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

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

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

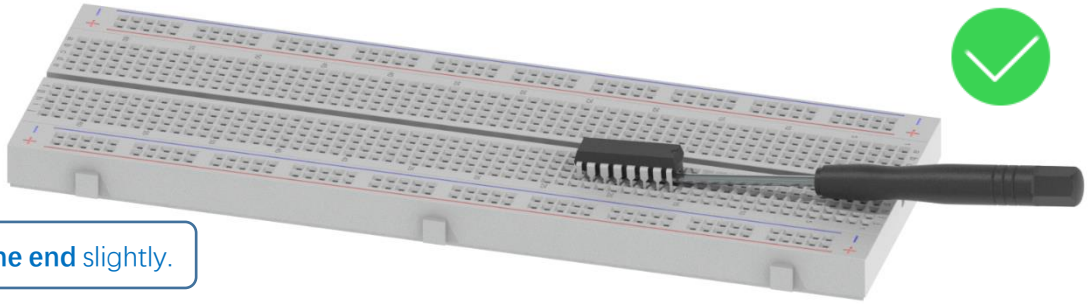
Freenove brand and logo are copyright of Freenove Creative Technology Co., Ltd. and cannot be used without written permission.

TM

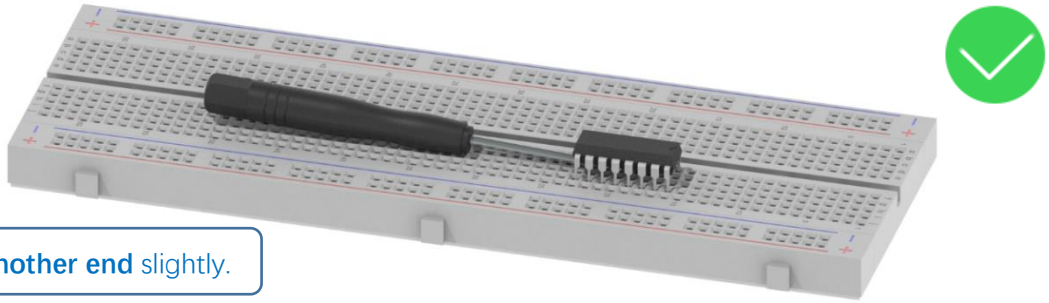
FREENOVE

Remove the Chips

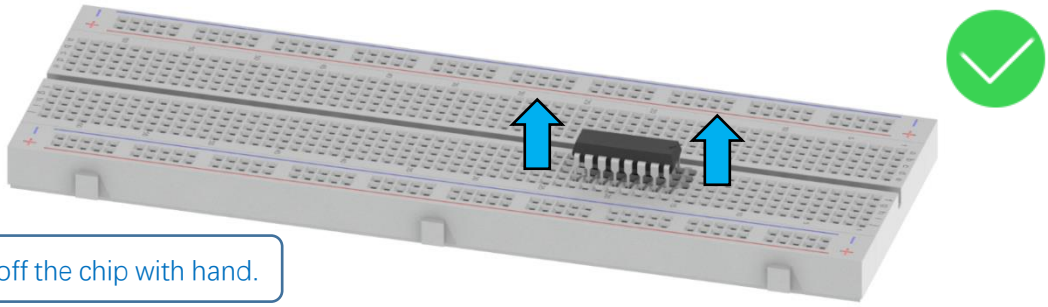
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



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

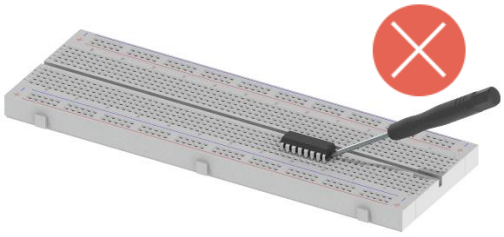


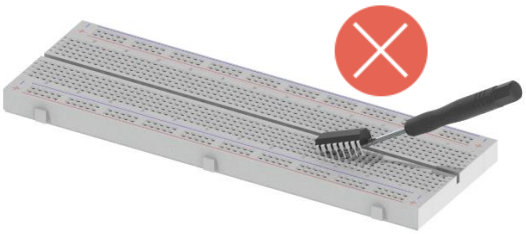
Step 2, lift **another end** slightly.



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 LEDPixel.....68
Project 6.1 LEDPixel.....68
Project 6.2 Rainbow Light.....75
Chapter 7 Buzzer.....77

Project 7.1 Doorbell....77
Project 7.2 Alertor....83
Project 7.3 Alertor (use timer)....86
Chapter 8 Serial Communication....90
Project 8.1 Serial Print....90
Project 8.2 Serial Read and Write....95
Chapter 9 AD/DA Converter....98
Project 9.1 Read the Voltage of Potentiometer....98
Chapter 10 Touch Sensor....105
Project 10.1 Read Touch Sensor....105
Project 10.2 Touch Lamp....110
Chapter 11 Potentiometer & LED....115
Project 11.1 Soft Light....115
Project 11.2 Soft Colorful Light....118
Project 11.3 Soft Rainbow Light....121
Chapter 12 Photoresistor & LED....125
Project 12.1 NightLamp....125
Chapter 13 Thermistor....129
Project 13.1 Thermometer....129
Chapter 14 Joystick....134
Project 14.1 Joystick....134
Chapter 15 74HC595 & LED Bar Graph....139
Project 15.1 Flowing Water Light....139
Chapter 16 74HC595 & 7-Segment Display....145
Project 16.1 7-Segment Display....145
Project 16.2 4-Digit 7-Segment Display....152
Chapter 16 74HC595 & LED Matrix....159
Project 16.3 LED Matrix....159
Chapter 17 Relay & Motor....168
Project 17.1 Relay & Motor....168
Chapter 17.2 Motor & Driver....175
Project 17.2 Control Motor with Potentiometer....175

Chapter 18 Servo 183
Project 18.1 Servo Sweep 183
Project 18.2 Servo Knop 189
Chapter 19 Stepper Motor 193
Project 19.1 Stepper Motor 193
Chapter 20 LCD1602 201
Project 20.1 LCD1602 201
Chapter 21 Ultrasonic Ranging 210
Project 21.1 Ultrasonic Ranging 210
Project 21.2 Ultrasonic Ranging 216
Chapter 22 Matrix Keypad 219
Project 22.1 Matrix Keypad 219
Project 22.2 Keypad Door 226
Chapter 23 Infrared Remote 232
Project 23.1 Infrared Remote Control 232
Project 23.2 Control LED through Infrared Remote 239
Chapter 24 Hygrothermograph DHT11 244
Project 24.1 Hygrothermograph 244
Project 24.2 Hygrothermograph 250
Chapter 25 Infrared Motion Sensor 255
Project 25.1 Infrared Motion Detector with LED Indicator 255
Chapter 26 Attitude Sensor MPU6050 260
Project 26.1 Read a MPU6050 Sensor Module 260
Chapter 27 Bluetooth 268
Project 27.1 Bluetooth Passthrough 268
Project 27.2 Bluetooth Low Energy Data Passthrough 274
Project 27.3 Bluetooth Control LED 286
Chapter 28 Bluetooth Media by DAC 292
Project 28.1 Playing Bluetooth Music through DAC 292
Chapter 29 Bluetooth Media by Audio Module 299
Project 29.1 Playing Bluetooth Music through Audio Converter & Amplifier 299

Chapter 30 Read and Write the Sdcard 307
Project 30.1 SDMMC Test 307
Chapter 31 Play SD card music 318
Project 31.1 SDMMC Music 318
Project 31.2 SDMMC Music 326
Chapter 32 WiFi Working Modes 335
Project 32.1 Station mode 335
Project 32.2 AP mode 339
Project 30.3 AP+Station mode 344
Chapter 33 TCP/IP 348
Project 33.1 As Client 348
Project 33.2 As Server 360
Chapter 34 Camera Web Server 366
Project 34.1 Camera Web Server 366
Project 34.2 Video Web Server 375
Chapter 35 Camera Tcp Server 381
Project 35.1 Camera Tcp Server 381
Chapter 36 Soldering Circuit Board 398
Project 36.1 Soldering a Buzzer 398
Project 34.2 Soldering a Flowing Water Light 402
What's next? 406
End of the Tutorial 406

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**

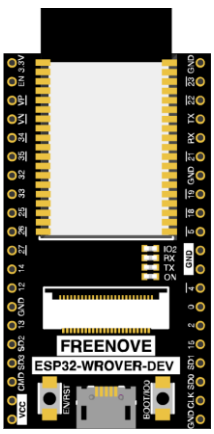
ESP32-WROVER

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.

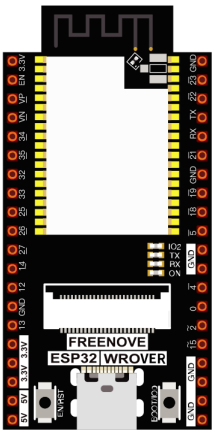
PCB on-board antenna	IPEX antenna

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

ESP32-WROVER

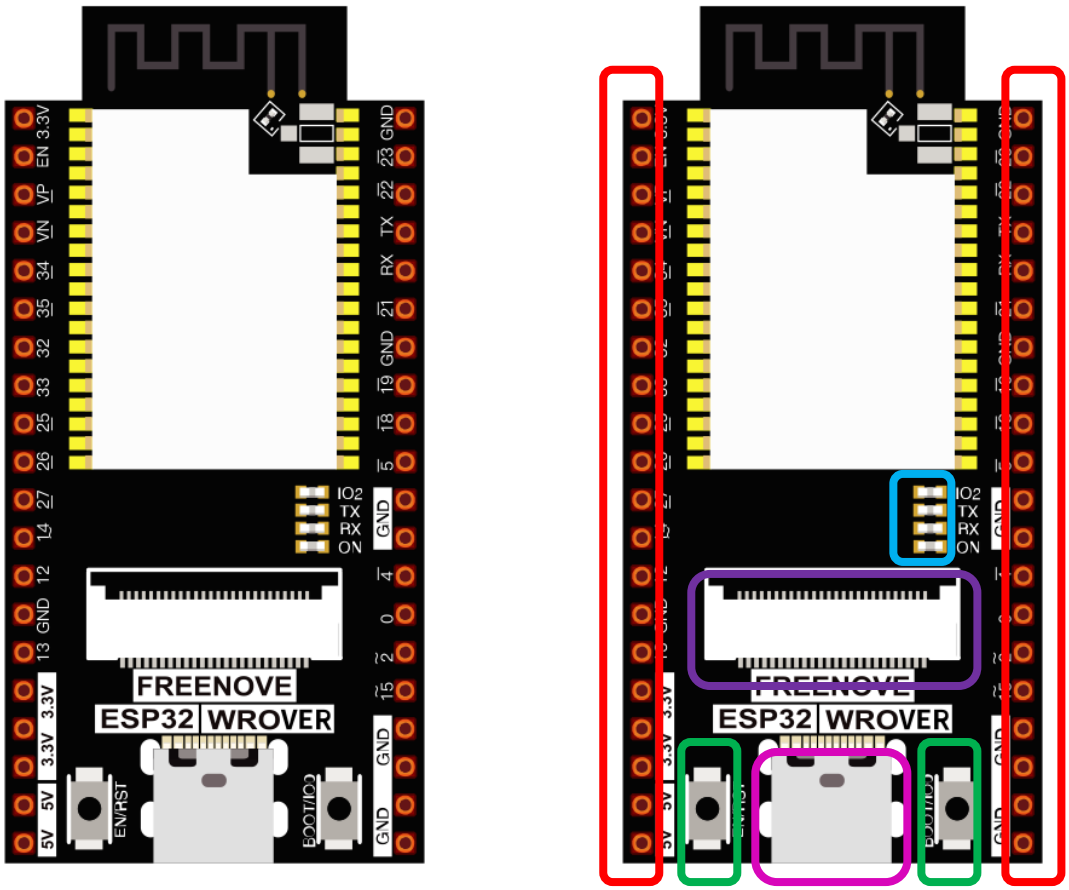


or



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

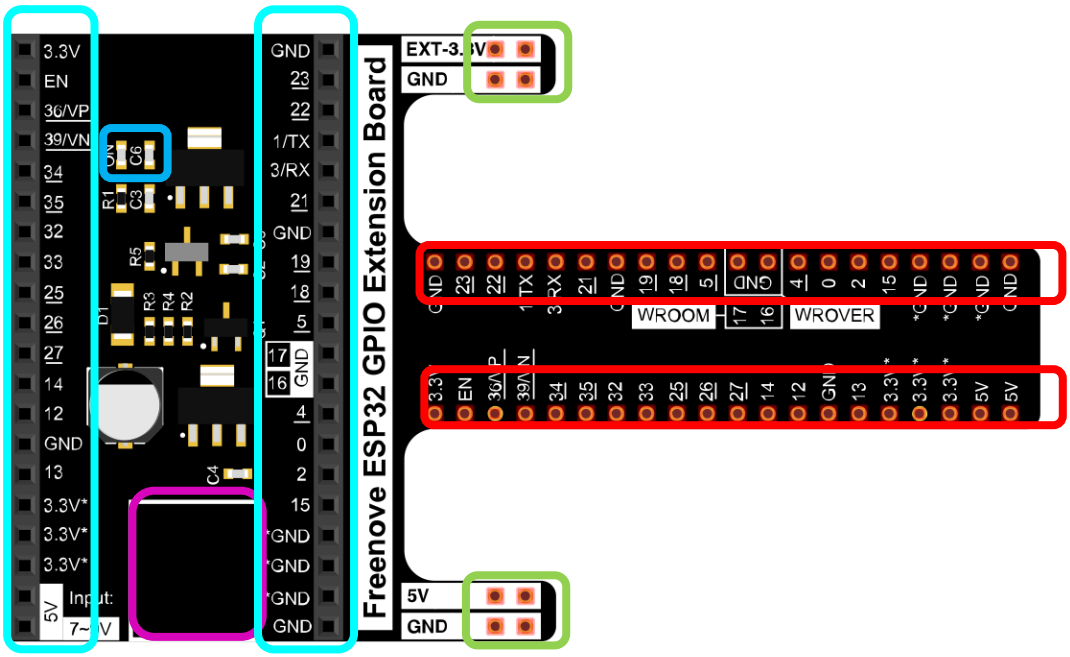
Notice:

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.

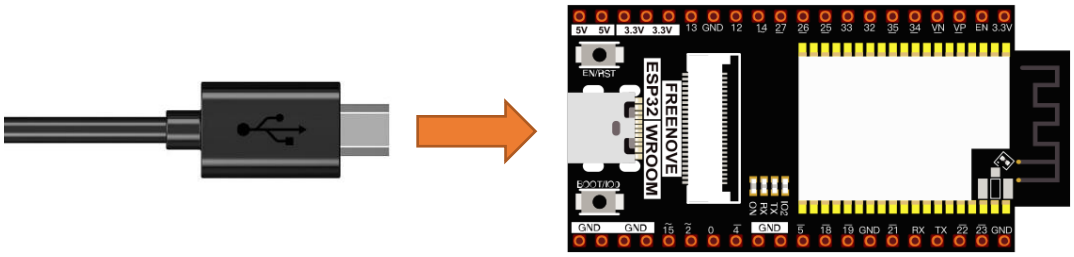
CH340 (Importance)

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

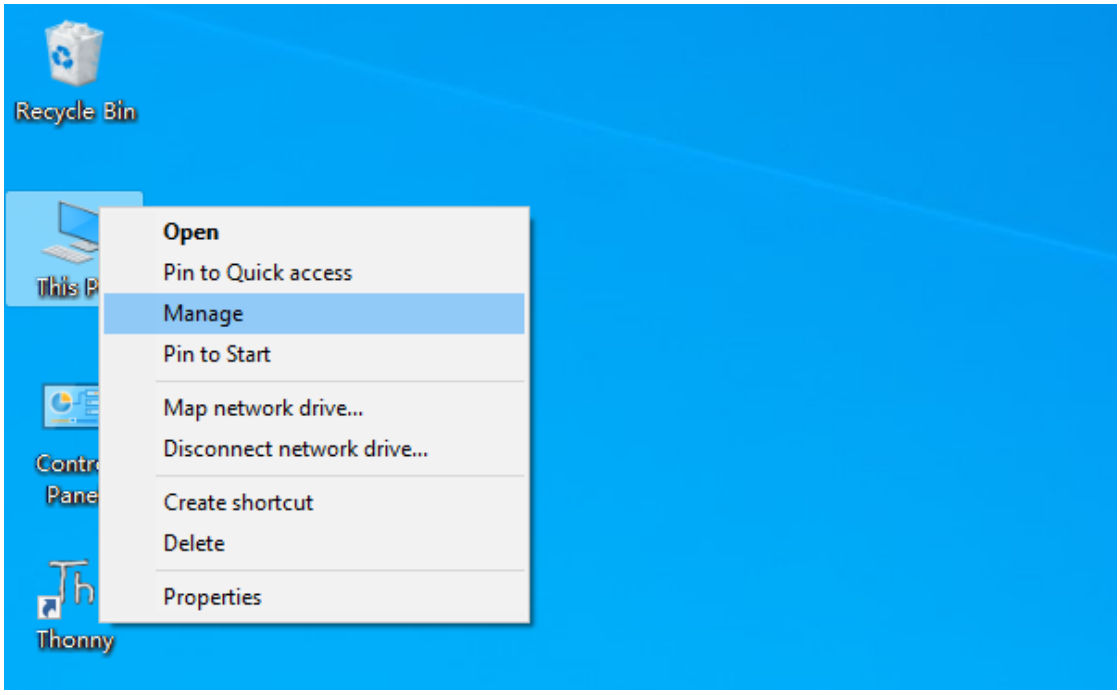
Windows

Check whether CH340 has been installed

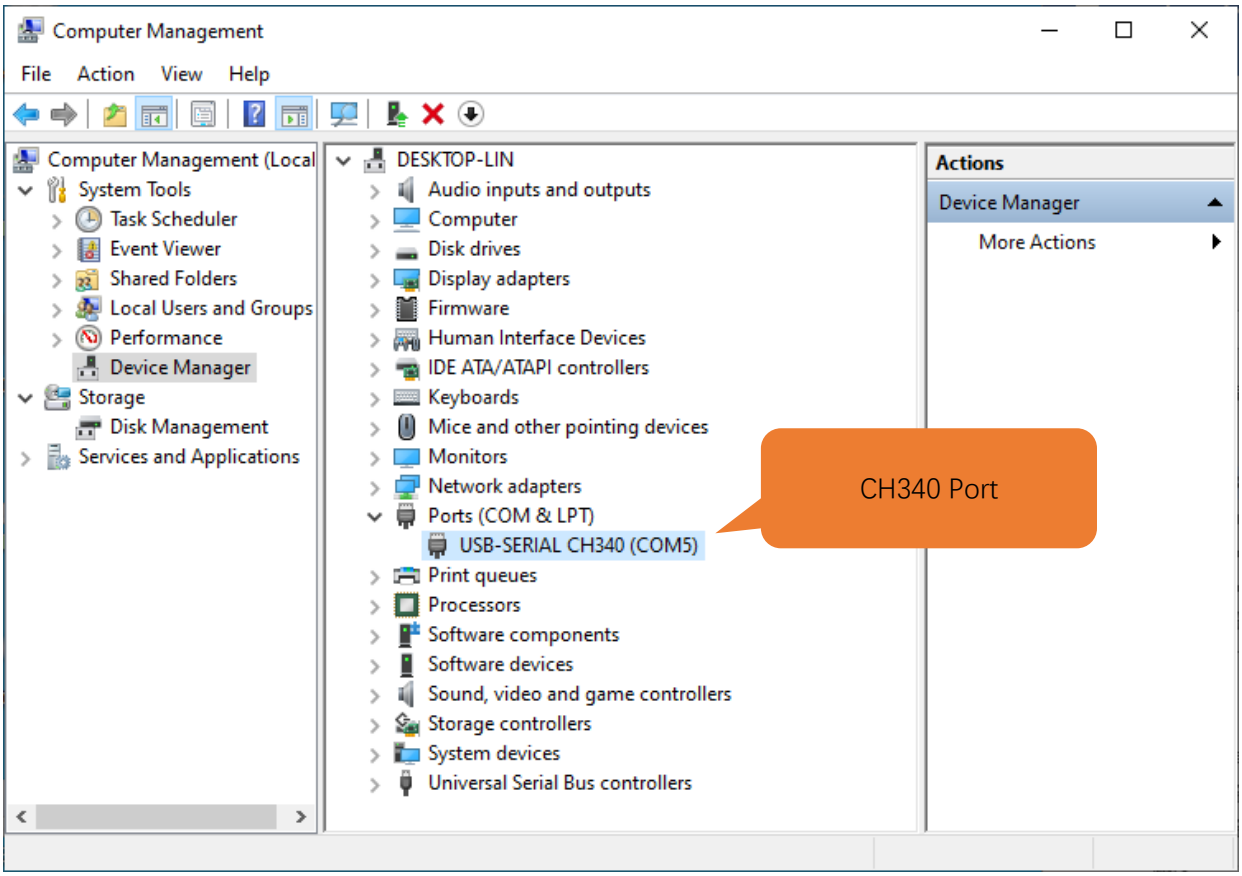
1. Connect your computer and ESP32 with a USB cable.



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

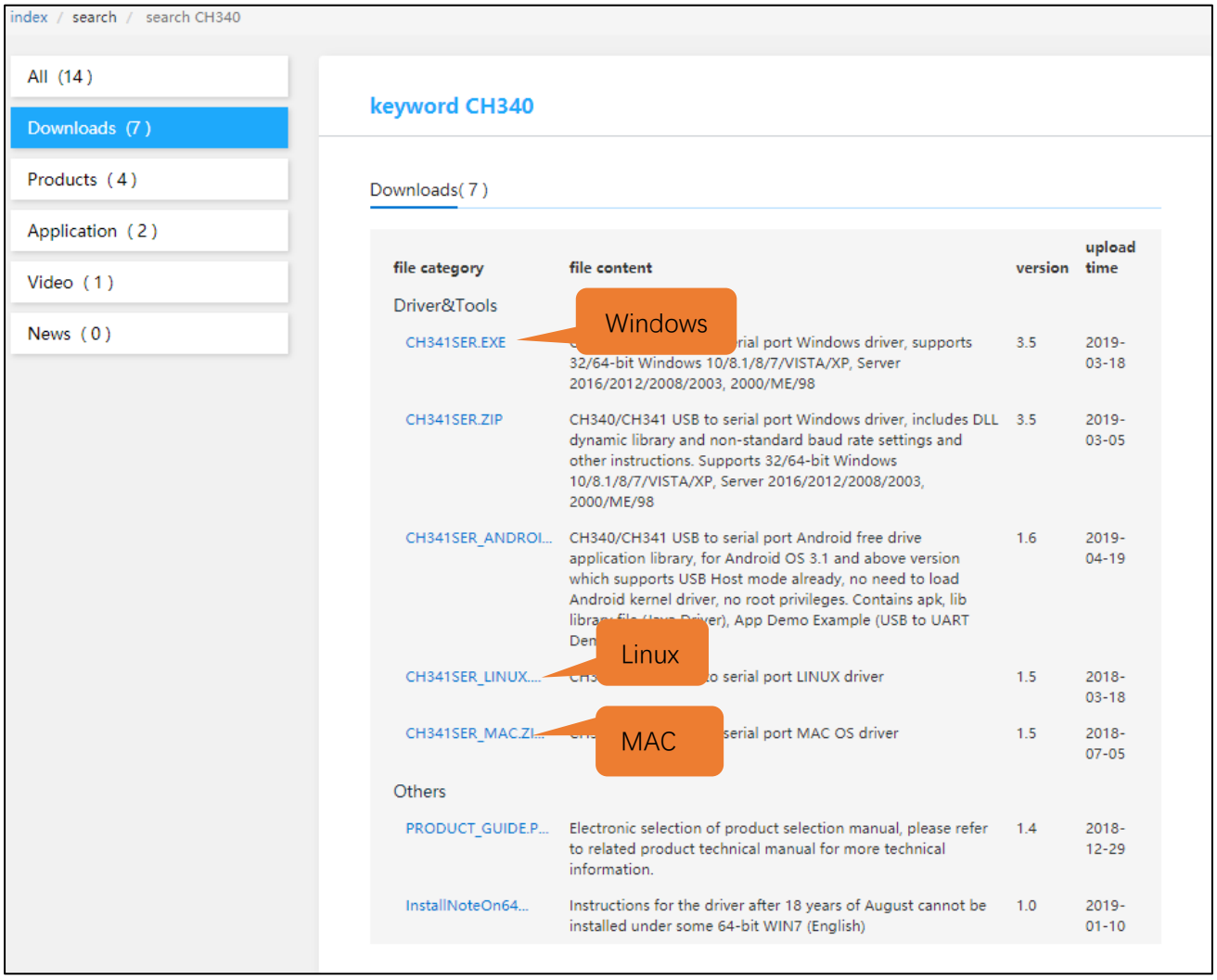


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

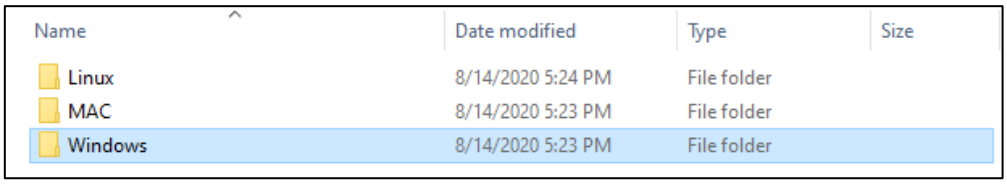


Installing CH340

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.

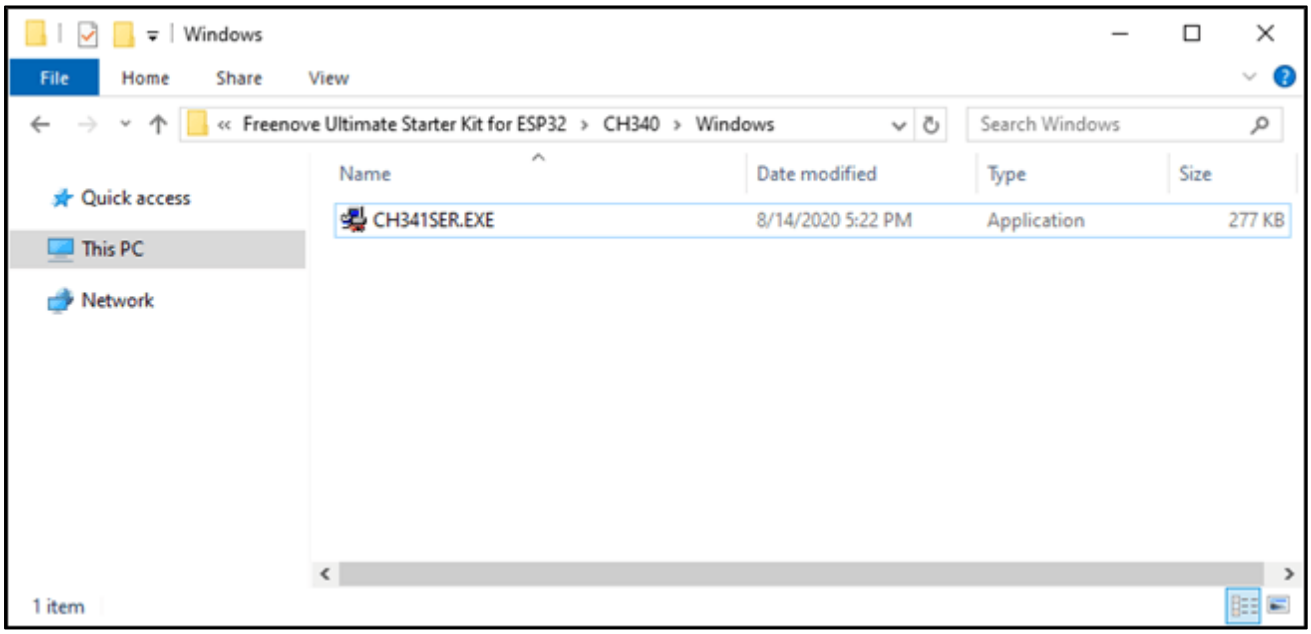


If you would not like to download the installation package, you can open **"Freenove_Ultimate_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.

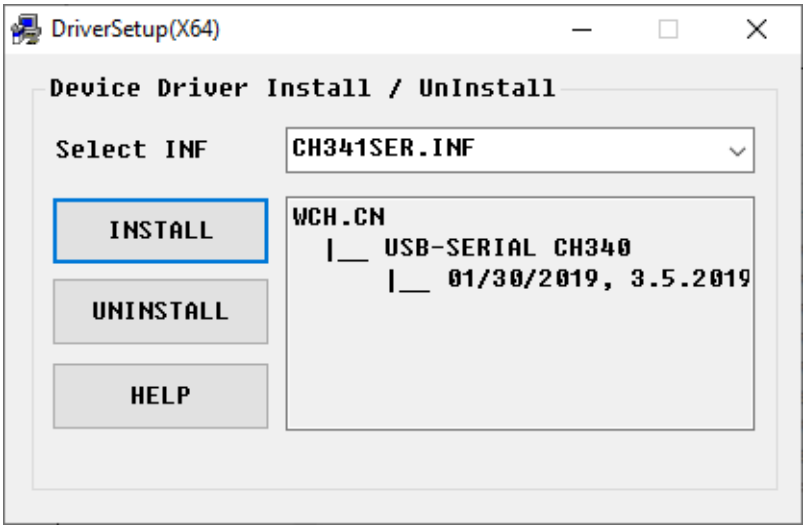


Name	Date modified	Type	Size
Linux	8/14/2020 5:24 PM	File folder	
MAC	8/14/2020 5:23 PM	File folder	
Windows	8/14/2020 5:23 PM	File folder	

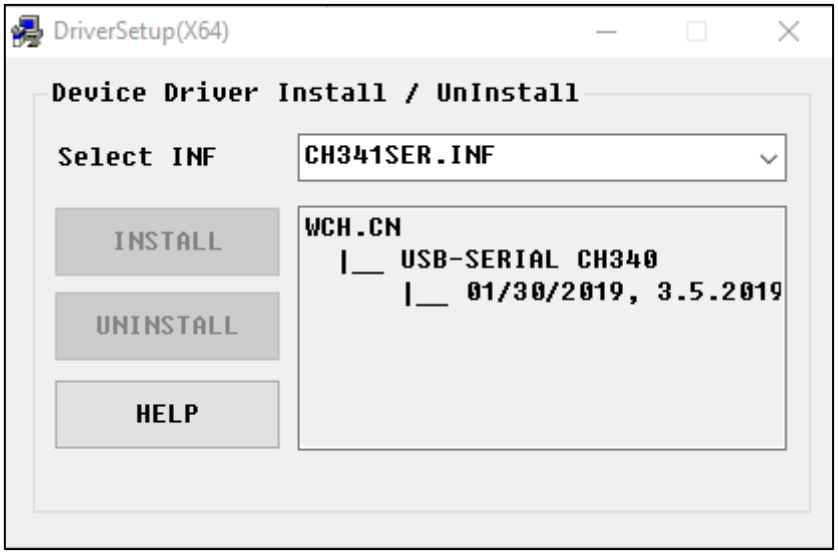
2. Open the folder "**Freenove_Ultimate_Starter_Kit_for_ESP32/CH340/Windows/**"



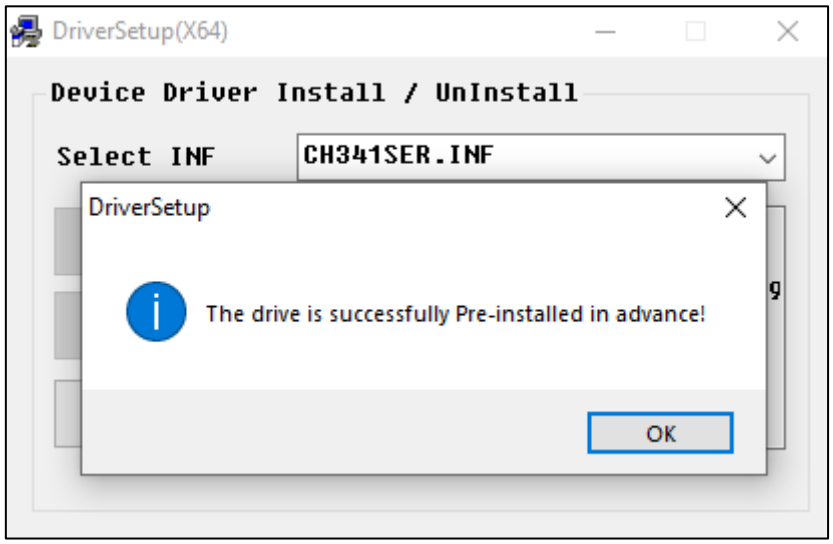
3. Double click "**CH341SER.EXE**".



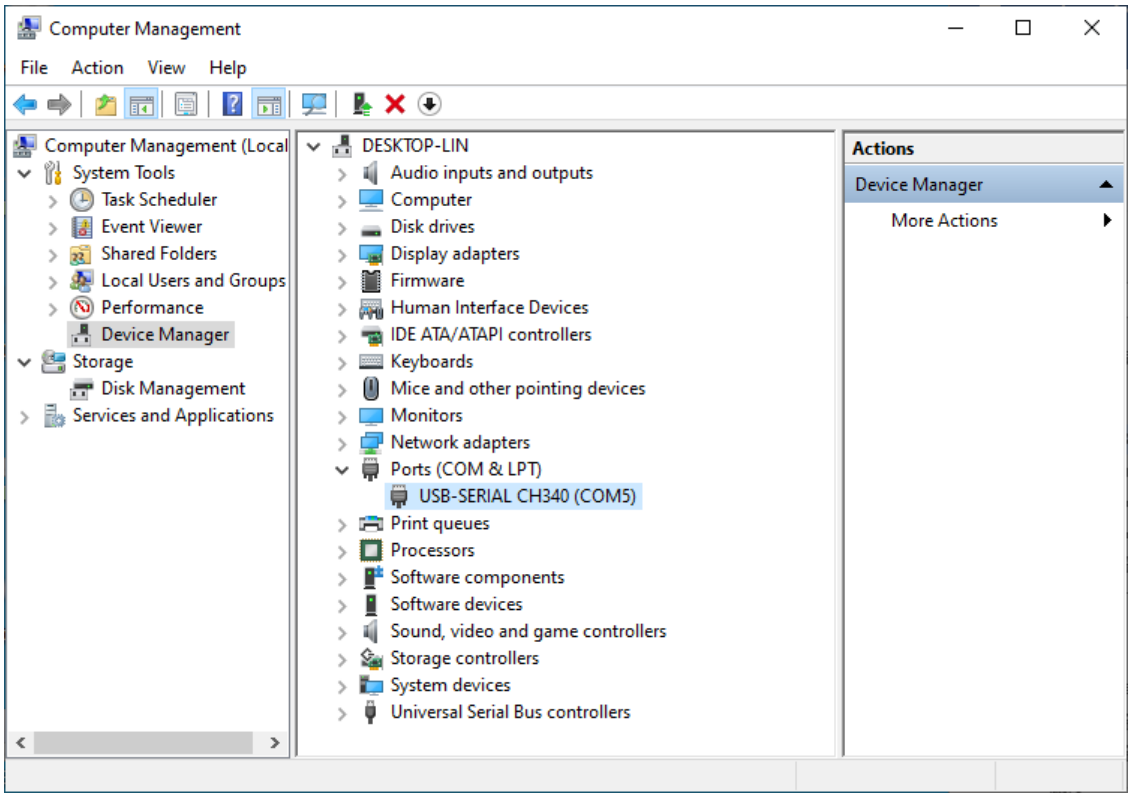
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



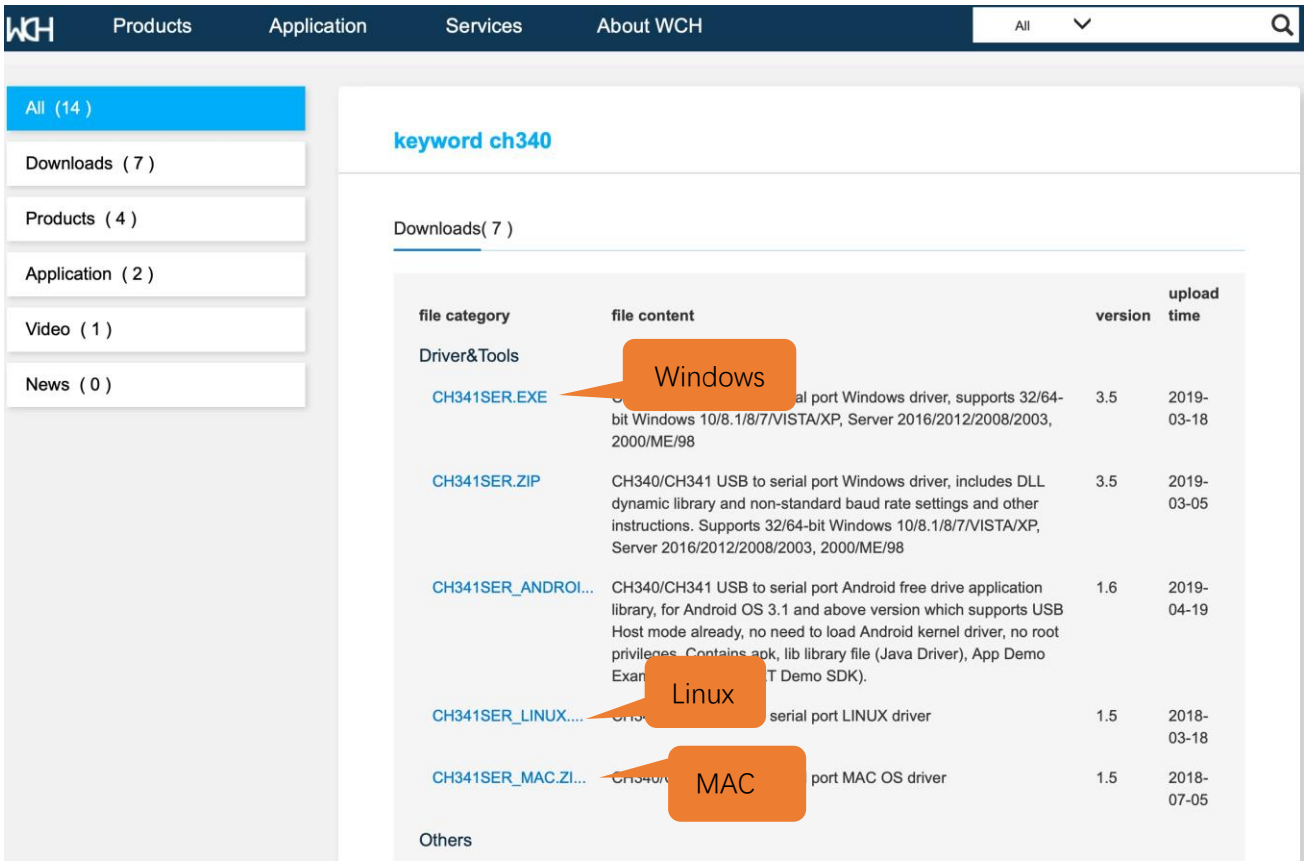
6. When ESP32 is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



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

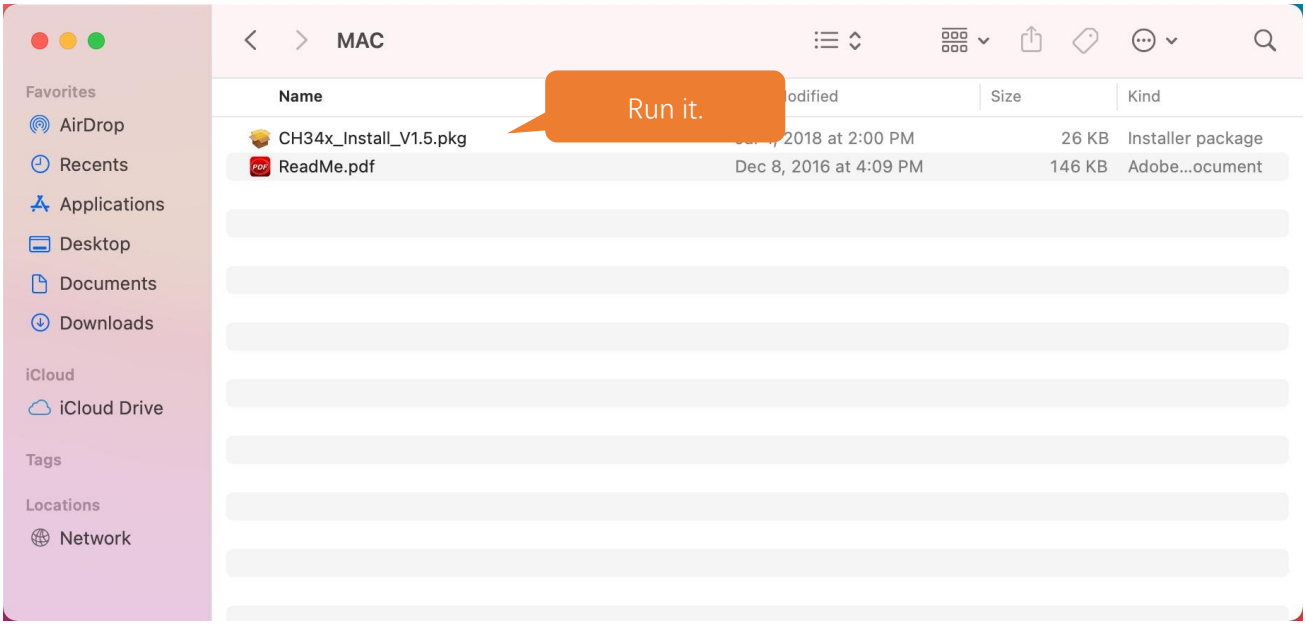
MAC

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.

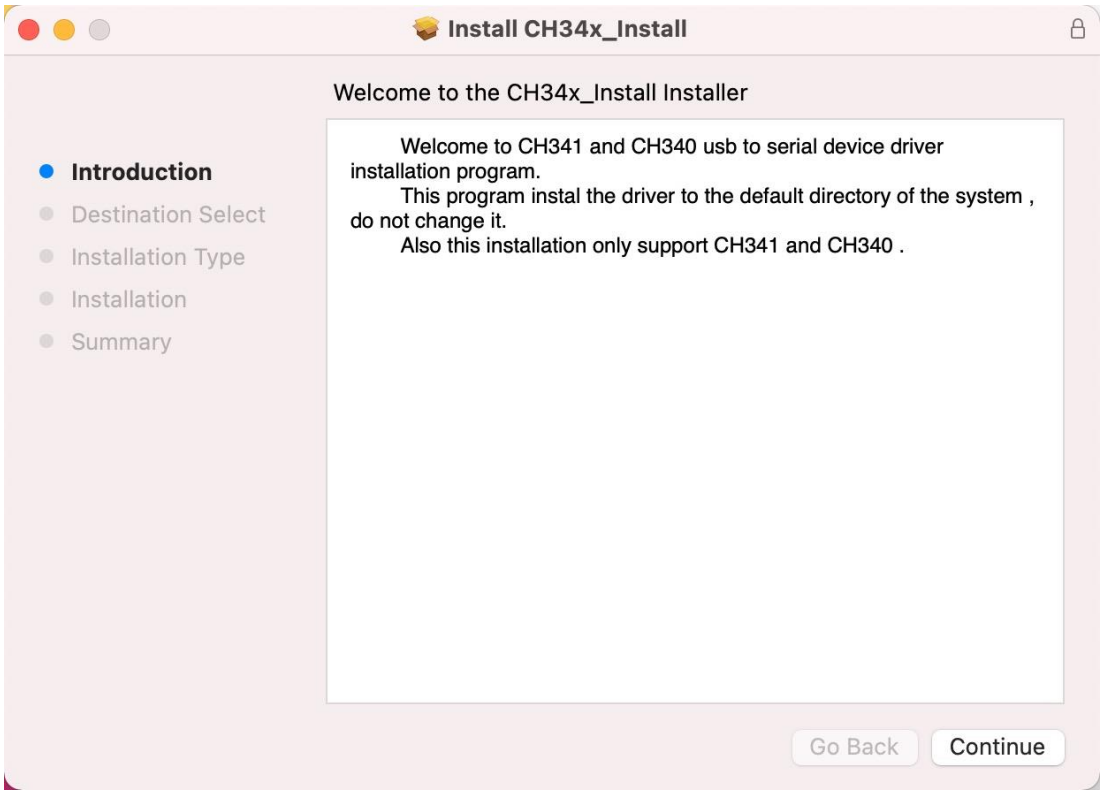


If you would not like to download the installation package, you can open **"Freenove_Ultimate_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.

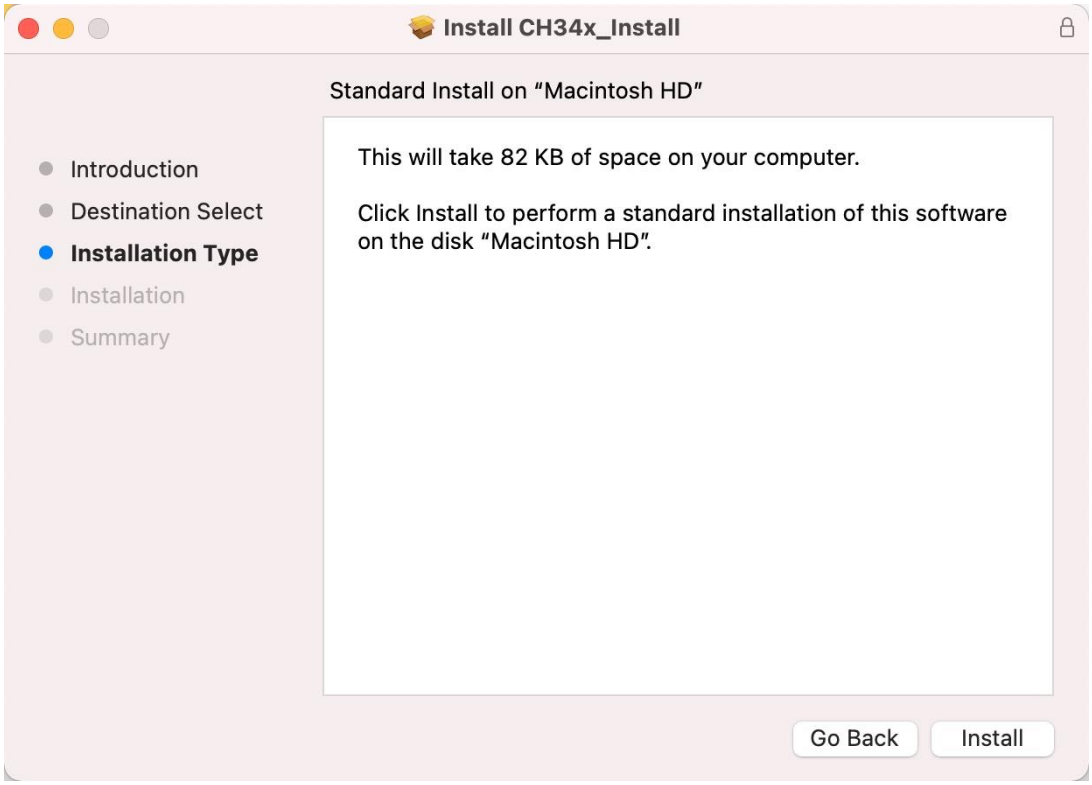
Second, open the folder "**Freenove_Ultimate_Starter_Kit_for_ESP32/CH340/MAC/**"



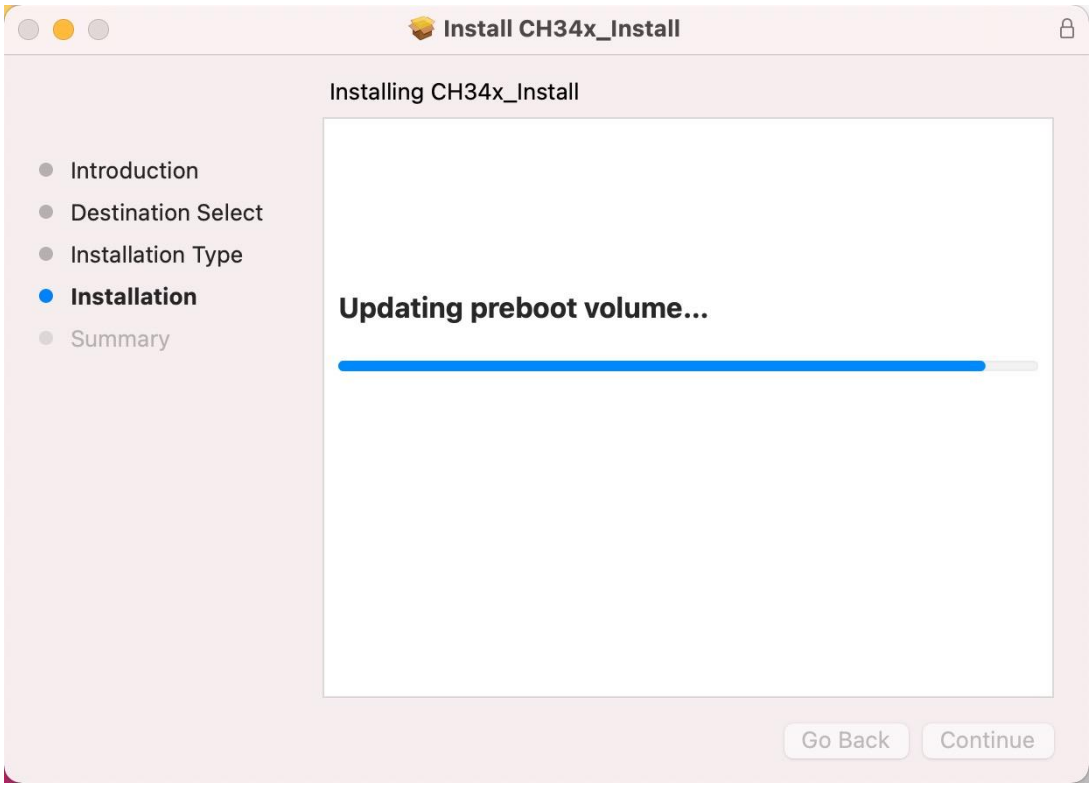
Third, click Continue.



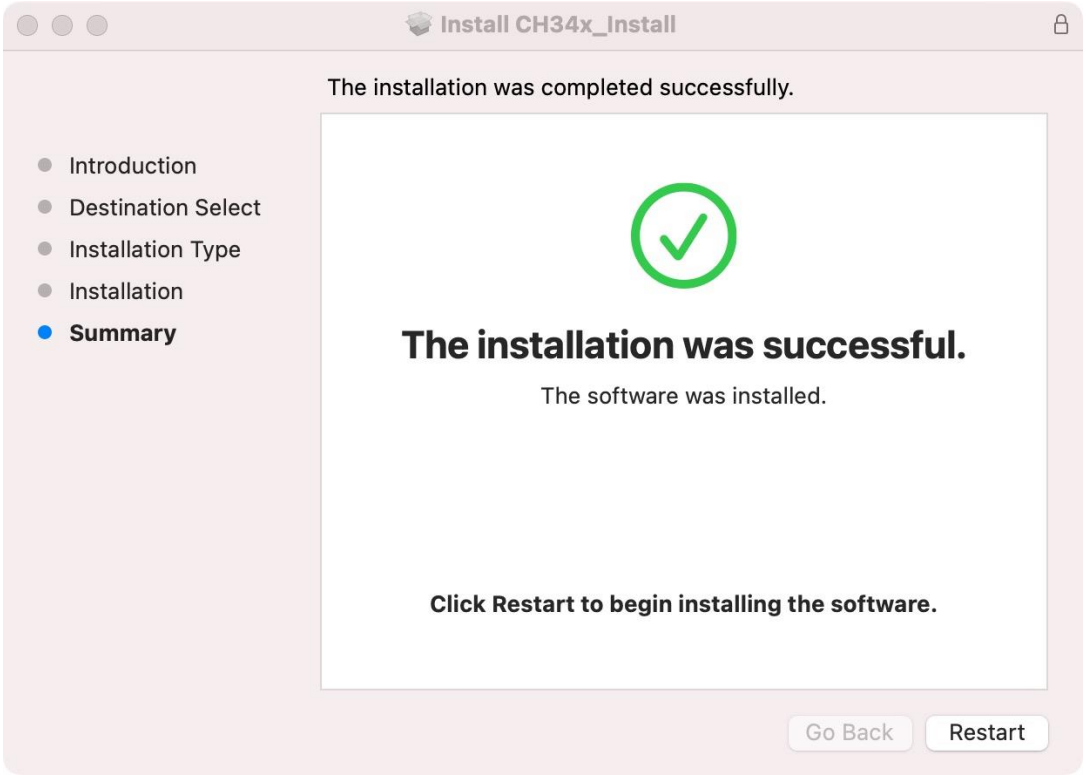
Fourth, click Install.



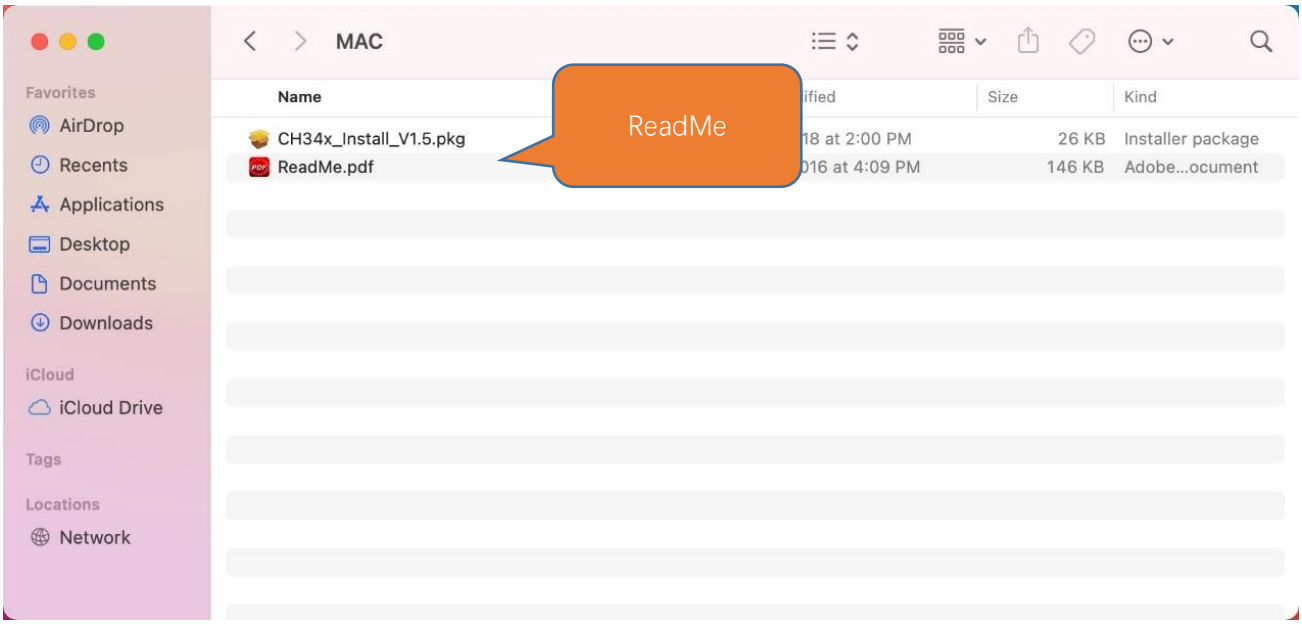
Then, waiting Finsh.



Finally, restart your PC.



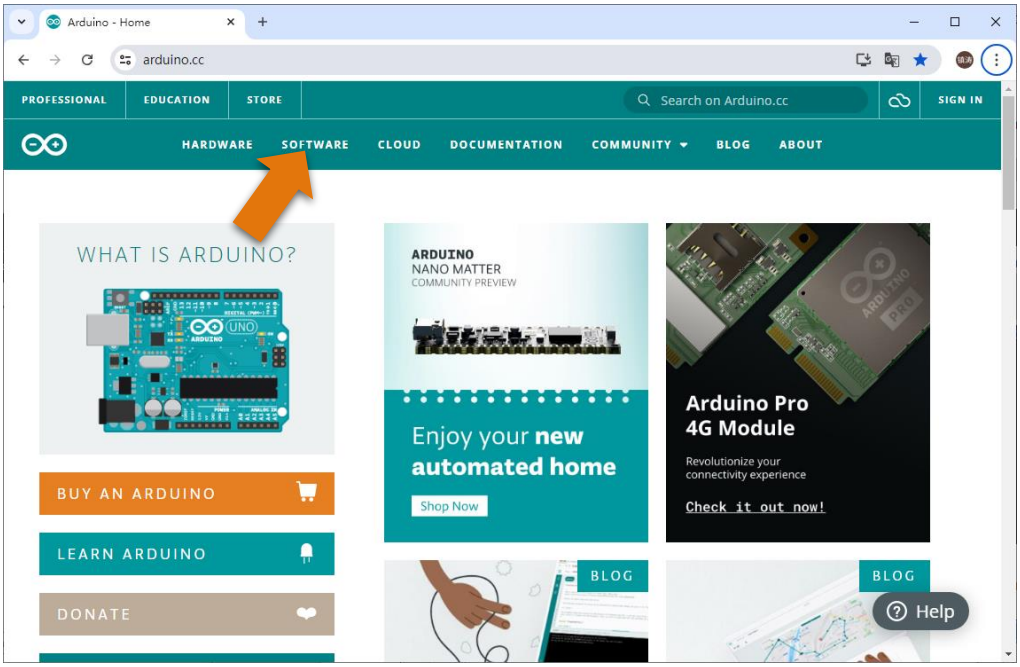
If you still haven't installed the CH340 by following the steps above, you can view readme.pdf to install it.



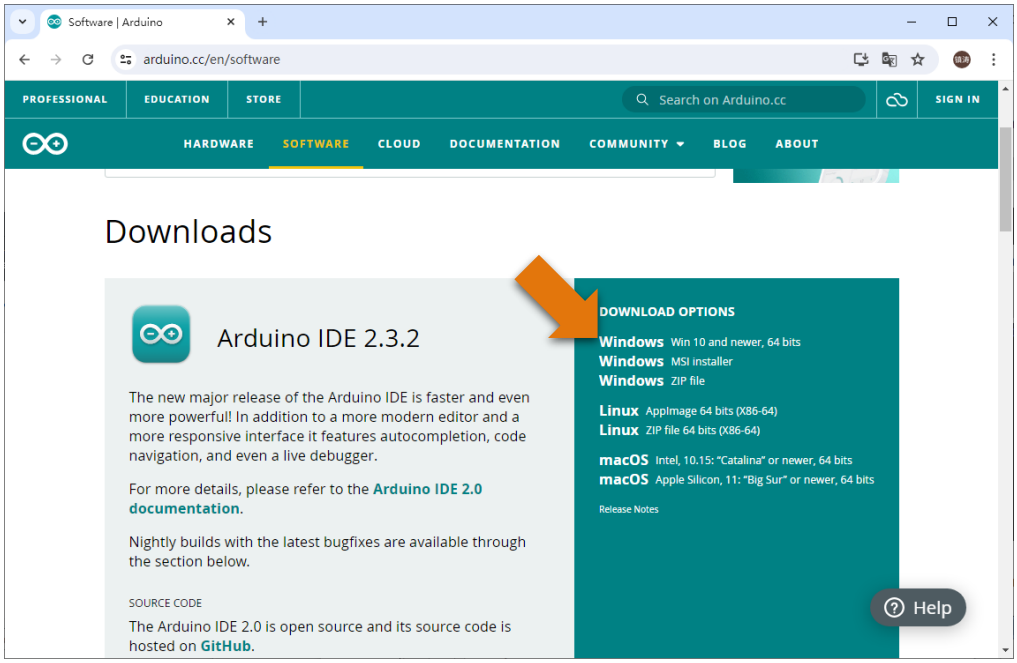
Programming Software

Arduino Software (IDE) is used to write and upload the code for Arduino Board.

First, install Arduino Software (IDE): visit https://www.arduino.cc, click "Download" to enter the download page.



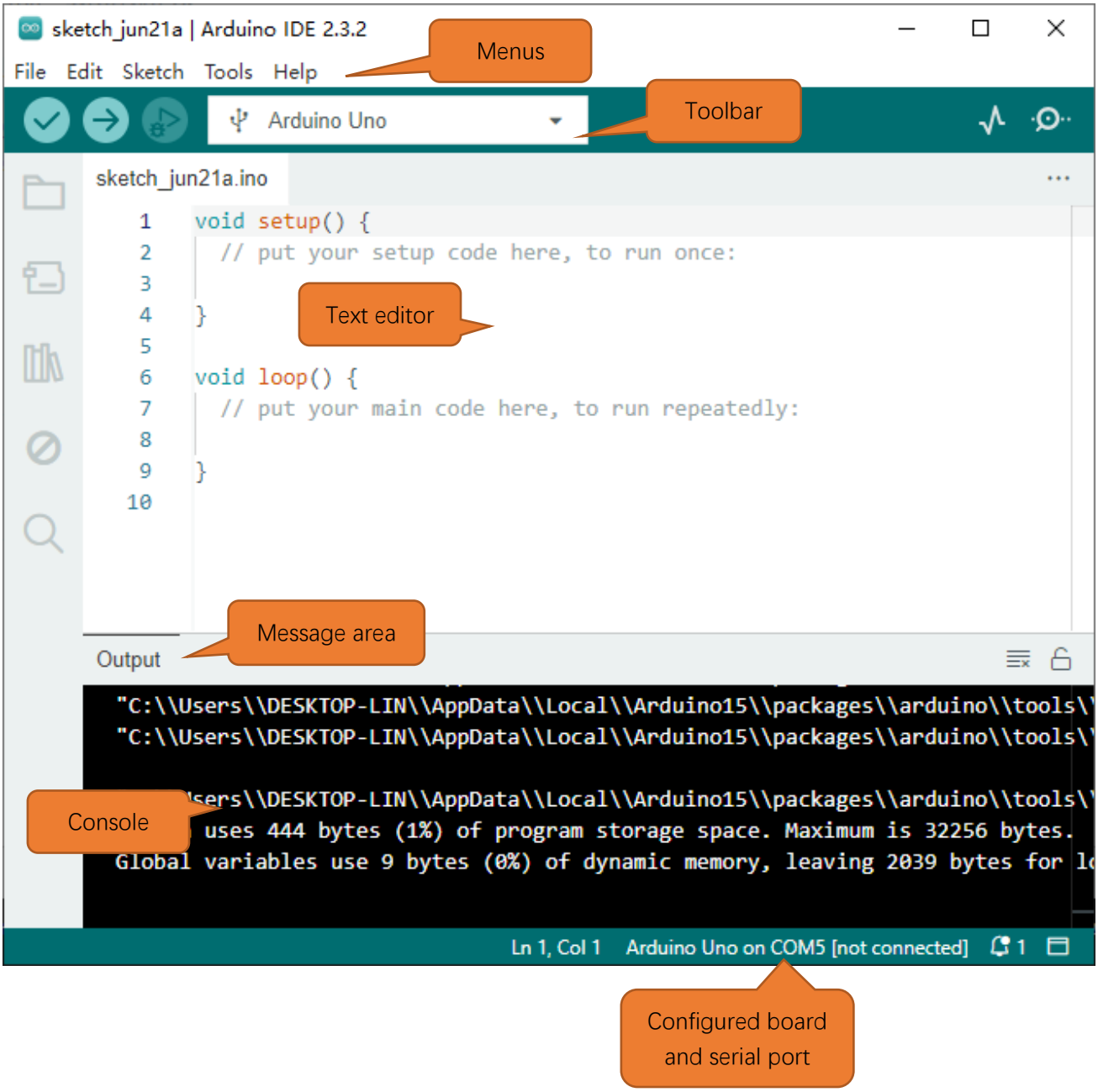
Select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer" to download to install the driver correctly.



After the download completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it popes up, please allow the installation.

After installation is complete, an Arduino Software shortcut will be generated in the desktop. Run the Arduino Software.

The interface of Arduino Software is as follows:



Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and saved with the file extension**.ino**. The editor has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (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, create, open, and save sketches, and open the serial monitor.

Verify
Check your code for compile errors .

Upload
Compile your code and upload them to the configured board.

Debug
With a debugger or USB port, debug the code.

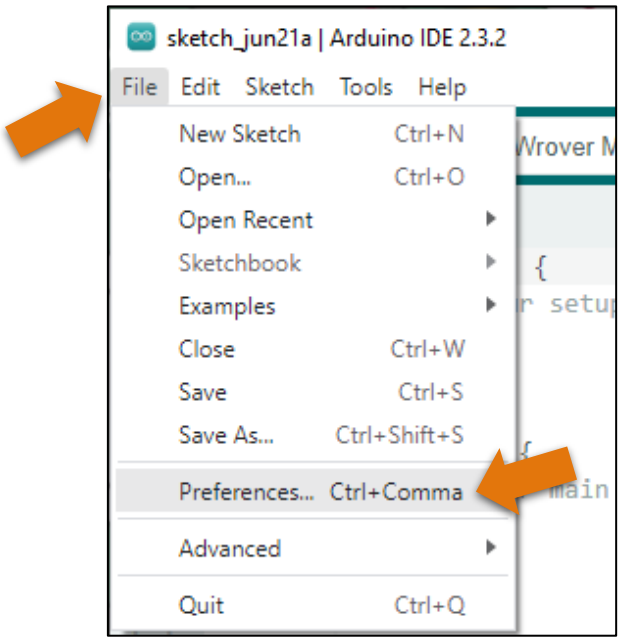
Serial Plotter
Open the serial plotter.

Serial Monitor
Open the serial monitor.

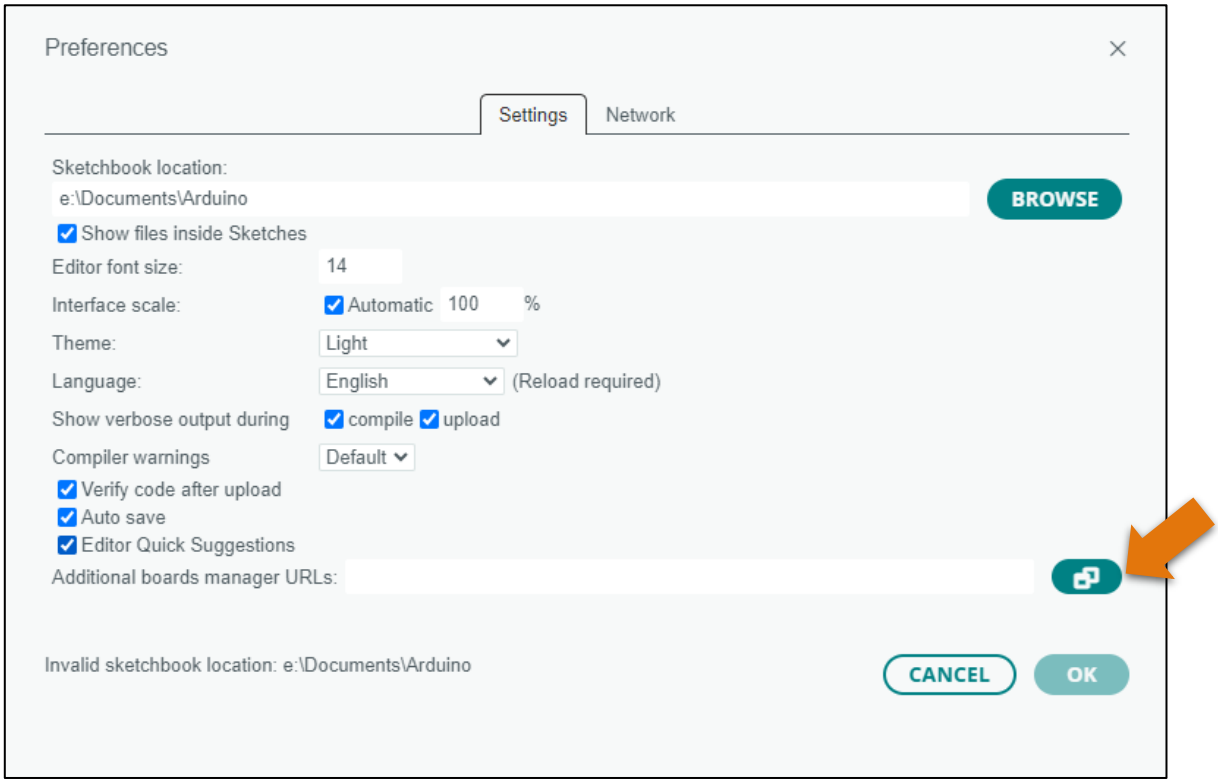
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

Environment Configuration

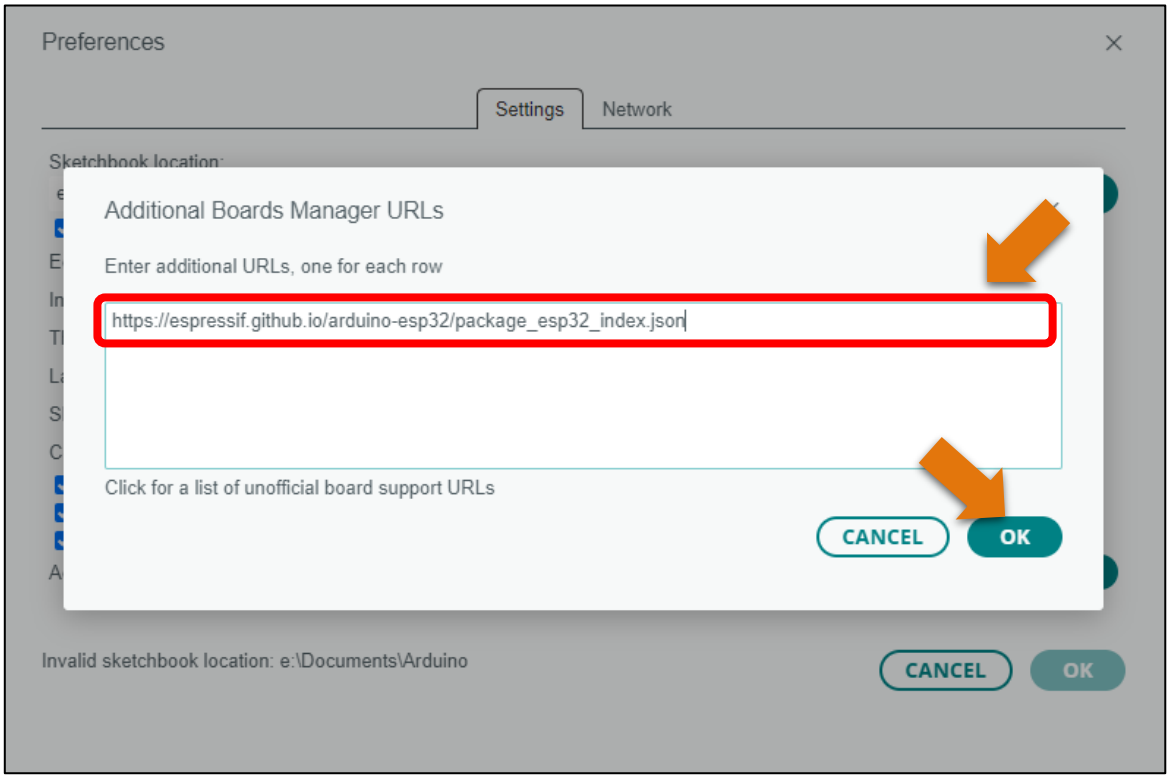
First, open the software platform arduino, and then click File in Menus and select Preferences.



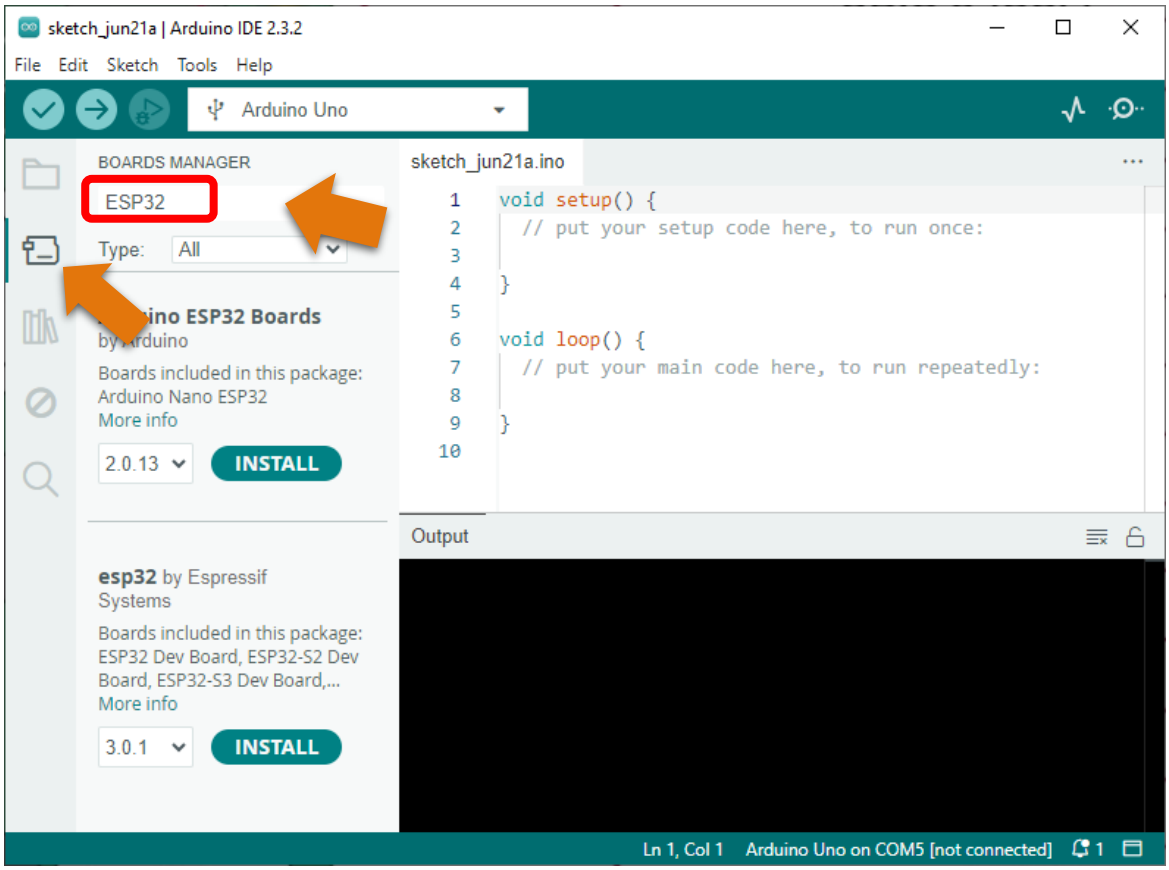
Second, click on the symbol behind "Additional Boards Manager URLs"



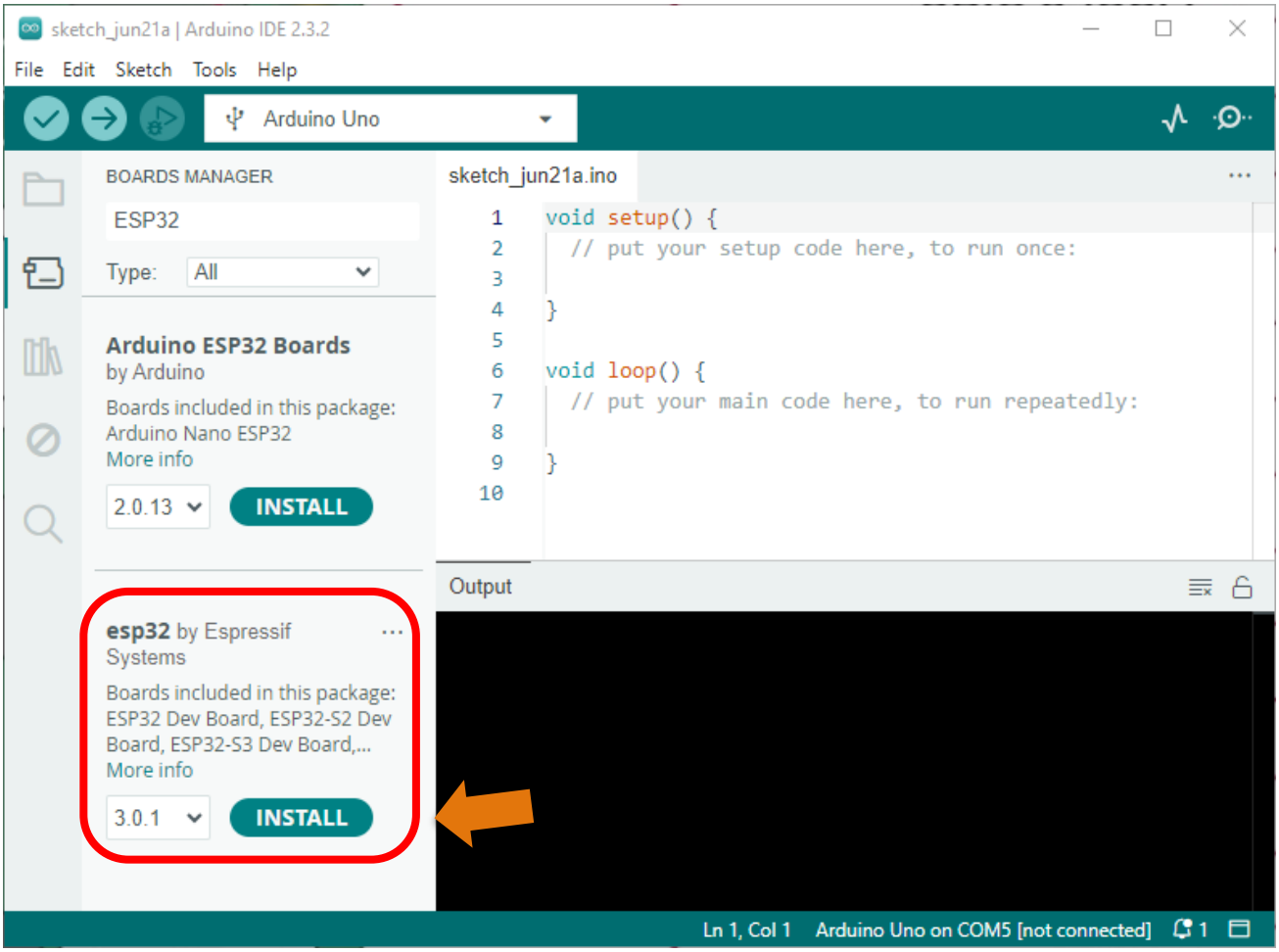
Third, fill in https://espressif.github.io/arduino-esp32/package_esp32_index.json in the new window, click OK, and click OK on the Preferences window again.



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

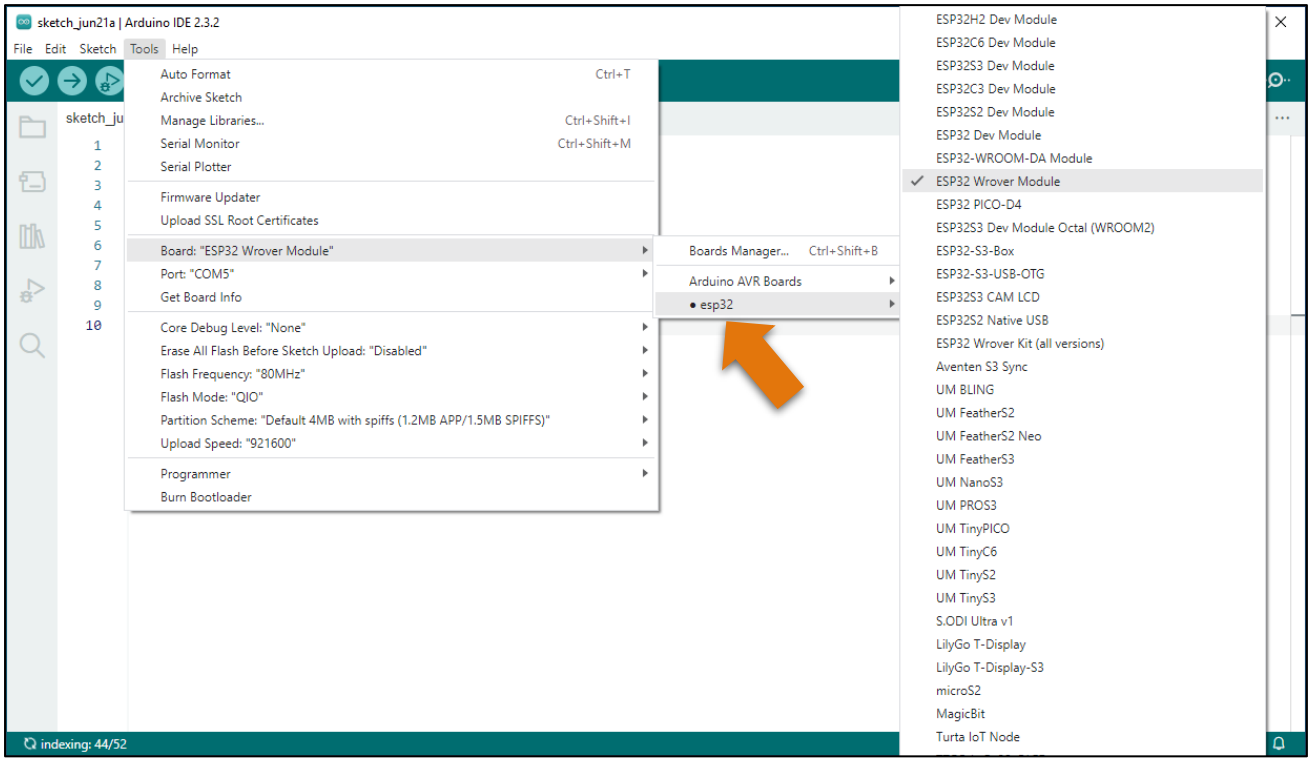


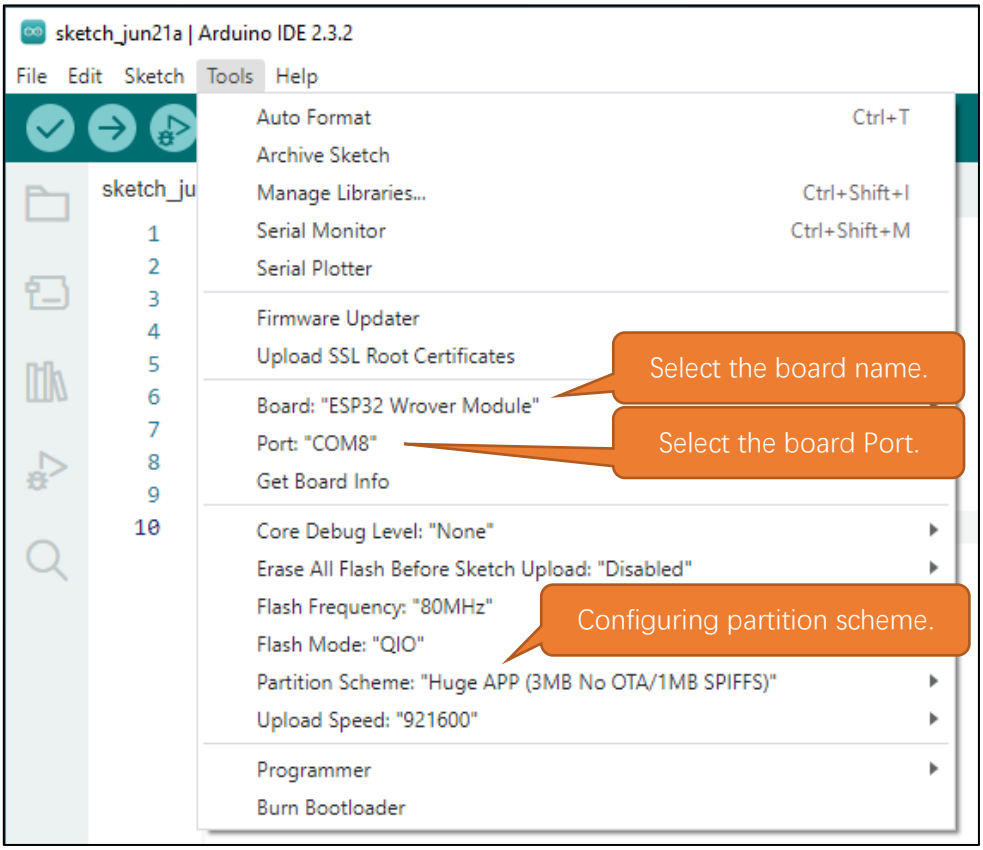
Fifth, select Espressif Systems' ESP32 and select version 3.0.x. Click "INSTALL" to install esp32.



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.

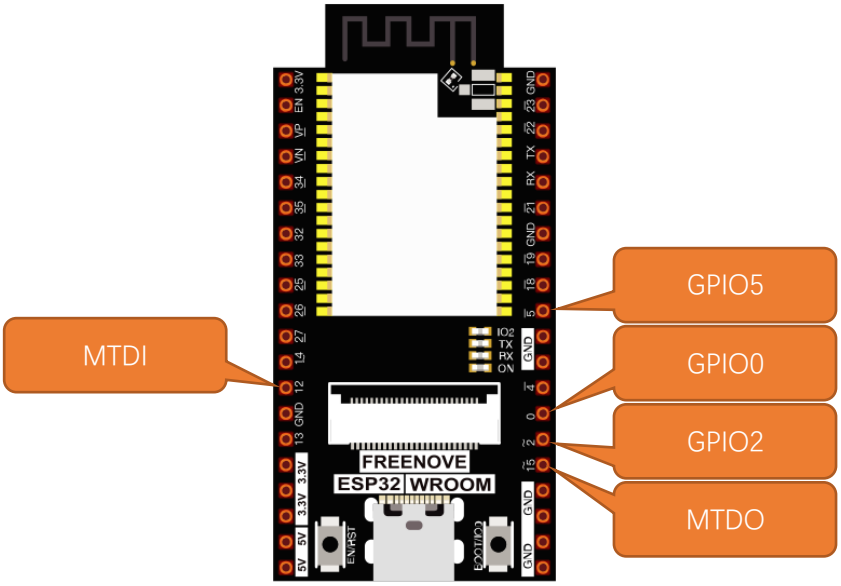
Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3 V		1.8 V	
MTDI	Pull-down	0		1	
Booting Mode					
Pin	Default	SPI Boot		Download Boot	
GPIO0	Pull-up	1		0	
GPIO2	Pull-down	Don't-care		0	
Enabling/Disabling Debugging Log Print over U0TXD During Booting					
Pin	Default	U0TXD Active		U0TXD Silent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

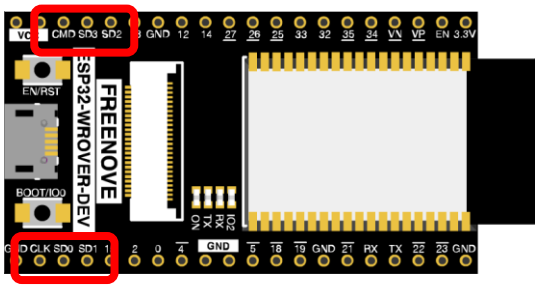
Strapping Pin



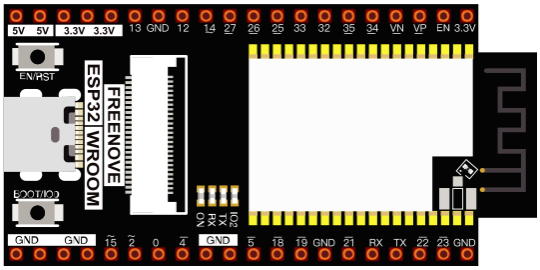
Flash Pin

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".

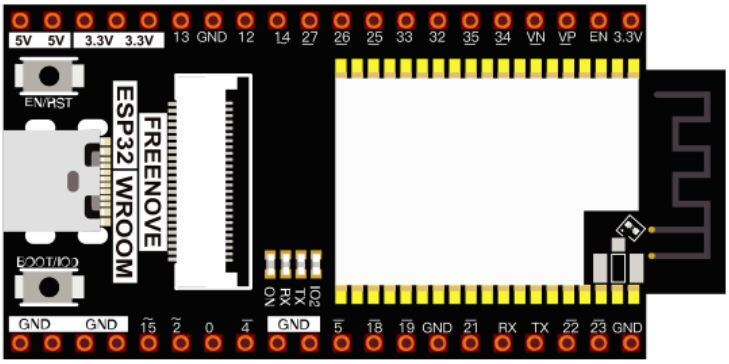
For more relevant information, please check:

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.

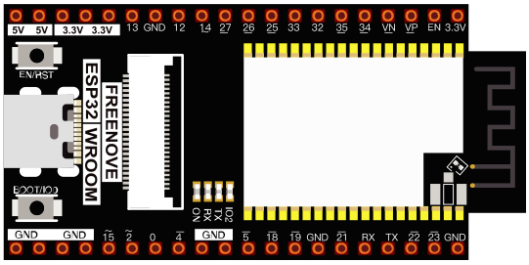
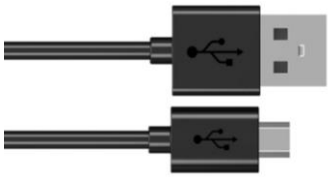
Chapter 0 LED

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

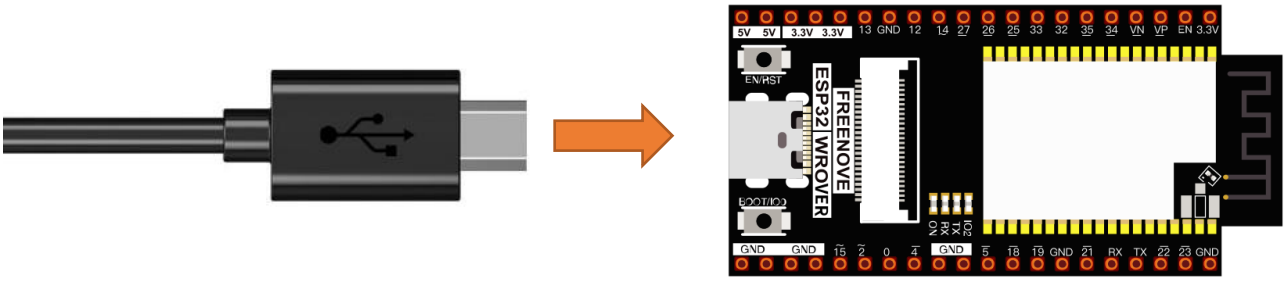
In this project, we will use ESP32 to control blinking a common LED.

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.

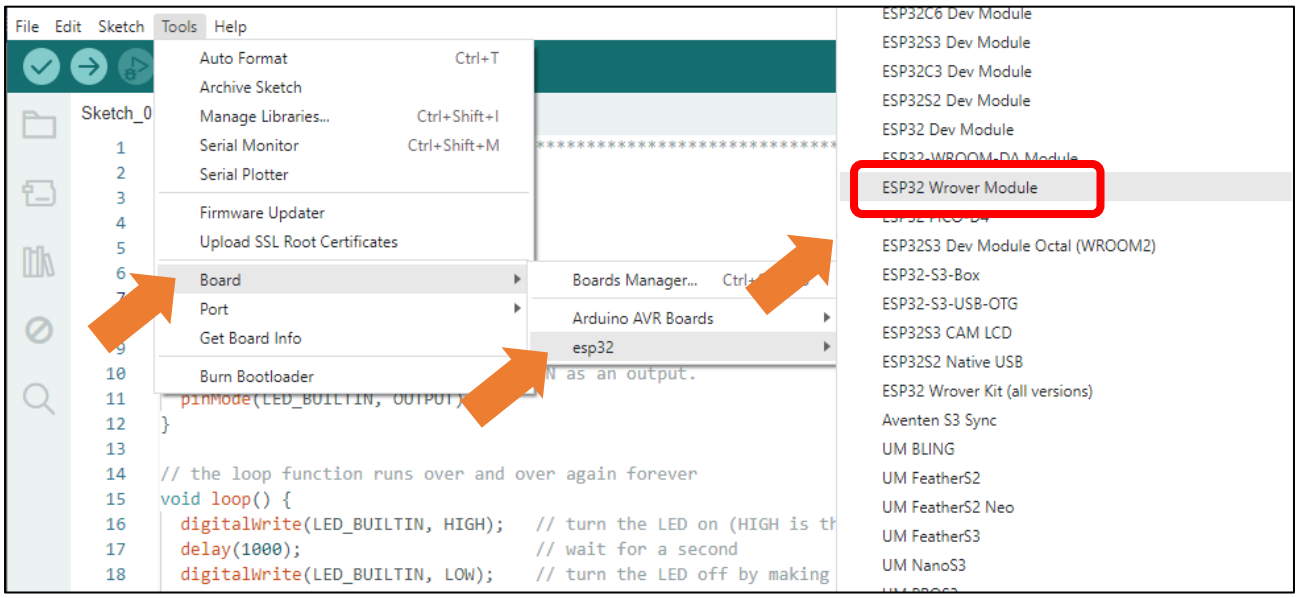
Sketch

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.

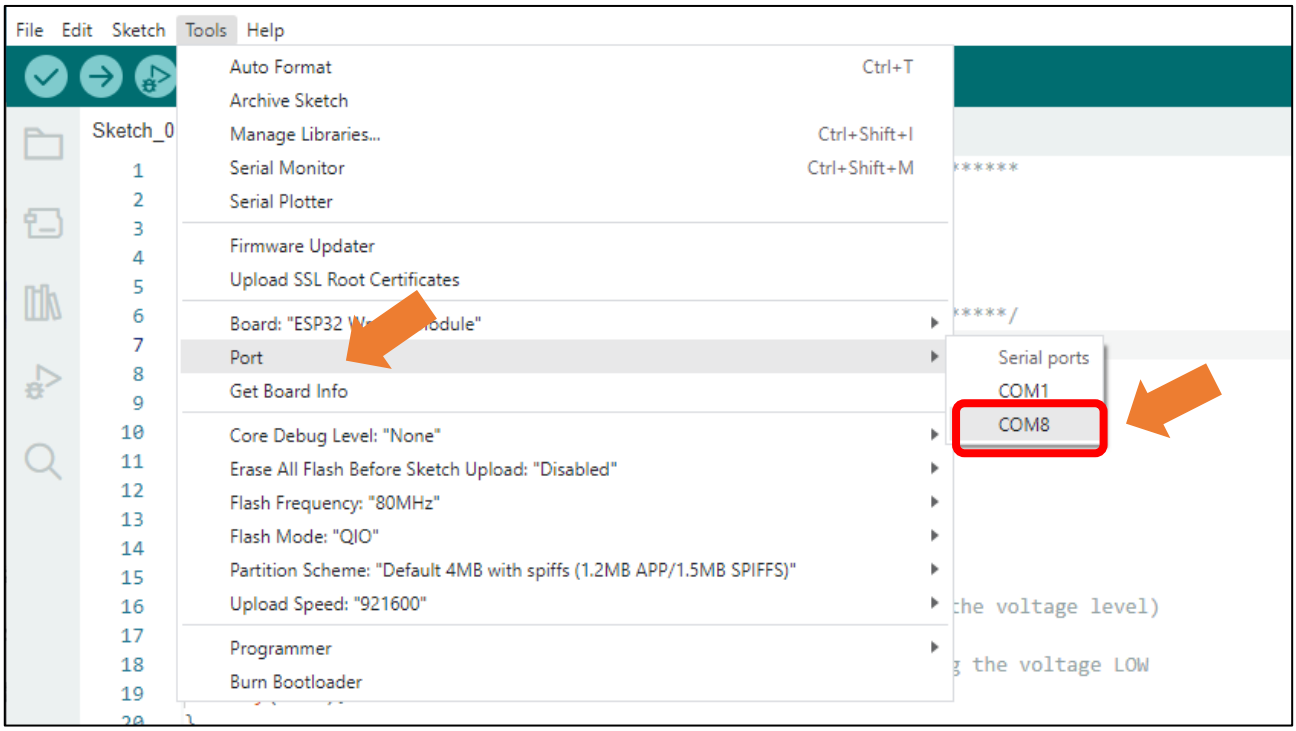
Upload the following Sketch:

Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_01.1_Blink.

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

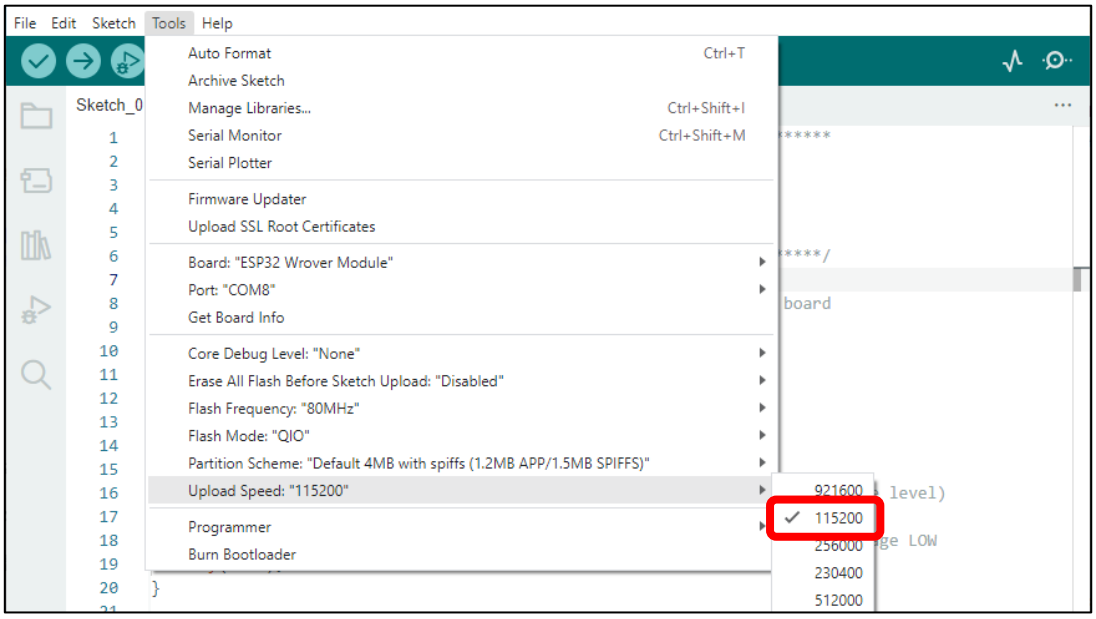


Select the serial port.

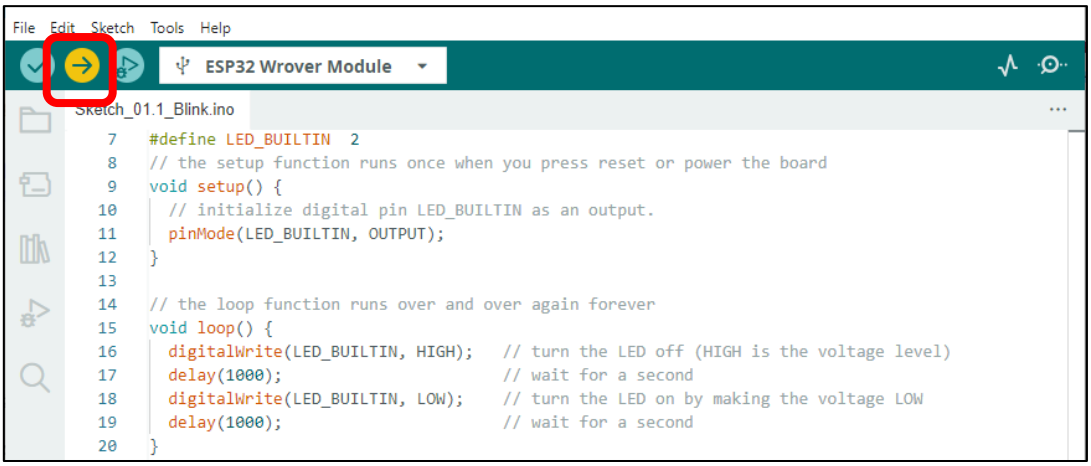


Note: For macOS users, if the uploading fails, please set the baud rate to 115200 before clicking

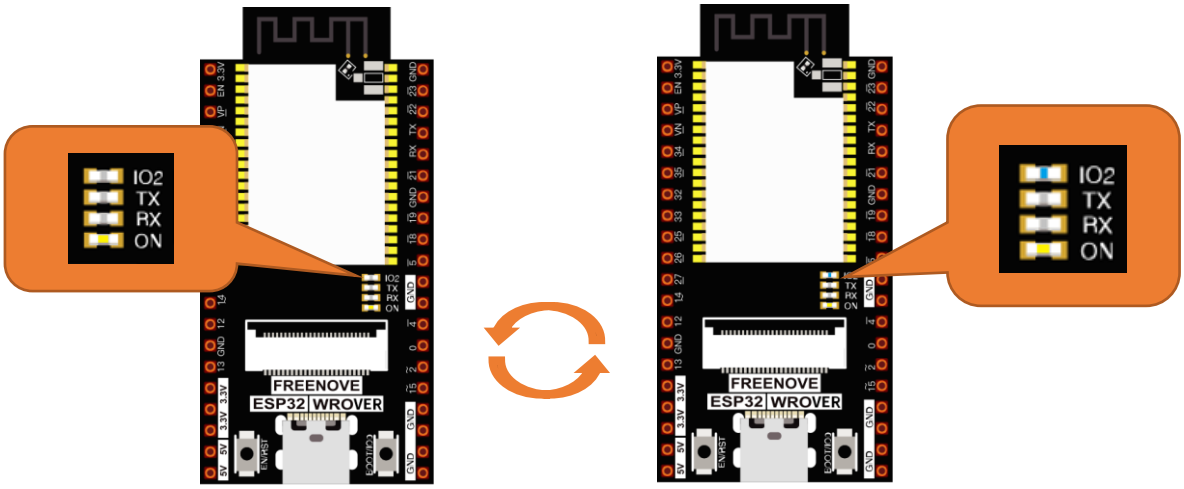
"Upload Speed".



Sketch_01.1_Blink



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



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.

```
1   // the setup function runs once when you press reset or power the board
2   void setup() {
…     …
5   }
6
7   // the loop function runs over and over again forever
8   void loop() {
…     …
13  }
```

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.

```
1   #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.

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

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

```
10  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.

```
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.

```
digitalWrite(PIN_LED, LOW);    // turn the LED on by making the voltage LOW
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/

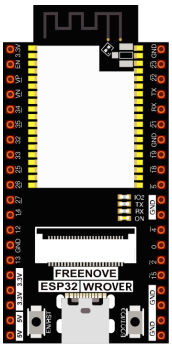
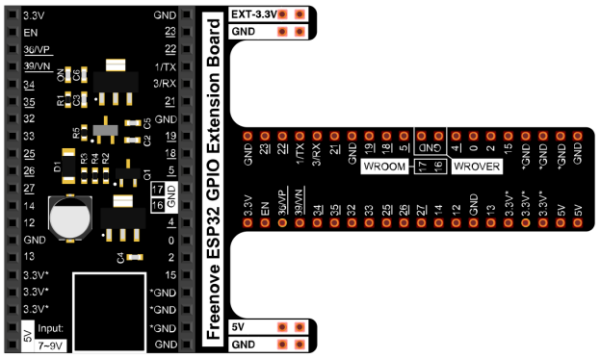
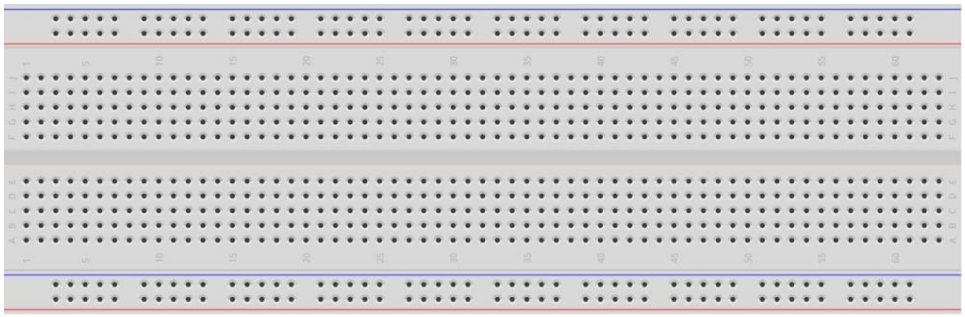
Chapter 1 LED

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

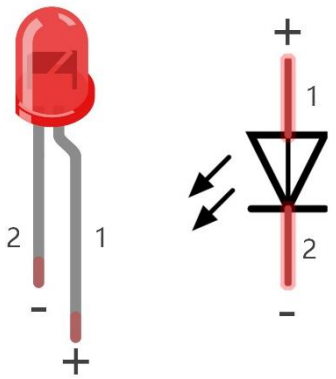
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
LED x1	Resistor 220Ω x1	Jumper M/M x2

Component knowledge

LED

A LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two poles. A LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-). Negative output is also referred to as Ground (GND). This type of component is known as "diodes" (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



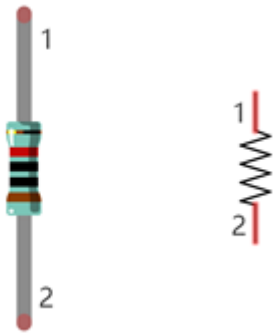
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

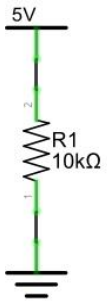
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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

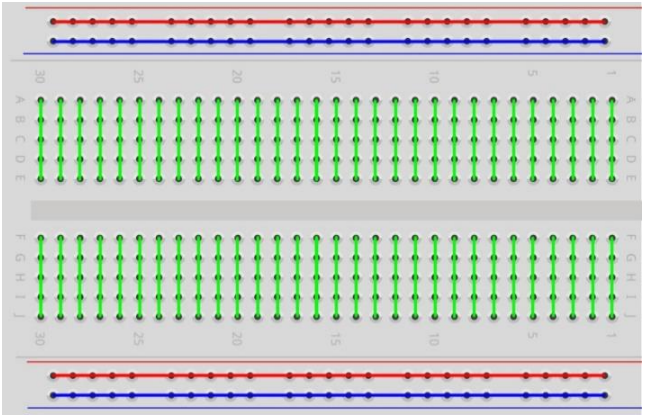


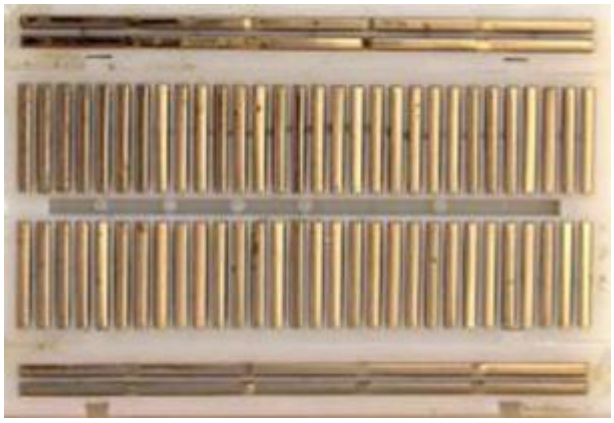
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

Breadboard

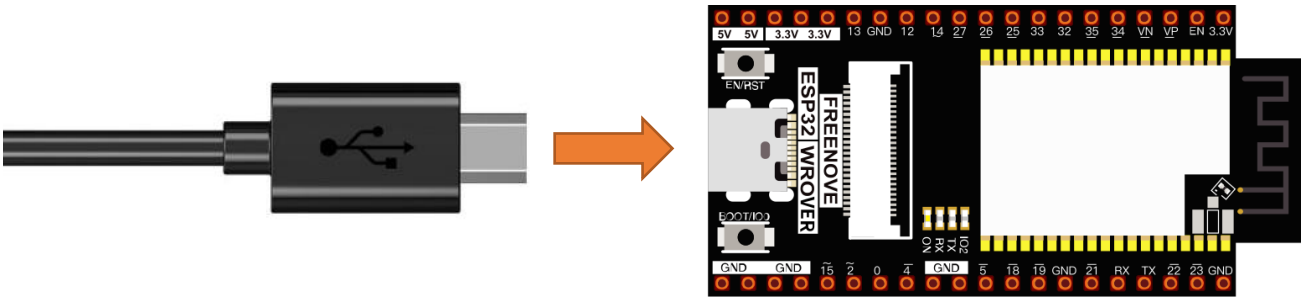
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

ESP32-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.

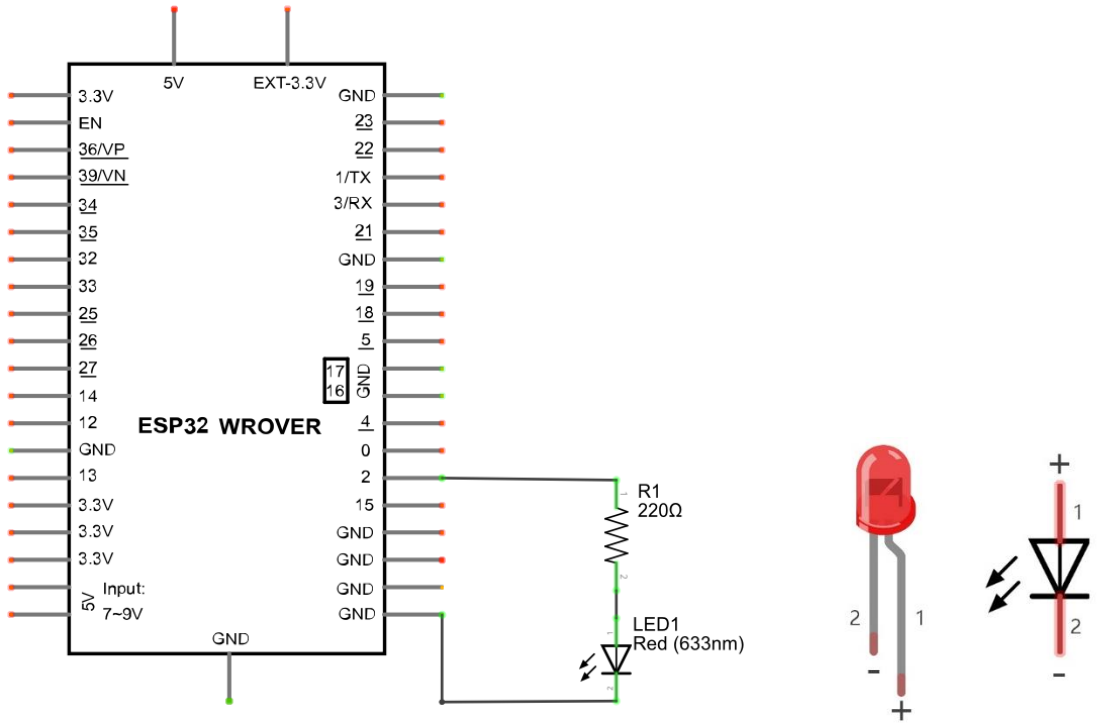
We can also use DC jack of extension board to power ESP32-WROVER.In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

Circuit

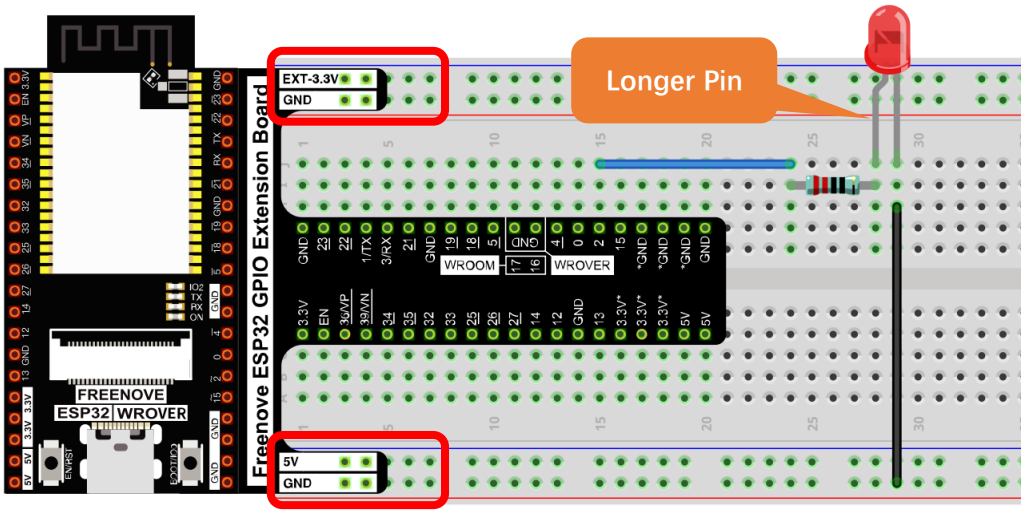
First, disconnect all power from the ESP32-WROVER. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP32-WROVER.

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.

