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

FREENOVE™

Contents

Preface

ESP32-S3 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 Micyoco(SCM) chip, or connected to the Internet and used as an Internet of Things device.

ESP32-S3 can be developed using the Arduino platform, which will definitely make it easier for people who have learned arduino to master. Moreover, the code of ESP32-S3 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.

Generally, ESP32-S3 projects consist of code and circuits. Don't worry even if you've never learned code and circuits, because we will gradually introduce the basic knowledge of C programming language and electronic circuits, from easy to difficult. Our products contain all the electronic components and modules needed to complete these projects. It's especially suitable for beginners.

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 SEP32 and 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**

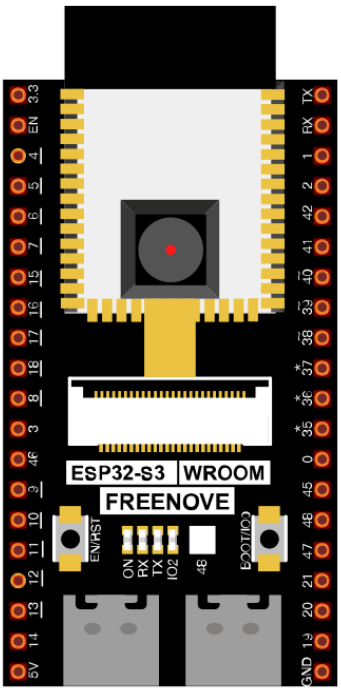
ESP32-S3 WROOM

ESP32-S3-WROOM-1 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.

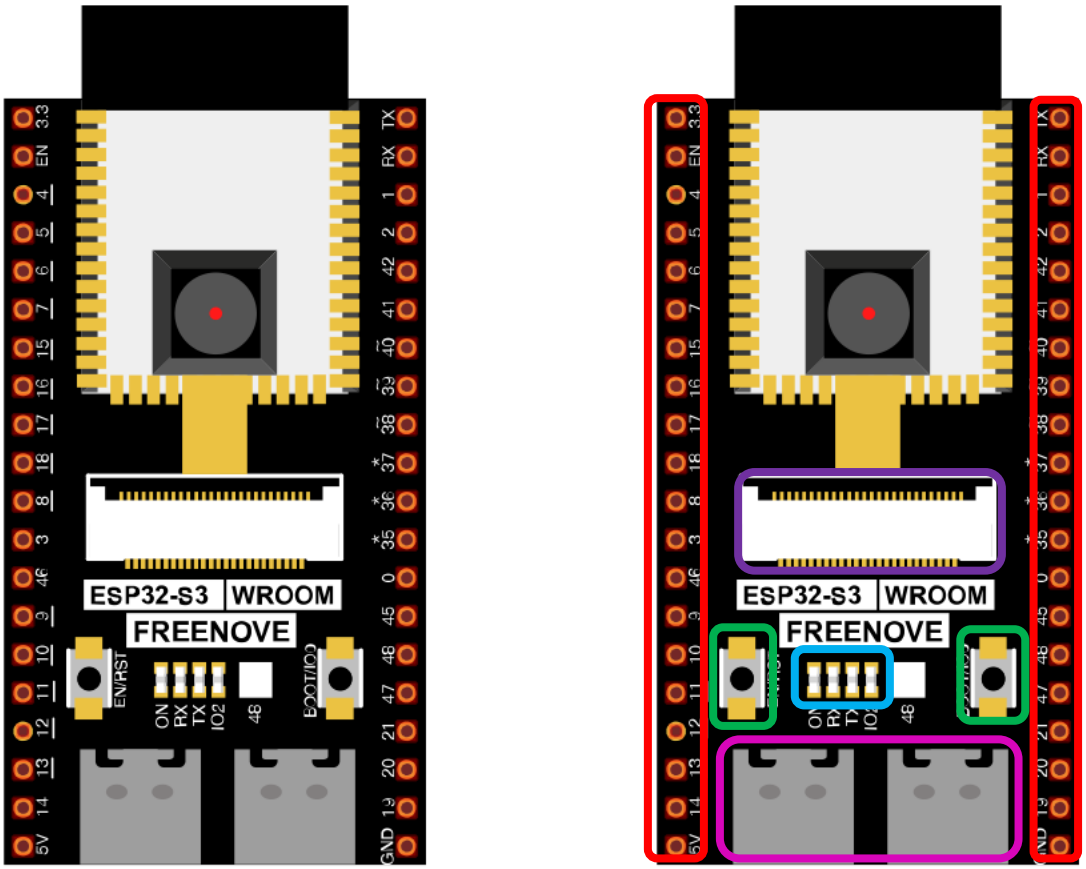
PCB on-board antenna	IPEX antenna
	

In this tutorial, the ESP32-S3 WROOM is designed based on the PCB on-board antenna-packaged ESP32-S3-WROOM-1 module.

ESP32-S3 WROOM



The hardware interfaces of ESP32-S3 WROOM are distributed as follows:



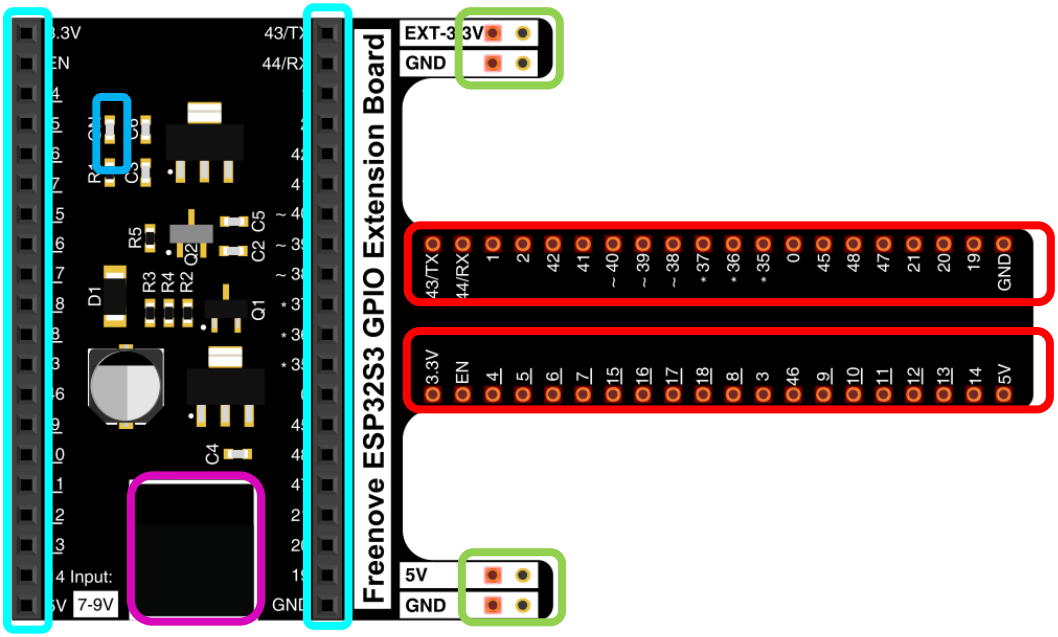
Compare the left and right images. We've boxed off the resources on the ESP32-S3 WROOM in different colors to facilitate your understanding of the ESP32-S3 WROOM.

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

For more information, please visit: https://www.espressif.com.cn/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf.

Extension board of the ESP32-S3 WROOM

And we also design an extension board, so that you can use the ESP32-S3 more easily in accordance with the circuit diagram provided. The followings are their photos.
The hardware interfaces of ESP32-S3 WROOM are distributed as follows:



We've boxed off the resources on the ESP32-S3 WROOM in different colors to facilitate your understanding of the ESP32-S3 WROOM.

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

In ESP32S3, GPIO is an interface to control peripheral circuit.

In the following projects, we only use USB cable to power ESP32-S3 WROOM by default.

In the whole tutorial, we don't use T extension to power ESP32-S3 WROOM. So 5V and 3.3V (including EXT 3.3V) on the extension board are provided by ESP32-S3 WROOM.

We can also use DC jack of extension board to power ESP32-S3 WROOM. In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

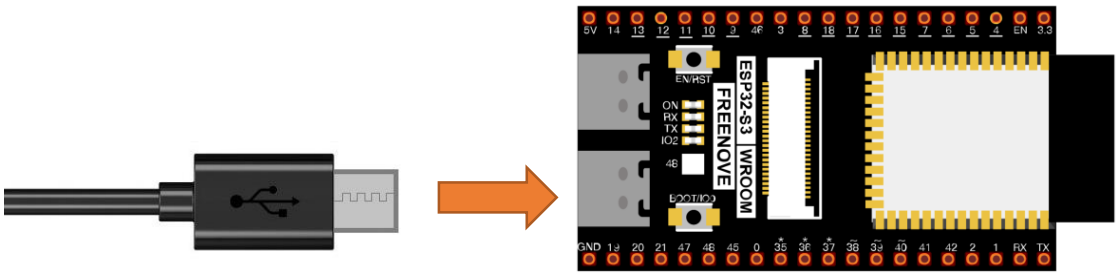
CH343 (Importance)

ESP32-S3 WROOM uses CH343 to download codes. So before using it, we need to install CH343 driver in our computers.

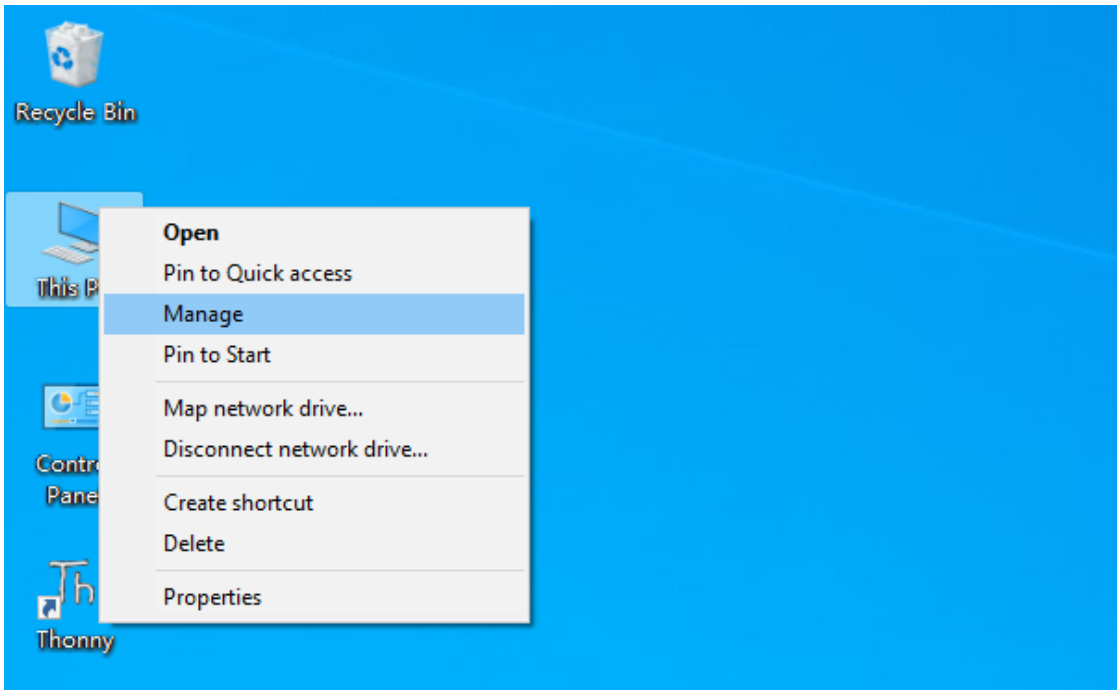
Windows

Check whether CH343 has been installed

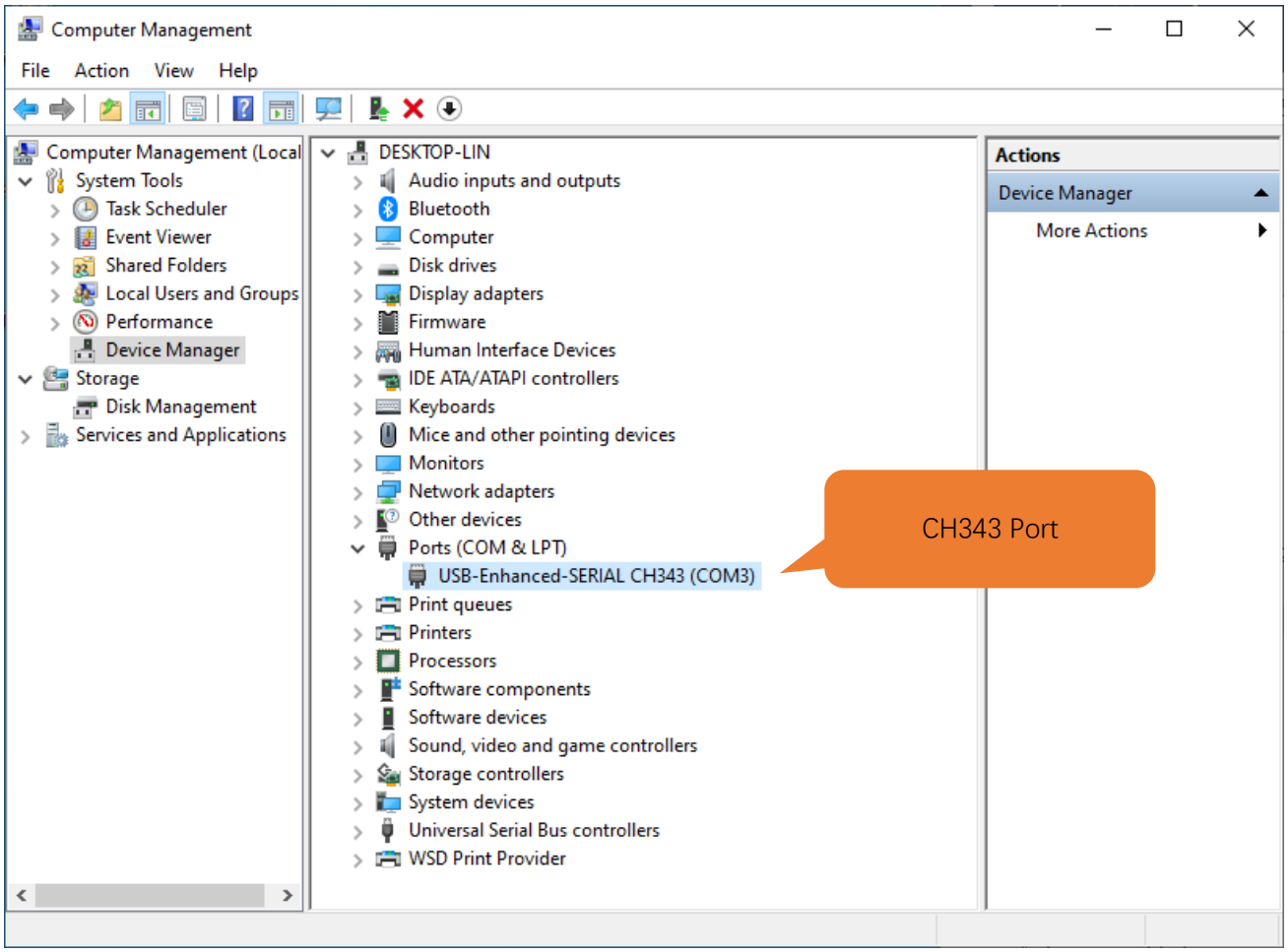
1. Connect your computer and ESP32-S3 WROOM with a USB cable.



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

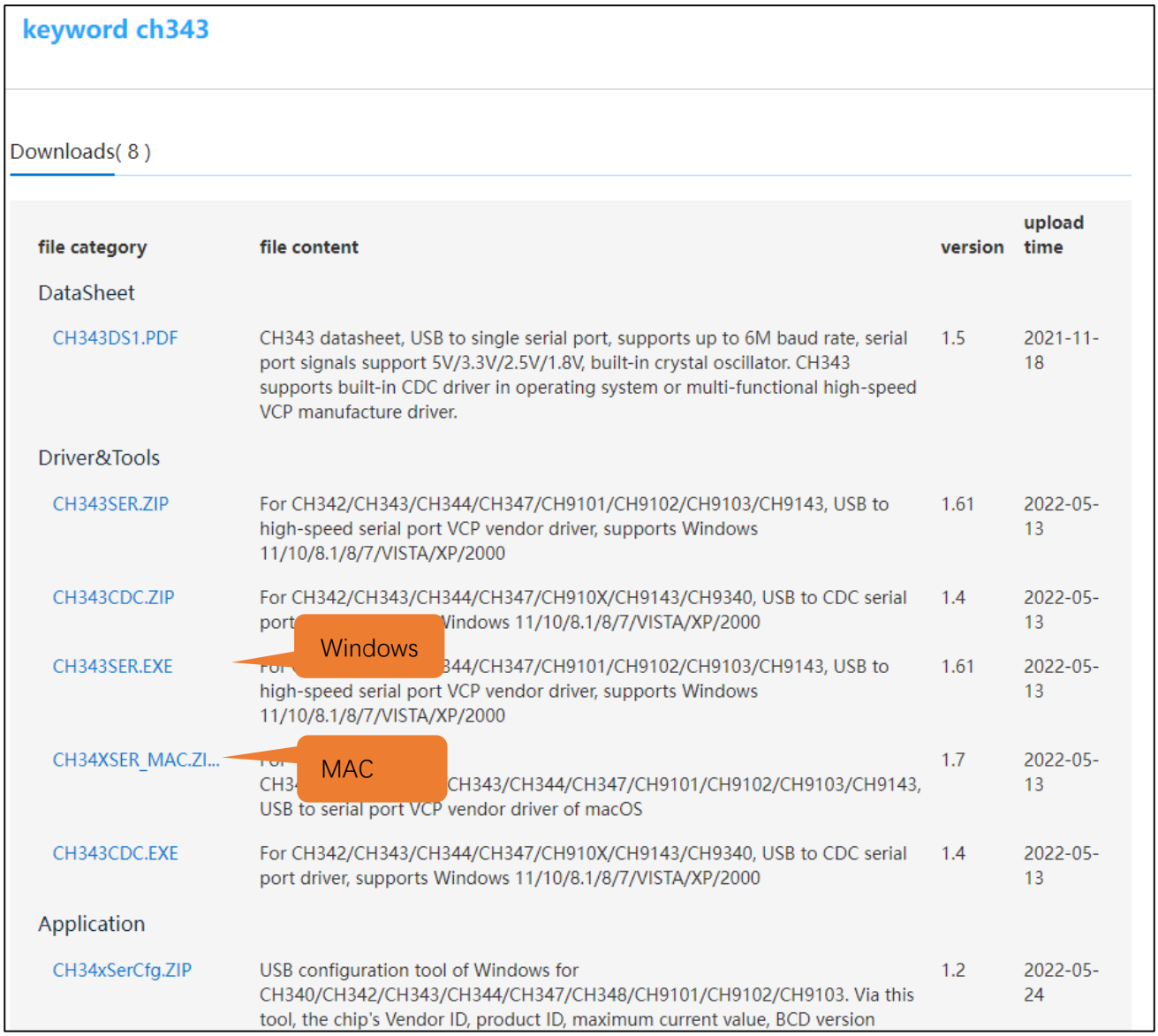


3. Click "Device Manager". If your computer has installed CH343, you can see"USB-Enhances-SERIAL CH343 (COMx)". And you can click here to move to the next step.

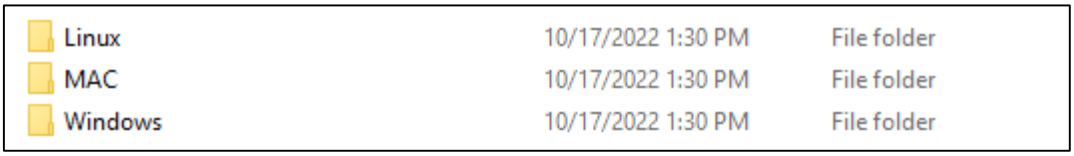


Installing CH343

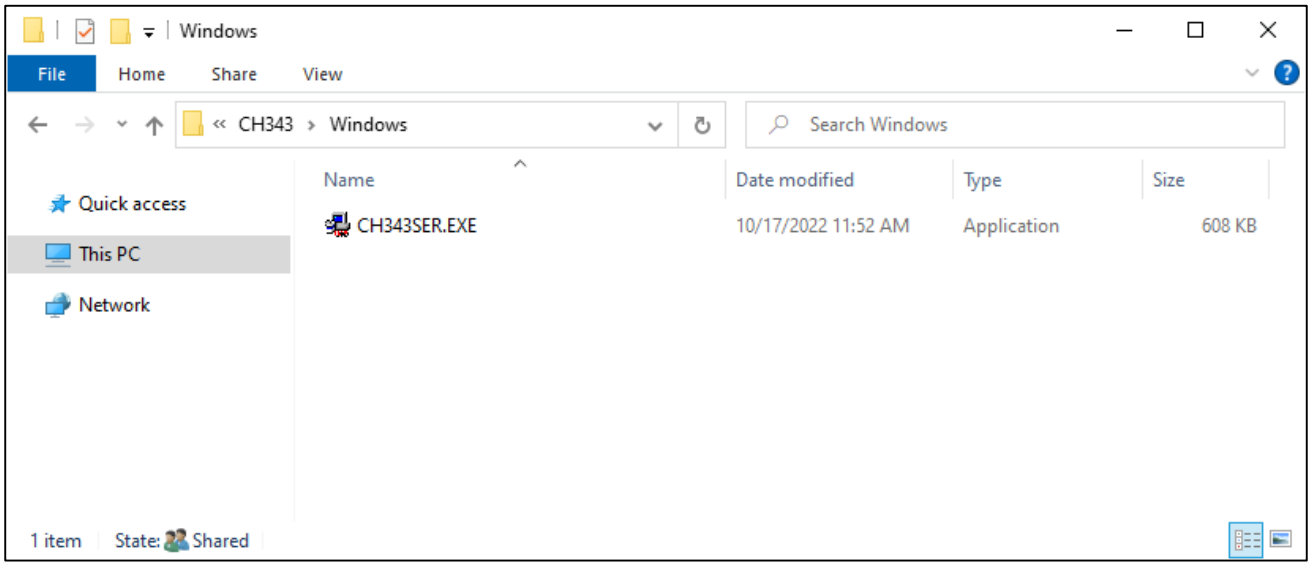
1. First, download CH343 driver, click http://www.wch-ic.com/search?t=all&q=ch343 to download the appropriate one based on your operating system.



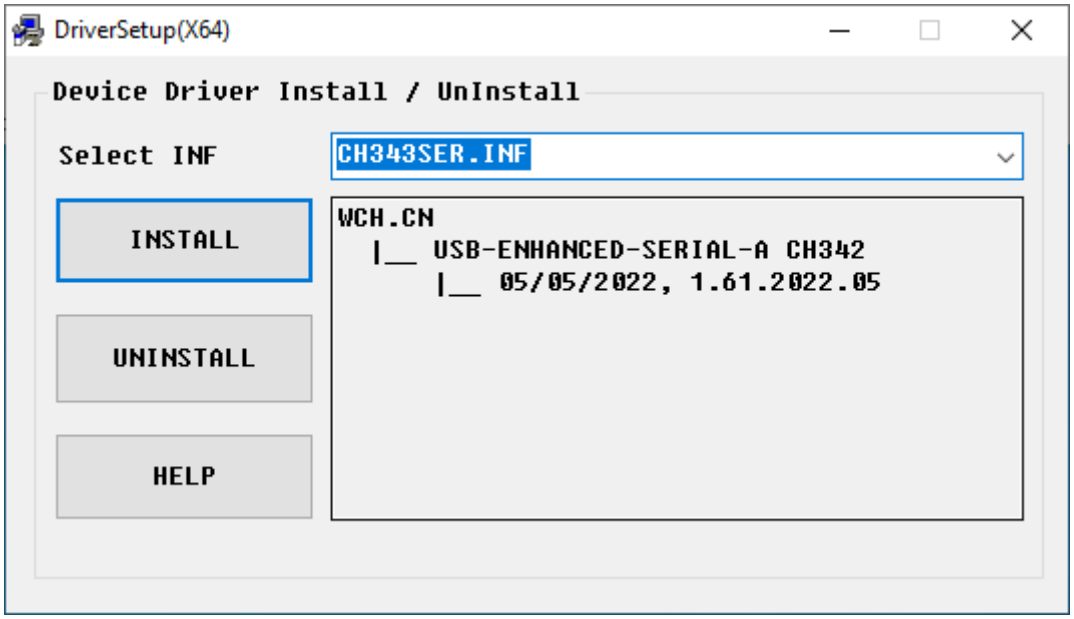
If you would not like to download the installation package, you can open **"Freenove_Super_Starter_Kit_for_ESP32_S3/CH343"**, we have prepared the installation package.



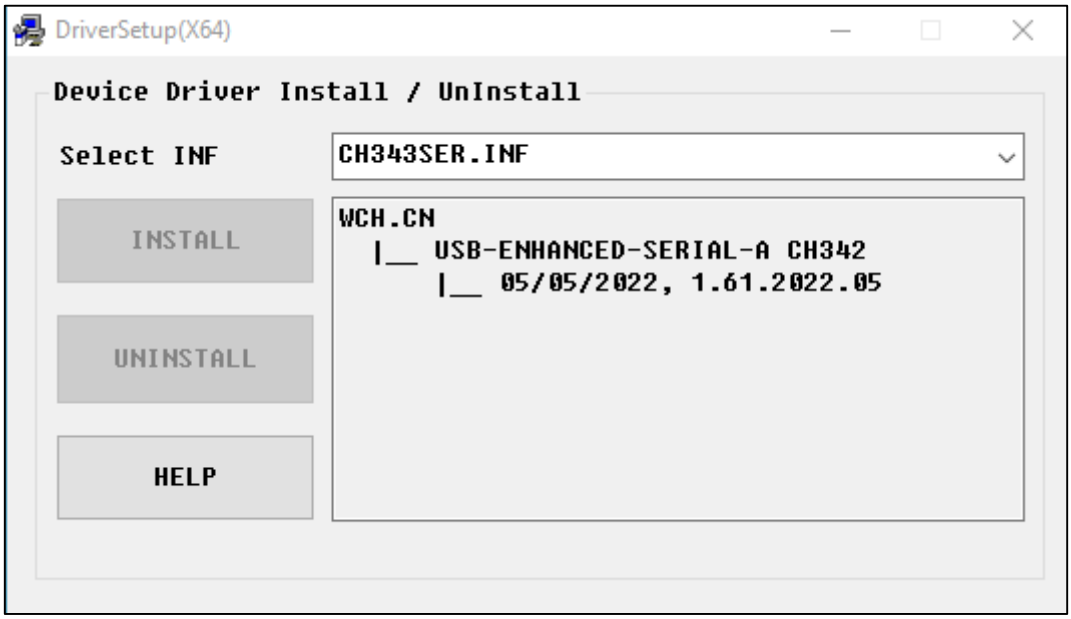
2. Open the folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/CH343/Windows/**"



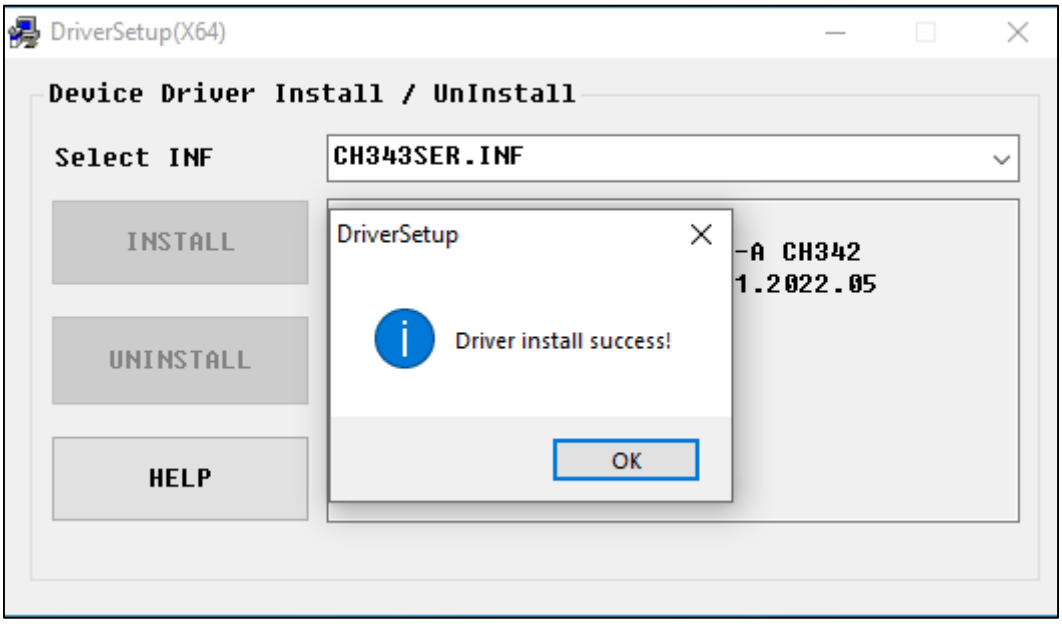
3. Double click "**CH343SER.EXE**".



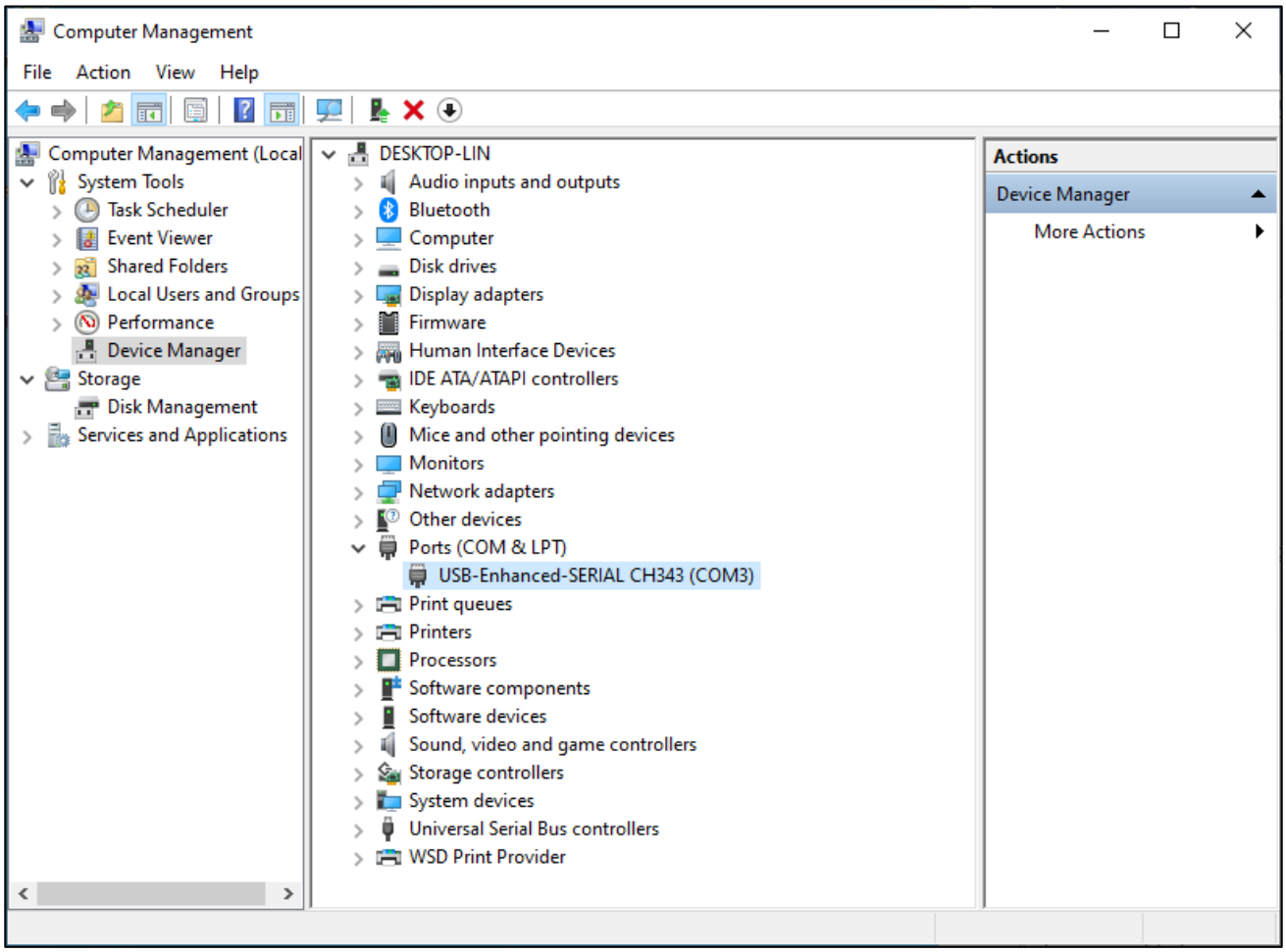
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



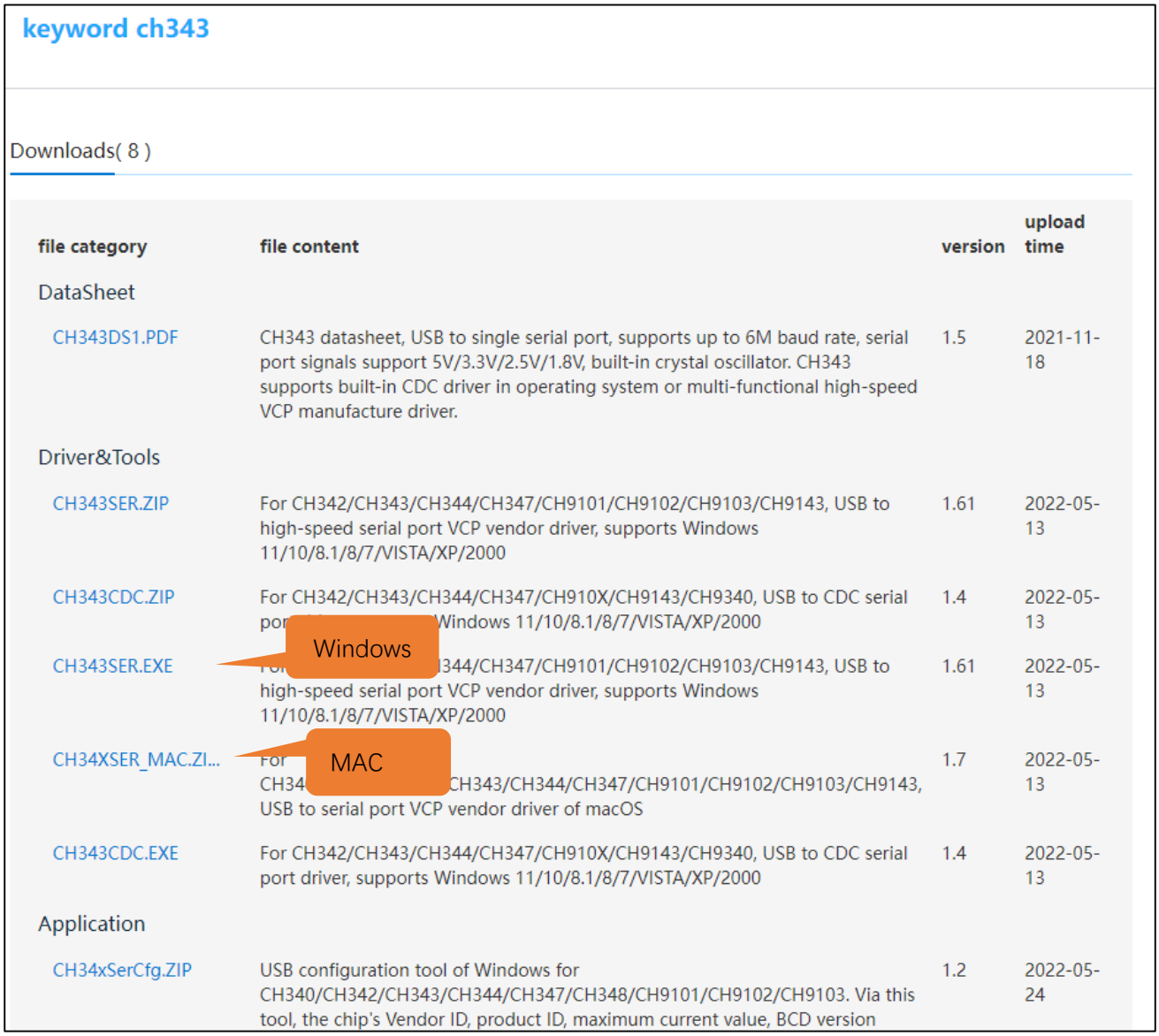
6. When ESP32-S3 WROOM is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



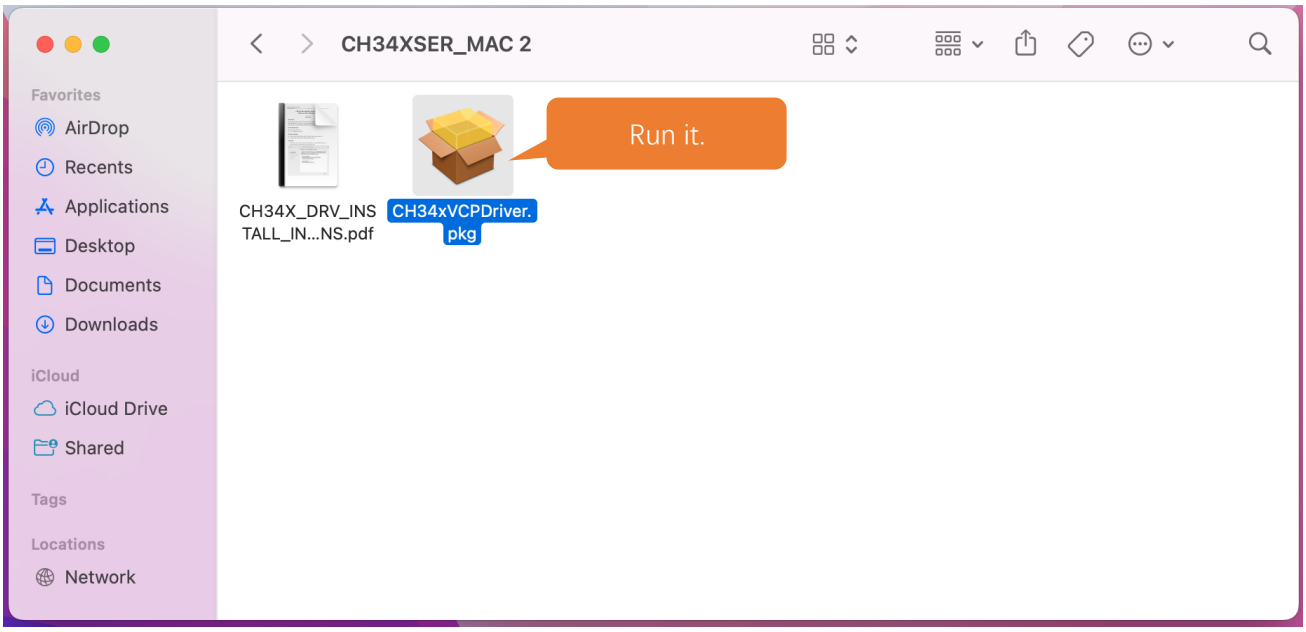
7. So far, CH343 has been installed successfully. Close all dialog boxes.

MAC

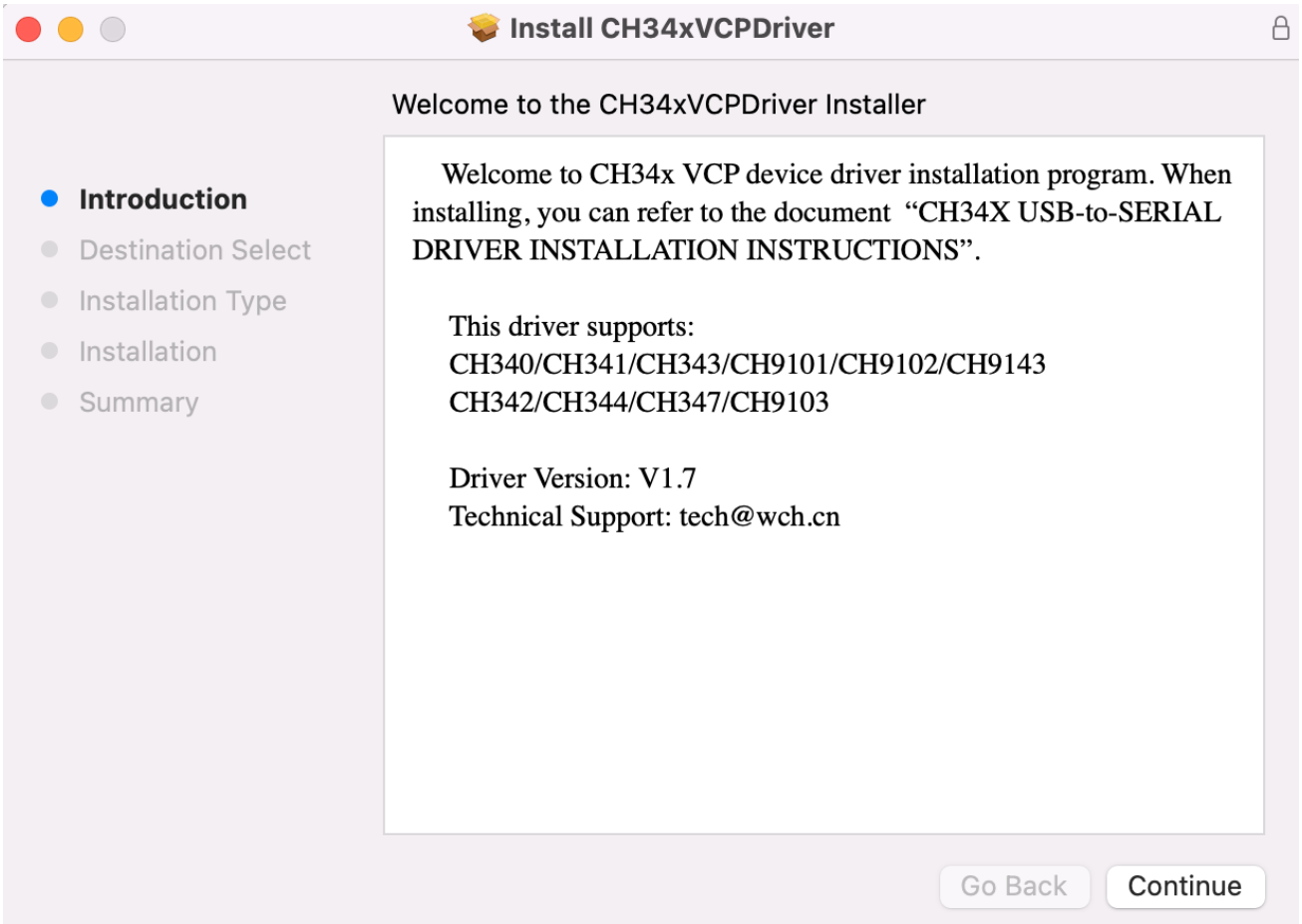
First, download CH343 driver, click http://www.wch-ic.com/search?t=all&q=ch343 to download the appropriate one based on your operating system.



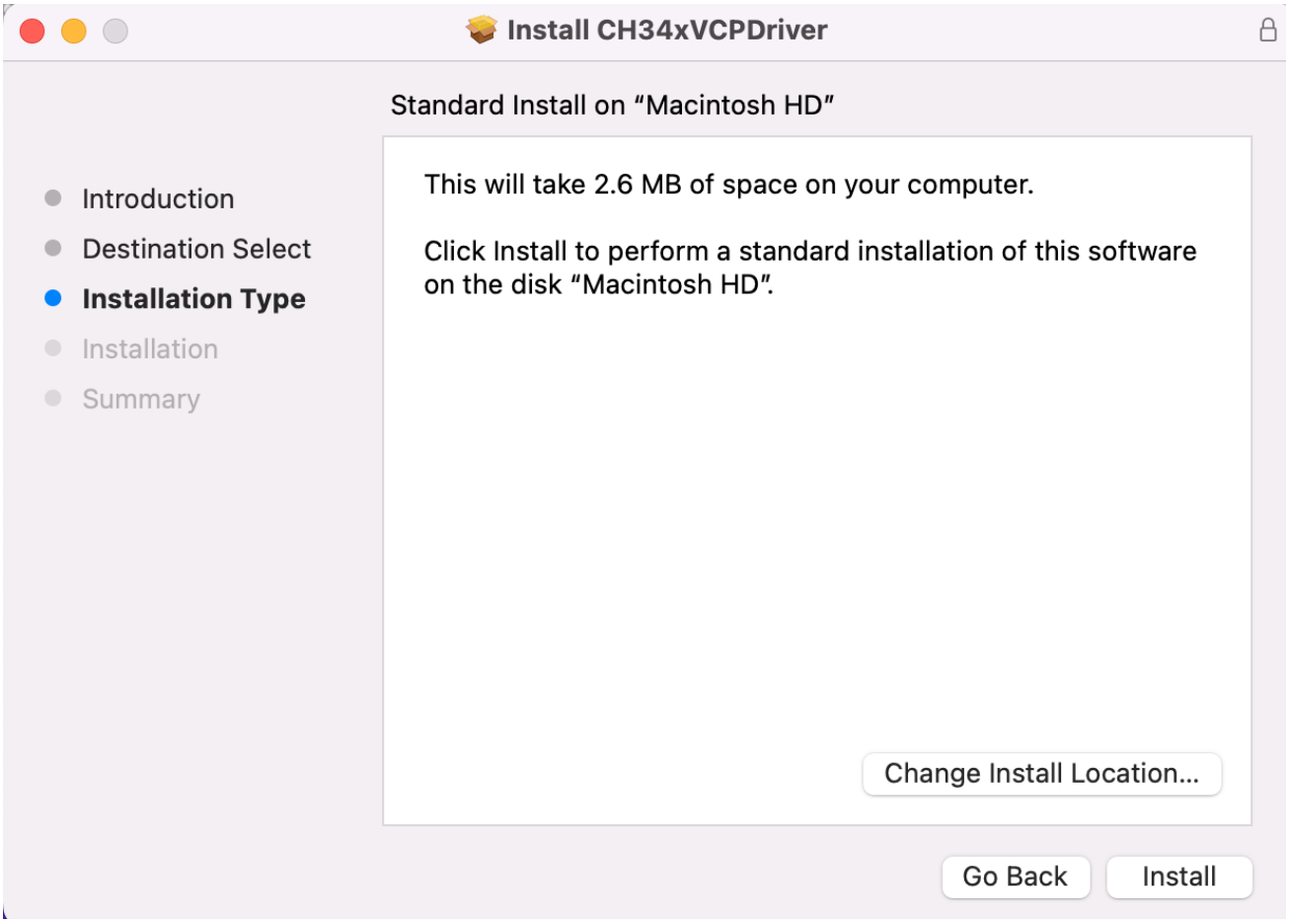
If you would not like to download the installation package, you can open **"Freenove_Super_Starter_Kit_for_ESP32_S3/CH343"**, we have prepared the installation package.
Second, open the folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/CH343/MAC/**"



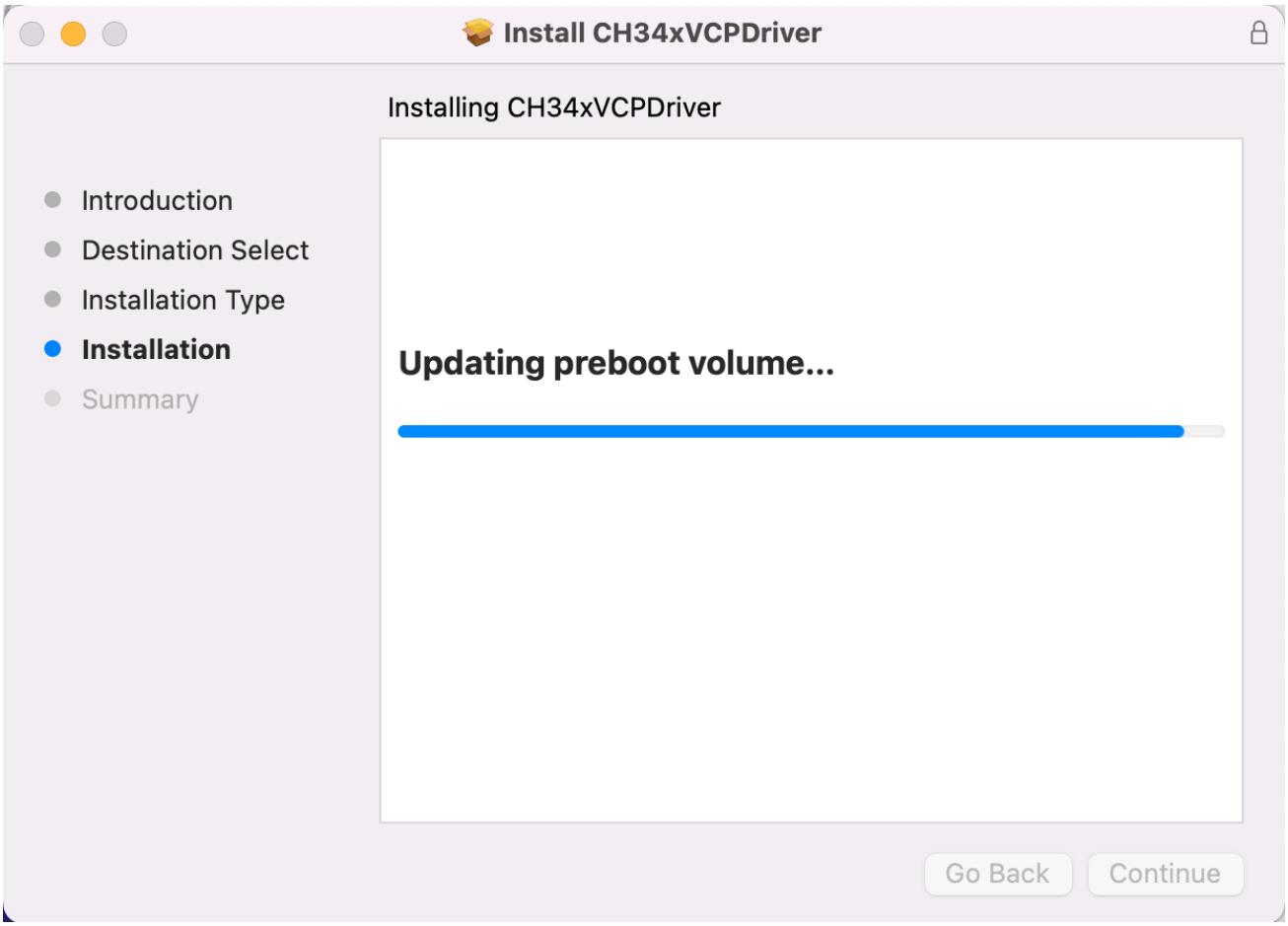
Third, click Continue.



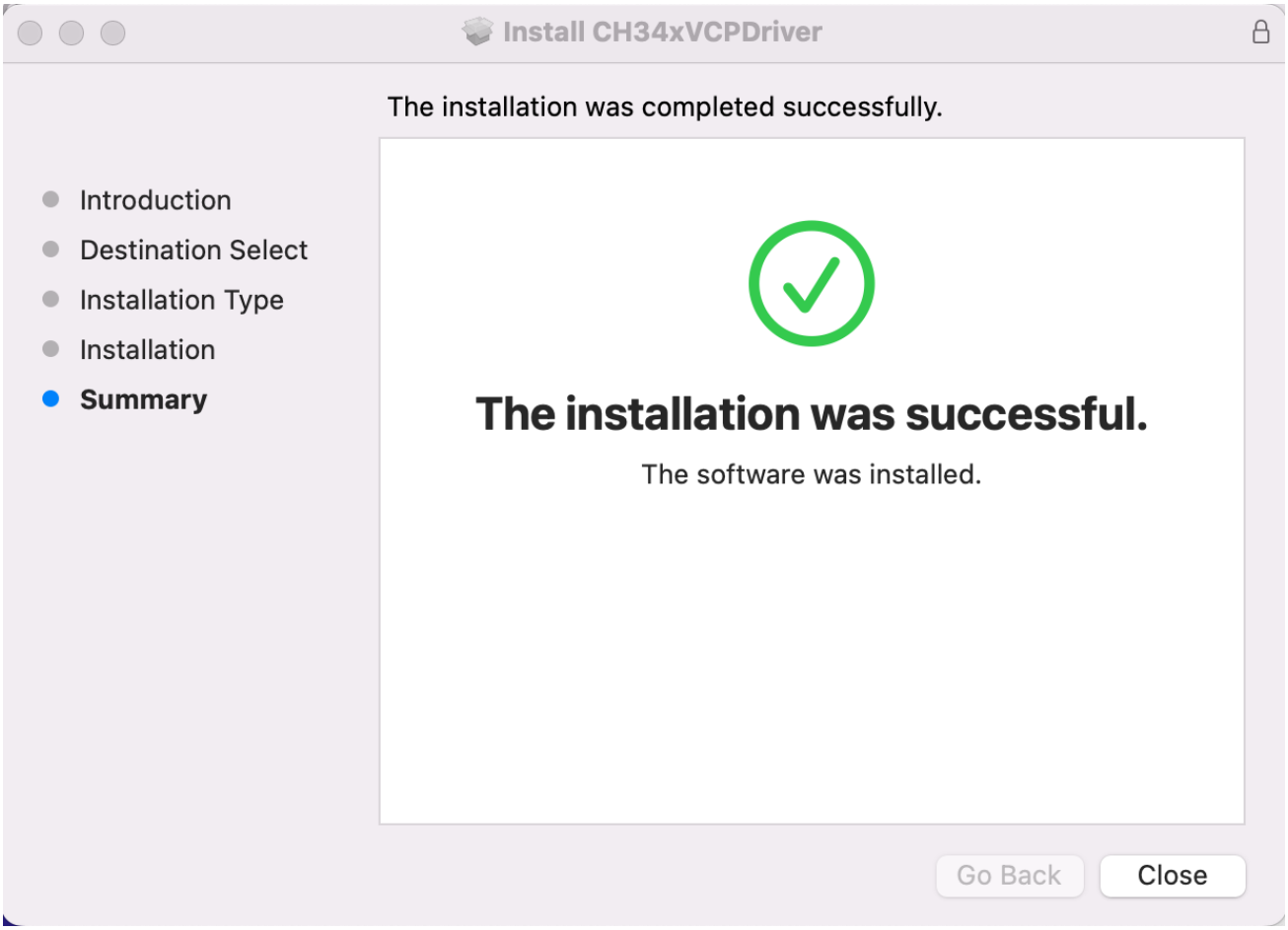
Fourth, click Install.


Install CH34xVCPDriver
Standard Install on "Macintosh HD"
Introduction
Destination Select
Installation Type
Installation
Summary
This will take 2.6 MB of space on your computer.
Click Install to perform a standard installation of this software on the disk "Macintosh HD".
Change Install Location...
Go Back
Install

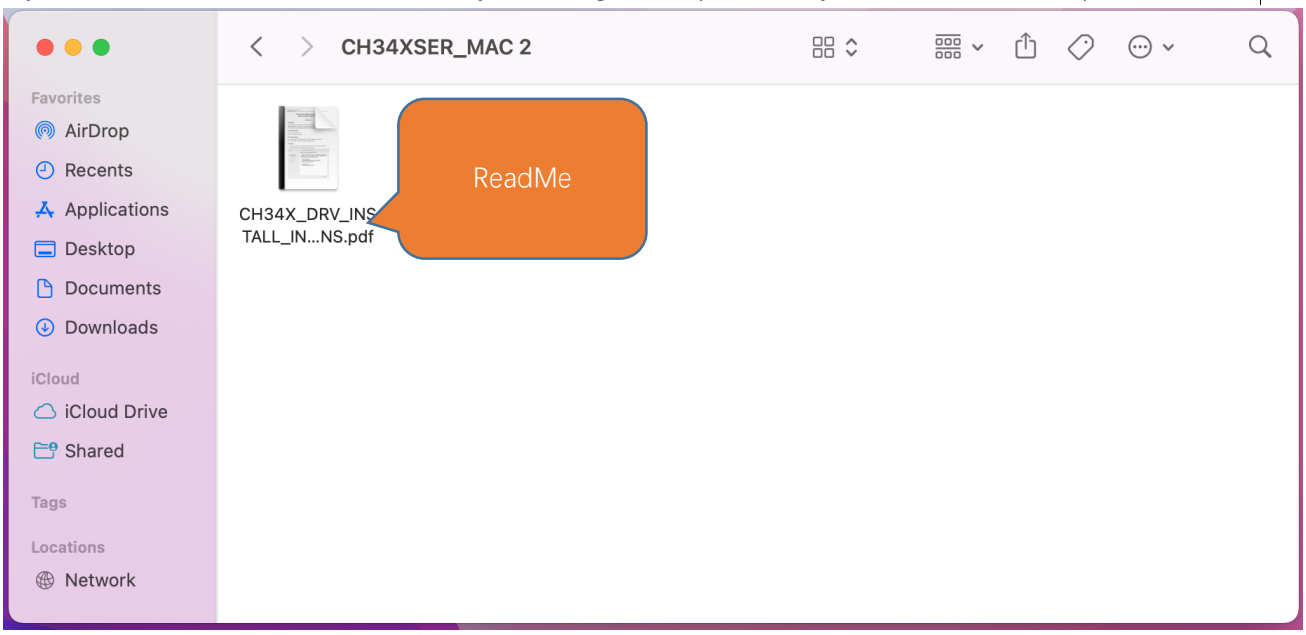
Then, waiting Finsh.



Finally, restart your PC.


Install CH34xVCPDriver
The installation was completed successfully.
Introduction
Destination Select
Installation Type
Installation
Summary
The installation was successful.
The software was installed.
Go Back
Close

If you still haven't installed the CH340 by following the steps above, you can view readme.pdf to install it.



Programming Software

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-S3 during the whole process.
Thonny supports various operating system, including Windows、Mac OS、Linux.

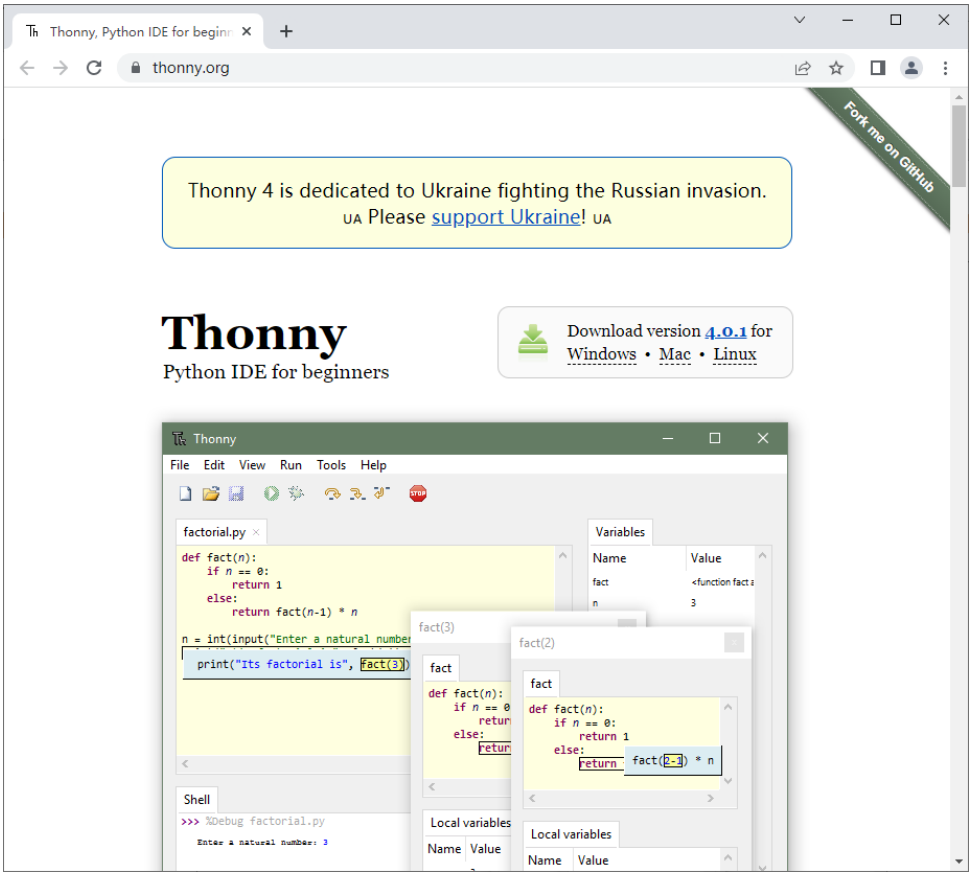
Downloading Thonny

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.)

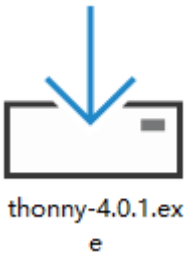
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v4.0.1/thonny-4.0.1.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v4.0.1/thonny-4.0.1.pkg
Linux	**Official downloads for Linux** **Installer** (installs private Python 3.10 on x86_64, uses existing python3 elsewhere) bash <(wget -O - https://thonny.org/installer-for-linux) **Re-using an existing Python installation** (for advanced users) pip3 install thonny **3rd party distributions** (may have older version) **Flatpak** flatpak install org.thonny.Thonny **Debian, Raspbian, Ubuntu, Mint and others** sudo apt install thonny **Fedora** sudo dnf install thonny

You can also open "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Software**", we have prepared it in advance.

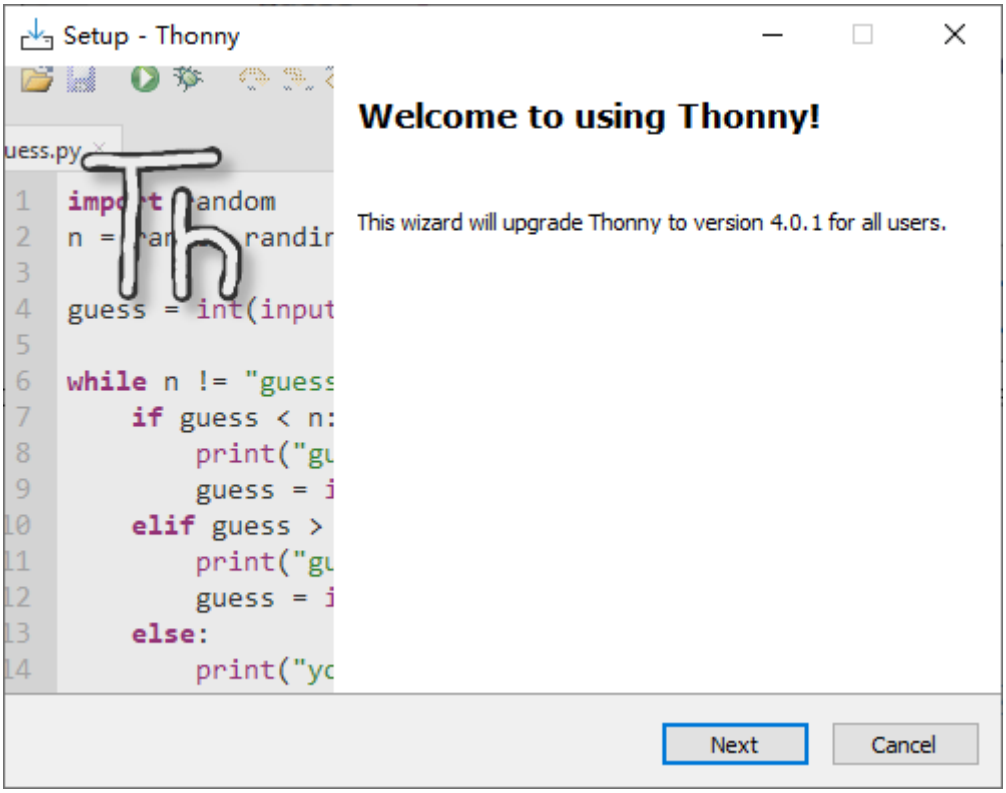


Installing on Windows

The icon of Thonny after downloading is as below. Double click “thonny-4.0.1.exe”.

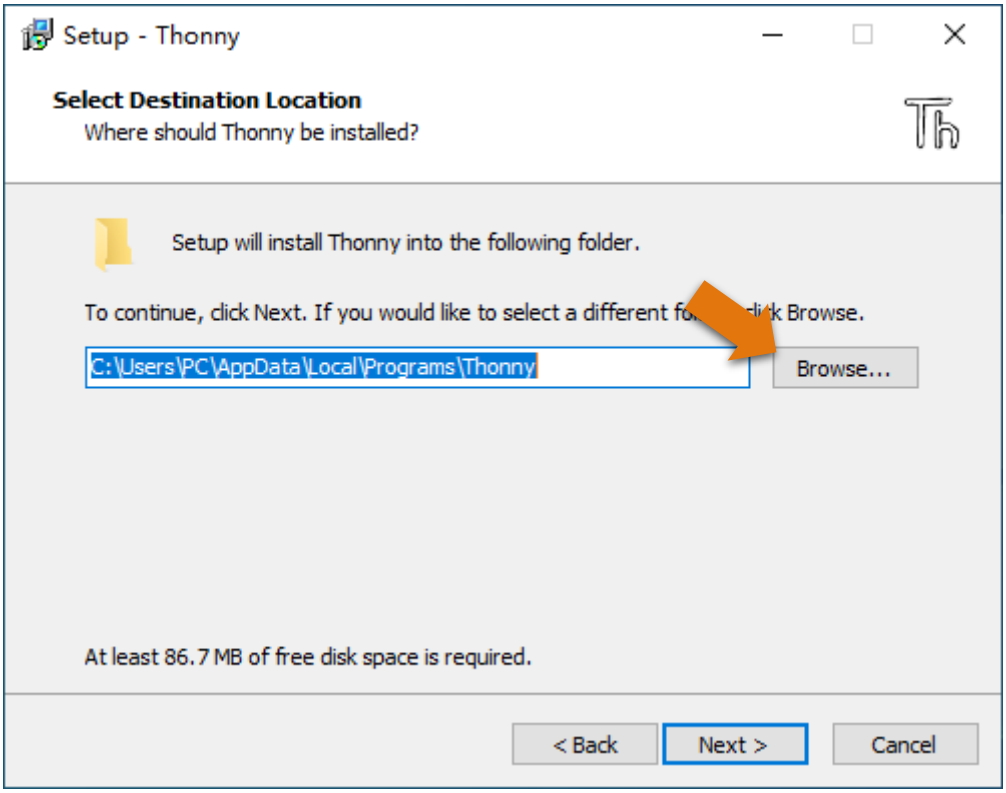


If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.

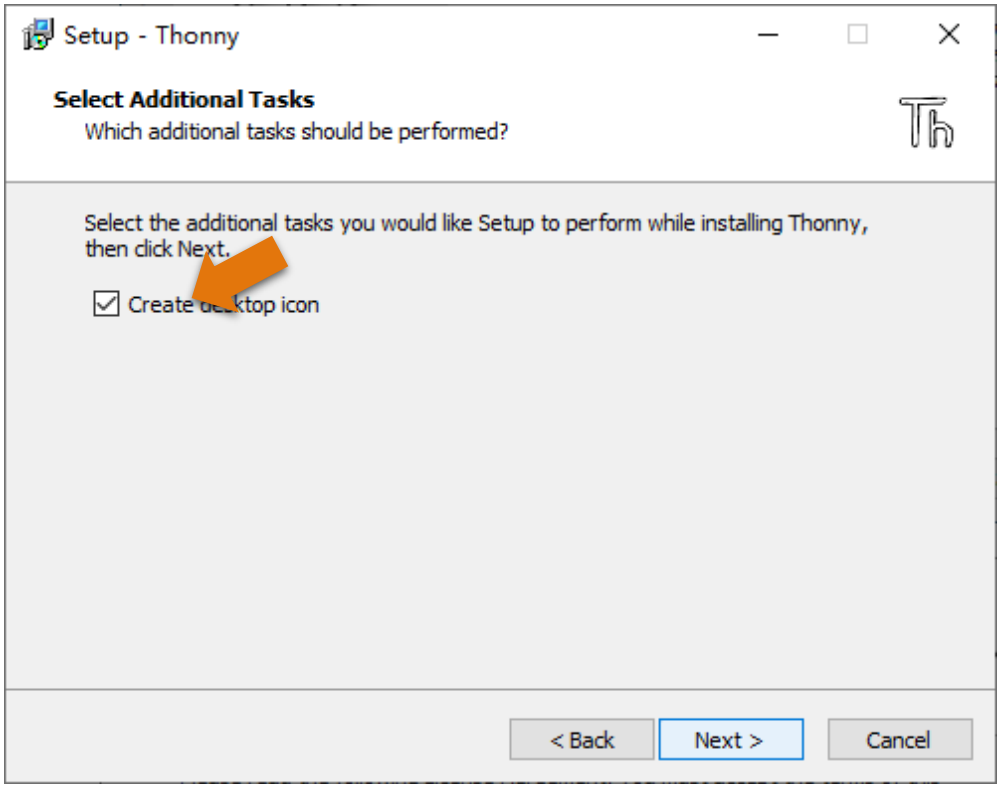


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

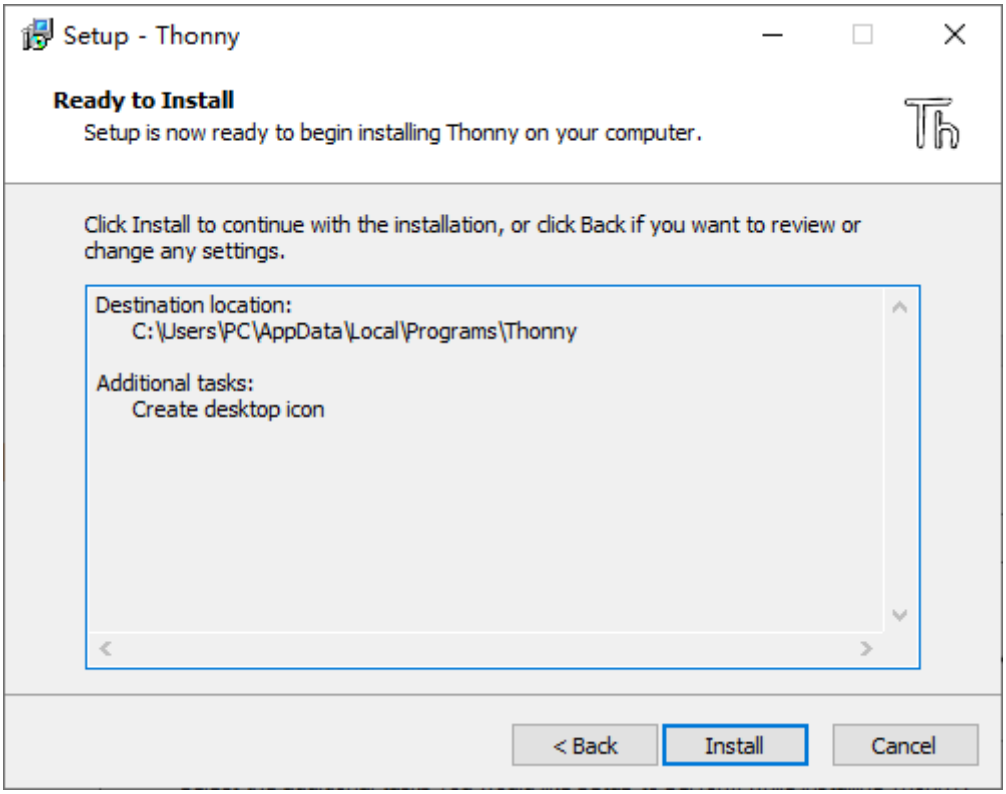
If you do not want to change it, just click "Next".



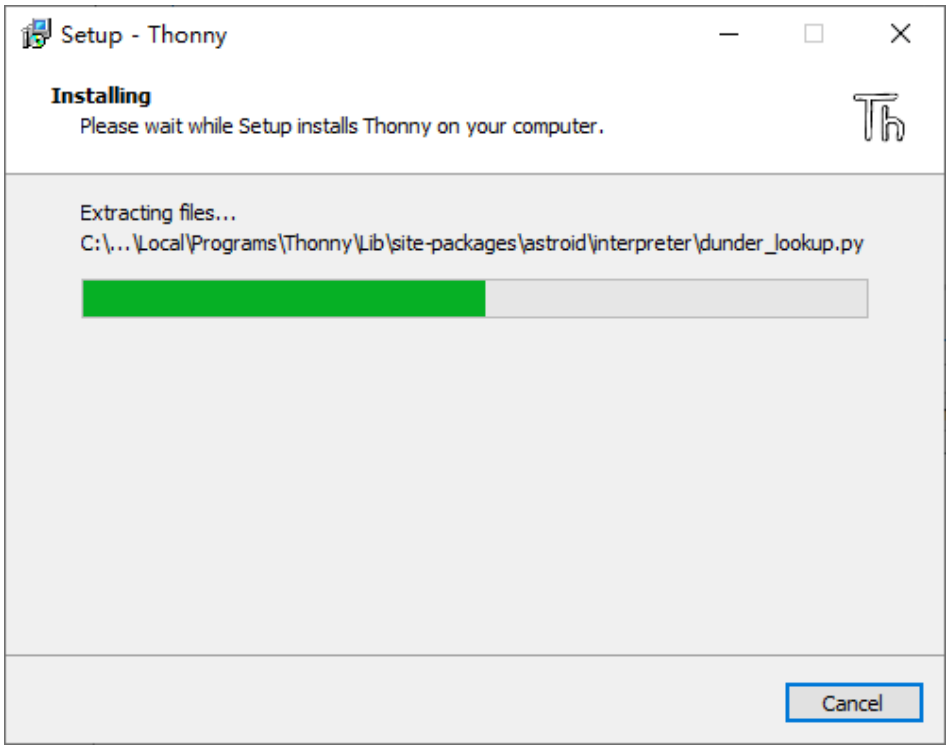
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



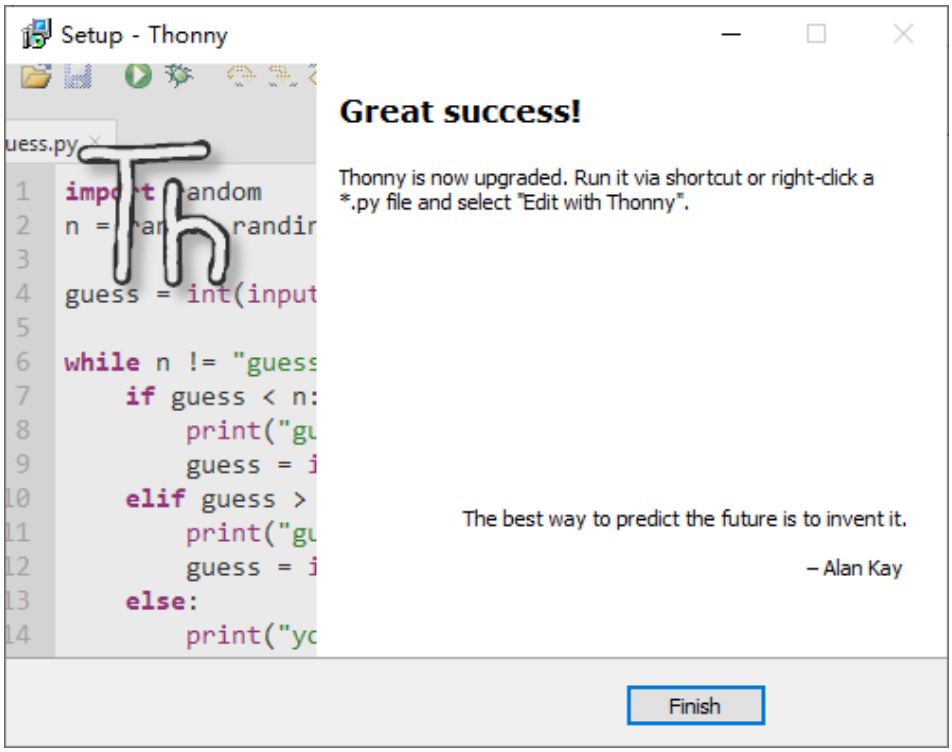
Click "install" to install the software.



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



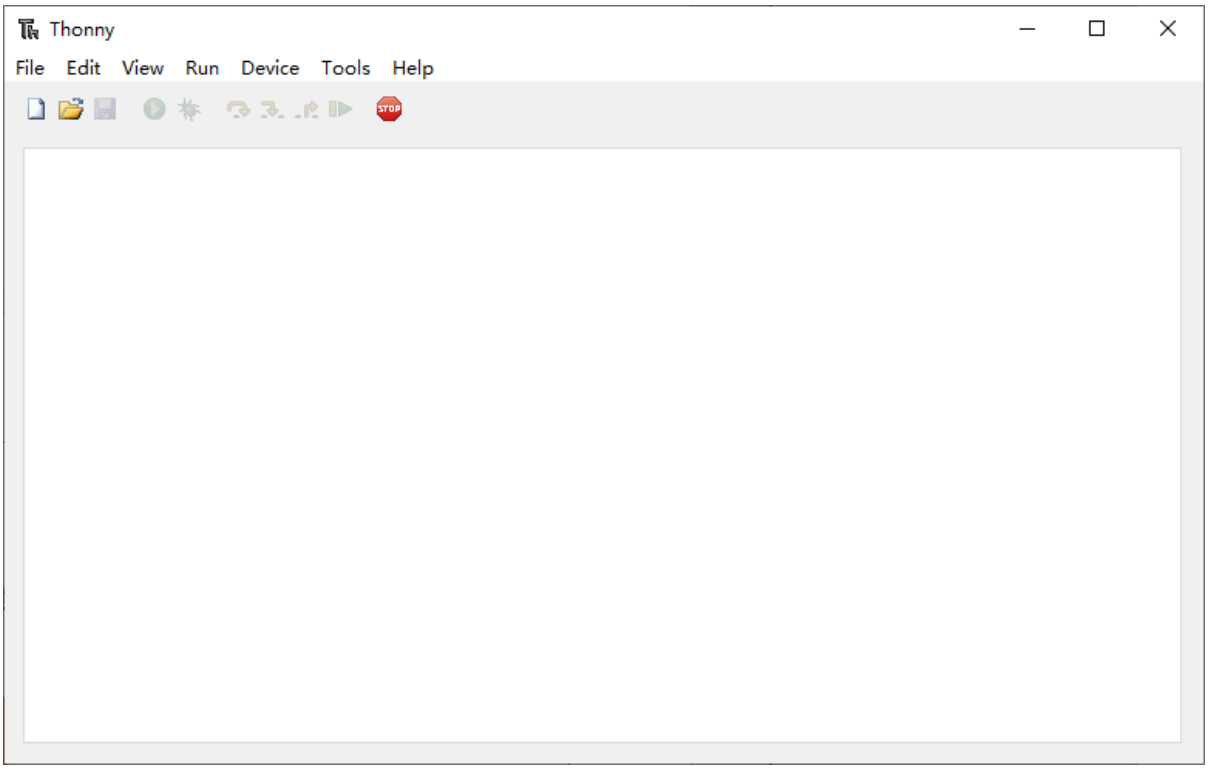
Once you see the interface as below, Thonny has been installed successfully.



