Important Information

Thank you for choosing Freenove products!

Getting Started

First, please read the **Start Here.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

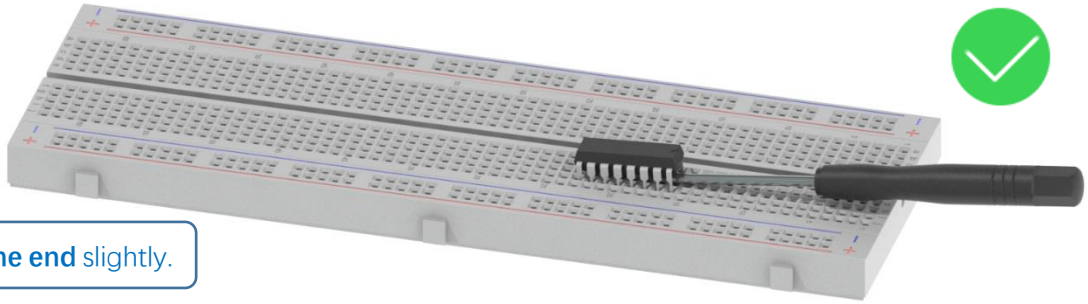
sale@freenove.com

Copyright

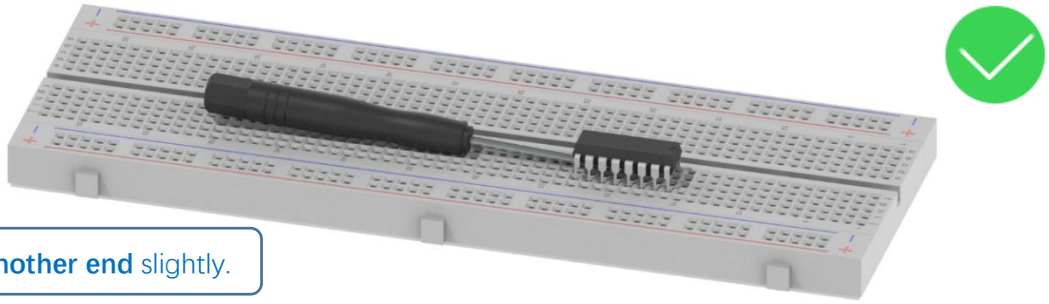
FREENOVE™

Remove the Chips

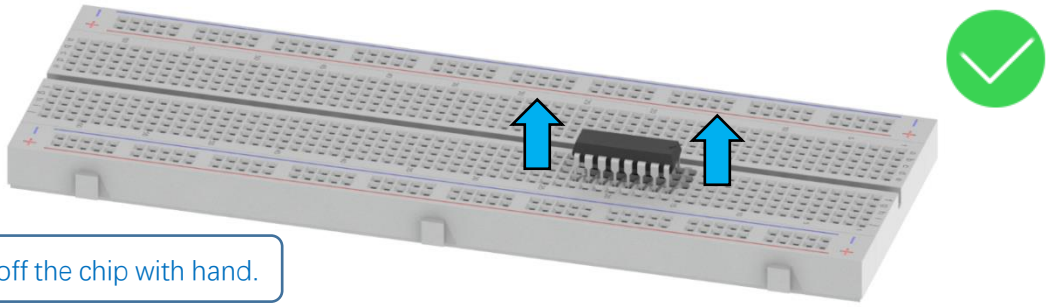
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



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

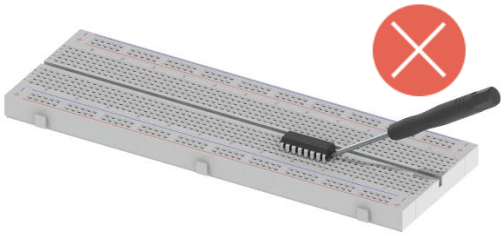


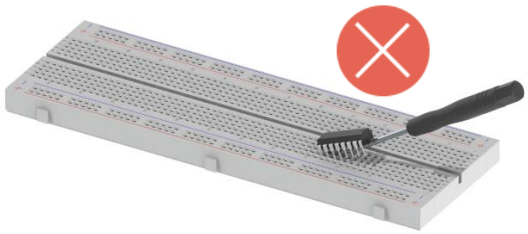
Step 2, lift **another end** slightly.



Step 3, take off the chip with hand.

Avoid lifting one end with big angle directly.





Contents

Important Information....1
Remove the Chips....3
Contents....1
Preface....1
ESP32-WROVER....2
Extension board of the ESP32-WROVER....5
CH340 (Importance)....6
Programming Software....16
Environment Configuration....19
Notes for GPIO....23
Chapter 0 LED....26
Project 0.1 Blink....26
Chapter 1 LED....31
Project 1.1 Blink....31
Chapter 2 Button & LED....37
Project 2.1 Button & LED....37
Project 2.2 MINI table lamp....42
Chapter 3 LED Bar....45
Project 3.1 Flowing Light....45
Chapter 4 Analog & PWM....50
Project 4.1 Breathing LED....50
Project 4.2 Meteor Flowing Light....57
Chapter 5 RGB LED....60
Project 5.1 Random Color Light....60
Project 5.2 Gradient Color Light....66
Chapter 6 Buzzer....68
Project 6.1 Doorbell....68
Project 6.2 Alertor....74
Project 6.3 Alertor (use timer)....77

Chapter 7 Serial Communication 81
Project 7.1 Serial Print 81
Project 7.2 Serial Read and Write 86
Chapter 8 AD/DA Converter 89
Project 8.1 Read the Voltage of Potentiometer 89
Chapter 9 Touch Sensor 96
Project 9.1 Read Touch Sensor 96
Project 9.2 Touch Lamp 101
Chapter 10 Potentiometer & LED 106
Project 10.1 Soft Light 106
Chapter 11 Photoresistor & LED 109
Project 11.1 NightLamp 109
Chapter 12 Thermistor 113
Project 12.1 Thermometer 113
Chapter 13 Bluetooth 118
Project 13.1 Bluetooth Passthrough 118
Project 13.2 Bluetooth Low Energy Data Passthrough 124
Project 13.3 Bluetooth Control LED 136
Chapter 14 Read and Write the Sdcard 142
Project 14.1 SDMMC Test 142
Chapter 15 WiFi Working Modes 153
Project 15.1 Station mode 153
Project 15.2 AP mode 157
Project 14.3 AP+Station mode 162
Chapter 16 TCP/IP 166
Project 16.1 As Client 166
Project 16.2 As Server 178
Chapter 17 Camera Web Server 184
Project 17.1 Camera Web Server 184
Project 17.2 Video Web Server 193
What's next? 199

End of the Tutorial.....199

Preface

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

ESP32 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 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 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**

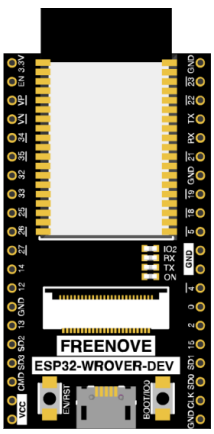
ESP32-WROVER

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

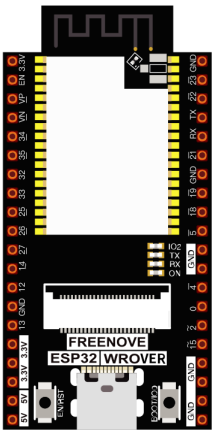
PCB on-board antenna	IPEX antenna
	

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

ESP32-WROVER

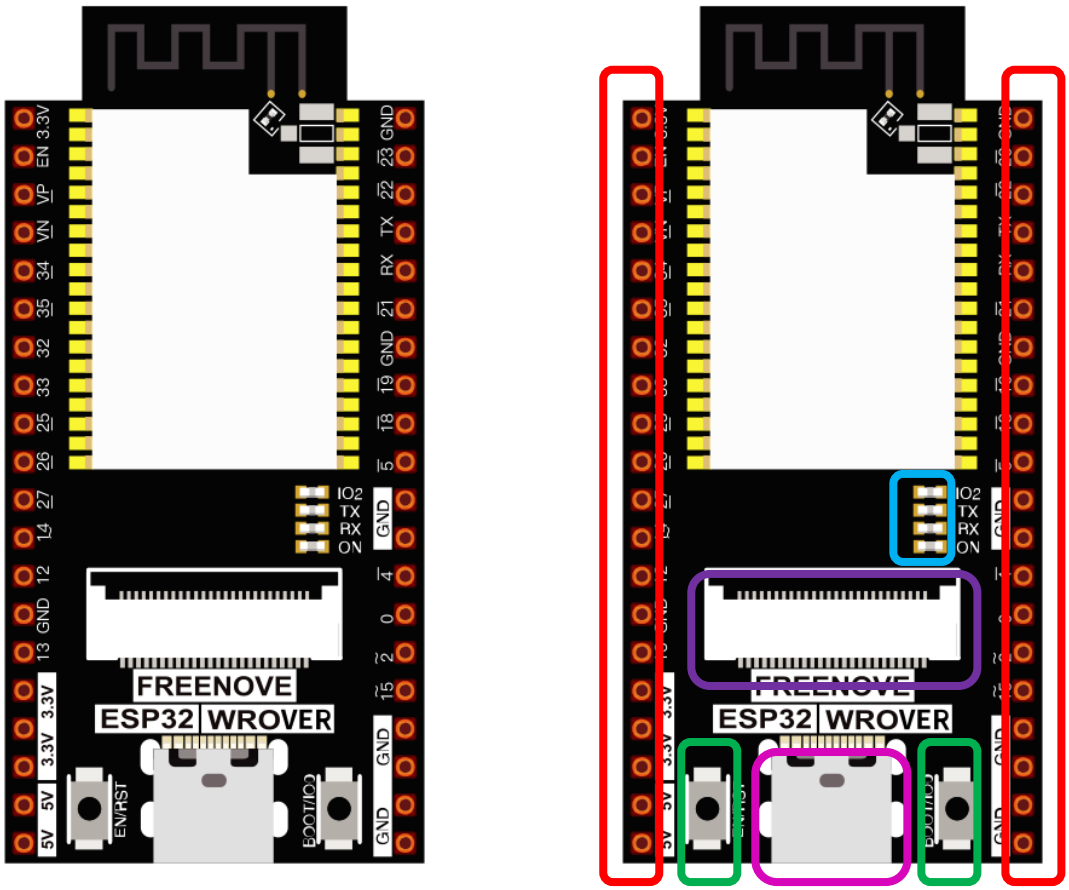


or



The version on the left is no longer mass-produced, and we mainly maintain the version on the right.
Please note that there are many pirated versions of the ESP32 WROVER that look very similar to the version on the left. None of them will carry our logo and Freenove font.
We do not sell pirated ESP32 WROVER, nor do we provide after-sales service for pirated.

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



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

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

Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.
SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12 [1]	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2 [2]	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3 [2]	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD [2]	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK [2]	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0 [2]	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1 [2]	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
NC1	27	-	-
NC2	28	-	-
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

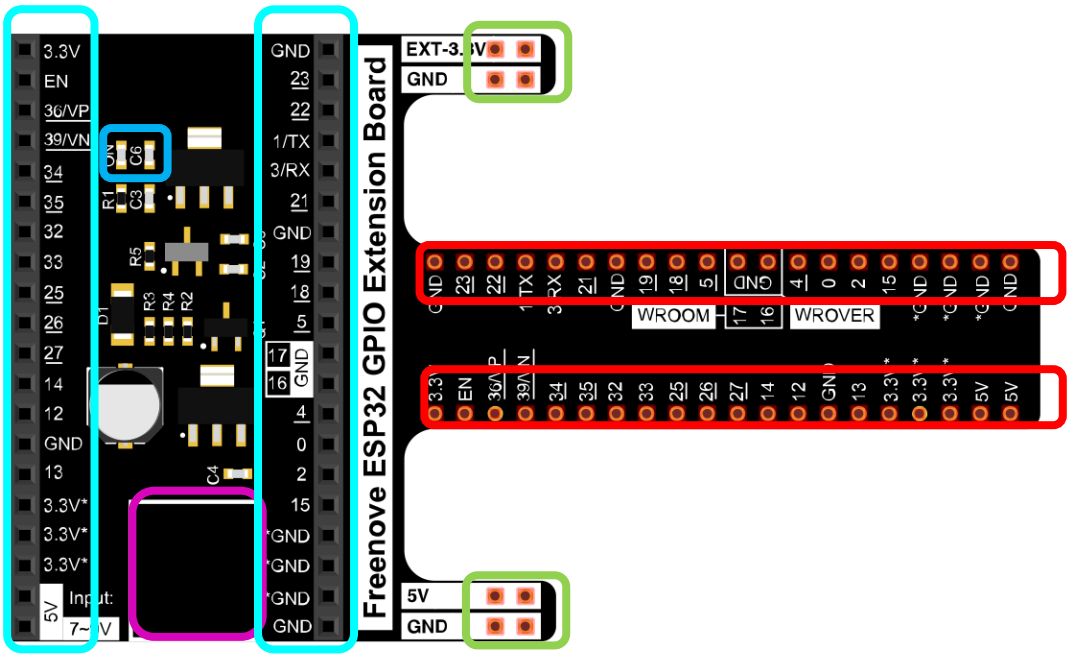
Notice:

1. GPIO12 is internally pulled high in the module and is not recommended for use as a touch pin.
2. Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the SPI flash integrated on the module and are not recommended for other uses.

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

Extension board of the ESP32-WROVER

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



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

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

In ESP32, GPIO is an interface to control peripheral circuit. For beginners, it is necessary to learn the functions of each GPIO. The following is an introduction to the GPIO resources of the ESP32-WROVER development board.
In the following projects, we only use USB cable to power ESP32-WROVER by default.
In the whole tutorial, we don't use T extension to power ESP32-WROVER. So 5V and 3.3V (including EXT 3.3V) on the extension board are provided by ESP32-WROVER.
We can also use DC jack of extension board to power ESP32-WROVER.In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

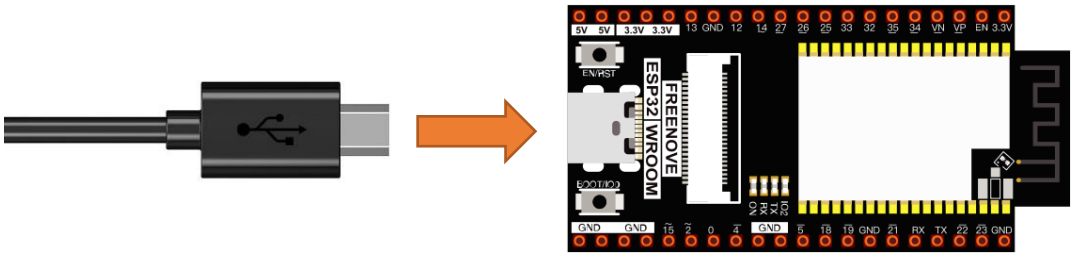
CH340 (Importance)

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

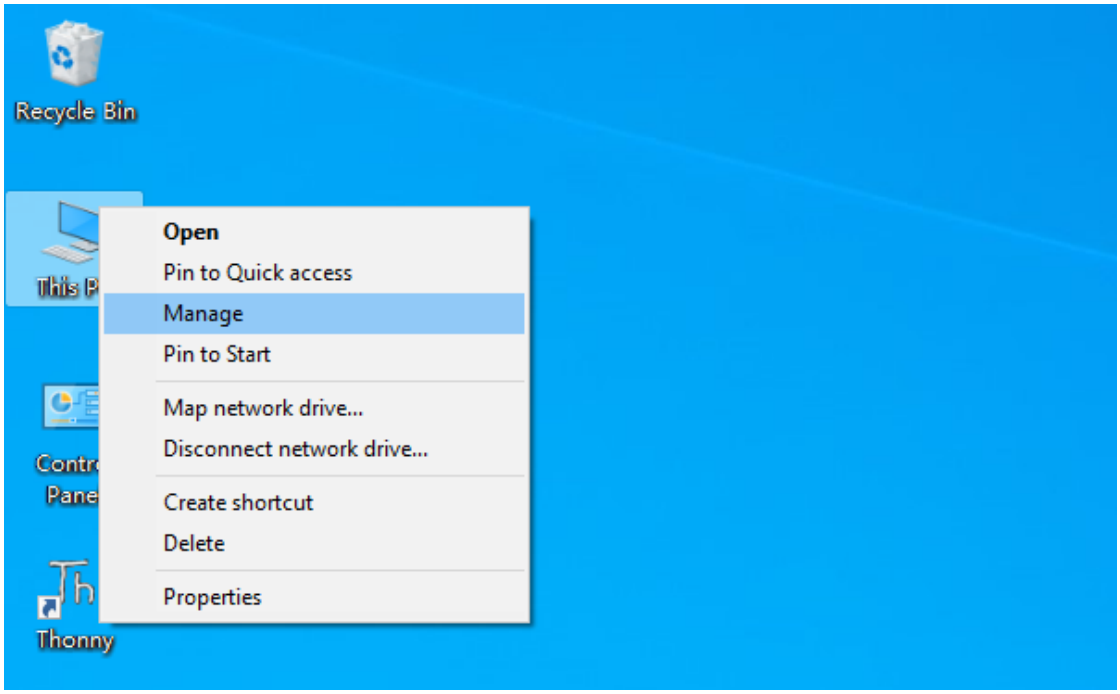
Windows

Check whether CH340 has been installed

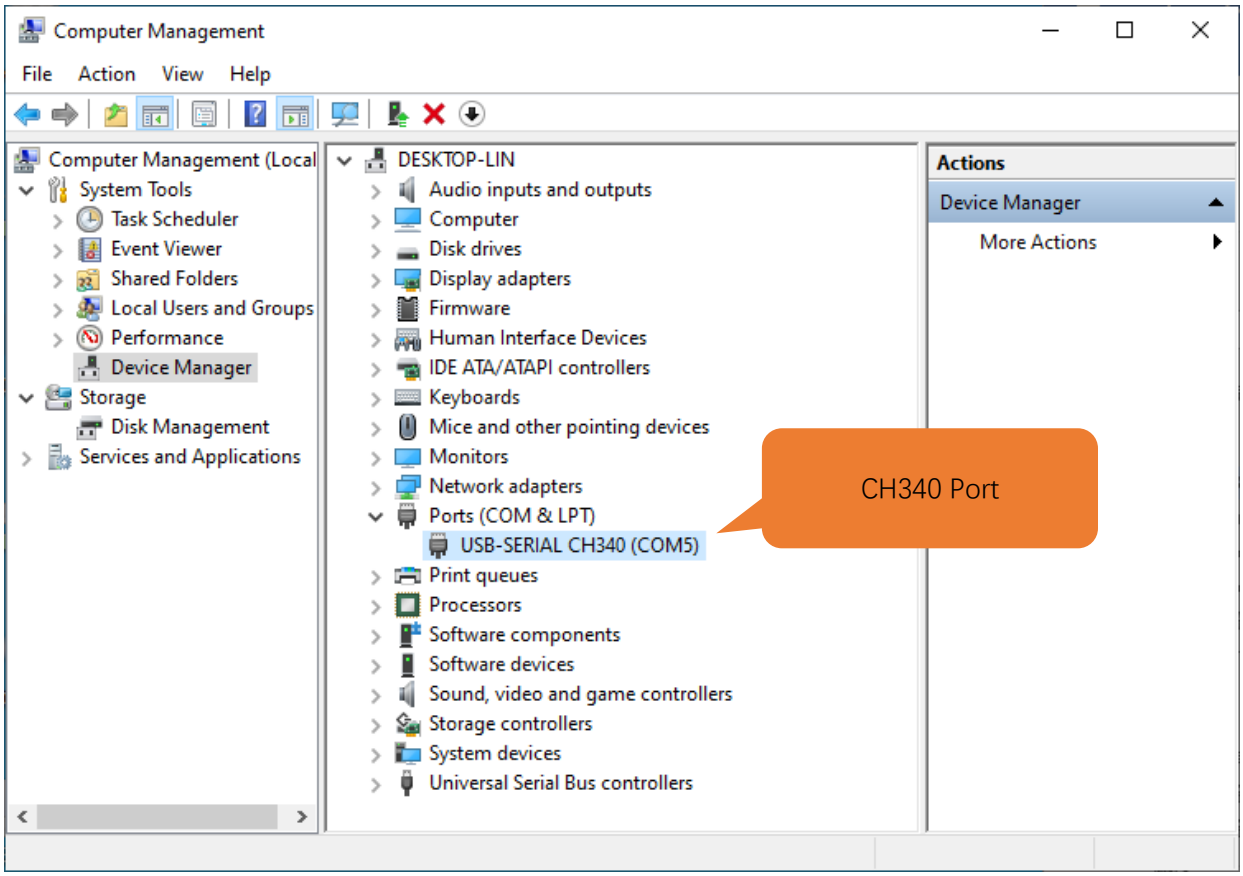
1. Connect your computer and ESP32 with a USB cable.



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

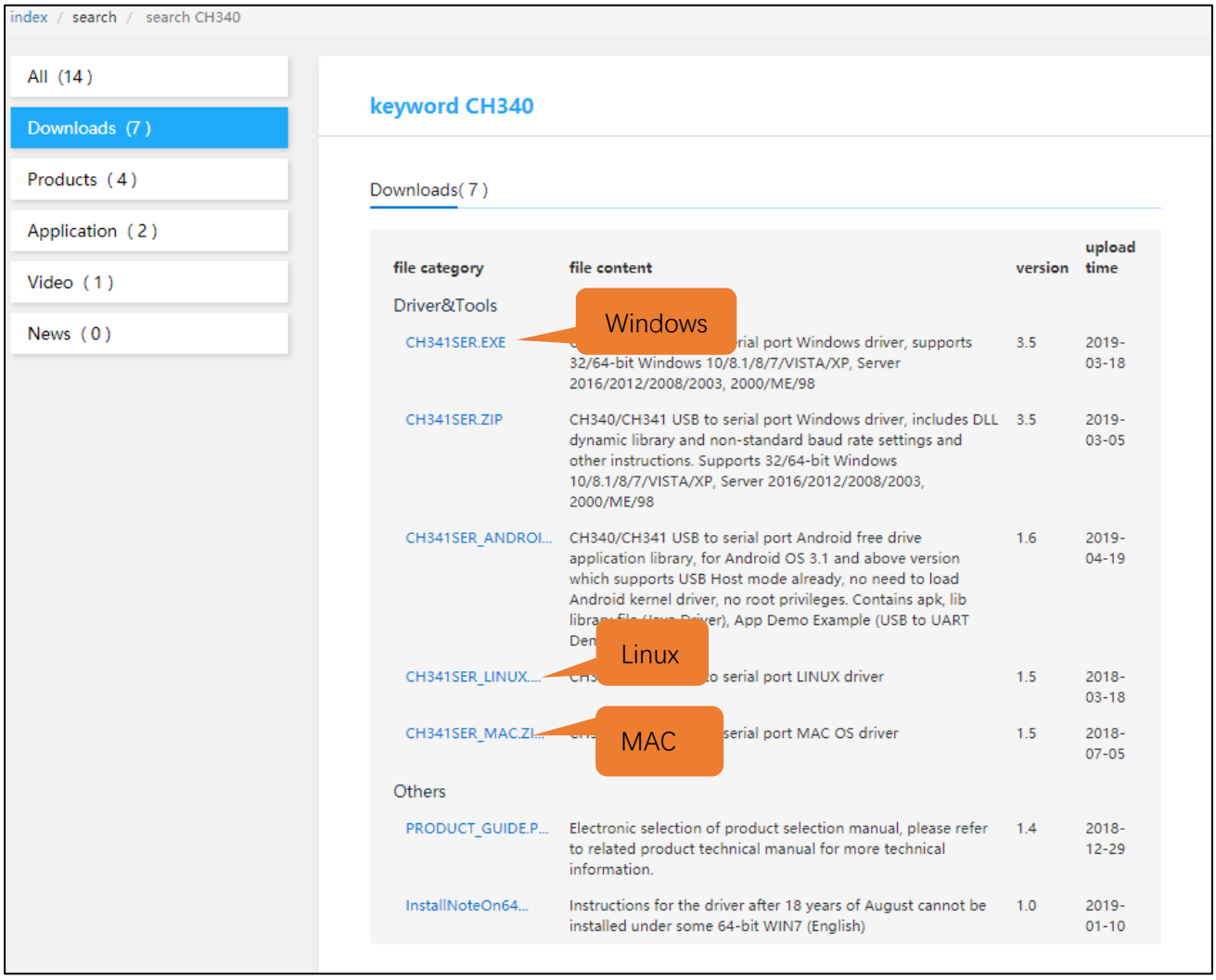


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

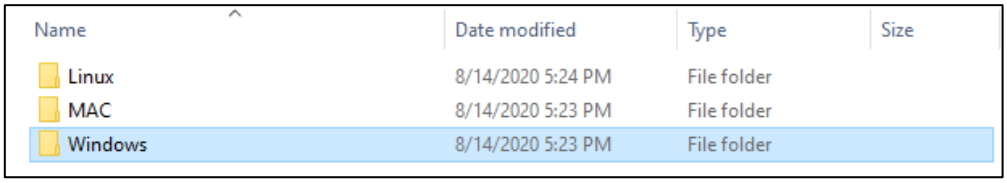


Installing CH340

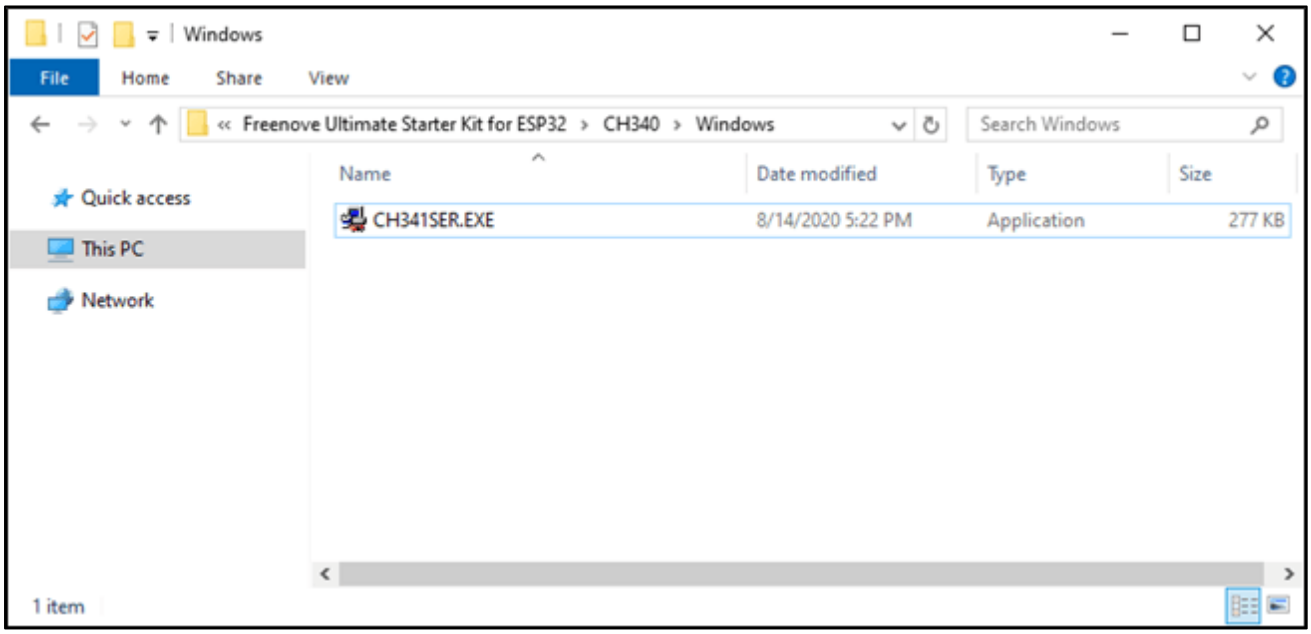
1. First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



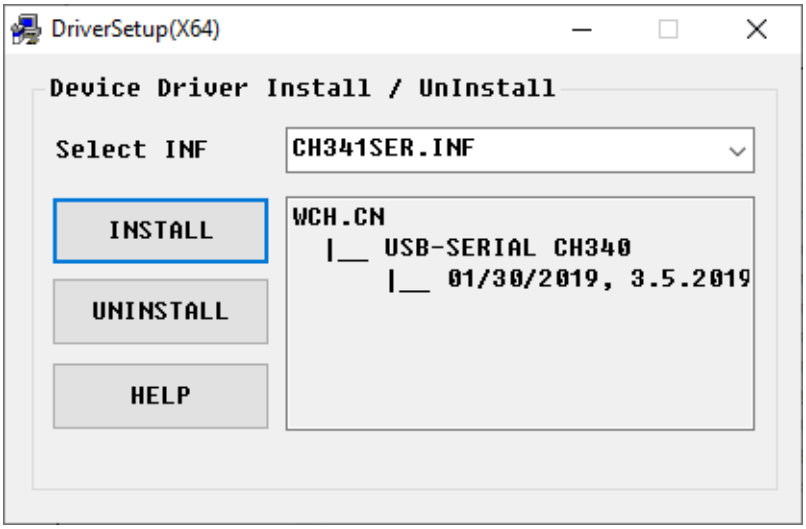
If you would not like to download the installation package, you can open **"Freenove_Basic_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.



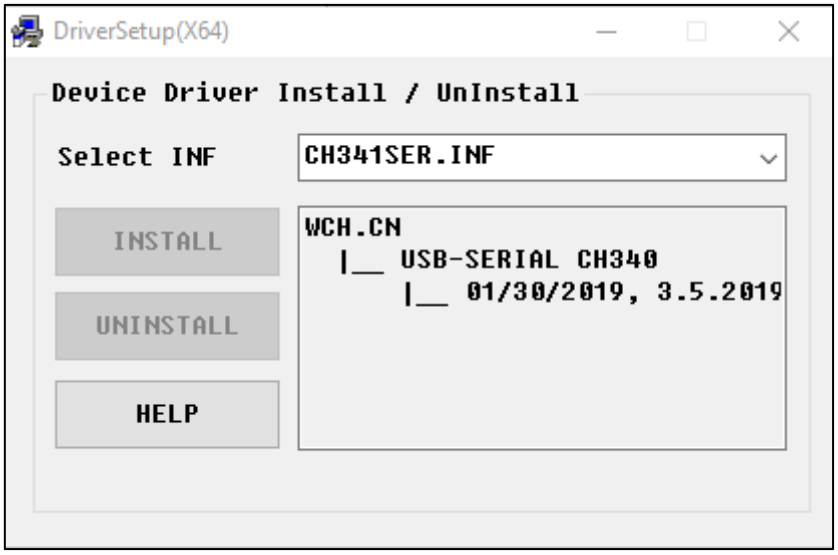
2. Open the folder "**Freenove_Basic_Starter_Kit_for_ESP32/CH340/Windows/**"



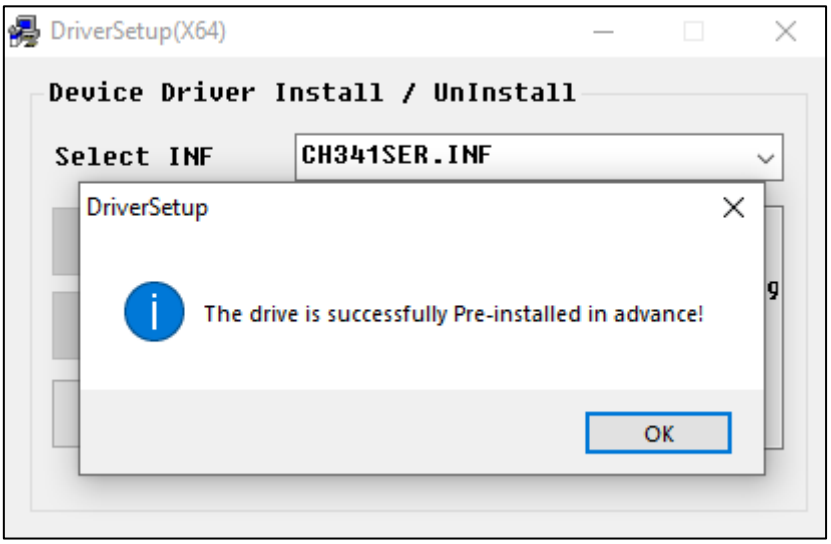
3. Double click "**CH341SER.EXE**".



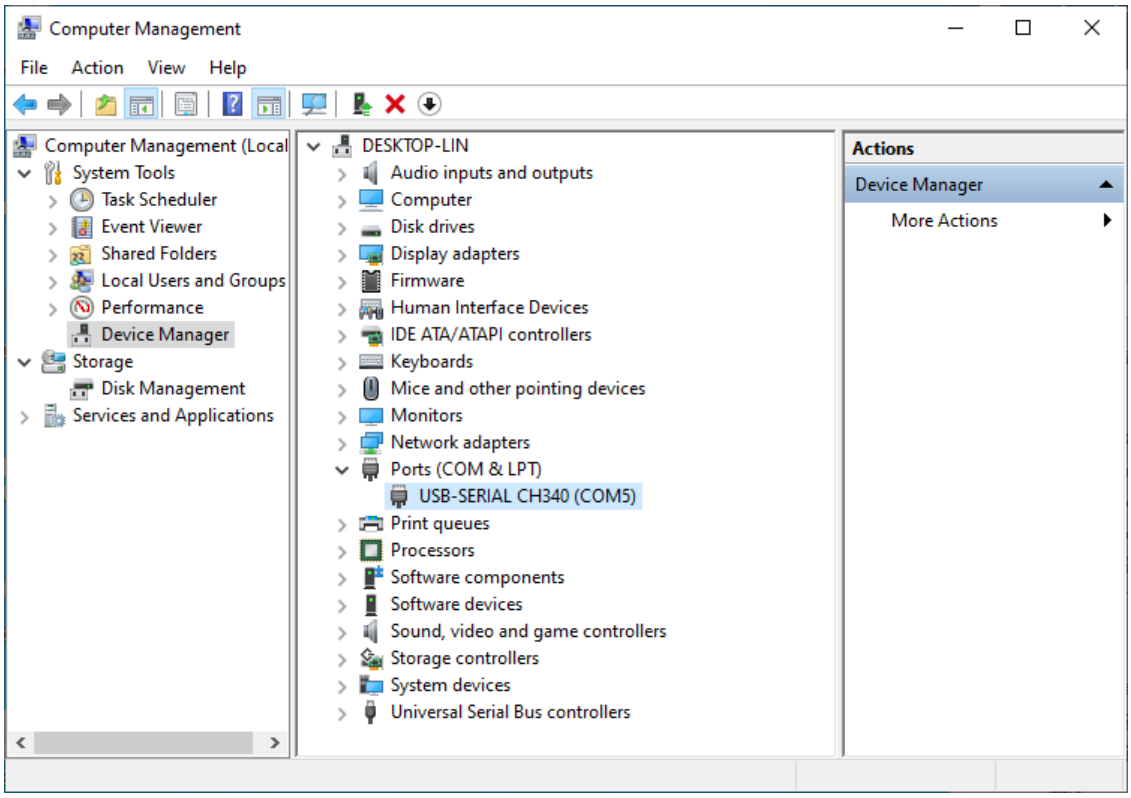
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



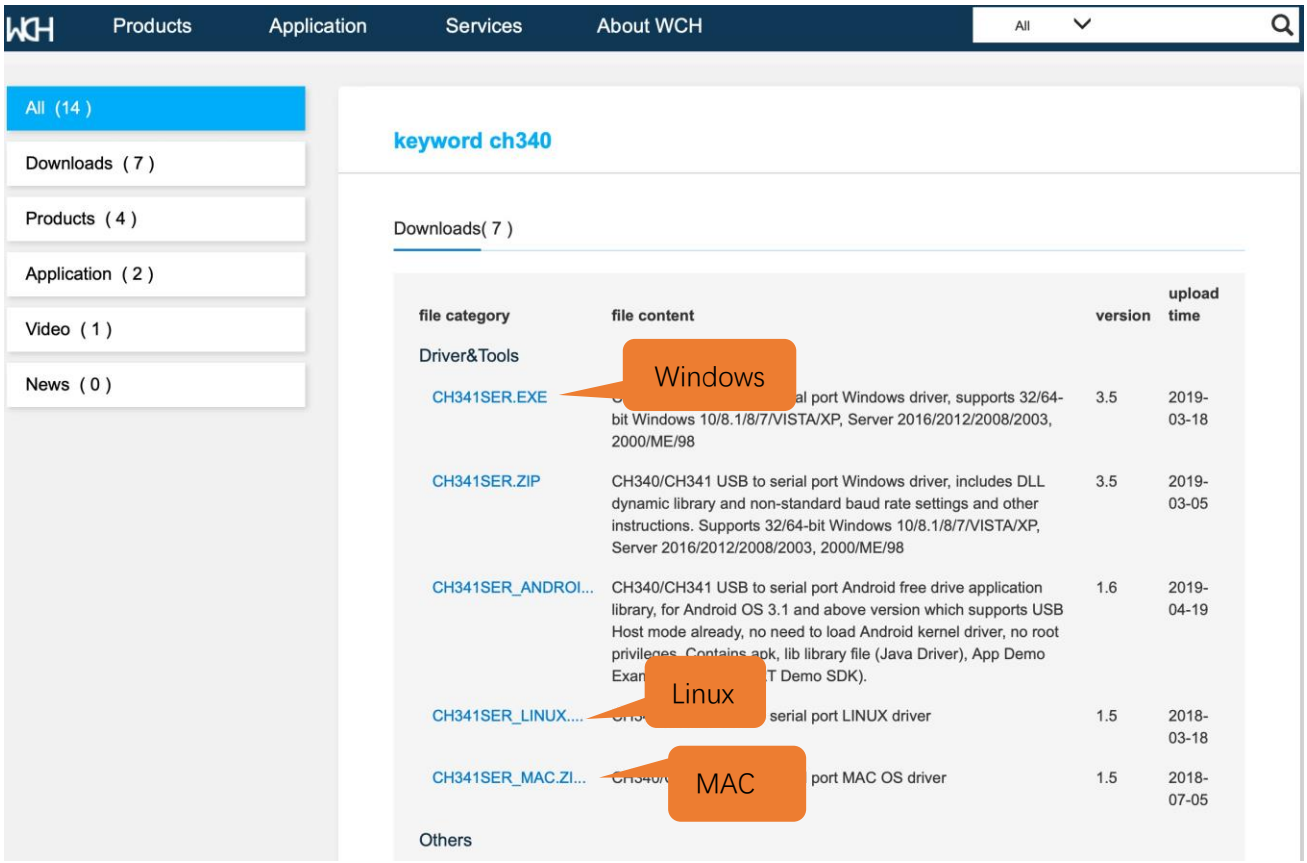
6. When ESP32 is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



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

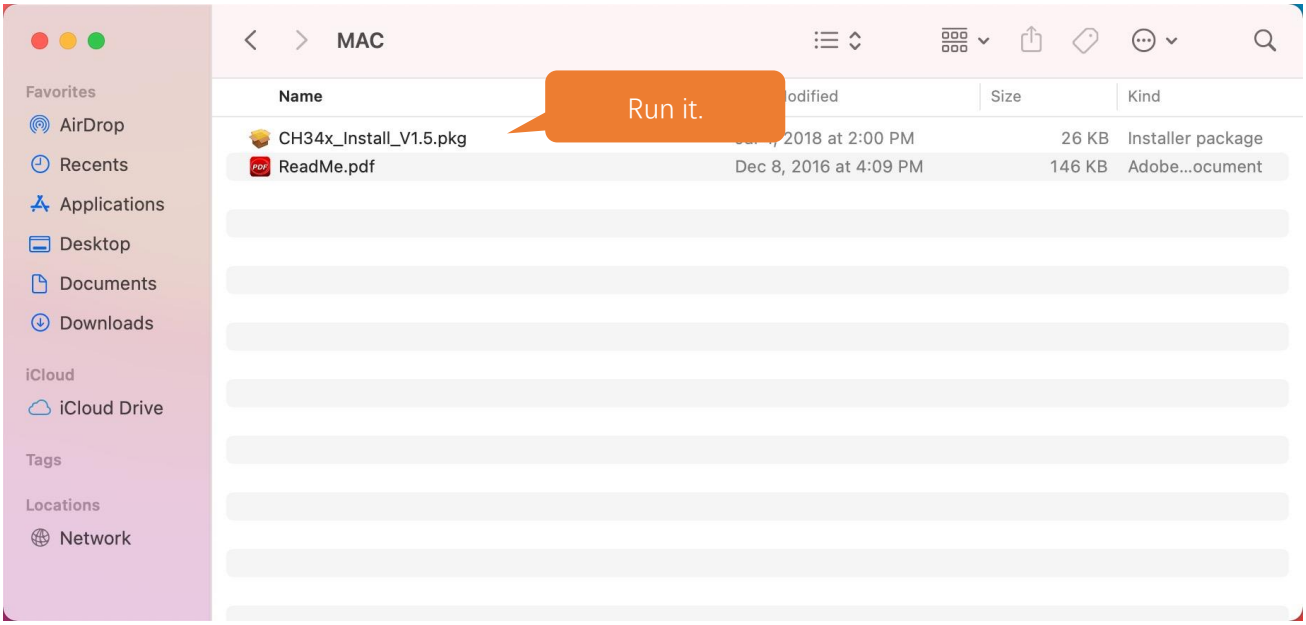
MAC

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

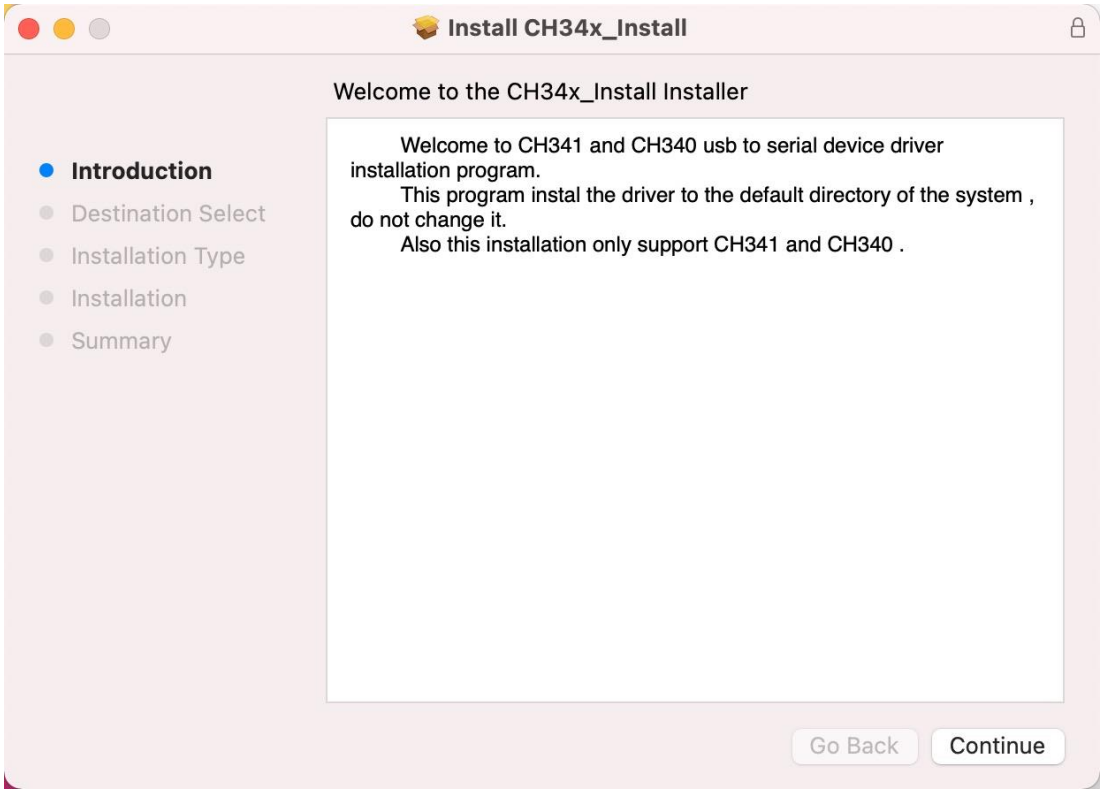


If you would not like to download the installation package, you can open **"Freenove_Basic_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.

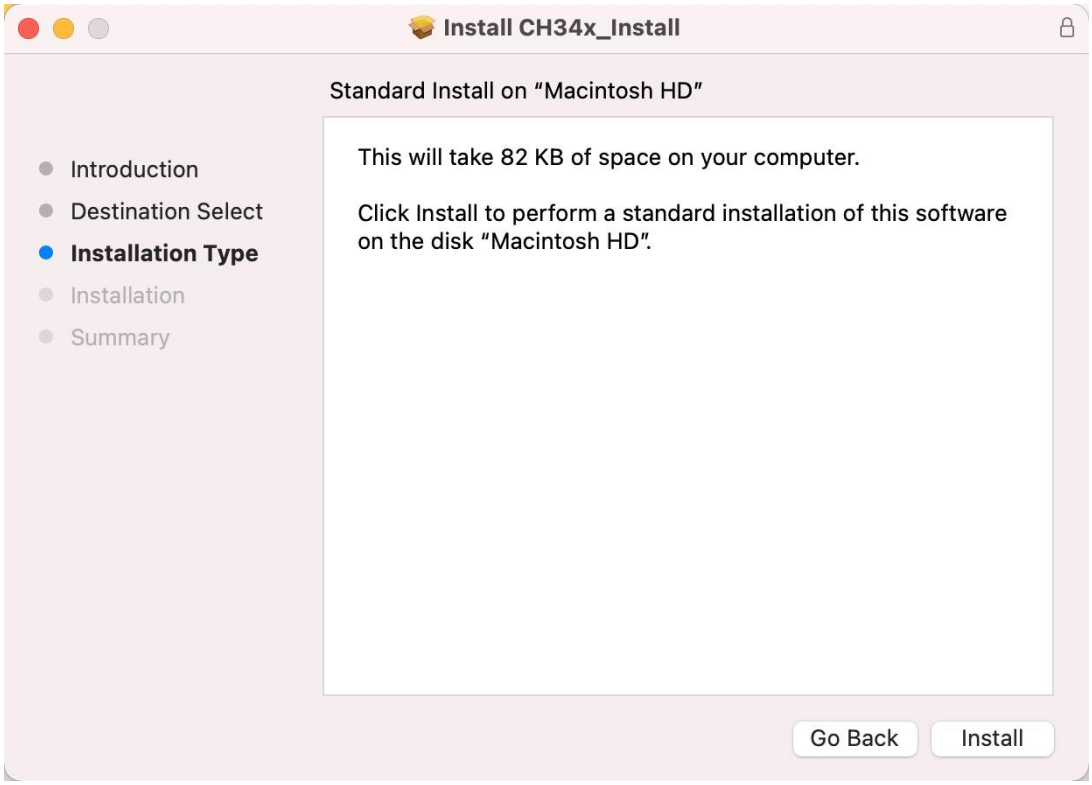
Second, open the folder "**Freenove_Basic_Starter_Kit_for_ESP32/CH340/MAC/**"



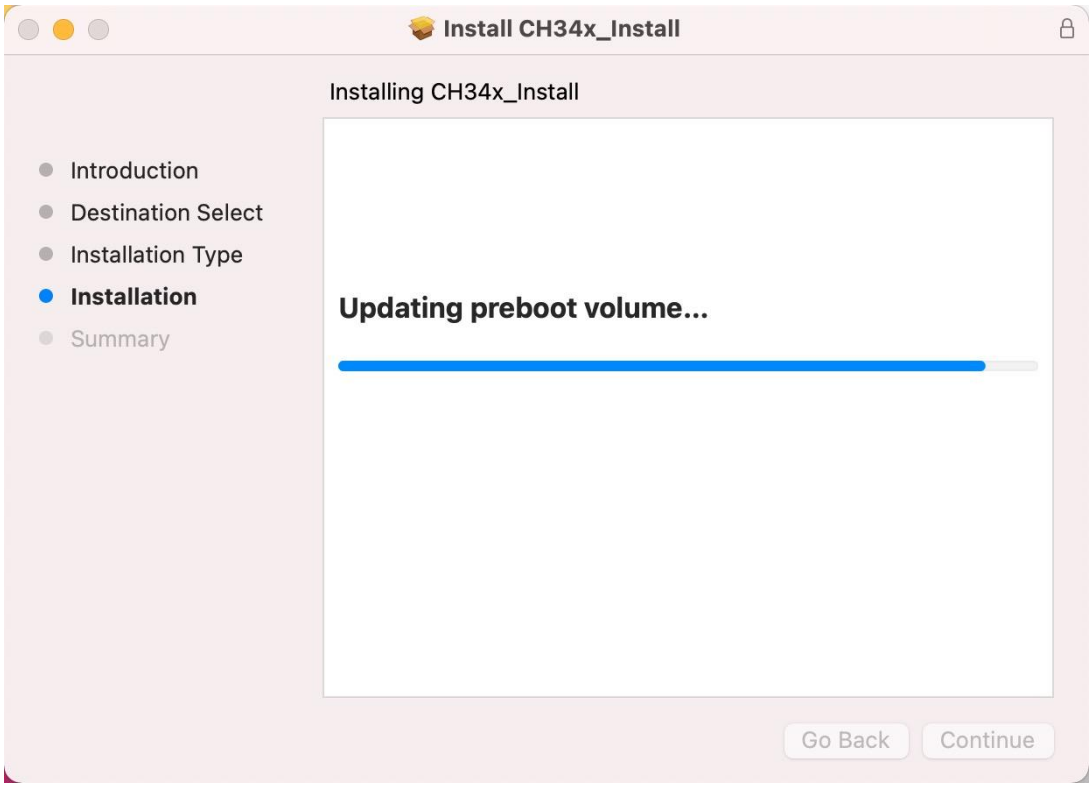
Third, click Continue.



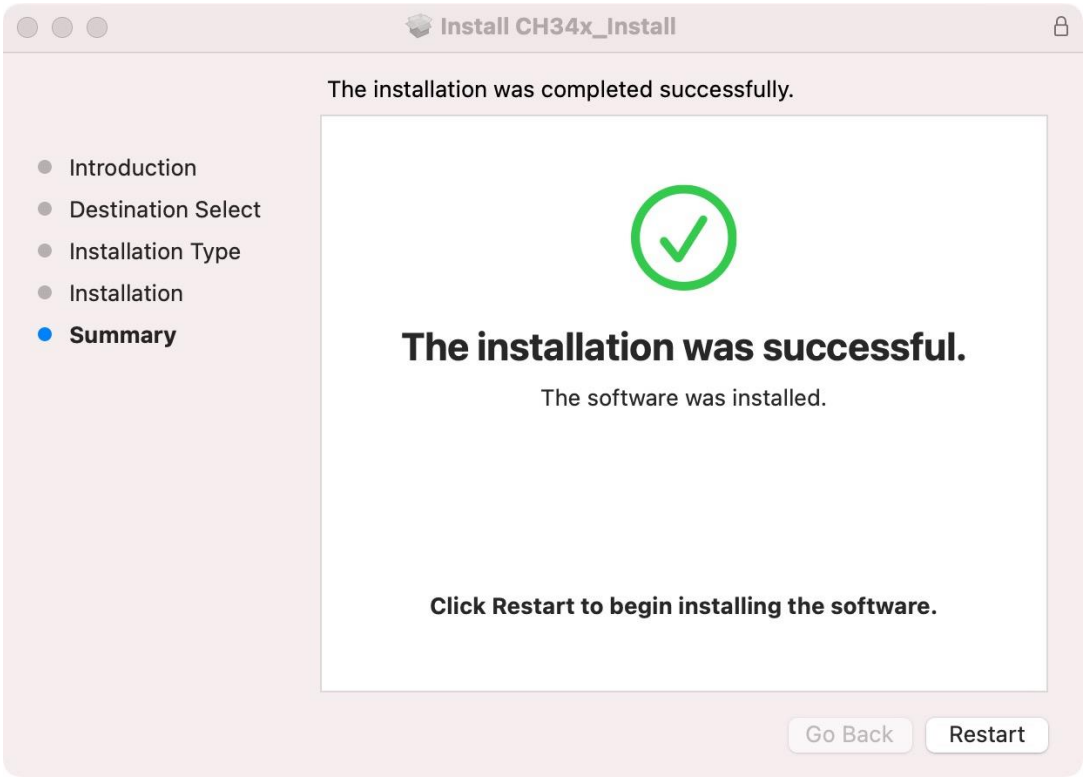
Fourth, click Install.



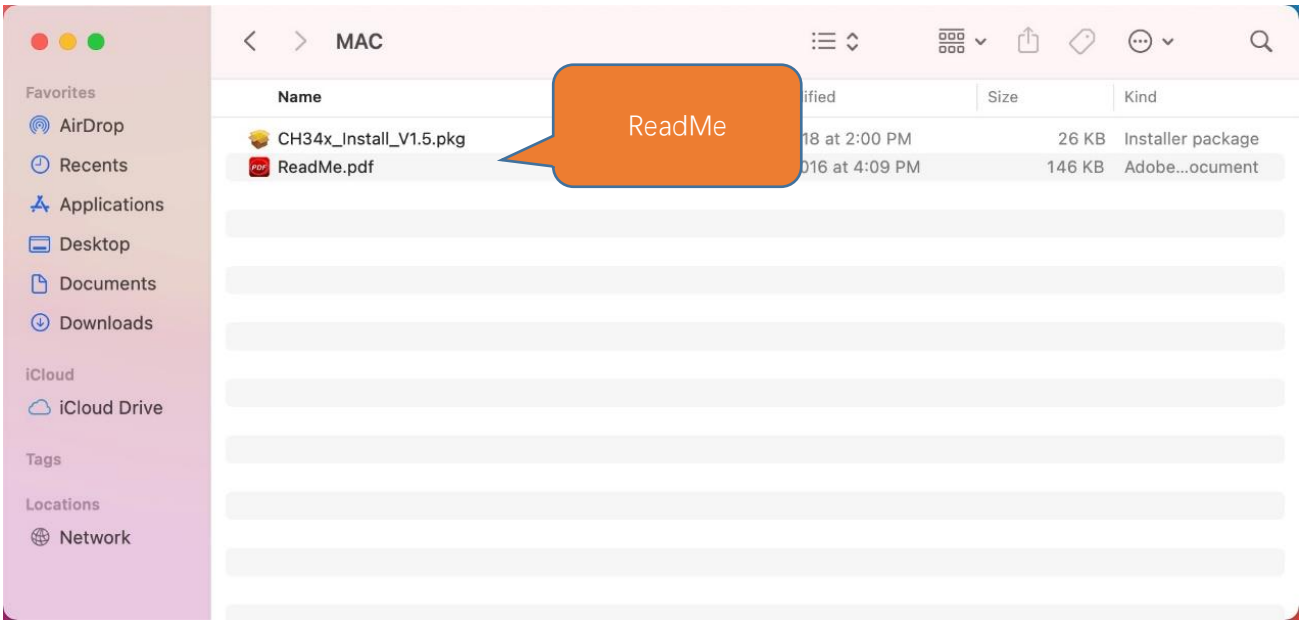
Then, waiting Finsh.



Finally, restart your PC.



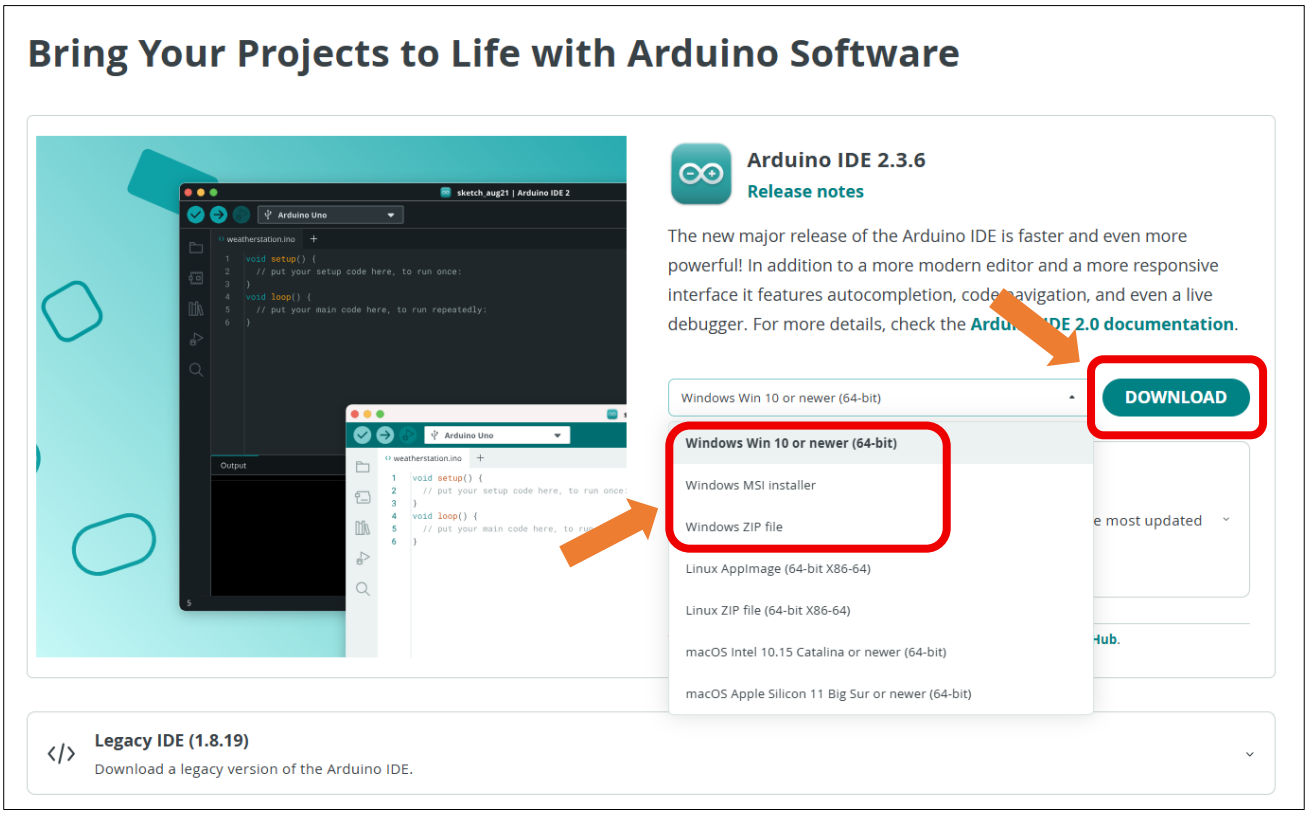
If you still haven't installed the CH340 by following the steps above, you can view readme.pdf to install it.



Programming Software

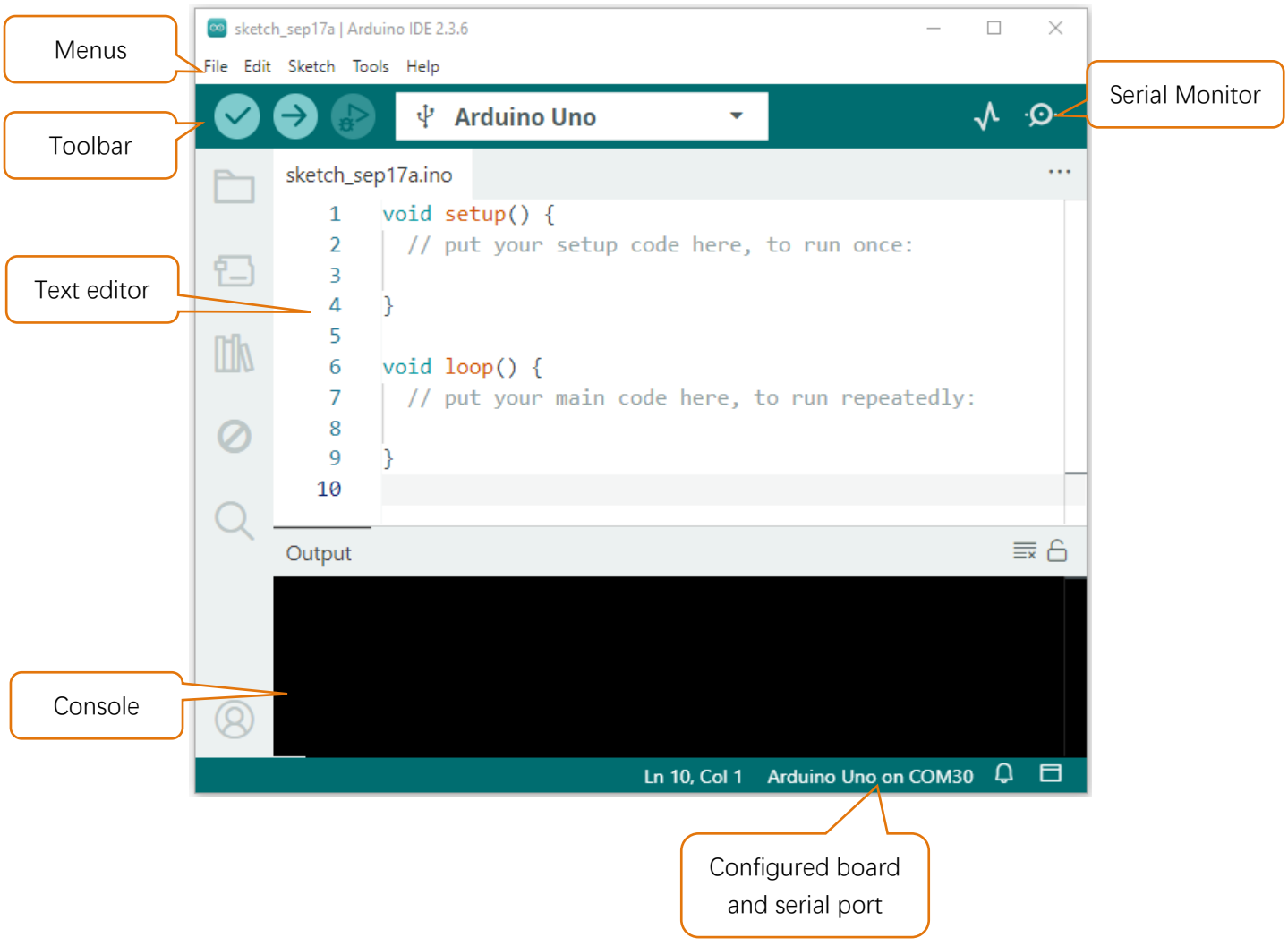
Arduino Software (IDE) is used to write and upload the code for Arduino Board.
First, install Arduino Software (IDE): visit https://www.arduino.cc/en/software/

Select and download corresponding installer based on your operating system. If you are a Windows user, please select the "Windows" to download and install the driver correctly.



After the downloading completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it is popped up, please allow the installation.
After installation is completed, an shortcut will be generated in the desktop.

Run it. The interface of the software is as follows:

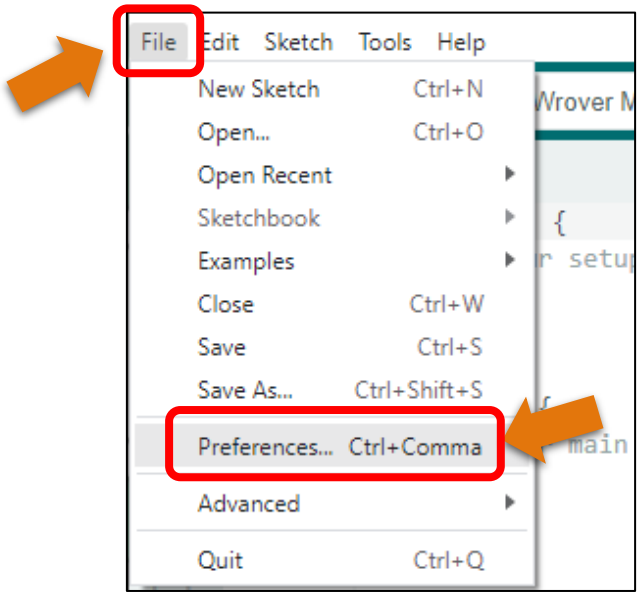


Programs written with Arduino IDE are called sketches. These sketches are written in a text editor and are saved with the file extension.ino. The editor has features for cutting/pasting and for searching/replacing text. The console displays text output by the Arduino IDE, including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, open the serial monitor, and access the serial plotter.

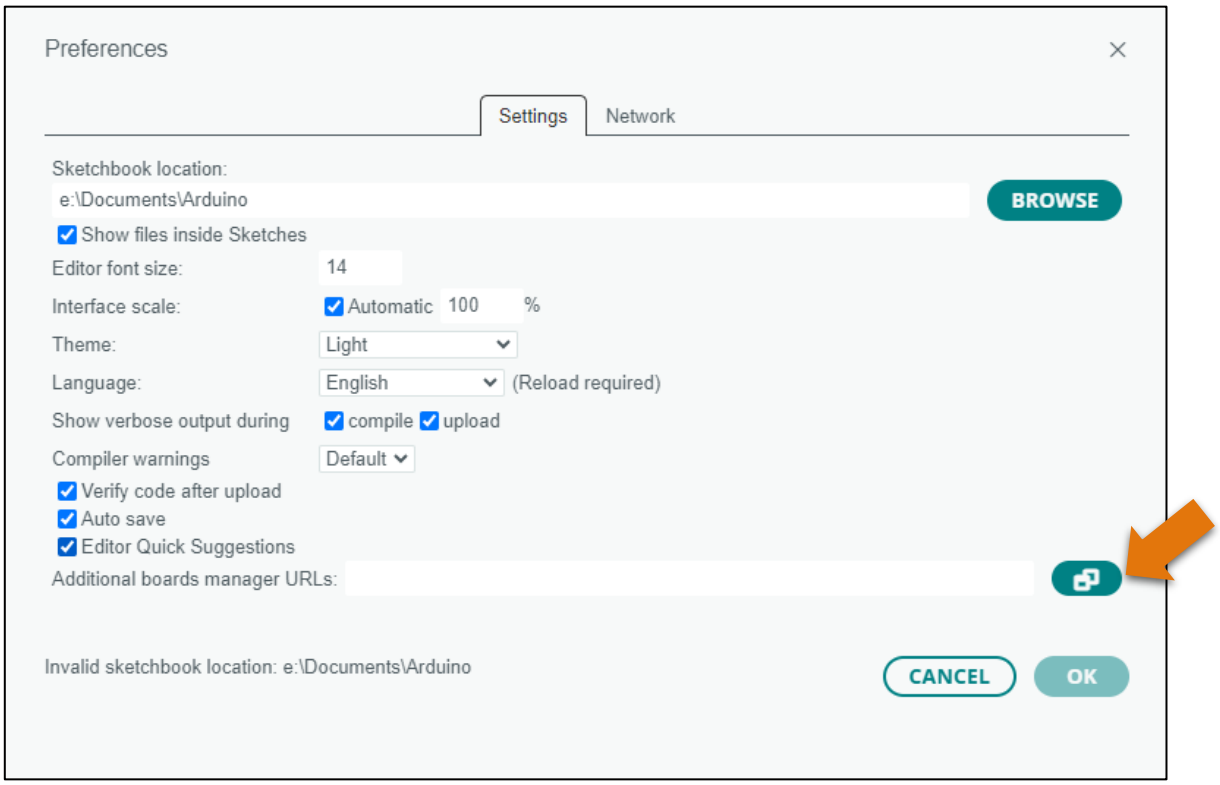
	Verify Checks your code for errors compiling it.
	Upload Compiles your code and uploads it to the configured board.
	Debug Troubleshoot code errors and monitor program running status.
	Serial Plotter Real-time plotting of serial port data charts.
	Serial Monitor Used for debugging and communication between devices and computers.