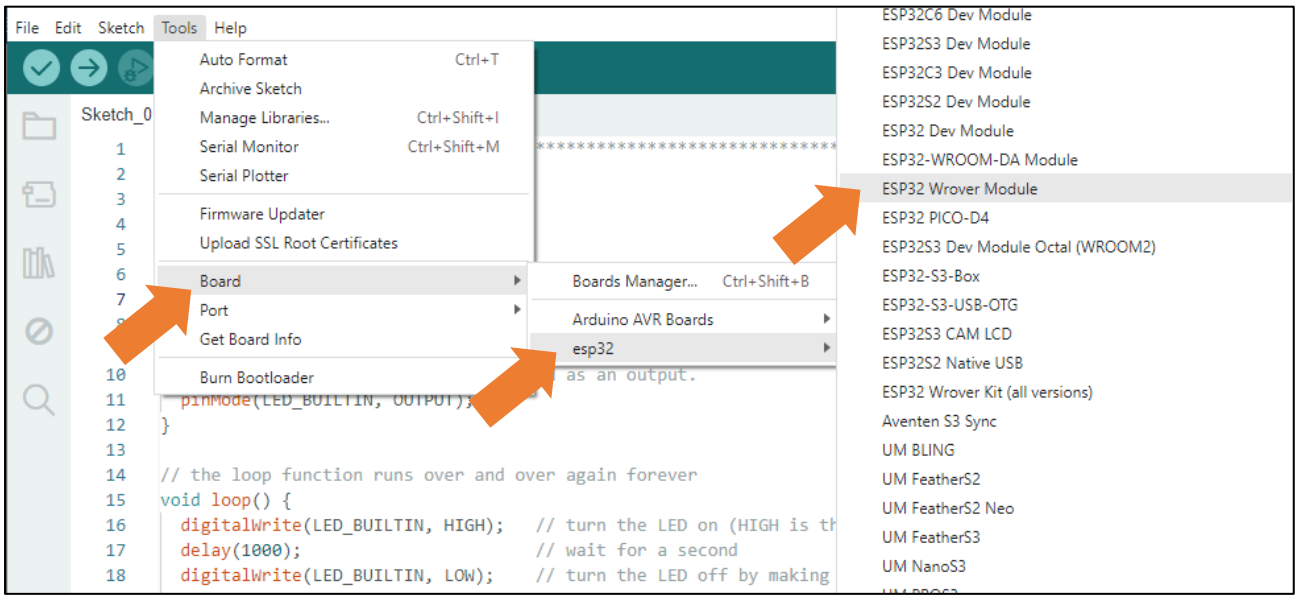
Sketch

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.

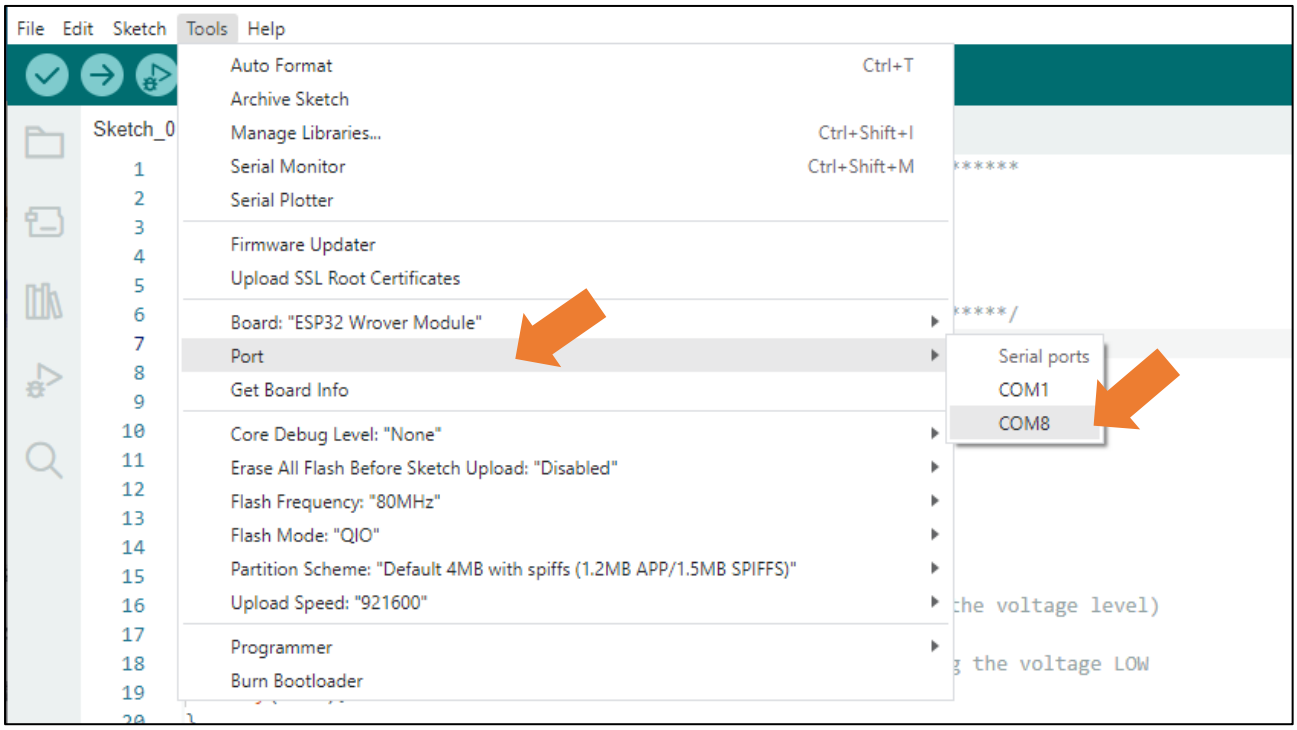
Upload the following Sketch:

Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_01.1_Blink.

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

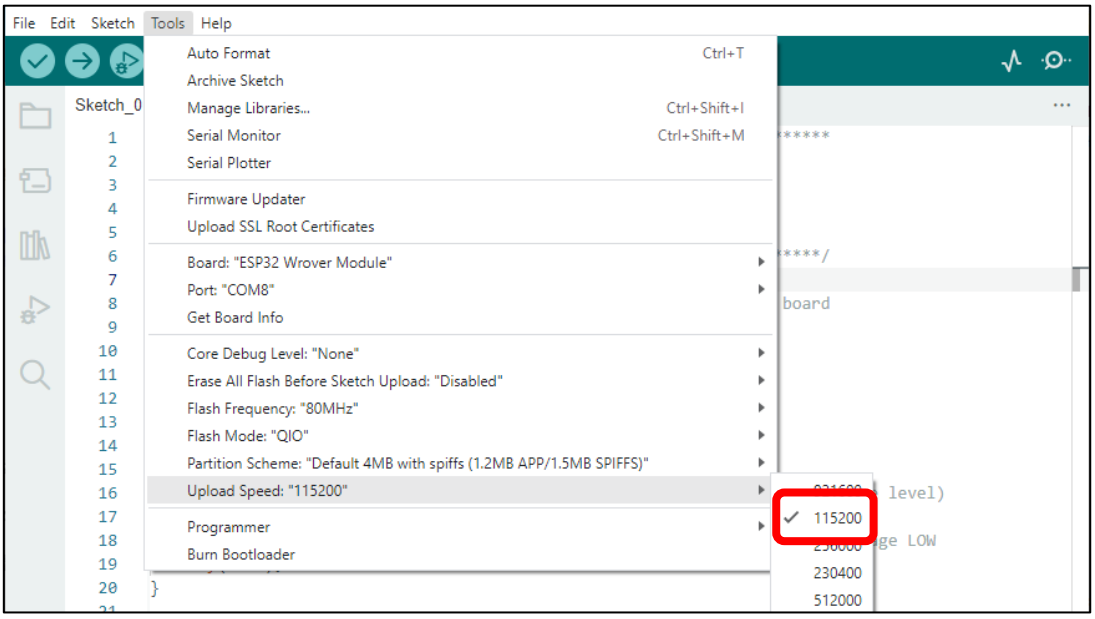


Select the serial port.

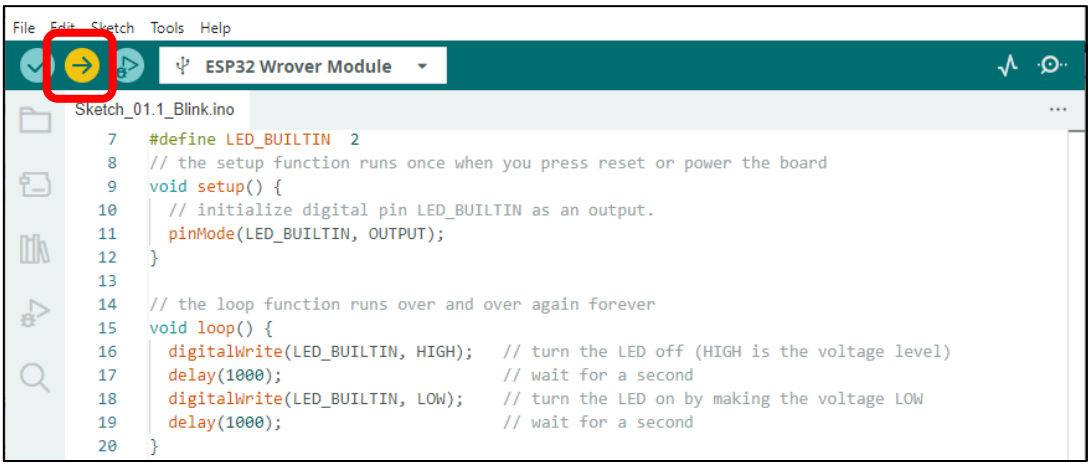


Note: For macOS users, if the uploading fails, please set the baud rate to 115200 before clicking

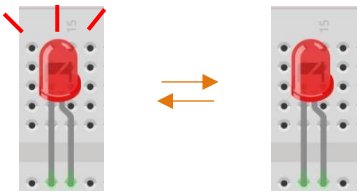
"Upload Speed".



Sketch_01.1_Blink



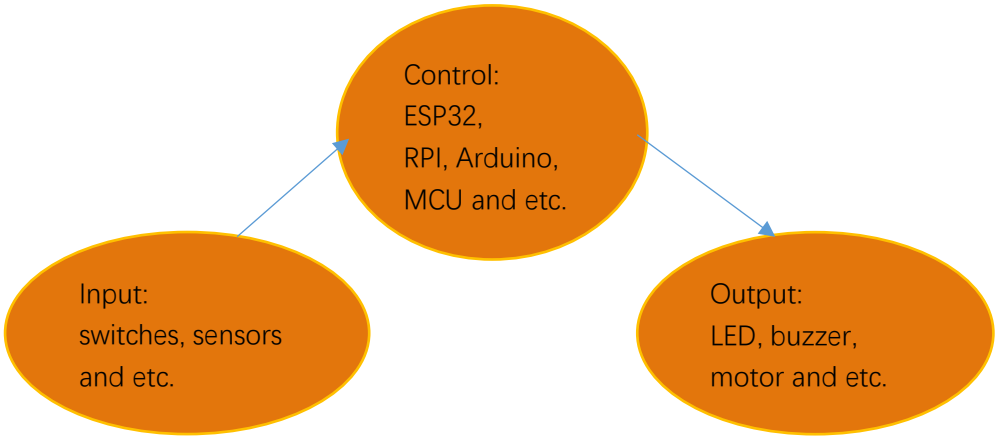
Click "Upload", Download the code to ESP32-WROVER and your LED in the circuit starts Blink.



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.

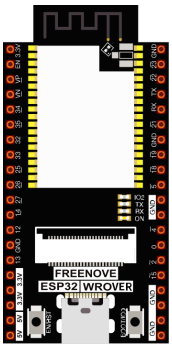
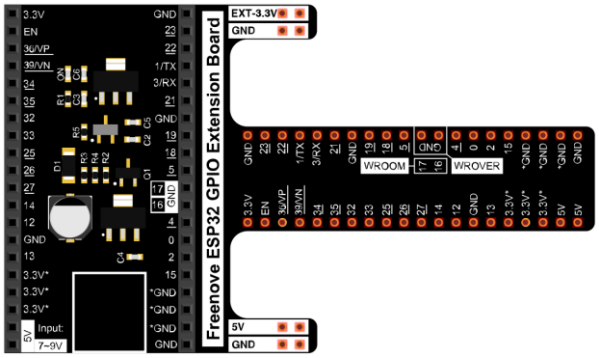
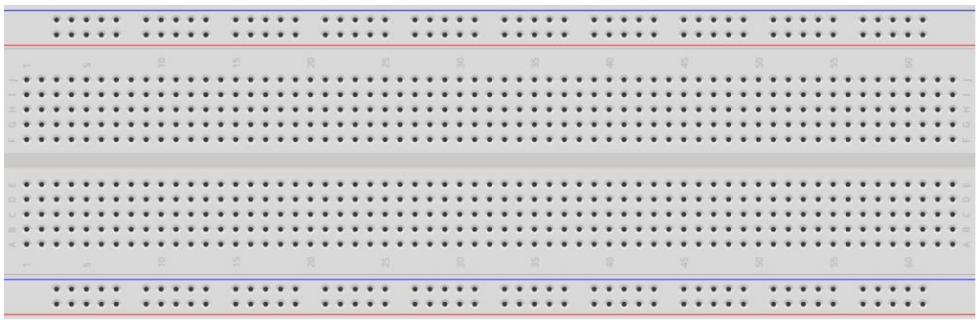


Next, we will build a simple control system to control a LED through a push button switch.

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.

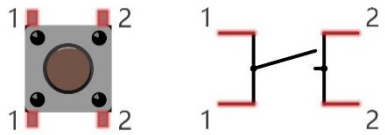
Component List

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

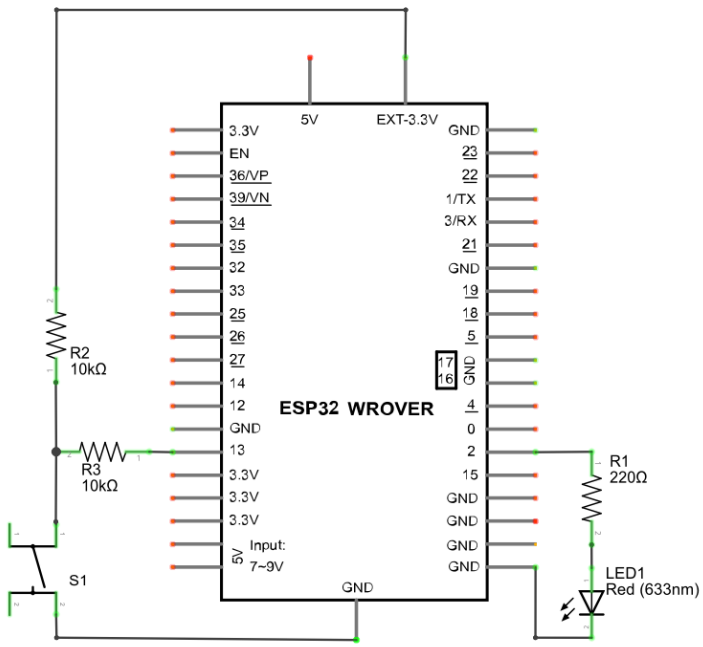
This type of push button switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



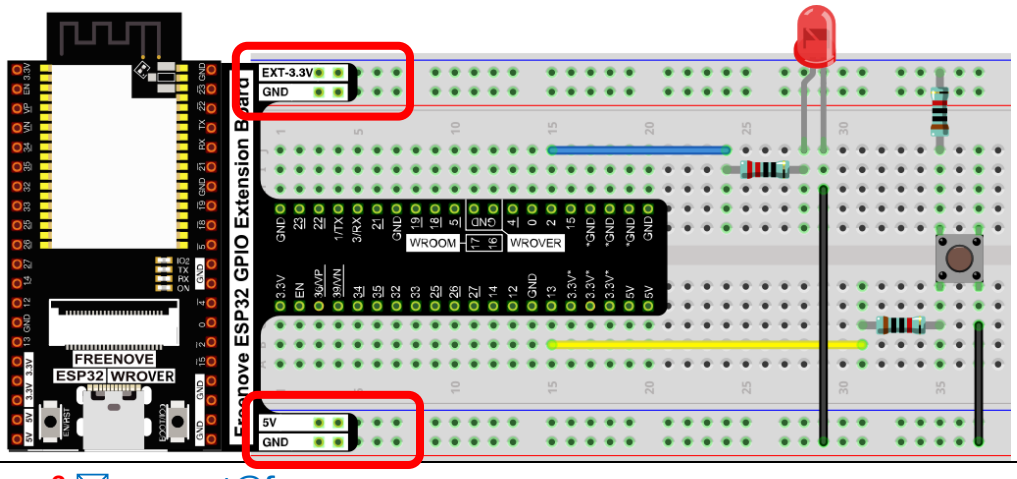
When the button on the switch is pressed, the circuit is completed (your project is powered ON).

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_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_02.1_ButtonAndLed.

Sketch_02.1_ButtonAndLed

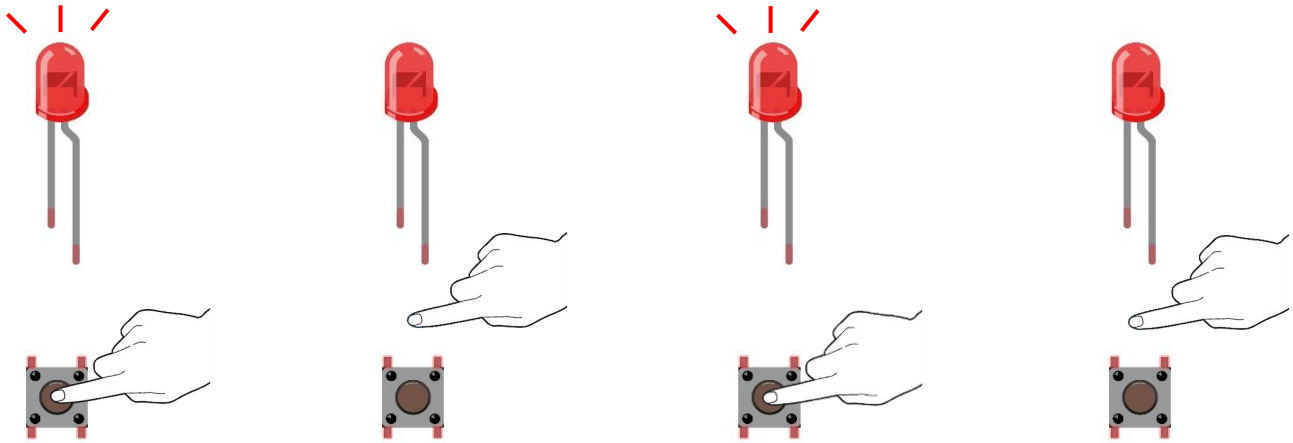
File Edit Sketch Tools Help

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);

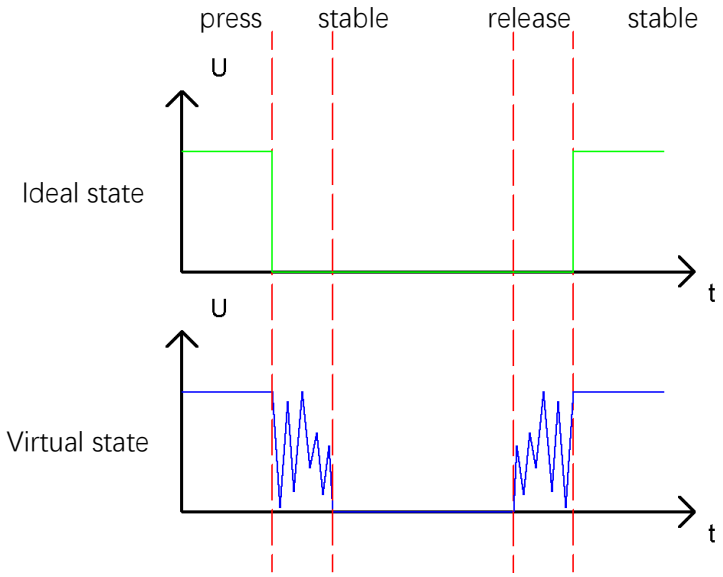
This function returns the value read at the given pin. It will be "HIGH" or "LOW"(1 or 0) depending on the logic level at the pin.

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

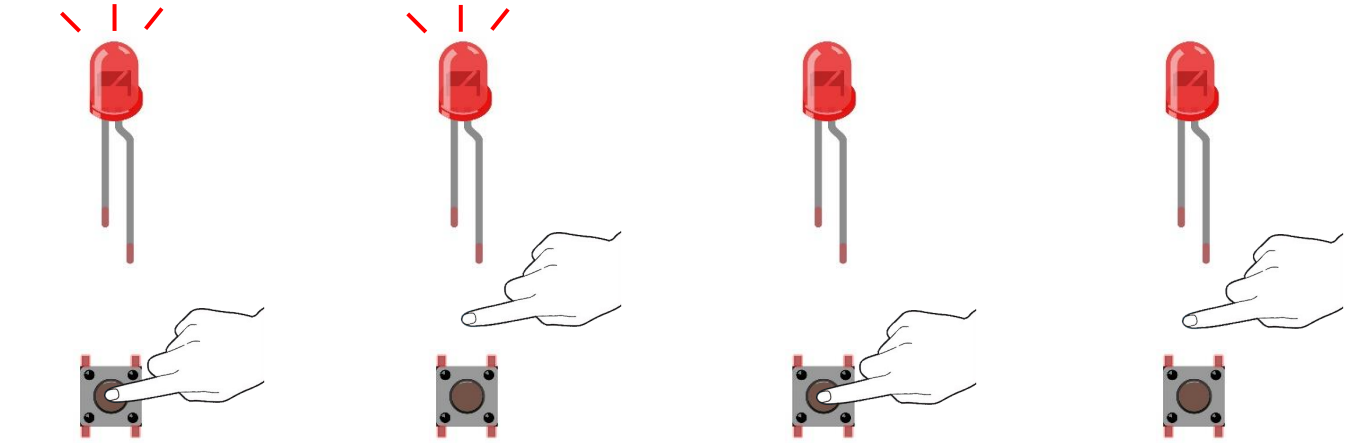
```
#define PIN_LED    2
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

// 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.



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));
}
```

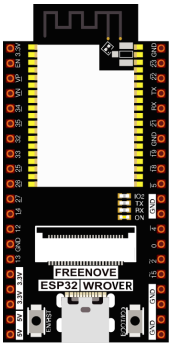
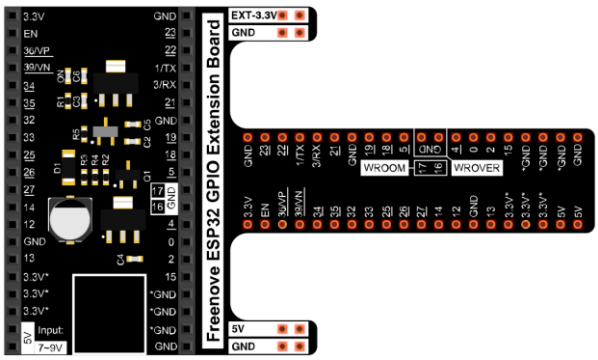
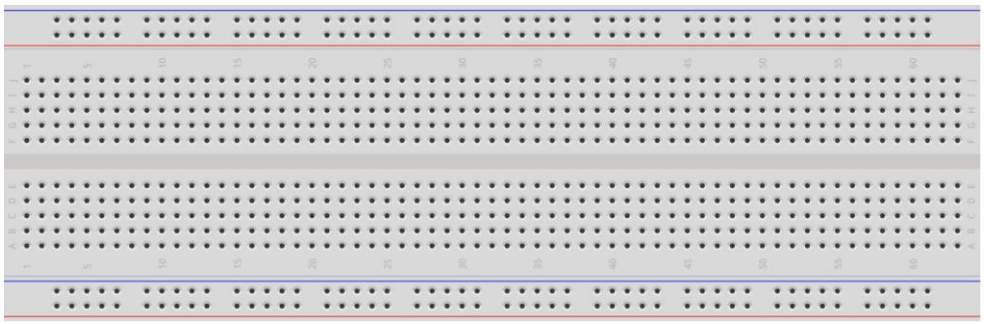
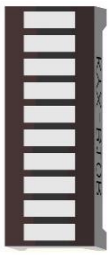
Chapter 3 LED Bar

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

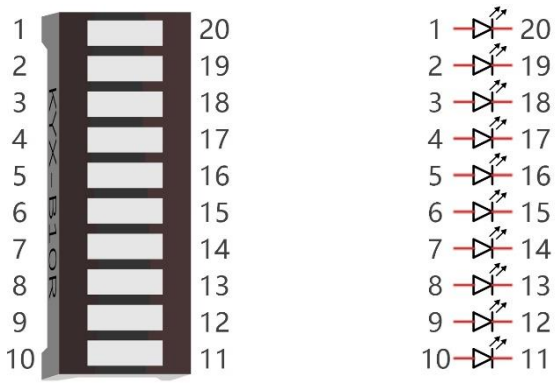
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x10	LED bar graph x1 	Resistor 220Ω x10

Component knowledge

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

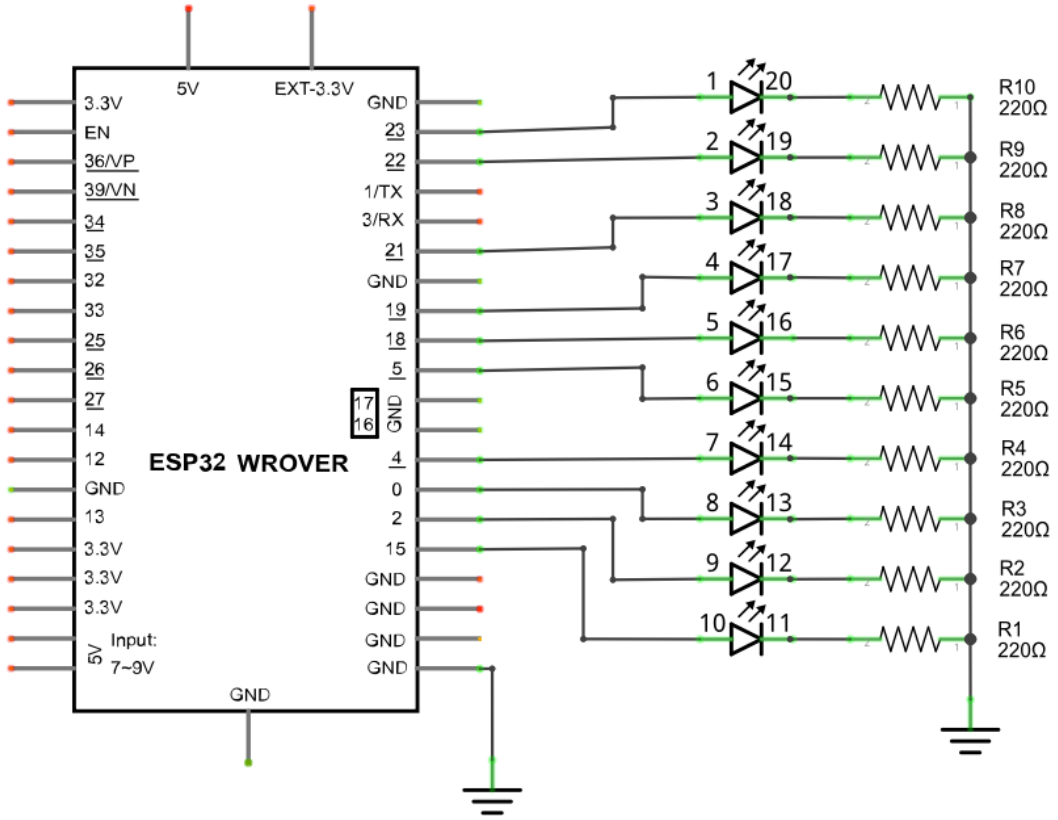
LED bar

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.

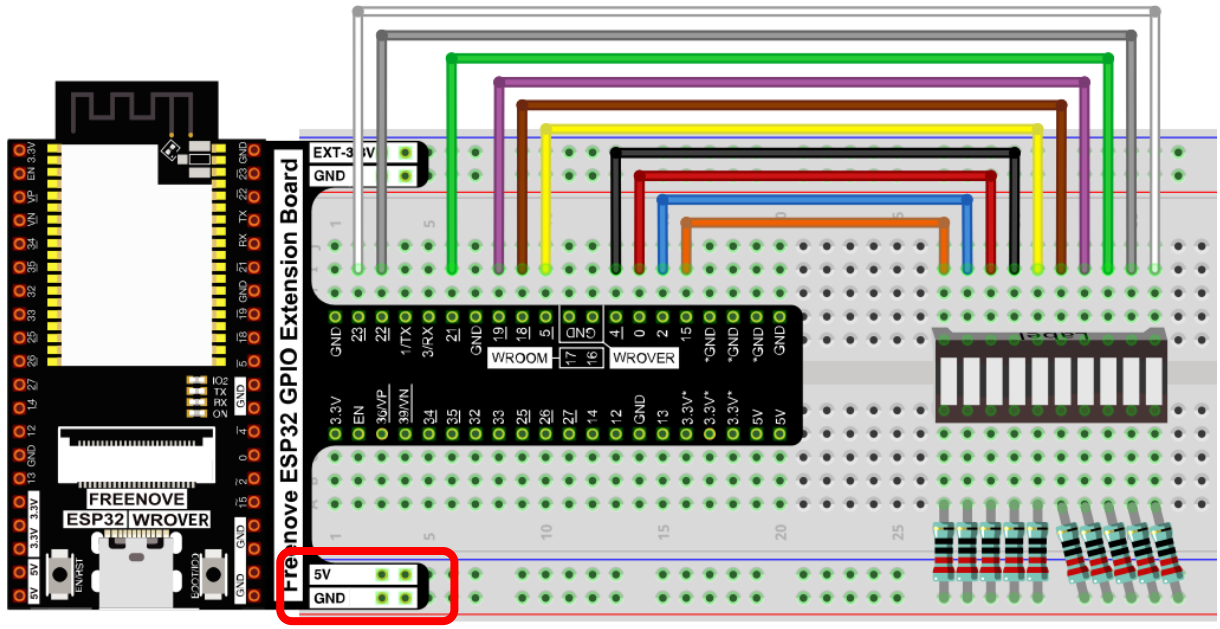


Circuit

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_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_03.1_FlowingLight.

Sketch_03.1_FlowingLight

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_03.1_FlowingLight.ino

```
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);
  }
```

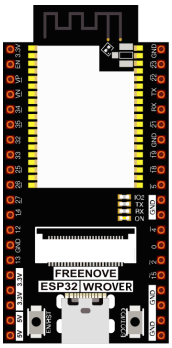
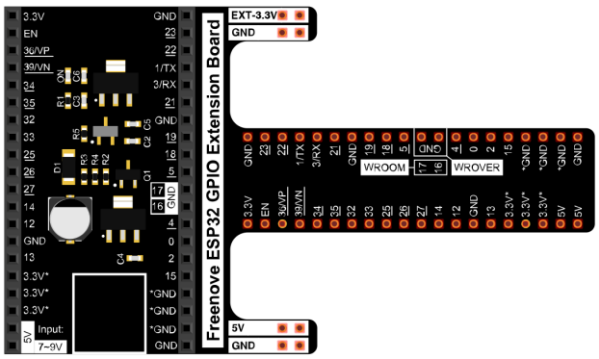
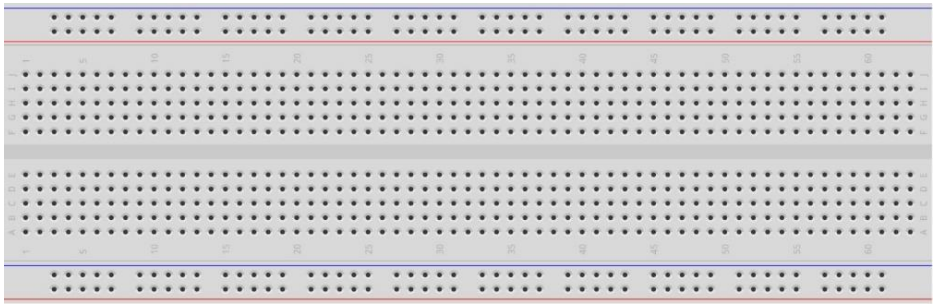
Chapter 4 Analog & PWM

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.

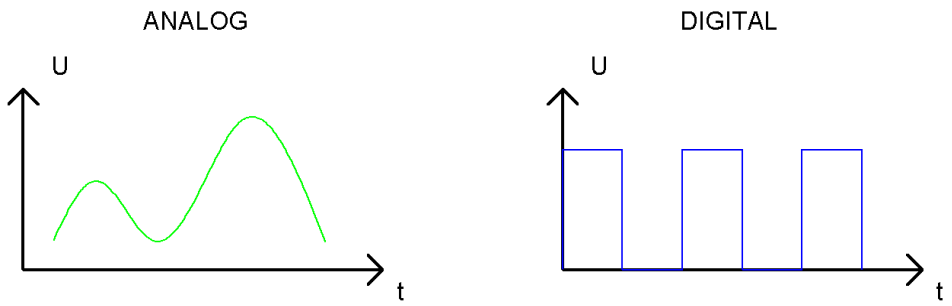
Component List

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

Related knowledge

Analog & Digital

An analog signal is a continuous signal in both time and value. On the contrary, a digital signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an analog signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0℃ 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.



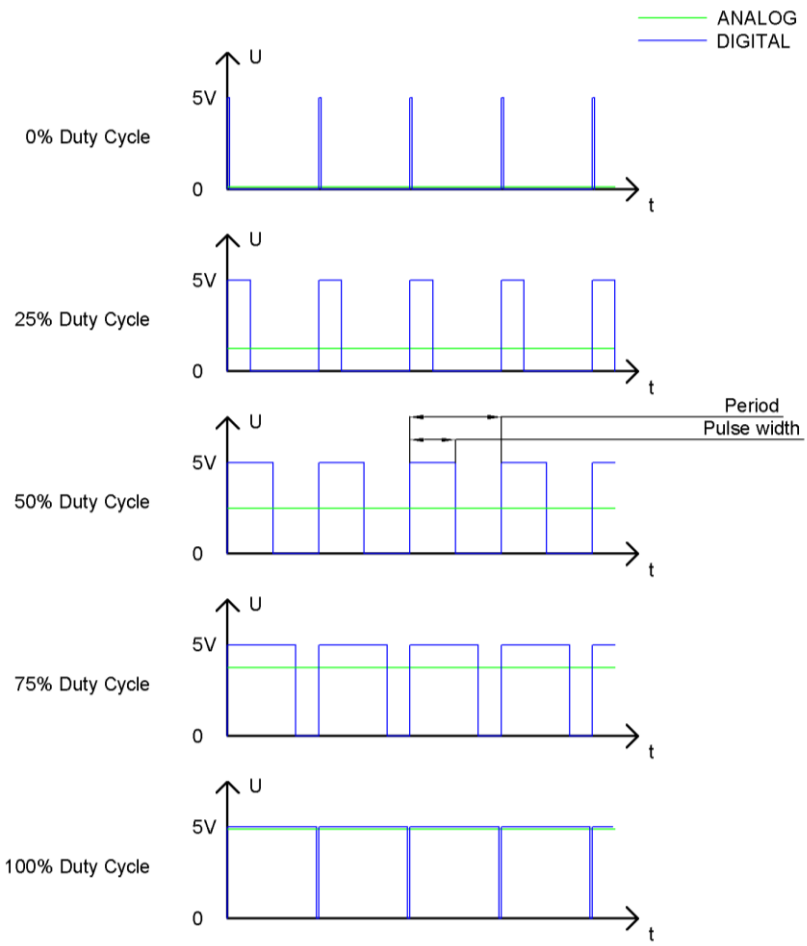
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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.

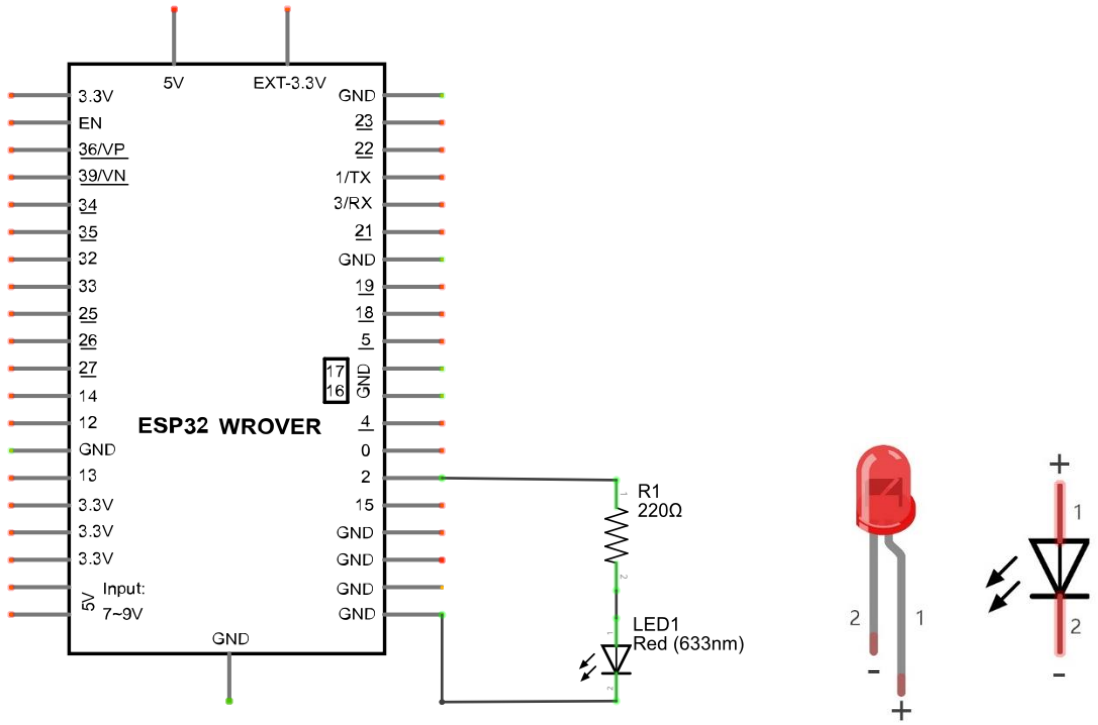
$$Freq_{max} = \frac{80{,}000{,}000}{1 \ll bit}$$

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.)

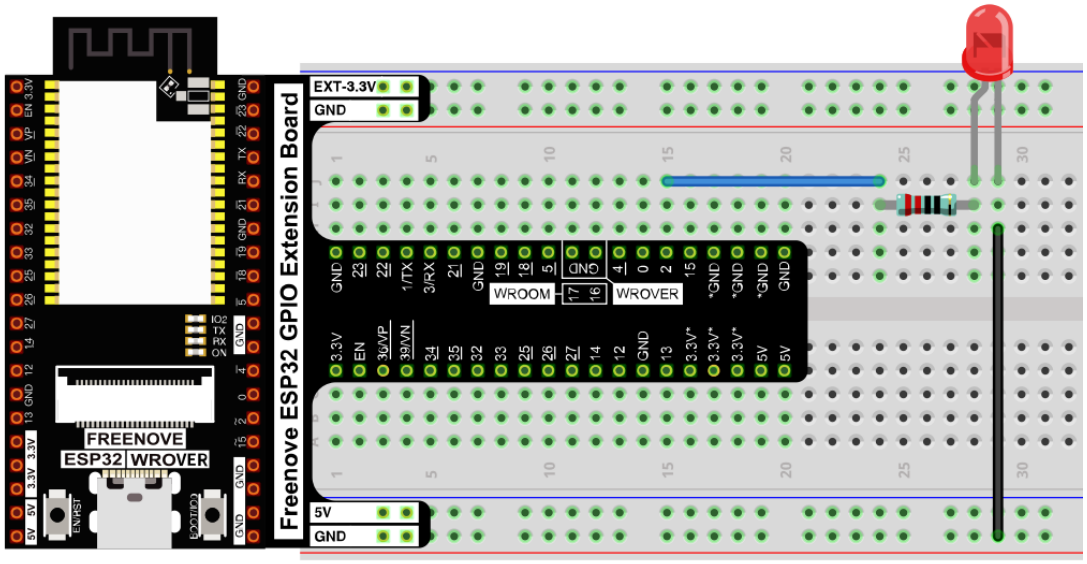
Circuit

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

```
#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() {
  ledcAttachChannel(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);
  }
}
```

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.

```
6   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。

```
11  for (int i = 0; i < 255; i++) {   //make light fade in
12    ledcWrite(PIN_LED, i);
13    delay(10);
14  }
15  for (int i = 255; i > -1; i--) {  //make light fade out
16    ledcWrite(PIN_LED, i);
17    delay(10);
18  }
```

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

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_04.2_FlowingLight2.ino

```
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
int ledCounts;
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
    ledcAttachChannel(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);
  }
}
```

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:

```
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
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
duty	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);
}
```

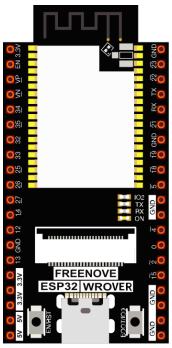
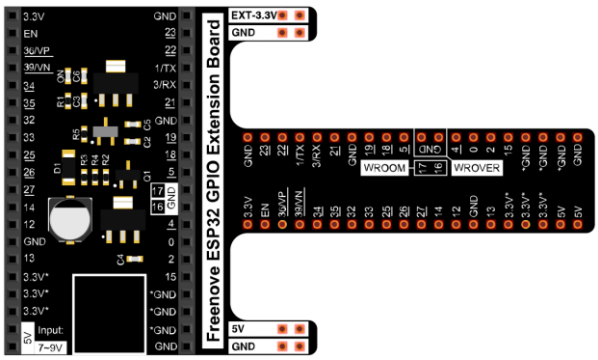
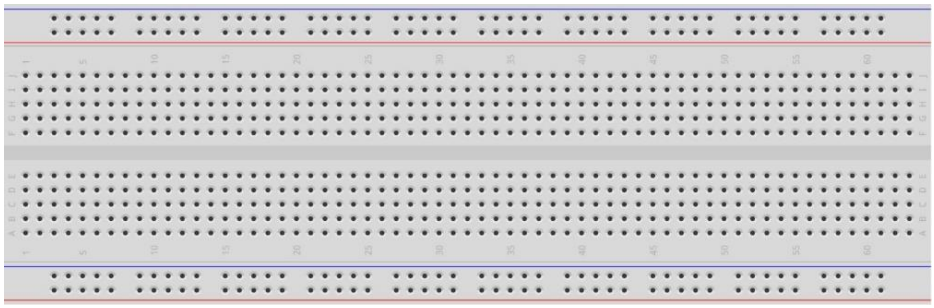
Chapter 5 RGB LED

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

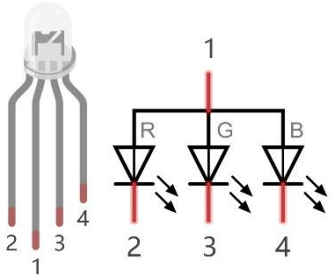
In this project, we will make a multicolored LED. And we can control RGB LED to switch different colors automatically.

Component List

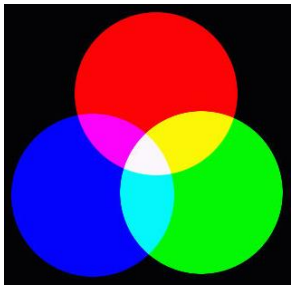
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
RGBLED x1	Resistor 220Ω x3	Jumper M/M x4

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.

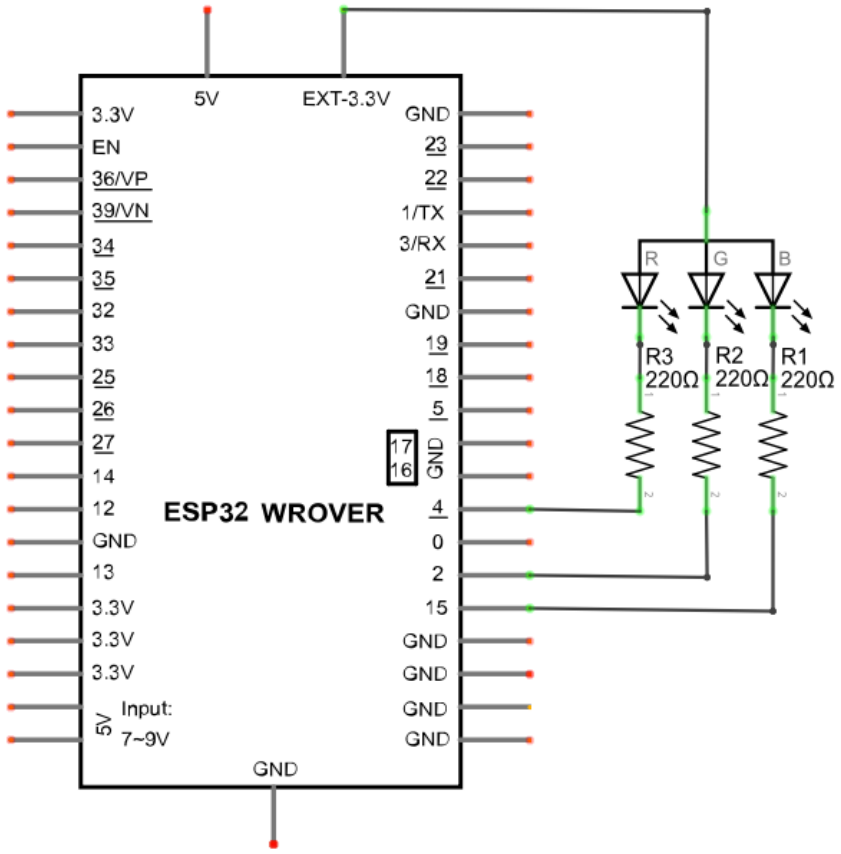


RGB

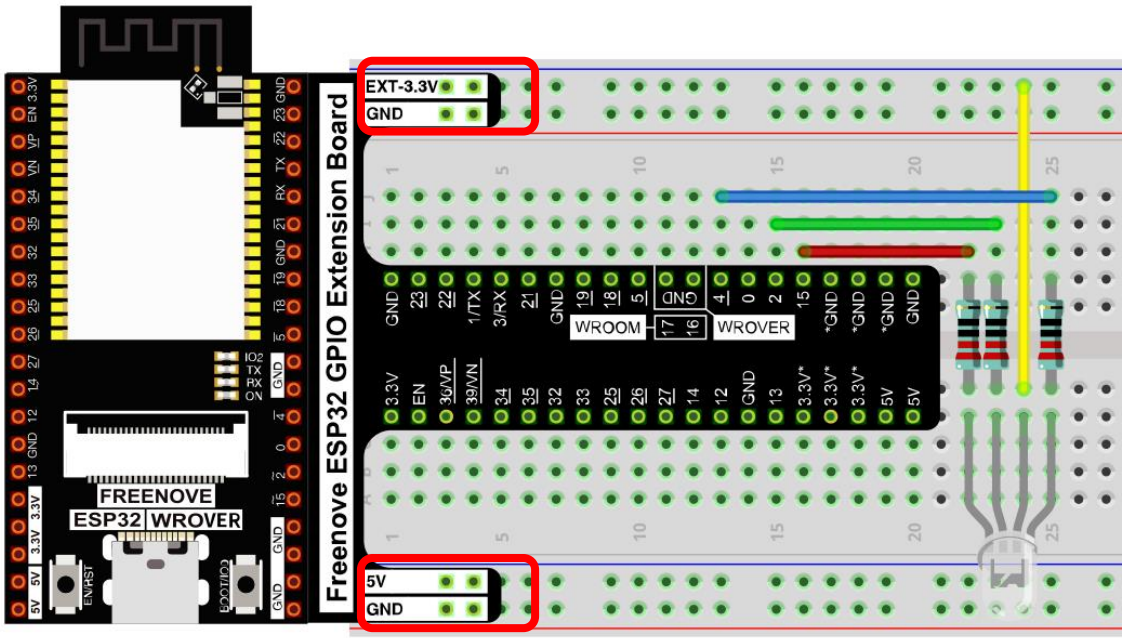
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

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

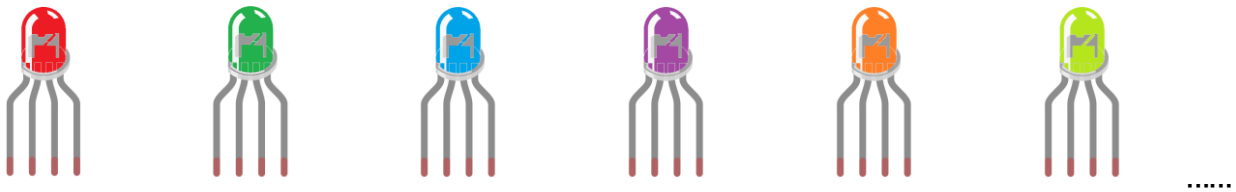
File Edit Sketch Tools Help

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.

```
19 void setColor(byte r, byte g, byte b) {
20   ledcWrite(ledPins[0], 255 - r); //Common anode LED, low level to turn on the led.
21   ledcWrite(ledPins[1], 255 - g);
22   ledcWrite(ledPins[2], 255 - b);
23 }
```

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

```
  red = random(0, 256);
  green = random(0, 256);
  blue = random(0, 256);
  setColor(red, green, blue);
  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) {
```

```
    WheelPos -= 85;
    return (((255 - WheelPos * 3) << 8) | (WheelPos * 3));
  } else {
    WheelPos -= 170;
    return ((WheelPos * 3) << 16 | (255 - WheelPos * 3));
  }
}
```

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

```
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);
}
```

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.

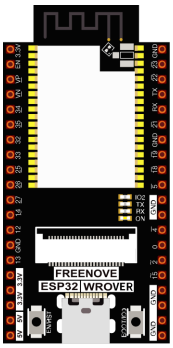
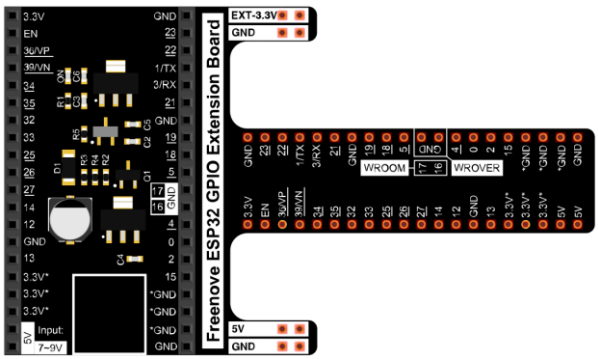
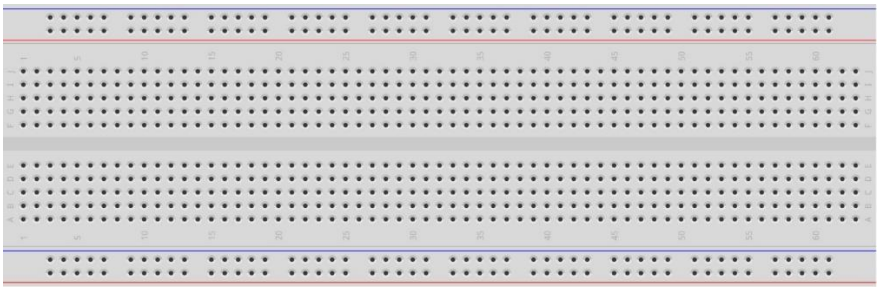
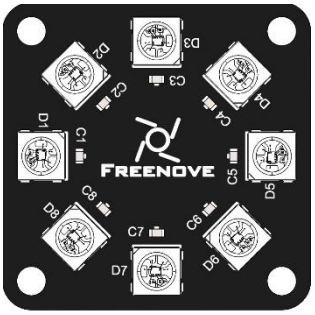
Chapter 6 LEDPixel

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

Project 6.1 LEDPixel

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

Component List

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

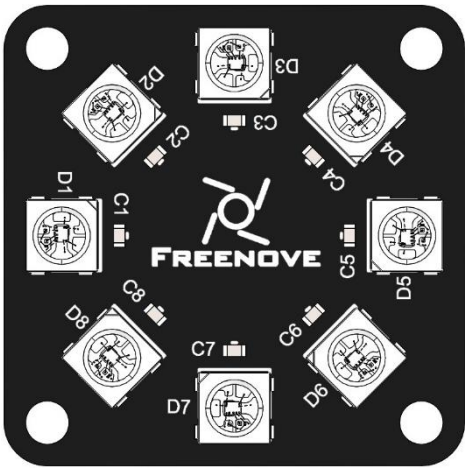
Related knowledge

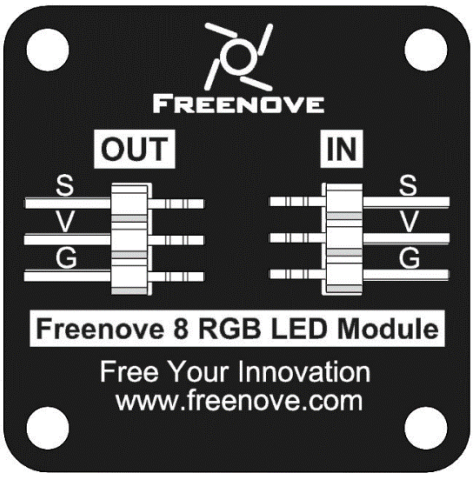
Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below.

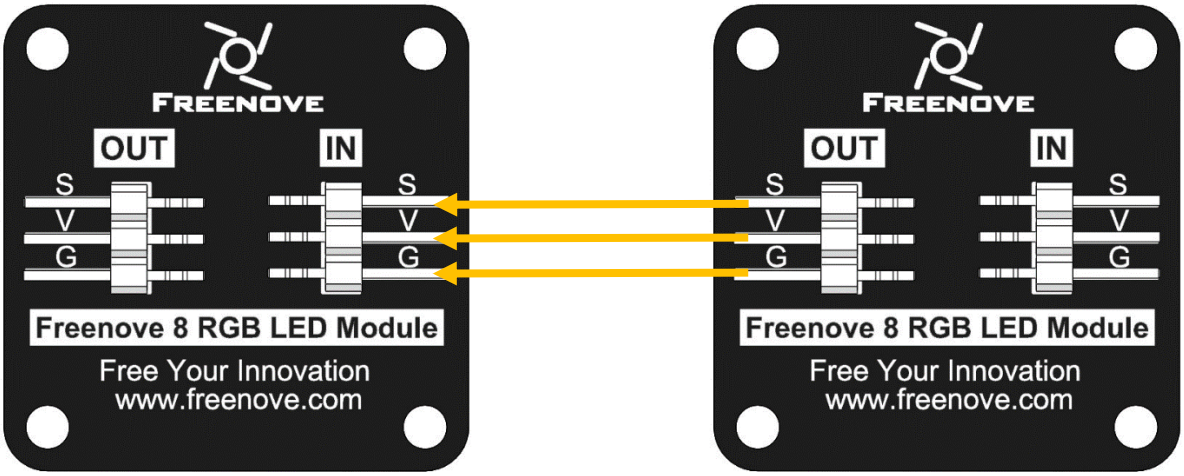
It consists of 8 WS2812, each of which requires only one pin to control and supports cascade. Each WS212 has integrated 3 LEDs, red, green and blue respectively, and each of them supports 256-level brightness adjustment, which means that each WS2812 can emit 2^24=16,777,216 different colors.

You can use only one data pin to control eight LEDs on the module. As shown below:





And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In this way, you can use one data pin to control 8, 16, 32 ··· LEDs.

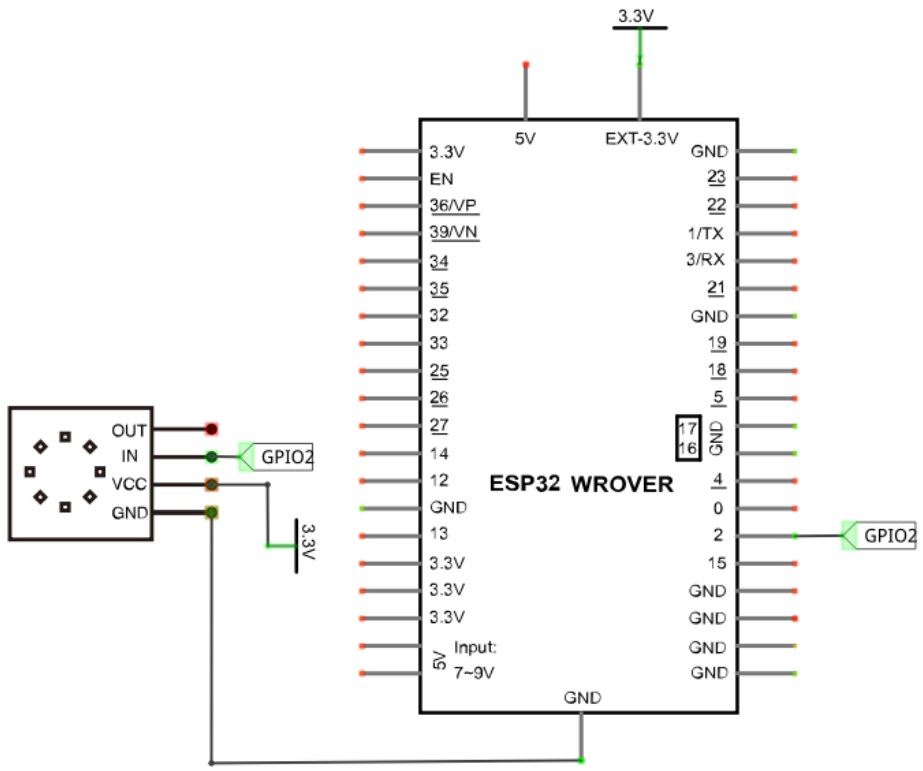


Pin description:

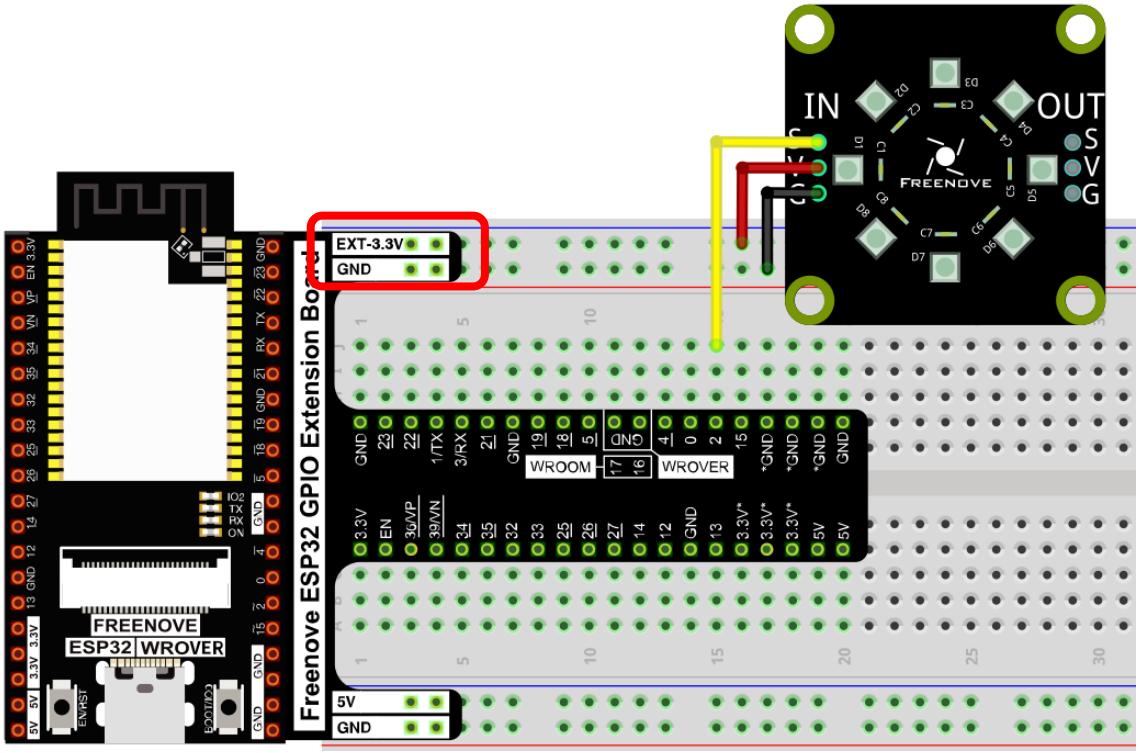
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



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



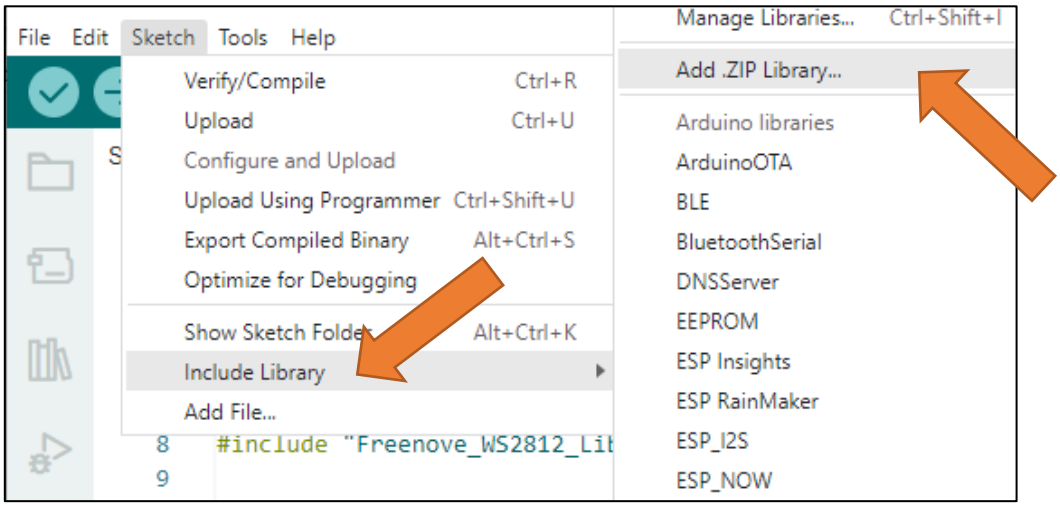
Sketch

This code uses a library named "Freenove_WS2812_Lib_for_ESP32", if you have not installed it, please do so first.

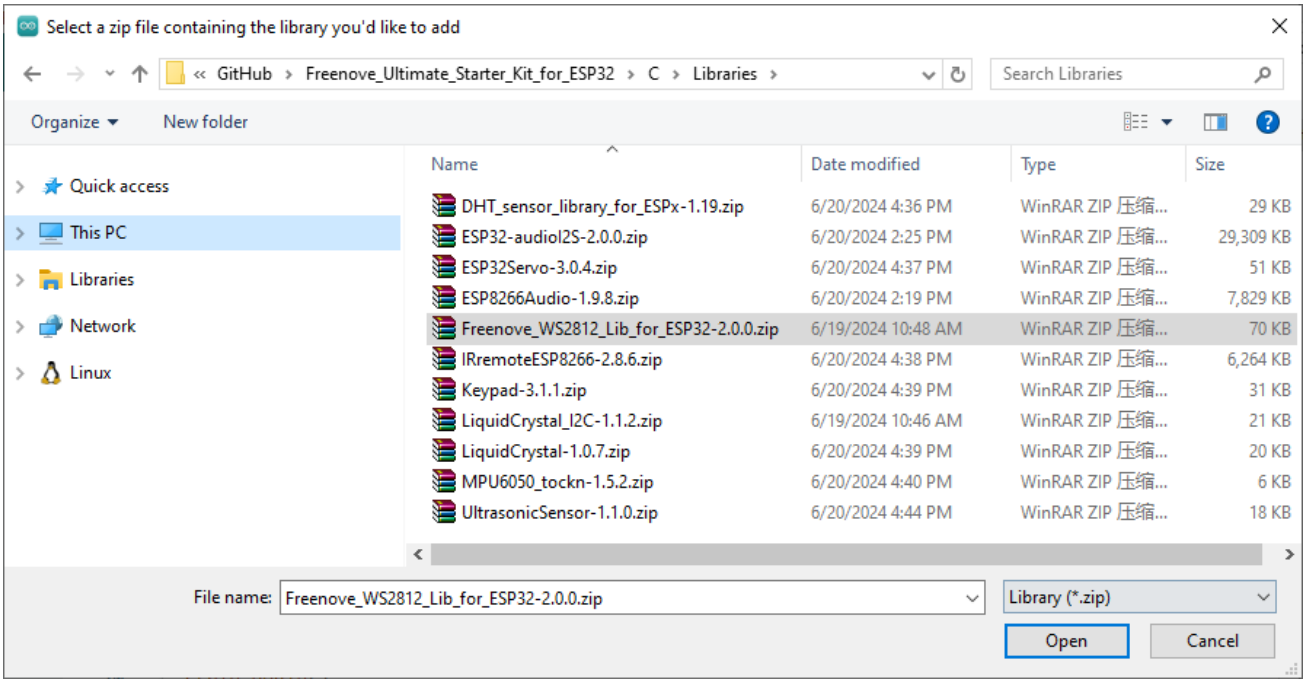
Library is an important feature of the open source world, and we know that Arduino is an open source platform that everyone can contribute to. Libraries are generally licensed under the LGPL, which means you can use them for free to apply to your creations.

How to install the library

Open Arduino IDE, click Sketch→Include Library→Add .ZIP Library, In the pop-up window, find the file named "./Libraries/Freenove_WS2812_Lib_for_ESP32.Zip" which locates in this directory, and click OPEN.



In the pop-up window, Library Manager, search for the name of the Library, "Freenove WS2812 Lib for ESP32". Then click Install.



Sketch_06.1_LEDPixel

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_06.1_LEDPixel.ino

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT  8  // The number of led
#define LEDS_PIN    2  // define the pin connected to the Freenove 8 led strip
#define CHANNEL     0  // RMT module channel

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

int m_color[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255}, {0, 0, 0} };
int delayval = 100;

void setup() {
  strip.begin();
  strip.setBrightness(10);
}
void loop() {
  for (int j = 0; j < 5; j++) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, m_color[j][0], m_color[j][1], m_color[j][2]);// Set color data.
      strip.show();   // Send color data to LED, and display.
      delay(delayval);// Interval time of each LED.
    }
    delay(500);       // Interval time of each group of colors.
  }
}
```

Download the code to ESP32-WROVER and RGB LED begins to light up in red, green, blue, white and black.

The following is the program code:

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT  8  // The number of led
#define LEDS_PIN    2  // define the pin connected to the Freenove 8 led strip
#define CHANNEL     0  // RMT channel

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

u8 m_color[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255}, {0, 0, 0} };
int delayval = 100;

void setup() {
    strip.begin();
    strip.setBrightness(10);
```

```
}
void loop() {
    for (int j = 0; j < 5; j++) {
        for (int i = 0; i < LEDS_COUNT; i++) {
            strip.setLedColorData(i, m_color[j][0], m_color[j][1], m_color[j][2]);
            strip.show();
            delay(delayval);
        }
        delay(500);
    }
}
```

To use some libraries, first you need to include the library's header file.

```
#include "Freenove_WS2812_Lib_for_ESP32.h"
```

Define the pins connected to the ring, the number of LEDs on the ring, and RMT channel values。

```
#define LEDS_COUNT  8  // The number of led
#define LEDS_PIN    2  // define the pin connected to the Freenove 8 led strip
#define CHANNEL     0  // RMT channel
```

Use the above parameters to create a LEDPixel object strip.

```
Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);
```

Define the color values to be used, as red, green, blue, white, and black.

```
u8 m_color[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255}, {0, 0, 0} };
```

Define a variable to set the time interval for each led to light up. The smaller the value is, the faster it will light up.

```
int delayval = 50;
```

Initialize strip() in setup() and set the brightness.

```
  strip.begin();
  strip.setBrightness(10);
```

In the loop(), there are two “for” loops, the internal for loop to light the LED one by one, and the external for loop to switch colors. strip.setLedColorData() is used to set the color, but it does not change immediately. Only when strip.show() is called will the color data be sent to the LED to change the color.

```
    for (int j = 0; j < 5; j++) {
        for (int i = 0; i < LEDS_COUNT; i++) {
            strip.setLedColorData(i, m_color[j][0], m_color[j][1], m_color[j][2]);
            strip.show();
            delay(delayval);
        }
        delay(500);
    }
```

Reference

`Freenove_ESP32_WS2812(u16 n = 8, u8 pin_gpio = 2, u8 chn = 0, LED_TYPE t = TYPE_GRB)`
Constructor to create a LEDPixel object. Before each use of the constructor, please add "**#include "Freenove_WS2812_Lib_for_ESP32.h**" **Parameters** **n**: The number of led. **pin_gpio**: A pin connected to an led. **Chn:** RMT channel, which uses channel 0 by default, has a total of eight channels, 0-7. This means that you can use eight LEDPixel modules for the display at the same time, and these modules do not interfere with each other **t**: Types of LED. TYPE_RGB: The sequence of LEDPixel module loading color is red, green and blue. TYPE_RBG: The sequence of LEDPixel module loading color is red, blue and green. TYPE_GRB: The sequence of LEDPixel module loading color is green, red and blue. TYPE_GBR: The sequence of LEDPixel module loading color is green, blue and red. TYPE_BRG: The sequence of LEDPixel module loading color is blue, red and green. TYPE_BGR: The sequence of LEDPixel module loading color is blue,g reen and red.
`void begin(void);`
Initialize the LEDPixel object

`void setLedColorData (u8 index, u8 r, u8 g, u8 b);`
`void setLedColorData (u8 index, u32 rgb);`
`void setLedColor (u8 index, u8 r, u8 g, u8 b);`
`void setLedColor (u8 index, u32 rgb);`

Set the color of led with order number n.
`void show(void);`
Send the color data to the led and display the set color immediately.
`void setBrightness(uint8_t);`
Set the brightness of the LED.

If you want to learn more about this library, you can visit the following website:
https://github.com/Freenove/Freenove_WS2812_Lib_for_ESP32

Project 6.2 Rainbow Light

In the previous project, we have mastered the use of LEDPixel. This project will realize a slightly complicated rainbow light. The component list and the circuit are exactly the same as the project fashionable light.

Sketch

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

Sketch_06.2_RainbowLight

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT  8
#define LEDS_PIN  2
#define CHANNEL   0

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

void setup() {
  strip.begin();
}

void loop() {
  for (int j = 0; j < 255; j += 2) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
    }
    strip.show();
    delay(5);
  }
}
```

Download the code to ESP32-WROVER, and the Freenove 8 RGB LED Strip displays different colors and the color changes gradually.

The following is the program code:

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT  8  // The number of led
#define LEDS_PIN    2  // define the pin connected to the Freenove 8 led strip
#define CHANNEL     0  // RMT channel

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

void setup() {
  strip.begin();
}

void loop() {
  for (int j = 0; j < 255; j += 1) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
    }
    strip.show(); // Send color data to LED, and display.
    delay(5);
  }
}
```

In the loop(), two “for” loops are used, the internal “for” loop(for-j) is used to set the color of each LED, and the external “for” loop(for-i) is used to change the color, in which the self-increment value in i+=1 can be changed to change the color step distance. Changing the delay parameter changes the speed of the color change. Wheel(i * 256 / LEDS_COUNT + j) & 255) will take color from the color model at equal intervals starting from i.

```
  for (int j = 0; j < 255; j += 1) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
    }
    strip.show(); // Send color data to LED, and display.
    delay(5);
  }
```

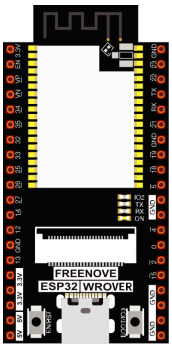
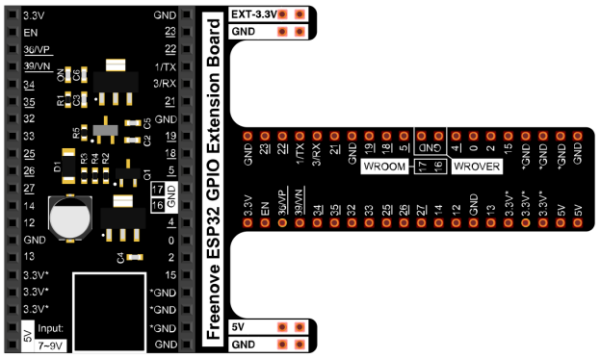
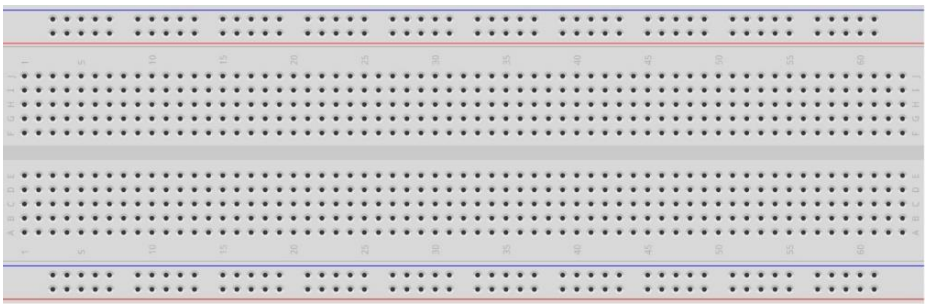
Chapter 7 Buzzer

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

Project 7.1 Doorbell

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

Component List

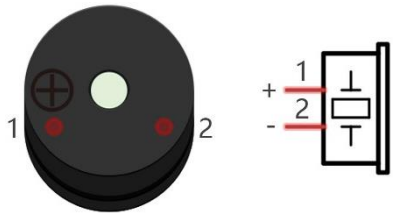
ESP32-WROVER x1 	GPIO Extension Board x1 			
Breadboard x1 				
Jumper M/M x6				
NPN transistorx1 (S8050)	Active buzzer x1	Push button x1	Resistor 1kΩ x1	Resistor 10kΩ x2

Component knowledge

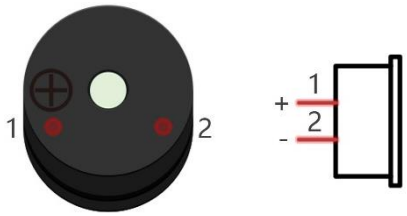
Buzzer

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

Active buzzer



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

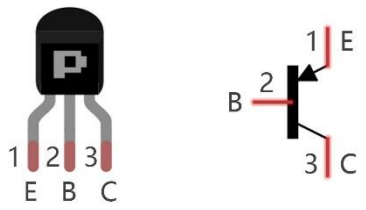
Passive buzzer

Transistor

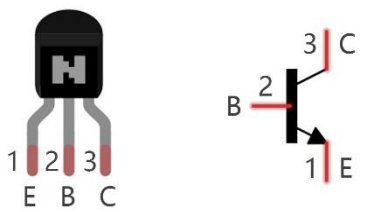
Because the buzzer requires such large current that GPIO of ESP32 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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



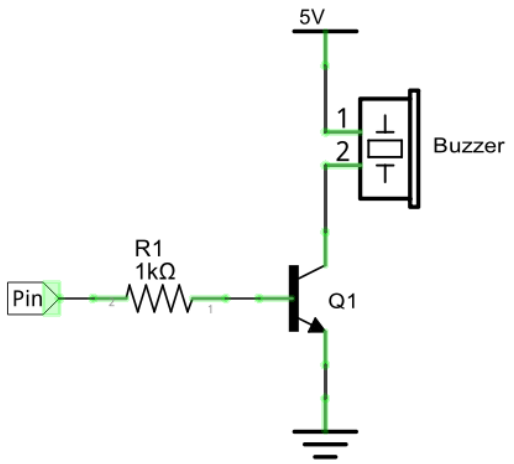
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

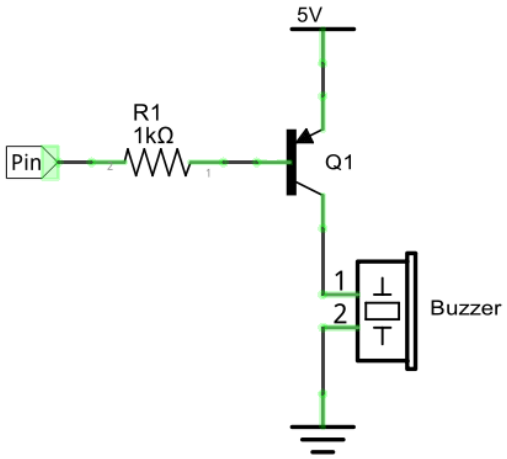
When use NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

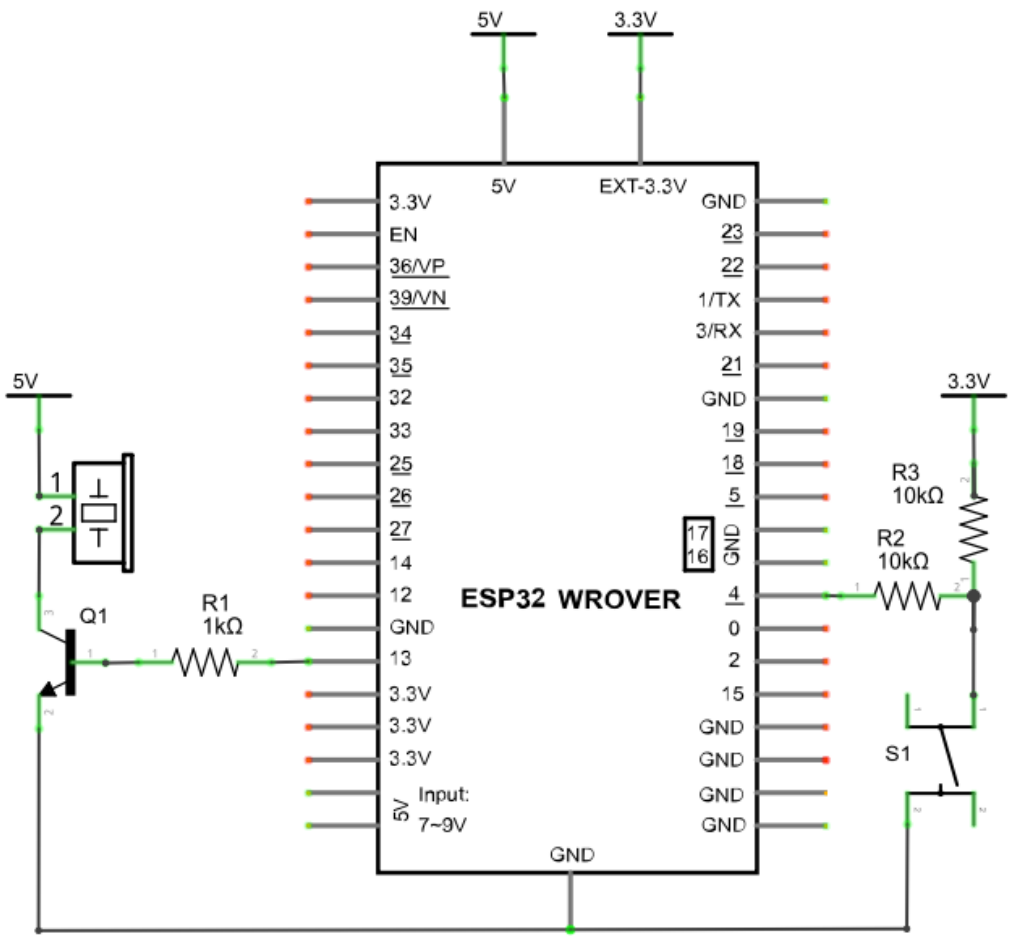


PNP transistor to drive buzzer

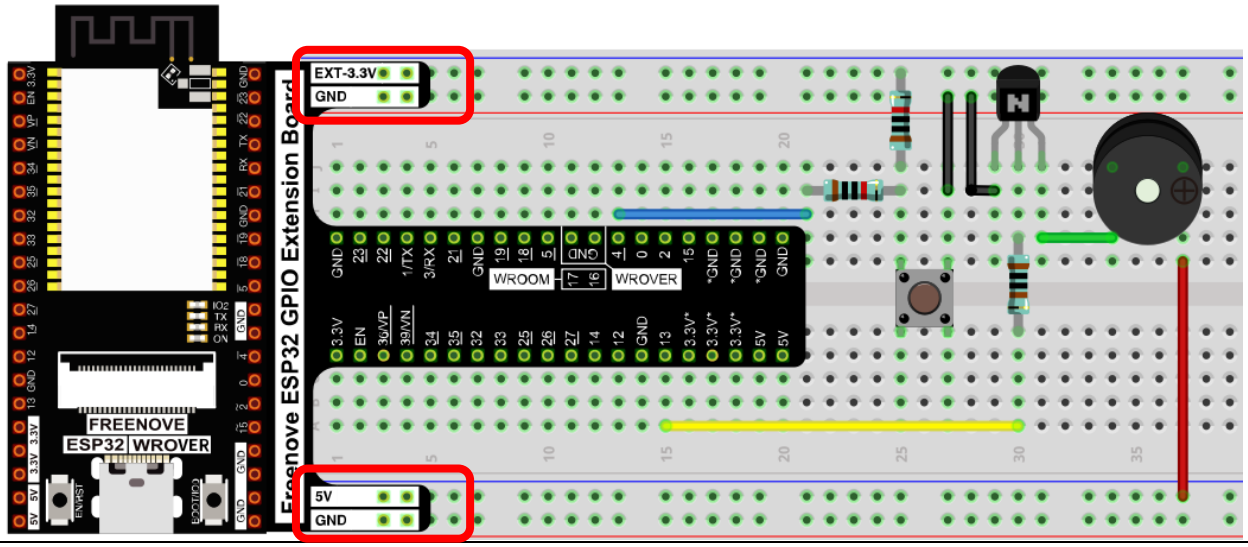


Circuit

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_07.1_Doorbell

Sketch_07.1_Doorbell | Arduino 1.8.12

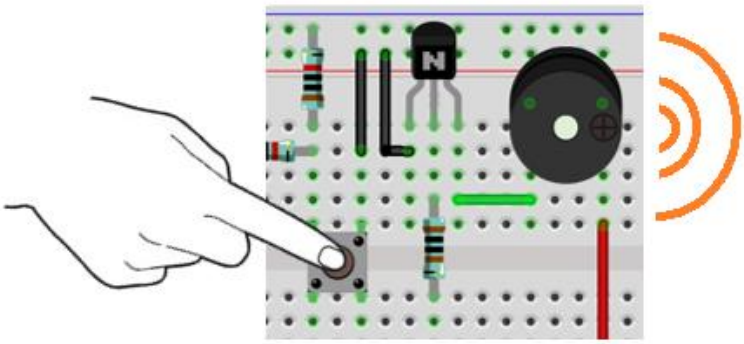
File Edit Sketch Tools Help

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 7.2 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_07.2_Alertor

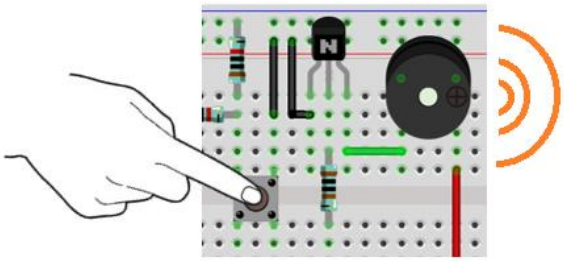
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_07.2_Aleror.ino

```
#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.

```
  ledcAttachChannal(PIN_BUZZER, 0, 10, CHN);  //attach the led pin to pwm channel
  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.

```
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);
  }
}
```

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

```
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 7.3 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_07.3_Alertor

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_07.3_Aleror.ino

```
#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;
```

```
    timerAlarmWrite(timer, 500000 / toneVal, true, 0);
    delay(1);
  }
}
```

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.

```
  hw_timer_t * timer = NULL;

  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);
```

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.

```
    if (! isAlerting) {
      isAlerting = true;
      // Set alarm, 1ms, repeat
      timerAlarmWrite(timer, 1000, true, 0);
      // Start an alarm
      timerAlarmEnable(timer);
    }
```

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

```
    if (isAlerting) {
      isAlerting = false;
      timerAlarmDisable(timer);
      digitalWrite(BUZZER_PIN, LOW);
    }
```

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.

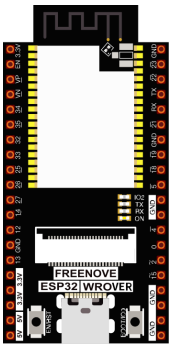
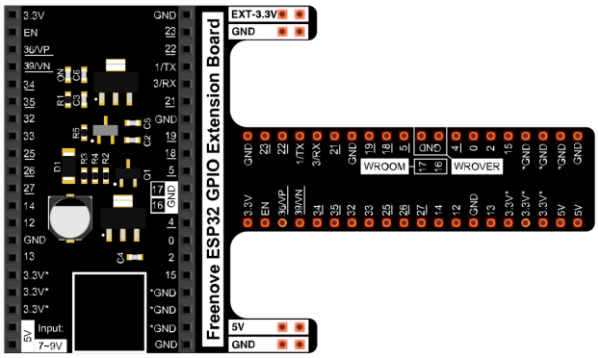
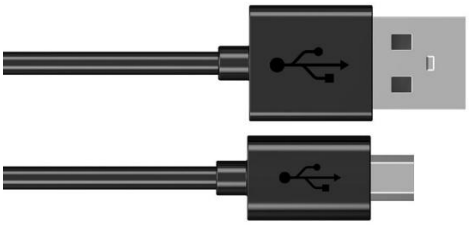
Chapter 8 Serial Communication

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

Project 8.1 Serial Print

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

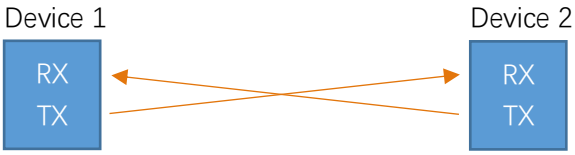
Component List

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

Related knowledge

Serial communication

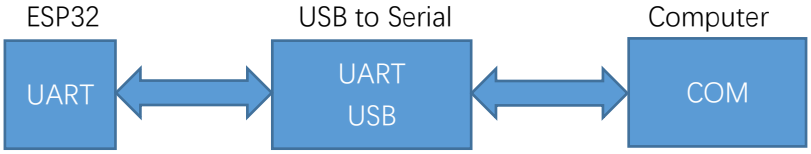
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



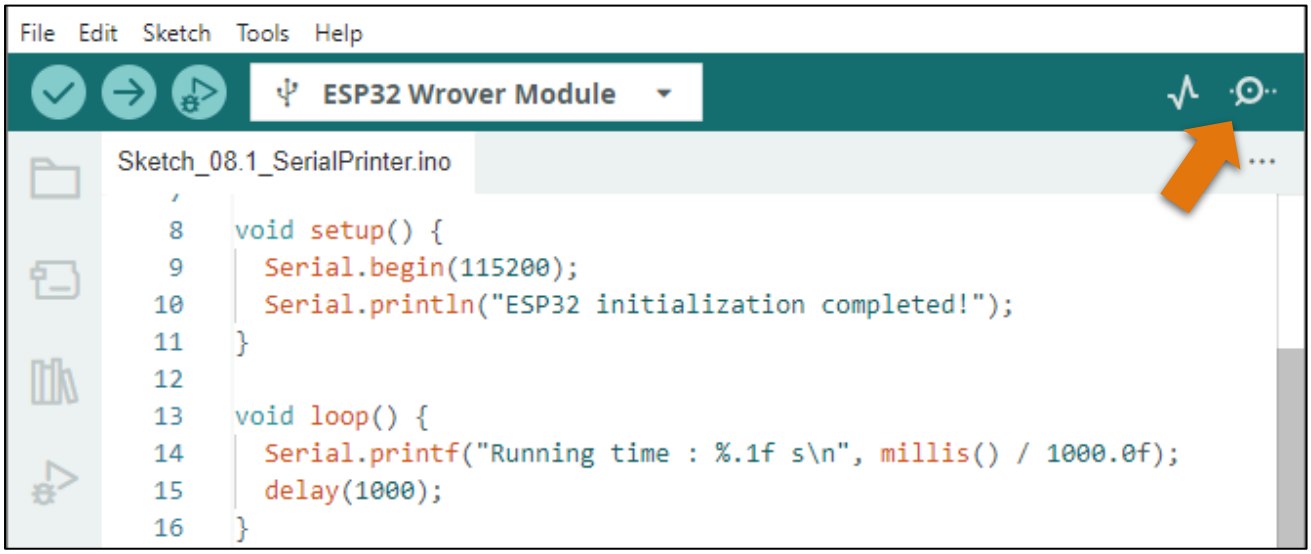
Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

Serial port on ESP32

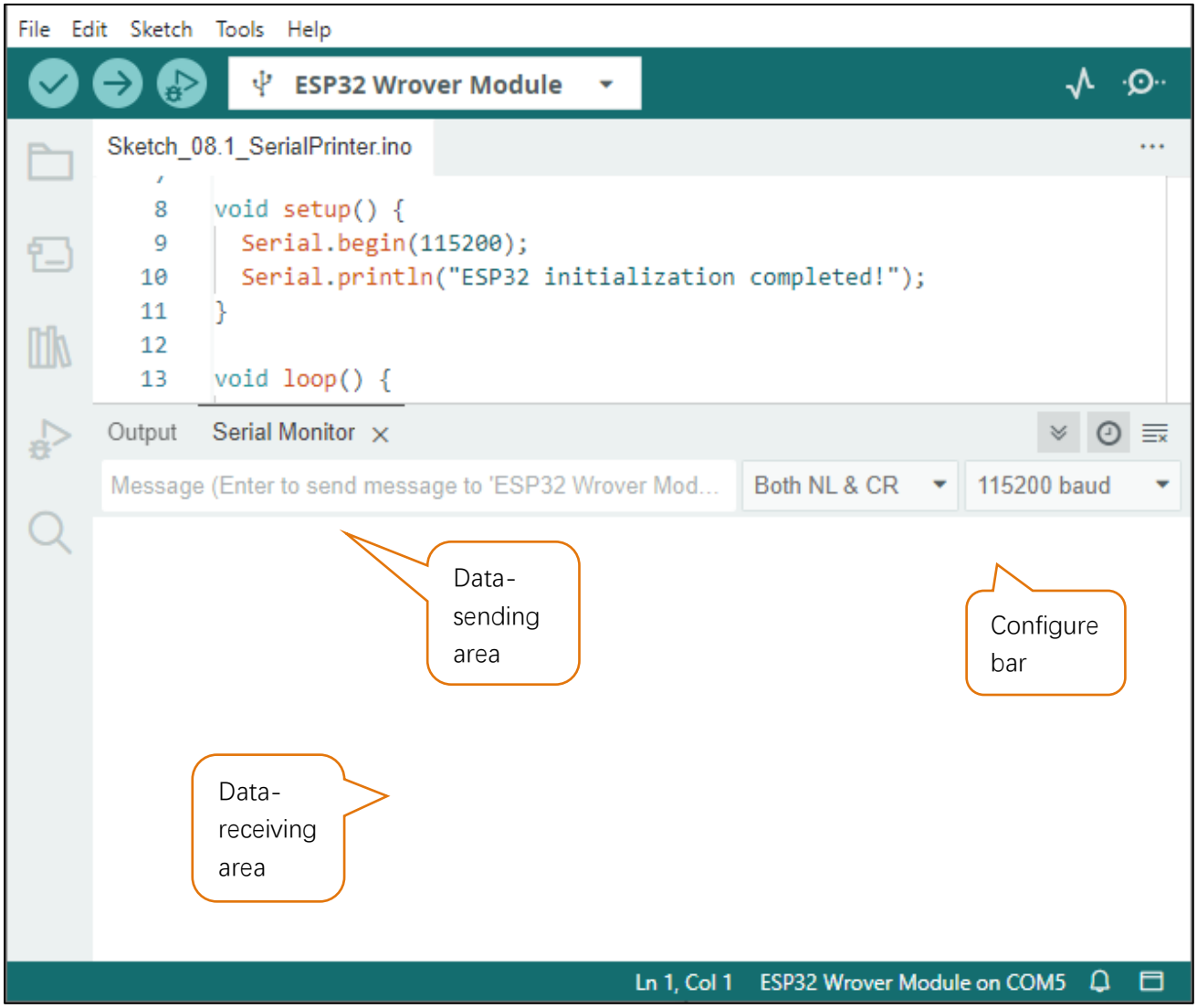
Freenove ESP32 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



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.

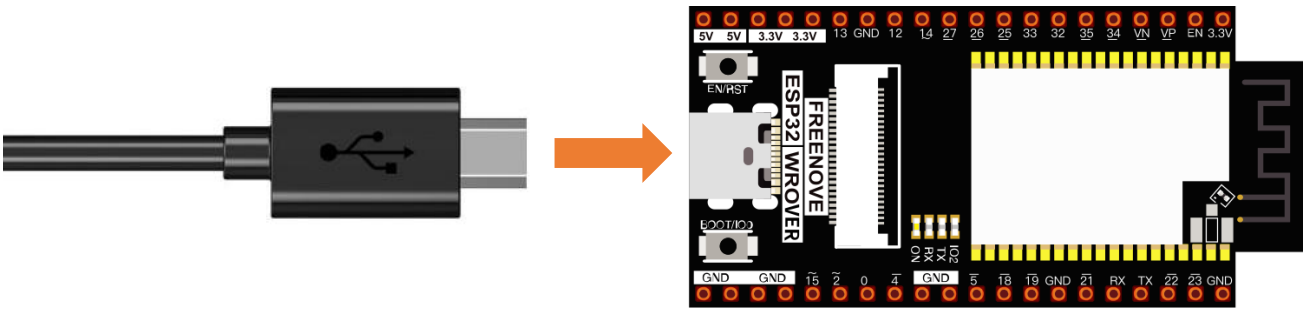


Interface of serial monitor window is as follows. If you can't open it, make sure Freenove ESP32 has been connected to the computer, and choose the right serial port in the menu bar "Tools".



Circuit

Connect Freenove ESP32 to the computer with USB cable

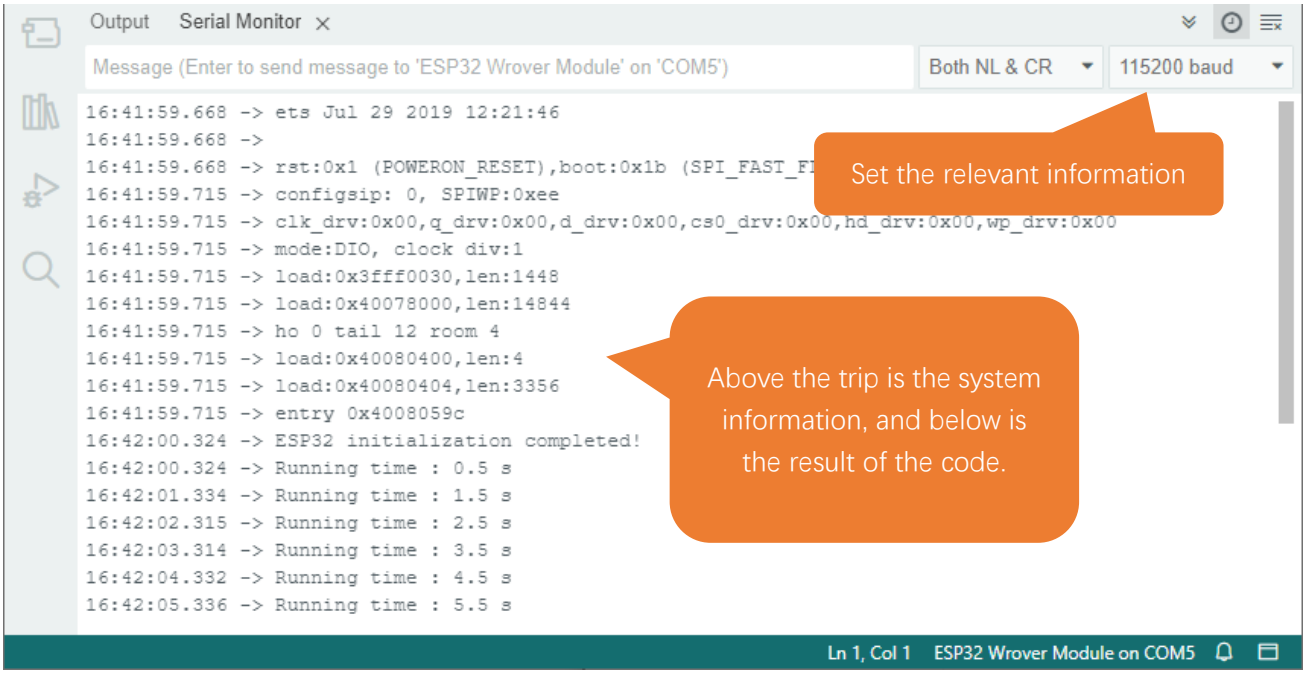


Sketch

Sketch_08.1_SerialPrinter

```
void setup() {
  Serial.begin(115200);
  Serial.println("ESP32 initialization completed!");
}

void loop() {
  Serial.printf("Running time : %.1f s\n", millis() / 1000.0f);
  delay(1000);
}
```

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:

```
void setup() {
  Serial.begin(115200);
  Serial.println("ESP32 initialization completed! ");
}

void loop() {
  Serial.printf("Running time : %.1f s\n", millis() / 1000.0f);
  delay(1000);
}
```

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 8.2 Serial Read and Write

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

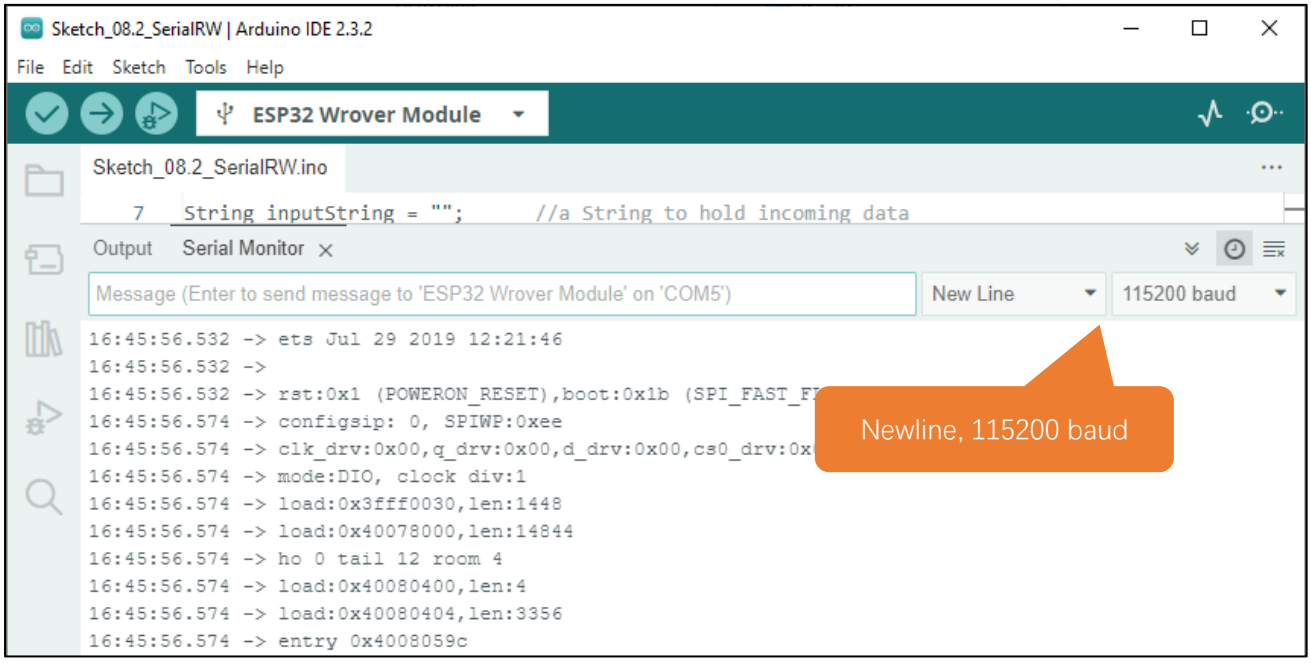
Component and circuit are the same as in the previous project.

Sketch

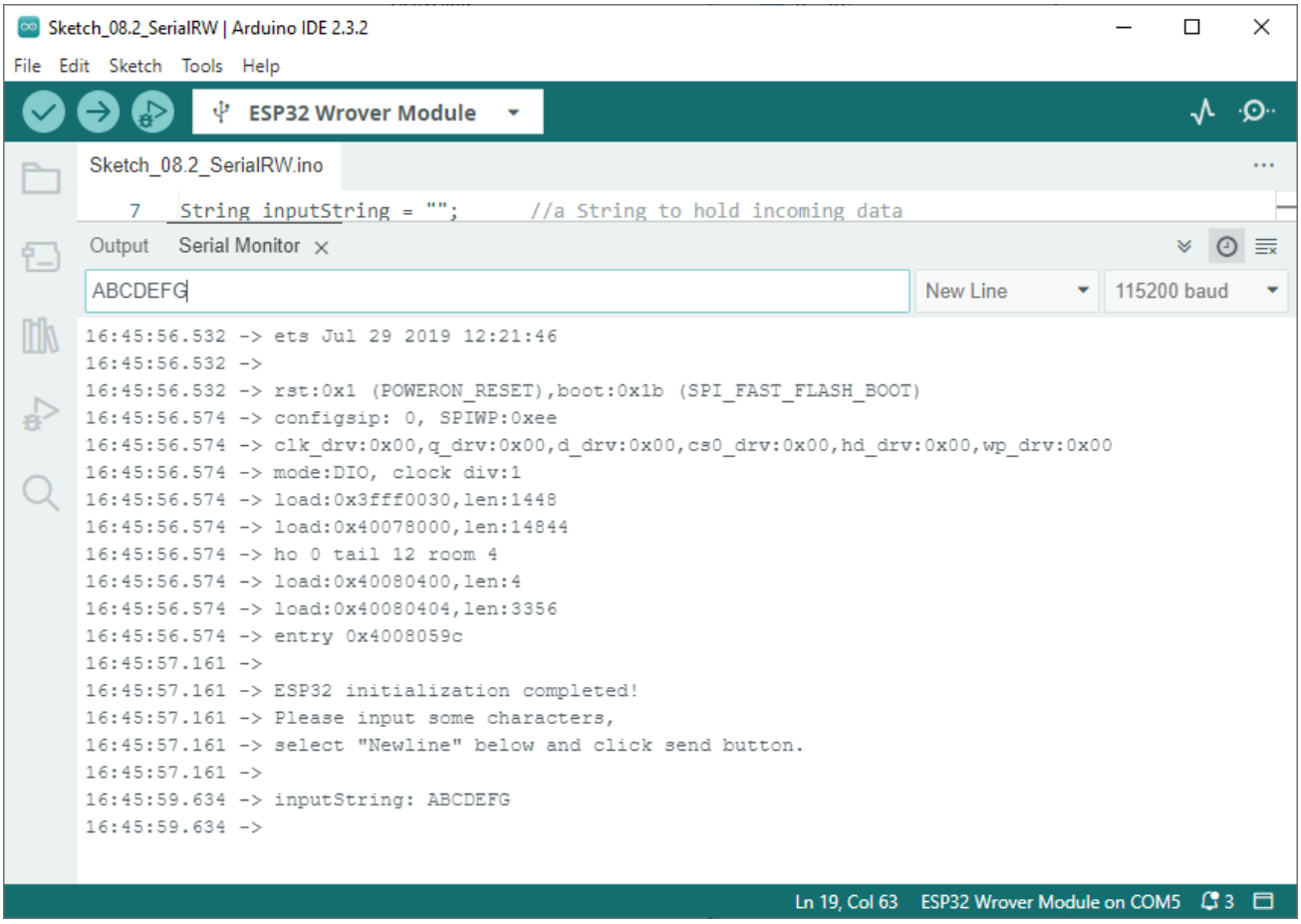
Sketch_08.2_SerialRW

```
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.

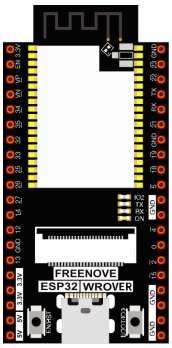
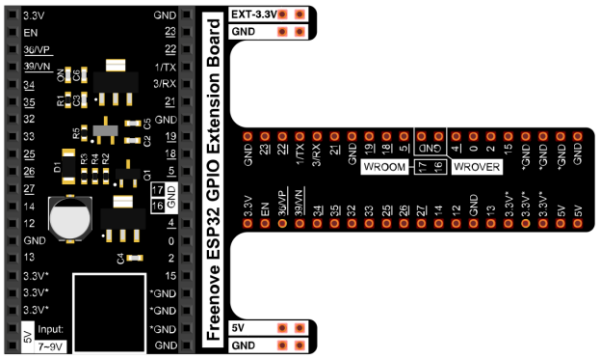
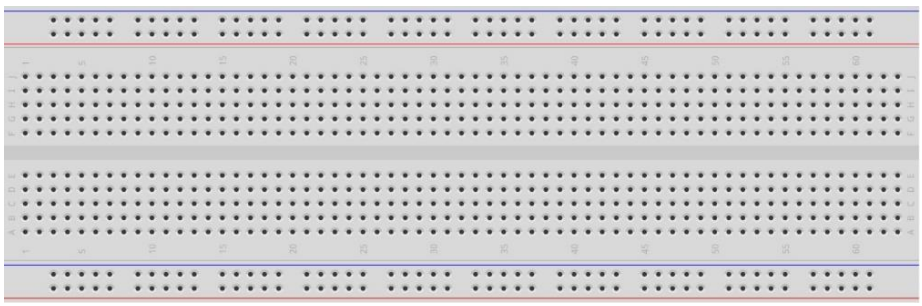
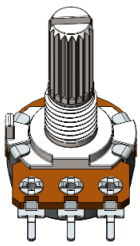
Chapter 9 AD/DA Converter

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

Project 9.1 Read the Voltage of Potentiometer

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

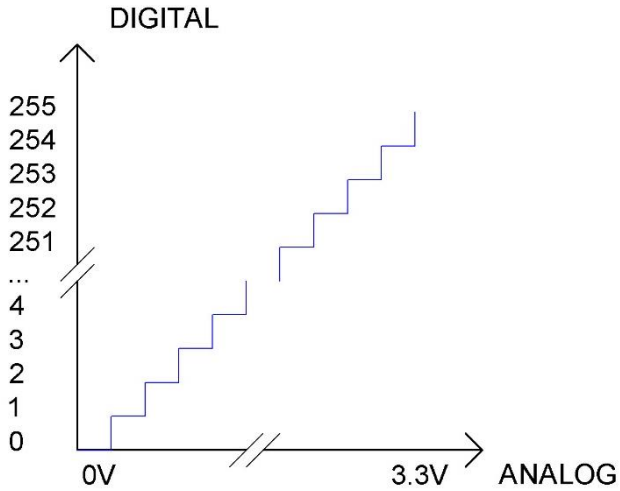
Component List

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

Related knowledge

ADC

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

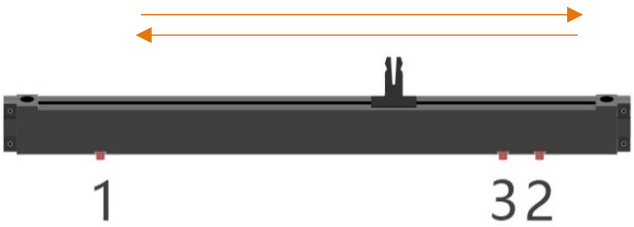
The DAC pin number is already defined in ESP32's code base; for example, you can replace GPIO25 with DAC1 in the code.

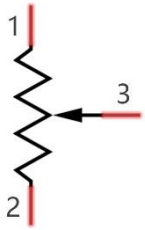
Note: In this ESP32, GPIO26 is used as the camera's IIC-SDA pin, which is connected to 3.3V through a resistor.Therefore, DAC2 cannot be used.

Component knowledge

Potentiometer

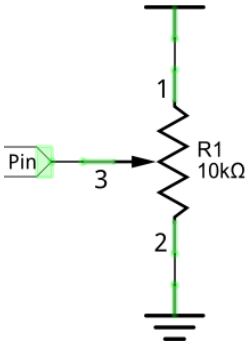
A potentiometer is a three-terminal resistor. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact 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.





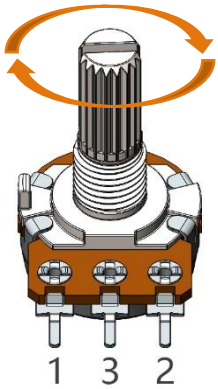
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

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.



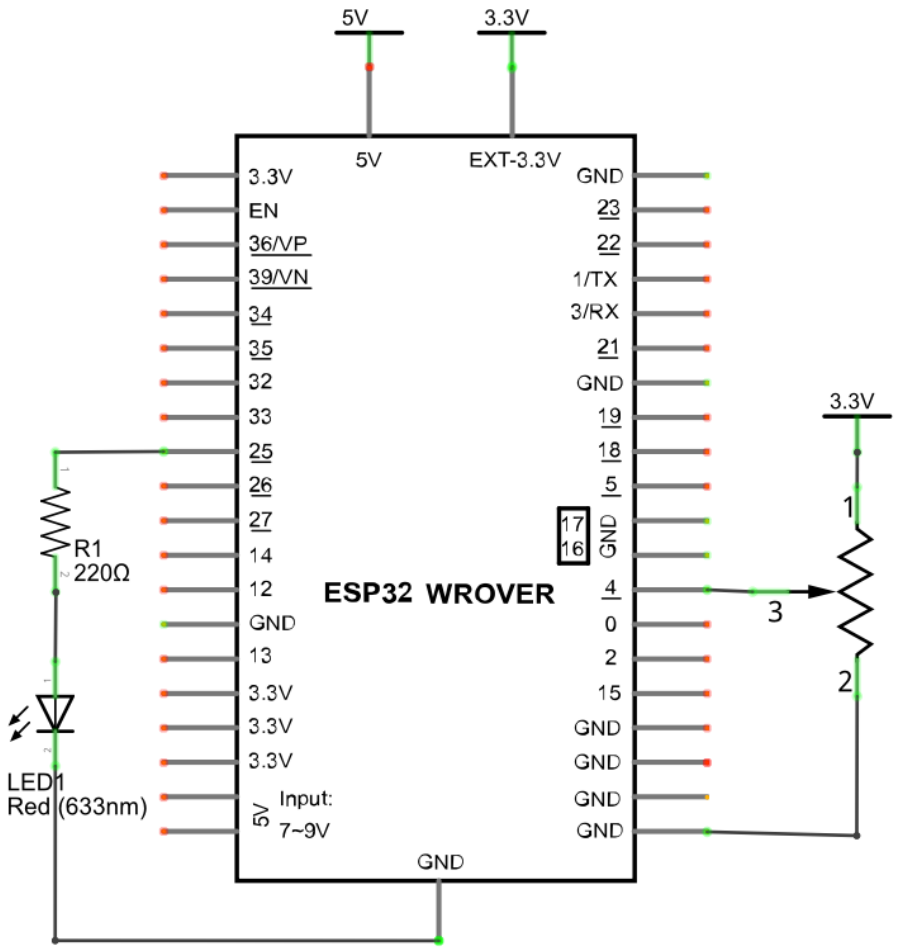
Rotary potentiometer

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

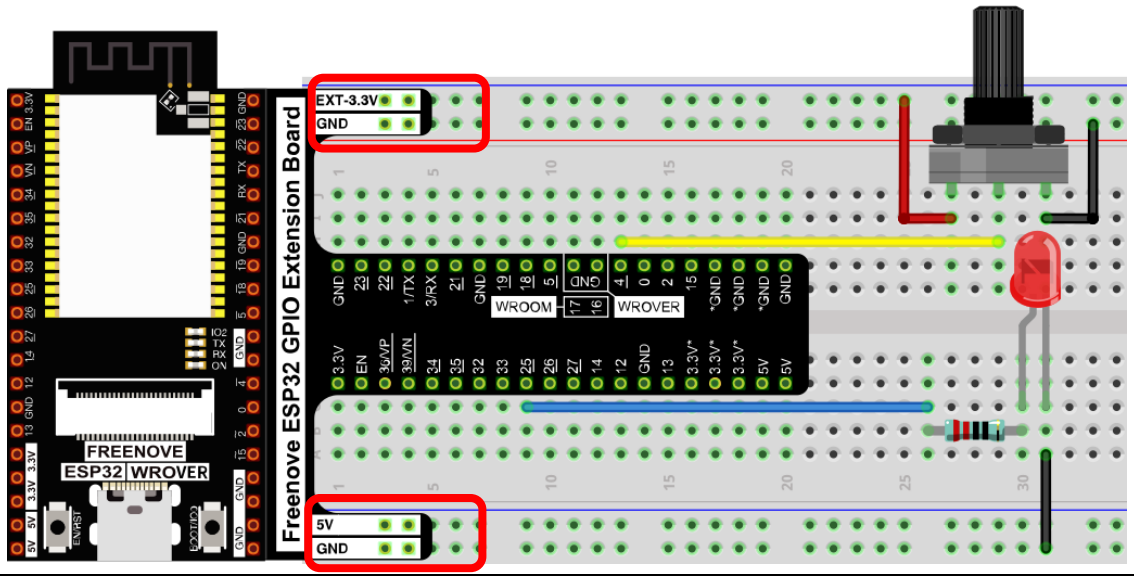


Circuit

Schematic diagram



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



Sketch

Sketch_09.1_ADC_DAC

```
#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);
}
```

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

```
17:14:25.046 -> ADC Val: 3751,   DAC Val: 233,   Voltage: 3.02V
17:14:25.231 -> ADC Val: 3733,   DAC Val: 232,   Voltage: 3.01V
17:14:25.418 -> ADC Val: 3582,   DAC Val: 223,   Voltage: 2.89V
17:14:25.637 -> ADC Val: 3341,   DAC Val: 208,   Voltage: 2.69V
17:14:25.822 -> ADC Val: 3057,   DAC Val: 190,   Voltage: 2.46V
17:14:26.000 -> ADC Val: 2875,   DAC Val: 179,   Voltage: 2.32V
17:14:26.226 -> ADC Val: 2897,   DAC Val: 180,   Voltage: 2.33V
17:14:26.400 -> ADC Val: 2874,   DAC Val: 178,   Voltage: 2.32V
17:14:26.628 -> ADC Val: 2906,   DAC Val: 180,   Voltage: 2.34V
17:14:26.800 -> ADC Val: 2849,   DAC Val: 177,   Voltage: 2.30V
17:14:27.000 -> ADC Val: 2756,   DAC Val: 171,   Voltage: 2.22V
17:14:27.231 -> ADC Val: 3003,   DAC Val: 187,   Voltage: 2.42V
17:14:27.418 -> ADC Val: 2966,   DAC Val: 184,   Voltage: 2.39V
17:14:27.635 -> ADC Val: 2951,   DAC Val: 183,   Voltage: 2.38V
17:14:27.800 -> ADC Val: 2815,   DAC Val: 175,   Voltage: 2.27V
17:14:28.023 -> ADC Val: 2862,   DAC Val: 178,   Voltage: 2.31V
17:14:28.200 -> ADC Val: 3035,   DAC Val: 188,   Voltage: 2.45V
```

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.

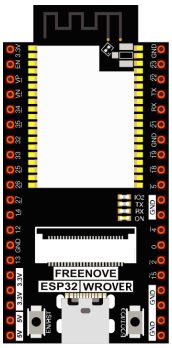
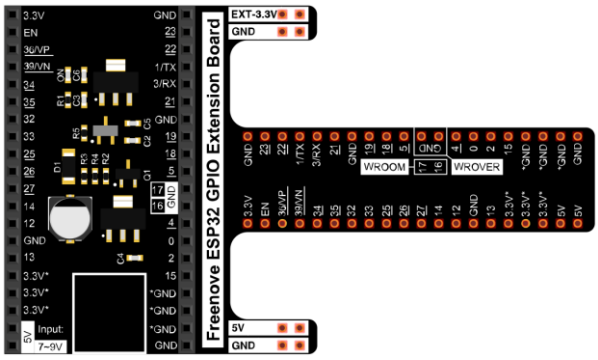
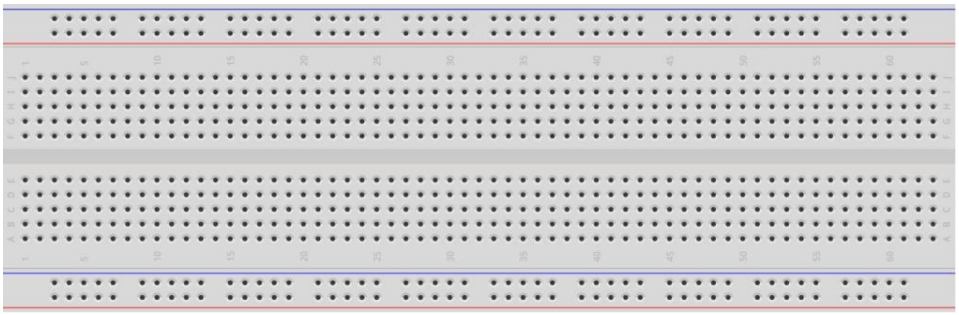
Chapter 10 Touch Sensor

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 10.1 Read Touch Sensor

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

Component List

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

Related knowledge

Touch sensor

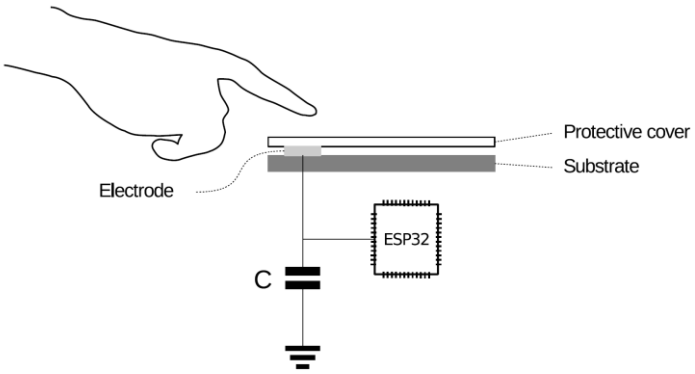
ESP32's touch sensor supports up to 10 GPIO channels as capacitive touch pins. Each pin can be used separately as an independent touch switch or be combined to produce multiple touch points. The following table is a list of available touch pins on ESP32.

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.

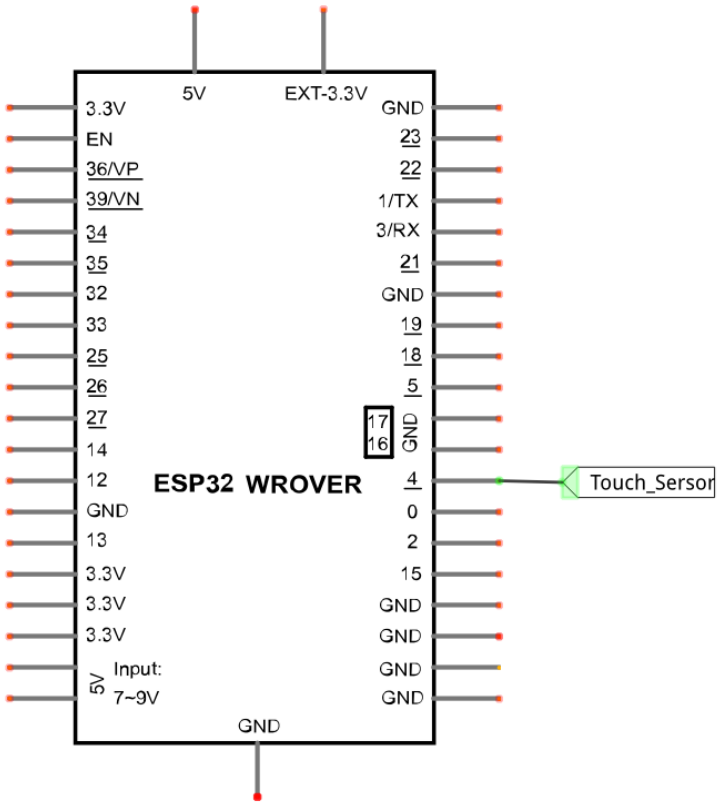
The electrical signals generated by touch are analog data, which are converted by an internal ADC converter. You may have noticed that all touch pins have ADC functionality.

The hardware connection method is shown in the following figure.

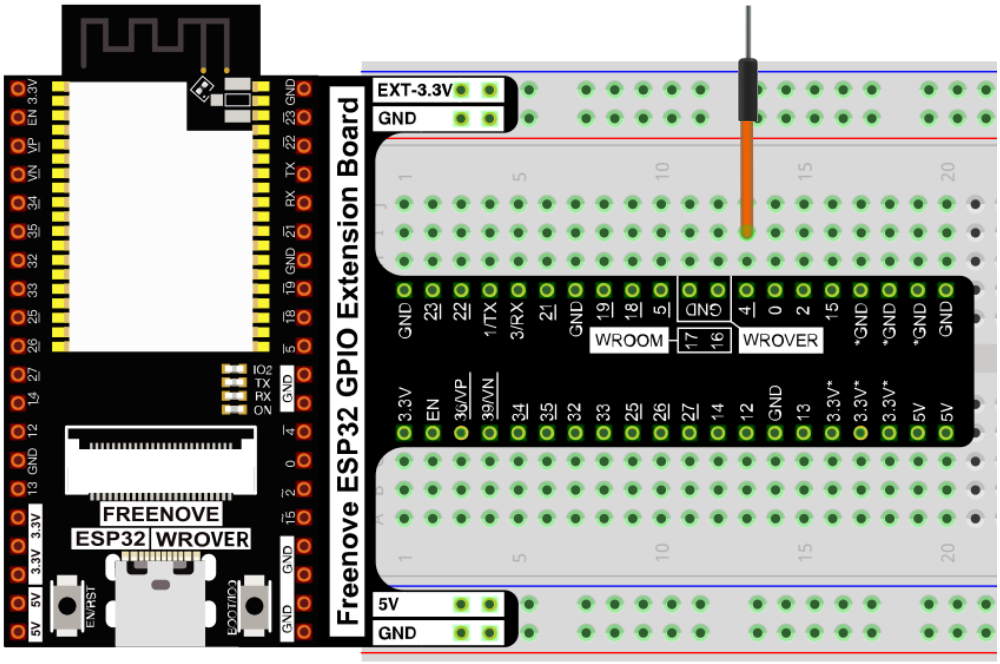


Circuit

Schematic diagram



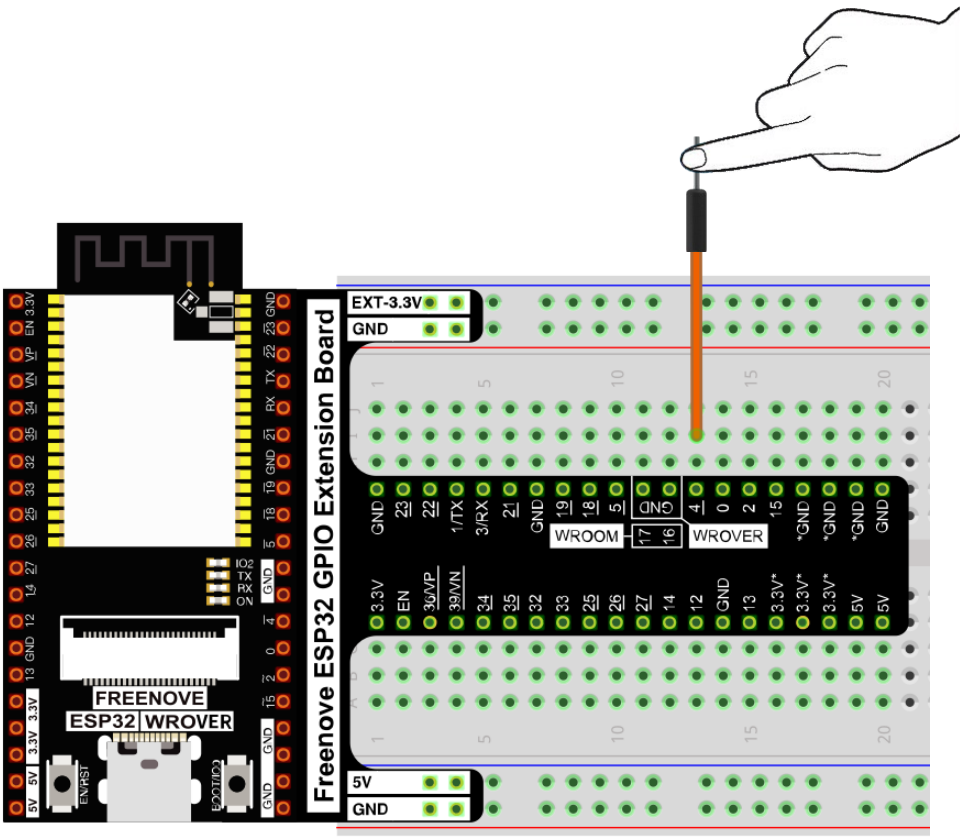
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



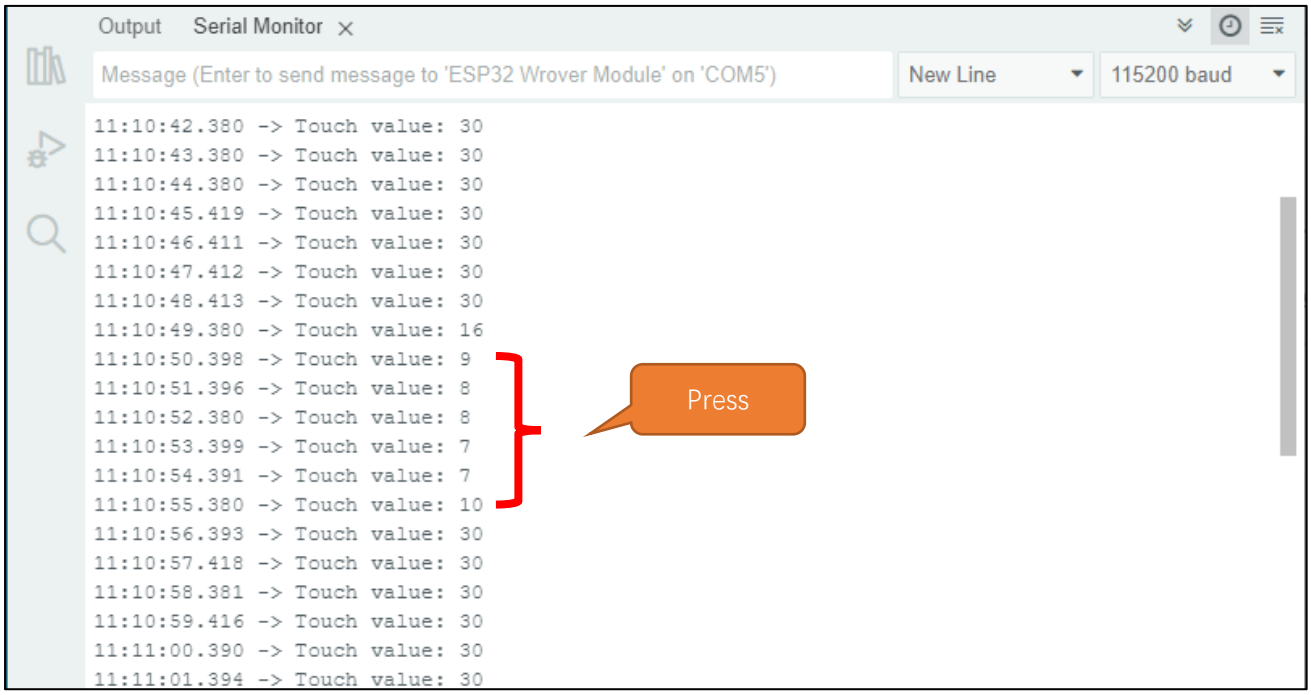
Sketch

Sketch_10.1_TouchRead

```
/*****************************************************************
  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.

