

Important Information

Thank you for choosing Freenove products!

Getting Started

First, please read the **Read Me First.pdf** document in the unzipped folder you created.

If you have not yet downloaded the zip file, associated with this kit, please do so now and unzip it.

Get Support and Offer Input

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

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

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

support@freenove.com

Safety and Precautions

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

- Keep this product out of the reach of children under 6 years old.
- This product should be **used only when there is adult supervision present** as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. **Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.**
- When the product is turned ON, activated or tested, some parts will move or rotate. **To avoid injuries to hands and fingers keep them away from any moving parts!**
- It is possible that an improperly connected or shorted circuit may cause overheating. **Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down!** When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.



About Freenove

Freenove provides open source electronic products and services worldwide.

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

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

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

<http://www.freenove.com>

sale@freenove.com

Copyright

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



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

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



Contents

Important Information.....	1
Contents.....	3
Preface.....	1
ESP32-WROVER	2
Chapter 0 Ready (Important)	5
0.1 Installing Thonny	5
0.2 Basic Configuration of Thonny	10
0.3 Installing CH340	12
0.4 Burning Micropython Firmware	21
0.5 Testing codes	26
0.6 Thonny Common Operation	34
0.7 Note.....	39
Chapter 1 LED	41
Project 1.1 Blink	41
Chapter 2 Button & LED	53
Project 2.1 Button & LED	53
Project 2.2 MINI table lamp.....	60
Chapter 3 LED Bar	65
Project 3.1 Flowing Light	65
Chapter 4 Analog & PWM	71
Project 4.1 Breathing LED.....	71
Project 4.2 Meteor Flowing Light	78
Chapter 5 RGBLED	84
Project 5.1 Random Color Light.....	84
Project 5.2 Gradient Color Light.....	90
Chapter 6 NeoPixel	92
Project 6.1 NeoPixel	92
Project 6.2 Rainbow Light.....	98
Chapter 7 Buzzer.....	101
Project 7.1 Doorbell.....	101



Project 7.2 Alertor	107
Chapter 8 Serial Communication.....	109
Project 8.1 Serial Print.....	109
Project 8.2 Serial Read and Write	114
Chapter 9 AD/DA Converter	115
Project 9.1 Read the Voltage of Potentiometer.....	115
Chapter 10 TouchSensor.....	123
Project 10.1 Read Touch Sensor	123
Project 10.2 TouchLamp	128
Chapter 11 Potentiometer & LED.....	133
Project 11.1 Soft Light	133
Project 11.2 Soft Colorful Light	136
Project 11.3 Soft Rainbow Light.....	140
Chapter 12 Photoresistor & LED.....	144
Project 12.1 NightLamp	144
Chapter 13 Thermistor	148
Project 13.1 Thermometer	148
Chapter 14 Joystick	153
Project 14.1 Joystick	153
Chapter 15 74HC595 & LED Bar Graph.....	158
Project 15.1 Flowing Water Light.....	158
Chapter 16 74HC595 & 7-Segment Display.....	164
Project 16.1 7-Segment Display.....	164
Project 16.2 4-Digit 7-Segment Display.....	170
Chapter 16 74HC595 & LED Matrix	175
Project 16.3 LED Matrix.....	175
Chapter 17 Relay & Motor.....	184
Project 17.1 Relay & Motor	184
Project 17.2 Control Motor with Potentiometer	190
Chapter 18 Servo	198
Project 18.1 Servo Sweep.....	198
Project 18.2 Servo Knop	204

Chapter 19 Stepper Motor	208
Project 19.1 Stepping Motor	208
Chapter 20 LCD1602.....	216
Project 20.1 LCD1602	216
Chapter 21 Ultrasonic Ranging	223
Project 21.1 Ultrasonic Ranging	223
Project 21.2 Ultrasonic Ranging	229
Chapter 22 Matrix Keypad	232
Project 22.1 Matrix Keypad.....	232
Project 22.2 Keypad Door	237
Chapter 23 Infrared Remote	242
Project 23.1 Infrared Remote Control	242
Project 23.2 Control LED through Infrared Remote	248
Chapter 24 Hygrothermograph DHT11	253
Project 24.1 Hygrothermograph.....	253
Project 24.2 Hygrothermograph.....	259
Chapter 25 Infrared Motion Sensor.....	263
Project 25.1 Infrared Motion Detector with LED Indicator.....	263
Chapter 26 Attitude Sensor MPU6050	268
Project 26.1 Read a MPU6050 Sensor Module	268
Chapter 27 WiFi Working Modes	274
Project 27.1 Station mode.....	274
Project 27.2 AP mode.....	278
Project 27.3 AP+Station mode	282
Chapter 28 TCP/IP	286
Project 28.1 As Client.....	286
Project 28.2 As Server.....	297
Chapter 29 Soldering Circuit Board	302
Project 29.1 Soldering a Buzzer	302
Project 29.2 Soldering a Flowing Water Light	306
What's Next?.....	310

Preface

ESP32 is a micro control unit with integrated Wi-Fi launched by Espressif, which features strong properties and integrates rich peripherals. It can be designed and studied as an ordinary Single Chip Microcontroller(SCM) chip, or connected to the Internet and used as an Internet of Things device.

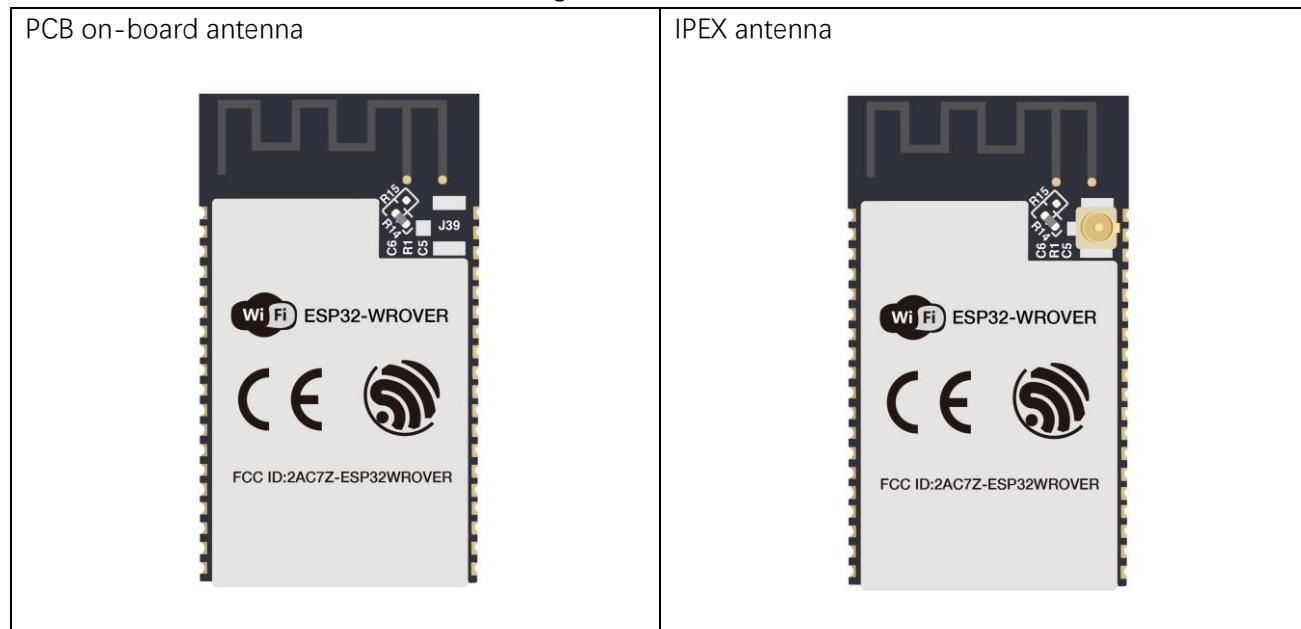
ESP32 can be developed both either with C/C++ language or micropython language. In this tutorial, we use micropython. With Micropython is as easy to learn as Python with little code, making it ideal for beginners. Moreover, the code of ESP32 is completely open-source, so beginners can quickly learn how to develop and design IOT smart household products including smart curtains, fans, lamps and clocks.

We divide each project into four parts, namely Component List, Component Knowledge, Circuit and Code. Component List helps you to prepare material for the experiment more quickly. Component Knowledge allows you to quickly understand new electronic modules or components, while Circuit helps you understand the operating principle of the circuit. And Code allows you to easily master the use of ESP32 and its accessory kit. After finishing all the projects in this tutorial, you can also use these components and modules to make products such as smart household, smart cars and robots to transform your creative ideas into prototypes and new and innovative products.

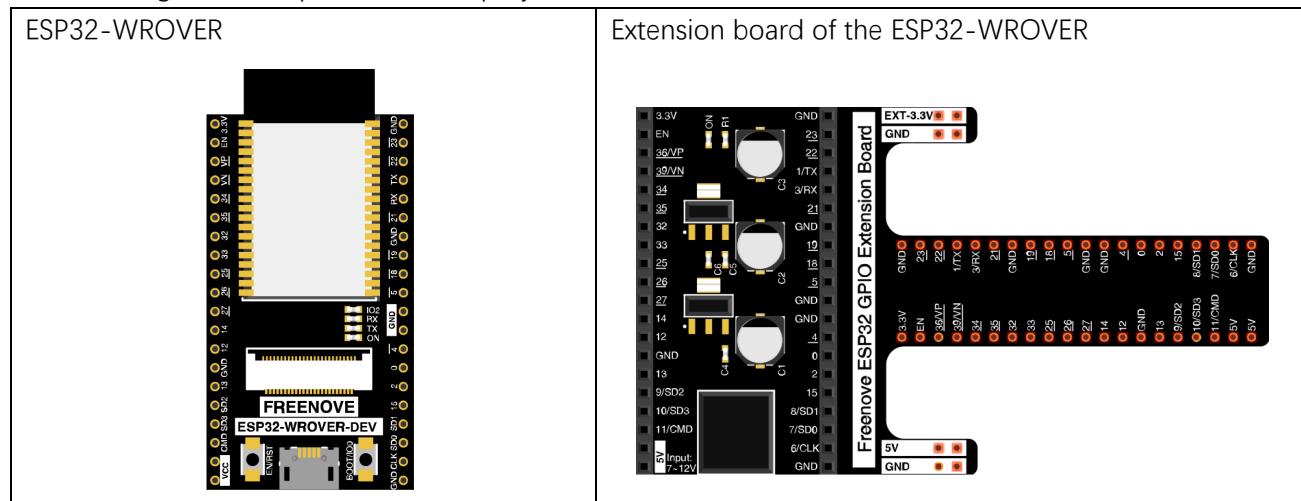
In addition, if you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through support@freenove.com

ESP32-WROVER

ESP32-WROVER has launched a total of two antenna packages, PCB on-board antenna and IPEX antenna respectively. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design. The IPEX antenna is a metal antenna derived from the integrated antenna of the chip module itself, which is used to enhance the signal of the module.



In this tutorial, the ESP32-WROVER is designed based on the IPEX antenna package. And we also design an extension board, so that you can use the ESP32 more easily in accordance with the circuit diagram provided. The followings are their photos. All the projects in this tutorial are studied with this ESP32-WROVER.



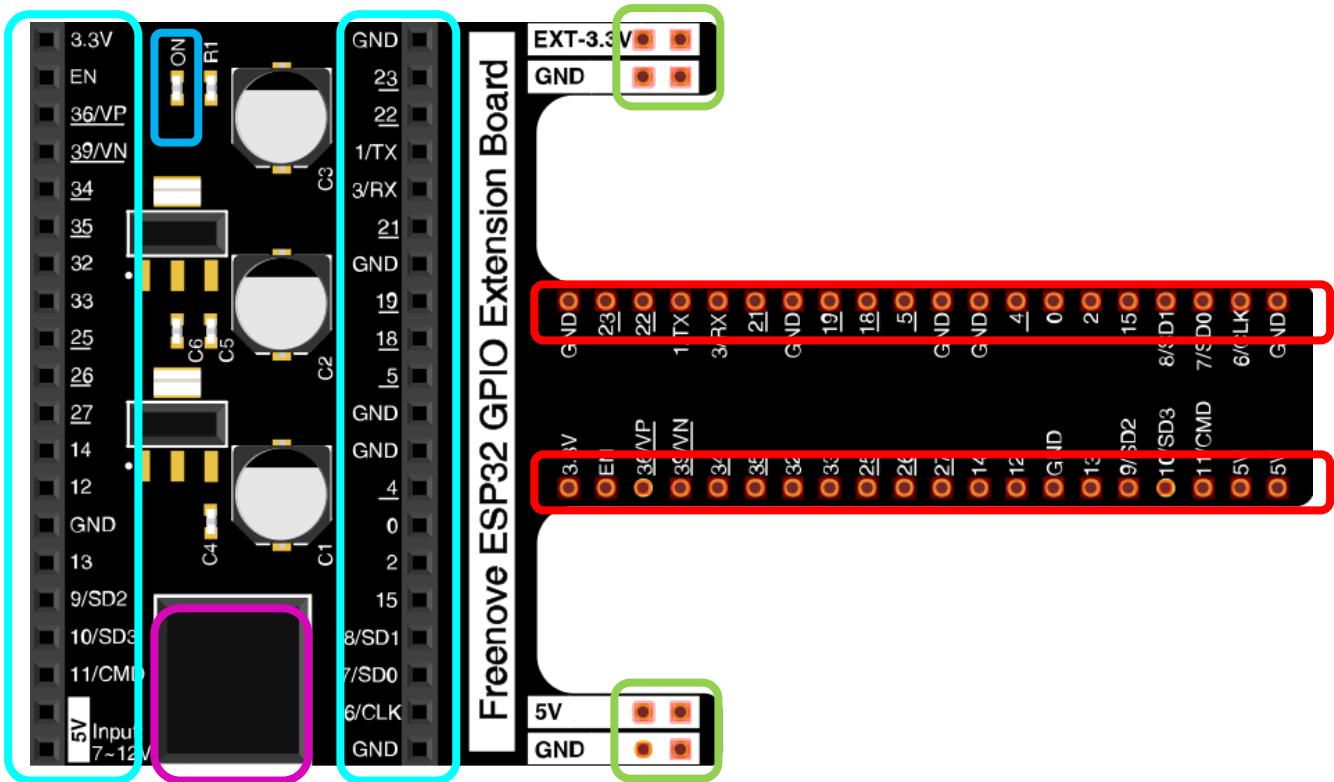
The hardware interfaces of ESP32-WROVER are distributed as follows:



Compare the left and right images. We've boxed off the resources on the ESP32-WROVER in different colors to facilitate your understanding of the ESP32-WROVER.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	Camera interface
	Reset button, Boot mode selection button
	USB port

The hardware interfaces of ESP32-WROVER are distributed as follows:



We've boxed off the resources on the ESP32-WROVER in different colors to facilitate your understanding of the ESP32-WROVER.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	GPIO interface of development board
	Power supplied by the extension board
	External power supply

In ESP32, GPIO is an interface to control peripheral circuit. For beginners, it is necessary to learn the functions of each GPIO. The following is an introduction to the GPIO resources of the ESP32-WROVER development board.

Later, we only use USB cable to power ESP32-WROVER in default.

In the whole tutorial, we don't use T extension to power ESP32-WROVER. So 5V and 3.3V (include EXT 3.3V) on the extension board are from ESP32-WROVER.

We can also use DC jack of extension board to power ESP32-WROVER. Then 5v and EXT 3.3v on extension board are from external power resource.

For more information, please visit: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

Chapter 0 Ready (Important)

Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

0.1 Installing Thonny

Thonny is a free, open-source software platform with compact size, simple interface, simple operation and rich functions, making it a Python IDE for beginners. In this tutorial, we use this IDE to develop ESP32 during the whole process.

Thonny supports various operating system, including Windows、Mac OS、Linux.

Downloading Thonny

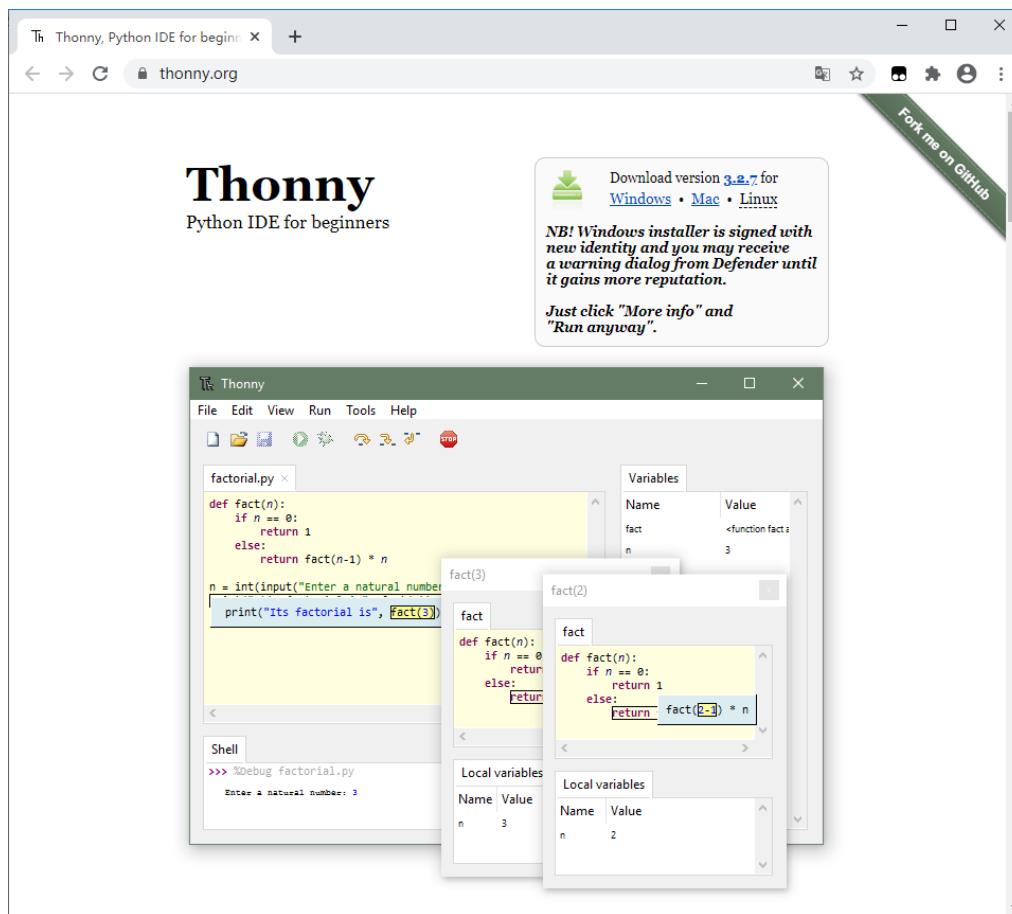
Official website of Thonny: <https://thonny.org>

Open-source code repositories of Thonny: <https://github.com/thonny/thonny>

Follow the instruction of official website to install Thonny or click the links below to download and install. (Select the appropriate one based on your operating system.)

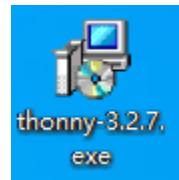
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.pkg
Linux	The latest version: Binary bundle for PC (Thonny+Python): bash <(wget -O - https://thonny.org/installer-for-linux) With pip: pip3 install thonny Distro packages (may not be the latest version): Debian, Rasbian, Ubuntu, Mint and others: sudo apt install thonny Fedora: sudo dnf install thonny

If you would not like to download the installation package, you can open “**Freenove_Ultimate_Kit_for_ESP32/MicroPython_Software**”, we have prepared it in advance.



Installing on Windows

The icon of Thonny after downloading is as below. Double click "thonny-3.2.7.exe".

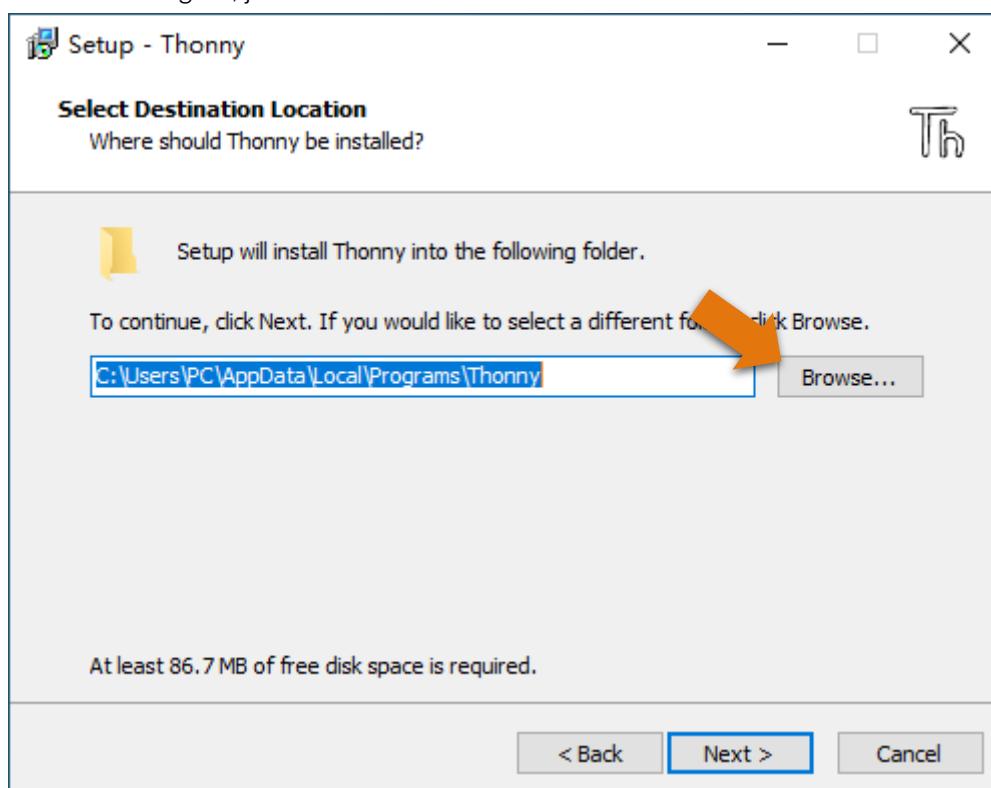


If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.

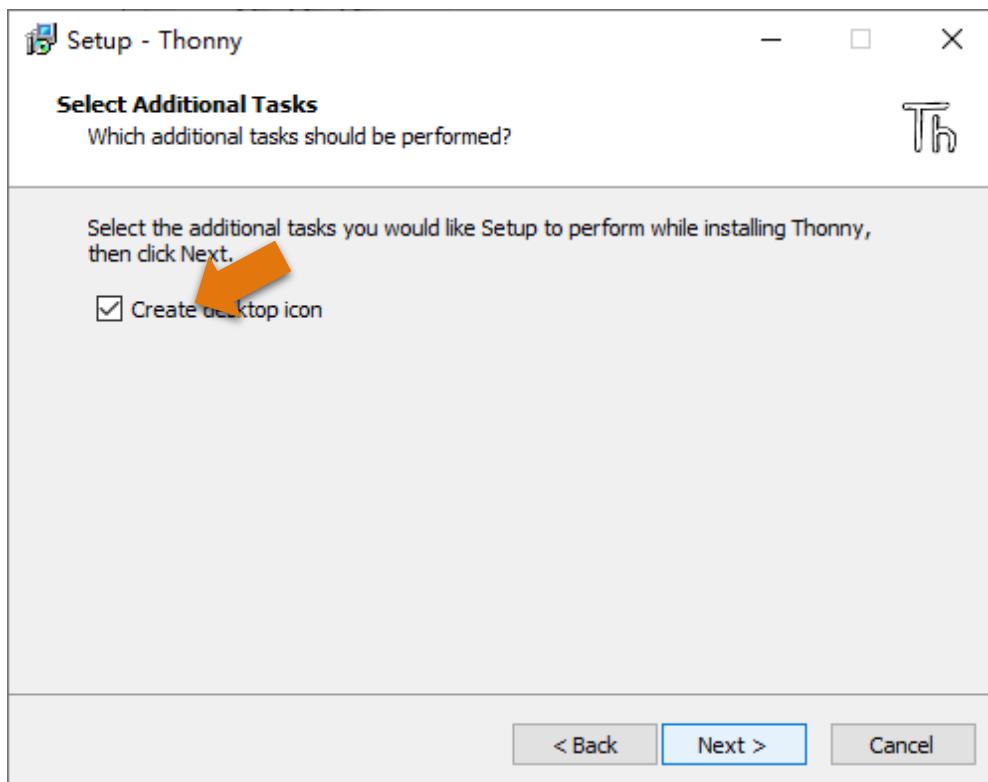


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

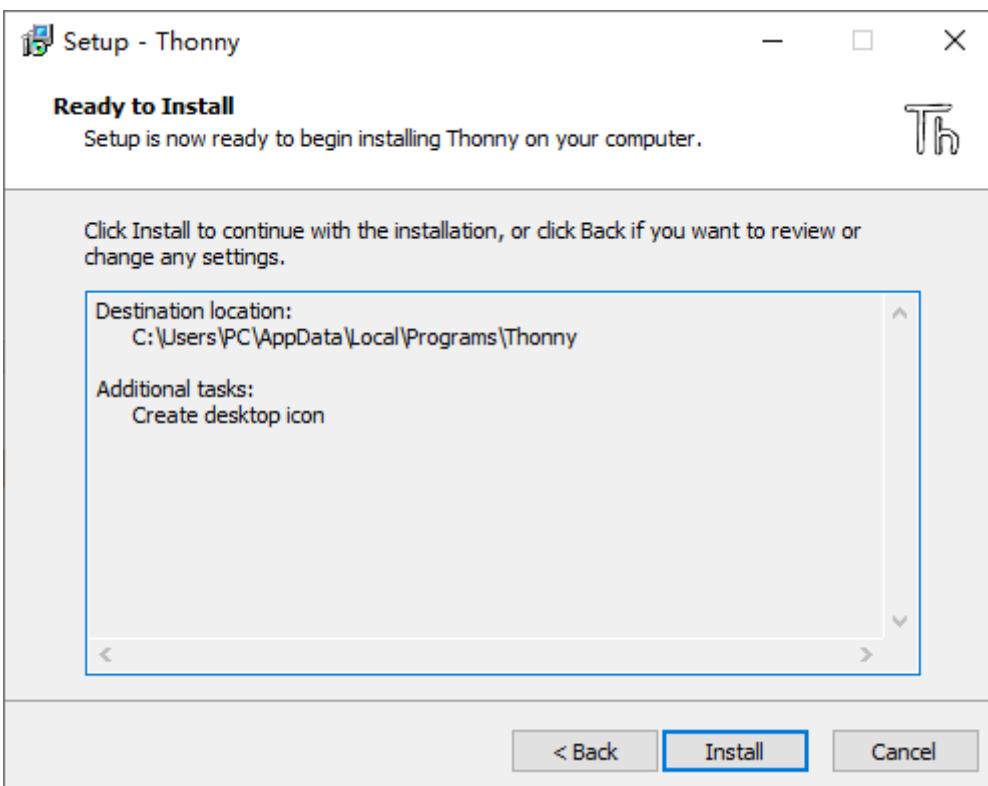
If you do not want to change it, just click "Next".



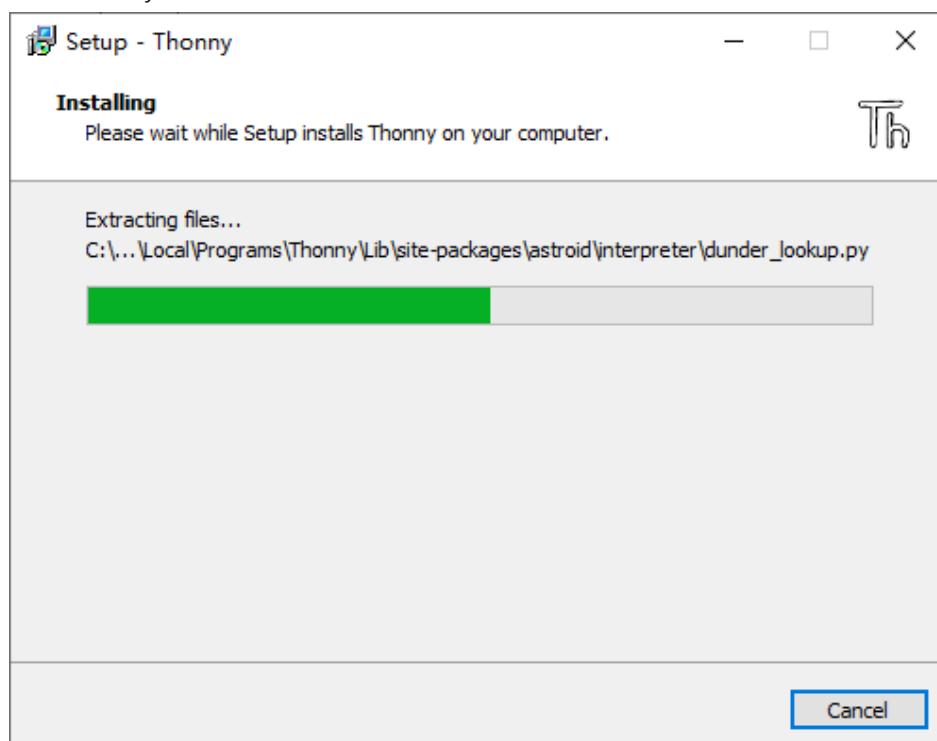
Check “Create desktop icon” and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



Click “install” to install the software.



During the installation process, you only need to wait for the installation to complete, and you must not click "Cancel", otherwise Thonny will fail to be installed.



Once you see the interface as below, Thonny has been installed successfully.

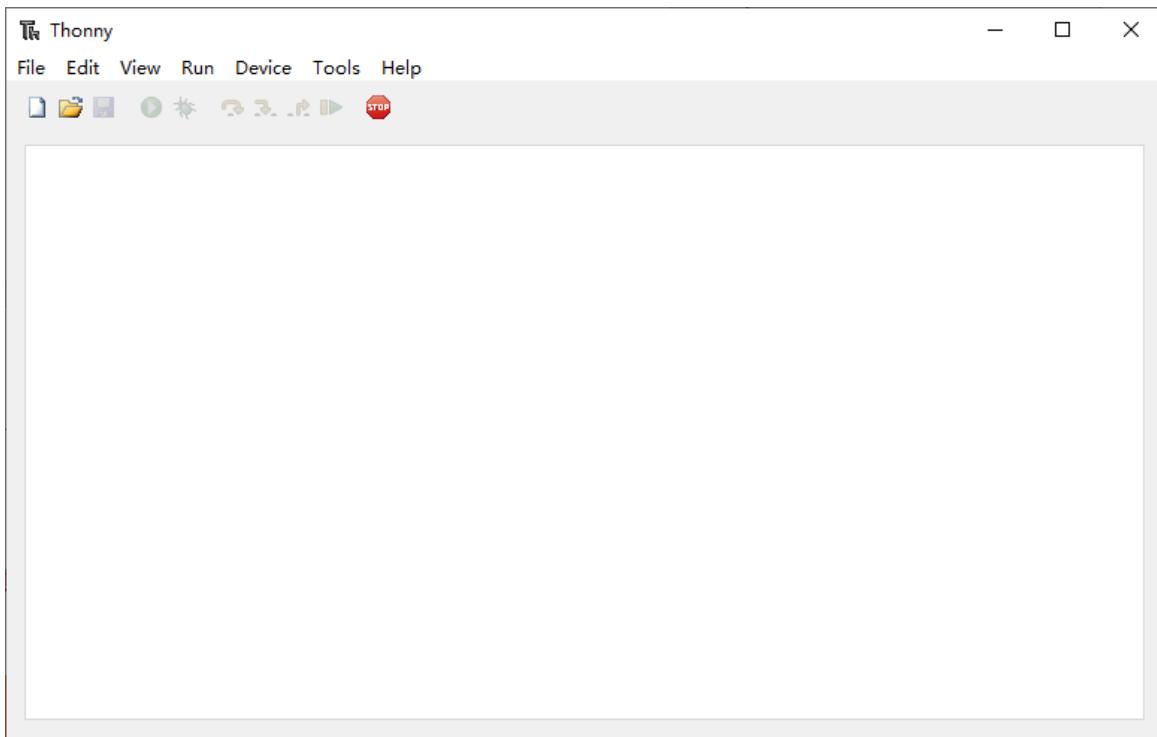


If you've checked "Create desktop icon" during the installation process, you can see the below icon on your desktop.

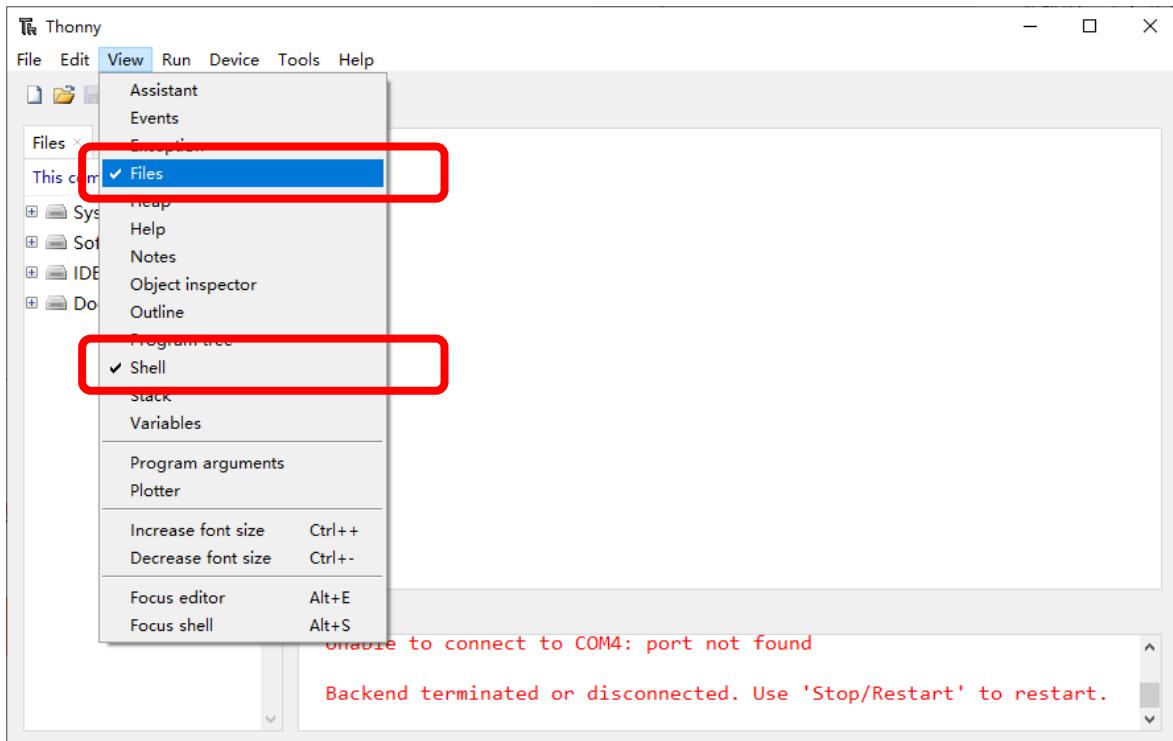


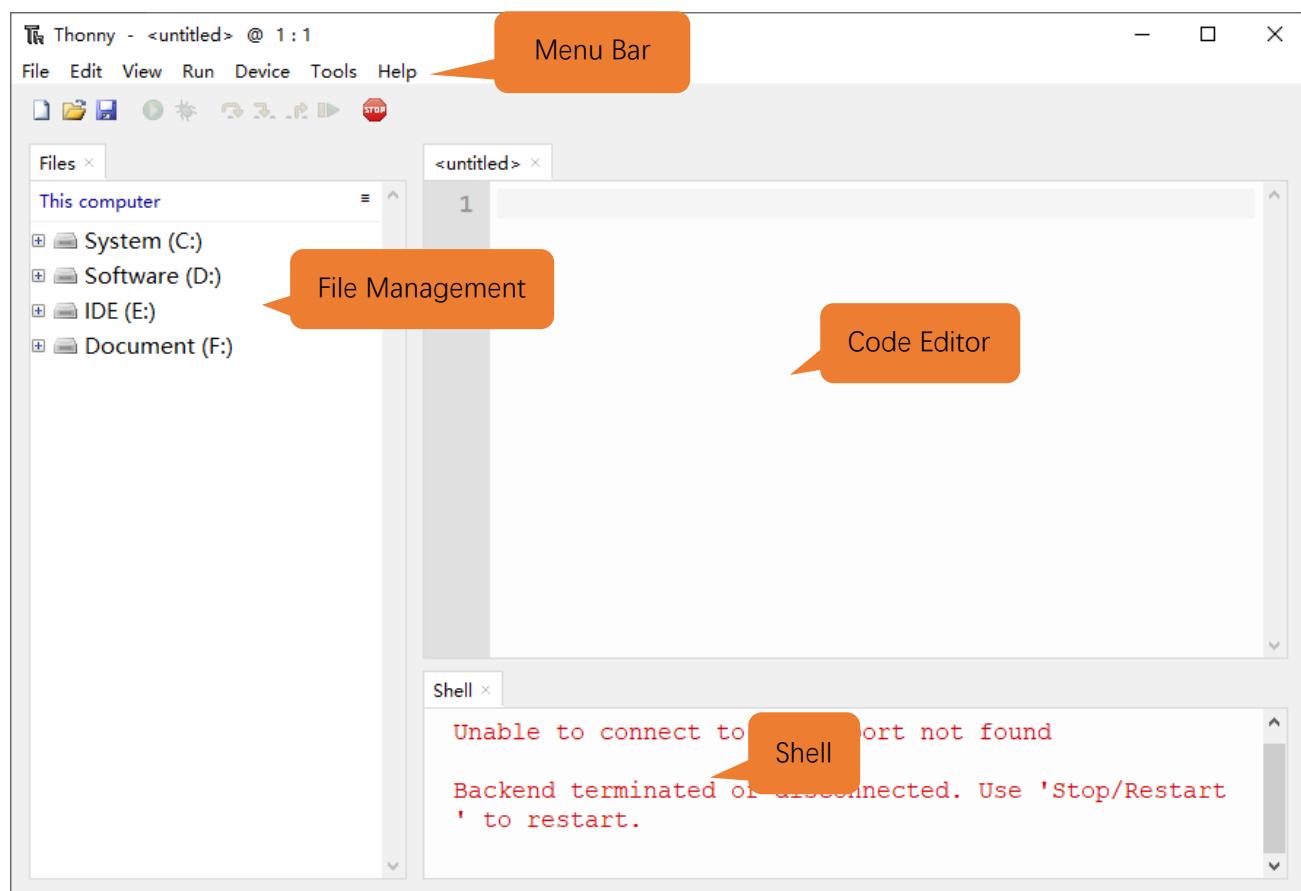
0.2 Basic Configuration of Thonny

Click the desktop icon of Thonny and you can see the interface of it as follows:



Select "View" → "Files" and "Shell".



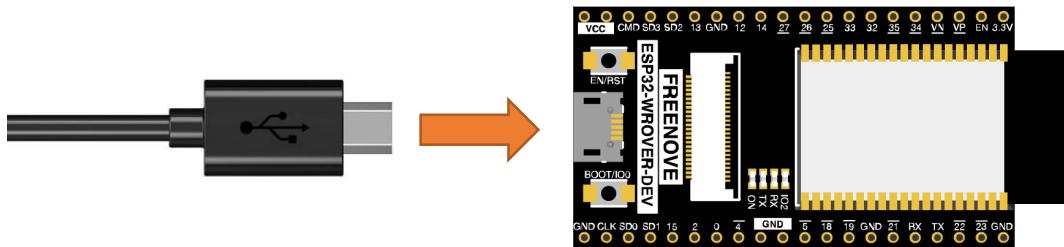


0.3 Installing CH340

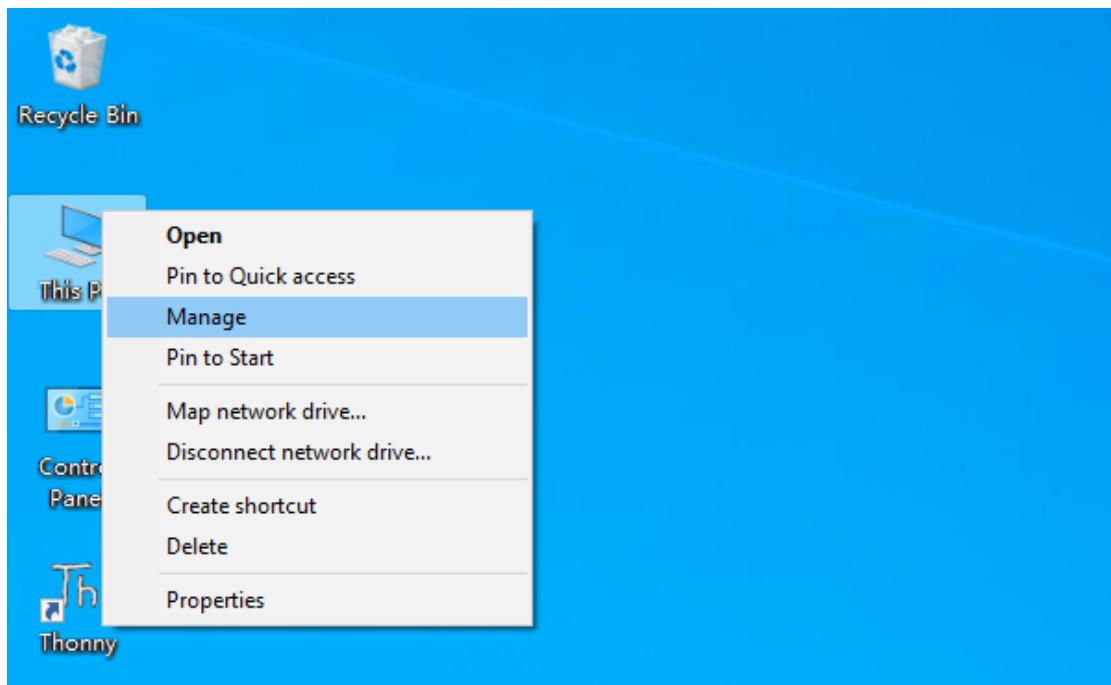
ESP32 uses CH340 to download codes. So before using it, we need to install CH340 driver in our computers.

Check whether CH340 has been installed

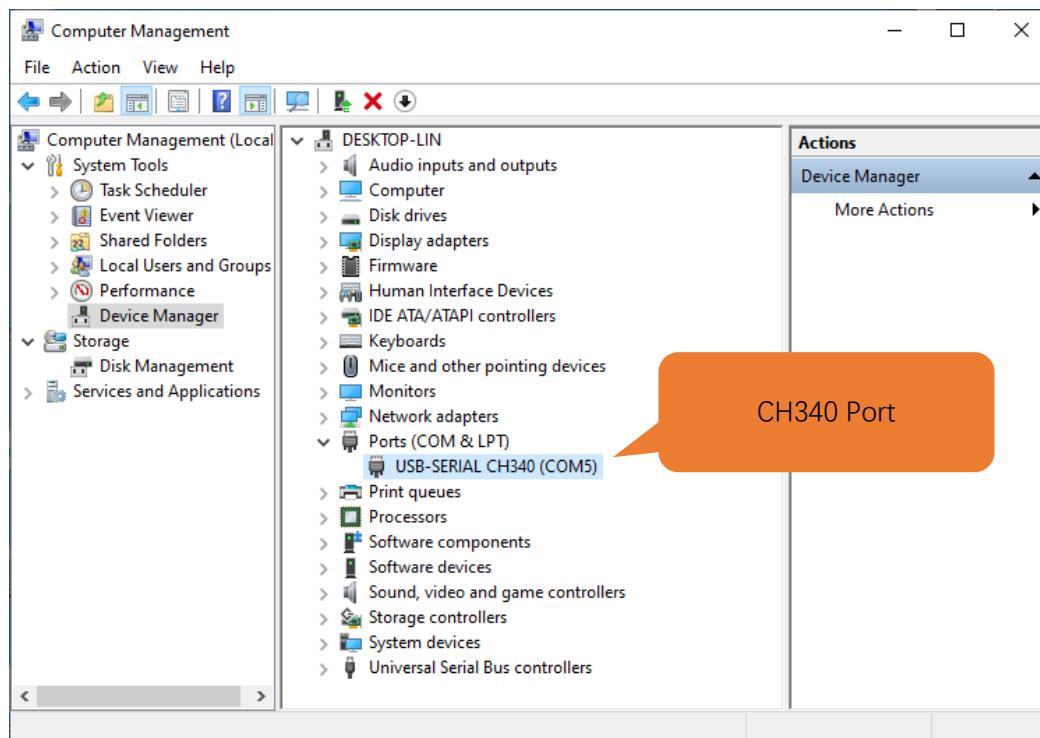
1. Connect your computer and ESP32 with a USB cable.



2. Turn to the main interface of your computer, select "This PC" and right-click to select "Manage".



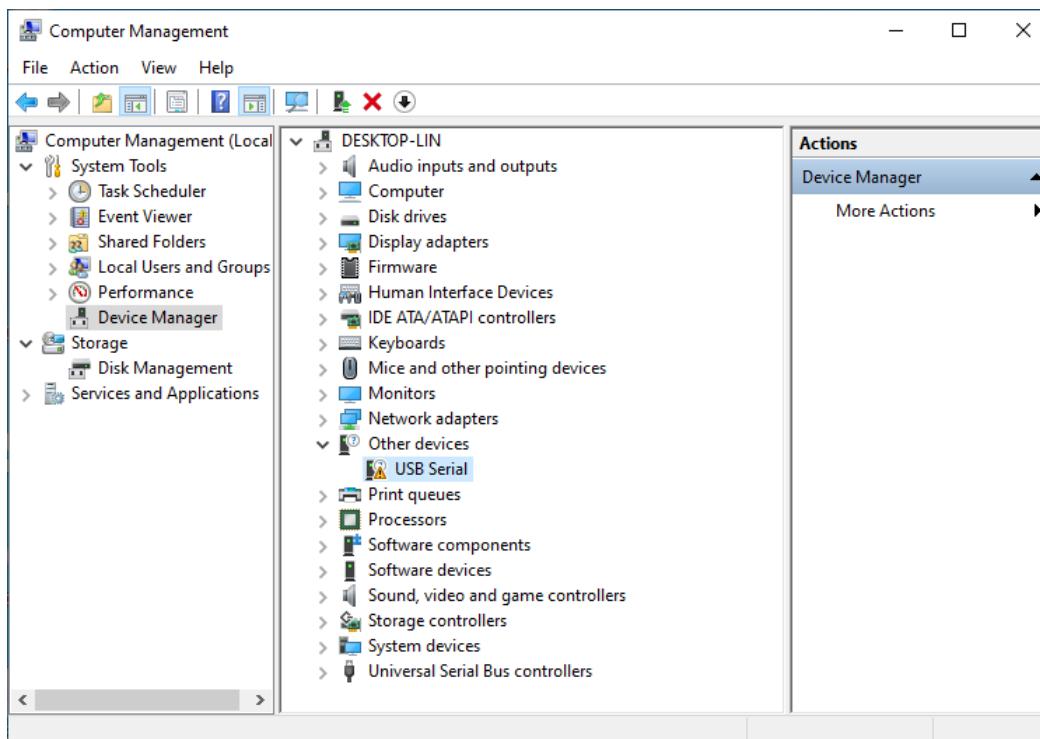
3. Click “Device Manager”. If your computer has installed CH340, you can see “USB-SERIAL CH340 (COMx)”. And you can click [here](#) to move to the next step.



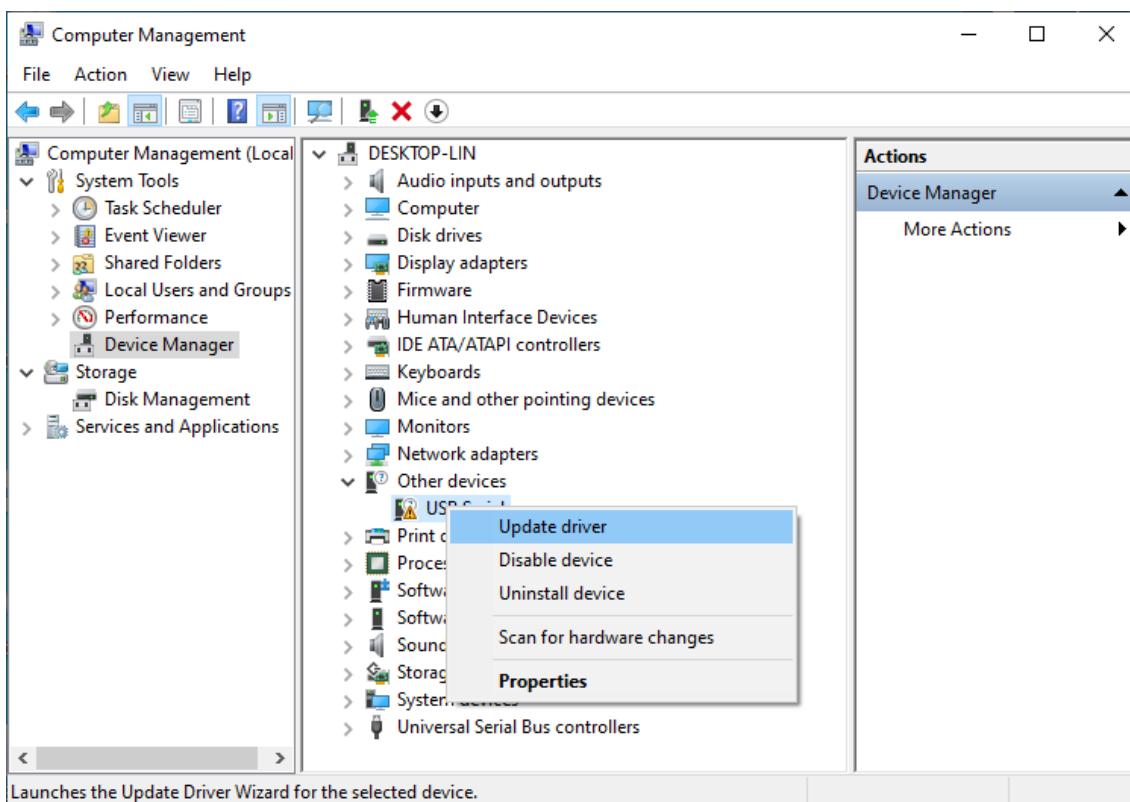
Installing CH340

Method1

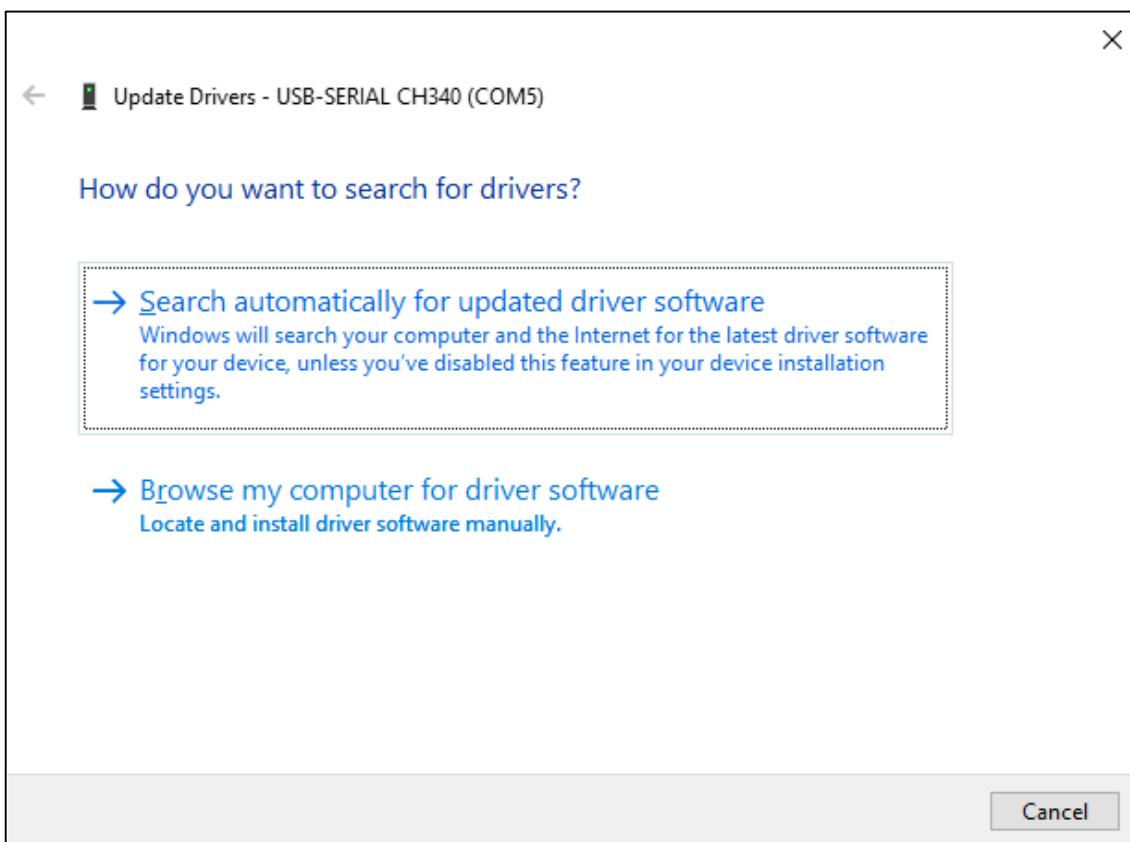
1. If you have not yet installed CH340, you'll see the following interface.



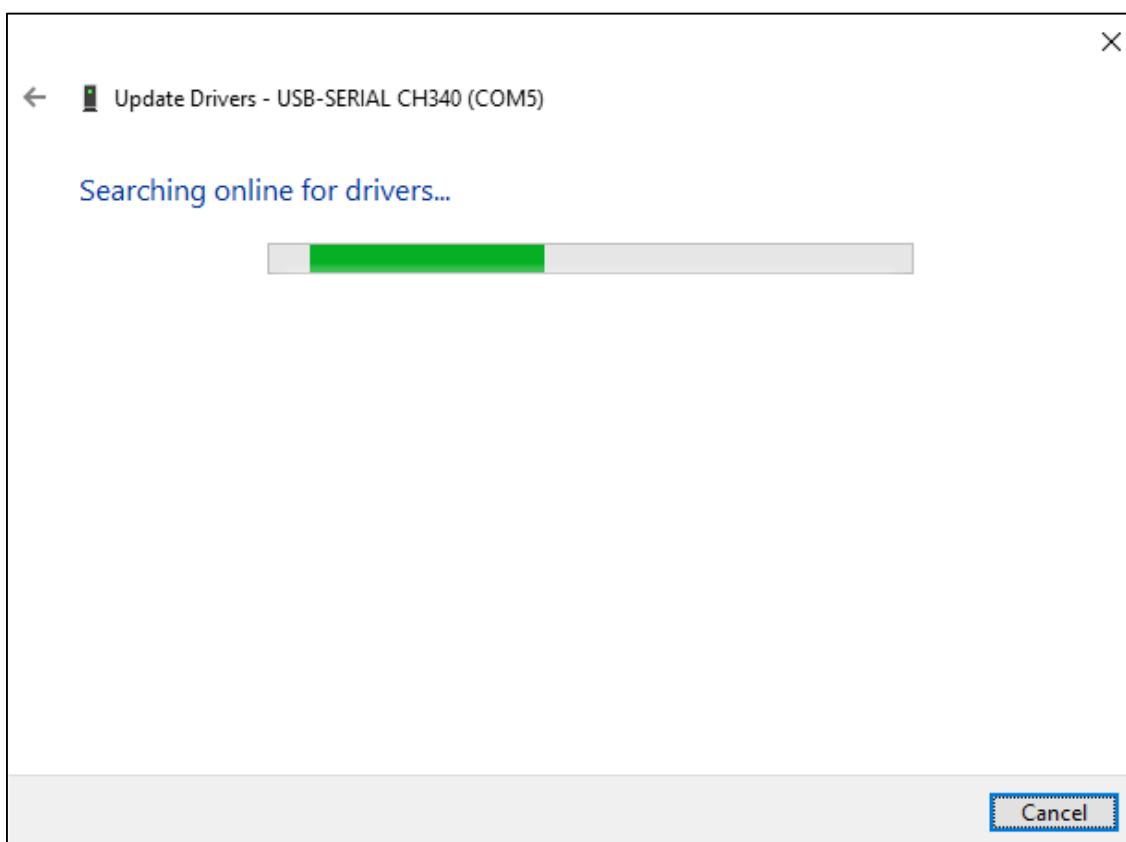
2. Click "USB Serial" and right-click to select "Update driver".



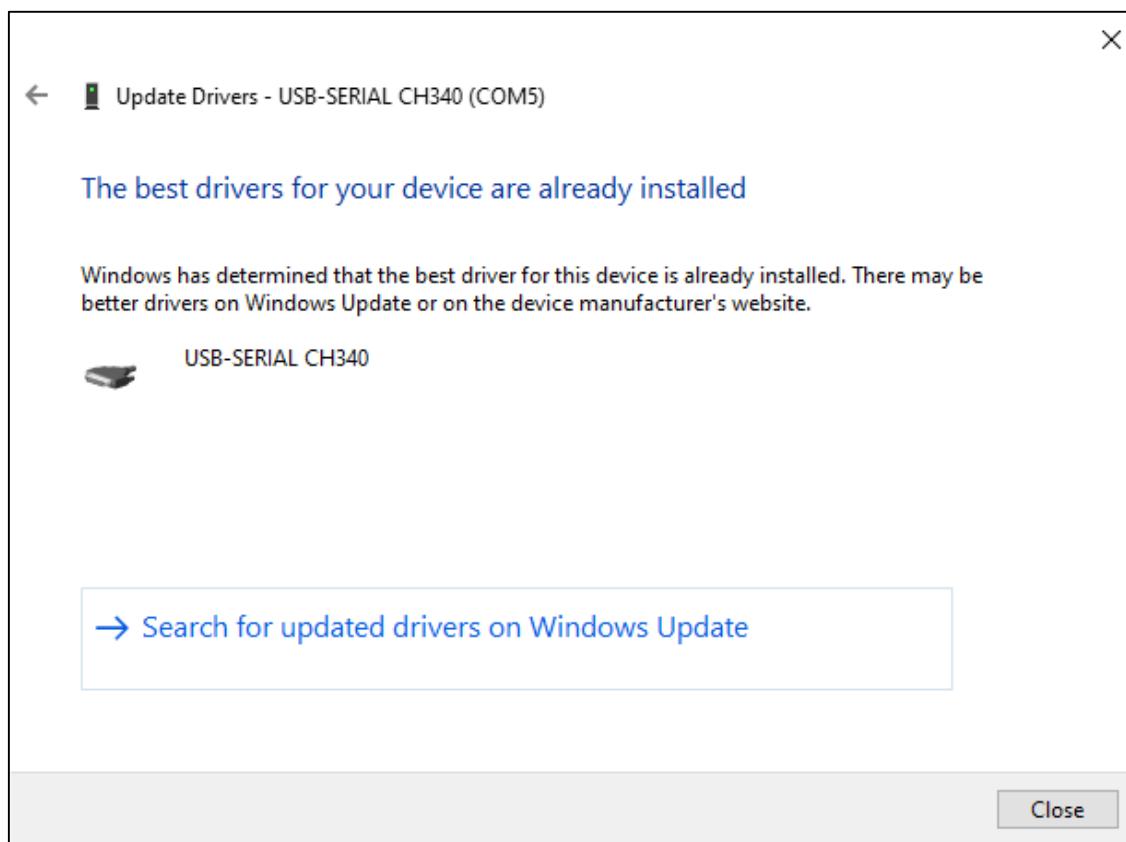
3. Click "Search automatically for updated driver software".



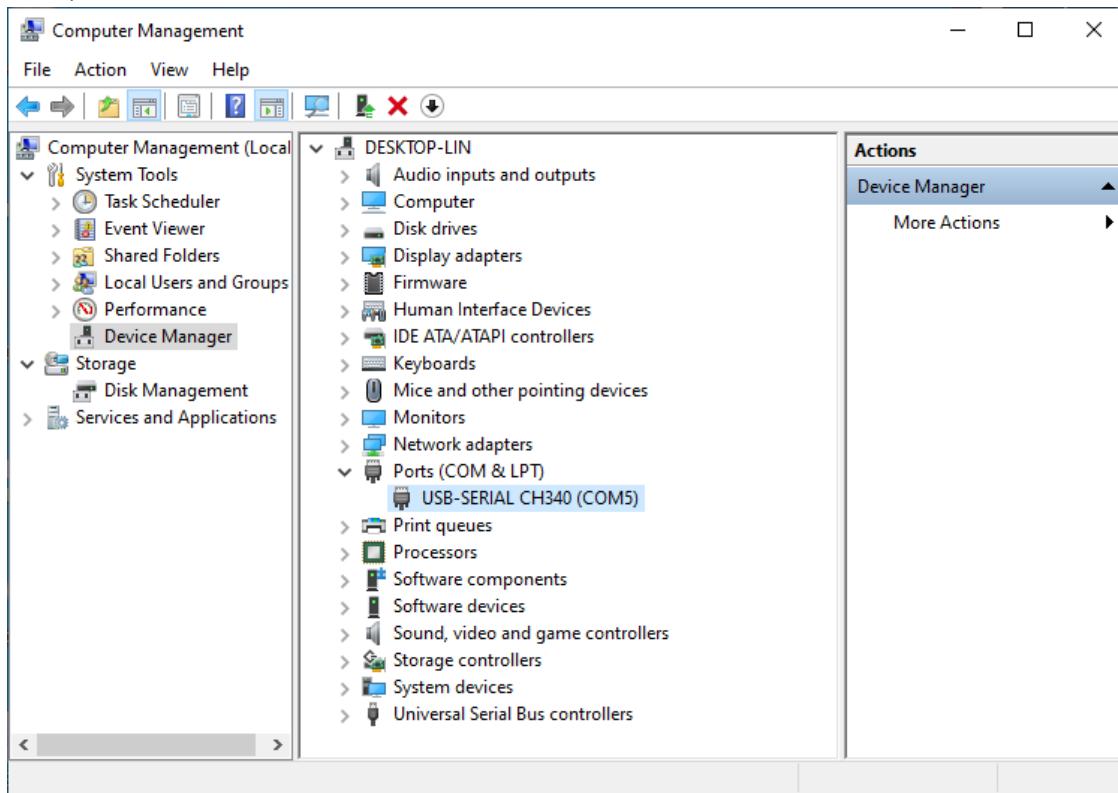
4. Wait for CH340 to finish installation.



5. When you see the following interface, it indicates that CH340 has been installed to your computer. You can close the interface.



6. When ESP32 is connected to computer, you can see the following interface. Click here to move to the next step.



Method2

- First, download CH340 driver, click <http://www.wch-ic.com/search?q=CH340&t=downloads> to download the appropriate one based on your operating system.

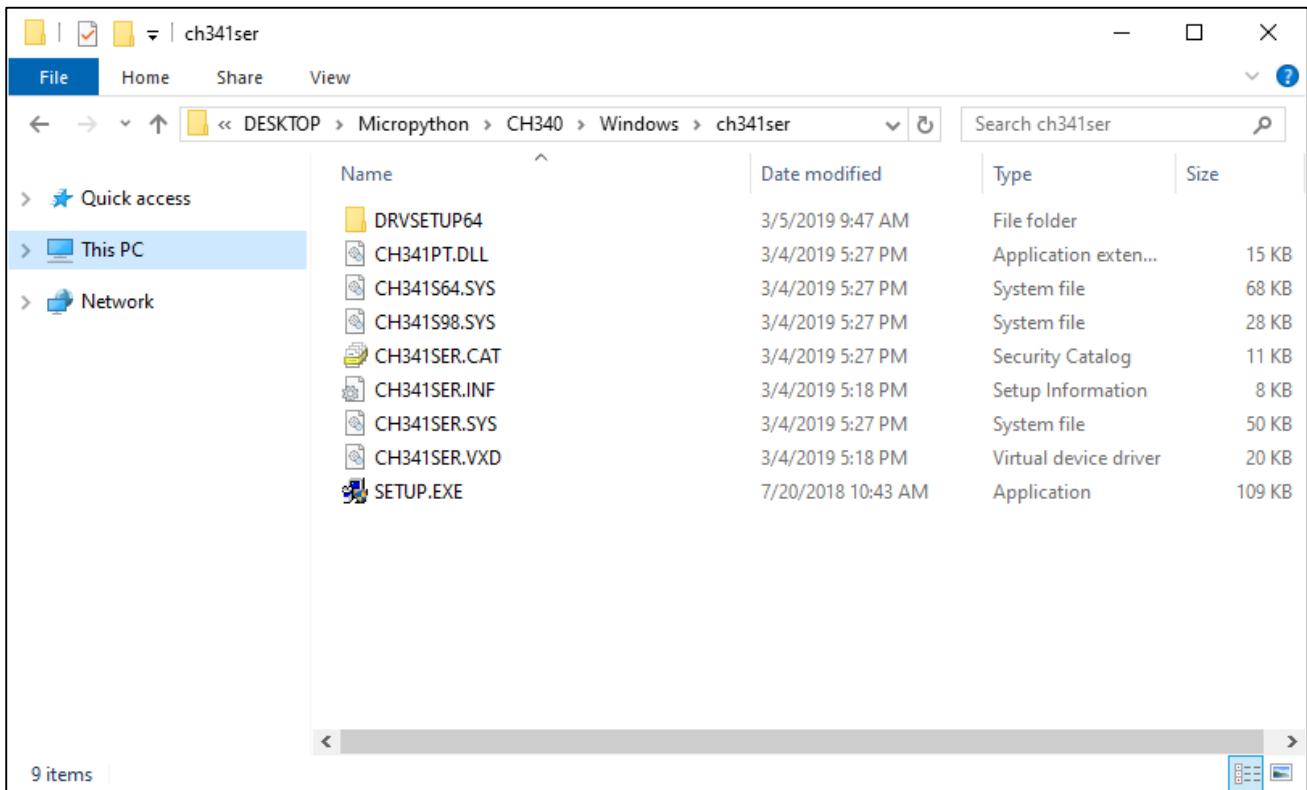
The screenshot shows a search results page for 'CH340' on a website. The left sidebar has categories: All (14), Downloads (7) [highlighted in blue], Products (4), Application (2), Video (1), and News (0). The main area is titled 'keyword CH340' and shows 'Downloads(7)'. A table lists the files:

file category	file content	version	upload time
Driver&Tools	Windows		
CH341SER.EXE	CH340/CH341 USB to serial port Windows driver, supports 32/64-bit Windows 10/8.1/8/7/VISTA/XP, Server 2016/2012/2008/2003, 2000/ME/98	3.5	2019-03-18
CH341SER.ZIP	CH340/CH341 USB to serial port Windows driver, includes DLL dynamic library and non-standard baud rate settings and other instructions. Supports 32/64-bit Windows 10/8.1/8/7/VISTA/XP, Server 2016/2012/2008/2003, 2000/ME/98	3.5	2019-03-05
CH341SER_ANDROID...	CH340/CH341 USB to serial port Android free drive application library, for Android OS 3.1 and above version which supports USB Host mode already, no need to load Android kernel driver, no root privileges. Contains apk, lib library file (linux Driver), App Demo Example (USB to UART Demo)	1.6	2019-04-19
CH341SER_LINUX...	CH340/CH341 USB to serial port LINUX driver	1.5	2018-03-18
CH341SER_MAC.ZI...	CH340/CH341 USB to serial port MAC OS driver	1.5	2018-07-05
Others			
PRODUCT_GUIDE.P...	Electronic selection of product selection manual, please refer to related product technical manual for more technical information.	1.4	2018-12-29
InstallNoteOn64...	Instructions for the driver after 18 years of August cannot be installed under some 64-bit WIN7 (English)	1.0	2019-01-10

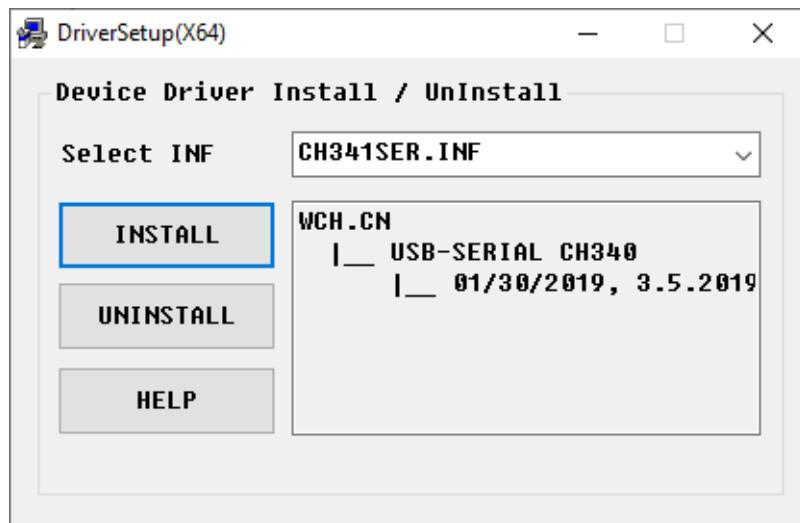
If you would not like to download the installation package, you can open "Freenove_Ultimate_Kit_for_ESP32/CH340", we have prepared the installation package.

Name	Date modified	Type	Size
Linux	8/14/2020 5:24 PM	File folder	
MAC	8/14/2020 5:23 PM	File folder	
Windows	8/14/2020 5:23 PM	File folder	

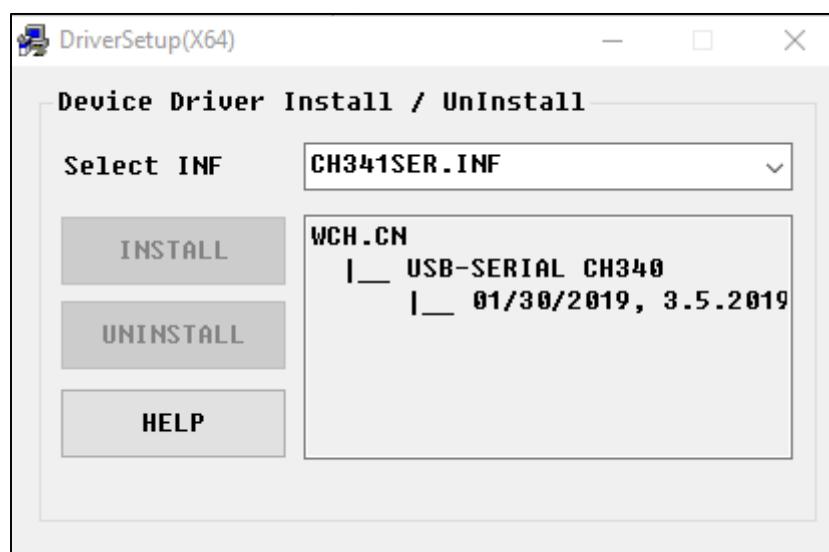
2. Open the folder "Freenove_Ultimate_Kit_for_ESP32/CH340/Windows/ch341ser"



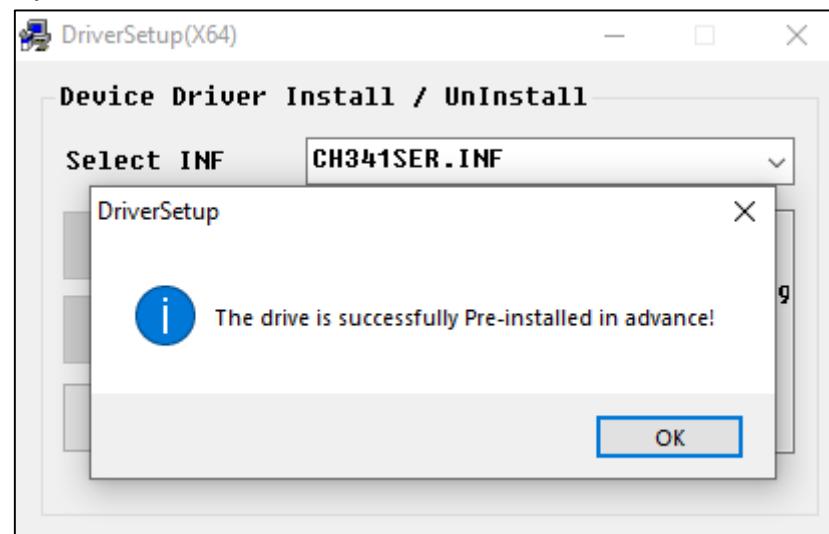
3. Double click "SETUP.EXE".



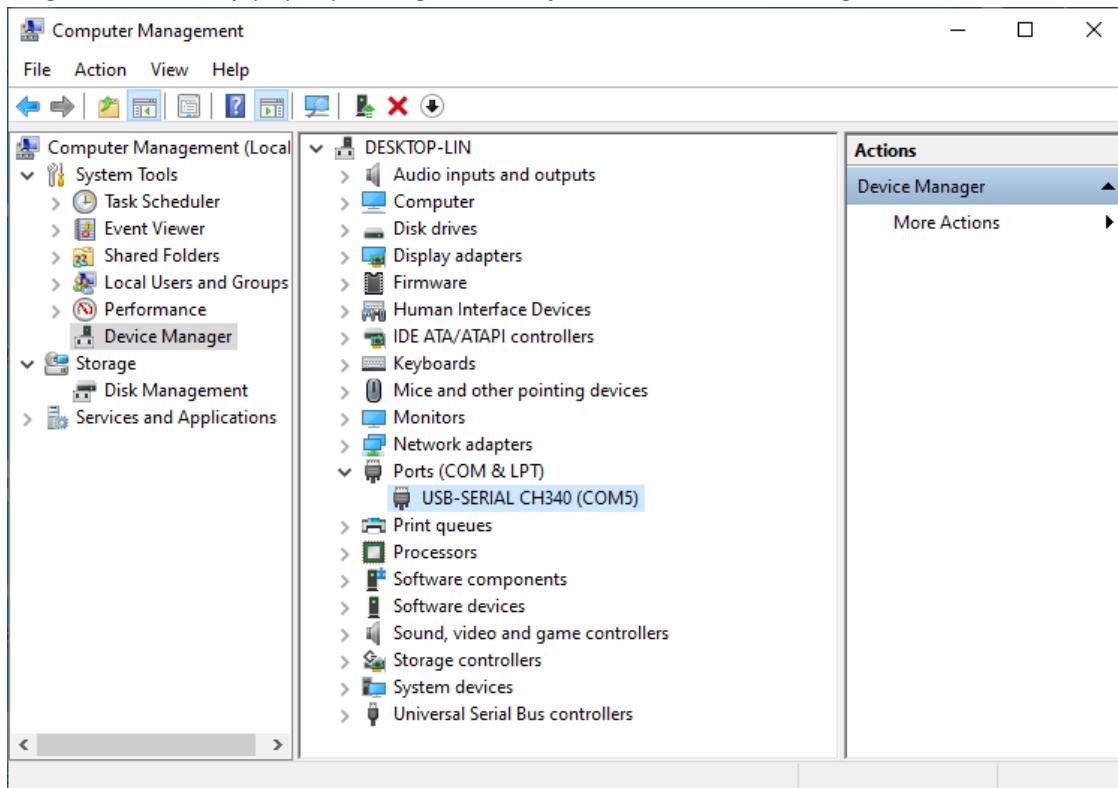
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



6. When ESP32 is connected to computer, select "This PC", right-click to select "Manage" and click "Device Manager" in the newly pop-up dialog box, and you can see the following interface.



7. So far, CH340 has been installed successfully. Close all dialog boxes.

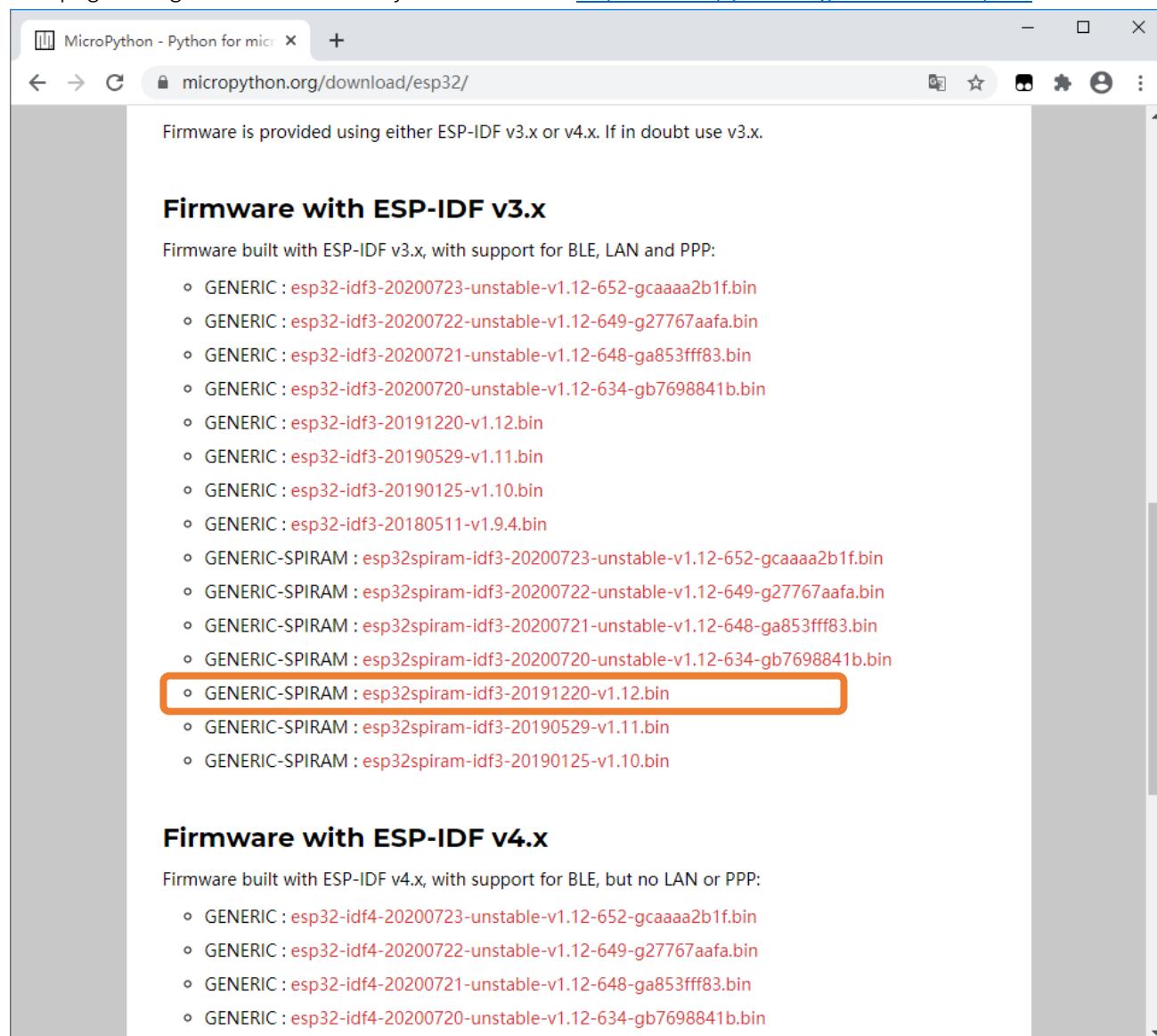
0.4 Burning Micropython Firmware

To run Python programs on ESP32, we need to burn a firmware to ESP32 first.

Downloading Micropython Firmware

Official website of microPython: <http://micropython.org/>

Webpage listing firmware of microPython for ESP32: <https://micropython.org/download/esp32/>



Firmware is provided using either ESP-IDF v3.x or v4.x. If in doubt use v3.x.

Firmware with ESP-IDF v3.x

Firmware built with ESP-IDF v3.x, with support for BLE, LAN and PPP:

- GENERIC : [esp32-idf3-20200723-unstable-v1.12-652-gcaaaa2b1f.bin](#)
- GENERIC : [esp32-idf3-20200722-unstable-v1.12-649-g27767aafa.bin](#)
- GENERIC : [esp32-idf3-20200721-unstable-v1.12-648-ga853fff83.bin](#)
- GENERIC : [esp32-idf3-20200720-unstable-v1.12-634-gb7698841b.bin](#)
- GENERIC : [esp32-idf3-20191220-v1.12.bin](#)
- GENERIC : [esp32-idf3-20190529-v1.11.bin](#)
- GENERIC : [esp32-idf3-20190125-v1.10.bin](#)
- GENERIC : [esp32-idf3-20180511-v1.9.4.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20200723-unstable-v1.12-652-gcaaaa2b1f.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20200722-unstable-v1.12-649-g27767aafa.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20200721-unstable-v1.12-648-ga853fff83.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20200720-unstable-v1.12-634-gb7698841b.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20191220-v1.12.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20190529-v1.11.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20190125-v1.10.bin](#)

Firmware with ESP-IDF v4.x

Firmware built with ESP-IDF v4.x, with support for BLE, but no LAN or PPP:

- GENERIC : [esp32-idf4-20200723-unstable-v1.12-652-gcaaaa2b1f.bin](#)
- GENERIC : [esp32-idf4-20200722-unstable-v1.12-649-g27767aafa.bin](#)
- GENERIC : [esp32-idf4-20200721-unstable-v1.12-648-ga853fff83.bin](#)
- GENERIC : [esp32-idf4-20200720-unstable-v1.12-634-gb7698841b.bin](#)

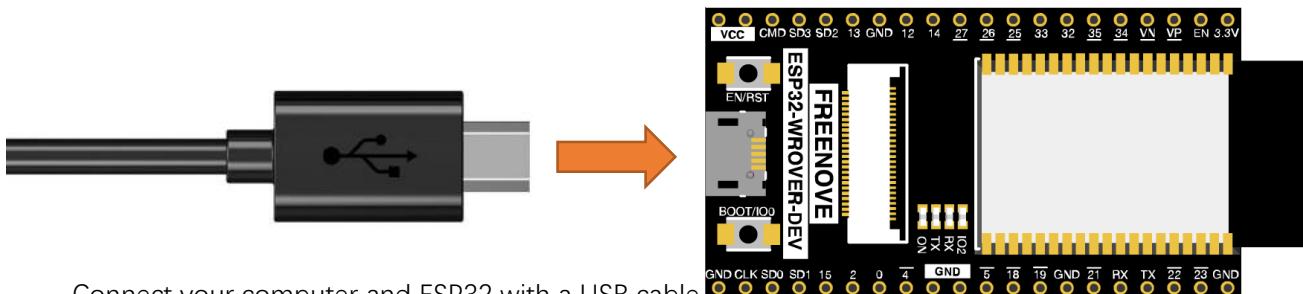
Firmware used in this tutorial is **esp32spiram-idf3-20191220-v1.12.bin**

Click the following link to download directly: <https://micropython.org/resources/firmware/esp32spiram-idf3-20191220-v1.12.bin>

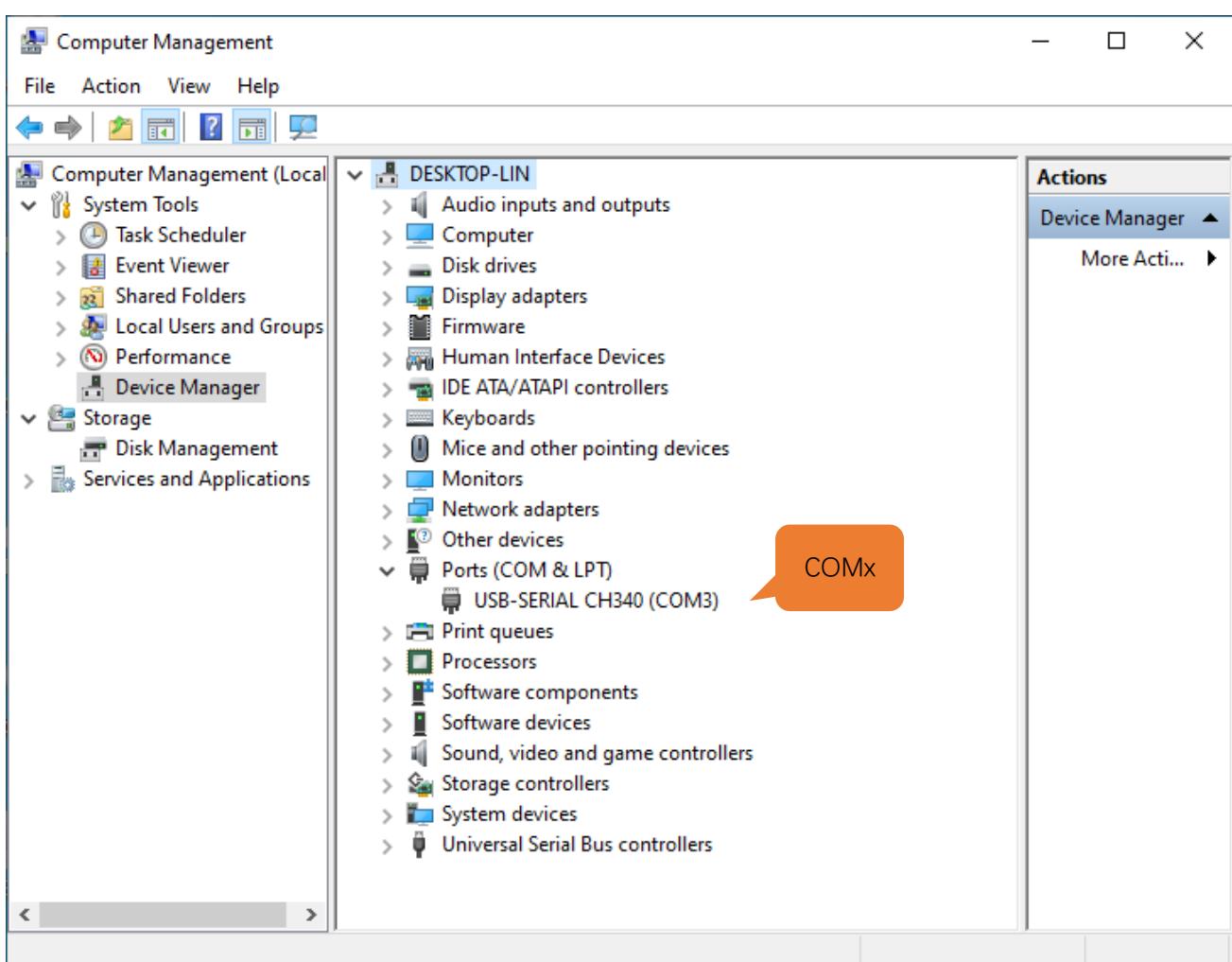
This file is also provided in our data folder

"Freenove_Ultimate_Kit_for_ESP32/MicroPython_Firmware" ..

Burning a Micropython Firmware

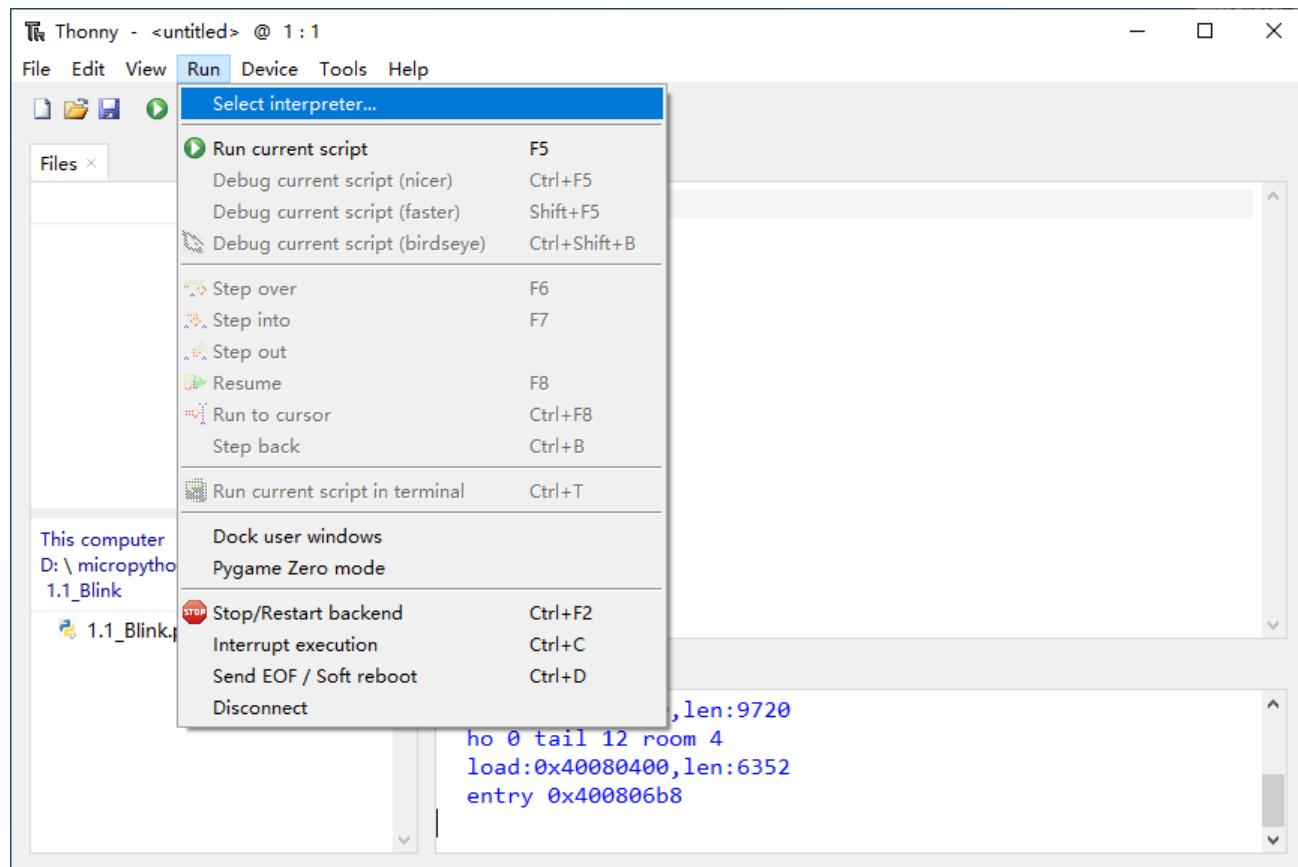


Make sure that the driver has been installed successfully and that it can recognize COM port correctly. Open device manager and expand "Ports".

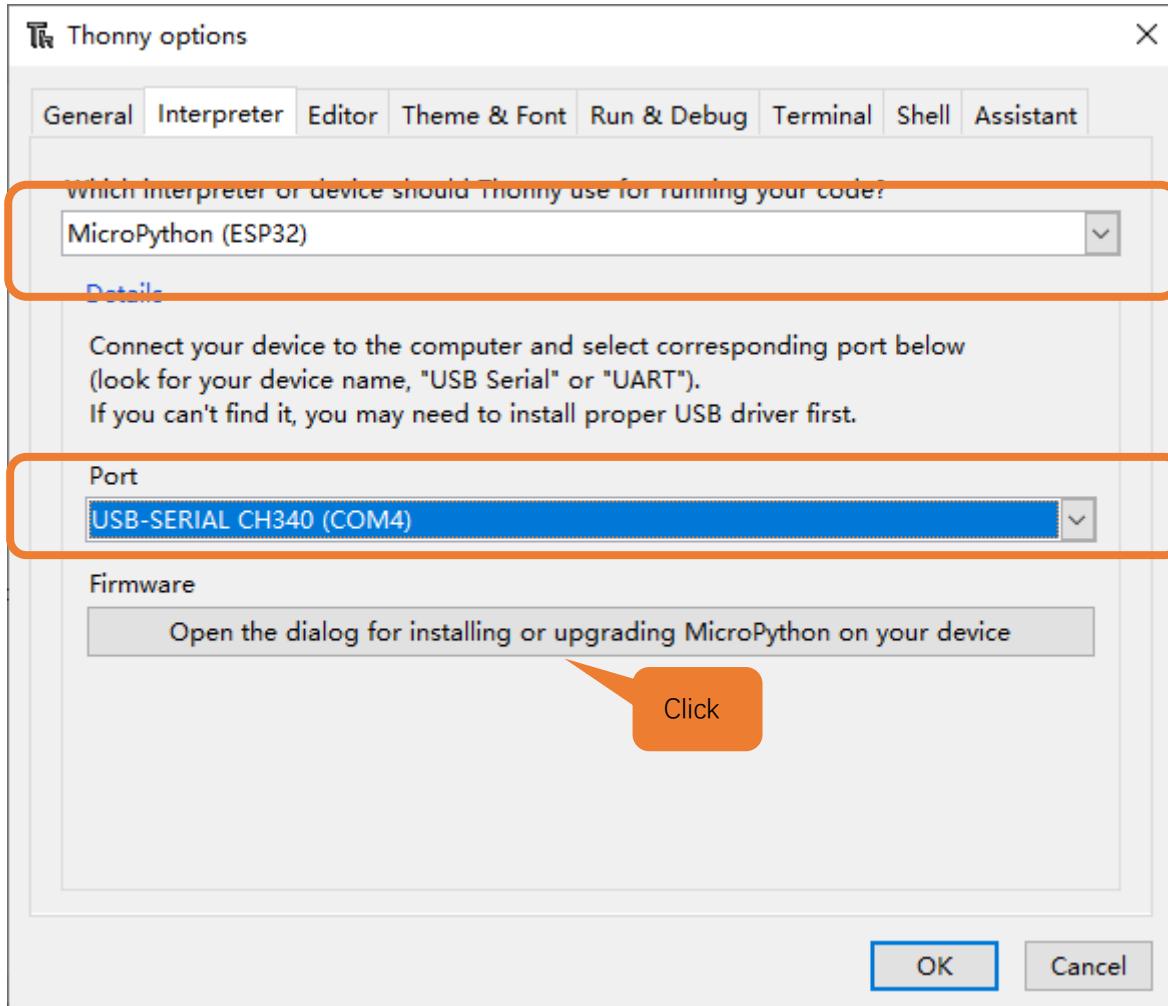


Note: the port of different people may be different, which is a normal situation.

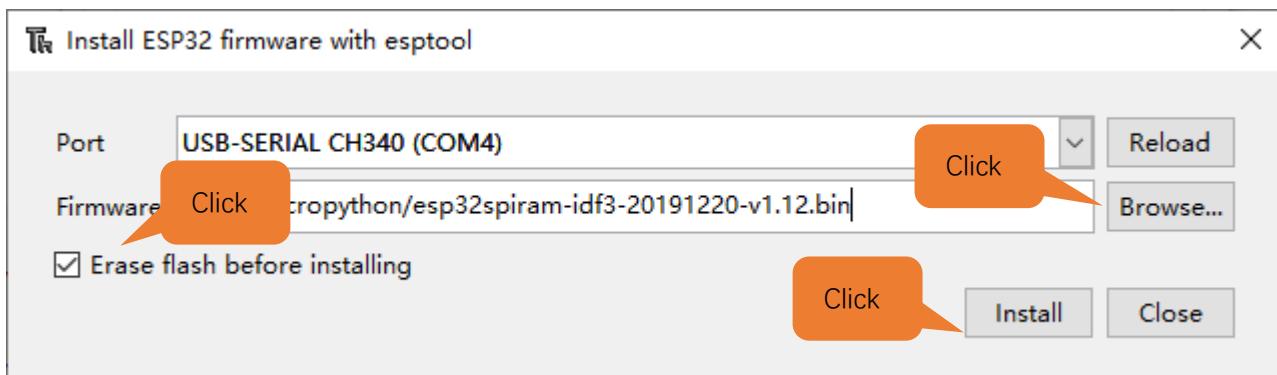
1. Open Thonny, click "run" and select "Select interpreter..."



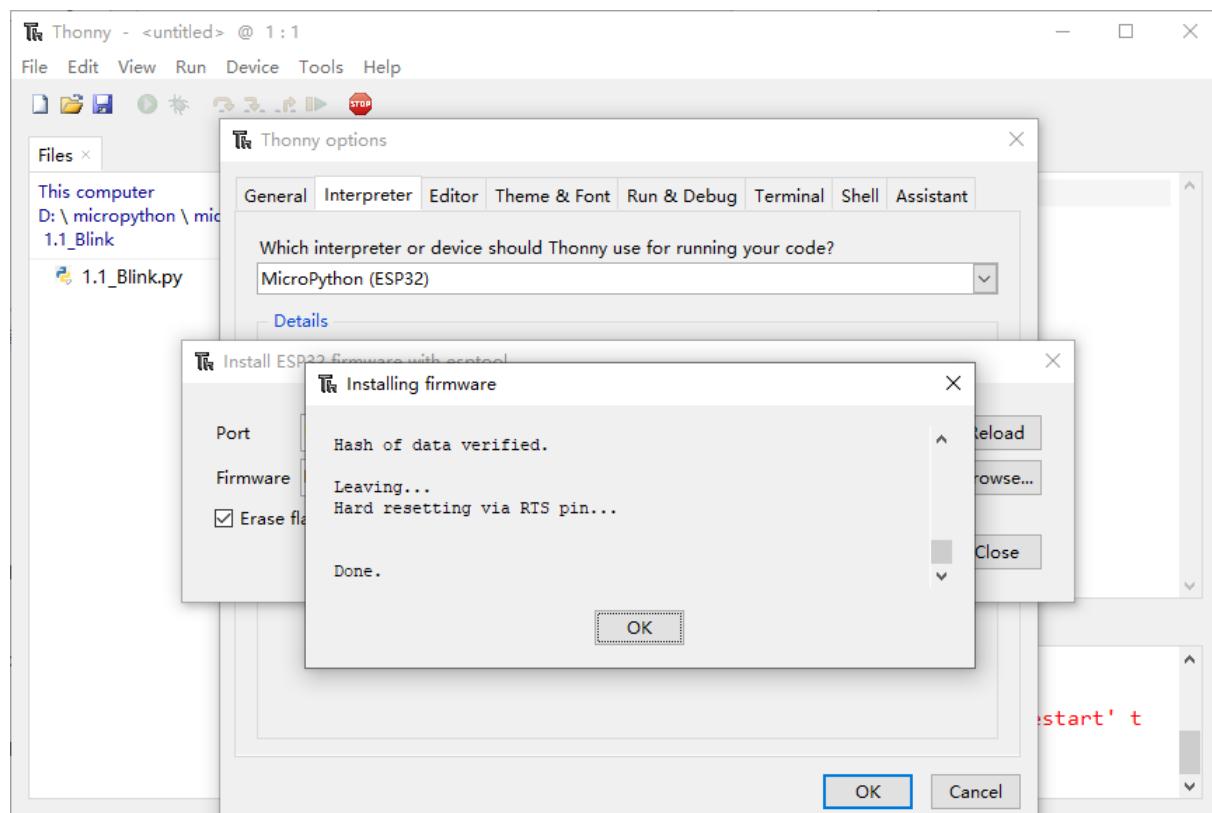
2. Select “Micropython (ESP32)”, select “USB-SERIAL CH340 (COM4)”, and then click the long button under “Firmware”.