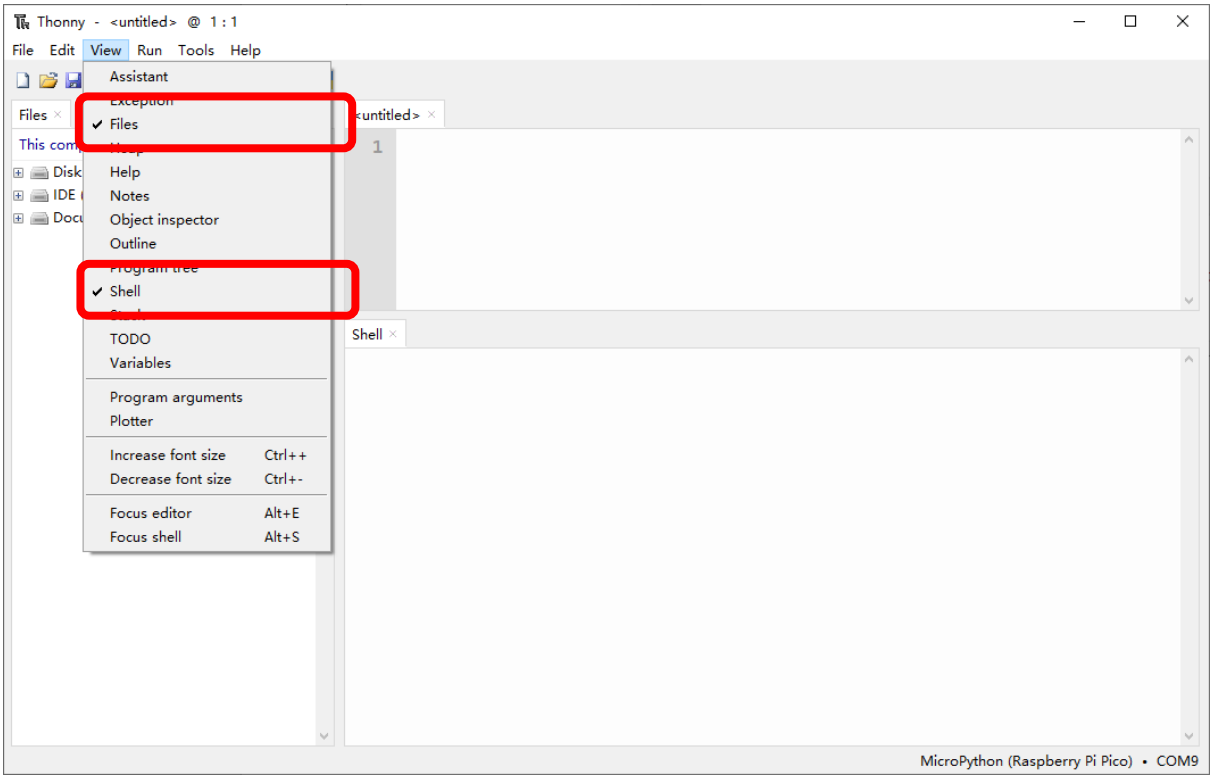
If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.。

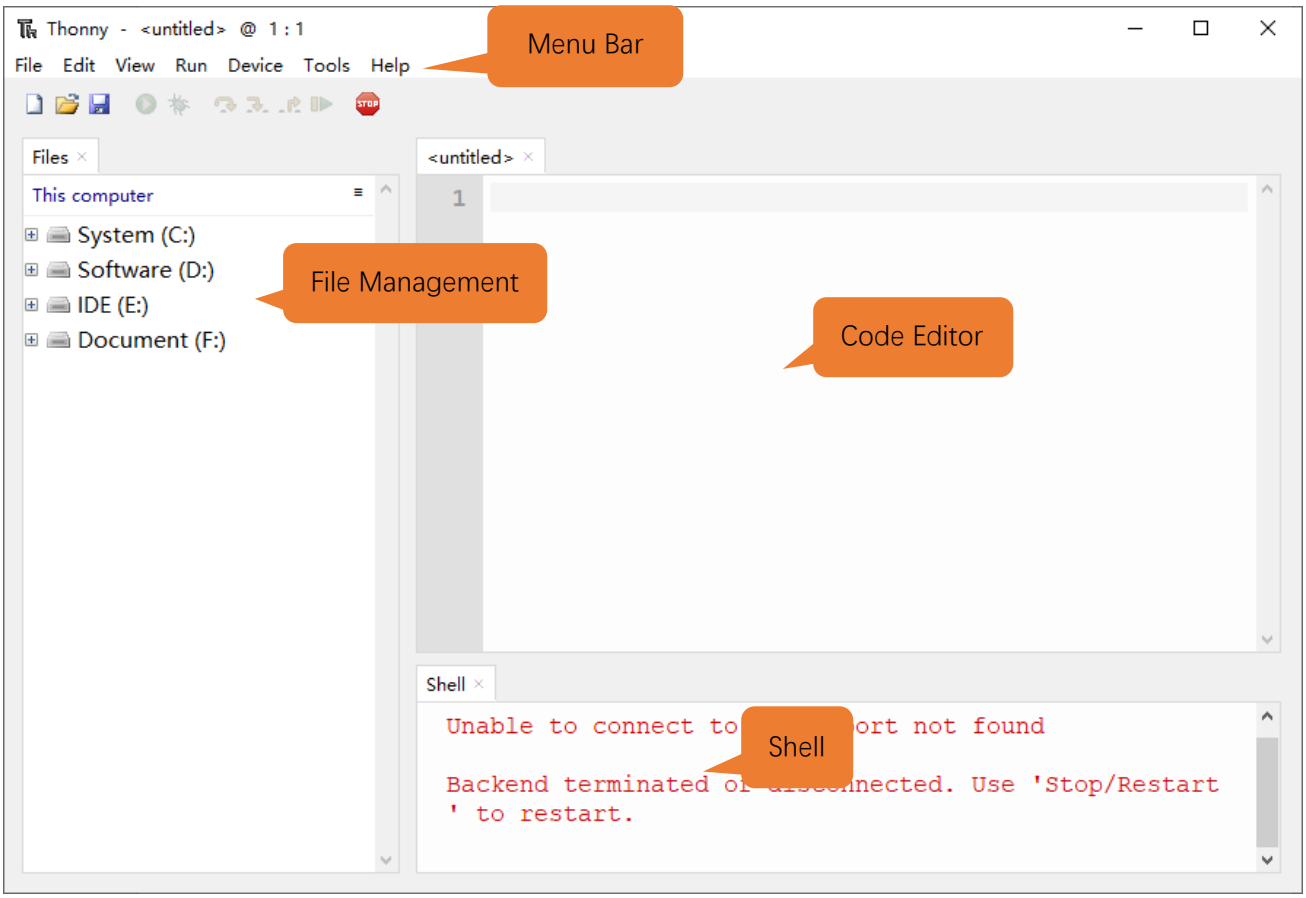
Basic Configuration of Thonny

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



Select "View"→ "Files" and "Shell".



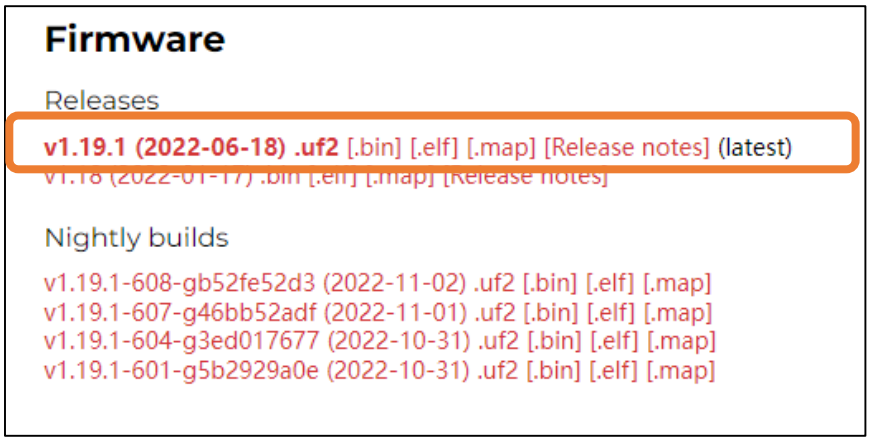

Thonny - <untitled> @ 1 : 1
File Edit View Run Device Tools Help
Menu Bar
Files
This computer
System (C:)
Software (D:)
IDE (E:)
Document (F:)
File Management
<untitled>
1
Code Editor
Shell
Unable to connect to
ort not found
Backend terminated
nected. Use 'Stop/Restart
' to restart.
Shell

Burning Micropython Firmware (Important)

To run Python programs on ESP32S3, we need to burn a firmware to ESP32-S3 first.

Downloading Micropython Firmware

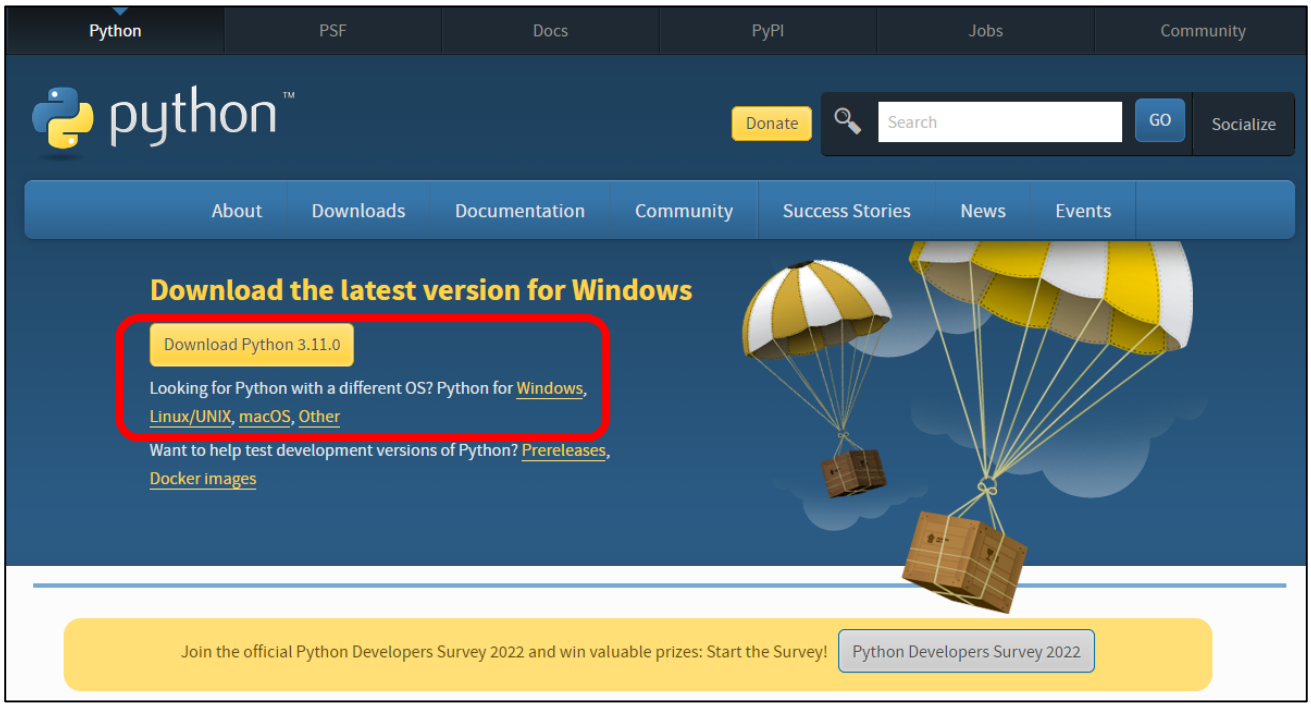
Official website of microPython: http://micropython.org/
Webpage listing firmware of microPython for ESP32S3: https://micropython.org/download/GENERIC_S3/



Firmware used in this tutorial is **GENERIC_S3-20220618-v1.19.1.bin**
This file is also provided in our data folder "**Freenove_Super_Starter_Kit_for_ESP32_S3 /Python/Python_Firmware**".。

Install python3

Before burning the firmware to ESP32S3, we need to ensure that Python 3 has been installed on the computer. If you have not already installed it, please install it first. Python Official download address is: https://www.python.org/downloads/

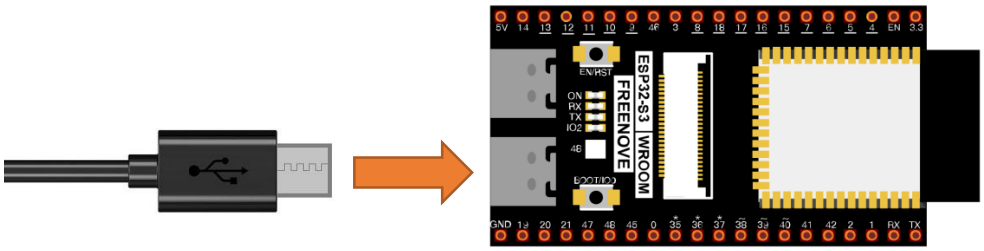


Please follow the official instructions to download and install.

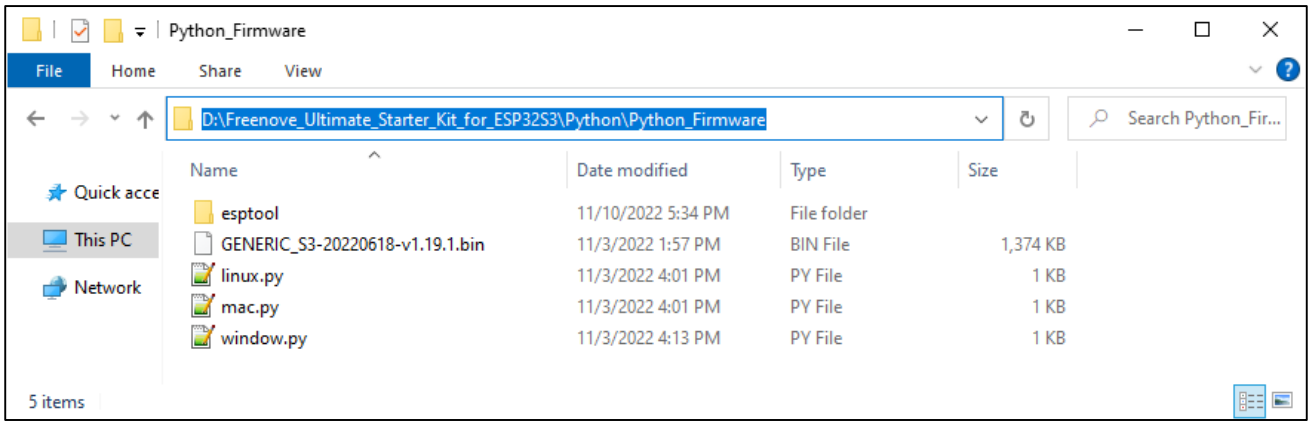
Burning a Micropython Firmware

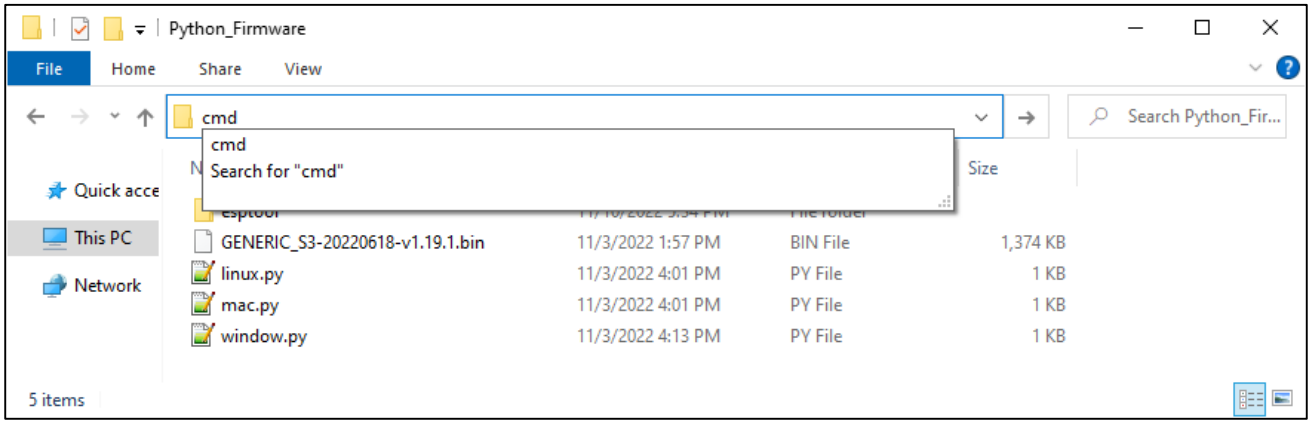
Window

Connect your computer and ESP32-S3 with a USB cable.



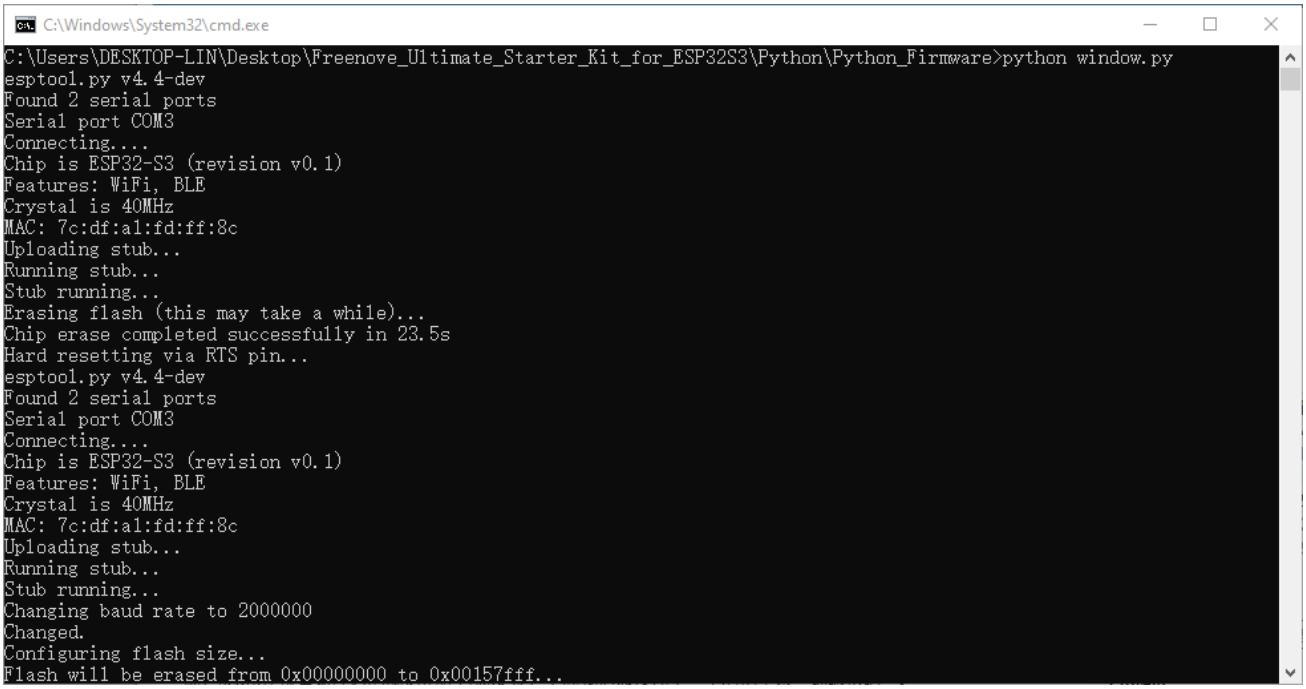
Open **Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Firmware**
Enter cmd on path bar then press Enter.



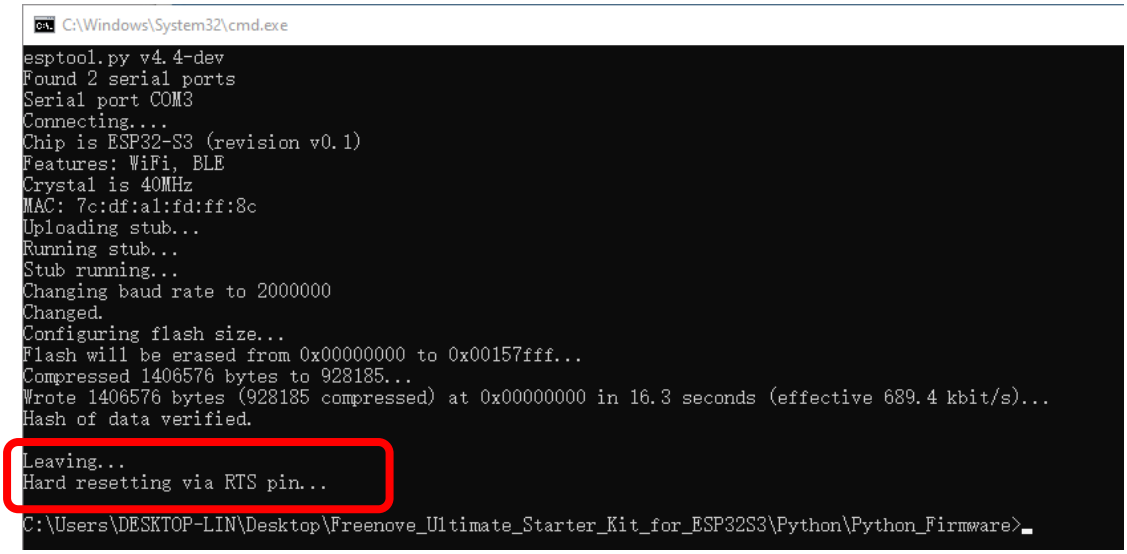


Here my python3 version is 3.8.1.

Enter “python window.py”. Micropython firmware will be automatically burned to ESP32S3.

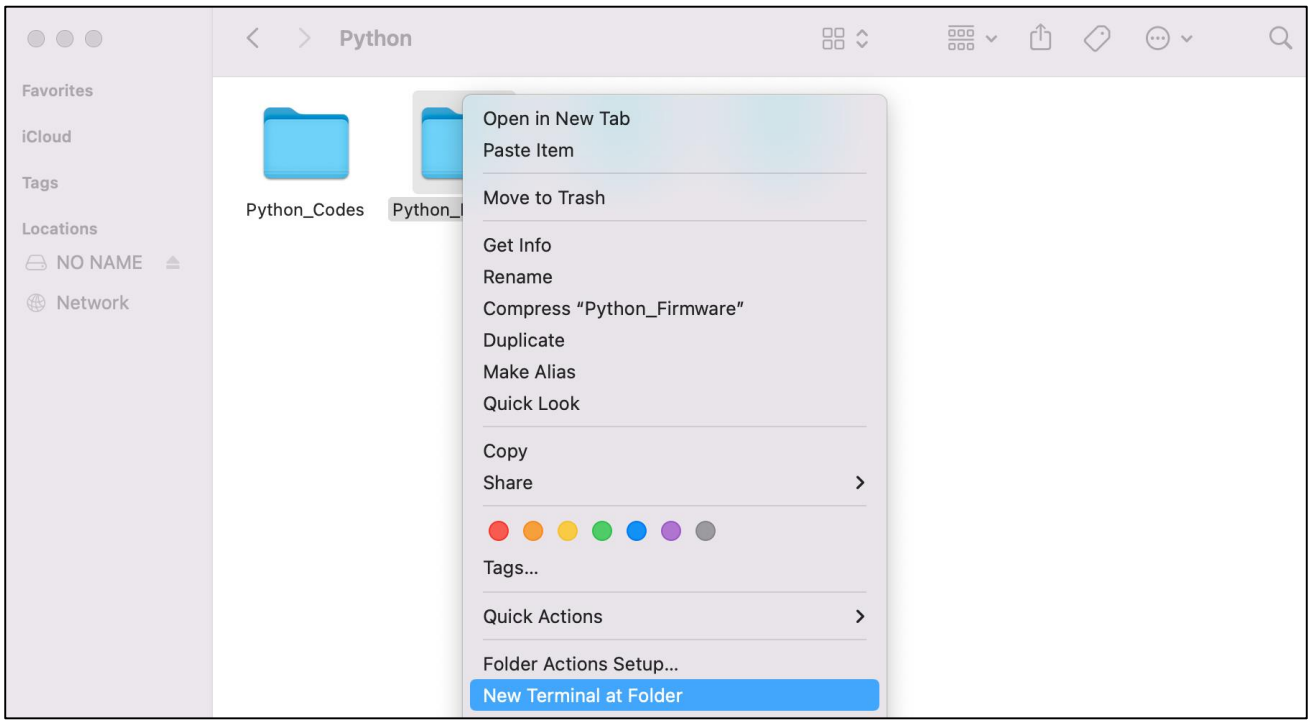

C:\Windows\System32\cmd.exe
C:\Users\DESKTOP-LIN\Desktop\Freenove_Ultimate_Starter_Kit_for_ESP32S3\Python\Python_Firmware>python window.py
esptool.py v4.4-dev
Found 2 serial ports
Serial port COM3
Connecting....
Chip is ESP32-S3 (revision v0.1)
Features: WiFi, BLE
Crystal is 40MHz
MAC: 7c:df:a1:fd:ff:8c
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 23.5s
Hard resetting via RTS pin...
esptool.py v4.4-dev
Found 2 serial ports
Serial port COM3
Connecting....
Chip is ESP32-S3 (revision v0.1)
Features: WiFi, BLE
Crystal is 40MHz
MAC: 7c:df:a1:fd:ff:8c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 2000000
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x00157fff...

As shown in the figure below after completion.

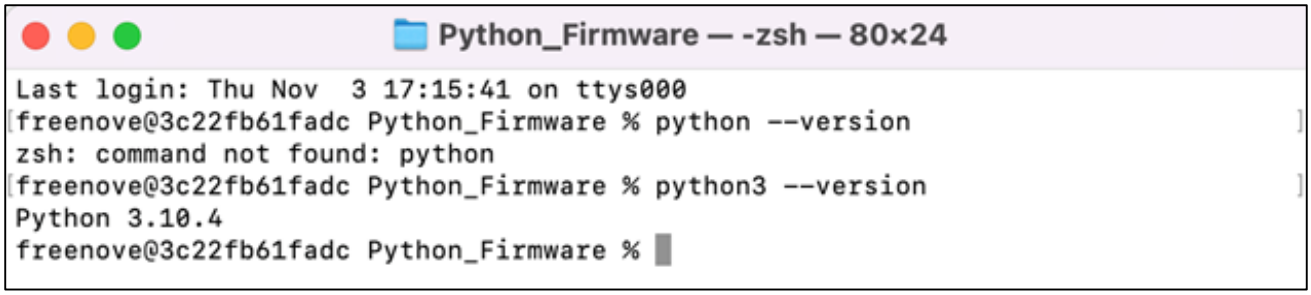


Mac

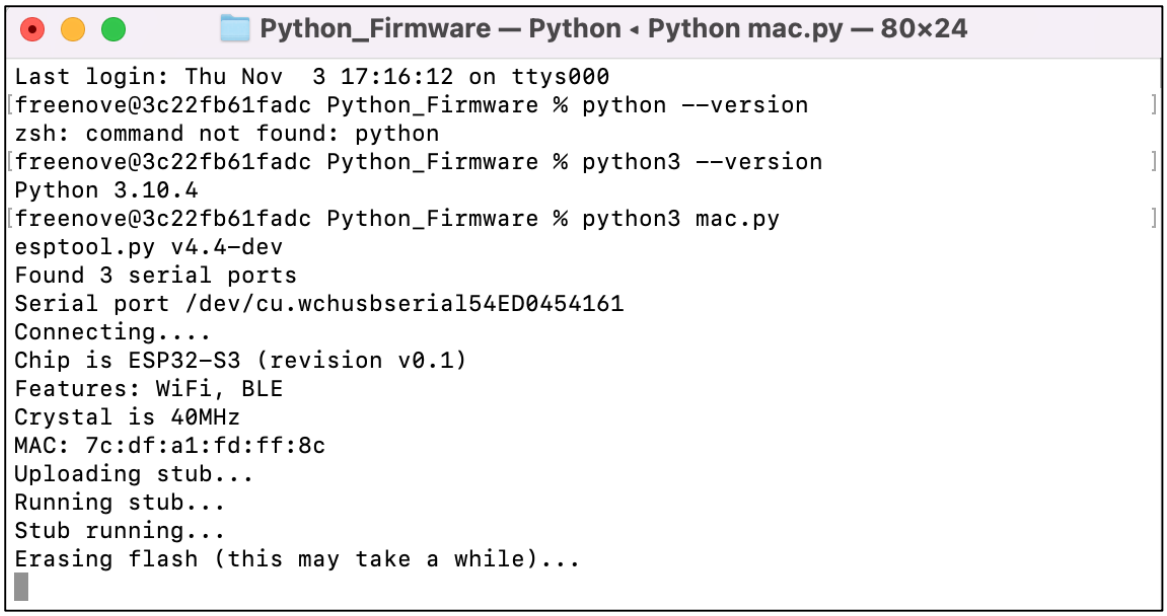
Open **Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Firmware**. Right- click and select New Terminal at Folder.



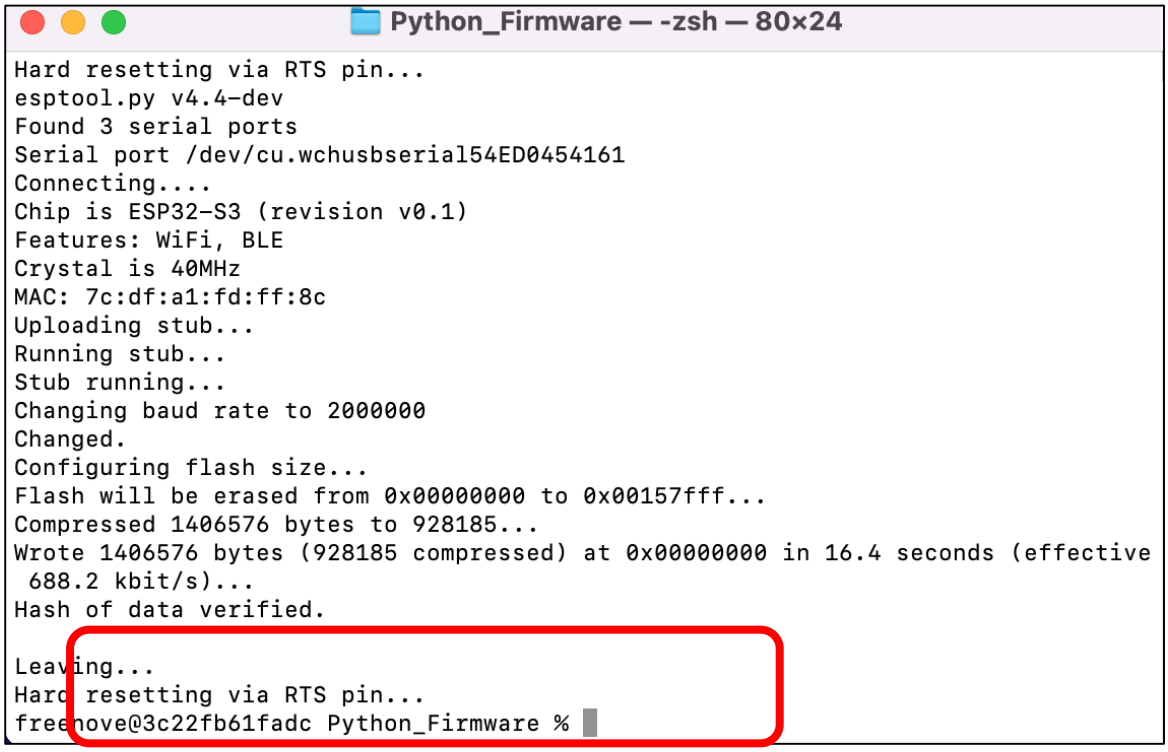
Here, my python3 version is 3.10.4



Enter "python3 mac. py" in the terminal, press Enter, and wait for the code to automatically burn the microython firmware into ESP32S3.



After completion, it is shown below.

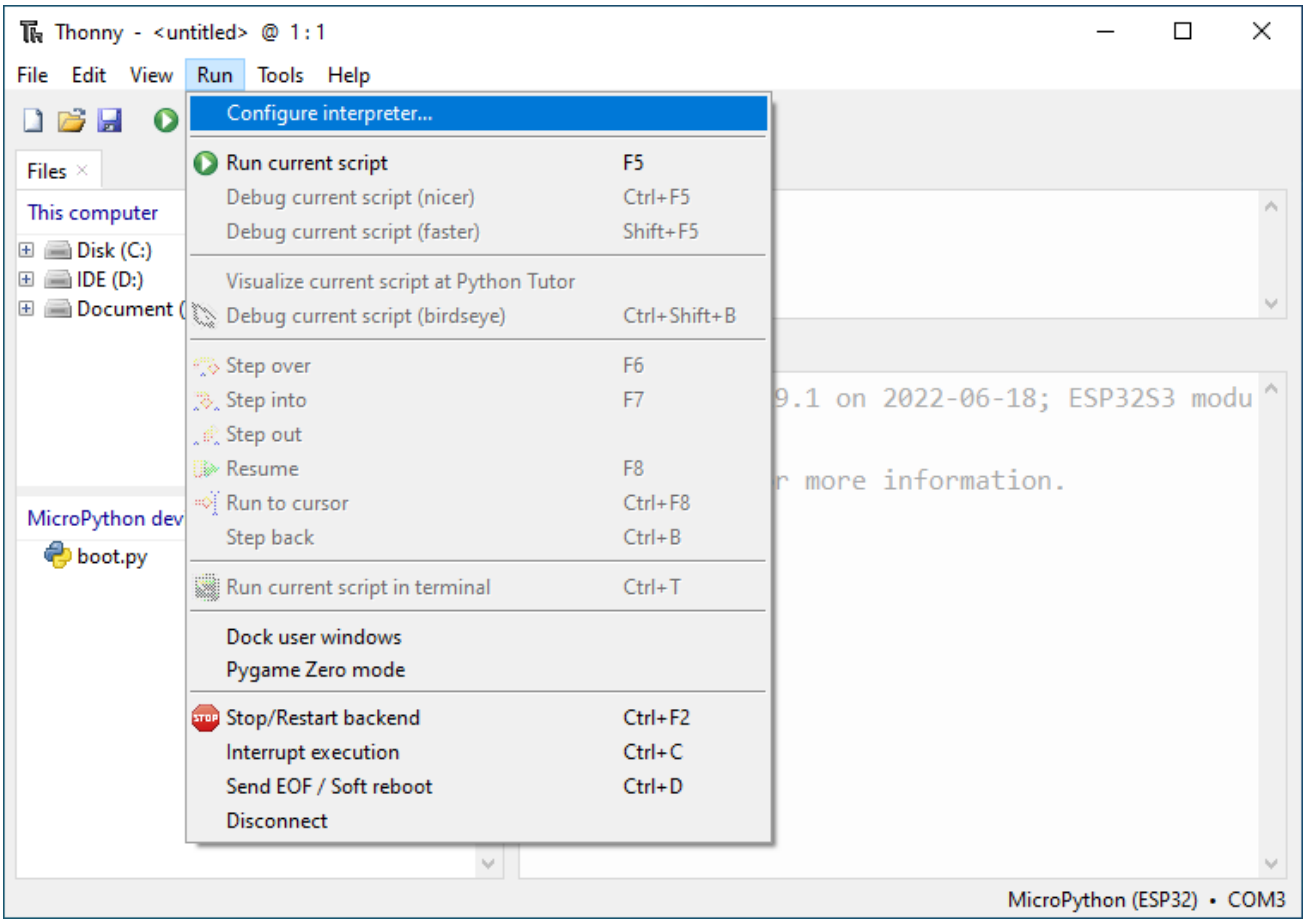


Note: The operation of the Linux system is similar to that of the Mac system. Please refer to the Mac system.

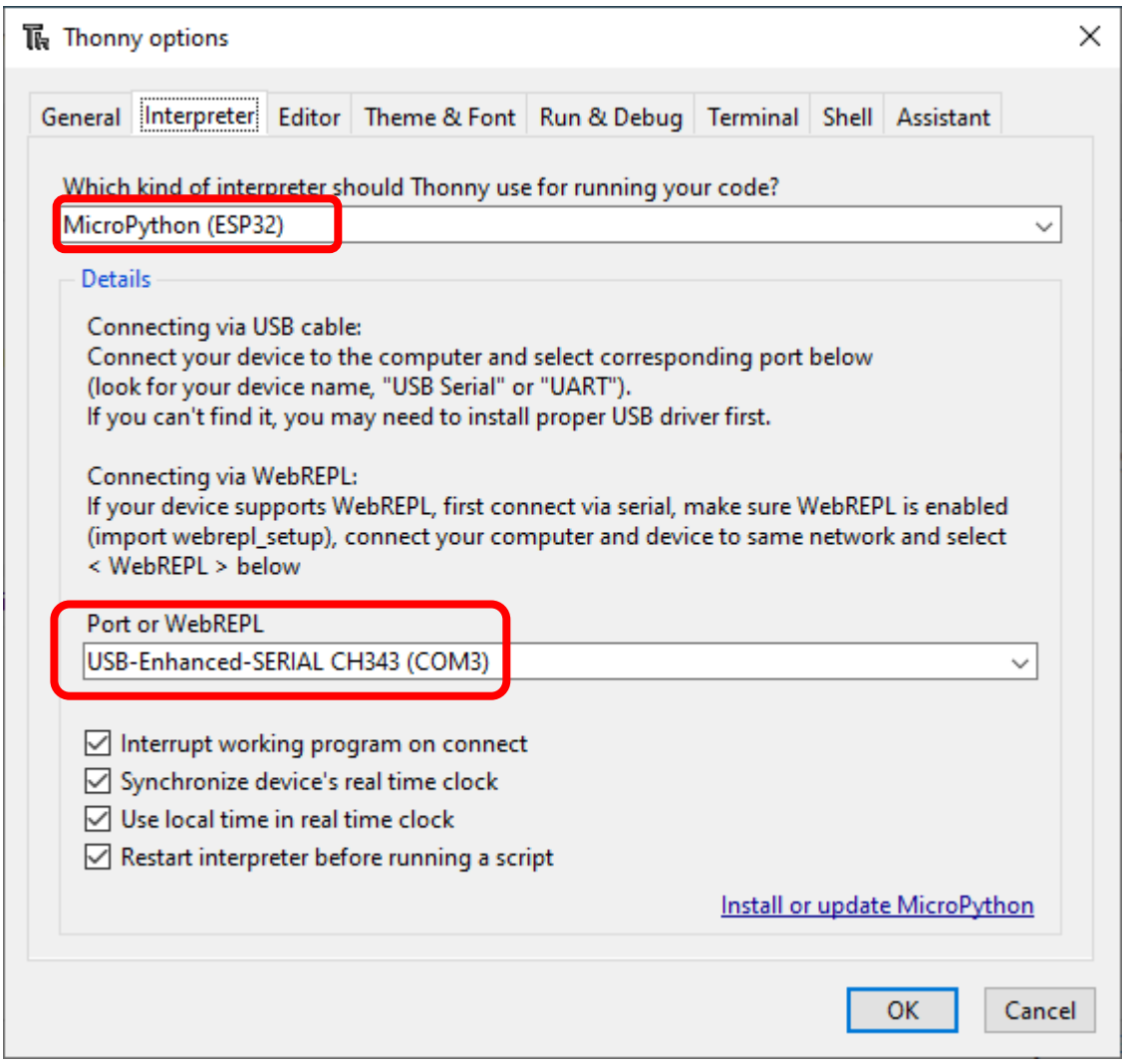
Testing codes (Important)

Testing Shell Command

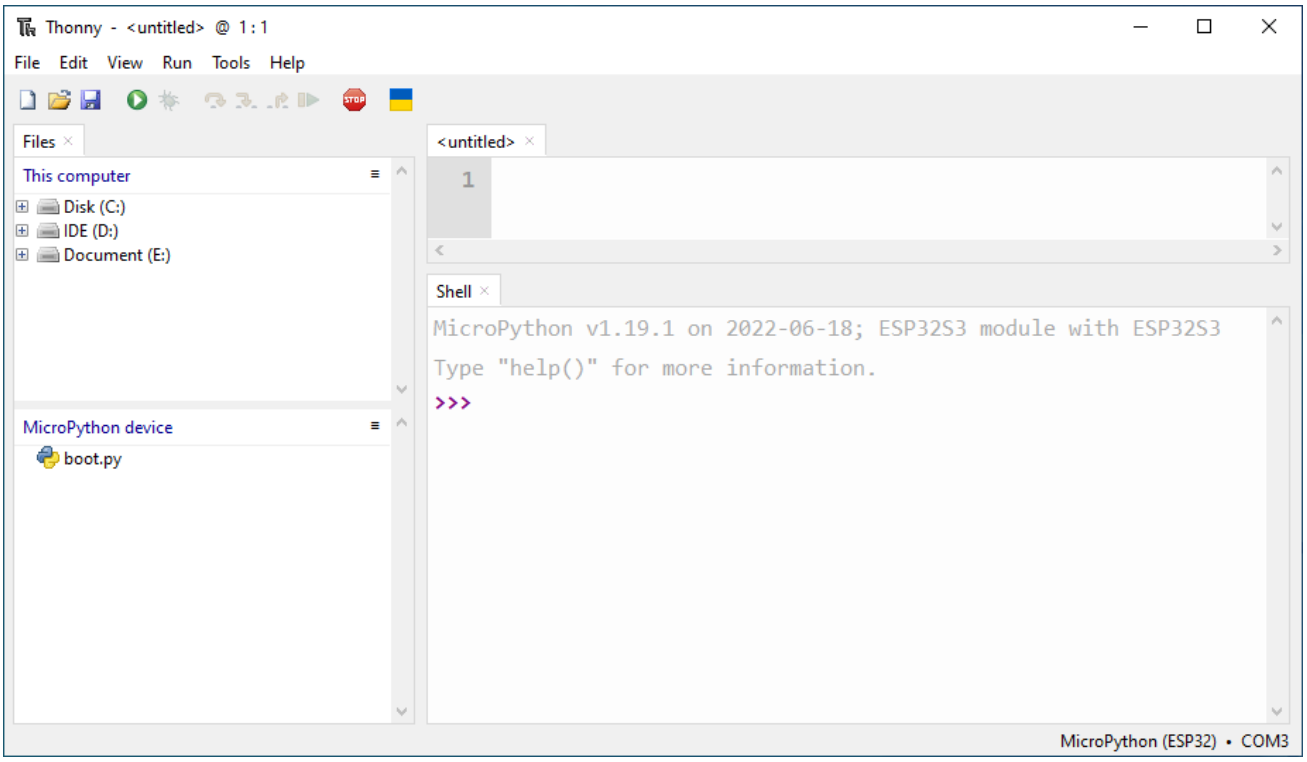
Make sure that the ESP 32S3 has burned the firmware and is connected to the computer through the data cable. Run Thonny. Click Run and select Configure interpreter.



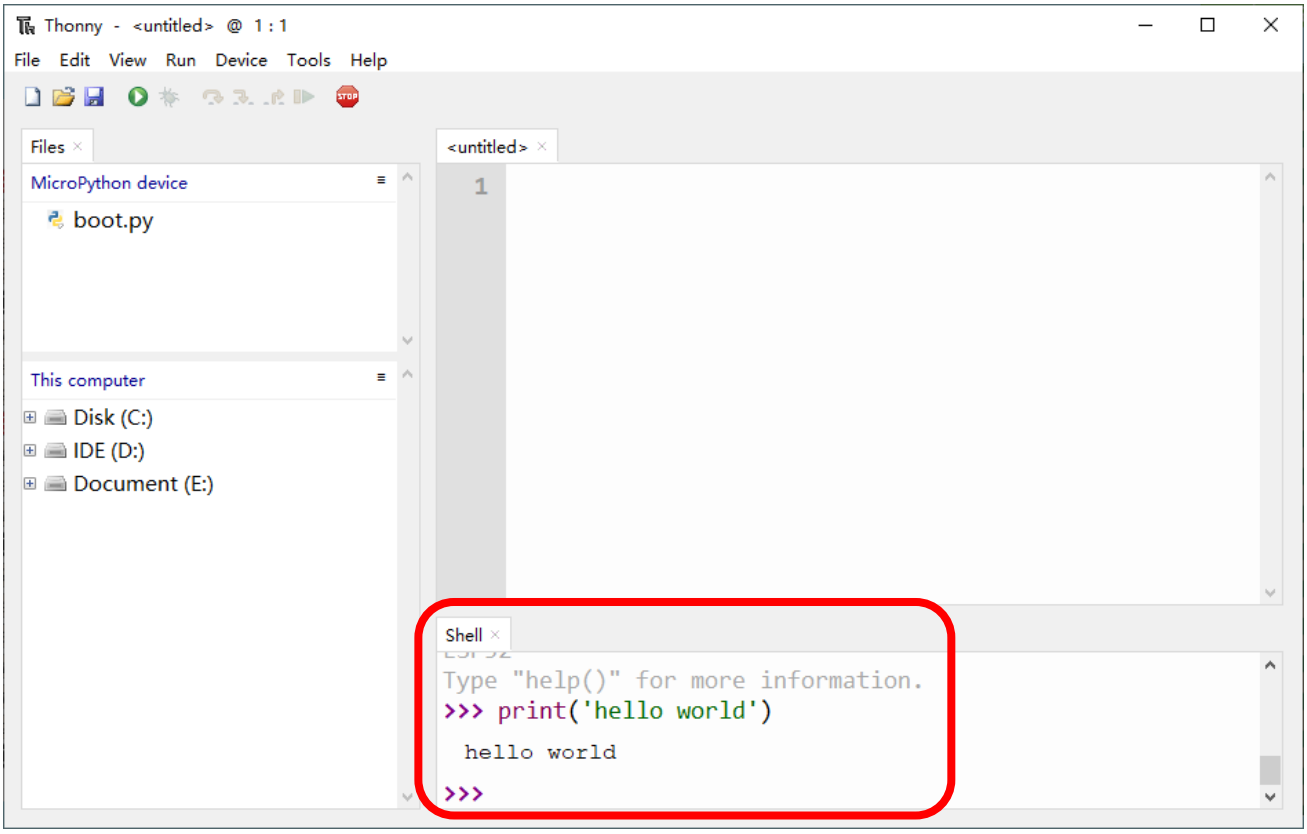
Please configure according to the following figure. Note that the port numbers of USB Enhanced SERIAL may be different for different systems. Please select according to the actual situation. After configuration, click OK.



After configuration, every time you open Thonny, it will communicate with ESP32S3. The interface is shown below.



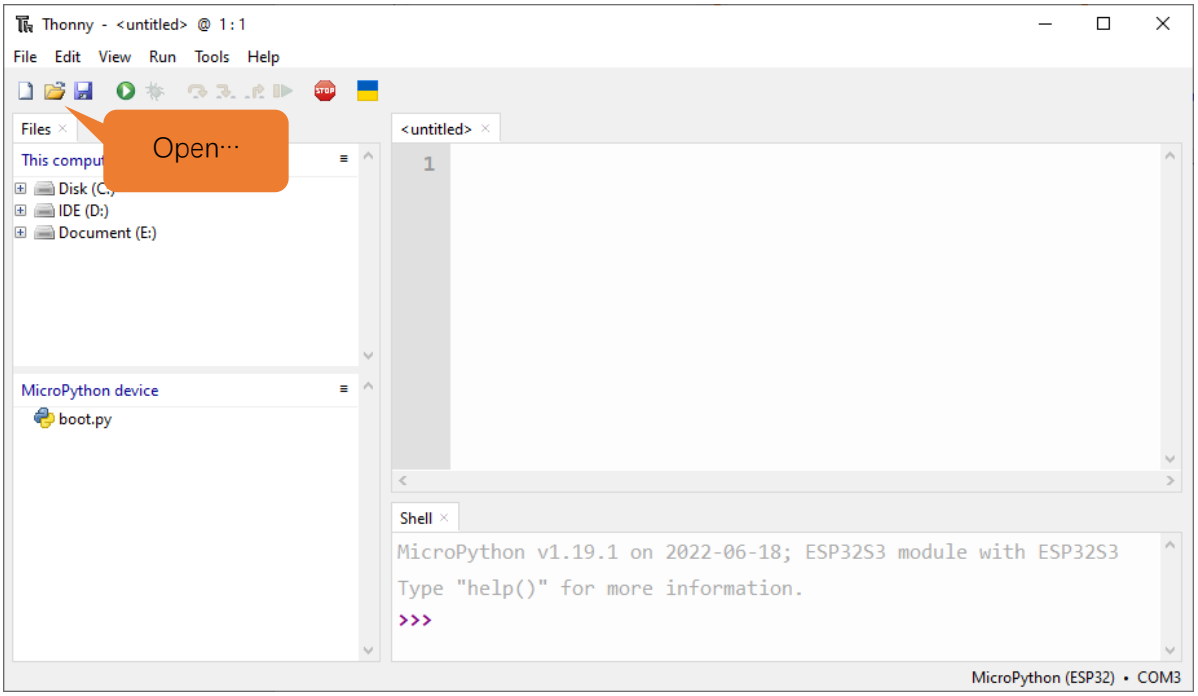
Enter "print('hello world')" in "Shell" and press Enter.



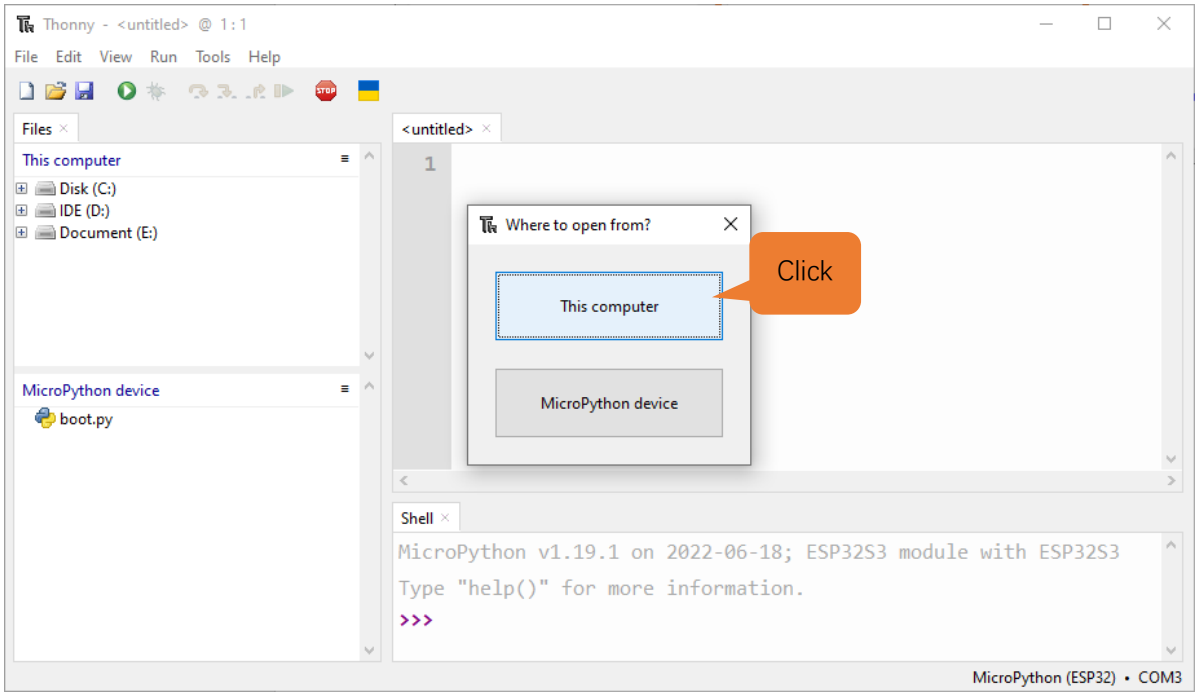
Running Online

ESP32-S3 needs to be connected to a computer when it is run online. Users can use Thonny to writer and debug programs.

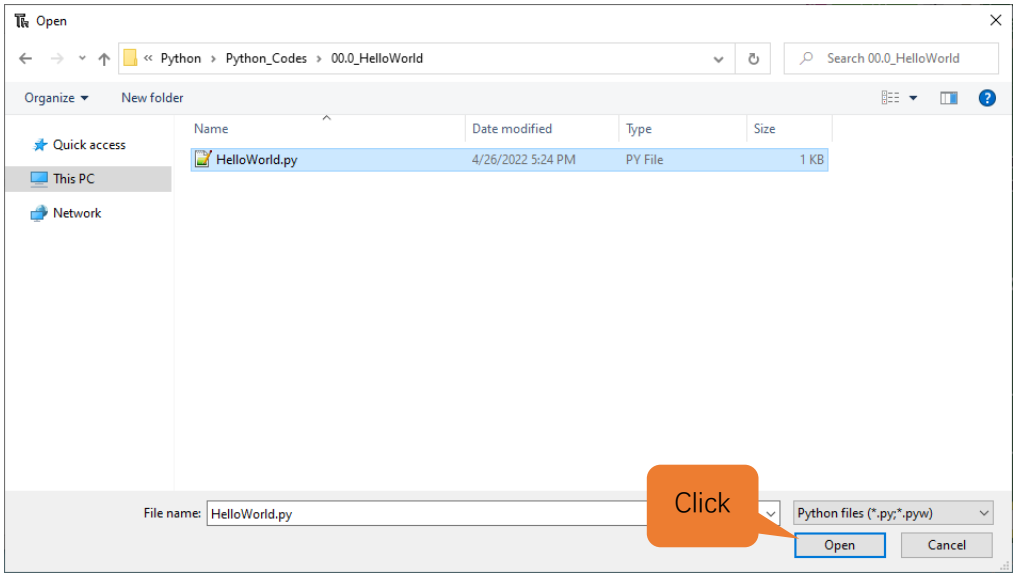
1. Open Thonny and click “Open···”.



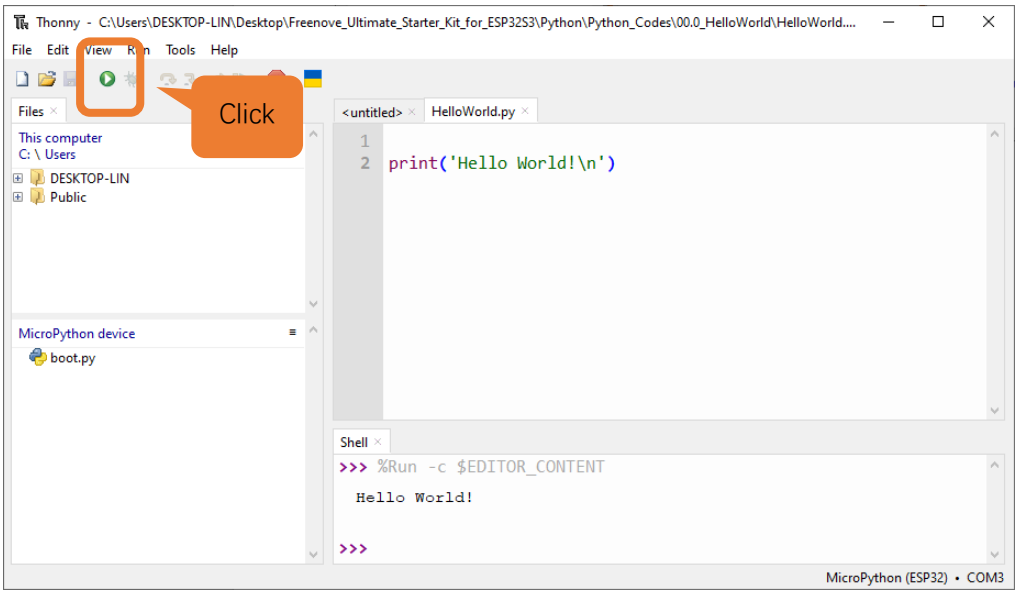
2. On the newly pop-up window, click “This computer”.



In the new dialog box, select "**HelloWorld.py**" in "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes/00.0_HelloWorld**" folder.



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

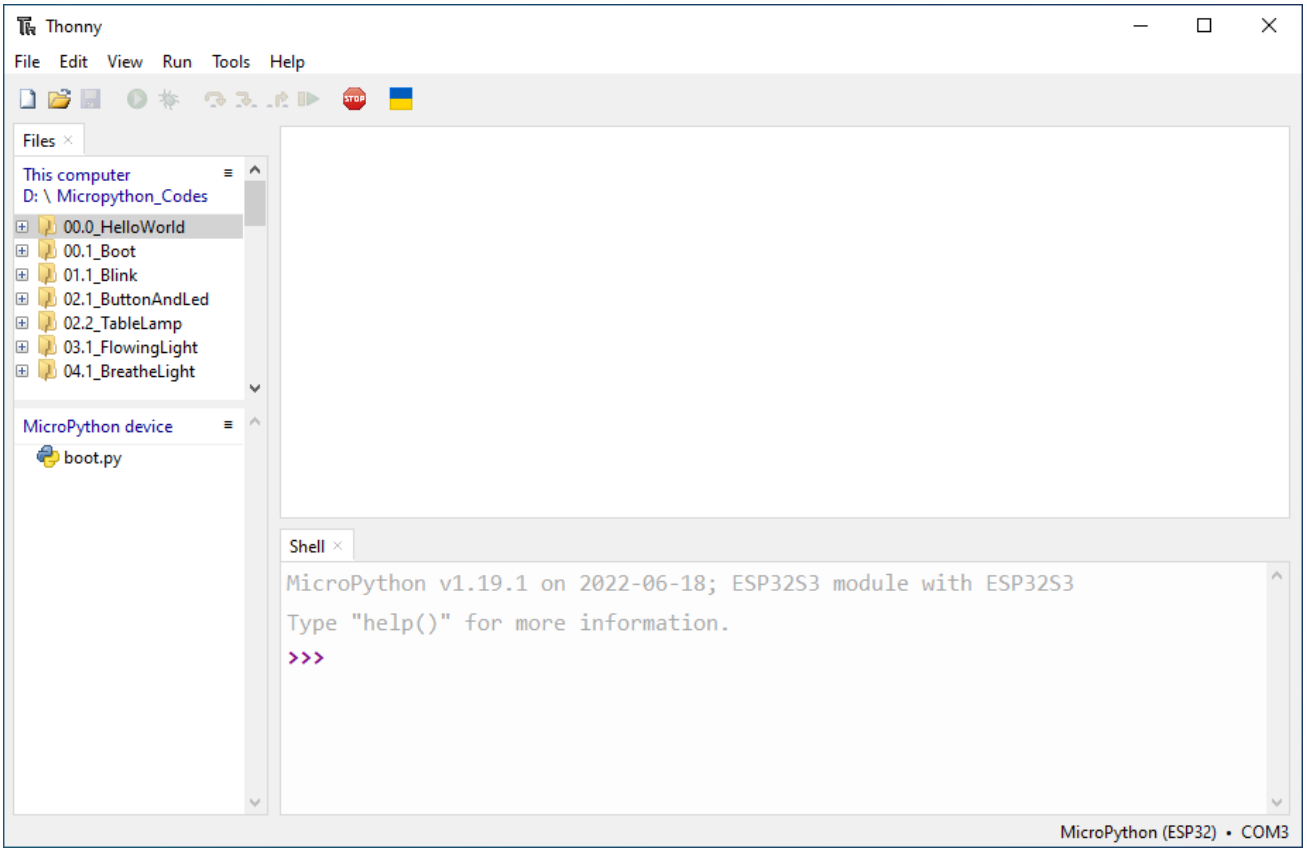


Note: When running online, if you press the reset key of ESP32S3, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following Running Offline.

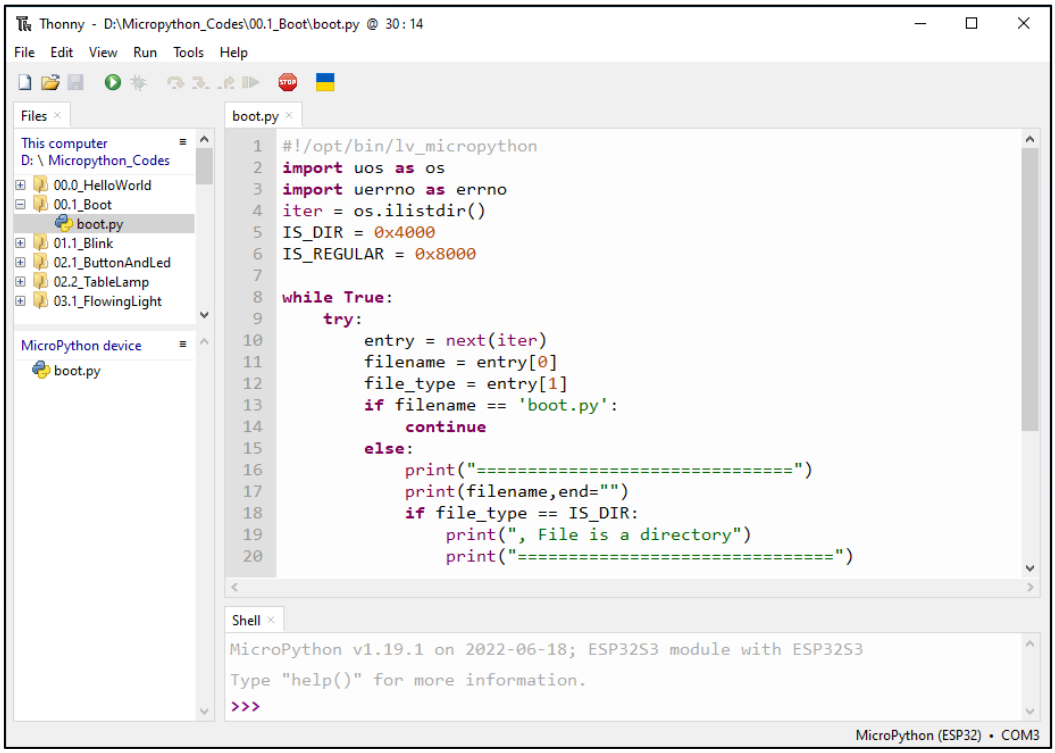
Running Offline（Importance）

After ESP32-S3 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-S3 execute user's programs after resetting, we need to add a guiding program in boot.py to execute user's code.

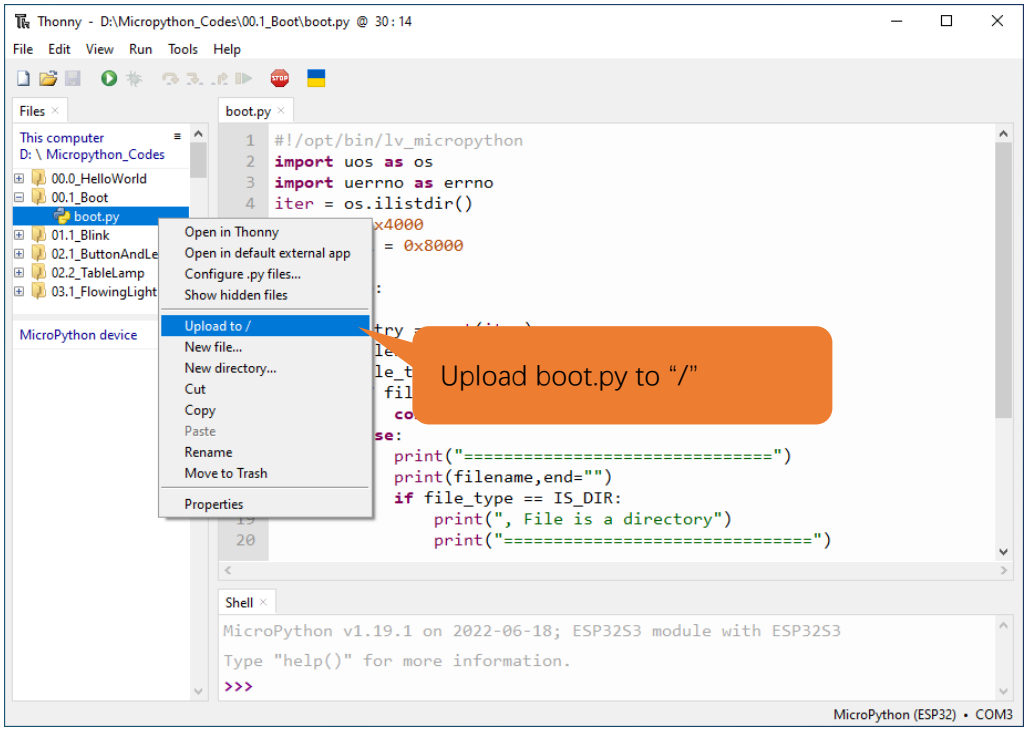
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**". Open "Thonny"。



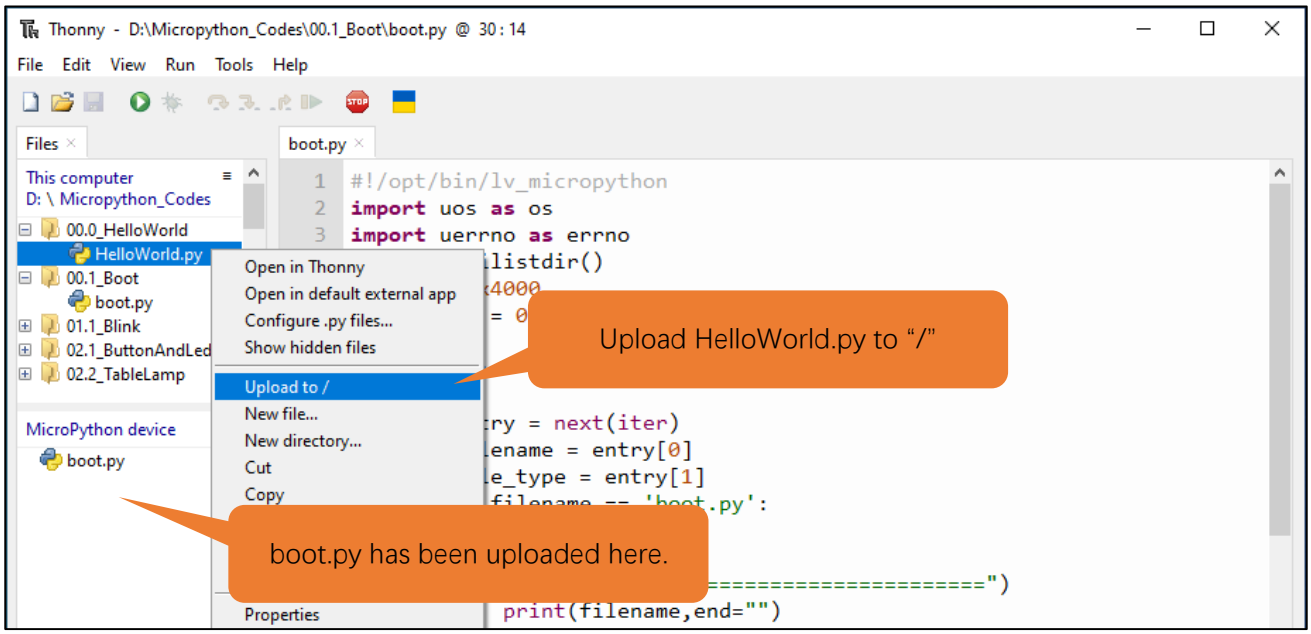
Expand "00.1_Boot" in the "Micropython_Codes" in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in "MicroPython device" to run offline.



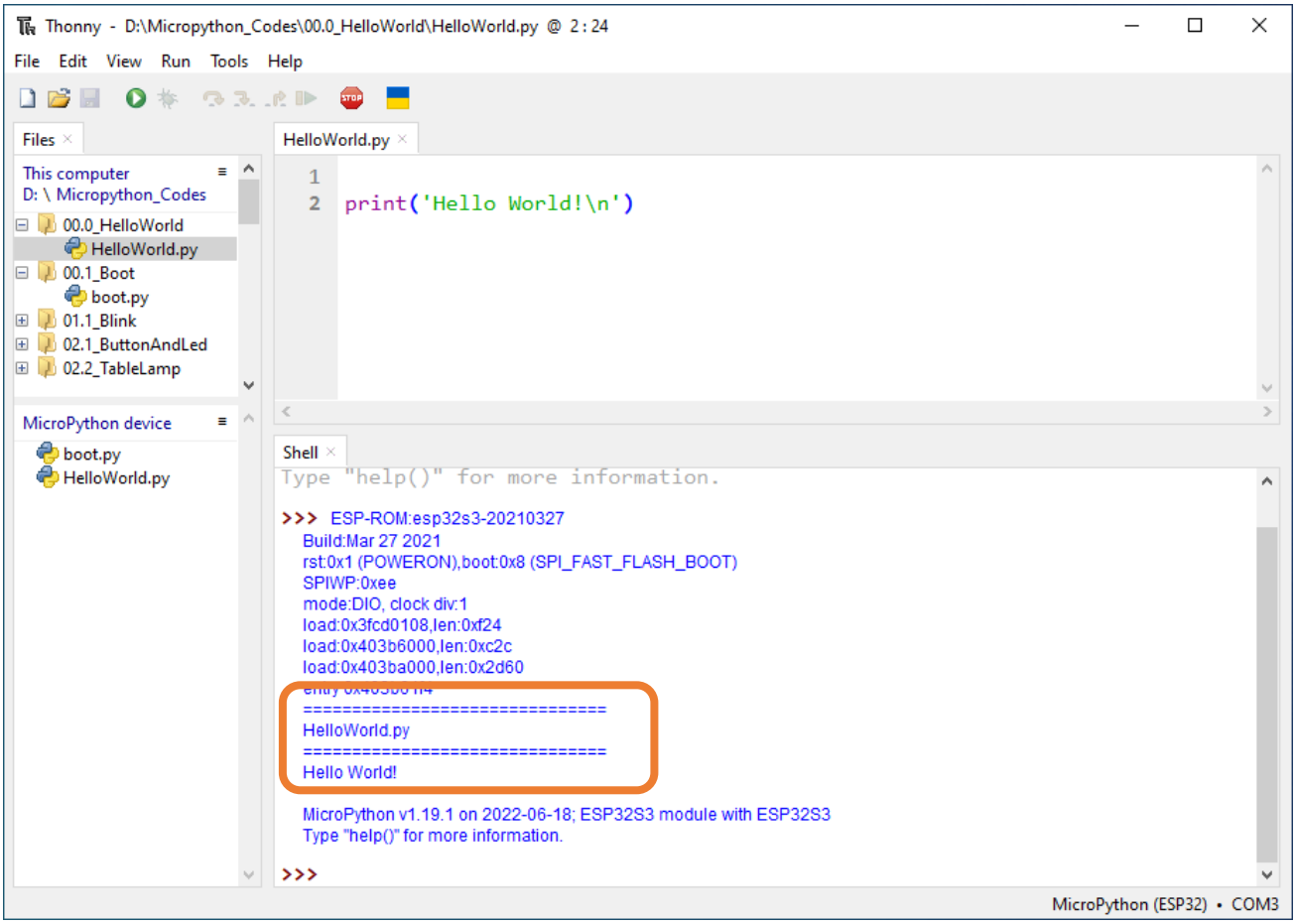
If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to "MicroPython device" and press ESP32S3's reset key. Here we use programs 00.0 and 00.1 as examples. Select "boot.py", right-click to select "Upload to /".



Similarly, upload "HelloWorld.py" to "MicroPython device".



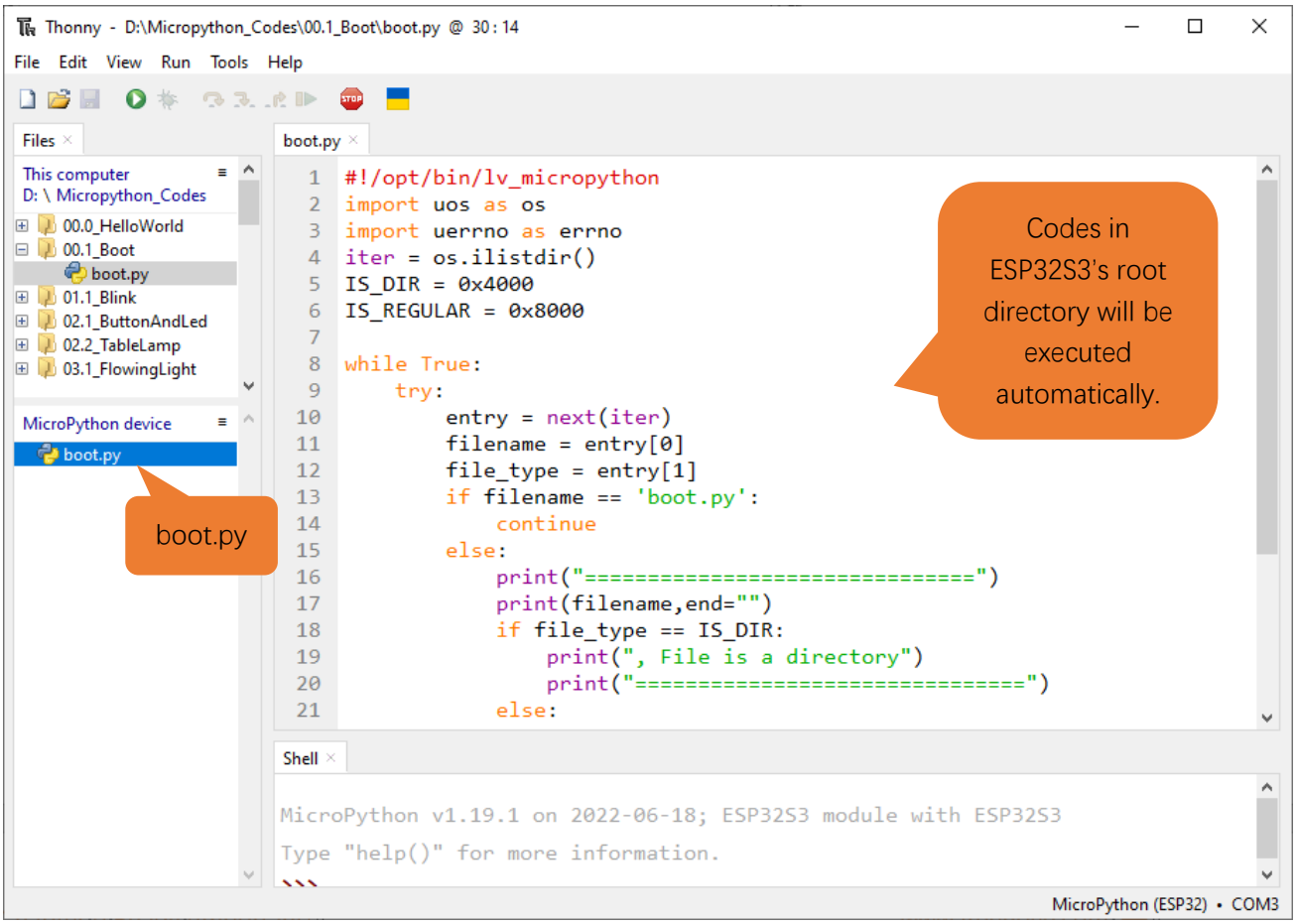
Press the reset key and in the box of the illustration below, you can see the code is executed.



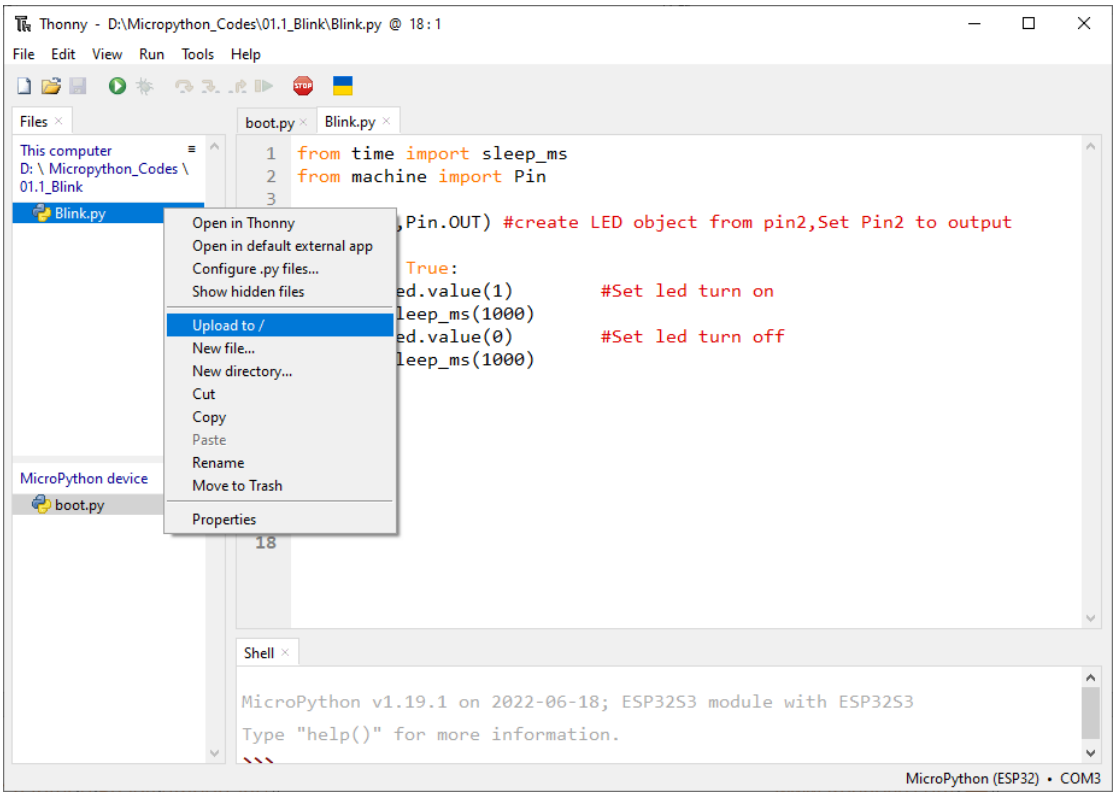
Thonny Common Operation

Uploading Code to ESP32S3

Each time when ESP32-S3 restarts, if there is a “boot.py” in the root directory, it will execute this code first.

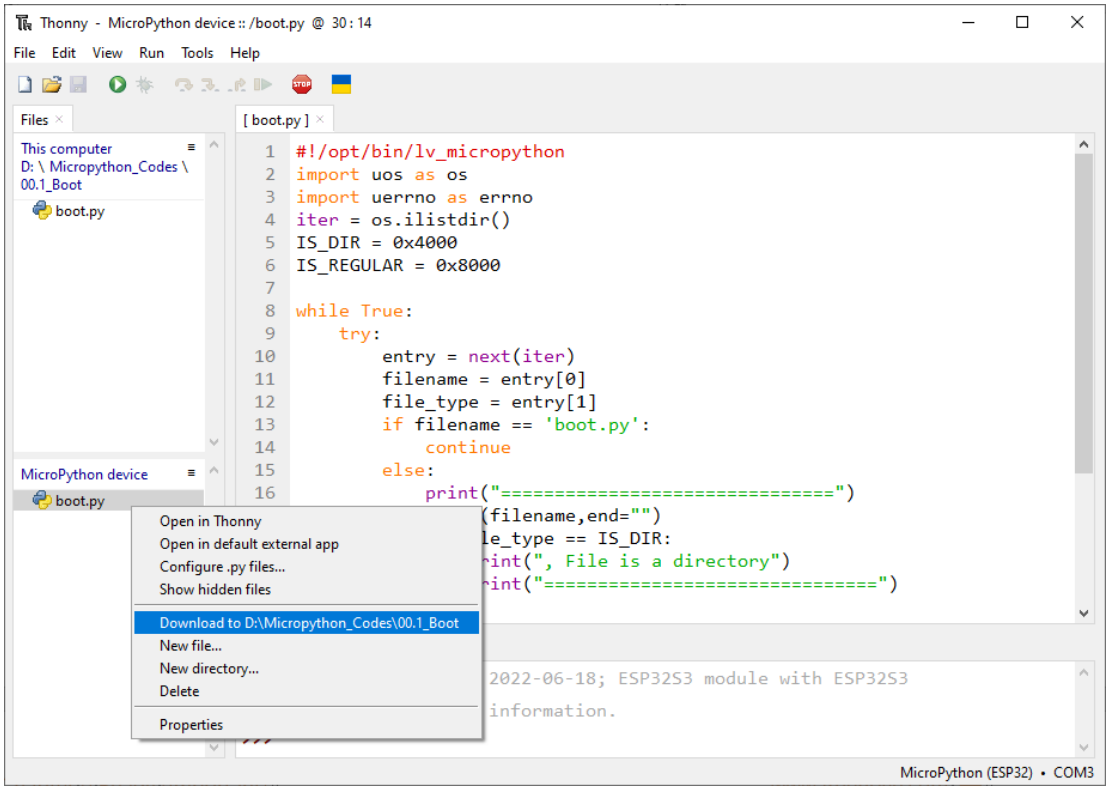


Select “Blink.py” in “01.1_Blink”, right-click your mouse and select “Upload to /” to upload code to ESP32S3’s root directory.



