Because in program code the first number is 0. We choose to follow this rule. In this chapter, we will do some necessary foundational preparation work: Start your Raspberry Pi and install some necessary libraries.

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 should be run in Terminal.



When you click the Terminal icon, following interface appears.



Note: The Linux is case sensitive.

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

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

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

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

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

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

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

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

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

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

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

Shortcut Key

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

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

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

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

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

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

Install WiringPi

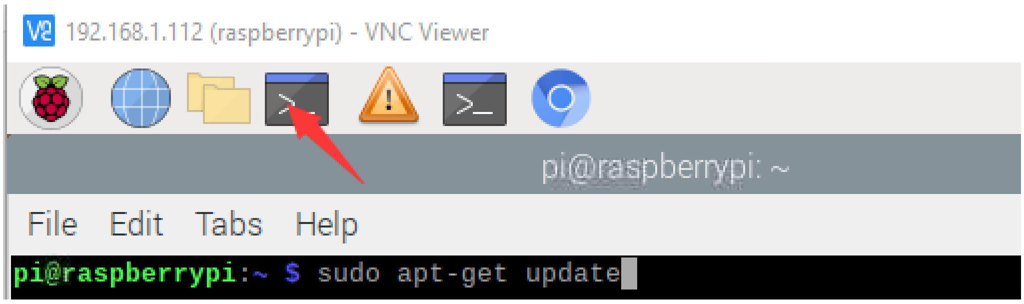
WiringPi is a GPIO access library written in C language for the BCM2835/BMC2836/BMC2837 used in the Raspberry Pi. It is released under the GNU LGPLv3 license and is usable from C, C++ and many other languages with suitable wrappers (See below). It is designed to be user friendly for those people who have had prior experience with the Arduino "wiring" system. (for more details, please refer to http://wiringpi.com/)

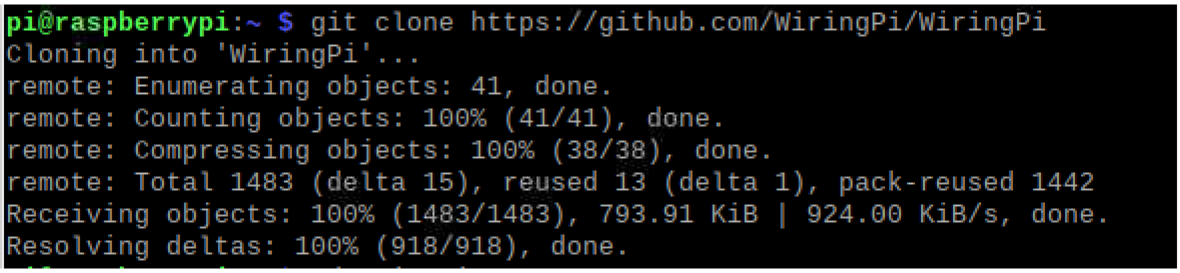
WiringPi Installation Steps

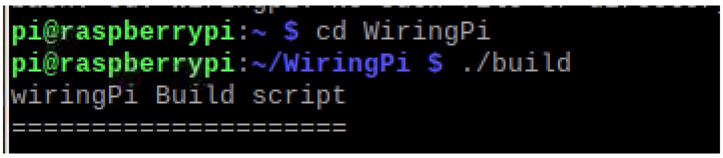
To install the WiringPi library, please open the Terminal and then follow the steps and commands below.
Note: For a command containing many lines, execute them **one line at a time.**
Enter the following command in the terminal to install WiringPi:

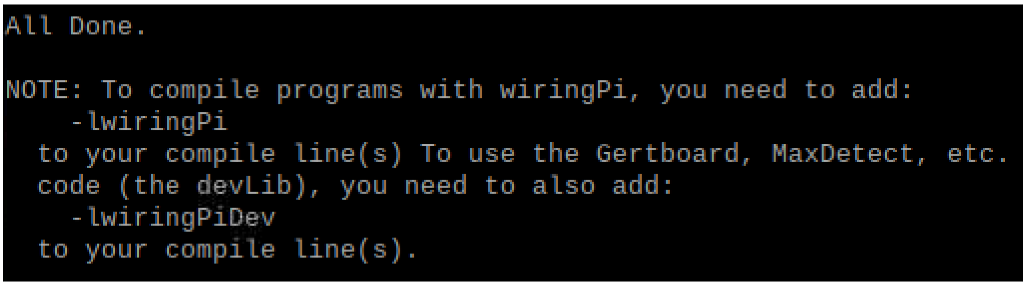
```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```

And then the installation will complete quickly as shown below.









Run the gpio command to check the installation:

```
gpio -v
```

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

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

Raspberry Pi Details:
```

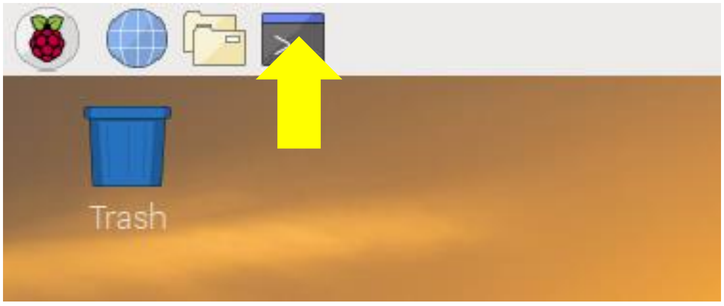
Obtain the Project Code

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

This is the method for obtaining the code:
In the pi directory of the RPi terminal, enter the following command.

```
cd
git clone https://github.com/Freenove/Freenove_Projects_Kit_for_Raspberry_Pi.git
```

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

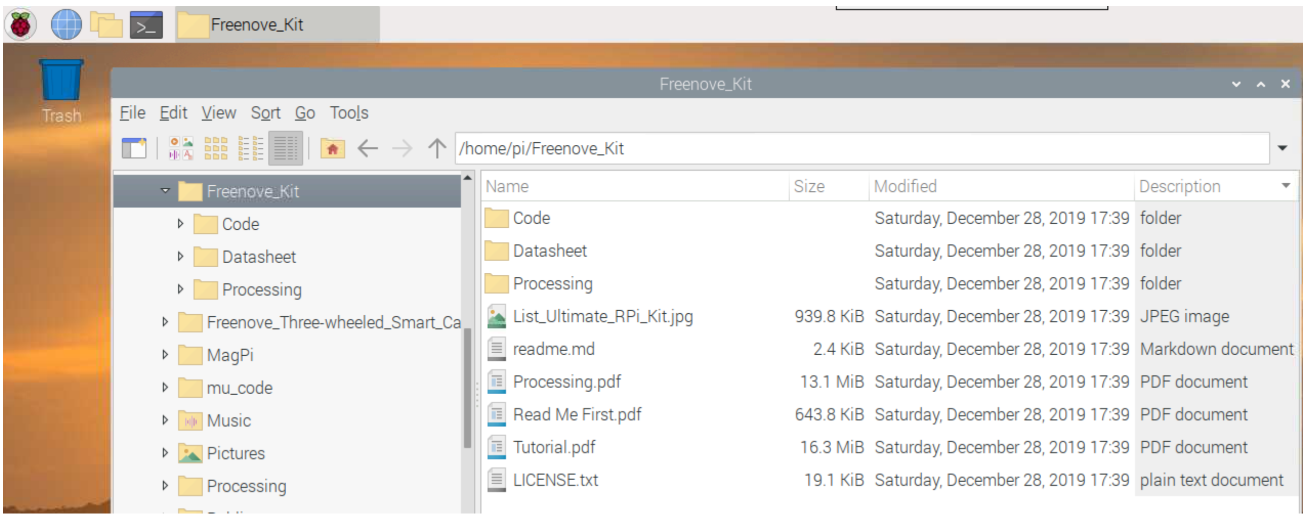


After the download is completed, a new folder "Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

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

```
mv Freenove_Projects_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

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



If you have no experience with Python, we suggest that you refer to this website for basic information and knowledge.
https://python.swaroopch.com/basics.html

Python2 & Python3

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

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

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

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

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

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

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

1. Enter directory /usr/bin

```
cd /usr/bin
```

2. Delete the old python link.

```
sudo rm python
```

3. Create new python links to python3.

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

4. Execute python to check whether the link succeeds.

```
python
```

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

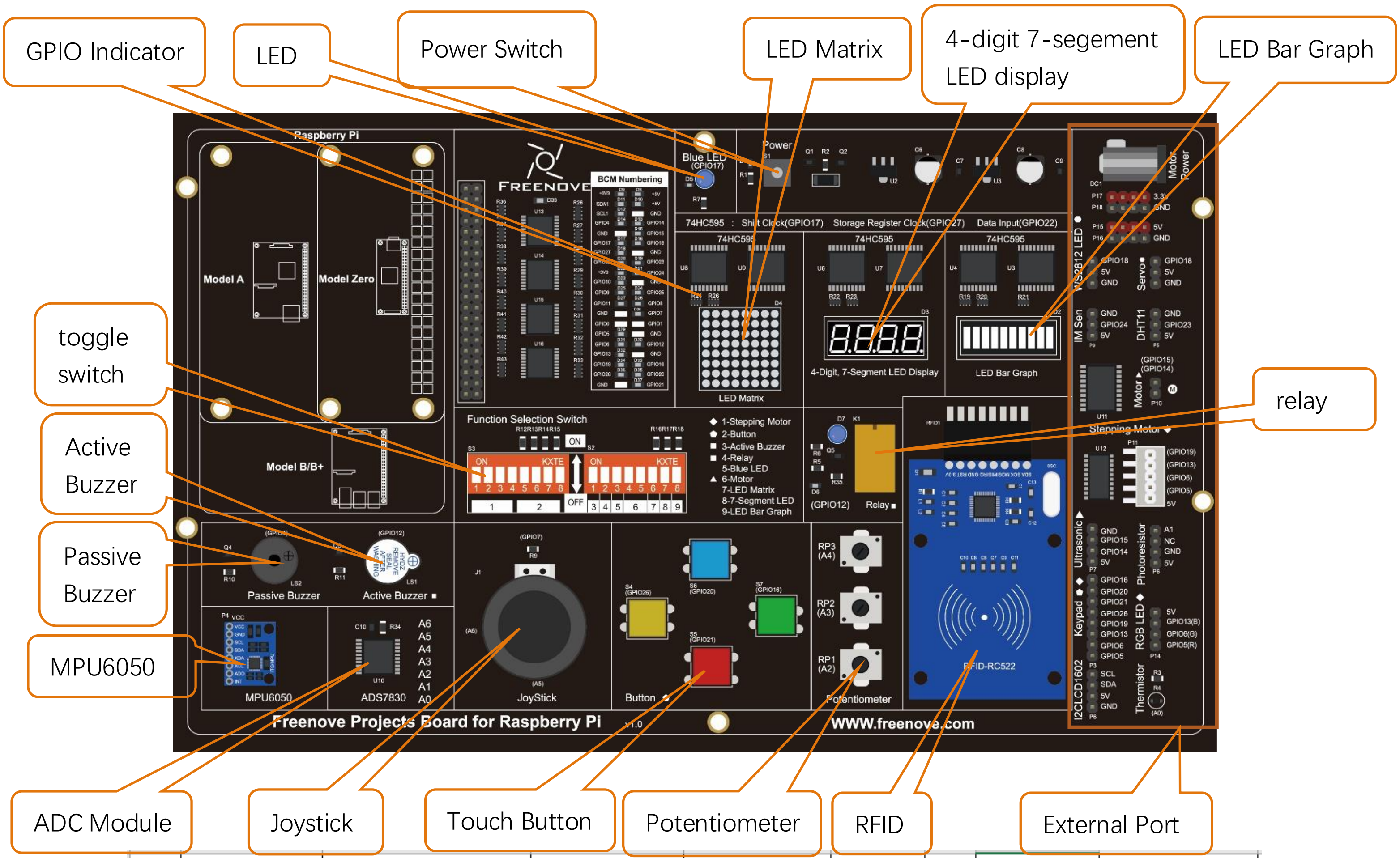
If you want to use Python2, repeat the steps above and just change the third command to the following:

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

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

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

Projects Board for Raspberry Pi

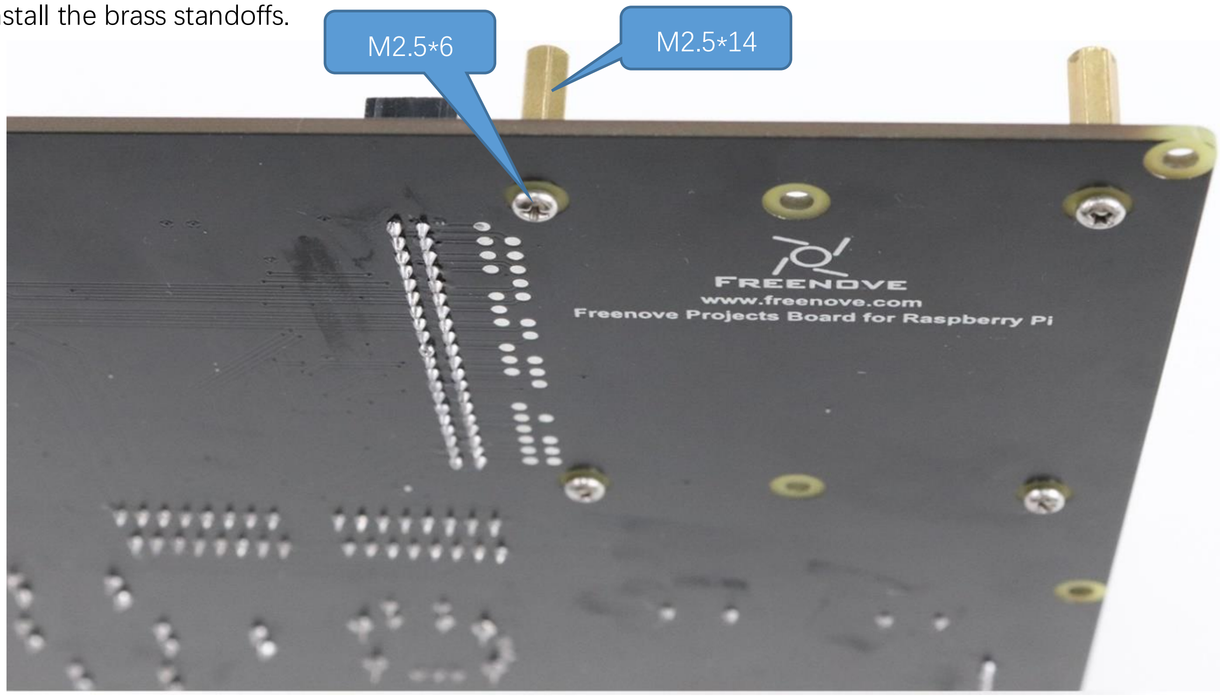


NO.	GPIO	Function1	Function2	Function3
1	GPIO2/SDA	LCD1602-SDA	MPU6050_SDA	ADS7830-SDA
2	GPIO3/SCL	LCD1602-SCL	MPU6050-SCL	ADS7830-SCL
3	GPIO4	Passive Buzzer		
4	GPIO17	Blue LED	LED Bar Graph	
5	GPIO27		Digital Tube	
6	GPIO22		LED Matrix	
7	GPIO10/MOSI	RFID-RC522		
8	GPIO9/MISO	RFID-RC522		
9	GPIO11/SCLK	RFID-RC522		
10	GPIO0/ID_SD			
11	GPIO5	Stepping Motor-IN4	Keypad	RGBLED-R
12	GPIO6	Stepping Motor-IN3	Keypad	RGBLED-G
13	GPIO13	Stepping Motor-IN2	Keypad	RGBLED-B
14	GPIO19/MISO	Stepping Motor-IN1	Keypad	
15	GPIO26	Button1	Keypad	
16	GPIO21/SCLK	Button2	Keypad	
17	GPIO20/MOSI	Button3	Keypad	
18	GPIO16/CE2#	Button4	Keypad	
19	GPIO12	Active Buzzer	Relay	
20	GPIO1/ID_SC			
21	GPIO7/CE1#	Joystick-z		
22	GPIO8/CE0#	RFID-RC522		
23	GPIO25	RFID-RC522		
24	GPIO24	IM Sensor		
25	GPIO23	DHT11		
26	GPIO18	WS2812	Servo	
27	GPIO15/RXD0	Motor-IN2		
28	GPIO14/TXD0	Motor-IN1		

NO.	ADC Pin	Function
1	A0	Thermistor
2	A1	Photoresistor
3	A2	Potentiometer3
4	A3	Potentiometer2
5	A4	Potentiometer1
6	A5	Joystick-x
7	A6	Joystick-y
8	A7	

Assembly

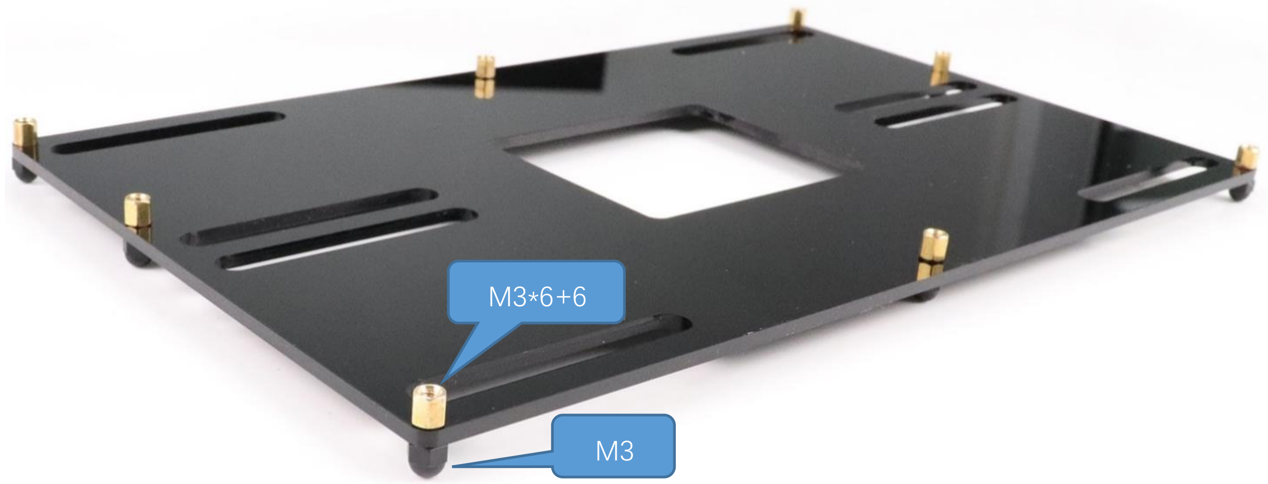
Install the brass standoffs.



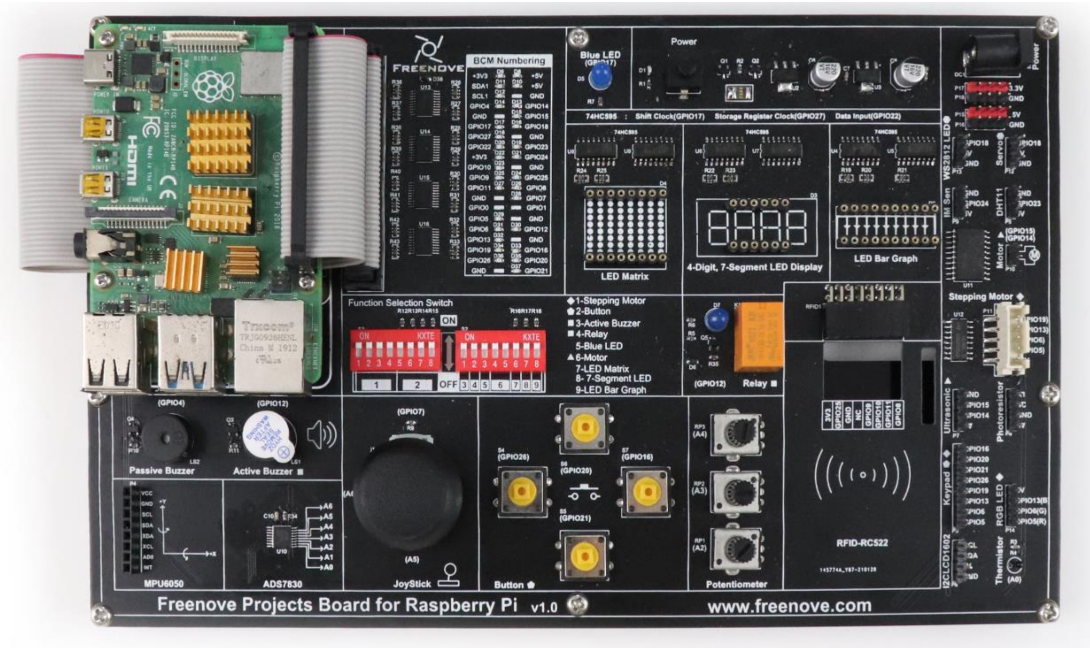
Finish

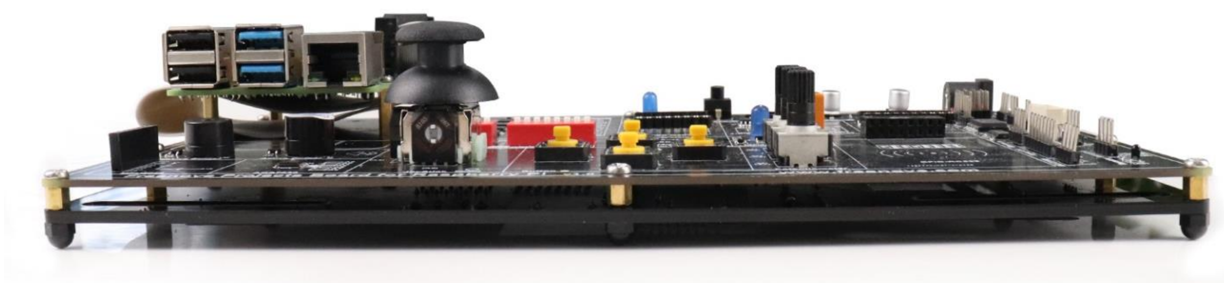
Install the Raspberry Pi.

Install the acrylic part



Finish


Freenove Projects Board for Raspberry Pi v1.0
www.freenove.com
Function Selection Switch
LED Matrix
4-Digit, 7-Segment LED Display
LED Bar Graph
Passive Buzzer
Active Buzzer
JoyStick
Button
Potentiometer
RFID-RC522
MPU6050
ADS7830
Stepping Motor
Relay



Chapter 1 LED

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

Project 1.1 Blink

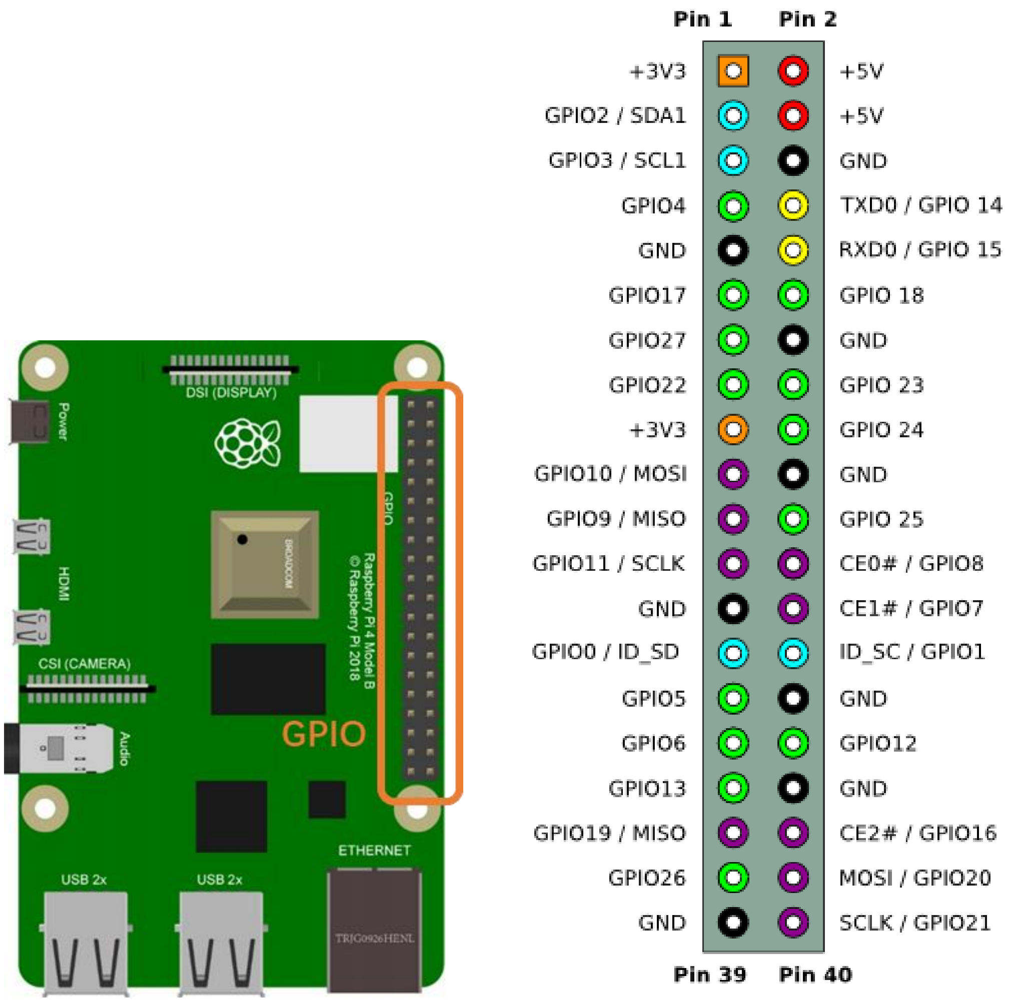
In this project, we will use RPi to control blinking a common LED.

GPIO

GPIO: General Purpose Input/Output. Here we will introduce the specific function of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions.
When programming GPIO pins, there are 3 different ways to reference them: GPIO Numbering, Physical Numbering and WiringPi GPIO Numbering.

BCM GPIO Numbering

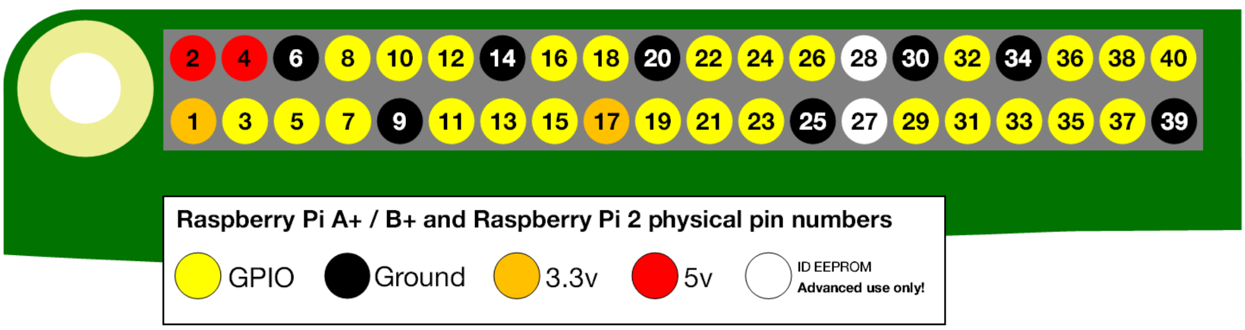
The Raspberry Pi CPU uses Broadcom (BCM) processing chips BCM2835, BCM2836 or BCM2837. GPIO pin numbers are assigned by the processing chip manufacturer and are how the computer recognizes each pin. The pin numbers themselves do not make sense or have meaning as they are only a form of identification.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



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

PHYSICAL Numbering

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



WiringPi GPIO Numbering

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

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

For Pi B (header pins 1–26)

For A+, B+, 2B, 3B, 3B+, 4B, Zero (header pins 1–40)

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

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

```
gpio readall
```

```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

Expect to have errors when executing the command “gpio readall” if you are using Raspberry Pi 4B (as shown below):

```
pi@raspberrypi:~ $ gpio readall
Oops - unable to determine board type... model: 17
```

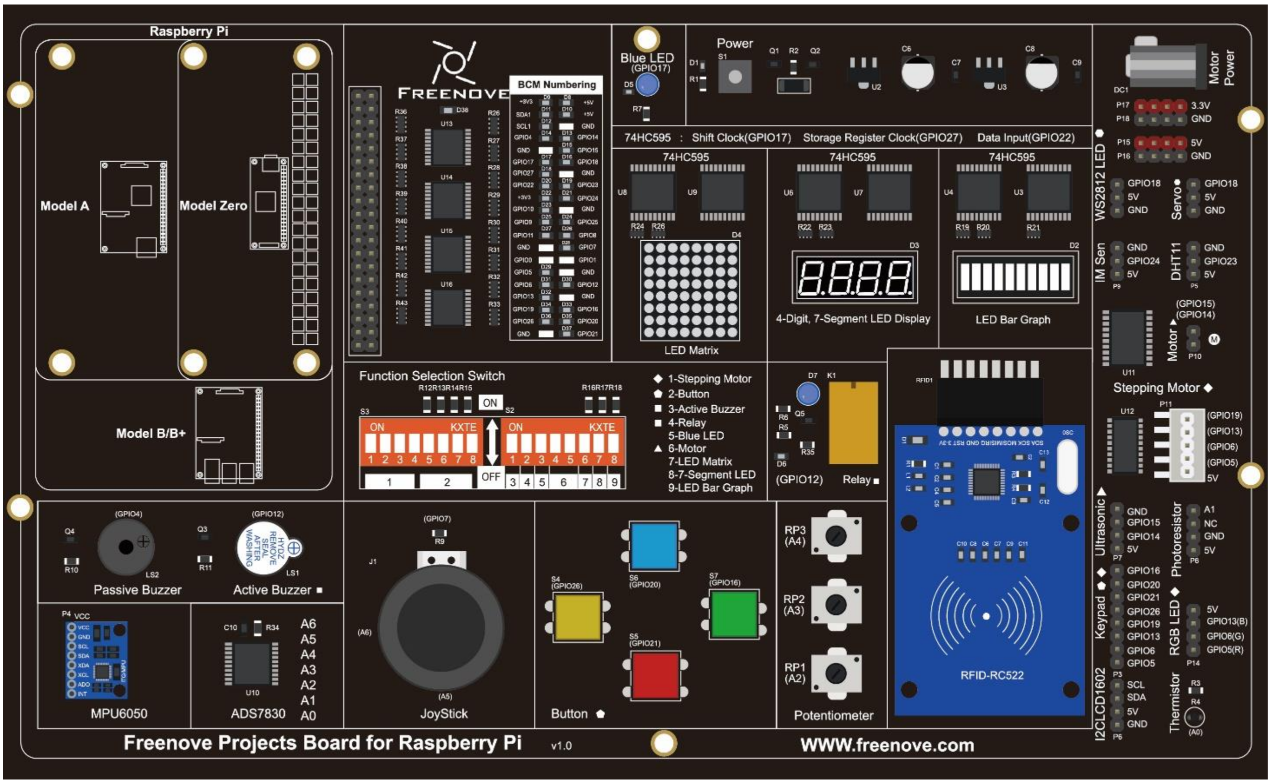
This is because the official version of the library supporting RPI 4B, as of this writing, has not yet been released. This results in some commands not functioning properly. However, the following projects will not be affected. This problem can be solved by installing a patch. Just execute the commands below in the Terminal.

```
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

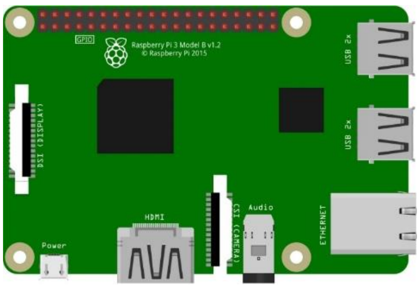
For more details about wiringPi, please refer to http://wiringpi.com/ .

Component List

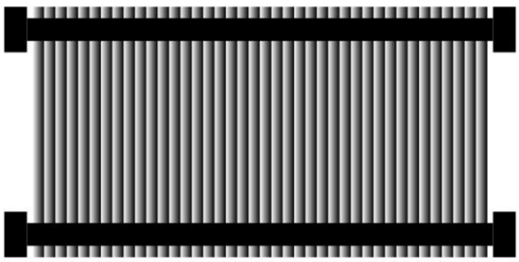
Freenove Projects Board for Raspberry Pi



Raspberry Pi

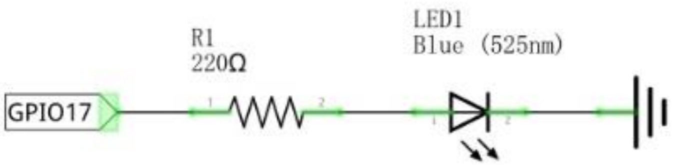


GPIO Ribbon Cable



Circuit

Schematic diagram



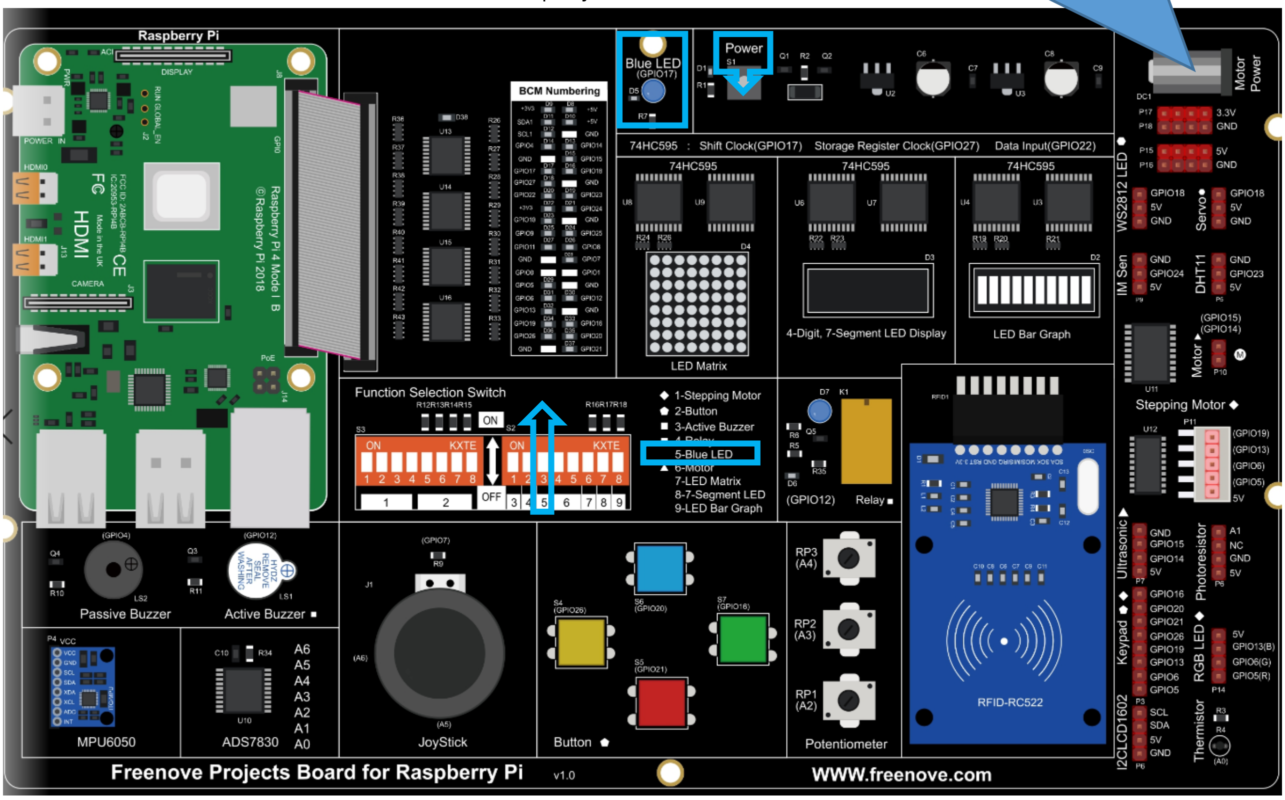
GPIO Resistor LED GND

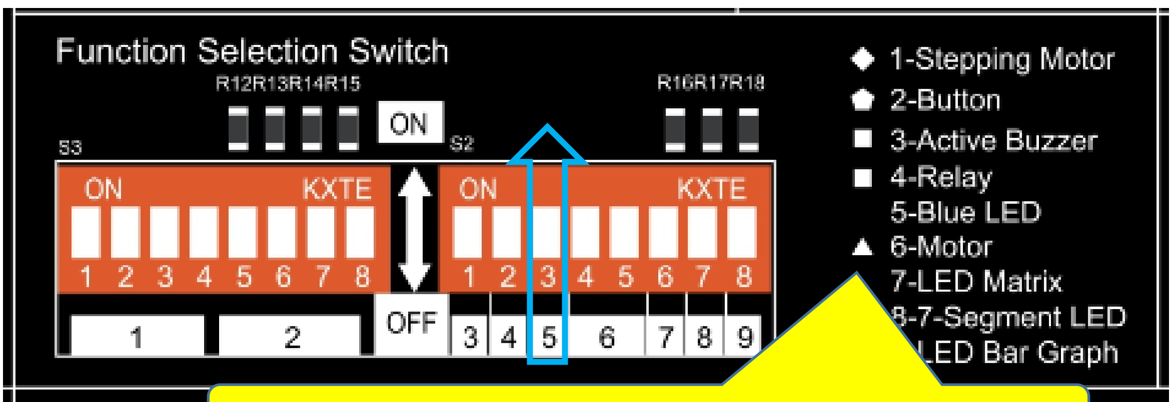
Hardware connection:

Turn ON the power switch and NO.5 toggle switch.

Power switch should be turned ON in all the projects.

It will be introducd when it is needed.





Modules with same mark can't be used as the same time.

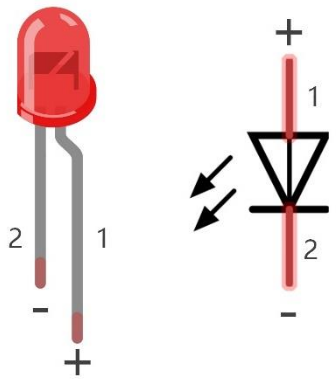
If you have any concerns, please send an email to: support@freenove.com

Component knowledge

LED

An LED is a type of diode. All diodes have two Poles and only work if current is flowing in the correct direction. 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.



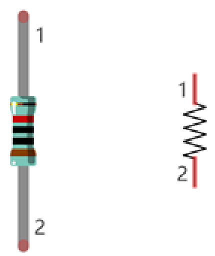
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

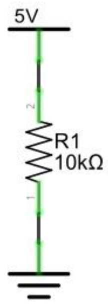
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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)

Code

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

C Code 1.1 Blink

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

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

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

It is recommended to execute the code via command line.

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

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

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

```
cd ~/Freenove_Kit/Code/C_Code/1_Blink
```

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

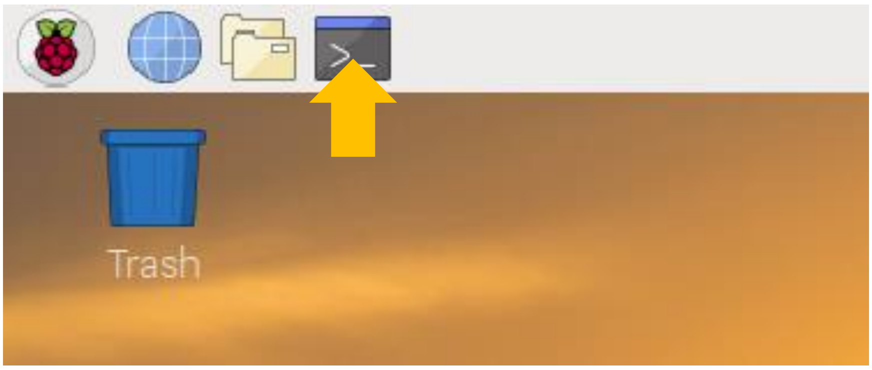
“l” of “lwiringPi” is low case of “L”.

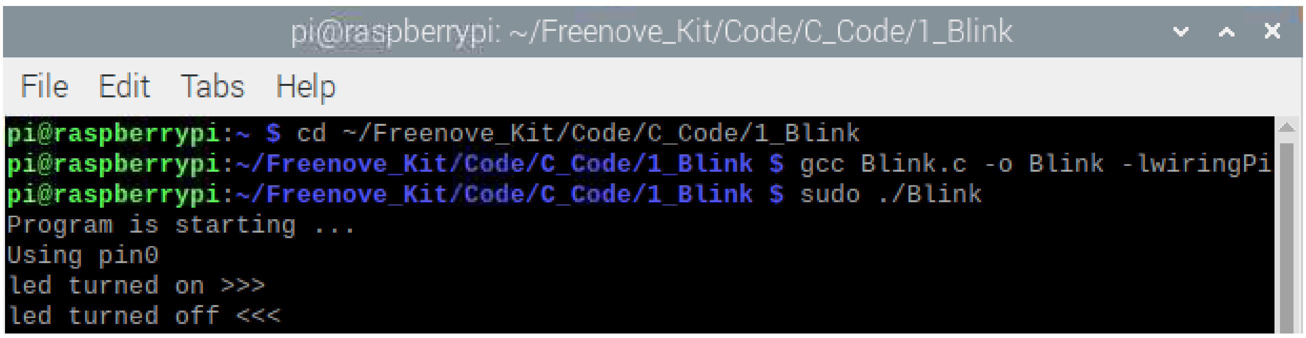
```
gcc Blink.c -o Blink -lwiringPi
```

4. Then run the generated file “blink”.

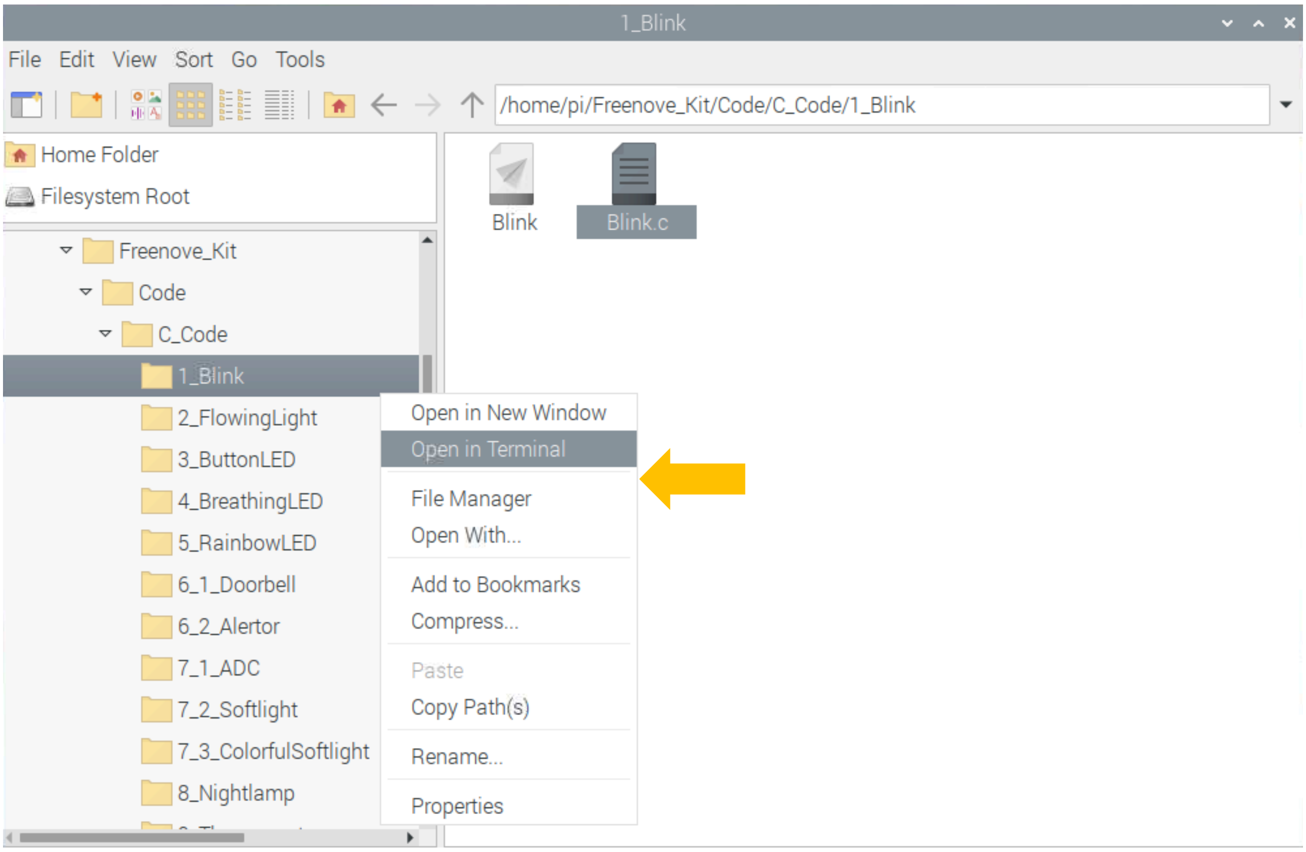
```
sudo ./Blink
```

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





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



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

```
#include <wiringPi.h>
#include <stdio.h>
#define  ledPin    0  //define the led pin number
void main(void)
{
    printf("Program is starting ... \n");
    wiringPiSetup();  //Initialize wiringPi.
    pinMode(ledPin, OUTPUT);//Set the pin mode
    printf("Using pin%d\n",ledPin);    //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
        printf("led turned on >>>\n");      //Output information on terminal
        delay(1000);                        //Wait for 1 second
        digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
        printf("led turned off <<<\n");         //Output information on terminal
        delay(1000);                        //Wait for 1 second
    }
}
```

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

`void pinMode(int pin, int mode);`	
This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes. This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program	
`void digitalWrite (int pin, int value);`	
Writes the value HIGH or LOW (1 or 0) to the given pin, which must have been previously set as an output.	

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

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

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

GPIO Numbering Relationship

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

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

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

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

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

Python Code 1.1 Blink

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

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

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

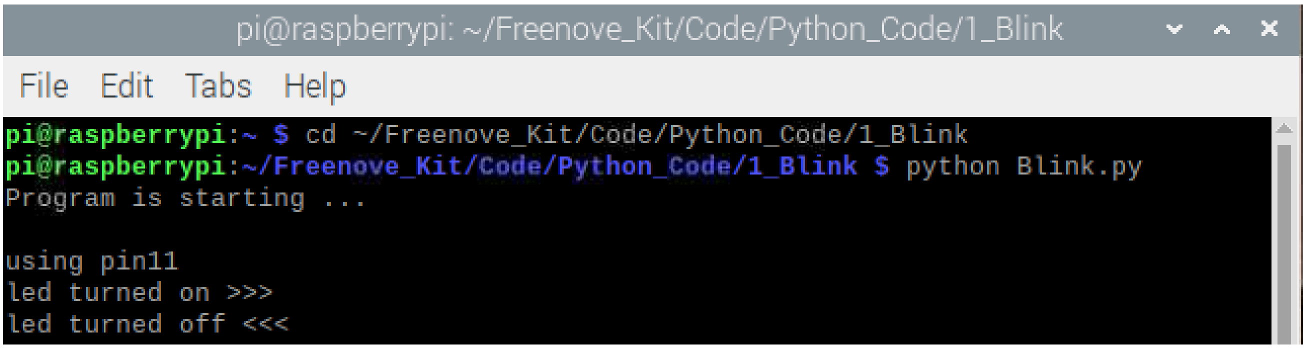
1. Use cd command to enter 1_Blink directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/1_Blink
```

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

```
python Blink.py
```

The LED starts blinking.



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

```python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
ledPin = 11    # define ledPin
def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)   # set the ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW)  # make ledPin output LOW level
    print ('using pin%d'%ledPin)

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

def destroy():
    GPIO.cleanup()                      # Release all GPIO

if __name__ == '__main__':     # Program entrance
```

```
    print ('Program is starting ... \n')
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

About RPi.GPIO:

RPi.GPIO
This is a Python module to control the GPIO on a Raspberry Pi. It includes basic output function and input function of GPIO, and functions used to generate PWM.
GPIO.setmode(mode)
Sets the mode for pin serial number of GPIO. mode=GPIO.BOARD, which represents the GPIO pin serial number based on physical location of RPi. mode=GPIO.BCM, which represents the pin serial number based on CPU of BCM chip.
GPIO.setup(pin,mode)
Sets pin to input mode or output mode, "pin" for the GPIO pin, "mode" for INPUT or OUTPUT.
GPIO.output(pin,mode)
Sets pin to output mode, "pin" for the GPIO pin, "mode" for HIGH (high level) or LOW (low level).

For more functions related to RPi.GPIO, please refer to:
https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/
"import time" time is a module of python.
https://docs.python.org/2/library/time.html?highlight=time%20time#module-time

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

```
ledPin = 11    # define ledPin
def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)   # set the ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW)  # make ledPin output LOW level
    print ('using pin%d'%ledPin)
```

GPIO Numbering Relationship

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

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

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

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

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

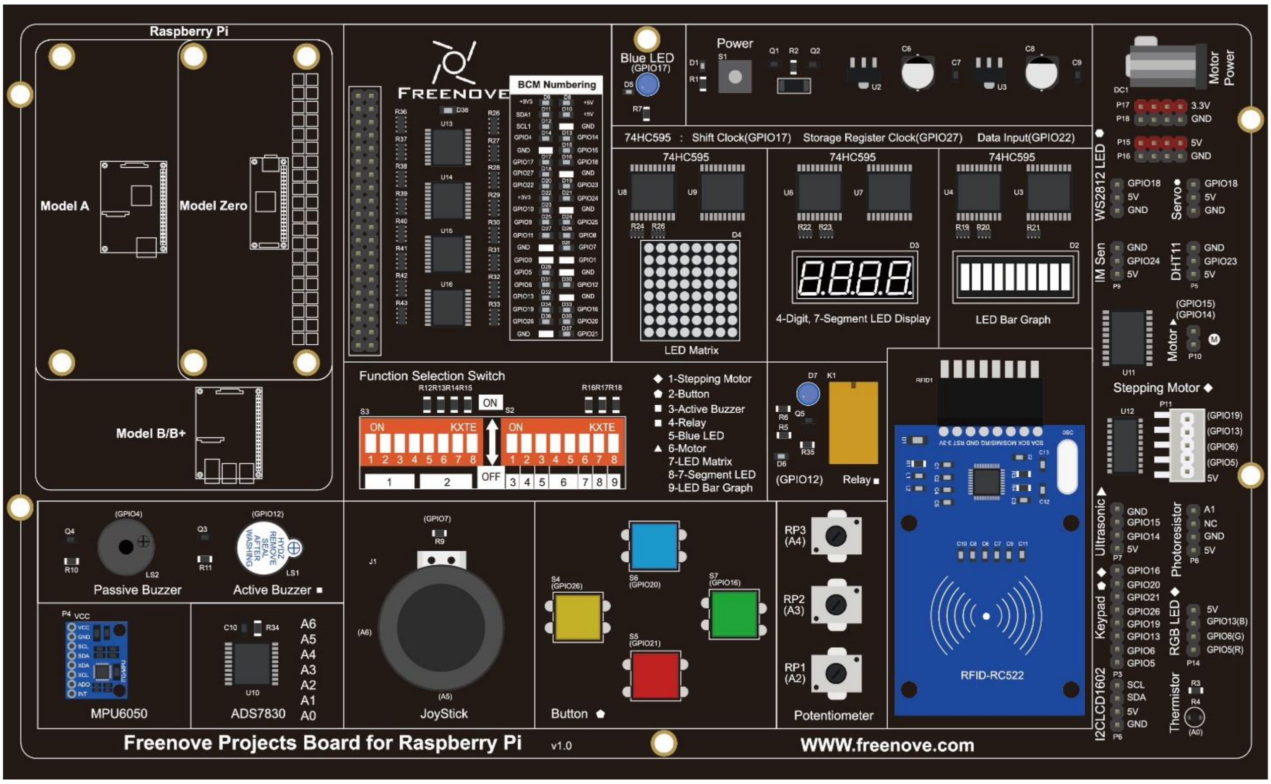
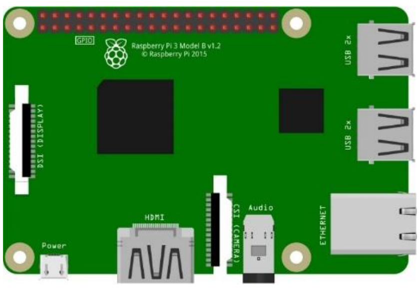
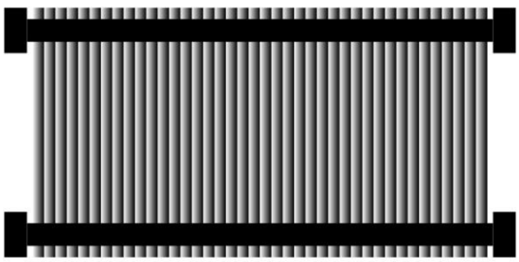
Chapter 2 FlowingLight

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

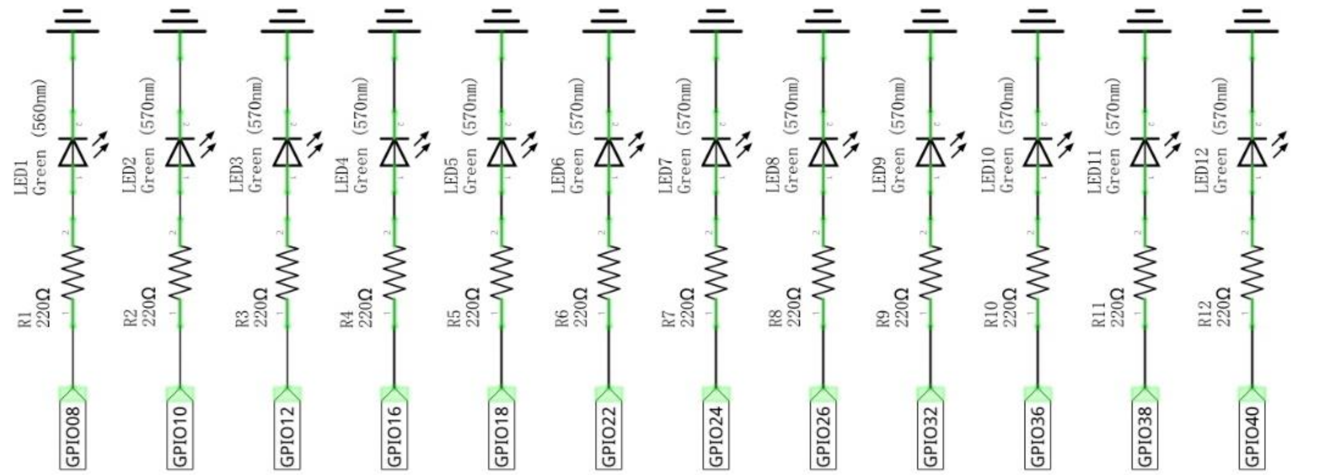
In this project, we use a number of LEDs to make a flowing water light.

Component List

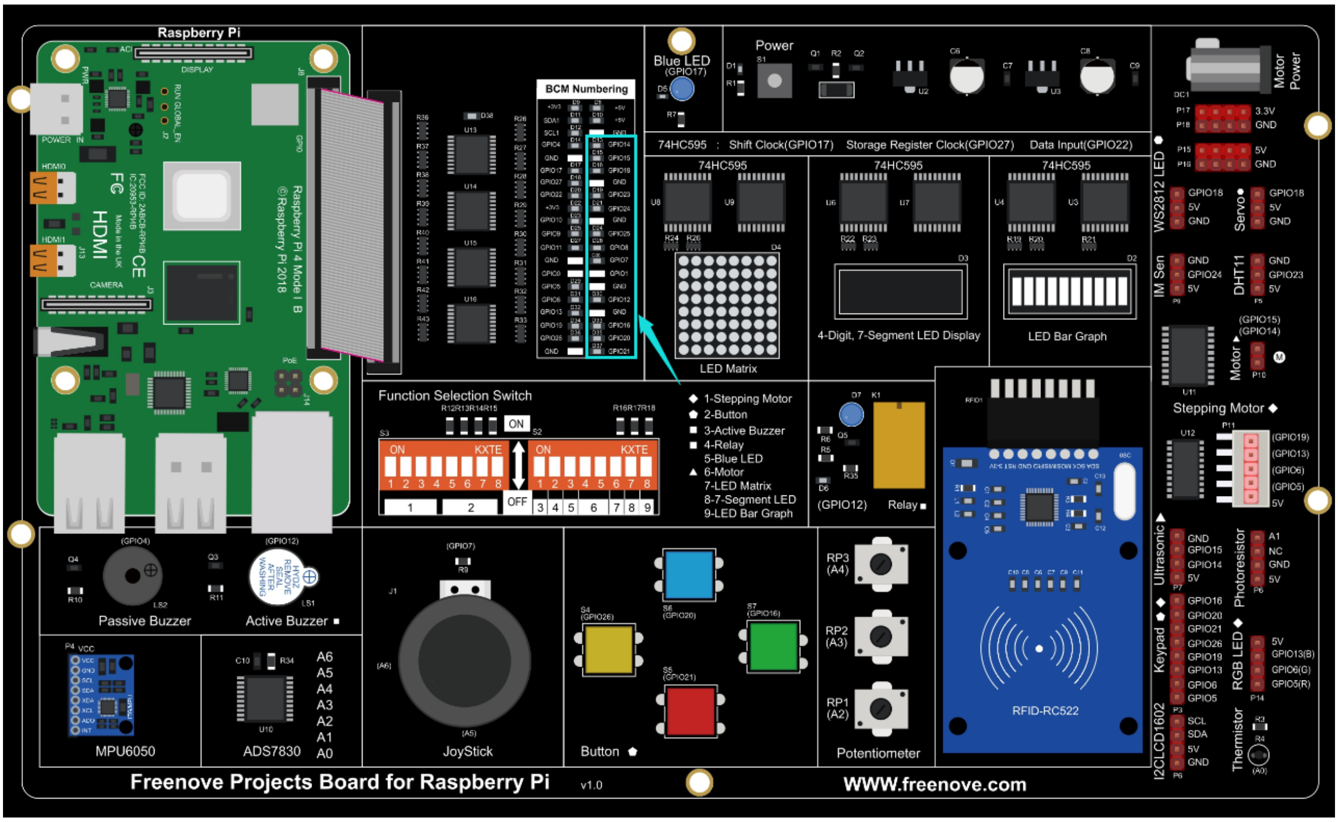
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td>
<td>GPIO Ribbon Cable
</td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 2.1 LightWater

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

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

1. Use cd command to enter 2_FlowingLight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/2_FlowingLight
```

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

```
gcc FlowingLight.c -o FlowingLight -lwiringPi
```

3. Then run the generated file "LightWater".

```
sudo ./FlowingLight
```

You can see the LEDs lighting from top to bottom and then back from bottom to top.

The following is the program code:

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

#define ledCounts 12
int pins[ledCounts] = {15,16,1,4,5,6,10,11,26,27,28,29};

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

    wiringPiSetup(); //Initialize wiringPi.

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

In the “while” loop, apply two “for” loop to achieve the flowing water light lighting from top to bottom and then back from bottom to top.

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

Python Code 2.1 LightWater

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

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

1. Use cd command to enter 2_FlowingLight directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/2_FlowingLight
```

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

```
python LightWater.py
```

You can see the LEDs lighting from top to bottom and then back from bottom to top.

The following is the program code:

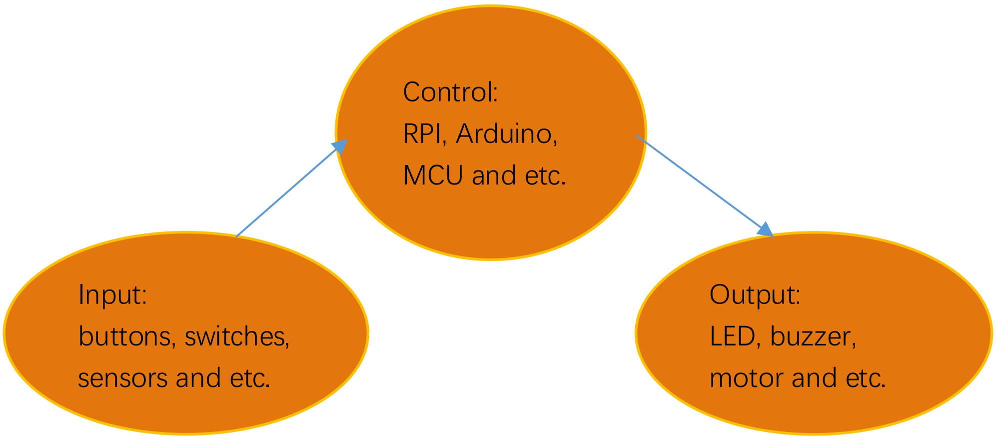
```
import RPi.GPIO as GPIO
import time

ledPins = [8, 10, 12, 16, 18, 22, 24, 26, 32, 36, 38, 40]

def setup():
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPins, GPIO.OUT)   # set all ledPins to OUTPUT mode
    GPIO.output(ledPins, GPIO.HIGH) # make all ledPins output HIGH level, turn off all led

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

def destroy():
    GPIO.cleanup()                     # Release all GPIO

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

In the “while” loop, apply two “for” loop to achieve the flowing water light lighting from top to bottom and then back from bottom to top.