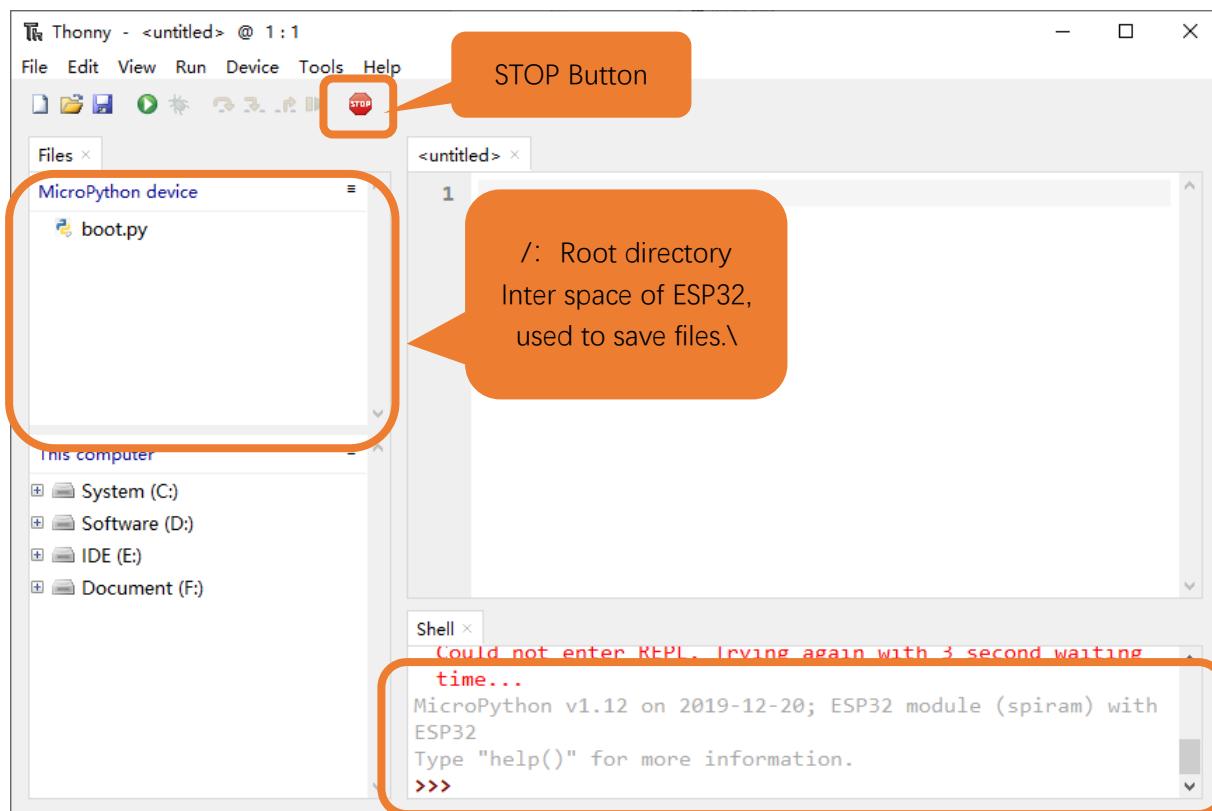
3. The following dialog box pops up. Select “USB-SERIAL CH340 (COM4)” for “Port” and then click “Browse...”. Select the previous prepared microPython firmware “**esp32spiram-idf3-20191220-v1.12.bin**”. Check “Erase flash before installing” and click “install” to wait for the prompt of finishing installation



4. Wait for the installation to be done.



5. Close all dialog boxes, turn to main interface and click "STOP". As shown in the illustration below

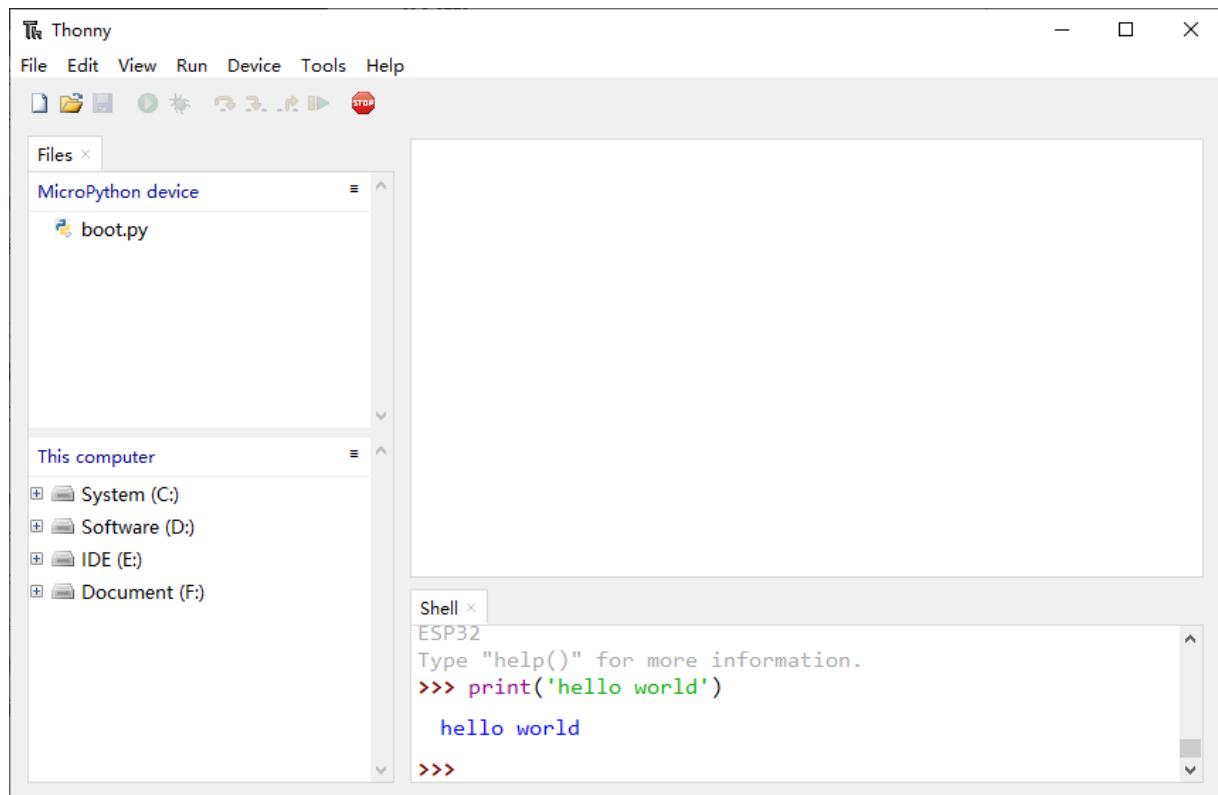


6. So far, all the preparations have been made.

0.5 Testing codes

Testing Shell Command

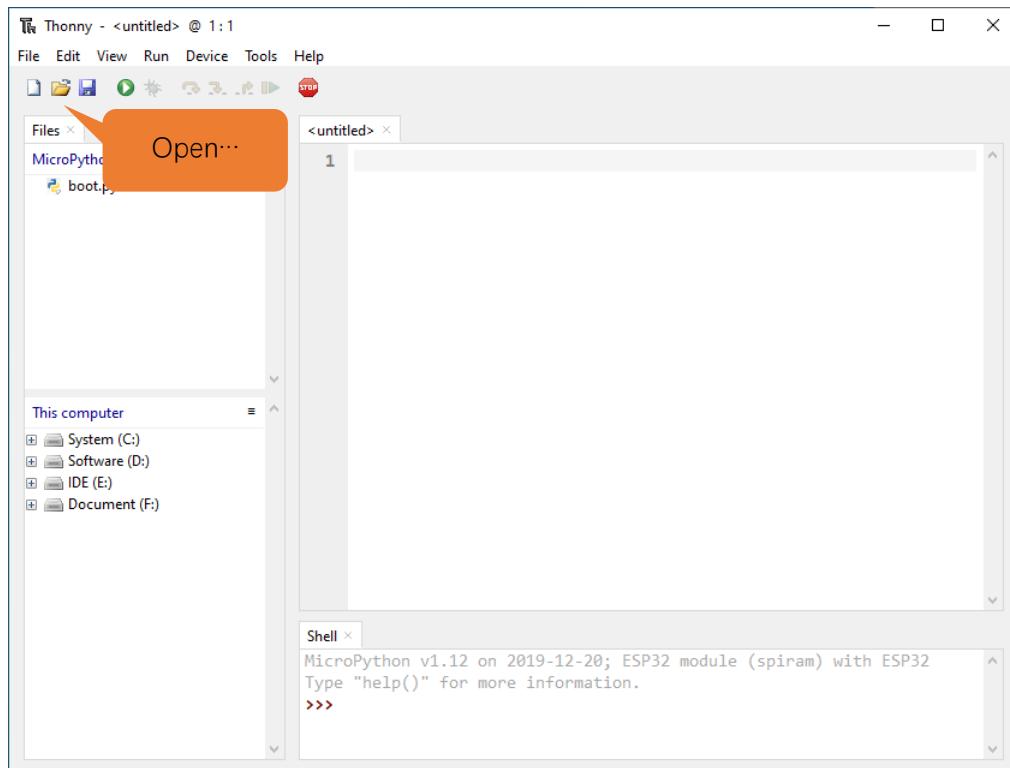
Enter “print('hello world')” in “Shell” and press Enter.



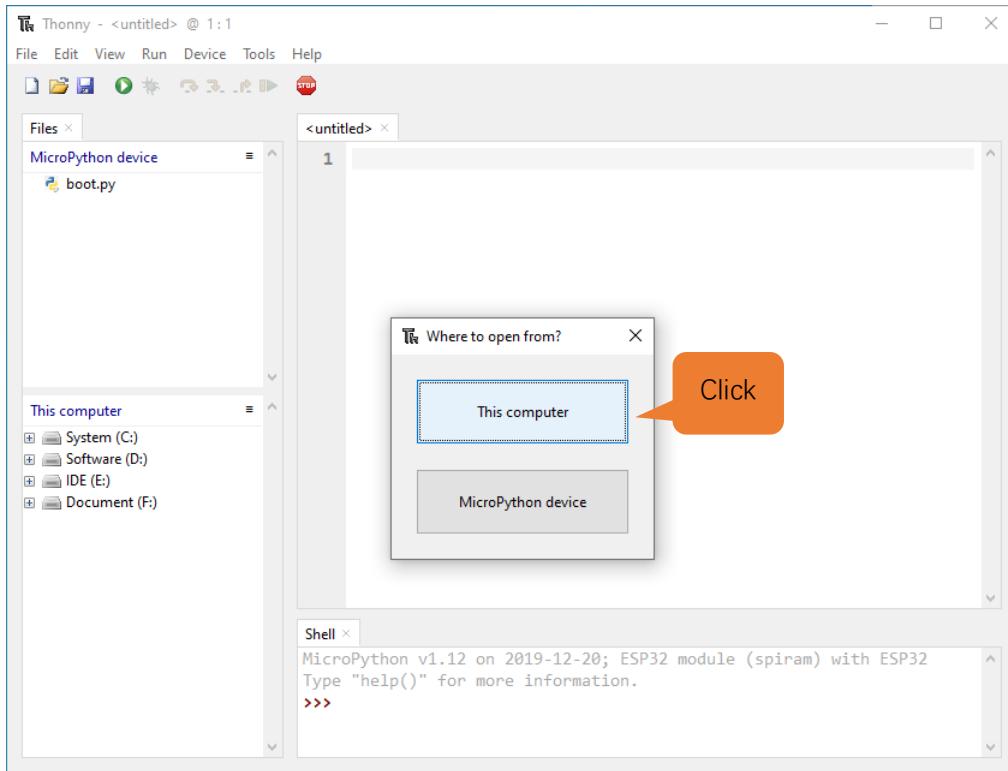
Running Online

ESP32 needs to be connected to a computer when it is run online. Users can use Thonny to writer and debug programs.

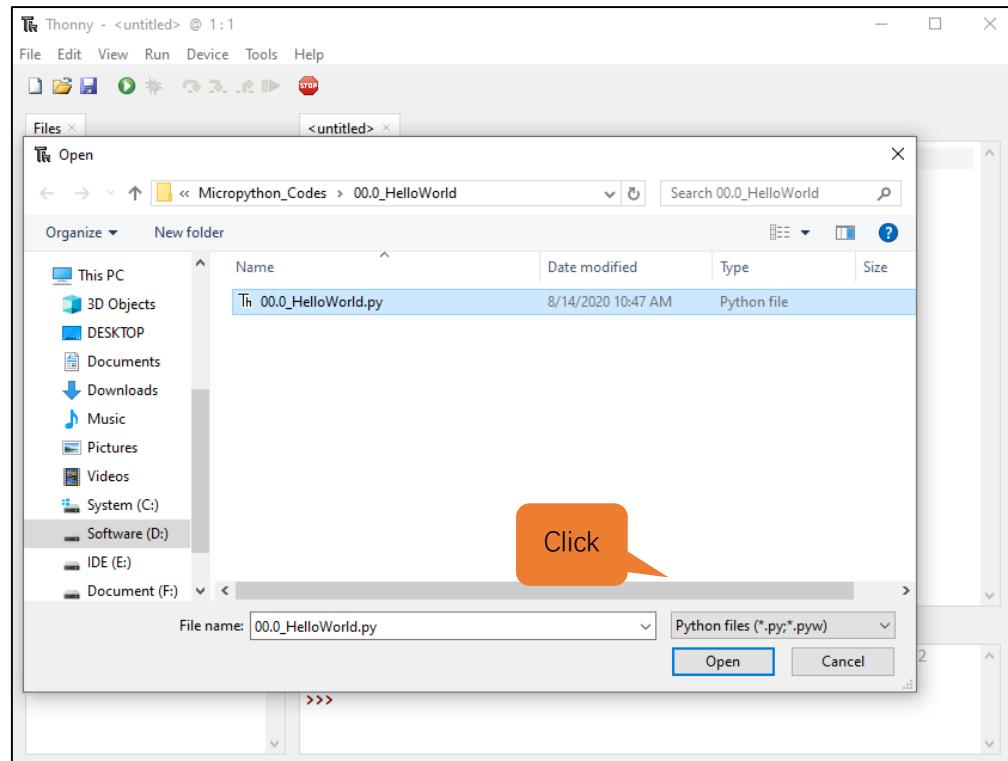
1. Open Thonny and click "Open...".



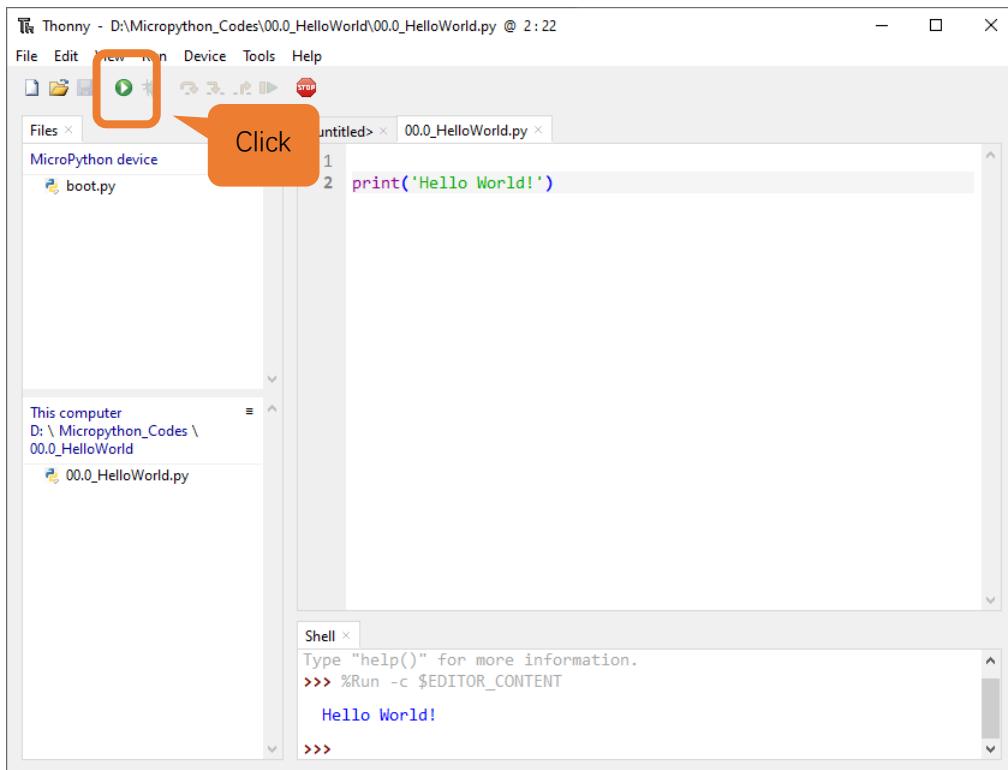
2. On the newly pop-up window, click "This computer".



In the new dialog box, select “00.0_HelloWorld.py” in
“Freenove_Ultimate_Kit_for_ESP32/Micropython_Codes/00.0_HelloWorld” folder.



Click “Run current script” to execute the program and “Hello World” will be printed in “Shell”.

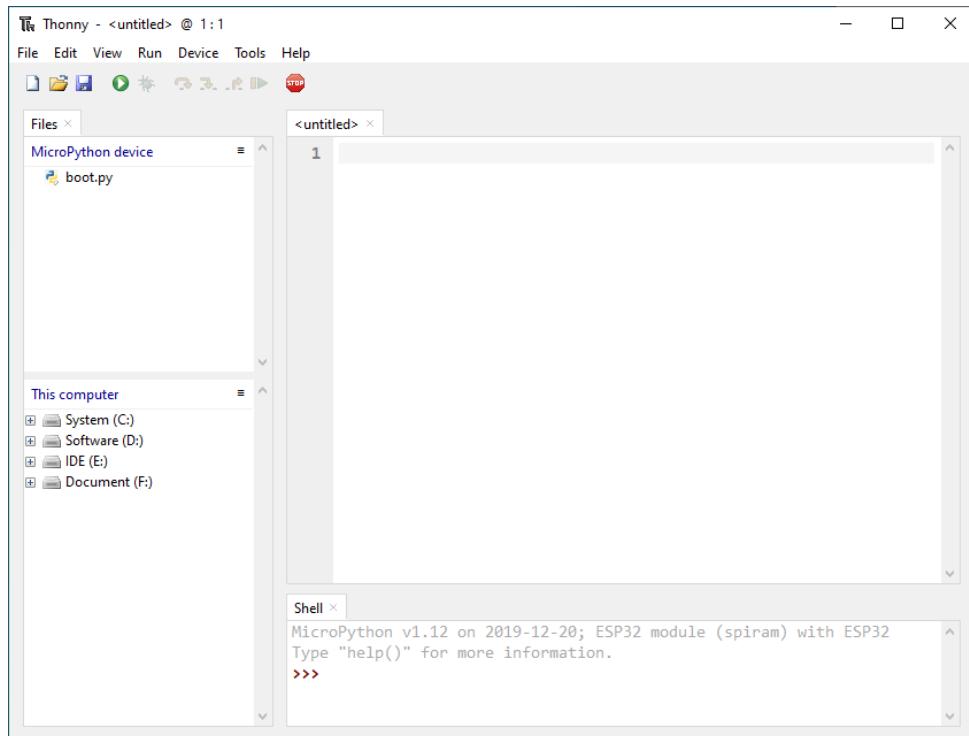


Note: When running online, if you press the reset key of ESP32, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following [Running Offline](#).

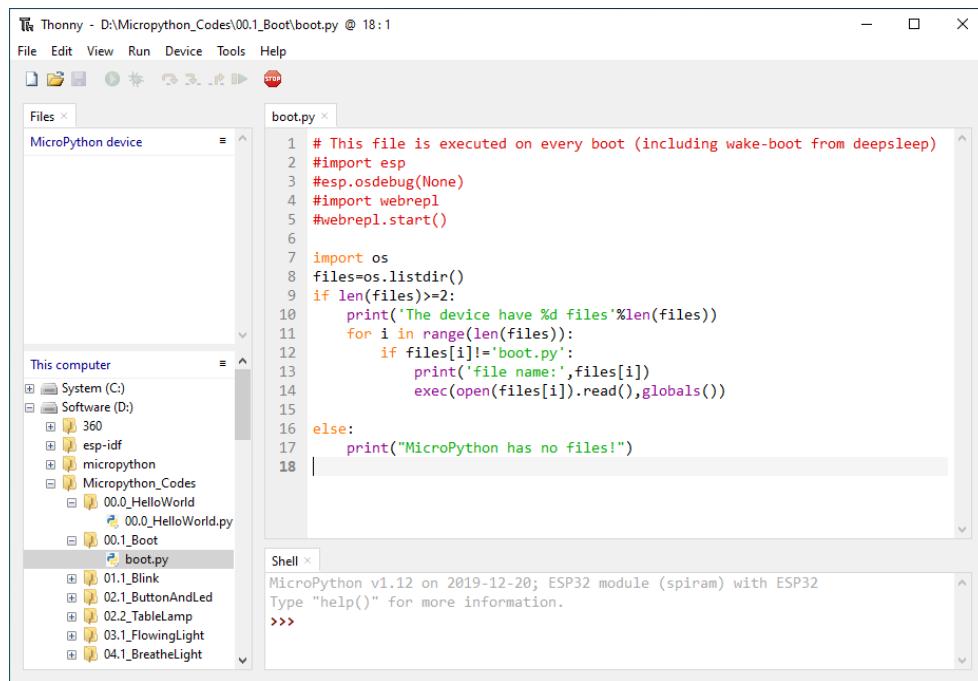
Running Offline

After ESP32 is reset, it runs the file boot.py in root directory first and then runs file main.py, and finally, it enters "Shell". Therefore, to make ESP32 execute user's programs after resetting, we need to add a guiding program in boot.py to execute user's code.

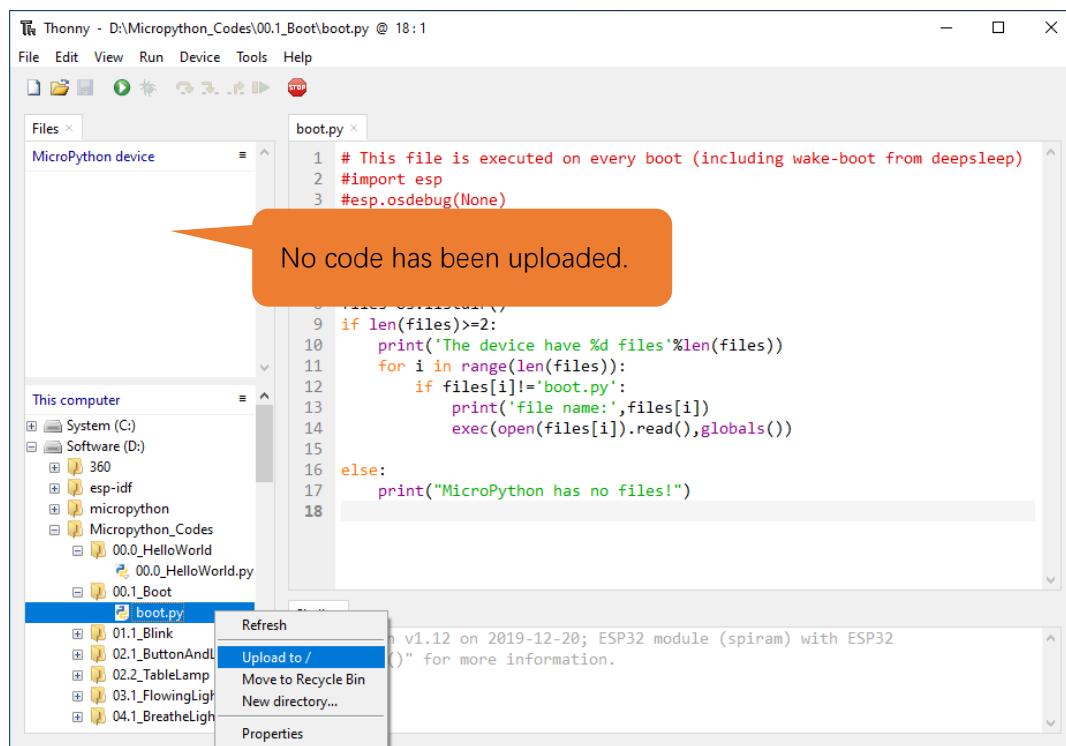
1. Move the program folder "**Freenove_Ultimate_Kit_for_ESP32/Micropython_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**". Open "Thonny".



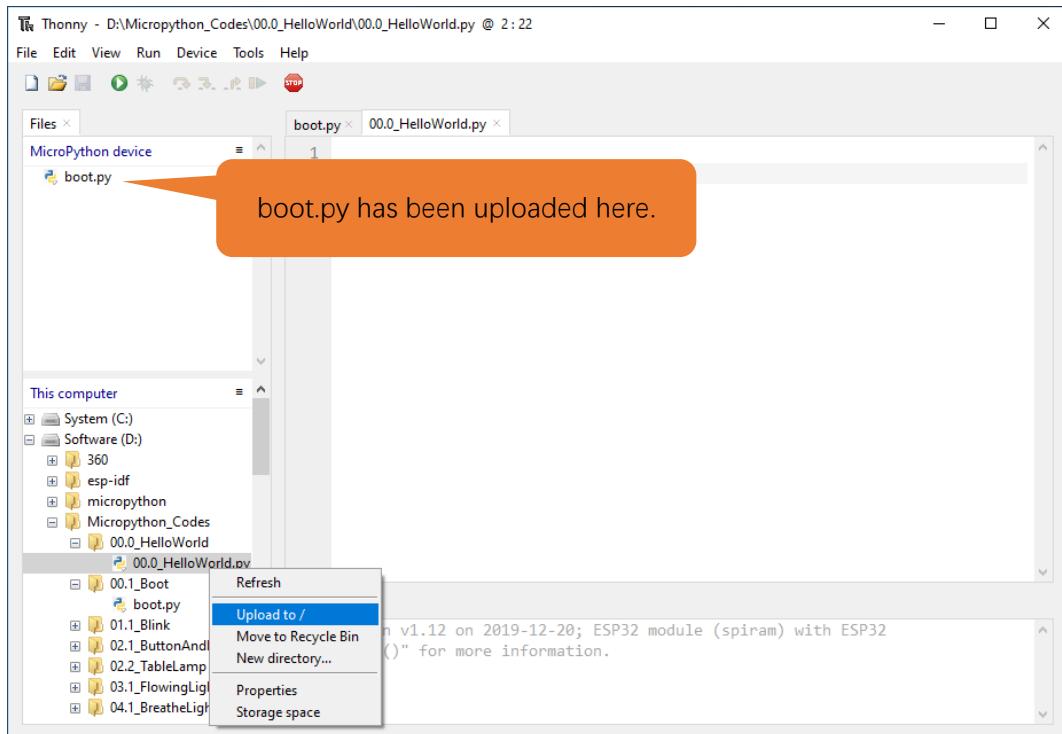
2. Expand "00.1_Boot" in the "Micropython_Codes" in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in "MicroPython device" to run offline.



If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to “MicroPython device” and press ESP32’s reset key. Here we use programs 00.0 and 00.1 as examples. Select “boot.py”, right-click to select “Upload to /”.



Similarly, upload “00.0_HelloWorld.py” to “MicroPython device”.



3. Press the reset key and in the box of the illustration below, you can see the code is executed.

The screenshot shows the Thonny IDE interface. On the left, the file tree displays a project structure under 'D:\Micropython_Codes\00.0_HelloWorld'. The 'boot.py' and '00.0_HelloWorld.py' files are listed. The '00.0_HelloWorld.py' file contains the code:

```
1 print('Hello World!\n')
```

In the bottom right pane, the 'Shell' window shows the output of the code execution:

```
I (150) cpu_start: Starting scheduler on PRO CPU.  
I (150) cpu_start: Starting scheduler on APP CPU.  
The device have 2 files  
file name: 00.0_HelloWorld.py  
Hello World!
```

The last two lines, 'Hello World!', are highlighted with a red box.

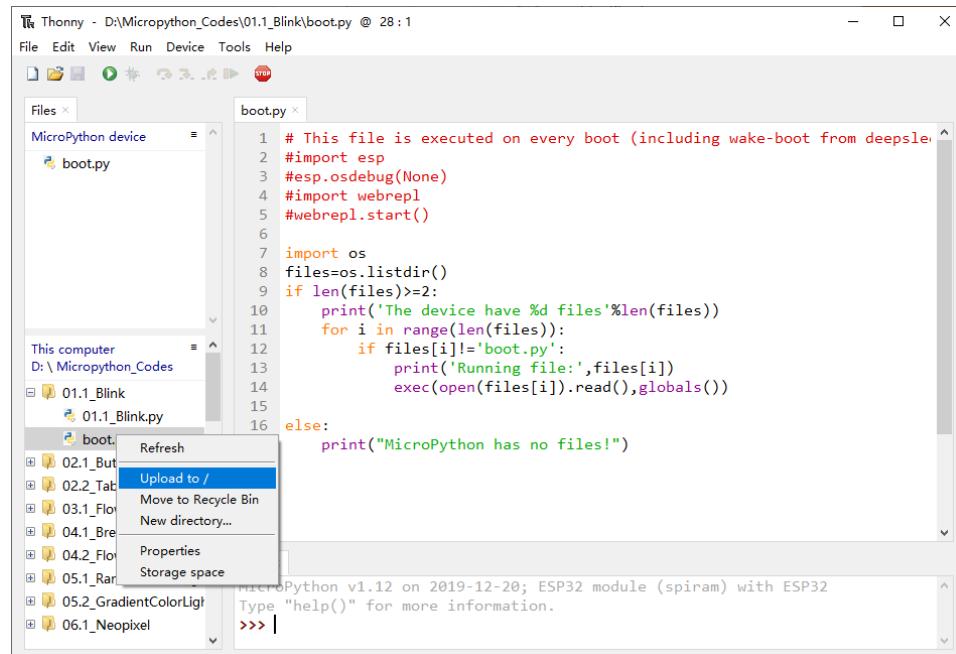
0.6 Thonny Common Operation

Uploading Code to ESP32

For convenience, we take the operation on “boot.py” as an example here. We have added “boot.py” to every code directory. Each time when ESP32 restarts, if there is a “boot.py” in the root directory, it will execute this code first.

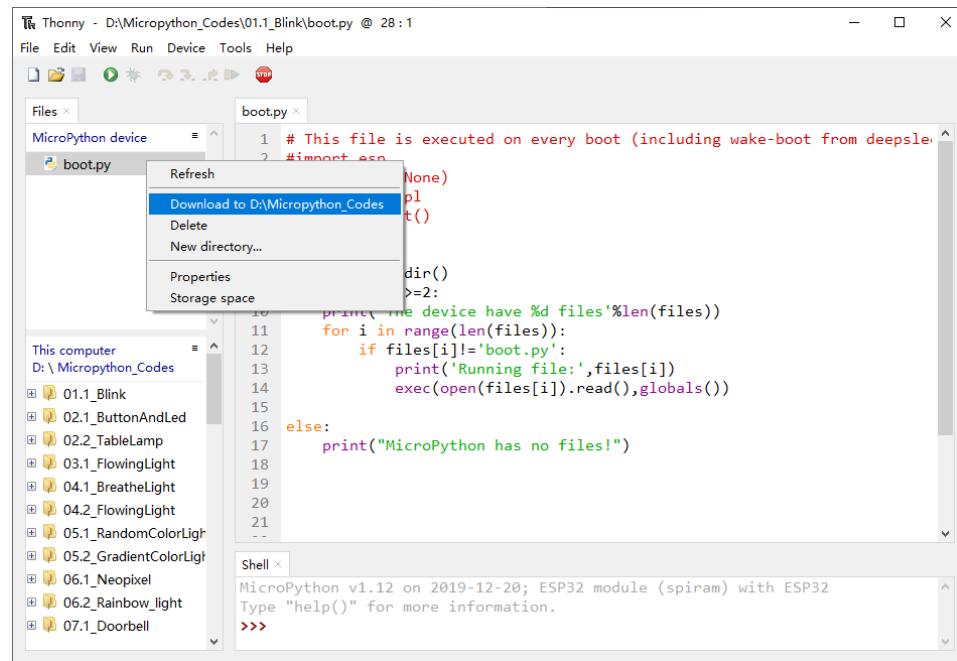


Select “boot.py” in “01.1_Blink”, right-click your mouse and select “Upload to /” to upload code to ESP32’s root directory.



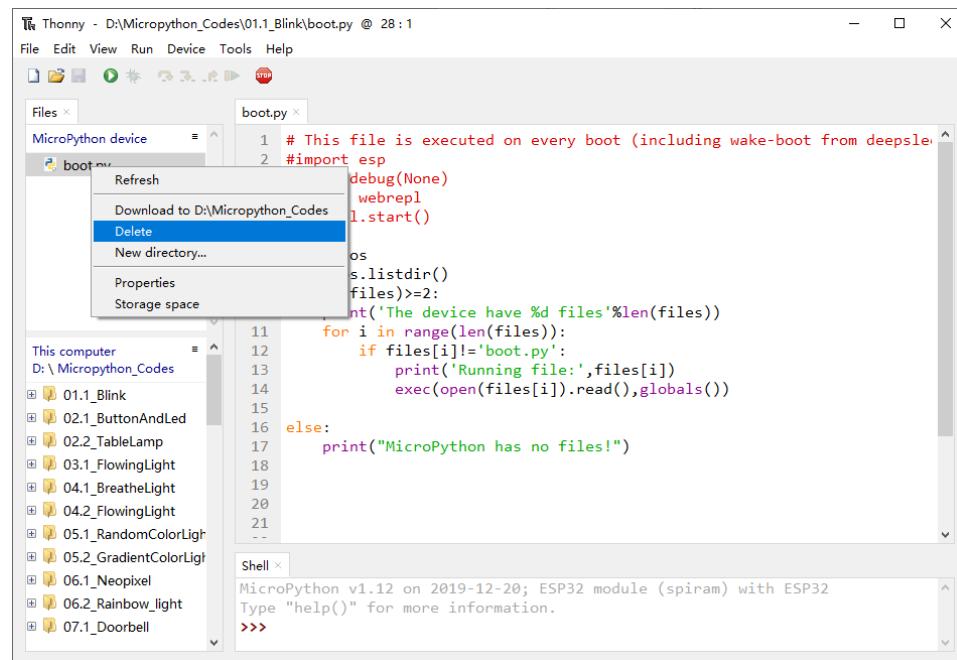
Downloading Code to Computer

Select “boot.py” in “MicroPython device”, right-click to select “Download to ...” to download the code to your computer.



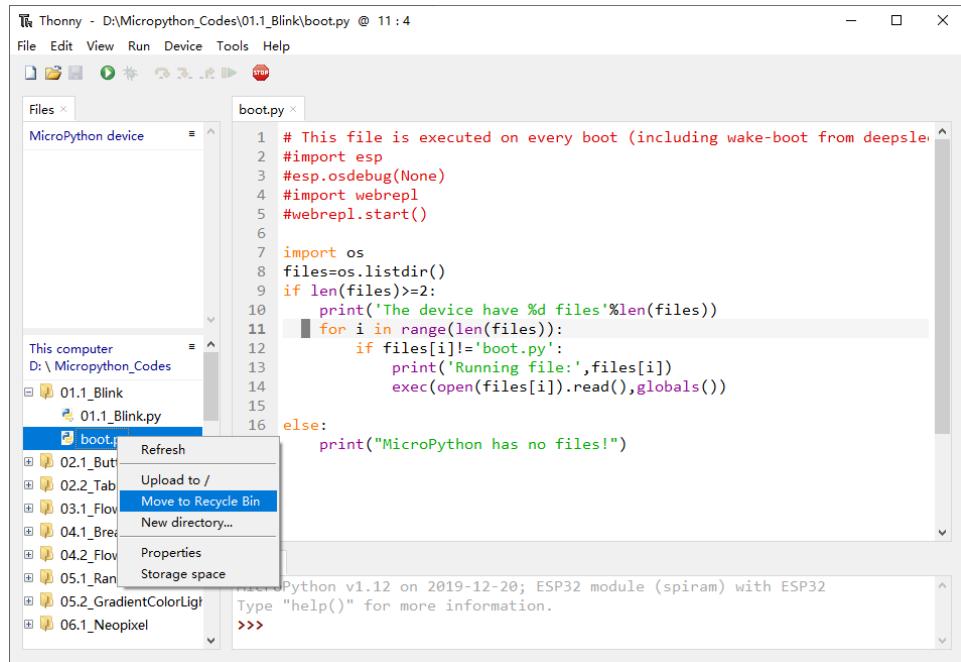
Deleting Files from ESP32's Root Directory

Select “boot.py” in “MicroPython device”, right-click it and select “Delete” to delete “boot.py” from ESP32's root directory.



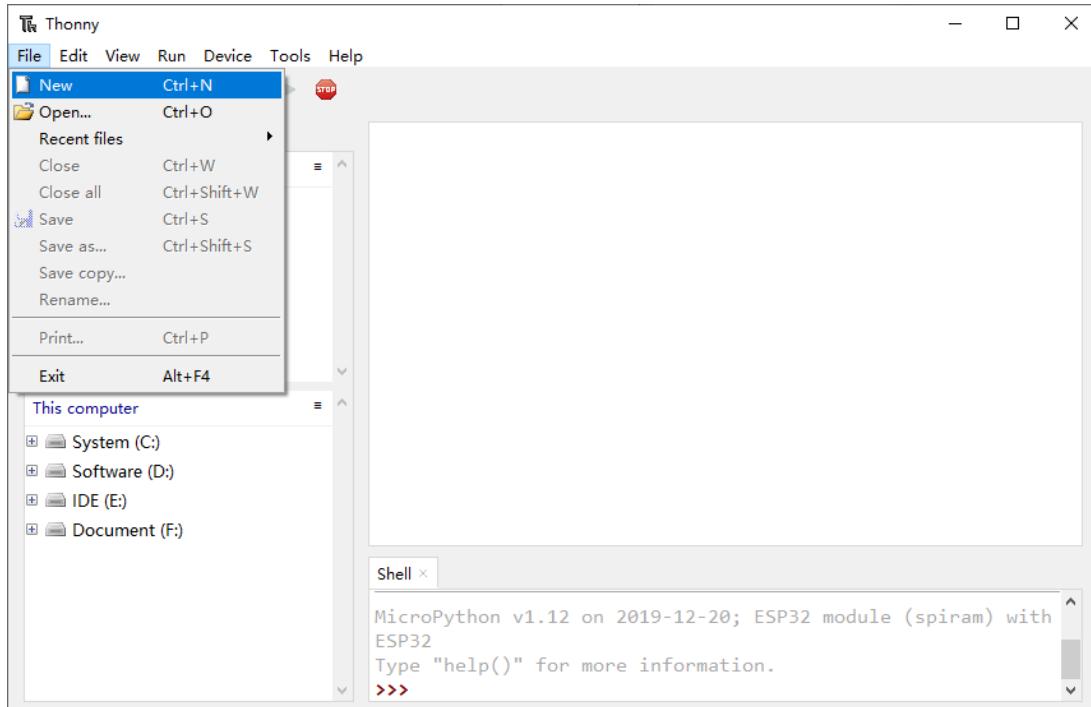
Deleting Files from your Computer Directory

Select “boot.py” in “01.1_Blink”, right-click it and select “Move to Recycle Bin” to delete it from “01.1_Blink”.



Creating and Saving the code

Click “File”→“New” to create and write codes.

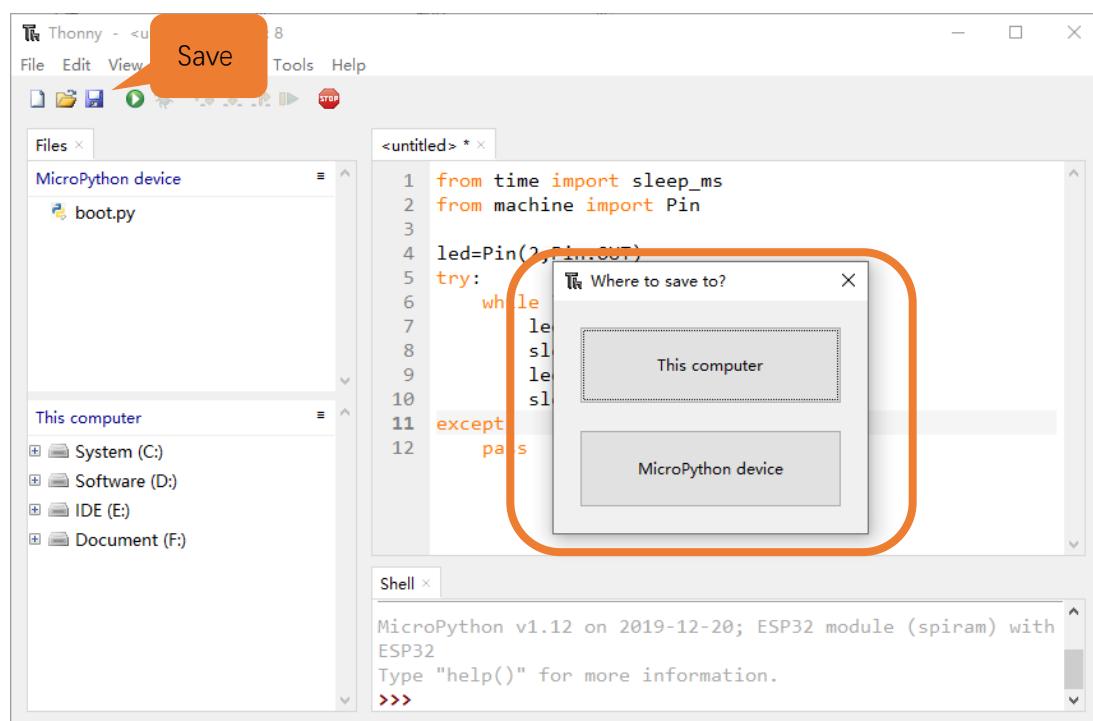


Enter codes in the newly opened file. Here we use codes of “1.1_Blink.py” as an example.

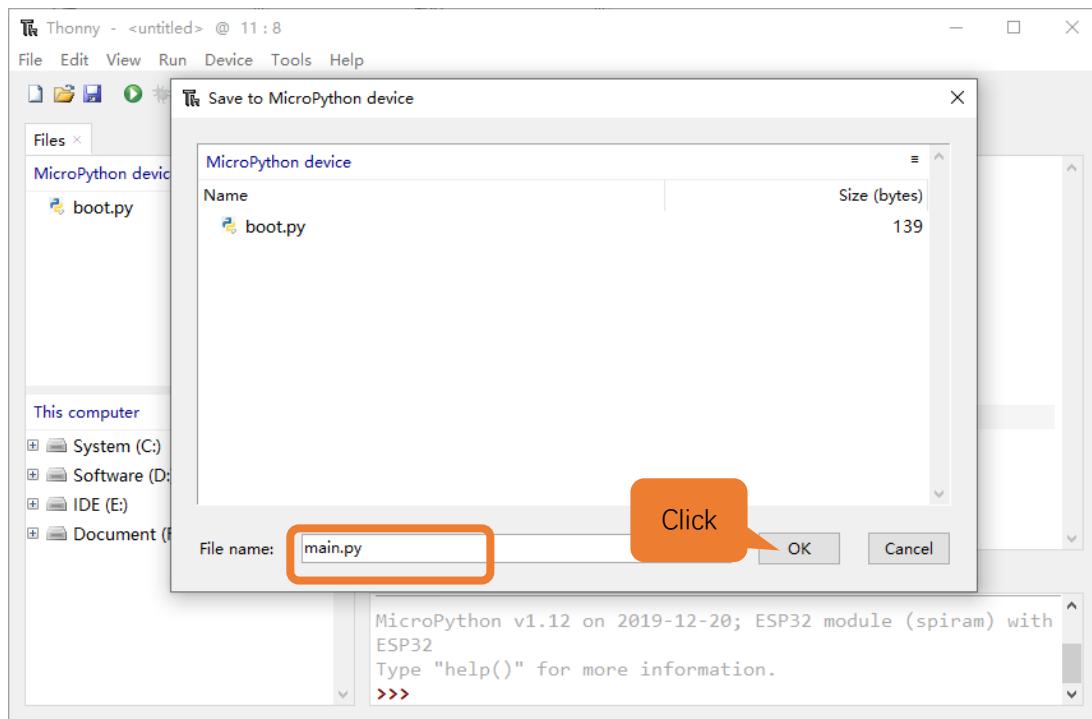
The screenshot shows the Thonny IDE interface. The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The left sidebar shows a 'Files' tree with 'MicroPython device' expanded, showing a file named 'boot.py'. Below it is 'This computer' with options like System (C:), Software (D:), IDE (E:), and Document (F:). The main code editor window contains the following Python code:

```
1 from time import sleep_ms
2 from machine import Pin
3
4 led=Pin(2,Pin.OUT)
5 try:
6     while True:
7         led.value(1)
8         sleep_ms(1000)
9         led.value(0)
10        sleep_ms(1000)
11 except:
12     pass
```

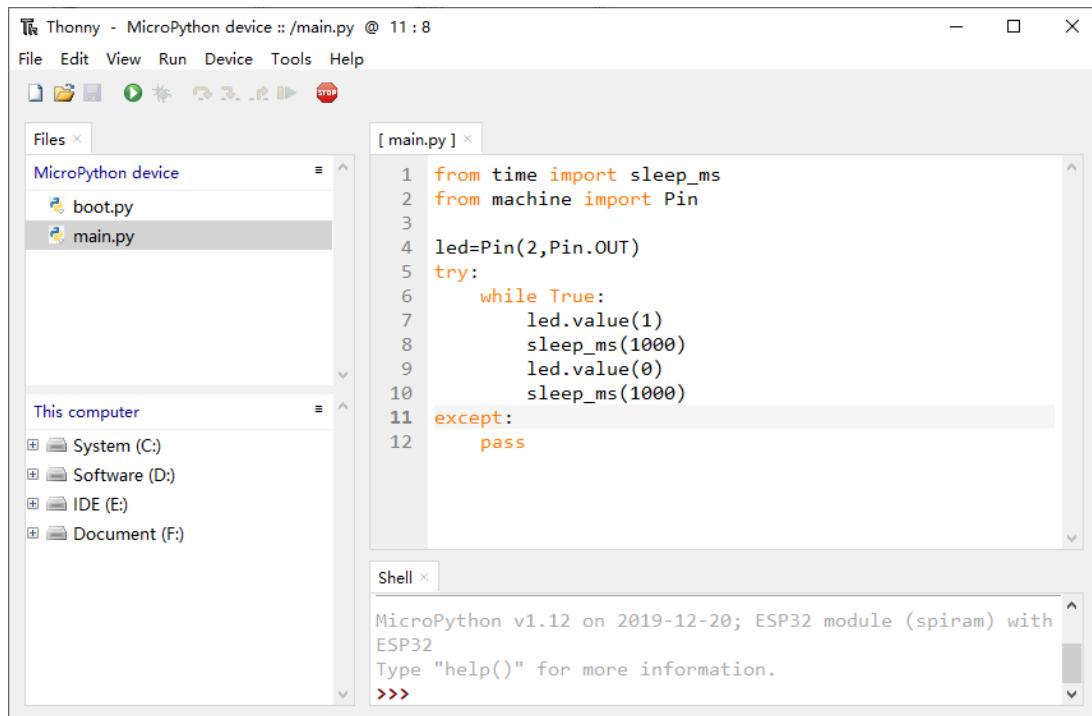
Click “Save” on the menu bar. You can save the codes either to your computer or to ESP32-WROVER.



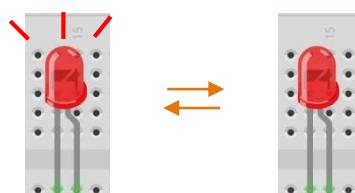
Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



You can see that codes have been uploaded to ESP32-WROVER.



Disconnect and reconnect USB cable, and you can see that LED is ON for one second and then OFF for one second, which repeats in an endless loop.



0.7 Note

Though there are many pins available on ESP32, some of them have been connected to peripheral equipment, so we should avoid using such pins to prevent pin conflicts. For example, when downloading programs, make sure that the pin state of Strapping Pin, when resetting, is consistent with the default level; do NOT use Flash Pin; Do NOT use Cam Pin when using Camera function.

Strapping Pin

The state of Strapping Pin can affect the functions of ESP32 after it is reset, as shown in the table below.

Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3 V	1.8 V		
MTDI	Pull-down	0	1		
Booting Mode					
Pin	Default	SPI Boot	Download Boot		
GPIO0	Pull-up	1	0		
GPIO2	Pull-down	Don't-care	0		
Enabling/Disabling Debugging Log Print over U0TXD During Booting					
Pin	Default	U0TXD Active	U0TXD Silent		
MTDO	Pull-up	1	0		
Timing of SDIO Slave					
Pin	Default	Falling-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

If you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through support@freenove.com at any time.

or check: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

Flash Pin

GPIO6-11 has been used to connect the integrated SPI flash on the module, and is used when GPIO 0 is power on and at high level. Flash is related to the operation of the whole chip, so the external pin GPIO6-11 cannot be used as an experimental pin for external circuits, otherwise it may cause errors in the operation of the program.

GPIO16-17 has been used to connect the integrated PSRAM on the module.

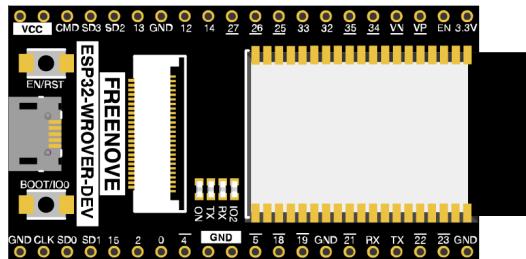
Because of external pull-up, MTDI pin is not suggested to be used as a touch sensor. For details, please refer to Peripheral Interface and Sensor chapter in "[ESP32 Data Sheet](#)".

For more relevant information, please click:

https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

Cam Pin

When using the cam camera of our ESP32-WROVER, please check the pins of it. Pins with underlined numbers are used by the cam camera function, if you want to use other functions besides it, please avoid using them.



CAM_Pin	GPIO_pin
I2C_SDA	GPIO26
I2C_SCL	GPIO27
CSI_VSYNC	GPIO25
CSI_HREF	GPIO23
CSI_Y9	GPIO35
XCLK	GPIO21
CSI_Y8	GPIO34
CSI_Y7	GPIO39
CSI_PCLK	GPIO22
CSI_Y6	GPIO36
CSI_Y2	GPIO4
CSI_Y5	GPIO19
CSI_Y3	GPIO5
CSI_Y4	GPIO18

If you have any questions about the information of GPIO, you can click [here](#) to go back to ESP32-WROVER to view specific information about GPIO.

Or check: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

Chapter 1 LED

This chapter is the Start Point in the journey to build and explore ESP32 electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

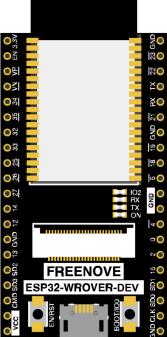
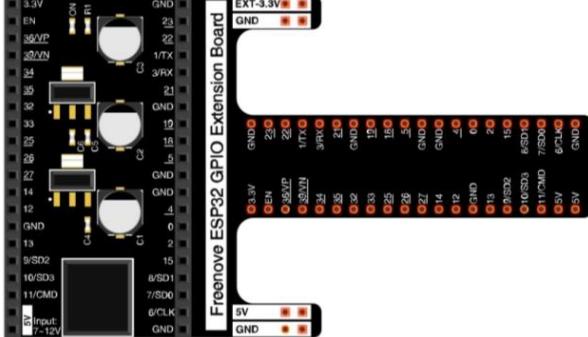
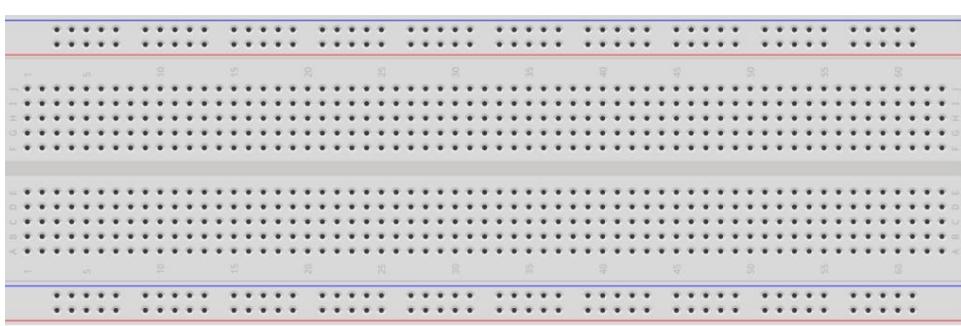
In this project, we will use ESP32 to control blinking a common LED.

If you have not yet installed Thonny, click [here](#).

If you have not yet downloaded MicroPython Firmware, click [here](#).

If you have not yet loaded MicroPython Firmware, click [here](#).

Component List

ESP32-WROVER x1	GPIO Extension Board x1	Breadboard x1
		

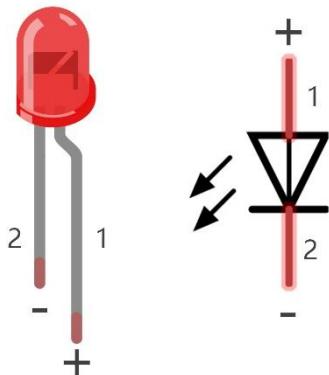
LED x1	Resistor 220Ω x1	Jumper M/M x2
		

Component knowledge

LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-). Negative output 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 burn out.



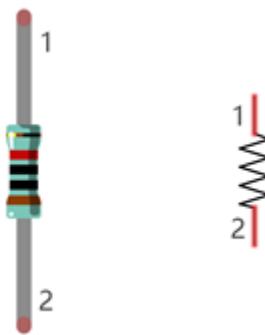
LED	Voltage	Maximum current	Recommended current
Red	1.9 - 2.2V	20mA	10mA
Green	2.9 - 3.4V	10mA	5mA
Blue	2.9 - 3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

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

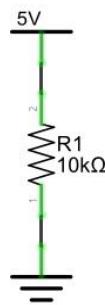
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the appendix of this tutorial.

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

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

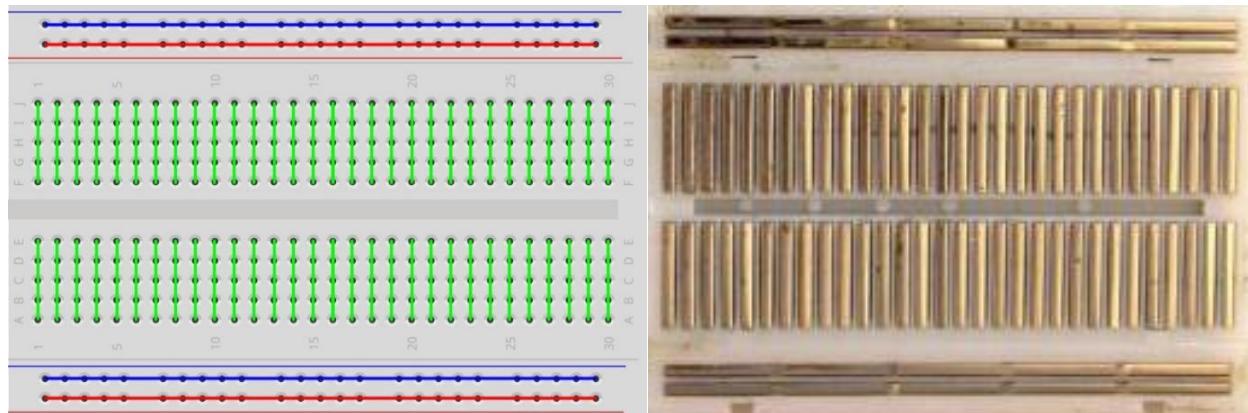


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

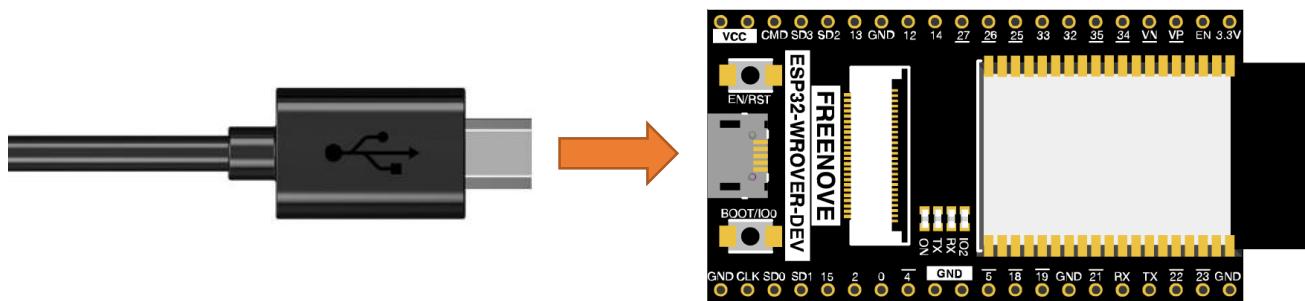
Breadboard

Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.



Power

ESP32-WROVER needs 5v power supply. In this tutorial, we need connect ESP32-WROVER to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



Later, we only use USB cable to power ESP32-WROVER in default.

In the whole tutorial, we don't use T extension to power ESP32-WROVER. So 5V and 3.3V (include EXT 3.3V) on the extension board are from ESP32-WROVER.

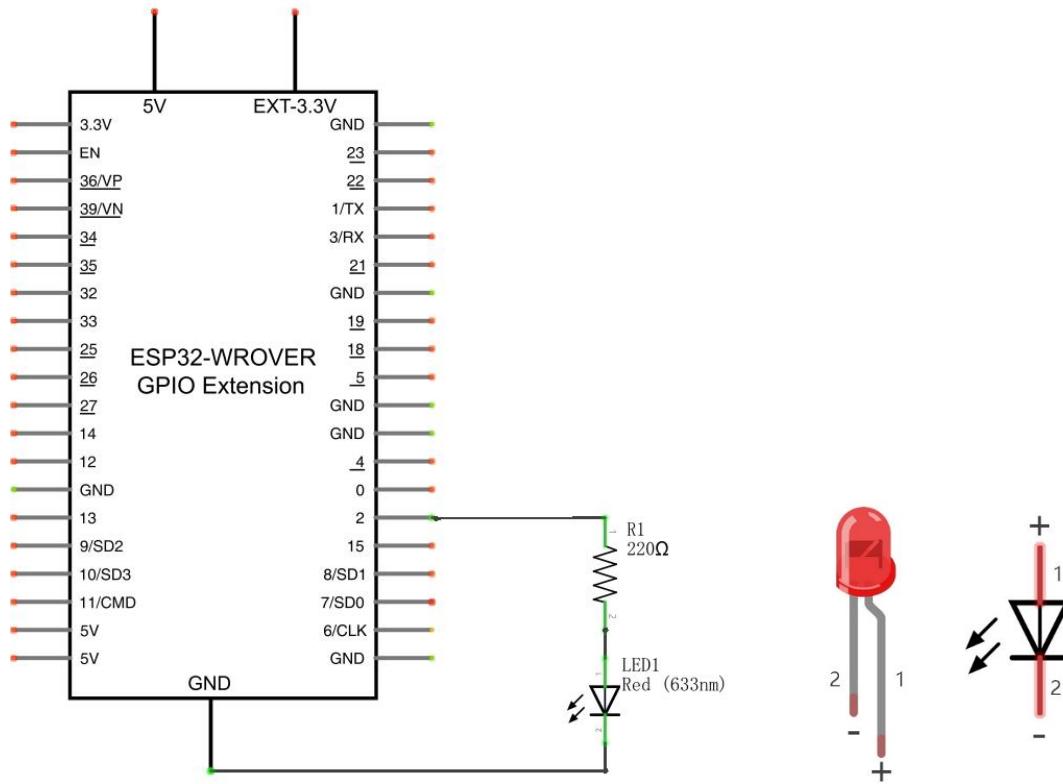
We can also use DC jack of extension board to power ESP32-WROVER. Then 5v and EXT 3.3v on extension board are from external power resource.

Circuit

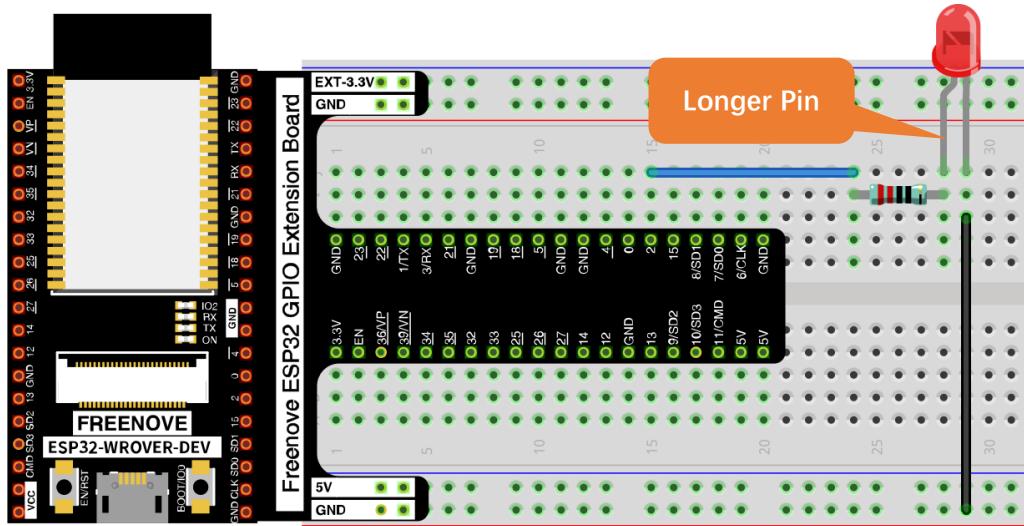
First, disconnect all power from the ESP32-WROVER. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP32-WROVER.

CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)! **WARNING:** A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram



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



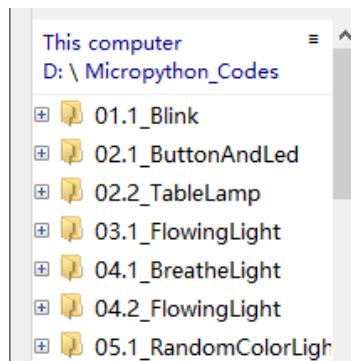
Don't rotate ESP32-WROVER 180° for connection.

Code

Codes used in this tutorial are saved in “**Freenove_Ultimate_Kit_for_ESP32/Micropython_Codes**”. You can move the codes to any location. For example, we save the codes in Disk(D) with the path of “**D:/Micropython_Codes**”.

01.1_Blink

Open “Thonny”, click “This computer”→“D:”→“Micropython_Codes”.



Expand folder “01.1_Blink” and double click “01.1_Blink.py” to open it. As shown in the illustration below.

```

from time import sleep_ms
from machine import Pin

led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
try:
    while True:
        led.value(1)      #Set led turn on
        sleep_ms(1000)
        led.value(0)      #Set led turn off
        sleep_ms(1000)
except:
    pass

```

Make sure ESP32 has been connected with the computer with ESP32 correctly. Click “Stop/Restart backend” or press the reset button, and then wait to see what interface will show up.

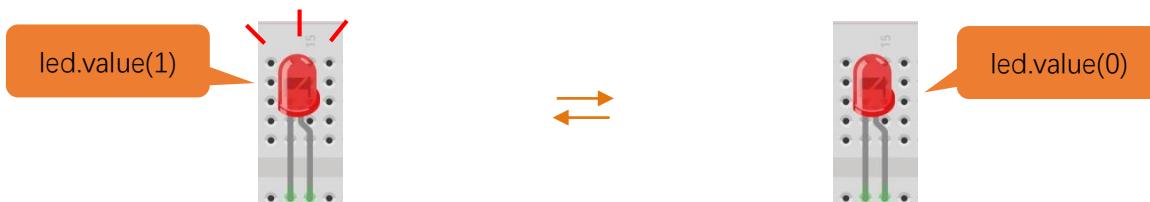
```

1 from time import sleep_ms
2 import machine
3 led = Pin(2,Pin.OUT) #create LED object from pin2,Set Pin 2 as output
4
5 while True:
6     led.value(1)           #Set led turn on
7     sleep_ms(1000)
8     led.value(0)           #Set led turn off
9     sleep_ms(1000)
10 except:
11     pass

```

This indicates that the connection is successful.

Click “Run current script” shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



Note:

This is the code [running online](#). If you disconnect USB cable and repower ESP32 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Device, Tools, and Help. The main window has two panes: 'Files' on the left and '01.1_Blink.py' on the right. The 'Files' pane lists several Python files under 'D:\Micropython_Codes\01.1_Blink'. The '01.1_Blink.py' pane contains the following code:

```

from time import sleep_ms
from machine import Pin

led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
try:
    while True:
        led.value(1)      #Set led turn on
        sleep_ms(1000)
        led.value(0)      #Set led turn off
        sleep_ms(1000)
except:
    pass

```

The 'Shell' pane at the bottom shows error messages related to connection issues:

```

raise ConnectionClosedException(self._error)
thonny.plugins.micropython.connection.ConnectionClosedException: GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

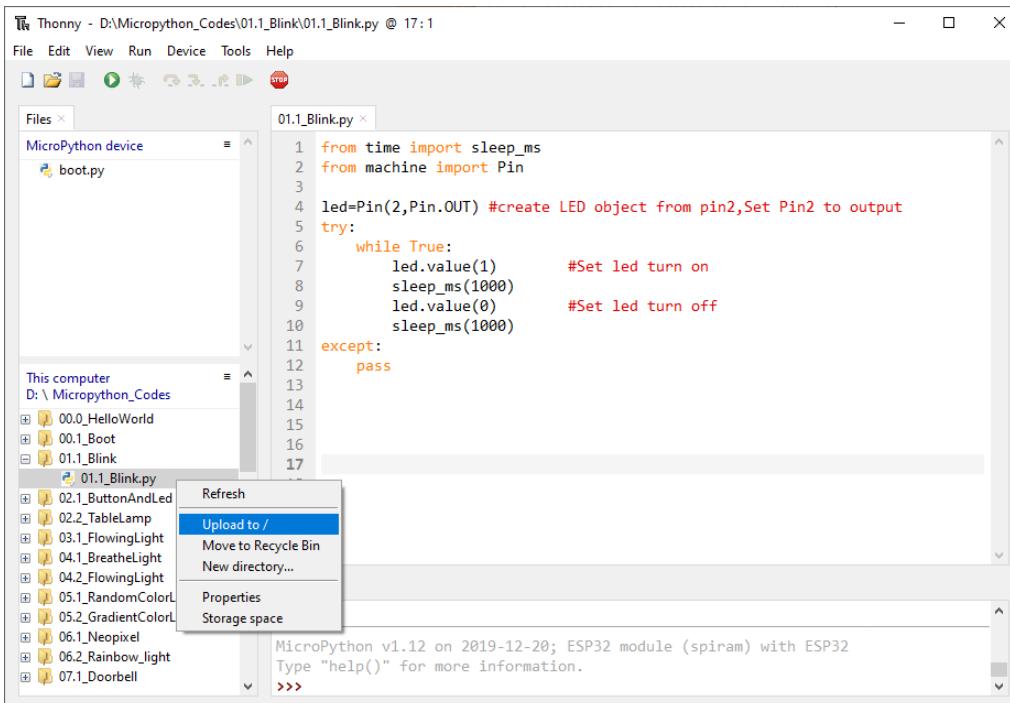
>>>
Connection lost (GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5)))

Use Stop/Restart to reconnect.

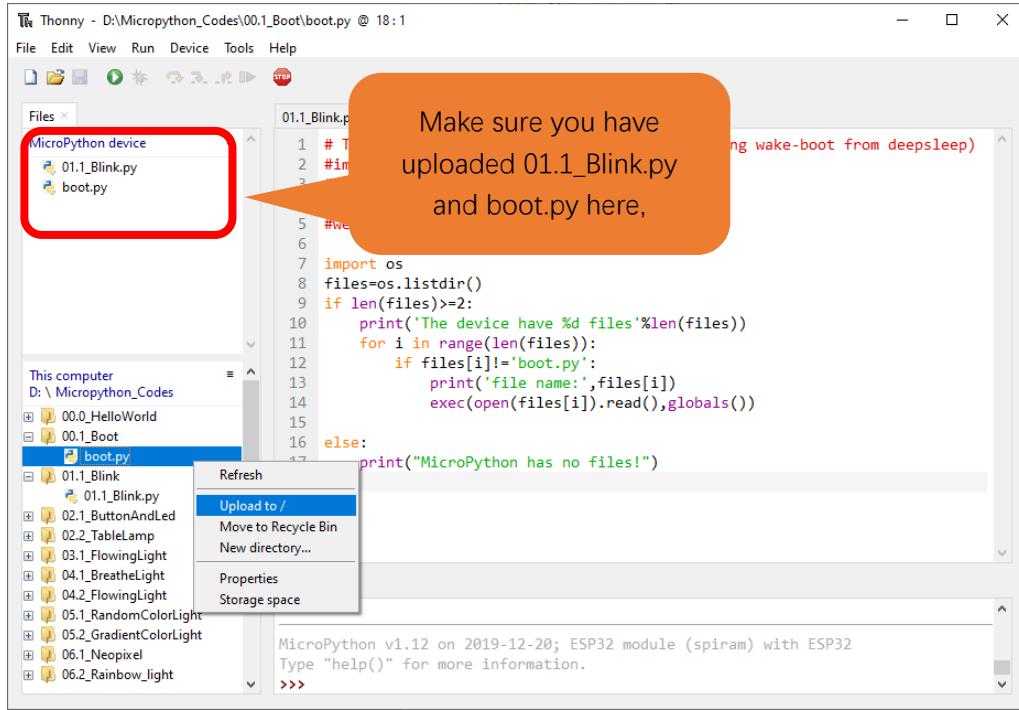
```

Uploading code to ESP32

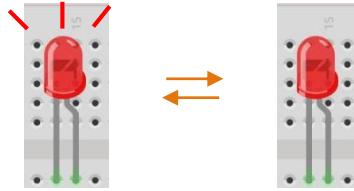
As shown in the following illustration, right-click the file 01.1_Blink.py and select “Upload to /” to upload code to ESP32.



Upload boot.py in the same way.

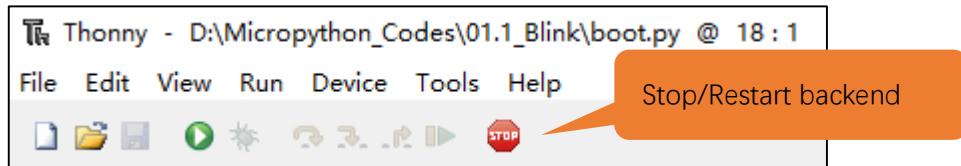


Press the reset key of ESP32 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



Note:

Codes here is run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.

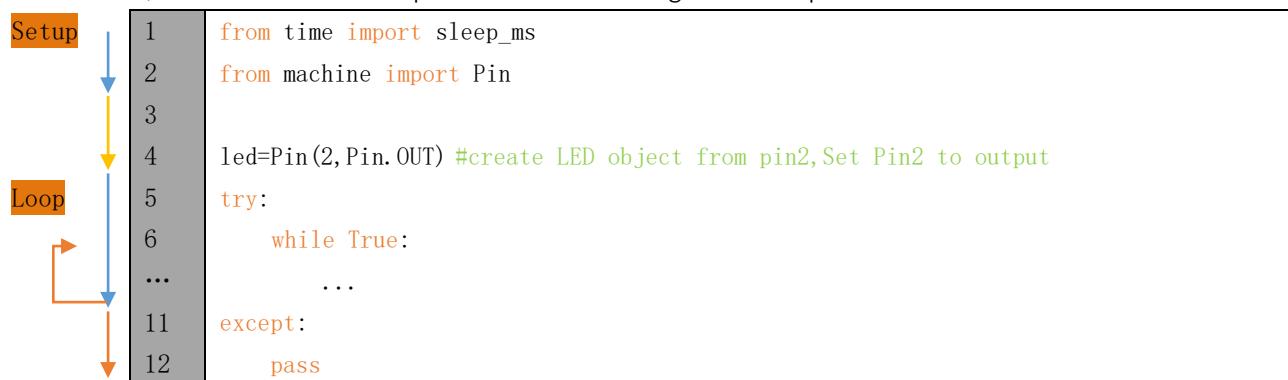


If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```
1 from time import sleep_ms
2 from machine import Pin
3
4 led=Pin(2,Pin.OUT) #create LED object from pin2, Set Pin2 to output
5 try:
6     while True:
7         led.value(1) #Set led turn on
8         sleep_ms(1000)
9         led.value(0) #Set led turn off
10        sleep_ms(1000)
11 except:
12     pass
```

Each time a new file is opened, the program will be executed from top to bottom. When encountering a loop construction, it will execute the loop statement according to the loop condition.



`Print()` function is used to print data to Terminal. It can be executed in Terminal directly or be written in a Python file and executed by running the file.

```
print("Hello world!")
```

Each time when using the functions of ESP32, you need to import modules corresponding to those functions:
Import sleep_ms module of time module and Pin module of machine module.

```
1 from time import sleep_ms  
2 from machine import Pin
```

Configure GPIO2 of ESP32-WROVER to output mode and assign it to an object named "led".

```
4     led=Pin(2,Pin.OUT) #create LED object from pin2, Set Pin2 to output
```

It means that from now on, LED represents 1.

```
7 led.value(1) #Set led turn on
```

Set the value of LED to 0 and GPIO2 will output low level

```
9     led.value(0) #Set led turn on
```

Execute codes in a while loop.

```
6     while True:
```

For more information about the study, please contact Dr. John Smith at (555) 123-4567 or email him at john.smith@researchinstitute.org.



Put statements that may cause an error in “try” block and the executing statements when an error occurs in “except” block. In general, when the program executes statements, it will execute those in “try” block.

However, when an error occurs to ESP32 due to some interference or other reasons, it will execute statements in “except” block.

“Pass” is an empty statement. When it is executed, nothing happens. It is useful as a placeholder to make the structure of a program look better.

```
5   try:  
...  
11  ...  
12  except:  
    pass
```

The single-line comment of Micropython starts with a “#” and continues to the end of the line. Comments help us to understand code. When programs are running, Thonny will ignore comments.

```
9 #Set led turn on
```

MicroPython uses indentations to distinguish different blocks of code instead of braces. The number of indentations is changeable, but it must be consistent throughout one block. If the indentation of the same code block is inconsistent, it will cause errors when the program runs.

```
6 while True:  
7     led.value(1) #Set led turn on  
8     sleep_ms(1000)  
9     led.value(0) #Set led turn off  
10    sleep_ms(1000)
```

How to import python files

Whether to import the built-in python module or to import that written by users, the command “import” is needed.

If you import the module directly you should indicate the module to which the function or attribute belongs when using the function or attribute (constant, variable) in the module. The format should be: <module name>.<function or attribute>, otherwise an error will occur.

```
import random  
  
num = random.randint(1, 100)  
print(num)
```

If you only want to import a certain function or attribute in the module, use the from...import statement. The format is as follows

```
from random import randint  
num = randint(1, 100)  
print(num)
```

When using “from...import” statement to import function, to avoid conflicts and for easy understanding, you can use “as” statement to rename the imported function, as follows

```
from random import randint as rand  
num = rand(1, 100)  
print(num)
```

Reference

Class machine

Before each use of the **machine** module, please add the statement “**import machine**” to the top of python file.

machine.freq(freq_val): When freq_val is not specified, it is to return to the current CPU frequency; Otherwise, it is to set the current CPU frequency.

freq_val: 80000000(80MHz)、160000000(160MHz)、240000000(240MHz)

machine.reset(): A reset function. When it is called, the program will be reset.

machine.unique_id(): Obtains MAC address of the device.

machine.idle(): Turns off any temporarily unused functions on the chip and its clock, which is useful to reduce power consumption at any time during short or long periods.

machine.disable_irq(): Disables interrupt requests and return the previous IRQ state. The disable_irq () function and enable_irq () function need to be used together; Otherwise the machine will crash and restart.

machine.enable_irq(state): To re-enable interrupt requests. The parameter **state** should be the value that was returned from the most recent call to the disable_irq() function

machine.time_pulse_us(pin, pulse_level, timeout_us=1000000):

Tests the duration of the external pulse level on the given pin and returns the duration of the external pulse level in microseconds. When pulse level = 1, it tests the high level duration; When pulse level = 0, it tests the low level duration.

If the setting level is not consistent with the current pulse level, it will wait until they are consistent, and then start timing. If the set level is consistent with the current pulse level, it will start timing immediately.

When the pin level is opposite to the set level, it will wait for timeout and return “-2”. When the pin level and the set level is the same, it will also wait timeout but return “-1”. **timeout_us** is the duration of timeout.

Class Pin(id[, mode, pull, value])

Before each use of the **Pin** module, please add the statement “**from machine import Pin**” to the top of python file.

id: Arbitrary pin number

mode: Mode of pins

Pin.IN: Input Mode

Pin.OUT: Output Mode

Pin.OPEN_DRAIN: Open-drain Mode

Pull: Whether to enable the internal pull up and down mode

None: No pull up or pull down resistors

Pin.PULL_UP: Pull-up Mode, outputting high level by default

Pin.PULL_DOWN: Pull-down Mode, outputting low level by default

Value: State of the pin level, 0/1

Pin.init(mode, pull): Initialize pins

Pin.value([value]): Obtain or set state of the pin level, return 0 or 1 according to the logic level of pins.

Without parameter, it reads input level. With parameter given, it is to set output level.

value: It can be either True/False or 1/0.

Pin.irq(trigger, handler): Configures an interrupt handler to be called when the pin level meets a condition.

trigger:

Pin.IRQ_FALLING: interrupt on falling edge

Pin.IRQ_RISING: interrupt on rising edge

3: interrupt on both edges

Handler: callback function

Class time

Before each use of the **time** module, please add the statement “**import time**” to the top of python file

time.sleep(sec): Sleeps for the given number of seconds

sec: This argument should be either an int or a float.

time.sleep_ms(ms): Sleeps for the given number of milliseconds, ms should be an int.

time.sleep_us(us): Sleeps for the given number of microseconds, us should be an int.

time.time(): Obtains the timestamp of CPU, with second as its unit.

time.ticks_ms(): Returns the incrementing millisecond counter value, which recounts after some values.

time.ticks_us(): Returns microsecond

time.ticks_cpu(): Similar to ticks_ms() and ticks_us(), but it is more accurate(return clock of CPU).

time.ticks_add(ticks, delta): Gets the timestamp after the offset.

ticks: ticks_ms()、 ticks_us()、 ticks_cpu()

delta: Delta can be an arbitrary integer number or numeric expression

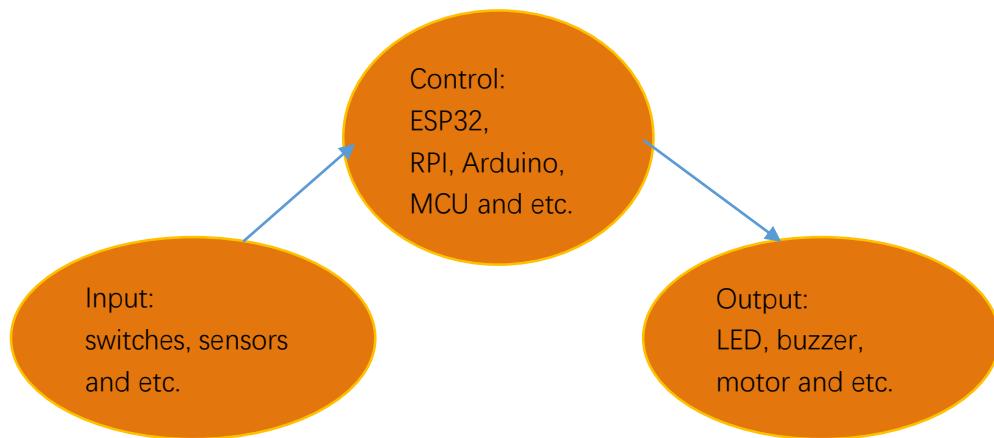
time.ticks_diff(old_t, new_t): Calculates the interval between two timestamps, such as ticks_ms(), ticks_us() or ticks_cpu().

old_t: Starting time

new_t: Ending time

Chapter 2 Button & LED

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 ESP32 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 LEDs light up, make a buzzer turn ON and so on.



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

Project 2.1 Button & LED

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



Component List

ESP32-WROVER x1 	GPIO Extension Board x1 			
Breadboard x1 				
Jumper M/M x4 	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push button x1

Component knowledge

Push button

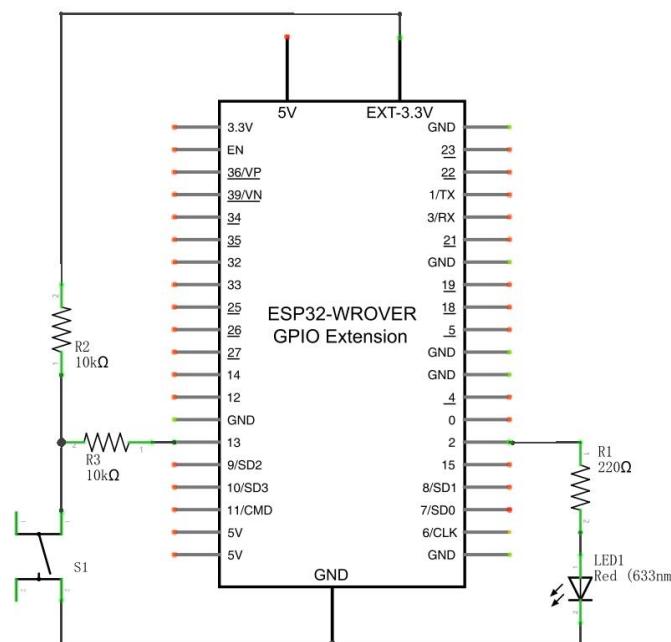
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



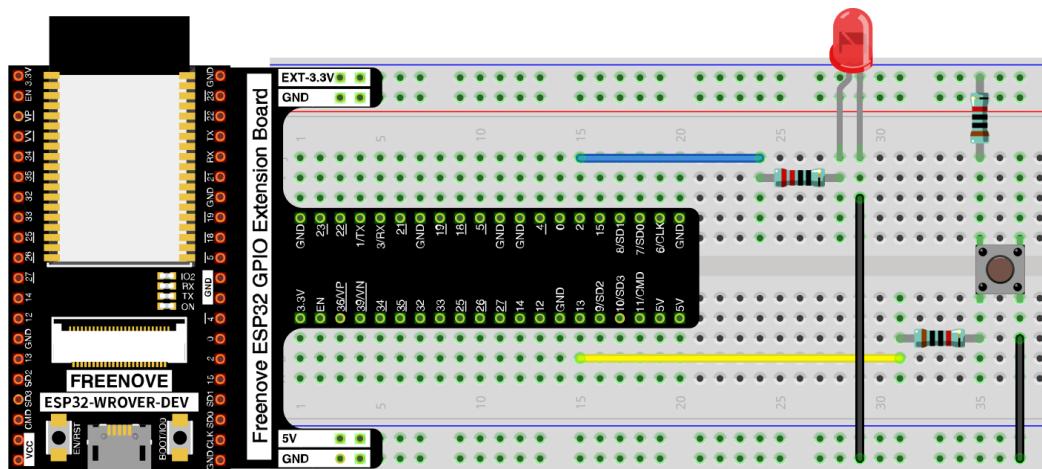
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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

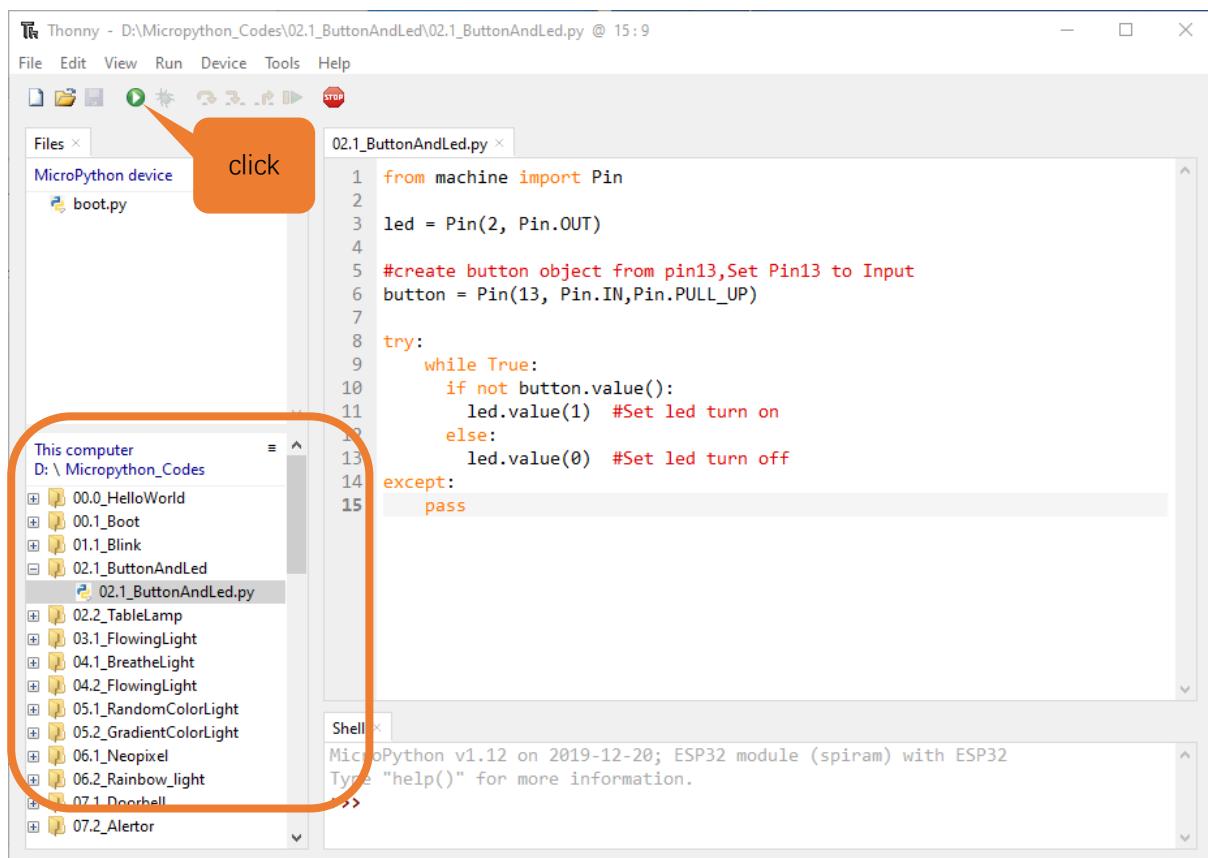


Code

This project is designed to learn to control an LED with a push button switch. First, we need to read the state of the switch and then decide whether the LED is turned on or not based on it.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “02.1_ButtonAndLed” and double click “02.1_ButtonAndLed.py”.

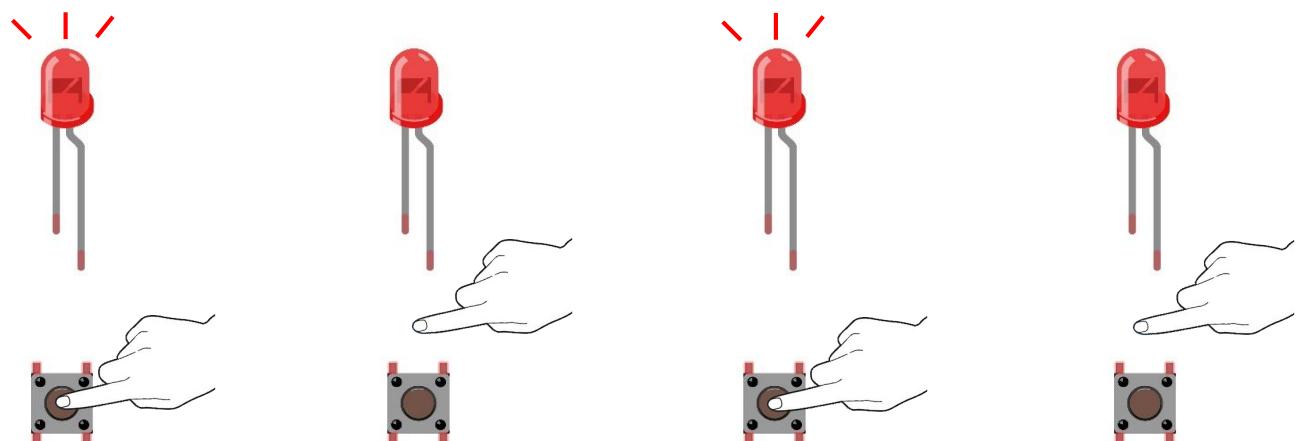
02.1 ButtonAndLed



```

1 from machine import Pin
2
3 led = Pin(2, Pin.OUT)
4
5 #create button object from pin13,Set Pin13 to Input
6 button = Pin(13, Pin.IN,Pin.PULL_UP)
7
8 try:
9     while True:
10         if not button.value():
11             led.value(1) #Set led turn on
12         else:
13             led.value(0) #Set led turn off
14 except:
15     pass
    
```

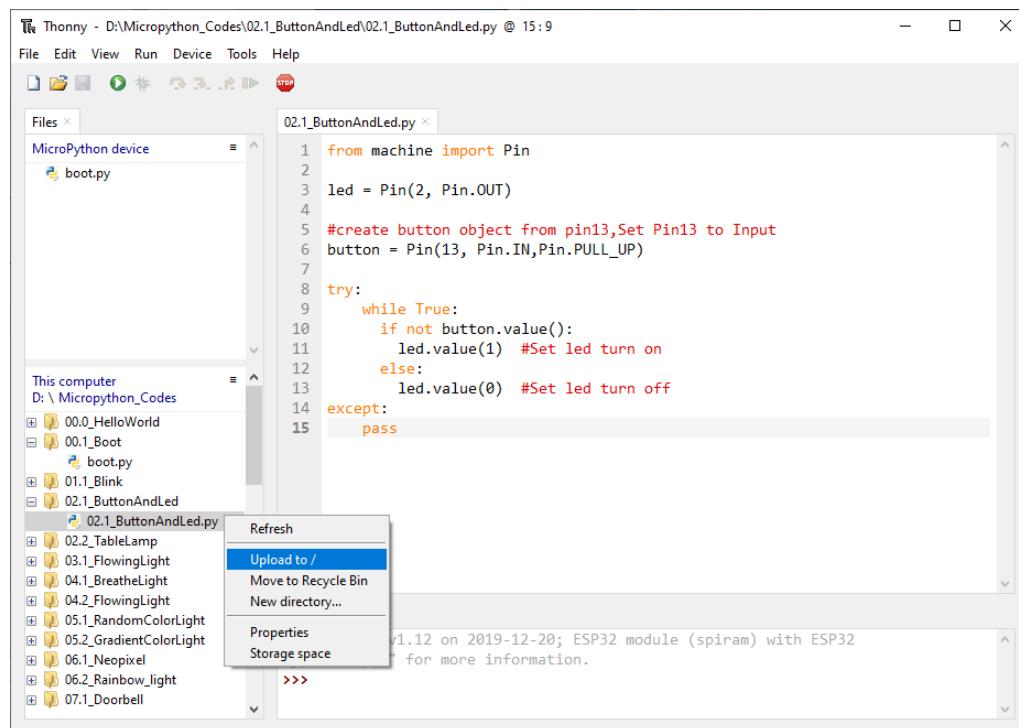
Click “Run current script” shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF.



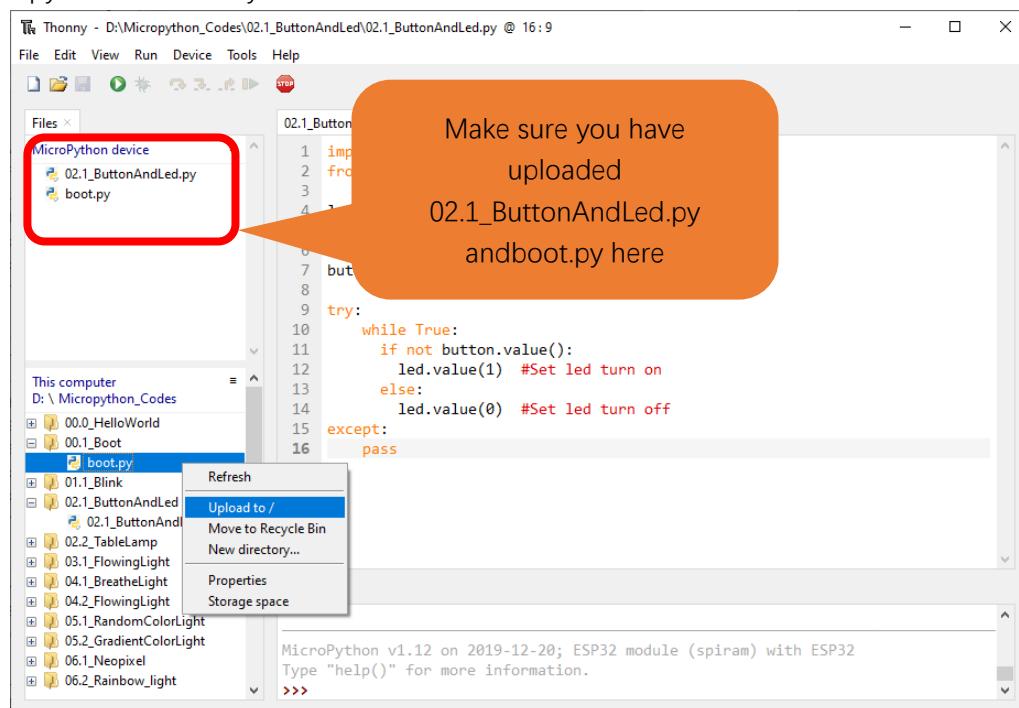
If you have any concerns, please contact us via: support@freenove.com

Upload Code to ESP32

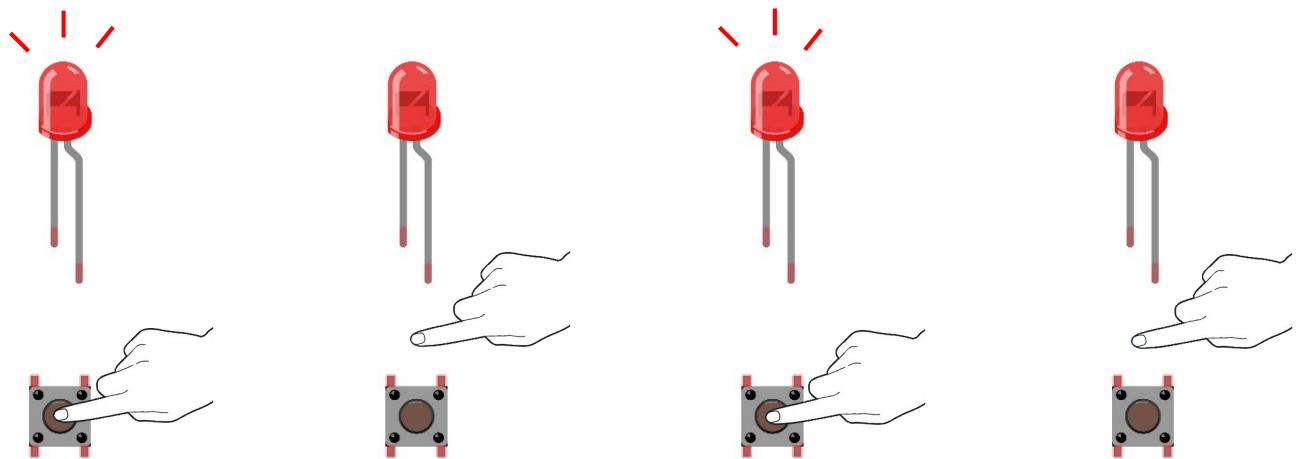
As shown in the following illustration, right-click file 02.1_ButtonAndLed and select “Upload to /” to upload code to ESP32.



Upload boot.py in the same way.



Press ESP32's reset key, and then push the button switch, LED turns ON; Push the button again, LED turns OFF.



The following is the program code:

```

1 from machine import Pin
2
3 led = Pin(2, Pin.OUT)
4
5 #create button object from pin13, Set Pin13 to Input
6 button = Pin(13, Pin.IN, Pin.PULL_UP)
7
8 try:
9     while True:
10        if not button.value():
11            led.value(1) #Set led turn on
12        else:
13            led.value(0) #Set led turn off
14 except:
15     pass

```

In this project, we use the Pin module of the machine, so before initializing the Pin, we need to import this module first.

```
1 from machine import Pin
```

In the circuit connection, LED and Button are connected with GPIO2 and GPIO13 respectively, so define led and button as 2 and 13 respectively.

```

3 led = Pin(2, Pin.OUT)
4
5 #create button object from pin13, Set Pin13 to Input
6 button = Pin(13, Pin.IN, Pin.PULL_UP)

```

Read the pin state of button with value() function. Press the button switch, the function returns low level and the result of "if" is true, and then LED will be turned ON; Otherwise, LED is turned OFF.

```
9  while True:  
10     if not button.value():  
11         led.value(1) #Set led turn on  
12     else:  
13         led.value(0) #Set led turn off
```

If statement is used to execute the next statement when a certain condition is proved to be true (or non0). It is often used together with "else" statement, which judges other statements except the if statement. If you need to judge if the result of a condition is 0, you can use if not statement.