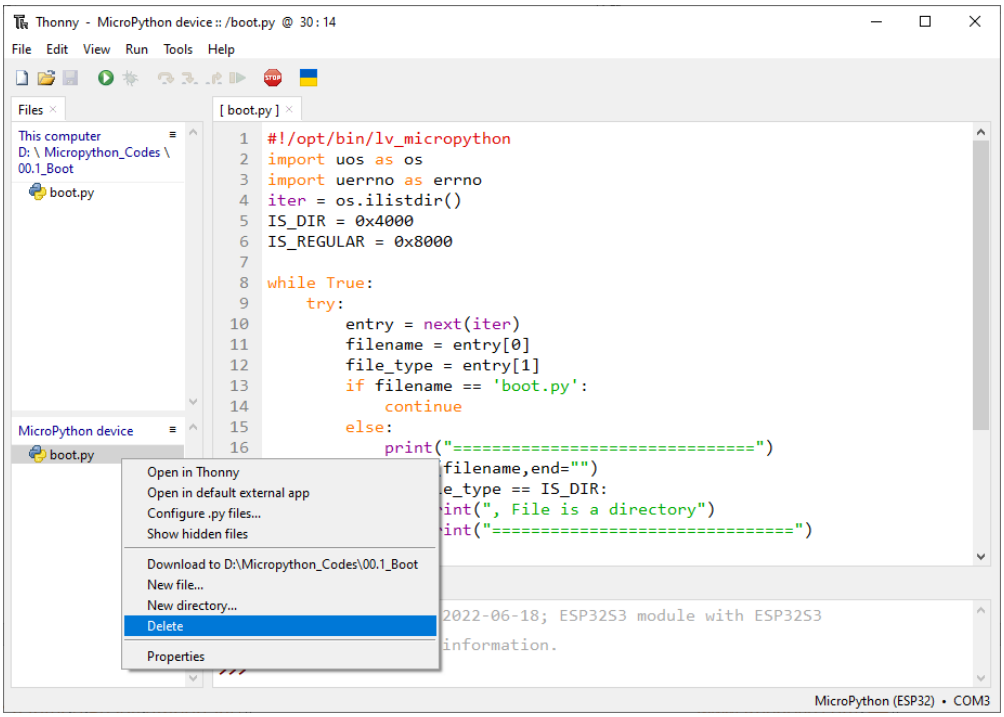
Downloading Code to Computer

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



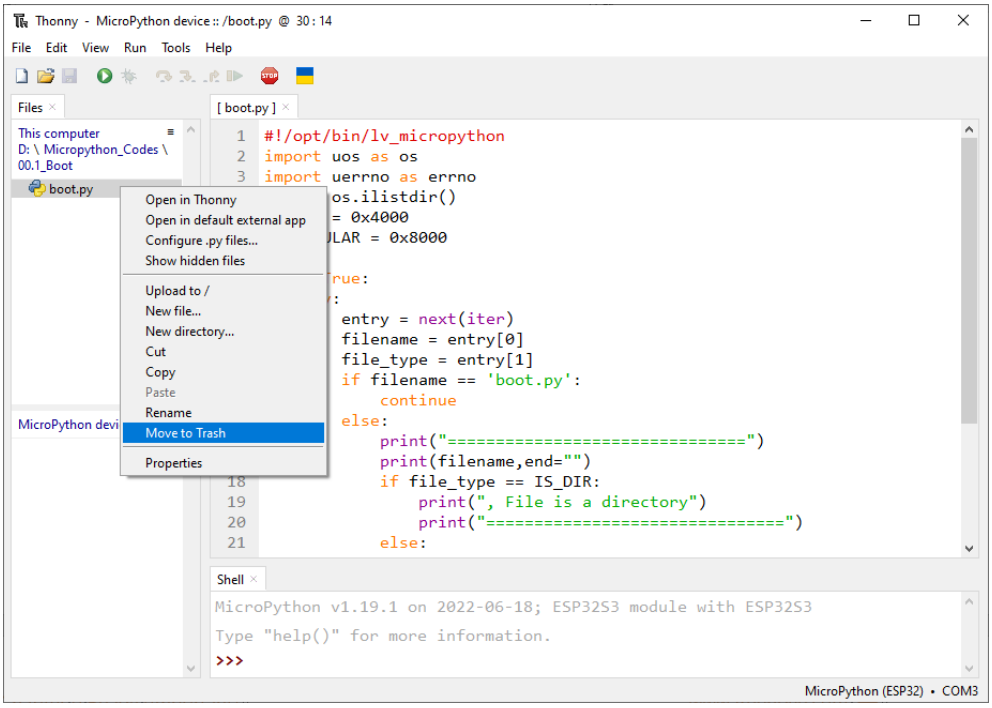
Deleting Files from ESP32S3's Root Directory

Select "boot.py" in "MicroPython device", right-click it and select "Delete" to delete "boot.py" from ESP32S3's root directory.



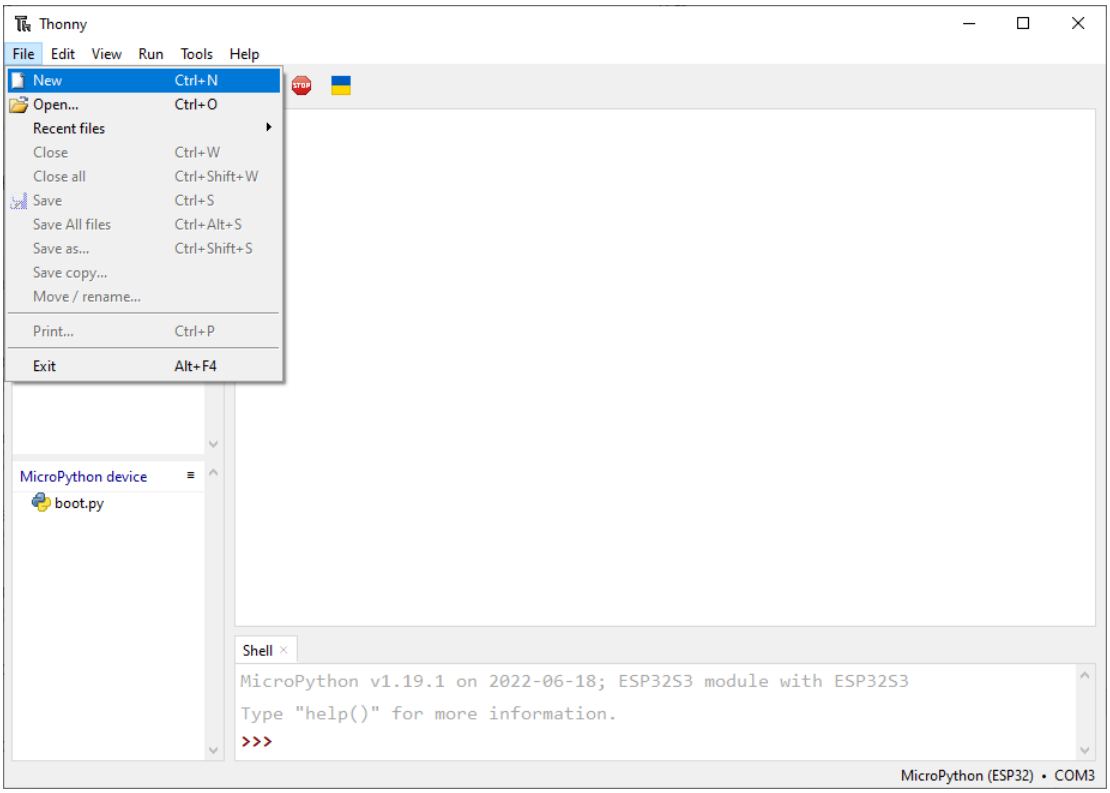
Deleting Files from your Computer Directory

Select "boot.py" in "00.1_Boot", right-click it and select "Move to Recycle Bin" to delete it from "00.1_Boot".

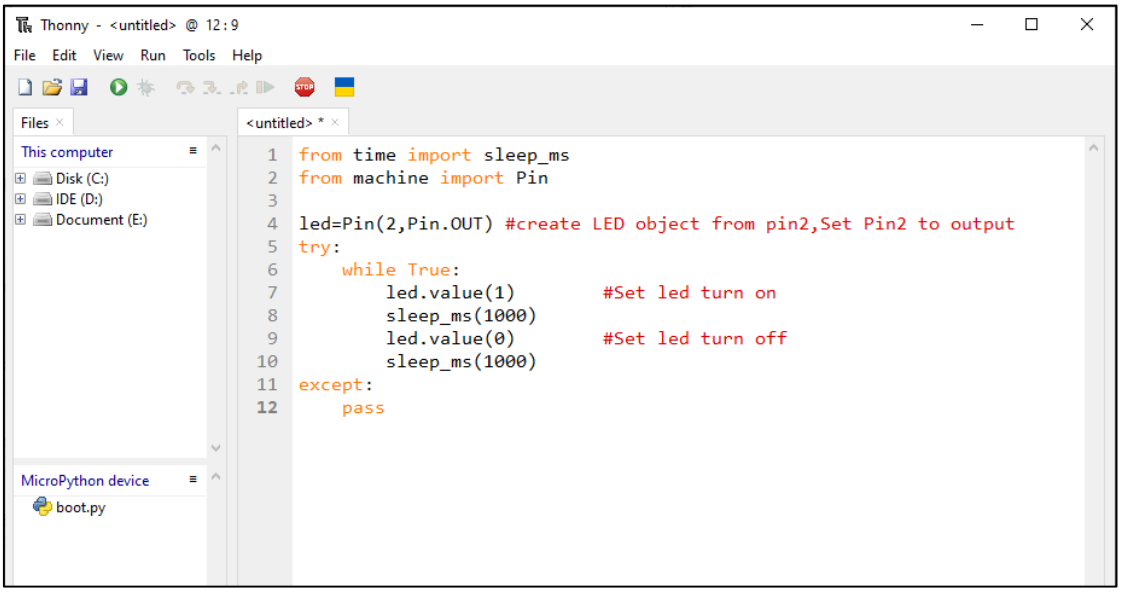


Creating and Saving the code

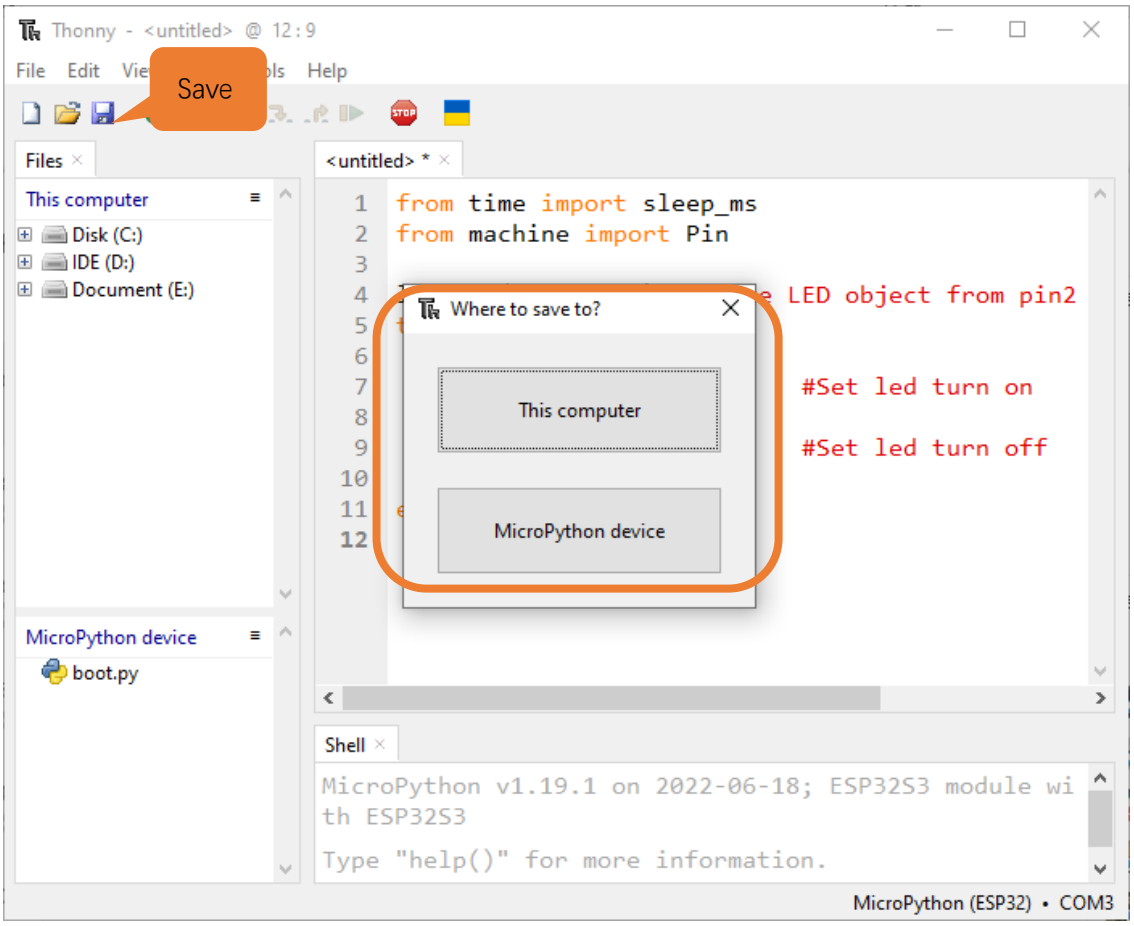
Click "File"→"New" to create and write codes.



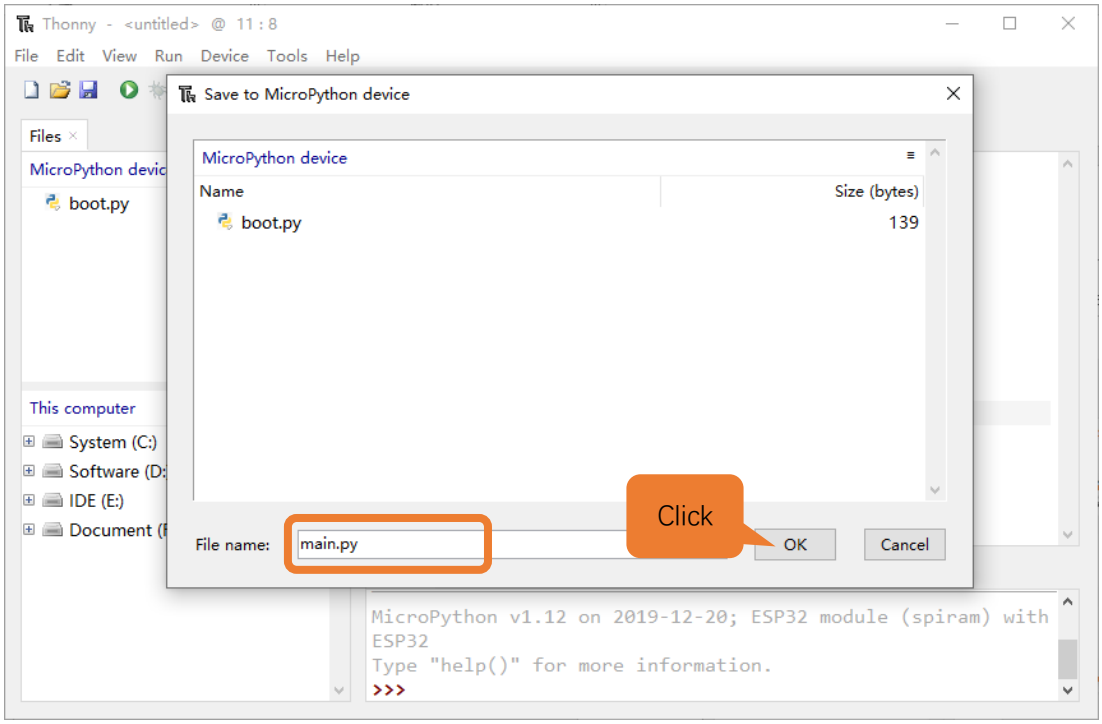
Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



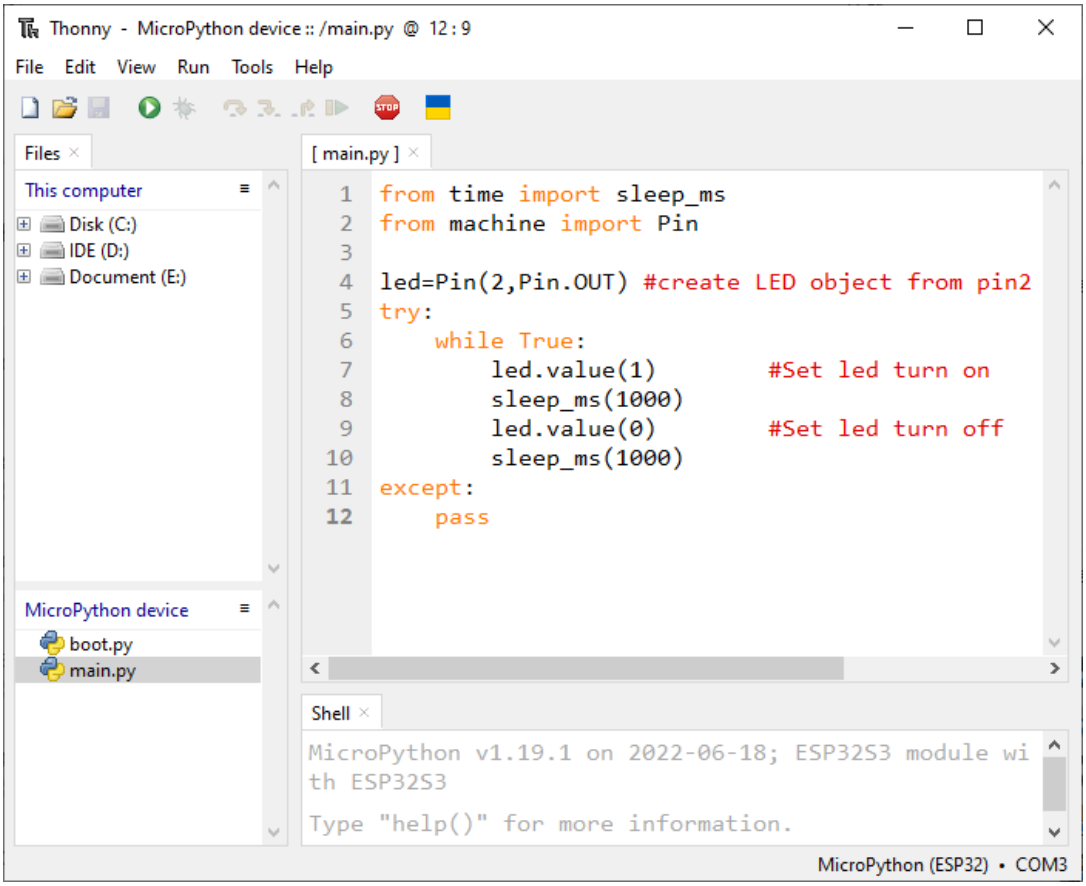
Click “Save” on the menu bar. You can save the codes either to your computer or to ESP32S3.



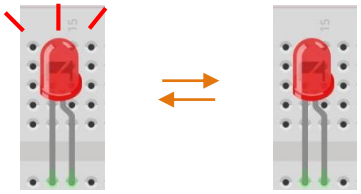
Select "MicroPython device", enter "main.py" in the newly pop-up window and click "OK".



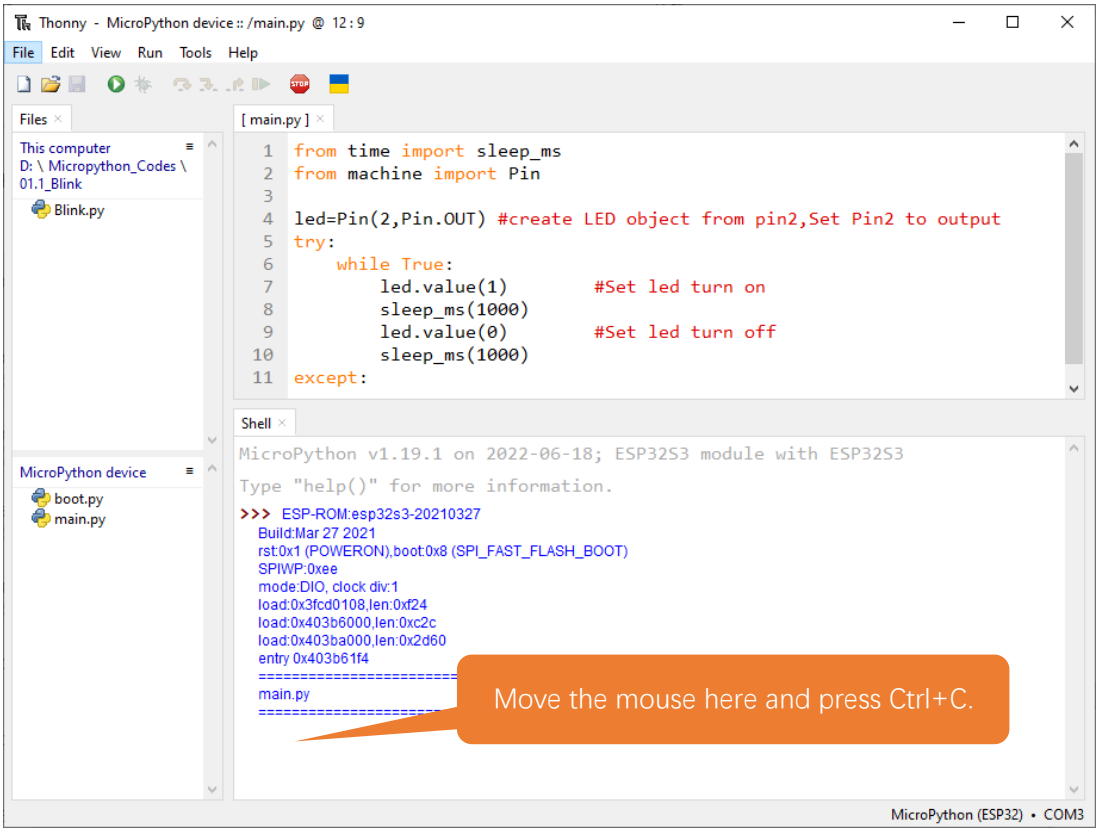
You can see that codes have been uploaded to ESP32S3.



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.



If you want to exit the offline operation mode, you can press Ctrl+C at the same time in the shell to let the ESP32-S3 exit the offline operation mode.



If there is no response after pressing, it is recommended to press again until exiting.

Notes for GPIO

Strapping Pin

There are four Strapping pins for ESP32S3: GPIO0、GPIO45、GPIO46、GPIO3。
With the release of the chip's system reset (power-on reset, RTC watchdog reset, undervoltage reset), the strapping pins sample the level and store it in the latch as "0" or "1" ", and keep it until the chip is powered off or turned off.
Each Strapping pin is connecting to internal pull-up/pull-down. Connecting to high-impedance external circuit or without an external connection, a strapping pin's default value of input level will be determined by internal weak pull-up/pull-down. To change the value of the Strapping, users can apply an external pull-down/pull-up resistor, or use the GPIO of the host MCU to control the level of the strapping pin when the ESP32-S3's power on reset is released.
When releasing the reset, the strapping pin has the same function as a normal pin.
The followings are default configurations of these four strapping pins at power-on and their functions under the corresponding configuration.

VDD_SPI Voltage			
Pin	Default	3.3 V	1.8 V
GPIO45	Pull-down	0	1
Booting Mode [1]			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO46	Pull-down	Don't care	0
Enabling/Disabling ROM Messages Print During Booting [2]			
Pin	Default	Enabled	Disabled
GPIO46	Pull-down	See the 2nd note	See the 2nd note
JTAG Signal Selection			
Pin	Default	EFUSE_DIS_USB_JTAG = 0, EFUSE_DIS_PAD_JTAG = 0, EFUSE_STRAP_JTAG_SEL=1	
GPIO3	N/A	0: JTAG signal from on-chip JTAG pins 1: JTAG signal from USB Serial/JTAG controller	

Note:

1. The strapping combination of GPIO46 = 1 and GPIO0 = 0 is invalid and will trigger unexpected behavior.
2. By default, the ROM boot messages are printed over UART0 (U0TXD pin) and USB Serial/JTAG controller together. The ROM code printing can be disabled through configuration register and eFuse. For detailed information, please refer to Chapter Chip Boot Control in *ESP32-S3 Technical Reference Manual*.

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-s3-wroom-1_wroom-1u_datasheet_en.pdf

PSRAM Pin

The modules on the ESP32-S3 WROOM board use the ESP32-S3R8 chip with 8MB external Flash. When using OPI PSRAM, please note that GPIO35-GPIO37 on the ESP32-S3 WROOM board cannot be used for other purposes. When OPI PSRAM is not used, GPIO35-GPIO37 on the board can be used as a common GPIO.

ESP32-S3R8 / ESP32-S3R8V	In-package PSRAM (8 MB, Octal SPI)
SPICLK	CLK
SPICS1	CE#
SPID	DQ0
SPIQ	DQ1
SPIWP	DQ2
SPIHD	DQ3
GPIO33	DQ4
GPIO34	DQ5
GPIO35	DQ6
GPIO36	DQ7
GPIO37	DQS/DM

SDcard Pin

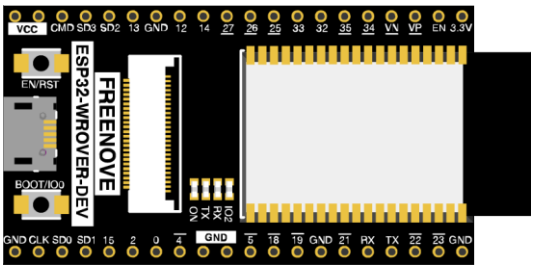
An SD card slot is integrated on the back of the ESP32-S3 WROOM board. We can use GPIO38-GPIO40 of ESP32-S3 WROOM to drive the SD card.

USB Pin

In Micropython, GPIO19 and GPIO20 are used for the USB function of ESP32S3, so they cannot be used as other functions!

Cam Pin

When using the camera of our ESP32-S3 WROOM, please check the pins of it. Pins with underlined numbers are used by the camera function, if you want to use other functions besides it, please avoid using them.



CAM_Pin	GPIO_pin
SIOD	GPIO4
SIOC	GPIO5
CSI_VYSNC	GPIO6
CSI_HREF	GPIO7
CSI_Y9	GPIO16
XCLK	GPIO15
CSI_Y8	GPIO17
CSI_Y7	GPIO18
CSI_PCLK	GPIO13
CSI_Y6	GPIO12
CSI_Y2	GPIO11
CSI_Y5	GPIO10
CSI_Y3	GPIO9
CSI_Y4	GPIO8

If you have any questions about the information of GPIO, you can click here to go back to ESP32-S3 WROOM to view specific information about GPIO.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf.

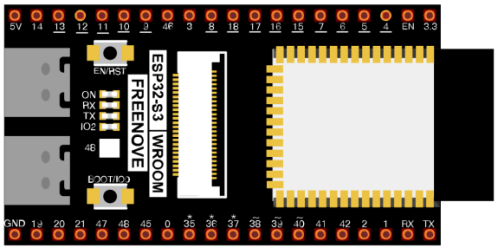
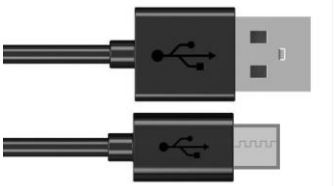
Chapter 1 LED (Important)

This chapter is the Start Point in the journey to build and explore ESP32-S3 WROOM electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

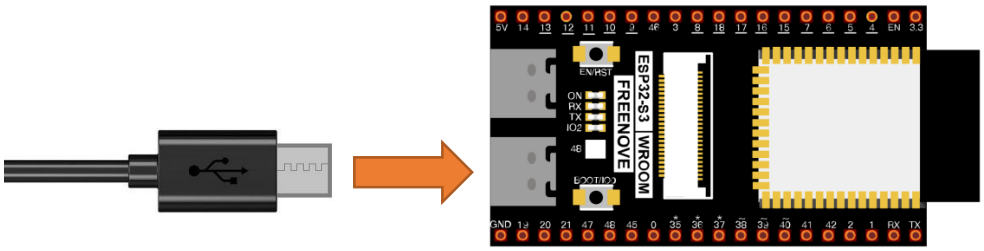
In this project, we will use ESP32-S3 WROOM to control blinking a common LED.

Component List

ESP32-S3 WROOM x1	USB cable x1
	

Power

ESP32-S3 WROOM needs 5v power supply. In this tutorial, we need connect ESP32-S3 WROOM to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



In the following projects, we only use USB cable to power ESP32-S3 WROOM by default.

In the whole tutorial, we don’t use T extension to power ESP32-S3 WROOM. So 5V and 3.3V (includeing EXT 3.3V) on the extension board are provided by ESP32-S3 WROOM.

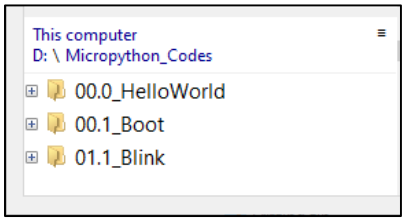
We can also use DC jack of extension board to power ESP32-S3 WROOM. In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

Code

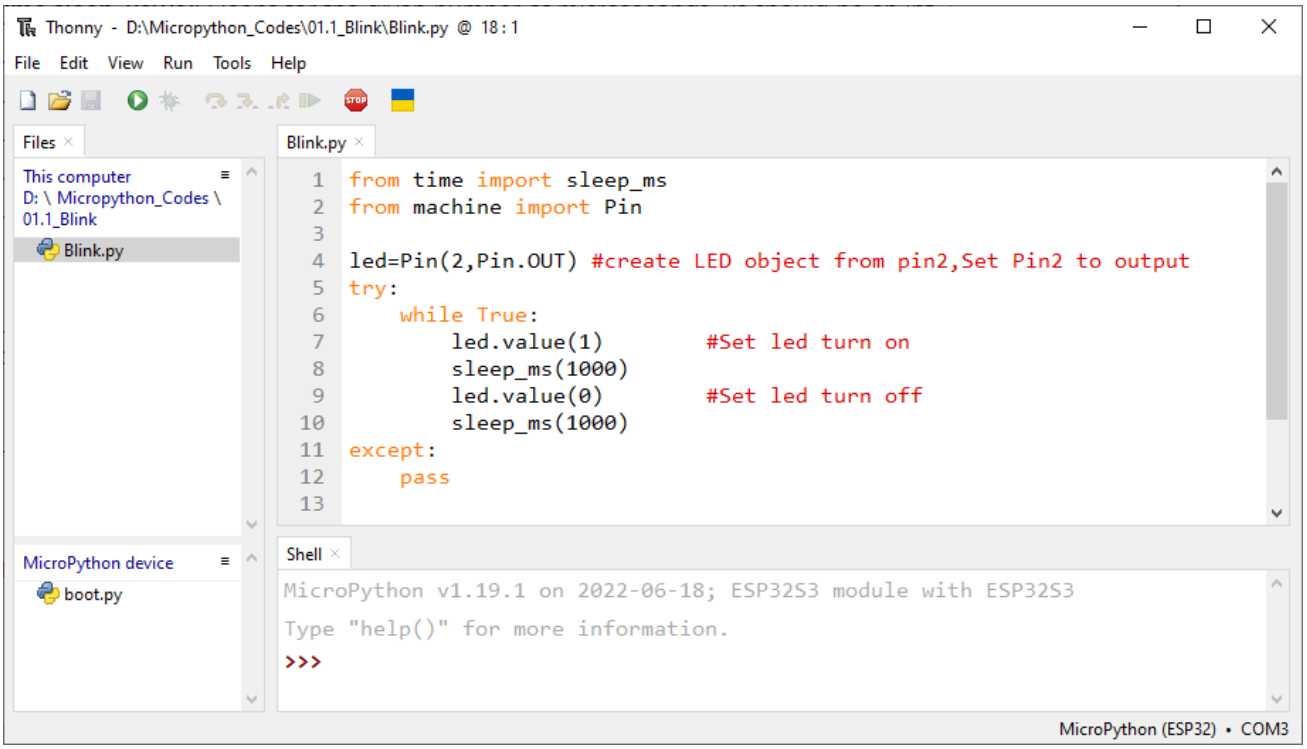
Codes used in this tutorial are saved in "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

01.1_Blink

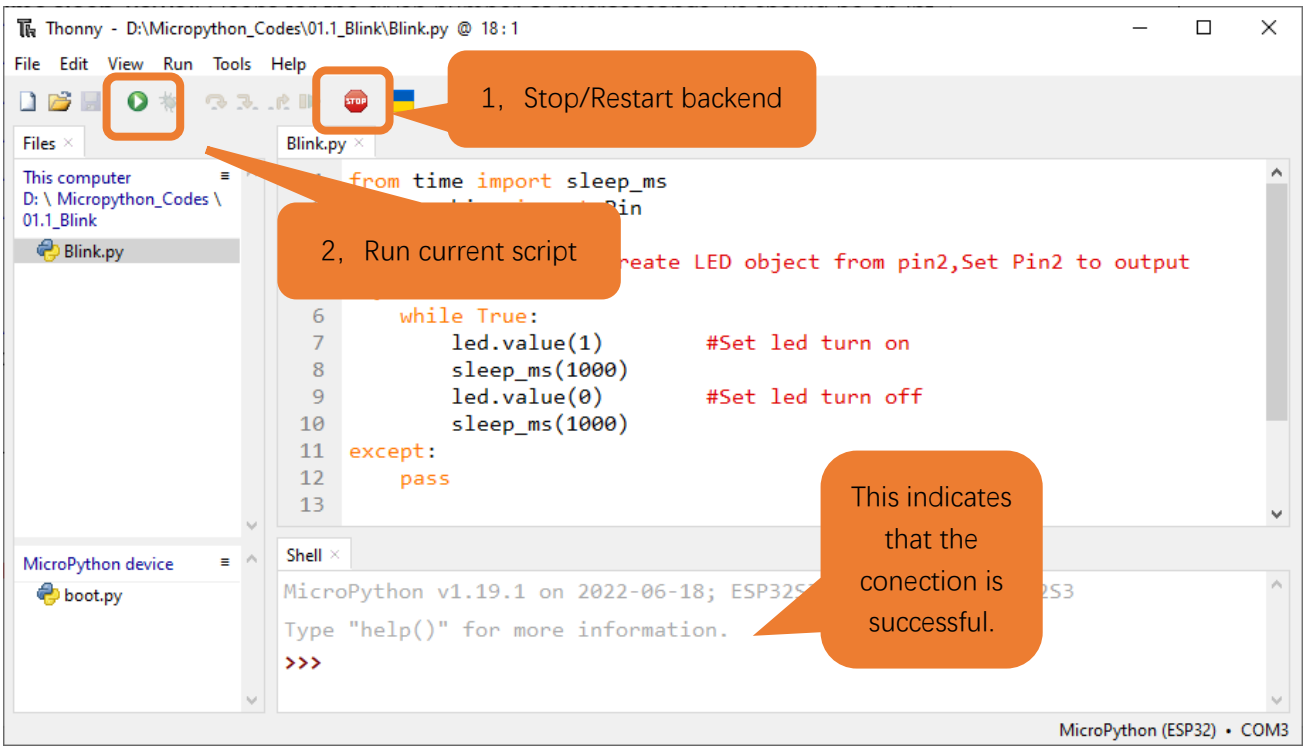
Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".



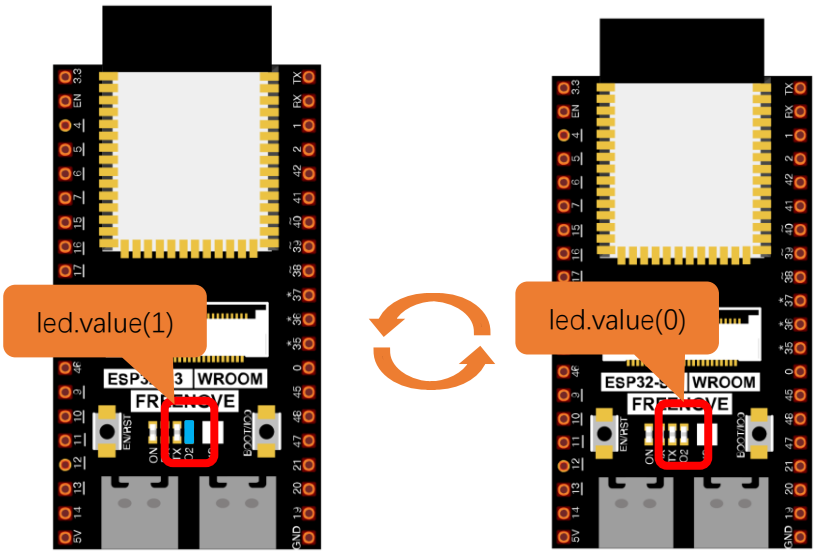
Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.



Make sure ESP32-S3 has been connected with the computer with ESP32-S3 correctly. Click "Stop/Restart backend" or press the reset button, and then wait to see what interface will show up.

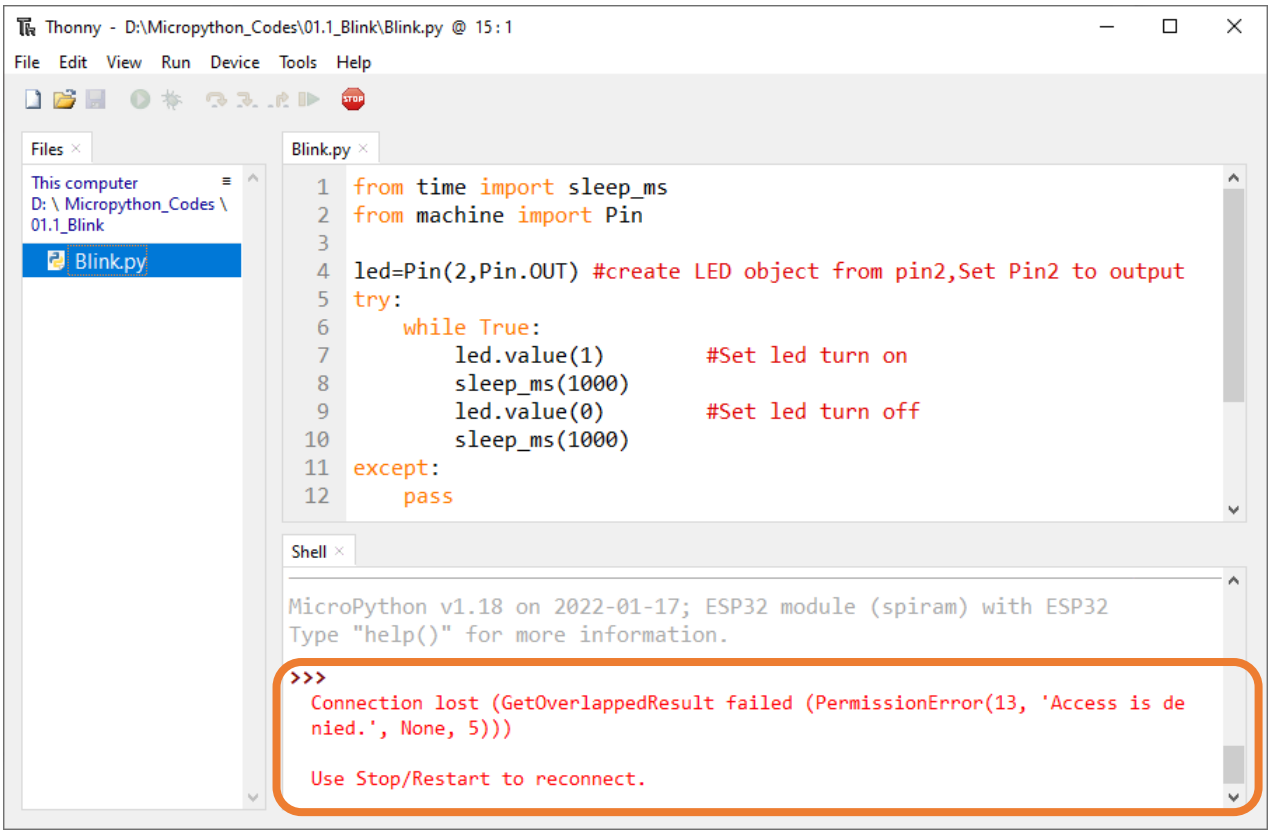


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



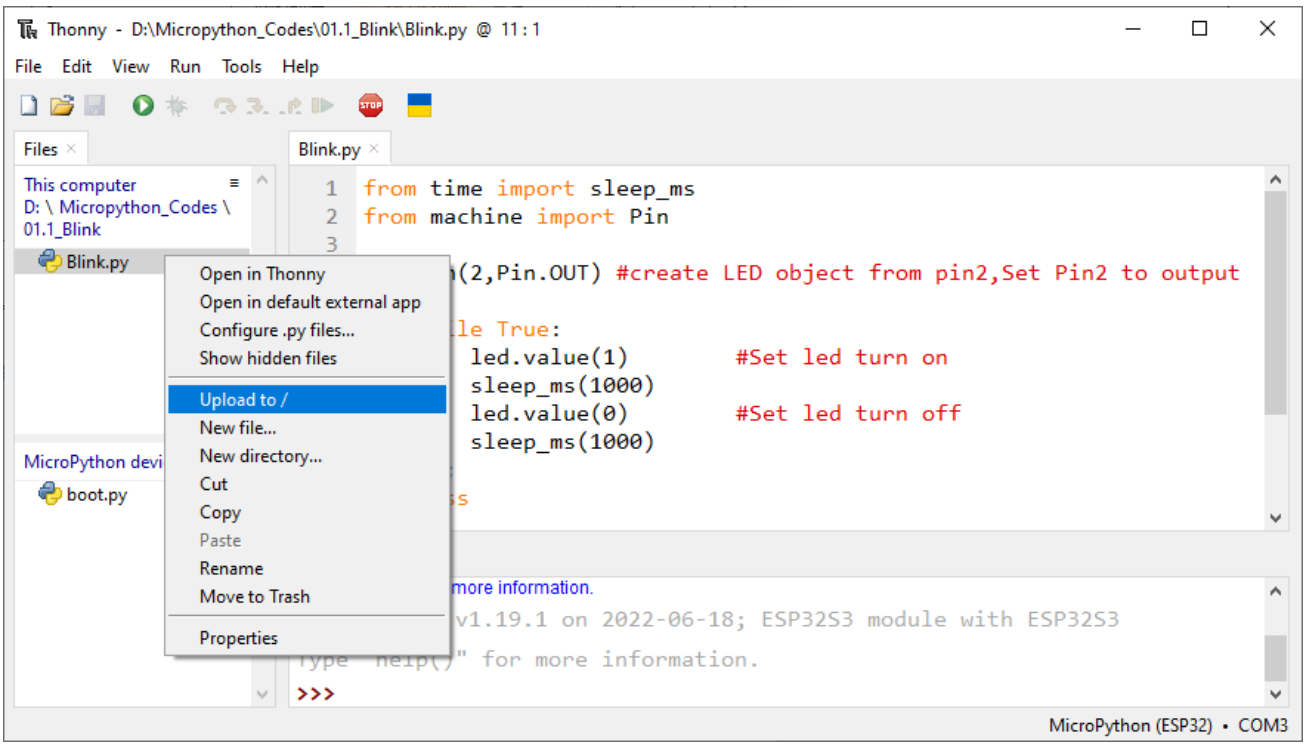
Note:

This is the code running online. If you disconnect USB cable and repower ESP32-S3 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

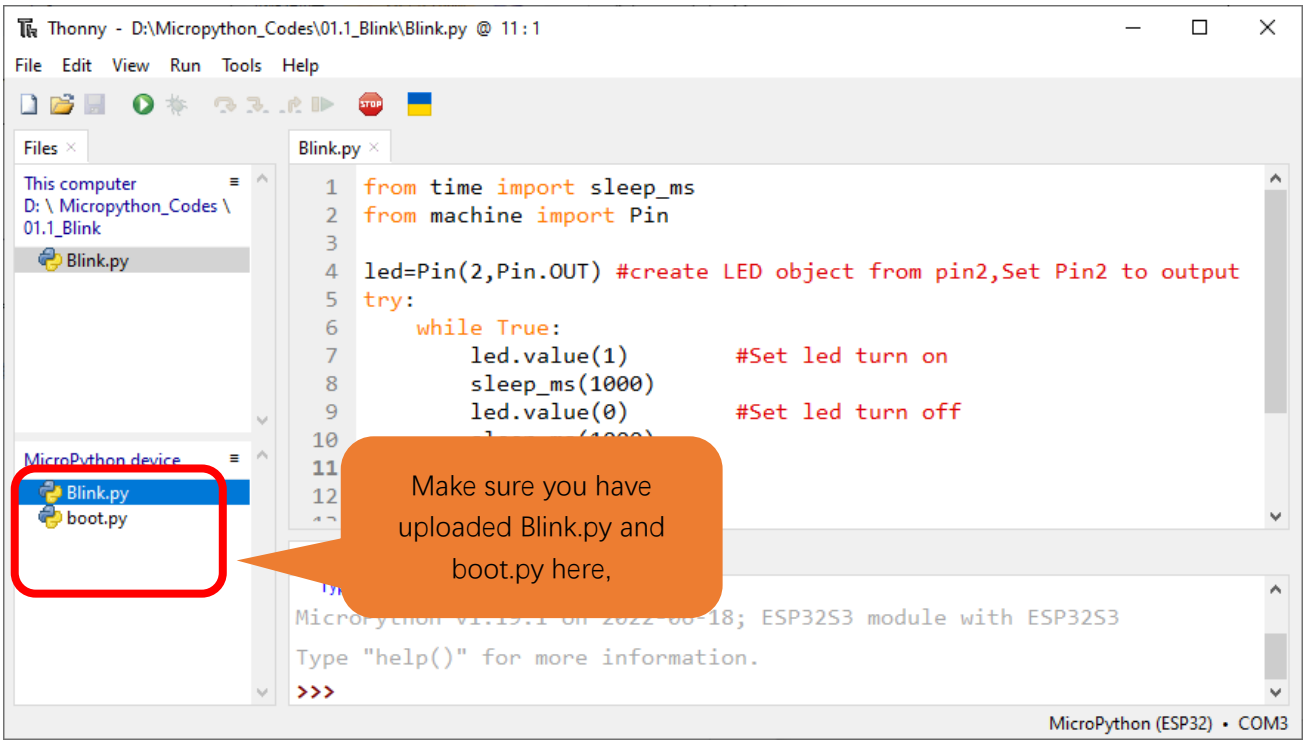


Uploading code to ESP32S3

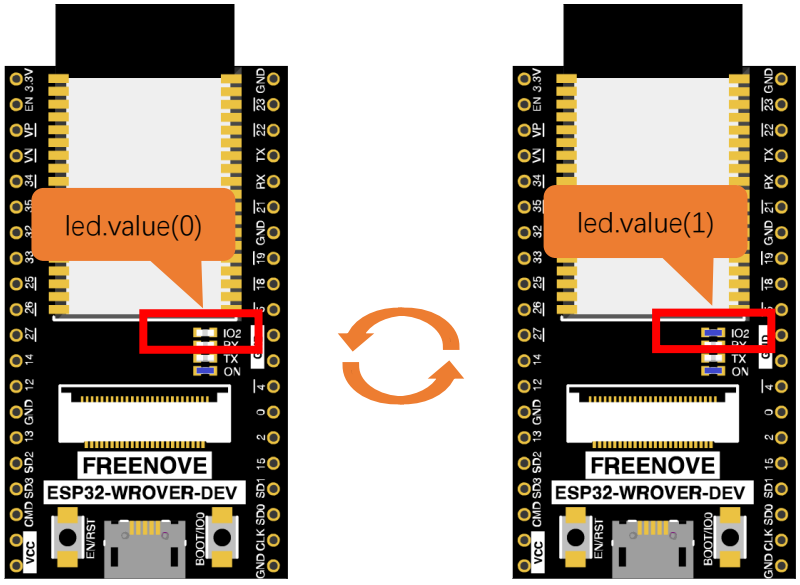
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP32S3.



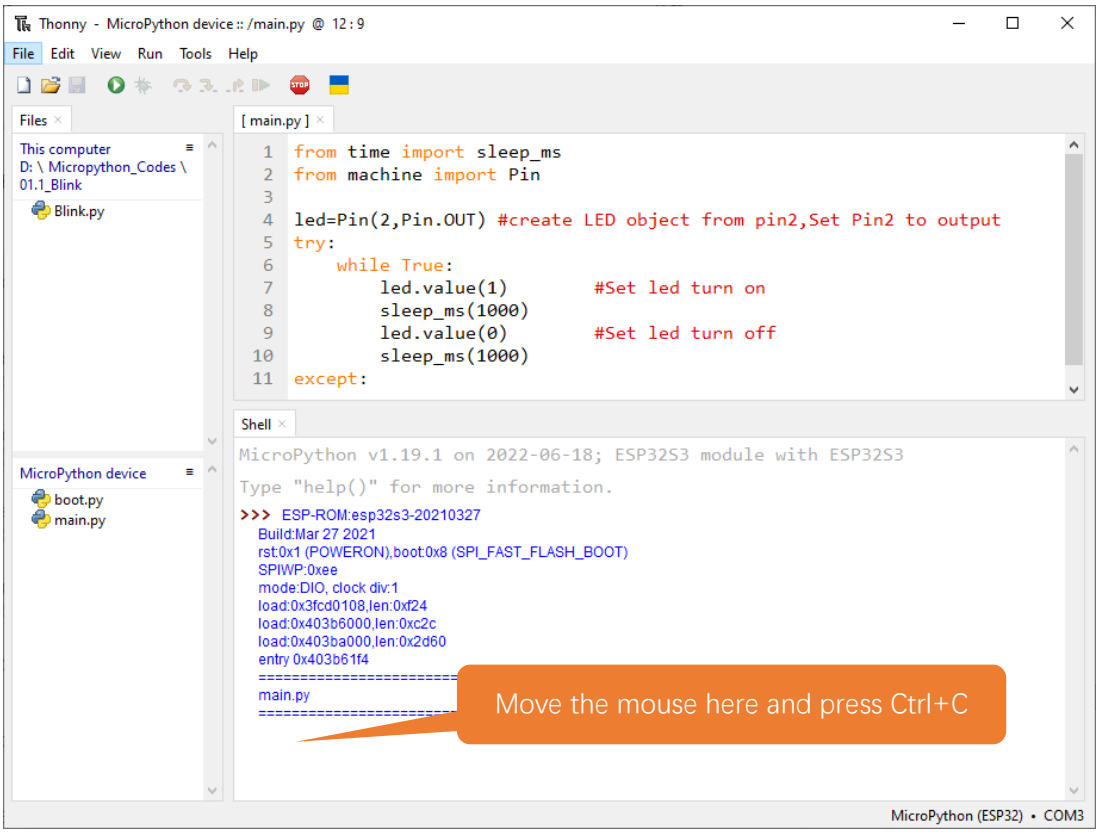
Upload boot.py in the same way.



Press the reset key of ESP32-S3 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



If you want to exit the offline operation mode, you can press Ctrl+C at the same time in the shell to let the ESP32-S3 exit the offline operation mode.



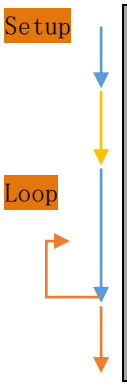
If there is no response after pressing, it is recommended to press again until exiting.

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.




```
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:
…          ...
11 except:
12     pass
```

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 ESP32S3, 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-S3 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 GPIO2 that is in output mode.

Set the value of LED to 1 and GPIO2 will output high level.

```
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:
…          ...
```

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-S3 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  except:
12      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().

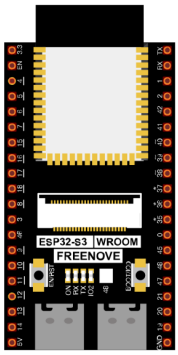
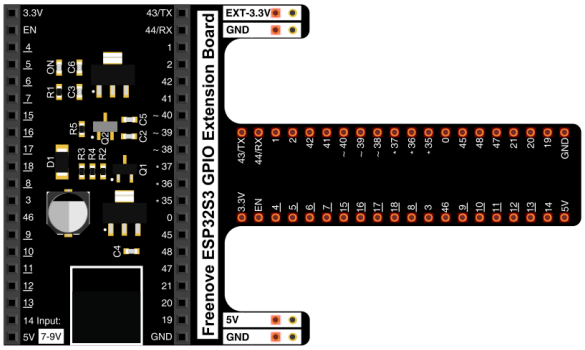
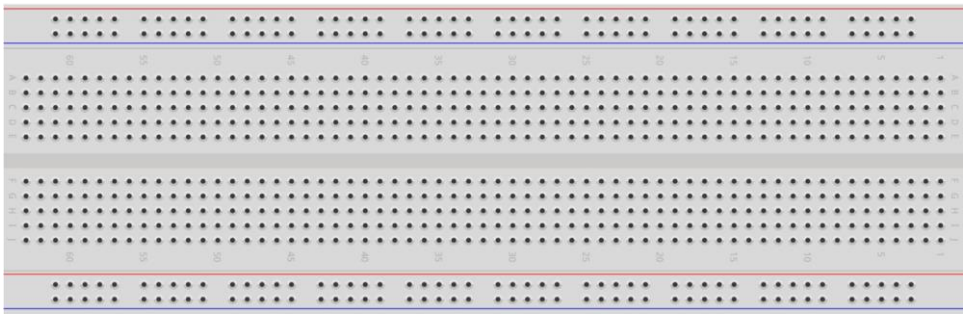
old_t: Starting time

new_t: Ending time

Project 1.2 Blink

In this project, we will use ESP32-S3 WROOM to control blinking a common LED.

Component List

ESP32-S3 WROOM x1 	GPIO Extension Board x1 	
Breadboard x1 		
LED x1 	Resistor 220Ω x1 	Jumper M/M x2 

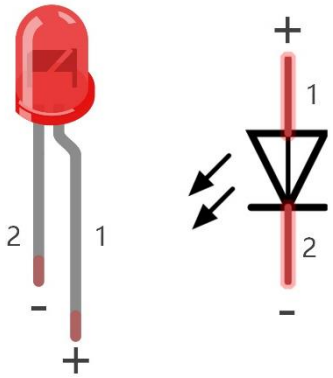
Component knowledge

LED

A LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two poles. A 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 "diodes" (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.



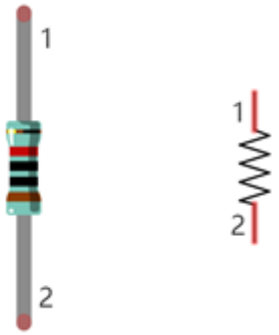
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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). 1MΩ=1000kΩ, 1kΩ=1000Ω.

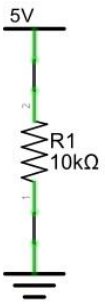
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the 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/10kΩ=0.0005A=0.5mA.

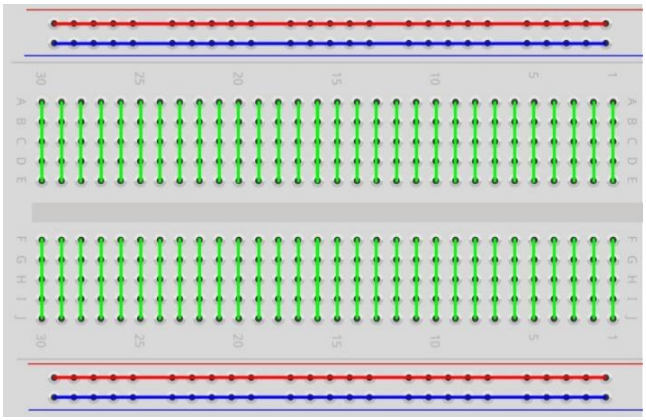


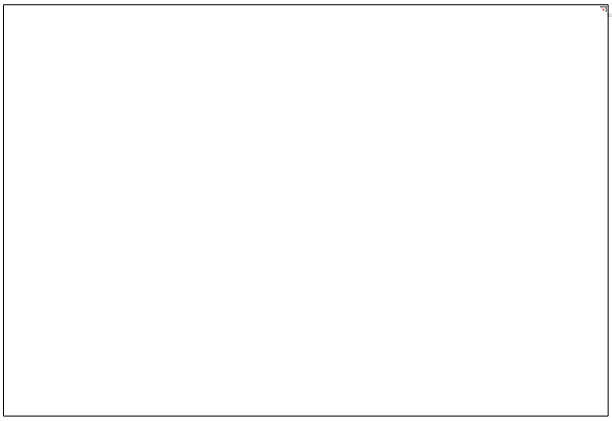
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

Breadboard

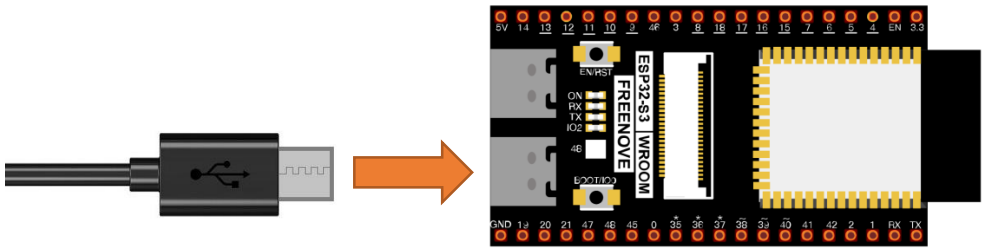
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

ESP32-S3 WROOM needs 5v power supply. In this tutorial, we need connect ESP32-S3 WROOM to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



In the following projects, we only use USB cable to power ESP32-S3 WROOM by default.

In the whole tutorial, we don't use T extension to power ESP32-S3 WROOM. So 5V and 3.3V (includeing EXT 3.3V) on the extension board are provided by ESP32-S3 WROOM.

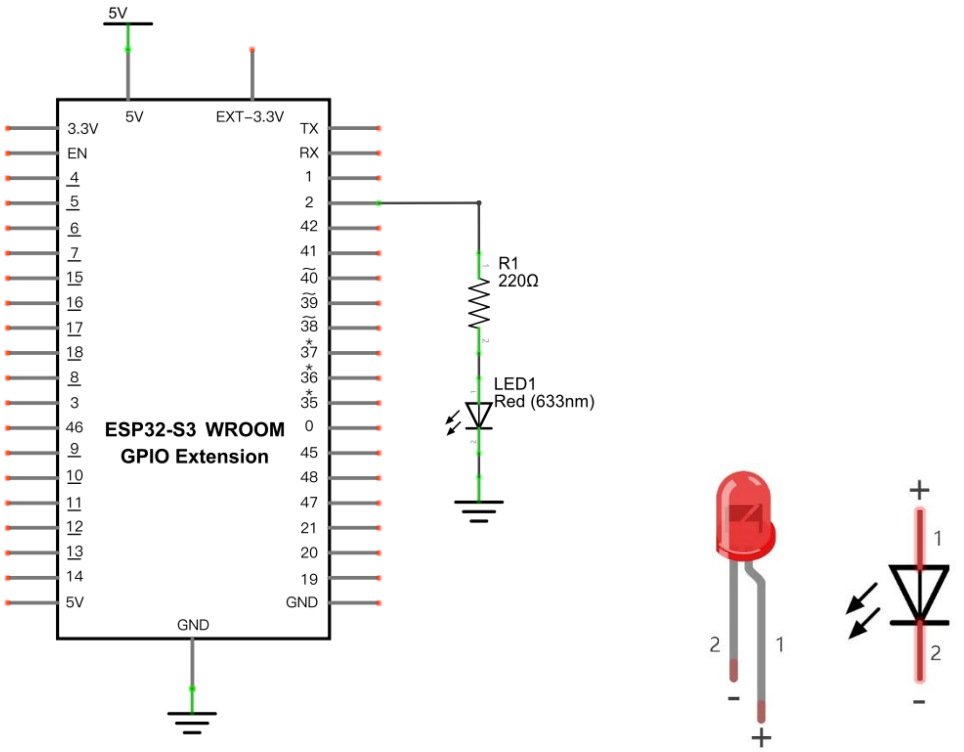
We can also use DC jack of extension board to power ESP32-S3 WROOM.In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

Circuit

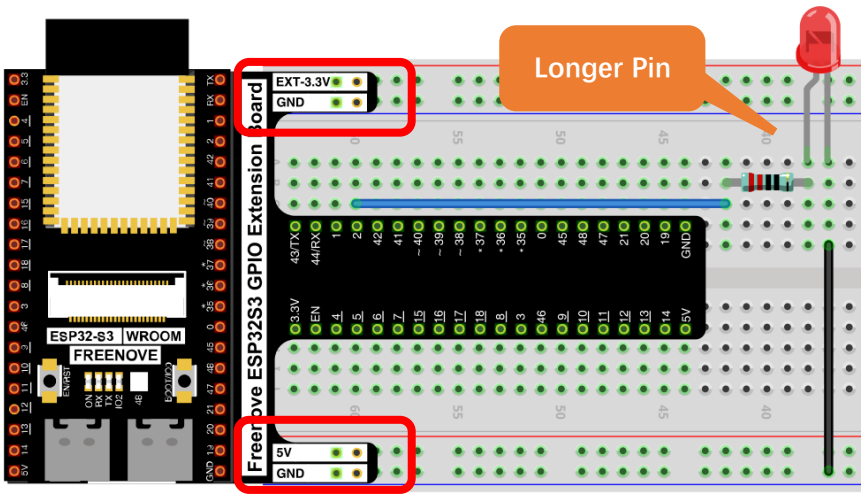
First, disconnect all power from the ESP32-S3 WROOM. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP32-S3 WROOM.

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, generate 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**



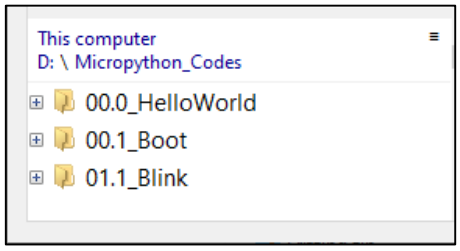
Don't rotate ESP32-S3 WROOM 180° for connection.

Code

Codes used in this tutorial are saved in "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_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**".

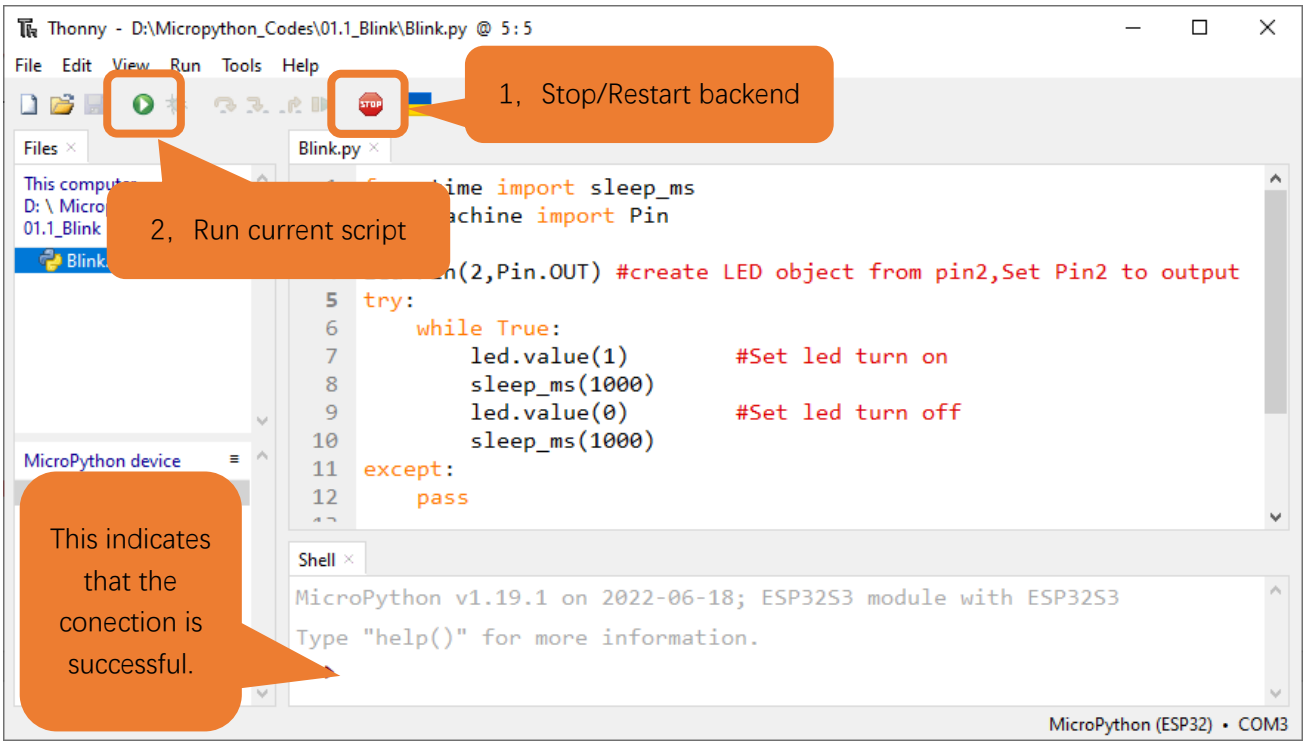
01.1_Blink

Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".

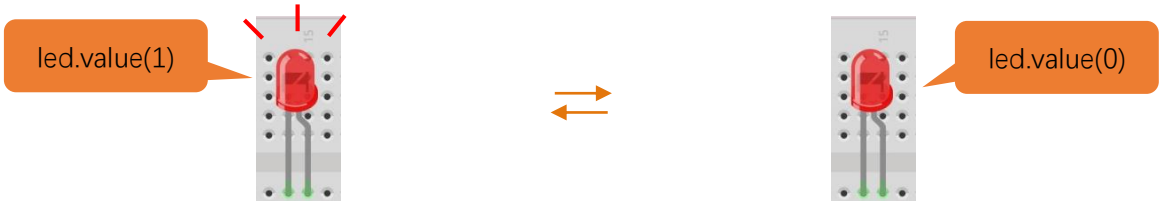


Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.

Make sure ESP32-S3 has been connected with the computer with ESP32-S3 correctly. Click "Stop/Restart backend" or press the reset button, and then wait to see what interface will show up.

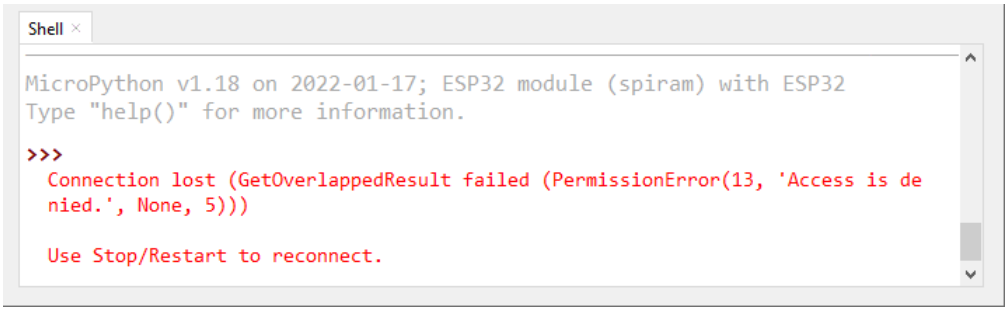


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



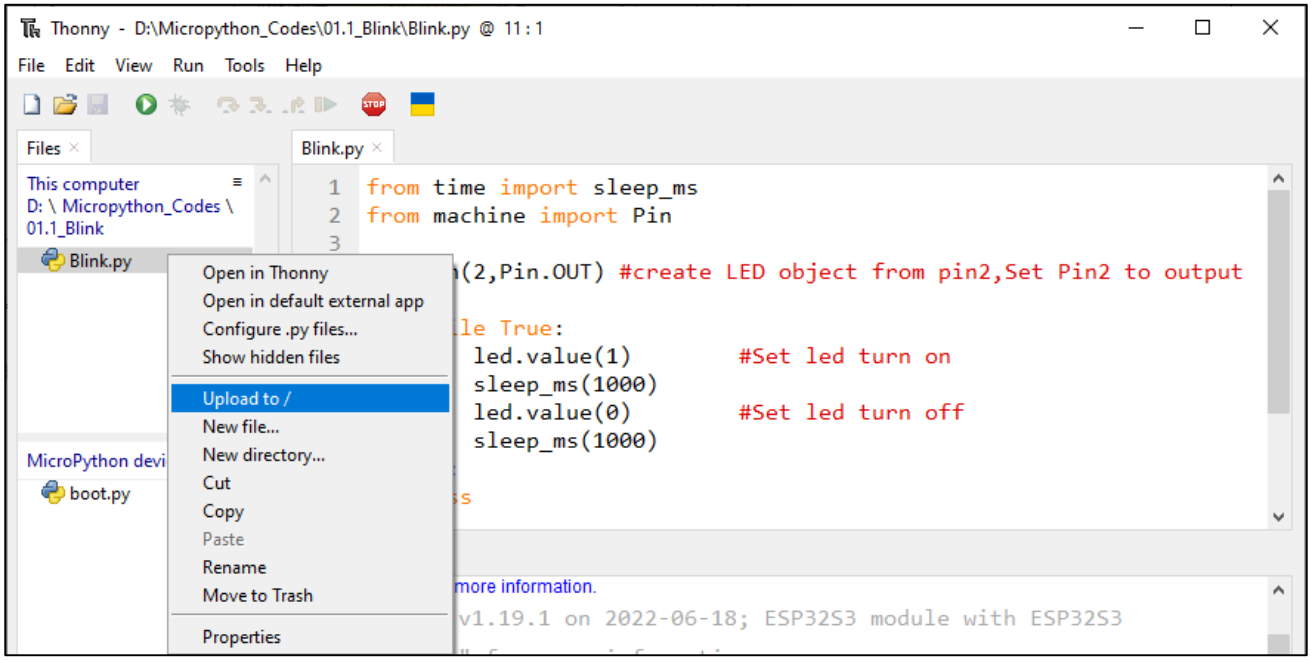
Note:

This is the code running online. If you disconnect USB cable and repower ESP32-S3 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

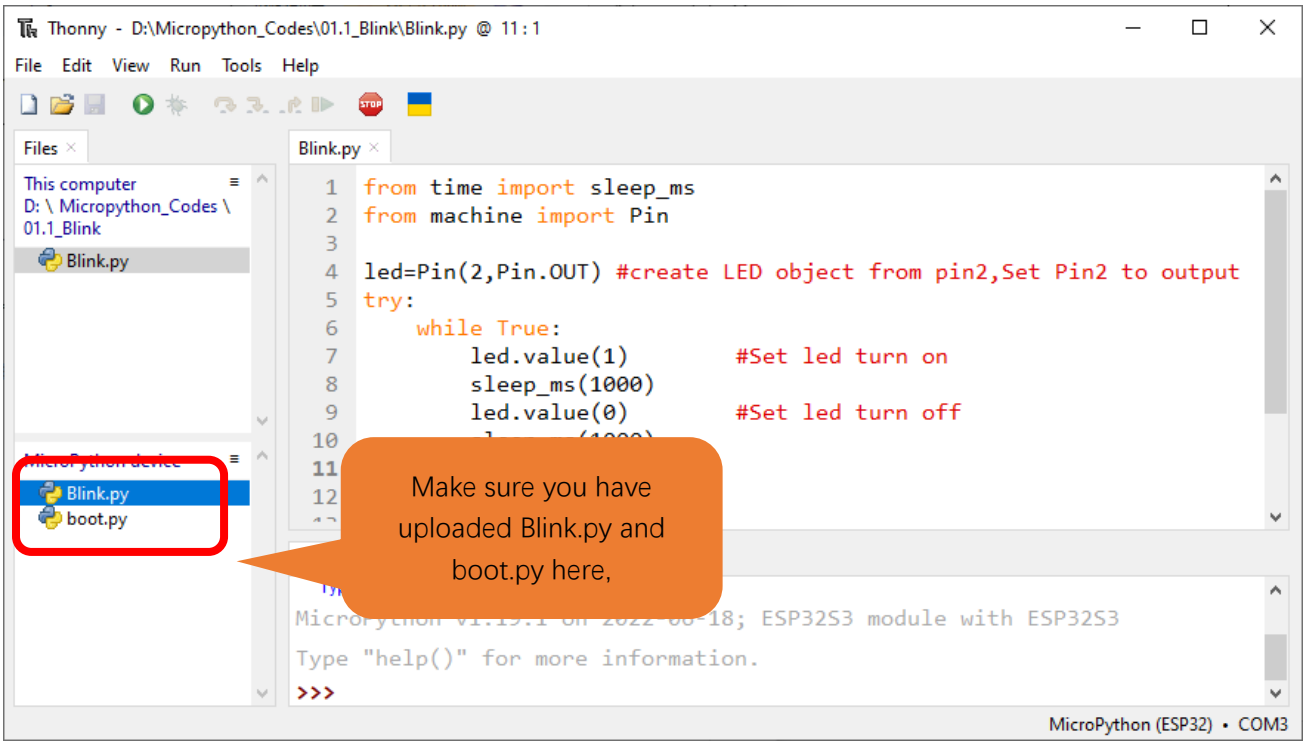


Uploading code to ESP32S3

As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP32S3.

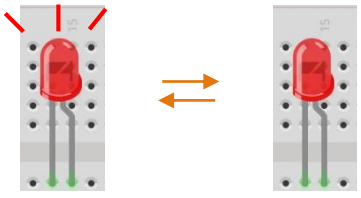


Upload boot.py in the same way.

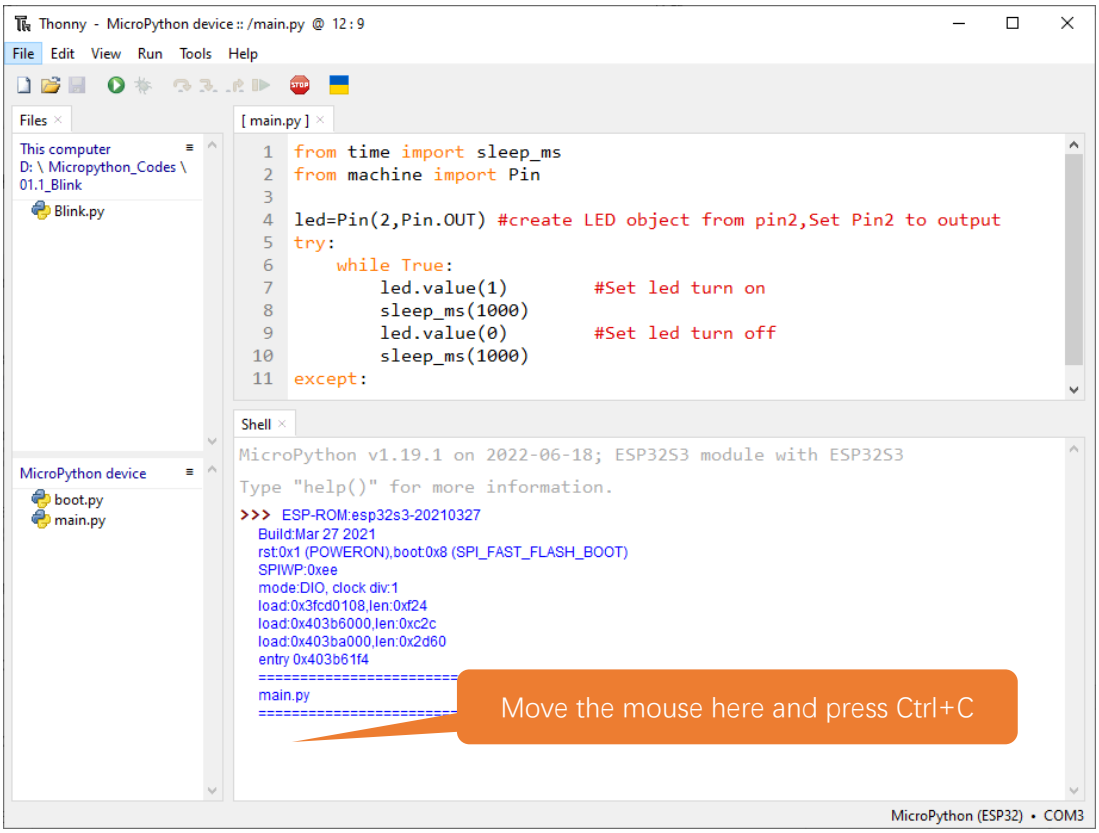


Press the reset key of ESP32-S3 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.

Press the reset key of ESP32-S3 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



If you want to exit the offline operation mode, you can press Ctrl+C at the same time in the shell to let the ESP32-S3 exit the offline operation mode.

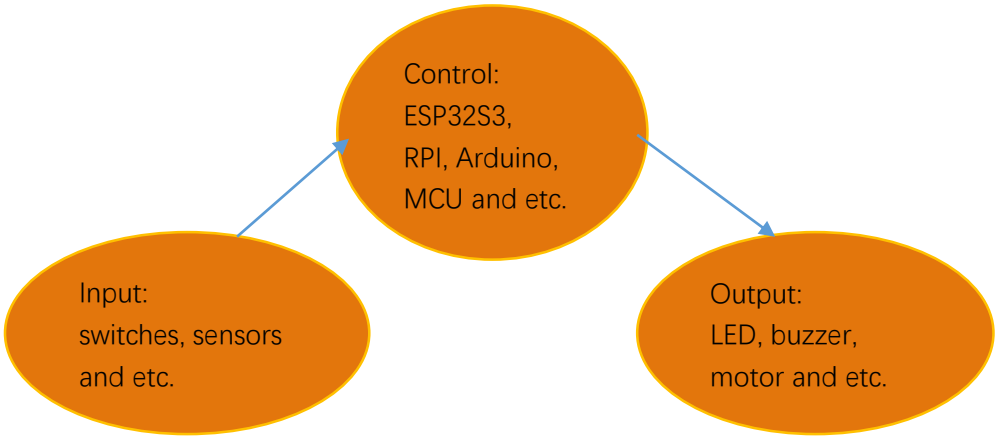


If there is no response after pressing, it is recommended to press again until exiting.

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

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-S3 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.