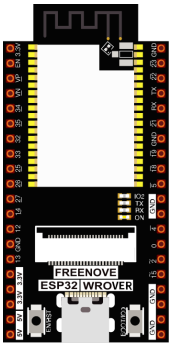
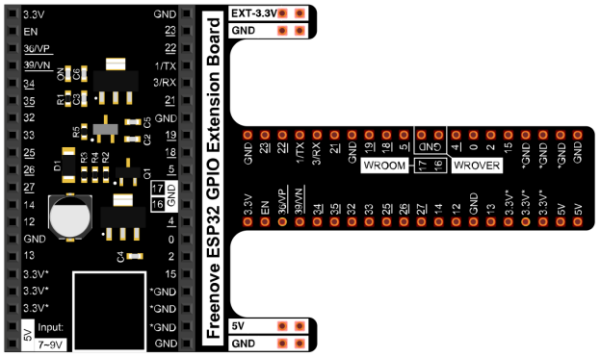
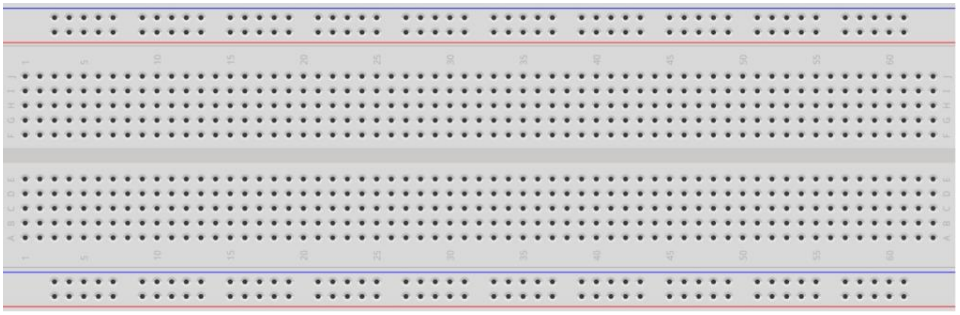
Reference

`uint16_t touchRead(uint8_t pin);`
Read touch sensor value. (values close to 0 mean touch detected)

Project 10.2 Touch Lamp

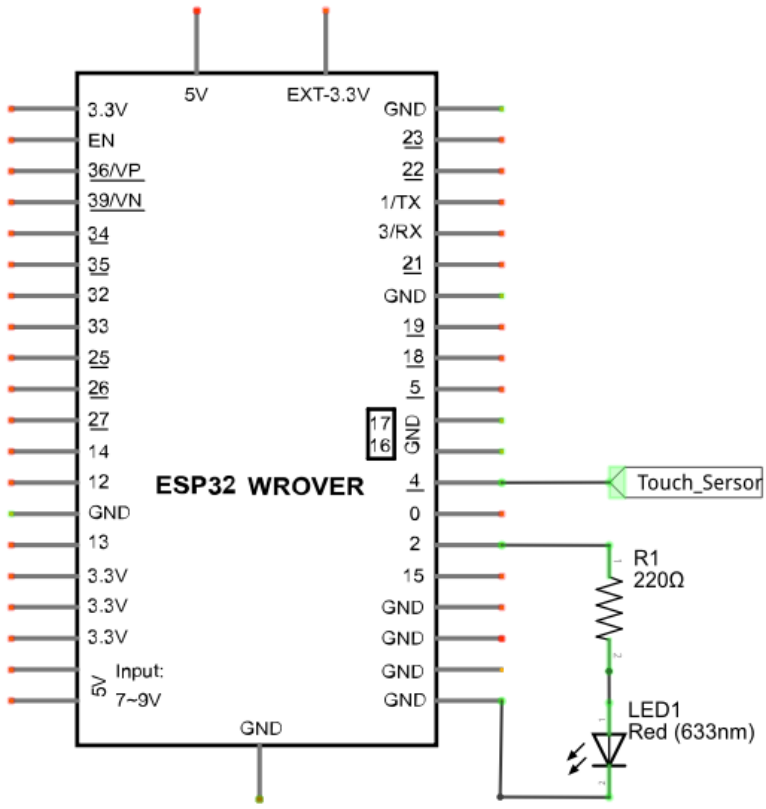
In this project, we will use ESP32's touch sensor to create a touch switch lamp.

Component List

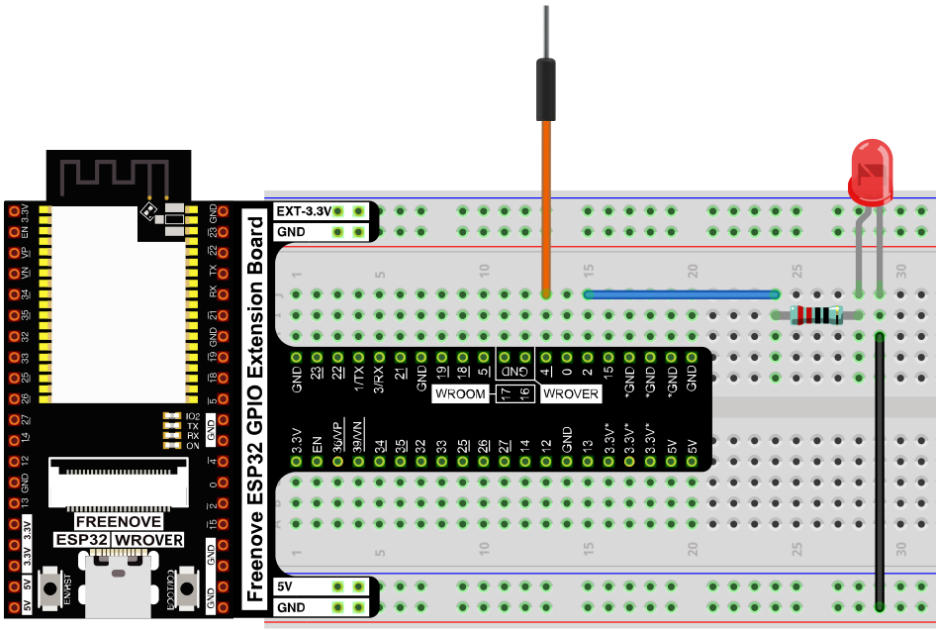
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x3	LED x1	Resistor 220Ω x1

Circuit

Schematic diagram



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



Sketch

Sketch_10.2_TouchLamp

```
#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! ");
    }
  }
}
```

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

```
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.

```
  if (touchRead(T0) < PRESS_VAL) {
    if (! isProcessed) {
      isProcessed = true;
      Serial.println("Touch detected!  ");
      reverseGPIO(PIN_LED);
    }
  }
```

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.

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

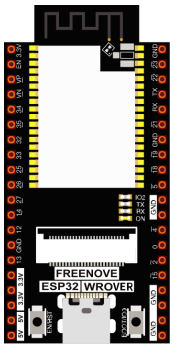
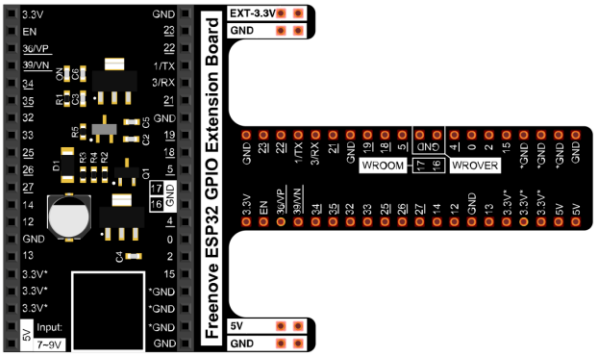
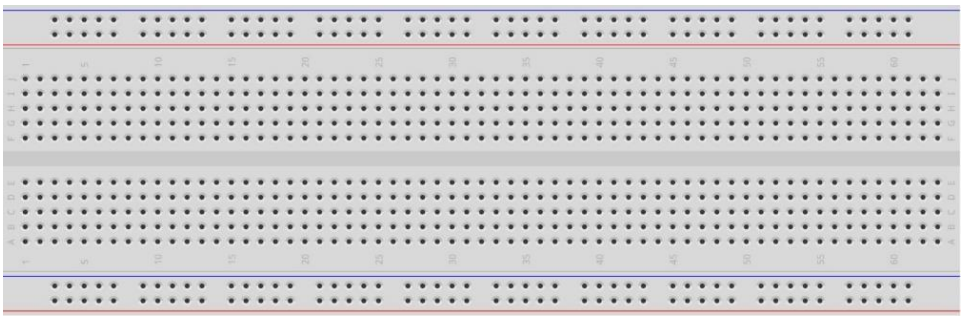
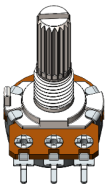
Chapter 11 Potentiometer & LED

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

Project 11.1 Soft Light

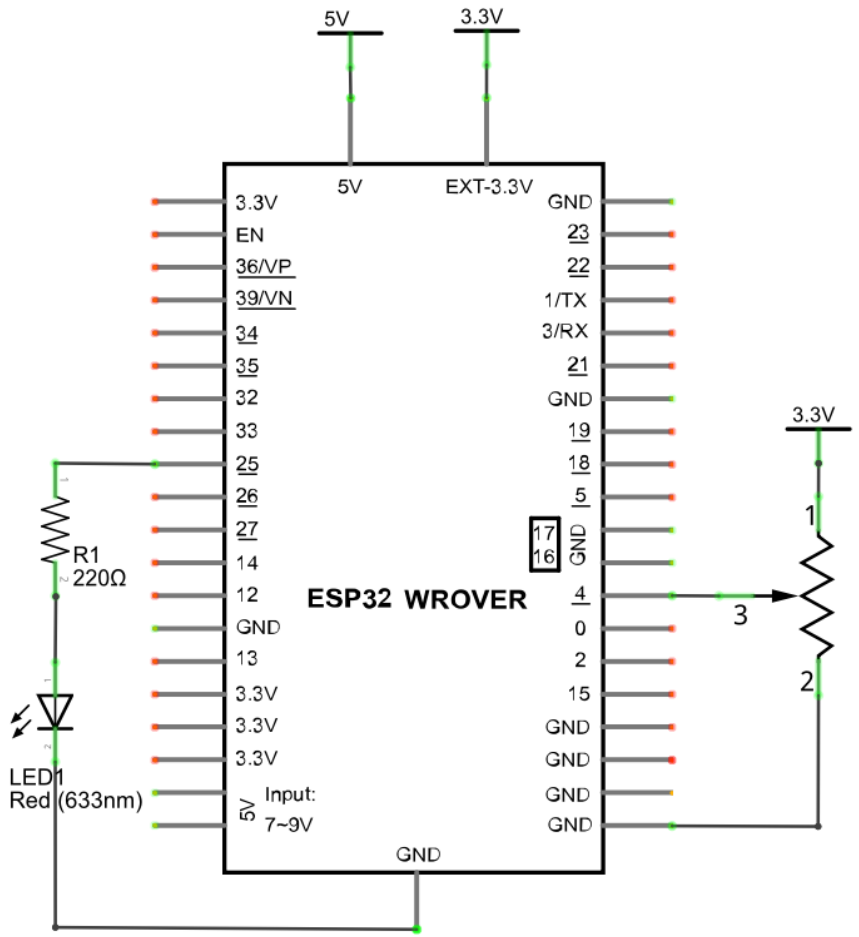
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of a LED. Then you can change the brightness of a LED by adjusting the potentiometer.

Component List

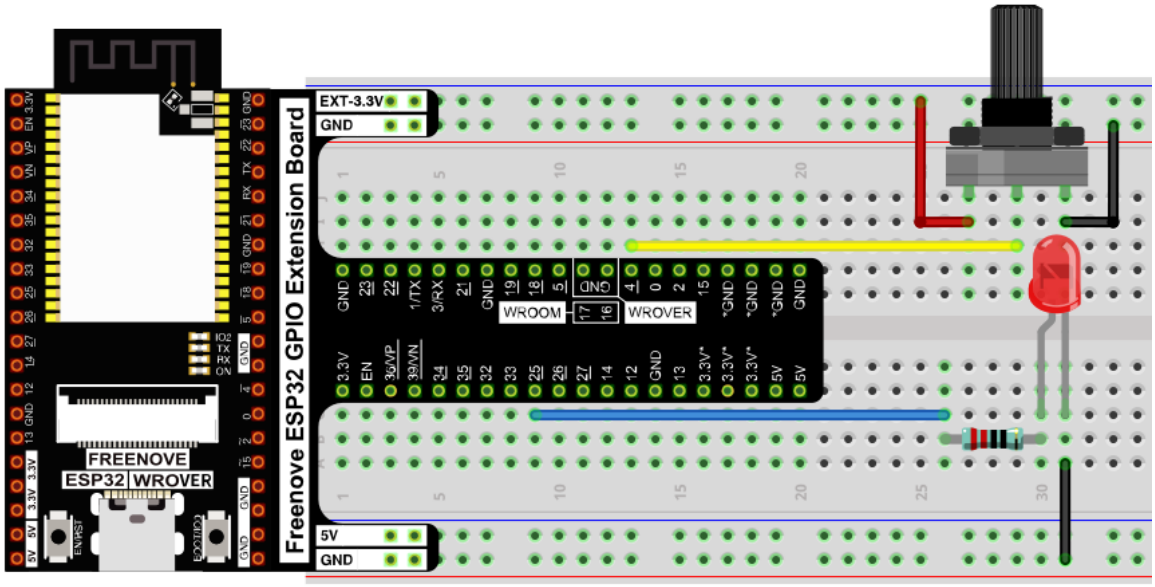
ESP32-WROVER x1 	GPIO Extension Board x1 		
Breadboard x1 			
Rotary potentiometer x1 	Resistor 220Ω x1	LED x1	Jumper M/M x5

Circuit

Schematic diagram



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



Sketch

Sketch_11.1_Softlight

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_11.1_SoftLight.ino

```
/*********************************************************************
  Filename    : SoftLight
  Description : Controlling the brightness of LED by potentiometer.
  Auther      : www.freenove.com
  Modification: 2024/06/18
*********************************************************************/
#define PIN_ANALOG_IN   4
#define PIN_LED         25
#define CHAN            0
void setup() {
  ledcAttachChannel(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);
}
```

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:

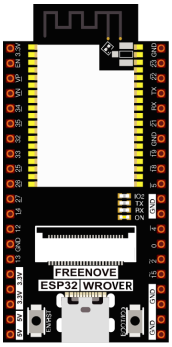
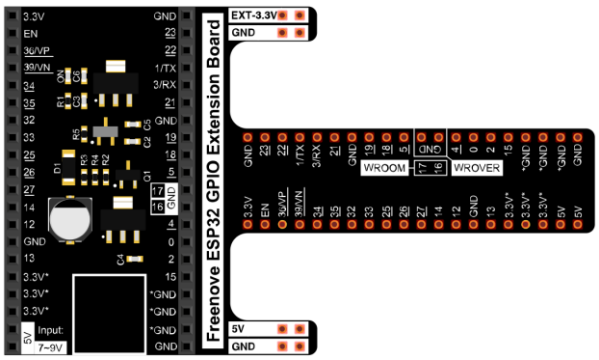
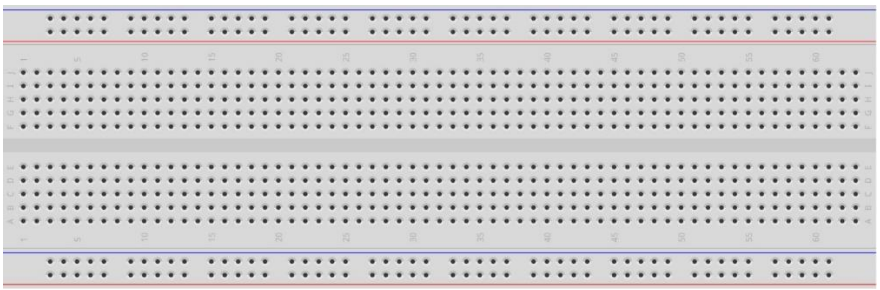
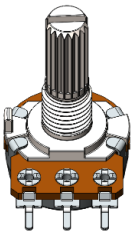
```
#define PIN_ANALOG_IN   4
#define PIN_LED         25
#define CHAN            0
void setup() {
  ledcAttachChannal(PIN_LED, 1000, 12, CHAN);
}

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.

Project 11.2 Soft Colorful Light

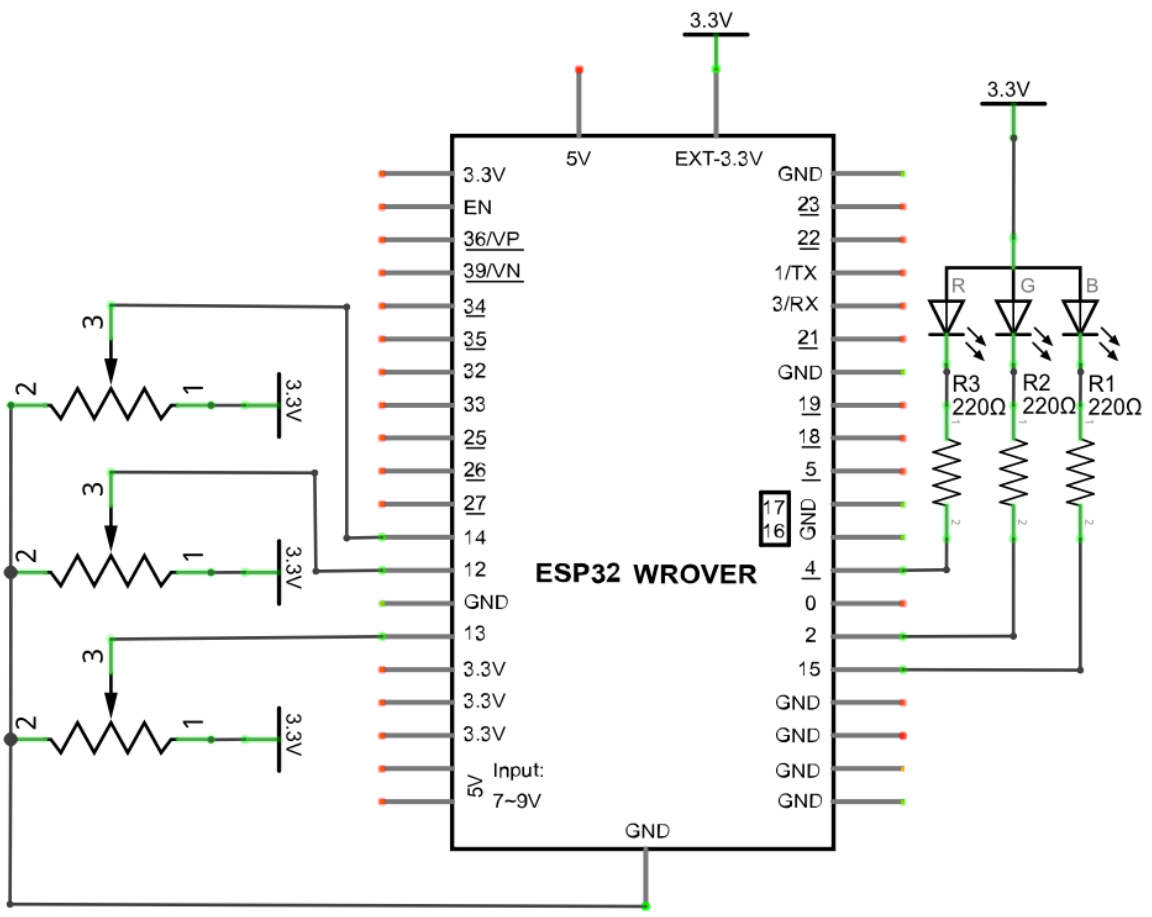
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED,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

Component List

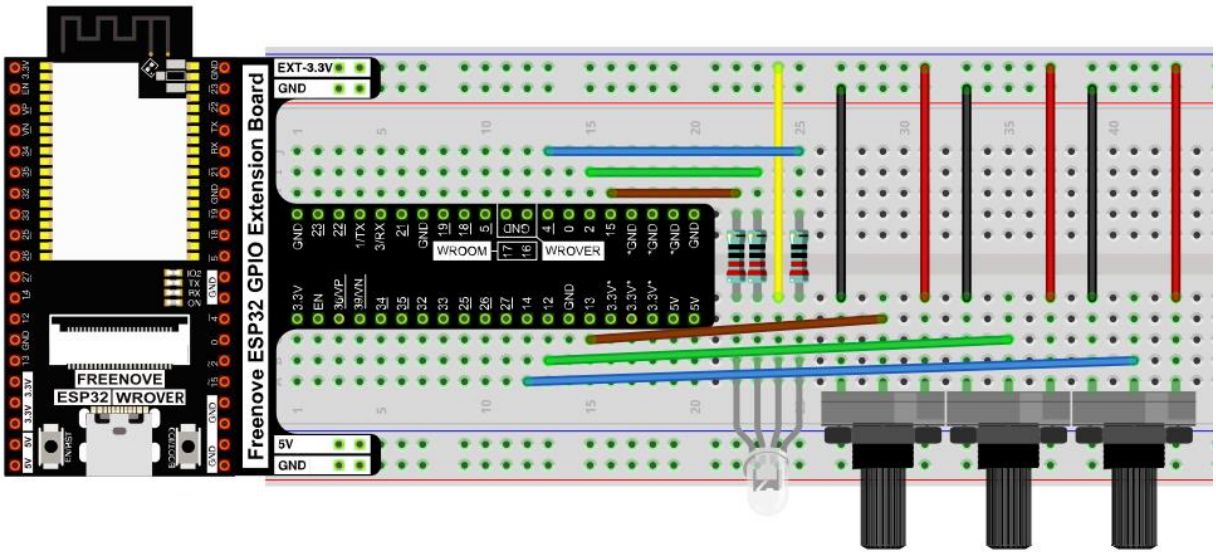
ESP32-WROVER x1 	GPIO Extension Board x1 		
Breadboard x1 			
Rotary potentiometer x3 	Resistor 220Ω x3	RGBLED x1	Jumper M/M x13

Circuit

Schematic diagram

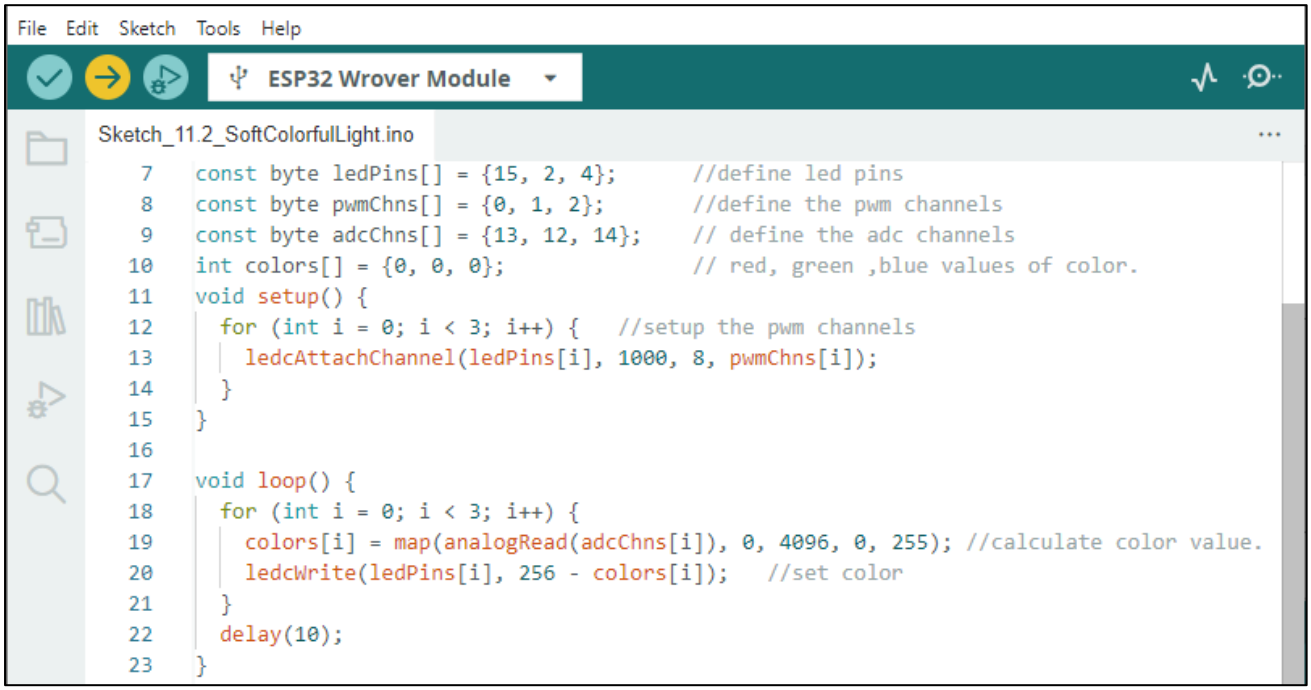


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_11.2_SoftColorfulLight



Download the code to ESP32-WROVER, rotate one of the potentiometers, then the color of RGB LED will change.

The following is the program code:

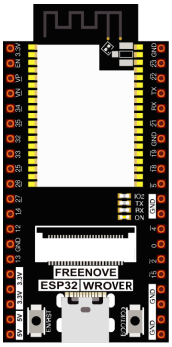
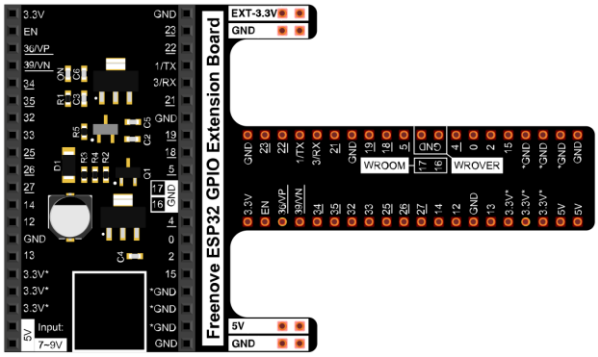
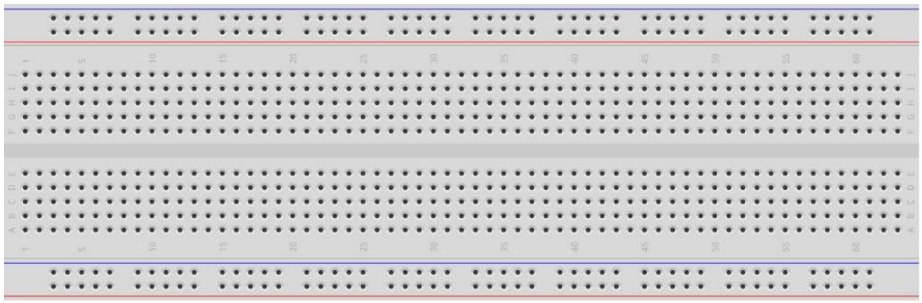
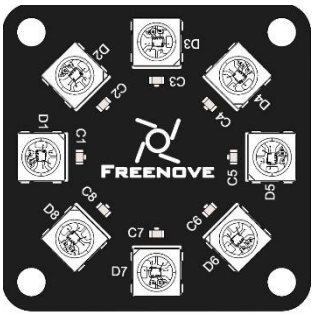
```
const byte ledPins[] = {15, 2, 4};      //define led pins
const byte chns[] = {0, 1, 2};         //define the pwm channels
const byte adcChns[] = {13, 12, 14};    // define the adc channels
int colors[] = {0, 0, 0};               // red, green,blue values of color.
void setup() {
  for (int i = 0; i < 3; i++) {   //setup the pwm channels
    ledcAttachChannal(ledPins[i], 1000, 8, chns[i]); //1KHz, 8bit(0-255).
  }
}

void loop() {
  for (int i = 0; i < 3; i++) {
    colors[i] = map(analogRead(adcChns[i]), 0, 4096, 0, 255);
    ledcWrite(ledPins[i], 256 - colors[i]);   //set color
  }
    delay(10);
}
```

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

Project 11.3 Soft Rainbow Light

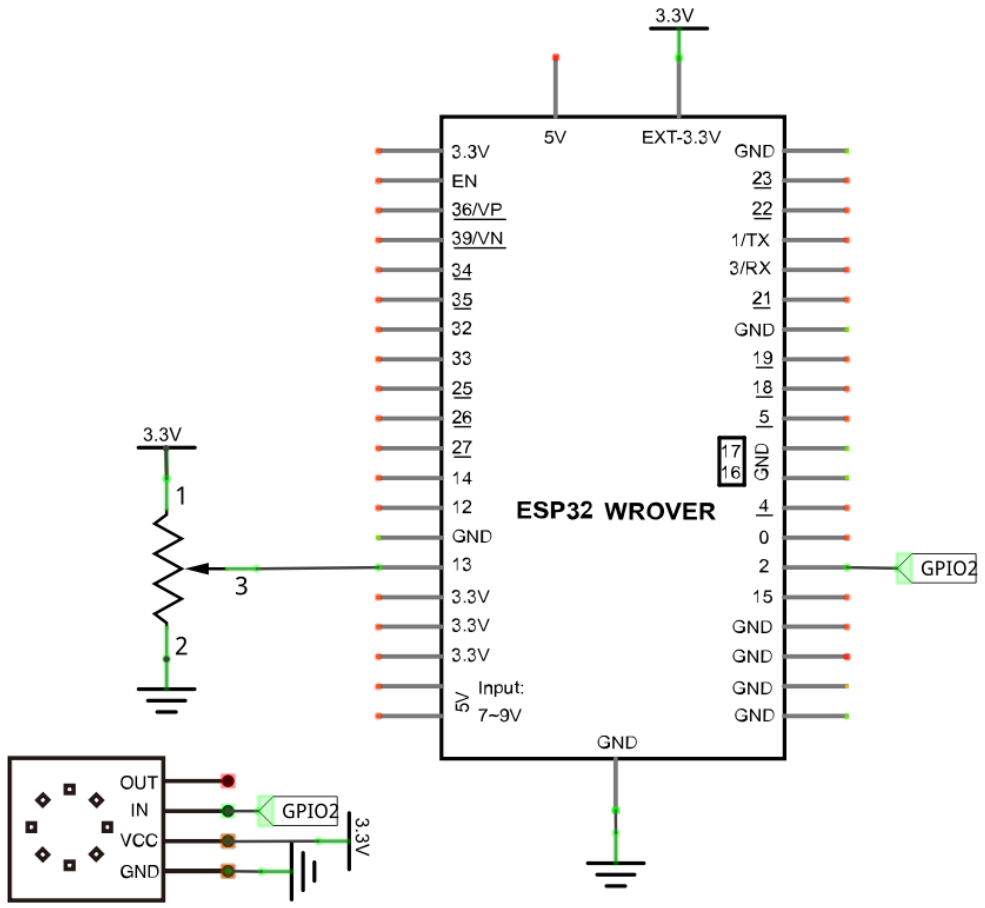
In this project, we use potentiometer to control Freenove 8 RGB LED Module.

Component List

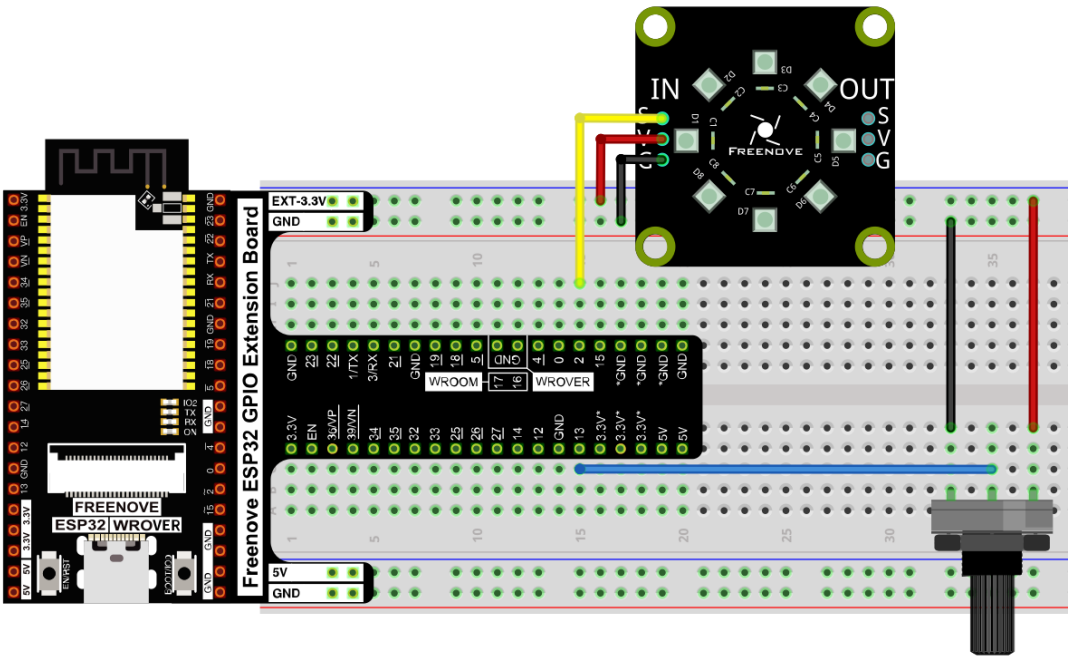
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Freenove 8 RGB LED Module x1 	Jumper F/M x3 Jumper M/M x3

Circuit

Schematic diagram



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



Sketch

Sketch_11.3_Soft_Rainbow_Light

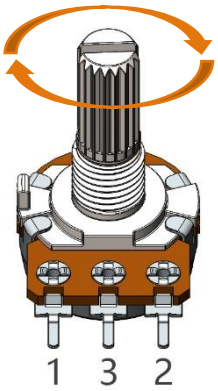
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_11.3_SoftRainbowLight.ino

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define PIN_POT      13 // Define analog input pins
#define LEDS_COUNT   8  // The number of led
#define LEDS_PIN     2  // define the pin connected to the Freenove 8 led strip
#define CHANNEL      0  // RMT module channel
#define BRIGHTNESS   100   // breghtness, the value range is 0-255.
Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

void setup() {
  strip.setBrightness(BRIGHTNESS);
  strip.begin();
}

void loop() {
  int colorPos = map(analogRead(PIN_POT), 0, 4095, 0, 255);
  for (int i = 0; i < LEDS_COUNT; i++) {
    strip.setLedColorData(i,strip.Wheel(colorPos + i * 255 / 8)); // Set color data.
  }
  strip.show(); // Send color data to LED, and display.
  delay(10);
}
```

Download the code to ESP32-WROVER, rotate the handle of the potentiometer, and the color of the lamp ring will change.



The following is the program code:

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define PIN_POT      13    // Define analog input pins
#define LEDS_COUNT   8     // The number of led
#define LEDS_PIN     2     // define the pin connected to the Freenove 8 led strip
#define CHANNEL      0     // RMT module channel
#define BRIGHTNESS   100   // breghtness, the value range is 0-255.
Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

void setup() {
  strip.setBrightness(BRIGHTNESS);
  strip.begin();
}

void loop() {
  int colorPos = map(analogRead(PIN_POT), 0, 4095, 0, 255);
  for (int i = 0; i < LEDS_COUNT; i++) {
    strip.setLedColorData(i,strip.Wheel(colorPos + i * 255 / 8)); // Set color data.
  }
  strip.show(); // Send color data to LED, and display.
  delay(10);
}
```

The overall logical structure of the code is the same as the previous project rainbow light, except that the starting point of the color in this code is controlled by potentiometer.

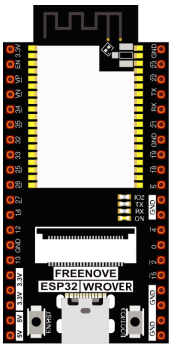
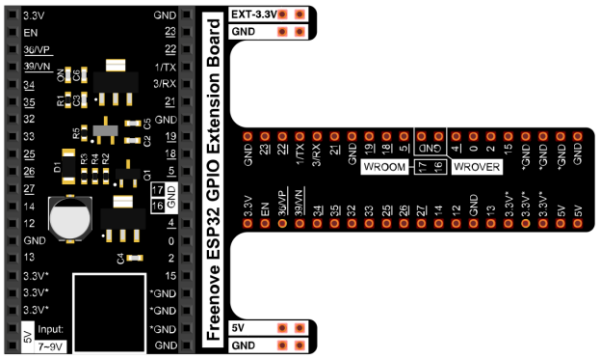
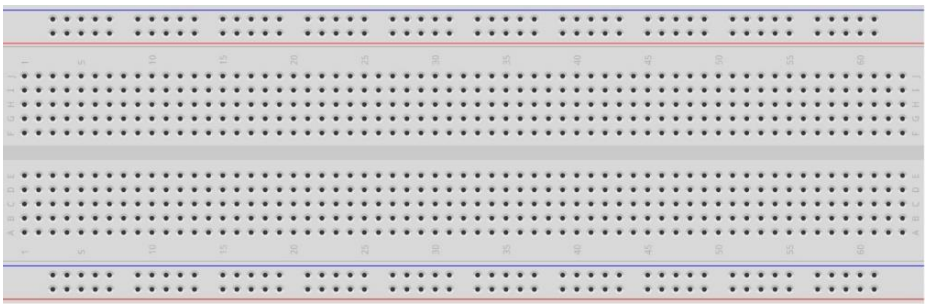
Chapter 12 Photoresistor & LED

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

Project 12.1 NightLamp

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

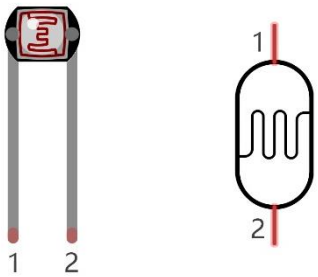
Component List

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

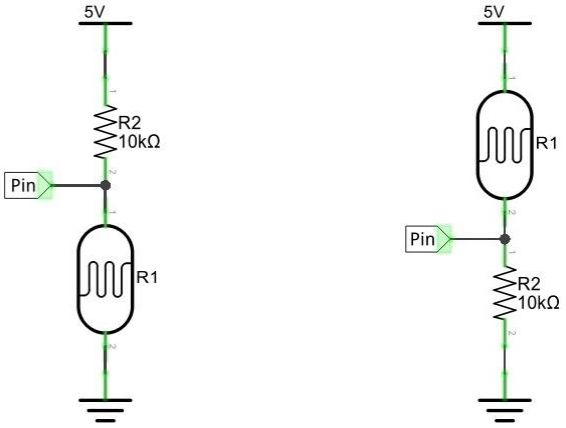
Component knowledge

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:

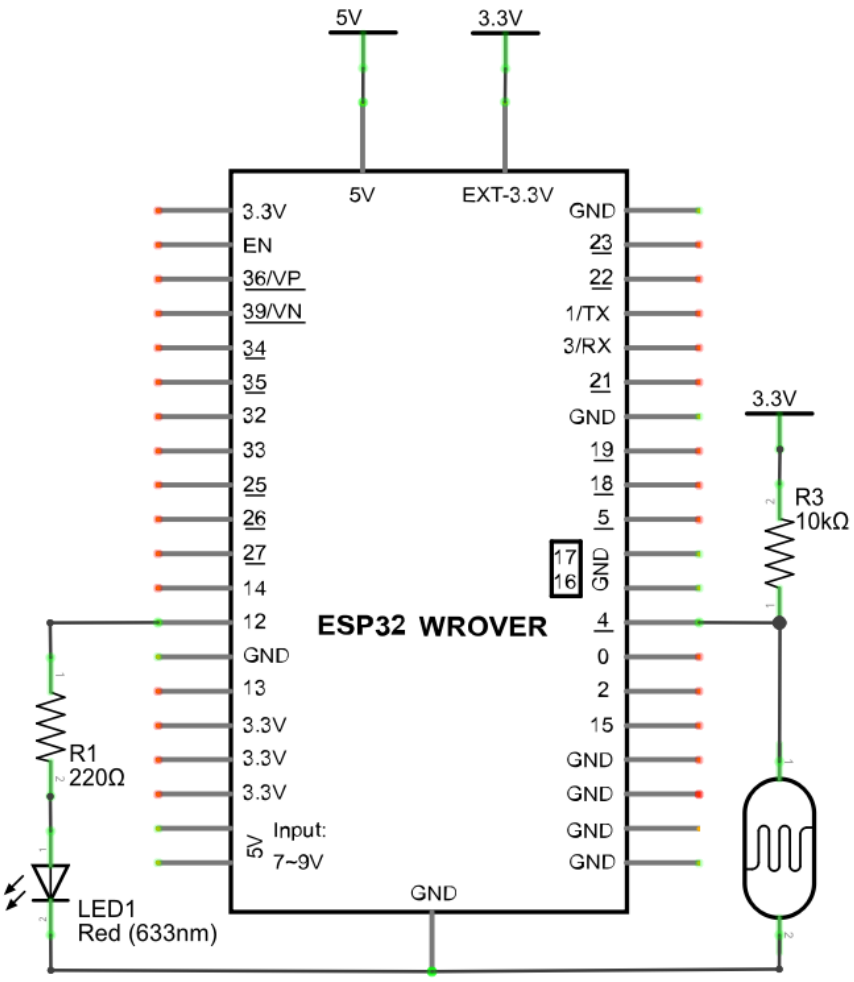


In the above circuit, when a photoresistor's resistance vale changes due to a change in light intensity, the voltage between the photoresistor and resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

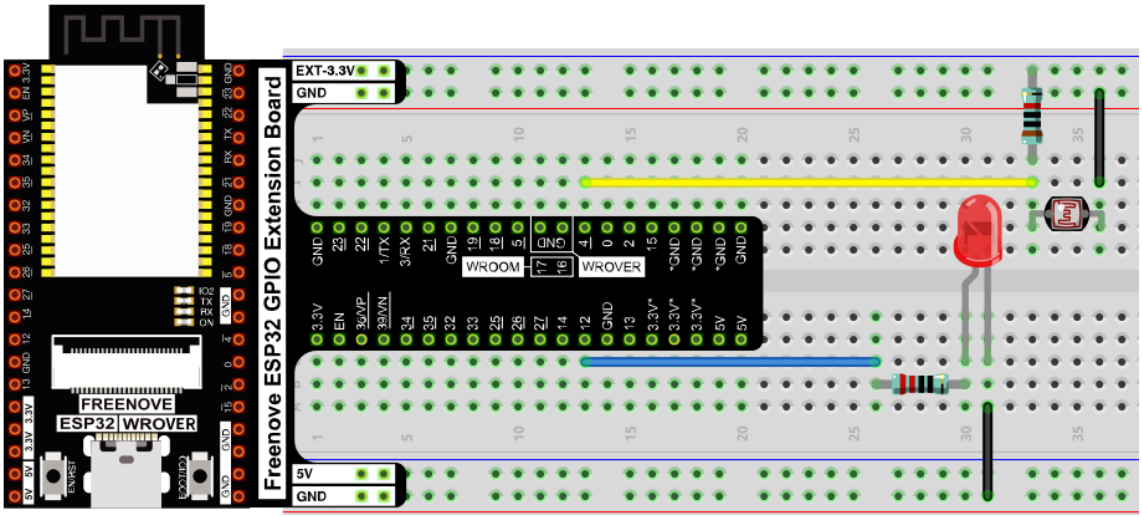
Circuit

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

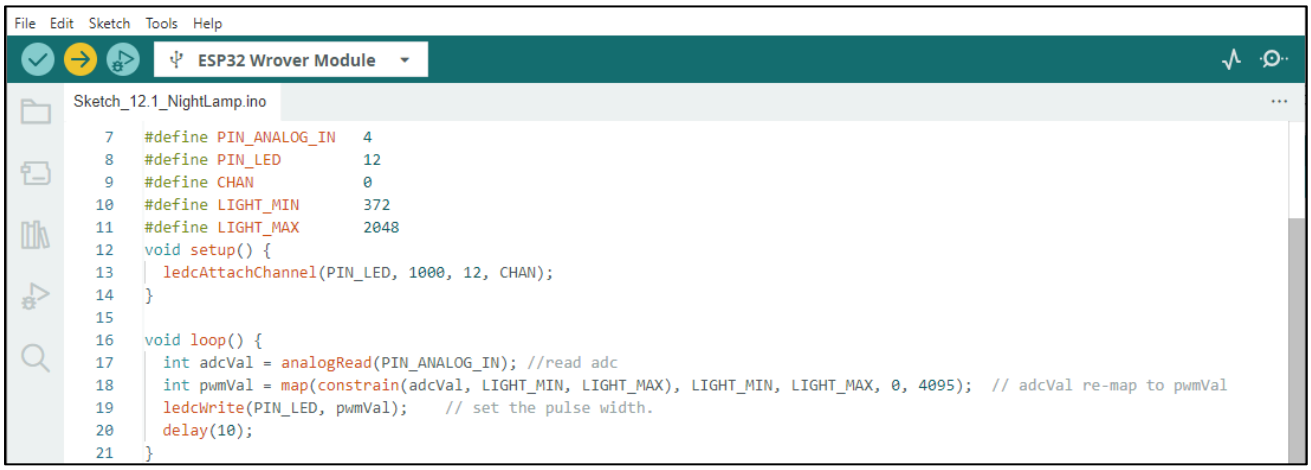


Any concerns? ✉ support@freenove.com

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_12.1_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.

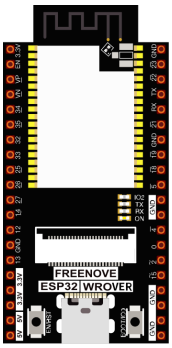
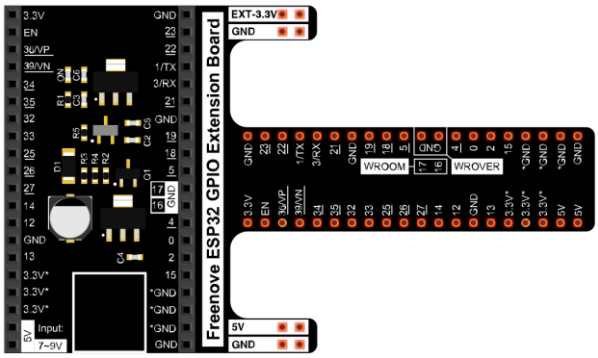
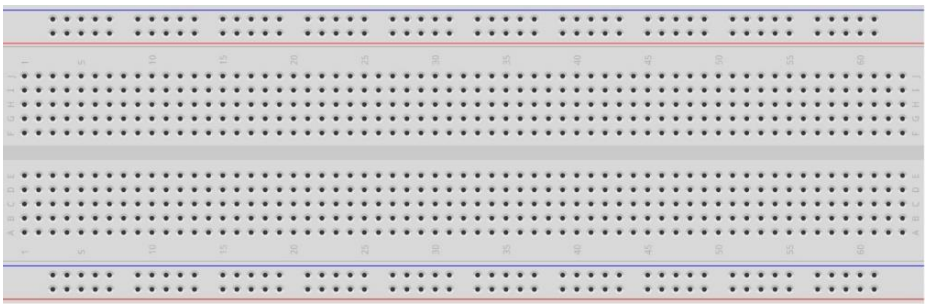
Chapter 13 Thermistor

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

Project 13.1 Thermometer

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

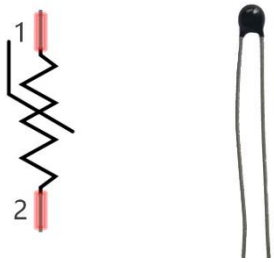
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Thermistor x1	Resistor 10kΩ x1	Jumper M/M x3

Component knowledge

Thermistor

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



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

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

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

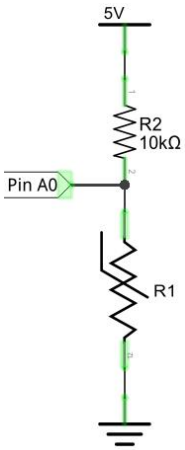
EXP[n] is nth power of E;

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.

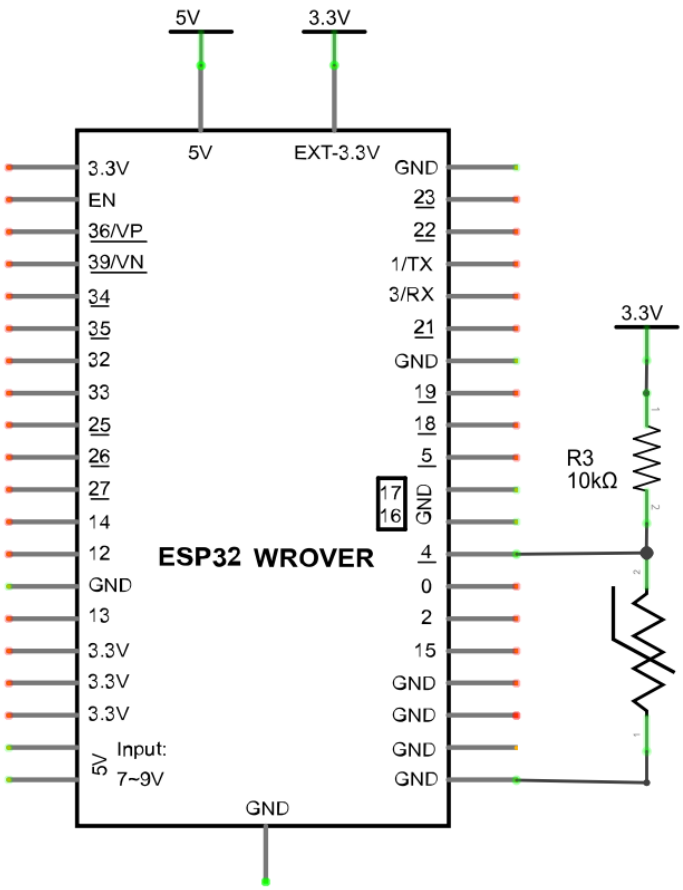
Therefore, the temperature formula can be derived as:

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

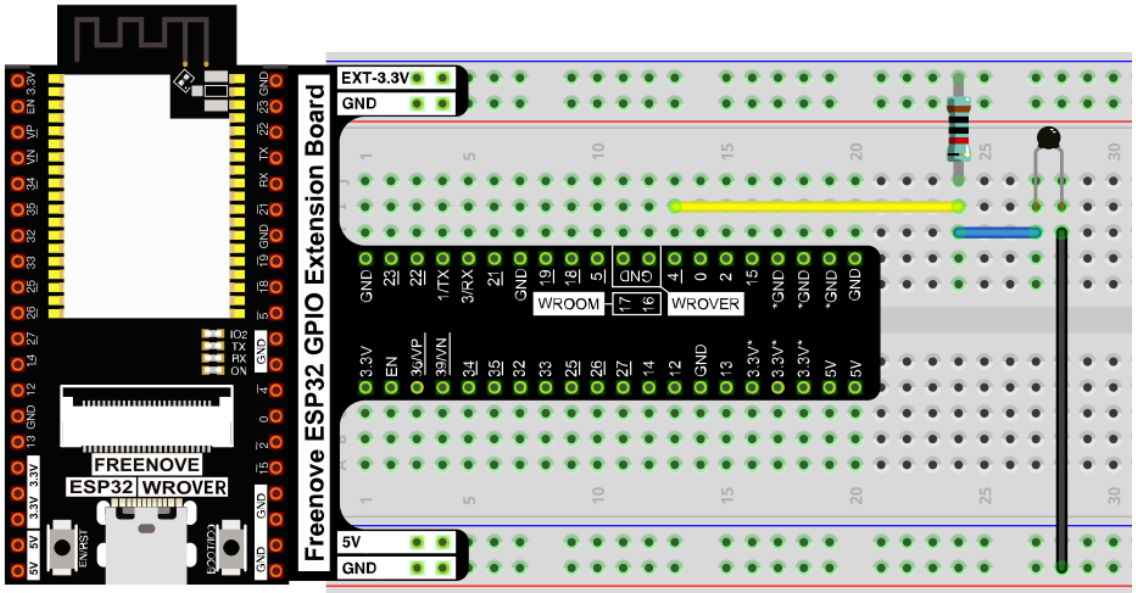
Circuit

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_13.1_Thermometer

```
#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);
}
```

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

```
14:45:16.766 -> ADC value : 1835,       Voltage : 1.48V,        Temperature : 29.76C
14:45:17.769 -> ADC value : 1838,       Voltage : 1.48V,        Temperature : 29.69C
14:45:18.807 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:19.789 -> ADC value : 1834,       Voltage : 1.48V,        Temperature : 29.79C
14:45:20.788 -> ADC value : 1836,       Voltage : 1.48V,        Temperature : 29.74C
14:45:21.769 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:22.793 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:23.784 -> ADC value : 1839,       Voltage : 1.48V,        Temperature : 29.67C
14:45:24.800 -> ADC value : 1839,       Voltage : 1.48V,        Temperature : 29.67C
14:45:25.788 -> ADC value : 1838,       Voltage : 1.48V,        Temperature : 29.69C
14:45:26.769 -> ADC value : 1838,       Voltage : 1.48V,        Temperature : 29.69C
14:45:27.793 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:28.766 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:29.769 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:30.766 -> ADC value : 1839,       Voltage : 1.48V,        Temperature : 29.67C
14:45:31.767 -> ADC value : 1839,       Voltage : 1.48V,        Temperature : 29.67C
14:45:32.794 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
14:45:33.797 -> ADC value : 1840,       Voltage : 1.48V,        Temperature : 29.65C
```

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.

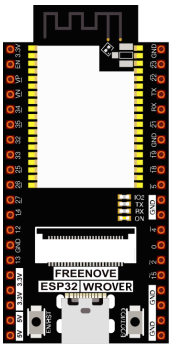
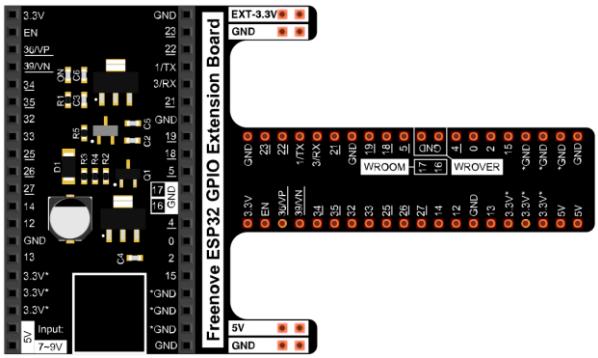
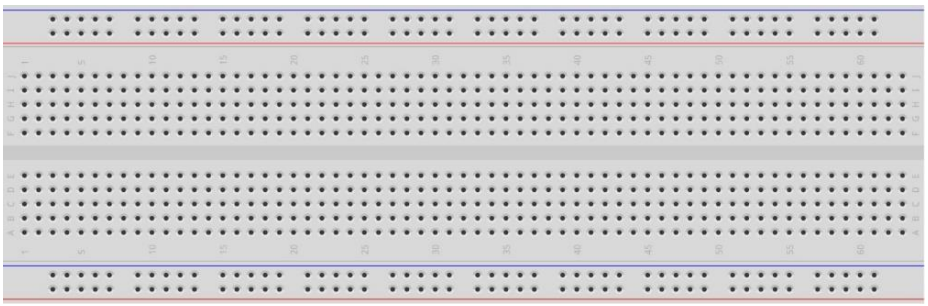
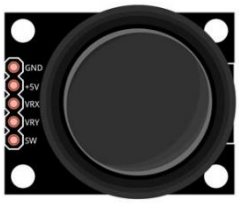
Chapter 14 Joystick

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

Project 14.1 Joystick

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

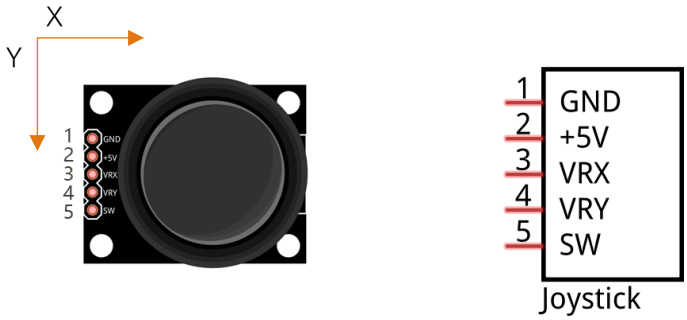
Component List

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

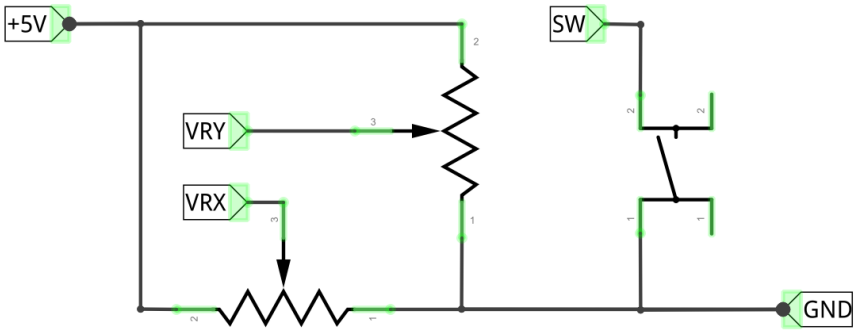
Component knowledge

Joystick

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



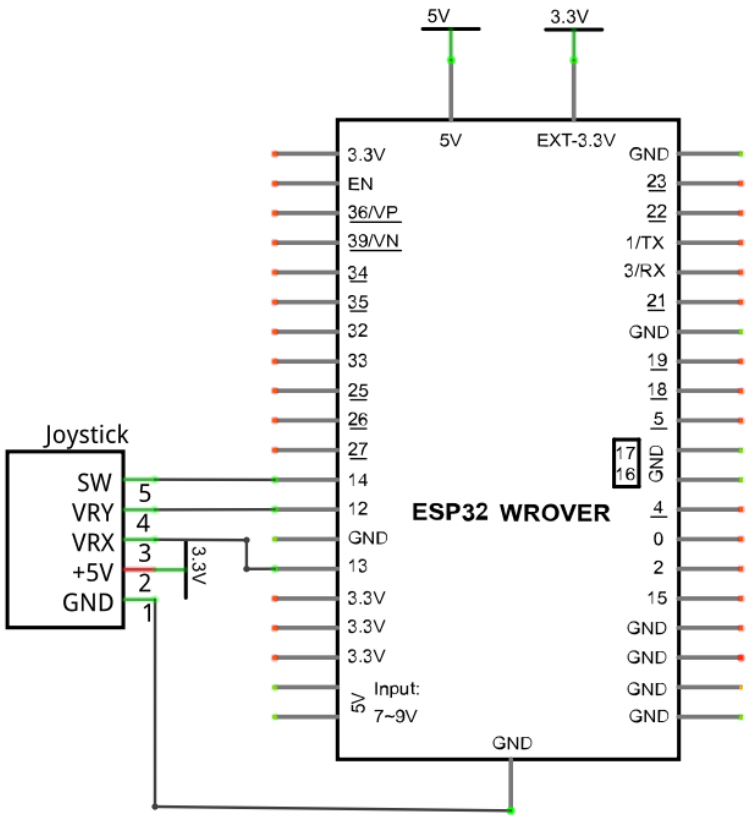
This is accomplished by incorporating two rotary potentiometers inside the joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a push button switch in the "vertical" axis, which can detect when a User presses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



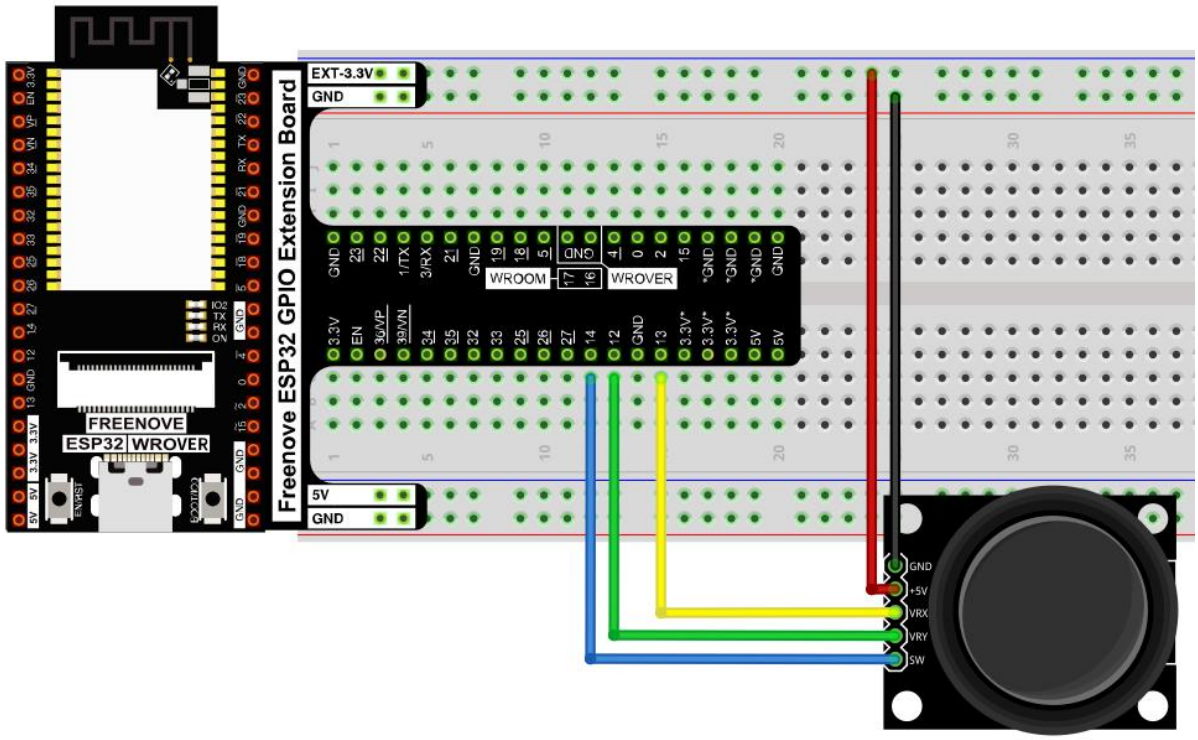
When the joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



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



Sketch

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

Sketch_14.1_Joystick

```
int xyzPins[] = {13, 12, 14};   //x,y,z pins
void setup() {
  Serial.begin(115200);
  pinMode(xyzPins[2], INPUT_PULLUP);  //z axis is a button.
}

void loop() {
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.printf("X,Y,Z: %d,\t%d,\t%d\n", xVal, yVal, zVal);
  delay(500);
}
```

```
Writing at 0x0004bf5d... (90 %)
Writing at 0x000514ea... (100 %)
Wrote 285008 bytes (159968 compressed) at 0x00010000 in 2.6 seconds (effective 863.9 kbit/s)...
Hash of data verified.

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

Download the code to ESP32-WROVER, open the serial port monitor, the baud rate is 115200, as shown in the figure below, shift (moving) the joystick or pressing it down will make the data change.

```
17:40:56.661 -> X,Y,Z: 511,     0,      1
17:40:57.147 -> X,Y,Z: 55,      884,    1
17:40:57.676 -> X,Y,Z: 870,     2,      1
17:40:58.172 -> X,Y,Z: 1853,    278,    1
17:40:58.628 -> X,Y,Z: 464,     0,      0
17:40:59.160 -> X,Y,Z: 498,     0,      0
17:40:59.653 -> X,Y,Z: 495,     0,      0
17:41:00.143 -> X,Y,Z: 548,     0,      0
17:41:00.653 -> X,Y,Z: 578,     878,    1
17:41:01.168 -> X,Y,Z: 688,     0,      1
17:41:01.652 -> X,Y,Z: 675,     0,      1
17:41:02.158 -> X,Y,Z: 751,     16,     1
17:41:02.628 -> X,Y,Z: 766,     35,     1
17:41:03.143 -> X,Y,Z: 816,     67,     1
17:41:03.628 -> X,Y,Z: 789,     92,     1
17:41:04.128 -> X,Y,Z: 854,     135,    1
17:41:04.649 -> X,Y,Z: 894,     163,    1
```

The following is the code:

```
int xyzPins[] = {13, 12, 14};   //x,y,z pins
void setup() {
  Serial.begin(115200);
  pinMode(xyzPins[2], INPUT_PULLUP);  //z axis is a button.
}

void loop() {
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.printf("X,Y,Z: %d,\t%d,\t%d\n", xVal, yVal, zVal);
  delay(500);
}
```

In the code, configure xyzPins[2] to pull-up input mode. In loop(), use analogRead () to read the value of axes X and Y and use digitalRead () to read the value of axis Z, then display them.

```
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.printf("X,Y,Z: %d,\t%d,\t%d\n", xVal, yVal, zVal);
  delay(500);
```

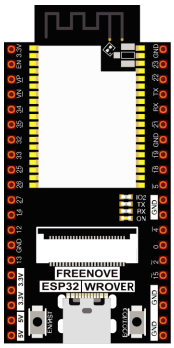
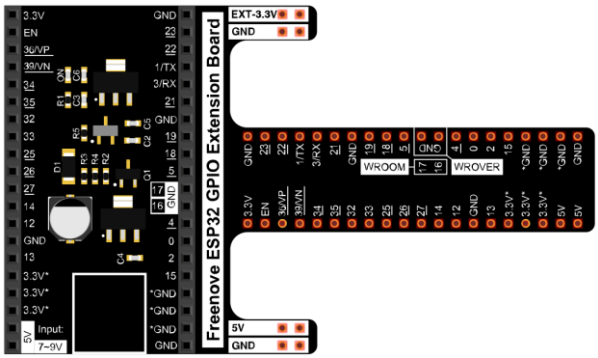
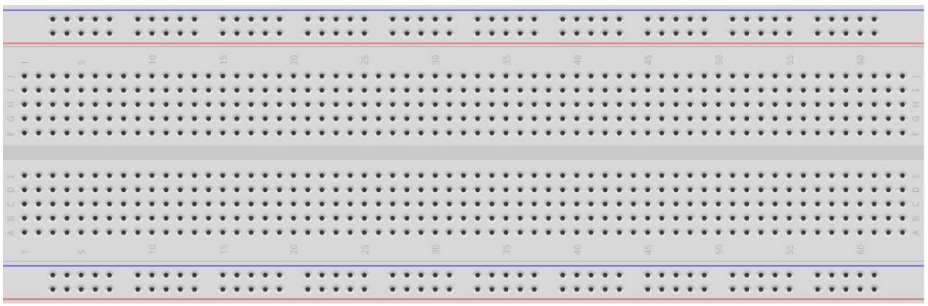
Chapter 15 74HC595 & LED Bar Graph

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

Project 15.1 Flowing Water Light

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

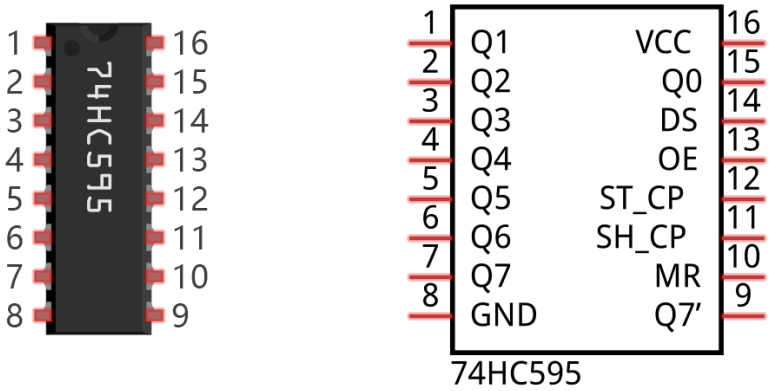
Component List

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

Related knowledge

74HC595

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



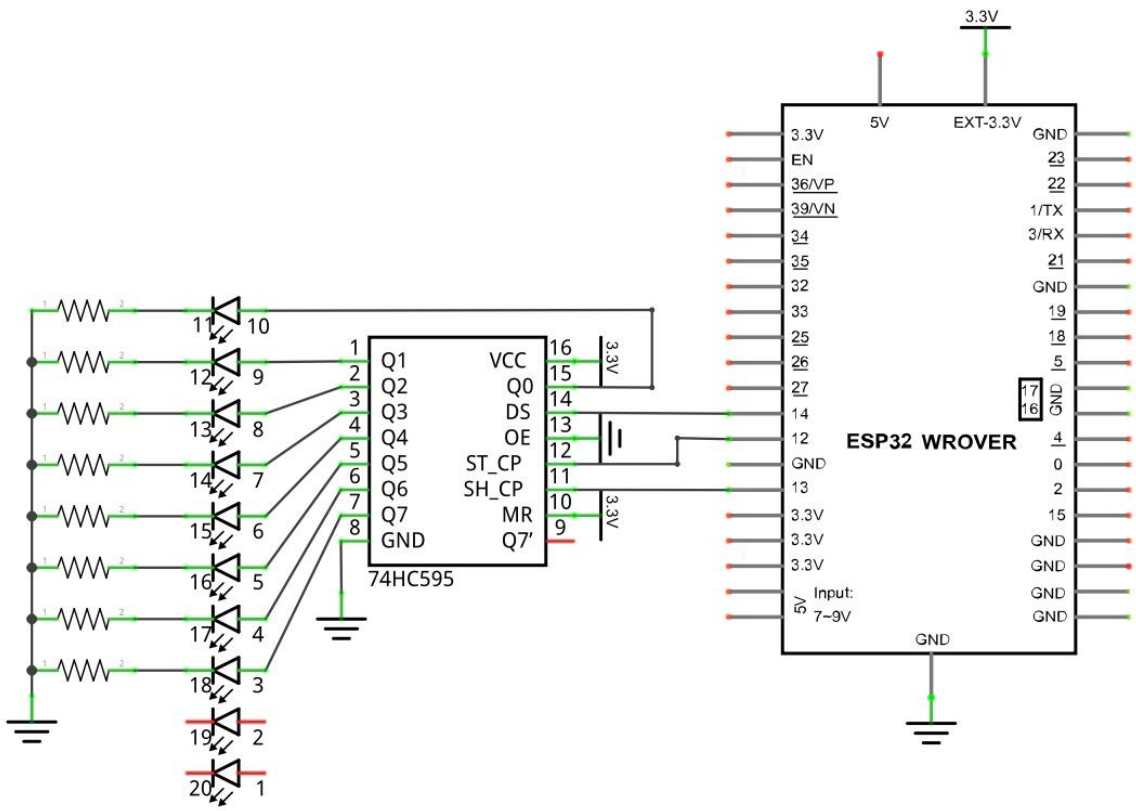
The ports of the 74HC595 chip are described as follows:

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

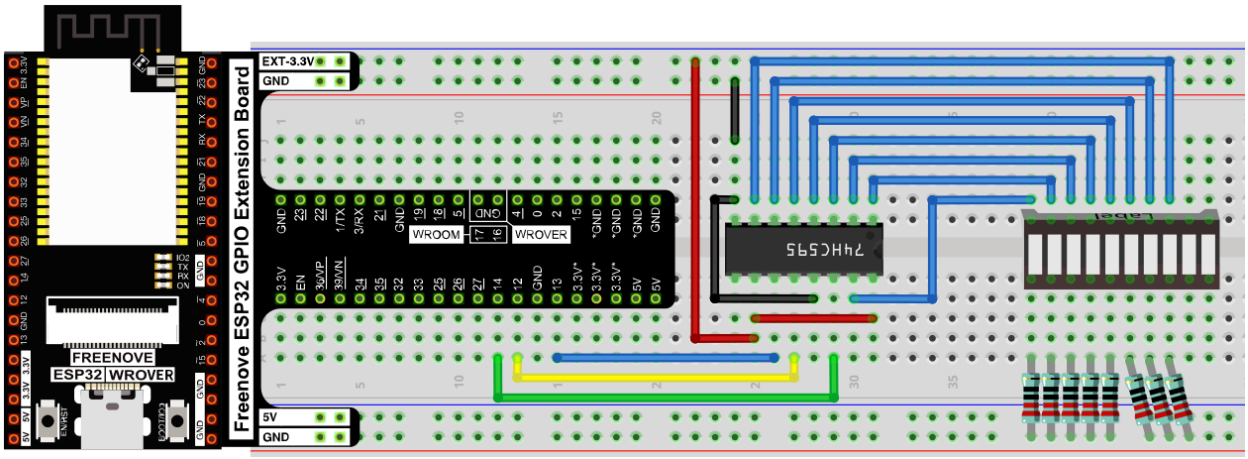
For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



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



Sketch

In this project, we will make a flowing water light with a 74HC595 chip to learn about its functions.

Sketch_15.1_FlowingLight2

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595(Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595(Pin11)
int dataPin = 14;           // Pin connected to DS of 74HC595(Pin14)

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Define a one-byte variable to use the 8 bits to represent the state of 8 LEDs of LED bar graph.
  // This variable is assigned to 0x01, that is binary 00000001, which indicates only one LED light on.
  byte x = 0x01;    // 0b 0000 0001
  for (int j = 0; j < 8; j++) { // Let led light up from right to left
    writeTo595(LSBFIRST, x);
    x <<= 1; // make the variable move one bit to left once, then the bright LED move one step to the left once.
    delay(50);
  }
  delay(100);
  x = 0x80;       //0b 1000 0000
  for (int j = 0; j < 8; j++) { // Let led light up from left to right
    writeTo595(LSBFIRST, x);
    x >>= 1;
    delay(50);
  }
  delay(100);
}
void writeTo595(int order, byte _data ) {
  // Output low level to latchPin
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, order, _data);
  // Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
  digitalWrite(latchPin, HIGH);
}
```

Download the code to ESP32-WROVER. You will see that LED bar graph starts with the flowing water pattern flashing from left to right and then back from right to left.

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

The following is the program code:

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595(Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595(Pin11)
int dataPin = 14;           // Pin connected to DS of 74HC595(Pin14)

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
```

```

void loop() {
  // Define a one-byte variable to use the 8 bits to represent the state of 8 LEDs of LED bar graph.
  // This variable is assigned to 0x01, that is binary 00000001, which indicates only one LED light on.
  byte x = 0x01;    // 0b 0000 0001
  for (int j = 0; j < 8; j++) { // Let led light up from right to left
    writeTo595(LSBFIRST, x);
    x <<= 1; // make the variable move one bit to left once, then the bright LED move one step to the left once.
    delay(50);
  }
  delay(1000);
  x = 0x80;       //0b 1000 0000
  for (int j = 0; j < 8; j++) { // Let led light up from left to right
    writeTo595(LSBFIRST, x);
    x >>= 1;
    delay(50);
  }
  delay(1000);
}
void writeTo595(int order, byte _data ) {
  // Output low level to latchPin
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, order, _data);
  // Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
  digitalWrite(latchPin, HIGH);
}
```

In the code, we configure three pins to control the 74HC595 chip and define a one-byte variable to control the state of the 8 LEDs (in the LED bar graph Module) through the 8 bits of the variable. The LEDs light ON when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED ON.

```
x=0x01;
```

In the loop(), use “for” loop to send x to 74HC595 output pin to control the LED. In "for" loop, x will shift one bit to the LEFT in one cycle, then when data of x is sent to 74HC595, the LED that is turned ON will move one bit to the LEFT once.

```
  for (int j = 0; j < 8; j++) { // Let led light up from right to left
    writeTo595(LSBFIRST, x);
    x <<= 1;
    delay(50);
  }
```

In second “for” loop, the situation is the same. The difference is that x is shift from 0x80 to the RIGHT in order. The subfunction writeTo595() is used to write data to 74HC595 and immediately output on the port of 74HC595.

Reference

<< operator

"<<" is the left shift operator, which can make all bits of 1 byte shift by several bits to the left (high) direction and add 0 on the right (low). For example, shift binary 00000001 by 1 bit to left:

```
byte x = 1 << 1;
```

← ← ← ← ← ← ←

← | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ← | 0 |

The result of x is 2（binary 00000010）.

0	0	0	0	0	0	1	0

There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

```
byte x = 1 >> 1;
```

→ → → → → → →

0	→	0	0	0	0	0	0	0	1	→

The result of x is 0（00000000）.

0	0	0	0	0	0	0	0

X <<= 1 is equivalent to x = x << 1 and x >>= 1 is equivalent to x = x >> 1

```
void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val);
```

This is used to shift an 8-bit data value in with the data appearing on the dataPin and the clock being sent out on the clockPin. Order is as above. The data is sampled after the cPin goes high. (So clockPin high, sample data, clockPin low, repeat for 8 bits) The 8-bit value is returned by the function.

Parameters

dataPin: the pin on which to output each bit. Allowed data types: int.

clockPin: the pin to toggle once the dataPin has been set to the correct value. Allowed data types: int.

bitOrder: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First).

value: the data to shift out. Allowed data types: byte.

For more details about shift function, please refer to:
https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/

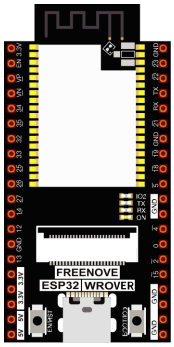
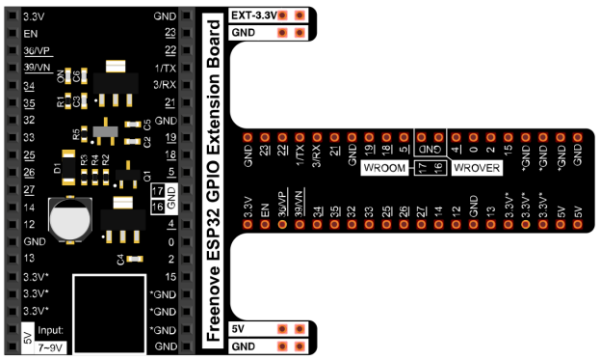
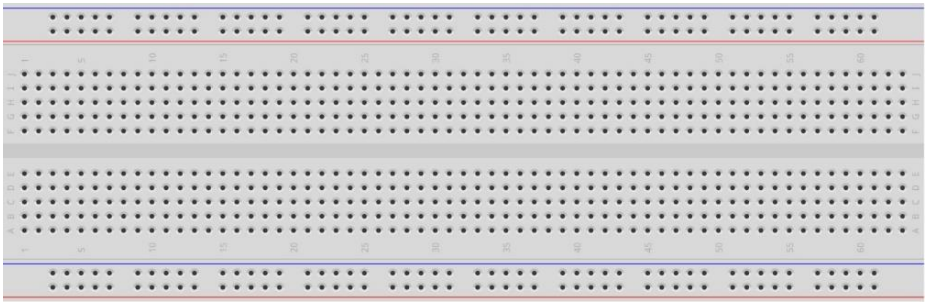
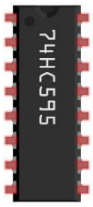
Chapter 16 74HC595 & 7-Segment Display.

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

Project 16.1 7-Segment Display.

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

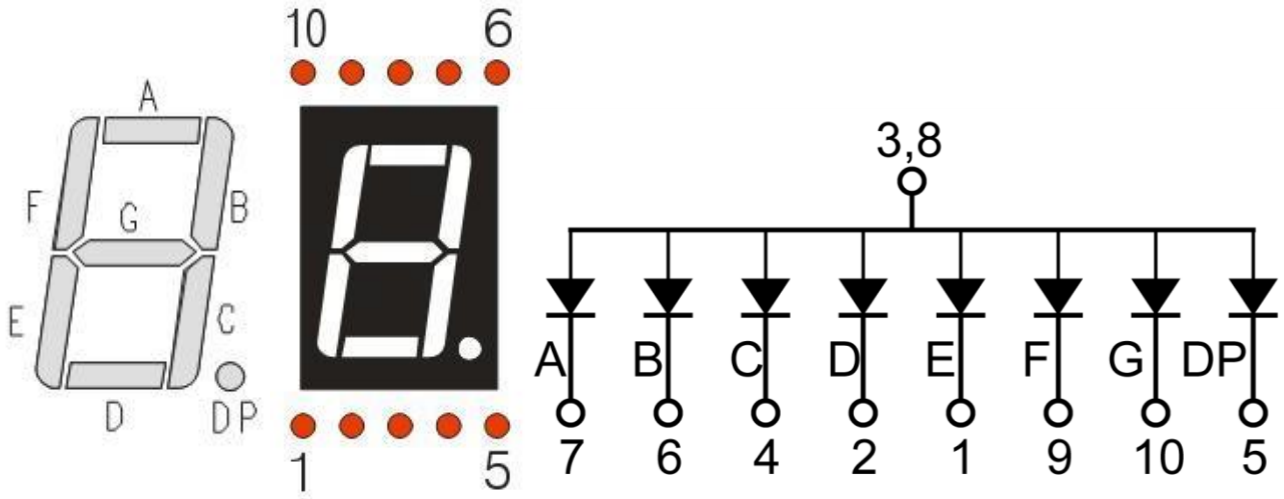
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 		
Breadboard x1 			
74HC595 x1 	7-segment display x1	Resistor 220Ω x8	Jumper M/M

Component knowledge

7-segment display

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



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

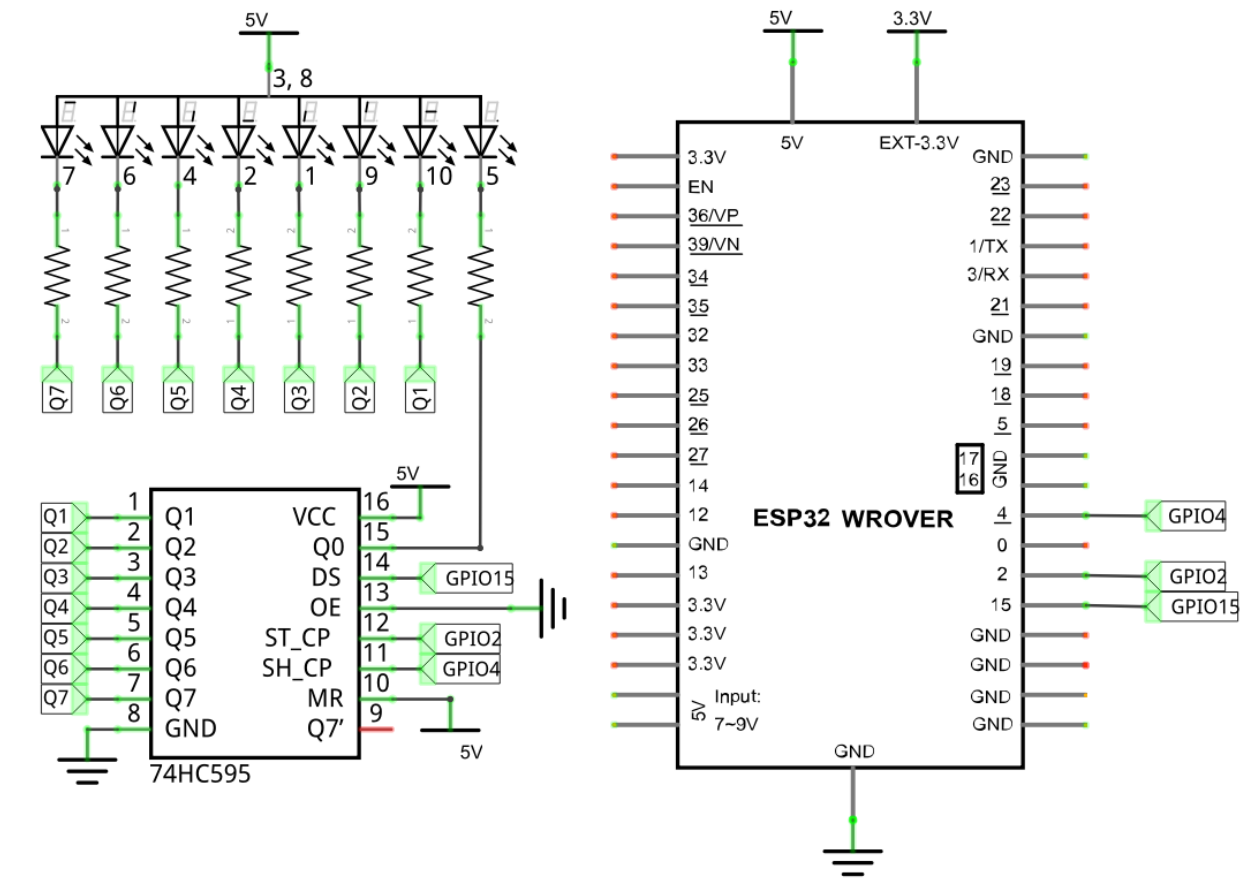
In this project, we will use a 7-Segment Display with a common anode. Therefore, when there is an input low level to a LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”,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

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

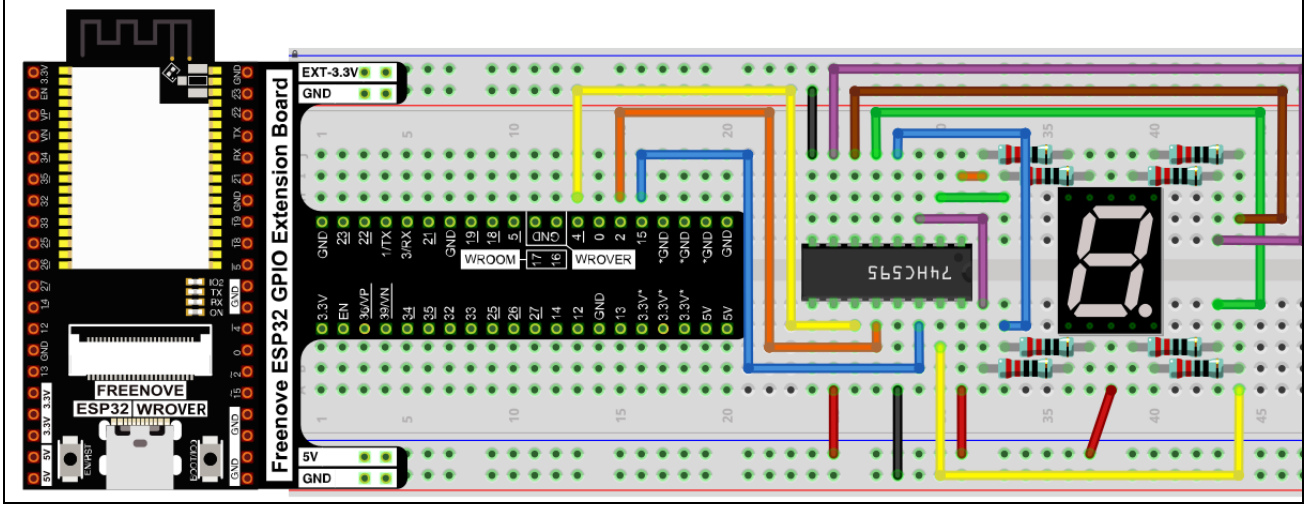
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0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

Circuit

Schematic diagram



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



Sketch

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

Sketch_16.1_7_Segment_Display

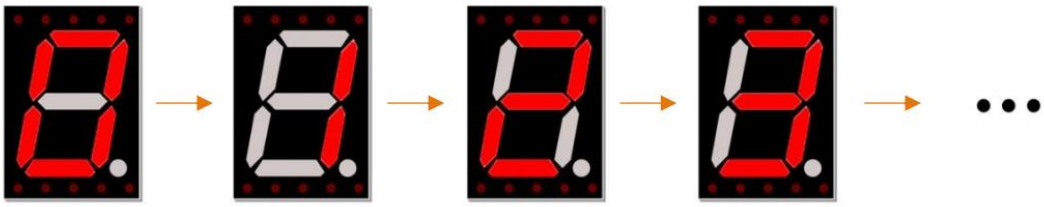
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_16.1_1_Digit_7-Segment_Display.ino

```
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)

// Define the encoding of characters 0-F for the common-anode 7-Segment Display
byte num[] = {
  0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
  0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // display 0-F on digital tube
  for (int i = 0; i < 16; i++) {
    writeData(num[i]);// Send data to 74HC595
    delay(1000);      // delay 1 second
    writeData(0xff);  // Clear the display content
  }
}

void writeData(int value) {
  // Make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  // Make latchPin output high level, then 74HC595 will update the data to parallel output
  digitalWrite(latchPin, HIGH);
}
```

Verify and upload the code, and you'll see a 1-bit, 7-segment display displaying 0-f in a loop.



The following is the program code:

```
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
// Define the encoding of characters 0-F for the common-anode 7-Segment Display
byte num[] = {
  0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
  0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // display 0-F on digital tube
  for (int i = 0; i < 16; i++) {
    writeData(num[i]);// Send data to 74HC595
    delay(1000);      // delay 1 second
    writeData(0xff);  // Clear the display content
  }
}

void writeData(int value) {
  // Make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  // Make latchPin output high level
  digitalWrite(latchPin, HIGH);
}
```

First, put encoding of “0”- “F” into the array.

```
// Define the encoding of characters 0-F for the common-anode 7-Segment Display
byte num[] = {
  0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
  0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
};
```

Then, in the loop, we transfer the member of the “num” to 74HC595 by calling the writeData function, so that the digital tube displays what we want. After each display, “0xff” is used to eliminate the previous effect and prepare for the next display.

```
void loop() {
  // display 0-F on digital tube
  for (int i = 0; i < 16; i++) {
    writeData(num[i]);// Send data to 74HC595
    delay(1000);      // delay 1 second
    writeData(0xff);  // Clear the display content
  }
}
```

In the shiftOut() function, whether to use LSBFIRST or MSBFIRST as the parameter depends on the physical situation.

```
void writeData(int value) {
  // Make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  // Make latchPin output high level, then 74HC595 will update data to parallel output
  digitalWrite(latchPin, HIGH);
}
```

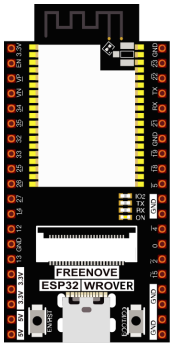
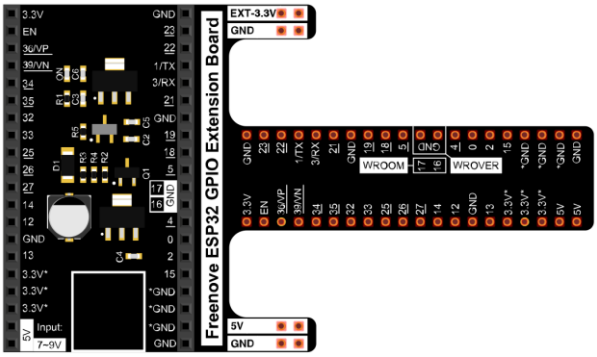
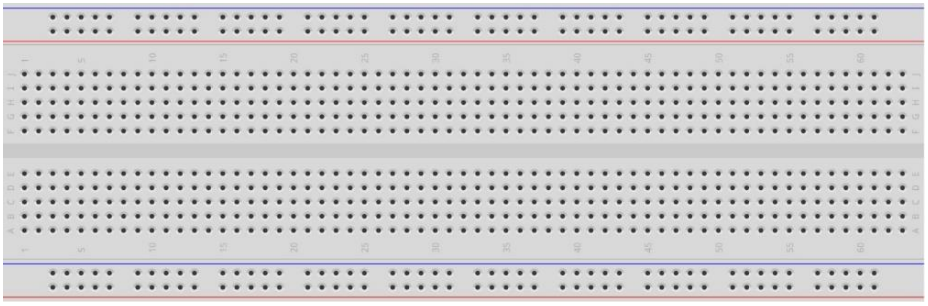
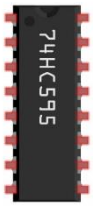
If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

```
shiftOut(dataPin,clockPin, LSBFIRST, value & 0x7f);
```

Project 16.2 4-Digit 7-Segment Display

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

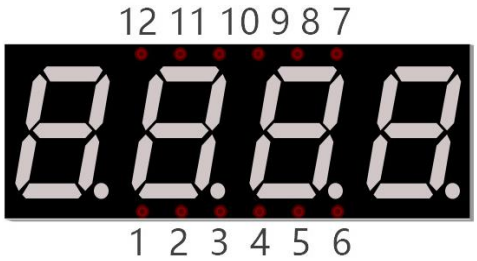
Component List

ESP32-WROVER x1 		GPIO Extension Board x1 	
Breadboard x1 			
74HC595 x1 	7-segment display x1	Resistor 220Ω x8	Jumper M/M

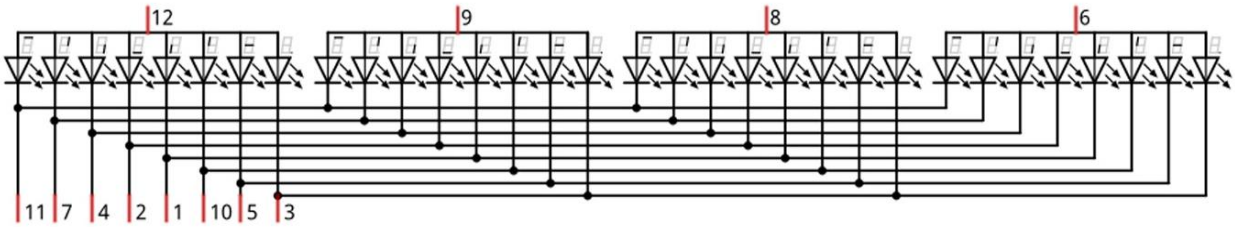
Component knowledge

4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-segment displays into one module, therefore it can display more characters. All of the LEDs contained have a common anode and individual cathodes. Its internal structure and pin designation diagram is shown below:



The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-segment display are connected together.

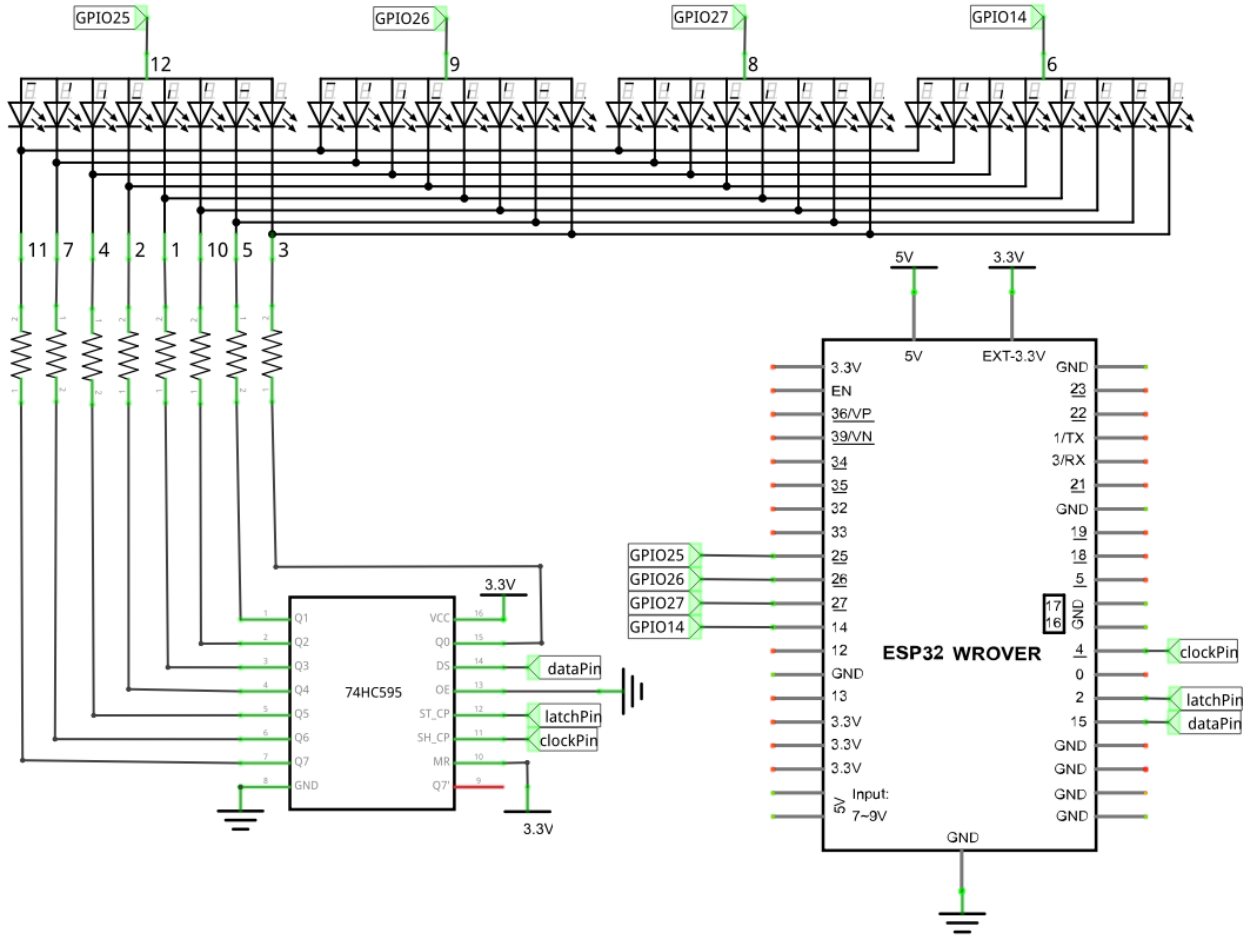


Display method of 4 digit 7-segment display is similar to 1 digit 7-segment display. The difference between them is that the 4-digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first digit display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-segment display will show visible content and the remaining three will be OFF.

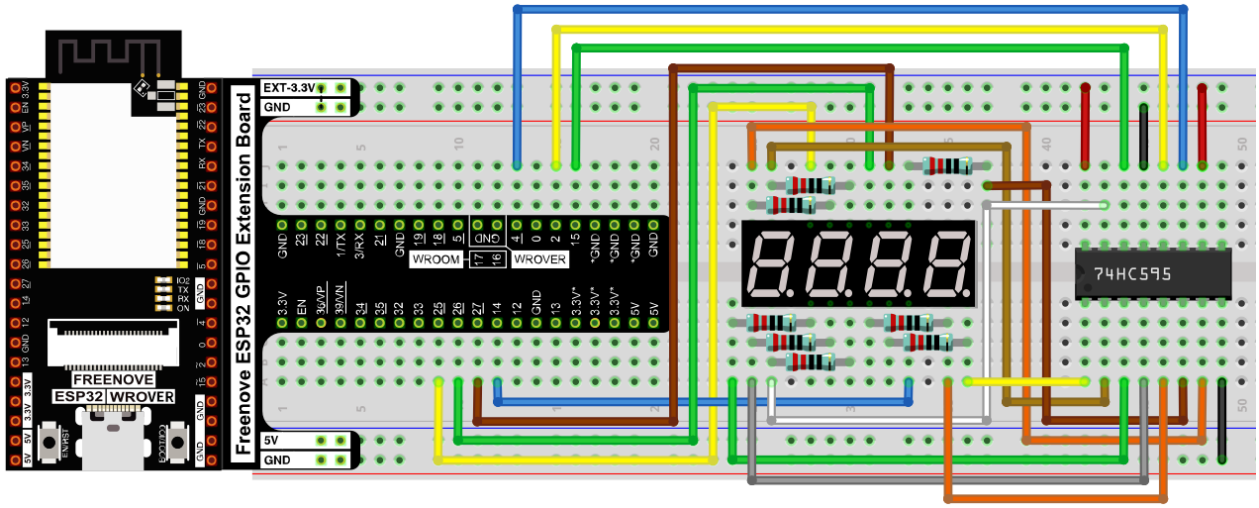
Similarly, the second, third and fourth 7-segment displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is imperceptible to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

Circuit

Schematic diagram



Hardware connection:



Sketch

In this code, we use the 74HC595 IC chip to control the 4-digit 7-segment display, and use the dynamic scanning method to show the changing number characters.

Sketch_16.2_4_Dight_7-Segment_Display

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_16.2_4_Digit_7-Segment_Display.ino

```
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {25,26,27,14};// Common pin (anode) of 4 digit 7-segment display

// Define the encoding of characters 0-F of the common-anode 7-Segment Display
byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
              0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (int i = 0; i < 4; i++) {
    pinMode(comPin[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 4; i++) {
    // Select a single 7-segment display
    selectDigitalDisplay (i);
    // Send data to 74HC595
    writeData(num[i]);
    delay(5);
    // Clear the display content
    writeData(0xff);
  }
}
```

Compile and upload code to ESP32-WROVER, then the digital tube displays as shown.

The following is the program code:

```
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {25,26,27,14};// Common pin (anode) of 4 digit 7-segment display

// Define the encoding of characters 0-F of the common-anode 7-Segment Display
byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
              0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (int i = 0; i < 4; i++) {
    pinMode(comPin[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 4; i++) {
    // Select a single 7-segment display
    selectDigitalDisplay (i);
    // Send data to 74HC595
    writeData(num[i]);
    delay(5);
    // Clear the display content
    writeData(0xff);
  }
}

void selectDigitalDisplay(byte com) {
  // Close all single 7-segment display
  for (int i = 0; i < 4; i++) {
    digitalWrite(comPin[i], LOW);
  }
```

```
  // Open the selected single 7-segment display
  digitalWrite(comPin[com], HIGH);
}

void writeData(int value) {
  // Make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);  // Make latchPin output high level
  // Make latchPin output high level, then 74HC595 will update data to parallel output
  digitalWrite(latchPin, HIGH);
}
```

First, define the pin of 74HC595 and 7-segment display common end, character encoding.

```
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {25,26,27,14};// Common pin (anode) of 4 digit 7-segment display

// Define the encoding of characters 0-F of the common-anode 7-Segment Display
byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
              0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};
```

Second, initialize all the pins to output mode.

```
void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (int i = 0; i < 4; i++) {
    pinMode(comPin[i], OUTPUT);
  }
}
```

Then, since there are four digital tubes, we need to write a subfunction to control it to turn ON any digital tube. In order not to affect a new display, each time we want to turn ON a digital tube, we need to set the other digital tube OFF.