```
10  if not button.value():  
11      ...  
12  else:  
13      ...
```



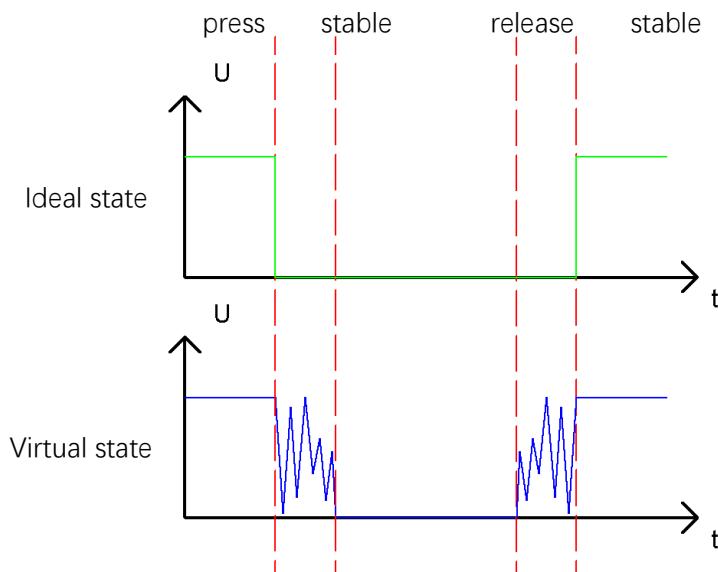
Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and ESP32 to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

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

Debounce for Push Button

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



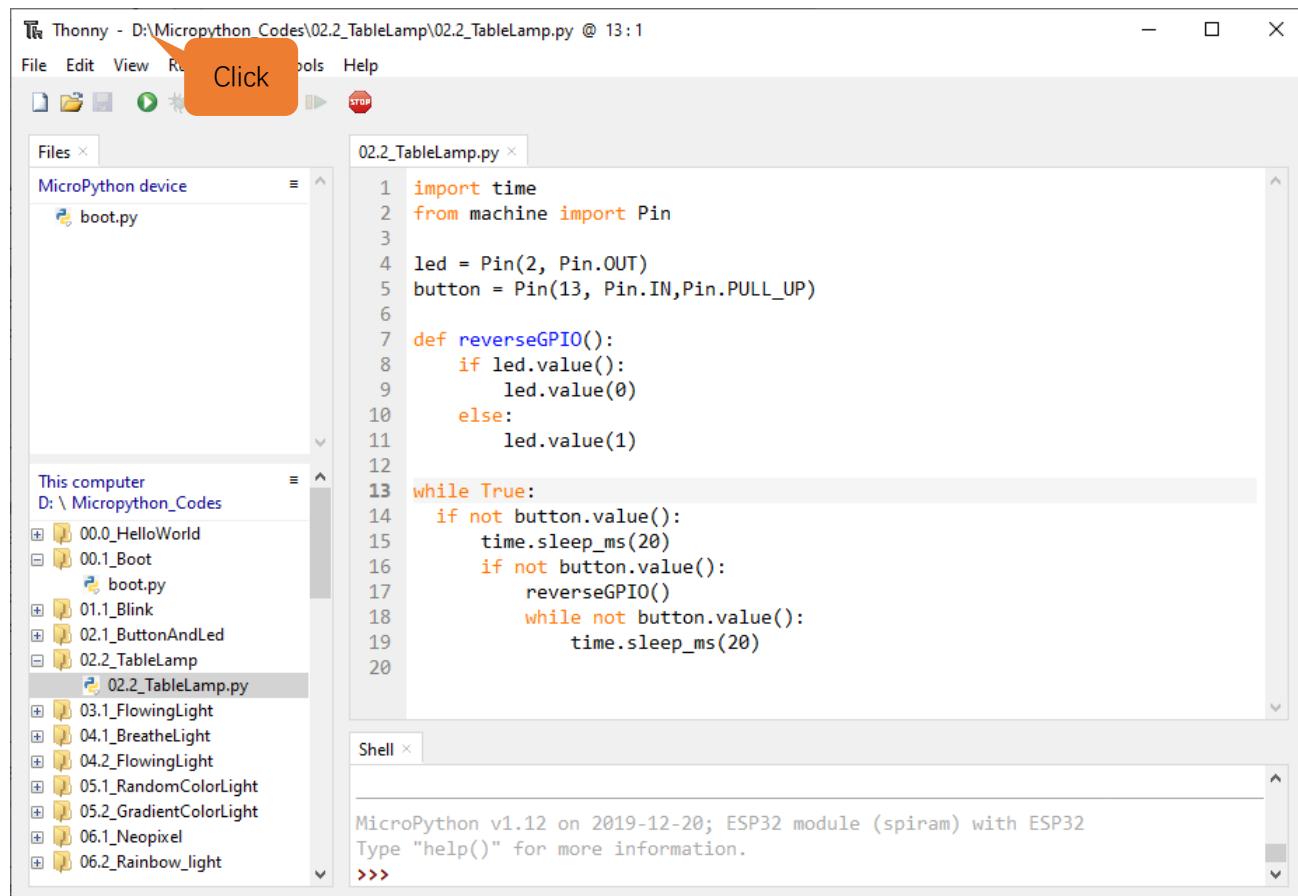
Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting.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

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

Code

02.2 Tablelamp

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “02.2_TableLamp” and double click “02.2_TableLamp.py”.

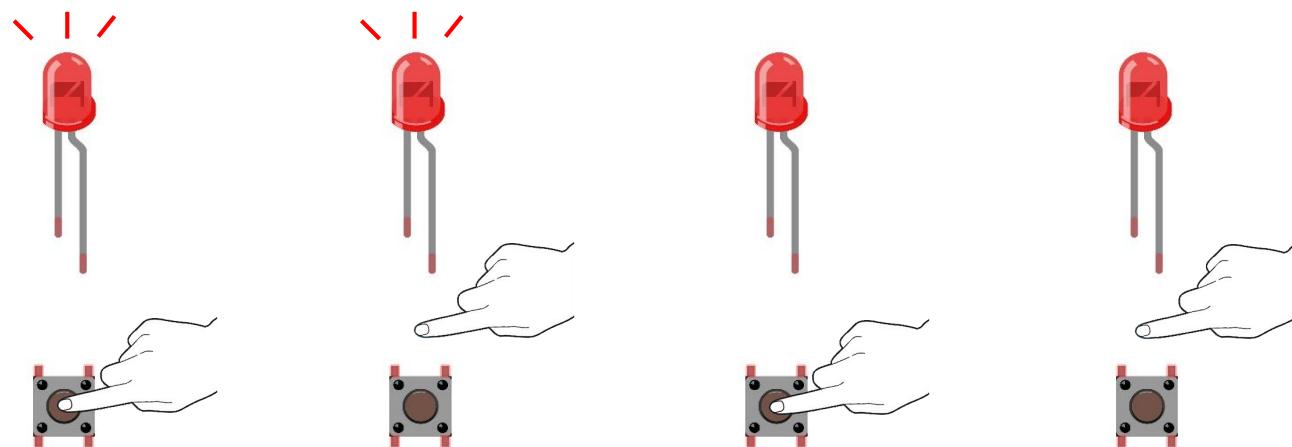


```

Thonny - D:\Micropython_Codes\02.2_TableLamp\02.2_TableLamp.py @ 13:1
File Edit View Run Tools Help
Files × 02.2_TableLamp.py ×
MicroPython device
boot.py
This computer
D:\Micropython_Codes
00.0_HelloWorld
00.1_Boot
boot.py
01.1_Blink
02.1_ButtonAndLed
02.2_TableLamp
02.2_TableLamp.py
03.1_FlowingLight
04.1_BreatheLight
04.2_FlowingLight
05.1_RandomColorLight
05.2_GradientColorLight
06.1_Neopixel
06.2_Rainbow_light
Shell ×
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

```

Click “Run current script” shown in the box of the above illustration, press the push button switch, LED turns ON; press it again, LED turns OFF.

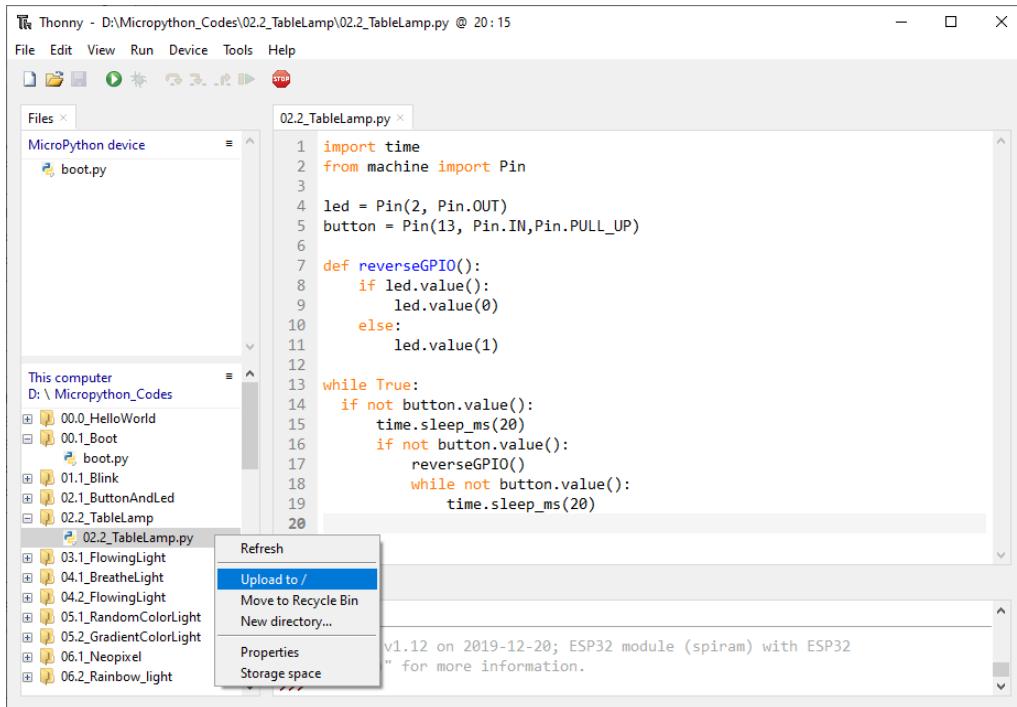


If you have any concerns, please contact us via: support@freenove.com

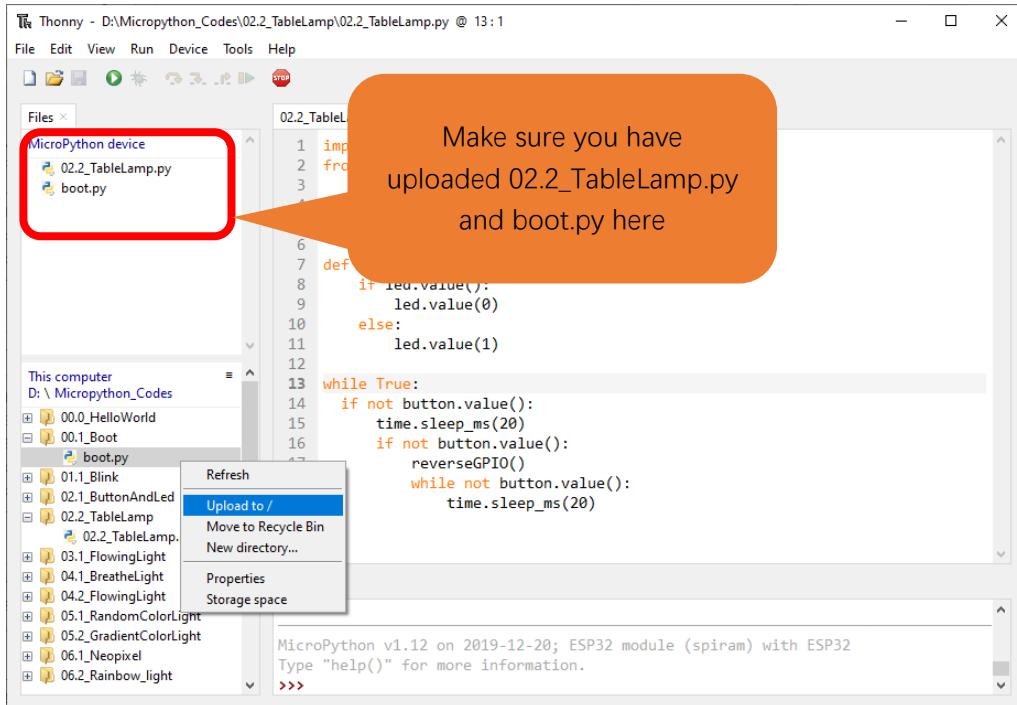


Upload code to ESP32

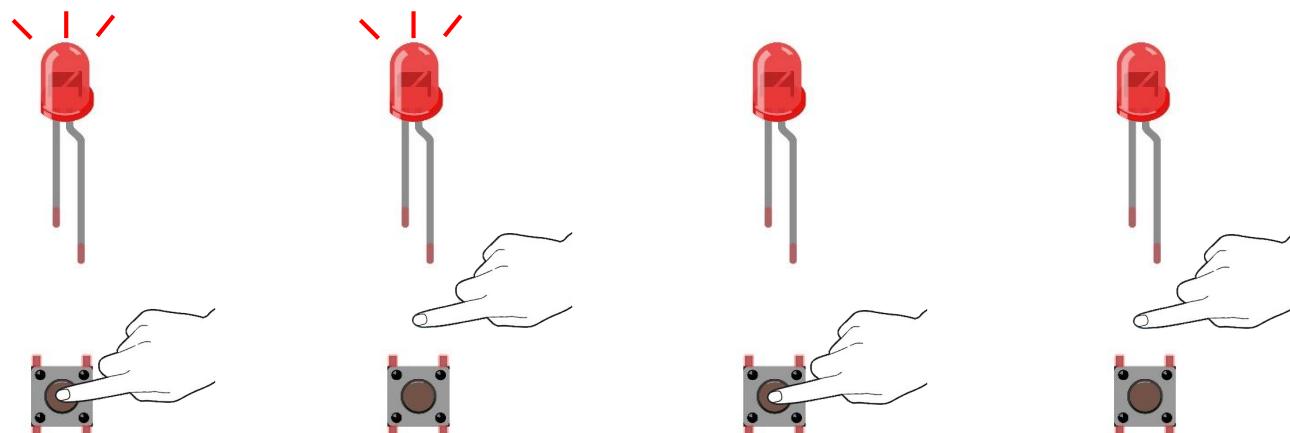
As shown in the following illustration, right-click file 02.2_TableLamp and select “Upload to /” to upload code to ESP32.



Upload boot.py in the same way.



Press ESP32's reset key, and then push the button switch, LED turns ON; Push the button again, LED turns OFF.



The following is the program code:

```

1 import time
2 from machine import Pin
3
4 led = Pin(2, Pin.OUT)
5 button = Pin(13, Pin.IN, Pin.PULL_UP)
6
7 def reverseGPIO():
8     if led.value():
9         led.value(0)
10    else:
11        led.value(1)
12
13 while True:
14     if not button.value():
15         time.sleep_ms(20)
16         if not button.value():
17             reverseGPIO()
18             while not button.value():
19                 time.sleep_ms(20)
20

```

When the button is detected to be pressed, delay 20ms to avoid the effect of bounce, and then check whether the button has been pressed again. If so, the conditional statement will be executed, otherwise it will not be executed.

```

13 while True:
14     if not button.value():
15         time.sleep_ms(20)
16         if not button.value():
17             reverseGPIO()
18             while not button.value():
19                 time.sleep_ms(20)

```



Customize a function and name it reverseGPIO(), which reverses the output level of the LED.

```
7  def reverseGPIO():
8      if led.value():
9          led.value(0)
10     else:
11         led.value(1)
```

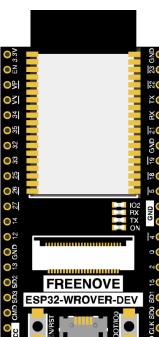
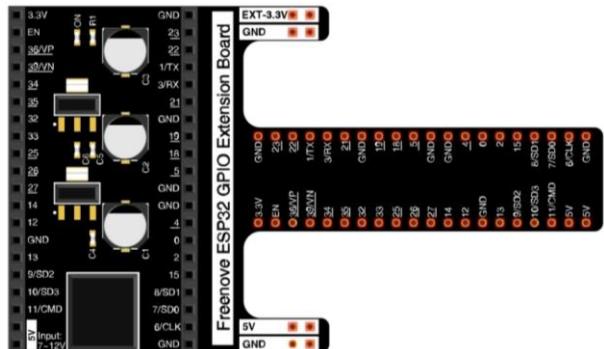
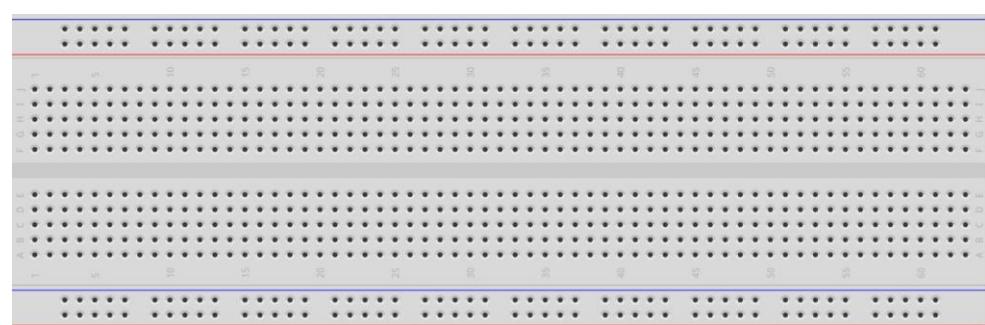
Chapter 3 LED Bar

We have learned how to control a LED blinking, next we will learn how to control a number of LEDs.

Project 3.1 Flowing Light

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

Component List

ESP32-WROVER x1	GPIO Extension Board x1	Breadboard x1	Jumper M/M x10	LED bar graph x1	Resistor 220Ω x10
					

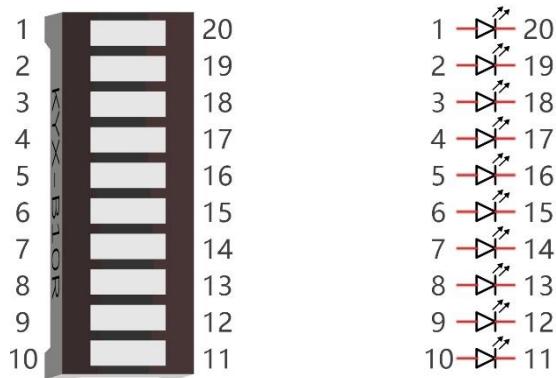


Component knowledge

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

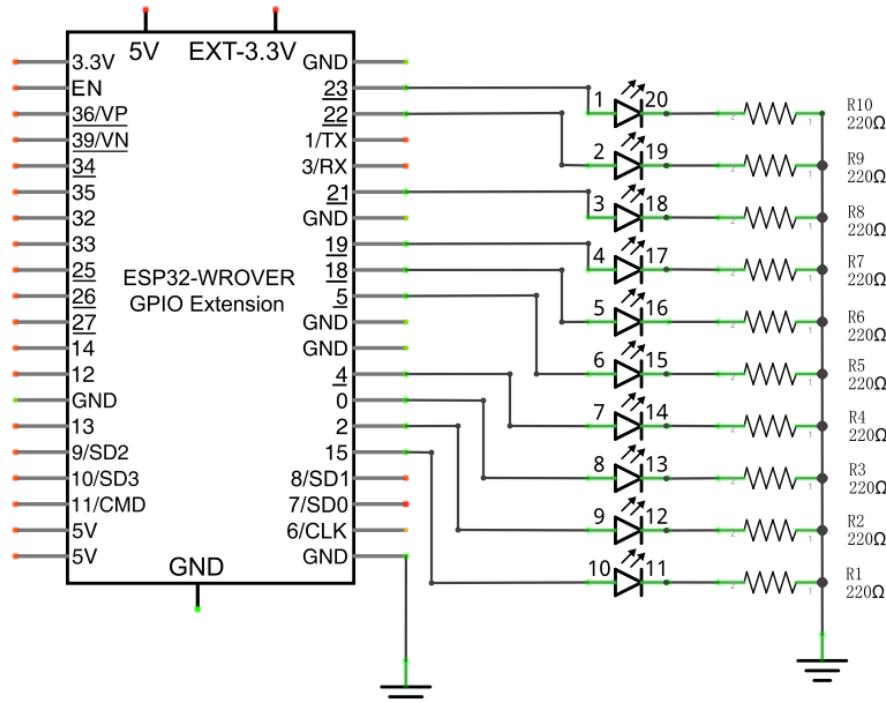
LED bar

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

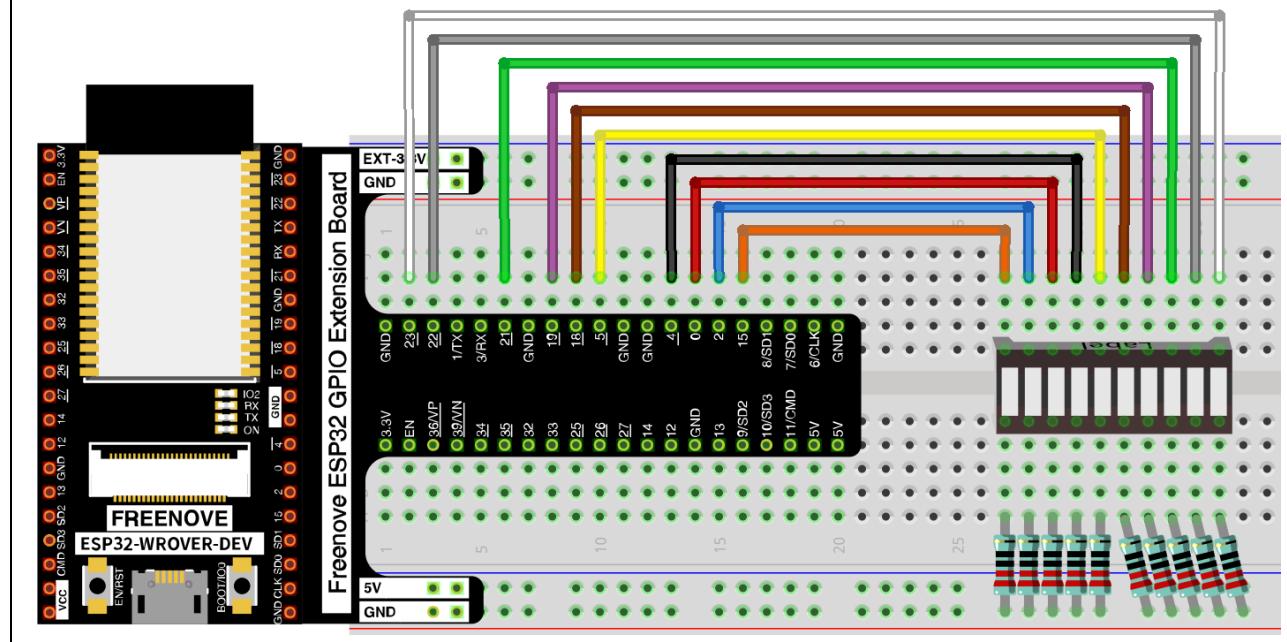


Circuit

Schematic diagram



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



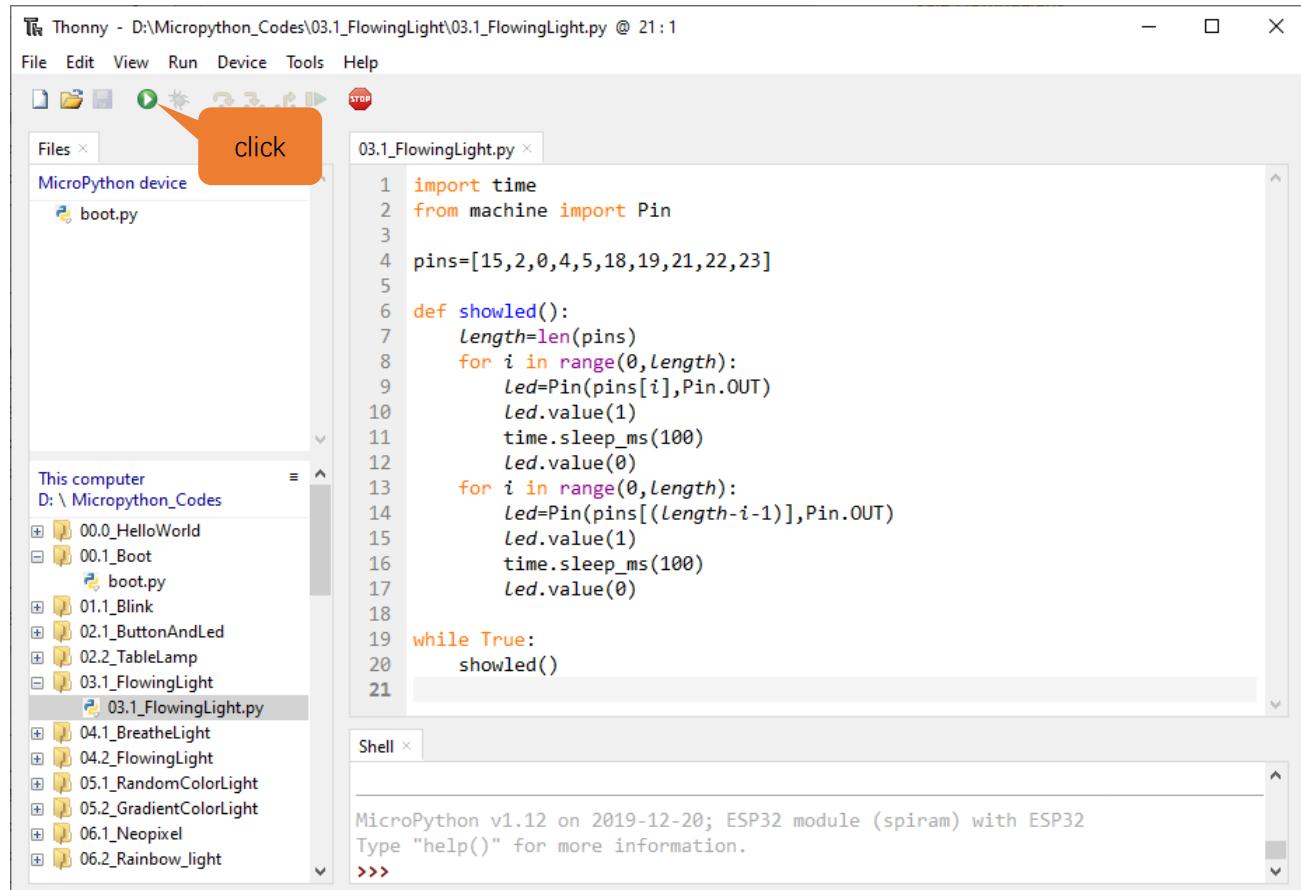
If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

Code

This project is designed to make a flowing water lamp. Which are these actions: First turn LED #1 ON, then turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the “movements” of flowing water.

03.1 FlowingLight

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “03.1_FlowingLight” and double click “03.1_FlowingLight.py”.

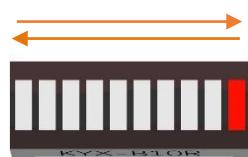


```

1 import time
2 from machine import Pin
3
4 pins=[15,2,0,4,5,18,19,21,22,23]
5
6 def showed():
7     length=len(pins)
8     for i in range(0,length):
9         Led=Pin(pins[i],Pin.OUT)
10        Led.value(1)
11        time.sleep_ms(100)
12        led.value(0)
13    for i in range(0,length):
14        Led=Pin(pins[(length-i-1)],Pin.OUT)
15        Led.value(1)
16        time.sleep_ms(100)
17        led.value(0)
18
19 while True:
20     showed()
21

```

Click “Run current script” shown in the box above, LED Bar Graph will light up from left to right and then back from right to left.



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

The following is the program code:

```
1 import time
2 from machine import Pin
3
4 pins=[15, 2, 0, 4, 5, 18, 19, 21, 22, 23]
5
6 def showled():
7     length=len(pins)
8     for i in range(0, length):
9         led=Pin(pins[i], Pin.OUT)
10        led.value(1)
11        time.sleep_ms(100)
12        led.value(0)
13     for i in range(0, length):
14         led=Pin(pins[(length-i-1)], Pin.OUT)
15         led.value(1)
16         time.sleep_ms(100)
17         led.value(0)
18
19 while True:
20     showled()
```

Use an array to define 10 GPIO ports connected to LED Bar Graph for easier operation.

```
4 pins=[15, 2, 0, 4, 5, 18, 19, 21, 22, 23]
```

Use len() function to obtain the amount of elements in the list and use a for loop to configure pins as output mode.

```
7 length=len(pins)
8 for i in range(0, length):
9     led=Pin(pins[i], Pin.OUT)
```

Use two for loops to turn on LEDs separately from left to right and then back from right to left.

```
8 for i in range(0, length):
9     led=Pin(pins[i], Pin.OUT)
10    led.value(1)
11    time.sleep_ms(100)
12    led.value(0)
13 for i in range(0, length):
14     led=Pin(pins[(length-i-1)], Pin.OUT)
15     led.value(1)
16     time.sleep_ms(100)
17     led.value(0)
```



Reference

for i in range(start,end,num: int=1)

For loop is used to execute a program endlessly and iterate in the order of items (a list or a string) in the sequence

start: The initial value, the for loop starts with it

end: The ending value, the for loop end with it

num: Num is automatically added each time to the data. The default value is 1

Chapter 4 Analog & PWM

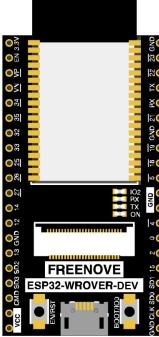
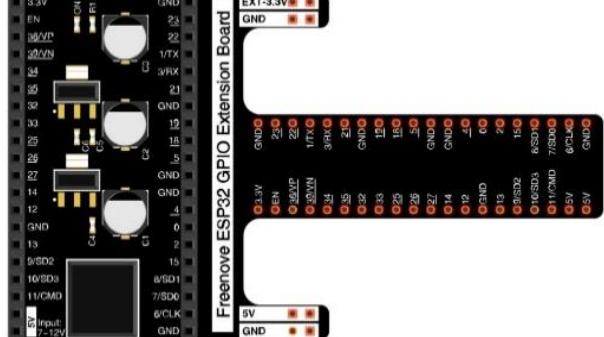
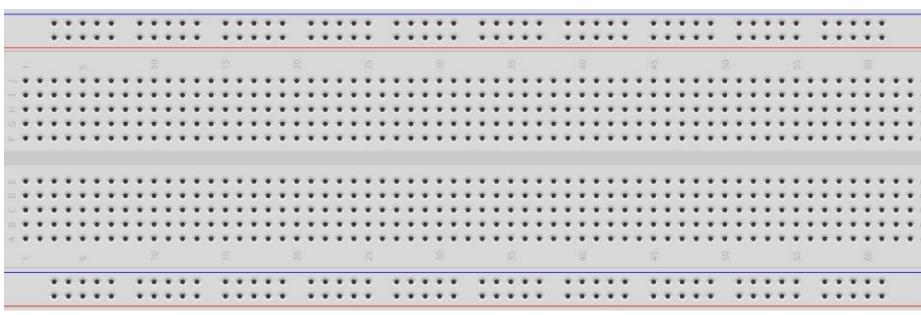
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That's what we're going to learn.

First, let's learn how to control the brightness of a LED.

Project 4.1 Breathing LED

Breathing light, that is, LED is turned from off to on gradually, and gradually from on to off, just like "breathing". So, how to control the brightness of a LED? We will use PWM to achieve this target.

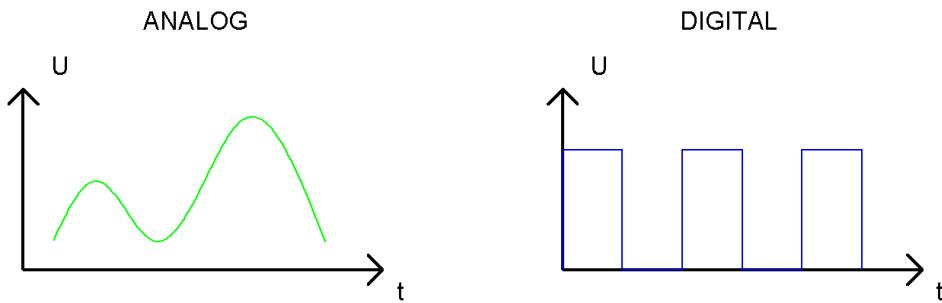
Component List

ESP32-WROVER x1	GPIO Extension Board x1	
		
Breadboard x1		
		
LED x1	Resistor 220Ω x1	Jumper M/M x2
		

Related knowledge

Analog & Digital

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



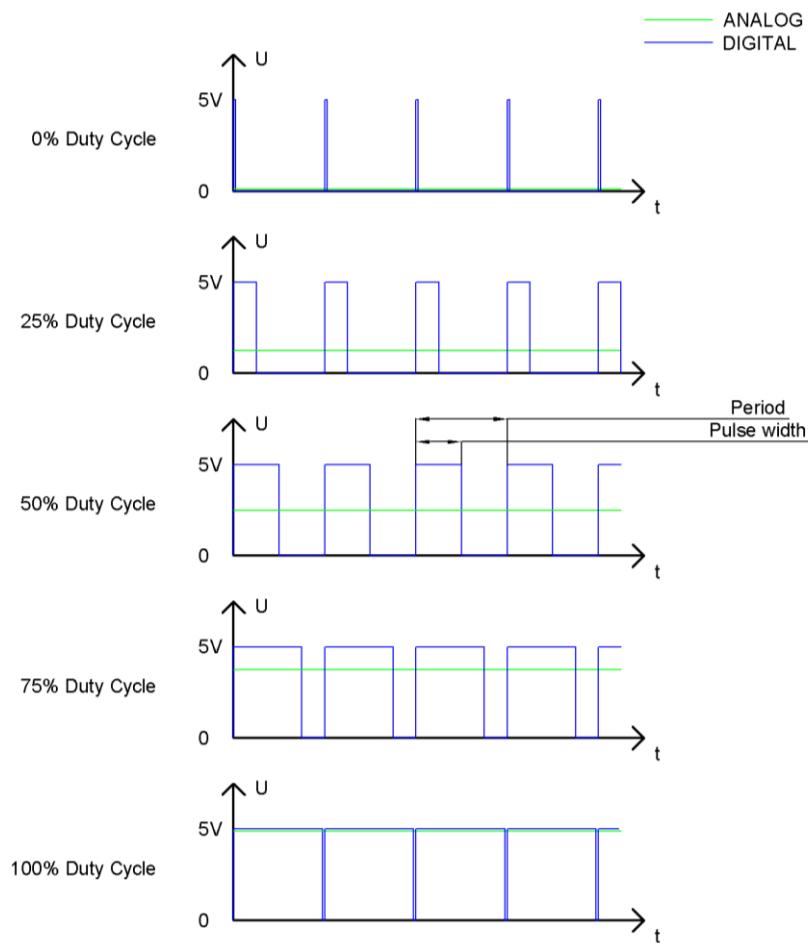
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Common 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 real analog, and the effective value of the voltage is equivalent to the corresponding analog. so, we can control the output power of the LED and other output modules to achieve different effects.

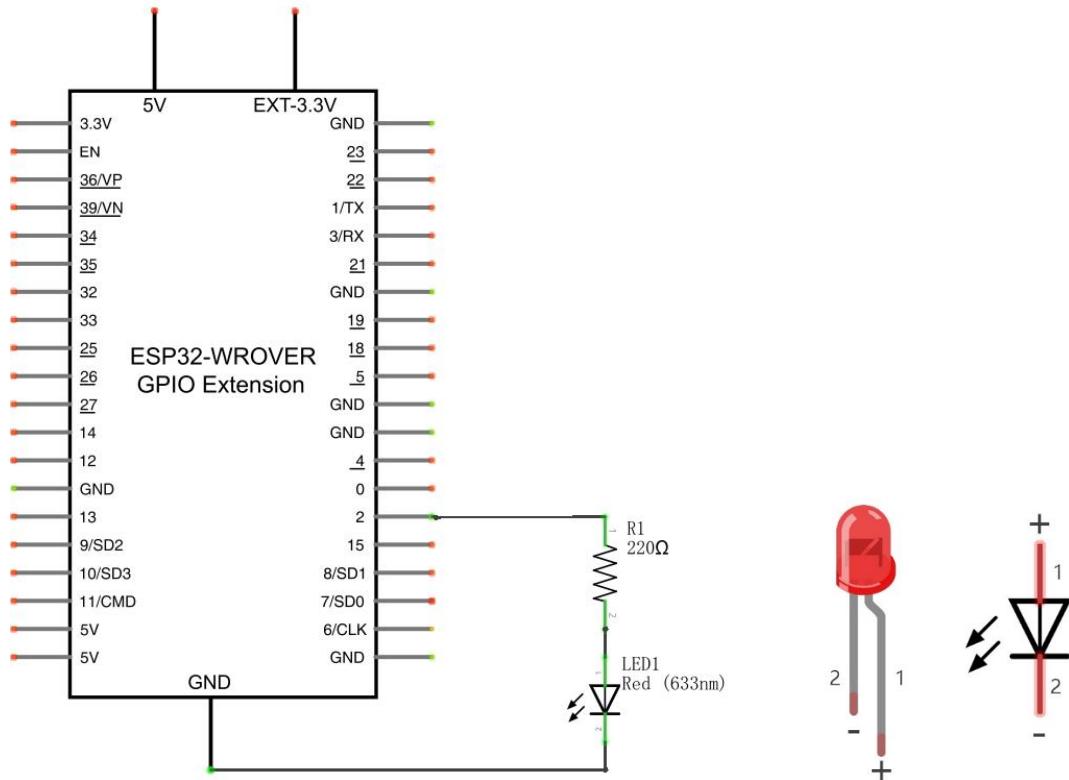
ESP32 and PWM

The ESP32 PWM controller has 8 independent channels, each of which can independently control frequency, duty cycle, and even accuracy. Unlike traditional PWM pins, the PWM output pins of ESP32 are configurable and they can be configured to PWM.

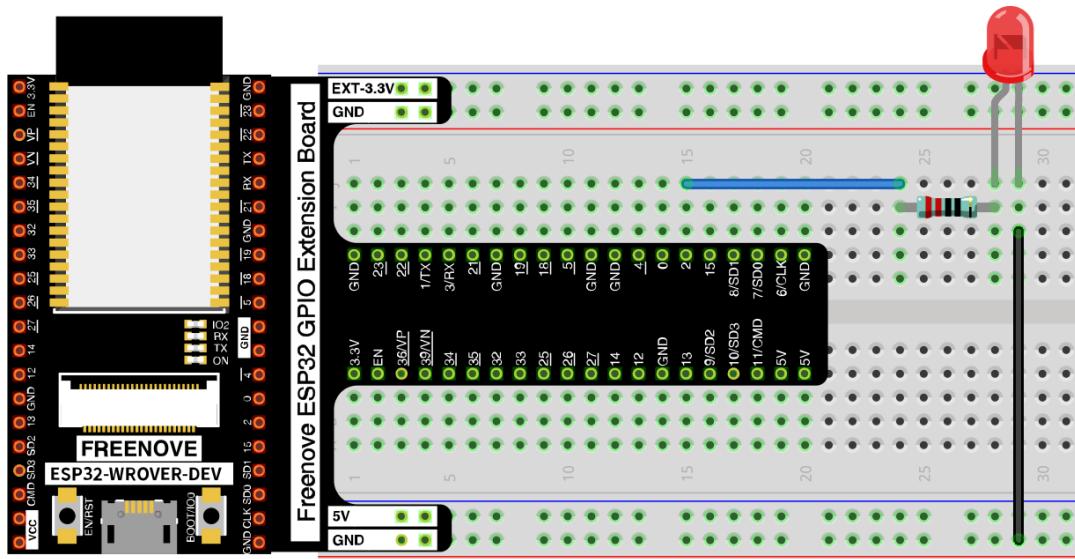
Circuit

This circuit is the same as the one in project Blink.

Schematic diagram



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

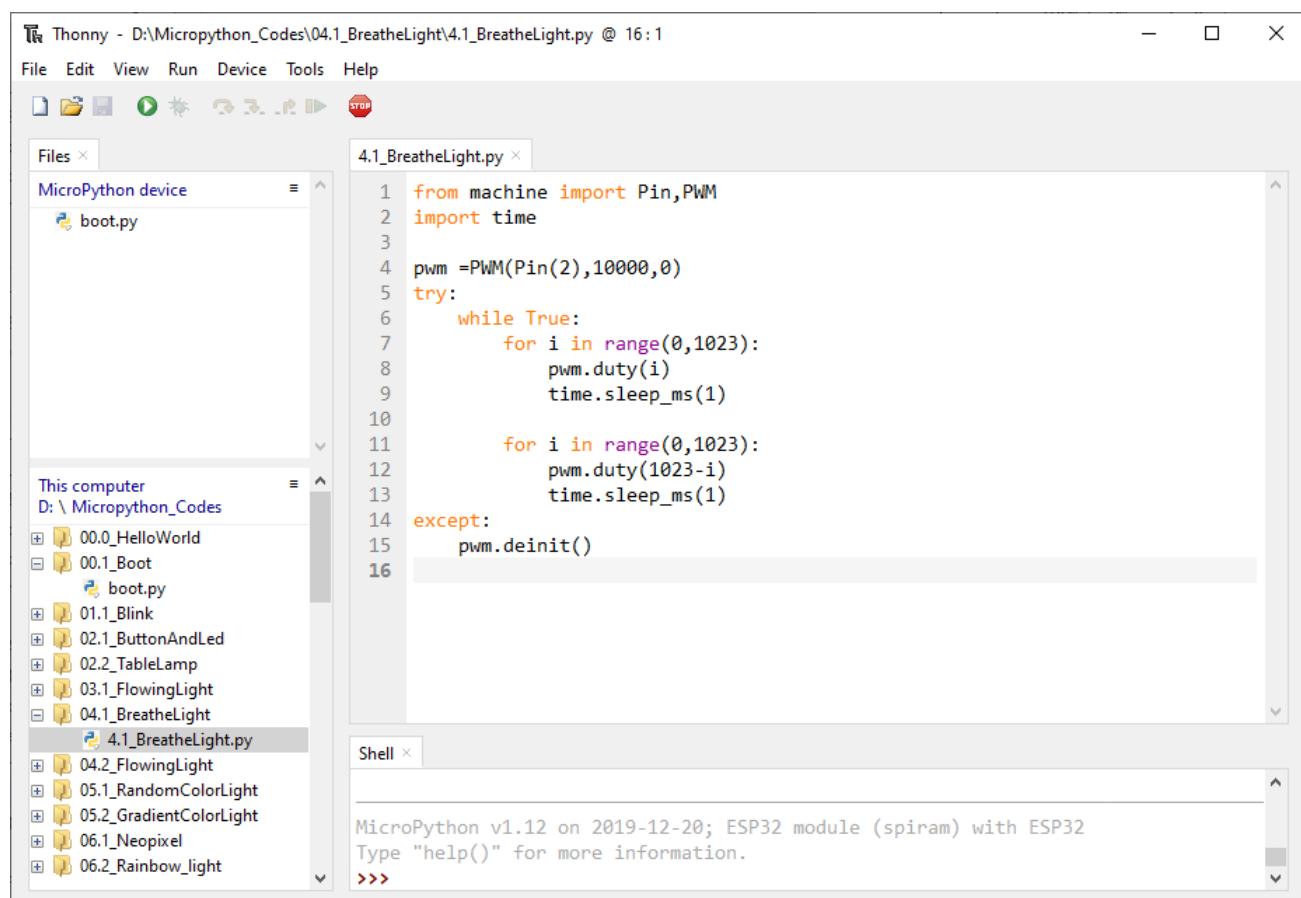


Code

This project is designed to make PWM output GPIO2 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “04.1_BreatheLight” and double click “04.1_BreatheLight.py”.

04.1 BreatheLight



```

from machine import Pin,PWM
import time

pwm =PWM(Pin(2),10000,0)
try:
    while True:
        for i in range(0,1023):
            pwm.duty(i)
            time.sleep_ms(1)

        for i in range(0,1023):
            pwm.duty(1023-i)
            time.sleep_ms(1)
except:
    pwm.deinit()

```

Click “Run current script”, and you'll see that LED is turned from ON to OFF and then back from OFF to ON gradually like breathing.





The following is the program code:

```

1  from machine import Pin, PWM
2  import time
3
4  pwm = PWM(Pin(2, Pin.OUT), 10000, 0)
5  try:
6      while True:
7          for i in range(0, 1023):
8              pwm.duty(i)
9              time.sleep_ms(1)
10
11         for i in range(0, 1023):
12             pwm.duty(1023-i)
13             time.sleep_ms(1)
14 except:
15     pwm.deinit()

```

The way that the ESP32 PWM pins output is different from traditionally controllers. It can change frequency and duty cycle by configuring PWM's parameters at the initialization stage. Define GPIO2's output frequency as 10000Hz and its duty cycle as 0, and assign them to PWM.

4	pwm =PWM(Pin(2, Pin.OUT), 10000, 0)
---	-------------------------------------

The range of duty cycle is 0-1023, so we use the first for loop to control PWM to change the duty cycle value, making PWM output 0% -100%; Use the second for loop to make PWM output 100%-0%.

7	for i in range(0, 1023):
8	pwm.duty(i)
9	time.sleep_ms(1)
10	
11	for i in range(0, 1023):
12	pwm.duty(1023-i)
13	time.sleep_ms(1)

Each time PWM is used, the hardware Timer will be turned ON to cooperate it. Therefore, after each use of PWM, deinit() needs to be called to turned OFF the timer. Otherwise, the PWM may fail to work next time.

15	pwm.deinit()
----	--------------

Reference

Class `PWM(pin, freq, duty)`

Before each use of PWM module, please add the statement “**from machine import PWM**” to the top of the python file.

pin: PWM pins are supported, such as Pin(0)、Pin(2)、Pin(4)、Pin(5)、Pin(10)、Pin(12~19)、Pin(21)、Pin(22)、Pin(23)、Pin(25~27).

freq: Output frequency, with the range of 0-78125 Hz

duty: Duty cycle, with the range of 0-1023.

PWM.init(freq, duty): Initialize PWM, parameters are the same as above.

PWM.freq([freq_val]): When there is no parameter, the function obtains and returns PWM frequency; When parameters are set, the function is used to set PWM frequency and returns nothing.

PWM.duty([duty_val]): When there is no parameter, the function obtains and returns PWM duty cycle; When parameters are set, the function is used to set PWM duty cycle.

PWM.deinit(): Turn OFF PWM.

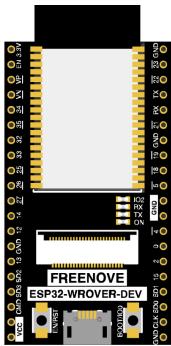


Project 4.2 Meteor Flowing Light

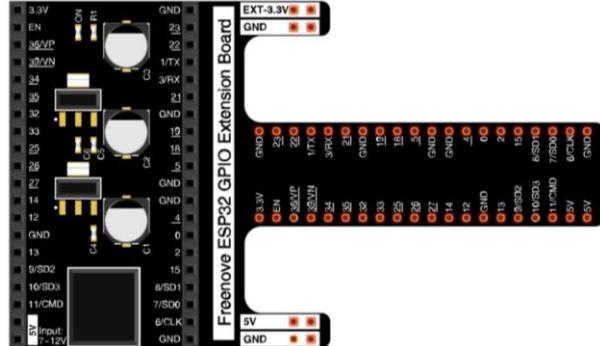
After learning about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

Component List

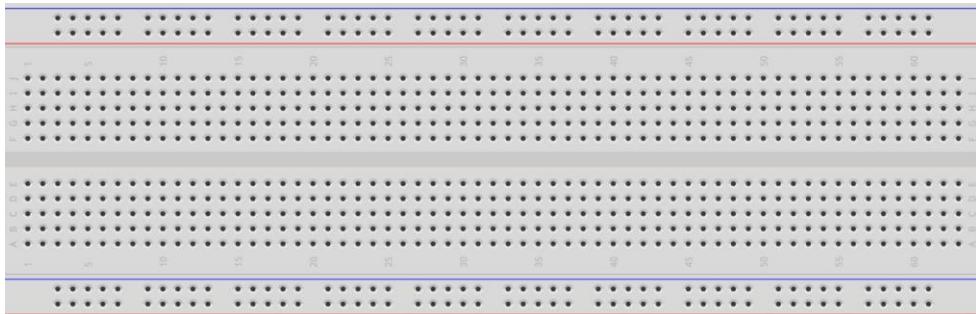
ESP32-WROVER x1



GPIO Extension Board x1



Breadboard x1



Jumper M/M x10



LED bar graph x1

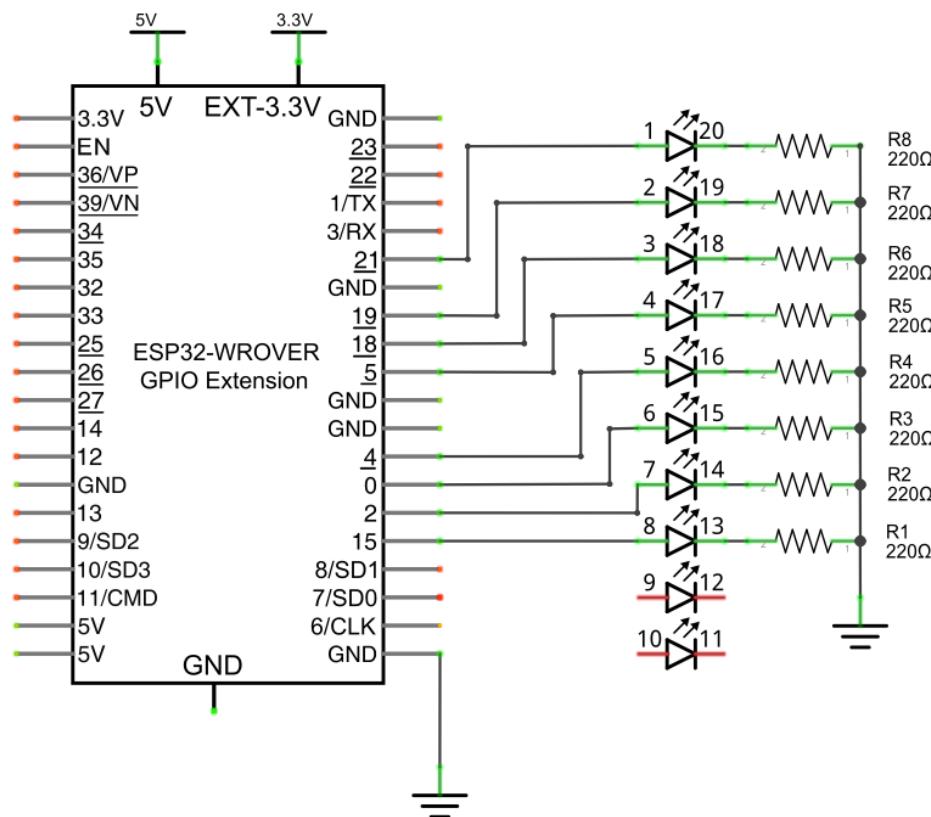


Resistor 220Ω x10

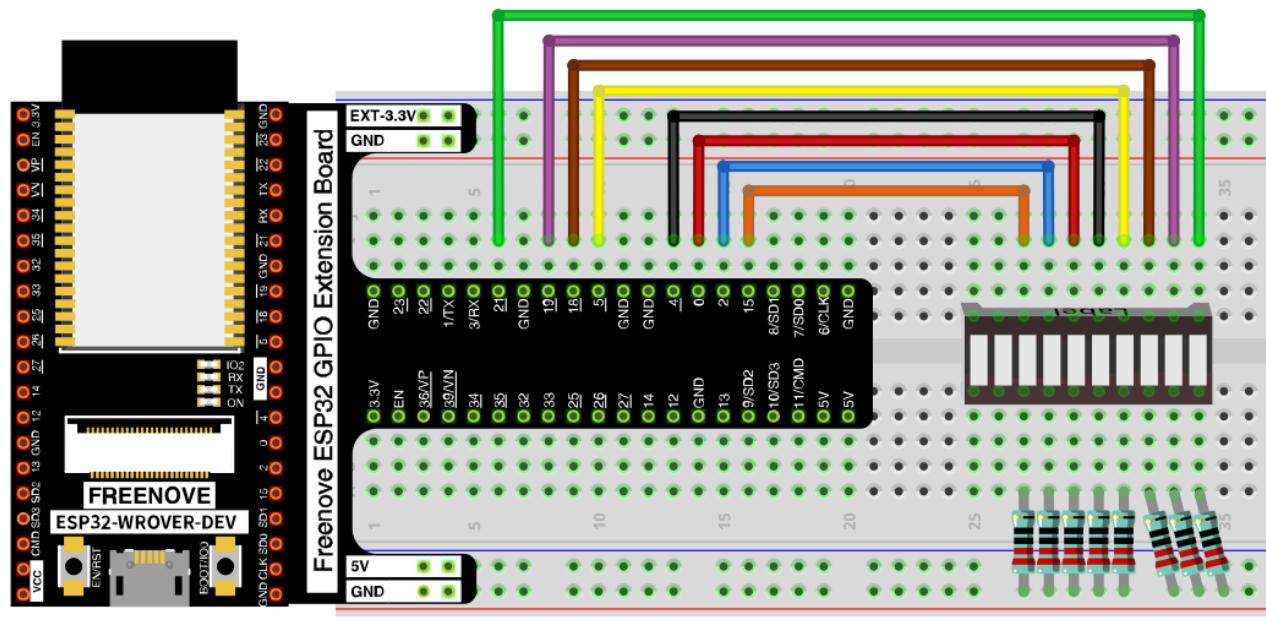


Circuit

Schematic diagram



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



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



Code

Flowing Light with tail was implemented with PWM.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “04.2_FlowingLight”. Select “pwm.py”, right click to select “Upload to /”, wait for “pwm.py” to be uploaded to ESP32-WROVER and then double click “04.2_FlowingLight.py”

04.2 FlowingLight

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\04.2_FlowingLight\04.2_FlowingLight.py @ 17:1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations. The left sidebar shows a tree view of the project structure under "MicroPython device": "boot.py", "This computer", "D:\Micropython_Codes\04.2_FlowingLight", "04.2_FlowingLight.py", and "pwm.py". A context menu is open over "pwm.py", with options: Refresh, Upload to / (highlighted in blue), Move to Recycle Bin, New directory..., Properties, and Storage space. The main editor window displays the Python code for "04.2_FlowingLight.py". The shell window at the bottom shows the MicroPython prompt: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32 Type "help()" for more information. >>>"

```

from machine import Pin,PWM
from pwm import myPWM
import time

mypwm = myPWM(15,2,0,4,5,18,19,21)
chns=[0,1,2,3,4,5,6,7];
dutys=[0,0,0,0,0,0,0,1023,512,256,128,64,32,16,8,0,0,0,0,0,0,0];
delayTimes=50

try:
    while True:
        for i in range(0,16):
            for j in range(0,8):
                mypwm.ledcWrite(chns[j],dutys[i+j])
                time.sleep_ms(delayTimes)

        for i in range(0,16):
            for j in range(0,8):
                mypwm.ledcWrite(chns[7-j],dutys[i+j])
                time.sleep_ms(delayTimes)
except:
    mypwm.deinit()

```

Click “Run current script”, and LED Bar Graph will gradually light up and out from left to right, then light up and out from right to left.

The following is the program code:

```
1  from machine import Pin, PWM
2  from pwm import myPWM
3  import time
4
5  mypwm = myPWM(15, 2, 0, 4, 5, 18, 19, 21)
6  chns=[0, 1, 2, 3, 4, 5, 6, 7];
7  dutys=[0, 0, 0, 0, 0, 0, 0, 1023, 512, 256, 128, 64, 32, 16, 8, 0, 0, 0, 0, 0, 0, 0, 0];
8  delayTimes=50
9
10 try:
11     while True:
12         for i in range(0, 16):
13             for j in range(0, 8):
14                 mypwm.ledcWrite(chns[j], dutys[i+j])
15                 time.sleep_ms(delayTimes)
16
17         for i in range(0, 16):
18             for j in range(0, 8):
19                 mypwm.ledcWrite(chns[7-j], dutys[i+j])
20                 time.sleep_ms(delayTimes)
21 except:
22     mypwm.deinit()
```

Import the object myPWM from pwm.py and set corresponding pins for PWM channel.

```
2  from pwm import myPWM
...
5  mypwm = myPWM(15, 2, 0, 4, 5, 18, 19, 21)
```

First we defined 8 GPIO, 8 PWM channels, and 24 pulse width values.

```
5  mypwm = myPWM(15, 2, 0, 4, 5, 18, 19, 21)
6  chns=[0, 1, 2, 3, 4, 5, 6, 7];
7  dutys=[0, 0, 0, 0, 0, 0, 0, 1023, 512, 256, 128, 64, 32, 16, 8, 0, 0, 0, 0, 0, 0, 0];
```

Call ledcWrite()to set duty cycle dutys[i+j] for the chns[j] channel of PWM.

```
14  mypwm.ledcWrite(chns[j], dutys[i+j])
```

Close the PWM of the object myPWM.

```
14  mypwm.deinit()
```

In the code, a nesting of two for loops are used to achieve this effect.

```

12     for i in range(0, 16):
13         for j in range(0, 8):
14             mypwm.ledcWrite(chns[j], dutys[i+j])
15             time.sleep_ms(delayTimes)
16
17     for i in range(0, 16):
18         for j in range(0, 8):
19             mypwm.ledcWrite(chns[7-j], dutys[i+j])
20             time.sleep_ms(delayTimes)

```

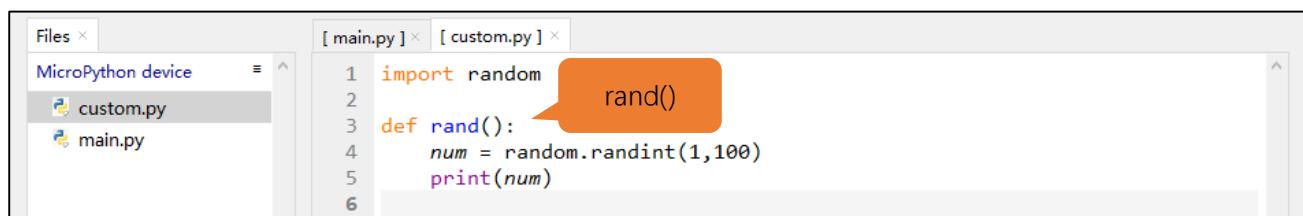
In the main function, a nested for loop is used to control the pulse width of the PWM. Every time *i* in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array *dutys*. As shown in the following table, the value in the second row is the value of the array *dutys*, and the 8 green grids in each row below represent the 8 LEDs on the LED Bar Graph. Each time *i* increases by 1, the value of the LED Bar Graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	2	2	2	2	2
d	0	0	0	0	0	0	0		1	5	2	1	6	3	1	8	0	0	0	0	0	0
i									0	1	5	2	4	2	6							
0																						
1																						
...																						
14																						
15																						
16																						

How to import a custom python module

Each Python file, as long as it's stored on the file system of ESP32, is a module. To import a custom module, the module file needs to be located in the MicroPython environment variable path or in the same path as the currently running program.

First, customize a python module "custom.py". Create a new py file and name it "custom.py". Write code to it and save it to ESP32.



Second, import custom module "custom" to main.py

The screenshot shows a code editor interface with two tabs: "main.py" and "custom.py". The "Files" sidebar on the left lists "MicroPython device", "custom.py", and "main.py". The "main.py" tab contains the following code:

```
1 import custom
2 import time
3 while True:
4     custom.rand()
5     time.sleep(1)
```

Two orange callout boxes point to specific parts of the code:

- An arrow points to the first line "import custom" with the text "Import custom module".
- An arrow points to the line "custom.rand()" with the text "Call function rand() of custom module".



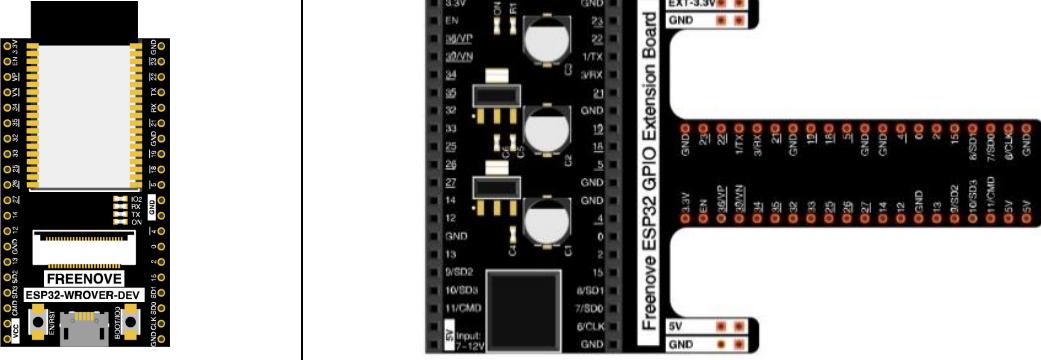
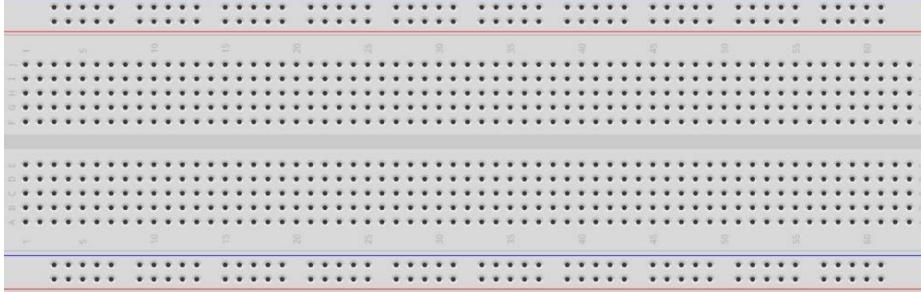
Chapter 5 RGBLED

In this chapter, we will learn how to control a RGBLED. It can emit different colors of light. Next, we will use RGBLED to make a multicolored light.

Project 5.1 Random Color Light

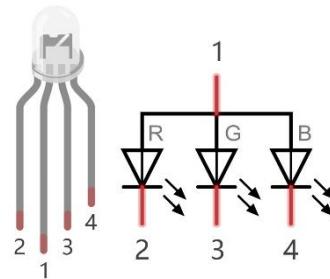
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

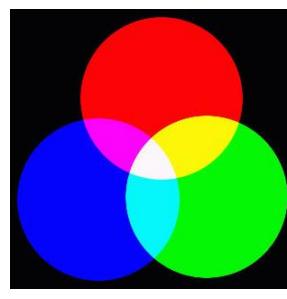
ESP32-WROVER x1	GPIO Extension Board x1	
		
Breadboard x1		
RGBLED x1	Resistor 220Ω x3	Jumper M/M x4
		

Related knowledge

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light. And it has 4 pins. The long pin (1) is the common port, that is, 3 LED's positive or negative port. The RGB LED with common positive port and its symbol is shown below. We can make RGB LED emit various colors of light by controlling these 3 LEDs to emit light with different brightness,



Red, green, and blue light are known as three primary colors. When you combine these three primary-color lights with different brightness, it can produce almost all kinds of visible lights. Computer screens, single pixel of cell phone screen, neon, and etc. are working under this principle.

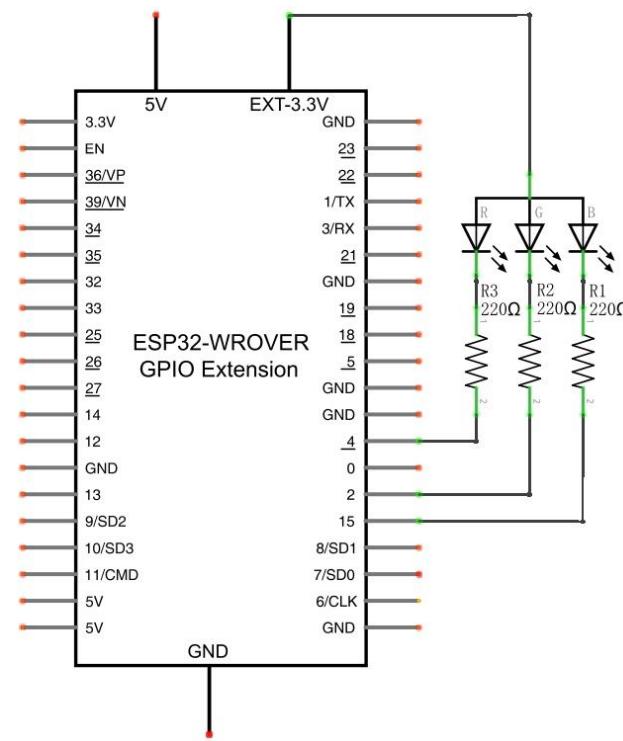


RGB

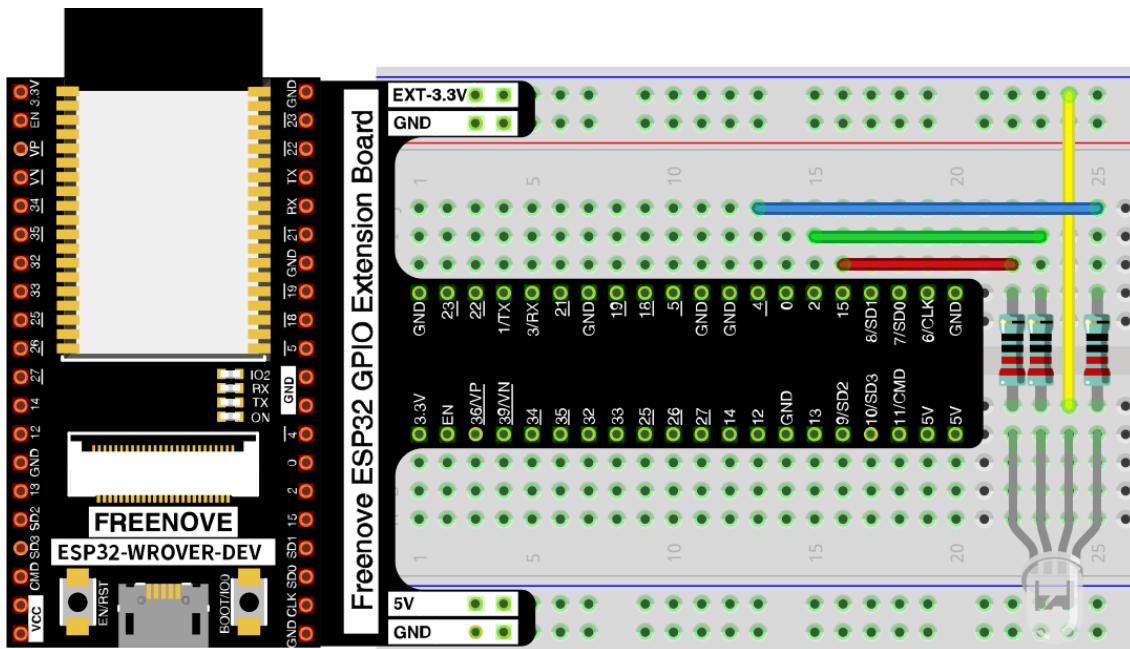
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10} * 2^{10} * 2^{10} = 1,073,741,824$ (1 billion) colors through different combinations.

Circuit

Schematic diagram



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



Code

We need to create three PWM channels and use random duty cycle to make random RGBLED color.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “05.1_RandomColorLight” and double click “05.1_RandomColorLight.py”.

05.1 RandomColorLight

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\05.1_RandomColorLight\05.1_RandomColorLight.py @ 26:18". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and a stop button. The left sidebar has a "Files" tab showing "MicroPython device" with "boot.py" and "This computer" with "D:\ Micropython_Codes \ 05.1_RandomColorLight" and "05.1_RandomColorLight.py" selected. The main area contains the code for "05.1_RandomColorLight.py":

```
1 from machine import Pin,PWM
2 from random import randint
3 import time
4
5 pins=[15,2,0]
6
7 pwm0=PWM(Pin(pins[0]),10000)
8 pwm1=PWM(Pin(pins[1]),10000)
9 pwm2=PWM(Pin(pins[2]),10000)
10
11 def setColor(r,g,b):
12     pwm0.duty(1023-r)
13     pwm1.duty(1023-g)
14     pwm2.duty(1023-b)
15
16 try:
17     while True:
18         red = randint(0,1023)
19         green = randint(0,1023)
20         blue = randint(0,1023)
21         setColor(red,green,blue)
22         time.sleep_ms(200)
23         current.
```

The bottom right pane is a "Shell" window displaying the MicroPython prompt:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script”, RGBLED begins to display random colors.

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



The following is the program code:

```

1  from machine import Pin, PWM
2  from random import randint
3  import time
4
5  pins=[15, 2, 0]
6
7  pwm0=PWM(Pin(pins[0]), 10000)
8  pwm1=PWM(Pin(pins[1]), 10000)
9  pwm2=PWM(Pin(pins[2]), 10000)
10
11 def setColor(r, g, b):
12     pwm0.duty(1023-r)
13     pwm1.duty(1023-g)
14     pwm2.duty(1023-b)
15
16 try:
17     while True:
18         red    = randint(0, 1023)
19         green  = randint(0, 1023)
20         blue   = randint(0, 1023)
21         setColor(red, green, blue)
22         time.sleep_ms(200)
23 except:
24     pwm0.deinit()
25     pwm1.deinit()
26     pwm2.deinit()
```

Import Pin, PWM and Randon Function modules.

```

12  from machine import Pin, PWM
13  from random import randint
14  import time
```

Configure ouput mode of GPIO15, GPIO2 and GPIO0 as PWM output and PWM frequency as 10000Hz

```

5  pins=[15, 2, 0]
6
7  pwm0=PWM(Pin(pins[0]), 10000)
8  pwm1=PWM(Pin(pins[1]), 10000)
9  pwm2=PWM(Pin(pins[2]), 10000)
```

Define a function to set the color of RGBLED.

```

11 def setColor(r, g, b):
12     pwm0.duty(1023-r)
13     pwm1.duty(1023-g)
14     pwm2.duty(1023-b)
```

Call random function randint()to generate a random number in the range of 0-1023 and assign the value to red.

```
18    red = randint(0, 1023)
```

Obtain 3 random number every 200 milliseconds and call function setColor to make RGBLED display dazzling colors.

```
17    while True:  
18        red = randint(0, 1023)  
19        green = randint(0, 1023)  
20        blue = randint(0, 1023)  
21        setColor(red, green, blue)  
22        time.sleep_ms(200)
```

Reference

Class random

Before each use of the module **random**, please add the statement “**import random**” to the top of Python file.

randint(start, end): Randomly generates an integer between the value of start and end.

start: Starting value in the specified range, which would be included in the range.

end: Ending value in the specified range, which would be included in the range.

random(): Randomly generates a floating point number between 0 and 1.

random.uniform(start, end): Randomly generates a floating point number between the value of start and end

start: Starting value in the specified range, which would be included in the range.

end: Ending value in the specified range, which would be included in the range.

random.getrandbits(size): Generates an integer with **size** random bits

For example:

size = 4, it generates an integer in the range of 0 to 0b1111

size = 8, it generates an integer in the range of 0 to 0b11111111

random.randrange(start, end, step): Randomly generates a positive integer in the range from start to end and increment to step.

start: Starting value in the specified range, which would be included in the range

end: Ending value in the specified range, which would be included in the range.

step: An integer specifying the incrementation.

random.seed(sed): Specifies a random seed, usually being applied in conjunction with other random number generators

sed: Random seed, a starting point in generating random numbers.

random.choice(obj): Randomly generates an element from the object obj.

obj: list of elements

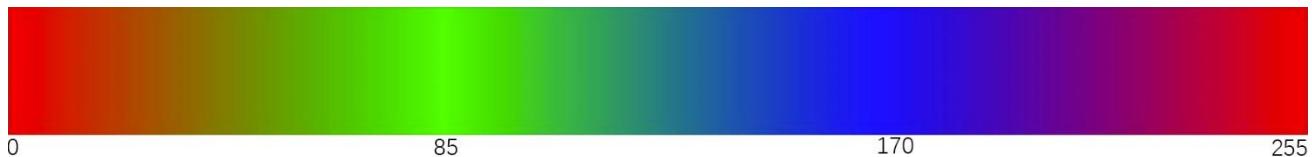


Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff. This project will realize a fashionable Light with soft color changes.

Component list, the circuit is exactly the same as the project random color light.

Using a color model, the color changes from 0 to 255 as shown below.



In this code, the color model will be implemented and RGBLED will change colors along the model.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “5.2_GradientColorLight” and double click “5.2_GradientColorLight.py”.

05.2 GradientColorLight

The following is the program code:

```

1  from machine import Pin, PWM
2  import time
3
4  pins=[15, 2, 0];
5
6  pwm0=PWM(Pin(pins[0]), 1000)
7  pwm1=PWM(Pin(pins[1]), 1000)
8  pwm2=PWM(Pin(pins[2]), 1000)
9
10 def setColor(rgb):
11     pwm0.duty(1023-(rgb>>20))
12     pwm1.duty(1023-(rgb>>10))
13     pwm2.duty(1023-(rgb>>0))
14
15 def wheel(pos):
16     WheelPos=pos%1023
17     if WheelPos<341:
18         return (((1023-WheelPos*3)<<20) | ((WheelPos*3)<<10))
19     elif WheelPos>=341 and WheelPos<682:
20         WheelPos -= 341;
21         return (((1023-WheelPos*3)<<10) | (WheelPos*3))
22     else :
23         WheelPos -= 682;
24         return (((WheelPos*3)<<20) | (1023-WheelPos*3))
25
26 try:

```

```
27     while True:  
28         for i in range(0, 1023):  
29             setColor(wheel( i ))  
30             time.sleep_ms(5)  
31     except:  
32         pwm0.deinit()  
33         pwm1.deinit()  
34         pwm2.deinit()
```

In the function **setColor()**, we use a variable to represent the value of RGB, making it more convenient for the passing of parameters. As the range of PWM's duty cycle is 0-1023, which is 2 to the tenth power, when split, the value of each color channel can be obtained with a simple bitwise operation.

```
10 def setColor(rgb):  
11     pwm0.duty(1023-(rgb>>20))  
12     pwm1.duty(1023-(rgb>>10))  
13     pwm2.duty(1023-(rgb>>0))
```

The function **wheel()** is a color selection method of the color model introduced earlier. The value range of the parameter pos is 0-1023. The function will return a data containing the duty cycle values of 3 pins.

```
15 def wheel(pos):  
16     WheelPos=pos%1023  
17     if WheelPos<341:  
18         return (((1023-WheelPos*3)<<20) | ((WheelPos*3)<<10))  
19     elif WheelPos>=341 and WheelPos<682:  
20         WheelPos -= 341;  
21         return (((1023-WheelPos*3)<<10) | (WheelPos*3))  
22     else :  
23         WheelPos -= 682;  
24         return (((WheelPos*3)<<20) | (1023-WheelPos*3))
```



Chapter 6 NeoPixel

This chapter will help you learn to use a more convenient RGBLED lamp, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 6.1 NeoPixel

Learn the basic usage of NeoPixel and use it to flash red, green, blue and white.

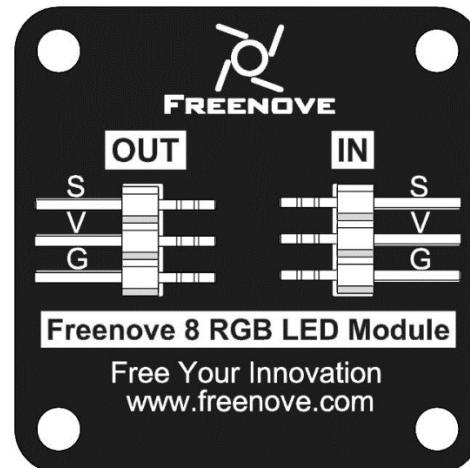
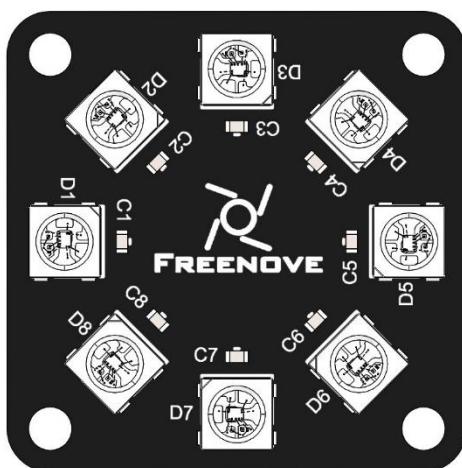
Component List

ESP32-WROVER x1	GPIO Extension Board x1
Breadboard x1	
Freenove 8 RGB LED Module x1	Jumper F/M x3

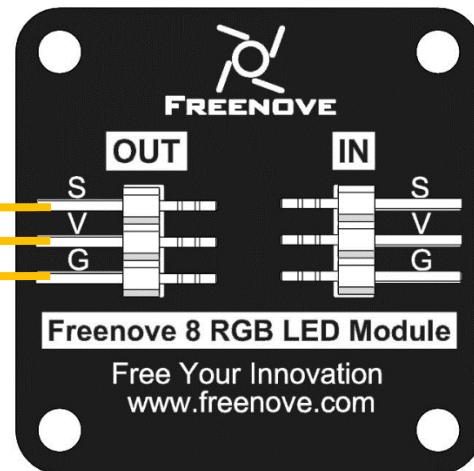
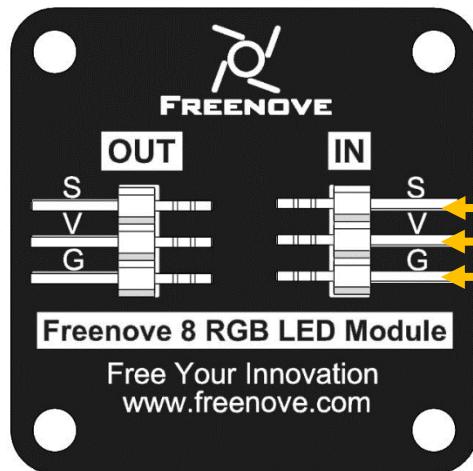
Related knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control 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 this way, you can use one data pin to control 8, 16, 32 ... LEDs.



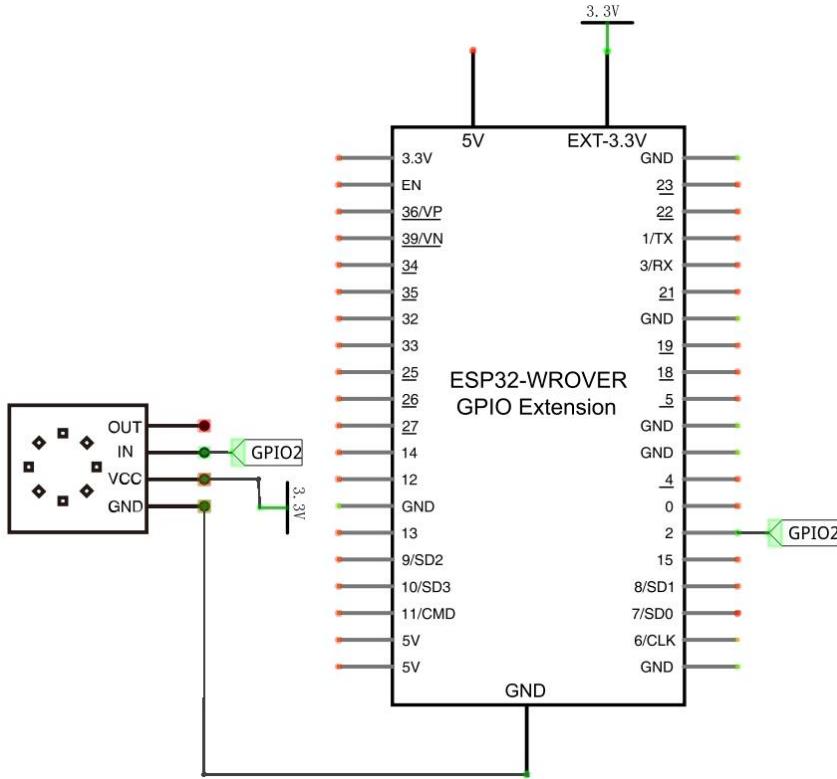
Pin description:

(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

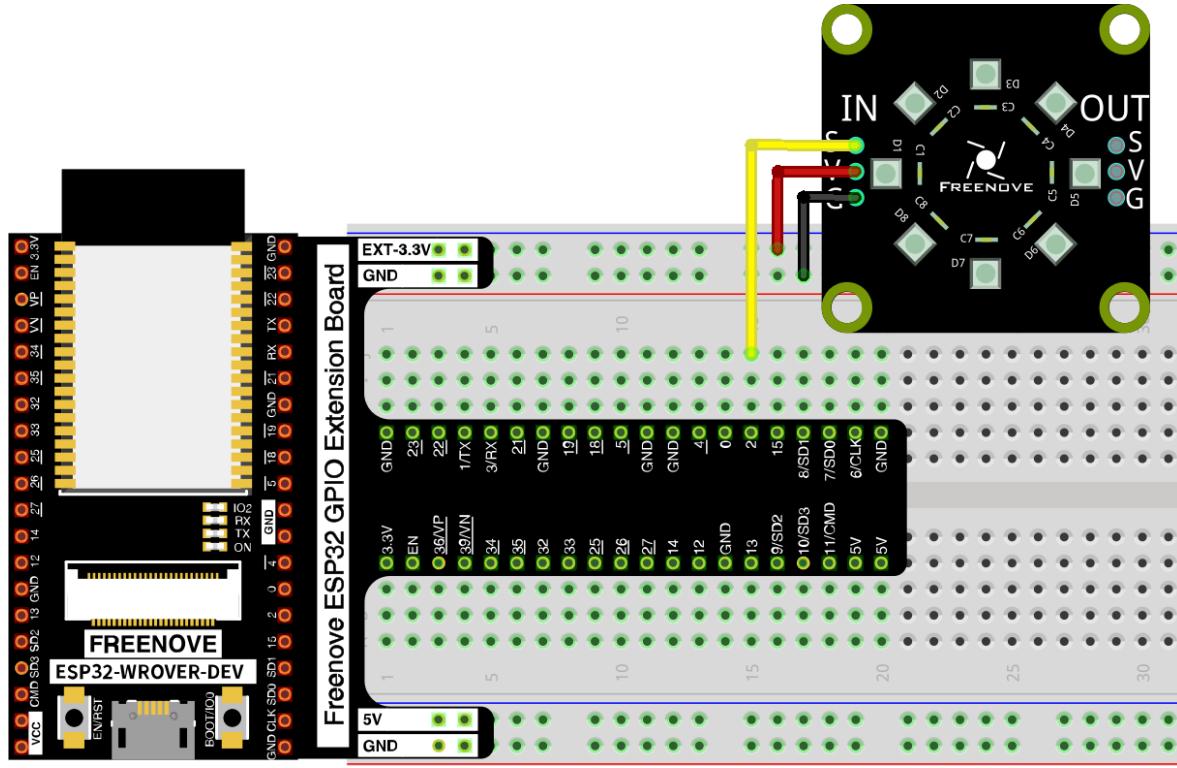


Circuit

Schematic diagram



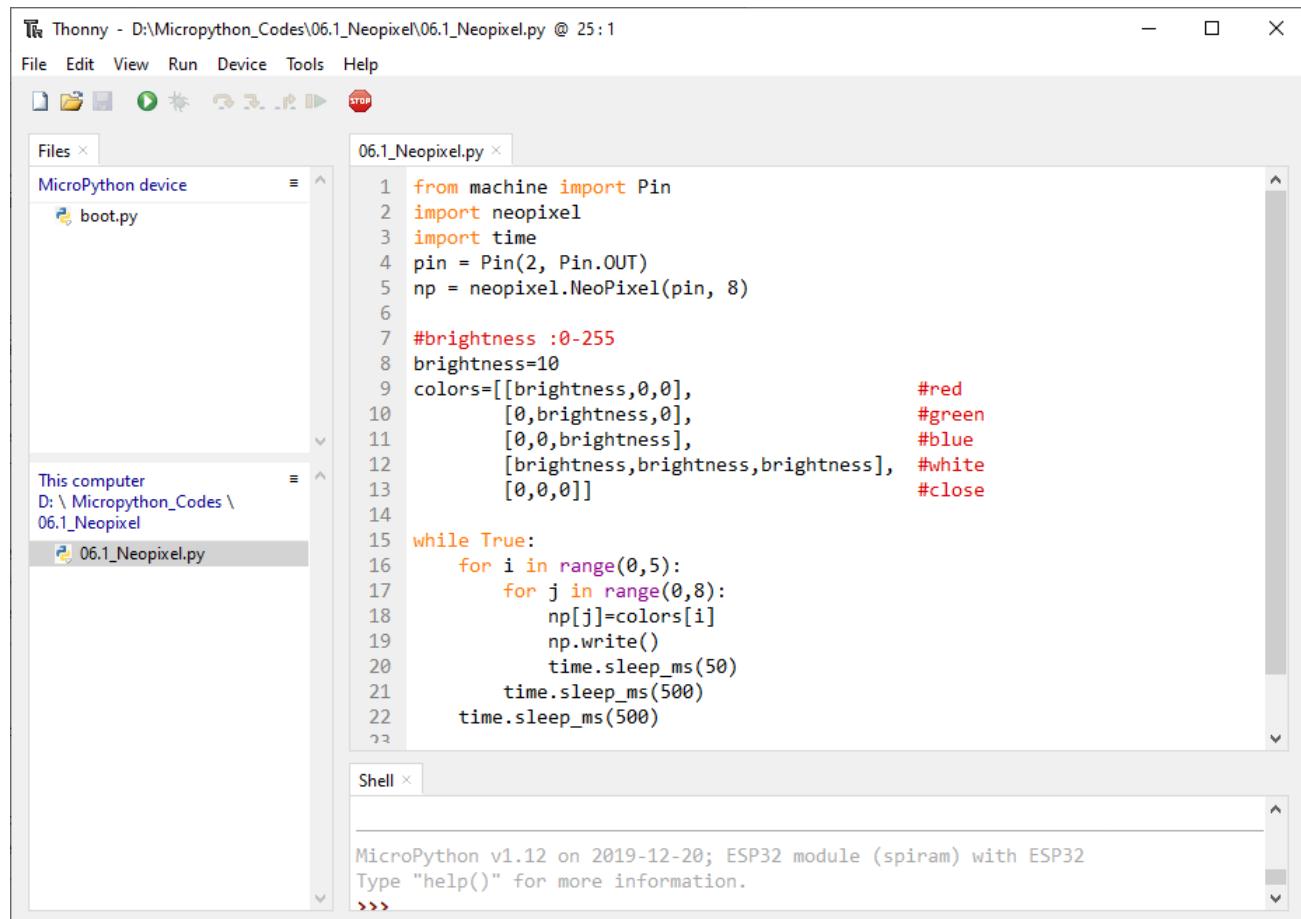
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “06.1_Neopixel” and double click “06.1_Neopixel.py”.

06.1 Neopixel



```

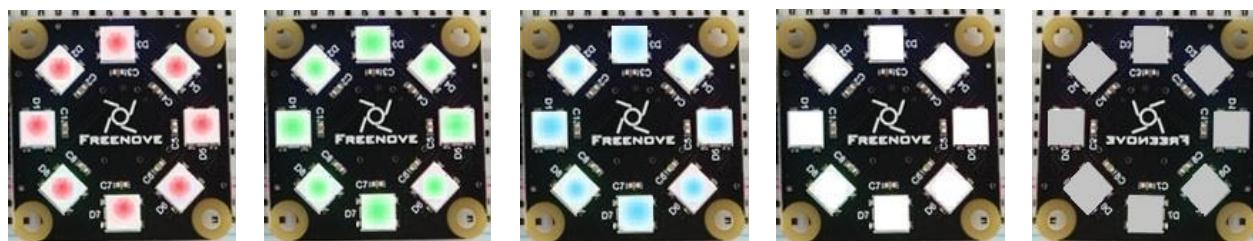
from machine import Pin
import neopixel
import time
pin = Pin(2, Pin.OUT)
np = neopixel.NeoPixel(pin, 8)

#brightness :0-255
brightness=10
colors=[[brightness,0,0], #red
        [0,brightness,0], #green
        [0,0,brightness], #blue
        [brightness,brightness,brightness], #white
        [0,0,0]] #close

while True:
    for i in range(0,5):
        for j in range(0,8):
            np[j]=colors[i]
            np.write()
            time.sleep_ms(50)
    time.sleep_ms(500)
    time.sleep_ms(500)

```

Click “Run current script”, and Neopixel begins to light up in red, green, blue, white and black.





The following is the program code:

```

1  from machine import Pin
2  import neopixel
3  import time
4  pin = Pin(2, Pin.OUT)
5  np = neopixel.NeoPixel(pin, 8)

6
7  #brightness :0~255
8  brightness=10
9  colors=[[brightness, 0, 0],           #red
10    [0, brightness, 0],               #green
11    [0, 0, brightness],             #blue
12    [brightness, brightness, brightness], #white
13    [0, 0, 0]]                      #close

14
15 while True:
16     for i in range(0, 5):
17         for j in range(0, 8):
18             np[j]=colors[i]
19             np.write()
20             time.sleep_ms(50)
21             time.sleep_ms(500)
22             time.sleep_ms(500)

```

Import Pin, neopixel and time modules.

```

1  from machine import Pin
2  import neopixel
3  import time

```

Define the number of pins and LEDs connected to neopixel.

```

4  pin = Pin(2, Pin.OUT)
5  np = neopixel.NeoPixel(pin, 8)

```

Define the brightness of neopixel's LED and an array to store color.

```

7  #brightness :0~255
8  brightness=10
9  colors=[[brightness, 0, 0],           #red
10    [0, brightness, 0],               #green
11    [0, 0, brightness],             #blue
12    [brightness, brightness, brightness], #white
13    [0, 0, 0]]                      #close

```

Assign the color data to the array np and call function write() to send np array data to neopixel module.

```

18          np[j]=colors[i]
19          np.write()

```

Nest two for loops to make the module repeatedly display five states of red, green, blue, white and OFF.

```
15 while True:  
16     for i in range(0, 5):  
17         for j in range(0, 8):  
18             np[j]=colors[i]  
19             np.write()  
20             time.sleep_ms(50)  
21             time.sleep_ms(500)  
22             time.sleep_ms(500)
```

Reference

Class neopixel

Before each usr of **neopixel** module, please add the statement “**import neopixel**” to the top of Python file.

NeoPixel(pin, n): Define the number of output pins and LEDs of neopixel module

pin: Output pins

n: The number of LEDs.

NeoPixel.write(): Write data to LEDs.

Project 6.2 Rainbow Light

In the previous project, we have mastered the usage of NeoPixel. This project will realize a slightly complicated Rainbow Light. The component list and the circuit are exactly the same as the project fashionable Light.

Code

Continue to use the following color model to equalize the color distribution of the 8 leds and gradually change.



Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “06.2_Rainbow_light” and then double click “06.2_Rainbow_light.py”.

06.2 Rainbow light

Thonny - D:\Micropython_Codes\06.2_Rainbow_light\06.2_Rainbow_light.py @ 38:1

File Edit View Run Device Tools Help

Files x

MicroPython device

boot.py

This computer

D:\ Micropython_Codes\06.2_Rainbow_light

06.2_Rainbow_light.py

06.2_Rainbow_light.py x

```
1 from machine import Pin
2 import neopixel
3 import time
4 pin = Pin(2, Pin.OUT)
5 np = neopixel.NeoPixel(pin, 8)
6
7 brightness=0.1          #brightness: 0-1.0
8 red=0                  #red
9 green=0                #green
10 blue=0                #blue
11
12 def wheel(pos):
13     global red,green,blue
14     WheelPos=pos%255
15     if WheelPos<85:
16         red=(255-WheelPos*3)
17         green=(WheelPos*3)
18         blue=0
19     elif WheelPos>=85 and WheelPos<170:
20         WheelPos -= 85;
21         red=0
22         green=(255-WheelPos*3)
23         blue=(WheelPos*3)
```

Shell x

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script”, and the Freenove 8 RGB LED Strip displays different colors and the color changes gradually.



The following is the program code:

```
1  from machine import Pin
2  import neopixel
3  import time
4  pin = Pin(2, Pin.OUT)
5  np = neopixel.NeoPixel(pin, 8)
6
7  brightness=0.1      #brightness: 0 ~ 1.0
8  red=0               #red
9  green=0              #green
10 blue=0               #blue
11
12 def wheel(pos):
13     global red,green,blue
14     WheelPos=pos%255
15     if WheelPos<85:
16         red=(255-WheelPos*3)
17         green=(WheelPos*3)
18         blue=0
19     elif WheelPos>=85 and WheelPos<170:
20         WheelPos -= 85;
21         red=0
22         green=(255-WheelPos*3)
23         blue=(WheelPos*3)
24     else :
25         WheelPos -= 170;
26         red=(WheelPos*3)
27         green=0
28         blue=(255-WheelPos*3)
29
30 while True:
31     for i in range(0,255):
32         for j in range(0,8):
```

```

33     wheel(i+j*255//8)
34     np[j]=(int(red*brightness), int(green*brightness), int(blue*brightness))
35     np.write()
36     time.sleep_ms(5)

```

Define a wheel() function to process the color data of neopixel module.

```

12 def wheel(pos):
13     global red, green, blue
14     WheelPos=pos%255
15     if WheelPos<85:
16         red=(255-WheelPos*3)
17         green=(WheelPos*3)
18         blue=0
19     elif WheelPos>=85 and WheelPos<170:
20         WheelPos -= 85;
21         red=0
22         green=(255-WheelPos*3)
23         blue=(WheelPos*3)
24     else :
25         WheelPos -= 170;
26         red=(WheelPos*3)
27         green=0
28         blue=(255-WheelPos*3)

```

Set the color brightness of the module.

7	brightness=0.1 #brightness: 0 ~ 1.0
---	--

Use a nesting of two for loops. The first for loop makes the value of i increase from 0 to 255 automatically and the wheel() function processes the value of i into data of the module's three colors; the second for loop writes the color data to the module.

31	for i in range(0, 255):
32	for j in range(0, 8):
33	wheel(i+j*255//8)
34	np[j]=(int(red*brightness), int(green*brightness), int(blue*brightness))
35	np.write()
36	time.sleep_ms(5)

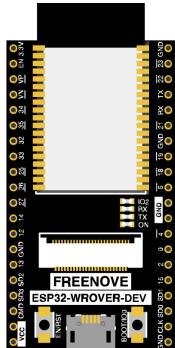
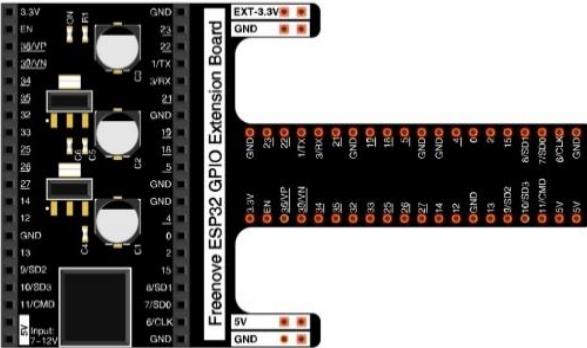
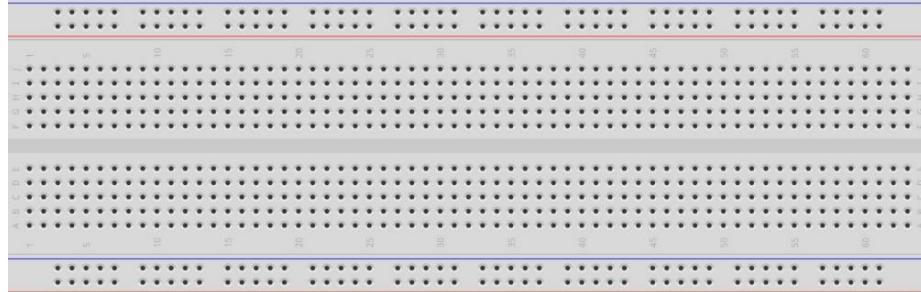
Chapter 7 Buzzer

In this chapter, we will learn about buzzers and the sounds they make.

Project 7.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

Component List

ESP32-WROVER x1	GPIO Extension Board x1
 	
Breadboard x1	
	
Jumper M/M x6	
NPN transistor x1 (S8050)	
Active buzzer x1	
Push button x1	
Resistor 1kΩ x1	
Resistor 10kΩ x2	

Component knowledge

Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock and alarm. Buzzer has two types: active and passive. Active buzzer has oscillator inside, which will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.



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

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

How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
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 viewing 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).

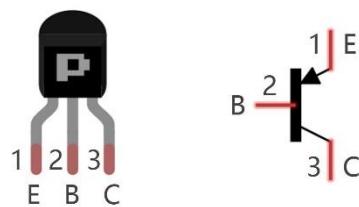


Transistor

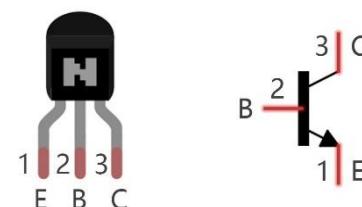
Because the buzzer requires such large current that GPIO of ESP32 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types as shown below: PNP and NPN,

PNP transistor



NPN transistor

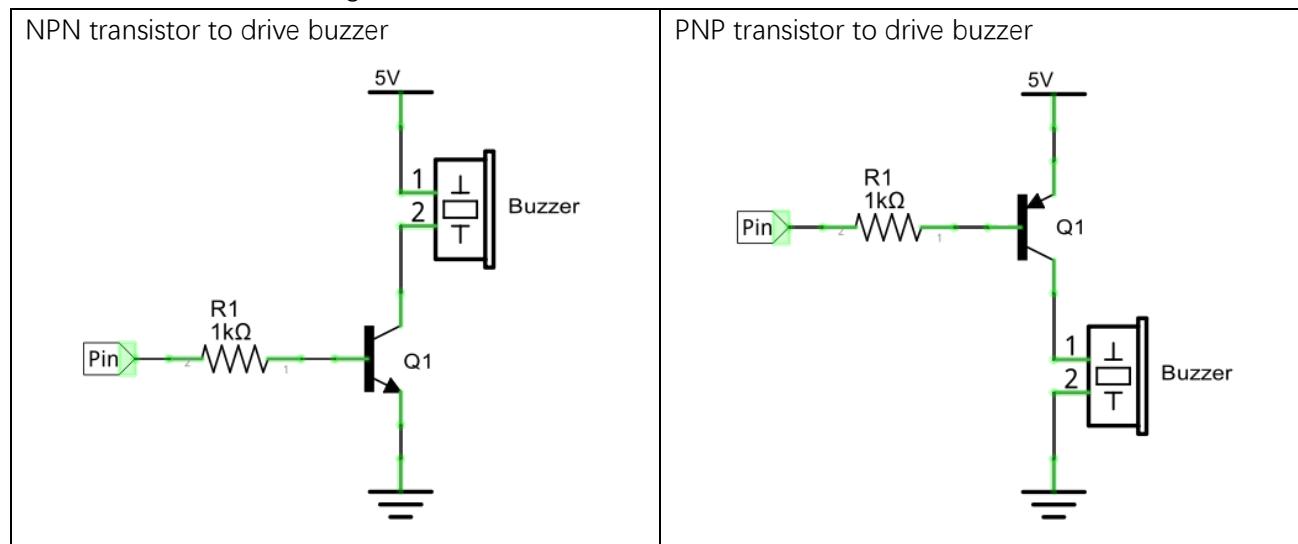


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

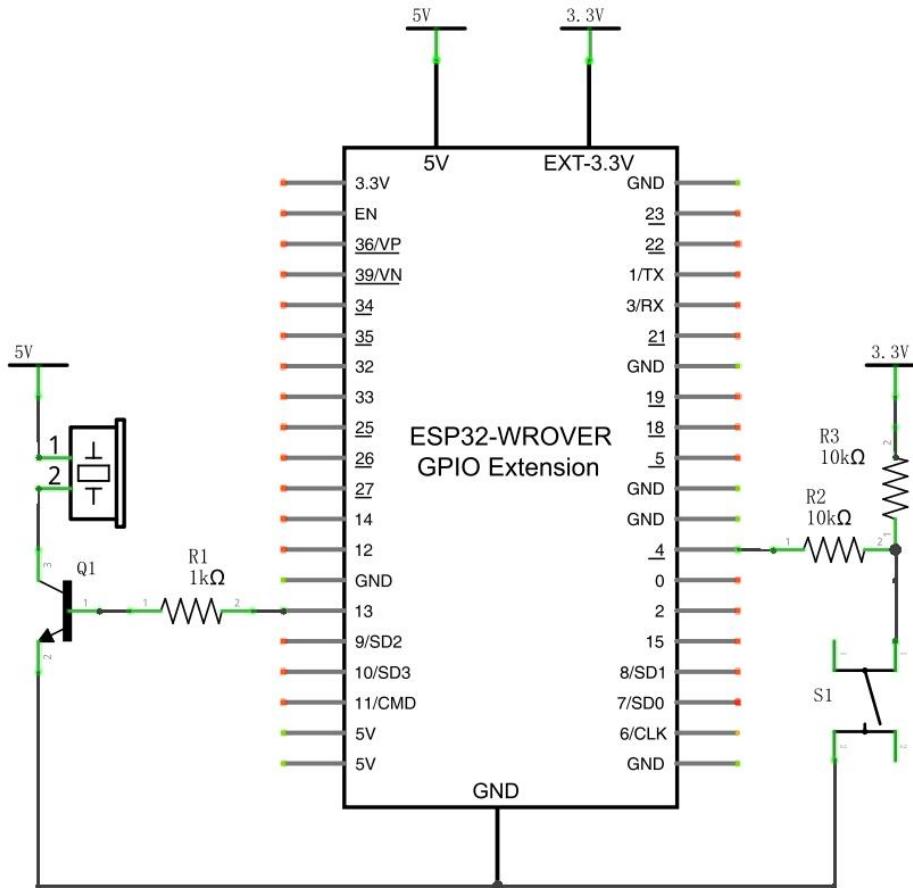
When using NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When using PNP transistor to drive buzzer, we often adopt the following method. If GPIO outputs low level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

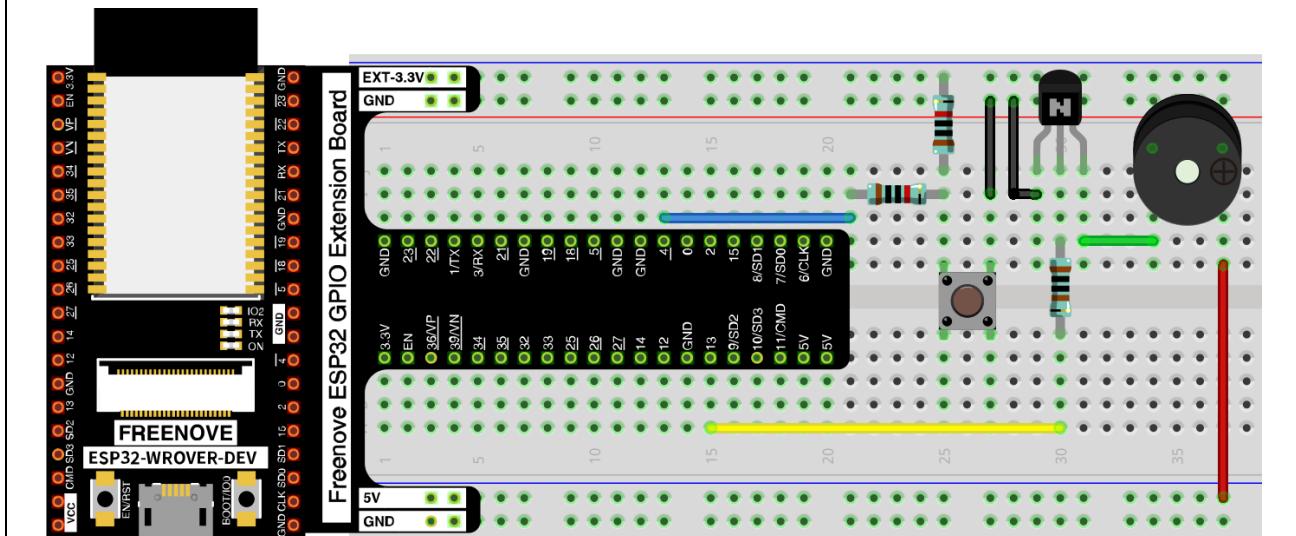


Circuit

Schematic diagram



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



Note: in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.

Code

In this 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.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.1_Doorbell” and double click “07.1_Doorbell.py”.

07.1 Doorbell

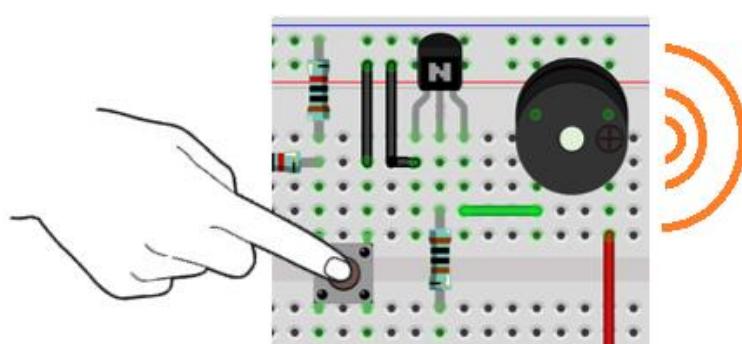
The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\07.1_Doorbell\07.1_Doorbell.py @ 13:1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and a stop button. The left sidebar has a "Files" tab showing "MicroPython device" with "boot.py" and "This computer" showing "D:\Micropython_Codes\07.1_Doorbell" with "07.1_Doorbell.py" selected. The main code editor window displays the following Python code:

```
1 from machine import Pin
2
3 button=Pin(4,Pin.IN,Pin.PULL_UP)
4 activeBuzzer=Pin(13,Pin.OUT)
5
6 activeBuzzer.value(0)
7
8 while True:
9     if not button.value():
10         activeBuzzer.value(1)
11     else:
12         activeBuzzer.value(0)
```

The bottom right window is a "Shell" showing the MicroPython environment:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script”, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.





The following is the program code:

```
1  from machine import Pin  
2  
3  button=Pin(4,Pin.IN,Pin.PULL_UP)  
4  activeBuzzer=Pin(13,Pin.OUT)  
5  
6  activeBuzzer.value(0)  
7  
8  while True:  
9      if not button.value():  
10         activeBuzzer.value(1)  
11     else:  
12         activeBuzzer.value(0)
```

The code is logically the same as using button to control LED.

Project 7.2 Alertor

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

Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer**.

Code

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. In the logic, it is the same as using button to control LED. In the control method, passive buzzer requires PWM of certain frequency to sound.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.2_Alertor”, and double click “07.2_Alertor.py”.

07.2 Alertor

```

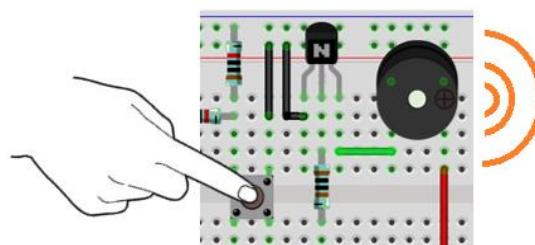
from machine import Pin,PWM
import math
import time
PI=3.14
button=Pin(4,Pin.IN,Pin.PULL_UP)
passiveBuzzer=PWM(Pin(13),2000,512)

def alert():
    for x in range(0,36):
        sinVal=math.sin(x*10*PI/180)
        toneVal=2000+int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)

try:
    while True:
        if not button.value():
            alert()
        else:
            passiveBuzzer.freq(0)
except:
    passiveBuzzer.deinit()

```

Click “Run current script”, press the button, then alarm sounds. And when the button is release, the alarm will stop sounding.





The following is the program code:

```

1  from machine import Pin, PWM
2  import math
3  import time
4
5  PI=3.14
6  button=Pin(4, Pin.IN, Pin.PULL_UP)
7  passiveBuzzer=PWM(Pin(13), 2000, 512)
8
9  def alert():
10     for x in range(0, 36):
11         sinVal=math.sin(x*10*PI/180)
12         toneVal=2000+int(sinVal*500)
13         passiveBuzzer.freq(toneVal)
14         time.sleep_ms(10)
15     try:
16         while True:
17             if not button.value():
18                 alert()
19             else:
20                 passiveBuzzer.freq(0)
21     except:
22         passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

```

1  from machine import Pin, PWM
2  import math
3  import time
```

Define the pins of the button and passive buzzer.

```

5  PI=3.14
6  button=Pin(4, Pin.IN, Pin.PULL_UP)
7  passiveBuzzer=PWM(Pin(13), 2000, 512)
```

Call sin function of math module to generate the frequency data of the passive buzzer.

```

9  def alert():
10     for x in range(0, 36):
11         sinVal=math.sin(x*10*PI/180)
12         toneVal=2000+int(sinVal*500)
13         passiveBuzzer.freq(toneVal)
14         time.sleep_ms(10)
```

When not using PWM, please turn it OFF in time.

```
22  passiveBuzzer.deinit()
```

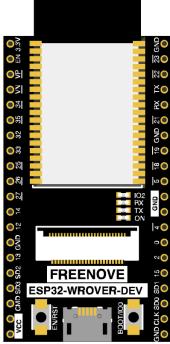
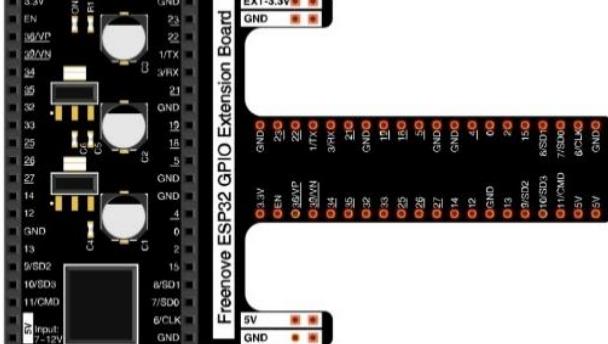
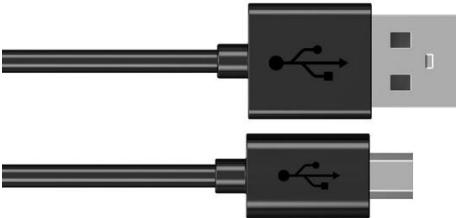
Chapter 8 Serial Communication

Serial Communication is a means of Communication between different devices/devices. This section describes ESP32's Serial Communication.