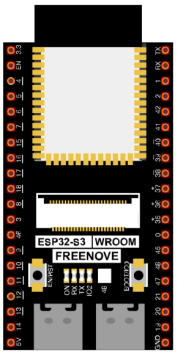
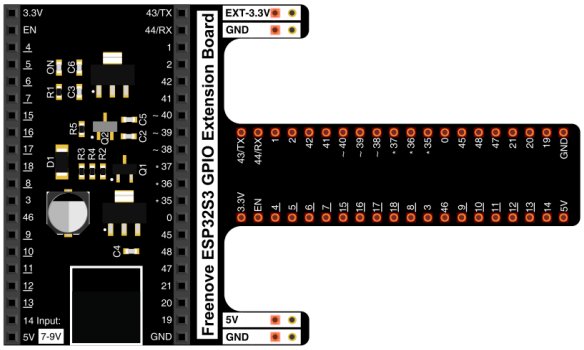
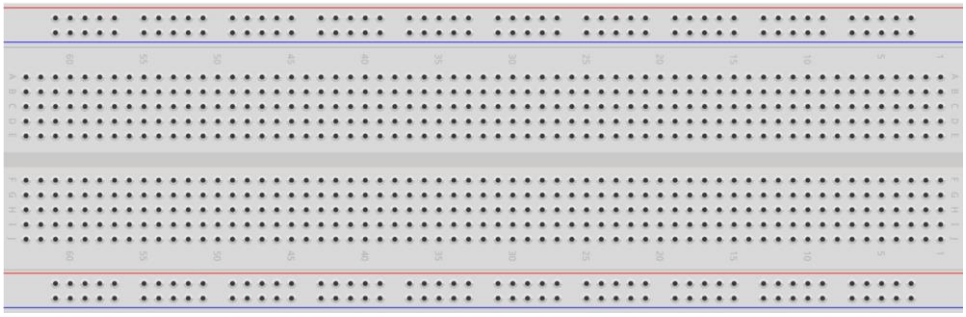


Next, we will build a simple control system to control a LED through a push button switch.

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.

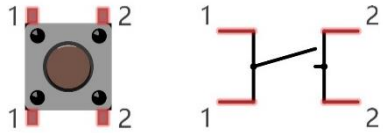
Component List

<table>
<tr><td colspan="3">ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td>Jumper M/M x4
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Resistor 10kΩ x2
  </td><td>Push button x1
  </td></tr>
</table>

Component knowledge

Push button

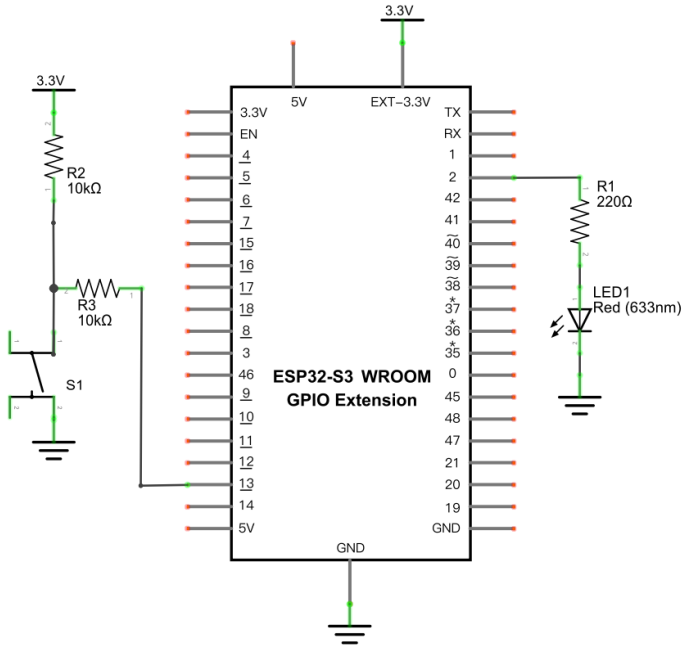
This type of push button switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



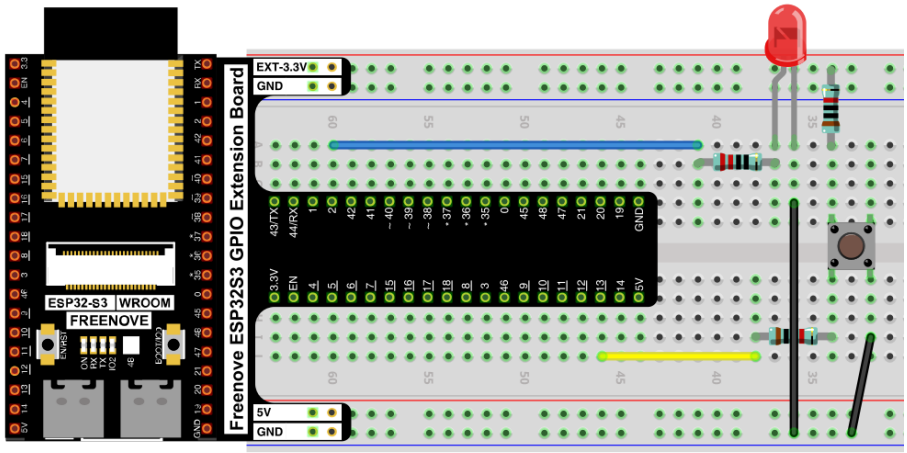
When the button on the switch is pressed, the circuit is completed (your project is powered ON).

Circuit

Schematic diagram



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



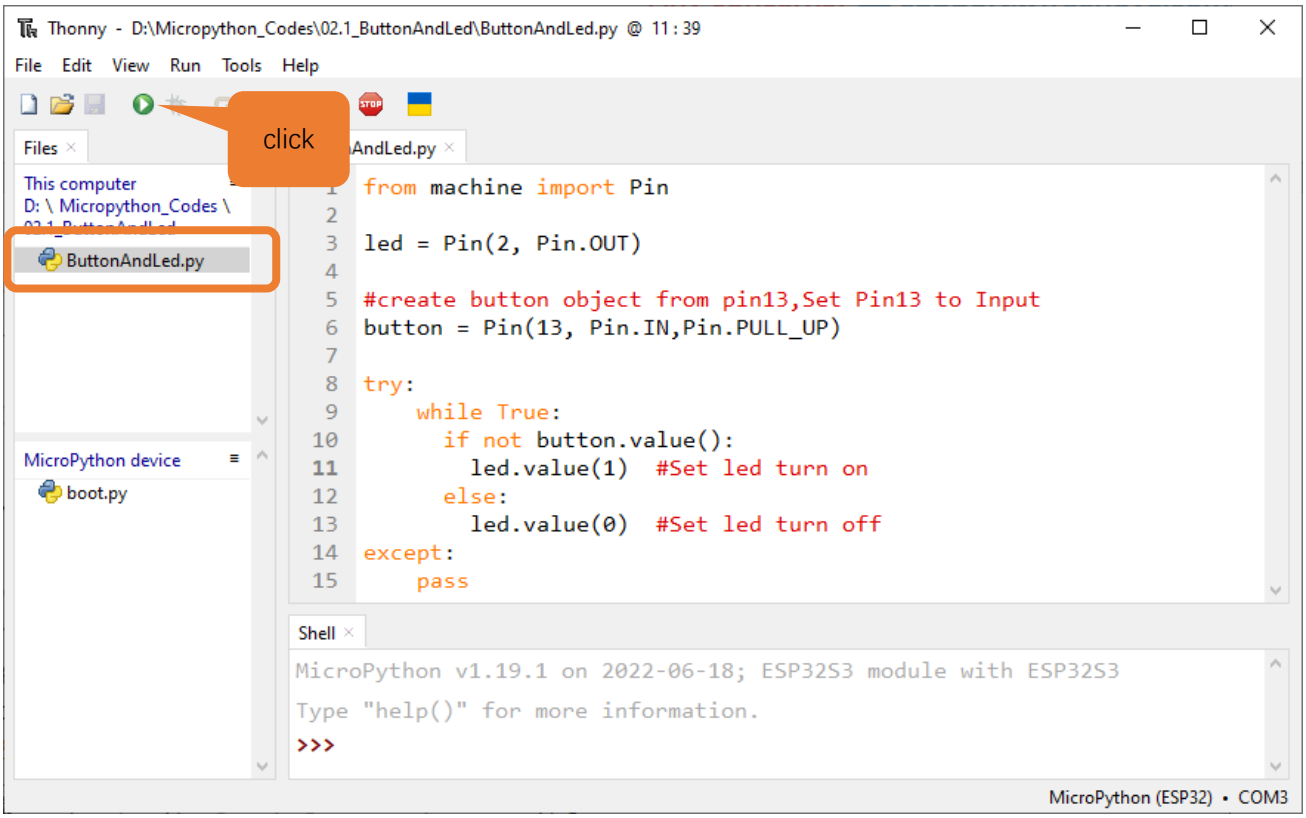
Code

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.

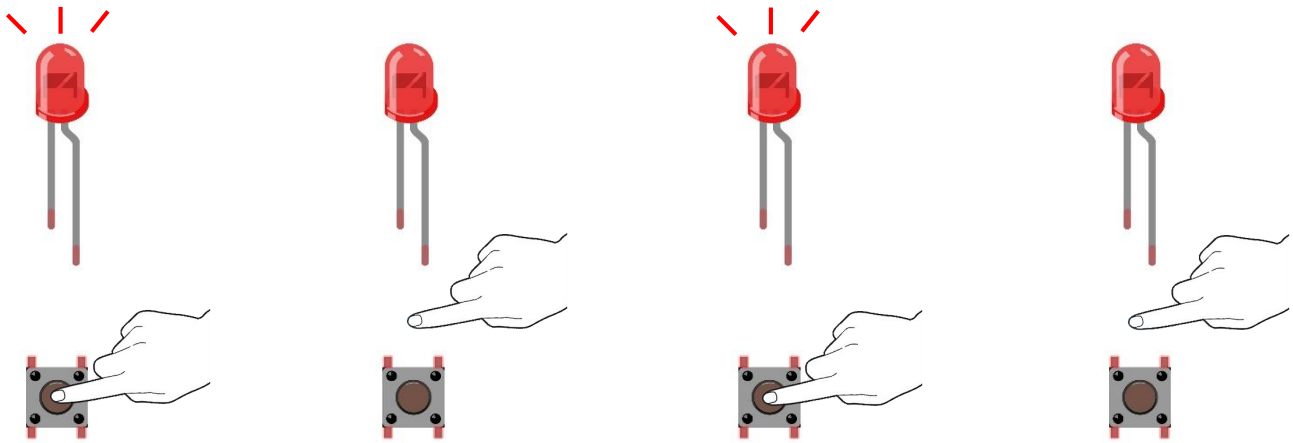
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

02.1_ButtonAndLed

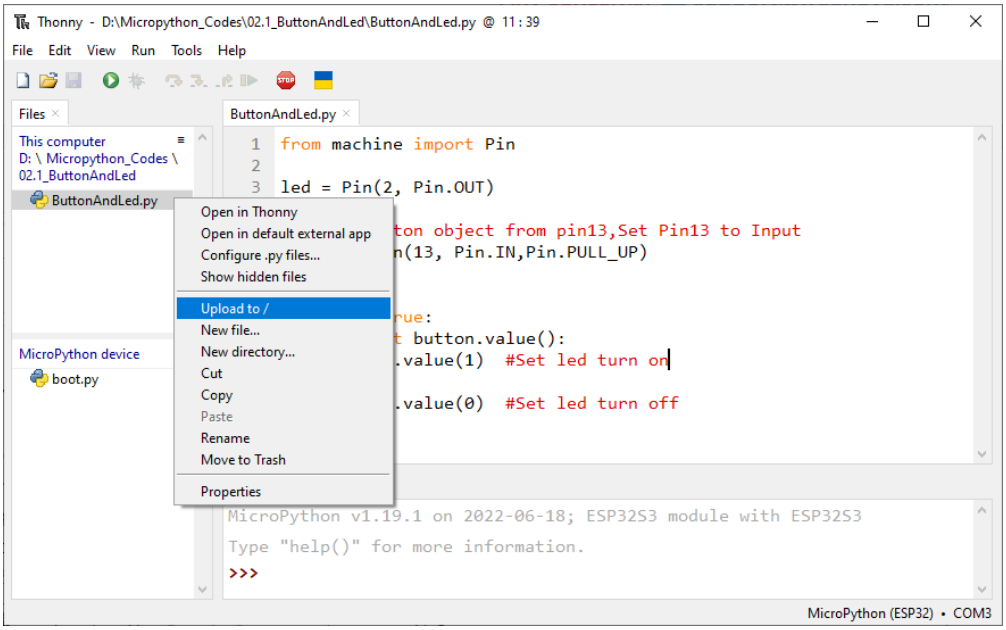


Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF.

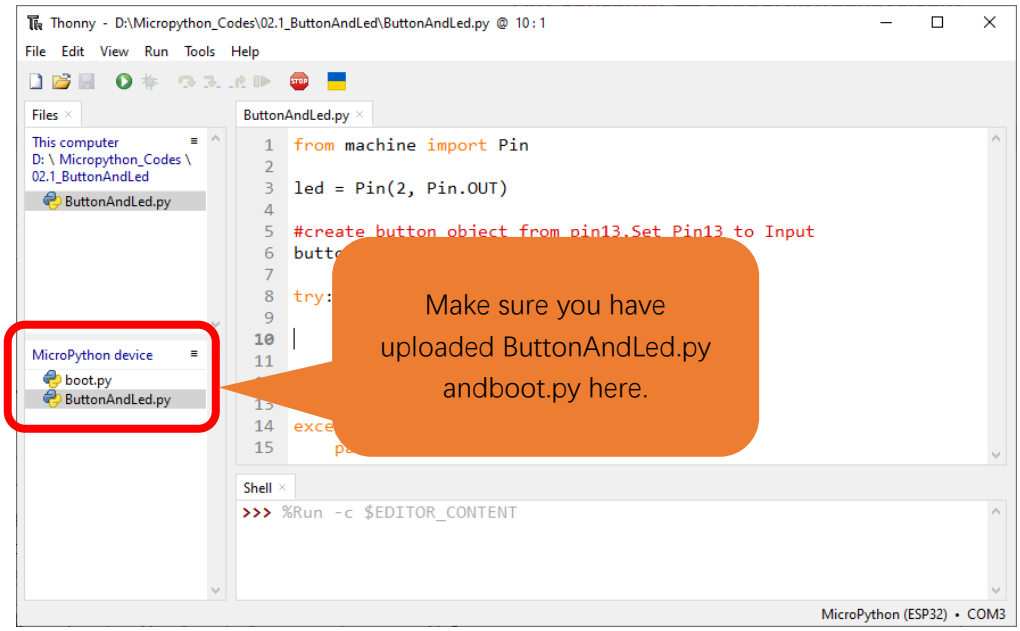


Upload Code to ESP32S3

As shown in the following illustration, right-click file 02.1_ButtonAndLed and select "Upload to /" to upload code to ESP32S3.



Upload boot.py in the same way.



The following is the program code:

```
from machine import Pin

led = Pin(2, Pin.OUT)

#create button object from pin13,Set Pin13 to Input
button = Pin(13, Pin.IN,Pin.PULL_UP)

try:
    while True:
      if not button.value():
        led.value(1)  #Set led turn on
      else:
        led.value(0)  #Set led turn off
except:
    pass
```

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

```
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.

```
led = Pin(2, Pin.OUT)

#create button object from pin13,Set Pin13 to Input
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.

```
    while True:
      if not button.value():
        led.value(1)  #Set led turn on
      else:
        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.

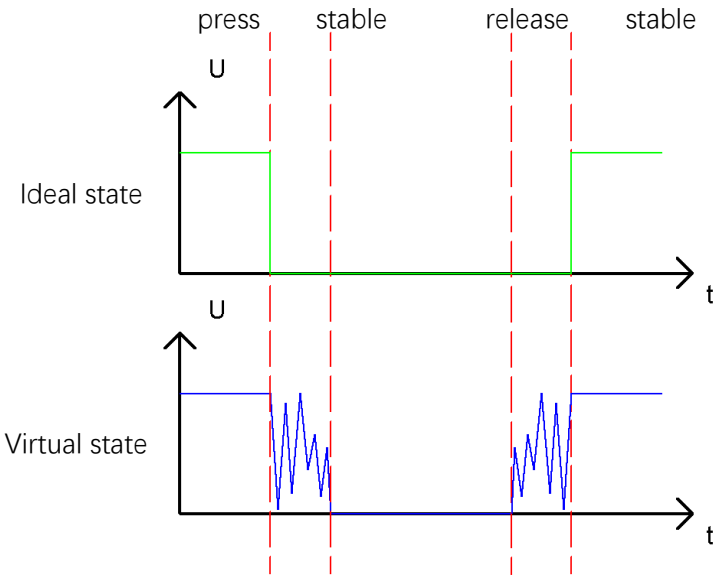
```
if not button.value():
    …
else:
    …
```

Project 2.2 MINI table lamp

We will also use a push button switch, LED and ESP32-S3 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

The moment when a 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”.



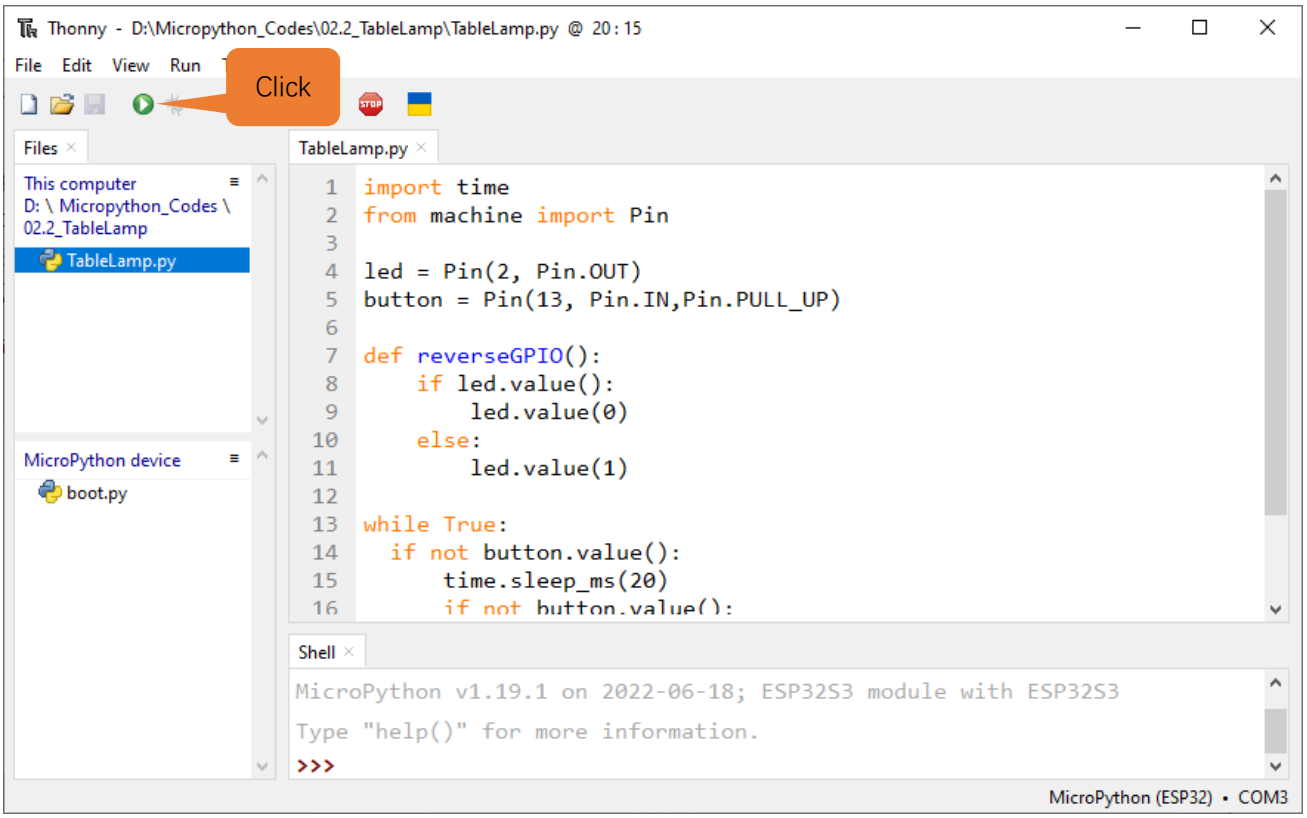
Therefore, if we can directly detect the state of the push button switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. 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

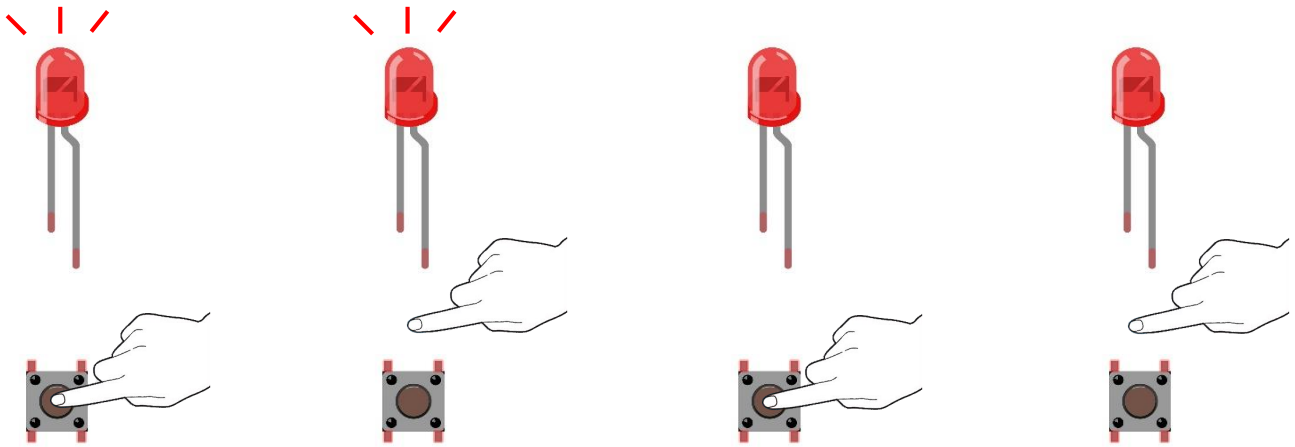
02.2_Tablelamp

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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



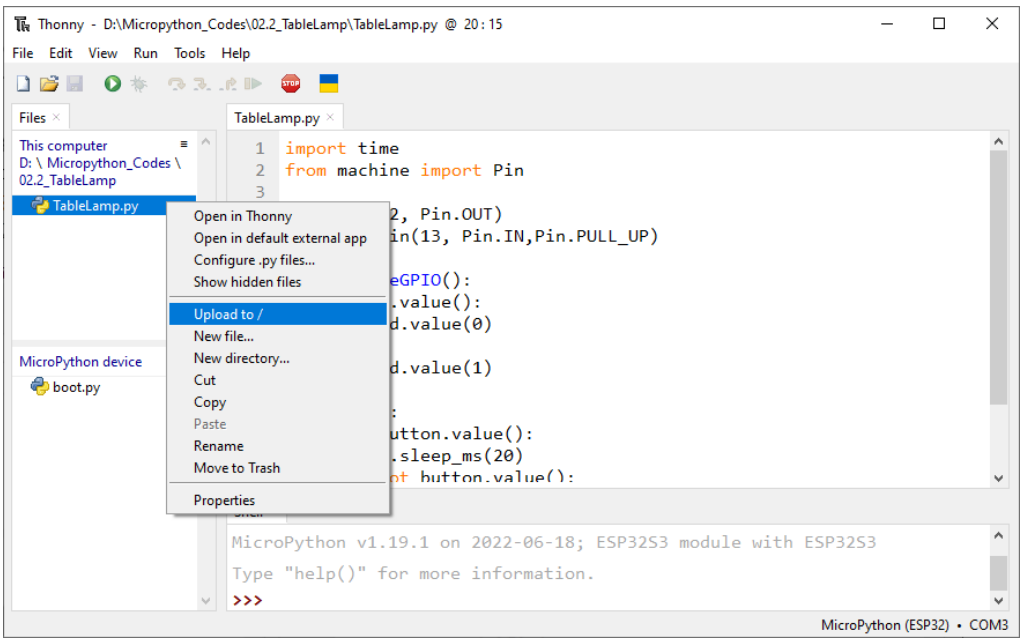
Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; press it again, LED turns OFF.



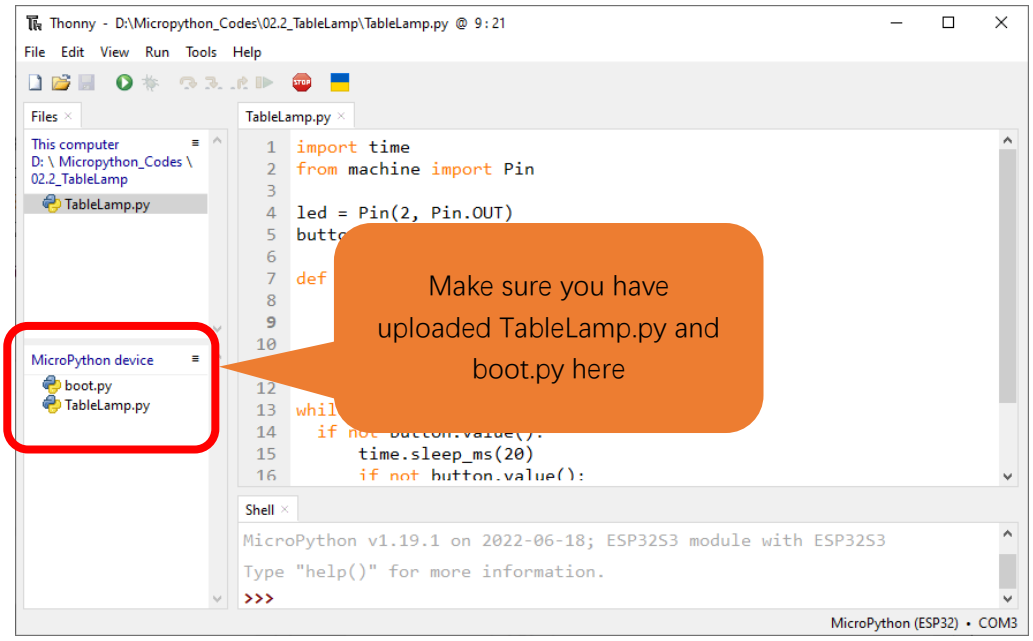
If you have any concerns, please contact us via: support@freenove.com

Upload code to ESP32S3

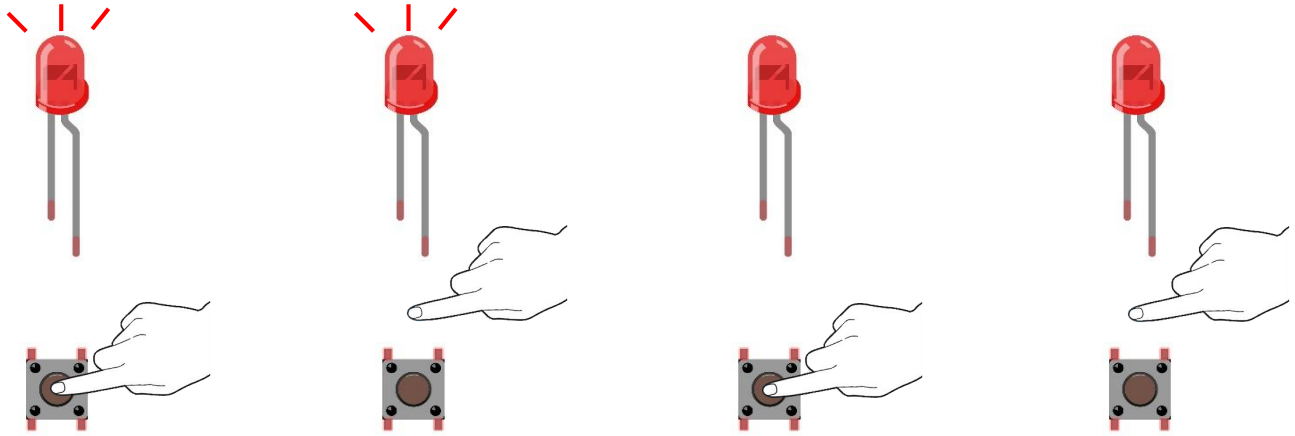
As shown in the following illustration, right-click file 02.2_TableLamp and select "Upload to /" to upload code to ESP32S3.



Upload boot.py in the same way.



Press ESP32S3'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)
```

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)
```

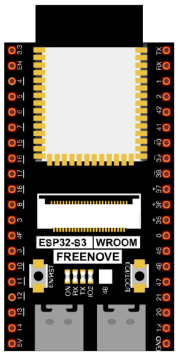
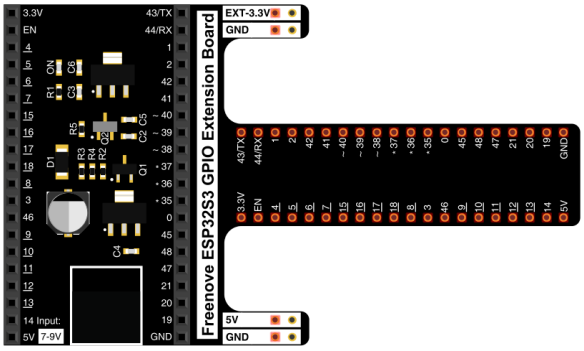
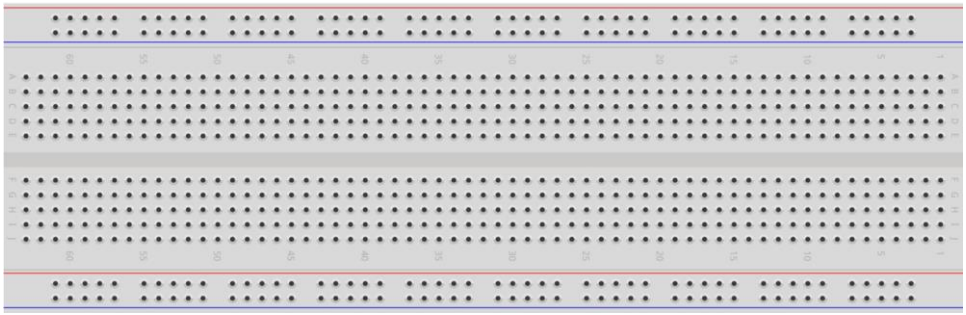
Chapter 3 LED Bar

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

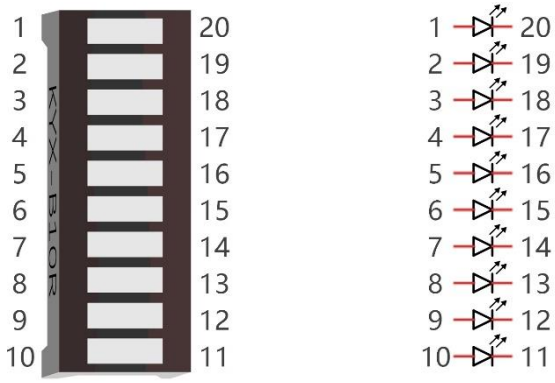
ESP32-S3 WROOM x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x10 	LED bar graph x1 	Resistor 220Ω x10 

Component knowledge

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

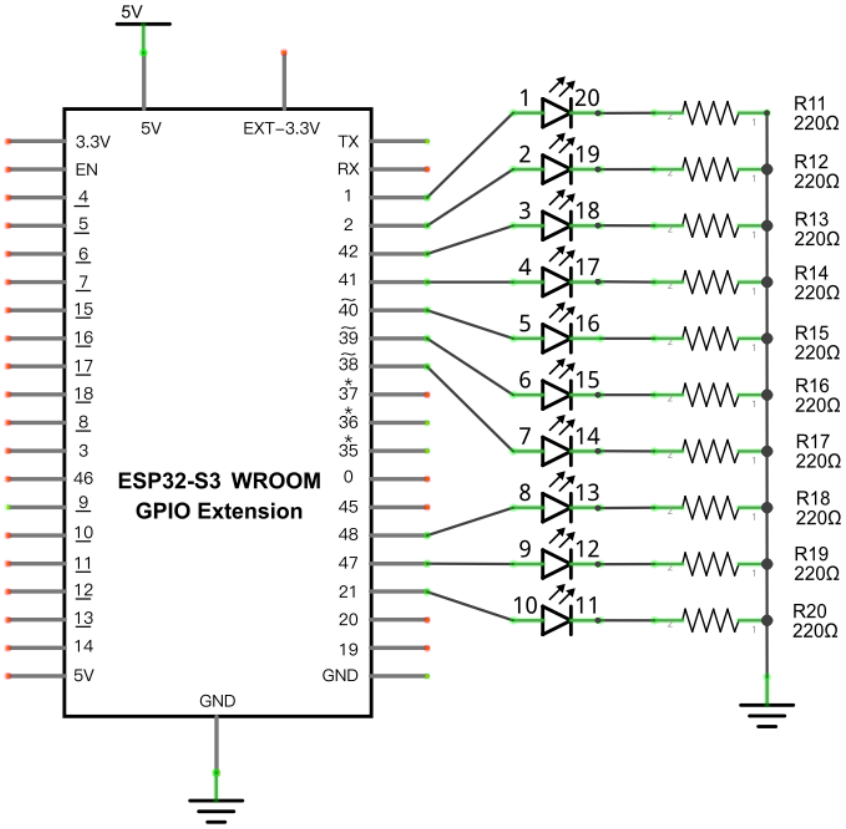
LED bar

A LED bar graph 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.

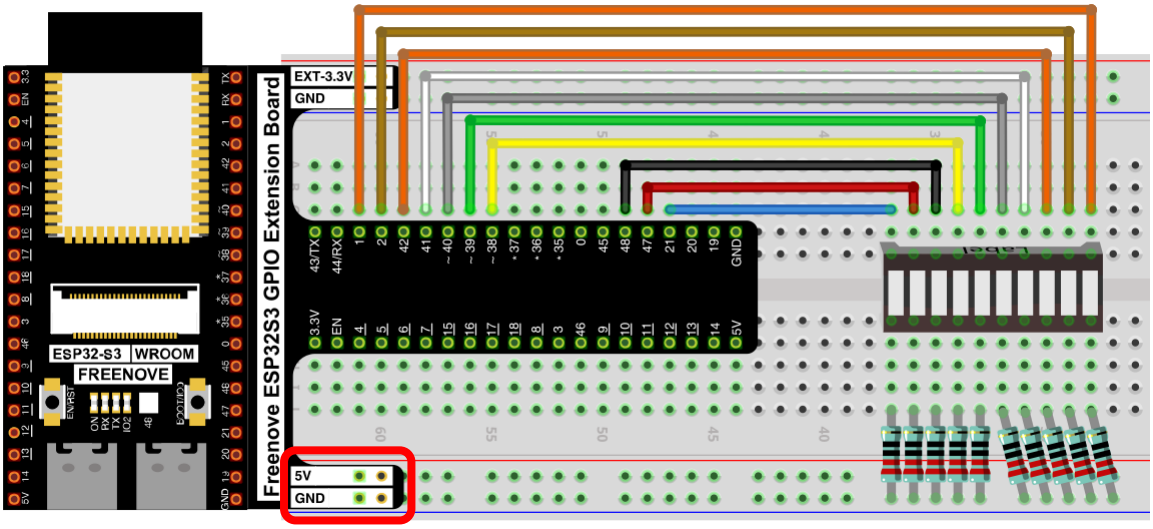


Circuit

Schematic diagram



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



If LED bar does not work, try to rotate it for 180°. The label is random.

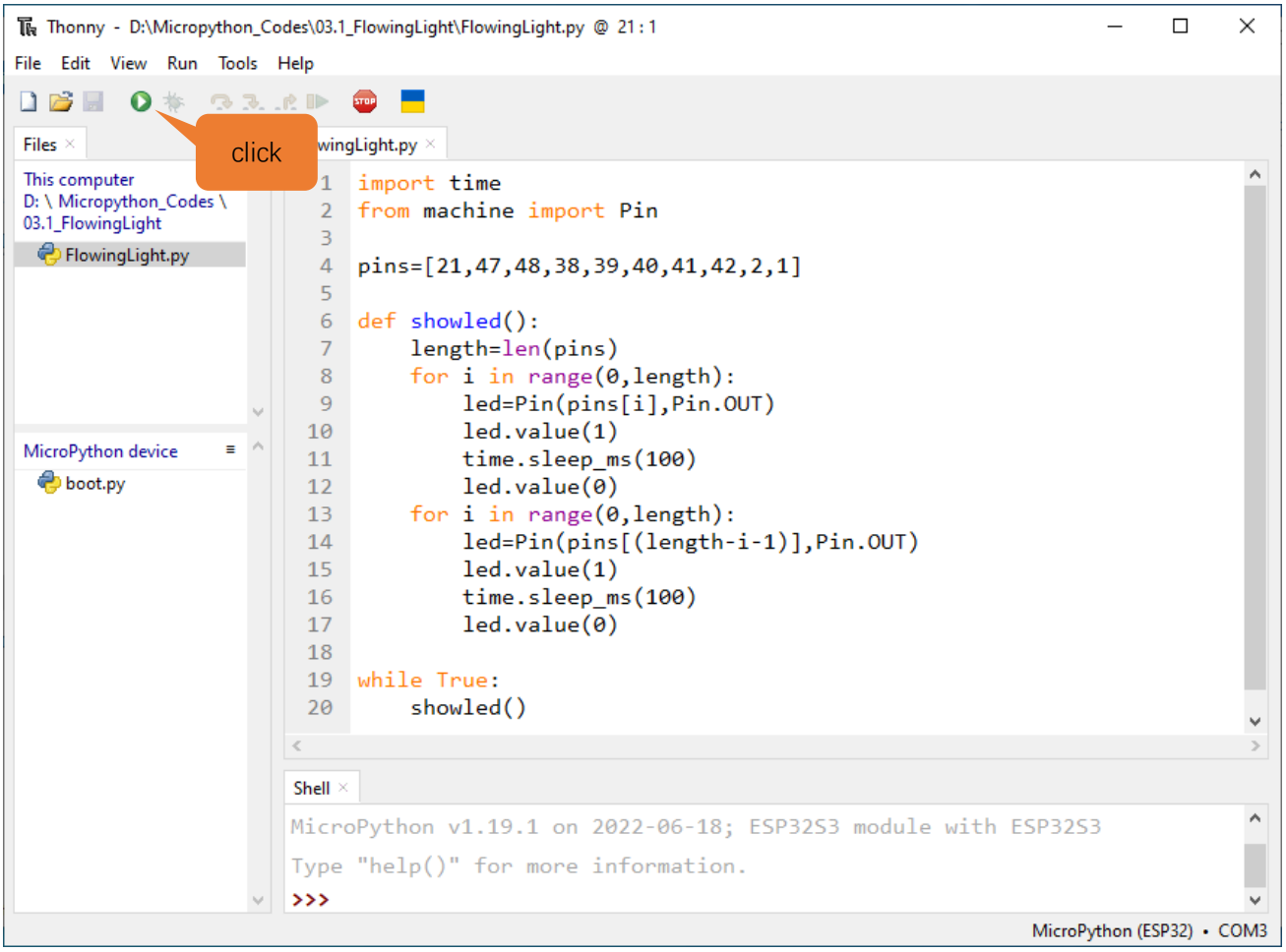
Code

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.

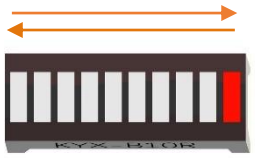
03.1_FlowingLight

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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



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.



The following is the program code:

```
1  import time
2  from machine import Pin
3
4  pins=[21,47,48,38,39,40,41,42,2,1]
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=[21,47,48,38,39,40,41,42,2,1]
```

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

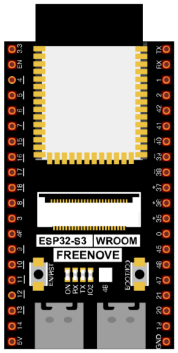
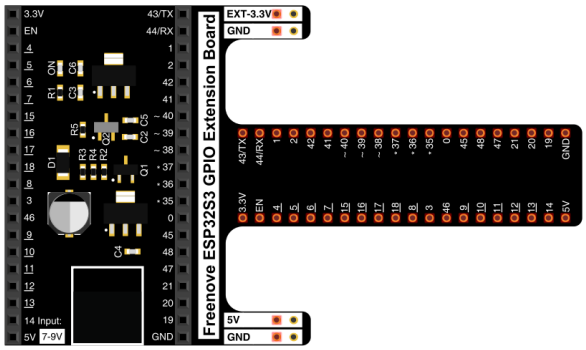
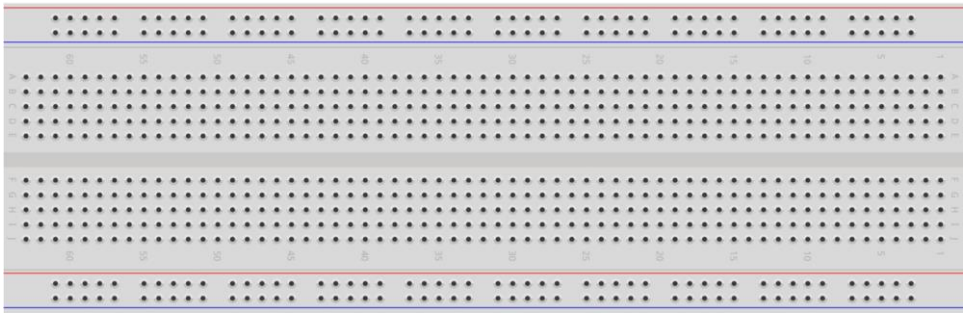
Chapter 4 Analog & PWM

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.

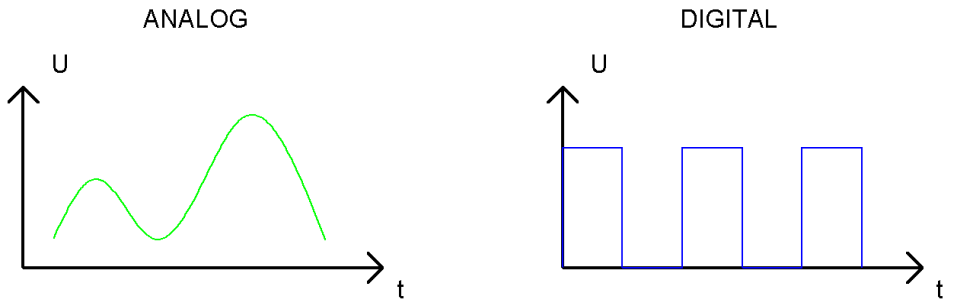
Component List

<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td>GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper M/M x2
  </td></tr>
</table>

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℃ to 10℃. However, digital signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



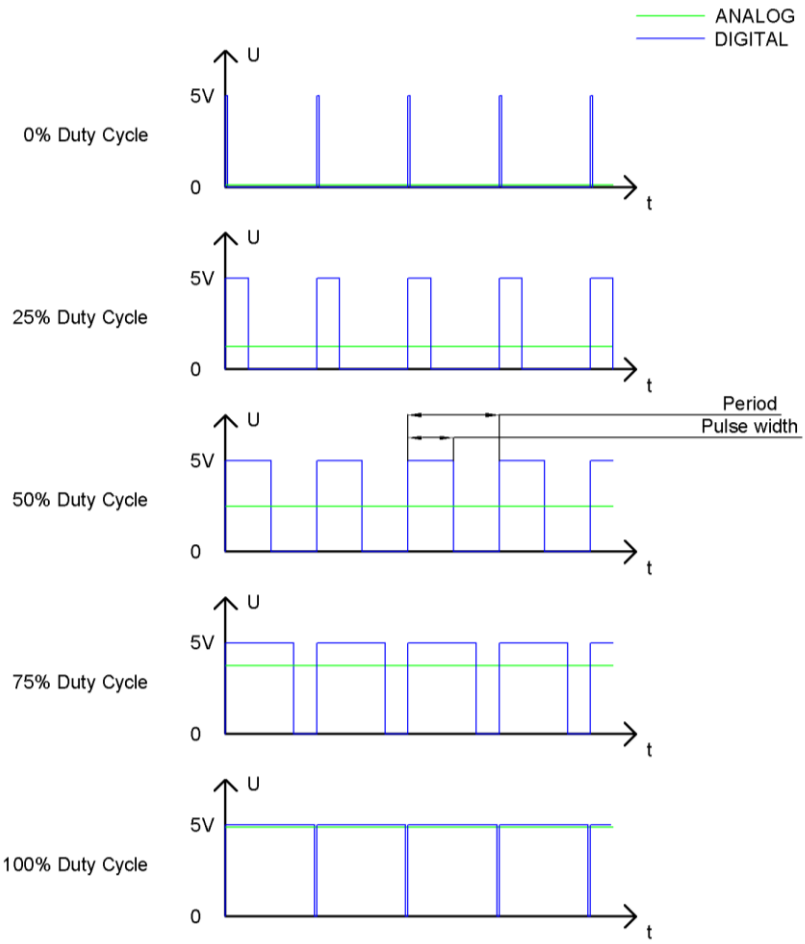
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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 outputs 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 a 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. Therefore, we can control the output power of the LED and other output modules to achieve different effects.

ESP32-S3 and PWM

On ESP32S3, the LEDC(PWM) controller has 8 separate channels, each of which can independently control frequency, duty cycle, and even accuracy. Unlike traditional PWM pins, the PWM output pins of ESP32-S3 are configurable, with one or more PWM output pins per channel. The relationship between the maximum frequency and bit precision is shown in the following formula, where the maximum value of bit is 31.

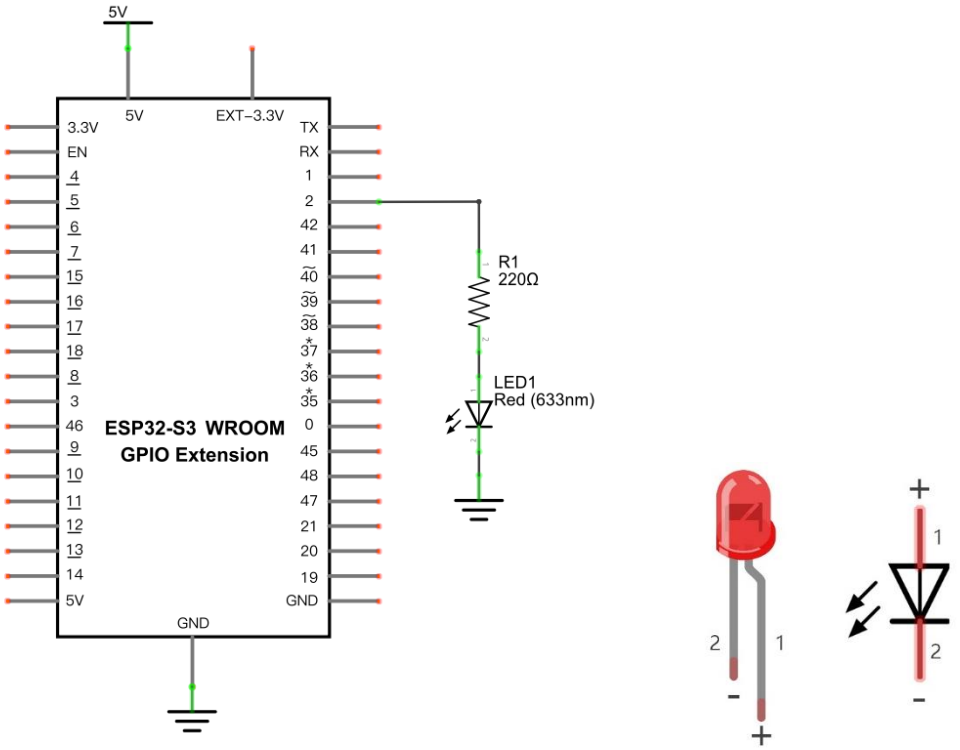
$$Freq_{max} = \frac{80{,}000{,}000}{1 \ll bit}$$

For example, generate a PWM with an 8-bit precision (2^8=256. Values range from 0 to 255) with a maximum frequency of 80,000,000/256 = 312,500Hz.)

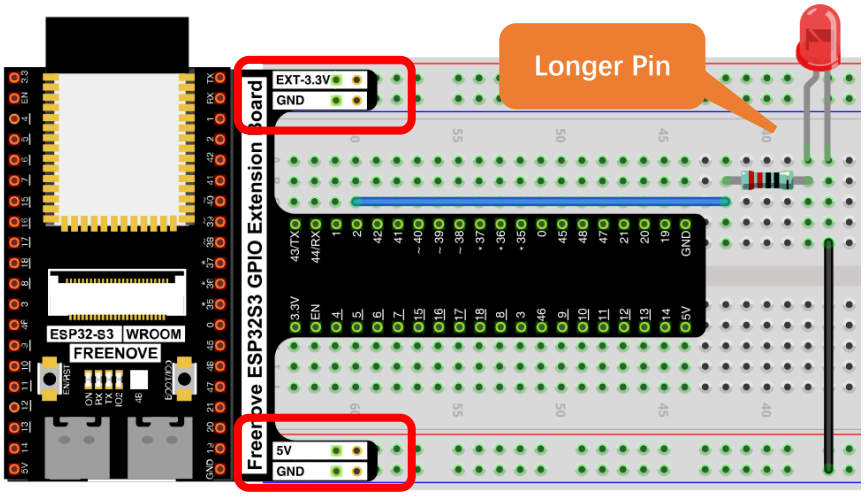
Circuit

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

Schematic diagram



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



Don't rotate ESP32-S3 WROOM 180° for connection.

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.

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

04.1_BreatheLight

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:

```
from machine import Pin,PWM
import time

pwm =PWM(Pin(2,Pin.OUT),10000)
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()
```

The way that the ESP32-S3 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 assign them to PWM.

```
pwm =PWM(Pin(2,Pin.OUT),10000)
```

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%.

```
        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)
```

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.

```
pwm.deinit()
```

Reference

Class PWM(pin, freq)

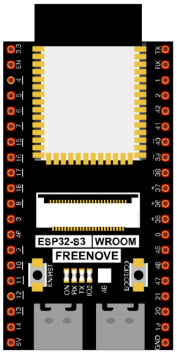
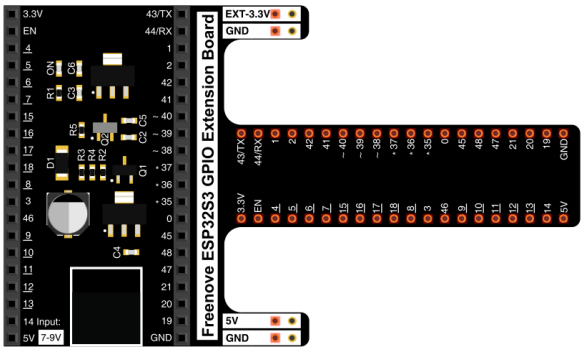
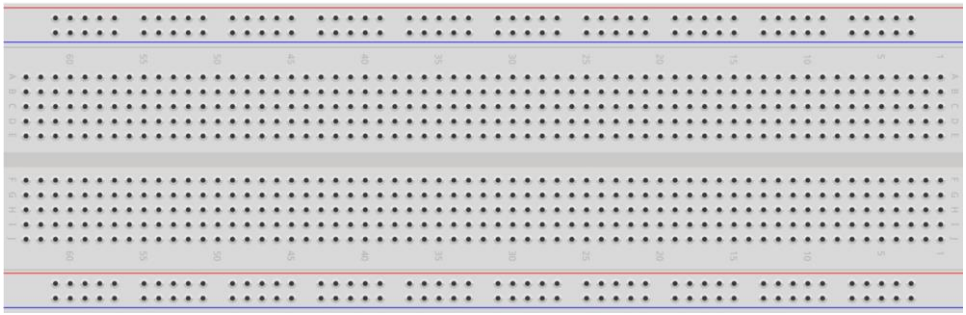
Before each use of PWM module, please add the statement "**from machine import PWM**" to the top of the python file.

pin: Any pin supports PWM output, except GPIO19,20
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

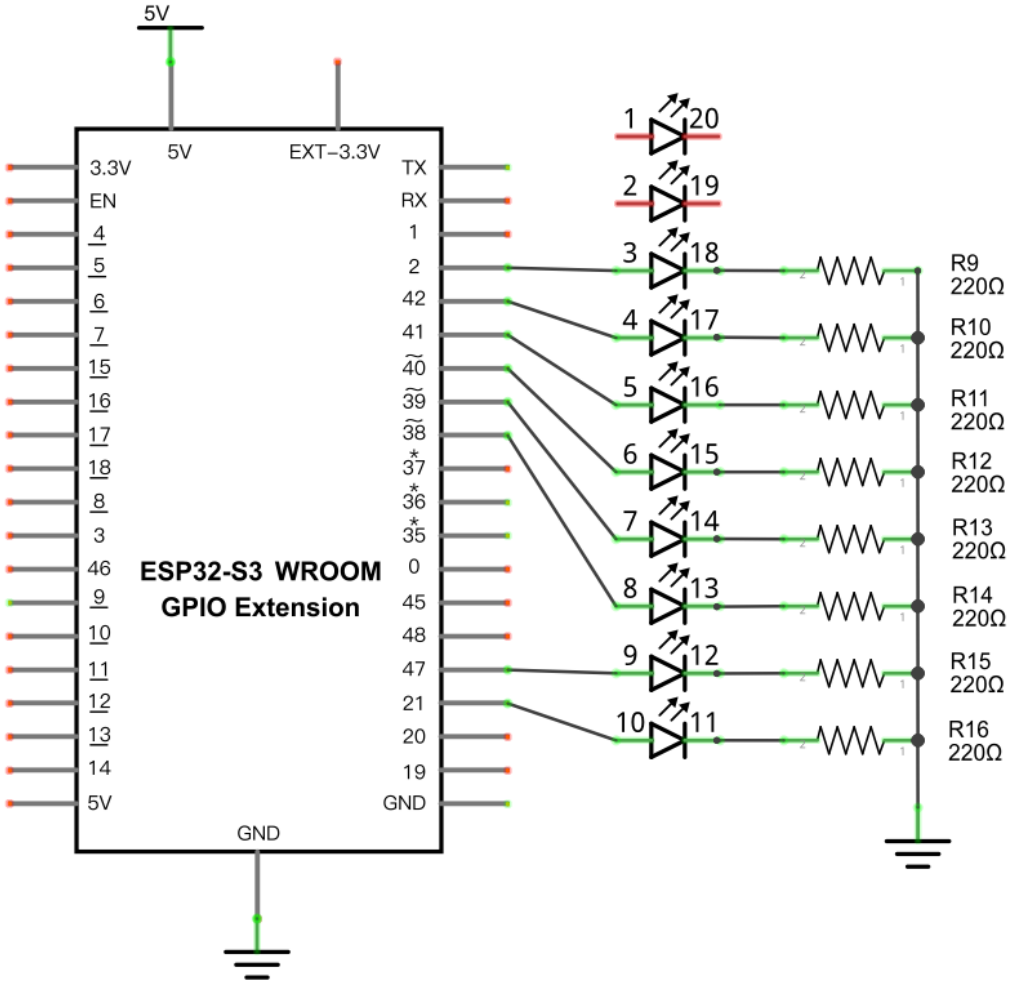
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light. The component list, circuit, and hardware are exactly consistent with the project Flowing Light.

Component List

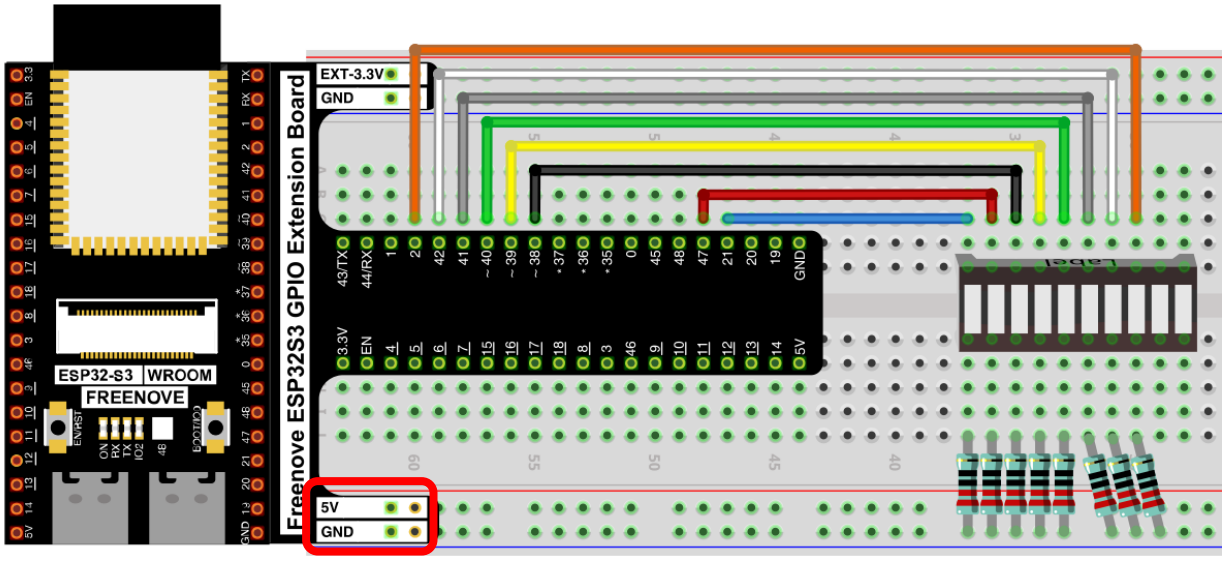
<table>
<tr><td>ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Jumper M/M x8
  </td><td>LED bar graph x1
  </td><td>Resistor 220Ω x8
  </td></tr>
</table>

Circuit

Schematic diagram



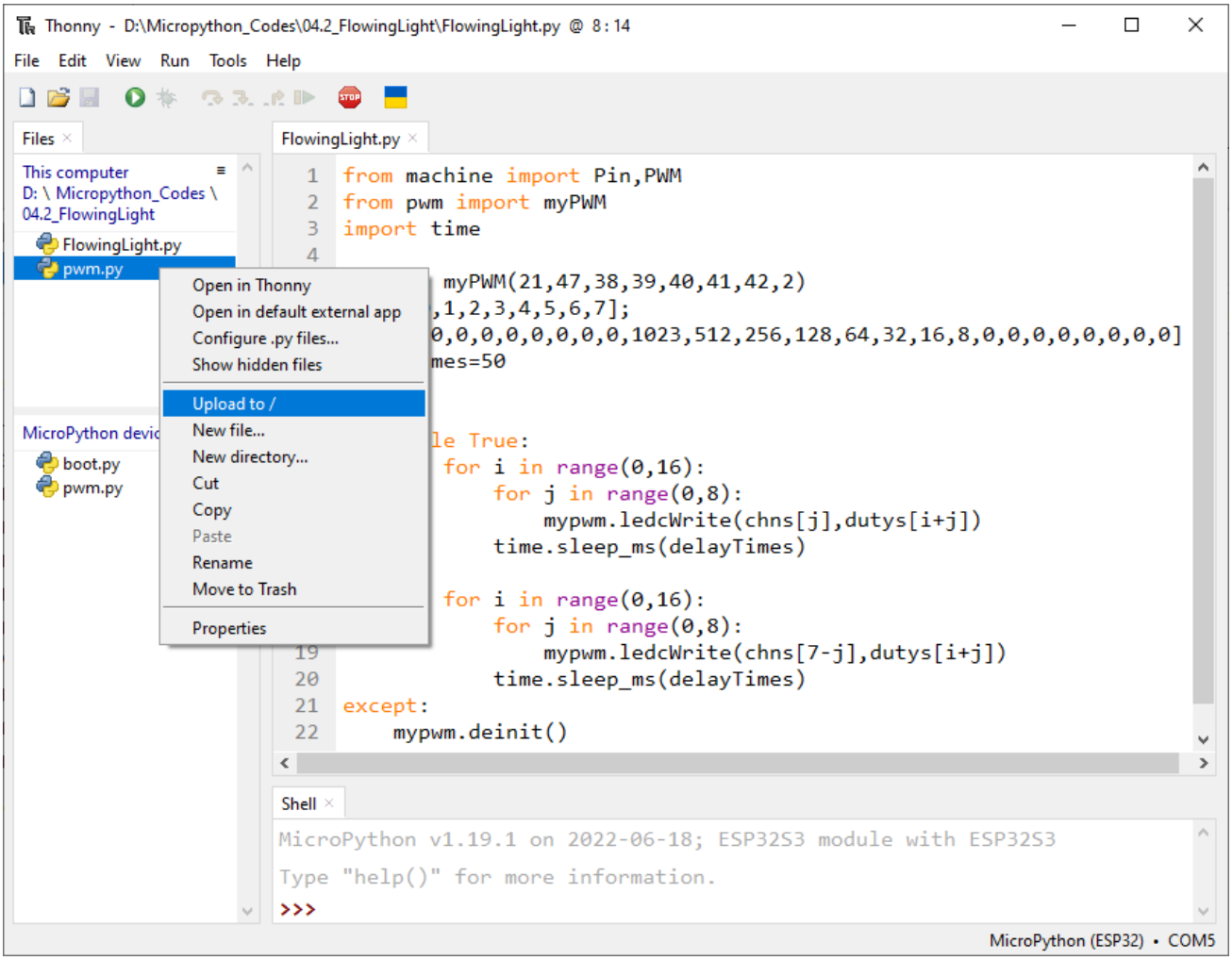
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



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-S3 and then double click "FlowingLight.py".

04.2_FlowingLight



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:

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

mypwm = myPWM(21,47,38,39,40,41,42,2)
chns=[0,1,2,3,4,5,6,7];
dutys=[0,0,0,0,0,0,0,0,1023,512,256,128,64,32,16,8,0,0,0,0,0,0,0,0];
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()
```

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

```
2   from pwm import myPWM
…   …
5   mypwm = myPWM(21,47,38,39,40,41,42,2)
```

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

```
5   mypwm = myPWM(21,47,38,39,40,41,42,2)
6   chns=[0,1,2,3,4,5,6,7];
7   dutys=[0,0,0,0,0,0,0,0,1023,512,256,128,64,32,16,8,0,0,0,0,0,0,0,0];
```

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)
```

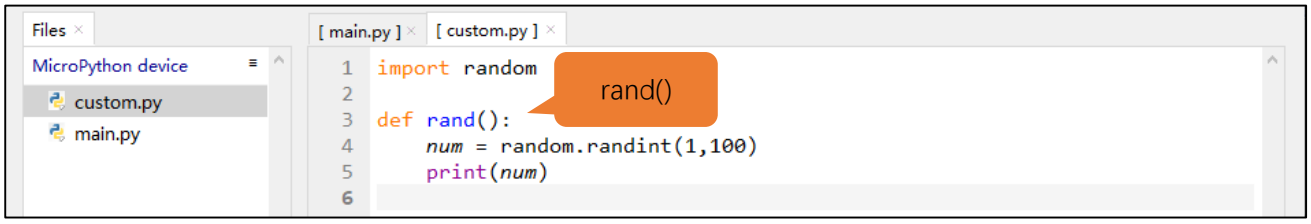
In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row 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	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
d i	0	0	0	0	0	0	0		1023	512	256	128	64	32	16	8	0	0	0	0	0	0	0	0
0																								
1																								
...																								
14																								
15																								
16																								

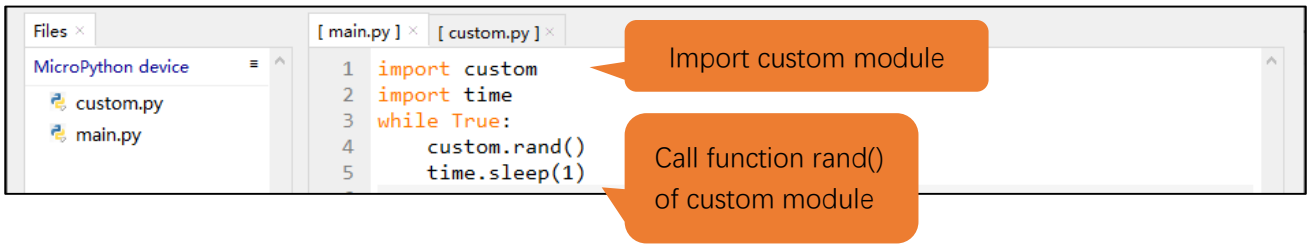
How to import a custom python module

Each Python file, as long as it's stored on the file system of ESP32S3, 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 ESP32S3.



Second, import custom module “custom” to main.py



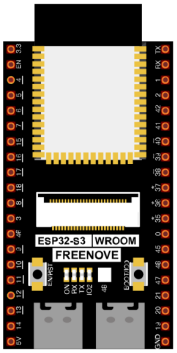
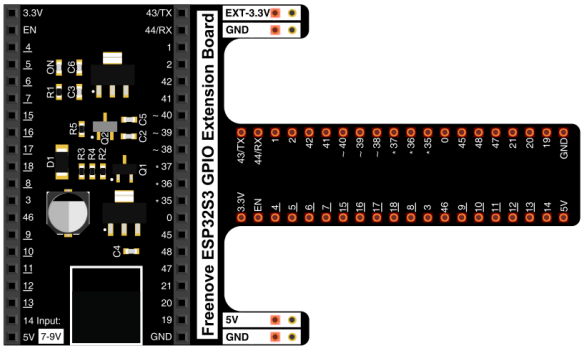
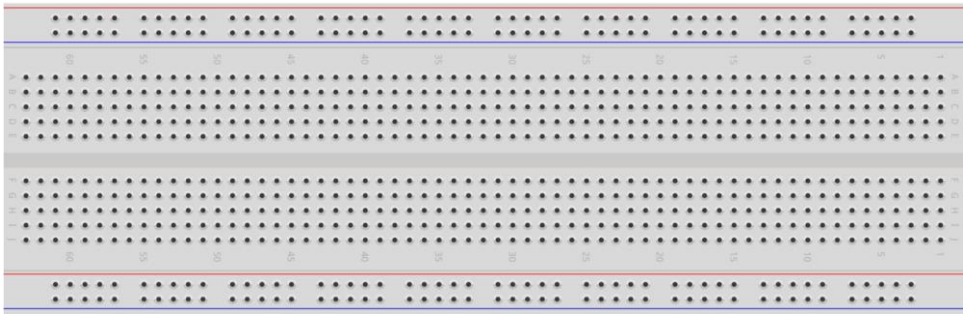
Chapter 5 RGB LED

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

Project 5.1 Random Color Light

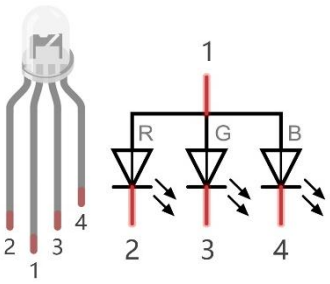
In this project, we will make a multicolored LED. And we can control RGB LED to switch different colors automatically.

Component List

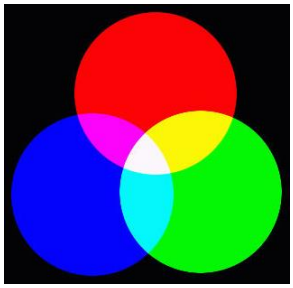
ESP32-S3 WROOM x1 	GPIO Extension Board x1 	
Breadboard x1 		
RGBLED x1 	Resistor 220Ω x3 	Jumper M/M x4 

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 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.

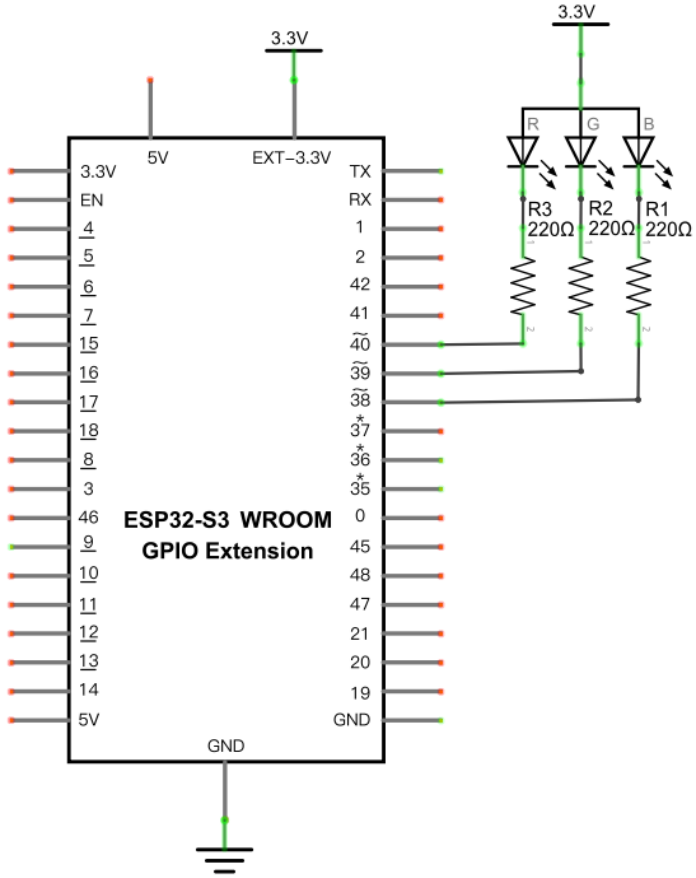


RGB

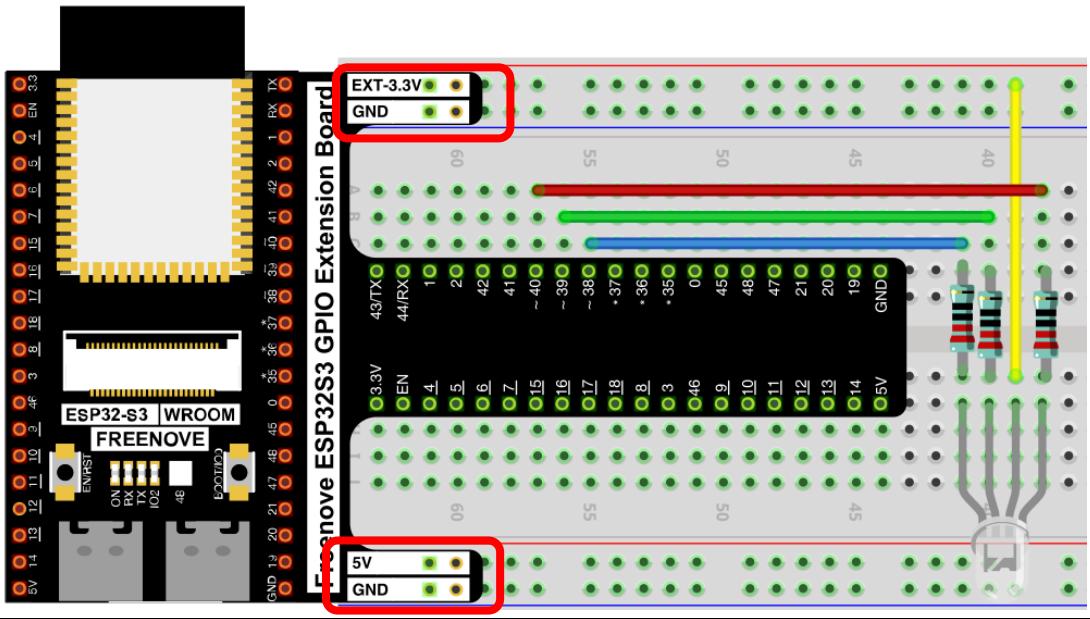
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

Circuit

Schematic diagram



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



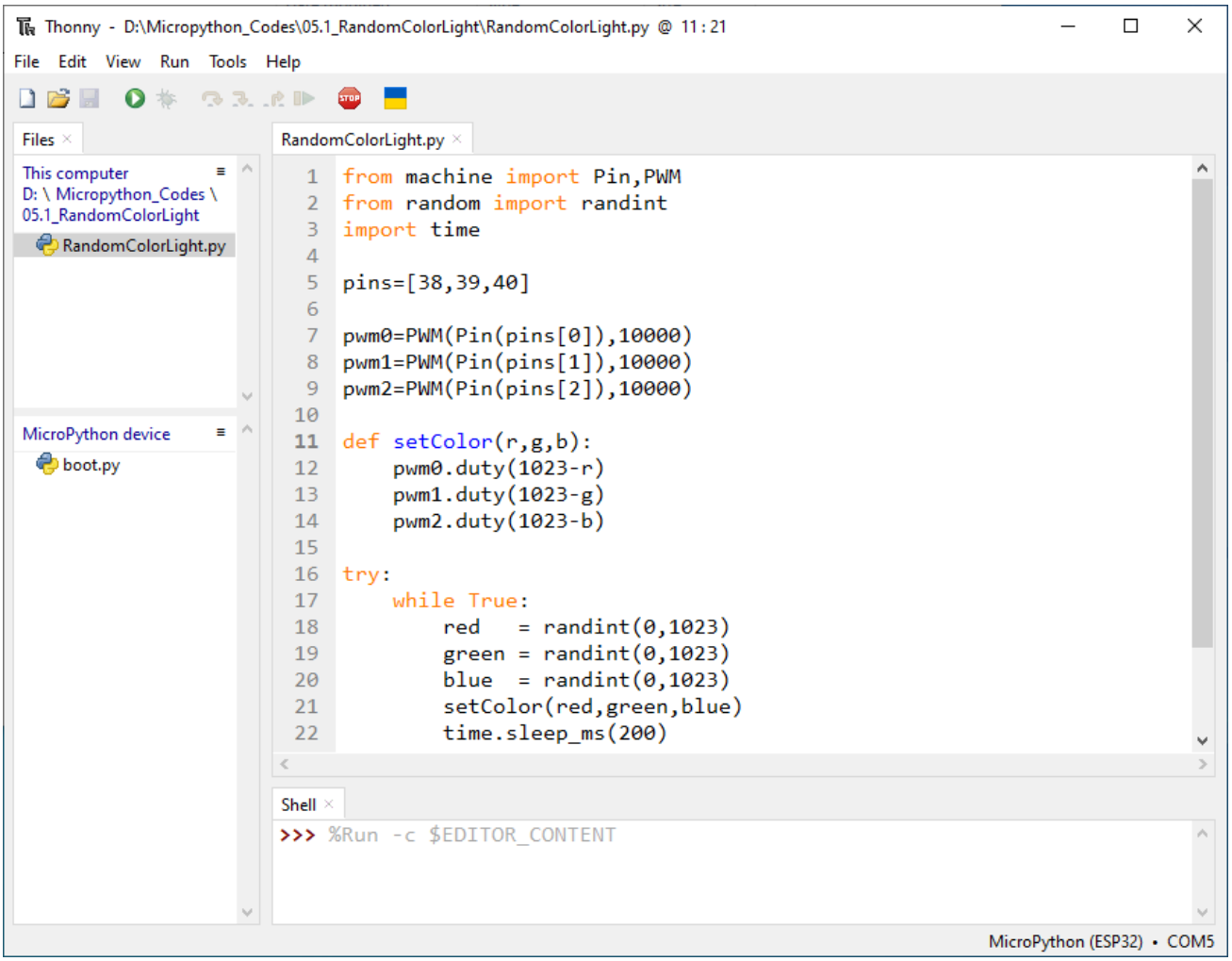
Code

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

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

05.1_RandomColorLight



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:

```
from machine import Pin,PWM
from random import randint
import time

pins=[40,39,38]

pwm0=PWM(Pin(pins[0]),10000)
pwm1=PWM(Pin(pins[1]),10000)
pwm2=PWM(Pin(pins[2]),10000)

def setColor(r,g,b):
    pwm0.duty(1023-r)
    pwm1.duty(1023-g)
    pwm2.duty(1023-b)

try:
    while True:
        red   = randint(0,1023)
        green = randint(0,1023)
        blue  = randint(0,1023)
        setColor(red,green,blue)
        time.sleep_ms(200)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

Import Pin, PWM and Randon Function modules.

```
from machine import Pin,PWM
from random import randint
import time
```

Configure ouput mode of GPIO38, GPIO39 and GPIO40 as PWM output and PWM frequency as 10000Hz

```
pins=[40,39,38]

pwm0=PWM(Pin(pins[0]),10000)
pwm1=PWM(Pin(pins[1]),10000)
pwm2=PWM(Pin(pins[2]),10000)
```

Define a function to set the color of RGBLED.

```
def setColor(r,g,b):
    pwm0.duty(1023-r)
    pwm1.duty(1023-g)
    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.unifrom(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 RGB LED, but the random display of colors is rather stiff. This project will realize a fashionable light with soft color changes.

Component list and the circuit are exactly the same as the 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" → "05.2_GradientColorLight" and double click "GradientColorLight.py".

05.2_GradientColorLight

The following is the program code:

```
from machine import Pin,PWM
import time

pins=[40,39,38];

pwm0=PWM(Pin(pins[0]),1000)
pwm1=PWM(Pin(pins[1]),1000)
pwm2=PWM(Pin(pins[2]),1000)

red=0                   #red
green=0                 #green
blue=0                  #blue

def setColor():
    pwm0.duty(red)
    pwm1.duty(green)
    pwm2.duty(blue)

def wheel(pos):
    global red,green,blue
    WheelPos=pos%1023
    print(WheelPos)
    if WheelPos<341:
        red=1023-WheelPos*3
        green=WheelPos*3
```

```
        blue=0

    elif WheelPos>=341 and WheelPos<682:
        WheelPos -= 341;
        red=0
        green=1023-WheelPos*3
        blue=WheelPos*3
    else :
        WheelPos -= 682;
        red=WheelPos*3
        green=0
        blue=1023-WheelPos*3

try:
    while True:
        for i in range(0,1023):
            wheel(i)
            setColor()
            time.sleep_ms(15)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

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.

```
def wheel(pos):
    global red,green,blue
    WheelPos=pos%1023
    print(WheelPos)
    if WheelPos<341:
        red=1023-WheelPos*3
        green=WheelPos*3
        blue=0

    elif WheelPos>=341 and WheelPos<682:
        WheelPos -= 341;
        red=0
        green=1023-WheelPos*3
        blue=WheelPos*3
    else :
        WheelPos -= 682;
        red=WheelPos*3
        green=0
        blue=1023-WheelPos*3
```

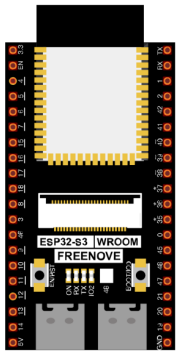
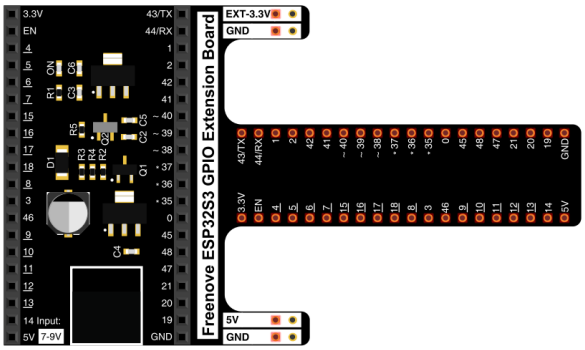
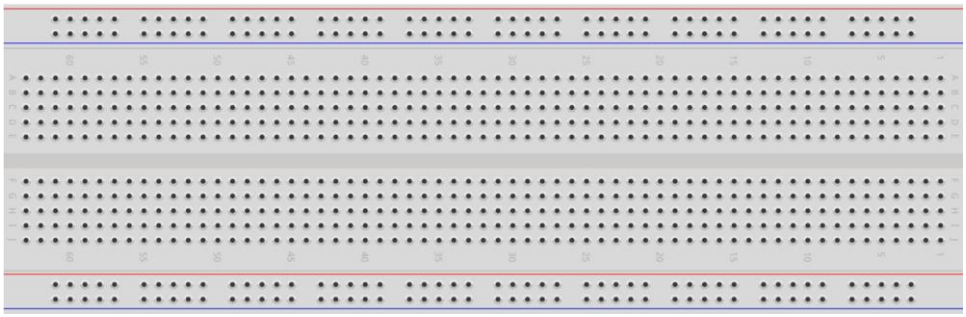
Chapter 6 Buzzer

In this chapter, we will learn about buzzers that can make sounds.