```
void selectDigitalDisplay(byte com) {
  // Close all single 7-segment display
  for (int i = 0; i < 4; i++) {
    digitalWrite(comPin[i], LOW);
  }
  // Open the selected single 7-segment display
  digitalWrite(comPin[com], HIGH);
}
```

The usage of the writeData function is the same as in the previous two sections, so it won't be covered again here.

```
void writeData(int value) {
  // Make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  // Make latchPin output high level, then 74HC595 will update data to parallel output
  digitalWrite(latchPin, HIGH);
}
```

In the loop function, because there are four digital tubes, a “for loop” is used to display the values of each one in turn. For example, when i =0, turn ON the first digital tube to display the first value, then turn ON the second digital tube to display the second value, until all four digital tubes display their own values. Because the displaying time from the first number to the fourth number is so short, it may display many times in one second, but our eyes can't keep up with the speed of the digital tube, so we look as if the digital tube is displaying different Numbers at the same time.

```
void loop() {
  for (int i = 0; i < 4; i++) {
    // Select a single 7-segment display
    selectDigitalDisplay(i);
    // Send data to 74HC595
    writeData(num[i]);
    delay(5);
    // Clear the display content
    writeData(0xff);
  }
}
```

If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

```
shiftOut(dataPin,clockPin, LSBFIRST, value & 0x7f);
```

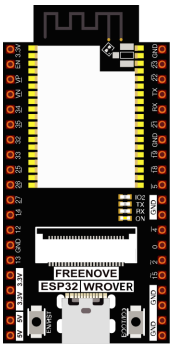
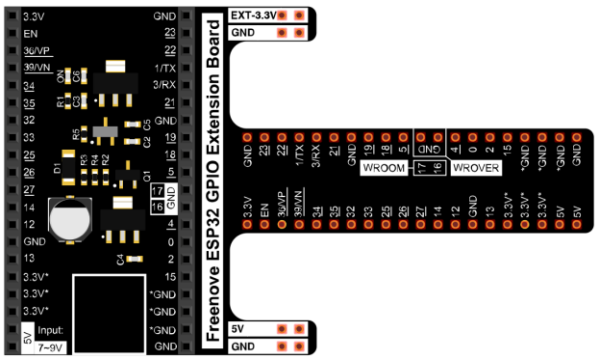
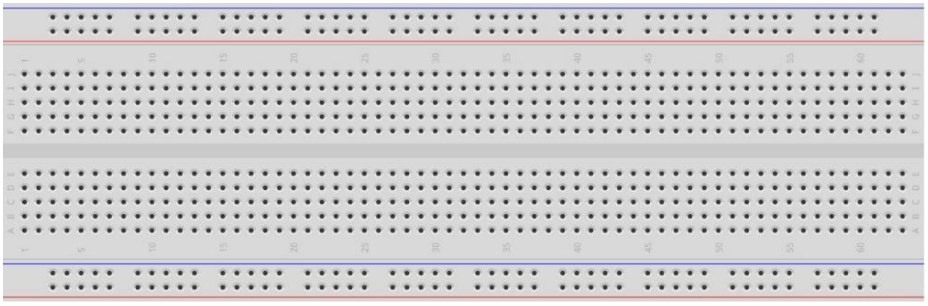
Chapter 16 74HC595 & LED Matrix

Thus far we have learned how to use the 74HC595 IC chip to control the LED bar graph and the 7-segment display. We will now use 74HC595 IC chips to control a LED matrix.

Project 16.3 LED Matrix

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

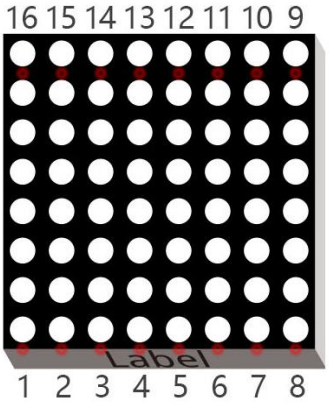
Component List

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

Component knowledge

LED Matrix

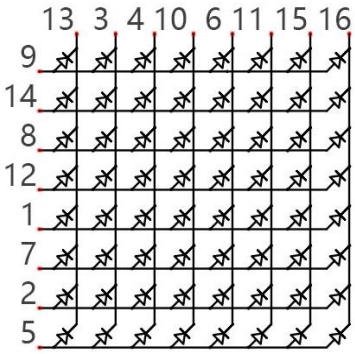
A LED matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED matrix containing 64 LEDs (8 rows by 8 columns).



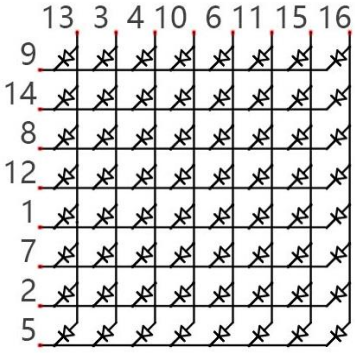
In order to facilitate the operation and reduce the number of ports required to drive this component, the positive poles of the LEDs in each row and negative poles of the LEDs in each column are respectively connected together inside the LED matrix module, which is called a common anode. There is another arrangement type. Negative poles of the LEDs in each row and the positive poles of the LEDs in each column are respectively connected together, which is called a common cathode.

The LED matrix that we use in this project is a common anode LED matrix.

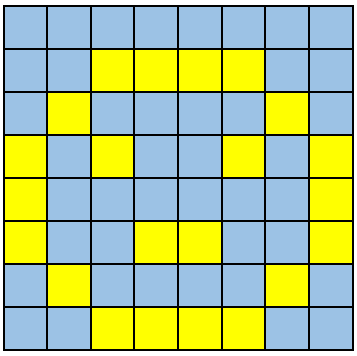
Connection mode of common anode



Connection mode of common cathode



Here is how a common anode LED matrix works. First, choose 16 ports on ESP32 board to connect to the 16 ports of LED matrix. Configure one port in columns for low level, which makes that column the selected port. Then configure the eight port in the row to display content in the selected column. Add a delay value and then select the next column that outputs the corresponding content. This kind of operation by column is called scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.



1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

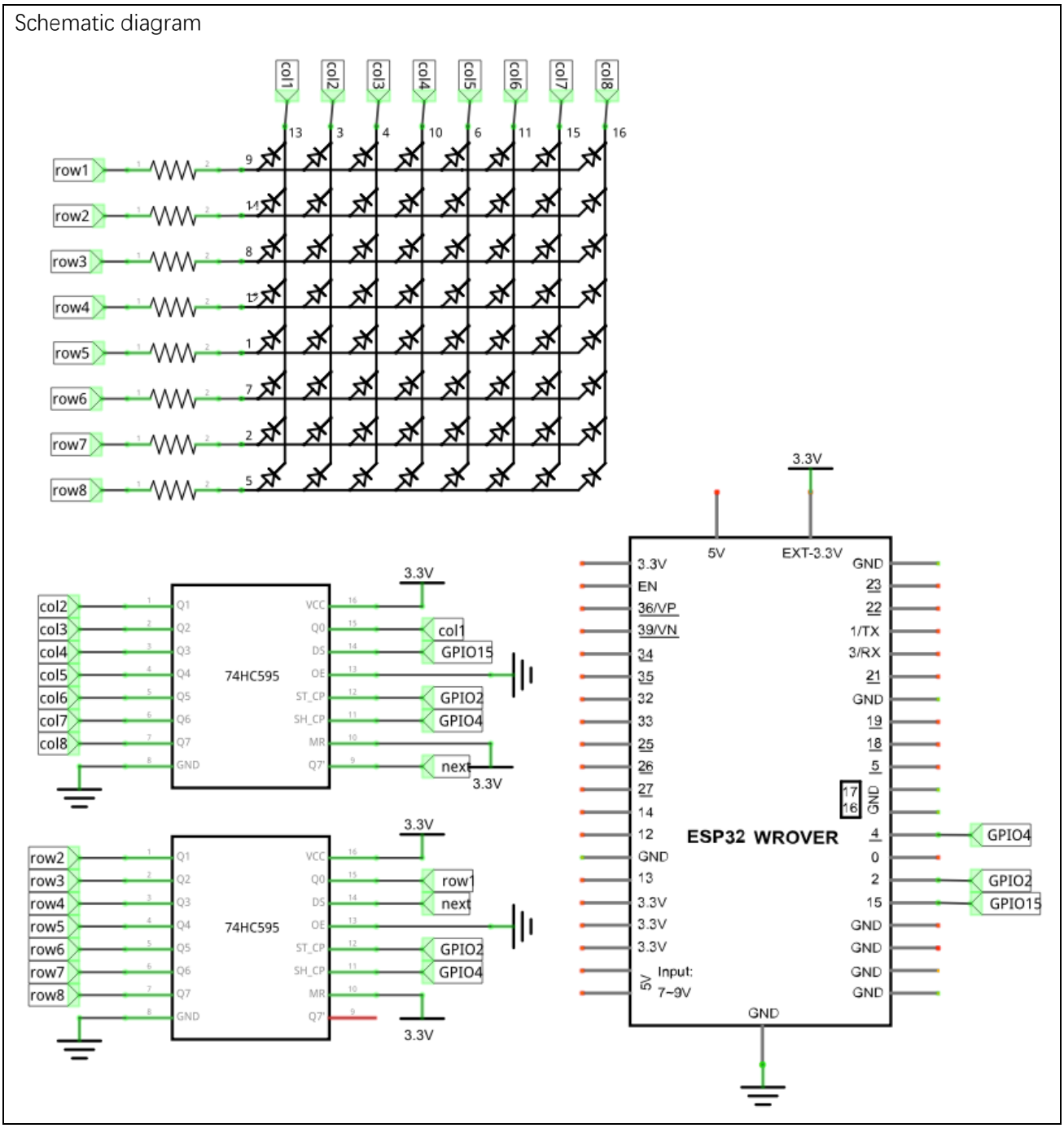
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED bar graph project. The whole process will be repeated rapidly in a loop. Due to the principle of optical afterglow effect and the vision persistence effect in human sight, we will see a picture of a smiling face directly rather than individual columns of LEDs turned ON one column at a time (although in fact this is the reality we cannot perceive).

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

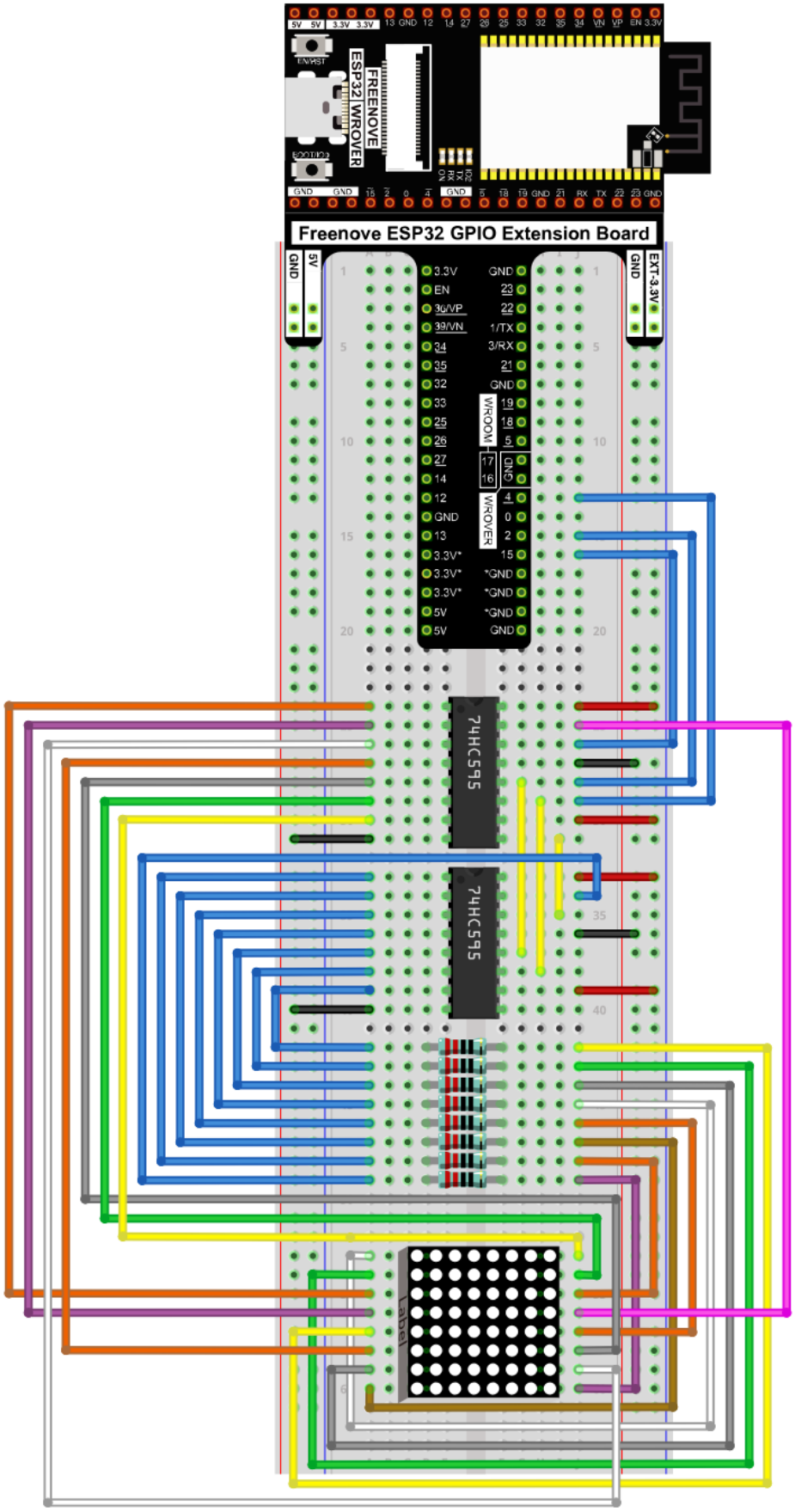
Circuit

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

Schematic diagram



Hardware connection:



Sketch

The following code will make LED matrix display a smiling face, and then display scrolling character "0-F".

```
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)

// Define the pattern data for a smiling face
const int smilingFace[] = {                        //"^v^"…
};
// Define the data of numbers and letters, and save them in flash area
const int data[] PROGMEM = {…
};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Define a one-byte variable (8 bits) which is used to represent the selected state of 8 colu
  int cols;
  // Display the static smiling pattern
  for (int j = 0; j < 500; j++ ) {  // repeat 500 times
    cols = 0x01;
    for (int i = 0; i < 8; i++) {   // display 8 column data by scaning
      matrixRowsVal(smilingFace[i]);// display the data in this column
      matrixColsVal(~cols);          // select this column
      delay(1);                     // display them for a period of time
      matrixRowsVal(0x00);          // clear the data of this column
      cols <<= 1;                   // shift"cols" 1 bit left to select the next column
    }
  }
  // Display the dynamic patterns of numbers and letters
  for (int i = 0; i < 128; i++) {
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
```

Download the code to ESP32-WROVER, and the LED matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED matrix.

Sketch_16.3_LED_Matrix

The following is the program code:

```
int latchPin = 2;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 4;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 15;          // Pin connected to DS of 74HC595 (Pin14)

```

```
// Define the pattern data for a smiling face
const int smilingFace[] = {                        //"^v^"
  0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x14
};
// Define the data of numbers and letters, and save them in flash area
const int data[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
  0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
  0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
  0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
  0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
  0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
  0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
  0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
  0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
  0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
  0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
  0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
  0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00  // "F"
};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Define a one-byte variable (8 bits) which is used to represent the selected state of 8
column.
  int cols;
  // Display the static smiling pattern
  for (int j = 0; j < 500; j++ ) {  // repeat 500 times
    cols = 0x01;
    for (int i = 0; i < 8; i++) {   // display 8 column data by scanning
      matrixRowsVal(smilingFace[i]);// display the data in this column
      matrixColsVal(~cols);          // select this column
      delay(1);                     // display them for a period of time
      matrixRowsVal(0x00);          // clear the data of this column
```

```
      cols <<= 1;                    // shift "cols" 1 bit left to select the next column
    }
  }
  // Display the dynamic patterns of numbers and letters
  for (int i = 0; i < 128; i++) {
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
        matrixColsVal(~cols);            // select this column
        delay(1);                       // display them for a period of time
        matrixRowsVal(0x00);            // close the data of this column
        cols <<= 1;                    // shift "cols" 1 bit left to select the next column
      }
    }
  }
}

void matrixRowsVal(int value) {
  // make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  // make latchPin output high level, then 74HC595 will update the data to parallel output
  digitalWrite(latchPin, HIGH);
}

void matrixColsVal(int value) {
  // make latchPin output low level
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, MSBFIRST, value);
  // make latchPin output high level, then 74HC595 will update the data to parallel output
  digitalWrite(latchPin, HIGH);
}
```

The first "for" loop in the "while" loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```
  int cols;
  // Display the static smiling pattern
  for (int j = 0; j < 500; j++ ) {   // repeat 500 times
    cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the first
column is selected.
    for (int i = 0; i < 8; i++) {     // display 8 column data by scanning
```

```
      matrixRowsVal(smilingFace[i]);// display the data in this column
      matrixColsVal(~cols);         // select this column
      delay(1);                     // display them for a period of time
      cols <<= 1;                   // shift "cols" 1 bit left to select the next column
    }
  }
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of 17* 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column..... and so on···128-136 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

```
  // Display the dynamic patterns of numbers and letters
  for (int i = 0; i < 128; i++) {
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
        matrixColsVal(~cols);            // select this column
        delay(1);                       // display them for a period of time
        matrixRowsVal(0x00);            // close the data of this column
        cols <<= 1;                 // shift "cols" 1 bit left to select the next column
      }
    }
  }
```

The amount of pins of ESP32 is limited, so you need to find ways to save pins. If you use ESP32's GPIO to control the lattice without using 74HC595, you need 16 pins for the use of LED matrix. In this example, we use two 74HC595 to drive the LED matrix, requiring only three pins, so that we could save the rest of 13 pins.

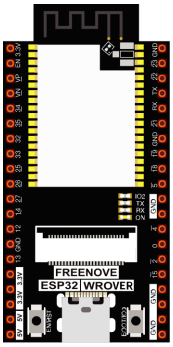
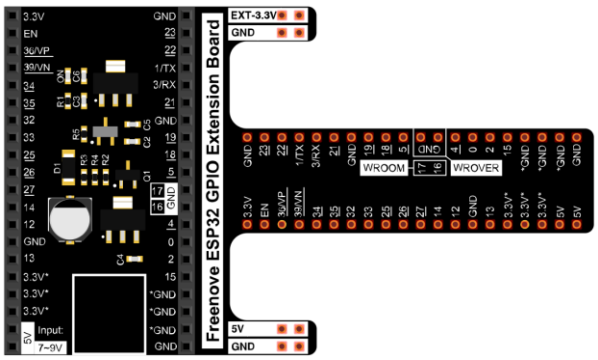
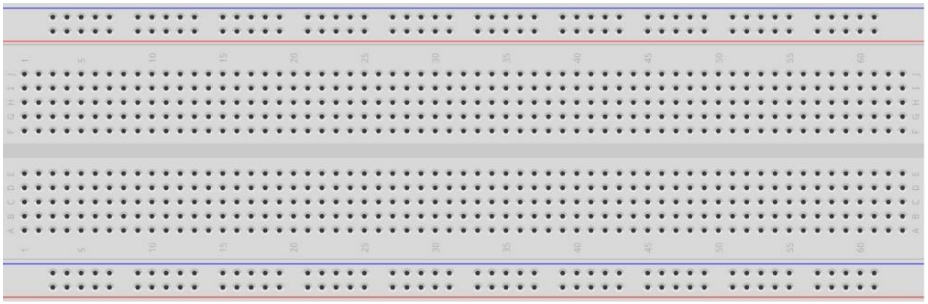
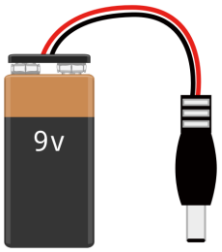
Chapter 17 Relay & Motor

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

Project 17.1 Relay & Motor

In this project, we will use a push button switch indirectly to control the motor via a relay.

Component List

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

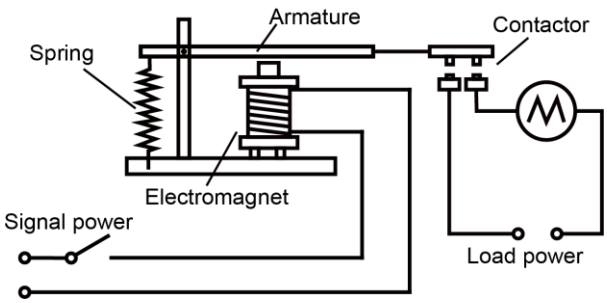
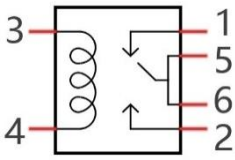
Resistor 10kΩ x2		Resistor 1kΩ x1	Resistor 220Ω x1		
NPN transistor x1	Relay x1	Motor x1	Push button x1	LED x1	Diode x1

Component knowledge

Relay

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

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

Diagram	Feature:	Symbol
		

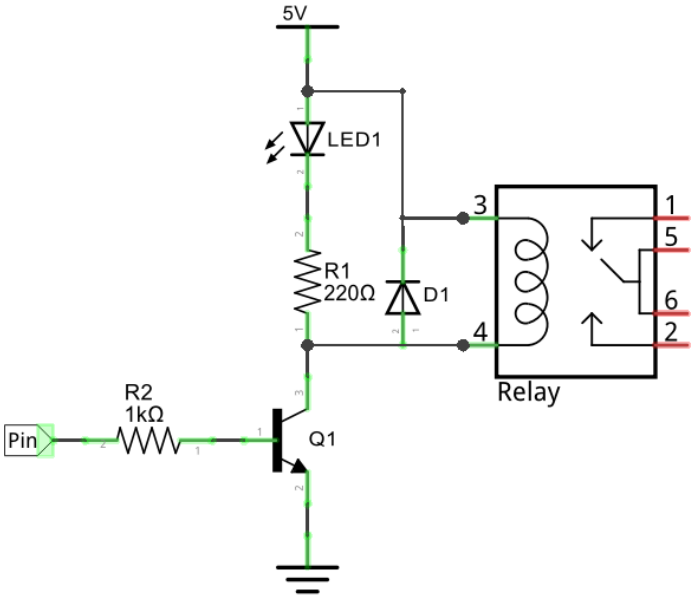
Pin 5 and pin 6 are connected to each other inside. When the coil pins 3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

Inductor

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

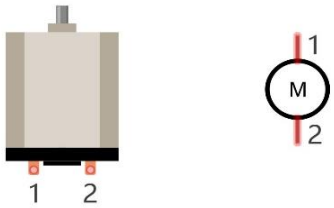
An inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductors hinder the change of current passing through it. When the current passing through it increases, it will attempt to hinder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

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

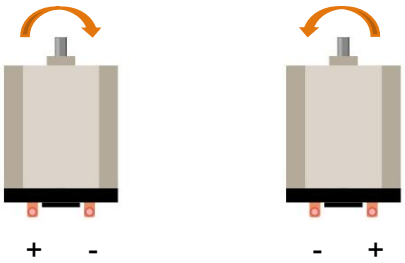


Motor

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

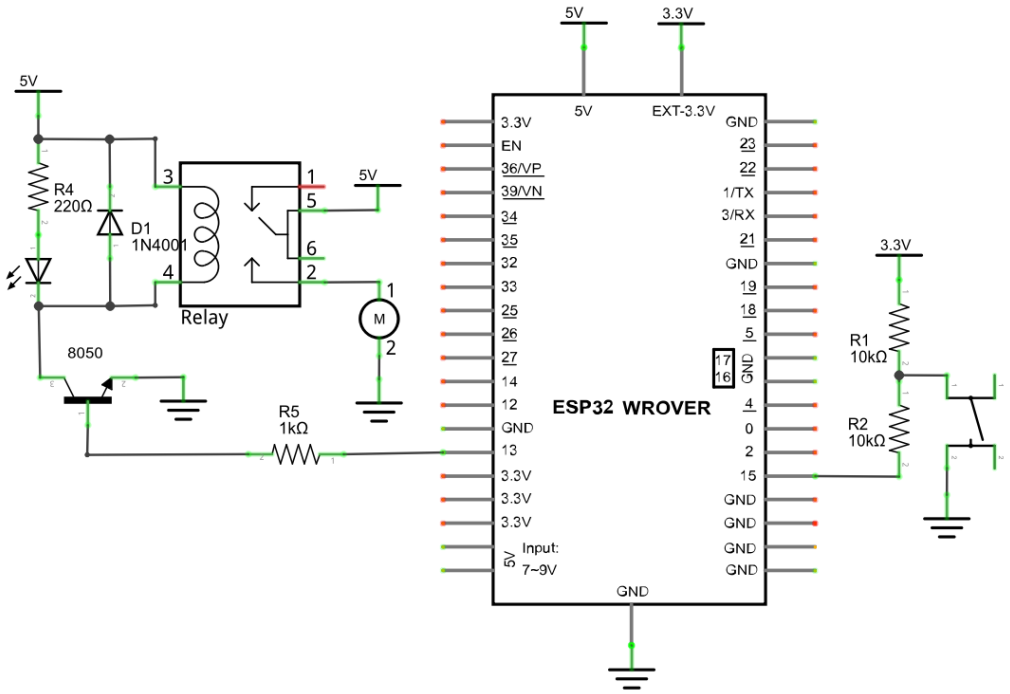


When a motor gets connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.

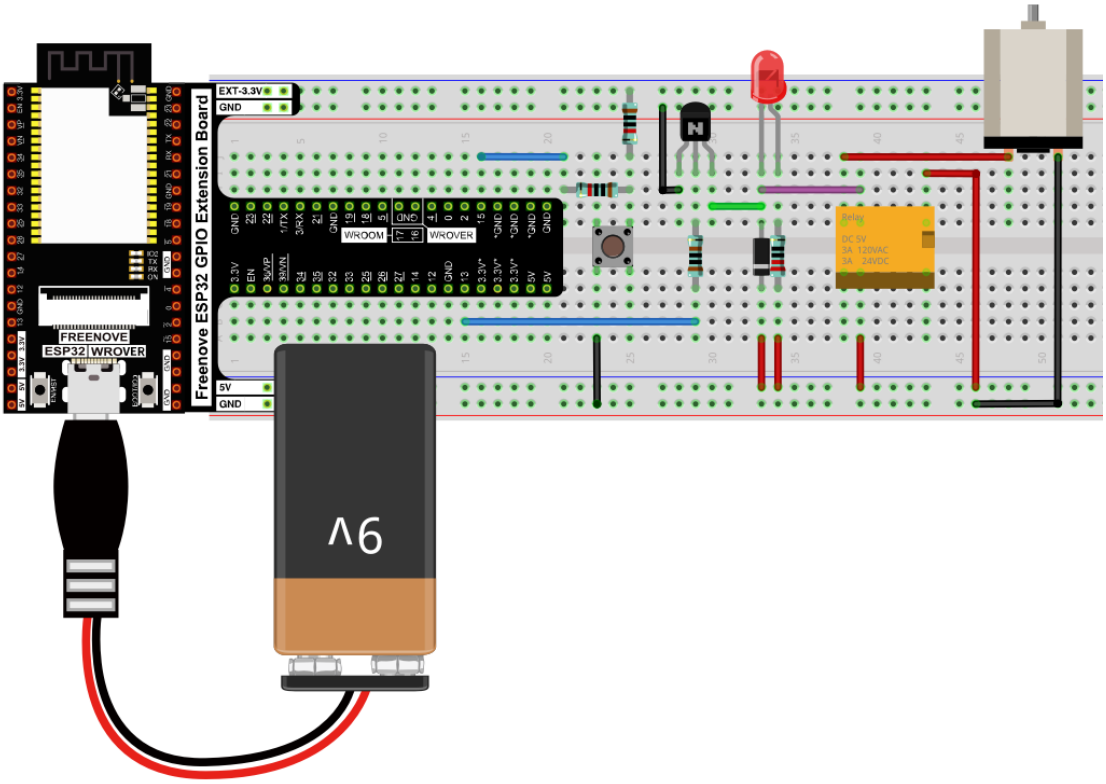


Circuit

Schematic diagram



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



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

Sketch

Use buttons to control the relays and motors.

Sketch_17.1_Control_Motor_by_Relay

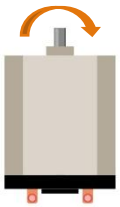
File Edit Sketch Tools Help

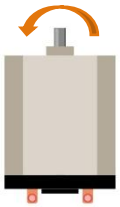
ESP32 Wrover Module

Sketch_17.1_Control_Motor_by_Relay.ino

```
int relayPin = 13;          // the number of the relay pin
int buttonPin = 15;         // the number of the push button pin

int buttonState = HIGH;     // Record button state, and initial the state to high level
int relayState = LOW;       // Record relay state, and initial the state to low level
int lastButtonState = HIGH; // Record the button state of last detection
long lastChangeTime = 0;    // Record the time point for button state change

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);           // Set push button pin into input mode
  pinMode(relayPin, OUTPUT);                  // Set relay pin into output mode
  digitalWrite(relayPin, relayState);         // Set the initial state of relay into "off"
}
void loop() {
  int nowButtonState = digitalRead(buttonPin);// Read current state of button pin
  // If button pin state has changed, record the time point
  if (nowButtonState != lastButtonState) {
    lastChangeTime = millis();
  }
  // If button state changes, and stays stable for a while, then it should have skipped the bounce area
  if (millis() - lastChangeTime > 10) {
    if (buttonState != nowButtonState) {    // Confirm button state has changed
      buttonState = nowButtonState;
      if (buttonState == LOW) {             // Low level indicates the button is pressed
        relayState = !relayState;           // Reverse relay state
        digitalWrite(relayPin, relayState); // Update relay state
      }
    }
  }
  lastButtonState = nowButtonState; // Save the state of last button
}
```

Download the code to ESP32-WROVER. When the DC motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC motor will rotate in opposite direction.





The following is the program code:

```
int relayPin = 13;          // the number of the relay pin
int buttonPin = 15;         // the number of the push button pin

int buttonState = HIGH;     // Record button state, and initial the state to high level
int relayState = LOW;       // Record relay state, and initial the state to low level
int lastButtonState = HIGH; // Record the button state of last detection
long lastChangeTime = 0;    // Record the time point for button state change

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);           // Set push button pin into input mode
  pinMode(relayPin, OUTPUT);                  // Set relay pin into output mode
  digitalWrite(relayPin, relayState);         // Set the initial state of relay into "off"
}
void loop() {
  int nowButtonState = digitalRead(buttonPin);// Read current state of button pin
  // If button pin state has changed, record the time point
  if (nowButtonState ! = lastButtonState) {
    lastChangeTime = millis();
  }
  // If button state changes, and stays stable for a while, then it should have skipped the
bounce area
  if (millis() - lastChangeTime > 10) {
    if (buttonState ! = nowButtonState) {    // Confirm button state has changed
      buttonState = nowButtonState;
      if (buttonState == LOW) {             // Low level indicates the button is pressed
        relayState = ! relayState;           // Reverse relay state
        digitalWrite(relayPin, relayState); // Update relay state
      }
    }
  }
  lastButtonState = nowButtonState; // Save the state of last button
}
```

In Chapter 2, the pressing and releasing of the button will result in mechanical vibrating. If we don't solve this problem, some unexpected consequences may happen to the procedure. Click here to return to Chapter 2 Button & LED

To eliminate the vibrating, we record the electrical level of the button with nowButtonState, and the time point for the last change of pin level with lastChangeTime If the state of the button changes, it will record the time point of the change.

```
  int nowButtonState = digitalRead(buttonPin);// Read current state of button pin
  // If button pin state has changed, record the time point
  if (nowButtonState ! = lastButtonState) {
    lastChangeTime = millis();
  }
```

If the state of the pin changes and keeps stable for a period of time, it can be considered as a valid key state change, update the key state variable buttonState, and determine whether the key is pressed or released according to the current state.

```
  // If button state changes, and stays stable for a while, then it should have skipped the
bounce area
  if (millis() - lastChangeTime > 10) {
    if (buttonState ! = nowButtonState) {    // Confirm button state has changed
      buttonState = nowButtonState;
      if (buttonState == LOW) {             // Low level indicates the button is pressed
        relayState = ! relayState;           // Reverse relay state
        digitalWrite(relayPin, relayState); // Update relay state
      }
    }
  }
  lastButtonState = nowButtonState;         // Save the state of last button
```

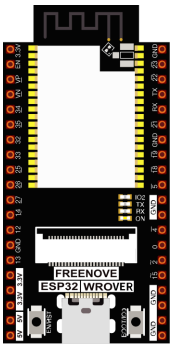
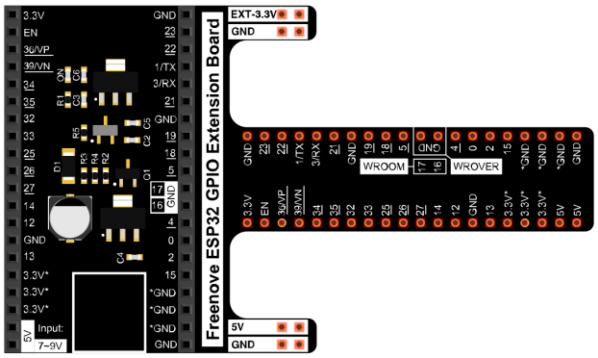
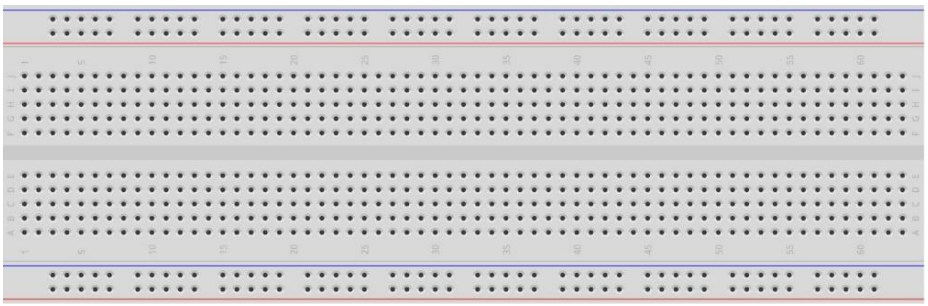
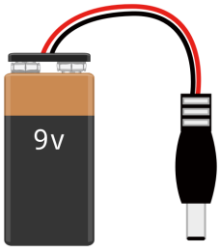
Chapter 17.2 Motor & Driver

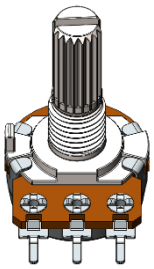
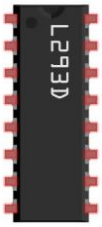
In this chapter, we will learn about DC motors and DC motor drivers and how to control the speed and direction of a DC motor.

Project 17.2 Control Motor with Potentiometer

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

Component List

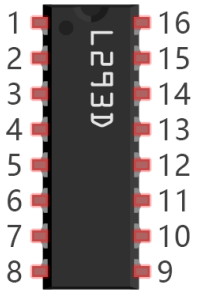
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper M/M	9V battery (prepared by yourself) & battery line 

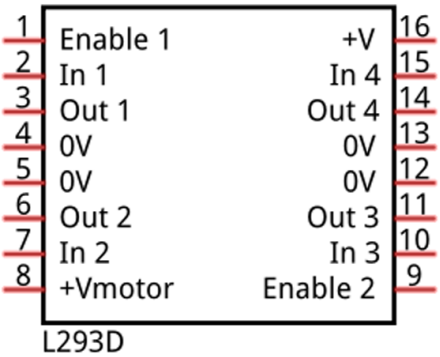
Rotary potentiometer x1	Motor x1	L293D
		

Component knowledge

L293D

L293D is an IC chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a unidirectional DC motor with 4 ports or a bi-directional DC motor with 2 ports or a stepper motor (stepper motors are covered later in this Tutorial).



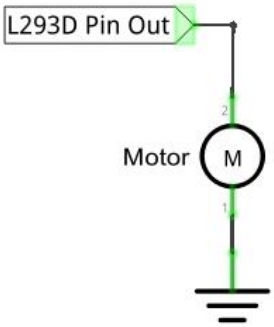


Port description of L293D module is as follows:

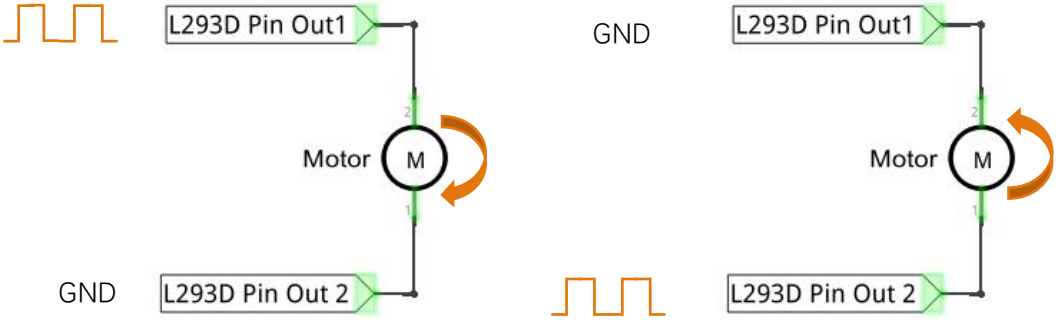
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 3.0~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

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

When using L293D to drive DC motor, there are usually two connection options.
The following connection option uses one channel of the L239D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.



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

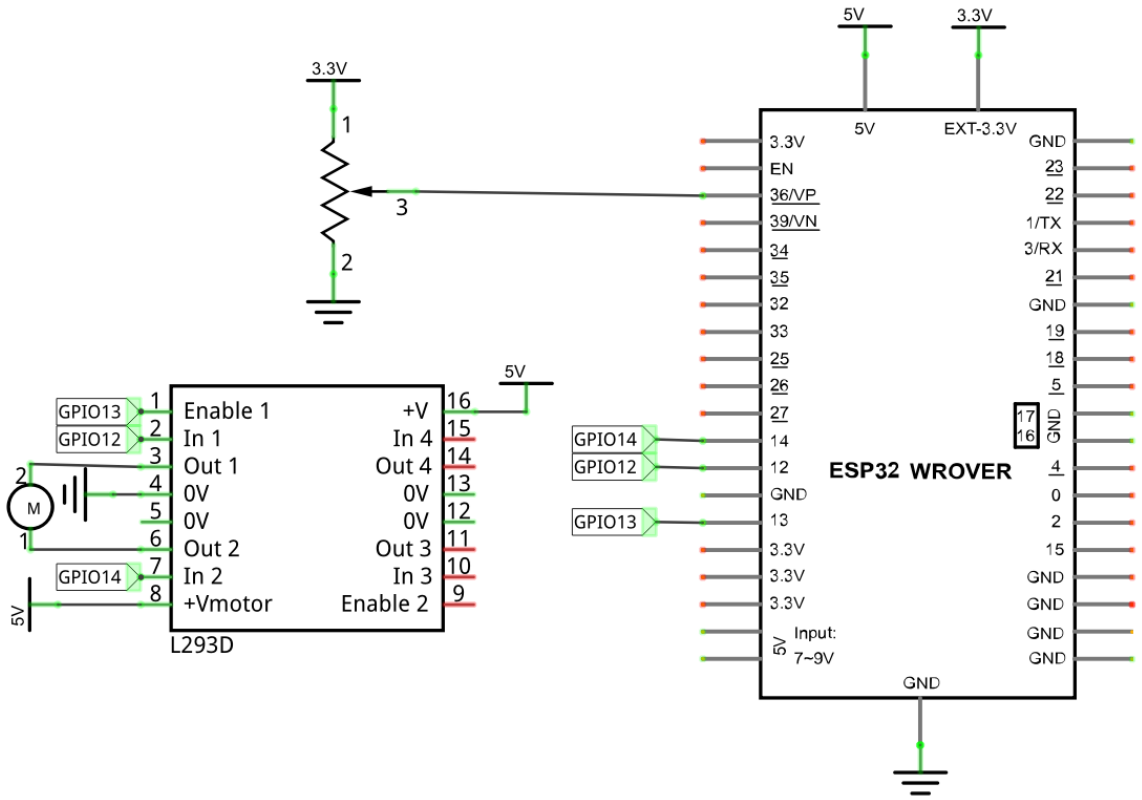


In practical use the motor is usually connected to channel 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

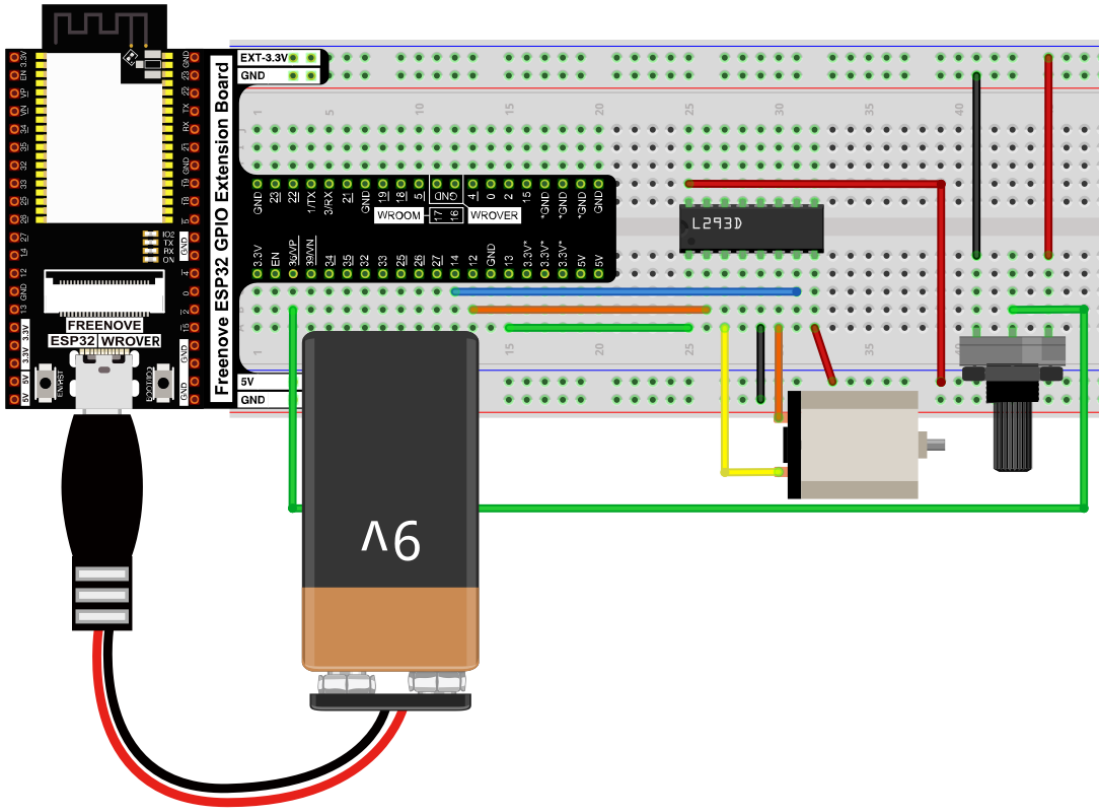
Circuit

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

Schematic diagram



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



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

Sketch

Sketch_17.2_Control_Motor_by_L293D

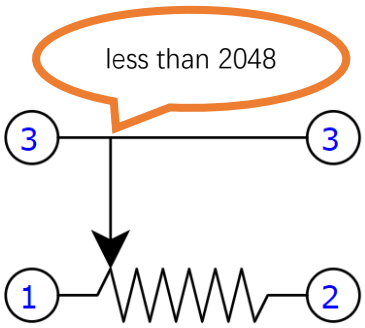
File Edit Sketch Tools Help

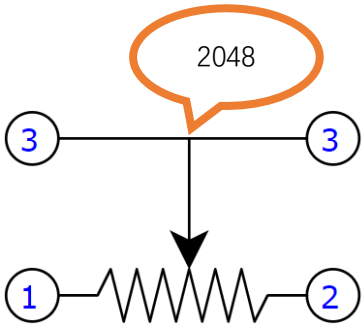
ESP32 Wrover Module

Sketch_17.2_Control_Motor_by_L293D.ino

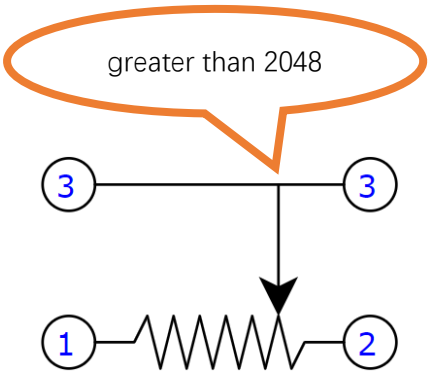
```
int in1Pin = 12;       // Define L293D channel 1 pin
int in2Pin = 14;       // Define L293D channel 2 pin
int enable1Pin = 13;   // Define L293D enable 1 pin
int channel = 0;

boolean rotationDir;   // Define a variable to save the motor's rotation direction
int rotationSpeed;     // Define a variable to save the motor rotation speed

void setup() {
  // Initialize the pin into an output mode:
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  ledcAttachChannel(enable1Pin,1000,11,channel);//Set PWM to 11 bits, range is 0-2047
}

void loop() {
  int potenVal = analogRead(A0);      // Convert the voltage of rotary potentiometer into digital
  //Compare the number with value 2048,
  //if more than 2048, clockwise rotates, otherwise, counter clockwise rotates
  rotationSpeed = potenVal - 2048;
  if (potenVal > 2048)
    rotationDir = true;
  else
    rotationDir = false;
  // Calculate the motor speed
  rotationSpeed = abs(potenVal - 2048);
  //Control the steering and speed of the motor
  driveMotor(rotationDir, constrain(rotationSpeed,0,2048));
}
```

Download code to ESP32-WROVER, rotate the potentiometer in one direction and the motor speeds up slowly in one direction. And then rotate the potentiometer in the other direction and the motor will slow down to stop. And then rotate it in an inverse direction to accelerate the motor.

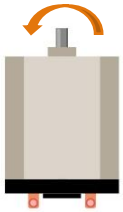



2048
3
3
1
2

1 3 2


greater than 2048
3
3
1
2

1 3 2



The following is the sketch:

```
int in1Pin = 12;      // Define L293D channel 1 pin
int in2Pin = 14;      // Define L293D channel 2 pin
int enable1Pin = 13;  // Define L293D enable 1 pin
int channel = 0;

boolean rotationDir;  // Define a variable to save the motor's rotation direction
int rotationSpeed;    // Define a variable to save the motor rotation speed

void setup() {
  // Initialize the pin into an output mode:
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  ledcAttachChannal(enable1Pin, 1000, 11, channel); //Set PWM to 11 bits, range is 0-2047
}

void loop() {
  int potenVal = analogRead(A0);// Convert the voltage of rotary potentiometer into digital
  rotationSpeed = potenVal - 2048;
  if (potenVal > 2048)
    rotationDir = true;
  else
    rotationDir = false;
  // Calculate the motor speed
  rotationSpeed = abs(potenVal - 2048);
  //Control the steering and speed of the motor
  driveMotor(rotationDir, constrain(rotationSpeed,0,2048));
}

void driveMotor(boolean dir, int spd) {
  if (dir) {// Control motor rotation direction
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  }
  else {
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  }
  ledcWrite(enable1Pin, spd); // Control motor rotation speed
}
```

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

ADC value and take the absolute value and use this result as the speed of the motor.

```
  int potenVal = analogRead(A0);// Convert the voltage of rotary potentiometer into digital
  rotationSpeed = potenVal - 2048;
  if (potenVal > 2048)
    rotationDir = true;
  else
    rotationDir = false;
  // Calculate the motor speed
  rotationSpeed = abs(potenVal - 2048);
  //Control the steering and speed of the motor
  driveMotor(rotationDir, constrain(rotationSpeed,0,2048));
}
```

Set the accuracy of the PWM to 11 bits and range from 0 to 2047 to control the rotation speed of the motor.

```
ledcAttachChannal(enable1Pin, 1000, 11, channel); //Set PWM to 11 bits, range is 0-2047
```

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

```
void driveMotor(boolean dir, int spd) {
  // Control motor rotation direction
  if (rotationDir) {
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  }
  else {
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  }
  // Control motor rotation speed
  ledcWrite(enable1Pin, spd);
}
```

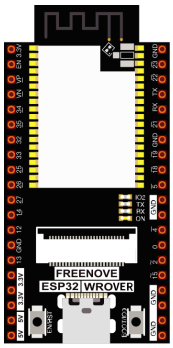
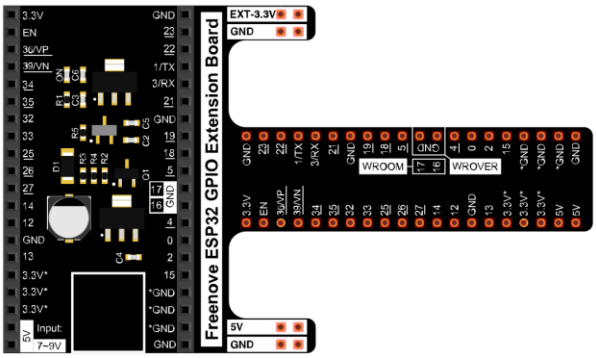
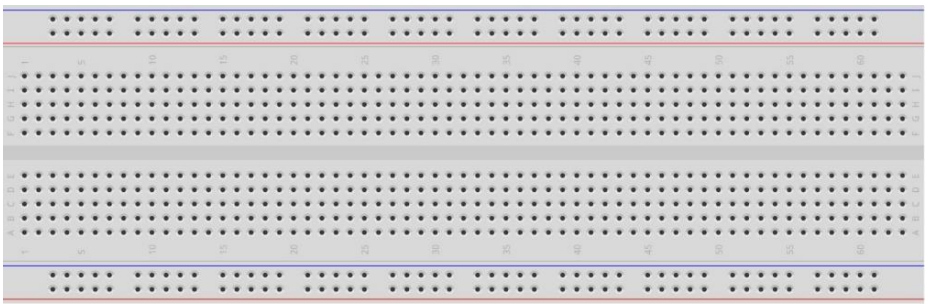
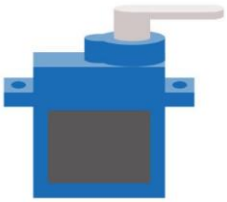
Chapter 18 Servo

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

Project 18.1 Servo Sweep

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

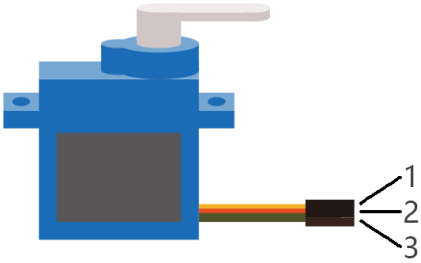
Component List

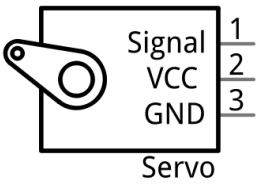
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Servo x1 	Jumper M/M x3

Component knowledge

Servo

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





We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The lasting time of 0.5ms-2.5ms of PWM single cycle high level corresponds to the servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

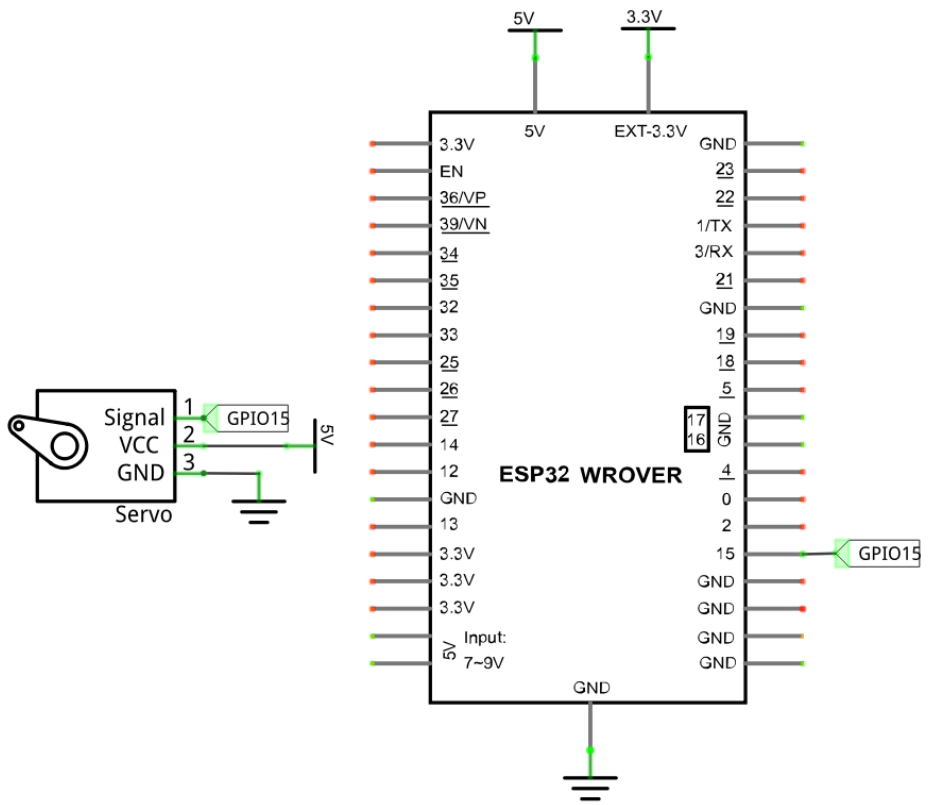
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

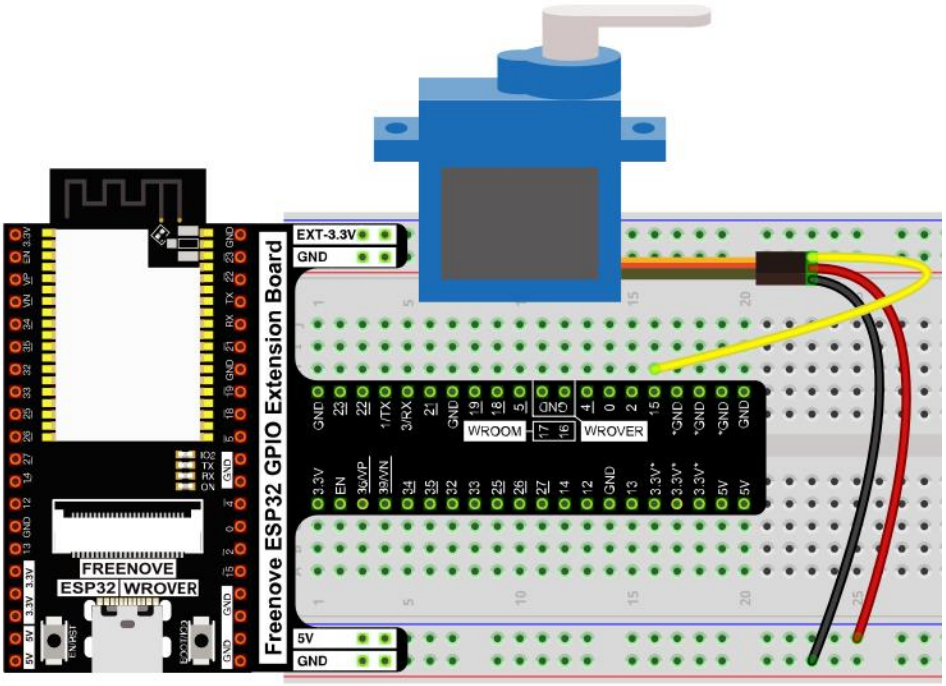
Circuit

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

Schematic diagram



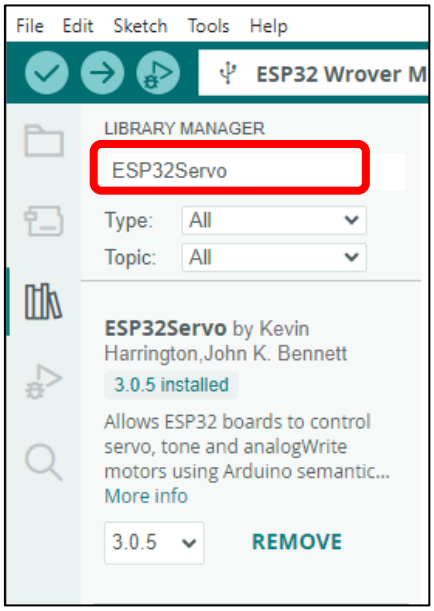
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

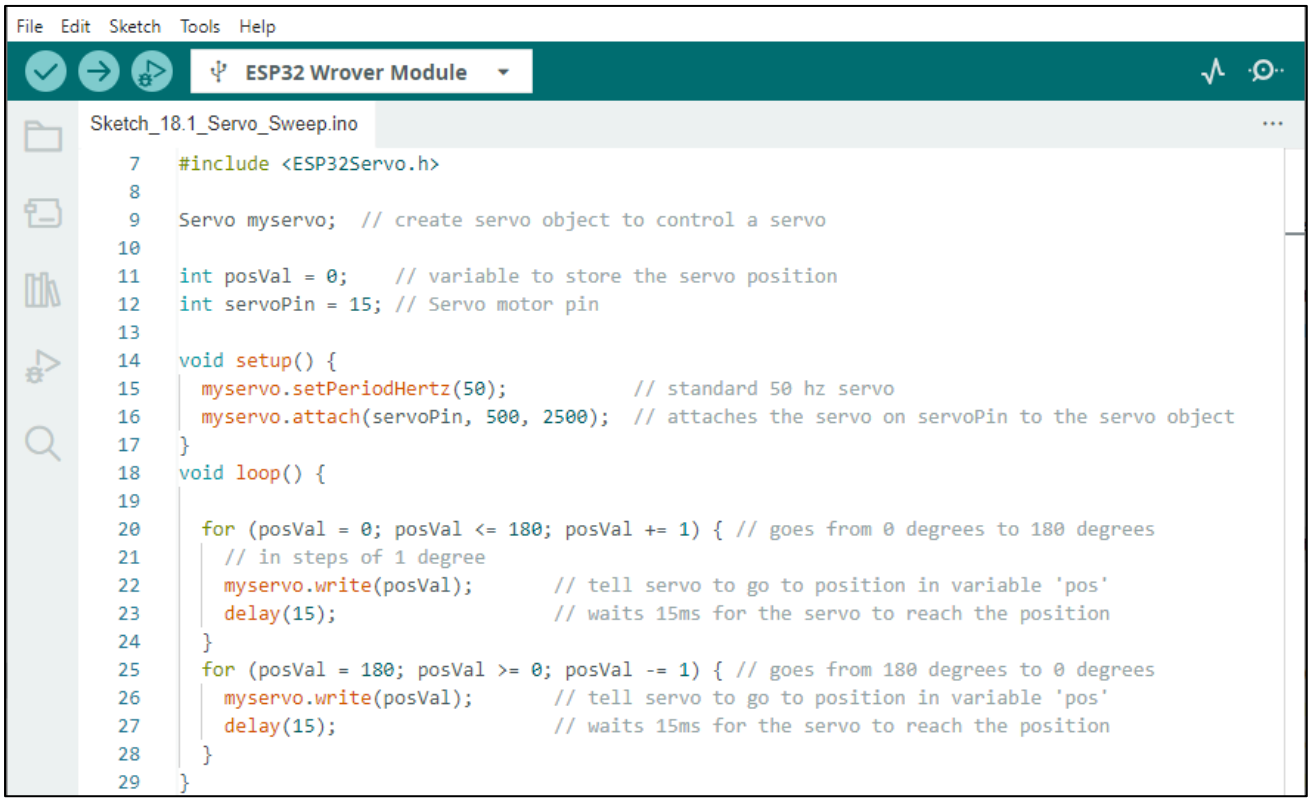
How to install the library

If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter " ESP32Servo" in the search bar and select "ESP32Servo" for installation. Refer to the following operations:



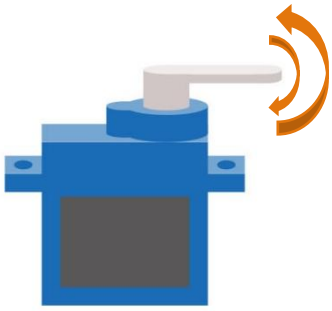
Use the ESP32Servo library to control the servo motor and let the servo motor rotate back and forth.

Sketch_18.1_Servo_Sweep




```
#include <ESP32Servo.h>

Servo myservo;  // create servo object to control a servo

int posVal = 0;    // variable to store the servo position
int servoPin = 15; // Servo motor pin

void setup() {
  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo object
}
void loop() {

  for (posVal = 0; posVal <= 180; posVal += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(posVal);       // tell servo to go to position in variable 'pos'
    delay(15);                   // waits 15ms for the servo to reach the position
  }
  for (posVal = 180; posVal >= 0; posVal -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(posVal);       // tell servo to go to position in variable 'pos'
    delay(15);                   // waits 15ms for the servo to reach the position
  }
}
```

Compile and upload the code to ESP32-WROVER, the servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.



The following is the program code:

```
1  #include <ESP32Servo.h>
2
3  Servo myservo;  // create servo object to control a servo
4  int posVal = 0;    // variable to store the servo position
5  int servoPin = 15; // Servo motor pin
6  void setup() {
7      myservo.setPeriodHertz(50);           // standard 50 hz servo
8      myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo
   object
9  }
10 void loop() {
11     for (posVal = 0; posVal <= 180; posVal += 1) { // goes from 0 degrees to 180 degrees
12         // in steps of 1 degree
13         myservo.write(posVal);       // tell servo to go to position in variable 'pos'
14         delay(15);                   // waits 15ms for the servo to reach the position
15     }
16     for (posVal = 180; posVal >= 0; posVal -= 1) { // goes from 180 degrees to 0 degrees
17         myservo.write(posVal);       // tell servo to go to position in variable 'pos'
18         delay(15);                   // waits 15ms for the servo to reach the position
19     }
20 }
```

Servo uses the ESP32Servo library, like the following reference to ESP32Servo library:

```
1  #include <ESP32Servo.h>
```

ESP32Servo library provides the ESP32Servo class that controls it. ESP32Servo class must be instantiated before using:

```
3  Servo myservo;  // create servo object to control a servo
```

Set the control servo motor pin, the time range of high level.

```
8  myservo.attach(servoPin,500,2500);
```

After initializing the servo, you can control the servo to rotate to a specific angle:

```
17 myservo.write(posVal);
```

Reference

class Servo	

Servo class must be instantiated when used, that is, define an object of Servo type, for example:

```
Servo myservo;
```

The function commonly used in the servo class is as follows:

`setPeriodHertz(data)`: Set the frequency of the servo motor.

attach(pin,low,high): Initialize the servo,

`pin`: the port connected to servo signal line.

`low`: set the time of high level corresponding to 0 degree.

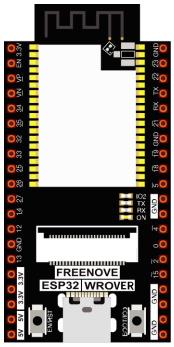
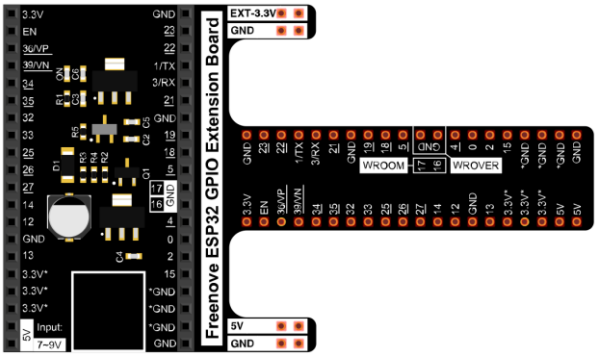
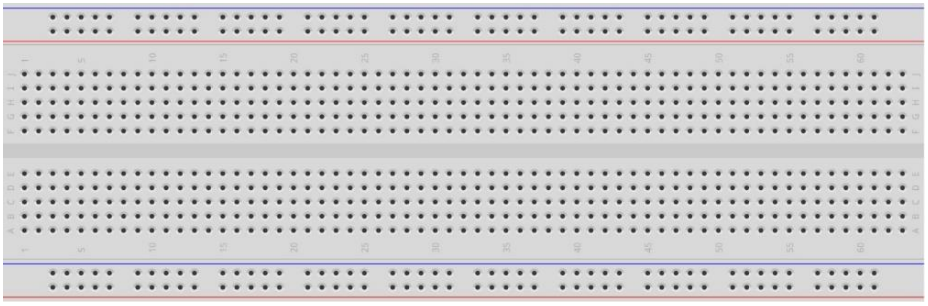
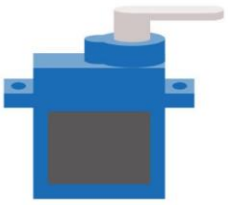
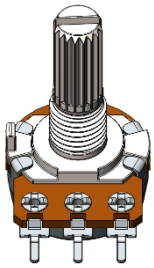
`high`: set the time of high level corresponding to 180 degrees.

write(angle): Control servo to rotate to the specified angle.

Project 18.2 Servo Knop

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

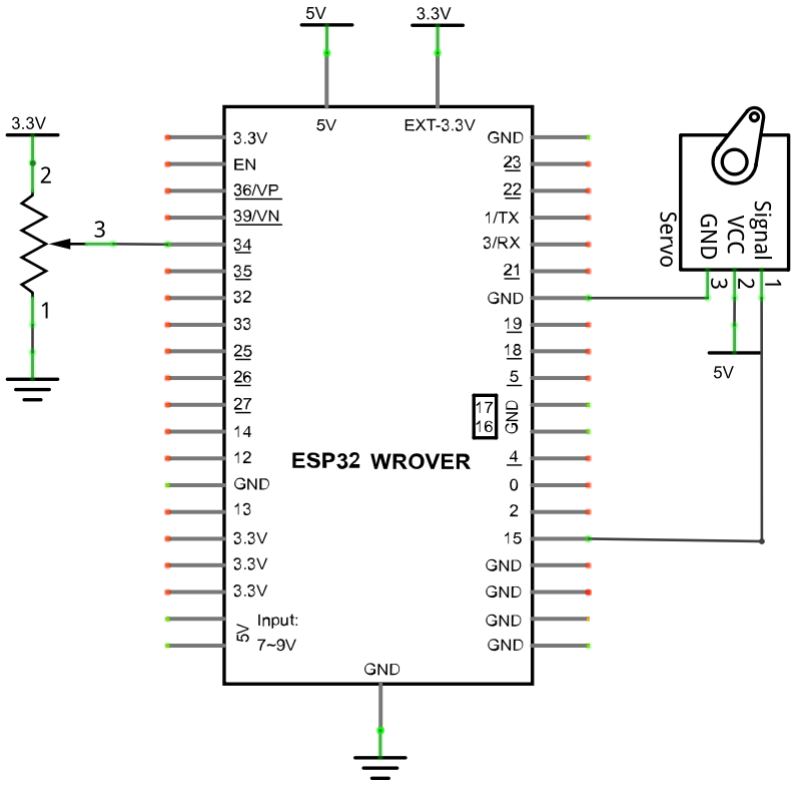
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Servo x1 	Jumper M/M x6	Rotary potentiometer x1 

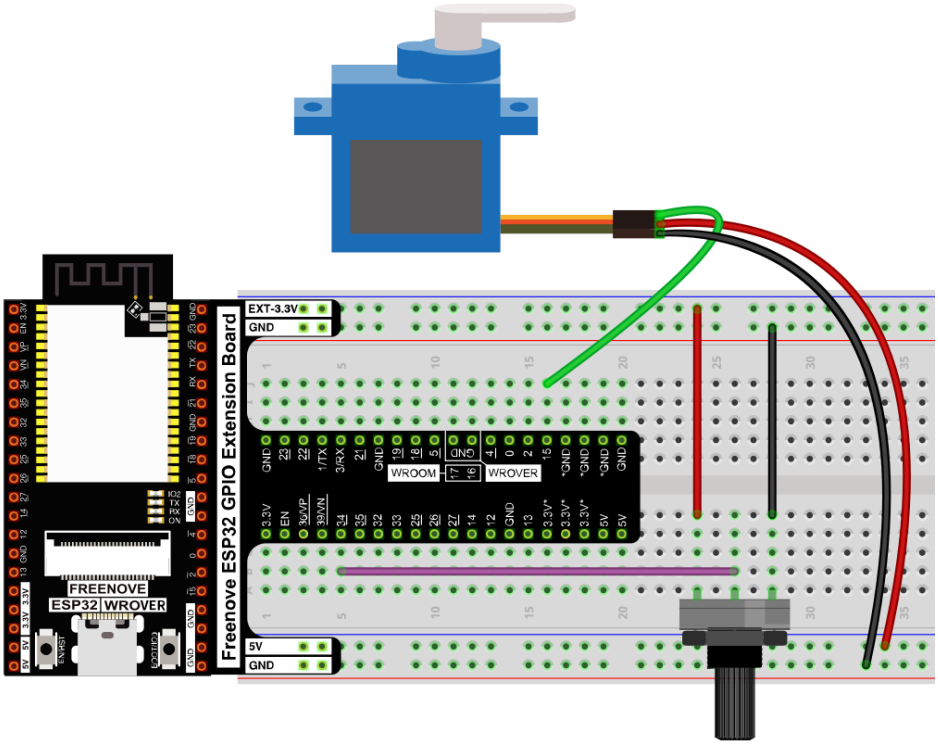
Circuit

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

Schematic diagram



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



Sketch

Sketch_18.2_Servo_Sweep

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_18.2_Control_Servo_by_Potentiometer.ino

```
#include <ESP32Servo.h>
#define ADC_Max 4095    // This is the default ADC max value on the ESP32 (12 bit ADC width);

Servo myservo;  // create servo object to control a servo

int servoPin = 15;      // GPIO pin used to connect the servo control (digital out)
int potPin = 34;        // GPIO pin used to connect the potentiometer (analog in)
int potVal;             //variable to read the value from the analog pin

void setup()
{
  myservo.setPeriodHertz(50);// Standard 50hz servo
  // attaches the servo on servoPin to the servo object
  myservo.attach(servoPin, 500, 2500);
  Serial.begin(115200);
}

void loop() {
  // read the value of the potentiometer (value between 0 and 4095)
  potVal = analogRead(potPin);
  Serial.printf("potVal_1: %d\t",potVal);
  // scale it to use it with the servo (value between 0 and 180)
  potVal = map(potVal, 0, ADC_Max, 0, 180);
  // set the servo position according to the scaled value
  myservo.write(potVal);
  Serial.printf("potVal_2: %d\n",potVal);
  delay(15);// wait for the servo to get there
}
```

Compile and upload the code to ESP32-WROVER, twist the potentiometer back and forth, and the servo motor rotates accordingly.

The following is the program code:

```
#include <ESP32Servo.h>
#define ADC_Max 4095    // This is the default ADC max value on the ESP32 (12 bits ADC width);

Servo myservo;          // create servo object to control a servo

int servoPin = 15;      // GPIO pin used to connect the servo control (digital out)
int potPin = 34;        // GPIO pin used to connect the potentiometer (analog in)
int potVal;             //variable to read the value from the analog pin

void setup()
{
  myservo.setPeriodHertz(50);// Standard 50hz servo
  myservo.attach(servoPin, 500, 2500);   // attaches the servo on servoPin to the servo object
  Serial.begin(115200);
}

void loop() {
  potVal = analogRead(potPin);             // read the value of the potentiometer (value between 0 and 1023)
  Serial.printf("potVal_1: %d\t",potVal);
  potVal = map(potVal, 0, ADC_Max, 0, 180);// scale it to use it with the servo (value between 0 and 180)
  myservo.write(potVal);                   // set the servo position according to the scaled value
  Serial.printf("potVal_2: %d\n",potVal);
  delay(15);                               // wait for the servo to get there
}
```

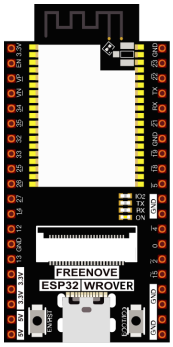
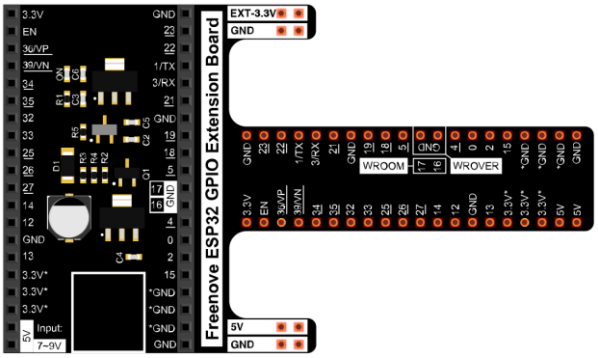
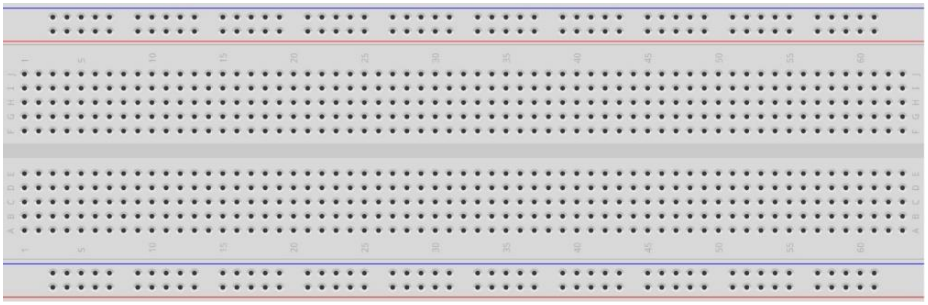
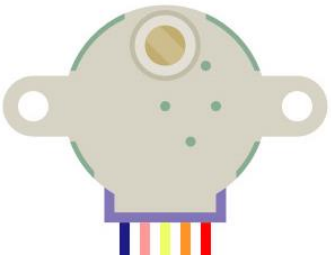
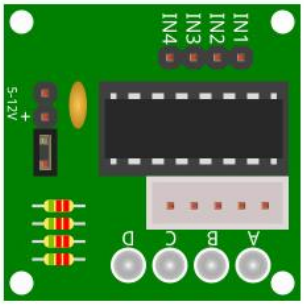
In this experiment, we obtain the ADC value of the potentiometer and store it in potVal. Use map function to convert it into corresponding angle value and we can control the motor to rotate to a specified angle, and print the value via serial.

Chapter 19 Stepper Motor

In this project, we will learn how to drive a stepper motor, and understand its working principle.

Project 19.1 Stepper Motor

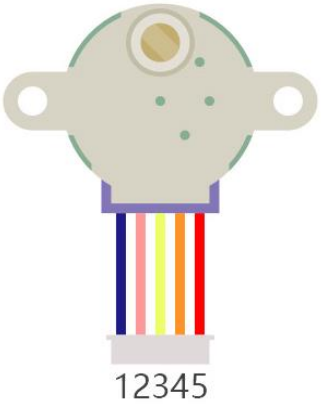
Component List

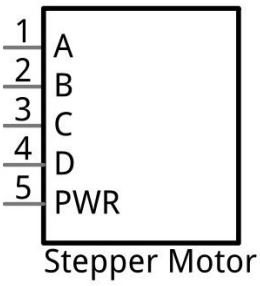
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Stepper Motor x1 	ULN2003 Stepper Motor Driver x1 	Jumper F/M x6

Component knowledge

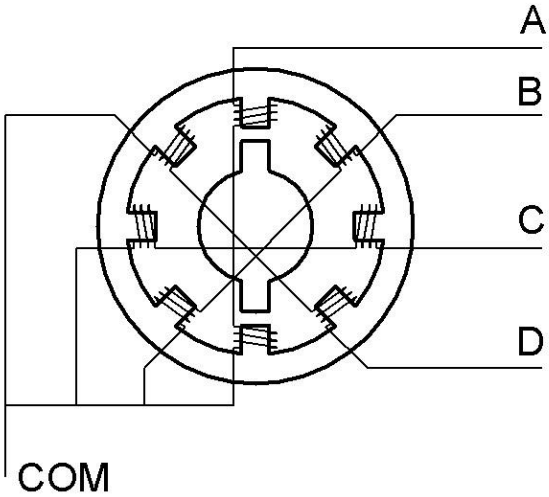
Stepper Motor

Stepper motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC motor. A small four-phase deceleration stepper motor is shown here:



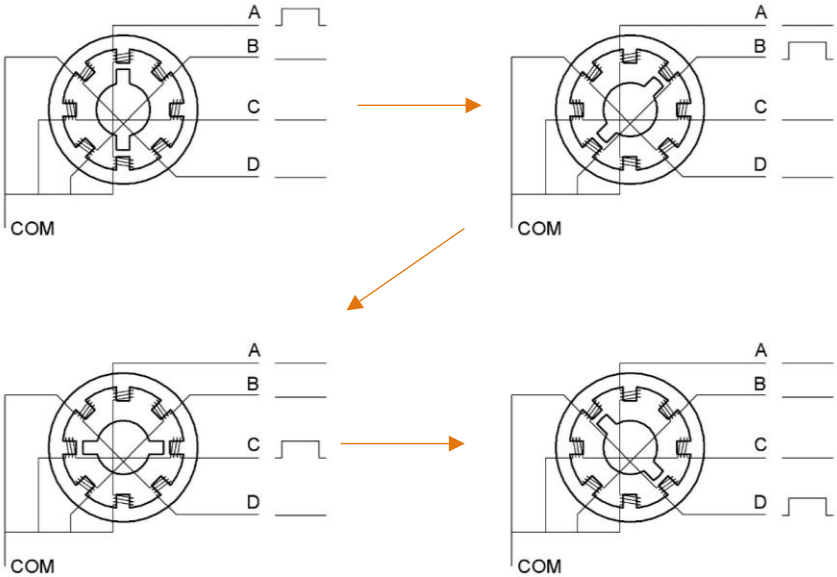


The electronic schematic diagram of a four-phase stepper motor is shown below:



The outside case or housing of the stepper motor is the stator and inside the stator is the rotor. There are a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the rotor's surface. The rotor is usually made of iron or a permanent magnet. Therefore, the stepper motor can be driven by powering the coils on the stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving process is as follows:



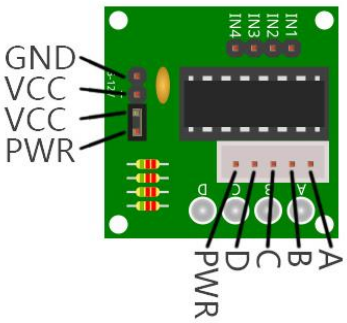
In the course above, the stepper motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepper motor rotation angle. By controlling the time between two steps, you can control the stepper motor rotation speed. When rotating clockwise, the order of coil powered on is: A→B→C→D→A→····· . And the rotor will rotate in accordance with the order, step by step down, called four steps four pats. If the coils is powered on in the reverse order, D→C→B→A→D→··· , the rotor will rotate in anti-clockwise direction.

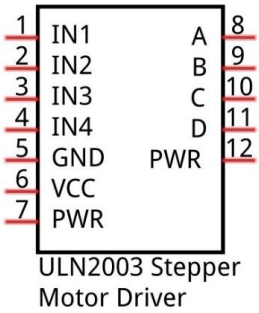
There are other methods to control stepper motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the stepper motor and reduces noise. The sequence of powering the coils looks like this: A→ AB→B →BC→C →CD→D→DA→A→·····, the rotor will rotate in accordance to this sequence at a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the stepper motor will rotate in the opposite direction.

The stator in the stepper motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the stepper motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the stepper motor's housing) requires 32 X 64 = 2048 steps to make one full revolution.

ULN2003 Stepper motor driver

A ULN2003 stepper motor driver is used to convert weak signals into more powerful control signals in order to drive the stepper motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the stepper motor. By default, PWR and VCC are connected.

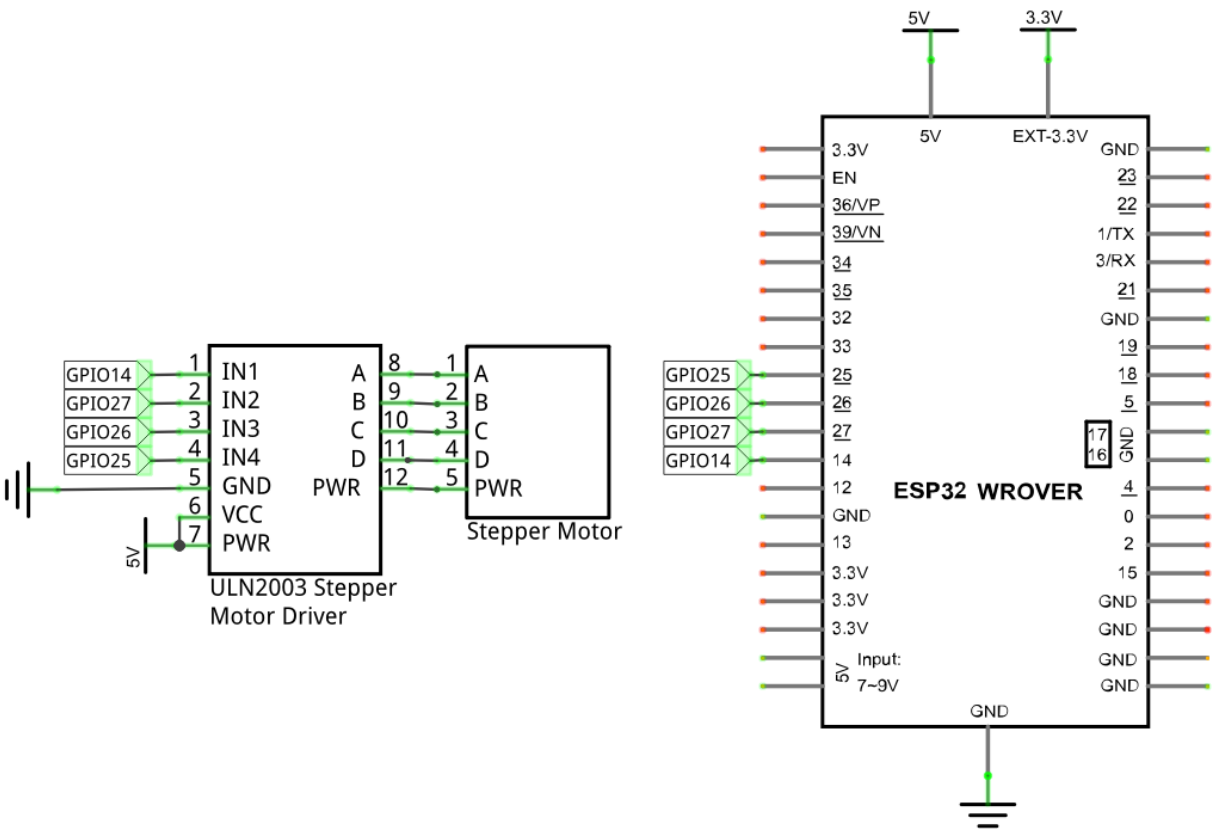




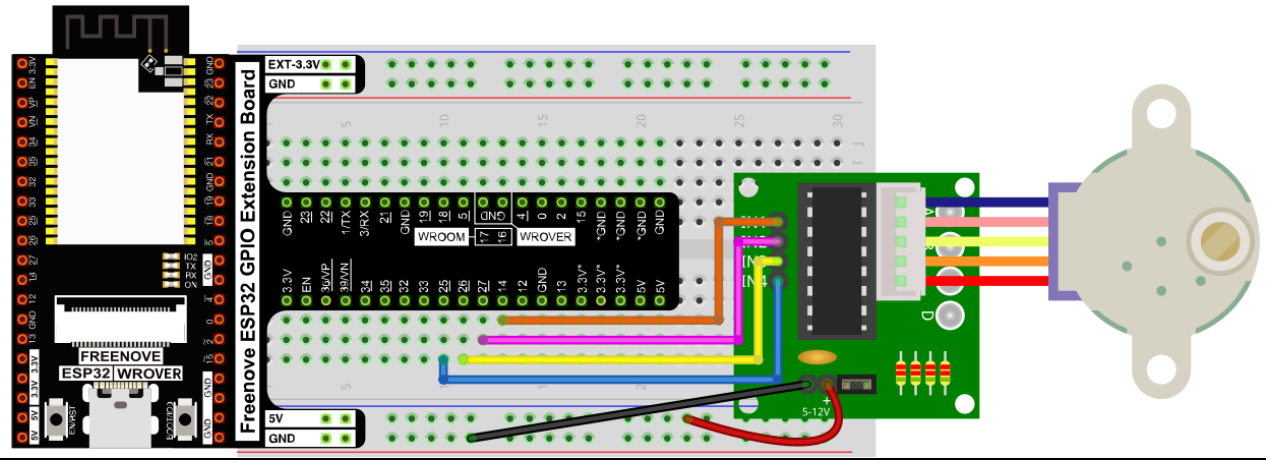
Circuit

When building the circuit, note that rated voltage of the stepper motor is 5V, and we need to use the breadboard power supply independently. Additionally, the breadboard power supply needs to share Ground with ESP32.

Schematic diagram



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



Sketch

This code uses the four-step, four-part mode to drive the stepper motor in the clockwise and anticlockwise directions.

Sketch_19.1_Drive_Stepper_Motor

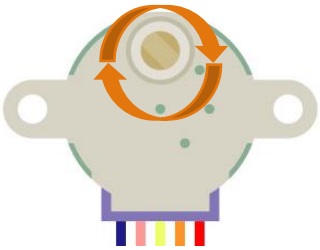
File Edit Sketch Tools Help

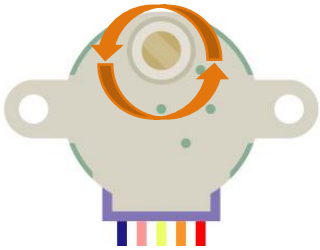
ESP32 Wrover Module

Sketch_19.1_Drive_Stepper_Motor.ino

```
// Conncet the port of the stepper motor driver
int outPorts[] = {14, 27, 26, 25};

void setup() {
  // set pins to output
  for (int i = 0; i < 4; i++) {
    pinMode(outPorts[i], OUTPUT);
  }
}

void loop()
{
  // Rotate a full turn
  moveSteps(true, 32 * 64, 3);
  delay(1000);
  // Rotate a full turn towards another direction
  moveSteps(false, 32 * 64, 3);
  delay(1000);
}

//Suggestion: the motor turns precisely when the ms range is between 3 and 20
void moveSteps(bool dir, int steps, byte ms) {
  for (unsigned long i = 0; i < steps; i++) {
    moveOneStep(dir); // Rotate a step
    delay(constrain(ms,3,20));        // Control the speed
  }
}

void moveOneStep(bool dir) {
  // Define a variable, use four low bit to indicate the state of port
  static byte out = 0x01;
  // Decide the shift direction according to the rotation direction
  if (dir) {  // ring shift left
    out != 0x08 ? out = out << 1 : out = 0x01;
  }
```

Compile and upload the code to the ESP32-WROVER, the stepper motor will rotate 360° clockwise and stop for 1s, and then rotate 360° anticlockwise and stop for 1s. And it will repeat this action in an endless loop.





The following is the program code:

```
// Connect the port of the stepper motor driver
int outPorts[] = {14, 27, 26, 25};

void setup() {
  // set pins to output
  for (int i = 0; i < 4; i++) {
    pinMode(outPorts[i], OUTPUT);
  }
}

void loop(){
  // Rotate a full turn
  moveSteps(true, 32 * 64, 3);
  delay(1000);
  // Rotate a full turn towards another direction
  moveSteps(false, 32 * 64, 3);
  delay(1000);
}

//Suggestion: the motor turns precisely when the ms range is between 3 and 20
void moveSteps(bool dir, int steps, byte ms) {
  for (unsigned long i = 0; i < steps; i++) {
    moveOneStep(dir); // Rotate a step
    delay(constrain(ms,3,20));        // Control the speed
  }
}

void moveOneStep(bool dir) {
  // Define a variable, use four low bit to indicate the state of port
  static byte out = 0x01;
  // Decide the shift direction according to the rotation direction
  if (dir) {  // ring shift left
    out ! = 0x08 ?  out = out << 1 : out = 0x01;
  }
  else {      // ring shift right
```

```
    out ! = 0x01 ?  out = out >> 1 : out = 0x08;
  }
  // Output singal to each port
  for (int i = 0; i < 4; i++) {
    digitalWrite(outPorts[i], (out & (0x01 << i)) ?  HIGH : LOW);
  }
}

void moveAround(bool dir, int turns, byte ms){
  for(int i=0;i<turns;i++)
    moveSteps(dir,32*64,ms);
}
void moveAngle(bool dir, int angle, byte ms){
  moveSteps(dir,(angle*32*64/360),ms);
}
```

In this project, we define four pins to drive stepper motor.

```
// Connect the port of the stepper motor driver
int outPorts[] = {14, 27, 26, 25};
```

moveOneStep Function is used to drive the stepper motor to rotate clockwise or counterclockwise. The parameter "dir" indicates the direction of rotation. If "dir" returns true, the stepper motor rotates clockwise, otherwise the stepper motor rotates counterclockwise.

```
void moveOneStep(bool dir) {
...
}
```

Define a static byte variable, calculate the value of the variable according to the rotation direction of the motor, and use the keyword static to save the position status of the previous step of the stepper motor. Use the four low bits of the variable to control the output state of the four pins.

```
    // Define a variable, use four low bit to indicate the state of port
  static byte out = 0x01;
  // Decide the shift direction according to the rotation direction
  if (dir) {   // ring shift left
    out ! = 0x08 ?  out = out << 1 : out = 0x01;
  }
  else {      // ring shift right
    out ! = 0x01 ?  out = out >> 1 : out = 0x08;
  }
```

Make the pin to output corresponding level based on the value of the variable.

```
  // Output singal to each port
  for (int i = 0; i < 4; i++) {
    digitalWrite(outPorts[i], (out & (0x01 << i)) ?  HIGH : LOW);
  }
```

The moveSteps function can control the direction of the stepper motor, the number of rotation steps, and the speed of rotation. According to the previous knowledge, the stepper motor needs 32*64 steps for one revolution. The speed of rotation is determined by the parameter ms. The larger the ms is, the slower the rotation speed is. There is a range for the speed of the motor, which is determined by the motor itself and according to our test, the value of ms is limited to 3-20.

```
//Suggestion: the motor turns precisely when the ms range is between 3 and 20
void moveSteps(bool dir,int steps, byte ms) {
  for (unsigned long i = 0; i < steps; i++) {
    moveOneStep(dir); // Rotate a step
    delay(constrain(ms,3,20));        // Control the speed
  }
}
```

The function moveTurns() is a further package of moveSteps(), which is used to control the stepper motor to rotate a specified number of turns. The parameter "turns" represents the number of turns that need to be rotated.

```
void moveAround(bool dir, int turns, byte ms){
  for(int i=0;i<turns;i++)
    moveSteps(dir,32*64,ms);
}
```

The function moveAround () is a further package of moveSteps (), which is used to control the stepper motor to rotate by a specified angle, and the parameter "angle" represents the angle to be rotated.

```
void moveAngle(bool dir, int angle, byte ms){
  moveSteps(dir,(angle*32*64/360),ms);
}
```

In the loop function, call the moveSteps function to loop the stepper motor: rotate clockwise one turn and stop for 1s, then rotate counterclockwise one turn and stop for 1s.

```
void loop(){
  // Rotate a full turn
  moveSteps(true, 32 * 64, 3);
  delay(1000);
  // Rotate a full turn towards another direction
  moveSteps(false, 32 * 64, 3);
  delay(1000);
}
```

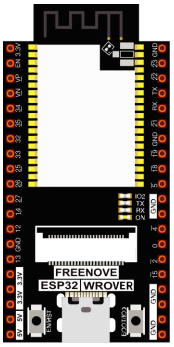
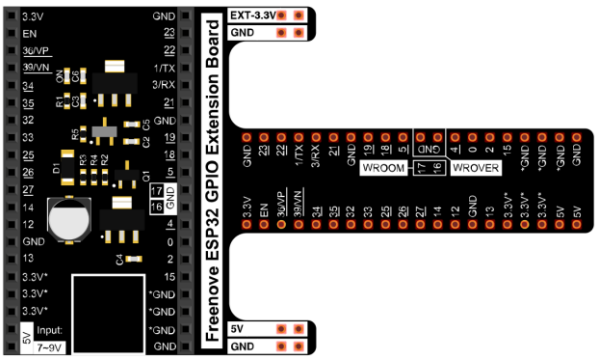
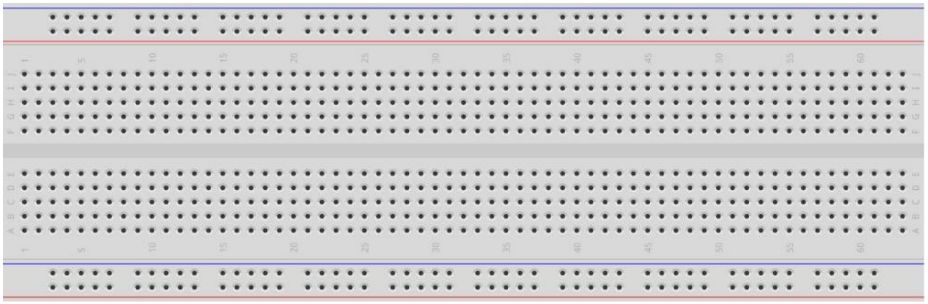
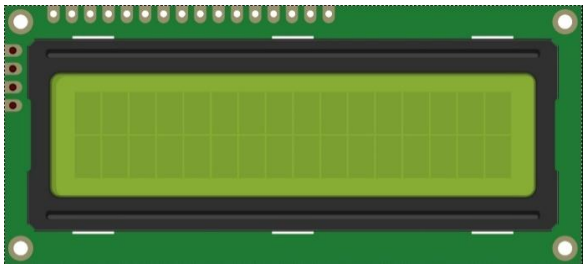
Chapter 20 LCD1602

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

Project 20.1 LCD1602

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

Component List

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

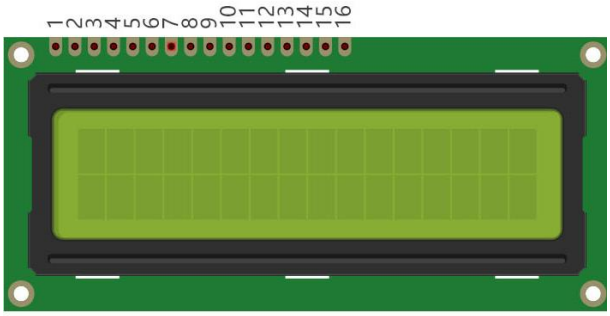
Component knowledge

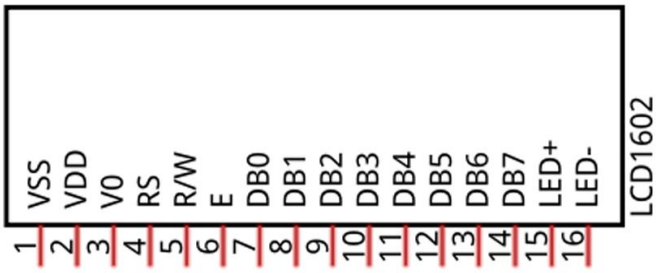
I2C communication

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

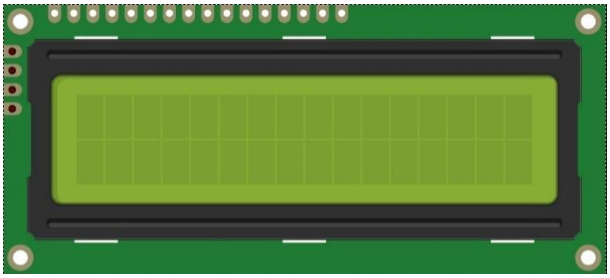
LCD1602 communication'

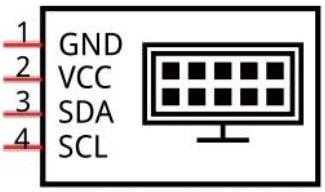
The LCD1602 display screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 display screen along with its circuit pin diagram





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

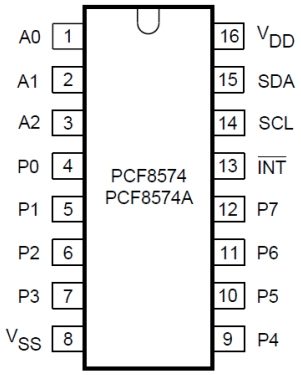
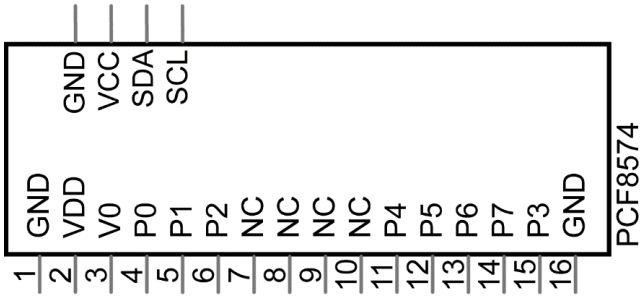




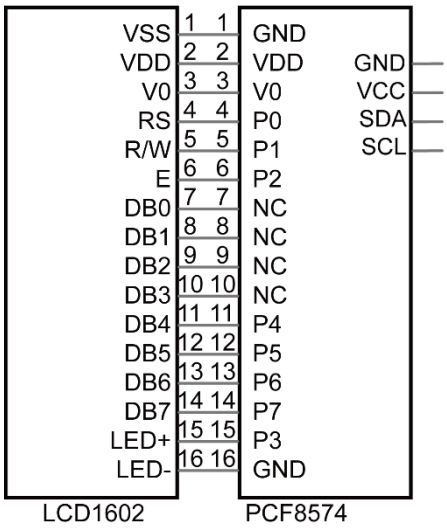
I2C LCD1602 Module

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

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

PCF8574 chip pin diagram:	PCF8574 module pin diagram
	

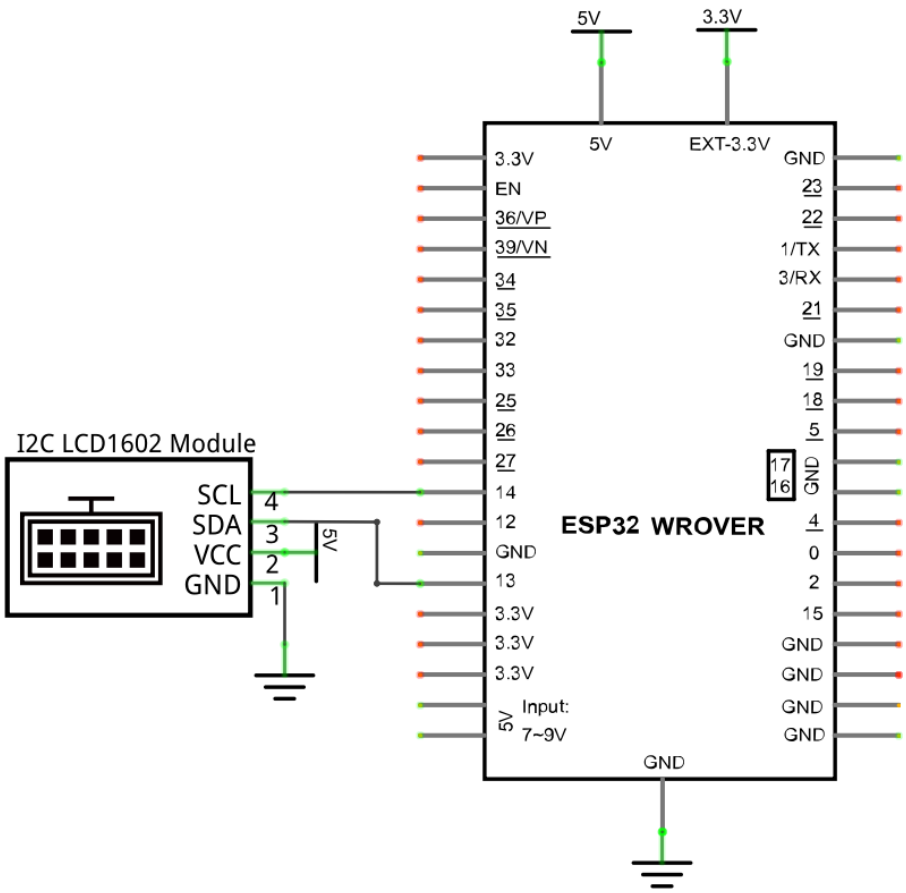
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



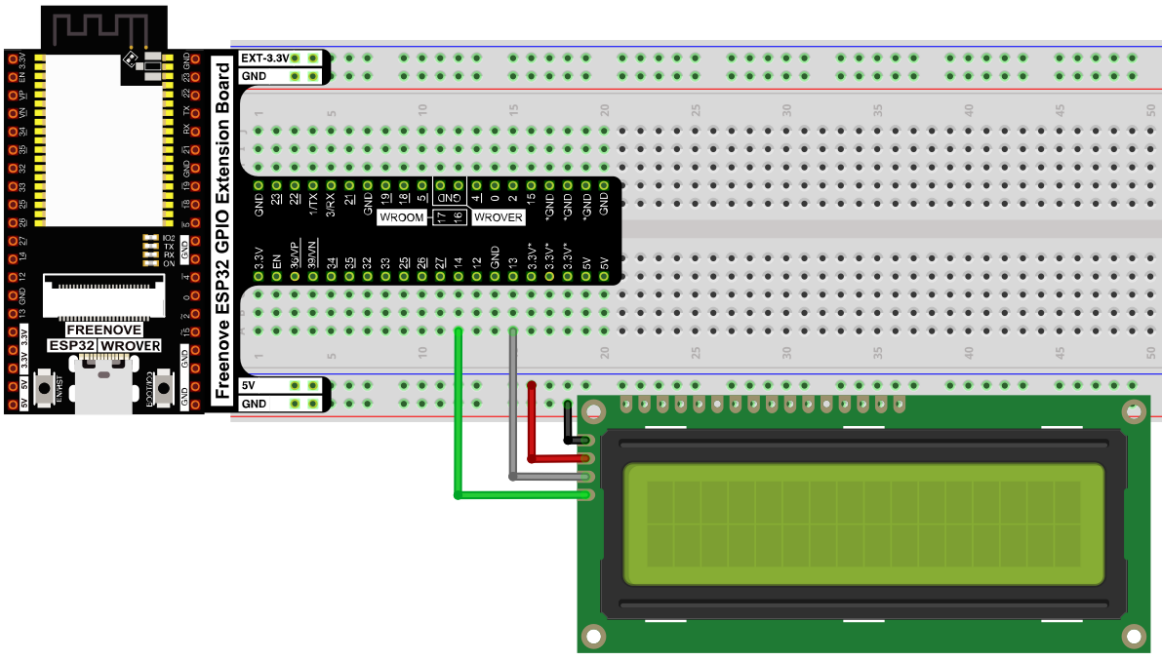
So we only need 4 pins to control the 16 pins of the LCD1602 display screen through the I2C interface. In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



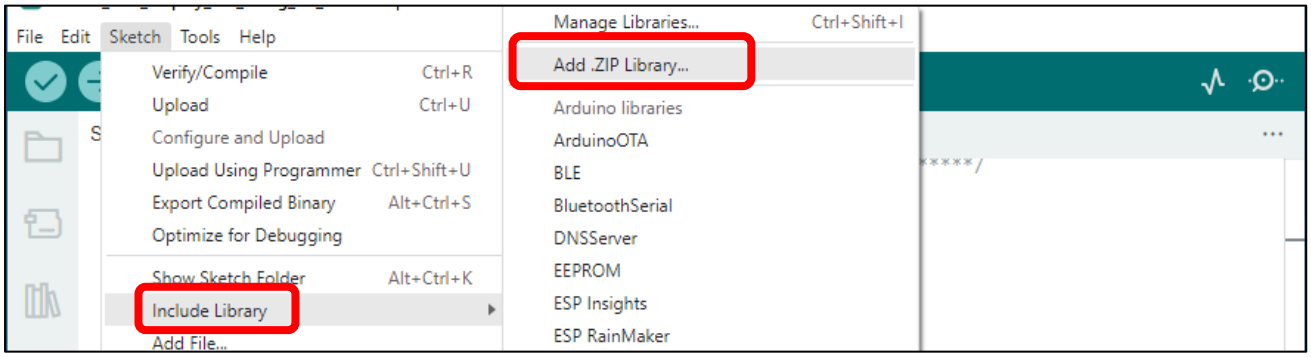
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



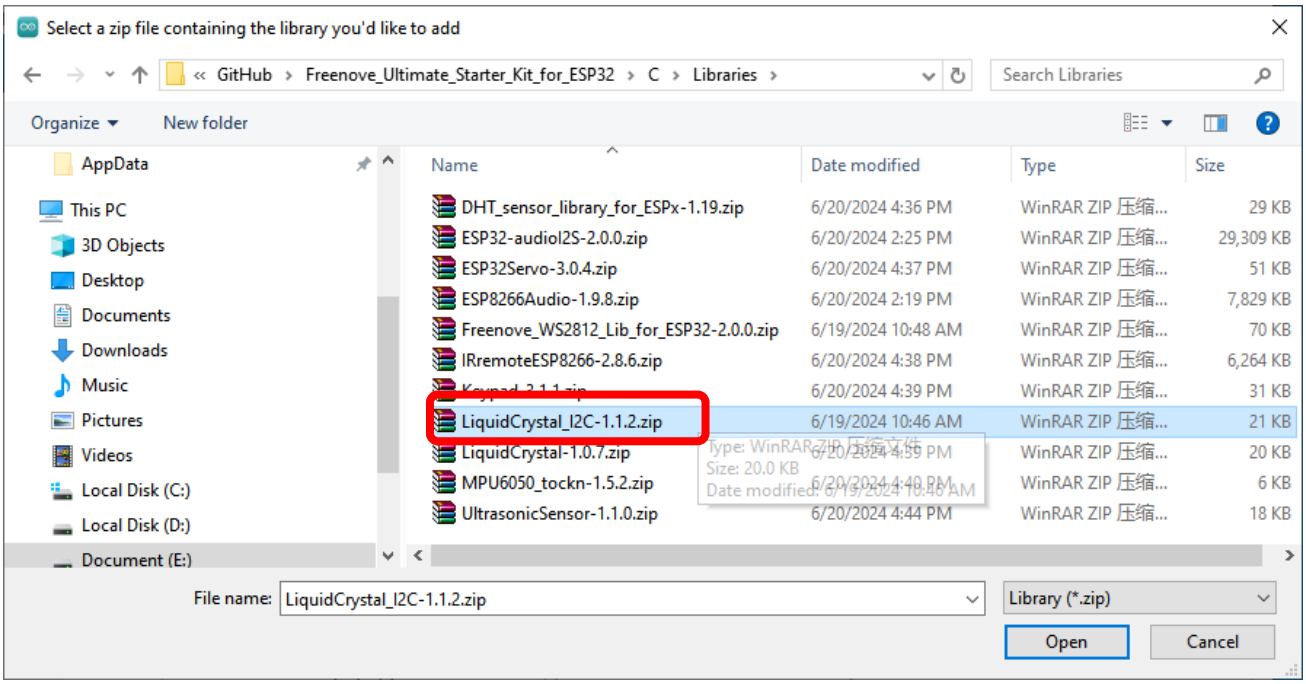
Sketch

How to install the library

We use the third party library LiquidCrystal I2C. If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library->Add .ZIP Library...



In the **Freenove_Ultimate_Starter_Kit_for_ESP32/C/Libraries** folder, select **LiquidCrystal_I2c-1.2.zip** and click open.



Use I2C LCD 1602 to display characters and variables.

Sketch_20.1_Display_the_string_on_LCD1602

File Edit Sketch Tools Help

ESP32 Wrover Module

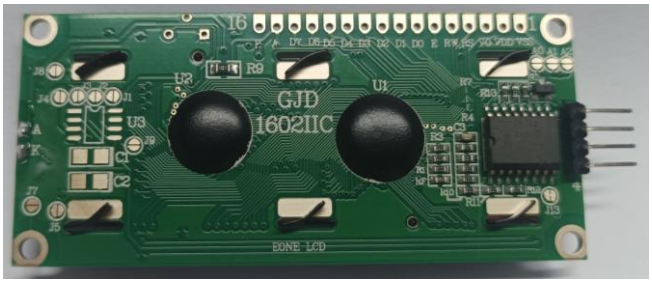
Sketch_20.1_Display_the_string_on_LCD1602.ino

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

#define SDA 13                    //Define SDA pins
#define SCL 14                    //Define SCL pins

/*
 * note:If lcd1602 uses PCF8574T, IIC's address is 0x27,
 *      or lcd1602 uses PCF8574AT, IIC's address is 0x3F.
*/
LiquidCrystal_I2C lcd(0x27,16,2);
void setup() {
  Wire.begin(SDA, SCL);           // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  lcd.setCursor(0,0);             // Move the cursor to row 0, column 0
  lcd.print("hello, world!");     // The print content is displayed on the LCD
}

void loop() {
  lcd.setCursor(0,1);             // Move the cursor to row 1, column 0
  lcd.print("Counter:");          // The count is displayed every second
  lcd.print(millis() / 1000);
  delay(1000);
}

bool i2CAddrTest(uint8_t addr) {
  Wire.beginTransmission(addr);
  if (Wire.endTransmission() == 0) {
    return true;
  }
  return false;
}
```

Compile and upload the code to ESP32-WROVER and the LCD1602 displays characters.

So far, at this writing, we have two types of LCD1602 on sale. One needs to adjust the backlight, and the other does not.

The LCD1602 that does not need to adjust the backlight is shown in the figure below.



The following is the program code:

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

#define SDA 13                    //Define SDA pins
#define SCL 14                    //Define SCL pins

LiquidCrystal_I2C lcd(0x27,16,2);
void setup() {
  Wire.begin(SDA, SCL);           // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  lcd.setCursor(0,0);             // Move the cursor to row 0, column 0
  lcd.print("hello, world!");     // The print content is displayed on the LCD
}

void loop() {
  lcd.setCursor(0,1);             // Move the cursor to row 1, column 0
  lcd.print("Counter:");          // The count is displayed every second
  lcd.print(millis() / 1000);
  delay(1000);
}

bool i2CAddrTest(uint8_t addr) {
  Wire.begin();
  Wire.beginTransmission(addr);
  if (Wire.endTransmission() == 0) {
    return true;
  }
  return false;
}
```

Include header file of Liquid Crystal Display (LCD)1602 and I2C.

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
```

Instantiate the I2C LCD1602 screen. It should be noted here that if your LCD driver chip uses PCF8574T, set the I2C address to 0x27, and if uses PCF8574AT, set the I2C address to 0x3F.

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

Initialize I2C and set its pins as 13,14. And then initialize LCD1602 and turn on the backlight of LCD.

```
  Wire.begin(SDA, SCL);           // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
```

Move the cursor to the first row, first column, and then display the character.

```
  lcd.setCursor(0,0);             // Move the cursor to row 0, column 0
  lcd.print("hello, world! ");      // The print content is displayed on the LCD
```

Print the number on the second line of LCD1602.

```
void loop(){
  lcd.setCursor(0,1);             // Move the cursor to row 1, column 0
  lcd.print("Counter:");          // The count is displayed every second
  lcd.print(millis() / 1000);
  delay(1000);
}
```

Check whether the I2C address exists.

```
bool i2CAddrTest(uint8_t addr) {
  Wire.begin();
  Wire.beginTransmission(addr);
  if (Wire.endTransmission() == 0) {
    return true;
  }
  return false;
}
```

Reference

class **LiquidCrystal**

The LiquidCrystal class can manipulate common LCD screens. The first step is defining an object of LiquidCrystal, for example:

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

Instantiate the Lcd1602 and set the I2C address to 0x27, with 16 columns per row and 2 rows per column.

```
init();
```

Initializes the Lcd1602's device

```
backlight();
```

Turn on Lcd1602's backlight.

```
setCursor(column,row);
```

Sets the screen's column and row.

```
column: The range is 0 to 15.
row: The range is 0 to 1.
```

```
print(String);
```

Print the character string on Lcd1602

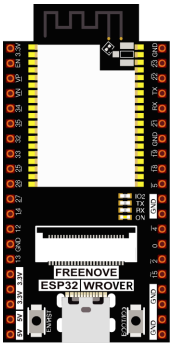
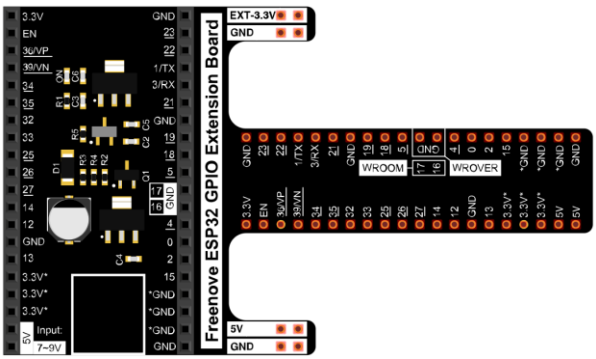
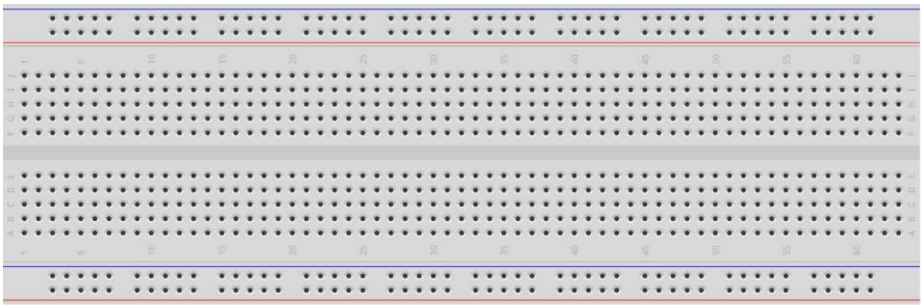
Chapter 21 Ultrasonic Ranging

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

Project 21.1 Ultrasonic Ranging

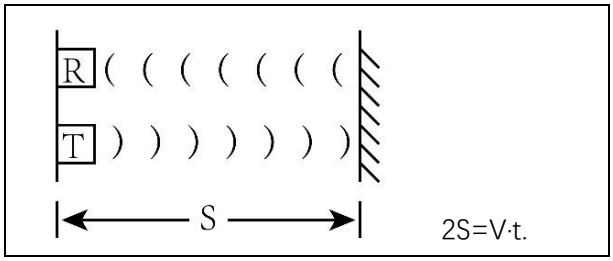
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

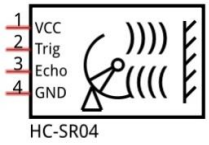
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper F/M x4	HC SR04 x1

Component Knowledge

The ultrasonic ranging module uses the principle that ultrasonic waves will be sent back when encounter obstacles. We can measure the distance by counting the time interval between sending and receiving of the ultrasonic waves, and the time difference is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, about v=340m/s, we can calculate the distance between the ultrasonic ranging module and the obstacle: s=vt/2.