Project 8.1 Serial Print

This project uses ESP32's serial communicator to send data to the computer and print it on the serial monitor.

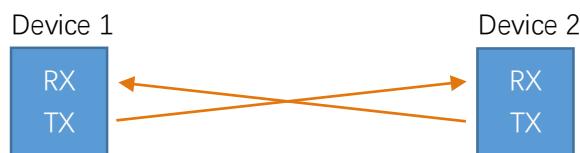
Component List

ESP32-WROVER x1	GPIO Extension Board x1	Micro USB Wire x1
		

Related knowledge

Serial communication

Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections between two devices use is as follows:



Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used are 9600 and 115200.

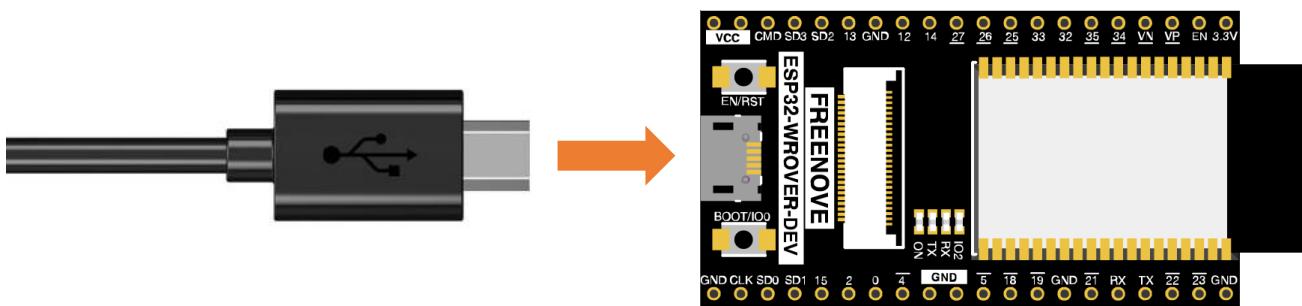
Serial port on ESP32

Freenove ESP32 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Circuit

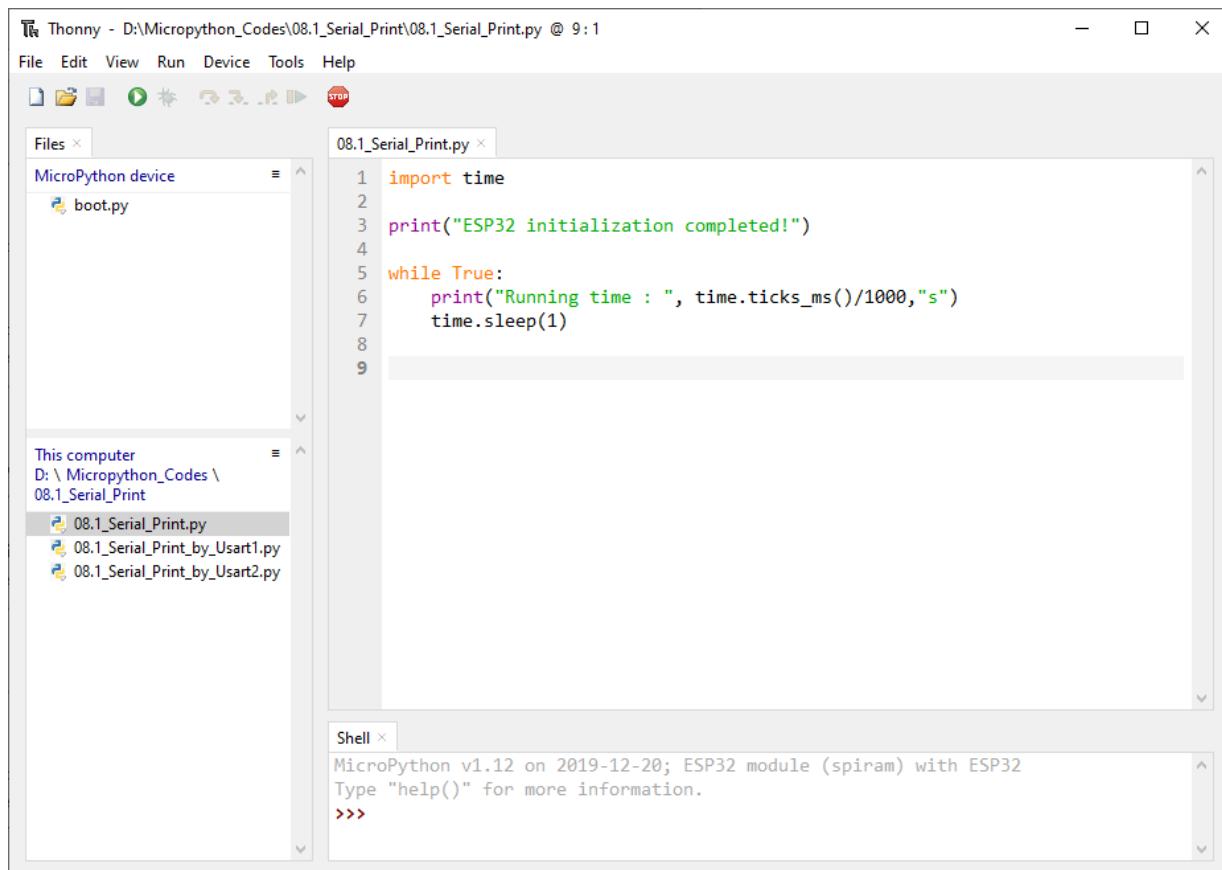
Connect Freenove ESP32 to the computer with USB cable



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “08.1_Serial_Print” and double “08.1_Serial_Print.py”.

08.1 Serial_Print



Click “Run current script” and observe the changes of “Shell”, which will display the time when ESP32 is powered on once per second.



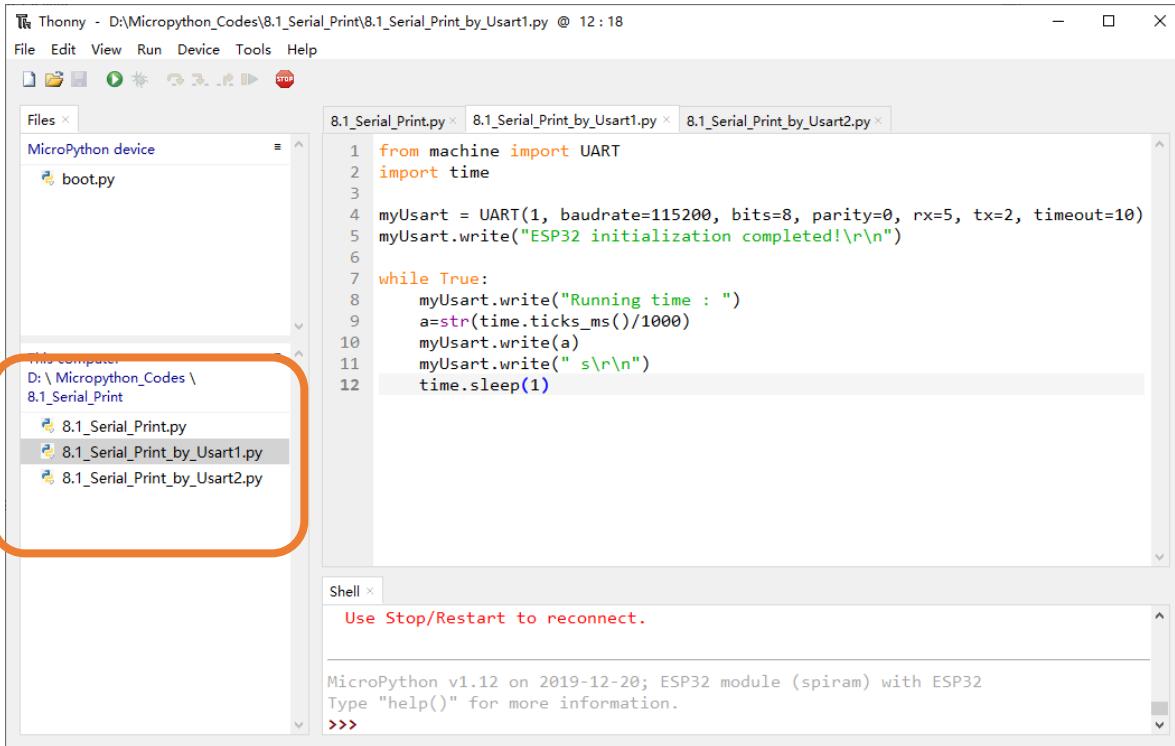
The following is the program code:

```

1 import time
2
3 print("ESP32 initialization completed!")
4
5 while True:
6     print("Running time : ", time.ticks_ms()/1000, "s")
7     time.sleep(1)

```

ESP32-WROVER has 3 serial ports, one of which is used as REPL, that is, Pin(1) and Pin(3) are occupied, and generally it is not recommended to be used as tx, rx. The other two serial ports can be configured simply by calling the UART module.



Reference

Class UART

Before each use of **UART** module, please add the statement “**from machine import UART**” to the top of python file.

UART(id, baudrate, bits, parity, rx, tx, stop, timeout): Define serial ports and configure parameters for them.

id: Serial Number. The available serial port number is 1 or 2

baudrate: Baud rate

bits: The number of each character.

parity: Check even or odd, with 0 for even checking and 1 for odd checking.

rx, tx: UAPTS's reading and writing pins

Pin(0)、Pin(2)、Pin(4)、Pin(5)、Pin(9)、Pin(10)、Pin(12~19)、Pin(21~23)、Pin(25)、Pin(26)、
Pin(34~36)、Pin(39)

Note: Pin(1) and Pin(3) are occupied and not recommend to be used as tx,rx.

stop: The number of stop bits, and the stop bit is 1 or 2.

timeout: timeout period (Unit: millisecond)

0 < timeout ≤ 0xFFFF FFFF (decimal: 0 < timeout ≤ 2147483647)

UART.init(baudrate, bits, parity, stop, tx, rx, rts, cts): Initialize serial ports

tx: writing pins of uart

rx: reading pins of uart

rts: rts pins of uart

cts: cts pins of uart

UART.read(nbytes): Read nbytes bytes

UART.read(): Read data

UART.write(buf): Write byte buffer to UART bus

UART.readline(): Read a line of data, ending with a newline character.

UART.readinto(buf): Read and write data into buffer.

UART.readinto(buf, nbytes): Read and write data into buffer.

UART.any(): Determine whether there is data in serial ports. If yes, return the number of bytes; Otherwise, return 0.

Project 8.2 Serial Read and Write

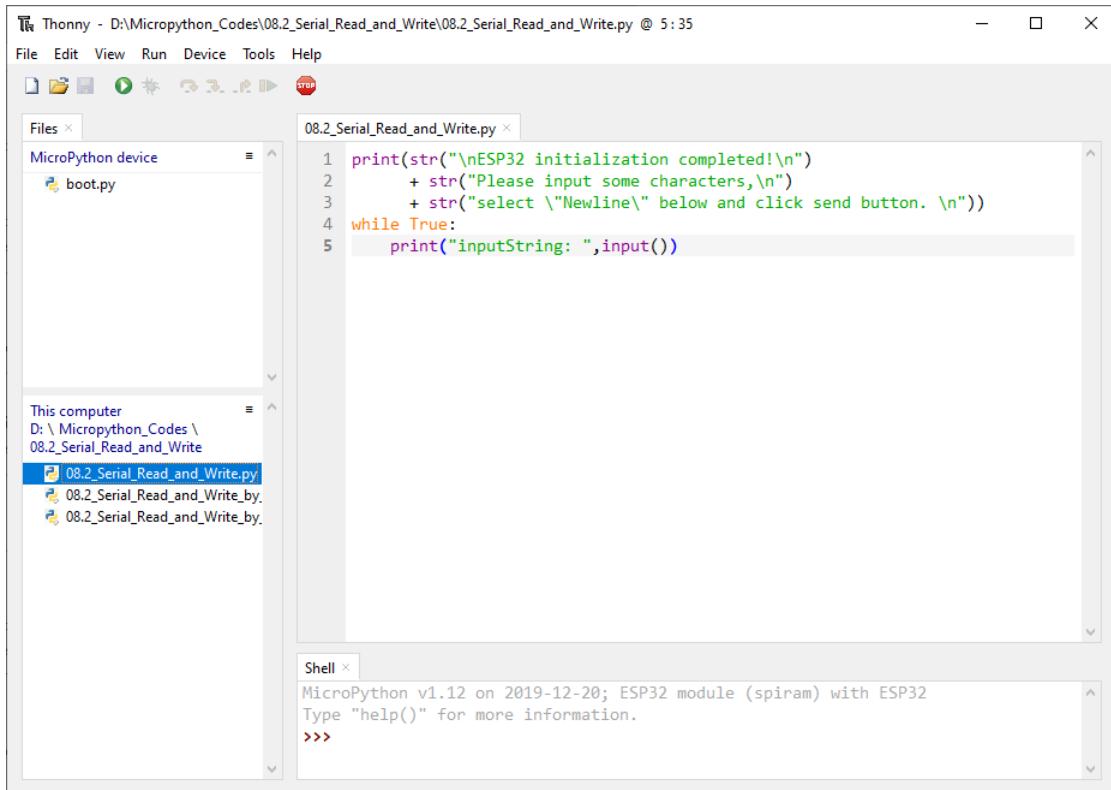
From last section, we use Serial port on Freenove ESP32 to send data to a computer, now we will use that to receive data from computer.

Component and Circuit are the same as in the previous project.

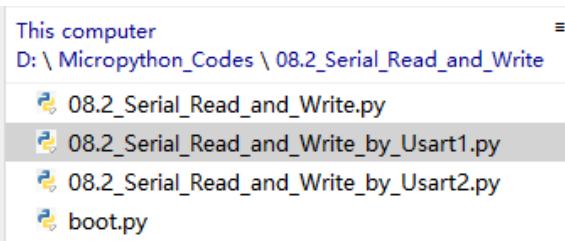
Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “08.2_Serial_Read_and_Write” and double click “08.2_Serial_Read_and_Write.py”.

08.2 Serial Read and Write



Click “Run current script” and ESP32 will print out data at “Shell” and wait for users to enter any messages. Press Enter to end the input, and “Shell” will print out data that the user entered. If you want to use other serial ports, you can use other python files in the same directory.



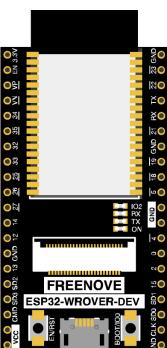
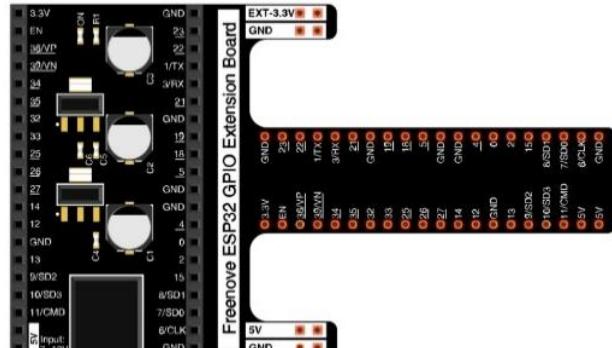
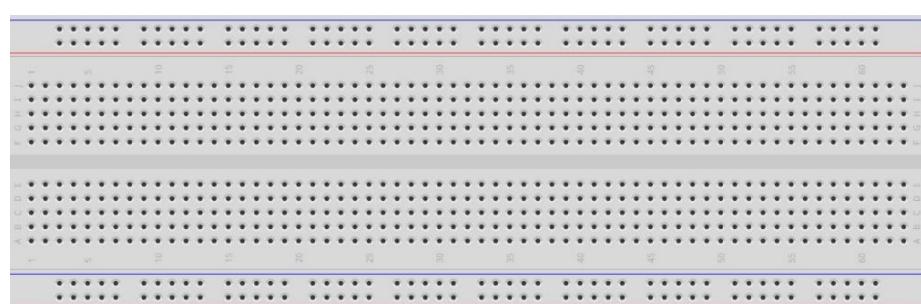
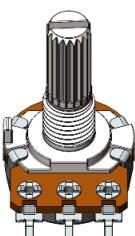
Chapter 9 AD/DA Converter

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

Project 9.1 Read the Voltage of Potentiometer

In this project, we will use the ADC function of ESP32 to read the voltage value of potentiometer. And then output the voltage value through the DAC to control the brightness of LED.

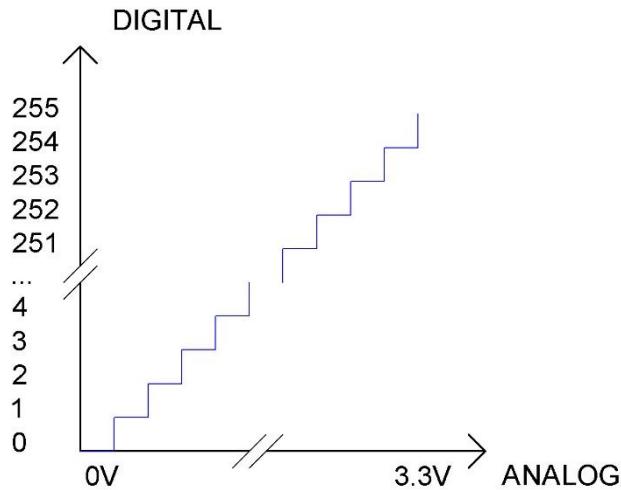
Component List

ESP32-WROVER x1	GPIO Extension Board x1		
			
Breadboard x1			
			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	Jumper M/M x5
			

Related 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 on ESP32 is 12 bits, that means the resolution is $2^{12}=4096$, and it represents a range (at 3.3V) will be divided equally to 4096 parts. The range of analog values corresponds to ADC values. 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---3.3/4095 V corresponds to digital 0;

Subsection 2: the analog in range of 3.3/4095 V---2*3.3 /4095V corresponds to digital 1;

...

The following analog will be divided accordingly.

The conversion formula is as follows:

$$ADCValue = \frac{\text{Analog Voltage}}{3.3} * 4095$$

DAC

The reversing of 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. ESP32 has two DAC output pins with 8-bit accuracy, GPIO25 and GPIO26, which can divide VDD (here is 3.3V) into $2^8=256$ parts. For example, when the digital quantity is 1, the output voltage value is $3.3/256 * 1$ V, and when the digital quantity is 128, the output voltage value is $3.3/256 * 128=1.65$ V, the higher the accuracy of DAC, the higher the accuracy of output voltage value will be.

The conversion formula is as follows:

$$Analog\ Voltage = \frac{DAC\ Value}{255} * 3.3\ (V)$$

ADC on ESP32

ESP32 has 6 pins can be used to measure analog signals. GPIO pin sequence number and analog pin definition are shown in the following table.

ADC number in ESP32	ESP32 GPIO number
ADC1	GPIO 36
ADC2	GPIO 39
ADC3	GPIO 34
ADC4	GPIO 35
ADC5	GPIO 32
ADC6	GPIO 33

DAC on ESP32

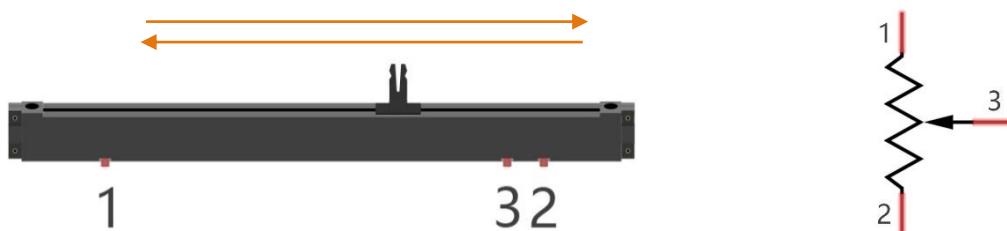
ESP32 has two 8-bit digital analog converters to be connected to GPIO25 and GPIO26 pins, respectively, and it is immutable. As shown in the following table,

Simulate pin number	GPIO number
DAC1	25
DAC2	26

Component knowledge

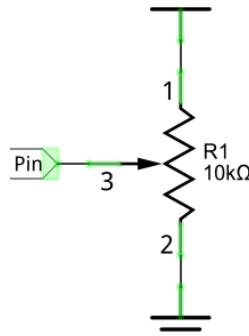
Potentiometer

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.



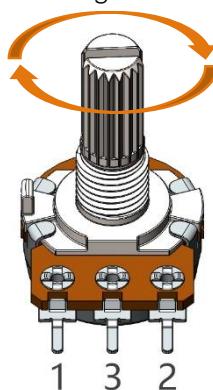
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

In the circuit. The both sides of resistance body are often connected to the positive and negative electrode of the power. When you slide the brush pin 3, you can get a certain voltage in the range of the power supply.



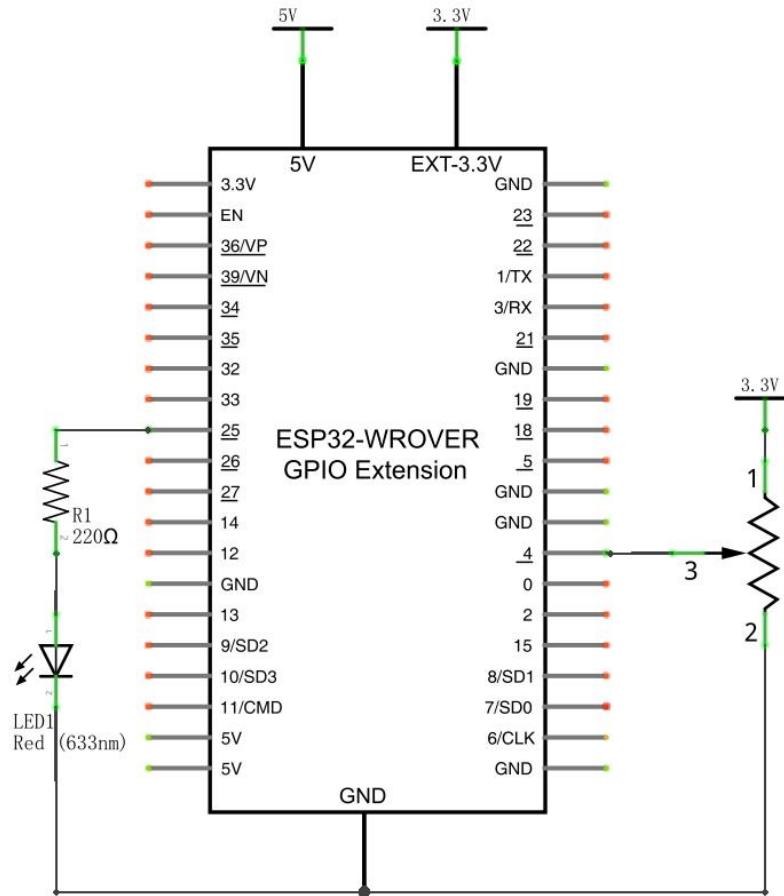
Rotary potentiometer

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

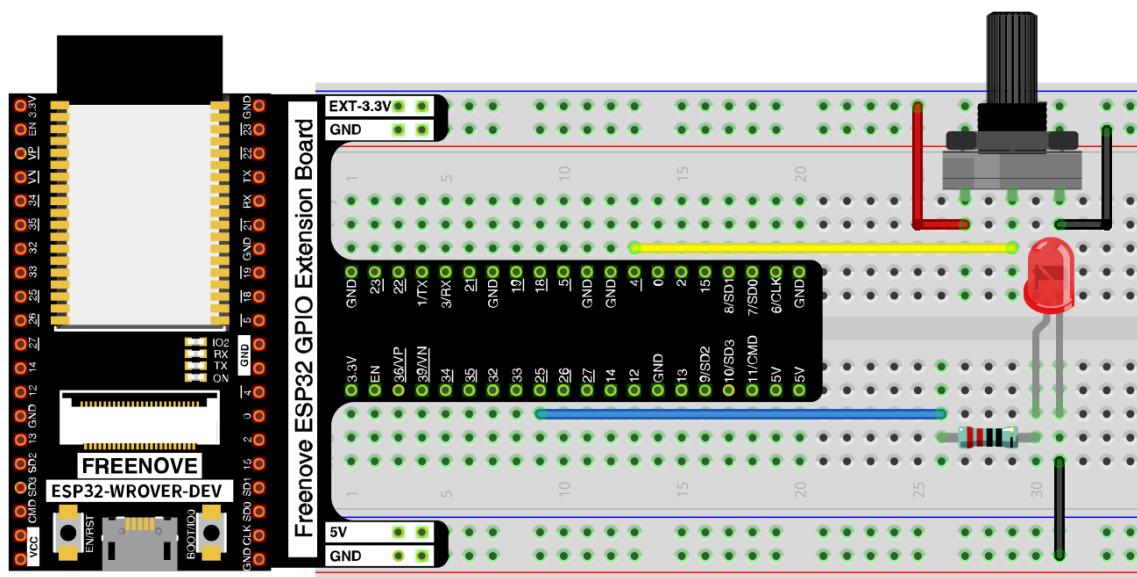


Circuit

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “09.1_AnalogRead” and then click “09.1_AnalogRead.py”.

09.1 AnalogRead

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\09.1_AnalogRead\09.1_AnalogRead.py @ 21:1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The left sidebar has two sections: "Files" and "This computer". Under "Files", there is a "MicroPython device" section containing "boot.py" and a "This computer" section containing "D:\Micropython_Codes\09.1_AnalogRead\09.1_AnalogRead.py". The main editor window displays the code for "09.1_AnalogRead.py":

```
1 from machine import ADC,Pin,DAC
2 import time
3
4 adc=ADC(Pin(36))
5 adc.atten(ADC.ATTN_11DB)
6 adc.width(ADC.WIDTH_12BIT)
7 dac =DAC(Pin(25))
8
9 try:
10     while True:
11         adcVal=adc.read()
12         dacVal=adcVal//16
13         voltage = adcVal / 4095.0 * 3.3
14         dac.write(dacVal)
15         print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
16         time.sleep_ms(100)
17 except:
18     pass
19
20
21
```

The bottom right panel is a "Shell" window showing the MicroPython v1.12 environment:

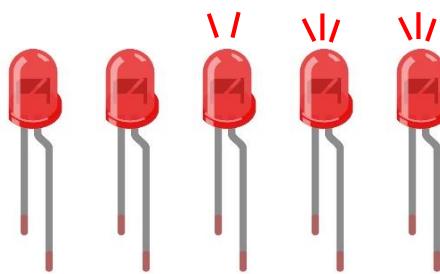
```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script” and observe the message printed in “Shell”.

The screenshot shows the Thonny Shell window. It starts with the MicroPython prompt and then shows the output of the script execution:

```
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
ADC Val: 0 DACVal: 0 Voltage: 0.0 V
ADC Val: 1023 DACVal: 63 Voltage: 0.8243956 V
ADC Val: 2819 DACVal: 176 Voltage: 2.271722 V
ADC Val: 4095 DACVal: 255 Voltage: 3.3 V
ADC Val: 4095 DACVal: 255 Voltage: 3.3 V
```

LEDs display as below:



"Shell" prints ADC value, DAC value, the output voltage of potentiometer and other information. In the code, we make the output voltage of the DAC pin equal to the input voltage of the ADC pin. Rotate the handle of the potentiometer, the printed information will change. When the voltage is greater than 1.6V (turn-on voltage of red LED), the LED starts to emit light. If you continue to increase the output voltage, the LED will gradually become brighter. And when the voltage is less than 1.6V, the LED will not light up, because this does not reach the turn-on voltage of the LED, which indirectly proves the difference between DAC and PWM. (If you have an oscilloscope, you can view the waveform output by the DAC through the oscilloscope)

The following is the code:

```

1  from machine import ADC, Pin, DAC
2  import time
3
4  adc=ADC(Pin(36))
5  adc.atten(ADC.ATTN_11DB)
6  adc.width(ADC.WIDTH_12BIT)
7  dac =DAC(Pin(25))
8
9  try:
10     while True:
11         adcVal=adc.read()
12         dacVal=adcVal//16
13         voltage = adcVal / 4095.0 * 3.3
14         dac.write(dacVal)
15         print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
16         time.sleep_ms(100)
17     except:
18         pass

```

Import Pin, ADC and DAC modules.

```

1  from machine import ADC, Pin, DAC
2  import time

```

Turn on and configure the ADC with the range of 0-3.3V and the data width of 12-bit data width, and turn on the DAC pin.

```

4  adc=ADC(Pin(36))
5  adc.atten(ADC.ATTN_11DB)
6  adc.width(ADC.WIDTH_12BIT)
7  dac =DAC(Pin(25))

```



Read ADC value once every 100 millisecods, convert ADC value to DAC value and output it, control the brightness of LED and print these data to “Shell”.

```

10     while True:
11         adcVal=adc.read()
12         dacVal=adcVal//16
13         voltage = adcVal / 4095.0 * 3.3
14         dac.write(dacVal)
15         print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
16         time.sleep_ms(100)

```

Reference

Class ADC

Before each use of ACD module, please add the statement “**from machine import ADC**” to the top of the python file.

machine.ADC(pin): Create an ADC object associated with the given pin.

pin: Available pins are: Pin(36)、Pin(39)、Pin(34)、Pin(35)、Pin(32)、Pin(33)。

ADC.read(): Read ADC and return the value.

ADC.attenu(db): Set attenuation ration (that is, the full range voltage, such as the voltage of 11db full range is 3.3V)

db: attenuation ratio

ADC.ATTIN_0DB —full range of 1.2V

ADC.ATTN_2_5_DB —full range of 1.5V

ADC.ATTN_6DB —full range of 2.0 V

ADC.ATTN_11DB —full range of 3.3V

ADC.width(bit): Set data width.

bit: data bit

ADC.WIDTH_9BIT —9 data width

ADC.WIDTH_10BIT — 10 data width

ADC.WIDTH_11BIT — 11 data width

ADC.WIDTH_12BIT — 12 data width

Class DAC

Before each use of **DAC** module, please add the statement “**from machine import DAC**” to the top of the python file.

machine.DAC(pin): Create a DAC object associated with the given pin.

pin: Available pins are: Pin(25)、Pin(26)

DAC.write(value): Output voltage

value: The range of data value: 0-255, corresponding output voltage of 0-3.3V

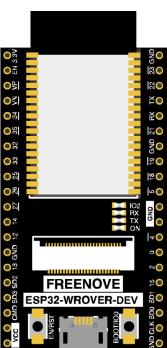
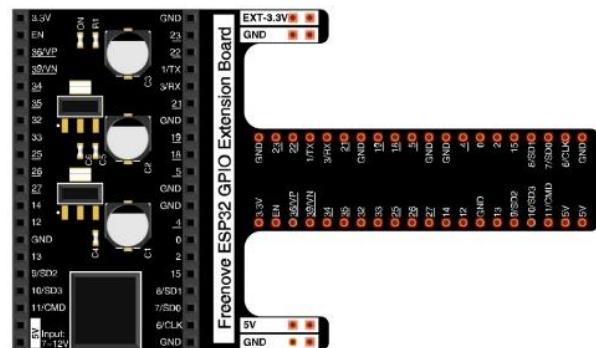
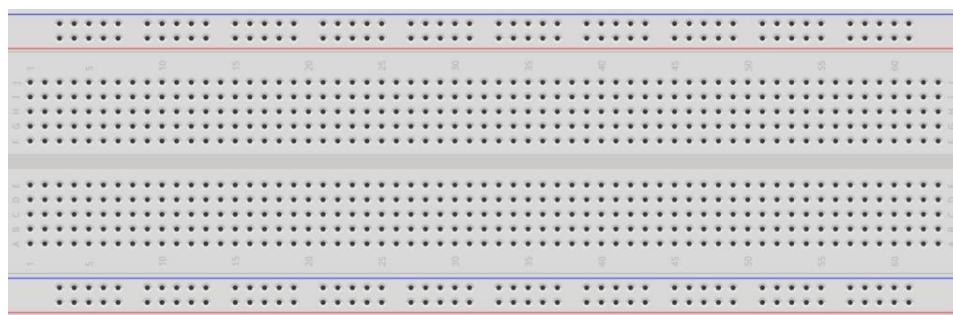
Chapter 10 TouchSensor

ESP32 offers up to 10 capacitive touch GPIO, and as you can see from the previous section, mechanical switches are prone to jitter that must be eliminated when used, which is not the case with ESP32's built-in touch sensor. In addition, on the service life, the touch switch also has advantages that mechanical switch is completely incomparable.

Project 10.1 Read Touch Sensor

This project reads the value of the touch sensor and prints it out.

Component List

ESP32-WROVER x1	GPIO Extension Board x1	Breadboard x1	Jumper M/M x1
			



Related knowledge

Touch sensor

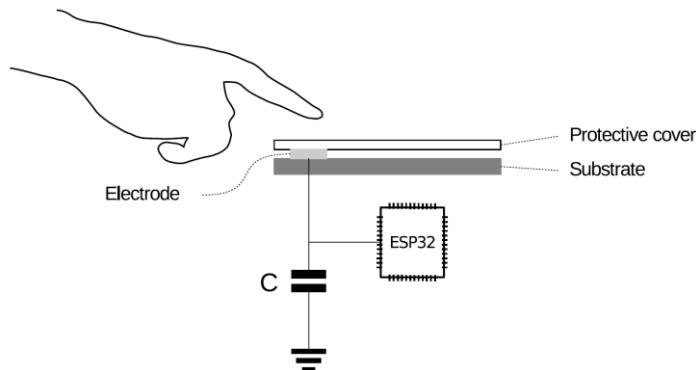
ESP32's touch sensor supports up to 7 GPIO channels as capacitive touch pins. Each pin can be used separately as an independent touch switch or be combined to produce multiple touch points. The following table is a list of available touch pins on ESP32.

Functions of pins	ESP32 GPIO number
GPIO4	GPIO4
MTDO	GPIO15
MTCK	GPIO13
MTDI	GPIO12
MTMS	GPIO14
32K_XN	GPIO33
32K_XP	GPIO32

The pin numbers are shown in the figure above. When you need to use the TouchPad, you only need to call the TouchPad function to initialize the corresponding pins.

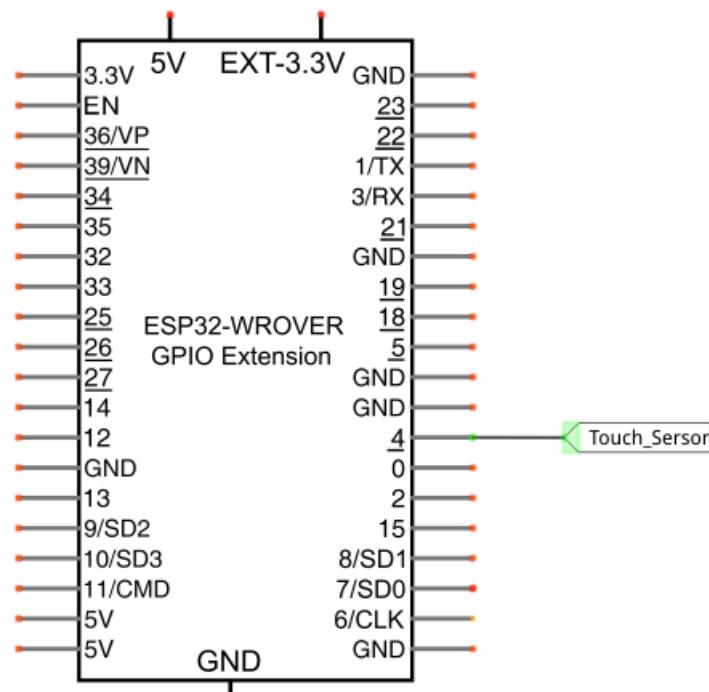
The electronic signals generated with the touch are analog, which are converted by the internal ADC. You may have found that all touch pins have featured with ADC.

The way to connect the hardware is as follows:

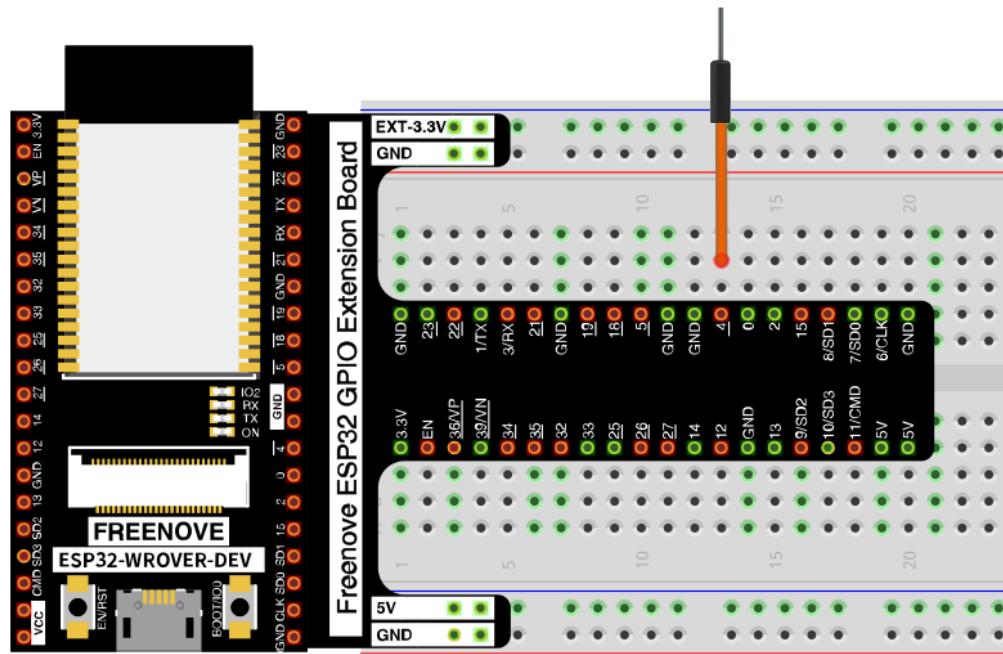


Circuit

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.1_Read_Touch_Sensor” and double click “10.1_Read_Touch_Sensor.py”.

10.1 Read Touch Sensor

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - D:\Micropython_Codes\10.1_Read_Touch_Sensor\10.1_Read_Touch_Sensor.py @ 8:1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and a stop button. The left sidebar has a "Files" tab and lists "MicroPython device" with "boot.py" and "This computer" with "D:\Micropython_Codes\10.1_Read_Touch_Sensor\10.1_Read_Touch_Sensor.py" selected. The main area contains a code editor titled "10.1_Read_Touch_Sensor.py" with the following code:

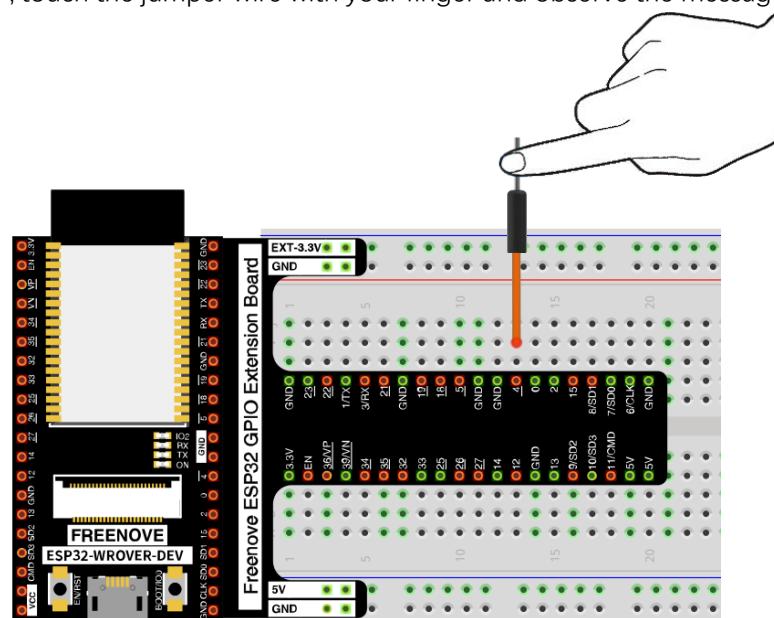
```
from machine import TouchPad, Pin
import time

tp = TouchPad(Pin(4,Pin.IN,Pin.PULL_UP))
while True:
    print("Touch value:",tp.read())
    time.sleep_ms(1000)
```

Below the code editor is a "Shell" tab with the output:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script”, touch the jumper wire with your finger and observe the messages printed in “Shell”.



Messages printed at "Shell":

The screenshot shows the Thonny IDE interface. In the top menu bar, it says "Thonny - D:\Micropython_Codes\10.1_Read_Touch_Sensor\10.1_Read_Touch_Sensor.py @ 8:1". The menu includes File, Edit, View, Run, Device, Tools, Help. Below the menu is a toolbar with icons for file operations like Open, Save, Run, Stop, and Help. On the left, there's a "Files" sidebar showing "MicroPython device" with "boot.py" and "This computer" with "D:\Micropython_Codes\10.1_Read_Touch_Sensor\10.1_Read_Touch_Sensor.py". The main area has tabs for "10.1_Read_Touch_Sensor.py" and "Shell". The code tab contains the following Python script:

```
1 from machine import TouchPad, Pin
2 import time
3
4 tp = TouchPad(Pin(4,Pin.IN,Pin.PULL_UP))
5 while True:
6     print("Touch value:",tp.read())
7     time.sleep_ms(1000)
8
```

The "Shell" tab shows the output of the script running on an ESP32 module:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Touch value: 329
Touch value: 47
Touch value: 330
Touch value: 329
Touch value: 330
Touch value: 47
Touch value: 330
Touch value: 52
Touch value: 49
Touch value: 330
Touch value: 48
```

Reference

Class TouchPad

Before each use of **TouchPad** module, please add the statement "**from machine import TouchPad**" to the top of the python file.

TouchPad(pin): Initialize the TouchPad object and associate it with ESP32 pins.

pin: Pin(4)、Pin(15)、Pin(13)、Pin(12)、Pin(14)、Pin(32)、Pin(33)

TouchPad.read(): Read the capacitance of touchpad. If your fingers touch TouchPad pins, the capacitance decreases; Otherwise, it will not change.



Project 10.2 TouchLamp

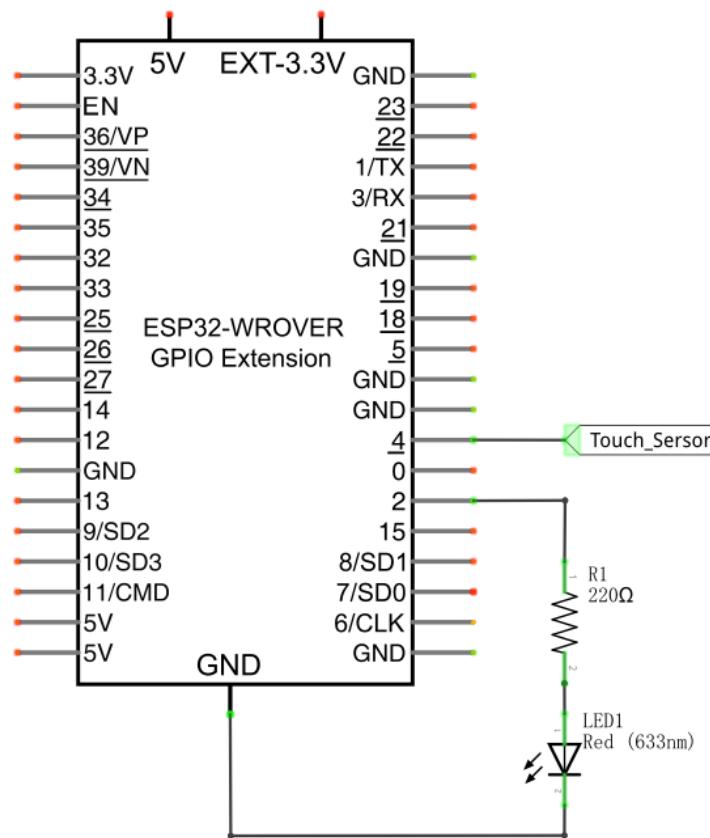
In this project, we will use ESP32's touch sensor to create a touch switch lamp.

Component List

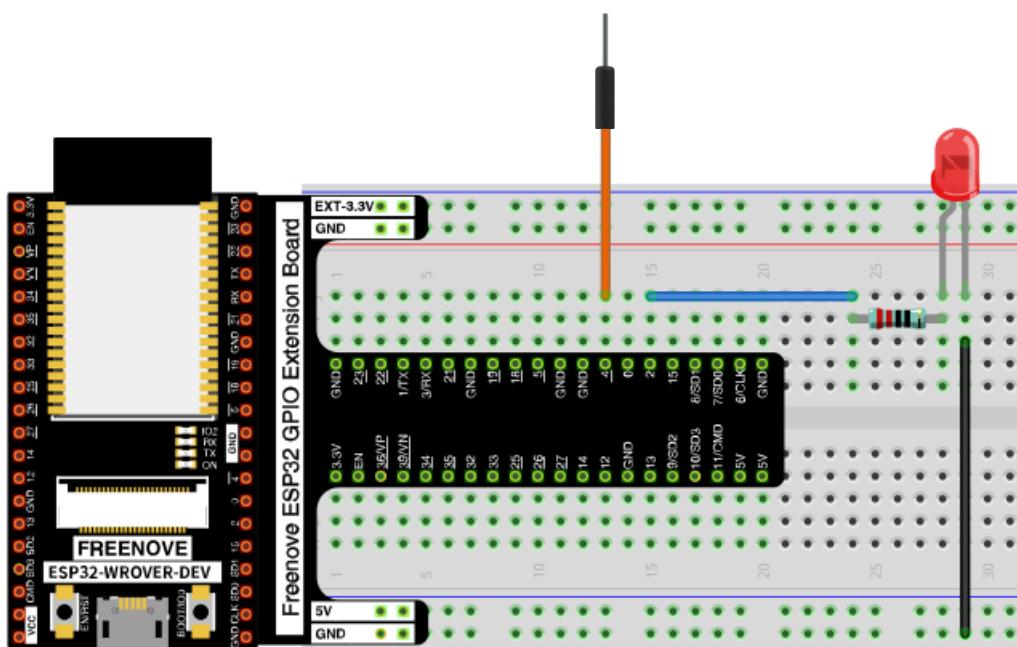
ESP32-WROVER x1	GPIO Extension Board x1	
Breadboard x1		
Jumper M/M x3	LED x1	Resistor 220Ω x1

Circuit

Schematic diagram



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



Code

Open “Thonny, click “This computer” → “D:” → “Micropython_Codes” → “10.2_TouchLamp”and double click “10.2_TouchLamp.py”.

10.2 TouchLamp

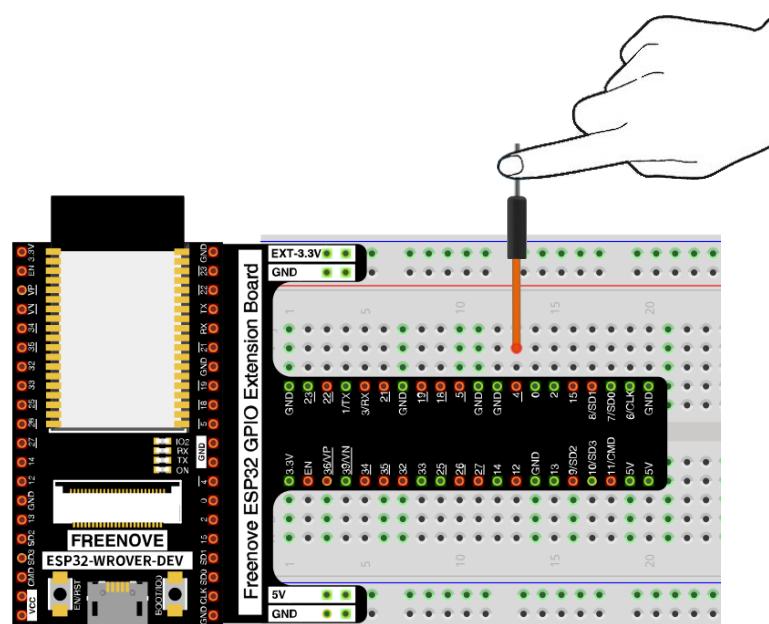
The screenshot shows the Thonny IDE interface with the following details:

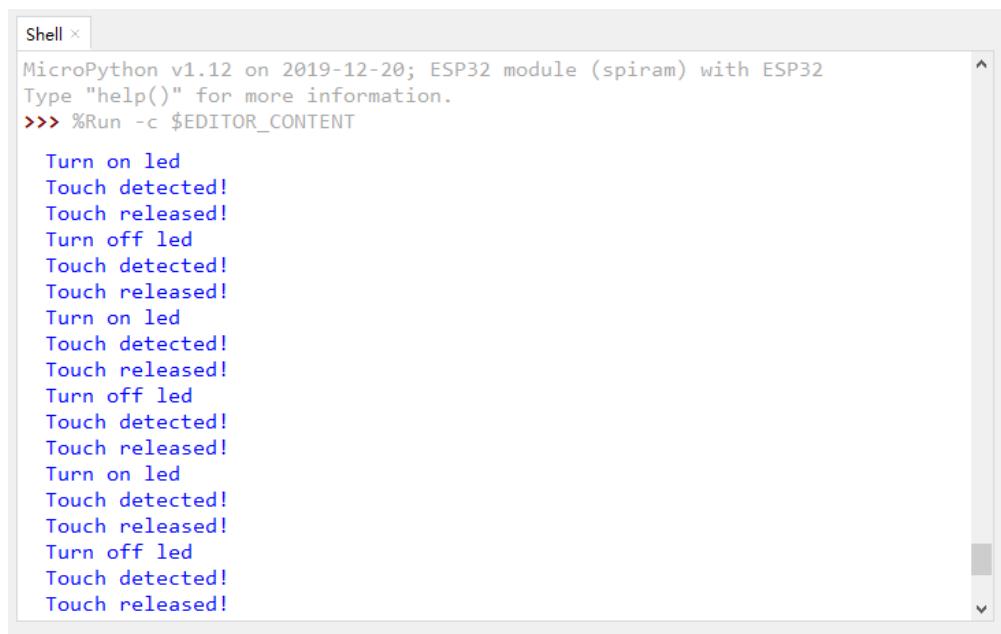
- Title Bar:** Thonny - D:\Micropython_Codes\10.2_TouchLamp\10.2_TouchLamp.py @ 33:1
- Menu Bar:** File, Edit, View, Run, Device, Tools, Help
- Toolbar:** Includes icons for file operations like Open, Save, and Run, along with a Stop button.
- File Explorer (Files tab):** Shows the project structure:
 - MicroPython device
 - boot.py
 - This computer
 - D:\Micropython_Codes\10.2_TouchLamp
 - 10.2_TouchLamp.py
- Code Editor (10.2_TouchLamp.py tab):** Displays the following Python code:

```
1 from machine import TouchPad, Pin
2 import time
3
4 PRESS_VAL=70      #Set a threshold to judge touch
5 RELEASE_VAL=200 #Set a threshold to judge release
6
7 led=Pin(2,Pin.OUT)
8 tp = TouchPad(Pin(4,Pin.IN,Pin.PULL_UP))
9 isPressed = 0
10
11 def reverseGPIO():
12     if led.value():
13         led.value(0)
14         print("Turn off led")
15     else:
16         led.value(1)
17         print("Turn on led")
18
19 while True:
20     if tp.read() < PRESS_VAL:
21         if not isPressed:
22             isPressed = 1
23             reverseGPIO()\n
```
- Shell (Shell tab):** Displays the MicroPython prompt:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click "Run current script" and then touch the jumper wire with your finger. The state of LED will change with each touch and the detection state of the touch sensor will be printed in the "Shell"





```

Shell >
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Turn on led
Touch detected!
Touch released!
Turn off led
Touch detected!
Touch released!
Turn on led
Touch detected!
Touch released!
Turn off led
Touch detected!
Touch released!
Turn on led
Touch detected!
Touch released!
Turn off led
Touch detected!
Touch released!

```

LED displays as follows:



The following is the program code:

```

1 from machine import TouchPad, Pin
2 import time
3
4 PRESS_VAL=70      #Set a threshold to judge touch
5 RELEASE_VAL=200 #Set a threshold to judge release
6
7 led=Pin(2,Pin.OUT)
8 tp = TouchPad(Pin(4,Pin.IN,Pin.PULL_UP))
9 isPressed = 0
10
11 def reverseGPIO():
12     if led.value():
13         led.value(0)
14         print("Turn off led")
15     else:
16         led.value(1)
17         print("Turn on led")
18 while True:
19     if tp.read() < PRESS_VAL:

```

```

20     if not isPressed:
21         isPressed = 1
22         reverseGPIO()
23         print("Touch detected!")
24         time.sleep_ms(100)
25     if tp.read() > RELEASE_VAL:
26         if isPressed:
27             isPressed = 0
28             print("Touch released!")
29             time.sleep_ms(100)

```

Import Pin and TouchPad modules.

```

1  from machine import TouchPad, Pin
2  import time

```

The closer the return value of the function read() is to 0, the more obviously the touch action is detected. As this is not a fixed value, a threshold value needs to be defined. When the value of the sensor is less than the threshold, it is considered a valid touch action. Similarly, define a threshold value for the released state, and the value between the sensor value and the threshold is regarded as an invalid interference value.

```

4  PRESS_VAL=70      #Set a threshold to judge touch
5  RELEASE_VAL=200 #Set a threshold to judge release

```

First, decide whether the touch is detected. If yes, print some messages, reverse the state of LED and set the flag bit isProcessed to 1 to avoid repeatedly executing the program after a touch is detected.

```

19    if tp.read() < PRESS_VAL:
20        if not isPressed:
21            isPressed = 1
22            reverseGPIO()
23            print("Touch detected!")
24            time.sleep_ms(100)

```

And then decide whether the touch key is released. If yes, print some messages, and set isProcessed to 0 to avoid repeatedly executing the program after a touch is released and to prepare for the next touch detector.

```

25    if tp.read() > RELEASE_VAL:
26        if isPressed:
27            isPressed = 0
28            print("Touch released!")
29            time.sleep_ms(100)

```

Customize a function that reverses the output level of the LED each time it is called.

```

11  def reverseGPIO():
12      if led.value():
13          led.value(0)
14          print("Turn off led")
15      else:
16          led.value(1)
17          print("Turn on led")

```

Chapter 11 Potentiometer & LED

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

Project 11.1 Soft Light

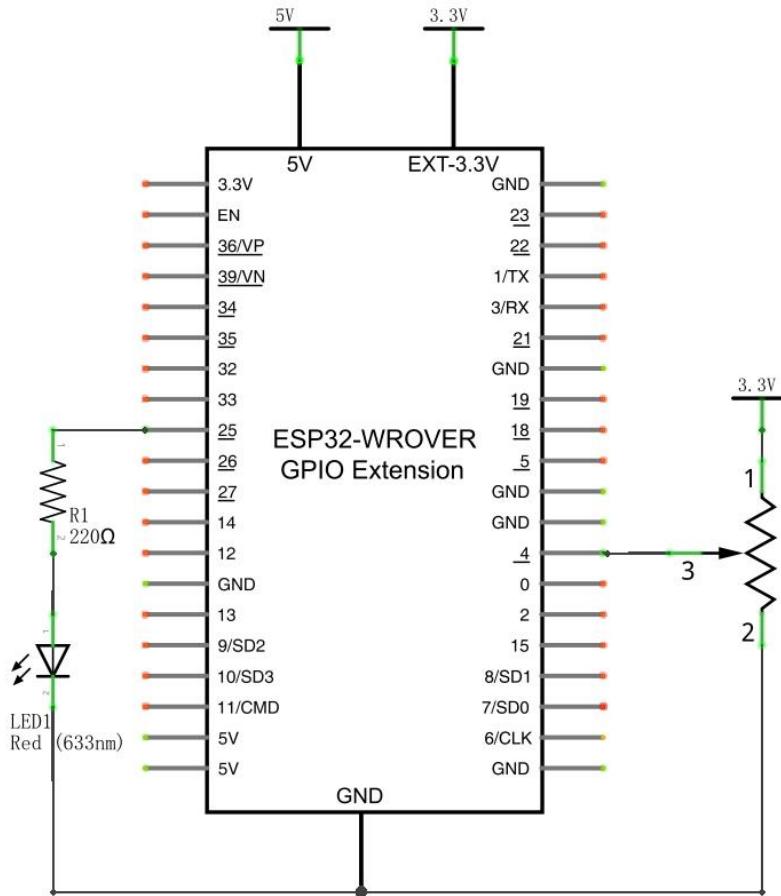
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

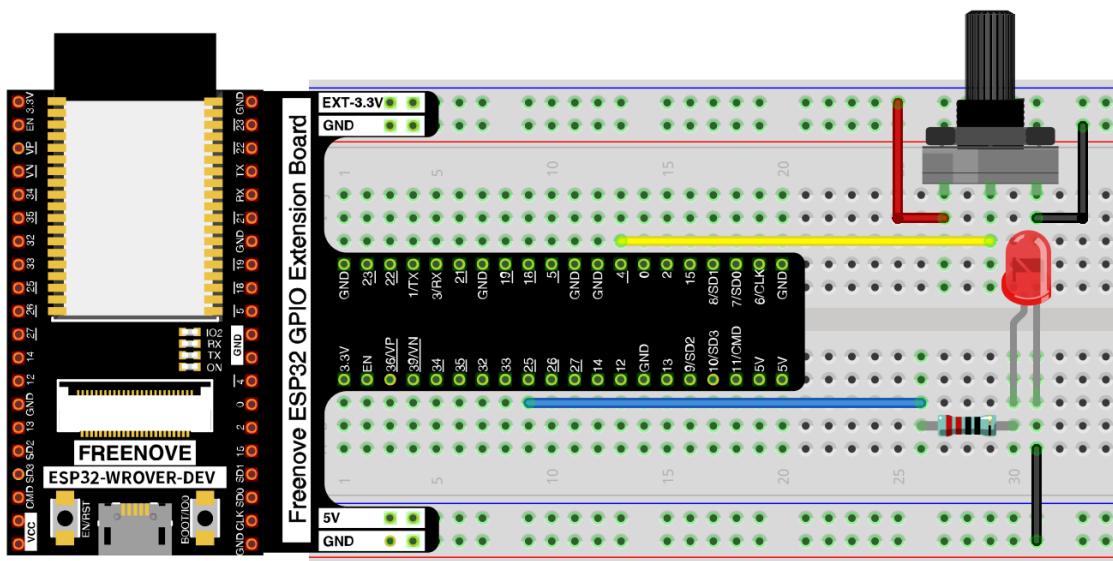
ESP32-WROVER x1	GPIO Extension Board x1		
Breadboard x1			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	Jumper M/M x5

Circuit

Schematic diagram



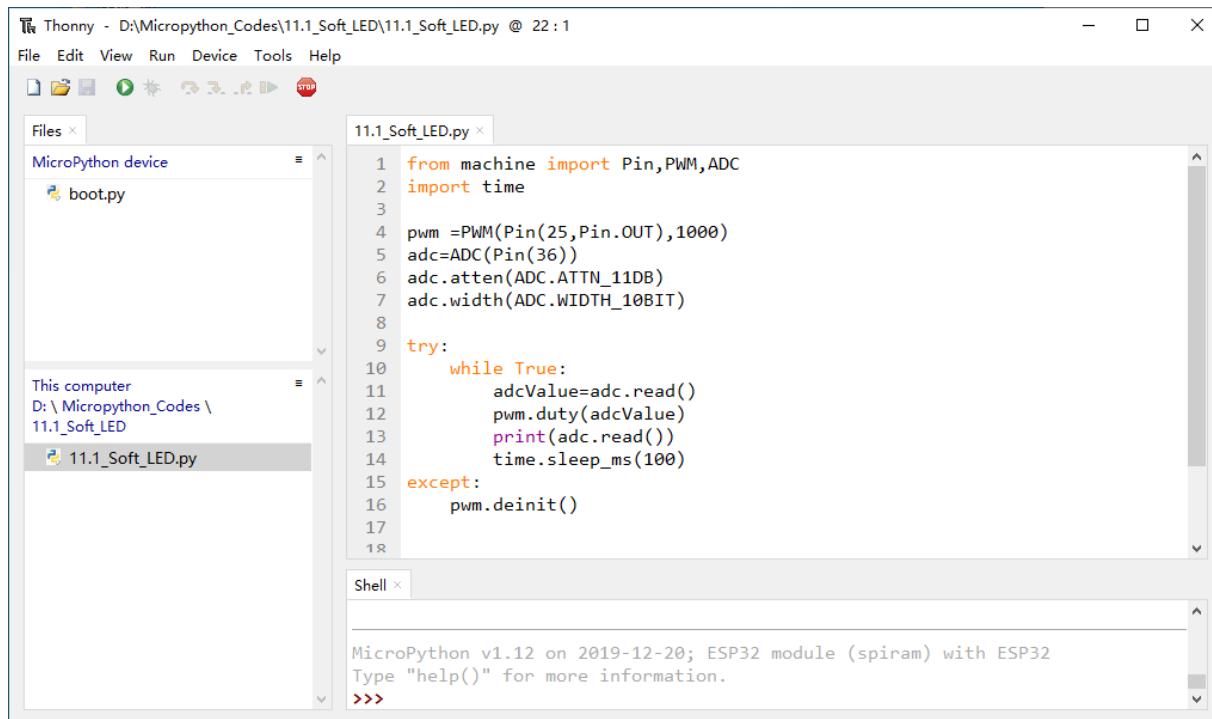
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “11.1_Soft_LED” and double click “11.1_Soft_LED.py”.

11.1 Soft LED



Click “Run current script”. Rotate the handle of potentiometer and the brightness of LED will change correspondingly.

The following is the code:

```
1 from machine import Pin, PWM, ADC
2 import time
3
4 pwm = PWM(Pin(25, Pin.OUT), 1000)
5 adc=ADC(Pin(36))
6 adc.atten(ADC.ATTN_11DB)
7 adc.width(ADC.WIDTH_10BIT)
8
9 try:
10     while True:
11         adcValue=adc.read()
12         pwm.duty(adcValue)
13         print(adc.read())
14         time.sleep_ms(100)
15
16 except:
17     pwm.deinit()
```

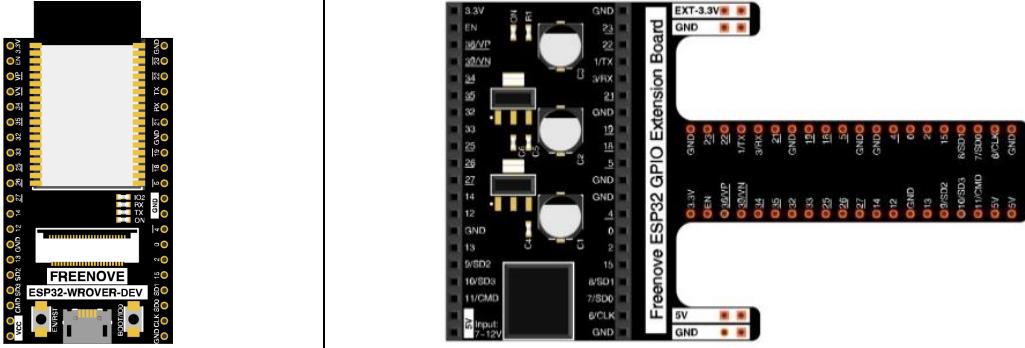
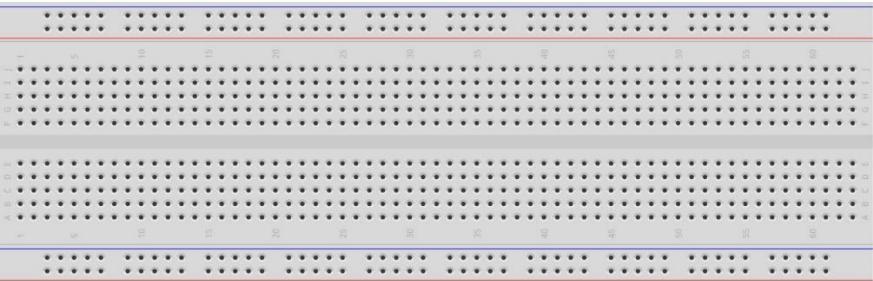
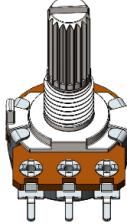
In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED

brightness.

Project 11.2 Soft Colorful Light

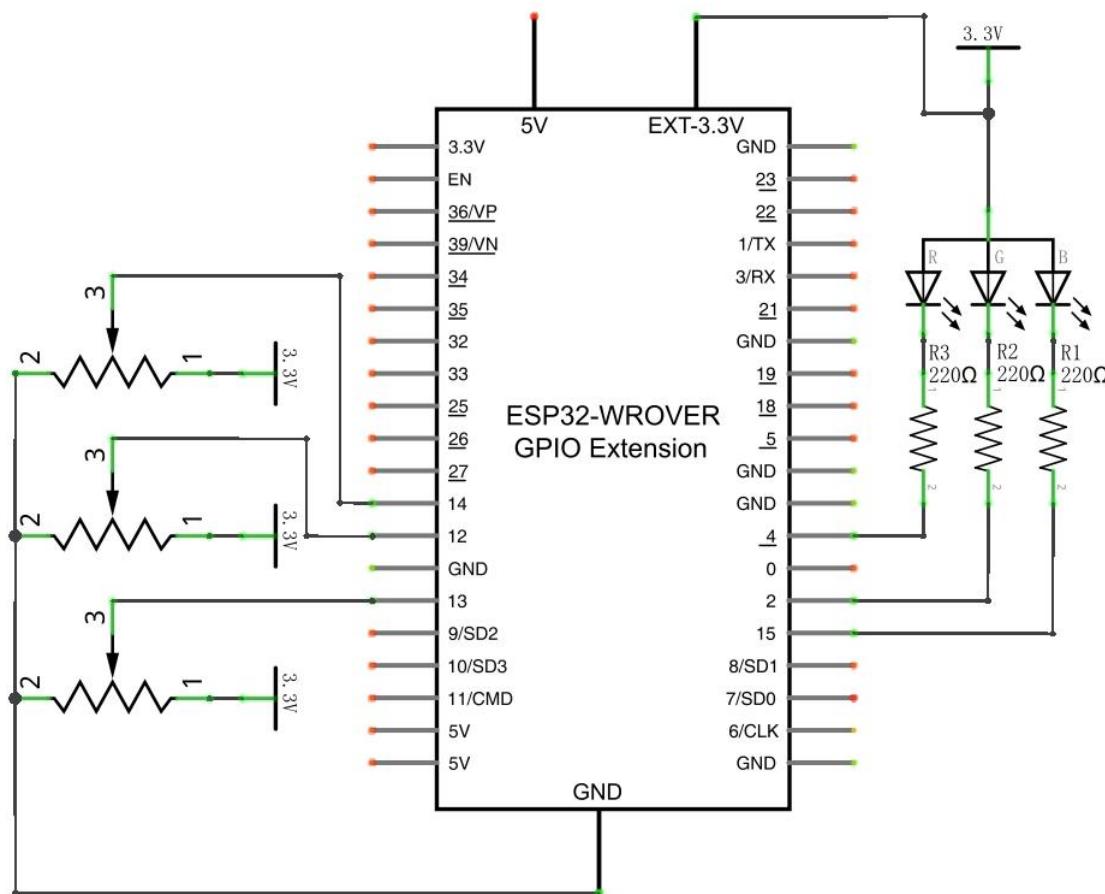
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED,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

Component List

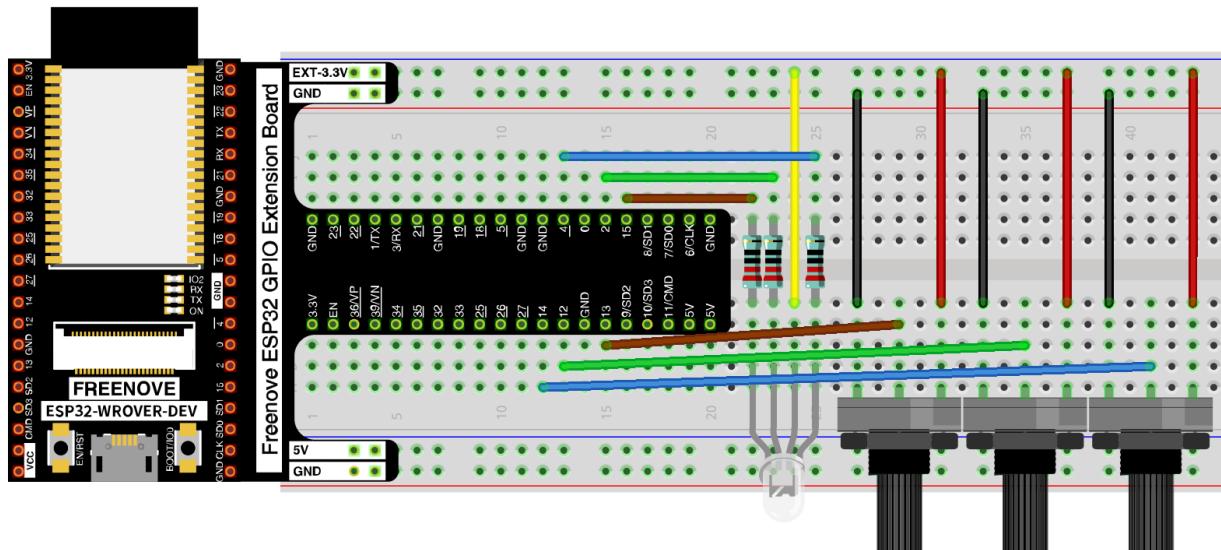
ESP32-WROVER x1	GPIO Extension Board x1
	
Breadboard x1	
Rotary potentiometer x3	Resistor 220Ω x3
	
	RGBLED x1
	
	Jumper M/M x13
	

Circuit

Schematic diagram



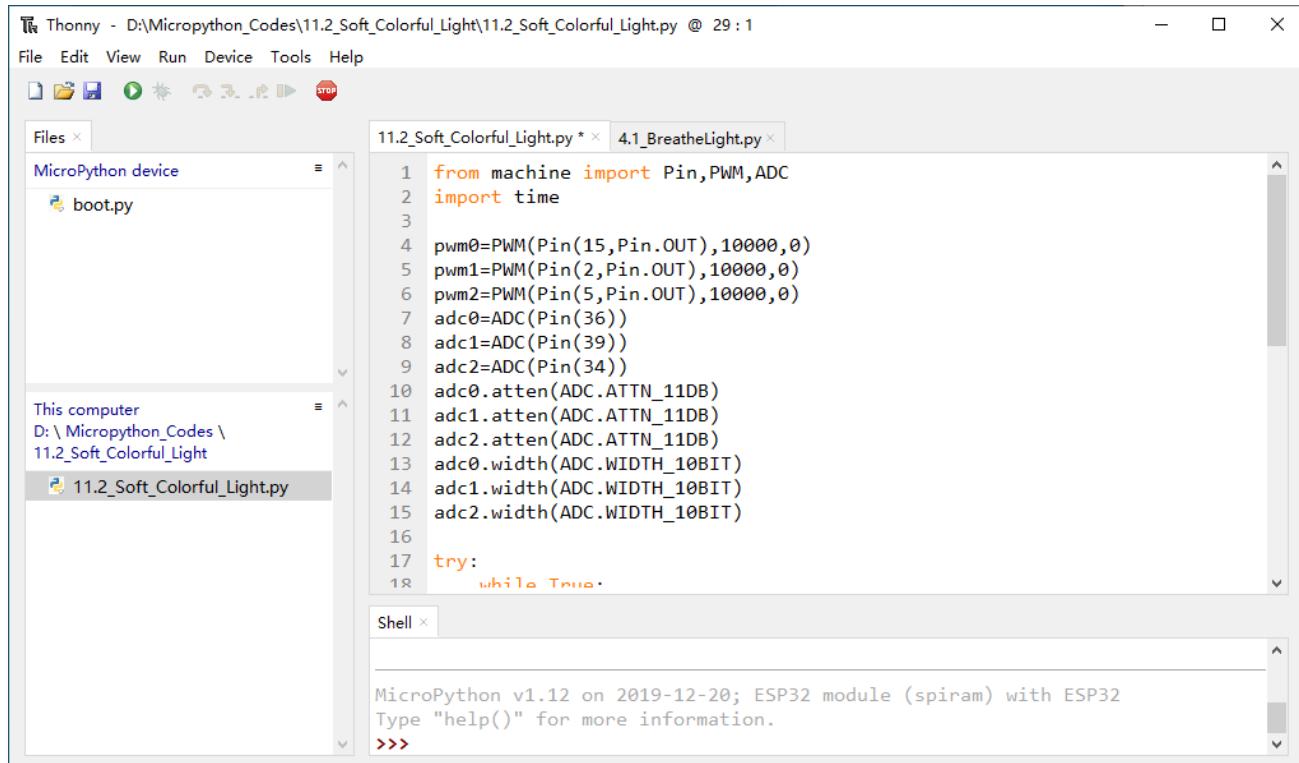
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “11.2_Soft_Colorful_Light” and double click “11.2_Soft_Colorful_Light.py”.

11.2 Soft Colorful Light



Click “Run current script” and control the change of RGBLED color by rotating the handles of three rotary potentiometers.

The following is the program code:

```

from machine import Pin, PWM, ADC
import time

pwm0=PWM(Pin(15,Pin.OUT),10000,0)
pwm1=PWM(Pin(2,Pin.OUT),10000,0)
pwm2=PWM(Pin(5,Pin.OUT),10000,0)
adc0=ADC(Pin(36))
adc1=ADC(Pin(39))
adc2=ADC(Pin(34))
adc0.atten(ADC.ATTN_11DB)
adc1.atten(ADC.ATTN_11DB)
adc2.atten(ADC.ATTN_11DB)
adc0.width(ADC.WIDTH_10BIT)
adc1.width(ADC.WIDTH_10BIT)
adc2.width(ADC.WIDTH_10BIT)

```

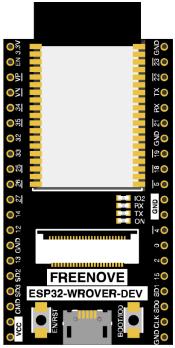
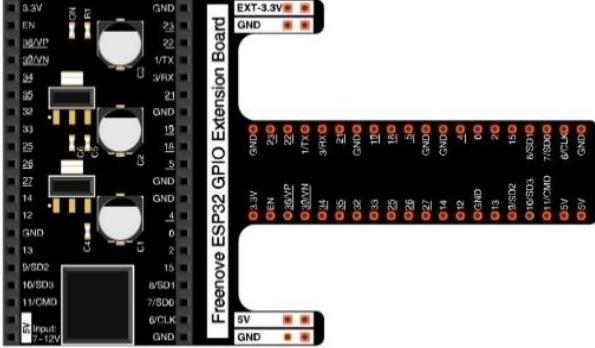
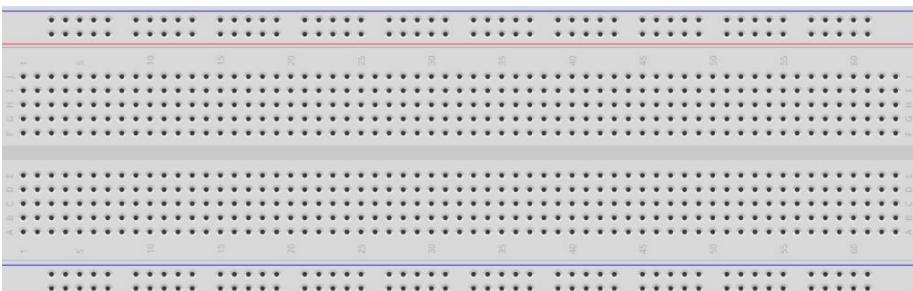
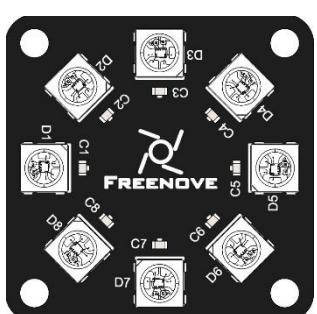
```
17 try:  
18     while True:  
19         pwm0.duty(1023-adc0.read())  
20         pwm1.duty(1023-adc1.read())  
21         pwm2.duty(1023-adc2.read())  
22         time.sleep_ms(100)  
23     except:  
24         pwm0.deinit()  
25         pwm1.deinit()  
26         pwm2.deinit()
```

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

Project 11.3 Soft Rainbow Light

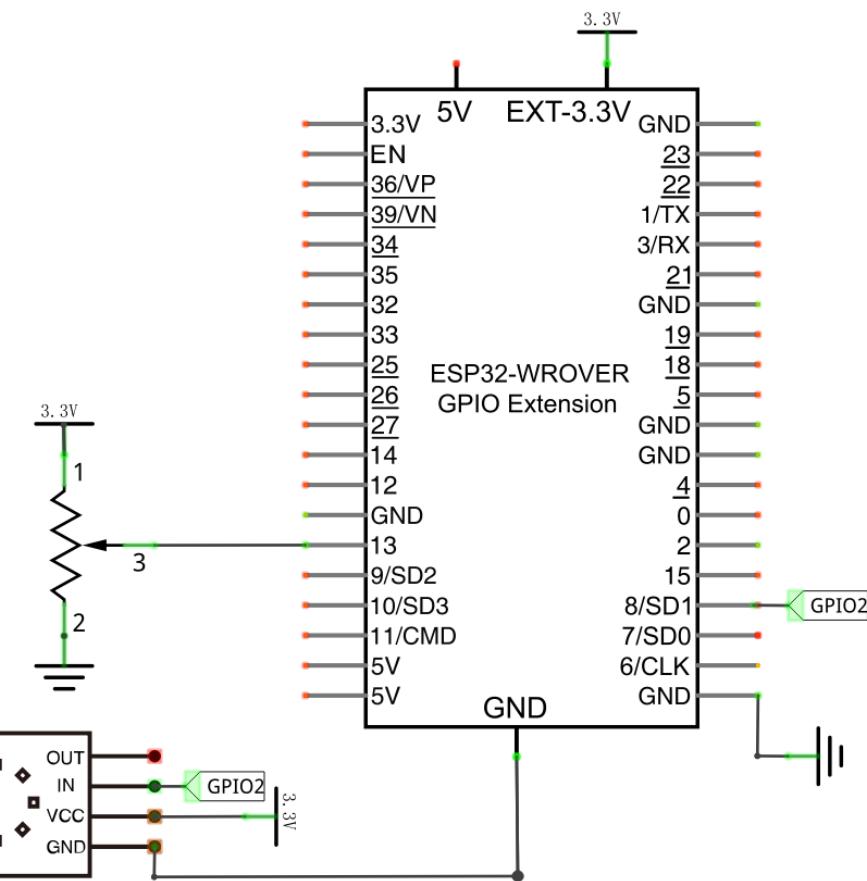
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

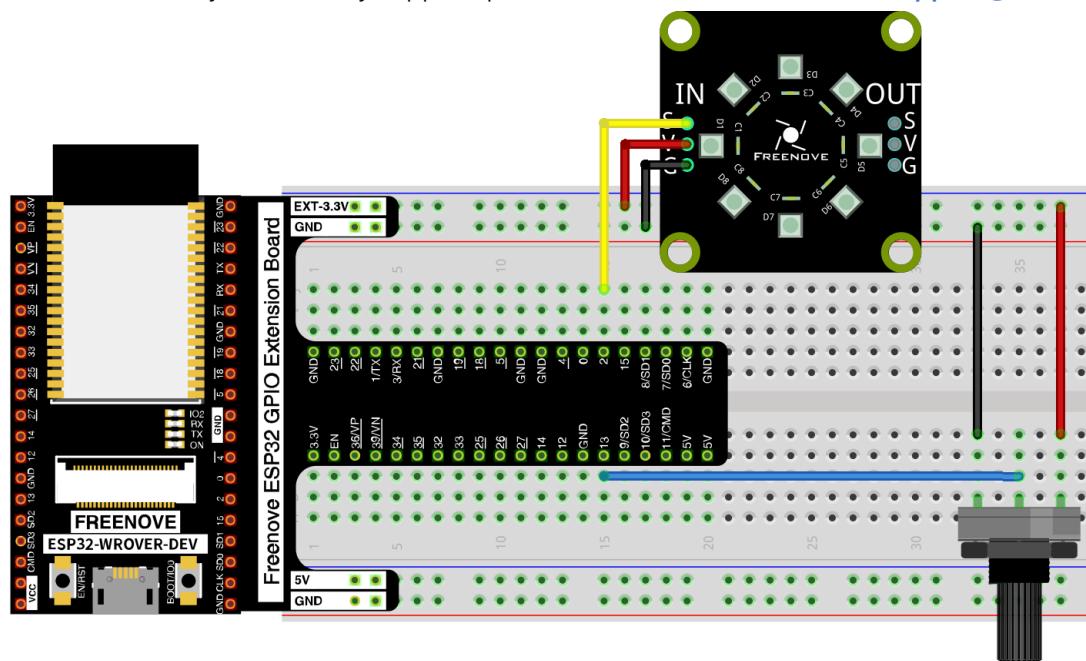
ESP32-WROVER x1	GPIO Extension Board x1	
		
Breadboard x1		
Freenove 8 RGB LED Module x1	Jumper F/M x3 Jumper M/M x3	

Circuit

Schematic diagram



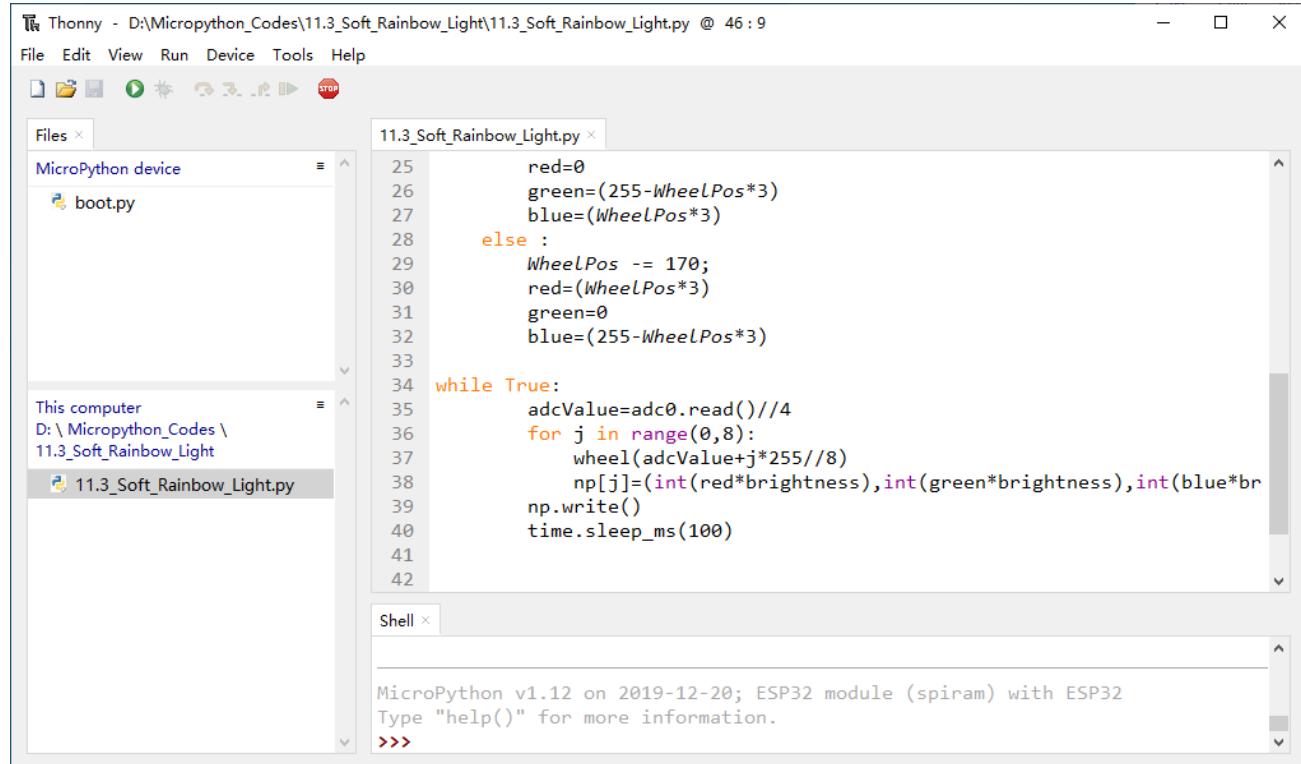
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “11.3_Soft_Rainbow_Light” and double click “11.3_Soft_Rainbow_Light.py”.

11.3 Soft Rainbow Light



```

Thonny - D:\Micropython_Codes\11.3_Soft_Rainbow_Light\11.3_Soft_Rainbow_Light.py @ 46 : 9
File Edit View Run Device Tools Help
STOP

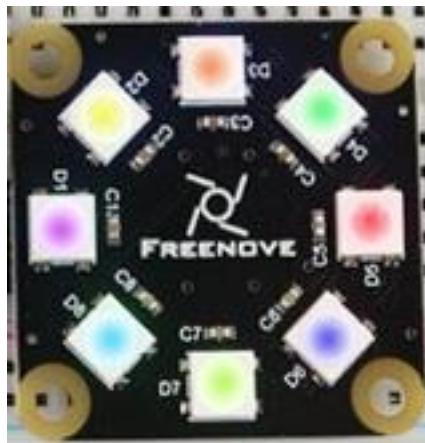
Files ×
MicroPython device
boot.py

This computer
D:\Micropython_Codes\
11.3_Soft_Rainbow_Light
11.3_Soft_Rainbow_Light.py

11.3_Soft_Rainbow_Light.py ×
25     red=0
26     green=(255-WheelPos*3)
27     blue=(WheelPos*3)
28     else :
29         WheelPos -= 170;
30         red=(WheelPos*3)
31         green=0
32         blue=(255-WheelPos*3)
33
34 while True:
35     adcValue=adc0.read()//4
36     for j in range(0,8):
37         wheel(adcValue+j*255//8)
38         np[j]=(int(red*brightness),int(green*brightness),int(blue*brightness))
39         np.write()
40         time.sleep_ms(100)
41
42
Shell ×
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

```

Click “Run current script”. Rotate the handle of potentiometer and the color of the lights will change.



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

The following is the program code:

```
1  from machine import Pin, ADC
2  import neopixel
3  import time
4
5  np = neopixel.NeoPixel(Pin(2, Pin.OUT), 8)
6
7  brightness=0.1      #brightbess
8  red=0               #red
9  green=0              #green
10 blue=0               #blue
11
12 adc0=ADC(Pin(36))
13 adc0.atten(ADC.ATTN_11DB)
14 adc0.width(ADC.WIDTH_10BIT)
15
16 def wheel(pos):
17     global red, green, blue
18     WheelPos=pos%255
19     if WheelPos<85:
20         red=(255-WheelPos*3)
21         green=(WheelPos*3)
22         blue=0
23     elif WheelPos>=85 and WheelPos<170:
24         WheelPos -= 85;
25         red=0
26         green=(255-WheelPos*3)
27         blue=(WheelPos*3)
28     else :
29         WheelPos -= 170;
30         red=(WheelPos*3)
31         green=0
32         blue=(255-WheelPos*3)
33
34 while True:
35     adcValue=adc0.read()//4
36     for j in range(0,8):
37         wheel(adcValue+j*255//8)
38         np[j]=(int(red*brightness), int(green*brightness), int(blue*brightness))
39         np.write()
40         time.sleep_ms(100)
```

The logic of the code is basically the same as the previous project [Rainbow Light](#). The difference is that in this code, the starting point of the color is controlled by the potentiometer.

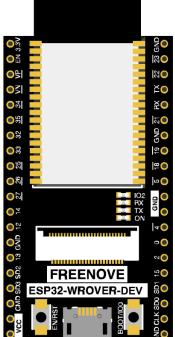
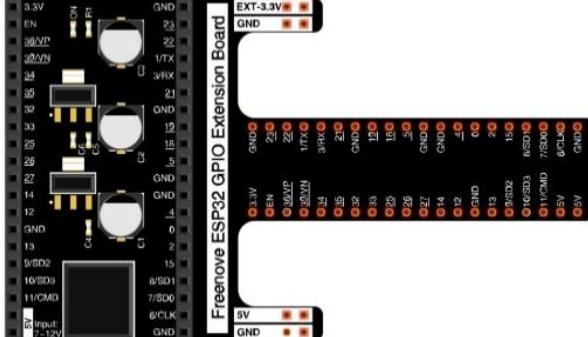
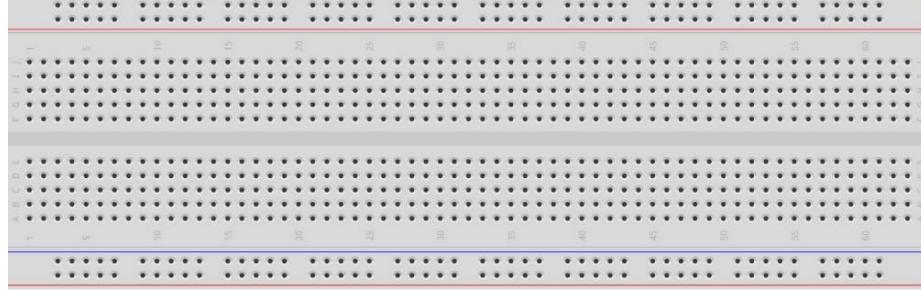
Chapter 12 Photoresistor & LED

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

Project 12.1 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.

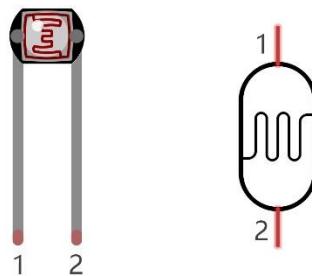
Component List

ESP32-WROVER x1	GPIO Extension Board x1
	
	
Breadboard x1	
	
Photoresistor x1	Resistor
	220Ω x1 10KΩ x1
	LED x1
	
	Jumper M/M x4
	

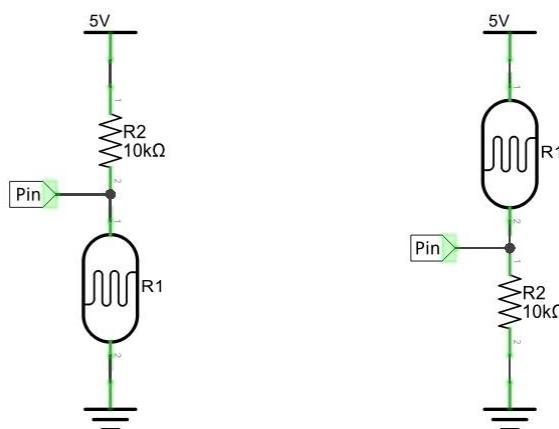
Component knowledge

Photoresistor

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



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

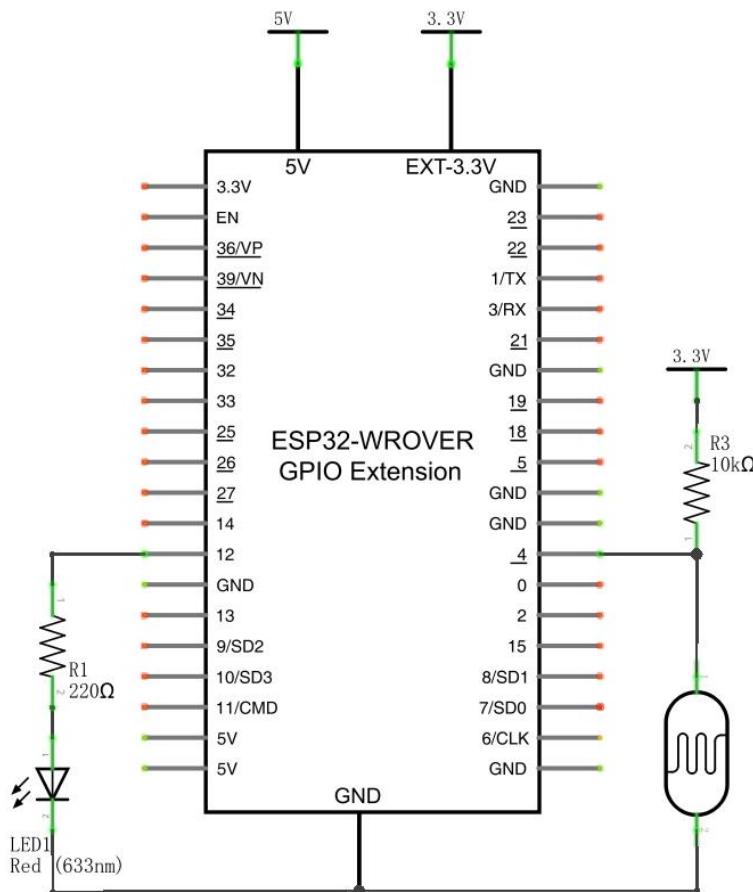


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

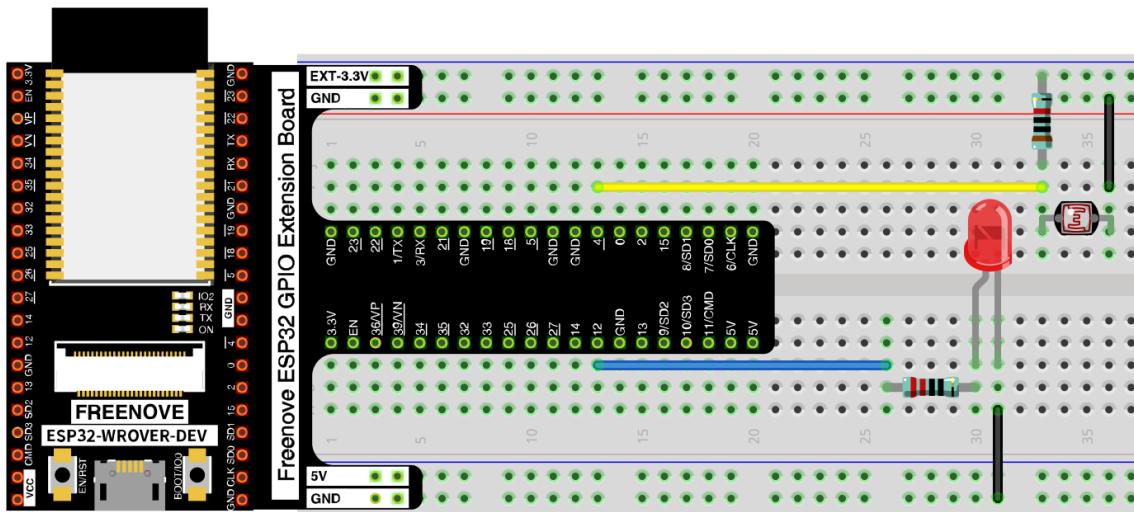
Circuit

The circuit of this project is similar to SoftLight. The only difference is that the input signal is changed from a potentiometer to a combination of a photoresistor and a resistor.

Schematic diagram



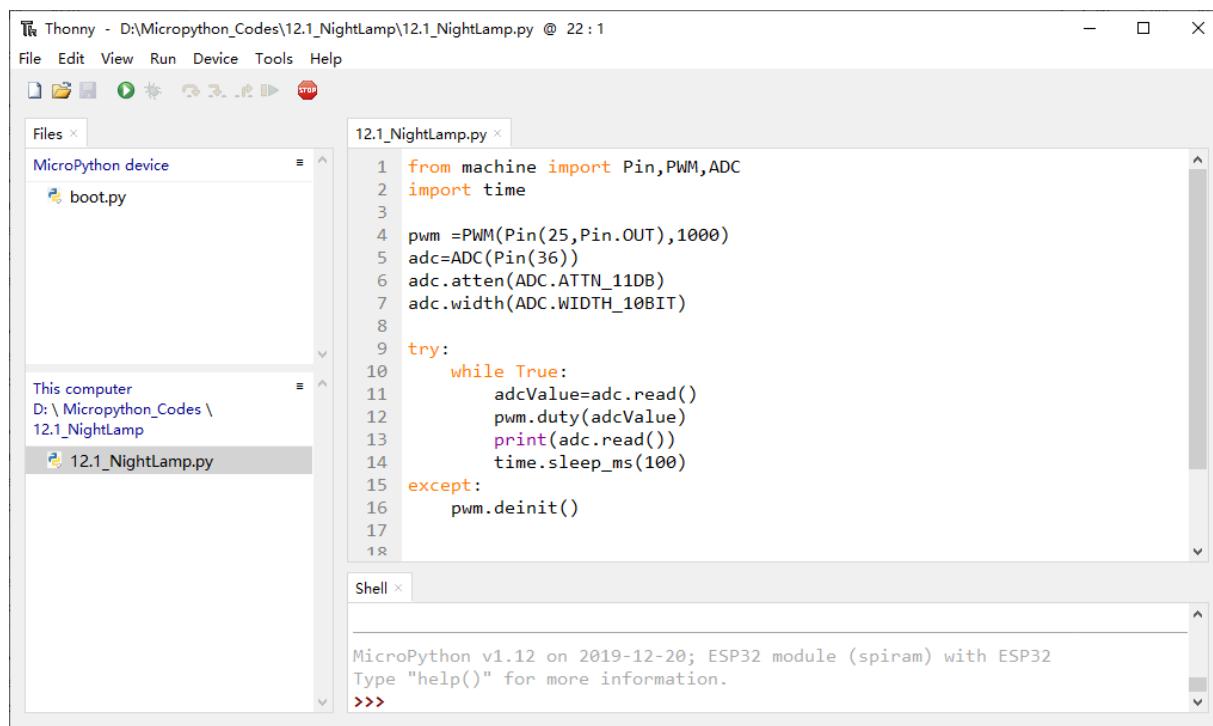
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Codes of this project is logically the same as the project [Soft Light](#).

12.1 Nightlamp



Click “Run current script”. Cover the photoresistor with your hands or illuminate it with lights, the brightness of LEDs will change.