Project 6.1 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

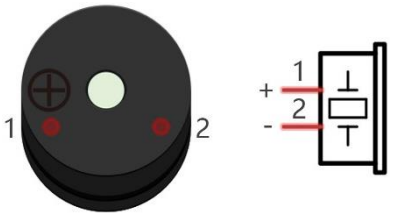
<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td colspan="3">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td colspan="5">Jumper M/M x6
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

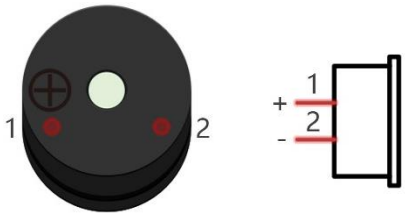
Buzzer

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



Passive buzzer



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.

Active buzzer

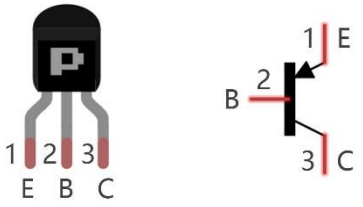
Passive buzzer

Transistor

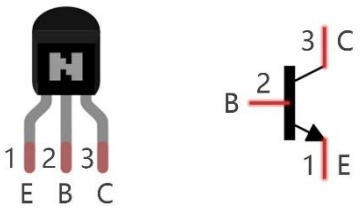
Because the buzzer requires such large current that GPIO of ESP32-S3 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

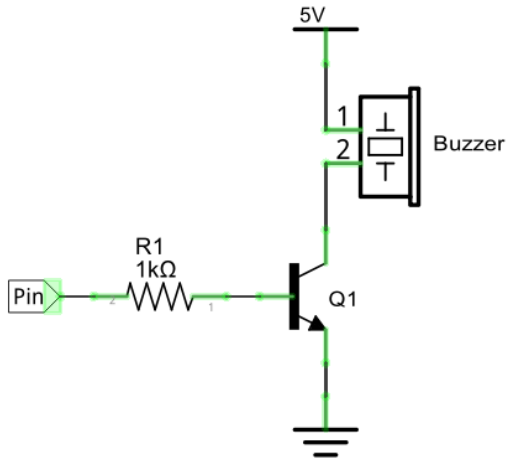
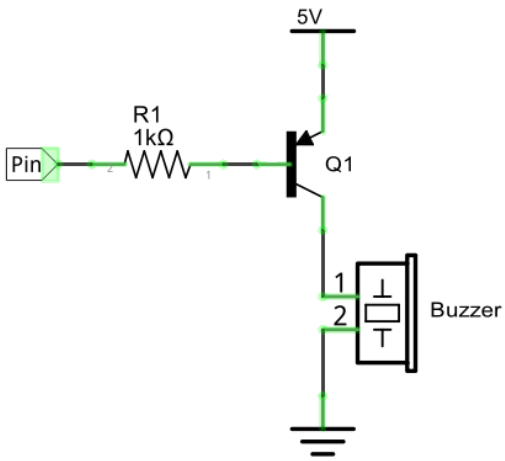


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

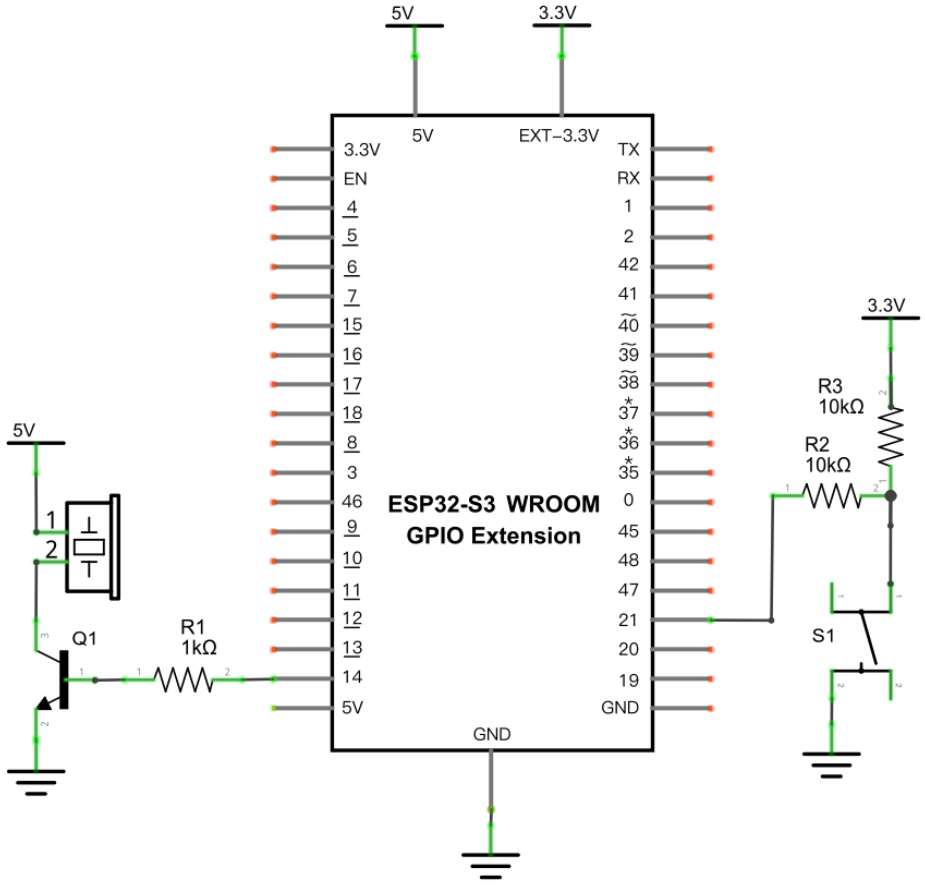
When use NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

When use 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.

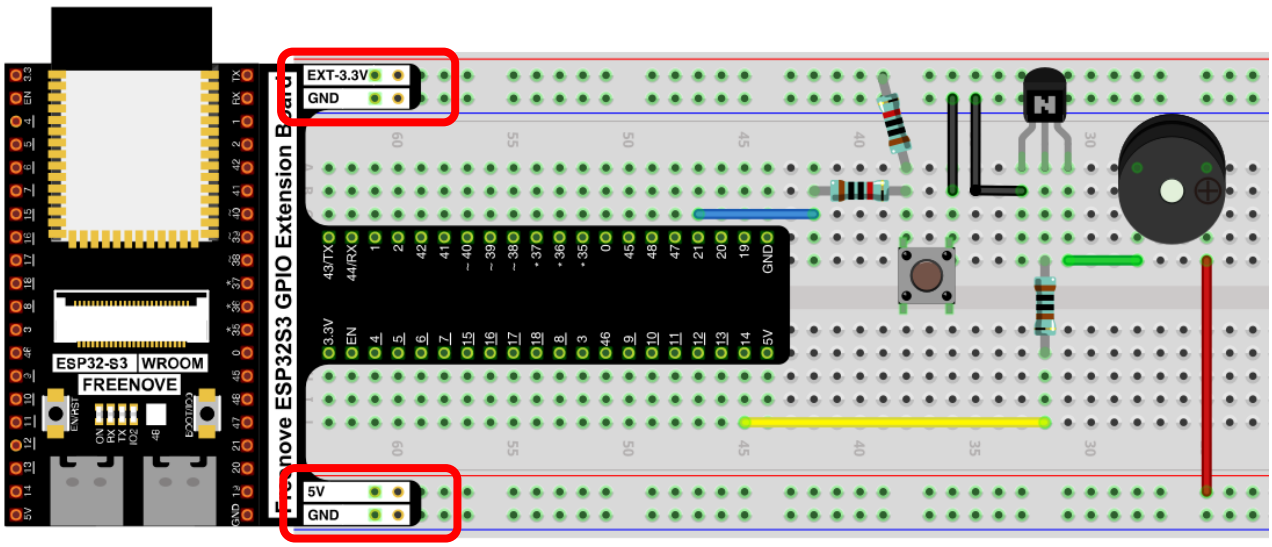
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

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.

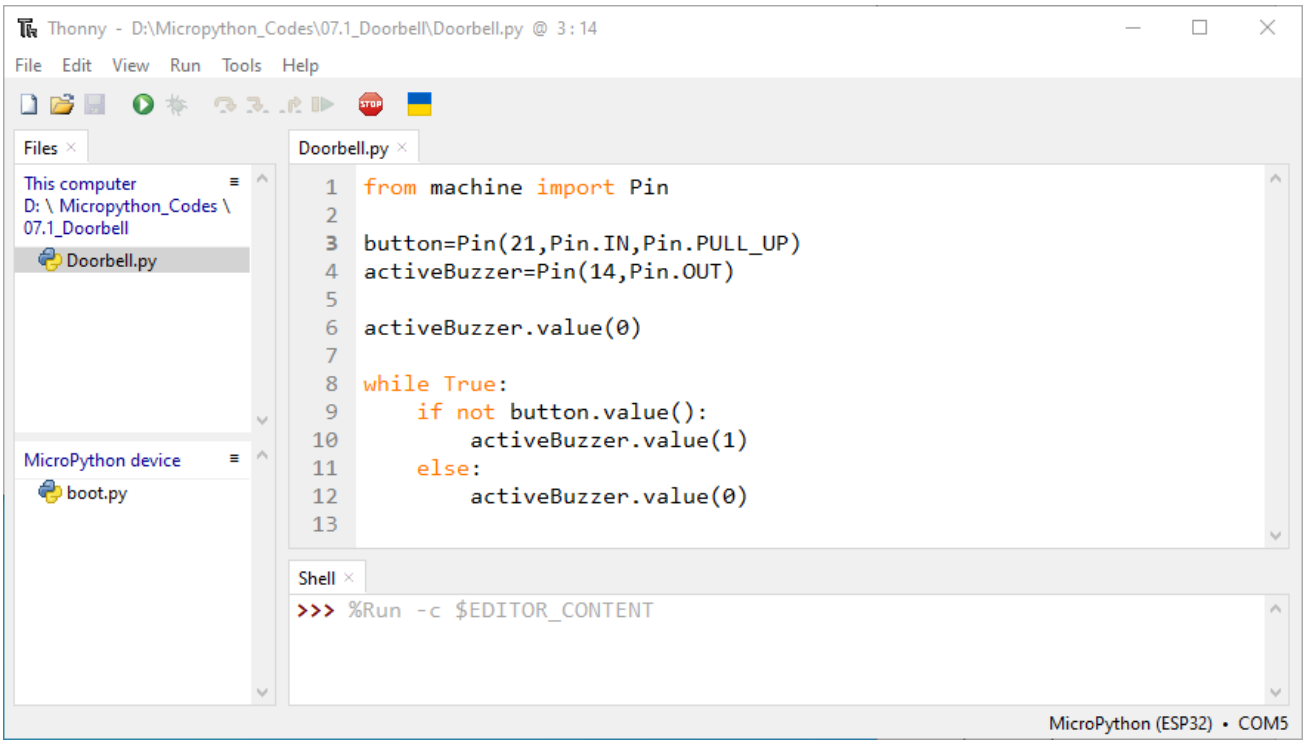
Code

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.

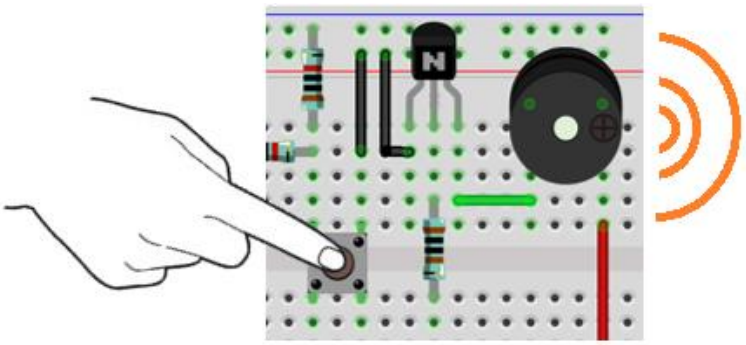
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "06.1_Doorbell" and double click "Doorbell.py".

06.1_Doorbell



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:

```
from machine import Pin

button=Pin(21,Pin.IN,Pin.PULL_UP)
activeBuzzer=Pin(14,Pin.OUT)

activeBuzzer.value(0)

while True:
    if not button.value():
        activeBuzzer.value(1)
    else:
        activeBuzzer.value(0)
```

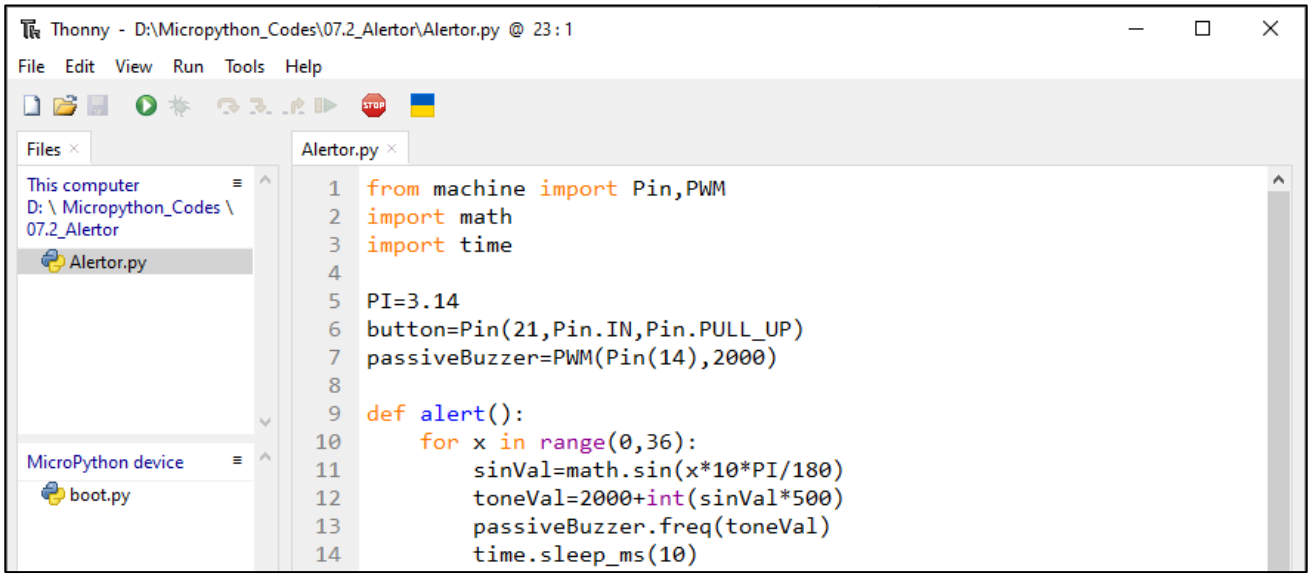
The code is logically the same as using button to control LED.

Project 6.2 Alertor

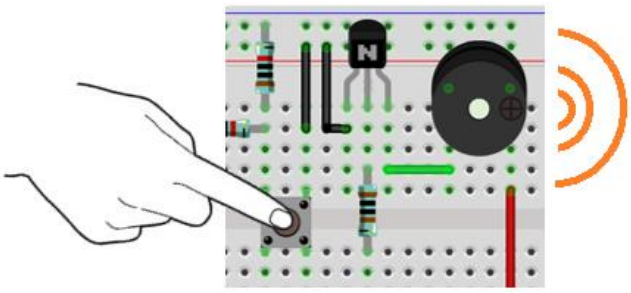
Next, we will use a passive buzzer to make an alarm.
Component list and the circuit is similar to the last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer.**

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" → "06.2_Alertor", and double click "Alertor.py".
06.2_Alertor



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
```

```
button=Pin(21,Pin.IN,Pin.PULL_UP)
passiveBuzzer=PWM(Pin(14),2000)

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():
            passiveBuzzer.init()
            alert()
        else:
            passiveBuzzer.deinit()
except:
    passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

```
from machine import Pin,PWM
import math
import time
```

Define the pins of the button and passive buzzer.

```
PI=3.14
button=Pin(21,Pin.IN,Pin.PULL_UP)
passiveBuzzer=PWM(Pin(14),2000,512)
```

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

```
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)
```

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

```
passiveBuzzer.deinit()
```

Reference

```
double ledcWriteTone(uint8_t channel, double freq);
```

This updates the tone frequency value on the given channel.

This function has some bugs in the current version (V1.0.4): when the call interval is less than 20ms, the resulting PWM will have an exception. We will get in touch with the authorities to solve this problem and give solutions in the following two projects.

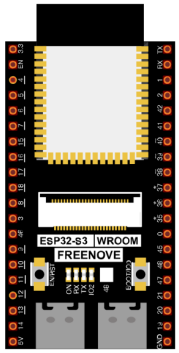
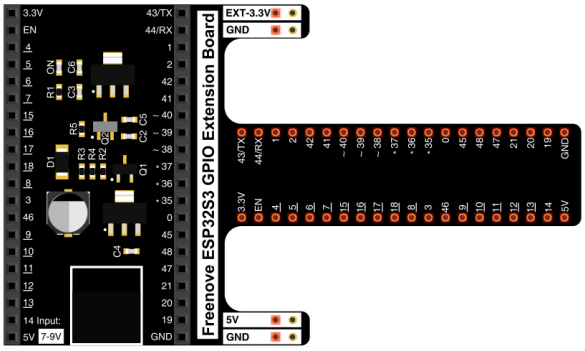
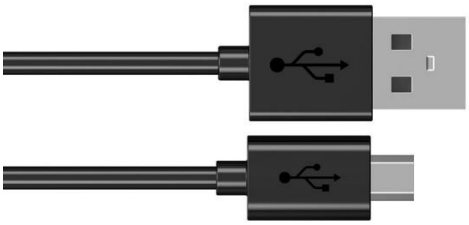
Chapter 7 Serial Communication

Serial Communication is a means of communication between different devices/devices. This section describes ESP32-S3's Serial Communication.

Project 7.1 Serial Print

This project uses ESP32-S3's serial communicator to send data to the computer and print it on the serial monitor.

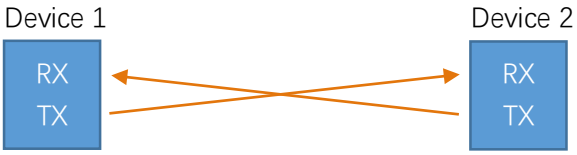
Component List

ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Micro USB Wire x1 	

Related knowledge

Serial communication

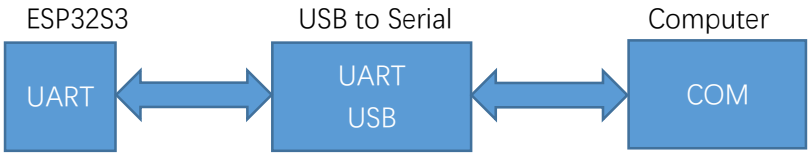
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections 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 is 9600 and 115200.

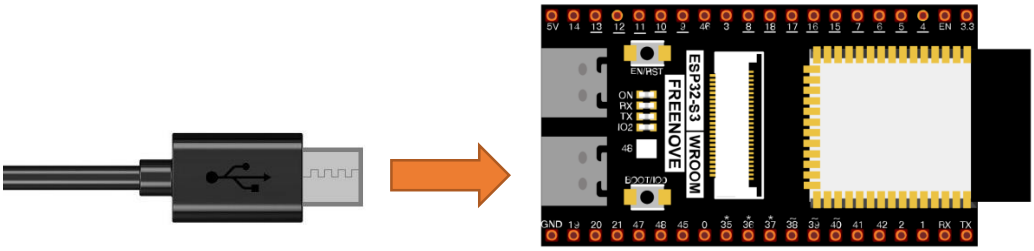
Serial port on ESP32S3

Freenove ESP32-S3 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Circuit

Connect Freenove ESP32-S3 to the computer with USB cable.

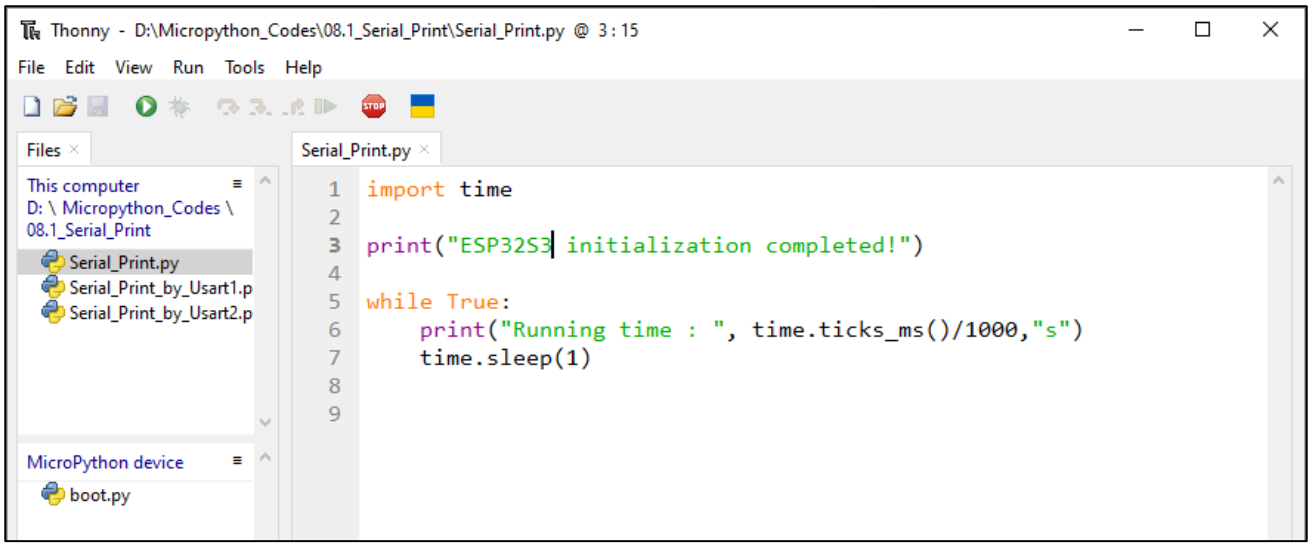


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "07.1_Serial_Print" and double "Serial_Print.py".

07.1_Serial_Print

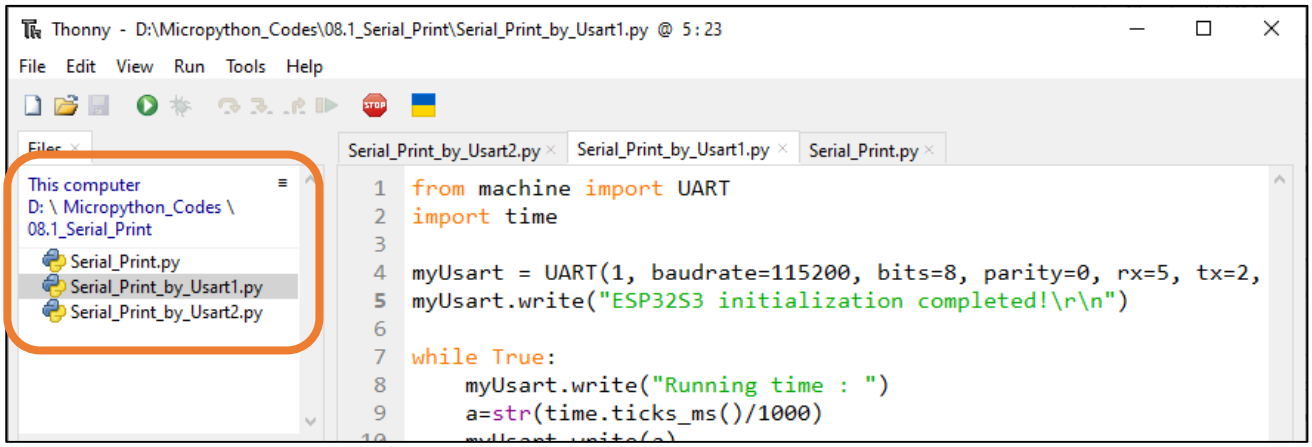


Click "Run current script" and observe the changes of "Shell", which will display the time when ESP32-S3 is powered on once per second.

The following is the program code:

```
1  import time
2
3  print("ESP32-S3 initialization completed!")
4
5  while True:
6      print("Running time : ", time.ticks_ms()/1000, "s")
7      time.sleep(1)
```

ESP32-S3 has 3 serial ports, one of which is used as REPL, that is, Pin(43) and Pin(44) 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: UAPT's reading and writing pins

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 ≤ 0x7FFF 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.

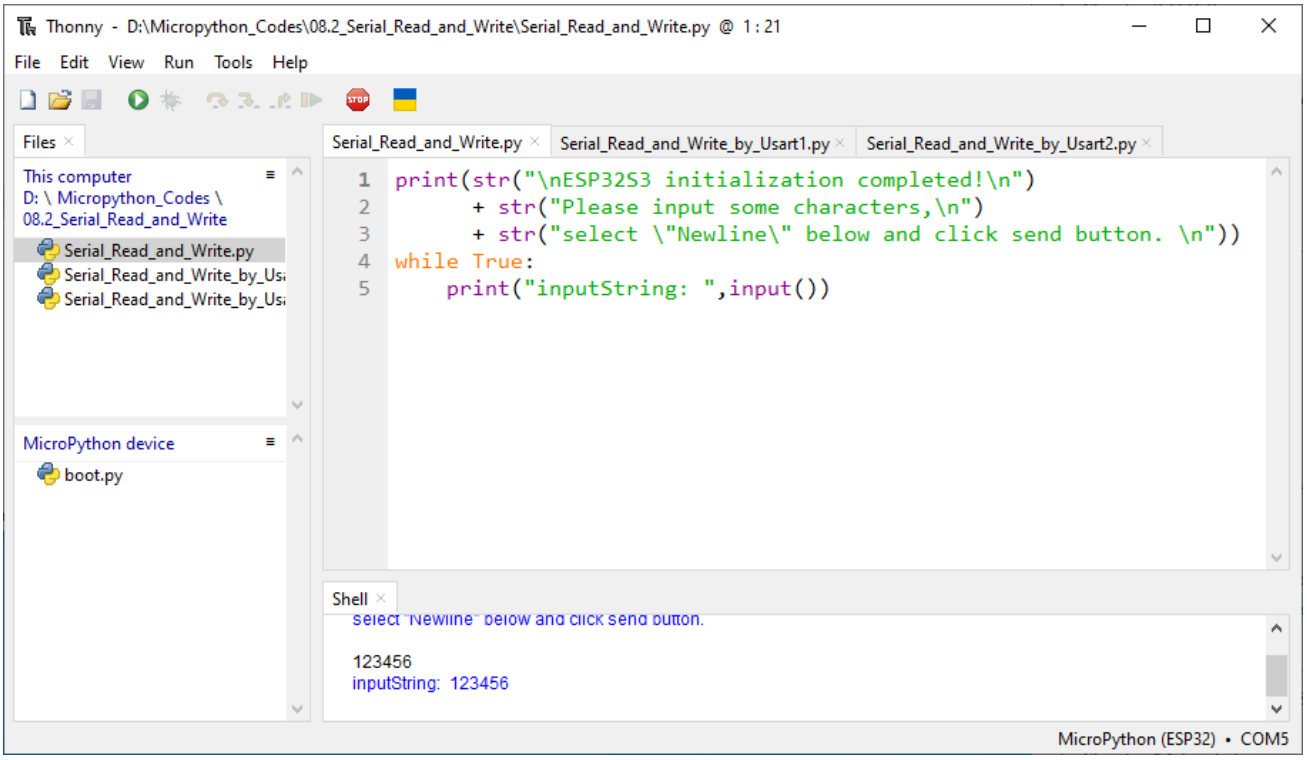
Project 7.2 Serial Read and Write

From last section, we use serial port on Freenove ESP32-S3 to send data to a computer, now we will use that to receive data from computer.

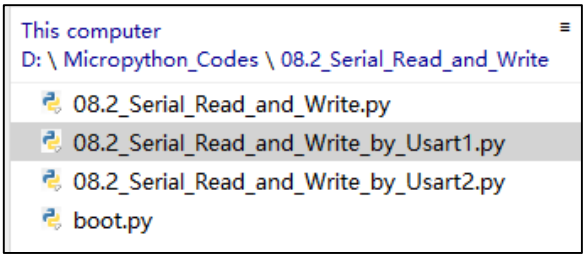
Component and circuit are the same as in the previous project.

Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "07.2_Serial_Read_and_Write" and double click "Serial_Read_and_Write.py".
07.2_Serial_Read_and_Write



Click "Run current script" and ESP32-S3 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.



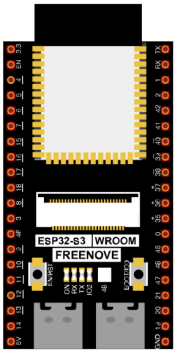
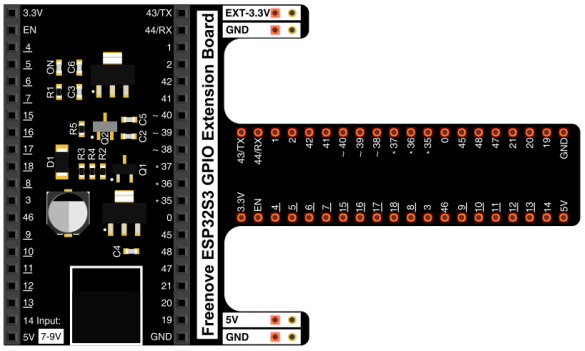
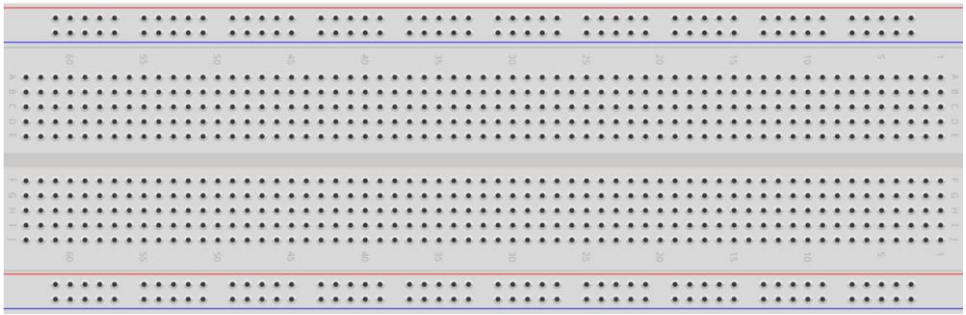
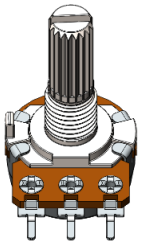
Chapter 8 AD Converter

Signals in life are divided into digital signals and analog signals. In this chapter, we will learn how to use ESP32-S3 to read analog signals.

Project 8.1 Read the Voltage of Potentiometer

In this project, we will use the ADC function of ESP32-S3 to read the voltage value of the potentiometer and print it through the serial port monitor.

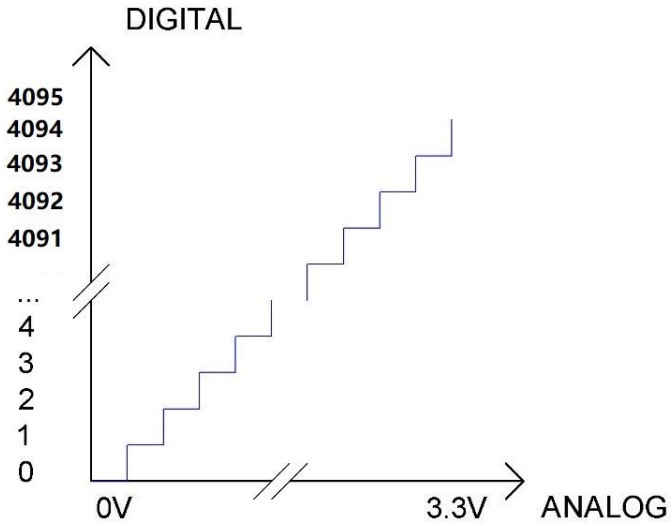
Component List

ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
Rotary potentiometer x1 	Jumper M/M x3 

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-S3 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 rage 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 rang of 0V---3.3/4095 V corresponds to digital 0;
Subsection 2: the analog in rang 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:

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

ADC on ESP32S3

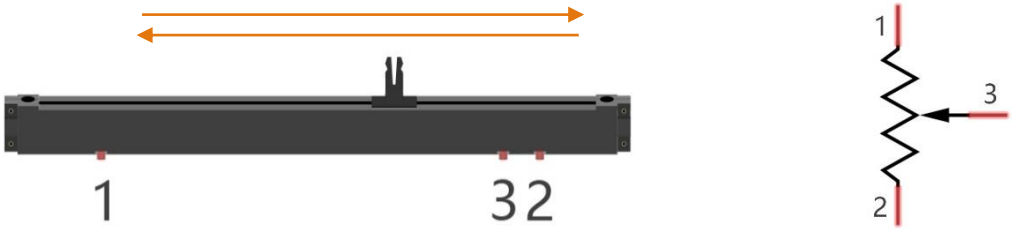
ESP32-S3 has two digital analog converters with successive approximations of 12-bit accuracy, and a total of 18 pins can be used to measure analog signals. GPIO pin sequence number and analog pin definition are shown in the following table.

GPIO number	ADC channel
GPIO 1	ADC1_CH0
GPIO 2	ADC1_CH1
GPIO 3	ADC1_CH2
GPIO 4	ADC1_CH3
GPIO 5	ADC1_CH4
GPIO 6	ADC1_CH5
GPIO 7	ADC1_CH6
GPIO 8	ADC1_CH7
GPIO 9	ADC1_CH8
GPIO 10	ADC1_CH9
GPIO 11	ADC2_CH0
GPIO 12	ADC2_CH1
GPIO 13	ADC2_CH2
GPIO 14	ADC2_CH3
GPIO 15	ADC2_CH4
GPIO 16	ADC2_CH5
GPIO 17	ADC2_CH6
GPIO 18	ADC2_CH7

Component knowledge

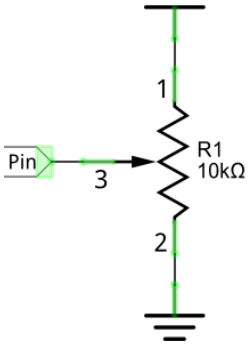
Potentiometer

A potentiometer is a three-terminal resistor. 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.



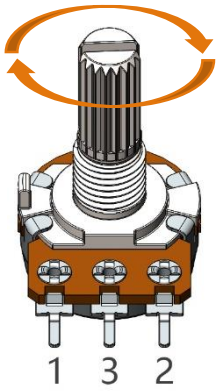
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

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.



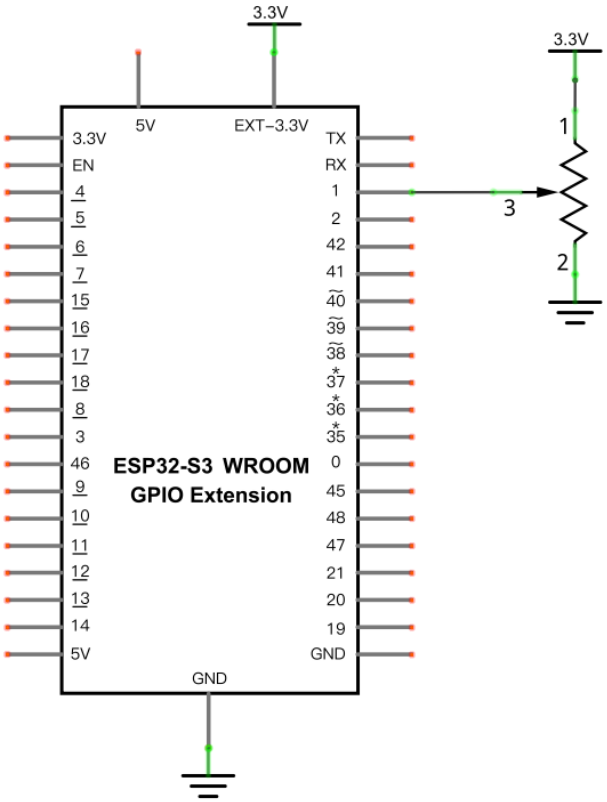
Rotary potentiometer

Rotary potentiometer and linear potentiometer have similar function; their only difference is: the resistance is adjusted by rotating the potentiometer.

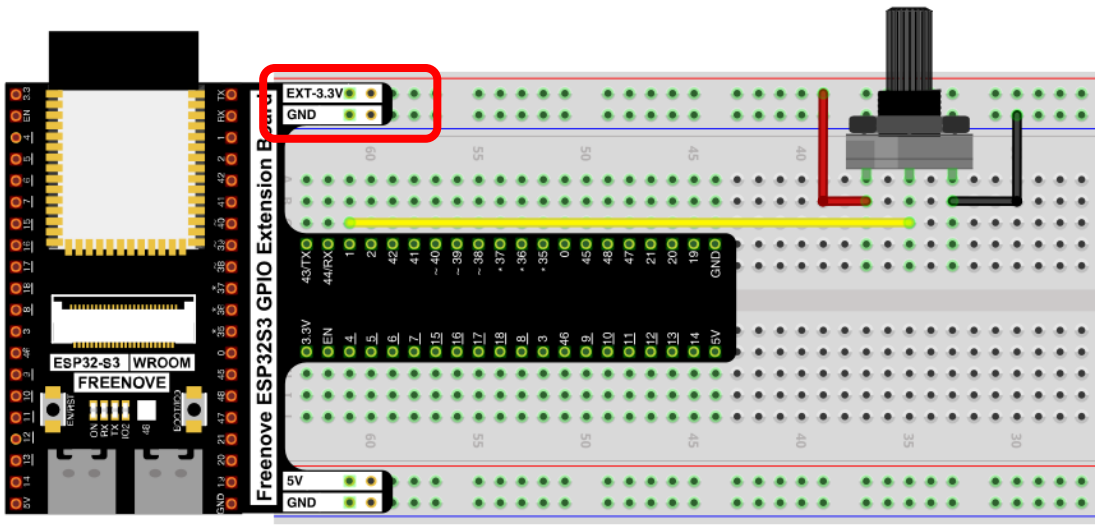


Circuit

Schematic diagram



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

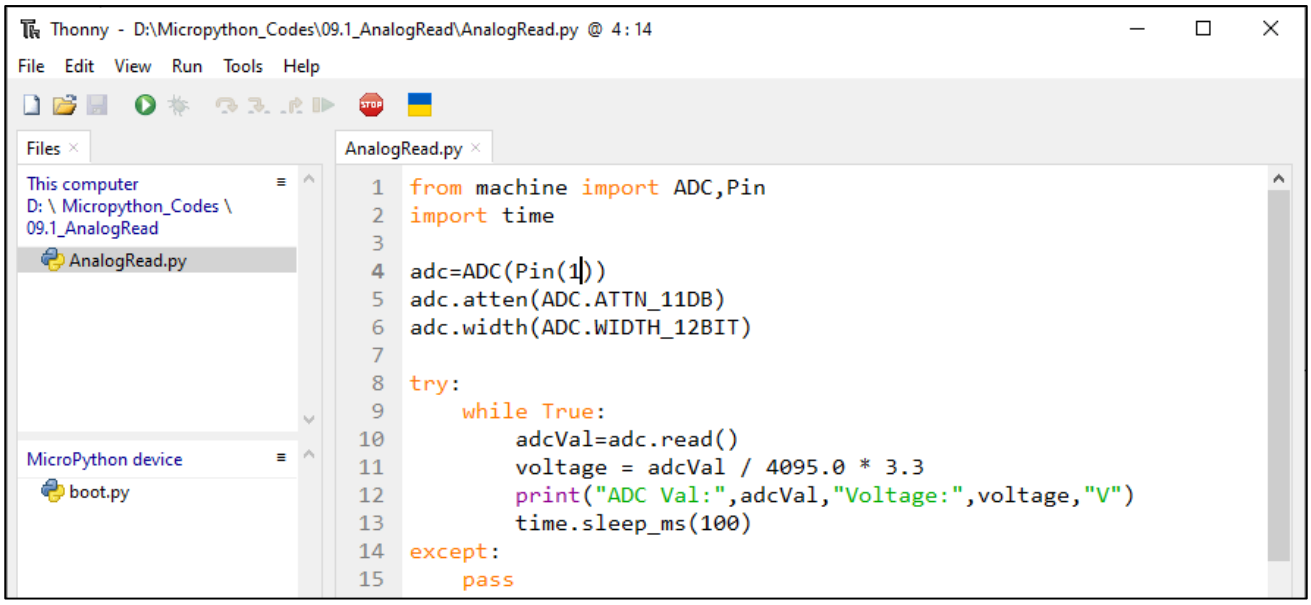


Code

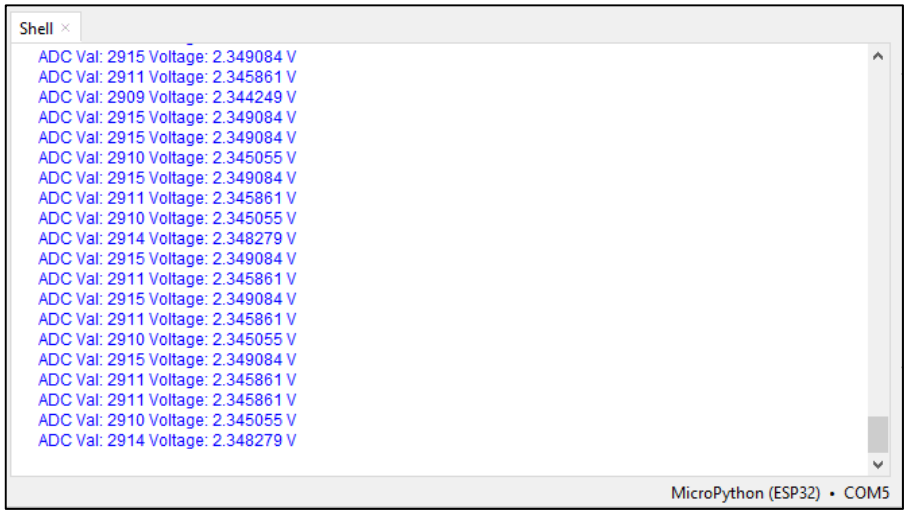
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "08.1_AnalogRead and then click "AnalogRead.py".

08.1_AnalogRead



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



The following is the code:

```
1 from machine import ADC,Pin
2 import time
3
4 adc=ADC(Pin(1))
5 adc.atten(ADC.ATTN_11DB)
6 adc.width(ADC.WIDTH_12BIT)
```

```
7
8   try:
9       while True:
10          adcVal=adc.read()
11          voltage = adcVal / 4095.0 * 3.3
12          print("ADC Val:",adcVal,"Voltage:",voltage,"V")
13          time.sleep_ms(100)
14  except:
15      pass
```

Import Pin, ADC modules.

```
1   from machine import ADC,Pin
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.

```
4   adc=ADC(Pin(1))
5   adc.atten(ADC.ATTN_11DB)
6   adc.width(ADC.WIDTH_12BIT)
```

Read ADC value once every 100 millisecods, convert ADC value and print these data to “Shell”.

```
9       while True:
10          adcVal=adc.read()
11          voltage = adcVal / 4095.0 * 3.3
12          print("ADC Val:",adcVal,"Voltage:",voltage,"V")
13          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(1-18)。

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

ADC.atten(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

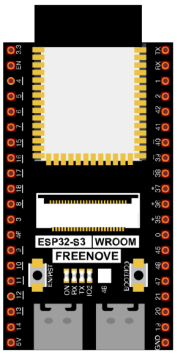
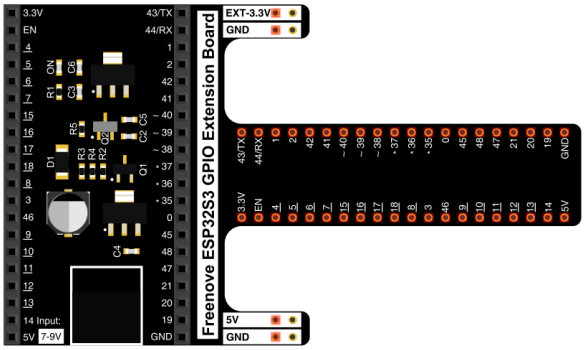
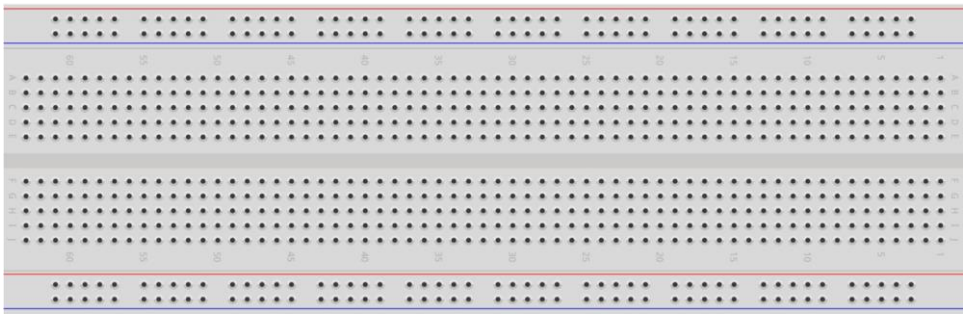
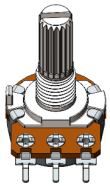
Chapter 9 Potentiometer & LED

We have already learned about the use of ADC and PWM. In this chapter, we will learn how to use a potentiometer to control the brightness of LED.

Project 9.1 Soft Light

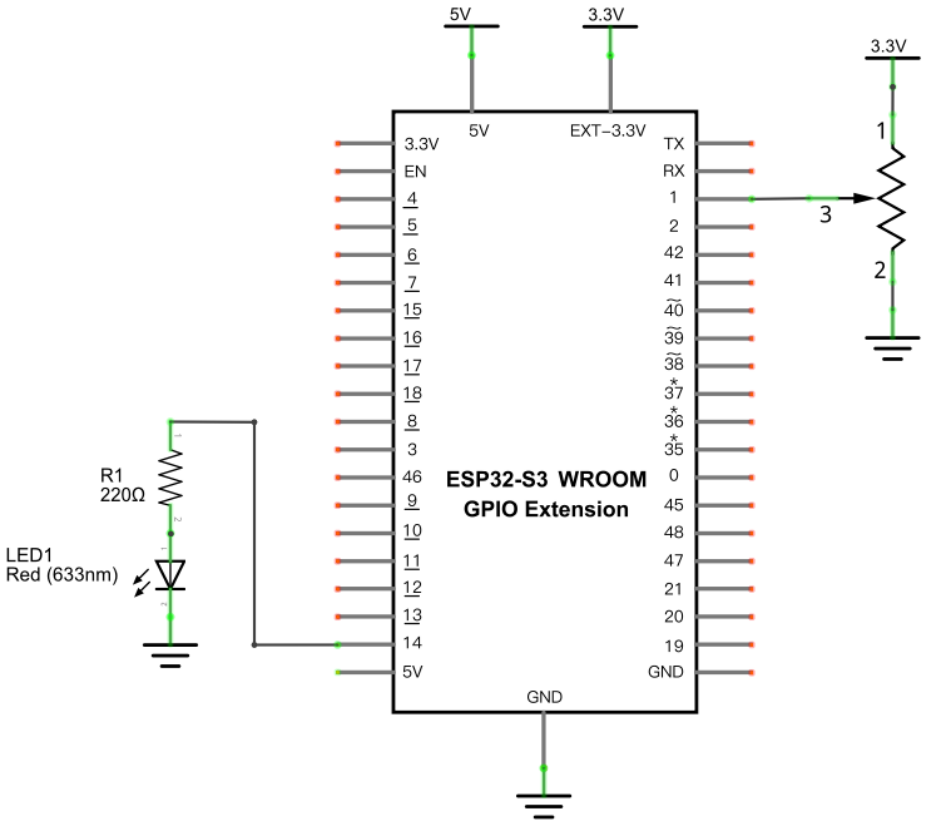
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of a LED. Then you can change the brightness of a LED by adjusting the potentiometer.

Component List

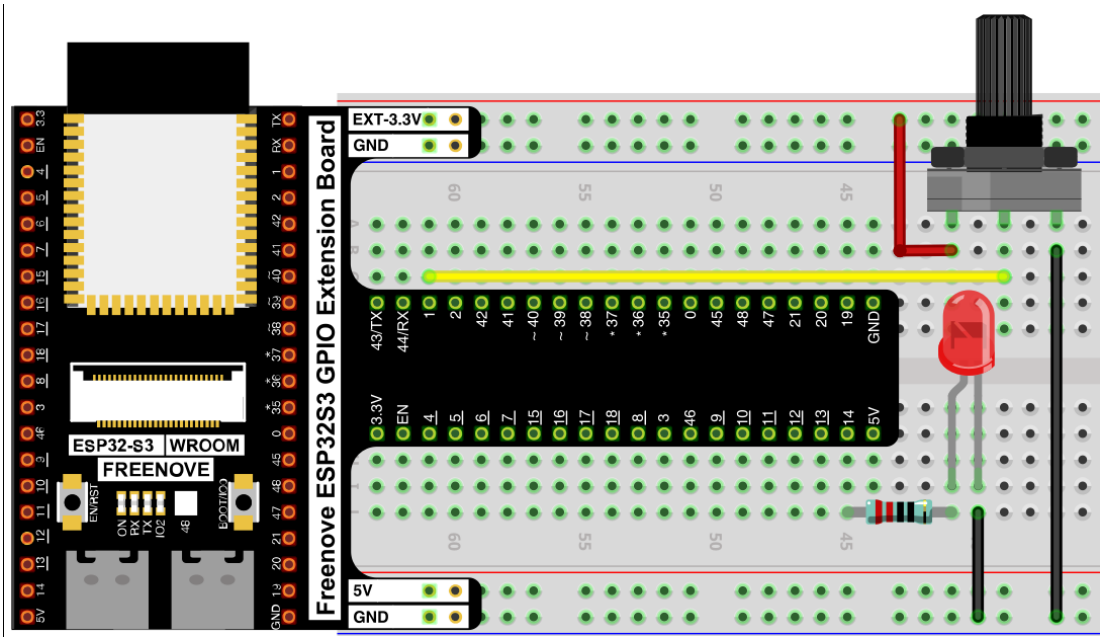
<table>
<tr><td>ESP32-S3 WROOM x1
  </td><td colspan="3">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td><td>Jumper M/M x5
</td></tr>
</table>

Circuit

Schematic diagram



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

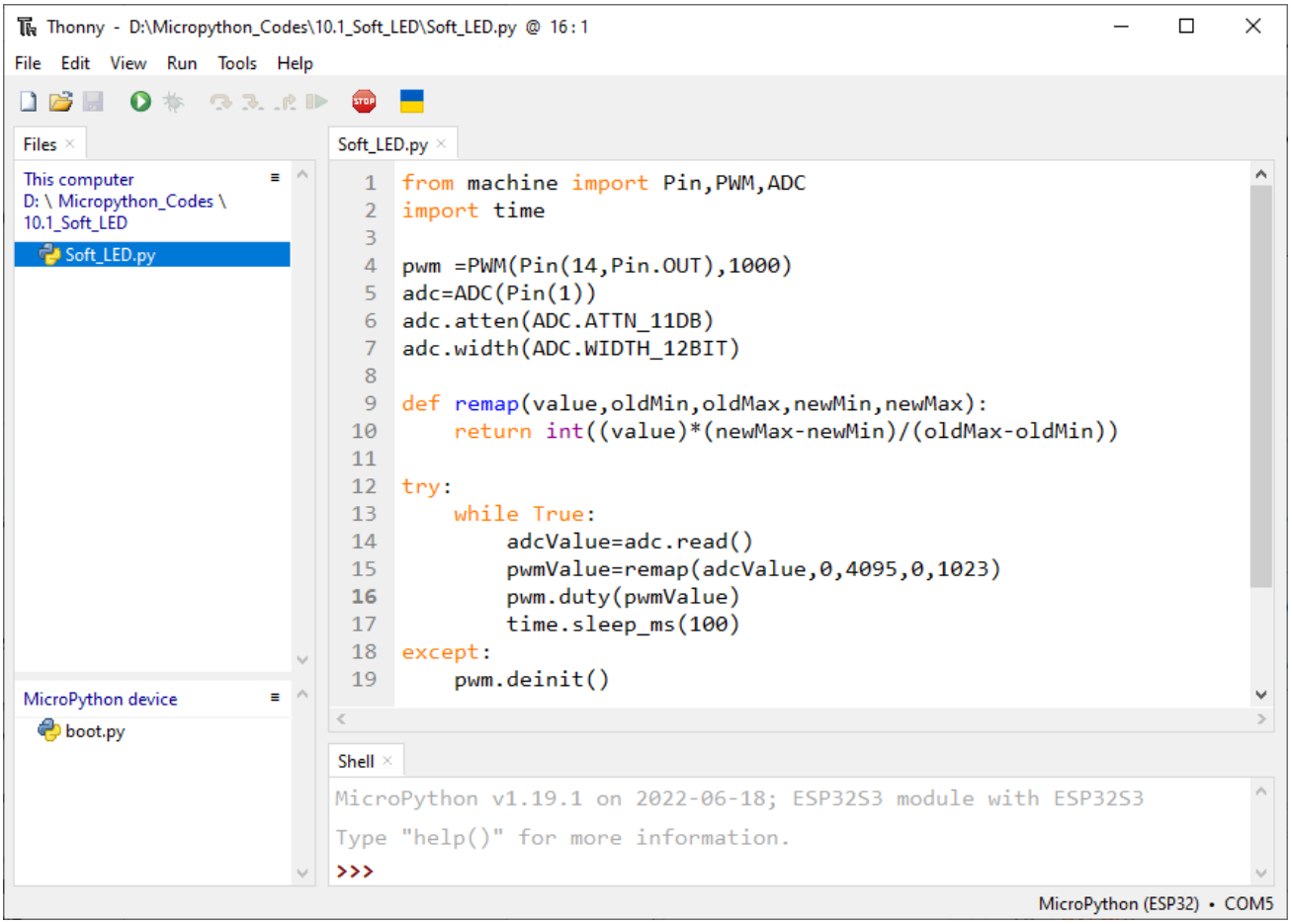


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "09.1_Soft_LED" and double click "Soft_LED.py".

09.1_Soft_LED



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

The following is the code:

```
from machine import Pin,PWM,ADC
import time

pwm =PWM(Pin(14,Pin.OUT),1000)
adc=ADC(Pin(1))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

def remap(value,oldMin,oldMax,newMin,newMax):
    return int((value)*(newMax-newMin)/(oldMax-oldMin))
```

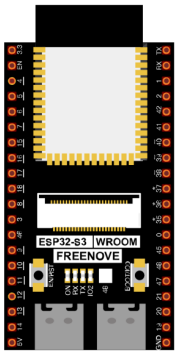
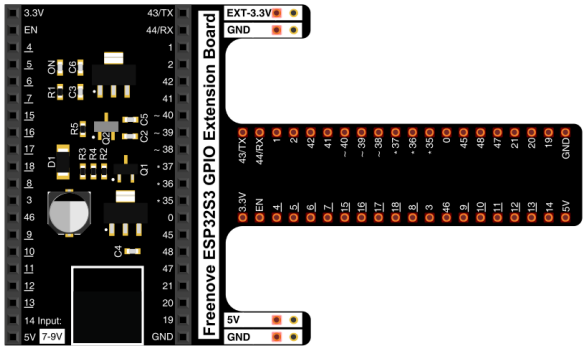
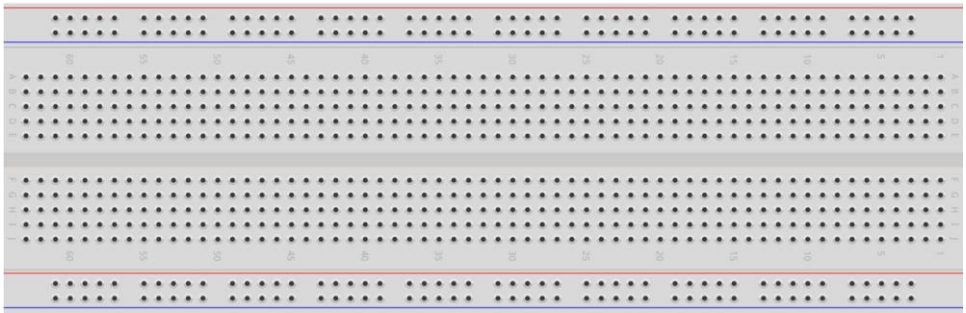
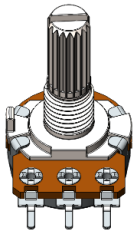
```
11
12  try:
13      while True:
14          adcValue=adc.read()
15          pwmValue=remap(adcValue, 0, 4095, 0, 1023)
16          pwm.duty(pwmValue)
17          print(adcValue,pwmValue)
18          time.sleep_ms(100)
19  except:
20      pwm.deinit()
```

In the code, read the ADC value of the potentiometer. The ADC value range is 0-4095. However, the default input range of the duty() function is 0-1023. Therefore, we need to write a remap() function to map the read ADC value to a value that conforms to the input range of the duty() function.

Project 9.2 Soft Colorful Light

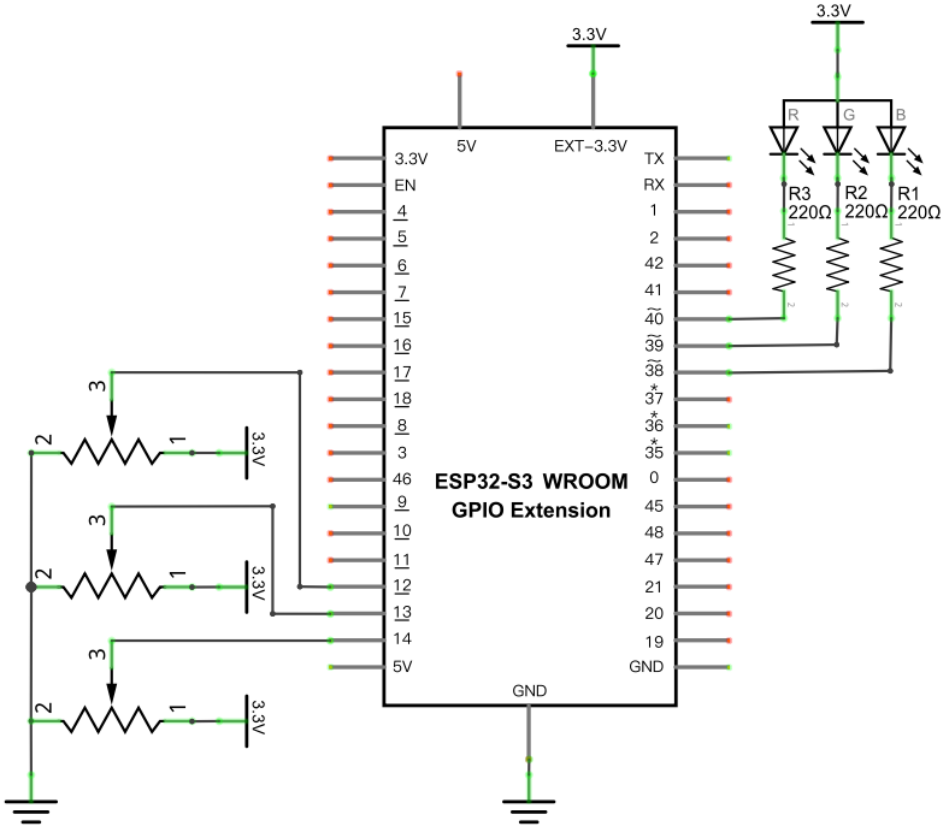
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED,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

Component List

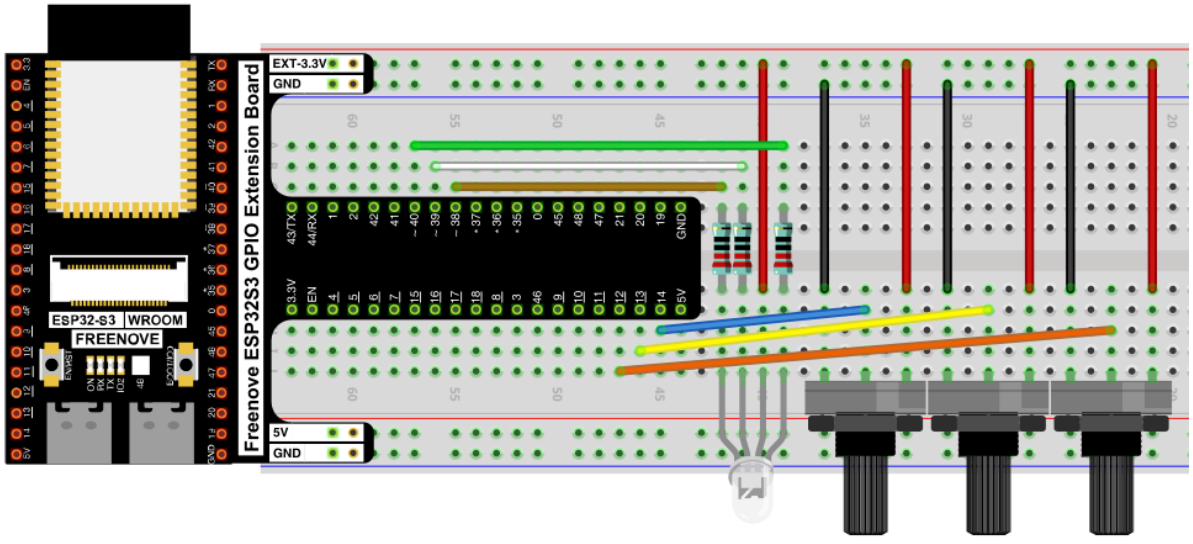
<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td>Resistor 220Ω x3
  </td><td>RGBLED x1
  </td><td>Jumper M/M x13
  </td></tr>
</table>

Circuit

Schematic diagram



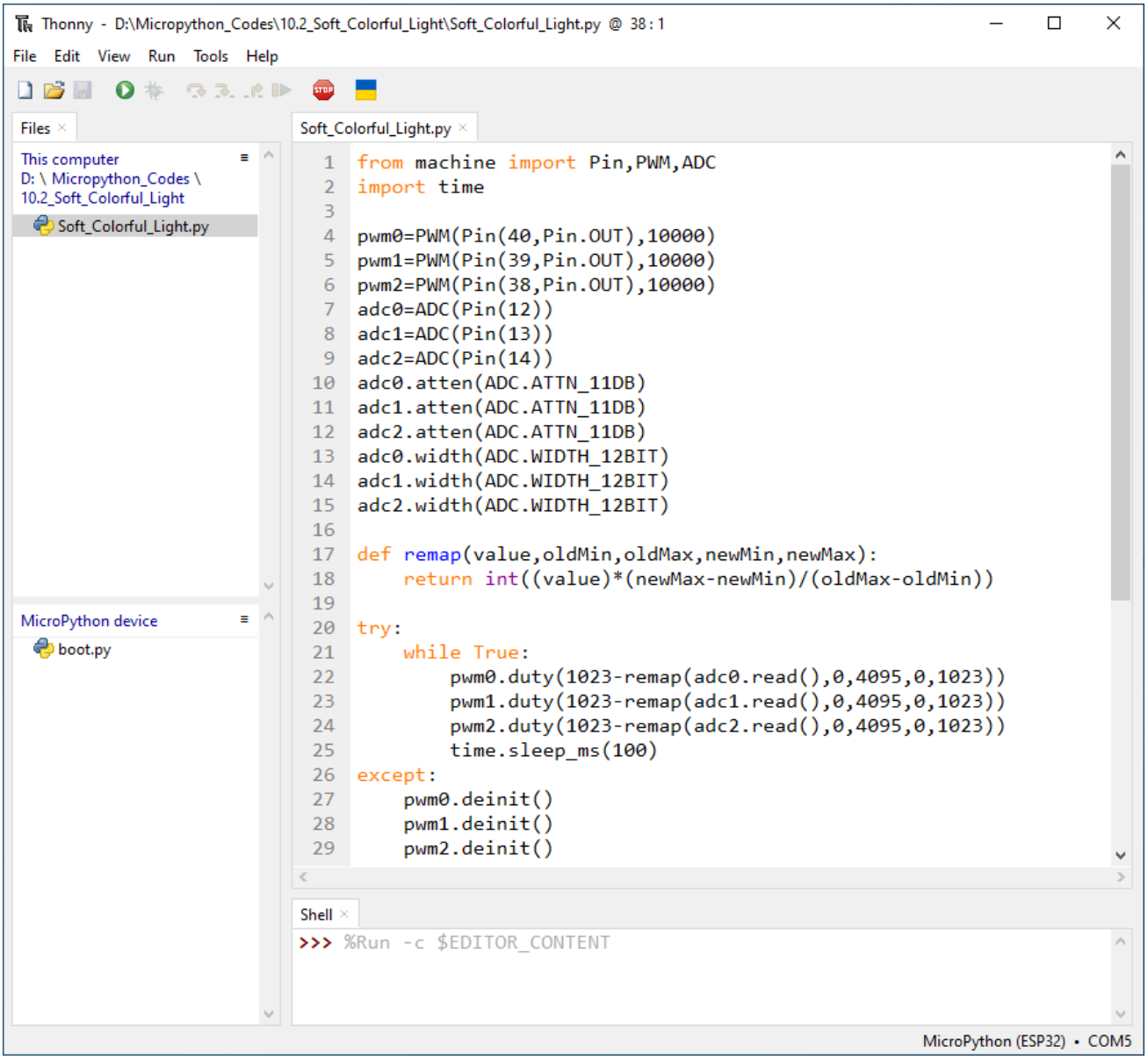
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "09.2_Soft_Colorful_Light" and double click "Soft_Colorful_Light.py".

09.2_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(40,Pin.OUT),10000)
pwm1=PWM(Pin(39,Pin.OUT),10000)
pwm2=PWM(Pin(38,Pin.OUT),10000)
```

```
adc0=ADC(Pin(12))
adc1=ADC(Pin(13))
adc2=ADC(Pin(14))
adc0.atten(ADC.ATTN_11DB)
adc1.atten(ADC.ATTN_11DB)
adc2.atten(ADC.ATTN_11DB)
adc0.width(ADC.WIDTH_12BIT)
adc1.width(ADC.WIDTH_12BIT)
adc2.width(ADC.WIDTH_12BIT)

def remap(value,oldMin,oldMax,newMin,newMax):
    return int((value)*(newMax-newMin)/(oldMax-oldMin))

try:
    while True:
        pwm0.duty(1023-remap(adc0.read(),0,4095,0,1023))
        pwm1.duty(1023-remap(adc1.read(),0,4095,0,1023))
        pwm2.duty(1023-remap(adc2.read(),0,4095,0,1023))
        time.sleep_ms(100)
except:
    pwm0.deinit()
    pwm1.deinit()
    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.

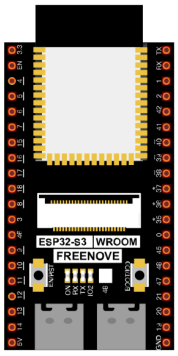
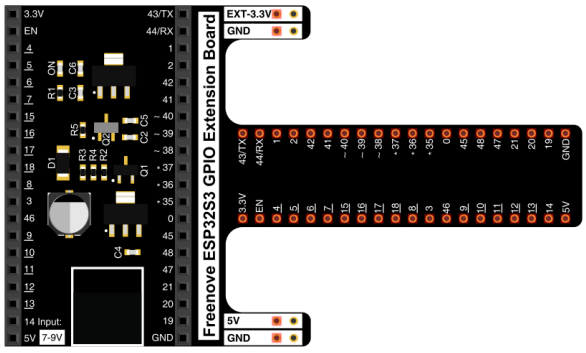
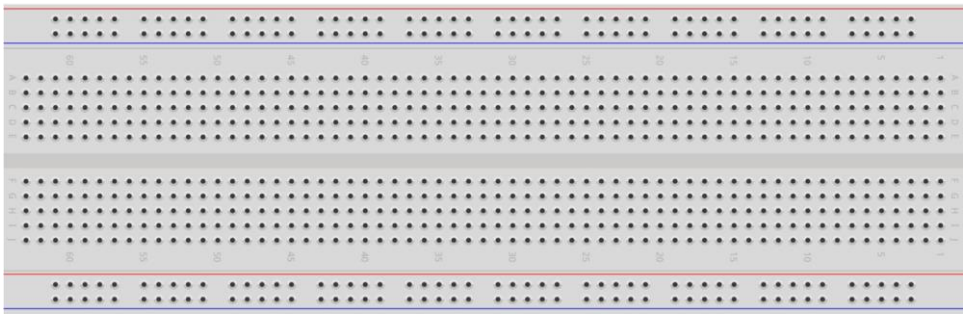
Chapter 10 Photoresistor & LED

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

Project 10.1 NightLamp

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

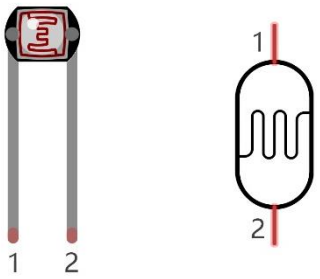
Component List

<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td colspan="3">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">Resistor
220Ω x1
 
10KΩ x1
  </td><td>LED x1
  </td><td>Jumper M/M x4
  </td></tr>
</table>

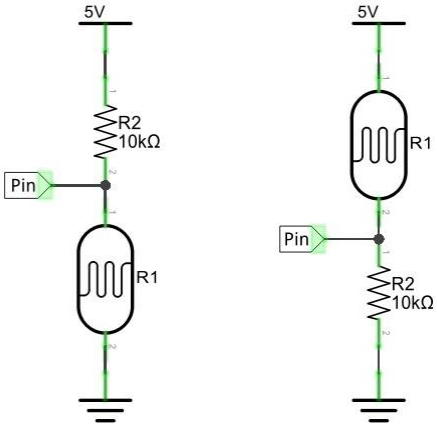
Component knowledge

Photoresistor

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



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

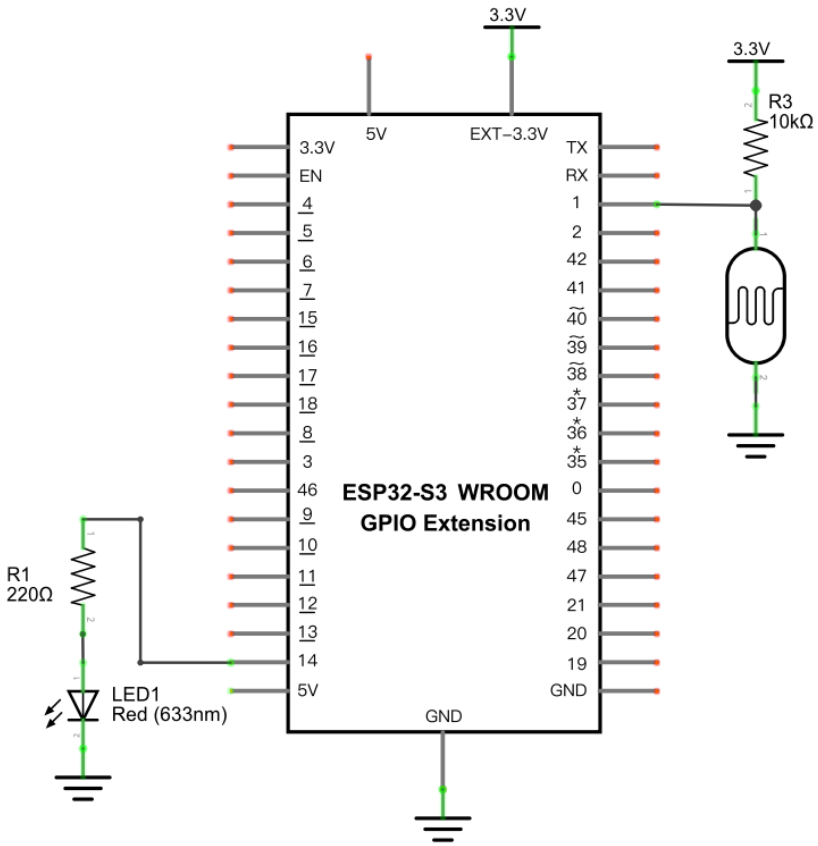


In the above circuit, when a photoresistor's resistance vale changes due to a change in light intensity, the voltage between the photoresistor and resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

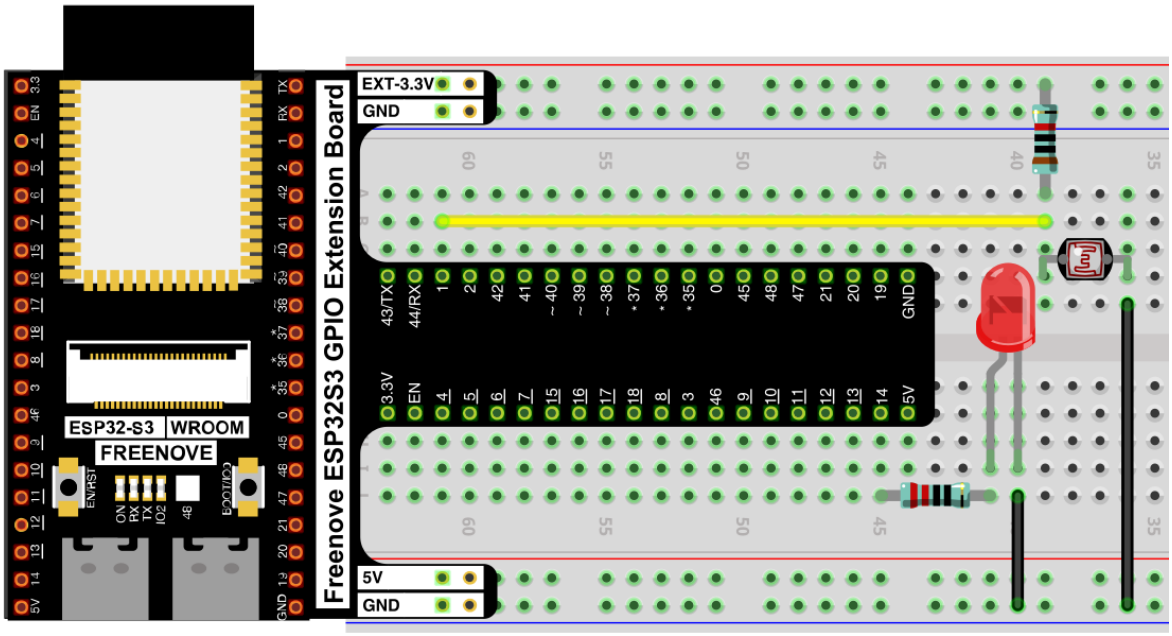
Circuit

The circuit of this project is similar to project Soft Light. 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

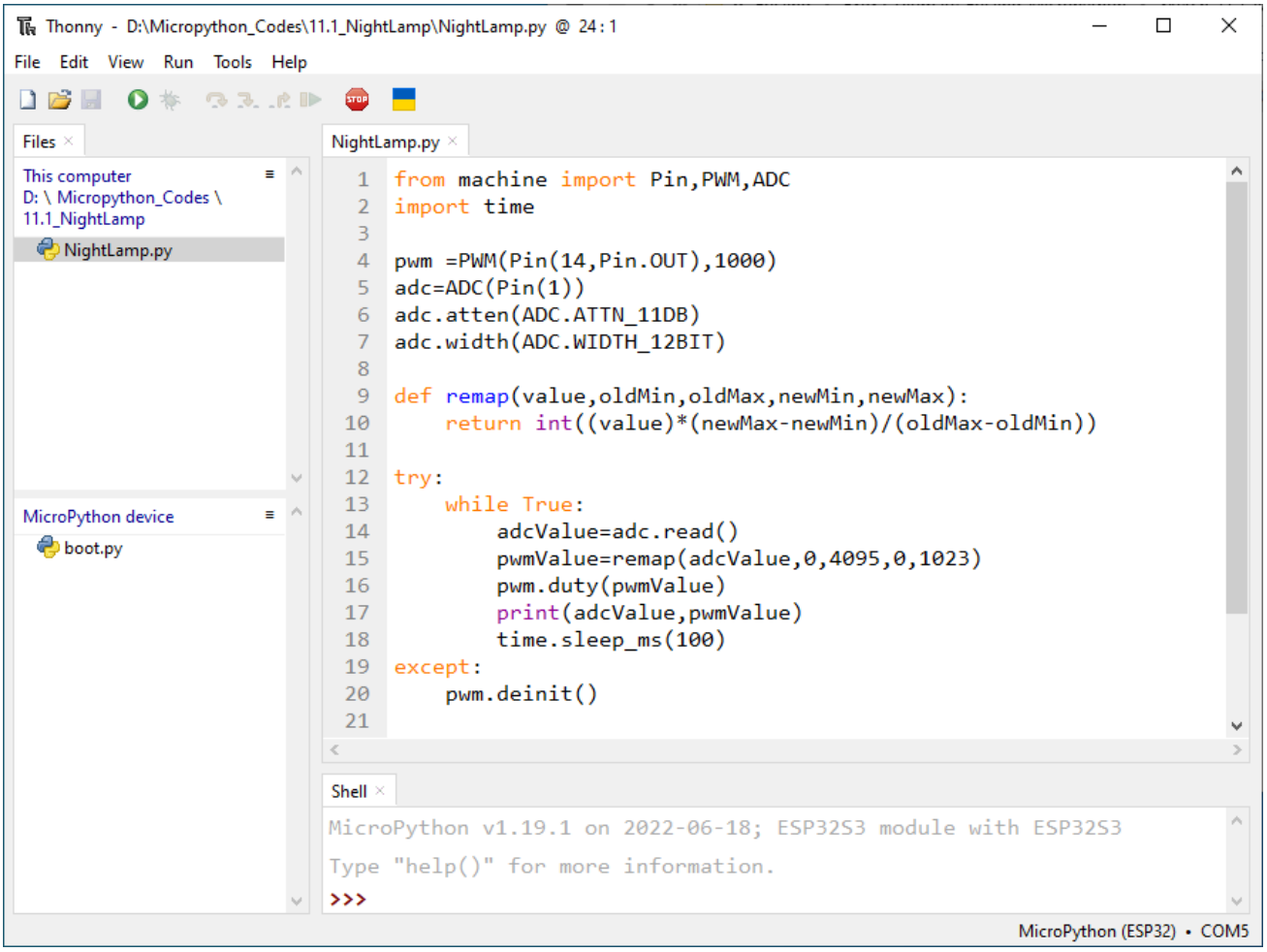


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Codes of this project is logically the same as the project Soft Light.

10.1_Nightlamp



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

The following is the program code:

```
from machine import Pin,PWM,ADC
import time

pwm =PWM(Pin(14,Pin.OUT),1000)
adc=ADC(Pin(1))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

def remap(value,oldMin,oldMax,newMin,newMax):
    return int((value)*(newMax-newMin)/(oldMax-oldMin))
```

```
try:
    while True:
        adcValue=adc.read()
        pwmValue=remap(adcValue,0,4095,0,1023)
        pwm.duty(pwmValue)
        print(adcValue,pwmValue)
        time.sleep_ms(100)
except:
    pwm.deinit()
```

In this code, we use ADC to read the ADC value of the photoresist and map it to the PWM of the control LED, so that the brightness of the LED can change accordingly with the change of the ambient light intensity.