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

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.

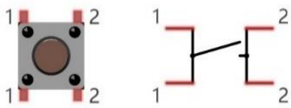
Project 3.1 Push Button Switch & LED

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 knowledge

Push Button Switch

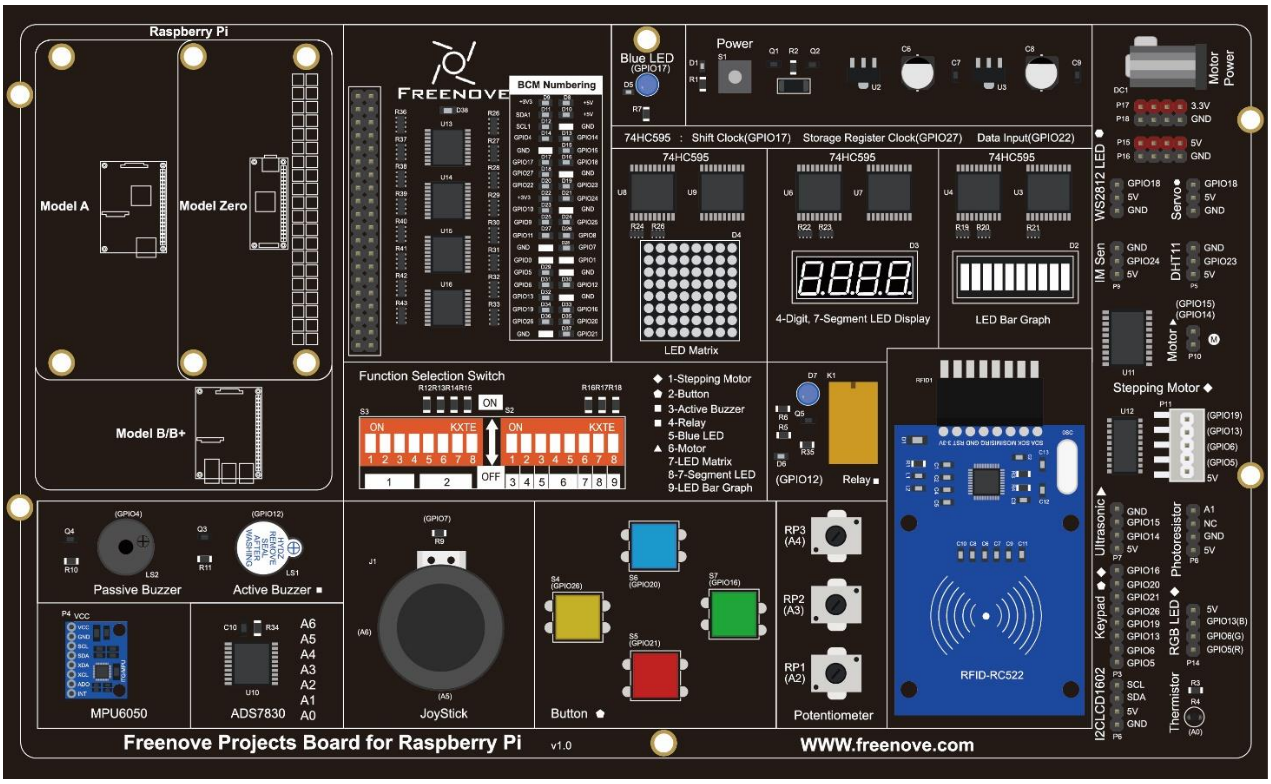
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same as per the illustration:



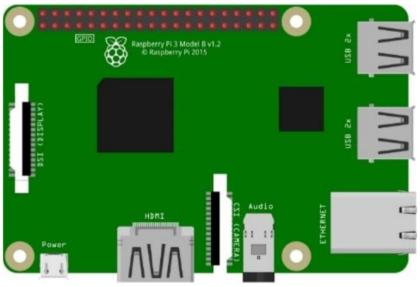
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Component List

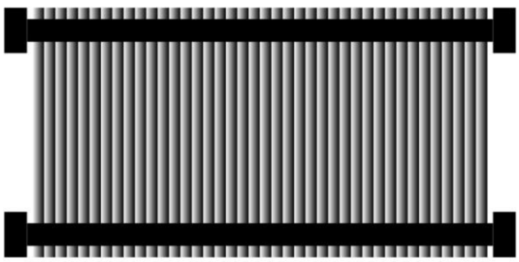
Freenove Projects Board for Raspberry Pi



Raspberry Pi

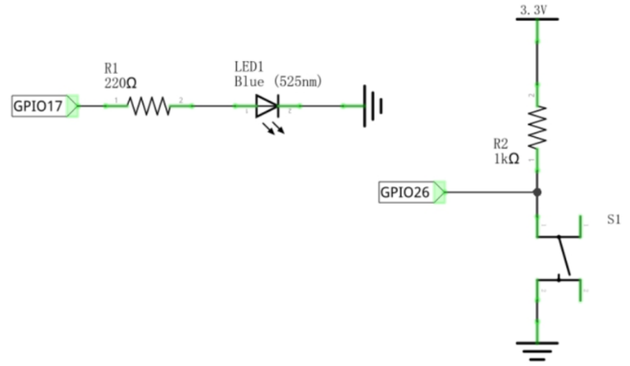


GPIO Ribbon Cable



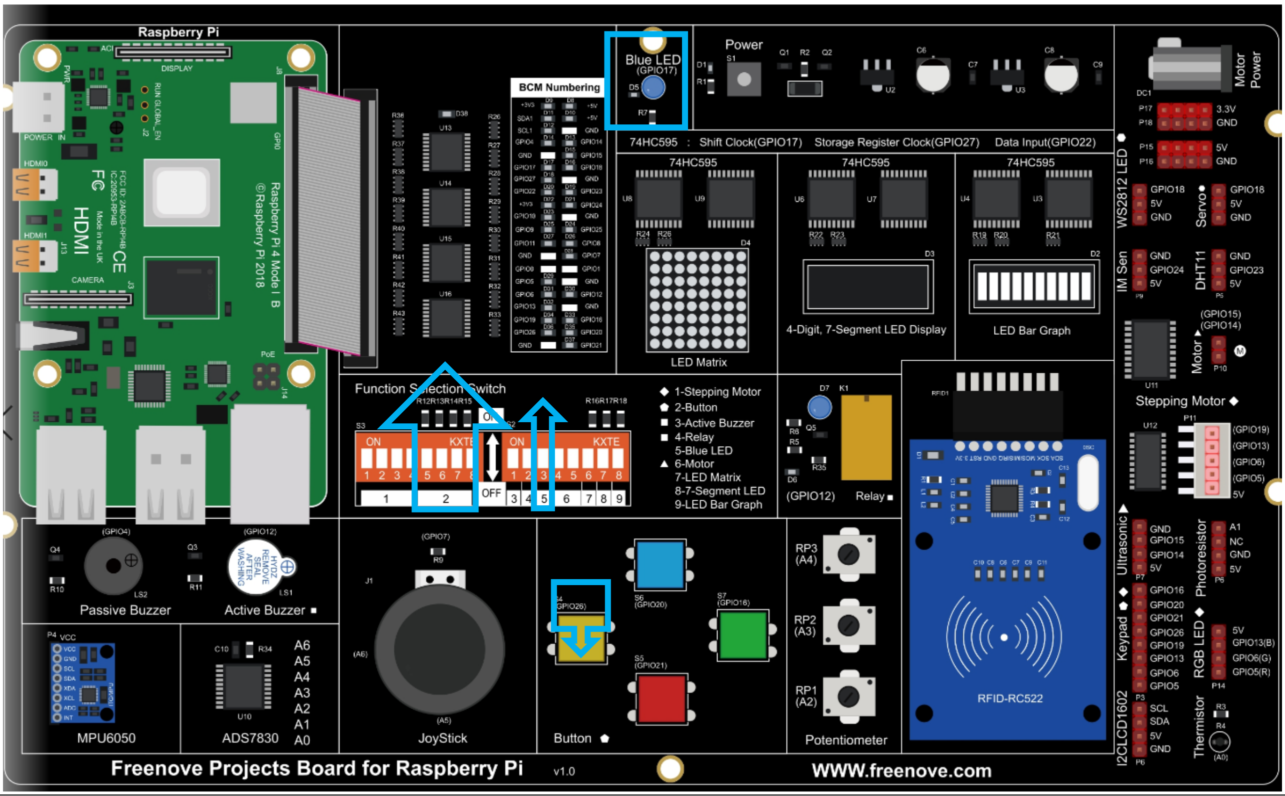
Circuit

Schematic diagram



Hardware connection.

Switch ON NO.5 switch and the four switches of NO.2.



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

Code

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

C Code 3.1 ButtonLED

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

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

1. Use cd command to enter 3_ButtonLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/3_ButtonLED
```

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

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

3. Then run the generated file "ButtonLED".

```
sudo ./ButtonLED
```

Later, the terminal window continues to print out the characters "led off…". Press the **S4** button, then LED is turned on and then terminal window prints out the "led on…". Release the button, then LED is turned off and then terminal window prints out the "led off…". You can press "Ctrl+C" to terminate the program.

The following is the program code:

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

#define ledPin    0   //define the ledPin
#define buttonPin 25      //define the buttonPin

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

    wiringPiSetup();  //Initialize wiringPi.

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

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

```
terminal
        }
    }
}
```

Define ledPin and buttonPin as 0 and 25 respectively.

```
#define ledPin     0    //define the ledPin
#define buttonPin 25       //define the buttonPin
```

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

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

Reference:

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

Python Code 3.1 ButtonLED

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

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

1. Use cd command to enter 3_ButtonLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/3_ButtonLED
```

2. Use Python command to execute btnLED.py.

```
python ButtonLED.py
```

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

The following is the program code:

```
import RPi.GPIO as GPIO

ledPin = 11     # define ledPin
buttonPin = 37     # define buttonPin

def setup():

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

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

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

if __name__ == '__main__':      # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

In subfunction setup (), GPIO.setmode (GPIO.BOARD) is used to set the serial number of the GPIO, which is based on physical location of the pin. Therefore, GPIO17 and GPIO26 correspond to pin11 and pin37 respectively in the circuit. Then set ledPin to output mode, buttonPin to input mode with a pull resistor.

```
ledPin = 11     # define ledPin
buttonPin = 37    # define buttonPin

def setup():

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

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

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

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

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

About function GPIO.input ():

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

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.

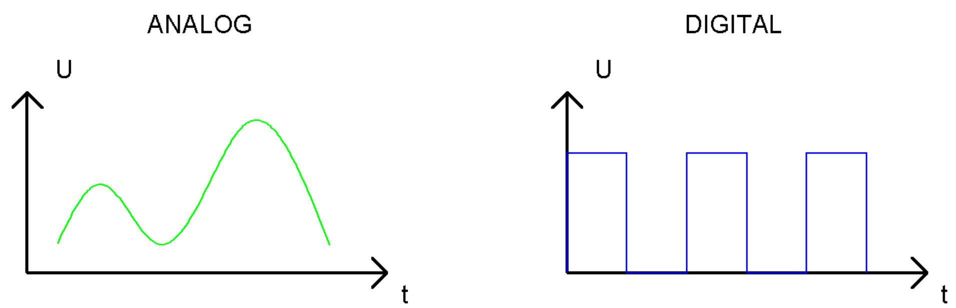
Project 4.1 Breathing LED

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 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.

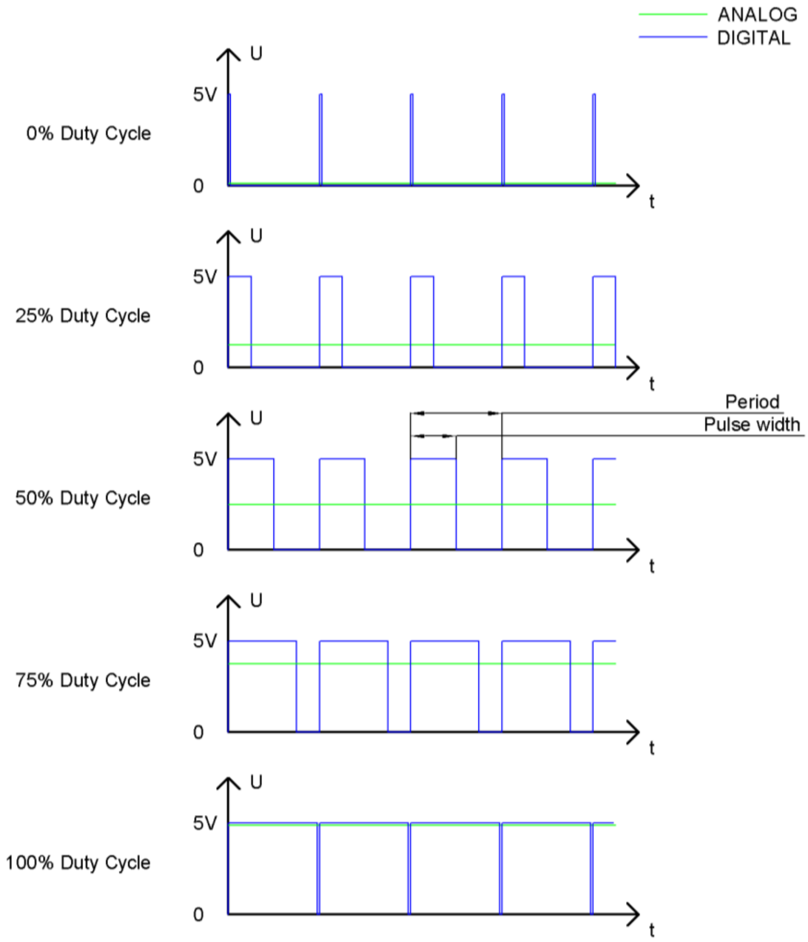


Note that the Analog signals are curved waves and the Digital signals are "Square Waves".
In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog 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%:



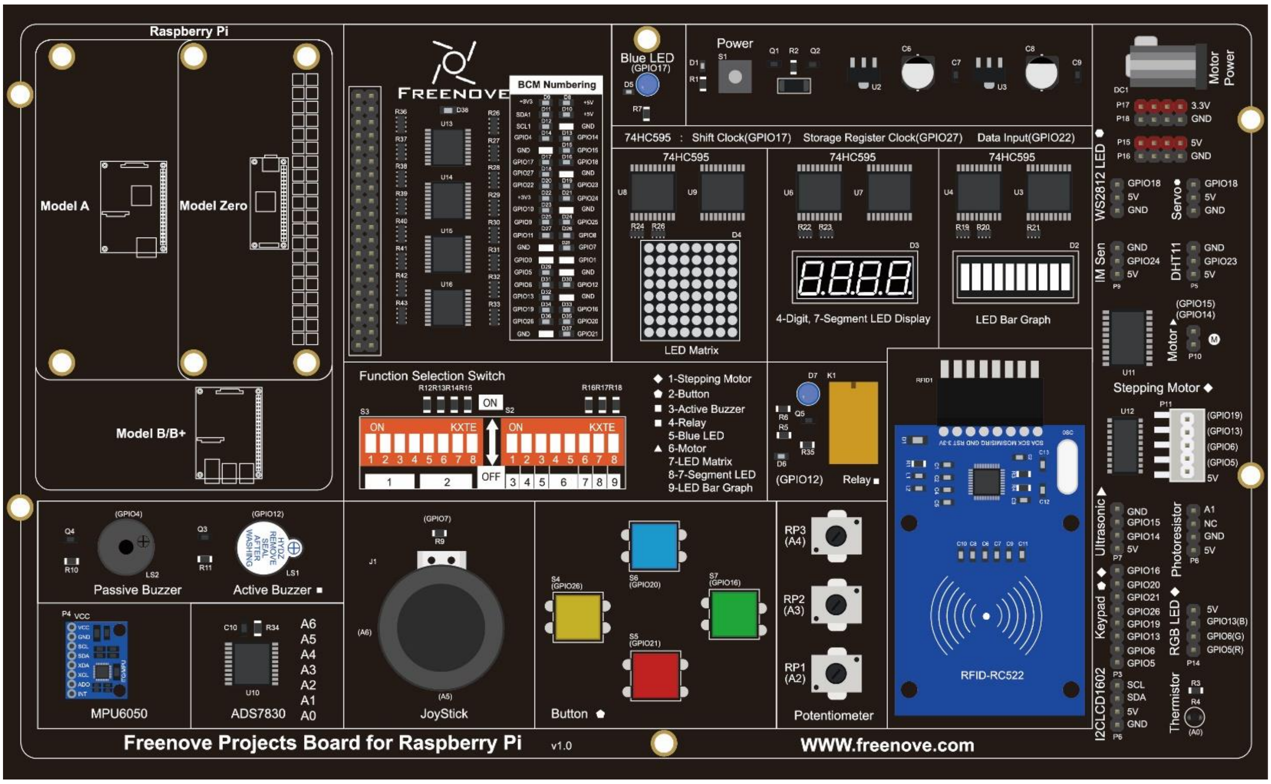
The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.
It is evident, from the above, that PWM is not actually analog but the effective value of voltage is 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, while the wiringPi library of Python does not provide a hardware PWM method. There is only a software PWM option for Python.

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

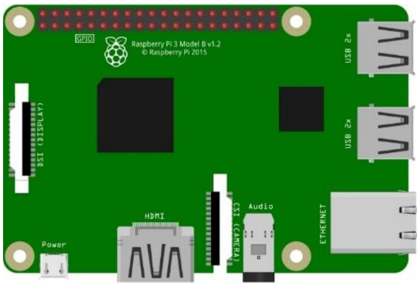
In order to keep the results running consistently, we will use PWM.

Component List

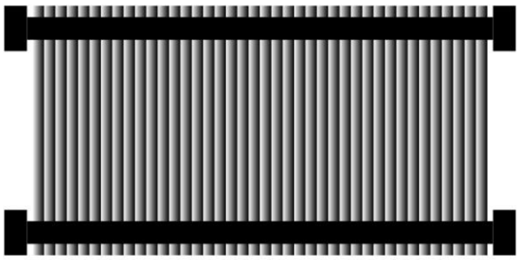
Freenove Projects Board for Raspberry Pi



Raspberry Pi

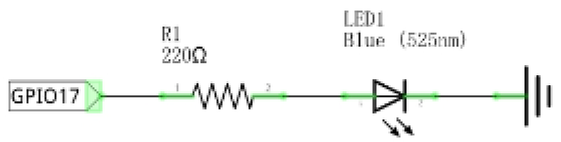


GPIO Ribbon Cable

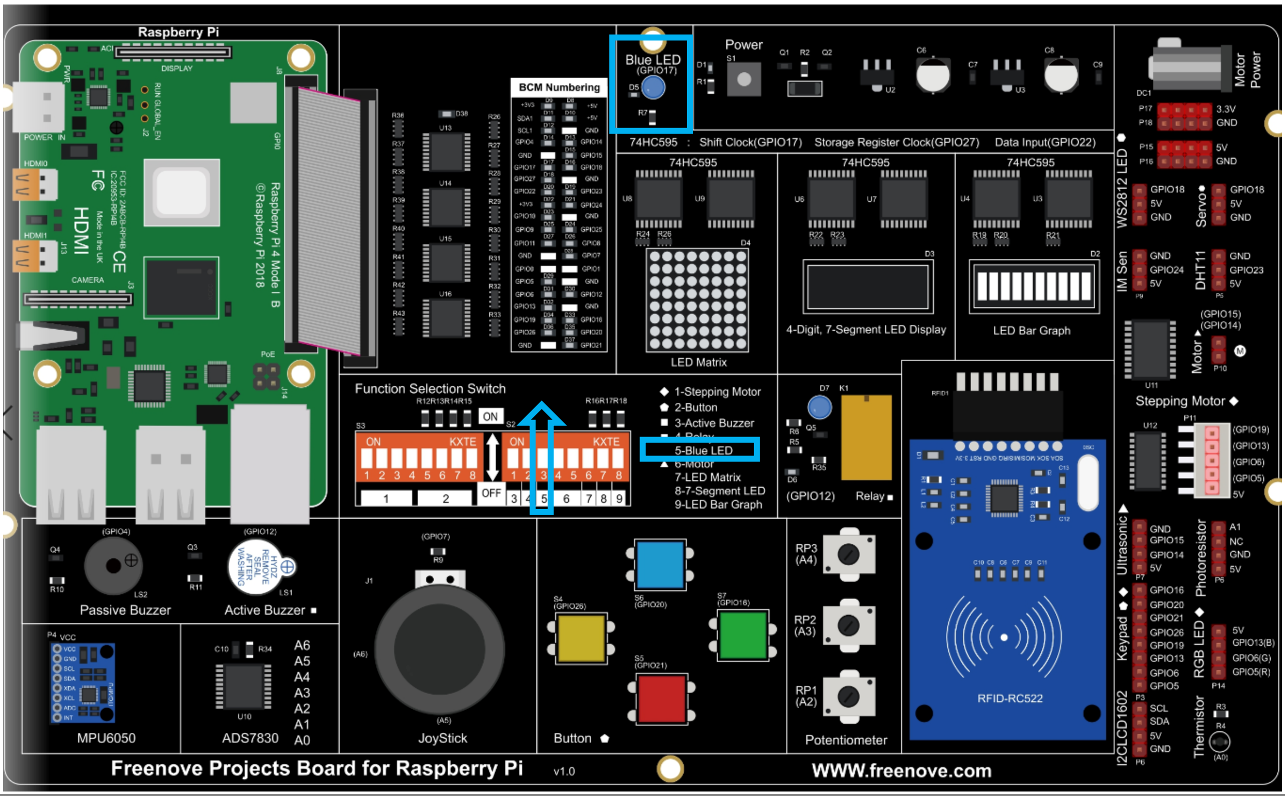


Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 4.1 BreathingLED

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

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

1. Use cd command to enter 4_BreathingLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/4_BreathingLED
```

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

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

3. Then run the generated file “BreathingLED”

```
sudo ./BreathingLED
```

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

The following is the program code:

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

#define ledPin    0

void main(void)
{
    int i;

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

    wiringPiSetup();  //Initialize wiringPi.

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

    while(1){
        for(i=0;i<100;i++){  //make the led brighter
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
        for(i=100;i>=0;i--){  //make the led darker
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
    }
}
```

First, create a software PWM pin.

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

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

```
    while(1){
        for(i=0;i<100;i++){  //make the led brighter
            softPwmWrite(ledPin, i);
            delay(20);
        }
```

```
        delay(300);
        for(i=100;i>=0;i--){  //make the led darker
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
    }
```

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

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

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

Python Code 4.1 BreathingLED

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

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

1. Use cd command to enter 4_BreathingLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/4_BreathingLED
```

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

```
python BreathingLED.py
```

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

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LedPin = 11      # define the LedPin

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

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

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

def destroy():
    p.stop() # stop PWM
    GPIO.cleanup() # Release all GPIO

if __name__ == '__main__':     # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
```

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

The LED is connected to the IO port called GPIO17. The LedPin is defined as pin 11 and set to output mode according to the corresponding chart for pin designations. Then create a PWM instance and set the PWM frequency to 500HZ and the initial duty cycle to 0%.

```
LedPin = 11      # define the LedPin

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

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

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

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

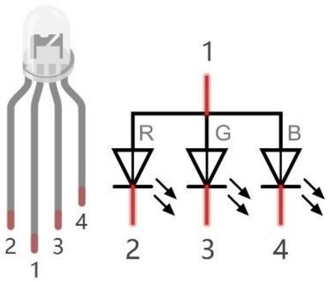
The related functions of PWM are described as follows:

p = GPIO.PWM(channel, frequency)
To create a PWM instance:
p.start(dc)
To start PWM, where dc is the duty cycle (0.0 <= dc <= 100.0)
p.ChangeFrequency(freq)
To change the frequency, where freq is the new frequency in Hz
p.ChangeDutyCycle(dc)
To change the duty cyclewhere 0.0 <= dc <= 100.0
p.stop()
To stop PWM.

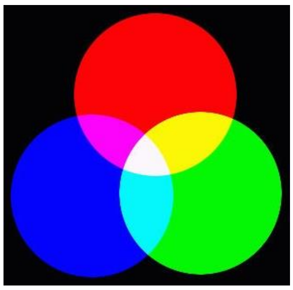
For more details regarding methods for using PWM with RPi.GPIO, please refer to:
https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/

Chapter 5 RGB LED

In this chapter, we will learn how to control 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.



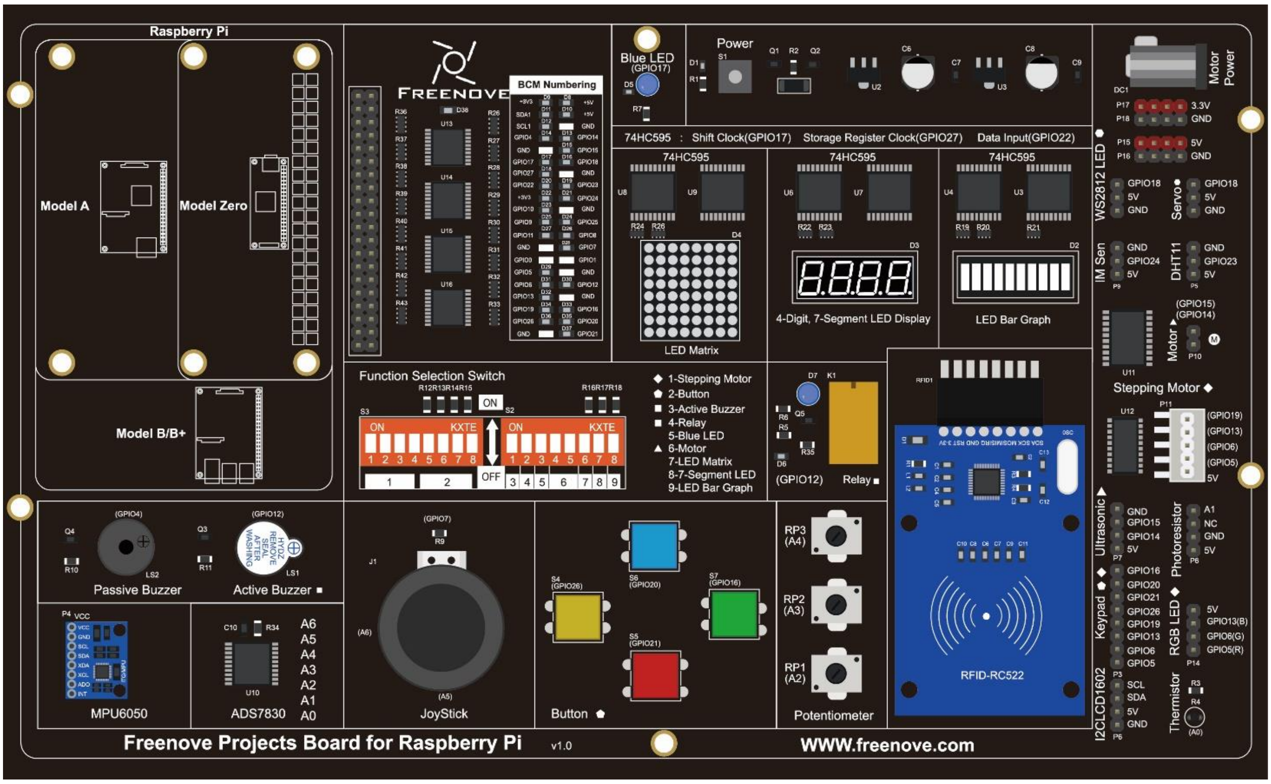
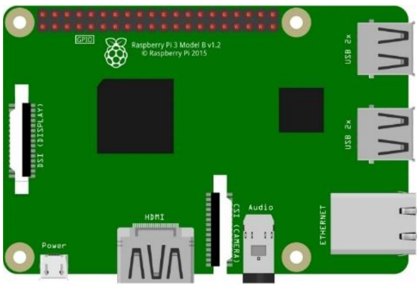
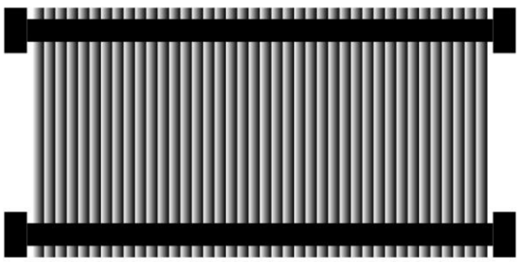
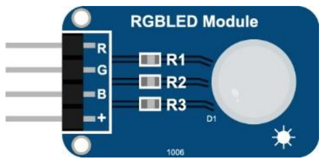
RGB

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

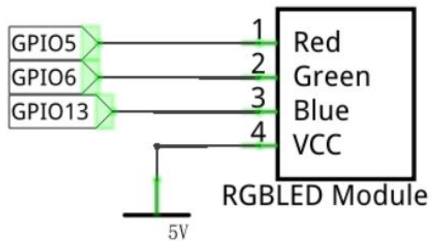
In this project, we will make a multicolored LED, which we can program the RGB LED to automatically change colors.

Component List

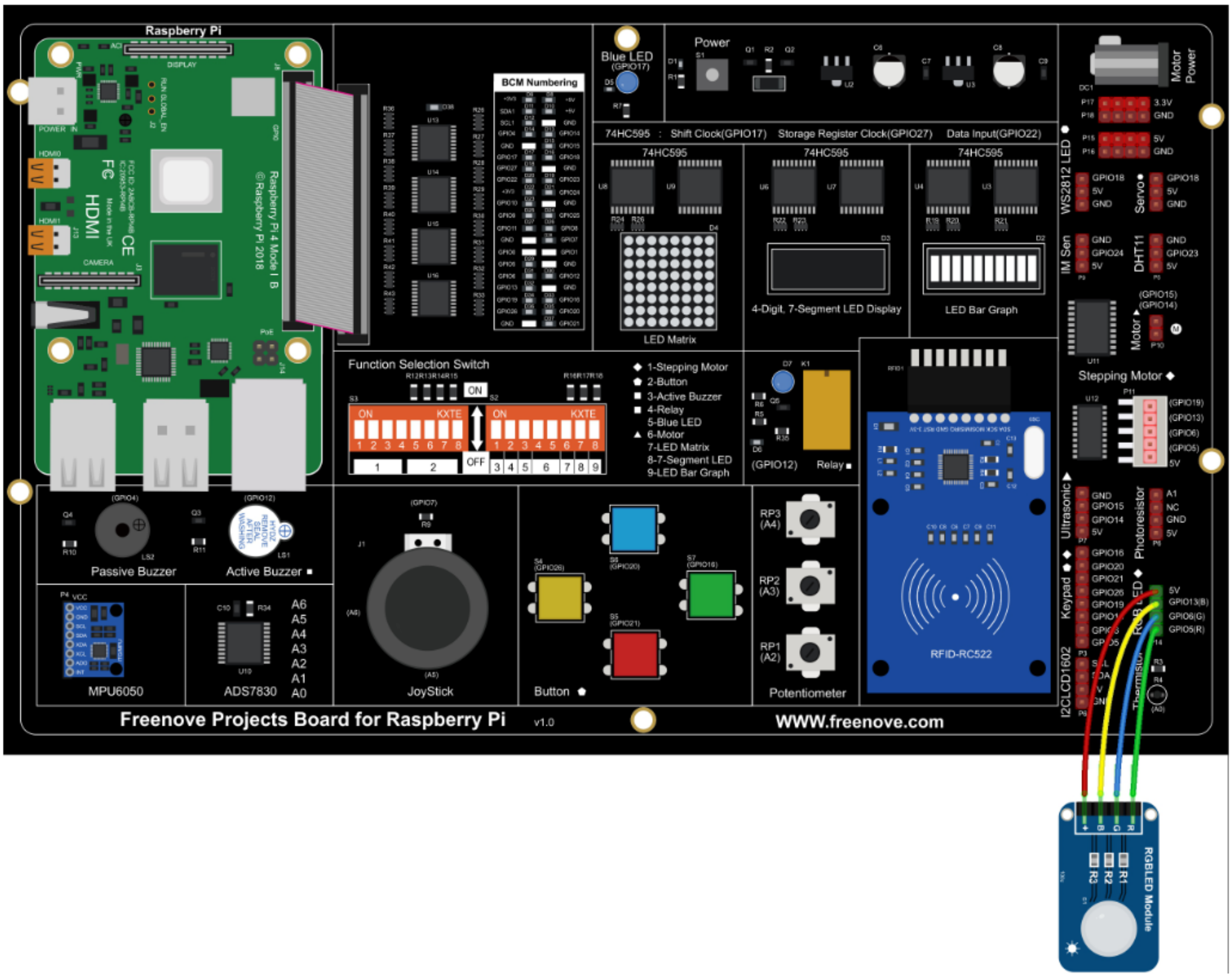
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>RGBLED Module

</td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



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

Code

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

C Code 5.1 RainbowLED

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

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

1. Use cd command to enter 5_ RainbowLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/5_RainbowLED
```

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

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

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

```
sudo ./RainbowLED
```

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

The following is the program code:

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

#define ledPinRed    21
#define ledPinGreen  22
#define ledPinBlue   23

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

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

int main(void)
```

```
{
    int r,g,b;

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

    wiringPiSetup();  //Initialize wiringPi.

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

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

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

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

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

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

```
    while(1){
        r=random()%100;  //get a random in (0,100)
        g=random()%100;  //get a random in (0,100)
        b=random()%100;  //get a random in (0,100)
        setLedColor(r,g,b);//set random as the duty cycle value
```

```
        printf("r=%d,  g=%d,  b=%d \n",r,g,b);
        delay(1000);
    }
```

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

long random();
This function will return a random number.

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

Python Code 5.1 RainbowLED

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

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

1. Use cd command to enter 5_RainbowLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/5_RainbowLED
```

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

```
python RainbowLED.py
```

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

The following is the program code:

```python
import RPi.GPIO as GPIO
import time
import random

pins = [29, 31, 33]         # define the pins for R:29,G:31,B:33

def setup():
    global pwmRed,pwmGreen,pwmBlue
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(pins, GPIO.OUT)     # set RGBLED pins to OUTPUT mode
    GPIO.output(pins, GPIO.HIGH)   # make RGBLED pins output HIGH level
    pwmRed = GPIO.PWM(pins[0], 2000)       # set PWM Frequence to 2kHz
    pwmGreen = GPIO.PWM(pins[1], 2000)  # set PWM Frequence to 2kHz
    pwmBlue = GPIO.PWM(pins[2], 2000)     # set PWM Frequence to 2kHz
    pwmRed.start(0)       # set initial Duty Cycle to 0
    pwmGreen.start(0)
    pwmBlue.start(0)

def setColor(r_val,g_val,b_val):      # change duty cycle for three pins to r_val,g_val,b_val
    pwmRed.ChangeDutyCycle(r_val)      # change pwmRed duty cycle to r_val
    pwmGreen.ChangeDutyCycle(g_val)
    pwmBlue.ChangeDutyCycle(b_val)

def loop():
    while True :
        r=random.randint(0,100)  #get a random in (0,100)
        g=random.randint(0,100)
        b=random.randint(0,100)

        setColor(r,g,b)          #set random as a duty cycle value
        print ('r=%d, g=%d, b=%d ' %(r ,g, b))
        time.sleep(1)

```

```
def destroy():
    pwmRed.stop()
    pwmGreen.stop()
    pwmBlue.stop()
    GPIO.cleanup()

if __name__ == '__main__':     # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

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

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

About the randint() function :

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

Chapter 6 Buzzer

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

Project 6.1 Doorbell

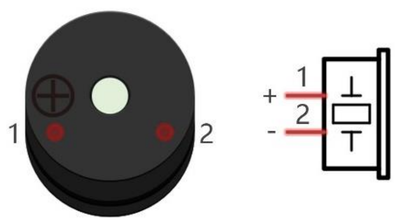
We will make a doorbell with this functionality: when the Push Button Switch is pressed the buzzer sounds and when the button is released, the buzzer stops. This is a momentary switch function.

Component knowledge

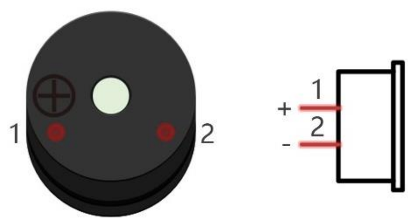
Buzzer

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

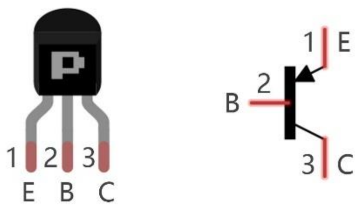
Passive buzzer bottom

Transistors

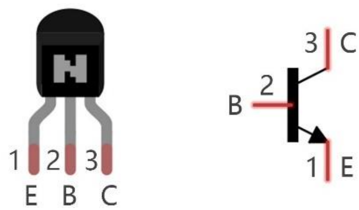
A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to amplify the current.

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

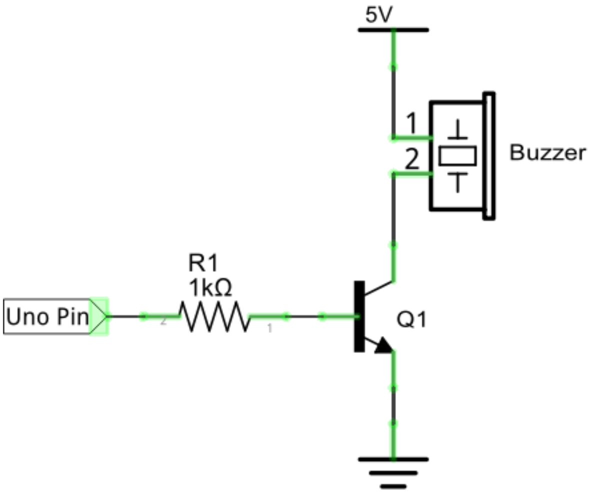
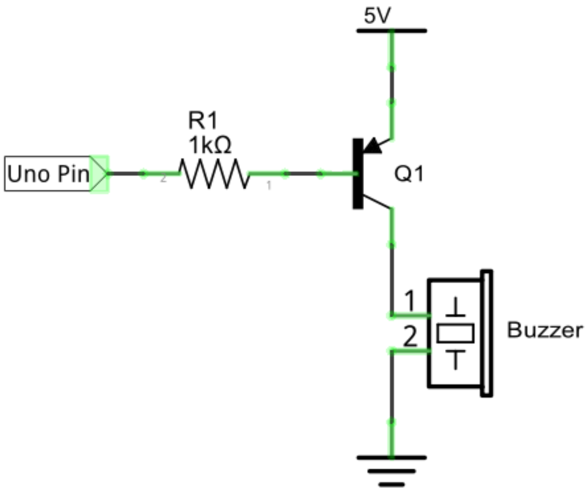


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

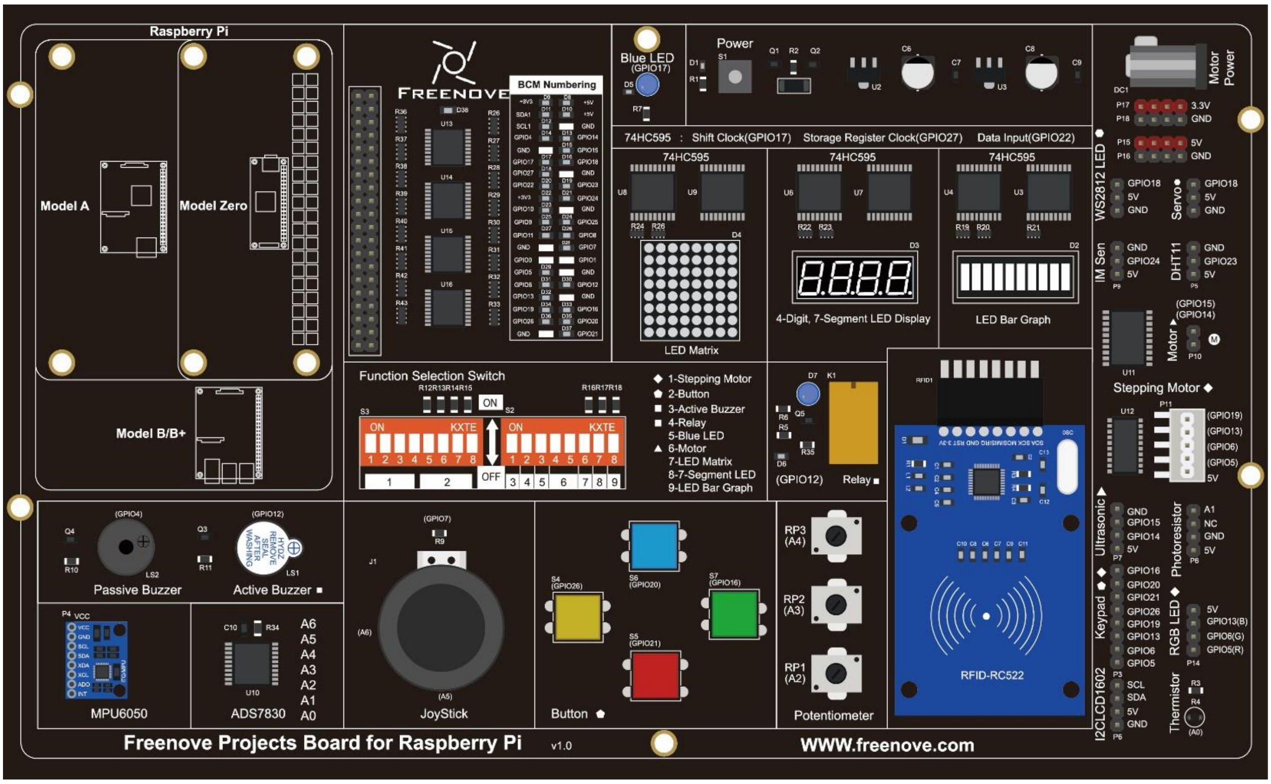
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not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

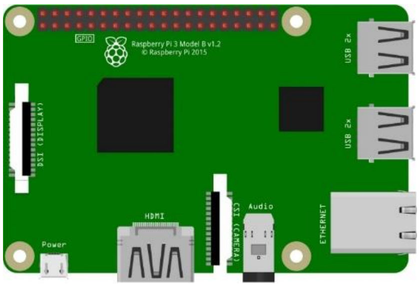
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

Component List

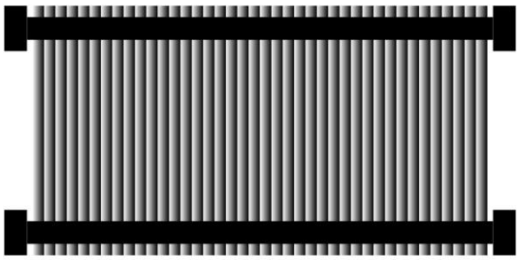
Freenove Projects Board for Raspberry Pi



Raspberry Pi

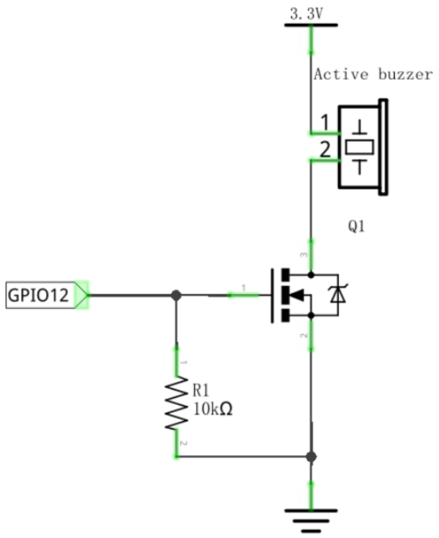


GPIO Ribbon Cable

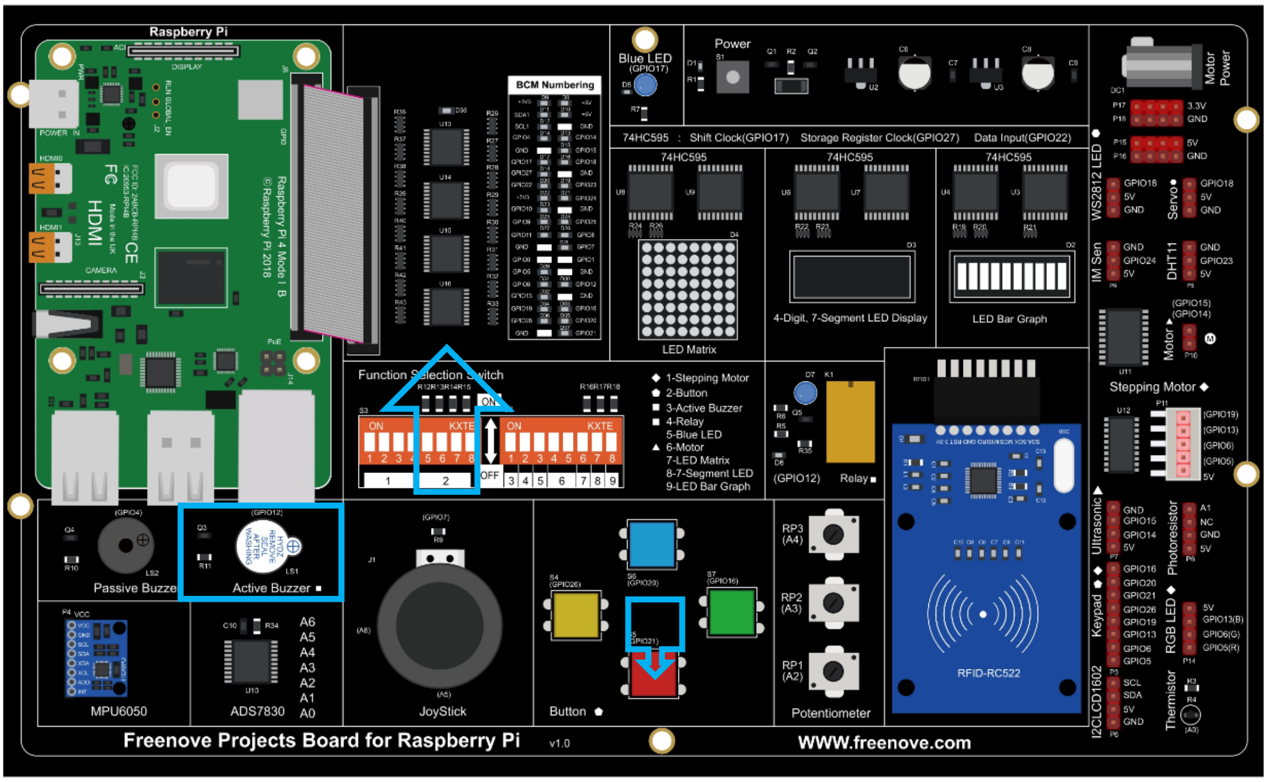


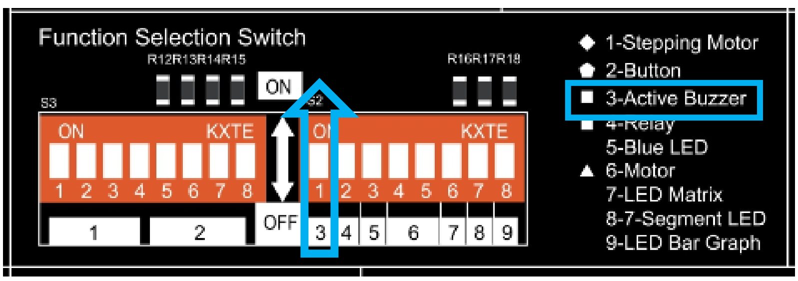
Circuit

Schematic diagram with RPi GPIO Extension Shield



Hardware connection.





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

Code

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

C Code 6.1 Doorbell

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

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

1. Use cd command to enter 6_1_Doorbell directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/6_1_Doorbell
```

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

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

3. Then run the generated file “Doorbell”.

```
sudo ./Doorbell
```

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

The following is the program code:

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

#define buzzerPin 26  //define the buzzerPin
#define buttonPin 29      //define the buttonPin

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

    wiringPiSetup();

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

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

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

Python Code 6.1 Doorbell

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

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

1. Use cd command to enter 6_1_Doorbell directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/6_1_Doorbell
```

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

```
python Doorbell.py
```

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

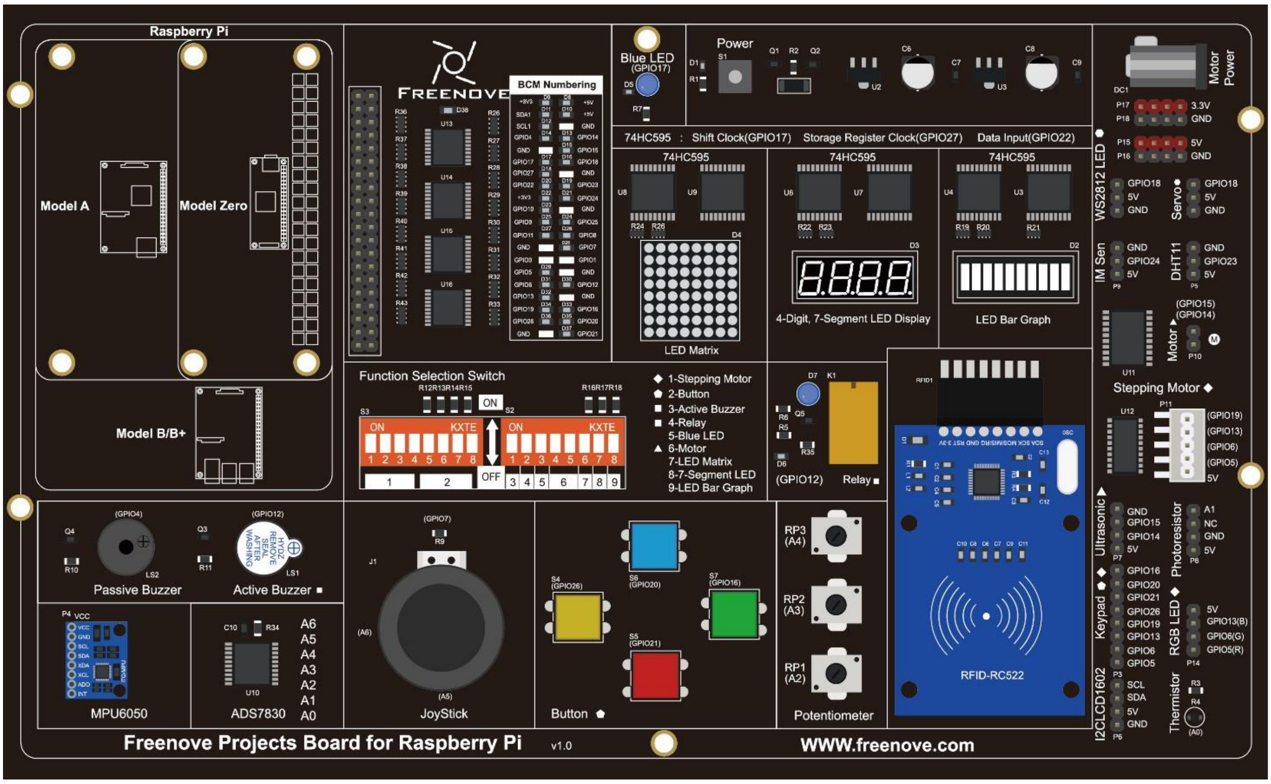
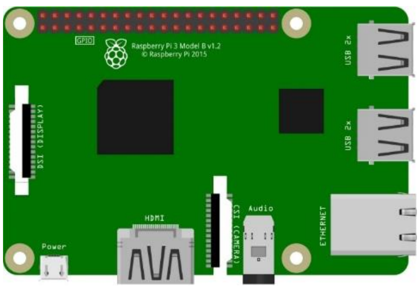
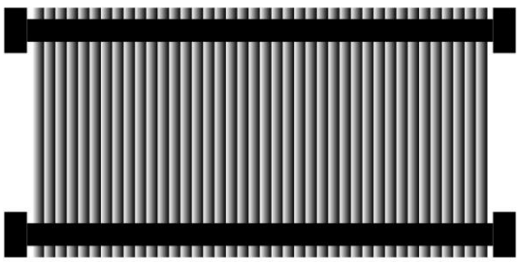
The following is the program code:

```python
import RPi.GPIO as GPIO

buzzerPin = 32    # define buzzerPin
buttonPin = 40    # define buttonPin

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

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
            GPIO.output(buzzerPin,GPIO.HIGH) # turn on buzzer
            print ('buzzer turned on >>>')
        else : # if button is relessed
            GPIO.output(buzzerPin,GPIO.LOW) # turn off buzzer
            print ('buzzer turned off <<<')

def destroy():
    GPIO.cleanup()                     # Release all GPIO

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

Project 6.2 Alertor

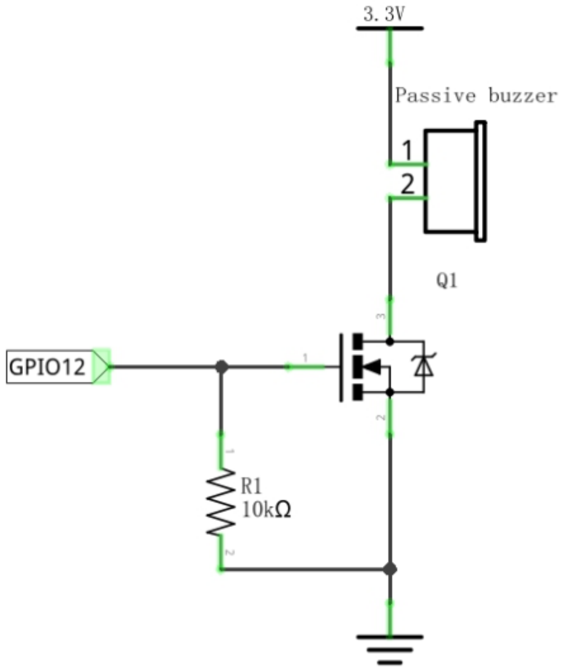
Next, we will use a passive buzzer to make an alarm.

Component List

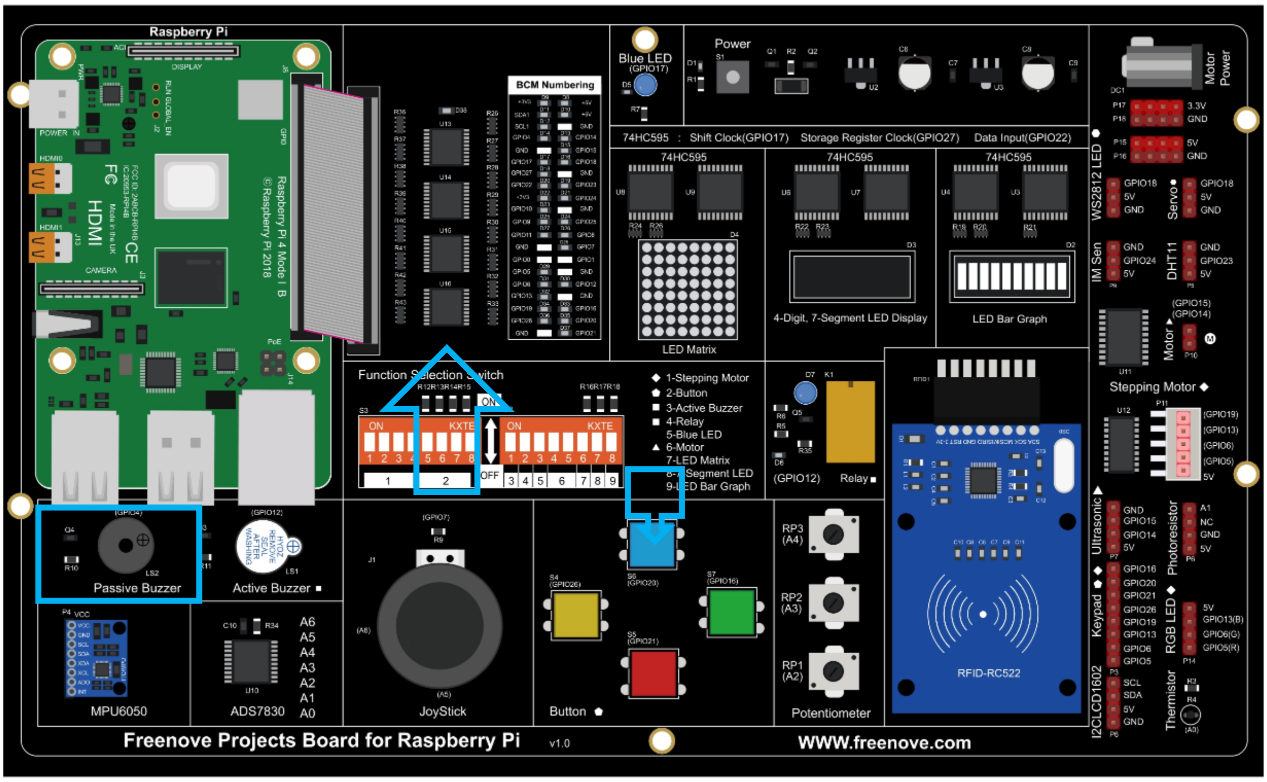
Freenove Projects Board for Raspberry Pi	
	
Raspberry Pi 	GPIO Ribbon Cable 

Circuit

Schematic diagram with RPi GPIO Extension Shield



Hardware connection.



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

Code

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

C Code 6.2 Alertor

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

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

1. Use cd command to enter 6_2_Alertor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/6_2_Alertor
```

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

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

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

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

The following is the program code:

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

#define buzzerPin    7     //define the buzzerPin
#define buttonPin      28      //define the buttonPin

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

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

    wiringPiSetup();

    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    softToneCreate(buzzerPin); //set buzzerPin
    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){
        if(digitalRead(buttonPin) == LOW){ //button is pressed
            alertor(buzzerPin);   // turn on buzzer
            printf("alertor turned on >>> \n");
        }
        else {                //button is released
            stopAlertor(buzzerPin);   // turn off buzzer
            printf("alertor turned off <<< \n");
        }
    }
    return 0;
}
```

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

```
softToneCreate(buzzeRPin);
```

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

```
void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ //The frequency is based on the sine curve.
        sinVal = sin(x * (M_PI / 180));
        toneVal = 2000 + sinVal * 500;
        softToneWrite(pin,toneVal);
        delay(1);
    }
```

```
}
```

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

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

The related functions of softTone are described as follows:

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

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

Python Code 6.2 Alertor

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

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

1. Use cd command to enter 6_2_Alertor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/6_2_Alertor
```

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

```
python Alertor.py
```

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

The following is the program code:

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

buzzerPin = 7    # define the buzzerPin
buttonPin = 38    # define the buttonPin

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

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            alertor()
            print ('alertor turned on >>> ')
        else :
            stopAlertor()
            print ('alertor turned off <<<')
def alertor():
    p.start(50)
    for x in range(0,361):      # Make frequency of the alertor consistent with the sine wave
        sinVal = math.sin(x * (math.pi / 180.0))        # calculate the sine value
        toneVal = 2000 + sinVal * 500   # Add to the resonant frequency with a Weighted
        p.ChangeFrequency(toneVal)      # Change Frequency of PWM to toneVal
        time.sleep(0.001)
```

```

def stopAlertor():
    p.stop()

def destroy():
    GPIO.output(buzzerPin, GPIO.LOW)     # Turn off buzzer
    GPIO.cleanup()                       # Release GPIO resource

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

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

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

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

```
def alertor():
    p.start(50)
    for x in range(0,361):      # Make frequency of the alertor consistent with the sine wave
        sinVal = math.sin(x * (math.pi / 180.0))        # calculate the sine value
        toneVal = 2000 + sinVal * 500   # Add to the resonant frequency with a Weighted
        p.ChangeFrequency(toneVal)      # Change Frequency of PWM to toneVal
        time.sleep(0.001)
```

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

```
def stopAlertor():
    p.stop()
```

(**Important**) Chapter 7 ADC

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

Project 7.1 Read the Voltage of Potentiometer

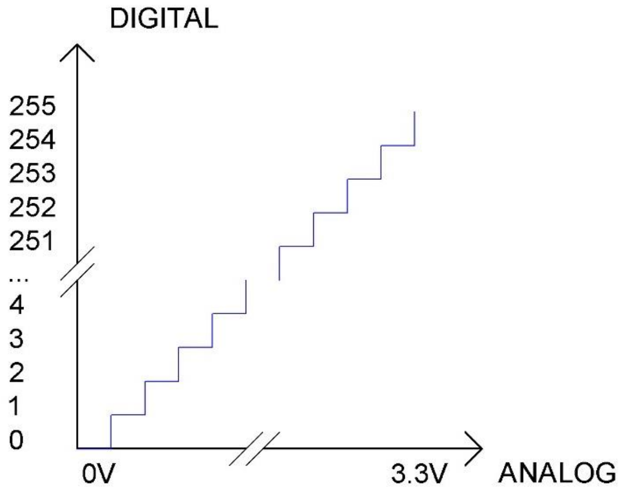
In this project, we will use the ADC function of an ADC Module to read the voltage value of a potentiometer.