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

The following is the program code:

```

1 from machine import Pin, PWM, ADC
2 import time
3
4 pwm =PWM(Pin(25, Pin.OUT), 1000)
5 adc=ADC(Pin(36))
6 adc.atten(ADC.ATTN_11DB)
7 adc.width(ADC.WIDTH_10BIT)
8
9 try:
10     while True:
11         adcValue=adc.read()
12         pwm.duty(adcValue)
13         print(adc.read())
14         time.sleep_ms(100)
15 except:
16     pwm.deinit()

```



Chapter 13 Thermistor

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

Project 13.1 Thermometer

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

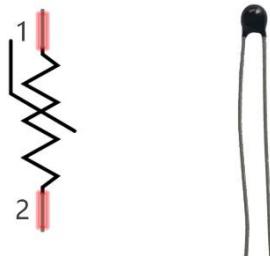
Component List

ESP32-WROVER x1	GPIO Extension Board x1	
Breadboard x1		
Thermistor x1	Resistor 1kΩ x1	Jumper M/M x3

Component knowledge

Thermistor

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



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

$$R_t = R * \text{EXP} \left[B * \left(\frac{1}{T_2} - \frac{1}{T_1} \right) \right]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

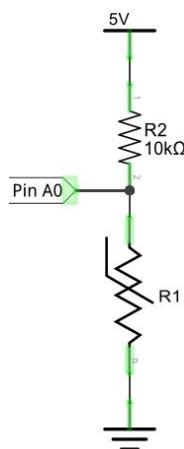
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



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

Therefore, the temperature formula can be derived as:

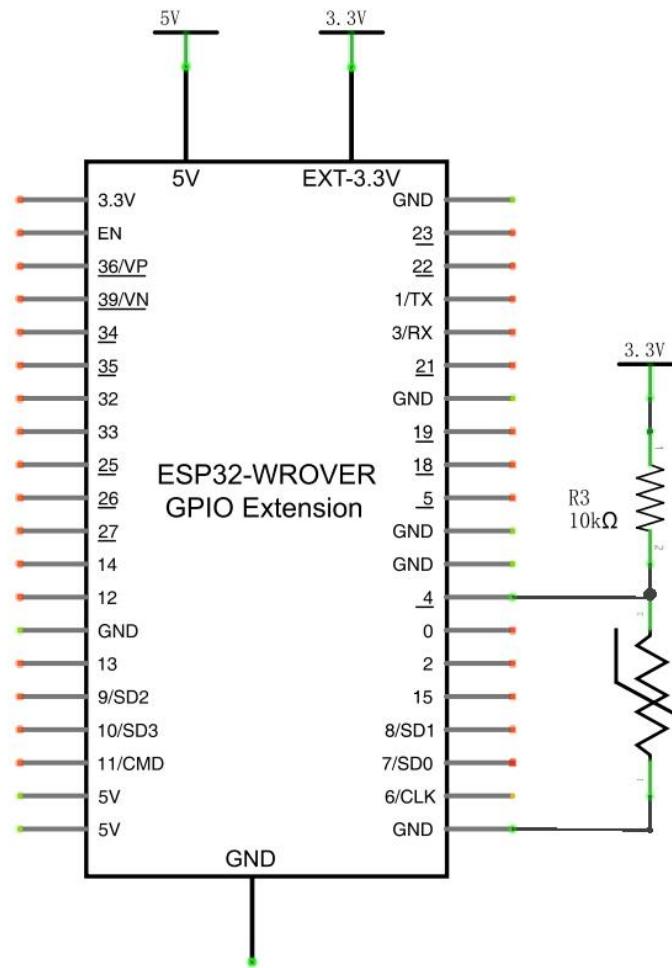
$$T_2 = 1 / \left(\frac{1}{T_1} + \ln \left(\frac{R_t}{R} \right) / B \right)$$



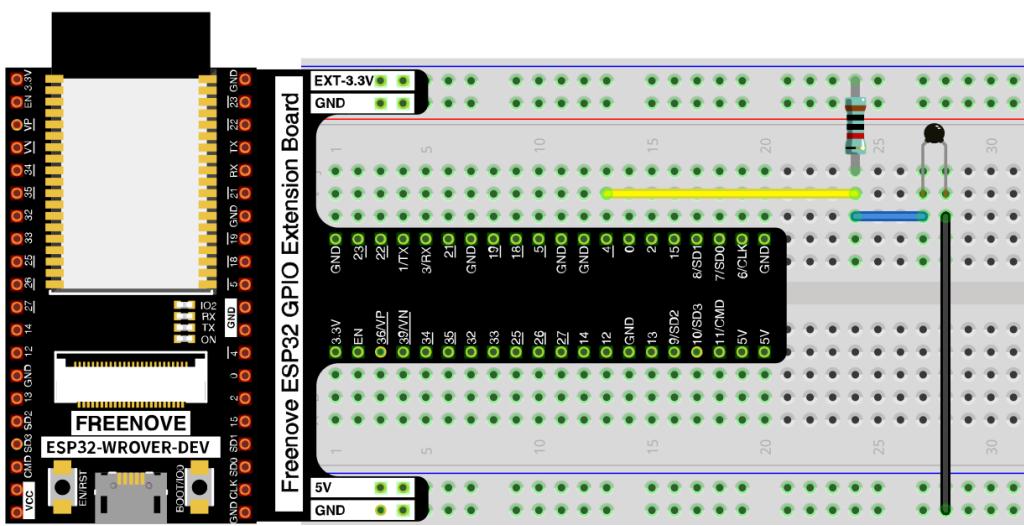
Circuit

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

Schematic diagram



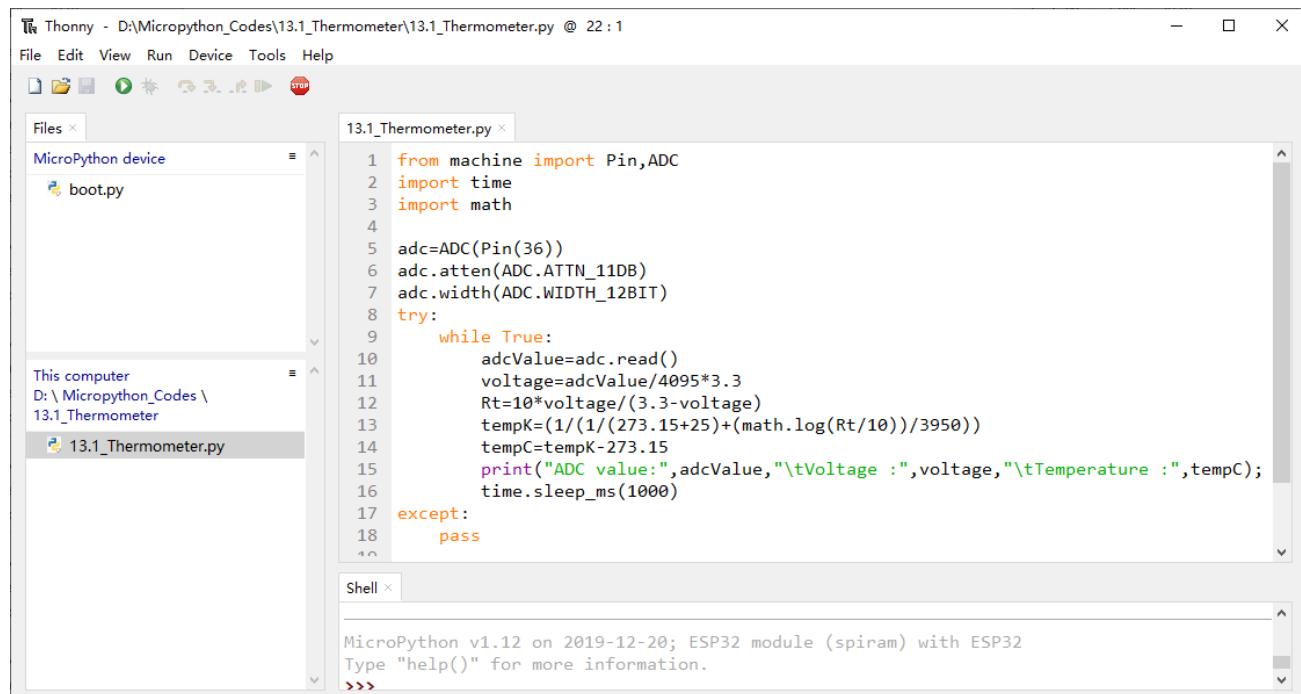
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “13.1_Thermometer” and double click “13.1_Thermometer.py”.

13.1 Thermometer



```

from machine import Pin,ADC
import time
import math

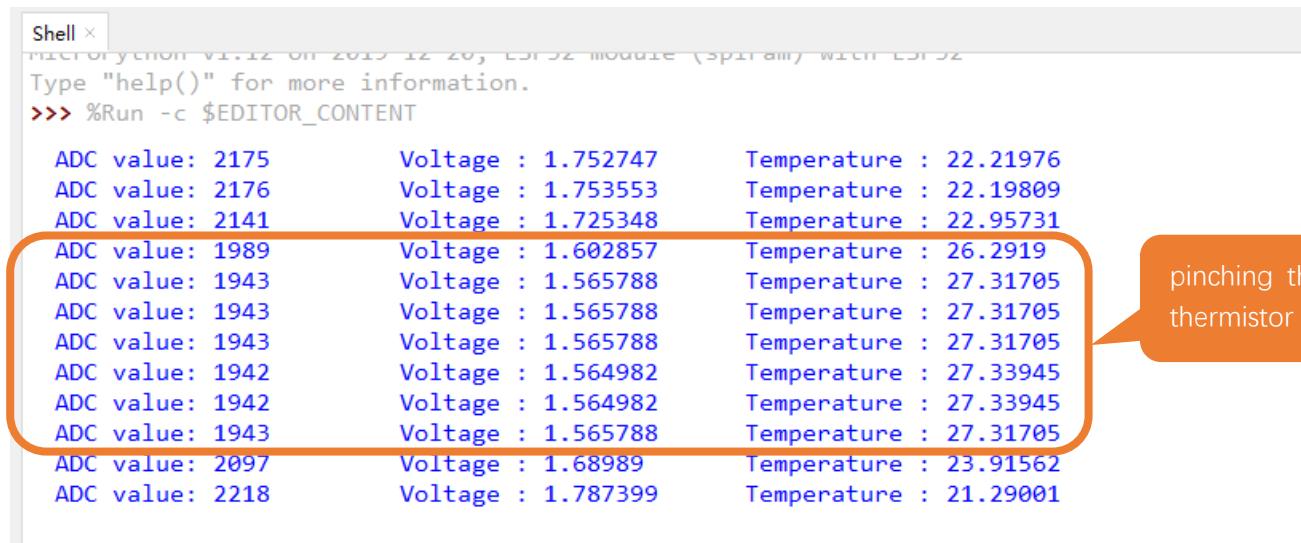
adc=ADC(Pin(36))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

try:
    while True:
        adcValue=adc.read()
        voltage=adcValue/4095*3.3
        Rt=10*voltage/(3.3-voltage)
        tempK=(1/(1/(273.15+25)+(math.log(Rt/10))/3950))
        tempC=tempK-273.15
        print("ADC value:",adcValue,"Voltage : ",voltage,"Temperature : ",tempC)
        time.sleep_ms(1000)
except:
    pass

```

Click “Run current script” and “Shell” will constantly display the current ADC value, voltage value and temperature value. Try to “pinch” the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

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



```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

ADC value: 2175      Voltage : 1.752747      Temperature : 22.21976
ADC value: 2176      Voltage : 1.753553      Temperature : 22.19809
ADC value: 2141      Voltage : 1.725348      Temperature : 22.95731
ADC value: 1989      Voltage : 1.602857      Temperature : 26.2919
ADC value: 1943      Voltage : 1.565788      Temperature : 27.31705
ADC value: 1943      Voltage : 1.565788      Temperature : 27.31705
ADC value: 1943      Voltage : 1.565788      Temperature : 27.31705
ADC value: 1942      Voltage : 1.564982      Temperature : 27.33945
ADC value: 1942      Voltage : 1.564982      Temperature : 27.33945
ADC value: 1943      Voltage : 1.565788      Temperature : 27.31705
ADC value: 2097      Voltage : 1.68989       Temperature : 23.91562
ADC value: 2218      Voltage : 1.787399      Temperature : 21.29001

```



The following is the code:

```
1  from machine import Pin, ADC
2  import time
3  import math
4
5  adc=ADC(Pin(36))
6  adc.atten(ADC.ATTN_11DB)
7  adc.width(ADC.WIDTH_12BIT)
8
9  try:
10     while True:
11         adcValue=adc.read()
12         voltage=adcValue/4095*3.3
13         Rt=10*voltage/(3.3-voltage)
14         tempK=(1/(1/(273.15+25)+(math.log(Rt/10))/3950))
15         tempC=tempK-273.15
16         print("ADC value:",adcValue,"\\tVoltage :",voltage,"\\tTemperature :",tempC);
17         time.sleep_ms(1000)
18     except:
19         pass
```

In the code, the ADC value of ADC module A0 port is read, and then it 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 14 Joystick

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

Project 14.1 Joystick

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

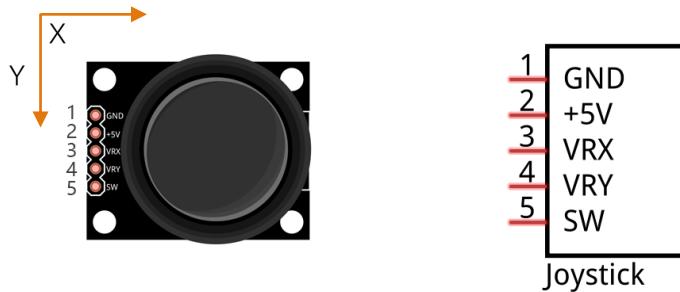
Component List

ESP32-WROVER x1	GPIO Extension Board x1
<img alt="Diagram showing the connection between the ESP32-WROVER and the GPIO Extension Board. The ESP32 pins are connected to the extension board pins as follows: 3.3V to EXT-3.3V, GND to GND, IO2 to 25, IO3 to 26, IO4 to 27, IO5 to 28, IO12 to 14, IO13 to 15, IO14 to 16, IO15 to 17, IO16 to 18, IO17 to 19, IO18 to 20, IO19 to 21, IO20 to 22, IO21 to 23, IO22 to 24, IO23 to 25, IO24 to 26, IO25 to 27, IO26 to 28, IO27 to 29, IO28 to 30, IO29 to 31, IO30 to 32, IO31 to 33, IO32 to 34, IO33 to 35, IO34 to 36, IO35 to 37, IO36 to 38, IO37 to 39, IO38 to 40, IO39 to 41, IO40 to 42, IO41 to 43, IO42 to 44, IO43 to 45, IO44 to 46, IO45 to 47, IO46 to 48, IO47 to 49, IO48 to 50, IO49 to 51, IO50 to 52, IO51 to 53, IO52 to 54, IO53 to 55, IO54 to 56, IO55 to 57, IO56 to 58, IO57 to 59, IO58 to 60, IO59 to 61, IO60 to 62, IO61 to 63, IO62 to 64, IO63 to 65, IO64 to 66, IO65 to 67, IO66 to 68, IO67 to 69, IO68 to 70, IO69 to 71, IO70 to 72, IO71 to 73, IO72 to 74, IO73 to 75, IO74 to 76, IO75 to 77, IO76 to 78, IO77 to 79, IO78 to 80, IO79 to 81, IO80 to 82, IO81 to 83, IO82 to 84, IO83 to 85, IO84 to 86, IO85 to 87, IO86 to 88, IO87 to 89, IO88 to 90, IO89 to 91, IO90 to 92, IO91 to 93, IO92 to 94, IO93 to 95, IO94 to 96, IO95 to 97, IO96 to 98, IO97 to 99, IO98 to 100, IO99 to 101, IO100 to 102, IO101 to 103, IO102 to 104, IO103 to 105, IO104 to 106, IO105 to 107, IO106 to 108, IO107 to 109, IO108 to 110, IO109 to 111, IO110 to 112, IO111 to 113, IO112 to 114, IO113 to 115, IO114 to 116, IO115 to 117, IO116 to 118, IO117 to 119, IO118 to 120, IO119 to 121, IO120 to 122, IO121 to 123, IO122 to 124, IO123 to 125, IO124 to 126, IO125 to 127, IO126 to 128, IO127 to 129, IO128 to 130, IO129 to 131, IO130 to 132, IO131 to 133, IO132 to 134, IO133 to 135, IO134 to 136, IO135 to 137, IO136 to 138, IO137 to 139, IO138 to 140, IO139 to 141, IO140 to 142, IO141 to 143, IO142 to 144, IO143 to 145, IO144 to 146, IO145 to 147, IO146 to 148, IO147 to 149, IO148 to 150, IO149 to 151, IO150 to 152, IO151 to 153, IO152 to 154, IO153 to 155, IO154 to 156, IO155 to 157, IO156 to 158, IO157 to 159, IO158 to 160, IO159 to 161, IO160 to 162, IO161 to 163, IO162 to 164, IO163 to 165, IO164 to 166, IO165 to 167, IO166 to 168, IO167 to 169, IO168 to 170, IO169 to 171, IO170 to 172, IO171 to 173, IO172 to 174, IO173 to 175, IO174 to 176, IO175 to 177, IO176 to 178, IO177 to 179, IO178 to 180, IO179 to 181, IO180 to 182, IO181 to 183, IO182 to 184, IO183 to 185, IO184 to 186, IO185 to 187, IO186 to 188, IO187 to 189, IO188 to 190, IO189 to 191, IO190 to 192, IO191 to 193, IO192 to 194, IO193 to 195, IO194 to 196, IO195 to 197, IO196 to 198, IO197 to 199, IO198 to 200, IO199 to 201, IO200 to 202, IO201 to 203, IO202 to 204, IO203 to 205, IO204 to 206, IO205 to 207, IO206 to 208, IO207 to 209, IO208 to 210, IO209 to 211, IO210 to 212, IO211 to 213, IO212 to 214, IO213 to 215, IO214 to 216, IO215 to 217, IO216 to 218, IO217 to 219, IO218 to 220, IO219 to 221, IO220 to 222, IO221 to 223, IO222 to 224, IO223 to 225, IO224 to 226, IO225 to 227, IO226 to 228, IO227 to 229, IO228 to 230, IO229 to 231, IO230 to 232, IO231 to 233, IO232 to 234, IO233 to 235, IO234 to 236, IO235 to 237, IO236 to 238, IO237 to 239, IO238 to 240, IO239 to 241, IO240 to 242, IO241 to 243, IO242 to 244, IO243 to 245, IO244 to 246, IO245 to 247, IO246 to 248, IO247 to 249, IO248 to 250, IO249 to 251, IO250 to 252, IO251 to 253, IO252 to 254, IO253 to 255, IO254 to 256, IO255 to 257, IO256 to 258, IO257 to 259, IO258 to 260, IO259 to 261, IO260 to 262, IO261 to 263, IO262 to 264, IO263 to 265, IO264 to 266, IO265 to 267, IO266 to 268, IO267 to 269, IO268 to 270, IO269 to 271, IO270 to 272, IO271 to 273, IO272 to 274, IO273 to 275, IO274 to 276, IO275 to 277, IO276 to 278, IO277 to 279, IO278 to 280, IO279 to 281, IO280 to 282, IO281 to 283, IO282 to 284, IO283 to 285, IO284 to 286, IO285 to 287, IO286 to 288, IO287 to 289, IO288 to 290, IO289 to 291, IO290 to 292, IO291 to 293, IO292 to 294, IO293 to 295, IO294 to 296, IO295 to 297, IO296 to 298, IO297 to 299, IO298 to 300, IO299 to 301, IO300 to 302, IO301 to 303, IO302 to 304, IO303 to 305, IO304 to 306, IO305 to 307, IO306 to 308, IO307 to 309, IO308 to 310, IO309 to 311, IO310 to 312, IO311 to 313, IO312 to 314, IO313 to 315, IO314 to 316, IO315 to 317, IO316 to 318, IO317 to 319, IO318 to 320, IO319 to 321, IO320 to 322, IO321 to 323, IO322 to 324, IO323 to 325, IO324 to 326, IO325 to 327, IO326 to 328, IO327 to 329, IO328 to 330, IO329 to 331, IO330 to 332, IO331 to 333, IO332 to 334, IO333 to 335, IO334 to 336, IO335 to 337, IO336 to 338, IO337 to 339, IO338 to 340, IO339 to 341, IO340 to 342, IO341 to 343, IO342 to 344, IO343 to 345, IO344 to 346, IO345 to 347, IO346 to 348, IO347 to 349, IO348 to 350, IO349 to 351, IO350 to 352, IO351 to 353, IO352 to 354, IO353 to 355, IO354 to 356, IO355 to 357, IO356 to 358, IO357 to 359, IO358 to 360, IO359 to 361, IO360 to 362, IO361 to 363, IO362 to 364, IO363 to 365, IO364 to 366, IO365 to 367, IO366 to 368, IO367 to 369, IO368 to 370, IO369 to 371, IO370 to 372, IO371 to 373, IO372 to 374, IO373 to 375, IO374 to 376, IO375 to 377, IO376 to 378, IO377 to 379, IO378 to 380, IO379 to 381, IO380 to 382, IO381 to 383, IO382 to 384, IO383 to 385, IO384 to 386, IO385 to 387, IO386 to 388, IO387 to 389, IO388 to 390, IO389 to 391, IO390 to 392, IO391 to 393, IO392 to 394, IO393 to 395, IO394 to 396, IO395 to 397, IO396 to 398, IO397 to 399, IO398 to 400, IO399 to 401, IO400 to 402, IO401 to 403, IO402 to 404, IO403 to 405, IO404 to 406, IO405 to 407, IO406 to 408, IO407 to 409, IO408 to 410, IO409 to 411, IO410 to 412, IO411 to 413, IO412 to 414, IO413 to 415, IO414 to 416, IO415 to 417, IO416 to 418, IO417 to 419, IO418 to 420, IO419 to 421, IO420 to 422, IO421 to 423, IO422 to 424, IO423 to 425, IO424 to 426, IO425 to 427, IO426 to 428, IO427 to 429, IO428 to 430, IO429 to 431, IO430 to 432, IO431 to 433, IO432 to 434, IO433 to 435, IO434 to 436, IO435 to 437, IO436 to 438, IO437 to 439, IO438 to 440, IO439 to 441, IO440 to 442, IO441 to 443, IO442 to 444, IO443 to 445, IO444 to 446, IO445 to 447, IO446 to 448, IO447 to 449, IO448 to 450, IO449 to 451, IO450 to 452, IO451 to 453, IO452 to 454, IO453 to 455, IO454 to 456, IO455 to 457, IO456 to 458, IO457 to 459, IO458 to 460, IO459 to 461, IO460 to 462, IO461 to 463, IO462 to 464, IO463 to 465, IO464 to 466, IO465 to 467, IO466 to 468, IO467 to 469, IO468 to 470, IO469 to 471, IO470 to 472, IO471 to 473, IO472 to 474, IO473 to 475, IO474 to 476, IO475 to 477, IO476 to 478, IO477 to 479, IO478 to 480, IO479 to 481, IO480 to 482, IO481 to 483, IO482 to 484, IO483 to 485, IO484 to 486, IO485 to 487, IO486 to 488, IO487 to 489, IO488 to 490, IO489 to 491, IO490 to 492, IO491 to 493, IO492 to 494, IO493 to 495, IO494 to 496, IO495 to 497, IO496 to 498, IO497 to 499, IO498 to 500, IO499 to 501, IO500 to 502, IO501 to 503, IO502 to 504, IO503 to 505, IO504 to 506, IO505 to 507, IO506 to 508, IO507 to 509, IO508 to 510, IO509 to 511, IO510 to 512, IO511 to 513, IO512 to 514, IO513 to 515, IO514 to 516, IO515 to 517, IO516 to 518, IO517 to 519, IO518 to 520, IO519 to 521, IO520 to 522, IO521 to 523, IO522 to 524, IO523 to 525, IO524 to 526, IO525 to 527, IO526 to 528, IO527 to 529, IO528 to 530, IO529 to 531, IO530 to 532, IO531 to 533, IO532 to 534, IO533 to 535, IO534 to 536, IO535 to 537, IO536 to 538, IO537 to 539, IO538 to 540, IO539 to 541, IO540 to 542, IO541 to 543, IO542 to 544, IO543 to 545, IO544 to 546, IO545 to 547, IO546 to 548, IO547 to 549, IO548 to 550, IO549 to 551, IO550 to 552, IO551 to 553, IO552 to 554, IO553 to 555, IO554 to 556, IO555 to 557, IO556 to 558, IO557 to 559, IO558 to 560, IO559 to 561, IO560 to 562, IO561 to 563, IO562 to 564, IO563 to 565, IO564 to 566, IO565 to 567, IO566 to 568, IO567 to 569, IO568 to 570, IO569 to 571, IO570 to 572, IO571 to 573, IO572 to 574, IO573 to 575, IO574 to 576, IO575 to 577, IO576 to 578, IO577 to 579, IO578 to 580, IO579 to 581, IO580 to 582, IO581 to 583, IO582 to 584, IO583 to 585, IO584 to 586, IO585 to 587, IO586 to 588, IO587 to 589, IO588 to 590, IO589 to 591, IO590 to 592, IO591 to 593, IO592 to 594, IO593 to 595, IO594 to 596, IO595 to 597, IO596 to 598, IO597 to 599, IO598 to 600, IO599 to 601, IO600 to 602, IO601 to 603, IO602 to 604, IO603 to 605, IO604 to 606, IO605 to 607, IO606 to 608, IO607 to 609, IO608 to 610, IO609 to 611, IO610 to 612, IO611 to 613, IO612 to 614, IO613 to 615, IO614 to 616, IO615 to 617, IO616 to 618, IO617 to 619, IO618 to 620, IO619 to 621, IO620 to 622, IO621 to 623, IO622 to 624, IO623 to 625, IO624 to 626, IO625 to 627, IO626 to 628, IO627 to 629, IO628 to 630, IO629 to 631, IO630 to 632, IO631 to 633, IO632 to 634, IO633 to 635, IO634 to 636, IO635 to 637, IO636 to 638, IO637 to 639, IO638 to 640, IO639 to 641, IO640 to 642, IO641 to 643, IO642 to 644, IO643 to 645, IO644 to 646, IO645 to 647, IO646 to 648, IO647 to 649, IO648 to 650, IO649 to 651, IO650 to 652, IO651 to 653, IO652 to 654, IO653 to 655, IO654 to 656, IO655 to 657, IO656 to 658, IO657 to 659, IO658 to 660, IO659 to 661, IO660 to 662, IO661 to 663, IO662 to 664, IO663 to 665, IO664 to 666, IO665 to 667, IO666 to 668, IO667 to 669, IO668 to 670, IO669 to 671, IO670 to 672, IO671 to 673, IO672 to 674, IO673 to 675, IO674 to 676, IO675 to 677, IO676 to 678, IO677 to 679, IO678 to 680, IO679 to 681, IO680 to 682, IO681 to 683, IO682 to 684, IO683 to 685, IO684 to 686, IO685 to 687, IO686 to 688, IO687 to 689, IO688 to 690, IO689 to 691, IO690 to 692, IO691 to 693, IO692 to 694, IO693 to 695, IO694 to 696, IO695 to 697, IO696 to 698, IO697 to 699, IO698 to 700, IO699 to 701, IO700 to 702, IO701 to 703, IO702 to 704, IO703 to 705, IO704 to 706, IO705 to 707, IO706 to 708, IO707 to 709, IO708 to 710, IO709 to 711, IO710 to 712, IO711 to 713, IO712 to 714, IO713 to 715, IO714 to 716, IO715 to 717, IO716 to 718, IO717 to 719, IO718 to 720, IO719 to 721, IO720 to 722, IO721 to 723, IO722 to 724, IO723 to 725, IO724 to 726, IO725 to 727, IO726 to 728, IO727 to 729, IO728 to 730, IO729 to 731, IO730 to 732, IO731 to 733, IO732 to 734, IO733 to 735, IO734 to 736, IO735 to 737, IO736 to 738, IO737 to 739, IO738 to 740, IO739 to 741, IO740 to 742, IO741 to 743, IO742 to 744, IO743 to 745, IO744 to 746, IO745 to 747, IO746 to 748, IO747 to 749, IO748 to 750, IO749 to 751, IO750 to 752, IO751 to 753, IO752 to 754, IO753 to 755, IO754 to 756, IO755 to 757, IO756 to 758, IO757 to 759, IO758 to 760, IO759 to 761, IO760 to 762, IO761 to 763, IO762 to 764, IO763 to 765, IO764 to 766, IO765 to 767, IO766 to 768, IO767 to 769, IO768 to 770, IO769 to 771, IO770 to 772, IO771 to 773, IO772 to 774, IO773 to 775, IO774 to 776, IO775 to 777, IO776 to 778, IO777 to 779, IO778 to 780, IO779 to 781, IO780 to 782, IO781 to 783, IO782 to 784, IO783 to 785, IO784 to 786, IO785 to 787, IO786 to 788, IO787 to 789, IO788 to 790, IO789 to 791, IO790 to 792, IO791 to 793, IO792 to 794, IO793 to 795, IO794 to 796, IO795 to 797, IO796 to 798, IO797 to 799, IO798 to 800, IO799 to 801, IO800 to 802, IO801 to 803, IO802 to 804, IO803 to 805, IO804 to 806, IO805 to 807, IO806 to 808, IO807 to 809, IO808 to 810, IO809 to 811, IO810 to 812, IO811 to 813, IO812 to 814, IO813 to 815, IO814 to 816, IO815 to 817, IO816 to 818, IO817 to 819, IO818 to 820, IO819 to 821, IO820 to 822, IO821 to 823, IO822 to 824, IO823 to 825, IO824 to 826, IO825 to 827, IO826 to 828, IO827 to 829, IO828 to 830, IO829 to 831, IO830 to 832, IO831 to 833, IO832 to 834, IO833 to 835, IO834 to 836, IO835 to 837, IO836 to 838, IO837 to 839, IO838 to 840, IO839 to 841, IO840 to 842, IO841 to 843, IO842 to 844, IO843 to 845, IO844 to 846, IO845 to 847, IO846 to 848, IO847 to 849, IO848 to 850, IO849 to 851, IO850 to 852, IO851 to 853, IO852 to 854, IO853 to 855, IO854 to 856, IO855 to 857, IO856 to 858, IO857 to 859, IO858 to 860, IO859 to 861, IO860 to 862, IO861 to 863, IO862 to 864, IO863 to 865, IO864 to 866, IO865 to 867, IO866 to 868, IO867 to 869, IO868 to 870, IO869 to 871, IO870 to 872, IO871 to 873, IO872 to 874, IO873 to 875, IO874 to 876, IO875 to 877, IO876 to 878, IO877 to 879, IO878 to 880, IO879 to 881, IO880 to 882, IO881 to 883, IO882 to 884, IO883 to 885, IO884 to 886, IO885 to 887, IO886 to 888, IO887 to 889, IO888 to 890, IO889 to 891, IO890 to 892, IO891 to 893, IO892 to 894, IO893 to 895, IO894 to 896, IO895 to 897, IO896 to 898, IO897 to 899, IO898 to 900, IO899 to 901, IO900 to 902, IO901 to 903, IO902 to 904, IO903 to 905, IO904 to 906, IO905 to 907, IO906 to 908, IO907 to 909, IO908 to 910, IO909 to 911, IO910 to 912, IO911 to 913, IO912 to 914, IO913 to 915, IO914 to 916, IO915 to 917, IO916 to 918, IO917 to 919, IO918 to 920, IO919 to 921, IO920 to 922, IO921 to 923, IO922 to 924, IO923 to 925, IO924 to 926, IO925 to 927, IO926 to 928, IO927 to 929, IO928 to 930, IO929 to 931, IO930 to 932, IO931 to 933, IO932 to 934, IO933 to 935, IO934 to 936, IO935 to 937, IO936 to 938, IO937 to 939, IO938 to 940, IO939 to 941, IO940 to 942, IO941 to 943, IO942 to 944, IO943 to 945, IO944 to 946, IO945 to 947, IO946 to 948, IO947 to 949, IO948 to 950, IO949 to 951, IO950 to 952, IO951 to 953, IO952 to 954, IO953 to 955, IO954 to 956, IO955 to 957, IO956 to 958, IO957 to 959, IO958 to 960, IO959 to 961, IO960 to 962, IO961 to 963, IO962 to 964, IO963 to 965, IO964 to 966, IO965 to 967, IO966 to 968, IO967 to 969, IO968 to 970, IO969 to 971, IO970 to 972, IO971 to 973, IO972 to 974, IO973 to 975, IO974 to 976, IO975 to 977, IO976 to 978, IO977 to 979, IO978 to 980, IO979 to 981, IO980 to 982, IO981 to 983, IO982 to 984, IO983 to 985, IO984 to 986, IO985 to 987, IO986 to 988, IO987 to 989, IO988 to 990, IO989 to 991, IO990 to 992, IO991 to 993, IO992 to 994, IO993 to 995, IO994 to 996, IO995 to 997, IO996 to 998, IO997 to 999, IO998 to 1000, IO999 to 1001, IO1000 to 1002, IO1001 to 1003, IO1002 to 1004, IO1003 to 1005, IO1004 to 1006, IO1005 to 1007, IO1006 to 1008, IO1007 to 1009, IO1008 to 1010, IO1009 to 1011, IO1010 to 1012, IO1011 to 1013, IO1012 to 1014, IO1013 to 1015, IO1014 to 1016, IO1015 to 1017, IO1016 to 1018, IO1017 to 1019, IO1018 to 1020, IO1019 to 1021, IO1020 to 1022, IO1021 to 1023, IO1022 to 1024, IO1023 to 1025, IO1024 to 1026, IO1025 to 1027, IO1026 to 1028, IO1027 to 1029, IO1028 to 1030, IO1029 to 1031, IO1030 to 1032, IO1031 to 1033, IO1032 to 1034, IO1033 to 1035, IO1034 to 1036, IO1035 to 1037, IO1036 to 1038, IO1037 to 1039, IO1038 to 1040, IO1039 to 1041, IO1040 to 1042, IO1041 to 1043, IO1042 to 1044, IO1043 to 1045, IO1044 to 1046, IO1045 to 1047, IO1046 to 1048, IO1047 to 1049, IO1048 to 1050, IO1049 to 1051, IO1050 to 1052, IO1051 to 1053, IO1052 to 1054, IO1053 to 1055, IO1054 to 1056, IO1055 to 1057, IO1056 to 1058, IO1057 to 1059, IO1058 to 1060, IO1059 to 1061, IO1060 to 1062, IO1061 to 1063, IO1062 to 1064, IO1063 to 1065, IO1064 to 1066, IO1065 to 1067, IO1066 to 1068, IO1067 to 1069, IO1068 to 1070,	

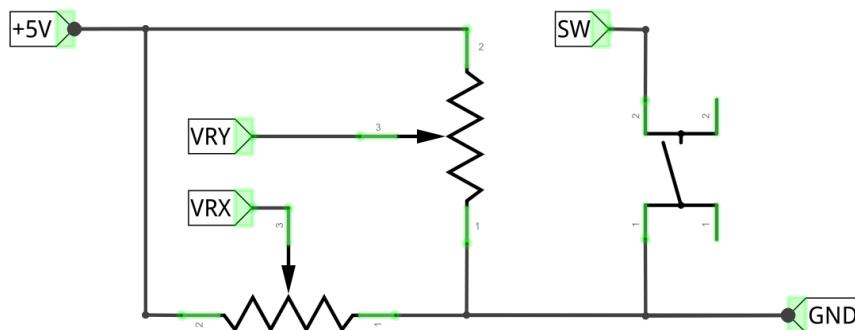
Component knowledge

Joystick

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



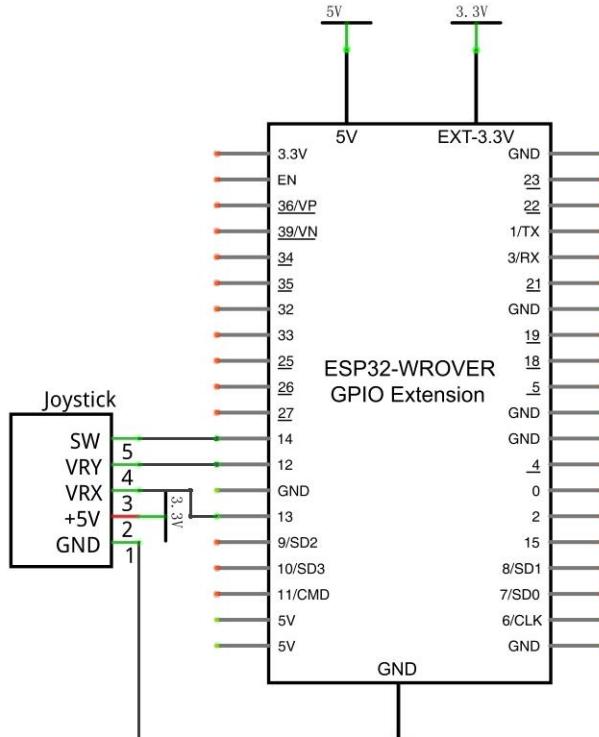
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



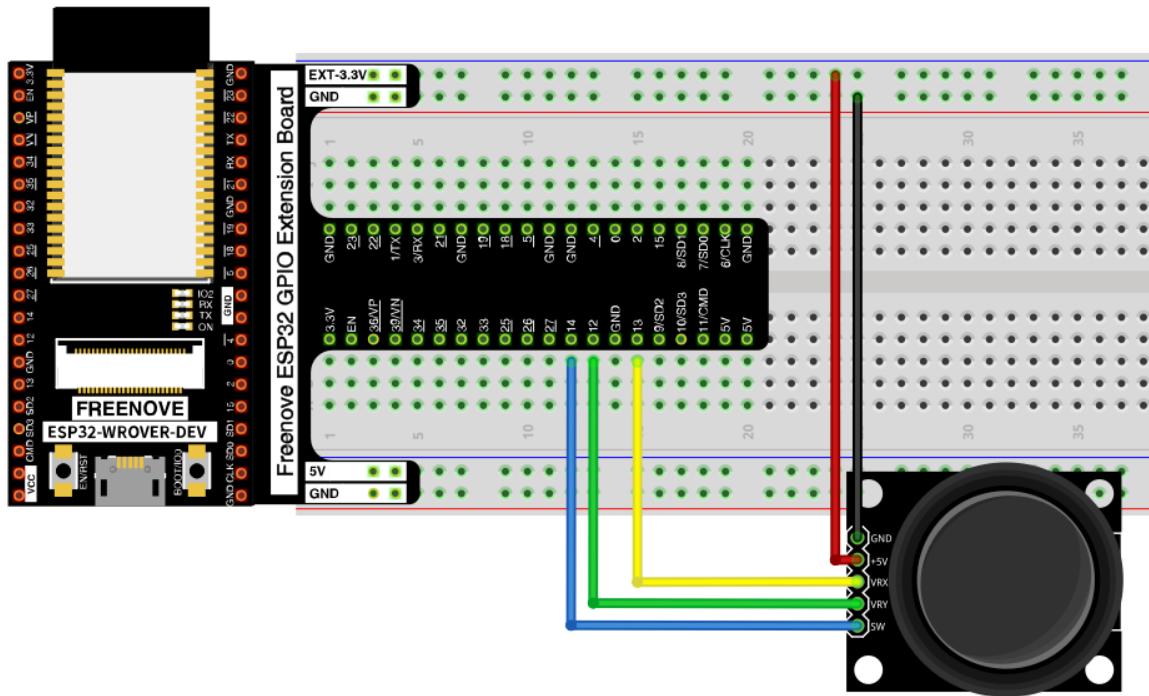
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



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





Code

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

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.1_Joystick" and double click "14.1_Joystick.py".

14.1 Joystick

```

  Thonny - D:\Micropython_Codes\14.1_Joystick\14.1_Joystick.py @ 21 : 3
File Edit View Run Device Tools Help
Files x 14.1_Joystick.py x
MicroPython device
boot.py
This computer
D: \ Micropython_Codes \ 14.1_Joystick
14.1_Joystick.py
1   from machine import ADC,Pin
2   import time
3
4   xVal=ADC(Pin(36))
5   yVal=ADC(Pin(39))
6   zVal=Pin(14,Pin.IN,Pin.PULL_UP)
7
8   xVal.atten(ADC.ATTN_11DB)
9   yVal.atten(ADC.ATTN_11DB)
10  xVal.width(ADC.WIDTH_12BIT)
11  yVal.width(ADC.WIDTH_12BIT)
12
13 while True:
14     print("X,Y,Z:",xVal.read()," ",yVal.read()," ",zVal.value())
15     time.sleep(0.1)
16
17
18
19
  Shell x
  MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
  Type "help()" for more information.
  >>>

```

Click "Run current script". Shifting the Joystick or pressing it down will make the printed data in "Shell" change.

```

  Shell x
  MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
  Type "help()" for more information.
  >>> %Run -c $EDITOR_CONTENT
  X,Y,Z: 1802 , 503 , 1
  X,Y,Z: 2255 , 1130 , 1
  X,Y,Z: 1781 , 61 , 1
  X,Y,Z: 2519 , 0 , 1
  X,Y,Z: 3021 , 0 , 1
  X,Y,Z: 0 , 1873 , 1
  X,Y,Z: 0 , 1872 , 1
  X,Y,Z: 1781 , 1871 , 0
  X,Y,Z: 1783 , 1872 , 0
  X,Y,Z: 1783 , 1872 , 1
  Shifting Y axis
  Shifting X axis
  Pressing Z axis

```

The flowing is the code:

```
1 from machine import ADC, Pin
2 import time
3
4 xVal=ADC(Pin(36))
5 yVal=ADC(Pin(39))
6 zVal=Pin(14, Pin.IN, Pin.PULL_UP)
7
8 xVal.atten(ADC.ATTN_11DB)
9 yVal.atten(ADC.ATTN_11DB)
10 xVal.width(ADC.WIDTH_12BIT)
11 yVal.width(ADC.WIDTH_12BIT)
12
13 while True:
14     print("X, Y, Z:", xVal.read(), ", ", yVal.read(), ", ", zVal.value())
15     time.sleep(1)
```

Set the acquisition range of voltage of the two ADC channels to 0-3.3V, and the acquisition width of data to 0-4095.

```
8 xVal.atten(ADC.ATTN_11DB)
9 yVal.atten(ADC.ATTN_11DB)
10 xVal.width(ADC.WIDTH_12BIT)
11 yVal.width(ADC.WIDTH_12BIT)
```

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

```
14 print("X, Y, Z:", xVal.read(), ", ", yVal.read(), ", ", zVal.value())
```

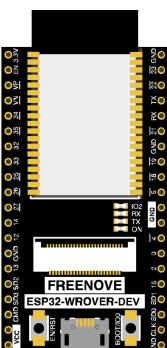
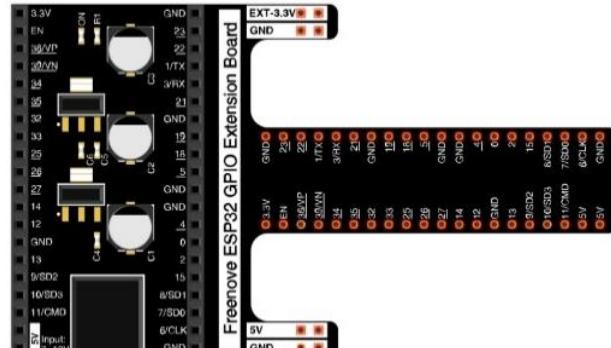
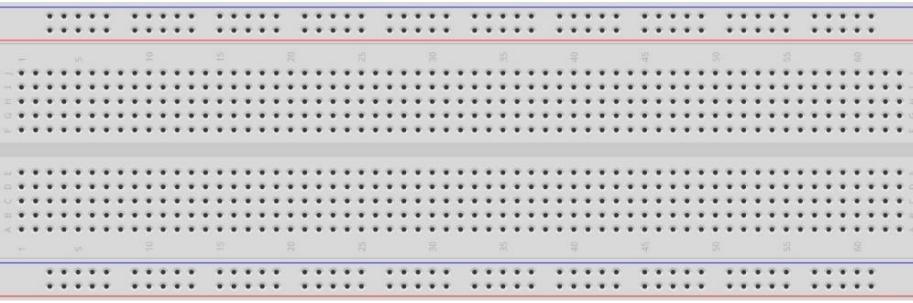
Chapter 15 74HC595 & LED Bar Graph

We have used LED Bar Graph to make a flowing water light, in which 10 GPIO ports of ESP32 is occupied. More GPIO ports mean that more peripherals can be connected to ESP32, so GPIO resource is very precious. Can we make flowing water light with less GPIO? In this chapter, we will learn a component, 74HC595, which can achieve the target.

Project 15.1 Flowing Water Light

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

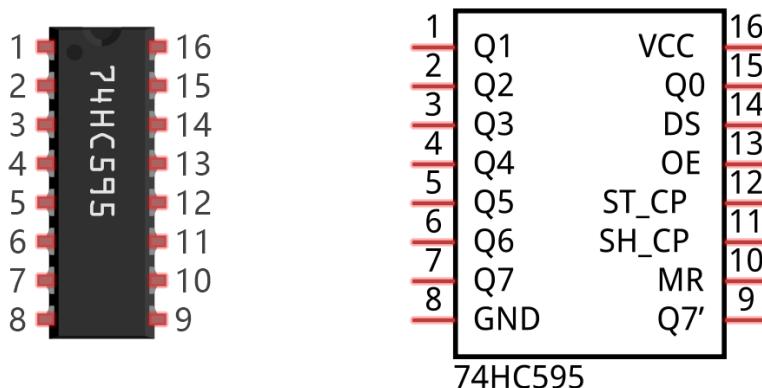
Component List

ESP32-WROVER x1	GPIO Extension Board x1	Breadboard x1	74HC595 x1	LED Bar Graph x1	Resistor 220Ω x8	Jumper M/M x15
						

Related knowledge

74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of an ESP32. At least 3 ports are required to control the 8 ports of the 74HC595 chip.



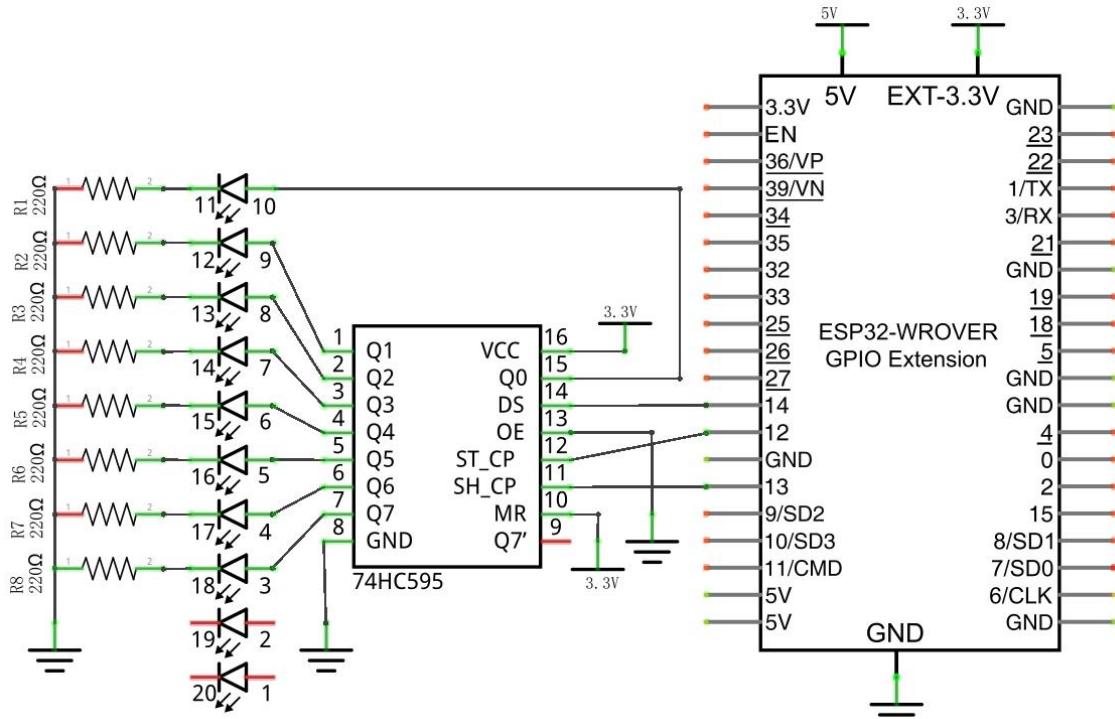
The ports of the 74HC595 chip are described as follows:

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

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

Circuit

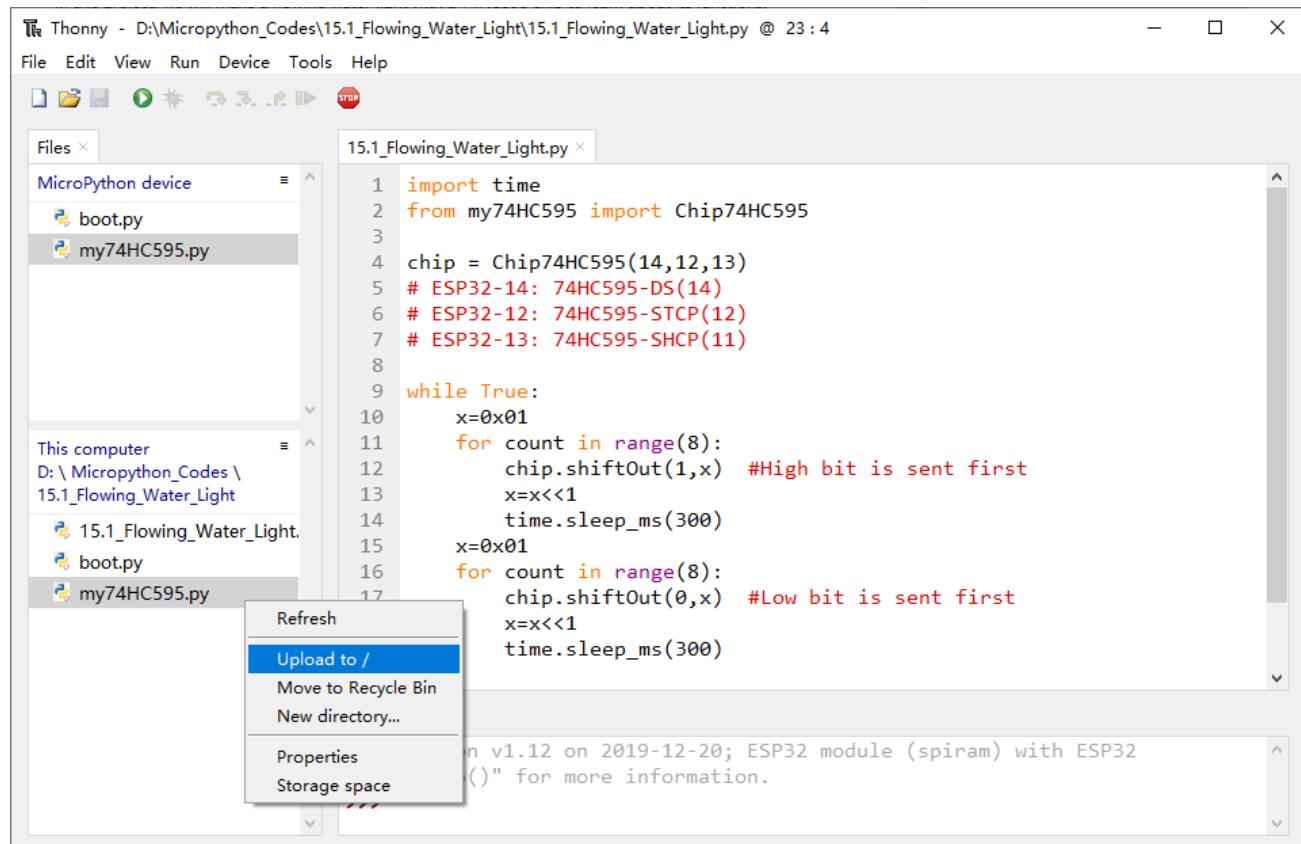
Schematic diagram



Code

In this project, we will make a flowing water light with a 74HC595 chip to learn about its functions. Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “15.1_Flowing_Water_Light”. Select “my74HC595.py”, right click your mouse to select “Upload to /”, wait for “my74HC595.py” to be uploaded to ESP32-WROVER and then double click “15.1_Flowing_Water_Light.py”.

15.1 Flowing Water Light



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\15.1_Flowing_Water_Light\15.1_Flowing_Water_Light.py @ 23 : 4". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations. The left sidebar shows a "Files" tree with "MicroPython device" expanded, listing "boot.py" and "my74HC595.py". The main editor window displays the Python code for "15.1_Flowing_Water_Light.py". A context menu is open over "my74HC595.py", with "Upload to /" highlighted in blue. The status bar at the bottom indicates "v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32 ()" for more information.

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 chip = Chip74HC595(14,12,13)
5 # ESP32-14: 74HC595-DS(14)
6 # ESP32-12: 74HC595-STCP(12)
7 # ESP32-13: 74HC595-SHCP(11)
8
9 while True:
10     x=0x01
11     for count in range(8):
12         chip.shiftOut(1,x) #High bit is sent first
13         x=x<<1
14         time.sleep_ms(300)
15     x=0x01
16     for count in range(8):
17         chip.shiftOut(0,x) #Low bit is sent first
18         x=x<<1
19         time.sleep_ms(300)
```

Click “Run current script” and you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left. If it displays nothing, maybe the LED Bar is connected upside down, please unplug it and then re-plug it reversely.

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

The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 chip = Chip74HC595(14, 12, 13)
# ESP32-14: 74HC595-DS (14)
# ESP32-12: 74HC595-STCP (12)
# ESP32-13: 74HC595-SHCP (11)
5
6
7
8
9 while True:
10     x=0x01
11     for count in range(8):
12         chip.shiftOut(1, x) #High bit is sent first
13         x=x<<1
14         time.sleep_ms(300)
15     x=0x01
16     for count in range(8):
17         chip.shiftOut(0, x) #Low bit is sent first
18         x=x<<1
19         time.sleep_ms(300)
```

Import time and my74HC595 modules.

```
1 import time
2 from my74HC595 import Chip74HC595
```

Assign pins for ESP32-WROVER to connect to 74HC595.

```
4 chip = Chip74HC595(14, 12, 13)
```

The first for loop makes LED Bar display separately from left to right while the second for loop make it display separately from right to left.

```
10 x=0x01
11 for count in range(8):
12     chip.shiftOut(1, x) #High bit is sent first
13     x=x<<1
14     time.sleep_ms(300)
15 x=0x01
16 for count in range(8):
17     chip.shiftOut(0, x) #Low bit is sent first
18     x=x<<1
19     time.sleep_ms(300)
```

Reference

Class Chip74HC595

Before each use of the object **Chip74HC595**, make sure my74HC595.py has been uploaded to "/" of ESP32, and then add the statement "**from my74HC595 import Chip74HC595**" to the top of the python file.

Chip74HC595():An object. By default, 74HC595's DS pin is connected to Pin(14) of ESP32, ST_CP pin is connected to ESP32's Pin(12) and OE pin is connected to ESP's Pin(5). If you need to modify the pins, just do the following operations.

chip=Chip74HC595() or **chip=Chip74HC595(14,12,13,5)**

shiftOut(direction, data): Write data to 74HC595.

direction: 1/0. "1" presents that high-order byte will be sent first while "0" presents that low-order byte will be sent first.

data: The content that is sent, which is one-byte data.

clear(): Clear the latch data of 74HC595..

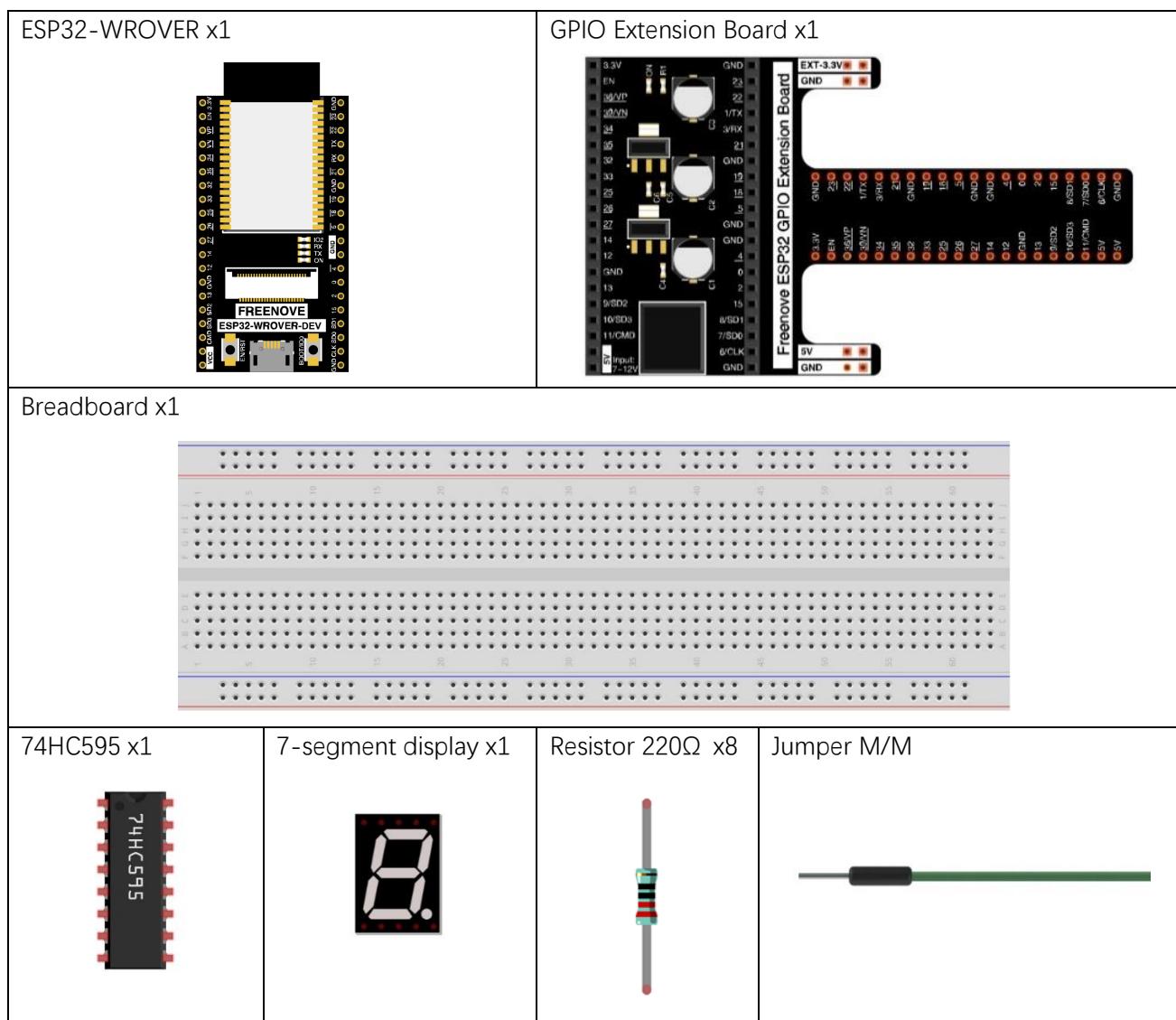
Chapter 16 74HC595 & 7-Segment Display.

In this chapter, we will introduce the 7-Segment Display.

Project 16.1 7-Segment Display.

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

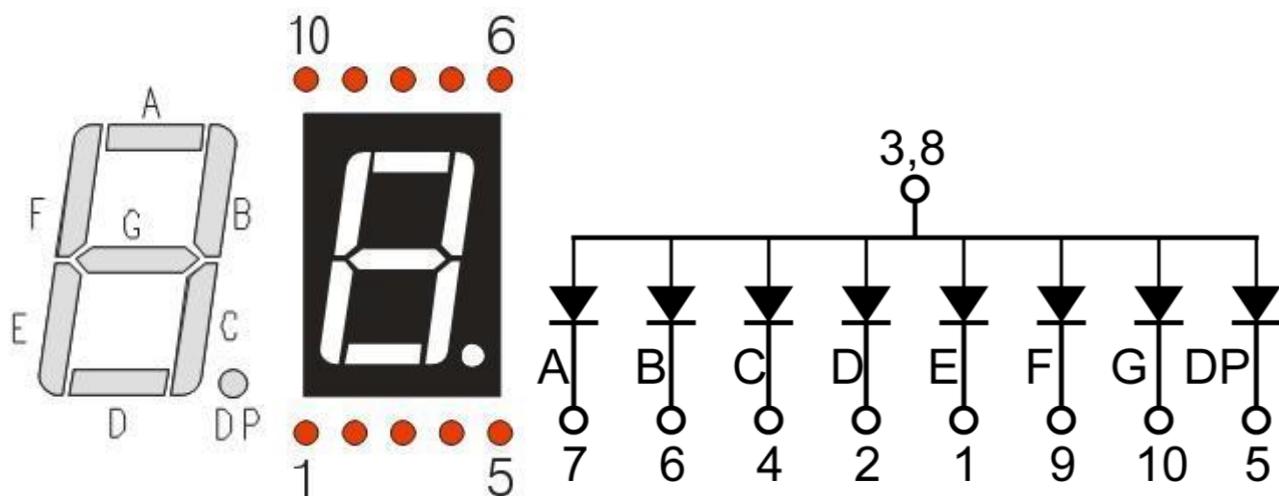
Component List



Component knowledge

7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



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

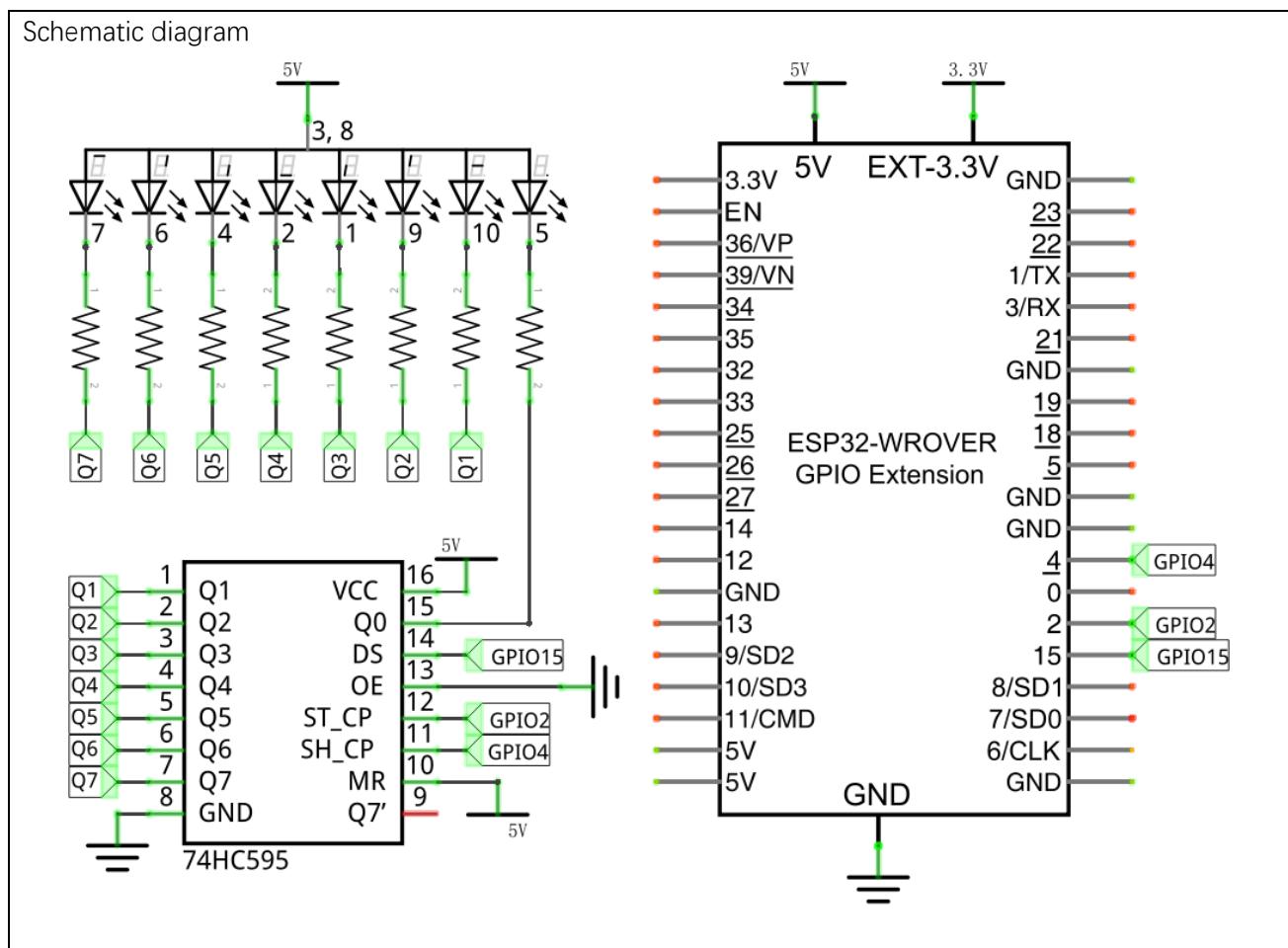


In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment "A" as the lowest level and segment "DP" as the highest level, from high to low would look like this: "DP", "G", "F", "E", "D", "C", "B", "A". Character "0" corresponds to the code: $1100\ 0000_2 = 0xc0$.

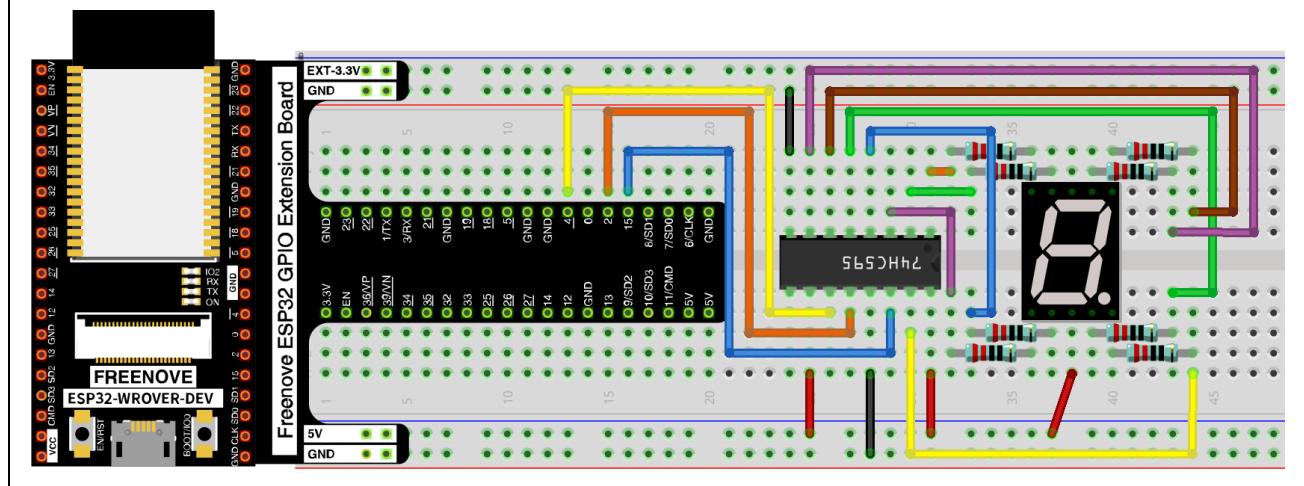
For detailed code values, please refer to the following table (common anode).

CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0	0	0	0	0	0	0	0x80	1000 0000
9	1	0	0	1	0	0	0	0	0x90	1001 0000
A	1	0	0	0	1	0	0	0	0x88	1000 1000
B	1	0	0	0	0	0	1	1	0x83	1000 0011
C	1	1	0	0	0	1	1	0	0xc6	1100 0110
D	1	0	1	0	0	0	0	1	0xa1	1010 0001
E	1	0	0	0	0	1	1	0	0x86	1000 0110
F	1	0	0	0	1	1	1	0	0x8e	1000 1110

Circuit



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

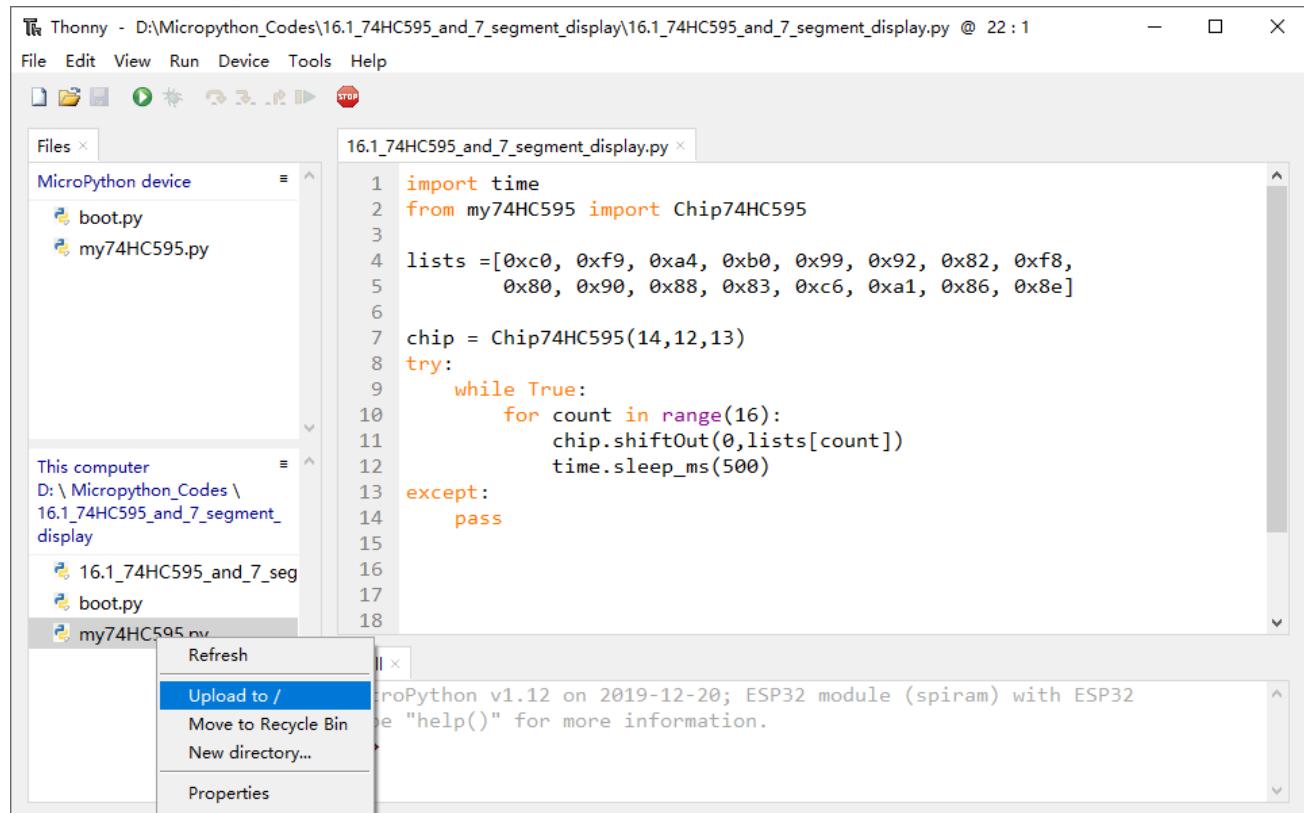


Code

In this section, the 74HC595 is used in the same way as in the previous section, but with different values transferred. We can learn how to master the digital display by sending the code value of "0" - "F".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "16.1_74HC595_and_7_segment_display". Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to ESP32-WROVER and then double click "16.1_74HC595_and_7_segment_display.py".

16.1 74HC595 and 7 segment_display

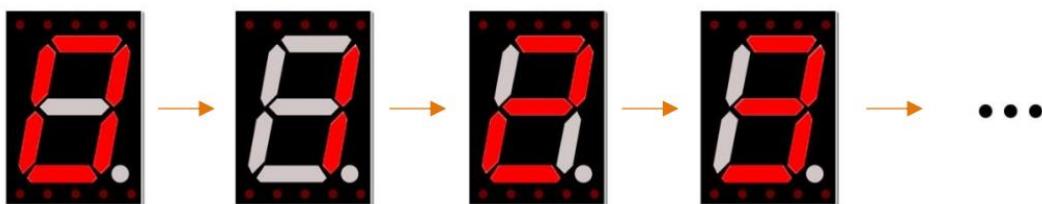


```

Thonny - D:\Micropython_Codes\16.1_74HC595_and_7_segment_display\16.1_74HC595_and_7_segment_display.py @ 22 : 1
File Edit View Run Device Tools Help
File folder Go Stop
Files x
MicroPython device
boot.py
my74HC595.py
This computer
D:\Micropython_Codes\
16.1_74HC595_and_7_segment_
display
16.1_74HC595_and_7_seg
boot.py
my74HC595.py
Refresh
Upload to /
Move to Recycle Bin
New directory...
Properties
1 import time
2 from my74HC595 import Chip74HC595
3
4 lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
6
7 chip = Chip74HC595(14,12,13)
8 try:
9     while True:
10        for count in range(16):
11            chip.shiftOut(0,lists[count])
12            time.sleep_ms(500)
13    except:
14        pass
15
16
17
18
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Use "help()" for more information.

```

Click "Run current script", and you'll see a 1-bit, 7-segment display displaying 0-f in a loop.



The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
6
7 chip = Chip74HC595(14, 12, 13)
8 try:
9     while True:
10        for count in range(16):
11            chip.shiftOut(0, lists[count])
12            time.sleep_ms(500)
13    except:
14        pass
```

Import time and my74HC595 modules.

```
1 import time
2 from my74HC595 import Chip74HC595
```

Put the encoding "0" - "F" into the list.

```
4 lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
```

Define an object, whose pins applies default configuration, to drive 74HC595.

```
7 chip = Chip74HC595(14, 12, 13)
```

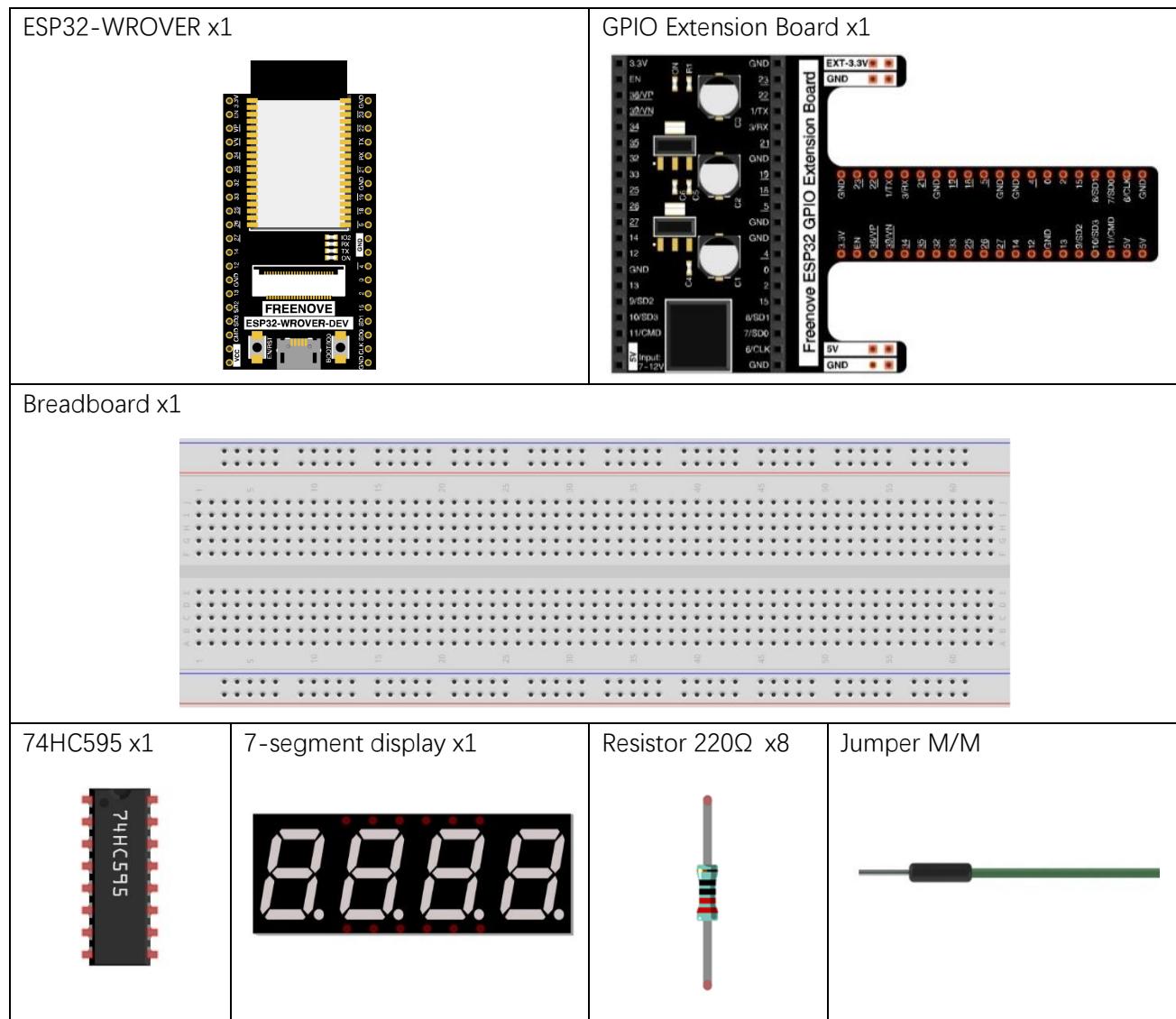
Send data of digital tube to 74HC595 chip.

```
11 chip.shiftOut(0, lists[count])
```

Project 16.2 4-Digit 7-Segment Display

Now, let's try to control a more-digit 7-segment display

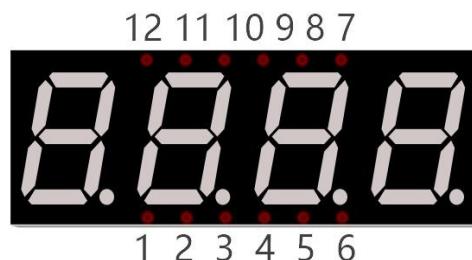
Component List



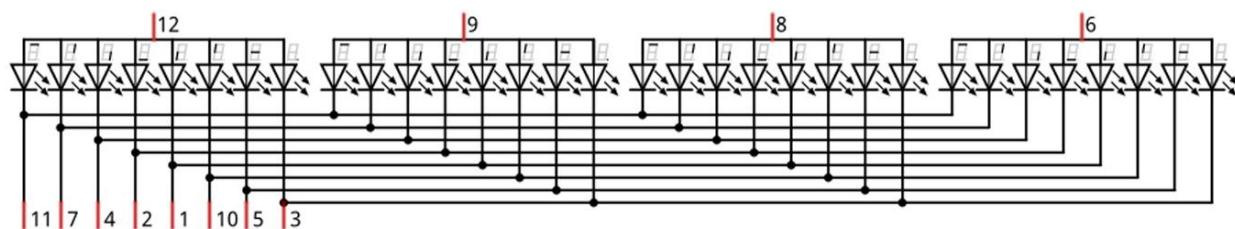
Component knowledge

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.

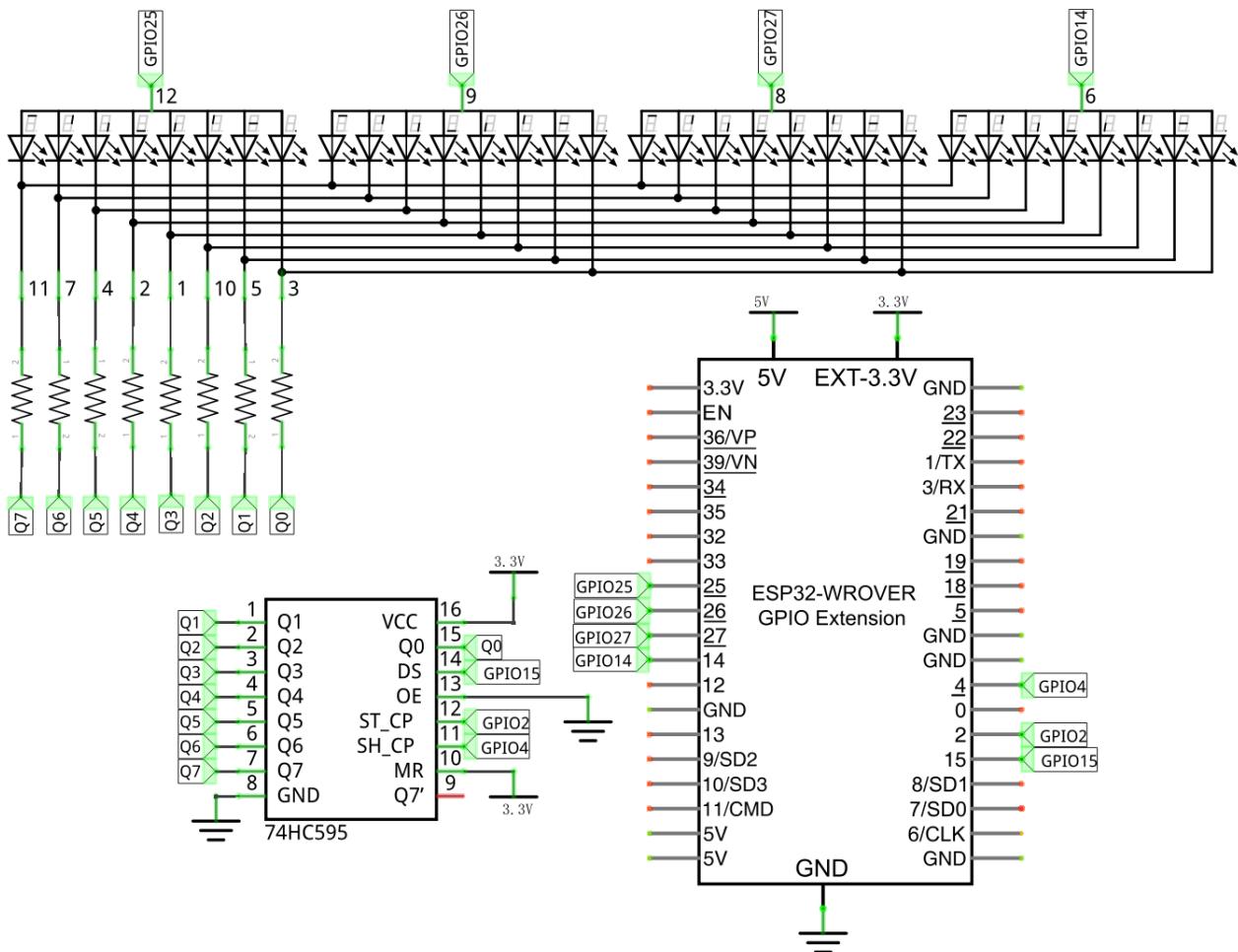


Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit visibly in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the 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 imperceptible to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

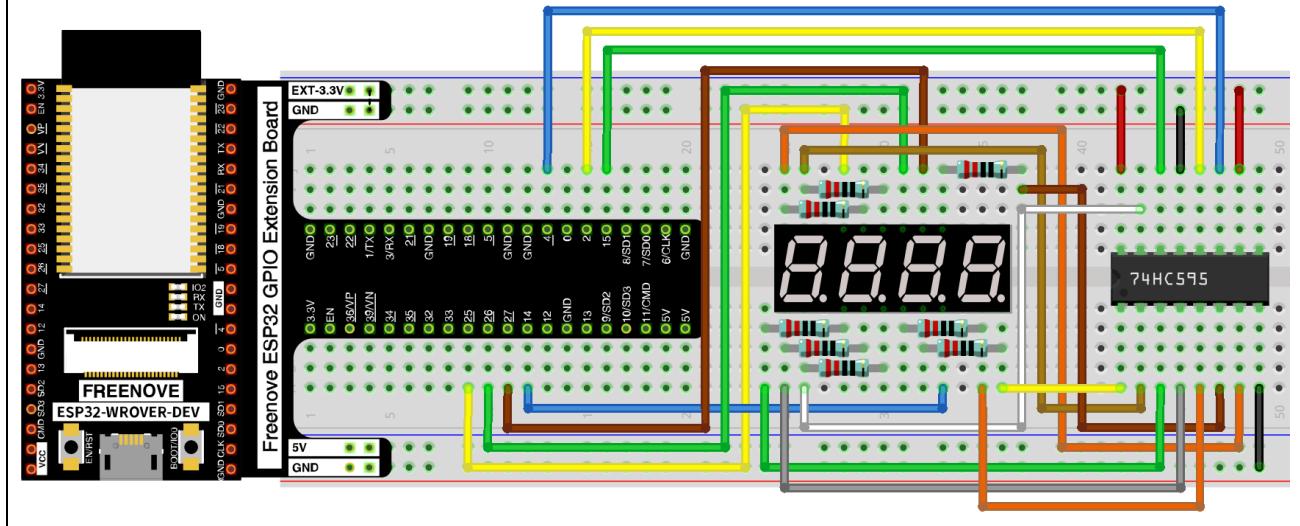
Circuit

Schematic diagram



Q0-Q7 connecting in inverted sequence

Hardware connection:



Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “16.2_4_Digit_7_Segment_Display”. Select “my74HC595.py”, right click your mouse to select “Upload to /”, wait for “my74HC595.py” to be uploaded to ESP32-WROVER and double click “16.2_4_Digit_7_Segment_Display.py”.

16.2 4_Digit_7_Segment_Display

```

1 import time
2 from my74HC595 import Chip74HC595
3 from machine import Pin
4
5 comPin = [25,26,27,14];
6 num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0xe]
8
9 def led_display():
10     for i in range(0,4):
11         chns=Pin(comPin[i],Pin.OUT)
12         chns.value(1)
13         chip.shiftOut(0,num[i])
14         time.sleep_ms(1)
15         chip.clear()
16         chns.value(0)
17
n(15)-74hc595.ds, Pin(4)-74hc595.st_cp, Pin(2)-74hc595.sh_cp
p = Chip74HC595(15,4,2)
:
while True:
    led display()

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

Click “Run current script”, and the Nixie tube display as shown in the image below.



The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3 from machine import Pin
4
5 comPin = [25, 26, 27, 14];
6 num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
8
9 def led_display():
10     for i in range(0, 4):
11         chns=Pin(comPin[i], Pin.OUT)
12         chns.value(1)
13         chip.shiftOut(0, num[i])
14         time.sleep_ms(1)
15         chip.clear()
16         chns.value(0)
17 #Pin(15)-74hc595.ds, Pin(4)-74hc595.st_cp, Pin(2)-74hc595.sh_cp
18 chip = Chip74HC595(15, 4, 2)
19 try:
20     while True:
21         led_display()
22 except:
23     pass
```

Import time, my74HC595 and Pin modules.

```
1 import time  
2 from my74HC595 import Chip74HC595  
3 from machine import Pin
```

Define common anode pins for digital tubes and request a list to put character encodings in it.

```
5 comPin = [25, 26, 27, 14];  
6 num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,  
7           0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0xe8]
```

Request an object to drive 74HC595 and associate pins with it

18 chip = Chip74HC595(15, 4, ?)

Make the digital tube display “0123”

```
9     def led_display():
10        for i in range(0, 4):
11            chns=Pin(comPin[i], Pin.OUT)
12            chns.value(1)
13            chip.shiftOut(0, num[i])
14            time.sleep_ms(1)
15            chip.clear()
16            chns.value(0)
```

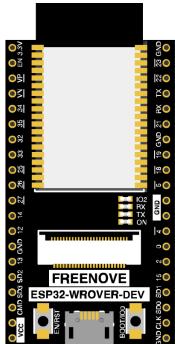
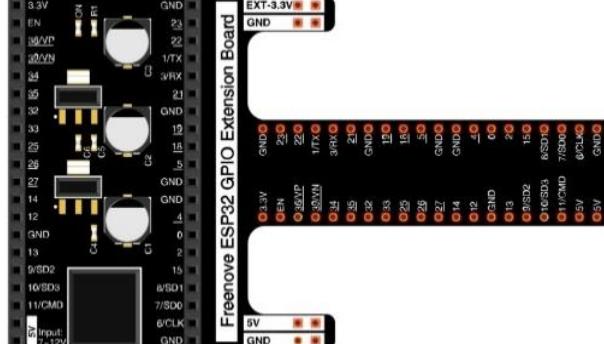
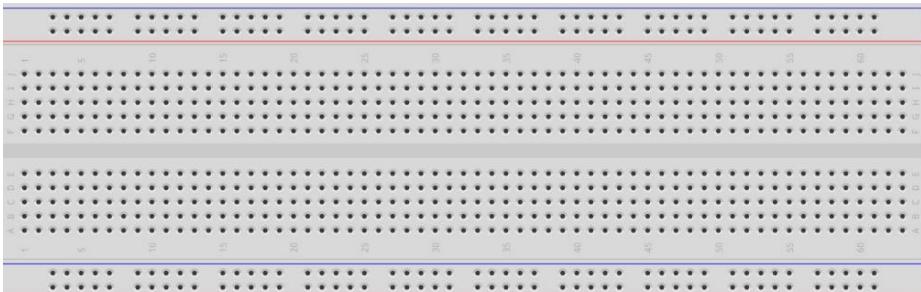
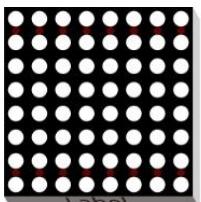
Chapter 16 74HC595 & LED Matrix

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 16.3 LED Matrix

In this project, we will use two 74HC595 IC chips to control a monochrome (one color) (8X8) LED Matrix to make it display both simple graphics and characters.

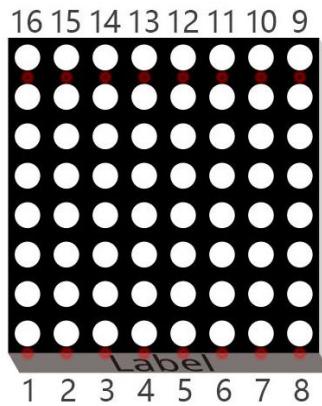
Component List

ESP32-WROVER x1	GPIO Extension Board x1		
			
Breadboard x1			
			
74HC595 x2	8*8 LEDMatrix x1	Resistor 220Ω x8	Jumper M/M
			

Component knowledge

LED matrix

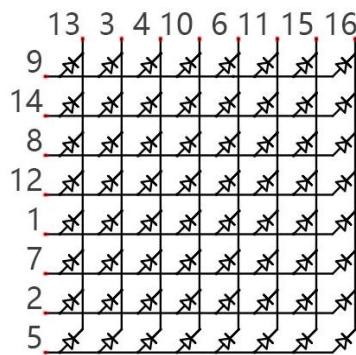
A LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



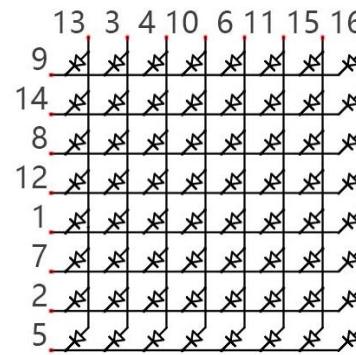
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

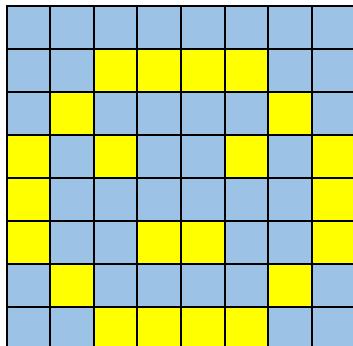
Connection mode of common anode



Connection mode of common cathode



Here is how a Common Anode LED Matrix works. First, choose 16 ports on ESP32 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

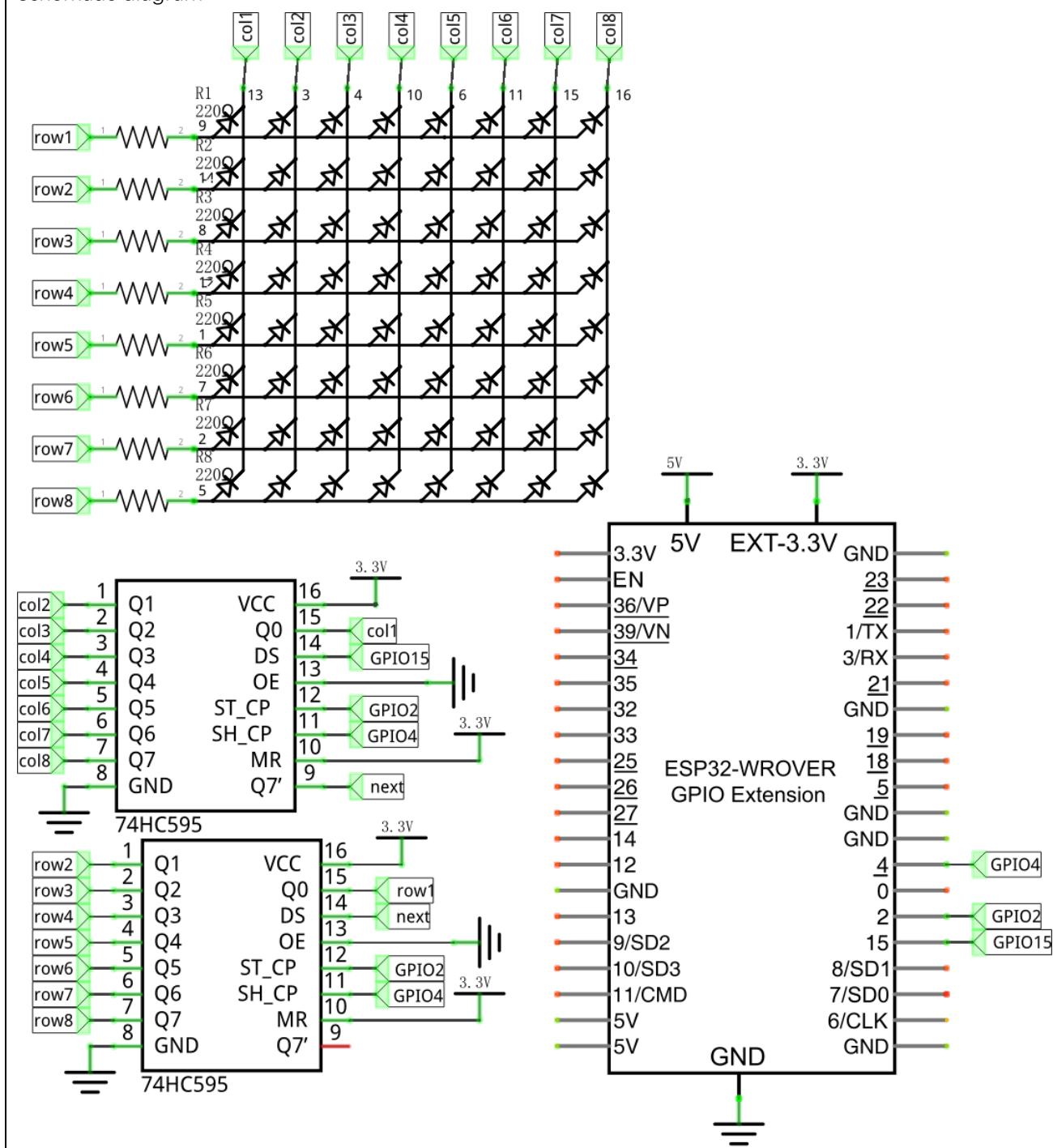
To begin, display the first column, then turn off the first column and display the second column. (and so on) turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated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

Then, to save the number of GPIO, we use a 74HC595. When the first column is turned ON, set the lights that need to be displayed in the first column to "1", otherwise to "0", as shown in the above example, where the value of the first column is 0x1c. This value is sent to 74HC595 to control the display of the first column of the LEDMatrix. Following the above idea, turn OFF the display of the first column, then turn ON the second column, and then send the value of the second column to 74HC595 Until each column is displayed, the LEDMatrix is displayed again from the first column.

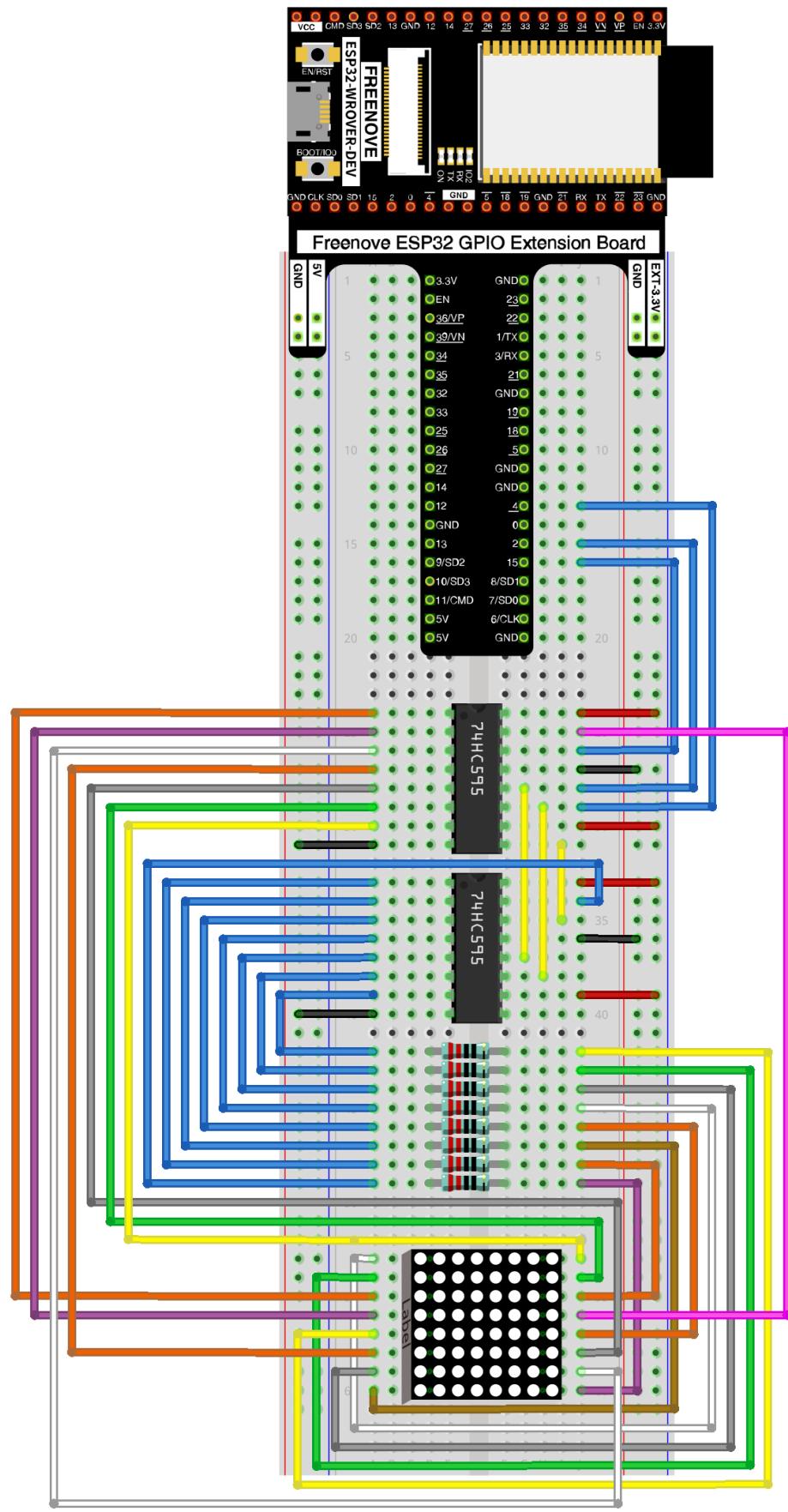
Circuit

In circuit of this project, the power pin of the 74HC595 IC Chip is connected to 3.3V. It can also be connected to 5V to make LED Matrix brighter.

Schematic diagram



Hardware connection:



Code

The following code will make LEDMatrix display a smiling face, and then display scrolling character "0-F". Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "Micropython_Codes". Select "HC595.py", right click your mouse to select "Upload to /", wait for "HC595.py" to be uploaded to ESP32-WROVER and double click "16.3_LED_Matrix.py".

16.3 LED_Matrix

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\16.3_LED_Matrix\16.3_LED_Matrix.py @ 7 : 12". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations. The left sidebar has a "Files" tab and lists "MicroPython device" with files "boot.py" and "my74HC595.py" selected. It also shows "This computer" with files "16.3_LED_Matrix.py", "boot.py", and "my74HC595.py". A context menu is open over "my74HC595.py" with options: Refresh, Upload to / (which is highlighted in blue), Move to Recycle Bin, New directory..., Properties, and Storage space. The main editor window displays the Python code for "16.3_LED_Matrix.py". The code initializes a 74HC595 chip and loops through various functions to display a smiling face and scroll characters. The bottom right corner shows a "Shell" window with the text: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32" and "Type "help()" for more information."

```
37 chip = Chip74HC595(15,2,4,5)
38 try:
39     while True:
40         for j in range(100):
41             cols=0x01
42             for i in range(8):
43                 chip.disable()
44                 chip.shiftOut(1,smilingFace[i])
45                 chip.shiftOut(1,~cols)
46                 time.sleep_us(500)
47                 cols<<=1
48                 chip.enable()
49             for i in range(136):
50                 for k in range(5):
51                     cols=0x01
52                     for j in range(i,8+i):
53                         chip.disable()
54                         chip.shiftOut(1,numdata[j])
55                         chip.shiftOut(0,~cols)
56                         cols<<=1
57                         chip.enable()
58                         time.sleep_us(500)
```

Click "Run current script", and the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 smilingFace=[0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x14]#^_^#
5 numdata = [
6     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
7     0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
8     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
9     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
10    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
11    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
12    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
13    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
14    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
15    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
16    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
17    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
18    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
19    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
20    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
21    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
22    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
23    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
24 ]
25
26 ,,
27 Pin(15)      -  74HC595(1).ds,
28 Pin(2)        -  74HC595(1).st_cp,
29 Pin(4)        -  74HC595(1).sh_cp,
30 Pin(5)        -  74HC595(1).oe
31
32 74HC595(1).q7' -  74HC595(2).ds,
33 Pin(2)        -  74HC595(2).st_cp,
34 Pin(4)        -  74HC595(2).sh_cp,
35 Pin(5)        -  74HC595(2).oe
36 ,,
37 chip = Chip74HC595(15, 2, 4, 5)
38 try:
39     while True:
40         for j in range(100):
41             cols=0x01
42             for i in range(8):
43                 chip.disable()
```

```

44         chip.shiftOut(1, smilingFace[i])
45         chip.shiftOut(1, ~cols)
46         time.sleep_us(500)
47         cols<<=1
48         chip.enable()
49     for i in range(136):
50         for k in range(5):
51             cols=0x01
52             for j in range(i, 8+i):
53                 chip.disable()
54                 chip.shiftOut(1, numdata[j])
55                 chip.shiftOut(0, ~cols)
56                 cols<<=1
57                 chip.enable()
58                 time.sleep_us(500)
59     except:
60         pass

```

Import time and my 74HC595 modules.

```

1 import time
2 from my74HC595 import Chip74HC595

```

Use a nesting of two for loops to display a smiling face.

```

20    for j in range(100):
21        cols=0x01
22        for i in range(8):
23            chip.disable()
24            chip.shiftOut(1, smilingFace[i])
25            chip.shiftOut(1, ~cols)
26            time.sleep_us(500)
27            cols<<=1
28            chip.enable()

```

Use a nesting of two for loops to display "0"-“F”.

```

32     for i in range(136):
33         for k in range(5):
34             cols=0x01
35             for j in range(i, 8+i):
36                 chip.disable()
37                 chip.shiftOut(1, numdata[j])
38                 chip.shiftOut(0, ~cols)
39                 cols<<=1
40                 chip.enable()
41                 time.sleep_us(500)

```

The amount of pins of ESP32 is limited, so we need to find ways to save pins. If we use ESP32's GPIO to control the LEDMatrix instead of 74HC595, we need 16 pins to drive LED matrix. In this example, we use two 74HC595 chips to drive the LED matrix, requiring only three pins, so that we could save the rest of 13 pins.