Environment Configuration

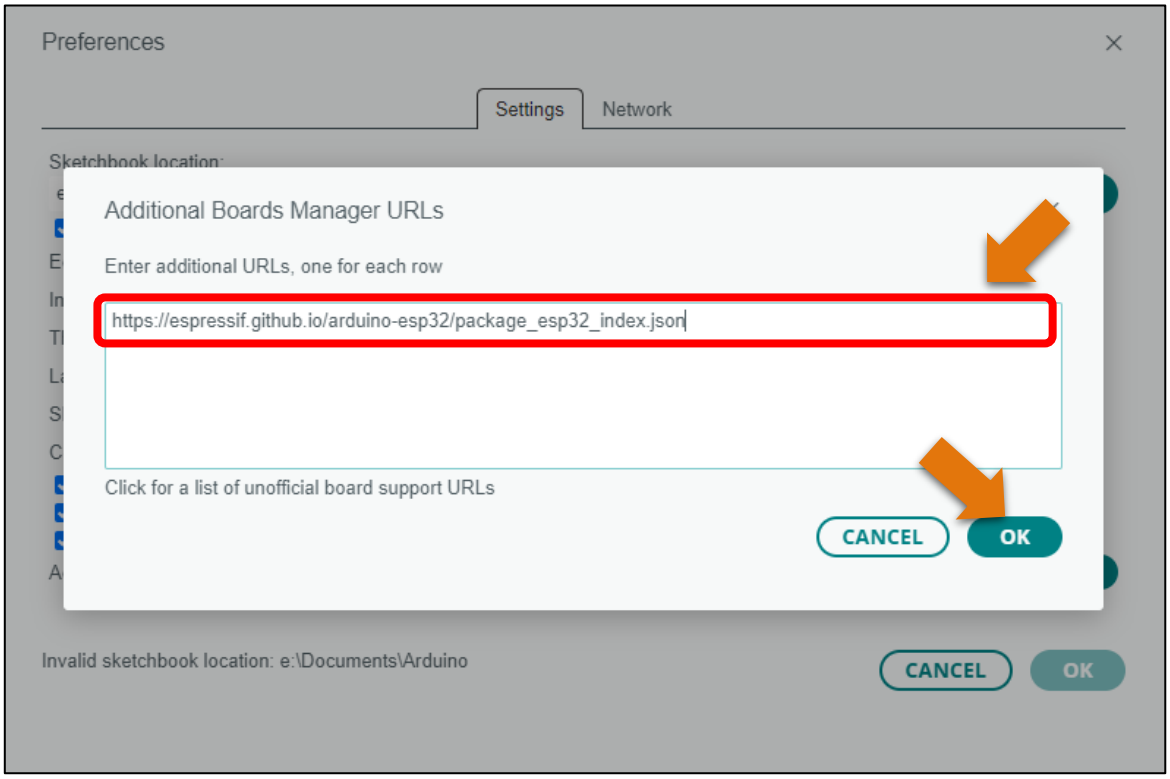
First, open the software platform arduino, and then click File in Menus and select Preferences.



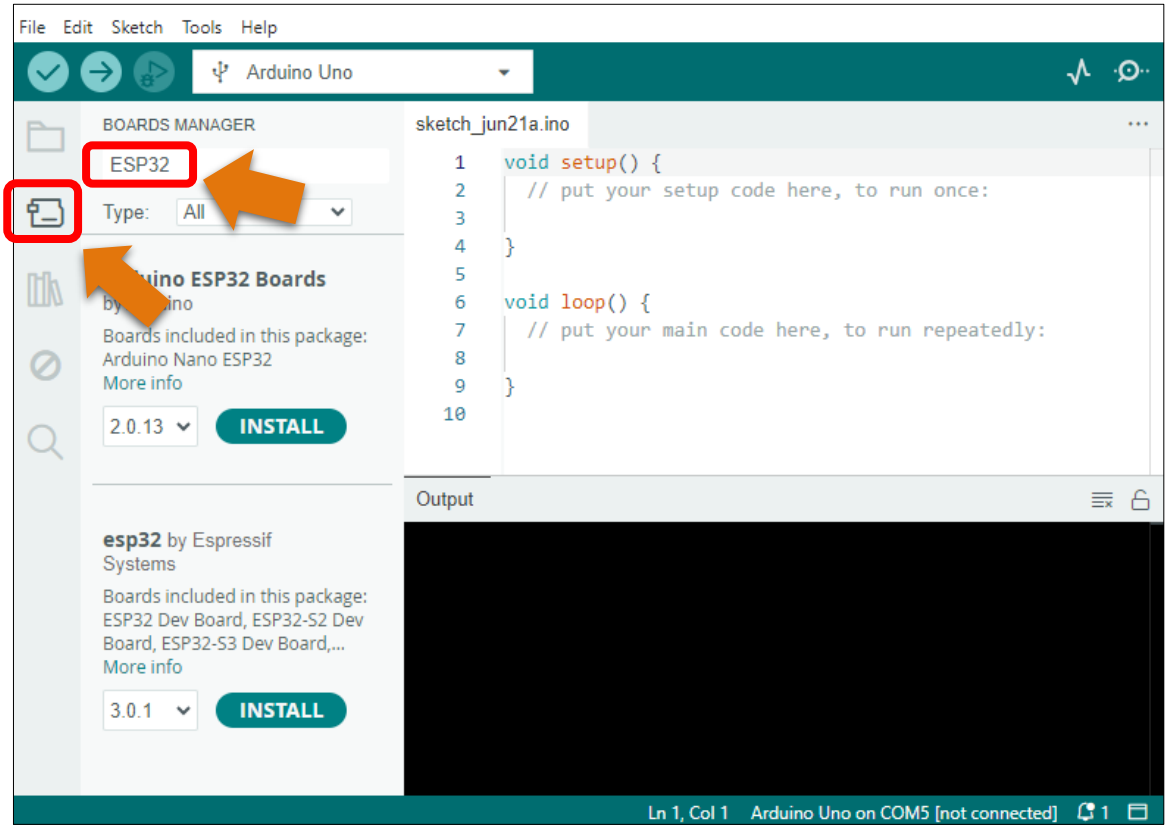
Second, click on the symbol behind "Additional Boards Manager URLs"



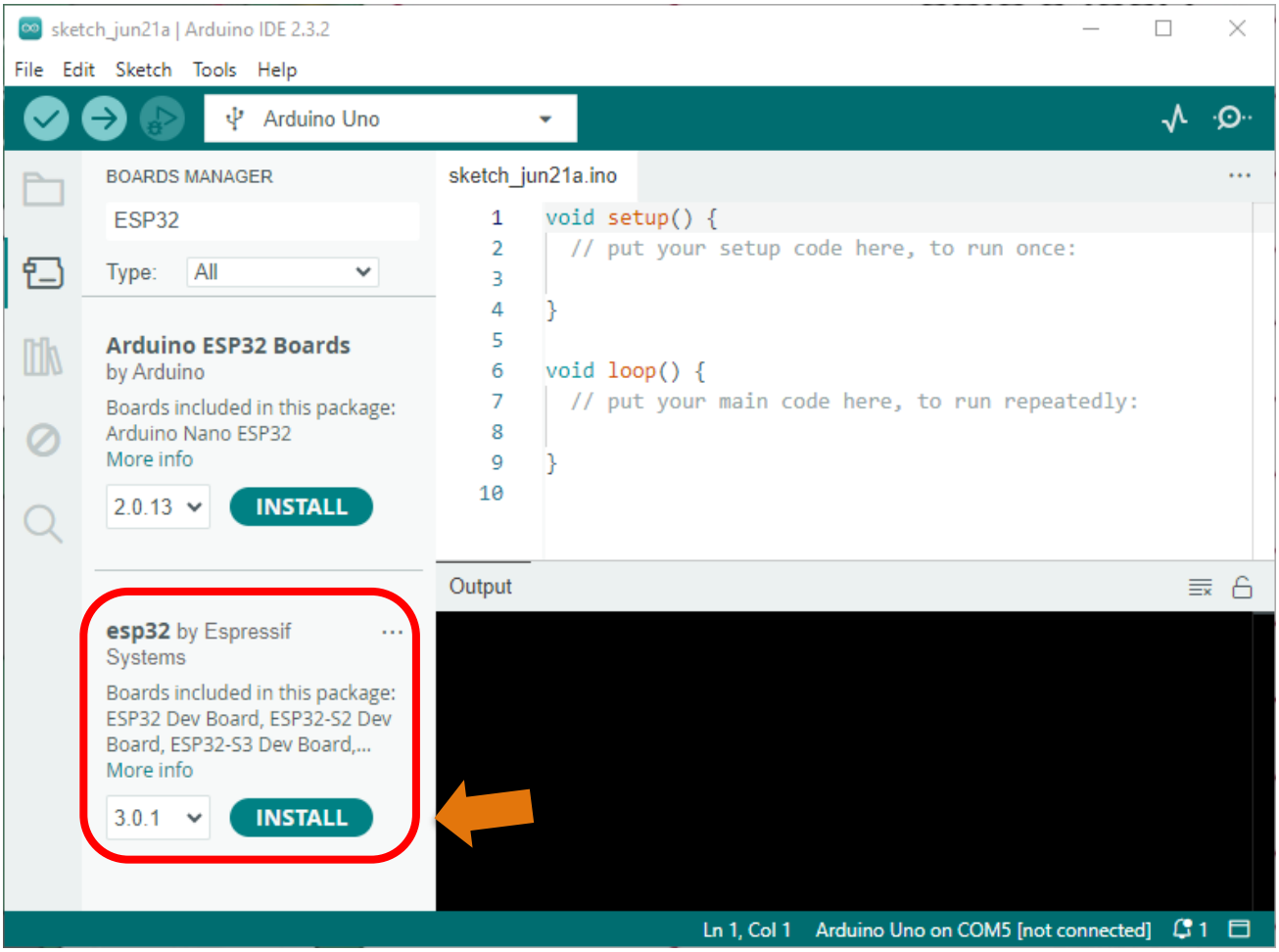
Third, fill in https://espressif.github.io/arduino-esp32/package_esp32_index.json in the new window, click OK, and click OK on the Preferences window again.



Fourth, click **"BOARDS MANAGER"** on the left and type **"ESP32"** in the search box.

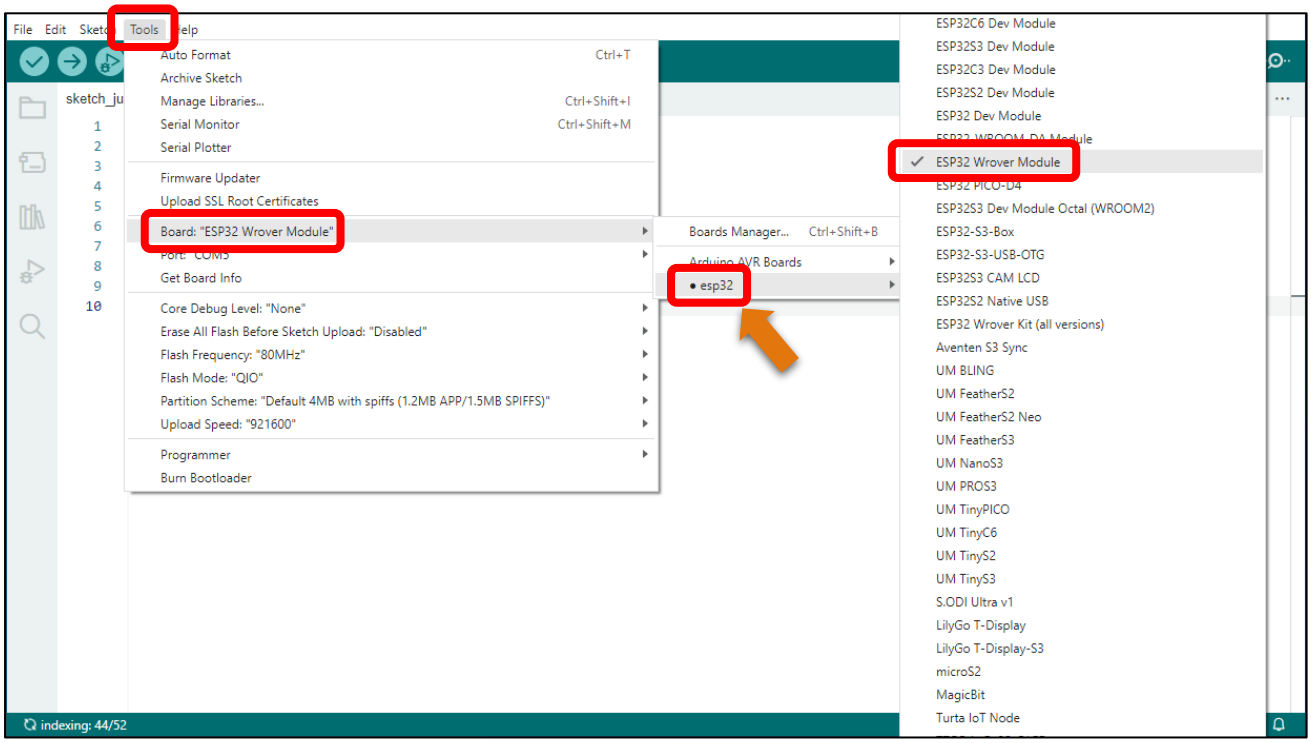


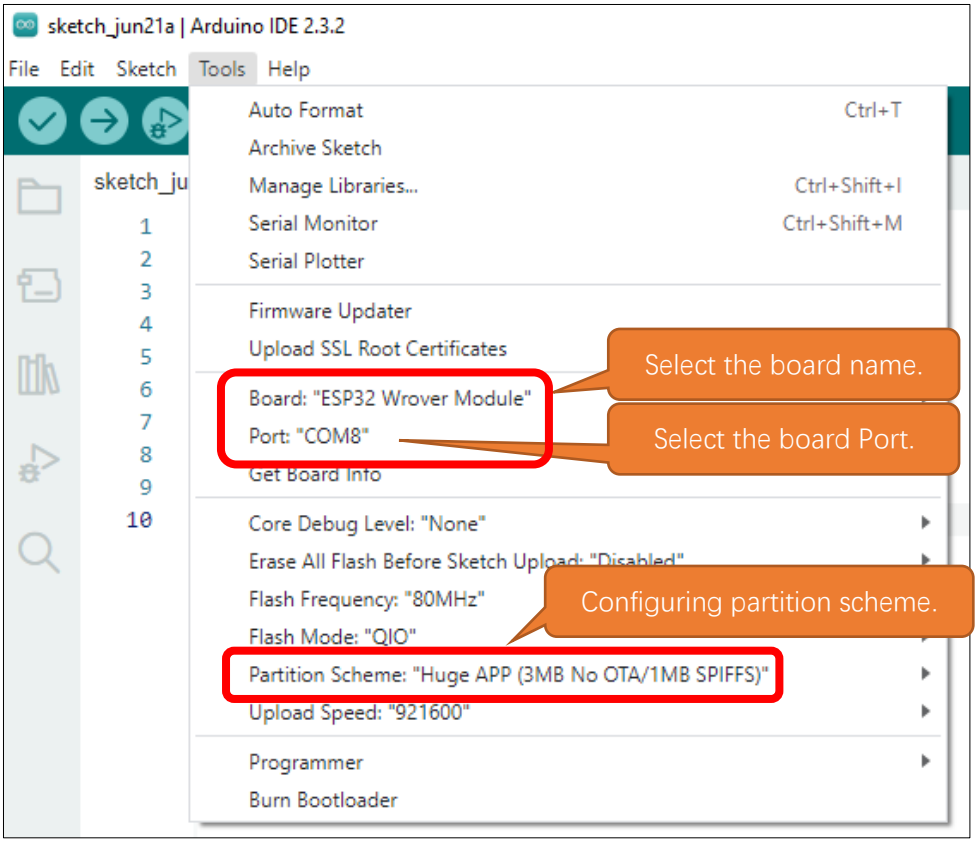
Fifth, select Espressif Systems' ESP32 and select version 3.0.x. Click **"INSTALL"** to install esp32.



Note: it takes a while to install the ESP32, make sure your network is stable.

When finishing installation, click Tools in the Menus again and select Board: "Arduino Uno", and then you can see information of **ESP32 Wrover Module**. Click " **ESP32 Wrover Module**" so that the ESP32 programming development environment is configured.





Notes for GPIO

Strapping Pin

There are five Strapping pins for ESP32：MTDI、GPIO0、GPIO2、MTDO、GPIO5。
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'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 five strapping pins at power-on and their functions under the corresponding configuration.

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

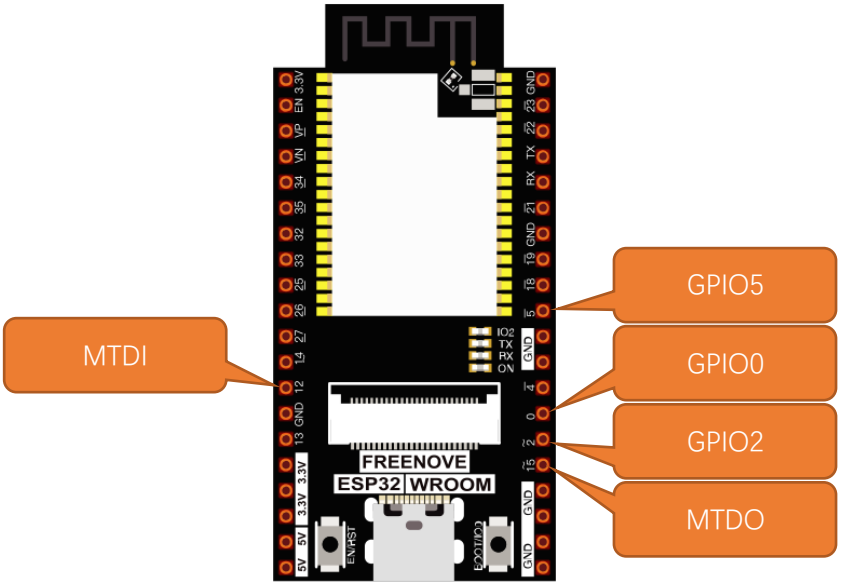
Timing of SDIO Slave					
Pin	Default	Falling-edge Sampling Falling-edge Output	Falling-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Note:

- Firmware can configure register bits to change the settings of "Voltage of Internal LDO (VDD_SDIO)" and "Timing of SDIO Slave" after booting.
- The MTDI is internally pulled high in the module, as the flash and SRAM in ESP32-WROVER only support a power voltage of 1.8 V (output by VDD_SDIO).

If you have any questions about the information of GPIO, you can click here to go back to ESP32-WROVER to view specific information about GPIO.
If you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through support@freenove.com at any time.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

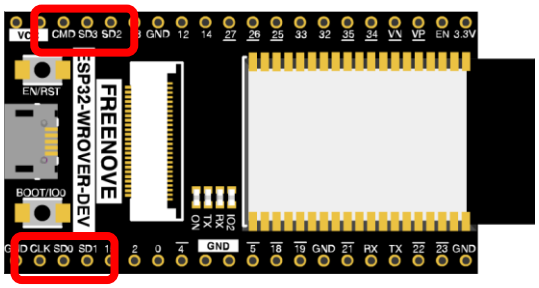
Strapping Pin



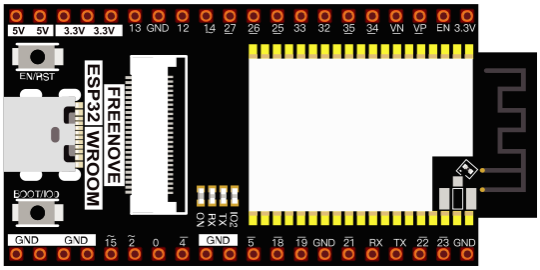
Flash Pin

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

In older versions, the flash pin looks like the image below.



In the new release, we no longer introduce GPIO6-11.



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

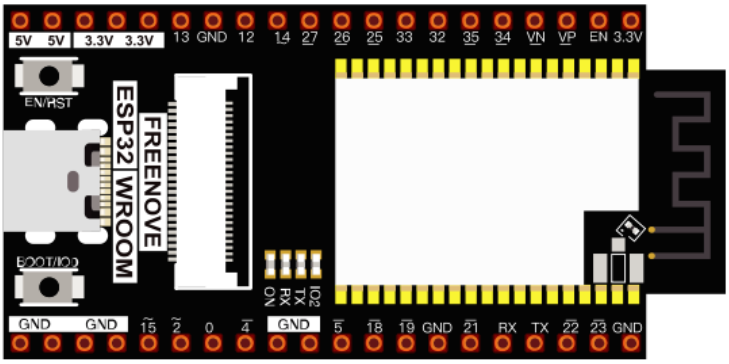
Because of external pull-up, MTDI pin is not suggested to be used as a touch sensor. For details, please refer to Peripheral Interface and Sensor chapter in "ESP32 Data_Sheet".

For more relevant information, please check:

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

Cam Pin

When using the camera of our ESP32-WROVER, 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
I2C_SDA	GPIO26
I2C_SCL	GPIO27
CSI_VYSNC	GPIO25
CSI_HREF	GPIO23
CSI_Y9	GPIO35
XCLK	GPIO21
CSI_Y8	GPIO34
CSI_Y7	GPIO39
CSI_PCLK	GPIO22
CSI_Y6	GPIO36
CSI_Y2	GPIO4
CSI_Y5	GPIO19
CSI_Y3	GPIO5
CSI_Y4	GPIO18

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

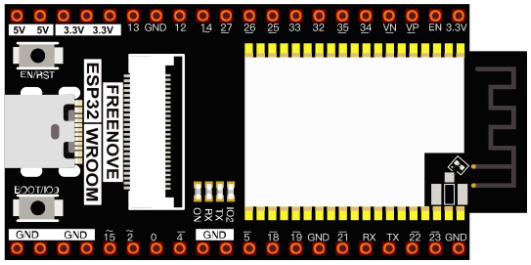
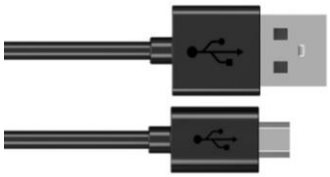
Chapter 0 LED

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

Project 0.1 Blink

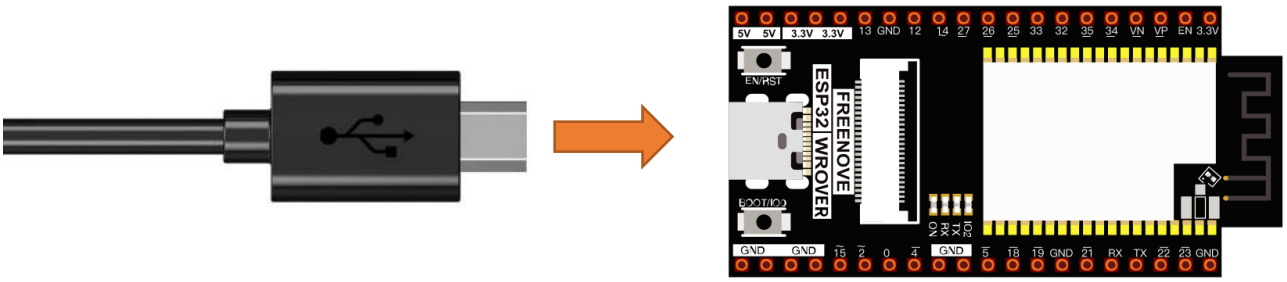
In this project, we will use ESP32 to control blinking a common LED.

Component List

ESP32-WROVER x1	USB cable
	

Power

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



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

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

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

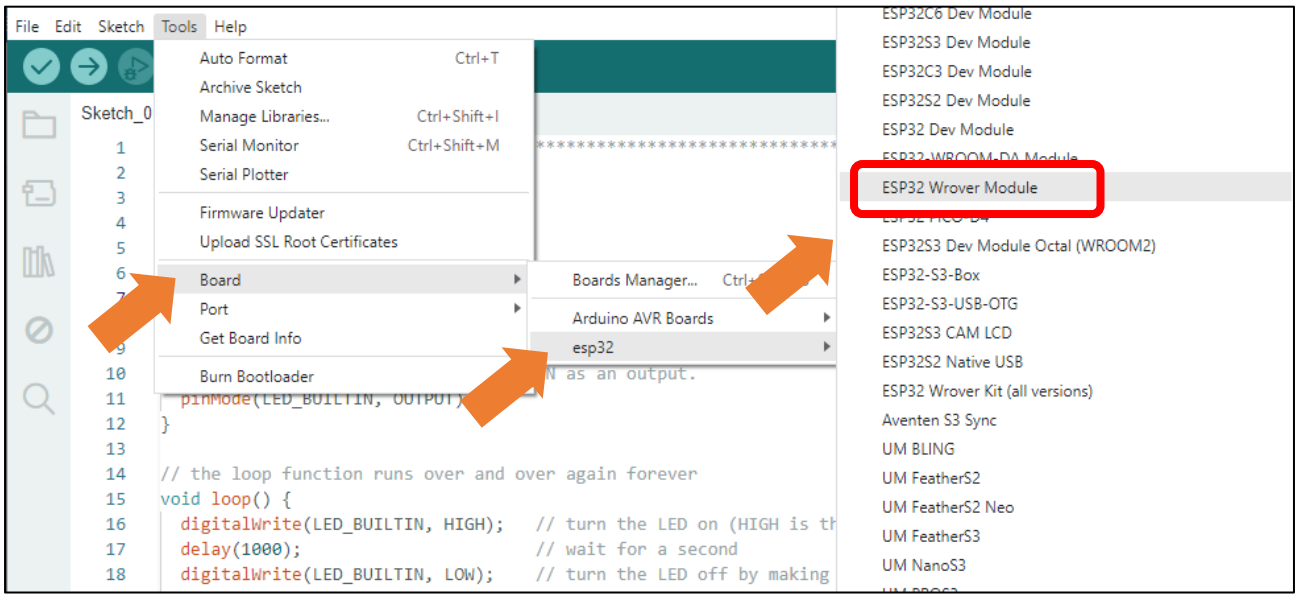
Sketch

According to the circuit, when the GPIO2 of ESP32-WROVER output level is low, the LED turns ON. Conversely, when the GPIO2 ESP32-WROVER output level is high, the LED turns OFF. Therefore, we can let GPIO2 circularly output high and low level to make the LED blink.

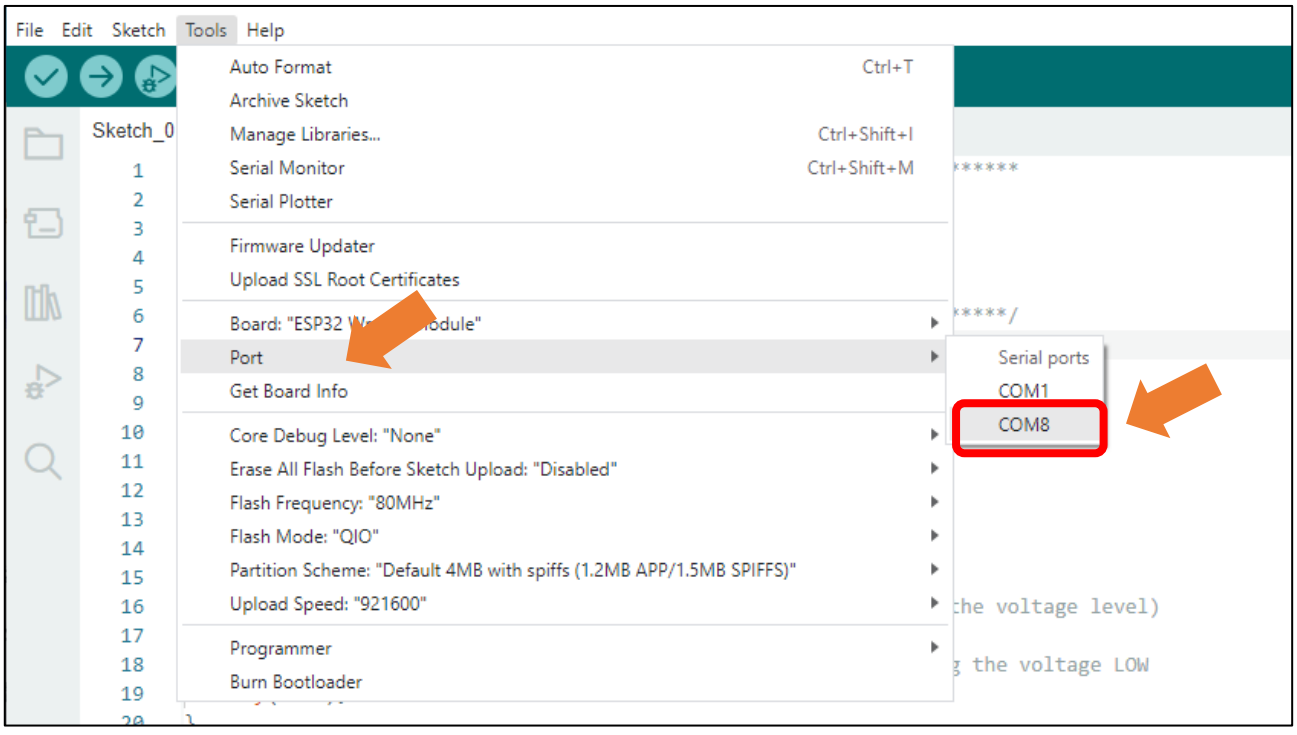
Upload the following Sketch:

Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketch_01.1_Blink.

Before uploading the code, click "**Tools**", "**Board**" and select "**ESP32 Wrover Module**".

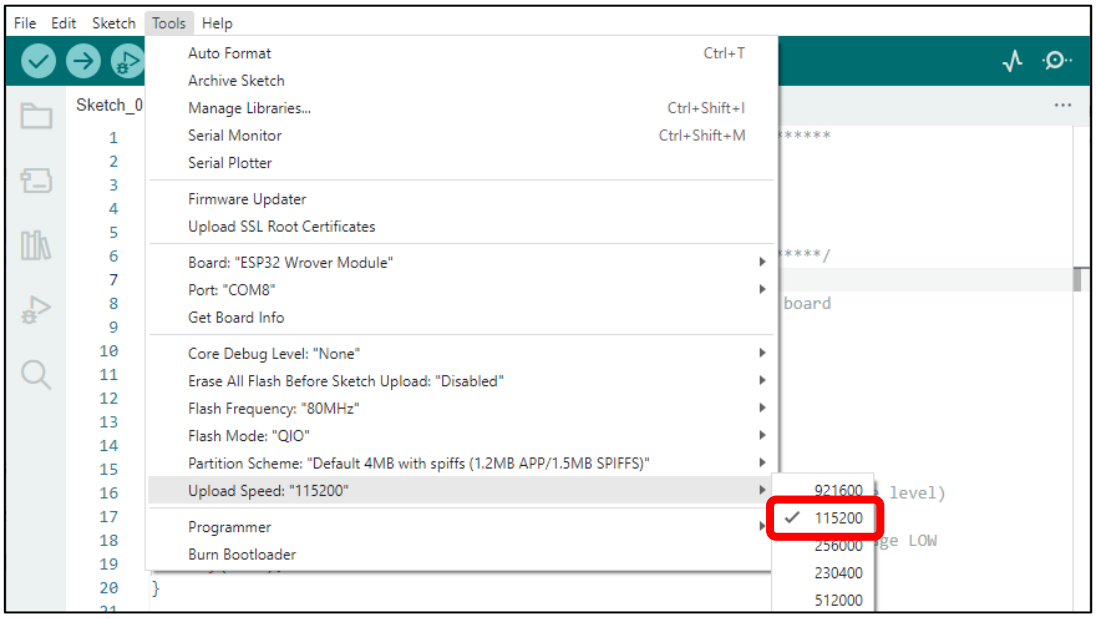


Select the serial port.

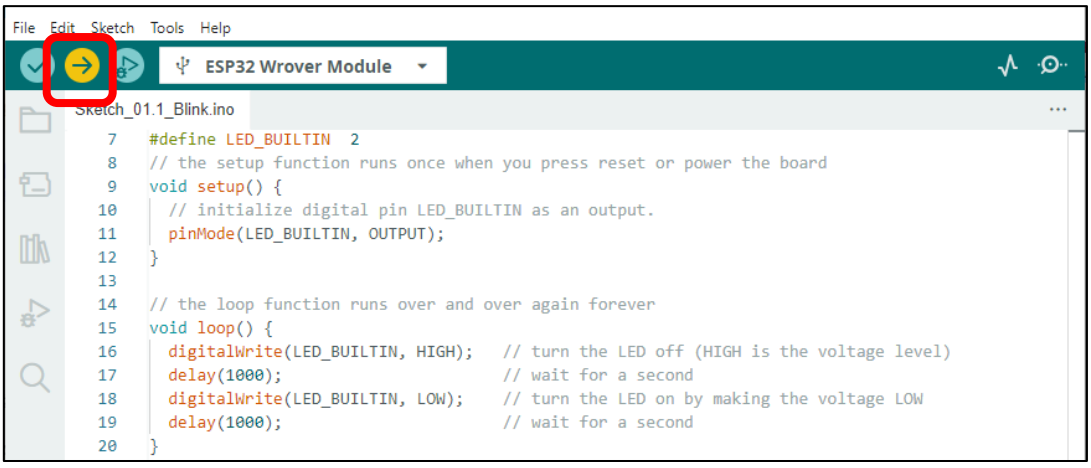


Note: For macOS users, if the uploading fails, please set the baud rate to 115200 before clicking

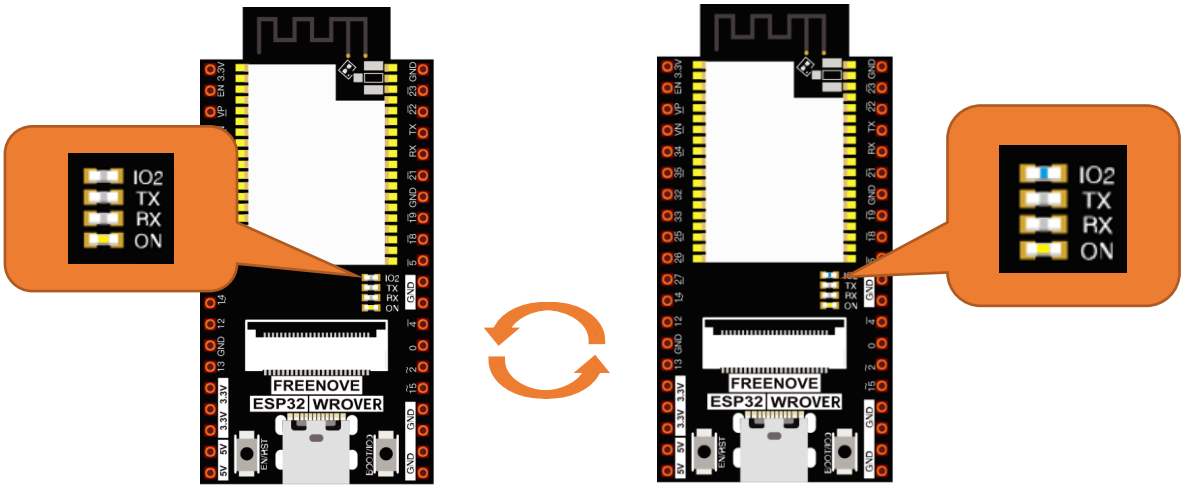
"Upload Speed".



Sketch_01.1_Blink



Click "Upload", Download the code to ESP32-WROVER and your LED in the circuit starts Blink.

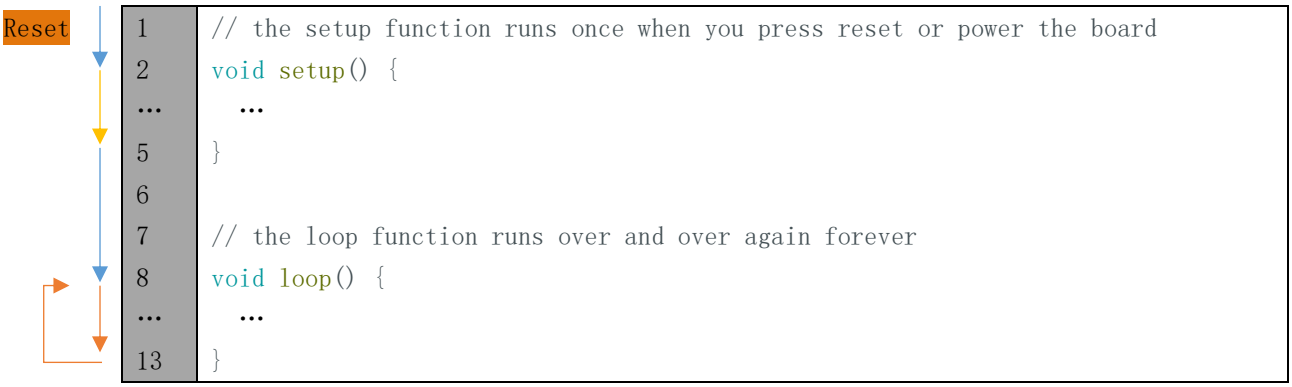


If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```
#define PIN_LED 2
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(PIN_LED, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(PIN_LED, HIGH);   // turn the LED off (HIGH is the voltage level)
  delay(1000);                   // wait for a second
  digitalWrite(PIN_LED, LOW);    // turn the LED on by making the voltage LOW
  delay(1000);                   // wait for a second
}
```

The Arduino IDE code usually contains two basic functions: void setup() and void loop().
After the board is reset, the setup() function will be executed firstly, and then the loop() function.
setup() function is generally used to write code to initialize the hardware. And loop() function is used to write code to achieve certain functions. loop() function is executed repeatedly. When the execution reaches the end of loop(), it will jump to the beginning of loop() to run again.



Reset

Reset operation will lead the code to be executed from the beginning. Switching on the power, finishing uploading the code and pressing the reset button will trigger reset operation.

In the circuit, ESP32-WROVER's GPIO2 is connected to the LED, so the LED pin is defined as 2.

```
#define PIN_LED 2
```

This means that after this line of code, all PIN_LED will be treated as 2.
In the setup () function, first, we set the PIN_LED as output mode, which can make the port output high level or low level.

```
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
```

Then, in the loop () function, set the PIN_LED to output high level to make LED light off.

```
  digitalWrite(PIN_LED, HIGH);  // turn the LED off (HIGH is the voltage level)
```

Wait for 1000ms, that is 1s. Delay () function is used to make control board wait for a moment before executing the next statement. The parameter indicates the number of milliseconds to wait for.

```
11    delay(1000);                    // wait for a second
```

Then set the PIN_LED to output low level, and LED light up. One second later, the execution of loop () function will be completed.

```
12    digitalWrite(PIN_LED, LOW);    // turn the LED on by making the voltage LOW
13    delay(1000);                    // wait for a second
```

The loop() function is constantly being executed, so LED will keep blinking.

Reference

void pinMode(int pin, int mode);

Configures the specified pin to behave either as an input or an output.

Parameters

pin: the pin number to set the mode of.

mode: INPUT, OUTPUT, INPUT_PULLDOWM, or INPUT_PULLUP.

void digitalWrite (int pin, int value);

Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

For more related functions, please refer to https://www.arduino.cc/reference/en/

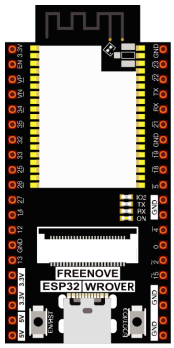
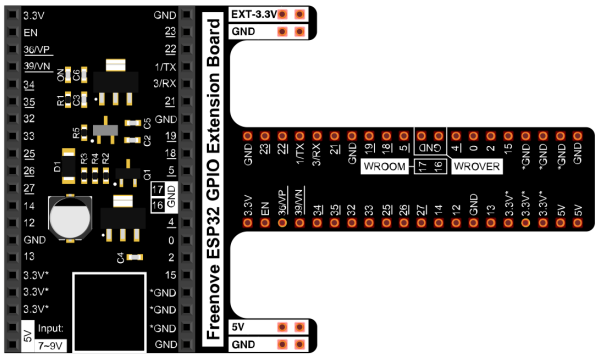
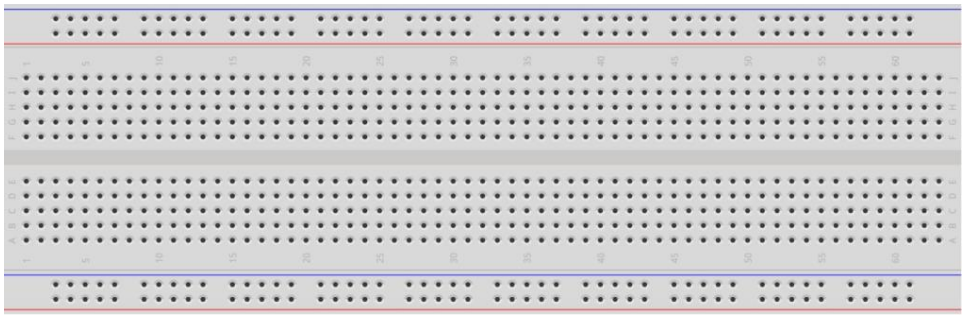
Chapter 1 LED

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

Project 1.1 Blink

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

Component List

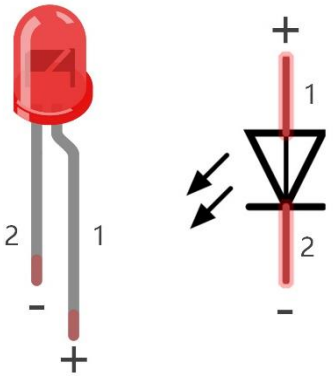
<table>
<tr><td>ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper M/M x2
  </td></tr>
</table>

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.



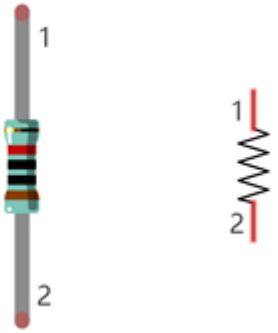
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

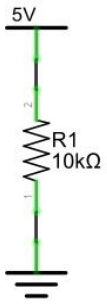
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

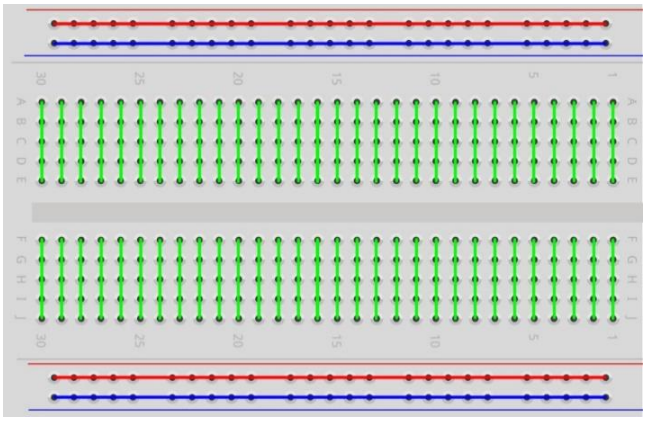


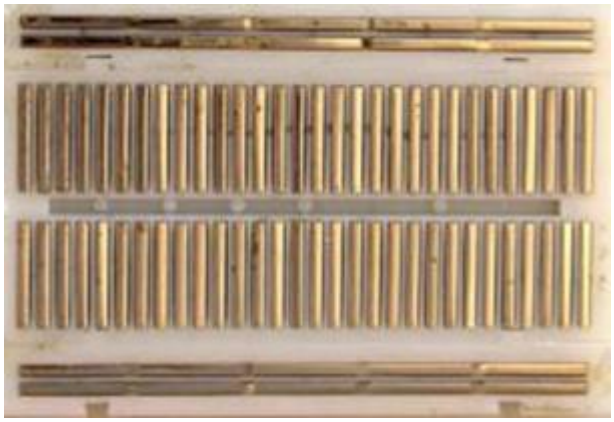
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

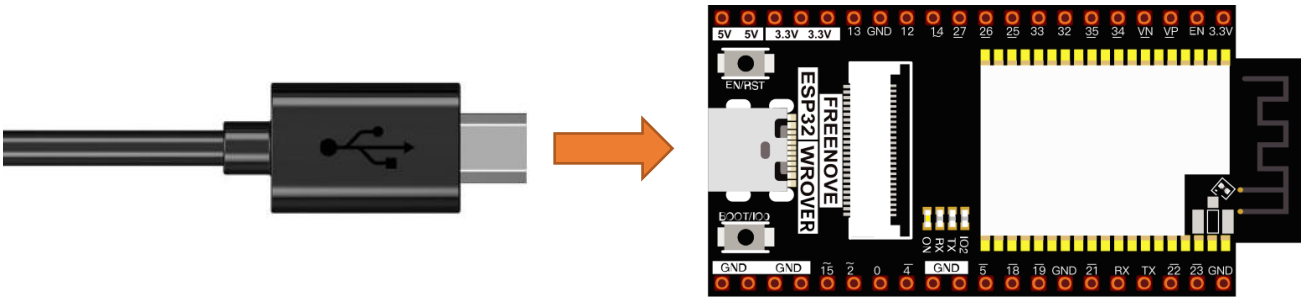
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

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



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

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

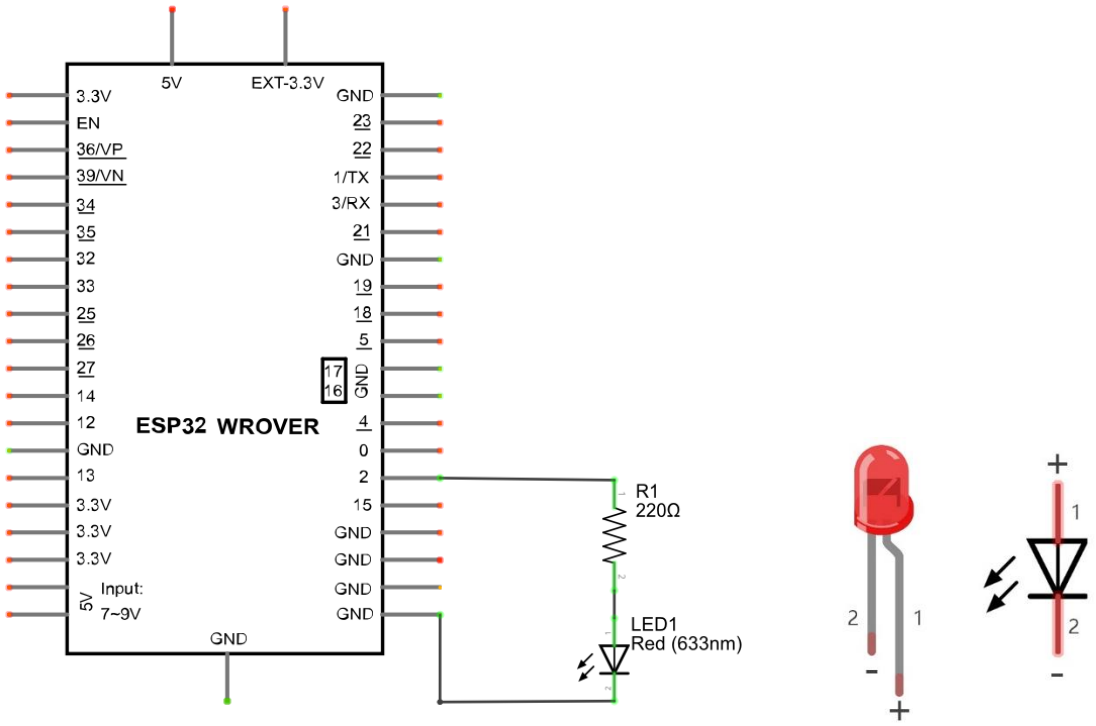
We can also use DC jack of extension board to power ESP32-WROVER.In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

Circuit

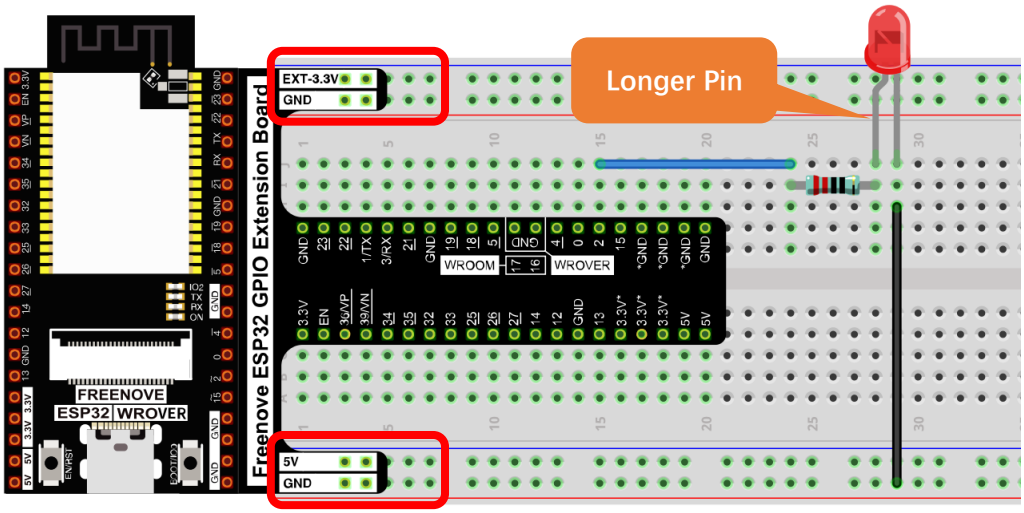
First, disconnect all power from the ESP32-WROVER. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP32-WROVER.

CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)! WARNING: A short circuit can cause high current in your circuit, 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-WROVER 180° for connection.

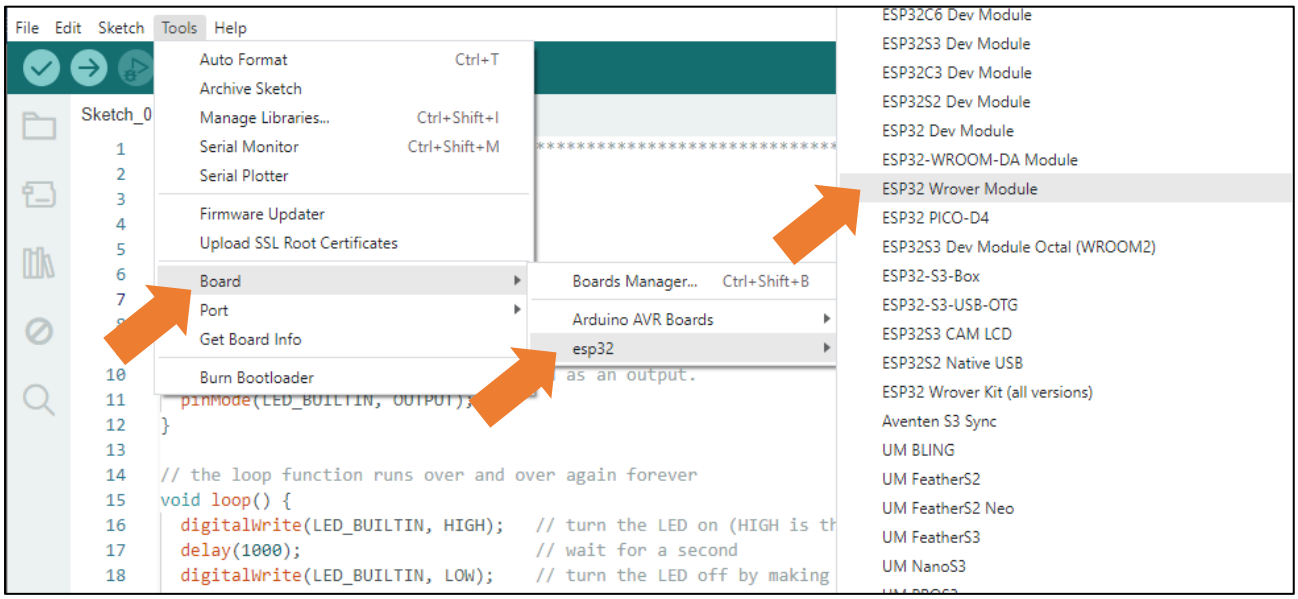
Sketch

According to the circuit, when the GPIO2 of ESP32-WROVER output level is high, the LED turns ON. Conversely, when the GPIO2 ESP32-WROVER output level is low, the LED turns OFF. Therefore, we can let GPIO2 circularly output high and low level to make the LED blink.

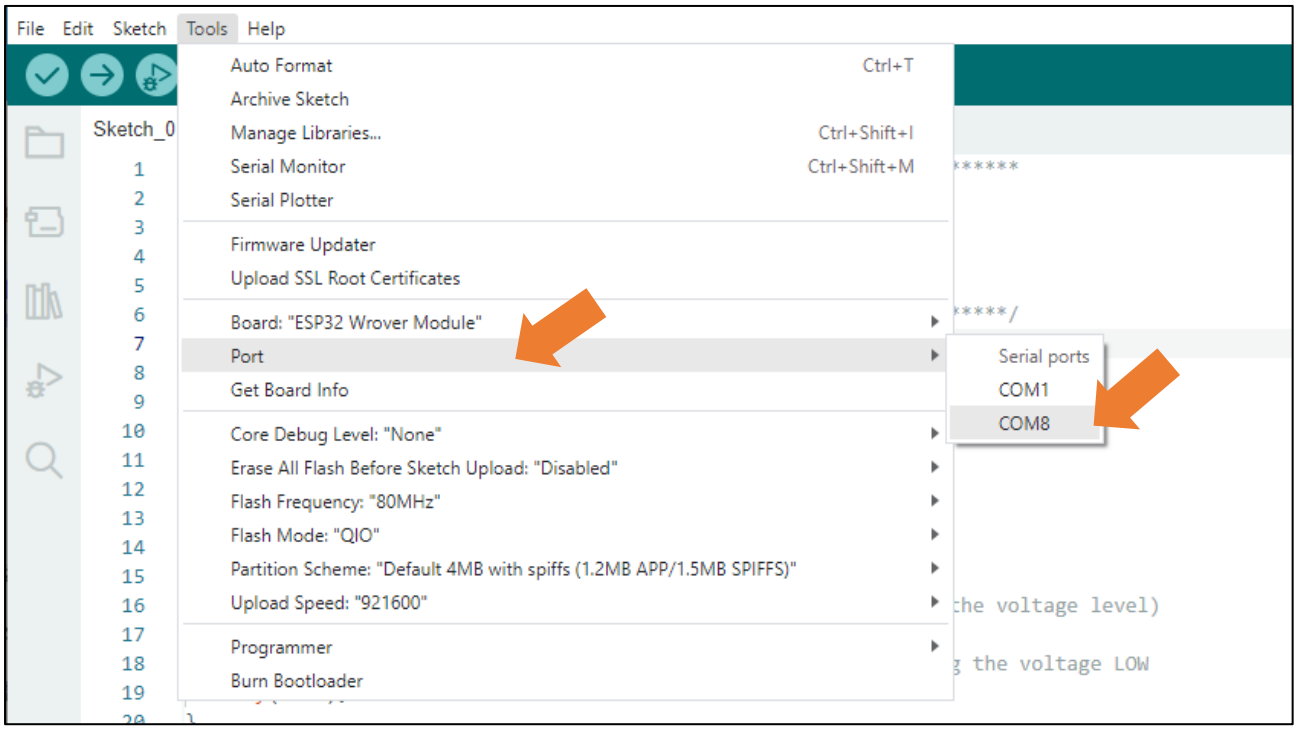
Upload the following Sketch:

Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketch_01.1_Blink.

Before uploading the code, click "**Tools**", "**Board**" and select "**ESP32 Wrover Module**".

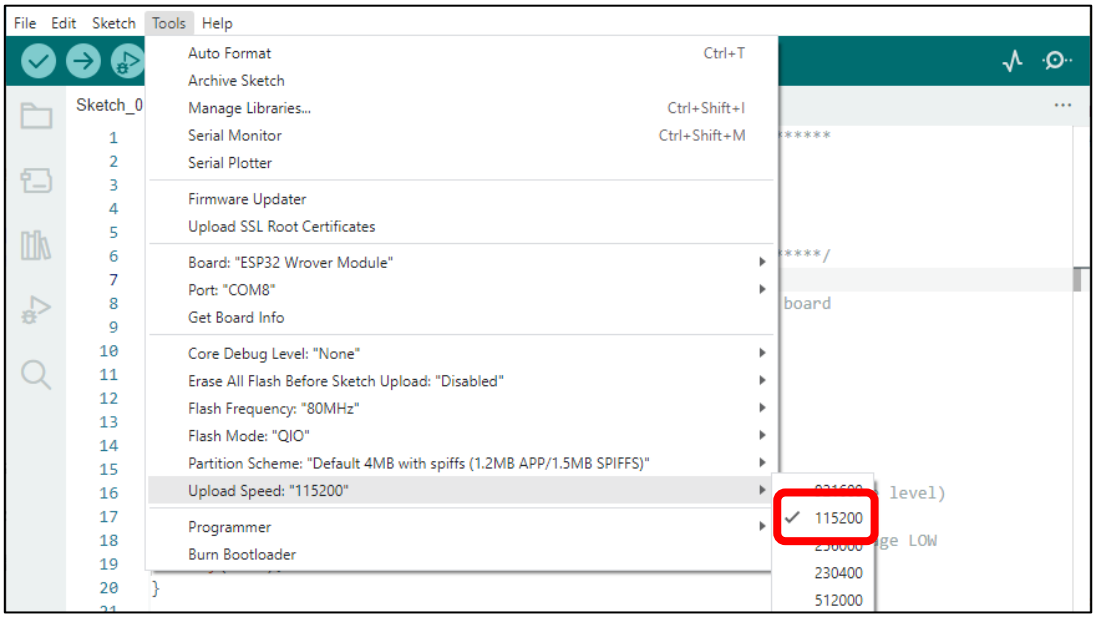


Select the serial port.

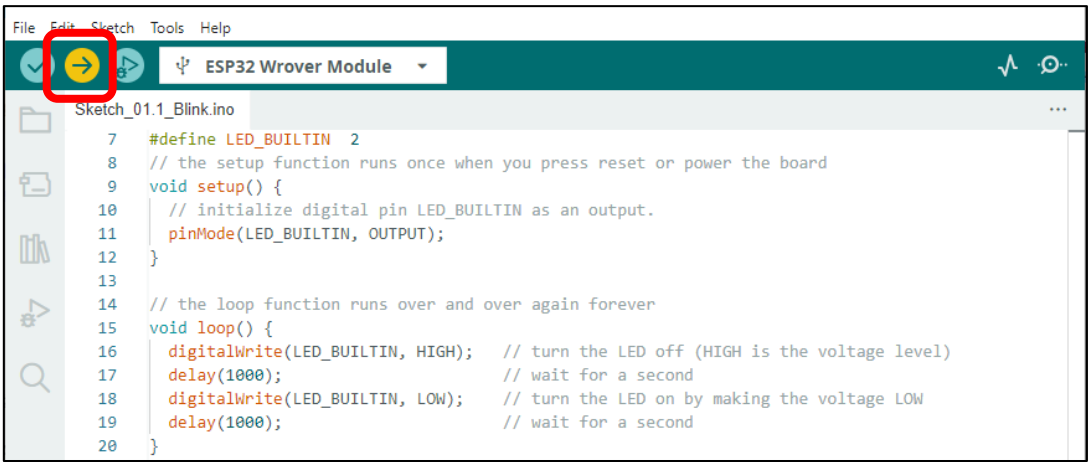


Note: For macOS users, if the uploading fails, please set the baud rate to 115200 before clicking

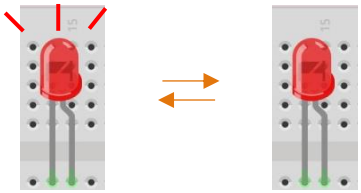
“Upload Speed”.



Sketch_01.1_Blink



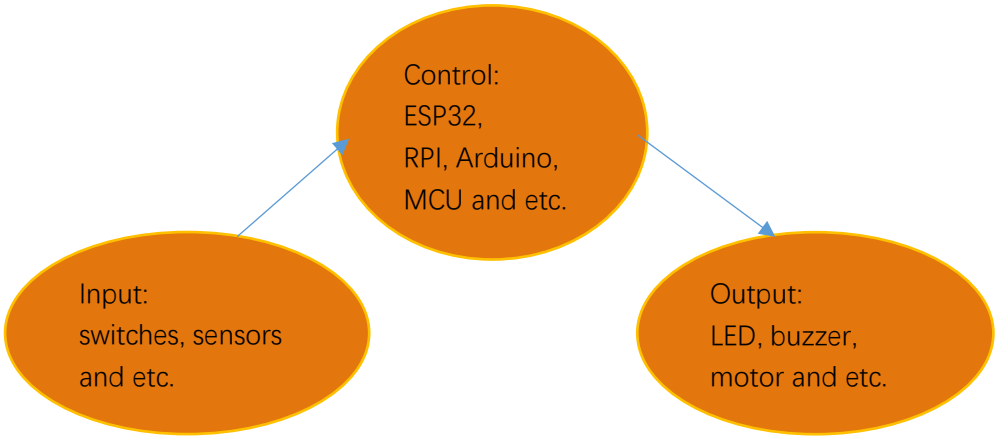
Click “Upload”, Download the code to ESP32-WROVER and your LED in the circuit starts Blink.



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 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.

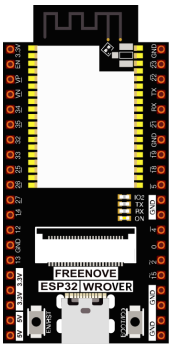
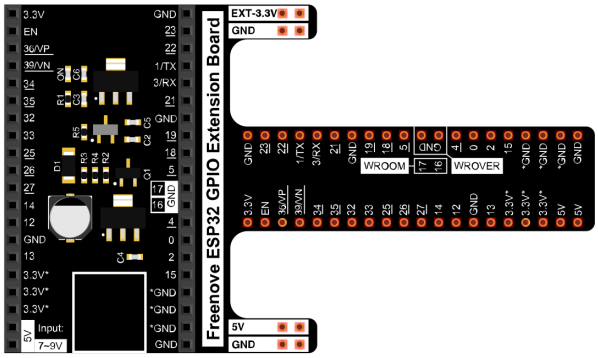
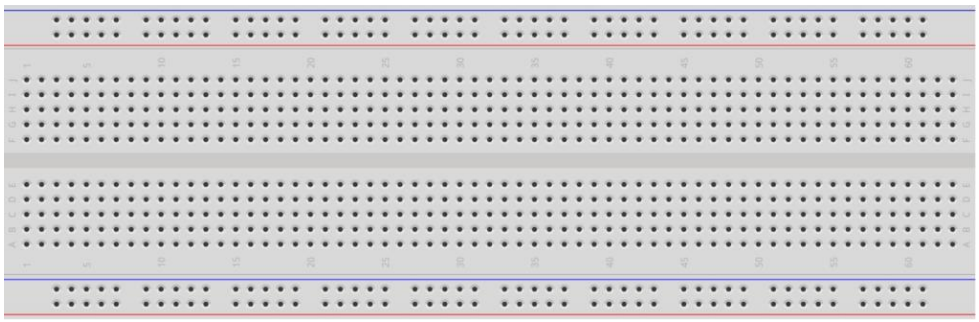


Next, we will build a simple control system to control a LED through a push button switch.

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.

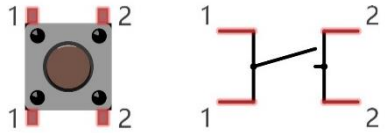
Component List

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

Component knowledge

Push button

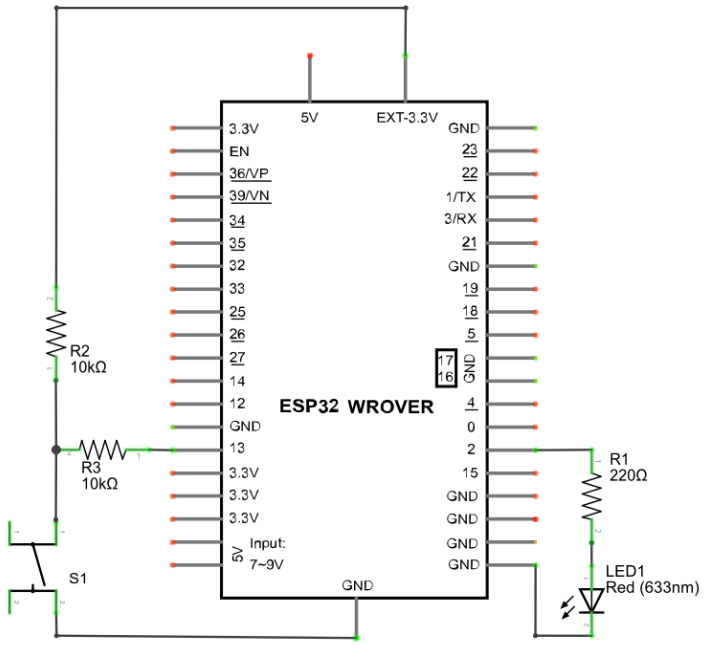
This type of push button switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



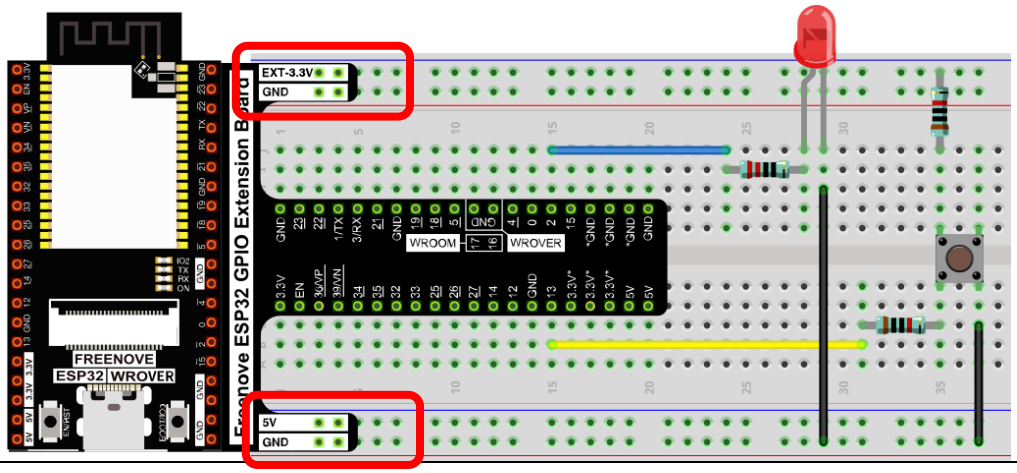
When the button on the switch is pressed, the circuit is completed (your project is powered ON).

Circuit

Schematic diagram



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



Any concerns? ✉ support@freenove.com

Sketch

This project is designed for learning how to use push button switch to control a LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch. Upload following sketch:

Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketch_02.1_ButtonAndLed.

Sketch_02.1_ButtonAndLed

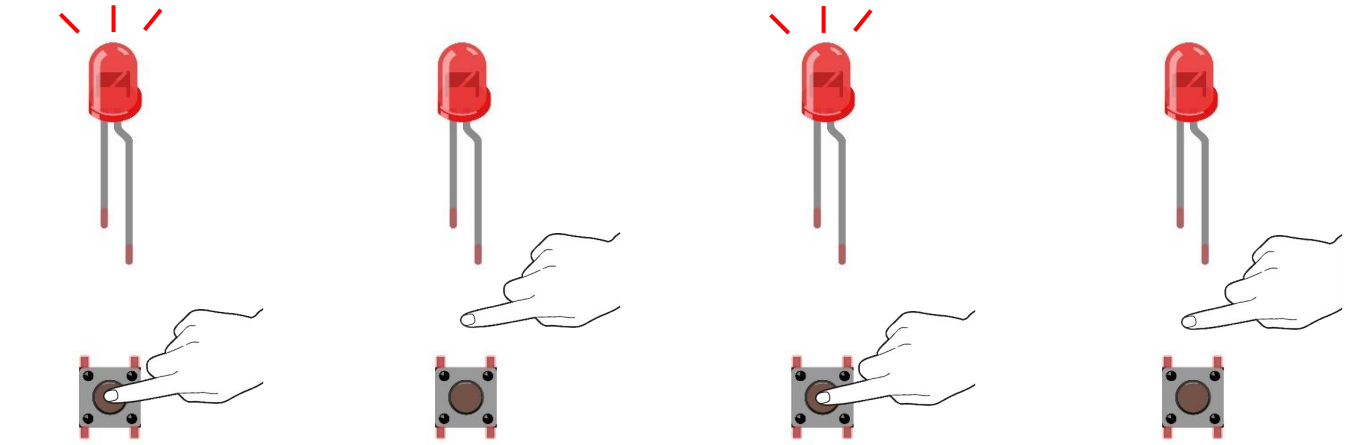
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_02.1_ButtonAndLed.ino

```
/**************************************************************************
  Filename    : ButtonAndLed
  Description : Control led by button.
  Auther      : www.freenove.com
  Modification: 2024/06/18
**************************************************************************/
#define PIN_LED    2
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

// the loop function runs over and over again forever
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    digitalWrite(PIN_LED,LOW);
  }else{
    digitalWrite(PIN_LED,HIGH);
  }
}
```

Download the code to ESP32-WROVER, then press the key, the LED turns ON, release the switch, the LED turns OFF.



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

The following is the program code:

```
#define PIN_LED    2
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

// the loop function runs over and over again forever
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    digitalWrite(PIN_LED,LOW);
  }else{
    digitalWrite(PIN_LED,HIGH);
  }
}
```

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

```
#define PIN_LED    2
#define PIN_BUTTON 13
```

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

```
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    digitalWrite(PIN_LED,LOW);
  }else{
    digitalWrite(PIN_LED,HIGH);
  }
}
```

Reference

int digitalRead (int pin);

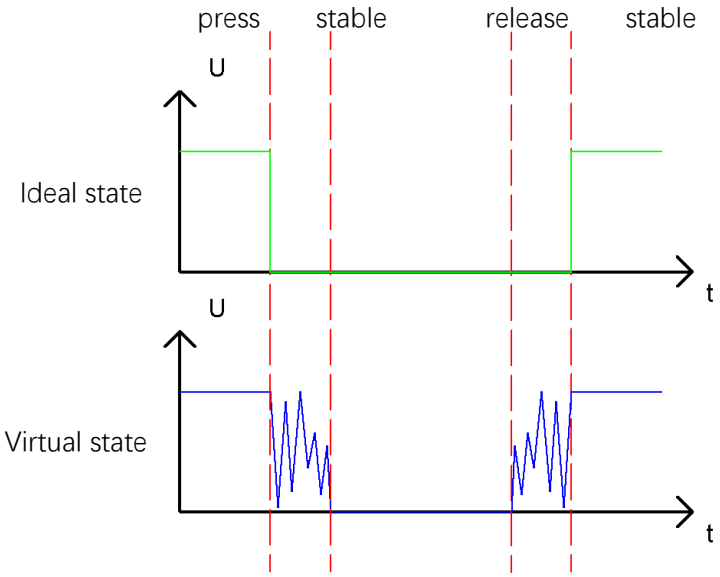
This function returns the value read at the given pin. It will be “HIGH” or “LOW”(1 or 0) depending on the logic level at the pin.

Project 2.2 MINI table lamp

We will also use a push button switch, LED and ESP32 to make a MINI table lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.
First, let us learn something about the push button switch.

Debounce for Push Button

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.

Sketch

Sketch_02.2_Tablelamp

Sketch_02.2_TableLamp | Arduino 1.8.10

File Edit Sketch Tools Help

Sketch_02.2_TableLamp

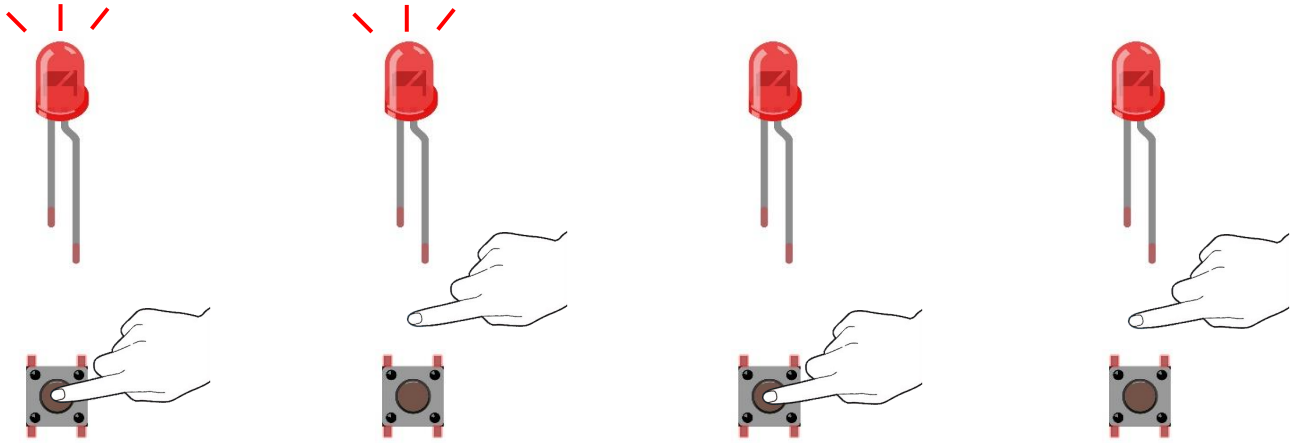
```
#define PIN_LED    2
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

// the loop function runs over and over again forever
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    delay(20);
    if (digitalRead(PIN_BUTTON) == LOW) {
      reverseGPIO(PIN_LED);
    }
    while (digitalRead(PIN_BUTTON) == LOW);
    delay(20);
    while (digitalRead(PIN_BUTTON) == LOW);
  }
}

void reverseGPIO(int pin) {
  digitalWrite(pin, !digitalRead(pin));
}
```

Done uploading.