Circuit knowledge

ADC

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

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



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

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

The resultant analog signal will be divided accordingly.

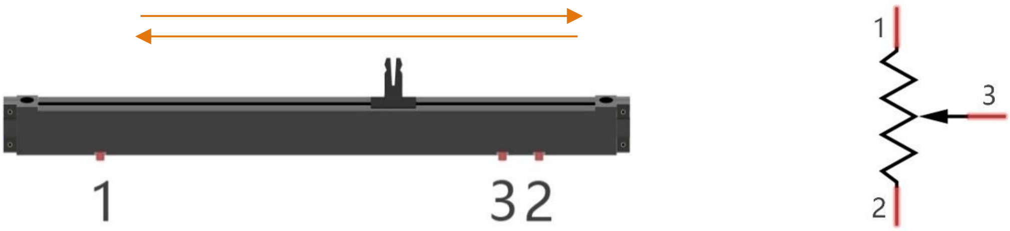
DAC

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

Component knowledge

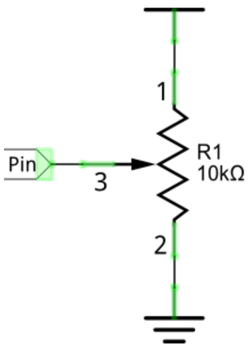
Potentiometer

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



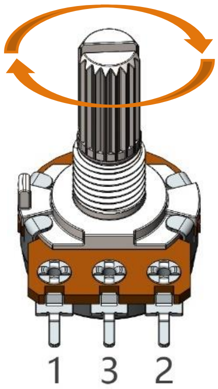
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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

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



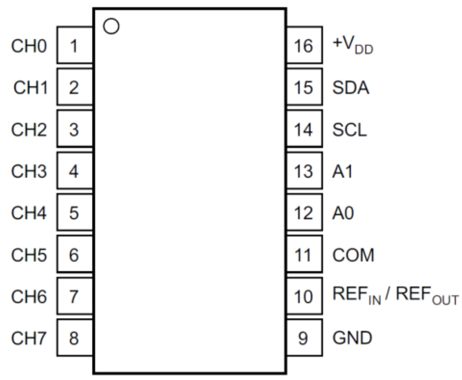
Rotary potentiometer

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



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.

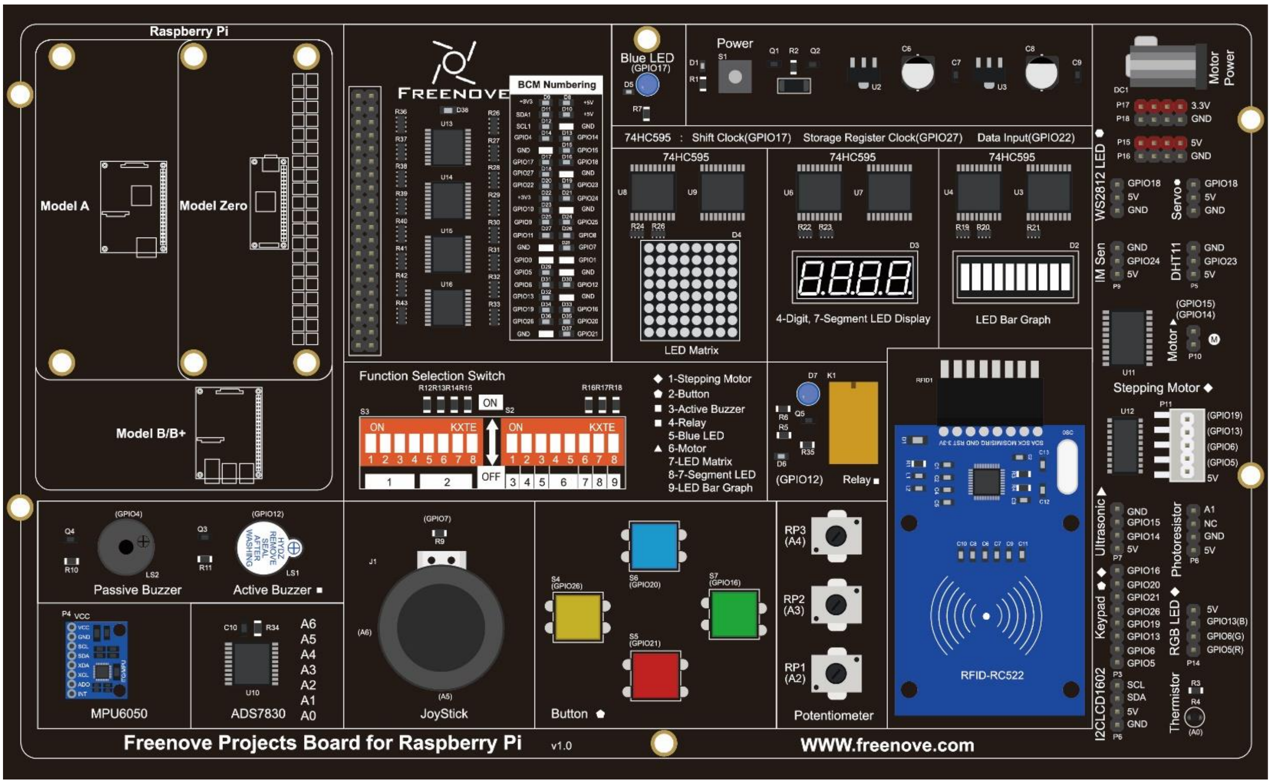
SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		
CH5	6		
CH6	7		
CH7	8		
GND	9	Ground	
REF in/out	10	Internal +2.5V Reference, External Reference Input	
COM	11	Common to Analog Input Channel	
A0	12	Hardware address	
A1	13		
SCL	14	Serial Clock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

I2C communication

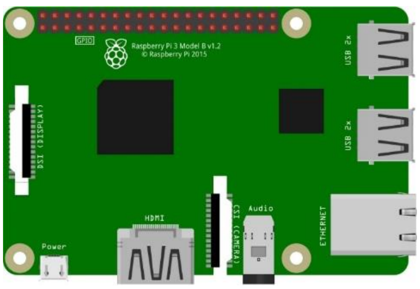
I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique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

Component List

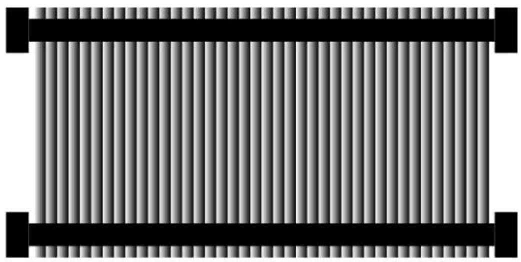
Freenove Projects Board for Raspberry Pi



Raspberry Pi

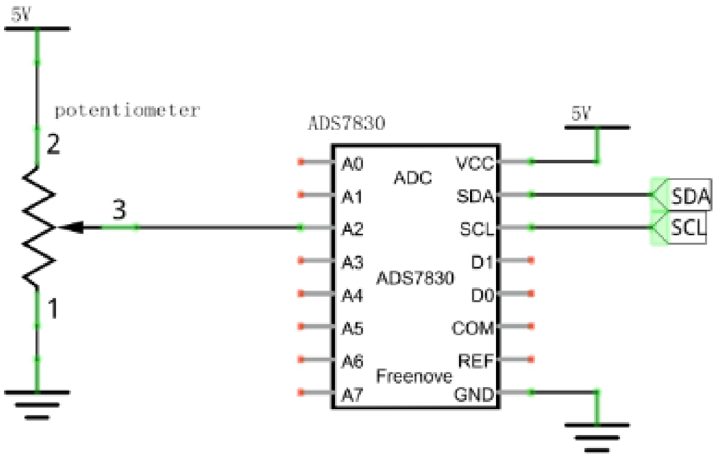


GPIO Ribbon Cable

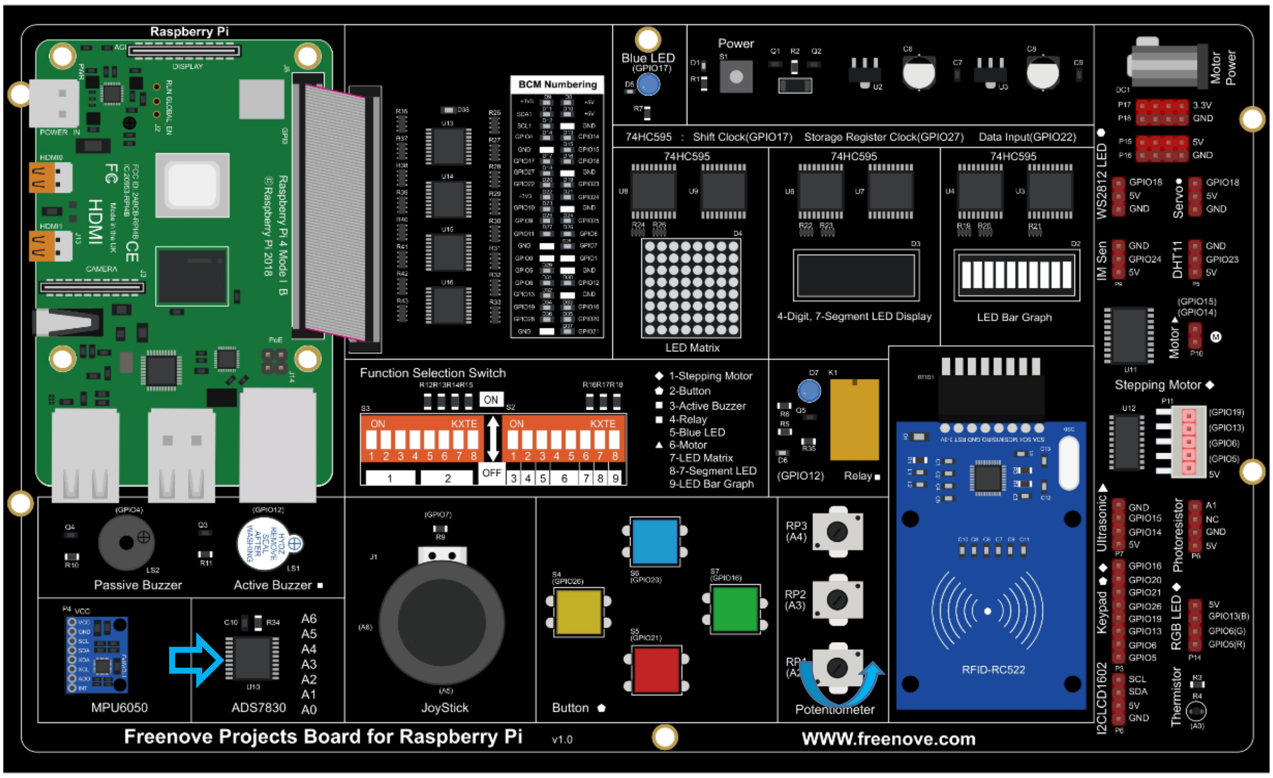


Circuit

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:

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

1 Change User Password          Change password for the current u
2 Network Options               Configure network settings
3 Boot Options                  Configure options for start-up
4 Localisation Options          Set up language and regional sett
5 Interfacing Options           Configure connections to peripher
6 Overclock                     Configure overclocking for your P
7 Advanced Options              Configure advanced settings
8 Update                        Update this tool to the latest ve
9 About raspi-config            Information about this configurat

                 <Select>                     <Finish>
```

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

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

```
lsmod | grep i2c
```

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

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

Install I2C-Tools

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

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

I2C device address detection:

```
i2cdetect -y 1
```

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

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

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

Install Smbus Module

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

Code

C Code 7.1 ADC

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

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

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

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

2. Execute command below to install the library.

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

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

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

Next, we will execute the code for this project.
First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 7_1_ADC directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/7_1_ADC
```

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

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

3. Then run the generated file "ADC".

```
sudo ./ADC
```

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

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

The following is the code:

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

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

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

```
 
    while(1){
        int adcValue = adc->analogRead(2);    //read analog value of A0 pin
        float voltage = (float)adcValue / 255.0 * 5.0;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
        delay(100);
    }
    return 0;
}
```

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

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

Then use the member function detectIC(addr) in the class to detect the I2C module in the circuit. Different modules have different I2C addresses. The default address of ADC module ADS7830 is 0x48.

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

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

```
int adcValue = adc->analogRead(2);    //read analog value of A2 pin
```

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

```
        float voltage = (float)adcValue / 255.0 * 5.0;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
```

Reference

```
class ADCDevice
```

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

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

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

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

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

```
class ADS7830:public ADCDevice
```

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

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

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

Python Code 7.1 ADC

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

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

1. Use cd command to enter folder of ADCDevice.

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

2. Unzip the file.

```
tar zxvf ADCDevice-1.0.4.tar.gz
```

3. Open the unzipped folder.

```
cd ADCDevice-1.0.4
```

4. Install library for python2 and python3.

```
sudo python2 setup.py install
sudo python3 setup.py install
```

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

```
Installed /usr/local/lib/python3.7/dist-packages/ADCDevice-1.0.4-py3.7.egg
Processing dependencies for ADCDevice==1.0.4
Finished processing dependencies for ADCDevice==1.0.4
```

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

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

1. Use cd command to enter 7_1_ADC directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/7_1_ADC
```

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

```
sudo python ADC.py
```

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

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

The following is the code:

```
import time
from ADCDevice import *

adc = ADCDevice(0x48) # Define an ADCDevice class object

```

```
def setup():
    global adc
    if(adc.detectI2C(0x48)):
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)

def loop():
    while True:
        value = adc.analogRead(2)    # read the ADC value of channel 2
        voltage = value / 255.0 * 5.0  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.1)

def destroy():
    adc.close()

if __name__ == '__main__':   # Program entrance
    print ('Program is starting ... ')
    try:
        setup()
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

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

```
adc = ADCDevice(0x48) # Define an ADCDevice class object
```

Then in setup(), use detecticIC(addr), the member function of ADCDevice, to detect the I2C module in the circuit. The default address of ADS7830 is 0x48.

```
def setup():
    global adc
    if(adc.detectI2C(0x48)):
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
```

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

```
value = adc.analogRead(2)    # read the ADC value of channel 2
```

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

```
voltage = value / 255.0 * 5.0  # calculate the voltage value
print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
time.sleep(0.1)
```

Reference

About smbus Module:

smbus Module

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

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

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

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

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

`class ADCDevice(object)`

This is a base class.

`int detectI2C(int addr);`

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

`class ADS7830(ADCDevice)`

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

`int analogRead(int chn);`

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

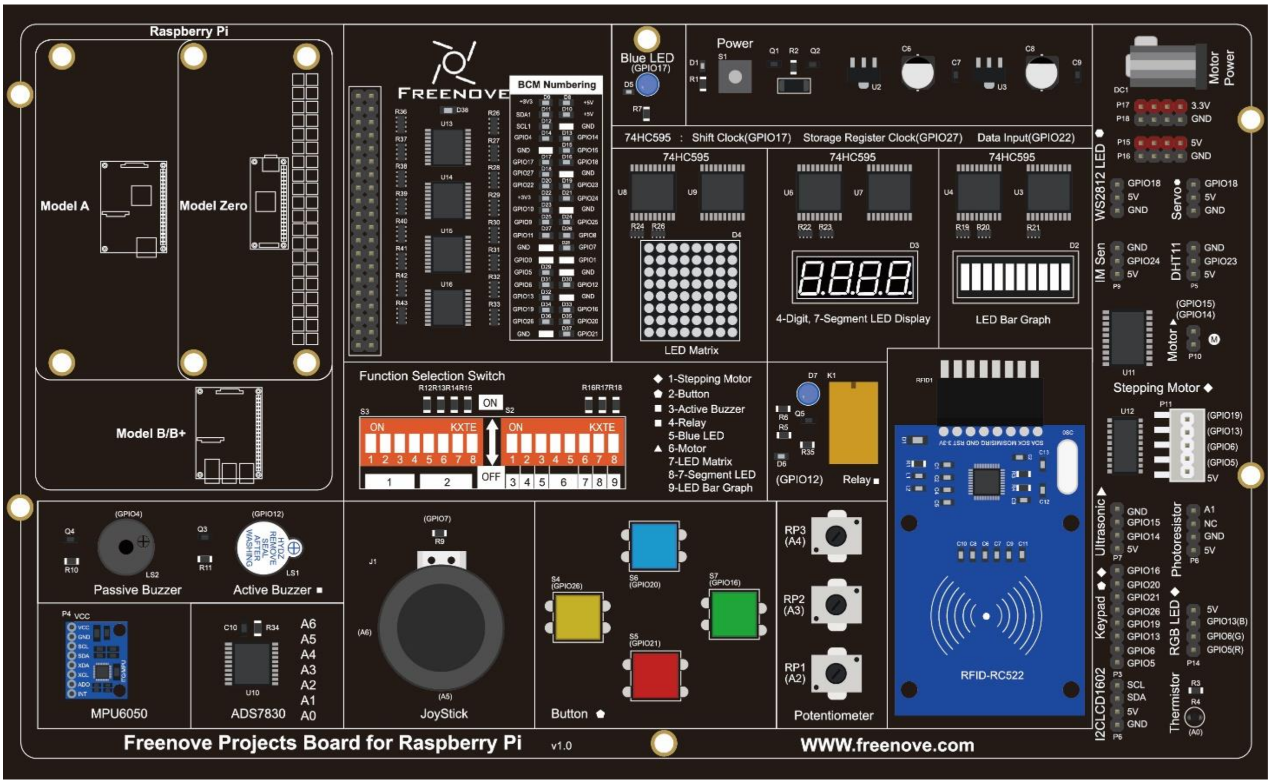
Parameter chn: For ADS7830, the range is 0, 1, 2, 3, 4, 5, 6, 7.

Project 7.2 Soft Light

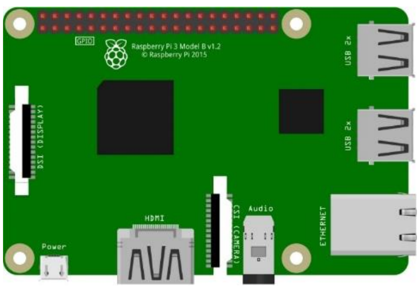
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle ratio of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

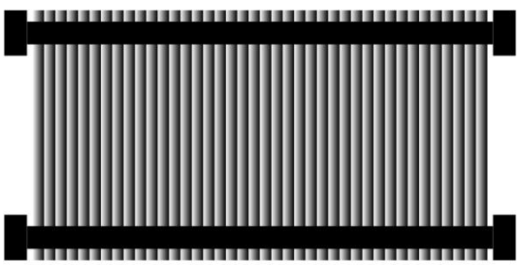
Freenove Projects Board for Raspberry Pi



Raspberry Pi

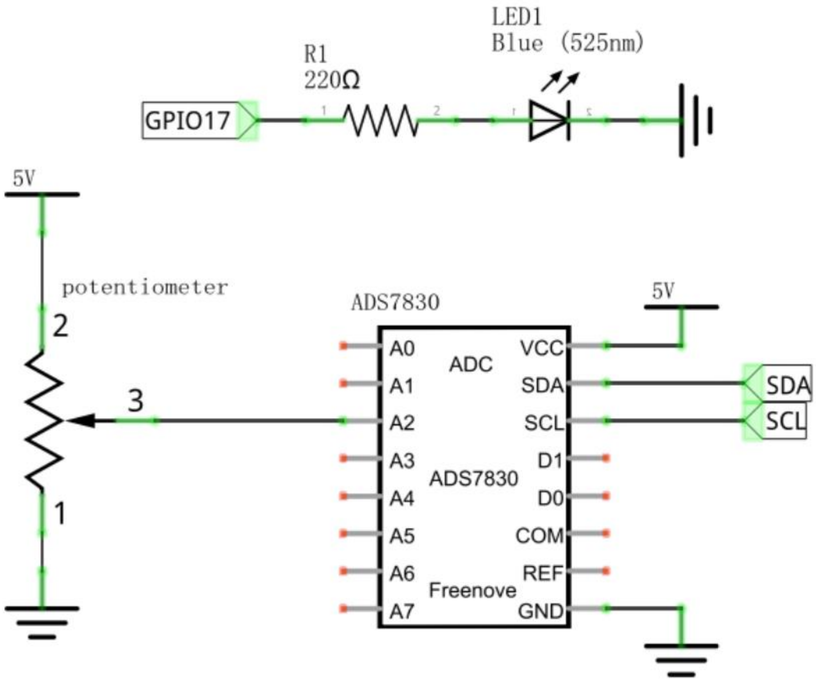


GPIO Ribbon Cable

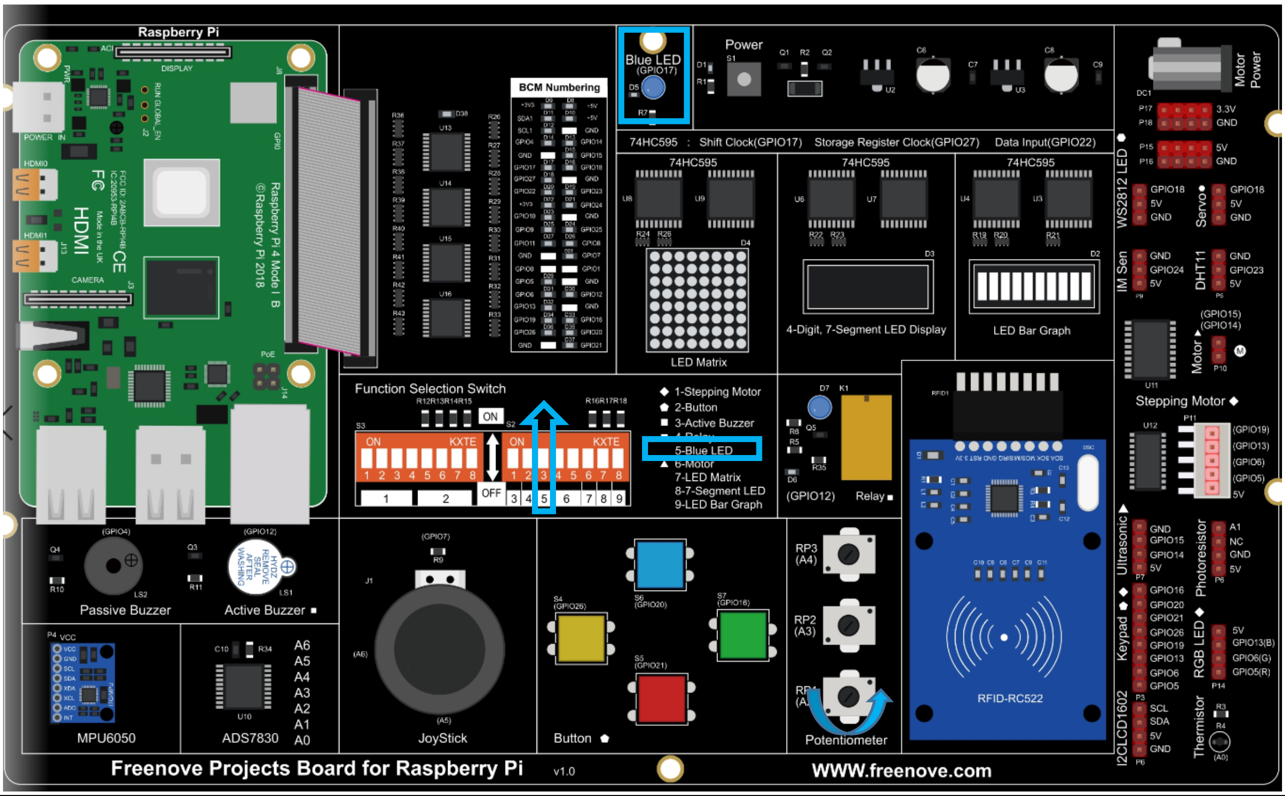


Circuit

Schematic diagram



Hardware connection



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

Code

C Code 7.2 Softlight

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please move on.
First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 7_2_Softlight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/7_2_Softlight
```

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

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

3. Then run the generated file "Softlight".

```
sudo ./Softlight
```

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

The following is the code:

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

#define ledPin 0

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

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

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

```
    softPwmCreate(ledPin,0,100);
    while(1){
        int adcValue = adc->analogRead(2);    //read analog value of A2 pin
        softPwmWrite(ledPin,adcValue*100/255);    // Mapping to PWM duty cycle
        float voltage = (float)adcValue / 255.0 * 5.0;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
        delay(30);
    }
    return 0;
}
```

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

```
        int adcValue = adc->analogRead(2);    //read analog value of A2 pin
        softPwmWrite(ledPin,adcValue*100/255);    // Mapping to PWM duty cycle
```

Python Code 7.2 Softlight

If you haven't configured I2C, please refer to Chapter 7. If you did, please continue.

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

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

1. Use cd command to enter 7_2_Softlight directory of Python code

```
cd ~/Freenove_Kit/Code/Python_Code/7_2_Softlight
```

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

```
sudo python Softlight.py
```

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

The following is the code:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

ledPin = 11
adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)):
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)
    p = GPIO.PWM(ledPin,1000)
    p.start(0)

def loop():
    while True:
        value = adc.analogRead(2)    # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)        # Mapping to PWM duty cycle
        voltage = value / 255.0 * 5.0  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.03)
```

```

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

if __name__ == '__main__':   # Program entrance
    print ('Program is starting ... ')
    try:
        setup()
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

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

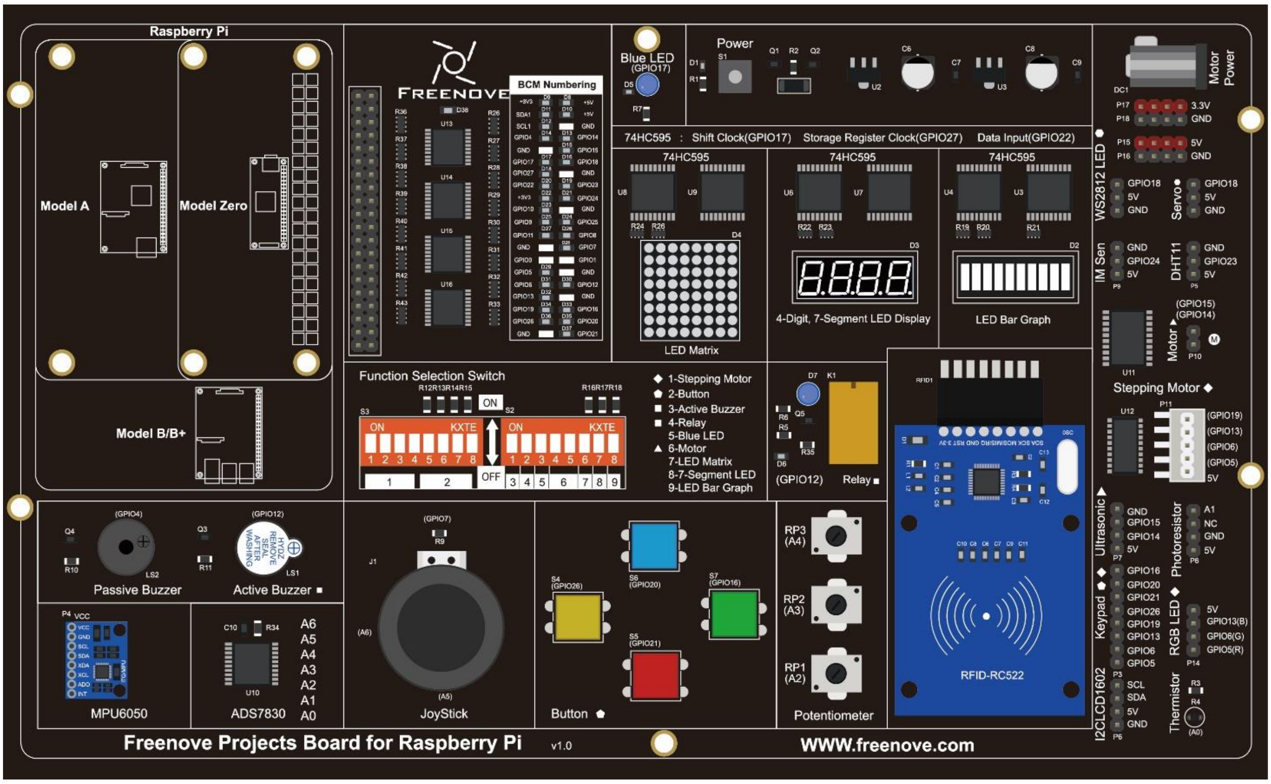
```
        value = adc.analogRead(2)     # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)        # Mapping to PWM duty cycle
```

Project 7.3 Colorful Light

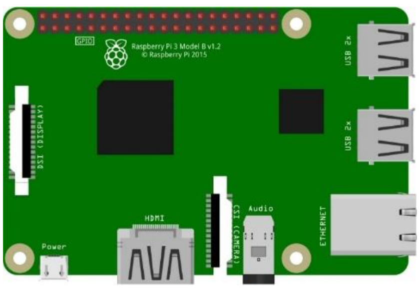
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as with the Soft Light. project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the previous soft light project needed only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

Component List

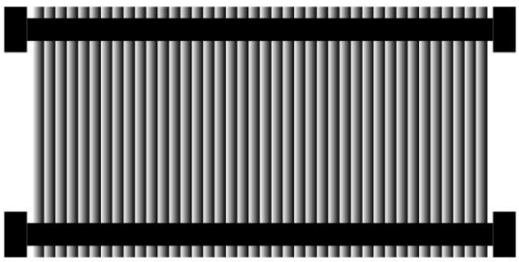
Freenove Projects Board for Raspberry Pi



Raspberry Pi

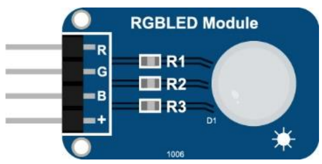


GPIO Ribbon Cable



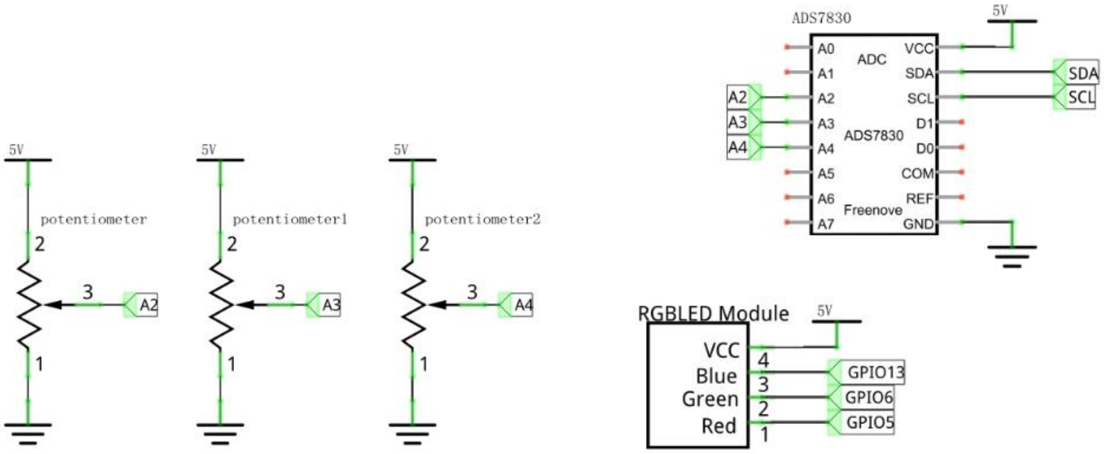
Jumper Wire

RGBLED Module

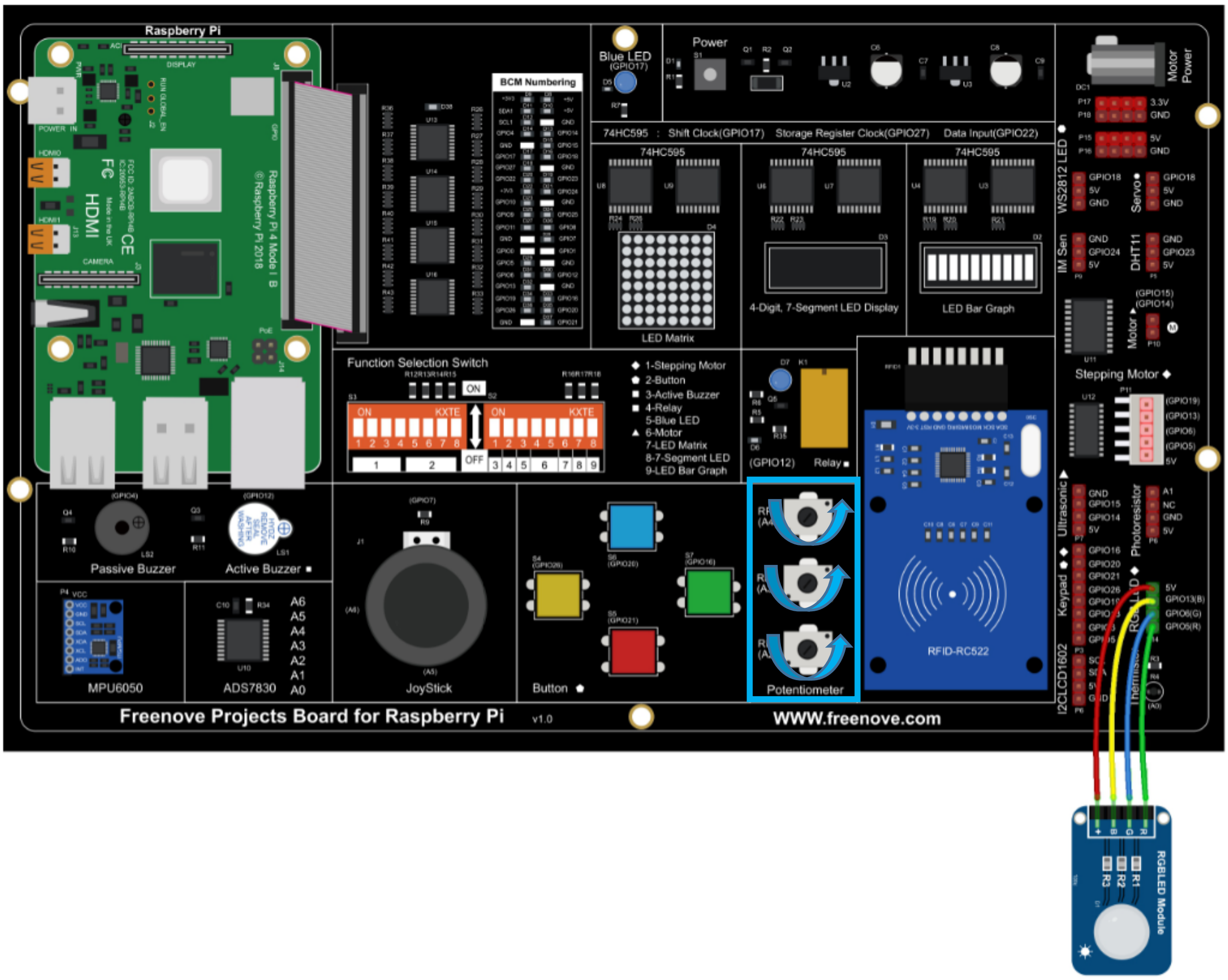


Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 7.3 Colorful Softlight

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 7_3_ColorfulSoftlight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/7_3_ColorfulSoftlight
```

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

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

3. Then run the generated file "ColorfulSoftlight".

```
sudo ./ColorfulSoftlight
```

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

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

The following is the program code:

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

#define ledRedPin 21          //define 3 pins for RGBLED
#define ledGreenPin 22
#define ledBluePin 23

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

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

    if(adc->detectI2C(0x48)){      // Detect the ads7830
        delete adc;                // Free previously pointed memory
```

```
        adc = new ADS7830(0x48);      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    softPwmCreate(ledRedPin,0,100);     //creat 3 PMW output pins for RGBLED
    softPwmCreate(ledGreenPin,0,100);
    softPwmCreate(ledBluePin,0,100);
    while(1){
        int val_Red = adc->analogRead(2);  //read analog value of 3 potentiometers
        int val_Green = adc->analogRead(3);
        int val_Blue = adc->analogRead(4);
        softPwmWrite(ledRedPin,100-val_Red*100/255);    //map the read value of potentiometers into PWM value and output it
        softPwmWrite(ledGreenPin,100-val_Green*100/255);
        softPwmWrite(ledBluePin,100-val_Blue*100/255);

        //print out the read ADC value
        printf("ADC value val_Red: %d  ,\tval_Green: %d  ,\tval_Blue: %d \n",val_Red,val_Green,val_Blue);
        delay(100);
    }
    return 0;
}
```

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

Python Code 7.3 ColorfulSoftlight

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 7_3_ColorfulSoftlight directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/7_3_ColorfulSoftlight
```

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

```
sudo python ColorfulSoftlight.py
```

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

The following is the program code:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

ledRedPin = 29      # define 3 pins for RGBLED
ledGreenPin = 31
ledBluePin = 33
adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)

    global p_Red,p_Green,p_Blue
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledRedPin,GPIO.OUT)      # set RGBLED pins to OUTPUT mode
    GPIO.setup(ledGreenPin,GPIO.OUT)
    GPIO.setup(ledBluePin,GPIO.OUT)

    p_Red = GPIO.PWM(ledRedPin,1000)    # configure PMW for RGBLED pins, set PWM Frequence to 1kHz
    p_Red.start(0)
    p_Green = GPIO.PWM(ledGreenPin,1000)
    p_Green.start(0)
```

```
    p_Blue = GPIO.PWM(ledBluePin,1000)
    p_Blue.start(0)

def loop():
    while True:
        value_Red = adc.analogRead(4)       # read ADC value of 3 potentiometers
        value_Green = adc.analogRead(3)
        value_Blue = adc.analogRead(2)
        p_Red.ChangeDutyCycle(100-value_Red*100/255)  # map the read value of potentiometers
into PWM value and output it
        p_Green.ChangeDutyCycle(100-value_Green*100/255)
        p_Blue.ChangeDutyCycle(100-value_Blue*100/255)

        # print read ADC value
        print ('ADC Value
value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value_Blue))
        time.sleep(0.01)

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

if __name__ == '__main__': # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

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

Chapter 8 Photoresistor & LED

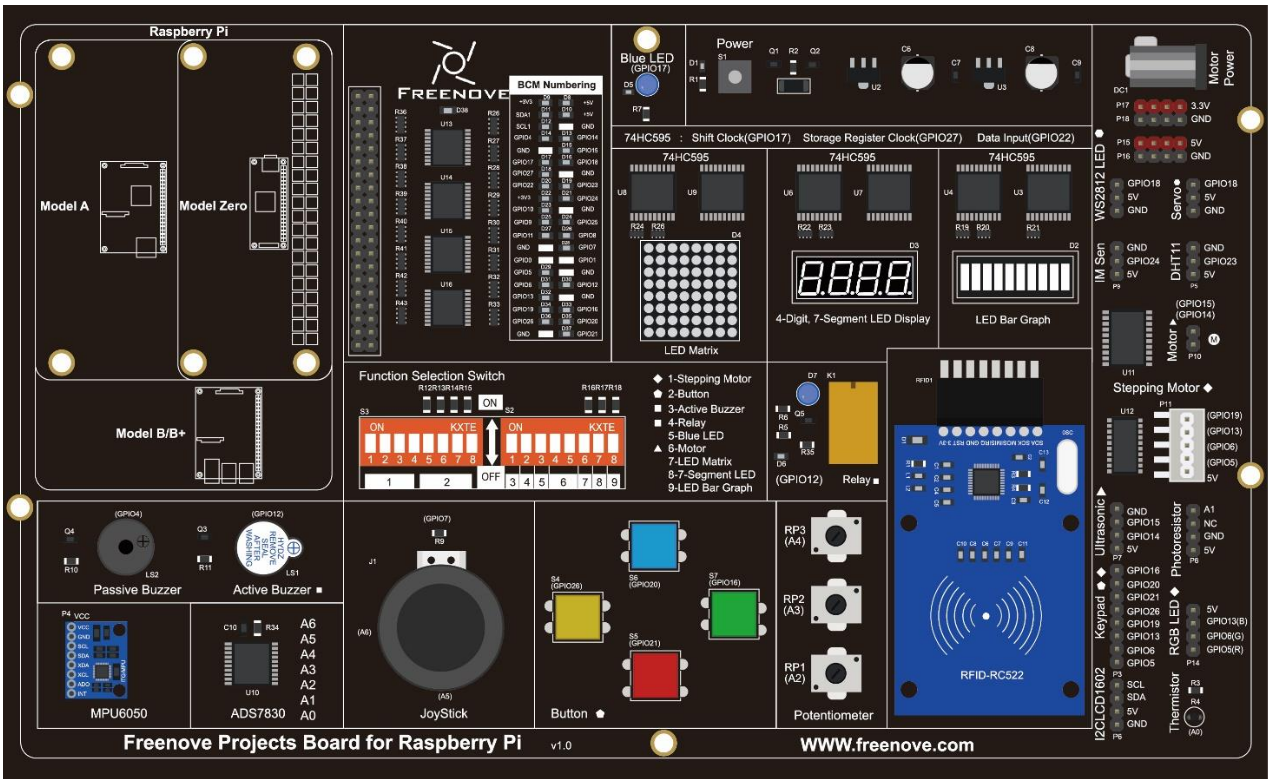
In this chapter, we will learn how to use a photoresistor to make an automatic dimming nightlight.

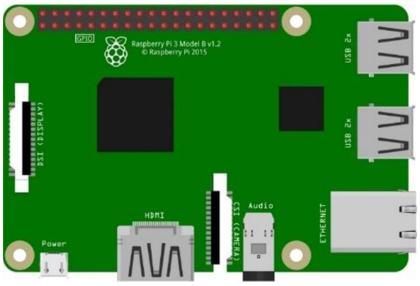
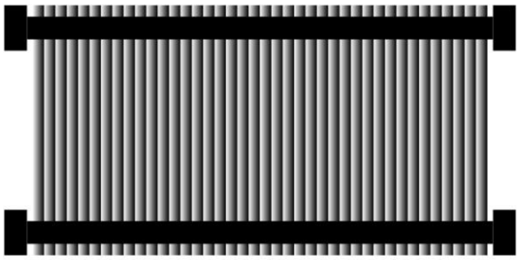
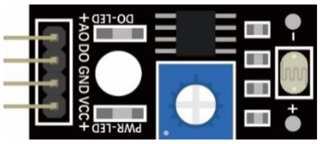
Project 8.1 NightLamp

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

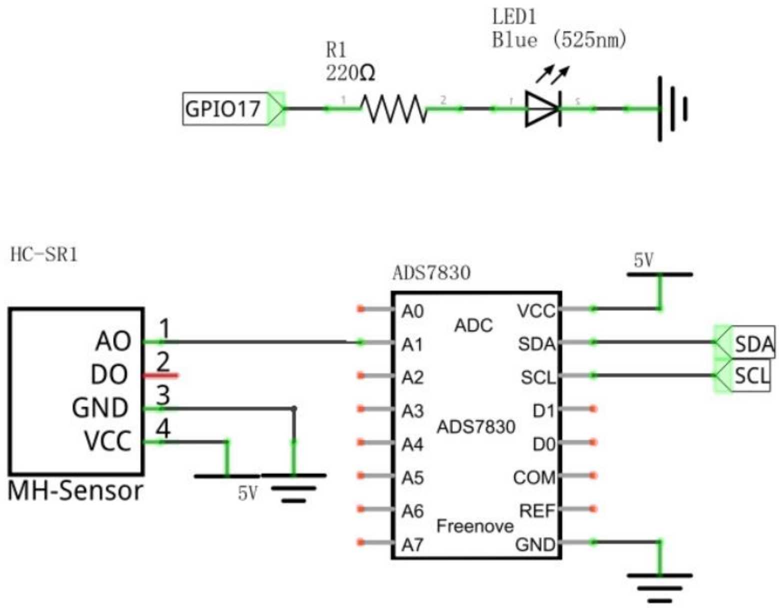
Freenove Projects Board for Raspberry Pi



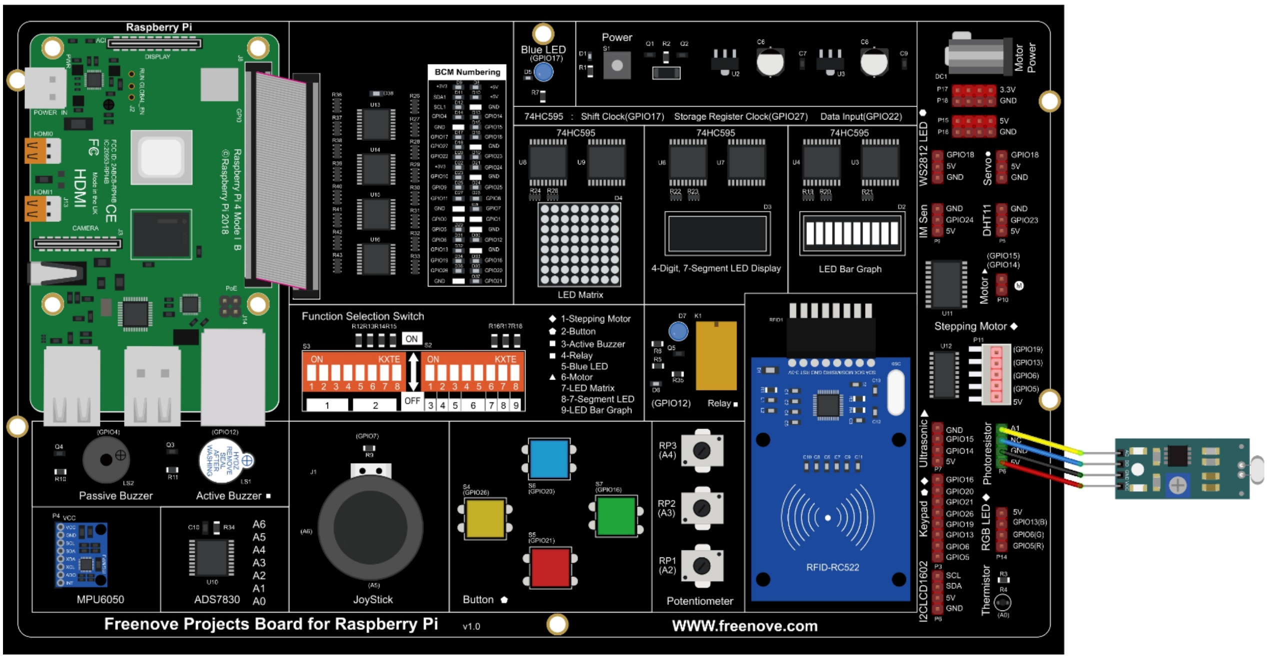
Raspberry Pi x1 	GPIO Ribbon Cable x1 
Jumper Wire 	Photoresistor 

Circuit

Schematic diagram



Hardware connection.



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

Code

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

C Code 8.1 Nightlamp

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 8_Nightlamp directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/8_Nightlamp
```

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

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

3. Then run the generated file "Nightlamp".

```
sudo ./Nightlamp
```

After the program is executed, if you cover the Photoresistor or increase the light shining on it, the brightness of the LED changes accordingly. As in previous projects the Terminal window will display the current input voltage value of ADC module A1 pin and the converted digital quantity.

The following is the program code:

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

#define ledPin 0

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

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

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

```
    wiringPiSetup();
    softPwmCreate(ledPin,0,100);
    while(1){
        int value = adc->analogRead(1);  //read analog value of A1 pin
        softPwmWrite(ledPin,value*100/255);
        float voltage = (float)value / 255.0 * 5.0;  // calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
    }
    return 0;
}
```

Python Code 8.1 Nightlamp

If you haven't configure I2C, please refer to Chapter 7. If you have done it, please continue.

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

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

1. Use cd command to enter 8_Nightlamp directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/8_Nightlamp
```

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

```
sudo python Nightlamp.py
```

After the program is executed, if you cover the Photoresistor or increase the light shining on it, the brightness of the LED changes accordingly. As in previous projects the Terminal window will display the current input voltage value of ADC module A1 pin and the converted digital quantity.

The following is the program code:

```python
import RPi.GPIO as GPIO
import time
from ADCDevice import *

ledPin = 11 # define ledPin
adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)   # set ledPin to OUTPUT mode
    GPIO.output(ledPin,GPIO.LOW)

    p = GPIO.PWM(ledPin,1000) # set PWM Frequence to 1kHz
    p.start(0)

def loop():
    while True:
        value = adc.analogRead(1)    # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 5.5
```

```
            print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
            time.sleep(0.01)

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

if __name__ == '__main__':   # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

Chapter 9 Thermistor

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

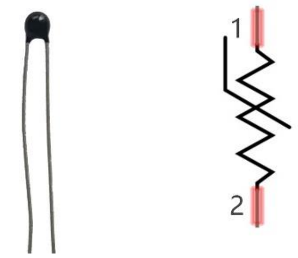
Project 9.1 Thermometer

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 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.



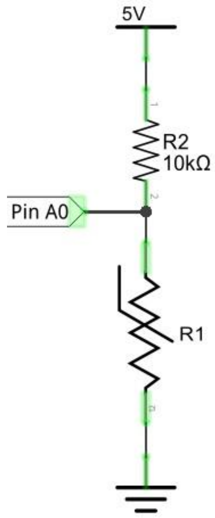
The relationship between resistance value and temperature of a thermistor is:

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:

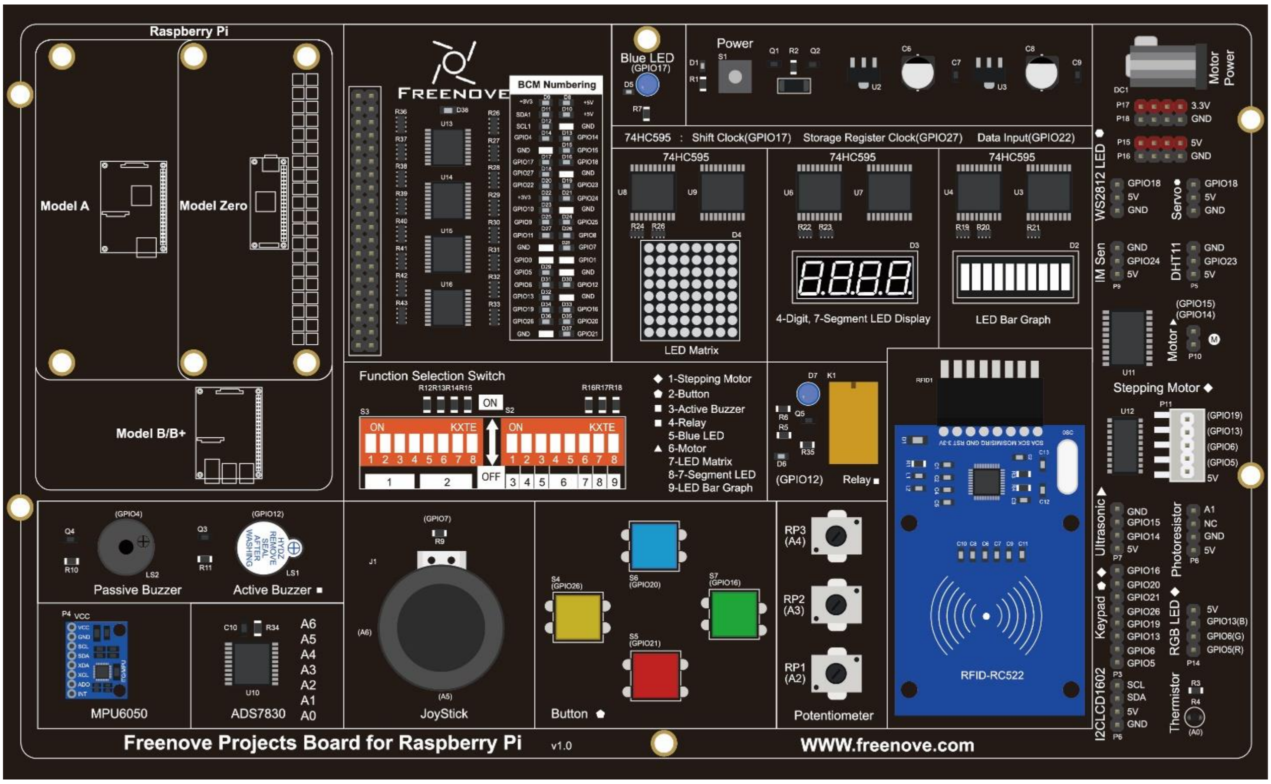
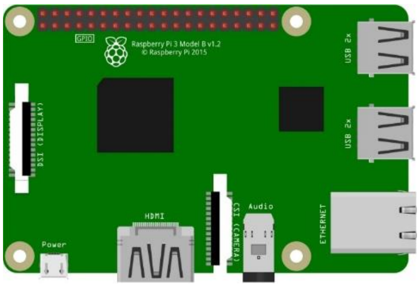
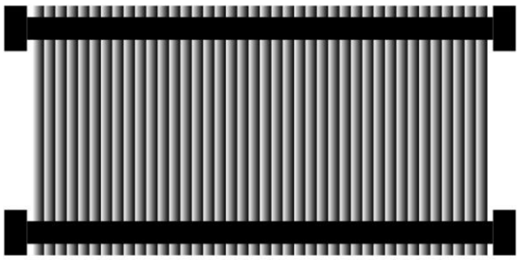


We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

Therefore, the temperature formula can be derived as:

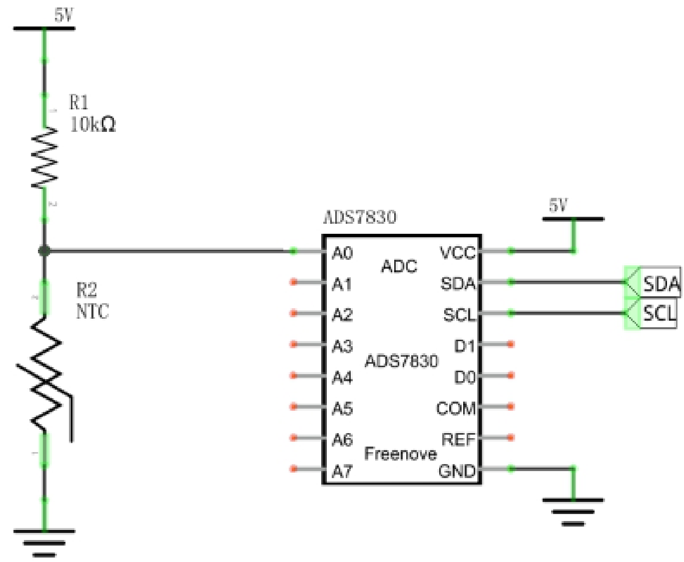
T2 = 1/(1/T1 + ln(Rt/R)/B)

Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
</table>

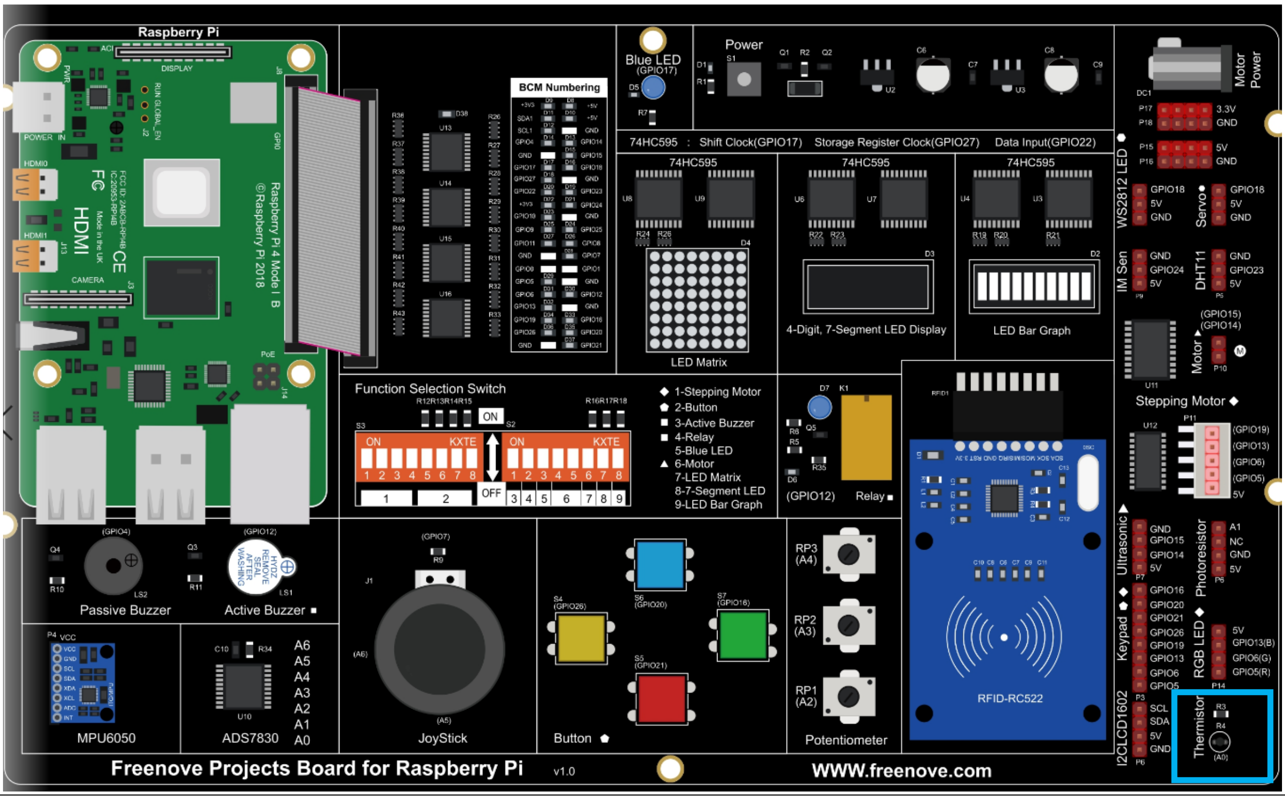
Circuit

Schematic diagram



Hardware connection.

After running the program, hold your finger against the sensor to observe the change.



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

Code

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

C Code 9.1 Thermometer

If you haven't configure I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 9_Thermometer directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/9_Thermometer
```

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

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

3 Then run the generated file "Thermometer".

```
sudo ./Thermometer
```

After the program is executed, the Terminal window will display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor with your index finger and thumb for a brief time, you should see that the temperature value increases.

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

The following is the code:

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

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

int main(void){
```

```
    adc = new ADCDevice();
    printf("Program is starting ... \n");
    if(adc->detectI2C(0x48)){    // Detect the ads7830
        delete adc;               // Free previously pointed memory
        adc = new ADS7830(0x48);      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    printf("Program is starting ... \n");
    while(1){
        int adcValue = adc->analogRead(0);  //read analog value A0 pin
        float voltage = (float)adcValue / 255.0 * 5.0;    // calculate voltage
        float Rt = 10 * voltage / (5.0 - voltage);        //calculate resistance value of
thermistor
        float tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0); //calculate temperature
(Kelvin)
        float tempC = tempK -273.15;        //calculate temperature (Celsius)
        printf("ADC value : %d  ,\tVoltage : %.2fV,
\tTemperature : %.2fC\n",adcValue,voltage,tempC);
        delay(100);
    }
    return 0;
}
```

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

Python Code 9.1 Thermometer

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.
First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 9_Thermometer directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/9_Thermometer
```

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

```
sudo python Thermometer.py
```

After the program is executed, the Terminal window will display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor with your index finger and thumb for a brief time, you should see that the temperature value increases.