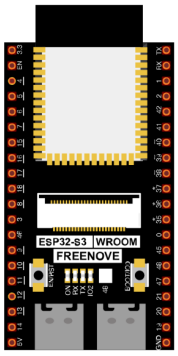
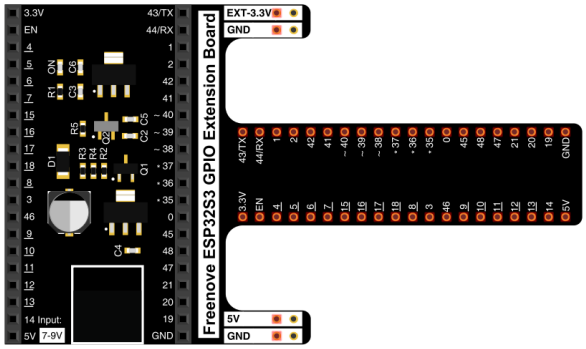
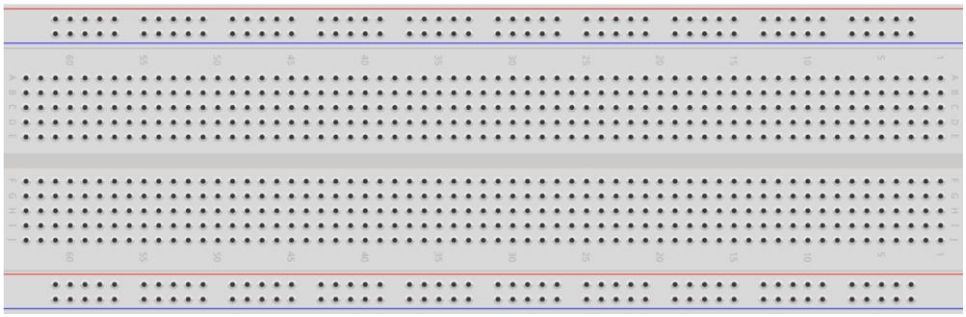
Chapter 11 Thermistor

In this chapter, we will learn about thermistors which are another kind of resistor

Project 11.1 Thermometer

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

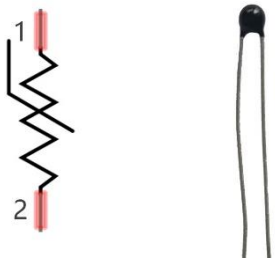
Component List

<table>
<tr><td>ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Thermistor x1
  </td><td>Resistor 10kΩ x1
  </td><td>Jumper M/M x3
  </td></tr>
</table>

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:

$$Rt = R * EXP[\ B * (\frac{1}{T2} - \frac{1}{T1})\ ]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

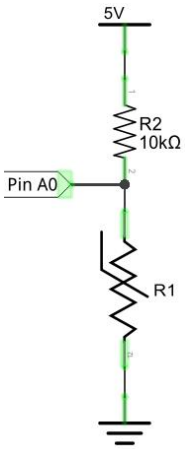
EXP[n] is nth power of E;

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.

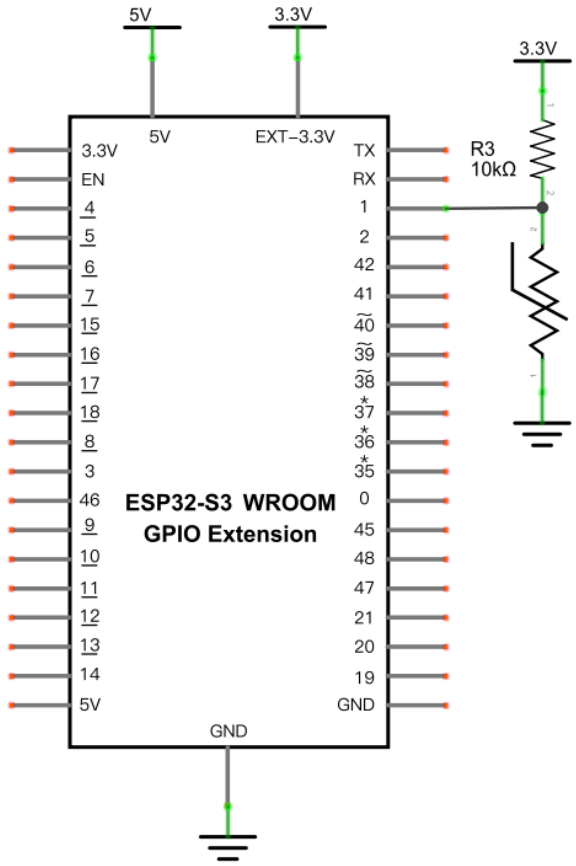
Therefore, the temperature formula can be derived as:

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

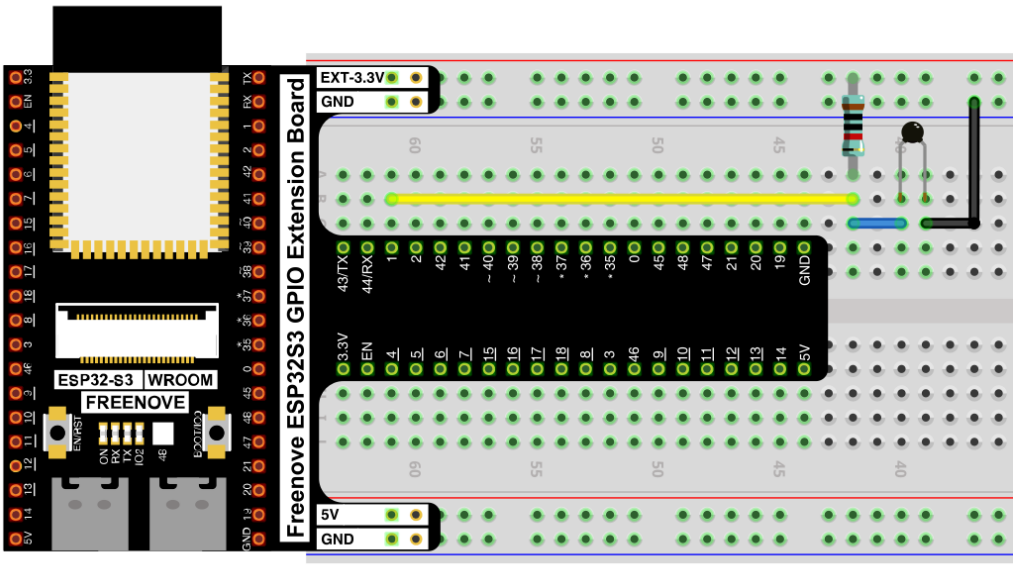
Circuit

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

Schematic diagram



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

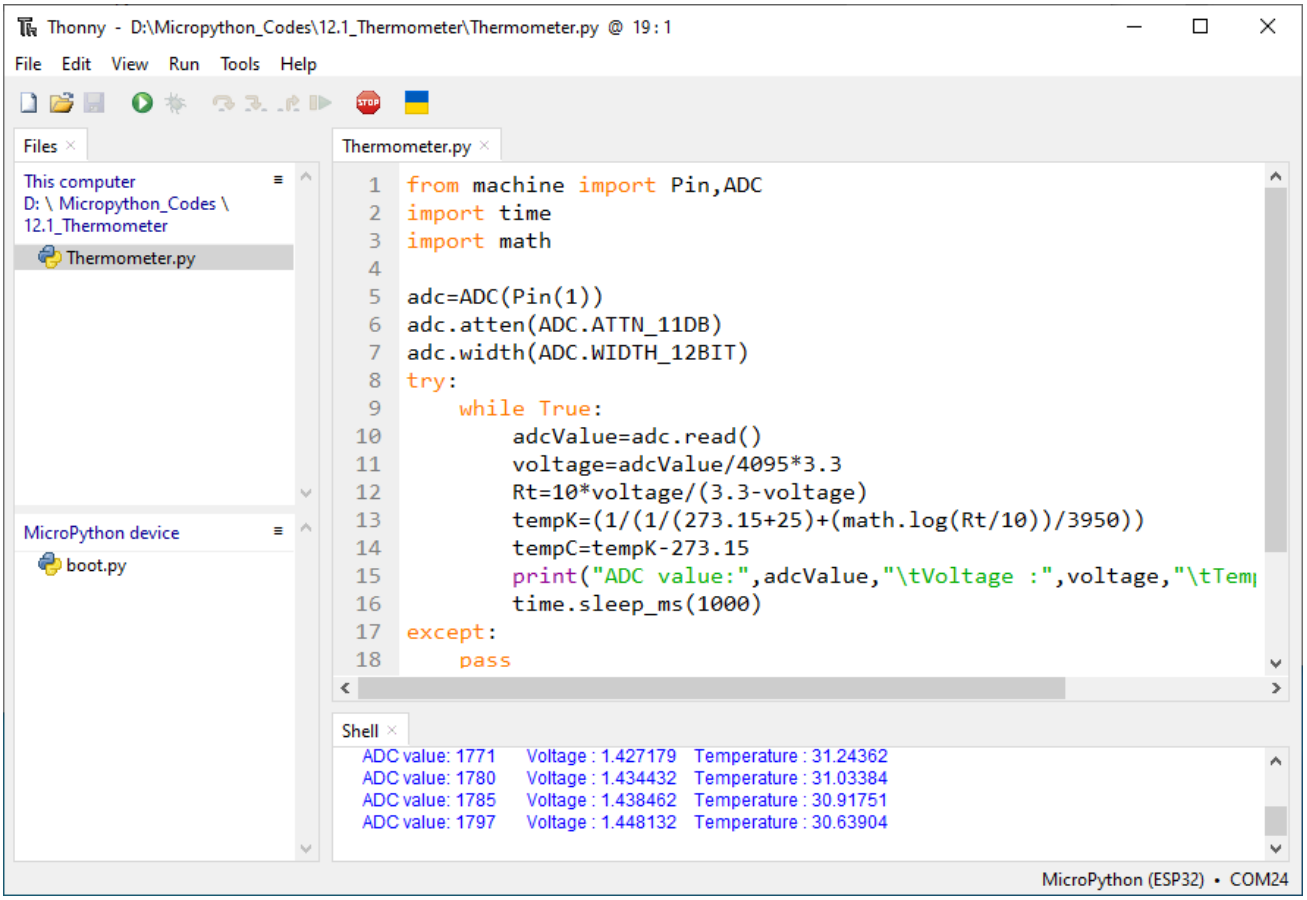


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

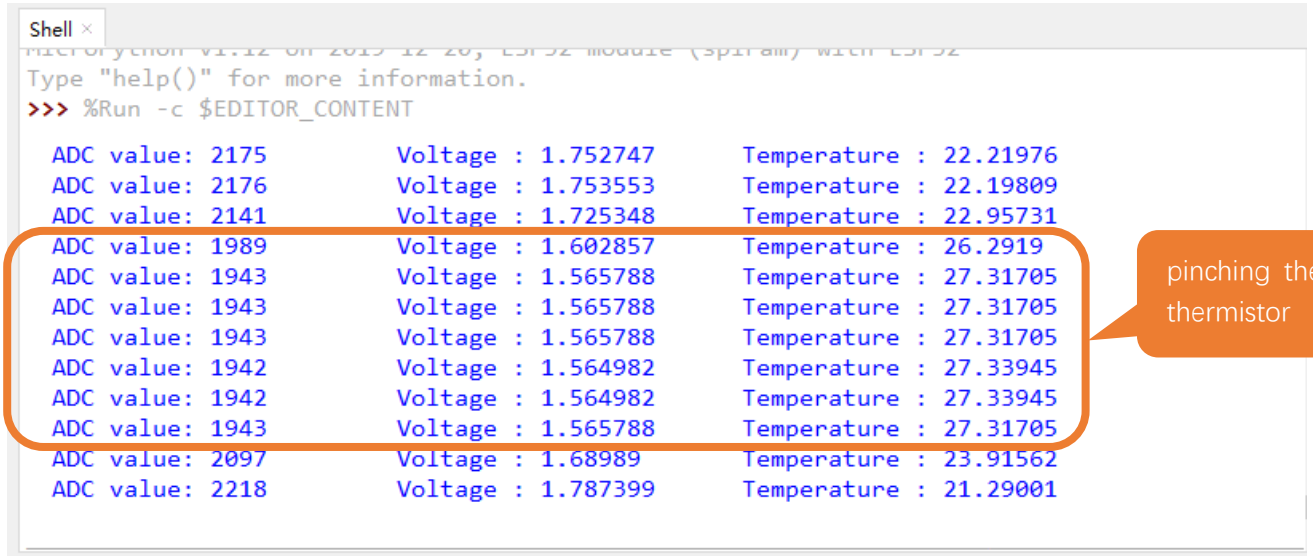
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "11.1_Thermometer" and double click "Thermometer.py".

11.1_Thermometer



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


Shell
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
pinching the thermistor

The following is the code:

```
from machine import Pin,ADC
import time
import math

adc=ADC(Pin(1))
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,"\tVoltage :",voltage,"\tTemperature :",tempC);
        time.sleep_ms(1000)
except:
    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.

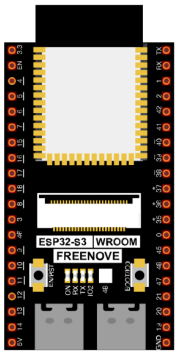
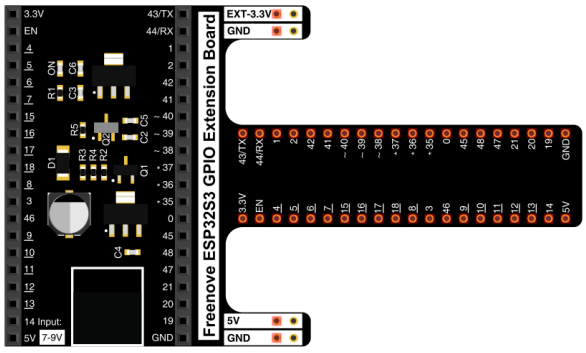
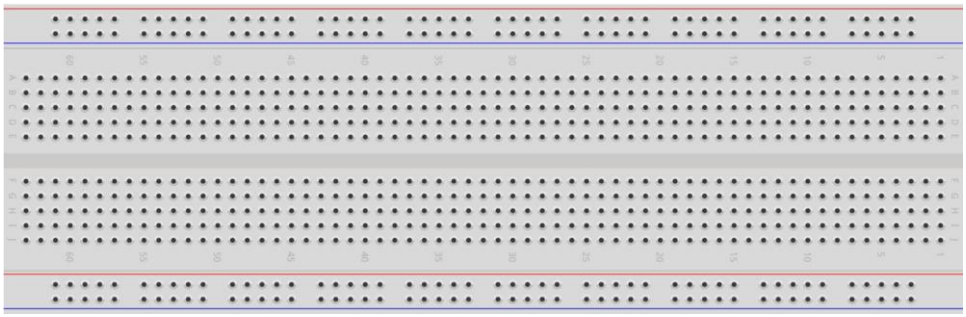
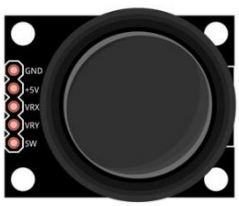
Chapter 12 Joystick

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 12.1 Joystick

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

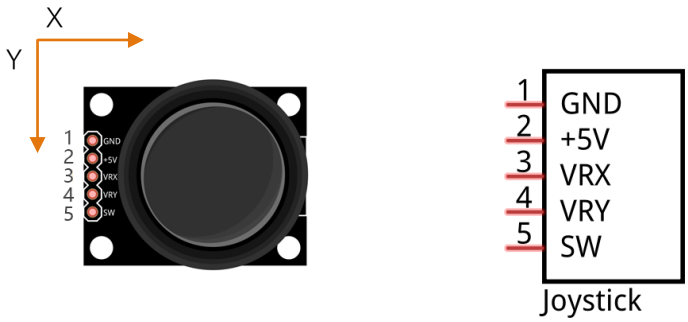
Component List

ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
Joystick x1 	Jumper F/M x5 

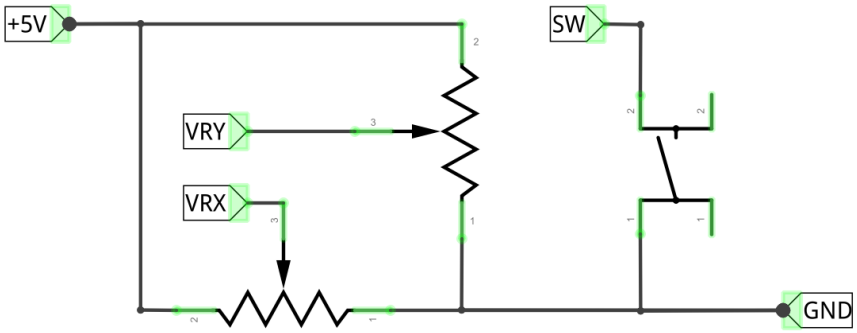
Component knowledge

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).



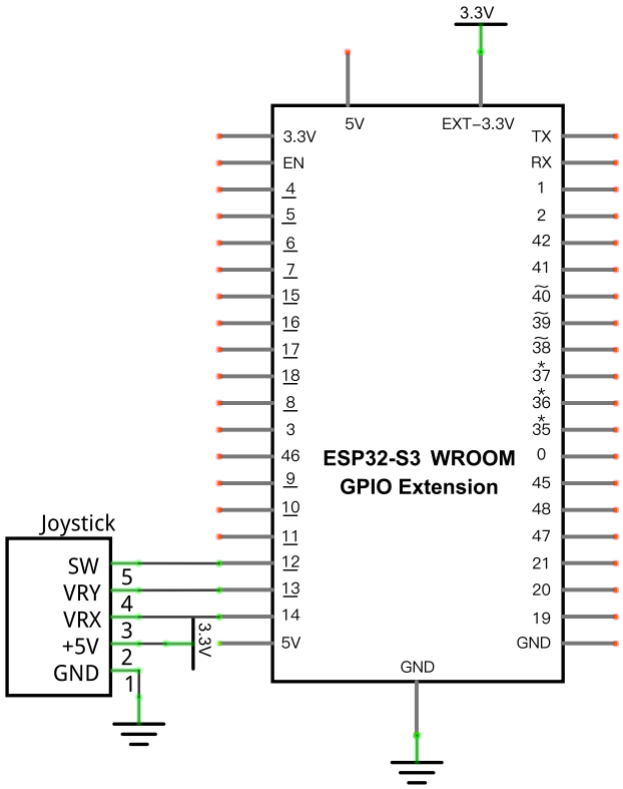
This is accomplished by incorporating two rotary potentiometers inside the joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a push button switch in the “vertical” axis, which can detect when a User presses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



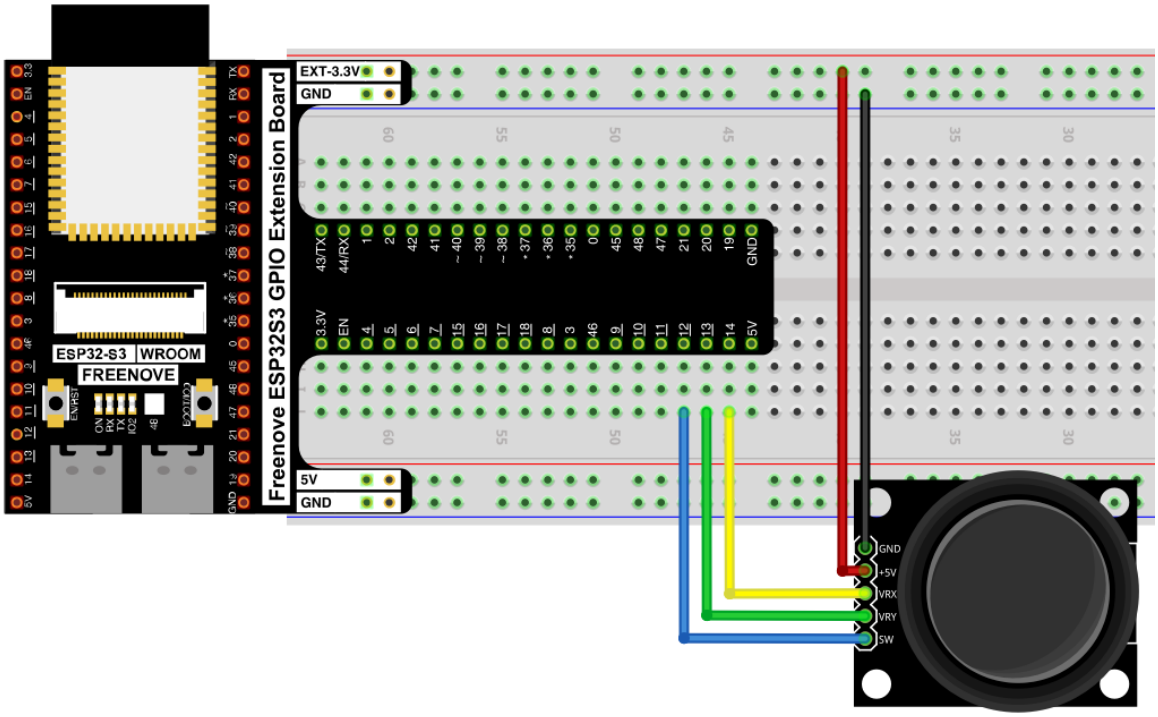
When the joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



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



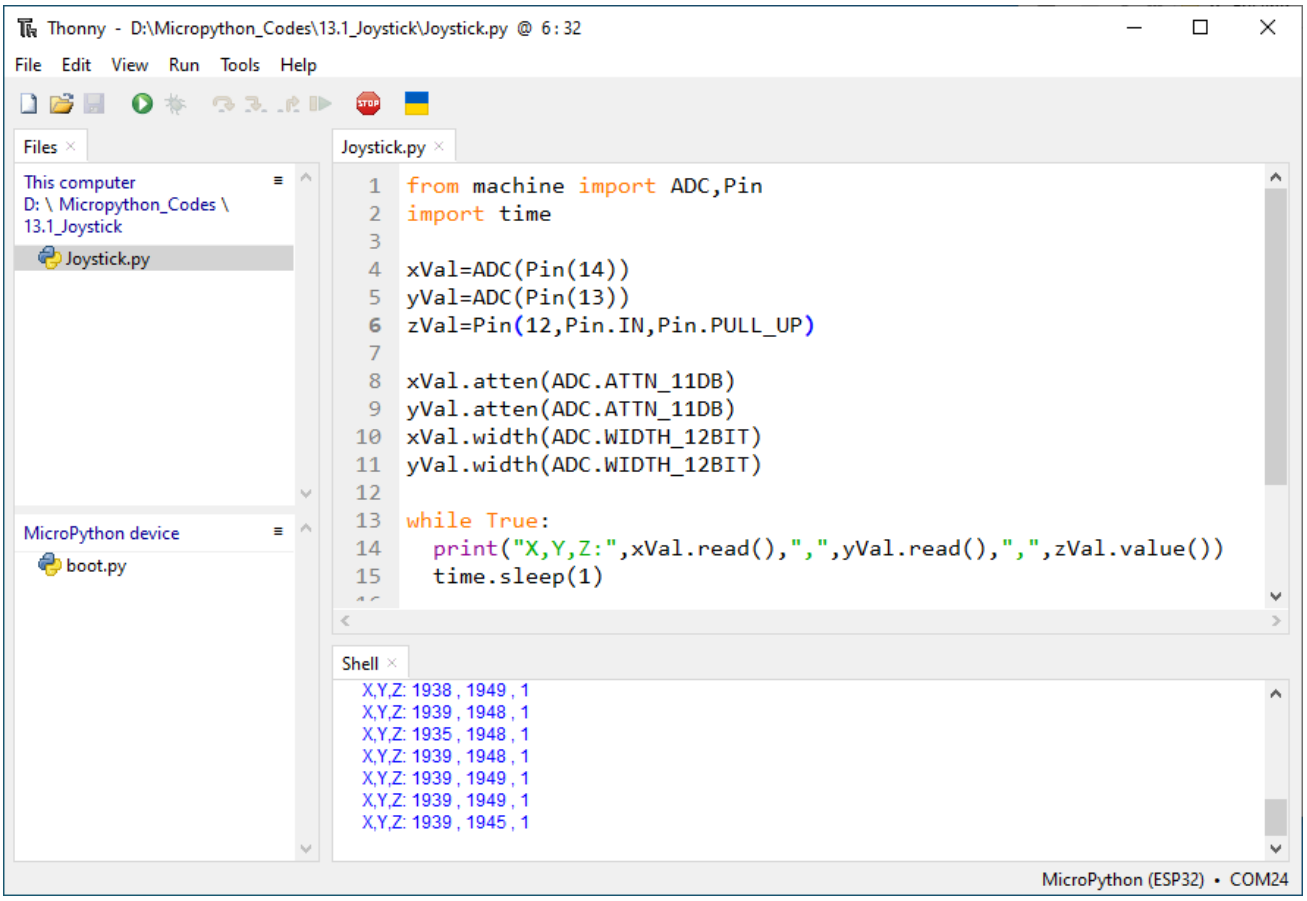
Code

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.

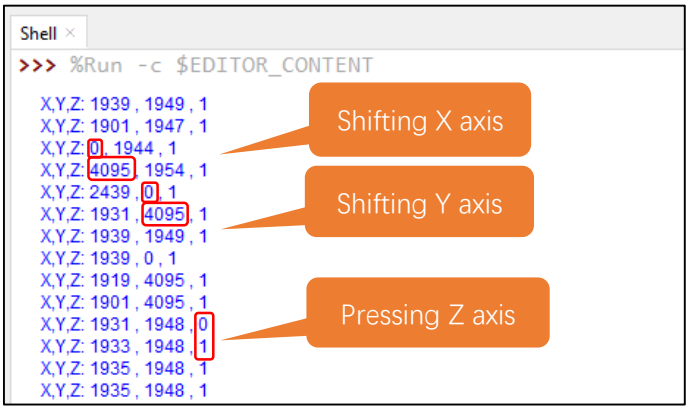
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "12.1_Joystick" and double click "Joystick.py".

12.1_Joystick



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



The flowing is the code:

```
1  from machine import ADC,Pin
2  import time
3
4  xVal=ADC(Pin(14))
5  yVal=ADC(Pin(13))
6  zVal=Pin(12,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())
```

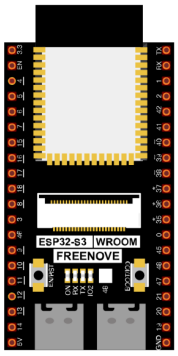
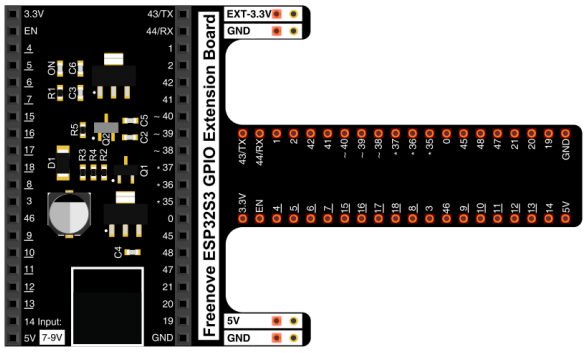
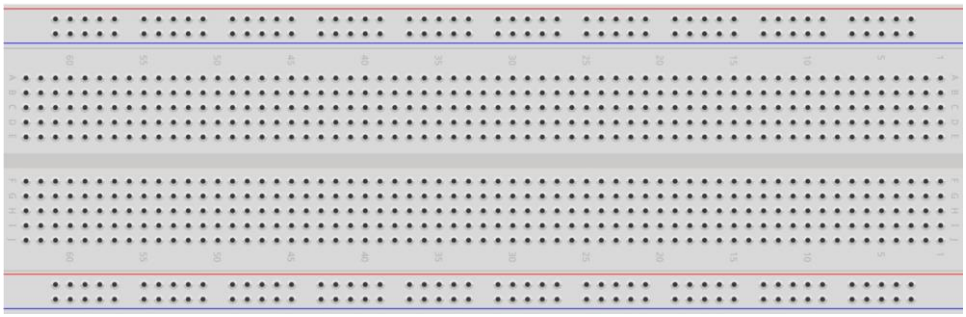
Chapter 13 74HC595 & LED Bar Graph

We have used LED bar graph to make a flowing water light, in which 10 GPIO ports of ESP32-S3 is occupied. More GPIO ports mean that more peripherals can be connected to ESP32S3, 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 13.1 Flowing Water Light

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

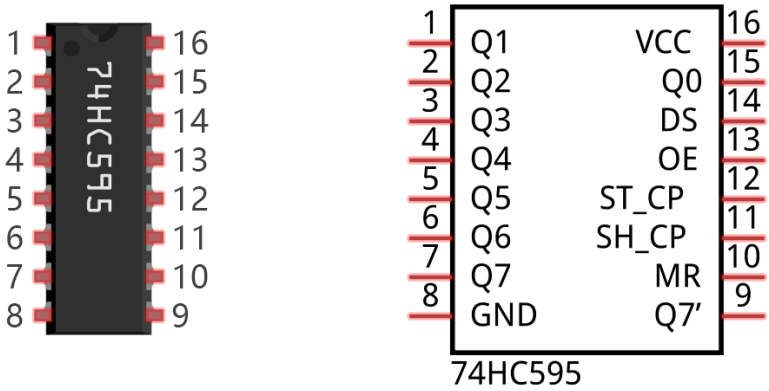
Component List

<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x1
  </td><td>LED Bar Graph x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper M/M x15
  </td></tr>
</table>

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 a ESP32S3. At least 3 ports are required to control the 8 ports of the 74HC595 chip.



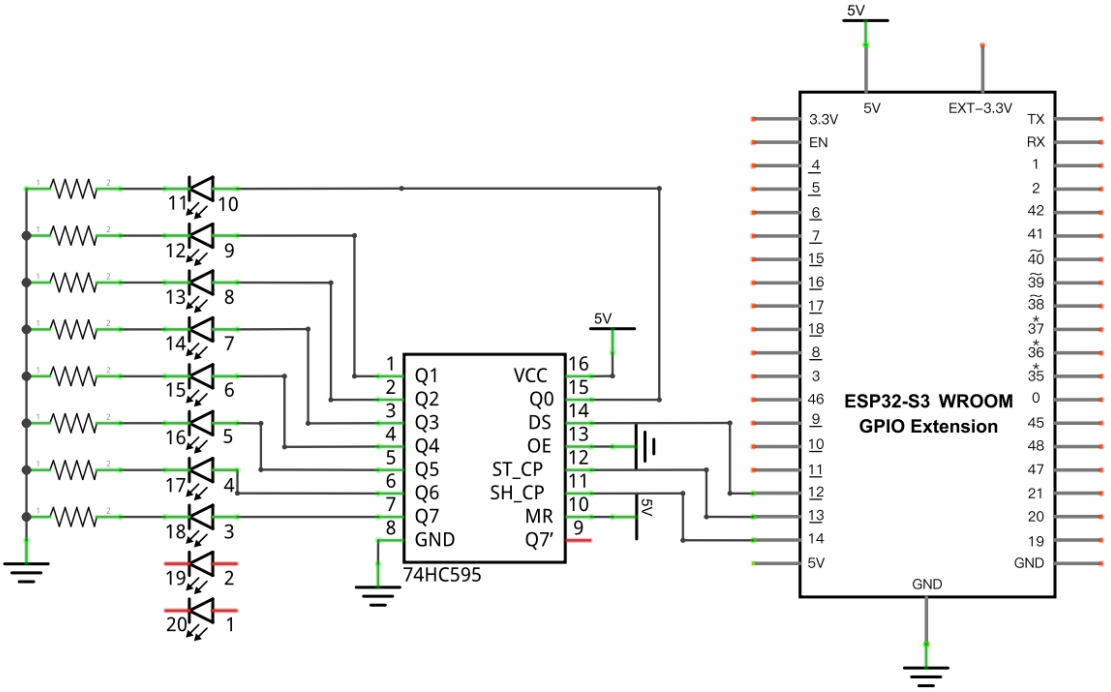
The ports of the 74HC595 chip are described as follows:

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.

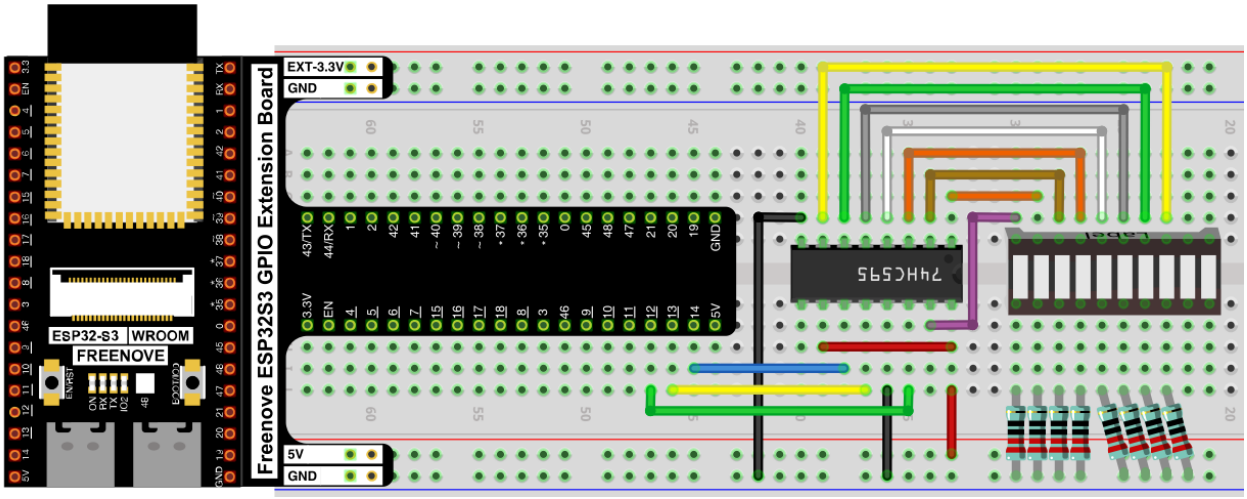
For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



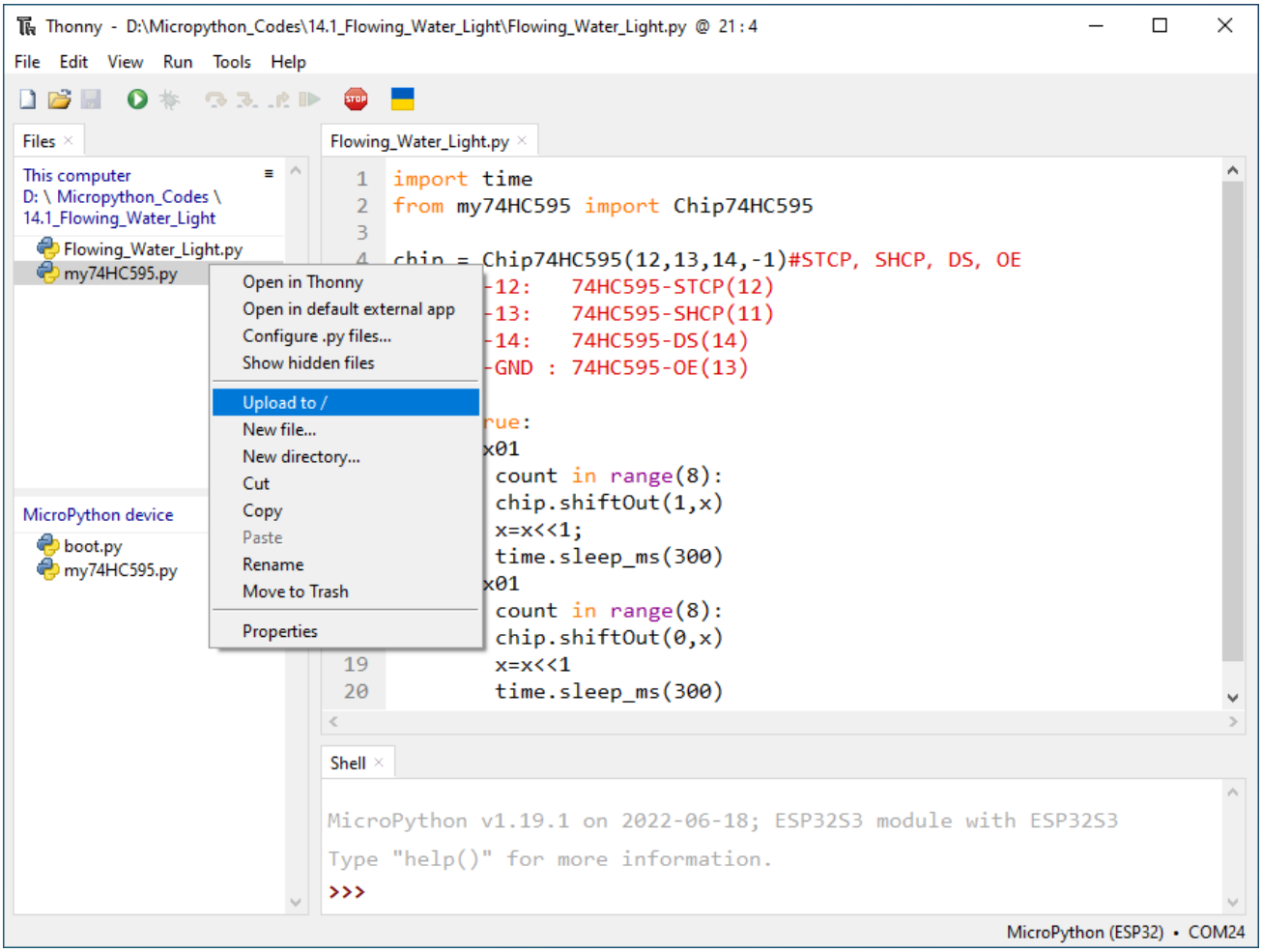
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

In this project, we will make a flowing water light with a 74HC595 chip to learn about its functions.
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "13.1_Flowing_Water_Light".
Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py"to be uploaded to ESP32-S3 and then double click "Flowing_Water_Light.py".

13.1_Flowing_Water_Light



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:

```
import time
from my74HC595 import Chip74HC595

chip = Chip74HC595(12,13,14,-1)#STCP, SHCP, DS, OE
# ESP32-12:   74HC595-STCP(12)
# ESP32-13:   74HC595-SHCP(11)
# ESP32-14:   74HC595-DS(14)
# ESP32-GND : 74HC595-OE(13)

while True:
    x=0x01
    for count in range(8):
        chip.shiftOut(1,x)
        x=x<<1;
        time.sleep_ms(300)
    x=0x01
    for count in range(8):
        chip.shiftOut(0,x)
        x=x<<1
        time.sleep_ms(300)
```

Import time and my74HC595 modules.

```
import time
from my74HC595 import Chip74HC595
```

Assign pins for ESP32-S3 to connect to 74HC595.

```
chip = Chip74HC595(12,13,14,-1)#STCP, SHCP, DS, OE
```

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.

```
    x=0x01
    for count in range(8):
        chip.shiftOut(1,x)  #High bit is sent first
        x=x<<1
        time.sleep_ms(300)
    x=0x01
    for count in range(8):
        chip.shiftOut(0,x)  #Low bit is sent first
        x=x<<1
        time.sleep_ms(300)
```

Reference

Class Chip74HC595

Before each use of the object **Chip74HC595**, make sure my74HC595.py has been uploaded to "/" of ESP32S3, 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 ESP32S3, ST_CP pin is connected to ESP32S3's Pin(12) and OE pin is connected to ESP32S3's Pin(5). If you need to modify the pins, just do the following operations.

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

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.。

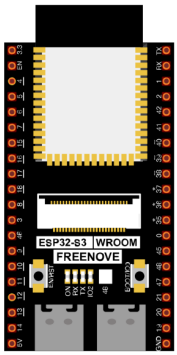
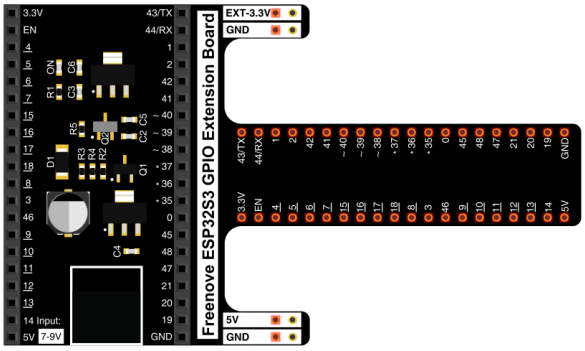
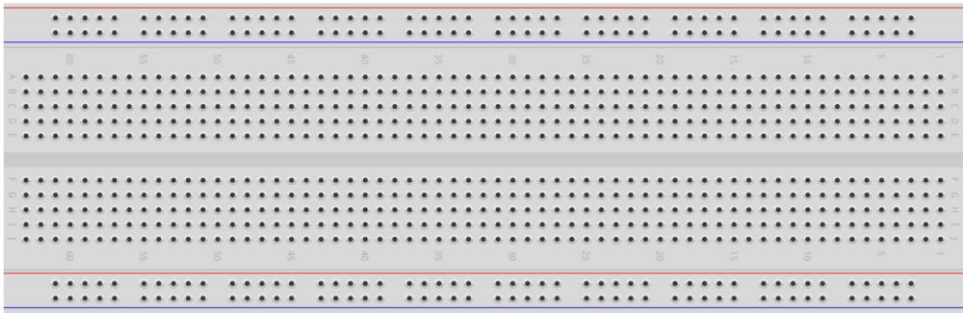
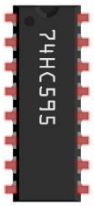
Chapter 14 74HC595 & 7-Segment Display.

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

Project 14.1 7-Segment Display.

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

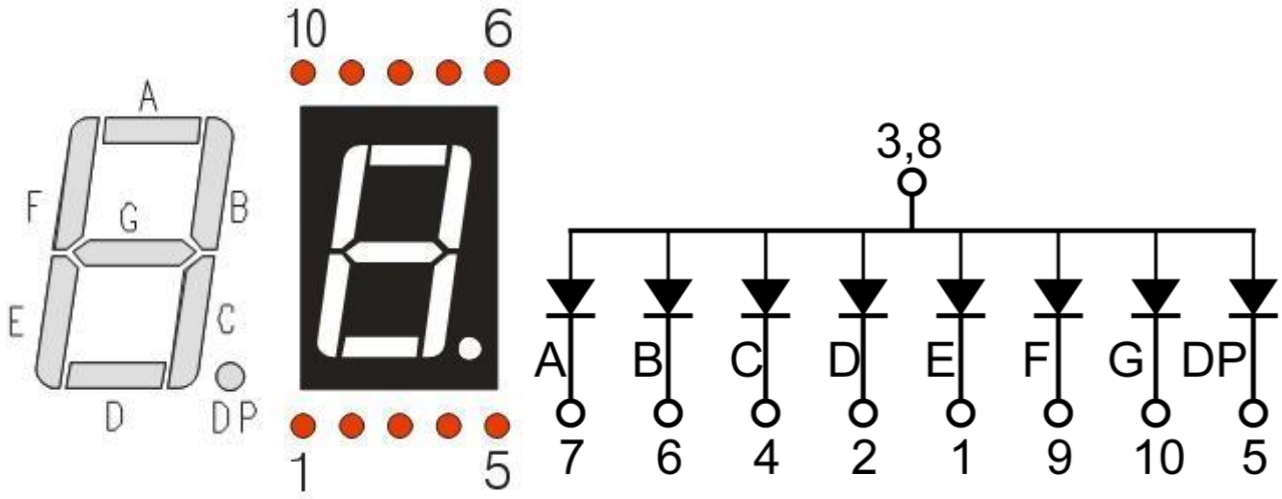
Component List

<table>
<tr><td colspan="2">ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x1
  </td><td>7-segment display x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper M/M
  </td></tr>
</table>

Component knowledge

7-segment display

A 7-segment display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a common anode and individual cathodes. Its internal structure and pin designation diagram is shown below:



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

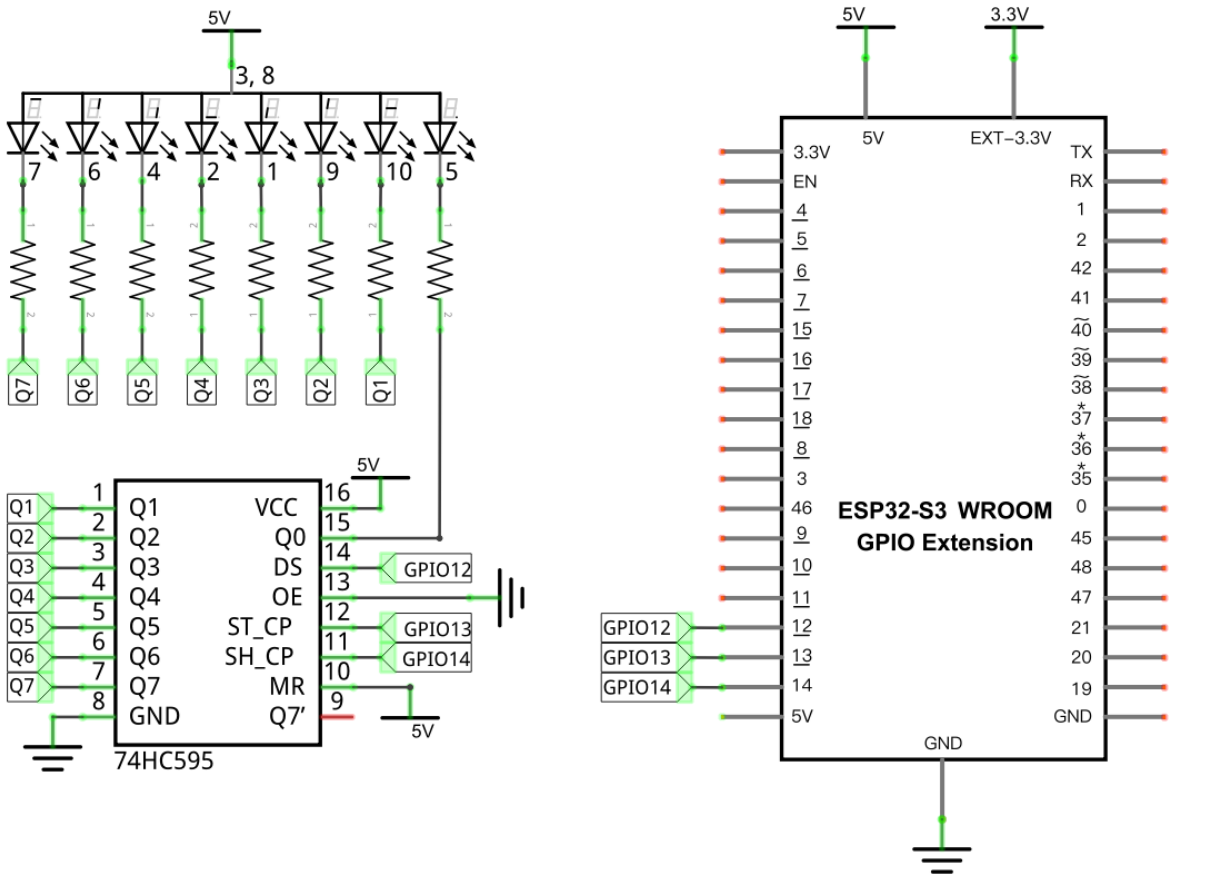
In this project, we will use a 7-Segment Display with a common anode. Therefore, when there is an input low level to a LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”,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

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

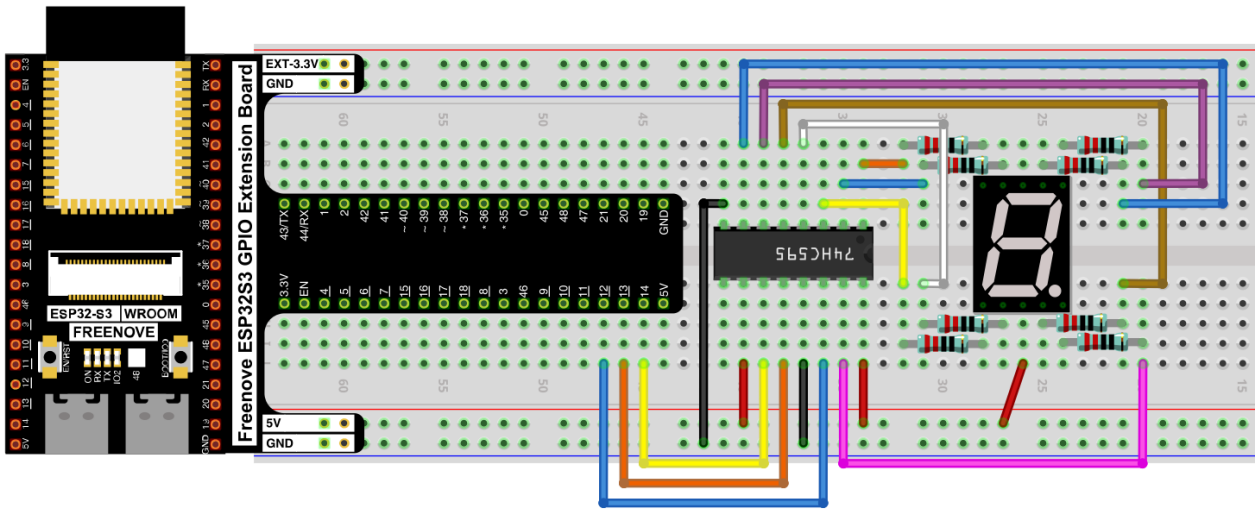
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0	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

Schematic diagram



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



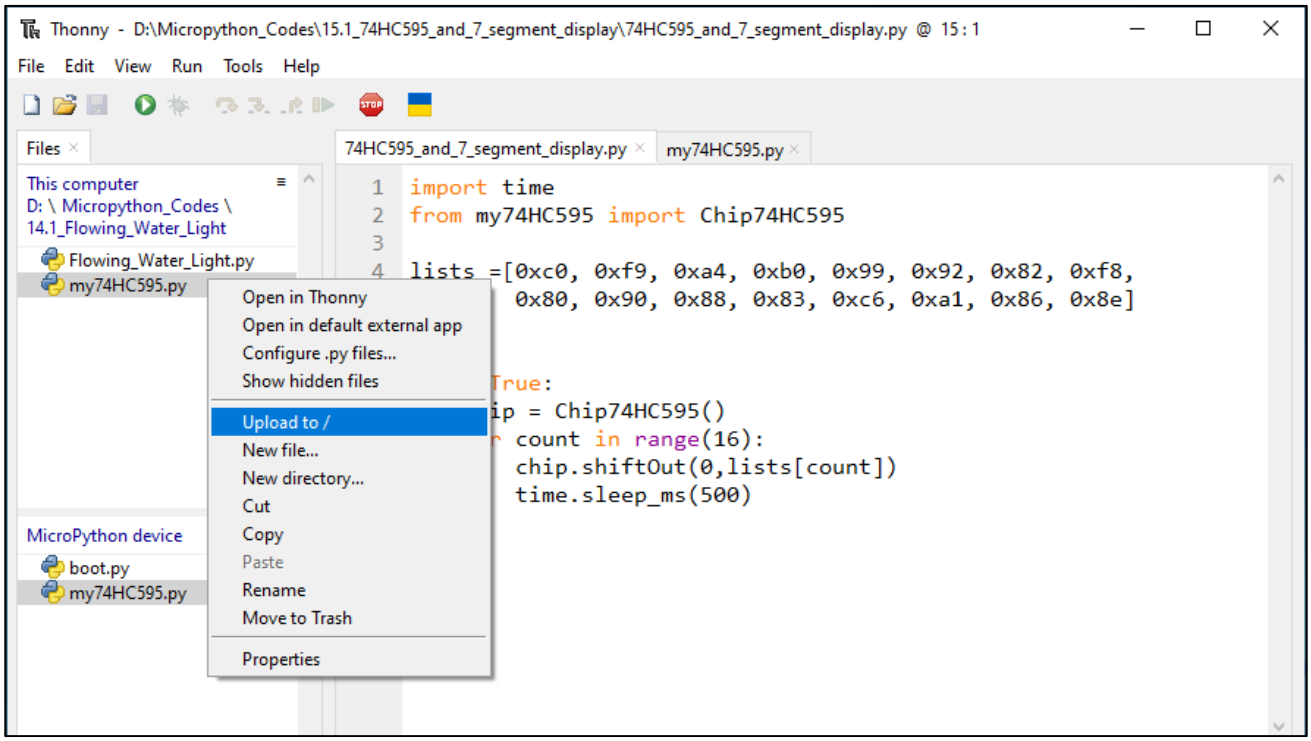
Code

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".
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

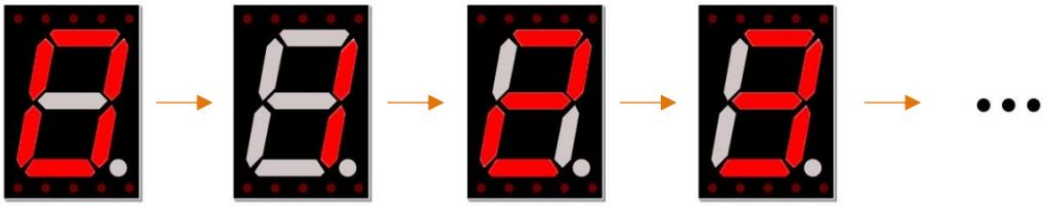
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.1_74HC595_and_7_segment_display".

Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to ESP32-S3 and then double click"74HC595_and_7_segment_display.py".

14.1_74HC595_and_7_segment_display



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:

```
import time
from my74HC595 import Chip74HC595

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

chip = Chip74HC595(12,13,14)
try:
    while True:
        for count in range(16):
            chip.shiftOut(0,lists[count])
            time.sleep_ms(500)
except:
    pass
```

Import time and my74HC595 modules.

```
import time
from my74HC595 import Chip74HC595
```

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

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

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

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

Send data of digital tube to 74HC595 chip.

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

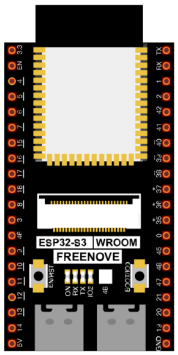
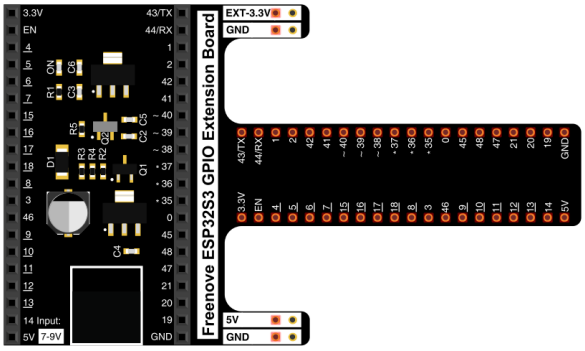
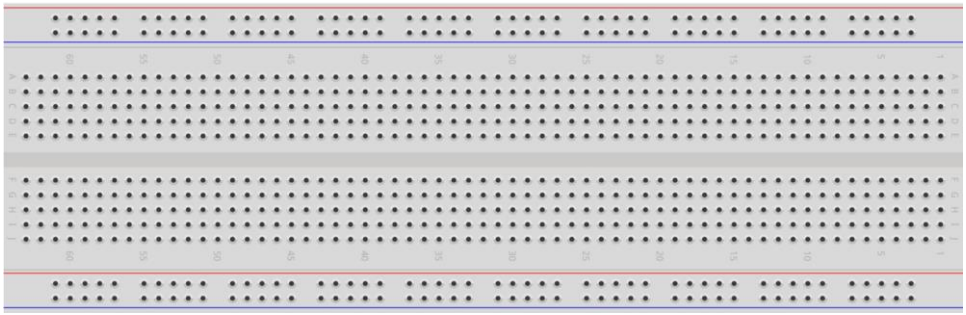
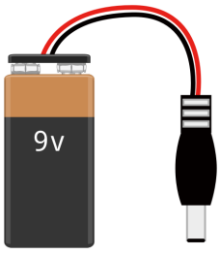
Chapter 15 Relay & Motor

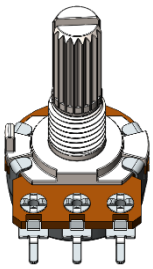
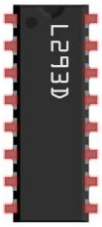
In this chapter, we will learn a kind of special switch module, relay module.

Project 15.2 Control Motor with Potentiometer

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

Component List

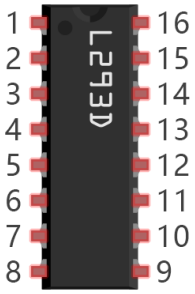
ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper M/M 	9V battery (prepared by yourself) & battery line 

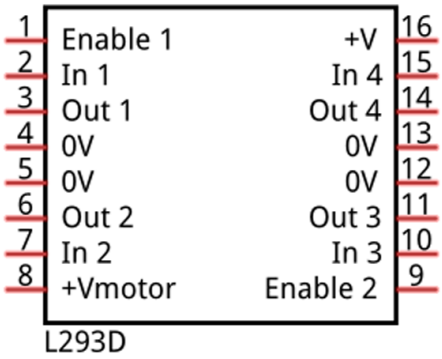
Rotary potentiometer x1	Motor x1	L293D
		

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).



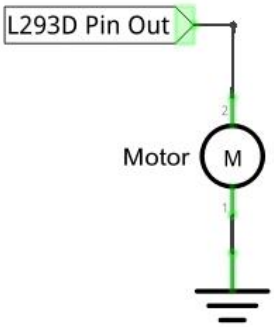


Port description of L293D module is as follows:

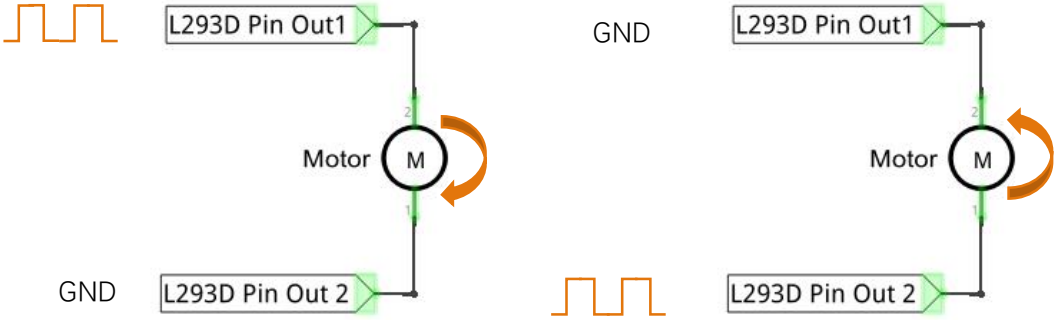
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 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 L239D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.



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

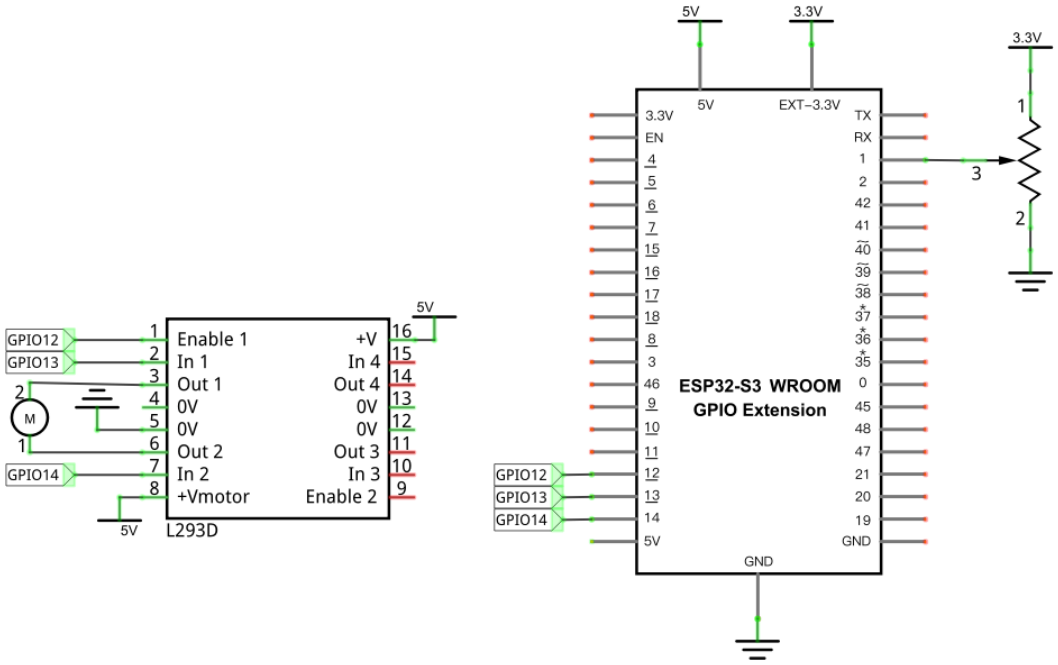


In practical use the motor is usually connected to channel 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

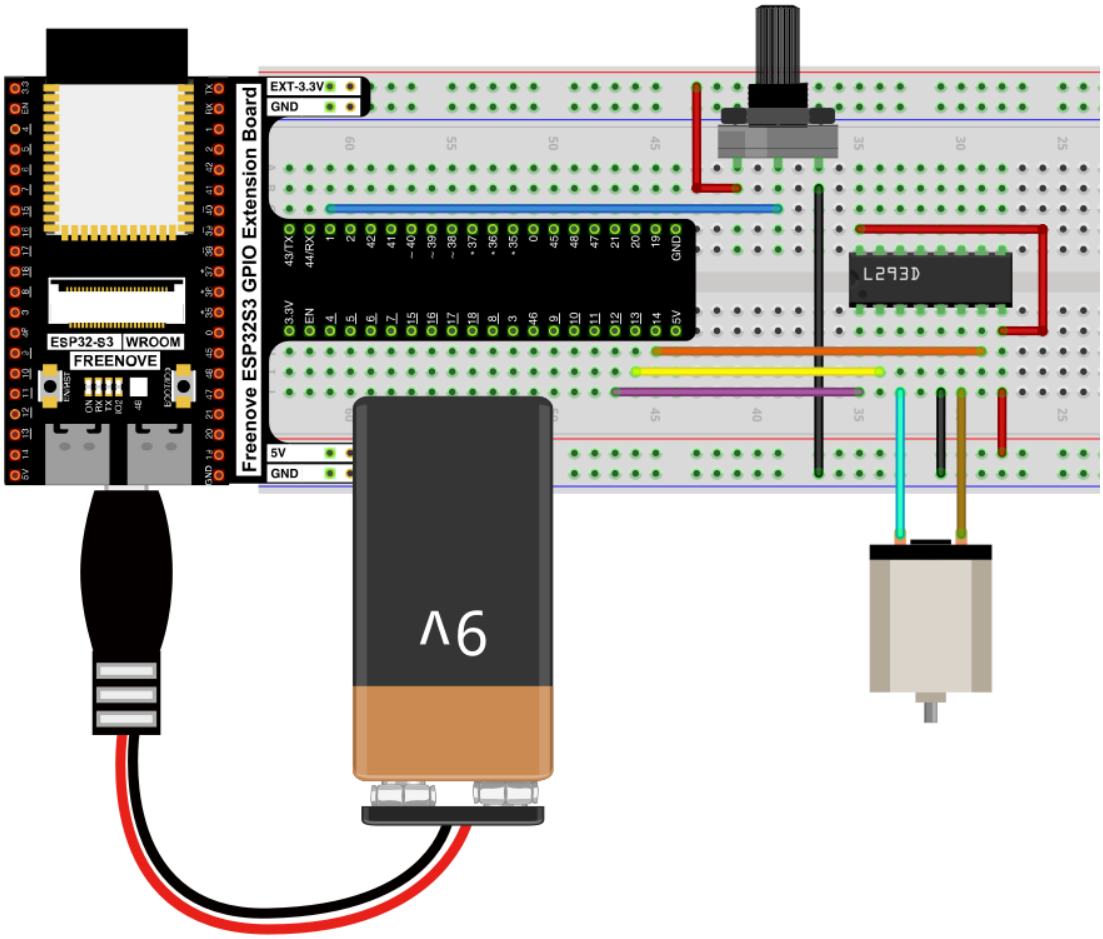
Circuit

Use caution when connecting this circuit, because the DC motor is a high-power component, do not use the power provided by the ESP32-S3 to power the motor directly, which may cause permanent damage to your ESP32-S3! The logic circuit can be powered by the ESP32-S3 power or an external power supply, which should share a common ground with ESP32-S3.

Schematic diagram



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



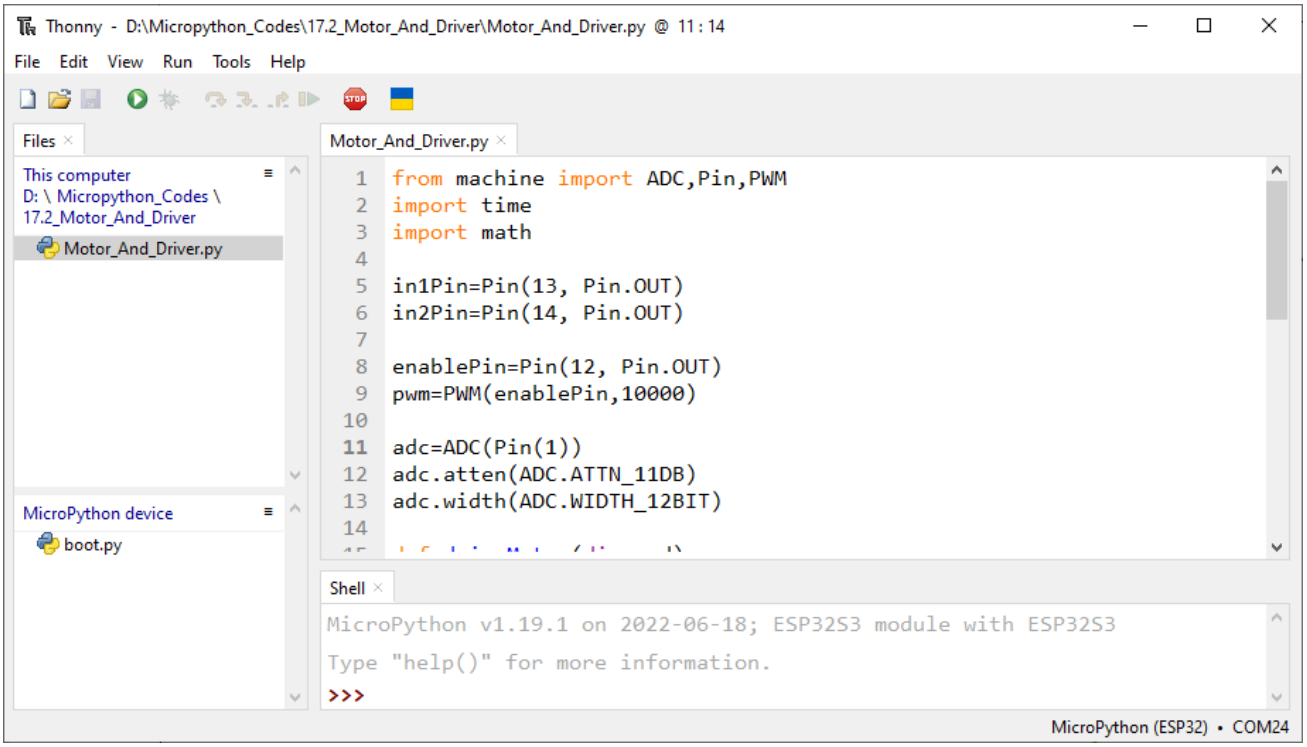
Note: the motor circuit uses A large current, about 0.2-0.3A without load.We recommend that you use a 9V battery to power the extension board.

Code

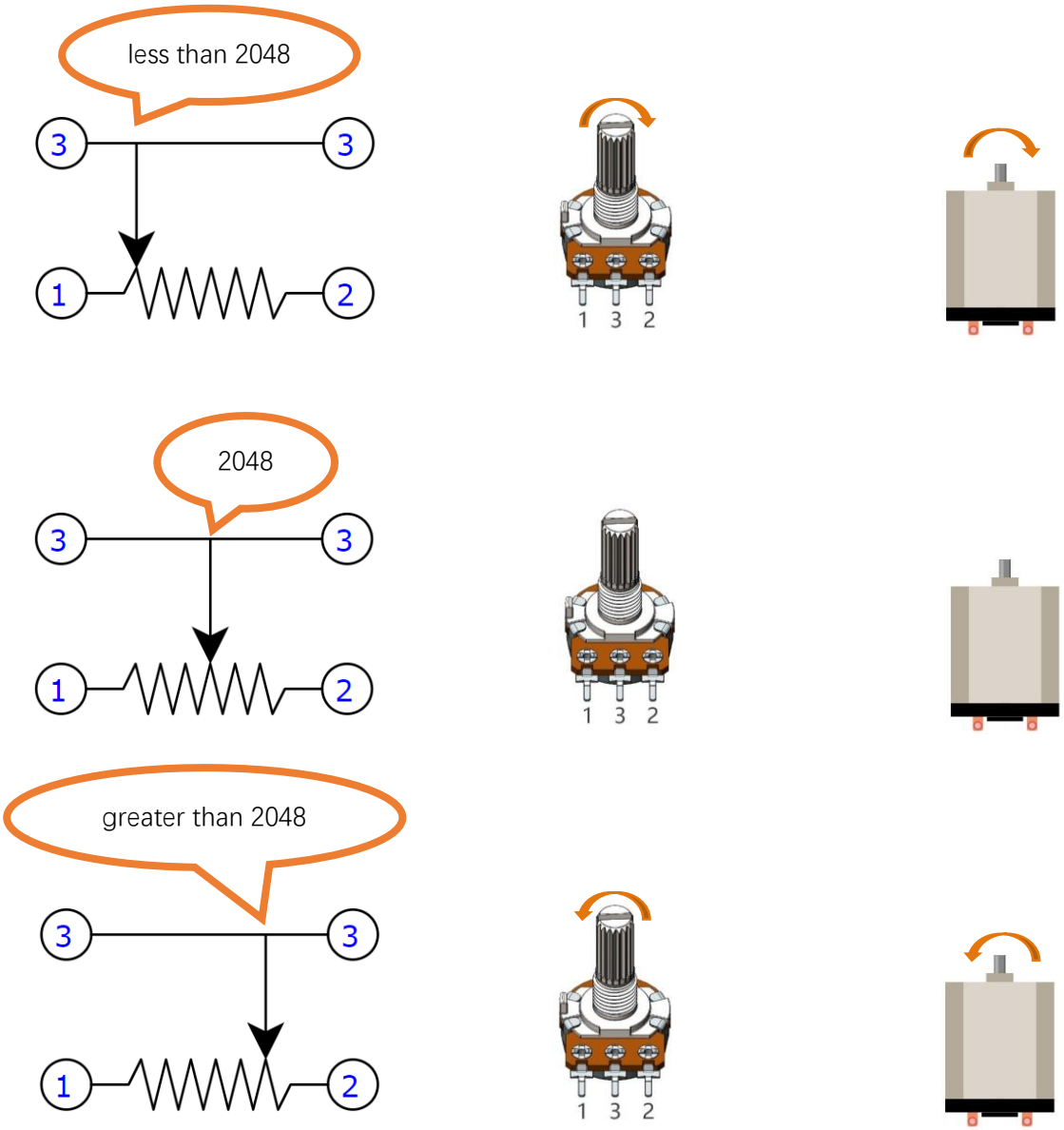
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "15.2_Motor_And_Driver" and double click "Motor_And_Driver.py".

15.2_Motor_And_Driver



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.


less than 2048
3
3
1
2
1 3 2
2048
3
3
1
2
1 3 2
greater than 2048
3
3
1
2
1 3 2

The following is the Code:

```
from machine import ADC,Pin,PWM
import time
import math

in1Pin=Pin(13, Pin.OUT)
in2Pin=Pin(14, Pin.OUT)

enablePin=Pin(12, Pin.OUT)
pwm=PWM(enablePin,10000)

adc=ADC(Pin(1))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

def driveMotor(dir,spd):
    if dir :
        in1Pin.value(1)
        in2Pin.value(0)
    else :
        in1Pin.value(0)
        in2Pin.value(1)
    pwm.duty(spd)

try:
    while True:
        potenVal = adc.read()
        rotationSpeed = potenVal - 2048
        if (potenVal > 2048):
            rotationDir = 1;
        else:
            rotationDir = 0;
        rotationSpeed=int(math.fabs((potenVal-2047)//2)-1)
        driveMotor(rotationDir,rotationSpeed)
        time.sleep_ms(10)
except:
    pass
```

The ADC of ESP32-S3 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(13, Pin.OUT)
6   in2Pin=Pin(14, Pin.OUT)
7
8   enablePin=Pin(12, Pin.OUT)
9   pwm=PWM(enablePin,10000)
```

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(1))
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)
```

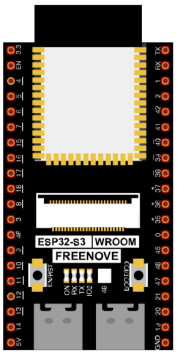
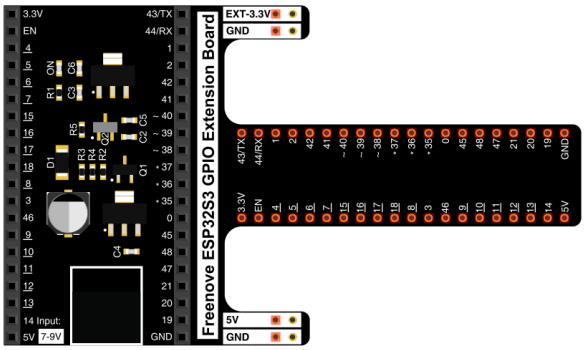
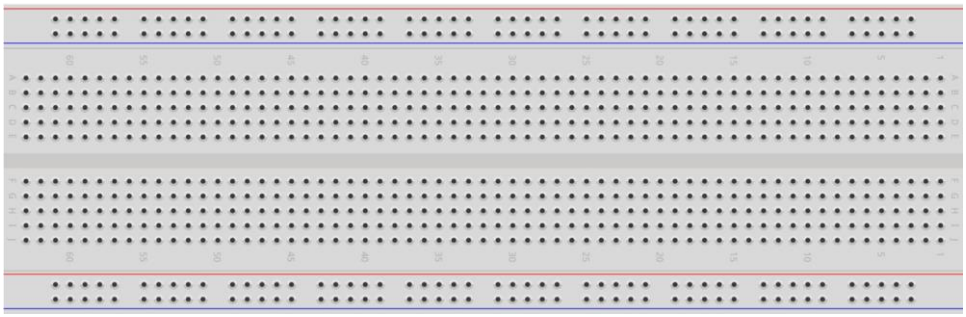
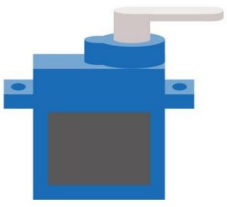
Chapter 16 Servo

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 16.1 Servo Sweep

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

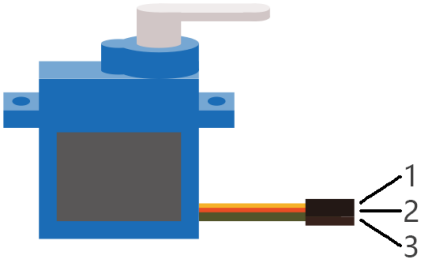
Component List

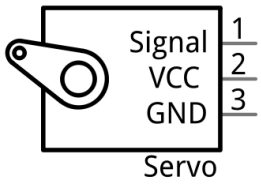
ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
Servo x1 	Jumper M/M x3 

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 lasting time 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:

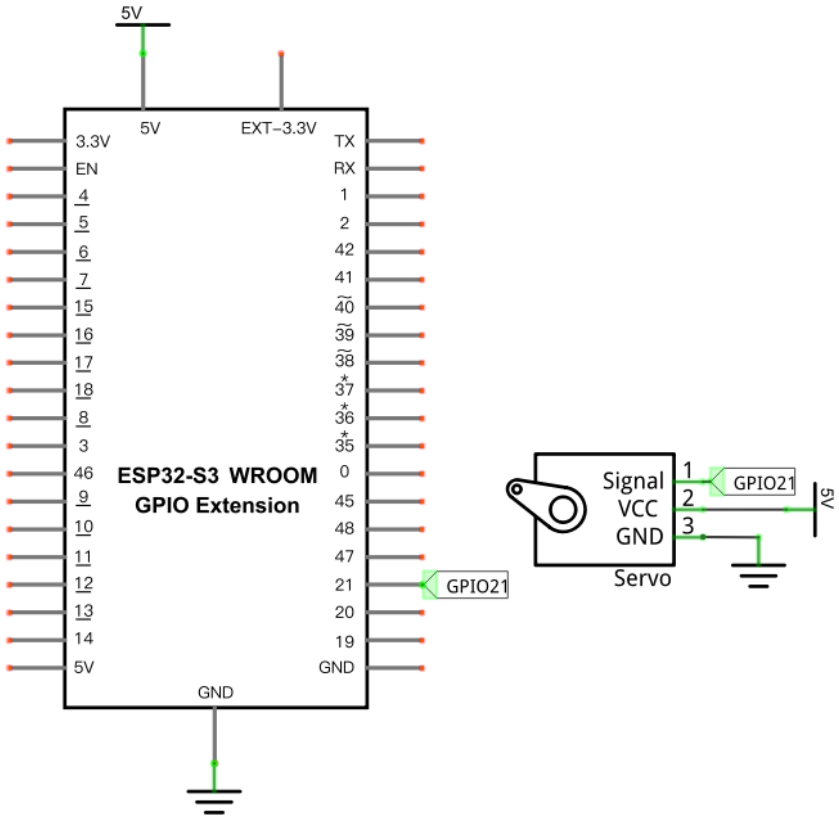
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

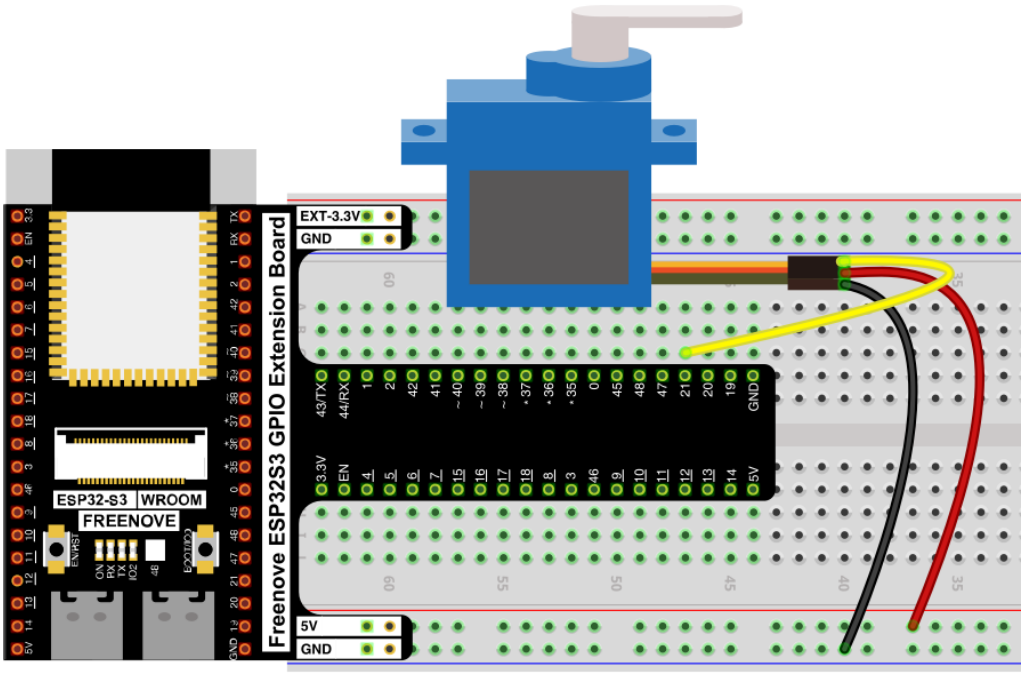
Circuit

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

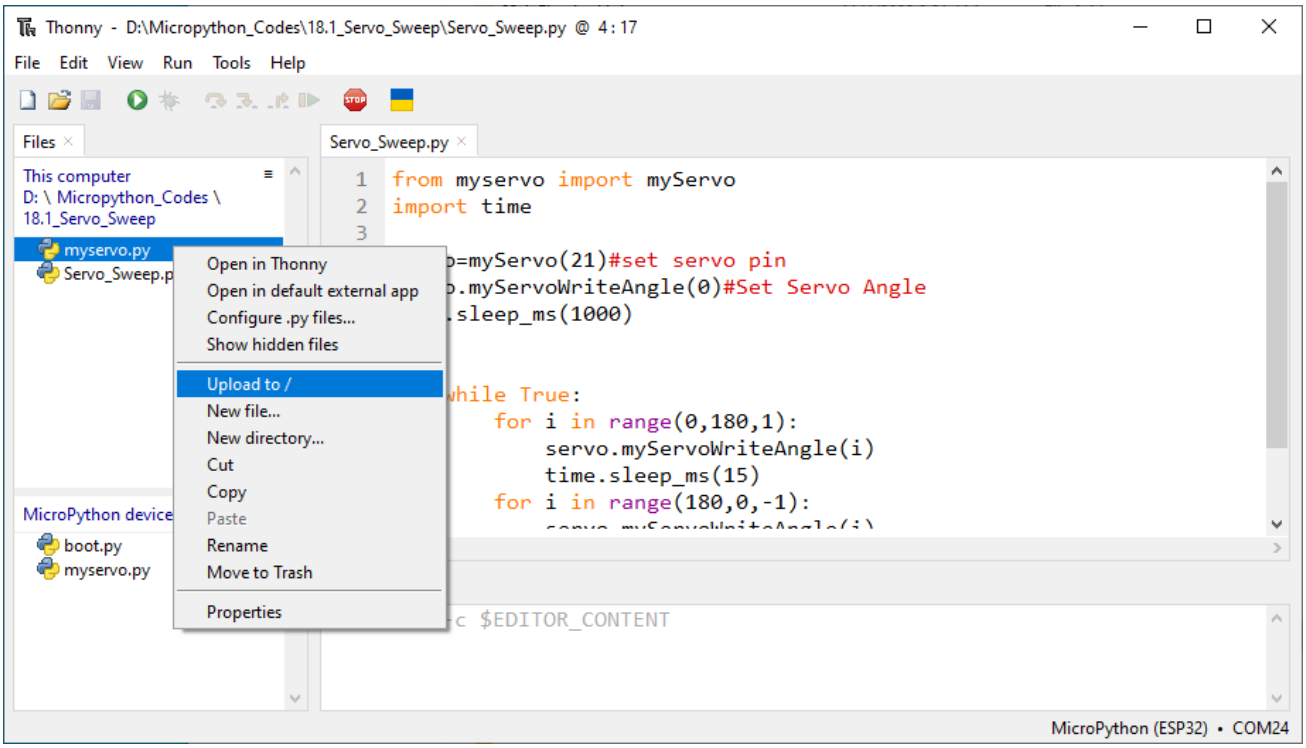


Code

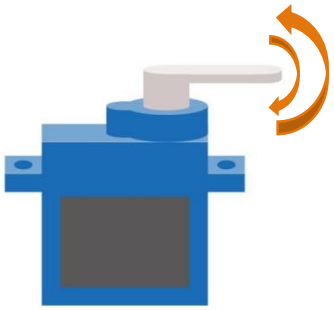
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "16.1_Servo_Sweep". Select "myservo.py", right click your mouse to select "Upload to /", wait for "myservo.py" to be uploaded to ESP32-S3 and then double click "Servo_Sweep.py".

16.1_Servo_Sweep



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:

```
from myservo import myServo
import time

servo=myServo(21) #Set Servo Pin
servo.myServoWriteAngle(0) #Set Servo Angle
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()
```

Import myservo module.