```
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 6143.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

7 ESP32 Wrover Module on COM9

Download the code to the ESP32-WROVER, press the button, the LED turns ON, and press the button again, the LED turns OFF.



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

Any concerns? ✉ support@freenove.com

The following is the program code:

```
#define PIN_LED    2
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

// the loop function runs over and over again forever
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    delay(20);
    if (digitalRead(PIN_BUTTON) == LOW) {
      reverseGPIO(PIN_LED);
    }
    while (digitalRead(PIN_BUTTON) == LOW);
    delay(20);
    while (digitalRead(PIN_BUTTON) == LOW);
  }
}

void reverseGPIO(int pin) {
  digitalWrite(pin, ! digitalRead(pin));
}
```

When judging the push button state, if it is detected as "pressed down", wait for a certain time to detect again to eliminate the effect of bounce. When confirmed, flip the LED on and off. Then it starts to wait for the pressed button to be released, and waits for a certain time to eliminate the effect of bounce after it is released.

```
  if (digitalRead(PIN_BUTTON) == LOW) {
    delay(20);
    if (digitalRead(PIN_BUTTON) == LOW) {
      reverseGPIO(PIN_LED);
    }
    while (digitalRead(PIN_BUTTON) == LOW);
    delay(20);
    while (digitalRead(PIN_BUTTON) == LOW);
  }
```

The subfunction reverseGPIO() means reading the state value of the specified pin, taking the value back and writing it to the pin again to achieve the function of flipping the output state of the pin.

```
void reverseGPIO(int pin) {
  digitalWrite(pin, ! digitalRead(pin));
}
```

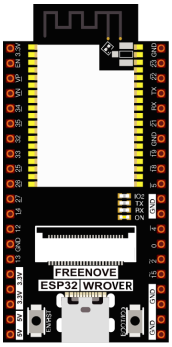
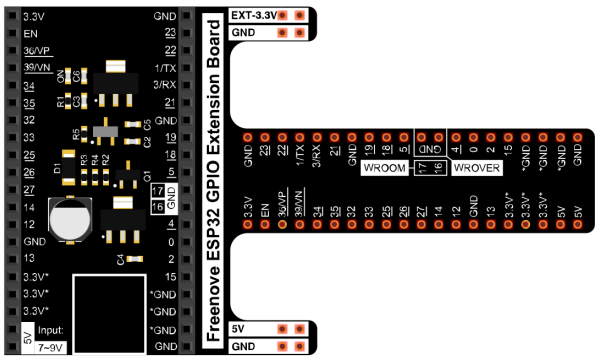
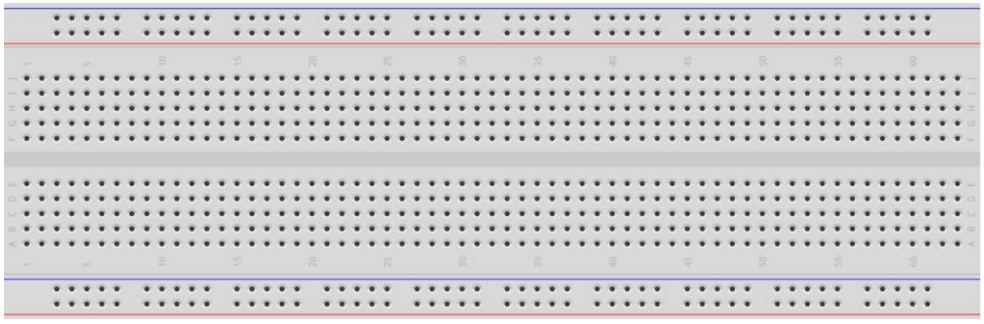
Chapter 3 LED Bar

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

Project 3.1 Flowing Light

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

Component List

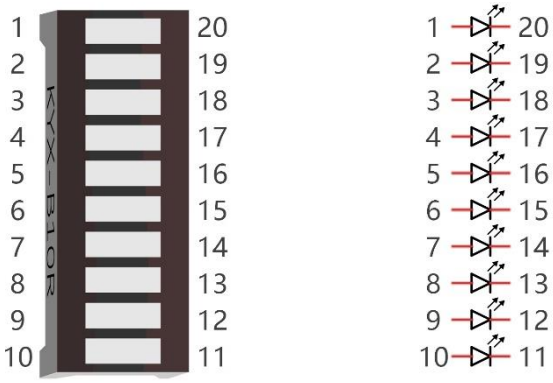
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x10 	LED bar graph x1 	Resistor 220Ω x10 

Component knowledge

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

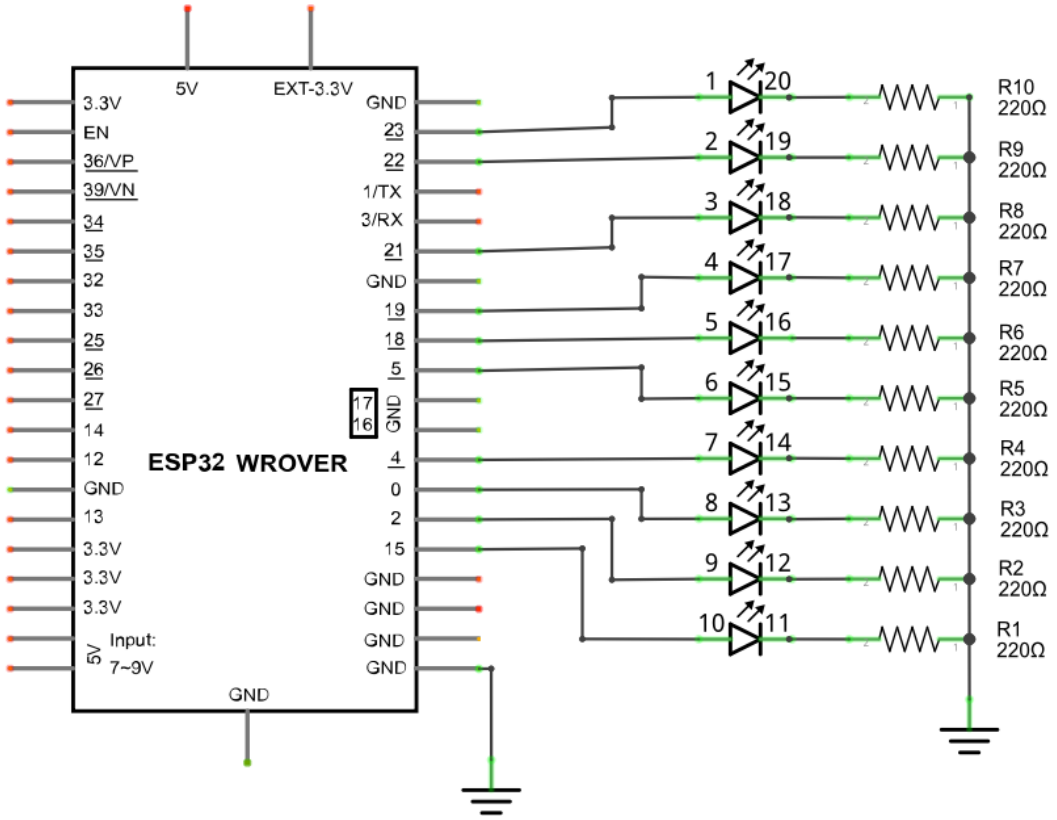
LED bar

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.

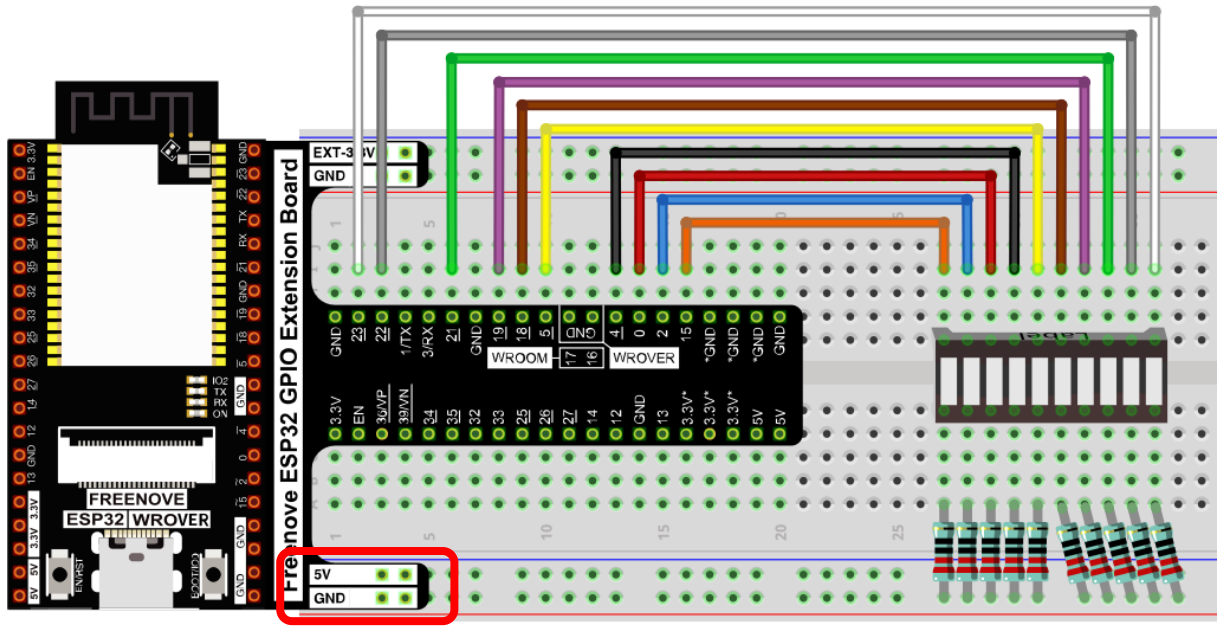


Circuit

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.

Any concerns? ✉ support@freenove.com

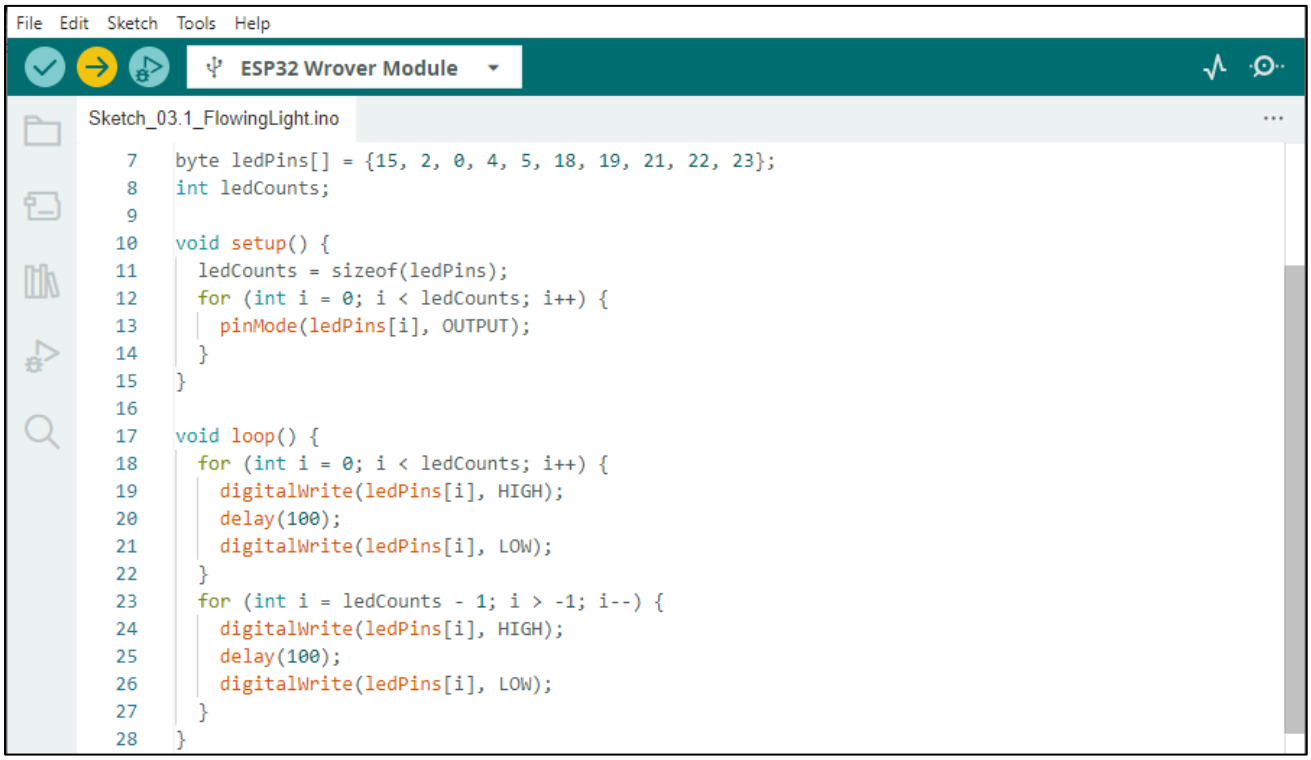
Sketch

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.

Upload following sketch:

Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketch_03.1_FlowingLight.

Sketch_03.1_FlowingLight




```
byte ledPins[] = {15, 2, 0, 4, 5, 18, 19, 21, 22, 23};
int ledCounts;

void setup() {
  ledCounts = sizeof(ledPins);
  for (int i = 0; i < ledCounts; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < ledCounts; i++) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
  for (int i = ledCounts - 1; i > -1; i--) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
}
```

Download the code to ESP32-WROVER and LED bar graph will light up from left to right and from right to left.

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

The following is the program code:

```
byte ledPins[] = {15, 2, 0, 4, 5, 18, 19, 21, 22, 23};
int ledCounts;

void setup() {
  ledCounts = sizeof(ledPins);
  for (int i = 0; i < ledCounts; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < ledCounts; i++) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
  for (int i = ledCounts - 1; i > -1; i--) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
}
```

Use an array to define 10 GPIO ports connected to LED bar graph for easier operation.

```
byte ledPins[] = {15, 2, 0, 4, 5, 18, 19, 21, 22, 23};
```

In setup(), use sizeof() to get the number of array, which is the number of LEDs, then configure the GPIO port to output mode.

```
  ledCounts = sizeof(ledPins);
  for (int i = 0; i < ledCounts; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
```

Then, in loop(), use two “for” loop to realize flowing water light from left to right and from right to left.

```
  for (int i = 0; i < ledCounts; i++) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
  for (int i = ledCounts - 1; i > -1; i--) {
    digitalWrite(ledPins[i], HIGH);
    delay(100);
    digitalWrite(ledPins[i], LOW);
  }
```

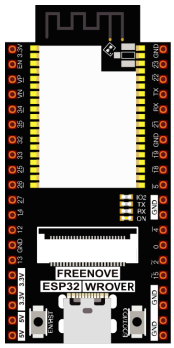
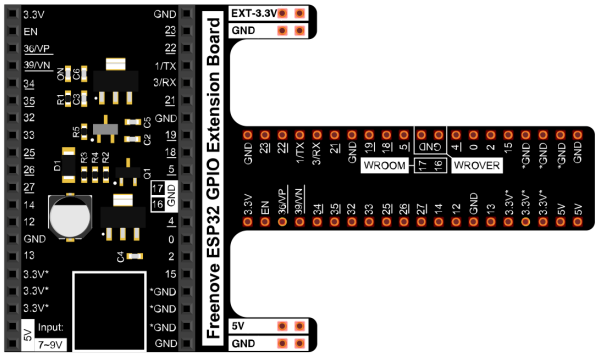
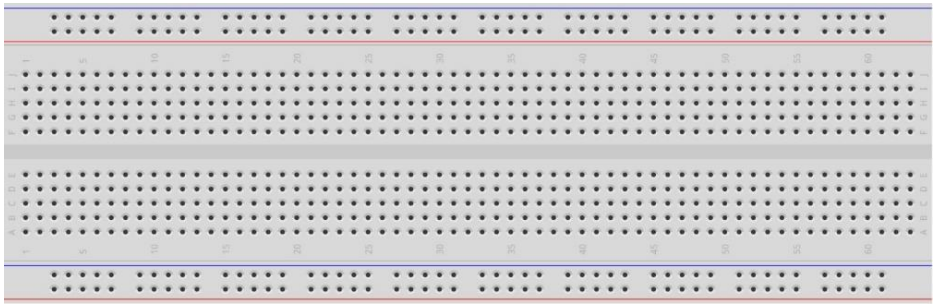
Chapter 4 Analog & PWM

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.

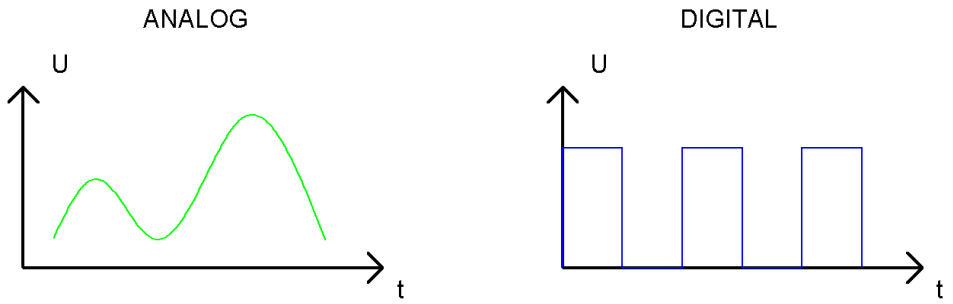
Component List

<table>
<tr><td>ESP32-WROVER x1
  </td><td colspan="2">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.



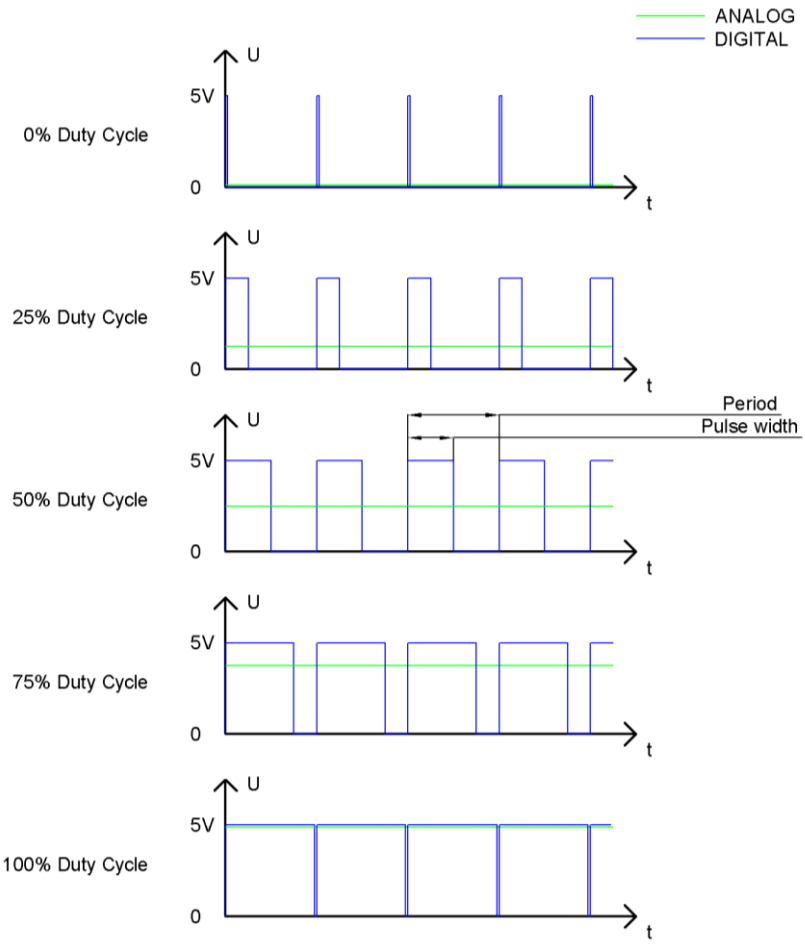
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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 and PWM

On ESP32, the LEDC(PWM) controller has 16 separate channels, each of which can independently control frequency, duty cycle, and even accuracy. Unlike traditional PWM pins, the PWM output pins of ESP32 are configurable, 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.

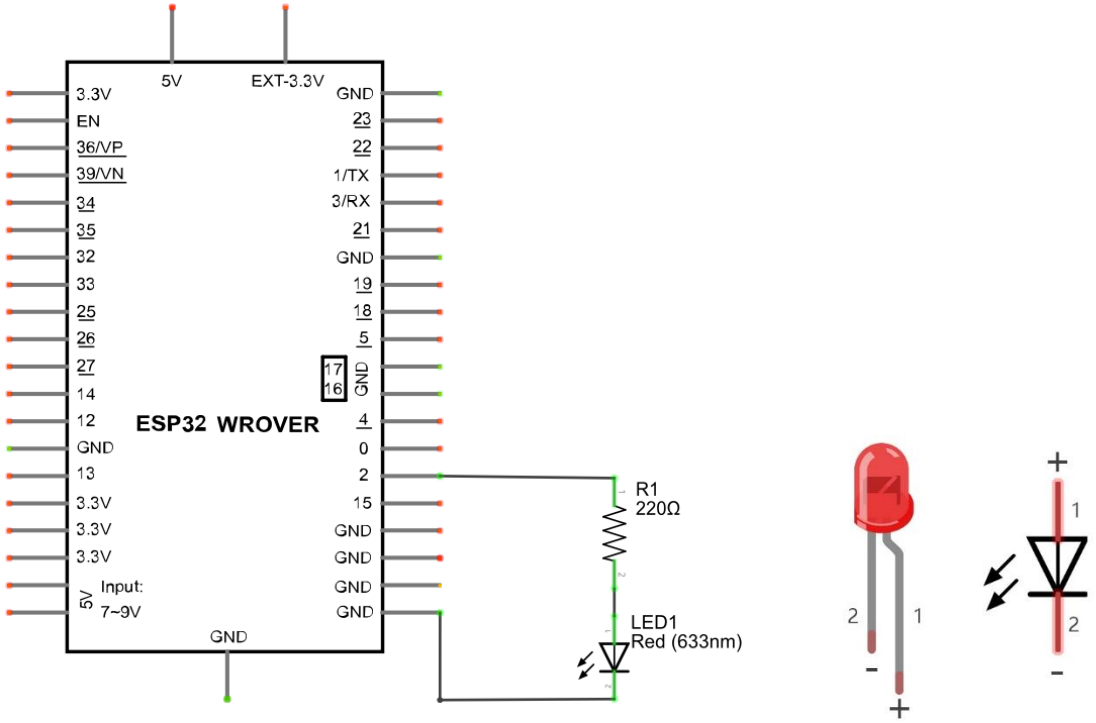
$$Freq_{max} = \frac{80{,}000{,}000}{1 \ll bit}$$

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.)

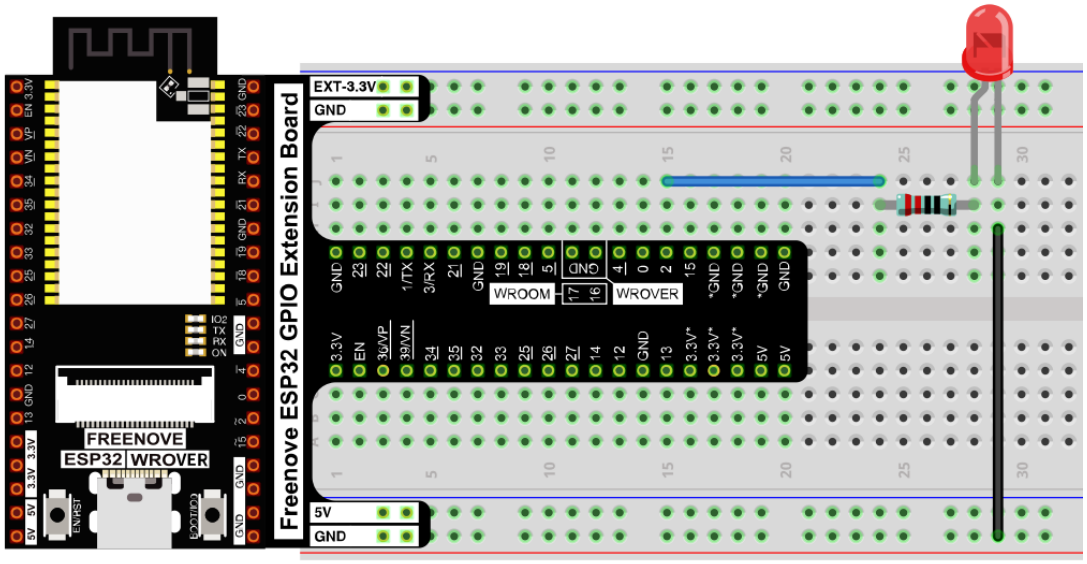
Circuit

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

Schematic diagram



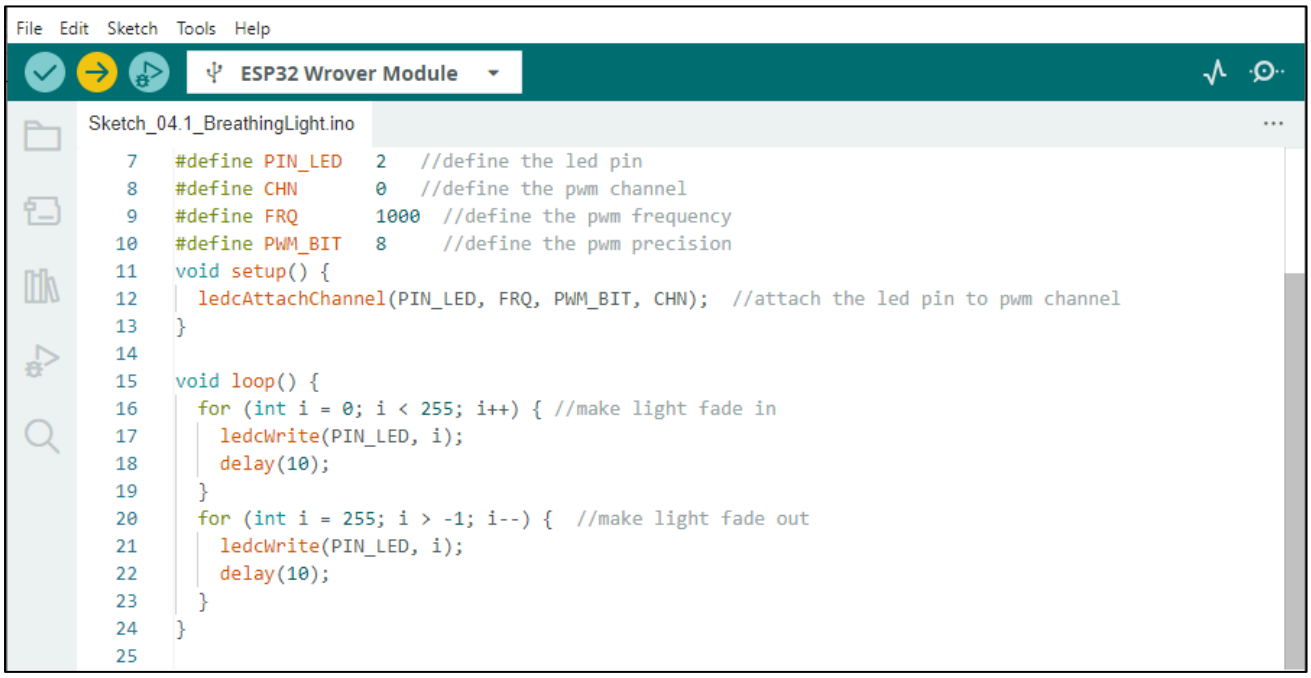
Hardware connection. **If you need any support, please contact us via: support@freenove.com**



Sketch

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.

Sketch_04.1_BreathingLight



Download the code to ESP32-WROVER, and you'll see that LED is turned from on to off and then from off to on gradually like breathing.

The following is the program code:

```
#define PIN_LED   2     //define the led pin
#define CHN       0     //define the pwm channel
#define FRQ       1000  //define the pwm frequency
#define PWM_BIT   8     //define the pwm precision
void setup() {
  ledcAttachChannal(PIN_LED, FRQ, PWM_BIT, CHN);  //attach the led pin to pwm channel
}

void loop() {
  for (int i = 0; i < 255; i++) { //make light fade in
    ledcWrite(PIN_LED, i);
    delay(10);
  }
  for (int i = 255; i > -1; i--) {  //make light fade out
    ledcWrite(PIN_LED, i);
    delay(10);
  }
}
```

The PWM pin output mode of ESP32 is not the same as the traditional controller. It controls each parameter of PWM by controlling the PWM channel. Any number of GPIO can be connected with the PWM channel to output PWM. In ledcAttachChannal(), you first configure a PWM channel and set the frequency and precision. Then the GPIO is associated with the PWM channel.

```
ledcAttachChannal(PIN_LED, FRQ, PWM_BIT, CHN);  //attach the led pin to pwm channel
```

In the loop(), There are two “for” loops. The first makes the ledPin output PWM from 0% to 100% and the second makes the ledPin output PWM from 100% to 0%. This allows the LED to gradually light and extinguish。

```
  for (int i = 0; i < 255; i++) {   //make light fade in
    ledcWrite(PIN_LED, i);
    delay(10);
  }
  for (int i = 255; i > -1; i--) {  //make light fade out
    ledcWrite(PIN_LED, i);
    delay(10);
  }
```

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

`bool ledcAttachChannel(uint8_t pin, uint32_t freq, uint8_t resolution, uint8_t channel)`
Set the pin, frequency and accuracy of a PWM channel. **Parameters** **pip**: The pin of PWM. **freq**: Frequency value of PWM. **resolution**: Pwm precision control bit. **channal**: channel index. Value range :0-15
`void ledcAttach(uint8_t pin, uint32_t freq, uint8_t resolution);` `void ledcDetach(uint8_t pin);`
Bind/unbind a GPIO to a PWM channel.
`void ledcWrite(uint8_t pin, uint32_t duty);`
Writes the pulse width value to a PWM channel.

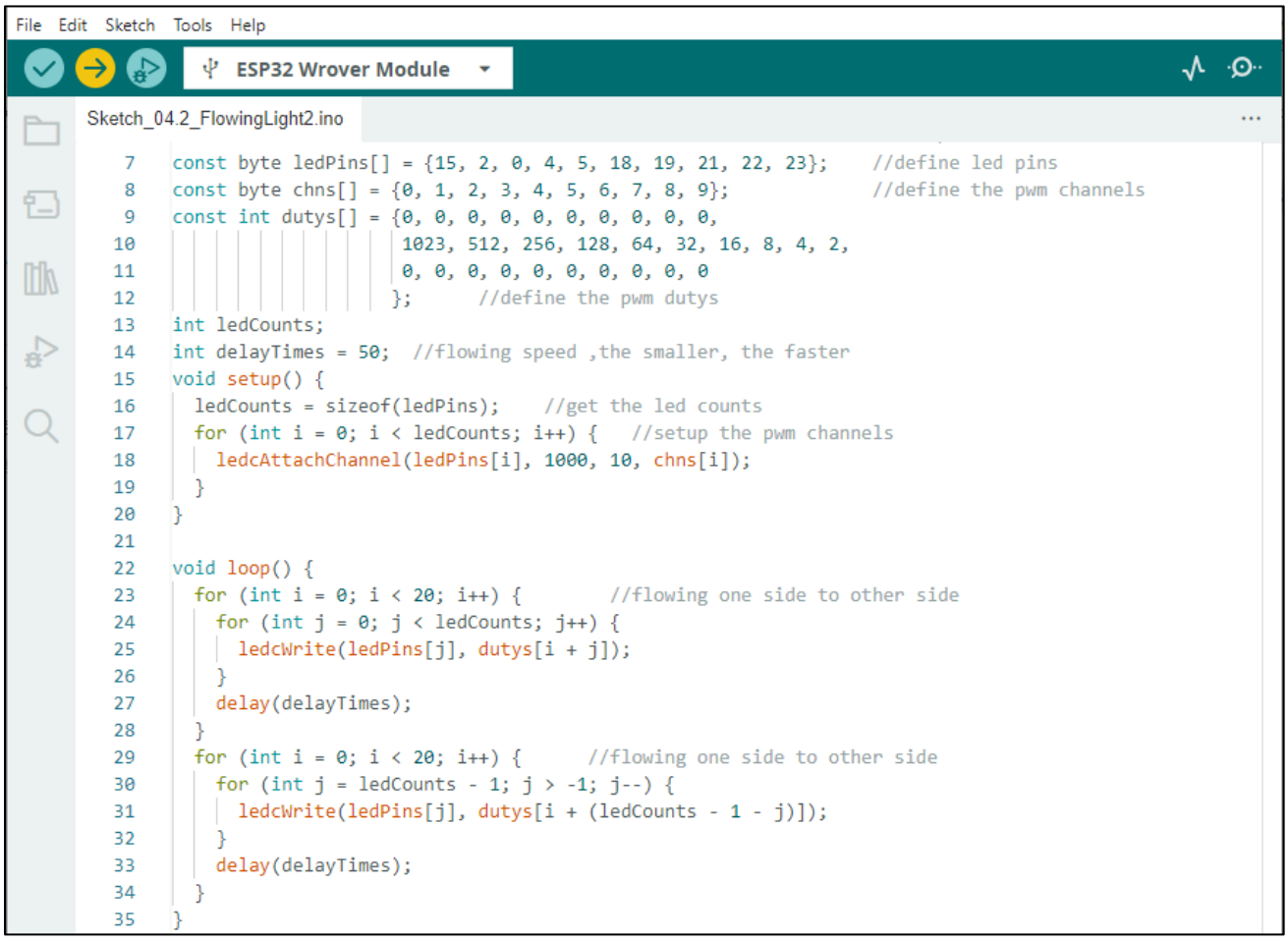
Project 4.2 Meteor Flowing Light

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.

Sketch

Meteor flowing light will be implemented with PWM.

Sketch_04.2_FlowingLight2



Download the code to ESP32-WROVER, and LED bar graph will gradually light up and out from left to right, then light up and out from right to left.

The following is the program code:

```
1  const byte ledPins[] = {15, 2, 0, 4, 5, 18, 19, 21, 22, 23};    //define led pins
2  const byte chns[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};             //define the pwm channels
3  const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4                       1023, 512, 256, 128, 64, 32, 16, 8, 4, 2,
5                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0
6                      };      //define the pwm dutys
7  int ledCounts;               //led counts
```

```
int delayTimes = 50;        //flowing speed ,the smaller, the faster
void setup() {
  ledCounts = sizeof(ledPins);    //get the led counts
  for (int i = 0; i < ledCounts; i++) {   //setup the pwm channels
    ledcAttachChannal(ledPins[i], 1000, 10, chns[i]);
  }
}

void loop() {
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      ledcWrite(ledPins[j], dutys[i + j]);
    }
    delay(delayTimes);
  }
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = ledCounts - 1; j > -1; j--) {
      ledcWrite(ledPins[j], dutys[i + (ledCounts - 1 - j)]);
    }
    delay(delayTimes);
  }
}
```

First we defined 10 GPIO, 10 PWM channels, and 30 pulse width values.

```
const byte ledPins[] = {15, 2, 0, 4, 5, 18, 19, 21, 22, 23};    //define led pins
const byte chns[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};             //define the pwm channels
const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     1023, 512, 256, 128, 64, 32, 16, 8, 4, 2,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                    };                                           //define the pwm dutys
```

In setup(), set the frequency of 10 PWM channels to 1000Hz, the accuracy to 10bits, and the maximum pulse width to 1023. Attach GPIO to these PWM channels.

```
  for (int i = 0; i < ledCounts; i++) {   //setup the pwm channels
    ledcWrite(ledPins[j], dutys[i + j]);
  }
```

In loop(), a nested for loop is used to control the pulse width of the PWM, and LED bar graph moves one grid after each 1 is added in the first for loop, gradually changing according to the values in the array duties. As shown in the table below, the value of the second row is the value in the array duties, and the 10 green squares in each row below represent the 10 LEDs on the LED bar graph. Every 1 is added to I, 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	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
di	0	0	0	0	0	0	0	0	0	1023	512	256	128	64	32	16	8	4	2	0	0	0	0	0	0	0	0	0	0
0																													
1																													
2																													
3																													
...																													
18																													
19																													
20																													

In the code, two nested for loops are used to achieve this effect.

```
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      ledcWrite(ledPins[j], dutys[i + j]);
    }
    delay(delayTimes);
  }
  for (int i = 0; i < 20; i++) {       //flowing from one side to the other
    for (int j = ledCounts - 1; j > -1; j--) {
      ledcWrite(ledPins[j], dutys[i + (ledCounts - 1 - j)]);
    }
    delay(delayTimes);
  }
```

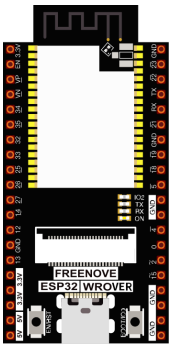
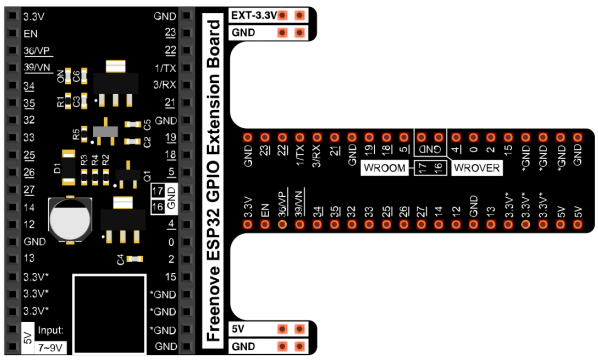
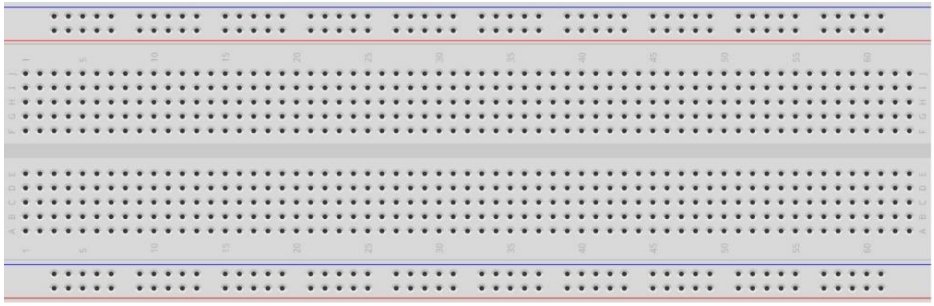
Chapter 5 RGB LED

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

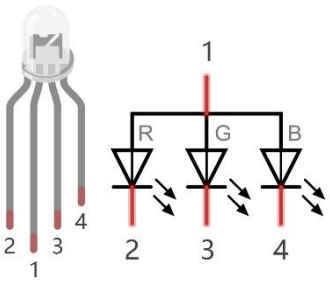
In this project, we will make a multicolored LED. And we can control RGB LED to switch different colors automatically.

Component List

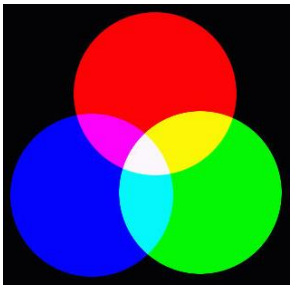
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
RGBLED x1 	Resistor 220Ω x3 	Jumper M/M x4 

Related knowledge

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



Red, green, and blue 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.

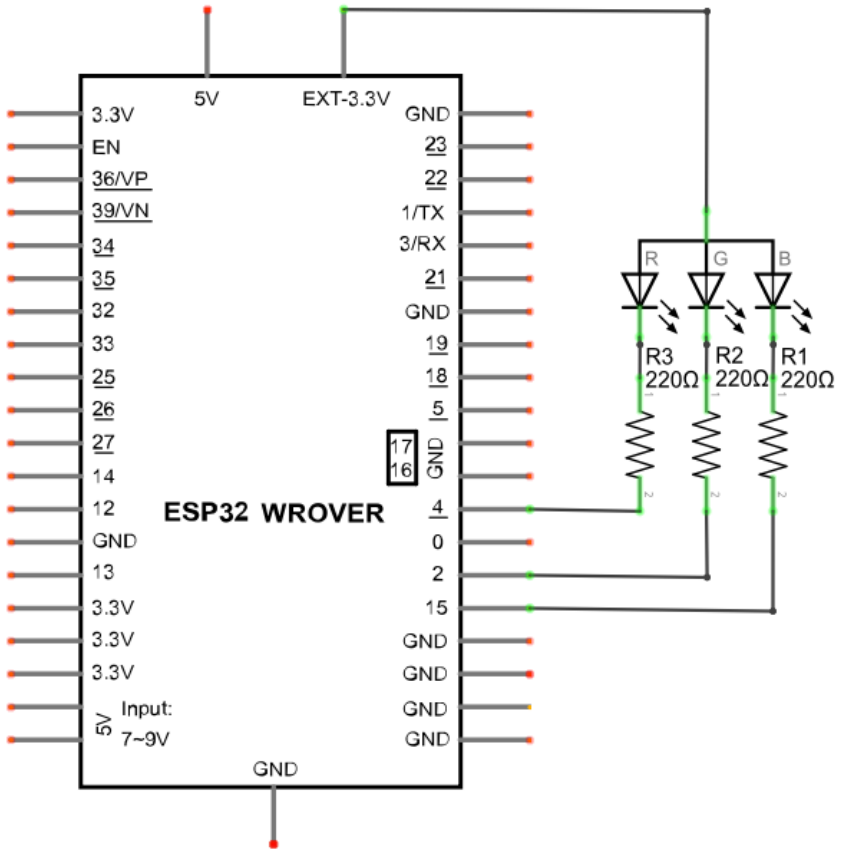


RGB

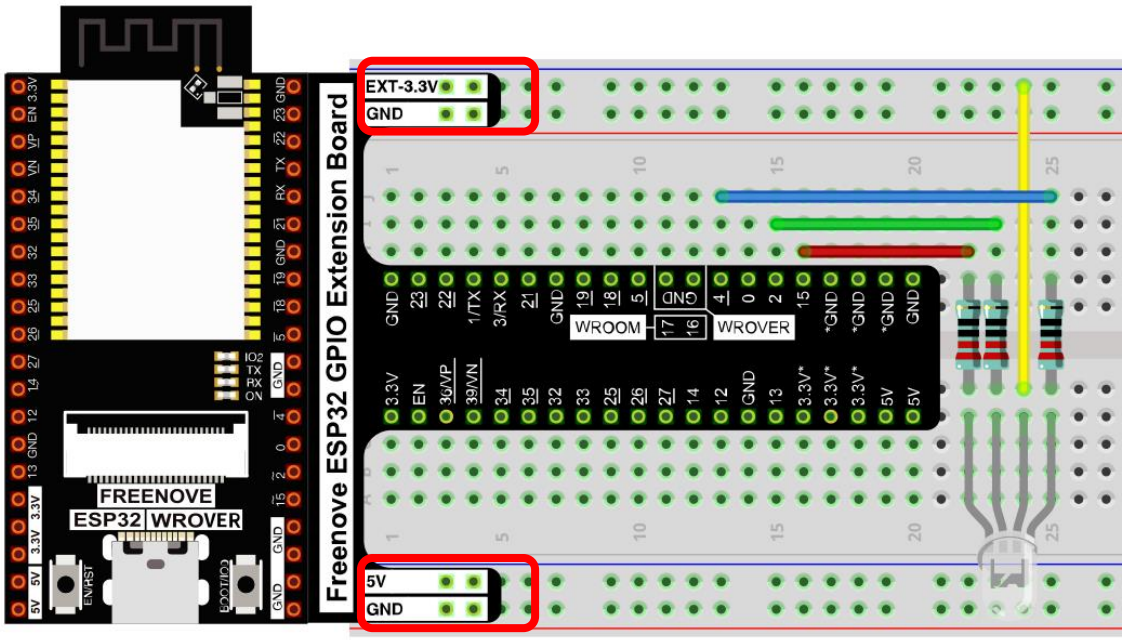
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

Circuit

Schematic diagram



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



Sketch

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

Sketch_05.1_ColorfulLight

Sketch_05.1_RandomColorLight | Arduino IDE 2.3.2

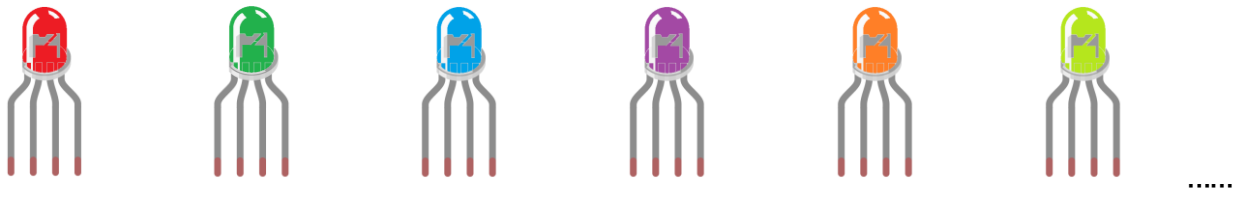
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_05.1_RandomColorLight.ino

```
/**********************************************************************
  Filename    : ColorfulLight
  Description : Use RGBLED to show random color.
  Auther      : www.freenove.com
  Modification: 2024/06/18
**********************************************************************/
const byte ledPins[] = {4, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;
void setup() {
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcAttachChannel(ledPins[i], 1000, 8, chns[i]);
  }
}

void loop() {
  red = random(0, 256);
  green = random(0, 256);
  blue = random(0, 256);
  setColor(red, green, blue);
  delay(200);
}

void setColor(byte r, byte g, byte b) {
  ledcWrite(ledPins[0], 255 - r); //Common anode LED, low level to turn on the led.
  ledcWrite(ledPins[1], 255 - g);
  ledcWrite(ledPins[2], 255 - b);
}
```

With the code downloaded to ESP32-WROVER, RGB LED begins to display random colors.



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

The following is the program code:

```
const byte ledPins[] = {4, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;
void setup() {
  for (int i = 0; i < 3; i++) {       //setup the pwm channels,1KHz,8bit
    ledcAttachChannel(ledPins[i], 1000, 8, chns[i]);
  }
}

void loop() {
  red = random(0, 256);
  green = random(0, 256);
  blue = random(0, 256);
  setColor(red, green, blue);
  delay(200);
}

void setColor(byte r, byte g, byte b) {
  ledcWrite(chns[0], 255 - r); //Common anode LED, low level to turn on the led.
  ledcWrite(chns[1], 255 - g);
  ledcWrite(chns[2], 255 - b);
}
```

Define the PWM channel and associate it with the pin connected to RGB LED, and define the variable to hold the color value and initialize it in setup().

```
const byte ledPins[] = {4, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;
void setup() {
  for (int i = 0; i < 3; i++) {       //setup the pwm channels,1KHz,8bit
    ledcAttachChannel(ledPins[i], 1000, 8, chns[i]);
  }
}
```

In setColor(), this function controls the output color of RGB LED by the given color value. Because the circuit uses a common anode, the LED lights up when the GPIO outputs low power. Therefore, in PWM, low level is the active level, so 255 minus the given value is necessary.

```
void setColor(byte r, byte g, byte b) {
  ledcWrite(ledPins[0], 255 - r); //Common anode LED, low level to turn on the led.
  ledcWrite(ledPins[1], 255 - g);
  ledcWrite(ledPins[2], 255 - b);
}
```

In loop(), get three random Numbers and set them as color values.

```
12    red = random(0, 256);
13    green = random(0, 256);
14    blue = random(0, 256);
15    setColor(red, green, blue);
16    delay(200);
```

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

long random(min, max);
This function will return a random number(min --- max-1).

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 RGB LED will change colors along the model.

Sketch_05.2_SoftColorfulLight

The following is the program code:

```
const byte ledPins[] = {4, 2, 15};    //define led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels

void setup() {
  for (int i = 0; i < 3; i++) {       //setup the pwm channels
    ledcAttachChannel(ledPins[i], 1000, 8, chns[i]);
  }
}

void loop() {
  for (int i = 0; i < 256; i++) {
    setColor(wheel(i));
    delay(20);
  }
}

void setColor(long rgb) {
  ledcWrite(ledPins[0], 255 - (rgb >> 16) & 0xFF);
  ledcWrite(ledPins[1], 255 - (rgb >> 8) & 0xFF);
  ledcWrite(ledPins[2], 255 - (rgb >> 0) & 0xFF);
}

long wheel(int pos) {
  long WheelPos = pos % 0xff;
  if (WheelPos < 85) {
    return ((255 - WheelPos * 3) << 16) | ((WheelPos * 3) << 8);
  } else if (WheelPos < 170) {
```

```
28      WheelPos -= 85;
29      return (((255 - WheelPos * 3) << 8) | (WheelPos * 3));
30    } else {
31      WheelPos -= 170;
32      return ((WheelPos * 3) << 16 | (255 - WheelPos * 3));
33    }
34  }
```

In setColor(), a variable represents the value of RGB, and a hexadecimal representation of color is a common representation, such as 0xAABBCC, where AA represents the red value, BB represents the green value, and CC represents the blue value. The use of a variable can make the transmission of parameters more convenient, in the split, only a simple operation can take out the value of each color channel

```
18  void setColor(long rgb) {
19    ledcWrite(ledPins[0], 255 - (rgb >> 16) & 0xFF);
20    ledcWrite(ledPins[1], 255 - (rgb >> 8) & 0xFF);
21    ledcWrite(ledPins[2], 255 - (rgb >> 0) & 0xFF);
22  }
```

The wheel() function is the color selection method for the color model introduced earlier. The **pos** parameter ranges from 0 to 255 and outputs a color value in hexadecimal.

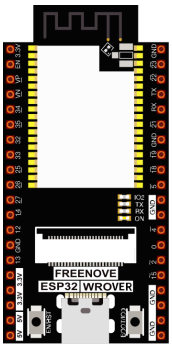
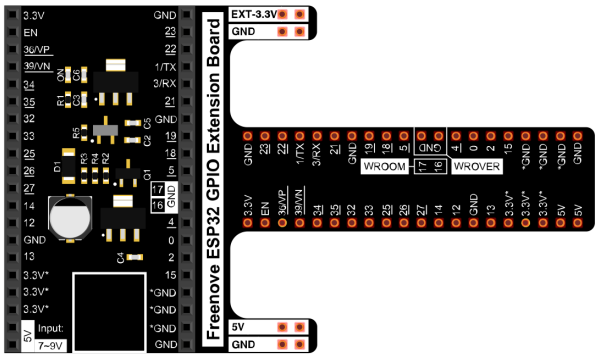
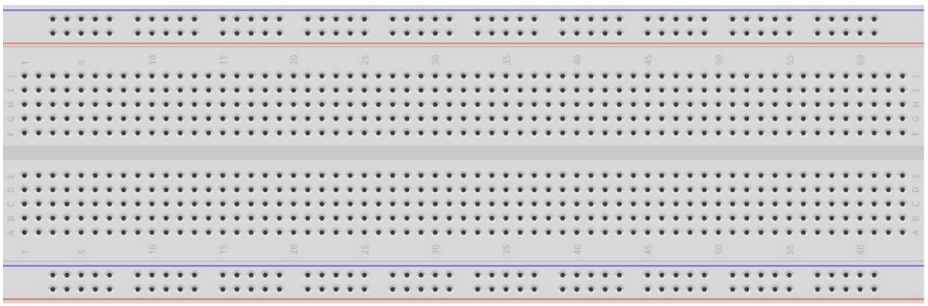
Chapter 6 Buzzer

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

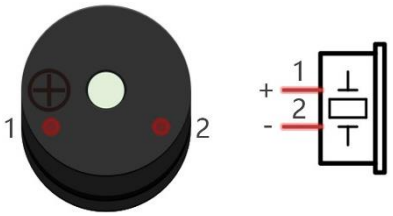
<table>
<tr><td colspan="3">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td colspan="5">Jumper M/M x6
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor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

Component knowledge

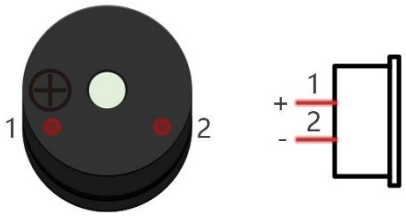
Buzzer

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

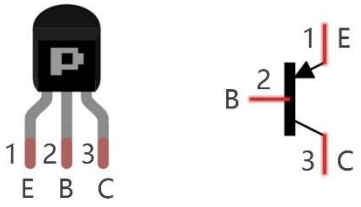
Passive buzzer

Transistor

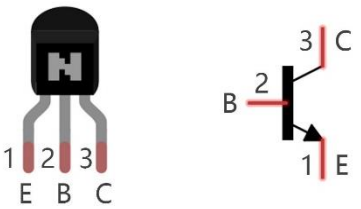
Because the buzzer requires such large current that GPIO of ESP32 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

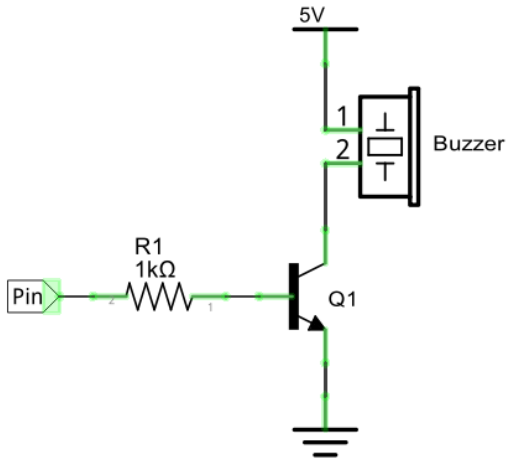
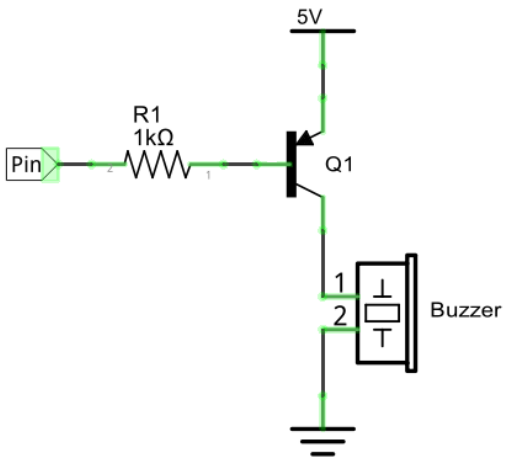


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

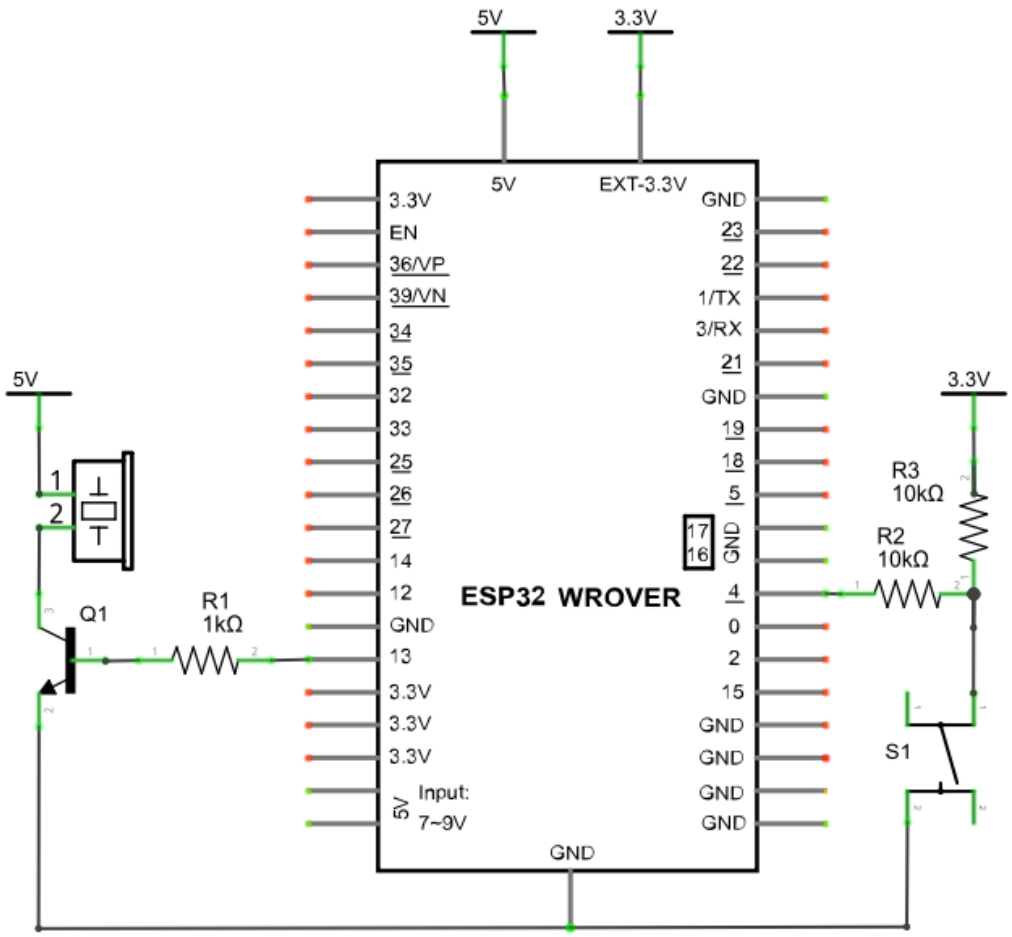
When use NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

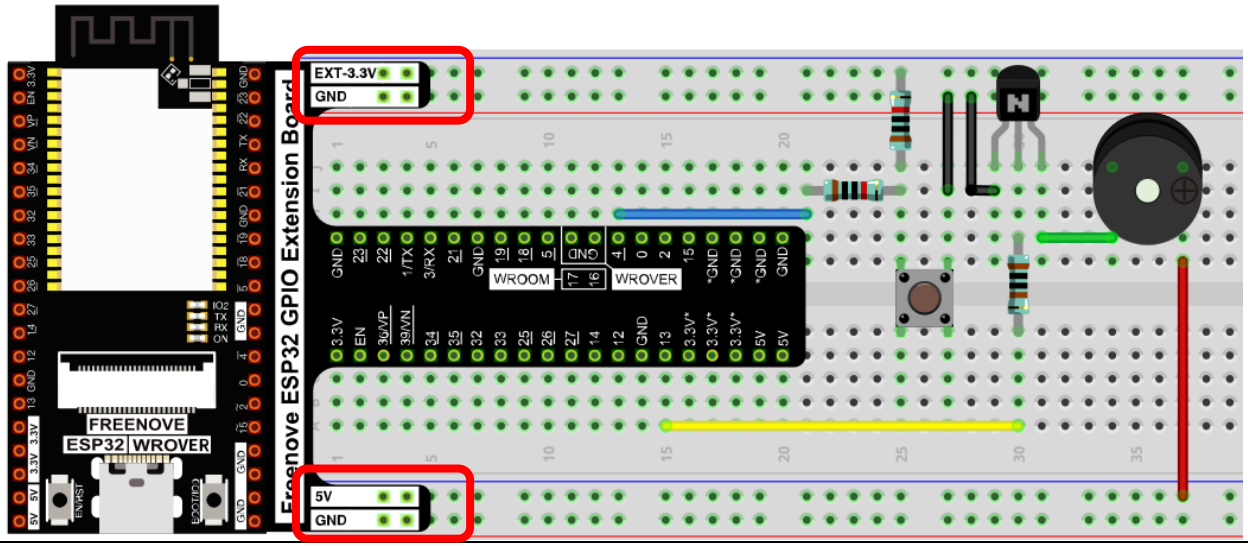
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

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.