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

The following is the code:

```python
import RPi.GPIO as GPIO
import time
import math
from ADCDevice import *

adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)

```

```python
def loop():
    while True:
        value = adc.analogRead(0)        # read ADC value A0 pin
        voltage = value / 255.0 * 5.0        # calculate voltage
        Rt = 10 * voltage / (5.0 - voltage)     # calculate resistance value of thermistor
        tempK =  1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) # calculate temperature (Kelvin)
        tempC = tempK -273.15        # calculate temperature (Celsius)
        print ('ADC Value : %d, Voltage : %.2f, Temperature : %.2f'%(value,voltage,tempC))
        time.sleep(0.01)

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

if __name__ == '__main__':  # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

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

Chapter 10 Joystick

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

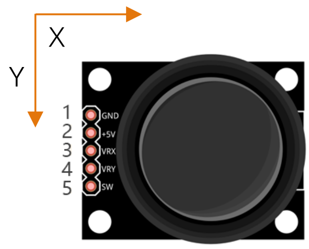
Project 10.1 Joystick

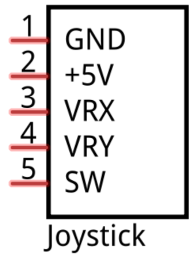
In this project, we will read the output data of a joystick and display it to the Terminal screen.

Component knowledge

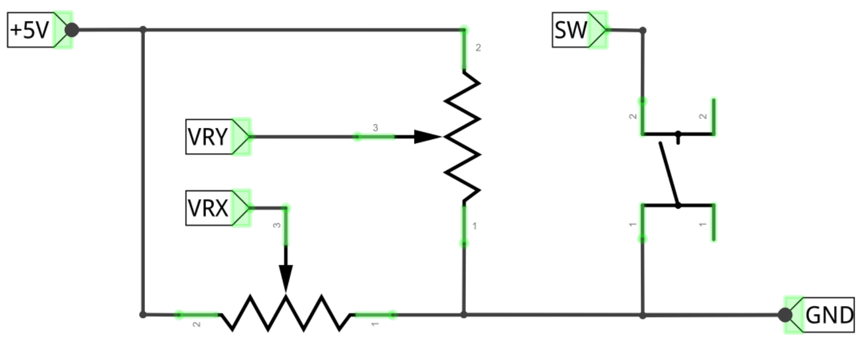
Joystick

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





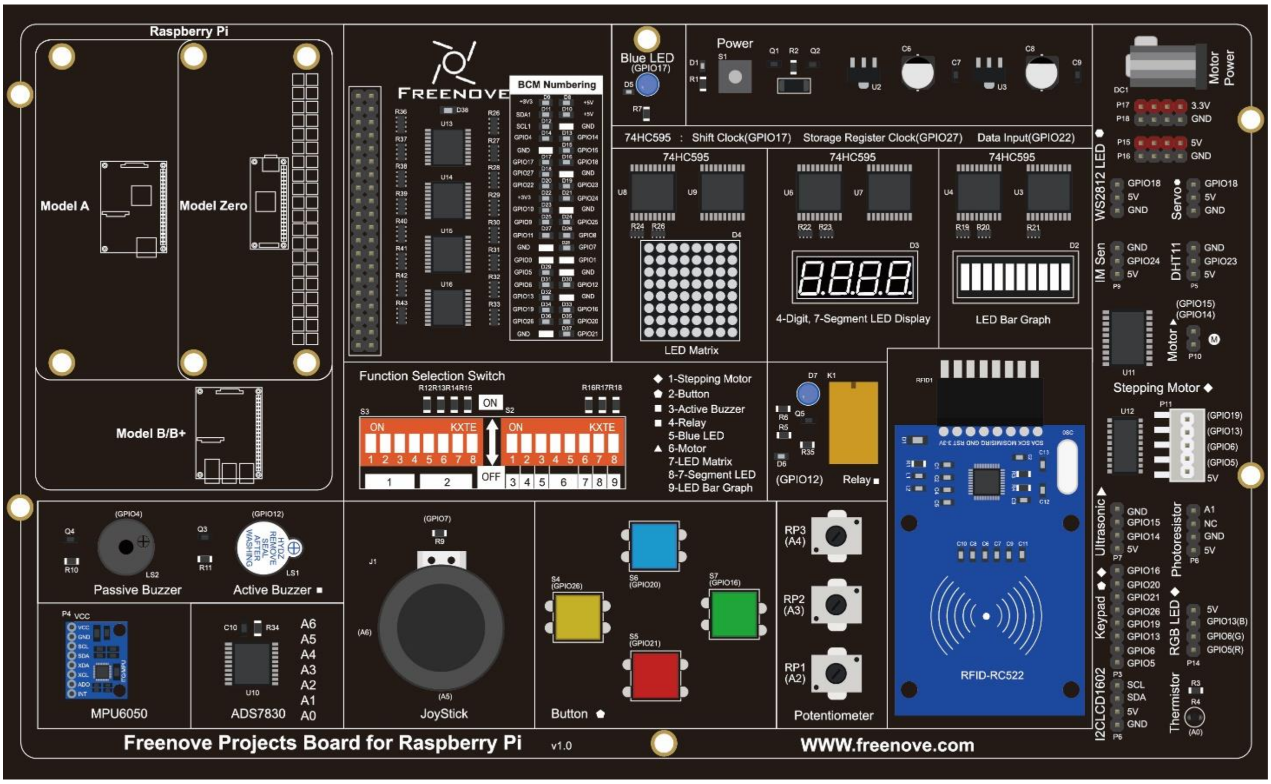
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in two directions simultaneously and with a Push Button Switch in the "vertical" axis, which can detect when a User presses on the Joystick.



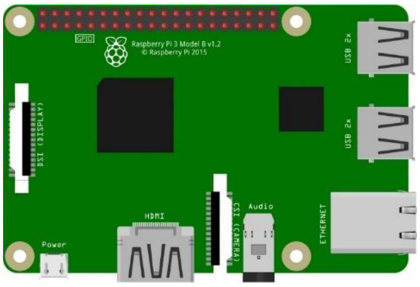
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Component List

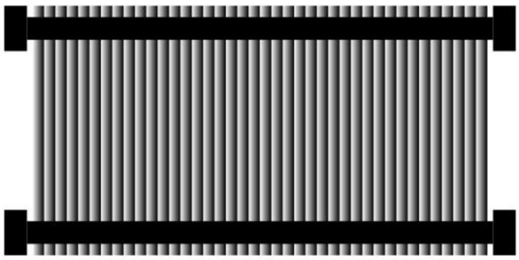
Freenove Projects Board for Raspberry Pi



Raspberry Pi

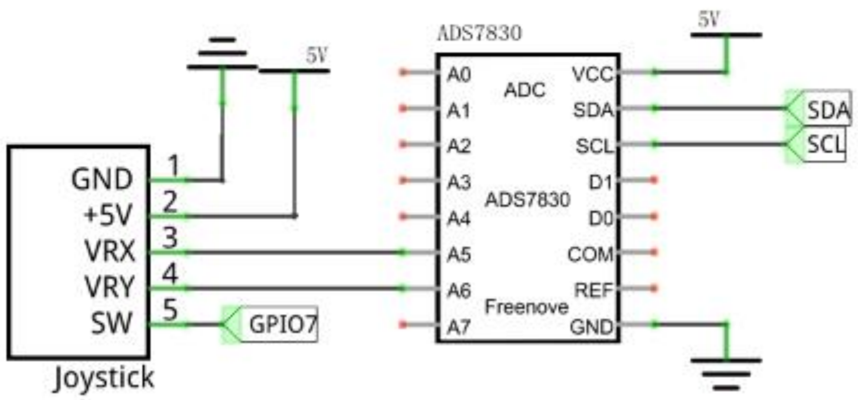


GPIO Ribbon Cable

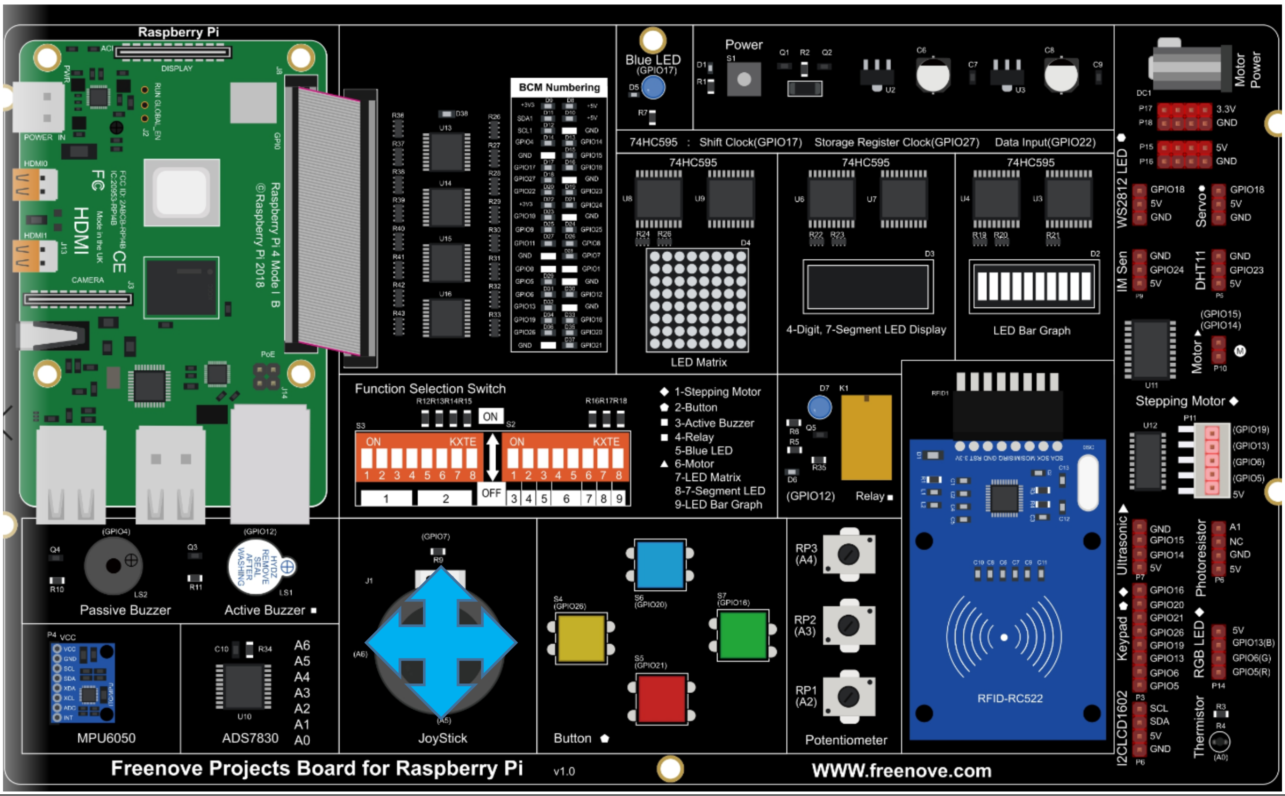


Circuit

Schematic diagram



Hardware connection.



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

Code

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

C Code 10.1 Joystick

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 10_Joystick directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/10_Joystick
```

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

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

3. Then run the generated file "Joystick".

```
sudo ./Joystick
```

After the program is executed, the terminal window will display the data of 3 axes X, Y and Z. Shifting (moving) the Joystick or pressing it down will make the data change.

```
val_X: 128  ,   val_Y: 135  ,   val_Z: 1
val_X: 128  ,   val_Y: 155  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 181  ,   val_Y: 255  ,   val_Z: 1
val_X: 128  ,   val_Y: 255  ,   val_Z: 1
val_X: 128  ,   val_Y: 180  ,   val_Z: 0
val_X: 128  ,   val_Y: 138  ,   val_Z: 0
val_X: 128  ,   val_Y: 137  ,   val_Z: 0
val_X: 128  ,   val_Y: 139  ,   val_Z: 0
val_X: 128  ,   val_Y: 139  ,   val_Z: 1
```

The flowing is the code:

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

#define Z_Pin 11      //define pin for axis Z

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

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

```

    if(adc->detectI2C(0x48)){    // Detect the ads7830
        delete adc;               // Free previously pointed memory
        adc = new ADS7830(0x48);      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    pinMode(Z_Pin,INPUT);        //set Z_Pin as input pin and pull-up mode
    pullUpDnControl(Z_Pin,PUD_UP);
    while(1){
        int val_Z = digitalRead(Z_Pin);  //read digital value of axis Z
        int val_Y = adc->analogRead(5);      //read analog value of axis X and Y
        int val_X = adc->analogRead(6);
        printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
        delay(100);
    }
    return 0;
}
```

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

```
    while(1){
        int val_Z = digitalRead(Z_Pin);  //read digital value of axis Z
        int val_Y = adc->analogRead(5);      //read analog value of axis X and Y
        int val_X = adc->analogRead(6);
        printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
        delay(100);
    }
```

Python Code 10.1 Joystick

If you haven't configured I2C, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 10_Joystick directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/10_Joystick
```

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

```
python Joystick.py
```

After the program is executed, the Terminal window will display the data of 3 axes X, Y and Z. Shifting (moving) the joystick or pressing it down will make the data change.

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

The following is the program code:

```python
import RPi.GPIO as GPIO
import time
from ADCDevice import *

Z_Pin = 26      # define Z_Pin
adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(Z_Pin,GPIO.IN,GPIO.PUD_UP)   # set Z_Pin to pull-up mode
def loop():
    while True:
        val_Z = GPIO.input(Z_Pin)       # read digital value of axis Z
        val_Y = adc.analogRead(5)           # read analog value of axis X and Y
        val_X = adc.analogRead(6)
```

```
        print ('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
        time.sleep(0.01)

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

if __name__ == '__main__':
    print ('Program is starting ... ') # Program entrance
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

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

```
    while True:
        val_Z = GPIO.input(Z_Pin)       # read digital value of axis Z
        val_Y = adc.analogRead(5)           # read analog value of axis X and Y
        val_X = adc.analogRead(6)
        print ('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
        time.sleep(0.01)
```

Chapter 11 Motor & Driver

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

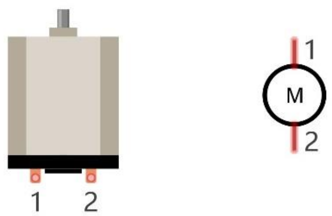
Project 11.1 Control a DC Motor with a Potentiometer

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 knowledge

DC Motor

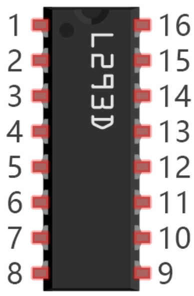
DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair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

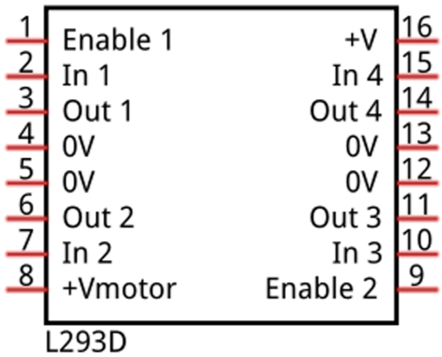


When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.

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).





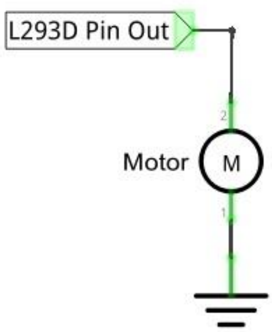
Port description of L293D module is as follows:

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

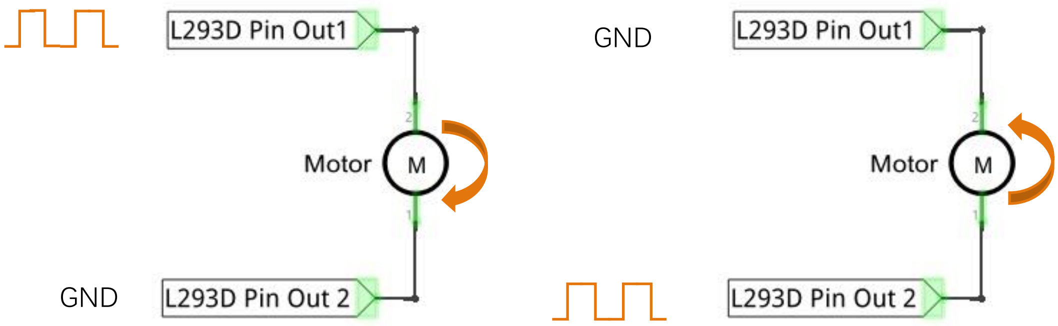
For more details, please see the datasheet for this IC Chip.

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.

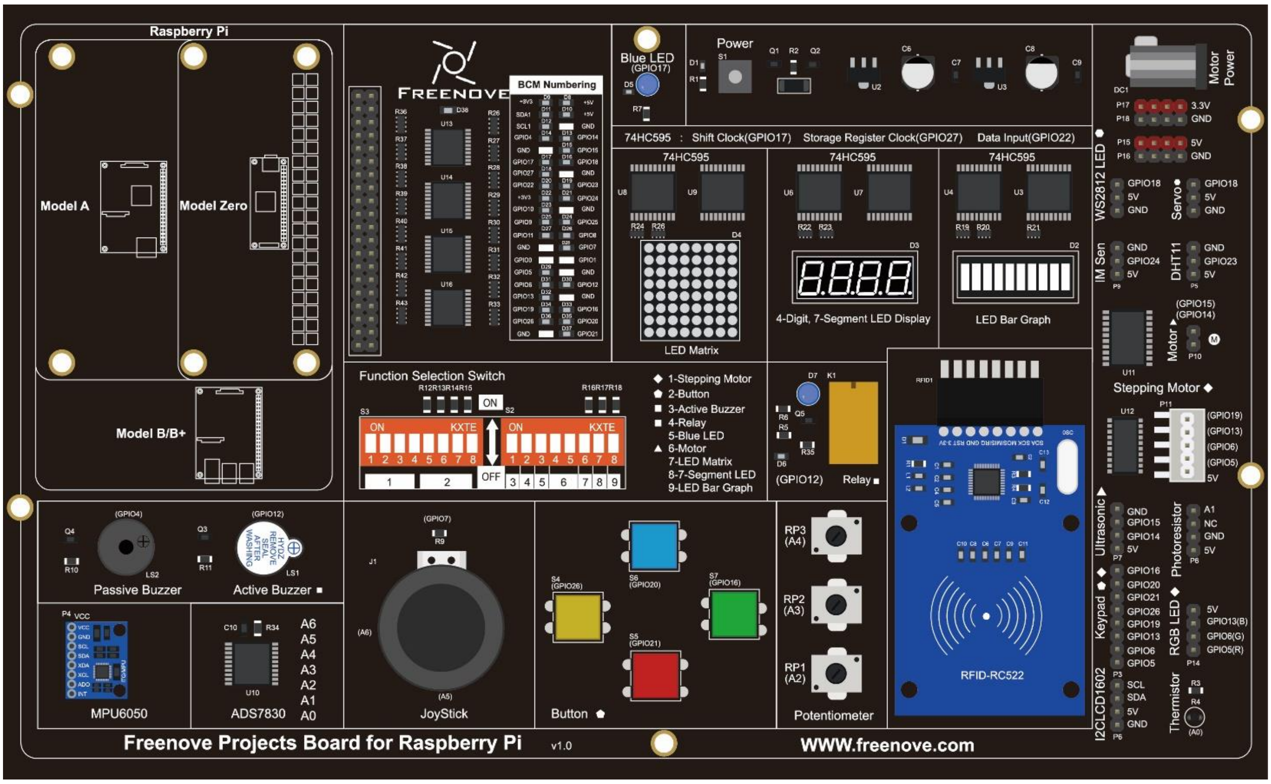
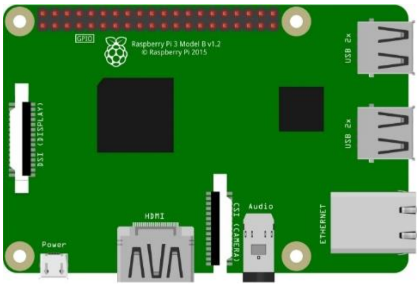
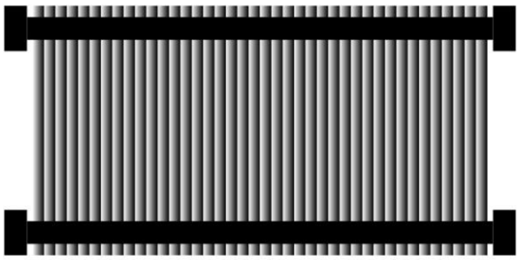
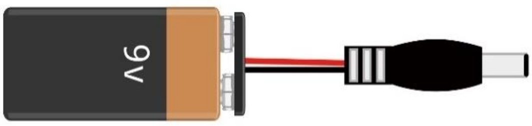


The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, it not only controls the speed of motor, but also can control the direction of the motor.



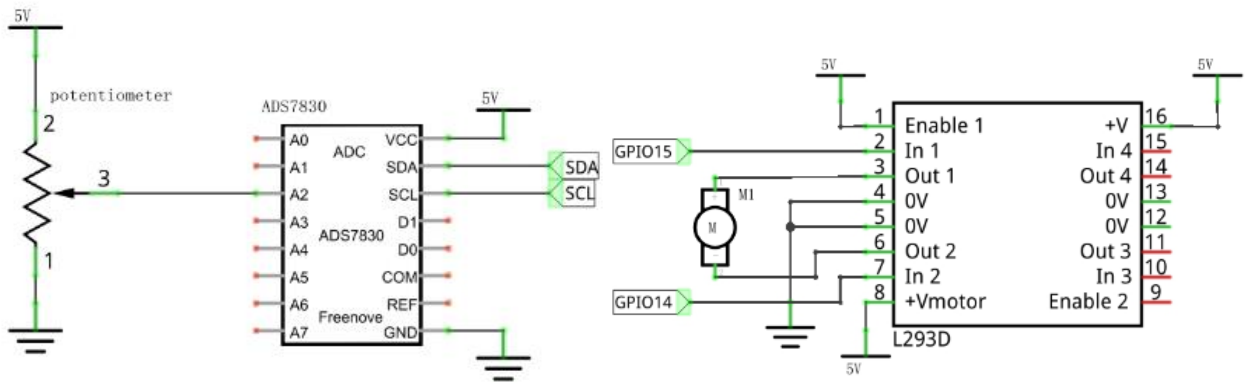
In practical use, the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

Component List

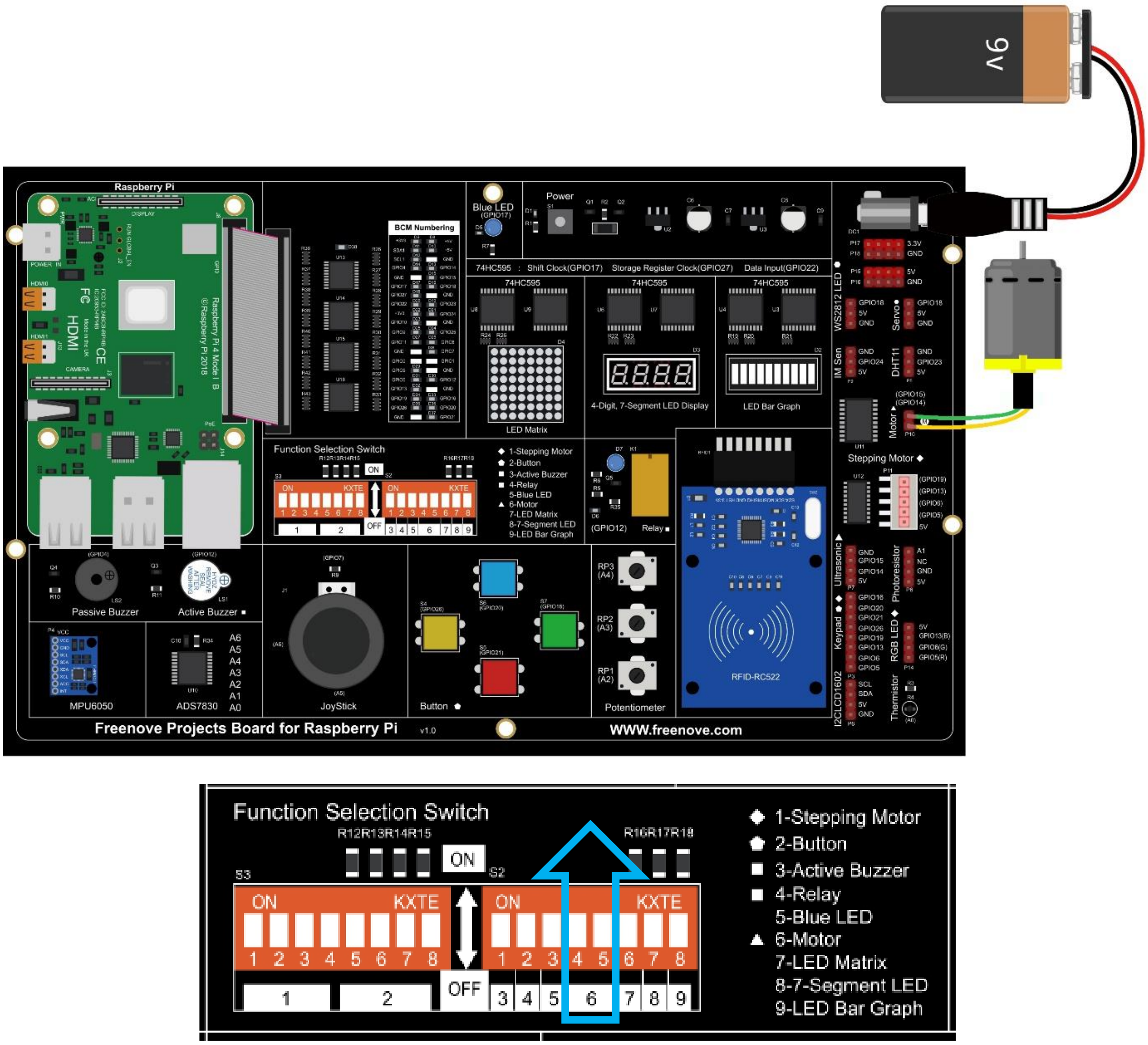
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi

</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>Motor
</td></tr>
<tr><td>9V Battery (you provide) & 9V Battery Cable

</td><td></td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



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

Code

In code for this project, first read the ADC value and then control the rotation direction and speed of the DC Motor according to the value of the ADC.

C Code 11.1 Motor

If you haven't configured I2C, please refer to Chapter 7. If you have done it, please continue.

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

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

1. Use cd command to enter 11_Motor directory of the C code.

```
cd ~/Freenove_Kit/Code/C_Code/11_Motor
```

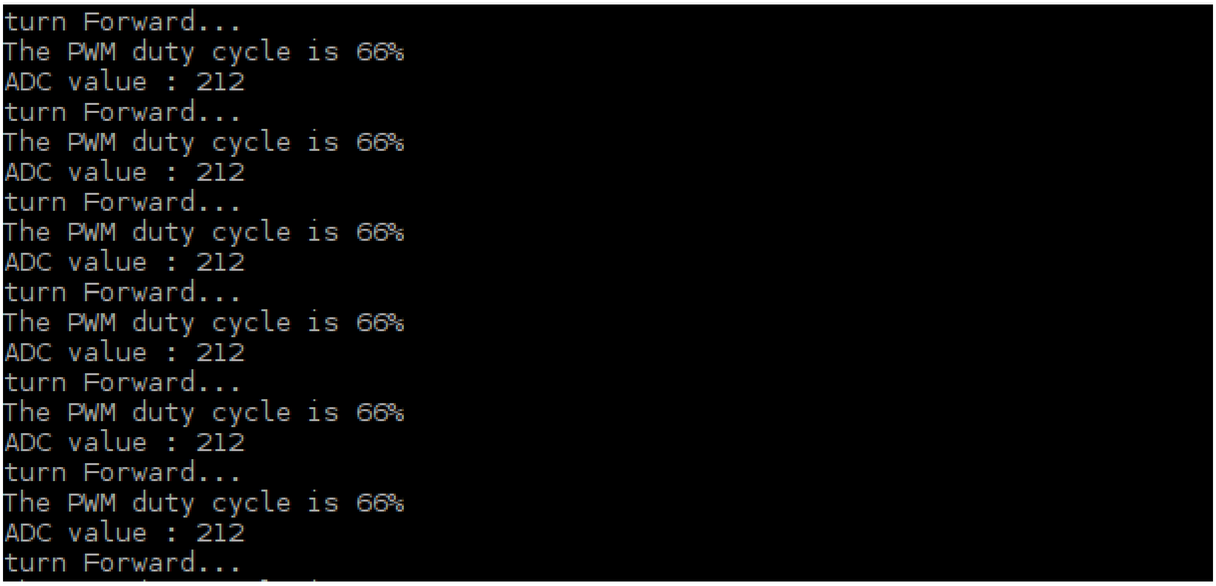
2. Use the following command to compile "Motor.cpp" and generate the executable file "Motor".

```
sudo g++ Motor.cpp -o Motor -lwiringPi -lADCDevice
```

3. Then run the generated file "Motor".

```
sudo ./Motor
```

After the program runs, you can use the Potentiometer to control the DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reaches the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint, the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction. You will also see the ADC value of the potentiometer displayed in the Terminal with the motor direction and the PWM duty cycle used to control the DC Motor's speed.



The following is the code:

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

```
#include <ADCDevice.hpp>

#define motorPin1    15        //define the pin connected to L293D
#define motorPin2    16
#define enablePin    3

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

//Map function: map the value from a range to another range.
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
//motor function: determine the direction and speed of the motor according to the ADC
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        softPwmWrite(motorPin1,map(abs(value),0,128,0,100));
        softPwmWrite(motorPin2,0);
        printf("turn Forward...\n");
    }
    else if (value<0){
        softPwmWrite(motorPin1,0);
        softPwmWrite(motorPin2,map(-value,0,128,0,100));
        printf("turn Back...\n");
    }
    else {
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,LOW);
        printf("Motor Stop...\n");
    }

    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);//print the PMW duty cycle
}
int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

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

```
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    pinMode(motorPin1,OUTPUT);
    pinMode(motorPin2,OUTPUT);

    softPwmCreate(motorPin1,0,100);//define PMW pin
    softPwmCreate(motorPin2,0,100);//define PMW pin

    while(1){
        int value = adc->analogRead(2);  //read analog value of A0 pin
        printf("ADC value : %d \n",value);
        motor(value);       //make the motor rotate with speed(analog value of A0 pin)
        delay(100);
    }
    return 0;
}
```

When ADC value is greater than 128, motorPin2 outputs low lever and motorPin1 output high level. When ADC value is less than 128, motorPin2 outputs high lever and motorPin1 output low level. The difference between ADC and 128 determines the duty cycle for the PWM.

```
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        softPwmWrite(motorPin1,map(abs(value),0,128,0,100));
        softPwmWrite(motorPin2,0);
        printf("turn Forward...\n");
    }
    else if (value<0){
        softPwmWrite(motorPin1,0);
        softPwmWrite(motorPin2,map(-value,0,128,0,100));
        printf("turn Back...\n");
    }
    else {
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,LOW);
        printf("Motor Stop...\n");
    }

    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);//print the PMW duty cycle
}
```

Python Code 11.1 Motor

If you haven't configured I2C and installed Smbus, please refer to Chapter 7. If you've done it, please Continue. First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 11_Motor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/11_Motor
```

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

```
sudo python Motor.py
```

After the program runs, you can use the Potentiometer to control the DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reaches the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint, the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction. You will also see the ADC value of the potentiometer displayed in the Terminal with the motor direction and the PWM duty cycle used to control the DC Motor's speed.

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

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

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

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

The following is the code:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

# define the pins connected to L293D
motoRPin1 = 8
motoRPin2 = 10
adc = ADCDevice(0x48) # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = ADS7830(0x48)
    else:
```

```
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    global p1
    global p2
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(motoRPin1,GPIO.OUT)   # set pins to OUTPUT mode
    GPIO.setup(motoRPin2,GPIO.OUT)

    p1 = GPIO.PWM(motoRPin1,1000) # creat PWM and set Frequence to 1KHz
    p1.start(0)
    p2 = GPIO.PWM(motoRPin2,1000) # creat PWM and set Frequence to 1KHz
    p2.start(0)

# mapNUM function: map the value from a range of mapping to another range.
def mapNUM(value,fromLow,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

# motor function: determine the direction and speed of the motor according to the input ADC
value input
def motor(ADC):
    value = ADC -128

    if (value > 0):  # make motor turn forward
        print(abs(value)*100/127)
        p1.ChangeDutyCycle(abs(value)*100/127)
        p2.ChangeDutyCycle(0)
        print ('Turn Forward...')
    elif (value < 0): # make motor turn backward
        print(abs(value)*100/128)
        p1.ChangeDutyCycle(0)
        p2.ChangeDutyCycle(abs(value)*100/128)
        print ('Turn Backward...')
    else :
        p1.ChangeDutyCycle(0)
        p2.ChangeDutyCycle(0)
        print ('Motor Stop...')

def loop():
    while True:
        value = adc.analogRead(2) # read ADC value of channel 0
        print ('ADC Value : %d'%(value))
        motor(value)
```

```
        time.sleep(0.05)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':  # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

When ADC value is greater than 128, motorPin2 outputs low lever and motorPin1 output high level.
When ADC value is less than 128, motorPin2 outputs high lever and motorPin1 output low level.
The difference between ADC and 128 determines the duty cycle for the PWM.

```
def motor(ADC):
    value = ADC -128

    if (value > 0):  # make motor turn forward
        print(abs(value)*100/127)
        p1.ChangeDutyCycle(abs(value)*100/127)
        p2.ChangeDutyCycle(0)
        print ('Turn Forward...')
    elif (value < 0): # make motor turn backward
        print(abs(value)*100/128)
        p1.ChangeDutyCycle(0)
        p2.ChangeDutyCycle(abs(value)*100/128)
        print ('Turn Backward...')
    else :
        p1.ChangeDutyCycle(0)
        p2.ChangeDutyCycle(0)
        print ('Motor Stop...')
```

Chapter 12 Relay & LED

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

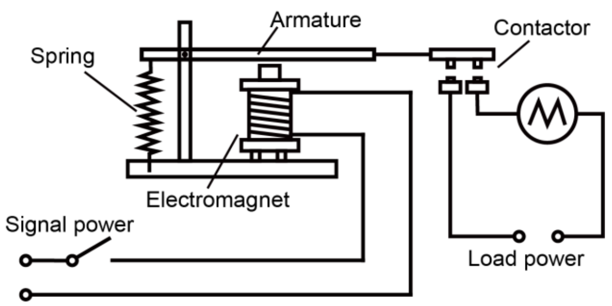
Project 12.1 Relay & LED

Component knowledge

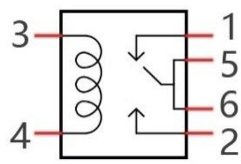
Relay

Relays are a type of Switch that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit using an electromagnet to initiate the Switch action. When the electromagnet is energized (powered), it will attract internal contacts completing a circuit, which act as a Switch. Many times Relays are used to allow a low powered circuit (and a small low amperage switch) to safely turn ON a larger more powerful circuit. They are commonly found in automobiles, especially from the ignition to the starter motor.

The following is a basic diagram of a common Relay and the image and circuit symbol diagram of the 5V relay used in this project:

<table>
<tr><th>Diagram</th><th>Feature：</th><th>Symbol</th></tr>
<tr><td>


</td><td>


</td><td>


</td></tr>
</table>

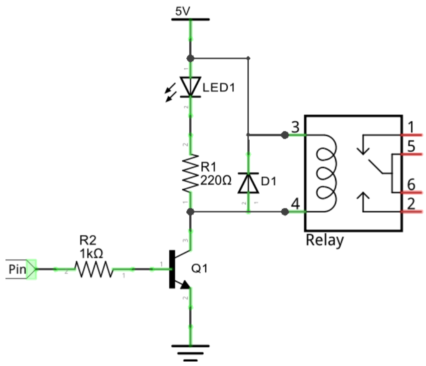
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

Inductor

The symbol of Inductance is “L” and the unit of inductance is the “Henry” (H). Here is an example of how this can be encountered: 1H=1000mH, 1mH=1000μH.

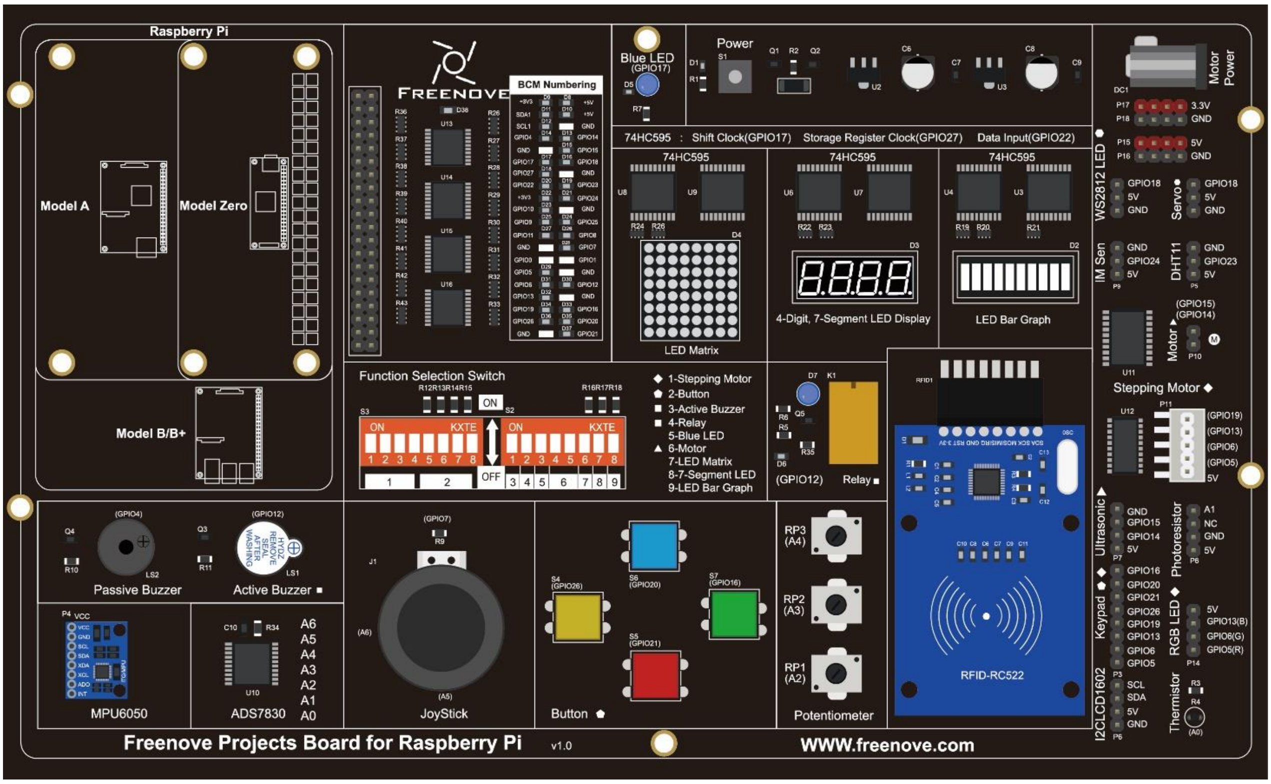
An Inductor is a passive device that stores energy in its Magnetic Field and returns energy to the circuit whenever required. An Inductor is formed by a Cylindrical Core with many Turns of conducting wire (usually copper wire). Inductors will hinder the changing current passing through it. When the current passing through the Inductor increases, it will attempt to hinder the increasing movement of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing movement of current. So the current passing through an Inductor is not transient.

The circuit for a Relay is as follows: The coil of Relay can be equivalent to an Inductor, when a Transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay’s coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes in parallel are placed on both ends of the Relay coil pins in opposite polar direction. Having the current pass through the diodes will avoid any adverse effect on the power supply.

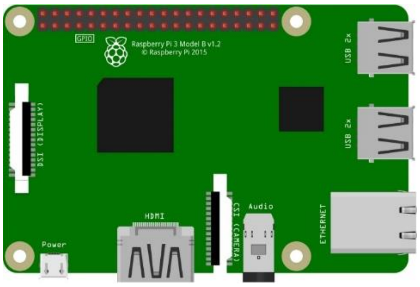


Component List

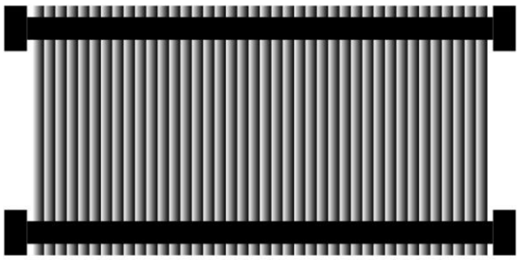
Freenove Projects Board for Raspberry Pi



Raspberry Pi

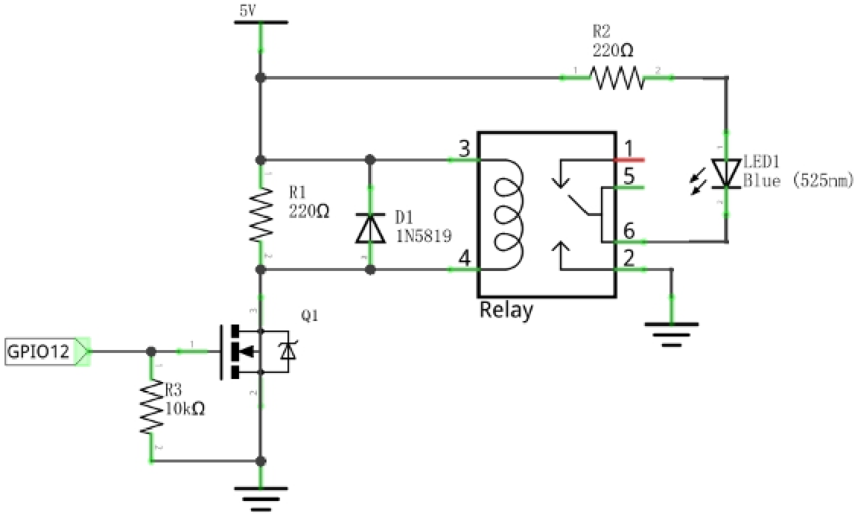


GPIO Ribbon Cable

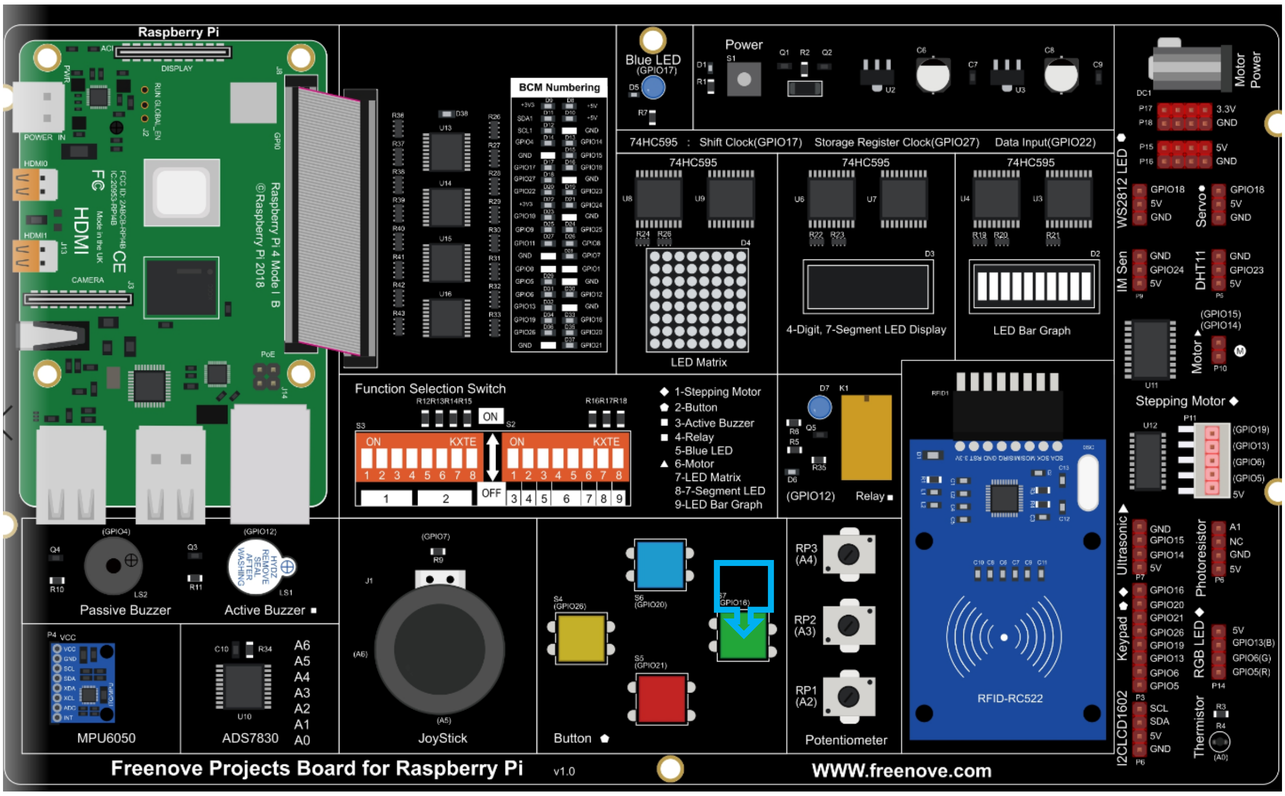


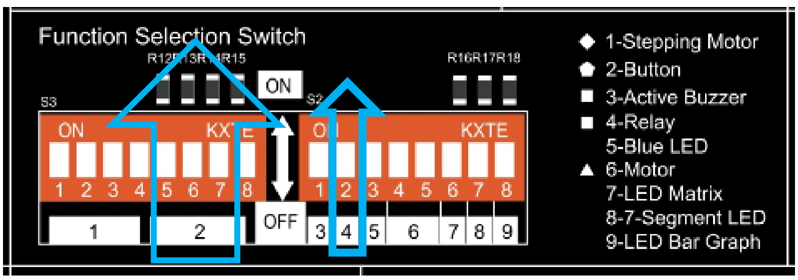
Circuit

Schematic diagram



Hardware connection.





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

Code

C Code 12.1 Relay

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

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

1. Use cd command to enter 12_Relay directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/12_Relay
```

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

```
gcc Relay.c -o Relay -lwiringPi
```

3. Run the generated file "Relay".

```
sudo ./Relay
```

After running the program, press the button, the LED near relay will light up.
Press the button again, the LED will light OFF.

The following is the program code:

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

#define relayPin    26    //define the relayPin
#define buttonPin     27      //define the buttonPin
int relayState=LOW;           //store the State of relay
int buttonState=HIGH;      //store the State of button
int lastbuttonState=HIGH;  //store the lastState of button
long lastChangeTime;       //store the change time of button state
long captureTime=50;       //set the button state stable time
int reading;
int main(void)
{
    printf("Program is starting...\n");

    wiringPiSetup();

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

```
            //if changing-state of the button last beyond the time we set,we considered that
            //the current button state is an effective change rather than a buffeting
            if(millis() - lastChangeTime > captureTime){
                //if button state is changed, update the data.
                if(reading != buttonState){
                    buttonState = reading;
                    //if the state is low, the action is pressing.
                    if(buttonState == LOW){
                        printf("Button is pressed!\n");
                        relayState = !relayState;
                        if(relayState){
                            printf("turn on relay ...\n");
                        }
                        else {
                            printf("turn off relay ...\n");
                        }
                    }
                    //if the state is high, the action is releasing.
                    else {
                        printf("Button is released!\n");
                    }
                }
            }
            digitalWrite(relayPin,relayState);
            lastbuttonState = reading;
        }

        return 0;
}
```

Python Code 12.1 Relay

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

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

1. Use cd command to enter 12_Relay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/12_Relay
```

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

```
python Relay.py
```

After running the program, press the button, the LED near relay will light up.

Press the button again, the LED will light OFF.

he following is the program code:

```python
import RPi.GPIO as GPIO
import time

relayPin = 32     # define the relayPin
buttonPin = 38    # define the buttonPin
debounceTime = 50

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(relayPin, GPIO.OUT)   # set relayPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN)   # set buttonPin to INTPUT mode

def loop():
    relayState = False
    lastChangeTime = round(time.time()*1000)
    buttonState = GPIO.HIGH
    lastButtonState = GPIO.HIGH
    reading = GPIO.HIGH
    while True:
        reading = GPIO.input(buttonPin)
        if reading != lastButtonState :
            lastChangeTime = round(time.time()*1000)
        if ((round(time.time()*1000) - lastChangeTime) > debounceTime):
            if reading != buttonState :
                buttonState = reading;
                if buttonState == GPIO.LOW:
                    print("Button is pressed!")
                    relayState = not relayState
                    if relayState:
                        print("Turn on relay ...")
                    else :
                        print("Turn off relay ... ")
```

```python
                else :
                    print("Button is released!")
        GPIO.output(relayPin,relayState)
        lastButtonState = reading # lastButtonState store latest state

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

Chapter 13 Servo

Previously, we learned how to control the speed and rotational direction of a DC Motor. In this chapter, we will learn about Servos which are a rotary actuator type motor that can be controlled rotate to specific angles.

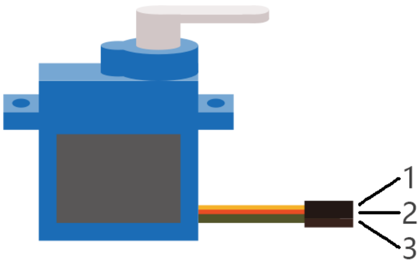
Project 13.1 Sweep

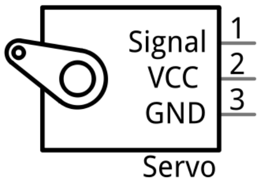
First, we need to learn how to make a Servo rotate.

Component knowledge

Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.



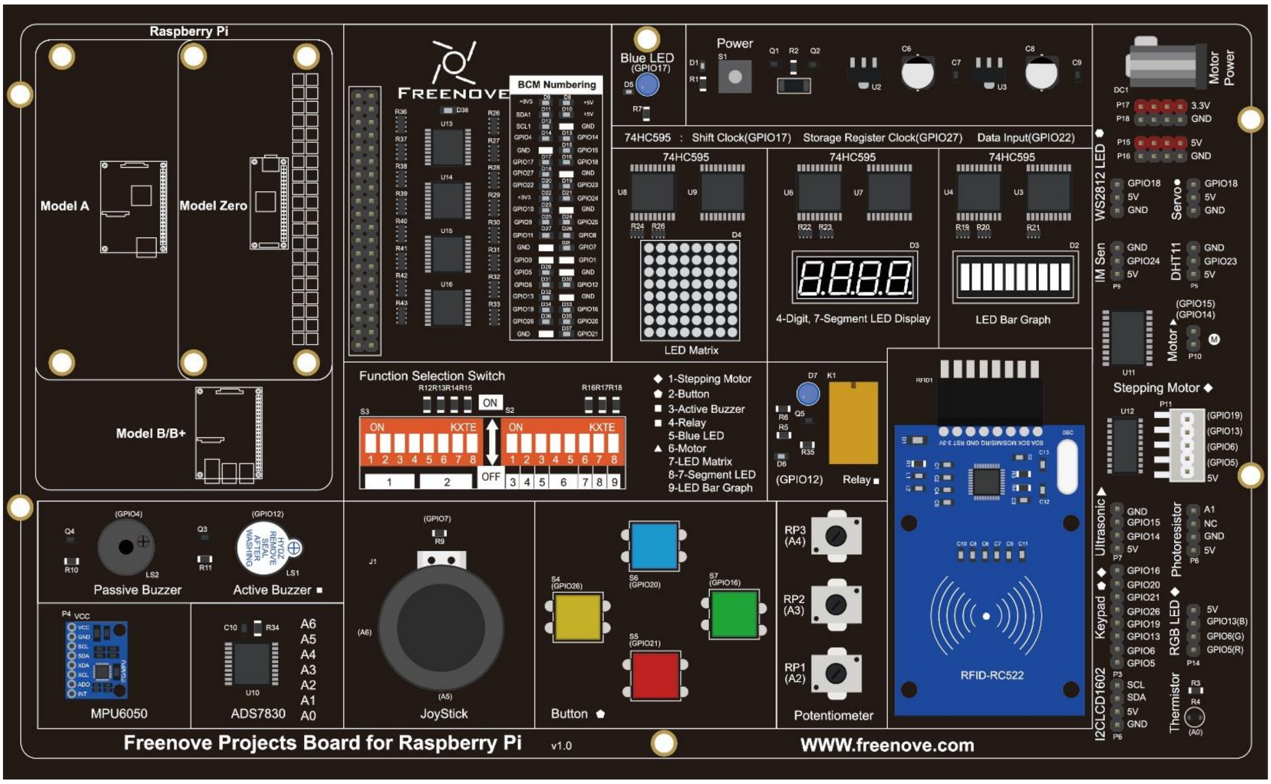
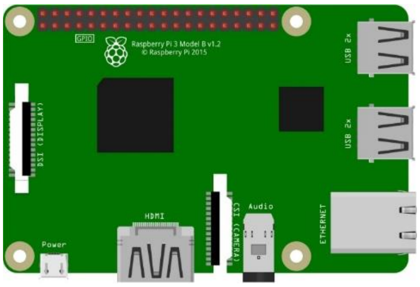
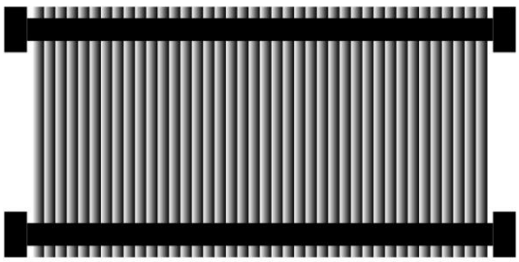
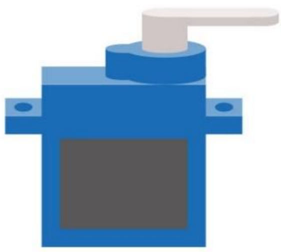


We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:
Note: the lasting time of high level corresponding to the servo angle is absolute instead of accumulating. For example, the high level time lasting for 0.5ms correspond to the 0 degree of the servo. If the high level time lasts for another 1ms, the servo rotates to 45 degrees.

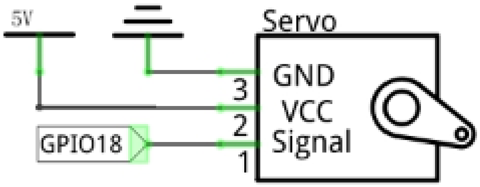
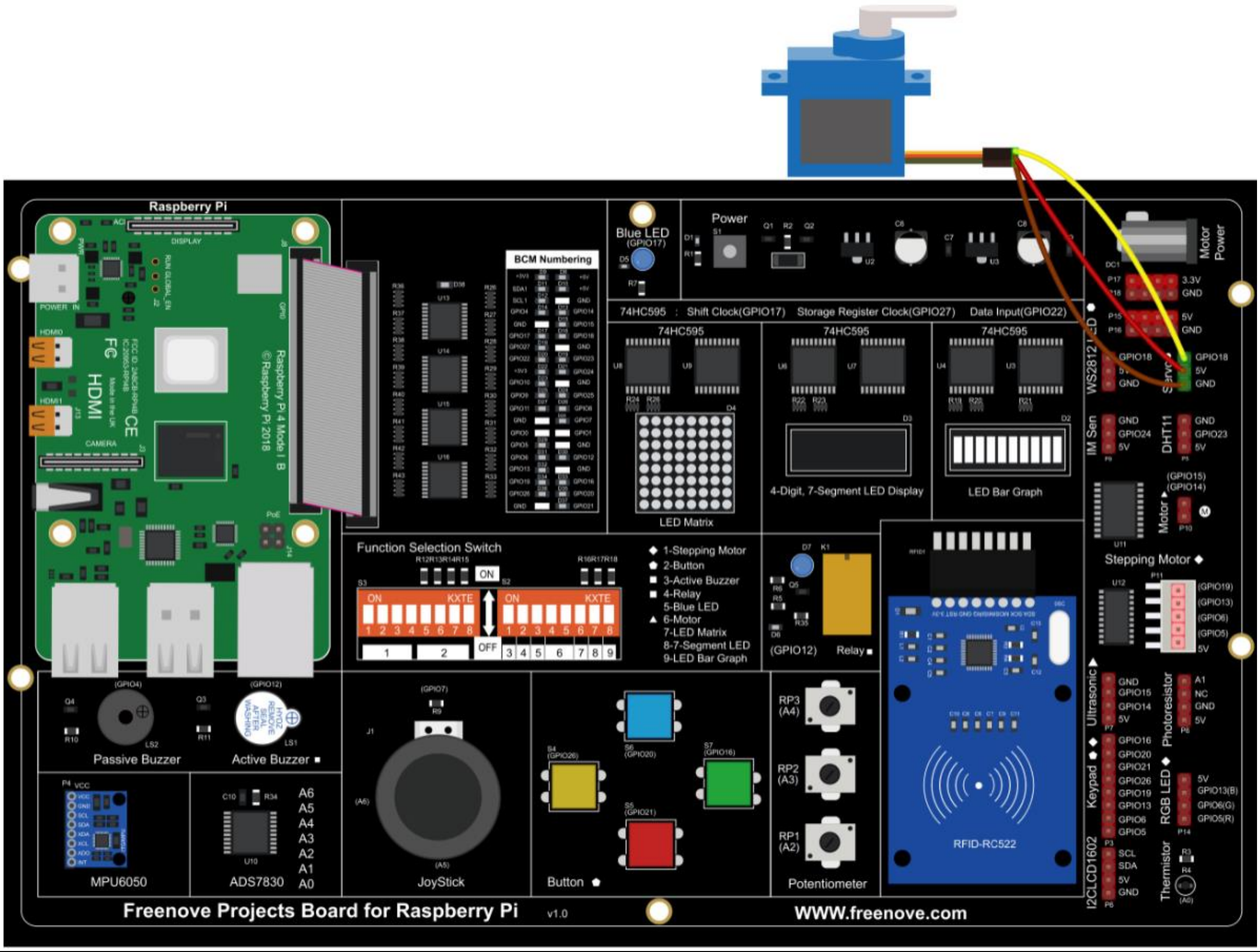
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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

Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>Servo
</td></tr>
</table>

Circuit

Schematic diagram 
Hardware connection. 

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

Code

In this project, we will make a Servo rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

C Code 13.1 Sweep

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

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

1. Use cd command to enter 13_1_Sweep directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/13_1_Sweep
```

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

```
gcc Sweep.c -o Sweep -lwiringPi
```

3. Run the generated file "Sweep".

```
sudo ./Sweep
```

After the program runs, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

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

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

```
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}

int main(void)
{
    int i;

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

    wiringPiSetup();
    servoInit(servoPin);        //initialize PMW pin of servo
    while(1){
        for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle to
maximum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
        for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle to
minimum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
    }
    return 0;
}
```

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

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

Since 0-180 degrees of the Servo's motion corresponds to the PWM pulse width of 0.5-2.5ms, with a PwmRange of 200 ms, we then need the function **softPwmWrite** (int pin, int value) and the scope 5-25 of the parameter values to correspond to 0-180 degrees' motion of the Servo. What's more, the number written in subfunction **servoWriteMS** () should be within the range of 5-25. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

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

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

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

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

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

Python Code 13.1 Sweep

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

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

1. Use cd command to enter 13_1_Sweep directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/13_1_Sweep
```

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

```
python Sweep.py
```

After the program runs, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

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

def map( value, fromLow, fromHigh, toLow, toHigh):  # map a value from one range to another range
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)         # use PHYSICAL GPIO Numbering
    GPIO.setup(servoPin, GPIO.OUT)   # Set servoPin to OUTPUT mode
    GPIO.output(servoPin, GPIO.LOW)  # Make servoPin output LOW level

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

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

def loop():
    while True:
        for dc in range(0, 181, 1):   # make servo rotate from 0 to 180 deg
            servoWrite(dc)     # Write dc value to servo
```

```
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): # make servo rotate from 180 to 0 deg
            servoWrite(dc)
            time.sleep(0.001)
        time.sleep(0.5)

def destroy():
    p.stop()
    GPIO.cleanup()

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

A 50 Hz pulse for a 20ms cycle is required to control the Servo, so we need to set the PWM frequency of servoPin to 50Hz.