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



Pin description:

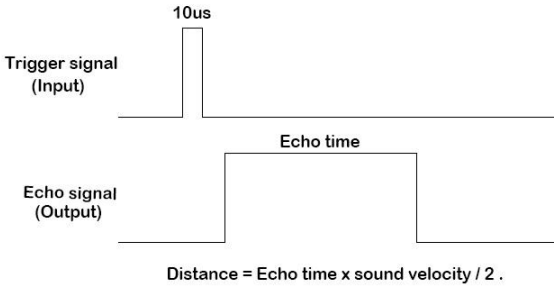
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

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

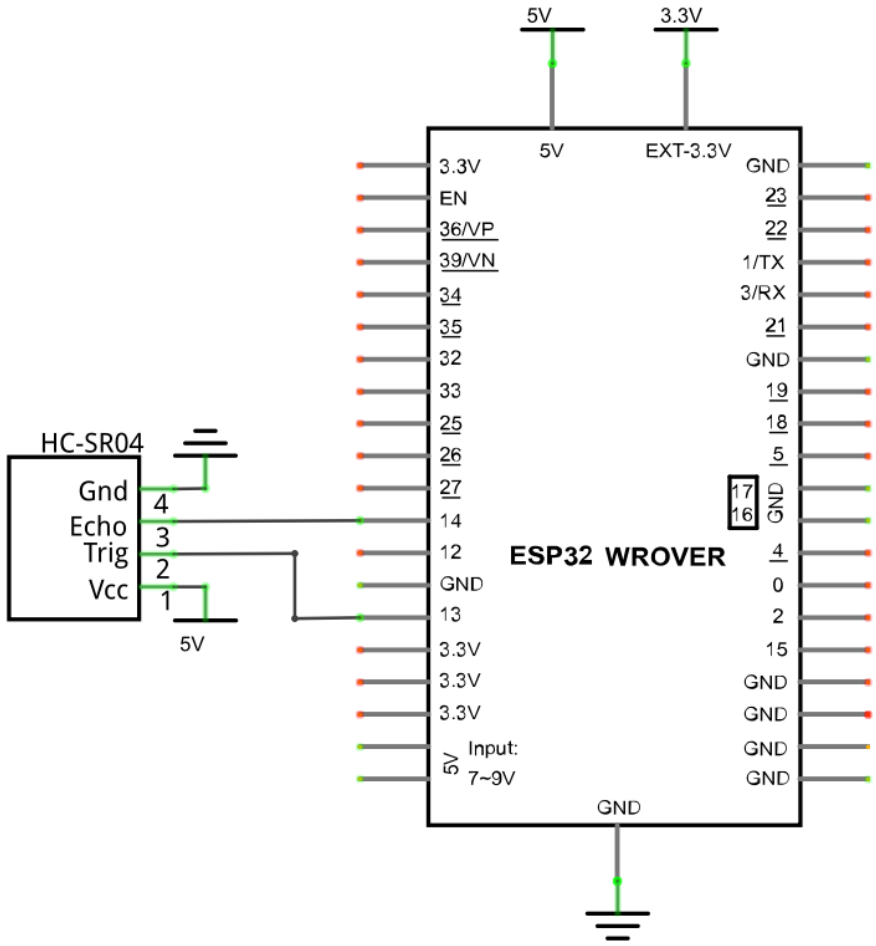
Instructions for use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The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



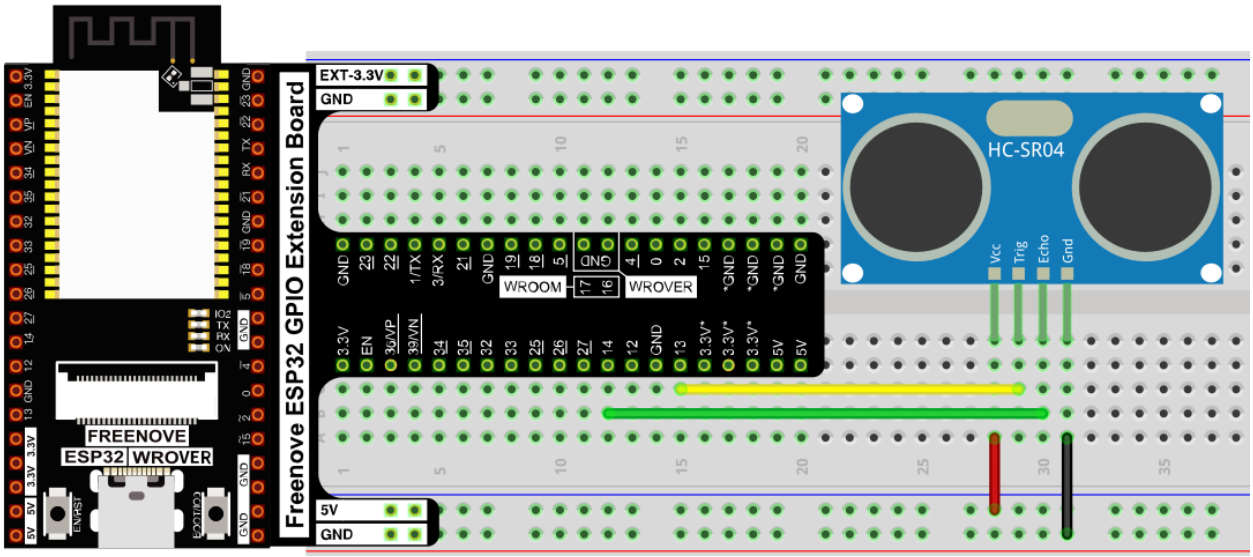
Circuit

Note that the voltage of ultrasonic module is 5V in the circuit.

Schematic diagram



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



Sketch

Sketch_21.1_Ultrasonic_Ranging

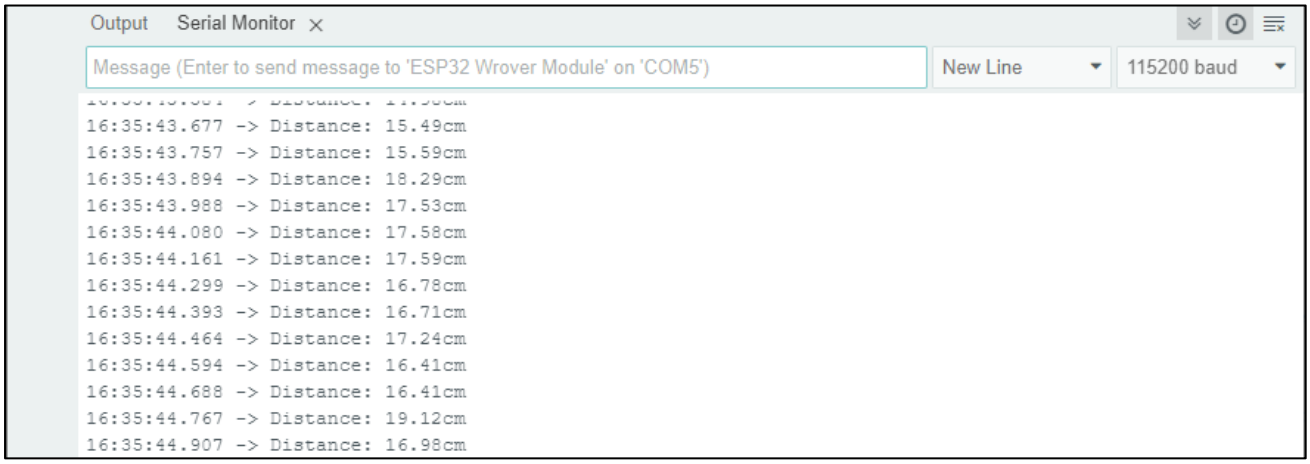
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_21.1_Ultrasonic_Ranging.ino

```
#define trigPin 13 // define TrigPin
#define echoPin 14 // define EchoPin.
#define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s

void setup() {
  pinMode(trigPin,OUTPUT);// set trigPin to output mode
  pinMode(echoPin,INPUT); // set echoPin to input mode
  Serial.begin(115200);   // Open serial monitor at 115200 baud to see ping results.
}

void loop() {
  delay(100); // Wait 100ms between pings (about 20 pings/sec).
  Serial.printf("Distance: ");
  Serial.print(getSonar()); // Send ping, get distance in cm and print result
  Serial.println("cm");
}

float getSonar() {
  unsigned long pingTime;
  float distance;
  // make trigPin output high level lasting for 10μs to triger HC_SR04
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Wait HC-SR04 returning to the high level and measure out this waitting time
  pingTime = pulseIn(echoPin, HIGH, timeOut);
  // calculate the distance according to the time
  distance = (float)pingTime * soundVelocity / 2 / 10000;
  return distance; // return the distance value
}
```

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200 and you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure:



The following is the program code:

```
#define trigPin 13 // define trigPin
#define echoPin 14 // define echoPin.
#define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s

void setup() {
  pinMode(trigPin,OUTPUT);// set trigPin to output mode
  pinMode(echoPin,INPUT); // set echoPin to input mode
  Serial.begin(115200);   // Open serial monitor at 115200 baud to see ping results.
}

void loop() {
  delay(100); // Wait 100ms between pings (about 20 pings/sec).
  Serial.printf("Distance: ");
  Serial.print(getSonar()); // Send ping, get distance in cm and print result
  Serial.println("cm");
}

float getSonar() {
  unsigned long pingTime;
  float distance;
  // make trigPin output high level lasting for 10us to trigger HC_SR04
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Wait HC-SR04 returning to the high level and measure out this waiting time
```

```
  pingTime = pulseIn(echoPin, HIGH, timeOut);
  // calculate the distance according to the time
  distance = (float)pingTime * soundVelocity / 2 / 10000;
  return distance; // return the distance value
}
```

First, define the pins and the maximum measurement distance.

```
#define trigPin 13 // define trigPin
#define echoPin 14 // define echoPin.
#define MAX_DISTANCE 700        //define the maximum measured distance
```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance, that is, time Out. timeOut= 2*MAX_DISTANCE/100/340*1000000. The result of the constant part in this formula is approximately 58.8.

```
float timeOut = MAX_DISTANCE * 60;
```

Subfunction getSonar () function is used to start the ultrasonic module to begin measuring, and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the ultrasonic module. Then use pulseIn () to read the ultrasonic module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
float getSonar() {
  unsigned long pingTime;
  float distance;
  // make trigPin output high level lasting for 10μs to triger HC_SR04?
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Wait HC-SR04 returning to the high level and measure out this waitting time
  pingTime = pulseIn(echoPin, HIGH, timeOut);
  // calculate the distance according to the time
  distance = (float)pingTime * soundVelocity / 2 / 10000;
  return distance; // return the distance value
}
```

Lastly, in loop() function, get the measurement distance and display it continually.

```
void loop() {
  delay(100); // Wait 100ms between pings (about 20 pings/sec).
  Serial.printf("Distance: ");
  Serial.print(getSonar()); // Send ping, get distance in cm and print result
  Serial.println("cm");
}
```

About function **pulseIn**():

int pulseIn(int pin, int level, int timeout);
pin: the number of the Arduino pin on which you want to read the pulse. Allowed data types: int. **value**: type of pulse to read: either HIGH or LOW. Allowed data types: int. **timeout** (optional): the number of microseconds to wait for the pulse to start; default is one second.

Project 21.2 Ultrasonic Ranging

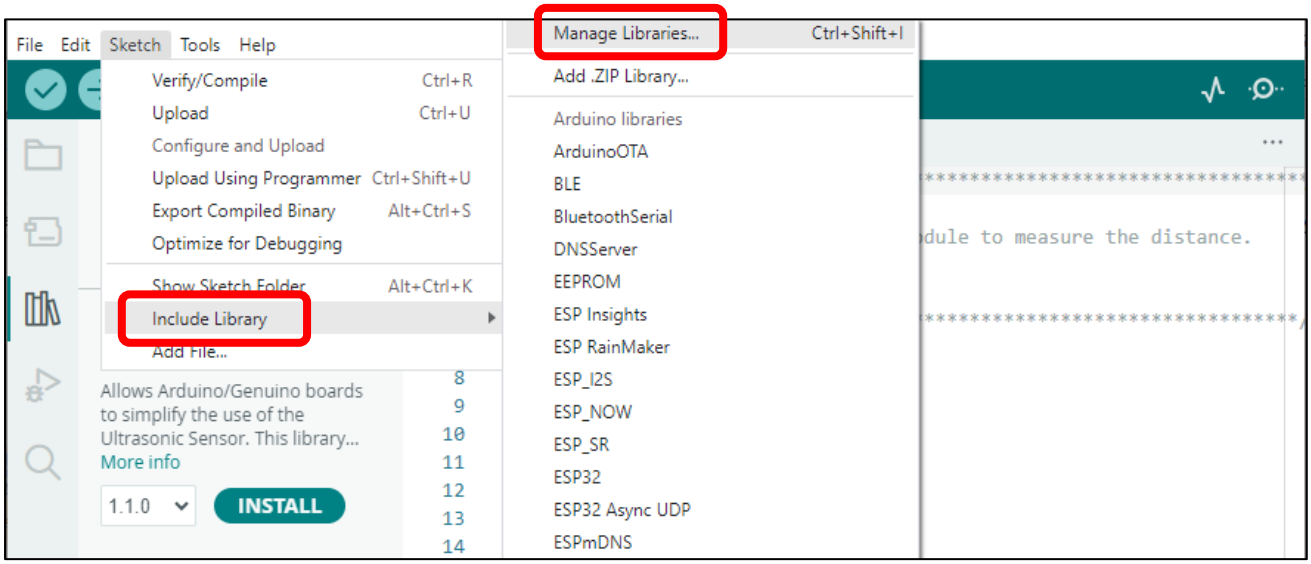
Component List and Circuit

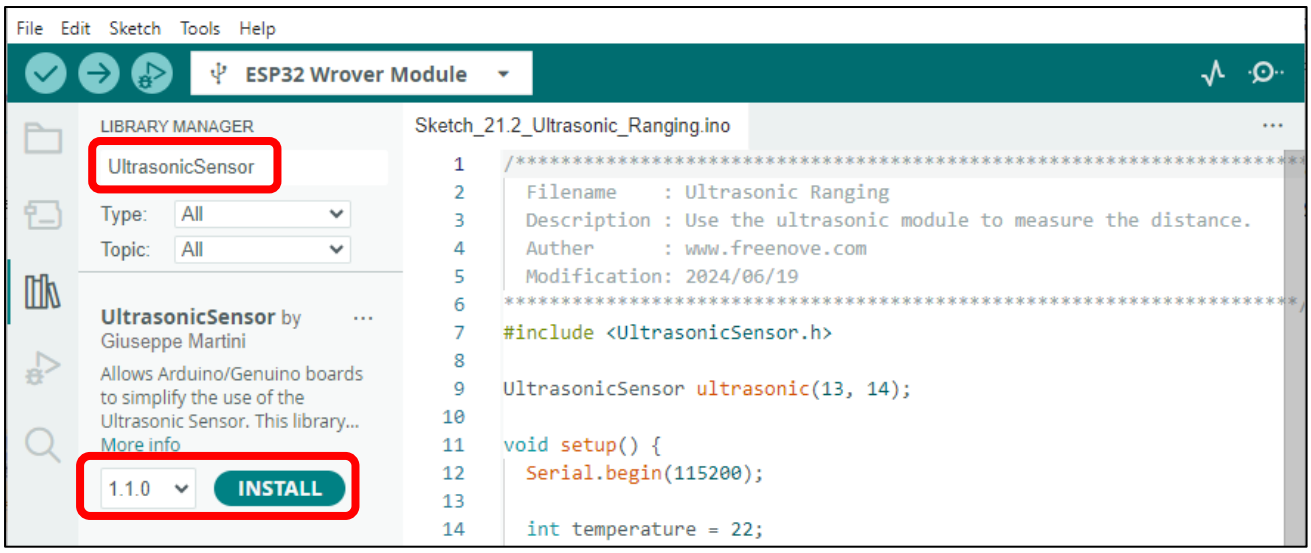
Component List and Circuit are the same as the previous section.

Sketch

How to install the library

We use the third party library UltrasonicSensor. If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter "UltrasonicSensor" in the search bar and select "UltrasonicSensor" for installation.
Refer to the following operations:





Sketch_21.2_Ultrasonic_Ranging

```
#include <UltrasonicSensor.h>

UltrasonicSensor ultrasonic(13, 14);

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

  int temperature = 22;
  ultrasonic.setTemperature(temperature);
}

void loop() {
  int distance = ultrasonic.distanceInCentimeters();
  Serial.printf("Distance: %dcm\n",distance);
  delay(300);
}
```

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200. Use the ultrasonic module to measure distance. As shown in the following figure:

```
16:35:43.677 -> Distance: 15.49cm
16:35:43.757 -> Distance: 15.59cm
16:35:43.894 -> Distance: 18.29cm
16:35:43.988 -> Distance: 17.53cm
16:35:44.080 -> Distance: 17.58cm
16:35:44.161 -> Distance: 17.59cm
16:35:44.299 -> Distance: 16.78cm
16:35:44.393 -> Distance: 16.71cm
16:35:44.464 -> Distance: 17.24cm
16:35:44.594 -> Distance: 16.41cm
16:35:44.688 -> Distance: 16.41cm
16:35:44.767 -> Distance: 19.12cm
16:35:44.907 -> Distance: 16.98cm
```

The following is the program code:

```
#include <UltrasonicSensor.h>
//Attach the trigger and echo pins to pins 13 and 14 of esp32
UltrasonicSensor ultrasonic(13, 14);

void setup() {
  Serial.begin(115200);
  //set the speed of sound propagation according to the temperature to reduce errors
  int temperature = 22; //Setting ambient temperature
  ultrasonic.setTemperature(temperature);
}

void loop() {
  int distance = ultrasonic.distanceInCentimeters();
  Serial.printf("Distance: %dcm\n",distance);
  delay(300);
}
```

First, add UltrasonicSensor library.

```
1  #include <UltrasonicSensor.h>
```

Define an ultrasonic object and associate the pins.

```
3  UltrasonicSensor ultrasonic(13, 14);
```

Set the ambient temperature to make the module measure more accurately.

```
9  ultrasonic.setTemperature(temperature);
```

Use the distanceInCentimeters function to get the distance measured by the ultrasound and print it out through the serial port.

```
16 void loop() {
17   int distance = ultrasonic.distanceInCentimeters();
18   Serial.printf("Distance: %dcm\n",distance);
19   delay(300);
20 }
```

Reference

class UltrasonicSensor

class UltrasonicSensor must be instantiated when used, that is, define an object of Servo type, for example:

UltrasonicSensor ultrasonic(13, 14);

setTemperature(value): The speed of sound propagation is different at different temperatures. In order to get more accurate data, this function needs to be called. **value** is the temperature value of the current environment.

`distanceInCentimeters()`: The ultrasonic distance acquisition function returns the value in centimeters.

distanceInMillimeters(): The ultrasonic distance acquisition function returns the value in millimeter.

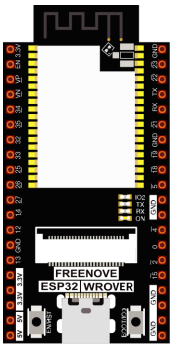
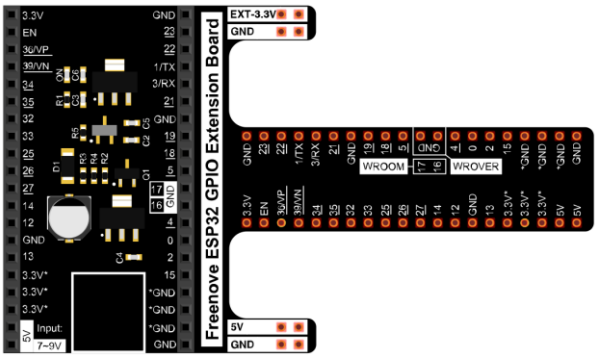
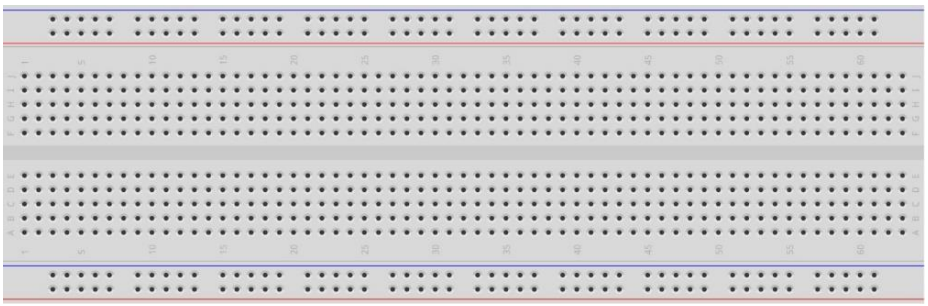
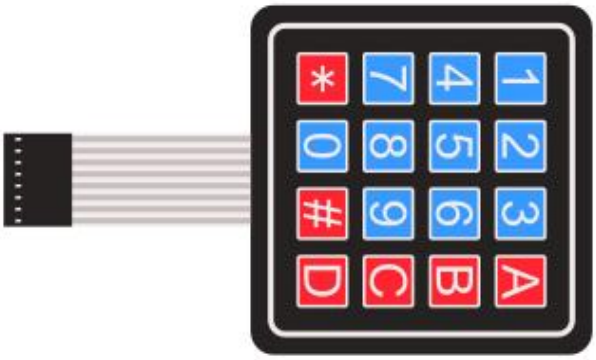
Chapter 22 Matrix Keypad

Earlier we learned about a single push button switch. In this chapter, we will learn about matrix keyboards, which integrates a number of push button switches as keys for the purposes of input.

Project 22.1 Matrix Keypad

In this project, we will attempt to get every key code on the matrix keypad to work.

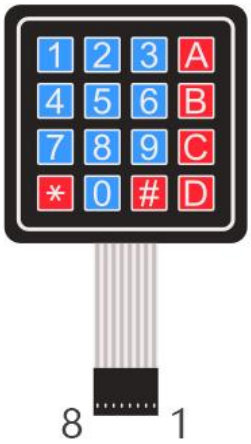
Component List

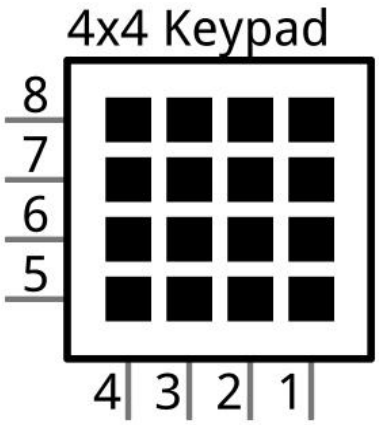
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper M/M x8	4x4 Matrix Keypad x1 

Component knowledge

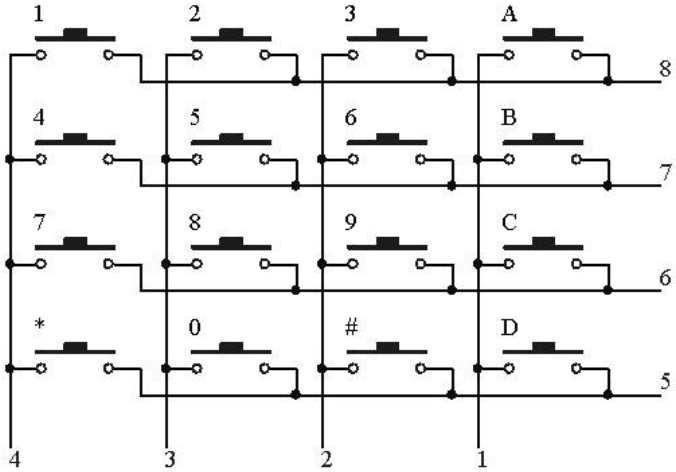
4x4 Matrix Keypad

A keypad matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 keypad matrix integrates 16 keys:





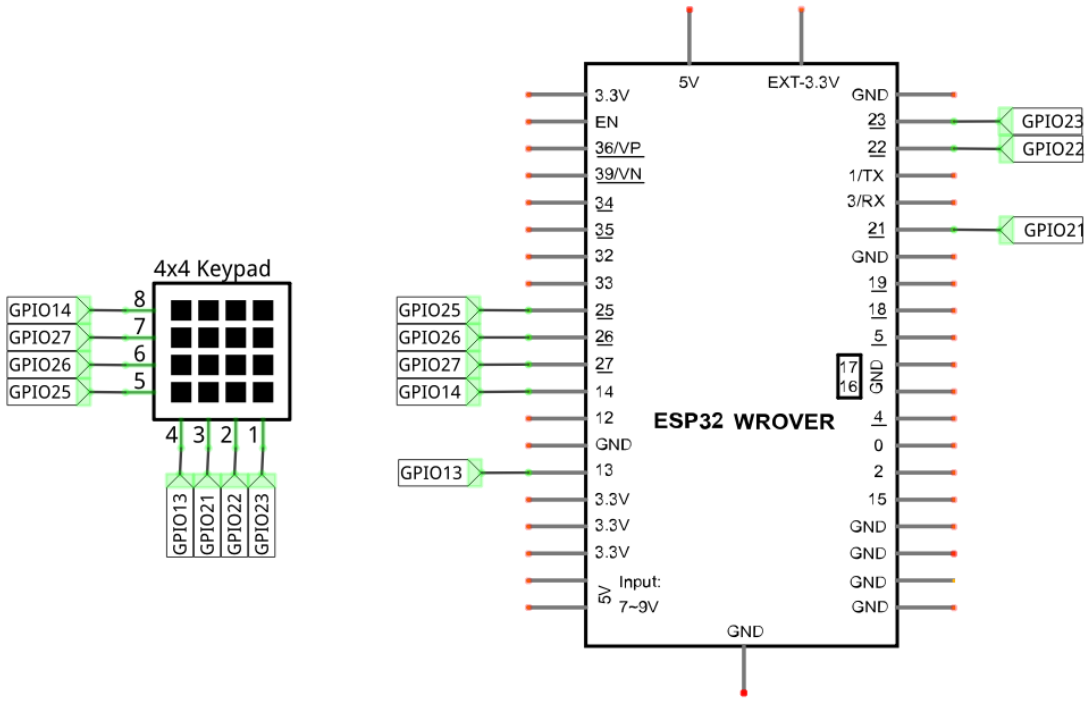
Similar to the integration of a LED matrix, the 4x4 keypad matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



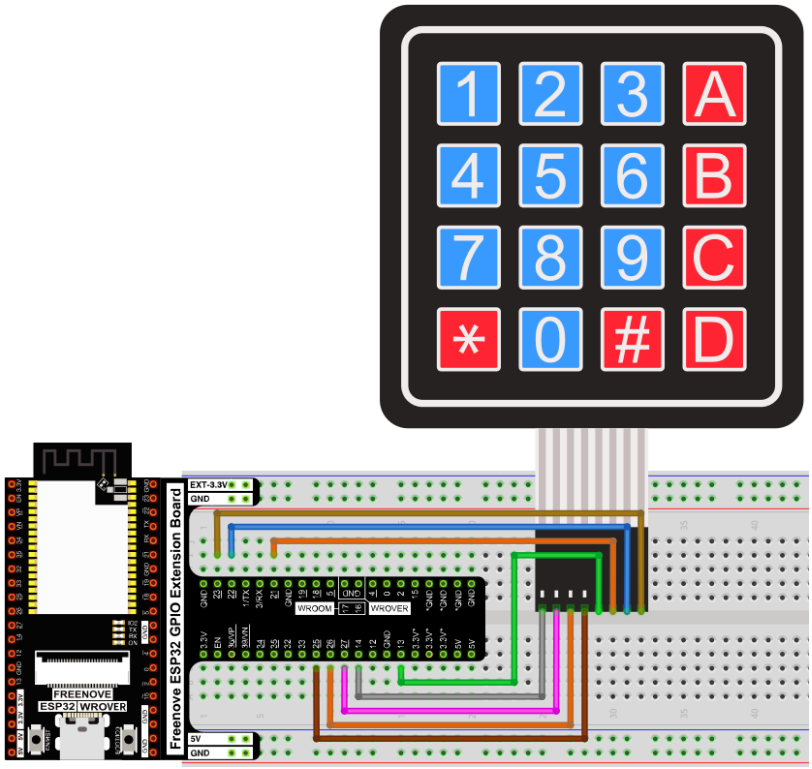
The usage is similar to the LED matrix, using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Therefore, you can get the state of all of the keys.

Circuit

Schematic diagram



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



Sketch

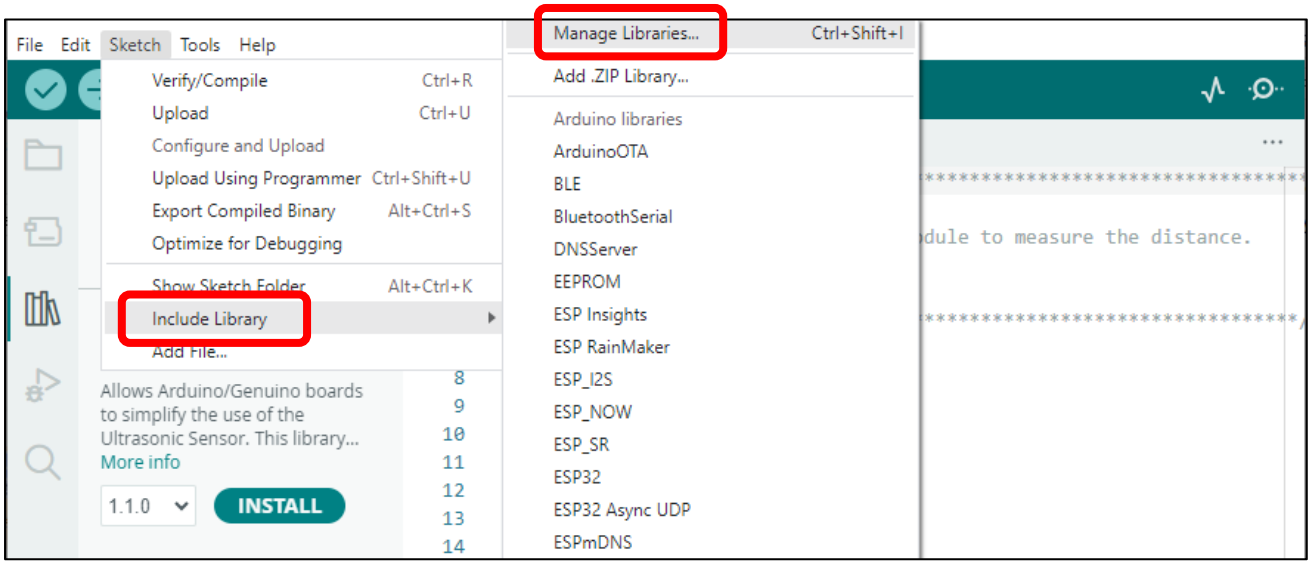
This code is used to obtain all key codes of the 4x4 matrix keypad, when one of the keys is pressed, the key code will be printed out via serial port.

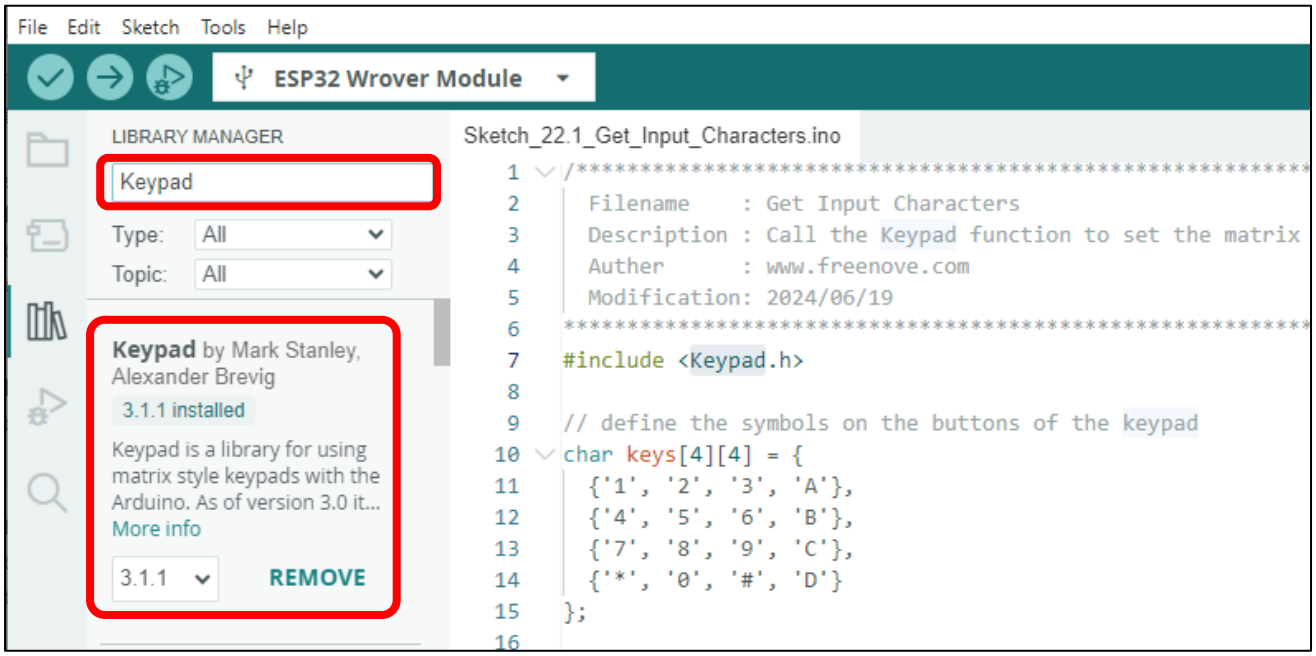
How to install the library

We use the third party library **Keypad**. If you haven't installed it yet, please do so before learning.

The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter " Keypad" in the search bar and select "Keypad" for installation.

Refer to the following operations:





Sketch_22.1_Get_Input_Characters

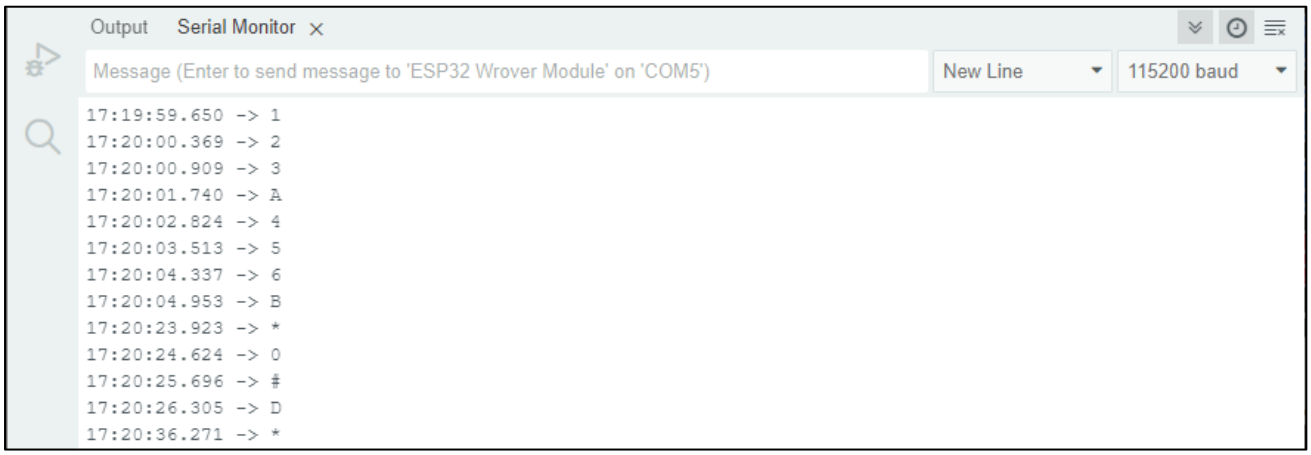
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_22.1_Get_Input_Characters.ino

```
#include <Keypad.h>

// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[4] = {14, 27, 26, 25}; // connect to the row pinouts of the keypad
byte colPins[4] = {13, 21, 22, 23};   // connect to the column pinouts of the keypad

// initialize an instance of class NewKeypad
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);

void setup() {
  Serial.begin(115200); // Initialize the serial port and set the baud rate to 115200
  Serial.println("ESP32 is ready!");  // Print the string "UNO is ready!"
}

void loop() {
  // Get the character input
  char keyPressed = myKeypad.getKey();
  // If there is a character input, sent it to the serial port
  if (keyPressed) {
    Serial.println(keyPressed);
  }
}
```

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200, press the keyboard, the value of the pressed keys will be printed out via the serial port. As shown in the following figure:



The following is the program code:

```
#include <Keypad.h>
// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[4] = {14, 27, 26, 25}; // connect to the row pinouts of the keypad
byte colPins[4] = {13, 21, 22, 23};   // connect to the column pinouts of the keypad

// initialize an instance of class NewKeypad
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);

void setup() {
  Serial.begin(115200); // Initialize the serial port and set the baud rate to 115200
  Serial.println("ESP32 is ready! "); // Print the string "ESP32 is ready! "
}

void loop() {
  // Get the character input
  char keyPressed = myKeypad.getKey();
  // If there is a character input, sent it to the serial port
  if (keyPressed) {
    Serial.println(keyPressed);
  }
}
```

First, add header file, define 4*4 matrix keyboard key value and the matrix keyboard pin.

```
#include <Keypad.h>
// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[4] = {14, 27, 26, 25}; // connect to the row pinouts of the keypad
byte colPins[4] = {13, 21, 22, 23};   // connect to the column pinouts of the keypad
```

Second, define a matrix keyboard object and associate the keys and pins with it.

```
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
```

Finally, get the key value and print it out via the serial port.

```
void loop() {
  // Get the character input
  char keyPressed = myKeypad.getKey();
  // If there is a character input, sent it to the serial port
  if (keyPressed) {
    Serial.println(keyPressed);
  }
}
```

Reference

class Keypad You need to add the library each time you use the Keypad.

Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols);

Constructor, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.

char getKey();

Get the key code of the pressed key. If no key is pressed, the return value is NULL.

void setDebounceTime(uint);

Set the debounce time with a default time of 10ms.

void setHoldTime(uint);

Set the duration for the key to keep stable state after pressed.

bool isPressed(char keyChar);

Judge whether the key with code "keyChar" is pressed.

char waitForKey();

Wait for a key to be pressed, and return key code of the pressed key.

KeyState getState();

Get the state of the keys.

bool keyStateChanged();

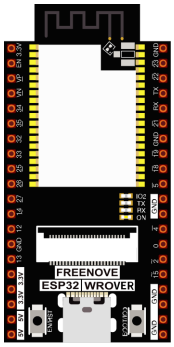
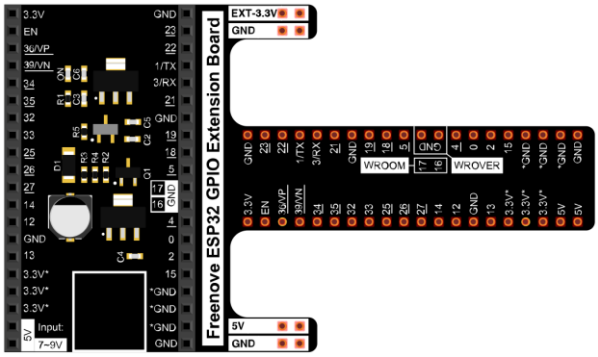
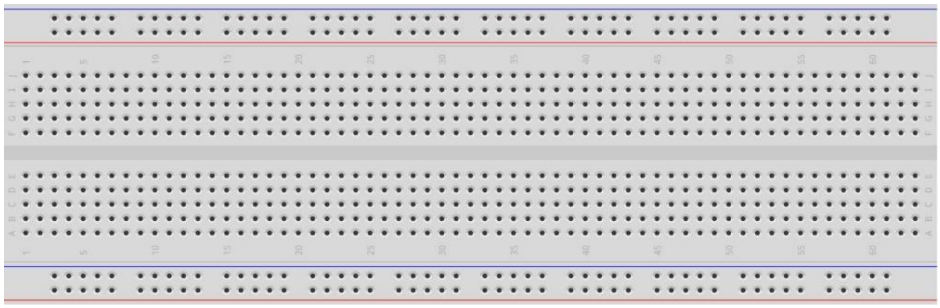
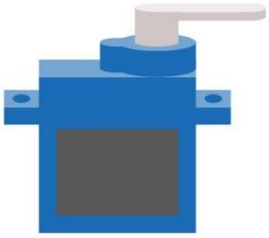
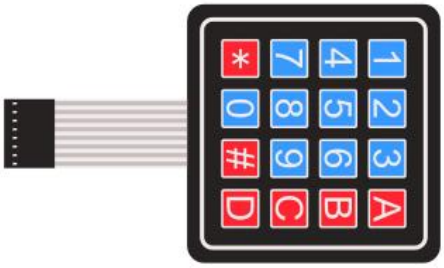
Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad

Project 22.2 Keypad Door

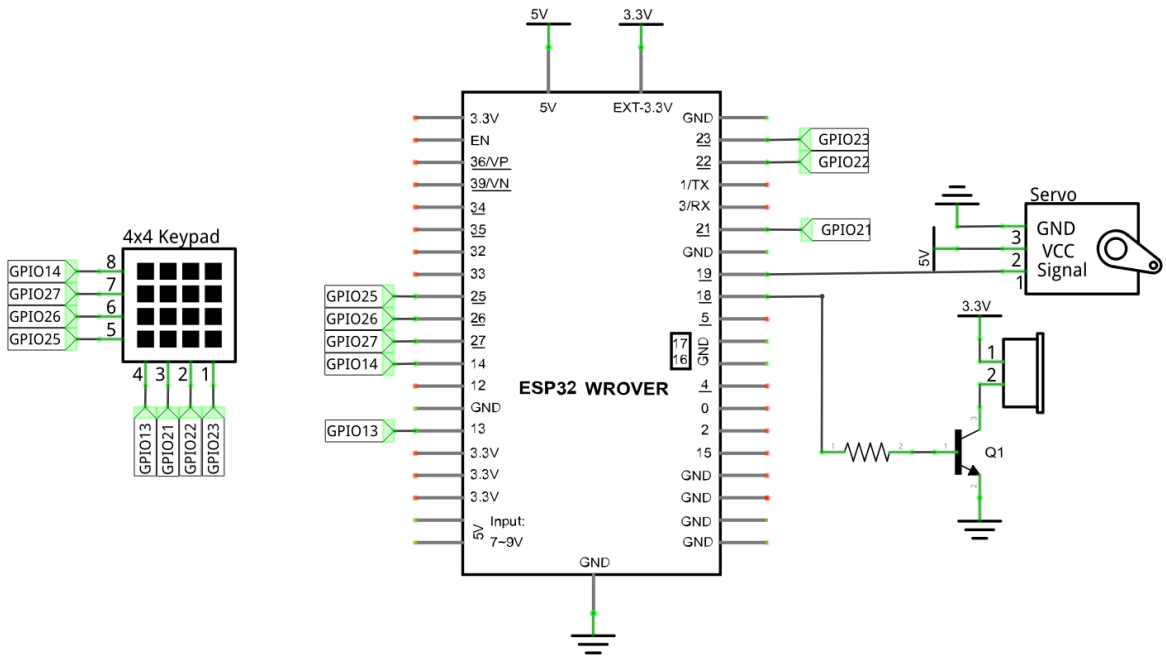
In this project, we use keypad as a keyboard to control the action of the servo motor.

Component List

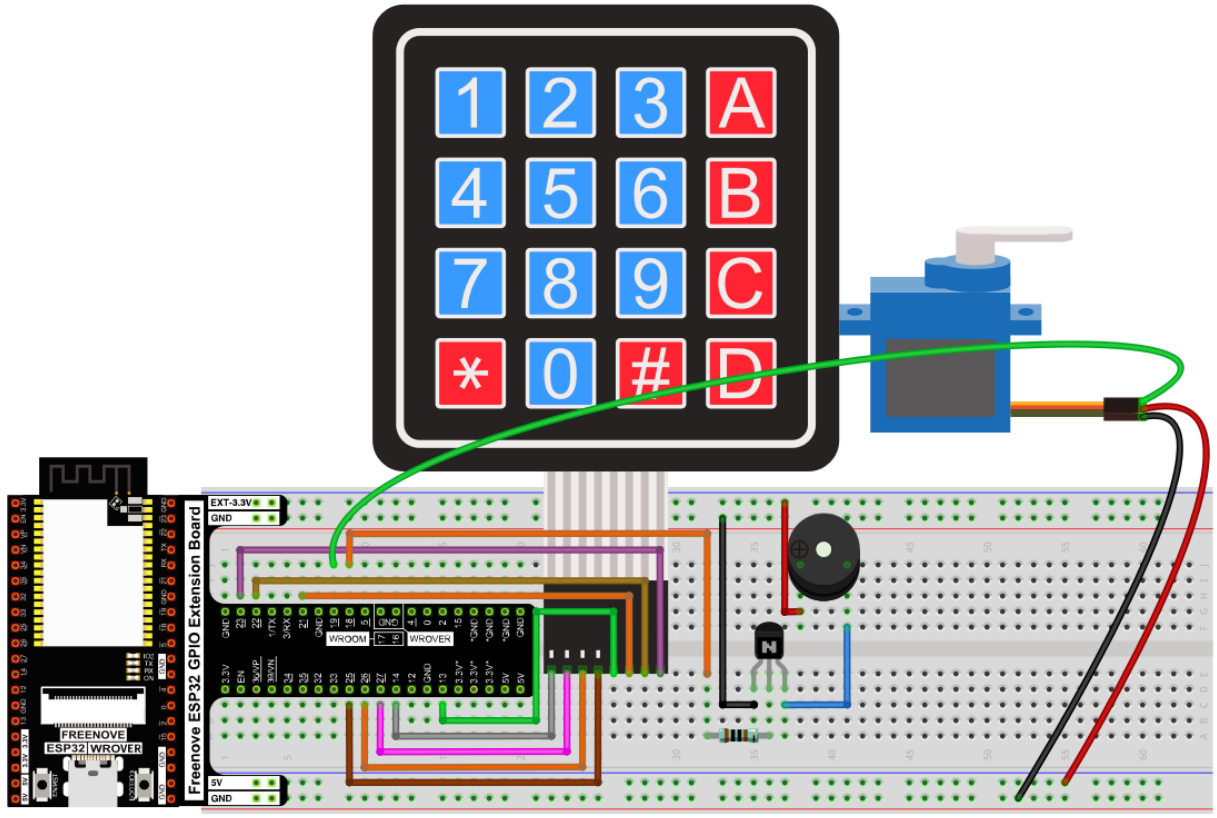
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M	Servo x1 	4x4 Matrix Keypad x1 
NPN transistorx1 (S8050)	Active buzzer x1	Resistor 1kΩ x1

Circuit

Schematic diagram



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



Sketch

Sketch_22.2_Keypad_Door

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_22.2_Combination_Lock.ino

```
void loop() {
  static char keyIn[4];     // Save the input character
  static byte keyInNum = 0; // Save the the number of input characters
  char keyPressed = myKeypad.getKey();  // Get the character input
  // Handle the input characters
  if (keyPressed) {
    // Make a prompt tone each time press the key
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    // Save the input characters
    keyIn[keyInNum++] = keyPressed;
    // Judge the correctness after input
    if (keyInNum == 4) {
      bool isRight = true;            // Save password is correct or not
      for (int i = 0; i < 4; i++) {   // Judge each character of the password is correct or not
        if (keyIn[i] != passWord[i])
          isRight = false;            // Mark wrong passageword if there is any wrong character.
      }
      if (isRight) {                  // If the input password is right
        myservo.write(90);            // Open the switch
        delay(2000);                  // Delay a period of time
        myservo.write(0);             // Close the switch
        Serial.println("passWord right!");
      }
      else {                          // If the input password is wrong
        digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
        delay(1000);
        digitalWrite(buzzerPin, LOW);
        Serial.println("passWord error!");
      }
      keyInNum = 0; // Reset the number of the input characters to 0
    }
  }
}
```

Verify and upload the code to the ESP32-WROVER and press the keypad to input password with 4 characters. If the input is correct, the servo will move to a certain degree, then return to the original position. If the input is wrong, an input error alarm will be generated.

The following is the program code:

```
#include <Keypad.h>
#include <ESP32Servo.h>

// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[4] = {14, 27, 26, 25};   // connect to the row pinouts of the keypad
byte colPins[4] = {13, 21, 22, 23};   // connect to the column pinouts of the keypad

// initialize an instance of class NewKeypad
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);

Servo  myservo;     // Create servo object to control a servo
int servoPin = 19;  // Define the servo pin
int buzzerPin = 18; // Define the buzzer pin

char passWord[] = {"1234"}; // Save the correct password

void setup() {
  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo object
                                        // set the high level time range of the servo motor
for an accurate 0°-180° sweep
  myservo.write(0);                    // Set the starting position of the servo motor
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  static char keyIn[4];     // Save the input character
  static byte keyInNum = 0; // Save the number of input characters
  char keyPressed = myKeypad.getKey();  // Get the character input
  // Handle the input characters
  if (keyPressed) {
    // Make a prompt tone each time press the key
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
```

```
    // Save the input characters
    keyIn[keyInNum++] = keyPressed;
    // Judge the correctness after input
    if (keyInNum == 4) {
      bool isRight = true;            // Save password is correct or not
      for (int i = 0; i < 4; i++) {  // Judge each character of the password is correct or not
        if (keyIn[i] ! = passWord[i])
          isRight = false;            // Mark wrong password if there is any wrong character.
      }
      if (isRight) {                  // If the input password is right
        myservo.write(90);            // Open the switch
        delay(2000);                  // Delay a period of time
        myservo.write(0);             // Close the switch
        Serial.println("passWord right! ");
      }
      else {                          // If the input password is wrong
        digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
        delay(1000);
        digitalWrite(buzzerPin, LOW);
        Serial.println("passWord error! ");
      }
      keyInNum = 0; // Reset the number of the input characters to 0
    }
  }
}
```

First, we need to set the value of the password.

```
char passWord[] = {"1234"}; // Save the correct password
```

Second, each time the key is pressed, the buzzer makes a short sound and stores the key value entered.

```
  char keyPressed = myKeypad.getKey();  // Get the character input
  // Handle the input characters
  if (keyPressed) {
    // Make a prompt tone each time press the key
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    // Save the input characters
    keyIn[keyInNum++] = keyPressed;
```

Third, if the button has been pressed for four times, ESP32 begins to judge if the password is correct.

```
    if (keyInNum == 4) {
      bool isRight = true;            // Save password is correct or not
      for (int i = 0; i < 4; i++) {   // Judge each character of the password is correct or not
        if (keyIn[i] ! = passWord[i])
          isRight = false;            // Mark wrong password if there is any wrong character.
      }
```

If the password is correct, control the servo motor to open the lock and wait for 2 seconds before closing the lock. If it is not correct, the buzzer makes a long sound and prints the error message through the serial port.

```
      if (isRight) {                  // If the input password is right
        myservo.write(90);           // Open the switch
        delay(2000);                  // Delay a period of time
        myservo.write(0);            // Close the switch
        Serial.println("passWord right! ");
      }
      else {                          // If the input password is wrong
        digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
        delay(1000);
        digitalWrite(buzzerPin, LOW);
        Serial.println("passWord error! ");
      }
```

Finally, remember to empty the keyInNum every time.

```
keyInNum = 0; // Reset the number of the input characters to 0
```

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad .

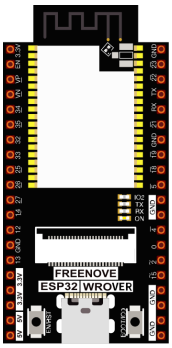
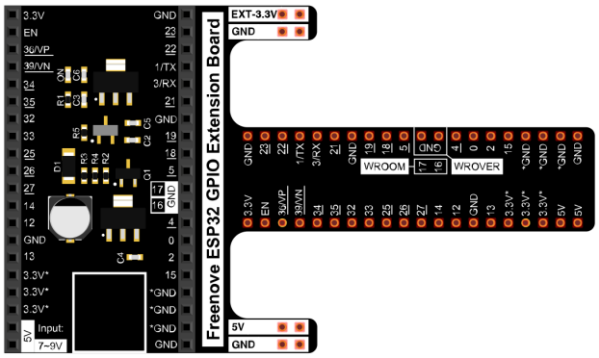
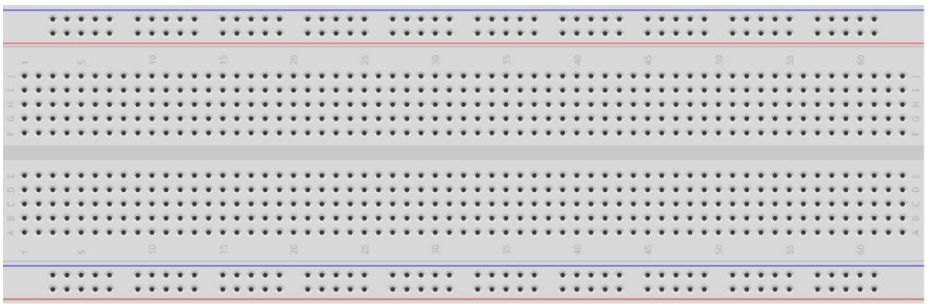
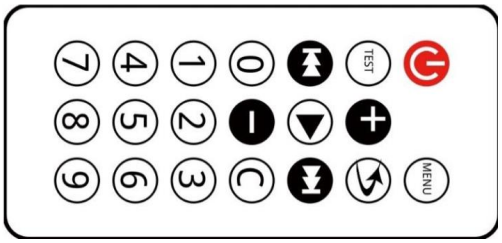
Chapter 23 Infrared Remote

In this chapter, we'll learn how to use an infrared remote control, and control a LED.

Project 23.1 Infrared Remote Control

First, we need to understand how infrared remote control works, then get the command sent from infrared remote control.

Component List

ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper M/M x4	Infrared Remote x1 (May need CR2025 battery x1, please check the holder) 
Infrared Remote x1	Resistor 10kΩ x1

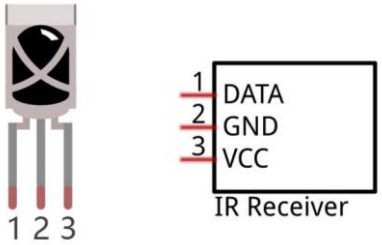
Component knowledge

Infrared Remote

An infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared ray with different command. Infrared remote control technology is widely used in electronic products such as TV, air conditioning, etc. Thus making it possible for you to switch TV programs and adjust the temperature of the air conditioning when away from them. The remote control we use is shown below:

Infrared receiver

An infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.



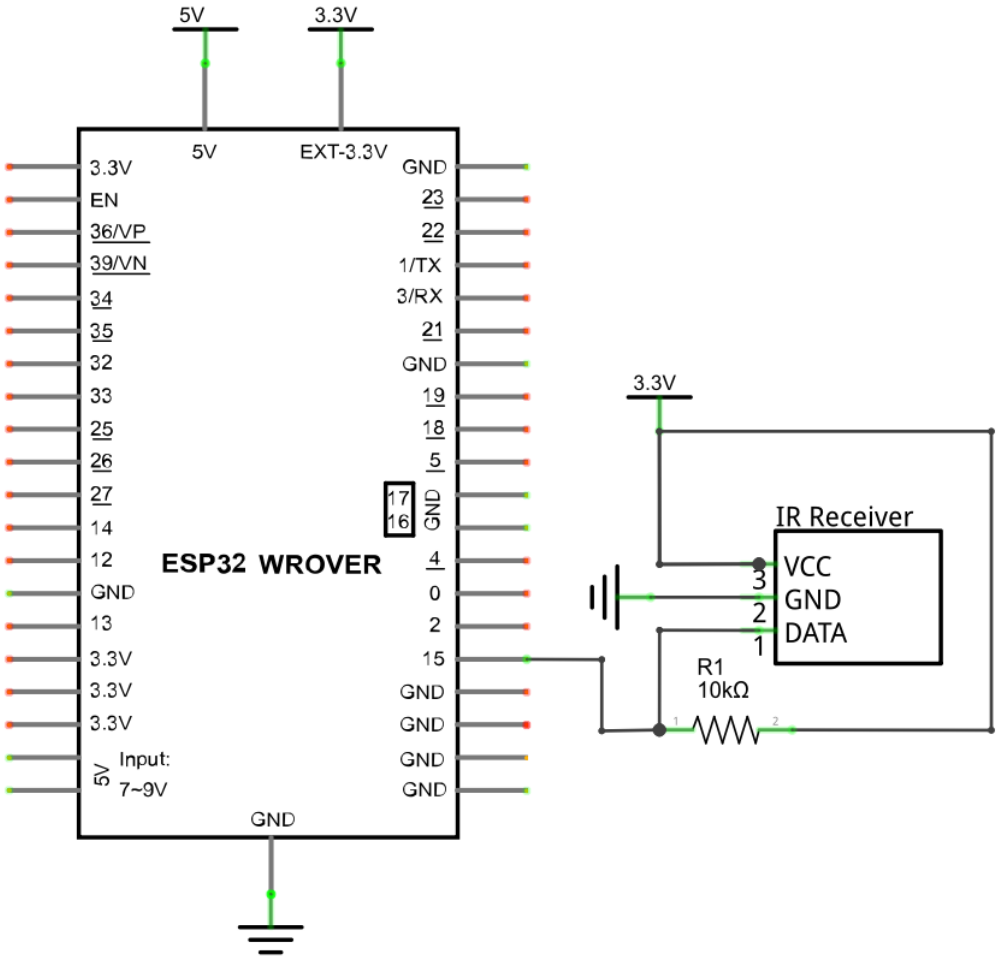
When you use the infrared remote control, the infrared remote control sends a key value to the receiving circuit according to the pressed keys. We can program the ESP32-WROVER to do things like lighting, when a key value is received.

The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

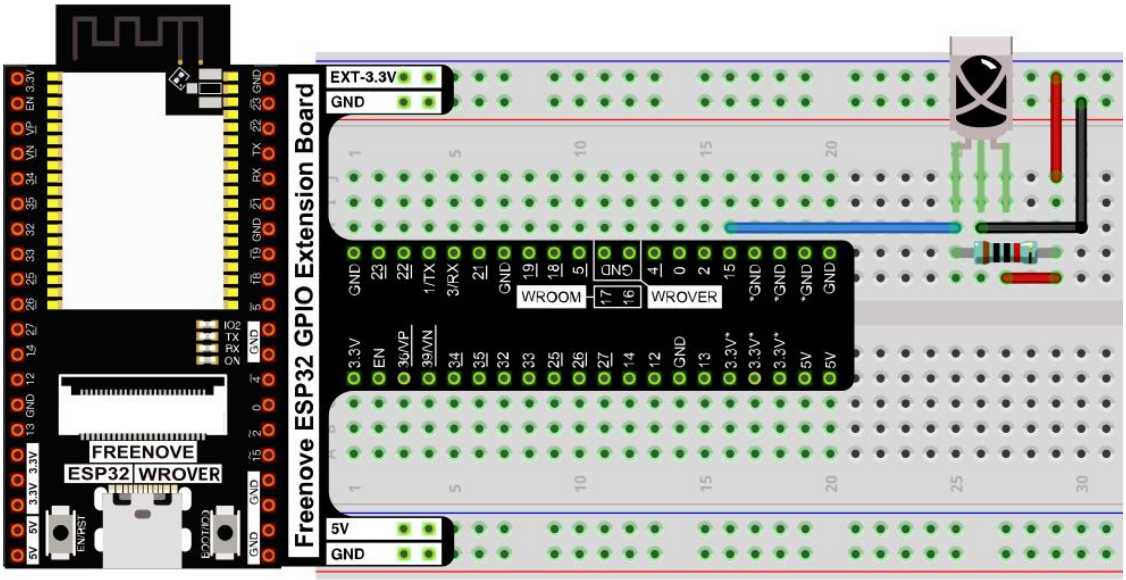
ICON	KEY Value	ICON	KEY Value
	FFA25D	C	FFB04F
MENU	FFE21D	1	FF30CF
TEST	FF22DD	2	FF18E7
+	FF02FD	3	FF7A85
	FFC23D	4	FF10EF
	FFE01F	5	FF38C7
	FFA857	6	FF5AA5
	FF906F	7	FF42BD
0	FF6897	8	FF4AB5
−	FF9867	9	FF52AD

Circuit

Schematic diagram



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

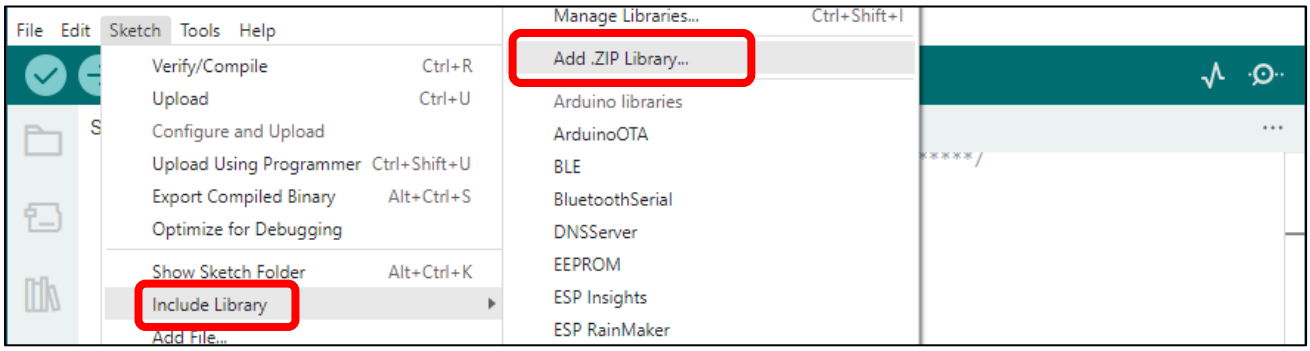


Sketch

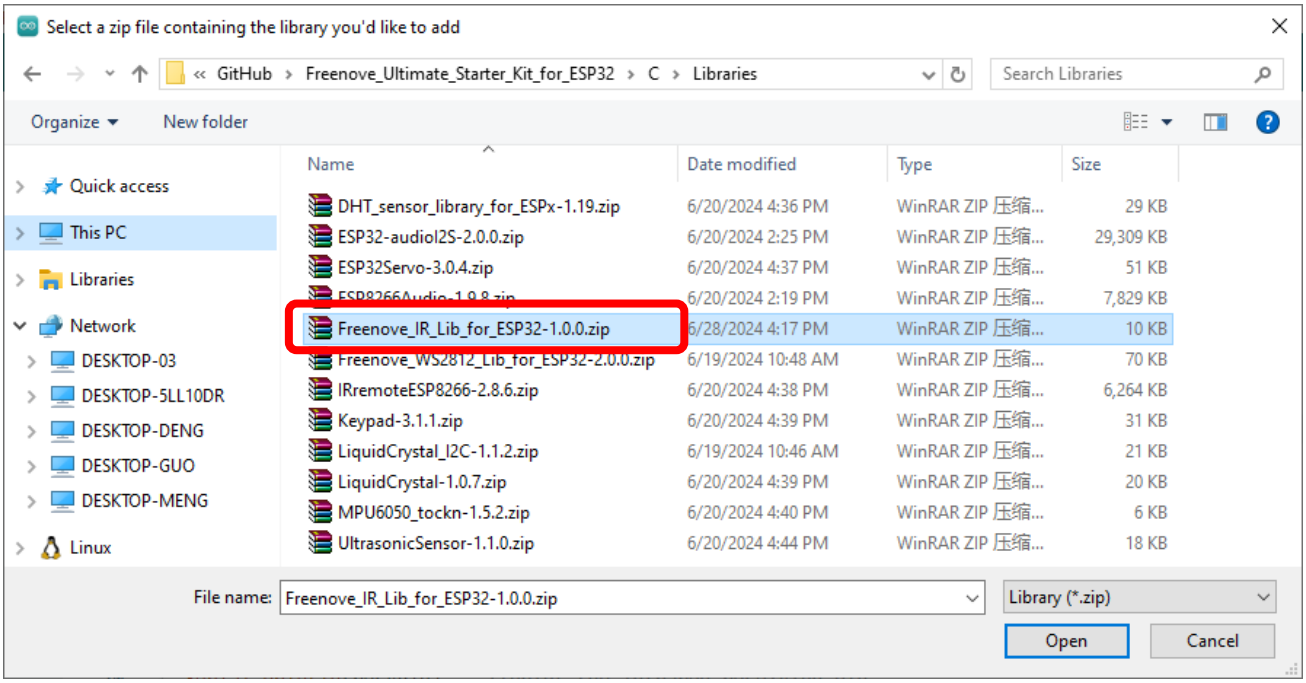
This sketch uses the infrared receiving tube to receive the value sent form the infrared remote control, and print it out via the serial port.

How to install the library

We use the third party library **Freenove_IR_Lib_for_ESP32**. If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Add .ZIP Library···.



In the **Freenove_Ultimate_Starter_Kit_for_ESP32/C/Libraries** folder, select **Freenove_IR_Lib_for_ESP32.zip** and click open.



Sketch_23.1_Infrared_Remote_Control

```
#include "Freenove_IR_Lib_for_ESP32.h"

const uint16_t recvPin = 15;              // Infrared receiving pin
Freenove_ESP32_IR_Recv ir_recv(recvPin);  // Create a class object used to receive class

void setup() {
  Serial.begin(115200);       // Initialize the serial port and set the baud rate to 115200
  Serial.print("IRrecvDemo is now running and waiting for IR message on Pin ");
  Serial.println(recvPin);   //print the infrared receiving pin
}

void loop() {
  ir_recv.task();                   //get IR receice data.
  if(ir_recv.nec_available()){      //The data of the NEC protocol is checked
    //Print IR data.
    Serial.printf("IR Code:  %#x\r\n", ir_recv.data());
  }
}
```

```
Writing at 0x00056fea... (100 %)
Wrote 296480 bytes (167747 compressed) at 0x00010000 in 2.9 seconds (effective 812.7 kbit/s)...
Hash of data verified.

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

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200, press the IR remote control, the pressed keys value will be printed out through the serial port. As shown in the following figure: (Note that when the remote control button is pressed for a long time, the infrared receiving circuit receives a continuous high level, that is, it receives a hexadecimal "F")

```
16:43:31.827 -> IR Code:  0xffa25d
16:43:31.873 -> IR Code:  0xffffffff
16:43:33.474 -> IR Code:  0xffe21d
16:43:34.012 -> IR Code:  0xff22dd
16:43:34.058 -> IR Code:  0xffffffff
16:43:34.422 -> IR Code:  0xff02fd
16:43:34.930 -> IR Code:  0xffc23d
16:43:35.421 -> IR Code:  0xffe01f
16:43:36.401 -> IR Code:  0xff9867
16:43:36.901 -> IR Code:  0xffb04f
16:43:37.484 -> IR Code:  0xffa25d
16:43:37.568 -> IR Code:  0xffffffff
16:43:37.660 -> IR Code:  0xffffffff
16:43:38.171 -> IR Code:  0xffa25d
16:43:38.216 -> IR Code:  0xffffffff
16:43:38.301 -> IR Code:  0xffffffff
16:43:38.409 -> IR Code:  0xffffffff
```

Short pressed

Long pressed

The following is the program code:

```
#include "Freenove_IR_Lib_for_ESP32.h"

const uint16_t recvPin = 15;                    // Infrared receiving pin
Freenove_ESP32_IR_Recv ir_recv(recvPin);   // Create a class object used to receive class

void setup() {
    Serial.begin(115200);           // Initialize the serial port and set the baud rate to 115200
    Serial.print("IRrecvDemo is now running and waiting for IR message on Pin ");
    Serial.println(recvPin);     //print the infrared receiving pin
}

void loop() {
    ir_recv.task();                          //get IR receice data.
    if(ir_recv.nec_available()){             //The data of the NEC protocol is checked
        Serial.printf("IR Code:   %#x\r\n", ir_recv.data());//Print IR data.
    }
}
```

First, include header file. Each time you use the infrared library, you need to include the header file at the beginning of the program.

```
#include "Freenove_IR_Lib_for_ESP32.h"
```

Second, define an infrared receive pin and associates it with the receive class.

```
const uint16_t recvPin = 15;                    // Infrared receiving pin
Freenove_ESP32_IR_Recv ir_recv(recvPin);   // Create a class object used to receive class
```

Third, call the infrared reception function, if you do not use this function, you won't receive the value from the infrared remote control.

```
ir_recv.task();                          //get IR receice data.
```

Finally, determine whether the detection IR data has been obtained, and if so, print the data.

```
void loop() {
    ir_recv.task();                          //get IR receice data.
    if(ir_recv.nec_available()){             //The data of the NEC protocol is checked
        Serial.printf("IR Code:   %#x\r\n", ir_recv.data());//Print IR data.
    }
}
```

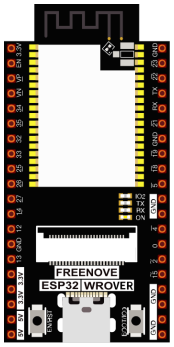
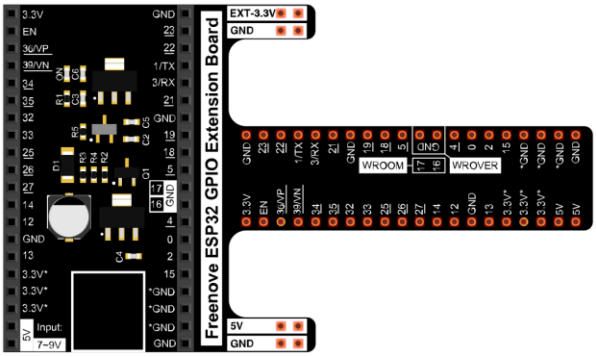
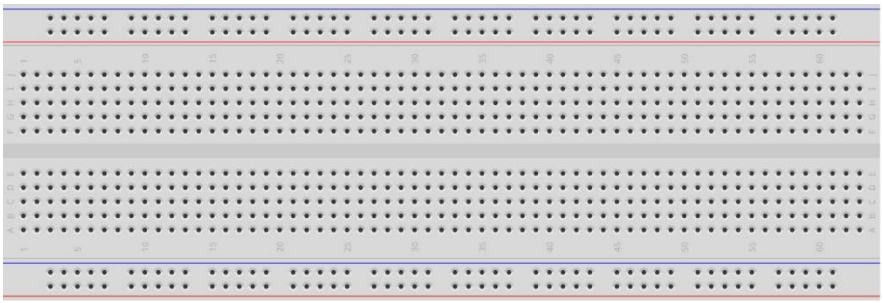
Reference

`class Freenove_ESP32_IR_Recv`	You need to add the library each time you use the Infrared Reception.
`Freenove_ESP32_IR_Recv irrecv(Pin)`: Create a class object used to receive class, and associated with **Pin**.	
task(): You need to keep calling this function, so that IR can accurately get the data.	
nec_available(): Check whether IR data is obtained from the buffer.	
data(): Get IR data.	

Project 23.2 Control LED through Infrared Remote

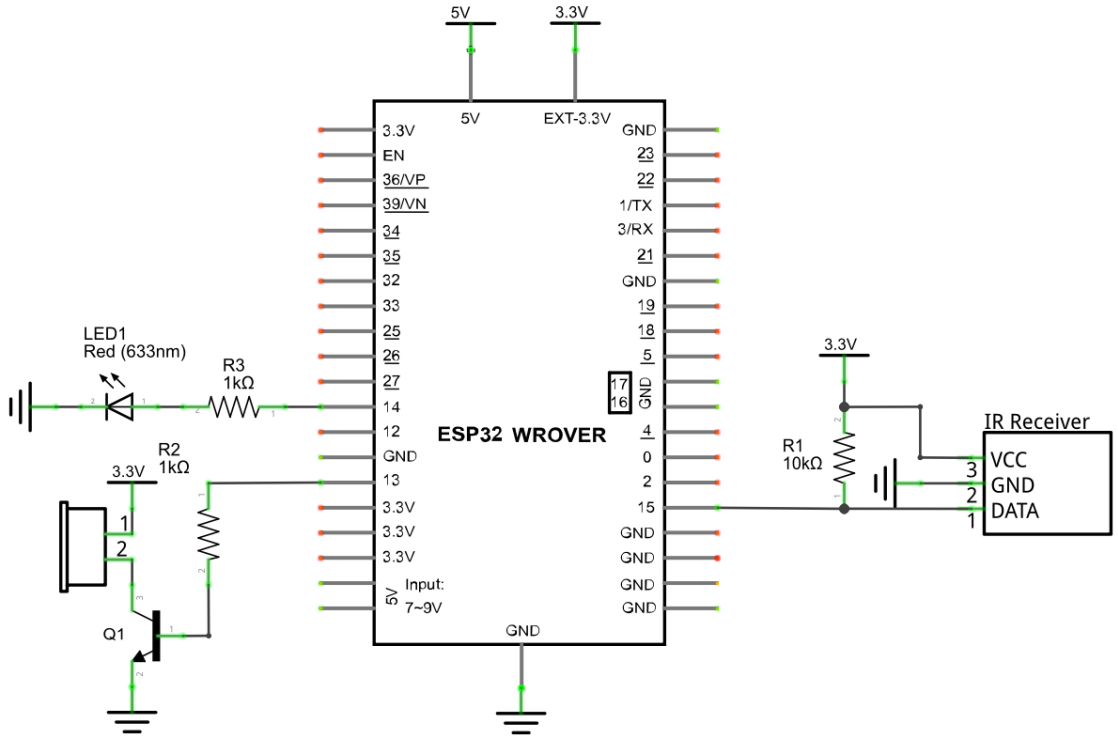
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

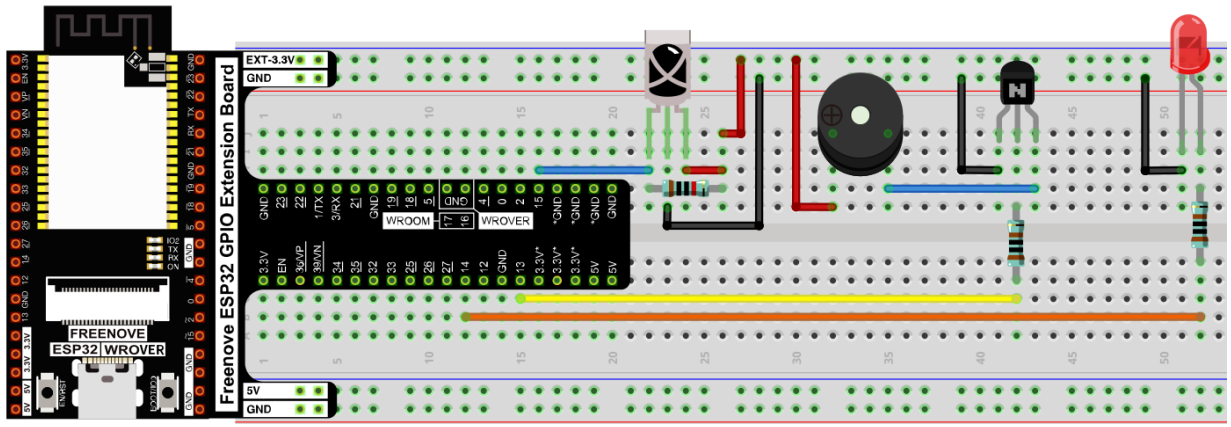
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x10		Infrared Remote x1 (May need CR2025 battery x1, please check the holder)
LED x1	Resistor 1kΩ x2	Resistor 10k Ω x1
Infrared receiver x1	NPN transistorx1 (S8050)	Active buzzer x1

Circuit

Schematic diagram



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



Sketch

The sketch controls the brightness of the LED by determining the key value of the infrared received.

Sketch_23.2_Control_LED_through_Infrared_Remote

Sketch_23.2_Control_LED_through_Infrared_Remote | Arduino IDE 2.3.2

File Edit Sketch Tools Help

ESP32 Wrover Module

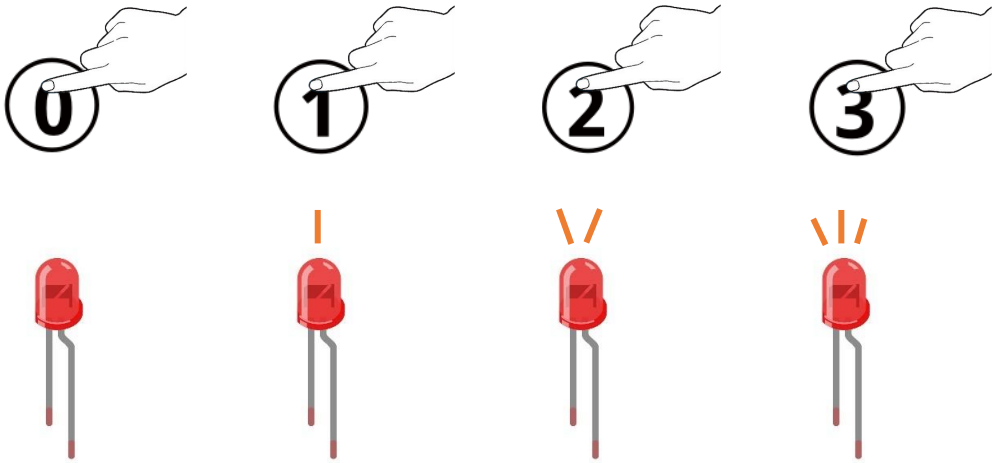
Sketch_23.2_Control_LED_through_Infrared_Remote.ino

```
#include "Freenove_IR_Lib_for_ESP32.h"

const uint16_t recvPin = 15;              // Infrared receiving pin
Freenove_ESP32_IR_Recv ir_recv(recvPin);  // Create a class object used to receive class

int ledPin = 14;              // the number of the LED pin
int buzzerPin = 13;           // the number of the buzzer pin

void setup(){
  ledcAttachChannel(ledPin, 1000, 8, 0);// set the frequency,solution_bits,channel to GPIO pin
  pinMode(buzzerPin, OUTPUT);           // set buzzer pin into output mode
}

void loop() {
  ir_recv.task();                   // Get IR receice data.
  if(ir_recv.nec_available()){      // The data of the NEC protocol is checked
    handleControl(ir_recv.data());  // Handle the commands from remote control
  }
}

void handleControl(unsigned long value) {
  // Make a sound when it rereives commands
  digitalWrite(buzzerPin, HIGH);
  delay(100);
  digitalWrite(buzzerPin, LOW);
  switch (value) {
    case 0xff6897:              // Receive the number '0'
      ledcWrite(ledPin, 0);     // Turn off LED
      break;
    case 0xff30cf:              // Receive the number '1'
      ledcWrite(ledPin, 7);     // Dimmest brightness
      break;
    case 0xff18e7:              // Receive the number '2'
      ledcWrite(ledPin, 63);    // Medium brightness
      break;
    case 0xff7a85:              // Receive the number '3'
```

Output

```
Writing at 0x00033d77... (60 %)
Writing at 0x0003c5e9... (70 %)
Writing at 0x0004460c... (80 %)
Writing at 0x0004bfcb... (90 %)
Writing at 0x0005175b... (100 %)
```

Compile and upload the code to the ESP32-WROVER. When pressing "0", "1", "2", "3" of the infrared remote control, the buzzer will sound once, and the brightness of the LED light will change correspondingly.
rendering



The following is the program code:

```
#include "Freenove_IR_Lib_for_ESP32.h"

const uint16_t recvPin = 15;                    // Infrared receiving pin
Freenove_ESP32_IR_Recv ir_recv(recvPin);   // Create a class object used to receive class
int ledPin = 14;                        // the number of the LED pin
int buzzerPin = 13;                     // the number of the buzzer pin

void setup(){
   ledcAttachChannel(ledPin, 1000, 8, 0);// set the frequency,solution_bits,channel to GPIO pin
   pinMode(buzzerPin, OUTPUT);                  // set buzzer pin into output mode
}

void loop() {
   ir_recv.task();                              // Get IR receice data.
   if(ir_recv.nec_available()){          // The data of the NEC protocol is checked
      handleControl(ir_recv.data());    // Handle the commands from remote control
   }
}

void handleControl(unsigned long value) {
   // Make a sound when it rereives commands
   digitalWrite(buzzerPin, HIGH);
   delay(100);
   digitalWrite(buzzerPin, LOW);
   switch (value) {
      case 0xff6897:                    // Receive the number '0'
         ledcWrite(ledPin, 0);         // Turn off LED
```

```
            break;
        case 0xff30cf:                  // Receive the number '1'
            ledcWrite(ledPin, 7);       // Dimmest brightness
            break;
        case 0xff18e7:                  // Receive the number '2'
            ledcWrite(ledPin, 63);      // Medium brightness
            break;
        case 0xff7a85:                  // Receive the number '3'
            ledcWrite(ledPin, 255);     // Strongest brightnss
            break;
    }
}
```

The handleControl() function is used to execute events corresponding to infrared code values. Every time when the function is called, the buzzer sounds once and determine the brightness of the LED based on the infrared key value. If the key value is not "0", "1", "2", "3", the buzzer sounds once, but the brightness of LED will not change.

```
void handleControl(unsigned long value) {
    // Make a sound when it rereives commands
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    switch (value) {
        case 0xff6897:                  // Receive the number '0'
            ledcWrite(ledPin, 0);       // Turn off LED
            break;
        case 0xff30cf:                  // Receive the number '1'
            ledcWrite(ledPin, 7);       // Dimmest brightness
            break;
        case 0xff18e7:                  // Receive the number '2'
            ledcWrite(ledPin, 63);      // Medium brightness
            break;
        case 0xff7a85:                  // Receive the number '3'
            ledcWrite(ledPin, 255);     // Strongest brightnss
            break;
    }
}
```

Each time when the command is received, the function above will be called in the loop() function.

```
void loop() {
    ir_recv.task();                         // Get IR receice data.
    if(ir_recv.nec_available()){            // The data of the NEC protocol is checked
        handleControl(ir_recv.data());      // Handle the commands from remote control
    }
}
```

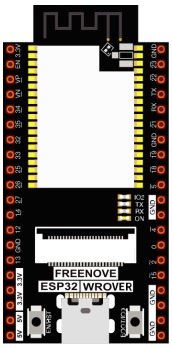
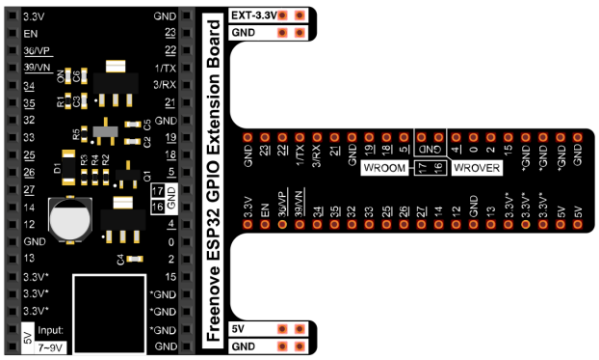
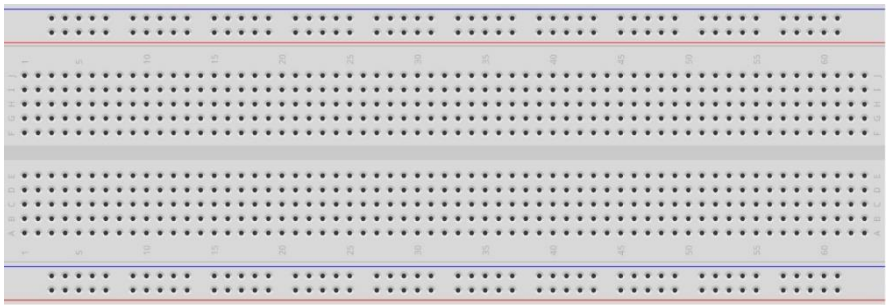
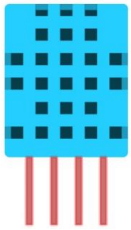
Chapter 24 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

Project 24.1 Hygrothermograph

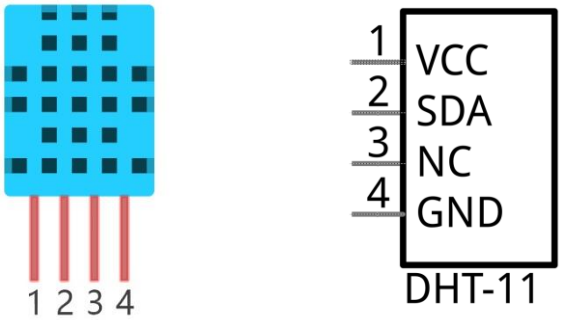
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the ESP32 to read temperature and humidity data of the DHT11 Module.

Component List

ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x4	DHT11 x1 	Resistor 10kΩ x1

Component knowledge

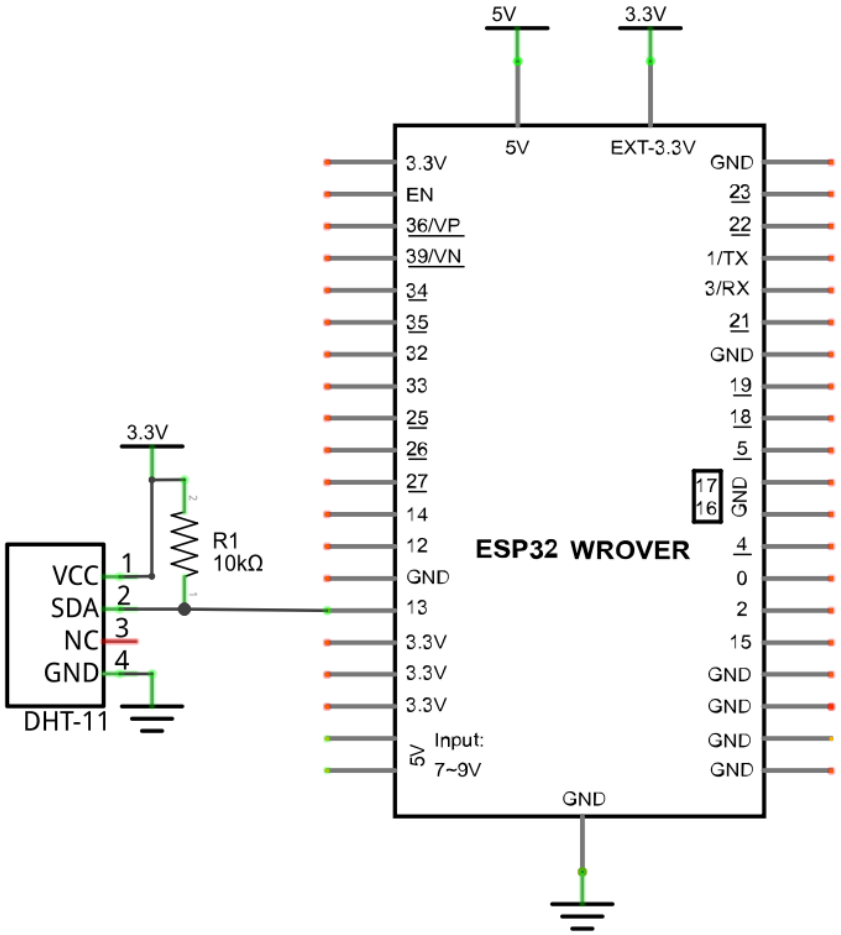
The temperature & humidity sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.



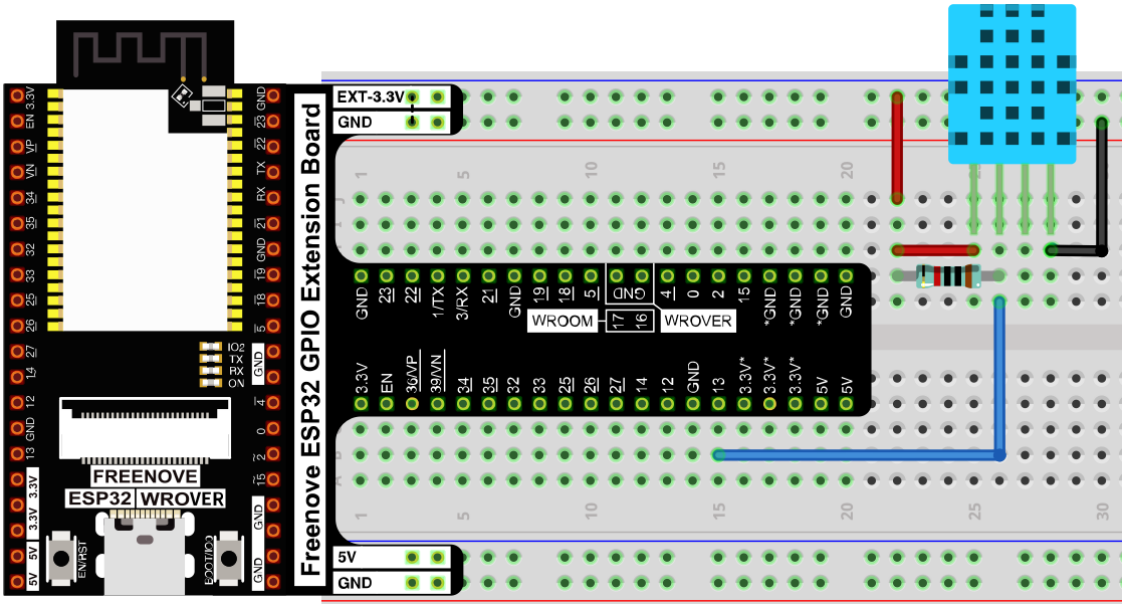
DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.
After being powered up, it will initialize in 1s. Its operating voltage is within the range of 3.3V-5.5V.
The SDA pin is a data pin, which is used to communicate with other devices.
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



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



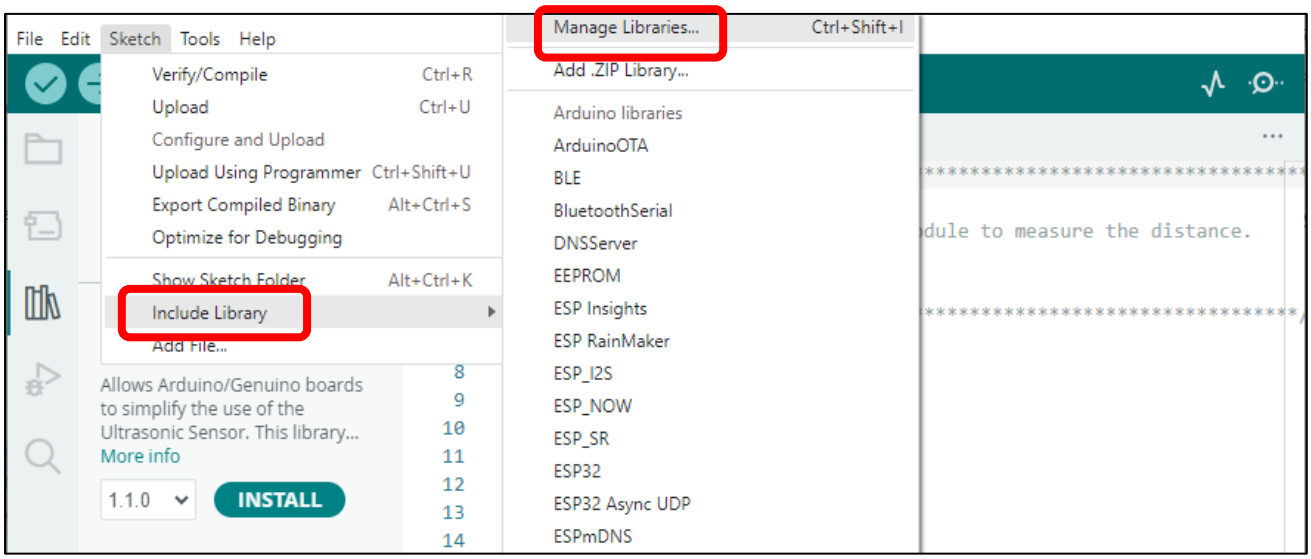
Sketch

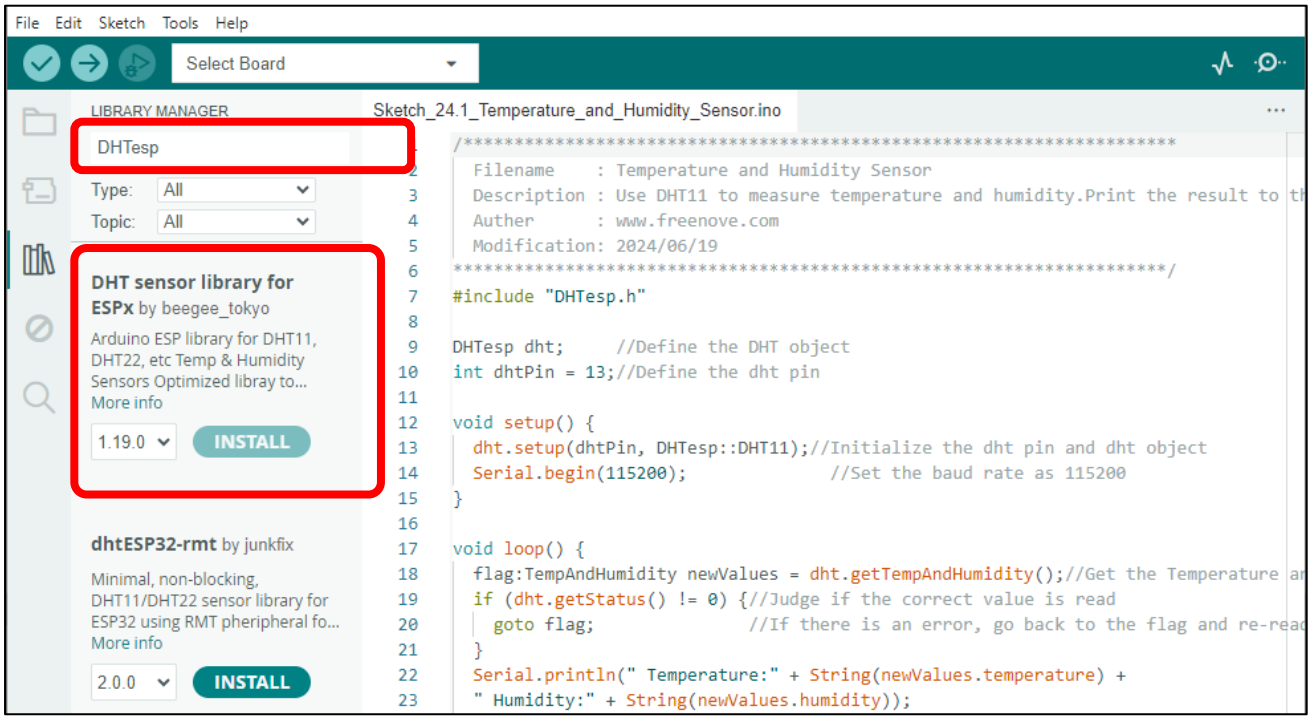
How to install the library

The code is used to read the temperature and humidity data of DHT11, and print them out.

We use the third party library **DHTesp**. If you haven't installed it yet, please do so now. The steps to add third-party libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter " **DHTesp** " in the search bar and select " **DHTesp** " for installation.

Refer to the following operations:





Sketch_24.1_Temperature_and_Humidity_Sensor

```
#include "DHTesp.h"

DHTesp dht;     //Define the DHT object
int dhtPin = 13;//Define the dht pin

void setup() {
  dht.setup(dhtPin, DHTesp::DHT11);//Initialize the dht pin and dht object
  Serial.begin(115200);              //Set the baud rate as 115200
}

void loop() {
  flag:TempAndHumidity newValues = dht.getTempAndHumidity();//Get the Temperature and humidity
  if (dht.getStatus() != 0) {//Judge if the correct value is read
    goto flag;               //If there is an error, go back to the flag and re-read the data
  }
  Serial.println(" Temperature:" + String(newValues.temperature) +
  " Humidity:" + String(newValues.humidity));
  delay(2000);
}
```

Compile and upload the code to the ESP32-WROVER, turn on the serial monitor, and set the baud rate to 115200. Print out data of temperature and humidity sensor via the serial port.

```
17:39:18.047 ->  Temperature:29.80 Humidity:68.00
17:39:20.055 ->  Temperature:29.80 Humidity:67.00
17:39:22.069 ->  Temperature:29.80 Humidity:66.00
17:39:24.091 ->  Temperature:29.80 Humidity:65.00
17:39:26.111 ->  Temperature:29.80 Humidity:64.00
17:39:28.131 ->  Temperature:29.80 Humidity:63.00
17:39:30.122 ->  Temperature:29.80 Humidity:62.00
17:39:32.170 ->  Temperature:29.80 Humidity:61.00
17:39:34.202 ->  Temperature:29.80 Humidity:60.00
17:39:36.234 ->  Temperature:29.80 Humidity:59.00
17:39:38.217 ->  Temperature:29.60 Humidity:58.00
17:39:40.255 ->  Temperature:29.30 Humidity:58.00
17:39:42.254 ->  Temperature:29.30 Humidity:57.00
17:39:44.276 ->  Temperature:29.30 Humidity:57.00
17:39:46.344 ->  Temperature:29.30 Humidity:56.00
17:39:48.320 ->  Temperature:29.30 Humidity:56.00
17:39:50.387 ->  Temperature:29.30 Humidity:56.00
17:39:52.402 ->  Temperature:29.30 Humidity:55.00
17:39:54.425 ->  Temperature:29.30 Humidity:55.00
```

The following is the program code:

```
#include "DHTesp.h"

DHTesp dht;     //Define the DHT object
int dhtPin = 13;//Define the dht pin

void setup() {
  dht.setup(dhtPin, DHTesp::DHT11);//Initialize the dht pin and dht object
  Serial.begin(115200);              //Set the baud rate to 115200
}

```

```
11 void loop() {
12   flag:TempAndHumidity newValues = dht.getTempAndHumidity();//Get the Temperature and humidity
13   if (dht.getStatus() ! = 0) {                              //Judge if the correct value is
14 read
15     goto flag;                                             //If there is an error, go back to
16 the flag and re-read the data
17   }
18   Serial.println(" Temperature:" + String(newValues.temperature) +
     " Humidity:" + String(newValues.humidity));
19   delay(2000);
20 }
```

In this project code, we use a third party library, DHTesp, and we need to define the objects for it first; Otherwise we could not use its functionality.

```
1 #include "DHTesp.h"
3 DHTesp dht;     //Define the DHT object
```

Initialize the connection pin of DHT and select the type of temperature and humidity sensor as DHT11. If the temperature and humidity sensor is DHT12, we can also change it to DHT12.

```
7 dht.setup(dhtPin, DHTesp::DHT11);//Initialize the dht pin and dht object
```

Due to the use of the single-line protocol, data may be lost in the transmission process. So each time when getting the data of the temperature and humidity sensor, we need to call the getStatus function to determine whether the data is normal. If not, use goto to go back to line 12 and re-execute the program.

```
12 flag:TempAndHumidity newValues = dht.getTempAndHumidity();//Get the Temperature and
13 humidity
14   if (dht.getStatus() ! = 0) {      //Judge if the correct value is read
15     goto flag;                     //If there is an error, go back to the flag and re-read
16 the data
17   }
```

Get the temperature and humidity data and store it in a TempAndHumidity class called newValues.

```
12 TempAndHumidity newValues = dht.getTempAndHumidity();//Get the Temperature and humidity
```

Reference

class DHTesp

Make sure that the library and header files are added before using the object every time.

setup(Pin, DHTesp::DHTxx): Select the type of DHTxx and associate Pin with the DHTesp class.

Parameter 1: the pin to be associated.

Parameter 2: select the type of sensor, DHT11 or DHT12.

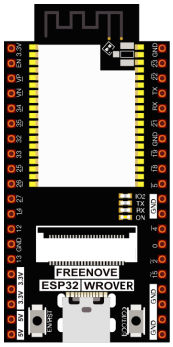
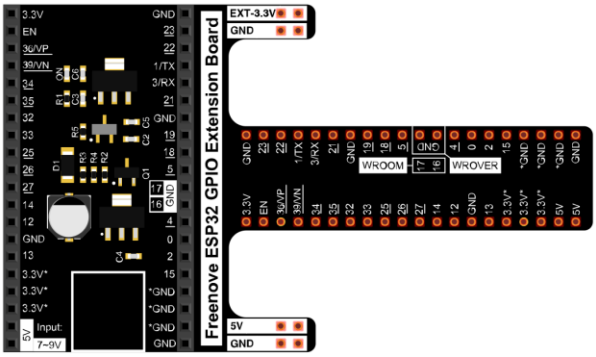
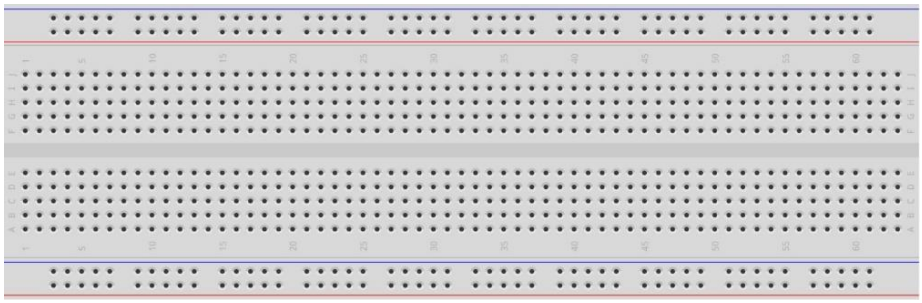
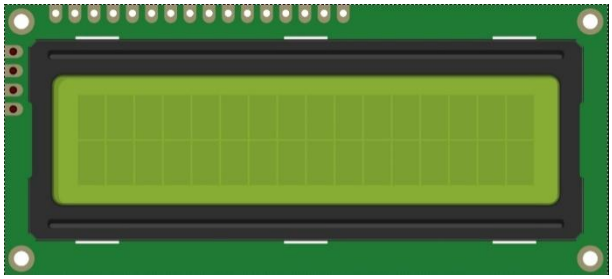
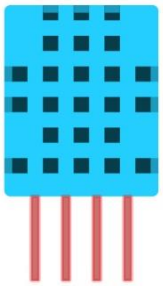
getTempAndHumidity(): Obtain temperature and humidity data. The received data must be stored in the 'TempAndHumidity' class.

getStatus(): To judge whether the obtained data format is normal, the return value of 0 means the data is normal, and the return value of non-0 means the data is abnormal or the data fails to be obtained.

Project 24.2 Hygrothermograph

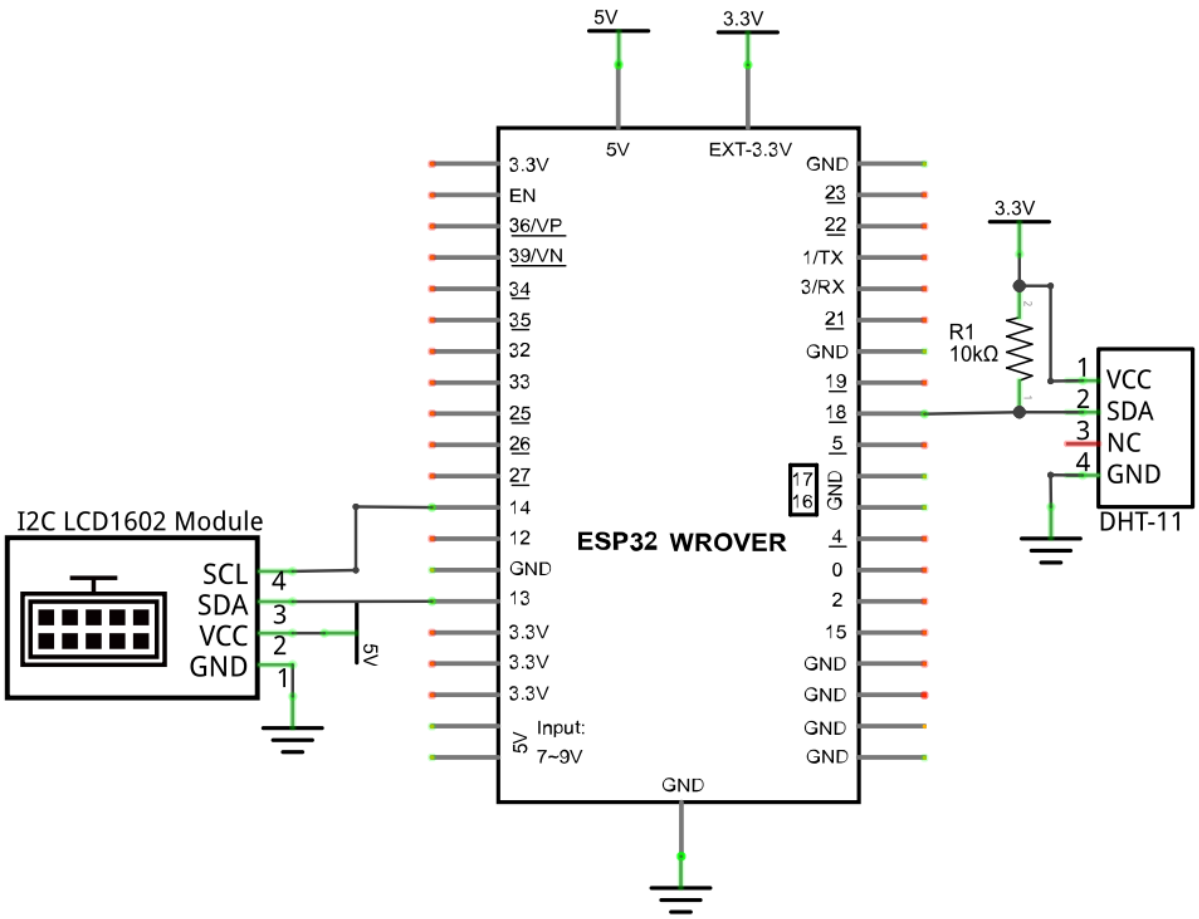
In this project, we use L2C-LCD1602 to display data collected by DHT11.

Component List

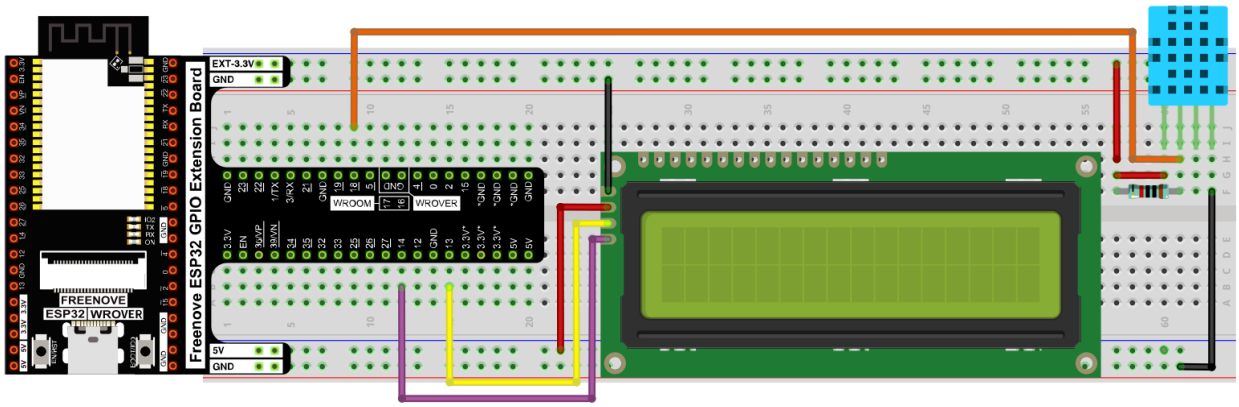
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
LCD1602 Module x1 	Resistor 10kΩ x1
Jumper F/M x4 Jumper M/M x4	DHT11 x1 

Circuit

Schematic diagram



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



Sketch

This code uses the DHTesp and LiquidCrystal_I2C libraries, so make sure the relevant library files are added before writing the program.

Sketch_24.2_Temperature_and_Humidity_Sensor

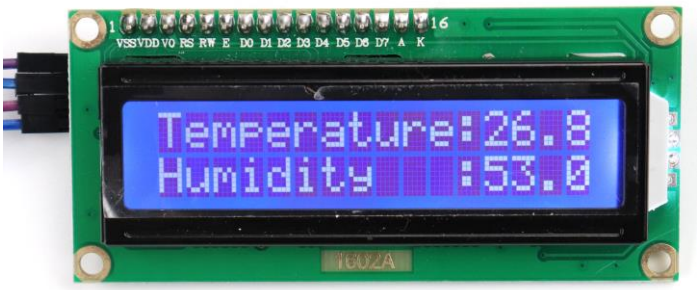
File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_24.2_Temperature_and_Humidity_Sensor.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHTesp.h"

#define SDA 13                    //Define SDA pins
#define SCL 14                    //Define SCL pins

DHTesp dht;                       // create dht object
LiquidCrystal_I2C lcd(0x27,16,2); //initialize the LCD
int dhtPin = 18;                  // the number of the DHT11 sensor pin

void setup() {
  Wire.begin(SDA, SCL);           // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  dht.setup(dhtPin, DHTesp::DHT11); //attach the dht pin and initialize it
}

void loop() {
```

Download the code to ESP32-WROVER. The first line of LCD1602 shows the temperature value, and the second line shows the humidity value. Try to "pinch" the thermistor (without touching the leads) with your index finger and thumb for a brief time to observe the change in the LCD display value.



The following is the program code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHTesp.h"

#define SDA 13                    //Define SDA pins
#define SCL 14                    //Define SCL pins
DHTesp dht;                       // create dht object
LiquidCrystal_I2C lcd(0x27,16,2); //initialize the LCD
int dhtPin = 18;                  // the number of the DHT11 sensor pin

void setup() {
  Wire.begin(SDA, SCL);           // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  dht.setup(dhtPin, DHTesp::DHT11); //attach the dht pin and initialize it
}

void loop() {
  // read DHT11 data and save it
  flag:TempAndHumidity DHT = dht.getTempAndHumidity();
  if (dht.getStatus() != 0) {       //Determine if the read is successful, and if it fails, go back to flag and re-read the data
    goto flag;
  }
  lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
  lcd.print("Temperature:");        //display the Humidity on the LCD1602
  lcd.print(DHT.temperature);
  lcd.setCursor(0, 1);              //set the cursor to column 0, line 0
  lcd.print("Humidity   :");        //display the Humidity on the LCD1602
  lcd.print(DHT.humidity);
  delay(2000);
}
bool i2CAddrTest(uint8_t addr) {
  Wire.begin();
  Wire.beginTransmission(addr);
  if (Wire.endTransmission() == 0) {
    return true;
  }
  return false;
}
```

First, add the library function header file.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHTesp.h"
```

Second, initialize the pins associated with the DHT11 sensor and I2C-LCD1602.

```
DHTesp dht;                        // create dht object
LiquidCrystal_I2C lcd(0x27,16,2); //initialize the LCD
int dhtPin = 18;                   // the number of the DHT11 sensor pin

void setup() {
  Wire.begin(SDA, SCL);            // attach the IIC pin
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                      // LCD driver initialization
  lcd.backlight();                 // Open the backlight
  dht.setup(dhtPin, DHTesp::DHT11); //attach the dht pin and initialize it
}
```

Finally, the data of temperature and humidity sensor are obtained and displayed on LCD1602. The first row shows the temperature and the second shows the humidity.