Reference

Class HC595

Before each use of HC595, please make sure HC595.py has been uploaded to “/” of ESP32, and then add the statement “**import HC595**” to the top of the python file.

Chip74HC595(): The object to control LEDMatrix.

chip=Chip74HC595() or **chip=Chip74HC595(15,2,4,5)**

set_bit_data(data): Write data to 74HC595.

clear(): Clear the latch data of 74HC595.



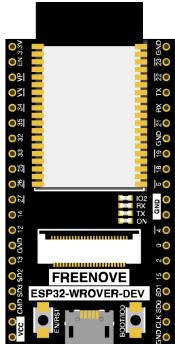
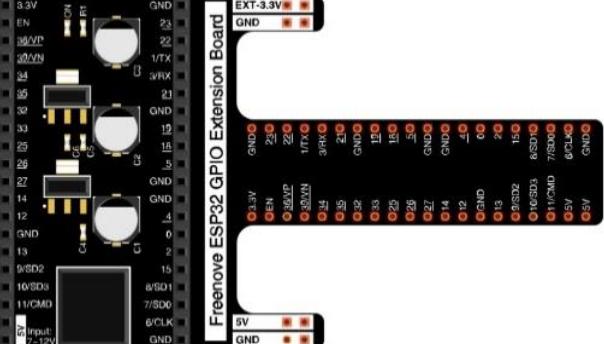
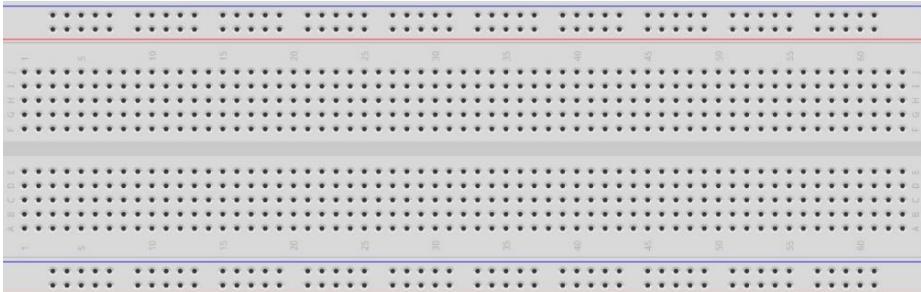
Chapter 17 Relay & Motor

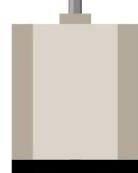
In this chapter, we will learn a kind of special switch module, Relay Module.

Project 17.1 Relay & Motor

In this project, we will use a Push Button Switch indirectly to control the motor via a Relay.

Component List

ESP32-WROVER x1		GPIO Extension Board x1	
Breadboard x1			
Jumper M/M			
	3V battery (prepared by yourself) & battery line 		

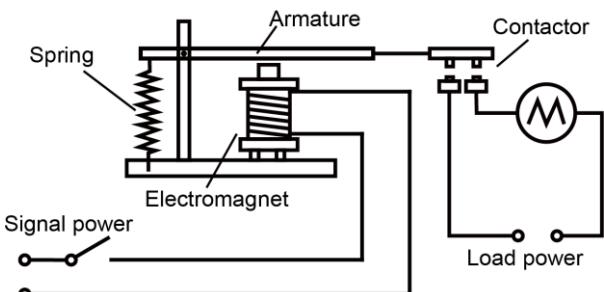
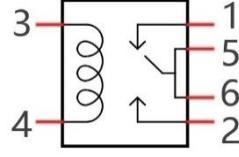
Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1			
					
NPN transistor x1	Relay x1	Motor x1	Push button x1	LED x1	Diode x1
					

Component knowledge

Relay

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

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

Diagram	Feature:	Symbol
		

Pin 5 and pin 6 are connected to each other inside. When the coil pin 3 and 4 get connected to 5V power supply, pin 1 will be disconnected from pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

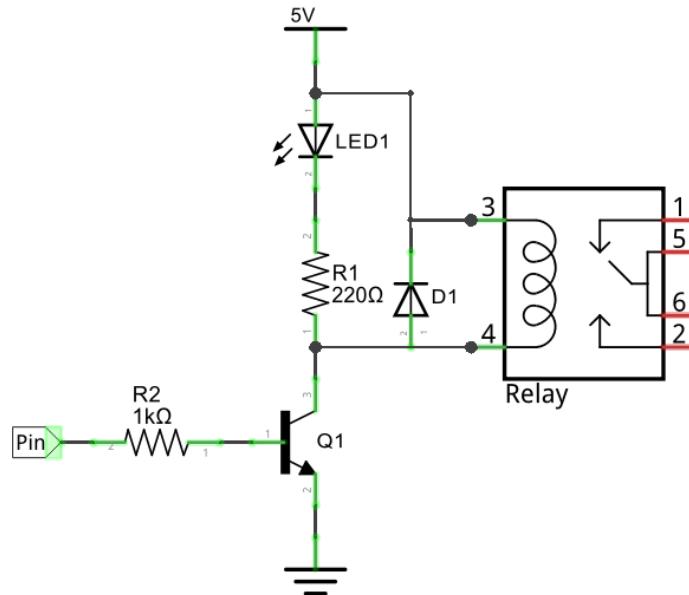
Inductor

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

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

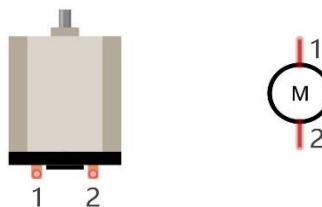


The reference circuit for relay is as follows. The coil of relays can be equivalent to that of inductors, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to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

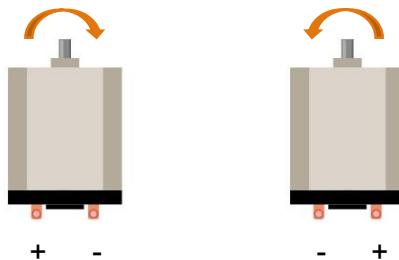


Motor

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

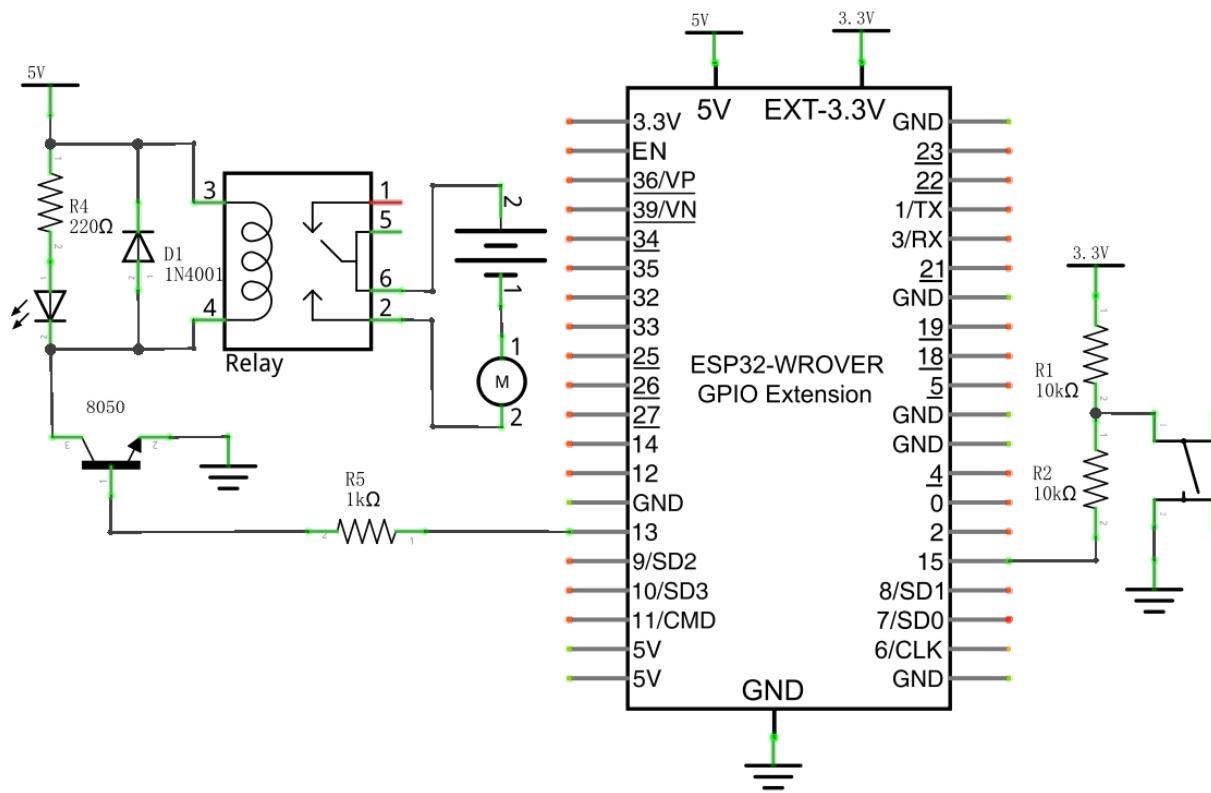


When a motor gets connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.

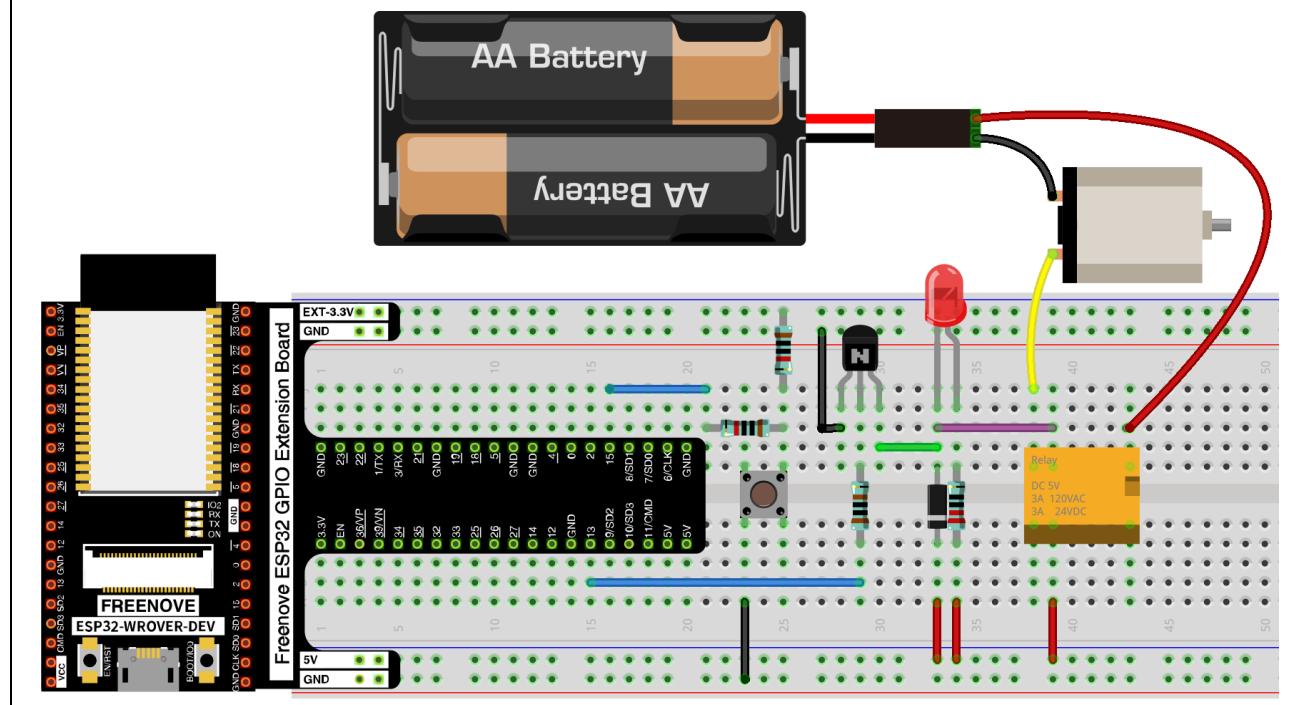


Circuit

Schematic diagram



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





Code

Use buttons to control the relays and motors.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “17.1_Relay_And_Motor” and double click “17.1_Relay_And_Motor.py”.

17.1 Relay and Motor

```

1 import time
2 from machine import Pin
3
4 relay = Pin(13, Pin.OUT)
5 button = Pin(15, Pin.IN,Pin.PULL_UP)
6
7 def reverseRelay():
8     if relay.value():
9         relay.value(0)
10    else:
11        relay.value(1)
12
13 while True:
14     if not button.value():
15         time.sleep_ms(20)
16         if not button.value():
17             reverseRelay()
18             while not button.value():
19                 time.sleep_ms(20)

```

Shell

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

```

Click “Run current script”. When the 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.



The following is the program code:

```
1 import time
2 from machine import Pin
3
4 relay = Pin(13, Pin.OUT)
5 button = Pin(15, Pin.IN, Pin.PULL_UP)
6
7 def reverseRelay():
8     if relay.value():
9         relay.value(0)
10    else:
11        relay.value(1)
12
13 while True:
14     if not button.value():
15         time.sleep_ms(20)
16         if not button.value():
17             reverseRelay()
18             while not button.value():
19                 time.sleep_ms(20)
```

This section of code is basically the same as that of project Tablelamp. If you don't understand the program, you can click [here](#) to go back to the Tablelamp and study again.

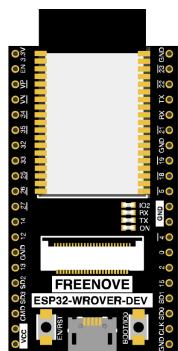


Project 17.2 Control Motor with Potentiometer

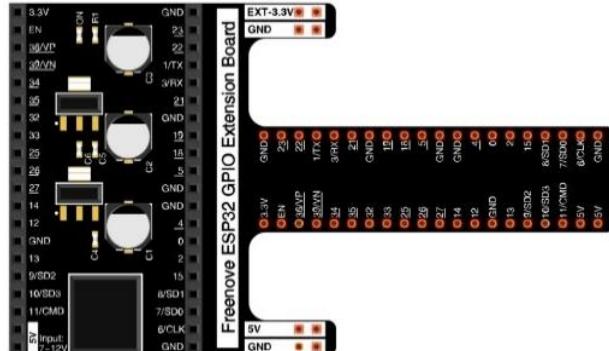
Control the direction and speed of the motor with a potentiometer.

Component List

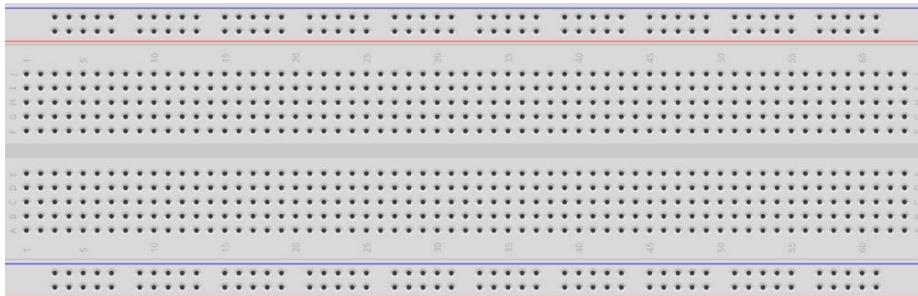
ESP32-WROVER x1



GPIO Extension Board x1



Breadboard x1



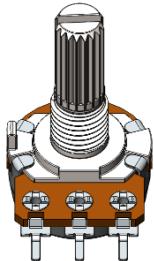
Jumper M/M



3V battery (prepared by yourself) & battery line



Rotary potentiometer x1



Motor x1



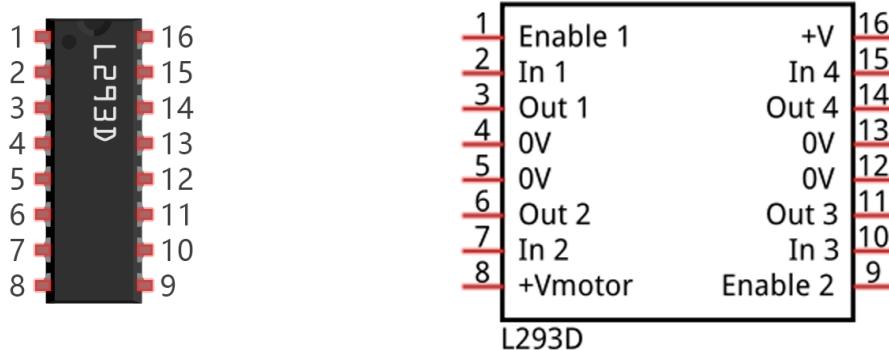
L293D



Component knowledge

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).



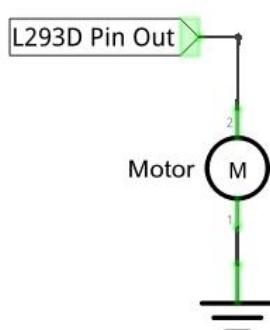
Port description of L293D module is as follows:

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, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 3.0~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

For more detail, please refer to the datasheet for this IC Chip.

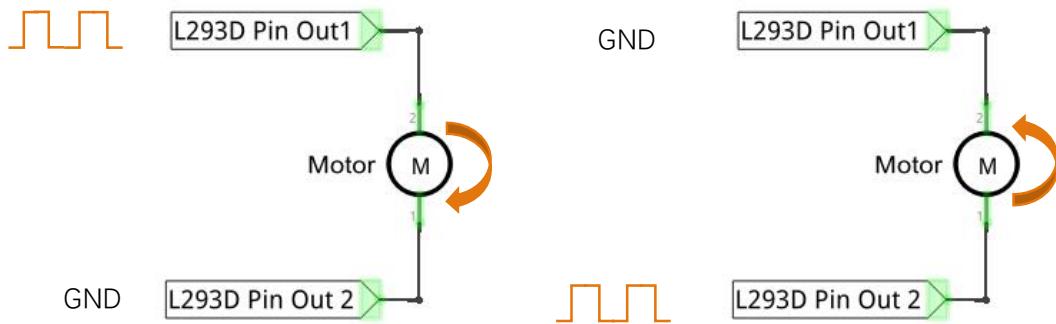
When using L293D to drive DC motor, there are usually two connection options.

The following connection option uses one channel of the L293D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.





The following connection uses two channels of the L293D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the steering of the motor.

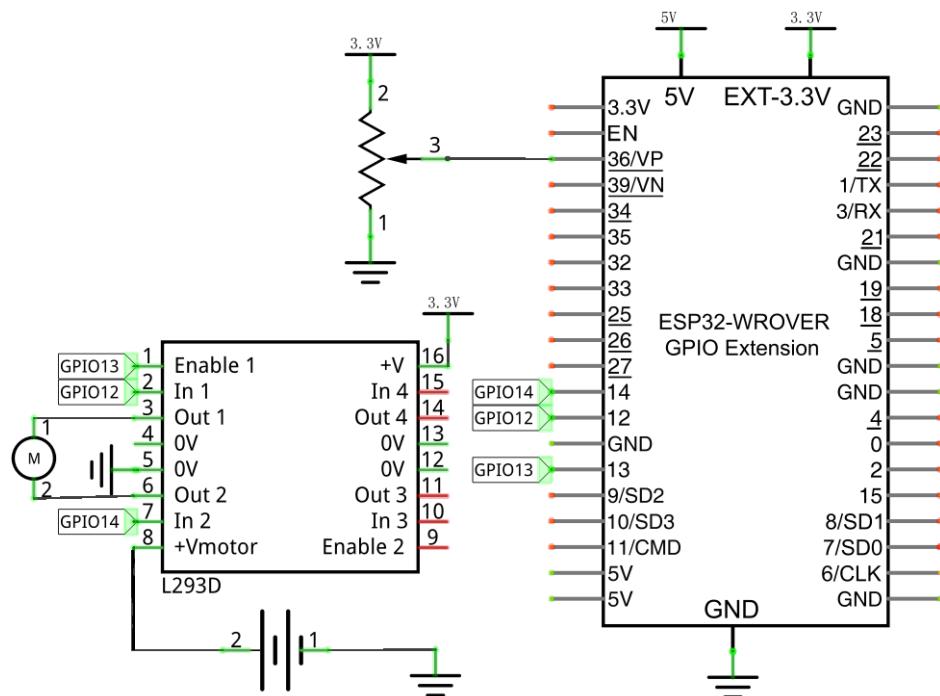


In practical use the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

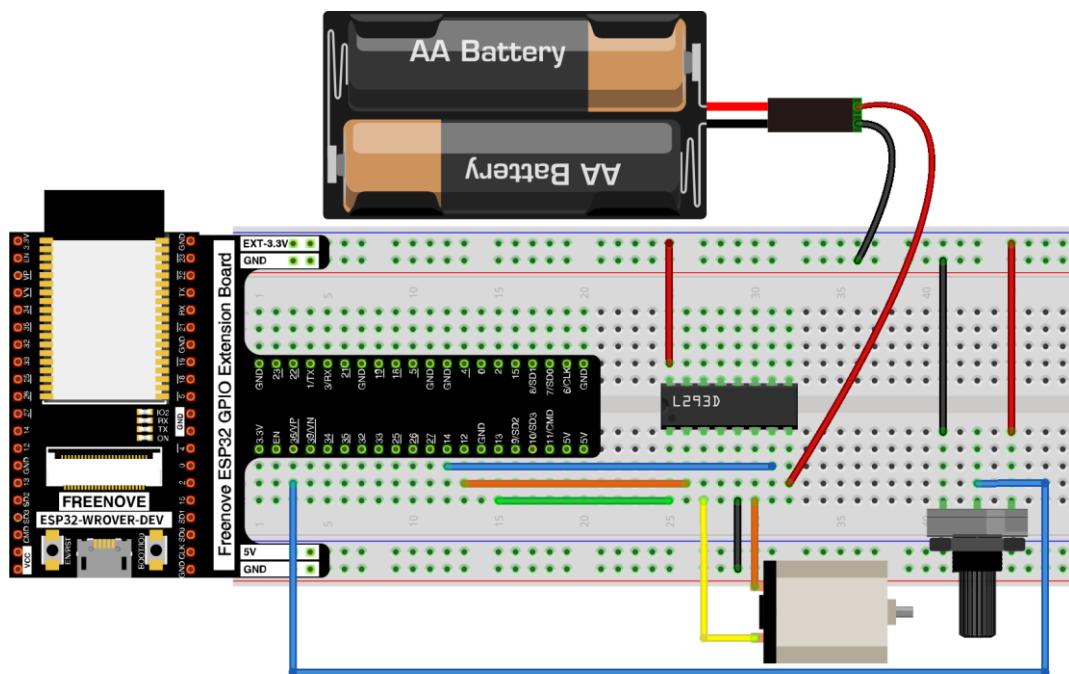
Circuit

Use caution: when connecting this circuit because the DC Motor is a high-power component, do NOT use the power provided by the ESP32 to power the motor directly, as this may cause permanent damage to your ESP32! The logic circuit can be powered by the ESP32 power or an external power supply, which should share a common ground with ESP32.

Schematic diagram



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





Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “17.2_Motor_And_Driver” and double click “17.2_Motor_And_Driver.py”.

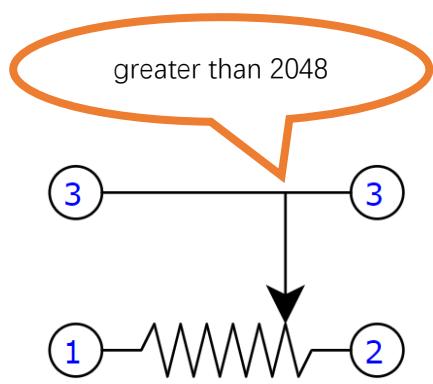
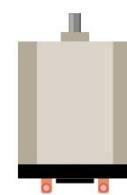
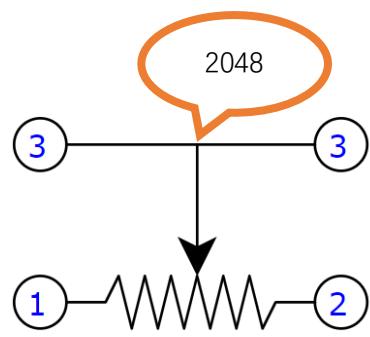
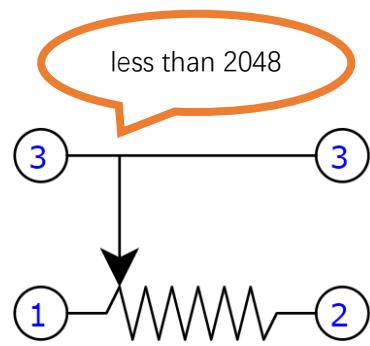
17.2_Motor_And_Driver

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\17.2_Motor_And_Driver\17.2_Motor_And_Driver.py @ 36 : 17". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The left sidebar has a "Files" tab showing a "MicroPython device" folder containing "boot.py" and a "This computer" folder containing "D:\Micropython_Codes\17.2_Motor_And_Driver\17.2_Motor_And_Driver.py". The main code editor window displays the following Python code:

```
14 def driveMotor(dir,spd):
15     if dir:
16         in1Pin.value(1)
17         in2Pin.value(0)
18     else :
19         in1Pin.value(0)
20         in2Pin.value(1)
21     pwm.duty(spd)
22
23 try:
24     while True:
25         potenVal = adc.read()
26         rotationSpeed = potenVal - 2048
27         if (potenVal > 2048):
28             rotationDir = 1;
29         else:
30             rotationDir = 0;
31         rotationSpeed=int(math.fabs((potenVal-2047)//2)-1)
32         driveMotor(rotationDir,rotationSpeed)
33         time.sleep_ms(10)
```

The bottom right pane is a "Shell" window showing the MicroPython prompt: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32 Type "help()" for more information. >>>"

Click “Run current script”, rotate the potentiometer in one direction and the motor speeds up slowly in one direction. Rotate the potentiometer in the other direction and the motor will slow down to stop. And then rotate it in the original direction to accelerate the motor.





The following is the Code:

```
1  from machine import ADC, Pin, PWM
2  import time
3  import math
4
5  in1Pin=Pin(12, Pin.OUT)
6  in2Pin=Pin(14, Pin.OUT)
7
8  enablePin=Pin(13, Pin.OUT)
9  pwm=PWM(enablePin, 10000, 512)
10
11 adc=ADC(Pin(36))
12 adc.atten(ADC.ATTN_11DB)
13 adc.width(ADC.WIDTH_12BIT)
14
15 def driveMotor(dir, spd):
16     if dir :
17         in1Pin.value(1)
18         in2Pin.value(0)
19     else :
20         in1Pin.value(0)
21         in2Pin.value(1)
22     pwm.duty(spd)
23
24 try:
25     while True:
26         potenVal = adc.read()
27         rotationSpeed = potenVal - 2048
28         if (potenVal > 2048):
29             rotationDir = 1;
30         else:
31             rotationDir = 0;
32         rotationSpeed=int(math.fabs((potenVal-2047)//2)-1)
33         driveMotor(rotationDir, rotationSpeed)
34         time.sleep_ms(10)
35 except:
36     pass
```

The ADC of ESP32 has a 12-bit accuracy, corresponding to a range from 0 to 4095. In this program, set the number 2048 as the midpoint. If the value of ADC is less than 2048, make the motor rotate in one direction. If the value of ADC is greater than 2048, make the motor rotate in the other direction. Subtract 2048 from the ADC value and take the absolute value, and then divide this result by 2 to be the speed of the motor.

```
26     potenVal = adc.read()
27     rotationSpeed = potenVal - 2048
28     if (potenVal > 2048):
29         rotationDir = 1;
30     else:
31         rotationDir = 0;
32     rotationSpeed=int(math.fabs((potenVal-2047)//2)-1)
33     driveMotor(rotationDir, rotationSpeed)
34     time.sleep_ms(10)
```

Initialize pins of L293D chip.

```
5     in1Pin=Pin(12, Pin.OUT)
6     in2Pin=Pin(14, Pin.OUT)
7
8     enablePin=Pin(13, Pin.OUT)
9     pwm=PWM(enablePin, 10000, 512)
```

Initialize ADC pins, set the range of voltage to 0-3.3V and the acquisition width of data to 0-4095.

```
11    adc=ADC(Pin(36))
12    adc.atten(ADC.ATTN_11DB)
13    adc.width(ADC.WIDTH_12BIT)
```

Function `driveMotor` is used to control the rotation direction and speed of the motor. The `dir` represents direction while `spd` refers to speed.

```
15    def driveMotor(dir, spd):
16        if dir :
17            in1Pin.value(1)
18            in2Pin.value(0)
19        else :
20            in1Pin.value(0)
21            in2Pin.value(1)
22        pwm.duty(spd)
```



Chapter 18 Servo

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

Project 18.1 Servo Sweep

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

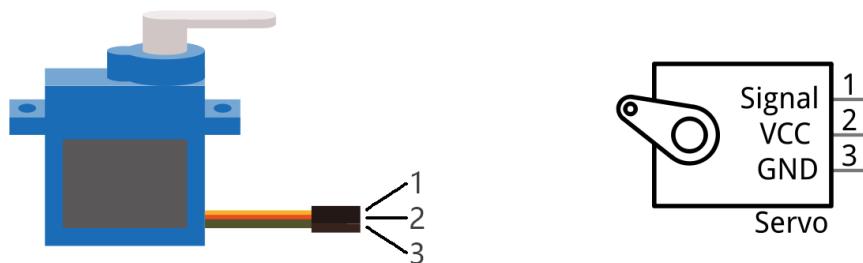
Component List

ESP32-WROVER x1	GPIO Extension Board x1
Breadboard x1	
Servo x1	Jumper M/M x3

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 time interval of 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

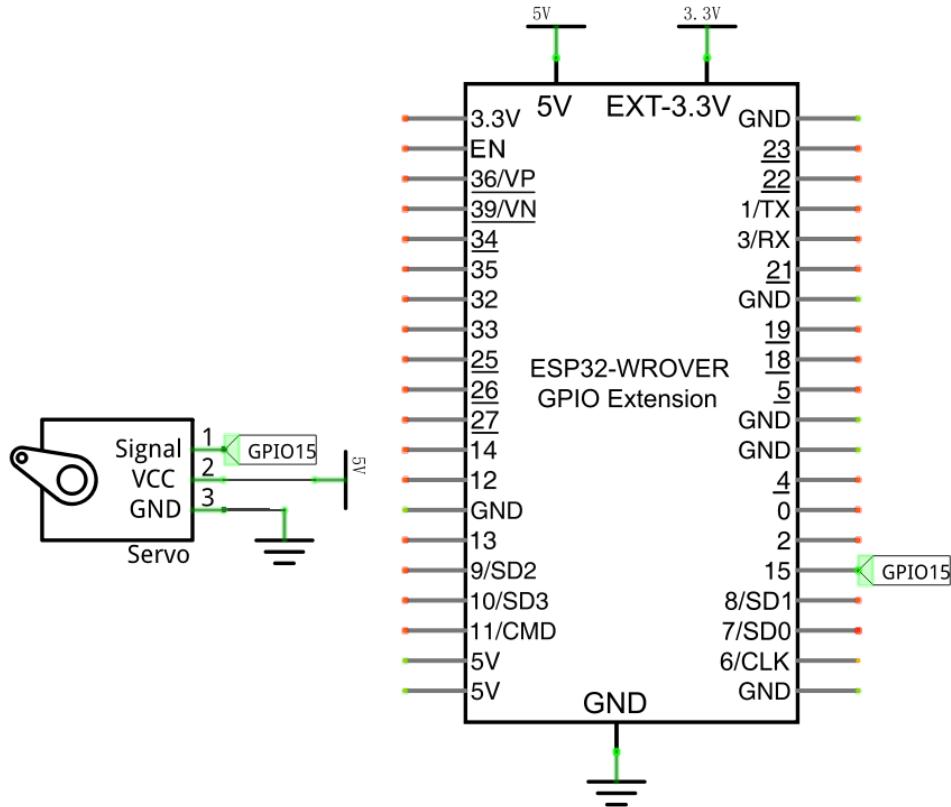
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

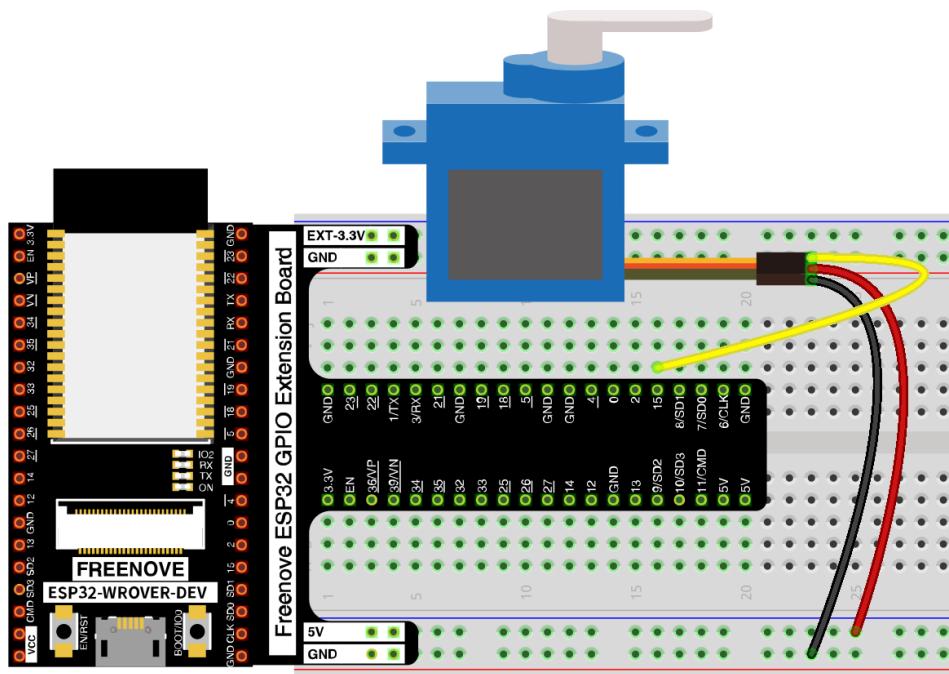
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “18.1_Servo_Sweep”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to ESP32-WROVER and then double click “18.1_Servo_Sweep.py”.

18.1 Servo_Sweep

The screenshot shows the Thonny IDE interface. In the center, there is a code editor window titled "18.1_Servo_Sweep.py" containing Python code for a servo sweep. To the left, there is a file browser window titled "Files" showing files on a "MicroPython device" and "This computer". The file "myservo.py" is selected in the "This computer" list. A context menu is open over this file, with the "Upload to /" option highlighted in blue. At the bottom of the screen, a terminal window displays the MicroPython version and a prompt for commands.

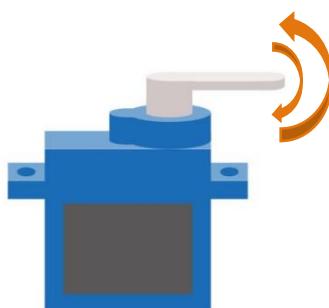
```
from myservo import myServo
import time

servo=myServo(15)
servo.myServoWriteAngle(0)
time.sleep_ms(1000)

try:
    while True:
        for i in range(0,180,1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
        for i in range(180,0,-1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
except:
    servo.deinit()
```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

Click “Run current script”, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.





The following is the program code:

```
1  from myservo import myServo
2  import time
3
4  servo=myServo(15)
5  servo.myServoWriteAngle(0)
6  time.sleep_ms(1000)
7
8  try:
9      while True:
10         for i in range(0, 180, 1):
11             servo.myServoWriteAngle(i)
12             time.sleep_ms(15)
13         for i in range(180, 0, -1):
14             servo.myServoWriteAngle(i)
15             time.sleep_ms(15)
16     except:
17         servo.deinit()
```

Import myservo module.

```
1  from myservo import myServo
```

Initialize pins of the servo and set the starting point of the servo to 0 degree.

```
4  servo=myServo(15)
5  servo.myServoWriteAngle(0)
6  time.sleep_ms(1000)
```

Control the servo to rotate to a specified angle within the range of 0-180 degrees.

```
8  servo.myServoWriteAngle(i)
```

Use two for loops. The first one controls the servo to rotate from 0 degree to 180 degrees while the other controls it to rotate back from 180 degrees to 0 degree.

```
10    for i in range(0, 180, 1):
11        servo.myServoWriteAngle(i)
12        time.sleep_ms(15)
13    for i in range(180, 0, -1):
14        servo.myServoWriteAngle(i)
15        time.sleep_ms(15)
```

Reference

```
class myServo
```

Before each use of **myServo**, please make sure myservo.py has been uploaded to “/” of ESP32, and then add the statement “**from myservo import myServo**” to the top of the python file.

myServo (): The object that controls the servo, with the default pin Pin(15), default frequency 50Hz and default duty cycle 512.

myServoWriteDuty(duty): The function that writes duty cycle to control the servo.

duty: Range from 26 to 128, with 26 corresponding to the servo's 0 degree and 128 corresponding to 180 degrees.

myServoWriteAngle(pos): Function that writes angle value to control the servo.

pos: Ranging from 0-180, corresponding the 0-180 degrees of the servo.

myServoWriteTime(us): Writes time to control the servo.

us: Range from 500-2500, with 500 corresponding to the servo's 0 degree and 2500 corresponding to 180 degrees.



Project 18.2 Servo Knop

Use a potentiometer to control the servo motor to rotate at any angle.

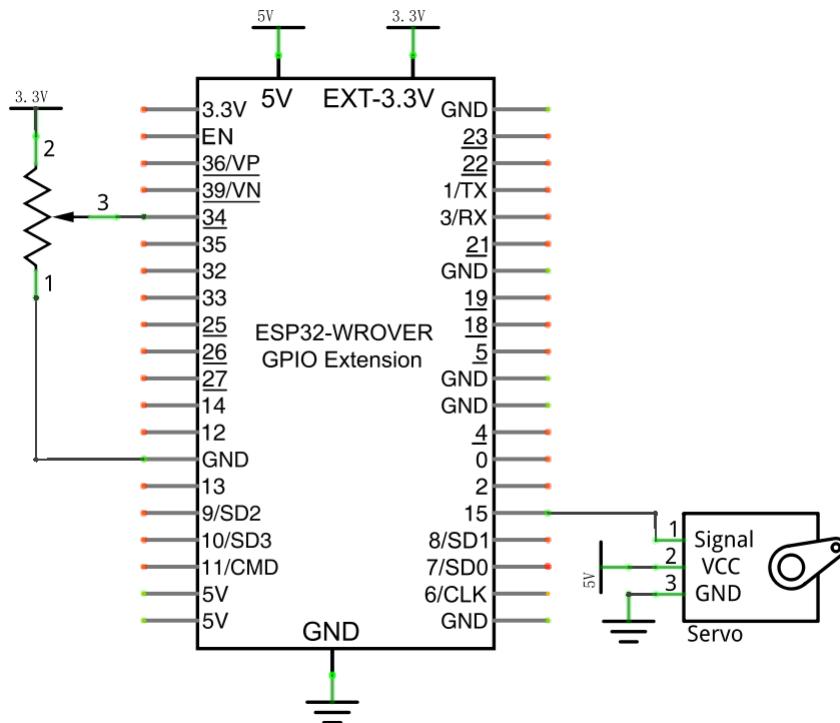
Component List

ESP32-WROVER x1	GPIO Extension Board x1	
Breadboard x1		
Servo x1	Jumper M/M x6	Rotary potentiometer x1

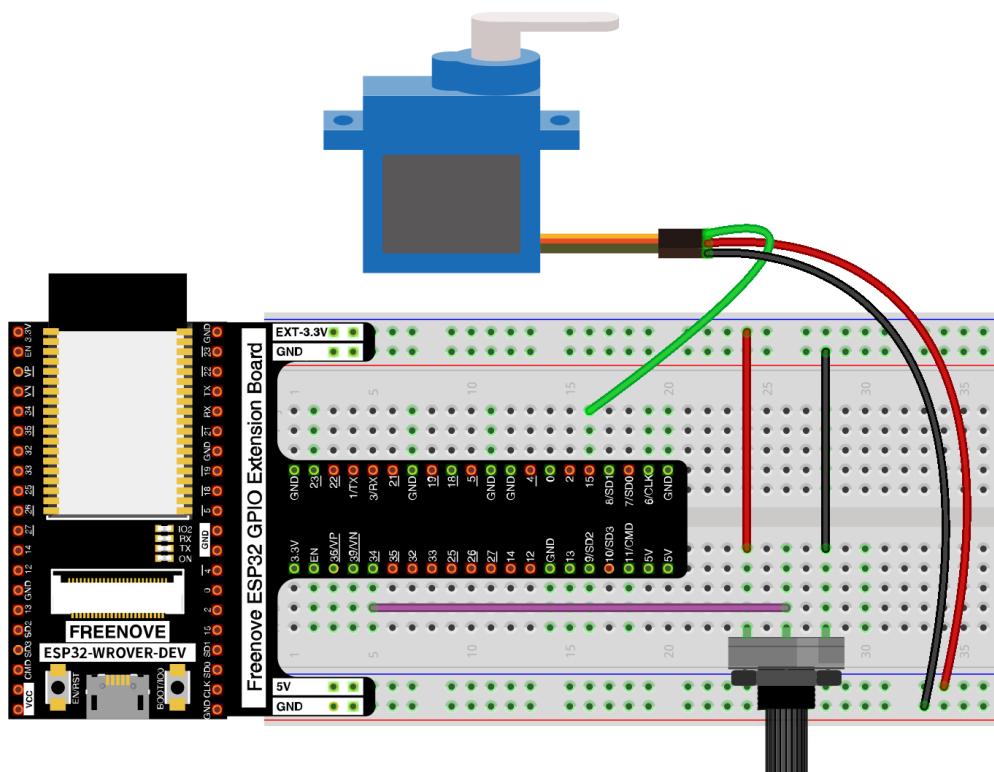
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “18.2_Servo_Knop”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to ESP32-WROVER and then double click “18.2_Servo_Knop.py”.

18.2 Servo_Knop

```

from myservo import myServo
from machine import ADC,Pin
import time

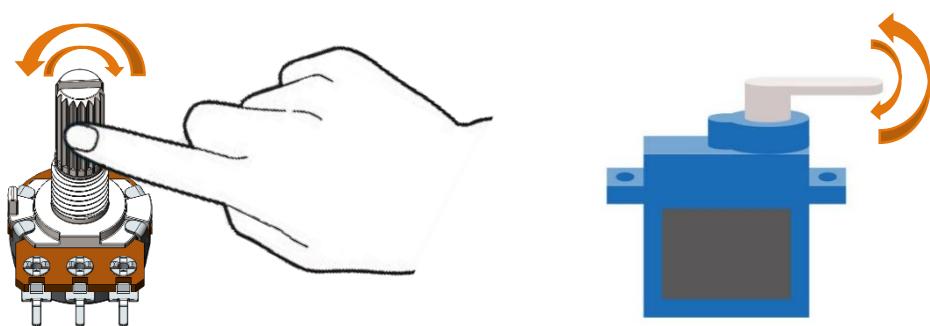
servo=myServo(15)

adc2=ADC(Pin(34))
adc2.atten(ADC.ATTN_11DB)
adc2.width(ADC.WIDTH_12BIT)

try:
    while True:
        adcValue=adc2.read()
        angle=(adcValue*180)/4096
        servo.myServoWriteAngle(int(angle))
        time.sleep_ms(50)
except:
    servo.deinit()

```

Click “Run current script”, twist the potentiometer back and forth, and the servo motor rotates accordingly.



The following is the program code:

```
1  from myservo import myServo
2  from machine import ADC, Pin
3  import time
4
5  servo=myServo(15)
6
7  adc2=ADC(Pin(34))
8  adc2.atten(ADC.ATTN_11DB)
9  adc2.width(ADC.WIDTH_12BIT)
10
11 try:
12     while True:
13         adcValue=adc2.read()
14         angle=(adcValue*180)/4096
15         servo.myServoWriteAngle(int(angle))
16         time.sleep_ms(50)
17     except:
18         servo.deinit()
```

In this project, we will use Pin(34) of ESP32 to read the ADC value of the rotary potentiometer and then convert it to the angle value required by the servo and control the servo to rotate to the corresponding angle.



Chapter 19 Stepper Motor

In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

Project 19.1 Stepping Motor

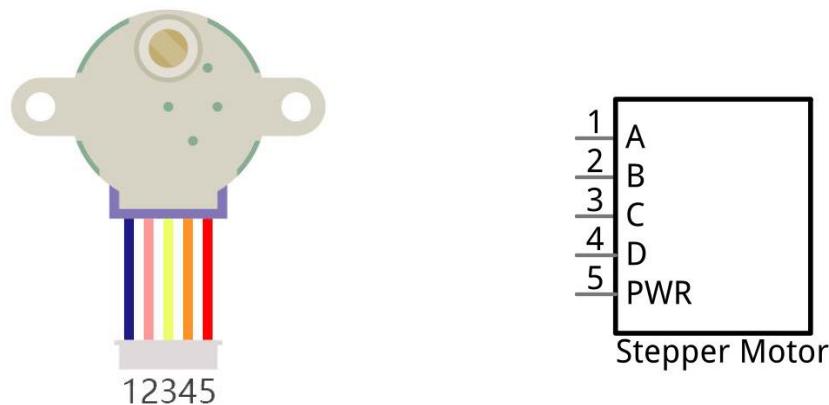
Component List

ESP32-WROVER x1	GPIO Extension Board x1
Breadboard x1	
Stepping Motor x1	ULN2003 Stepper motorDriver x1
	Jumper F/M x6

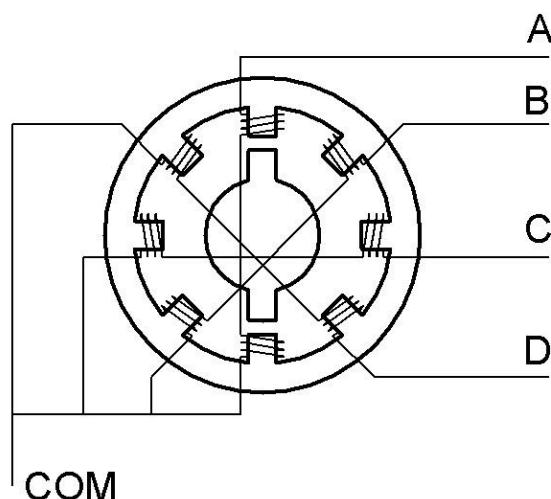
Component knowledge

Stepper Motor x1

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 depend only on the pulse signal frequency as well as the number of pulses and are not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:

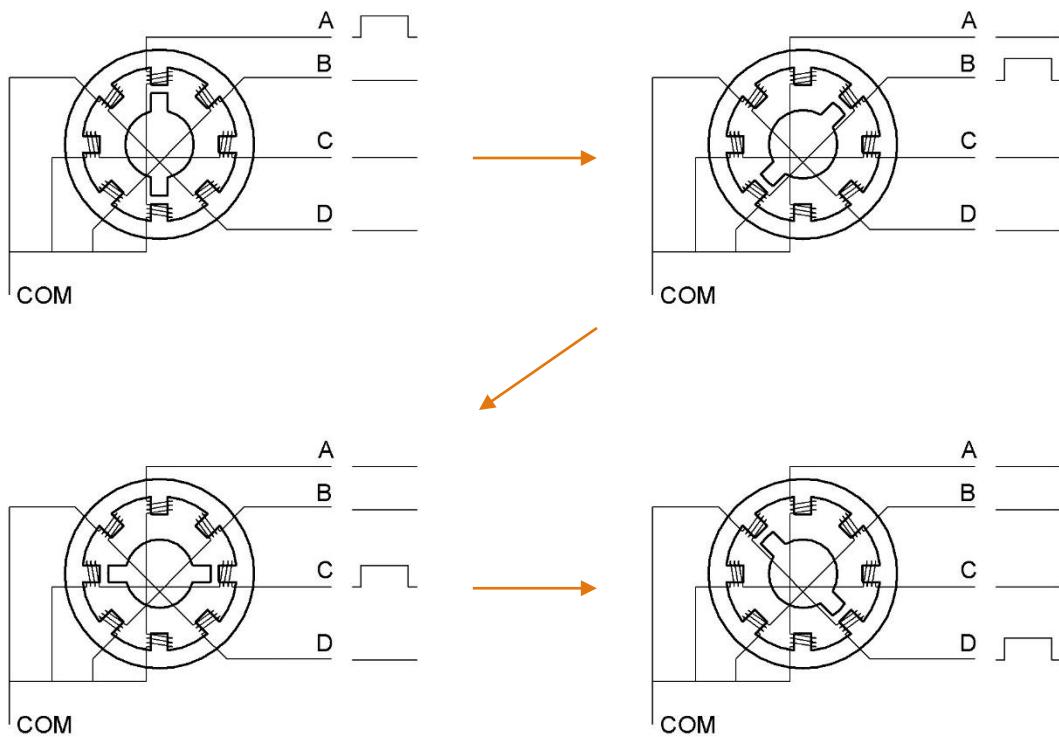


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There are 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 process is as follows:



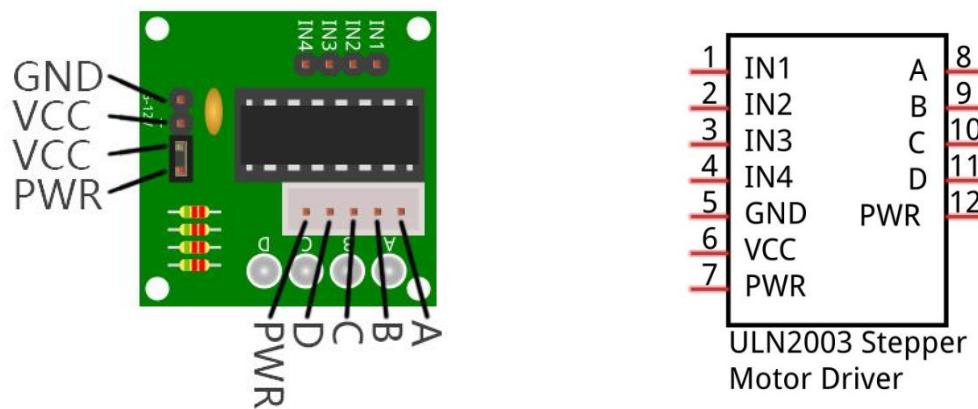
In the course above, the stepping motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise,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

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. The sequence of powering the coils looks like this: A→ AB→B →BC→C →CD→D→DA→A→…… , the rotor will rotate in accordance to this sequence at a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the Stepper Motor will rotate in the opposite direction.

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor's housing) requires $32 \times 64 = 2048$ steps to make one full revolution.

ULN2003 Stepping motor driver

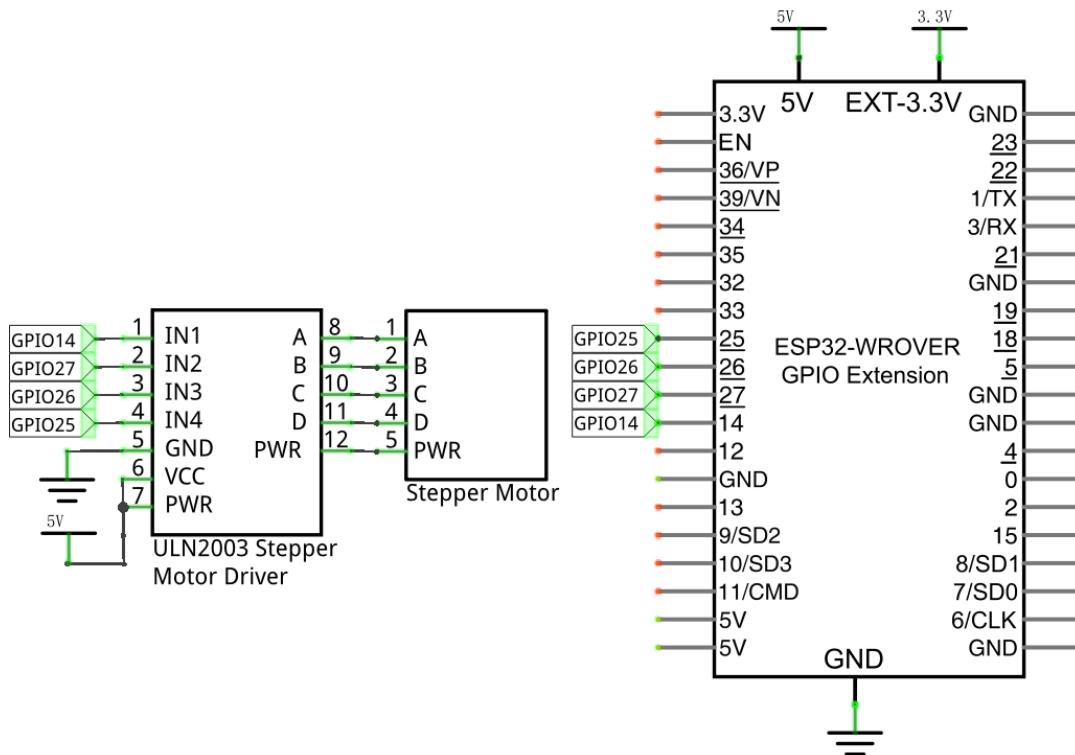
A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.



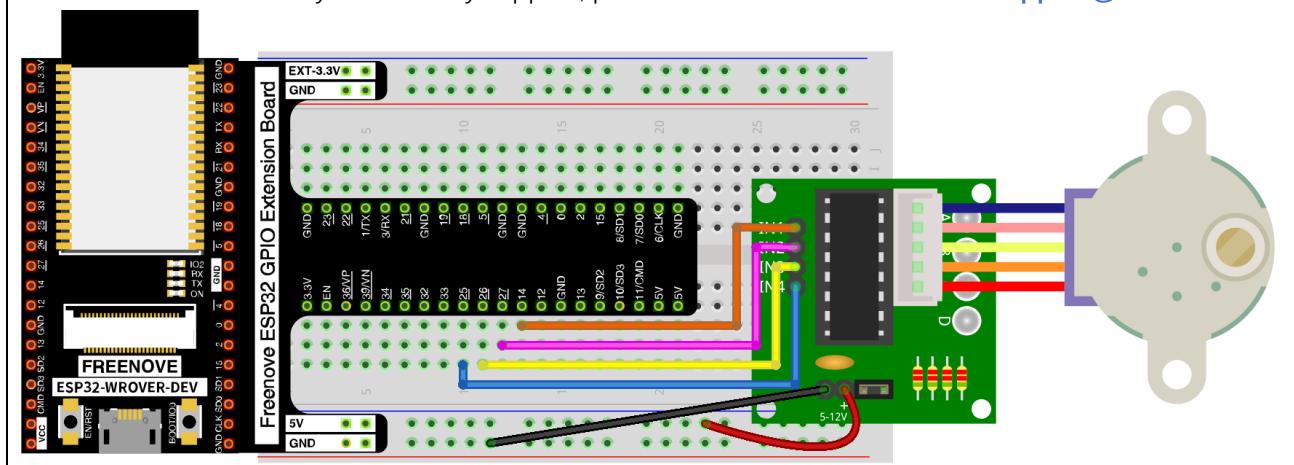
Circuit

When building the circuit, note that rated voltage of the Stepper Motor is 5V, and we need to use the Breadboard power supply independently. Additionally, the Breadboard power supply needs to share Ground with ESP32.

Schematic diagram



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

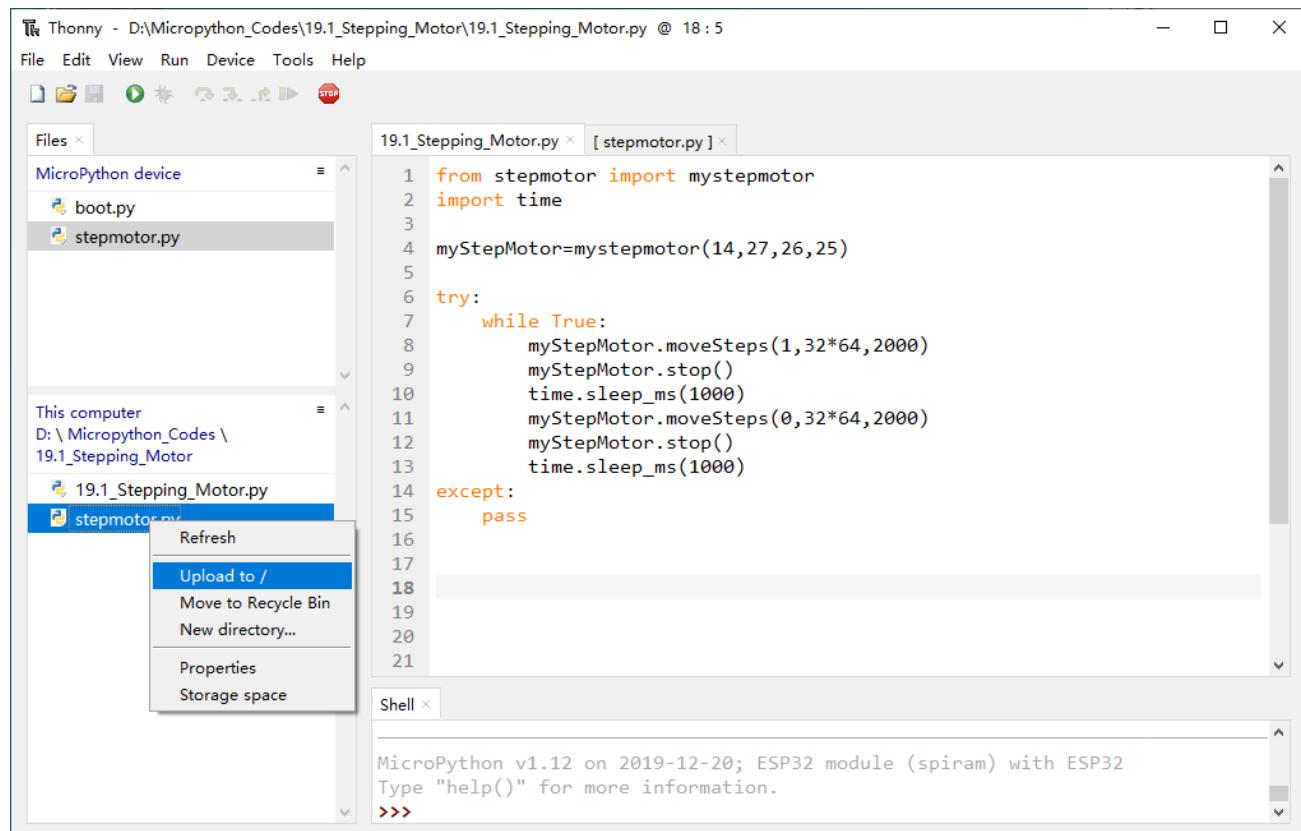


Code

This code uses the four-step, four-part mode to drive the Stepper Motor in the clockwise and anticlockwise directions.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “19.1_Stepping_Motor”. Select “stepmotor.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to ESP32-WROVER and then double click “19.1_Stepping_Motor.py”.

19.1 Stepping_Motor



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\19.1_Stepping_Motor\19.1_Stepping_Motor.py @ 18 : 5". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar shows a "Files" tree with "MicroPython device" expanded, showing "boot.py" and "stepmotor.py". Below it, "This computer" shows "D:\ Micropython_Codes \ 19.1_Stepping_Motor" with "19.1_Stepping_Motor.py" selected. A context menu is open over "stepmotor.py" with options: Refresh, Upload to / (highlighted in blue), Move to Recycle Bin, New directory..., Properties, and Storage space. The main code editor window contains the following Python code:

```

1 from stepmotor import mystepmotor
2 import time
3
4 myStepMotor=mystepmotor(14,27,26,25)
5
6 try:
7     while True:
8         myStepMotor.moveSteps(1,32*64,2000)
9         myStepMotor.stop()
10        time.sleep_ms(1000)
11        myStepMotor.moveSteps(0,32*64,2000)
12        myStepMotor.stop()
13        time.sleep_ms(1000)
14 except:
15     pass
16
17
18
19
20
21

```

The bottom shell window shows the MicroPython version and a prompt:

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

```

Click “Run current script”, the stepper motor will rotate 360° clockwise and stop for 1s, and then rotate 360° anticlockwise and stop for 1s. And it will repeat this action in an endless loop.





The following is the program code:

```
1 from stepmotor import mystepmotor
2 import time
3
4 myStepMotor=mystepmotor(14, 27, 26, 25)
5
6 try:
7     while True:
8         myStepMotor.moveSteps(1, 32*64, 2000)
9         myStepMotor.stop()
10        time.sleep_ms(1000)
11        myStepMotor.moveSteps(0, 32*64, 2000)
12        myStepMotor.stop()
13        time.sleep_ms(1000)
14 except:
15     pass
```

Import time and stepmotor modules.

```
1 from stepmotor import mystepmotor
2 import time
```

In this project, we define four pins to drive the stepper motor.

```
4 myStepMotor=mystepmotor(14, 27, 26, 25)
```

Call the function moveSteps to control the stepper motor to rotate for 360° and then call function stop() to stop it.

```
8     myStepMotor.moveSteps(1, 32*64, 2000)
9     myStepMotor.stop()
```

Repeatedly control the stepmotor to rotate 360° clockwise and then rotate 360° anti-clockwise.

```
7     while True:
8         myStepMotor.moveSteps(1, 32*64, 2000)
9         myStepMotor.stop()
10        time.sleep_ms(1000)
11        myStepMotor.moveSteps(0, 32*64, 2000)
12        myStepMotor.stop()
13        time.sleep_ms(1000)
```

Reference

class myServo

Before each use of the object **mystepper**, please make sure that stepmotor.py has been uploaded to "/" of ESP32, and then add the statement "**from stepmotor import mystepper**" to the top of the python file.

mystepper(): The object to control the stepper motor. The default control pins are Pin(14), Pin(27), Pin(26) and Pin(25).

moveSteps(direction,steps,us): Control the stepper motor to rotate a specified number of steps.

direction: The rotation direction of stepper motor.

Steps: Rotation steps of the stepper motor.

us: Time required by the stepper motor to rotate for one step.

moveAround(direction,turns,us): Control the stepper motor to rotate a specific number of turns.

Turns: Number of turns that the stepper motor rotates.

moveAngle(direction,angles,us): Control the stepper motor to rotate a specific angle.

Angles: Rotation angles that the stepper motor rotates.

stop(): Stop the stepper motor.



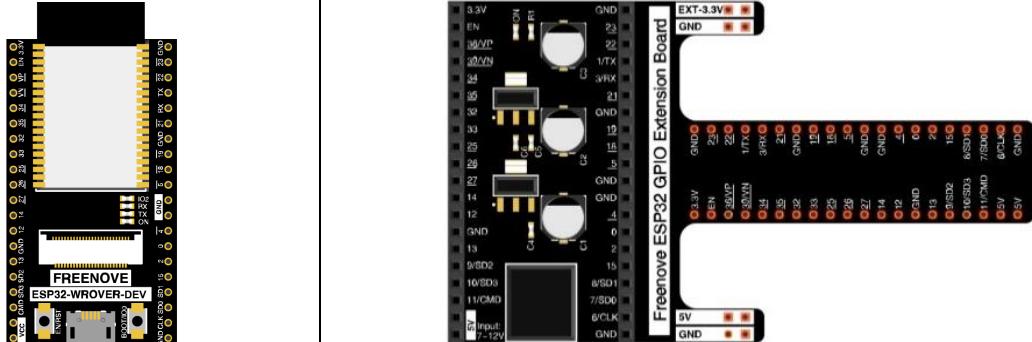
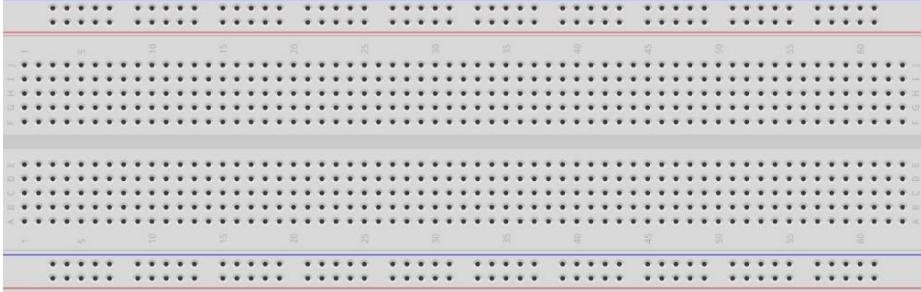
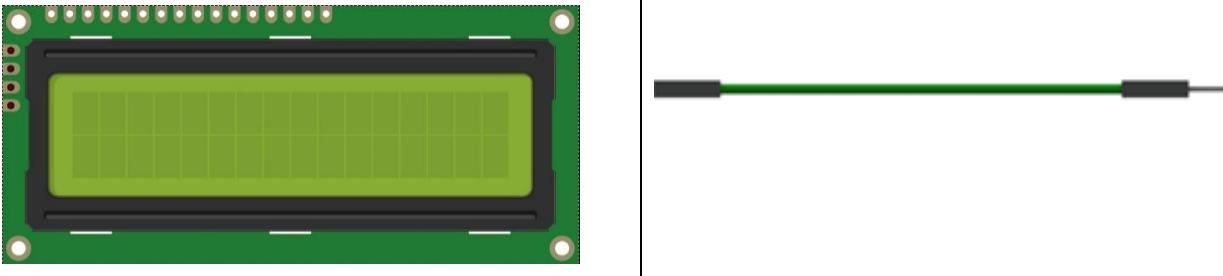
Chapter 20 LCD1602

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

Project 20.1 LCD1602

In this section we learn how to use lcd1602 to display something.

Component List

ESP32-WROVER x1	GPIO Extension Board x1
	
Breadboard x1	
LCD1602 Module x1	Jumper F/M x4
	

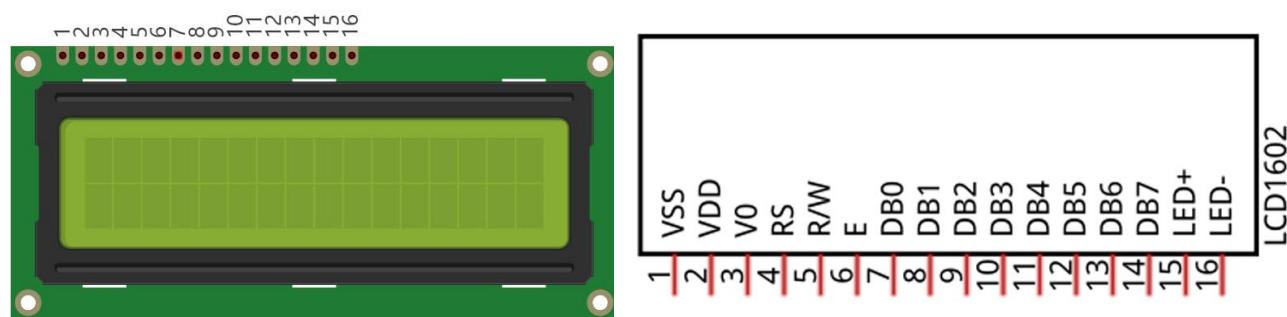
Component knowledge

I2C communication

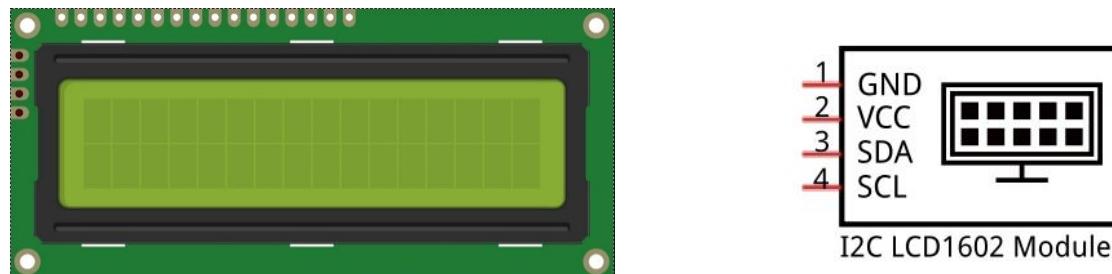
I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used for the connection of micro controllers and their peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device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

LCD1602 communication

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



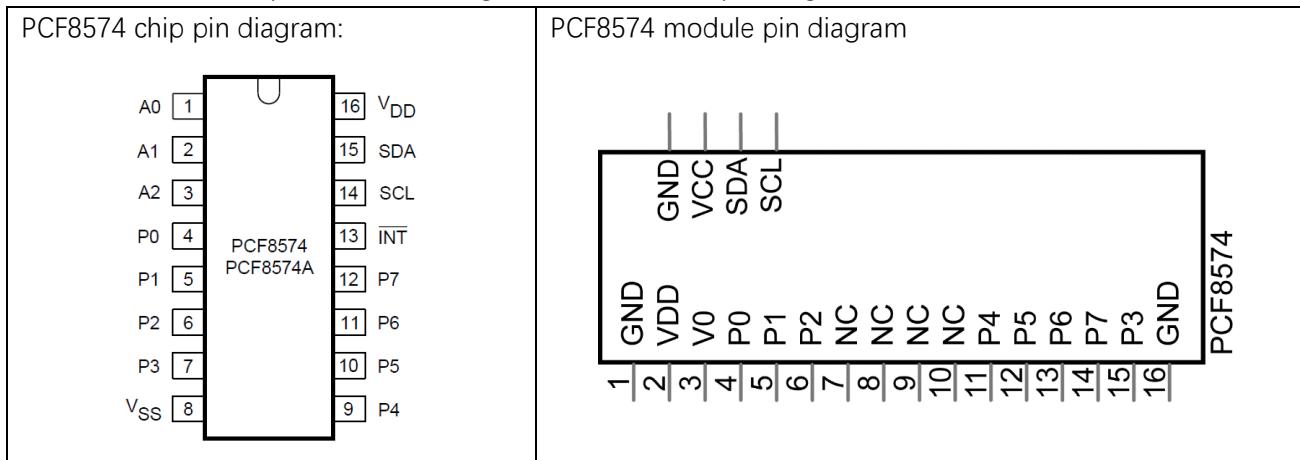
I2C LCD1602 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to use only 4 lines to operate the LCD1602.



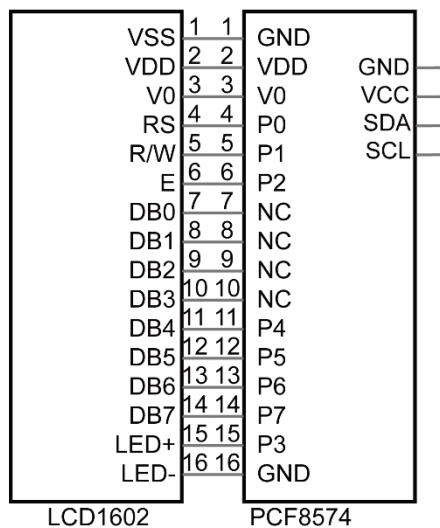
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the ESP32 bus on your I2C device address through command "i2cdetect -y 1".



Below is the PCF8574 pin schematic diagram and the block pin diagram:



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

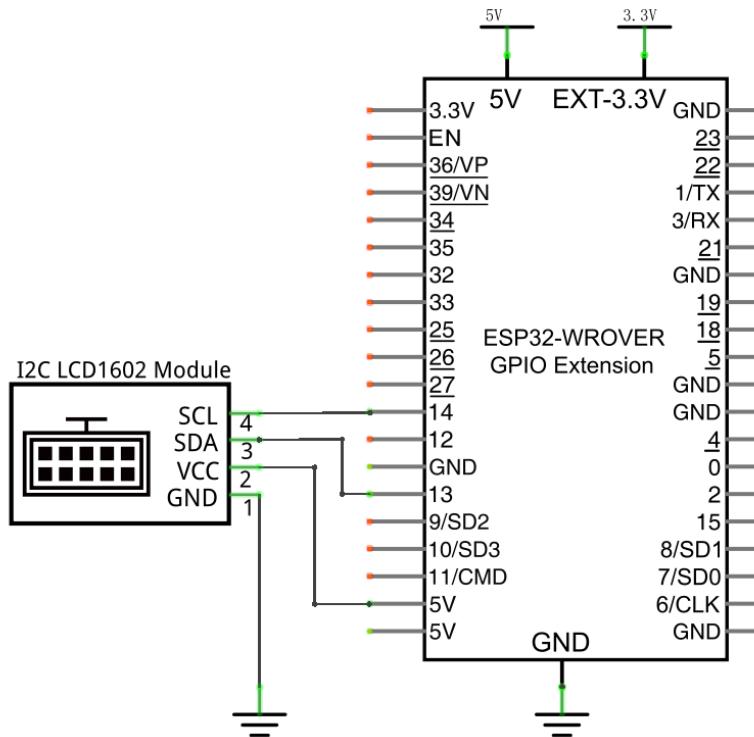


LCD1602 PCF8574

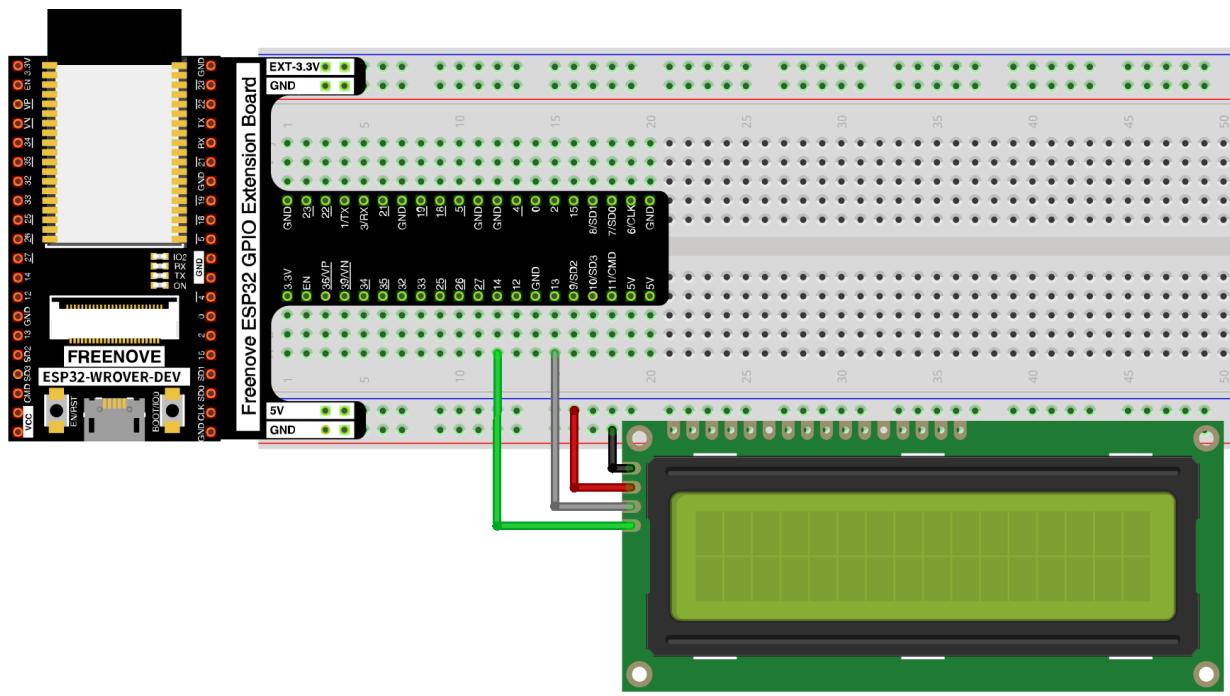
So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I₂C interface. In this project, we will use the I₂C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



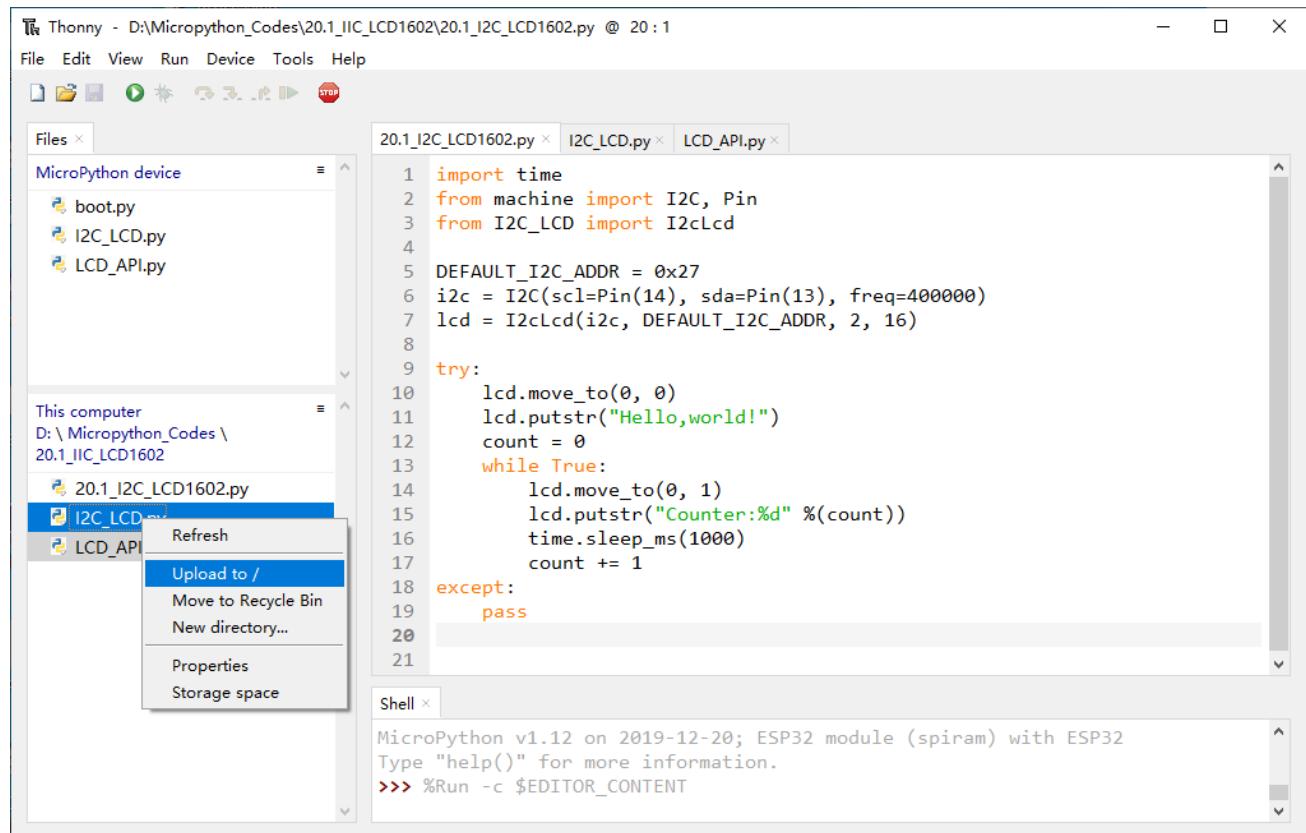
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “20.1_I2C_LCD1602”. Select “I2C_LCD.py” and “LCD_API.py”, right click your mouse to select “Upload to /”, wait for “I2C_LCD.py” and “LCD_API.py” to be uploaded to ESP32-WROVER and then double click “20.1_I2C_LCD1602.py”.

20.1 I2C_LCD1602



The screenshot shows the Thonny IDE interface. The left sidebar displays a file tree with the following structure:

```

D:\Micropython_Codes\20.1_IIC_LCD1602
  - 20.1_I2C_LCD1602.py
  - I2C_LCD.py
  - LCD_API.py

```

The right pane contains three tabs: "20.1_I2C_LCD1602.py" (selected), "I2C_LCD.py", and "LCD_API.py". The code editor shows the following Python script:

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2cLcd
4
5 DEFAULT_I2C_ADDR = 0x27
6 i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
7 lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
8
9 try:
10     lcd.move_to(0, 0)
11     lcd.putstr("Hello,world!")
12     count = 0
13     while True:
14         lcd.move_to(0, 1)
15         lcd.putstr("Counter:%d" %(count))
16         time.sleep_ms(1000)
17         count += 1
18 except:
19     pass
20
21

```

The bottom pane is a "Shell" window showing the MicroPython prompt:

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

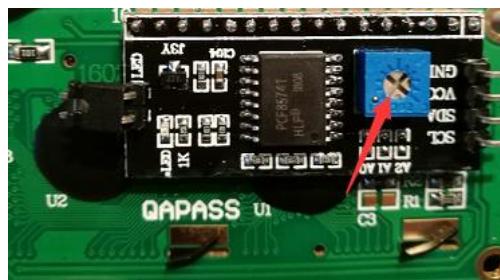
```

A context menu is open over the "I2C_LCD.py" file in the file tree, with the "Upload to /" option highlighted.

Click “Run current script” and LCD1602 displays some characters.



If you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display clearly.



The following is the program code:

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2cLcd
4
5 DEFAULT_I2C_ADDR = 0x27
6 i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
7 lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
8
9 try:
10     lcd.move_to(0, 0)
11     lcd.putstr("Hello, world!")
12     count = 0
13     while True:
14         lcd.move_to(0, 1)
15         lcd.putstr("Counter:%d" %(count))
16         time.sleep_ms(1000)
17         count += 1
18     except:
19         pass

```

Import time, I2C and I2C_LCD modules.

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2cLcd

```

Instantiate the I2C LCD1602 screen. It should be noted here that if your LCD driver chip uses PCF8574T, set the I2C address to 0x27, and if uses PCF8574AT, set the I2C address to 0x3F.

```
5 DEFAULT_I2C_ADDR = 0x27
```

Initialize I2C pins and associate them with I2CLCD module, and then set the number of rows and columns for LCD1602.

```

6 i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
7 lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

```

Move the cursor of LCD1602 to the first row, first column, and print out "Hello, world!"

```

10 lcd.move_to(0, 0)
11 lcd.putstr("Hello, world!")

```



The second line of LCD1602 continuously prints the number of seconds after the ESP32 program runs.

```
13     while True:  
14         lcd.move_to(0, 1)  
15         lcd.putstr("Counter:%d" %(count))  
16         time.sleep_ms(1000)  
17         count += 1
```

Reference

Class I2cLcd

Before each use of the object **I2cLcd**, please make sure that **I2C_LCD.py** and **LCD_API.py** have been uploaded to “/” of ESP32, and then add the statement “**from I2C_LCD import I2cLcd**” to the top of the python file.

clear(): Clear the LCD1602 screen display.

show_cursor(): Show the cursor of LCD1602.

hide_cursor(): Hide the cursor of LCD1602.

blink_cursor_on(): Turn on cursor blinking.

blink_cursor_off(): Turn off cursor blinking.

display_on(): Turn on the display function of LCD1602.

display_off(): Turn on the display function of LCD1602.

backlight_on(): Turn on the backlight of LCD1602.

backlight_off(): Turn on the backlight of LCD1602.

move_to(cursor_x, cursor_y): Move the cursor to a specified position.

cursor_x: Column cursor_x

cursor_y : Row cursor_y

putchar(char) : Print the character in the bracket on LCD1602

putstr(string) : Print the string in the bracket on LCD1602.

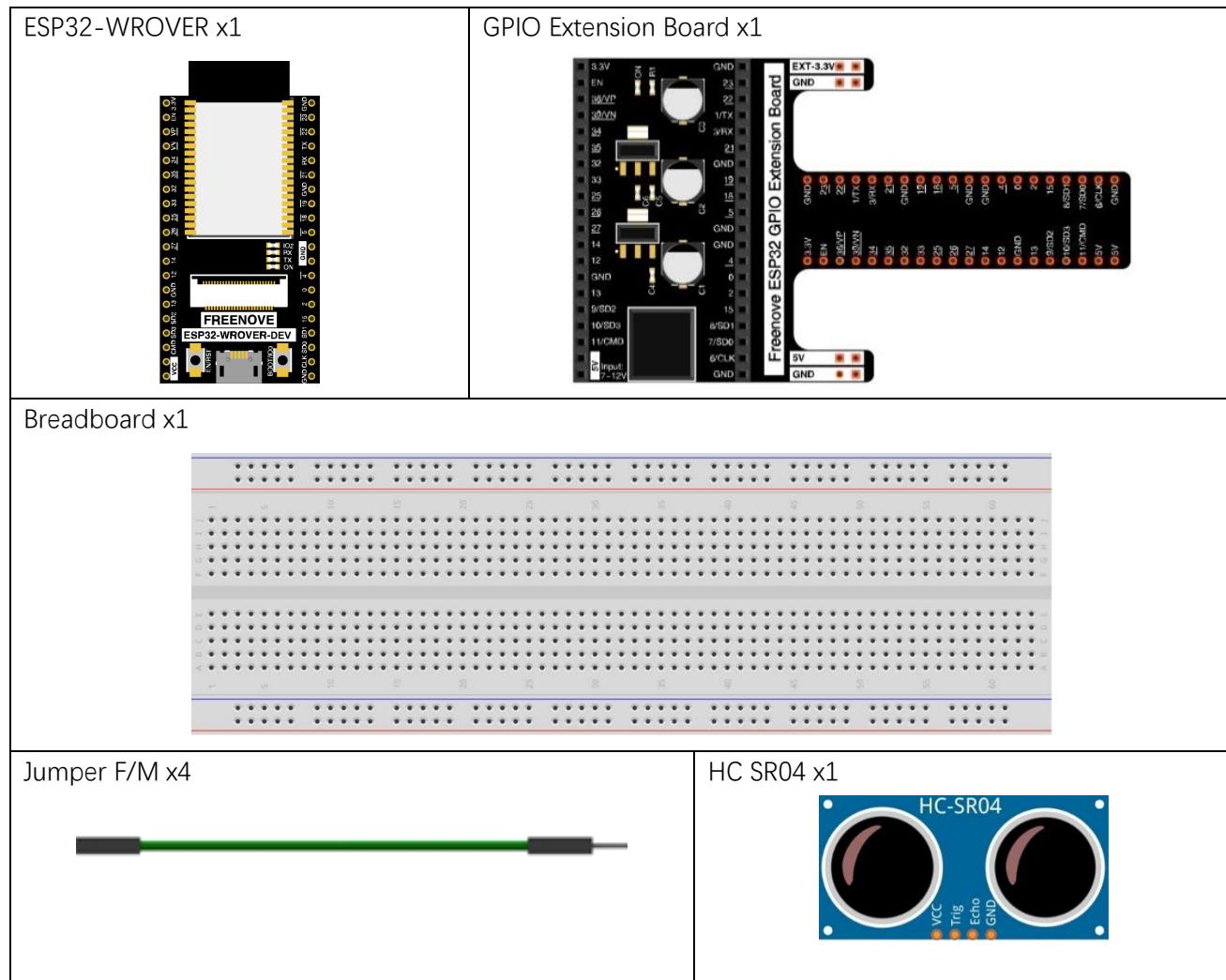
Chapter 21 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 21.1 Ultrasonic Ranging

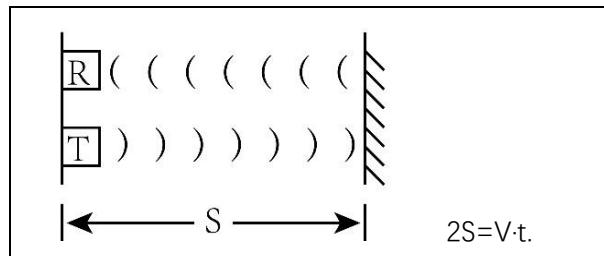
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

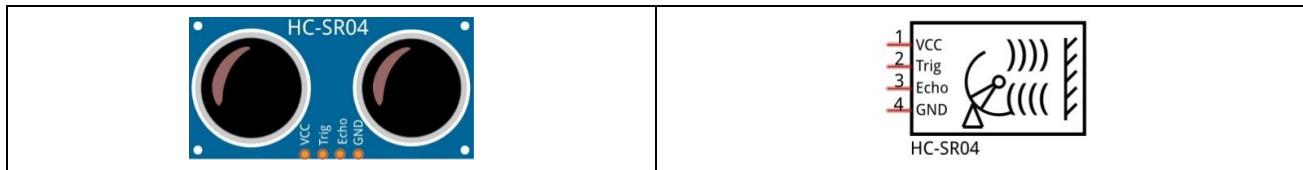


Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about $v=340\text{m/s}$, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: $s=vt/2$.



The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

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

Technical specs:

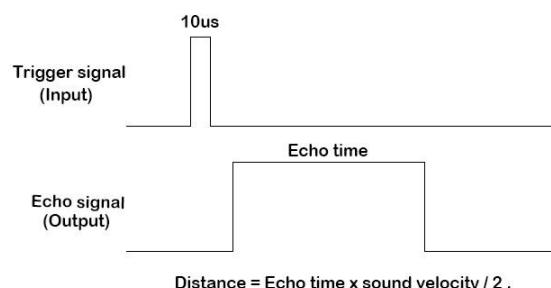
Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

Maximum measured distance: 200cm

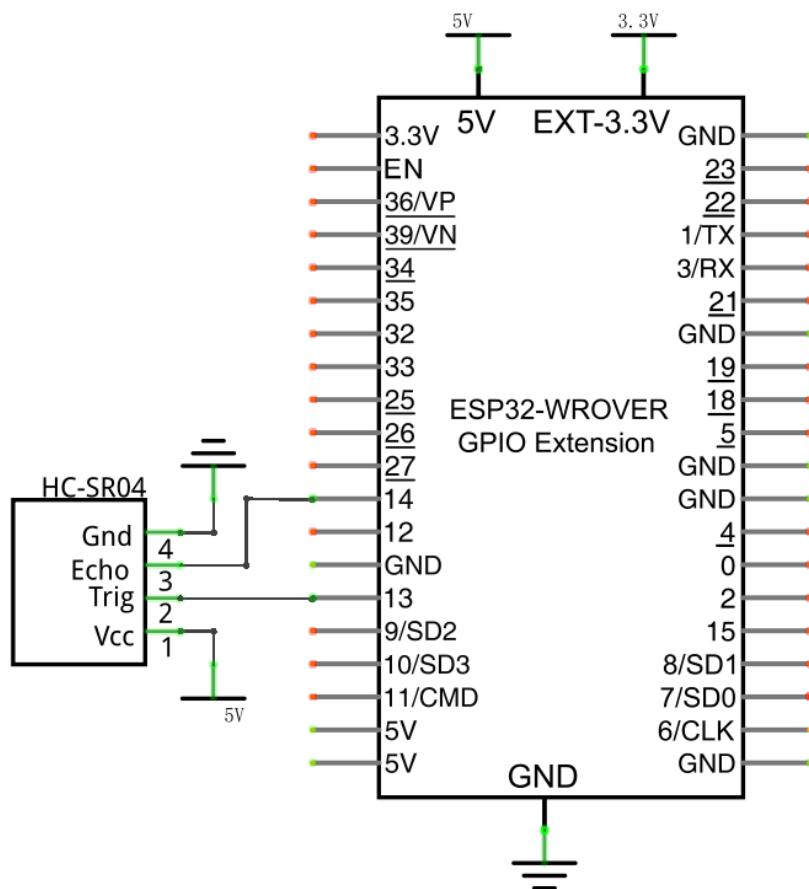
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, $s=vt/2$.



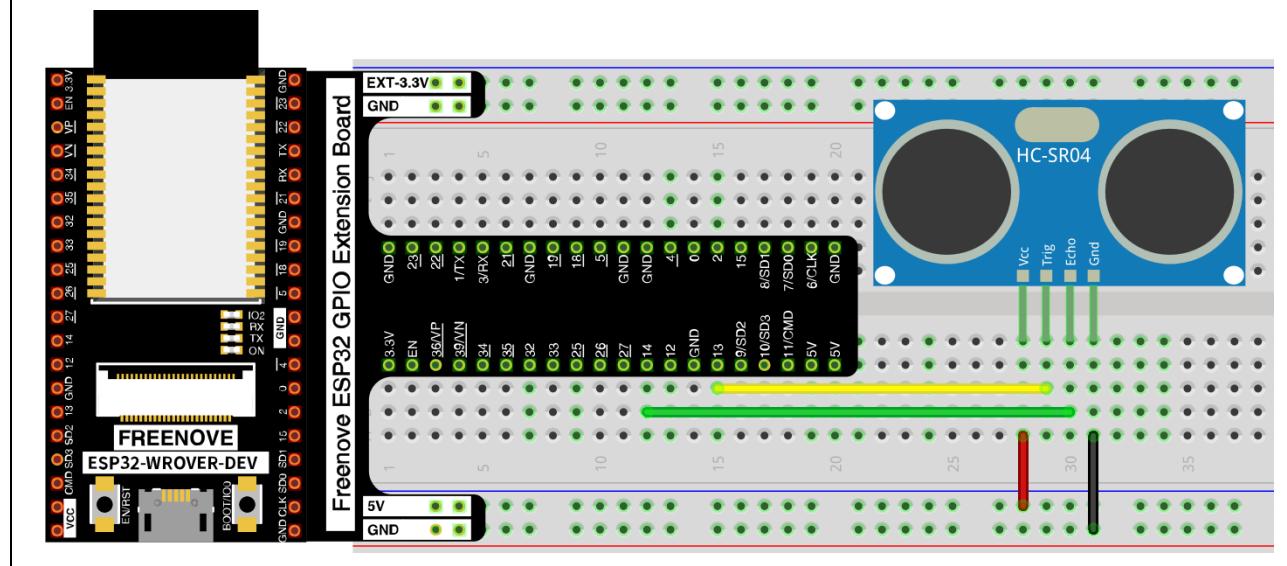
Circuit

Note that the voltage of ultrasonic module is 5V in the circuit.

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “21.1_Ultrasonic_Ranging” and double click “21.1_Ultrasonic_Ranging.py”.

21.1 Ultrasonic_Ranging

Thonny - D:\Micropython_Codes\21.1_Ultrasonic_Ranging\21.1_Ultrasonic_Ranging.py @ 38 : 5

File Edit View Run Device Tools Help

Files x

MicroPython device

- boot.py

This computer

D:\ Micropython_Codes \ 21.1_Ultrasonic_Ranging

21.1_Ultrasonic_Ranging.py

21.1_Ultrasonic_Ranging.py x

```
7 soundVelocity=340
8 distance=0
9
10 def getSonar():
11     trigPin.value(1)
12     time.sleep_us(10)
13     trigPin.value(0)
14     while not echoPin.value():
15         pass
16     pingStart=time.ticks_us()
17     while echoPin.value():
18         pass
19     pingStop=time.ticks_us()
20     pingTime=time.ticks_diff(pingStop,pingStart)
21     distance=pingTime*soundVelocity//2//10000
22     return int(distance)
23
24 time.sleep_ms(2000)
25 while True:
26     time.sleep_ms(500)
27     print('Distance: ',getSonar(),'cm' )
```

Shell x

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click “Run current script”, you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure:

```
Shell x
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Distance: 14 cm
Distance: 15 cm
Distance: 18 cm
Distance: 21 cm
Distance: 22 cm
Distance: 14 cm
Distance: 9 cm
Distance: 7 cm
Distance: 7 cm
Distance: 12 cm
```

The following is the program code:

```

1  from machine import Pin
2  import time
3
4  trigPin=Pin(13, Pin.OUT, 0)
5  echoPin=Pin(14, Pin.IN, 0)
6
7  soundVelocity=340
8  distance=0
9
10 def getSonar():
11     trigPin.value(1)
12     time.sleep_us(10)
13     trigPin.value(0)
14     while not echoPin.value():
15         pass
16     pingStart=time.ticks_us()
17     while echoPin.value():
18         pass
19     pingStop=time.ticks_us()
20     pingTime=time.ticks_diff(pingStop, pingStart)
21     distance=pingTime*sounVelocity//2//10000
22     return int(distance)
23
24 time.sleep_ms(2000)
25 while True:
26     time.sleep_ms(500)
27     print('Distance: ', getSonar(), 'cm')

```

Define the control pins of the ultrasonic ranging module.

```

4  trigPin=Pin(13, Pin.OUT, 0)
5  echoPin=Pin(14, Pin.IN, 0)

```

Set the speed of sound.

```

7  soundVelocity=340
8  distance=0

```

Subfunction `getSonar()` is used to start the Ultrasonic Module to begin measurements, and return the measured distance in centimeters. 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.

```

10 def getSonar():
11     trigPin.value(1)
12     time.sleep_us(10)
13     trigPin.value(0)
14     while not echoPin.value():
15         pass

```

```
16 pingStart=time.ticks_us()
17 while echoPin.value():
18     pass
19 pingStop=time.ticks_us()
20 pingTime=time.ticks_diff(pingStop,pingStart)
21 distance=pingTime*soundVelocity//2//10000
22 return int(distance)
```

Delay for 2 seconds and wait for the ultrasonic module to stabilize. Print data obtained from ultrasonic module every 500 milliseconds

```
24 time.sleep_ms(2000)
25 while True:
26     time.sleep_ms(500)
27     print('Distance: ',getSonar(),'cm')
```

Project 21.2 Ultrasonic Ranging

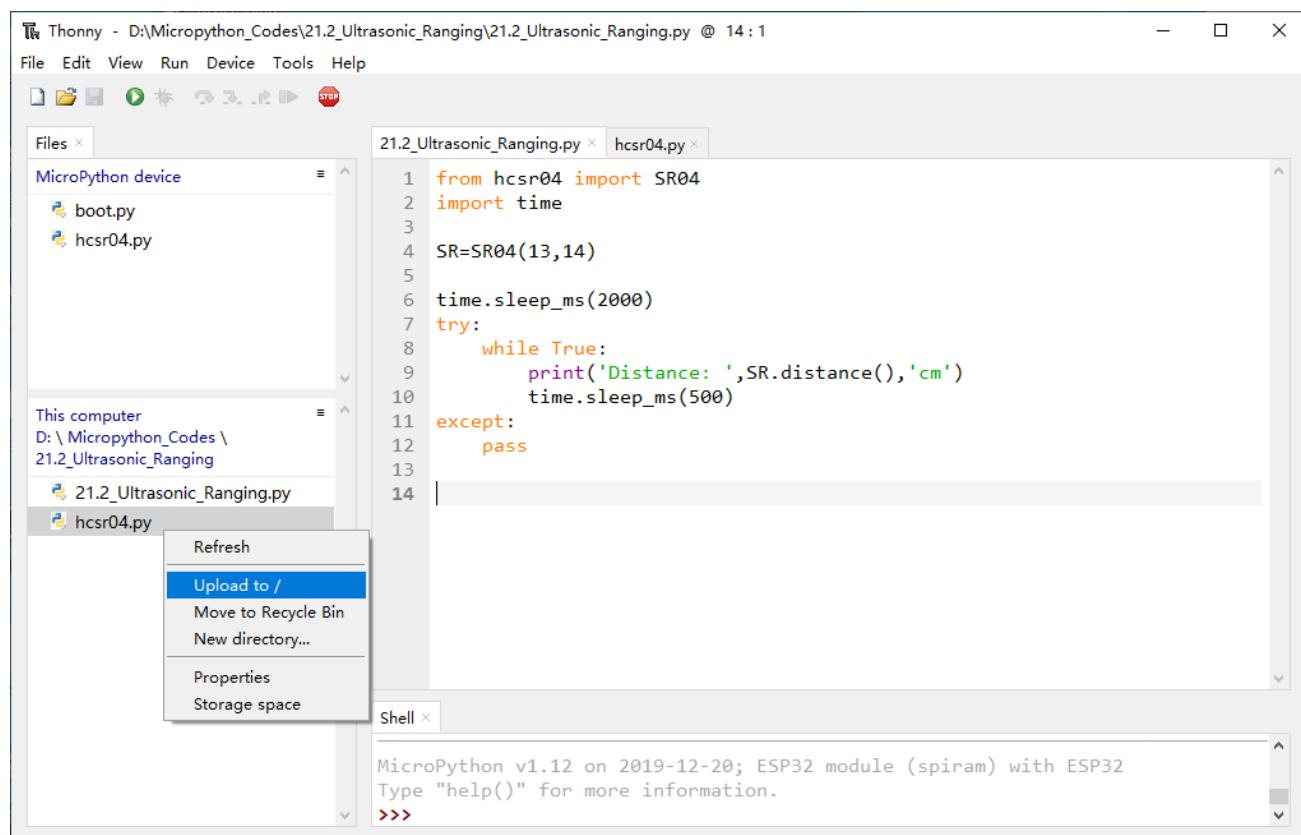
Component List and Circuit

Component List and Circuit are the same as the previous section.

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “21.2_Ultrasonic_Ranging”. Select “hcsr04.py”, right click your mouse to select “Upload to /”, wait for “hcsr04.py” to be uploaded to ESP32-WROVER and then double click “21.2_Ultrasonic_Ranging.py”.

21.2 Ultrasonic_Ranging



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\21.2_Ultrasonic_Ranging\21.2_Ultrasonic_Ranging.py @ 14 : 1". The menu bar includes File, Edit, View, Run, Device, Tools, Help. The toolbar has icons for file operations like Open, Save, Run, Stop, and Device. The left sidebar shows a tree view of files: "MicroPython device" contains "boot.py" and "hcsr04.py"; "This computer" shows "D:\ Micropython_Codes \ 21.2_Ultrasonic_Ranging" which contains "21.2_Ultrasonic_Ranging.py" and "hcsr04.py". A context menu is open over "hcsr04.py" with options: Refresh, Upload to / (which is highlighted in blue), Move to Recycle Bin, New directory..., Properties, and Storage space. The main code editor window displays the following Python code:

```
1 from hcsr04 import SR04
2 import time
3
4 SR=SR04(13,14)
5
6 time.sleep_ms(2000)
7 try:
8     while True:
9         print('Distance: ',SR.distance(),'cm')
10        time.sleep_ms(500)
11 except:
12     pass
13
14 |
```

The bottom shell window shows the MicroPython prompt: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32" and "Type "help()" for more information." with a ">>>" prompt.