Sketch

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 a LED ON and OFF.

Sketch_06.1_Doorbell

Sketch_07.1_Doorbell | Arduino 1.8.12

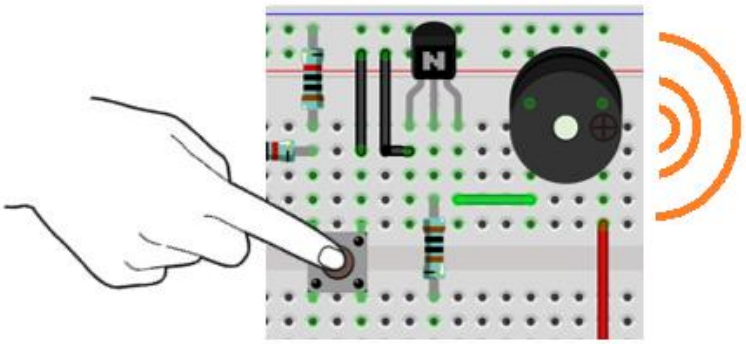
File Edit Sketch Tools Help

Sketch_07.1_Doorbell

```
/*****************************************************************************
  Filename    : Doorbell.c
  Description : Control active buzzer by button.
  Auther      : www.freenove.com
  Modification: 2020-1-6
*****************************************************************************/
#define PIN_BUZZER 13
#define PIN_BUTTON 4

void setup() {
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    digitalWrite(PIN_BUZZER,HIGH);
  }else{
    digitalWrite(PIN_BUZZER,LOW);
  }
}
```

19 ESP32 Wrover Module, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), DOUT, 80MHz, 921600, None on COM4

Download the code to ESP32-WROVER, 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:

```
#define PIN_BUZZER 13
#define PIN_BUTTON 4

void setup() {
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    digitalWrite(PIN_BUZZER,HIGH);
  }else{
    digitalWrite(PIN_BUZZER,LOW);
  }
}
```

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

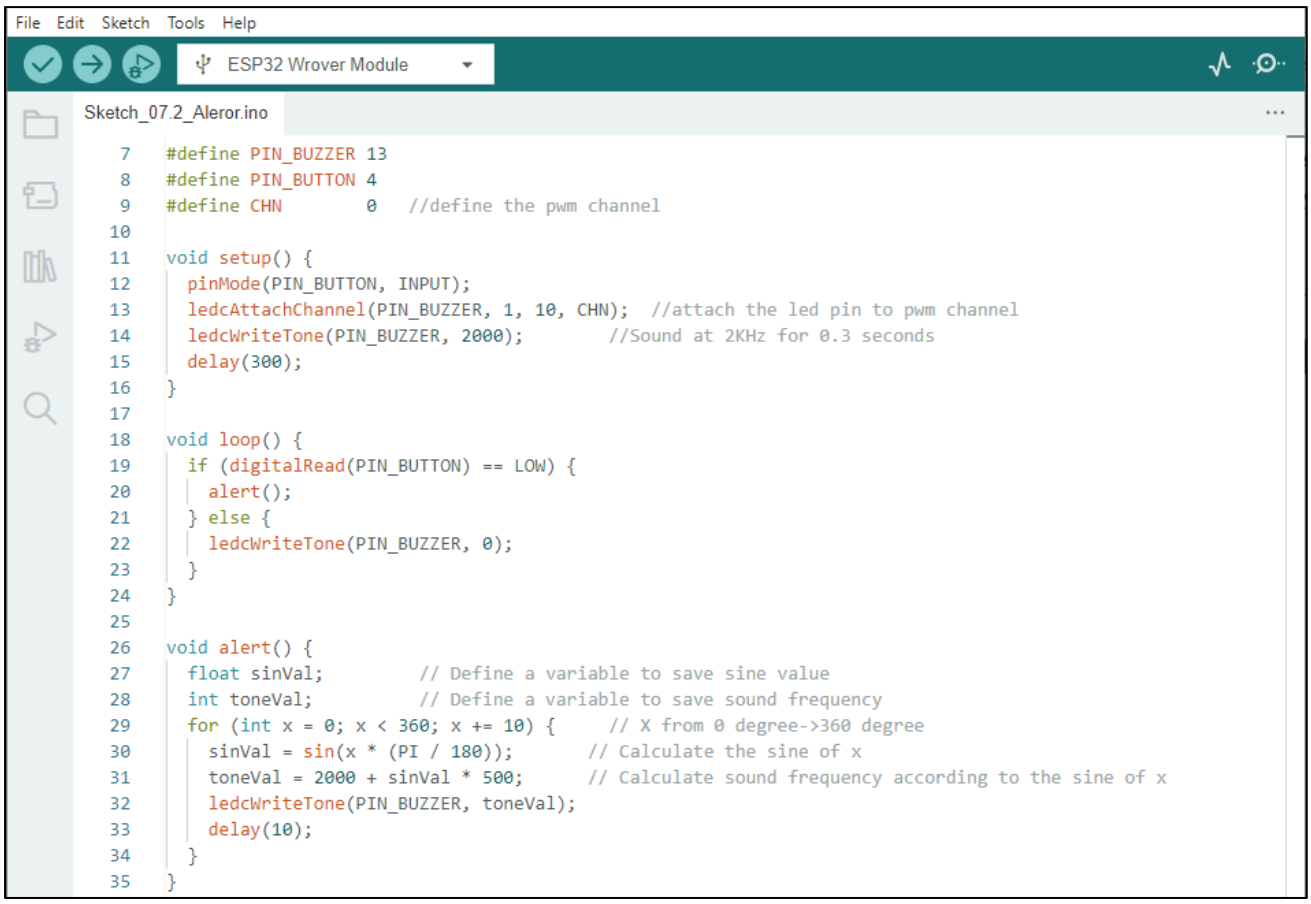
Project 6.2 Alertor

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.**

Sketch

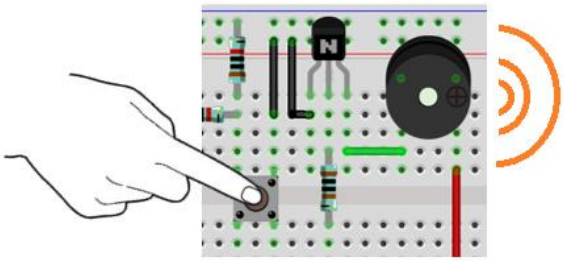
In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. It is logically the same as using button to control LED, but in the control method, passive buzzer requires PWM of certain frequency to sound.

Sketch_06.2_Alertor




```
#define PIN_BUZZER 13
#define PIN_BUTTON 4
#define CHN        0   //define the pwm channel

void setup() {
  pinMode(PIN_BUTTON, INPUT);
  ledcAttachChannel(PIN_BUZZER, 1, 10, CHN);  //attach the led pin to pwm channel
  ledcWriteTone(PIN_BUZZER, 2000);        //Sound at 2KHz for 0.3 seconds
  delay(300);
}

void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    alert();
  } else {
    ledcWriteTone(PIN_BUZZER, 0);
  }
}

void alert() {
  float sinVal;         // Define a variable to save sine value
  int toneVal;          // Define a variable to save sound frequency
  for (int x = 0; x < 360; x += 10) {     // X from 0 degree->360 degree
    sinVal = sin(x * (PI / 180));       // Calculate the sine of x
    toneVal = 2000 + sinVal * 500;      // Calculate sound frequency according to the sine of x
    ledcWriteTone(PIN_BUZZER, toneVal);
    delay(10);
  }
}
```

Download the code to ESP32-WROVER, press the button, then alarm sounds. And when the button is released, the alarm will stop sounding.



The following is the program code:

```
#define PIN_BUZZER 13
#define PIN_BUTTON 4
#define CHN        0   //define the pwm channel

void setup() {
  pinMode(PIN_BUTTON, INPUT);
  pinMode(PIN_BUZZER, OUTPUT);
  ledcAttachChannal(PIN_BUZZER, 0, 10, CHN);  //attach the led pin to pwm channel
  ledcWriteTone(PIN_BUZZER, 2000);        //Sound at 2KHz for 0.3 seconds
  delay(300);
}

void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    alert();
  } else {
    ledcWriteTone(PIN_BUZZER, 0);
  }
}

void alert() {
  float sinVal;         // Define a variable to save sine value
  int toneVal;          // Define a variable to save sound frequency
  for (int x = 0; x < 360; x += 10) {     // X from 0 degree->360 degree
    sinVal = sin(x * (PI / 180));       // Calculate the sine of x
    toneVal = 2000 + sinVal * 500;      //Calculate sound frequency according to the sine of x
    ledcWriteTone(PIN_BUZZER, toneVal);
    delay(10);
  }
}
```

The code is the same as the active buzzer logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so you need to create a PWM channel through ledcAttachChannal(). Here ledcWriteTone() is designed to generating square wave with variable frequency and duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

```
8   ledcAttachChannal(PIN_BUZZER, 0, 10, CHN);  //attach the led pin to pwm channel
9   ledcWriteTone(PIN_BUZZER, 2000);         //Sound at 2KHz for 0.3 seconds
```

In the while cycle of main function, when the button is pressed, subfunction alert() will be called and the alertor will issue a warning sound. The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (here is 500) and plus the resonant frequency of buzzer.

```
21  void alert() {
22    float sinVal;         // Define a variable to save sine value
23    int toneVal;          // Define a variable to save sound frequency
24    for (int x = 0; x < 360; x += 10) {     // X from 0 degree->360 degree
25      sinVal = sin(x * (PI / 180));       // Calculate the sine of x
26      toneVal = 2000 + sinVal * 500;      //Calculate sound frequency according to the sine of x
27      ledcWriteTone(PIN_BUZZER, toneVal);
28      delay(10);
29    }
30  }
```

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

```
17  ledcWriteTone(PIN_BUZZER, 0);
```

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.

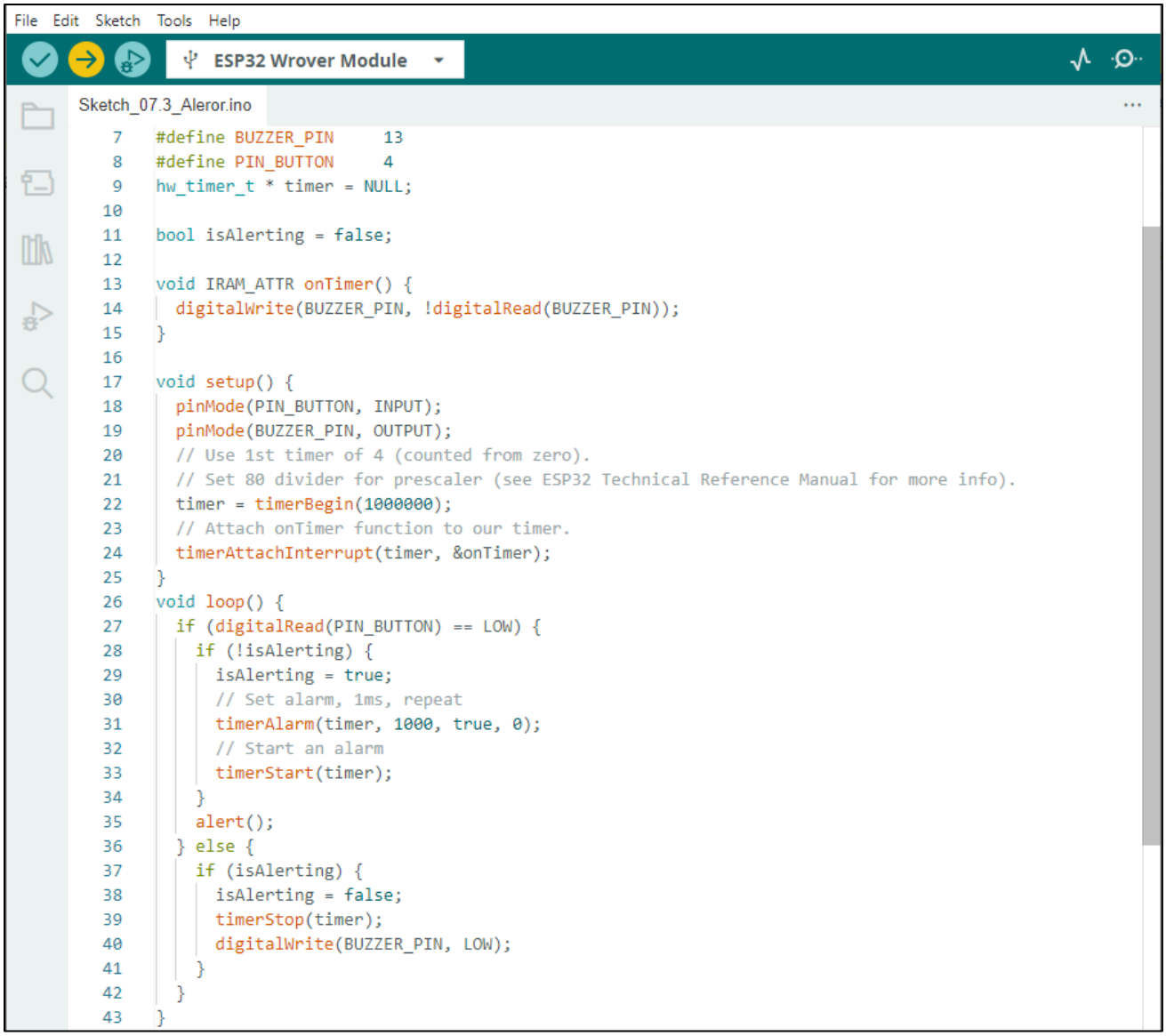
Project 6.3 Alertor (use timer)

Due to some bugs in the function ledcWriteTone(), this project uses timer to generate software PWM to control the buzzer. The circuit is exactly the same as the last project.

Sketch

The core of the code of this project is to create a timer to change the GPIO state to generate 50% pulse width PWM, and change the PWM frequency by changing the timing time of the timer.

Sketch_06.3_Alertor




```
#define BUZZER_PIN     13
#define PIN_BUTTON     4
hw_timer_t * timer = NULL;

bool isAlerting = false;

void IRAM_ATTR onTimer() {
  digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
}

void setup() {
  pinMode(PIN_BUTTON, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  // Use 1st timer of 4 (counted from zero).
  // Set 80 divider for prescaler (see ESP32 Technical Reference Manual for more info).
  timer = timerBegin(1000000);
  // Attach onTimer function to our timer.
  timerAttachInterrupt(timer, &onTimer);
}
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    if (!isAlerting) {
      isAlerting = true;
      // Set alarm, 1ms, repeat
      timerAlarm(timer, 1000, true, 0);
      // Start an alarm
      timerStart(timer);
    }
    alert();
  } else {
    if (isAlerting) {
      isAlerting = false;
      timerStop(timer);
      digitalWrite(BUZZER_PIN, LOW);
    }
  }
}
```

Download the code to ESP32-WROVER, press the button, then the alarm sounds. And when the button is released, the alarm will stop sounding.

The following is the program code:

```
#define BUZZER_PIN      13
#define PIN_BUTTON      4
hw_timer_t * timer = NULL;

bool isAlerting = false;

void IRAM_ATTR onTimer() {
  digitalWrite(BUZZER_PIN, ! digitalRead(BUZZER_PIN));
}

void setup() {
  pinMode(PIN_BUTTON, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  // Set the timer frequency to 1MHz.(see ESP32 Technical Reference Manual for more info).
  timer = timerBegin(1000000);
  // Attach onTimer function to our timer.
  timerAttachInterrupt(timer, &onTimer);
}
void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    if (!isAlerting) {
      isAlerting = true;
      // Set alarm, 1ms, repeat, auto-reload value.
      timerAlarmWrite(timer, 1000, true, 0);
      // Start an alarm
      timerAlarmEnable(timer);
    }
    alert();
  } else {
    if (isAlerting) {
      isAlerting = false;
      timerAlarmDisable(timer);
      digitalWrite(BUZZER_PIN, LOW);
    }
  }
}

void alert() {
  float sinVal;
  int toneVal;
  for (int x = 0; x < 360; x += 1) {
    sinVal = sin(x * (PI / 180));
    toneVal = 2000 + sinVal * 500;
```

```
44      timerAlarmWrite(timer, 500000 / toneVal, true, 0);
45      delay(1);
46    }
47  }
```

In the code, first define a timer variable, timer, and then create the timer in setup(), setting the function onTimer() that the timer will execute.

```
3   hw_timer_t * timer = NULL;
4
5   Set the timer frequency to 1MHz.(see ESP32 Technical Reference Manual for more info).
6   timer = timerBegin(1000000);
7   // Attach onTimer function to our timer.
8   timerAttachInterrupt(timer, &onTimer);
```

In the loop(), use the timerAlarmWrite() to set the timer time and use timerAlarmEnable() to start the timer. Using the flag bit isAlerting, the code to set and start the timer is executed only when the key is pressed.

```
22      if (! isAlerting) {
23        isAlerting = true;
24        // Set alarm, 1ms, repeat
25        timerAlarmWrite(timer, 1000, true, 0);
26        // Start an alarm
27        timerAlarmEnable(timer);
28      }
```

After the key is released, stop the timer and make the buzzer's GPIO output low.

```
31      if (isAlerting) {
32        isAlerting = false;
33        timerAlarmDisable(timer);
34        digitalWrite(BUZZER_PIN, LOW);
35      }
```

Reference

`typedef struct hw_timer_s hw_timer_t;`
Timer type, used to define a timer variable.
`hw_timer_t * timerBegin(uint32_t frequency);`
Initialize the timer. parameters frequency: Frequency of the timer.
`void timerAttachInterrupt(hw_timer_t *timer, void (*fn)(void));`
Bind the function to execute when the interrupt is generated for the timer.
`void timerAlarm(hw_timer_t *timer, uint64_t interruptAt, bool autoreload, uint64_t reload_count);`
Set the timer time.

`void timerStart(hw_timer_t *timer);`
`void timerStop(hw_timer_t *timer);`
`void timerRestart(hw_timer_t *timer);`
`void timerEnd(hw_timer_t *timer);`

Start/stop the timer.

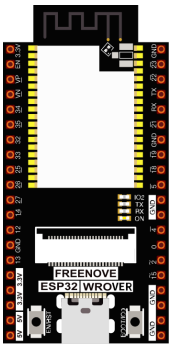
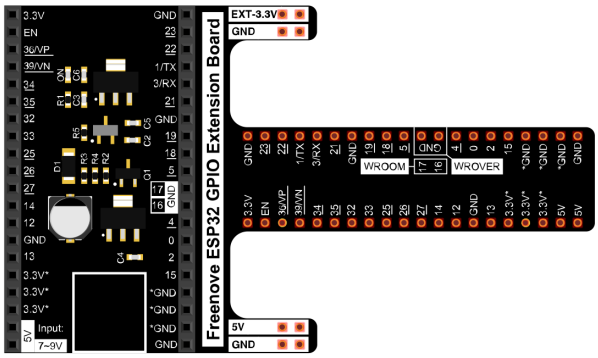
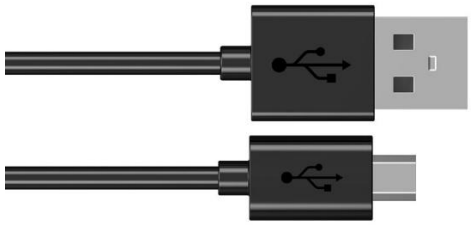
Chapter 7 Serial Communication

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

Project 7.1 Serial Print

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

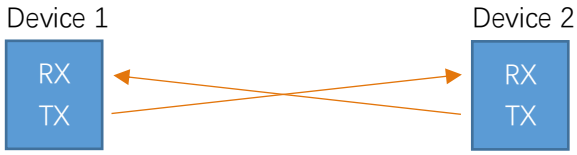
Component List

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

Related knowledge

Serial communication

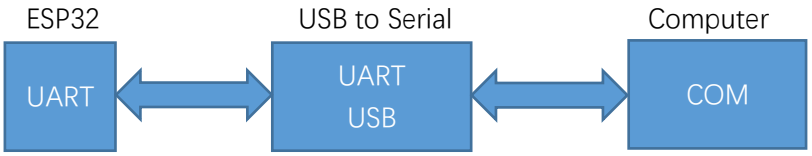
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



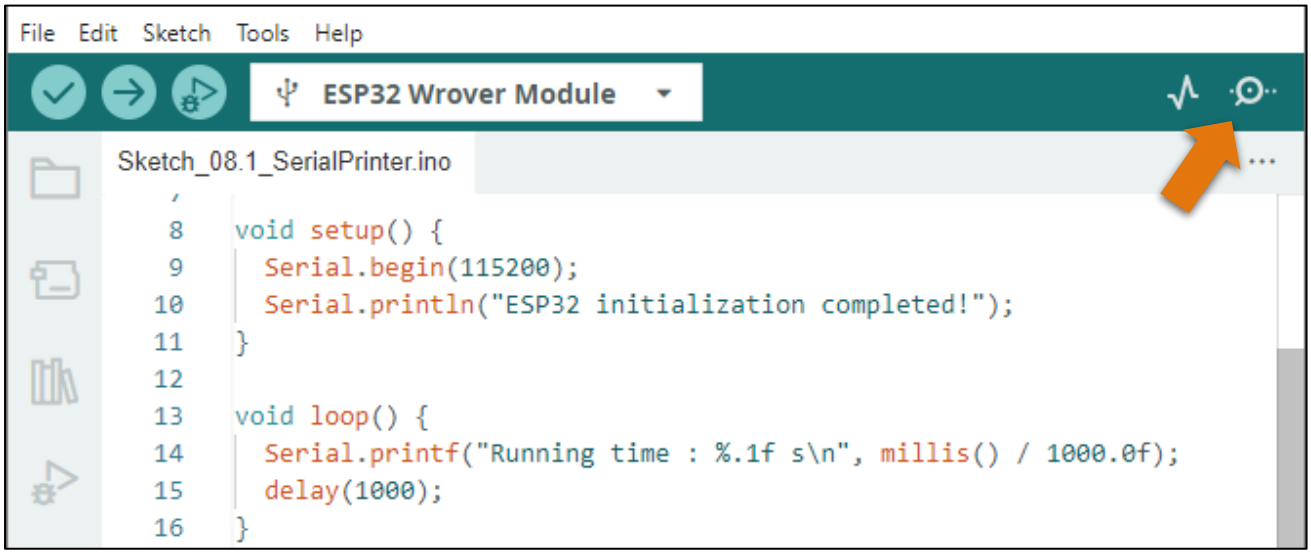
Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

Serial port on ESP32

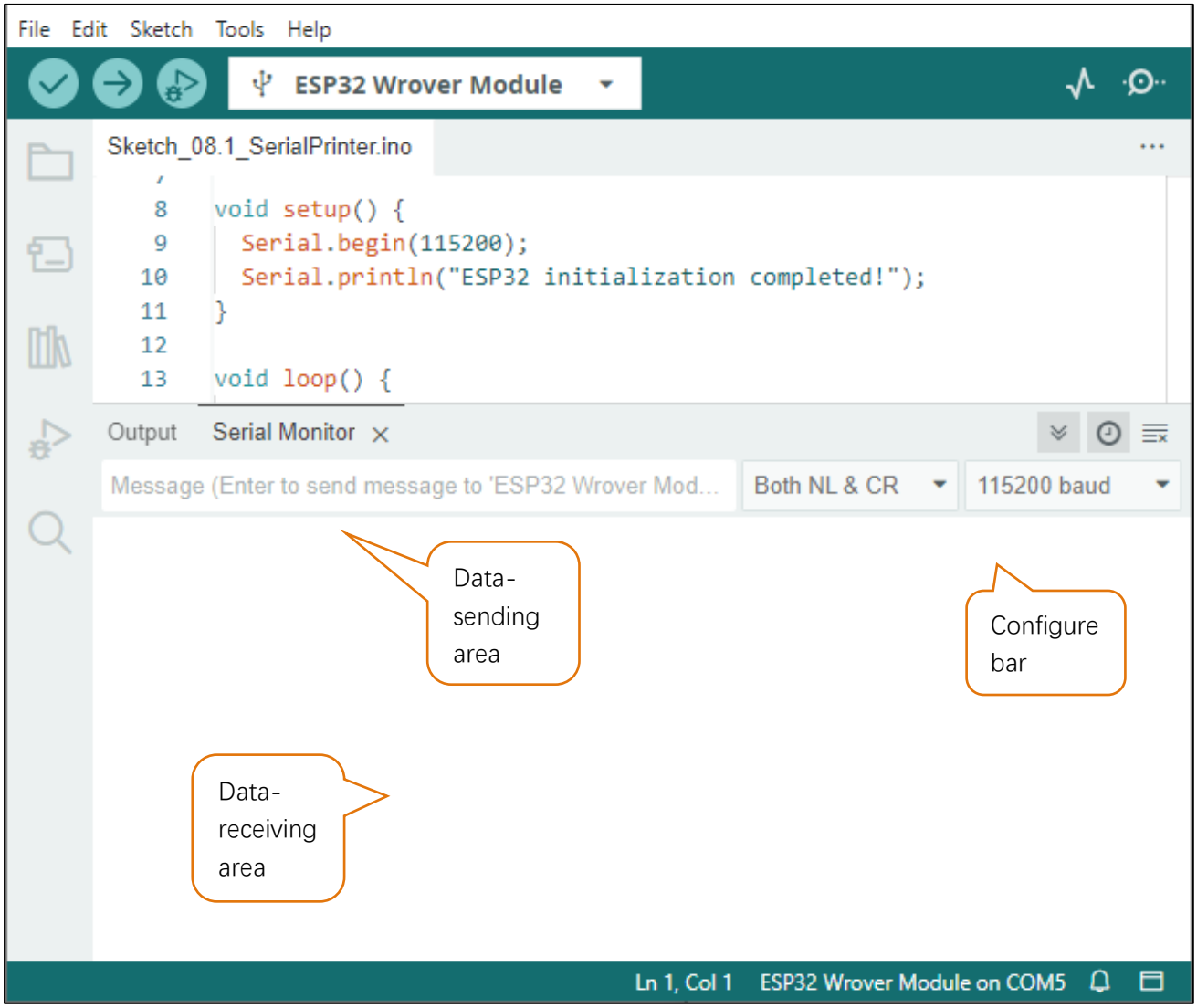
Freenove ESP32 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Arduino Software also uploads code to Freenove ESP32 through the serial connection.
Your computer identifies serial devices connecting to it as COMx. We can use the Serial Monitor window of Arduino Software to communicate with Freenove ESP32, connect Freenove ESP32 to computer through the USB cable, choose the correct device, and then click the Serial Monitor icon to open the Serial Monitor window.

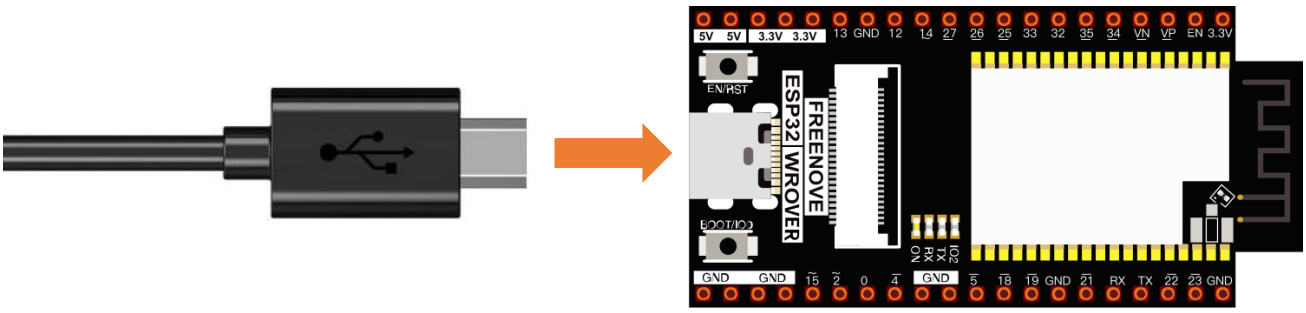


Interface of serial monitor window is as follows. If you can't open it, make sure Freenove ESP32 has been connected to the computer, and choose the right serial port in the menu bar "Tools".



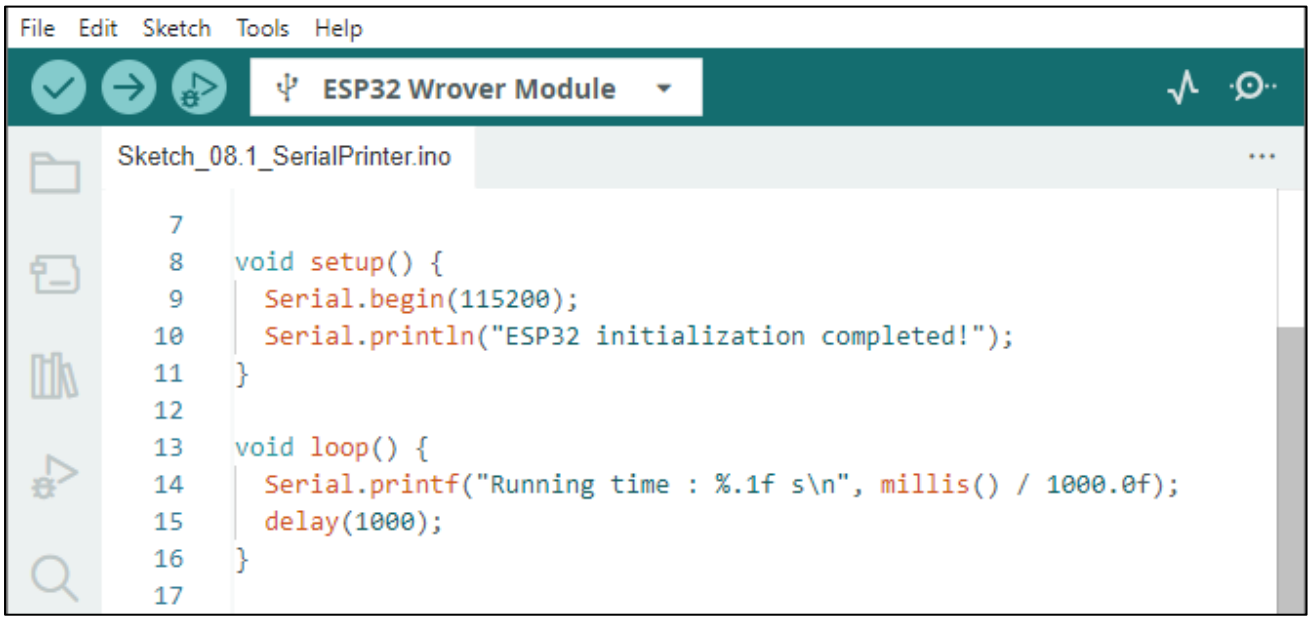
Circuit

Connect Freenove ESP32 to the computer with USB cable



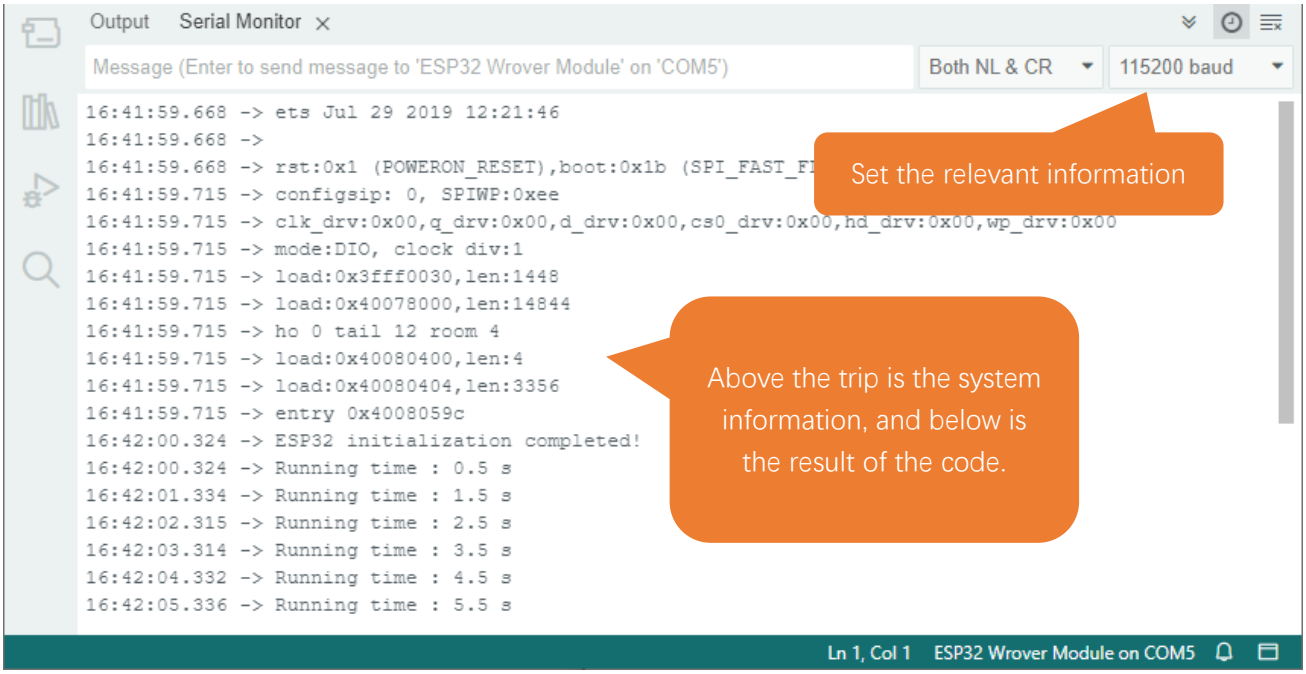
Sketch

Sketch_07.1_SerialPrinter




```
7
8   void setup() {
9     Serial.begin(115200);
10    Serial.println("ESP32 initialization completed!");
11  }
12
13  void loop() {
14    Serial.printf("Running time : %.1f s\n", millis() / 1000.0f);
15    delay(1000);
16  }
17
```

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200, and press the reset button. As shown in the following figure:



As shown in the image above, "ESP32 initialization completed! " The previous is the printing message when the system is started, it uses the baud rate of 120,000, which is incorrect, so the garbled code is displayed. The user program is then printed at a baud rate of 115200.

How do I disable messages printed at system startup?
Use a jumper wire and connect one of its ends to GPIO5 of development board and the other to GND, so that we can disable the system to print out startup messages.

For more information, click here.

The following is the program code:

```
1  void setup() {
2    Serial.begin(115200);
3    Serial.println("ESP32 initialization completed! ");
4  }
5
6  void loop() {
7    Serial.printf("Running time : %.1f s\n", millis() / 1000.0f);
8    delay(1000);
9  }
```

Reference

```
void begin(unsigned long baud, uint32_t config=SERIAL_8N1, int8_t rxPin=-1,
           int8_t txPin=-1, bool invert=false, unsigned long timeout_ms = 20000UL);
```

Initializes the serial port. Parameter baud is baud rate, other parameters generally use the default value.

```
size_t println( arg );
```

Print to the serial port and wrap. The parameter **arg** can be a number, a character, a string, an array of characters, etc.

```
size_t printf(const char * format, ...)  __attribute__ ((format (printf, 2, 3)));
```

Print formatted content to the serial port in the same way as print in standard C.

```
unsigned long millis();
```

Returns the number of milliseconds since the current system was booted.

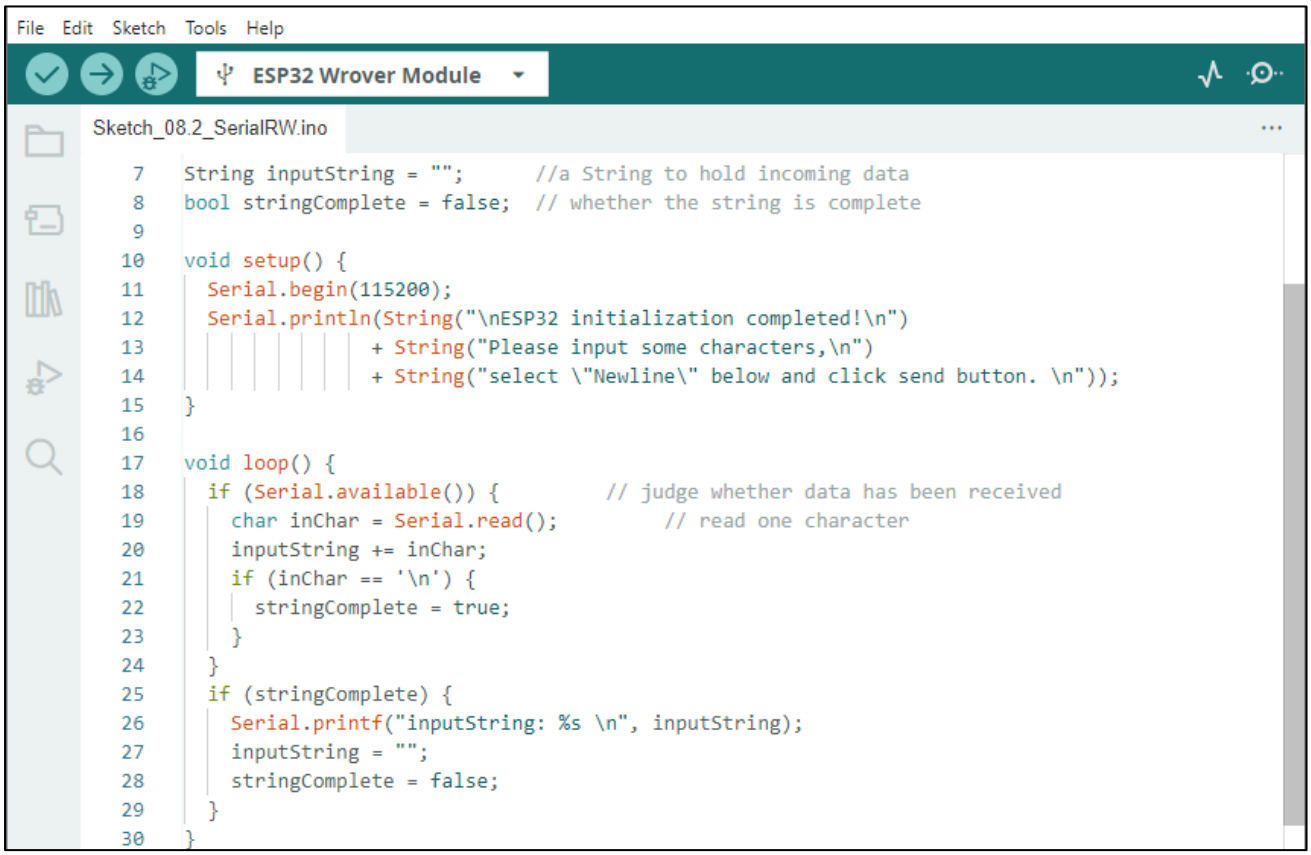
Project 7.2 Serial Read and Write

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

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

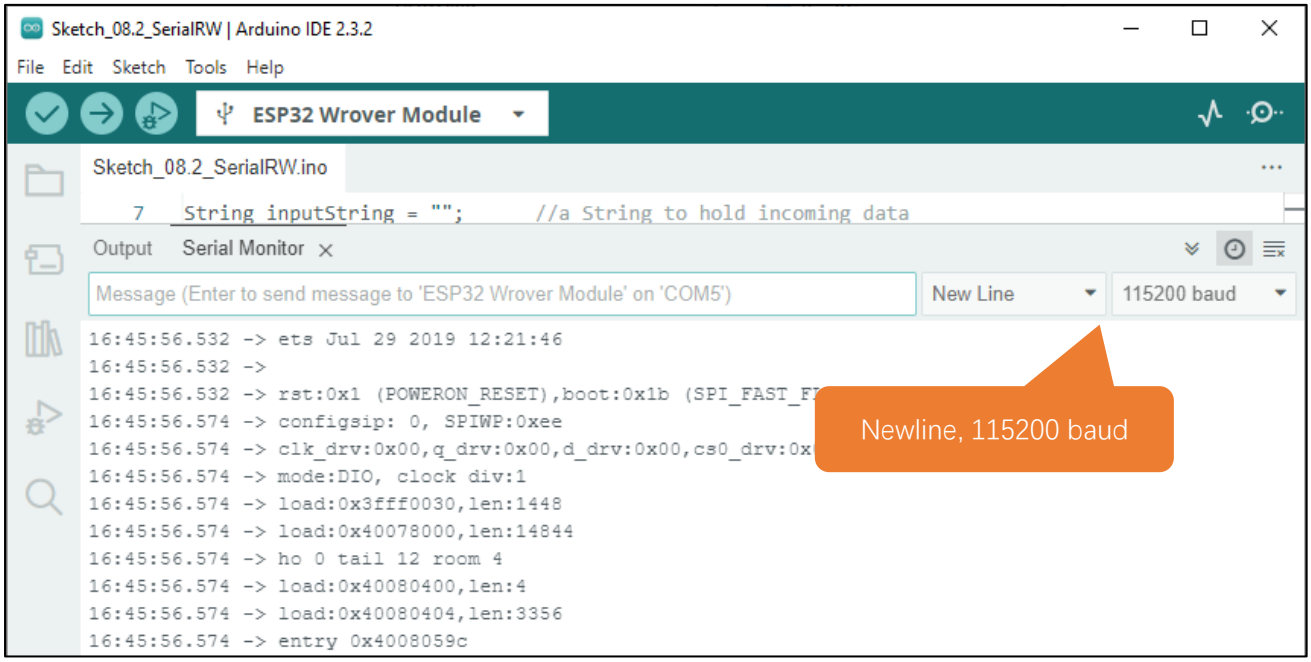
Sketch

Sketch_07.2_SerialRW

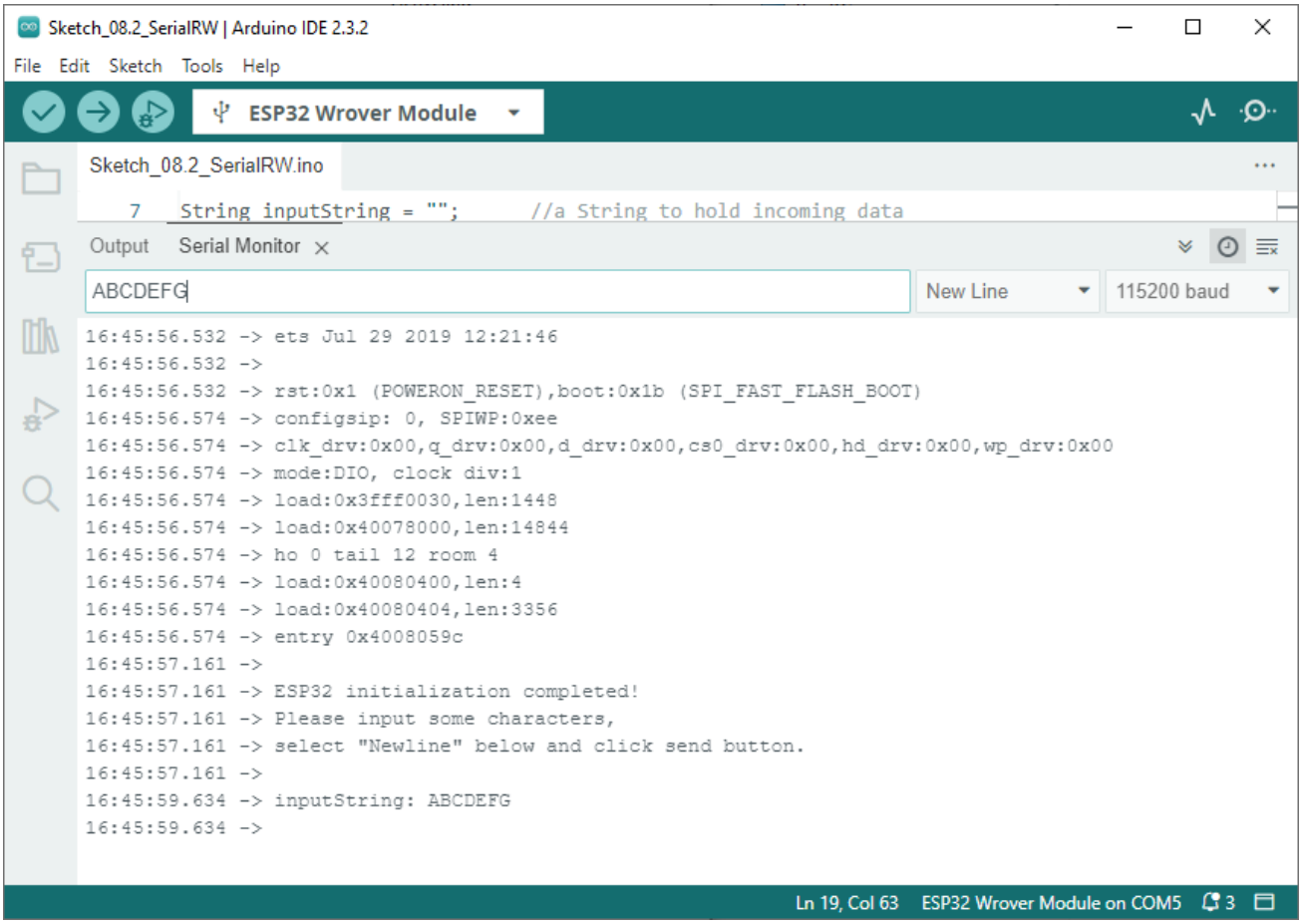



```
String inputString = "";      //a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

void setup() {
  Serial.begin(115200);
  Serial.println(String("\nESP32 initialization completed!\n")
                 + String("Please input some characters,\n")
                 + String("select \"Newline\" below and click send button. \n"));
}

void loop() {
  if (Serial.available()) {         // judge whether data has been received
    char inChar = Serial.read();          // read one character
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
  if (stringComplete) {
    Serial.printf("inputString: %s \n", inputString);
    inputString = "";
    stringComplete = false;
  }
}
```

Download the code to ESP32-WROVER, open the serial monitor, and set the bottom to Newline, 115200. As shown in the following figure:



Type characters such as 'ABCDEFG' at the top, then press Enter to send the data to the ESP32, and the serial port monitor will print out the data received and forwarded back by the ESP32.



The following is the program code:

```
String inputString = "";      //a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

void setup() {
  Serial.begin(115200);
  Serial.println(String("\nESP32 initialization completed! \n")
                 + String("Please input some characters,\n")
                 + String("select \"Newline\" below and click send button. \n"));
}

void loop() {
  if (Serial.available()) {         // judge whether data has been received
    char inChar = Serial.read();          // read one character
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
  if (stringComplete) {
    Serial.printf("inputString: %s \n", inputString);
    inputString = "";
    stringComplete = false;
  }
}
```

In loop(), determine whether the serial port has data, if so, read and save the data, and if the newline character is read, print out all the data that has been read.

Reference

`String();`
Constructs an instance of the String class. For more information, please visit https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/
`int available(void);`
Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer.
`Serial.read();`
Reads incoming serial data.

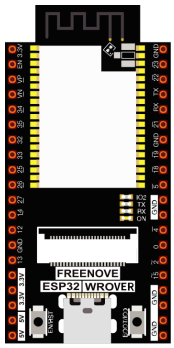
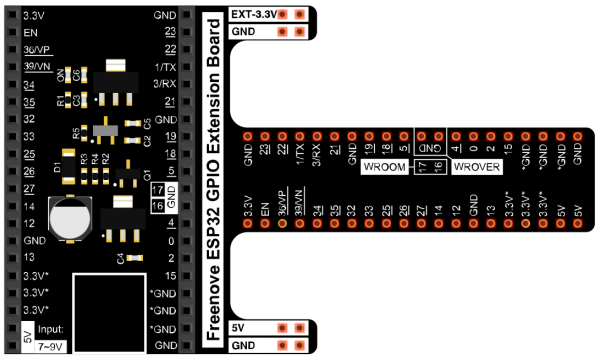
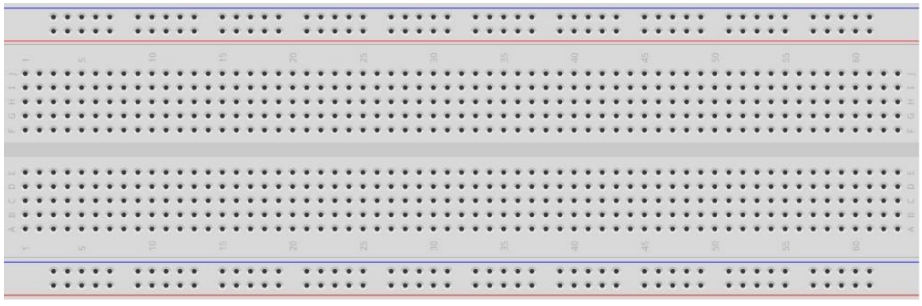
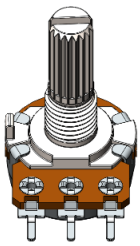
Chapter 8 AD/DA Converter

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

Project 8.1 Read the Voltage of Potentiometer

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

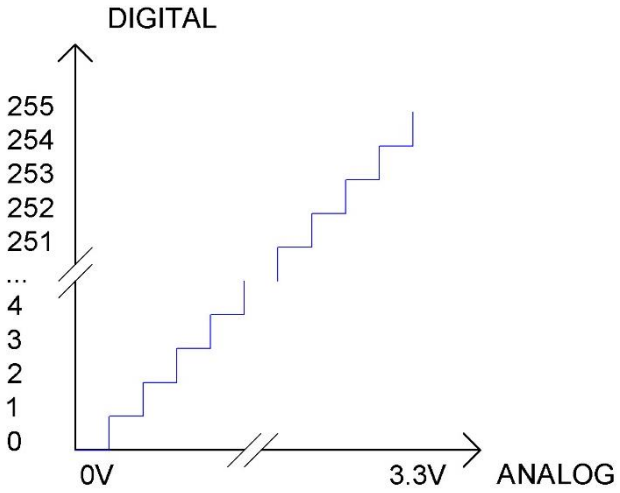
Component List

<table>
<tr><td>ESP32-WROVER 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>

Related knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC on ESP32 is 12 bits, that means the resolution is 2^12=4096, and it represents a range (at 3.3V) will be divided equally to 4096 parts. The 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:

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

DAC

The reversing of this process requires a DAC, Digital-to-Analog Converter. The digital I/O port can output high level and low level (0 or 1), but cannot output an intermediate voltage value. This is where a DAC is useful. ESP32 has two DAC output pins with 8-bit accuracy, GPIO25 and GPIO26, which can divide VDD (here is 3.3V) into 2*8=256 parts. For example, when the digital quantity is 1, the output voltage value is 3.3/256 *1 V, and when the digital quantity is 128, the output voltage value is 3.3/256 *128=1.65V, the higher the accuracy of DAC, the higher the accuracy of output voltage value will be.

The conversion formula is as follows:

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

ADC on ESP32

ESP32 has two digital analog converters with successive approximations of 12-bit accuracy, and a total of 16 pins can be used to measure analog signals. GPIO pin sequence number and analog pin definition are shown in the following table.

Pin number in Arduino	GPIO number	ADC channel
A0	GPIO 36	**ADC1_CH0**
A3	GPIO 39	**ADC1_CH3**
A4	GPIO 32	**ADC1_CH4**
A5	GPIO 33	**ADC1_CH5**
A6	GPIO 34	**ADC1_CH6**
A7	GPIO 35	**ADC1_CH7**
A10	GPIO 4	ADC2_CH0
A11	GPIO 0	ADC2_CH1
A12	GPIO 2	ADC2_CH2
A13	GPIO 15	ADC2_CH3
A14	GPIO 13	ADC2_CH4
A15	GPIO 12	ADC2_CH5
A16	GPIO 14	ADC2_CH6
A17	GPIO 27	ADC2_CH7
A18	GPIO 25	ADC2_CH8
A19	GPIO 26	ADC2_CH9

The analog pin number is also defined in ESP32's code base. For example, you can replace GPIO36 with A0 in the code.

Note: ADC2 is disabled when ESP32's WiFi function is enabled.

DAC on ESP32

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

Simulate pin number	GPIO number
DAC1	25
DAC2	26

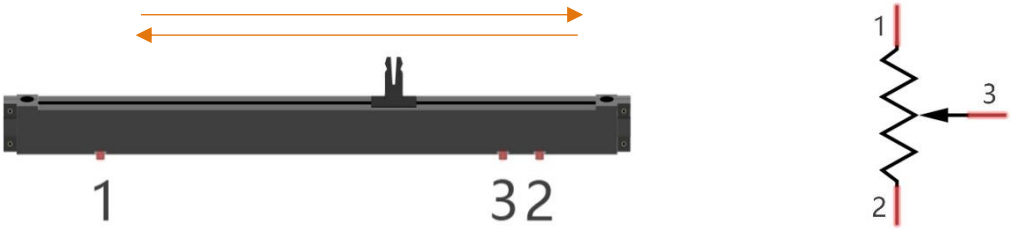
The DAC pin number is already defined in ESP32's code base; for example, you can replace GPIO25 with DAC1 in the code.

Note: In this ESP32, GPIO26 is used as the camera's IIC-SDA pin, which is connected to 3.3V through a resistor.Therefore, DAC2 cannot be used.

Component knowledge

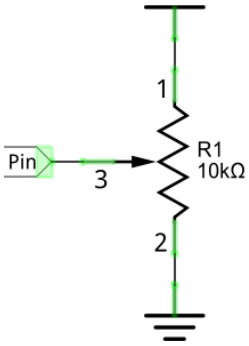
Potentiometer

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.



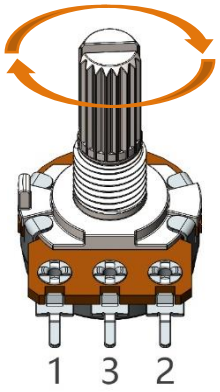
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

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.



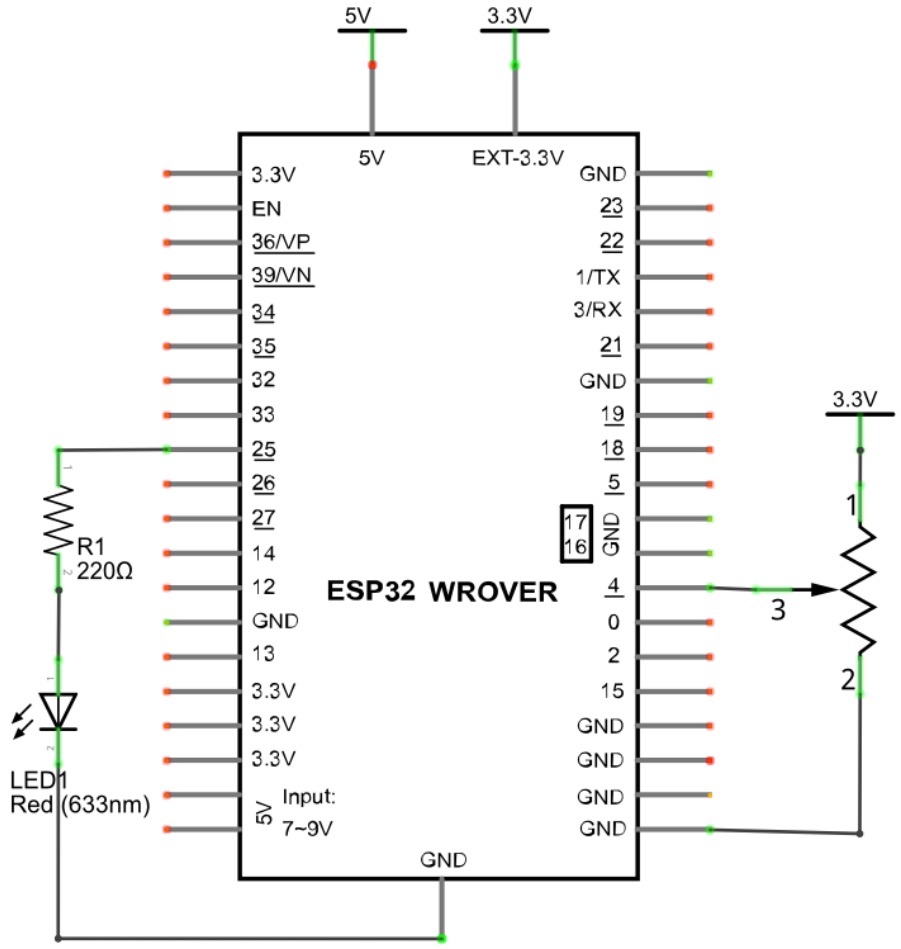
Rotary potentiometer

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

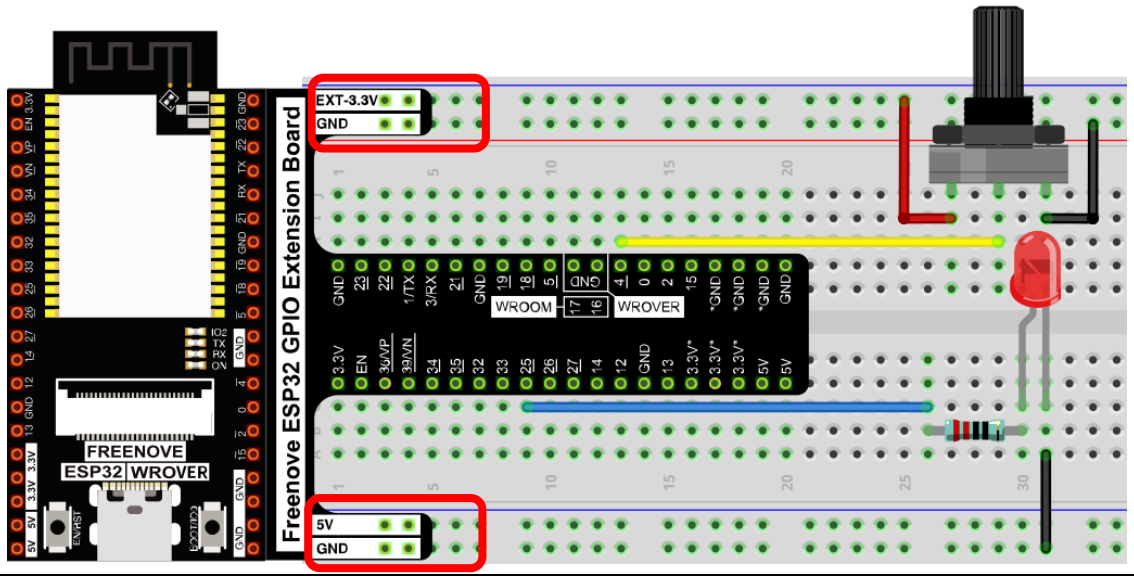


Circuit

Schematic diagram

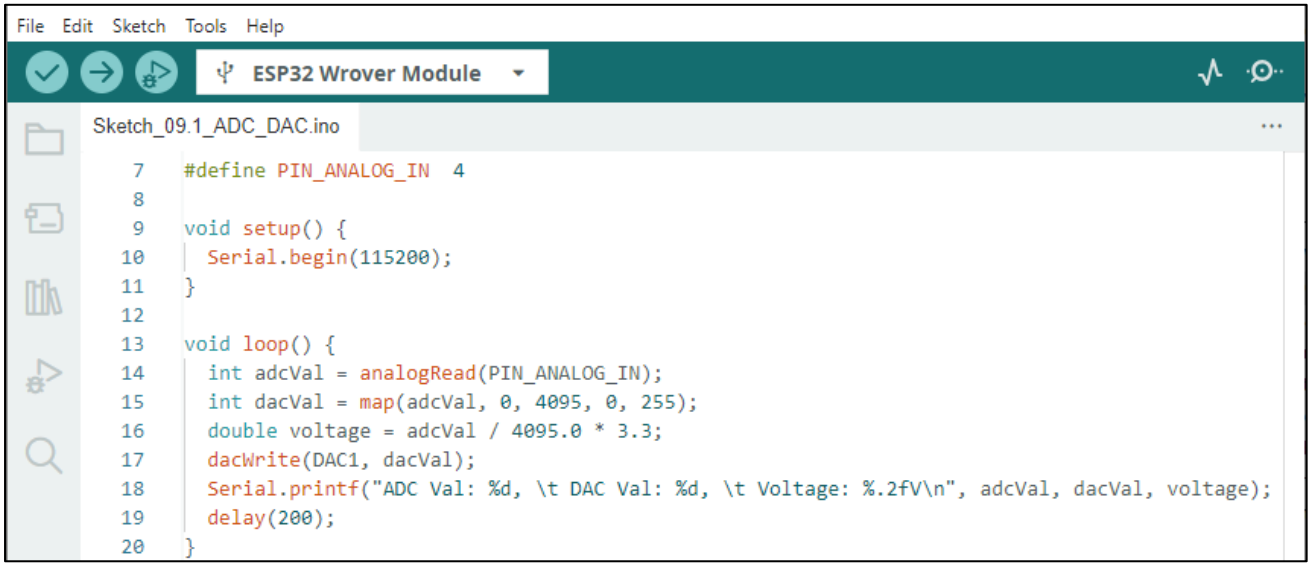


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

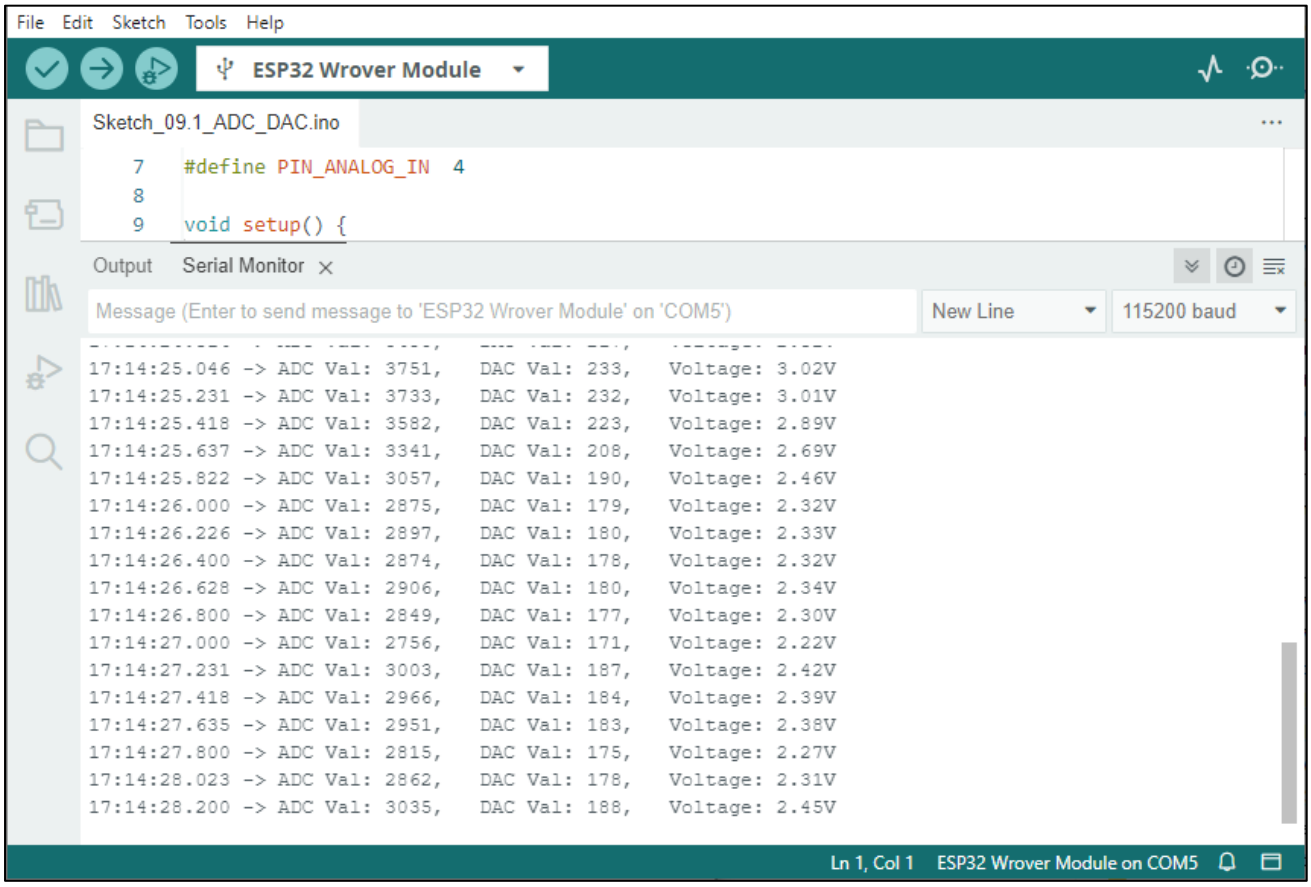


Sketch

Sketch_08.1_ADC_DAC



Download the code to ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. As shown in the following figure,



The serial monitor prints ADC values, DAC values, and the output voltage of the potentiometer. In the code, we made the voltage output from the DAC pin equal to the voltage input from the ADC pin. Rotate the handle of the potentiometer and the print will change. When the voltage is greater than 1.6V (voltage needed to turn on red LED), LED starts emitting light. If you continue to increase the output voltage, the LED will become more and more brighter. When the voltage is less than 1.6v, the LED will not light up, because it does not reach the voltage to turn on LED, which indirectly proves the difference between DAC and PWM. (if you have an oscilloscope, you can check the waveform of the DAC output through it.)

The following is the code:

```
#define PIN_ANALOG_IN  4

void setup() {
  Serial.begin(115200);
}

void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  dacWrite(DAC1, dacVal);
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
```

In loop(), the analogRead() function is used to obtain the ADC value, and then the map() function is used to convert the value into an 8-bit precision DAC value. The function dacWrite() is used to output the value. The input and output voltage are calculated according to the previous formula, and the information is finally printed out.

```
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  dacWrite(DAC1, dacVal);
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
```

Reference

`uint16_t analogRead(uint8_t pin);`
Reads the value from the specified analog pin. Return the analog reading on the pin. (0-4095 for 12 bits).
`void dacWrite(uint8_t pin, uint8_t value);`
This writes the given value to the supplied analog pin.
`long map(long value,long fromLow,long fromHigh,long toLow,long toHigh);`
Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.