```
p = GPIO.PWM(servoPin, 50)      # Set Frequency to 50Hz
```

As 0-180 degrees of the Servo's rotation corresponds to the PWM pulse width 0.5-2.5ms within cycle 20ms and to duty cycle 2.5%-12.5%. In subfunction **servoWrite** (angle), map the angle to duty cycle to output the PWM, then the Servo will rotate to specifically determined angle. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

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

Finally, in the "while" loop of main function, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

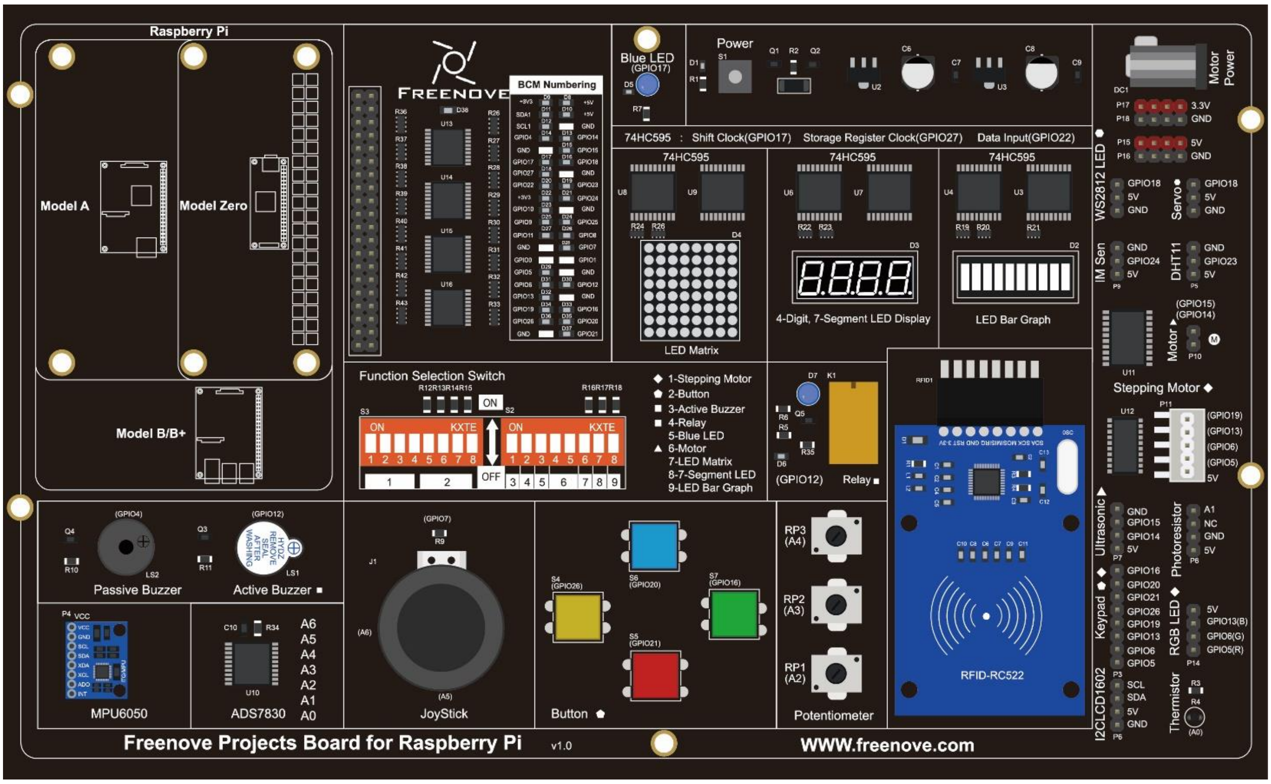
```
def loop():
    while True:
        for dc in range(0, 181, 1):   # make servo rotate from 0 to 180 deg
            servoWrite(dc)     # Write dc value to servo
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): # make servo rotate from 180 to 0 deg
     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

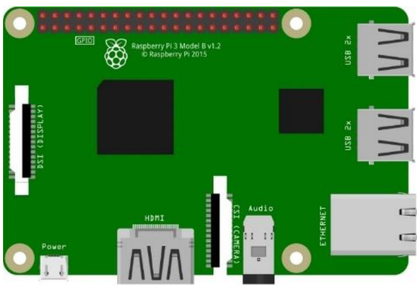
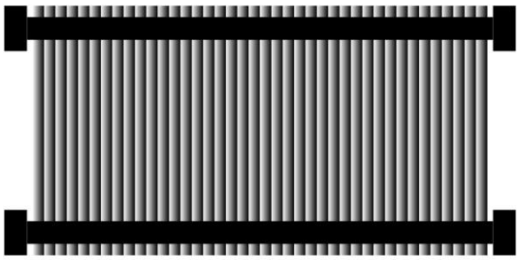
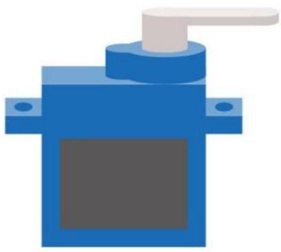
Project 13.2 Knob

In this project, we will learn how to control the servo with a potentiometer.

Component List

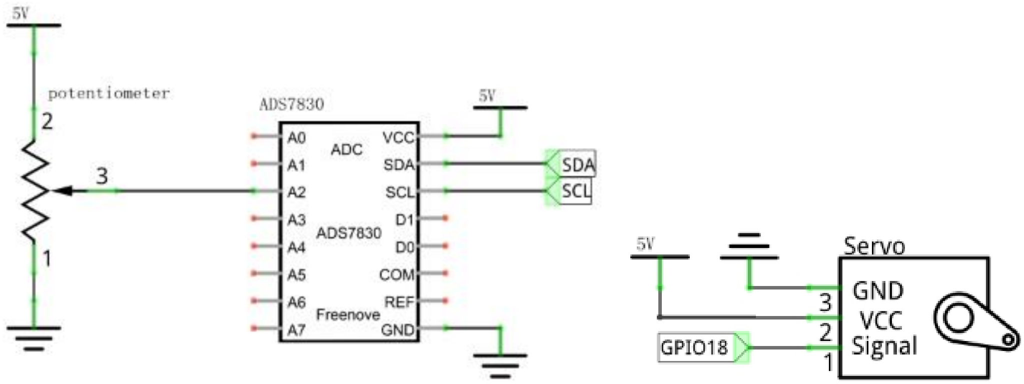
Freenove Projects Board for Raspberry Pi



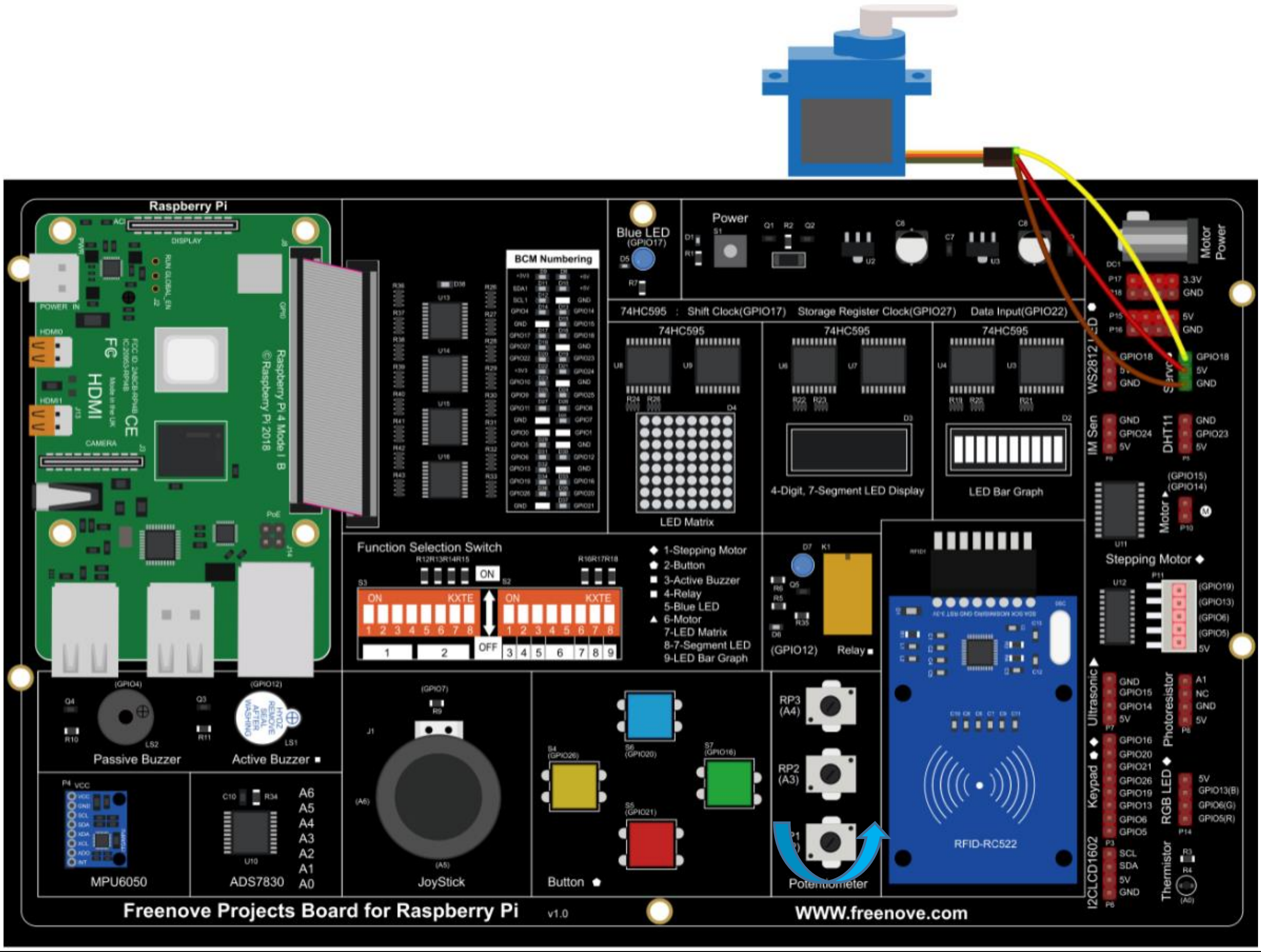
Raspberry Pi	GPIO Ribbon Cable
	
Jumper Wire	Servo
	

Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 13.2 Knob

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

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

4. Use cd command to enter 13_2_Knob directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/13_2_Knob
```

5. Use following command to compile " Knob.cpp" and generate executable file " Knob".

```
sudo g++ Knob.cpp -o Knob -lwiringPi -lADCDevice
```

6. Run the generated file " Knob ".

```
sudo ./Knob
```

After running the program, you can change the angle of the servo by rotating the potentiometer.

The following is the program code:

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

#define servoPin    1       //define the GPIO number connected to servo

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

long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}

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

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

```
void servoWriteMS(int pin, int ms){     //specific the unit for pulse(5-25ms) with specific
duration output by servo pin: 0.1ms
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}

int main(void)
{
    int i;
    printf("Program is starting ...\n");
    wiringPiSetup();
    servoInit(servoPin);              //initialize PMW pin of servo
    adc = new ADCDevice();
    if(adc->detectI2C(0x48)){         // Detect the ads7830
        delete adc;                   // Free previously pointed memory
        adc = new ADS7830(0x48);      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    while(1){
        int adcValue = adc->analogRead(2);    //read analog value of A2 pin
        printf("ADC value : %d  \n",adcValue);
        servoWrite(servoPin,map(adcValue,0,255,0,180));
        delay(10);
    }
    return 0;
}
```

Read the ADC value of channle2, and then the servo will rotate to corresponding angle.

```
    while(1){
        int adcValue = adc->analogRead(2);    //read analog value of A2 pin
        printf("ADC value : %d  \n",adcValue);
        servoWrite(servoPin,map(adcValue,0,255,0,180));
        delay(10);
    }
```

Python Code 13.2 Knob

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

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

3. Use cd command to enter 13_2_Knob directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/13_2_Knob
```

4. Use python command to execute code " Knob.py".

```
sudo python Knob.py
```

After running the program, you can change the angle of the servo by rotating the potentiometer.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

adc = ADCDevice(0x48) # Define an ADCDevice class object

OFFSE_DUTY = 0.5        #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
servoPin = 12

def map( value, fromLow, fromHigh, toLow, toHigh):  # map a value from one range to another
range
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

def setup():
    global adc
    if(adc.detectI2C(0x48)):
        adc = ADS7830(0x48)
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)

    global p
    GPIO.setmode(GPIO.BOARD)          # use PHYSICAL GPIO Numbering
    GPIO.setup(servoPin, GPIO.OUT)    # Set servoPin to OUTPUT mode
    GPIO.output(servoPin, GPIO.LOW)   # Make servoPin output LOW level

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

def servoWrite(angle):      # make the servo rotate to specific angle, 0-180
    if(angle<0):
        angle = 0
```

```
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY)) # map the angle to duty cycle and output it

def loop():
    while True:
        value = adc.analogRead(2)    # read the ADC value of channel 2
        servoWrite(round(value/255.0*180.0))
        print ('ADC Value : %d'%(value))
        time.sleep(0.1)

def destroy():
    p.stop()
    GPIO.cleanup()

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

Read the ADC value of channle2, and then the servo will rotate to corresponding angle.

```
while True:
        value = adc.analogRead(2)    # read the ADC value of channel 2
        servoWrite(round(value/255.0*180.0))
        print ('ADC Value : %d'%(value))
        time.sleep(0.1)
```

Finally, in the loop of main function, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

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

Chapter 14 Stepper Motor

Thus far, we have learned about DC Motors and Servos. A DC motor can rotate constantly in on direction but we cannot control the rotation to a specific angle. On the contrary, a Servo can rotate to a specific angle but cannot rotate constantly in one direction. In this chapter, we will learn about a Stepper Motor which is also a type of motor. A Stepper Motor can rotate constantly and also to a specific angle. Using a Stepper Motor can easily achieve higher accuracies in mechanical motion.

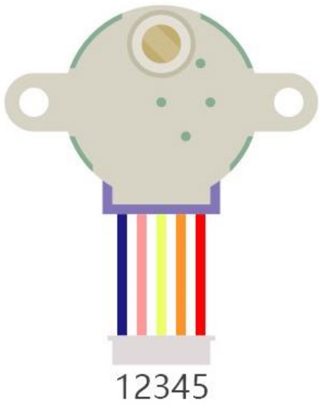
Project 14.1 Stepper Motor

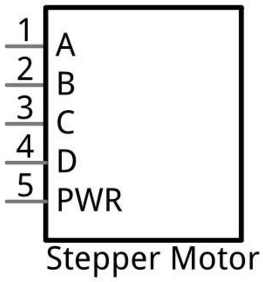
In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

Component knowledge

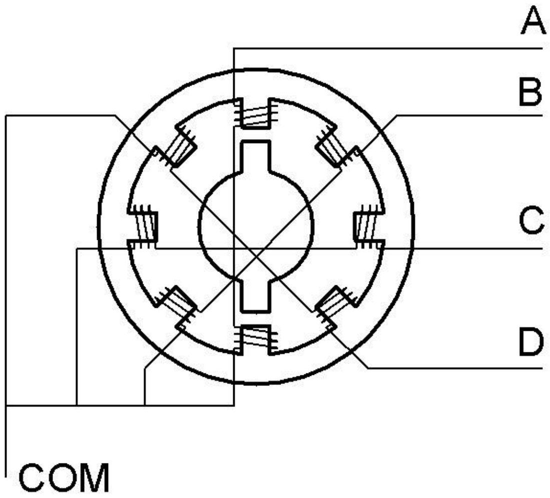
Stepper Motor

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:



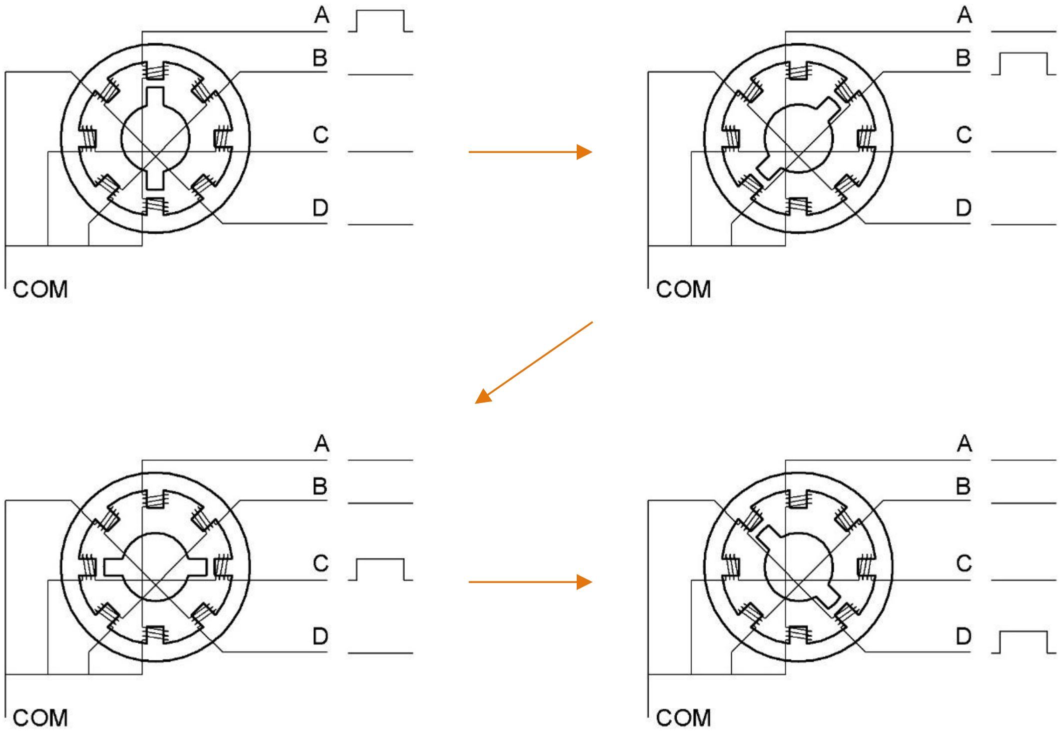


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There is a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor's surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving sequence is shown here:



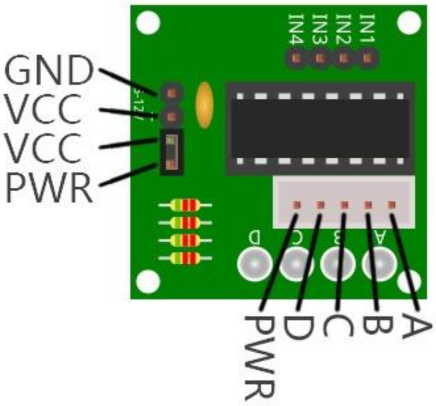
In the sequence above, the Stepper Motor rotates by a certain angle at once, which is called a “step”. By controlling the number of rotational steps, you can then control the Stepper Motor’s rotation angle. By defining the time between two steps, you can control the Stepper Motor’s rotation speed. When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

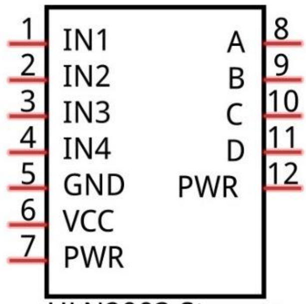
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. Tise sequence of powering the coils looks like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

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor’s housing) requires 32 X 64 = 2048 steps to make one full revolution.

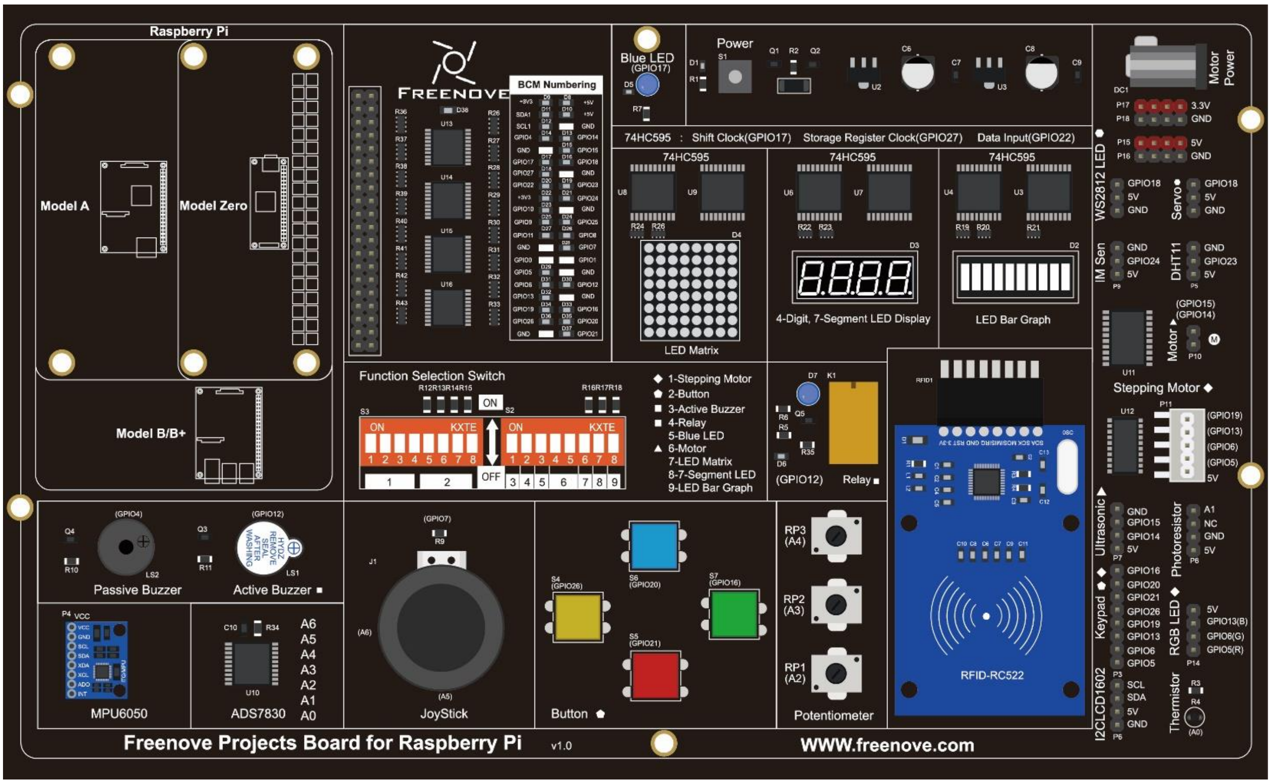
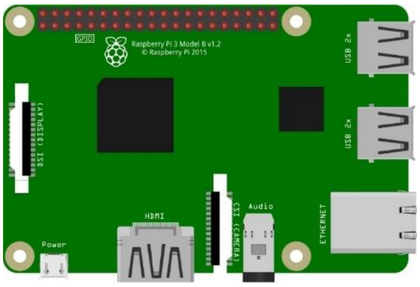
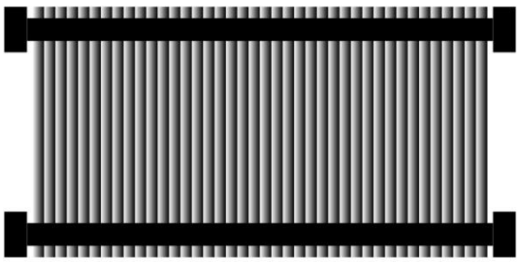
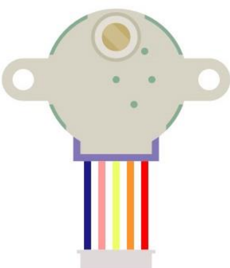
ULN2003 Stepper Motor driver

A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.



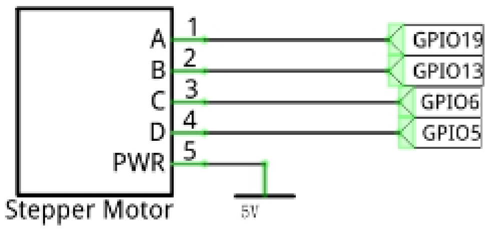


Component List

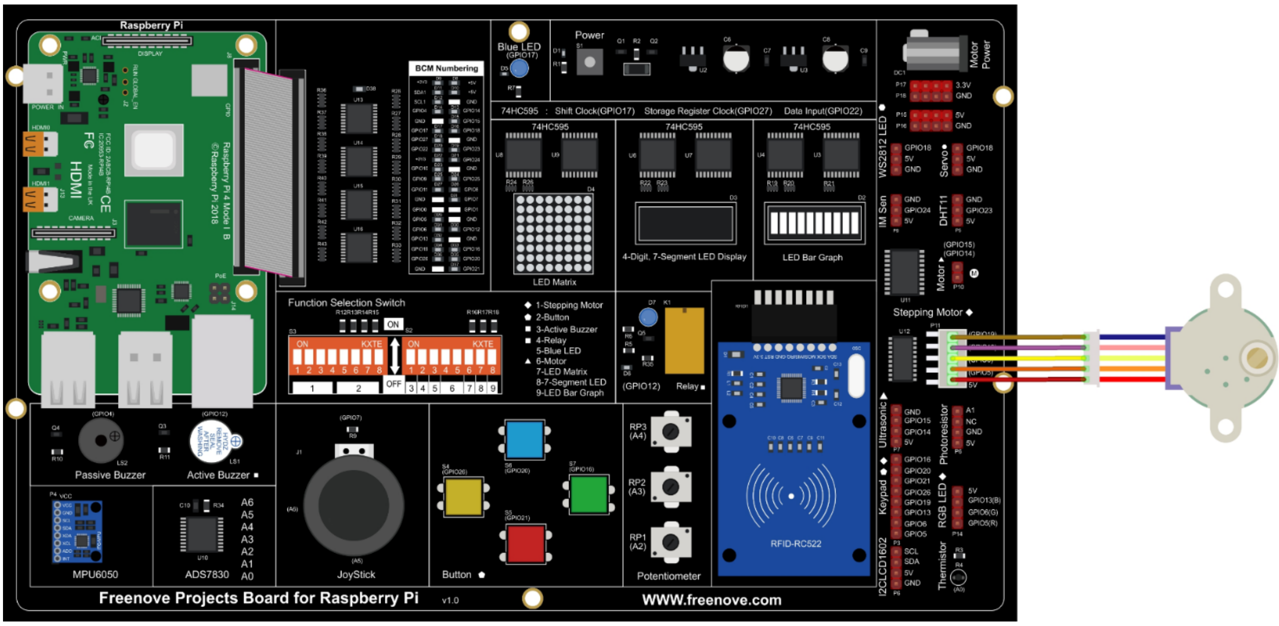
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Stepper Motor
</td><td></td></tr>
</table>

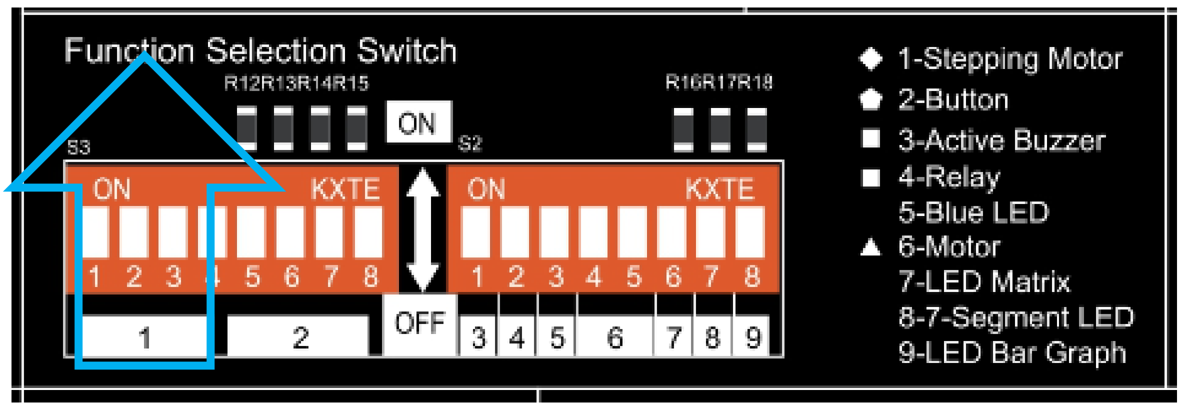
Circuit

Schematic diagram



Hardware connection.





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

Code

This code uses the four-step, four-part mode to drive the Stepper Motor in the clockwise and anticlockwise directions.

C Code 14.1 SteppingMotor

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

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

1. Use cd command to enter 14_SteppingMotor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/14_SteppingMotor
```

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

```
gcc SteppingMotor.c -o SteppingMotor -lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./SteppingMotor
```

After the program runs, the Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

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

const int motorPins[]={21,22,23,24};     //define pins connected to four phase ABCD of stepper motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for rotating anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};    //define power supply order for coil for rotating clockwise
//as for four phase stepping motor, four steps is a cycle. the function is used to drive the stepping motor clockwise or anticlockwise to take four steps
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order
        for (i=0;i<4;i++){  //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH : LOW);
            else        //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH : LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        printf("Step cycle!\n");
        if(ms<3)        //the delay can not be less than 3ms, otherwise it will exceed speed limit of the motor
            ms=3;
```

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

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

    wiringPiSetup();

    for(i=0;i<4;i++){
        pinMode(motorPins[i],OUTPUT);
    }

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

In the code we define the four pins of the Stepper Motor and the order to supply power to the coils for a four-step rotation mode.

```
const int motorPins[]={21,22,23,24};    //define pins connected to four phase ABCD of stepper motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for rotating anticlockwise
```

```
const int CWStep[]={0x08,0x04,0x02,0x01};    //define power supply order for coil for rotating
clockwise
```

Subfunction **moveOnePeriod** ((int dir,int ms) will drive the Stepper Motor rotating four-step clockwise or anticlockwise, four-step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

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

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

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

Subfunction **motorStop** () is used to stop the Stepper Motor.

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

Finally, in the while loop of main function, rotate one revolution clockwise, and then one revolution anticlockwise. According to the previous material covered, the Stepper Motor rotating for one revolution requires 2048 steps, that is, 2048/4=512 cycle.

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

Python Code 14.1 SteppingMotor

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

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

1. Use cd command to enter 14_StepperMotor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/14_StepperMotor
```

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

```
python SteppingMotor.py
```

After the program runs, the Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

motorPins = (29, 31, 33, 35)    # define pins connected to four phase ABCD of stepper motor
CCWStep = (0x01,0x02,0x04,0x08) # define power supply order for rotating anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  # define power supply order for rotating clockwise

def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    for pin in motorPins:
        GPIO.setup(pin,GPIO.OUT)

# as for four phase stepping motor, four steps is a cycle. the function is used to drive the 
stepping motor clockwise or anticlockwise to take four steps
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      # cycle for power supply order
        for i in range(0,4,1):  # assign to each pin
            if (direction == 1):# power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
            else :              # power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
```

```
        if(ms<3):       # the delay can not be less than 3ms, otherwise it will exceed speed limit of the motor
            ms = 3
        time.sleep(ms*0.001)

# continuous rotation function, the parameter steps specifies the rotation cycles, every four steps is a cycle
def moveSteps(direction, ms, steps):
    for i in range(steps):
        moveOnePeriod(direction, ms)

# function used to stop motor
def motorStop():
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)

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

def destroy():
    GPIO.cleanup()             # Release resource

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

In the code we define the four pins of the Stepper Motor and the order to supply power to the coils for a four-step rotation mode.

```
motorPins = (29, 31, 33, 35)     # define pins connected to four phase ABCD of stepper motor
CCWStep = (0x01,0x02,0x04,0x08) # define power supply order for rotating anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  # define power supply order for rotating clockwise
```

Subfunction **moveOnePeriod** ((int dir, int ms) will drive the Stepper Motor rotating four-step clockwise or anticlockwise, four-step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

```
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      # cycle for power supply order
        for i in range(0,4,1):  # assign to each pin
            if (direction == 1):# power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
            else :              # power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        if(ms<3):       # the delay can not be less than 3ms, otherwise it will exceed speed 
limit of the motor
            ms = 3
        time.sleep(ms*0.001)
```

Subfunction **moveSteps** (direction, ms, steps) is used to specify the cycle number of Stepper Motor.

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

Subfunction **motorStop** () is used to stop the Stepper Motor.

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

Finally, in the while loop of main function, rotate one revolution clockwise, and then one revolution anticlockwise. According to the previous material covered, the Stepper Motor rotating for one revolution requires 2048 steps, that is, 2048/4=512 cycle.

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

Chapter 15 LEDpixel

In this chapter, we will learn Freenove 8 RGB LED Module

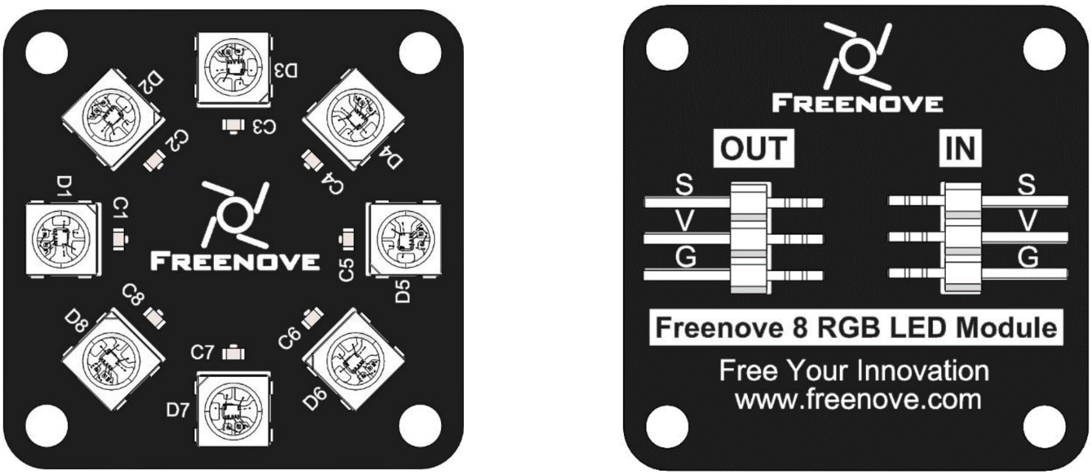
Project 15.1 LEDpixel

This project will achieve an RGB triple colored flowing water.

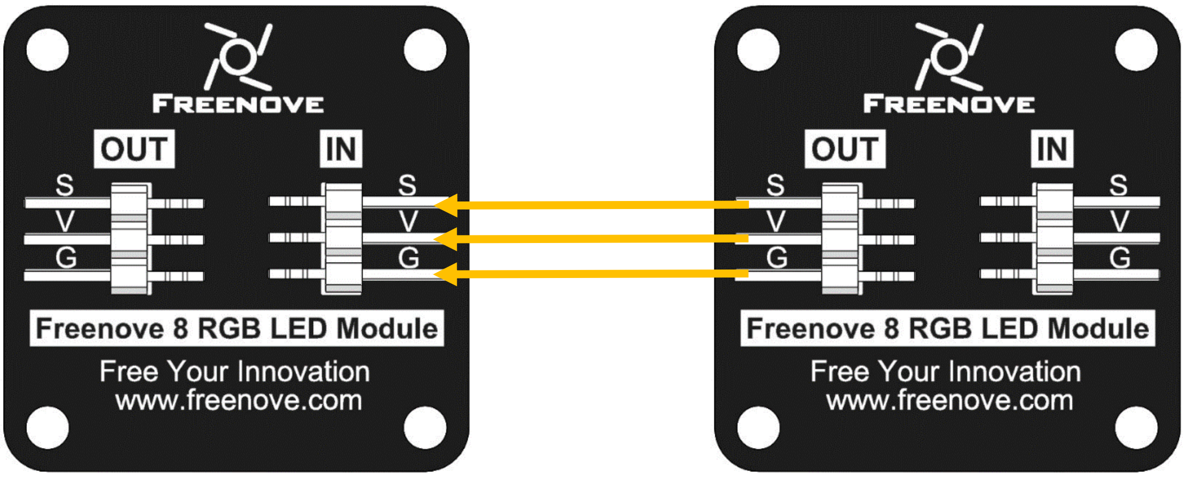
Component knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control the eight LEDs on the module. As shown below:



And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In such way, you can use one data pin to control 8, 16, 32 … LEDs.

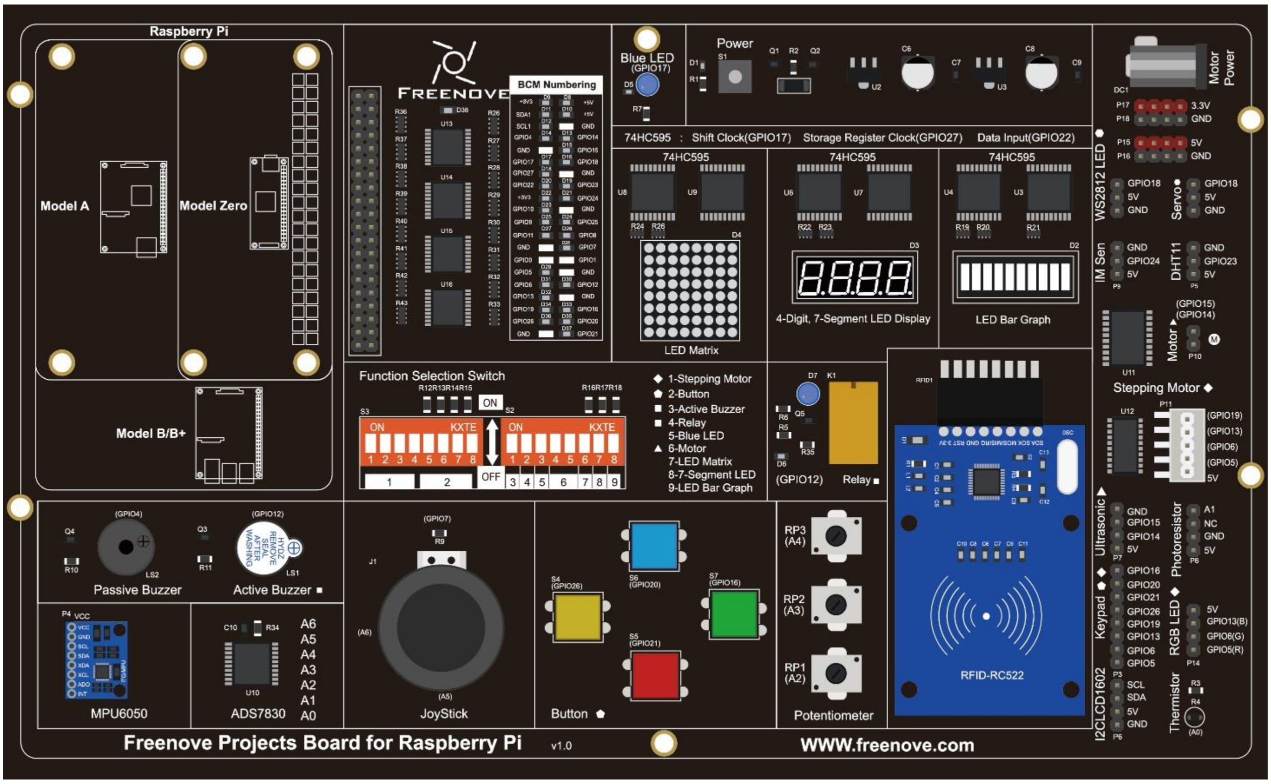


Pin description:

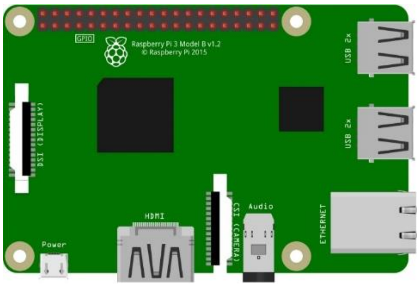
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

Component List

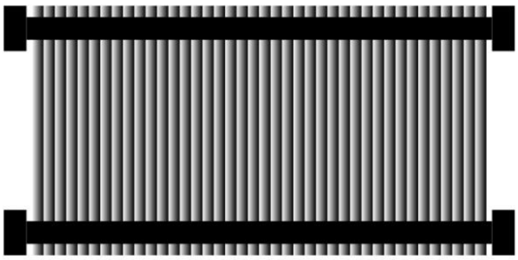
Freenove Projects Board for Raspberry Pi



Raspberry Pi

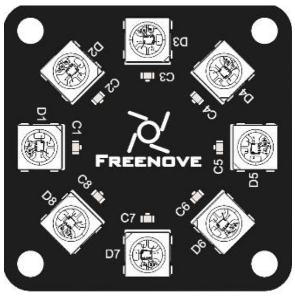


GPIO Ribbon Cable



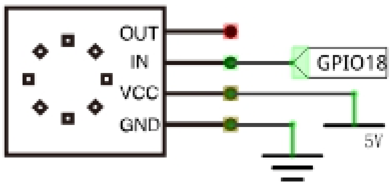
Jumper Wire

Freenove 8 RGB LED Module

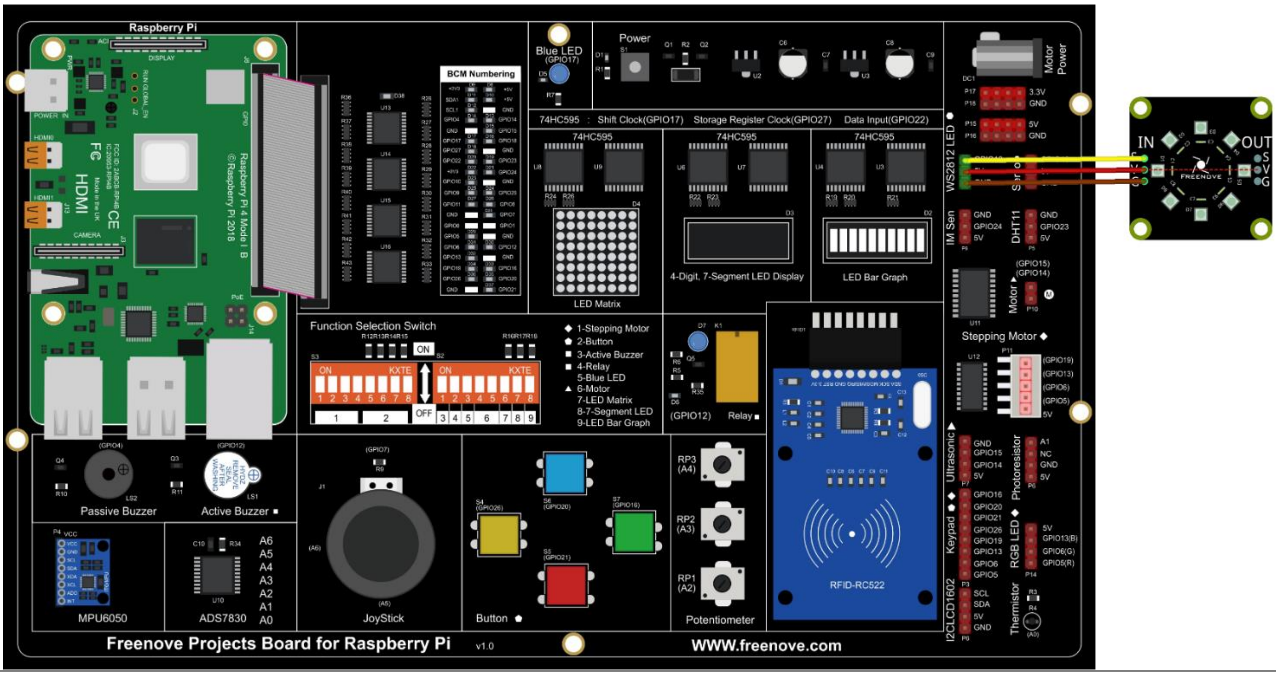


Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 15.1 Ledpixel

Before running C code, please install WS281X library.

1. Enter the directory where the library locates:

```
cd ~/Freenove_Kit/Libs/C-Libs/libWS281X
```

2. Run the program

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

The installation is completed as shown in the figure below

```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/15_1_Ledpixel $ cd ~/Freenove_Kit/Libs
/C-Libs/libWS281X
pi@raspberrypi:~/Freenove_Kit/Libs/C-Libs/libWS281X $ sudo sh ./build.sh
build completed!
```

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

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

3. Use cd command to enter 15_1_Ledpixel directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/15_1_Ledpixel
```

4. Use following command to compile " Ledpixel.cpp" and generate executable file "Ledpixel".

```
sudo g++ Ledpixel.cpp -o Ledpixel -lwiringPi -lWS281X
```

5. Run the generated file " Ledpixel".

```
sudo ./Ledpixel
```

After the program runs, the LEDpixel will emit red, blue and green colors in turn like flowing water.

The following is the program code:

```
#include <wiringPi.h>
#include "Freenove_WS2812_Lib_for_Raspberry_Pi.hpp"
Freenove_WS2812 *a;
int constrain(int value,int min,int max){
    if (value>max){
        return max;
    }
    else if (value<min){
        return min;
    }
    else {
        return value;
    }
}
int main(){
    printf("Program is starting ...\n");
    int i;
    a= new Freenove_WS2812(18,8,GRB);//pin led_count type
    a->set_Led_Brightness(50);
    for(i=0;i<8;i++){
        a->set_Led_Color(i,255,0,0);
        a->show();
        delay(100);
        }
    for(i=0;i<8;i++){
        a->set_Led_Color(i,0,255,0);
        a->show();
        delay(100);
        }
    for(i=0;i<8;i++){
        a->set_Led_Color(i,0,0,255);
        a->show();
```

```
        delay(100);
        }
    a->clear();
    return 0;
}
```

Include "Freenove_WS2812_Lib_for_Raspberry_Pi.hpp"

```
#include "Freenove_WS2812_Lib_for_Raspberry_Pi.hpp"
```

Create the object of the class and set the brightness to 50%. The eight LEDs will then light up red, green and blue in turn.

```
int main(){
    printf("Program is starting ...\n");
    int i;
    a= new Freenove_WS2812(18,8,GRB);//pin led_count type
    a->set_Led_Brightness(50);
    for(i=0;i<8;i++){
        a->set_Led_Color(i,255,0,0);
        a->show();
        delay(100);
        }
    for(i=0;i<8;i++){
        a->set_Led_Color(i,0,255,0);
        a->show();
        delay(100);
        }
    for(i=0;i<8;i++){
        a->set_Led_Color(i,0,0,255);
        a->show();
        delay(100);
        }
    a->clear();
    return 0;
}
```

Python Code 15.1 Ledpixel

Before running python code, please install WS281X library first.

1. Enter the following command to install.

```
sudo pip3 install rpi_ws281x
```

The installation is completed as shown in the figure below.

```
pi@raspberrypi:~ $ sudo pip3 install rpi_ws281x
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting rpi_ws281x
  Downloading https://files.pythonhosted.org/packages/66/03/bda698d5429b918e1ef5
acfb3d745d3a12c8fec8078925f10e571aa0a8e2/rpi_ws281x-4.2.5-cp37-cp37m-linux_armv7
l.whl (115kB)
    100% |████████████████████████████████| 122kB 1.7MB/s
Installing collected packages: rpi-ws281x
Successfully installed rpi-ws281x-4.2.5
```

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

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

1. Use cd command to enter 15_1_Ledpixel directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/15_1_Ledpixel
```

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

```
sudo python Led.py
```

After the program runs, the LEDpixel will emit red, green and blue colors in turn like flowing water.

If you want to run Led.py via thonny, you need use `sudo thonny Led.py` to open it first.

The following is the program code:

```
import time
from rpi_ws281x import *
# LED strip configuration:
LED_COUNT      = 8      # Number of LED pixels.
LED_PIN        = 18      # GPIO pin connected to the pixels (18 uses PWM!).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN transistor level shift)
LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53
# Define functions which animate LEDs in various ways.
class Led:
    def __init__(self):
        #Control the sending order of color data
        self.ORDER = "RGB"
        # Create NeoPixel object with appropriate configuration.
        self.strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT,
LED_BRIGHTNESS, LED_CHANNEL)
        # Intialize the library (must be called once before other functions).
        self.strip.begin()
        #self.strip.setPixelColor(i, color)
        #self.strip.show()
```

```


led=Led()
# Main program logic follows:
if __name__ == '__main__':
    print ('Program is starting ... ')
    col=[Color(255,0,0),Color(0,255,0),Color(0,0,255)]
    try:
        while True:
            for c in range(3):
                for i in range(8):
                    led.strip.setPixelColor(i,col[c])
                    time.sleep(0.1)
                    led.strip.show()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
executed.
        for i in range(8):
            led.strip.setPixelColor(i, Color(0,0,0))
        led.strip.show()
```

Import rpi_ws281x modile. Set the number, pins and brightness of the LED.

```
from rpi_ws281x import *
# LED strip configuration:
LED_COUNT      = 8      # Number of LED pixels.
LED_PIN        = 18      # GPIO pin connected to the pixels (18 uses PWM!).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN transistor level shift)
LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53
```

Define LED class.

```
class Led:
    def __init__(self):
        #Control the sending order of color data
        self.ORDER = "RGB"
        # Create NeoPixel object with appropriate configuration.
        self.strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT,
LED_BRIGHTNESS, LED_CHANNEL)
        # Intialize the library (must be called once before other functions).
        self.strip.begin()
        #self.strip.setPixelColor(i, color)
        #self.strip.show()
```

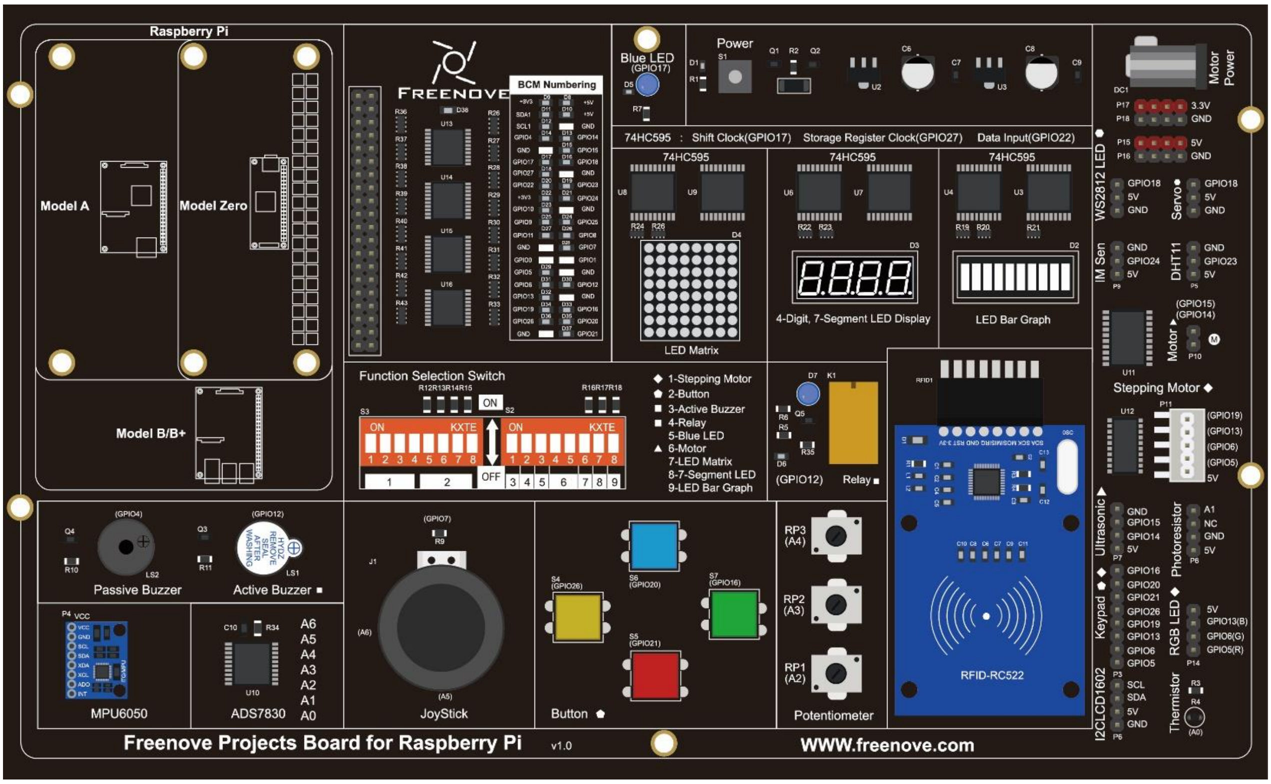
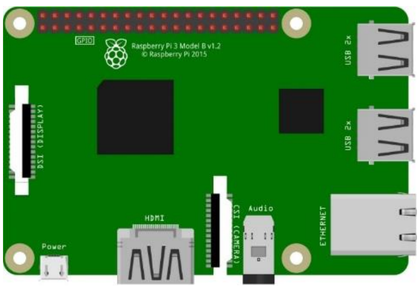
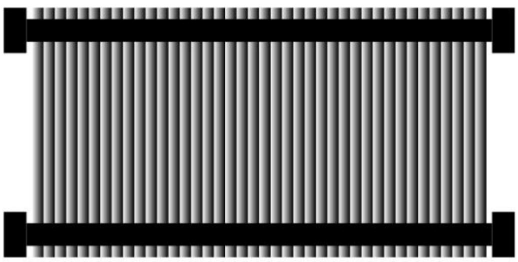
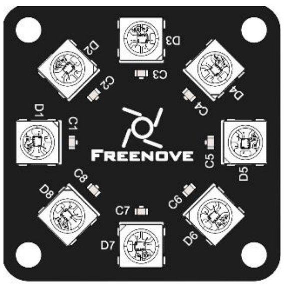
Light up the eight LEDs in red, green and blue in turn.

```
    col=[Color(255,0,0),Color(0,255,0),Color(0,0,255)]
    try:
        while True:
            for c in range(3):
                for i in range(8):
                    led.strip.setPixelColor(i,col[c])
                    time.sleep(0.1)
                    led.strip.show()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
executed.
        for i in range(8):
            led.strip.setPixelColor(i, Color(0,0,0))
        led.strip.show()
```

Project 15.2 Rainbow Light

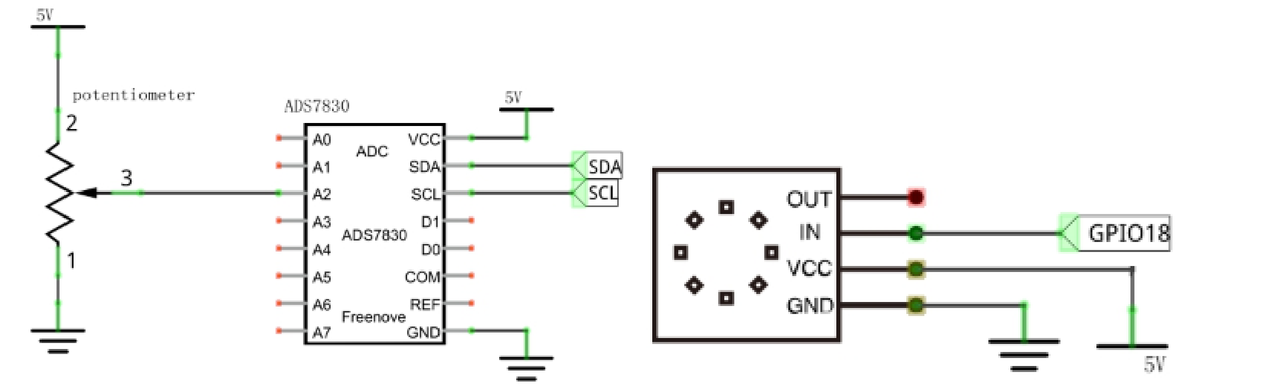
In this project, we will learn to control the LED module with a potentiometer.

Component List

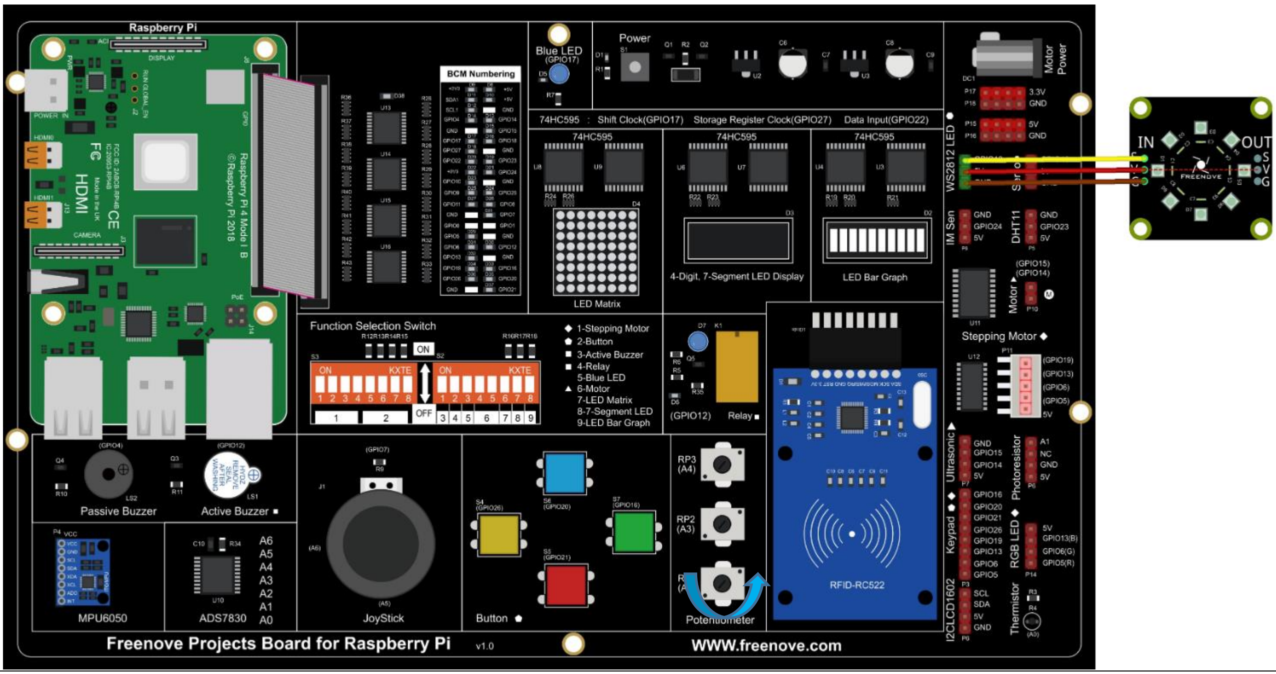
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>Freenove 8 RGB LED Module

</td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



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

Code

C Code 15.2 Rainbow Light

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

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

3. Use cd command to enter 15_2_RainbowLight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/15_2_RainbowLight
```

4. Use following command to compile " RainbowLight.cpp " and generate executable file " RainbowLight ".

```
sudo g++ RainbowLight.cpp -o RainbowLight -lwiringPi -lWS281X -lADCDevice
```

5. Run the generated file " RainbowLight ".

```
sudo ./RainbowLight
```

After running the program, you can change the color of the LED module by rotating the potentiometer.

The following is the program code:

```
#include <stdio.h>
#include <wiringPi.h>
#include <ADCDevice.hpp>
#include "Freenove_WS2812_Lib_for_Raspberry_Pi.hpp"

Freenove_WS2812 *led;
ADCDevice *adc;
int red,green,blue;

void HSL_RGB(int degree){
    degree=degree/360.0*255;
    if (degree < 85){
            red = 255 - degree * 3;
            green = degree * 3;
            blue = 0;
        }
    else if (degree < 170){
            degree = degree - 85;
            red = 0;
            green = 255 - degree * 3;
            blue = degree * 3;
        }
    else{
            degree = degree - 170;
            red = degree * 3;
            green = 0;
            blue = 255 - degree * 3;
        }
```

```
}

int main(){
    printf("Program is starting ...\n");
    adc = new ADCDevice();
    int i;
    led= new Freenove_WS2812(18,8,GRB);//pin led_count type
    led->set_Led_Brightness(50);

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


    while(1){
        for(i=0;i<8;i++){
            int degree = (int)(adc->analogRead(2)/255.0*360+i*45);    //read analog value of A0 pin
            if (degree > 360){
                degree=degree-360;
                }
            HSL_RGB(degree);
            led->set_Led_Color(i,red,green,blue);
            led->show();

        }
    }

    return 0;
}
```

This function converts HSL colors to RGB colors.

```
void HSL_RGB(int degree){
    degree=degree/360.0*255;
    if (degree < 85){
            red = 255 - degree * 3;
            green = degree * 3;
```

```
            blue = 0;
        }
    else if (degree < 170){
            degree = degree - 85;
            red = 0;
            green = 255 - degree * 3;
            blue = degree * 3;
        }
    else{
            degree = degree - 170;
            red = degree * 3;
            green = 0;
            blue = 255 - degree * 3;
        }
}
```

Read the ADC value of channel 2 in an infinite loop. Let the color of the eight LEDs change according to the value of the ADC.

```
while(1){
        for(i=0;i<8;i++){
            int degree = (int)(adc->analogRead(2)/255.0*360+i*45);    //read analog value of
A2 pin
            if (degree > 360){
                degree=degree-360;
                }
            HSL_RGB(degree);
            led->set_Led_Color(i,red,green,blue);
            led->show();

        }
    }
```

Python Code 15.2 Rainbow Light

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

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

6. Use cd command to enter 15.2 Rainbow Light directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/15_2_RainbowLight
```

7. Use python command to execute code " `Led.py` ".

```
sudo python Led.py
```

After running the program, you can change the color of the LED module by rotating the potentiometer.

The following is the program code:

```
import time
from rpi_ws281x import *
from ADCDevice import *
# LED strip configuration:
LED_COUNT      = 8      # Number of LED pixels.
LED_PIN        = 18      # GPIO pin connected to the pixels (18 uses PWM!).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN transistor level shift)
LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53
# Define functions which animate LEDs in various ways.
class Led:
    def __init__(self):
        #Control the sending order of color data
        self.ORDER = "RGB"
        # Create NeoPixel object with appropriate configuration.
        self.strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT,
LED_BRIGHTNESS, LED_CHANNEL)
        # Intialize the library (must be called once before other functions).
        self.strip.begin()

        self.adc = ADCDevice(0x48) # Define an ADCDevice class object
        if(self.adc.detectI2C(0x48)):
            self.adc = ADS7830(0x48)
        else:
            print("No correct I2C address found, \n"
            "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
            "Program Exit. \n");
            exit(-1)


    def HSL_RGB(self,degree):
```

```
        degree=degree/360*255
        if degree < 85:
            red = 255 - degree * 3
            green = degree * 3
            blue = 0
        elif degree < 170:
            degree = degree - 85
            red = 0
            green = 255 - degree * 3
            blue = degree * 3
        else:
            degree = degree - 170
            red = degree * 3
            green = 0
            blue = 255 - degree * 3
        return int(red),int(green),int(blue)
led=Led()
# Main program logic follows:
if __name__ == '__main__':
    print ('Program is starting ... ')
    try:
        while True:
                for i in range(8):
                    value = round(led.adc.analogRead(2) / 255.0 * 360+i*45)    # read the ADC value of channel 2
                    if value > 360 :
                        value = value-360
                    red,green,blue=led.HSL_RGB(value)
                    led.strip.setPixelColor(i, Color(red,green,blue))
                time.sleep(0.1)
                led.strip.show()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
        led.adc.close()
        for i in range(8):
            led.strip.setPixelColor(i, Color(0,0,0))
        led.strip.show()
```

This function converts HSL colors to RGB colors.