```
from myservo import myServo
```

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

```
servo=myServo(21)
servo.myServoWriteAngle(0)
time.sleep_ms(1000)
```

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

```
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.

```
        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)
```

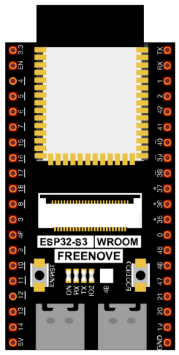
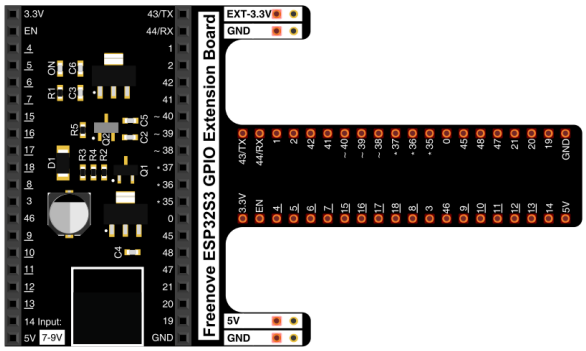
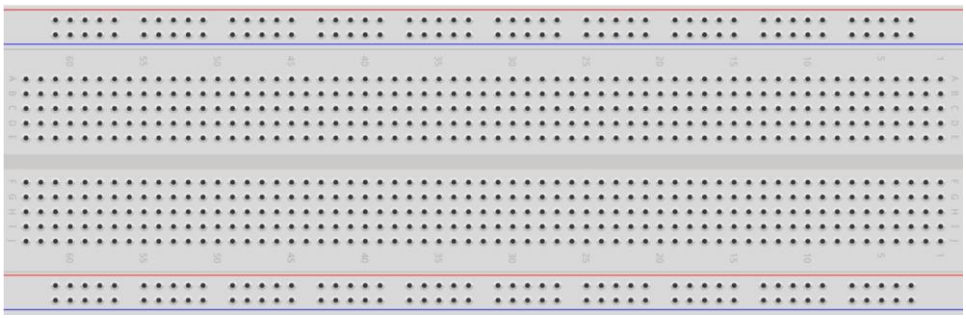
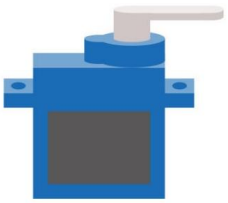
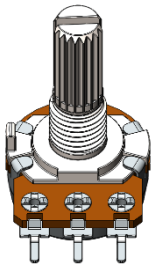
Reference

class myServo	
Before each use of **myServo**, please make sure myservo.py has been uploaded to "/" of ESP32-S3, 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 16.2 Servo Knop

Use a potentiometer to control the servo motor to rotate at any angle.

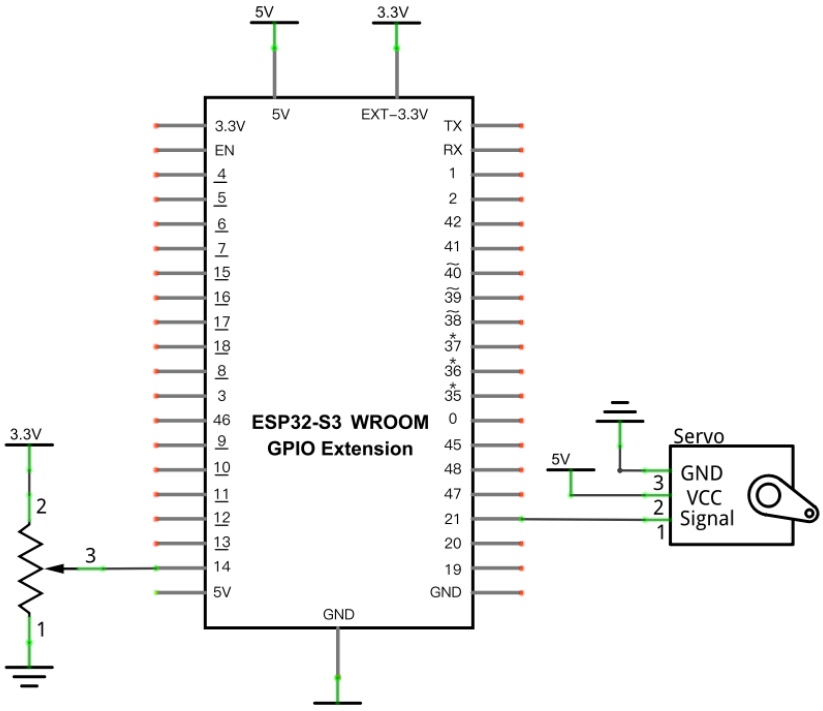
Component List

<table>
<tr><td>ESP32-S3 WROOM x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Servo x1
  </td><td>Jumper M/M x6
  </td><td>Rotary potentiometer x1
  </td></tr>
</table>

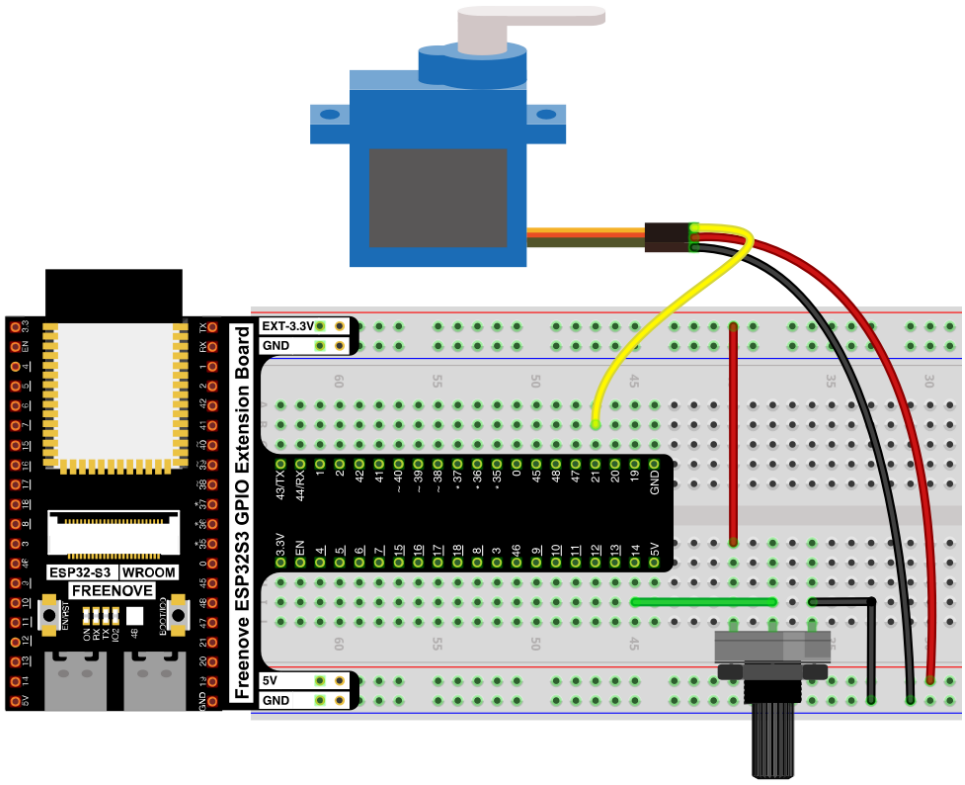
Circuit

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

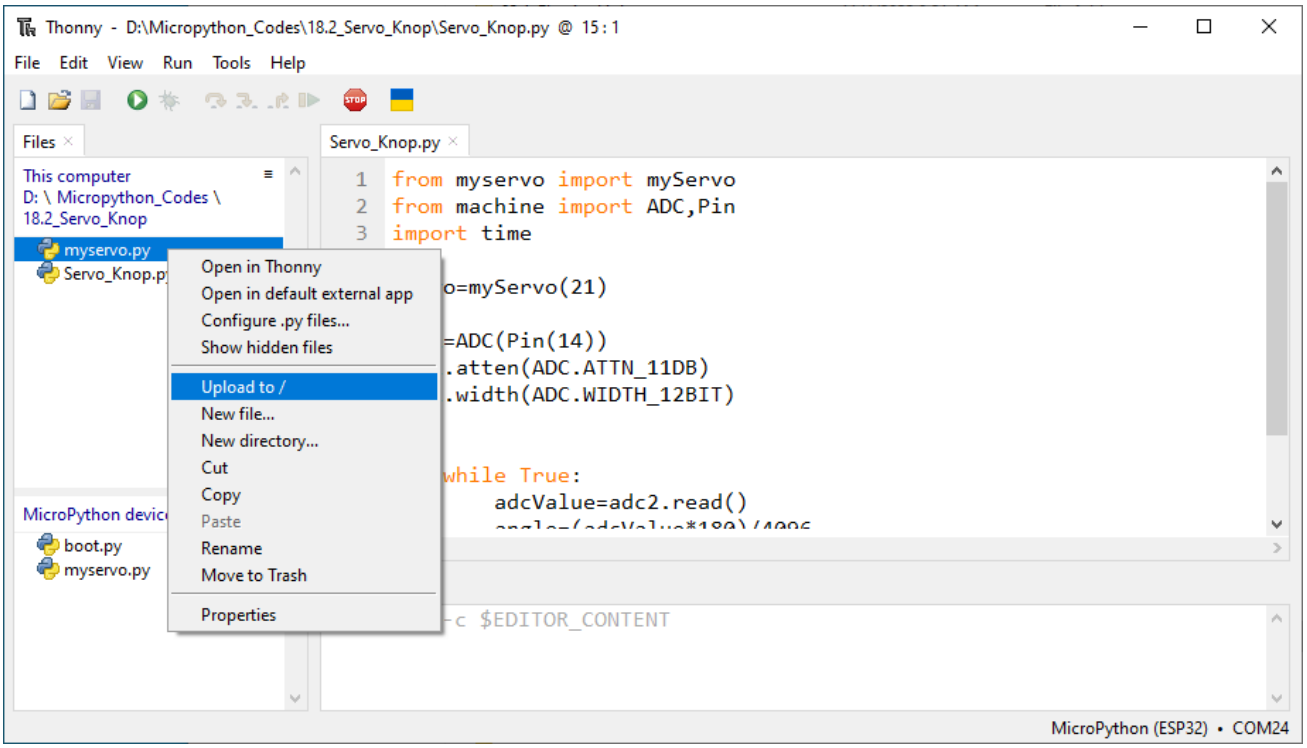


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "16.2_Servo_Knop". Select "myservo.py", right click your mouse to select "Upload to /", wait for "myservo.py" to be uploaded to ESP32-S3 and then double click "Servo_Knop.py".

16.2_Servo_Knop



Click "Run current script", twist the potentiometer back and forth, and the servo motor rotates accordingly.

The following is the program code:

```
from myservo import myServo
from machine import ADC,Pin
import time

servo=myServo(21)

adc2=ADC(Pin(14))
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()
```

In this project, we will use Pin(14) of ESP32-S3 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.

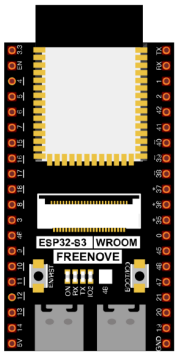
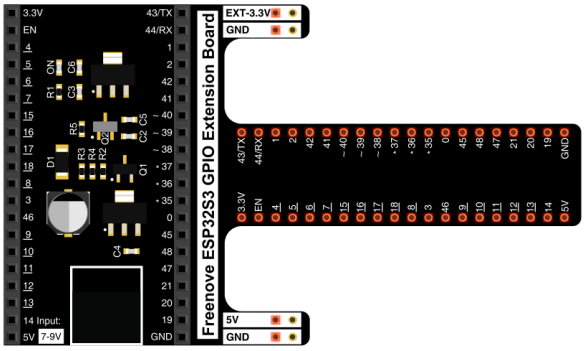
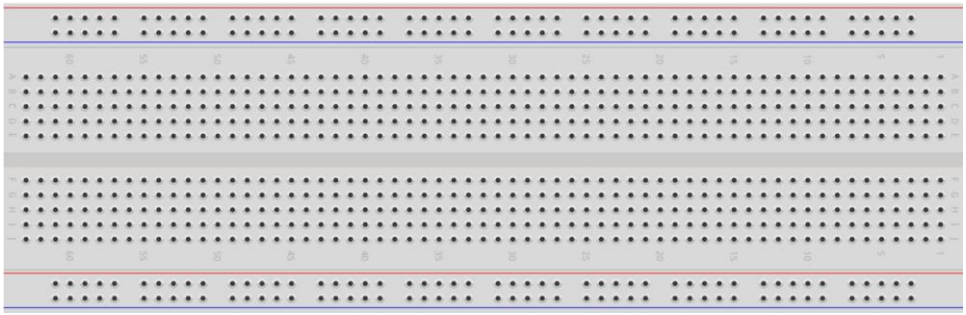
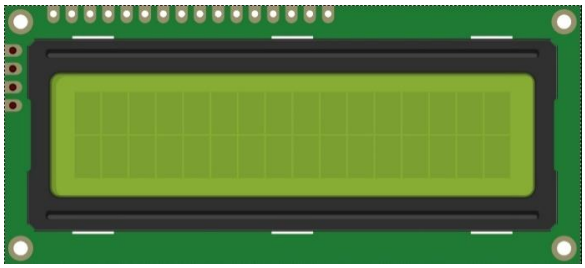
Chapter 17 LCD1602

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

Project 17.1 LCD1602

In this section we learn how to use LCD1602 to display something.

Component List

ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
LCD1602 Module x1 	Jumper F/M x4 

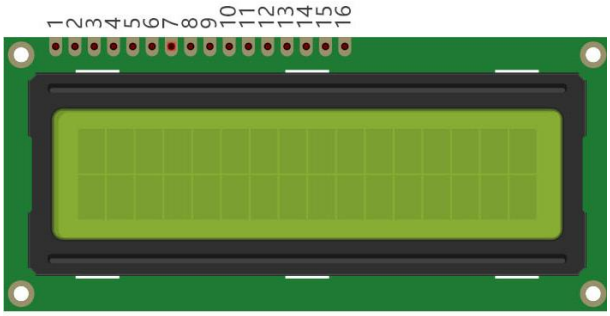
Component knowledge

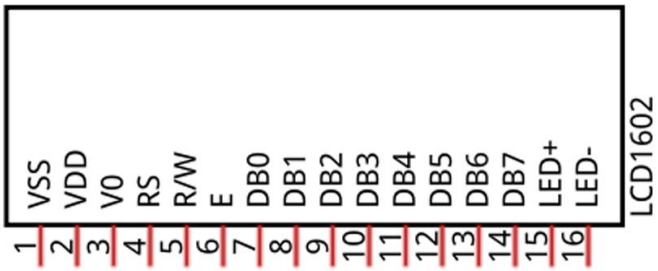
I2C communication

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.

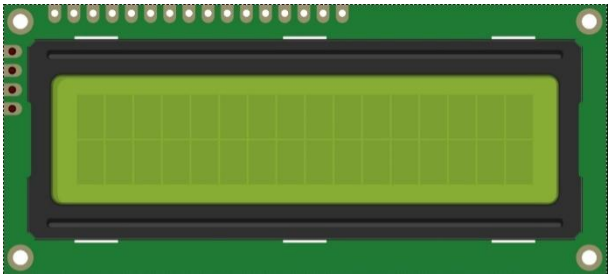
LCD1602 communication'

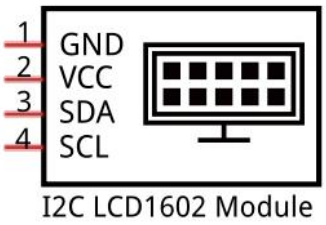
The LCD1602 display screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 display screen along with its circuit pin diagram





I2C LCD1602 display screen integrates a I2C interface, which connects the serial-input & parallel-output module to the LCD1602 display screen. This allows us to only use 4 lines to the operate the LCD1602.

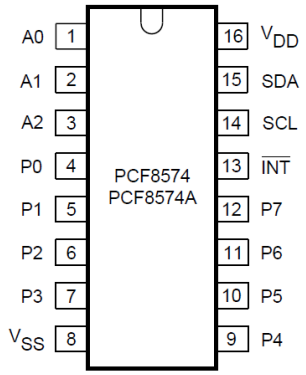




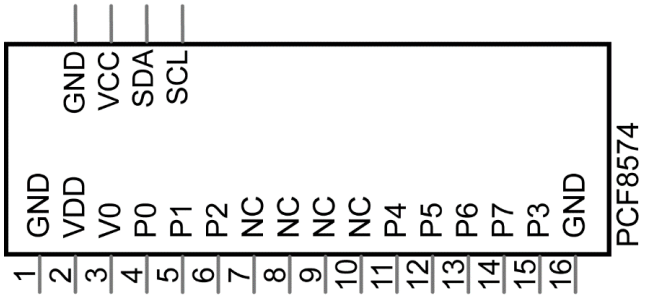
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

Below is the PCF8574 pin schematic diagram and the block pin diagram:

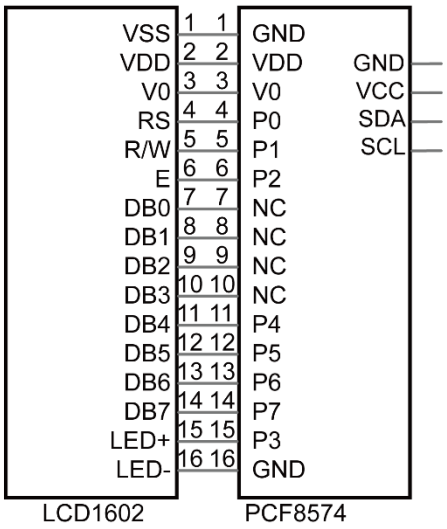
PCF8574 chip pin diagram:



PCF8574 module pin diagram



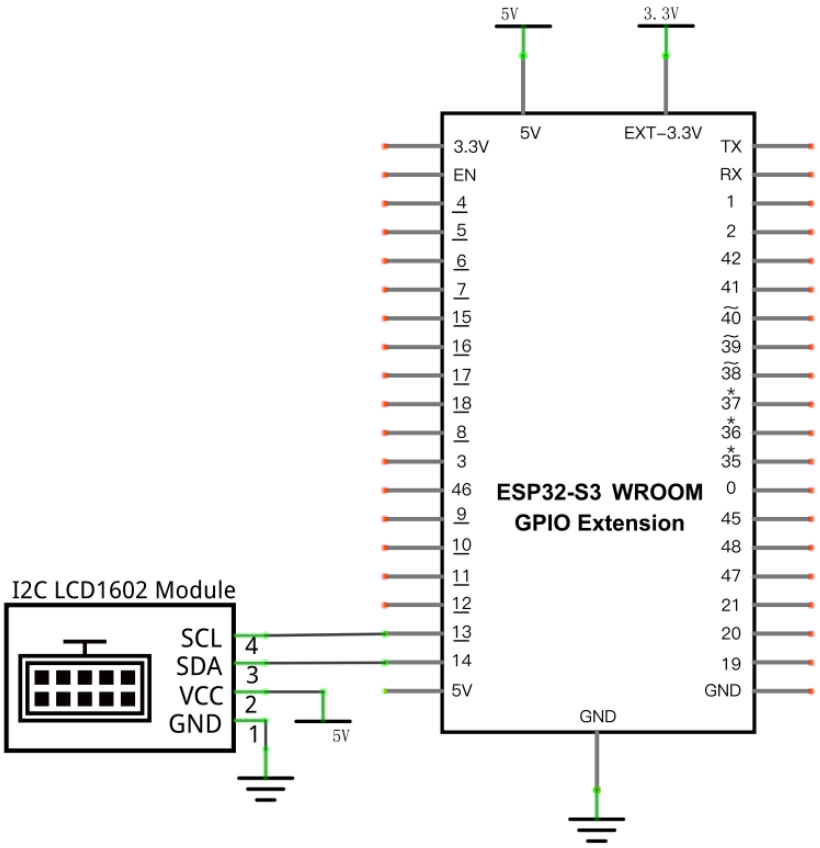
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



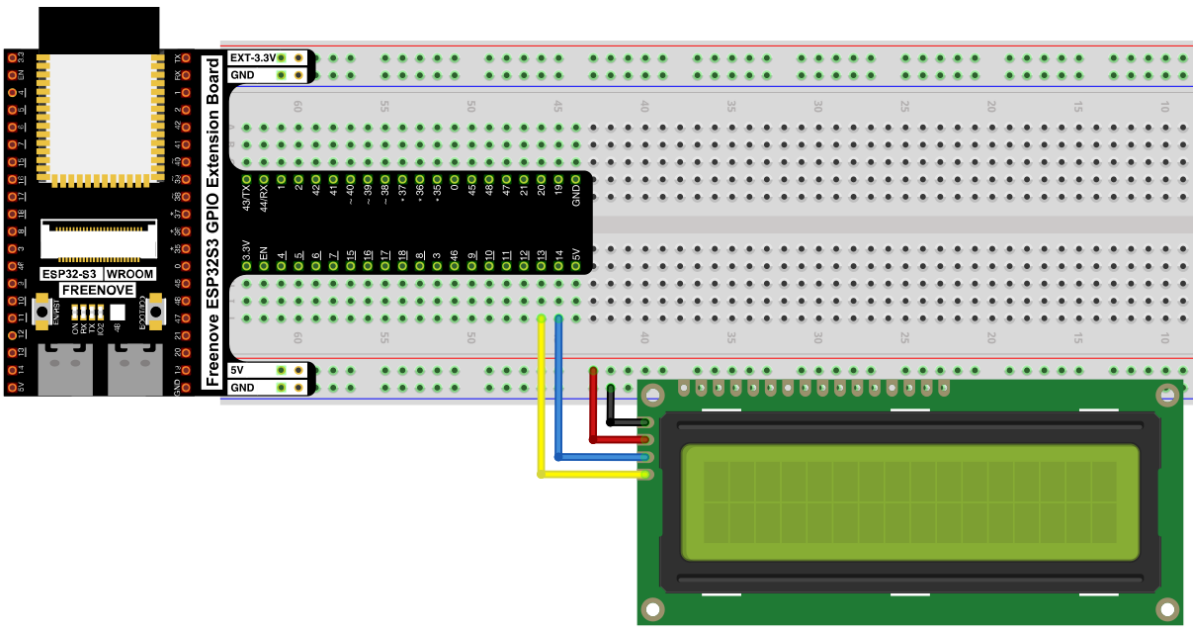
So we only need 4 pins to control the 16 pins of the LCD1602 display screen through the I2C interface.
In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



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

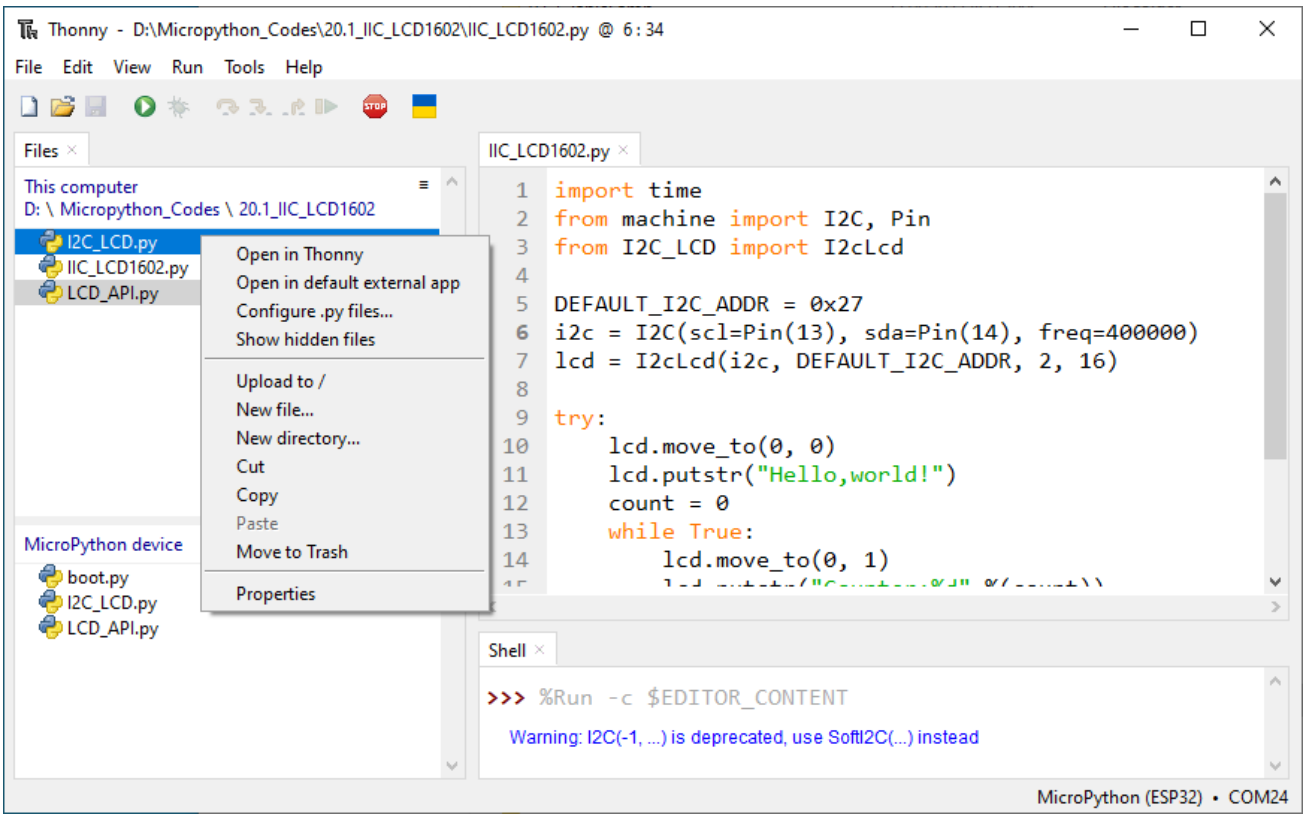


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "17.1_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-S3 and then double click "I2C_LCD1602.py".

17.1_I2C_LCD1602



Click "Run current script" and LCD1602 displays some characters.

The following is the program code:

```
import time
from machine import I2C, Pin
from I2C_LCD import I2cLcd

DEFAULT_I2C_ADDR = 0x27
i2c = I2C(scl=Pin(13), sda=Pin(14), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

try:
    lcd.move_to(0, 0)
    lcd.putstr("Hello,world!")
    count = 0
    while True:
        lcd.move_to(0, 1)
        lcd.putstr("Counter:%d" %(count))
        time.sleep_ms(1000)
        count += 1
except:
    pass
```

Import time, I2C and I2C_LCD modules.

```
import time
from machine import I2C, Pin
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.

```
DEFAULT_I2C_ADDR = 0x27
```

Initialize I2C pins and associate them with I2CLCD module, and then set the number of rows and columns for LCD1602.

```
i2c = I2C(scl=Pin(13), sda=Pin(14), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
```

Move the cursor of LCD1602 to the first row, first column, and print out "Hello, world!"

```
    lcd.move_to(0, 0)
    lcd.putstr("Hello,world!")
```

The second line of LCD1602 continuously prints the number of seconds after the ESP32-S3 program runs.

```
    while True:
        lcd.move_to(0, 1)
        lcd.putstr("Counter:%d" %(count))
        time.sleep_ms(1000)
        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 ESP32S3, 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.

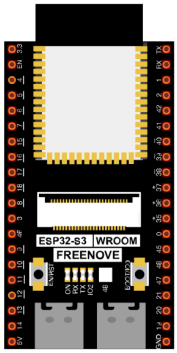
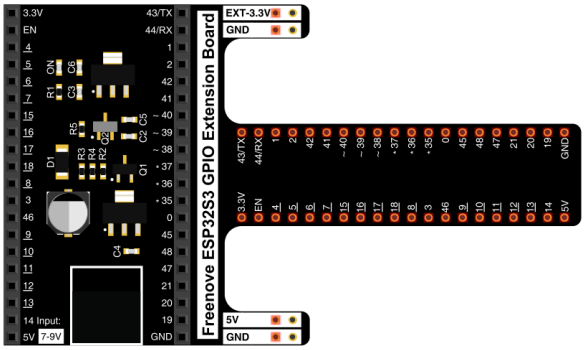
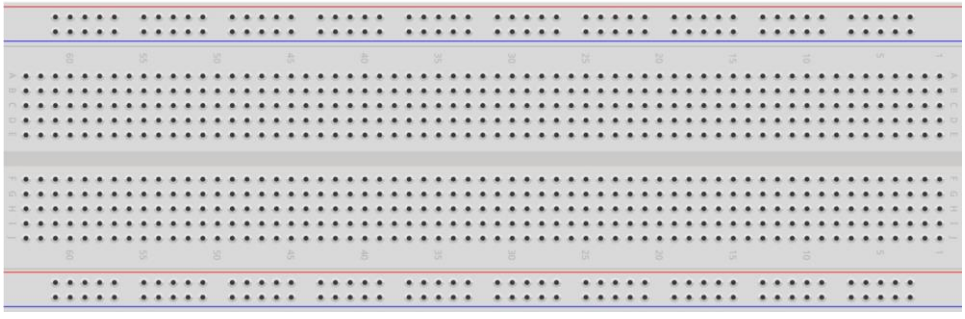
Chapter 18 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 18.1 Ultrasonic Ranging

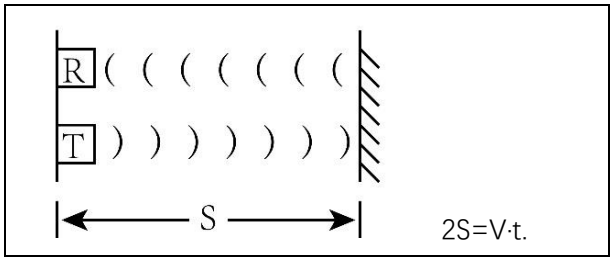
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

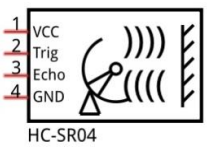
ESP32-S3 WROOM x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper F/M x4 	HC SR04 x1 

Component Knowledge

The ultrasonic ranging module uses the principle that ultrasonic waves will be sent back when encounter obstacles. We can measure the distance by counting the time interval between sending and receiving of the ultrasonic waves, and the time difference 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, about v=340m/s, we can calculate the distance between the ultrasonic ranging module and the obstacle: s=vt/2.



The HC-SR04 ultrasonic ranging module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 ultrasonic ranging module are shown below:



Pin description:

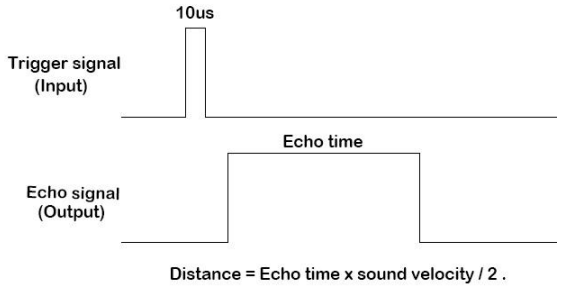
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

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

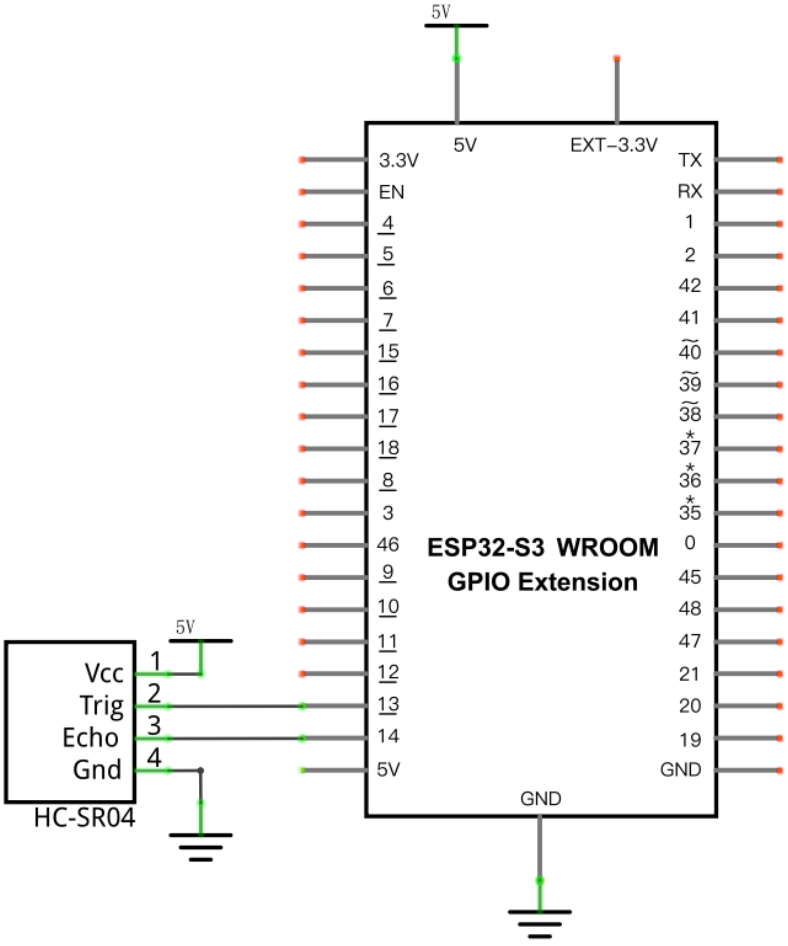
Instructions for use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2.



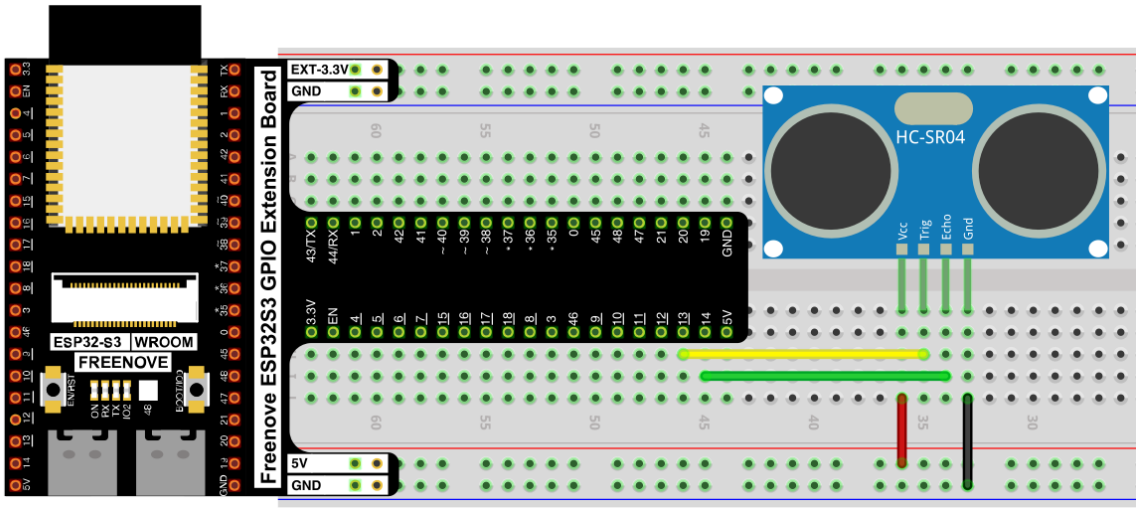
Circuit

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

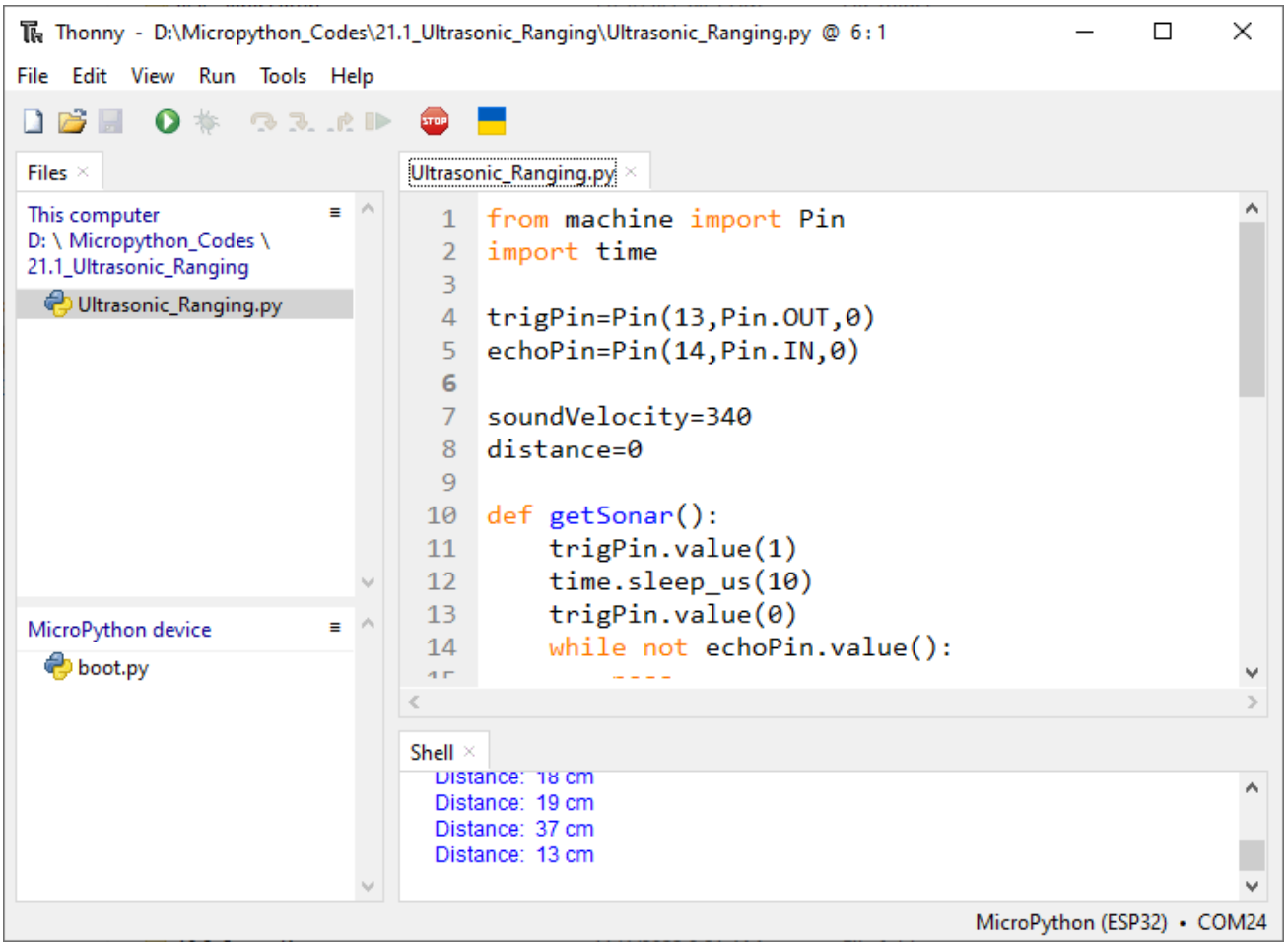


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "18.1_Ultrasonic_Ranging" and double click "Ultrasonic_Ranging.py".

18.1_Ultrasonic_Ranging



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:

```
>>> %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:

```
from machine import Pin
import time

trigPin=Pin(13,Pin.OUT,0)
echoPin=Pin(14,Pin.IN,0)

soundVelocity=340
distance=0

def getSonar():
    trigPin.value(1)
    time.sleep_us(10)
    trigPin.value(0)
    while not echoPin.value():
        pass
    pingStart=time.ticks_us()
    while echoPin.value():
        pass
    pingStop=time.ticks_us()
    pingTime=time.ticks_diff(pingStop,pingStart)
    distance=pingTime*soundVelocity//2//10000
    return int(distance)

time.sleep_ms(2000)
while True:
    time.sleep_ms(500)
    print('Distance: ',getSonar(),'cm')
```

Define the control pins of the ultrasonic ranging module.

```
trigPin=Pin(13,Pin.OUT,0)
echoPin=Pin(14,Pin.IN,0)
```

Set the speed of sound.

```
soundVelocity=340
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.

```
def getSonar():
    trigPin.value(1)
    time.sleep_us(10)
    trigPin.value(0)
    while not echoPin.value():
        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 18.2 Ultrasonic Ranging

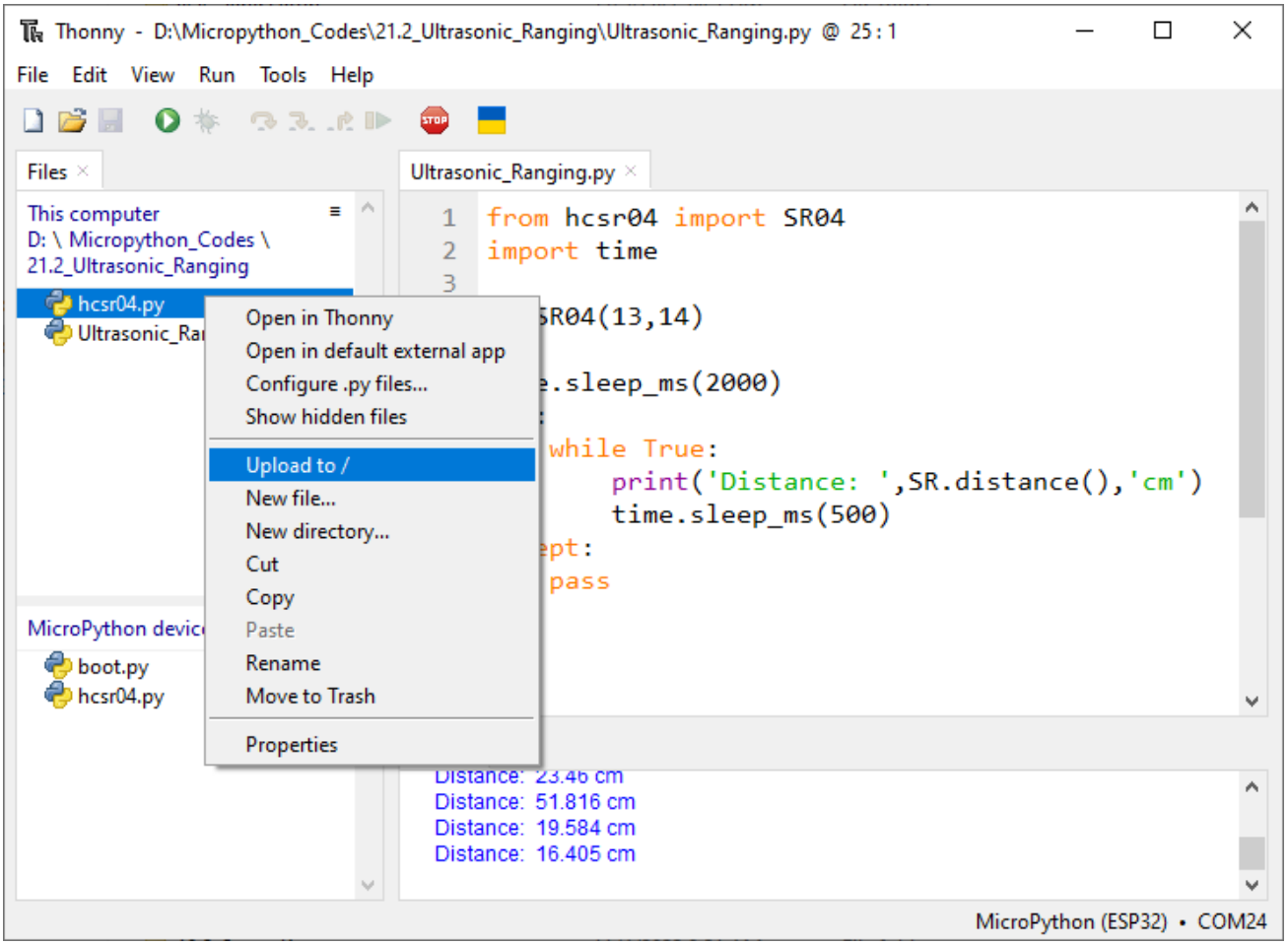
Component List and Circuit

Component List and Circuit are the same as the previous section.

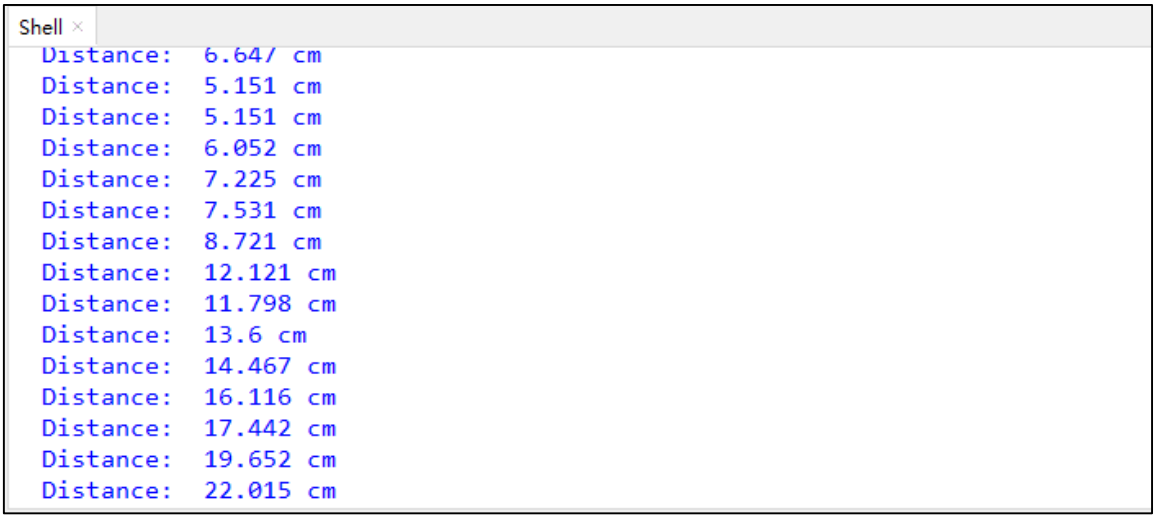
Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "18.2_Ultrasonic_Ranging". Select "hcsr04.py", right click your mouse to select "Upload to /", wait for "hcsr04.py" to be uploaded to ESP32-S3 and then double click "Ultrasonic_Ranging.py".

18.2_Ultrasonic_Ranging



Click "Run current script". Use the ultrasonic module to measure distance. As shown in the following figure:



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.

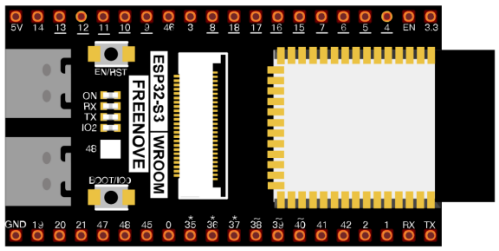
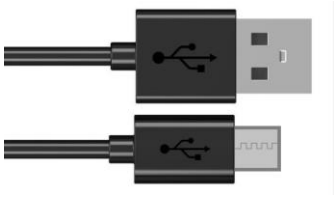
distance(): Obtain the distance from the ultrasonic to the measured object with the data type being float type, and the unit being cm.

Chapter 19 Bluetooth

This chapter mainly introduces how to make simple data transmission through Bluetooth of ESP32-S3 WROOM and mobile phones.

Project 19.1 Bluetooth Low Energy Data Passthrough

Component List

ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

ESP32S3's integrated Bluetooth function Bluetooth is a short-distance communication system, which can be divided into two types, namely Bluetooth Low Energy(BLE) and Classic Bluetooth. There are two modes for simple data transmission: master mode and slave mode.

Master mode

In this mode, works are done in the master device and it can connect with a slave device. And we can search and select slave devices nearby to connect with. When a device initiates connection request in master mode, it requires information of the other Bluetooth devices including their address and pairing passkey. After finishing pairing, it can connect with them directly.

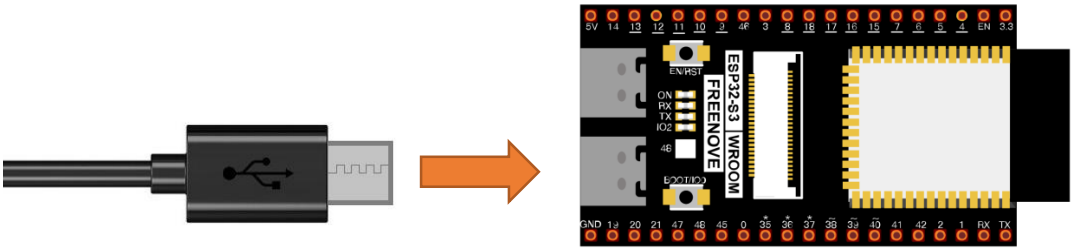
Slave mode

The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

Bluetooth devices can make data interaction with each other, as one is in master mode and the other in slave mode. When they are making data interaction, the Bluetooth device in master mode searches and selects devices nearby to connect to. When establishing connection, they can exchange data. When mobile phones exchange data with ESP32S3, they are usually in master mode and ESP32-S3 in slave mode.

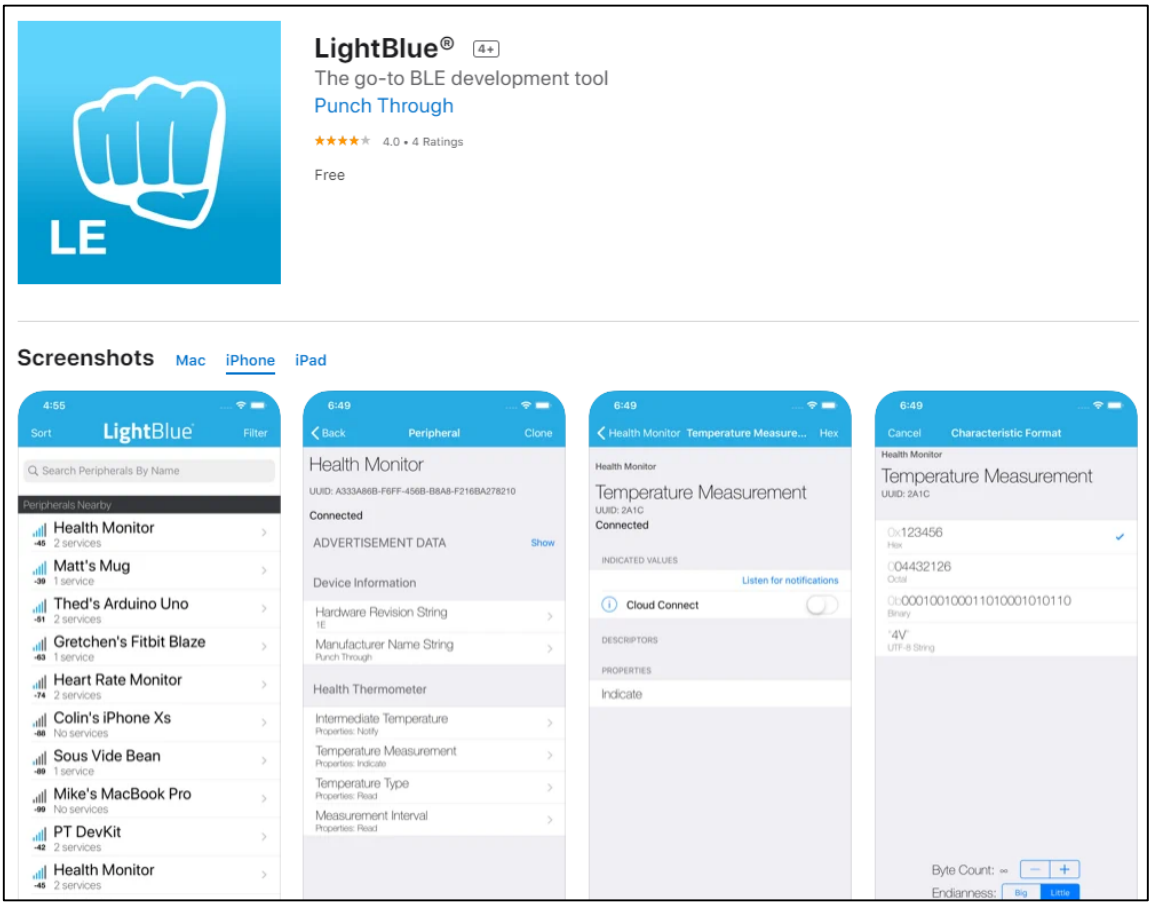
Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.



Lightblue

If you can't install Serial Bluetooth on your phone, try LightBlue.If you do not have this software installed on your phone, you can refer to this link:
https://apps.apple.com/us/app/lightblue/id557428110#?platform=iphone.

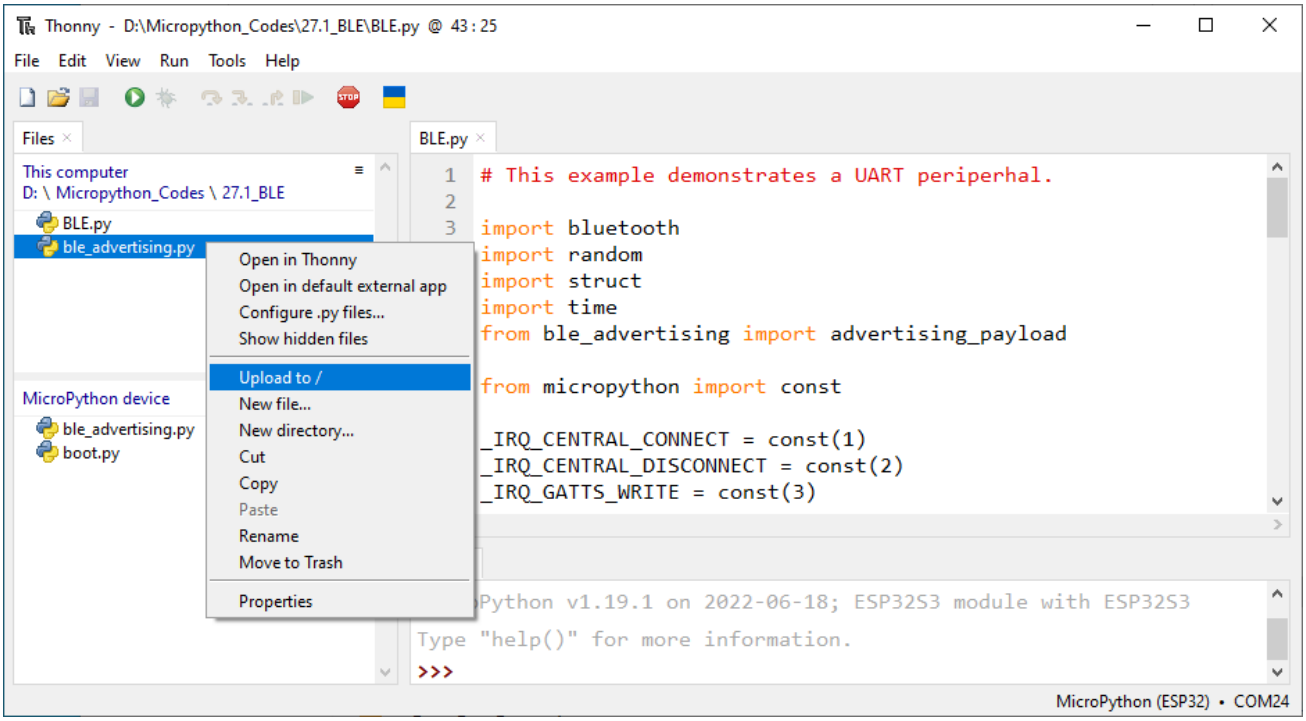


Code

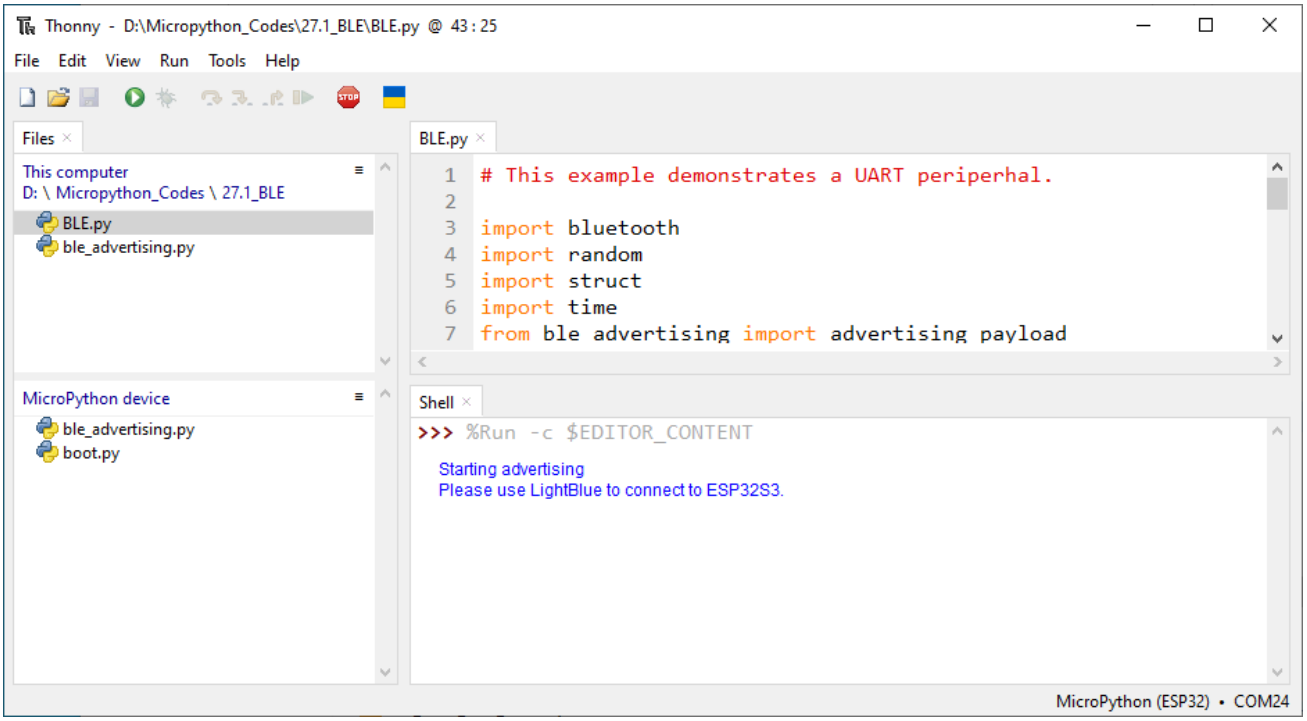
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "19.1_BLE". Select "ble_advertising.py", right click your mouse to select "Upload to /", wait for "ble_advertising.py" to be uploaded to ESP32-S3 and then double click "BLE.py".

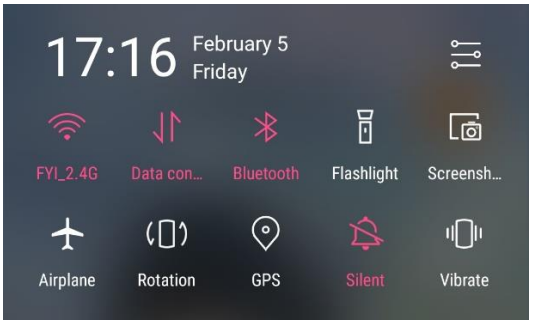
19.1_BLE



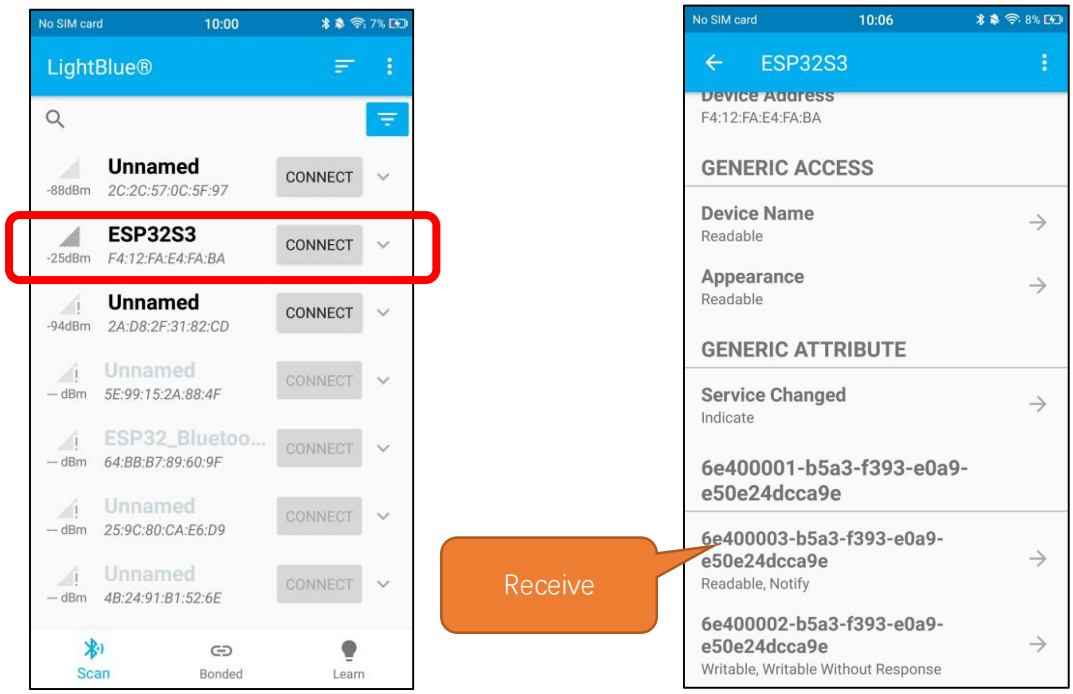
Click run for BLE.py.



Turn ON Bluetooth on your phone, and open the Lightblue APP.

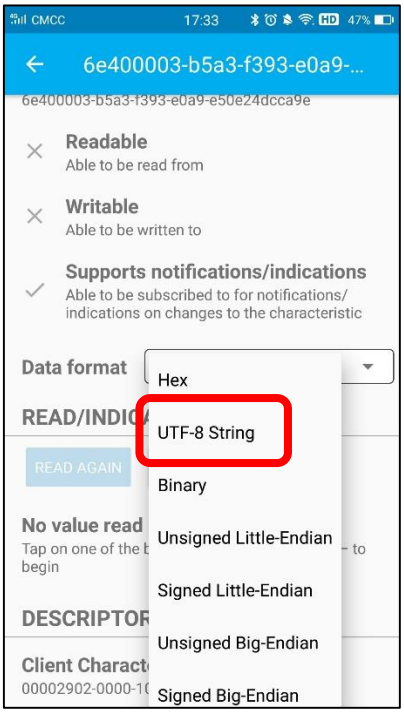


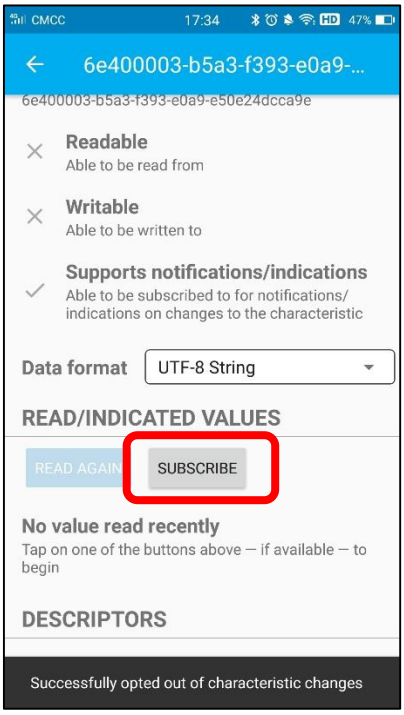
In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32S3.



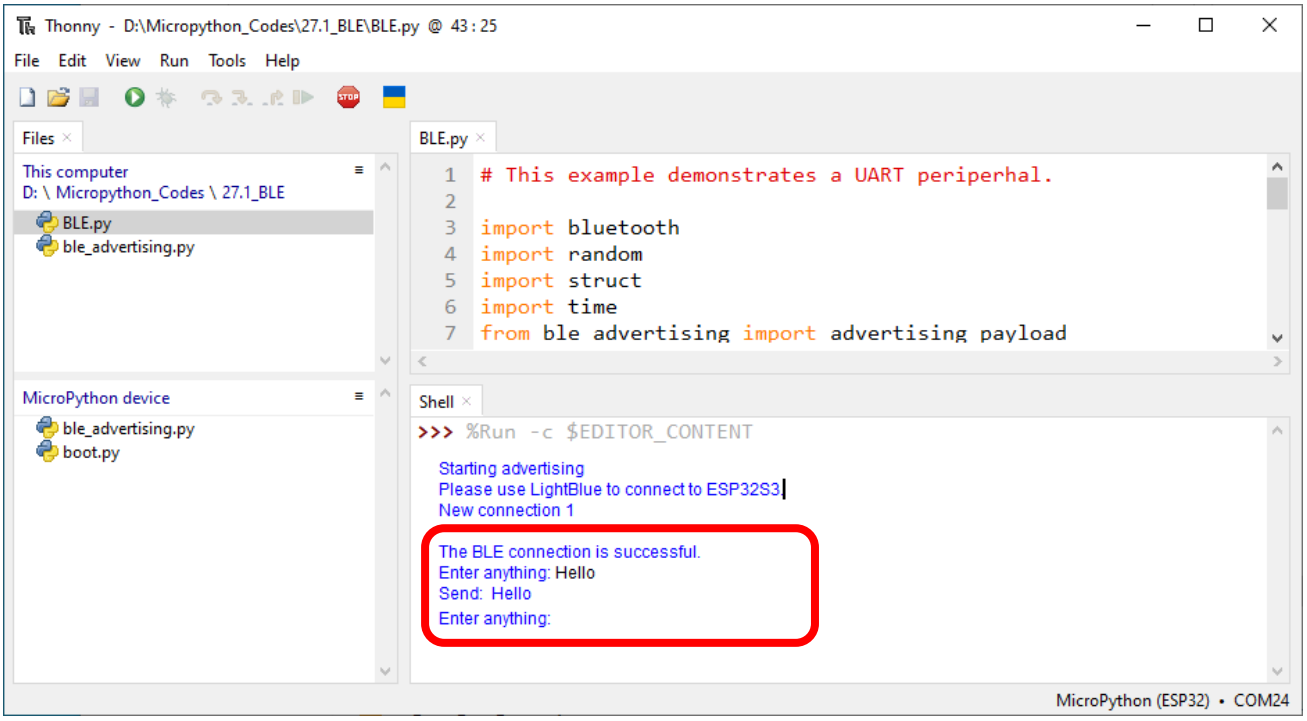
After Bluetooth is connect successfully, Shell will printer the information.

Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.

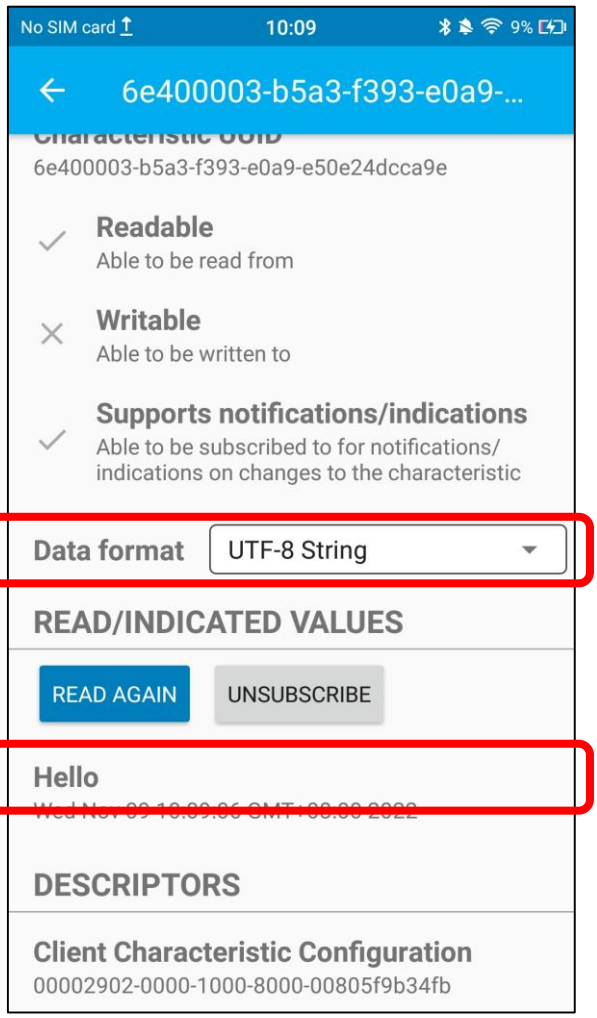




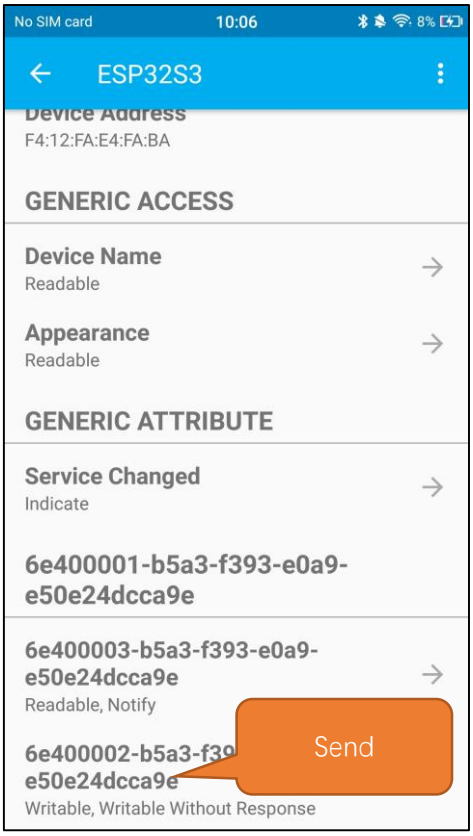
You can type "Hello" in Shell and press "Enter" to send.

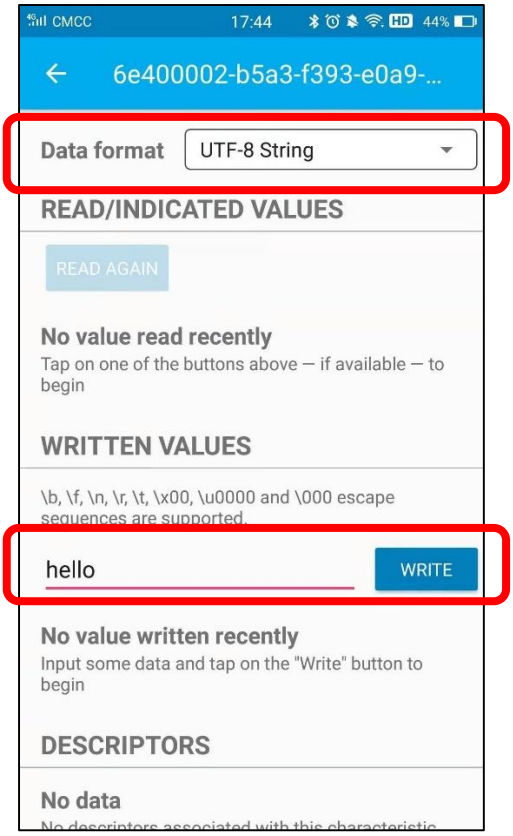


And then you can see the mobile Bluetooth has received the message.

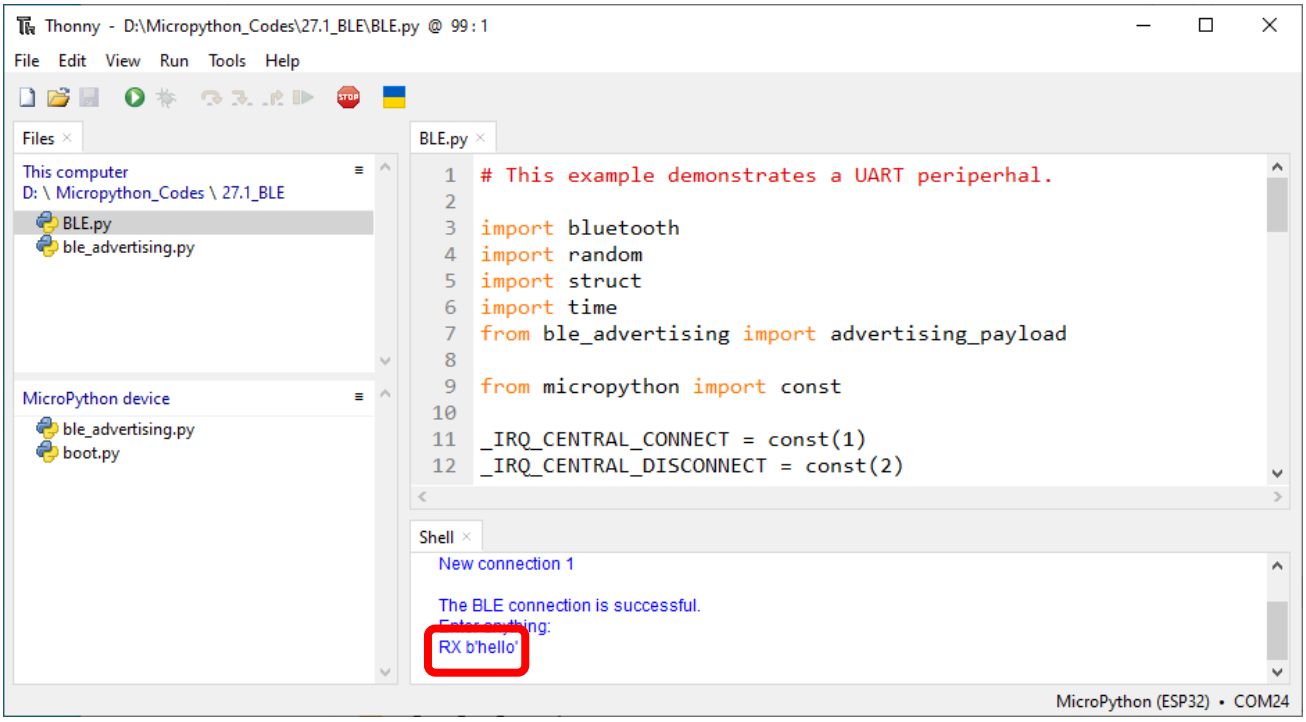


Similarly, you can select "Send" on your phone. Set Data format, and then enter anything in the sending box and click Write to send.





You can check the message from Bluetooth in "Shell".



And now data can be transferred between your mobile phone and computer via ESP32S3.
The following is the program code:

```
import bluetooth
import random
import struct
import time
from ble_advertising import advertising_payload
from micropython import const

_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_WRITE = const(3)
_FLAG_READ = const(0x0002)
_FLAG_WRITE_NO_RESPONSE = const(0x0004)
_FLAG_WRITE = const(0x0008)
_FLAG_NOTIFY = const(0x0010)

_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
_UART_SERVICE = (
    _UART_UUID,
    (_UART_TX, _UART_RX),
)
class BLESimplePeripheral:
    def __init__(self, ble, name="ESP32S3"):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
self._ble.gatts_register_services((_UART_SERVICE,))
        self._connections = set()
        self._write_callback = None
        self._payload = advertising_payload(name=name, services=[_UART_UUID])
        self._advertise()
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
```

```
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
    def send(self, data):
        for conn_handle in self._connections:
            self._ble.gatts_notify(conn_handle, self._handle_tx, data)
    def is_connected(self):
        return len(self._connections) > 0
    def _advertise(self, interval_us=500000):
        print("Starting advertising")
        self._ble.gap_advertise(interval_us, adv_data=self._payload)
    def on_write(self, callback):
        self._write_callback = callback
def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32S3.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
if __name__ == "__main__":
    demo()
```

Define the specified UUID number for BLE vendor.

```
_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
```

```
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
```

Write an _irq function to manage BLE interrupt events.

```
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
```

Initialize the BLE function and name it.