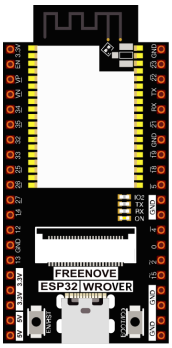
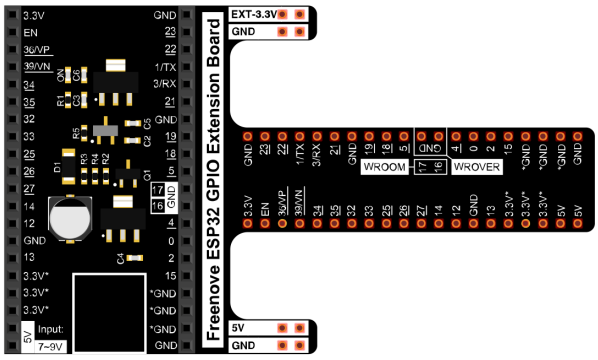
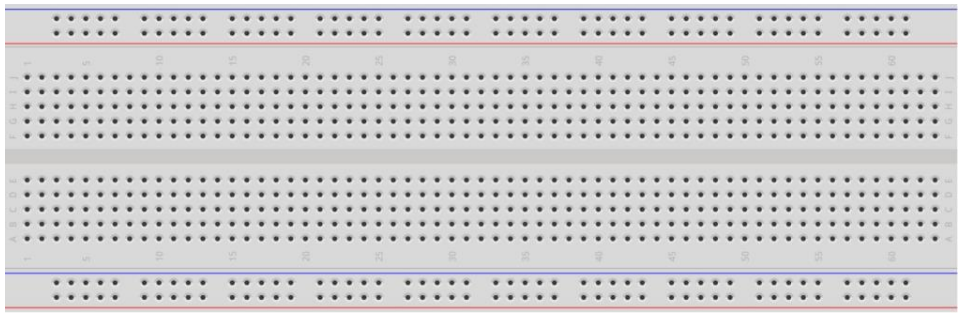
Chapter 9 Touch Sensor

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

Project 9.1 Read Touch Sensor

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

Component List

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

Related knowledge

Touch sensor

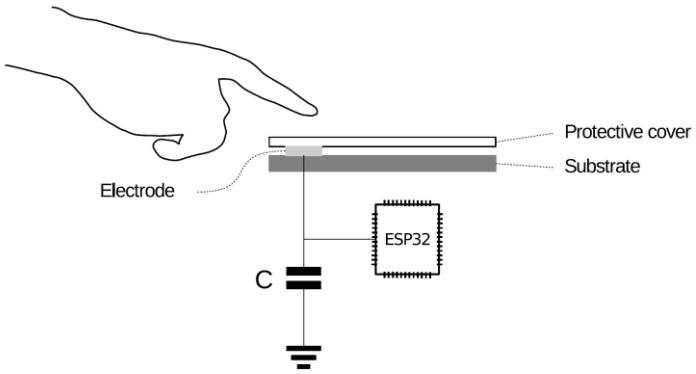
ESP32's touch sensor supports up to 10 GPIO channels as capacitive touch pins. Each pin can be used separately as an independent touch switch or be combined to produce multiple touch points. The following table is a list of available touch pins on ESP32.

Name of touch sensing signal	Functions of pins	GPIO number
T0	GPIO4	GPIO4
T1	GPIO0	GPIO0
T2	GPIO2	GPIO2
T3	MTDO	GPIO15
T4	MTCK	GPIO13
T5	MTDI	GPIO12
T6	MTMS	GPIO14
T7	GPIO27	GPIO27
T8	32K_XN	GPIO33
T9	32K_XP	GPIO32

The touch pin number is already defined in ESP32's code base. For example, in the code, you can use T0 to represent GPIO4.

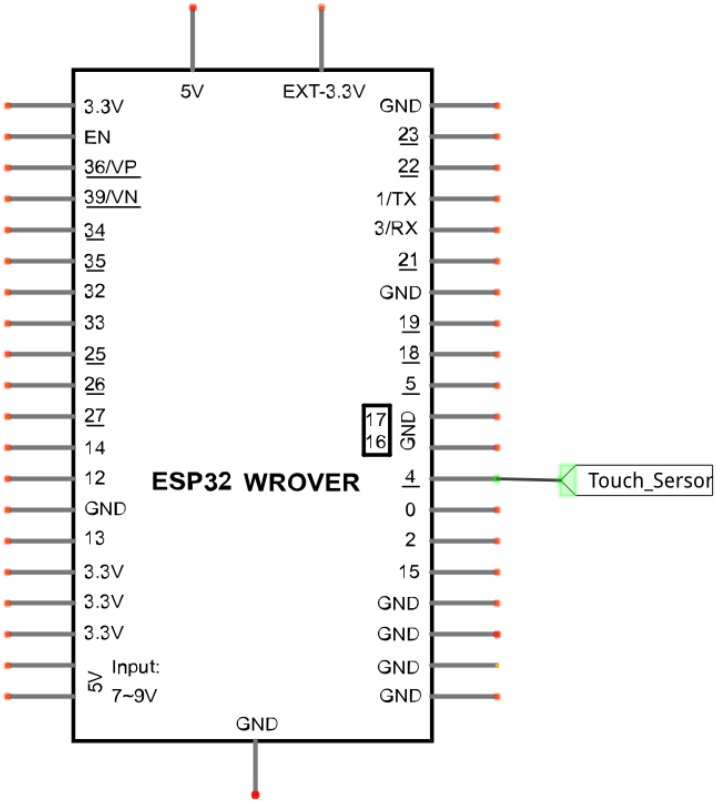
The electrical signals generated by touch are analog data, which are converted by an internal ADC converter. You may have noticed that all touch pins have ADC functionality.

The hardware connection method is shown in the following figure.

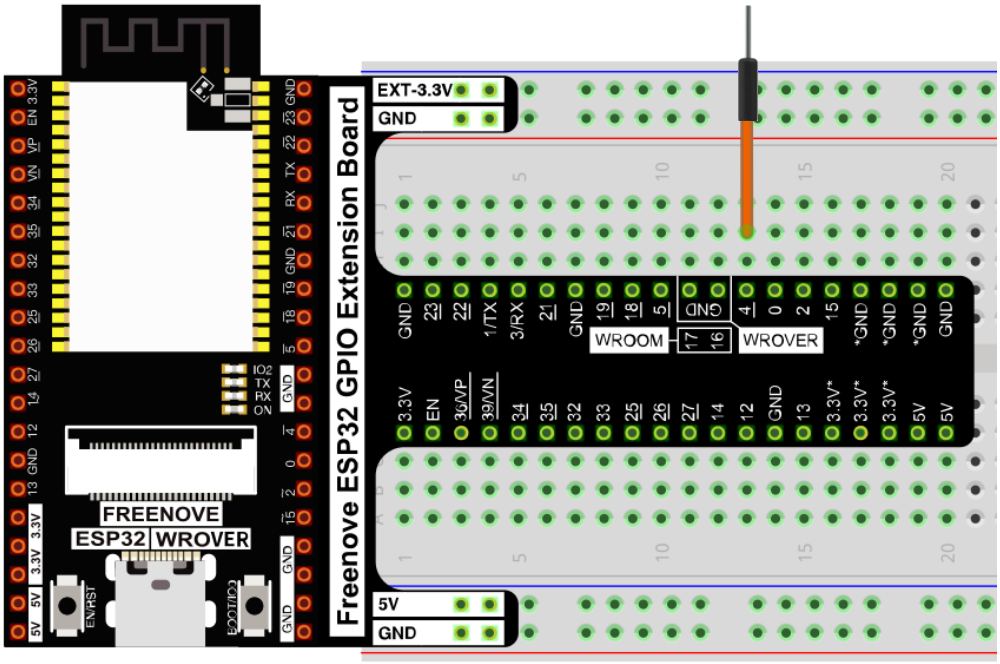


Circuit

Schematic diagram



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



Sketch

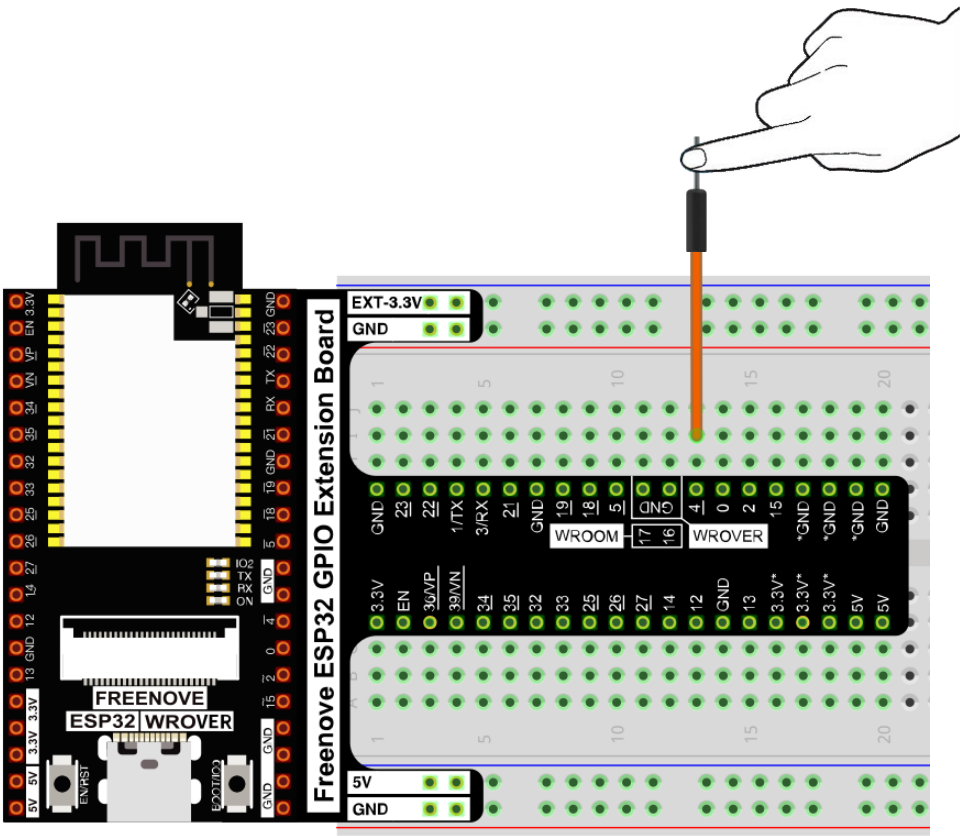
Sketch_09.1_TouchRead

File Edit Sketch Tools Help

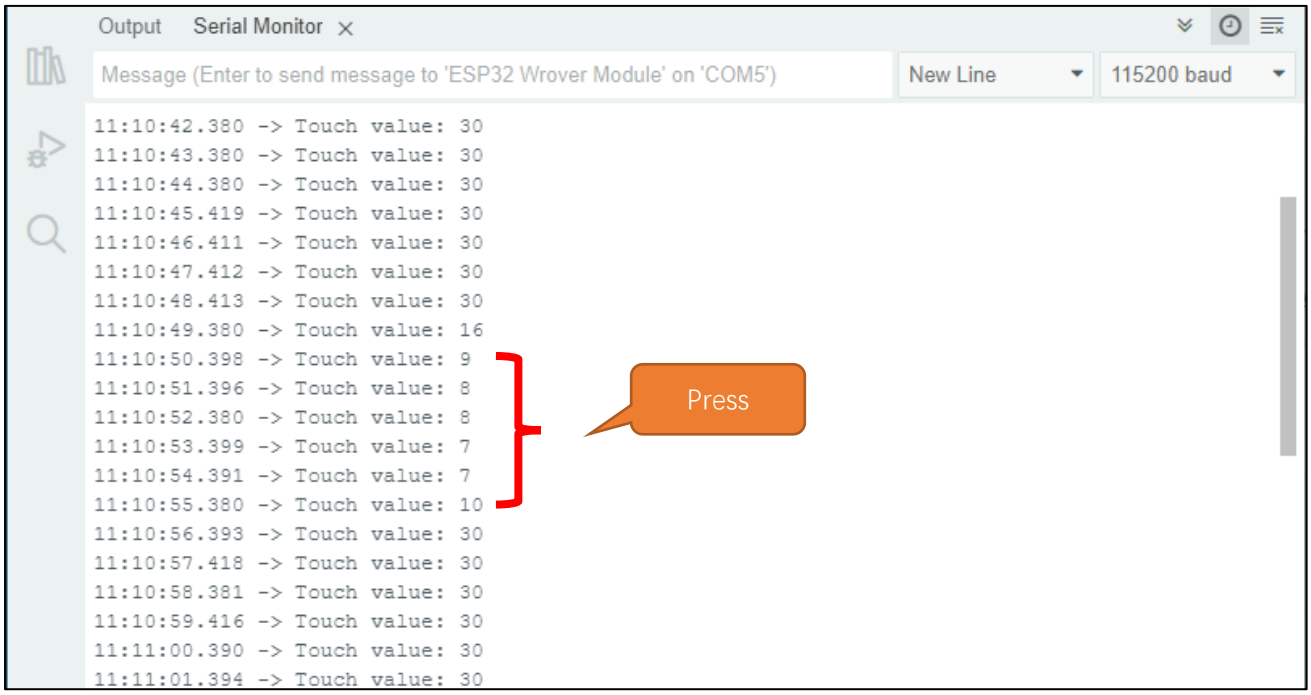
ESP32 Wrover Module

Sketch_10.1_TouchRead.ino

```
/*****************************************************************************
  Filename    : TouchRead
  Description : Read touch sensor value.
  Auther      : www.freenove.com
  Modification: 2024/06/18
*****************************************************************************/

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.printf("Touch value: %d \n",touchRead(T0));  // get value using T0 (GPIO4)
  delay(1000);
}
```



Download the code to ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. As shown in the following figure.



Touched by hands, the value of the touch sensor will change. The closer the value is to zero, the more obviously the touch action will be detected. The value detected by the sensor may be different in different environments or when different people touch it. The code is very simple, just look at Reference.

Reference

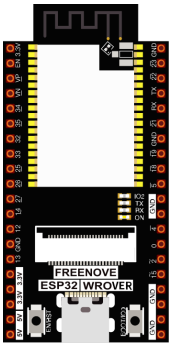
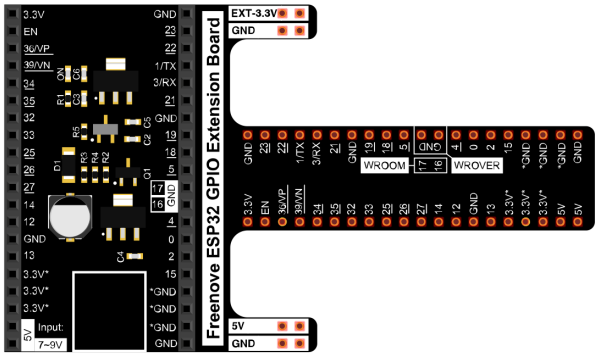
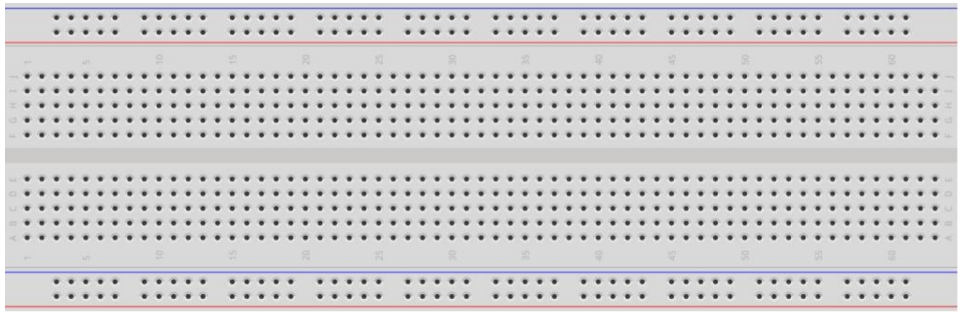
```
uint16_t touchRead(uint8_t pin);
```

Read touch sensor value. (values close to 0 mean touch detected)

Project 9.2 Touch Lamp

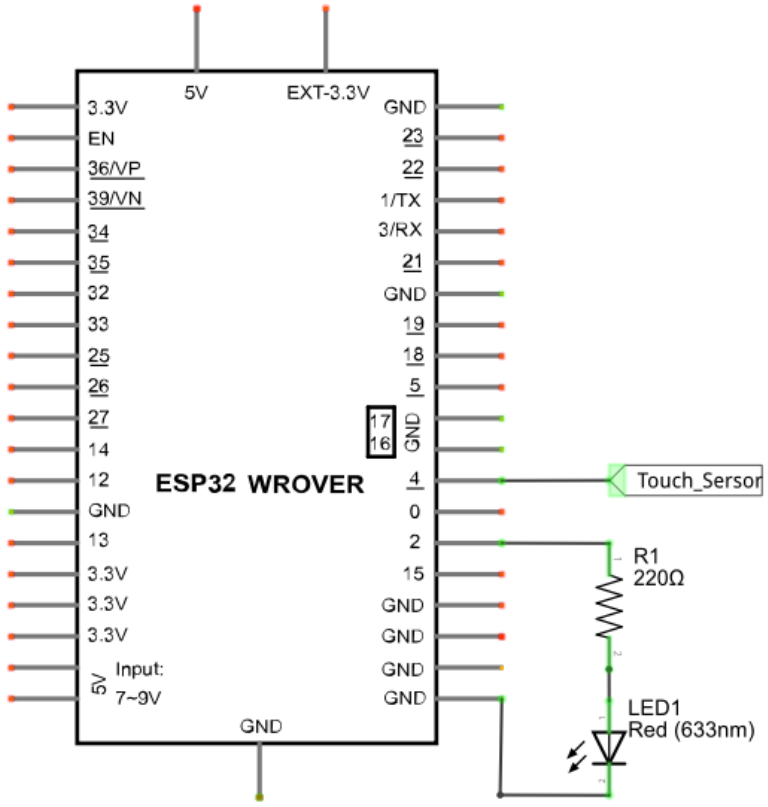
In this project, we will use ESP32's touch sensor to create a touch switch lamp.

Component List

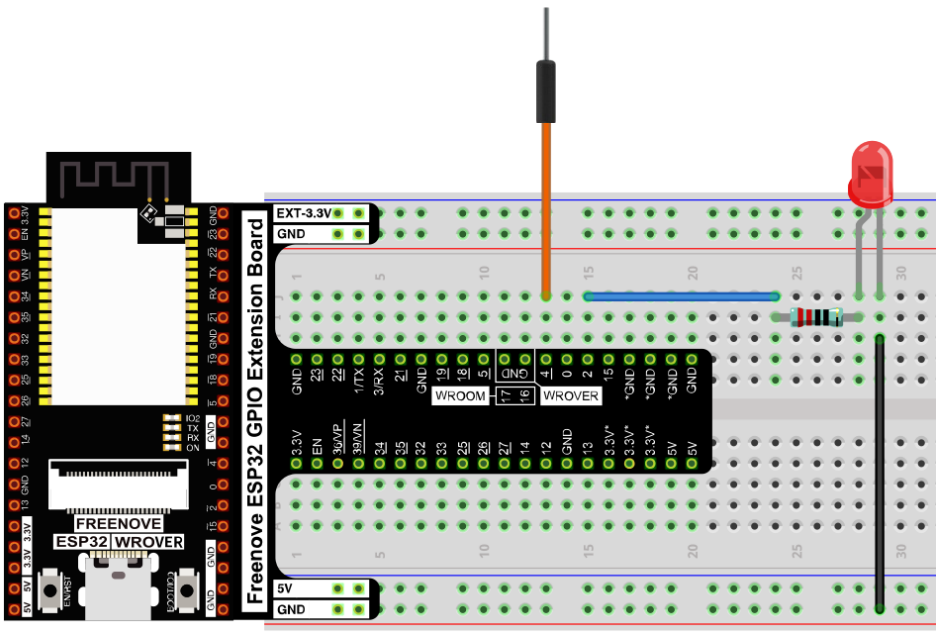
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x3 	LED x1 	Resistor 220Ω x1 

Circuit

Schematic diagram



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



Sketch

Sketch_09.2_TouchLamp

```
File Edit Sketch Tools Help
ESP32 Wrover Module
Sketch_10.2_TouchLamp.ino
```

```
7   #define PIN_LED   2
8   #define PRESS_VAL 14    //Set a threshold to judge touch
9   #define RELEASE_VAL 25  //Set a threshold to judge release
10
11  bool isProcessed = false;
12  void setup() {
13    Serial.begin(115200);
14    pinMode(PIN_LED, OUTPUT);
15  }
16  void loop() {
17    if (touchRead(T0) < PRESS_VAL) {
18      if (!isProcessed) {
19        isProcessed = true;
20        Serial.println("Touch detected! ");
21        reverseGPIO(PIN_LED);
22      }
23    }
24
25    if (touchRead(T0) > RELEASE_VAL) {
26      if (isProcessed) {
27        isProcessed = false;
28        Serial.println("Released! ");
29      }
30    }
31  }
```

Download the code to ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. As shown in the following figure,

```
Output  Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')   New Line   115200 baud
13:29:49.761 -> Touch detected!
13:29:50.049 -> Released!
13:29:52.723 -> Touch detected!
13:29:53.198 -> Released!
13:29:54.558 -> Touch detected!
13:29:55.092 -> Released!
13:29:56.041 -> Touch detected!
13:29:56.777 -> Released!
13:29:57.499 -> Touch detected!
13:29:57.931 -> Released!
13:29:58.476 -> Touch detected!
13:29:58.914 -> Released!
13:29:59.340 -> Touch detected!
13:29:59.721 -> Released!
```

With a touch pad, the state of the LED changes with each touch, and the detection state of the touch sensor is printed in the serial monitor.

The following is the program code:

```
#define PIN_LED   2
#define PRESS_VAL 14     //Set a threshold to judge touch
#define RELEASE_VAL 25   //Set a threshold to judge release

bool isProcessed = false;
void setup() {
  Serial.begin(115200);
  pinMode(PIN_LED, OUTPUT);
}
void loop() {
  if (touchRead(T0) < PRESS_VAL) {
    if (! isProcessed) {
      isProcessed = true;
      Serial.println("Touch detected!  ");
      reverseGPIO(PIN_LED);
    }
  }

  if (touchRead(T0) > RELEASE_VAL) {
    if (isProcessed) {
      isProcessed = false;
      Serial.println("Released!  ");
    }
  }
}

void reverseGPIO(int pin) {
  digitalWrite(pin, ! digitalRead(pin));
}
```

The closer the return value of the function touchRead() is to 0, the more obviously the touch is detected. This is not a fixed value, so you need to define a threshold that is considered valid (when the value of the sensor is less than this threshold). Similarly, a threshold value is to be defined in the release state, and a value in between is considered an invalid disturbance value.

```
#define PRESS_VAL 14    //Set a threshold to judge touch
#define RELEASE_VAL 25  //Set a threshold to judge release
```

In loop(), first determine whether the touch was detected. If yes, print some messages, flip the state of the LED, and set the flag bit **isProcessed** to true to avoid repeating the program after the touch was successful.

```
11    if (touchRead(T0) < PRESS_VAL) {
12      if (! isProcessed) {
13        isProcessed = true;
14        Serial.println("Touch detected!  ");
15        reverseGPIO(PIN_LED);
16      }
17    }
```

It then determines if the touch key is released, and if so, prints some messages and sets the **isProcessed** to false to avoid repeating the process after the touch release and to prepare for the next touch probe.

```
19    if (touchRead(T0) > RELEASE_VAL) {
20      if (isProcessed) {
21        isProcessed = false;
22        Serial.println("Released!  ");
23      }
24    }
```

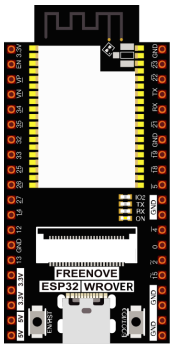
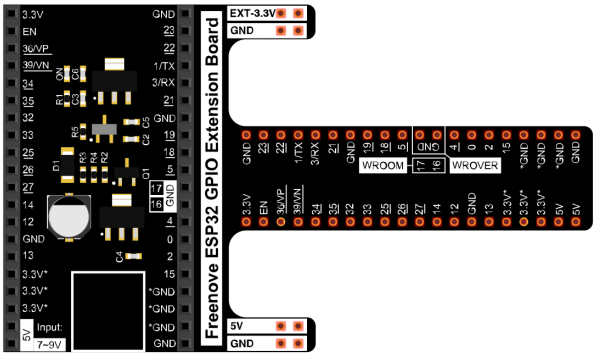
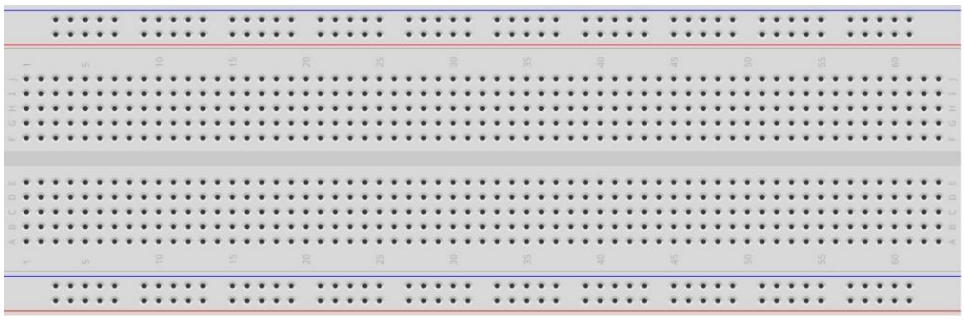
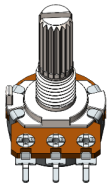
Chapter 10 Potentiometer & LED

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

Project 10.1 Soft Light

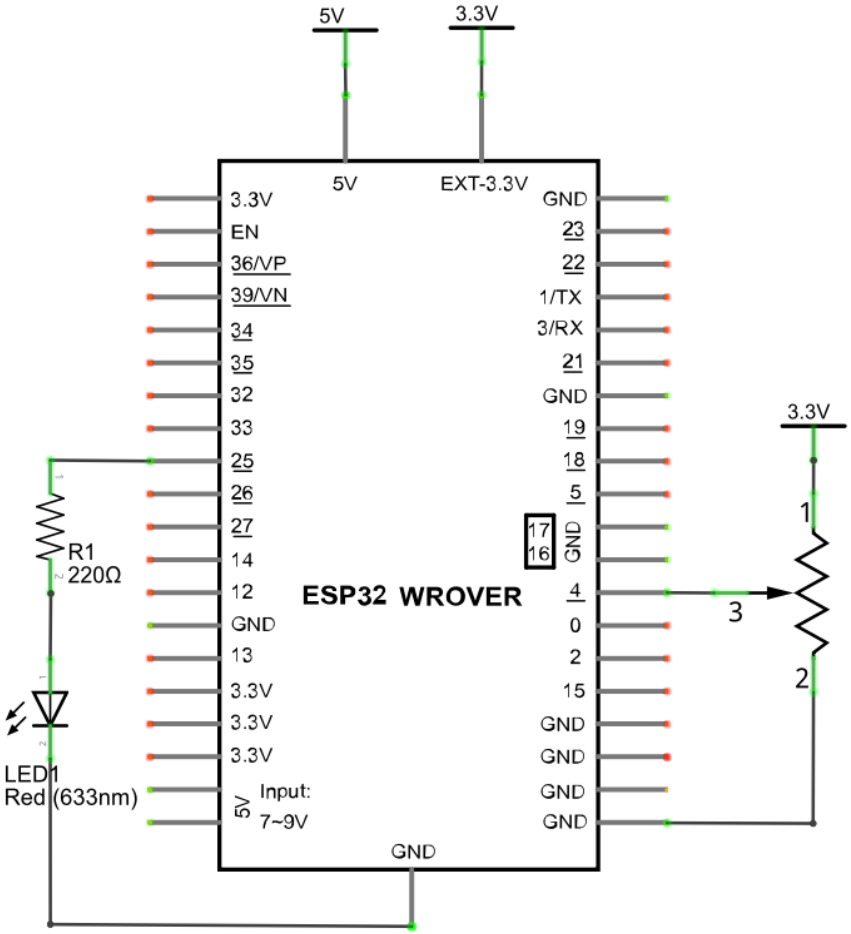
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of a LED. Then you can change the brightness of a LED by adjusting the potentiometer.

Component List

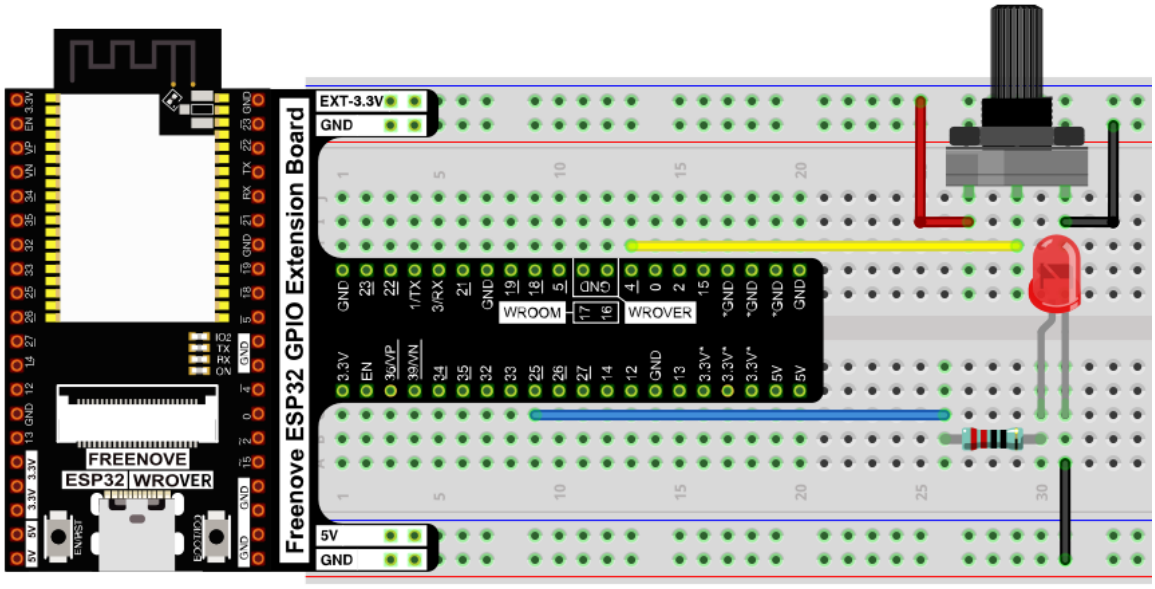
<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
</td><td>LED x1
  </td><td>Jumper M/M x5
</td></tr>
</table>

Circuit

Schematic diagram

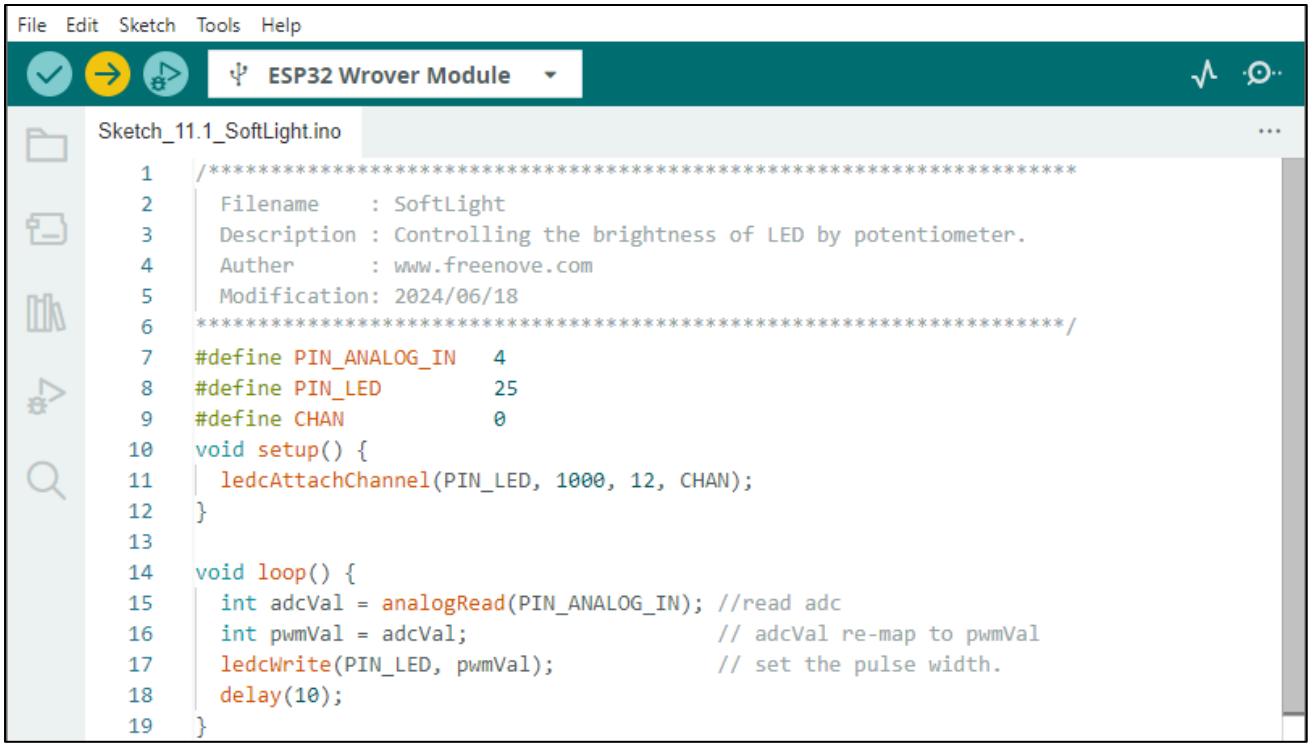


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_10.1_Softlight



Download the code to ESP32-WROVER, by turning the adjustable resistor to change the input voltage of GPIO25, ESP32 changes the output voltage of GPIO4 according to this voltage value, thus changing the brightness of the LED.

The following is the code:

```
#define PIN_ANALOG_IN   4
#define PIN_LED         25
#define CHAN            0
void setup() {
  ledcAttachChannal(PIN_LED, 1000, 12, CHAN);
}

void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN); //read adc
  int pwmVal = adcVal;                    // adcVal re-map to pwmVal
  ledcWrite(PIN_LED, pwmVal);             // set the pulse width.
  delay(10);
}
```

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

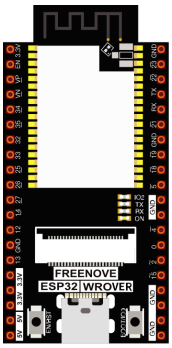
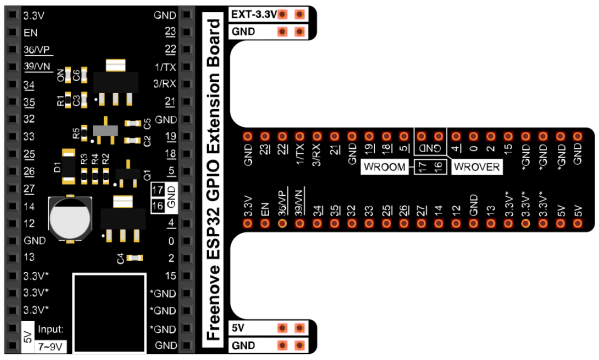
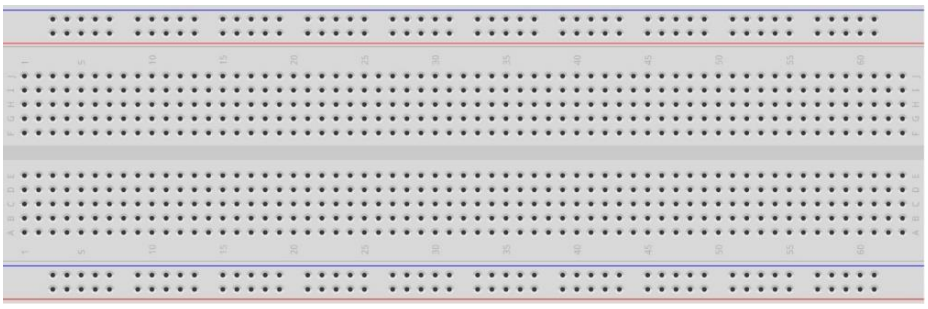
Chapter 11 Photoresistor & LED

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

Project 11.1 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.

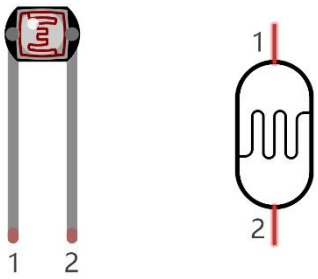
Component List

<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">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>

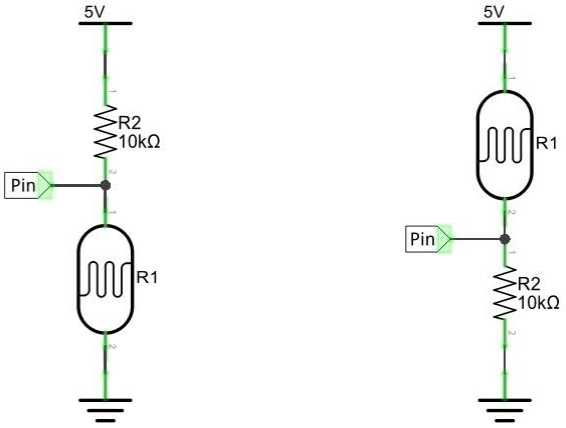
Component knowledge

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:

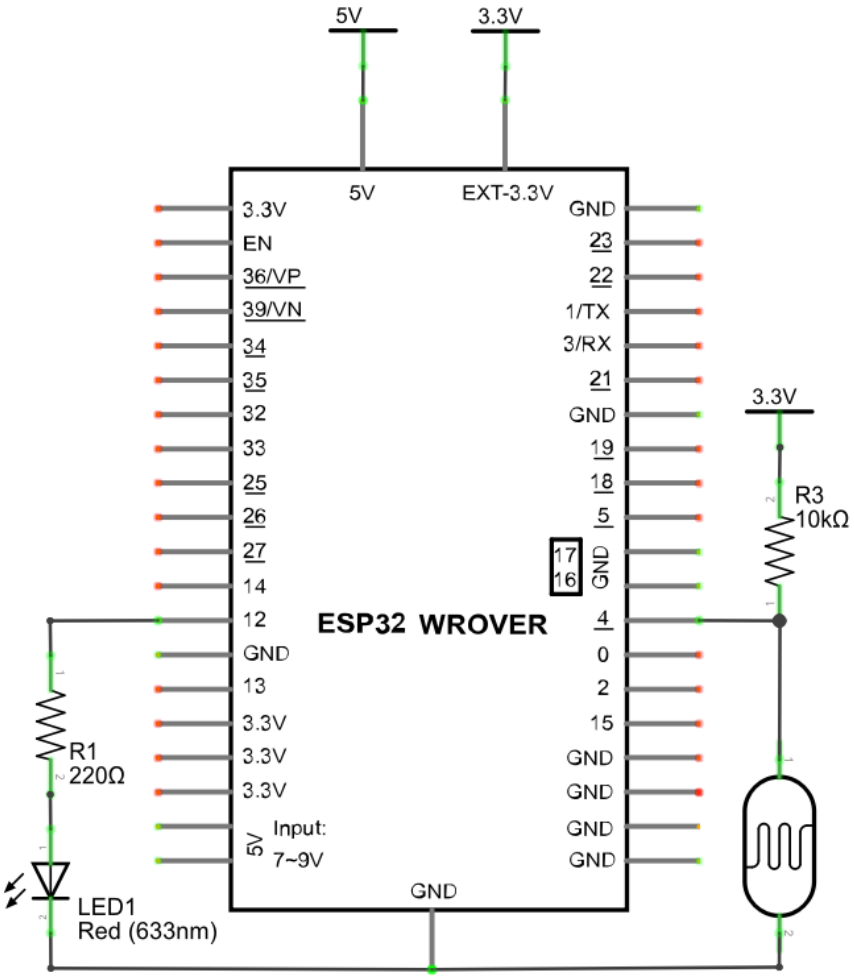


In the above circuit, when a photoresistor's resistance vale changes due to a change in light intensity, the voltage between the photoresistor and resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

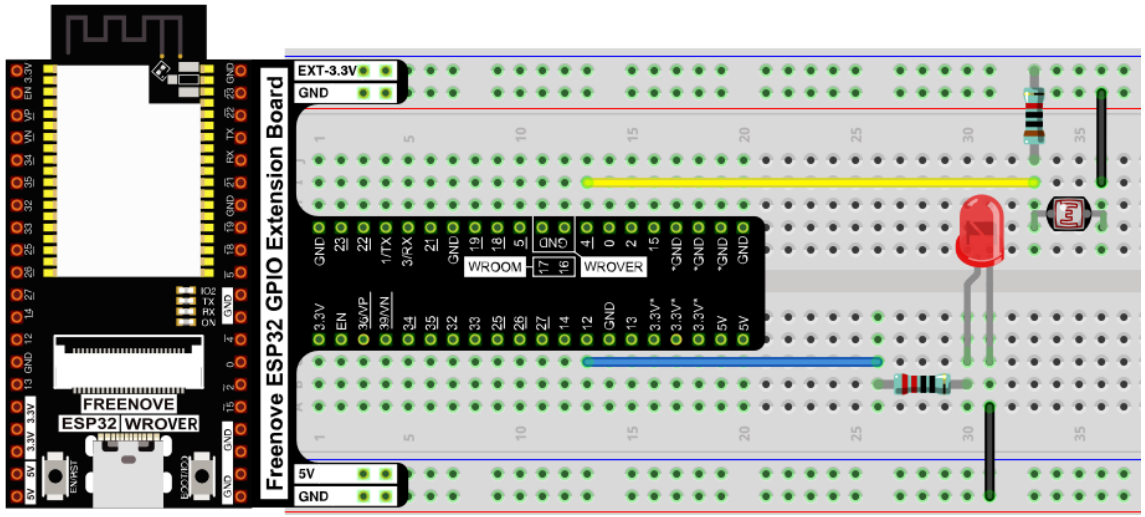
Circuit

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

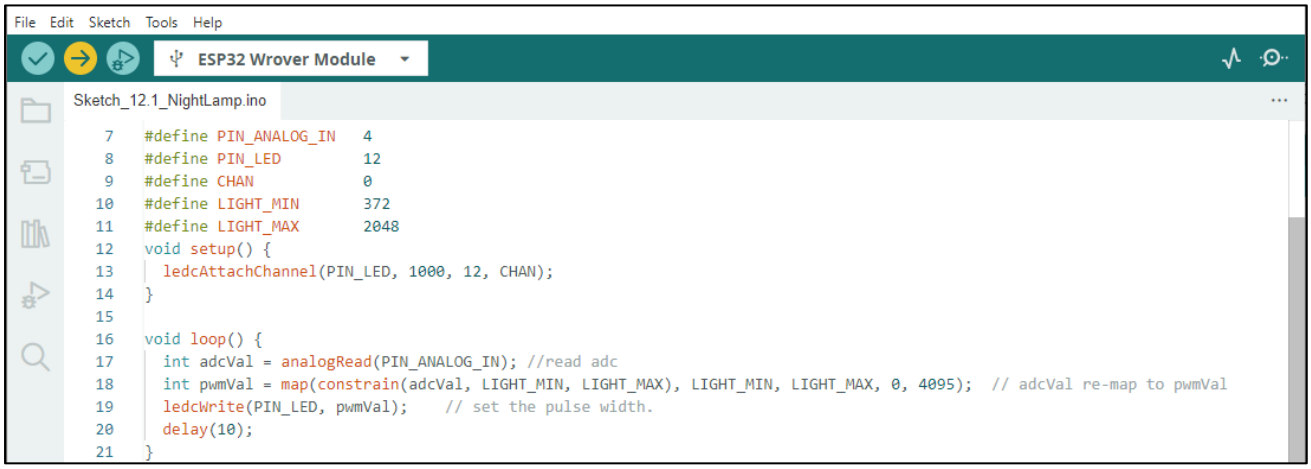


Any concerns? ✉ support@freenove.com

Sketch

The circuit used is similar to the project Soft Light. The only difference is that the input signal of the AIN0 pin of ADC changes from a potentiometer to a combination of a photoresistor and a resistor.

Sketch_11.1_Nightlamp



Download the code to ESP32-WROVER, if you cover the photoresistor or increase the light shining on it, the brightness of the LED changes accordingly.

The following is the program code:

```
#define PIN_ANALOG_IN   4
#define PIN_LED         12
#define CHAN            0
#define LIGHT_MIN       372
#define LIGHT_MAX       2048
void setup() {
  ledcAttachChannal(PIN_LED, 1000, 12, CHAN);
}

void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN); //read adc
  // adcVal re-map to pwmVal
  int pwmVal = map(constrain(adcVal, LIGHT_MIN, LIGHT_MAX), LIGHT_MIN, LIGHT_MAX, 0, 4095);
  ledcWrite(PIN_LED, pwmVal);    // set the pulse width.
  delay(10);
}
```

Reference

constrain(amt,low,high)

```
#define constrain(amt,low,high) ((amt)<(low)? (low):((amt)>(high)? (high):(amt)))
```

Constrain the value amt between low and high.

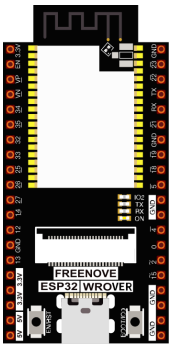
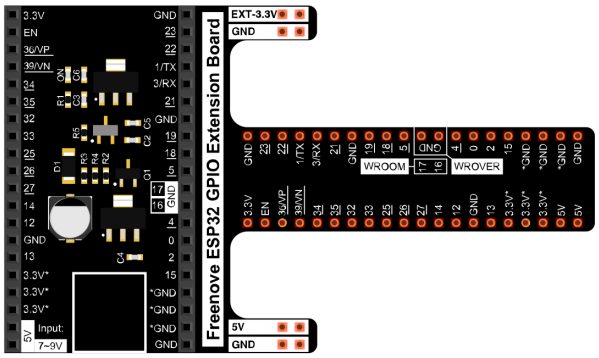
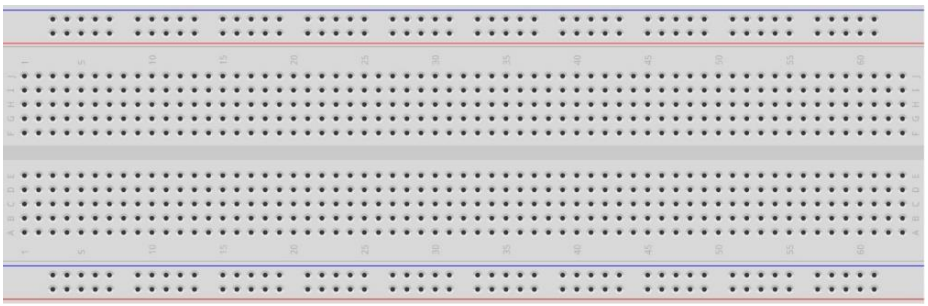
Chapter 12 Thermistor

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

Project 12.1 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.

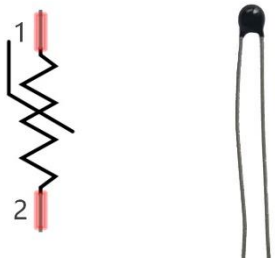
Component List

<table>
<tr><td>ESP32-WROVER 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;

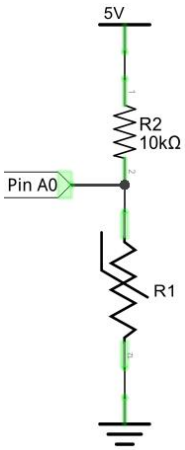
EXP[n] is nth power of E;

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.

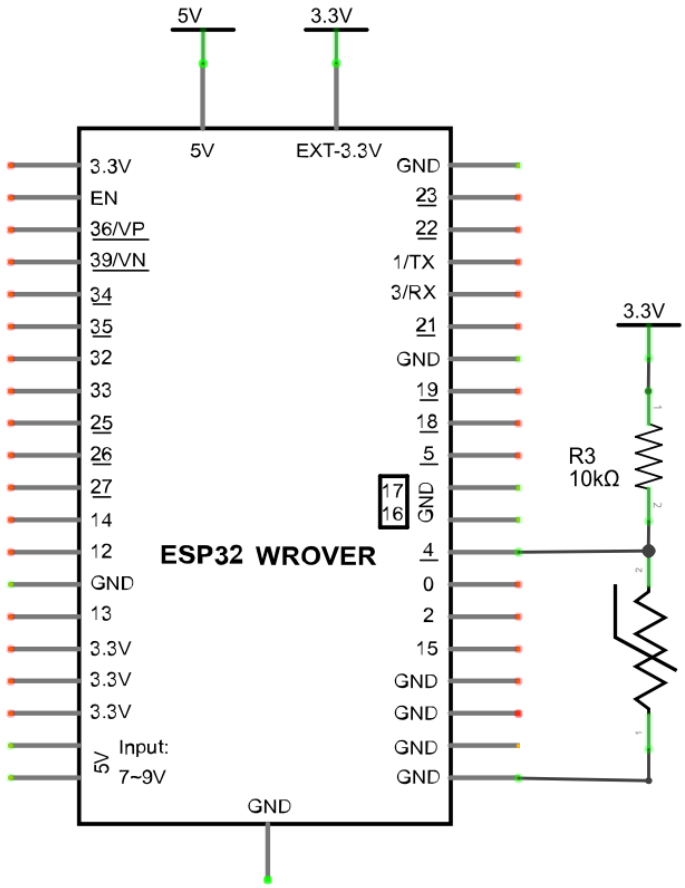
Therefore, the temperature formula can be derived as:

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

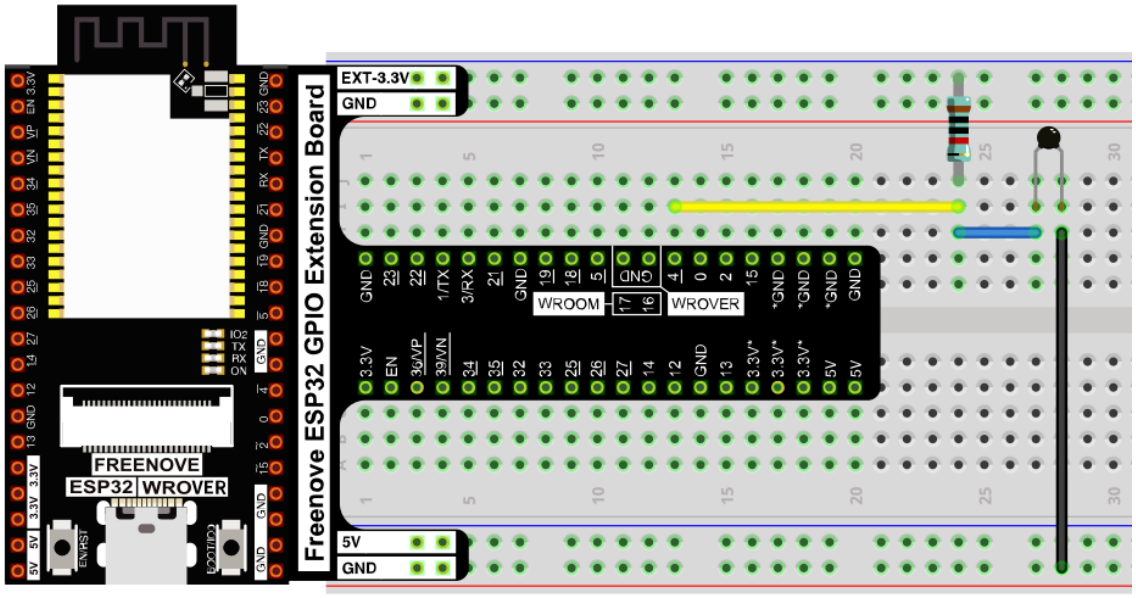
Circuit

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

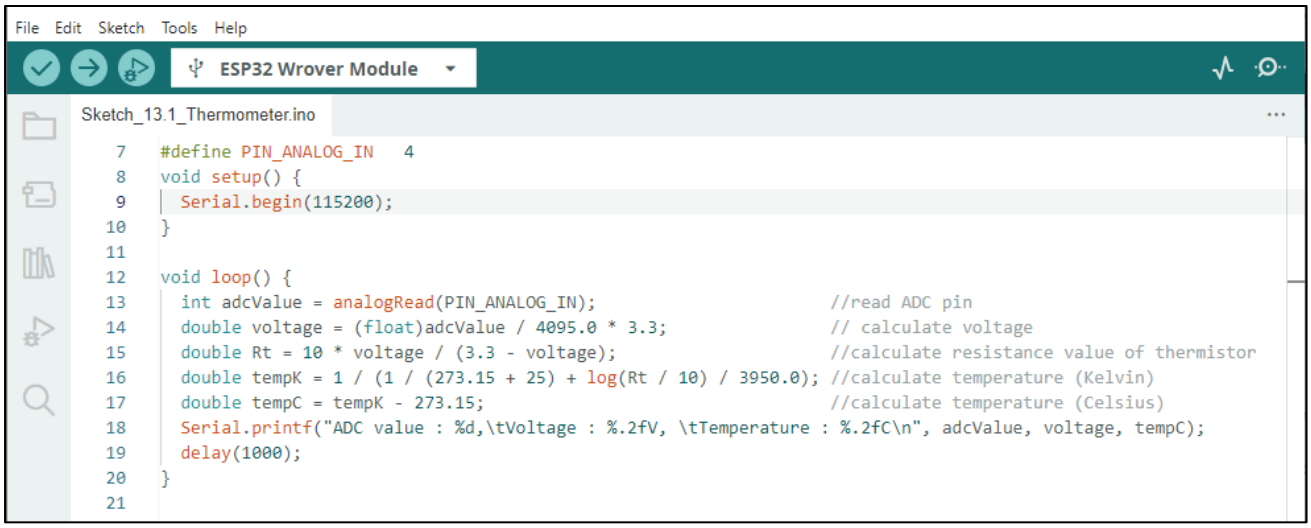


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



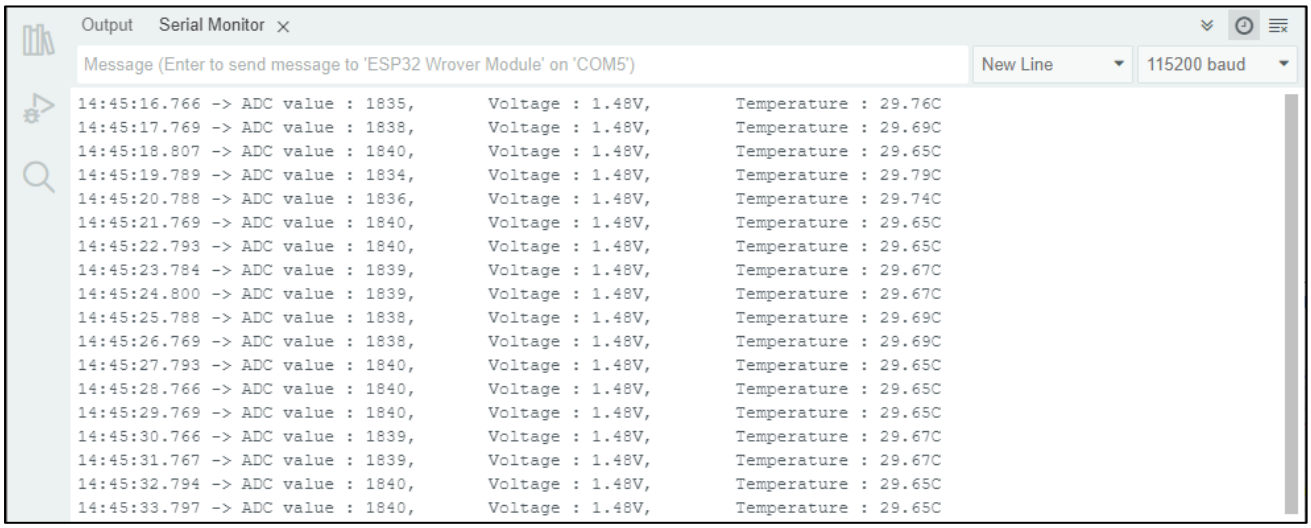
Sketch

Sketch_12.1_Thermometer



Download the code to ESP32-WROVER, the terminal window will display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

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



The following is the code:

```
#define PIN_ANALOG_IN   4
void setup() {
  Serial.begin(115200);
}

void loop() {
  int adcValue = analogRead(PIN_ANALOG_IN);        //read ADC pin
  double voltage = (float)adcValue / 4095.0 * 3.3;// calculate voltage
  double Rt = 10 * voltage / (3.3 - voltage);     //calculate resistance value of thermistor
  double tempK = 1 / (1/(273.15 + 25) + log(Rt / 10)/3950.0);//calculate temperature (Kelvin)
  double tempC = tempK - 273.15;                  //calculate temperature (Celsius)
  Serial.printf("ADC value : %d,\tVoltage : %.2fV, \tTemperature : %.2fC\n", adcValue,
voltage, tempC);
  delay(1000);
}
```

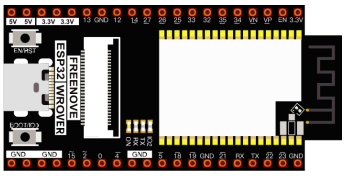
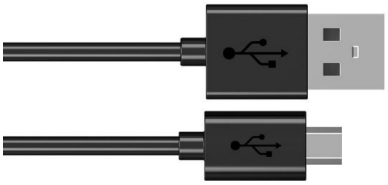
In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the thermistor, according to the formula.

Chapter 13 Bluetooth

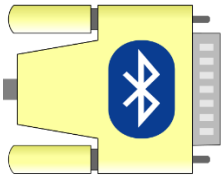
This chapter mainly introduces how to make simple data transmission through Bluetooth of ESP32-WROVER and mobile phones.
Project 13.1 is classic Bluetooth and Project 13.2 is low power Bluetooth.If you are an iPhone user, please start with Project 13.2.

Project 13.1 Bluetooth Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

In this tutorial we need to use a Bluetooth APP called Serial Bluetooth Terminal to assist in the experiment. If you've not installed it yet, please do so by clicking: https://www.appsapk.com/serial-bluetooth-terminal/ The following is its logo.



Component knowledge

ESP32'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.

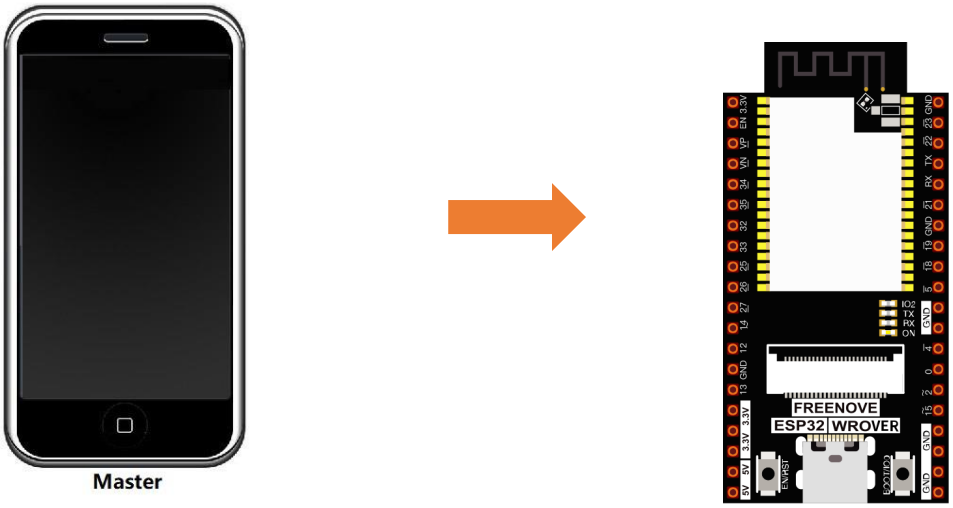
Master mode

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

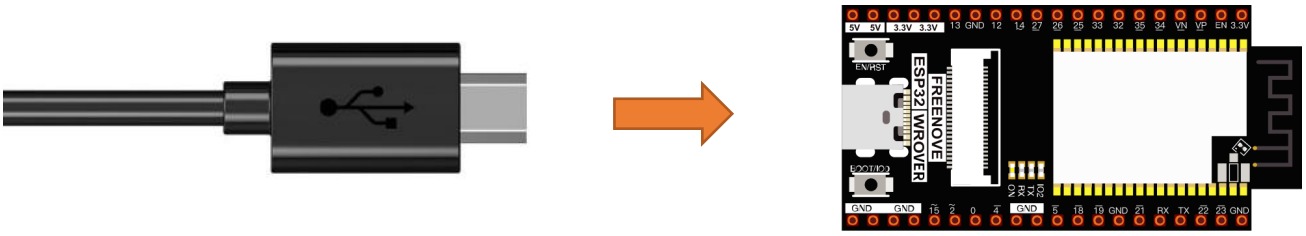
The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

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 ESP32, they are usually in master mode and ESP32 in slave mode.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_13.1_SerialToSerialBT

```
File Edit Sketch Tools Help
ESP32 Wrover Module
Sketch_27.1_SerialToSerialBT.ino
```

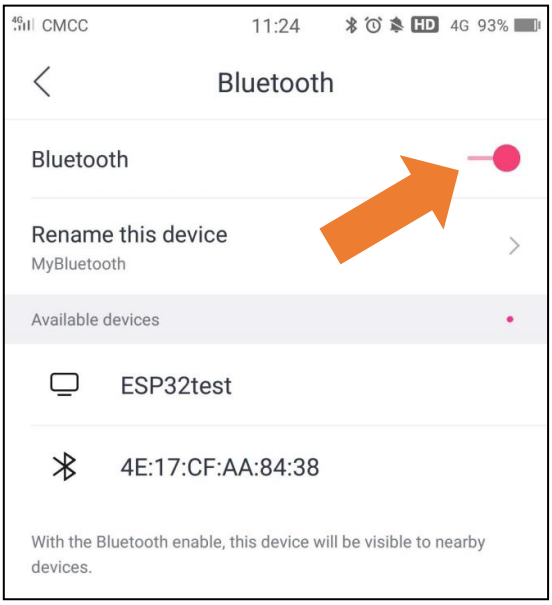
```
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;
String buffer;
void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("\nThe device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

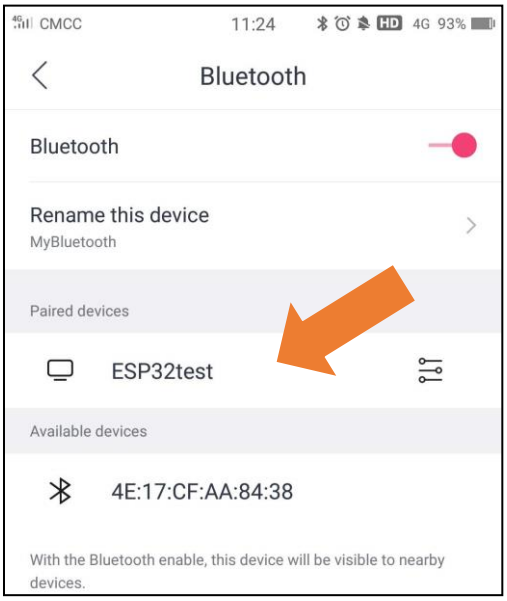
Compile and upload the code to the ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. When you see the serial printing out the character string as below, it indicates that the Bluetooth of ESP32 is ready and waiting to connect with the mobile phone.