```
    def HSL_RGB(self,degree):
        degree=degree/360*255
        if degree < 85:
            red = 255 - degree * 3
            green = degree * 3
```

```
            blue = 0
        elif degree < 170:
            degree = degree - 85
            red = 0
            green = 255 - degree * 3
            blue = degree * 3
        else:
            degree = degree - 170
            red = degree * 3
            green = 0
            blue = 255 - degree * 3
        return int(red),int(green),int(blue)
```

Read the ADC value of channel 2 in an infinite loop. Let the color of the eight LEDs change according to the value of the ADC.

```
        while True:
                for i in range(8):
                    value = round(led.adc.analogRead(2) / 255.0 * 360+i*45)     # read the ADC
value of channel 2
                    if value > 360 :
                        value = value-360
                    red,green,blue=led.HSL_RGB(value)
                    led.strip.setPixelColor(i, Color(red,green,blue))
                time.sleep(0.1)
                led.strip.show()
```

Finally, in the loop of main function, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

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

Chapter 16 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.

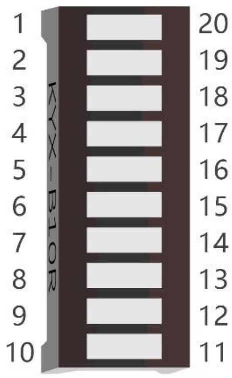
Project 16.1 Flowing Water Light

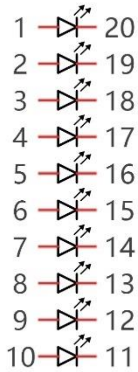
Now let us learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

Component knowledge

Bar Graph LED

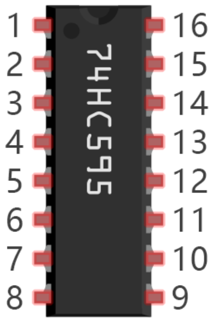
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.

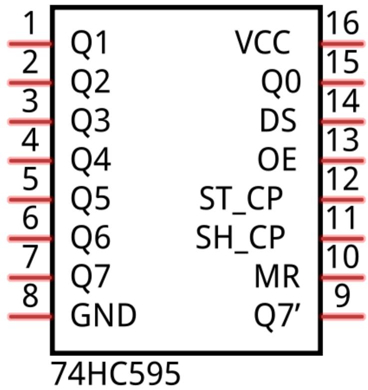




74HC595

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.



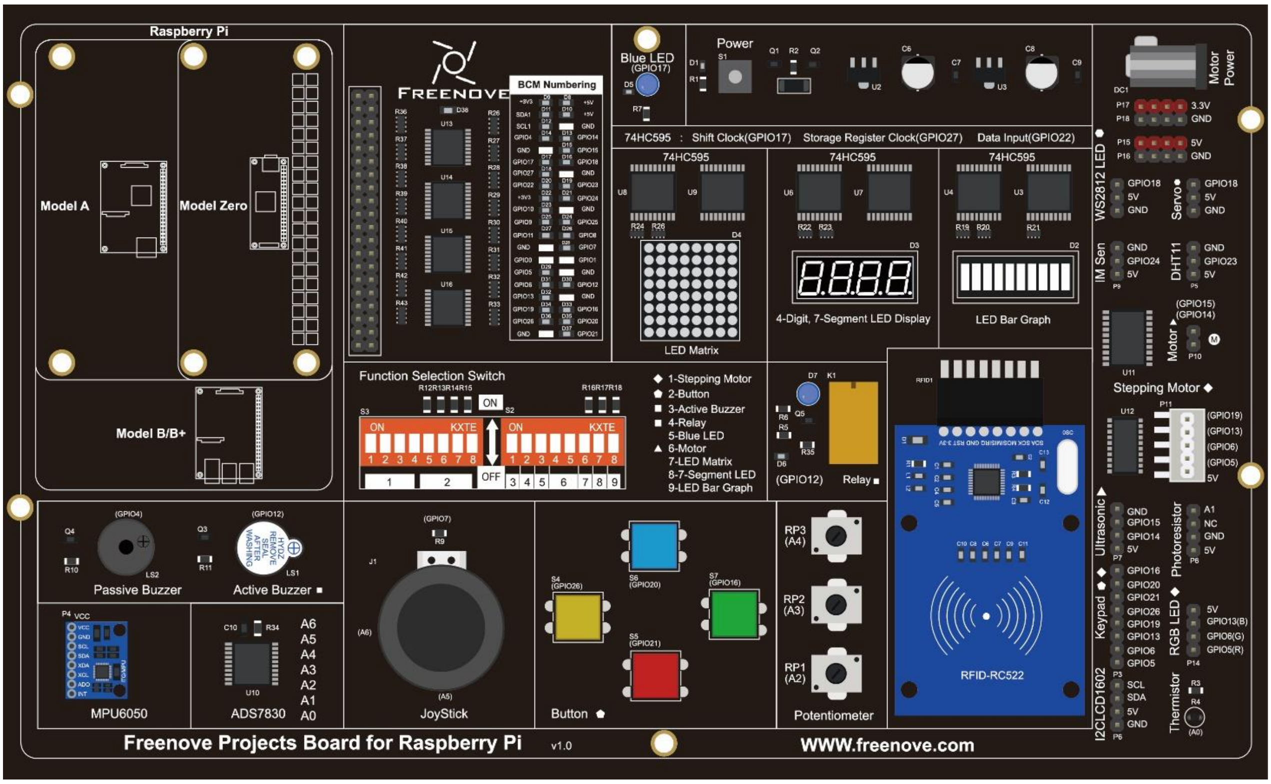
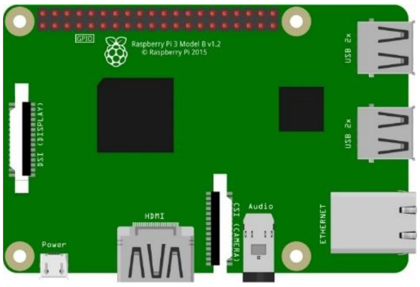
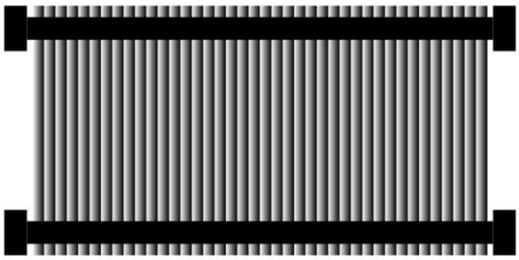


The ports of the 74HC595 chip are described as follows:

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.

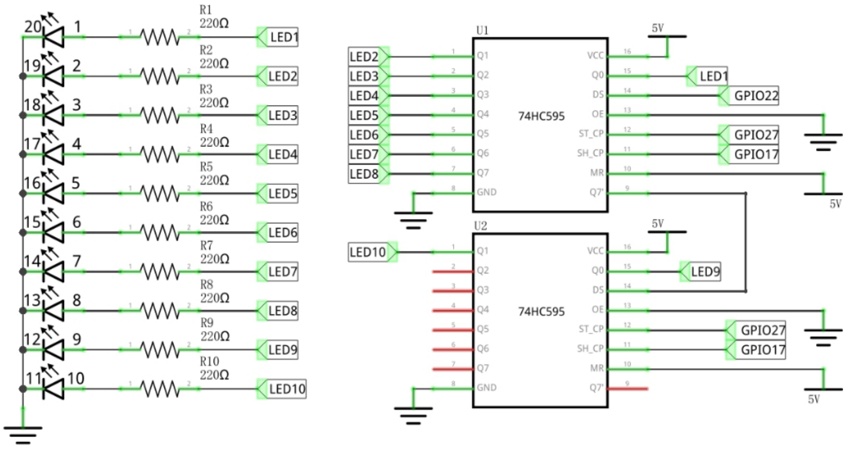
For more details, please refer to the datasheet on the 74HC595 chip.

Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Bar Graph LED
</td><td></td></tr>
</table>

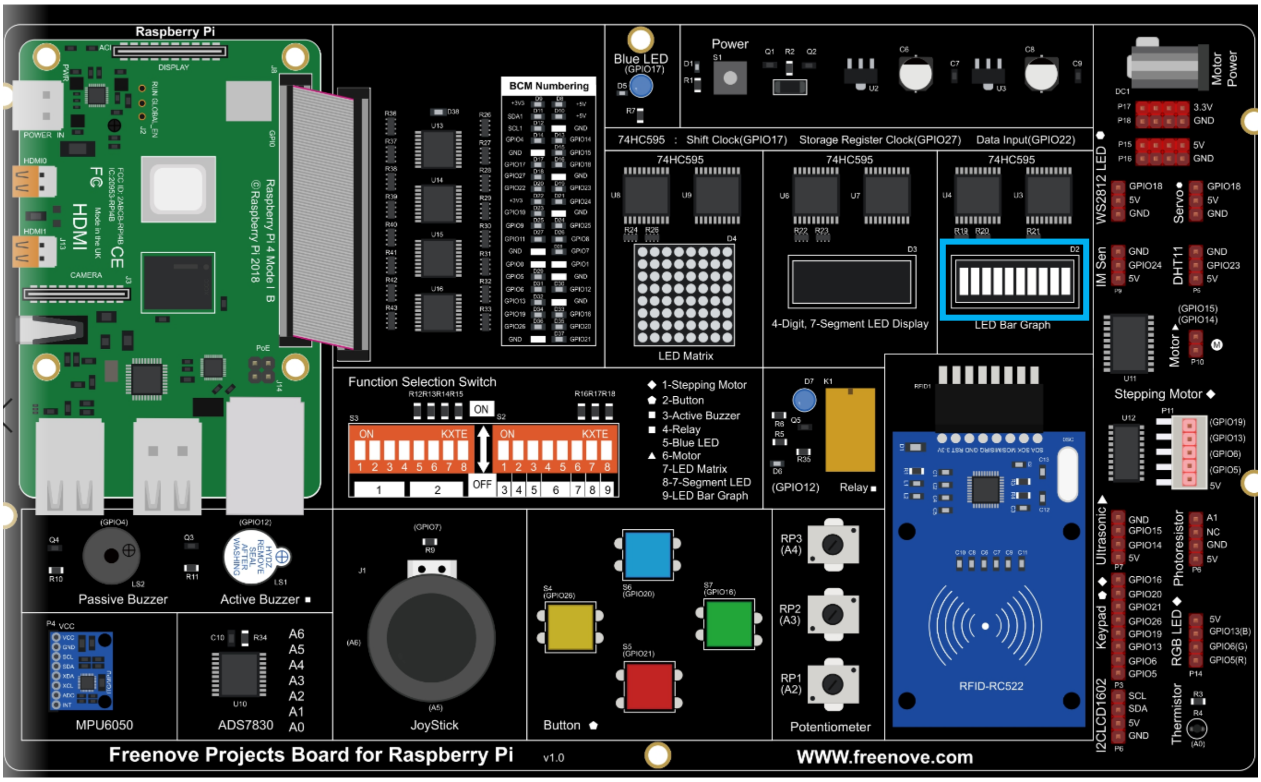
Circuit

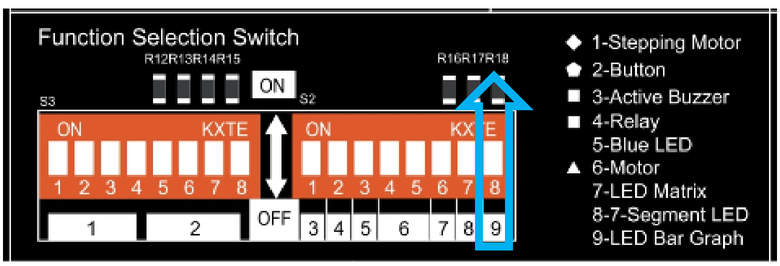
Schematic diagram



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

Code

In this project we will make a flowing water light with a 74HC595 chip to learn about its functions.

C Code 16.1 LightWater02

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

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

1. Use cd command to enter 16_FlowingLight02 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/16_FlowingLight02
```

2. Use following command to compile "FlowingLight02.c" and generate executable file "FlowingLight02".

```
gcc FlowingLight02.c -o FlowingLight02 -lwiringPi
```

3. Then run the generated file "FlowingLight02".

```
sudo ./FlowingLight02
```

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

The following is the program code:

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

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

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

int main(void)
```

```
{
    int i;
    unsigned long x;

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

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        x=0x0001;
        for(i=0;i<10;i++){
            digitalWrite(latchPin,LOW);         // Output low level to latchPin
            _shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
            digitalWrite(latchPin,HIGH);    //Output high level to latchPin, and 74HC595 will
update the data to the parallel output port.
            x<<=1;      //make the variable move one bit to left once, then the bright LED
move one step to the left once.
            delay(100);
        }
        x=0x0200;
        for(i=0;i<10;i++){
            digitalWrite(latchPin,LOW);
            _shiftOut(dataPin,clockPin,LSBFIRST,x);
            digitalWrite(latchPin,HIGH);
            x>>=1;
            delay(100);
        }
    }
    return 0;
}
```

In the code, we configure three pins to control the 74HC595 chip and define a one-byte variable to control the state of the 10 LEDs (in the Bar Graph LED Module) through the 10 bits of the variable. The LEDs light ON when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED ON.

```
x=0x0001;
```

In the "while" loop of main function, use two loops to send x to 74HC595 output pin to control the LED. In one cycle, x will shift one bit to the LEFT in one cycle, then when data of x is sent to 74HC595, the LED that is turned ON will move one bit to the LEFT once.

```
        for(i=0;i<10;i++){
```

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

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

<< operator

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

byte x = 1 << 1;

←	0	0	0	0	0	0	0	1	←	0

The result of x is 2 (binary 00000010) .

0	0	0	0	0	0	1	0

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

byte x = 1 >> 1;

0	→	0	0	0	0	0	0	0	1	→

The result of x is 0 (00000000) .

0	0	0	0	0	0	0	0

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

About shift function

void _shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val) ;

This is used to shift a 10-bit data value out with the data being sent out on dPin and the clock being sent out on the cPin. order is as above. Data is clocked out on the rising or falling edge - ie. dPin is set, then cPin is taken high then low - repeated for the 10 bits.

Python Code 16.1 LightWater02

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

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

1. Use cd command to enter 16_FlowingLight02 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/16_FlowingLight02
```

2. Use python command to execute Python code “FlowingLight02.py”.

```
python FlowingLight02.py
```

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

The following is the program code:

```
import RPi.GPIO as GPIO
import time
# Defines the data bit that is transmitted preferentially in the shiftOut function.
LSBFIRST = 1
MSBFIRST = 2
# define the pins for 74HC595
dataPin   = 15      # DS Pin of 74HC595(Pin14)
latchPin  = 13      # ST_CP Pin of 74HC595(Pin12)
clockPin = 11       # CH_CP Pin of 74HC595(Pin11)

def setup():
    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(dataPin, GPIO.OUT) # set pin to OUTPUT mode
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

# shiftOut function, use bit serial transmission.
def shiftOut(dPin,cPin,order,val):
    for i in range(0,10):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        x=0x0001
        for i in range(0,10):
            GPIO.output(latchPin,GPIO.LOW)  # Output low level to latchPin
            shiftOut(dataPin,clockPin,LSBFIRST,x) # Send serial data to 74HC595
```

```
            GPIO.output(latchPin,GPIO.HIGH) # Output high level to latchPin, and 74HC595 will
update the data to the parallel output port.
            x<<=1 # make the variable move one bit to left once, then the bright LED move one
step to the left once.
            time.sleep(0.1)
        print(hex(x))
        x=0x0200
        for i in range(0,10):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,LSBFIRST,x)
            GPIO.output(latchPin,GPIO.HIGH)
            x>>=1
            time.sleep(0.1)
        print(hex(x),int(x))


def destroy():
    GPIO.cleanup()

if __name__ == '__main__': # Program entrance
    print ('Program is starting...' )
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

In the code, we define a shiftOut() function, which is used to output values with bits in order, where the dPin for the data pin, cPin for the clock and order for the priority bit flag (high or low). This function conforms to the operational modes of the 74HC595. LSBFIRST and MSBFIRST are two different flow directions.

```
def shiftOut(dPin,cPin,order,val):
    for i in range(0,10):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);
```

In the loop() function, we use two loops to achieve the action goal. First, define a variable x=0x0001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then an LED turns ON. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED that turns ON will be shifted. Repeat the operation, over and over and the effect of a flowing water light will be visible. If the direction of the shift operation for x is different, the flowing direction is different.

```
def loop():
    while True:
        x=0x0001
        for i in range(0,10):
            GPIO.output(latchPin,GPIO.LOW)  # Output low level to latchPin
            shiftOut(dataPin,clockPin,LSBFIRST,x) # Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH) # Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
            x<<=1 # make the variable move one bit to left once, then the bright LED move one step to the left once.
            time.sleep(0.1)
        print(hex(x))
        x=0x0200
        for i in range(0,10):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,LSBFIRST,x)
            GPIO.output(latchPin,GPIO.HIGH)
            x>>=1
            time.sleep(0.1)
        print(hex(x),int(x))
```

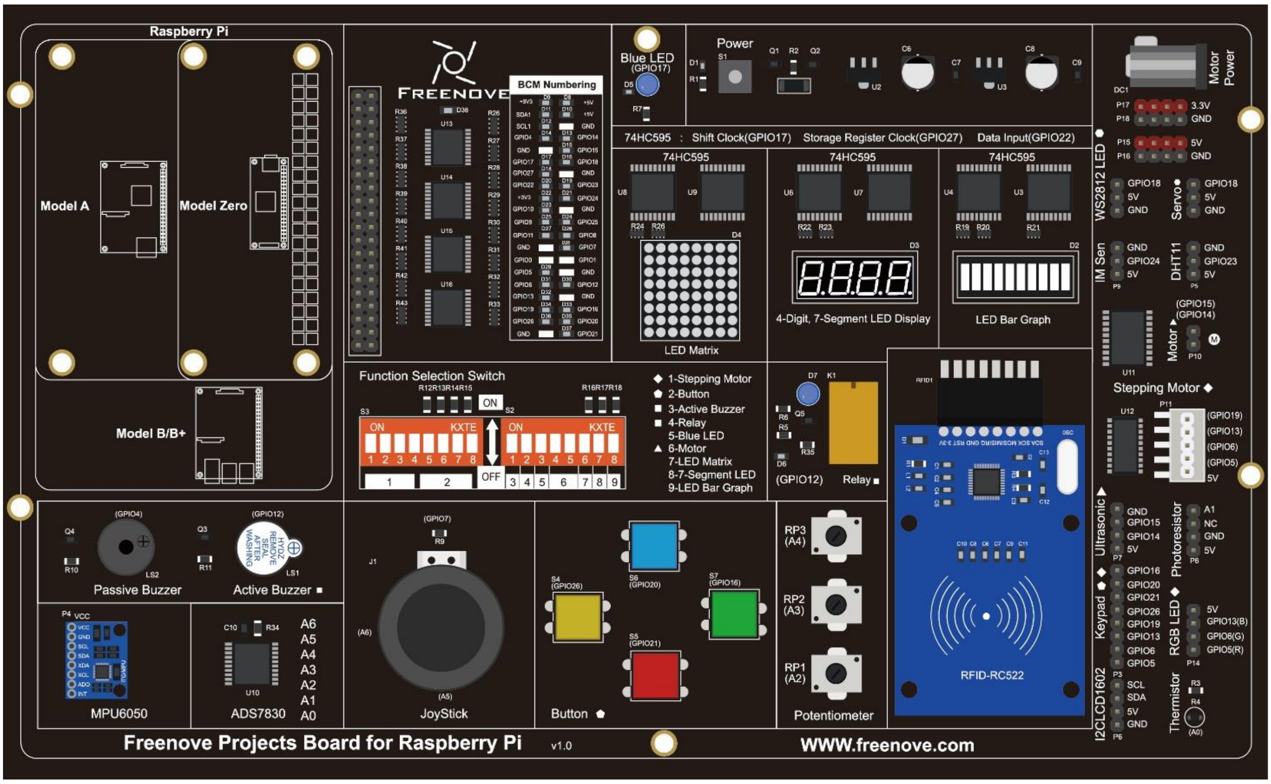
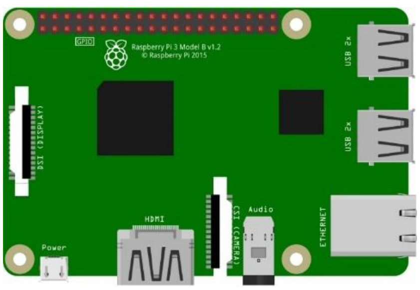
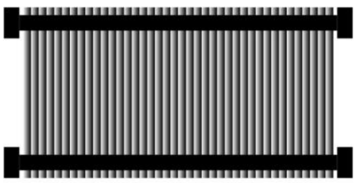
Chapter 17 74HC595 & 4-Digit 7-Segment Display

In this chapter, we will introduce the 7-Segment Display.

Project 17.1 4-Digit 7-Segment Display

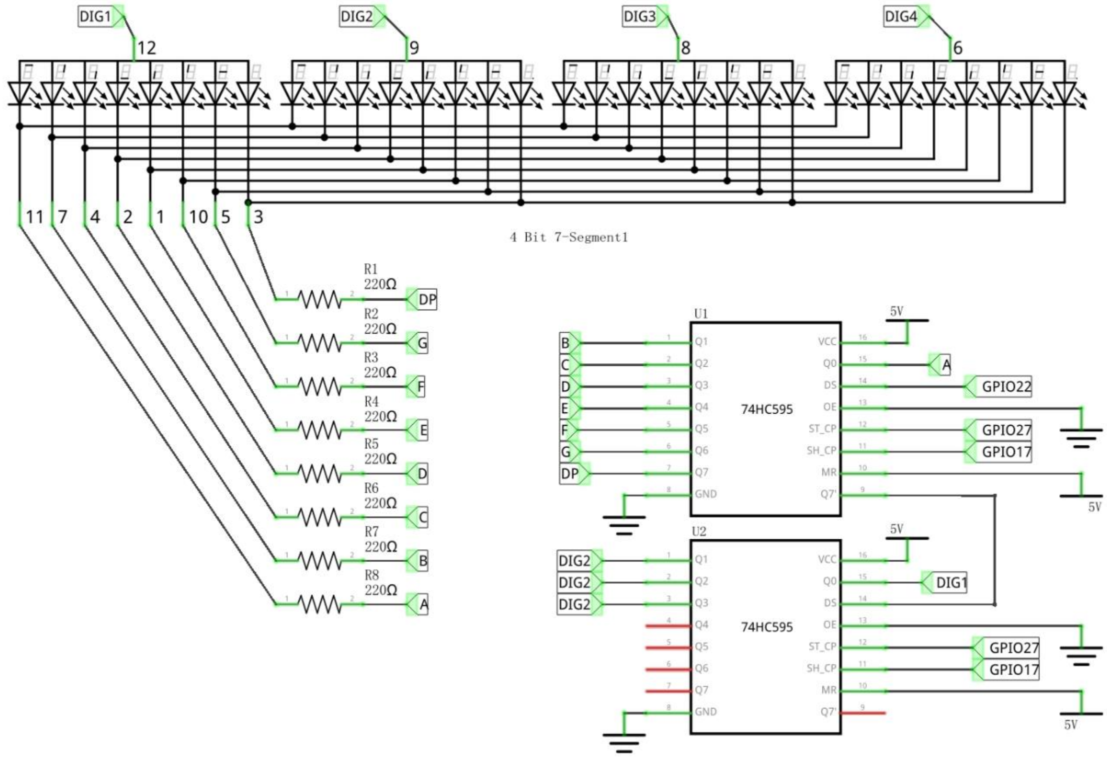
We will use a 74HC595 IC Chip to control a 4-Digit 7-Segment Display and make it display sixteen decimal characters "0" to "F".

Component List

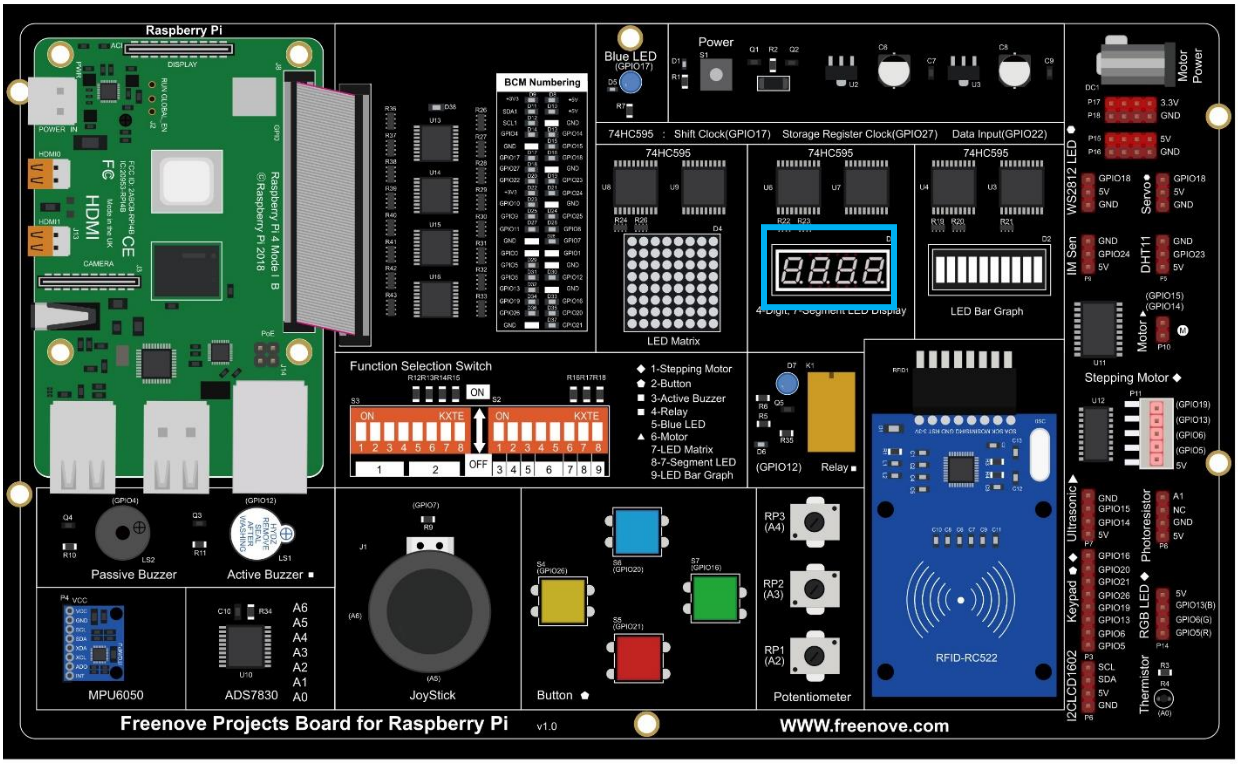
<table>
<tr><td colspan="3">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td><td>4-Digit 7-Segment Display
</td></tr>
</table>

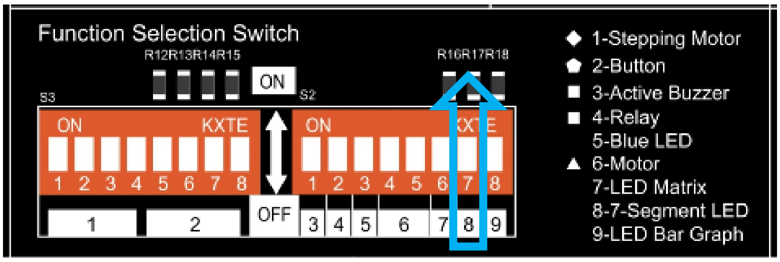
Circuit

Schematic diagram



Hardware connection.



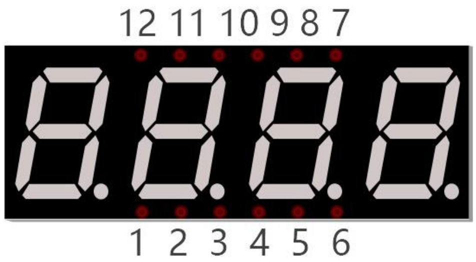


If you have any concerns, please send an email to: support@freenove.com

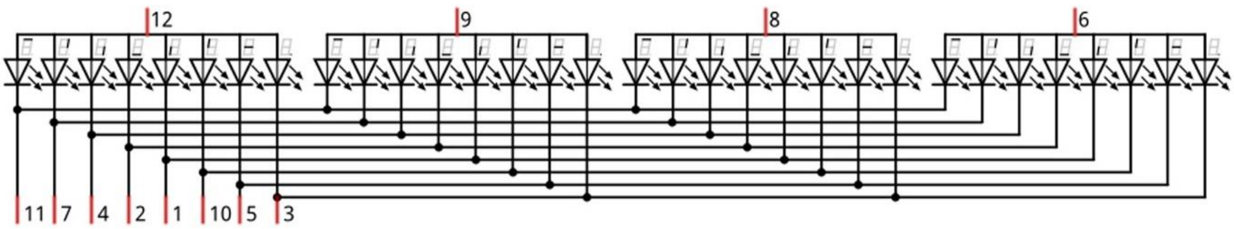
Component knowledge

4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-Segment Displays into one module, therefore it can display more characters. All of the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



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



Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is unperceivable to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

Code

This code uses a 74HC595 IC Chip to control the 4-Digit 7-Segment Display. The use of the 74HC595 IC Chip is generally the same throughout this Tutorial. We need code to display the characters “0” to “F” one character at a time, and then output to display them with the 74HC595 IC Chip.

C Code 17.1 SevenSegmentDisplay

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

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

1. Use cd command to enter 17_1_SevenSegmentDisplay directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/17_1_SevenSegmentDisplay
```

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

```
gcc SevenSegmentDisplay.c -o SevenSegmentDisplay -lwiringPi
```

3. Then run the generated file “SevenSegmentDisplay”.

```
sudo ./SevenSegmentDisplay
```

After the program runs, the 4-Digit 7-Segment Display starts to display the characters “0” to “F” in succession.

The following is the program code:

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

#define   dataPin   3   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin  0    //CH_CP Pin of 74HC595(Pin11)
//encoding for character 0-F of common anode SevenSegmentDisplay.
unsigned long
num[]={0xffc0,0xfff9,0xffa4,0xffb0,0xff99,0xff92,0xff82,0xfff8,0xff80,0xff90,0xff88,0xff83,0xf
fc6,0xffa1,0xff86,0xff8e};

void _shiftOut(int dPin,int cPin,int order,int val){
     int i;
    for(i = 0; i < 16; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {
            digitalWrite(dPin,((0x8000&(val<<i)) == 0x8000) ? HIGH : LOW);
            delayMicroseconds(10);
        }
```

```
            digitalWrite(cPin,HIGH);
            delayMicroseconds(10);
        }
}

int main(void)
{
    int i;

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

    wiringPiSetup();

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

First, we need to create encoding for characters “0” to “F” in the array.

```
unsigned long
num[]={0xffc0,0xfff9,0xffa4,0xffb0,0xff99,0xff92,0xff82,0xfff8,0xff80,0xff90,0xff88,0xff83,0xffc6
,0xffa1,0xff86,0xff8e};
```

In the “for” loop of loop() function, use the 74HC595 IC Chip to output contents of array “num” successively. SevenSegmentDisplay can then correctly display the corresponding characters.

```
while(1){
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            _shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the highest
level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
    }
```

Python Code 17.1 SevenSegmentDisplay

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

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

1. Use cd command to enter 17_1_SevenSegmentDisplay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/17_1_SevenSegmentDisplay
```

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

```
python SevenSegmentDisplay.py
```

After the program runs, the 4-Digit 7-Segment Display starts to display the characters “0” to “F” in succession.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LSBFIRST = 1
MSBFIRST = 2
# define the pins for 74HC595
dataPin   = 15      # DS Pin of 74HC595(Pin14)
latchPin  = 13      # ST_CP Pin of 74HC595(Pin12)
clockPin = 11       # CH_CP Pin of 74HC595(Pin11)
# SevenSegmentDisplay display the character "0"- "F" successively
num =
[0xffc0,0xfff9,0xffa4,0xffb0,0xff99,0xff92,0xff82,0xfff8,0xff80,0xff90,0xff88,0xff83,0xffc6,0x
ffa1,0xff86,0xff8e]

def setup():
    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,16):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x8000&(val<<i)==0x8000) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
```

```
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])  # Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)

        '''for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i]&0x7f) # Use "&0x7f" to display the
decimal point.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)'''


def destroy():
    GPIO.cleanup()

if __name__ == '__main__': # Program entrance
    print ('Program is starting...' )
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

First, we need to create encoding for characters “0” to “F” in the array.

```
num=[0xffc0,0xfff9,0xffa4,0xffb0,0xff99,0xff92,0xff82,0xfff8,0xff80,0xff90,0xff88,0xff83,0xffc6,0
xffa1,0xff86,0xff8e]
```

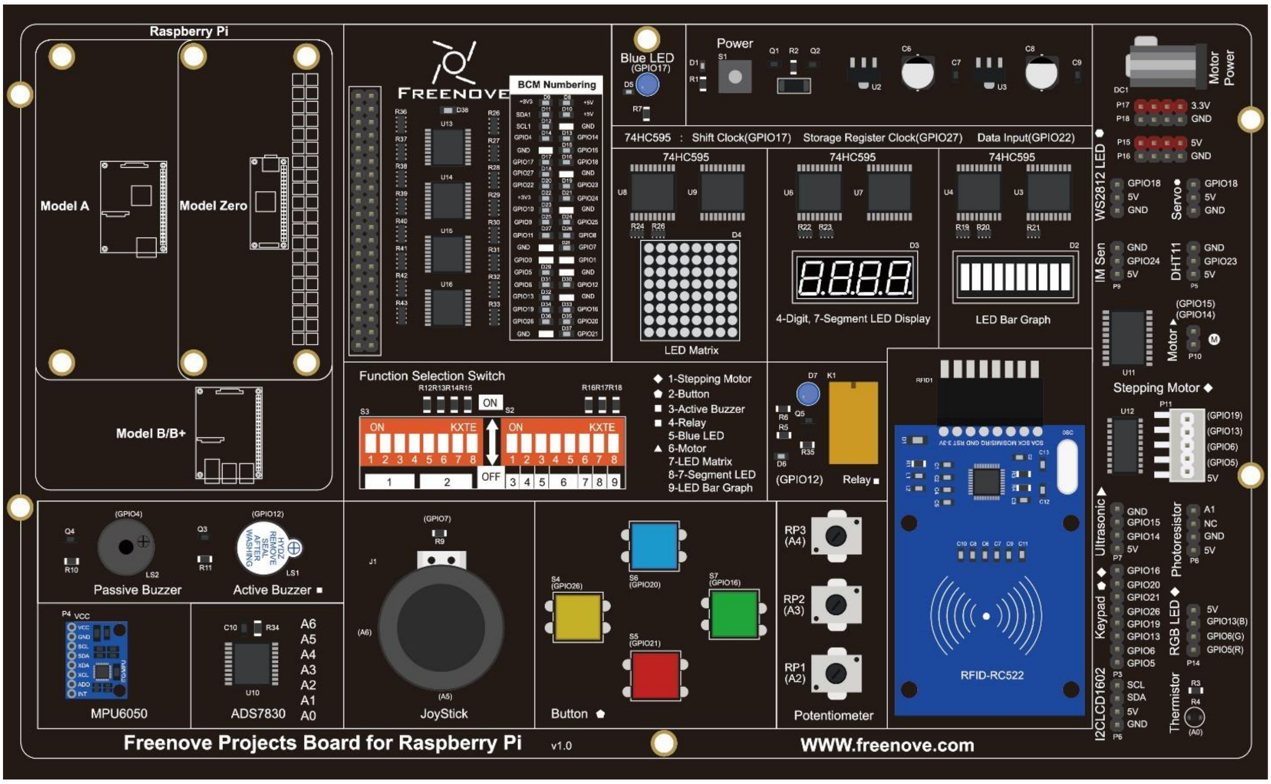
In the “for” loop of loop() function, use the 74HC595 IC Chip to output contents of array “num” successively. SevenSegmentDisplay can then correctly display the corresponding characters.

```
while True:
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])  # Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
```

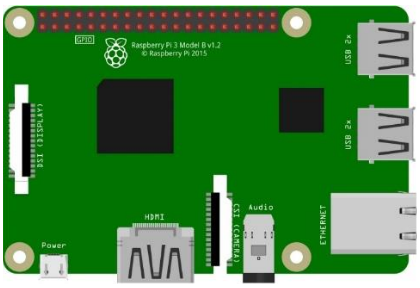
Project 17.2 4-Digit 7-Segment Display

Component List

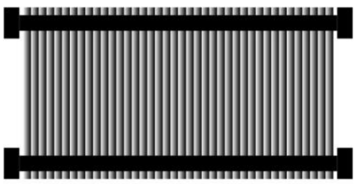
Freenove Projects Board for Raspberry Pi



Raspberry Pi



GPIO Ribbon Cable



4-Digit 7-Segment Display

Circuit

Schematic diagram

The same as that of 17.1

Hardware connection

The same as that of 17.1

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

Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

C Code 17.2 StopWatch

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

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

1. Use cd command to enter 17_2_StopWatch directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/17_2_StopWatch
```

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

```
gcc StopWatch.c -o StopWatch -lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./StopWatch
```

After the program runs, the 4-Digit 7-Segment Display starts displaying a four-digit number dynamically, and the numeric value of this number will increase by plus 1 each second thereafter.

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define     dataPin     3   //DS Pin of 74HC595(Pin14)
#define     latchPin    2   //ST_CP Pin of 74HC595(Pin12)
#define     clockPin    0    //CH_CP Pin of 74HC595(Pin11)
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;    //variable counter,the number will be displayed by 7-segment display
//Open one of the 7-segment display and close the remaining three, the parameter digit is 
optional for 1,2,4,8
unsigned long selectDigit(unsigned long digit){
    if (digit==0x01){
        return (0x08<<8);
        }
    else if (digit==0x02){
        return (0x04<<8);
        }
    else if (digit==0x04){
        return (0x02<<8);
        }
    else if (digit==0x08){
        return (0x01<<8);
```

```
        }
    else{
        return (0xf0<<8);
        }

}
void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 16; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(1);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x8000&(val<<i)) == 0x8000) ? HIGH : LOW);
            delayMicroseconds(1);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(1);
    }
}
void outData(unsigned long data){      //function used to output data for 74HC595
    digitalWrite(latchPin,LOW);
    _shiftOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
void display(int dec){  //display function for 7-segment display
    int delays = 1;
    unsigned long  digit;
    outData(0xffff);
    digit=selectDigit(0x01);      //select the first, and display the single digit
    outData(num[dec%10]|digit);
    delay(delays);          //display duration

    outData(0xffff);
    digit=selectDigit(0x02);      //select the second, and display the tens digit
    outData(num[dec%100/10]|digit);
    delay(delays);

    outData(0xffff);
    digit=selectDigit(0x04);      //select the third, and display the hundreds digit
    outData(num[dec%1000/100]|digit);
    delay(delays);
```

```

    outData(0xffff);
    digit=selectDigit(0x08);     //select the fourth, and display the thousands digit
    outData(num[dec%10000/1000]|digit);
    delay(delays);
}
void timer(int  sig){       //Timer function
    if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus 1, and update
the number displayed by 7-segment display
        counter ++;
        alarm(1);           //set the next timer time
        printf("counter : %d \n",counter);
    }
}
int main(void)
{
    int i;

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

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);        //set the pin connected to74HC595 for output mode
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);

    signal(SIGALRM,timer);  //configure the timer
    alarm(1);               //set the time of timer to 1s
    while(1){
        display(counter);   //display the number counter
    }
    return 0;
}
```

First, we define the pin of the 74HC595 IC Chip and the 7-Segment Display Common Anode, use character encoding and a variable "counter" to enable the counter to be visible on the 7-Segment Display.

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

Subfunction **selectDigit** (int digit) function is used to open one of the 7-Segment Displays while closing the other 7-Segment Displays, where the parameter digit value can be 1,2,4,8. Using "|" can open a number of a 7-Segment Display.

```
unsigned long selectDigit(unsigned long digit){
    if (digit==0x01){
        return (0x08<<8);
        }
    else if (digit==0x02){
        return (0x04<<8);
        }
    else if (digit==0x04){
        return (0x02<<8);
        }
    else if (digit==0x08){
        return (0x01<<8);
        }
    else{
        return (0xf0<<8);
        }
}
```

Subfunction **outData** (int8_t data) is used to make the 74HC595 IC Chip output a 16-bit data immediately.

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

Subfunction **display** (int dec) is used to make a 4-Digit 7-Segment Display a 4-bit integer. First open the common end of first 7-Segment Display Digit and turn OFF the other three Digits, now it can be used as 1-Digit 7-Segment Display. The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively. Each digit will be displayed for a period by using **delay** (). The time in this code is very brief, so you will see digits all together. If the time is set long enough, you will see that every digit is displayed independently.

```
void display(int dec){  //display function for 7-segment display
     int delays = 1;
    unsigned long  digit;
     outData(0xffff);
    digit=selectDigit(0x01);      //select the first, and display the single digit
    outData(num[dec%10]|digit);
    delay(delays);          //display duration

    outData(0xffff);
    digit=selectDigit(0x02);      //select the second, and display the tens digit
    outData(num[dec%100/10]|digit);
    delay(delays);

    outData(0xffff);
    digit=selectDigit(0x04);      //select the third, and display the hundreds digit
    outData(num[dec%1000/100]|digit);
    delay(delays);

    outData(0xffff);
    digit=selectDigit(0x08);      //select the fourth, and display the thousands digit
    outData(num[dec%10000/1000]|digit);
    delay(delays);
}
```

Subfunction **timer** (int sig) is the timer function, which will set an alarm to signal. This function will be executed once at set time intervals. Accompanied by the execution, “1” will be added as the variable counter and then reset the time of timer to 1s.

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

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

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

In the while loop, make the digital display variable counter value "1". The value will change in function timer (), so the content displayed by the 7-Segment Display will change accordingly.

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

Python Code 17.2 StopWatch

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

1. Use cd command to enter 17_2_StopWatch directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/17_2_StopWatch
```

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

```
python StopWatch.py
```

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

The following is the program code:

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

LSBFIRST = 1
MSBFIRST = 2
# define the pins connect to 74HC595
dataPin   = 15      # DS Pin of 74HC595
latchPin  = 13      # ST_CP Pin of 74HC595
clockPin  = 11       # SH_CP Pin of 74HC595
num = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
counter = 0         # Variable counter, the number will be dislayed by 7-segment display
t = 0               # define the Timer object
def setup():
    GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
    GPIO.setup(dataPin, GPIO.OUT)       # Set pin mode to OUTPUT
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,16):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x8000&(val<<i)==0x8000) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH)

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

def selectDigit(digit): # Open one of the 7-segment display and close the remaining three, the
```

```
parameter digit is optional for 1,2,4,8
    if digit==0x01:
        return (0x08<<8)
    elif digit==0x02:
        return (0x04<<8)
    elif digit==0x04:
        return (0x02<<8)
    elif digit==0x08:
        return (0x01<<8)
    else:
        return (0xf0<<8)

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

    outData(0xffff)
    digit=selectDigit(0x02)   # Select the second, and display the tens digit
    outData(num[dec%100//10]|digit)
    time.sleep(0.003)

    outData(0xffff)
    digit=selectDigit(0x04)   # Select the third, and display the hundreds digit
    outData(num[dec%1000//100]|digit)
    time.sleep(0.003)

    outData(0xffff)
    digit=selectDigit(0x08)   # Select the fourth, and display the thousands digit
    outData(num[dec%10000//1000]|digit)
    time.sleep(0.003)

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

def loop():
    global t
    global counter
```

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

def destroy():
    global t
    GPIO.cleanup()
    t.cancel()

if __name__ == '__main__': # Program entrance
    print ('Program is starting...' )
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

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

```
dataPin   = 15       # DS Pin of 74HC595
latchPin  = 13       # ST_CP Pin of 74HC595
clockPin  = 11        # SH_CP Pin of 74HC595
num = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
counter = 0          # Variable counter, the number will be dislayed by 7-segment display
```

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

```
def selectDigit(digit): # Open one of the 7-segment display and close the remaining three, the parameter digit is optional for 1,2,4,8
    if digit==0x01:
        return (0x08<<8)
    elif digit==0x02:
        return (0x04<<8)
    elif digit==0x04:
        return (0x02<<8)
    elif digit==0x08:
        return (0x01<<8)
    else:
        return (0xf0<<8)
```

Subfunction **outData** (data) is used to make the 74HC595 output an 16-bit data immediately.

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

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

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

    outData(0xffff)
    digit=selectDigit(0x02)    # Select the second, and display the tens digit
    outData(num[dec%100//10]|digit)
    time.sleep(0.003)

    outData(0xffff)
    digit=selectDigit(0x04)    # Select the third, and display the hundreds digit
    outData(num[dec%1000//100]|digit)
    time.sleep(0.003)

    outData(0xffff)
    digit=selectDigit(0x08)    # Select the fourth, and display the thousands digit
    outData(num[dec%10000//1000]|digit)
    time.sleep(0.003)
```

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

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

Subfunction **setup**(), configure all input output modes for the GPIO pin used.
Finally, in loop function, make the digital tube display variable counter value in the while loop. The value will change in function **timer** (), so the content displayed by 7-segment display will change accordingly.

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

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

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

Chapter 18 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.

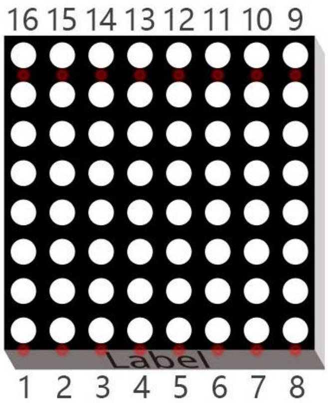
Project 18.1 LED Matrix

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 knowledge

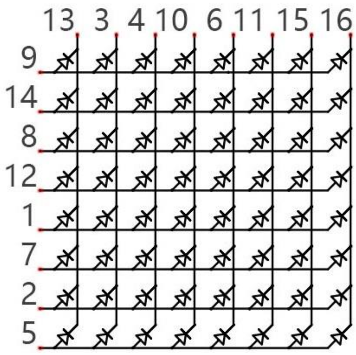
LED matrix

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).

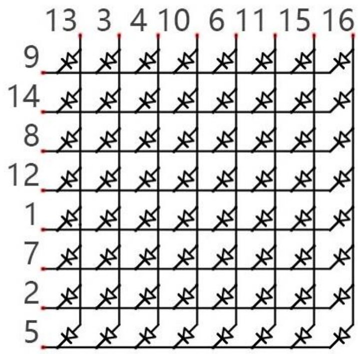


In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

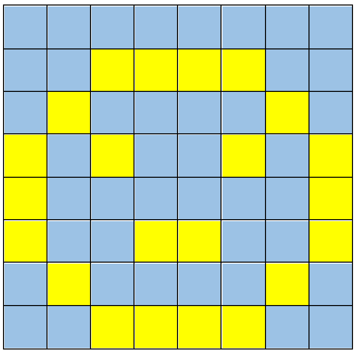
Connection mode of Common Anode



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.



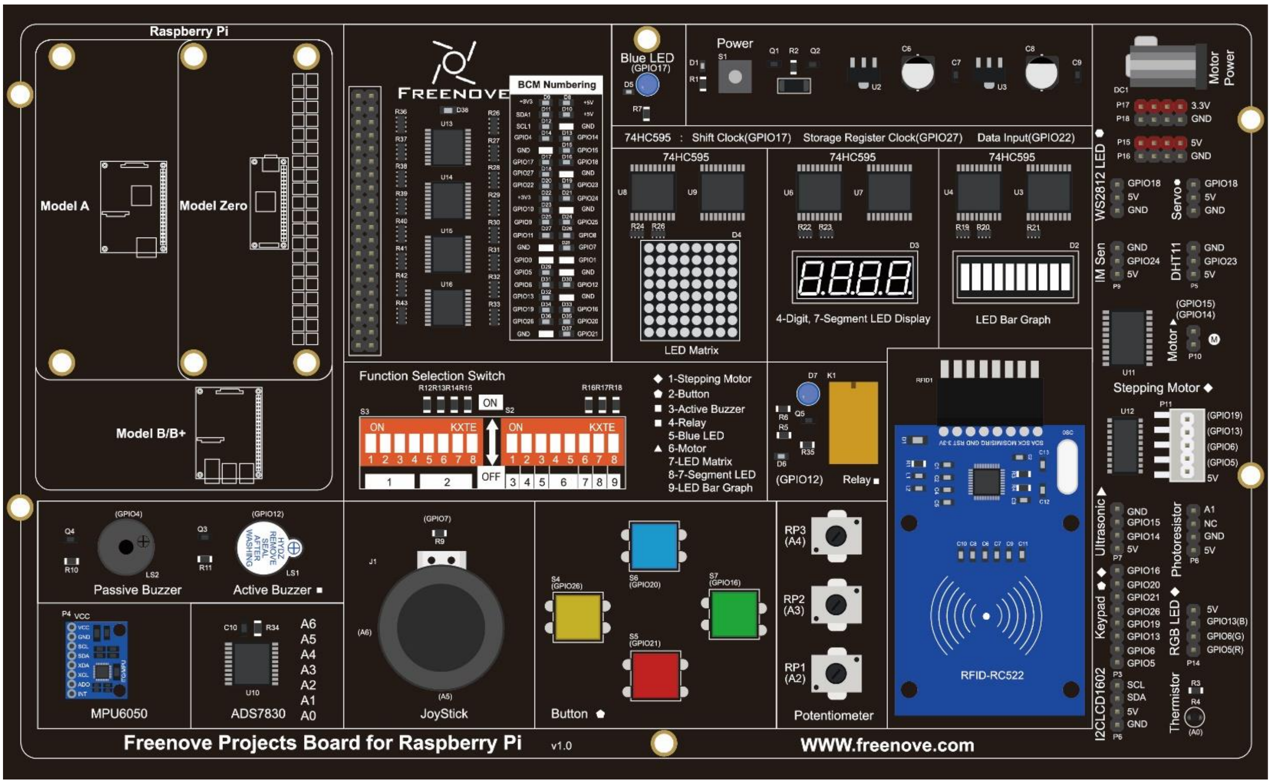
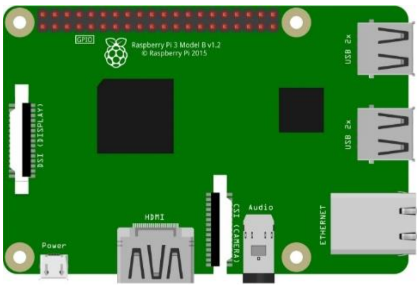
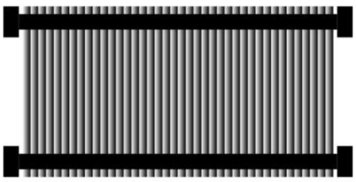
1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

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).

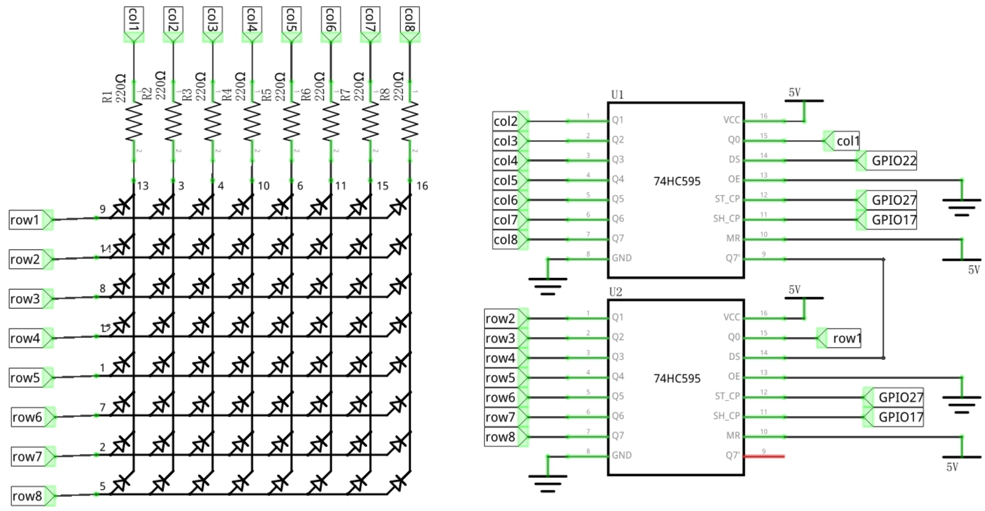
Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit. Every 74HC595 IC Chip has eight parallel output ports, so two of these have 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.

Component List

<table>
<tr><td colspan="3">Freenove Projects Board for Raspberry Pi x1

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td><td>4-Digit 7-Segment Display
</td></tr>
</table>

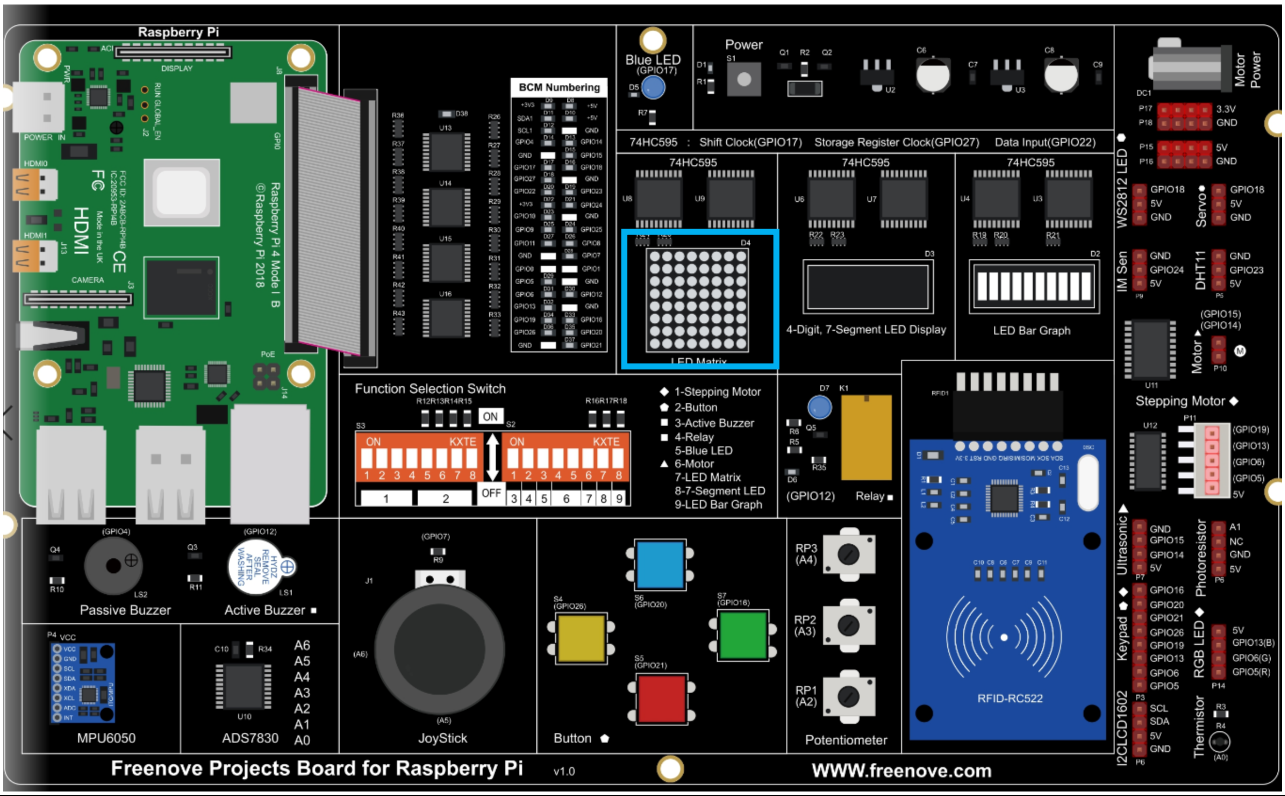
Circuit

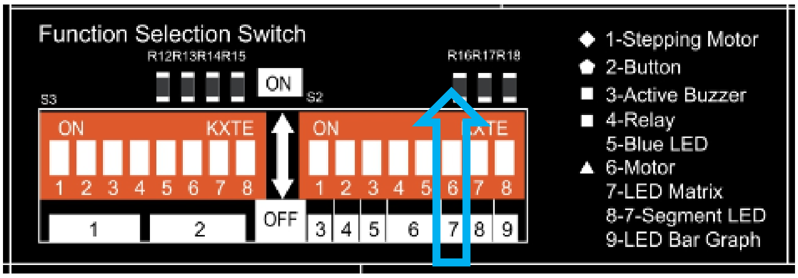
Schematic diagram



Hardware connection.

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





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

Code

Two 74HC595 IC Chips are used in this project, one for controlling the LED Matrix's columns and the other for controlling the rows. According to the circuit connection, row data should be sent first, then column data. The following code will make the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

C Code 18.1 LEDMatrix

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

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

1. Use cd command to enter 18_LEDMatrix directory of C language.

```
cd ~/Freenove_Kit/Code/C_Code/18_LEDMatrix
```

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

```
gcc LEDMatrix.c -o LEDMatrix -lwiringPi
```

3. Then run the generated file "LEDMatrix".

```
sudo ./LEDMatrix
```

After the program runs, the LED Matrix displays a smiling face, and then displays characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

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

#define   dataPin   3   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin  0   //SH_CP Pin of 74HC595(Pin11)
// data of smile face
unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
unsigned char data[]={  // data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  // " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00,  // "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00,  // "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00,  // "2"
```

```
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
};
void _shiftOut(int dPin,int cPin,int order,int val){
     int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
         }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
         }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}
int main(void)
{
    int i,j,k;
    unsigned char x;

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

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
```

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

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

The first “for” loop in the “while” loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

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

```
information to the second stage 74HC959

                digitalWrite(latchPin,HIGH);//Output data of two stage 74HC595 at the same
time
                x>>=1;    //display the next column
                delay(1);
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of 18 X 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column...... and so on…138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

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

Python Code 18.1 LEDMatrix

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

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

1. Use cd command to enter 18_LEDMatrix directory of Python language.

```
cd ~/Freenove_Kit/Code/Python_Code/18_LEDMatrix
```

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

```
python LEDMatrix.py
```

After the program runs, the LED Matrix displayss a smiling face, and then displays characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

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

```
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

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

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

                shiftOut(dataPin,clockPin,MSBFIRST,~x) #then shift data of column information to second stage 74HC959
                GPIO.output(latchPin,GPIO.HIGH) # Output data of two stage 74HC595 at the same time
                time.sleep(0.001) # display the next column
                x>>=1
        for k in range(0,len(data)-8): #len(data) total number of "0-F" columns
            for j in range(0,20): # times of repeated displaying LEDMatrix in every frame, the bigger the "j", the longer the display time.
                x=0x80      # Set the column information to start from the first column
                for i in range(k,k+8):
                    GPIO.output(latchPin,GPIO.LOW)
                    shiftOut(dataPin,clockPin,MSBFIRST,data[i])
                    shiftOut(dataPin,clockPin,MSBFIRST,~x)
                    GPIO.output(latchPin,GPIO.HIGH)
                    time.sleep(0.001)
                    x>>=1
def destroy():
    GPIO.cleanup()
if __name__ == '__main__':  # Program entrance
    print ('Program is starting...' )
```

```
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

The first “for” loop in the “while” loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

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

                shiftOut(dataPin,clockPin,MSBFIRST,~x) #then shift data of column information to second stage 74HC959
                GPIO.output(latchPin,GPIO.HIGH) # Output data of two stage 74HC595 at the same time
                time.sleep(0.001) # display the next column
                x>>=1
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of 18 X 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column...... and so on…138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

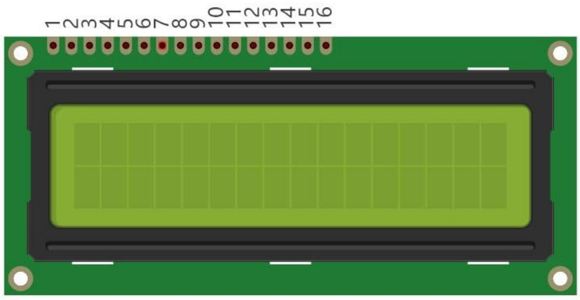
```
        for k in range(0,len(data)-8): #len(data) total number of "0-F" columns
            for j in range(0,20): # times of repeated displaying LEDMatrix in every frame, the bigger the "j", the longer the display time.
                x=0x80      # Set the column information to start from the first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

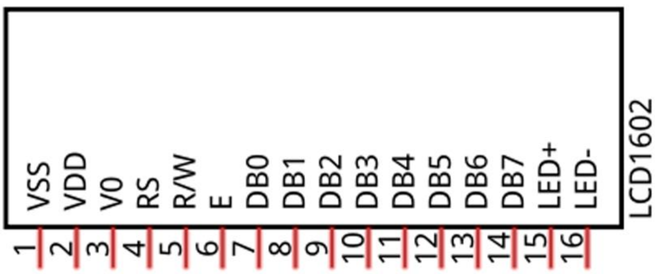
Chapter 19 LCD1602

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

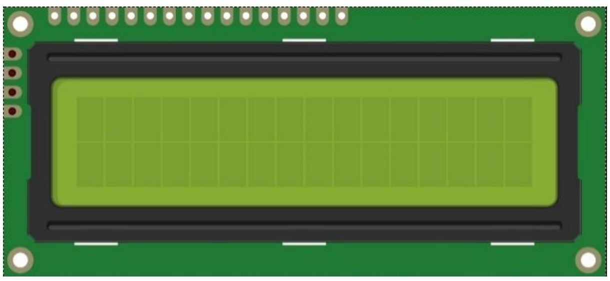
Project 19.1 I2C LCD1602

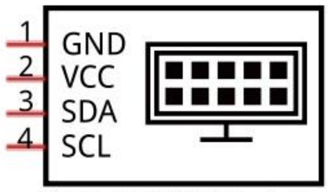
There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram





I2C LCD1602 Display Screen integrates a I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to only use 4 lines to operate the LCD1602.