Click “Run current script”. Use the ultrasonic module to measure distance. As shown in the following figure:

```
Shell ×
Distance: 6.647 cm
Distance: 5.151 cm
Distance: 5.151 cm
Distance: 6.052 cm
Distance: 7.225 cm
Distance: 7.531 cm
Distance: 8.721 cm
Distance: 12.121 cm
Distance: 11.798 cm
Distance: 13.6 cm
Distance: 14.467 cm
Distance: 16.116 cm
Distance: 17.442 cm
Distance: 19.652 cm
Distance: 22.015 cm
```

The following is the program code:

```
1 from hcsr04 import SR04
2 import time
3
4 SR=SR04(13, 14)
5
6 time.sleep_ms(2000)
7 try:
8     while True:
9         print('Distance: ',SR.distance(), 'cm')
10        time.sleep_ms(500)
11    except:
12        pass
```

Import hcsr04 module.

```
1 from hcsr04 import SR04
```

Define an ultrasonic object and associate with the pins.

```
3 SR=SR04(13, 14)
```

Obtain the distance data returned from the ultrasonic ranging module.

```
9 SR.distance()
```

Obtain the ultrasonic data every 500 milliseconds and print them out in “Shell”.

```
8 while True:
9     print('Distance: ',SR.distance(), 'cm')
10    time.sleep_ms(500)
```

Reference

Class hcsr04

Before each use of object **SR04**, please add the statement “**from hcsr04 import SR04**” to the top of python file.

SR04(): Object of ultrasonic module. By default, trig pin is Pin(13) and echo pinis Pin(14).

distanceCM(): Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being cm.

distanceMM(): Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being mm.

distance(): Obtain the distance from the ultrasonic to the measured object with the data type being float type, and the unit being cm.



Chapter 22 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 22.1 Matrix Keypad

In this project, we will attempt to get every key code on the Matrix Keypad to work.

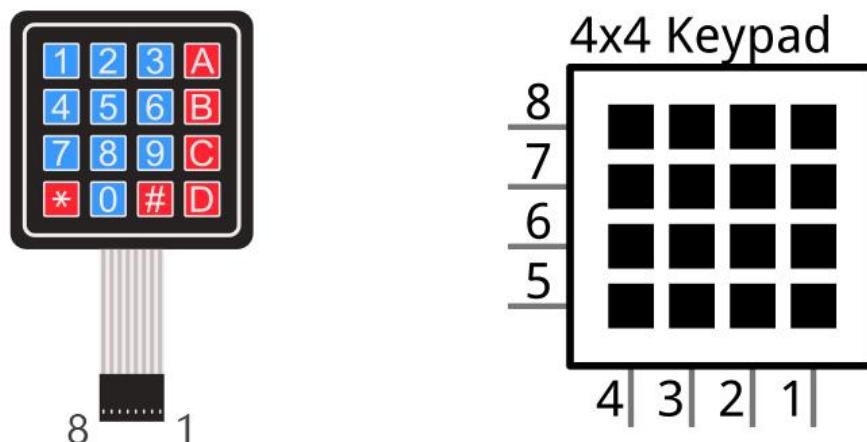
Component List

ESP32-WROVER x1	GPIO Extension Board x1
Breadboard x1	
Jumper M/M x8	4x4 Matrix Keypad x1

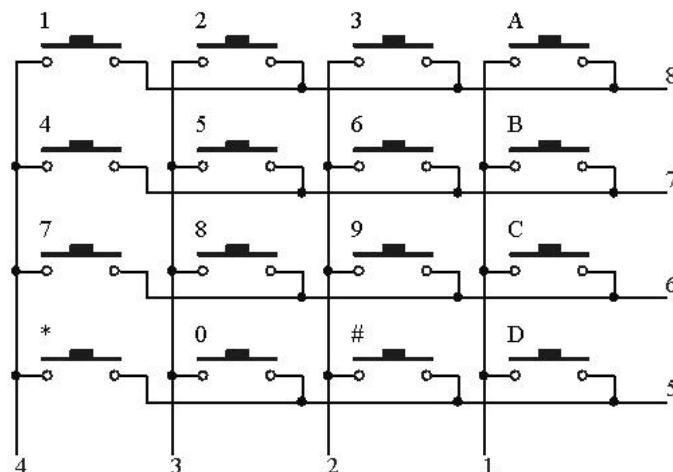
Component knowledge

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:



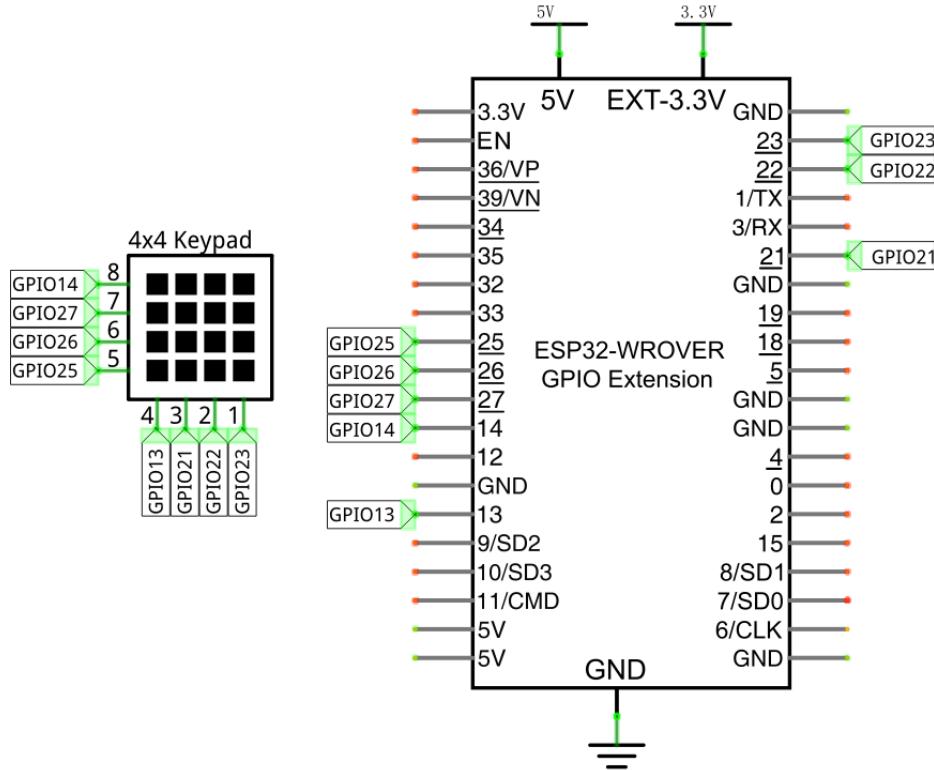
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



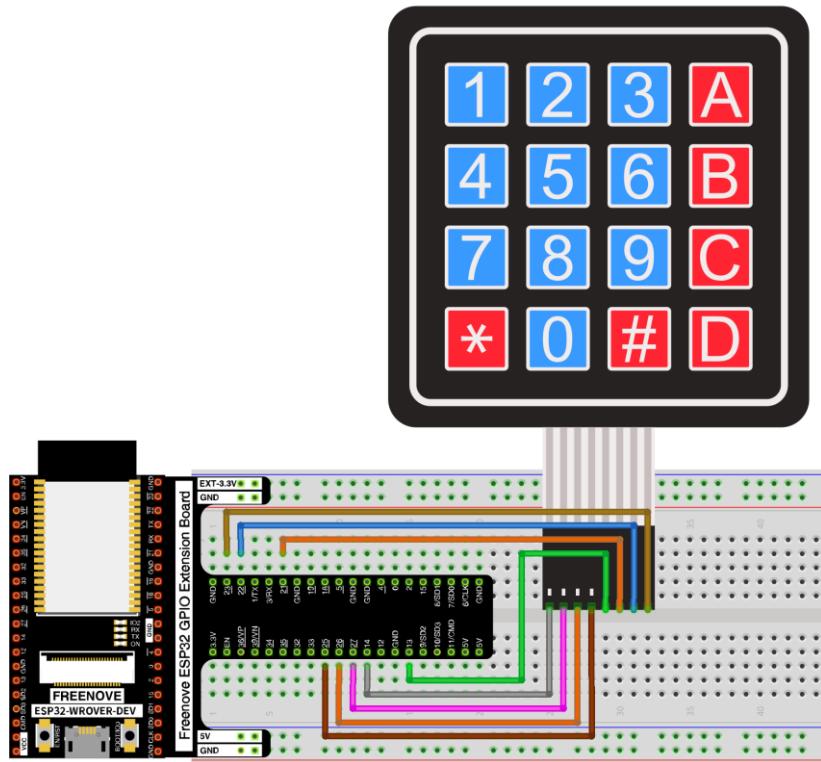
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. By this means, you can get the state of all of the keys.

Circuit

Schematic diagram



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

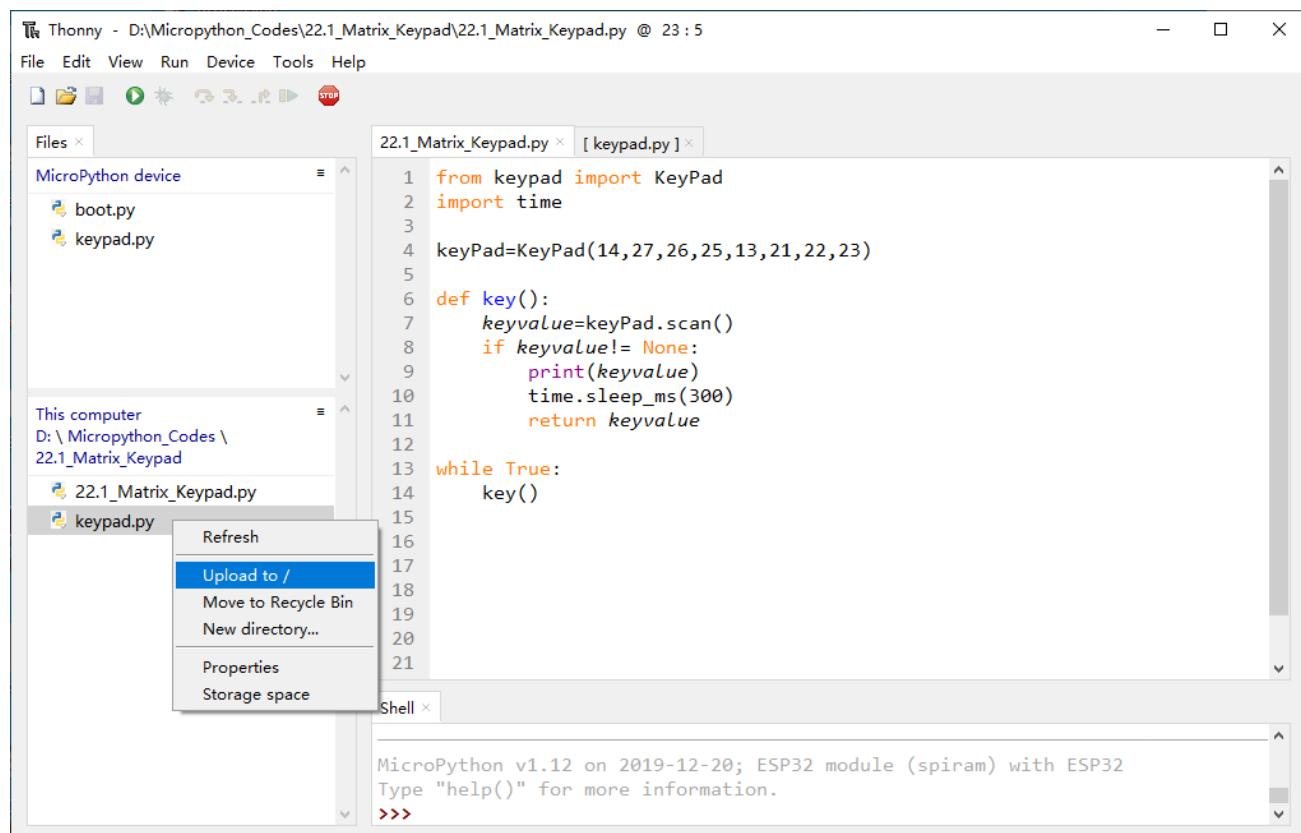


Code

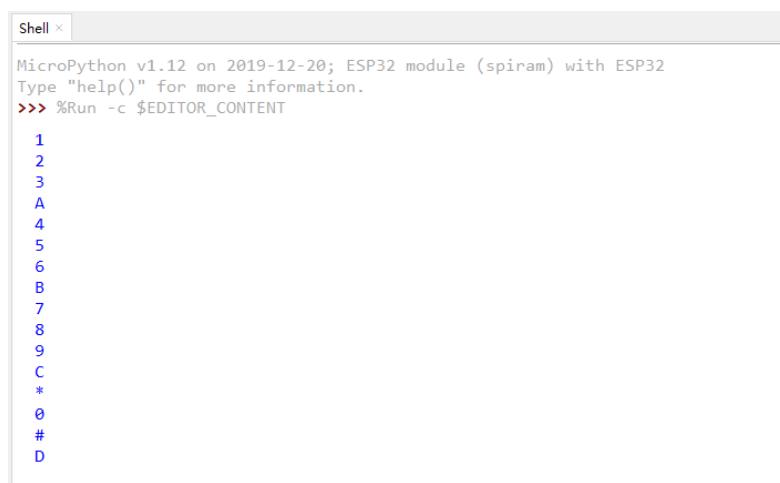
This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be printed out via serial port.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “22.1_Matrix_Keypad”. Select “keypad.py”, right click your mouse to select “Upload to /”, wait for “keypad.py” to be uploaded to ESP32-WROVER and then double click “22.1_Matrix_Keypad.py”.

22.1 Get_Input_Characters



Click “Run current script”, push the key board and the key value will be printed in “Shell”. As shown in the illustration below:





The following is the program code:

```

1  from keypad import KeyPad
2  import time
3
4  keyPad=KeyPad(14, 27, 26, 25, 13, 21, 22, 23)
5
6  def key():
7      keyvalue=keyPad.scan()
8      if keyvalue!= None:
9          print(keyvalue)
10         time.sleep_ms(300)
11         return keyvalue
12
13 while True:
14     key()

```

Import keypad module.

```
1  from keypad import KeyPad
```

Associate the keypad module to ESP32 pins.

```
4  keyPad=KeyPad(14, 27, 26, 25, 13, 21, 22, 23)
```

Call function `keypad.scan()` of the keypad module. When the keypad module detects that the key is pressed, it returns the value of the pressed key; when no key is pressed, the return value is `None`.

```
7  keyPad.scan()
```

Call function `keyPan.scan()` to obtain the value of the pressed key. Once it is obtained, print it out.

```

6  def key():
7      keyvalue=keyPad.scan()
8      if keyvalue!= None:
9          print(keyvalue)
10         time.sleep_ms(300)
11         return keyvalue

```

Reference

Class keypad

Before each use of the object `KeyPad`, please make sure `keypad.py` has been uploaded to "/" of ESP32 and then add the statement "`from keypad import KeyPad`" to the top of python file.

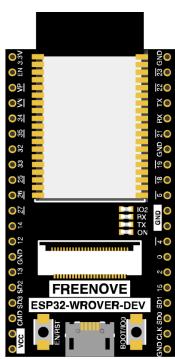
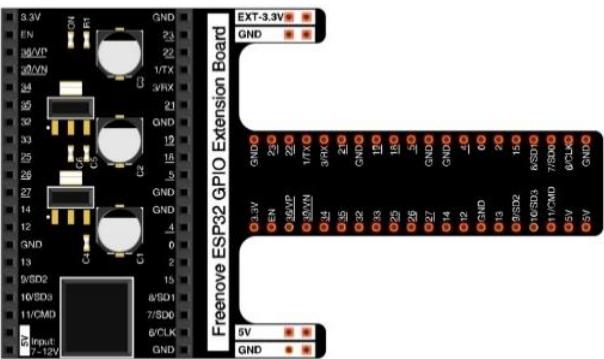
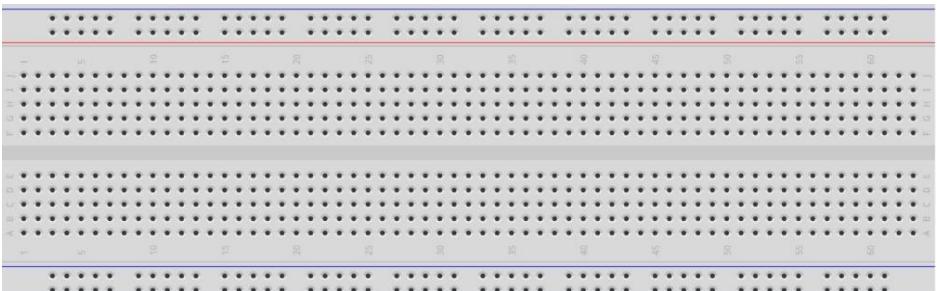
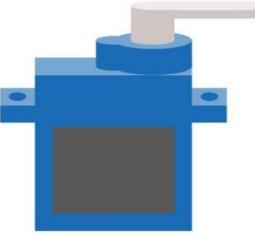
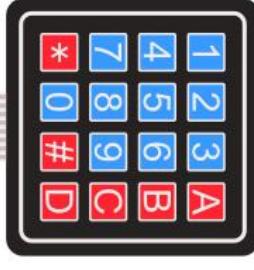
KeyPad(row1, row2, row3, row4, col1, col2, col3, col4): Intialize keypad module and associate its pins with ESP32.

scan(): Non-blocking keypad scan function. If no key is pressed, it returns `None`; Otherwise, it returns the valued of the pressed key.

Project 22.2 Keypad Door

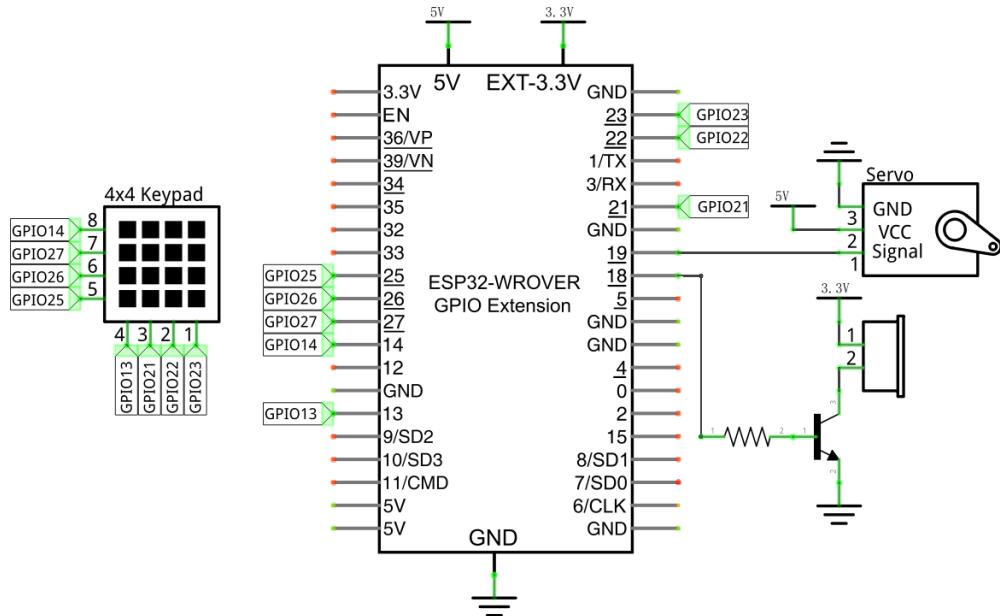
In this project, we use keypad as a keyboard to control the action of the servo motor.

Component List

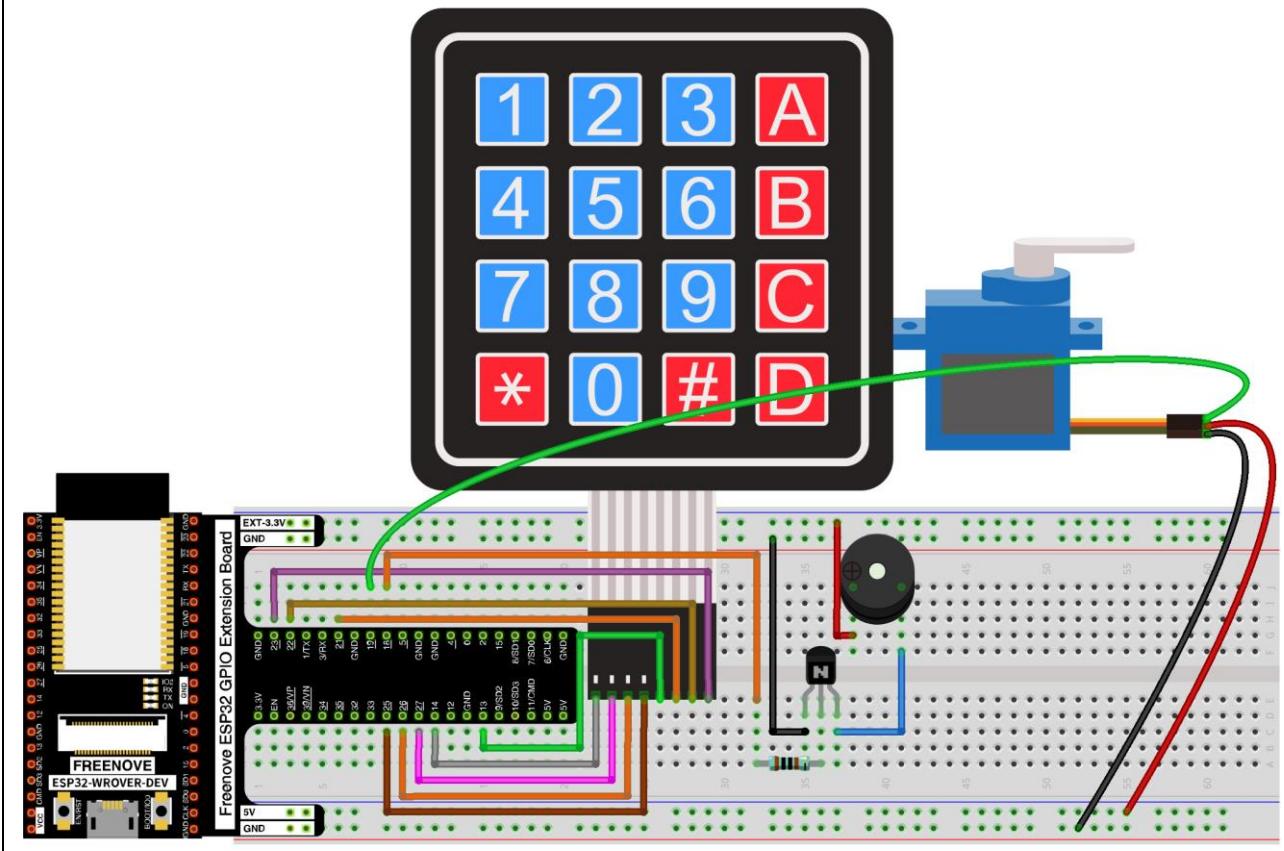
ESP32-WROVER x1	GPIO Extension Board x1	Breadboard x1
		
Jumper M/M	Servo x1	4x4 Matrix Keypad x1
		
NPN transistor x1 (S8050)	Active buzzer x1	Resistor 1kΩ x1
		

Circuit

Schematic diagram



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



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “22.2_Keypad_Door”. Select “keypad.py” and “myservo.py”, right click your mouse to select “Upload to /”, wait for “keypad.py” and “myservo.py” to be uploaded to ESP32-WROVER and then double click “22.2_Keypad_Door.py”.

22.2 Keypad_Door

```
21 while True:
22     keydata=key()
23     if keydata!=None:
24         activeBuzzer.value(1)
25         time.sleep_ms(100)
26         activeBuzzer.value(0)
27         keyIn[keyInNum]=keydata
28         keyInNum=keyInNum+1
29
30     if keyInNum==4:
31         if keyIn==keyOut:
32             print("passWord right!")
33             servo.myServoWriteAngle(90)
34             time.sleep_ms(1000)
35             servo.myServoWriteAngle(0)
36         else:
37             print("passWord error!")
38             activeBuzzer.value(1)
39             time.sleep_ms(1000)
40             activeBuzzer.value(0)
41         keyInNum=0
```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
=>>>

Click “Run current script”, press the keypad to input password with 4 characters. If the input is correct, the servo will move to a certain degree, and then return to the original position. If the input is wrong, an input error alarm will be generated.



The following is the program code:

```
1  from myservo import myServo
2  from keypad import KeyPad
3  from machine import Pin
4  import time
5
6  keyPad=KeyPad(14, 27, 26, 25, 13, 21, 22, 23)
7  servo=myServo(19)
8  servo.myServoWriteAngle(0)
9  activeBuzzer=Pin(18, Pin.OUT)
10
11 keyOut = [ '1', '2', '3', '4' ]
12 keyIn  = [ '', '', '', '' ]
13 def key():
14     keyvalue=keyPad.scan()
15     if keyvalue!= None:
16         print(' Your input:',keyvalue)
17         time.sleep_ms(200)
18         return keyvalue
19
20 keyInNum=0
21 while True:
22     keydata=key()
23     if keydata!=None:
24         activeBuzzer.value(1)
25         time.sleep_ms(100)
26         activeBuzzer.value(0)
27         keyIn[keyInNum]=keydata
28         keyInNum=keyInNum+1
29
30     if keyInNum==4:
31         if keyIn==keyOut:
32             print("passWord right!")
33             servo.myServoWriteAngle(90)
34             time.sleep_ms(1000)
35             servo.myServoWriteAngle(0)
36         else:
37             print("passWord error!")
38             activeBuzzer.value(1)
39             time.sleep_ms(1000)
40             activeBuzzer.value(0)
41         keyInNum=0
```

Define an array and set the password.

```
11 keyOut = ['1', '2', '3', '4']
```

Each time a key is pressed, the buzzer will short beep once, and the key value of the key will be stored in the keyIn array.

```
22 keydata=key()
23 if keydata!=None:
24     activeBuzzer.value(1)
25     time.sleep_ms(100)
26     activeBuzzer.value(0)
27     keyIn[keyInNum]=keydata
28     keyInNum=keyInNum+1
```

When 4 keys are pressed, it will judge whether the password is correct. If it is correct, the servo will rotate 90 degrees, and then turn back after 1 second. If the password is wrong, the buzzer will long beep once and the keyInNum value will be cleared.

```
30 if keyInNum==4:
31     if keyIn==keyOut:
32         print("passWord right!")
33         servo.myServoWriteAngle(90)
34         time.sleep_ms(1000)
35         servo.myServoWriteAngle(0)
36     else:
37         print("passWord error!")
38         activeBuzzer.value(1)
39         time.sleep_ms(1000)
40         activeBuzzer.value(0)
41     keyInNum=0
```

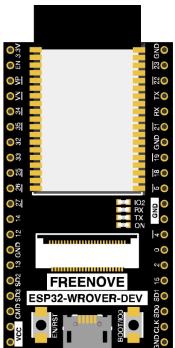
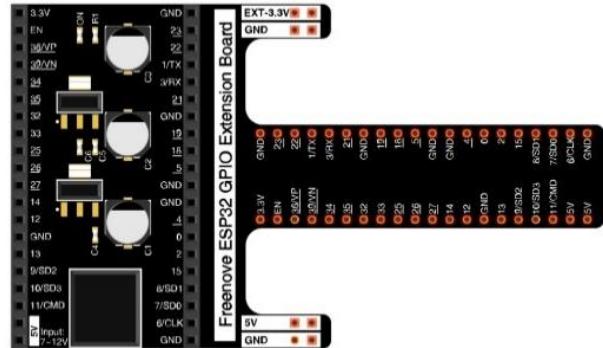
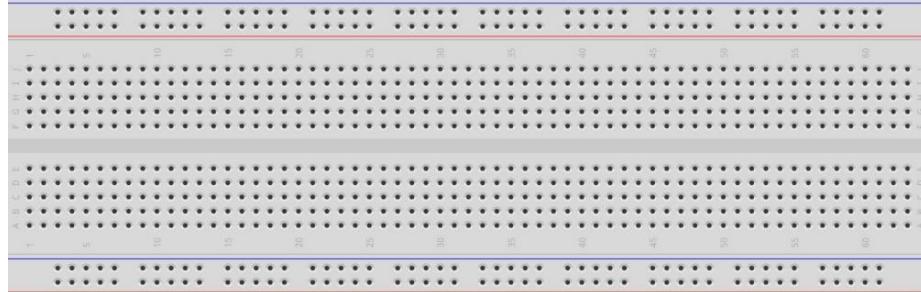
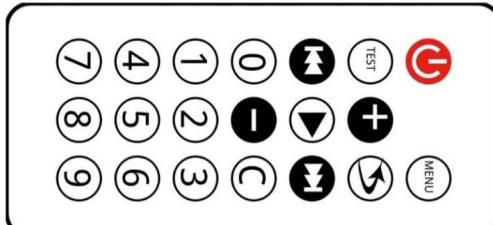
Chapter 23 Infrared Remote

In this chapter, we'll learn how to use an infrared remote control, and control a LED.

Project 23.1 Infrared Remote Control

First, we need to understand how infrared remote control works, then get the command sent from infrared remote control.

Component List

ESP32-WROVER x1	GPIO Extension Board x1	
		
		
Breadboard x1		
Jumper M/M x4	Infrared Remote x1 (May need CR2025 battery x1, please check the holder)	
		
Infrared Remote x1	Resistor 10kΩ x1	

Component knowledge

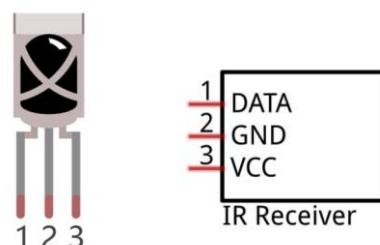
Infrared Remote

An infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared ray with different command. Infrared remote control technology is widely used in electronic products such as TV, air conditioning, etc. Thus making it possible for you to switch TV programs and adjust the temperature of the air conditioning when away from them. The remote control we use is shown below:



Infrared receiver

An infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.

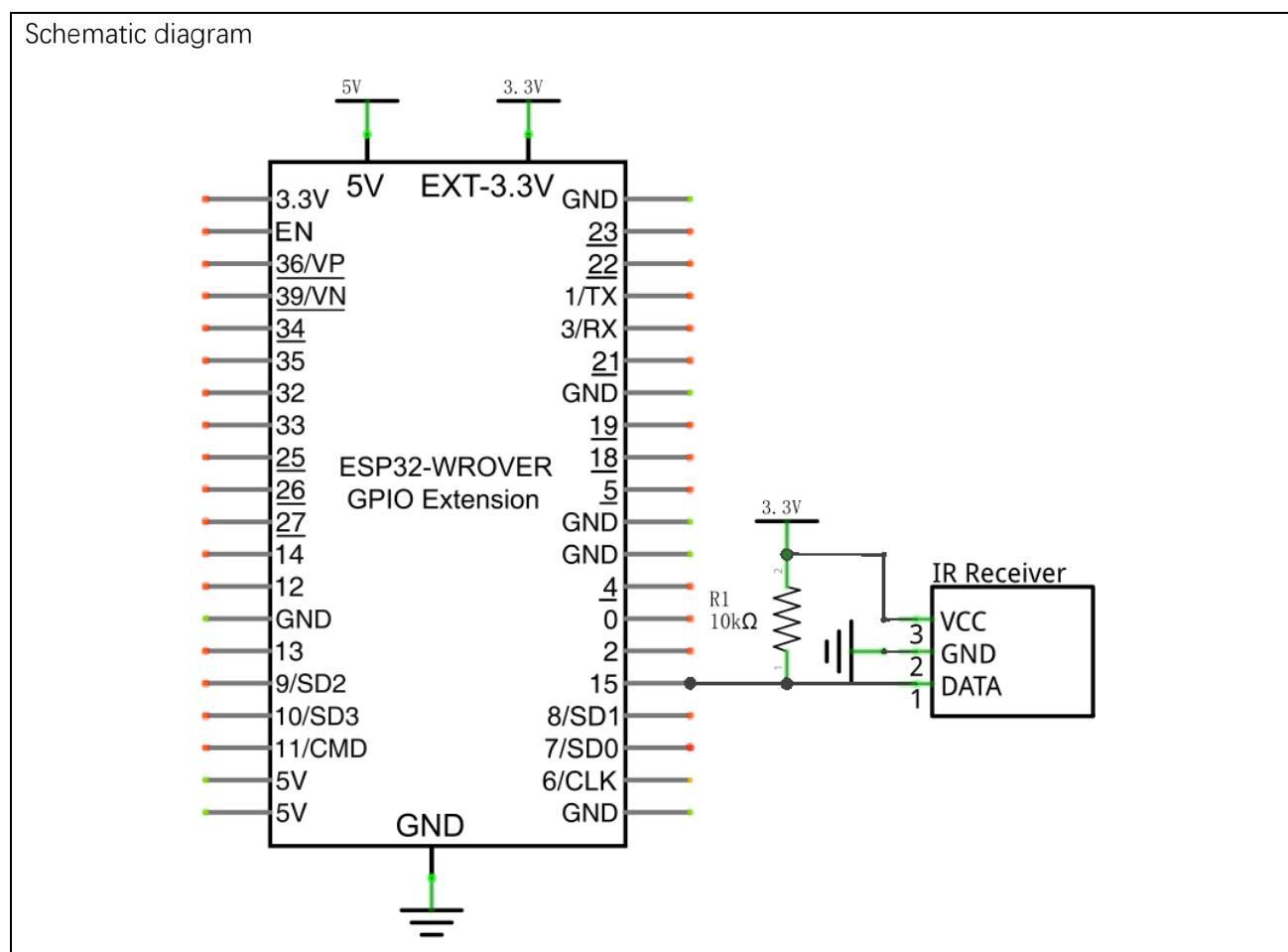


When you use the infrared remote control, the infrared remote control sends a key value to the receiving circuit according to the pressed keys. We can program the ESP32-WROVER to do things like lighting, when a key value is received.

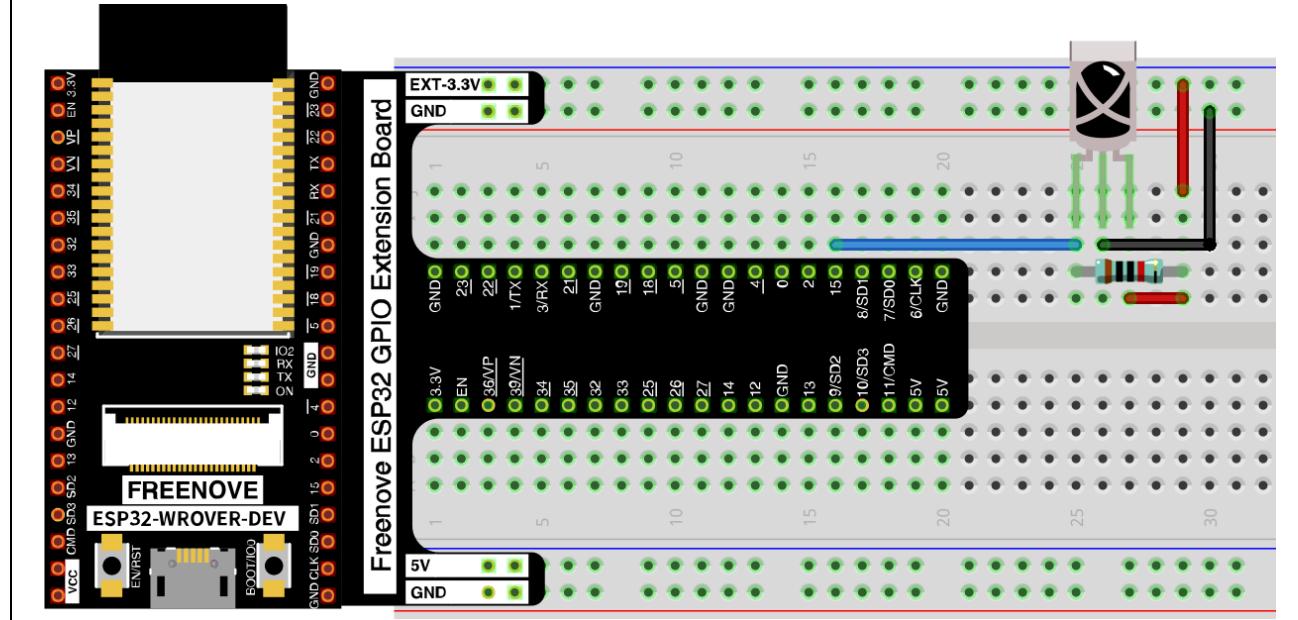
The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

ICON	KEY Value	ICON	KEY Value
	FFA25D		FFB04F
	FFE21D		FF30CF
	FF22DD		FF18E7
	FF02FD		FF7A85
	FFC23D		FF10EF
	FFE01F		FF38C7
	FFA857		FF5AA5
	FF906F		FF42BD
	FF6897		FF4AB5
	FF9867		FF52AD

Circuit



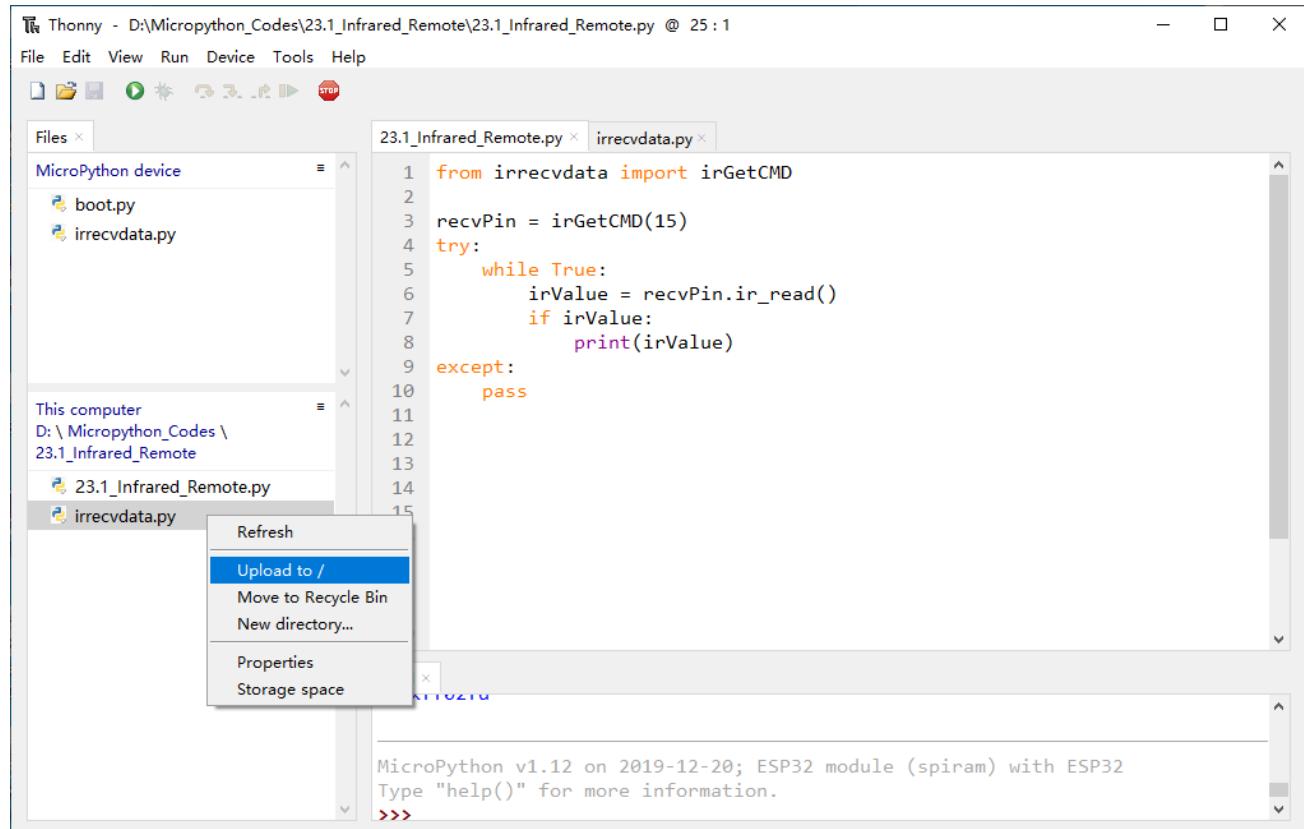
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “23.1_Infrared_Remote”. Select “irrecvdata.py”, right click your mouse to select “Upload to /”, wait for “irrecvdata.py” to be uploaded to ESP32-WROVER and then double click “23.1_Infrared_Remote.py”.

23.1 Infrared_Remote



Click “Run current script”, press the key of the infrared remote and the key value will be printed in “Shell”. As shown in the illustration below:

```

Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

0xff30cf
0xff18e7
0xff7a85
0xff10ef
0xff38c7
0xff5aa5
0xff42bd
0xff4ab5
0xff52ad
0xff6897
0xff9867

```

The following is the program code:

```
1  from irrecvdata import irGetCMD  
2  
3  recvPin = irGetCMD(15)  
4  try:  
5      while True:  
6          irValue = recvPin.ir_read()  
7          if irValue:  
8              print(irValue)  
9  except:  
10     pass
```

Import the infrared decoder.

```
1  from irrecvdata import irGetCMD
```

Associate the infrared decoder with Pin(15).

```
6  recvPin = irGetCMD(15)
```

Call `ir_read()` to read the value of the pressed key and assign it to `IRValue`.

```
12  irValue = recvPin.ir_read()
```

When infrared key value is obtained, print it out in "Shell".

```
5  while True:  
6      irValue = recvPin.ir_read()  
7      if irValue:  
8          print(irValue)
```

Reference

Class `irrecvdata`

Before each use of the object `irrecvdata`, please add the statement "`from irrecvdata import irGetCMD`" to the top of the python file.

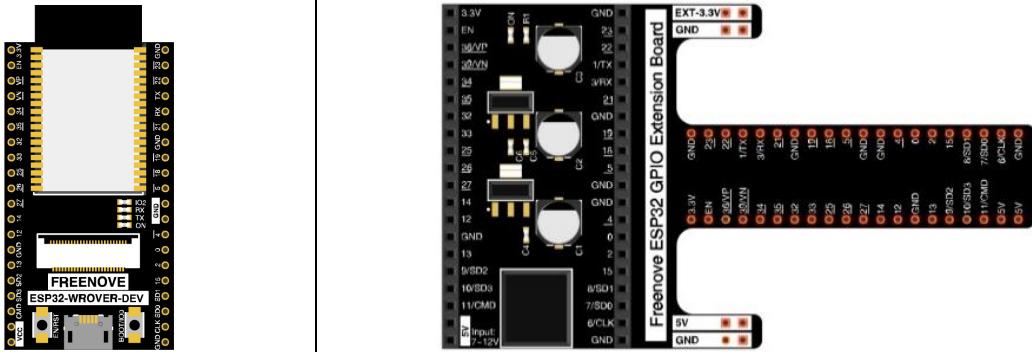
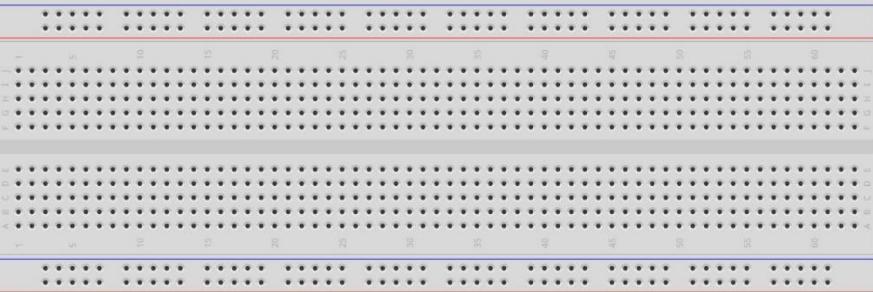
irGetCMD(): Object of infrared encoder, which is associated with Pin(15) by default.

ir_read(): The function that reads the key value of infrared remote. When the value is read, it will be returned; when no value is obtained, character **None** will be returned.

Project 23.2 Control LED through Infrared Remote

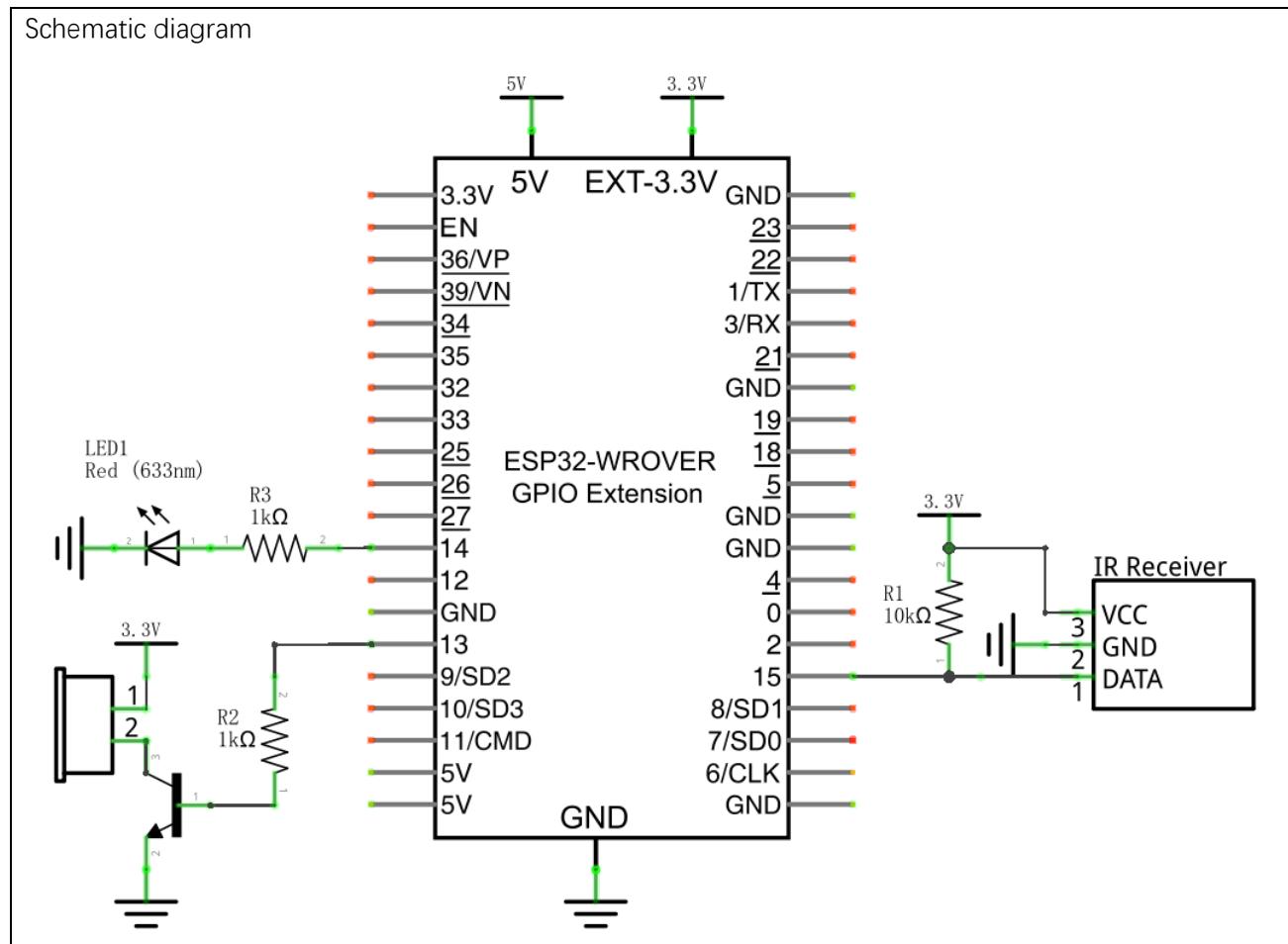
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

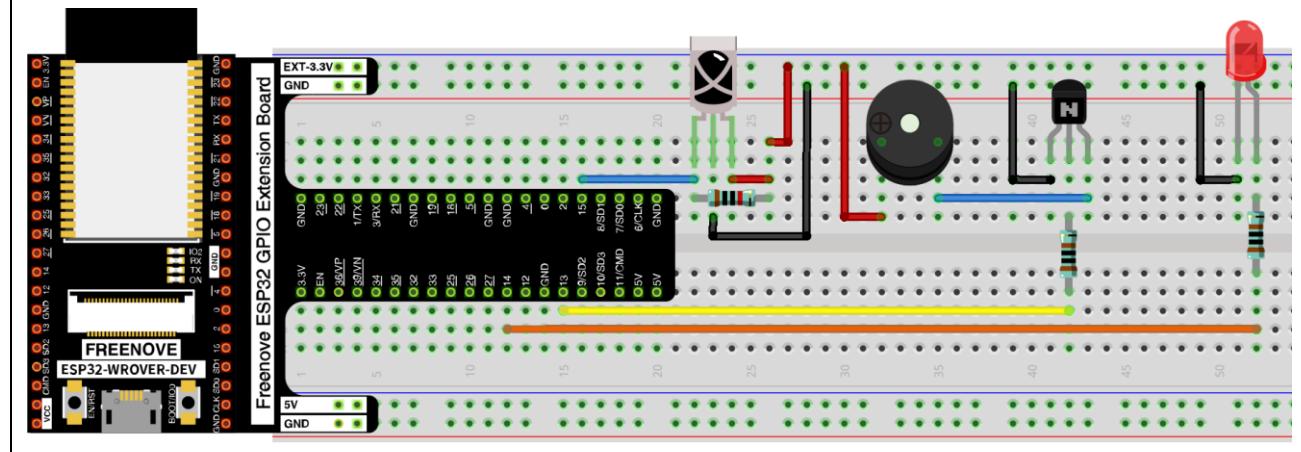
ESP32-WROVER x1	GPIO Extension Board x1
	
Breadboard x1	
	
Jumper M/M x10	Infrared Remote x1 (May need CR2025 battery x1, please check the holder)
LED x1	Resistor 1kΩ x2
	
Infrared receiver x1	Resistor 10kΩ x1
	

Circuit

Schematic diagram



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

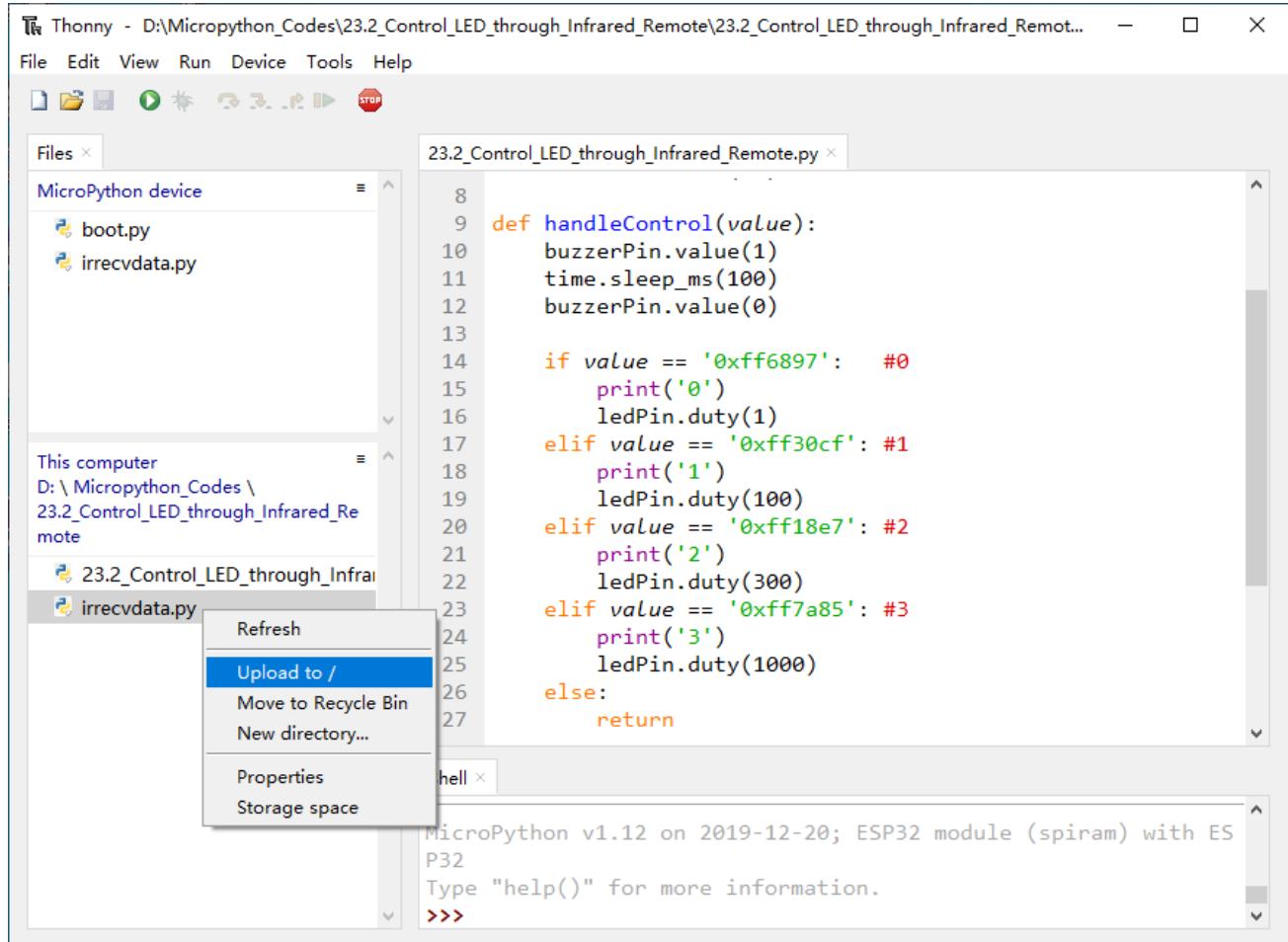


Code

The Code controls the brightness of the LED by determining the key value of the infrared received.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "23.2_Control_LED_through_Infrared_Remote". Select "irrecvdata.py", right click your mouse to select "Upload to /", wait for "irrecvdata.py" to be uploaded to ESP32-WROVER and then double click "23.2_Control_LED_through_Infrared_Remote.py".

23.2 Control_LED_through_Infrared_Remote



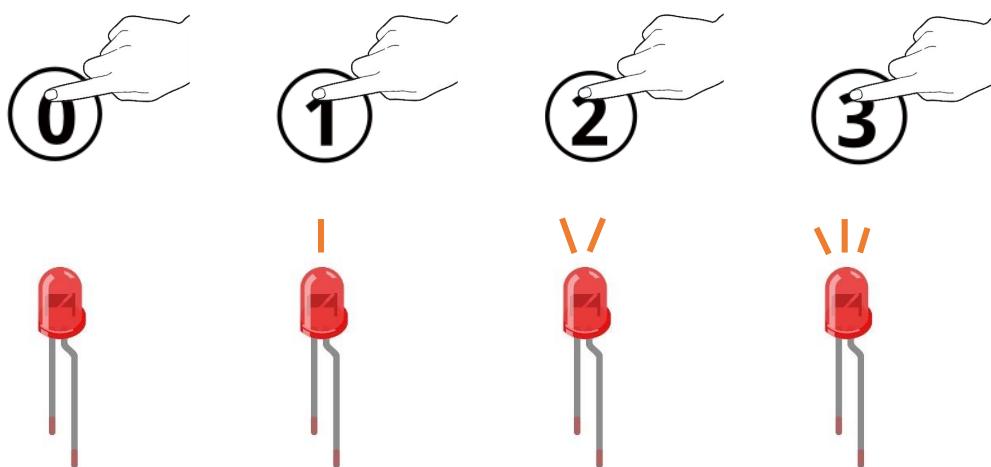
The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\23.2_Control_LED_through_Infrared_Remote\23.2_Control_LED_through_Infrared_Remote.py". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar shows a tree view of files under "MicroPython device": boot.py and irrecvdata.py. Under "This computer", it shows the directory structure: D:\ Micropython_Codes \ 23.2_Control_LED_through_Infrared_Remote. A context menu is open over "irrecvdata.py", with "Upload to /" highlighted in blue. The main code editor window displays the following Python code:

```
8 def handleControl(value):
9     buzzerPin.value(1)
10    time.sleep_ms(100)
11    buzzerPin.value(0)
12
13
14    if value == '0xff6897': #0
15        print('0')
16        ledPin.duty(1)
17    elif value == '0xff30cf': #1
18        print('1')
19        ledPin.duty(100)
20    elif value == '0xff18e7': #2
21        print('2')
22        ledPin.duty(300)
23    elif value == '0xff7a85': #3
24        print('3')
25        ledPin.duty(1000)
26    else:
27        return
```

The bottom status bar says "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32". It also includes a help message: "Type "help()" for more information." and a triple-greater-than symbol ">>>".

Click “Run current script”. When pressing “0”, “1”, “2”, “3” of the infrared remote control, the buzzer will sound once, and the brightness of the LED light will change correspondingly.

Rendering



The following is the program code:

```

1  from machine import Pin, PWM
2  import time
3  from irrecvdata import irGetCMD
4
5  ledPin=PWM(Pin(14,Pin.OUT),10000,512)
6  buzzerPin=Pin(13,Pin.OUT)
7  recvPin = irGetCMD(15)
8
9  def handleControl(value):
10     buzzerPin.value(1)
11     time.sleep_ms(100)
12     buzzerPin.value(0)
13
14     if value == '0xff6897': #0
15         print('0')
16         ledPin.duty(1)
17     elif value == '0xff30cf': #1
18         print('1')
19         ledPin.duty(100)
20     elif value == '0xff18e7': #2
21         print('2')
22         ledPin.duty(300)
23     elif value == '0xff7a85': #3
24         print('3')
25         ledPin.duty(1000)
26     else:
27         return

```

```

28 try:
29     while True:
30         irValue = recvPin.ir_read()
31         if irValue:
32             print(irValue)
33             handleControl(irValue)
34     except:
35         ledPin.deinit()

```

The handleControl() function is used to execute events corresponding to infrared code values. Every time when the function is called, the buzzer sounds once and determines the brightness of the LED based on the infrared key value. If the key value is not "0", "1", "2", "3", the buzzer sounds once, but the brightness of LED will not change.

```

9 def handleControl(value):
10     buzzerPin.value(1)
11     time.sleep_ms(100)
12     buzzerPin.value(0)
13
14     if value == '0xff6897': #0
15         print('0')
16         ledPin.duty(1)
17     elif value == '0xff30cf': #1
18         print('1')
19         ledPin.duty(100)
20     elif value == '0xff18e7': #2
21         print('2')
22         ledPin.duty(300)
23     elif value == '0xff7a85': #3
24         print('3')
25         ledPin.duty(1000)
26     else:
27         return

```

Each time the key value of IR remote is received, function handleControl() will be called to process it.

```

28 try:
29     while True:
30         irValue = recvPin.ir_read()
31         if irValue:
32             print(irValue)
33             handleControl(irValue)
34     except:
35         ledPin.deinit()

```

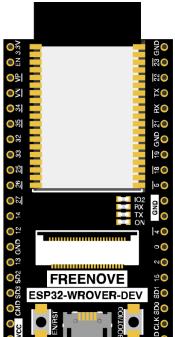
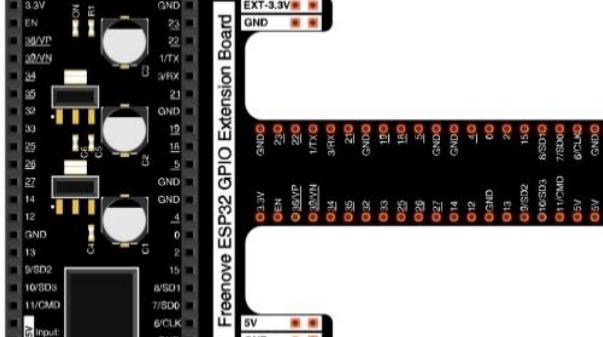
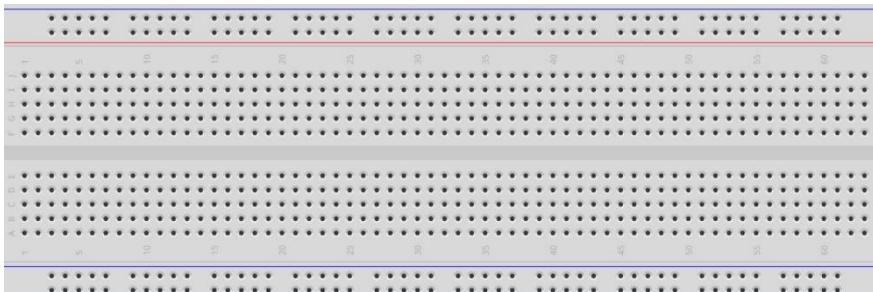
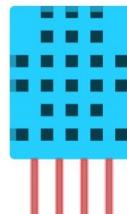
Chapter 24 Hygrothermograph DHT11

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

Project 24.1 Hygrothermograph

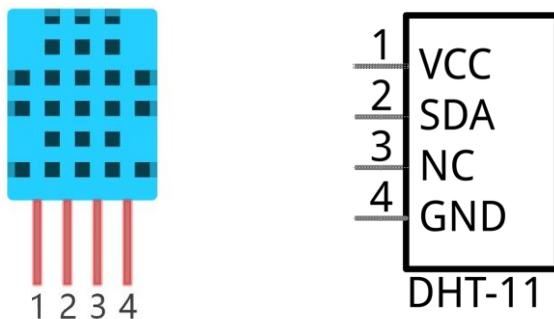
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the ESP32 to read Temperature and Humidity data of the DHT11 Module.

Component List

ESP32-WROVER x1	GPIO Extension Board x1	
		
Breadboard x1		
		
Jumper M/M x4	DHT11 x1	Resistor 10kΩ x1
		

Component knowledge

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



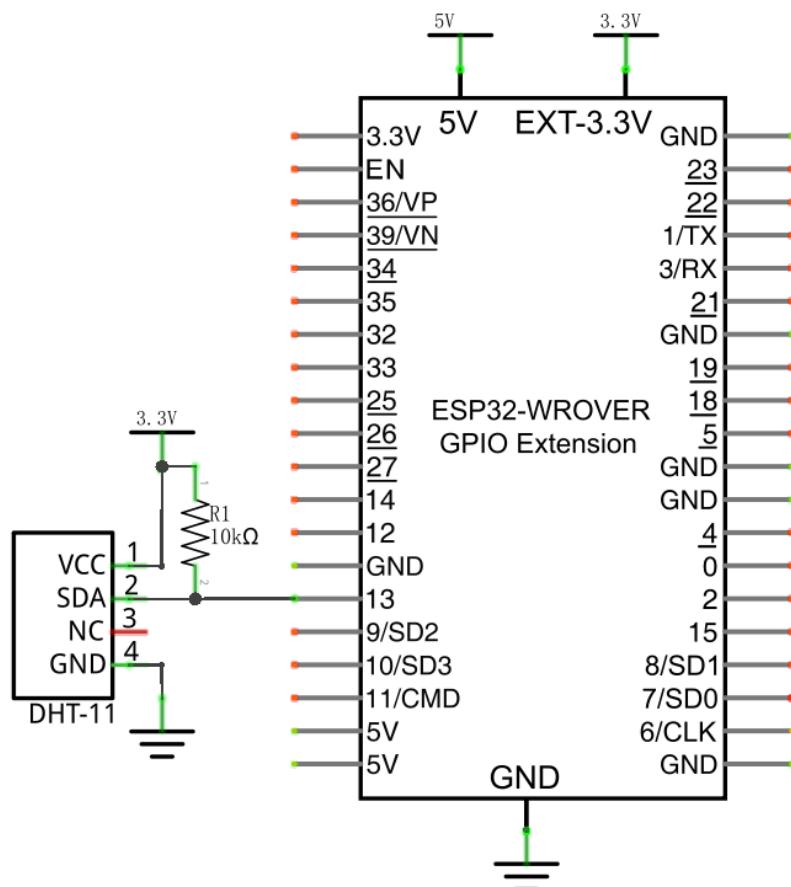
DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

After being powered up, it will initialize in 1S's time. 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.

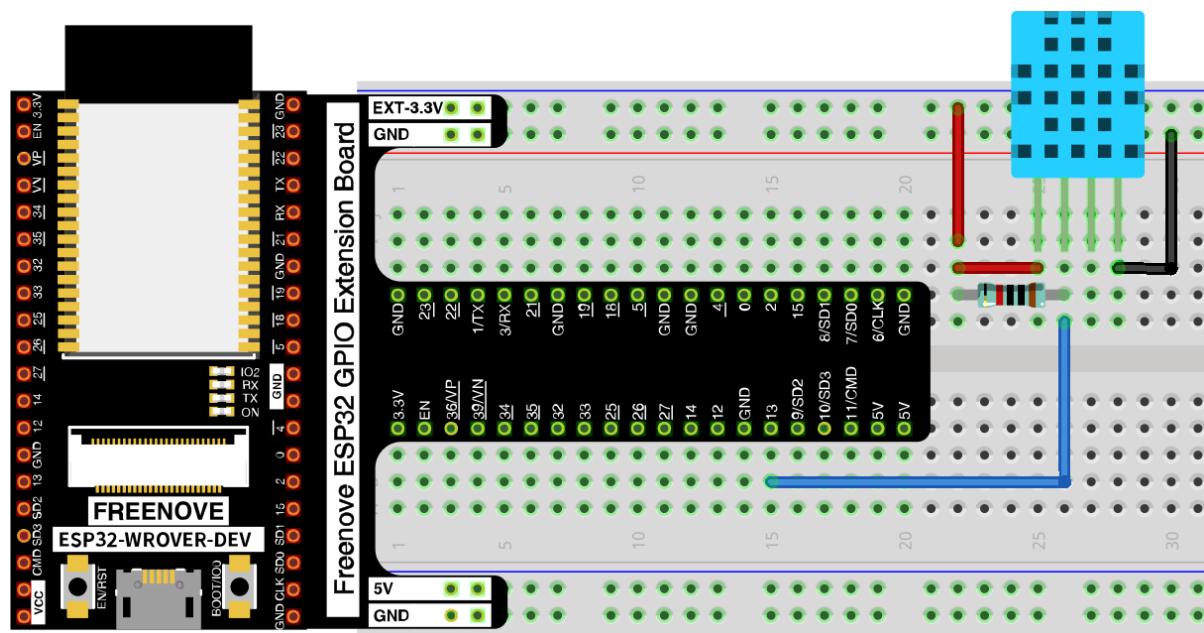
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



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



Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "24.1_Hygrothermograph" and double click "24.1_Hygrothermograph.py".

24.1 Hygrothermograph

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\24.1_Hygrothermograph\24.1_Hygrothermograph.py @ 24 : 1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and a stop button. The left sidebar has two sections: "MicroPython device" containing "boot.py" and "This computer" containing "D:\ Micropython_Codes \ 24.1_Hygrothermograph" and "24.1_Hygrothermograph.py" (which is currently selected). The main area is a code editor titled "24.1_Hygrothermograph.py" with the following code:

```
1 import machine
2 import time
3 import dht
4
5 DHT = dht.DHT11(machine.Pin(13))
6
7 while True:
8     DHT.measure()
9     print('temperature:', DHT.temperature(), 'humidity:', DHT.humidity())
10    time.sleep_ms(1000)
11
12
13
14
15
16
17
18
19
```

Below the code editor is a "Shell" window with the text:

```
>>>
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>
```

Click "Run current script". If your DHT11 is connected incorrectly, the following information will be printed in "Shell".

The screenshot shows the Thonny Shell window with the following output:

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 8, in <module>
    File "dht.py", line 16, in measure
      OSError: [Errno 110] ETIMEDOUT
```

Make sure your circuit is correctly connected and you will see the following messages printed in "Shell".

```
Shell ×
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

temperature: 28 humidity: 60
temperature: 28 humidity: 46
temperature: 28 humidity: 49
temperature: 28 humidity: 50
temperature: 28 humidity: 53
temperature: 28 humidity: 55
temperature: 28 humidity: 57
temperature: 28 humidity: 59
temperature: 28 humidity: 61
temperature: 28 humidity: 63
temperature: 28 humidity: 61
temperature: 28 humidity: 59
temperature: 28 humidity: 57
temperature: 28 humidity: 56
temperature: 28 humidity: 54
temperature: 28 humidity: 54
```

The following is the program code:

```
1 import machine
2 import time
3 import dht
4
5 DHT = dht.DHT11(machine.Pin(13))
6
7 while True:
8     DHT.measure()
9     print(' temperature:', DHT.temperature(), 'humidity:', DHT.humidity())
10    time.sleep_ms(1000)
```

Import machine, time and dht modules.

```
1 import machine
2 import time
3 import dht
```

Associate DHT11 with Pin(13).

```
5 DHT = dht.DHT11(machine.Pin(13))
```

Start DHT11 to measure data once.

```
8 DHT.measure()
```

Call the built-in function of DHT to obtain temperature and humidity data and print them in "Shell".

```
9 print(' temperature:', DHT.temperature(), 'humidity:', DHT.humidity())
```

Obtain temperature and humidity data once per second and print them out.

```
7 while True:
8     DHT.measure()
9     print(' temperature:', DHT.temperature(), 'humidity:', DHT.humidity())
10    time.sleep_ms(1000)
```

Reference

Class dht

Before each use of object **dht**, please add the statement “**import dht**” to the top of python file.

DHT11(): Object of DHT11

DHT12(): Object of DHT12

DHT11.measure(): Start DHT11 to measure temperature and humidity data once.

DHT11.temperature(): Return temperature data obtained by DHT11.

DHT11.humidity(): Return humidity data obtained by DHT11.

DHT12.measure(): Start DHT12 to measure temperature and humidity data once

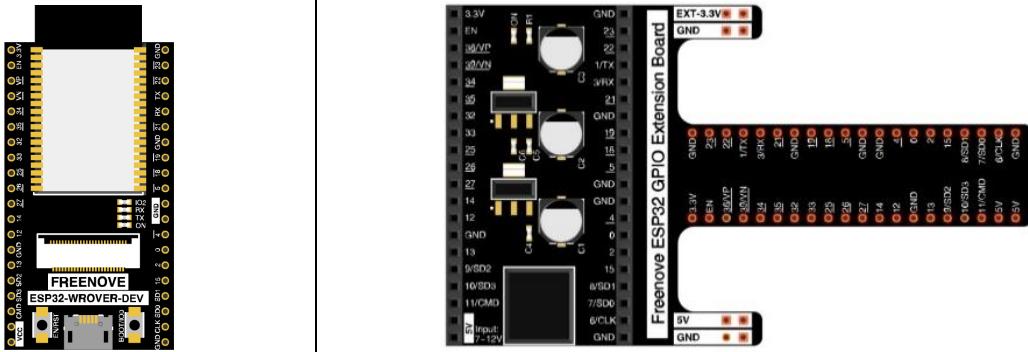
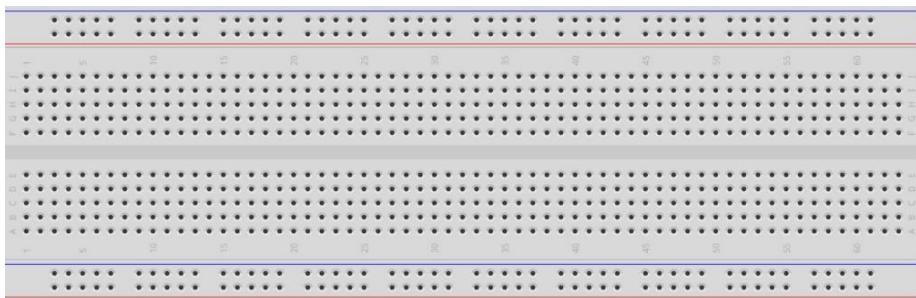
DHT12.temperature(): Return temperature data obtained by DHT12.

DHT12.humidity(): Return humidity data obtained by DHT12.

Project 24.2 Hygrothermograph

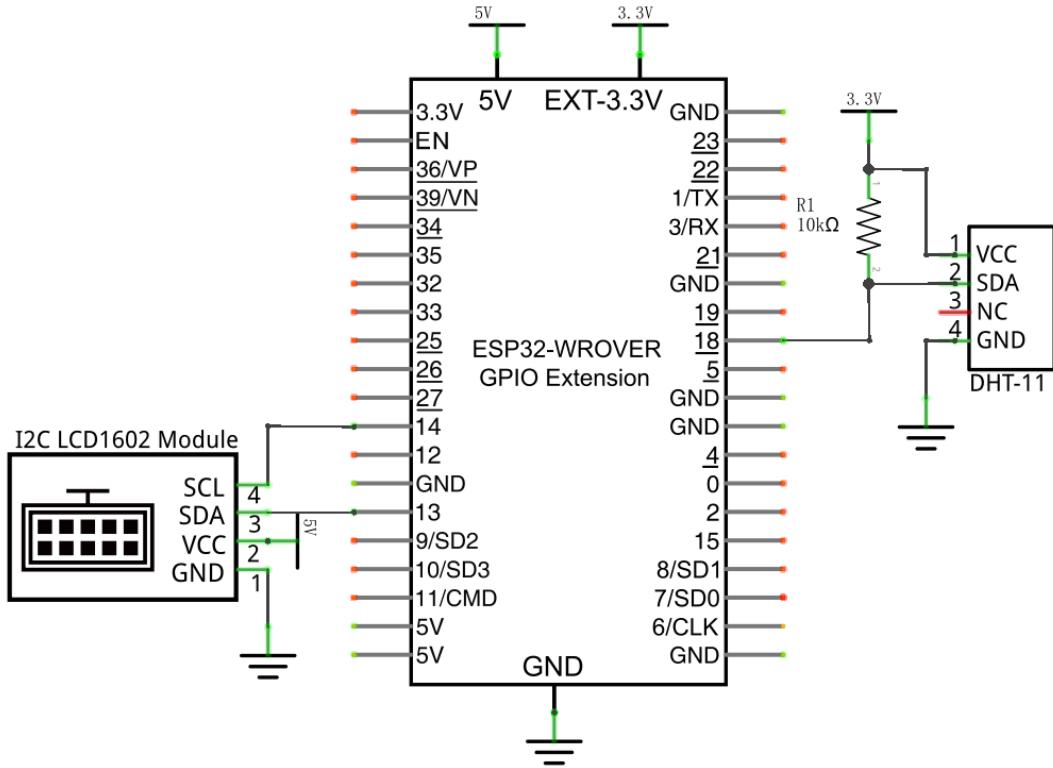
In this project, we use L2C-LCD1602 to display data collected by DHT11.

Component List

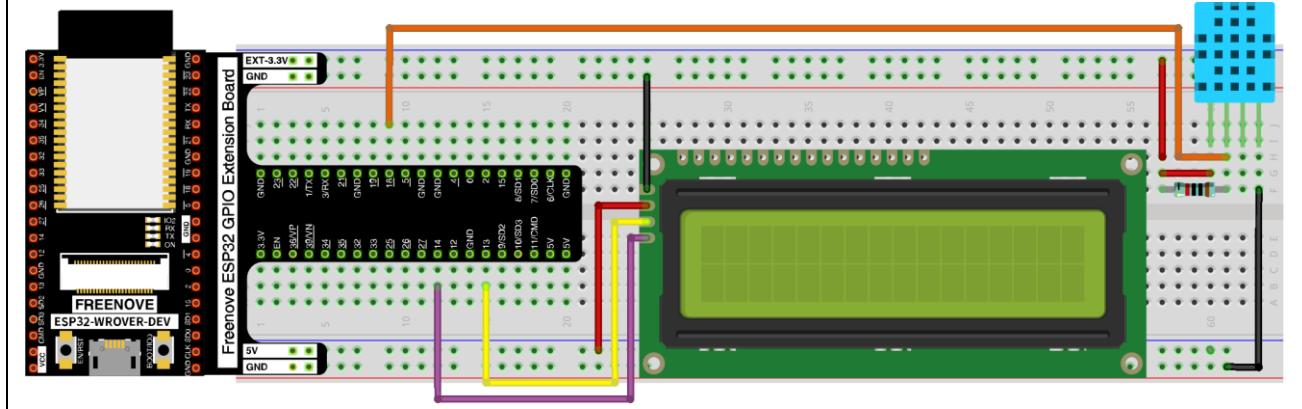
ESP32-WROVER x1	GPIO Extension Board x1
 <p>The image shows two boards: the top one is the Freenove ESP32-WROVER-DEV board, which is a development board for the ESP32 chip. It has a central ESP32 chip, a晶振 (crystal oscillator), and various pins labeled with numbers 1 through 32. The bottom one is the Freenove GPIO Extension Board, which provides additional pins and power options like 3.3V, 5V, and GND.</p>	
Breadboard x1	
 <p>A standard breadboard with two rows of 40 pins each, used for prototyping electronic circuits.</p>	
LCD1602 Module x1	Resistor 10kΩ x1
Jumper F/M x4 Jumper M/M x4	DHT11 x1
 <p>Two sets of jumper wires (F/M and M/M types) and a DHT11 digital temperature and humidity sensor module.</p>	

Circuit

Schematic diagram



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

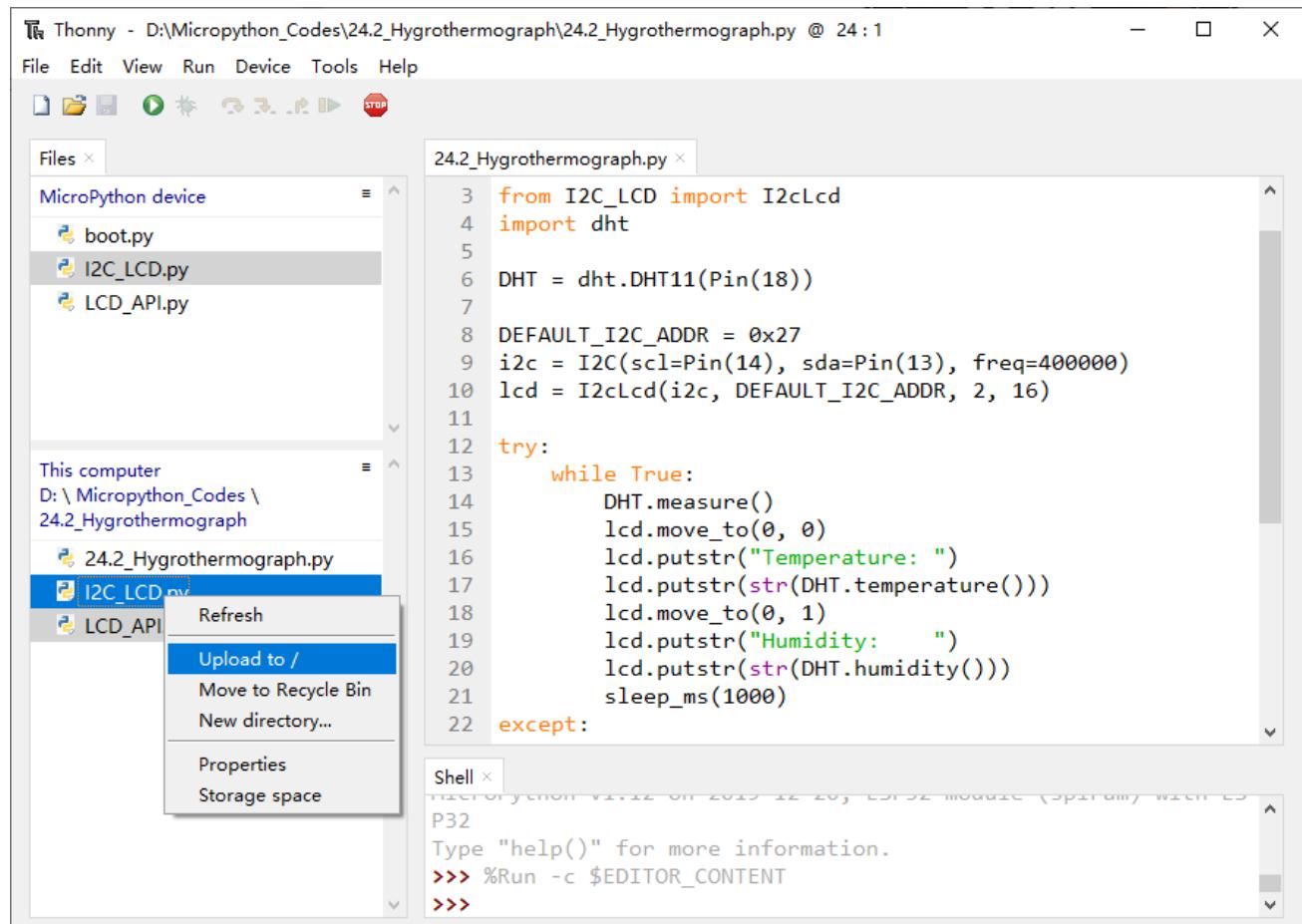


Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → 24.2_Hygrothermograph”.

Select “I2C_LCD.py” and “LCD_API.py”, right click your mouse to select “Upload to /”, wait for “I2C_LCD.py” and “LCD_API.py” to be uploaded to ESP32-WROVER and then double click “24.2_Hygrothermograph.py”.

24.2 Hygrothermograph

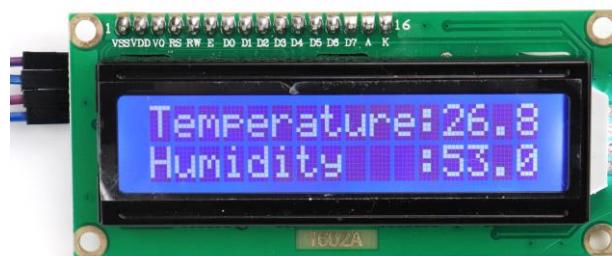


```

from I2C_LCD import I2cLcd
import dht
DHT = dht.DHT11(Pin(18))
DEFAULT_I2C_ADDR = 0x27
i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
try:
    while True:
        DHT.measure()
        lcd.move_to(0, 0)
        lcd.putstr("Temperature: ")
        lcd.putstr(str(DHT.temperature()))
        lcd.move_to(0, 1)
        lcd.putstr("Humidity:   ")
        lcd.putstr(str(DHT.humidity()))
        sleep_ms(1000)
except:

```

Click “Run current script”. The first row of LCD1602 is temperature value and the second row is humidity. Try to “pinch” the DHT11 (without touching the leads) with your index finger and thumb for a brief time, you should see that the displayed value on LCD1602 changes.



The following is the program code:

```

1  from time import sleep_ms
2  from machine import I2C, Pin
3  from I2C_LCD import I2cLcd
4  import dht
5
6  DHT = dht.DHT11(Pin(18))
7  DEFAULT_I2C_ADDR = 0x27
8  i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
9  lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
10
11 try:
12     while True:
13         DHT.measure()
14         lcd.move_to(0, 0)
15         lcd.putstr("Temperature: ")
16         lcd.putstr(str(DHT.temperature()))
17         lcd.move_to(0, 1)
18         lcd.putstr("Humidity: ")
19         lcd.putstr(str(DHT.humidity()))
20         sleep_ms(1000)
21 except:
22     pass

```

Import DHT11 and I2C LCD1602 modules.

```

1  from time import sleep_ms
2  from machine import I2C, Pin
3  from I2C_LCD import I2cLcd
4  import dht

```

Assign Pin(18) to DHT11, Pin(13) and Pin(14) to LCD1602.

```

6  DHT = dht.DHT11(Pin(18))
7  DEFAULT_I2C_ADDR = 0x27
8  i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
9  lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

```

Obtain data of Hygrothermograph every second and display them on LCD1602. The first line displays temperature and the second line displays humidity.

```

12 while True:
13     DHT.measure()
14     lcd.move_to(0, 0)
15     lcd.putstr("Temperature: ")
16     lcd.putstr(str(DHT.temperature()))
17     lcd.move_to(0, 1)
18     lcd.putstr("Humidity: ")
19     lcd.putstr(str(DHT.humidity()))
20     sleep_ms(1000)

```

Chapter 25 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

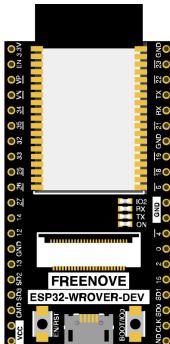
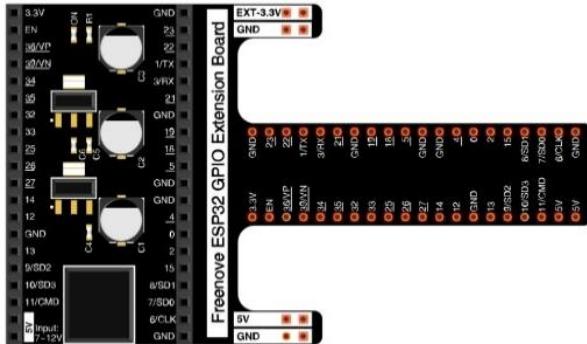
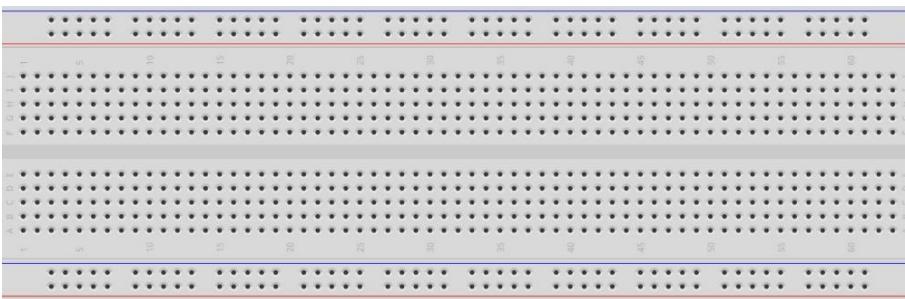
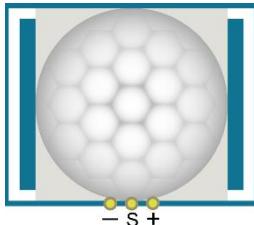
Project 25.1 Infrared Motion Detector with LED Indicator

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.

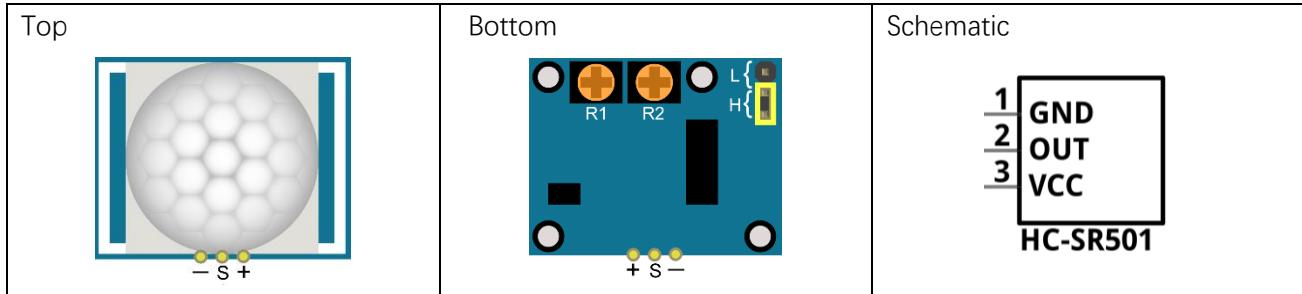
This Infrared Motion Sensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals.

Component List

ESP32-WROVER x1	GPIO Extension Board x1				
					
Breadboard x1					
HC SR501 x1	LED x1	Resistor 220Ω x1	Jumper F/M x3	Jumper M/M x2	
					

Component knowledge

The following is the diagram of the infrared Motion sensor (HC SR-501) :



Description:

Working voltage: 5v-20v(DC) Static current: 65uA.

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.

According to the position of jumper cap, you can choose non-repeatable trigger mode or repeatable mode.

L: non-repeatable trigger mode. The module output high level after sensing 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.

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.

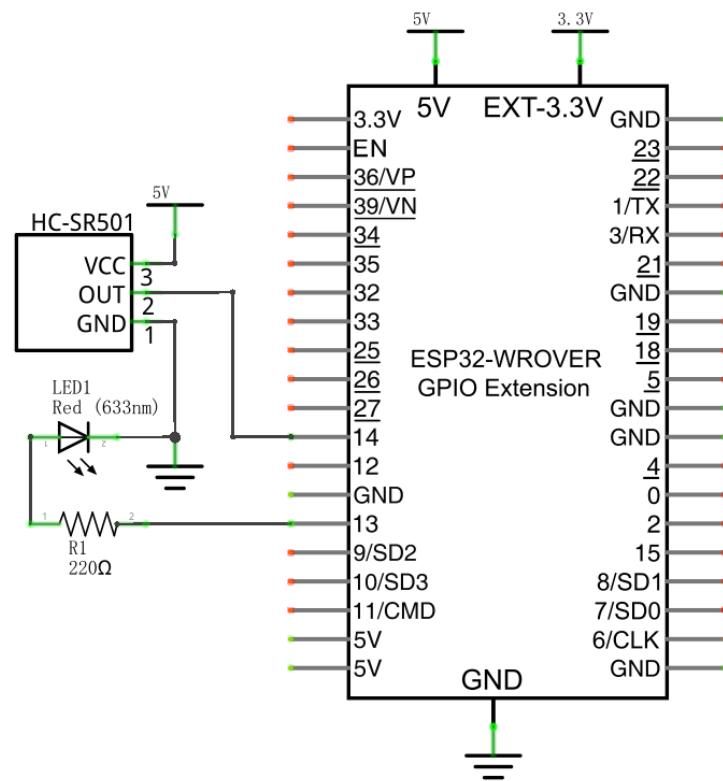
Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.

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 sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency). Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

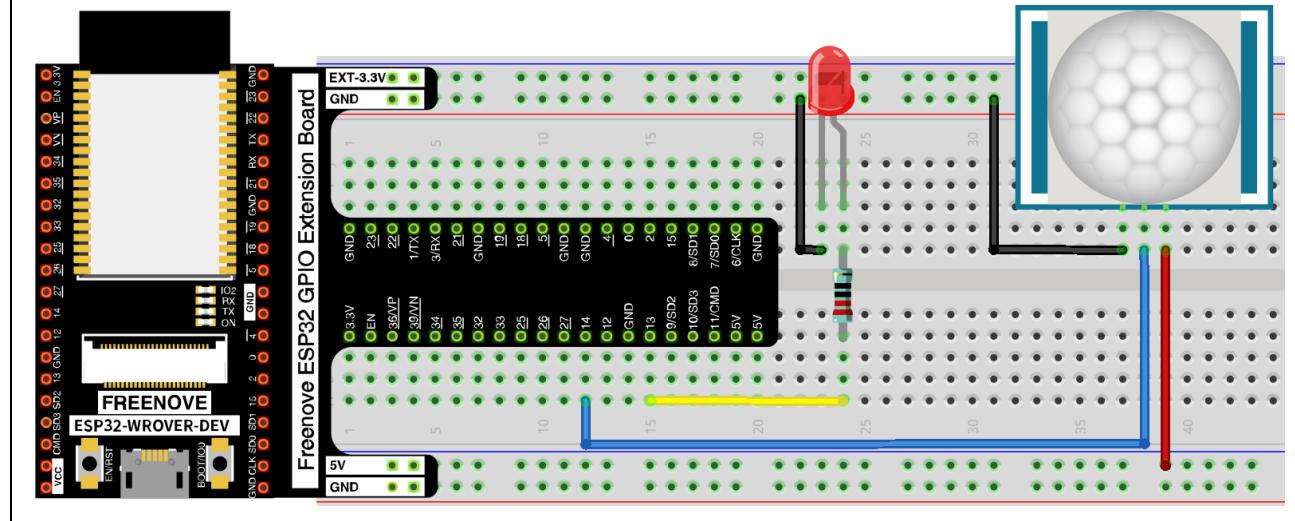
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



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



Code

In this project, we will use an infrared motion sensor to trigger an LED, essentially using the infrared motion sensor as a motion switch. So the code of this project is similar to that of project "[Button & Led](#)". The difference is when infrared motion sensor detects changes, it will output high level; when it detects nothing, it will output low level. When the sensor outputs high level, LED turns ON; Otherwise, LED turns OFF.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "25.1_Infrared_Motion_Detector" and then double click "25.1_Infrared_Motion_Detector.py".

25.1 Infrared_Motion_Detector

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - D:\Micropython_Codes\25.1_Infrared_Motion_Detector\25.1_Infrared_Motion_Detector.py @ 27 : 5". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations. The left sidebar has two tabs: "Files" and "Device". The "Files" tab shows a directory tree with "boot.py" under "MicroPython device" and "25.1_Infrared_Motion_Detector.py" under "This computer". The main editor window displays the Python script:

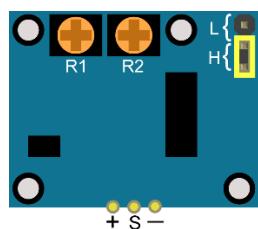
```
1 from machine import Pin
2 import time
3
4 sensorPin=Pin(14,Pin.IN)
5 ledPin=Pin(13,Pin.OUT)
6
7 try:
8     while True:
9         if not sensorPin.value():
10             ledPin.value(1) #Set led turn on
11         else:
12             ledPin.value(0) #Set led turn off
13     except:
14         pass
15
16
17
18
19
20
```

The bottom right pane is a "Shell" window showing the MicroPython prompt:

```
P32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>
```

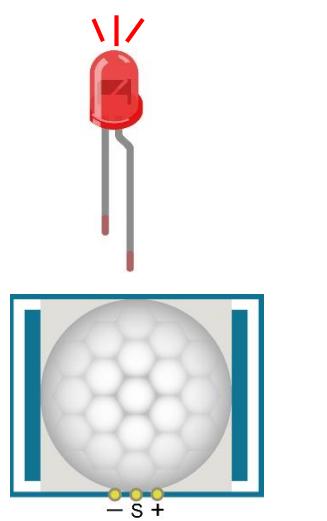
Click "Run current script". Put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the Infrared Motion Sensor and observe whether the LED turns ON or OFF automatically.

You can rotate the potentiometer on the sensor to adjust the detection effect, or use different modes by changing the jumper.

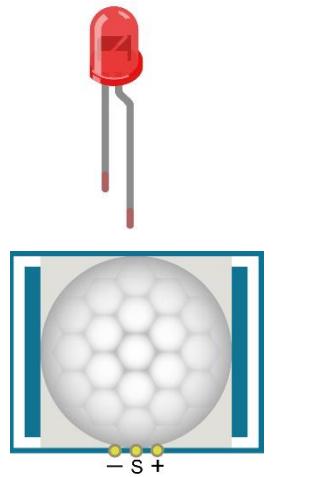


Apart from that, you can also use this sensor to control some other modules to implement different functions by reediting the code, such as the induction lamp, induction door.

Move to the Infrared Motion Sensor



Move away from the Infrared Motion Sensor



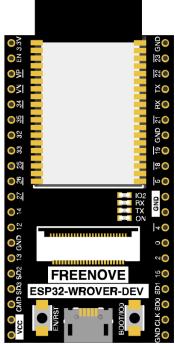
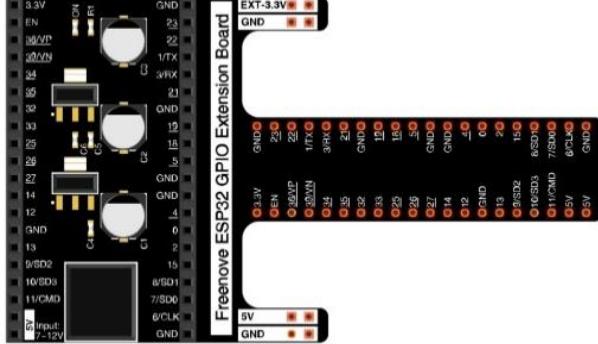
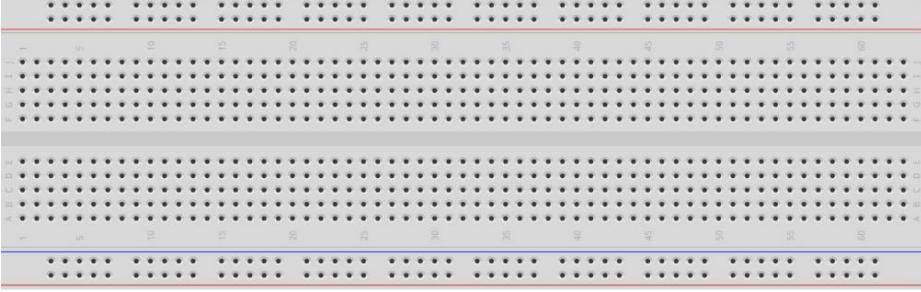
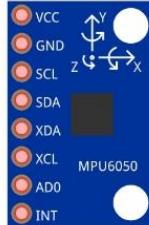
Chapter 26 Attitude Sensor MPU6050

In this chapter, we will learn about a MPU6050 Attitude Sensor which integrates an Accelerometer and Gyroscope.

Project 26.1 Read a MPU6050 Sensor Module

In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor

Component List

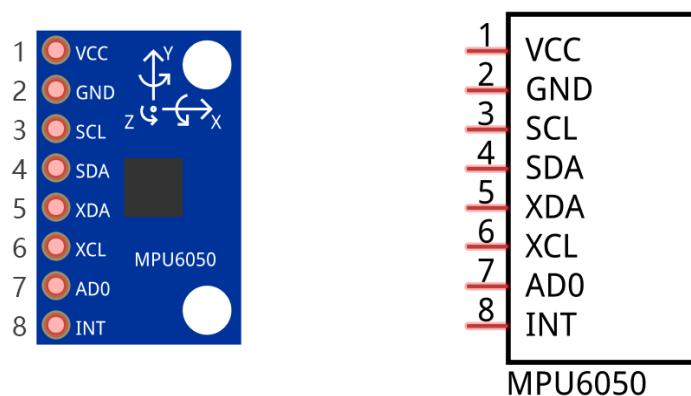
ESP32-WROVER x1	GPIO Extension Board x1
	
Breadboard x1	
Jumper F/M x4	MPU6050 x1
	

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.

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.



The port description of the MPU6050 module is as follows:

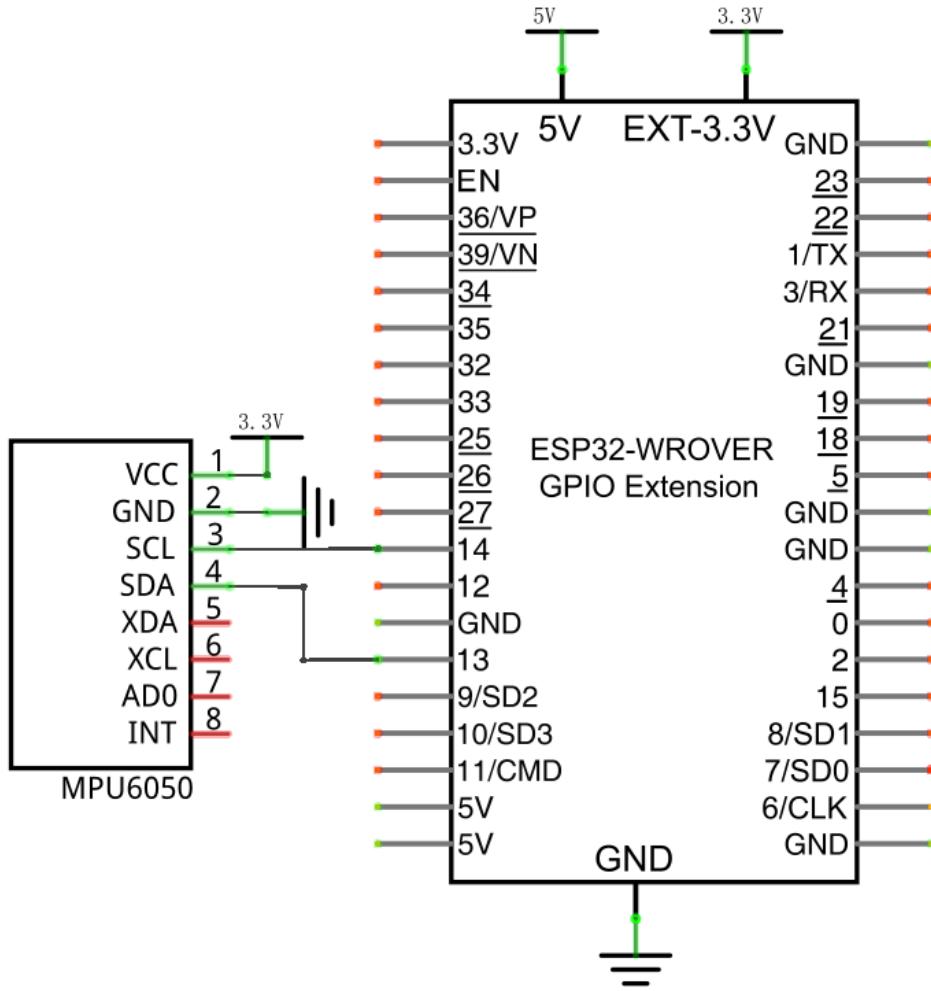
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected 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 datasheet.

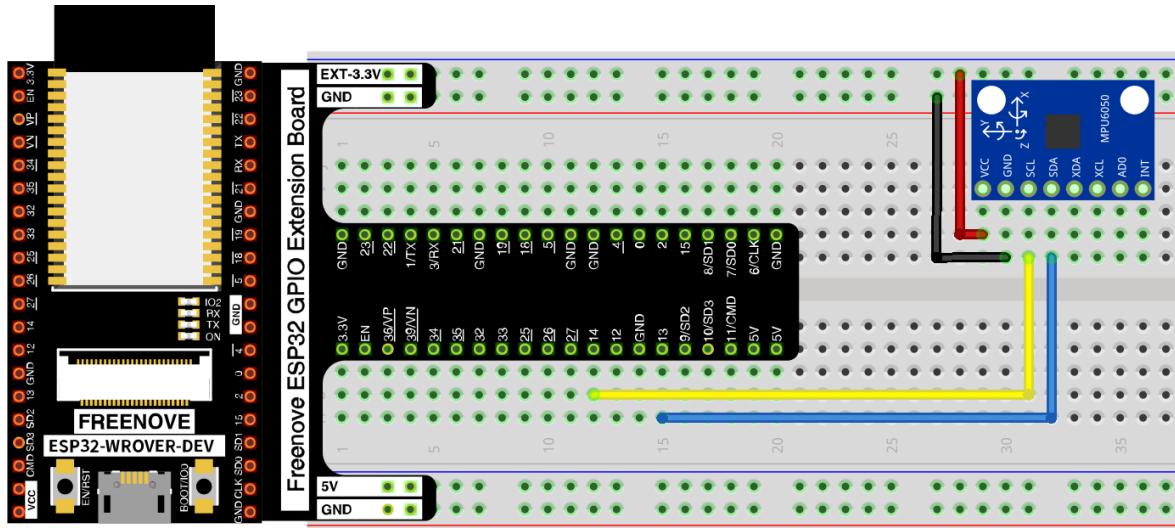
Circuit

Note that the power supply voltage for MPU6050 module is 5V in the circuit.

Schematic diagram



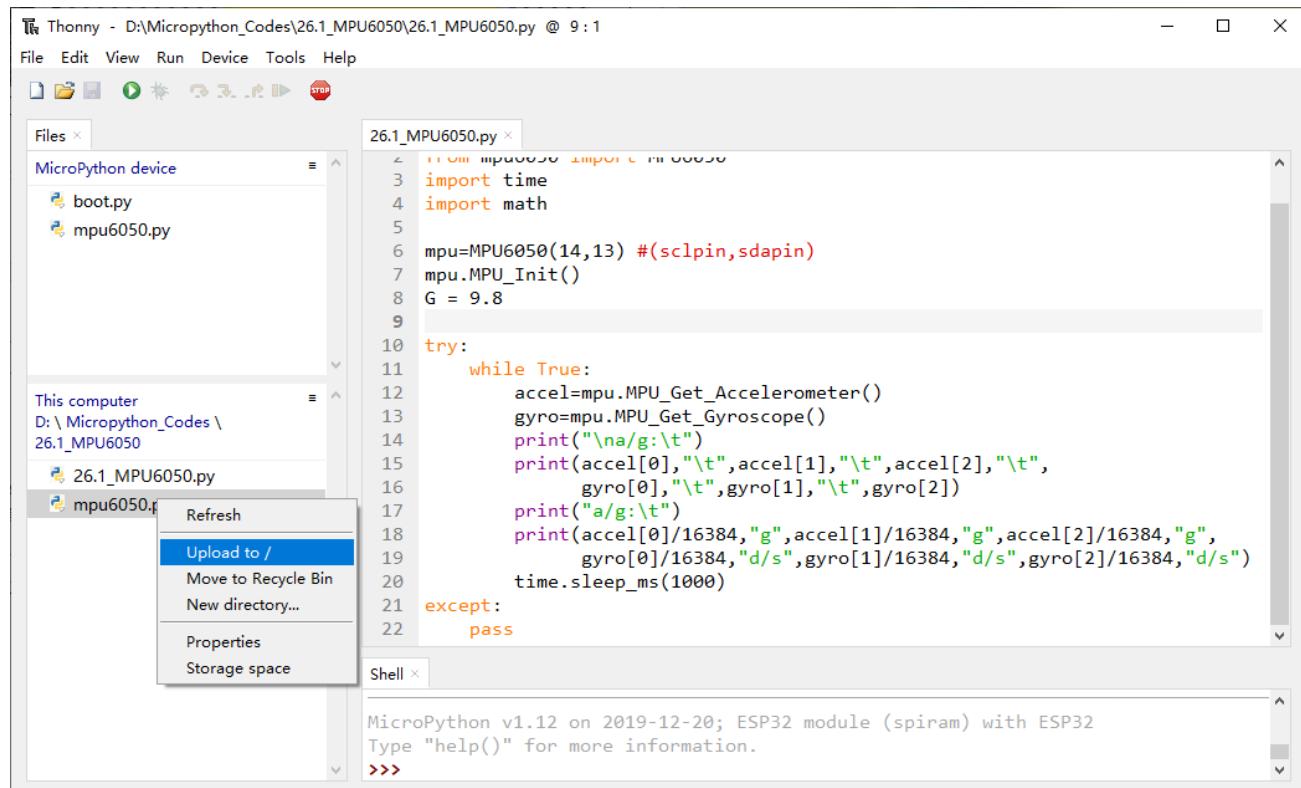
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “26.1_MPU6050”. Select “mpu6050.py”, right click your mouse to select “Upload to /”, wait for “mpu6050.py” to be uploaded to ESP32-WROVER and then double click “26.1_MPU6050.py”.

26.1 MPU6050



Click “Run current script”. When ESP32 obtains acceleration data and gyroscope data of MPU605, it will print them in “Shell”.