```
11:07:06.801 -> rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
11:07:06.842 -> configsip: 0, SPIWP:0xee
11:07:06.842 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
11:07:06.842 -> mode:DIO, clock div:1
11:07:06.843 -> load:0x3fff0030,len:1448
11:07:06.843 -> load:0x40078000,len:14844
11:07:06.843 -> ho 0 tail 12 room 4
11:07:06.843 -> load:0x40080400,len:4
11:07:06.843 -> load:0x40080404,len:3356
11:07:06.843 -> entry 0x4008059c
11:07:08.127 ->
11:07:08.127 -> The device started, now you can pair it with bluetooth!
```

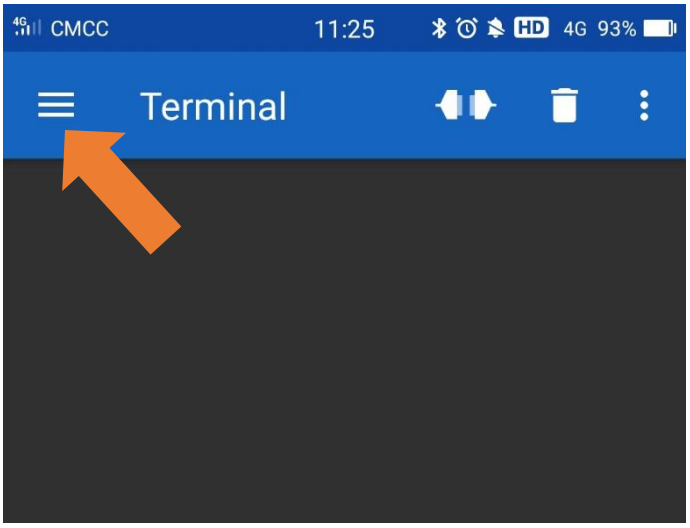
Make sure that the Bluetooth of your phone has been turned on and Serial Bluetooth Terminal has been installed.

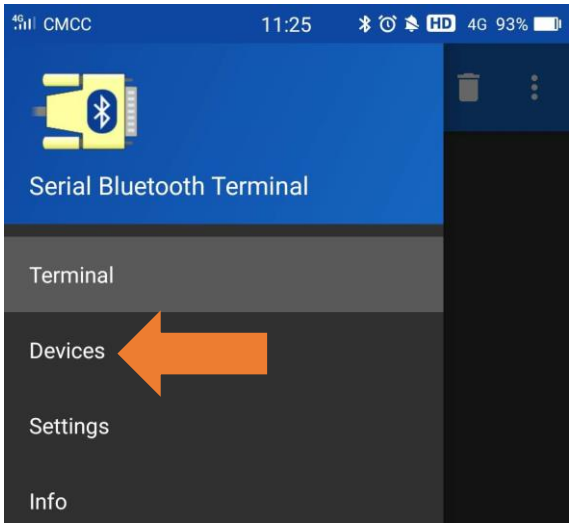
Click "Search" to search Bluetooth devices nearby and select "ESP32 test" to connect to.



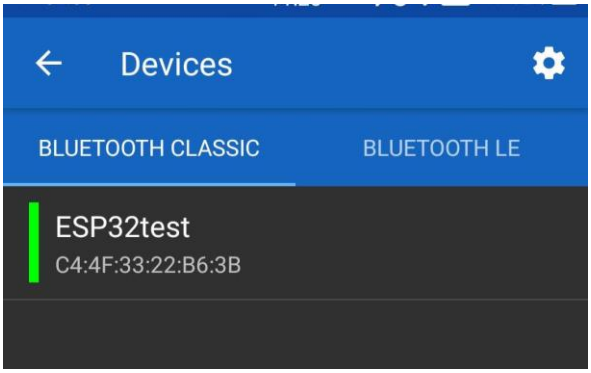


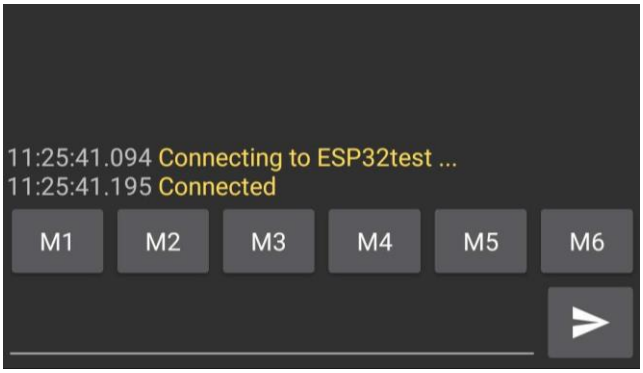
Turn on software APP, click the left of the terminal. Select "Devices"





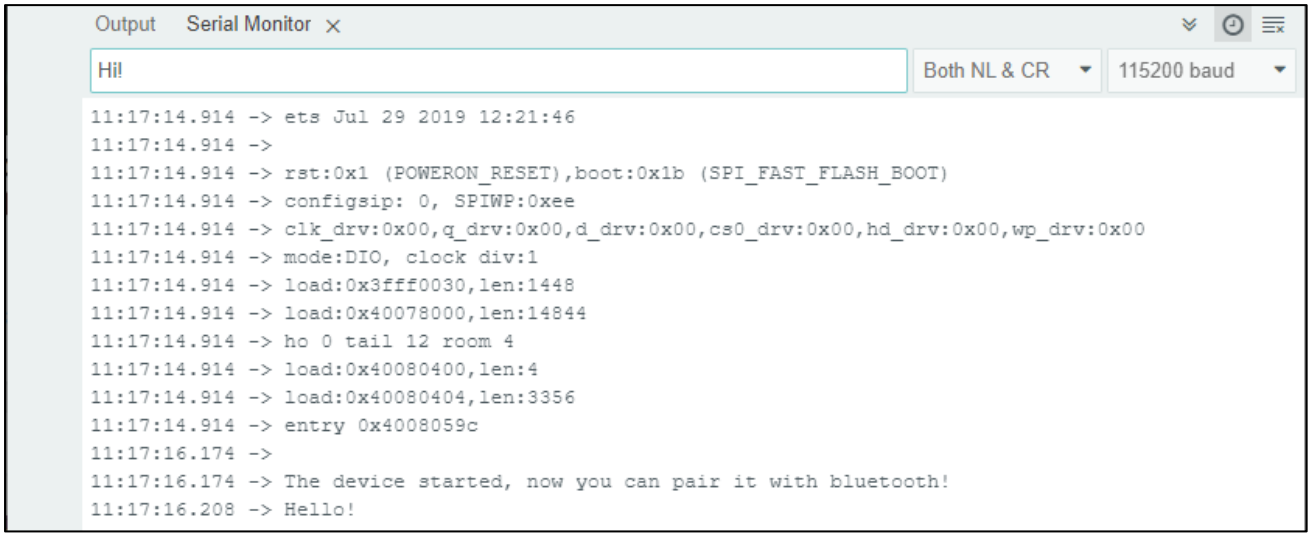
Select ESP32test in classic Bluetooth mode, and a successful connecting prompt will appear as shown on the right illustration.

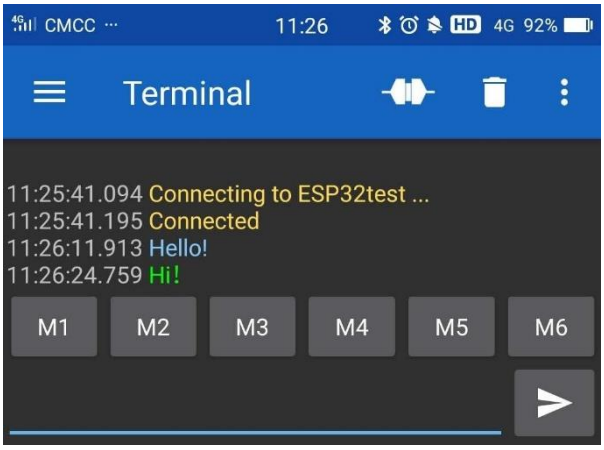




And now data can be transferred between your mobile phone and computer via ESP32-WROVER.

Send 'Hello!'' from your phone, when the computer receives it, reply "Hi" to your phone.





Reference

Class BluetoothSerial

This is a class library used to operate **BluetoothSerial**, which can directly read and set **BluetoothSerial**. Here are some member functions:

begin(localName, isMaster): Initialization function of the Bluetooth
name: name of Bluetooth module; Data type: String
isMaster: bool type, whether to set Bluetooth as Master. By default, it is false.

available(): acquire digits sent from the buffer, if not, return 0.

read(): read data from Bluetooth, data type of return value is int.

readString(): read data from Bluetooth, data type of return value is String.

write(val): send an int data val to Bluetooth.

write(str): send an Srtring data str to Bluetooth.

write(buf, len): Sends the first len data in the buf Array to Bluetooth.

setPin(const char *pin): set a four-digit Bluetooth pairing code. By default, it is 1234

connet(remoteName): connect a Bluetooth named remoteName, data type: String

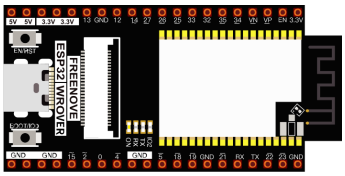
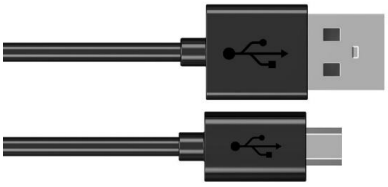
connect(remoteAddress[]): connect the physical address of Bluetooth, data type: uint8-t.

disconnect():disconnect all Bluetooth devices.

end(): disconnect all Bluetooth devices and turn off the Bluetooth, release all occupied space

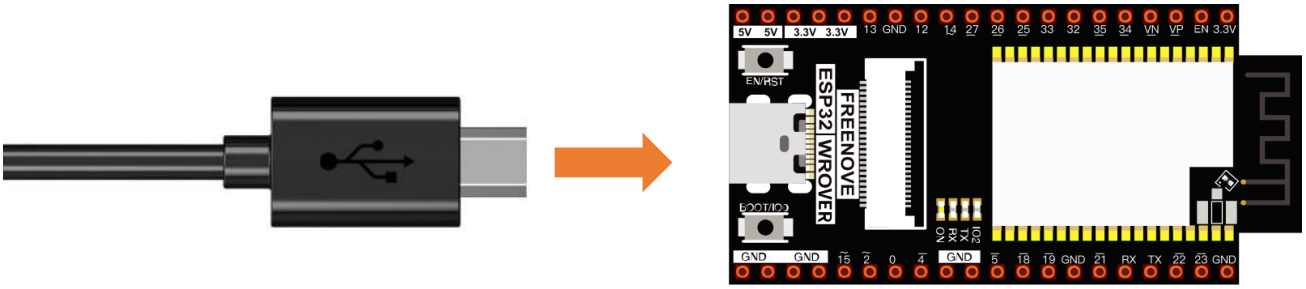
Project 13.2 Bluetooth Low Energy Data Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

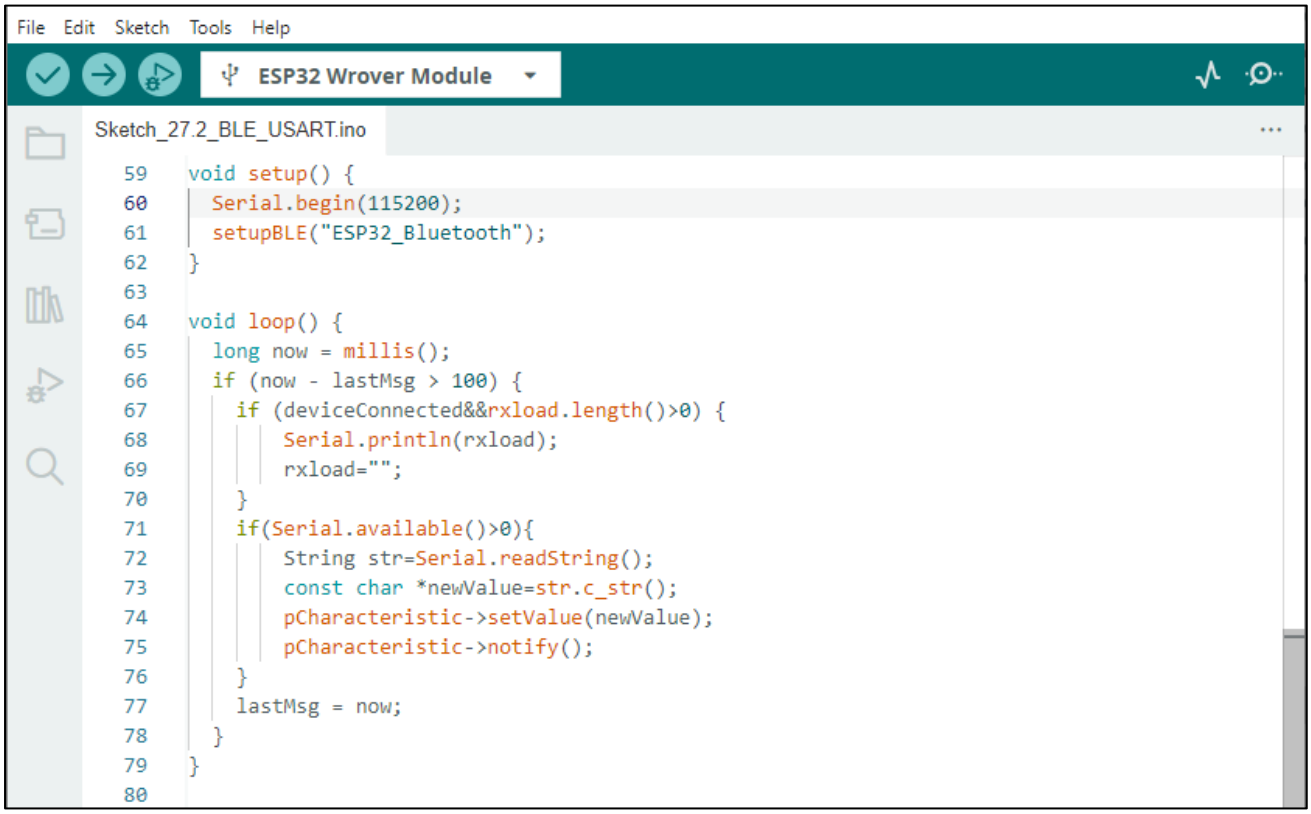
Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_13.2_BLE

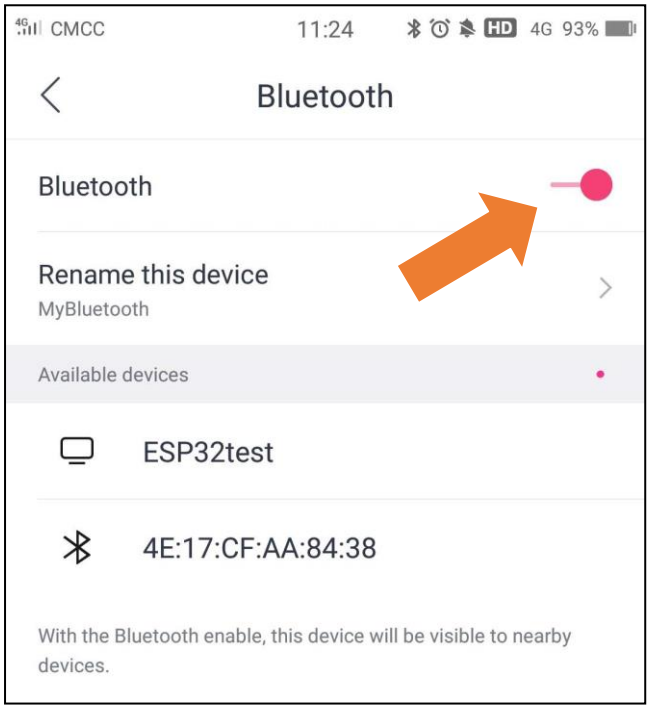


Serial Bluetooth

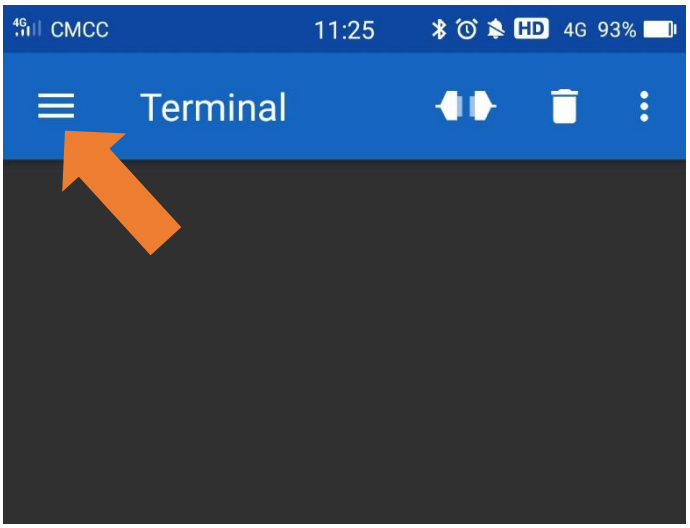
Compile and upload code to ESP32, the operation is similar to the last section.

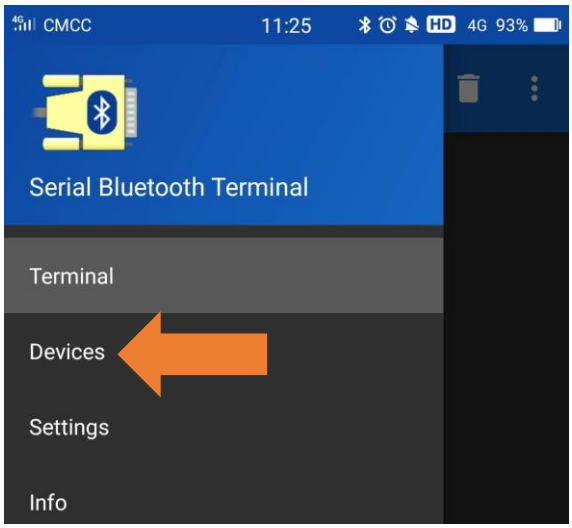
First, make sure you've turned on the mobile phone Bluetooth, and then open the software.

Click "Search" to search Bluetooth devices nearby and select "ESP32 test" to connect to.

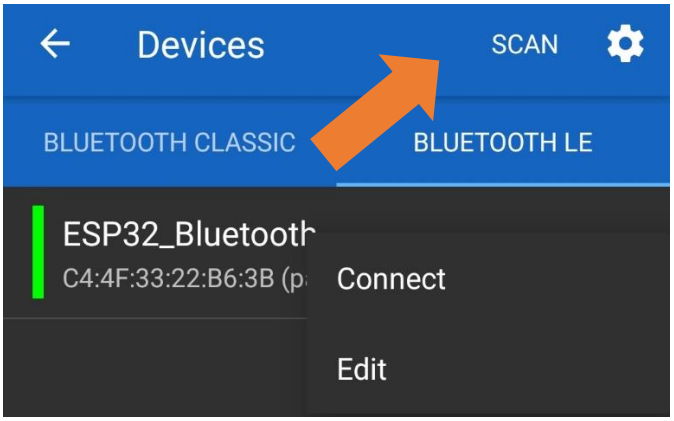


Turn on software APP, click the left of the terminal. Select "Devices"

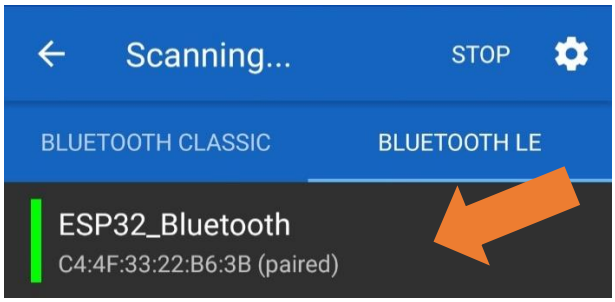




Select BLUETOOTHLE, click SCAN to scan Low Energy Bluetooth devices nearby.

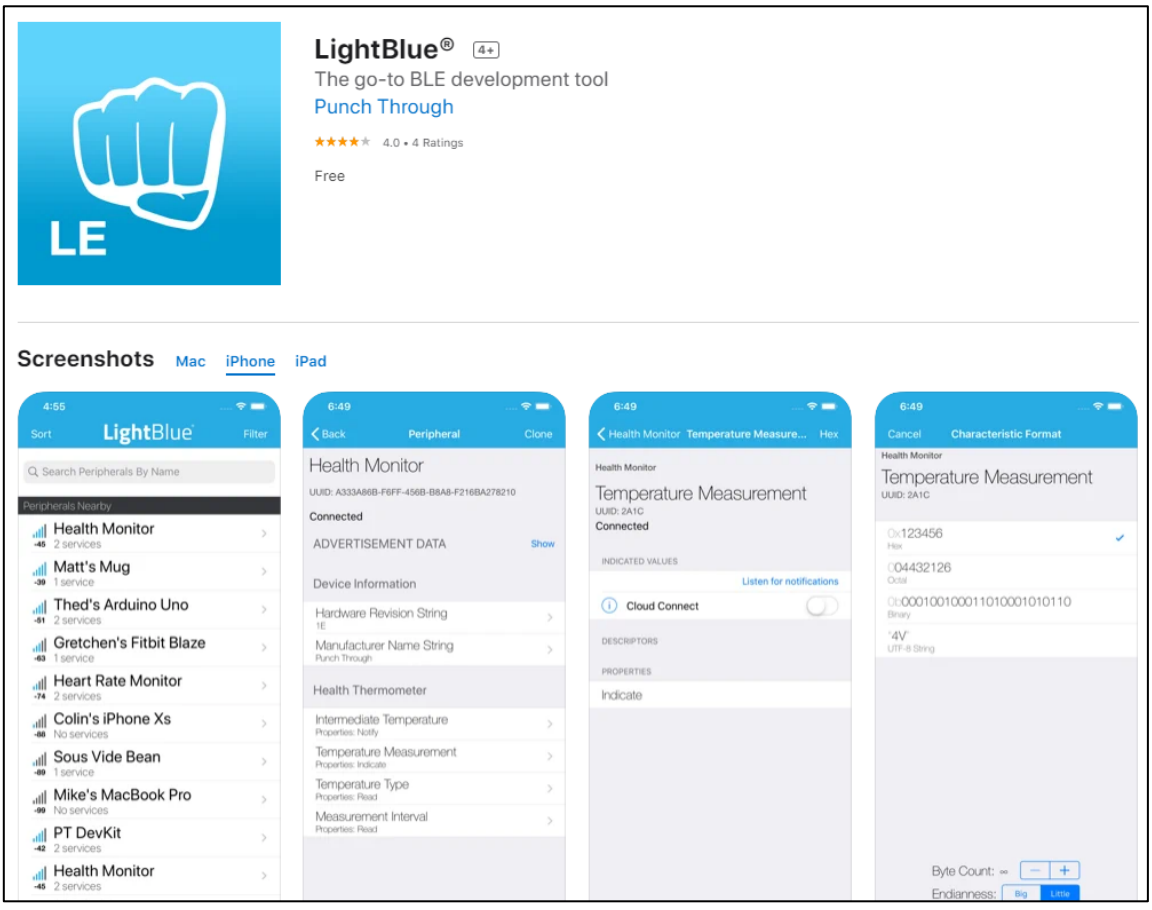


Select"ESP32-Bluetooth"



Lightblue

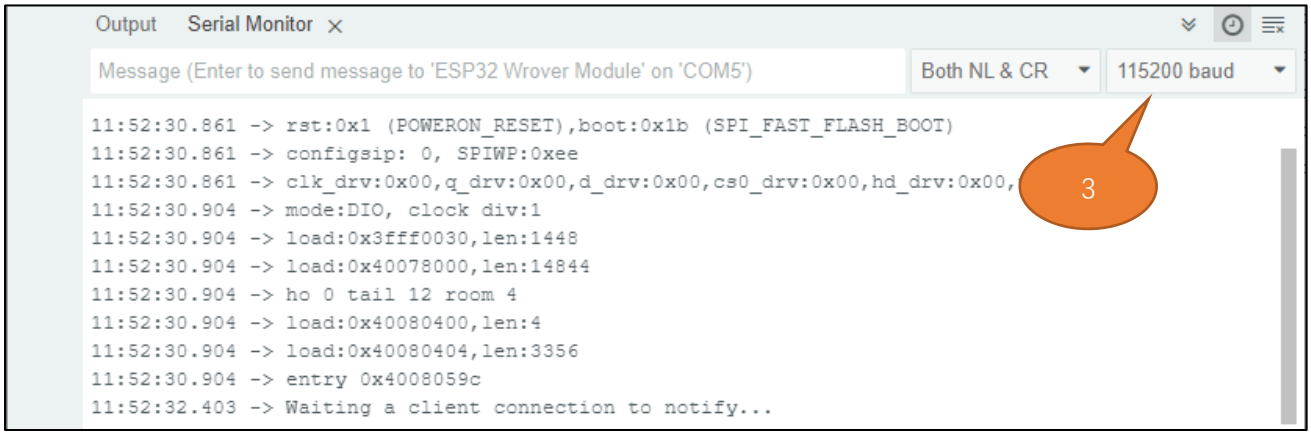
If you can't install Serial Bluetooth on your phone, try LightBlue.If you do not have this software installed on your phone, you can refer to this link:
https://apps.apple.com/us/app/lightblue/id557428110#?platform=iphone.



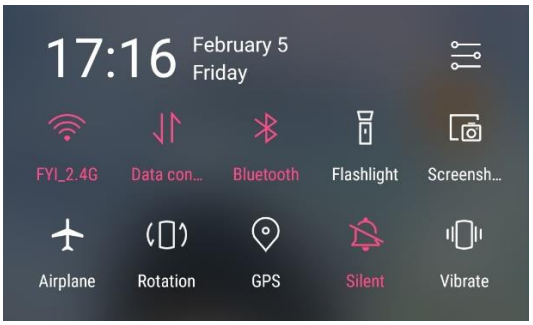
Step1. Upload the code of Project27.2 to ESP32.

Step2. Click on serial monitor.

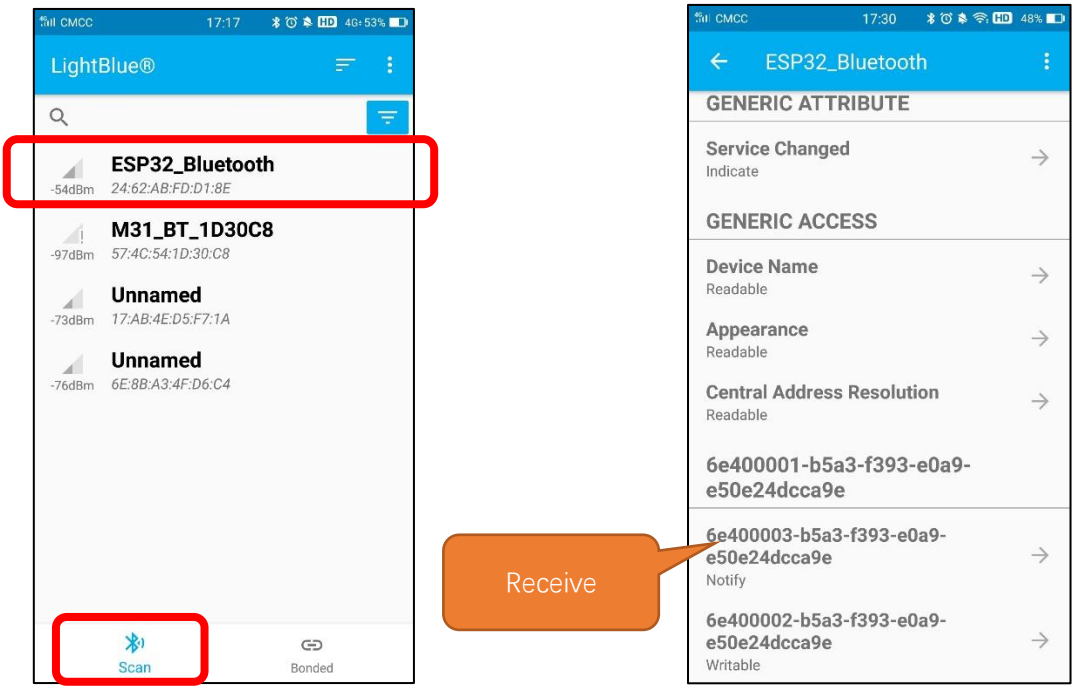
Step3. Set baud rate to 115200.



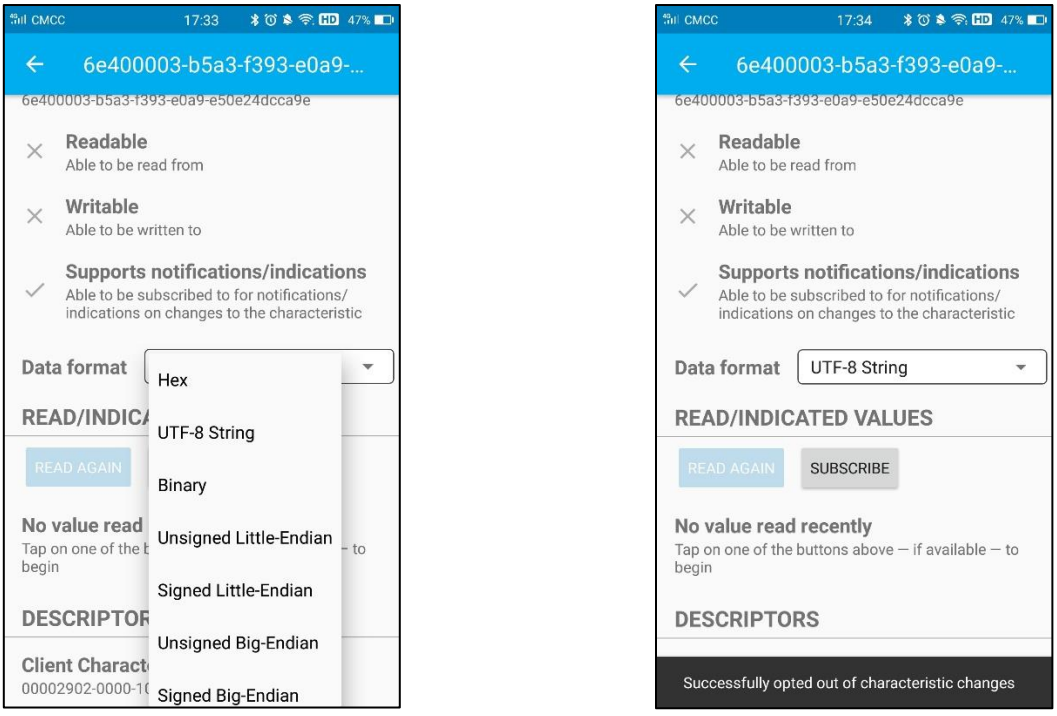
Turn ON Bluetooth on your phone, and open the Lightblue APP.



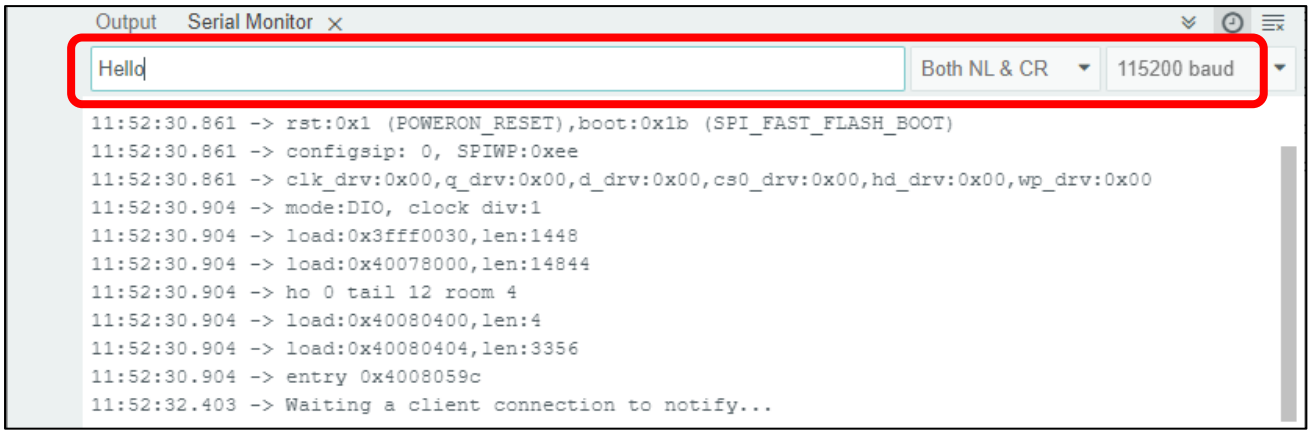
In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32_Bluetooth.



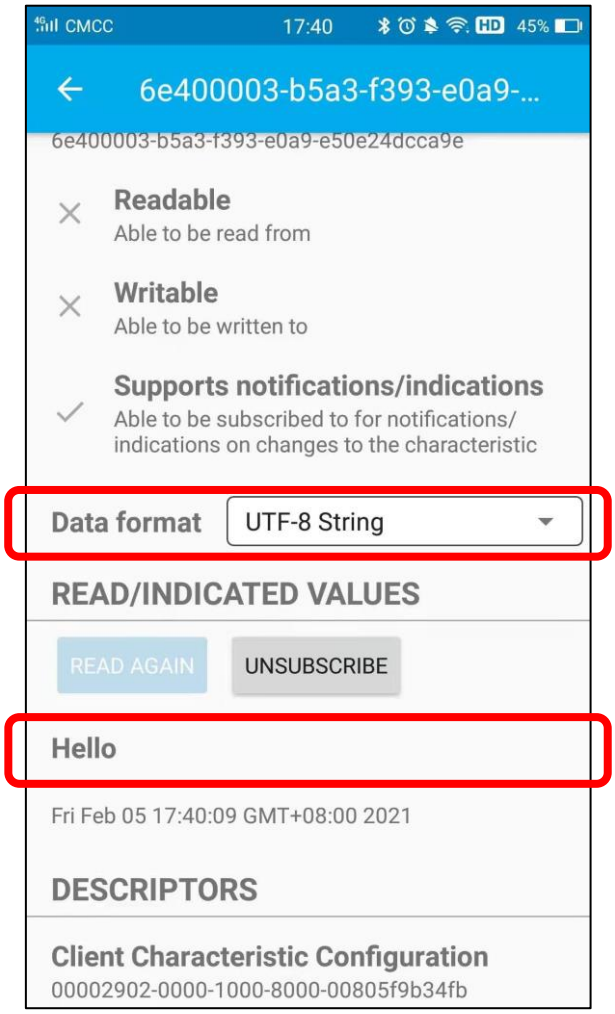
Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



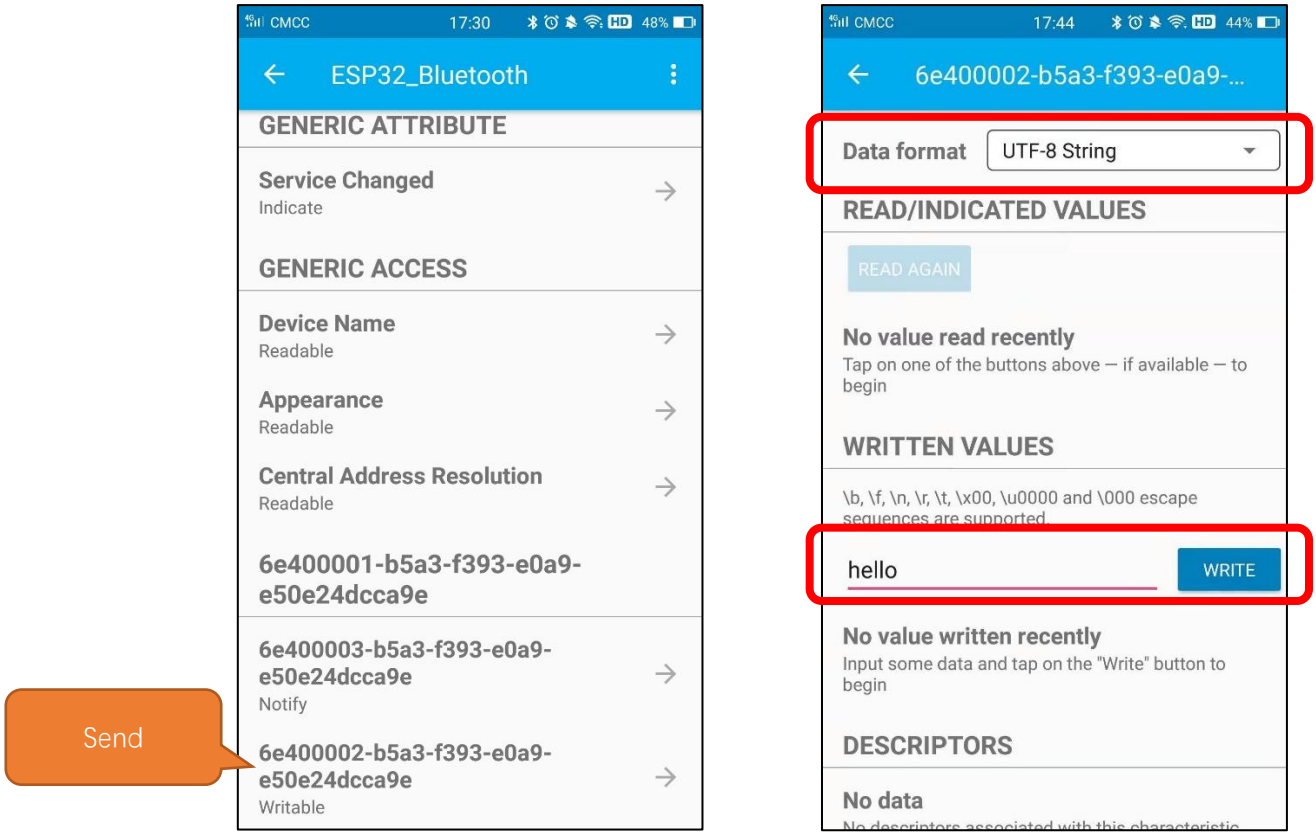
Back to the serial monitor on your computer. You can type anything in the left border of Send, and then click Send.



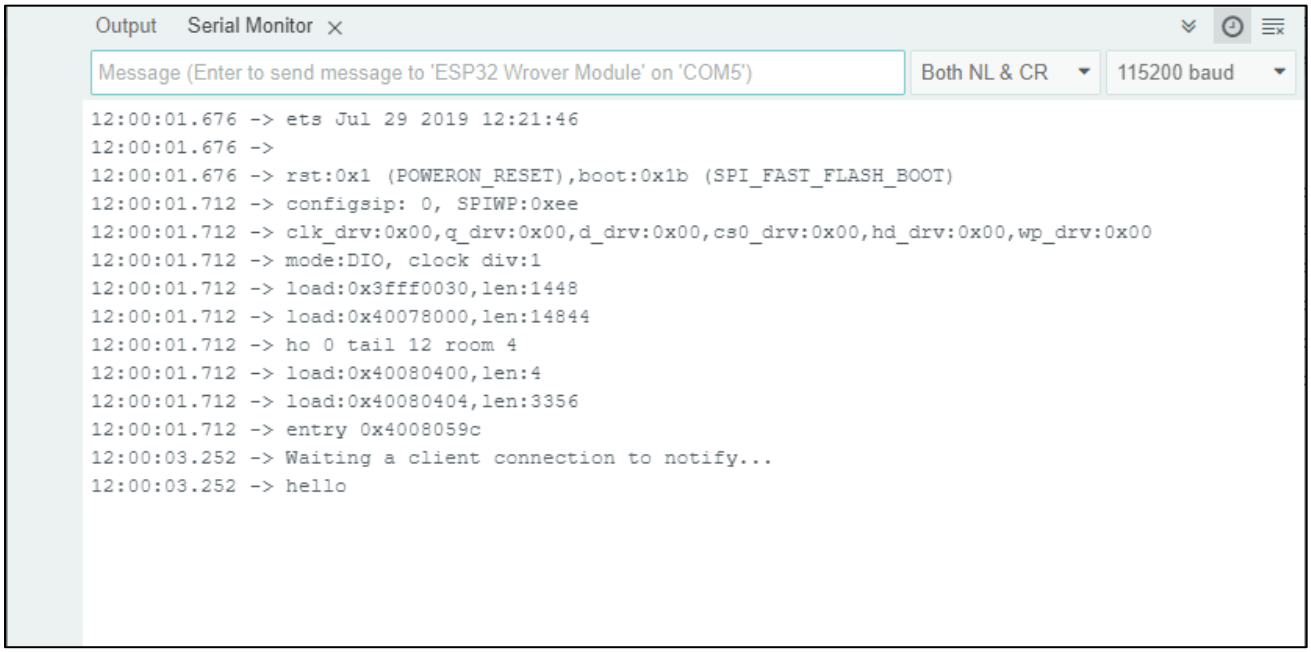
And then you can see the mobile Bluetooth has received the message.



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



And the computer will receive the message from the mobile Bluetooth.



And now data can be transferred between your mobile phone and computer via ESP32-WROVER.
The following is the program code:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <String.h>

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
uint8_t txValue = 0;
long lastMsg = 0;
String rxload="Test\n";

#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
      if (rxValue.length() > 0) {
        rxload="";
        for (int i = 0; i < rxValue.length(); i++){
          rxload +=(char)rxValue[i];
        }
      }
    }
};

void setupBLE(String BLEName){
  const char *ble_name=BLEName.c_str();
  BLEDevice::init(ble_name);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
```

```
43    BLEService *pService = pServer->createService(SERVICE_UUID);
44    pCharacteristic=
      pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
45    pCharacteristic->addDescriptor(new BLE2902());
46    BLECharacteristic *pCharacteristic =
      pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
47    pCharacteristic->setCallbacks(new MyCallbacks());
48    pService->start();
49    pServer->getAdvertising()->start();
50    Serial.println("Waiting a client connection to notify...");
51  }
52
53  void setup() {
54    Serial.begin(9600);
55    setupBLE("ESP32_Bluetooth");
56  }
57
58  void loop() {
59    long now = millis();
60    if (now - lastMsg > 1000) {
61      if (deviceConnected&&rxload.length()>0) {
62          Serial.println(rxload);
63          rxload="";
64      }
65      if(Serial.available()>0){
66          String str=Serial.readString();
67          const char *newValue=str.c_str();
68          pCharacteristic->setValue(newValue);
69          pCharacteristic->notify();
70      }
71      lastMsg = now;
72    }
73  }
```

Define the specified UUID number for BLE vendor.

```
13  #define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
14  #define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
15  #define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Write a Callback function for BLE server to manage connection of BLE.

```
17  class MyServerCallbacks: public BLEServerCallbacks {
18      void onConnect(BLEServer* pServer) {
19        deviceConnected = true;
20      };
21      void onDisconnect(BLEServer* pServer) {
22        deviceConnected = false;
23      }
24  };
```

Write Callback function with BLE features. When it is called, as the mobile terminal send data to ESP32, it will store them into reload.

```
26  class MyCallbacks: public BLECharacteristicCallbacks {
27      void onWrite(BLECharacteristic *pCharacteristic) {
28        std::string rxValue = pCharacteristic->getValue();
29        if (rxValue.length() > 0) {
30          rxload="";
31          for (int i = 0; i < rxValue.length(); i++){
32            rxload +=(char)rxValue[i];
33          }
34        }
35      }
36  };
```

Initialize the BLE function and name it.

```
55  setupBLE("ESP32_Bluetooth");
```

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

```
59    long now = millis();
60    if (now - lastMsg > 1000) {
61      if (deviceConnected&&rxload.length()>0) {
62          Serial.println(rxload);
63          rxload="";
64      }
65      if(Serial.available()>0){
66          String str=Serial.readString();
67          const char *newValue=str.c_str();
68          pCharacteristic->setValue(newValue);
69          pCharacteristic->notify();
70      }
71      lastMsg = now;
72    }
```

The design for creating the BLE server is:

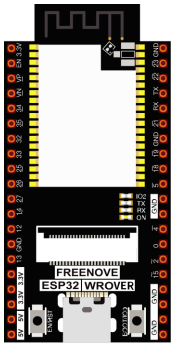
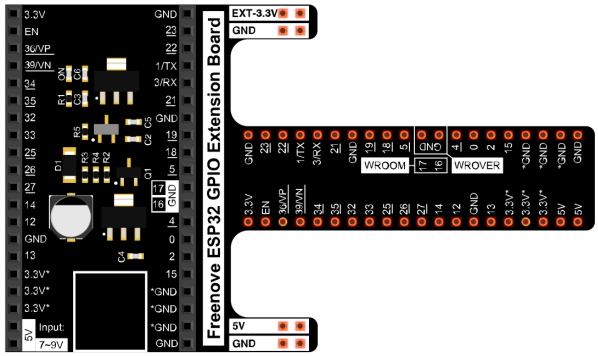
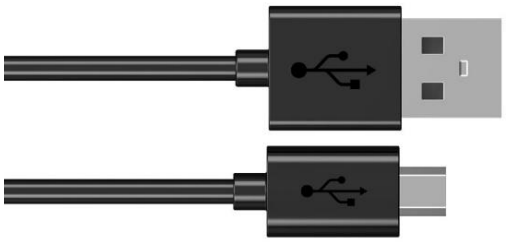
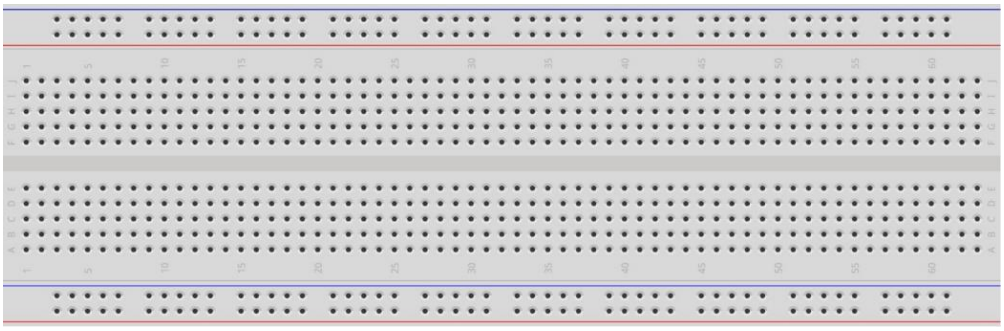
1. Create a BLE Server
2. Create a BLE Service
3. Create a BLE Characteristic on the Service
4. Create a BLE Descriptor on the characteristic
5. Start the service.
6. Start advertising.

```
38  void setupBLE(String BLEName){
39    const char *ble_name=BLEName.c_str();
40    BLEDevice::init(ble_name);
41    BLEServer *pServer = BLEDevice::createServer();
42    pServer->setCallbacks(new MyServerCallbacks());
43    BLEService *pService = pServer->createService(SERVICE_UUID);
44    pCharacteristic=
      pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
45    pCharacteristic->addDescriptor(new BLE2902());
46    BLECharacteristic *pCharacteristic =
      pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
47    pCharacteristic->setCallbacks(new MyCallbacks());
48    pService->start();
49    pServer->getAdvertising()->start();
50    Serial.println("Waiting a client connection to notify...");
51  }
```

Project 13.3 Bluetooth Control LED

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

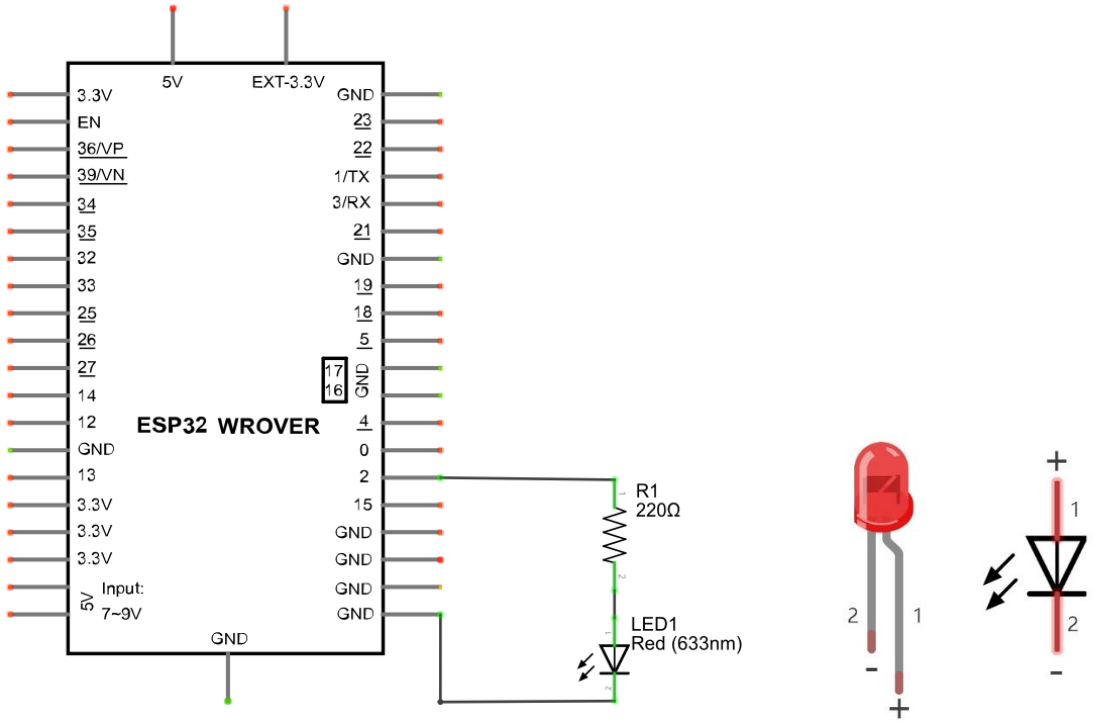
Component List

<table>
<tr><td>ESP32-WROVER 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>

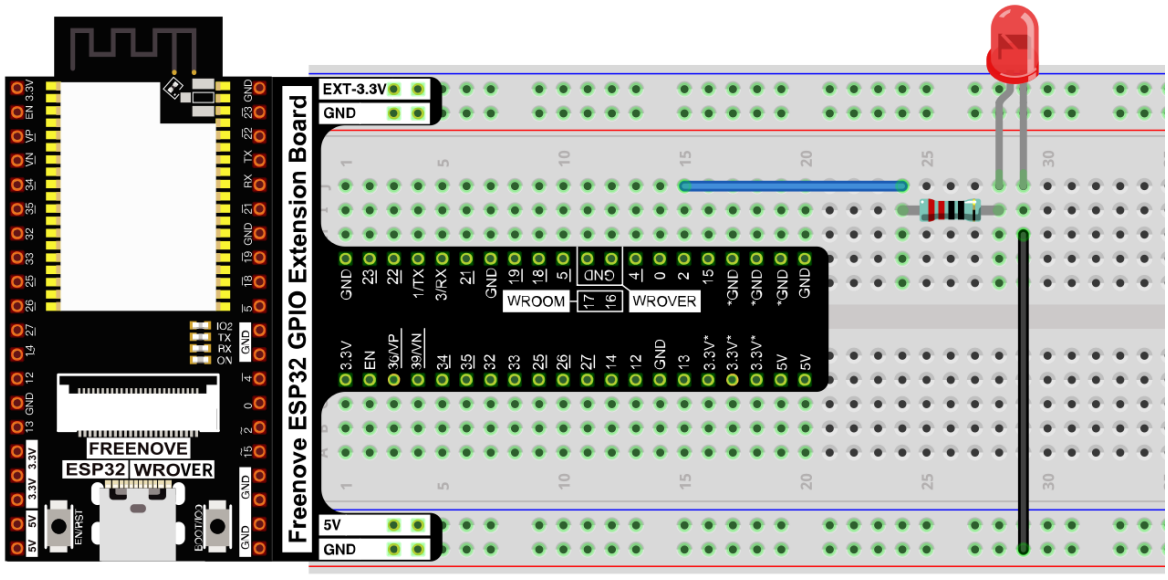
Circuit

Connect Freenove ESP32 to the computer using a USB cable.

Schematic diagram



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

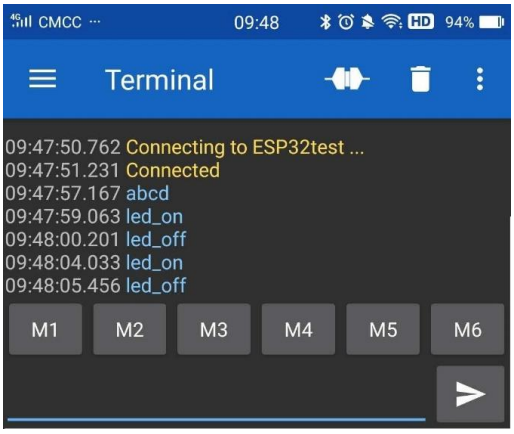


Sketch

Sketch_13.3_Bluetooth_Control_LED

```
#include "BluetoothSerial.h"
#include "string.h"
#define LED 2
BluetoothSerial SerialBT;
char buffer[20];
static int count = 0;
void setup() {
  pinMode(LED, OUTPUT);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.begin(115200);
  Serial.println("\nThe device started, now you can pair it with bluetooth!");
}

void loop() {
  while(SerialBT.available())
  {
    buffer[count] = SerialBT.read();
    count++;
  }
  if(count>0){
    Serial.print(buffer);
    if(strncmp(buffer,"led_on",6)==0){
      digitalWrite(LED,LOW);
    }
    if(strncmp(buffer,"led_off",7)==0){
      digitalWrite(LED,HIGH);
    }
    count=0;
    memset(buffer,0,20);
  }
}
```

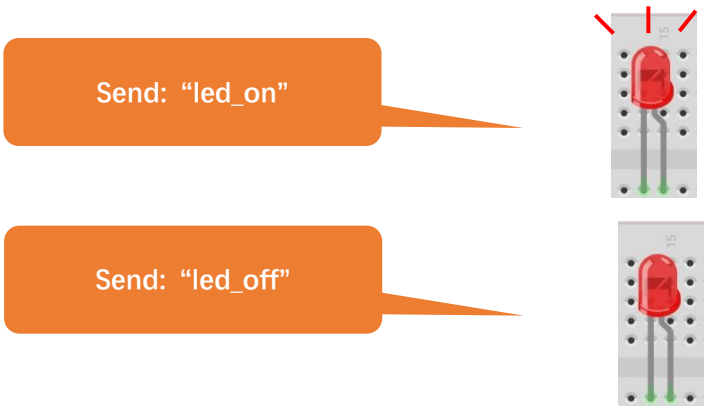
Compile and upload code to ESP32. The operation of the APP is the same as 27.1, you only need to change the sending content to "**led_on**" and "**led_off**" to operate LEDs on the ESP32-WROVER.
Data sent from mobile APP:



Display on the serial port of the computer:

```
Output  Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')   Both NL & CR   115200 baud
13:51:42.125 -> ets Jul 29 2019 12:21:46
13:51:42.125 ->
13:51:42.125 -> rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
13:51:42.157 -> configsip: 0, SPIWP:0xee
13:51:42.157 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
13:51:42.157 -> mode:DIO, clock div:1
13:51:42.157 -> load:0x3fff0030,len:1448
13:51:42.157 -> load:0x40078000,len:14844
13:51:42.157 -> ho 0 tail 12 room 4
13:51:42.157 -> load:0x40080400,len:4
13:51:42.157 -> load:0x40080404,len:3356
13:51:42.157 -> entry 0x4008059c
13:51:43.439 ->
13:51:43.439 -> The device started, now you can pair it with bluetooth!
13:51:59.612 -> led_on
13:52:02.181 -> led_off
13:52:05.056 -> led_on
13:52:07.429 -> led_off
```

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:

```
#include "BluetoothSerial.h"
#include "string.h"
#define LED 2
BluetoothSerial SerialBT;
char buffer[20];
static int count = 0;
void setup() {
  pinMode(LED, OUTPUT);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.begin(115200);
  Serial.println("\The device started, now you can pair it with Bluetooth! ");
}

void loop() {
  while(SerialBT.available())
  {
    buffer[count] = SerialBT.read();
    count++;
  }
  if(count>0){
    Serial.print(buffer);
    if(strncmp(buffer,"led_on",6)==0){
      digitalWrite(LED,LOW);
    }
    if(strncmp(buffer,"led_off",7)==0){
      digitalWrite(LED,HIGH);
    }
    count=0;
    memset(buffer,0,20);
  }
}
```

Use character string to handle function header file.

```
#include "string.h"
```

Define a buffer to receive data from Bluetooth, and use "count" to record the bytes of data received.

```
char buffer[20];
static int count = 0;
```

Initialize the classic Bluetooth and name it as "ESP32test"

```
SerialBT.begin("ESP32test"); //Bluetooth device name
```

When receive data, read the Bluetooth data and store it into buffer array.

```
15    while(SerialBT.available()){
16      buffer[count] = SerialBT.read();
17      count++;
18    }
```

Compare the content in buffer array with "led_on" and "led_off" to see whether they are the same. If yes, execute the corresponding operation.

```
22      if(strncmp(buffer,"led_on",6)==0){
23        digitalWrite(LED,LOW);
24      }
25      if(strncmp(buffer,"led_off",7)==0){
26        digitalWrite(LED,HIGH);
27      }
```

After comparing the content of array, to ensure successful transmission next time, please empty the array and set the count to zero.