```
def __init__(self, ble, name="ESP32S3"):
```

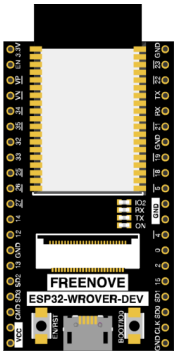
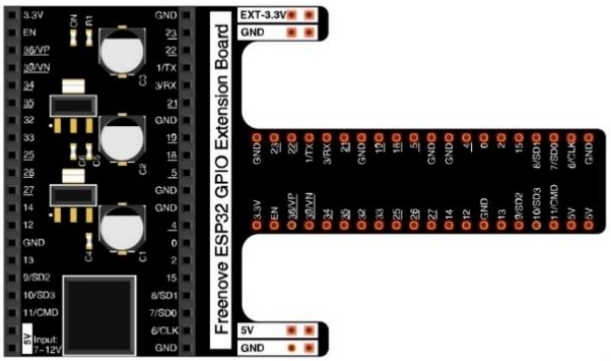
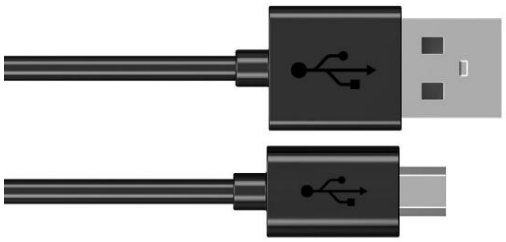
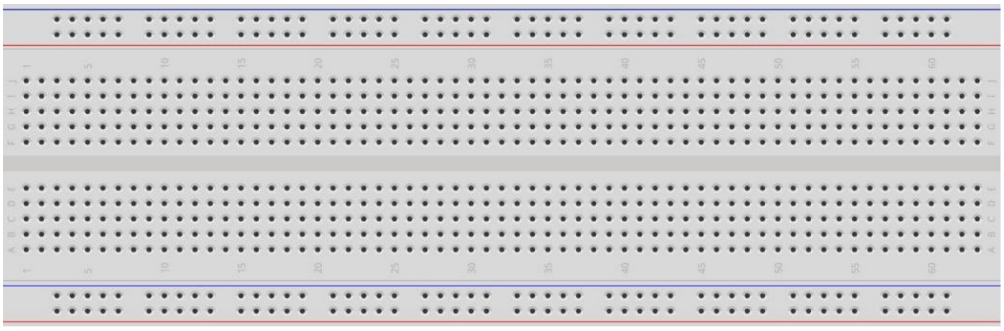
When the mobile phone send data to ESP32-S3 via BLE Bluetooth, it will print them out with serial port;
When the serial port of ESP32-S3 receive data, it will send them to mobile via BLE Bluetooth.

```
  def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32S3.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
    lastMsg = now;
  }
```

Project 19.2 Bluetooth Control LED

In this section, we will control the LED with Bluetooth.

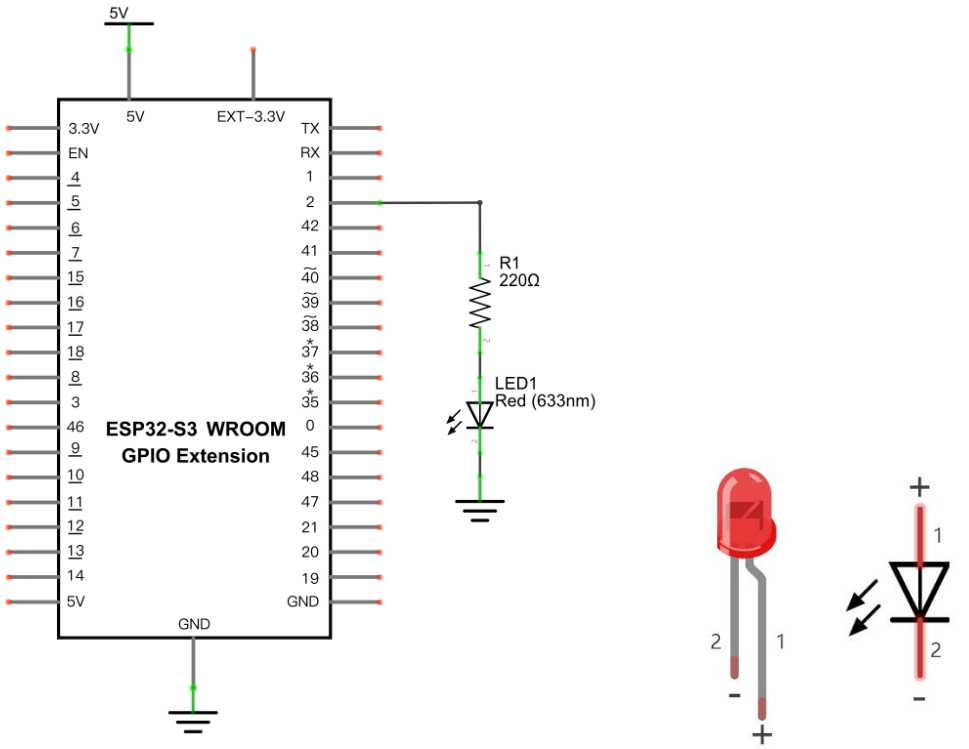
Component List

<table>
<tr><td>ESP32-S3 WROOM x1
  </td><td colspan="3">GPIO Extension Board x1
  </td></tr>
<tr><td>Micro USB Wire x1
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper M/M x2
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
</table>

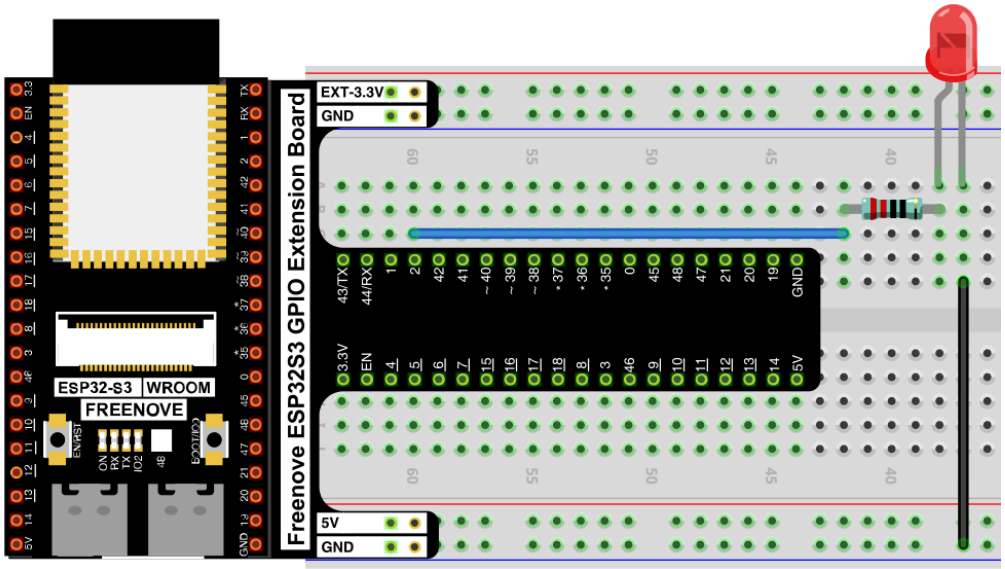
Circuit

Connect Freenove ESP32-S3 to the computer using a USB cable.

Schematic diagram



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

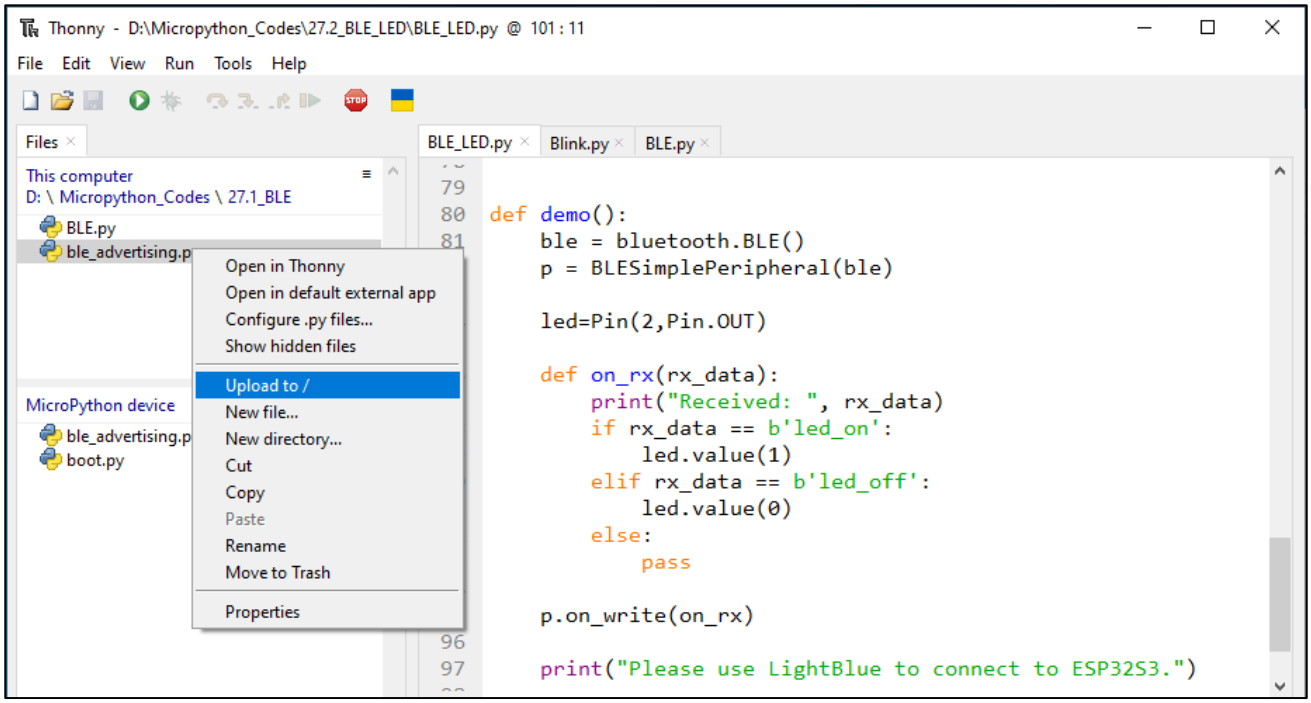


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

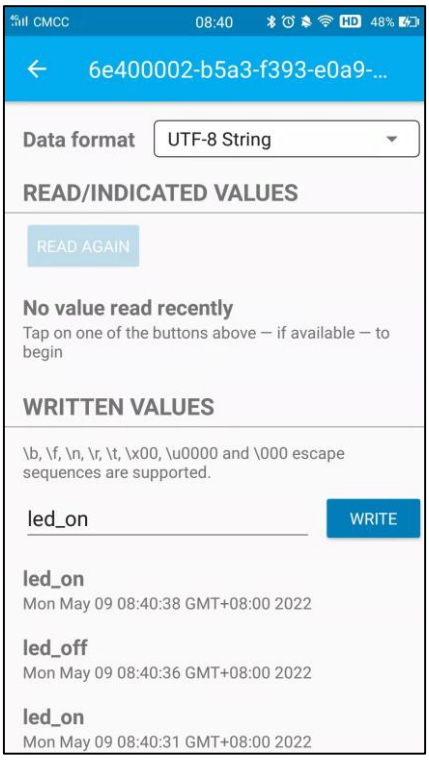
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "19.2_ BLE_LED". Select "ble_advertising.py", right click your mouse to select "Upload to /", wait for "ble_advertising.py" to be uploaded to ESP32-S3 and then double click "BLE_LED.py".

19.2_BLE_LED



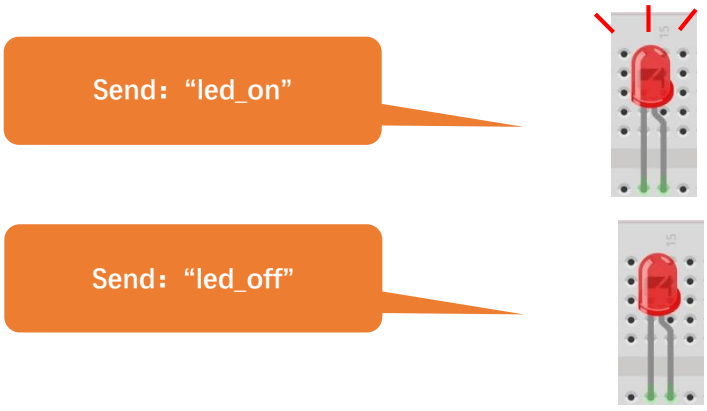
Compile and upload code to ESP32S3. The operation of the APP is the same as 19.1, you only need to change the sending content to "led_on" and "led_off" to operate LEDs on the ESP32S3.

Data sent from mobile APP:



You can check the message sent by Bluetooth in “Shell”.

The phenomenon of LED



Attention: If the sending content isn't "led_on' or "led_off", then the state of LED will not change. If the LED is on, when receiving irrelevant content, it keeps on; Correspondingly, if the LED is off, when receiving irrelevant content, it keeps off.

The following is the program code:

```
import bluetooth
import random
import struct
import time
from ble_advertising import advertising_payload
from machine import Pin
from micropython import const

_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_WRITE = const(3)

_FLAG_READ = const(0x0002)
_FLAG_WRITE_NO_RESPONSE = const(0x0004)
_FLAG_WRITE = const(0x0008)
_FLAG_NOTIFY = const(0x0010)

_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
_UART_SERVICE = (
    _UART_UUID,
    (_UART_TX, _UART_RX),
)
class BLESimplePeripheral:
    def __init__(self, ble, name="ESP32S3"):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
self._ble.gatts_register_services((_UART_SERVICE,))
        self._connections = set()
        self._write_callback = None
        self._payload = advertising_payload(name=name, services=[_UART_UUID])
        self._advertise()
    def _irq(self, event, data):
```

```
43          # Track connections so we can send notifications.
44          if event == _IRQ_CENTRAL_CONNECT:
45              conn_handle, _, _ = data
46              print("New connection", conn_handle)
47              print("\nThe BLE connection is successful.")
48              self._connections.add(conn_handle)
49          elif event == _IRQ_CENTRAL_DISCONNECT:
50              conn_handle, _, _ = data
51              print("Disconnected", conn_handle)
52              self._connections.remove(conn_handle)
53              # Start advertising again to allow a new connection.
54              self._advertise()
55          elif event == _IRQ_GATTS_WRITE:
56              conn_handle, value_handle = data
57              value = self._ble.gatts_read(value_handle)
58              if value_handle == self._handle_rx and self._write_callback:
59                  self._write_callback(value)
60      def send(self, data):
61          for conn_handle in self._connections:
62              self._ble.gatts_notify(conn_handle, self._handle_tx, data)
63      def is_connected(self):
64          return len(self._connections) > 0
65      def _advertise(self, interval_us=500000):
66          print("Starting advertising")
67          self._ble.gap_advertise(interval_us, adv_data=self._payload)
68      def on_write(self, callback):
69          self._write_callback = callback
70  def demo():
71      ble = bluetooth.BLE()
72      p = BLESimplePeripheral(ble)
73      led=Pin(2,Pin.OUT)
74      def on_rx(rx_data):
75          print("Received: ", rx_data)
76          if rx_data == b'led_on':
77              led.value(1)
78          elif rx_data == b'led_off':
79              led.value(0)
80          else:
81              pass
82      p.on_write(on_rx)
83      print("Please use LightBlue to connect to ESP32S3.")
84  if __name__ == "__main__":
85      demo()
```

Compare received message with "led_on" and "led_off" and take action accordingly.

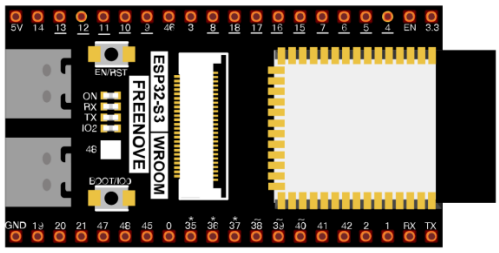
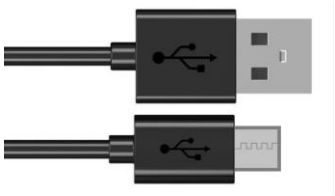
```
76          if rx_data == b'led_on':
77              led.value(1)
78          elif rx_data == b'led_off':
79              led.value(0)
```

Chapter 20 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-S3 WROOM.
ESP32-S3 WROOM 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 20.1 Station mode

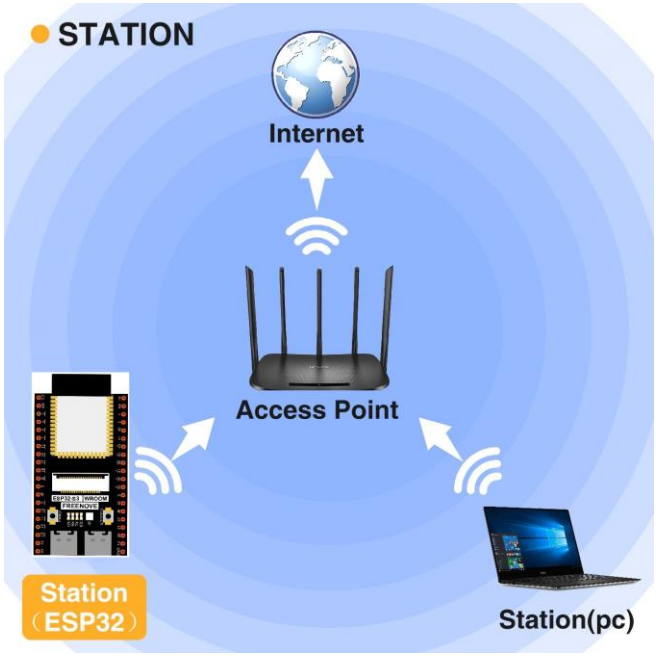
Component List

ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

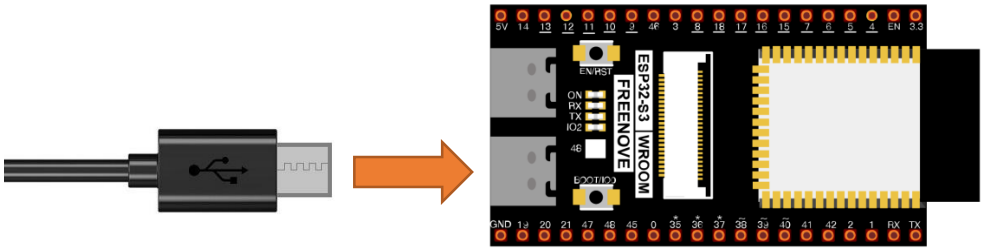
Station mode

When ESP32-S3 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-S3 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

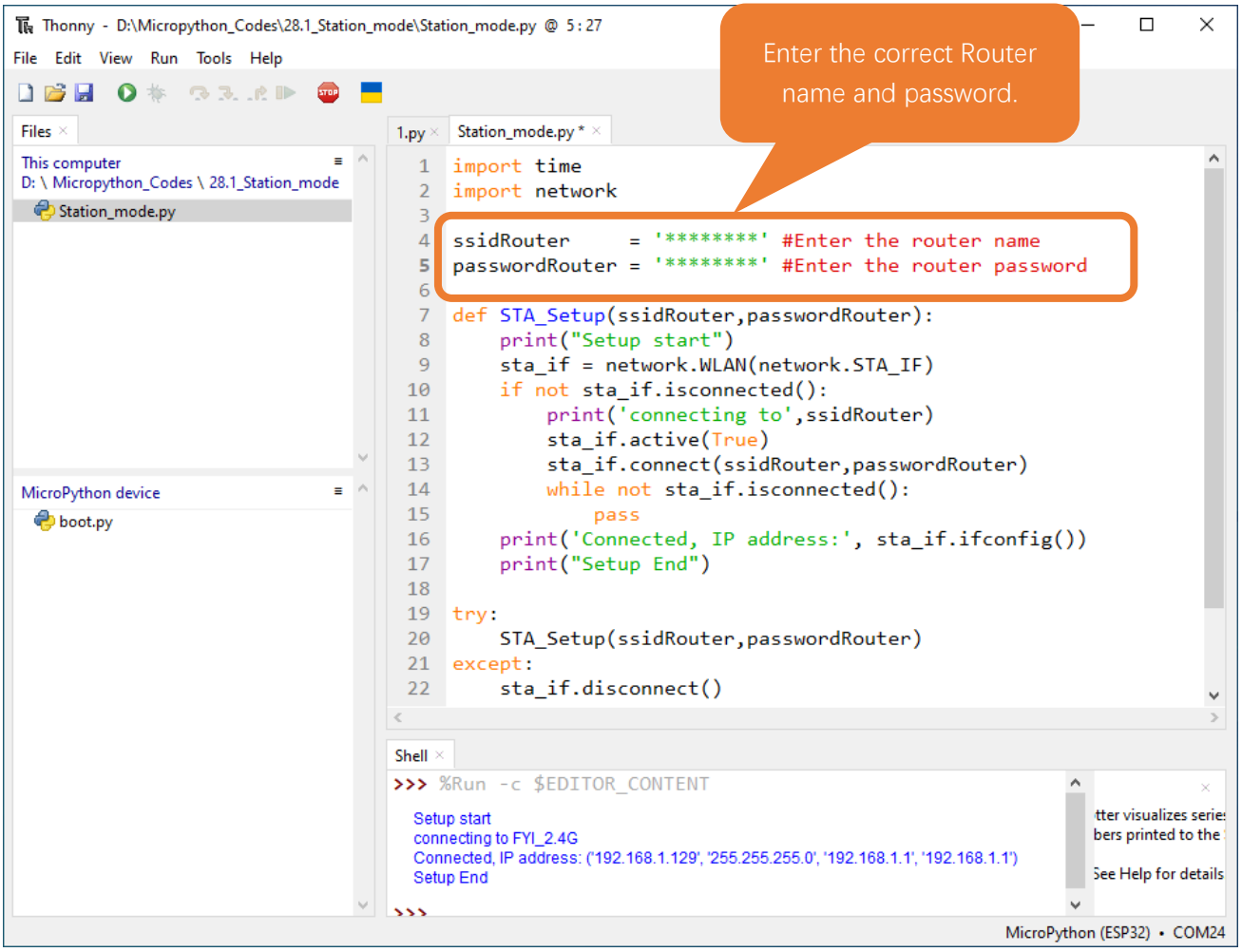


Code

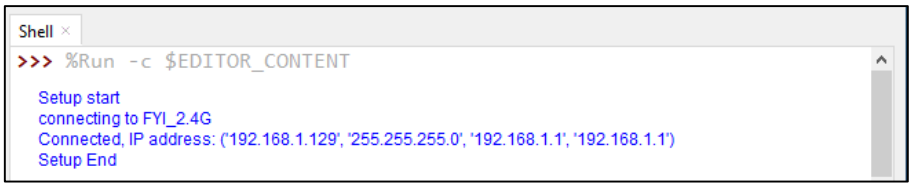
Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "20.1_Station_mode" and double click "Station_mode.py".

20.1_Station_mode



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 ESP32S3, wait for ESP32-S3 to connect to your router and print the IP address assigned by the router to ESP32-S3 in "Shell".



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-S3 in Station mode.

```
9    sta_if = network.WLAN(network.STA_IF)
```

Activate ESP32-S3'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-S3 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-S3 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-S3 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: WiFiname

password: WiFipassword

disconnect(): Disconnect from the currently connected wireless network.

Project 20.2 AP mode

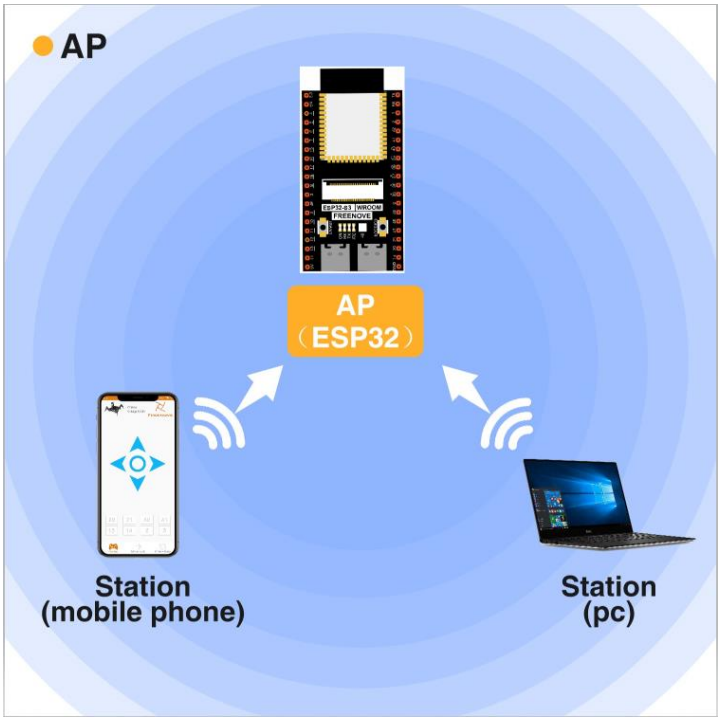
Component List & Circuit

Component List & Circuit are the same as in Project 20.1.

Component knowledge

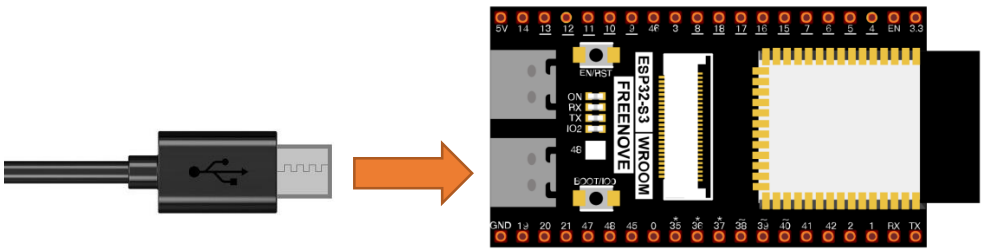
AP mode

When ESP32-S3 selects AP mode, it creates a hotspot network that is separate from the Internet and waits for other WiFi devices to connect. As shown in the figure below, ESP32-S3 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP32S3, it must be connected to the hotspot of ESP32S3. Only after a connection is established with ESP32-S3 can they communicate.



Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

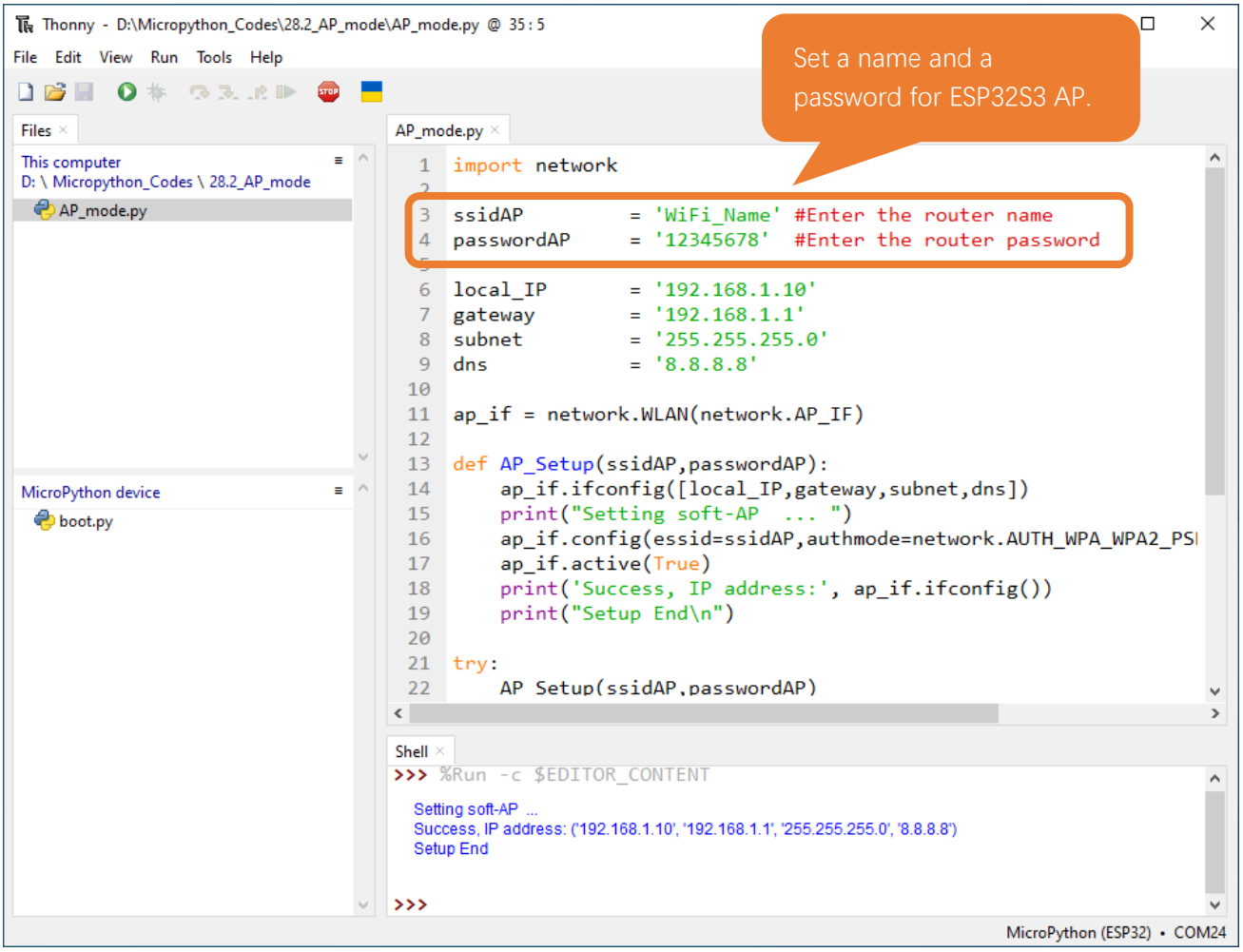


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

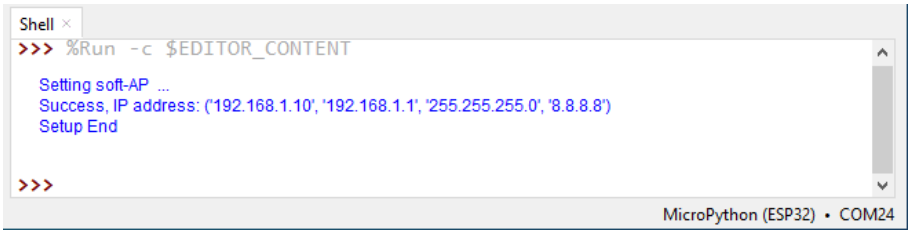
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "20.2_AP_mode". and double click "AP_mode.py".

20.2_AP_mode

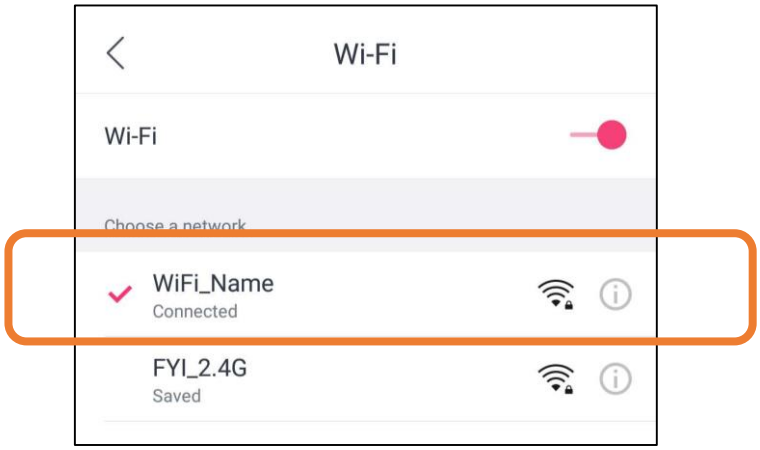


Before the Code runs, you can make any changes to the AP name and password for ESP32-S3 in the box as shown in the illustration above. Of course, you can leave it alone by default.

Click "Run current script", open the AP function of ESP32-S3 and print the access point information.



Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32S3, 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-S3 in AP mode.

```
11 ap_if = network.WLAN(network.AP_IF)
```

Configure IP address, gateway and subnet mask for ESP32S3.

```
14 ap_if.ifconfig([local_IP,gateway,subnet,dns])
```

Turn on an AP in ESP32S3, 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: WiFiname

password: WiFipassword

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: WiFichannel

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: IPaddress

subnet_mask: subnet mask

gateway: gateway

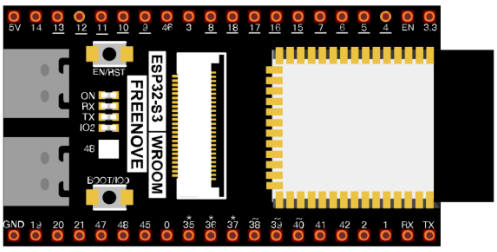
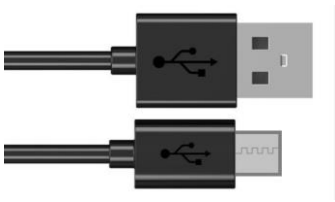
DNS_server: DNSserver

disconnect(): Disconnect from the currently connected wireless network

status(): Return the current status of the wireless connection

Project 20.3 AP+Station mode

Component List

ESP32-S3 WROOM x1 	USB cable x1 

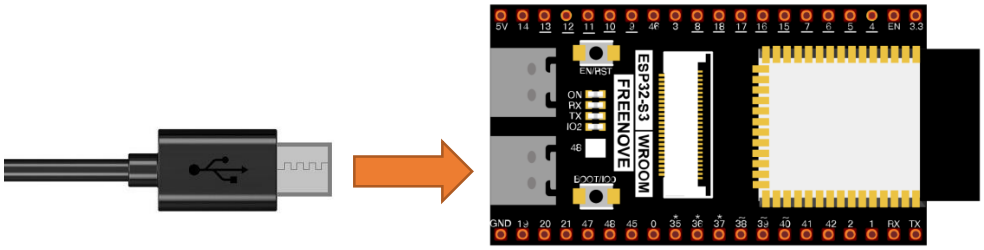
Component knowledge

AP+Station mode

In addition to AP mode and station mode, ESP32-S3 can also use AP mode and station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32S3'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 ESP32S3.

Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

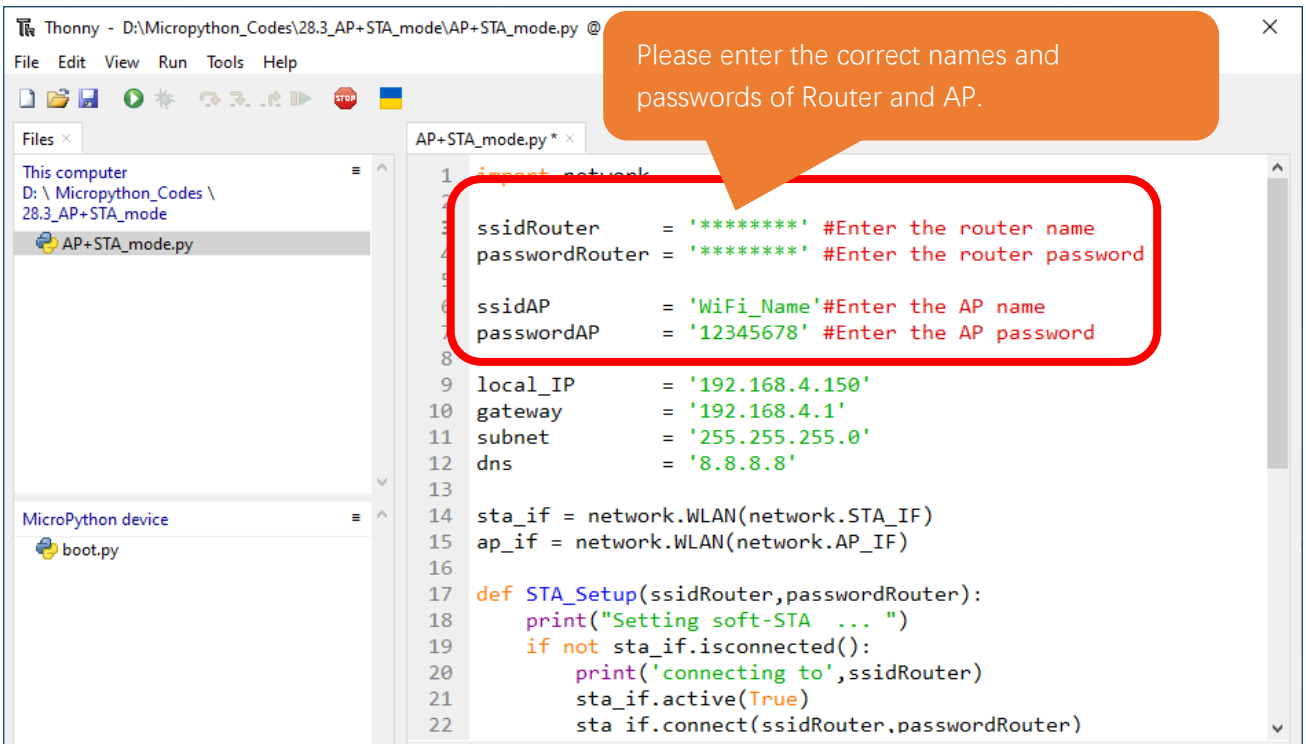


Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "20.3_AP+STA_mode"and double click "AP+STA_mode.py".

20.3_AP+STA_mode



It is analogous to Project 20.1 and Project 20.2. 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:

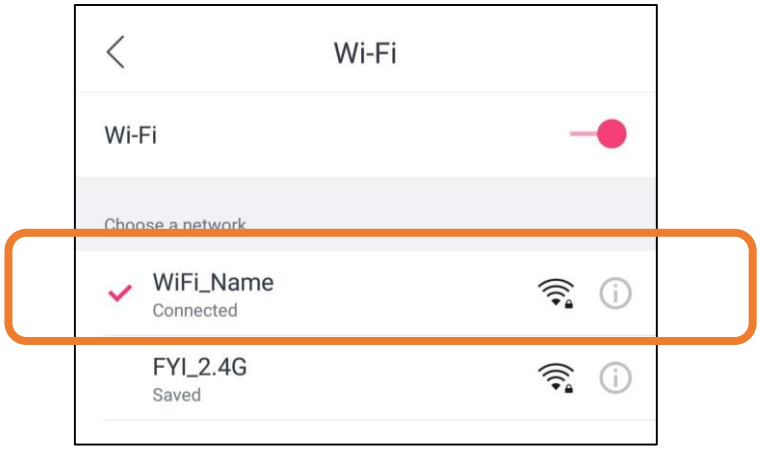
```
Shell
>>> %Run -c $EDITOR_CONTENT

  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 ...
  connecting to FYI_2.4G
  Connected, IP address: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
  Setup End

>>>
                                                    MicroPython (ESP32) • COM24
```

Turn on the WiFi scanning function of your phone, and you can see the ssidAP on ESP32-S3.



The following is the program code:

```
import network

ssidRouter     = '********' #Enter the router name
passwordRouter = '********' #Enter the router password

ssidAP         = 'WiFi_Name'#Enter the AP name
passwordAP     = '12345678' #Enter the AP password

local_IP       = '192.168.4.150'
gateway        = '192.168.4.1'
subnet         = '255.255.255.0'
dns            = '8.8.8.8'

sta_if = network.WLAN(network.STA_IF)
ap_if = network.WLAN(network.AP_IF)

def STA_Setup(ssidRouter,passwordRouter):
    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.idsconnect()
```

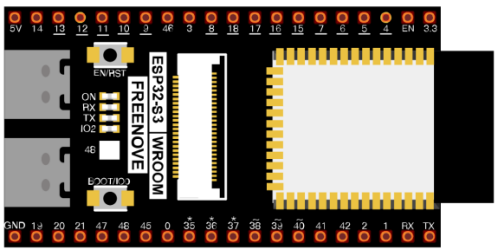
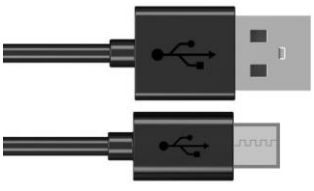
Chapter 21 TCP/IP

In this chapter, we will introduce how ESP32-S3 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 21.1 As Client

In this section, ESP32-S3 is used as Client to connect Server on the same LAN and communicate with it.

Component List

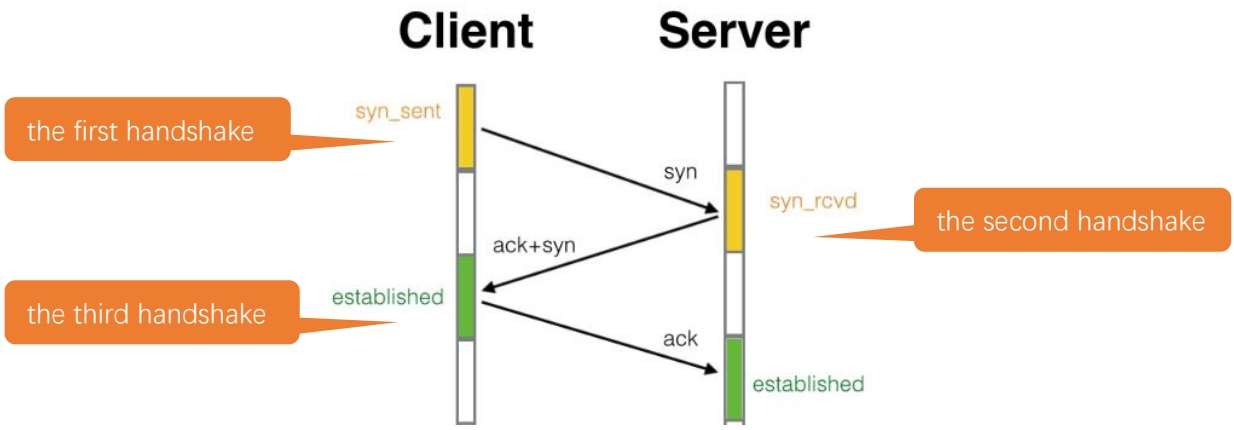
ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

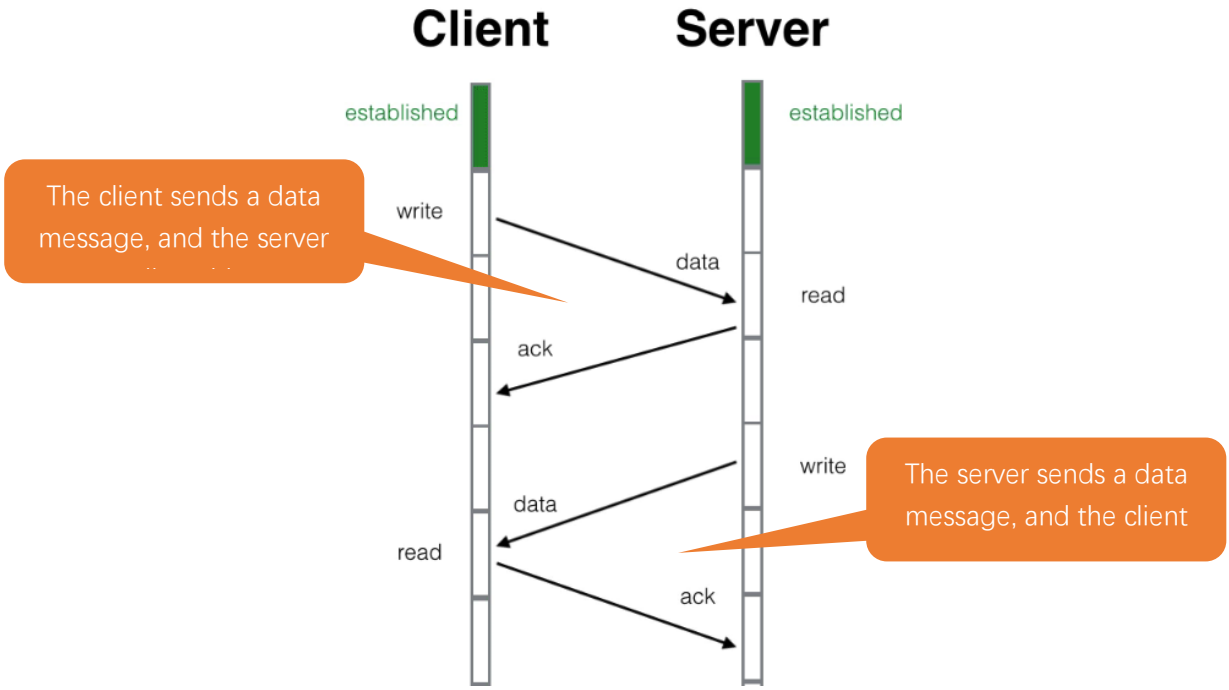
TCP connection

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

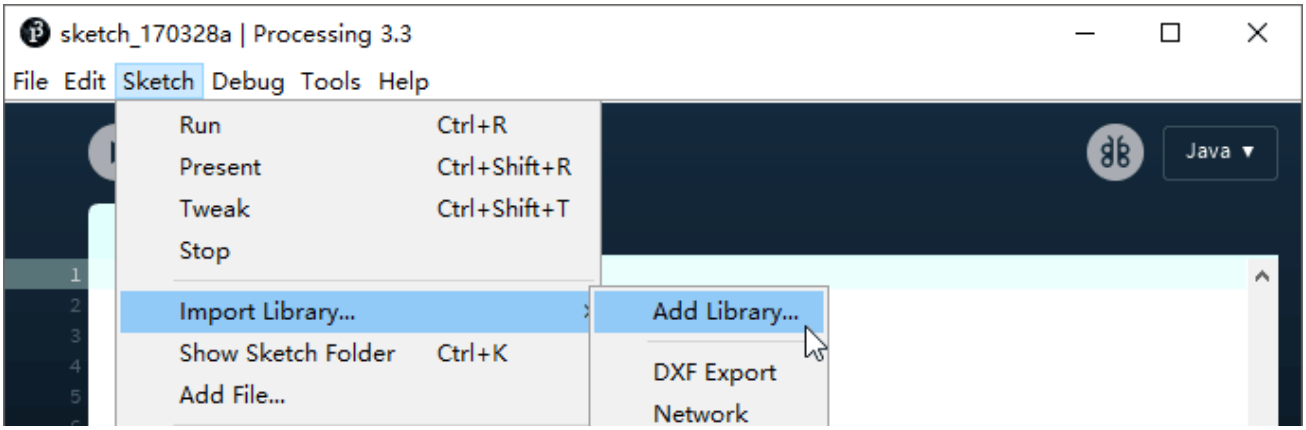
In this tutorial, we use Processing to build a simple TCP/IP communication platform.

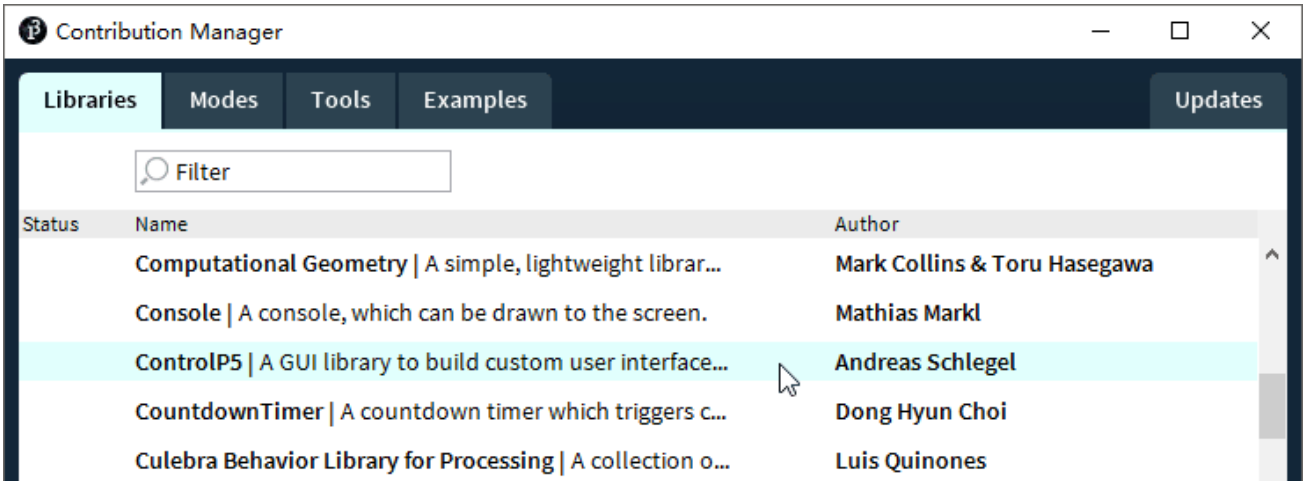
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

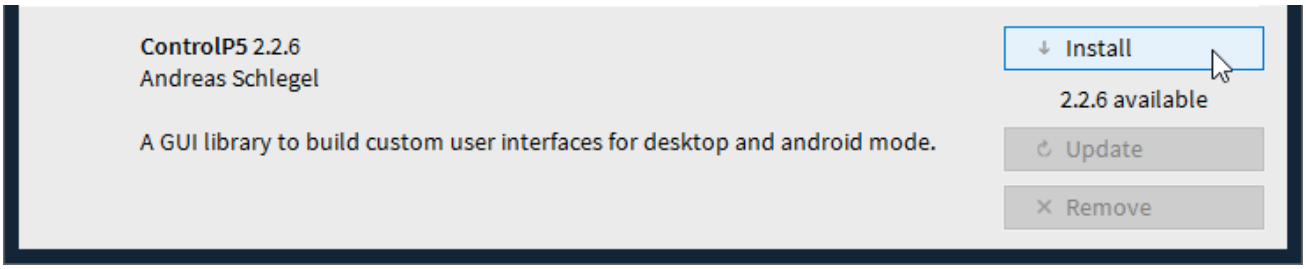
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

Use Server mode for communication

Install ControlP5.

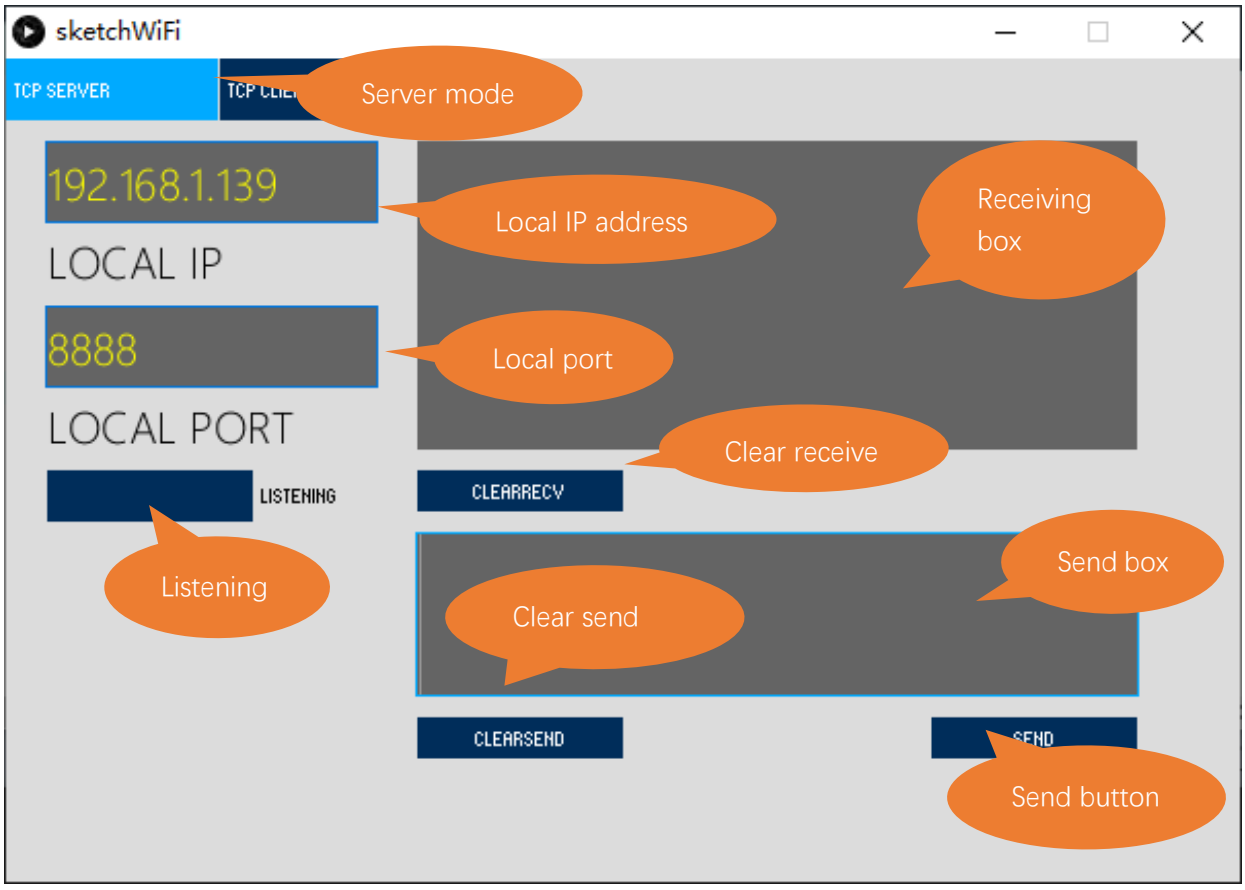






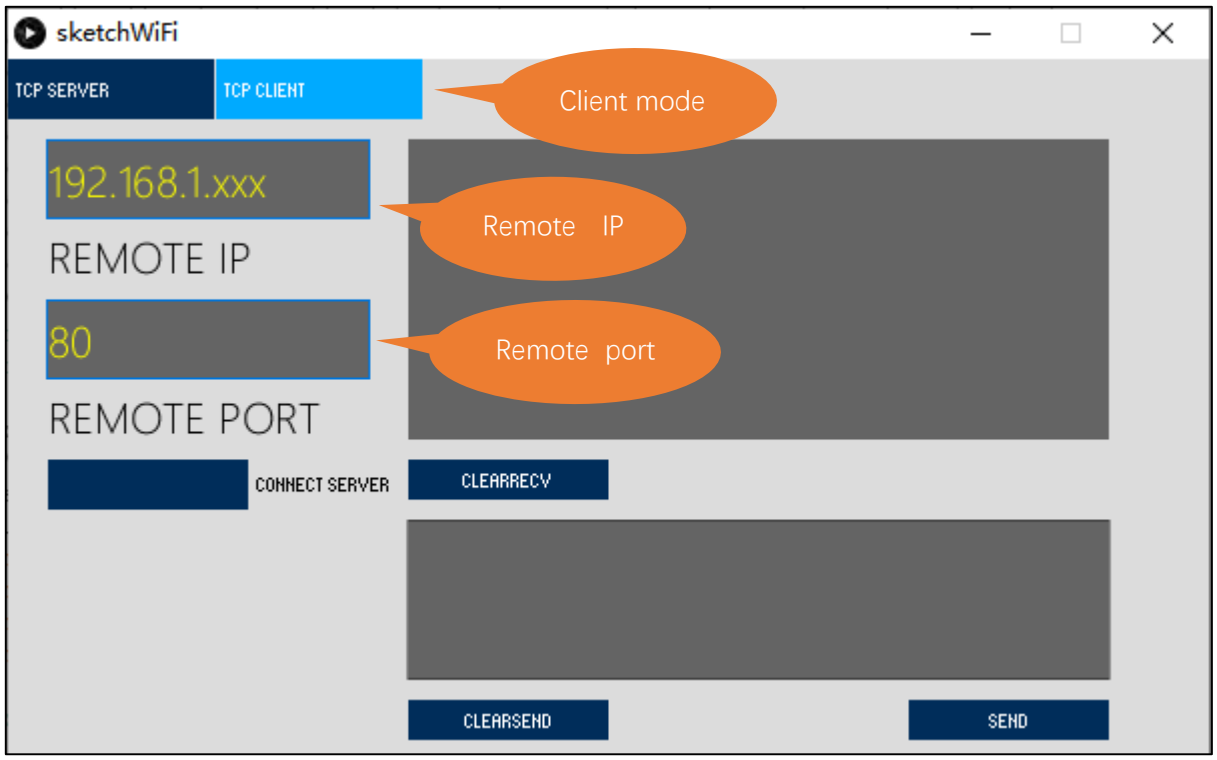
Open the "**Freenove_Super_Starter_Kit_for_ESP32_S3\Sketches\Sketches\Sketch_20.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

The new pop-up interface is as follows. If ESP32-S3 is used as client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, ESP32-S3 Sketch needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP32-S3 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, 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.

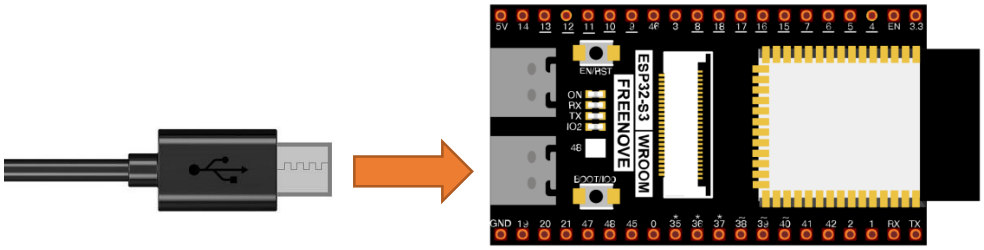
clear receive: clear out the content in the receiving text box

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.

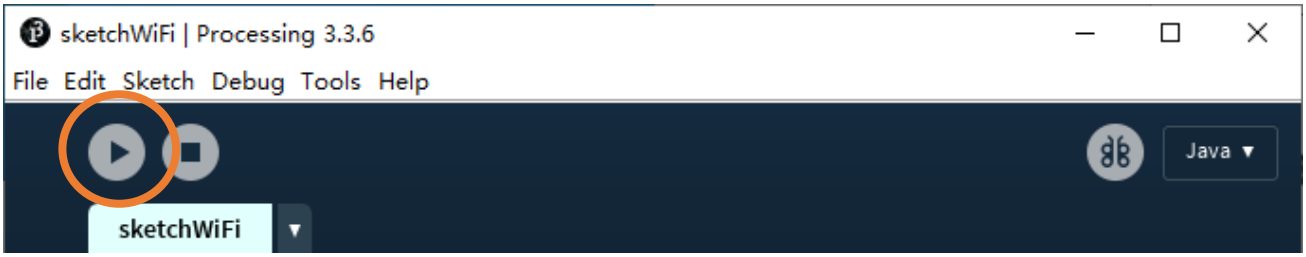
Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

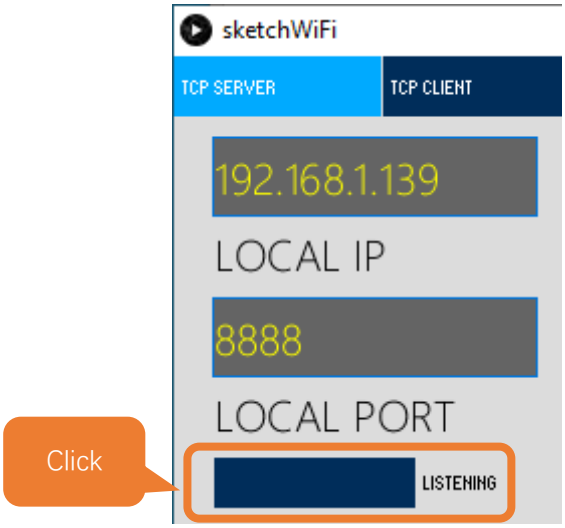


Code

Before running the Code, please open "sketchWiFi.pde." first, and click "Run".



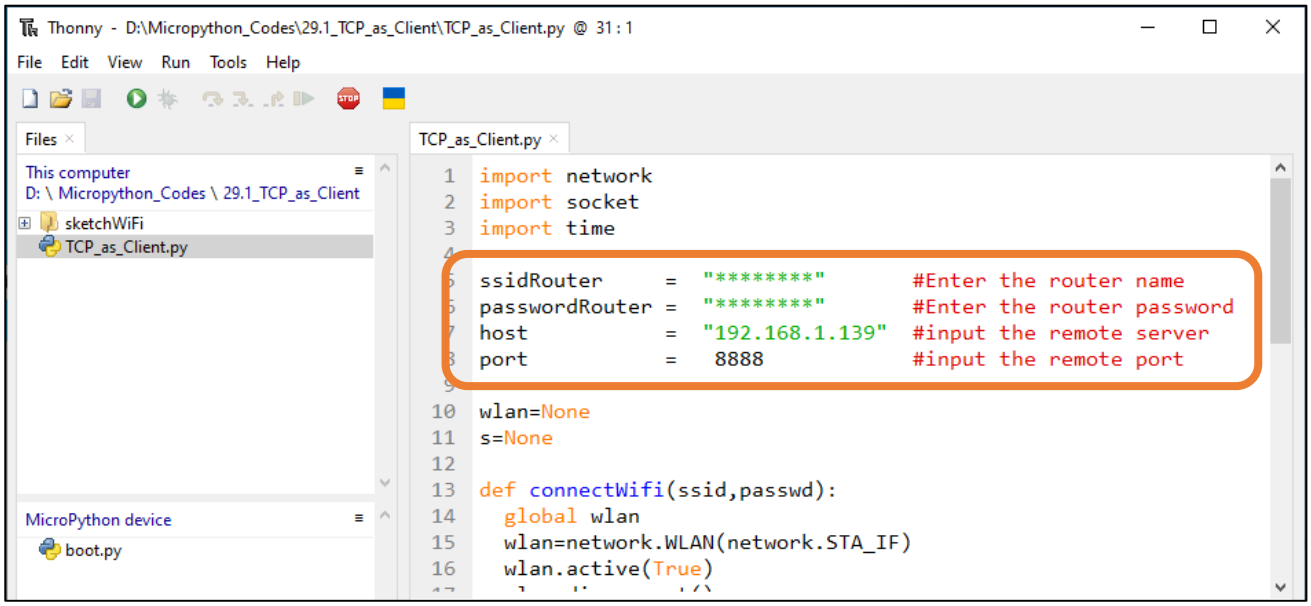
The newly pop up window will use the computer's IP address by default and open a data monitor port. Click "Listening"。



Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "21.1_TCP_as_Client" and double click "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:
21.1_TCP_as_Client

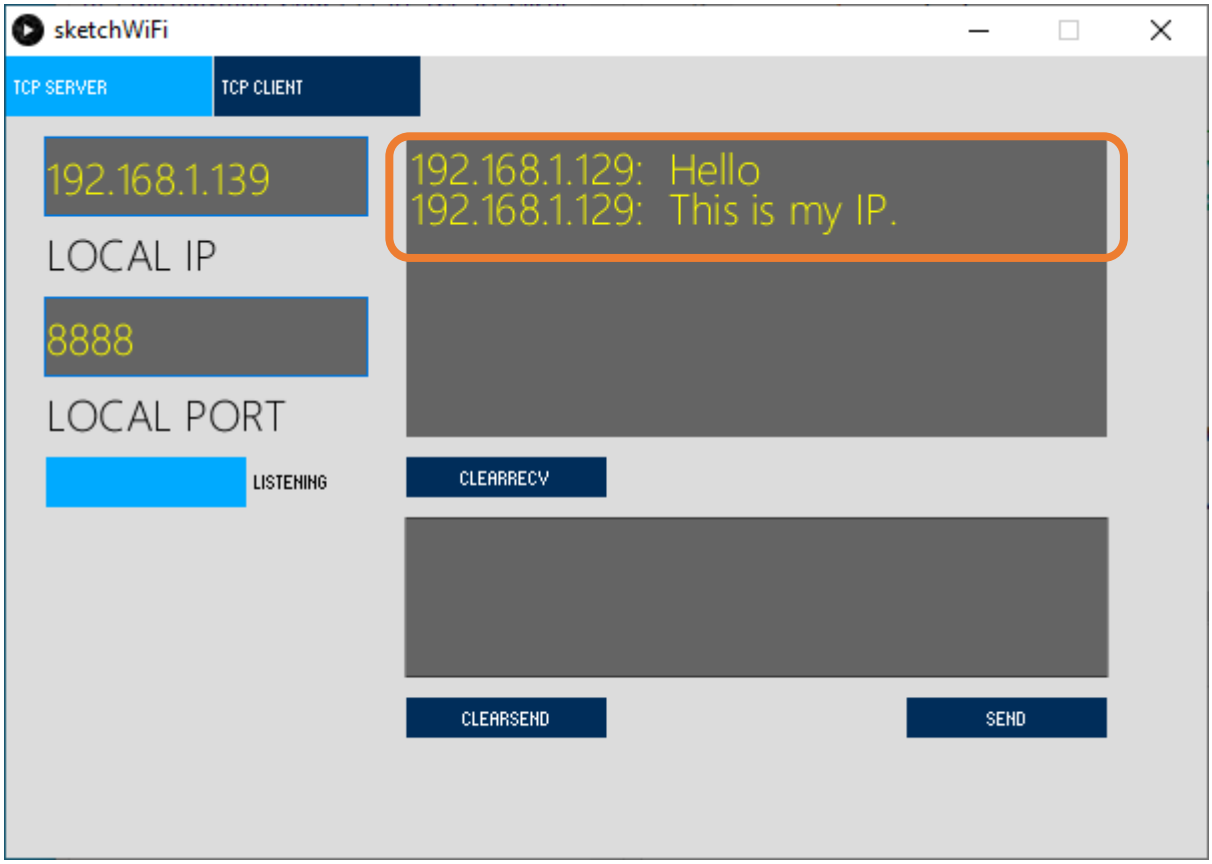


Click "Run current script" and in "Shell", you can see ESP32-S3 automatically connects to sketchWiFi.

If you don't click "Listening" for sketchWiFi, ESP32-S3 will fail to connect and will print information as follows:

```
Shell
  Close socket
>>> %Run -c $EDITOR_CONTENT
  TCP close, please reset!
>>>
```

ESP32-S3 connects with TCP SERVER, and TCP SERVER receives messages from ESP32S3, as shown in the figure below. You can enter any content in TCP SERVER, click SEND, and ESP32-S3 will receive this message



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
```

```
def connectWifi(ssid,passwd):
  global wlan
  wlan= network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.disconnect()
  wlan.connect(ssid,passwd)
  while(wlan.ifconfig()[0]=='0.0.0.0'):
    time.sleep(1)
  return True
try:
  connectWifi(ssidRouter,passwordRouter)
  s = socket.socket()
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  s.connect((host,port))
  print("TCP Connected to:", host, ":", port)
  s.send('Hello')
  s.send('This is my IP.')
  while True:
    data = s.recv(1024)
    if(len(data) == 0):
      print("Close socket")
      s.close()
      break
    print(data)
    ret=s.send(data)
except:
  print("TCP close, please reset!")
  if (s):
    s.close()
  wlan.disconnect()
  wlan.active(False)
```

Import network、socket、time modules.

```
import network
import socket
import time
```

Enter the actual router name, password, remote server IP address, and port number.

```
ssidRouter     =  "********"    #Enter the router name
passwordRouter =  "********"    #Enter the router password
host           =  "********"    #input the remote server
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.

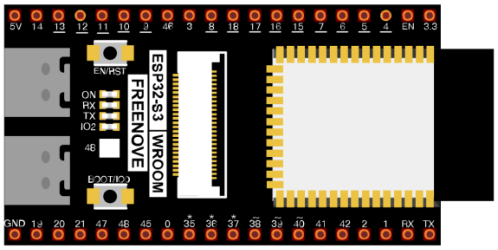
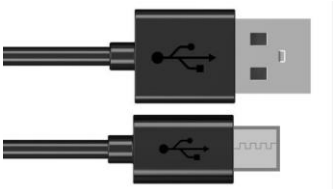
close(): Close socket.

To learn more please visit: http://docs.micropython.org/en/latest/

Project 21.2 As Server

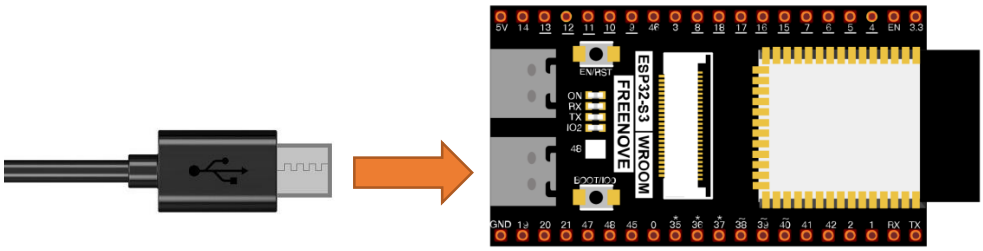
In this section, ESP32-S3 is used as a server to wait for the connection and communication of client on the same LAN.

Component List

ESP32-S3 WROOM x1 	USB cable x1 

Circuit

Connect Freenove ESP32-S3 to the computer using a USB cable.



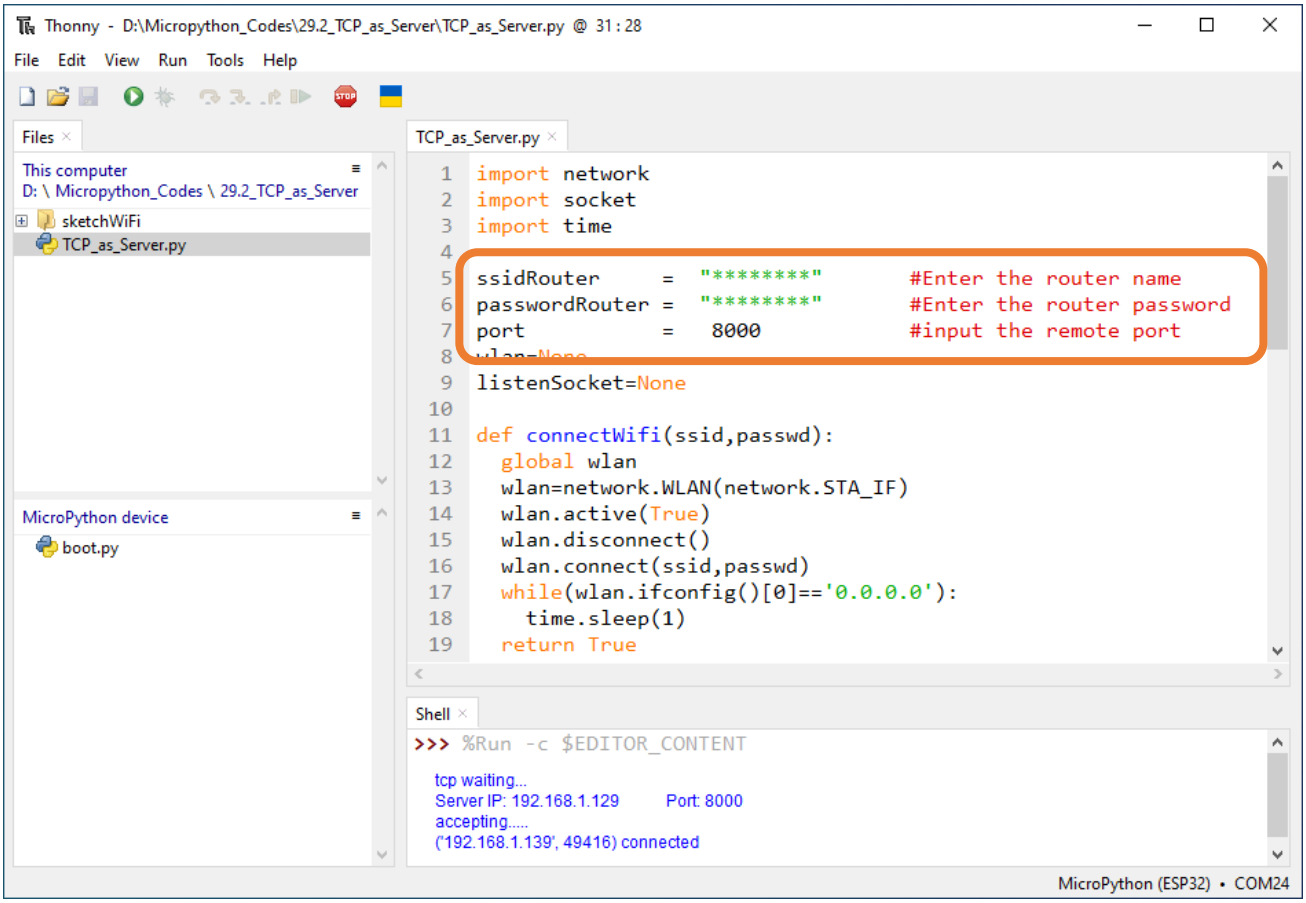
Code

Move the program folder "**Freenove_Super_Starter_Kit_for_ESP32_S3/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "21.2_TCP_as_Server" and double click "TCP_as_Server.py".

Before clicking "Run current script", please modify the name and password of your router shown in the box below.

21.2_TCP_as_Server

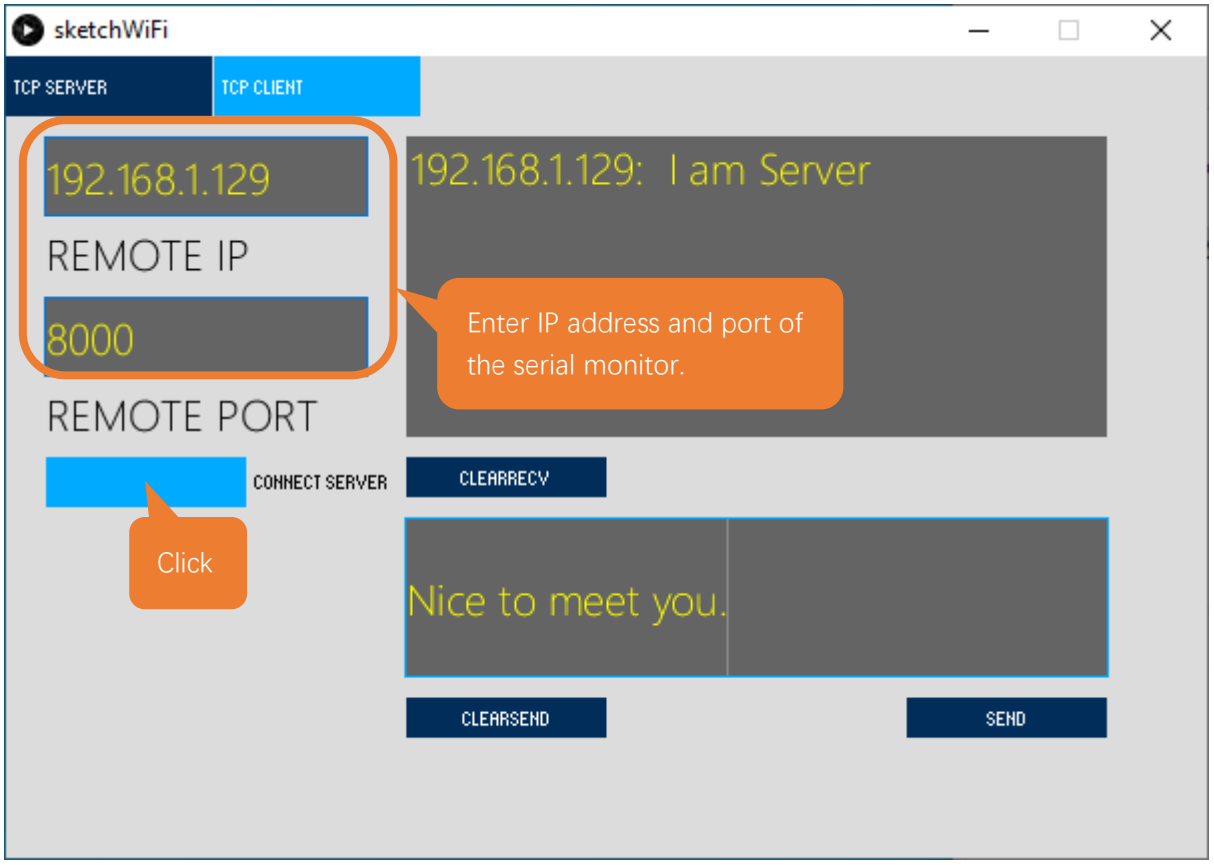


After making sure that the router's name and password are correct, click "Run current script" and in "Shell", you can see a server opened by the ESP32-S3 waiting to connecting to other network devices.

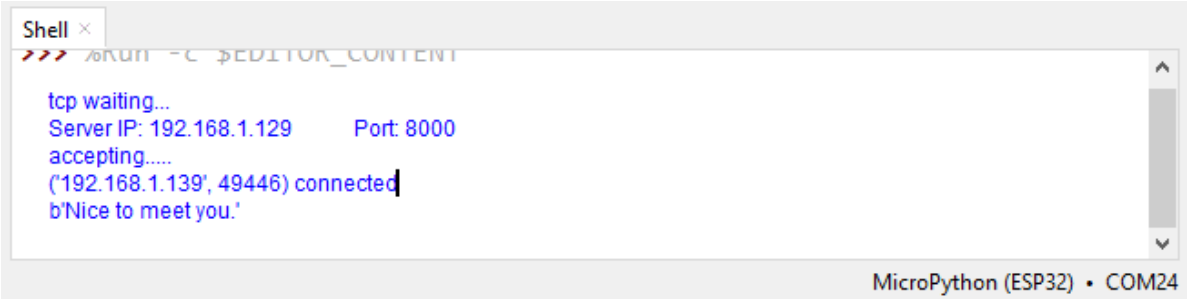
Processing:

Open the "**Freenove_Super_Starter_Kit_for_ESP32_S3/Codes/MicroPython_Codes/21.2_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-S3 to communicate.



You can enter any information in the "Send Box" of sketchWiFi. Click "Send" and ESP32-S3 will print the received messages to "Shell" and send them back to sketchWiFi.



The following is the program code:

```
import network
import socket
import time

ssidRouter     =  "********"       #Enter the router name
passwordRouter =  "********"       #Enter the router password
port           =   8000            #input the remote port
wlan           =  None
listenSocket   =  None

def connectWifi(ssid,passwd):
  global wlan
  wlan=network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.disconnect()
  wlan.connect(ssid,passwd)
  while(wlan.ifconfig()[0]=='0.0.0.0'):
    time.sleep(1)
  return True

try:
  connectWifi(ssidRouter,passwordRouter)
  ip=wlan.ifconfig()[0]
  listenSocket = socket.socket()
  listenSocket.bind((ip,port))
  listenSocket.listen(1)
  listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  print('tcp waiting...')
  while True:
    print("Server IP:",ip,"\tPort:",port)
    print("accepting.....")
    conn,addr = listenSocket.accept()
    print(addr,"connected")
    break
  conn.send('I am Server')
  while True:
    data = conn.recv(1024)
    if(len(data) == 0):
      print("close socket")
      listenSocket.close()
      wlan.disconnect()
      wlan.active(False)
      break
```

```
    else:
      print(data)
      ret = conn.send(data)
except:
  print("Close TCP-Server, please reset.")
  if(listenSocket):
    listenSocket.close()
  wlan.disconnect()
  wlan.active(False)
```

Call function connectWifi() to connect to router and obtain the dynamic IP that it assigns to ESP32-S3.

```
  connectWifi(ssidRouter,passwordRouter)
  ip=wlan.ifconfig()[0]
```

Open the socket server, bind the server to the dynamic IP, and open a data monitoring port.

```
  listenSocket = socket.socket()
  listenSocket.bind((ip,port))
  listenSocket.listen(1)
  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.

```
  while True:
    print("Server IP:",ip,"\tPort:",port)
    print("accepting.....")
    conn,addr = listenSocket.accept()
    print(addr,"connected")
    break
```

Each time receiving data, print them in "Shell" and send them back to the client.

```
  while True:
    data = conn.recv(1024)
    if(len(data) == 0):
      print("close socket")
      listenSocket.close()
      wlan.disconnect()
      wlan.active(False)
      break
    else:
      print(data)
      ret = conn.send(data)
```

If the client is disconnected, close the server and disconnect WiFi.

```
except:
  print("Close TCP-Server, please reset.")
  if(listenSocket):
    listenSocket.close()
  wlan.disconnect()
  wlan.active(False)
```

What's next?

Thanks for your reading. This tutorial is all over here. If you find any mistakes, omissions or you have other ideas and questions about contents of this tutorial or the kit and etc., please feel free to contact us:

support@freenove.com

We will check and correct it as soon as possible.

If you want learn more about ESP32S3, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_ESP32_S3/archive/master.zip

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and other interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

http://www.freenove.com/

End of the Tutorial

Thank you again for choosing Freenove products.