```
flag:TempAndHumidity DHT = dht.getTempAndHumidity();
  if (dht.getStatus() ! = 0) {       //Determine if the reading is successful, and if it fails, go back to flag and re-read the data
    goto flag;
  }
  lcd.setCursor(0, 0);               //set the cursor to column 0, line 1
  lcd.print("Temperature:");         //display the Humidity on the LCD1602
  lcd.print(DHT.temperature);
  lcd.setCursor(0, 1);               //set the cursor to column 0, line 0
  lcd.print("Humidity   :");         //display the Humidity on the LCD1602
  lcd.print(DHT.humidity);
```

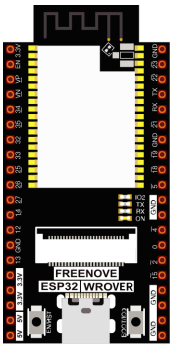
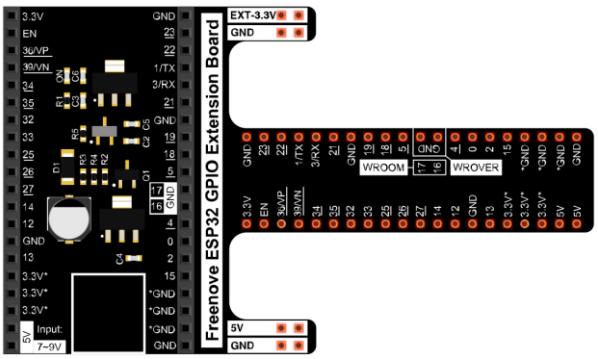
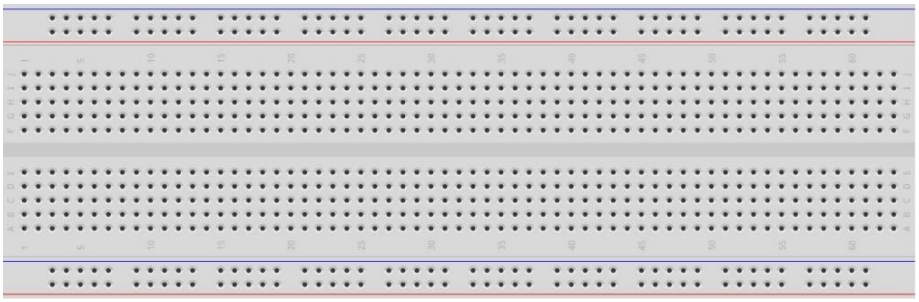
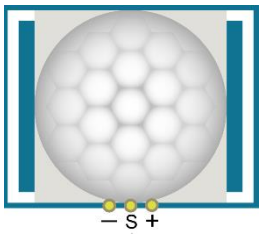
Chapter 25 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, infrared motion sensor.

Project 25.1 Infrared Motion Detector with LED Indicator

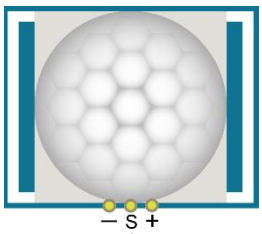
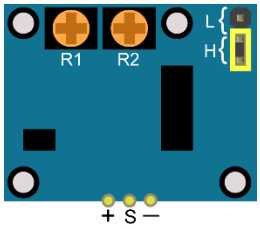
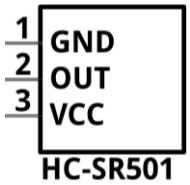
In this project, we will make a motion detector, with the human body infrared pyroelectric sensors.
When someone is in close proximity to the motion detector, it will automatically light up and when there is no one close by, it will be out.
This infrared motion sensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals.

Component List

ESP32-WROVER x1 	GPIO Extension Board x1 		
Breadboard x1 			
HC SR501 x1 	LED x1	Resistor 220Ω x1	Jumper F/M x3 Jumper M/M x2

Component knowledge

The following is the diagram of the infrared Motion sensor（HC SR-501）:

Top	Bottom	Schematic
		

Description:

Working voltage: 5v-20v(DC)　Static current: 65uA.

Automatic Trigger. When a living body enters into the active area of sensor, the module will output high level (3.3V). When the body leaves the sensor's active detection area, it will output high level lasting for time period T, then output low level(0V). Delay time T can be adjusted by the potentiometer R1.

According to the position of jumper cap, you can choose non-repeatable trigger mode or repeatable mode.

L: non-repeatable trigger mode. The module output high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves during the period of high level output. After this, it starts to time and output low level after delaying T time.

Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level

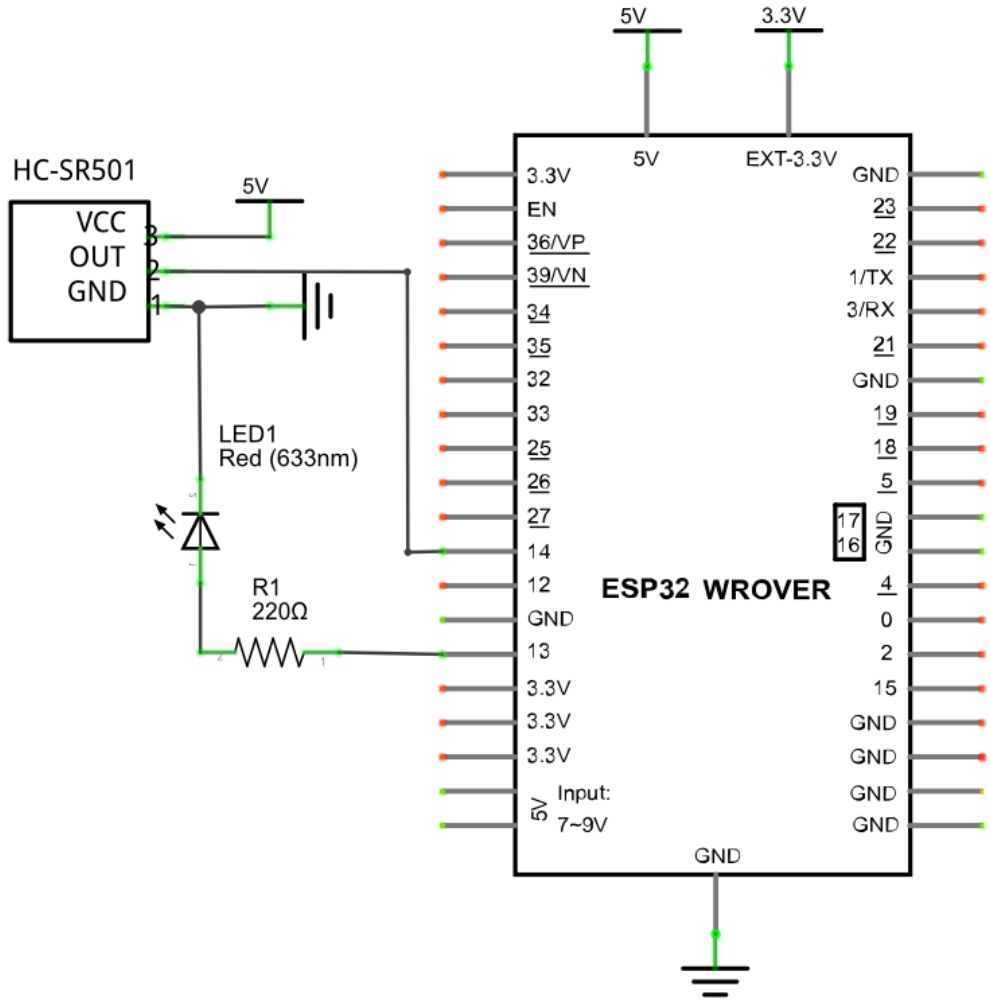
Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.

One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency). Note: The sensing range (distance before a body is detected) is adjusted by the potentiometer.

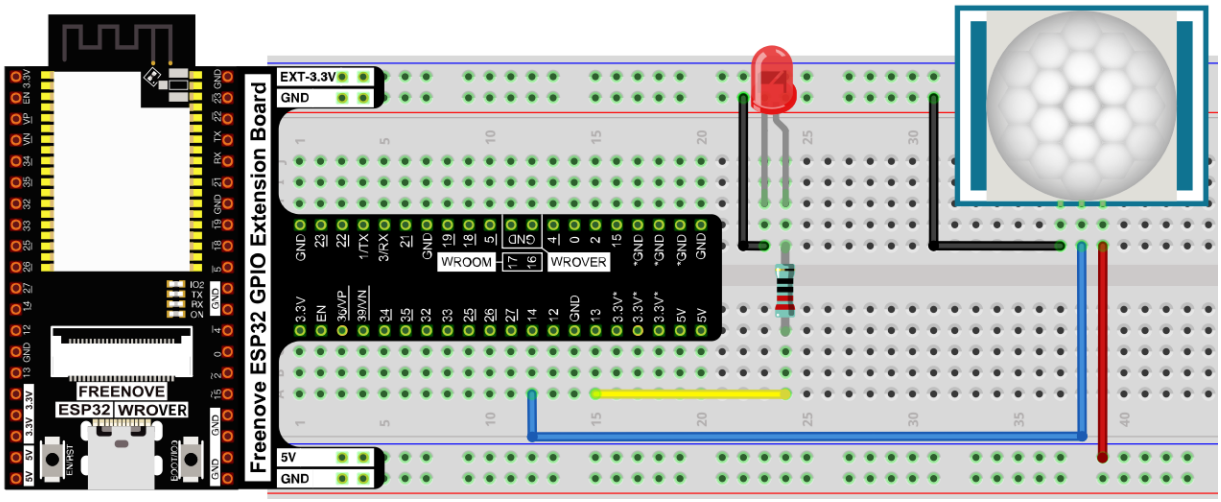
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



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



Sketch

In this project, we will use the infrared motion sensor to trigger a LED, essentially making the infrared motion sensor act as a motion switch. Therefore, the code is very similar to the earlier project "push button switch and LED". The difference is that, when infrared motion sensor detects change, it will output high level; when button is pressed, it will output low level. When the sensor output high level, the LED turns ON, or it will turn OFF.

Sketch_25.1_Infrared_Motion_Sensor

File Edit Sketch Tools Help

ESP32 Wrover Module

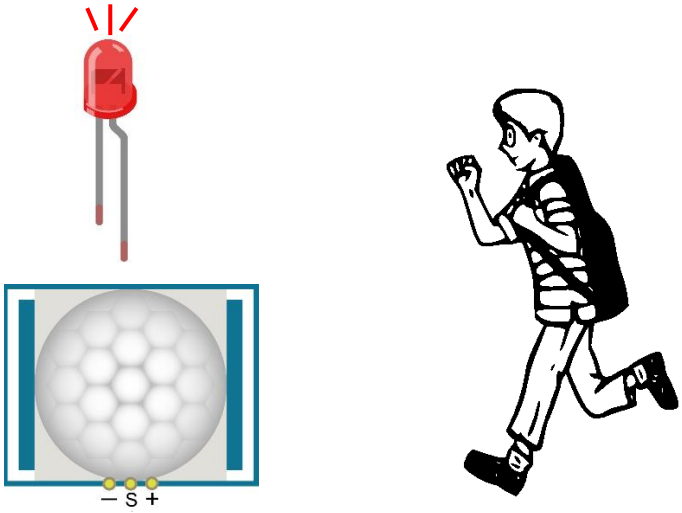
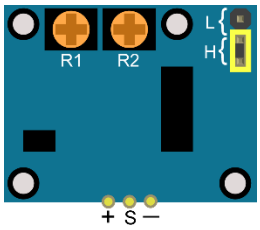
Sketch_25.1_Infrared_Motion_Sensor.ino

```
/*********************************************************************
  Filename    : Infrared Motion Sensor
  Description : LED lamp is controlled by Infrared Motion Sensor
  Auther      : www.freenove.com
  Modification: 2024/06/19
*********************************************************************/
int sensorPin = 14; // the number of the infrared motion sensor pin
int ledPin = 13;    // the number of the LED pin

void setup() {
  pinMode(sensorPin, INPUT);  // initialize the sensor pin as input
  pinMode(ledPin, OUTPUT);    // initialize the LED pin as output
}

void loop() {
  // Turn on or off LED according to Infrared Motion Sensor
  digitalWrite(ledPin, digitalRead(sensorPin));
  delay(1000);              // wait for a second
}
```

Verify and upload the code, and put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the infrared motion sensor and observe whether the LED turns ON or OFF automatically.

You can rotate the potentiometer on the sensor to adjust the detection effect, or use different modes by changing the jumper.

Apart from that, you can also use this sensor to control some other modules to implement different functions by reediting the code, such as the induction lamp, induction door.

Move to the Infrared Motion Sensor 	
Move away from the Infrared Motion Sensor	
	R1 is used to adjust the delay time. When rotated counterclockwise, when the sensor recognizes the person, the output high level time will be shorter, the shortest is about 0.5 seconds. When rotated clockwise, when the sensor recognizes a person, the output high level will be longer, up to about 300 seconds.
	R2 is used to adjust the sensor sensing distance. When rotated counterclockwise, when the sensor sensing distance will be shortened, the nearest is about 3 meters. When rotated clockwise, the sensor sensing distance will become far away, the farthest is about 7 meters.
	When H is selected on the right, when the sensor senses a person, it will always sense whether there is a person, and output a high level, until no person is sensed for a period of time, it will output a low level. When L is selected on the right, when the sensor senses a person, it no longer continues to sense whether the person exists, directly outputs a high level for a period of time, and then outputs a low level.

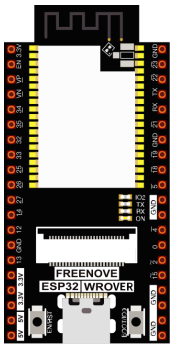
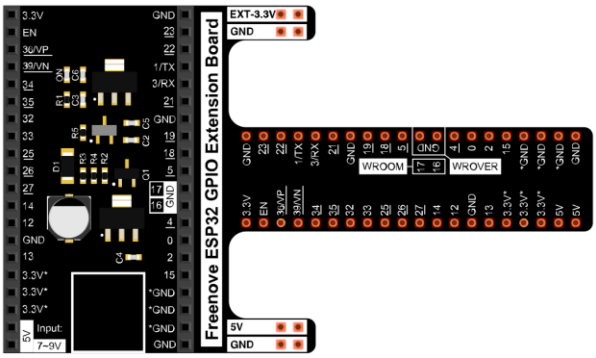
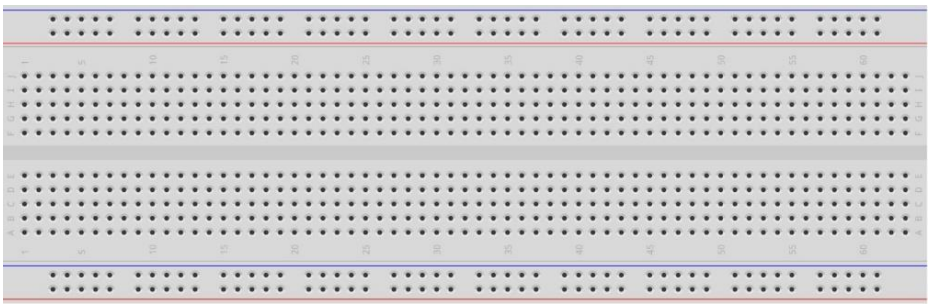
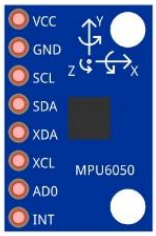
Chapter 26 Attitude Sensor MPU6050

In this chapter, we will learn about a MPU6050 attitude sensor which integrates an accelerometer and gyroscope.

Project 26.1 Read a MPU6050 Sensor Module

In this project, we will read acceleration and gyroscope data of the MPU6050 sensor

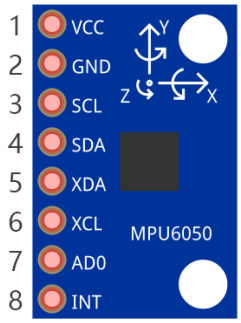
Component List

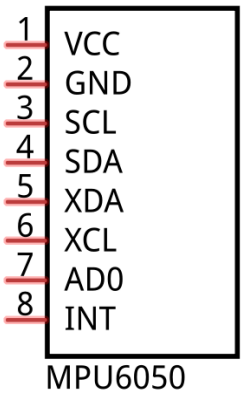
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper F/M x4	MPU6050 x1 

Component knowledge

MPU6050

MPU6050 sensor module is a complete 6-axis motion tracking device. It combines a 3-axis gyroscope, a 3-axis accelerometer and a DMP (Digital Motion Processor) all in a small package. The settings of the accelerometer and gyroscope of MPU6050 can be changed. A precision wide range digital temperature sensor is also integrated to compensate data readings for changes in temperature, and temperature values can also be read. The MPU6050 Module follows the I2C communication protocol and the default address is 0x68. MPU6050 is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.





The port description of the MPU6050 module is as follows:

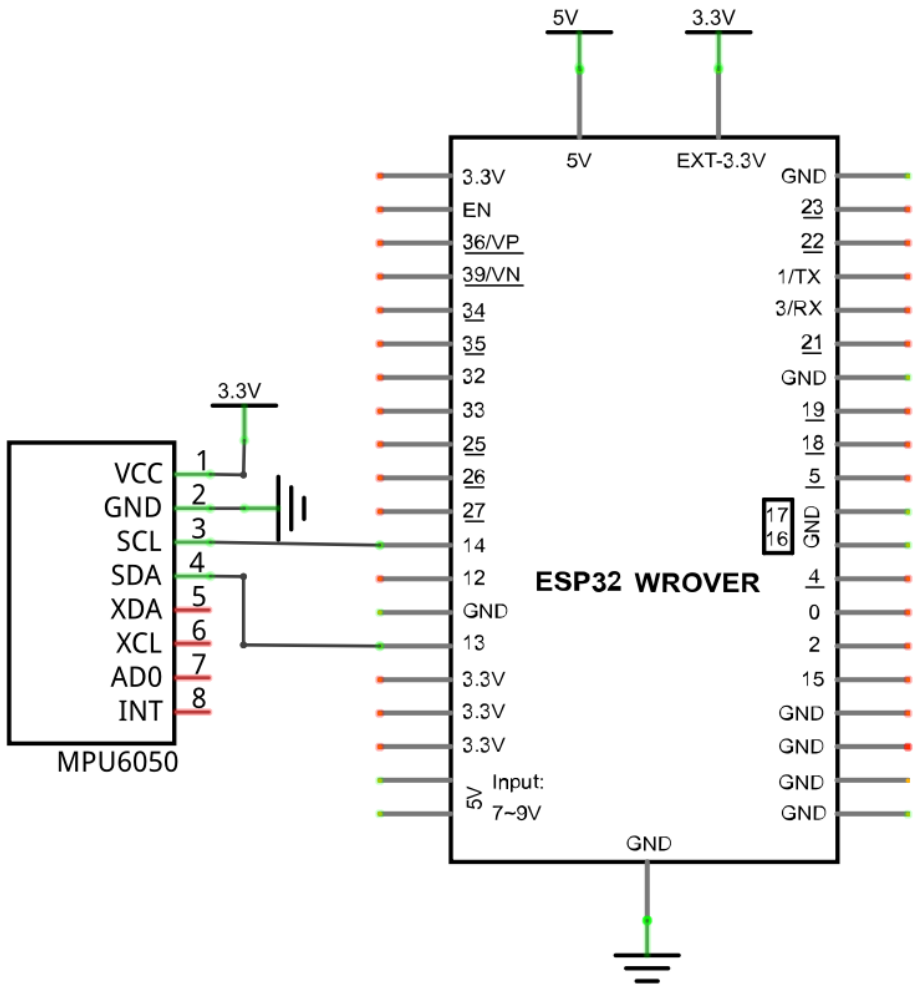
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 3.3V/5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69
INT	8	Output interrupt pin

For more detail, please refer to datasheet.

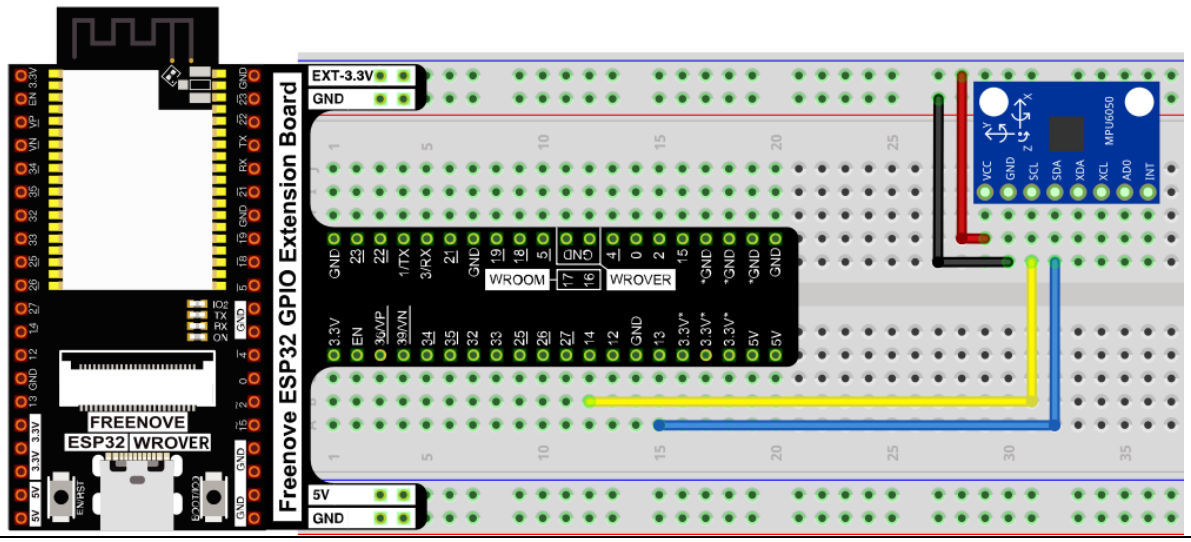
Circuit

Note that the power supply voltage for MPU6050 module is 3.3V in the circuit.

Schematic diagram



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

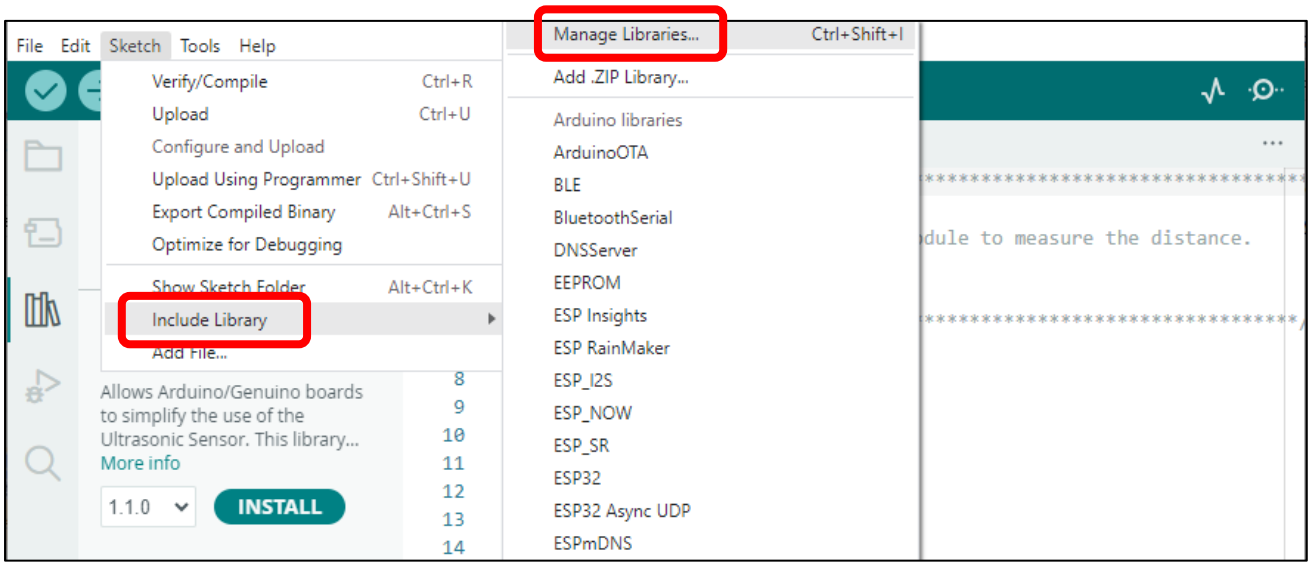


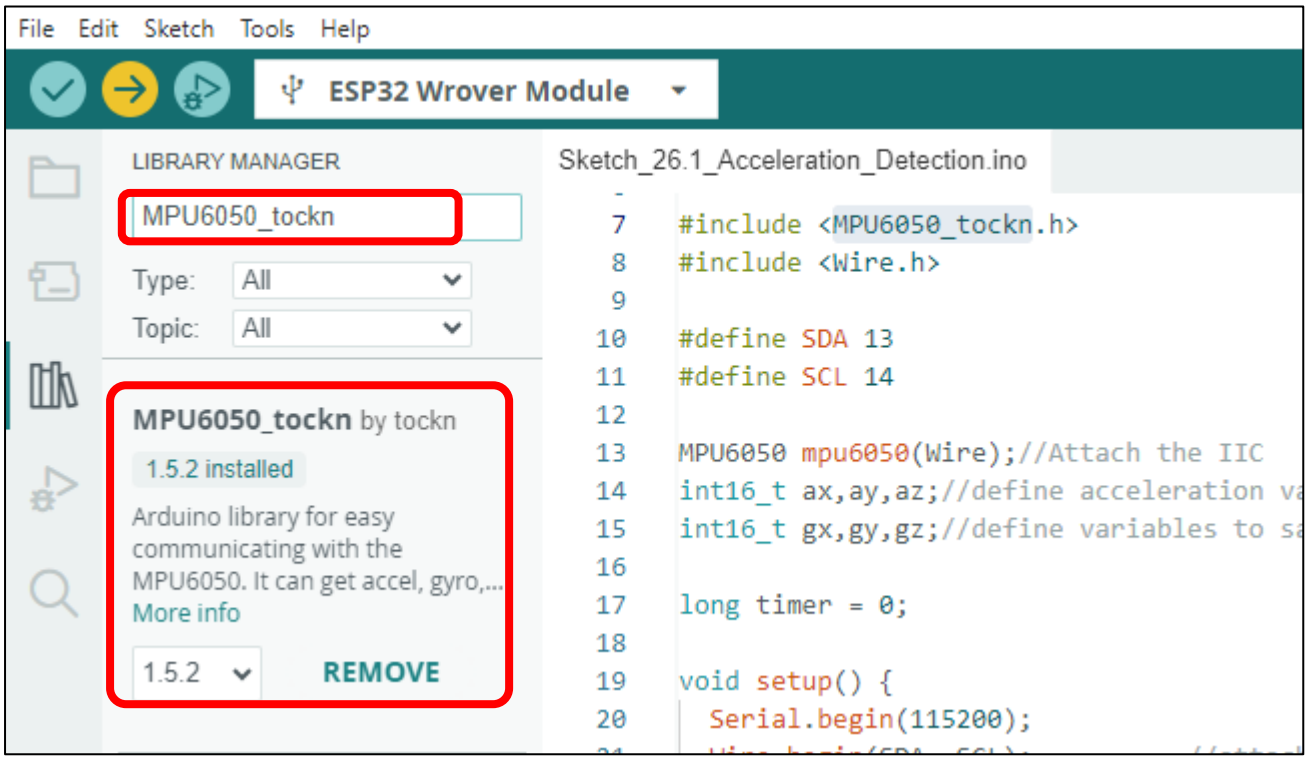
Sketch

How to install the library

In this project, we will read the acceleration data and gyroscope data of MPU6050, and print them out. We use the third party library **MPU6050_tockn**. If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries.

Enter " **MPU6050_tockn** " in the search bar and select " **MPU6050_tockn** " for installation.





Sketch_26.1_Acceleration_Detection

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_26.1_Acceleration_Detection.ino

```
#include <MPU6050_tockn.h>
#include <Wire.h>

#define SDA 13
#define SCL 14

MPU6050 mpu6050(Wire);//Attach the IIC
int16_t ax,ay,az;//define acceleration values of 3 axes
int16_t gx,gy,gz;//define variables to save the values in 3 axes of gyroscope

long timer = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA, SCL);          //attach the IIC pin
  mpu6050.begin();               //initialize the MPU6050
  mpu6050.calcGyroOffsets(true); //get the offsets value
}

void loop() {
  if(millis() - timer > 1000){   //each second printf the data
    mpu6050.update();            //update the MPU6050
    getMotion6();                //gain the values of Acceleration and Gyroscope value
    Serial.print("\na/g:\t");
    Serial.print(ax); Serial.print("\t");
    Serial.print(ay); Serial.print("\t");
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t\t");
    Serial.print(gy); Serial.print("\t\t");
    Serial.println(gz);
    Serial.print("a/g:\t");
    Serial.print((float)ax / 16384); Serial.print("g\t");
    Serial.print((float)ay / 16384); Serial.print("g\t");
    Serial.print((float)az / 16384); Serial.print("g\t");
    Serial.print((float)gx / 131); Serial.print("d/s \t");
    Serial.print((float)gy / 131); Serial.print("d/s \t");
    Serial.print((float)gz / 131); Serial.print("d/s \n");
    timer = millis();
  }
}
```

Download the code to ESP32-WROVER, open the serial port monitor, set the baud rate to 115200 and you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure:

```
Output   Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')        Both NL & CR        115200 baud
11:02:24.616 ->
11:02:24.616 -> a/g:    -12660  3360    -10424  -260            -260            -260
11:02:24.616 -> a/g:    -0.77g  0.21g   -0.64g  -1.98d/s        -1.98d/s        -1.98d/s
11:02:25.574 ->
11:02:25.574 -> a/g:    -12816  3416    -10544  -349            -349            -349
11:02:25.615 -> a/g:    -0.78g  0.21g   -0.64g  -2.66d/s        -2.66d/s        -2.66d/s
11:02:26.610 ->
11:02:26.610 -> a/g:    -12784  3324    -10520  -341            -341            -341
11:02:26.610 -> a/g:    -0.78g  0.20g   -0.64g  -2.60d/s        -2.60d/s        -2.60d/s
11:02:27.580 ->
11:02:27.580 -> a/g:    -12716  3340    -10536  -344            -344            -344
11:02:27.621 -> a/g:    -0.78g  0.20g   -0.64g  -2.63d/s        -2.63d/s        -2.63d/s
11:02:28.609 ->
11:02:28.609 -> a/g:    -12600  5052    -8984   1308            1308            1308
11:02:28.609 -> a/g:    -0.77g  0.31g   -0.55g  9.98d/s         9.98d/s         9.98d/s
11:02:29.588 ->
11:02:29.588 -> a/g:    -13376  412     -10108  324             324             324
11:02:29.588 -> a/g:    -0.82g  0.03g   -0.62g  2.47d/s         2.47d/s         2.47d/s
11:02:30.611 ->
11:02:30.611 -> a/g:    -14936  -1952   -8276   -1069           -1069           -1069
11:02:30.611 -> a/g:    -0.91g  -0.12g  -0.51g  -8.16d/s        -8.16d/s        -8.16d/s
```

The following is the program code:

```
#include <MPU6050_tockn.h>
#include <Wire.h>

#define SDA 13
#define SCL 14

MPU6050 mpu6050(Wire);//Attach the IIC
int16_t ax,ay,az;//define acceleration values of 3 axes
int16_t gx,gy,gz;//define variables to save the values in 3 axes of gyroscope

long timer = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA, SCL);          //attach the IIC pin
  mpu6050.begin();               //initialize the MPU6050
  mpu6050.calcGyroOffsets(true); //get the offsets value
}

void loop() {
  if(millis() - timer > 1000){   //each second print the data
    mpu6050.update();            //update the MPU6050
    getMotion6();                //gain the values of Acceleration and Gyroscope value
    Serial.print("\na/g:\t");
```

```
    Serial.print(ax); Serial.print("\t");
    Serial.print(ay); Serial.print("\t");
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t\t");
    Serial.print(gy); Serial.print("\t\t");
    Serial.println(gz);
    Serial.print("a/g:\t");
    Serial.print((float)ax / 16384); Serial.print("g\t");
    Serial.print((float)ay / 16384); Serial.print("g\t");
    Serial.print((float)az / 16384); Serial.print("g\t");
    Serial.print((float)gx / 131); Serial.print("d/s \t");
    Serial.print((float)gy / 131); Serial.print("d/s \t");
    Serial.print((float)gz / 131); Serial.print("d/s \n");
    timer = millis();
  }
}
void getMotion6(void){
  ax=mpu6050.getRawAccX();//gain the values of X axis acceleration raw data
  ay=mpu6050.getRawAccY();//gain the values of Y axis acceleration raw data
  az=mpu6050.getRawAccZ();//gain the values of Z axis acceleration raw data
  gx=mpu6050.getRawGyroX();//gain the values of X axis Gyroscope raw data
  gy=mpu6050.getRawGyroX();//gain the values of Y axis Gyroscope raw data
  gz=mpu6050.getRawGyroX();//gain the values of Z axis Gyroscope raw data
}
```

Two library files "**MPU6050_tockn.h**" and "**Wire.h**" are used in the code and will be compiled with others. Class MPU6050 is used to operate the MPU6050. When using it, please instantiate an object first.

```
MPU6050 mpu6050(Wire);//Attach the IIC
```

In the setup function, IIC and MPU6050 are initialized and the offset difference of MPU6050 is obtained.

```
void setup() {
  Serial.begin(115200);
  Wire.begin(SDA, SCL);            //attach the IIC pin
  mpu6050.begin();                 //initialize the MPU6050
  mpu6050.calcGyroOffsets(true);   //get the offsets value
}
```

The getMotion6 function is used to obtain the x, y, z axis acceleration raw data and the Gyroscope raw data.

```
void getMotion6(void){
  ax=mpu6050.getRawAccX();//gain the values of X axis acceleration raw data
  ay=mpu6050.getRawAccY();//gain the values of Y axis acceleration raw data
  az=mpu6050.getRawAccZ();//gain the values of Z axis acceleration raw data
  gx=mpu6050.getRawGyroX();//gain the values of X axis Gyroscope raw data
  gy=mpu6050.getRawGyroX();//gain the values of Y axis Gyroscope raw data
  gz=mpu6050.getRawGyroX();//gain the values of Z axis Gyroscope raw data
}
```

Finally, the original data of the gyroscope is updated and acquired every second, and the original data, the

processed acceleration and angular velocity data are printed out through the serial port.

```
void loop() {
  if(millis() - timer > 1000){          //each second print the data
    mpu6050.update();                   //update the MPU6050
    getMotion6();                        //gain the values of Acceleration and Gyroscope value
    Serial.print("\na/g:\t");
    Serial.print(ax); Serial.print("\t");
    Serial.print(ay); Serial.print("\t");cc
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t\t");
    Serial.print(gy); Serial.print("\t\t");
    Serial.println(gz);
    Serial.print("a/g:\t");
    Serial.print((float)ax / 16384); Serial.print("g\t");
    Serial.print((float)ay / 16384); Serial.print("g\t");
    Serial.print((float)az / 16384); Serial.print("g\t");
    Serial.print((float)gx / 131); Serial.print("d/s \t");
    Serial.print((float)gy / 131); Serial.print("d/s \t");
    Serial.print((float)gz / 131); Serial.print("d/s \n");
    timer = millis();
  }
}
```

Reference

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

`MPU6050 mpu6050(Wire)`: Associate MPU6050 with IIC.

`begin()`: Initialize the MPU6050.

`calcGyroOffsets(true)`: If the parameter is true, get the gyro offset and automatically correct the offset. If the parameter is false, the offset value is not obtained and the offset is not corrected.

`getRawAccX()`: Gain the values of X axis acceleration raw data.

`getRawAccY()`: Gain the values of Y axis acceleration raw data.

`getRawAccZ()`: Gain the values of Z axis acceleration raw data.

`getRawGyroX()`: Gain the values of X axis Gyroscope raw data.

`getRawGyroY()`: Gain the values of Y axis Gyroscope raw data.

`getRawGyroZ()`: gain the values of Z axis Gyroscope raw data.

`getTemp()`: Gain the values of MPU6050'temperature data.

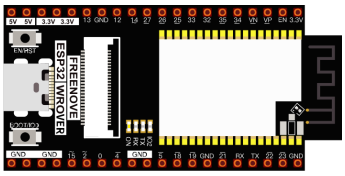
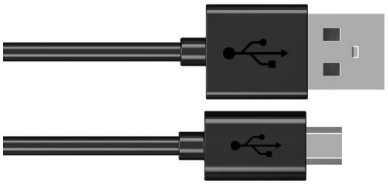
`update()`: Update the MPU6050. If the updated function is not used, the IIC will not be able to retrieve the new data.

Chapter 27 Bluetooth

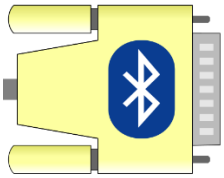
This chapter mainly introduces how to make simple data transmission through Bluetooth of ESP32-WROVER and mobile phones.
Project 27.1 is classic Bluetooth and Project 27.2 is low power Bluetooth.If you are an iPhone user, please start with Project 27.2.

Project 27.1 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.

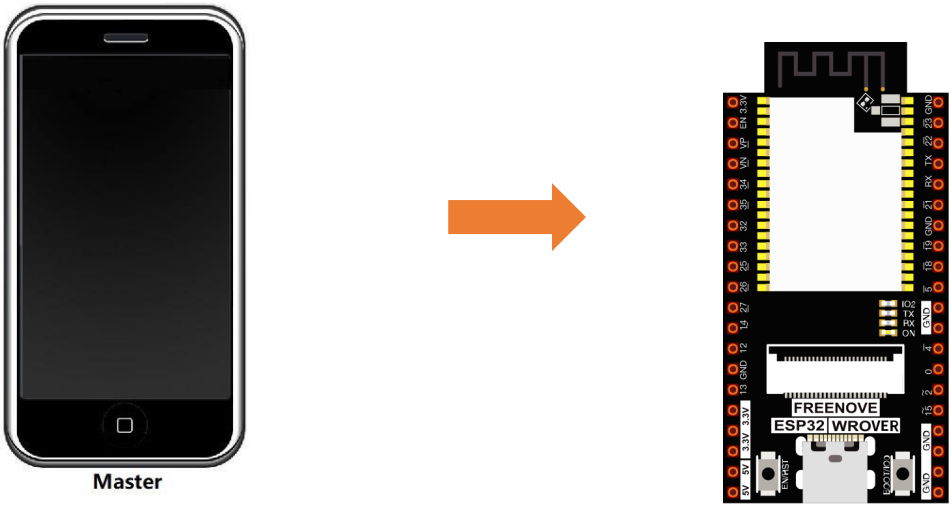
Master mode

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.

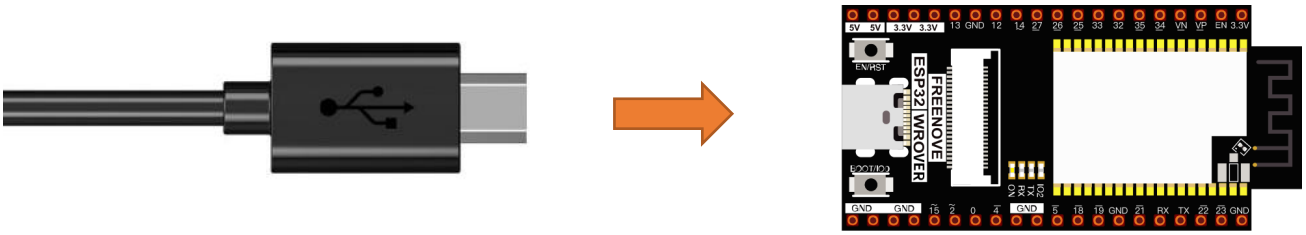


Master

Slave

Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_27.1_SerialToSerialBT

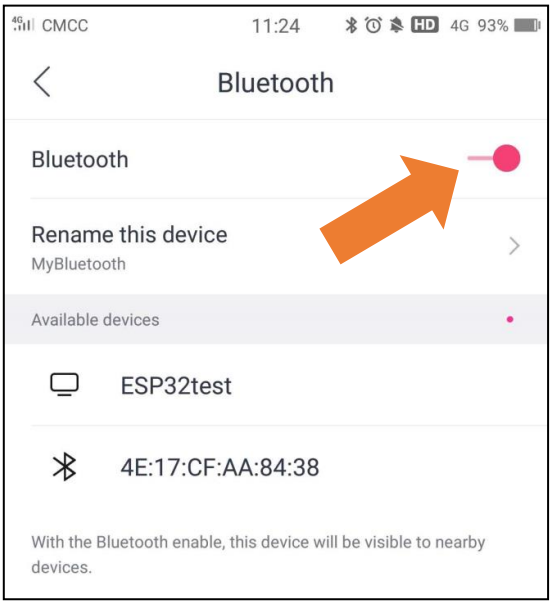
```
File Edit Sketch Tools Help
ESP32 Wrover Module
Sketch_27.1_SerialToSerialBT.ino
#include "BluetoothSerial.h"

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);
}
```

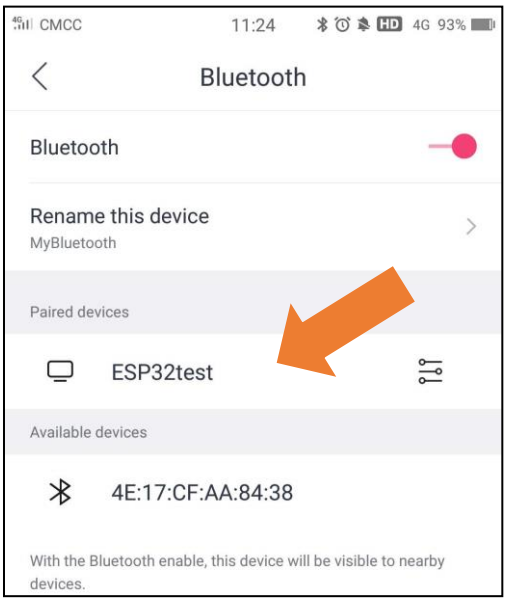
Compile and upload the code to the ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. When you see the serial printing out the character string as below, it indicates that the Bluetooth of ESP32 is ready and waiting to connect with the mobile phone.

```
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!
```

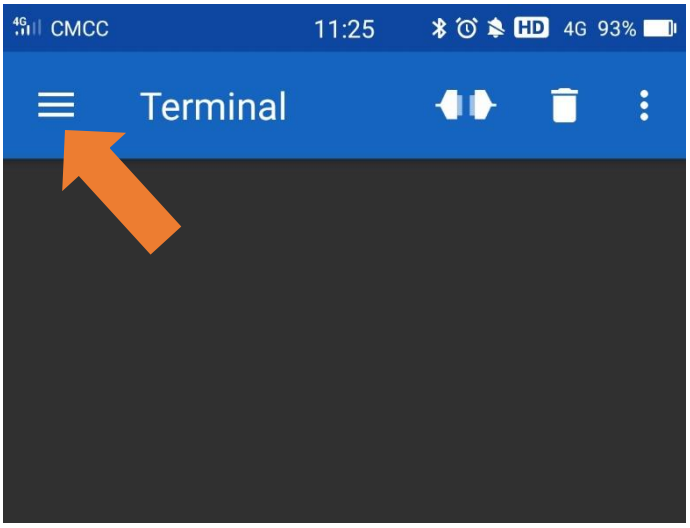
Make sure that the Bluetooth of your phone has been turned on and Serial Bluetooth Terminal has been installed.

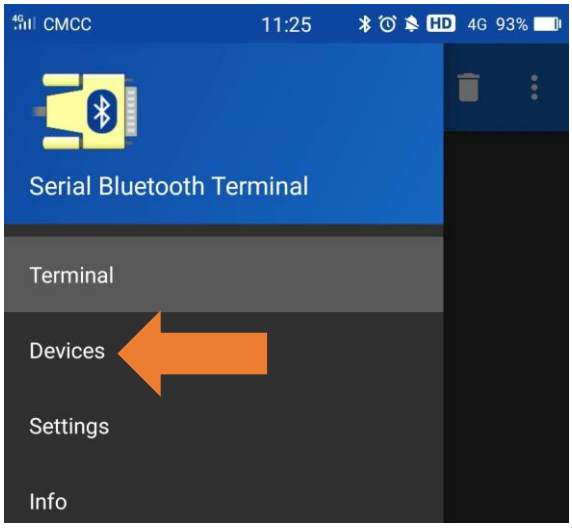
Click "Search" to search Bluetooth devices nearby and select "ESP32 test" to connect to.



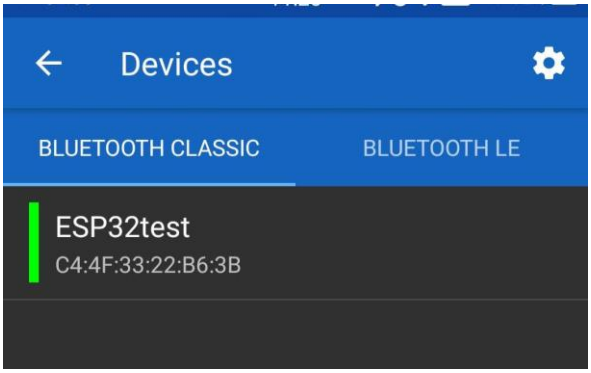


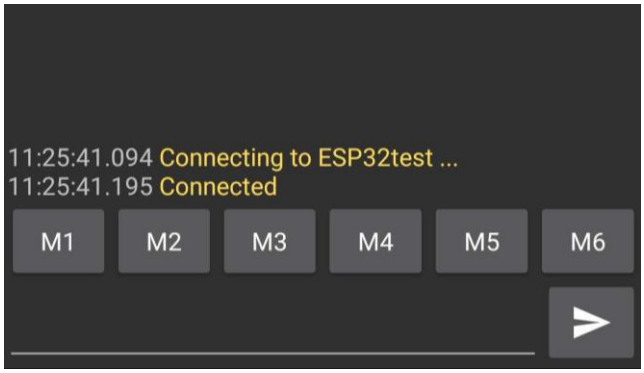
Turn on software APP, click the left of the terminal. Select "Devices"





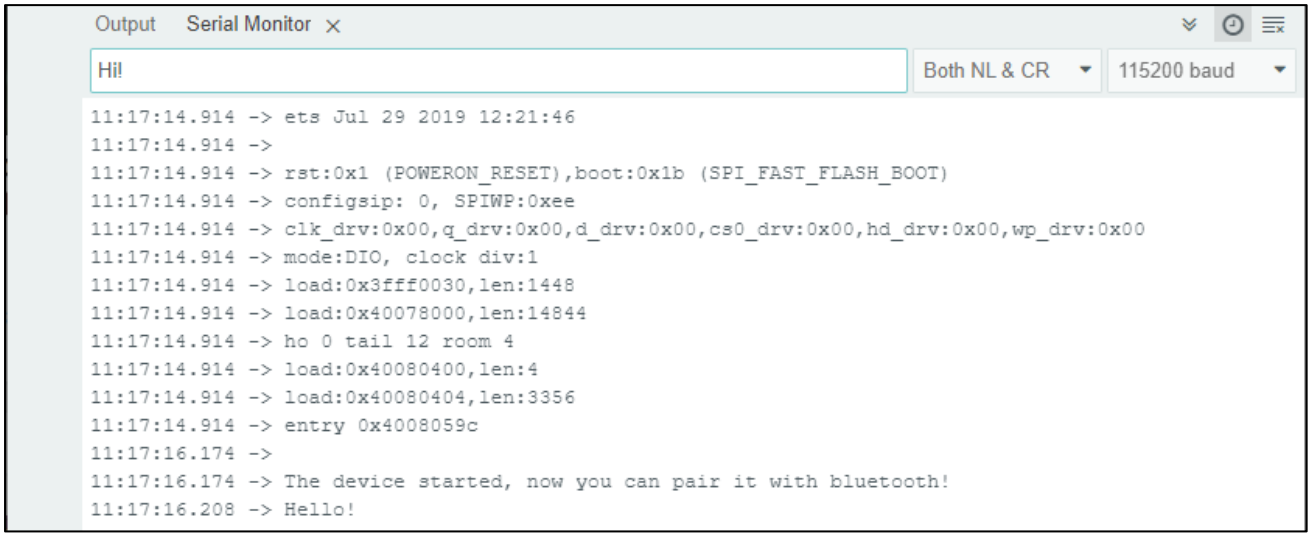
Select ESP32test in classic Bluetooth mode, and a successful connecting prompt will appear as shown on the right illustration.

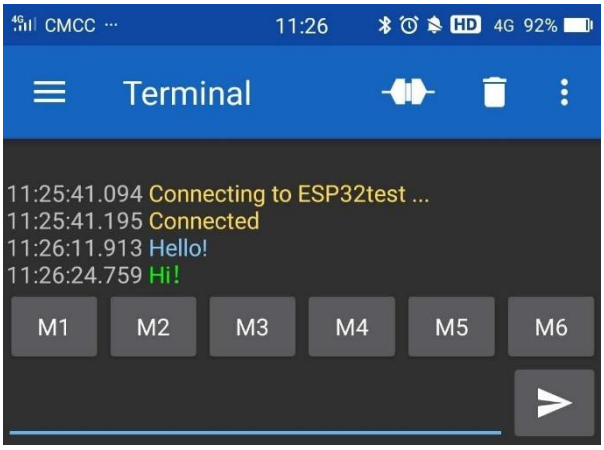




And now data can be transferred between your mobile phone and computer via ESP32-WROVER.

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

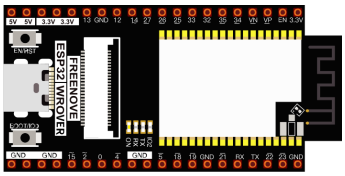
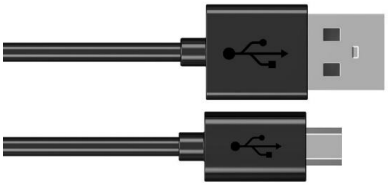
`connect(remoteAddress[])`: connect the physical address of Bluetooth, data type: uint8-t.

`disconnect()`:disconnect all Bluetooth devices.

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

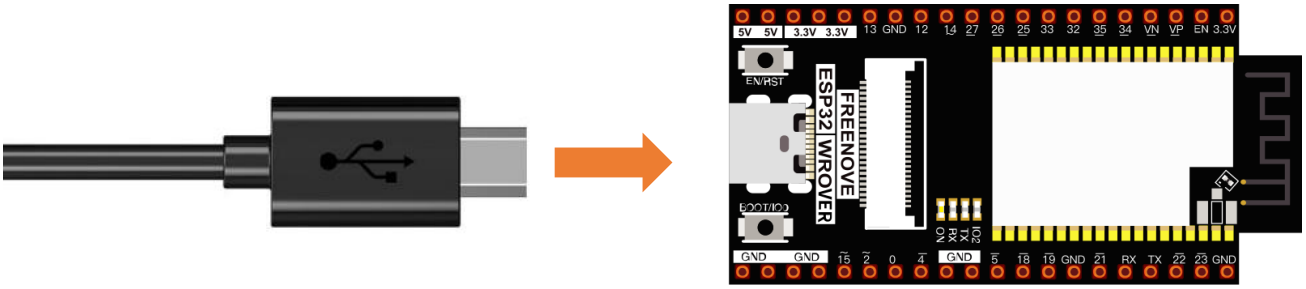
Project 27.2 Bluetooth Low Energy Data Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_27.2_BLE

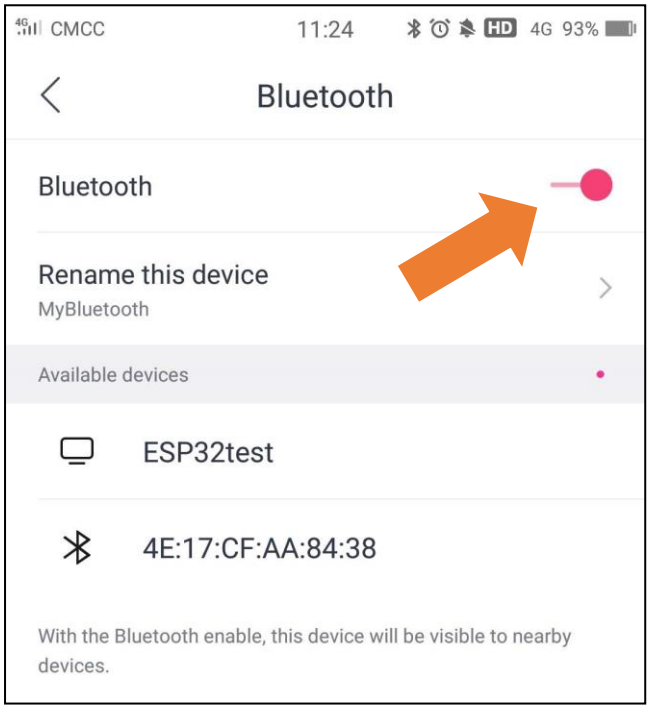
```
void setup() {
  Serial.begin(115200);
  setupBLE("ESP32_Bluetooth");
}

void loop() {
  long now = millis();
  if (now - lastMsg > 100) {
    if (deviceConnected&&rxload.length()>0) {
        Serial.println(rxload);
        rxload="";
    }
    if(Serial.available()>0){
        String str=Serial.readString();
        const char *newValue=str.c_str();
        pCharacteristic->setValue(newValue);
        pCharacteristic->notify();
    }
    lastMsg = now;
  }
}
```

Serial Bluetooth

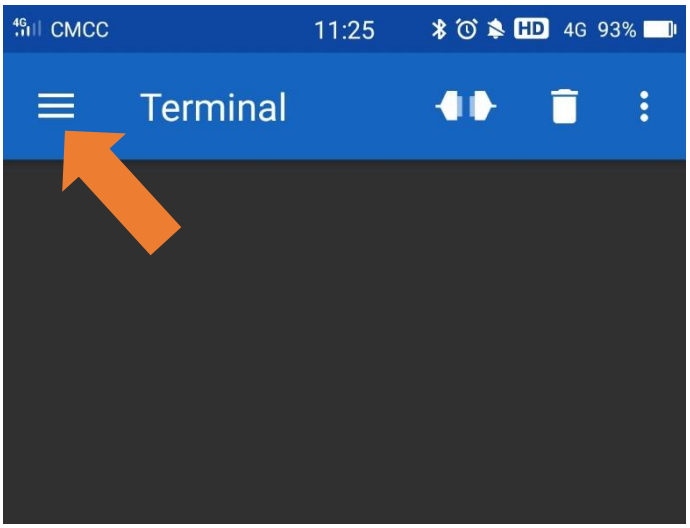
Compile and upload code to ESP32, the operation is similar to the last section.

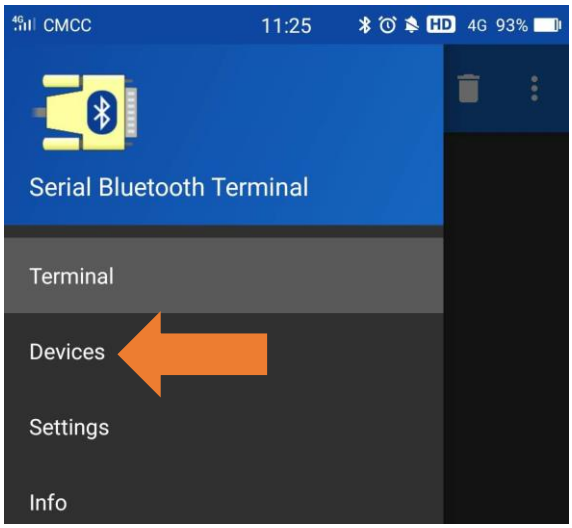
First, make sure you've turned on the mobile phone Bluetooth, and then open the software.

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

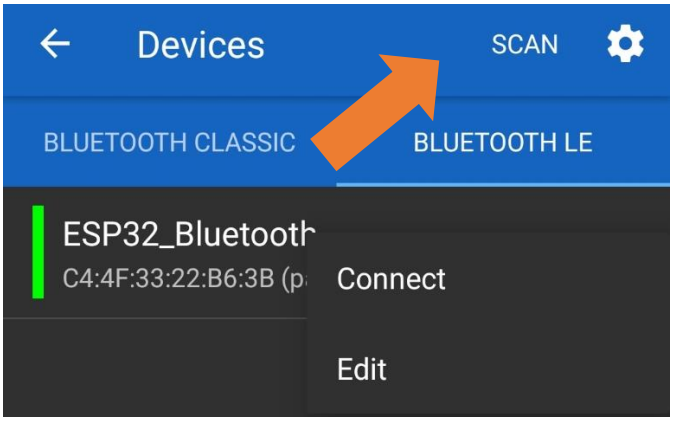


Turn on software APP, click the left of the terminal. Select "Devices"

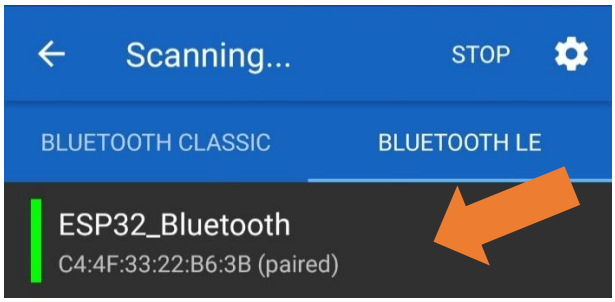




Select BLUETOOTHLE, click SCAN to scan Low Energy Bluetooth devices nearby.

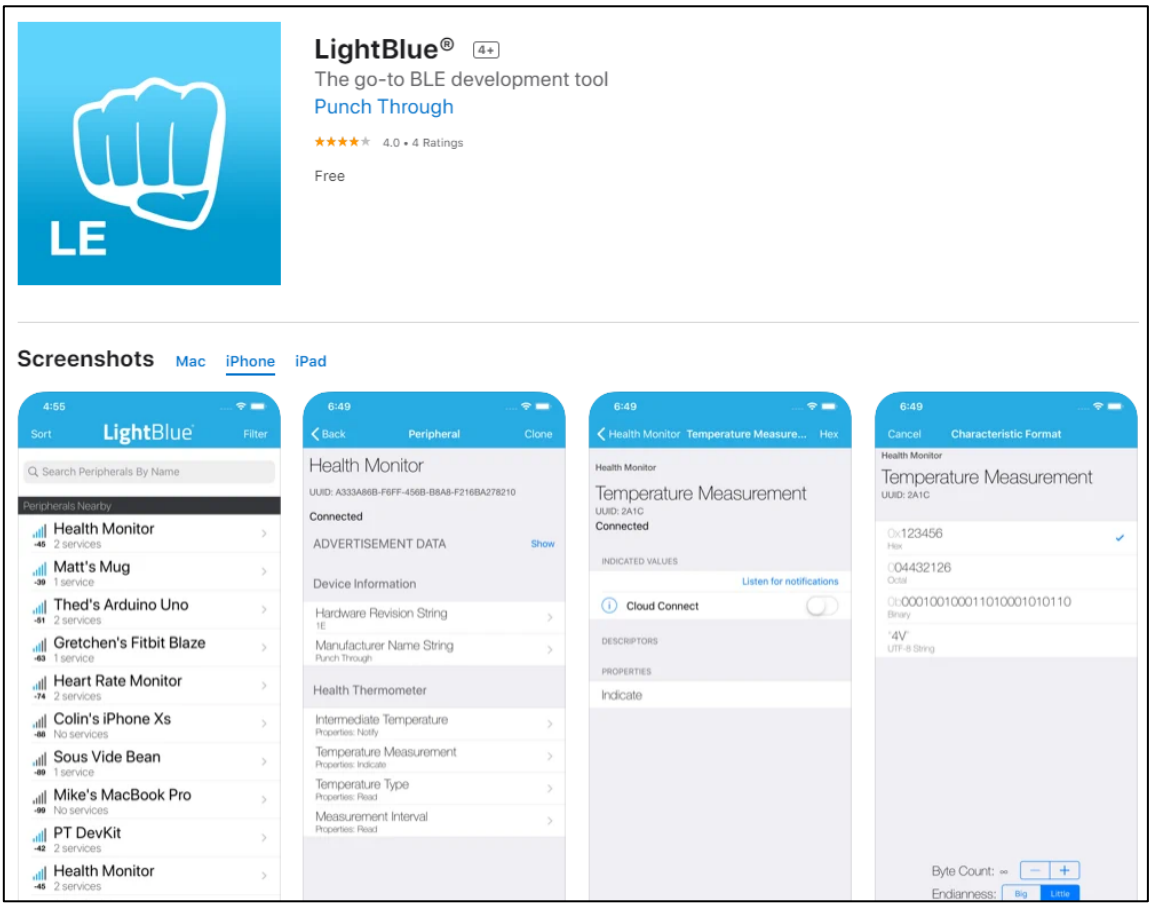


Select"ESP32-Bluetooth"



Lightblue

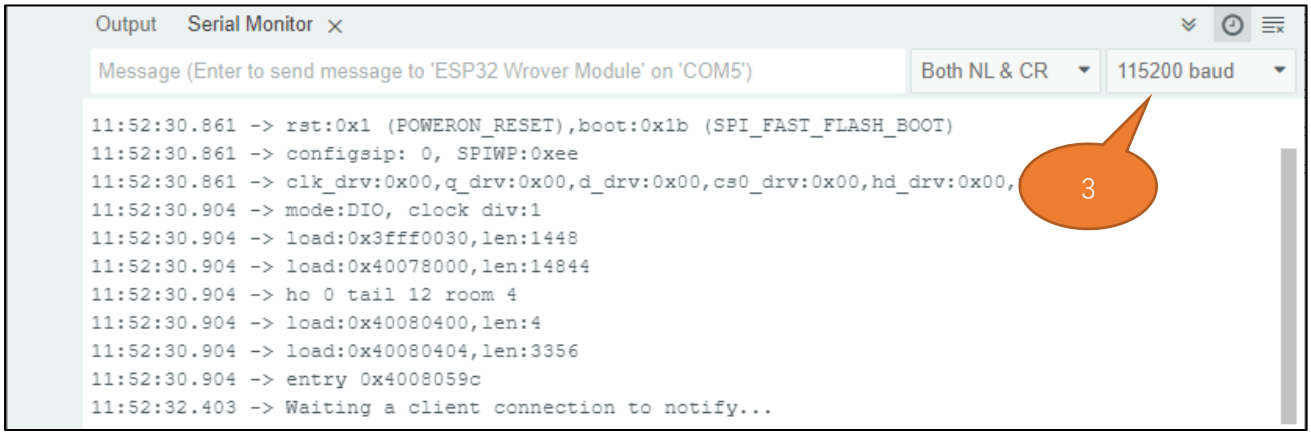
If you can't install Serial Bluetooth on your phone, try LightBlue.If you do not have this software installed on your phone, you can refer to this link:
https://apps.apple.com/us/app/lightblue/id557428110#?platform=iphone.



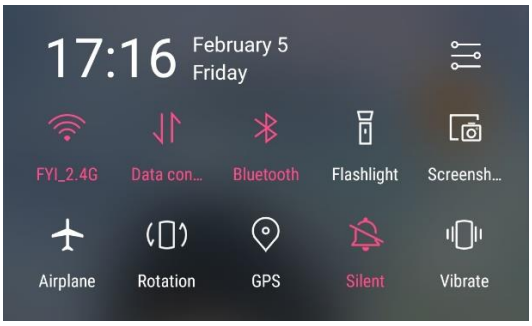
Step1. Upload the code of Project27.2 to ESP32.

Step2. Click on serial monitor.

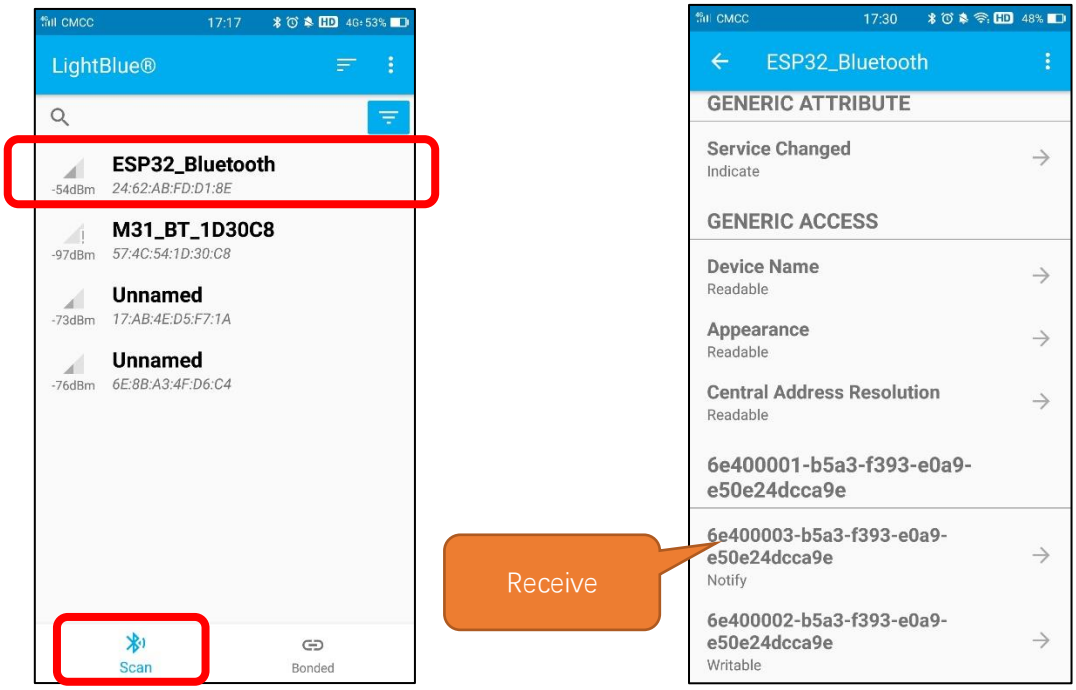
Step3. Set baud rate to 115200.



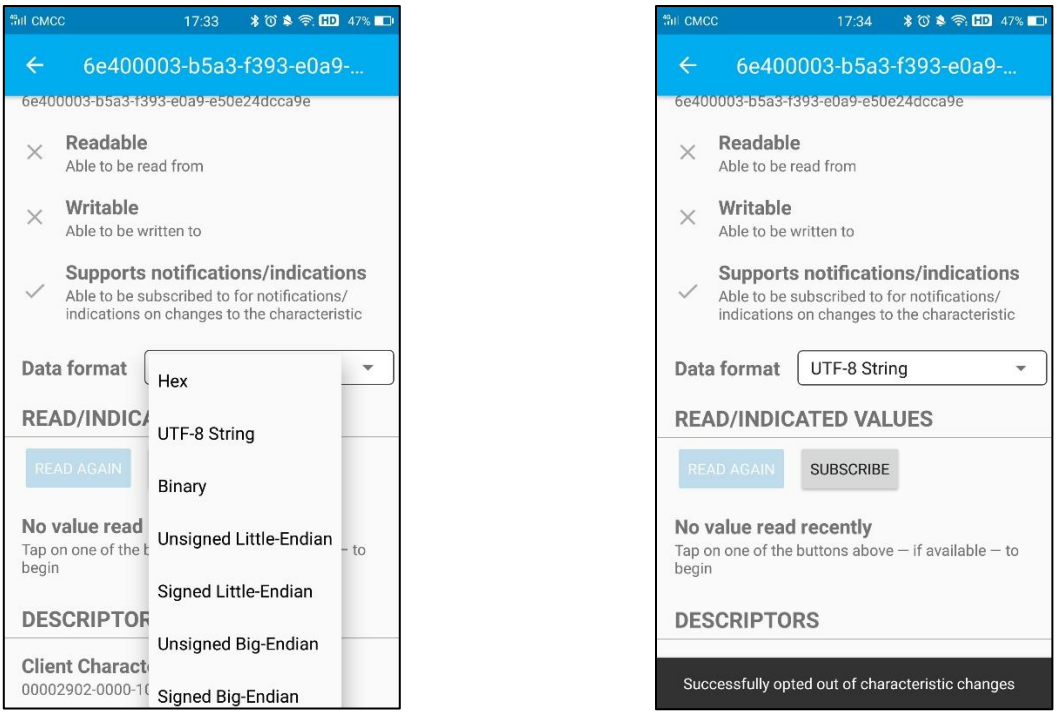
Turn ON Bluetooth on your phone, and open the Lightblue APP.



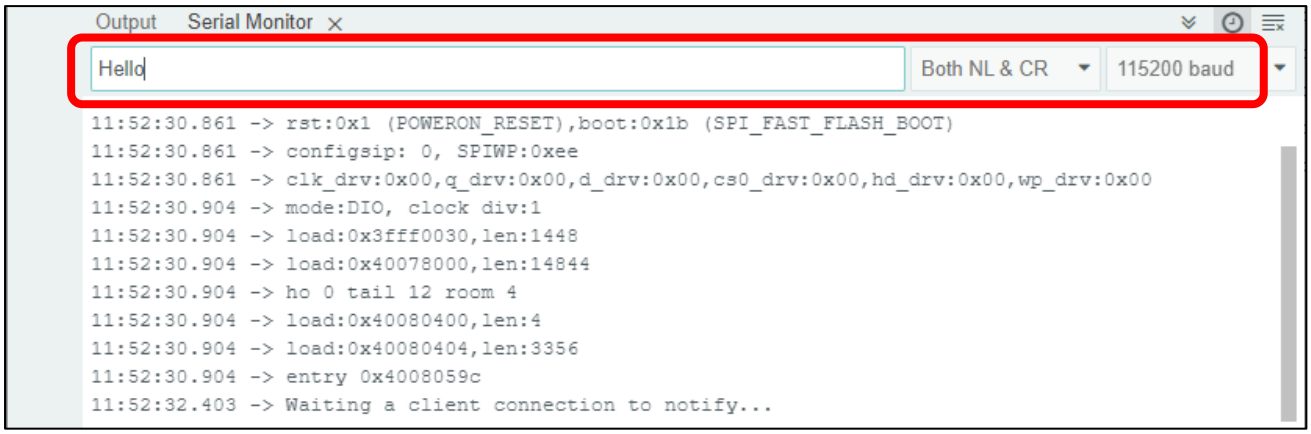
In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32_Bluetooth.



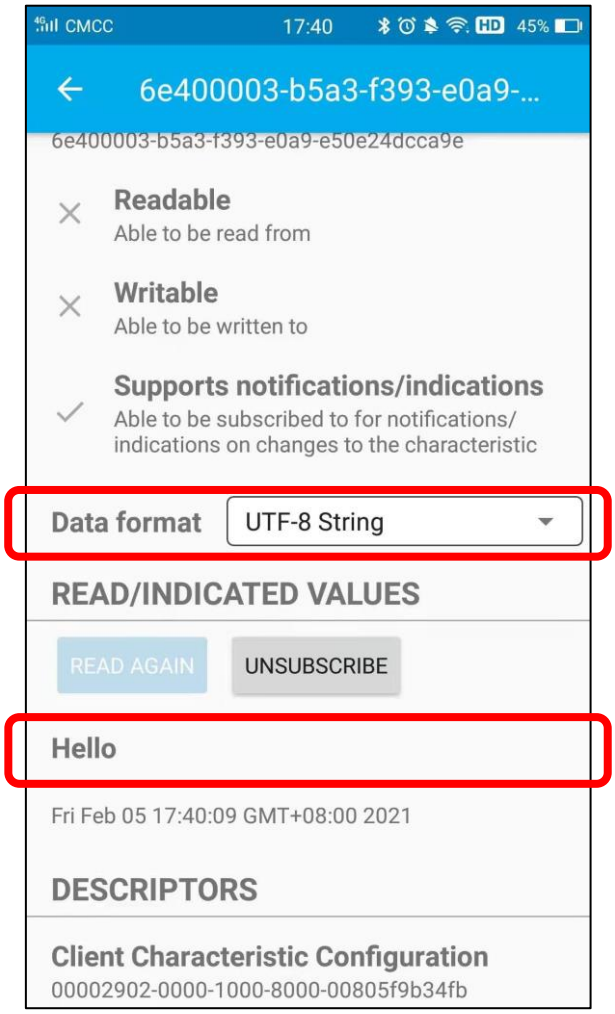
Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



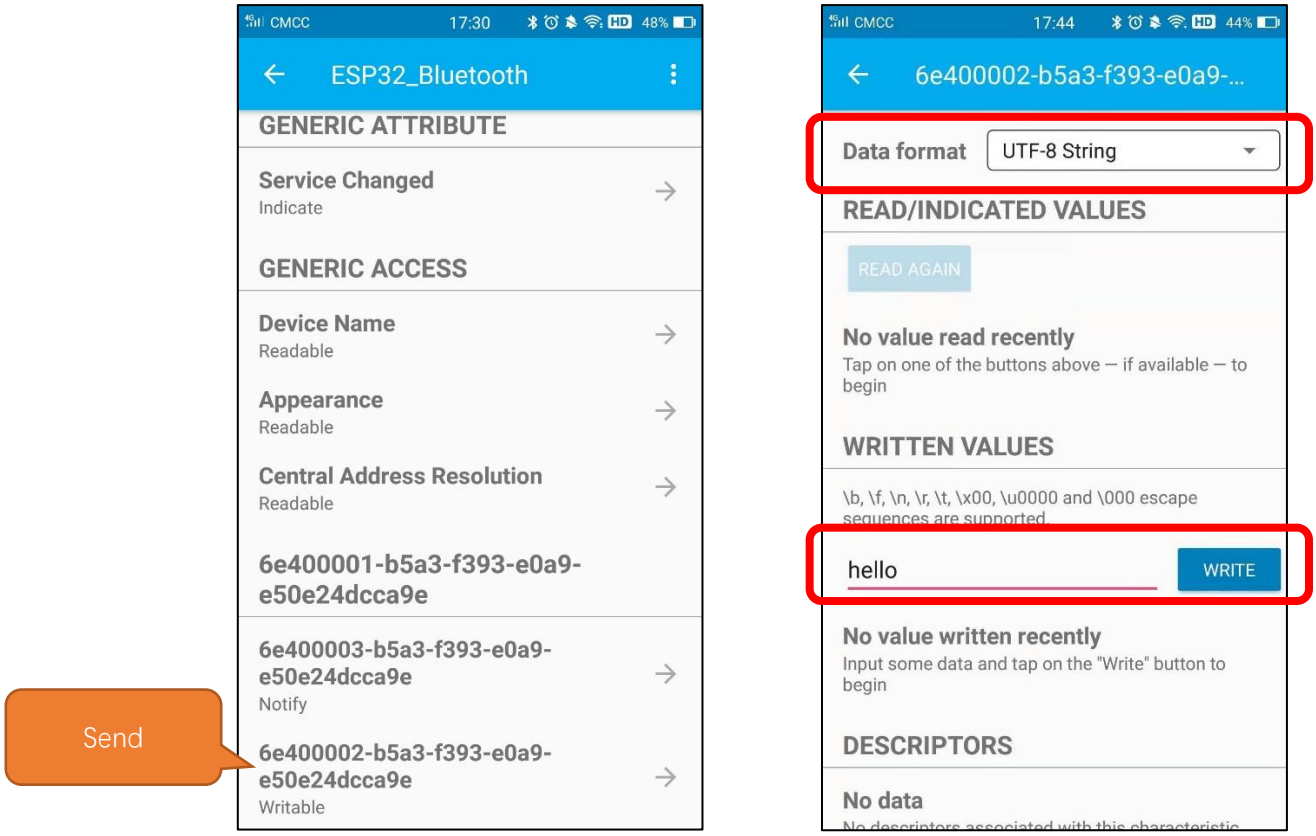
Back to the serial monitor on your computer. You can type anything in the left border of Send, and then click Send.



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());
```

```
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic=
pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
  pCharacteristic->addDescriptor(new BLE2902());
  BLECharacteristic *pCharacteristic =
pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic->setCallbacks(new MyCallbacks());
  pService->start();
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void setup() {
  Serial.begin(9600);
  setupBLE("ESP32_Bluetooth");
}

void loop() {
  long now = millis();
  if (now - lastMsg > 1000) {
    if (deviceConnected&&rxload.length()>0) {
        Serial.println(rxload);
        rxload="";
    }
    if(Serial.available()>0){
        String str=Serial.readString();
        const char *newValue=str.c_str();
        pCharacteristic->setValue(newValue);
        pCharacteristic->notify();
    }
    lastMsg = now;
  }
}
```

Define the specified UUID number for BLE vendor.

```
#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"
```

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

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

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

```
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];
        }
      }
    }
};
```

Initialize the BLE function and name it.

```
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.

```
  long now = millis();
  if (now - lastMsg > 1000) {
    if (deviceConnected&&rxload.length()>0) {
        Serial.println(rxload);
        rxload="";
    }
    if(Serial.available()>0){
        String str=Serial.readString();
        const char *newValue=str.c_str();
        pCharacteristic->setValue(newValue);
        pCharacteristic->notify();
    }
    lastMsg = now;
  }
```

The design for creating the BLE server is:

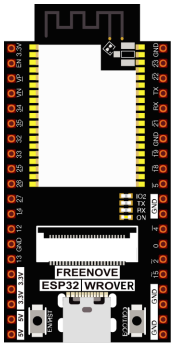
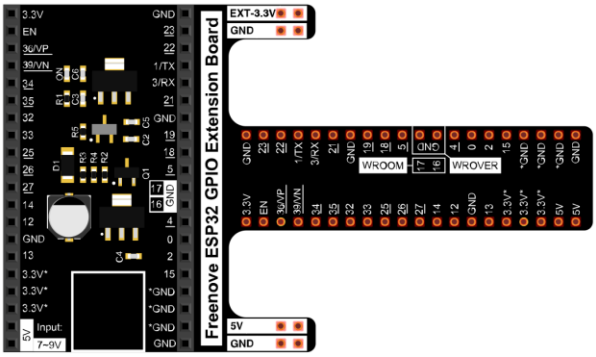
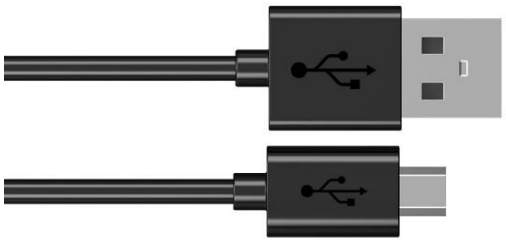
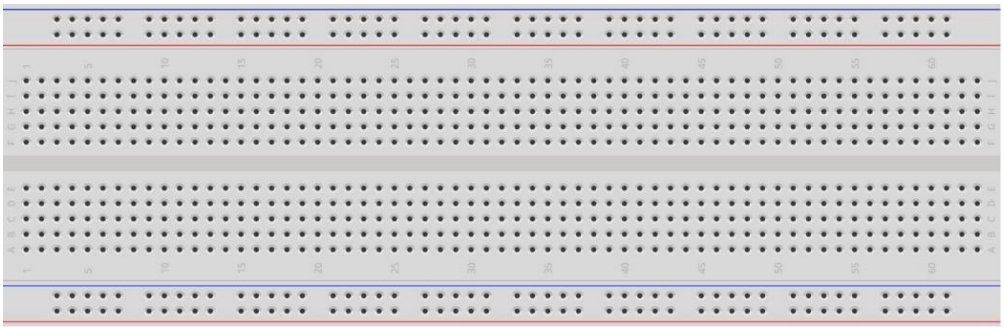
1. Create a BLE Server
2. Create a BLE Service
3. Create a BLE Characteristic on the Service
4. Create a BLE Descriptor on the characteristic
5. Start the service.
6. Start advertising.

```
void setupBLE(String BLEName){
  const char *ble_name=BLEName.c_str();
  BLEDevice::init(ble_name);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic=
pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
  pCharacteristic->addDescriptor(new BLE2902());
  BLECharacteristic *pCharacteristic =
pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic->setCallbacks(new MyCallbacks());
  pService->start();
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}
```

Project 27.3 Bluetooth Control LED

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

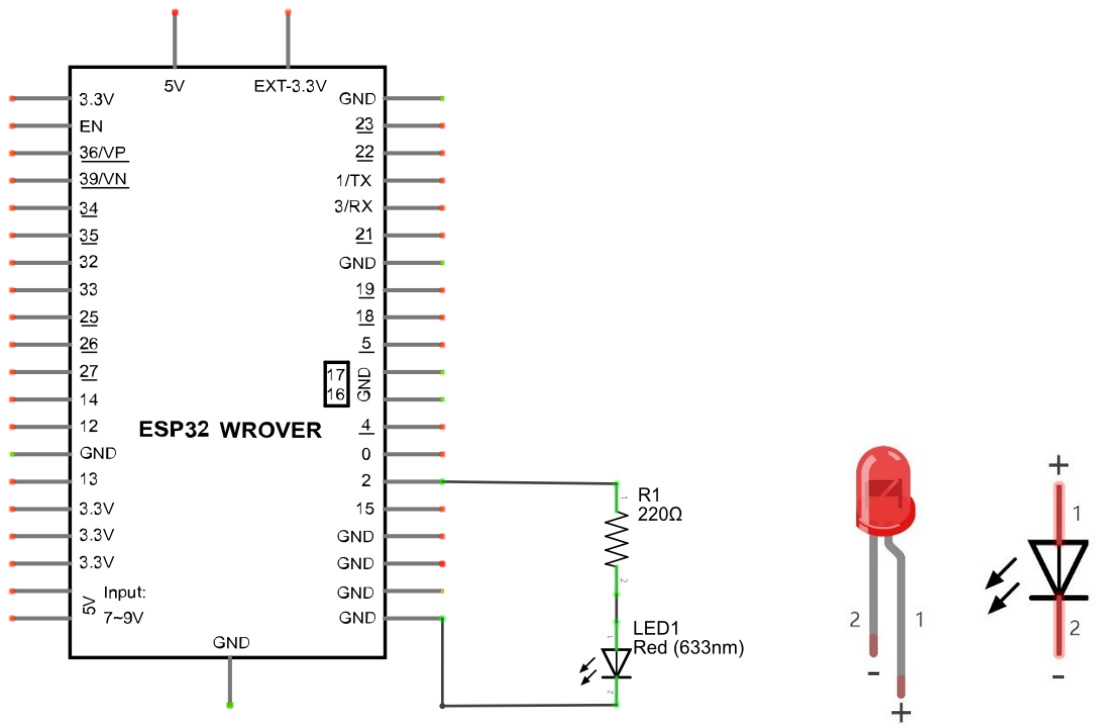
Component List

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

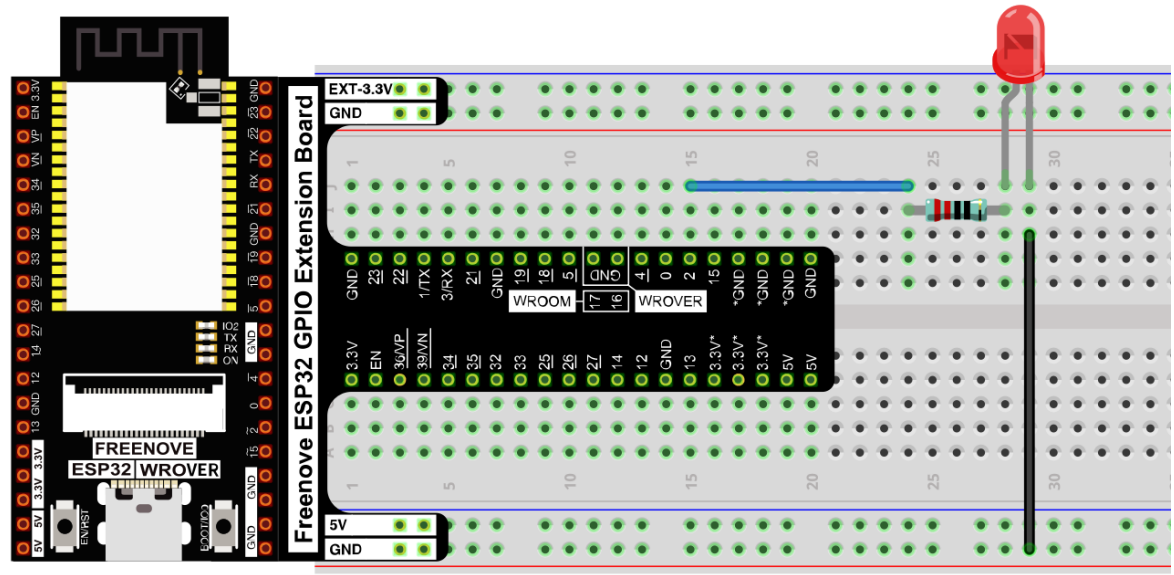
Circuit

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_27.3_Bluetooth_Control_LED

File Edit Sketch Tools Help

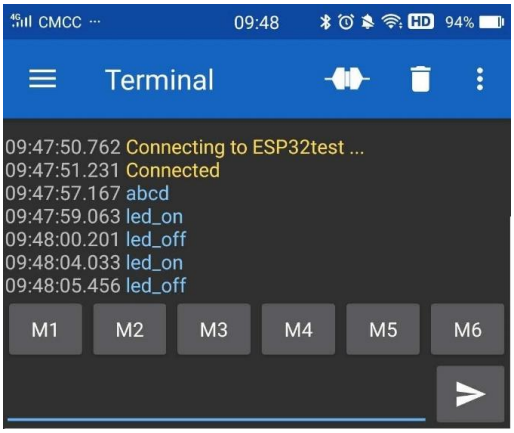
ESP32 Wrover Module

Sketch_27.3_BluetoothToLed.ino

```
#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.

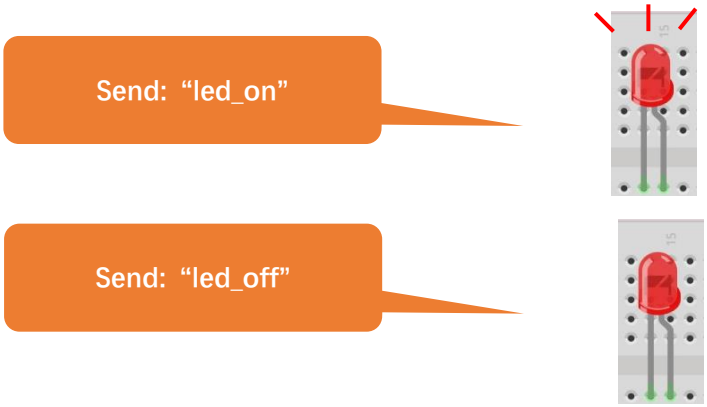
Data sent from mobile APP:



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.

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

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.

```
  if(strncmp(buffer,"led_on",6)==0){
    digitalWrite(LED,LOW);
  }
  if(strncmp(buffer,"led_off",7)==0){
    digitalWrite(LED,HIGH);
  }
```

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

```
  count=0;
  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

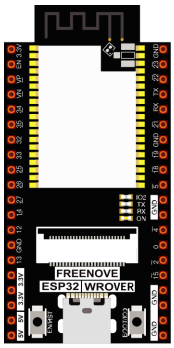
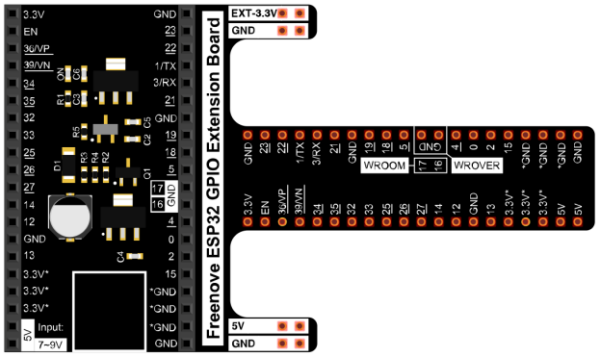
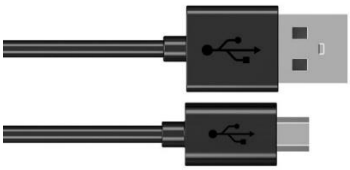
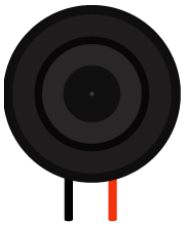
Chapter 28 Bluetooth Media by DAC

ESP32 integrates Classic Bluetooth and Bluetooth Low Energy(BLE). It can transmit not only simple data and orders, but also files including texts and audios. In this section, we will utilize the audio's receiving function of Bluetooth to receive music from mobile phones and play it.

Project 28.1 Playing Bluetooth Music through DAC

Use the Bluetooth audio receiving function of ESP32 to transcode the audio data from mobile phones and play the music through DAC output pin.
The accuracy of ESP32's DAC is only eight bits, so the music would be distorted to some extent using this tutorial. In order to highlight the difference between having and not having an iis decoder, we recommend learning Chapter 29.

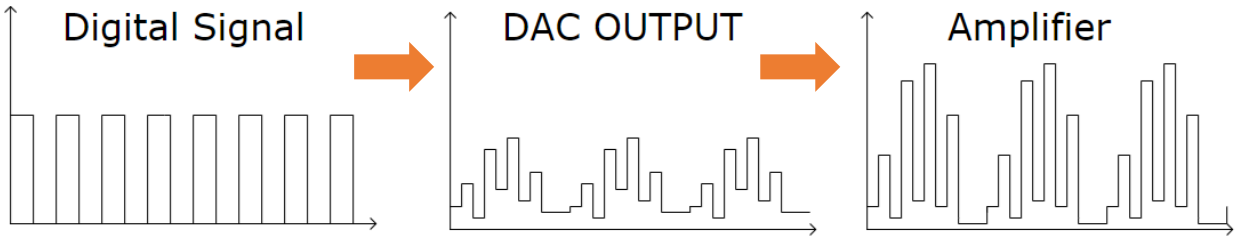
Component List

ESP32-WROVER x1 		GPIO Extension Board x1 	
Micro USB Wire x1 		Speaker 	
NPN transistorx1 (S8050)	Diode x1	Resistor 1kΩ x1	Capacitor 10uF x1 (Optional)

Component knowledge

signal conversion

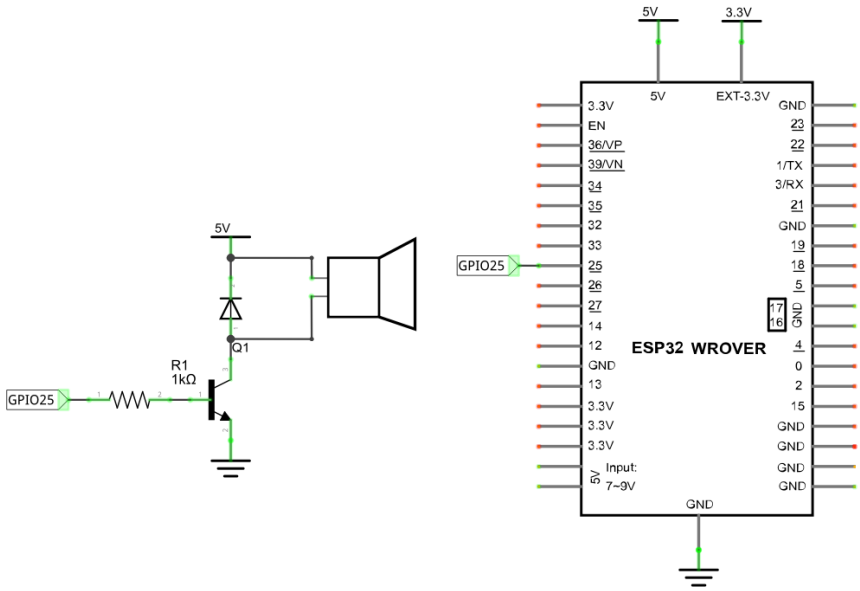
Bluetooth devices receive music data from mobile devices, which cannot play through earphones and speakers directly. To output DAC signal, Bluetooth devices need to decode these data with I2S decoding chip. The power of these audio signals is so small that it can only drives low-power music listening devices, such as earphone. Amplify the power of these DAC signals with power amplifier chip, so that it can drive relatively bigger-power music playing devices, such as speakers.

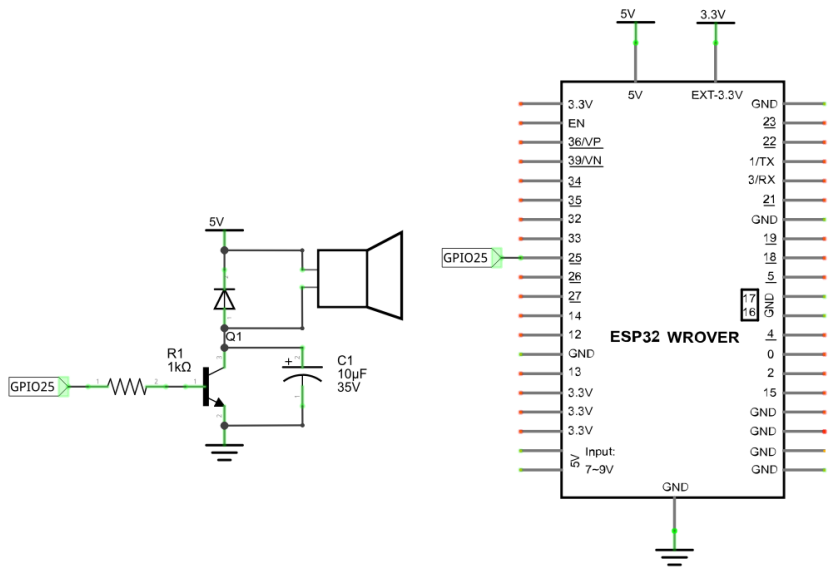


In this chapter, we use ESP32-WROVER's built-in audio decoding feature to convert the received Bluetooth data directly into an audio signal and output it via ESP32-WROVER's DAC pin.

Circuit

Schematic diagram

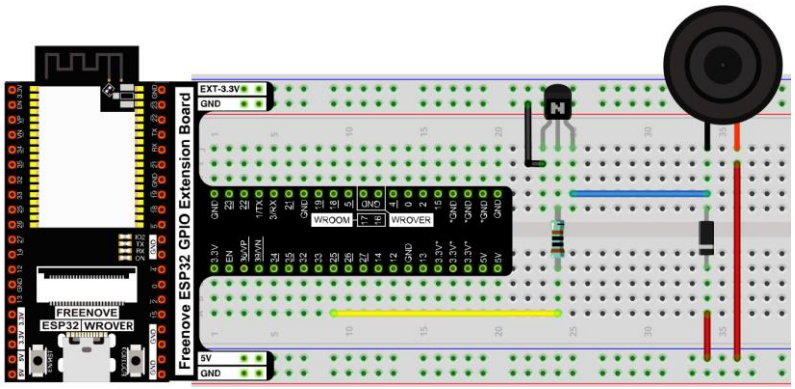


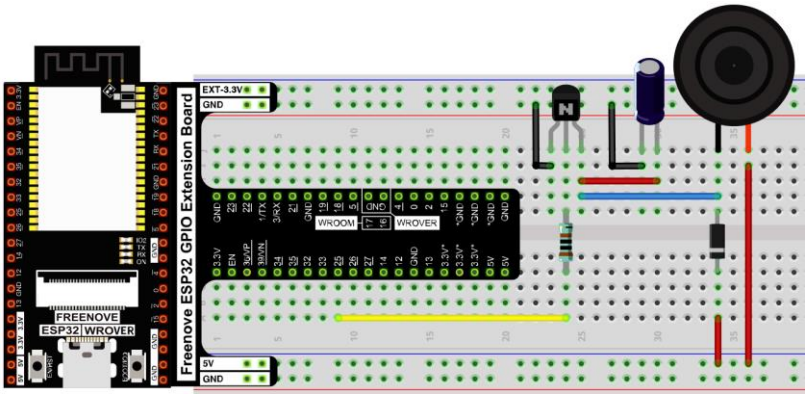


Please note that some kits do not include the 10uF capacitor.

Even if you do not have the capacitor in your kit, you can still build the circuit without it, which won't affect the function. If the sound is too low to hear, please try holding the speaker closer to your ear.

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





Please note that some kits do not include the 10uF capacitor.

Even if you do not have the capacitor in your kit, you can still build the circuit without it, which won't affect the function. If the sound is too low to hear, please try holding the speaker closer to your ear.

Sketch

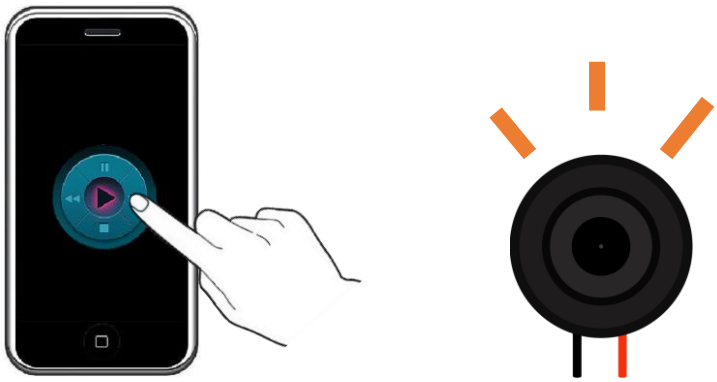
Sketch_28.1_Bluetooth_Music_by_DAC

```
File Edit Sketch Tools Help
ESP32 Wrover Module
Sketch_28.1_BluetoothByDAC.ino  bt_app_av.cpp  bt_app_av.h  bt_app_core.cpp  bt_app_core.h
  err = nvs_flash_init();
}
i2s_config_t i2s_config;
i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_PDM);
i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_24BIT;
i2s_config.sample_rate = 44100;
i2s_config.channel_format = I2S_CHANNEL_FMT_ALL_LEFT;
i2s_config.communication_format = I2S_COMM_FORMAT_STAND_I2S;
i2s_config.intr_alloc_flags = 0;
i2s_config.dma_buf_count = 6;
i2s_config.dma_buf_len = 60;
i2s_config.tx_desc_auto_clear = true;
i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);
i2s_set_pin(I2S_NUM_0, NULL);

esp_bluedroid_init();
esp_bluedroid_enable();
bt_app_task_start_up();
bt_app_work_dispatch(bt_av_hdl_stack_evt, 0, NULL, 0, NULL);
Serial.println("Please use your bluetooth device to connect the ESP32");
```

Compile and upload the code to the ESP32-WROVER and open the serial monitor. ESP32 takes a few seconds to initialize the program. When you see the prompt as shown in the figure below, it means that the Bluetooth function of ESP32 is ready and waiting for the connection of other Bluetooth devices.

```
14:05:33.780 -> rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
14:05:33.816 -> configsip: 0, SPIWP:0xee
14:05:33.816 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
14:05:33.816 -> mode:DIO, clock div:1
14:05:33.816 -> load:0x3fff0030,len:1448
14:05:33.816 -> load:0x40078000,len:14844
14:05:33.816 -> ho 0 tail 12 room 4
14:05:33.816 -> load:0x40080400,len:4
14:05:33.816 -> load:0x40080404,len:3356
14:05:33.816 -> entry 0x4008059c
14:05:35.134 -> Init seccess!
14:05:35.134 -> Please use your bluetooth device to connect the ESP32
```

Please use your mobile phone to search and connect a Bluetooth device named "ESP32". After the connection is successful, you can use ESP32 to play the audio files in your mobile phone.



The following is the program code:

```
#include "BluetoothSerial.h"
#include "driver/i2s.h"
#include "nvs.h"
#include "nvs_flash.h"
#include "esp_bt.h"
#include "bt_app_core.h"
#include "bt_app_av.h"
#include "esp_bt_main.h"
#include "esp_bt_device.h"
#include "esp_gap_bt_api.h"
#include "esp_a2dp_api.h"
#include "esp_avrc_api.h"

#define CONFIG_CLASSIC_BT_ENABLED
#define CONFIG_BT_ENABLED
#define CONFIG_BLUEDROID_ENABLED

BluetoothSerial SerialBT;

static void bt_av_hdl_stack_evt(uint16_t event, void *p_param){
  if(event==0) {
    /* initialize A2DP sink */
    esp_a2d_register_callback(&bt_app_a2d_cb);
    esp_a2d_sink_register_data_callback(bt_app_a2d_data_cb);
    esp_a2d_sink_init();
    /* initialize AVRCP controller */
    esp_avrc_ct_init();
    esp_avrc_ct_register_callback(bt_app_rc_ct_cb);
    /* set discoverable and connectable mode, wait to be connected */
    esp_bt_gap_set_scan_mode(ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE);
  }
}

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32");
  Serial.println("Init seccess! ");

  esp_err_t err = nvs_flash_init();
  if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    err = nvs_flash_init();
  }
```

```
  i2s_config_t i2s_config;
  i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN);
  i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_24BIT;
  i2s_config.sample_rate = 44100;
  i2s_config.channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT; //2-channels
  i2s_config.communication_format = I2S_COMM_FORMAT_I2S_MSB;
  i2s_config.intr_alloc_flags = 0;
  i2s_config.dma_buf_count = 6;
  i2s_config.dma_buf_len = 60;
  i2s_config.tx_desc_auto_clear = true;
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
  i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);
  i2s_set_pin(I2S_NUM_0, NULL);

  esp_bluedroid_init();
  esp_bluedroid_enable();
  bt_app_task_start_up();
  bt_app_work_dispatch(bt_av_hdl_stack_evt, 0, NULL, 0, NULL);
  Serial.println("Please use your Bluetooth device to connect the ESP32! ");
}

void loop(){
  ;
}
```

Add program files related to Bluetooth and API interface files.

```
#include "BluetoothSerial.h"
#include "driver/i2s.h"
#include "nvs.h"
#include "nvs_flash.h"
#include "esp_bt.h"
#include "bt_app_core.h"
#include "bt_app_av.h"
#include "esp_bt_main.h"
#include "esp_bt_device.h"
#include "esp_gap_bt_api.h"
#include "esp_a2dp_api.h"
#include "esp_avrc_api.h"
```

Set the Bluetooth in slave mode through macro definition and use it to receive data from other devices.

```
#define CONFIG_CLASSIC_BT_ENABLED
#define CONFIG_BT_ENABLED
#define CONFIG_BLUEDROID_ENABLED
```

Initialize the serial port and set the baud rate to 115200; initialize Bluetooth and name it as "ESP32".

```
  Serial.begin(115200);
  SerialBT.begin("ESP32");
  Serial.println("Init seccess! ");
```

Define an I2S interface variable and initialize it.

```
  i2s_config_t i2s_config;
  i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN);
  i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_24BIT;
  i2s_config.sample_rate = 44100;         // sampling frequency of audio data
  i2s_config.channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT;//Left and right channels
  i2s_config.communication_format = I2S_COMM_FORMAT_I2S_MSB;
  i2s_config.intr_alloc_flags = 0;          //Set interrupt priority
  i2s_config.dma_buf_count = 6;         //Set up DMA data partitions
  i2s_config.dma_buf_len = 60;          //Set the DMA capacity for each partition
  i2s_config.tx_desc_auto_clear = true;    //Set automatic clearing of DMA data
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);//Initialize I2S function
  i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);//Set the DAC to dual channel mode
  i2s_set_pin(I2S_NUM_0, NULL);                 //Set the output pin as GPIO25
```

Initialize the Bluetooth hardware device, establish a Bluetooth thread task, and print out messages to prompt the user to take the next step.

```
  esp_bluedroid_init();
  esp_bluedroid_enable();
  bt_app_task_start_up();
  bt_app_work_dispatch(bt_av_hdl_stack_evt, 0, NULL, 0, NULL);
  Serial.println("Please use your Bluetooth device to connect the ESP32! ");
```

Bluetooth thread task: Set Bluetooth to slave mode; initialize Bluetooth command resolution function; set Bluetooth to be visible to other devices and in waiting for connection mode.

```
static void bt_av_hdl_stack_evt(uint16_t event, void *p_param){
  if(event==0) {
    /* initialize A2DP sink */
    esp_a2d_register_callback(&bt_app_a2d_cb);
    esp_a2d_sink_register_data_callback(bt_app_a2d_data_cb);
    esp_a2d_sink_init();
    /* initialize AVRCP controller */
    esp_avrc_ct_init();
    esp_avrc_ct_register_callback(bt_app_rc_ct_cb);
    /* set discoverable and connectable mode, wait to be connected */
    esp_bt_gap_set_scan_mode(ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE);
  }
}
```

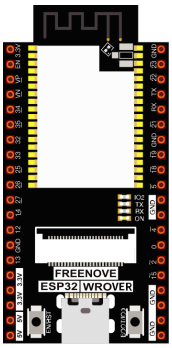
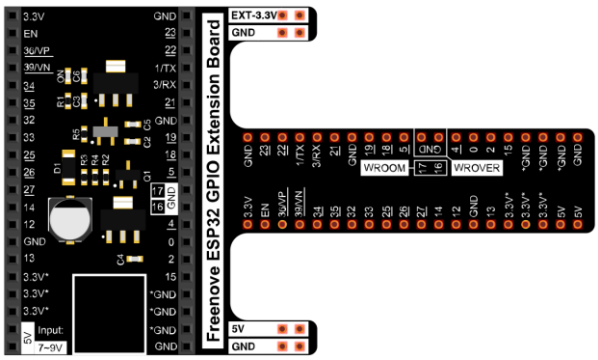
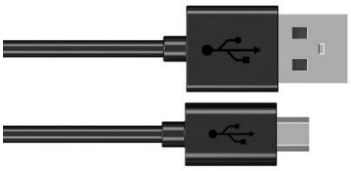
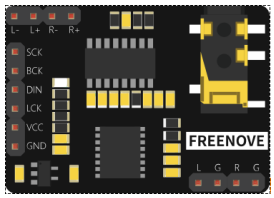
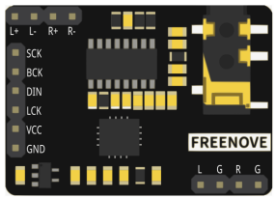
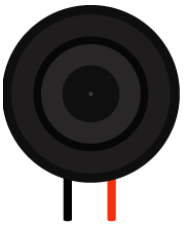
Chapter 29 Bluetooth Media by Audio Module

In the previous section, ESP32's own DAC was used to output the audio signal, and obviously there was some distortion in the sound quality. In this section, with the help of Audio Converter & Amplifier, you will enjoy higher-quality music.

Project 29.1 Playing Bluetooth Music through Audio Converter & Amplifier

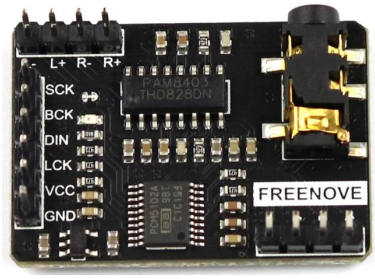
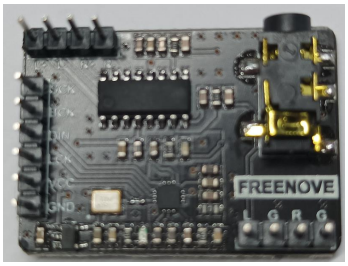
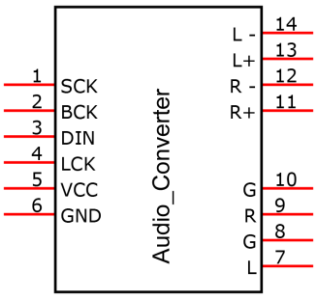
In this project, we will use Audio Converter & Amplifier to transcode audio data into stereo and output it.

Component List

ESP32-WROVER x1 	GPIO Extension Board x1 
Micro USB Wire x1 	Audio Converter & Amplifier  or 
Speaker 	
Jumper F/M x4 Jumper F/F x2	

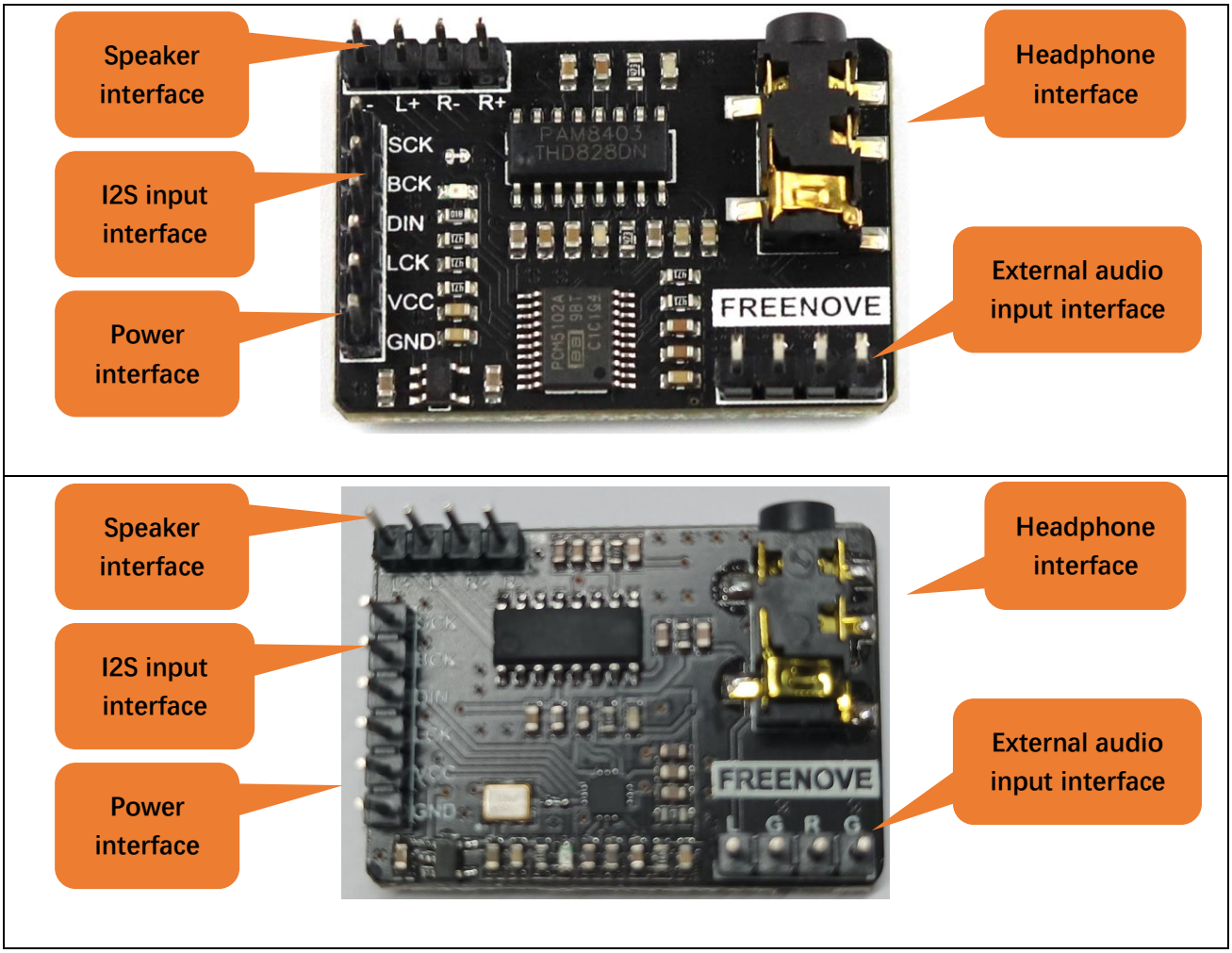
Component knowledge

The front and reverse view of Audio Converter & Amplifier module.

front view	reverse view	schematic diagram
 or 	or	

Interface description for Audio Converter & Amplifier module

Pin	Name	Introductions
1	SCK	System clock input
2	BCK	Audio data bit clock input
3	DIN	Audio data input
4	LCK	Audio data word clock input
5	VCC	Power input，3.3V~5.0V
6	GND	Power Ground
7	L	External audio left channel input
8	G	Power Ground
9	R	External audio right channel input
10	G	Power Ground
11	R+	Positive pole of right channel horn
12	R-	Negative pole of right channel horn
13	L+	Positive pole of left channel horn
14	L-	Negative pole of left channel horn



Speaker interface: Connect left channel speaker and right channel speaker. Group L: L+ & L-; Group R: R+& R-. The two interfaces of the speaker can be connected to the interfaces of group L or group R. But when one interface is connected to group L, the other cannot be connected to group R. Doing so may cause the module to malfunction.

Headphone interface: the interface to connect the headphones.

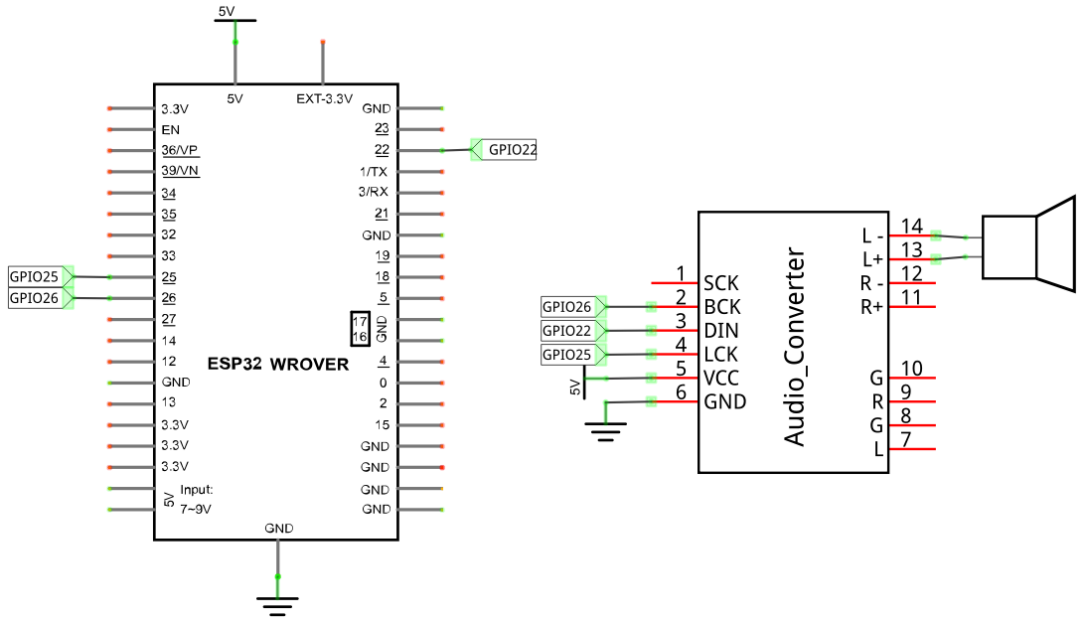
I2S input interface: connect to the device with I2S. Used to transcode audio data into DAC audio signals.

External audio input interface: connect to external audio equipment. Used to amplify externally input audio signals.

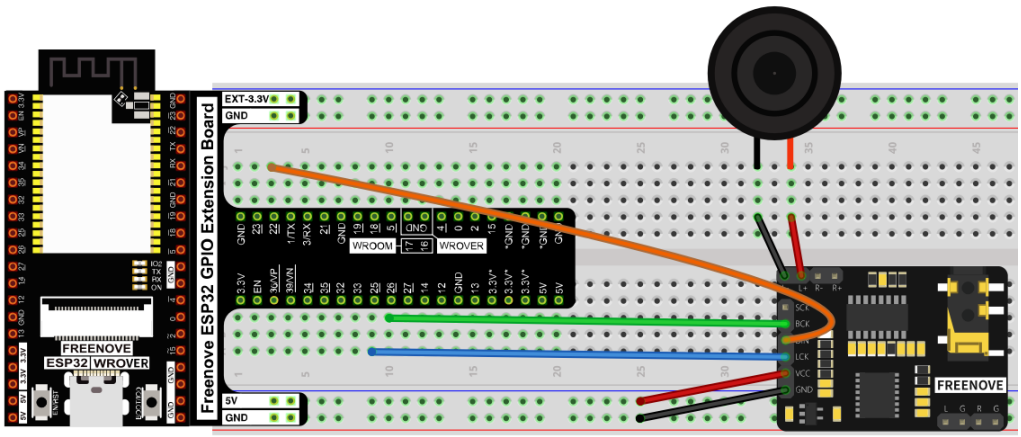
Power interface: connect to external power supply. External power supply selection range: 3.3V-5.0V.

Circuit

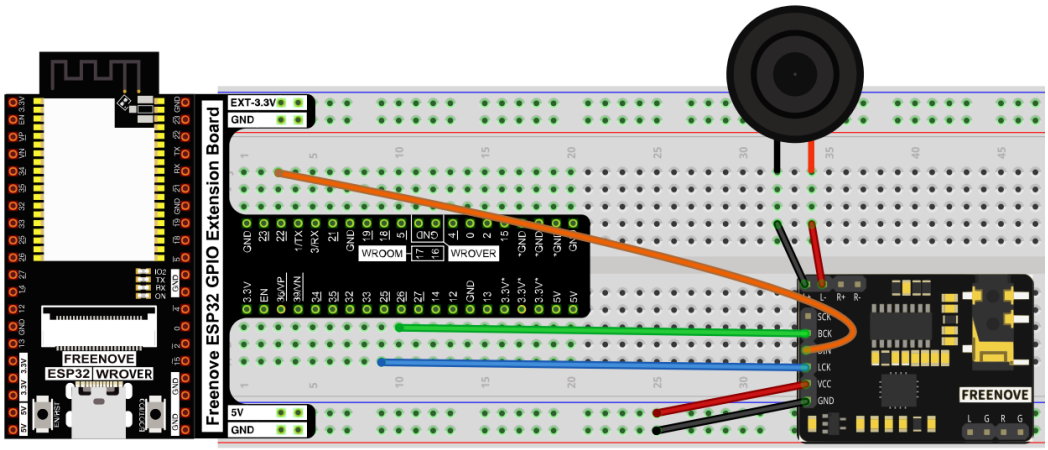
Schematic diagram



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



or



Sketch

Sketch_29.1_Bluetooth_Music_by_ Audio_Converter

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_29.1_BluetoothByPCM5102A.ino | bt_app_av.cpp | bt_app_av.h | bt_app_core.cpp | bt_app_core.h

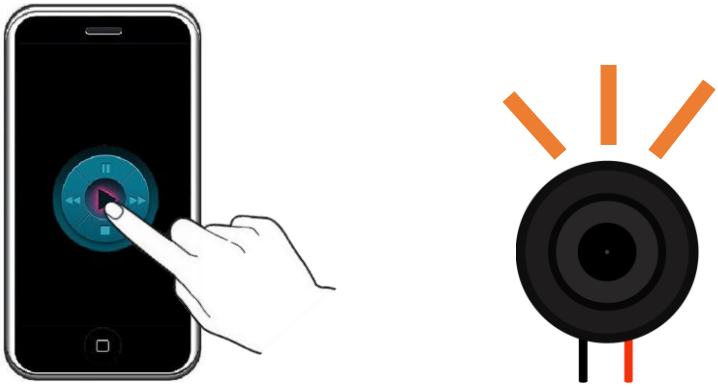
```
#include "BluetoothSerial.h"
#include "driver/i2s.h"
#include "nvs.h"
#include "nvs_flash.h"

#include "esp_bt.h"
#include "bt_app_core.h"
#include "bt_app_av.h"
#include "esp_bt_main.h"
#include "esp_bt_device.h"
#include "esp_gap_bt_api.h"
#include "esp_a2dp_api.h"
#include "esp_avrc_api.h"

#define CONFIG_I2S_LRCK_PIN 25
#define CONFIG_I2S_BCK_PIN  26
#define CONFIG_I2S_DATA_PIN 22
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32");
  Serial.println("Init seccess!");

  esp_err_t err = nvs_flash_init();
  if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    err = nvs_flash_init();
  }

  i2s_config_t i2s_config;
  i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX);

  i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT;
  i2s_config.sample_rate = 44100;
  i2s_config.channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT;
  i2s_config.communication_format = I2S_COMM_FORMAT_STAND_MSB;
  i2s_config.intr_alloc_flags = 0;
```

Compile and upload the code to the ESP32-WROVER and open the serial monitor. ESP32 takes a few seconds to initialize the program.

When you see the prompt as shown in the figure below, it means that the Bluetooth function of ESP32 is ready and waiting for the connection of other Bluetooth devices.