```
28      count=0;
29      memset(buffer,0,20);
```

Reference

strncmp() functions are often used for string comparisons, which are accurate and stable.

int strncmp(const char *str1, const char *str2, size_t n)

str1: the first string to be compared
str2: the second string to be compared
n: the biggest string to be compared
Return value: if stir1>str2, then return value>0.
If return value is 0, then the contents of str1 and str2 are the same.
If str1< str2, then return value<0.

Function memset is mainly used to clean and initialize the memory of array

void *memset(void *s, int c, unsigned long n)

Function memset() is to set the content of a certain internal storage as specified value.
*s: the initial address of the content to clear out.
c:to be replaced as specified value
n: the number of byte to be replaced

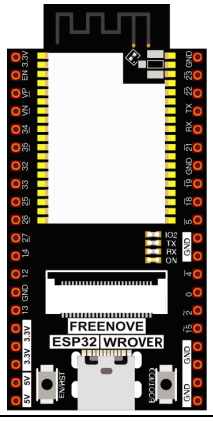
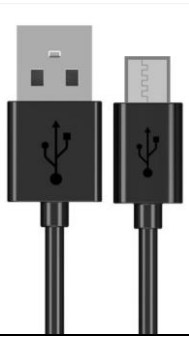
Chapter 14 Read and Write the Sdcard

Note: The SD card chapter only applies to the ESP32 WROVER development board with an SD card slot on the back. If your ESP32 WROVER does not have an SD card slot on the back, please skip this chapter.

An SDcard slot is integrated on the back of the ESP32 WROVER. In this chapter we learn how to use ESP32 to read and write SDcard.

Project 14.1 SDMMC Test

Component List

ESP32 WROVER x1 	USB cable x1 	SDcard reader x1 (random color)  (Not a USB flash drive.)	SDcard x1 

Component knowledge

SD card read and write method

ESP32 has two ways to use SD card, one is to use the SPI interface to access the SD card, and the other is to use the SDMMC interface to access the SD card. SPI mode uses 4 IOs to access SD card. The SDMMC has one-bit bus mode and four-bit bus mode. In one-bit bus mode, SDMMC use 3 IOs to access SD card. In four-bit bus mode, SDMMC uses 6 IOs to access the SD card.

The above three methods can all be used to access the SD card, the difference is that the access speed is different.

In the four-bit bus mode of SDMMC, the reading and writing speed of accessing the SD card is the fastest. In the one-bit bus mode of SDMMC, the access speed is about 80% of the four-bit bus mode. The access speed of SPI is the slowest, which is about 50% of the four-bit bus mode of SDMMC.

Usually, we recommend using the one-bit bus mode to access the SD card, because in this mode, we only need to use the least pin IO to access the SD card with good performance and speed.

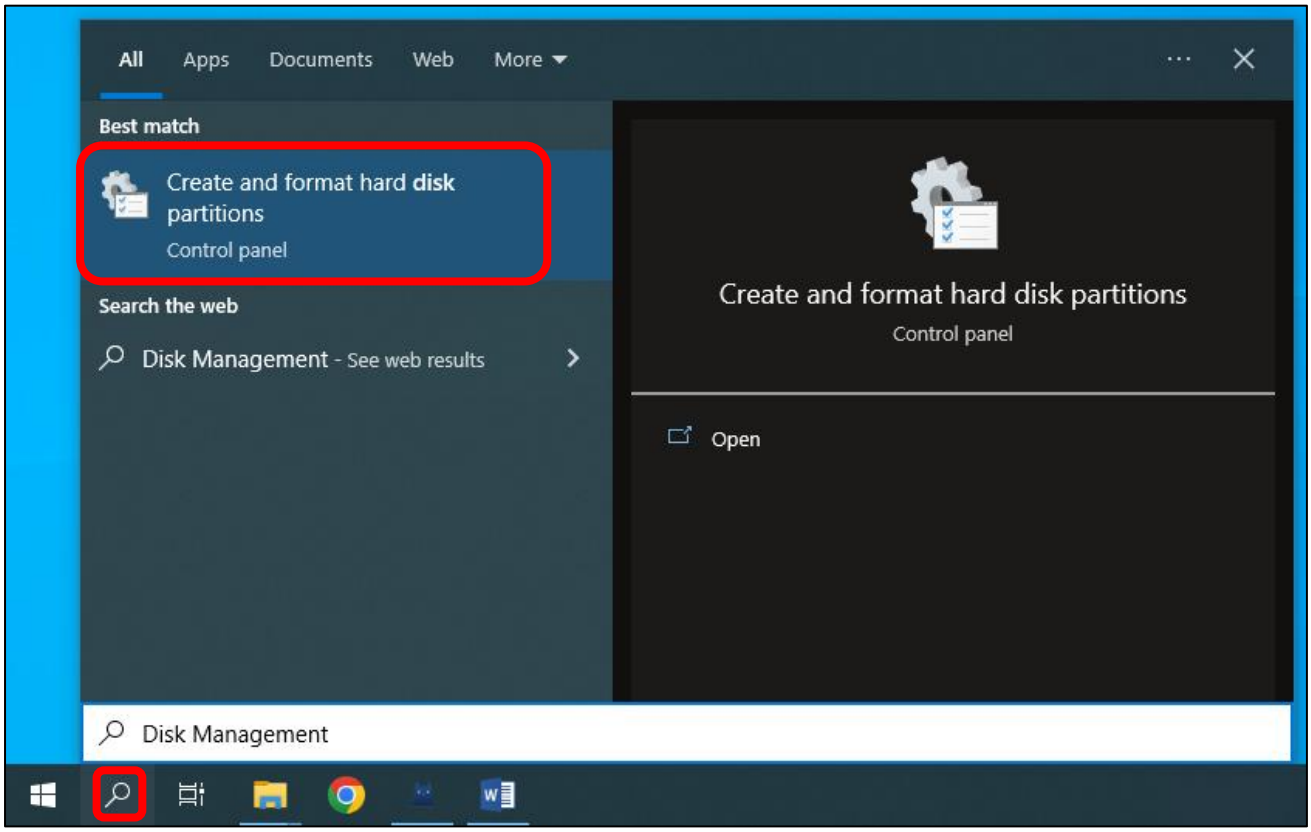
Format SD card

Before starting the tutorial, we need to create a drive letter for the blank SD card and format it. This step requires a card reader and SD card. Please prepare them in advance. Below we will guide you to do it on different computer systems. You can choose the guide that matches your computer.

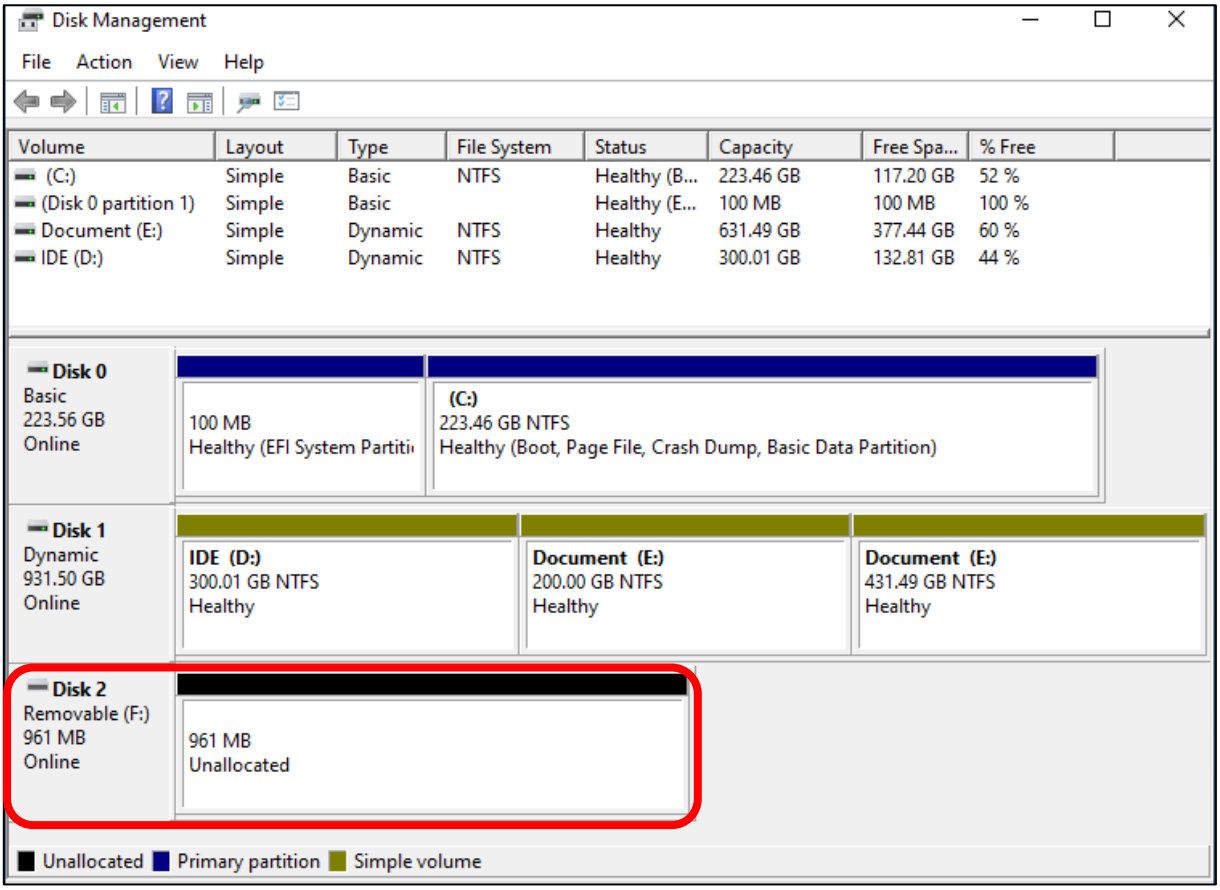
Windows

Insert the SD card into the card reader, then insert the card reader into the computer.

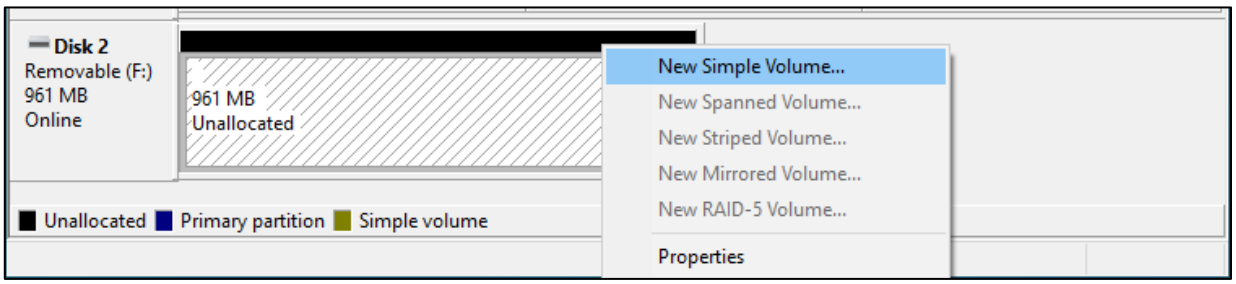
In the Windows search box, enter "Disk Management" and select "Create and format hard disk partitions".



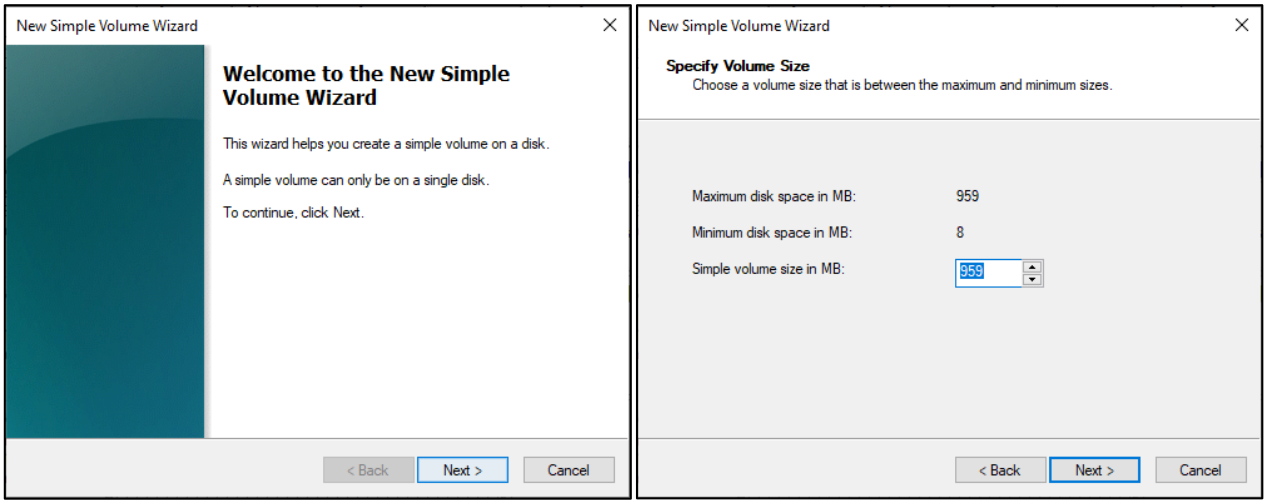
In the new pop-up window, find an unallocated volume close to 1G in size.



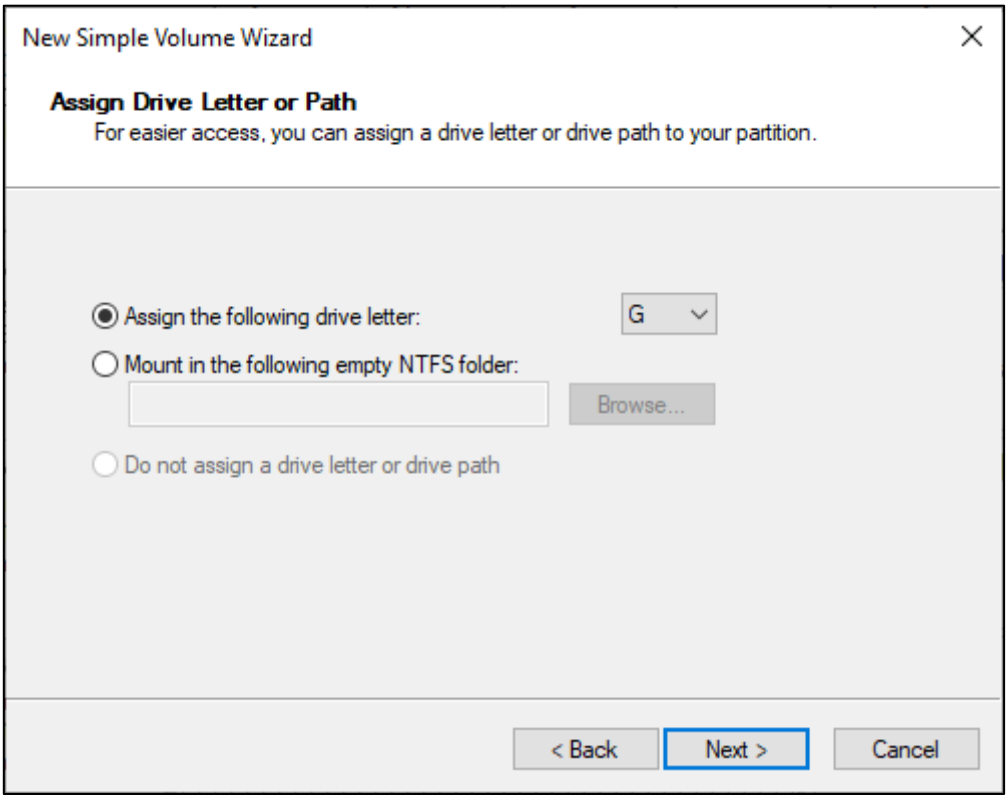
Click to select the volume, right-click and select "New Simple Volume".



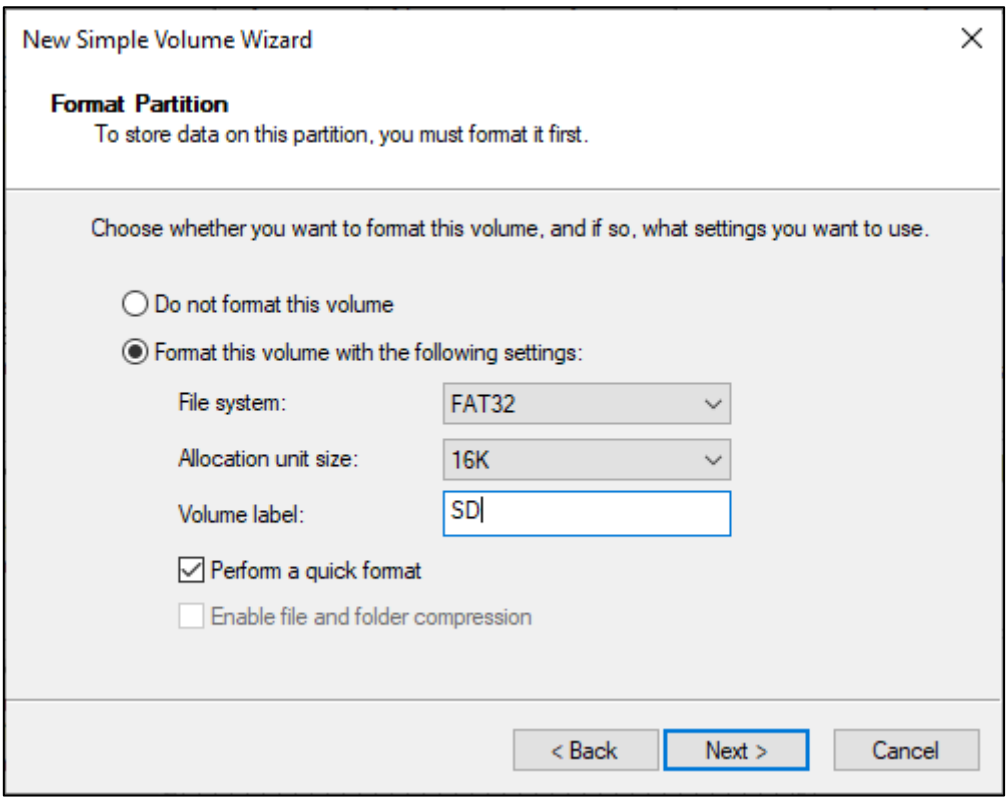
Click Next.



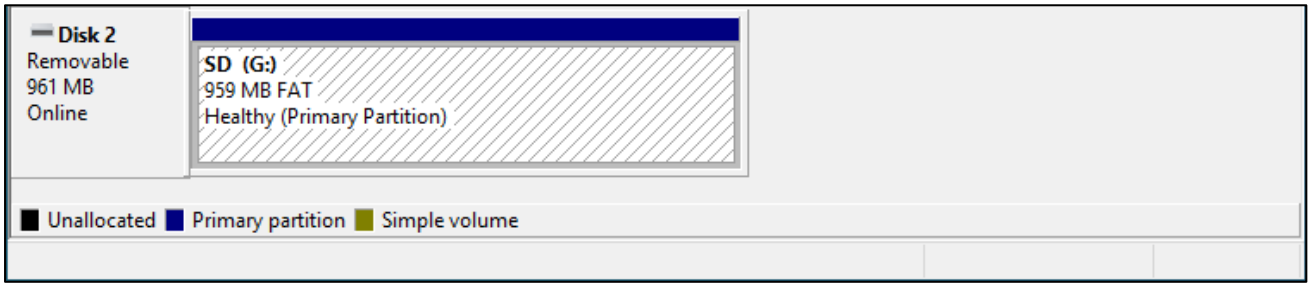
You can choose the drive letter on the right, or you can choose the default. By default, just click Next.



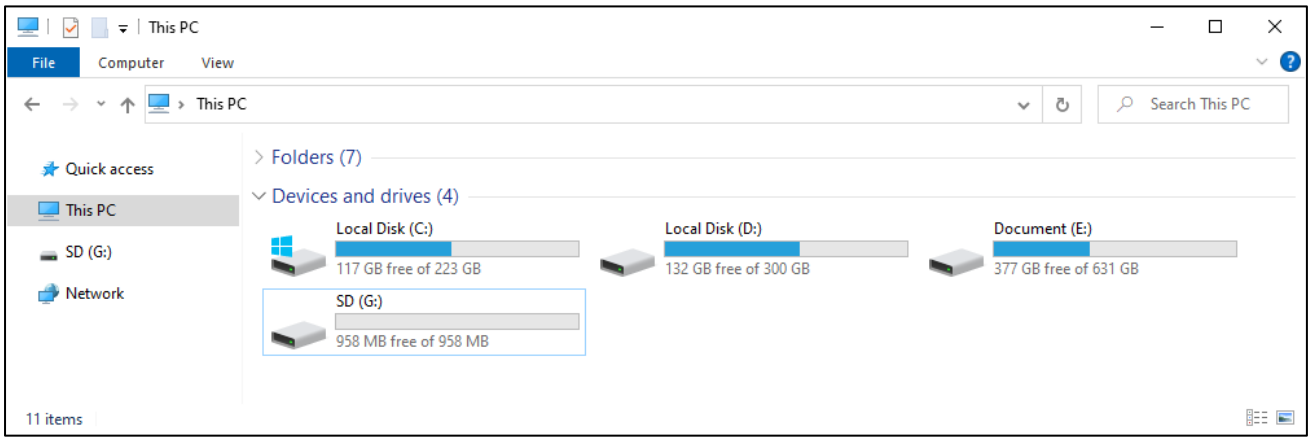
File system is FAT(or FAT32). The Allocation unit size is 16K, and the Volume label can be set to any name. After setting, click Next.



Click Finish. Wait for the SD card initialization to complete.

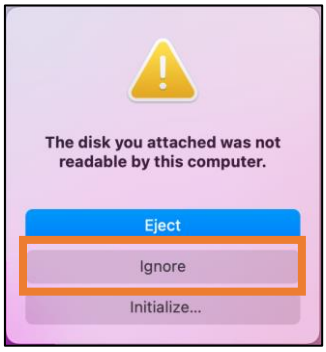


At this point, you can see the SD card in This PC.

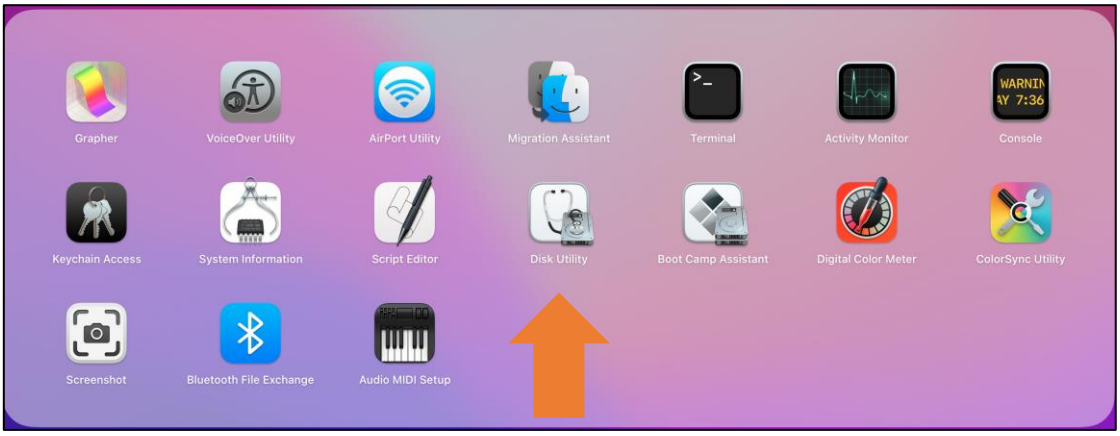


MAC

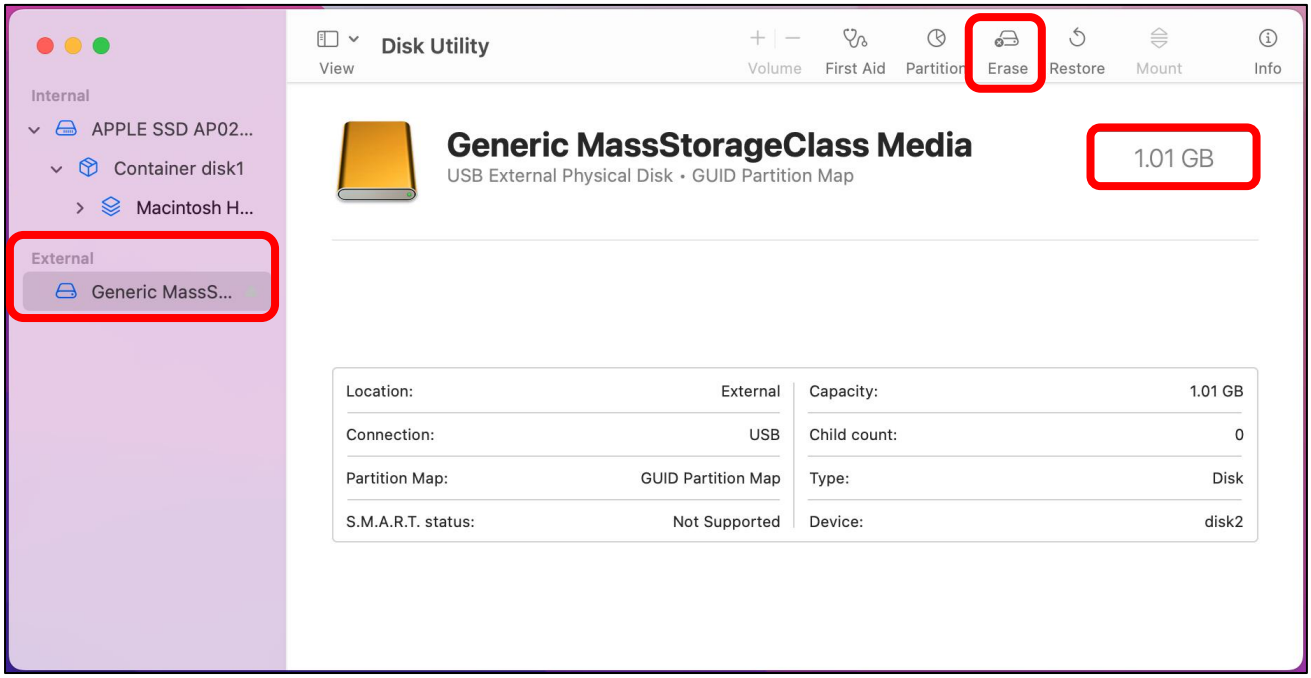
Insert the SD card into the card reader, then insert the card reader into the computer. Some computers will prompt the following information, please click to ignore it.



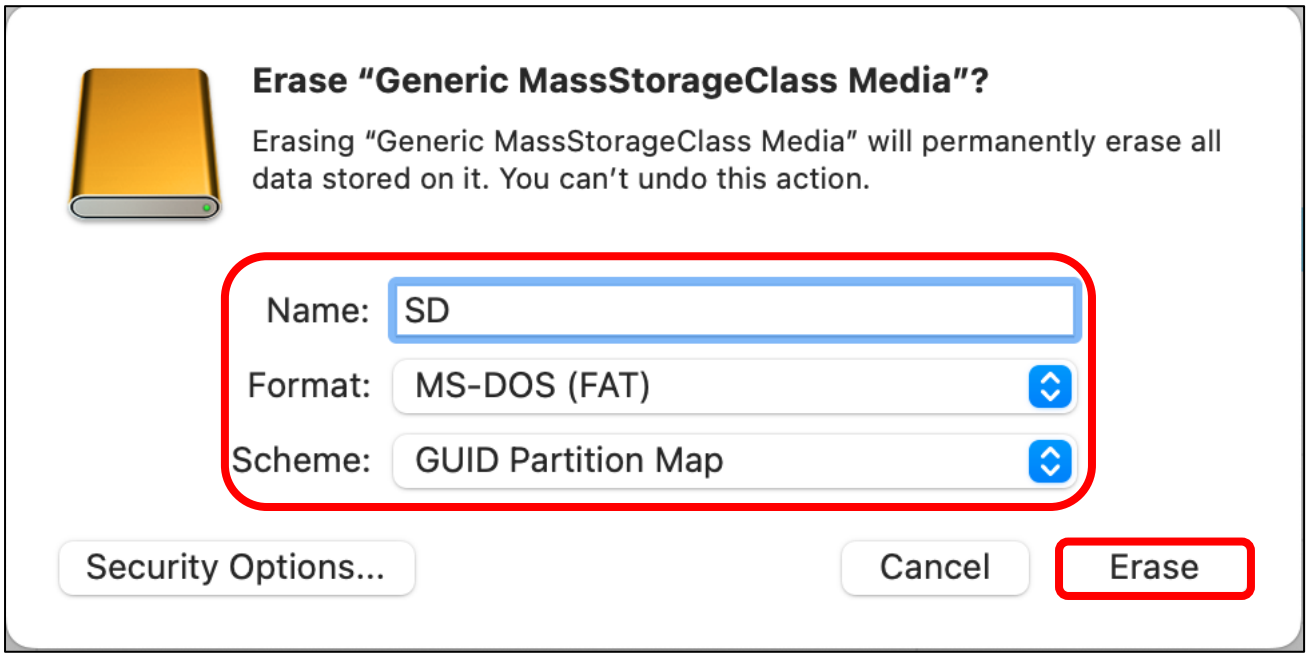
Find "Disk Utility" in the MAC system and click to open it.



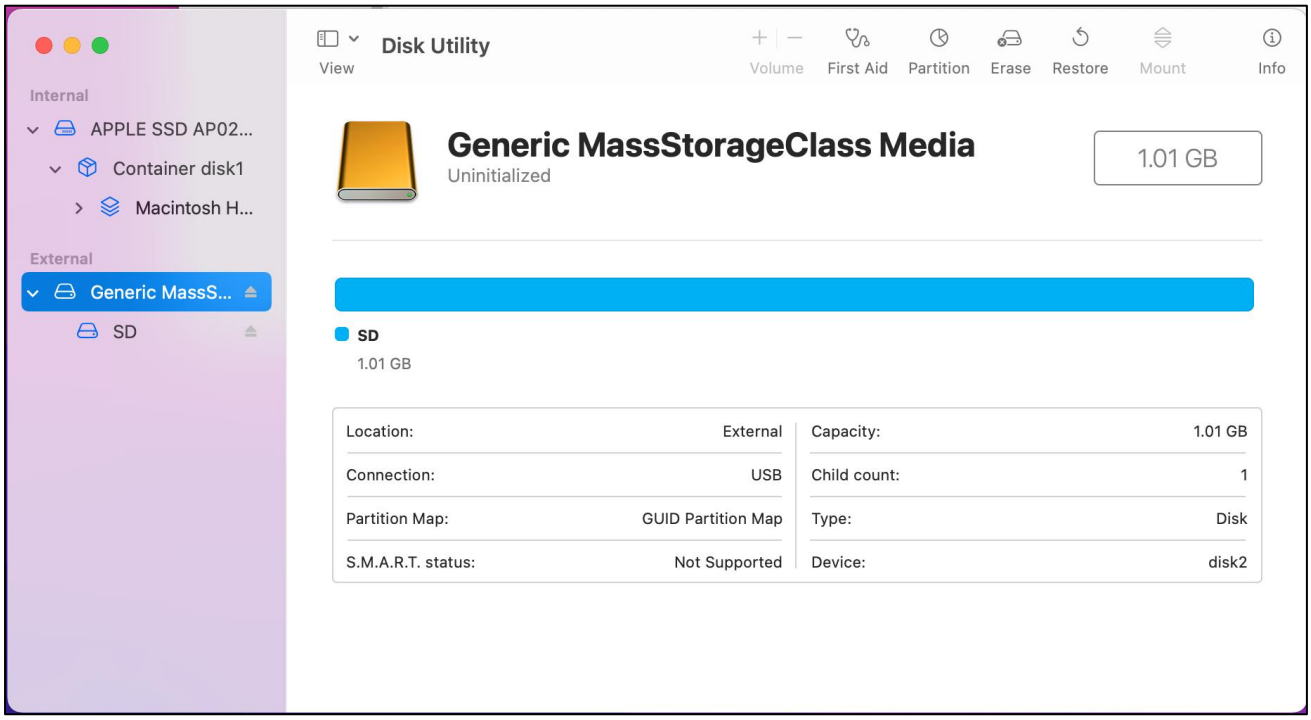
Select "Generic MassStorageClass Media", note that its size is about 1G. Please do not choose wrong item. Click "Erase".



Select the configuration as shown in the figure below, and then click "Erase".

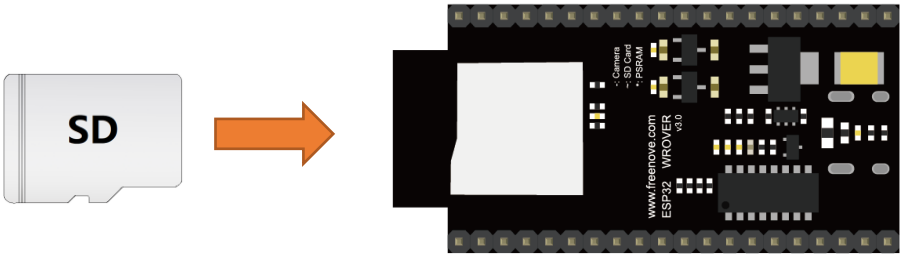


Wait for the formatting to complete. When finished, it will look like the picture below. At this point, you can see a new disk on the desktop named "SD".

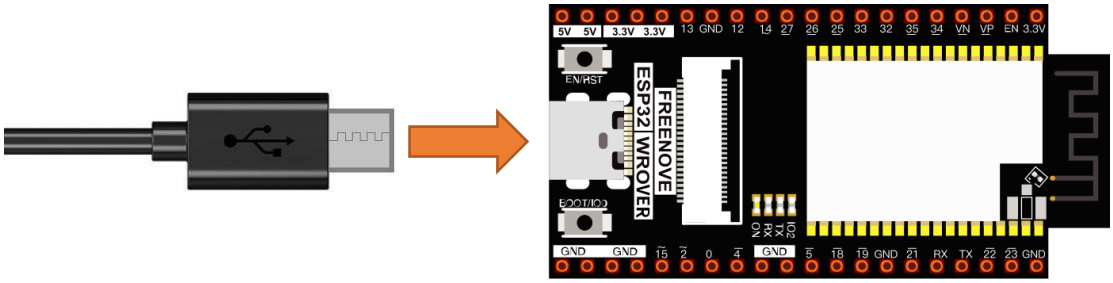


Circuit

Before connecting the USB cable, insert the SD card into the SD card slot on the back of the ESP32.

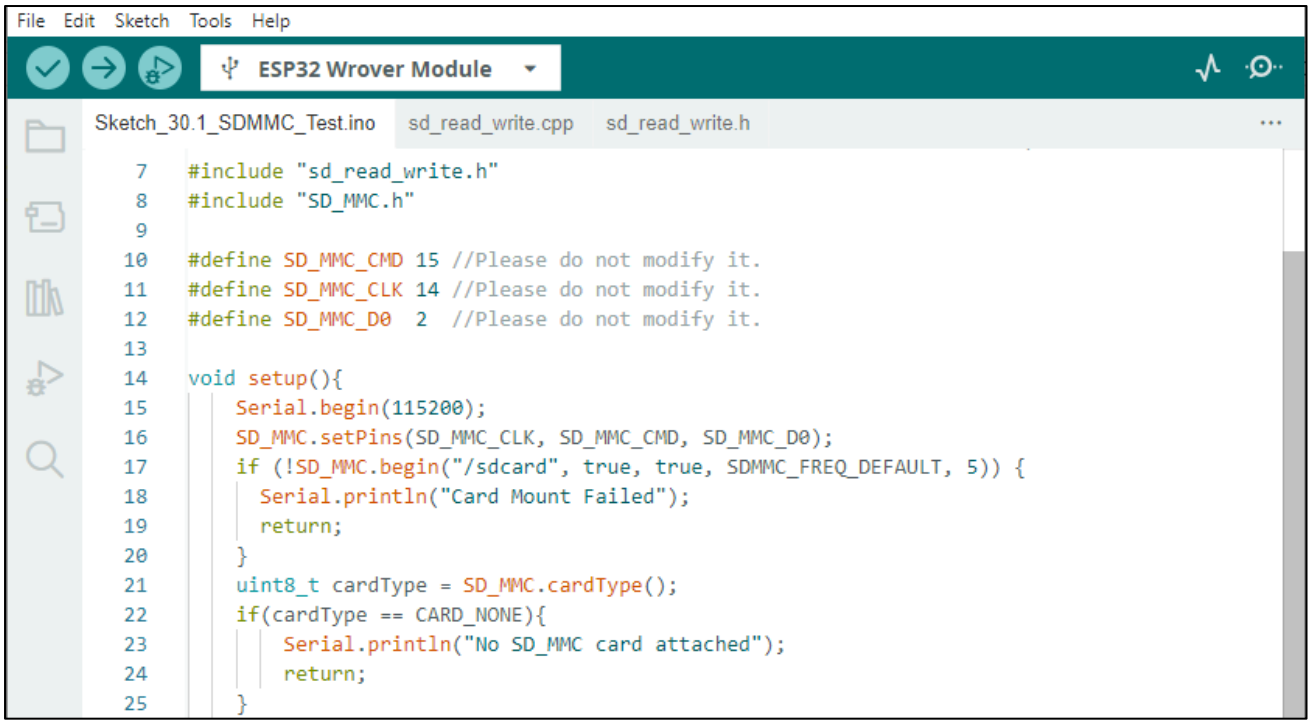


Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_14.1_SDMMC_Test



Compile and upload the code to ESP32, open the serial monitor, and press the RST button on the board. You can see the printout as shown below.

```
14:47:21.289 -> Listing directory: /System Volume Information
14:47:21.289 ->   FILE: WPSettings.dat  SIZE: 12
14:47:21.289 ->   FILE: IndexerVolumeGuid  SIZE: 76
14:47:21.289 ->   FILE: test.txt  SIZE: 1048576
14:47:21.289 ->   FILE: foo.txt  SIZE: 13
14:47:21.289 ->   DIR : music
14:47:21.289 -> Listing directory: /music
14:47:21.289 ->   FILE: 01.mp3  SIZE: 3528199
14:47:21.289 ->   FILE: Good Time.mp3  SIZE: 3291708
14:47:21.289 ->   FILE: Jingle Bells.mp3  SIZE: 8004409
14:47:21.289 -> Writing file: /hello.txt
14:47:21.289 -> File written
14:47:21.416 -> Appending to file: /hello.txt
14:47:21.416 -> Message appended
14:47:21.461 -> Reading file: /hello.txt
14:47:21.461 -> Read from file: Hello World!
14:47:21.461 -> Deleting file: /foo.txt
14:47:21.502 -> File deleted
14:47:21.502 -> Renaming file /hello.txt to /foo.txt
14:47:21.535 -> File renamed
14:47:21.535 -> Reading file: /foo.txt
14:47:21.535 -> Read from file: Hello World!
14:47:22.054 -> 1048576 bytes read for 547 ms
14:47:22.758 -> 1048576 bytes written for 687 ms
14:47:22.804 -> Total space: 953MB
14:47:22.804 -> Used space: 15MB
```

The following is the program code:

```
#include "sd_read_write.h"
#include "SD_MMC.h"

#define SD_MMC_CMD 15 //Please do not modify it.
#define SD_MMC_CLK 14 //Please do not modify it.
#define SD_MMC_D0  2  //Please do not modify it.

void setup(){
    Serial.begin(115200);
    SD_MMC.setPins(SD_MMC_CLK, SD_MMC_CMD, SD_MMC_D0);
    if (!SD_MMC.begin("/sdcard", true, true, SDMMC_FREQ_DEFAULT, 5)) {
      Serial.println("Card Mount Failed");
      return;
    }
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD_MMC card attached");
        return;
    }
    Serial.print("SD_MMC Card Type: ");
    if(cardType == CARD_MMC){
        Serial.println("MMC");
    } else if(cardType == CARD_SD){
        Serial.println("SDSC");
    } else if(cardType == CARD_SDHC){
        Serial.println("SDHC");
    } else {
        Serial.println("UNKNOWN");
    }

    uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
    Serial.printf("SD_MMC Card Size: %lluMB\n", cardSize);

    listDir(SD_MMC, "/", 0);

    createDir(SD_MMC, "/mydir");
    listDir(SD_MMC, "/", 0);

    removeDir(SD_MMC, "/mydir");
    listDir(SD_MMC, "/", 2);

    writeFile(SD_MMC, "/hello.txt", "Hello ");
    appendFile(SD_MMC, "/hello.txt", "World!\n");
```

```
    readFile(SD_MMC, "/hello.txt");

    deleteFile(SD_MMC, "/foo.txt");
    renameFile(SD_MMC, "/hello.txt", "/foo.txt");
    readFile(SD_MMC, "/foo.txt");

    testFileIO(SD_MMC, "/test.txt");

    Serial.printf("Total space: %lluMB\r\n", SD_MMC.totalBytes() / (1024 * 1024));
    Serial.printf("Used space: %lluMB\r\n", SD_MMC.usedBytes() / (1024 * 1024));
}

void loop(){
  delay(10000);
}
```

Add the SD card drive header file.

```
#include "sd_read_write.h"
#include "SD_MMC.h"
```

Defines the drive pins of the SD card. Please do not modify it. Because these pins are fixed.

```
#define SD_MMC_CMD 15 //Please do not modify it.
#define SD_MMC_CLK 14 //Please do not modify it.
#define SD_MMC_D0  2  //Please do not modify it.
```

Initialize the serial port function. Sets the drive pin for SDMMC one-bit bus mode.

```
    Serial.begin(115200);
    SD_MMC.setPins(SD_MMC_CLK, SD_MMC_CMD, SD_MMC_D0);
```

Set the mount point of the SD card, set SDMMC to one-bit bus mode, and set the read and write speed to 20MHz.

```
    if (!SD_MMC.begin("/sdcard", true, true, SDMMC_FREQ_DEFAULT, 5)) {
      Serial.println("Card Mount Failed");
      return;
    }
```

Get the type of SD card and print it out through the serial port.

```
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD_MMC card attached");
        return;
    }
    Serial.print("SD_MMC Card Type: ");
    if(cardType == CARD_MMC){
        Serial.println("MMC");
    } else if(cardType == CARD_SD){
        Serial.println("SDSC");
    } else if(cardType == CARD_SDHC){
        Serial.println("SDHC");
```

```
27      } else {
28          Serial.println("UNKNOWN");
29      }
```

Call the listDir() function to read the folder and file names in the SD card, and print them out through the serial port. This function can be found in "sd_read_write.cpp".

```
34  listDir(SD_MMC, "/", 0);
```

Call createDir() to create a folder, and call removeDir() to delete a folder.

```
36  createDir(SD_MMC, "/mydir");
39  removeDir(SD_MMC, "/mydir");
```

Call writeFile() to write any content to the txt file. If there is no such file, create this file first.

Call appendFile() to append any content to txt.

Call readFile() to read the content in txt and print it via the serial port.

```
42  writeFile(SD_MMC, "/hello.txt", "Hello ");
43  appendFile(SD_MMC, "/hello.txt", "World!\n");
44  readFile(SD_MMC, "/hello.txt");
```

Call deleteFile() to delete a specified file.

Call renameFile() to copy a file and rename it.

```
46      deleteFile(SD_MMC, "/foo.txt");
47      renameFile(SD_MMC, "/hello.txt", "/foo.txt");
```

Call the testFileIO() function to test the time it takes to read 512 bytes and the time it takes to write 2048*512 bytes of data.

```
50  testFileIO(SD_MMC, "/test.txt");
```

Print the total size and used size of the SD card via the serial port.

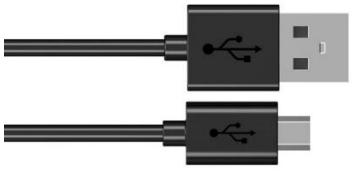
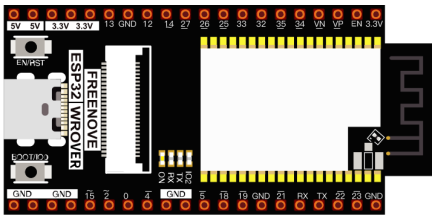
```
52      Serial.printf("Total space: %lluMB\r\n", SD_MMC.totalBytes() / (1024 * 1024));
53      Serial.printf("Used space: %lluMB\r\n", SD_MMC.usedBytes() / (1024 * 1024));
```

Chapter 15 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-WROVER.
ESP32-WROVER has 3 different WiFi operating modes: station mode, AP mode and AP+station mode. All WiFi programming projects must be configured with WiFi operating mode before using WiFi, otherwise WiFi cannot be used.

Project 15.1 Station mode

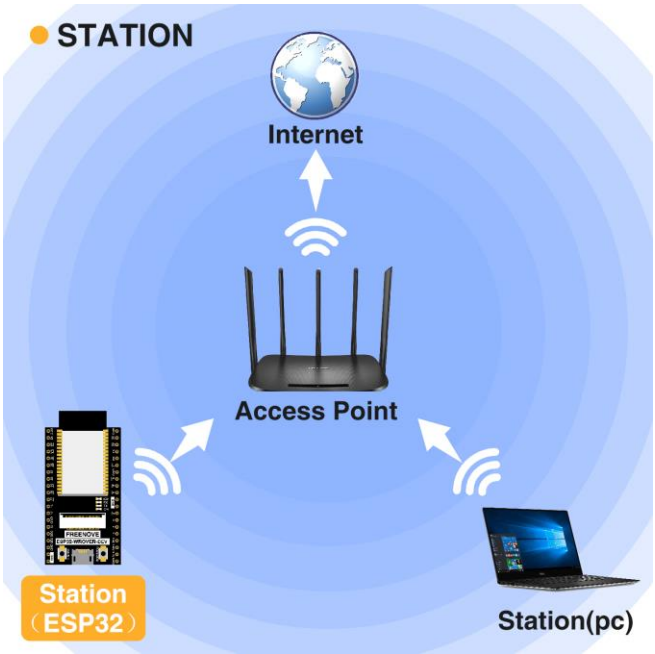
Component List

Micro USB Wire x1	ESP32-WROVER x1
	

Component knowledge

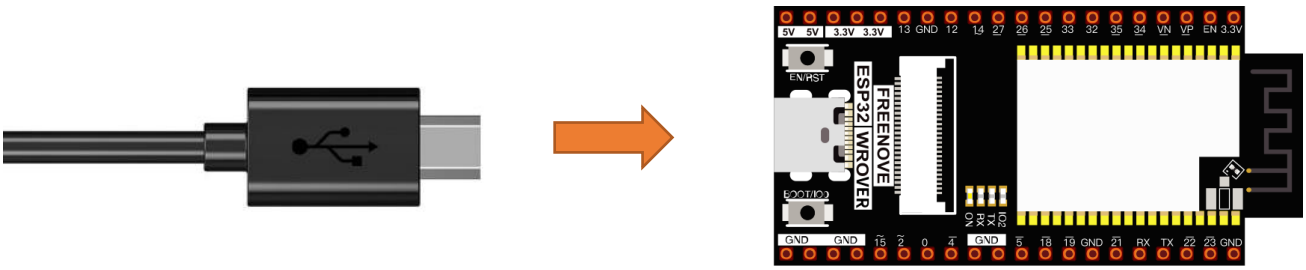
Station mode

When ESP32 selects Station mode, it acts as a WiFi client. It can connect to the router network and communicate with other devices on the router via WiFi connection. As shown below, the PC is connected to the router, and if ESP32 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.

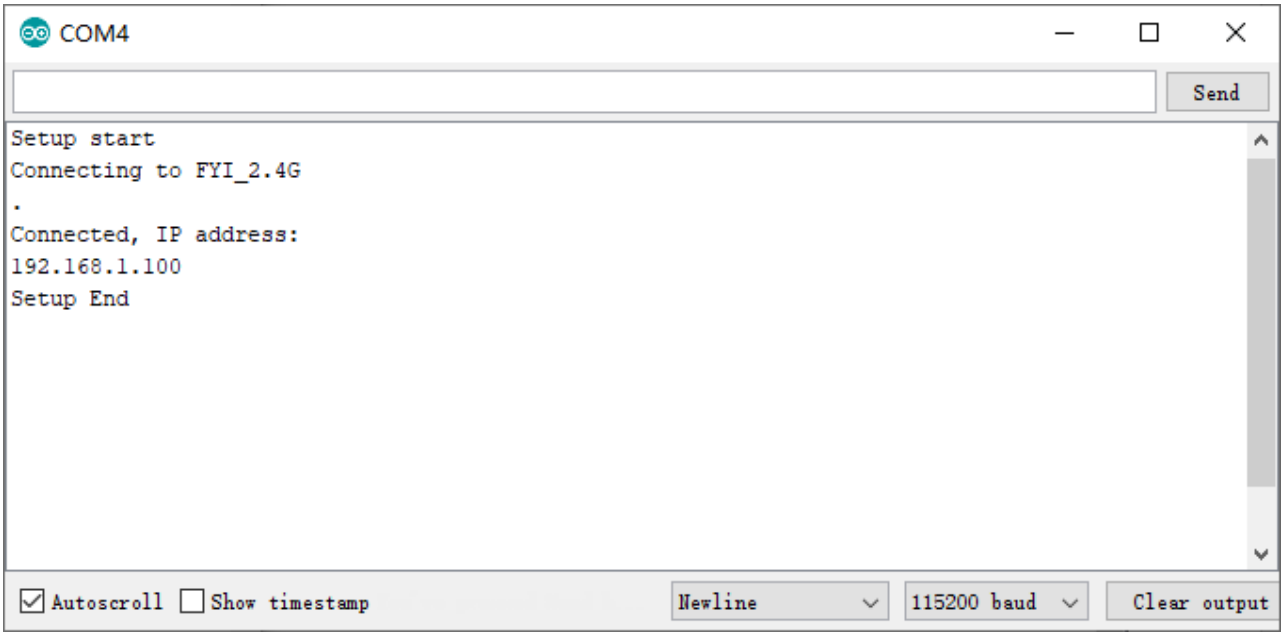


Sketch

Sketch_15.1_Station_mode

Because the names and passwords of routers in various places are different, before the Sketch runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to ESP32-WROVER, open serial monitor and set baud rate to 115200. And then it will display as follows:



When ESP32-WROVER successfully connects to "ssid_Router", serial monitor will print out the IP address assigned to ESP32-WROVER by the router.

The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     = "********"; //Enter the router name
const char *password_Router = "********"; //Enter the router password

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setup start");
  WiFi.begin(ssid_Router, password_Router);
  Serial.println(String("Connecting to ")+ssid_Router);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected, IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("Setup End");
}

void loop() {
}
```

Include the WiFi Library header file of ESP32.

```
#include <WiFi.h>
```

Enter correct router name and password.

```
3  const char *ssid_Router     = "********"; //Enter the router name
4  const char *password_Router = "********"; //Enter the router password
```

Set ESP32 in Station mode and connect it to your router.

```
10   WiFi.begin(ssid_Router, password_Router);
```

Check whether ESP32 has connected to router successfully every 0.5s.

```
12   while (WiFi.status() != WL_CONNECTED){
13     delay(500);
14     Serial.print(".");
15   }
```

Serial monitor prints out the IP address assigned to ESP32-WROVER

```
17   Serial.println(WiFi.localIP());
```

Reference

Class Station

Every time when using WiFi, you need to include header file "WiFi.h.".

begin(ssid, password,channel, bssid, connect): ESP32 is used as Station to connect hotspot.

ssid: WiFi hotspot name

password: WiFi hotspot password

channel: WiFi hotspot channel number; communicating through specified channel; optional parameter

bssid: mac address of WiFi hotspot, optional parameter

connect: blloean optional parameter, defaulting to true. If set as false, then ESP32 won't connect WiFi.

config(local_ip, gateway, subnet, dns1, dns2): set static local IP address.

local_ip: station fixed IP address.

subnet: subnet mask

dns1,dns2: optional parameter. define IP address of domain name server

status: obtain the connection status of WiFI

local IP(): obtian IP address in Station mode

disconnect(): disconnect wifi

setAutoConnect(boolen): set automatic connection Every time ESP32 is power on, it will connect WiFi aitomatically.

setAutoReconnect(boolen): set automatic reconnection Every time ESP32 disconnects WiFi, it will reconnect to WiFi automatically.

Project 15.2 AP mode

Component List & Circuit

Component List & Circuit are the same as in Section 30.1.

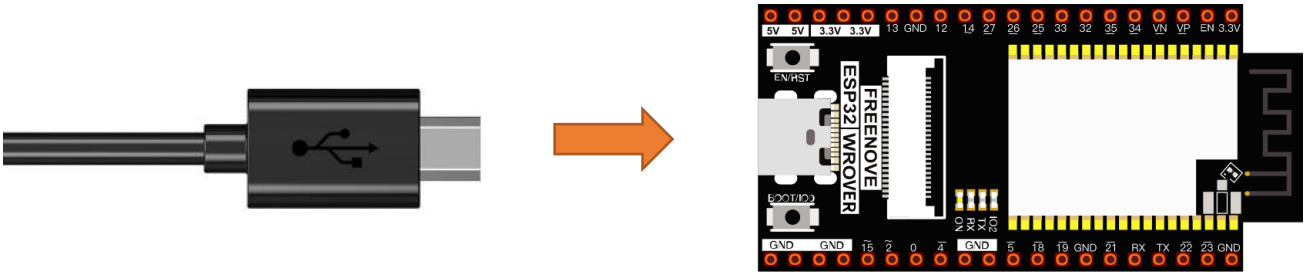
Component knowledge

AP mode

When ESP32 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 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP32, it must be connected to the hotspot of ESP32. Only after a connection is established with ESP32 can they communicate.

Circuit

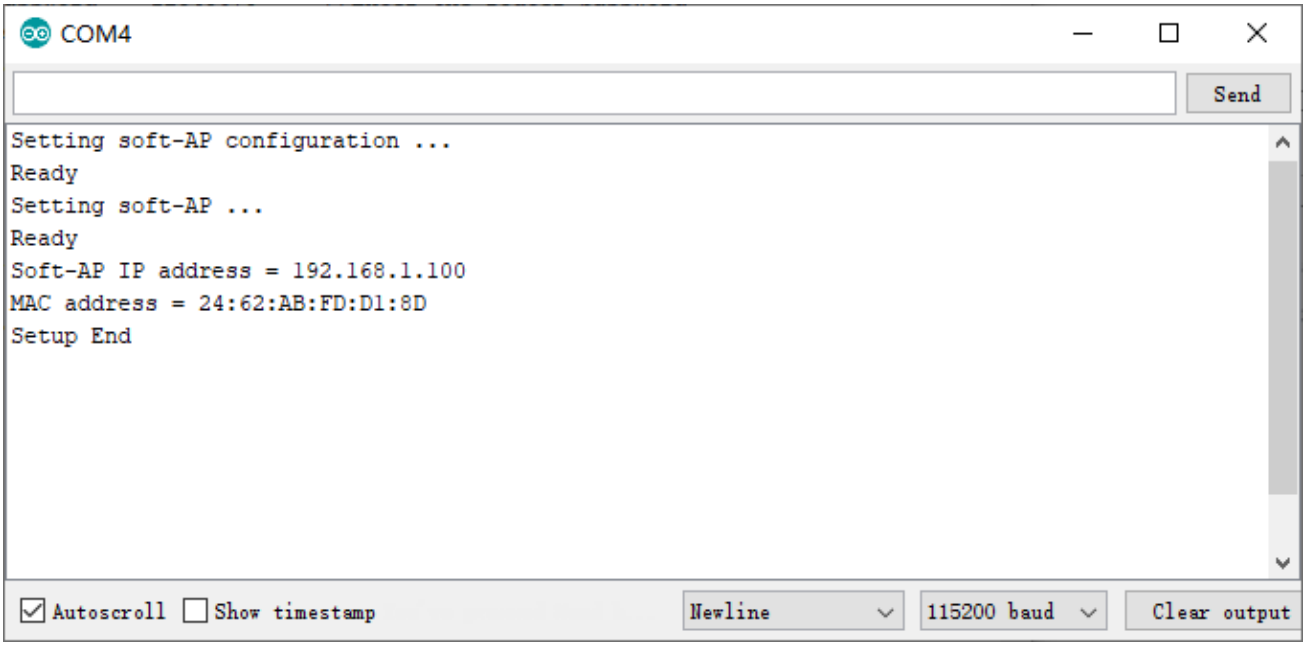
Connect Freenove ESP32 to the computer using the USB cable.



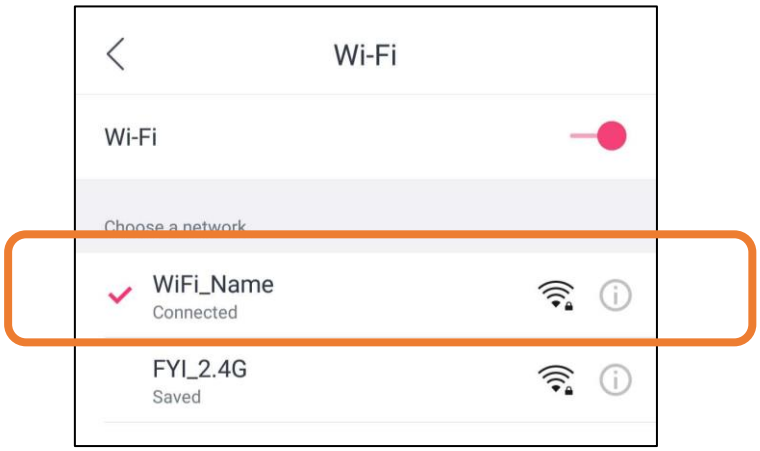
Sketch

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

Compile and upload codes to ESP32-WROVER, open the serial monitor and set the baud rate to 115200. And then it will display as follows.



When observing the print information of the serial monitor, turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32, which is called "WiFi_Name" in this Sketch. You can enter the password "12345678" to connect it or change its AP name and password by modifying Sketch.



Sketch_15.2_AP_mode

The following is the program code:

```
#include <WiFi.h>

const char *ssid_AP     = "WiFi_Name"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password

IPAddress local_IP(192,168,1,100);//Set the IP address of ESP32 itself
IPAddress gateway(192,168,1,10);   //Set the gateway of ESP32 itself
IPAddress subnet(255,255,255,0);  //Set the subnet mask for ESP32 itself

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setting soft-AP configuration ... ");
  WiFi.disconnect();
  WiFi.mode(WIFI_AP);
  Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
  Serial.println("Setting soft-AP ... ");
  boolean result = WiFi.softAP(ssid_AP, password_AP);
  if(result){
    Serial.println("Ready");
    Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
    Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
  }else{
    Serial.println("Failed!");
  }
  Serial.println("Setup End");
}

void loop() {
}
```

Include WiFi Library header file of ESP32.

```
#include <WiFi.h>
```

Enter correct AP name and password.

```
const char *ssid_AP     = "WiFi_Name"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password
```

Set ESP32 in AP mode.

```
WiFi.mode(WIFI_AP);
```

Configure IP address, gateway and subnet mask for ESP32.

```
WiFi.softAPConfig(local_IP, gateway, subnet)
```

Turn on an AP in ESP32, whose name is set by ssid_AP and password is set by password_AP.

```
WiFi.softAP(ssid_AP, password_AP);
```

Check whether the AP is turned on successfully. If yes, print out IP and MAC address of AP established by ESP32. If no, print out the failure prompt.

```
19  if(result){
20      Serial.println("Ready");
21      Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
22      Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
23    }else{
24      Serial.println("Failed!");
25    }
26    Serial.println("Setup End");
```

Reference

Class AP

Every time when using WiFi, you need to include header file "WiFi.h.".

softAP(ssid, password, channel, ssid_hidden, max_connection):

ssid: WiFi hotspot name

password: WiFi hotspot password

channel: Number of WiFi connection channels, range 1-13. The default is 1.

ssid_hidden: Whether to hide WiFi name from scanning by other devices. The default is not hide.

max_connection: Maximum number of WiFi connected devices. The range is 1-4. The default is 4.

softAPConfig(local_ip, gateway, subnet): set static local IP address.

local_ip: station fixed IP address.

Gateway: gateway IP address

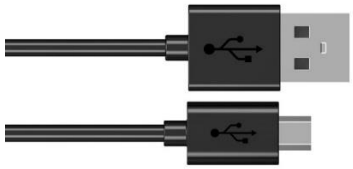
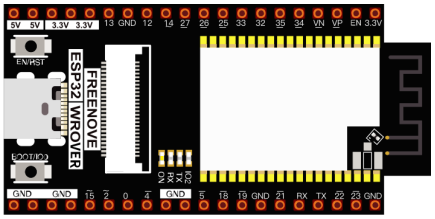
subnet: subnet mask

softAP(): obtian IP address in AP mode

softAPdisconnect (): disconnect AP mode.

Project 14.3 AP+Station mode

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

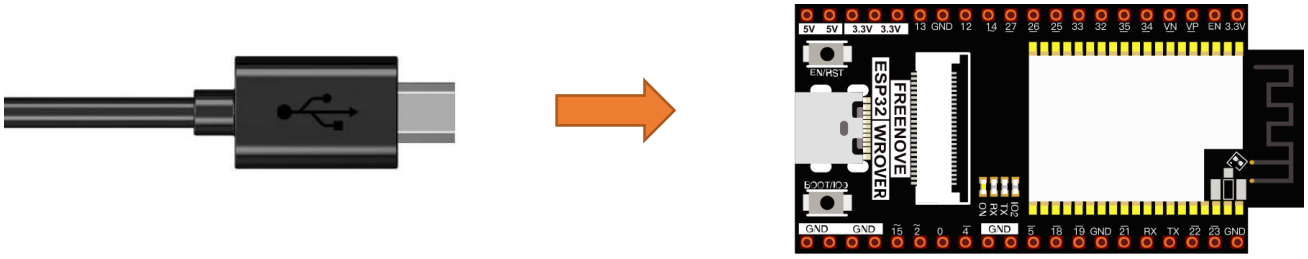
Component knowledge

AP+Station mode

In addition to AP mode and station mode, ESP32 can also use AP mode and station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32's station mode, connect it to the router network, and it can communicate with the Internet via the router. At the same time, turn on its AP mode to create a hotspot network. Other WiFi devices can choose to connect to the router network or the hotspot network to communicate with ESP32.

Circuit

Connect Freenove ESP32 to the computer using the USB cable.

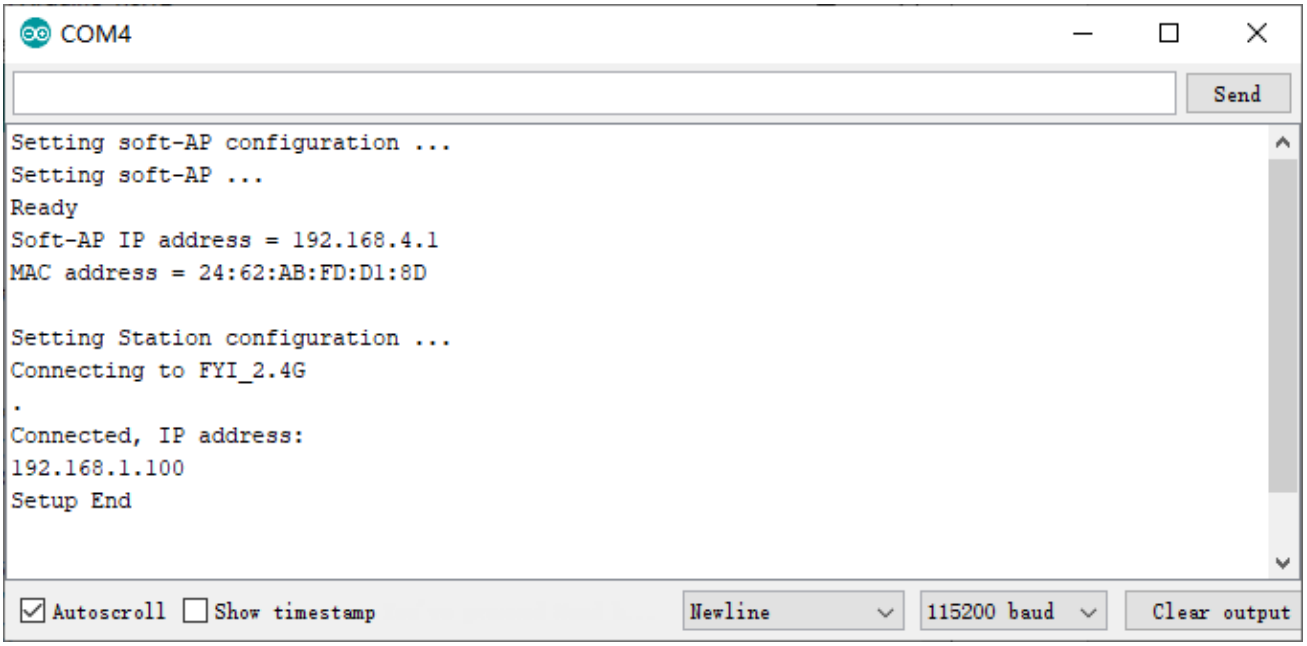


Sketch

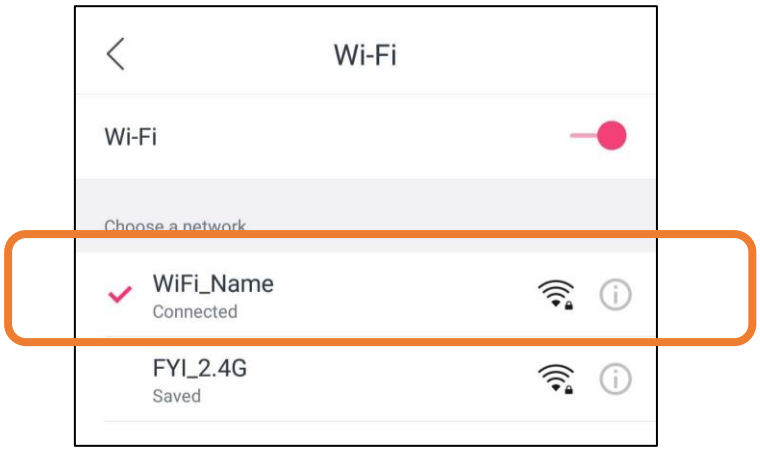
Sketch_15.3_AP_Station_mode

It is analogous to Project 15.1 and Project 15.2. Before running the Sketch, you need to modify ssid_Router, password_Router, ssid_AP and password_AP shown in the box of the illustration above.

After making sure that Sketch is modified correctly, compile and upload codes to ESP32-WROVER, open serial monitor and set baud rate to 115200. And then it will display as follows:



When observing the print information of the serial monitor, turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32.



The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     = "********";  //Enter the router name
const char *password_Router = "********";  //Enter the router password
const char *ssid_AP         = "WiFi_Name"; //Enter the AP name
const char *password_AP     = "12345678";  //Enter the AP password

void setup(){
  Serial.begin(115200);
  Serial.println("Setting soft-AP configuration ... ");
  WiFi.disconnect();
```