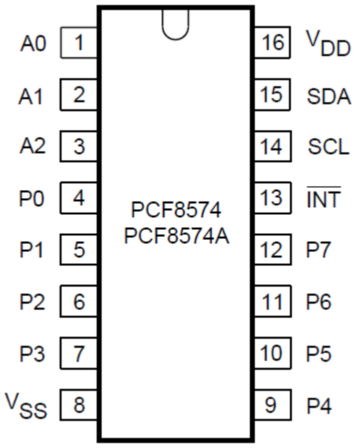
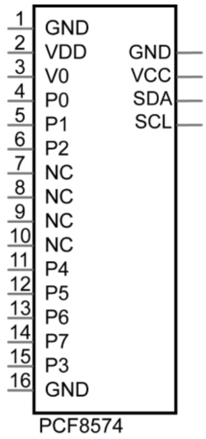




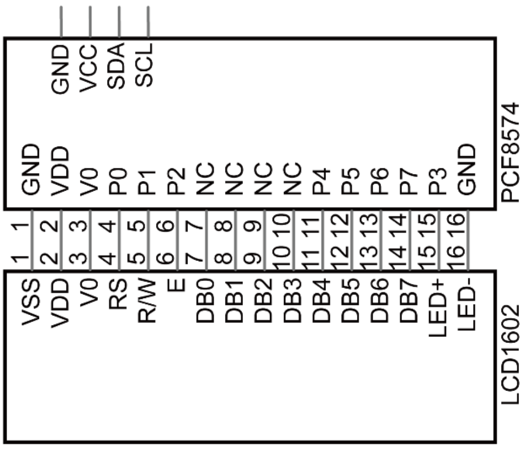
I2C LCD1602 Module

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

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

PCF8574 chip pin diagram:	PCF8574 module pin diagram
	

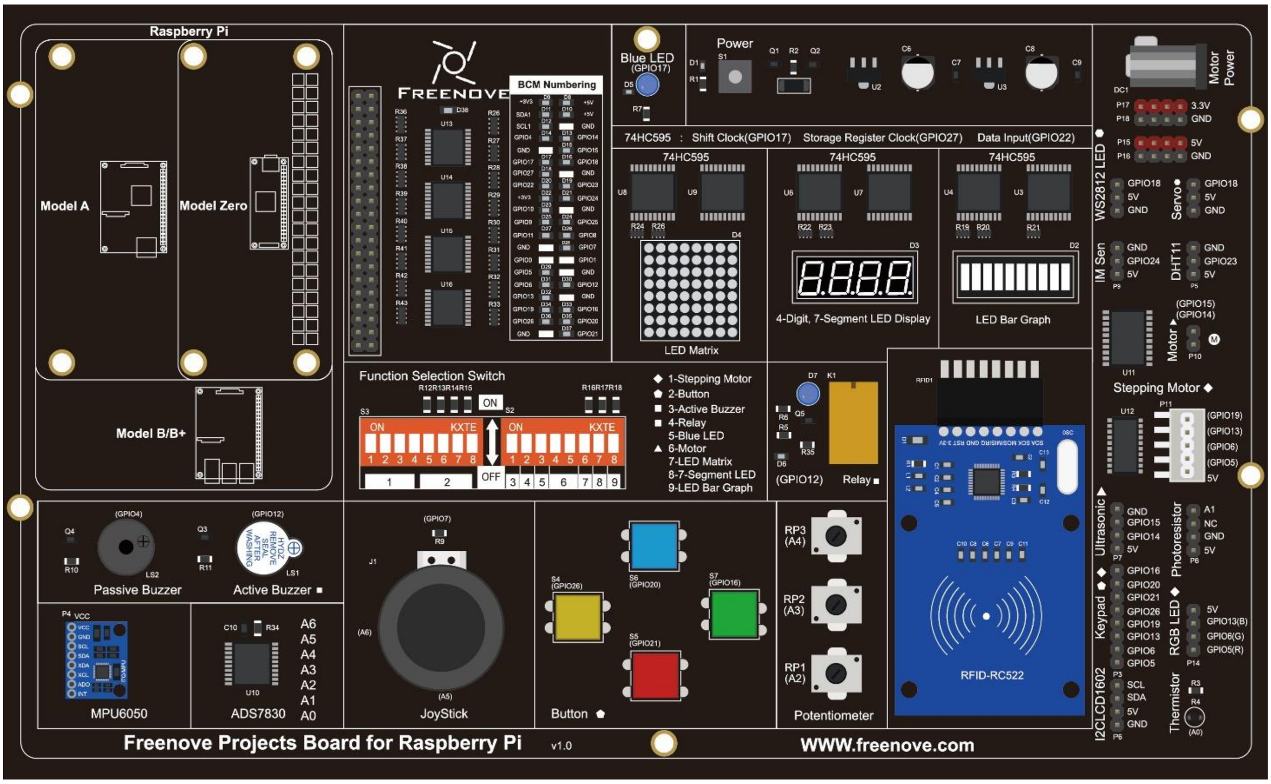
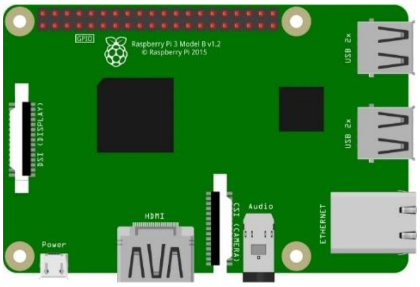
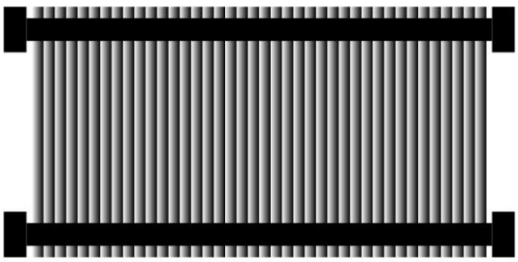
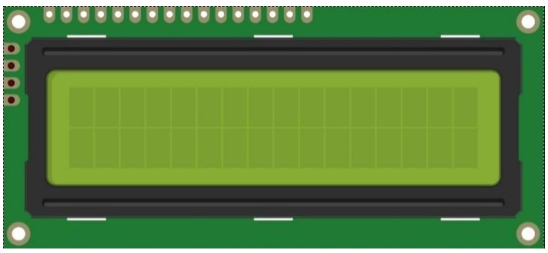
PCF8574 module pins and LCD1602 pins correspond to each other and are connected to each other:



Because of this, as stated earlier, we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface.

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

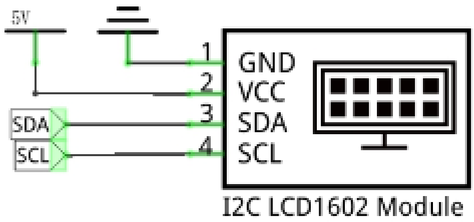
Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi

</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>I2C LCD1602 Module
</td></tr>
</table>

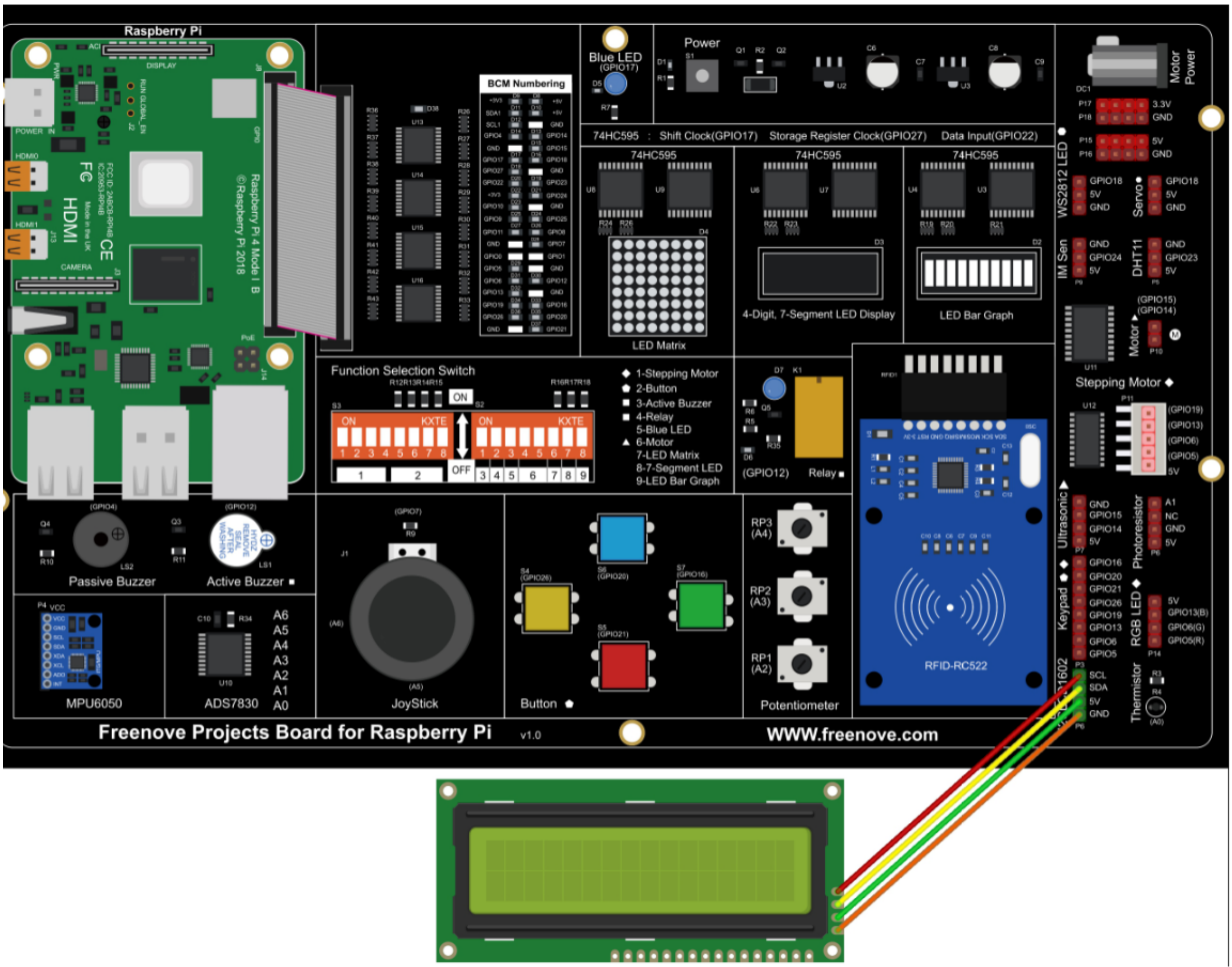
Circuit

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

Schematic diagram



Hardware connection.



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

Code

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

C Code 19.1 I2CLCD1602

If you haven't configured I2C and install Smbus, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 19_I2CLCD1602 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/19_I2CLCD1602
```

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

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

3. Then run the generated file "I2CLCD1602".

```
sudo ./I2CLCD1602
```

After the program runs, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program runs, if you nothing displays or the display is not clear, you can try to rotate the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display the Time and Temperature clearly.

The following is the program code:

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

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

```
#define LED     BASE+3
#define D4      BASE+4
#define D5      BASE+5
#define D6      BASE+6
#define D7      BASE+7

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

lcdPrintf(lcdhd,"Time:%02d:%02d:%02d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
int detectI2C(int addr){
    int _fd = wiringPiI2CSetup (addr);
    if (_fd < 0){
        printf("Error address : 0x%x \n",addr);
        return 0 ;
    }
    else{
        if(wiringPiI2CWrite(_fd,0) < 0){
            printf("Not found device in address 0x%x \n",addr);
            return 0;
        }
        else{
            printf("Found device in address 0x%x \n",addr);
```

```
            return 1 ;
        }
    }
}
int main(void){
    int i;

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

    wiringPiSetup();
    if(detectI2C(0x27)){
        pcf8574_address = 0x27;
    }else if(detectI2C(0x3F)){
        pcf8574_address = 0x3F;
    }else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    pcf8574Setup(BASE,pcf8574_address);//initialize PCF8574
    for(i=0;i<8;i++){
        pinMode(BASE+i,OUTPUT);     //set PCF8574 port to output mode
    }
    digitalWrite(LED,HIGH);     //turn on LCD backlight
    digitalWrite(RW,LOW);       //allow writing to LCD
     lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);// initialize LCD and return “handle” used to handle LCD
    if(lcdhd == -1){
        printf("lcdInit failed !");
        return 1;
    }
    while(1){
        printCPUTemperature();//print CPU temperature
        printDataTime();        // print system time
        delay(1000);
    }
    return 0;
}
```

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

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

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

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

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

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

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

Details about lcdInit():

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

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

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

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

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

Details about lcdPosition() and lcdPrintf():

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

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

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

Python Code 19.1 I2CLCD1602

If you haven't configured I2C and install Smbus, please refer to Chapter 7. If you've done it, please continue.

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

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

1. Use cd command to enter 19_I2CLCD1602 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/19_I2CLCD1602
```

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

```
python I2CLCD1602.py
```

After the program runs, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program runs, if nothing displays or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display the Time and Temperature clearly.

The following is the program code:

```
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

from time import sleep, strftime
from datetime import datetime

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

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

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

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

def destroy():
    lcd.clear()

PCF8574_address = 0x27  # I2C address of the PCF8574 chip.
PCF8574A_address = 0x3F  # I2C address of the PCF8574A chip.
# Create PCF8574 GPIO adapter.
try:
    mcp = PCF8574_GPIO(PCF8574_address)
except:
    try:
        mcp = PCF8574_GPIO(PCF8574A_address)
    except:
        print ('I2C Address Error !')
        exit(1)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)

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

Two modules are used in the code, PCF8574.py and Adafruit_LCD1602.py. These two documents and the code files are stored in the same directory, and neither of them is dispensable. Please DO NOT DELETE THEM! PCF8574.py is used to provide I2C communication mode and operation method of some of the ports for the RPi and PCF8574 IC Chip. Adafruit module Adafruit_LCD1602.py is used to provide some functional operation methods for the LCD1602 Display.

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

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

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

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

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

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

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

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

Subfunction used to get time:

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

Details about PCF8574.py and Adafruit_LCD1602.py:

Module PCF8574

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

Module Adafruit_LCD1602

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

Chapter 20 Hygrothermograph DHT11

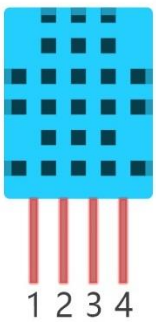
In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

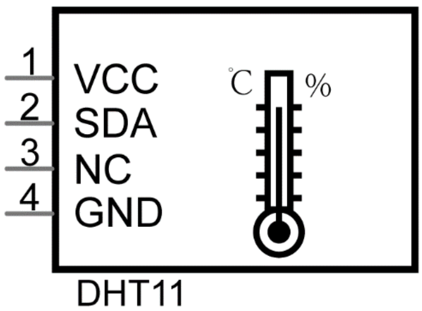
Project 20.1 Hygrothermograph

Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the RPi to read Temperature and Humidity data of the DHT11 Module.

Component knowledge

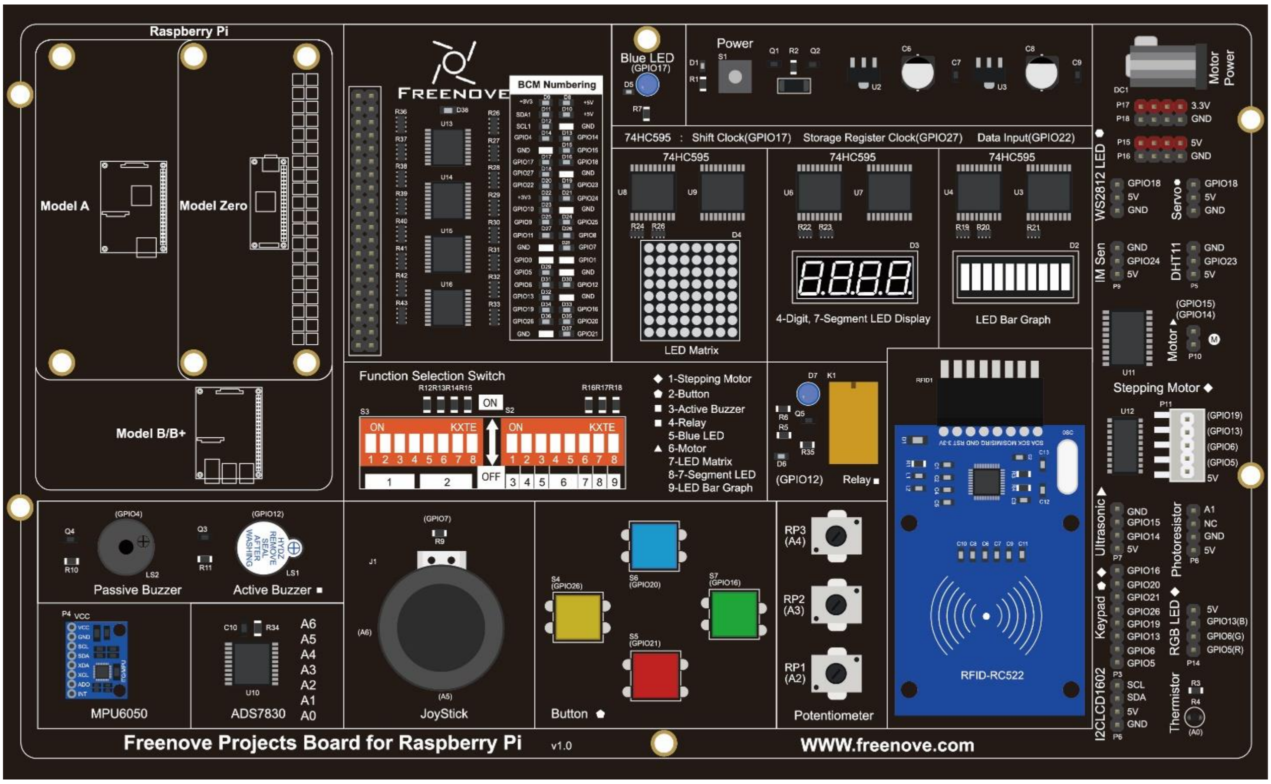
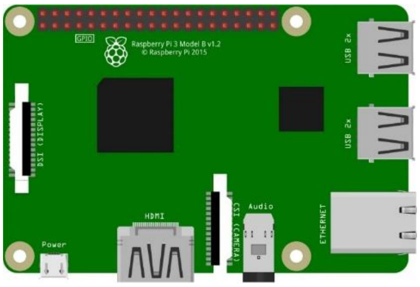
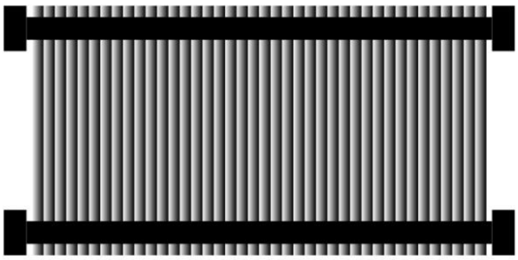
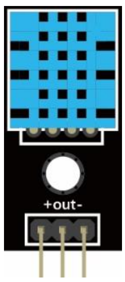
The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.





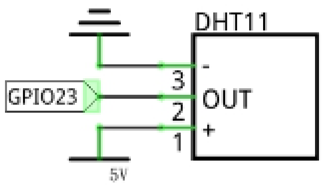
After being powered up, it will initialize in 1 second. Its operating voltage is within the range of 3.3V-5.5V.
The SDA pin is a data pin, which is used to communicate with other devices.
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Such pin has no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). It **should not be connected** to any of the circuit connections.

Component List

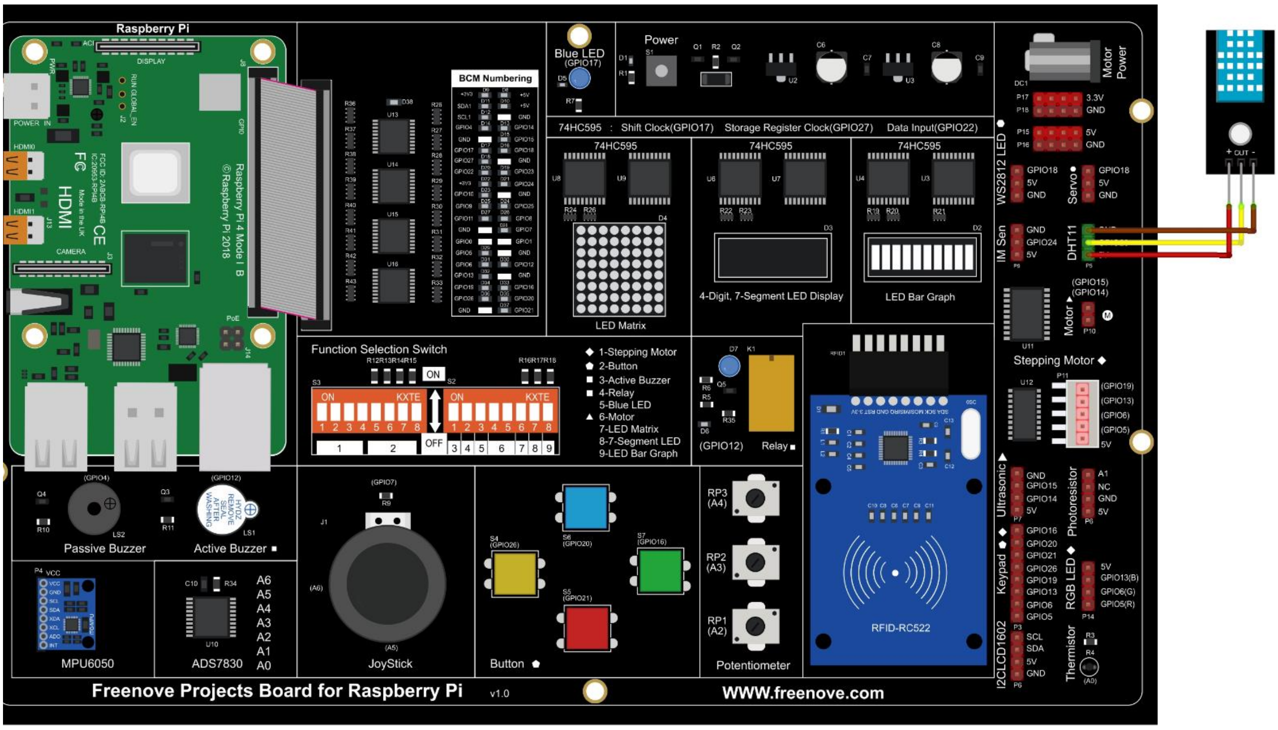
<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>DHT11 Module

</td></tr>
</table>

Circuit

Schematic diagram



Hardware connection.



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

Code

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

C Code 20.1 DHT11

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

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

1. Use cd command to enter 20_DHT11 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/20_DHT11
```

2. The code used in this project contains a custom header file. Use the following command to compile the code DHT11.cpp and DHT.cpp and generate executable file DHT11. The custom header file will be compiled at the same time.

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

3. Run the generated file "DHT11".

```
sudo ./DHT11
```

After the program runs, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts : 1
DHT11,OK!
Humidity is 50.00 %,     Temperature is 27.50 *C

Measurement counts : 2
DHT11,OK!
Humidity is 53.00 %,     Temperature is 27.50 *C

Measurement counts : 3
DHT11,OK!
Humidity is 54.00 %,     Temperature is 27.50 *C

Measurement counts : 4
DHT11,OK!
Humidity is 54.00 %,     Temperature is 27.50 *C
```

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdint.h>
#include "DHT.hpp"

#define DHT11_Pin  4     //define the pin of sensor

int main(){
    DHT dht;         //create a DHT class object
    int chk,sumCnt;//chk:read the return value of sensor; sumCnt:times of reading sensor

```

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

    wiringPiSetup();

    while(1){
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
        sumCnt++;       //counting number of reading times
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:     //if the return value is DHTLIB_OK, the data is normal.
                printf("DHT11,OK! \n");
                break;
            case DHTLIB_ERROR_CHECKSUM:     //data check has errors
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:      //reading DHT times out
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:      //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f
*C\n\n",dht.humidity,dht.temperature);
        delay(3000);
    }
    return 1;
}
```

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

```
    DHT dht;
```

In the "while" loop, the value of DHT11 is read every 3 seconds through the dht.readdht11 () function.

```
while(1){
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
        sumCnt++;       //counting number of reading times
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:     //if the return value is DHTLIB_OK, the data is normal.
                printf("DHT11,OK! \n");
```

```
                break;
            case DHTLIB_ERROR_CHECKSUM:     //data check has errors
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:      //reading DHT times out
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:      //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f
*C\n\n",dht.humidity,dht.temperature);
        delay(3000);
    }
```

Finally display the results:

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

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

```
#define _DHT_H_

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

////read return flag of sensor
#define DHTLIB_OK               0
#define DHTLIB_ERROR_CHECKSUM   -1
#define DHTLIB_ERROR_TIMEOUT    -2
#define DHTLIB_INVALID_VALUE    -999

#define DHTLIB_DHT11_WAKEUP     18
#define DHTLIB_DHT_WAKEUP       1

#define DHTLIB_TIMEOUT          100

class DHT{
    public:
        double humidity,temperature;    //use to store temperature and humidity data read
        int readDHT11(int pin);     //read DHT11
    private:
        uint8_t bits[5];    //Buffer to receiver data
        int readSensor(int pin,int wakeupDelay);    //
```

```
};
```

Python Code 20.1 DHT11

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

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

1. Use cd command to enter 20_DHT11 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/20_DHT11
```

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

```
python DHT11.py
```

After the program runs, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts:  2
DHT11,OK!
Humidity : 53.00,        Temperature : 27.60

Measurement counts:  3
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  4
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  5
DHT11,OK!
Humidity : 52.00,        Temperature : 27.50
```

The following is the program code:

```
import RPi.GPIO as GPIO
import time
import Freenove_DHT as DHT
DHTPin = 16     #define the pin of DHT11

def loop():
    dht = DHT.DHT(DHTPin)   #create a DHT class object
    counts = 0 # Measurement counts
    while(True):
        t=time.time()
        counts += 1
        print("Measurement counts: ", counts)
        for i in range(0,15):
            chk = dht.readDHT11()     #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
            if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
                print("DHT11,OK!")
```

```
                break
            time.sleep(0.1)
        print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature),time.time()-t)
        time.sleep(2)

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

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

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

Then in the "while" loop, use chk = dht.**readDHT11** (DHT11Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk".

```
    while(True):
        t=time.time()
        counts += 1
        print("Measurement counts: ", counts)
        for i in range(0,15):
            chk = dht.readDHT11()     #read DHT11 and get a return value. Then determine whether data read is normal according to the return value.
            if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine whether data read is normal according to the return value.
                print("DHT11,OK!")
                break
            time.sleep(0.1)
        print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature),time.time()-t)
        time.sleep(2)
```

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

Freenove_DHT Module

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

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

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

Chapter 21 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

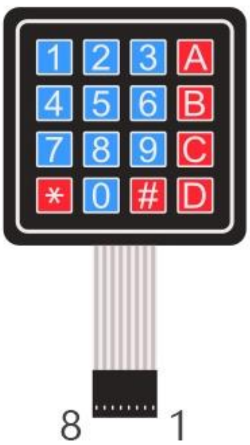
Project 21 Matrix Keypad

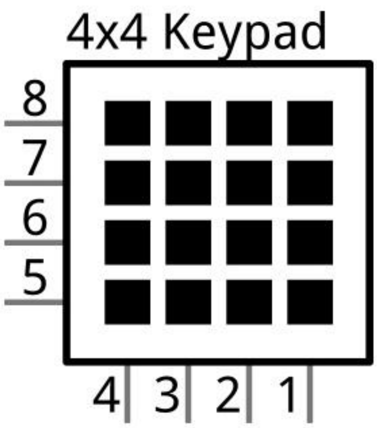
In this project, we will attempt to get every key code on the Matrix Keypad to work.

Component knowledge

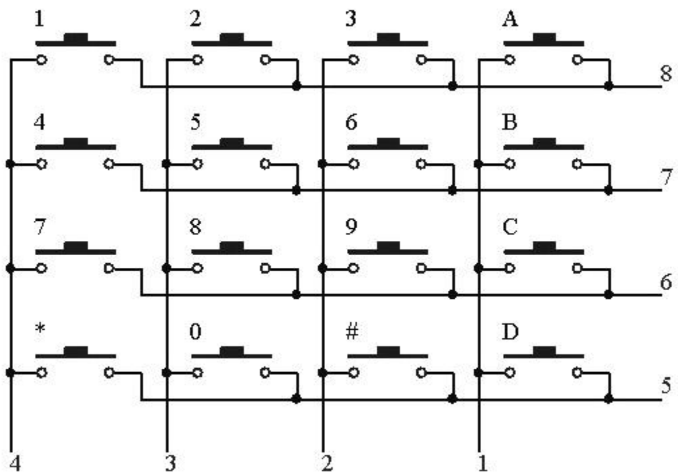
4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad Matrix integrates 16 keys (think of this as 16 Push Button Switches in one module):





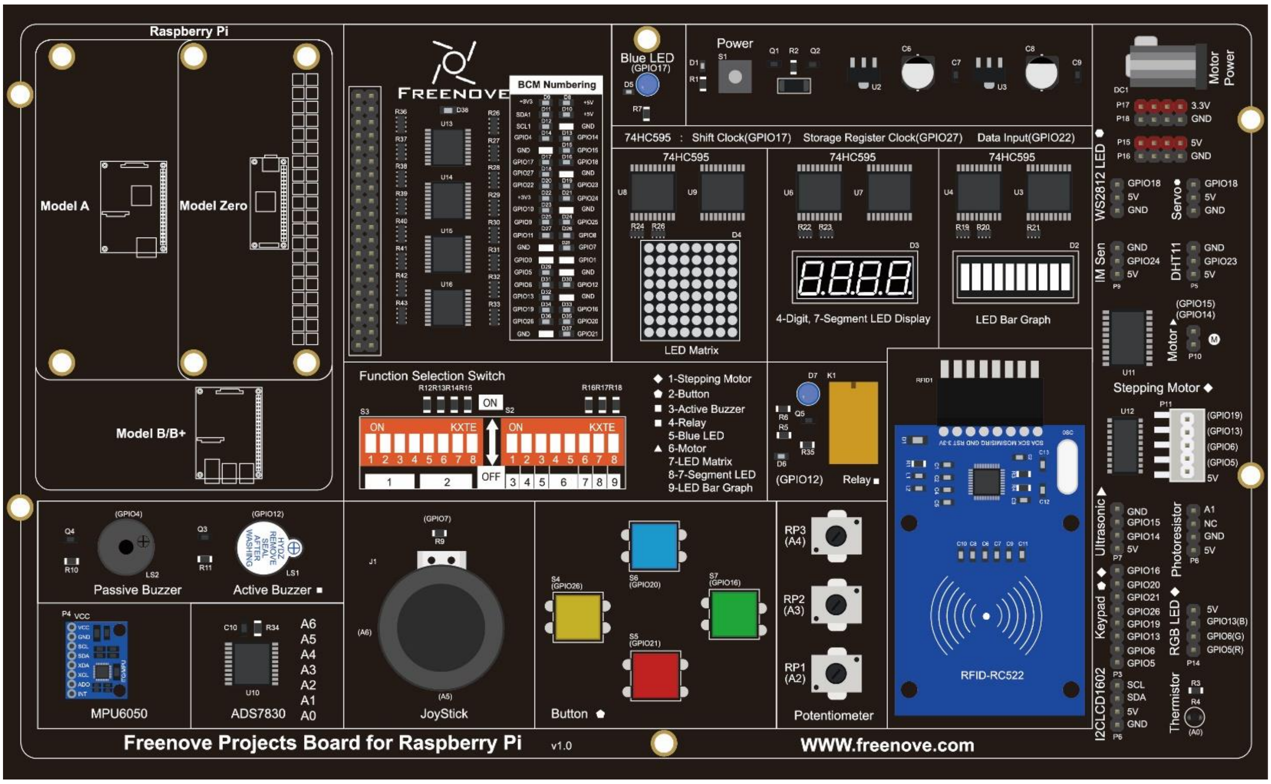
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



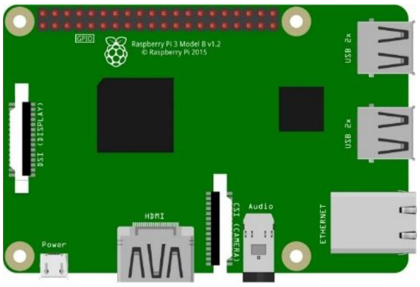
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of rows 5, 6, 7, 8 to judge whether the keys A, B, C, D are pressed. Then send low level to columns 2, 3, 4 in turn to detect whether other keys are pressed. Therefore, you can get the state of all of the keys.

Component List

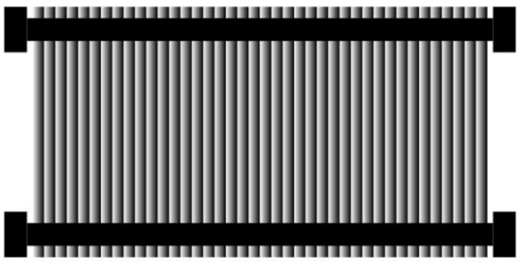
Freenove Projects Board for Raspberry Pi



Raspberry Pi



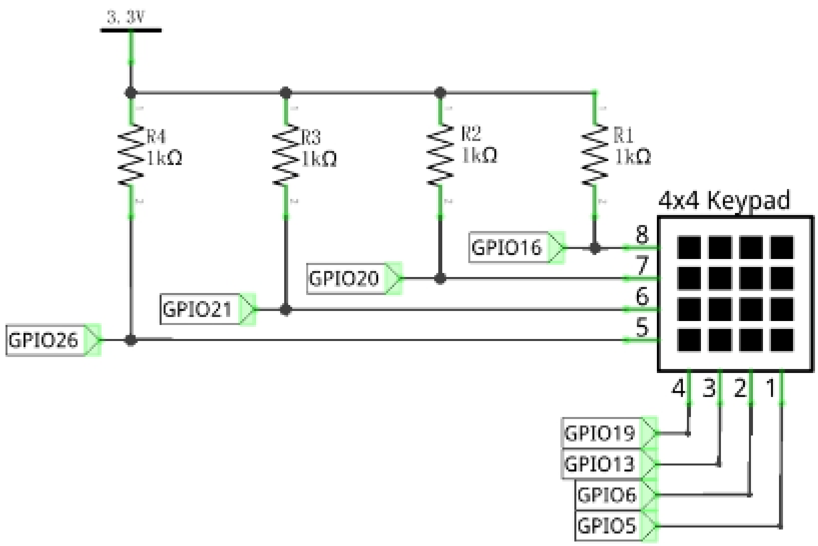
GPIO Ribbon Cable



4x4 Matrix Keypad

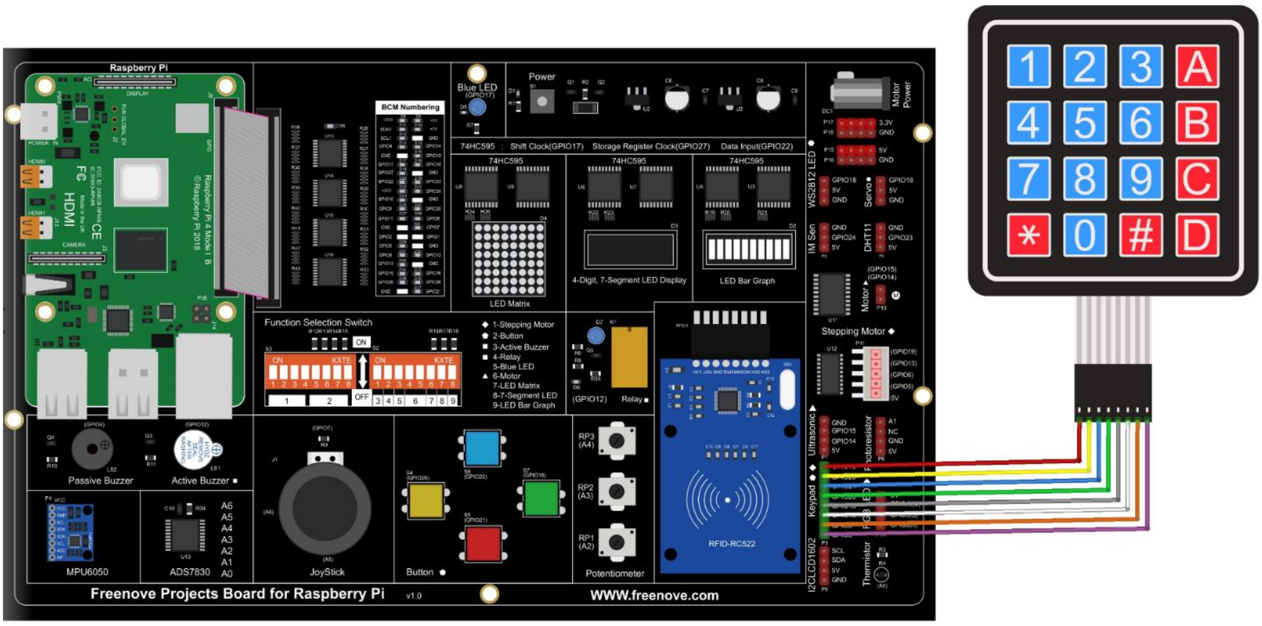
Circuit

Schematic diagram



All the rows are held high until a switch is pressed.

Hardware connection.



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

Code

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

C Code 21.1 MatrixKeypad

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

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

1. Use cd command to enter 21_MatrixKeypad directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/21_MatrixKeypad
```

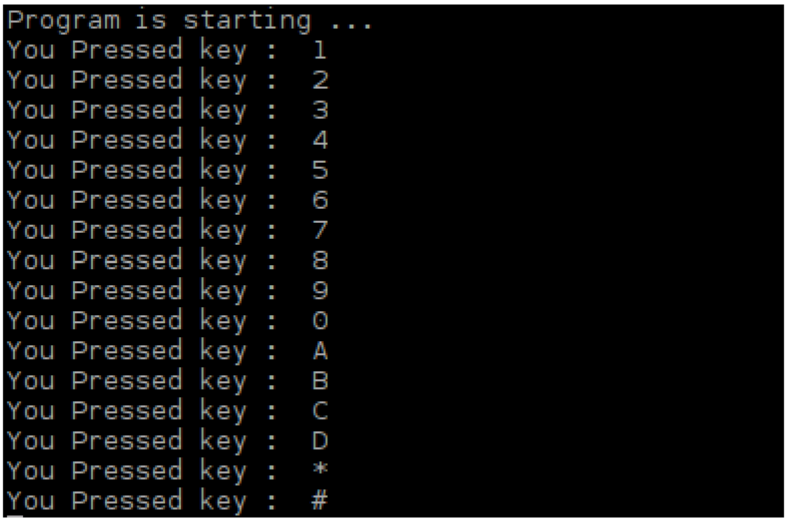
2. Code of this project contains a custom header file. Use the following command to compile the code MatrixKeypad.cpp, Keypad.cpp and Key.cpp generate executable file MatrixKeypad. The custom header file will be compiled at the same time.

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

3. Run the generated file "MatrixKeypad".

```
sudo ./MatrixKeypad
```

After the program runs, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:



The following is the program code:

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

```
byte rowPins[ROWS] = {27,28,29,25 }; //define the row pins for the keypad
byte colPins[COLS] = {24,23,22,21}; //define the column pins for the keypad
//create Keypad object
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

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

    wiringPiSetup();

    char key = 0;
    keypad.setDebounceTime(50);
    while(1){
        key = keypad.getKey();  //get the state of keys
        if (key){       //if a key is pressed, print out its key code
            printf("You Pressed key :  %c \n",key);
        }
    }
    return 1;
}
```

In this project code, we use two custom library file "**Keypad**.hpp" and "**Key**.hpp". They are located in the same directory with program files "**MatrixKeypad**.cpp", "**Keypad**.cpp" and "**Key**.cpp". The Library Keypad is “transplanted” from the Arduino Library Keypad. This library file provides a method to read the Matrix Keyboard’s input. By using this library, we can easily read the pressed keys of the Matrix Keyboard.

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

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

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

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

Set the debounce time to 50ms, and this value can be set based on the actual characteristics of the keyboard's flexibly, with a default time of 10ms.

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

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

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

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

class Keypad

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

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

`char getKey();`

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

`void setDebounceTime(uint);`

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

`void setHoldTime(uint);`

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

`bool isPressed(char keyChar);`

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

`char waitForKey();`

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

`KeyState getState();`

Get state of the keys.

`bool keyStateChanged();`

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

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

Python Code 21.1 MatrixKeypad

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

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

1. Use cd command to enter 21_MatrixKeypad directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/21_MatrixKeypad
```

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

```
python MatrixKeypad.py
```

After the program runs, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:

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

The following is the program code:

```
import RPi.GPIO as GPIO
import Keypad       #import module Keypad
ROWS = 4        # number of rows of the Keypad
COLS = 4        #number of columns of the Keypad
keys =  [   '1','2','3','A',     #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'     ]
#rowsPins = [12,16,18,22]        #connect to the row pinouts of the keypad
#colsPins = [19,15,13,11]        #connect to the column pinouts of the keypad
rowsPins = [36,38,40,37]        #connect to the row pinouts of the keypad
colsPins = [35,33,31,29]        #connect to the column pinouts of the keypad
def loop():
    keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)     #creat Keypad object
    keypad.setDebounceTime(100)      #set the debounce time
    while(True):
        key = keypad.getKey()       #obtain the state of keys
        if(key != keypad.NULL):     #if there is key pressed, print its key code.
            print ("You Pressed Key : %c "%(key))
```

```
if __name__ == '__main__':      #Program start from here
    print ("Program is starting ... ")
    try:
        loop()
    except KeyboardInterrupt:  #When 'Ctrl+C' is pressed, exit the program.
        GPIO.cleanup()
```

Import Keypad. Define row and column. Define key value variable. Define row pins and column pins.

```
import Keypad       #import module Keypad
ROWS = 4        # number of rows of the Keypad
COLS = 4        #number of columns of the Keypad
keys =  [   '1','2','3','A',     #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'     ]
rowsPins = [36,38,40,37]        #connect to the row pinouts of the keypad
colsPins = [35,33,31,29]         #connect to the column pinouts of the keypad
```

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

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

Set the debounce time to 100ms, and this value can be set based on the actual characteristics of the keyboard's flexibly, with a default time of 10ms.

```
keypad.setDebounceTime(100)       #set the debounce time
```

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

```
    while(True):
        key = keypad.getKey()       #obtain the state of keys
        if(key != keypad.NULL):     #if there is key pressed, print its key code.
            print ("You Pressed Key : %c "%(key))
```

```
class Keypad
def __init__(self,usrKeyMap,row_Pins,col_Pins,num_Rows,num_Cols):
```
Constructed function, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.
```
def getKey(self):
```
Get a pressed key. If no key is pressed, the return value is keypad NULL.
```
def setDebounceTime(self,ms):
```
Set the debounce time. And the default time is 10ms.
```
def setHoldTime(self,ms):
```
Set the time when the key holds stable state after pressed.
```
def isPressed(keyChar):
```
Judge whether the key with code "keyChar" is pressed.
```
def waitForKey():
```
Wait for a key to be pressed, and return key code of the pressed key.
```
def getState():
```
Get state of the keys.
```
def keyStateChanged():
```
Judge whether there is a change of key state, then return True or False.

Chapter 22 Infrared Motion Sensor

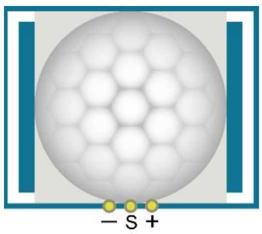
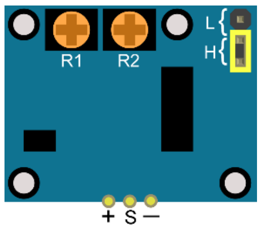
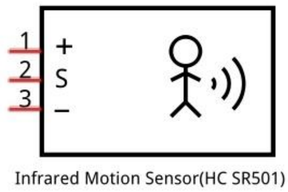
In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 22.1 PIR Infrared Motion Detector with LED Indicator

In this project, we will make a Motion Detector, with the human body infrared pyroelectric sensors.
When someone is in close proximity to the Motion Detector, it will automatically light up and when there is no one close by, it will be out.
This Infrared Motion Sensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals.

Component Knowledge

The following is the diagram of the Infrared Motion Sensor（HC SR-501）a PIR Sensor:

Top	Bottom	Schematic
		

Description:

1. Working voltage: 5v-20v(DC） Static current: 65uA.
2. Automatic Trigger. When a living body enters into the active area of sensor, the module will output high level (3.3V). When the body leaves the sensor's active detection area, it will output high level lasting for time period T, then output low level(0V). Delay time T can be adjusted by the potentiometer R1.
3. According to the position of Fresnel lenses dome, you can choose non-repeatable trigger modes or repeatable modes.

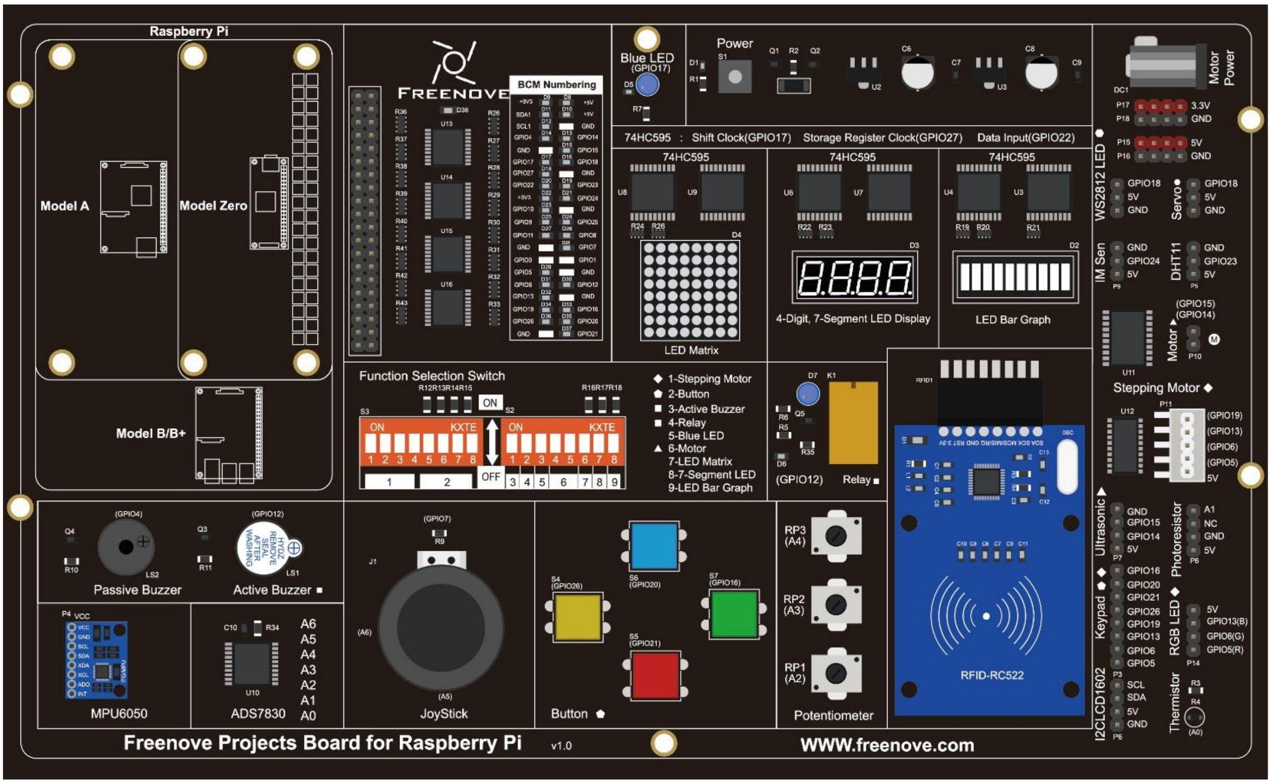
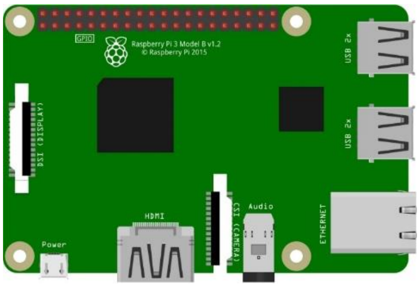
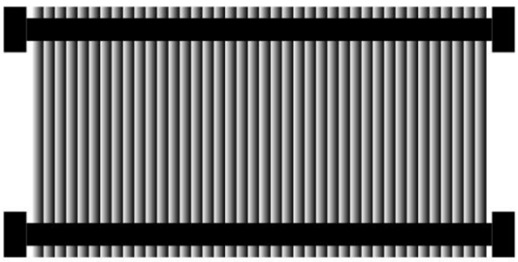
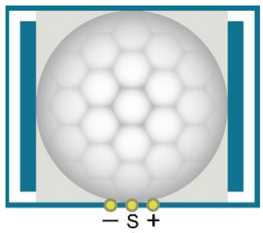
   L: non-repeatable trigger mode. The module outputs high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

   H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves during the period of high level output. After this, it starts to time and output low level after delaying T time.
4. Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level
5. Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.
6. One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome

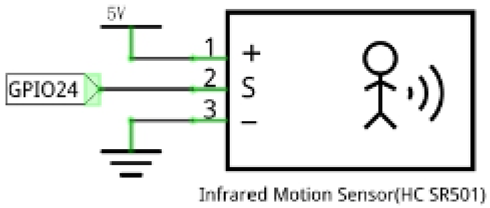
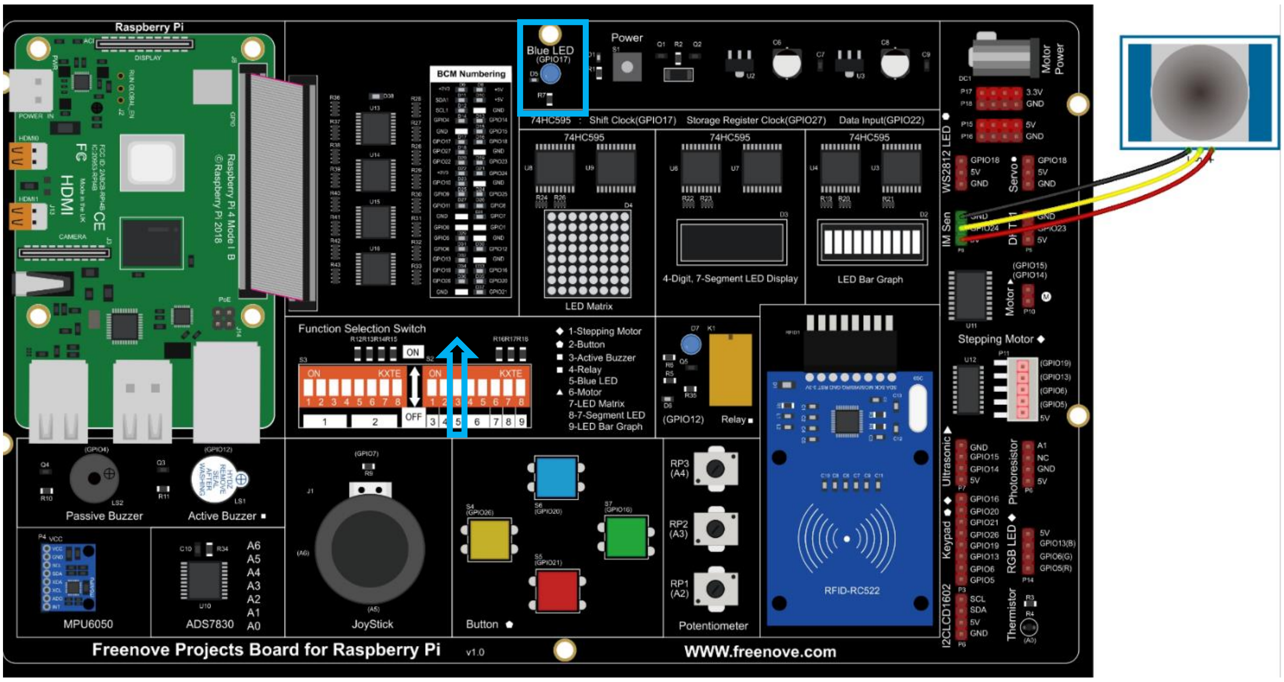
edge, the sensor will work at high sensitively. When a body moves close to or moves away from the sensor's dome in a vertical direction (perpendicular to the dome), the sensor cannot detect well (please take note of this deficiency). Actually this makes sense when you consider that this sensor is usually placed on a celling as part of a security product. Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

We can regard this sensor as a simple inductive switch when in use.

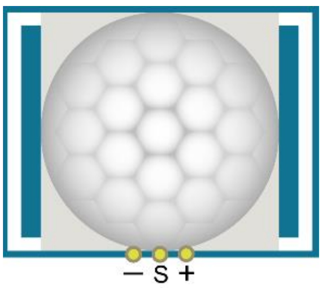
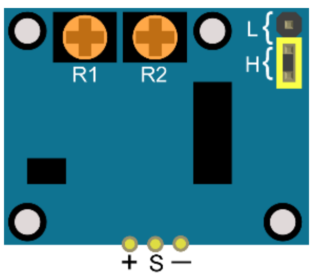
Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi

</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Jumper Wire
</td><td>HC SR501

</td></tr>
</table>

Circuit

Schematic diagram

Hardware connection.


How to use this sensor?

Top	Bottom
	

Description:

1. You can choose non-repeatable trigger modes or repeatable modes.

   L: non-repeatable trigger mode. The module outputs high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

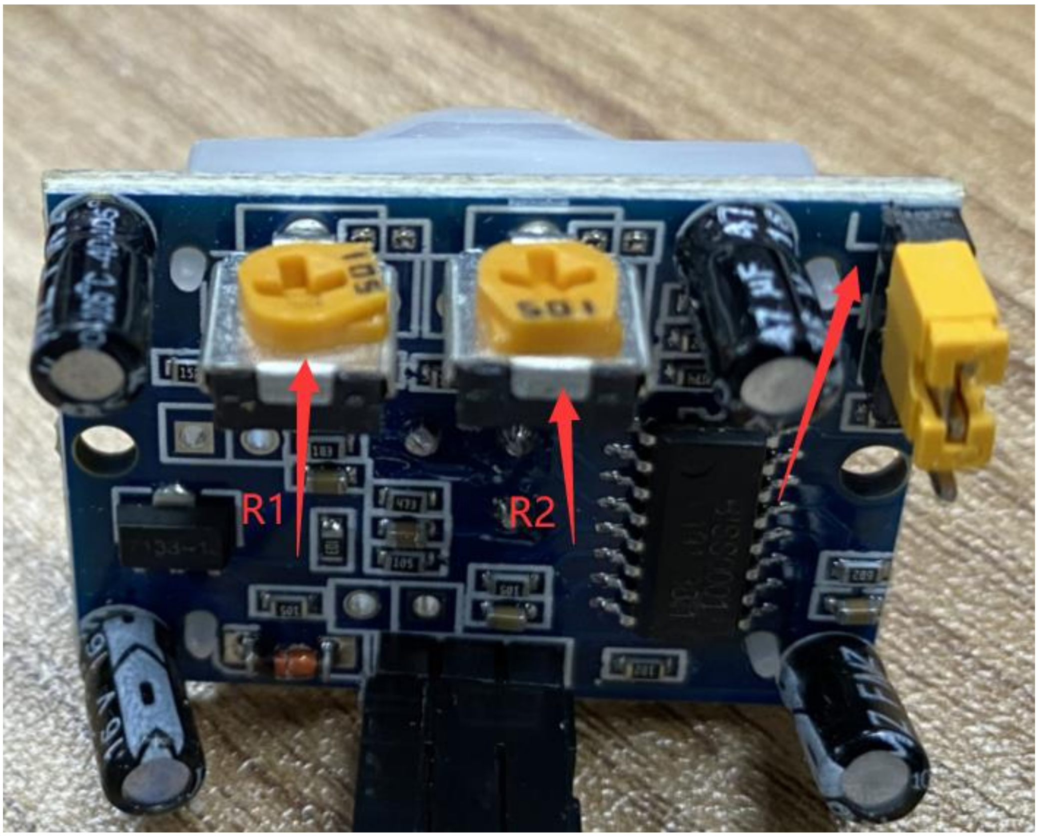
   H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves. After this, it starts to time and output low level after delaying T time.

2. R1 is used to adjust HIGH level lasting time when sensor detects human motion, 1.2s-320s.
3. R2 is used to adjust the maxmum distance the sensor can detect, 3~5m.

Here we connect L and adjust R1 and R2 like below to do this project.

Put you hand close and away from the sensor slowly. Obsever the LED in previous circuit.

It need some time between two detections.



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

Code

In this project, we will use the Infrared Motion Sensor to trigger an LED, essentially making the Infrared Motion sensor act as a Motion Switch. Therefore, the code is very similar to the earlier project "Push Button Switch and LED". The difference is that, when Infrared Motion Sensor detects change, it will output high level; when button is pressed, it will output low level. When the sensor output high level, the LED turns ON, or it will turn OFF.

C Code 22.1 SenseLED

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

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

1. Use cd command to enter 22_1_InfraredSensor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/22_1_InfraredSensor
```

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

```
gcc SenseLED.c -o SenseLED -lwiringPi
```

3. Run the generated file "SenseLED".

```
sudo ./SenseLED
```

After the program runs, **wait 1 minute for initialization**. Then move away from or move closer to the Infrared Motion Sensor and observe whether the LED turns ON or OFF. The Terminal window will continuously display the state of LED. As is shown below:

```
led on...
led on...
led on...
led on...
led on...
led on...
led on...
```

The following is the program code:

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

#define ledPin     0    //define the ledPin
#define sensorPin 5        //define the sensorPin

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

    wiringPiSetup();

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

    while(1){

        if(digitalRead(sensorPin) == HIGH){ //if read value of sensor is HIGH level
            digitalWrite(ledPin, HIGH);   //make led on
            printf("led turned on >>> \n");
        }
        else {
            digitalWrite(ledPin, LOW);   //make led off
            printf("led turned off <<< \n");
        }
    }

```

```
    return 0;
}
```

Python Code 22.1 SenseLED

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

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

1. Use cd command to enter 22_InfraredSensor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/22_InfraredSensor
```

2. Use Python command to execute code "SenseLED.py".