```
Output   Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')   Both NL & CR   115200 baud
14:41:56.597 -> ets Jul 29 2019 12:21:46
14:41:56.597 ->
14:41:56.597 -> rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
14:41:56.601 -> configsip: 0, SPIWP:0xee
14:41:56.601 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
14:41:56.601 -> mode:DIO, clock div:1
14:41:56.601 -> load:0x3fff0030,len:1448
14:41:56.601 -> load:0x40078000,len:14844
14:41:56.601 -> ho 0 tail 12 room 4
14:41:56.601 -> load:0x40080400,len:4
14:41:56.601 -> load:0x40080404,len:3356
14:41:56.601 -> entry 0x4008059c
14:41:57.916 -> Init seccess!
14:41:57.916 -> ok
```

Please use your mobile phone to search and connect a Bluetooth device named "ESP32". After the connection is successful, you can use ESP32 to play the audio files in your mobile phone.



The following is the program code:

```
#include "BluetoothSerial.h"
#include "driver/i2s.h"
#include "nvs.h"
#include "nvs_flash.h"
#include "esp_bt.h"
#include "bt_app_core.h"
#include "bt_app_av.h"
#include "esp_bt_main.h"
#include "esp_bt_device.h"
#include "esp_gap_bt_api.h"
#include "esp_a2dp_api.h"
#include "esp_avrc_api.h"

#define CONFIG_I2S_LRCK_PIN 25
#define CONFIG_I2S_BCK_PIN  26
#define CONFIG_I2S_DATA_PIN 22
```

```
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32");
  Serial.println("Init seccess! ");

  esp_err_t err = nvs_flash_init();
  if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    err = nvs_flash_init();
  }

  i2s_config_t i2s_config;
  i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX);
  i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT;
  i2s_config.sample_rate = 44100;
  i2s_config.channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT;
  i2s_config.communication_format = I2S_COMM_FORMAT_I2S_MSB;
  i2s_config.intr_alloc_flags = 0;
  i2s_config.dma_buf_count = 6;
  i2s_config.dma_buf_len = 60;
  i2s_config.tx_desc_auto_clear = true;
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
  i2s_pin_config_t pin_config;
  pin_config.bck_io_num = CONFIG_I2S_BCK_PIN;
  pin_config.ws_io_num = CONFIG_I2S_LRCK_PIN;
  pin_config.data_out_num = CONFIG_I2S_DATA_PIN;
  pin_config.data_in_num = -1;
  i2s_set_pin(I2S_NUM_0, &pin_config);

  bt_app_task_start_up();

 /* initialize A2DP sink */
  esp_a2d_register_callback(&bt_app_a2d_cb);
  esp_a2d_sink_register_data_callback(bt_app_a2d_data_cb);
  esp_a2d_sink_init();
  /* initialize AVRCP controller */
  esp_avrc_ct_init();
  esp_avrc_ct_register_callback(bt_app_rc_ct_cb);
  /* set discoverable and connectable mode, wait to be connected */
  esp_bt_gap_set_scan_mode(ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE);
  Serial.println("Please use your Bluetooth device to connect the ESP32! ");
}
```

```

void loop(){
  ;
}
```

Define an I2S configuration class and initialize it; define an I2S pin configuration class and associate I2S signals with these pins.

```
  i2s_config_t i2s_config;
  i2s_config.mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX);
  i2s_config.bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT;
  i2s_config.sample_rate = 44100;
  i2s_config.channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT;
  i2s_config.communication_format = I2S_COMM_FORMAT_I2S_MSB;
  i2s_config.intr_alloc_flags = 0;
  i2s_config.dma_buf_count = 6;
  i2s_config.dma_buf_len = 60;
  i2s_config.tx_desc_auto_clear = true;
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);//
  i2s_pin_config_t pin_config;
  pin_config.bck_io_num = CONFIG_I2S_BCK_PIN;
  pin_config.ws_io_num = CONFIG_I2S_LRCK_PIN;
  pin_config.data_out_num = CONFIG_I2S_DATA_PIN;
  pin_config.data_in_num = -1;
  i2s_set_pin(I2S_NUM_0, &pin_config);
```

In this chapter, the procedure is very similar to the previous section. The difference only lies in the I2S configuration. The audio quality of the DAC output from the Audio Converter & Amplifier after decoding is significantly higher than that of the ESP32's own 8-bit precision DAC.

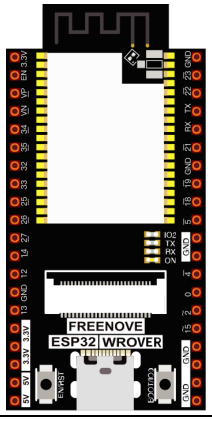
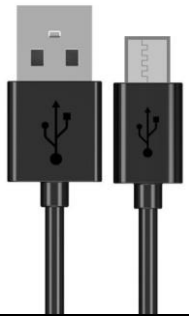
Chapter 30 Read and Write the Sdcard

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 30.1 SDMMC Test

Component List

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

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.

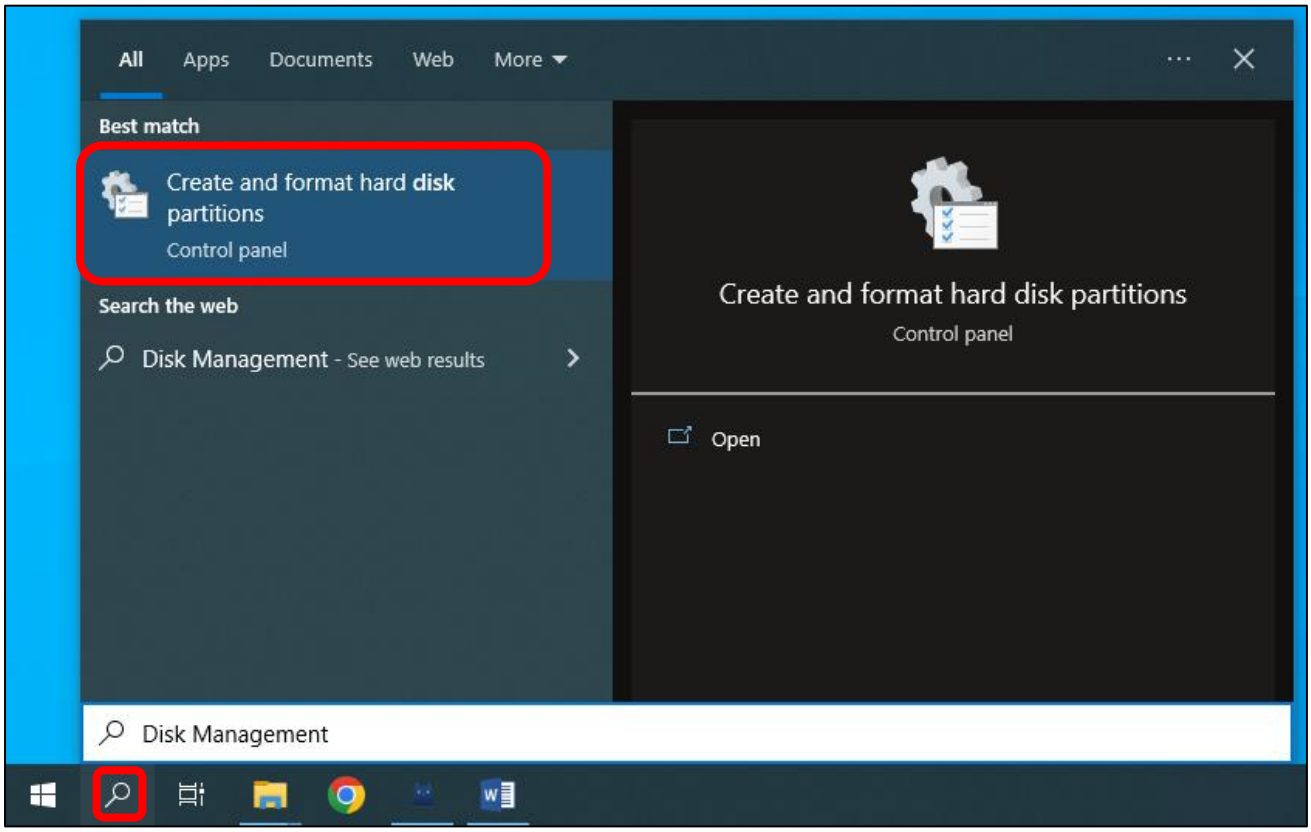
Format SD card

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.

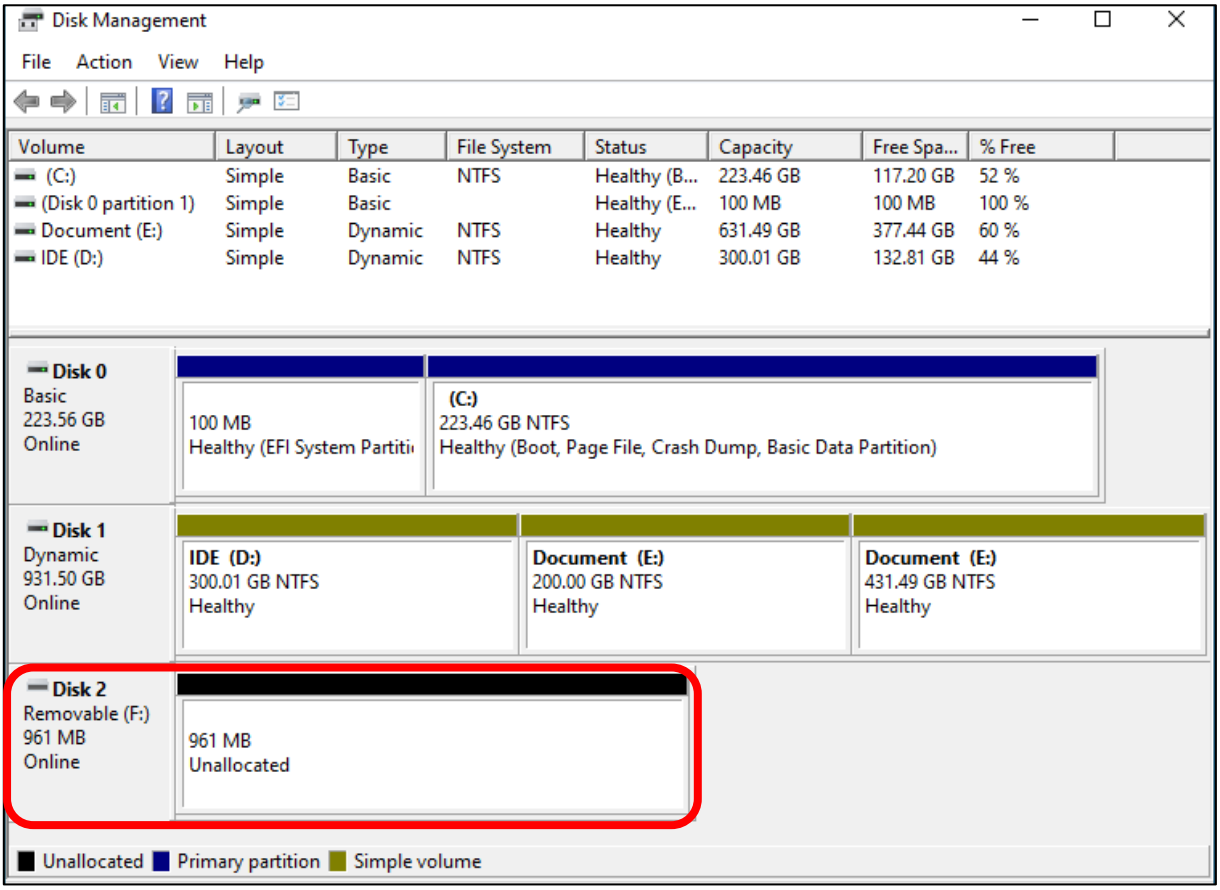
Windows

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

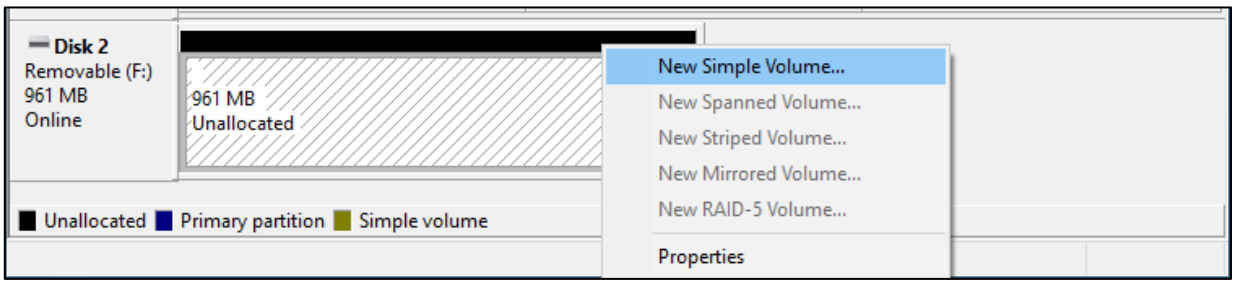
In the Windows search box, enter "Disk Management" and select "Create and format hard disk partitions".



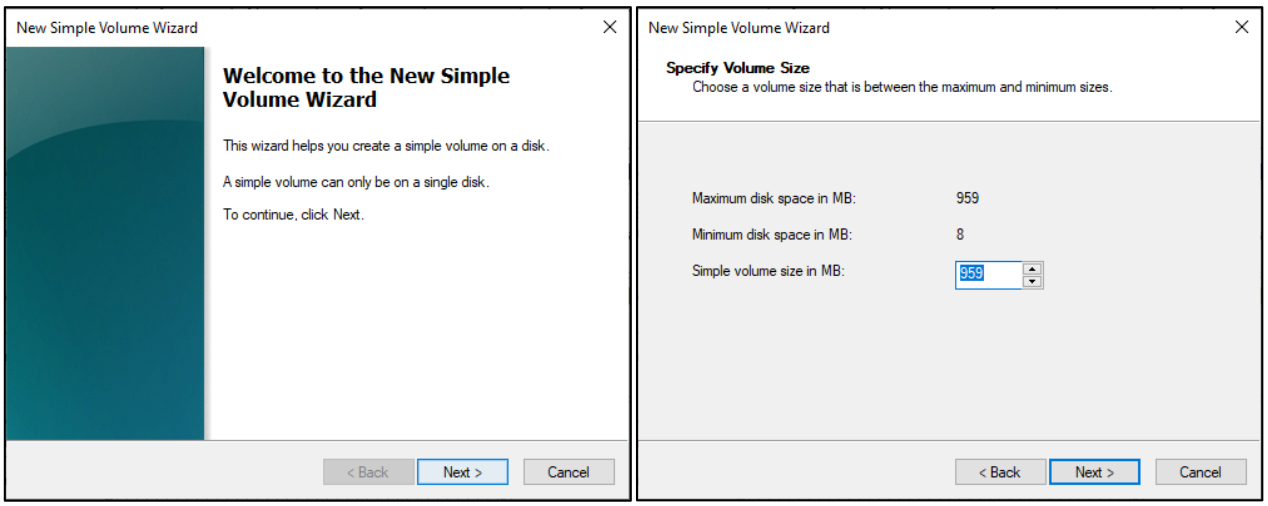
In the new pop-up window, find an unallocated volume close to 1G in size.



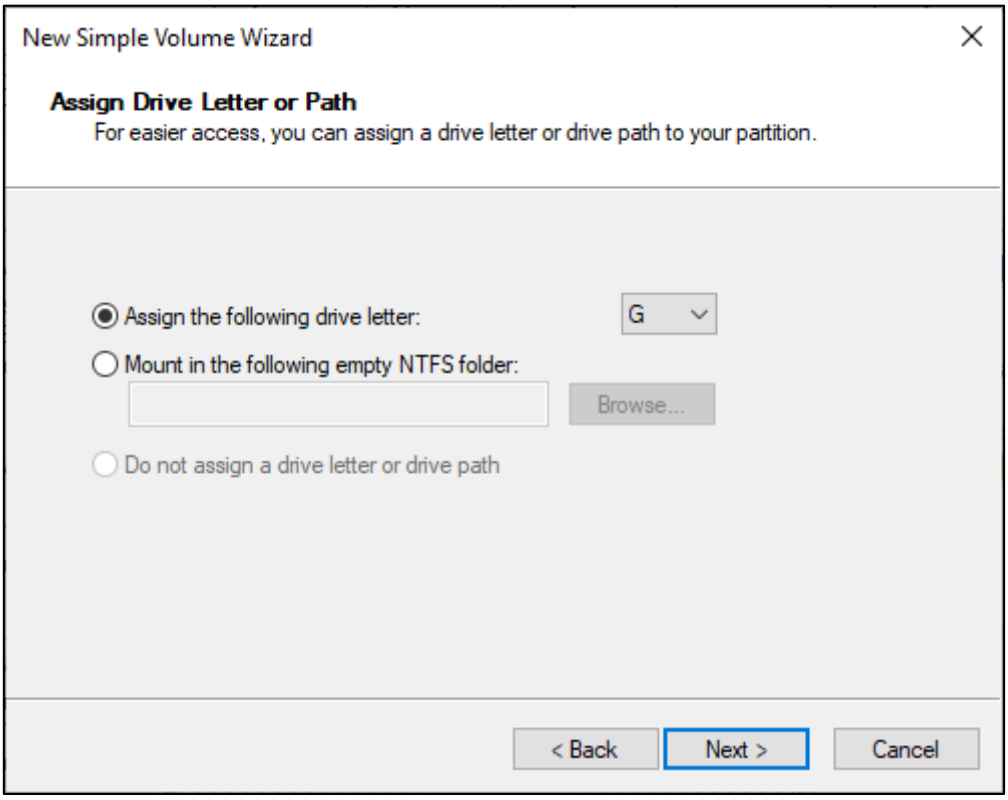
Click to select the volume, right-click and select "New Simple Volume".



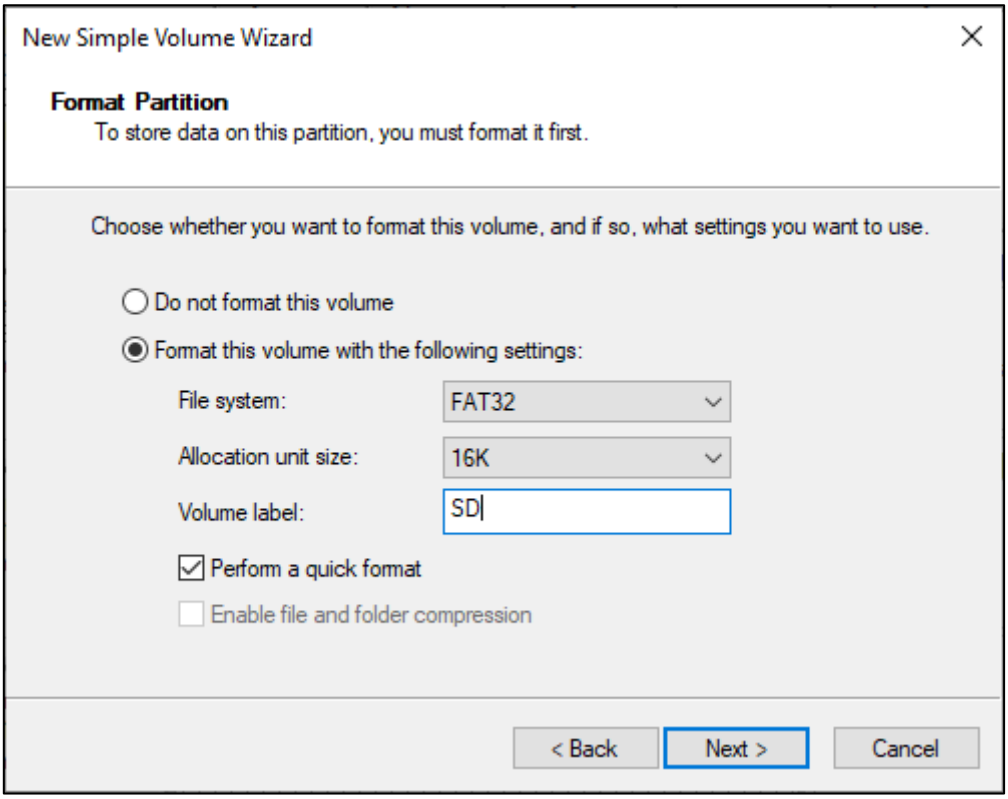
Click Next.



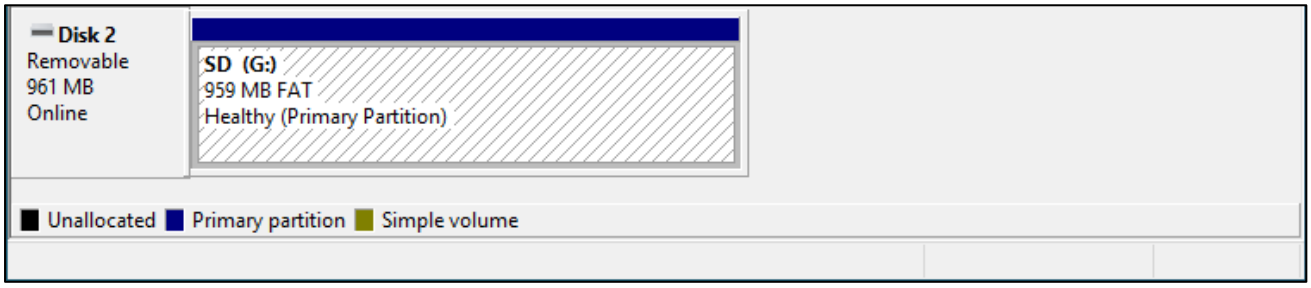
You can choose the drive letter on the right, or you can choose the default. By default, just click Next.



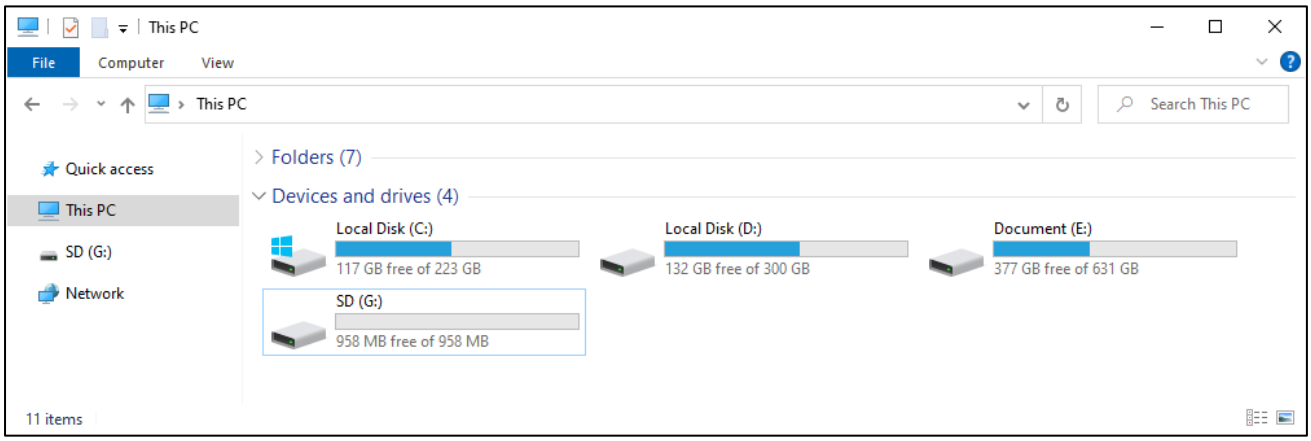
File system is FAT(or FAT32). The Allocation unit size is 16K, and the Volume label can be set to any name. After setting, click Next.



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

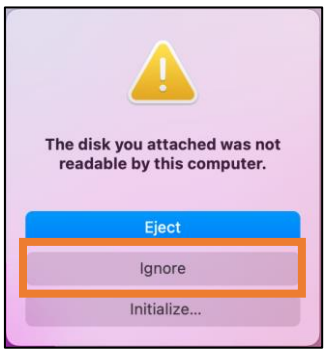


At this point, you can see the SD card in This PC.



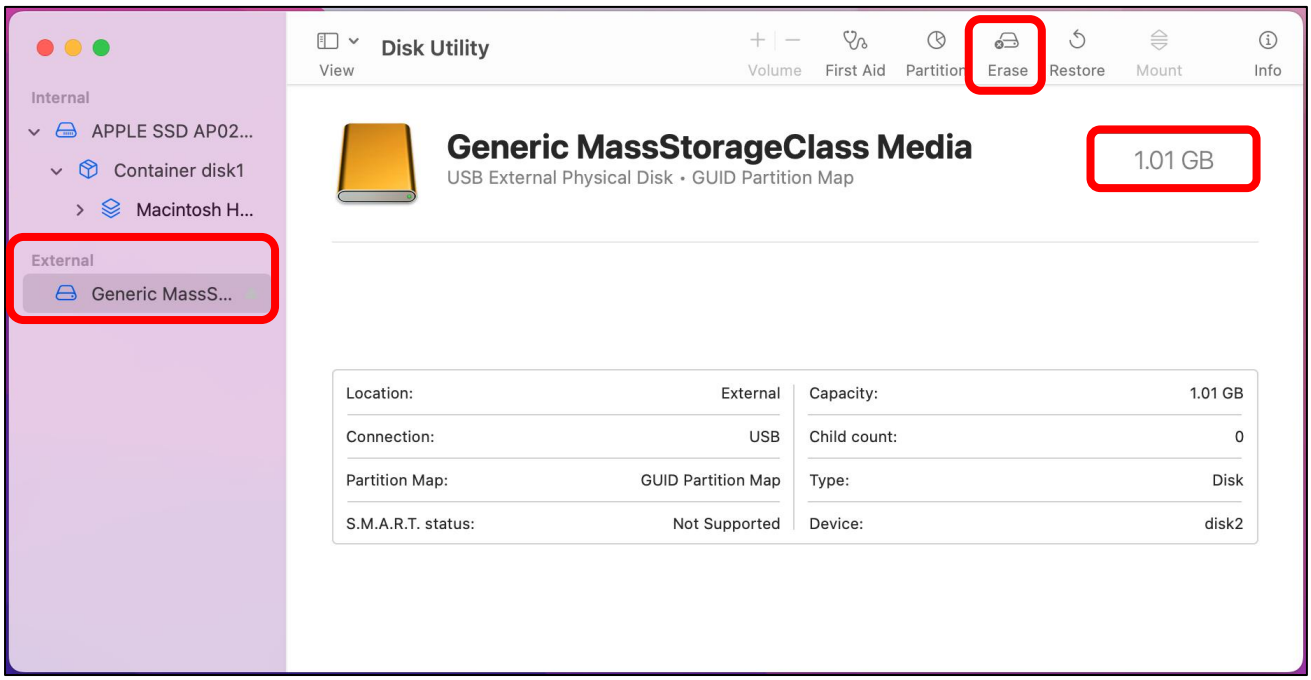
MAC

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.

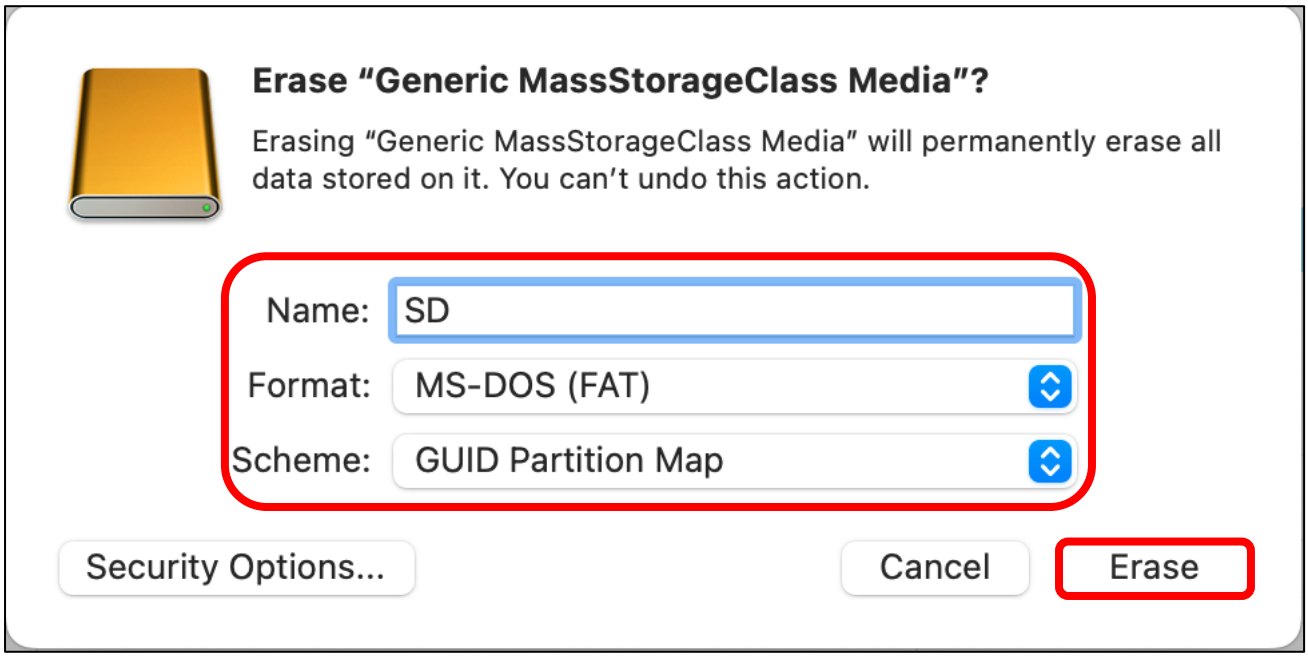


Find "Disk Utility" in the MAC system and click to open it.

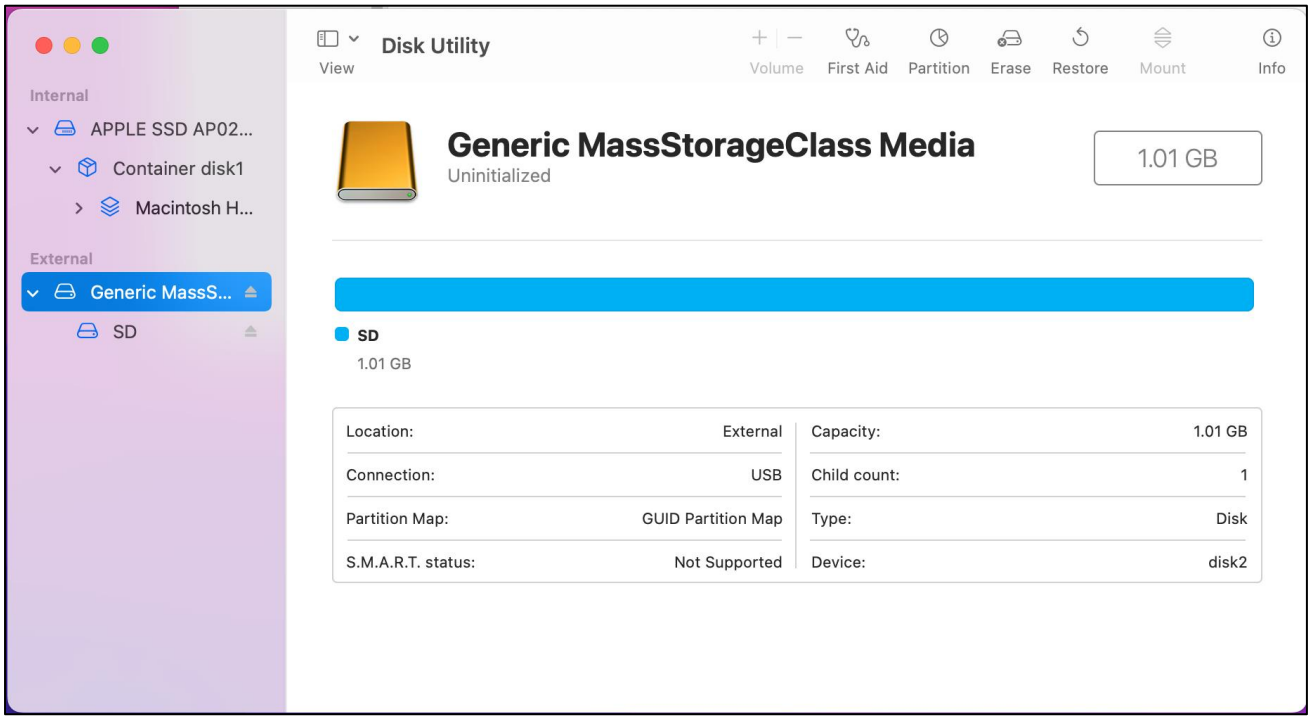
Select "Generic MassStorageClass Media", note that its size is about 1G. Please do not choose wrong item. Click "Erase".



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

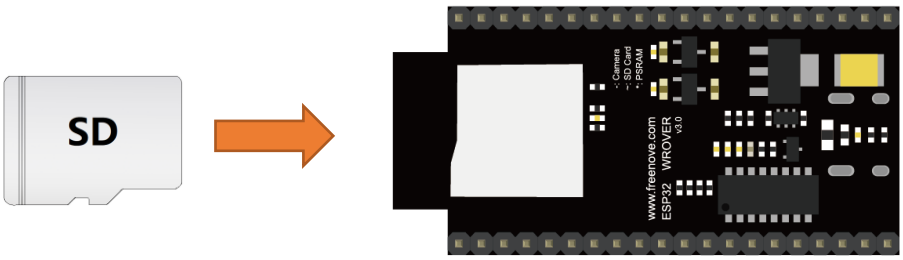


Wait for the formatting to complete. When finished, it will look like the picture below. At this point, you can see a new disk on the desktop named "SD".

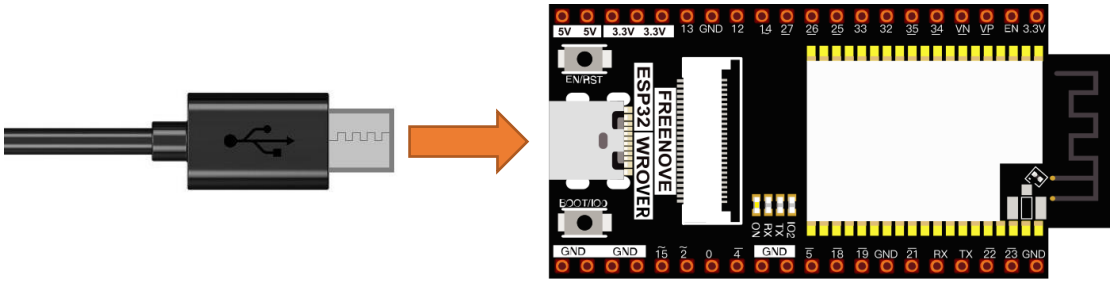


Circuit

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_30.1_SDMMC_Test

```
#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;
    }
```

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");
```

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

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".

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

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

```
createDir(SD_MMC, "/mydir");
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.

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

Call deleteFile() to delete a specified file.

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

```
    deleteFile(SD_MMC, "/foo.txt");
    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.

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

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

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

Chapter 31 Play SD card music

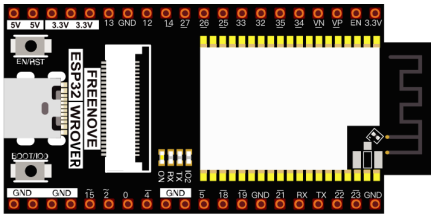
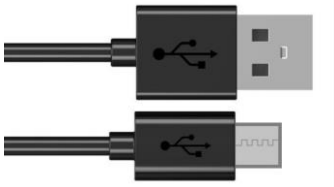
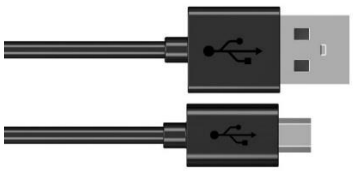
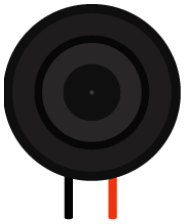
Note: The SD card chapter only applies to the ESP32 WROVER development board with an SD card slot on the back. If your ESP32 WROVER does not have an SD card slot on the back, please skip this chapter.

In the previous study, we have learned how to use the SD card, and then we will learn to play the music in the SD card.

Project 31.1 SDMMC Music

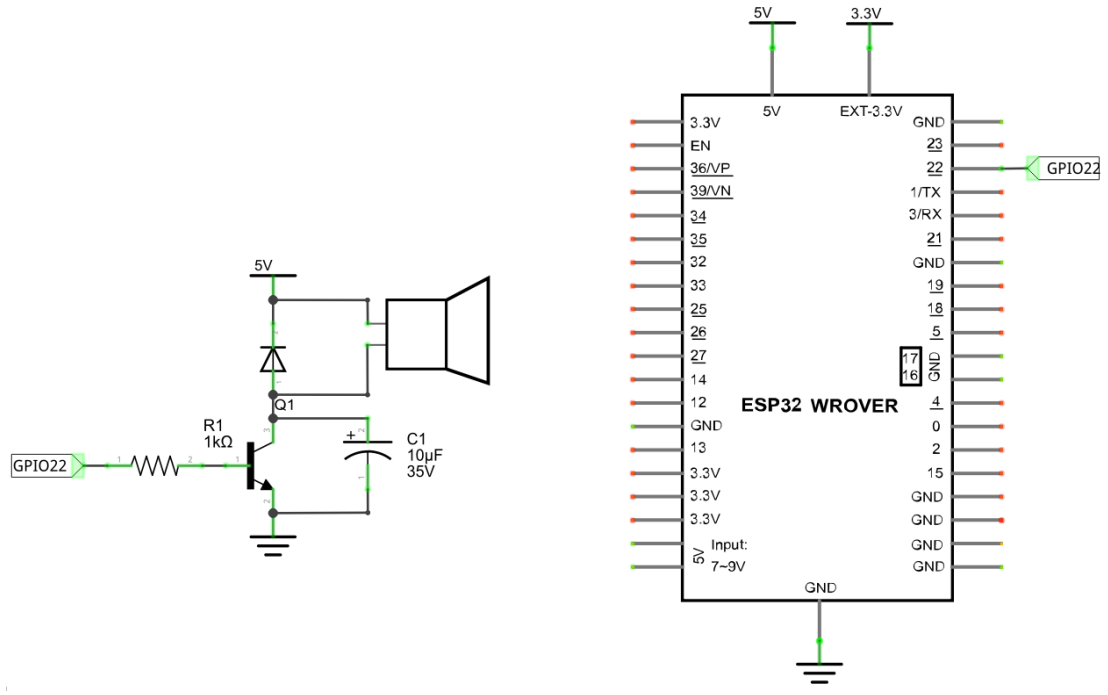
In this project, we will read an mp3 file from an SD card, decode it through ESP32, and use a speaker to play it.

Component List

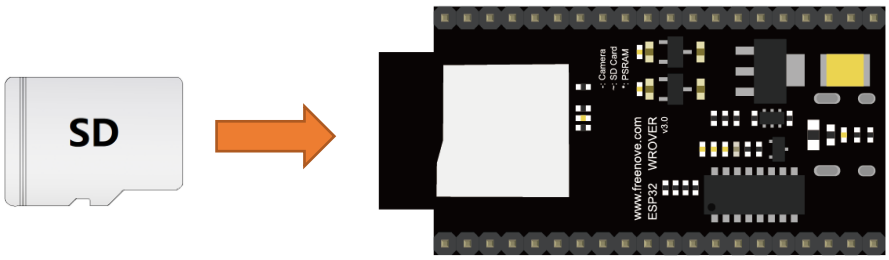
ESP32 WROVER x1 	USB cable x1 	SDcard x1
Micro USB Wire x1 	NPN transistorx1 (S8050)	Speaker 
Diode x1	Resistor 1kΩ x1	Capacitor 10uF x1
Jumper F/M x4 Jumper F/F x2	Card reader x1 (random color) (Not a USB flash drive.)	

Circuit

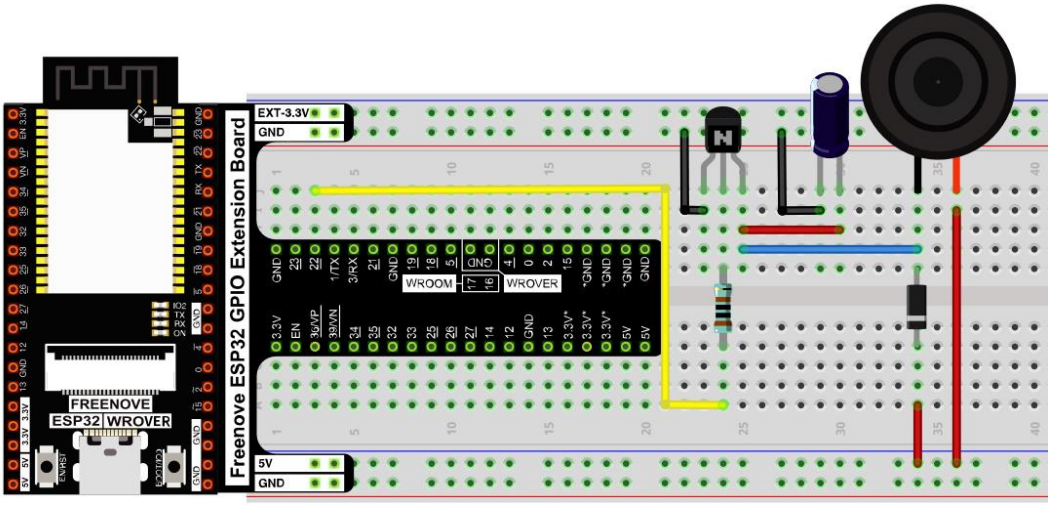
Schematic diagram



Please note that before connecting the USB cable, please put the music into the SD card and insert the SD card into the card slot on the back of the ESP32.



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

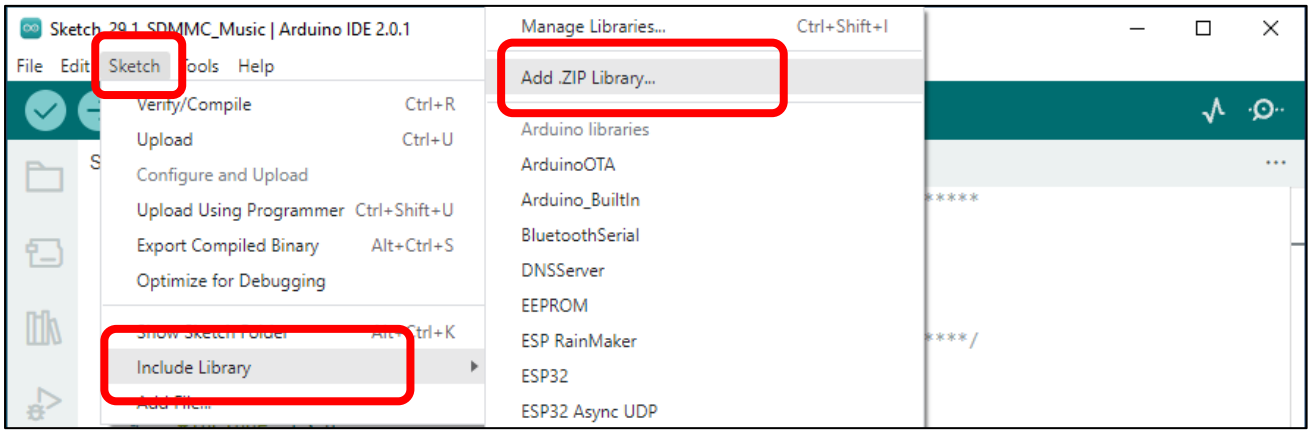


Sketch

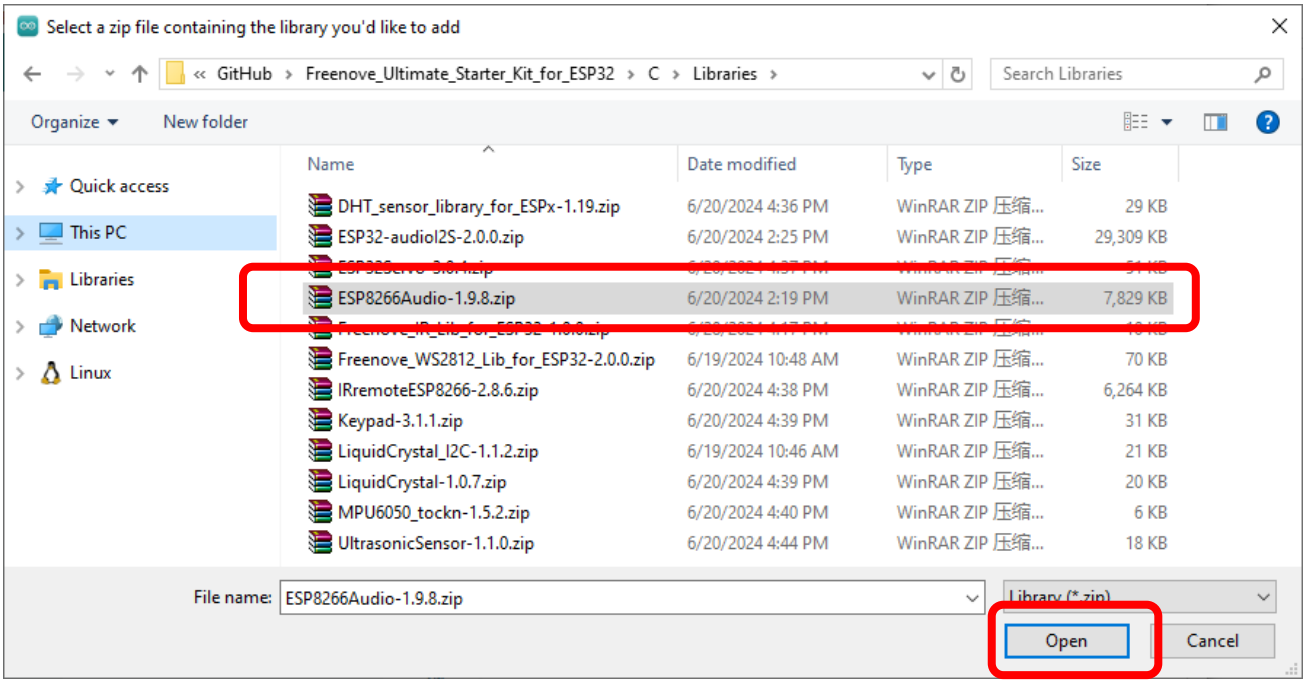
How to install the library

In this project, we will use the ESP8266Audio.zip library to decode the audio files in the SD card, and then output the audio signal through GPIO. If you have not installed this library, please follow the steps below to install it.

Open arduino->Sketch->Include library-> Add .ZIP Library.



In the new pop-up window, select "**Freenove_Ultimate_Starter_Kit_for_ESP32\C\Libraries\ESP8266Audio.zip**". Then click "Open".

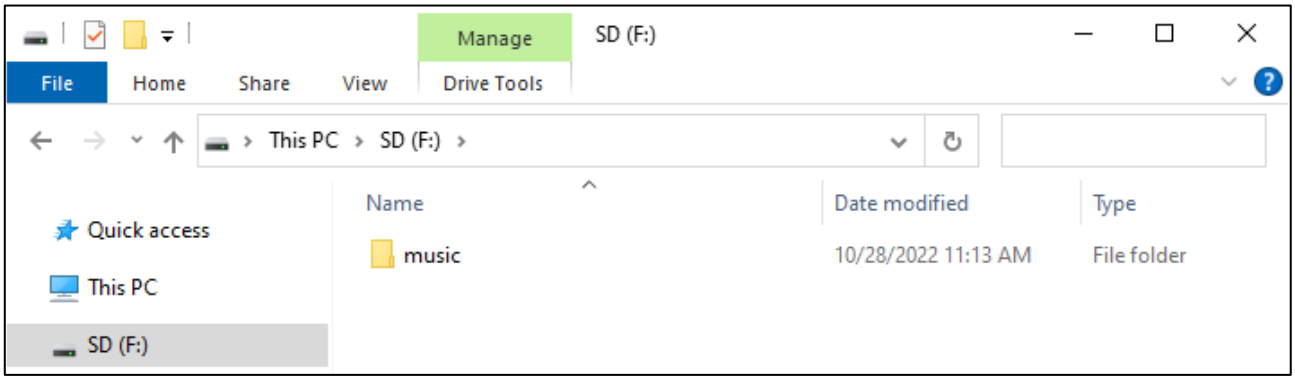


Sketch_31.1_PlayMP3FromSD

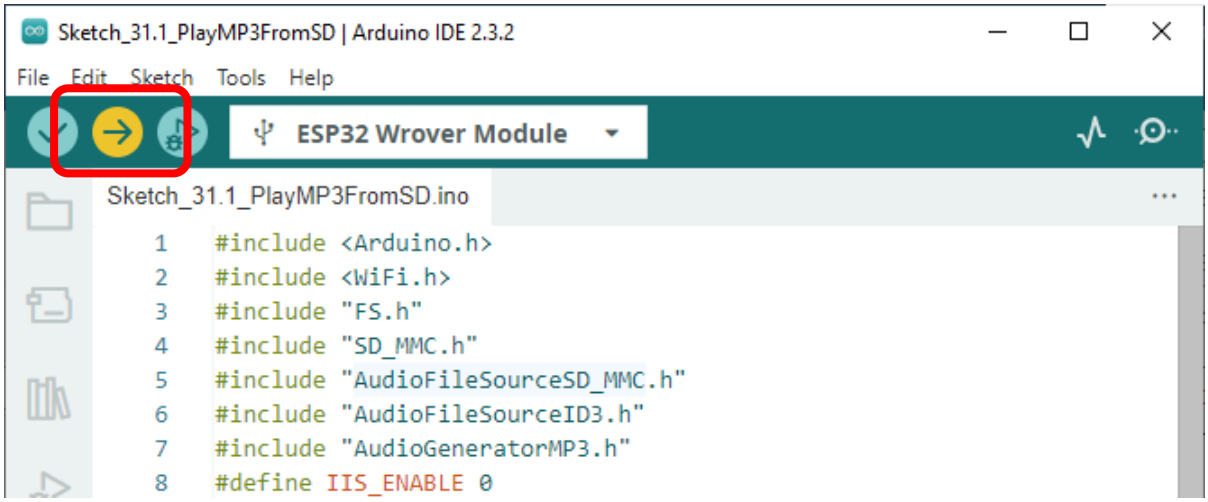
We placed a folder called "music" in:

Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_31.1_PlayMP3FromSD

User needs to copy this folder to SD card.



Click upload.



Compile and upload the code to the ESP32 WROVER and open the serial monitor. ESP32 takes a few seconds to initialize the program. When you see the message below, it means that ESP32 has started parsing the mp3 in sd and started playing music through Pin.

The following is the program code:

```
#include <Arduino.h>
#include <WiFi.h>
#include "FS.h"
#include "SD_MMC.h"
```

```
#include "AudioFileSourceSD_MMC.h"
#include "AudioFileSourceID3.h"
#include "AudioGeneratorMP3.h"
#define IIS_ENABLE 0

#ifdef IIS_ENABLE
#include "AudioOutputI2S.h"
#else
#include "AudioOutputI2SNoDAC.h"
#endif

#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.
#define I2S_BCLK   26
#define I2S_DOUT   22
#define I2S_LRC    25

AudioGeneratorMP3 *mp3;
AudioFileSourceID3 *id3;
#ifdef IIS_ENABLE
AudioOutputI2S *out;
#else
AudioOutputI2SNoDAC *out;
#endif
AudioFileSourceSD_MMC *file = NULL;

// Called when a metadata event occurs (i.e. an ID3 tag, an ICY block, etc.
void MDCallback(void *cbData, const char *type, bool isUnicode, const char *string)
{
  (void)cbData;
  Serial.printf("ID3 callback for: %s = '", type);

  if (isUnicode) {
    string += 2;
  }

  while (*string) {
    char a = *(string++);
    if (isUnicode) {
      string++;
    }
    Serial.printf("%c", a);
  }
```

```
  Serial.printf("'\n");
  Serial.flush();
}

void setup()
{
  WiFi.mode(WIFI_OFF);
  Serial.begin(115200);
  delay(1000);
  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;
  }
  Serial.printf("Sample MP3 playback begins...\n");

  audioLogger = &Serial;
  file = new AudioFileSourceSD_MMC("/music/01.mp3");
  id3 = new AudioFileSourceID3(file);
  id3->RegisterMetadataCB(MDCallback, (void*)"ID3TAG");
  #ifdef IIS_ENABLE
  out = new AudioOutputI2S();
  #else
  out = new AudioOutputI2SNoDAC();
  #endif

  out->SetPinout(I2S_BCLK,I2S_LRC,I2S_DOUT);//Set the audio output pin
  out->SetGain(3.5);//Setting the Volume
  mp3 = new AudioGeneratorMP3();
  mp3->begin(id3, out);
}

void loop()
{
  if (mp3->isRunning()) {
    if (!mp3->loop()) mp3->stop();
  } else {
    Serial.printf("MP3 done\n");
    delay(1000);
  }
}
```

Add music decoding header files and SD card drive files.

If you want to use the circuit in 31.2, you just need to modify **#define IIS_ENABLE 1.**

```
#include <Arduino.h>
#include <WiFi.h>
#include "FS.h"
#include "SD_MMC.h"
#include "AudioFileSourceSD_MMC.h"
#include "AudioFileSourceID3.h"
#include "AudioGeneratorMP3.h"
#define IIS_ENABLE 0              //Direct output audio

#ifdef IIS_ENABLE
#include "AudioOutputI2S.h"       //Output audio using IIS
#else
#include "AudioOutputI2SNoDAC.h"  //Direct output audio
#endif
```

Define the drive pins for SD card. Note that the SD card driver pins cannot be modified.

```
#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.
#define I2S_BCLK   26
#define I2S_DOUT   22
#define I2S_LRC    25
```

Apply for audio decoding class object.

```
AudioGeneratorMP3 *mp3;
AudioFileSourceID3 *id3;
#ifdef IIS_ENABLE
AudioOutputI2S *out;
#else
AudioOutputI2SNoDAC *out;
#endif
AudioFileSourceSD_MMC *file = NULL;
```

Set the audio file source and associate it with the decoder. Initialize the audio output pin and set the volume to 2.

```
  audioLogger = &Serial;
  file = new AudioFileSourceSD_MMC("/music/01.mp3");
  id3 = new AudioFileSourceID3(file);
  id3->RegisterMetadataCB(MDCallback, (void*)"ID3TAG");
  #ifdef IIS_ENABLE
  out = new AudioOutputI2S();
  #else
  out = new AudioOutputI2SNoDAC();
  #endif

```

```
out->SetPinout(12,13,14);//Set the audio output pin, Only 14 were used
out->SetGain(2);//Setting the Volume(0-3.9)
mp3 = new AudioGeneratorMP3();
mp3->begin(id3, out);
```

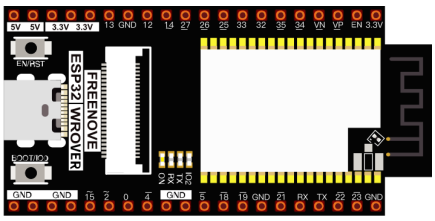
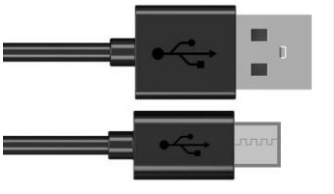
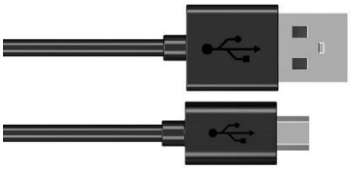
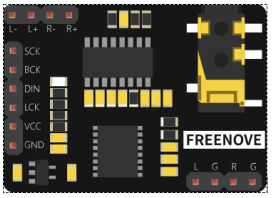
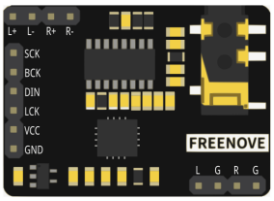
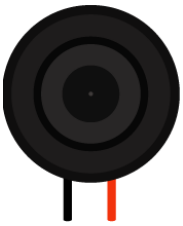
Determine whether the mp3 player is finished. If it is playing, continue playing. If it is finished, print a message.

```
if (mp3->isRunning()) {
  if (!mp3->loop()) mp3->stop();
} else {
  Serial.printf("MP3 done\n");
  delay(1000);
}
```

Project 31.2 SDMMC Music

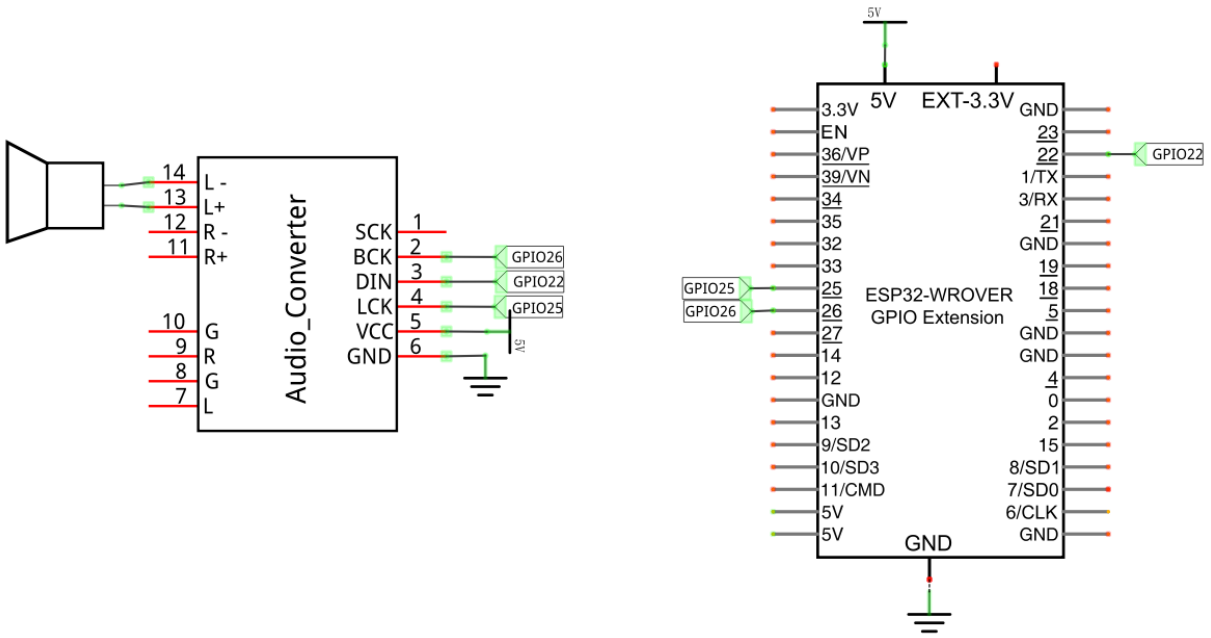
In this project, we will read mp3 files from SD card, decode them through ESP32, and use Audio Converter & Amplifier module to transcode into stereo output.

Component List

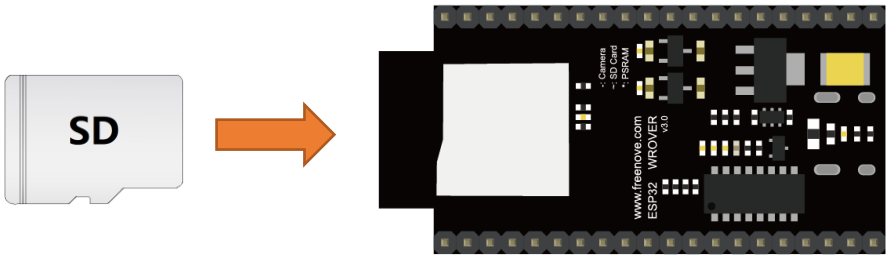
ESP32 WROVER x1 	USB cable x1 	SDcard x1
Micro USB Wire x1 	Audio Converter & Amplifier  or 	
Speaker 		
Jumper F/M x4 Jumper F/F x2	Card reader x1 (random color)	

Circuit

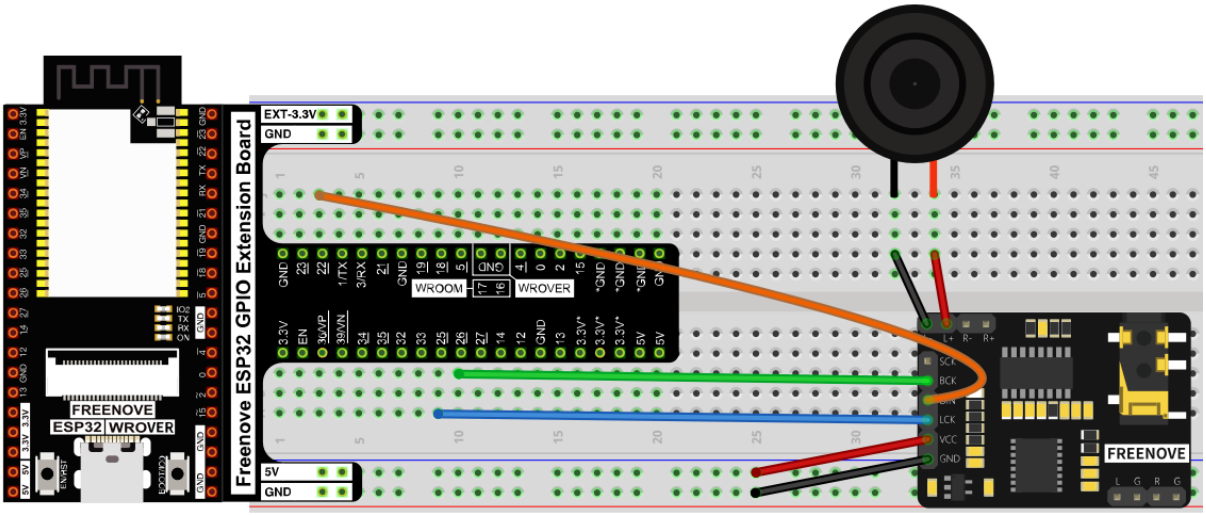
Schematic diagram



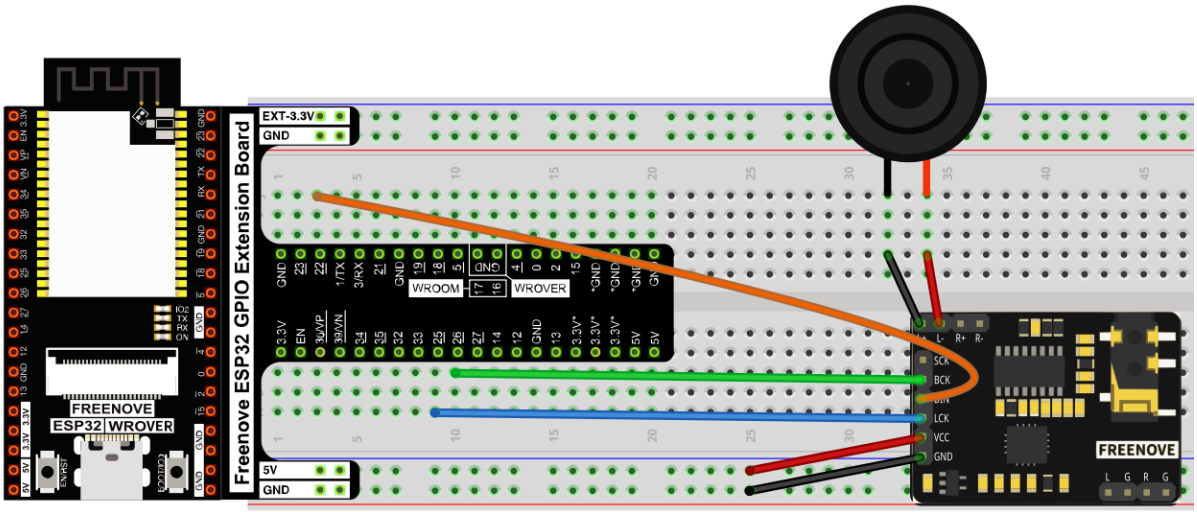
Please note that before connecting the USB cable, please put the music into the SD card and insert the SD card into the card slot on the back of the ESP32.



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



or

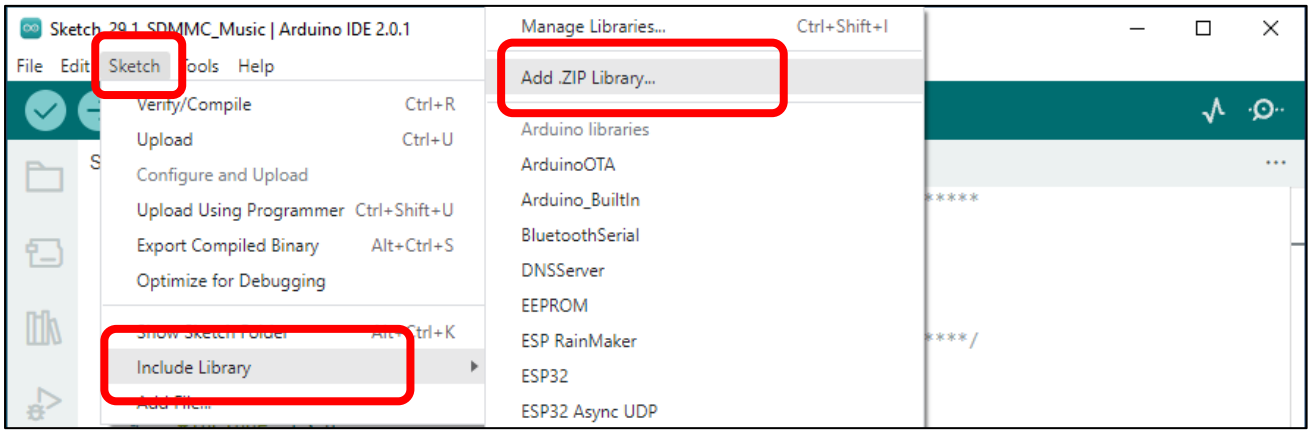


Sketch

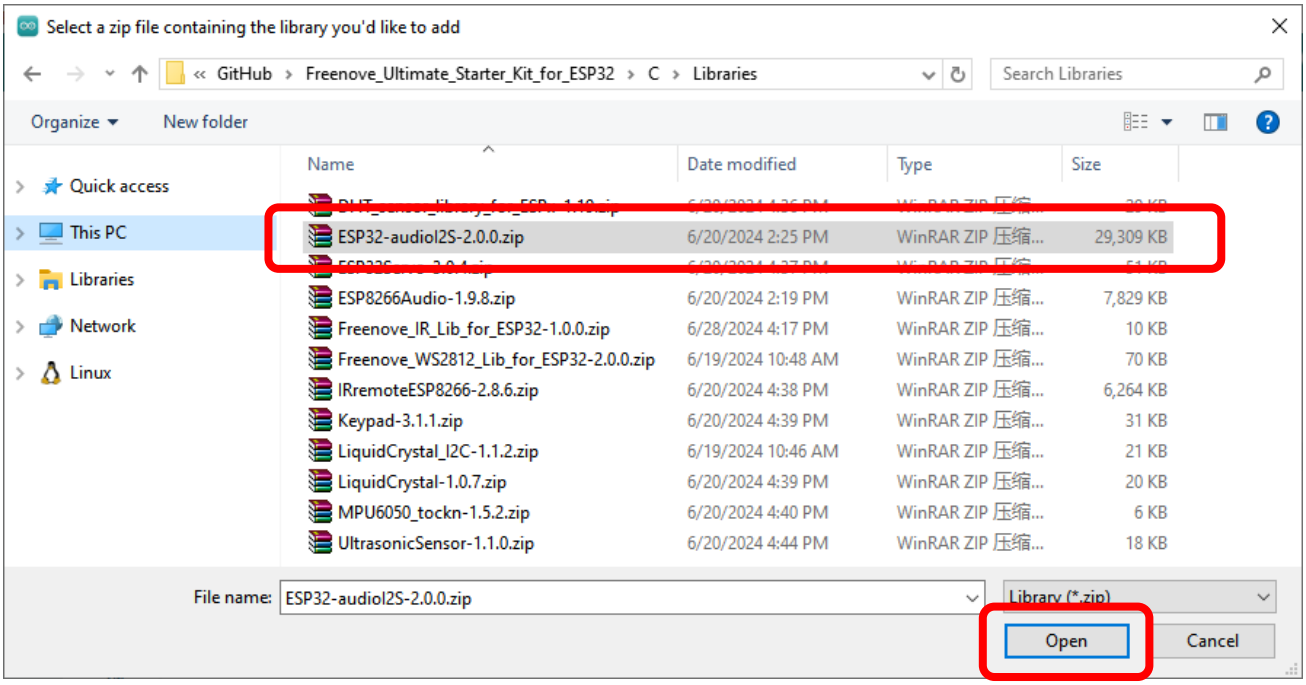
How to install the library

In this project, we will use the ESP32-audioI2S.zip library to decode the audio files in the SD card, and then output the audio signal through IIS. If you have not installed this library, please follow the steps below to install it.

Open arduino->Sketch->Include library-> Add .ZIP Library.



In the new pop-up window, select "**Freenove_Ultimate_Starter_Kit_for_ESP32\C\Libraries\ESP32-audioI2S.zip**". Then click "Open".

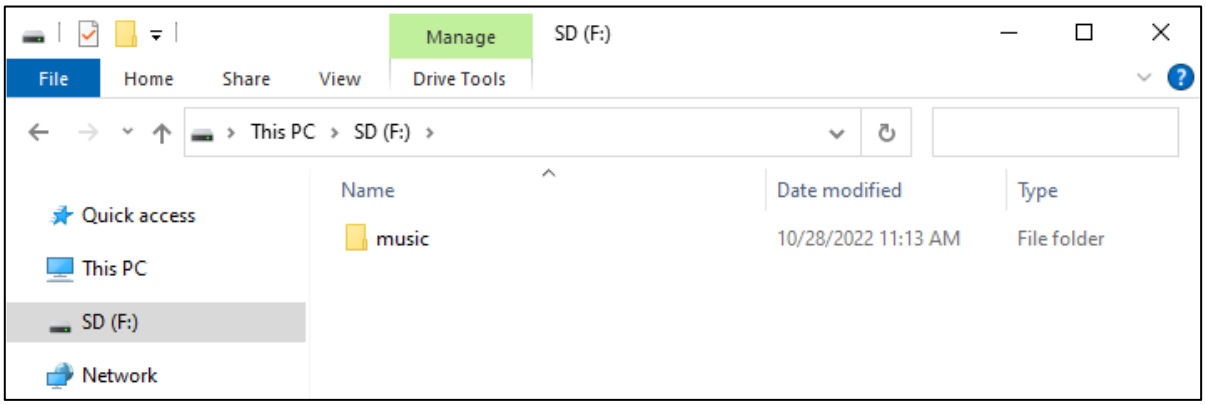


Sketch_31.2_SDMMC_Music

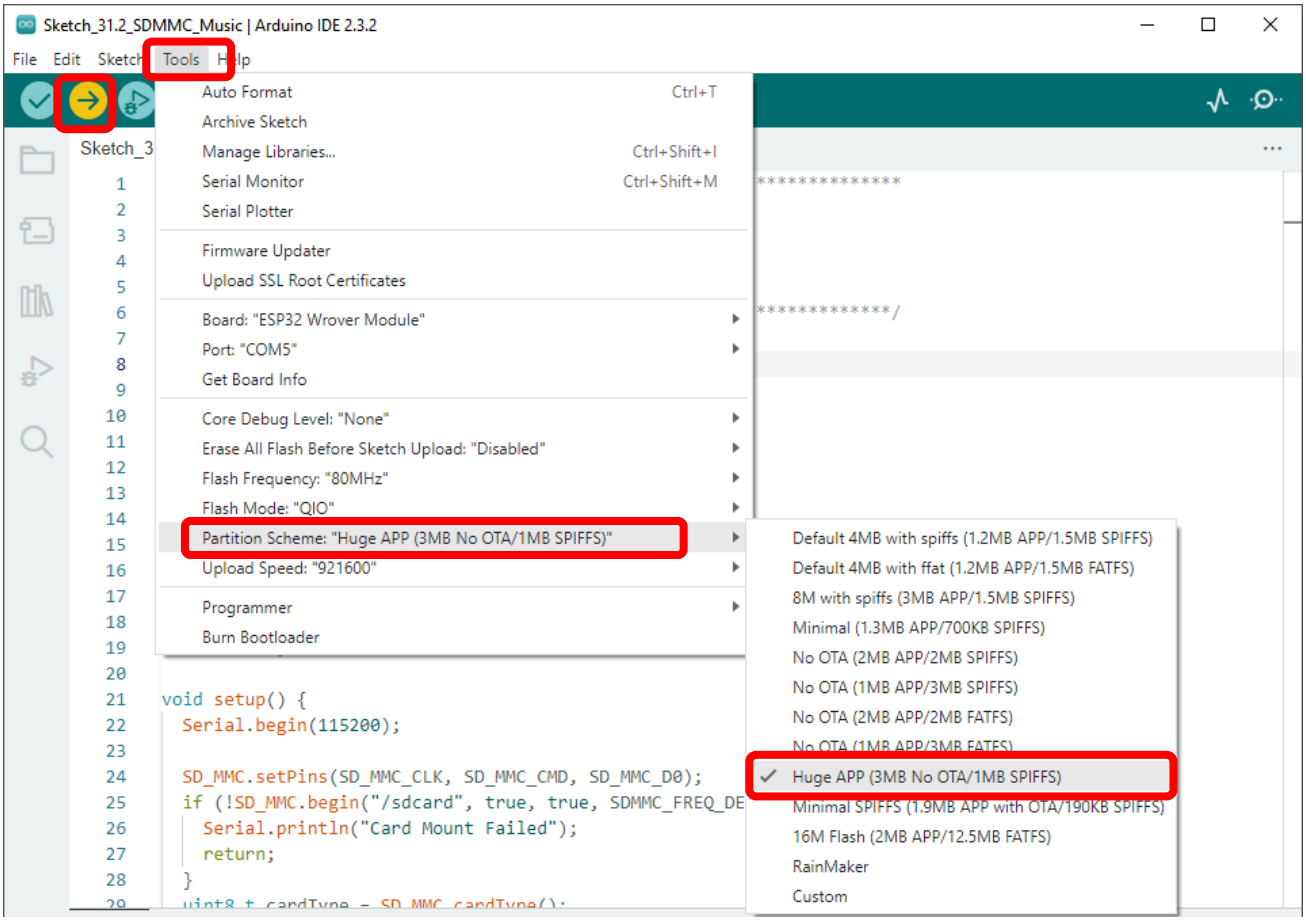
We placed a folder called "music" in:

Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketch_31.2_SDMMC_Music.

User needs to copy this folder to SD card.



Click Tools->Partition Scheme, Select Huge APP……, Then click upload.



Compile and upload the code to the ESP32 WROVER and open the serial monitor. ESP32 takes a few seconds to initialize the program.

When you see the message below, it means that ESP32 has started parsing the mp3 in sd and started playing music through iis.

```
SDHC
SD_MMC Card Size: 15193MB
info        PSRAM not found, inputBufferSize: 6399 bytes
info        buffers freed, free Heap: 258860 bytes
info        Reading file: "/music/Jingle Bells.mp3"
info        MP3Decoder has been initialized, free Heap: 231020 bytes
info        Content-Length: 8004409
info        ID3 framesSize: 1409
info        ID3 version: 2.3
info        ID3 normal frames
id3data     UserDefinedText: EXTENDXTEND
id3data     UserDefinedText: KGHASHGHASH
id3data     Year:
info        Audio-Length: 8003000
info        stream ready
info        syncword found at pos 0
info        Channels: 2
info        SampleRate: 44100
info        BitsPerSample: 16
info        BitRate: 320000
```

The following is the program code:

```
#include "Arduino.h"
#include "Audio.h"
#include "FS.h"
#include "SD_MMC.h"

#define SD_MMC_CMD 15
#define SD_MMC_CLK 14
#define SD_MMC_D0  2
#define I2S_BCLK   26
#define I2S_DOUT   22
#define I2S_LRC    25

Audio audio;

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;
  }
  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);

  audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
  audio.setVolume(12);  // 0...21
  audio.connecttoFS(SD_MMC, "/music/Jingle Bells.mp3");
}

void loop() {
  audio.loop();
  if (Serial.available()) {  // put streamURL in serial monitor
    audio.stopSong();
    String r = Serial.readString();
    r.trim();
    if (r.length() > 5) audio.connecttoFS(SD_MMC, r.c_str());
    log_i("free heap=%i", ESP.getFreeHeap());
  }
}

// optional
void audio_info(const char *info) {
  Serial.print("info        ");
  Serial.println(info);
}
void audio_id3data(const char *info) {  //id3 metadata
  Serial.print("id3data     ");
  Serial.println(info);
}
void audio_eof_mp3(const char *info) {  //end of file
  Serial.print("eof_mp3     ");
  Serial.println(info);
}
void audio_showstation(const char *info) {
  Serial.print("station     ");
  Serial.println(info);
}
```

```
void audio_showstreamtitle(const char *info) {
  Serial.print("streamtitle ");
  Serial.println(info);
}
void audio_bitrate(const char *info) {
  Serial.print("bitrate     ");
  Serial.println(info);
}
void audio_commercial(const char *info) {  //duration in sec
  Serial.print("commercial  ");
  Serial.println(info);
}
void audio_icyurl(const char *info) {  //homepage
  Serial.print("icyurl      ");
  Serial.println(info);
}
void audio_lasthost(const char *info) {  //stream URL played
  Serial.print("lasthost    ");
  Serial.println(info);
}
```

Add music decoding header files and SD card drive files.

```
#include "Arduino.h"
#include "Audio.h"
#include "FS.h"
#include "SD_MMC.h"
```

Define the drive pins for SD card and IIS. Note that the SD card driver pins cannot be modified, but the IIS drive pins can be modified.

```
#define SD_MMC_CMD 15
#define SD_MMC_CLK 14
#define SD_MMC_D0  2
#define I2S_BCLK   26
#define I2S_DOUT   22
#define I2S_LRC    25
```

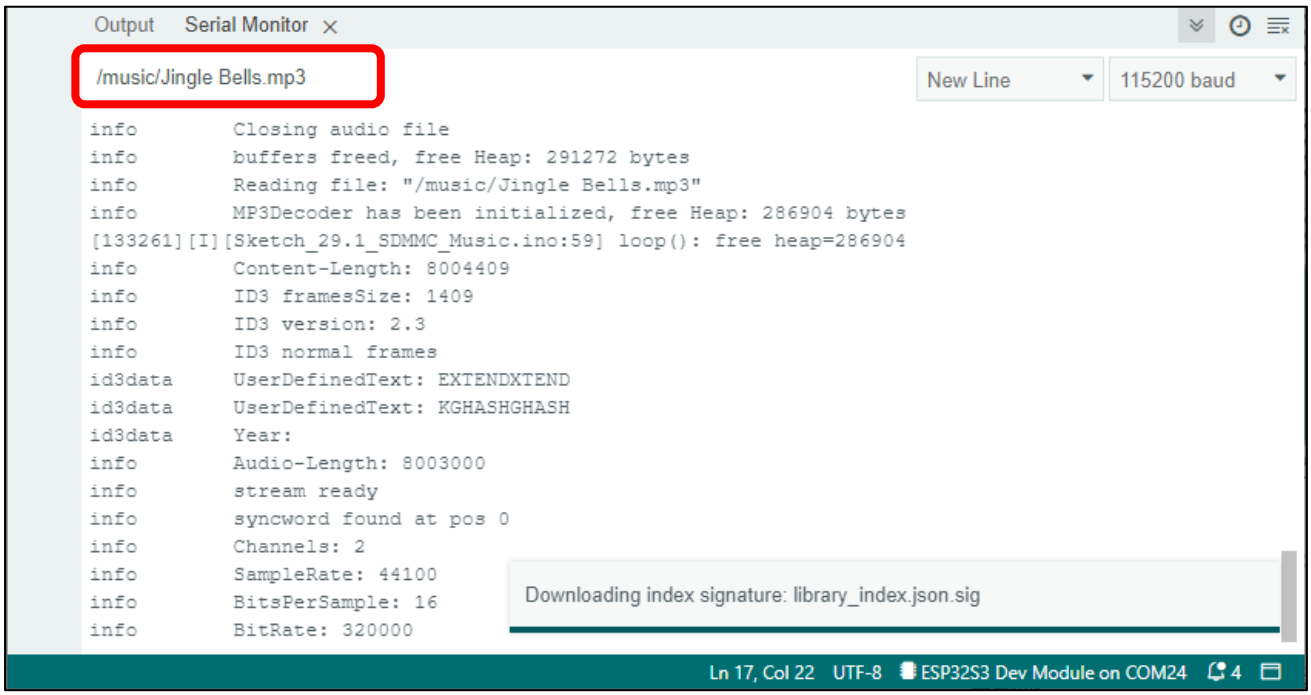
Declare an audio decoding object, associate it with the pin, set the volume, and set the decoding object.

```
  Audio audio;
  ...
  audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
  audio.setVolume(12);  // 0...21
  audio.connecttoFS(SD_MMC, "/music/Jingle Bells.mp3");
```

Play music until one piece of music finishes playing. If the serial port receives data, it will call the audio object to decode it after removing the spaces at the head and tail of the data.

```
  audio.loop();
  if (Serial.available()) {  // put streamURL in serial monitor
    audio.stopSong();
    String r = Serial.readString();
    r.trim();
    if (r.length() > 5) audio.connecttoFS(SD_MMC, r.c_str());
    log_i("free heap=%i", ESP.getFreeHeap());
  }
```

In other words, if you want to switch the music in the SD card, you can directly input the song through the serial port.



The following functions are used to print the audio decoding information. If you do not want to see the decoding information in the serial port, you can directly comment out these functions.

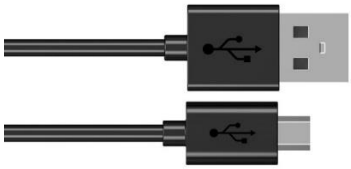
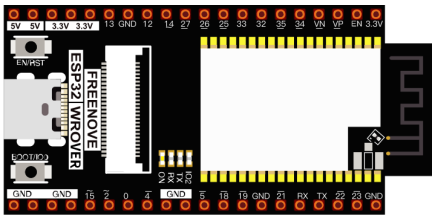
```
void audio_info(const char *info);
void audio_id3data(const char *info);
void audio_eof_mp3(const char *info);
void audio_showstation(const char *info);
void audio_showstreamtitle(const char *info);
void audio_bitrate(const char *info);
void audio_commercial(const char *info);
void audio_icyurl(const char *info);
void audio_lasthost(const char *info);
```

Chapter 32 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 32.1 Station mode

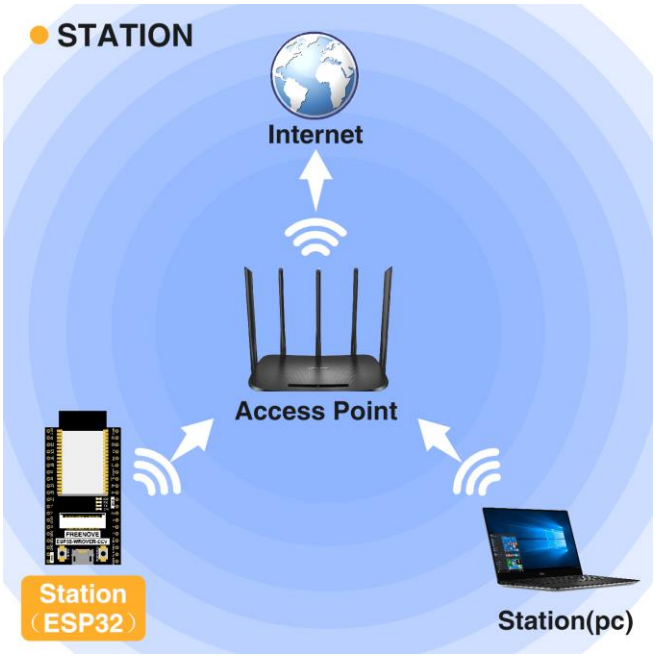
Component List

Micro USB Wire x1	ESP32-WROVER x1
	

Component knowledge

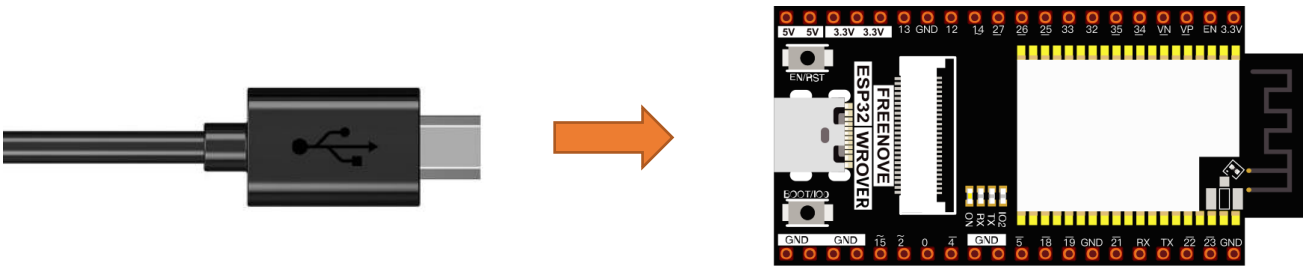
Station mode

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.

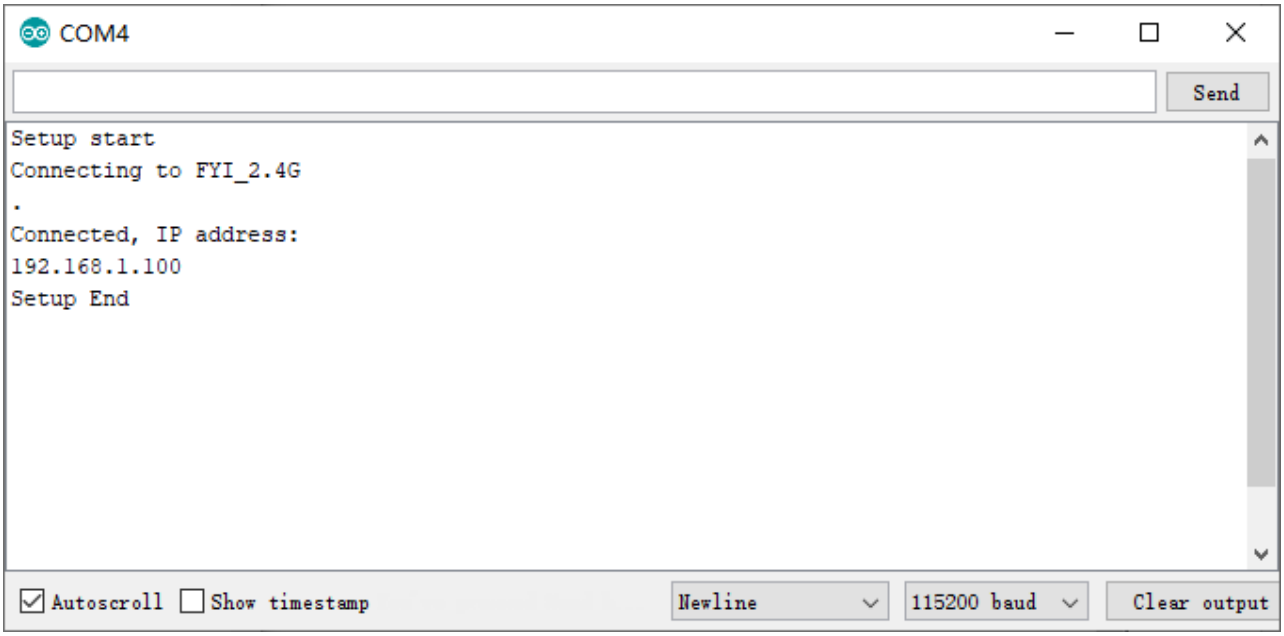


Sketch

Sketch_32.1_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 32.2 AP mode

Component List & Circuit

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

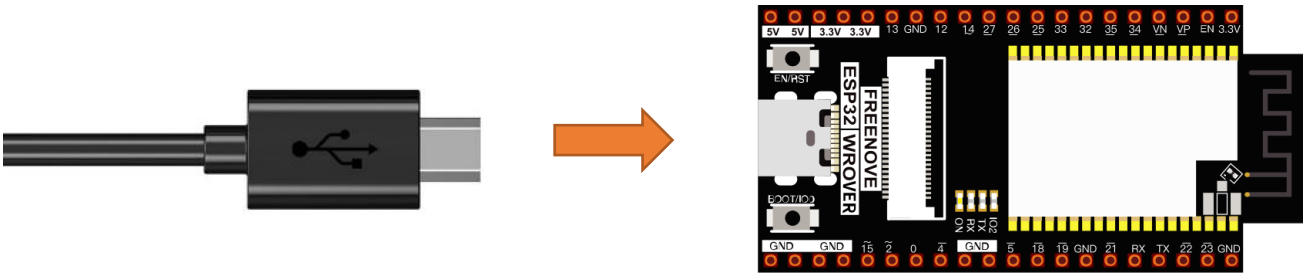
Component knowledge

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

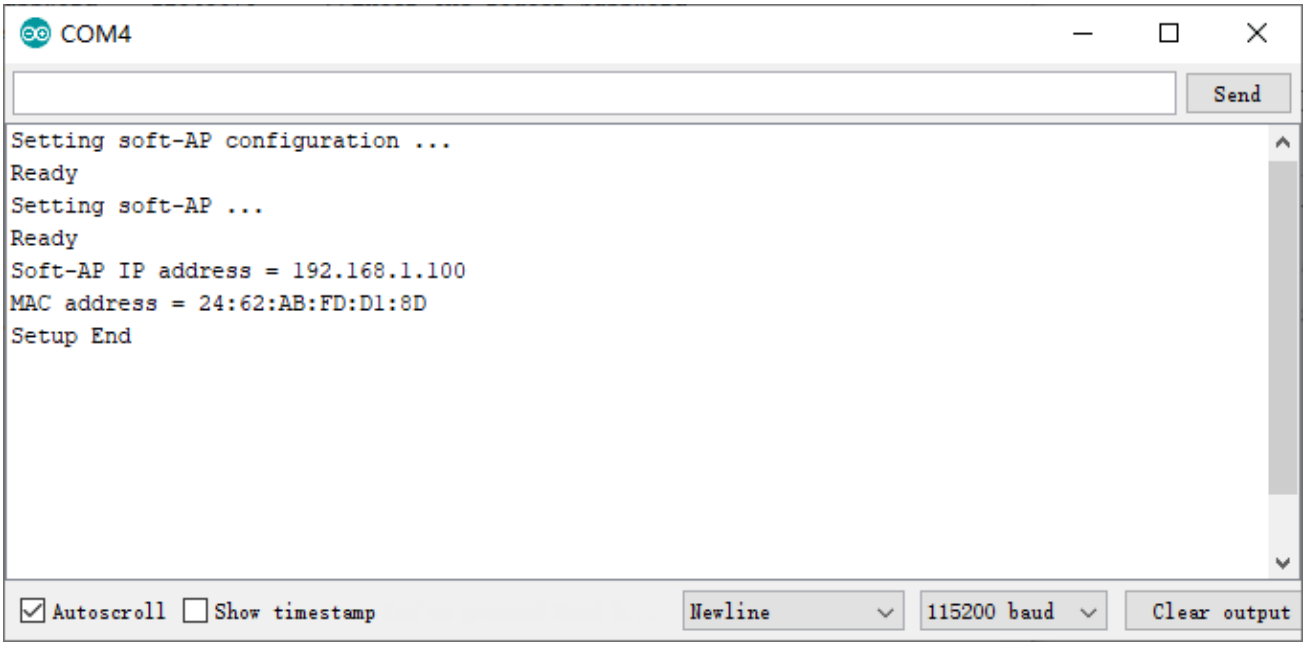
Connect Freenove ESP32 to the computer using the USB cable.



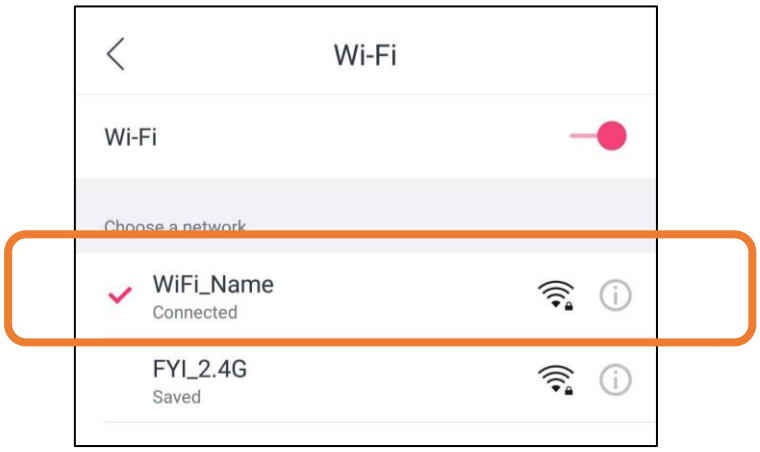
Sketch

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_32.2_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.

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

Enter correct AP name and password.

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

Set ESP32 in AP mode.

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

Configure IP address, gateway and subnet mask for ESP32.

```
16 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.

```
18 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.

```
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");
```

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

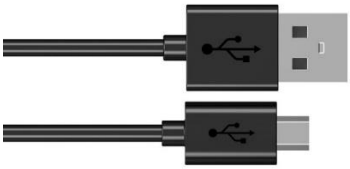
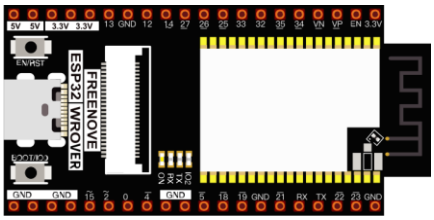
subnet: subnet mask

softAP(): obtian IP address in AP mode

softAPdisconnect (): disconnect AP mode.

Project 30.3 AP+Station mode

Component List

Micro USB Wire x1	ESP32-WROVER x1
	

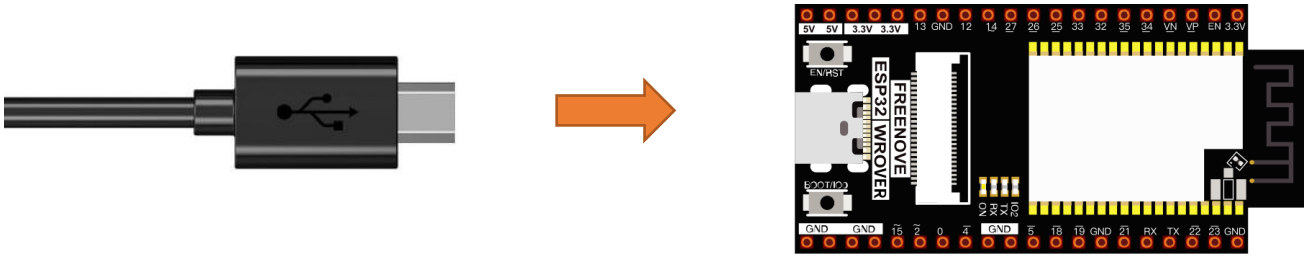
Component knowledge

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.

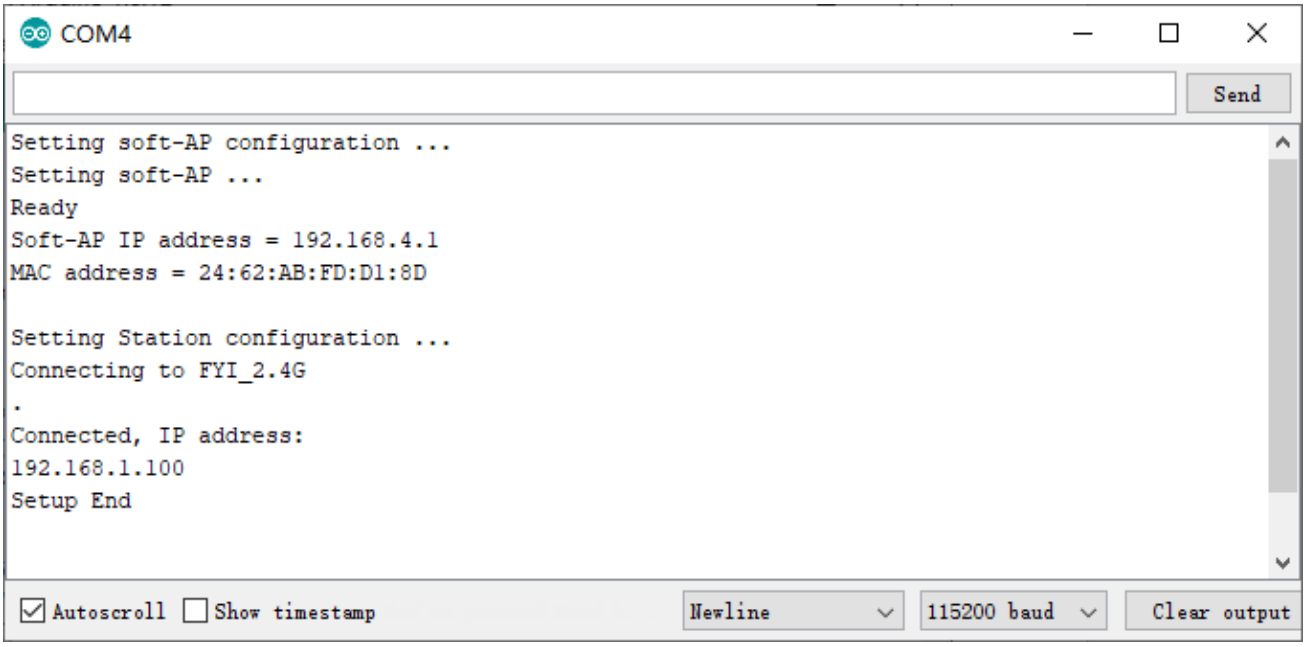


Sketch

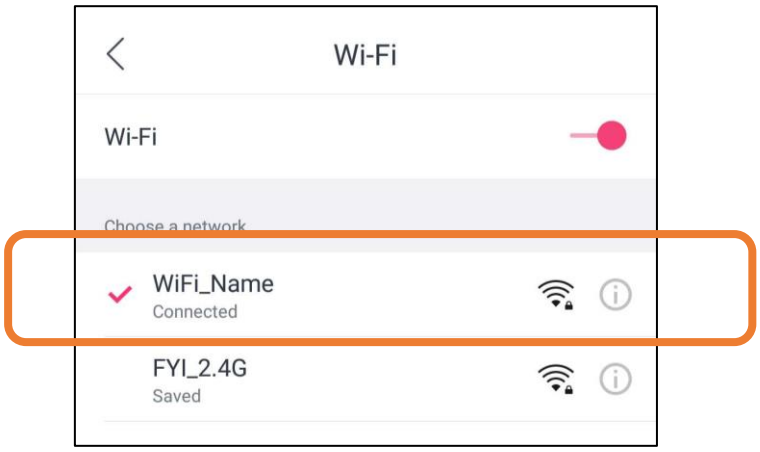
Sketch_32.3_AP_Station_mode

It is analogous to project 32.1 and project 32.2. 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();
```

```
  WiFi.mode(WIFI_AP);
  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("\nSetting Station configuration ... ");
  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() {
}
```

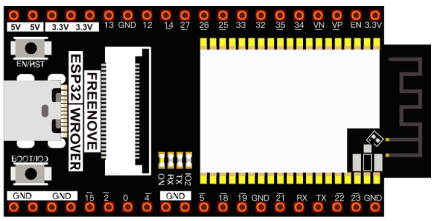
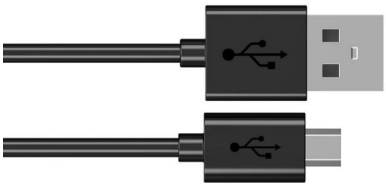
Chapter 33 TCP/IP

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 33.1 As Client

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

Component List

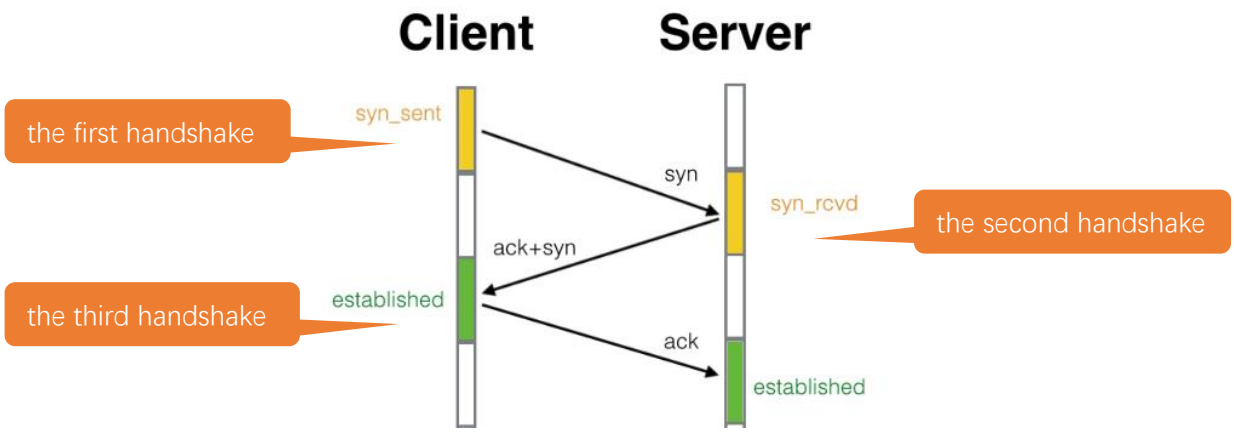
ESP32-WROVER x1 	Micro USB Wire x1 

Component knowledge

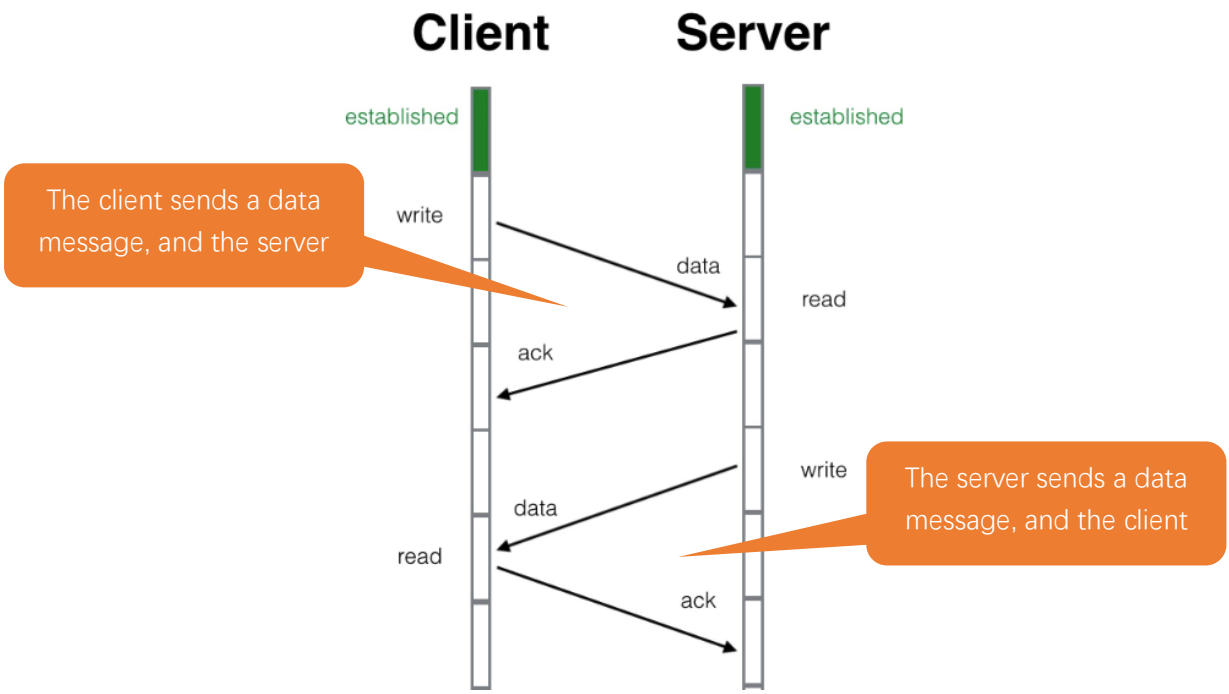
TCP connection

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

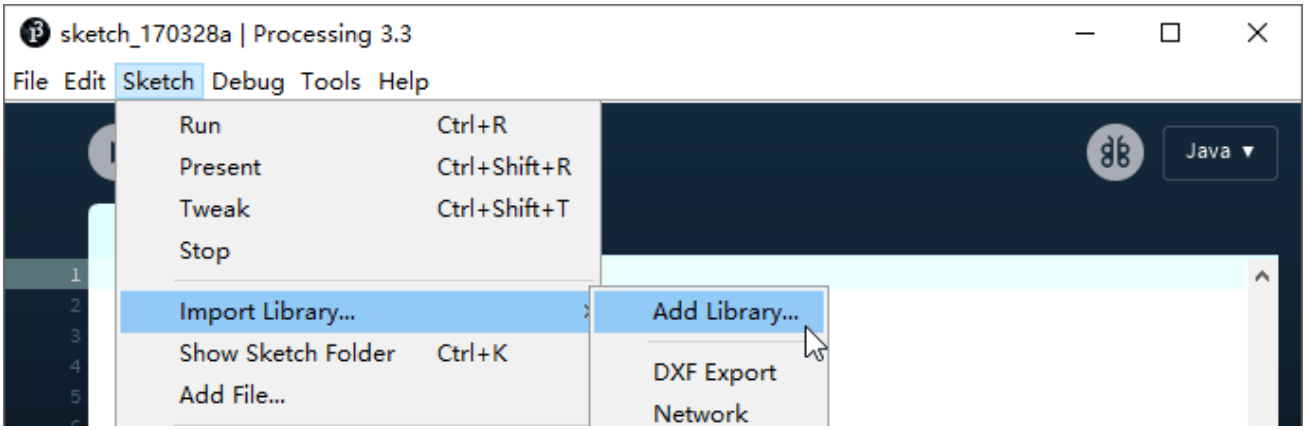
In this tutorial, we use Processing to build a simple TCP/IP communication platform.

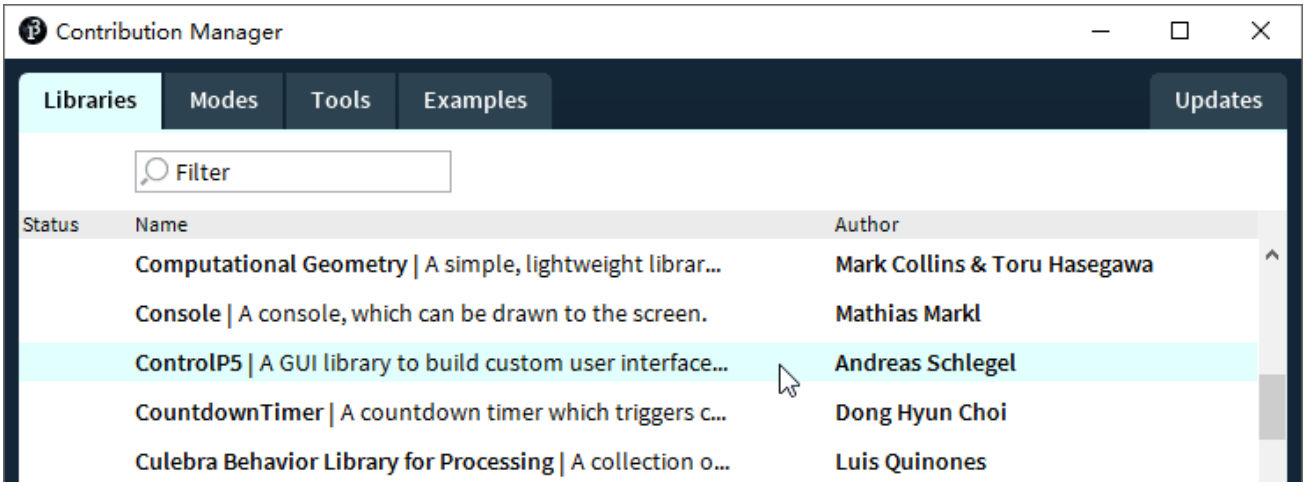
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

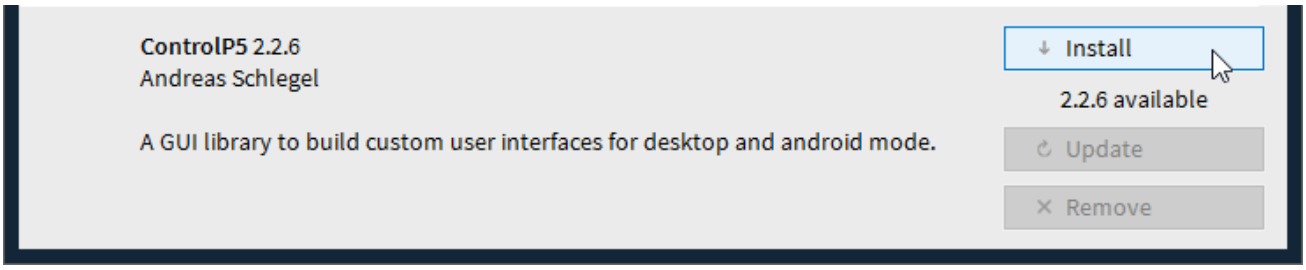
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

Use Server mode for communication

Install ControlP5.





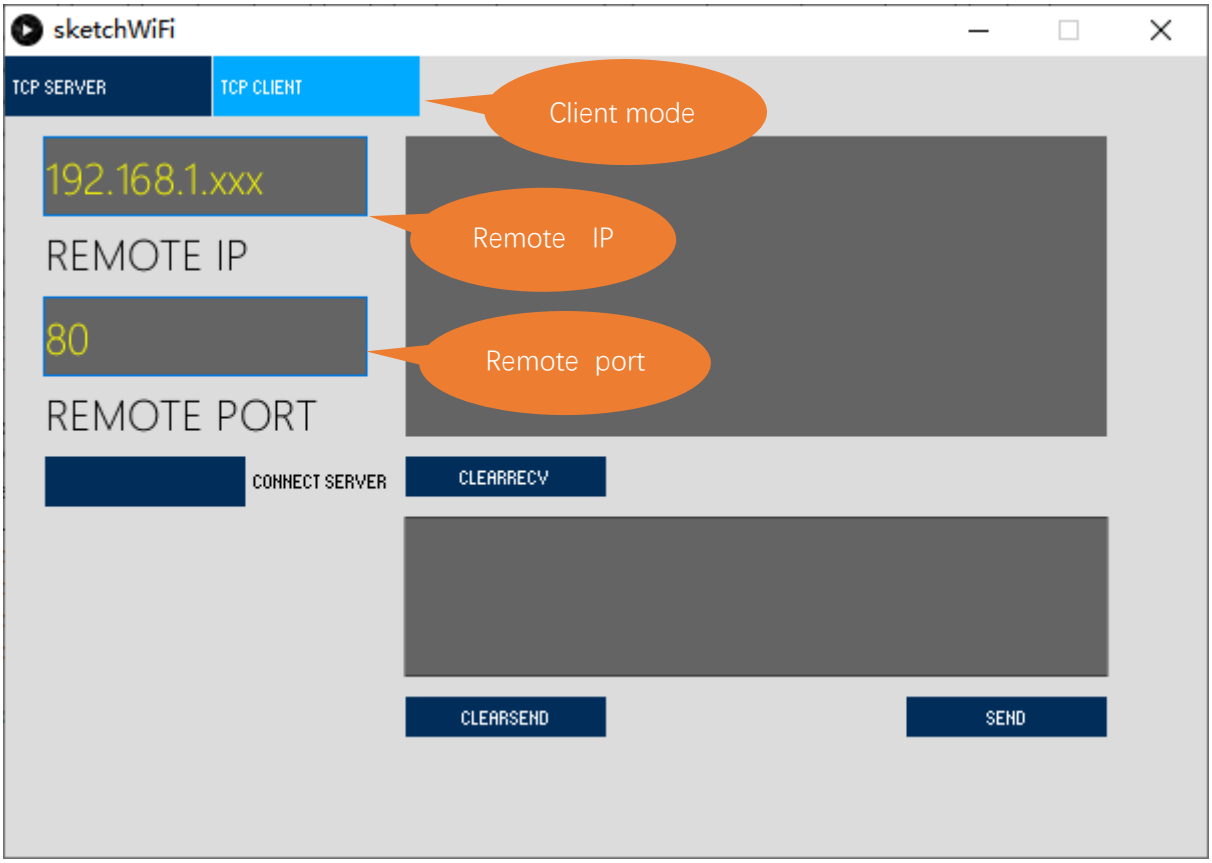


Open the "**Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketches\Sketch_33.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

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.

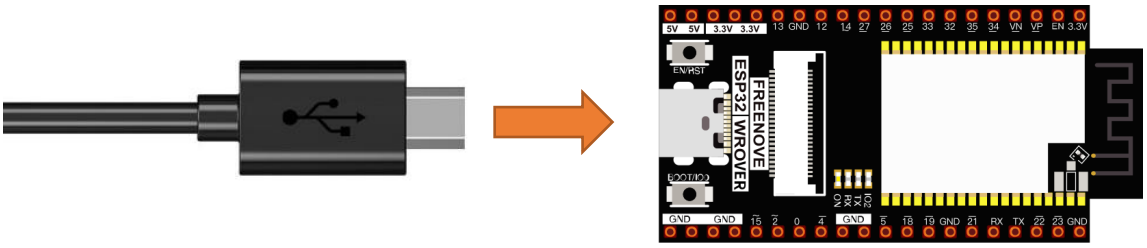
clear receive: clear out the content in the receiving text box

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.

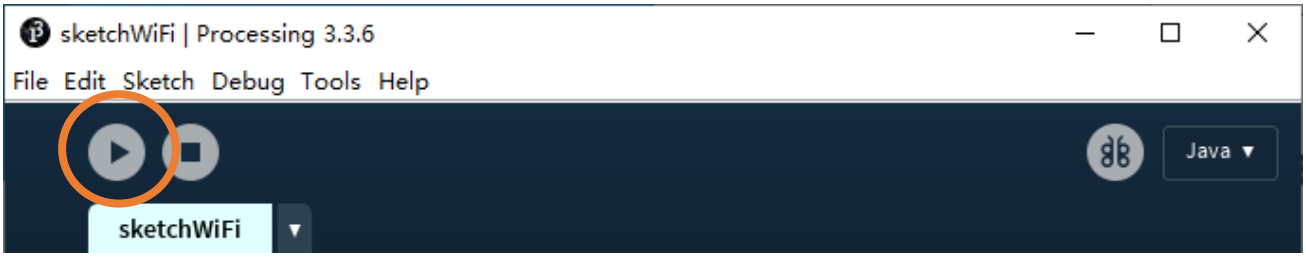
Circuit

Connect Freenove ESP32 to the computer using USB cable.

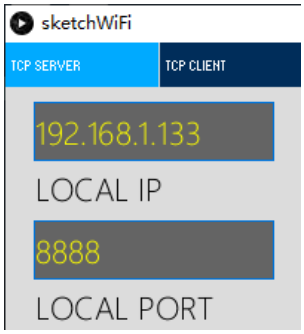


Sketch

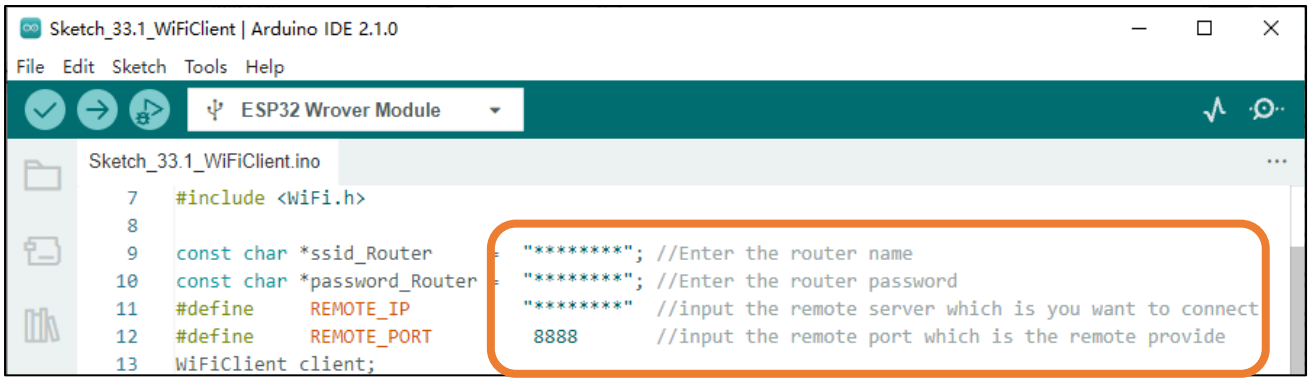
Before running the Sketch, please open “sketchWiFi.pde.” first, and click “Run”.



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

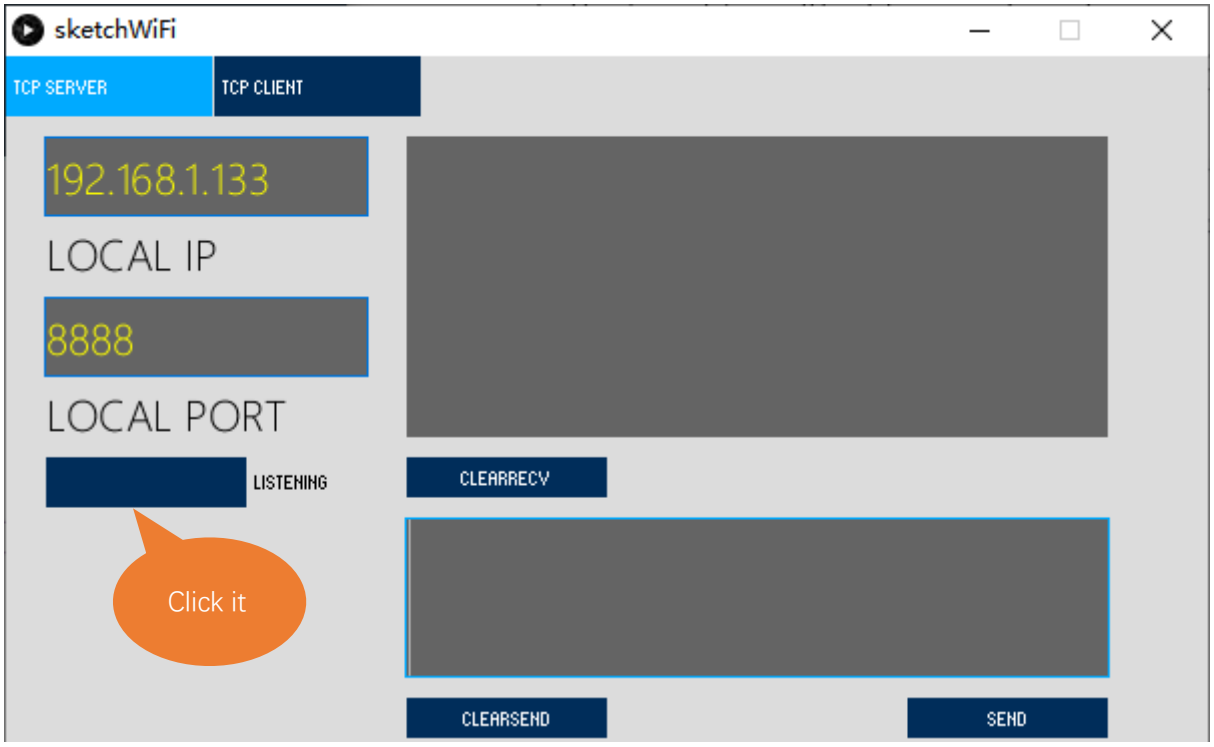


Next, open Sketch_33.1_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.



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.

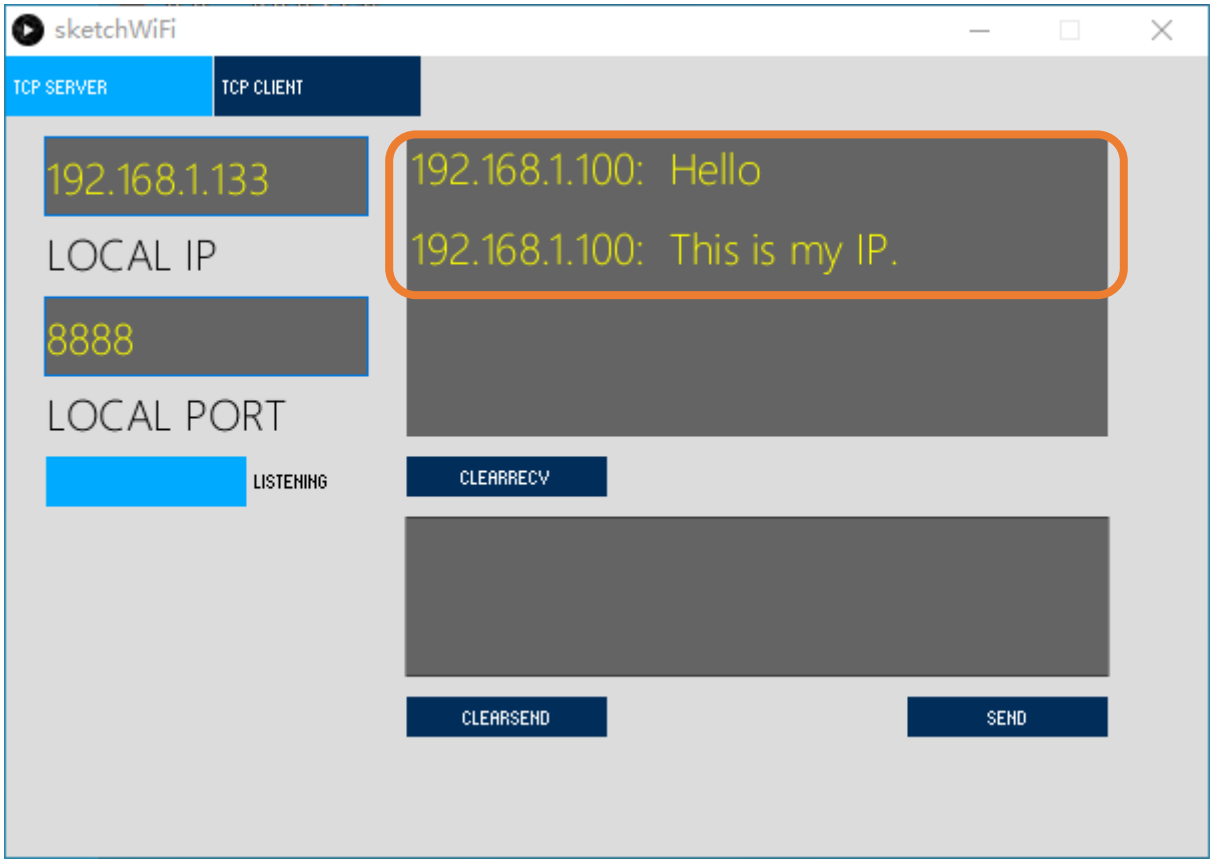
Click LISTENING, turn on TCP SERVER's data listening function and wait for ESP32 to connect.



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.