```
Shell x
a/g:
59470      52324      56884      65489      30      65525
a/g:
3.629761 g 3.193604 g 3.471924 g 3.997131 d/s 0.001831055 d/s 3.999329 d/s

a/g:
59470      52286      56814      65489      30      65526
a/g:
3.629761 g 3.191284 g 3.467651 g 3.997131 d/s 0.001831055 d/s 3.99939 d/s

a/g:
59474      52332      56904      65492      33      65525
a/g:
3.630005 g 3.194092 g 3.473145 g 3.997314 d/s 0.00201416 d/s 3.999329 d/s

a/g:
59470      52316      56944      65493      29      65530
a/g:
3.629761 g 3.193115 g 3.475586 g 3.997375 d/s 0.00177002 d/s 3.999634 d/s
```

Note: The data transmission of MPU6050 is very sensitive. Therefore, when using it, please make sure the jumper wire is in good contact, otherwise the data may fail to be obtained.

The following is the program code:

```

1  from mpu6050 import MPU6050
2  import time
3
4  mpu=MPU6050(14, 13) #attach the IIC pin(sclpin, sdapin)
5  mpu.MPU_Init()      #initialize the MPU6050
6  G = 9.8
7  time.sleep_ms(1000)#waiting for MPU6050 to work steadily
8  try:
9      while True:
10         accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
11         gyro=mpu.MPU_Get_Gyroscope()      #gain the values of Gyroscope
12         print("\n/a/g:\t")
13         print(accel[0], "\t", accel[1], "\t", accel[2], "\t",
14               gyro[0], "\t", gyro[1], "\t", gyro[2])
15         print("a/g:\t")
16         print(accel[0]/16384, "g", accel[1]/16384, "g", accel[2]/16384, "g",
17               gyro[0]/16384, "d/s", gyro[1]/16384, "d/s", gyro[2]/16384, "d/s")
18         time.sleep_ms(1000)
19     except:
20         pass

```

Import MPU6050 and time modules.

```

1  from mpu6050 import MPU6050
2  import time

```

Set I2C pins and associate them with MPU6050 module, and then initialize MPU6050 and wait for the initialization to complete.

```

4  mpu=MPU6050(14, 13) #attach the IIC pin(sclpin, sdapin)
5  mpu.MPU_Init()      #initialize the MPU6050
6  G = 9.8
7  time.sleep_ms(1000)#waiting for MPU6050 to work steadily

```

Obtain the acceleration data of MPU6050 and store it in accel. Obtain the gyroscope data and store it in gyro.

```

10    accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
11    gyro=mpu.MPU_Get_Gyroscope()      #gain the values of Gyroscope

```

Update and collect the original data of the gyroscope every second and print the original data and processed acceleration and angular velocity data in "Shell".

```

10    accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
11    gyro=mpu.MPU_Get_Gyroscope()      #gain the values of Gyroscope
12    print("\n/a/g:\t")
13    print(accel[0], "\t", accel[1], "\t", accel[2], "\t",
14          gyro[0], "\t", gyro[1], "\t", gyro[2])
15    print("a/g:\t")
16    print(accel[0]/16384, "g", accel[1]/16384, "g", accel[2]/16384, "g",
17          gyro[0]/16384, "d/s", gyro[1]/16384, "d/s", gyro[2]/16384, "d/s")
18    time.sleep_ms(1000)

```

Reference

Class mpu6050

Before each use of **mpu6050**, please add the statement “**from mpu6050 import MPU6050**” to the top of the python file.

MPU6050(sclpin,sdapin): Create an object MPU6050 and associate I2C pin with it.

MPU_Init(): Initialize MPU6050 module.

MPU_Get_Accelerometer(): Obtain original data of MPU6050's acceleration

MPU_Get_Gyroscope(): Obtain original data of MPU6050's Gyroscope

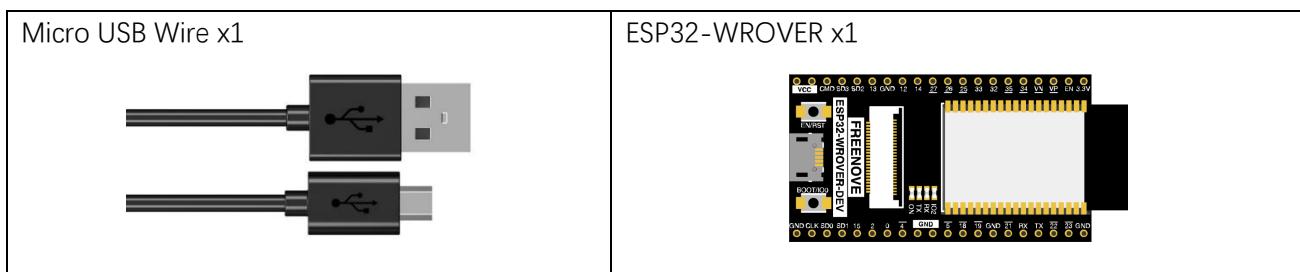
Chapter 27 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-WROVER.

ESP32-WROVER has 3 different WiFi operating modes: Station mode, AP mode and AP+Station mode. All WiFi programming projects must be configured with WiFi operating mode before using WiFi, otherwise WiFi cannot be used.

Project 27.1 Station mode

Component List



Component knowledge

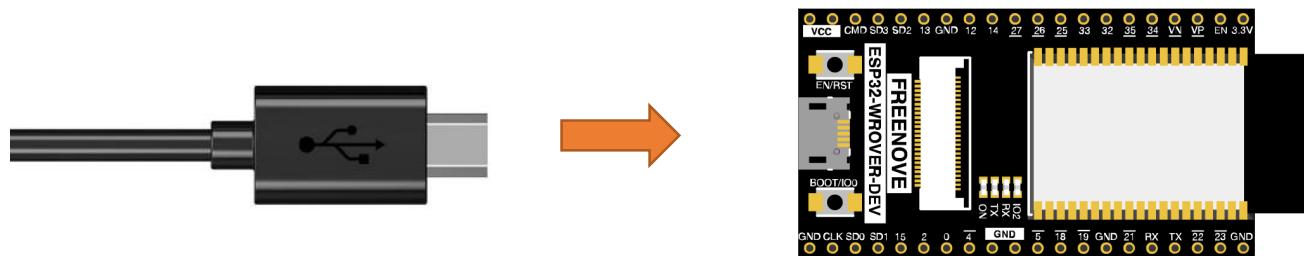
Station mode

When ESP32 selects Station mode, it acts as a WiFi client. It can connect to the router network and communicate with other devices on the router via WiFi connection. As shown below, the PC is connected to the router, and if ESP32 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “27.1_Station_mode” and double click “27.1_Station_mode.py”.

27.1 Station_mode

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Device, Tools, and Help. The left sidebar shows a file tree with 'boot.py' under 'MicroPython device' and '27.1_Station_mode.py' under 'This computer'. The main code editor window displays the following Python script:

```

1 import time
2 import network
3
4 ssidRouter      = '*****' #Enter the router name
5 passwordRouter = '*****' #Enter the router password
6
7 def STA_Setup(ssidRouter,passwordRouter):
8     print("Setup start")
9     sta_if = network.WLAN(network.STA_IF)
10    if not sta_if.isconnected():
11        print('connecting to',ssidRouter)
12        sta_if.active(True)
13        sta_if.connect(ssidRouter,passwordRouter)
14        while not sta_if.isconnected():
15            pass
16        print('Connected, IP address:', sta_if.ifconfig())
17        print("Setup End")
18
19 try:
20     STA_Setup(ssidRouter,passwordRouter)
21 except:
22     sta_if.disconnect()

```

The lines 4 and 5 are highlighted with a red rectangle, and an orange callout bubble points to them with the text: "Enter the correct Router name and password.". The bottom shell window shows the MicroPython version and a prompt: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32 Type "help()" for more information. >>>"

Because the names and passwords of routers in various places are different, before the Code runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to ESP32-WROVER, wait for ESP32 to connect to your router and print the IP address assigned by the router to ESP32 in “Shell”.

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

    Setup start
    connecting to FYI_2.4G
I (4155528) phy: phy_version: 4102, 2fa7a43, Jul 15 2019, 13:06:06,
0, 2
Connected, IP address: ('192.168.1.102', '255.255.255.0', '192.168.
1.1', '192.168.1.1')
Setup End

>>>

```

The following is the program code:

```

1 import time
2 import network
3
4 ssidRouter      = '*****' #Enter the router name
5 passwordRouter = '*****' #Enter the router password
6
7 def STA_Setup(ssidRouter, passwordRouter):
8     print("Setup start")
9     sta_if = network.WLAN(network.STA_IF)
10    if not sta_if.isconnected():
11        print(' connecting to',ssidRouter)
12        sta_if.active(True)
13        sta_if.connect(ssidRouter, passwordRouter)
14        while not sta_if.isconnected():
15            pass
16        print(' Connected, IP address:', sta_if.ifconfig())
17        print("Setup End")
18
19 try:
20     STA_Setup(ssidRouter, passwordRouter)
21 except:
22     sta_if.disconnect()

```

Import network module.

```
2 import network
```

Enter correct router name and password.

```
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
```

Set ESP32 in Station mode.

```
9     sta_if = network.WLAN(network.STA_IF)
```

Activate ESP32's Station mode, initiate a connection request to the router and enter the password to connect.

```
12         sta_if.active(True)
13         sta_if.connect(ssidRouter, passwordRouter)
```

Wait for ESP32 to connect to router until they connect to each other successfully.

```
14     while not sta_if.isconnected():
15         pass
```

Print the IP address assigned to ESP32-WROVER in “Shell”.

```
16     print('Connected, IP address:', sta_if.ifconfig())
```

Reference

Class network

Before each use of **network**, please add the statement “**import network**” to the top of the python file.

WLAN(interface_id): Set to WiFi mode.

network.STA_IF: Client, connecting to other WiFi access points.

network.AP_IF: Access points, allowing other WiFi clients to connect.

active(is_active): With parameters, it is to check whether to activate the network interface; Without parameters, it is to query the current state of the network interface.

scan(ssid, bssid, channel, RSSI, authmode, hidden): Scan for wireless networks available nearby (only scan on STA interface), return a tuple list of information about the WiFi access point.

bssid: The hardware address of the access point, returned in binary form as a byte object. You can use `ubinascii.hexlify()` to convert it to ASCII format.

authmode: Access type

```
AUTH_OPEN = 0
AUTH_WEP = 1
AUTH_WPA_PSK = 2
AUTH_WPA2_PSK = 3
AUTH_WPA_WPA2_PSK = 4
AUTH_MAX = 6
```

Hidden: Whether to scan for hidden access points

False: Only scanning for visible access points

True: Scanning for all access points including the hidden ones.

isconnected(): Check whether ESP32 is connected to AP in Station mode. In STA mode, it returns True if it is connected to a WiFi access point and has a valid IP address; Otherwise it returns False.

connect(ssid, password): Connecting to wireless network.

ssid: WiFi name

password: WiFi password

disconnect(): Disconnect from the currently connected wireless network.

Project 27.2 AP mode

Component List & Circuit

Component List & Circuit are the same as in Section 27.1.

Component knowledge

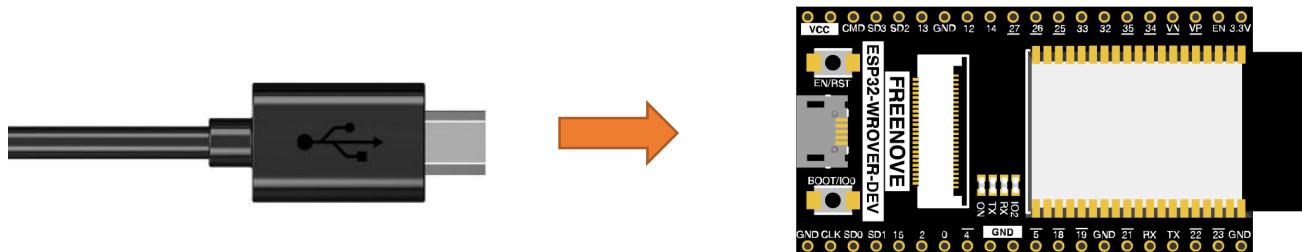
AP mode

When ESP32 selects AP mode, it creates a hotspot network that is separated from the Internet and waits for other WiFi devices to connect. As shown in the figure below, ESP32 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP32, it must be connected to the hotspot of ESP32. Only after a connection is established with ESP32 can they communicate.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “27.2_AP_mode”. and double click “27.2_AP_mode.py”.

27.2 AP_mode

```

  Thonny - D:\Micropython_Codes\27.2_AP_mode\27.2_AP_mode.py @ 33 : 5
  File Edit View Run Device Tools Help
  Files ×
  MicroPython device ×
    boot.py
  This computer ×
    D:\ Micropython_Codes \
      27.2_AP_mode
        27.2_AP_mode.py
  27.2_AP_mode.py ×
  1 import network
  2
  3     ssidAP      = 'WiFi_Name' #Enter the router name
  4     passwordAP  = '12345678' #Enter the router password
  5
  6 local_IP      = '192.168.1.10'
  7 gateway       = '192.168.1.1'
  8 subnet        = '255.255.255.0'
  9 dns           = '8.8.8.8'
  10
  11 ap_if = network.WLAN(network.AP_IF)
  12
  13 def AP_Setup(ssidAP,passwordAP):
  14     ap_if.ifconfig([local_IP,gateway,subnet,dns])
  15     print("Setting soft-AP ... ")
  16     ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
  17     ap_if.active(True)
  18     print('Success, IP address:', ap_if.ifconfig())
  19     print("Setup End\n")
  20
  21 try:
  22     AP_Setup(ssidAP,passwordAP)
  
```

Shell ×

```

  MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
  Type "help()" for more information.
  >>>
  
```

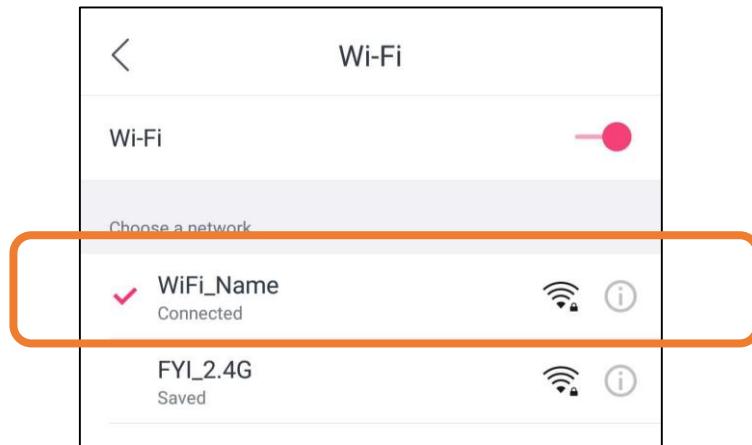
Before the Code runs, you can make any changes to the AP name and password for ESP32 in the box as shown in the illustration above. Of course, you can leave it alone by default.

Click “Run current script”, open the AP function of ESP32 and print the access point information.

```

  Shell ×
  MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
  Type "help()" for more information.
  >>> %Run -c $EDITOR_CONTENT
  Setting soft-AP ...
  Success, IP address: ('192.168.1.10', '192.168.1.1', '255.255.255.0', '8.8.8.8')
  Setup End
  >>>
  
```

Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32, which is called "WiFi_Name" in this Code. You can enter the password "12345678" to connect it or change its AP name and password by modifying Code.



The following is the program code:

```

1 import network
2
3 ssidAP      = 'WiFi_Name' #Enter the router name
4 passwordAP  = '12345678' #Enter the router password
5
6 local_IP    = '192.168.1.10'
7 gateway     = '192.168.1.1'
8 subnet      = '255.255.255.0'
9 dns         = '8.8.8.8'
10
11 ap_if = network.WLAN(network.AP_IF)
12
13 def AP_Setup(ssidAP, passwordAP):
14     ap_if.ifconfig([local_IP, gateway, subnet, dns])
15     print("Setting soft-AP ... ")
16     ap_if.config(essid=ssidAP, authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
17     ap_if.active(True)
18     print(' Success, IP address:', ap_if.ifconfig())
19     print("Setup End\n")
20
21 try:
22     AP_Setup(ssidAP, passwordAP)
23 except:
24     ap_if.disconnect()

```

Import network module.

1	import network
---	----------------

Enter correct AP name and password.

```
3 ssidAP      = 'WiFi_Name' #Enter the router name
4 passwordAP = '12345678' #Enter the router password
```

Set ESP32 in AP mode.

```
11 ap_if = network.WLAN(network.AP_IF)
```

Configure IP address, gateway and subnet mask for ESP32.

```
14 ap_if.ifconfig([local_IP, gateway, subnet, dns])
```

Turn on an AP in ESP32, whose name is set by ssid_AP and password is set by password_AP.

```
16 ap_if.config(essid=ssidAP, authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
17 ap_if.active(True)
```

If the program is running abnormally, the AP disconnection function will be called.

```
14 ap_if.disconnect()
```

Reference

Class network

Before each use of **network**, please add the statement “**import network**” to the top of the python file.

WLAN(interface_id): Set to WiFi mode.

network.STA_IF: Client, connecting to other WiFi access points

network.AP_IF: Access points, allowing other WiFi clients to connect

active(is_active): With parameters, it is to check whether to activate the network interface; Without parameters, it is to query the current state of the network interface

isconnected(): In AP mode, it returns True if it is connected to the station; otherwise it returns False.

connect(ssid, password): Connecting to wireless network

ssid: WiFi name

password: WiFi password

config(essid, channel): To obtain the MAC address of the access point or to set the WiFi channel and the name of the WiFi access point.

ssid: WiFi account name

channel: WiFi channel

ifconfig([(ip, subnet, gateway, dns)]): Without parameters, it returns a 4-tuple (ip, subnet_mask, gateway, DNS_server); With parameters, it configures static IP.

ip: IP address

subnet_mask: subnet mask

gateway: gateway

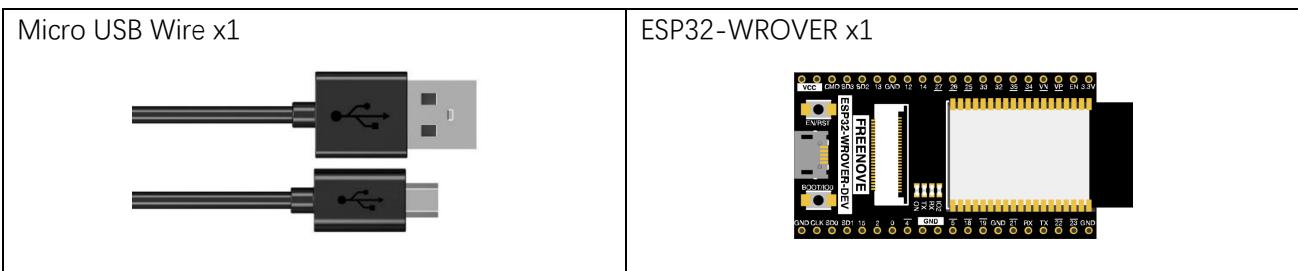
DNS_server: DNS server

disconnect(): Disconnect from the currently connected wireless network

status(): Return the current status of the wireless connection

Project 27.3 AP+Station mode

Component List



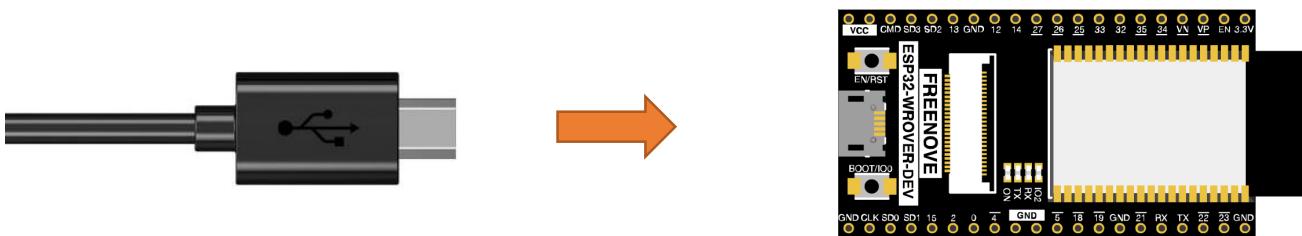
Component knowledge

AP+Station mode

In addition to AP mode and Station mode, ESP32 can also use AP mode and Station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32's Station mode, connect it to the router network, and it can communicate with the Internet via the router. At the same time, turn on its AP mode to create a hotspot network. Other WiFi devices can choose to connect to the router network or the hotspot network to communicate with ESP32.

Circuit

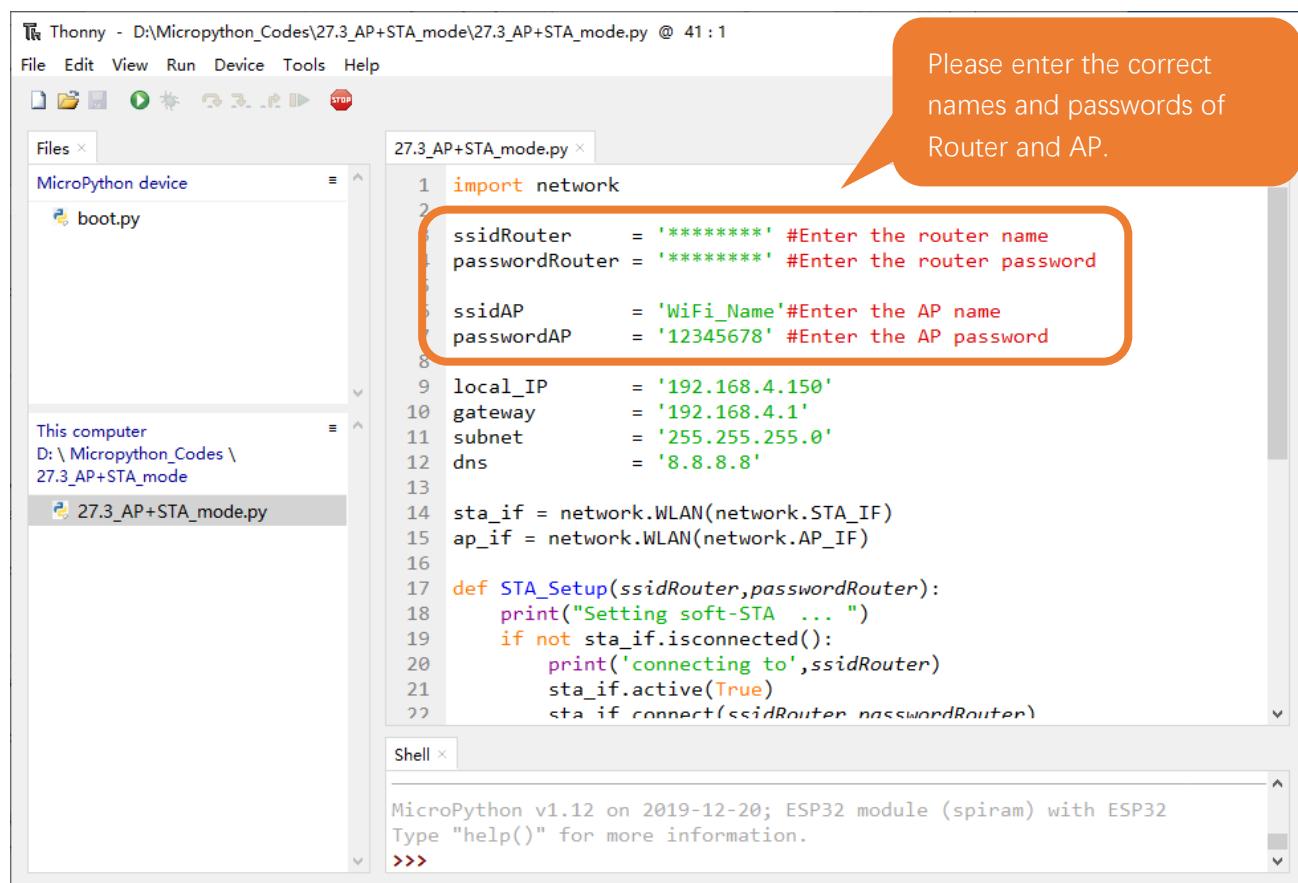
Connect Freenove ESP32 to the computer using the USB cable.



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “27.3_AP+STA_mode” and double click “27.3_AP+STA_mode.py”.

27.3 AP+STA_mode



The screenshot shows the Thonny IDE interface. In the top menu bar, it says "Thonny - D:\Micropython_Codes\27.3_AP+STA_mode\27.3_AP+STA_mode.py @ 41 : 1". Below the menu is a toolbar with icons for file operations. The left sidebar shows a "Files" tree with "MicroPython device" expanded, showing "boot.py" and "27.3_AP+STA_mode.py" selected. The main area contains the code for "27.3_AP+STA_mode.py". A callout bubble from the right side of the screen points to the configuration lines (lines 6-22) and contains the text: "Please enter the correct names and passwords of Router and AP." The code is as follows:

```

1 import network
2
3 ssidRouter = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP = 'WiFi_Name'#Enter the AP name
7 passwordAP = '12345678' #Enter the AP password
8
9 local_IP = '192.168.4.150'
10 gateway = '192.168.4.1'
11 subnet = '255.255.255.0'
12 dns = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16
17 def STA_Setup(ssidRouter,passwordRouter):
18     print("Setting soft-STA ... ")
19     if not sta_if.isconnected():
20         print('connecting to',ssidRouter)
21         sta_if.active(True)
22         sta_if.connect(ssidRouter, passwordRouter)

```

In the bottom right corner, there is a "Shell" window showing the MicroPython environment:

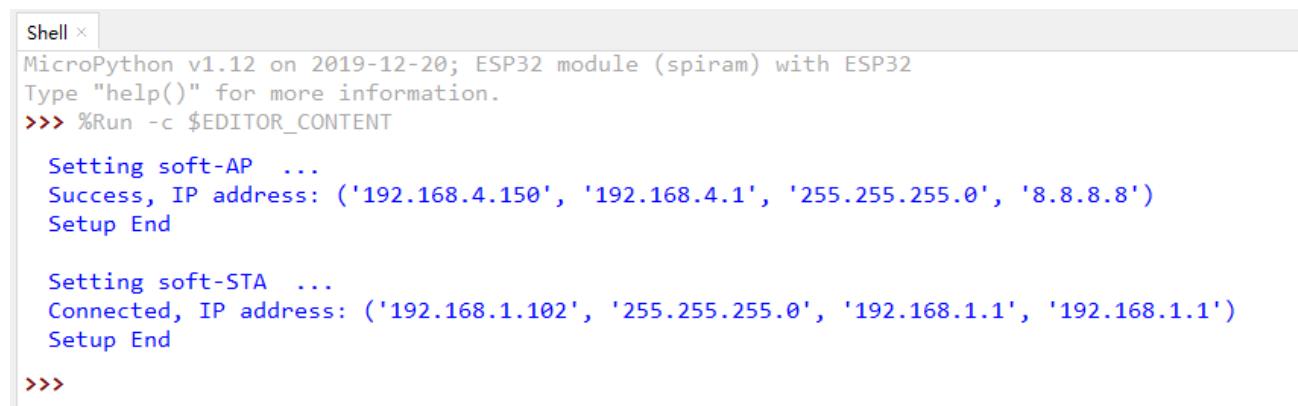
```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>>

```

It is analogous to project 27.1 and project 27.2. Before running the Code, you need to modify ssidRouter, passwordRouter, ssidAP and passwordAP shown in the box of the illustration above.

After making sure that the code is modified correctly, click “Run current script” and the “Shell” will display as follows:



The screenshot shows the Thonny IDE Shell window. It starts with the MicroPython environment information: "MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32" and "Type "help()" for more information.". Then it shows the command ">>> %Run -c \$EDITOR_CONTENT" followed by the output of the code execution:

```

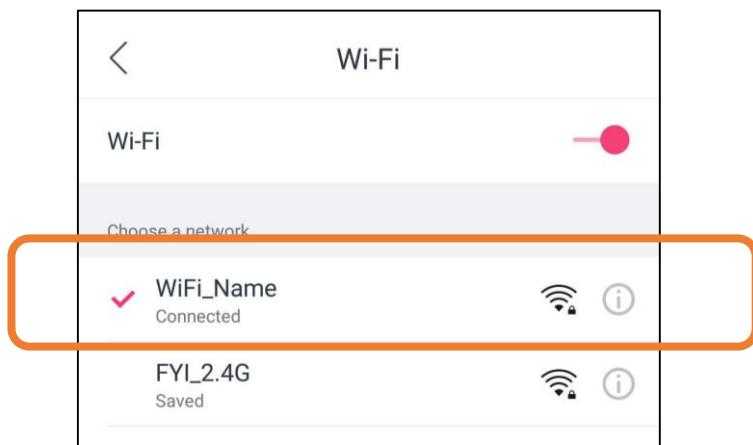
Setting soft-AP ...
Success, IP address: ('192.168.4.150', '192.168.4.1', '255.255.255.0', '8.8.8.8')
Setup End

Setting soft-STA ...
Connected, IP address: ('192.168.1.102', '255.255.255.0', '192.168.1.1', '192.168.1.1')
Setup End

>>>

```

Turn on the WiFi scanning function of your phone, and you can see the ssidAP on ESP32.



The following is the program code:

```
1 import network
2
3 ssidRouter      = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP          = 'WiFi_Name' #Enter the AP name
7 passwordAP      = '12345678' #Enter the AP password
8
9 local_IP        = '192.168.4.150'
10 gateway        = '192.168.4.1'
11 subnet         = '255.255.255.0'
12 dns            = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16
17 def STA_Setup(ssidRouter, passwordRouter):
18     print("Setting soft-STA ... ")
19     if not sta_if.isconnected():
20         print(' connecting to',ssidRouter)
21         sta_if.active(True)
22         sta_if.connect(ssidRouter, passwordRouter)
23         while not sta_if.isconnected():
24             pass
25     print('Connected, IP address:', sta_if.ifconfig())
26     print("Setup End")
27
28 def AP_Setup(ssidAP, passwordAP):
29     ap_if.ifconfig([local_IP, gateway, subnet, dns])
30     print("Setting soft-AP ... ")
```

```
31     ap_if.config(essid=ssidAP, authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
32     ap_if.active(True)
33     print(' Success, IP address:', ap_if.ifconfig())
34     print("Setup End\n")
35
36 try:
37     AP_Setup(ssidAP, passwordAP)
38     STA_Setup(ssidRouter, passwordRouter)
39 except:
40     sta_if.disconnect()
41     ap_if.disconnect()
```



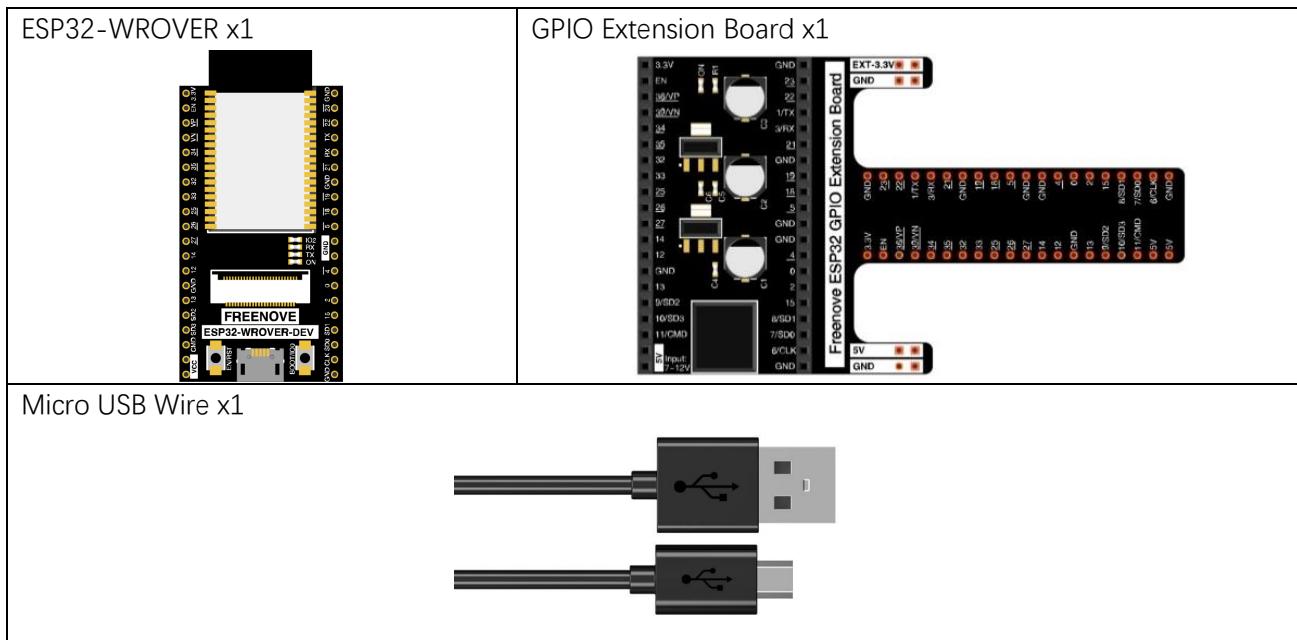
Chapter 28 TCP/IP

In this chapter, we will introduce how ESP32 implements network communications based on TCP/IP protocol. There are two roles in TCP/IP communication, namely Server and Client, which will be implemented respectively with two projects in this chapter.

Project 28.1 As Client

In this section, ESP32 is used as Client to connect Server on the same LAN and communicate with it.

Component List



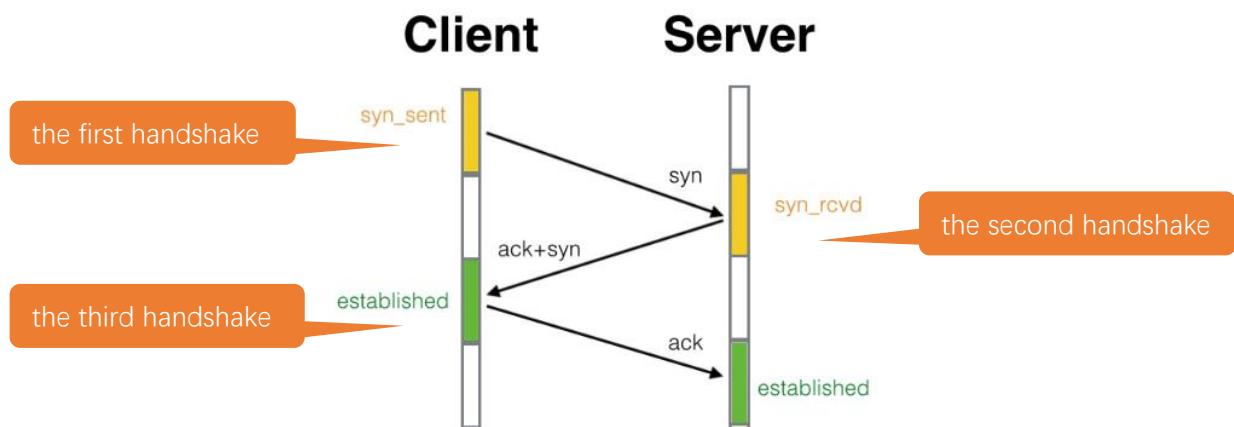
Component knowledge

TCP connection

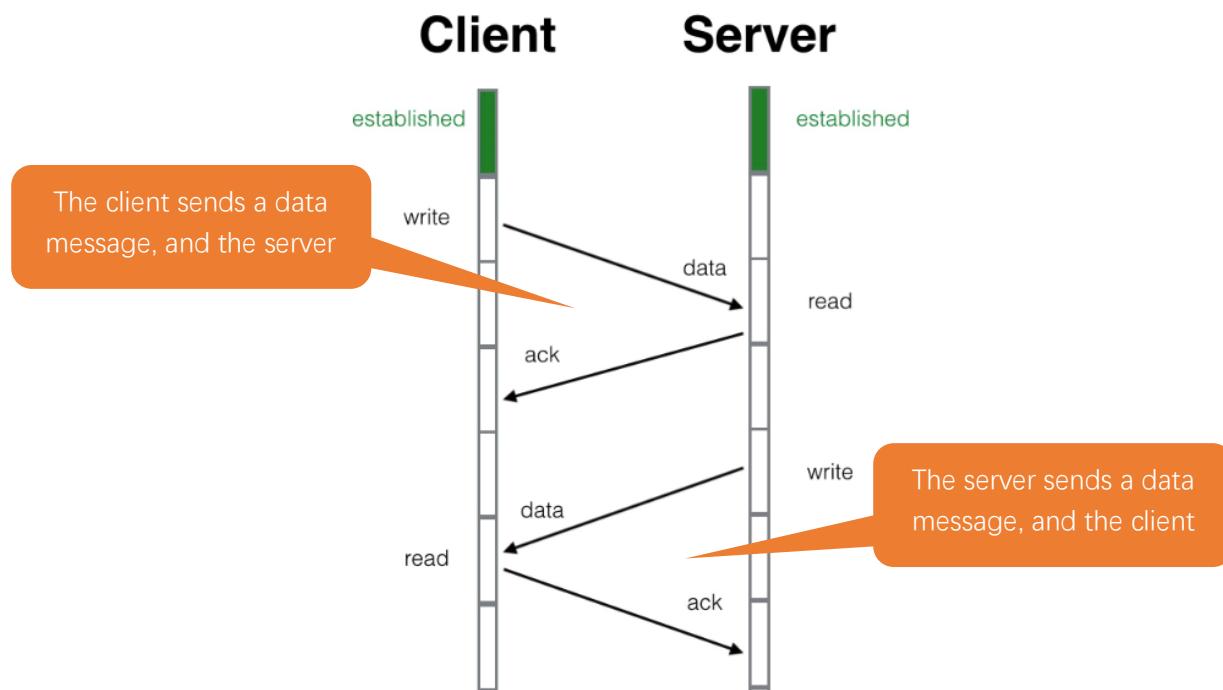
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request to the server, and each time such a request is proposed, a "three-times handshake" is required.

Three-times handshake: In the TCP protocol, during the preparation phase of sending data, the client and the server interact three times to ensure the reliability of the connection, which is called "three-times handshake". The first handshake, the client sends a connection request to the server and waits for the server to confirm. The second handshake, the server sends a response back to the client informing that it has received the connection request.

The third handshake, the client sends a confirmation message to the server again to confirm the connection.



TCP is a connection-oriented, low-level transmission control protocol. After TCP establishes a connection, the client and server can send and receive messages to each other, and the connection will always exist as long as the client or server does not initiate disconnection. Each time one party sends a message, the other party will reply with an ack signal.





Install Processing

In this tutorial, we use Processing to build a simple TCP/IP communication platform.

If you've not installed Processing, you can download it by clicking <https://processing.org/download/>. You can choose an appropriate version to download according to your PC system.

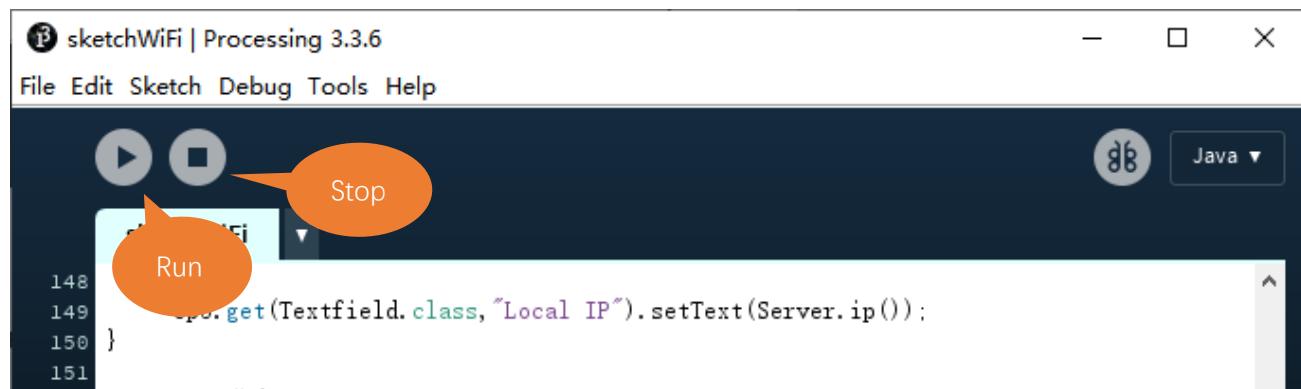
The screenshot shows the official Processing website's download section. At the top, there are links for "Processing", "p5.js", "Processing.py", "Processing for Android", "Processing for Pi", and "Processing Foundation". Below this is a search bar. The main content area features a large "Processing" logo with a geometric background. To the left is a sidebar with links: "Cover", "Download", "Donate", "Exhibition", "Reference", "Libraries", "Tools", "Environment", "Tutorials", "Examples", "Books", "Overview", and "People". In the center, it says "Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below." It shows the "3.5.4 (17 January 2020)" release. For Windows, there are links for "Windows 64-bit" and "Windows 32-bit". For Linux, there is a link for "Linux 64-bit". For Mac OS X, there is a link for "Mac OS X". Below the download links, there are links for "» Github", "» Report Bugs", "» Wiki", "» Supported Platforms", and a note about "Read about the changes in 3.0. The list of revisions covers the differences between releases in detail."

Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

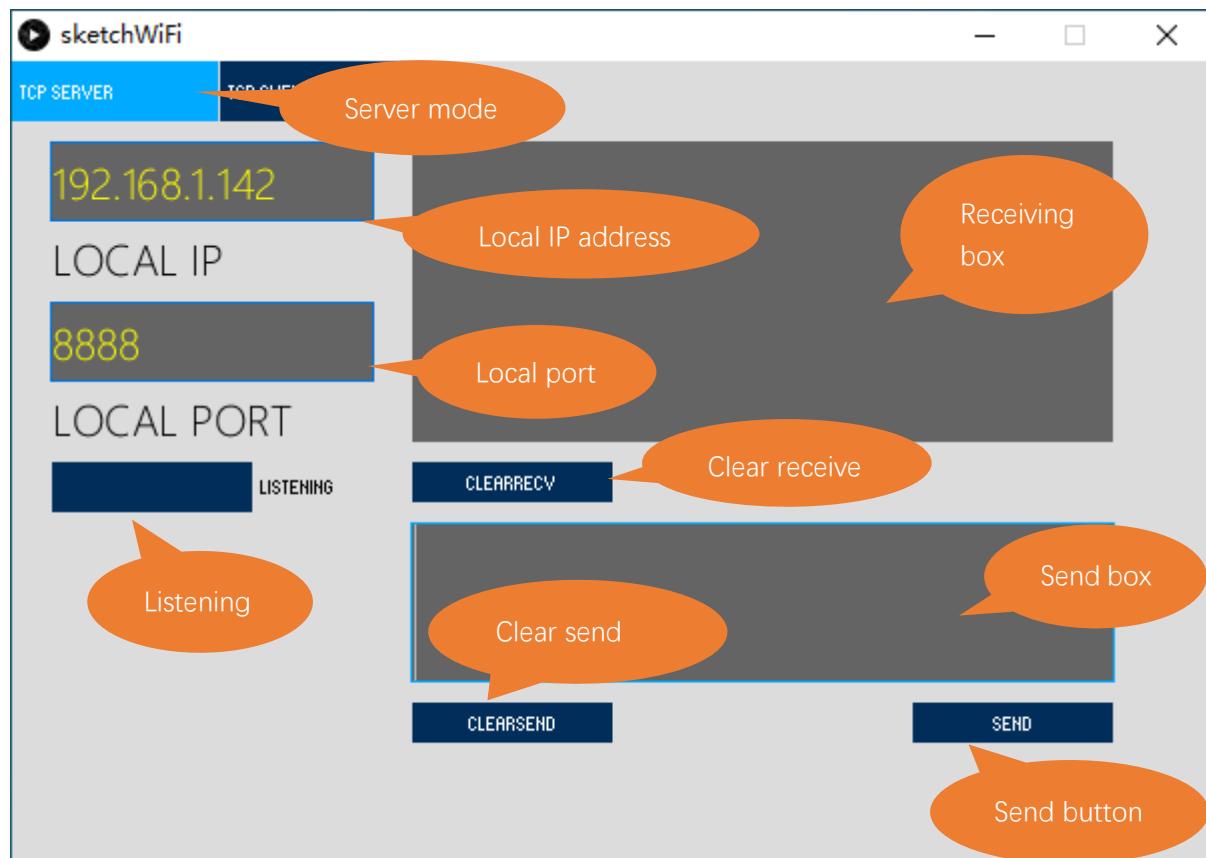
core	2020/1/17 12:16
java	2020/1/17 12:17
lib	2020/1/17 12:16
modes	2020/1/17 12:16
tools	2020/1/17 12:16
processing.exe	2020/1/17 12:16
processing-java.exe	2020/1/17 12:16
revisions.txt	2020/1/17 12:16

Use Server mode for communication

Open the “Freenove Ultimate Starter Kit for ESP32\ MicroPython_Codes\28.1_TCP_as_Client\sketchWiFi\sketchWiFi.pde”. Click “Run”.

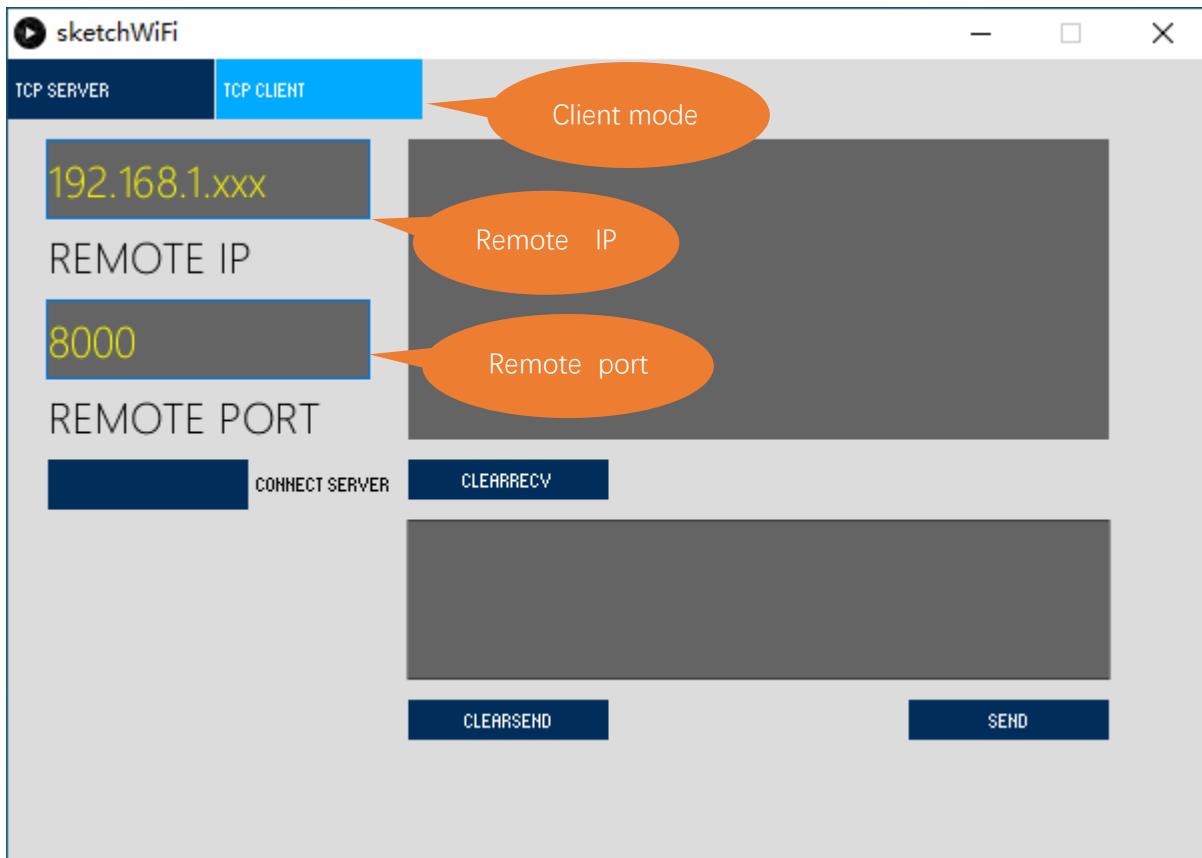


The new pop-up interface is as follows. If ESP32 is used as Client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, ESP32 Code needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP32 serves as Server, select TCP CLIENT mode for sketchWiFi.



When sketchWiFi selects TCP CLIENT mode, the LOCAL IP and LOCAL PORT of sketchWiFi need to be changed according to the IP address and port number printed by the serial monitor.

Mode selection: select **Server mode/Client mode**.

IP address: In Server mode, this option does not need to be filled in, and the computer will automatically obtain the IP address.

In Client mode, fill in the remote IP address to be connected.

Port number: In Server mode, fill in a port number for client devices to make an access connection.

In client mode, fill in port number given by the Server devices to make an access connection.

Start button: In server mode, push the button, and then the computer will serve as Server and open a port number for Client to make access connection. During this period, the computer will keep monitoring.

In client mode, before pushing the button, please make sure the server is on, remote IP address and remote port number is correct; push the button, and the computer will make access connection to the remote port number of the remote IP as a Client.

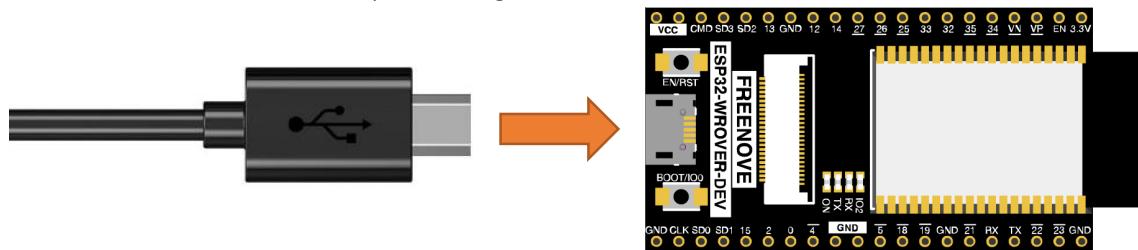
clear receive: clear out the content in the receiving text box

clear send: clear out the content in the sending text box

Sending button: push the sending button, the computer will send the content in the text box to others.

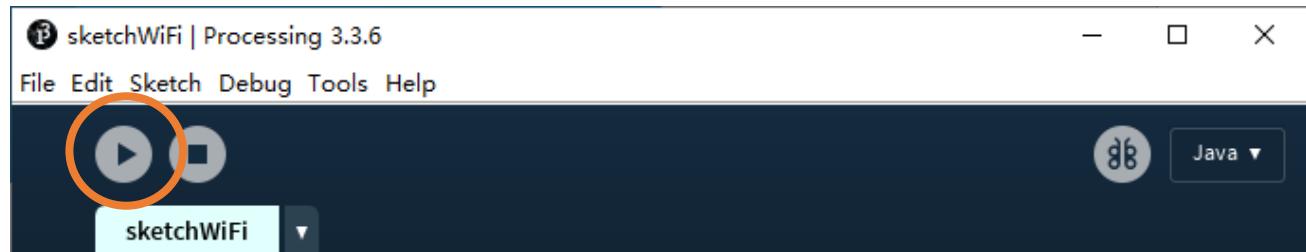
Circuit

Connect Freenove ESP32 to the computer using USB cable.

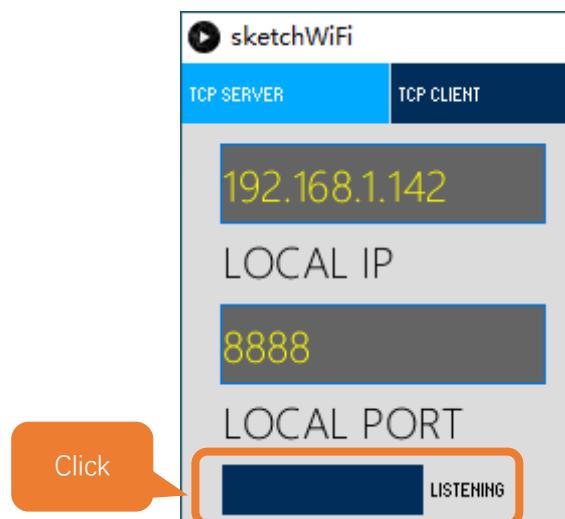


Code

Before running the Code, please open “sketchWiFi.pde.” first, and click “Run”.



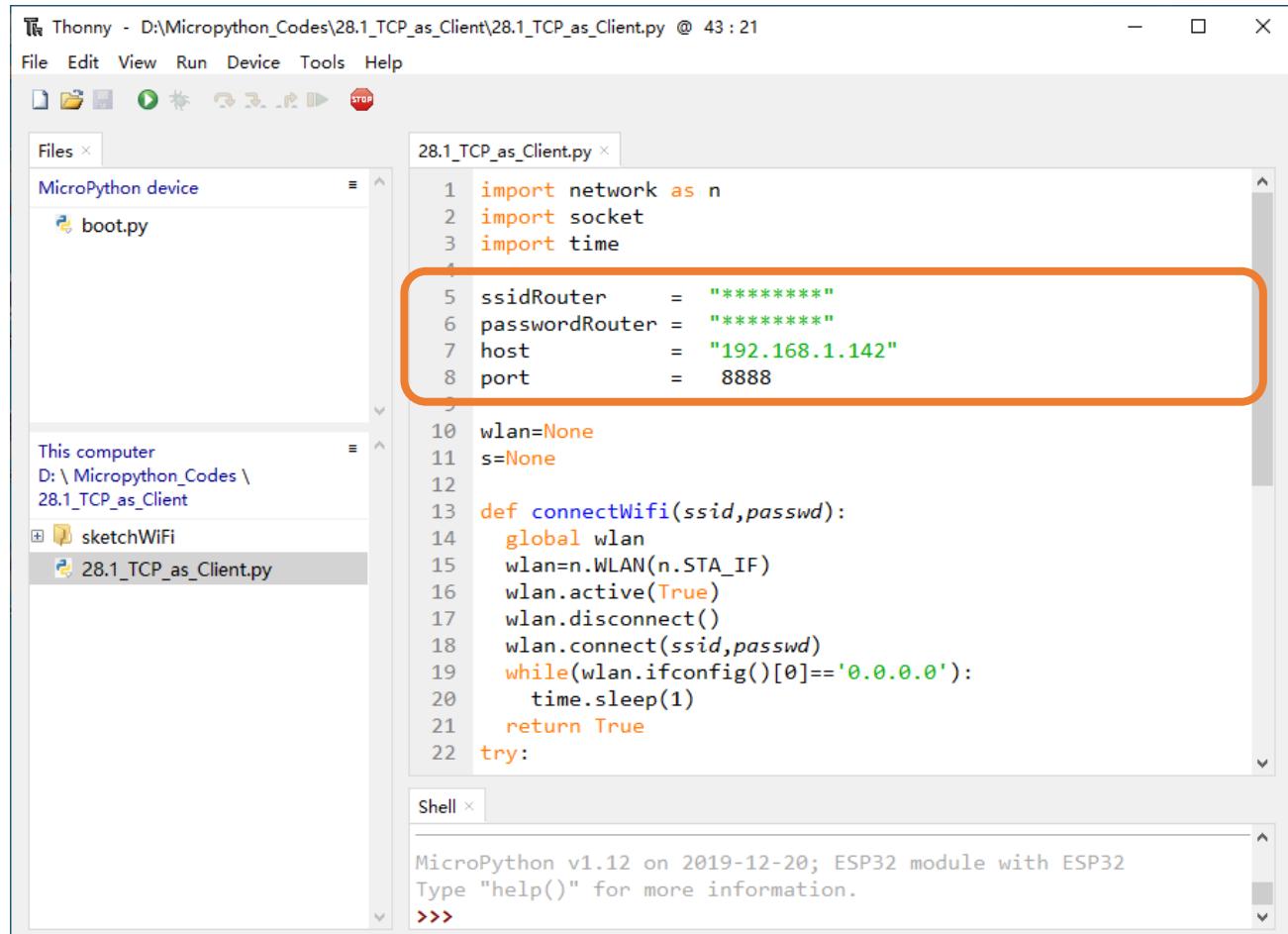
The newly pop up window will use the computer's IP address by default and open a data monitor port. Click “Listening”.



Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “28.1_TCP_as_Client” and double click “28.1_TCP_as_Client.py”.

Before clicking “Run current script”, please modify the name and password of your router and fill in the “host” and “port” according to the IP information shown in the box below:

28.1 TCP_as_Client



```

1 import network as n
2 import socket
3 import time
4
5 ssidRouter      = "*****"
6 passwordRouter = "*****"
7 host            = "192.168.1.142"
8 port            = 8888
9
10 wlan=None
11 s=None
12
13 def connectWifi(ssid,passwd):
14     global wlan
15     wlan=n.WLAN(n.STA_IF)
16     wlan.active(True)
17     wlan.disconnect()
18     wlan.connect(ssid,passwd)
19     while(wlan.ifconfig()[0]=='0.0.0.0'):
20         time.sleep(1)
21     return True
22 try:

```

Click “Run current script” and in “Shell”, you can see ESP32-WROVER automatically connects to sketchWiFi.



```

MicroPython v1.12 on 2019-12-20; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

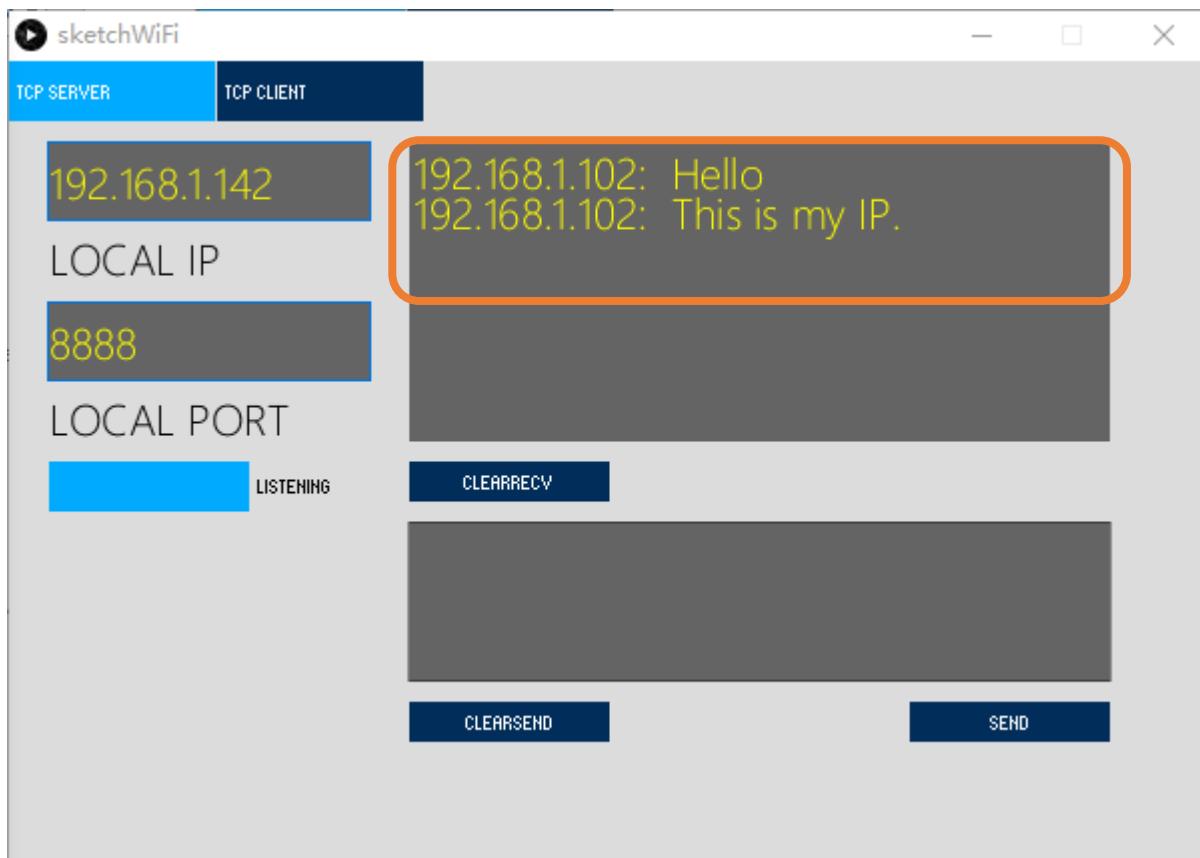
TCP Connected to: 192.168.1.142 : 8888

```

If you don't click "Listening" for sketchWiFi, ESP32-WROVER will fail to connect and will print information as follows:

```
Shell x
for connected to: 192.168.1.142 : 8888
Close socket
>>> %Run -c $EDITOR_CONTENT
TCP close, please reset!
>>>
```

ESP32 connects with TCP SERVER, and TCP SERVER receives messages from ESP32, as shown in the figure below.



The following is the program code:

```
1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"          #Enter the router name
6 passwordRouter = "*****"          #Enter the router password
7 host           = "*****"          #input the remote server
8 port           = 8888              #input the remote port
9
10 wlan=None
11 s=None
```

```

12
13 def connectWifi(ssid,passwd):
14     global wlan
15     wlan= network.WLAN(network.STA_IF)
16     wlan.active(True)
17     wlan.disconnect()
18     wlan.connect(ssid,passwd)
19     while(wlan.ifconfig()[0]=='0.0.0.0'):
20         time.sleep(1)
21     return True
22 try:
23     connectWifi(ssidRouter,passwordRouter)
24     s = socket.socket()
25     s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
26     s.connect((host,port))
27     print("TCP Connected to:", host, ":", port)
28     s.send('Hello')
29     s.send('This is my IP.')
30     while True:
31         data = s.recv(1024)
32         if(len(data) == 0):
33             print("Close socket")
34             s.close()
35             break
36         print(data)
37         ret=s.send(data)
38 except:
39     print("TCP close, please reset!")
40     if (s):
41         s.close()
42     wlan.disconnect()
43     wlan.active(False)

```

Import network、socket、time modules.

```

1 import network
2 import socket
3 import time

```

Enter the actual router name, password, remote server IP address, and port number.

```

5 ssidRouter      = "*****"      #Enter the router name
6 passwordRouter = "*****"      #Enter the router password
7 host           = "*****"      #input the remote server
8 port           = 8888          #input the remote port

```

Connect specified Router until it is successful.

```
13 def connectWifi(ssid,passwd):  
14     global wlan  
15     wlan= network.WLAN(network.STA_IF)  
16     wlan.active(True)  
17     wlan.disconnect()  
18     wlan.connect(ssid,passwd)  
19     while (wlan.ifconfig()[0]=='0.0.0.0'):  
20         time.sleep(1)  
21     return True
```

Connect router and then connect it to remote server.

```
23 connectWifi(ssidRouter,passwordRouter)  
24 s = socket.socket()  
25 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  
26 s.connect((host,port))  
27 print("TCP Connected to:", host, ":", port)
```

Send messages to the remote server, receive the messages from it and print them out, and then send the messages back to the server.

```
28 s.send('Hello')  
29 s.send('This is my IP.')  
30 while True:  
31     data = s.recv(1024)  
32     if(len(data) == 0):  
33         print("Close socket")  
34         s.close()  
35         break  
36     print(data)  
37     ret=s.send(data)
```

If an exception occurs in the program, for example, the remote server is shut down, execute the following program, turn off the socket function, and disconnect the WiFi.

```
39 print("TCP close, please reset!")  
40 if (s):  
41     s.close()  
42     wlan.disconnect()  
43     wlan.active(False)
```



Reference

Class socket

Before each use of **socket**, please add the statement “**import socket**” to the top of the python file.

socket([af, type, proto]): Create a socket.

af: address

socket.AF_INET: IPv4

socket.AF_INET6: IPv6

type: type

socket.SOCK_STREAM : TCP stream

socket.SOCK_DGRAM : UDP datagram

socket.SOCK_RAW : Original socket

socket.SO_REUSEADDR : socket reusable

proto: protocol number

socket.IPPROTO_TCP: TCPmode

socket.IPPROTO_UDP: UDPmode

socket.setsockopt(level, optname, value): Set the socket according to the options.

Level: Level of socket option

socket.SOL_SOCKET: Level of socket option. By default, it is 4095.

optname: Options of socket

socket.SO_REUSEADDR: Allowing a socket interface to be tied to an address that is already in use.

value: The value can be an integer or a bytes-like object representing a buffer.

socket.connect(address): To connect to server.

Address: Tuple or list of the server's address and port number

send(bytes): Send data and return the bytes sent.

recv(bufsize): Receive data and return a bytes object representing the data received.

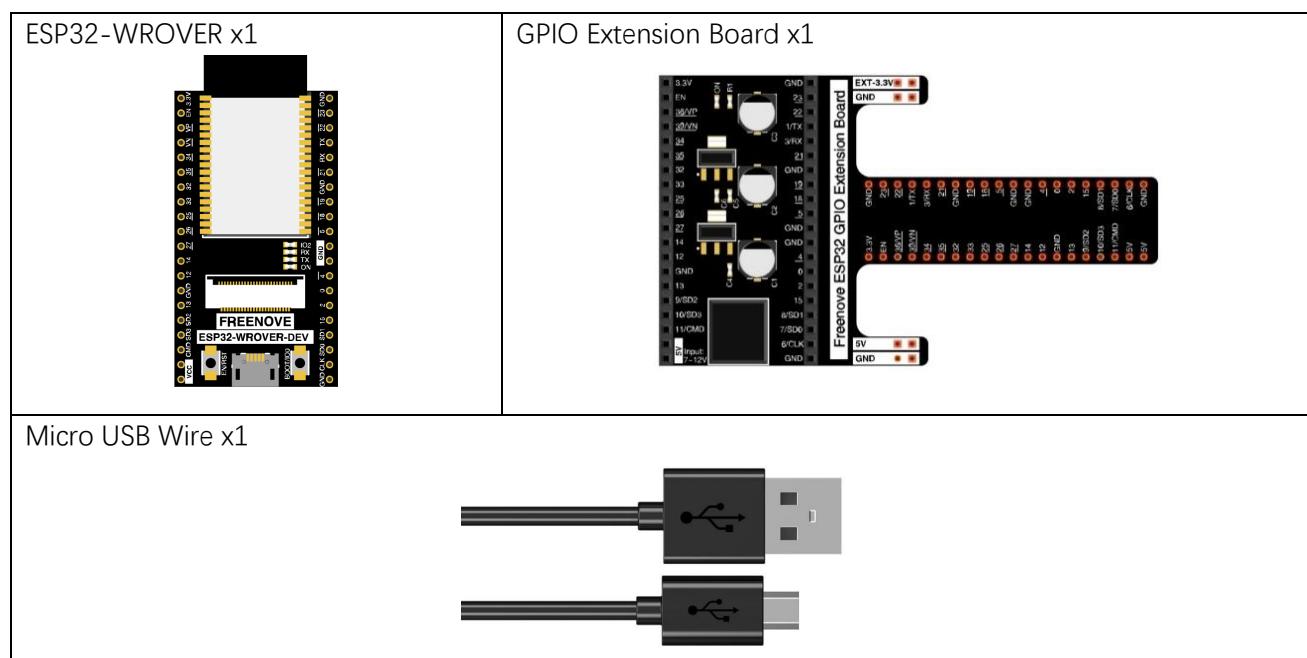
close(): Close socket.

To learn more please visit: <http://docs.micropython.org/en/latest/>

Project 28.2 As Server

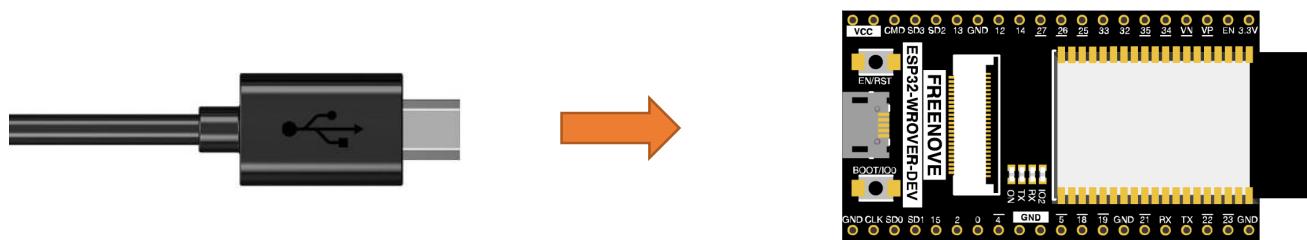
In this section, ESP32 is used as a Server to wait for the connection and communication with Client on the same LAN.

Component List



Circuit

Connect Freenove ESP32 to the computer using the USB cable.





Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “28.2_TCP_as_Server” and double click “28.2_TCP_as_Server.py”.

Before clicking “Run current script”, please modify the name and password of your router shown in the box below.

28.2 TCP_as_Server

```

1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"          #Enter the router name
6 passwordRouter = "*****"          #Enter the router password
7 port            = 8000             #Input the remote port
8
9 wlan=None
10 listenSocket=None
11
12 def connectWifi(ssid,passwd):
13     global wlan
14     wlan=network.WLAN(network.STA_IF)
15     wlan.active(True)
16     wlan.disconnect()
17     wlan.connect(ssid,passwd)
18     while(wlan.ifconfig()[0]!='0.0.0.0'):

```

After making sure that the router's name and password are correct, click “Run current script” and in “Shell”, you can see a server opened by the ESP32- WROVER waiting to connecting to other network devices.

```

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
tcp waiting...
accepting...
Server IP: 192.168.1.102          Port: 8000

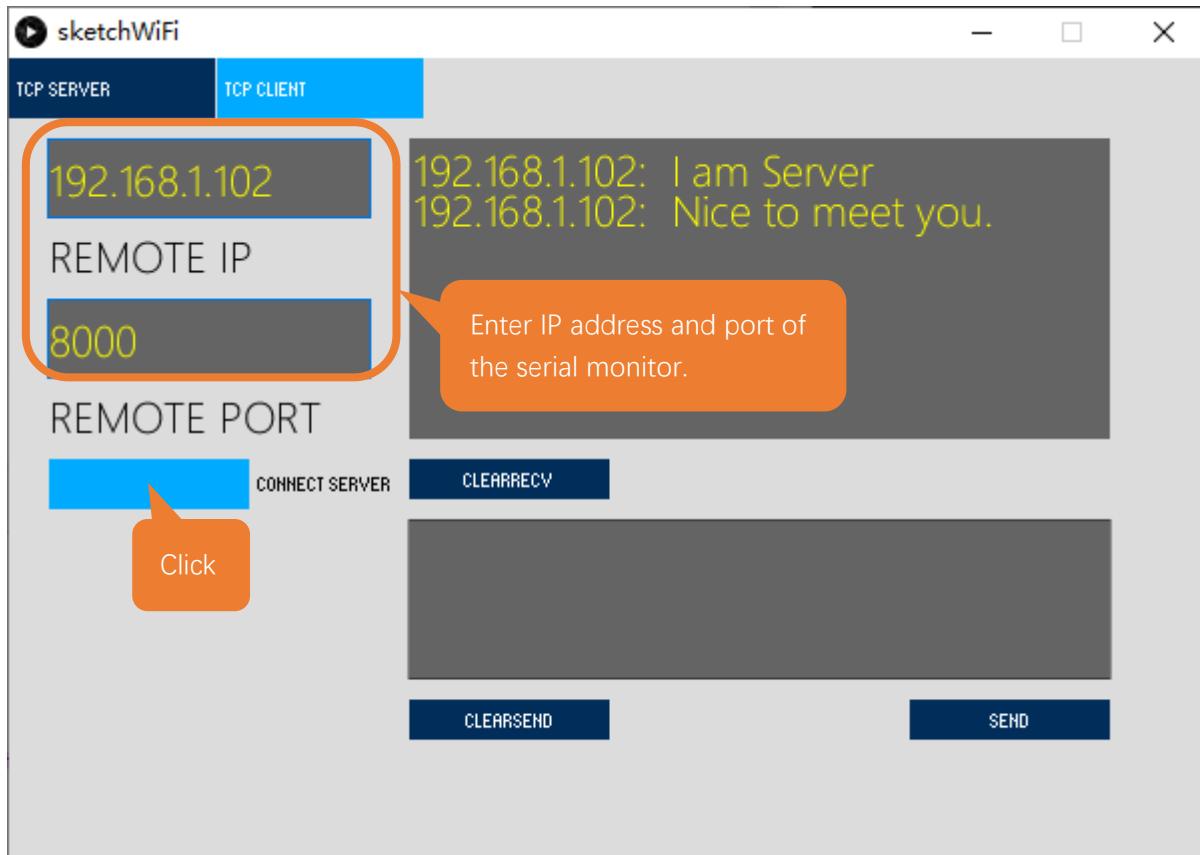
```

IP address and port

Processing:

Open the “**Freenove Ultimate Starter Kit for ESP32\MicroPython_Codes\28.2_TCP_as_Server\sketchWiFi\sketchWiFi.pde**”.

Based on the message printed in "Shell", enter the correct IP address and port when processing, and click to establish a connection with ESP32 to communicate.



You can enter any information in the “Send Box” of sketchWiFi. Click “Send” and ESP32 will print the received messages to “Shell” and send them back to sketchWiFi.

```
Shell ×

MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

tcp waiting...
accepting.....
Server IP: 192.168.1.102      Port: 8000
('192.168.1.142', 51326) connected
b'Nice to meet you.'
```

The following is the program code:

```
1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"          #Enter the router name
6 passwordRouter = "*****"          #Enter the router password
7 port           = 8000             #input the remote port
8 wlan            = None
9 listenSocket    = None
10
11 def connectWifi(ssid,passwd):
12     global wlan
13     wlan=network.WLAN(network.STA_IF)
14     wlan.active(True)
15     wlan.disconnect()
16     wlan.connect(ssid,passwd)
17     while(wlan.ifconfig()[0]=='0.0.0.0'):
18         time.sleep(1)
19     return True
20
21 try:
22     connectWifi(ssidRouter,passwordRouter)
23     ip=wlan.ifconfig()[0]
24     listenSocket = socket.socket()
25     listenSocket.bind((ip,port))
26     listenSocket.listen(1)
27     listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
28     print('tcp waiting... ')
29     while True:
30         print("Server IP:",ip,"\\tPort:",port)
31         print("accepting.....")
32         conn,addr = listenSocket.accept()
33         print(addr, "connected")
34         break
35     conn.send('I am Server')
36     while True:
37         data = conn.recv(1024)
38         if(len(data) == 0):
39             print("close socket")
40             listenSocket.close()
41             wlan.disconnect()
42             wlan.active(False)
43             break
```

```

44     else:
45         print(data)
46         ret = conn.send(data)
47     except:
48         print("Close TCP-Server, please reset.")
49         if(listenSocket):
50             listenSocket.close()
51             wlan.disconnect()
52             wlan.active(False)

```

Call function `connectWifi()` to connect to router and obtain the dynamic IP that it assigns to ESP32.

```

22     connectWifi(ssidRouter, passwordRouter)
23     ip=wlan.ifconfig()[0]

```

Open the socket server, bind the server to the dynamic IP, and open a data monitoring port.

```

24     listenSocket = socket.socket()
25     listenSocket.bind((ip, port))
26     listenSocket.listen(1)
27     listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

```

Print the server's IP address and port, monitor the port and wait for the connection of other network devices.

```

29     while True:
30         print("Server IP:", ip, "\tPort:", port)
31         print("accepting.....")
32         conn, addr = listenSocket.accept()
33         print(addr, "connected")
34         break

```

Each time receiving data, print them in "Shell" and send them back to the client.

```

36     while True:
37         data = conn.recv(1024)
38         if(len(data) == 0):
39             print("close socket")
40             listenSocket.close()
41             wlan.disconnect()
42             wlan.active(False)
43             break
44         else:
45             print(data)
46             ret = conn.send(data)

```

If the client is disconnected, close the server and disconnect WiFi.

```

47     except:
48         print("Close TCP-Server, please reset.")
49         if(listenSocket):
50             listenSocket.close()
51             wlan.disconnect()
52             wlan.active(False)

```

Chapter 29 Soldering Circuit Board

Project 29.1 Soldering a Buzzer

We have tried to use a buzzer in a previous chapter, and now we will solder a circuit that when the button is pressed, the buzzer sounds.

This circuit doesn't need programming and can work when it is powered on. And when the button is not pressed, there is no power consumption.

You can install it on your bike, bedroom door or any other places where it is needed.

Component List

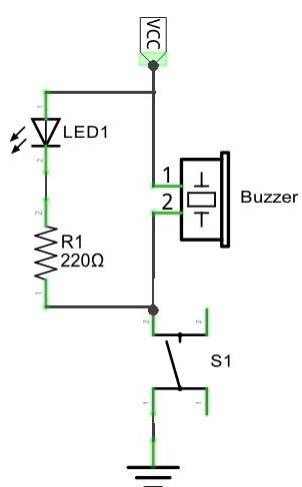
Pin header x2	LED x1	Resistor 220Ω x1	Active buzzer x1	Push button x1
				

AA Battery Holder x1


Circuit

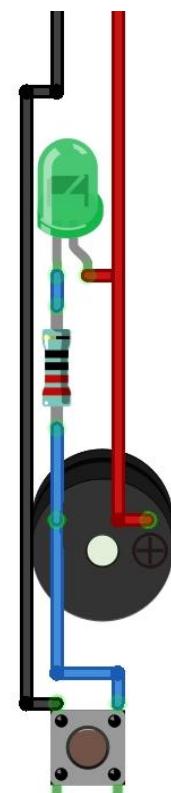
We will solder the following circuit on the main board.

Schematic diagram



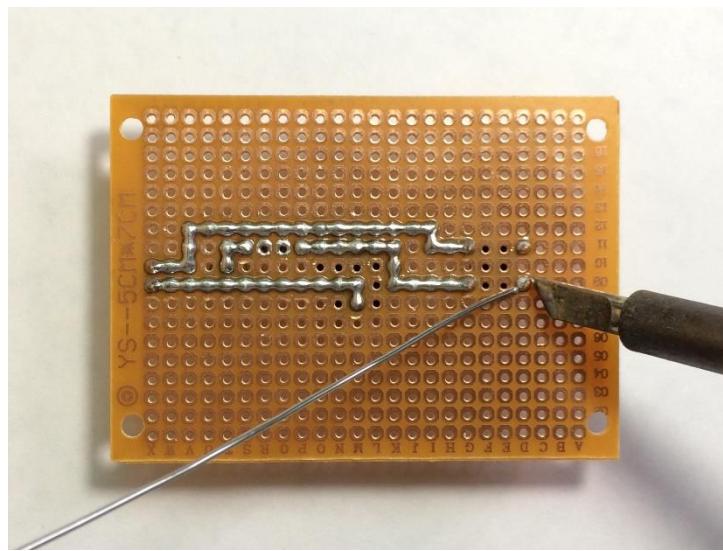
Hardware connection.

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

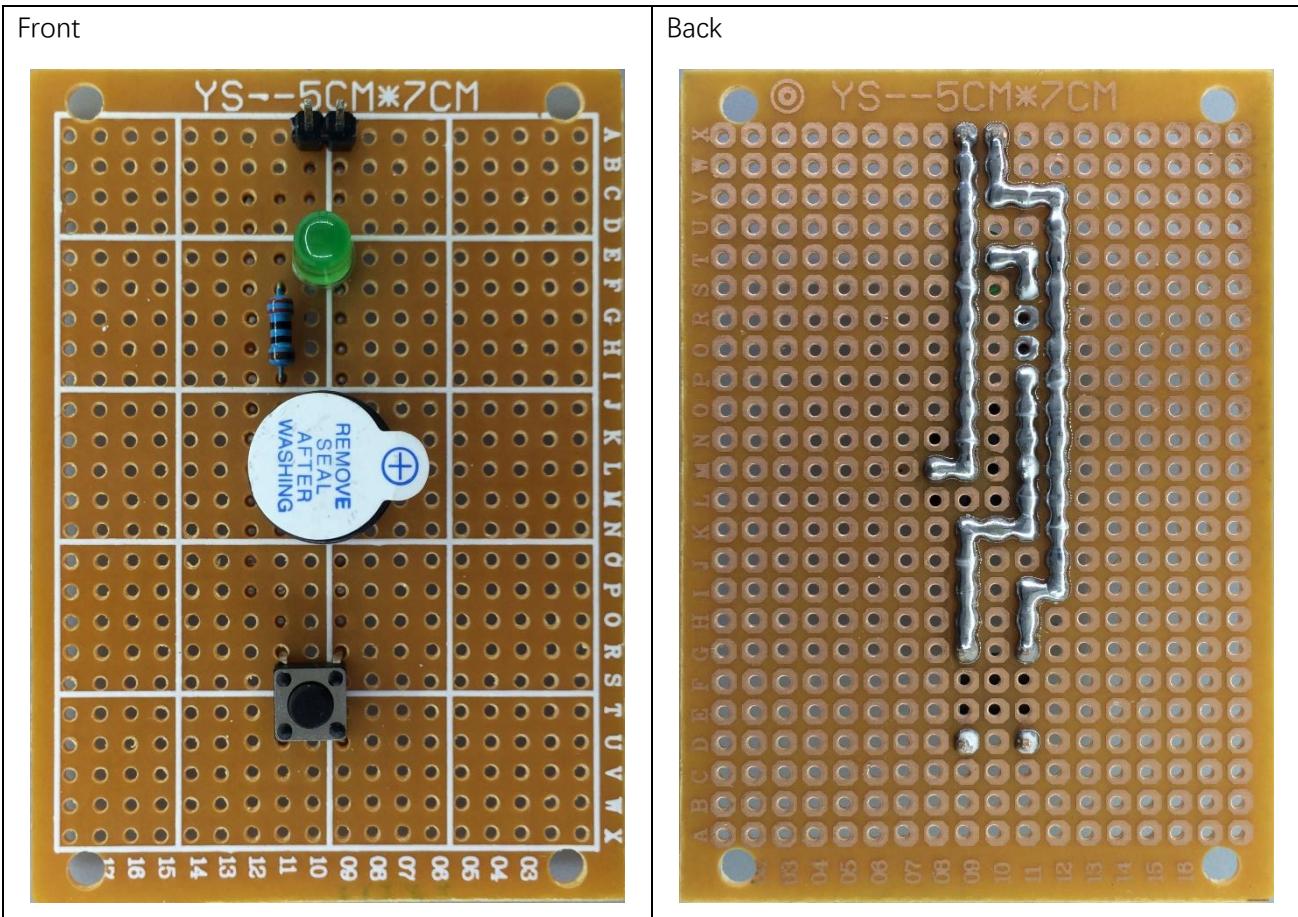


Solder the Circuit

Insert the components on the main board and solder the circuit on its back.

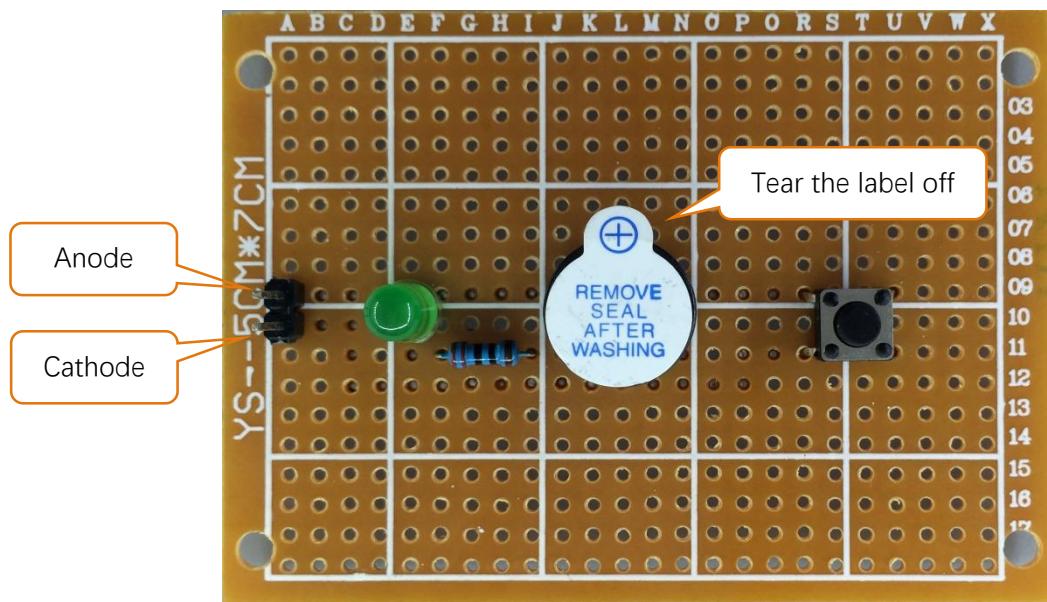


Rendering after soldering:



Test circuit

Connect the circuit board to power supply (3~5V). You can use ESP32 board or battery box as the power supply.



Press the push button after connecting the power, and then the buzzer will make a sound.

Project 29.2 Soldering a Flowing Water Light

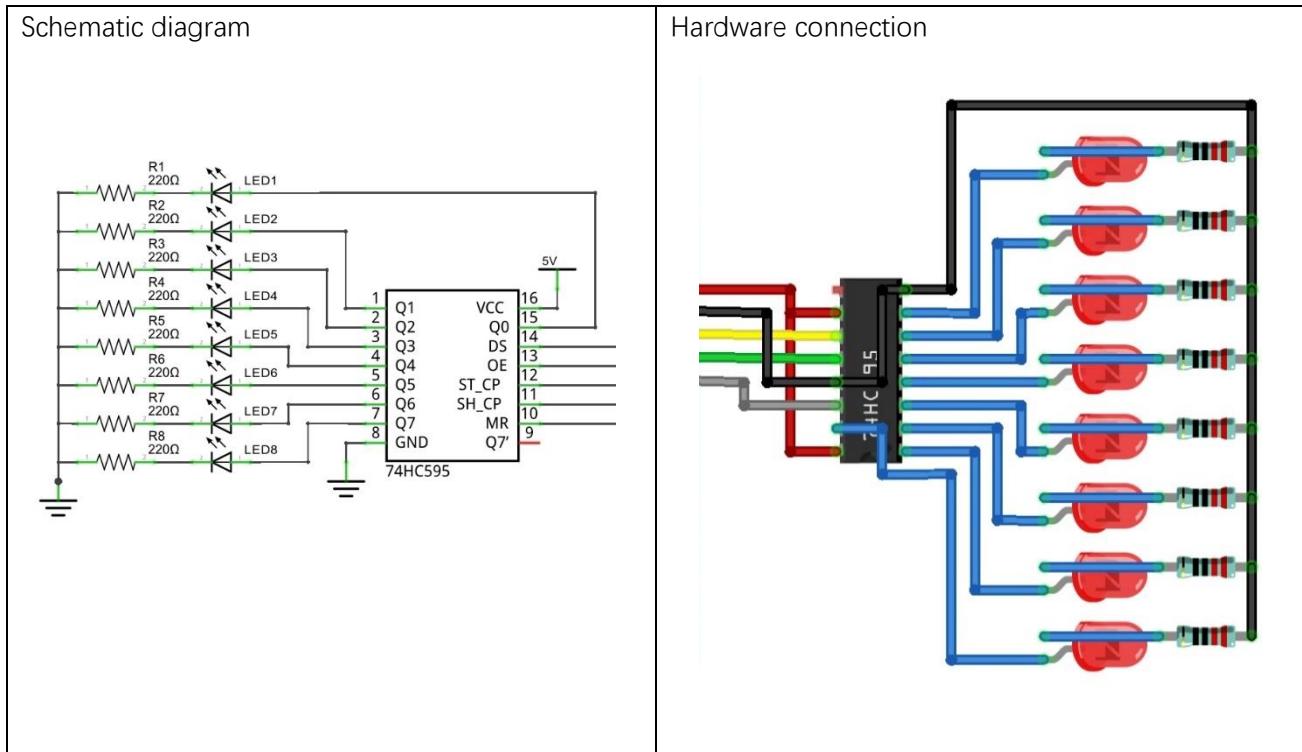
From previous chapter, we have learned to make a flowing water light with LED. Now, we will solder a circuit board, and use the improved code to make a more interesting flowing water light.

Component List

Pin header x5	Resistor 220Ω x8	LED x1	74HC595 x1

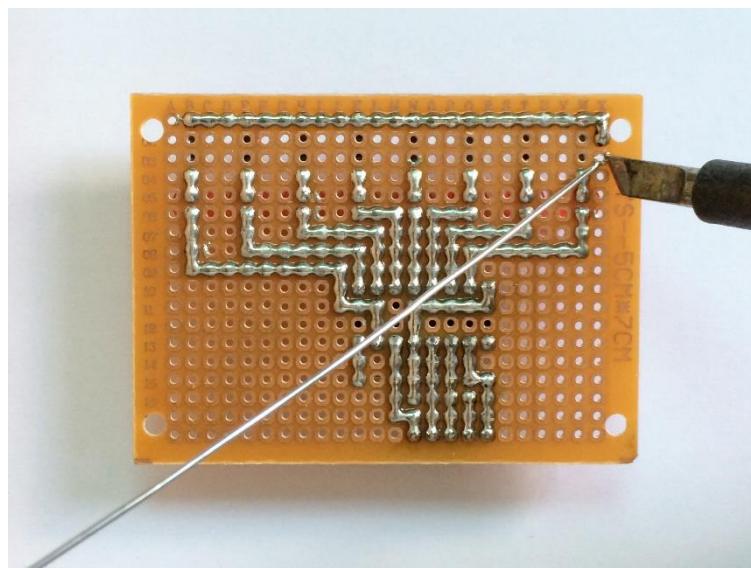
Circuit

Solder the following circuit on the main board.

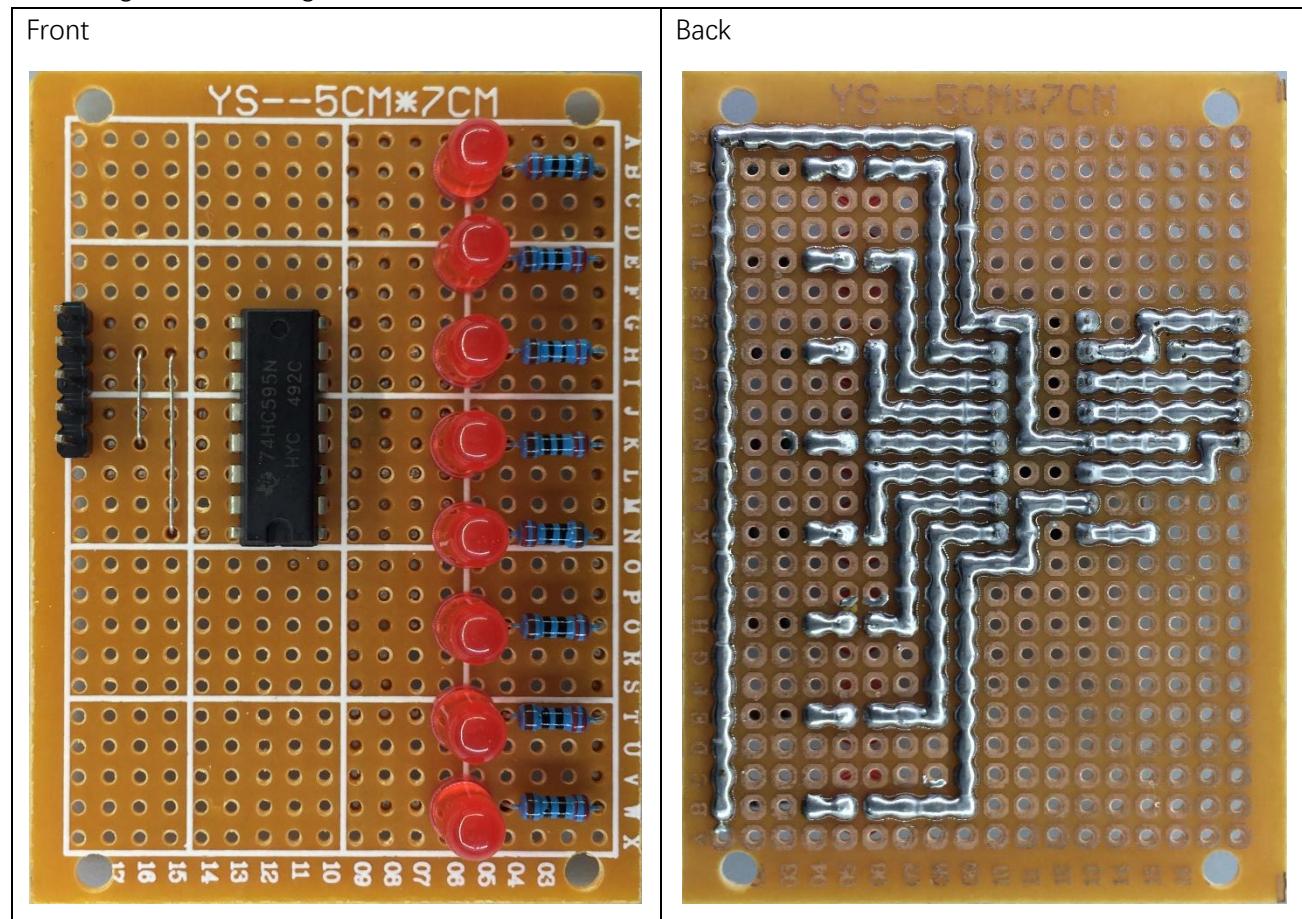


Soldering the Circuit

Insert the components on the main board and solder the circuit on its back.

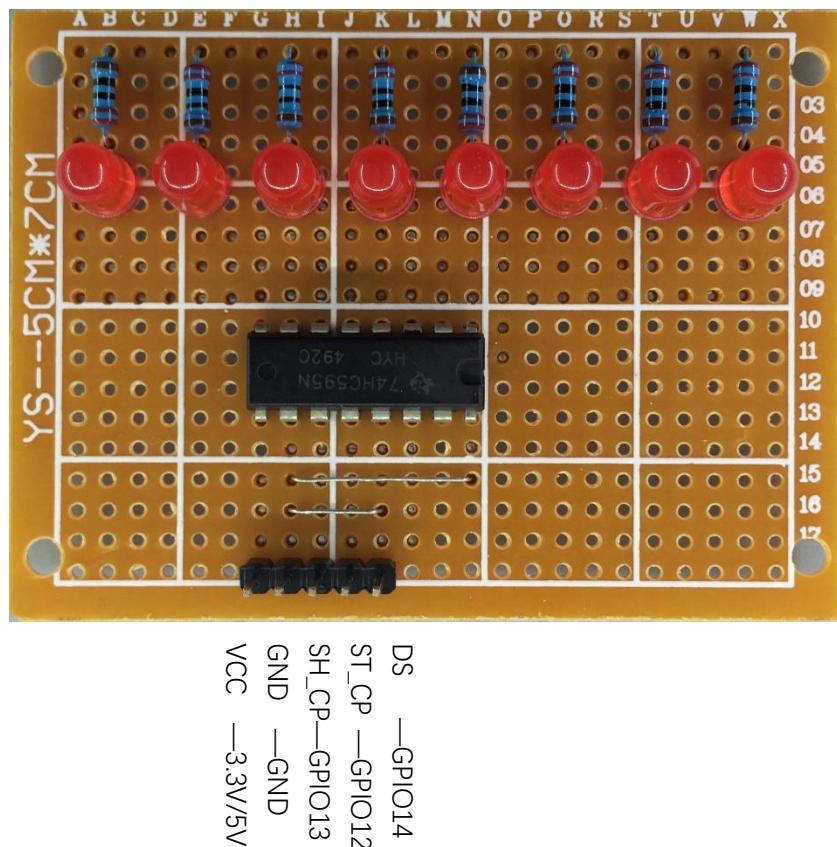


Rendering after soldering:



Connect the Circuit

Connect the board to ESP32 with jumper wire in the following way.



Code

The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 chip = Chip74HC595(14, 12, 13)
5 # ESP32-14: 74HC595-DS (14)
6 # ESP32-12: 74HC595-STCP (12)
7 # ESP32-13: 74HC595-SHCP (11)
8
9 while True:
10     x=0x01
11     for count in range(8):
12         chip.shiftOut(1,x) #High bit is sent first
13         x=x<<1
```

```
14     time.sleep_ms(300)
15     x=0x01
16     for count in range(8):
17         chip.shiftOut(0, x) #Low bit is sent first
18         x=x<<1
19         time.sleep_ms(300)
```

In fact, this code is copied from chapter 15. If you have any questions for the code, please click "[Chapter 15 74HC595 & LED Bar Graph](#)" to return to Chapter 15 to study again.

What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself an ESP32 Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:
support@freenove.com

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.

<http://www.freenove.com/>

Thank you again for choosing Freenove products.