```
python SenseLED.py
```

After the program runs, **wait 1 minute for initialization**. Then move away from or move closer to the Infrared Motion Sensor and observe whether the LED turns ON or OFF. The Terminal window will continuously display the state of LED. As is shown below:

The following is the program code:

```
import RPi.GPIO as GPIO

ledPin = 11       # define ledPin
sensorPin = 18    # define sensorPin

def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)   # set ledPin to OUTPUT mode
    GPIO.setup(sensorPin, GPIO.IN) # set sensorPin to INPUT mode

def loop():
    while True:
        if GPIO.input(sensorPin)==GPIO.HIGH:
            GPIO.output(ledPin,GPIO.LOW) # turn off led
            print ('led turned off >>>')
        else :
            GPIO.output(ledPin,GPIO.HIGH) # turn on led
            print ('led turned on <<<')

def destroy():
    GPIO.cleanup()                     # Release GPIO resource

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

Chapter 23 Ultrasonic Ranging

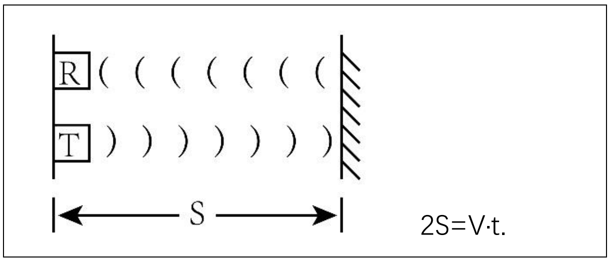
In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 23.1 Ultrasonic Ranging

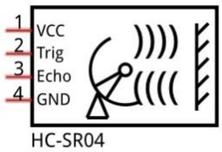
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will be reflected when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

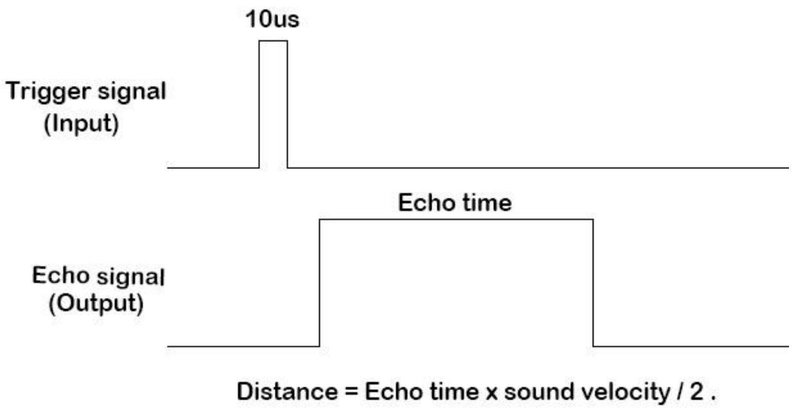
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

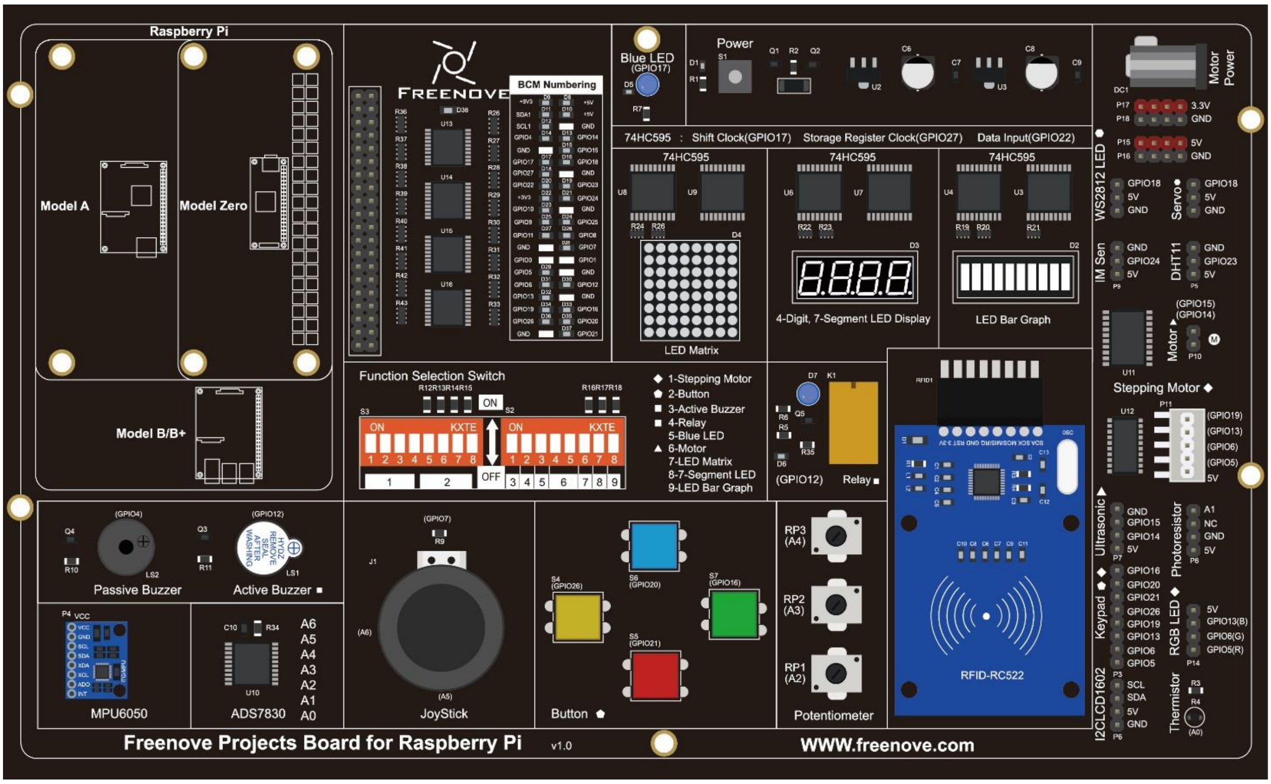
Minimum measured distance: 2cm Maximum measured distance: 200cm

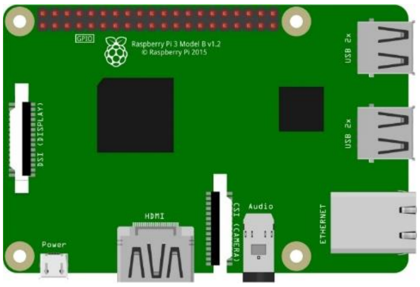
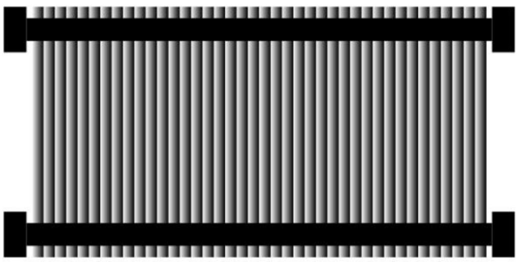
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2. This is done constantly.



Component List

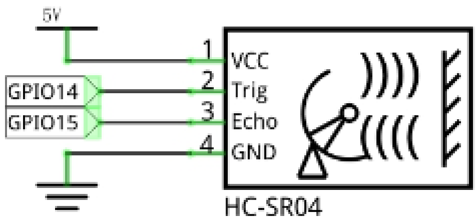
Freenove Projects Board for Raspberry Pi



Raspberry Pi	GPIO Ribbon Cable
	
Jumper Wire	HC SR04
	

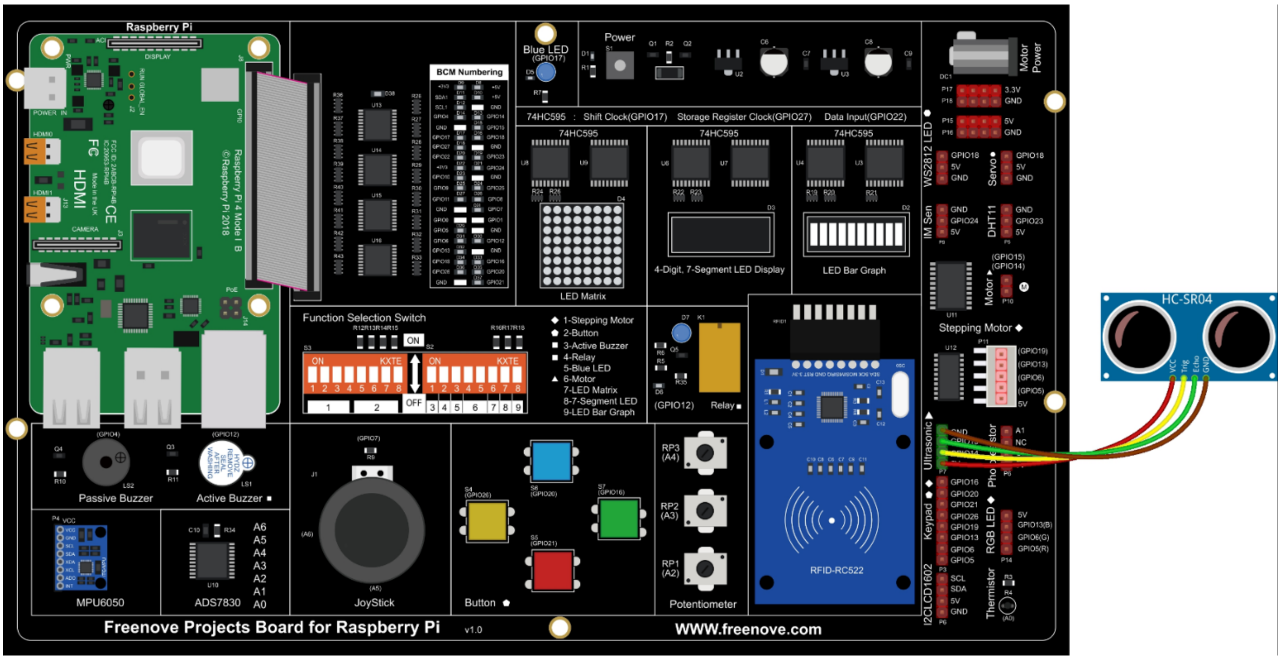
Circuit

Schematic diagram



Hardware connection.

After running the program, hold an object in front of the sensor and change their distance.



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

Code

C Code 23.1 UltrasonicRanging

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

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

1. Use cd command to enter 23_UltrasonicRanging directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/23_UltrasonicRanging
```

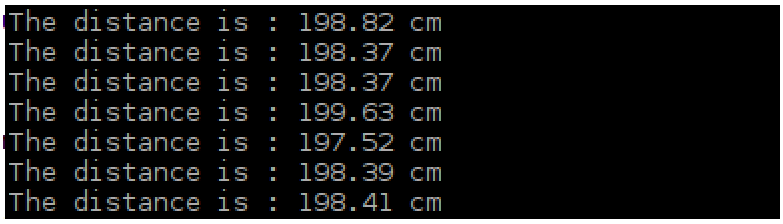
2. Use following command to compile "UltrasonicRanging.c" and generate executable file "UltrasonicRanging".

```
gcc UltrasonicRanging.c -o UltrasonicRanging -lwiringPi
```

3. Then run the generated file "UltrasonicRanging".

```
sudo ./UltrasonicRanging
```

After the program runs, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:



The following is the program code:

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

#define trigPin 15
#define echoPin 16
#define MAX_DISTANCE 220        // define the maximum measured distance
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured distance
//function pulseIn: obtain pulse time of a pin
int pulseIn(int pin, int level, int timeout);
float getSonar(){   //get the measurement result of ultrasonic module with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //send 10us high level to trigPin
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; //calculate distance with sound speed 340m/s
```

```
    return distance;
}

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

    wiringPiSetup();

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

int pulseIn(int pin, int level, int timeout)
{
   struct timeval tn, t0, t1;
   long micros;
   gettimeofday(&t0, NULL);
   micros = 0;
   while (digitalRead(pin) != level)
   {
      gettimeofday(&tn, NULL);
      if (tn.tv_sec > t0.tv_sec) micros = 1000000L; else micros = 0;
      micros += (tn.tv_usec - t0.tv_usec);
      if (micros > timeout) return 0;
   }
   gettimeofday(&t1, NULL);
   while (digitalRead(pin) == level)
   {
      gettimeofday(&tn, NULL);
      if (tn.tv_sec > t0.tv_sec) micros = 1000000L; else micros = 0;
      micros = micros + (tn.tv_usec - t0.tv_usec);
      if (micros > timeout) return 0;
   }
   if (tn.tv_sec > t1.tv_sec) micros = 1000000L; else micros = 0;
   micros = micros + (tn.tv_usec - t1.tv_usec);
   return micros;
}
```

First, define the pins and the maximum measurement distance.

```
#define trigPin 15
#define echoPin 16
#define MAX_DISTANCE 220        // define the maximum measured distance
```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance, that is, time Out. **timeOut= 2*MAX_DISTANCE/100/340*1000000**. This formula is (not approximately) 58.8 and 60 is used as an approximation.

```
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured distance
```

Subfunction **getSonar** () function is used to start the Ultrasonic Module to begin measurements and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn** () to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
float getSonar(){   //get the measurement result of ultrasonic module with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //send 10us high level to trigPin
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; //calculate distance with sound speed
340m/s
    return distance;
}
```

Lastly, in the while loop of main function, get the measurement distance and display it continually.

```
while(1){
        distance = getSonar();
        printf("The distance is : %.2f cm\n",distance);
        delay(1000);
    }
```

About function `pulseIn()`:

int pulseIn(int pin, int level, int timeout);
Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

Python Code 23.1 UltrasonicRanging

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

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

1. Use cd command to enter 23_UltrasonicRanging directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/23_UltrasonicRanging
```

2. Use Python command to execute code "UltrasonicRanging.py".

```
python UltrasonicRanging.py
```

After the program runs, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.75 cm
The distance is : 199.22 cm
The distance is : 198.42 cm
The distance is : 198.74 cm
The distance is : 198.37 cm
The distance is : 198.47 cm
The distance is : 198.41 cm
```

The following is the program code:

```python
import RPi.GPIO as GPIO
import time

trigPin = 8
echoPin = 10
MAX_DISTANCE = 220          # define the maximum measuring distance, unit: cm
timeOut = MAX_DISTANCE*60   # calculate timeout according to the maximum measuring distance

def pulseIn(pin,level,timeOut): # obtain pulse time of a pin under timeOut
    t0 = time.time()
    while(GPIO.input(pin) != level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    t0 = time.time()
    while(GPIO.input(pin) == level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    pulseTime = (time.time() - t0)*1000000
    return pulseTime

def getSonar():     # get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      # make trigPin output 10us HIGH level
    time.sleep(0.00001)     # 10us
    GPIO.output(trigPin,GPIO.LOW) # make trigPin output LOW level
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   # read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # calculate distance with sound speed
```

```
340m/s
    return distance

def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    GPIO.setup(trigPin, GPIO.OUT)   # set trigPin to OUTPUT mode
    GPIO.setup(echoPin, GPIO.IN)    # set echoPin to INPUT mode

def loop():
    while(True):
        distance = getSonar() # get distance
        print ("The distance is : %.2f cm"%(distance))
        time.sleep(1)

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        GPIO.cleanup()         # release GPIO resource
```

First, define the pins and the maximum measurement distance.

```
trigPin = 8
echoPin = 10
MAX_DISTANCE = 220          # define the maximum measuring distance, unit: cm
```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance (200cm). Then **timOut= 2*MAX_DISTANCE/100/340*1000000**. The result of the constant part in this formula is approximately 58.8.

```
timeOut = MAX_DISTANCE*60
```

Subfunction **getSonar** () function is used to start the Ultrasonic Module to begin measurements, and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn** () to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
def getSonar():     # get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      # make trigPin output 10us HIGH level
    time.sleep(0.00001)     # 10us
    GPIO.output(trigPin,GPIO.LOW) # make trigPin output LOW level
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   # read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # calculate distance with sound speed 
340m/s
    return distance
```

Finally, in the while loop of main function, get the measurement distance and display it continually.

```
    while(True):
        distance = getSonar()
        print ("The distance is : %.2f cm"%(distance))
        time.sleep(1)
```

About function `def pulseIn(pin,level,timeOut)`:

def pulseIn(pin,level,timeOut):
Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

Chapter 24 Attitude Sensor MPU6050

In this chapter, we will learn about a MPU6050 Attitude sensor, which integrates an Accelerometer and Gyroscope.

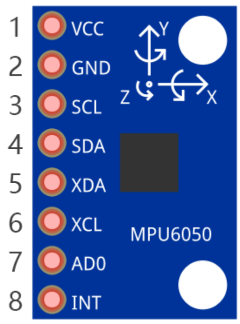
Project 24.1 Read an MPU6050 Sensor Module

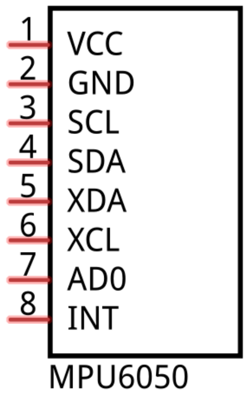
In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor.

Component knowledge

MPU6050

MPU6050 Sensor Module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, a 3-axis Accelerometer and a DMP (Digital Motion Processor) all in a small package. The settings of the Accelerometer and Gyroscope of MPU6050 can be changed. A precision wide range digital temperature sensor is also integrated to compensate data readings for changes in temperature, and temperature values can also be read. The MPU6050 Module follows the I2C communication protocol and the default address is 0x68.





The port description of the MPU6050 Module is as follows:

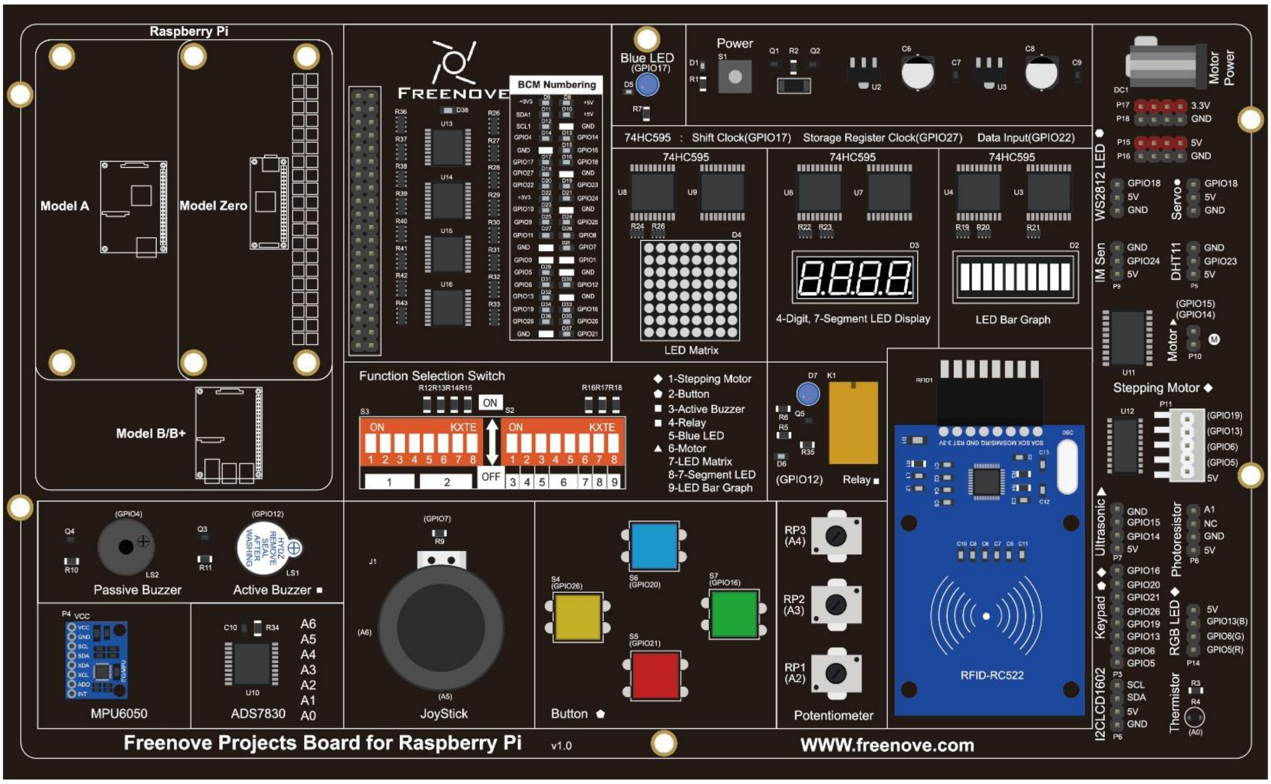
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication data pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69
INT	8	Output interrupt pin

For more detail, please refer to the MPU6050 datasheet.

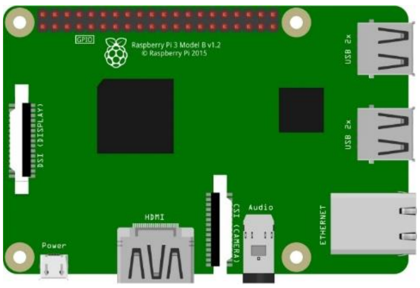
MPU6050 is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.

Component List

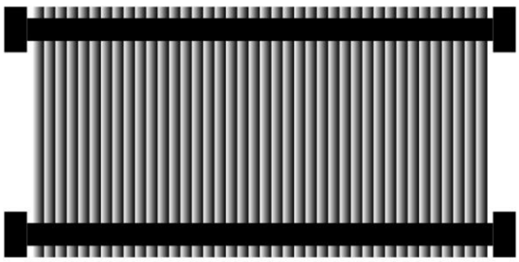
Freenove Projects Board for Raspberry Pi



Raspberry Pi

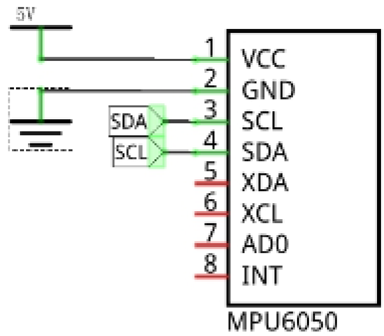


GPIO Ribbon Cable



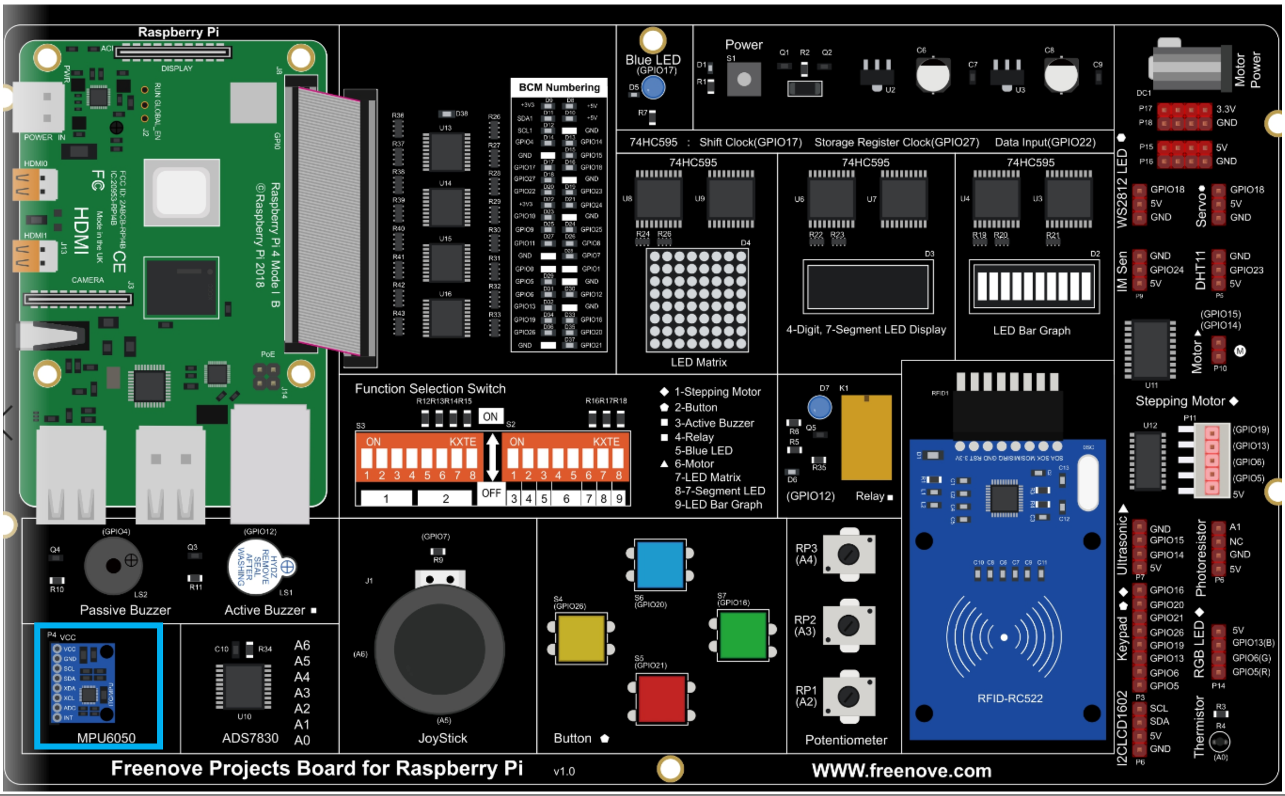
Circuit

Schematic diagram



Hardware connection.

After running the program, hold the board and turn it over to observe the changes in the running results.



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

Code

In this project, we will read the acceleration data and gyroscope data of MPU6050, and print them out.

C Code 24.1 MPU6050RAW

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

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

1. Use cd command to enter 24_MPU6050 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/24_MPU6050
```

2. Use following command to compile "MPU6050RAW.c", "MPU6050.cpp" and "I2Cdev.cpp", and generate executable file "MPU6050RAW".

```
gcc MPU6050RAW.cpp MPU6050.cpp I2Cdev.cpp -o MPU6050RAW
```

3. Then run the generated file "MPU6050RAW".

```
sudo ./MPU6050RAW
```

After the program runs, the Terminal will display active accelerometer and gyroscope data of the MPU6050, as well as the conversion to gravity acceleration and angular velocity as units of data. As shown in the following figure:

```
a/g:   1360     120  15840     -320   -193   -114
a/g: 0.08 g 0.01 g 0.97 g   -2.44 d/s -1.47 d/s -0.87 d/s
a/g:   1108     -88  15476     -354   -252   -115
a/g: 0.07 g -0.01 g 0.94 g   -2.70 d/s -1.92 d/s -0.88 d/s
a/g:   1344    -264  15764     -396   -236   -121
a/g: 0.08 g -0.02 g 0.96 g   -3.02 d/s -1.80 d/s -0.92 d/s
a/g:   1440    -180  15720     -375   -162   -114
a/g: 0.09 g -0.01 g 0.96 g   -2.86 d/s -1.24 d/s -0.87 d/s
a/g:   1436      56  16608     -400   -154   -136
a/g: 0.09 g 0.00 g 1.01 g   -3.05 d/s -1.18 d/s -1.04 d/s
a/g:   1008     144  14940     -345   -142   -129
a/g: 0.06 g 0.01 g 0.91 g   -2.63 d/s -1.08 d/s -0.98 d/s
```

The following is the program code:

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 accelgyro;      //creat MPU6050 class object

int16_t ax, ay, az;     //store acceleration data
int16_t gx, gy, gz;     //store gyroscope data

void setup() {
    // initialize device
    printf("Initializing I2C devices...\n");
```

```
    accelgyro.initialize();    //initialize MPU6050

    // verify connection
    printf("Testing device connections...\n");
    printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" : "MPU6050
connection failed\n");
}

void loop() {
    // read accel/gyro values of MPU6050
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    // display accel/gyro x/y/z values
    printf("a/g: %6hd %6hd %6hd   %6hd %6hd %6hd\n",ax,ay,az,gx,gy,gz);
    printf("a/g: %.2f g %.2f g %.2f g   %.2f d/s %.2f d/s %.2f d/s
\n",(float)ax/16384,(float)ay/16384,(float)az/16384,
        (float)gx/131,(float)gy/131,(float)gz/131);
}

int main()
{
    setup();
    while(1){
        loop();
    }
    return 0;
}
```

Two library files "**MPU6050**.h" and "**I2Cdev**.h" are used in the code and will be compiled with others. Class MPU6050 is used to operate the MPU6050 Sensor. When used, first it initiates an object.

```
MPU6050 accelgyro;      //creat MPU6050 class object
```

In the setup function, the MPU6050 is initialized and the result of the initialization will be tested.

```
void setup() {
    // initialize device
    printf("Initializing I2C devices...\n");
    accelgyro.initialize();    //initialize MPU6050

    // verify connection
    printf("Testing device connections...\n");
    printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" : "MPU6050
connection failed\n");
}
```

In the loop function, read the original data of MPU6050, display them and then convert the original data into the corresponding acceleration and angular velocity values, then display the converted data out.

```
void loop() {
    // read accel/gyro values of MPU6050
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    // display accel/gyro x/y/z values
    printf("a/g: %6hd %6hd %6hd   %6hd %6hd %6hd\n",ax,ay,az,gx,gy,gz);
    printf("a/g: %.2f g %.2f g %.2f g   %.2f d/s %.2f d/s %.2f d/s
\n",(float)ax/16384,(float)ay/16384,(float)az/16384,
        (float)gx/131,(float)gy/131,(float)gz/131);
}
```

Finally, the main functions, called setup function and loop function respectively.

```
int main()
{
    setup();
    while(1){
        loop();
    }
    return 0;
}
```

About class MPU6050:

Class MPU6050

This is a class library used to operate the MPU6050, which can directly read and set the MPU6050. Here are its functions:

`MPU6050()/MPU6050(uint8_t address);`

Constructor. The parameter is I2C address, and the default I2C address is 0x68.

`void initialize();`

Initialization function, used to wake up MPU6050. Range of accelerometer is ±2g and range of gyroscope is ±250 degrees/sec.

`void getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t* gz);`

Get the original data of accelerometer and gyroscope.

`int16_t getTemperature();`

Get the original temperature data of MPU6050.

Python Code 24.1 MPU6050RAW

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

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

1. Use cd command to enter 24_MPU6050 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/24_MPU6050
```

2. Use Python command to execute code "MPU6050RAW.py".

```
python MPU6050RAW.py
```

After the program runs, the Terminal will display active accelerometer and gyroscope data of the MPU6050, as well as the conversion to gravity acceleration and angular velocity as units of data. As shown in the following figure:

```
a/g:1326        -160    16548   -48     -25     -16
a/g:0.08 g      -0.01 g 1.01 g  -0.37 d/s       -0.19 d/s       -0.12 d/s
a/g:1174        -116    15972   -44     -25     -17
a/g:0.07 g      -0.01 g 0.97 g  -0.34 d/s       -0.19 d/s       -0.13 d/s
a/g:1134        -130    16066   -45     -21     -17
a/g:0.07 g      -0.01 g 0.98 g  -0.34 d/s       -0.16 d/s       -0.13 d/s
a/g:1234        -76     15976   -45     -30     -16
a/g:0.08 g      -0.00 g 0.98 g  -0.34 d/s       -0.23 d/s       -0.12 d/s
a/g:996 -88     15748   -45     -22     -16
a/g:0.06 g      -0.01 g 0.96 g  -0.34 d/s       -0.17 d/s       -0.12 d/s
a/g:1196        -174    16182   -46     -25     -15
a/g:0.07 g      -0.01 g 0.99 g  -0.35 d/s       -0.19 d/s       -0.11 d/s
```

The following is the program code:

```
import MPU6050
import time

mpu = MPU6050.MPU6050()      # instantiate a MPU6050 class object
accel = [0]*3                # define an arry to store accelerometer data
gyro = [0]*3                 # define an arry to store gyroscope data
def setup():
    mpu.dmp_initialize()     # initialize MPU6050

def loop():
    while(True):
        accel = mpu.get_acceleration()      # get accelerometer data
        gyro = mpu.get_rotation()           # get gyroscope data
        print("a/g:%d\t%d\t%d\t%d\t%d\t%d
"%(accel[0],accel[1],accel[2],gyro[0],gyro[1],gyro[2]))
        print("a/g:%.2f g\t%.2f g\t%.2f g\t%.2f d/s\t%.2f d/s\t%.2f
d/s"%(accel[0]/16384.0,accel[1]/16384.0,
            accel[2]/16384.0,gyro[0]/131.0,gyro[1]/131.0,gyro[2]/131.0))
        time.sleep(0.1)

if __name__ == '__main__':     # Program entrance
    print("Program is starting ... ")
```

```
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        pass
```

A module "**MPU6050**.py" is used in the code. The module includes a class used to operate MPU6050. When using it, first initiate an object.

```
mpu = MPU6050.MPU6050()      # instantiate a MPU6050 class object
```

In the setup function, the MPU6050 is initialized.

```
def setup():
    mpu.dmp_initialize()
```

In the loop function, read the original data of MPU6050, display them and then convert the original data into the corresponding acceleration and angular velocity values, then display the converted data out.

```
def loop():
    while(True):
        accel = mpu.get_acceleration()      #get accelerometer data
        gyro = mpu.get_rotation()           #get gyroscope data
        print("a/g:%d\t%d\t%d\t%d\t%d\t%d
"%(accel[0],accel[1],accel[2],gyro[0],gyro[1],gyro[2]))
        print("a/g:%.2f g\t%.2f g\t%.2f g\t%.2f d/s\t%.2f d/s\t%.2f
d/s"%(accel[0]/16384.0,accel[1]/16384.0,
            accel[2]/16384.0,gyro[0]/131.0,gyro[1]/131.0,gyro[2]/131.0))
        time.sleep(0.1)
```

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

```
def __init__(self, a_bus=1, a_address=C.MPU6050_DEFAULT_ADDRESS,
                 a_xAOff=None, a_yAOff=None, a_zAOff=None, a_xGOff=None,
                 a_yGOff=None, a_zGOff=None, a_debug=False):
```

Constructor

```
def dmp_initialize(self):
```

Initialization function, used to wake up MPU6050. Range of accelerometer is ±2g and range of gyroscope is ±250 degrees/sec.

```
def get_acceleration(self):   &   def get_rotation(self):
```

Get the original data of accelerometer and gyroscope.

For details of more relevant member functions, please refer to MPU6050.py in the code folder.

Chapter 25 RFID

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

Project 25.1 RFID

In this project, we will use RC522 RFID card reader to read and write the M1-S50 card.

Component Knowledge

RFID

RFID（Radio Frequency Identification）is a form of wireless communication technology. A complete RFID system is generally composed of a transponder and a reader. Generally, the transponder may be known as a tag, and each tag has a unique code, which is attached to an object to identify the target object. The reader is a device that reads (or writes) information in the tag.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products, among which, Passive RFID products are the earliest, the most mature and most widely used products in the market. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of them are classified as close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13-26mA(DC)\3.3V
Idle current	10-13mA(DC)\3.3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13.56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm

Operation temperature	20-80 degrees(Celsius)
Storage temperature	40-85 degrees (Celsius)
Operation humidity	5%-95%(Relative humidity)

Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years.

The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63).

And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control put in its last block, that is, Block 3, which is also known as sector trailer. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the card serial number and vendor code, which has been solidified and can't be changed. Except the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

(1) used as general data storage and can be operated for reading and writing data.

(2) used as data value, and can be operated for initializing, adding, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, a 4-byte access control and a 6-byte password B. For example, the control block of a brand new card is as follows:

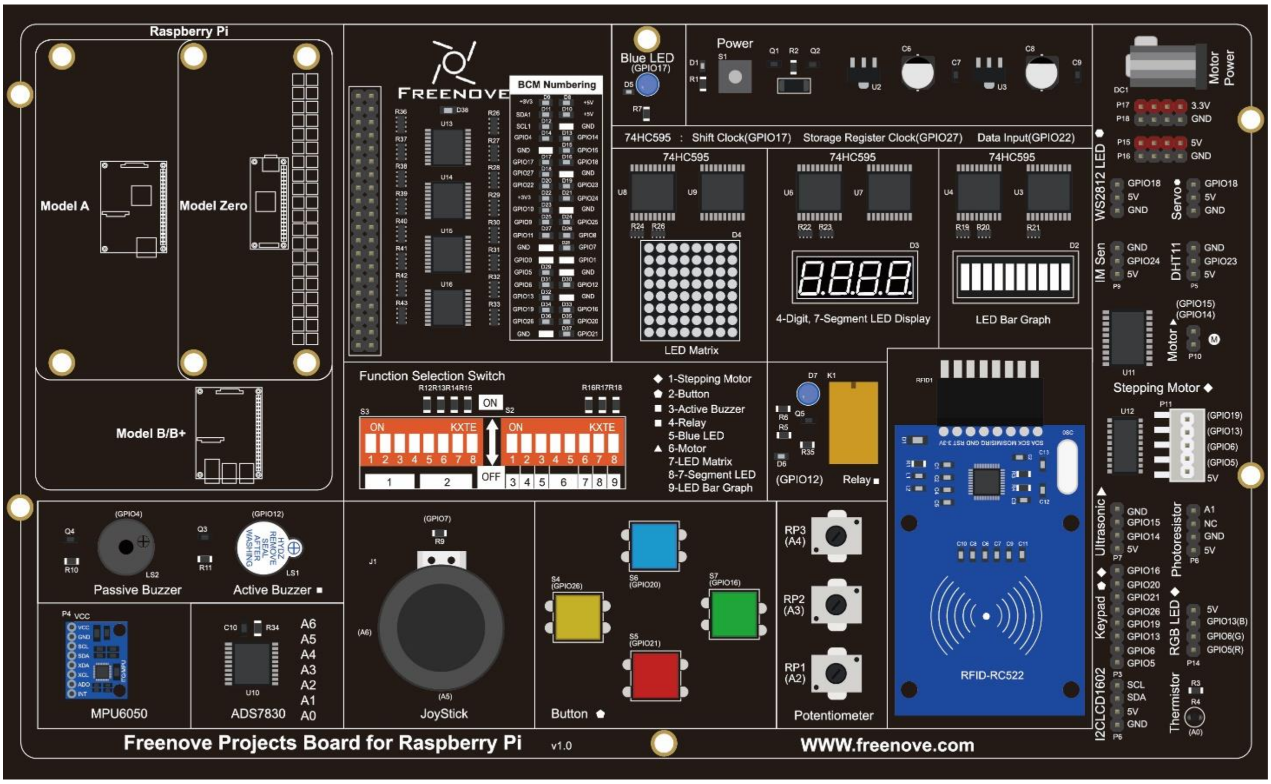
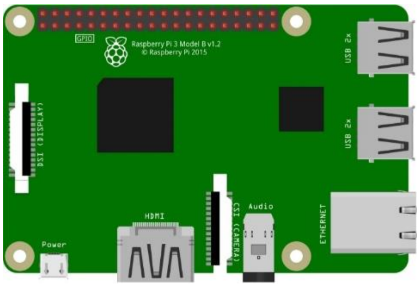
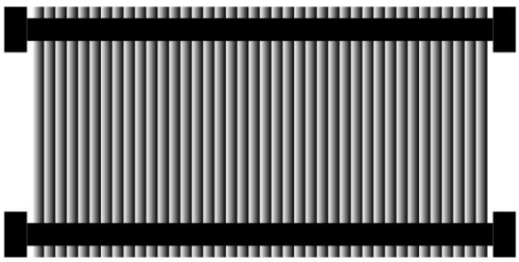
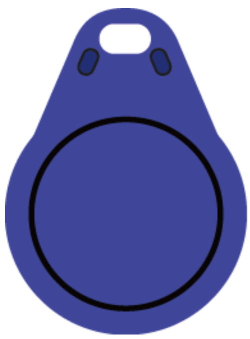
A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally 0A1A2A3A4A5 for password A and B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations,

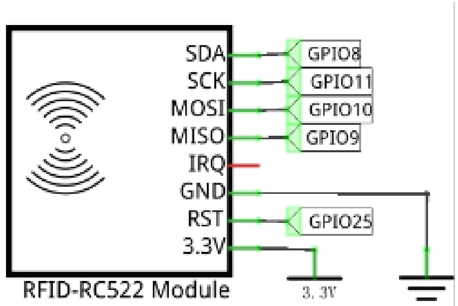
"read" and "write" for control blocks.
For more details about how to set data blocks and control blocks, please refer to Datasheet.
By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read, so if you choose to verify password A but forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**
For Mifare1 S50 card equipped in Freenove RFID Kit, the default password A and B are both FFFFFFFFFFFF.

Component List

<table>
<tr><td colspan="2">Freenove Projects Board for Raspberry Pi
</td></tr>
<tr><td>Raspberry Pi
</td><td>GPIO Ribbon Cable
</td></tr>
<tr><td>Mifare1 S50 Standard card
</td><td>Mifare1 S50 Non-standard card
</td></tr>
</table>

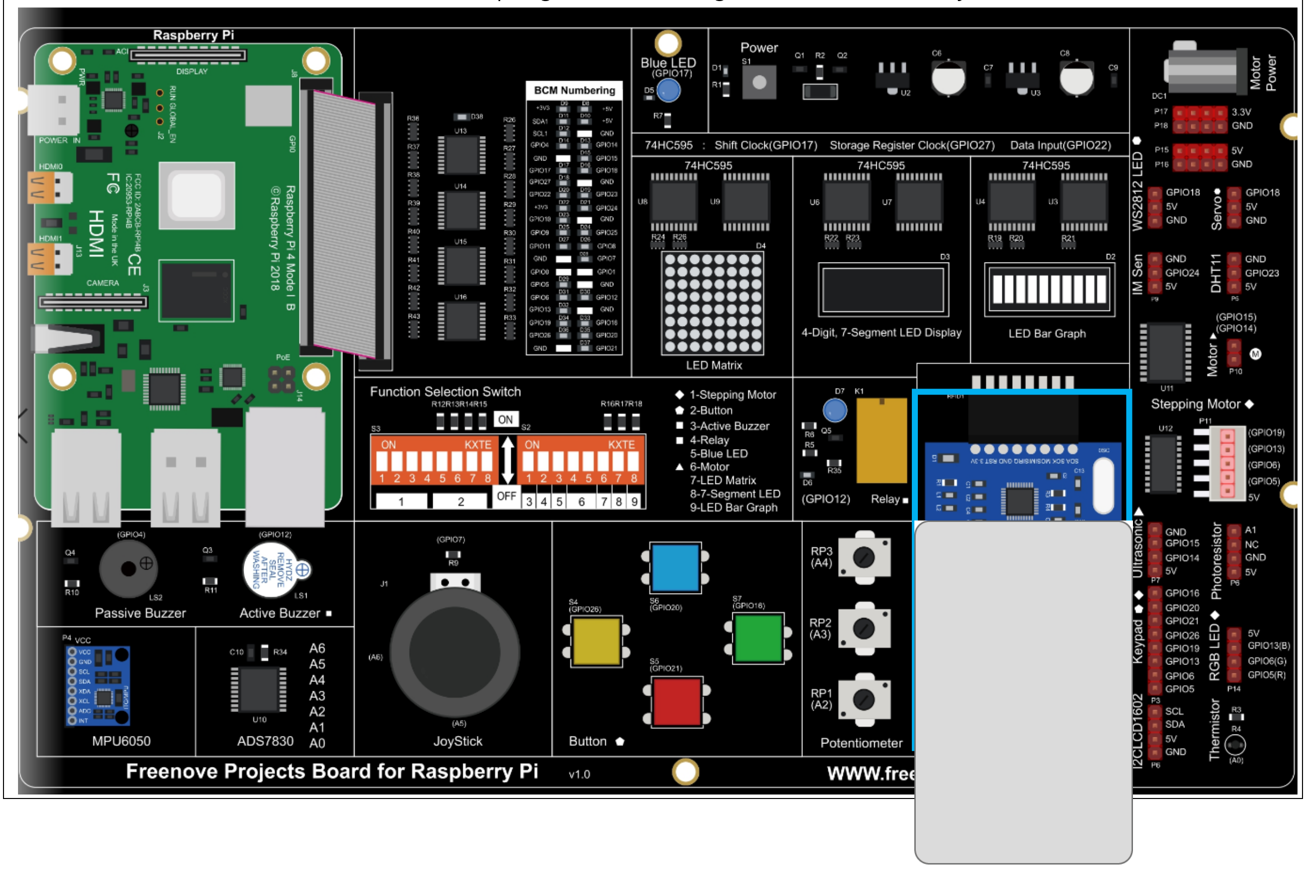
Circuit

Schematic diagram



Hardware connection.

Put RFID card down on here. When the program is running, don't move it away.



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

Configure SPI

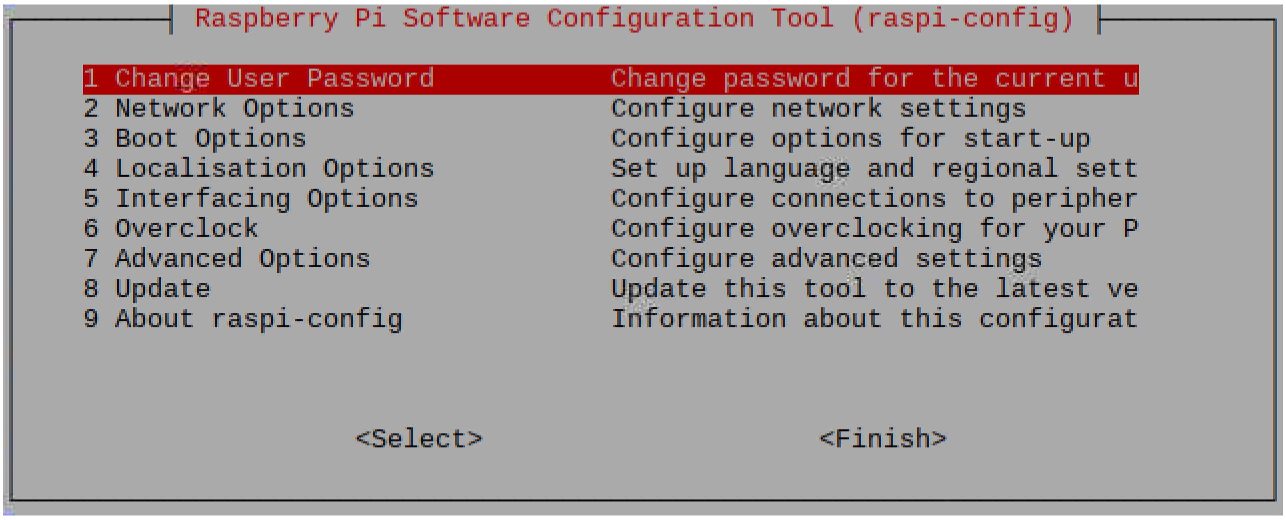
Enable SPI

The SPI interface of raspberry pi is closed by default. You need to open it manually. You can enable the SPI interface in the following way.

Type the following command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options”→“P4 SPI”→“Yes”→“Finish” in order and then restart your RPi. Then the SPI module is started.

Type the following command to check whether the module SPI is loaded successfully:

```
ls /dev/sp*
```

The following result indicates that the module SPI has been loaded successfully:

Install Python module SPI-Py

If you use Python language to write the code, please follow the steps below to install the module SPI-Py. If you use C/C++ language, you can skip this step.

Open the terminal and type the following command to install:

```
git clone https://github.com/Freenove/SPI-Py
cd SPI-Py
sudo python3 setup.py install
sudo python2 setup.py install
```

Code

The project code uses human-computer interaction command line mode to read and write the M1-S50 card.

C Code 25.1 RFID

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

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

1. Use cd command to enter 25_RFID directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/25_RFID
```

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

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

3. Then run the generated file "RFID".

```
sudo ./RFID
```

After the program runs, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo sh ./build.sh
Build finished!
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo ./RFID
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>
```

Here, type the command “quit” to exit the program.

Type command "scan", and then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522>scan
Scanning
................................................................................
.............Card detected    0xE6 0xCF 0x5C 0x8E, Check Sum = 0xFB
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>E6CF5C8E>
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
        read <blockstart>
        dump
        halt
        clean <blockaddr>
        write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to display all the data from blockstart address to the end of the sector. For example, sector 0 contains data block 0,1,2,3. Using the command "read 0" can display all contents of data block 0,1,2,3. Using the command "read 1" can display all contents of data block 1,2,3. As is shown below:

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
RC522>E6CF5C8E>read 1
read
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
RC522>E6CF5C8E>
```

Command "dump" is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write "data" to data block with address "address", where the address range is 0-63 and the data length is 0-16. For example, if you want to write the string "Freenove" to the data block with address "1", you can type the following command.

```
write 1 Freenove
```

```
RC522>E6CF5C8E>write 1 Freenove
write
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to write block 1 with 8 byte data...OK
```

Read the contents of this sector and check the data just written.

```
read 0
```

The following results indicate that the string "Freenove" has been written successfully into the data block 1.

```
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 46 72 65 65 6e 6f 76 65 00 00 00 00 00 00 00 00 : Freenove........
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
```

Command "clean <address>" is used to remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

```
clean 1
```

```
RC522>E6CF5C8E>clean 1
clean
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to clean block 1...OK
```

Read the contents of data blocks in this sector again to check whether the data is erased. The following results indicate that the contents of data block 1 have been erased.

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
```

Command "halt" is used to quit the selection state of the card.

```
RC522>E6CF5C8E>halt
halt
Halt
RC522>
```

The following is the program code:

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include <stdlib.h>
#include "mfrc522.h"
#define DISP_COMMANDLINE() printf("RC522>")

int scan_loop(uint8_t *CardID);
int tag_select(uint8_t *CardID);

int main(int argc, char **argv) {
    MFRC522_Status_t ret;
    //Recognized card ID
    uint8_t CardID[5] = { 0x00, };
    uint8_t tagType[16] = {0x00,};
    static char command_buffer[1024];

    ret = MFRC522_Init('B');
    if (ret < 0) {
        printf("Failed to initialize.\r\nProgram exit.\r\n");
        exit(-1);
    }

    printf("User Space RC522 Application\r\n");

    while (1) {
```

```
        /*Main Loop Start*/
        DISP_COMMANDLINE();

        scanf("%s", command_buffer);
        if (strcmp(command_buffer, "scan") == 0) {
            puts("Scanning ... ");
            while (1) {
                ret = MFRC522_Request(PICC_REQIDL, tagType);
                if (ret == MI_OK) {
                    printf("Card detected!\r\n");
                    ret = MFRC522_Anticoll(CardID);
                    if(ret == MI_OK){
                        ret = tag_select(CardID);
                        if (ret == MI_OK) {
                            ret = scan_loop(CardID);
                            if (ret < 0) {
                                printf("Card error...\r\n");
                                break;
                            } else if (ret == 1) {
                                puts("Halt...\r\n");
                                break;
                            }
                        }
                    }
                    else{
                        printf("Get Card ID failed!\r\n");
                    }
                }
                MFRC522_Halt();
            }
            MFRC522_Halt();
            MFRC522_Init('B');
        } else if (strcmp(command_buffer, "quit") == 0
                || strcmp(command_buffer, "exit") == 0) {
            return 0;
        } else {
            puts("Unknown command");
            puts("scan:scan card and dump");
            puts("quit:exit program");
        }
        /*Main Loop End*/
    }
}
int scan_loop(uint8_t *CardID) {
```

```
    while (1) {

        char input[32];
        int block_start;
        DISP_COMMANDLINE();
        printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2], CardID[3]);
        scanf("%s", input);
        puts((char*)input);
        if (strcmp(input, "halt") == 0) {
            MFRC522_Halt();
            return 1;
        } else if (strcmp(input, "dump") == 0) {
            if (MFRC522_Debug_CardDump(CardID) < 0)
                return -1;
        } else if (strcmp(input, "read") == 0) {
            scanf("%d", &block_start);
            if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
                return -1;
            }
        } else if(strcmp(input, "clean") == 0){
            char c;
            scanf("%d", &block_start);
            while ((c = getchar()) != '\n' && c != EOF)
                ;
            if (MFRC522_Debug_Clean(CardID, block_start)) {
                return -1;
            }

        } else if (strcmp(input, "write") == 0) {
            char write_buffer[256];
            size_t len = 0;
            scanf("%d", &block_start);
            scanf("%s",write_buffer);
            if (len >= 0) {
                if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
                        strlen(write_buffer)) < 0) {
                    return -1;
                }
            }
        } else {

            printf(
                    "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
```

```
                    "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
            //return 0;
        }
    }
    return 0;

}
int tag_select(uint8_t *CardID) {
    int ret_int;
    printf(
            "Card UID: 0x%02X 0x%02X 0x%02X 0x%02X, Check Sum = 0x%02X\r\n",
            CardID[0], CardID[1], CardID[2], CardID[3], CardID[4]);
    ret_int = MFRC522_SelectTag(CardID);
    if (ret_int == 0) {
        printf("Card Select Failed\r\n");
        return -1;
    } else {
        printf("Card Selected, Type:%s\r\n",
                MFRC522_TypeToString(MFRC522_ParseType(ret_int)));
    }
    ret_int = 0;
    return ret_int;
}
```

In the code, first initialize the MFRC522. If the initialization fails, the program will exit.

```
    ret = MFRC522_Init('B');
    if (ret < 0) {
        printf("Failed to initialize.\r\nProgram exit.\r\n");
        exit(-1);
    }
```

In the main function, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```
        scanf("%s", command_buffer);
        if (strcmp(command_buffer, "scan") == 0) {
            puts("Scanning ... ");
            while (1) {
                ret = MFRC522_Request(PICC_REQIDL, tagType);
                if (ret == MI_OK) {
                    printf("Card detected!\r\n");
                    ret = MFRC522_Anticoll(CardID);
                    if(ret == MI_OK){
```

```
                    ret = tag_select(CardID);
                    if (ret == MI_OK) {
                        ret = scan_loop(CardID);
                        if (ret < 0) {
                            printf("Card error...\r\n");
                            break;
                        } else if (ret == 1) {
                            puts("Halt...\r\n");
                            break;
                        }
                    }
                }
                else{
                    printf("Get Card ID failed!\r\n");
                }
            }
            MFRC522_Halt();
        }
        MFRC522_Halt();
        MFRC522_Init('B');
    } else if (strcmp(command_buffer, "quit") == 0
            || strcmp(command_buffer, "exit") == 0) {
        return 0;
    } else {
        puts("Unknown command");
        puts("scan:scan card and dump");
        puts("quit:exit program");
    }
    /*Main Loop End*/
```

The function scan_loop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```
int scan_loop(uint8_t *CardID) {

    while (1) {

        char input[32];
        int block_start;
        DISP_COMMANDLINE();
        printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2], CardID[3]);
        scanf("%s", input);
        puts((char*)input);
        if (strcmp(input, "halt") == 0) {
            MFRC522_Halt();
```

```
                return 1;
            } else if (strcmp(input, "dump") == 0) {
                if (MFRC522_Debug_CardDump(CardID) < 0)
                    return -1;
            } else if (strcmp(input, "read") == 0) {
                scanf("%d", &block_start);
                if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
                    return -1;
                }
            } else if(strcmp(input, "clean") == 0){
                char c;
                scanf("%d", &block_start);
                while ((c = getchar()) != '\n' && c != EOF)
                    ;
                if (MFRC522_Debug_Clean(CardID, block_start)) {
                    return -1;
                }

            } else if (strcmp(input, "write") == 0) {
                char write_buffer[256];
                size_t len = 0;
                scanf("%d", &block_start);
                scanf("%s",write_buffer);
                if (len >= 0) {
                    if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
                            strlen(write_buffer)) < 0) {
                        return -1;
                    }
                }
            } else {

                printf(
                        "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
                            "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
                //return 0;
            }
        }
        return 0;

}
```

Python Code 25.1 RFID

There are two code files for this project. They are respectively under Python2 folder and Python3 folder. **Their functions are the same, but they are not compatible.** Code under Python2 folder can only run on Python2. And code under Python3 folder can only run on Python3.

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

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

1. Use cd command to enter RFID directory of Python code.

   If you use Python2, it is needed to enter Python2 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/25_RFID/Python2
```

   If you use Python3, it is needed to enter Python3 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/25_RFID/Python3
```

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

```
python RFID.py
```

After the program runs, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3 $ python RFID.py
Program is starting ...
Press Ctrl-C to exit.
RC522>
```

Here, if you need to exit the program, you type the command quit.

Type command "scan", then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522> scan
scan
Scanning ...
Card detected
Card UID: ['0xe6', '0xcf', '0x5c', '0x8e', '0xfb']
Size: 8
RC522> E6CF5C8EFB>
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
        read <blockstart>
        dump
        halt
        clean <blockaddr>
        write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. As is shown below:

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to read the data of data block with address "blockstart". For example, using command "read 0" can display the content of data block 0. Using the command "read 1" can display the content of data block 1. As is shown below:

```
RC522> E6CF5C8EFB> read 0
['read', '0']
Sector 0 :  e6 cf 5c 8e fb 8 4 0 62 63 64 65 66 67 68 69  | ���cdefghi
RC522> E6CF5C8EFB> read 1
['read', '1']
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
RC522> E6CF5C8EFB>
```

Command “dump” is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write “data" to data block with address “address”, where the address range is 0-63 and the data length is 0-16. In the process of writing data to the data block, both the contents of data block before written and after written will be displayed. For example, if you want to write the string "Freenove" to the data block with address “1”, you can type the following command.

```
write 1 Freenove
```

```
RC522> E6CF5C8EFB> write 1 Freenove
['write', '1', 'Freenove']
Before writing , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
4 backdata &0x0F == 0x0A 10
Data written
After written , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
```

Command “clean <address>” is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

```
clean 1
```

```
RC522> E6CF5C8EFB> clean 1
['clean', '1']
Before cleaning , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
```

Command “halt” is used to quit the selection state of the card.

```
RC522> E6CF5C8EFB> halt
['halt']
RC522>
```

The following is the program code (python2 code):

```
import RPi.GPIO as GPIO
import MFRC522
import sys
import os

# Create an object of the class MFRC522
mfrc = MFRC522.MFRC522()

def dis_ConmandLine():
    print ("RC522>",end="")
def dis_CardID(cardID):
```

```
    print ("%2X%2X%2X%2X%2X>"%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]),end="")
def setup():
    print ("Program is starting ... "  )
    print ("Press Ctrl-C to exit.")
    pass

def loop():
    global mfrc3s
    while(True):
        dis_ConmandLine()
        inCmd = input()
        print (inCmd)
        if (inCmd == "scan"):
            print ("Scanning ... ")
            mfrc = MFRC522.MFRC522()
            isScan = True
            while isScan:
                # Scan for cards
                (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
                # If a card is found
                if status == mfrc.MI_OK:
                    print ("Card detected")
                # Get the UID of the card
                (status,uid) = mfrc.MFRC522_Anticoll()
                # If we have the UID, continue
                if status == mfrc.MI_OK:
                    print ("Card UID: "+ str(map(hex,uid)))
                    # Select the scanned tag
                    if mfrc.MFRC522_SelectTag(uid) == 0:
                        print ("MFRC522_SelectTag Failed!")
                    if cmdloop(uid) < 1 :
                        isScan = False

        elif inCmd == "quit":
            destroy()
            exit(0)
        else :
            print ("\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
program\n")

def cmdloop(cardID):
    pass
    while(True):
        dis_ConmandLine()
```

```
        dis_CardID(cardID)
        inCmd = input()
        cmd = inCmd.split(" ")
        print (cmd)
        if(cmd[0] == "read"):
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print ("Invalid Address!")
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print ("Authentication error")
                return 0

        elif cmd[0] == "dump":
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            mfrc.MFRC522_Dump_Str(key,cardID)

        elif cmd[0] == "write":
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print ("Invalid Address!")
            data = [0]*16
            if(len(cmd)<2):
                data = [0]*16
            else:
                data = cmd[2][0:17]
                data = map(ord,data)
                data = list(data)
                lenData = len(list(data))
                if lenData<16:
                    data+=[0]*(16-lenData)
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
```

```
                print ("Before writing , The data in block %d  is: "%(blockAddr))
                mfrc.MFRC522_Readstr(blockAddr)
                mfrc.MFRC522_Write(blockAddr, data)
                print ("After written , The data in block %d  is: "%(blockAddr))
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print ("Authentication error")
                return 0

        elif cmd[0] == "clean":
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print ("Invalid Address!")
            data = [0]*16
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
                print ("Before cleaning , The data in block %d  is: "%(blockAddr))
                mfrc.MFRC522_Readstr(blockAddr)
                mfrc.MFRC522_Write(blockAddr, data)
                print ("After cleaned , The data in block %d  is: "%(blockAddr))
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print ("Authentication error")
                return 0
        elif cmd[0] == "halt":
            return 0
        else :
            print ("Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n" "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n")

def destroy():
    GPIO.cleanup()

if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Ctrl+C captured, exit
        destroy()
```

In the code, first create an MFRC522 class object.

```
mfrc = MFRC522.MFRC522()
```

In the function loop, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```
        if (inCmd == "scan"):
            print "Scanning ... "
            isScan = True
            while isScan:
                ...........
                    if cmdloop(uid) < 1 :
                        isScan = False
        elif inCmd == "quit":
            destroy()
            exit(0)
        else :
            print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit 
program\n"
```

The function cmdloop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```
def cmdloop(cardID):
    pass
    while(True):
        dis_ConmandLine()
        dis_CardID(cardID)
        inCmd = raw_input()
        cmd = inCmd.split(" ")
        print cmd
        if(cmd[0] == "read"):
            ...........
        elif cmd[0] == "dump":
            ...........
        elif cmd[0] == "write":
            ...........
        elif cmd[0] == "clean":
            ...........
        elif cmd[0] == "halt":
            return 0
        else :
```

```
            print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
"\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"
```

The file "MFRC522.py" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself a Raspberry Pi Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interested in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.