```
Waiting for WiFi... .
WiFi connected
IP address:
192.168.1.100
Connecting to 192.168.1.133
Connected
```

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



Sketch_33.1_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);
  }
```

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

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

```
  if (client.connected () == false) {
    client.stop();
    WiFi.disconnect();
  }
```

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.

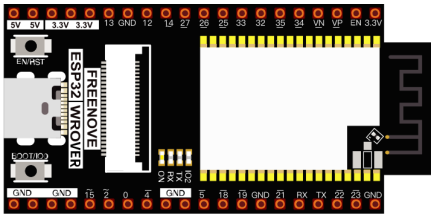
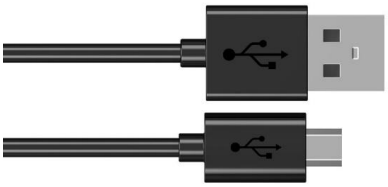
read(): read one byte of data in receive buffer

readString(): read string in receive buffer

Project 33.2 As Server

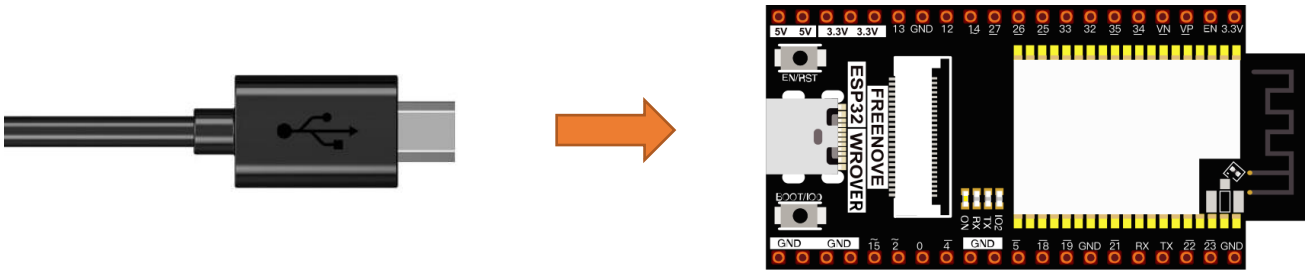
In this section, ESP32 is used as a server to wait for the connection and communication of client on the same LAN.

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

Circuit

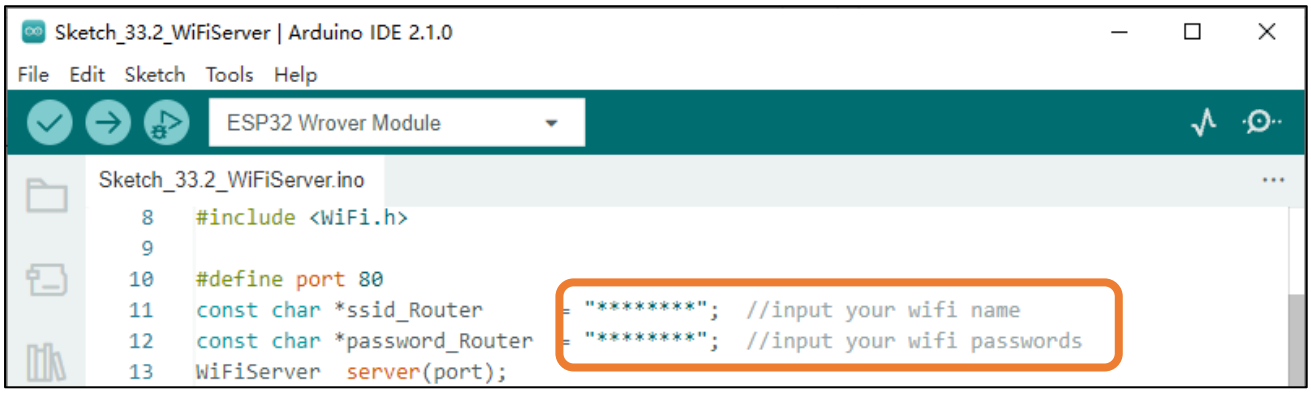
Connect Freenove ESP32 to the computer using a USB cable.



Sketch

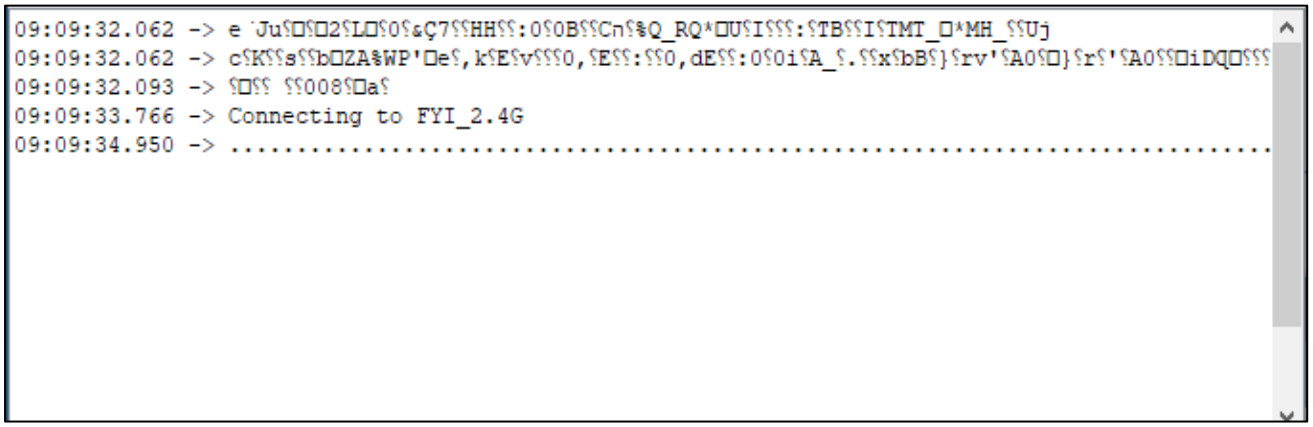
Before running Sketch, please modify the contents of the box below first.

Sketch_33.2_As_Server

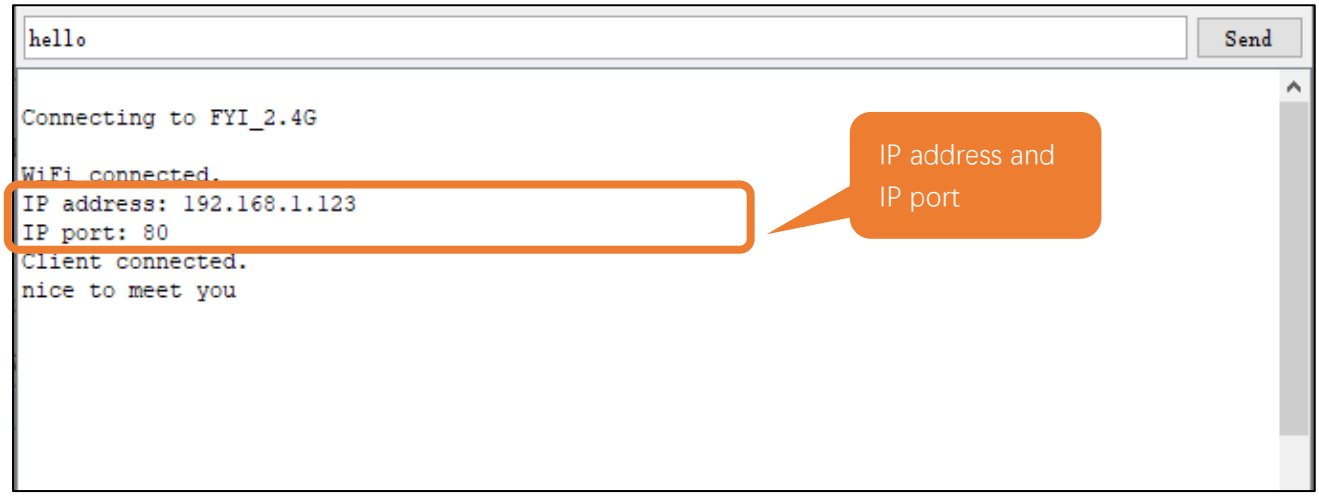


Compile and upload code to ESP32-WROVER board, open the serial monitor and set the baud rate to 115200. Turn on server mode for ESP32, waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other.

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



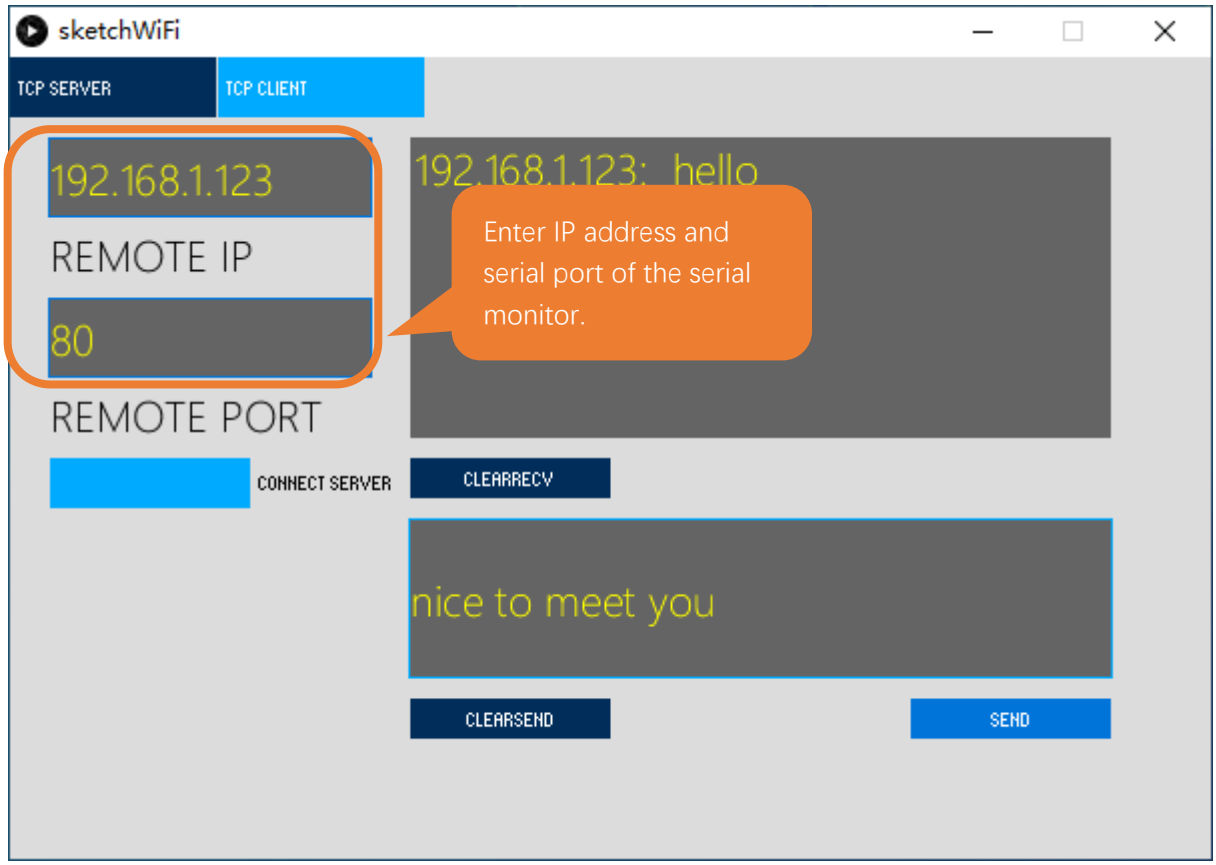
Serial Monitor



Processing:

Open the "**Freenove_Ultimate_Starter_Kit_for_ESP32\Sketches\Sketches\Sketch_33.2_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
      }
```

```
    }
    client.stop();                                  // stop the client connecting.
    Serial.println("Client Disconnected.");
  }
}
```

Apply for method class of WiFiServer.

```
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.

```
    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.");
```

Print out the IP address and port number of ESP32.

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

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

```
    server.begin();                                 //Turn ON ESP32 as Server mode
    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.

```
      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
      }
```

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.

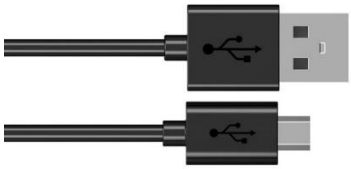
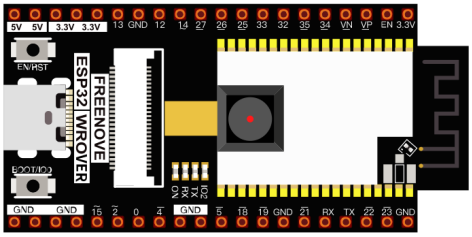
Chapter 34 Camera Web Server

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

Project 34.1 Camera Web Server

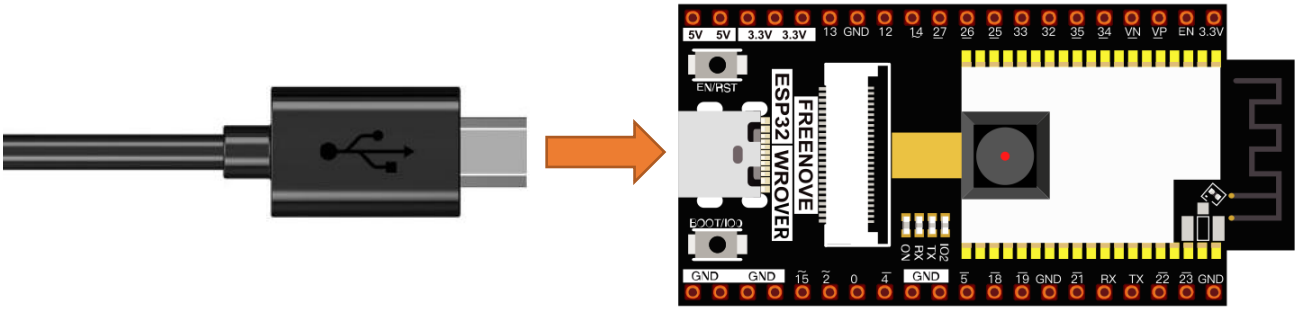
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

Component List

Micro USB Wire x1	ESP32-WROVER x1
	

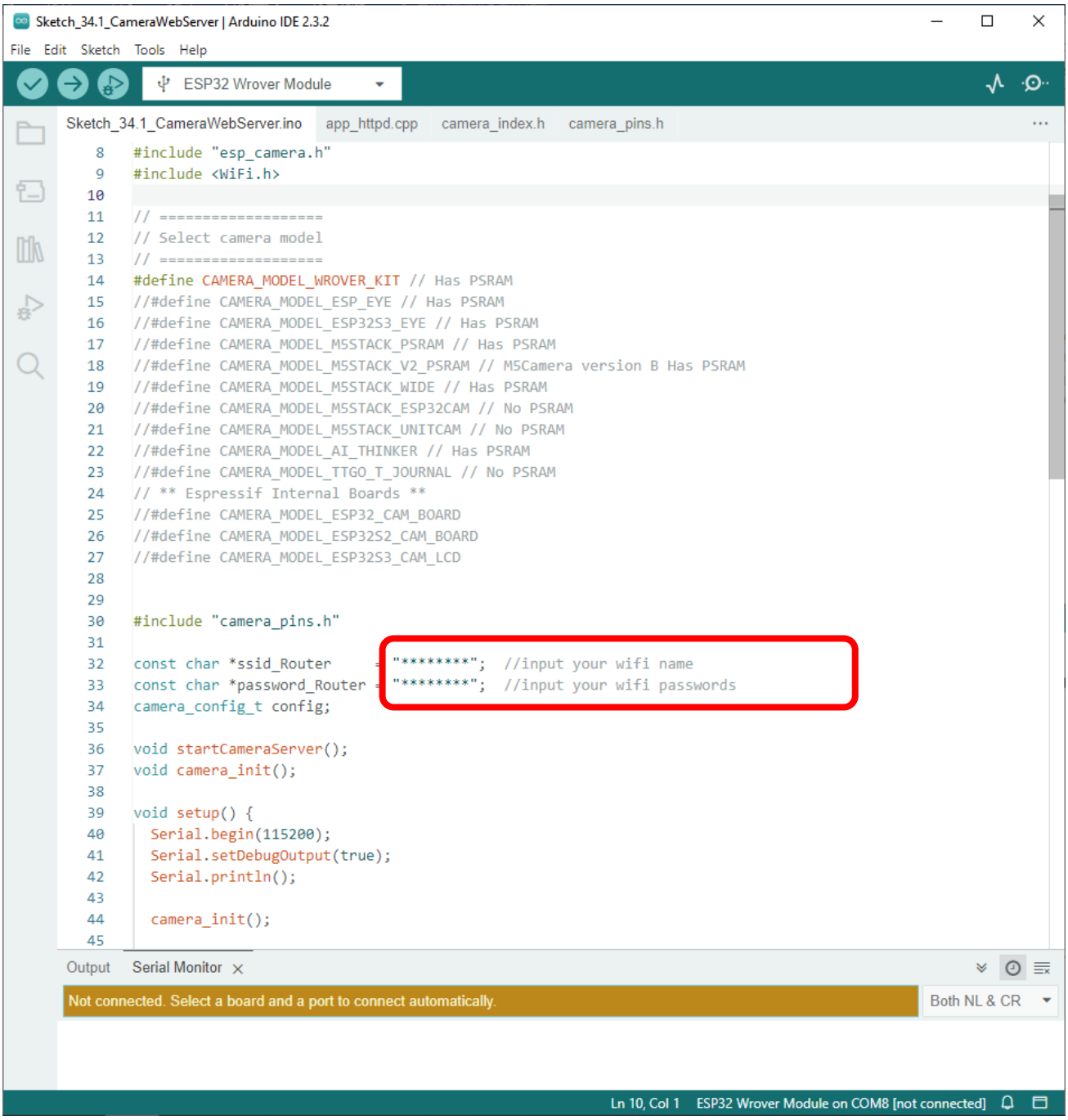
Circuit

Connect Freenove ESP32 to the computer using USB cable.



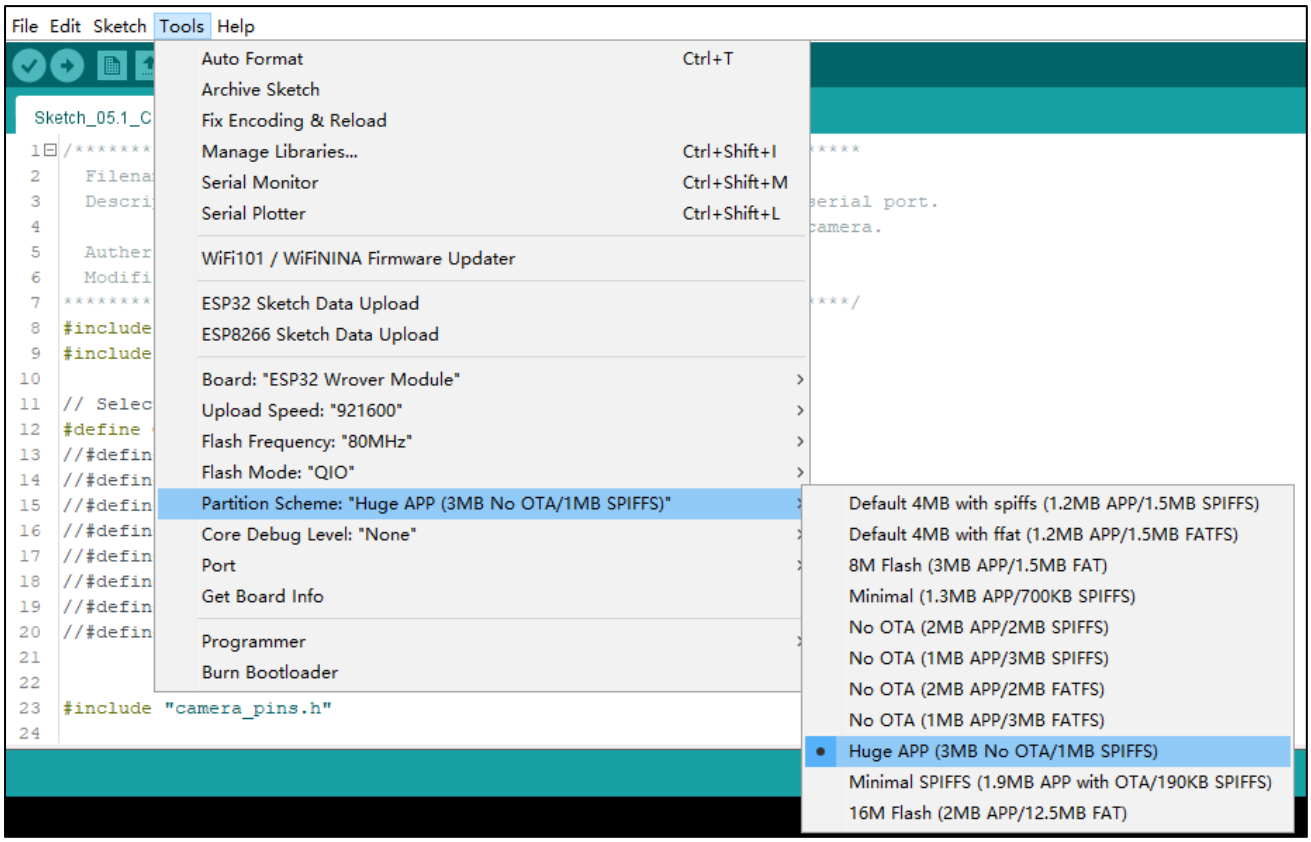
Sketch

Sketch_34.1_As_CameraWebServer

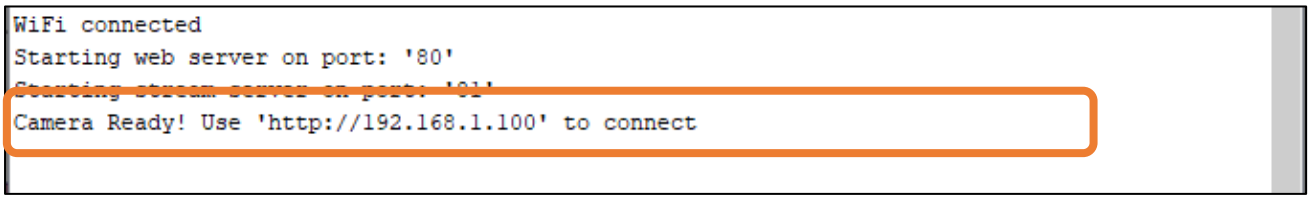


Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

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



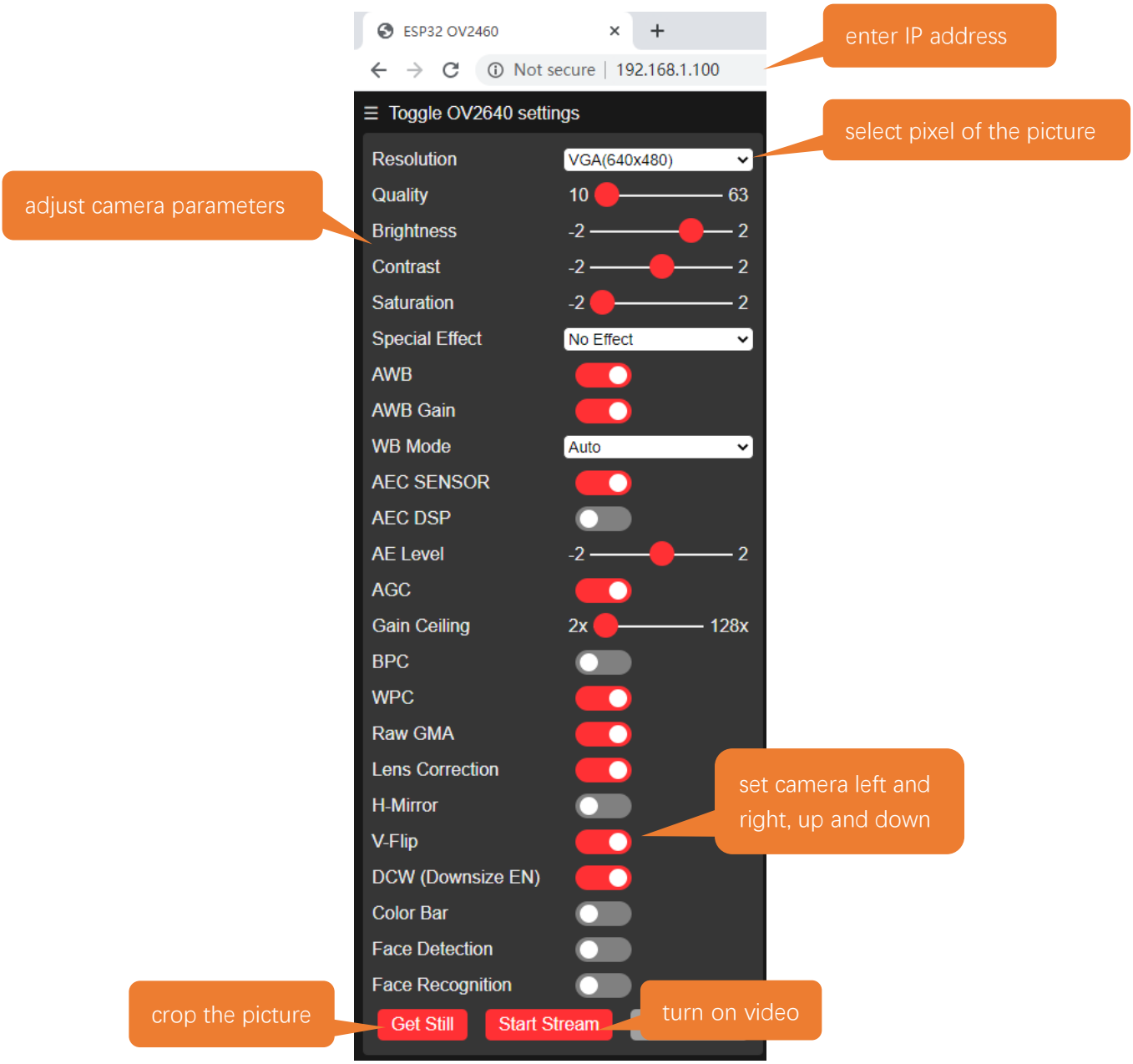
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.



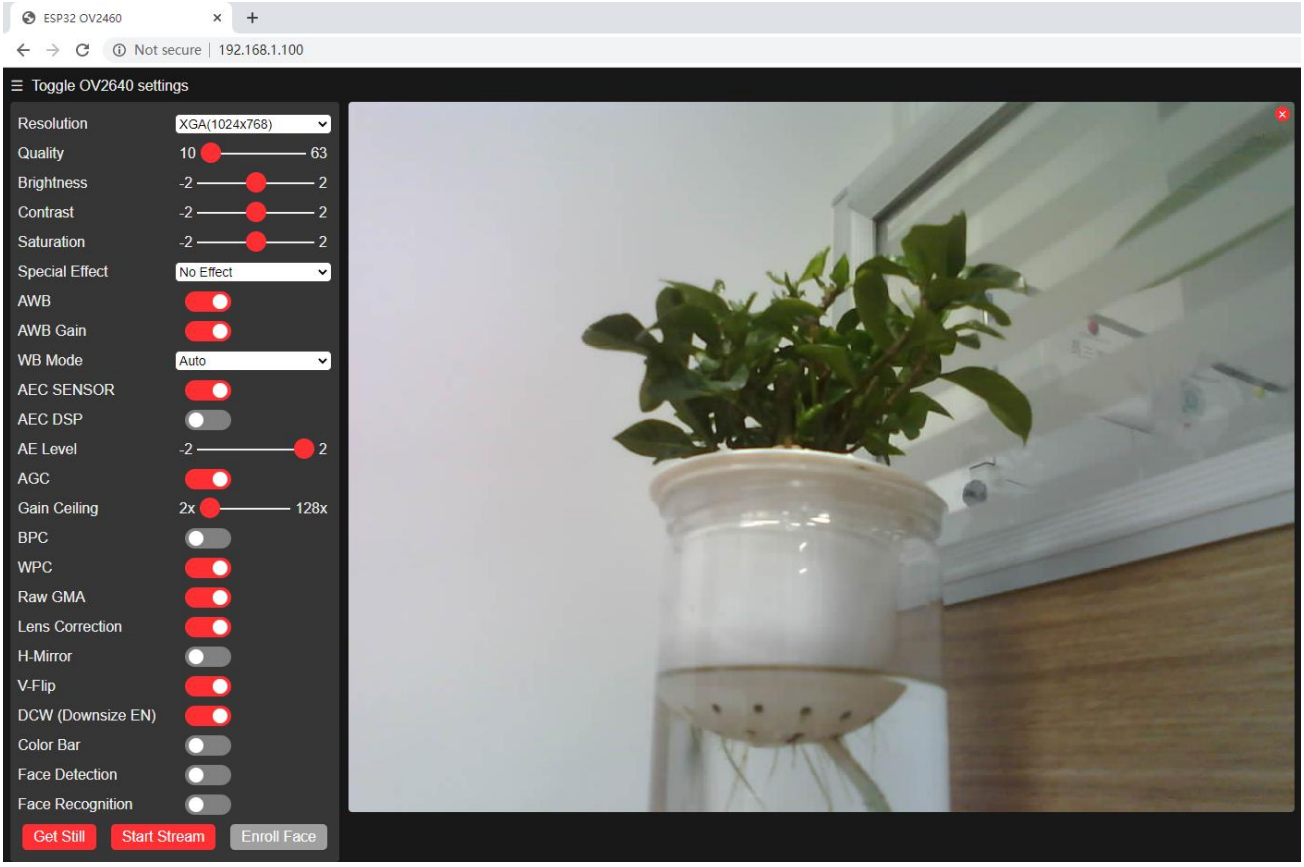
If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

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.

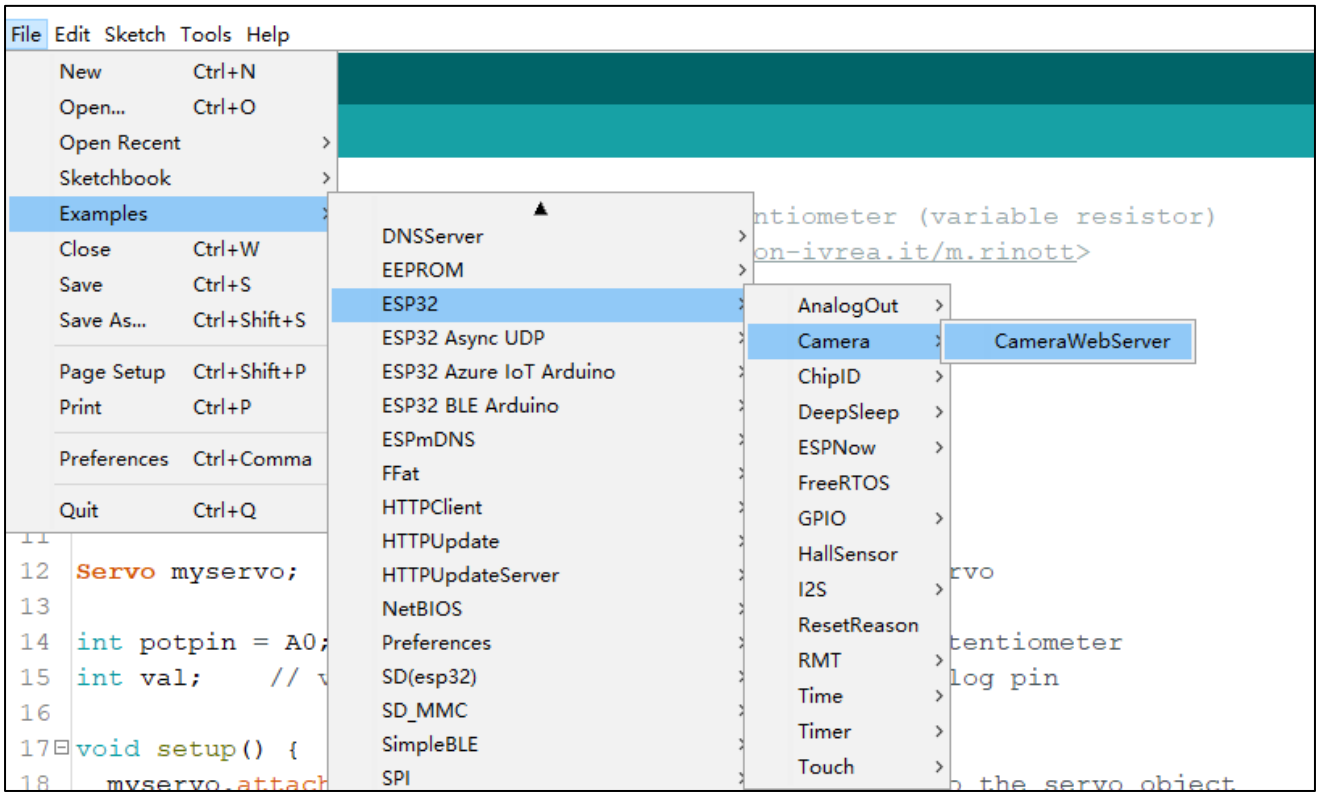
We recommend that the resolution not exceed VGA(640x480).



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.

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

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

```
startCameraServer();

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

```
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
```

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.

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

Reference

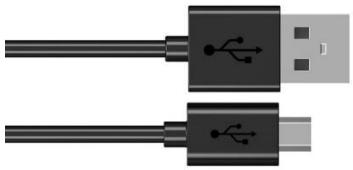
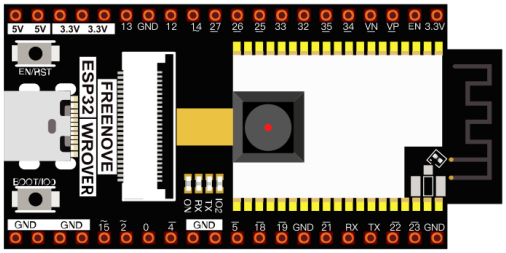
Image resolution	Sharpness	Image resolution	Sharpness
FRAMESIZE_96x96	96x96	FRAMESIZE_HVGA	480x320
FRAMESIZE_QQVGA	160x120	FRAMESIZE_VGA	640x480
FRAMESIZE_QCIF	176x144	FRAMESIZE_SVGA	800x600
FRAMESIZE_HQVGA	240x176	FRAMESIZE_XGA	1024x768
FRAMESIZE_240x240	240x240	FRAMESIZE_HD	1280x720
FRAMESIZE_QVGA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

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

Project 34.2 Video Web Server

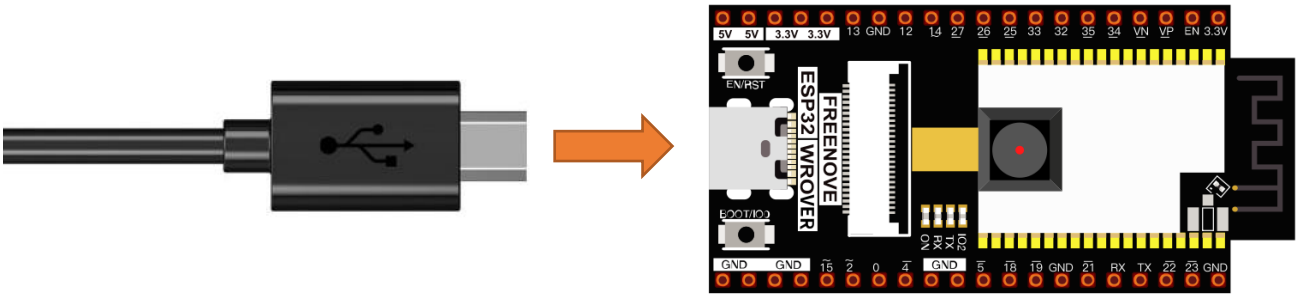
Connect to ESP32 using USB and view its IP address through a serial monitor. Access IP addresses through web pages to obtain real-time video data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Circuit

Connect Freenove ESP32 to the computer using USB cable.



Sketch

Sketch_34.2_As_VideoWebServer

```
File Edit Sketch Tools Help
Sketch_32.2_As_VideoWebServer
11
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;
Done Saving.
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
31                                  ESP32 Wrover Module on COM7
```

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.

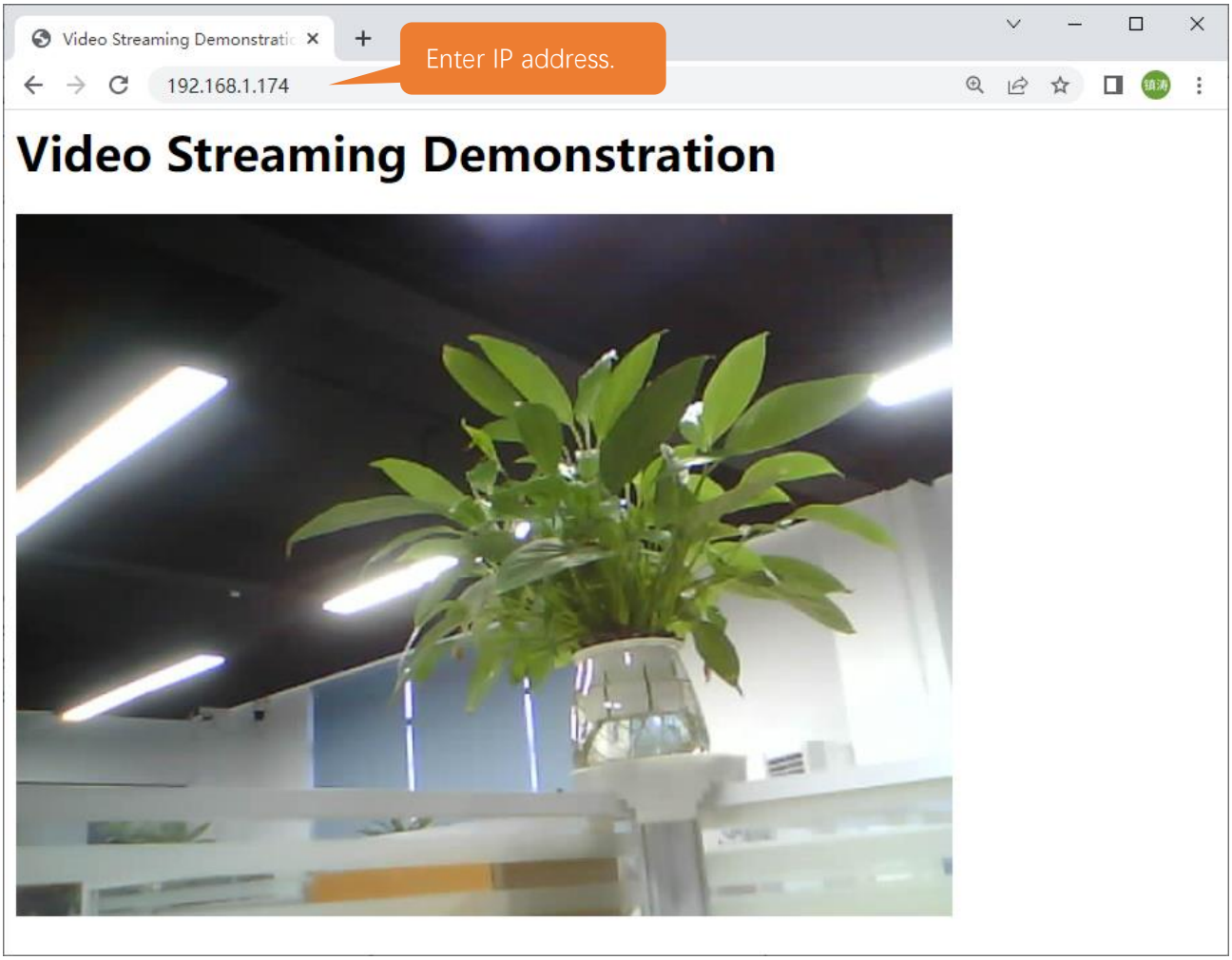
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.

```
COM7
Send
......
WiFi connected
Camera Ready! Use 'http://192.168.1.174' to connect
ets Jun  8 2016 00:22:57
```

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

```

```
#include "camera_pins.h"

const char* ssid     = "********";   //input your wifi name
const char* password = "********";  //input your wifi passwords

void startCameraServer();

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

  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;
  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;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //                      for larger pre-allocated frame buffer.
  if(psramFound()){
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_HVGA;
    config.jpeg_quality = 12;
    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;
```

```
  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;
```

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.

```
  WiFi.begin(ssid, password);

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

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

```
  startCameraServer();

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

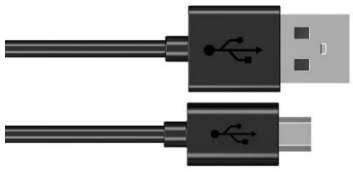
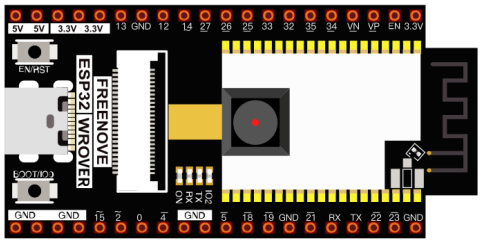
Chapter 35 Camera Tcp Server

In the previous section, we used web page to display the video data captured by ESP32, and in this section, we will use a mobile phone to display it.

Project 35.1 Camera Tcp Server

Connect ESP32 using USB and check its IP address through serial monitor. Use a mobile phone to obtain video and image data.

Component List

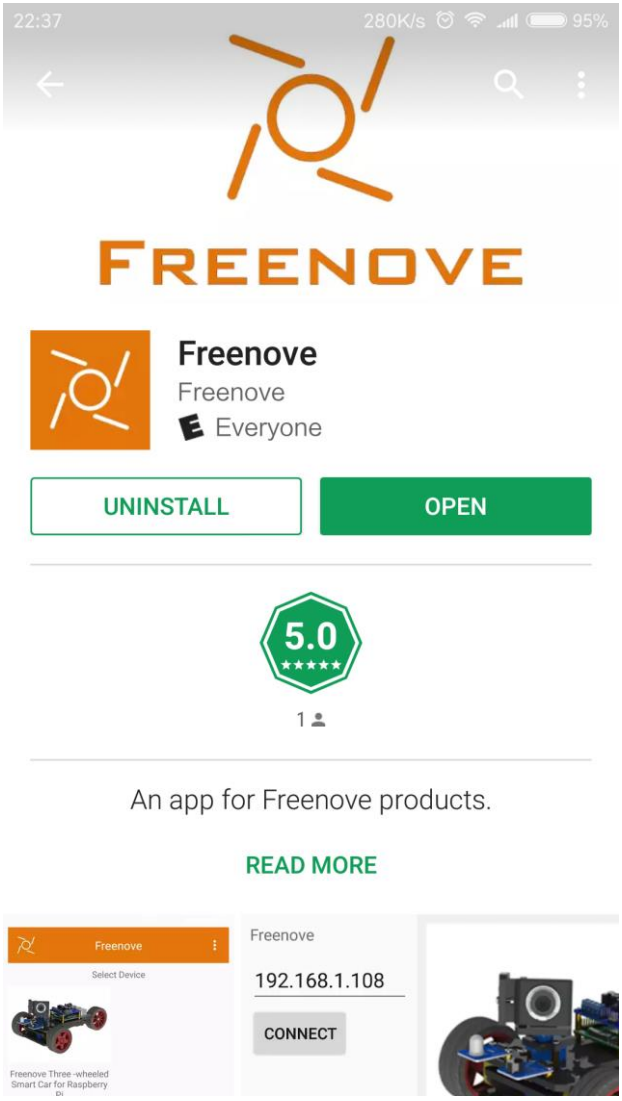
Micro USB Wire x1 	ESP32-WROVER x1 

Install Freenove app

There are three ways to install app, you can choose any one.

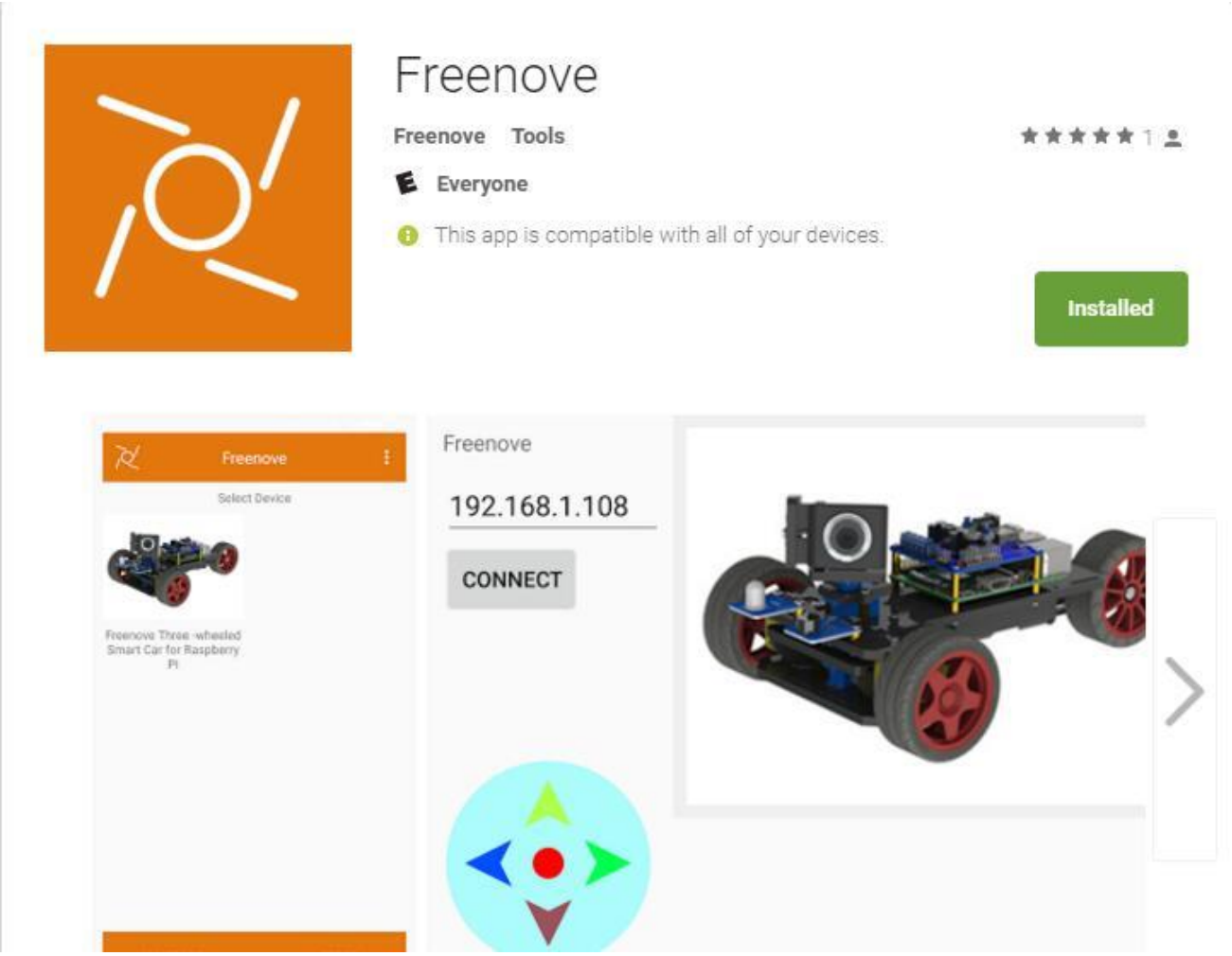
Method 1

Use Google play to search “Freenove”, download and install.


22:37
280K/s
95%
FREENOVE
Freenove
Freenove
Everyone
UNINSTALL
OPEN
5.0
1
An app for Freenove products.
READ MORE
Freenove
Select Device
Freenove Three-wheeled Smart Car for Raspberry
Freenove
192.168.1.108
CONNECT

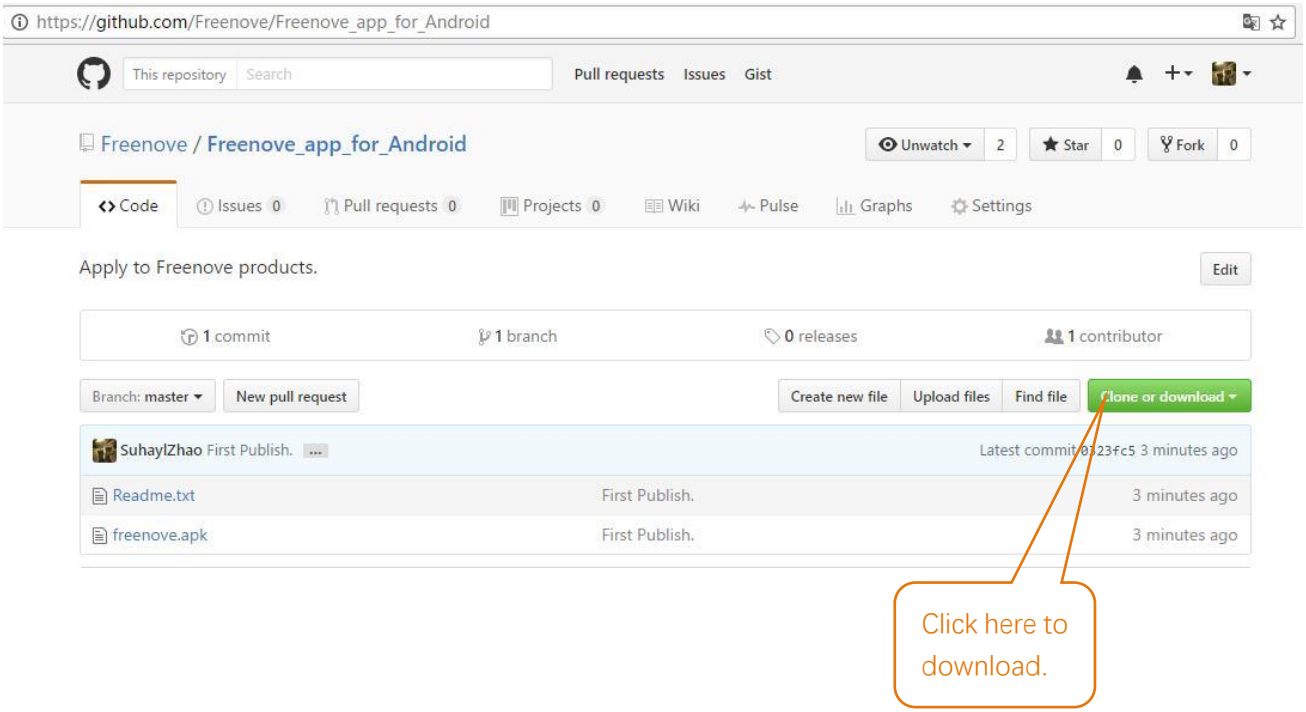
Method 2

Visit https://play.google.com/store/apps/details?id=com.freenove.suhayl.Freenove, and click install.



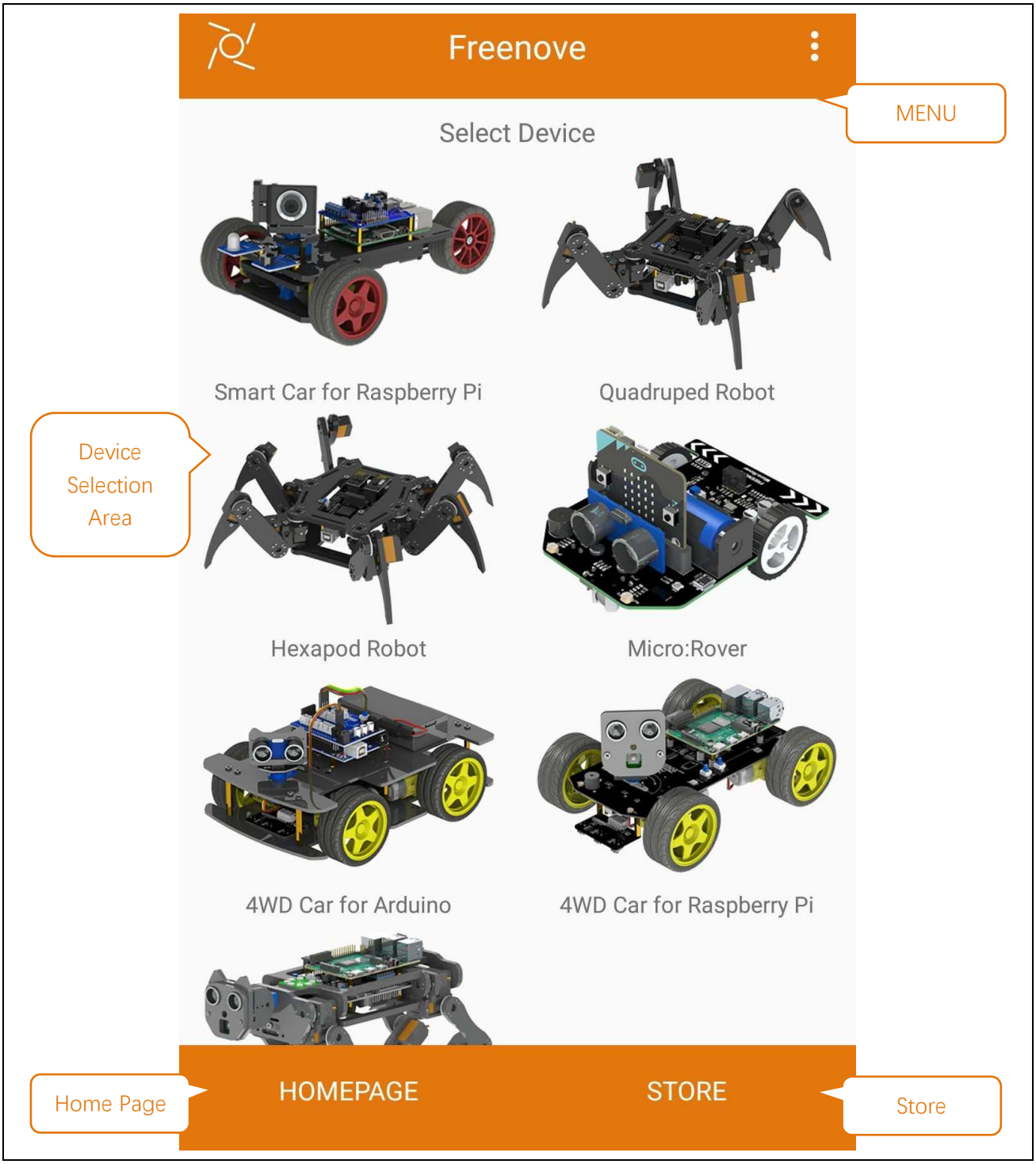
Method 3

Visit https://github.com/Freenove/Freenove_app_for_Android, download the files in this library, and install freenove.apk to your Android phone manually.



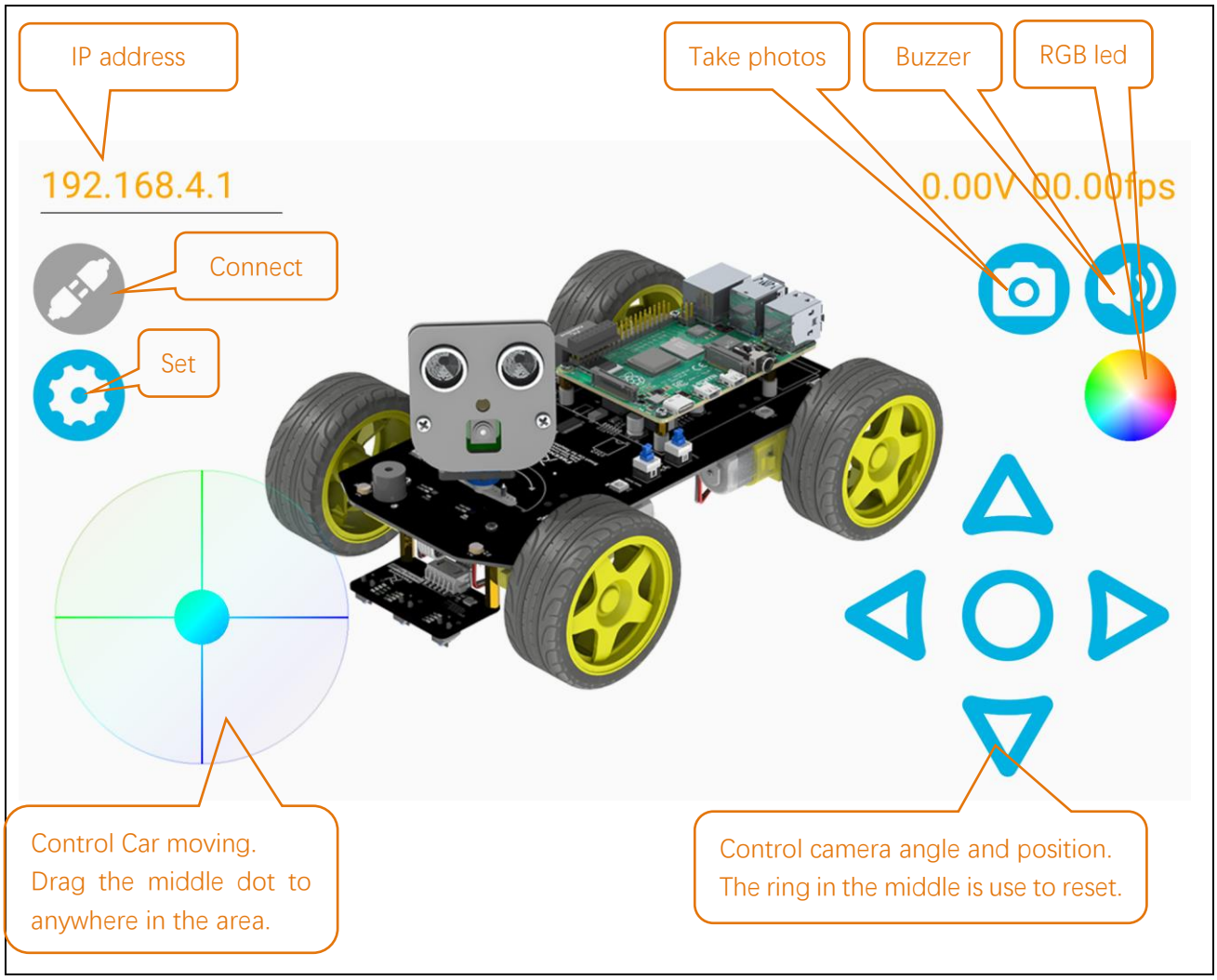
Menu

Open application “Freenove”, as shown below:



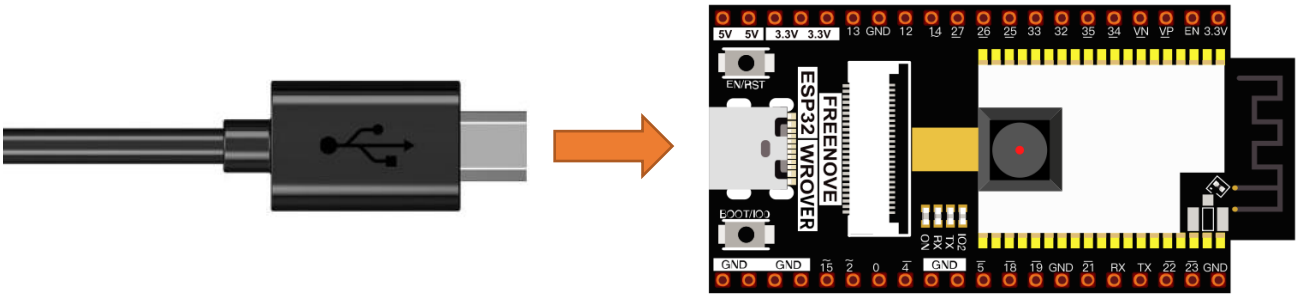
Freenove 4WD Car for Raspberry Pi

In this chapter, we use Freenove 4WD Car for Raspberry Pi, so it is necessary to understand the interface of this mode.



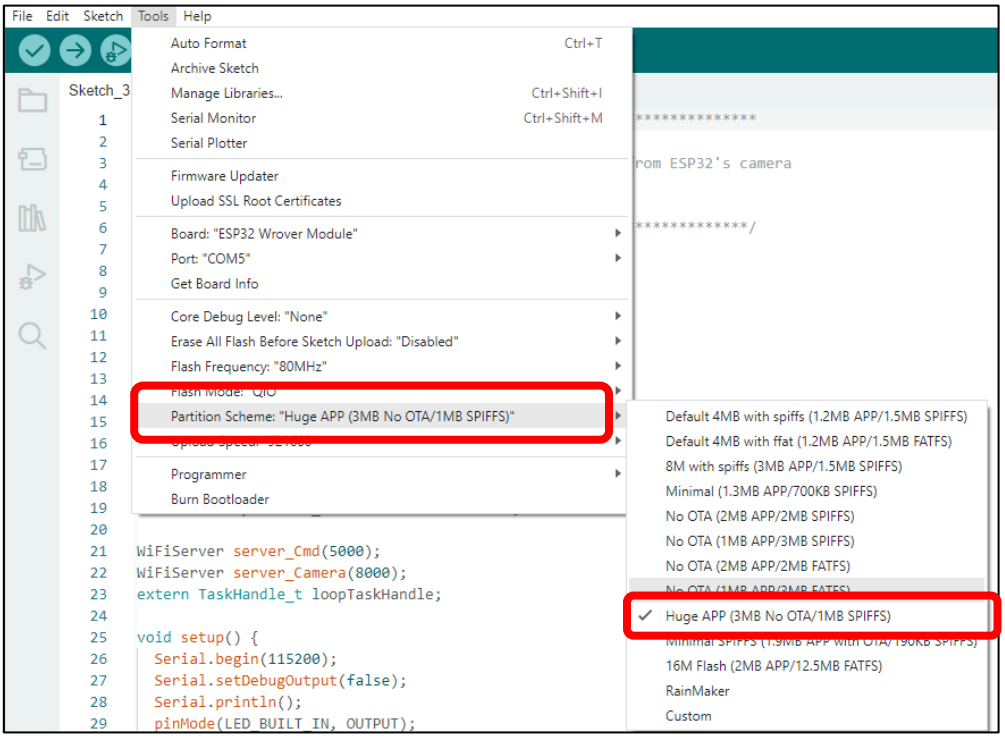
Circuit

Connect Freenove ESP32 to the computer using USB cable.

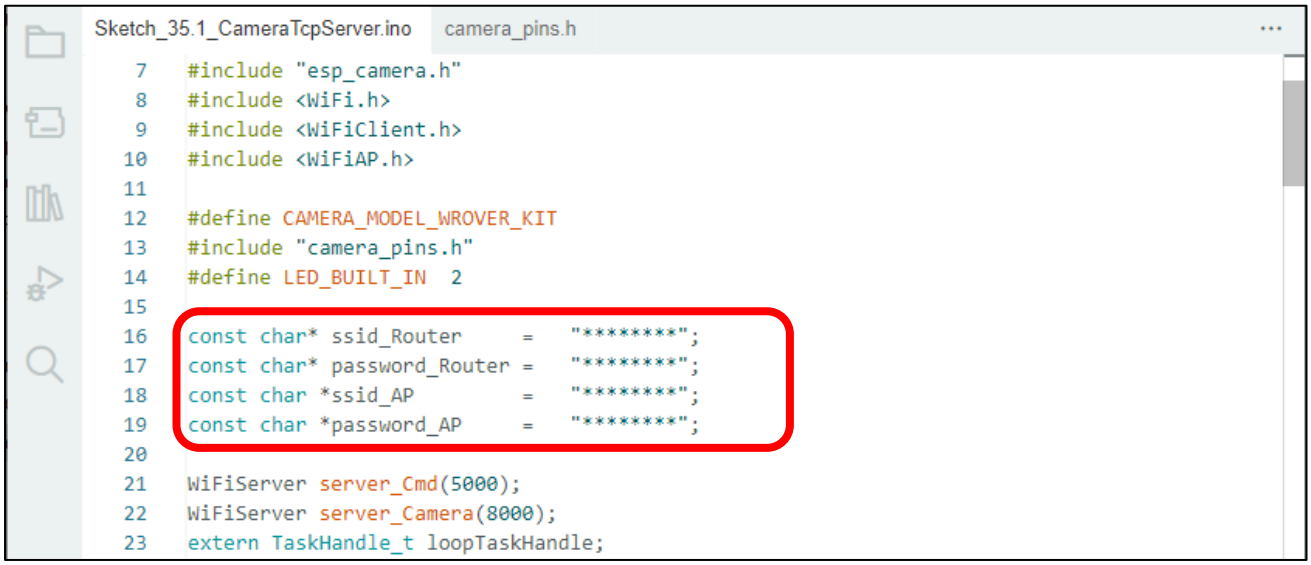


Sketch

Click Tools in the menu bar, select Partition Scheme:" **Huge APP (3MB No OTA/1MB SPIFFS)** ", as shown in the figure below.



After making sure the Tools is configured correctly, don't run Sketch. Due to WiFi, we need to modify Sketch a little bit based on physical situation.



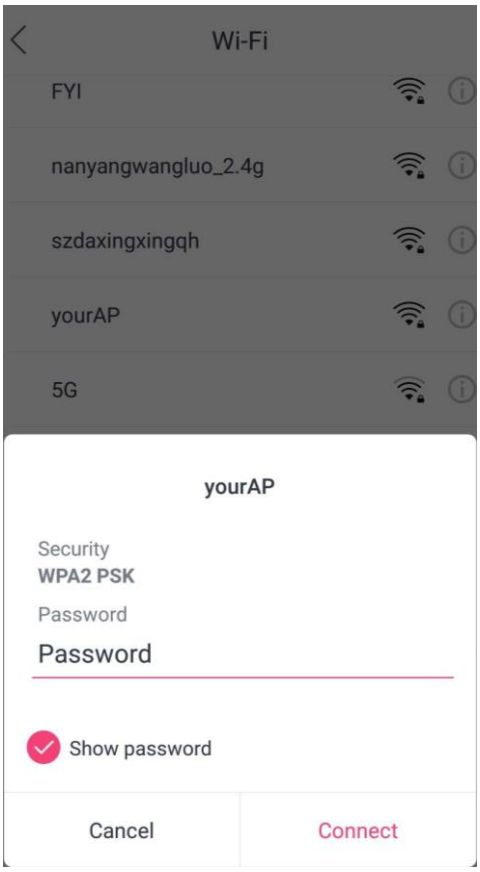
In the box in the figure above, ssid_Router and password_Router are the user's Router name and password, which need to be modified according to the actual name and password. ssid_AP and password_AP are name and password of a AP created by ESP32, and they are freely set by the user. When all settings are correct, compile and upload the code to ESP32, turn on the serial port monitor, and set the baud rate to 115200. The serial monitor will print out two IP addresses.

```
Camera configuration complete!
AP IP address: 192.168.4.1
Connecting FYI_2.4G..
WiFi connected
Camera Ready! Use '192.168.4.1 or 192.168.1.100' to connect in Freenove app.
Task cmd_server is starting ...
Task Blink is starting ...
```

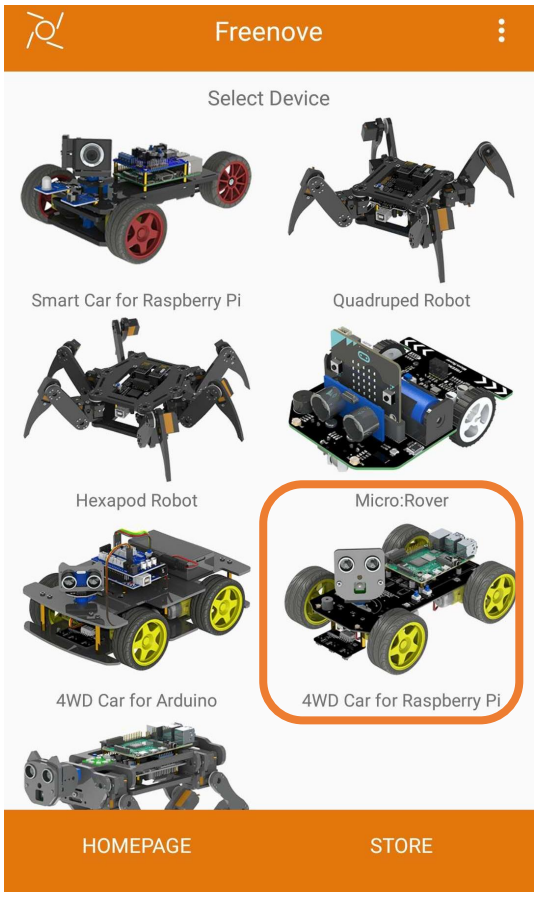
There are two methods for you to check camera data of ESP32 via mobile phone APP.

Method 1:

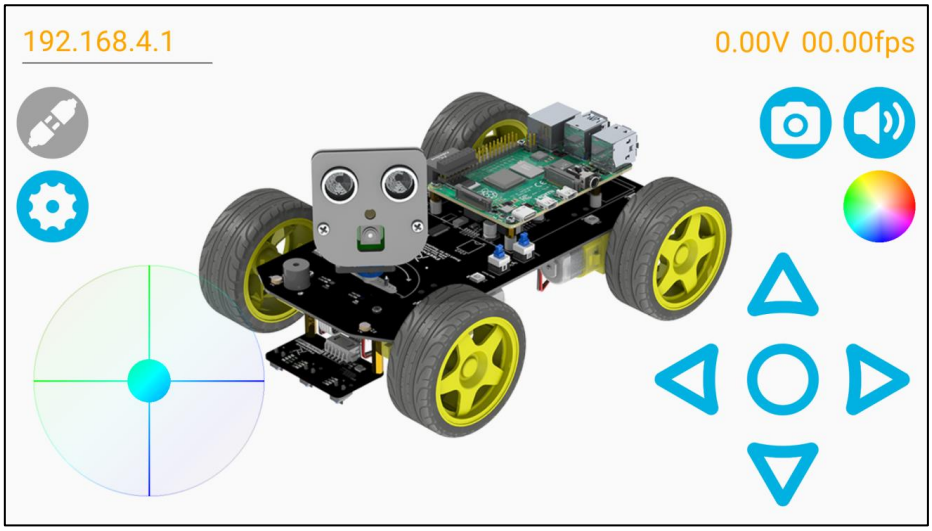
Using your phone's WiFi function, select the WiFi name represented by ssid_AP in Sketch and enter the password "password_AP" to connect.



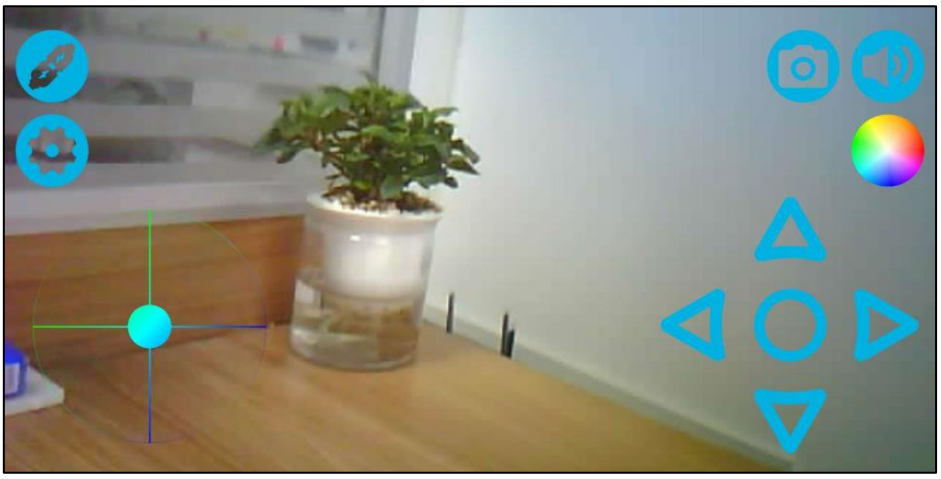
Next, open Freenove app and select 4WD Car for Raspberry Pi mode.



Enter the IP address printed by serial port in the new interface, which generally is “192.168.4.1”



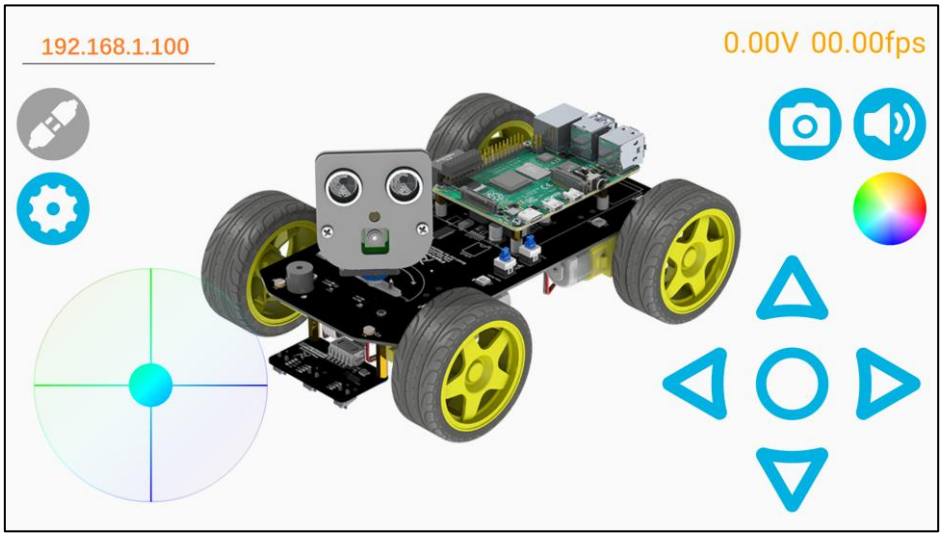
Click “Connect”.



Method 2:

Using your phone's WiFi function, select the router named ssid_Router and enter the password “ssid_password” to connect. And then open Freenove app and select 4WD Car for Raspberry Pi mode. The operation is similar to Method 1.

Enter the IP address printed by serial port in the new interface, which generally is not “192.168.4.1” but another one. The IP address in this example is “192.168.1.100”. After entering the IP address, click “Connect”.



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

Sketch_35.1_Camera_Tcp_Server

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

#define CAMERA_MODEL_WROVER_KIT
```

```
#include "camera_pins.h"
#define LED_BUILT_IN  2

const char *ssid_Router     =   "********";
const char *password_Router =   "********";
const char *ssid_AP         =   "********";
const char *password_AP     =   "********";

WiFiServer server_Cmd(5000);
WiFiServer server_Camera(8000);
extern TaskHandle_t loopTaskHandle;

void setup() {
    Serial.begin(115200);
    Serial.setDebugOutput(false);
    Serial.println();
    pinMode(LED_BUILT_IN, OUTPUT);
    cameraSetup();

    WiFi.softAP(ssid_AP, password_AP);
    IPAddress myIP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(myIP);
    server_Camera.begin(8000);
    server_Cmd.begin(5000);
    ////////////////////////////////////////////////////
    WiFi.begin(ssid_Router, password_Router);
    Serial.print("Connecting ");
    Serial.print(ssid_Router);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        WiFi.begin(ssid_Router, password_Router);
    }
    while (WiFi.STA.hasIP() != true) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("WiFi connected");
    ////////////////////////////////////////////////////
    Serial.print("Camera Ready! Use '");
    Serial.print(WiFi.softAPIP());
    Serial.print(" or ");
```

```
    Serial.print(WiFi.localIP());
    Serial.println("' to connect in Freenove app.");

    disableCore0WDT();
    xTaskCreateUniversal(loopTask_Cmd, "loopTask_Cmd", 8192, NULL, 1, &loopTaskHandle, 0);
    //loopTask_Cmd uses core 0.
    xTaskCreateUniversal(loopTask_Blink, "loopTask_Blink", 8192, NULL, 1, &loopTaskHandle, 0);//loopTask_Blink uses core 0.
}
//task loop uses core 1.
void loop() {
    WiFiClient client = server_Camera.accept();                // listen for incoming clients
    if (client) {                                              // if you get a client,
        Serial.println("Camera Server connected to a client.");// print a message out the serial port
        String currentLine = "";       // make a String to hold incoming data from the client
        while (client.connected()) {   // loop while the client's connected
            camera_fb_t * fb = NULL;
            while (client.connected()) {
                fb = esp_camera_fb_get();
                if (fb != NULL) {
                    uint8_t slen[4];
                    slen[0] = fb->len >> 0;
                    slen[1] = fb->len >> 8;
                    slen[2] = fb->len >> 16;
                    slen[3] = fb->len >> 24;
                    client.write(slen, 4);
                    client.write(fb->buf, fb->len);
                }
                else {
                    Serial.println("Camera Error");
                }
            }
        }
        // close the connection:
        client.stop();
        Serial.println("Camera Client Disconnected.");
    }
}

void loopTask_Cmd(void *pvParameters) {
    Serial.println("Task Cmd_Server is starting ... ");
    while (1) {
        WiFiClient client = server_Cmd.accept(); // listen for incoming clients
```

```
        if (client) {                                // if you get a client,
            Serial.println("Command Server connected to a client.");// print a message out the serial port
            String currentLine = ""; // make a String to hold incoming data from the client
            while (client.connected()) {        // loop while the client's connected
                if (client.available()) {       // if there's bytes to read from the client,
                    char c = client.read();     // read a byte, then
                    client.write(c);
                    Serial.write(c);            // print it out the serial monitor
                    if (c == '\n') {            // if the byte is a newline character
                        currentLine = "";
                    }
                    else {
                        currentLine += c;       // add it to the end of the currentLine
                    }
                }
            }
            // close the connection:
            client.stop();
            Serial.println("Command Client Disconnected.");
        }
    }
}
void loopTask_Blink(void *pvParameters) {
    Serial.println("Task Blink is starting ... ");
    while (1) {
        digitalWrite(LED_BUILT_IN, !digitalRead(LED_BUILT_IN));
        delay(1000);
    }
}

void cameraSetup() {
    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;
    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;

    psramFound();
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 10;
    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;
    }
    Serial.println("Camera configuration complete!");
}
```

Include header files that drive camera and WiFi.

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

#define CAMERA_MODEL_WROVER_KIT
#include "camera_pins.h"
#define LED_BUILT_IN  2
```

Set name and password for router that ESP32 needs to connect to. And set ESP32 to open two servers, whose port are 82 and 5000 respectively.

```
const char *ssid_Router     =   "********";
const char *password_Router =   "********";
const char *ssid_AP         =   "********";
const char *password_AP     =   "********";
```

Enable ESP32's server function and set two monitor ports as 5000 and 8000. In general, the two port numbers do not require modifications.

```
WiFiServer server_Cmd(5000);
WiFiServer server_Camera(8000);
extern TaskHandle_t loopTaskHandle;
```

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();
```

Define a variable for camera interface and initialize it.

```
void cameraSetup() {
    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;
    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;

    psramFound();
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 10;
    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;
    }
    Serial.println("Camera configuration complete!");
}
```

Loop function will constantly send camera data obtained to mobile phone APP.

```
        while (client.connected()) {     // loop while the client's connected
            camera_fb_t * fb = NULL;
            while (client.connected()) {
                fb = esp_camera_fb_get();
                if (fb != NULL) {
                    uint8_t slen[4];
                    slen[0] = fb->len >> 0;
                    slen[1] = fb->len >> 8;
                    slen[2] = fb->len >> 16;
                    slen[3] = fb->len >> 24;
                    client.write(slen, 4);
                    client.write(fb->buf, fb->len);
                }
                else {
                    Serial.println("Camera Error");
                }
            }
        }
```

The loopTask_Cmd() function sends the received instruction back to the phone app and prints it out through a serial port.

```
void loopTask_Cmd(void *pvParameters) {
    Serial.println("Task Cmd_Server is starting ... ");
    while (1) {
        WiFiClient client = server_Cmd.accept(); // listen for incoming clients
        if (client) {                             // if you get a client,
            Serial.println("Command Server connected to a client.");// print a message out the serial port
            String currentLine = "";   // make a String to hold incoming data from the client
            while (client.connected(){      // loop while the client's connected
                if (client.available()) {    // if there's bytes to read from the client,
                    char c = client.read();  // read a byte, then
                    client.write(c);
                    Serial.write(c);         // print it out the serial monitor
                    if (c == '\n') {         // if the byte is a newline character
                        currentLine = "";
                    }
                    else {
                        currentLine += c;   // add it to the end of the currentLine
                    }
                }
            }
            // close the connection:
            client.stop();
```

```
            Serial.println("Command Client Disconnected.");
        }
    }
}
```

loopTask_ Blink()function will control the blinking of LED. When you see LED blinking, it indicates that ESP32 has been configured and starts working.

```
void loopTask_Blink(void *pvParameters) {
    Serial.println("Task Blink is starting ... ");
    while (1) {
        digitalWrite(LED_BUILT_IN, !digitalRead(LED_BUILT_IN));
        delay(1000);
    }
}
```

If you do not have a router near you, or if you are outdoors, you can annotate the following code, and then compile and upload it to ESP32. And you can display the video images on your phone by Method 1.

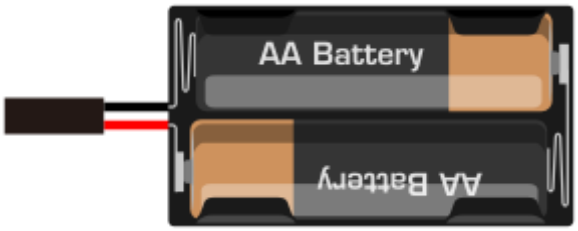
```
    ////////////////////////////////////////////////////
    WiFi.begin(ssid_Router, password_Router);
    Serial.print("Connecting ");
    Serial.print(ssid_Router);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        WiFi.begin(ssid_Router, password_Router);
    }
    while (WiFi.STA.hasIP() != true) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("WiFi connected");
    ////////////////////////////////////////////////////
```

Chapter 36 Soldering Circuit Board

Project 36.1 Soldering a Buzzer

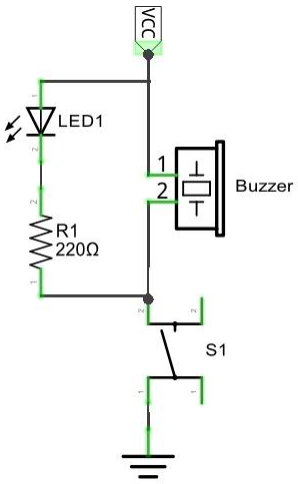
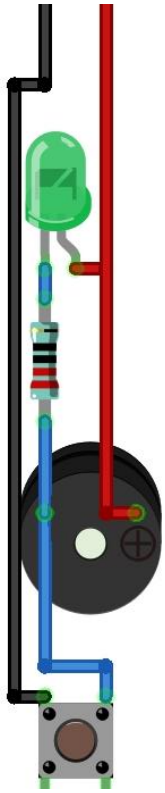
We have tried to use a buzzer in a previous chapter, and now we will solder a circuit that when the button is pressed, the buzzer sounds.
This circuit does not need programming and can work when it is powered on. And when the button is not pressed, there is no power consumption.
You can install it on your bike, bedroom door or any other places where it is needed.

Component List

Pin header x2	LED x1	Resistor 220Ω x1	Active buzzer x1	Push button x1
AA Battery Holder x1 				

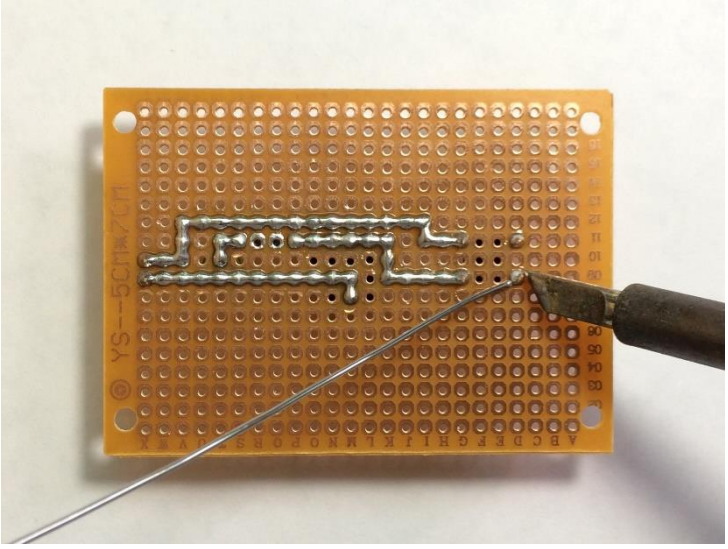
Circuit

We will solder the following circuit on the main board.

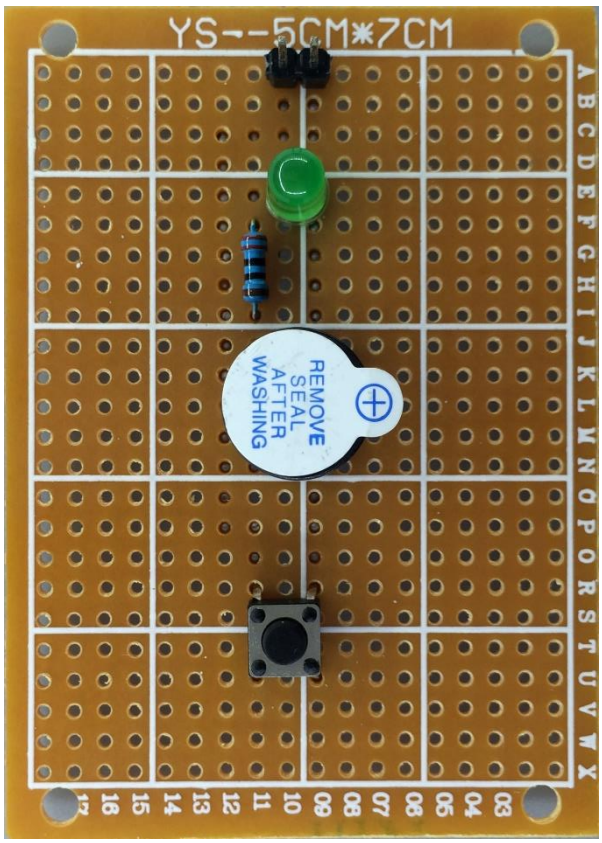
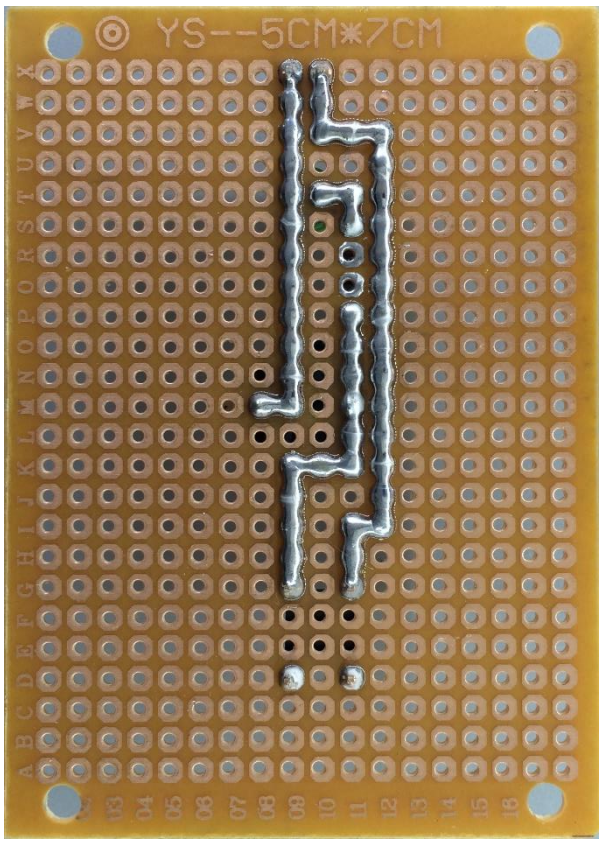
Schematic diagram	Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com
	

Soldering the Circuit

Insert the components on the main board and solder the circuit on its back.

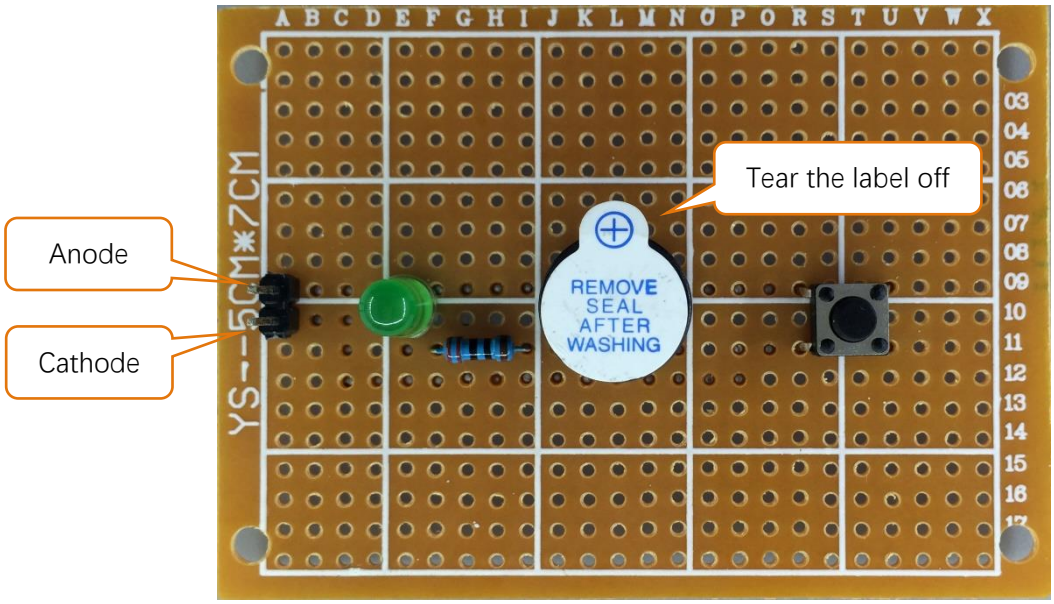


rendering after soldering:

Front	Back
	

Testing circuit

Connect the circuit board to power supply (3~5V). You can use ESP32 board or battery box as the power supply.

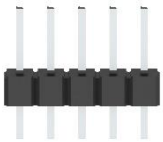
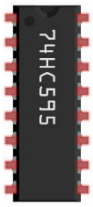


Press the push button after connecting the power, and then the buzzer will make a sound.

Project 34.2 Soldering a Flowing Water Light

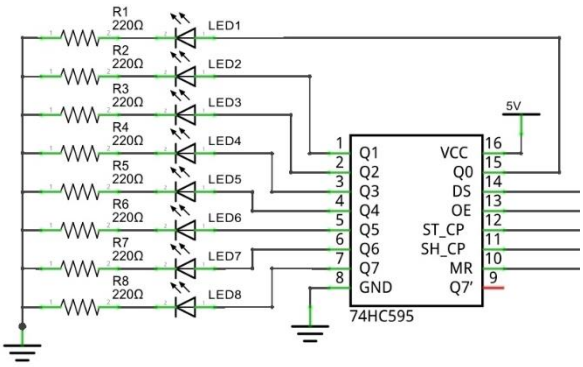
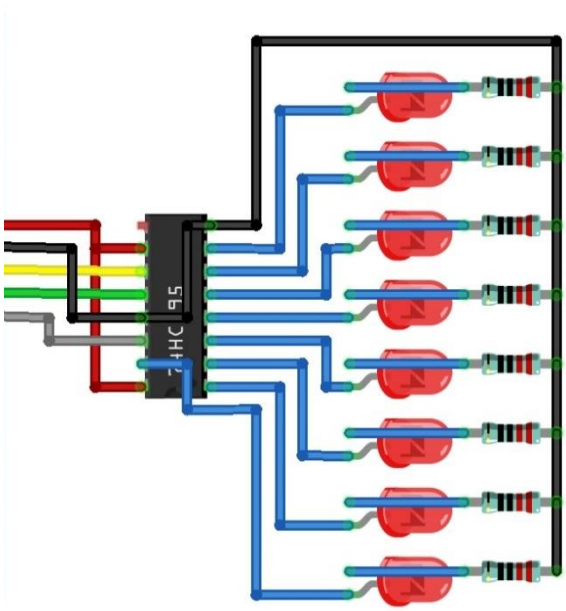
From previous chapter, we have learned to make a flowing water light with LED. Now, we will solder a circuit board, and use the improved code to make a more interesting flowing water light.

Component List

Pin header x5	Resistor 220Ω x8	LED x1	74HC595 x1
			

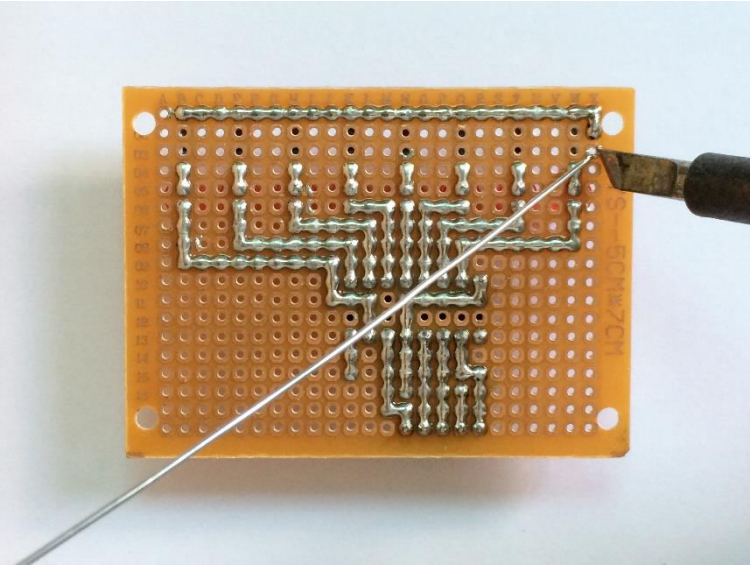
Circuit

Solder the following circuit on the main board.

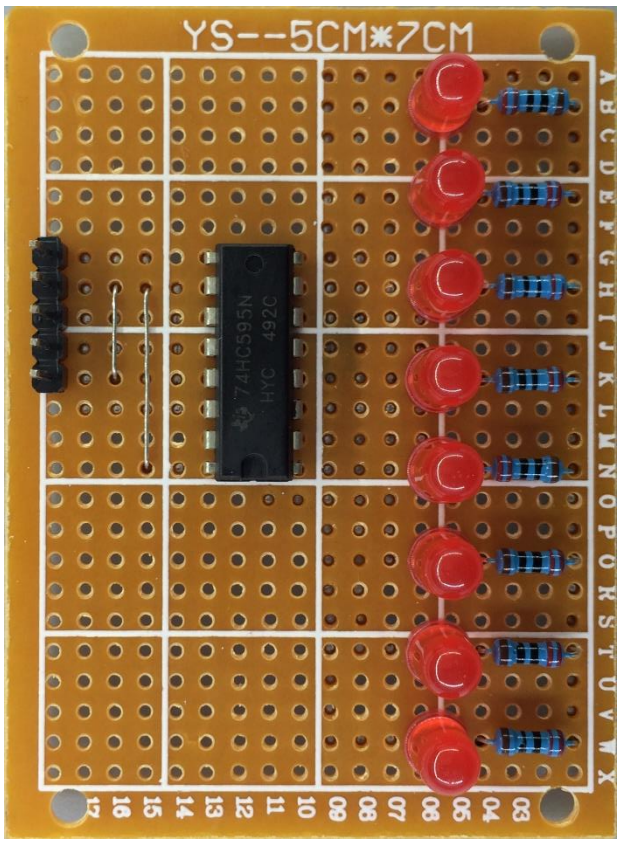
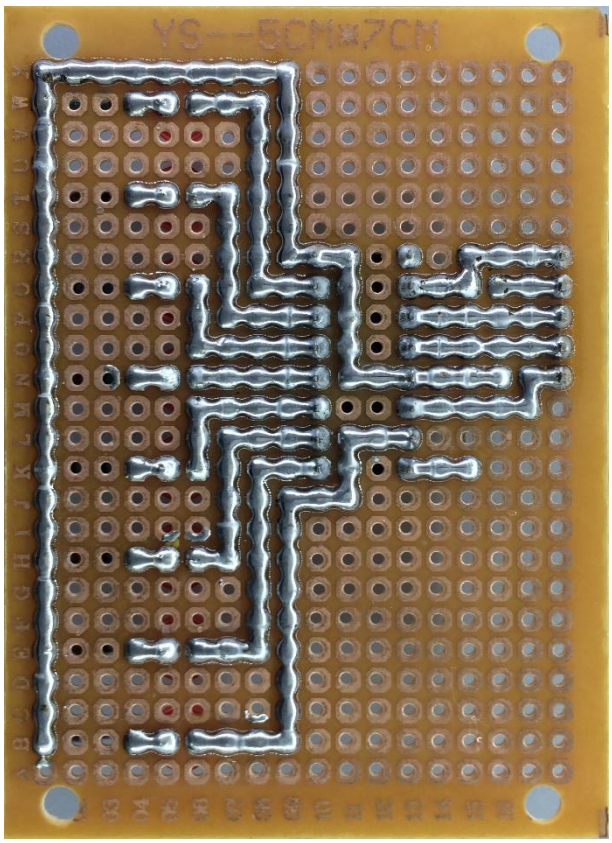
Schematic diagram	Hardware connection
	

Soldering the Circuit

Insert the components on the main board and solder the circuit on its back.

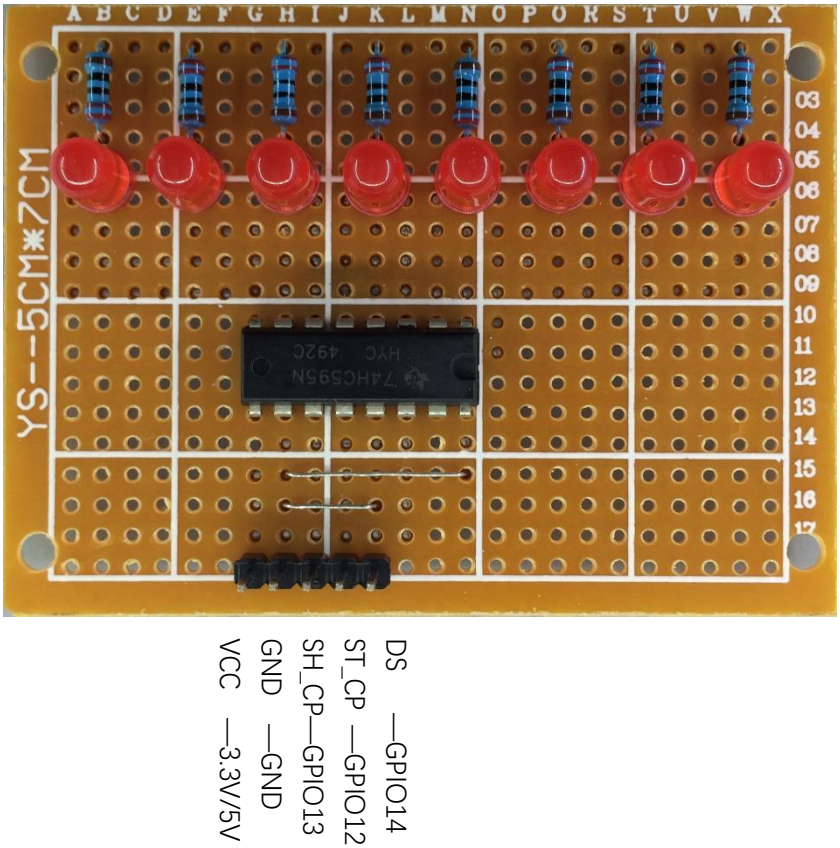


Rendering after soldering:

Front	Back
	

Connecting the Circuit

Connect the board to ESP32 with jumper wire in the following way.



Sketch

The following is the program code:

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595(Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595(Pin13)
int dataPin = 14;           // Pin connected to DS of 74HC595(Pin14)

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Define a one-byte variable to use the 8 bits to represent the state of 8 LEDs of LED bar
```

```
graph.
  // This variable is assigned to 0x01, that is, binary 00000001, which indicates only one LED light
on.
  byte x = 0x01;    // 0b 0000 0001
  for (int j = 0; j < 8; j++) { // Let LED light up from right to left
    writeTo595(LSBFIRST, x);
    x <<= 1; // make the variable move one bit to left once, then the bright LED move one step
to the left once.
    delay(50);
  }
  delay(100);
  x = 0x80;         //0b 1000 0000
  for (int j = 0; j < 8; j++) { // Let LED light up from left to right
    writeTo595(LSBFIRST, x);
    x >>= 1;
    delay(50);
  }
  delay(100);
}
void writeTo595(int order, byte _data ) {
  // Output low level to latchPin
  digitalWrite(latchPin, LOW);
  // Send serial data to 74HC595
  shiftOut(dataPin, clockPin, order, _data);
  // Output high level to latchPin, and 74HC595 will update the data to the parallel output
port.
  digitalWrite(latchPin, HIGH);
}
```

In fact, this code is copied from chapter 15. If you have any questions for the code, please click "Chapter 15 74HC595 & LED Bar Graph" to return to Chapter 15 to study again.

What's next?

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 are interesting in processing, you can learn the Processing.pdf in:
Freenove_Ultimate_Starter_Kit_for_ESP32\C\Processing.pdf

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.