```
12    WiFi.mode(WIFI_AP);
13    Serial.println("Setting soft-AP ... ");
14    boolean result = WiFi.softAP(ssid_AP, password_AP);
15    if(result){
16      Serial.println("Ready");
17      Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
18      Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
19    }else{
20      Serial.println("Failed!");
21    }
22
23    Serial.println("\nSetting Station configuration ... ");
24    WiFi.begin(ssid_Router, password_Router);
25    Serial.println(String("Connecting to ")+ ssid_Router);
26    while (WiFi.status() != WL_CONNECTED){
27      delay(500);
28      Serial.print(".");
29    }
30    Serial.println("\nConnected, IP address: ");
31    Serial.println(WiFi.localIP());
32    Serial.println("Setup End");
33  }
34
35  void loop() {
36  }
```

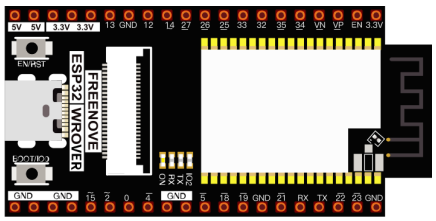
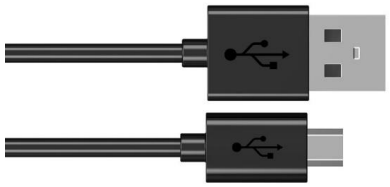
Chapter 16 TCP/IP

In this chapter, we will introduce how ESP32 implements network communications based on TCP/IP protocol. There are two roles in TCP/IP communication, namely Server and Client, which will be implemented respectively with two projects in this chapter.

Project 16.1 As Client

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

Component List

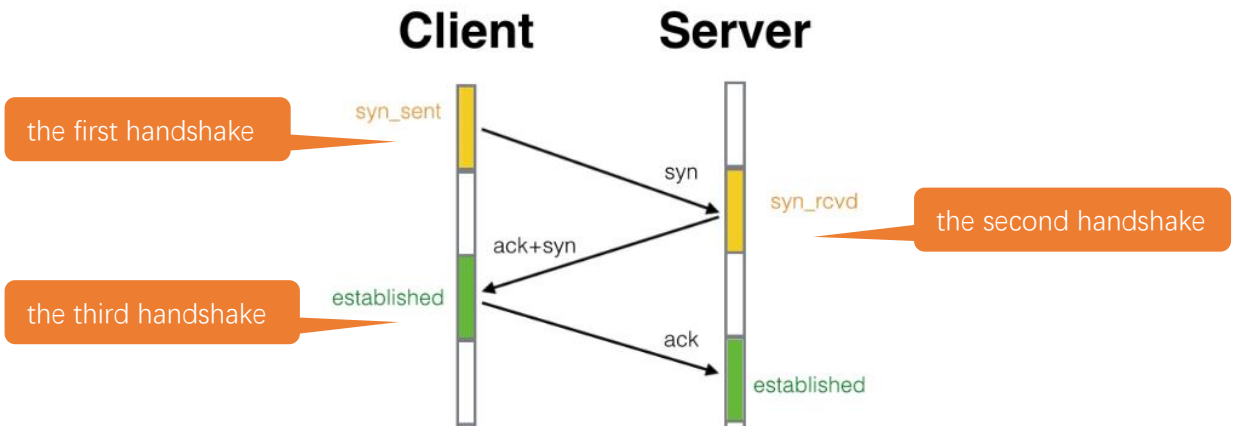
ESP32-WROVER x1 	Micro USB Wire x1 

Component knowledge

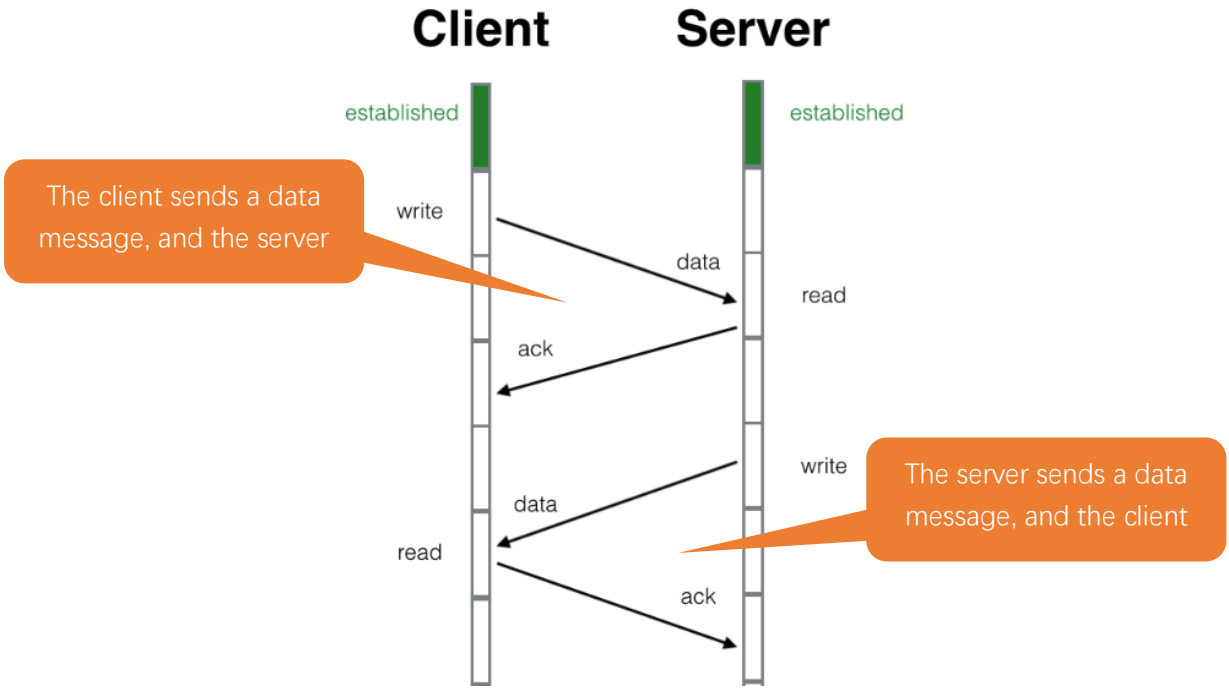
TCP connection

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

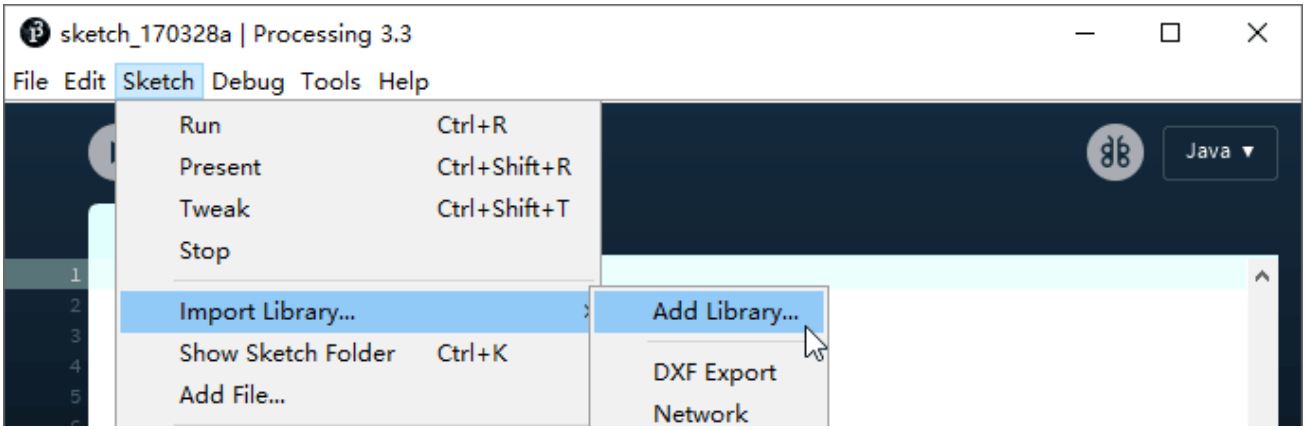
In this tutorial, we use Processing to build a simple TCP/IP communication platform.

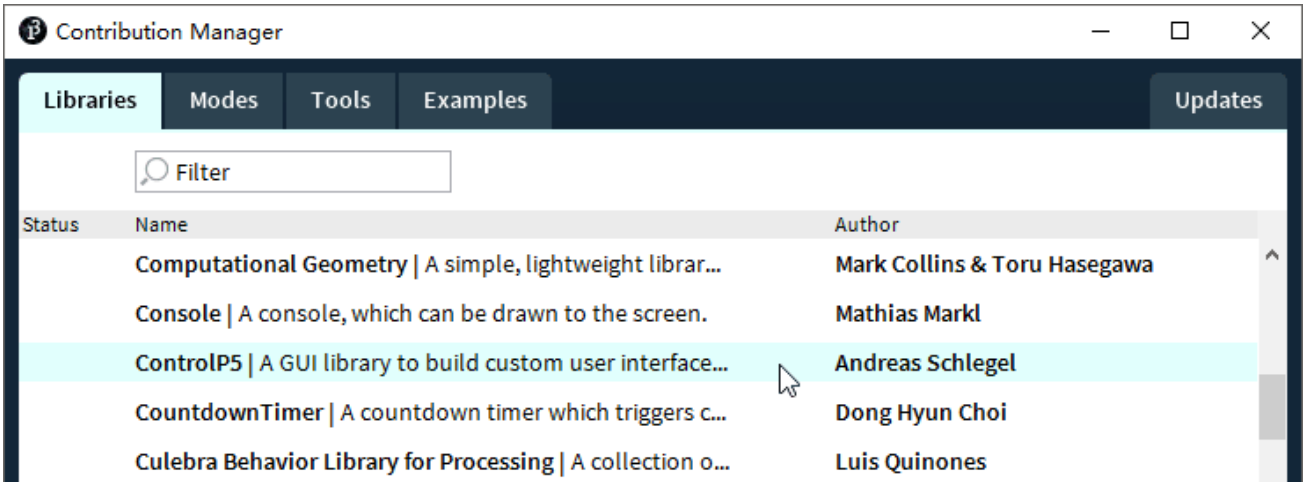
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

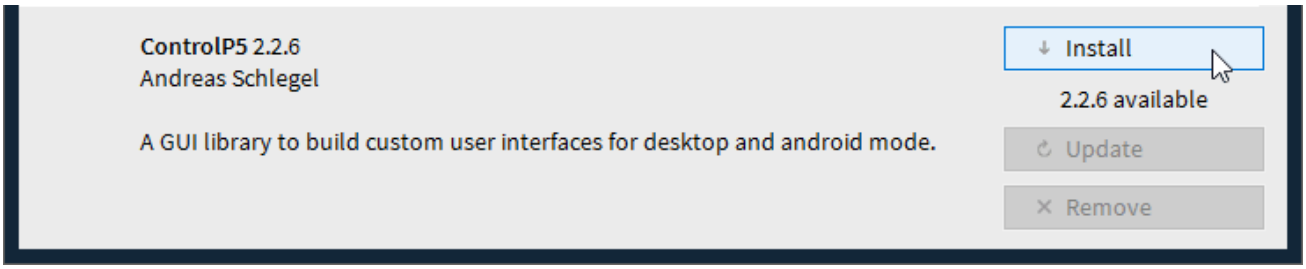
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

Use Server mode for communication

Install ControlP5.





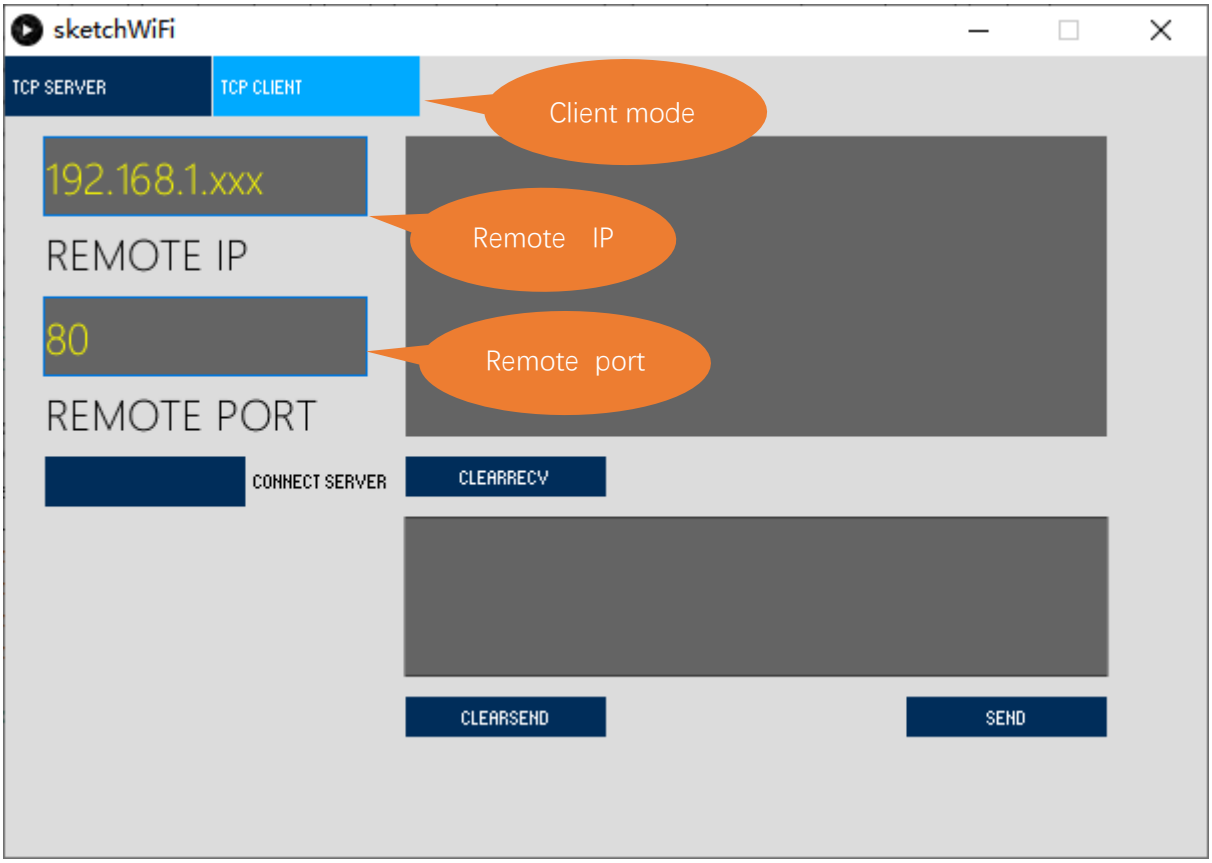


Open the "**Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketches\Sketch_16.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

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

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

If ESP32 serves as server, select TCP CLIENT mode for sketchWiFi.



When sketchWiFi selects TCP CLIENT mode, the LOCAL IP and LOCAL PORT of sketchWiFi need to be changed according to the IP address and port number printed by the serial monitor.

Mode selection: select **Server mode/Client mode**.

IP address: In server mode, this option does not need to be filled in, and the computer will automatically obtain the IP address.

In client mode, fill in the remote IP address to be connected.

Port number: In server mode, fill in a port number for client devices to make an access connection.

In client mode, fill in port number given by the Server devices to make an access connection.

Start button: In server mode, push the button, then the computer will serve as server and open a port number for client to make access connection. During this period, the computer will keep monitoring.

In client mode, before pushing the button, please make sure the server is on, remote IP address and remote port number is correct; push the button, and the computer will make access connection to the remote port number of the remote IP as a client.

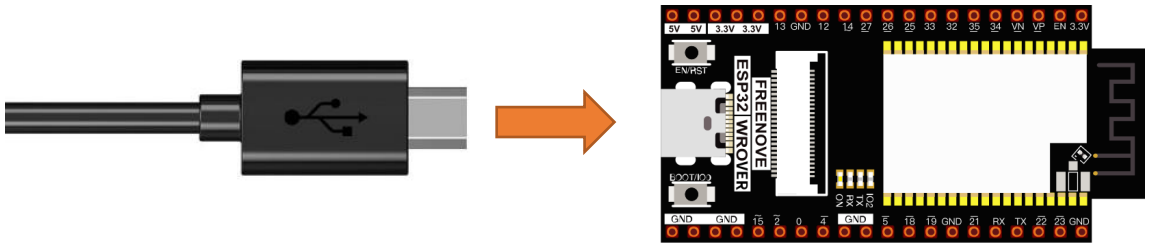
clear receive: clear out the content in the receiving text box

clear send: clear out the content in the sending text box

Sending button: push the sending button, the computer will send the content in the text box to others.

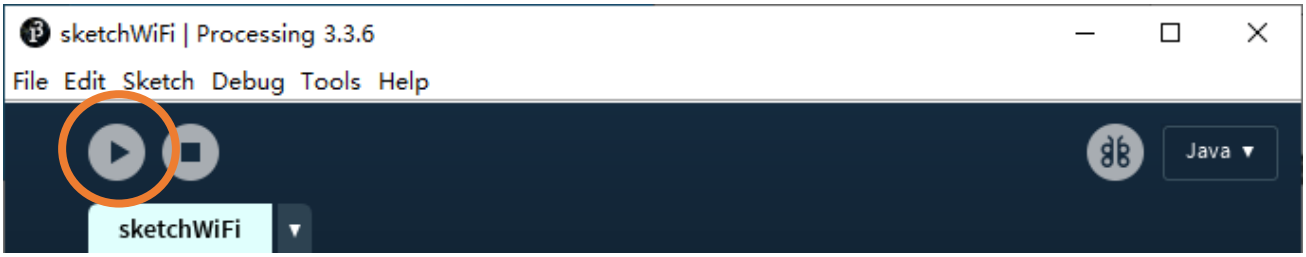
Circuit

Connect Freenove ESP32 to the computer using USB cable.

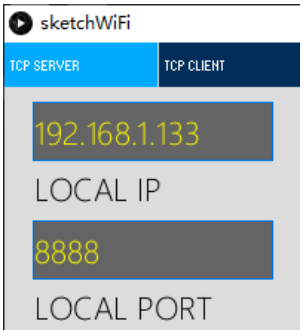


Sketch

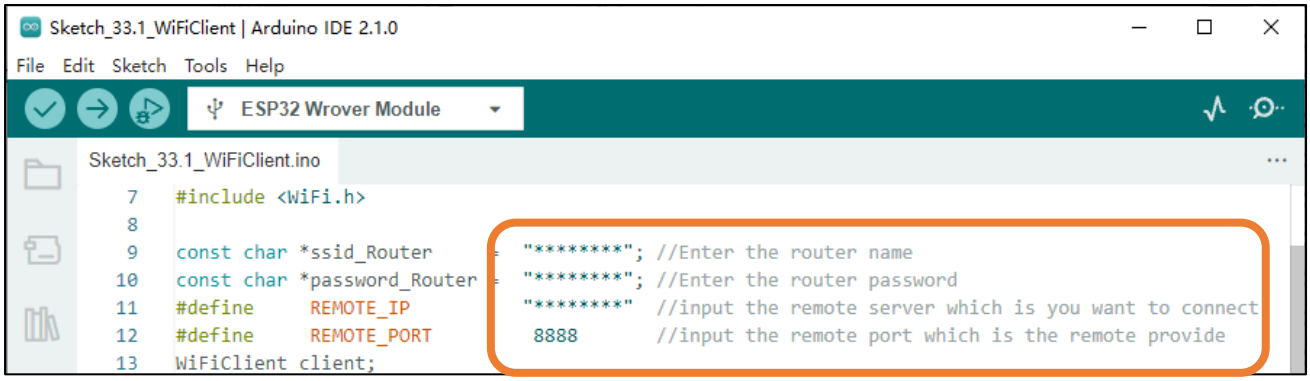
Before running the Sketch, please open “sketchWiFi.pde.” first, and click “Run”.



The newly pop up window will use the computer’s IP address by default and open a data monitor port.

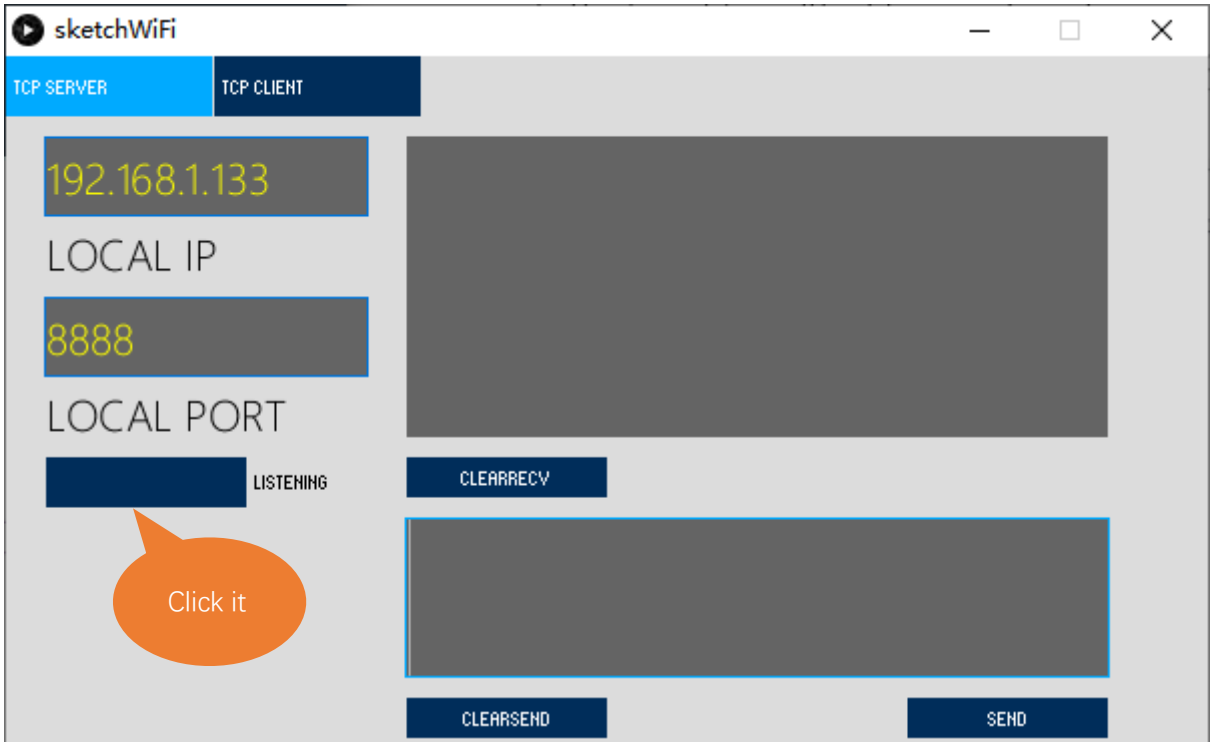


Next, open Sketch_16.1_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.

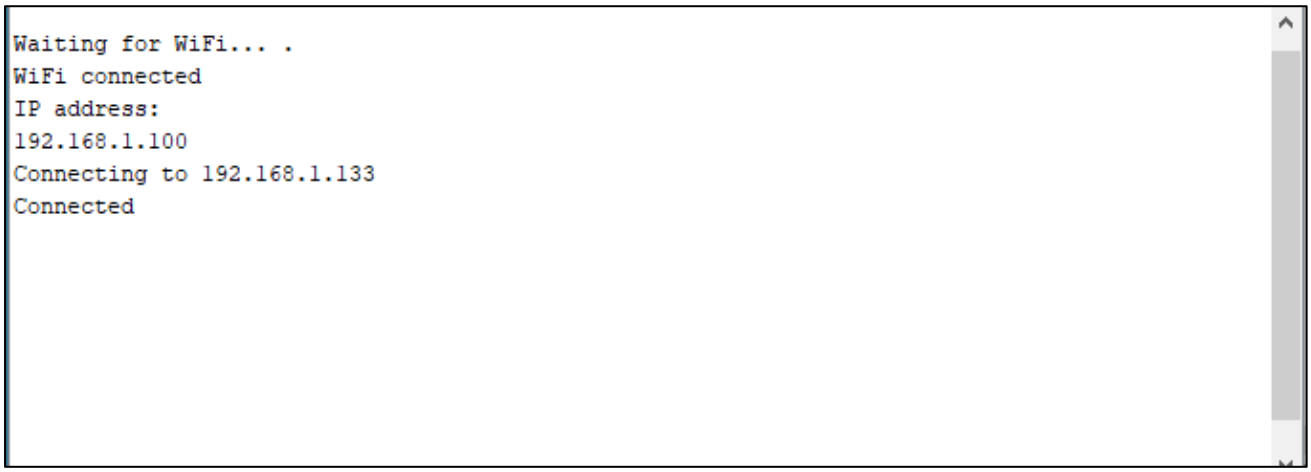


REMOTE_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE_IP is “192.168.1.133”. Generally, by default, the ports do not need to change its value.

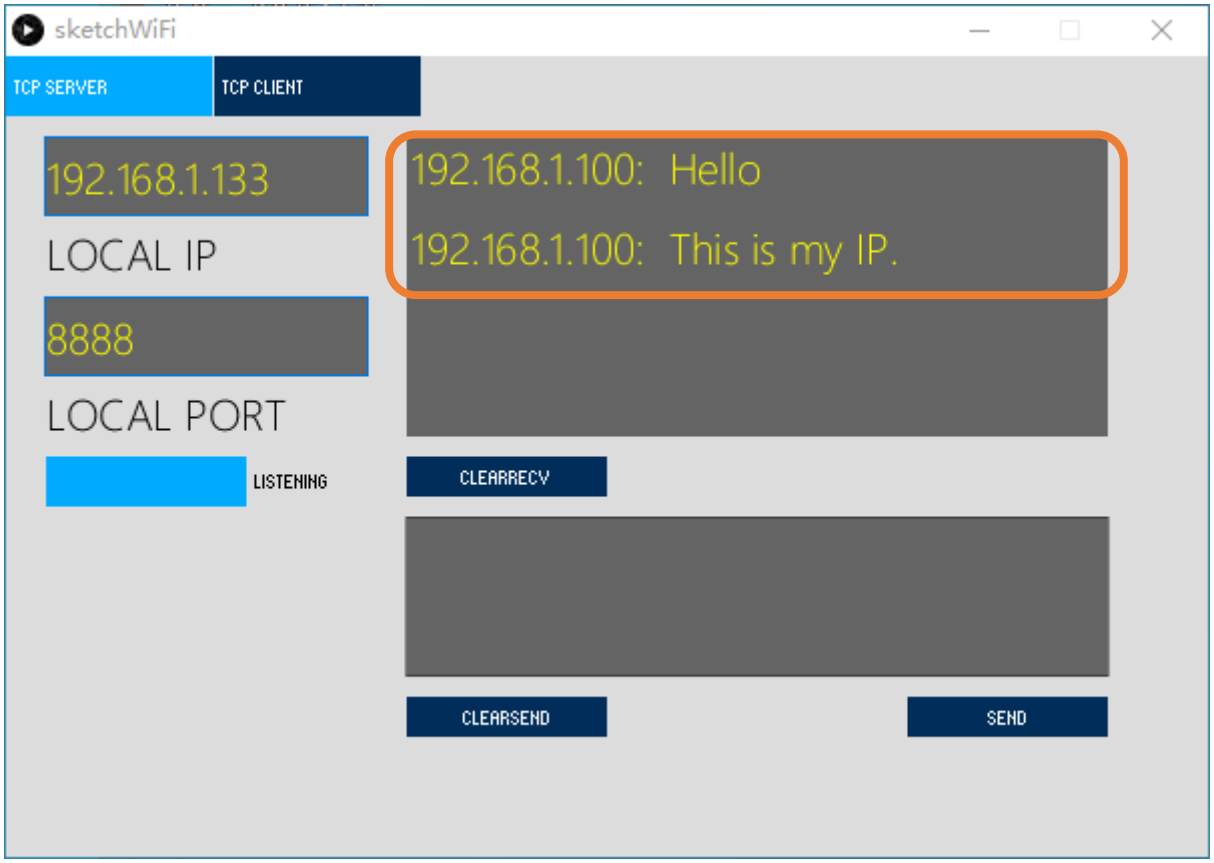
Click LISTENING, turn on TCP SERVER's data listening function and wait for ESP32 to connect.



Compile and upload code to ESP32-WROVER, open the serial monitor and set the baud rate to 115200. ESP32 connects router, obtains IP address and sends access request to server IP address on the same LAN till the connection is successful. When connect successfully, ESP32 can send messages to server.



ESP32 connects with TCP SERVER, and TCP SERVER receives messages from ESP32, as shown in the figure below.



Sketch_16.1_As_Client

The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     =  "********"; //Enter the router name
const char *password_Router =  "********"; //Enter the router password
#define     REMOTE_IP       "********"  //input the remote server which is you want to connect
#define     REMOTE_PORT      8888       //input the remote port which is the remote provide
WiFiClient client;

void setup() {
  Serial.begin(115200);
  delay(10);

  WiFi.begin(ssid_Router, password_Router);
  Serial.print("\nWaiting for WiFi... ");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(500);

  Serial.print("Connecting to ");
  Serial.println(REMOTE_IP);

  while (!client.connect(REMOTE_IP, REMOTE_PORT)) {
    Serial.println("Connection failed.");
    Serial.println("Waiting a moment before retrying...");
  }
  Serial.println("Connected");
  client.print("Hello\n");
  client.print("This is my IP.\n");
}

void loop() {
  if (client.available() > 0) {
    delay(20);
    //read back one line from the server
    String line = client.readString();
    Serial.println(REMOTE_IP + String(":") + line);
```

```
  }
  if (Serial.available() > 0) {
    delay(20);
    String line = Serial.readString();
    client.print(line);
  }
  if (client.connected () == 0) {
    client.stop();
    WiFi.disconnect();
  }
}
```

Add WiFi function header file.

```
#include <WiFi.h>
```

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

```
const char *ssid_Router     =  "********"; //Enter the router name
const char *password_Router =  "********"; //Enter the router password
#define     REMOTE_IP       "********"  //input the remote server which is you want to connect
#define     REMOTE_PORT      8888       //input the remote port which is the remote provide
```

Apply for the method class of WiFiClient.

```
WiFiClient client;
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

```
  WiFi.begin(ssid_Router, password_Router);
  Serial.print("\nWaiting for WiFi... ");
  while (WiFi.status() ! = WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
```

Send connection request to remote server until connect successfully. When connect successfully, print out the connecting prompt on the serial monitor and send messages to remote server.

```
  while (!client.connect(REMOTE_IP, REMOTE_PORT)) {//Connect to Server
    Serial.println("Connection failed.");
    Serial.println("Waiting a moment before retrying...");
  }
  Serial.println("Connected");
  client.print("Hello\n");
```

When ESP32 receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```
if (client.available() > 0) {
    delay(20);
    //read back one line from the server
    String line = client.readString();
    Serial.println(REMOTE_IP + String(":") + line);
  }
```

```
43    if (Serial.available() > 0) {
44      delay(20);
45      String line = Serial.readString();
46      client.print(line);
47    }
```

If the server is disconnected, turn off WiFi of ESP32.

```
48    if (client.connected () == false) {
49      client.stop();
50      WiFi.disconnect();
51    }
```

Reference

Class Client

Every time when using Client, you need to include header file "WiFi.h."

connect(ip, port, timeout)/connect(*host, port, timeout): establish a TCP connection.

ip, *host: ip address of target server

port: port number of target server

timeout: connection timeout

connected(): judge whether client is connecting. If return value is 1, then connect successfully; If return value is 0, then fail to connect.

stop(): stop tcp connection

print(): send data to server connecting to client

available(): return to the number of bytes readable in receive buffer, if no, return to 0 or -1.

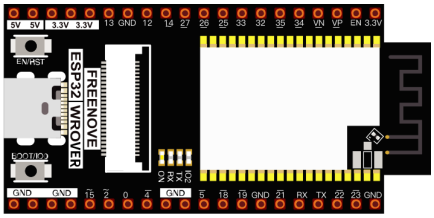
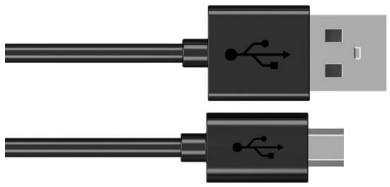
read(): read one byte of data in receive buffer

readString(): read string in receive buffer

Project 16.2 As Server

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

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

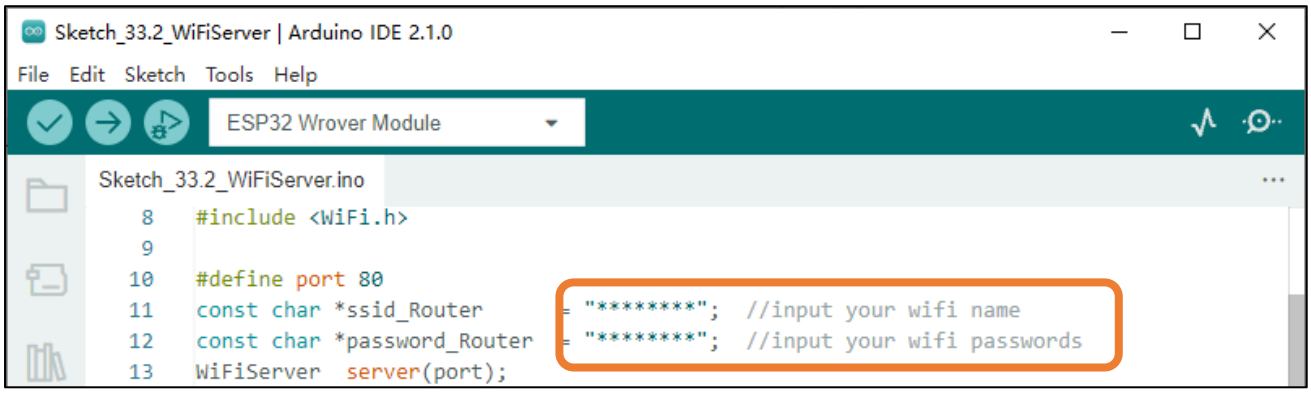
Circuit

Connect Freenove ESP32 to the computer using a USB cable.

Sketch

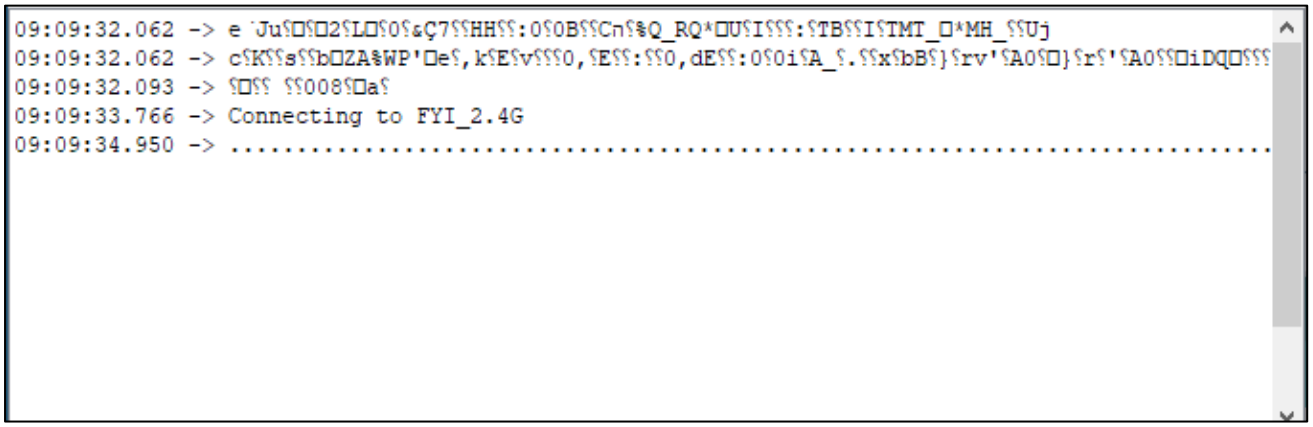
Before running Sketch, please modify the contents of the box below first.

Sketch_16.2_As_Server

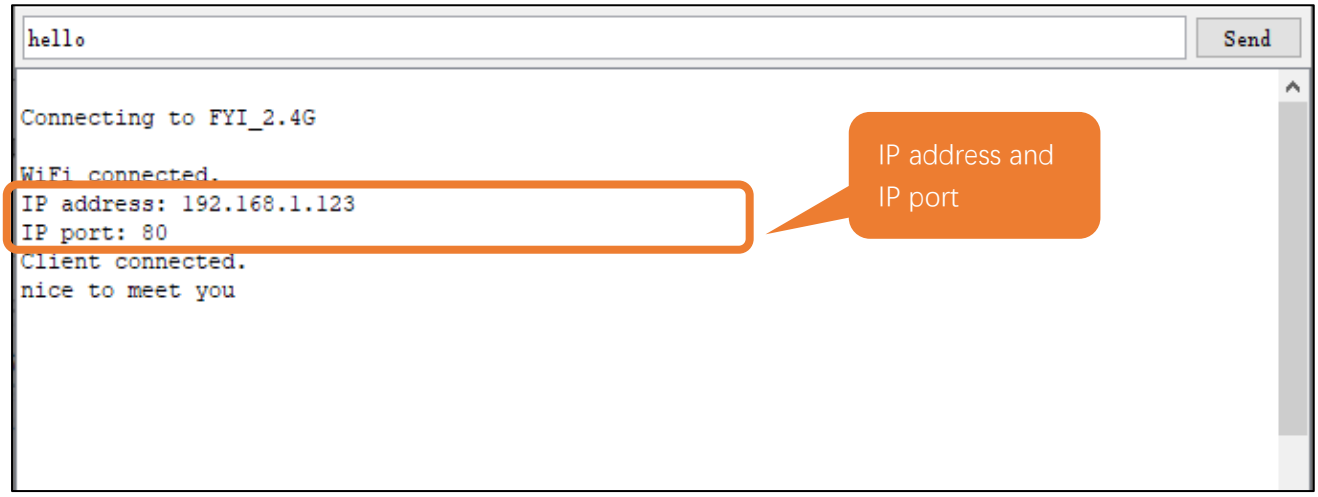


Compile and upload code to ESP32-WROVER board, open the serial monitor and set the baud rate to 115200. Turn on server mode for ESP32, waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other.

If the ESP32 fails to connect to router, press the reset button as shown below and wait for ESP32 to run again.



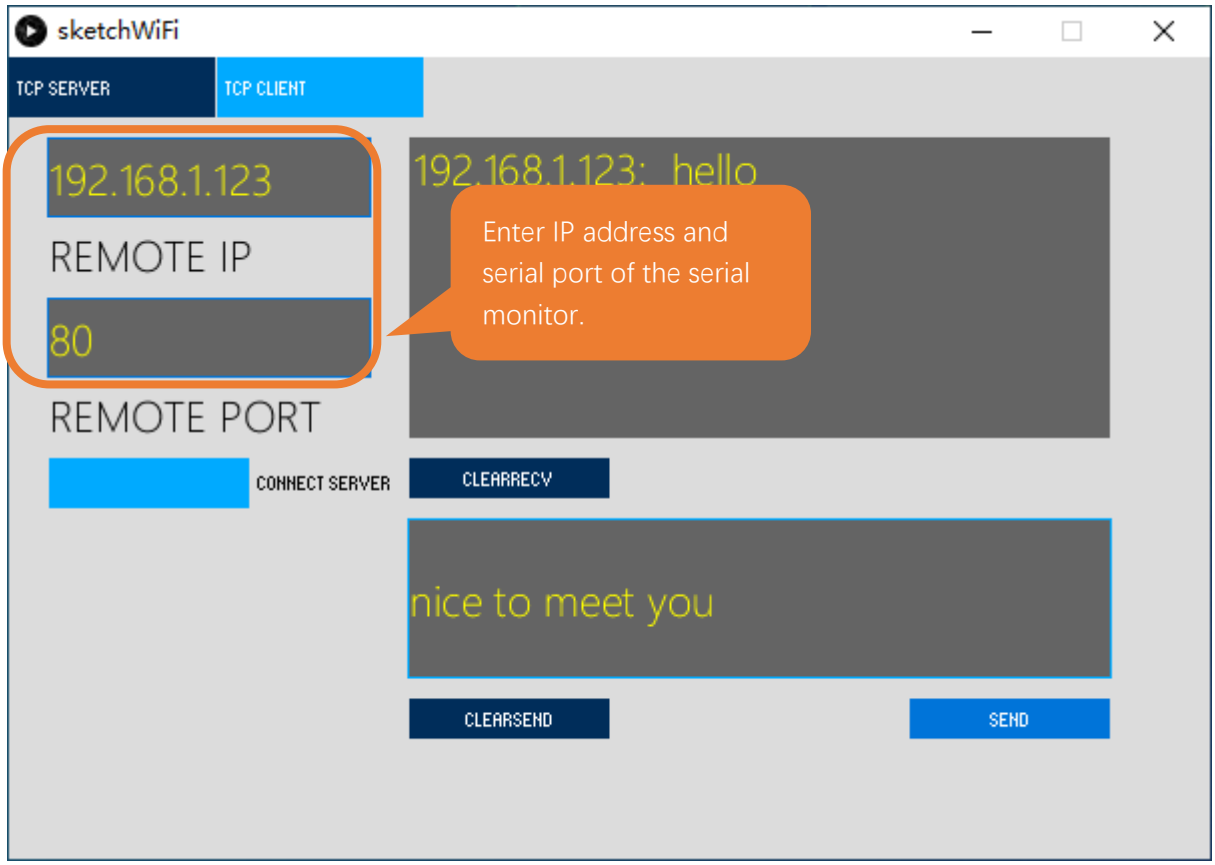
Serial Monitor



Processing:

Open the "**Freenove_Basic_Starter_Kit_for_ESP32\Sketches\Sketches\Sketch_16.2_WiFiServer\sketchWiFi\sketchWiFi.pde**".

Based on the messages printed by the serial monitor, enter correct IP address and serial port in Processing to establish connection and make communication.



The following is the program code:

```
#include <WiFi.h>

#define port 80
const char *ssid_Router      = "********";  //input your wifi name
const char *password_Router  = "********";  //input your wifi passwords
WiFiServer  server(port);

void setup()
{
    Serial.begin(115200);
    Serial.printf("\nConnecting to ");
    Serial.println(ssid_Router);
    WiFi.disconnect();
    WiFi.begin(ssid_Router, password_Router);
    delay(1000);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.printf("IP port: %d\n",port);
    server.begin(port);
    WiFi.setAutoReconnect(true);
}

void loop(){
 WiFiClient client = server.accept();            // listen for incoming clients
  if (client) {                                  // if you get a client
    Serial.println("Client connected.");
    while (client.connected()) {                 // loop while the client's connected
      if (client.available()) {                  // if there's bytes to read from the
client
        Serial.println(client.readStringUntil('\n')); // print it out the serial monitor
        while(client.read()>0);                  // clear the wifi receive area cache
      }
      if(Serial.available()){                    // if there's bytes to read from the
serial monitor
        client.print(Serial.readStringUntil('\n')); // print it out the client.
        while(Serial.read()>0);                  // clear the wifi receive area cache
      }
```

```
42      }
43      client.stop();                                  // stop the client connecting.
44      Serial.println("Client Disconnected.");
45    }
46  }
```

Apply for method class of WiFiServer.

```
6   WiFiServer server(port);          //Apply for a Server object whose port number is 80
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

```
13      WiFi.disconnect();
14      WiFi.begin(ssid_Router, password_Router);
15      delay(1000);
16      while (WiFi.status() != WL_CONNECTED) {
17          delay(500);
18          Serial.print(".");
19      }
20      Serial.println("");
21      Serial.println("WiFi connected.");
```

Print out the IP address and port number of ESP32.

```
22      Serial.print("IP address: ");
23      Serial.println(WiFi.localIP());                 //print out IP address of ESP32
24      Serial.printf("IP port: %d\n",port);            //Print out ESP32's port number
```

Turn on server mode of ESP32, turn on automatic reconnection.

```
25      server.begin();                                 //Turn ON ESP32 as Server mode
26      WiFi.setAutoReconnect(true);
```

When ESP32 receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```
34      if (client.available()) {                       // if there's bytes to read from the
    client
35        Serial.println(client.readStringUntil('\n')); // print it out the serial monitor
36        while(client.read()>0);                       // clear the wifi receive area cache
37      }
38      if(Serial.available()){                         // if there's bytes to read from the
    serial monitor
39        client.print(Serial.readStringUntil('\n')); // print it out the client.
40        while(Serial.read()>0);                       // clear the wifi receive area cache
41      }
```

Reference

Class Server

Every time use Server functionality, we need to include header file"WiFi.h".

WiFiServer(uint16_t port=80, uint8_t max_clients=4): create a TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 80.

max_clients: maximum number of clients with default number as 4.

begin(port): start the TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 0.

setNoDelay(bool nodelay): whether to turn off the delay sending functionality.

nodelay: true stands for forbidden Nagle algorithm.

close(): close tcp connection.

stop(): stop tcp connection.

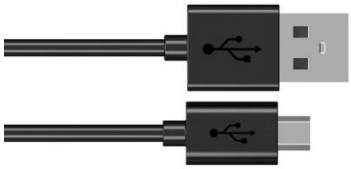
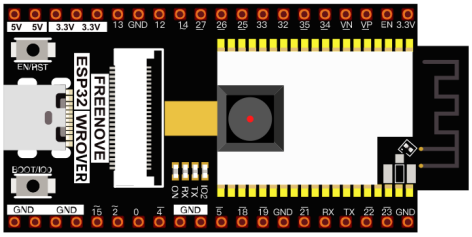
Chapter 17 Camera Web Server

In this section, we'll use ESP32's video function as an example to study.

Project 17.1 Camera Web Server

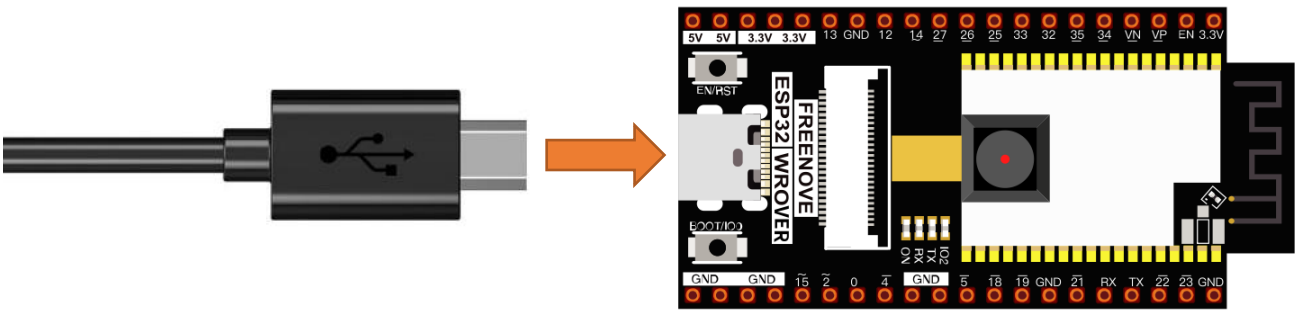
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

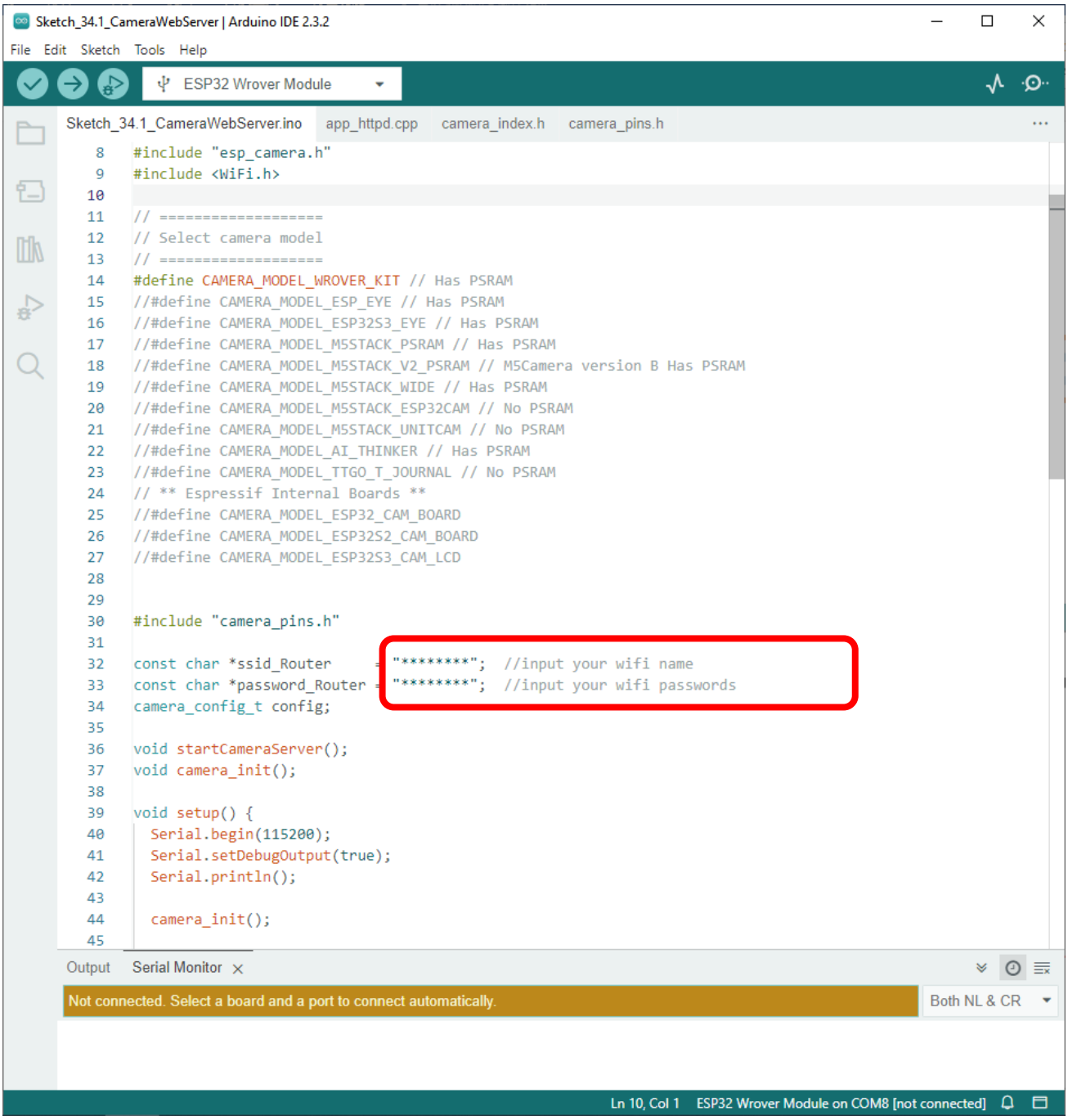
Circuit

Connect Freenove ESP32 to the computer using USB cable.



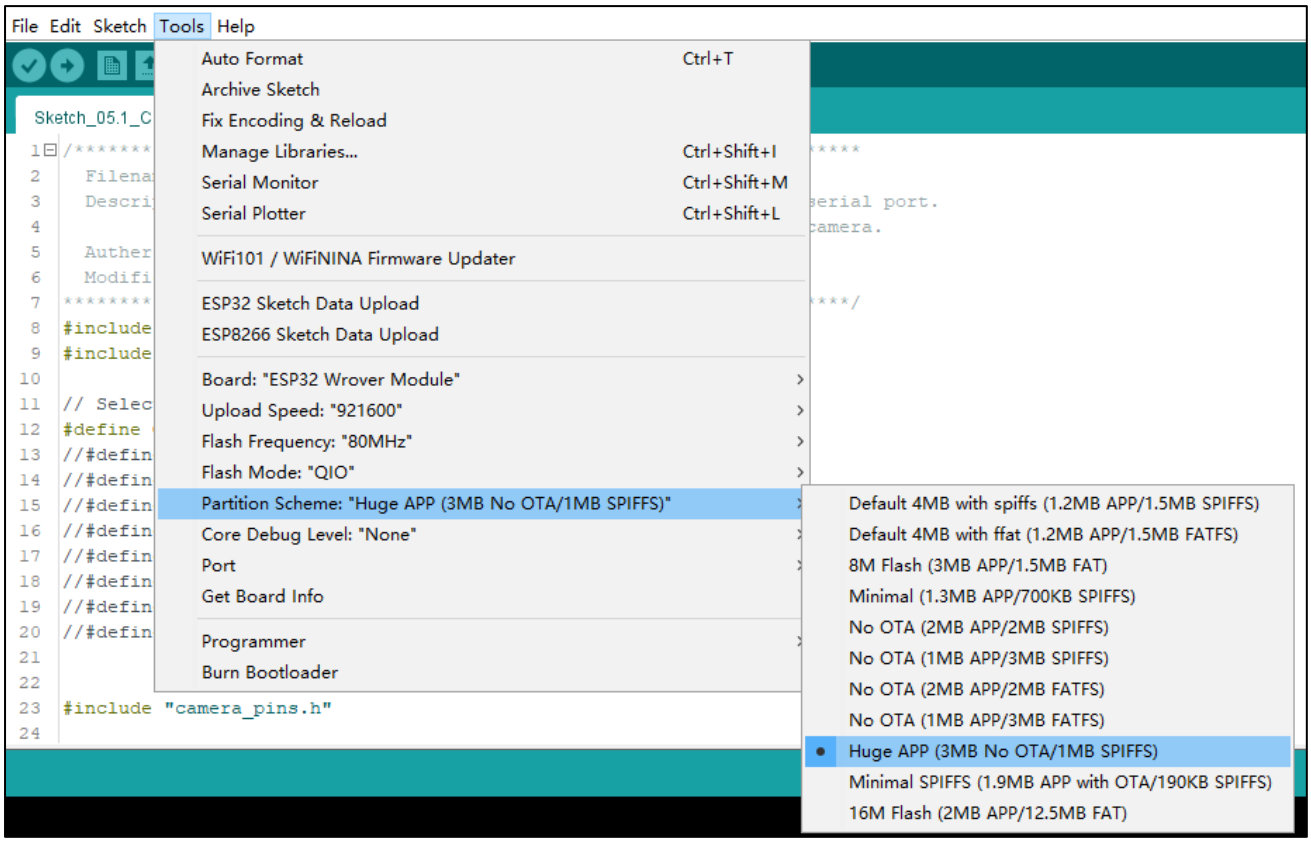
Sketch

Sketch_17.1_As_CameraWebServer

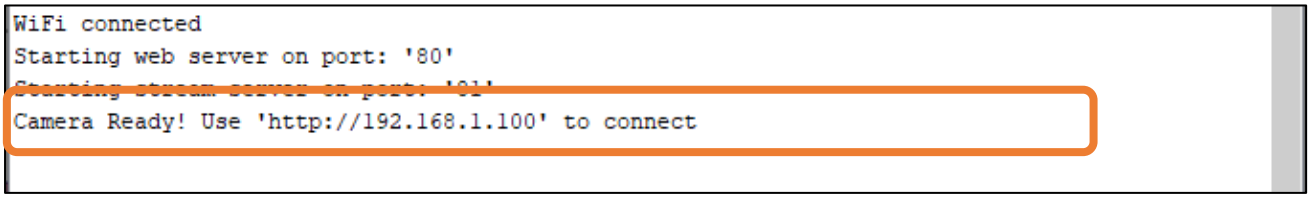


Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

If your Arduino IDE prompts you that your sketch is out of your project's storage space, compile the code again as configured below.



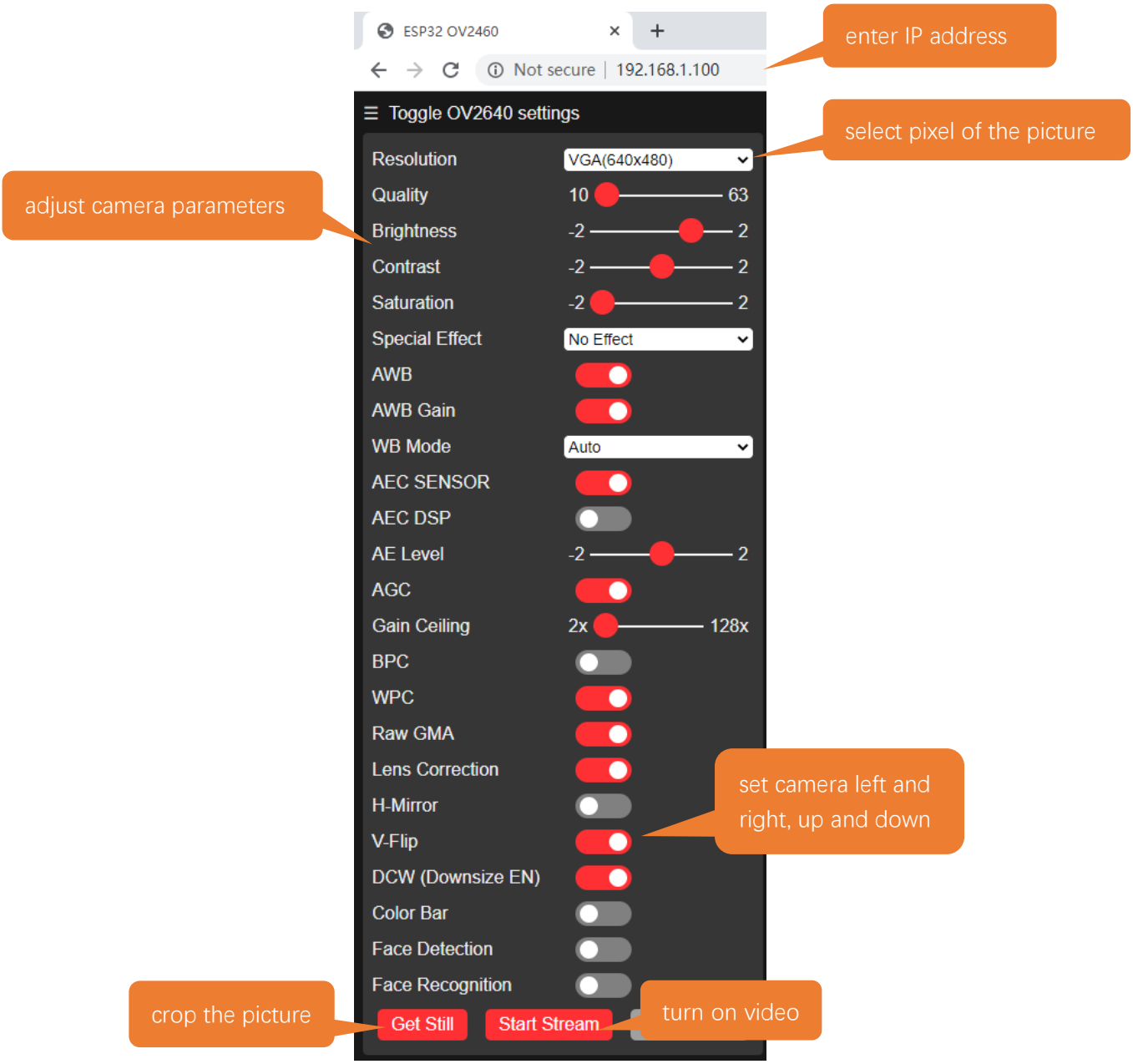
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.



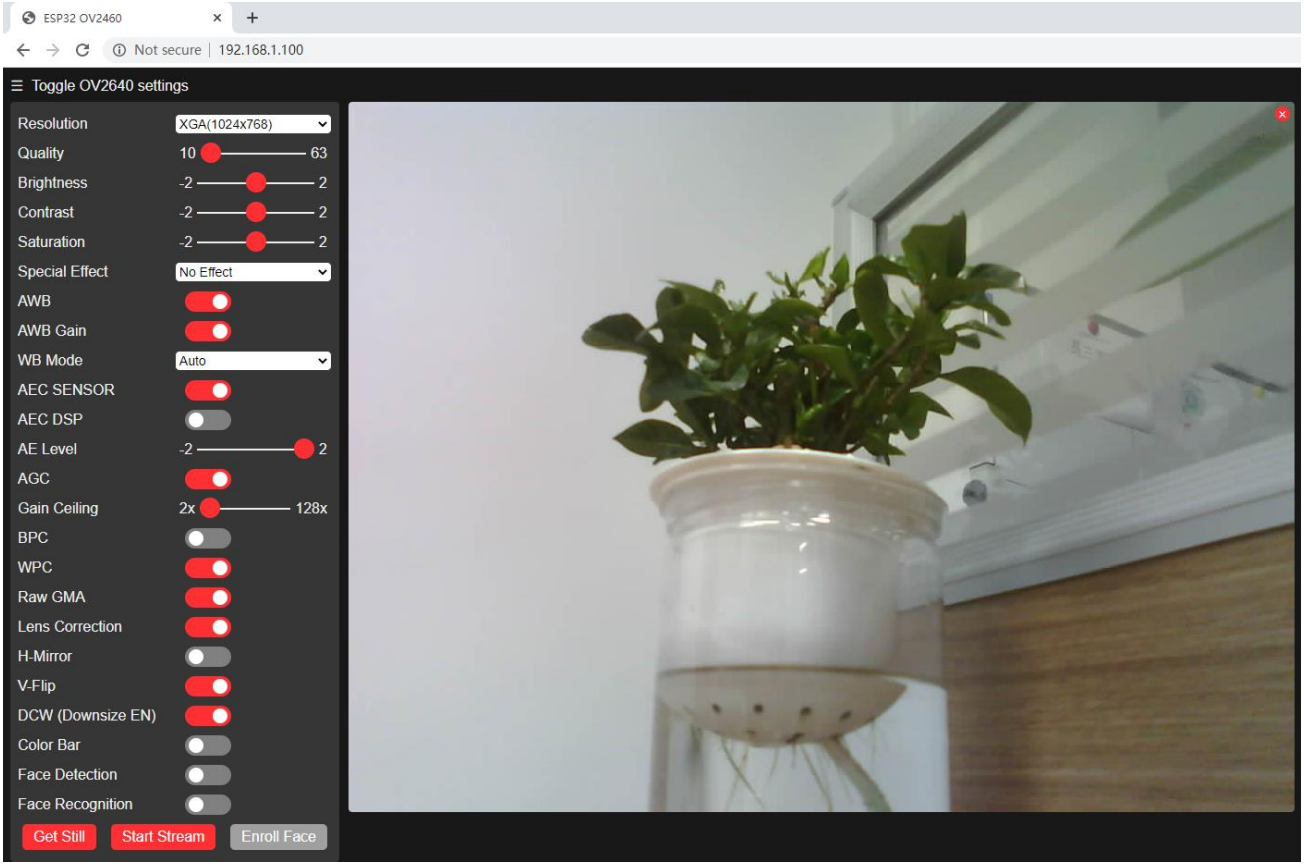
If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it. Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32's IP.

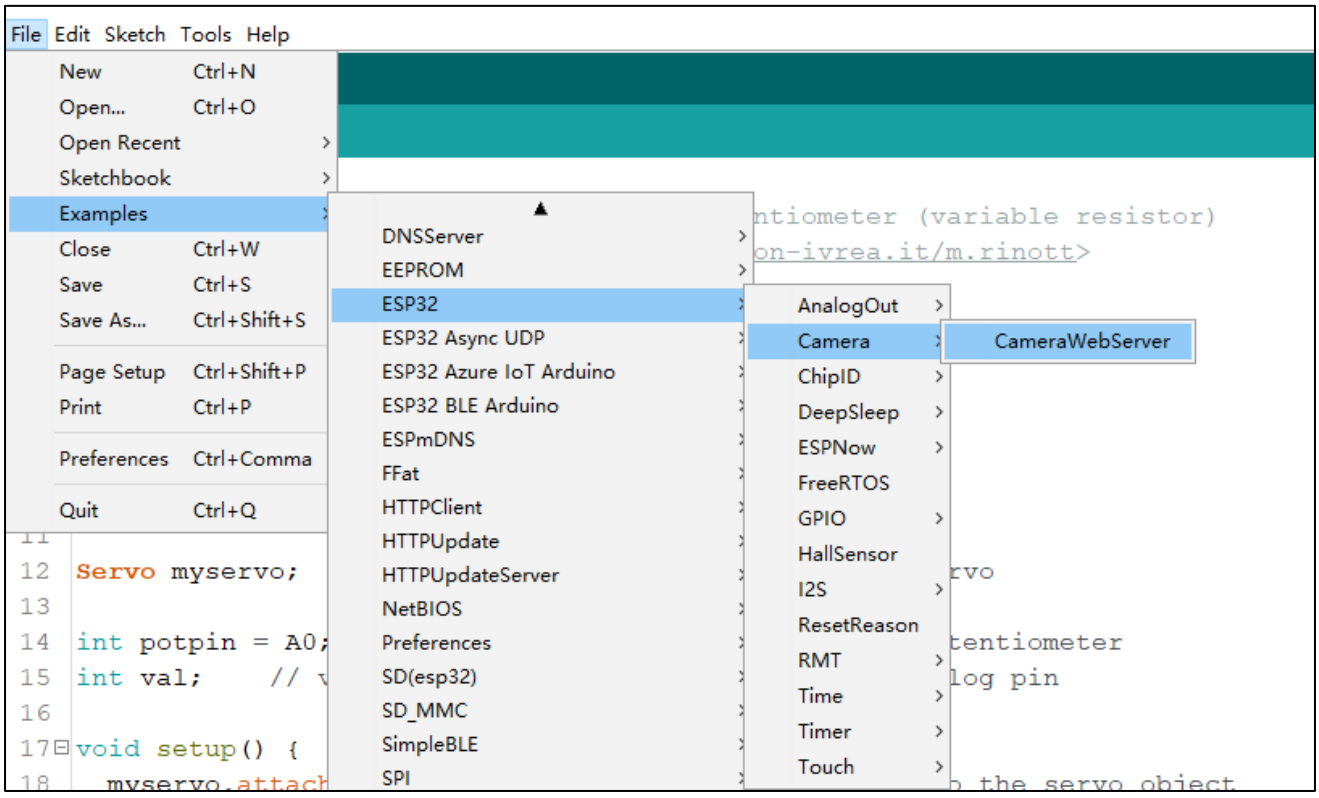
We recommend that the resolution not exceed VGA(640x480).



Click on Start Stream. The effect is shown in the image below.



Note: If sketch compilation fails due to ESP32 support package, follow the steps of the image to open the CameraWebServer. This sketch is the same as described in the tutorial above.



The following is the main program code. You need include other code files in the same folder when write your own code.

```
#include "esp_camera.h"
#include <WiFi.h>

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"

const char *ssid_Router     = "********";  //input your wifi name
const char *password_Router = "********";  //input your wifi passwords
camera_config_t config;

void startCameraServer();
void camera_init();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_init();
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 10;

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  s->set_vflip(s, 1);        //flip it back
  s->set_brightness(s, 1);   //up the blightness just a bit
  s->set_saturation(s, -1);  //lower the saturation

  WiFi.begin(ssid_Router, password_Router);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  ;
}

void camera_init(){
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 10000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.fb_count = 1;
}
```

Add procedure files and API interface files related to ESP32 camera.

```
#include "esp_camera.h"
#include <WiFi.h>

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"
```

Enter the name and password of the router

```
const char *ssid_Router     = "********";  //input your wifi name
const char *password_Router = "********";  //input your wifi passwords
```

Initialize serial port, set baud rate to 115200; open the debug and output function of the serial.

```
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
```

Configure parameters including interface pins of the camera. Note: It is generally not recommended to change them.

```
void camera_init(){
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 10000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.fb_count = 1;
}
```

ESP32 connects to the router and prints a successful connection prompt. If it has not been successfully connected, press the reset key on the ESP32-WROVER.

```
41    WiFi.begin(ssid_Router, password_Router);
42    while (WiFi.status() ! = WL_CONNECTED) {
43      delay(500);
44      Serial.print(".");
45    }
46    Serial.println("");
47    Serial.println("WiFi connected");
```

Open the video streams server function of the camera and print its IP address via serial port.

```
49    startCameraServer();
50
51    Serial.print("Camera Ready!  Use 'http://");
52    Serial.print(WiFi.localIP());
53    Serial.println("' to connect");
```

Configure the display image information of the camera.

The set_vflip() function sets whether the image is flipped 180°, with 0 for no flip and 1 for flip 180°.

The set_brightness() function sets the brightness of the image, with values ranging from -2 to 2.

The set_saturation() function sets the color saturation of the image, with values ranging from -2 to 2.

```
36    sensor_t * s = esp_camera_sensor_get();
37    s->set_vflip(s, 1);        //flip it back
38    s->set_brightness(s, 1);   //up the blightness just a bit
39    s->set_saturation(s, -1);  //lower the saturation
```

Modify the resolution and sharpness of the images captured by the camera. The sharpness ranges from 10 to 63, and the smaller the number, the sharper the picture. The larger the number, the blurrier the picture. Please refer to the table below.

```
26    config.frame_size = FRAMESIZE_VGA;
27    config.jpeg_quality = 10;
```

Reference

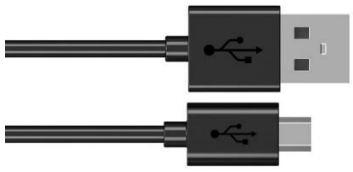
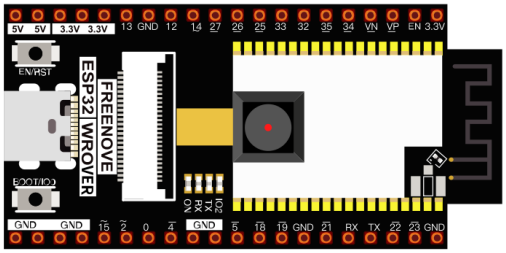
Image resolution	Sharpness	Image resolution	Sharpness
FRAMESIZE_96x96	96x96	FRAMESIZE_HVGA	480x320
FRAMESIZE_QQVGA	160x120	FRAMESIZE_VGA	640x480
FRAMESIZE_QCIF	176x144	FRAMESIZE_SVGA	800x600
FRAMESIZE_HQVGA	240x176	FRAMESIZE_XGA	1024x768
FRAMESIZE_240x240	240x240	FRAMESIZE_HD	1280x720
FRAMESIZE_QVGA	320x240	FRAMESIZE_SXGA	1280x1024
FRAMESIZE_CIF	400x296	FRAMESIZE_UXGA	1600x1200

We recommend that the resolution not exceed VGA(640x480).

Project 17.2 Video Web Server

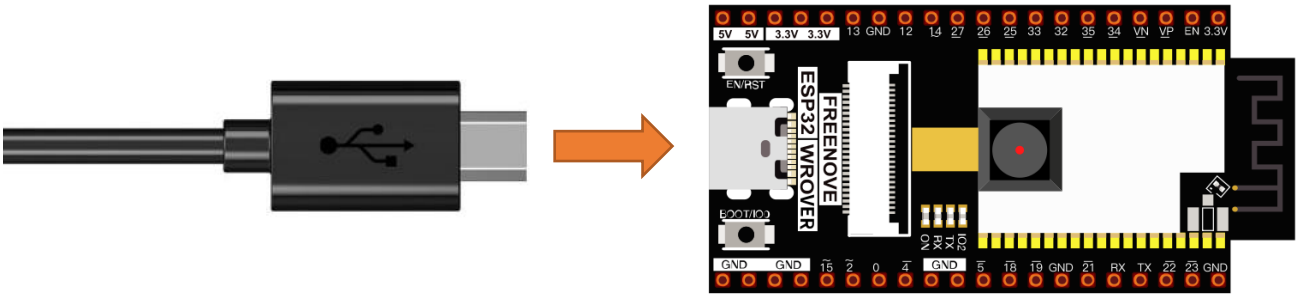
Connect to ESP32 using USB and view its IP address through a serial monitor. Access IP addresses through web pages to obtain real-time video data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Circuit

Connect Freenove ESP32 to the computer using USB cable.

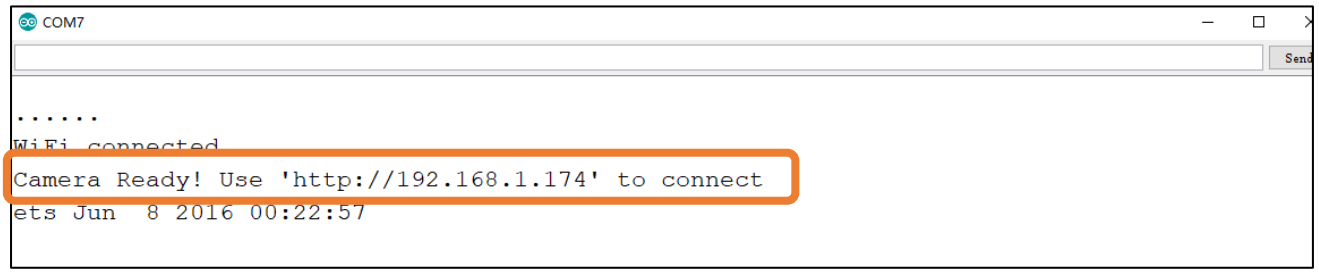


Sketch

Sketch_17.2_As_VideoWebServer

Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

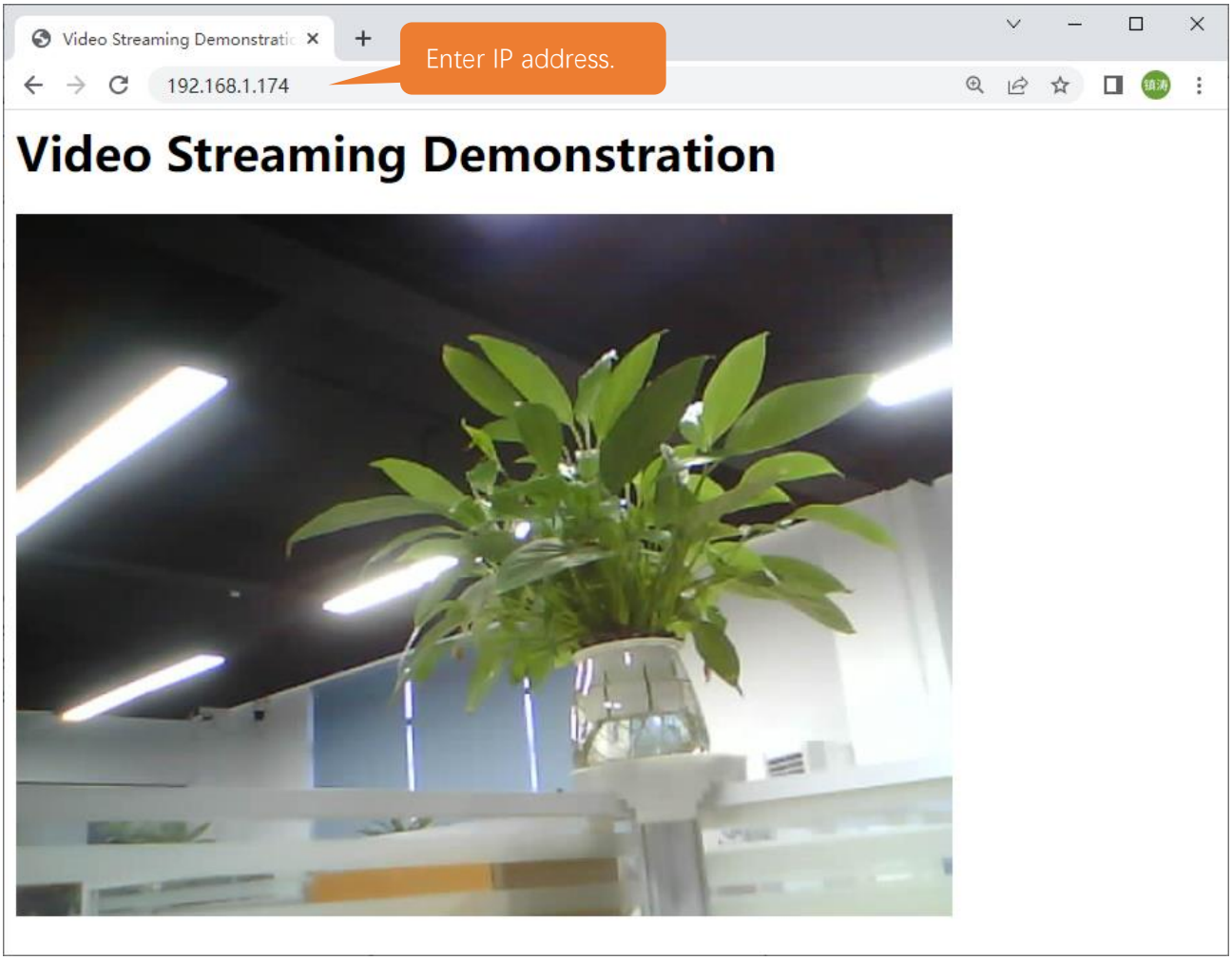
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.



If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it. Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32's IP.

The effect is shown in the image below.



The following is the main program code. You need include other code files in the same folder when write your own code.

```
#include "esp_camera.h"
#include <WiFi.h>
//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT // Has PSRAM

```

```
12  #include "camera_pins.h"
13
14  const char* ssid     = "********";   //input your wifi name
15  const char* password = "********";  //input your wifi passwords
16
17  void startCameraServer();
18
19  void setup() {
20    Serial.begin(115200);
21    Serial.setDebugOutput(true);
22    Serial.println();
23
24    camera_config_t config;
25    config.ledc_channel = LEDC_CHANNEL_0;
26    config.ledc_timer = LEDC_TIMER_0;
27    config.pin_d0 = Y2_GPIO_NUM;
28    config.pin_d1 = Y3_GPIO_NUM;
29    config.pin_d2 = Y4_GPIO_NUM;
30    config.pin_d3 = Y5_GPIO_NUM;
31    config.pin_d4 = Y6_GPIO_NUM;
32    config.pin_d5 = Y7_GPIO_NUM;
33    config.pin_d6 = Y8_GPIO_NUM;
34    config.pin_d7 = Y9_GPIO_NUM;
35    config.pin_xclk = XCLK_GPIO_NUM;
36    config.pin_pclk = PCLK_GPIO_NUM;
37    config.pin_vsync = VSYNC_GPIO_NUM;
38    config.pin_href = HREF_GPIO_NUM;
39    config.pin_sccb_sda = SIOD_GPIO_NUM;
40    config.pin_sccb_scl = SIOC_GPIO_NUM;
41    config.pin_pwdn = PWDN_GPIO_NUM;
42    config.pin_reset = RESET_GPIO_NUM;
43    config.xclk_freq_hz = 10000000;
44    config.pixel_format = PIXFORMAT_JPEG;
45
46    // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
47    //                      for larger pre-allocated frame buffer.
48    if(psramFound()){
49      config.frame_size = FRAMESIZE_VGA;
50      config.jpeg_quality = 10;
51      config.fb_count = 2;
52    } else {
53      config.frame_size = FRAMESIZE_HVGA;
54      config.jpeg_quality = 12;
55      config.fb_count = 1;
```

```
  }

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  // drop down frame size for higher initial frame rate
  s->set_framesize(s, FRAMESIZE_QVGA);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

Configure parameters including interface pins of the camera. Note: It is generally not recommended to change them.

```
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
```

```
32    config.pin_d5 = Y7_GPIO_NUM;
33    config.pin_d6 = Y8_GPIO_NUM;
34    config.pin_d7 = Y9_GPIO_NUM;
35    config.pin_xclk = XCLK_GPIO_NUM;
36    config.pin_pclk = PCLK_GPIO_NUM;
37    config.pin_vsync = VSYNC_GPIO_NUM;
38    config.pin_href = HREF_GPIO_NUM;
39    config.pin_sccb_sda = SIOD_GPIO_NUM;
40    config.pin_sccb_scl = SIOC_GPIO_NUM;
41    config.pin_pwdn = PWDN_GPIO_NUM;
42    config.pin_reset = RESET_GPIO_NUM;
43    config.xclk_freq_hz = 10000000;
44    config.pixel_format = PIXFORMAT_JPEG;
```

ESP32 connects to the router and prints a successful connection prompt. If it has not been successfully connected, press the reset key on the ESP32-WROVER.

```
69    WiFi.begin(ssid, password);
70
71    while (WiFi.status() != WL_CONNECTED) {
72      delay(500);
73      Serial.print(".");
74    }
75    Serial.println("");
76    Serial.println("WiFi connected");
```

Open the video streams server function of the camera and print its IP address via serial port.

```
78    startCameraServer();
79
80    Serial.print("Camera Ready! Use 'http://");
81    Serial.print(WiFi.localIP());
82    Serial.println("' to connect");
```

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 ESP32, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_ESP32/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.